Speculation in JavaScriptCore

Filip Pizlo
Apple Inc.

Speculation

- Is ideal for…
  - JavaScript
  - Java
  - Smalltalk
  - Python
  - Ruby
  - Scheme
  - *…many dynamic languages…*

Agenda

- Speculation Overview
- JavaScriptCore Overview
- Speculation
  - Bytecode (Common IR)
  - Control
  - Profiling
  - Compilation
  - OSR (On Stack Replacement)

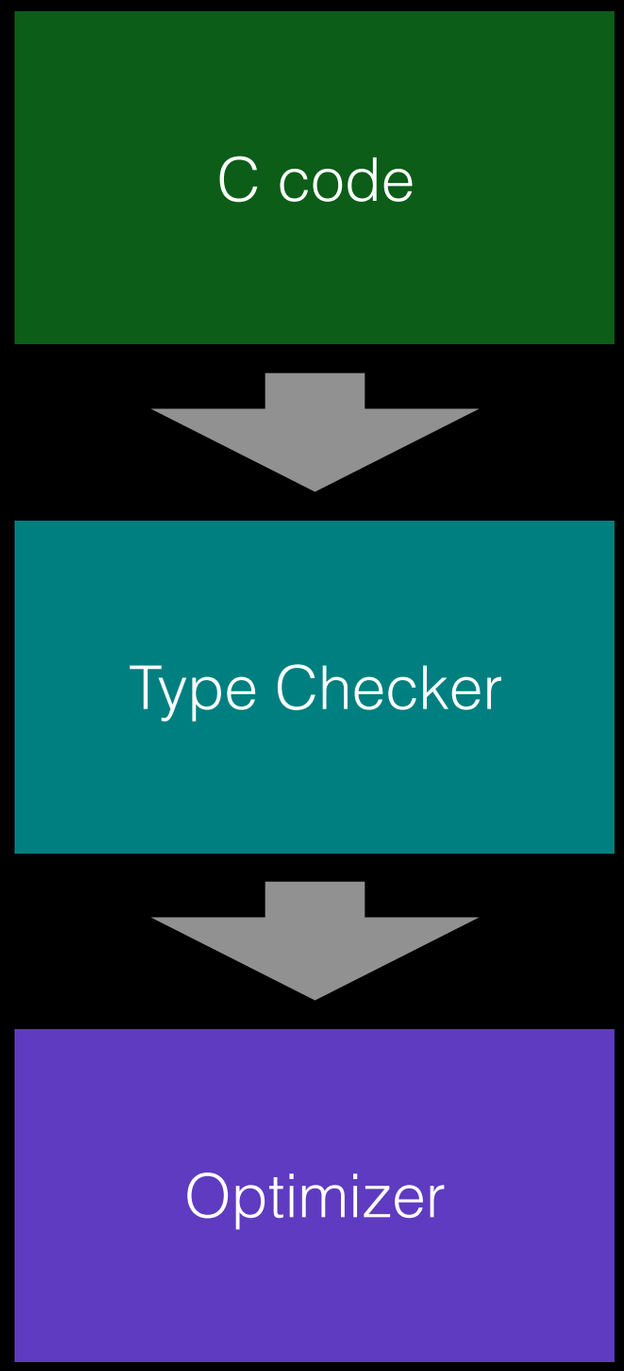
Agenda

- Speculation Overview
- JavaScriptCore Overview
- Speculation
  - Bytecode (Common IR)
  - Control
  - Profiling
  - Compilation
  - OSR (On Stack Replacement)

Intuition

Leverage traditional compiler technology to make dynamic languages as fast as possible.

Traditional Compiler



C function

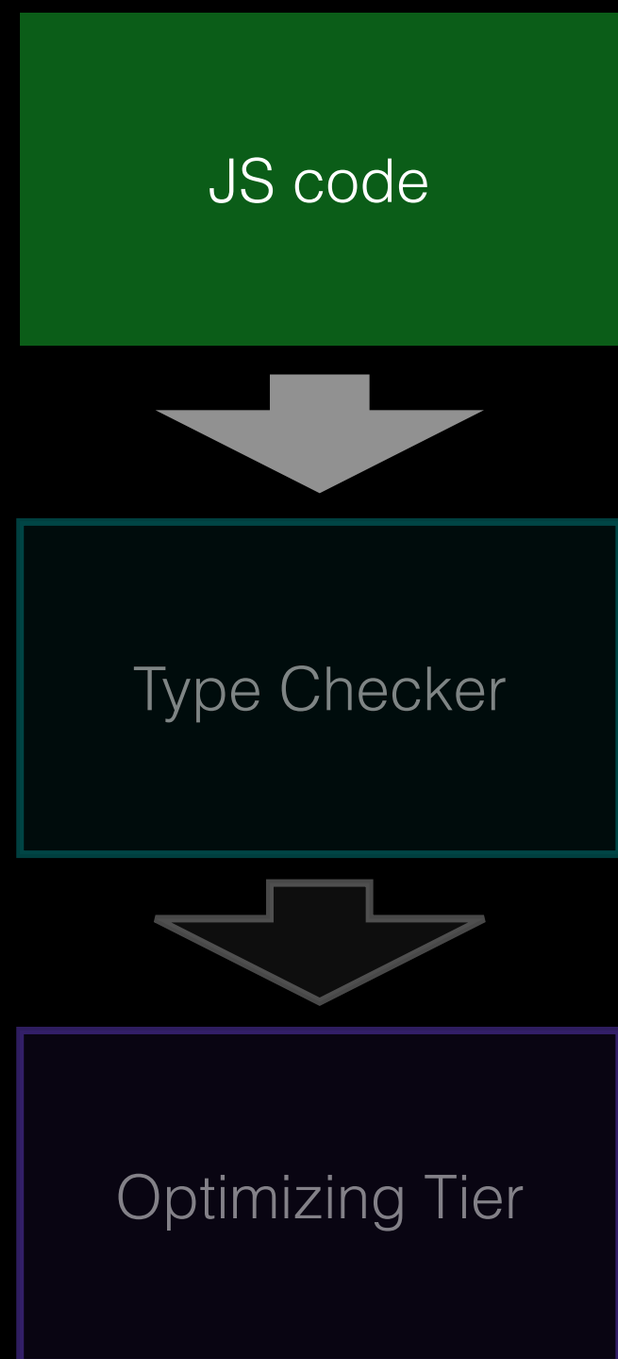
```
int foo(int a, int b)
{
    return a + b;
}
```

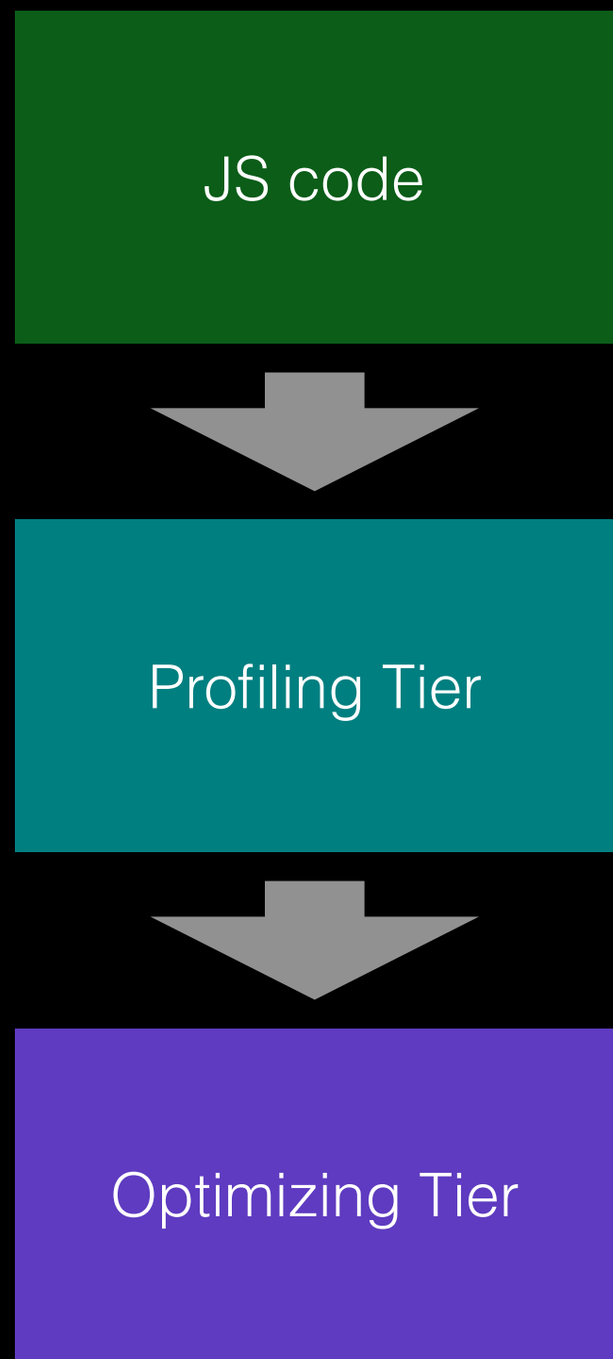
C function

```
int foo(int a, int b)
{
    return a + b;
}
```

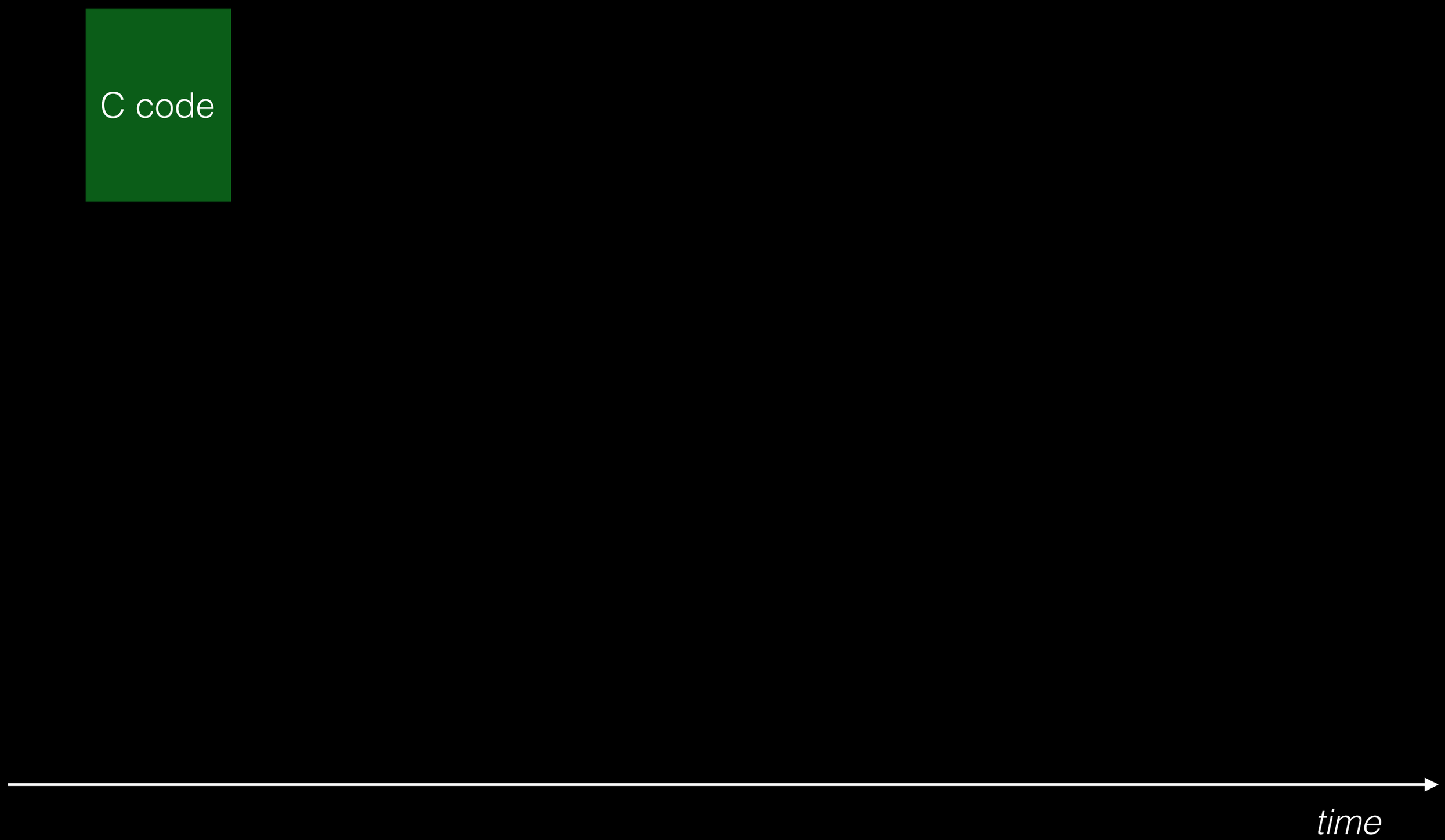
JS function

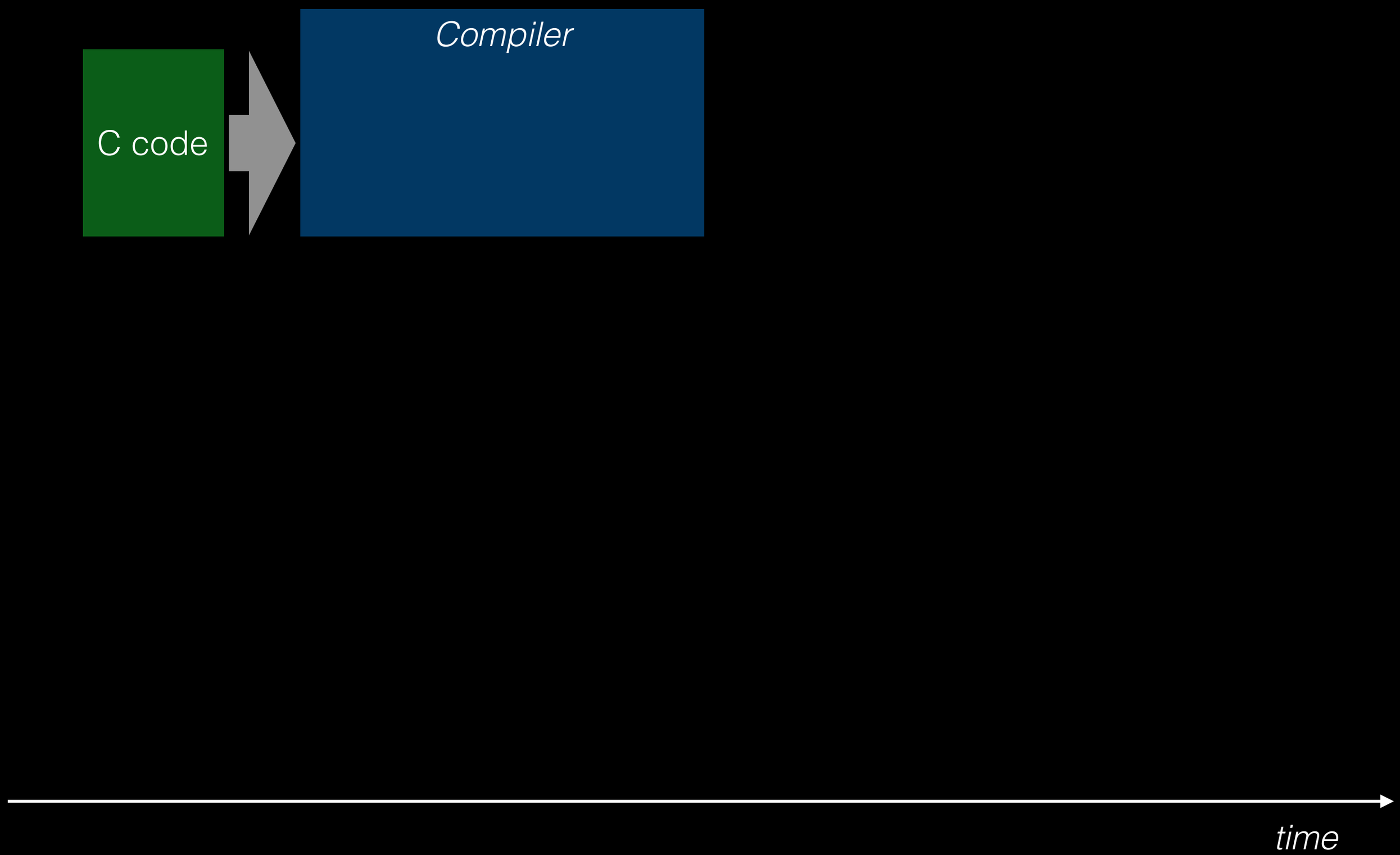
```
function foo(a, b)
{
    return a + b;
}
```

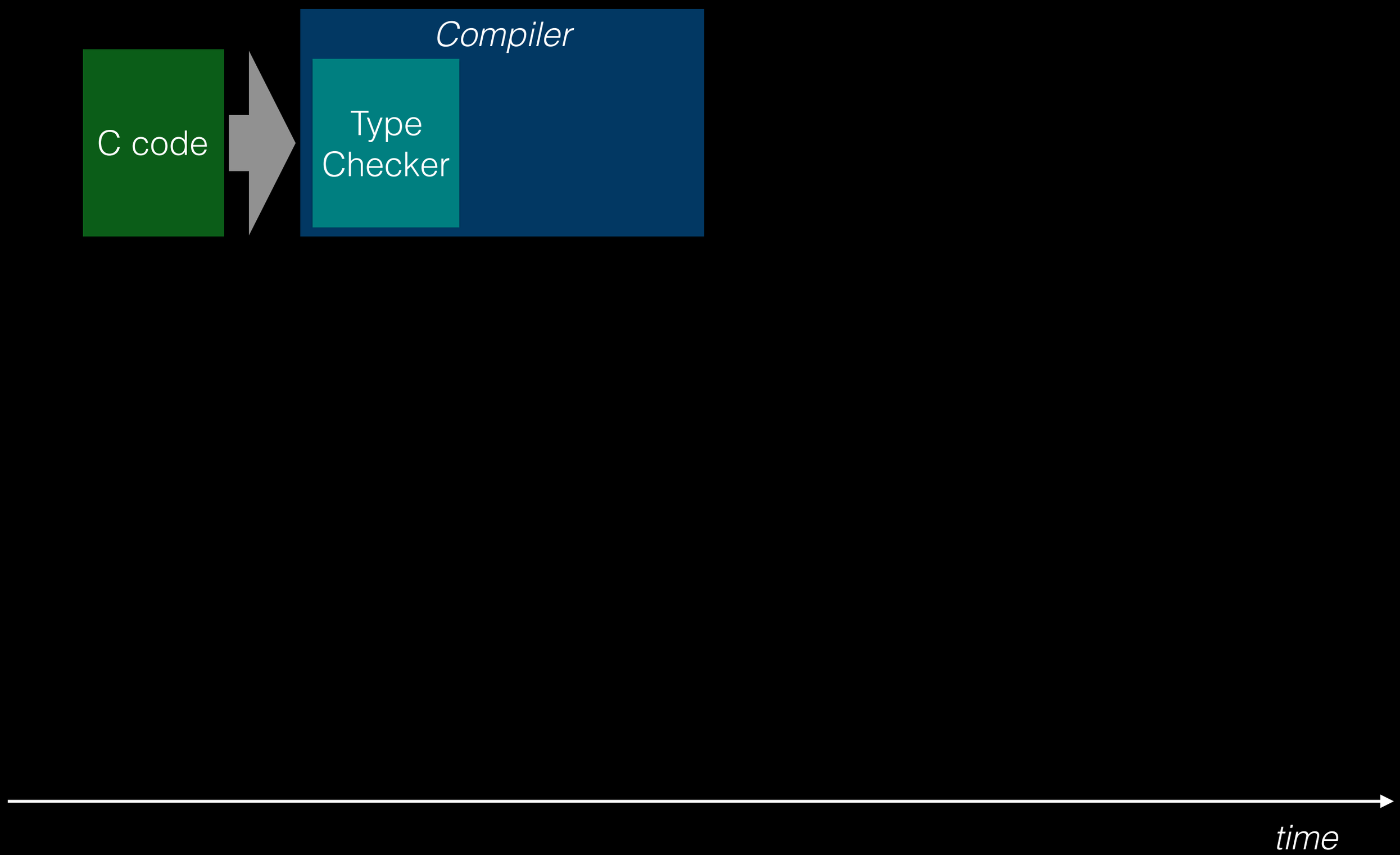

JS code
Type Checker
Optimizing Tier

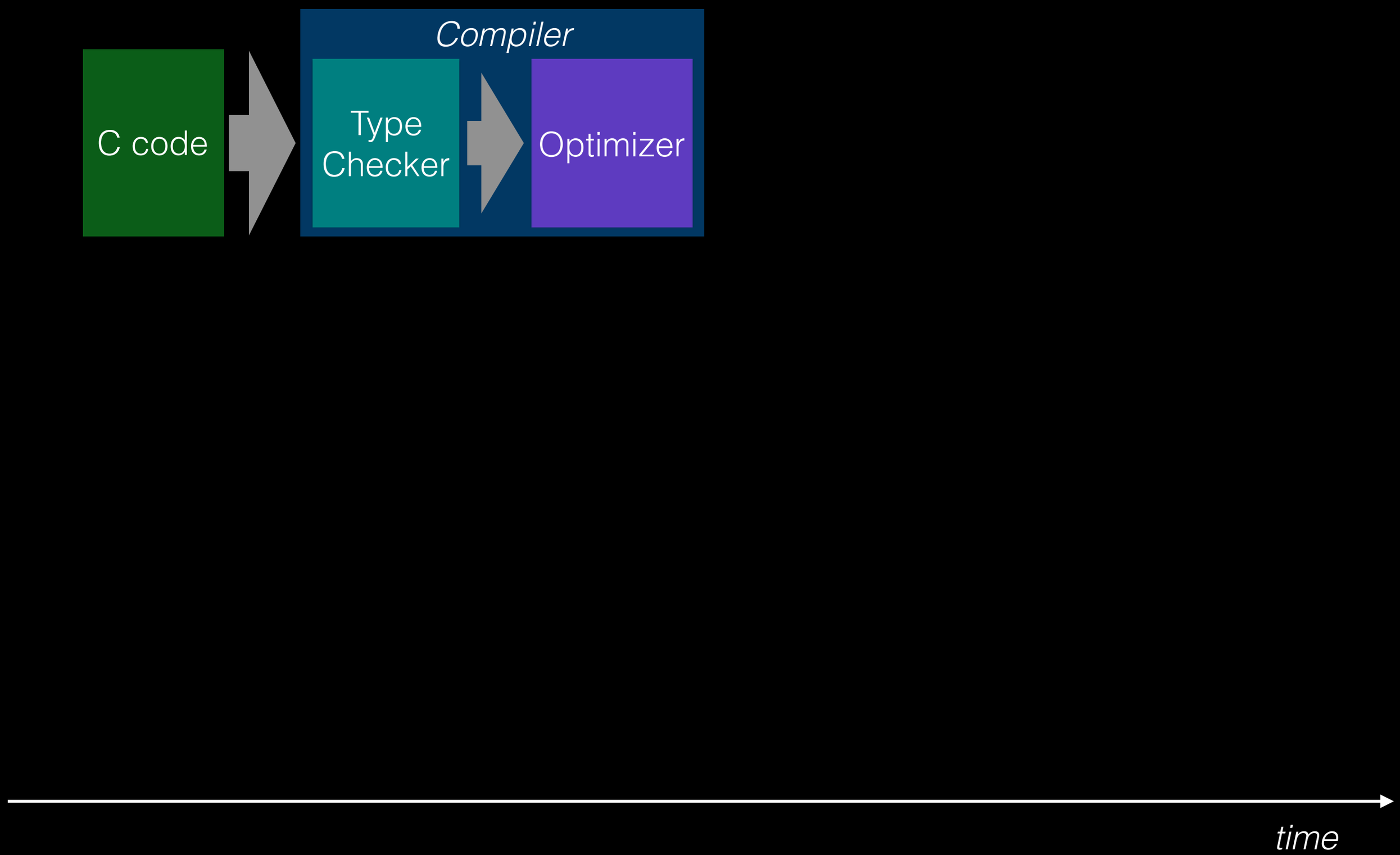

JS code
Profiling Tier
Optimizing Tier

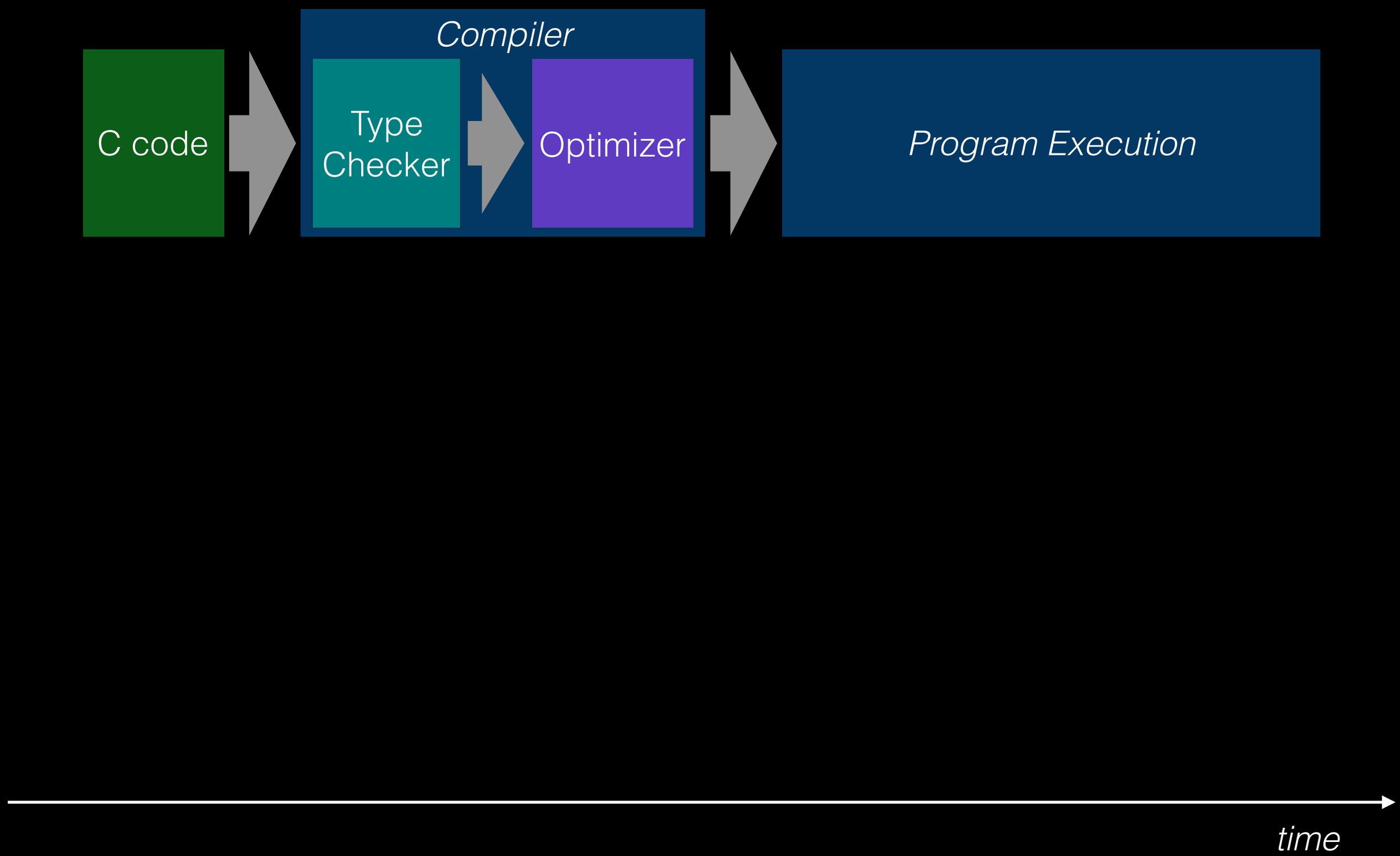

time

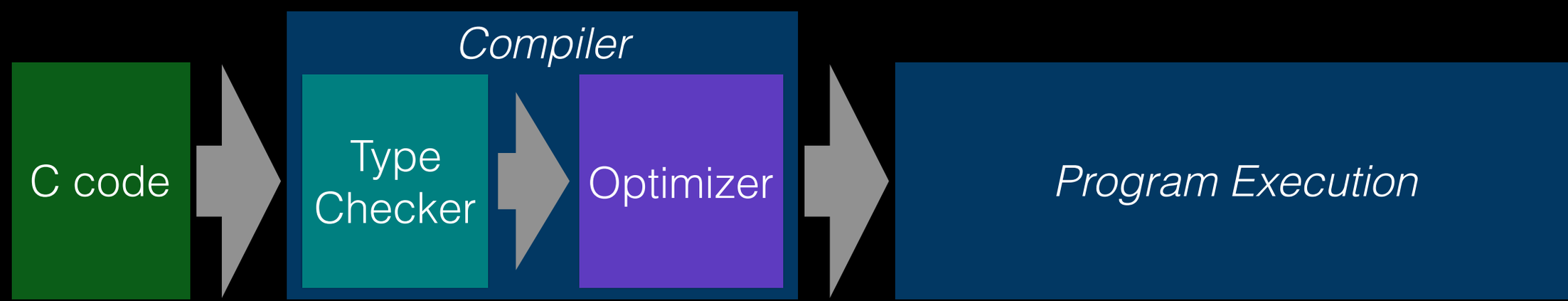

C code
time

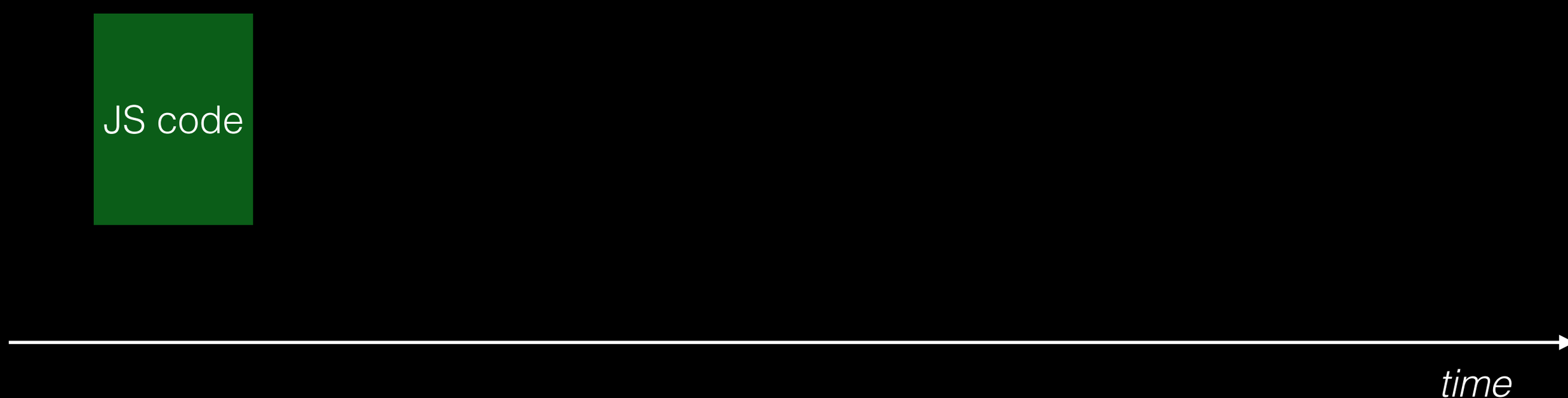

C code
Compiler
time

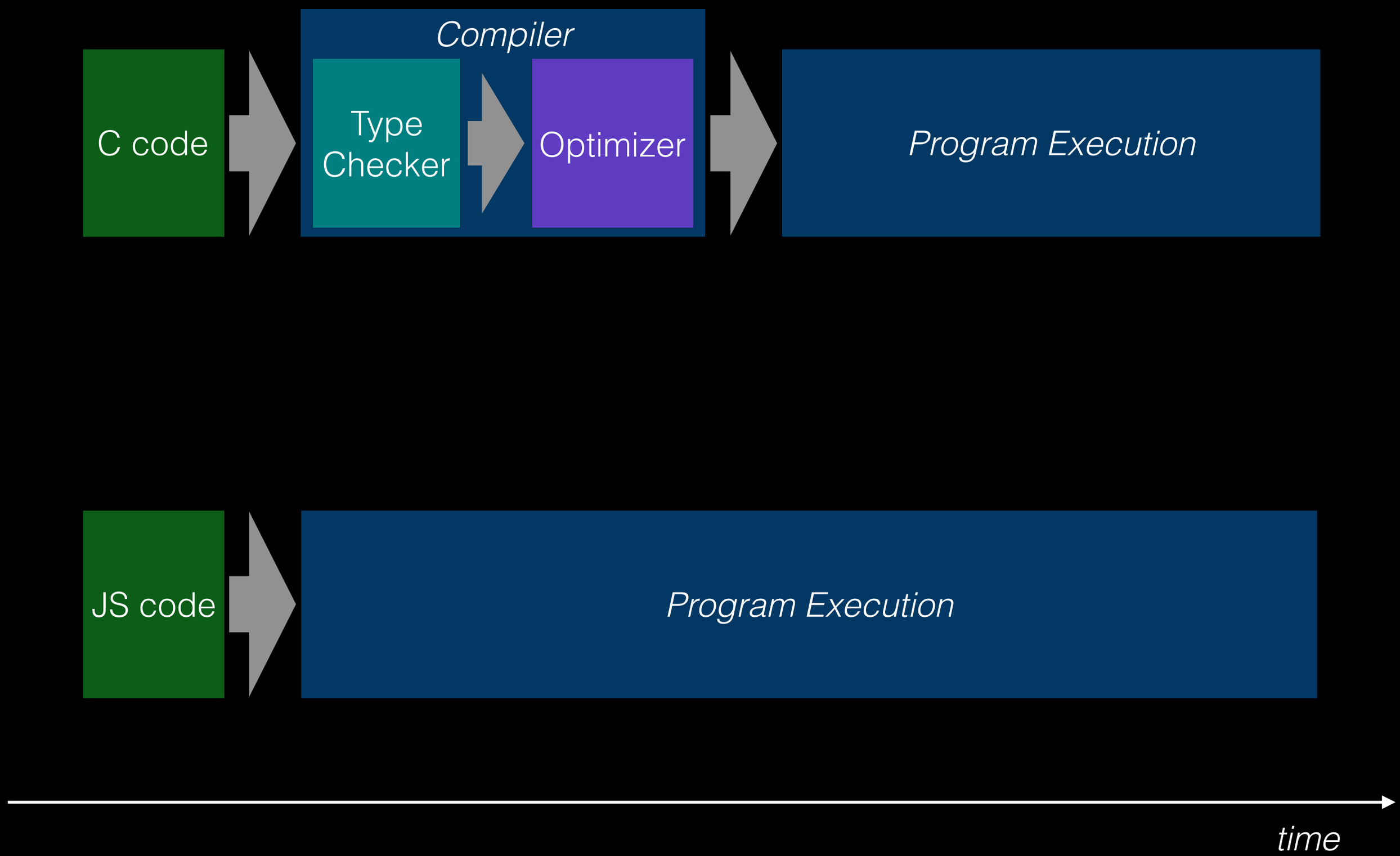

Compiler
C code
Type
Checker
time

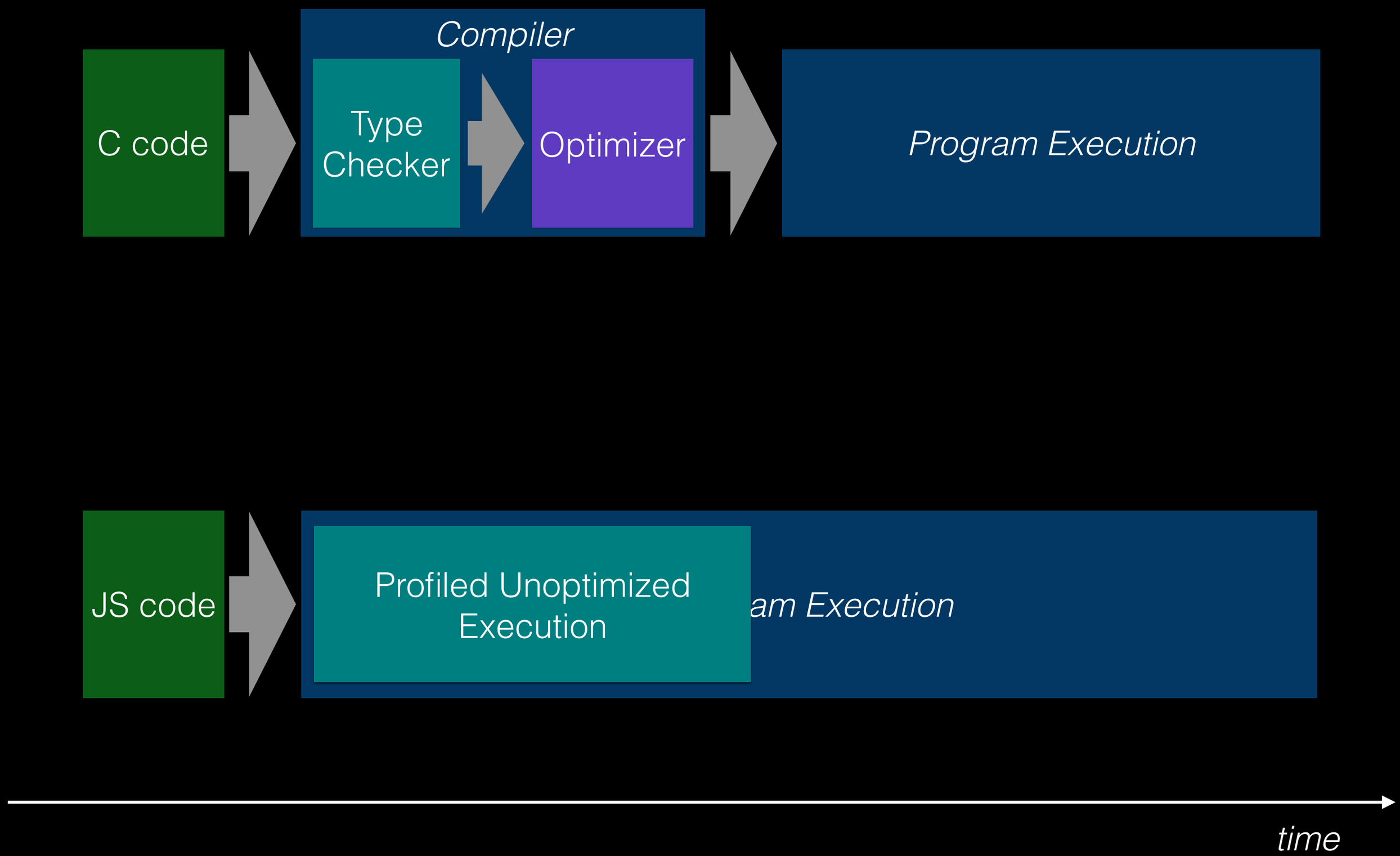

Compiler
C code
Type Checker
Optimizer
time

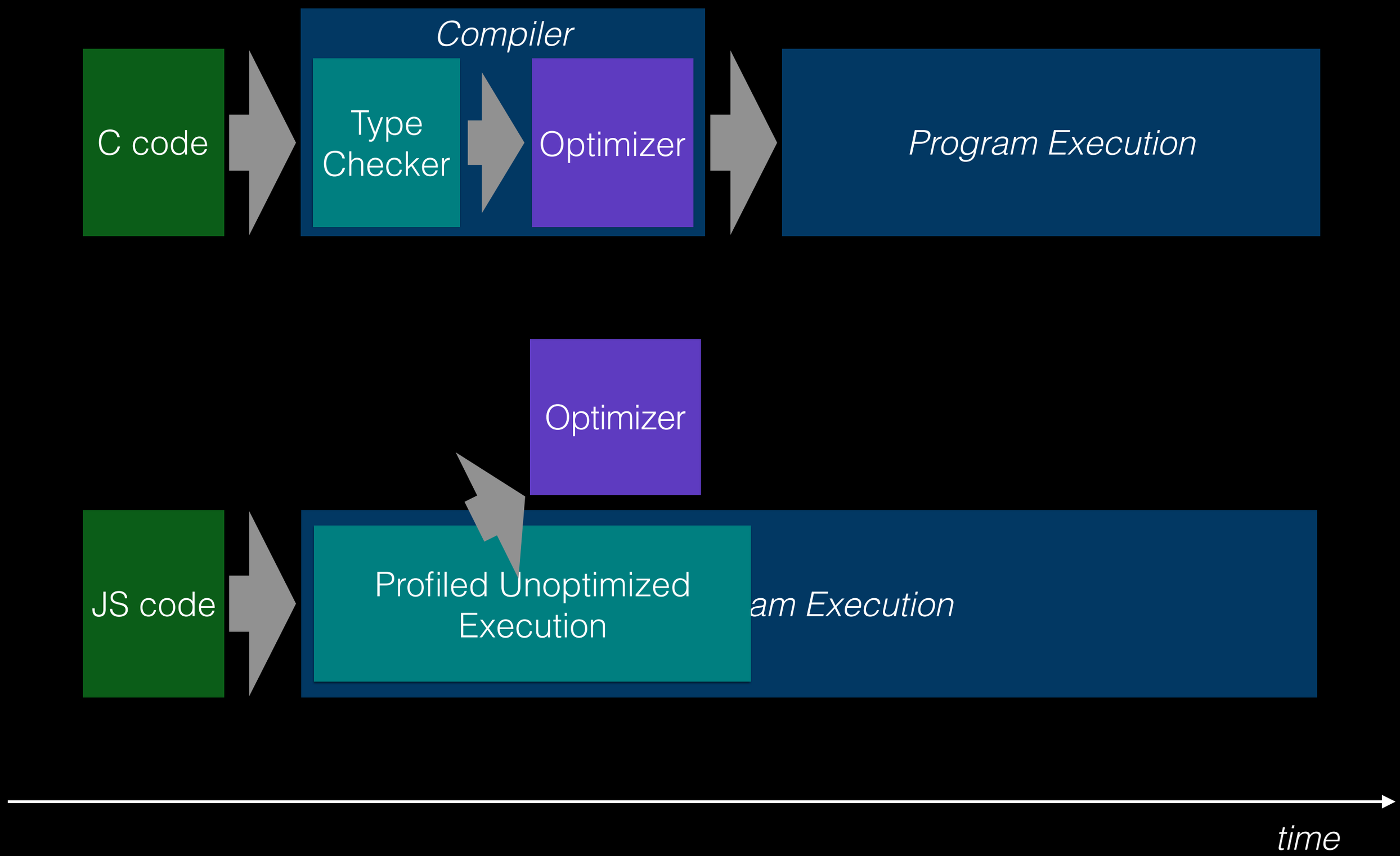

Compiler
C code
Type Checker
Optimizer
Program Execution
time

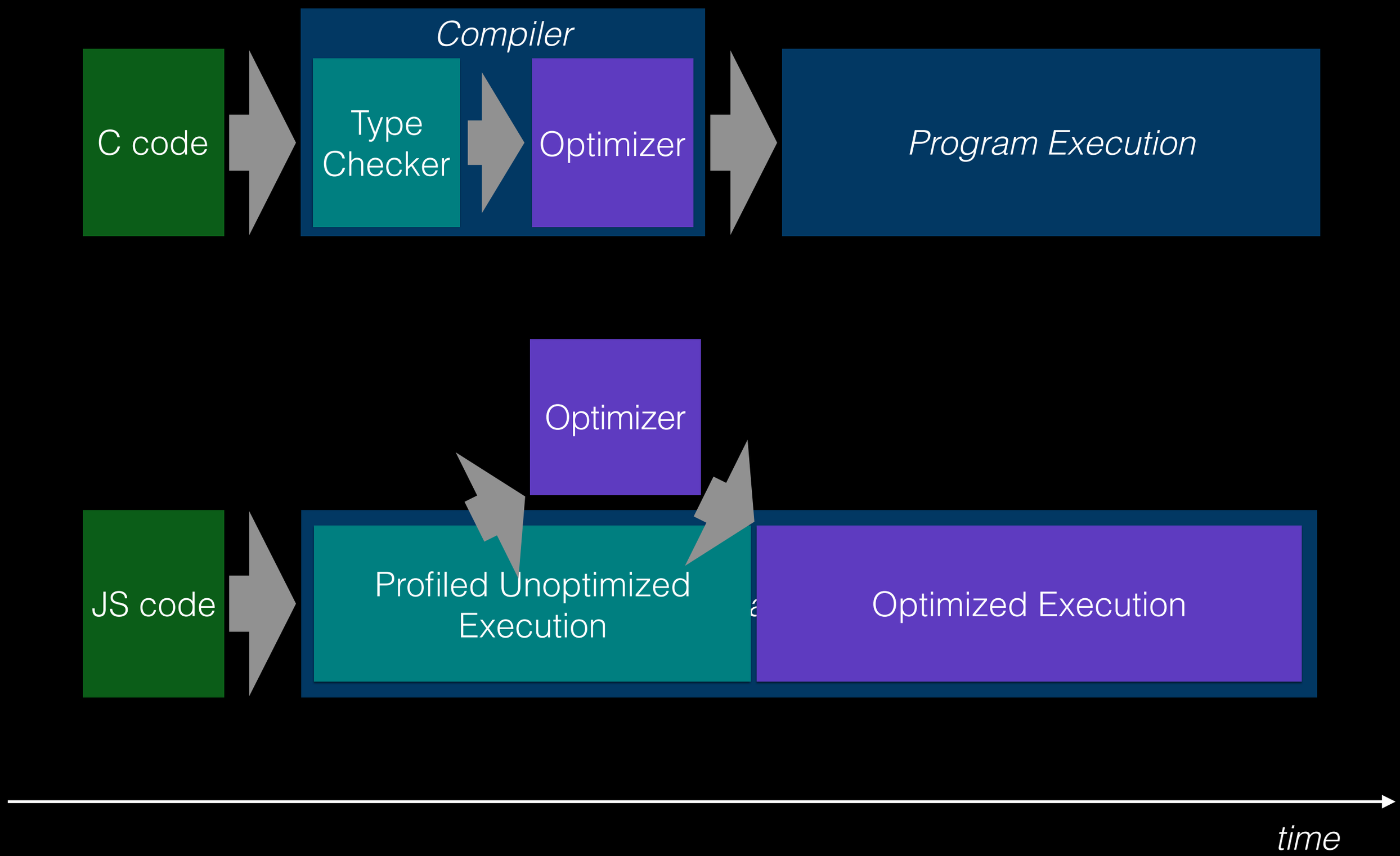

Compiler
C code
Type Checker
Optimizer
Program Execution

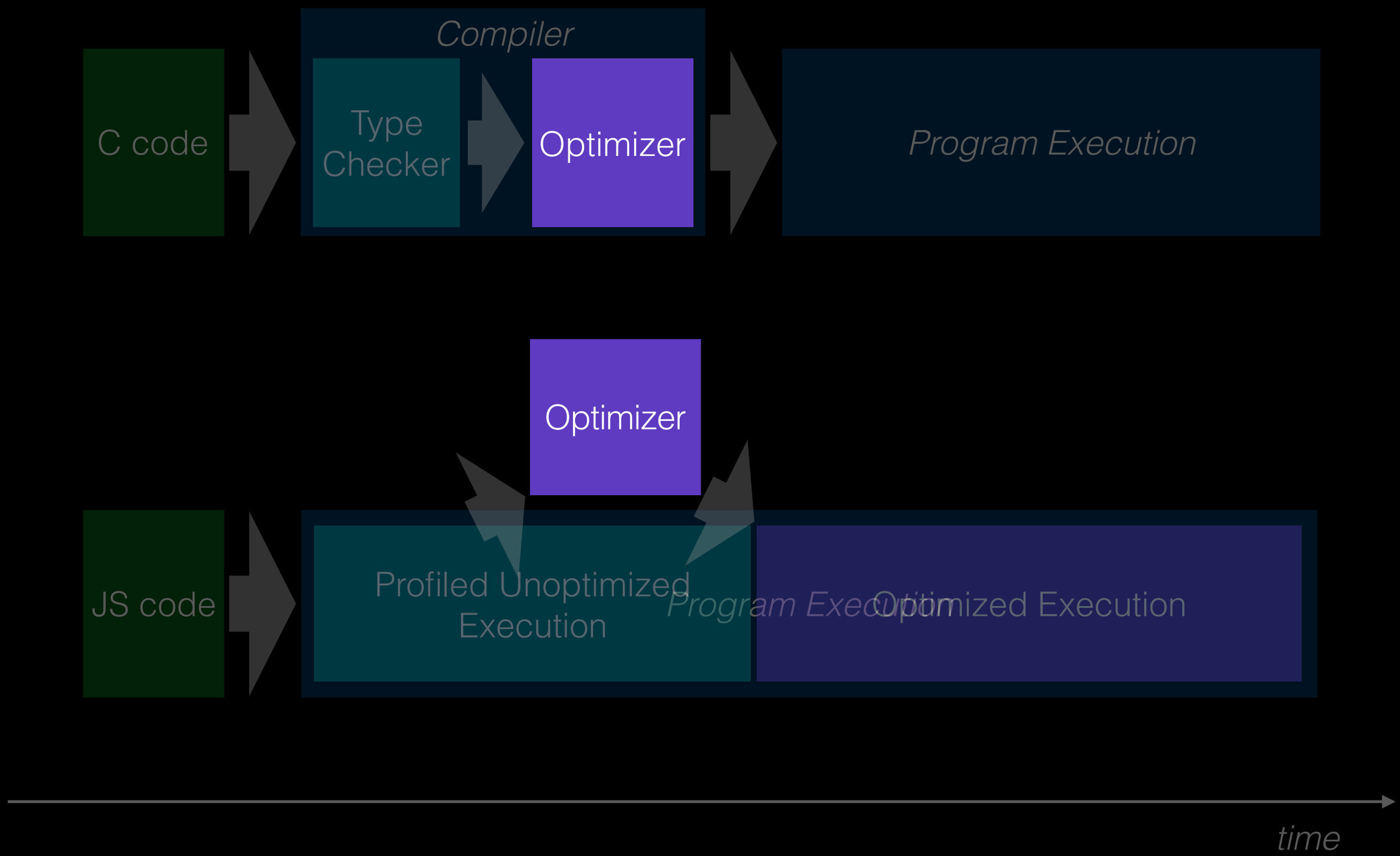

JS code
time


Compiler
C code
Type Checker
Optimizer
Program Execution
JS code
Program Execution
time


Compiler
C code
Type Checker
Optimizer
Program Execution
JS code
Profiled Unoptimized Execution
am Execution
time


Compiler
C code
Type Checker
Optimizer
Program Execution
Optimizer
JS code
Profiled Unoptimized Execution
am Execution
time

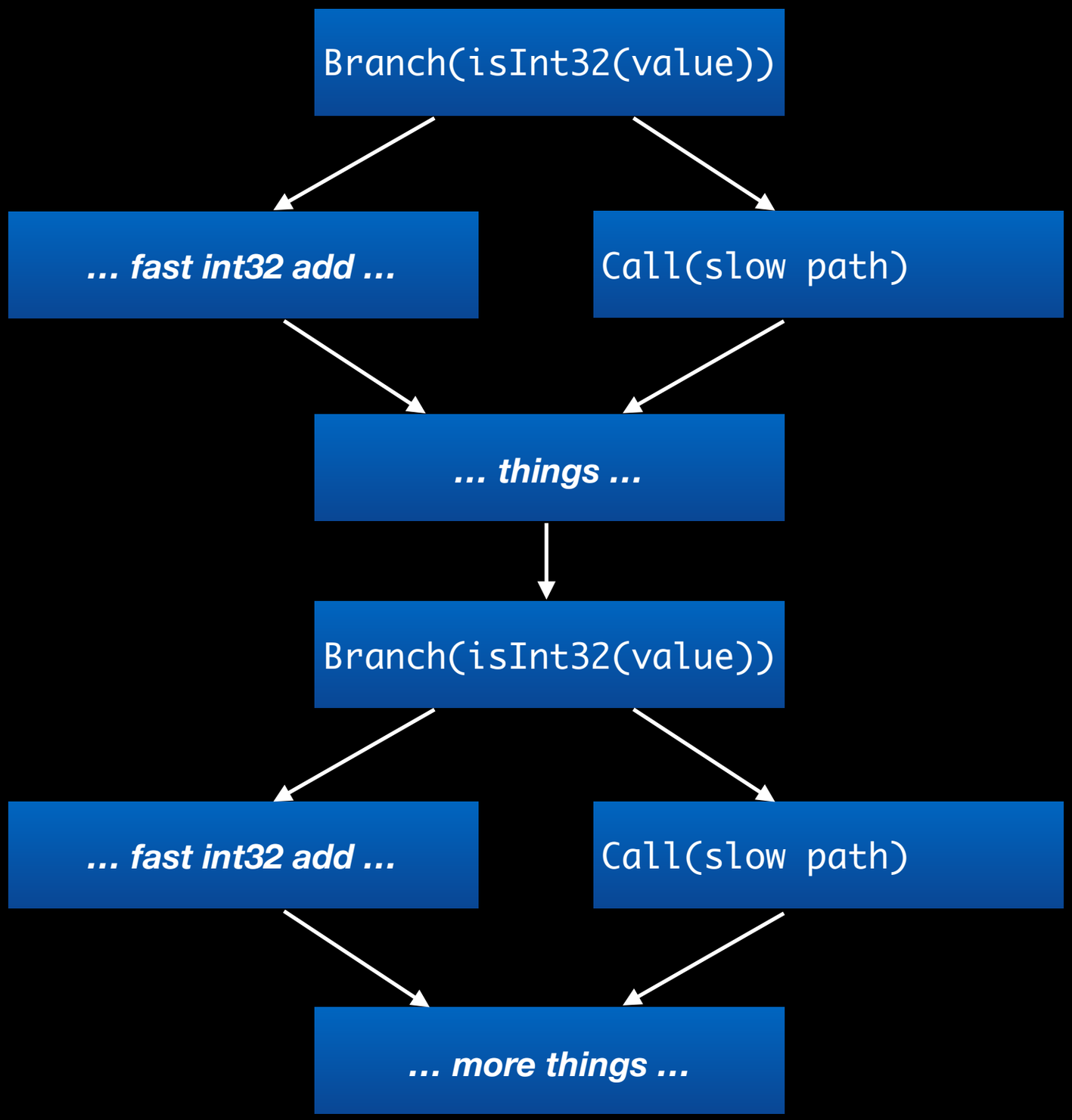

Compiler
C code
Type Checker
Optimizer
Program Execution
Optimizer
JS code
Profiled Unoptimized Execution
Optimized Execution
time


Compiler
C code
Type Checker
Optimizer
Program Execution
Optimizer
JS code
Profiled Unoptimized Execution
Program Execution
Optimized Execution
time

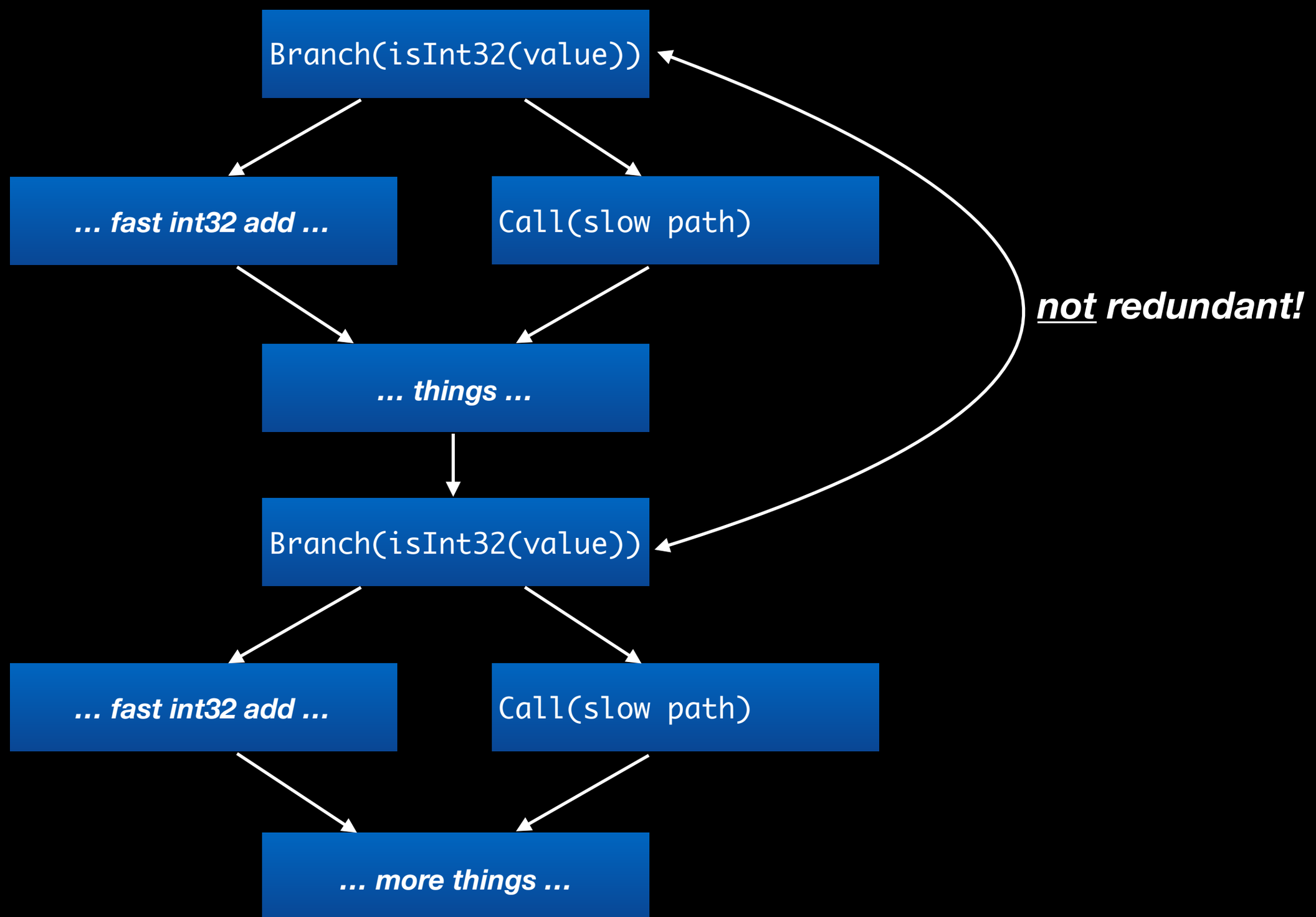
Optimized JS function

```
function foo(a, b)
{
    speculate(isInt32(a));
    speculate(isInt32(b));
    return a + b;
}
```

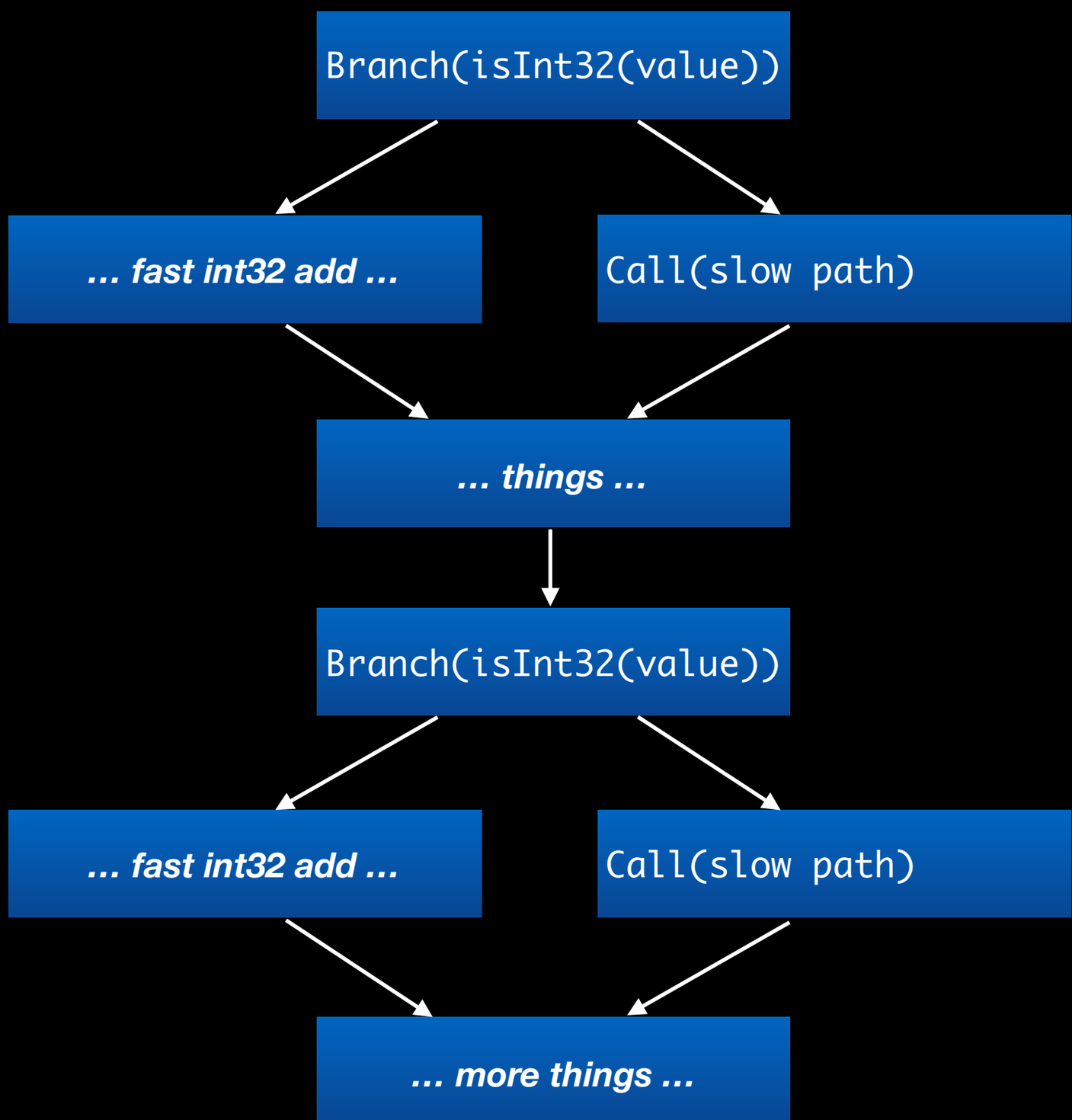
Speculation with Control Flow Diamond



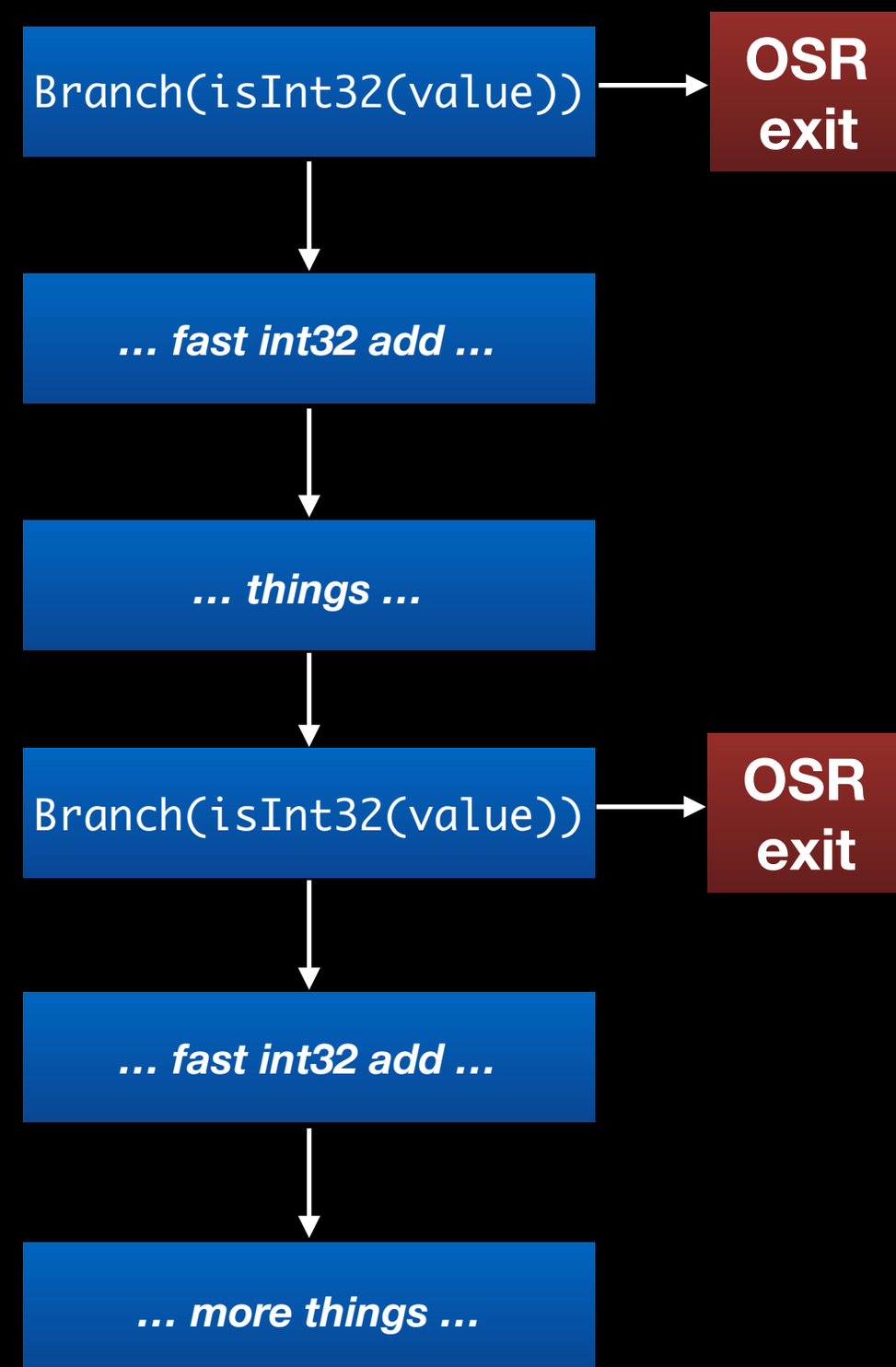
Speculation with Control Flow Diamond



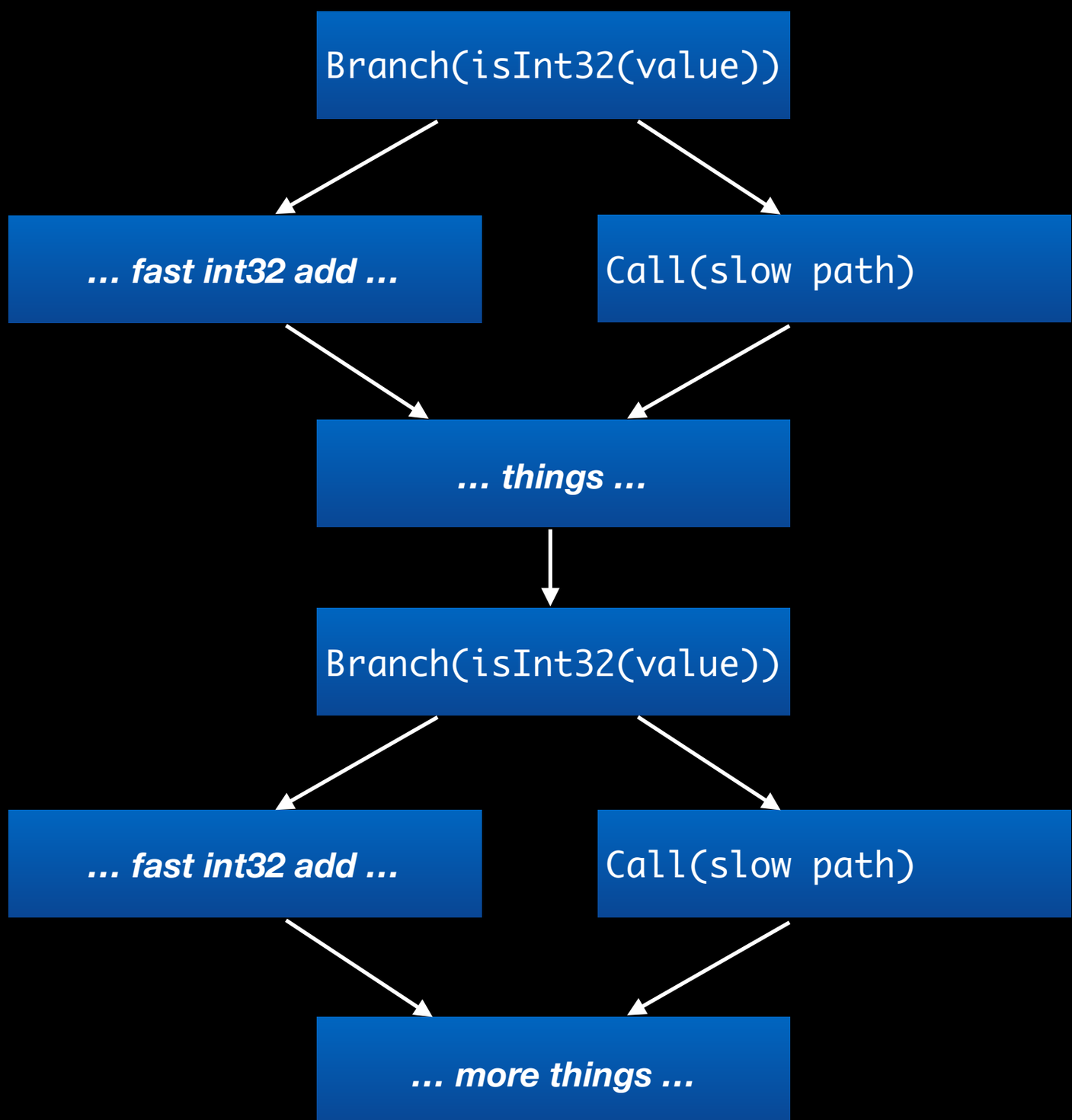
Speculation with Control Flow Diamond



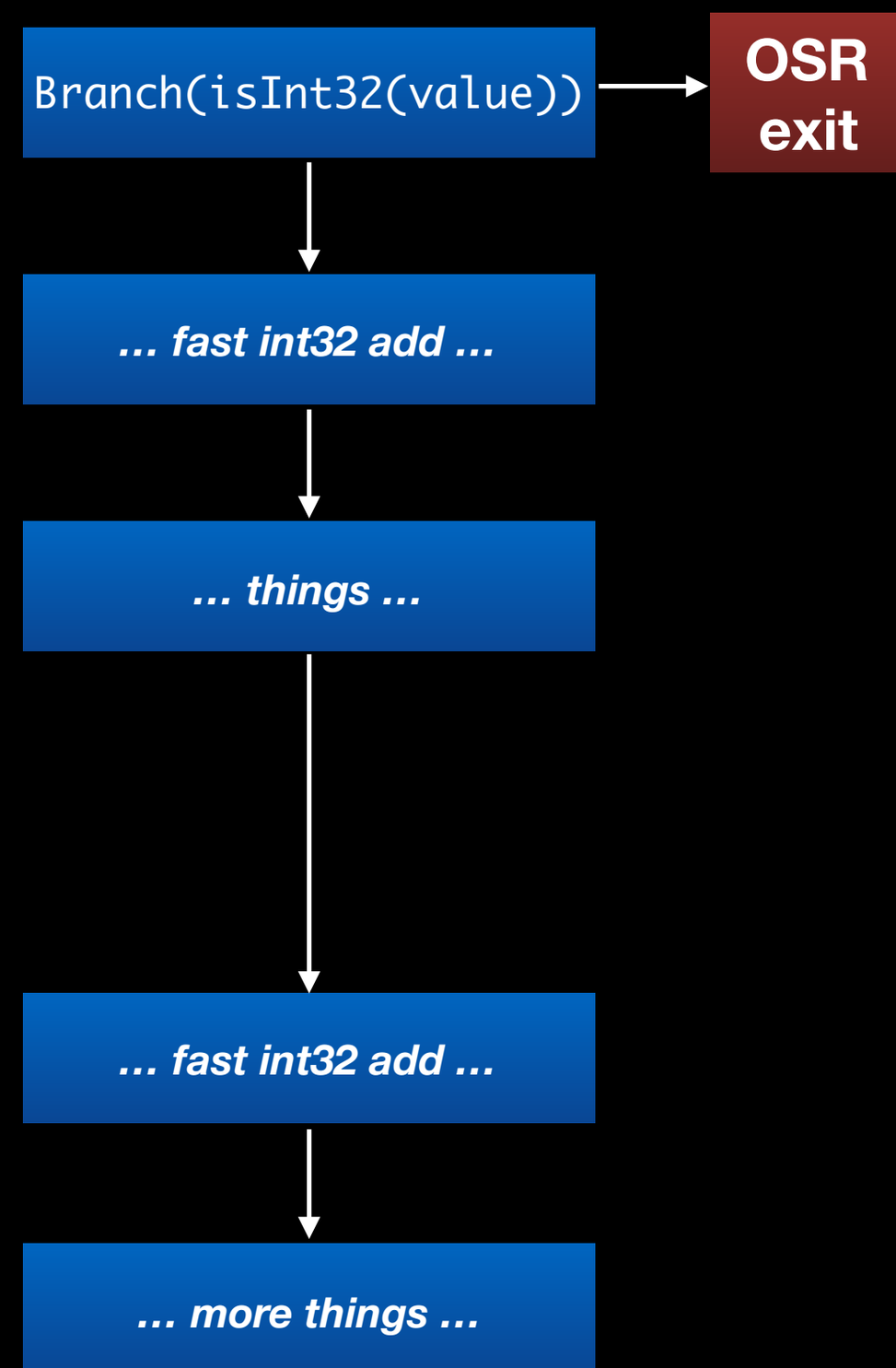
Speculation with OSR



Speculation with Control Flow Diamond



Speculation with OSR



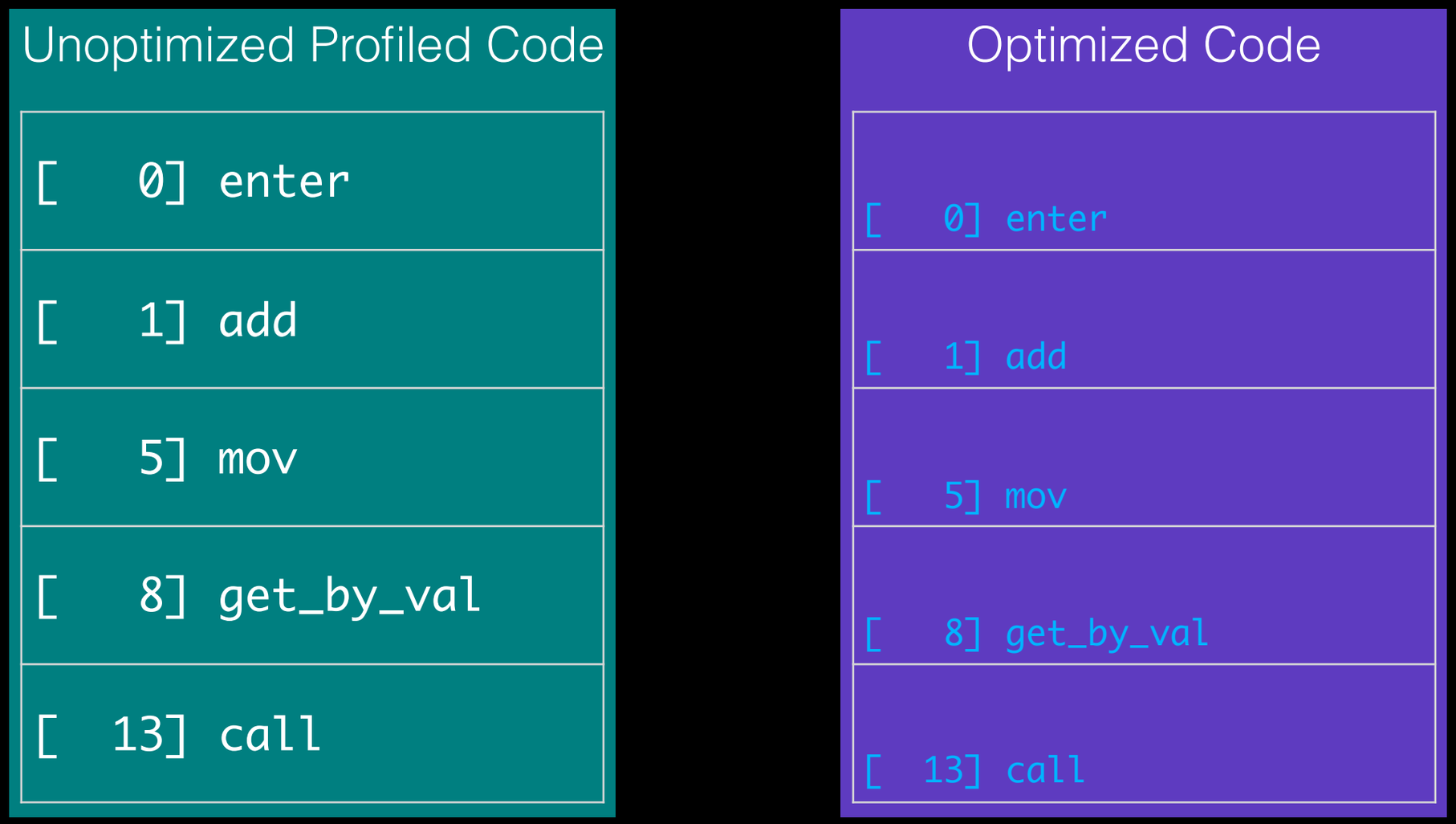
Speculation with OSR

Speculation with OSR

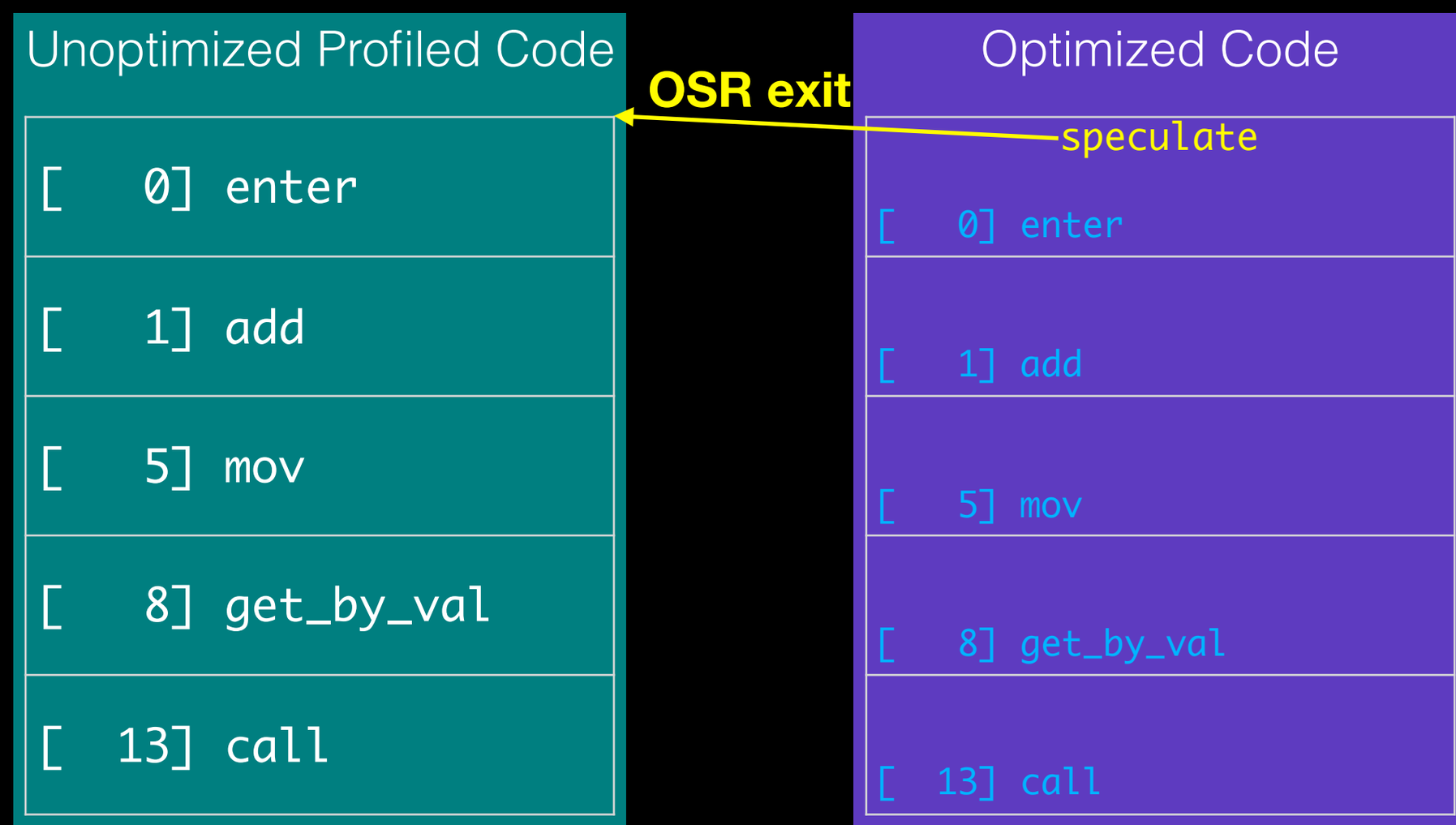
Unoptimized Profiled Code

```
[   0] enter
[   1] add
[   5] mov
[   8] get_by_val
[  13] call
```

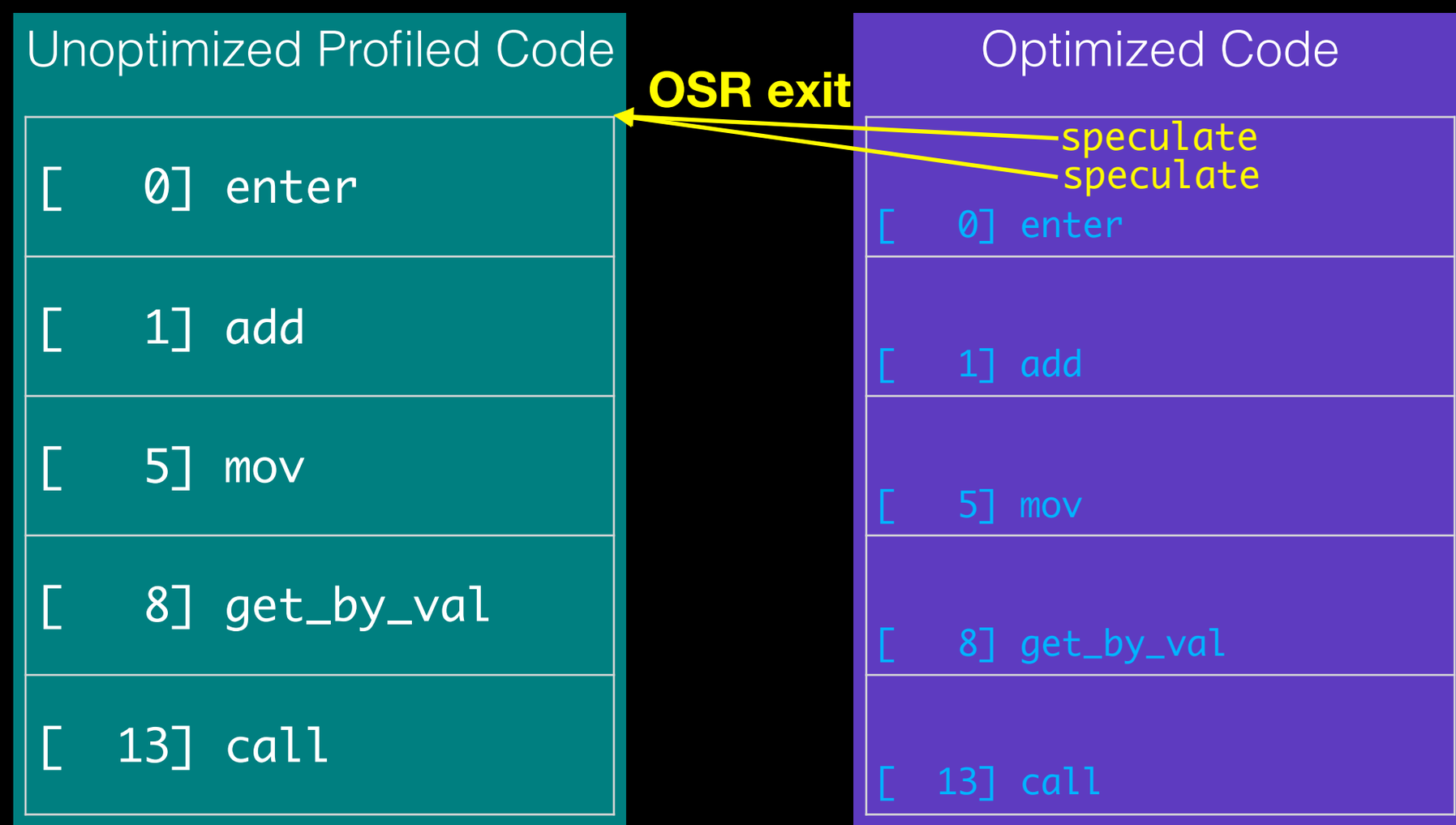
Speculation with OSR



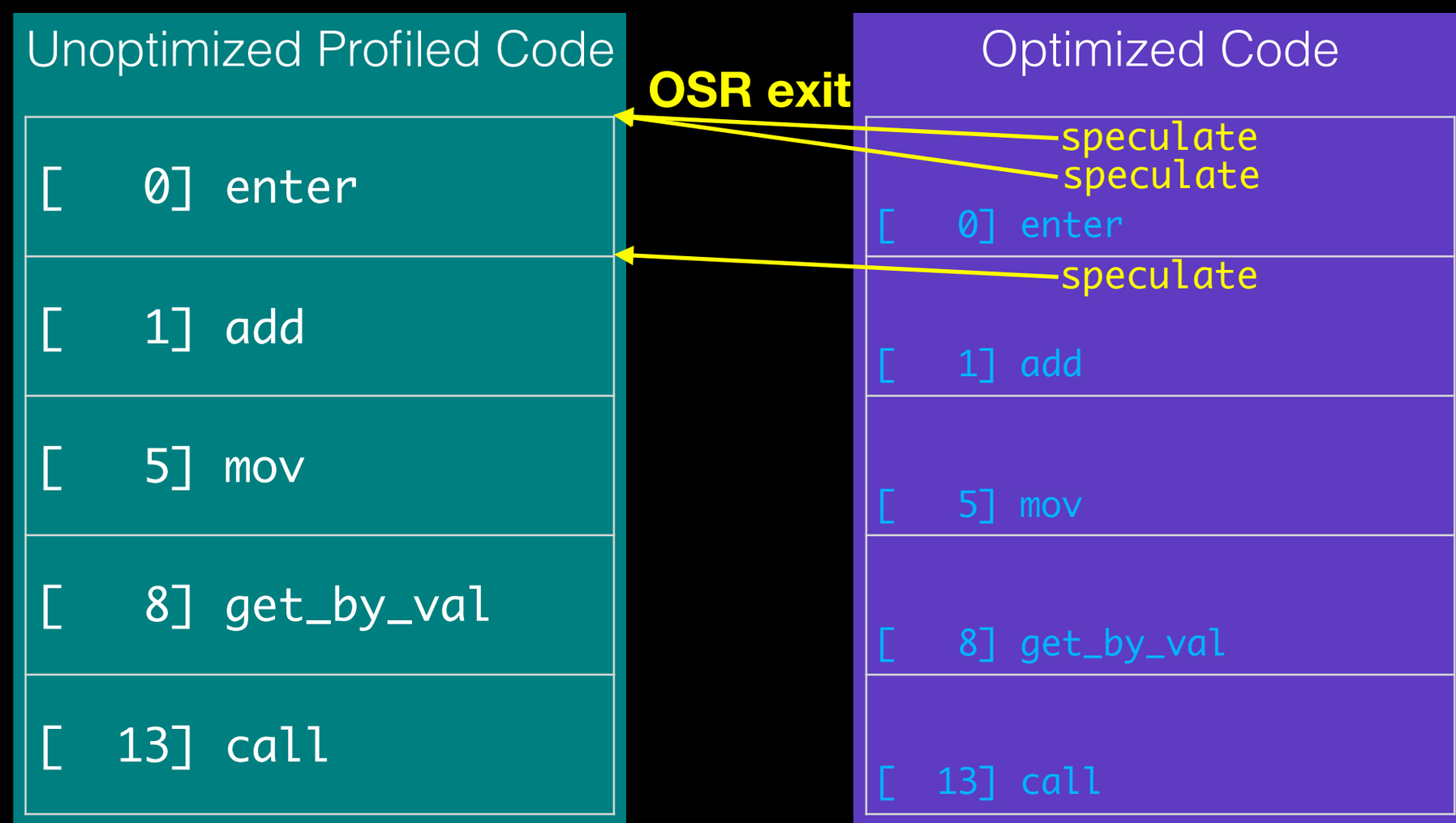
Speculation with OSR



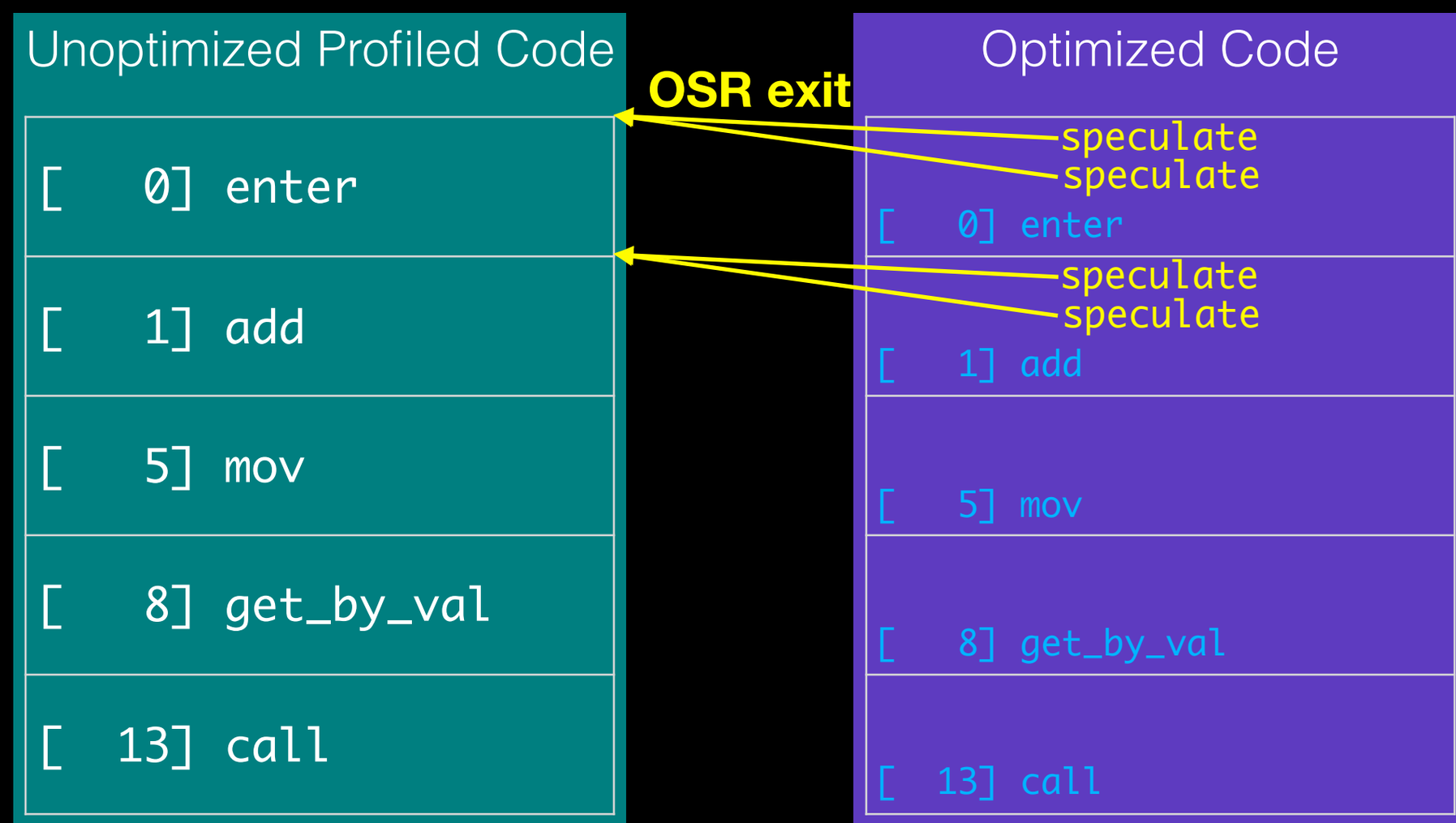
Speculation with OSR



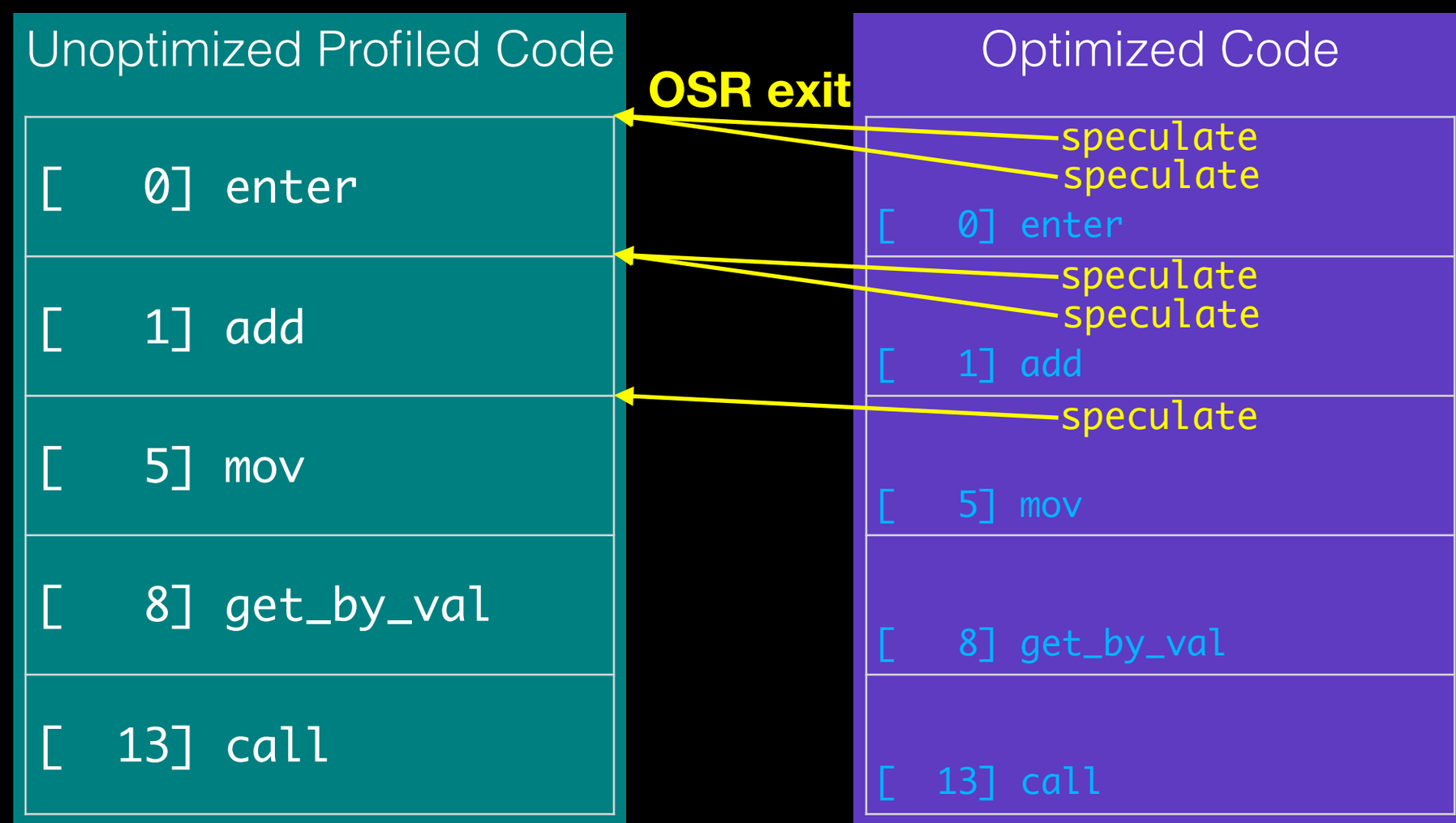
Speculation with OSR



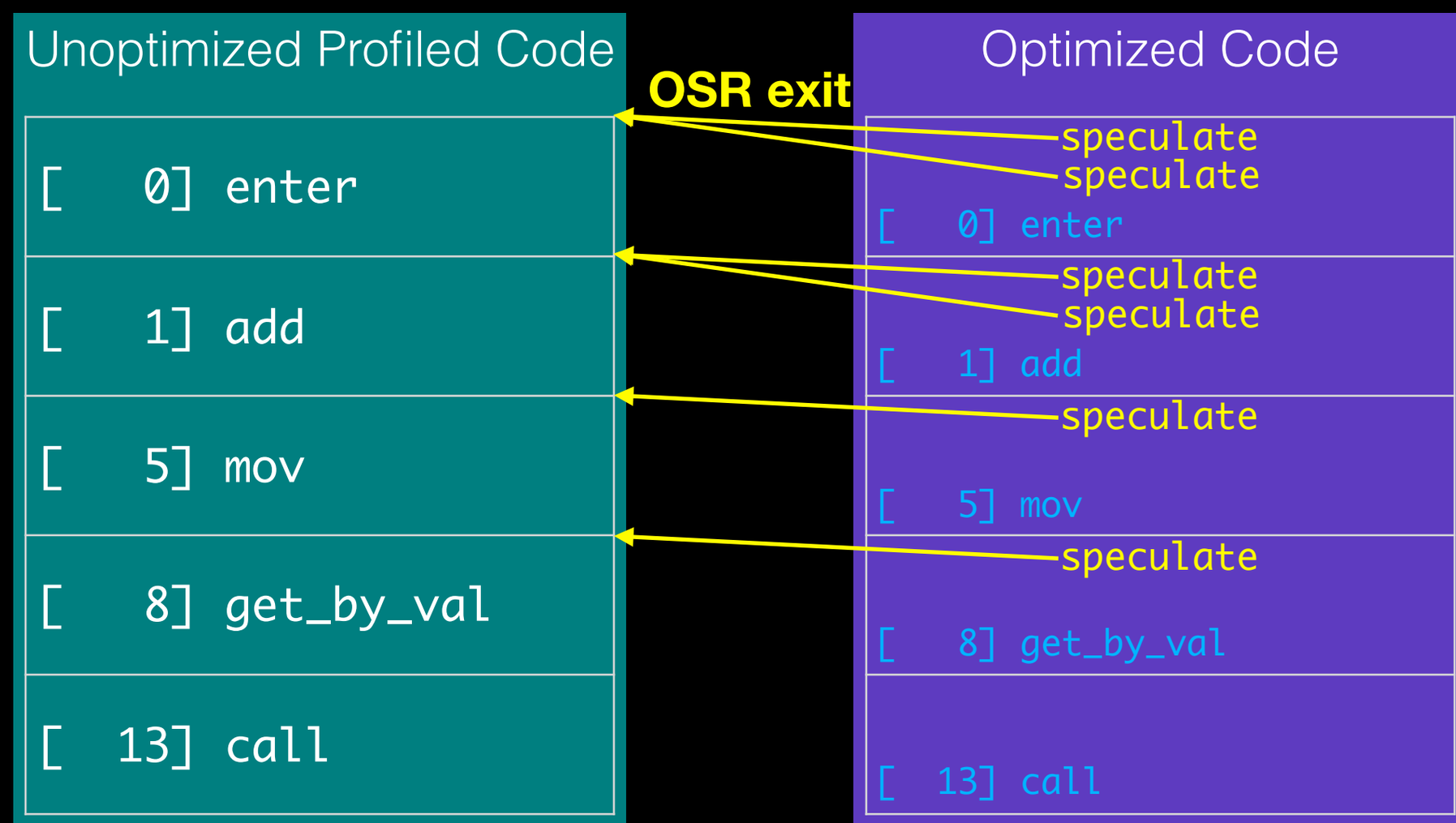
Speculation with OSR



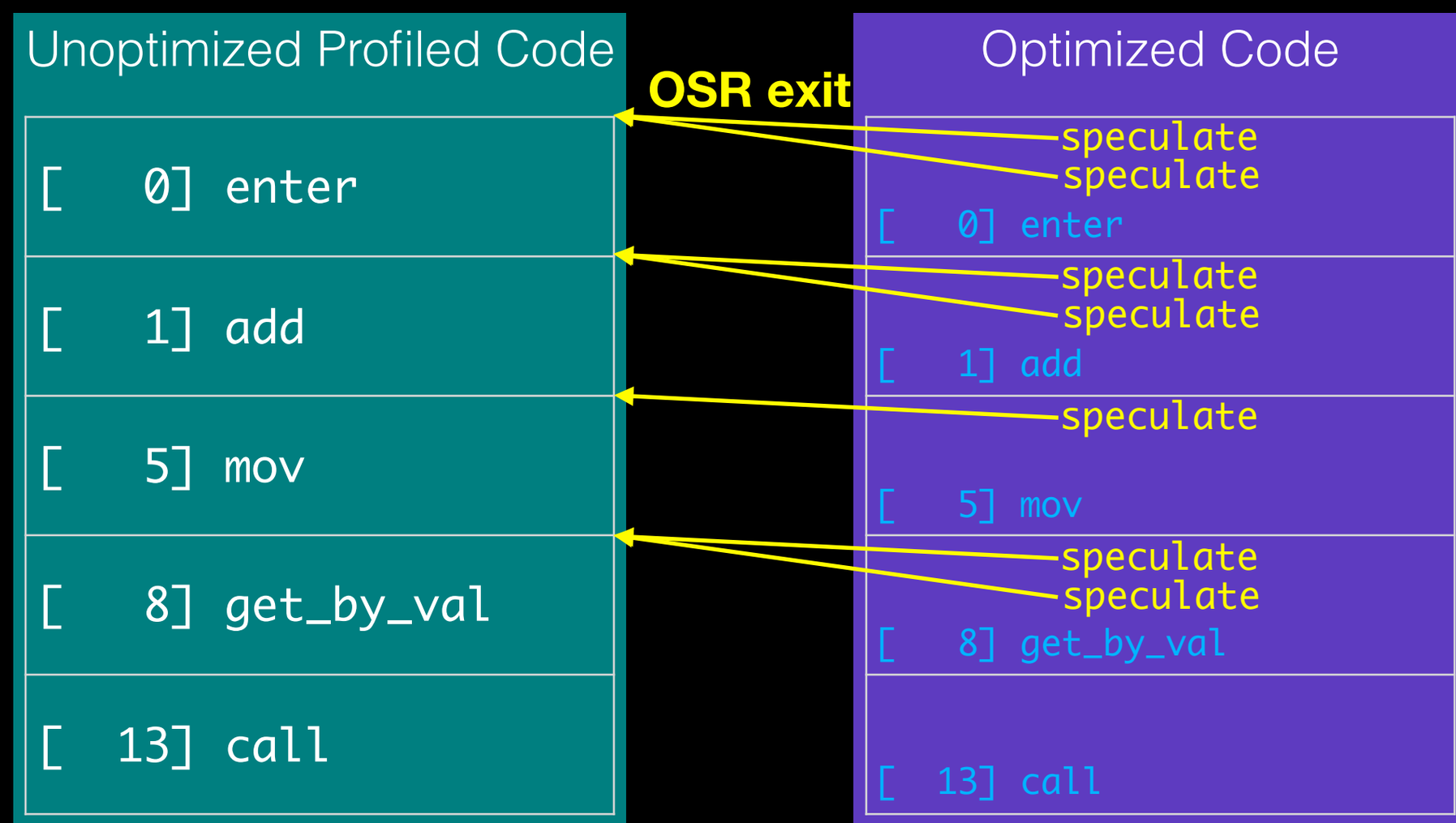
Speculation with OSR



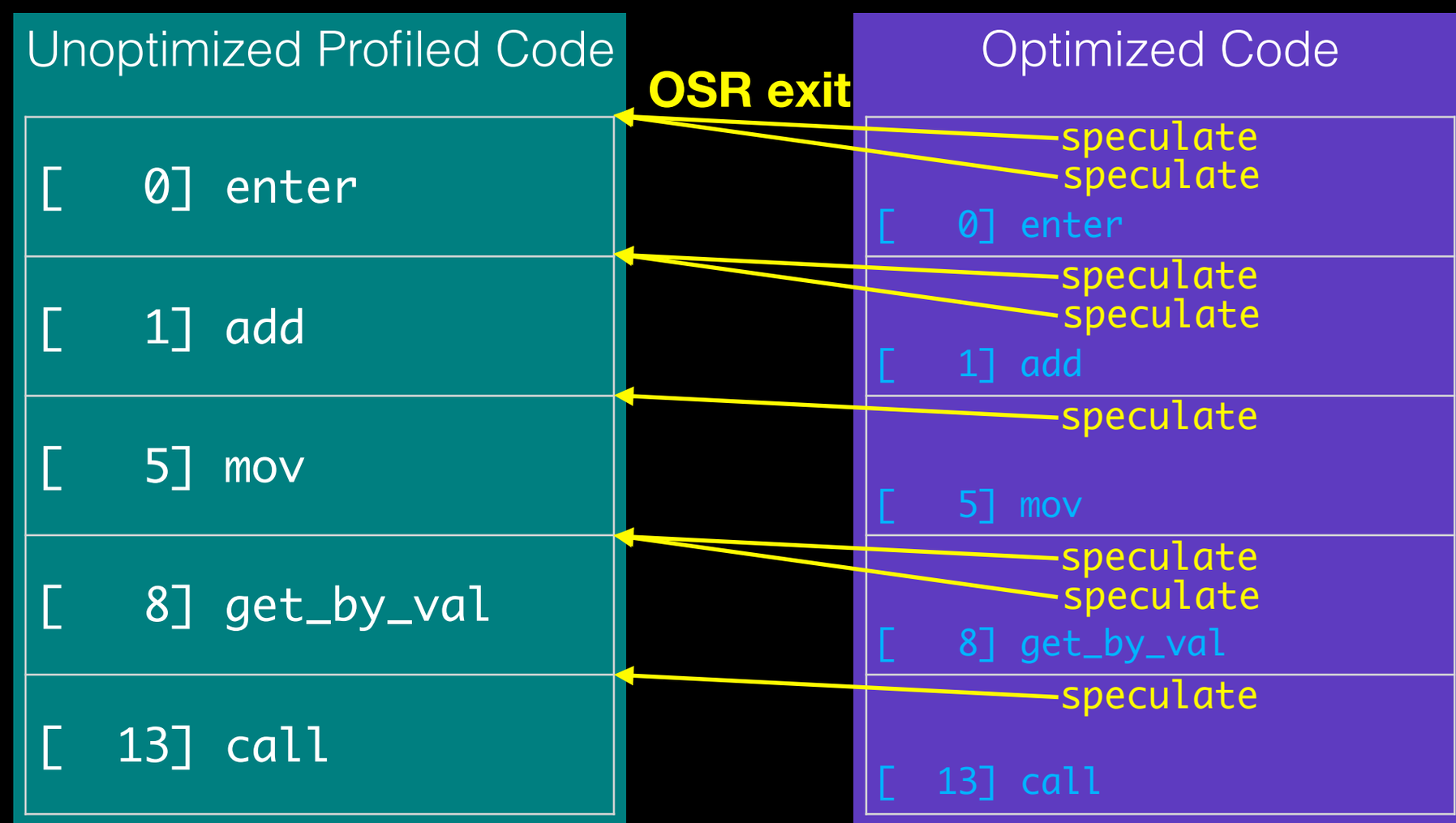
Speculation with OSR



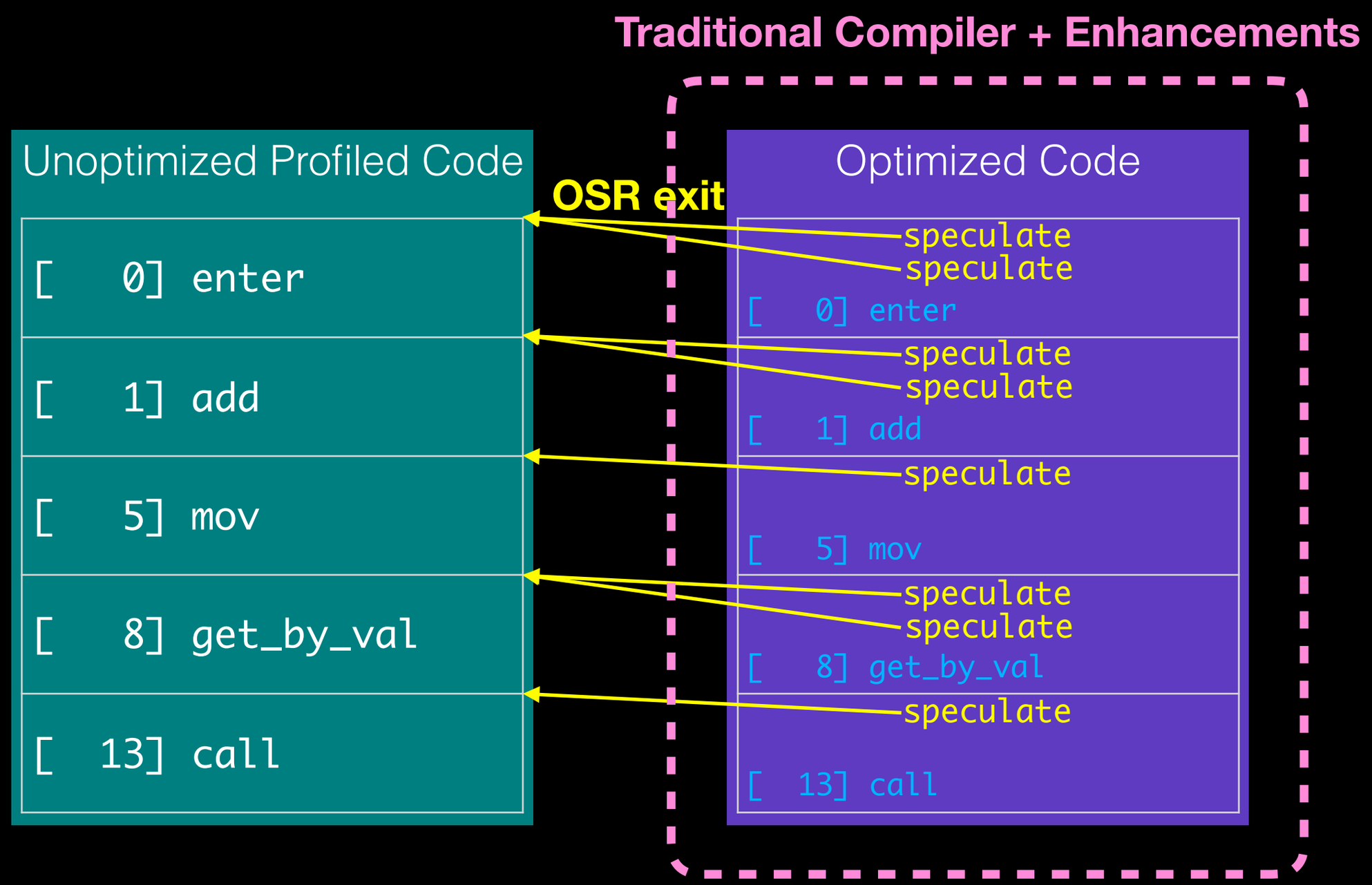
Speculation with OSR



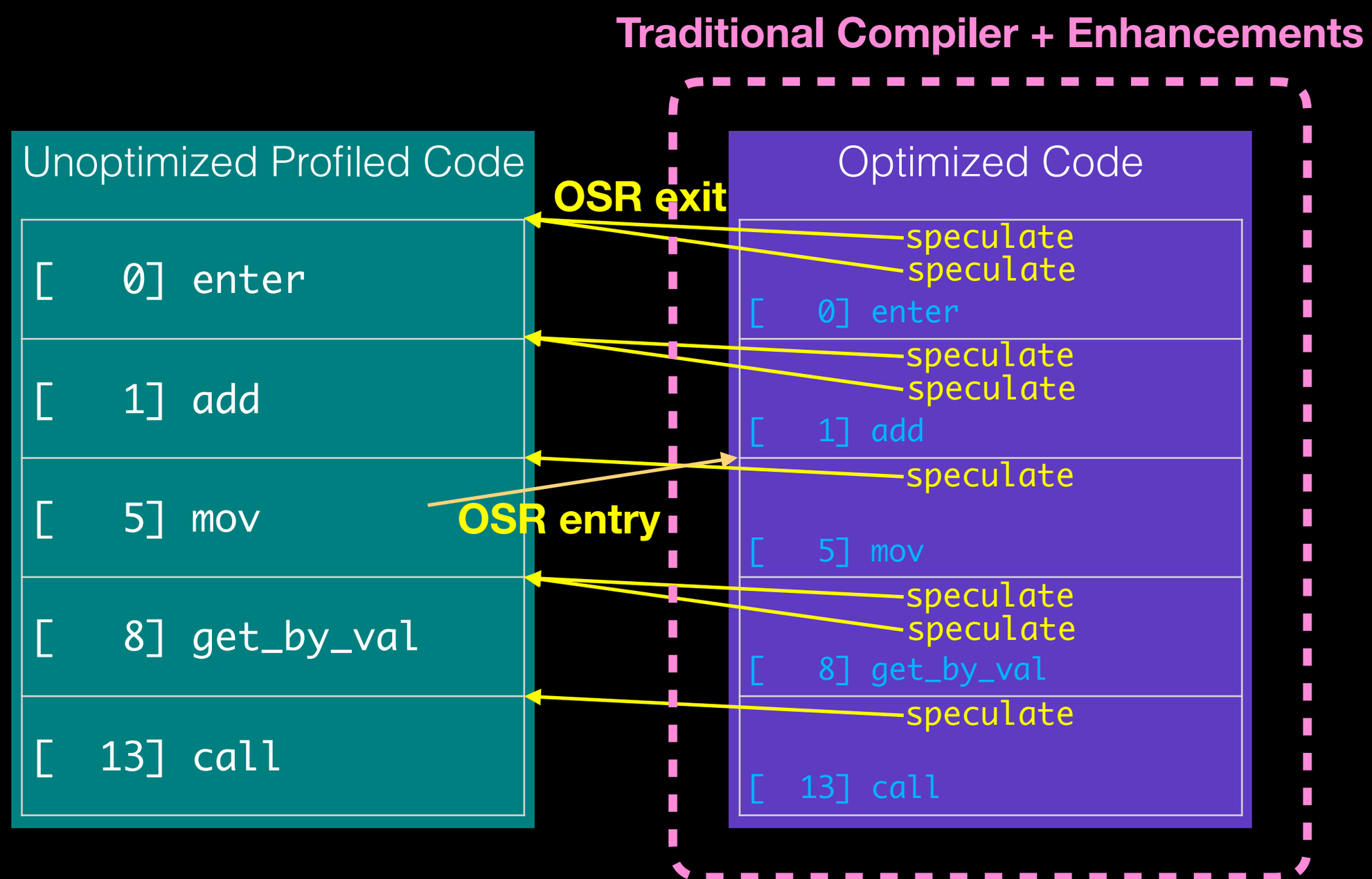
Speculation with OSR



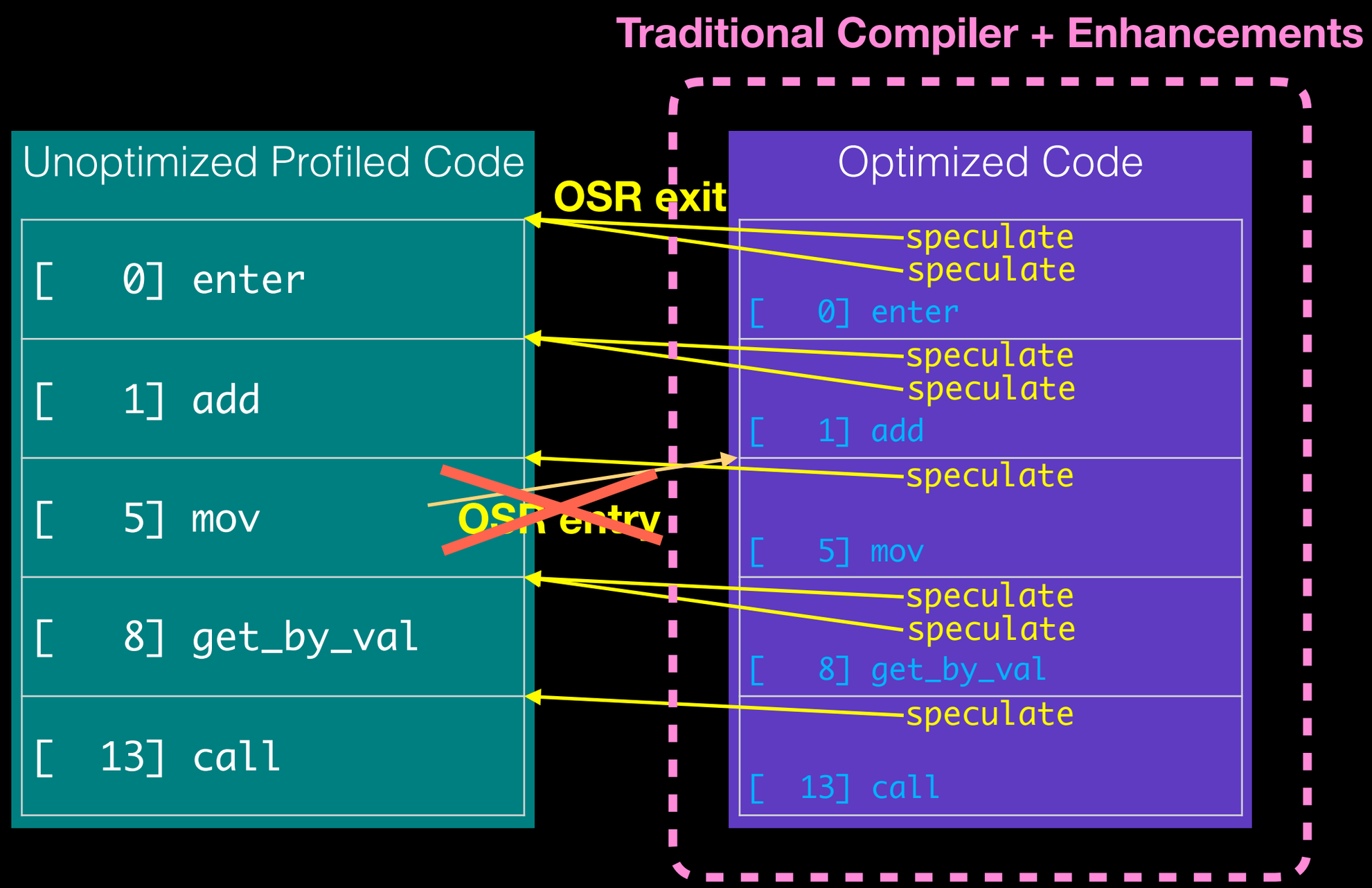
Speculation with OSR



Speculation with OSR



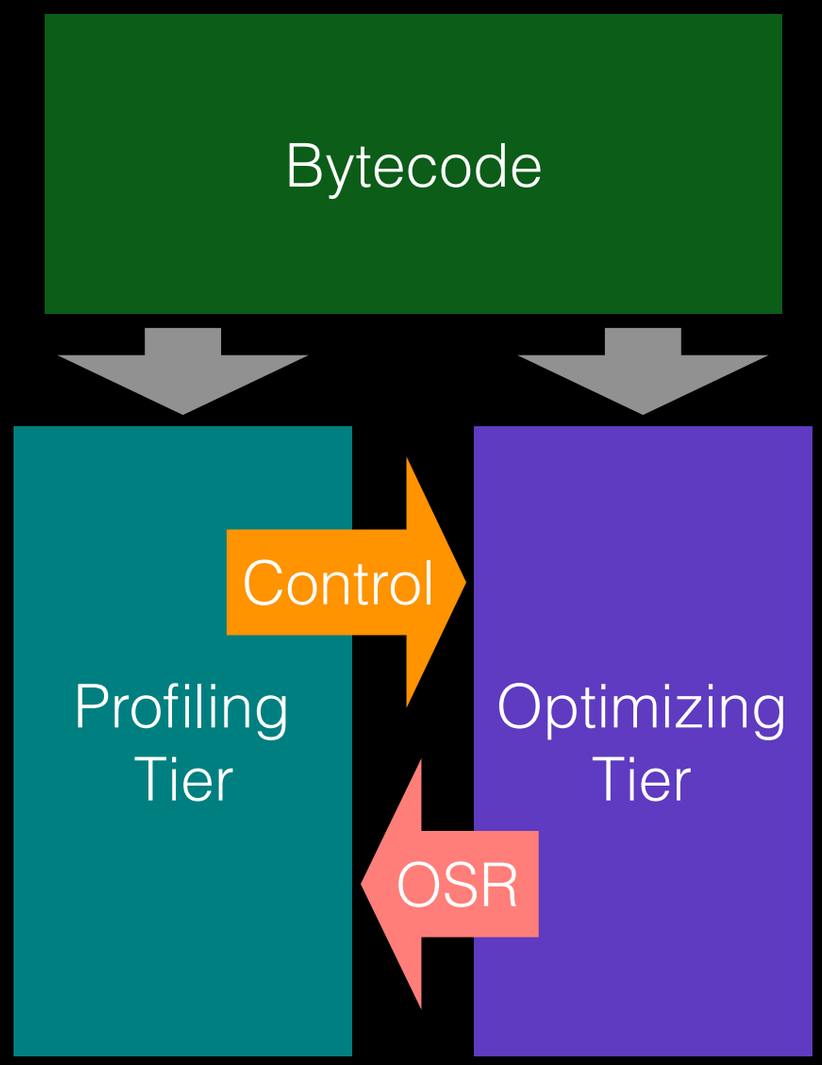
Speculation with OSR



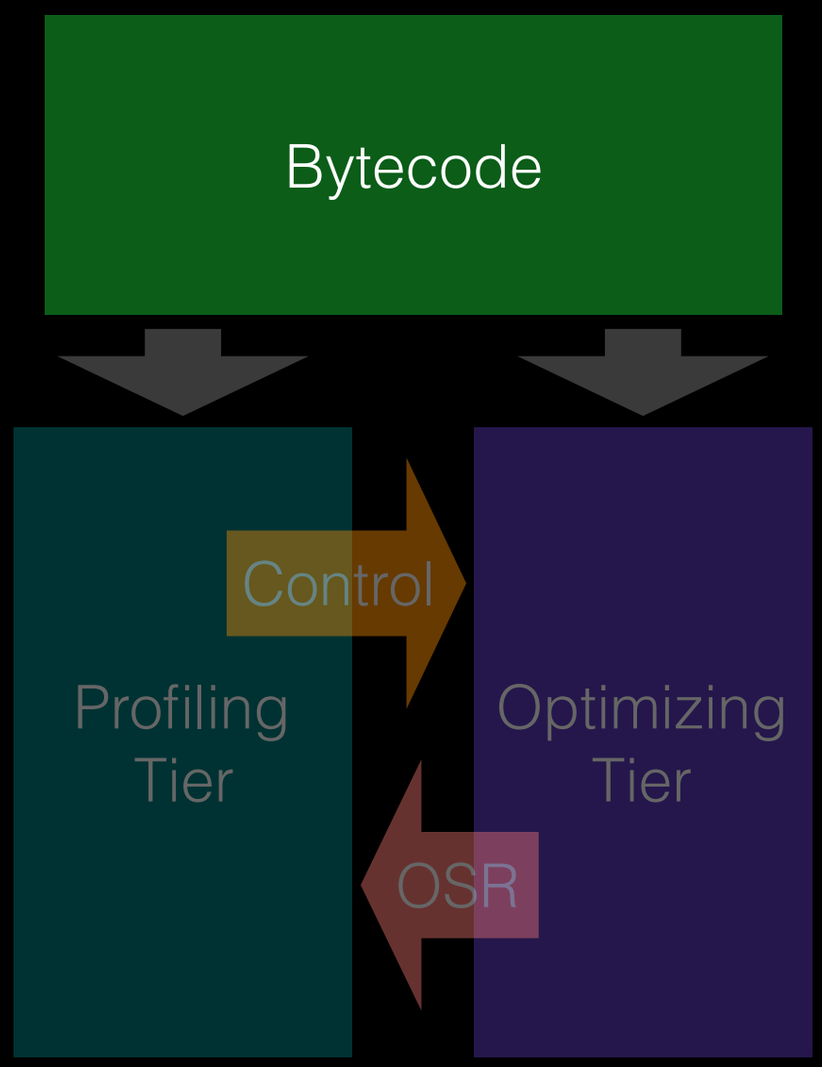
Speculation Has A Function Granularity Bias

- Compiler sees single-entrypoint function + inlines.
- Speculations exit the function and rarely reenter.

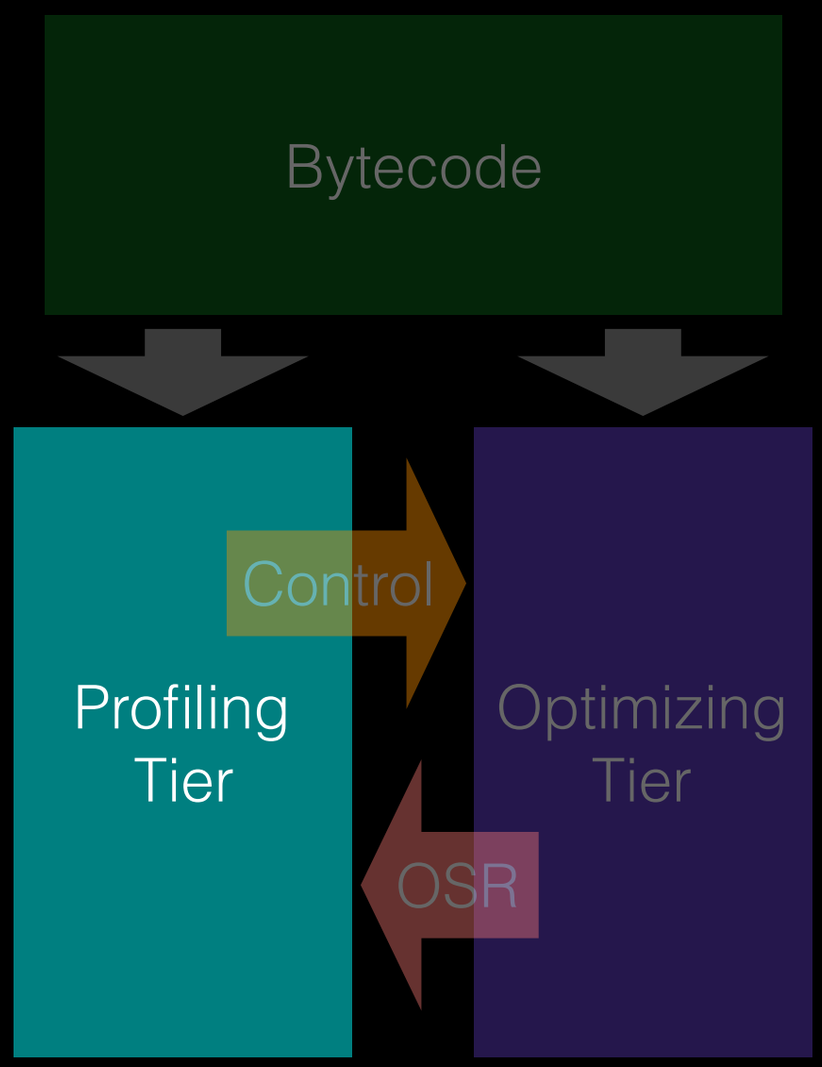
Speculation



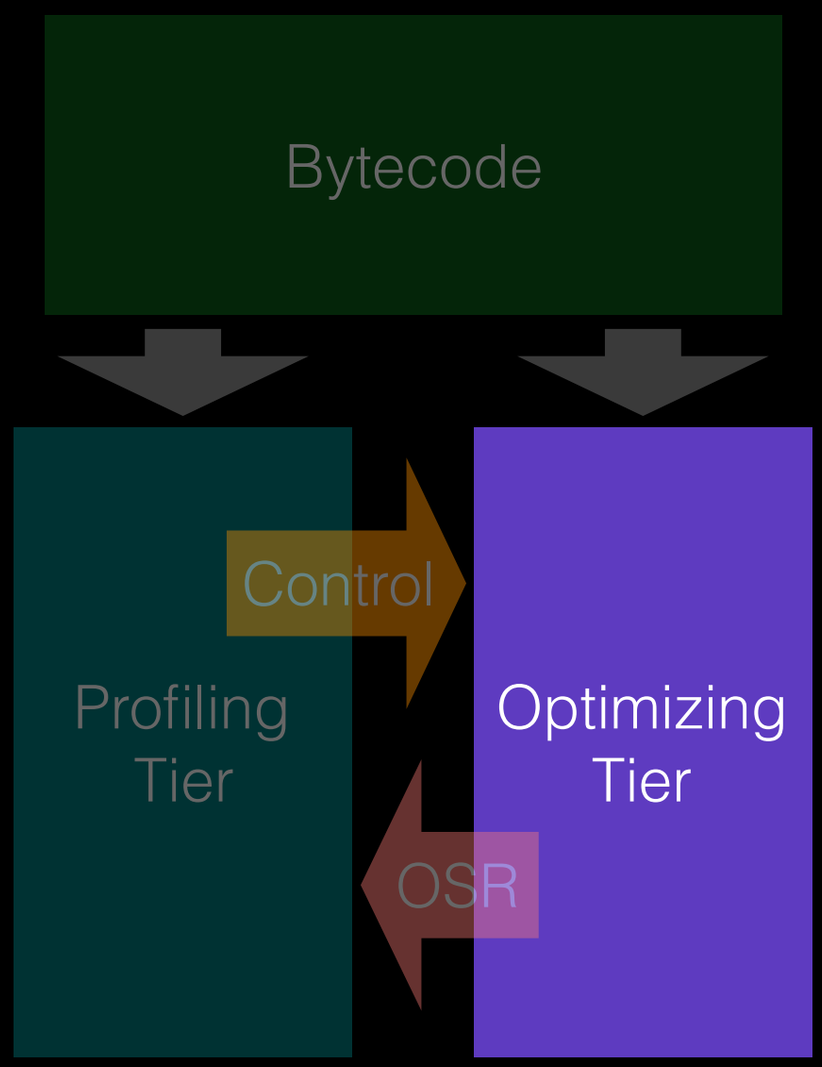
Speculation



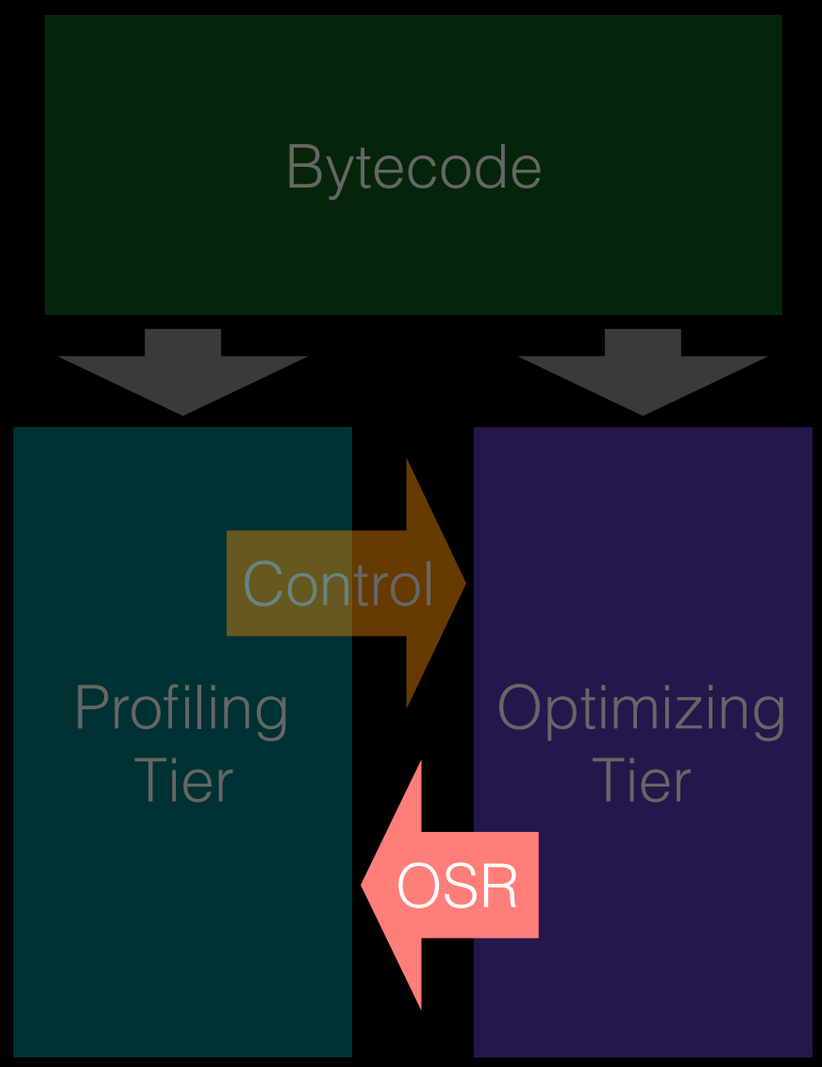
Speculation



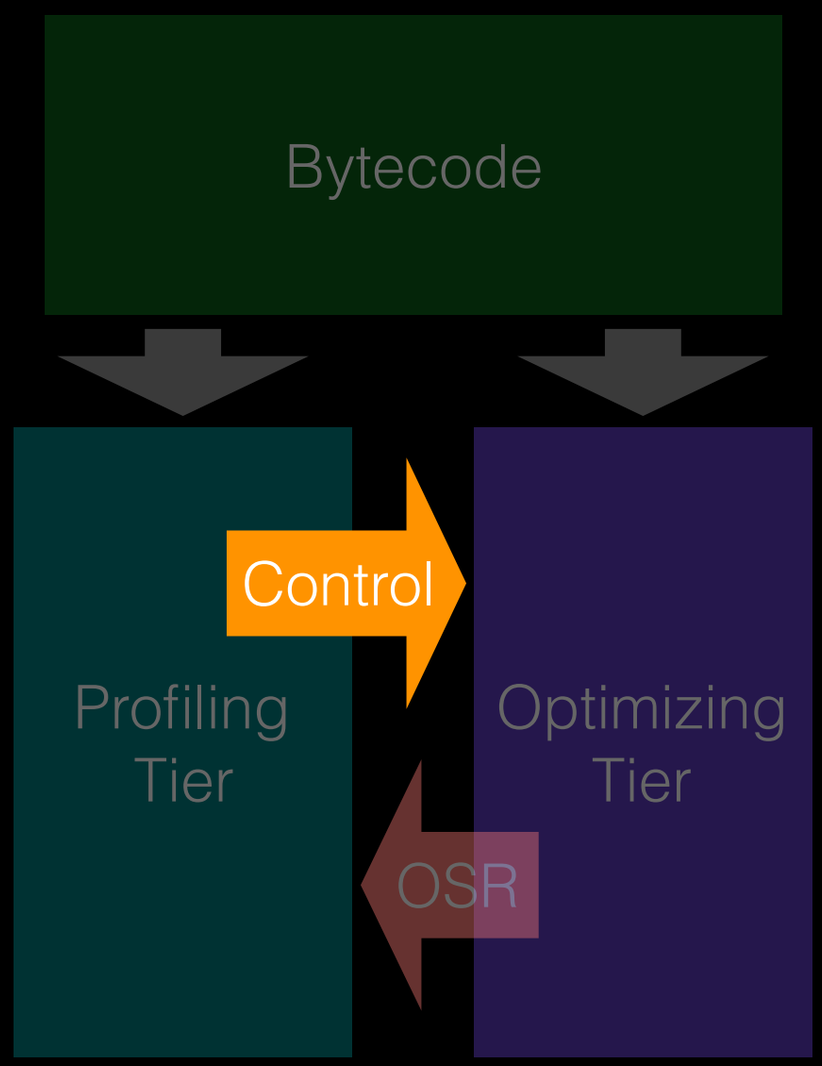
Speculation



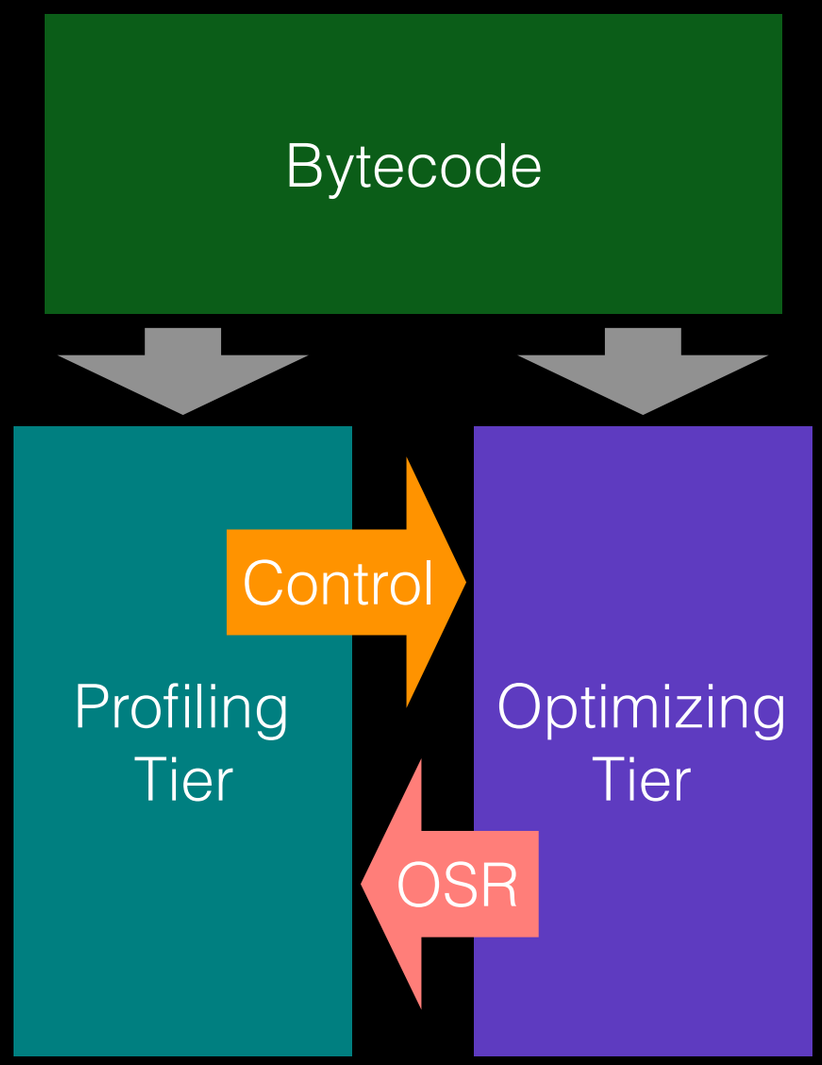
Speculation



Speculation



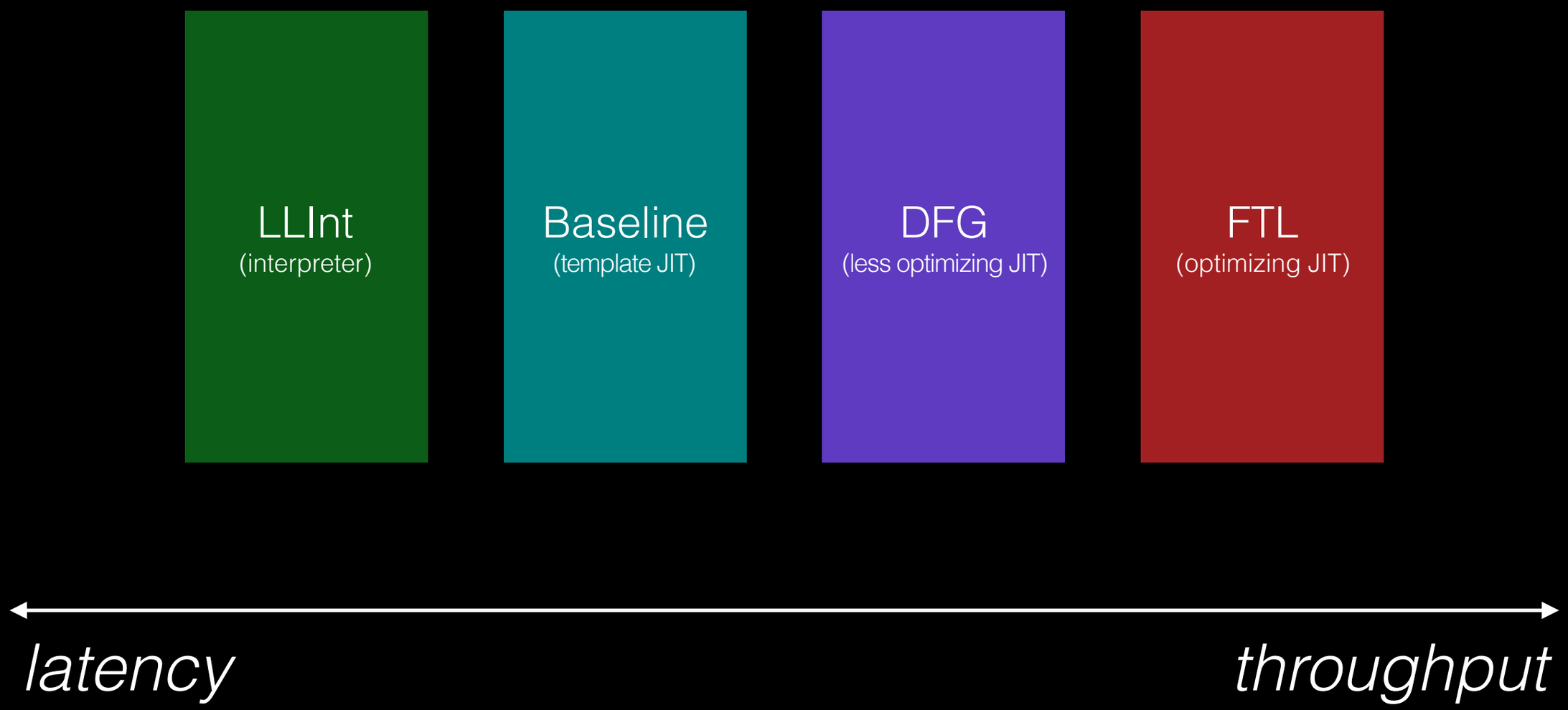
Speculation



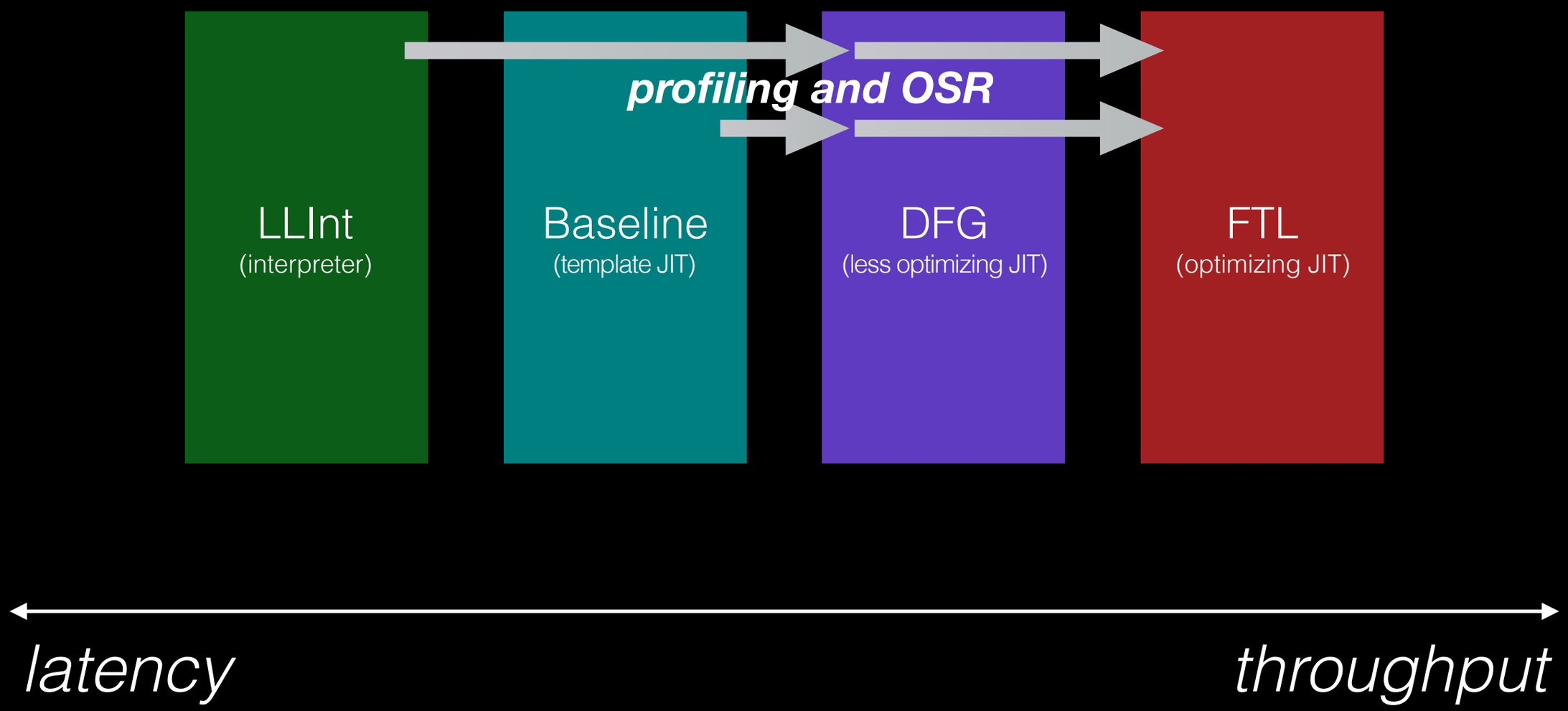
Agenda

- Speculation Overview
- JavaScriptCore Overview
- Speculation
  - Bytecode (Common IR)
  - Control
  - Profiling
  - Compilation
  - OSR (On Stack Replacement)

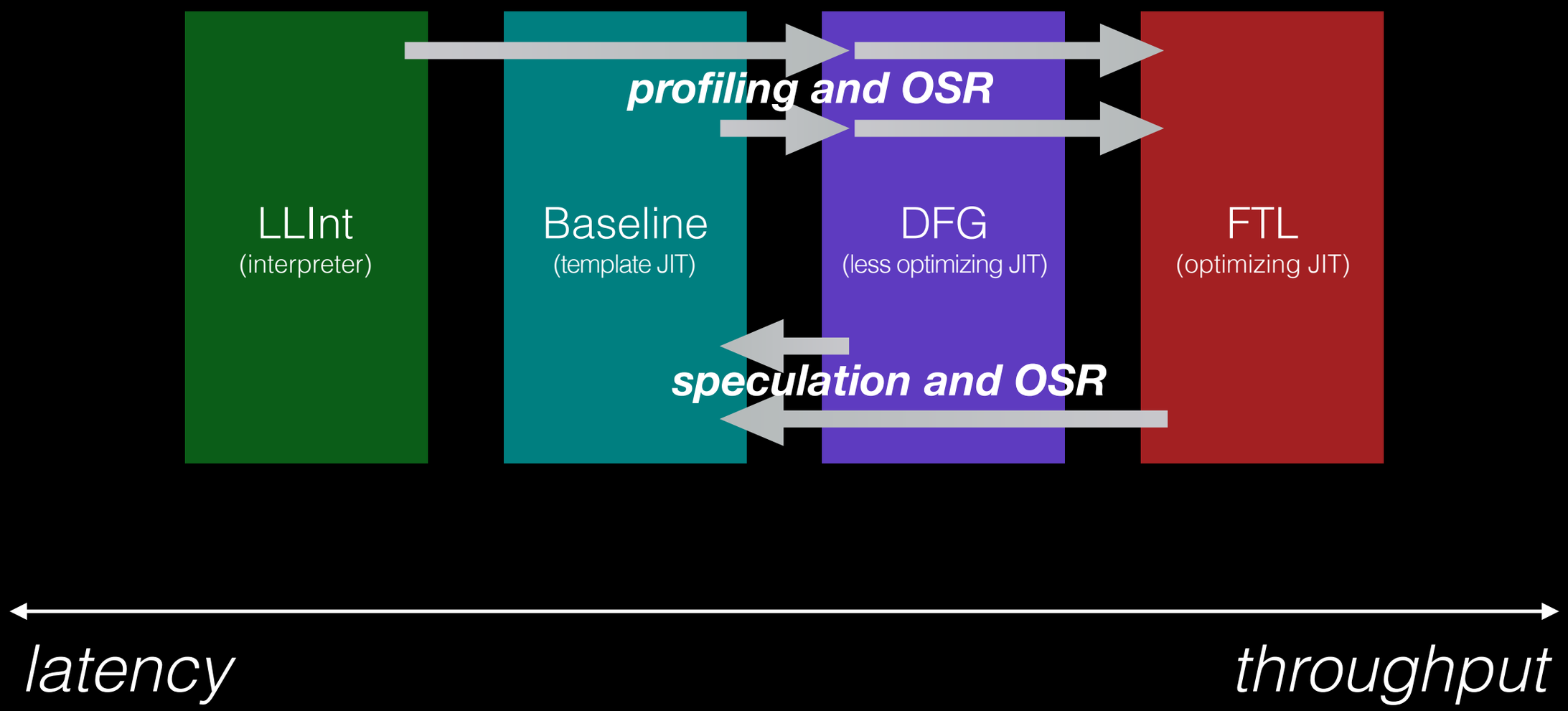
JSC's Four Tiers



JSC's Four Tiers



JSC's Four Tiers



```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```

```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```

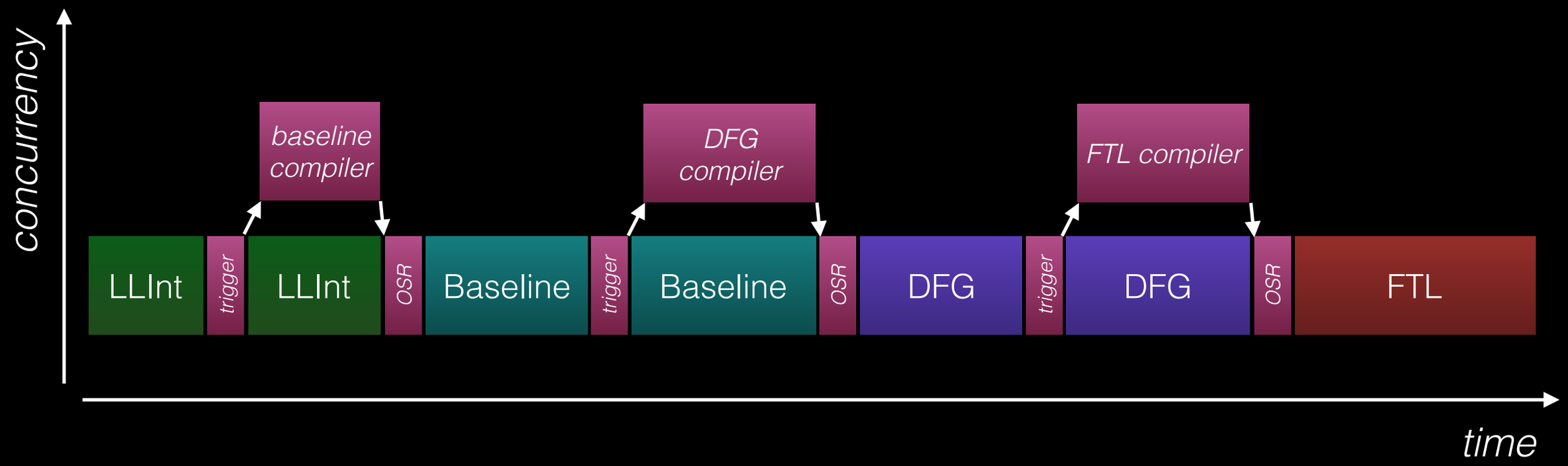
```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```

```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```

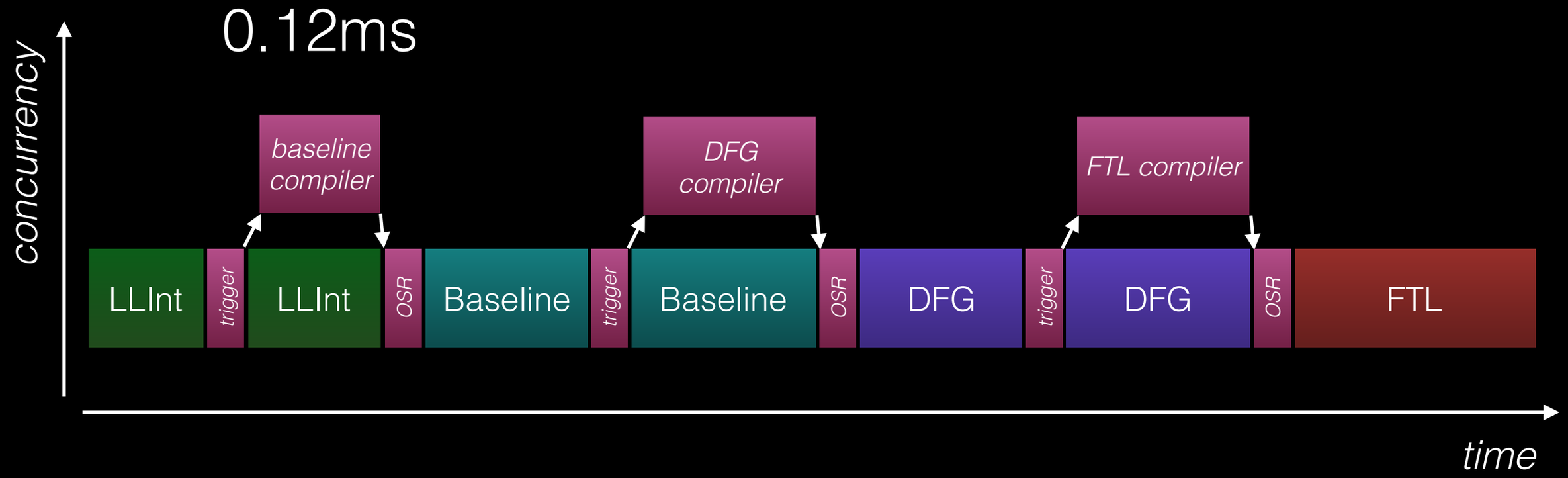
```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```

```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```

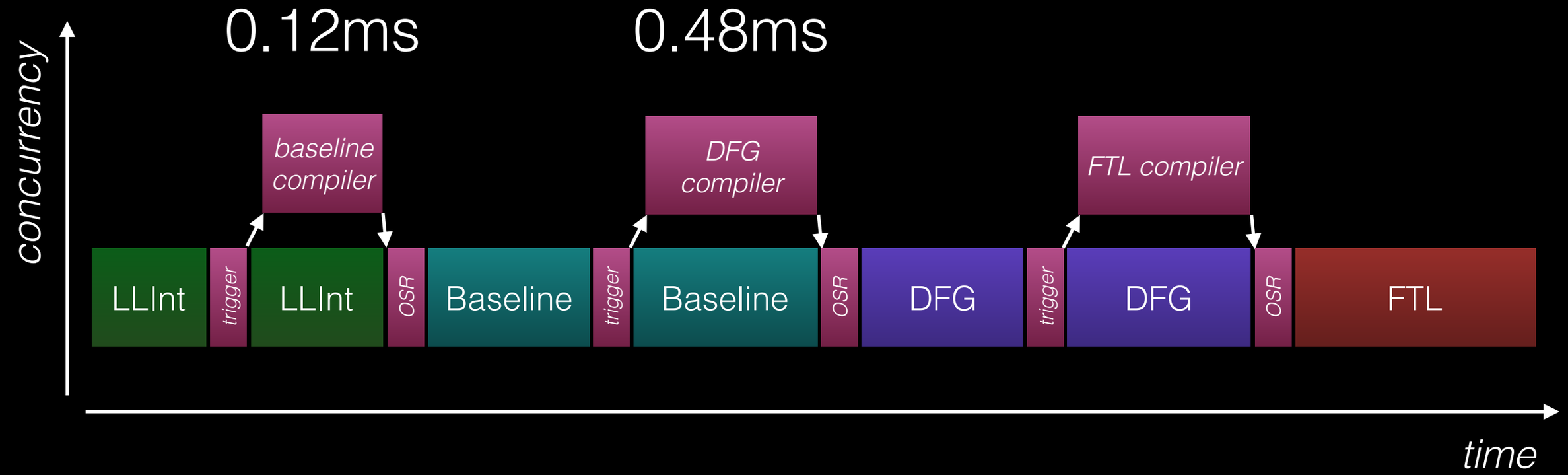
```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```



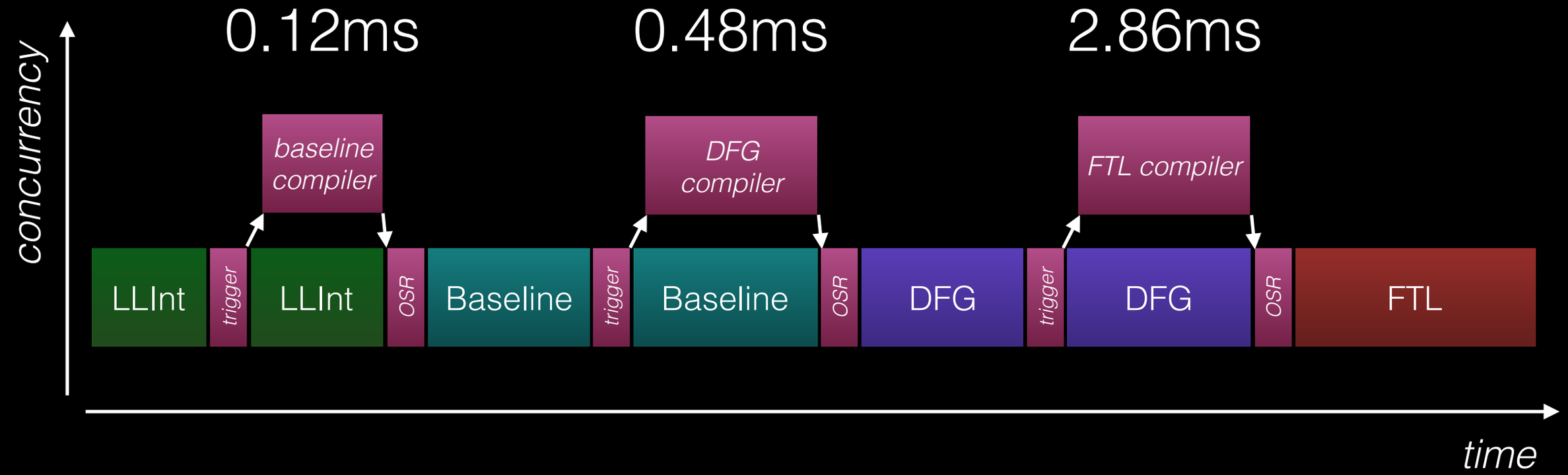
```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```



```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```



```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```



Parser

Parser

Bytecompiler

Parser

Bytecompiler

Generatorification

Parser

Bytecompiler

Generatorification

Bytecode Linker

Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt

Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT

Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT

DFG

Parser

Bytecompiler

Generatorification

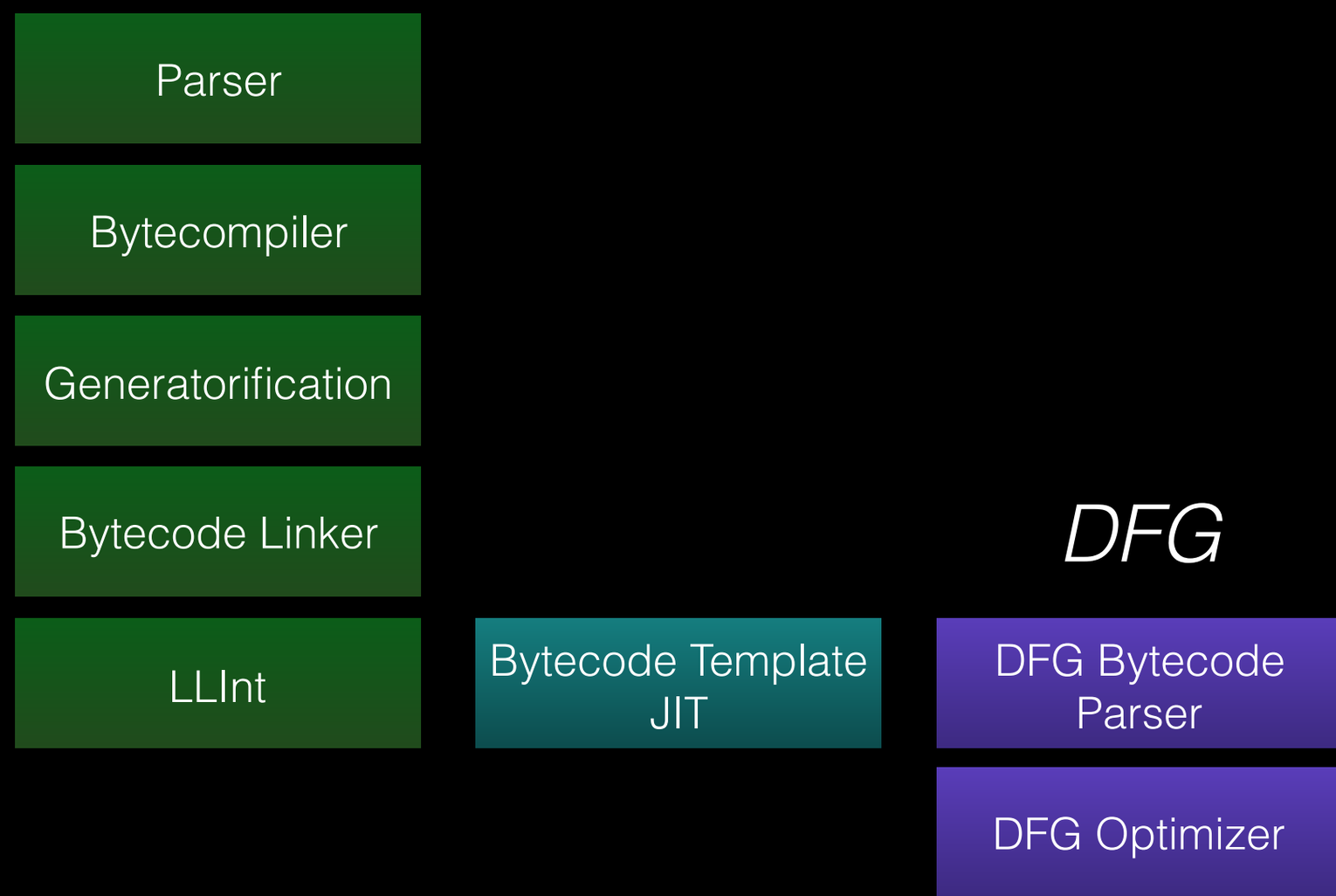
Bytecode Linker

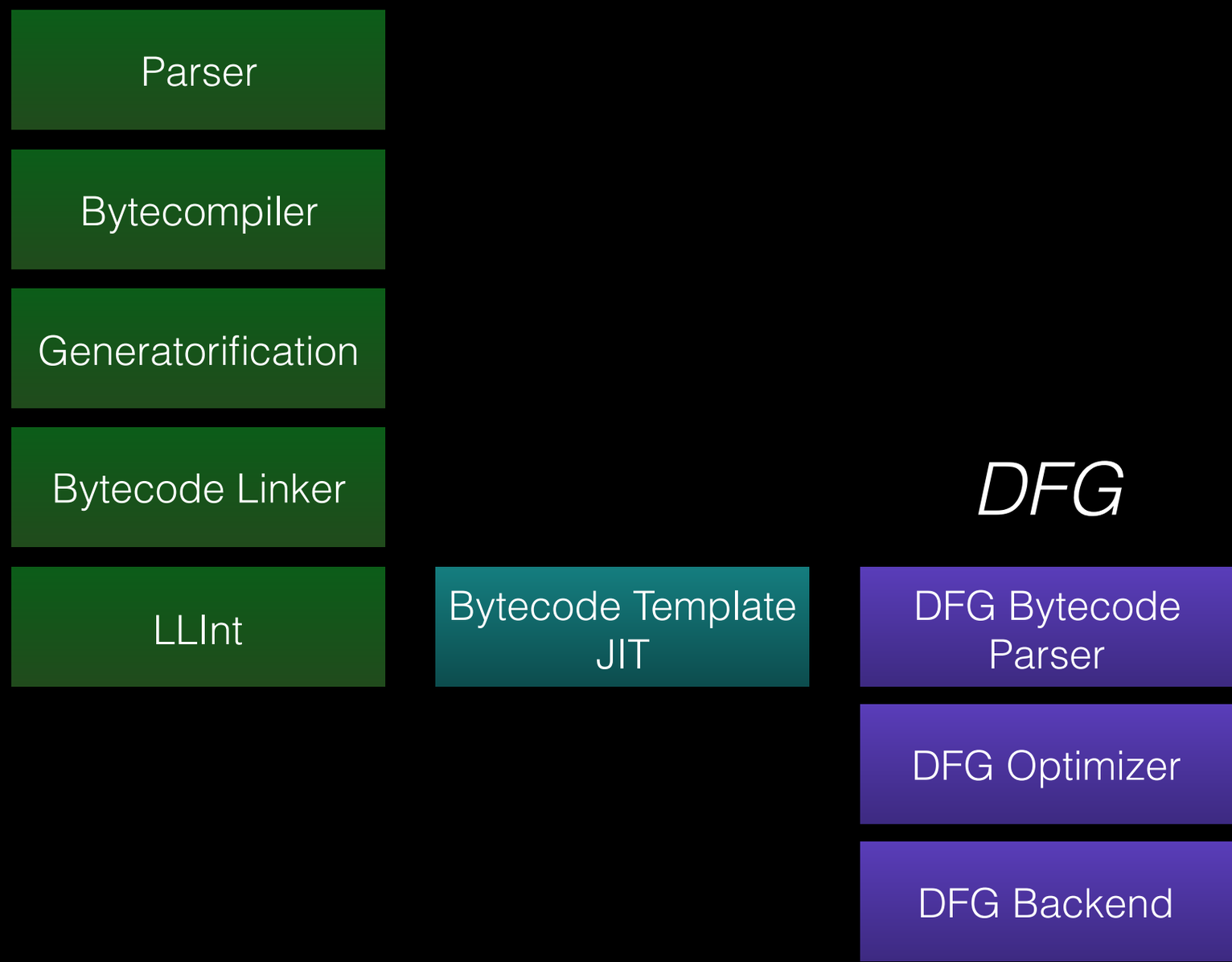
LLInt

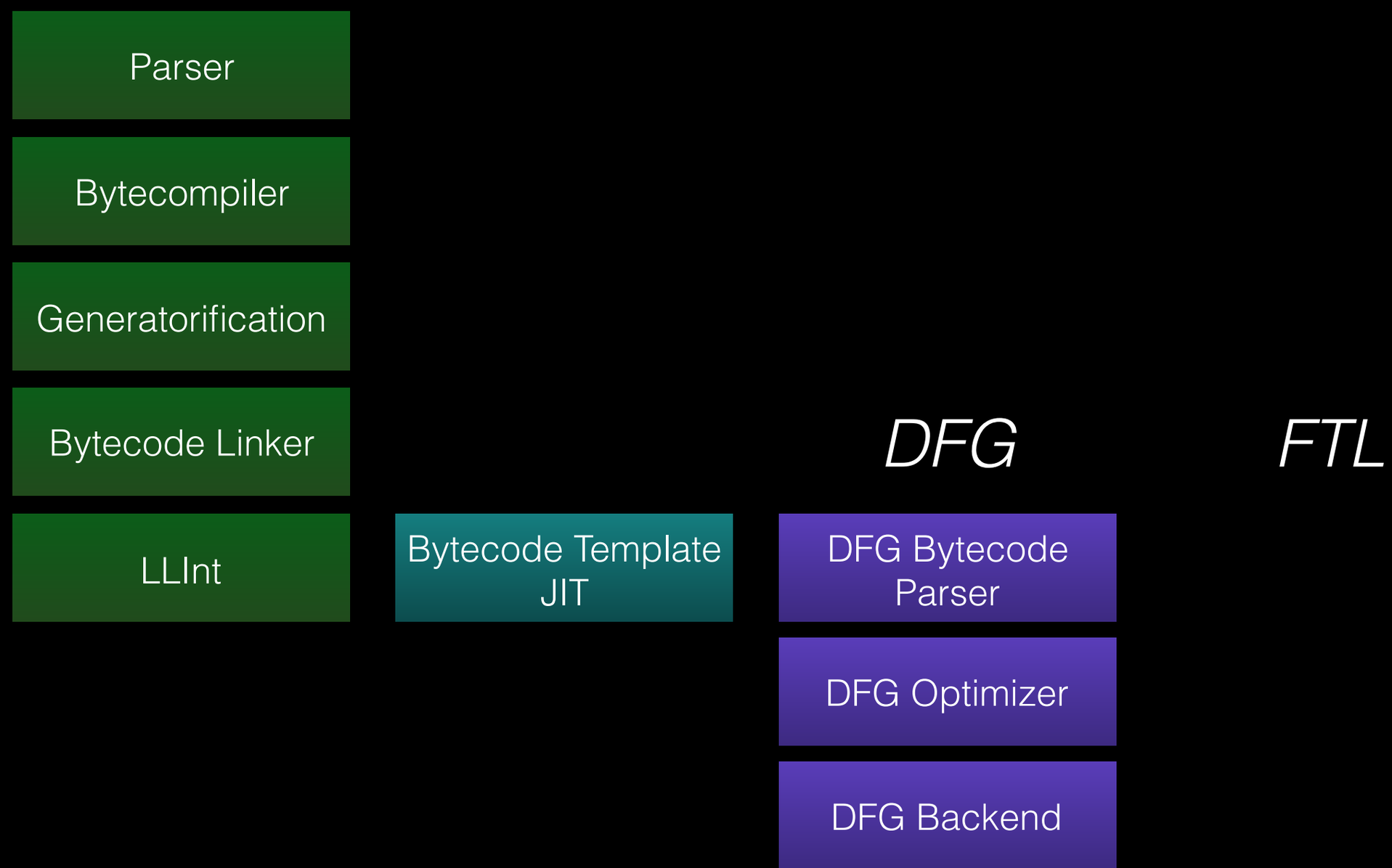
Bytecode Template JIT

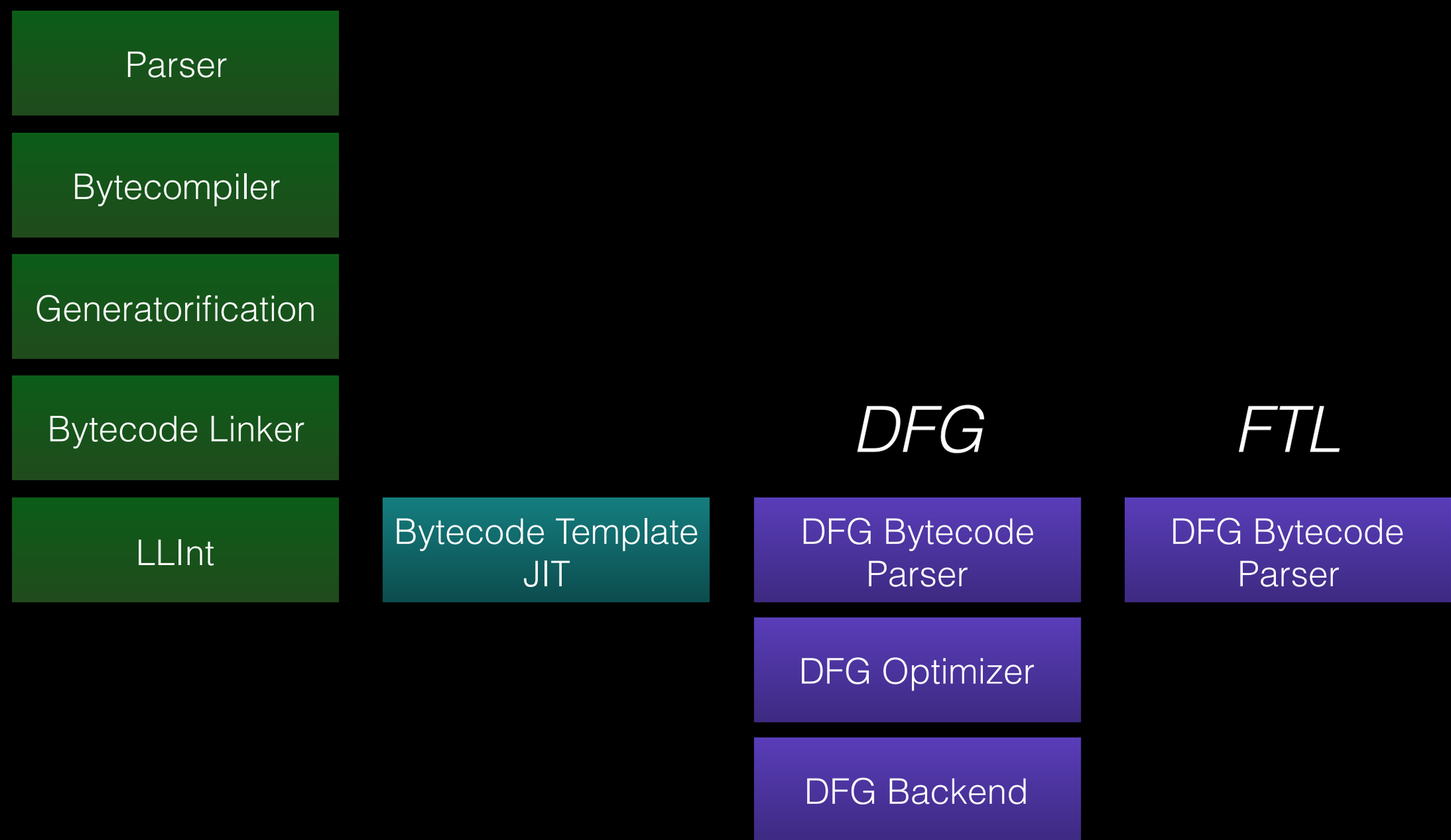
DFG

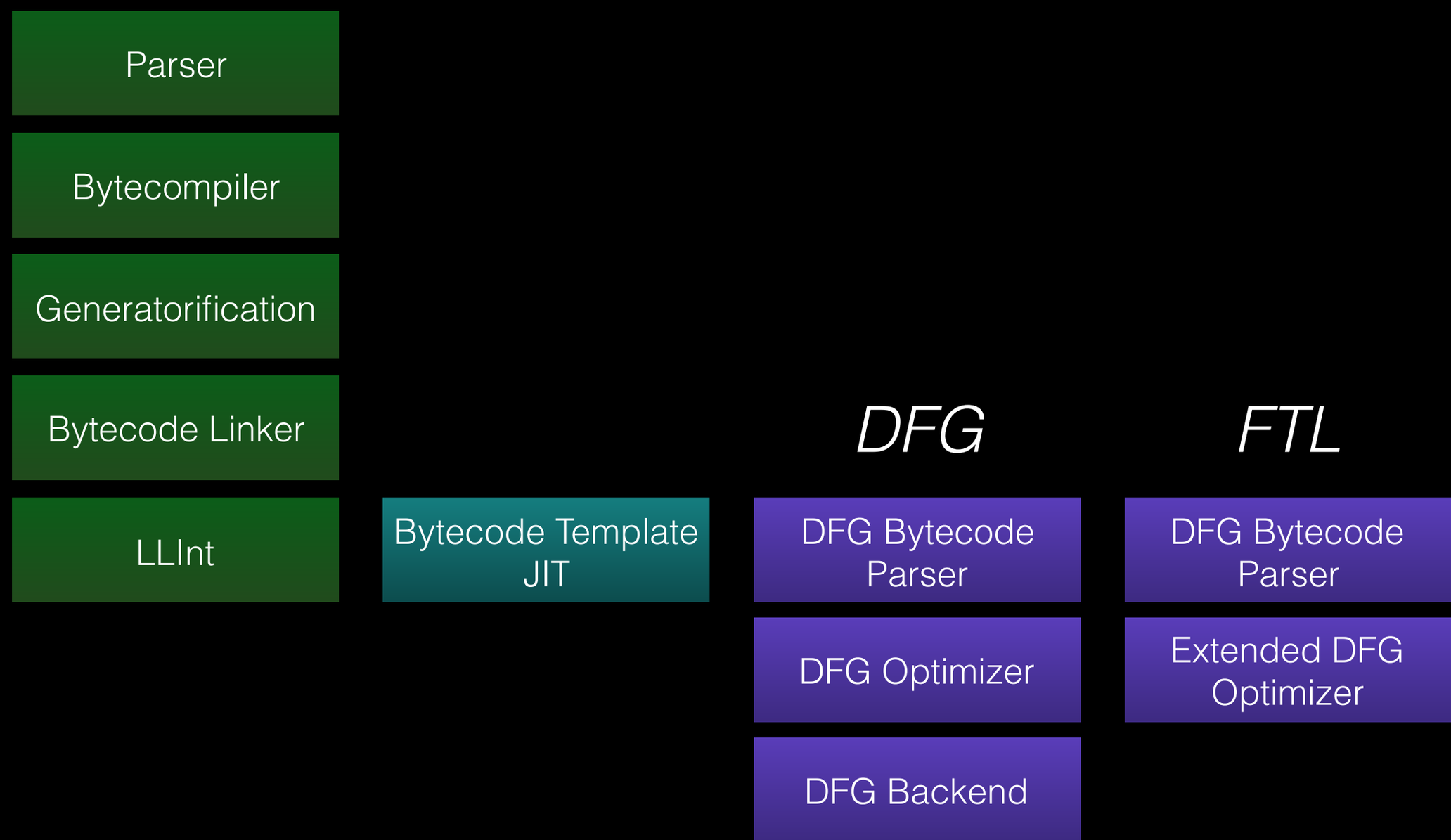
DFG Bytecode Parser

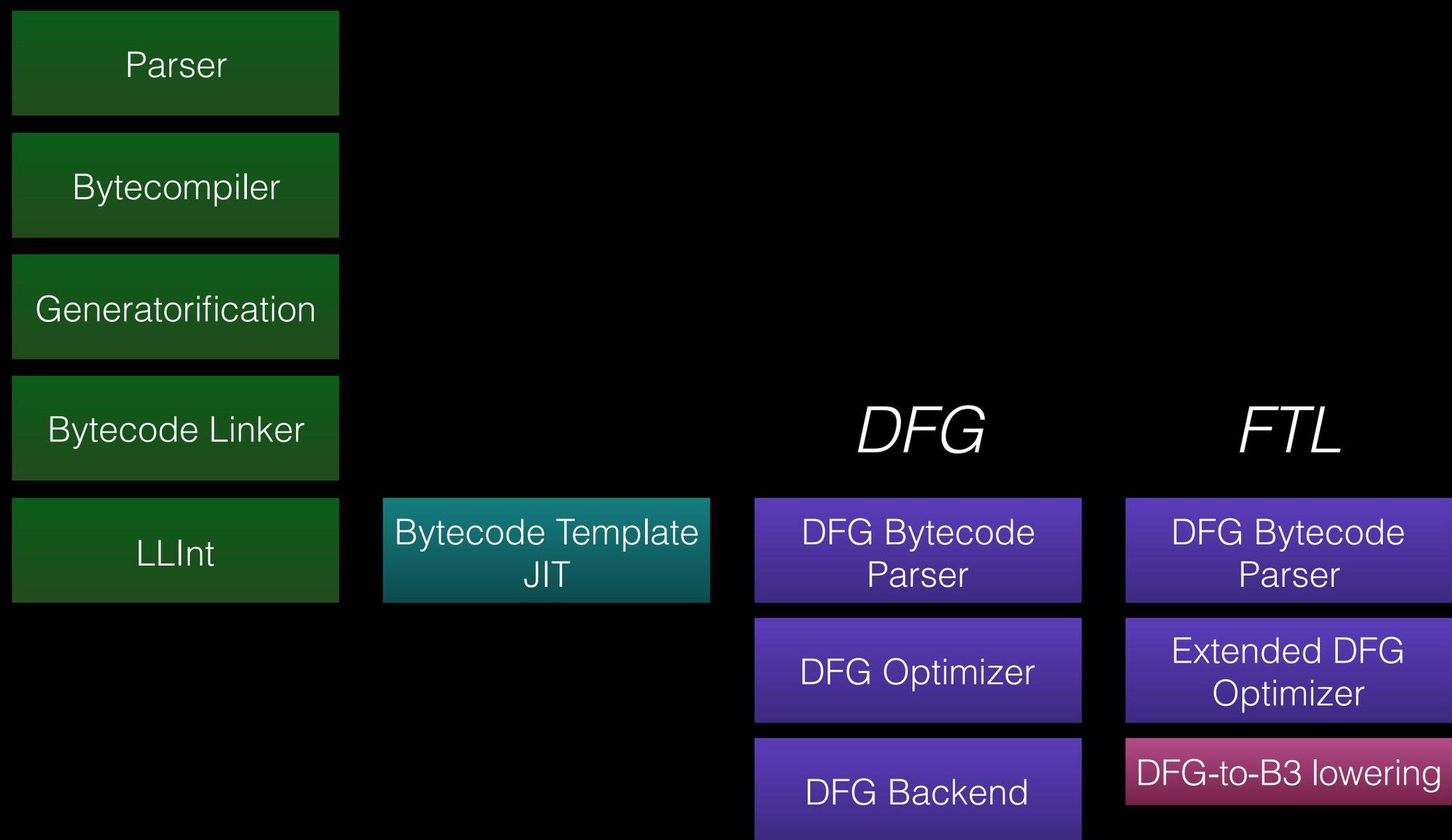

Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer

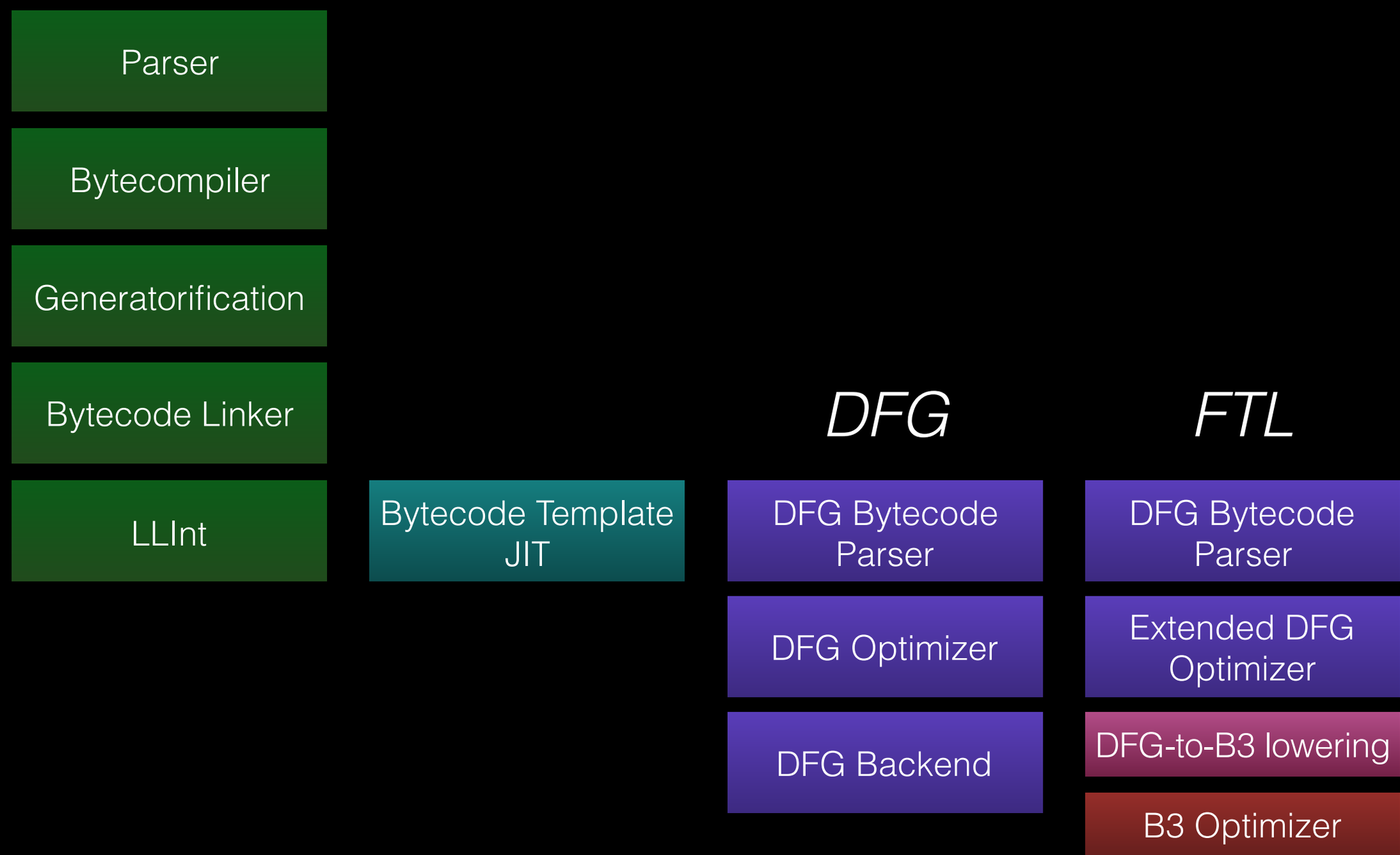

Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend

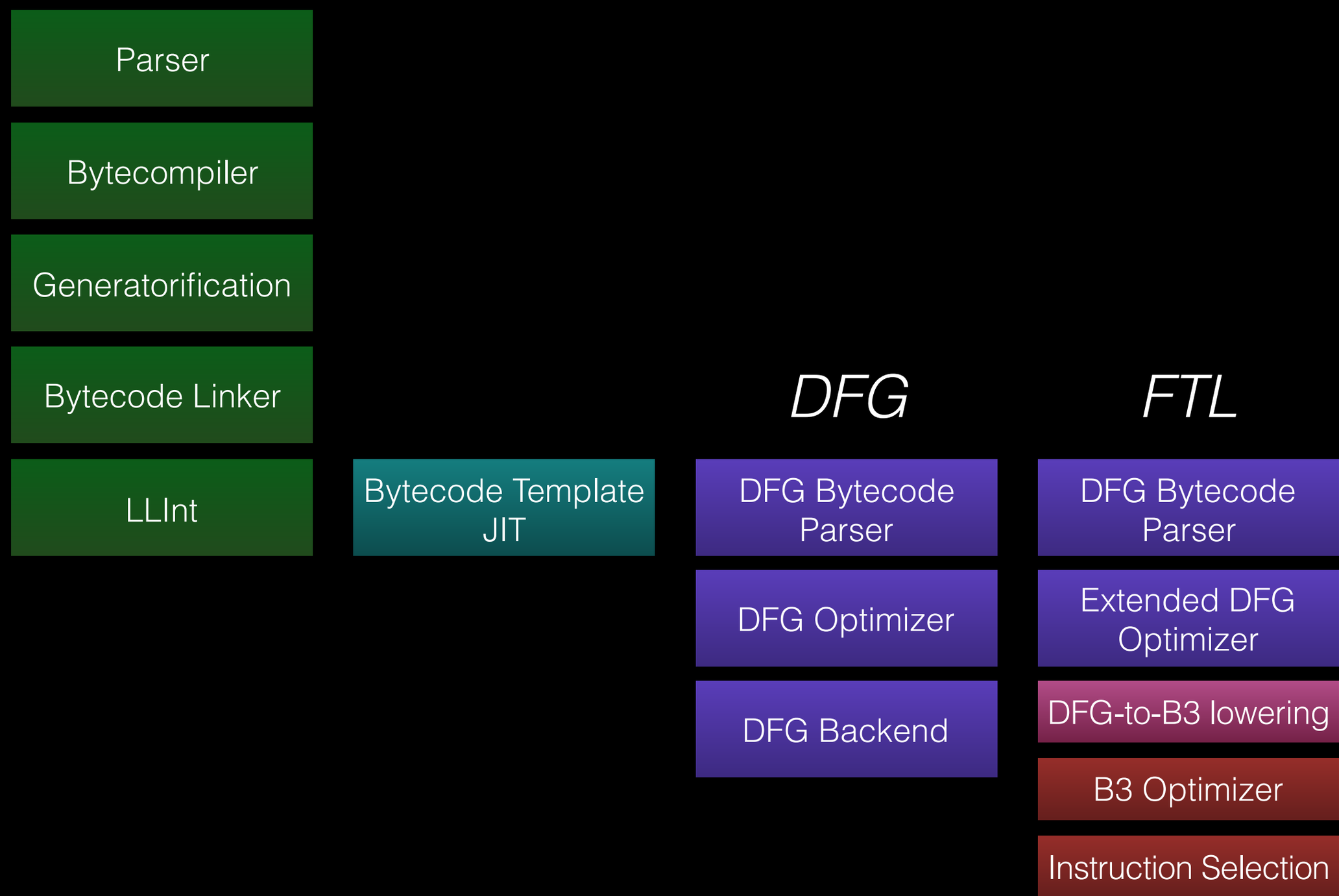

Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend
FTL

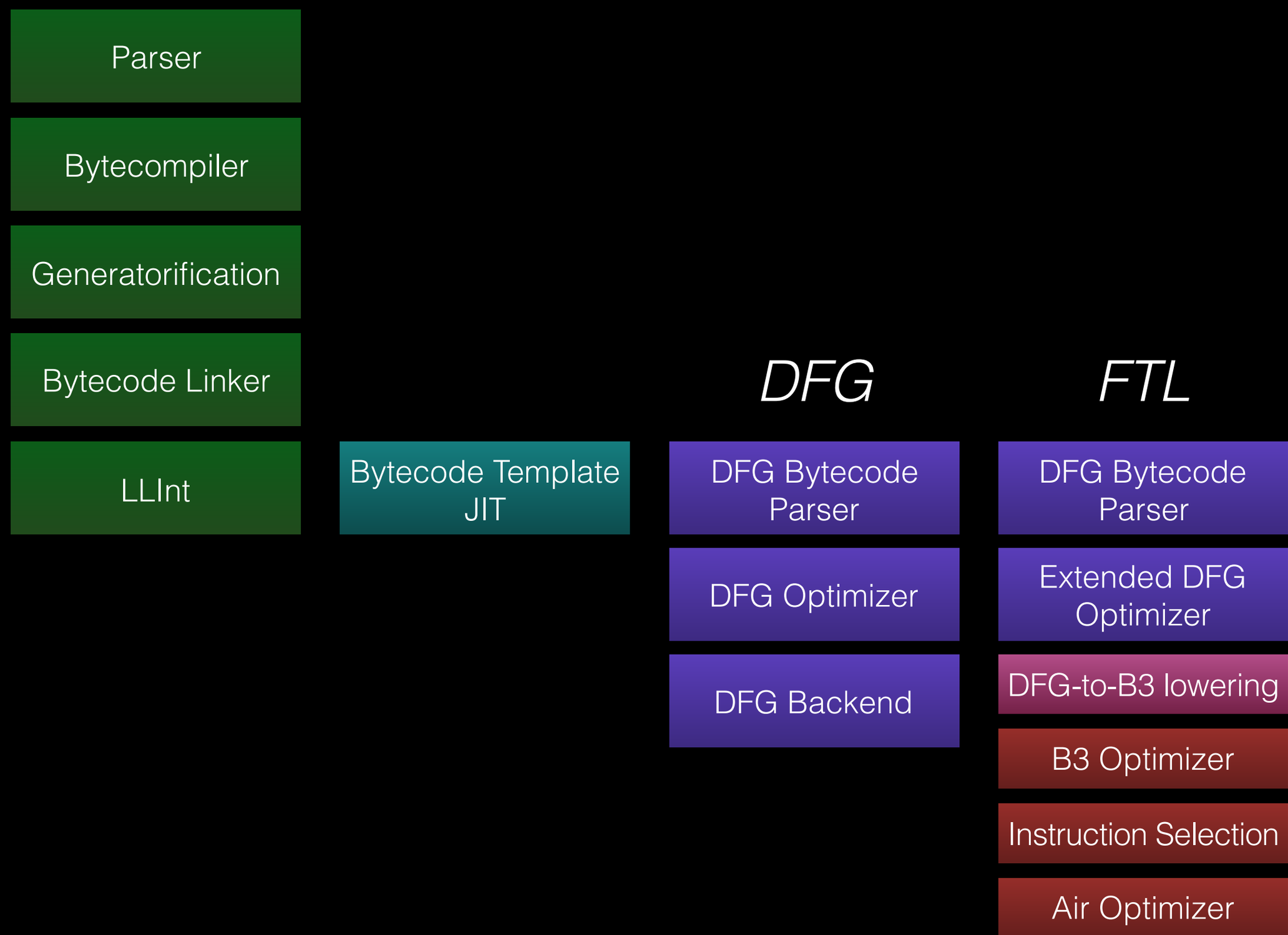

Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend
FTL
DFG Bytecode Parser

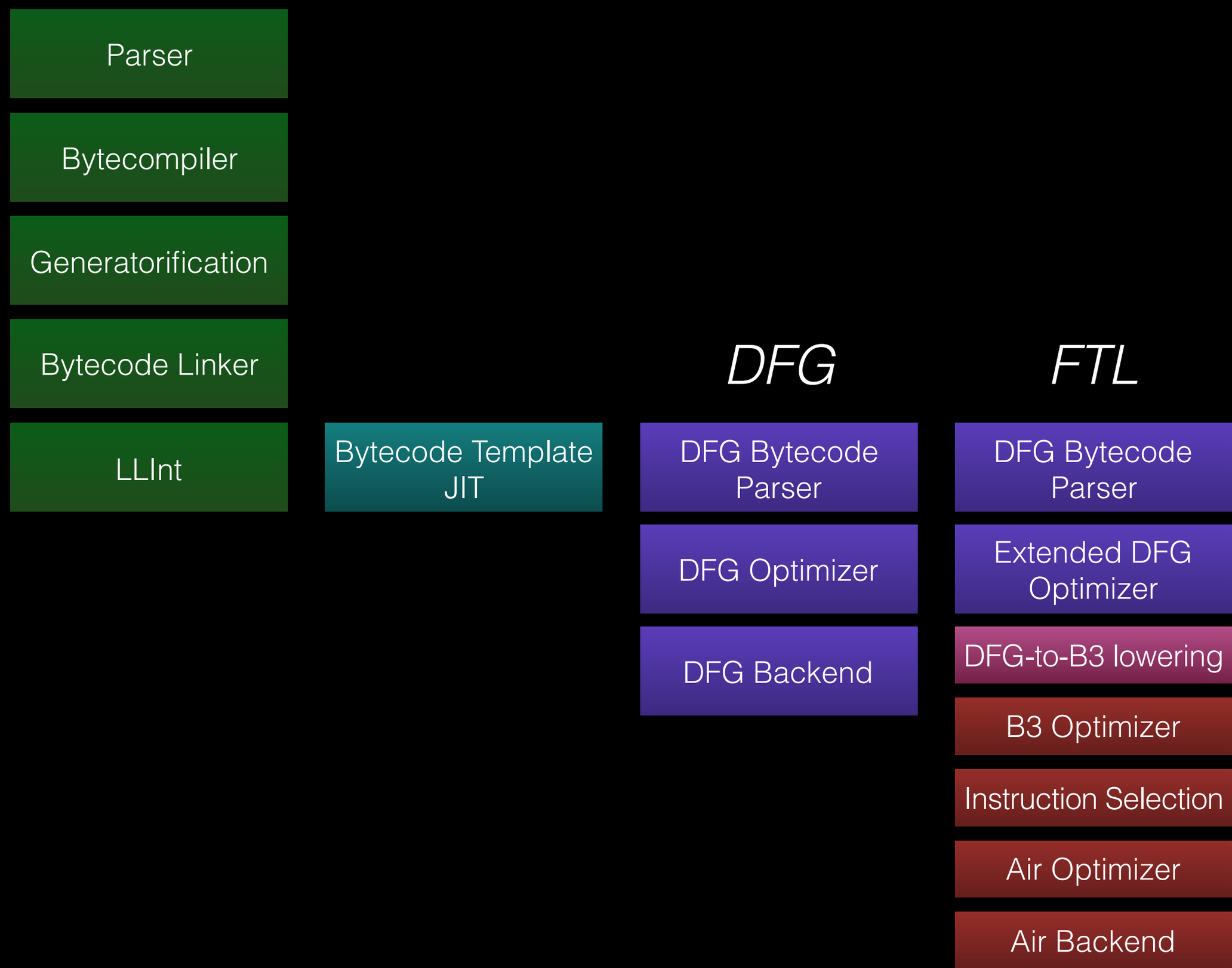

Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend
FTL
DFG Bytecode Parser
Extended DFG Optimizer


Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend
FTL
DFG Bytecode Parser
Extended DFG Optimizer
DFG-to-B3 lowering


Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend
FTL
DFG Bytecode Parser
Extended DFG Optimizer
DFG-to-B3 lowering
B3 Optimizer

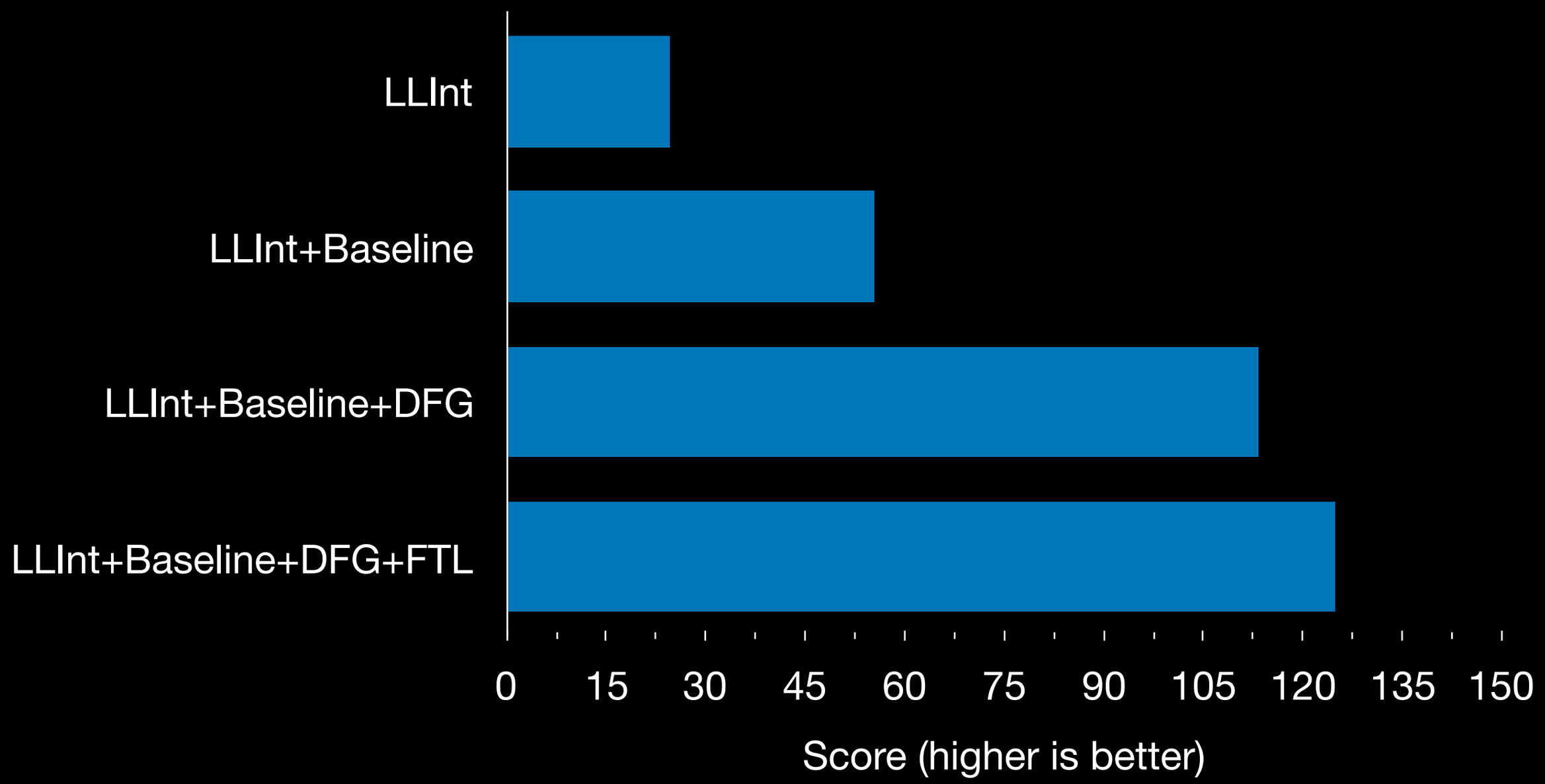

Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend
FTL
DFG Bytecode Parser
Extended DFG Optimizer
DFG-to-B3 lowering
B3 Optimizer
Instruction Selection


Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend
FTL
DFG Bytecode Parser
Extended DFG Optimizer
DFG-to-B3 lowering
B3 Optimizer
Instruction Selection
Air Optimizer


Parser
Bytecompiler
Generatorification
Bytecode Linker
LLInt
Bytecode Template JIT
DFG
DFG Bytecode Parser
DFG Optimizer
DFG Backend
FTL
DFG Bytecode Parser
Extended DFG Optimizer
DFG-to-B3 lowering
B3 Optimizer
Instruction Selection
Air Optimizer
Air Backend

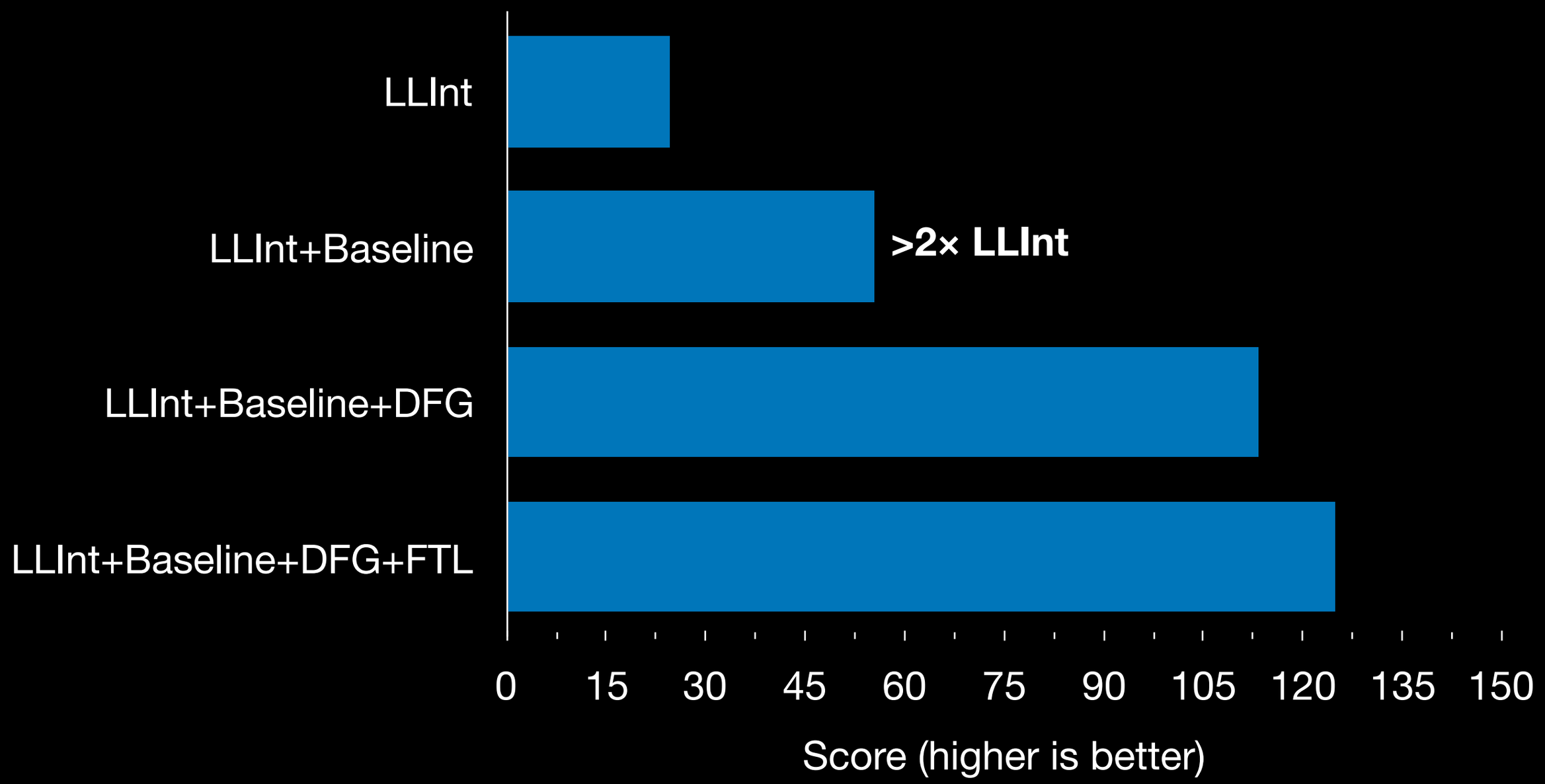
JetStream 2 Score

on my computer one day



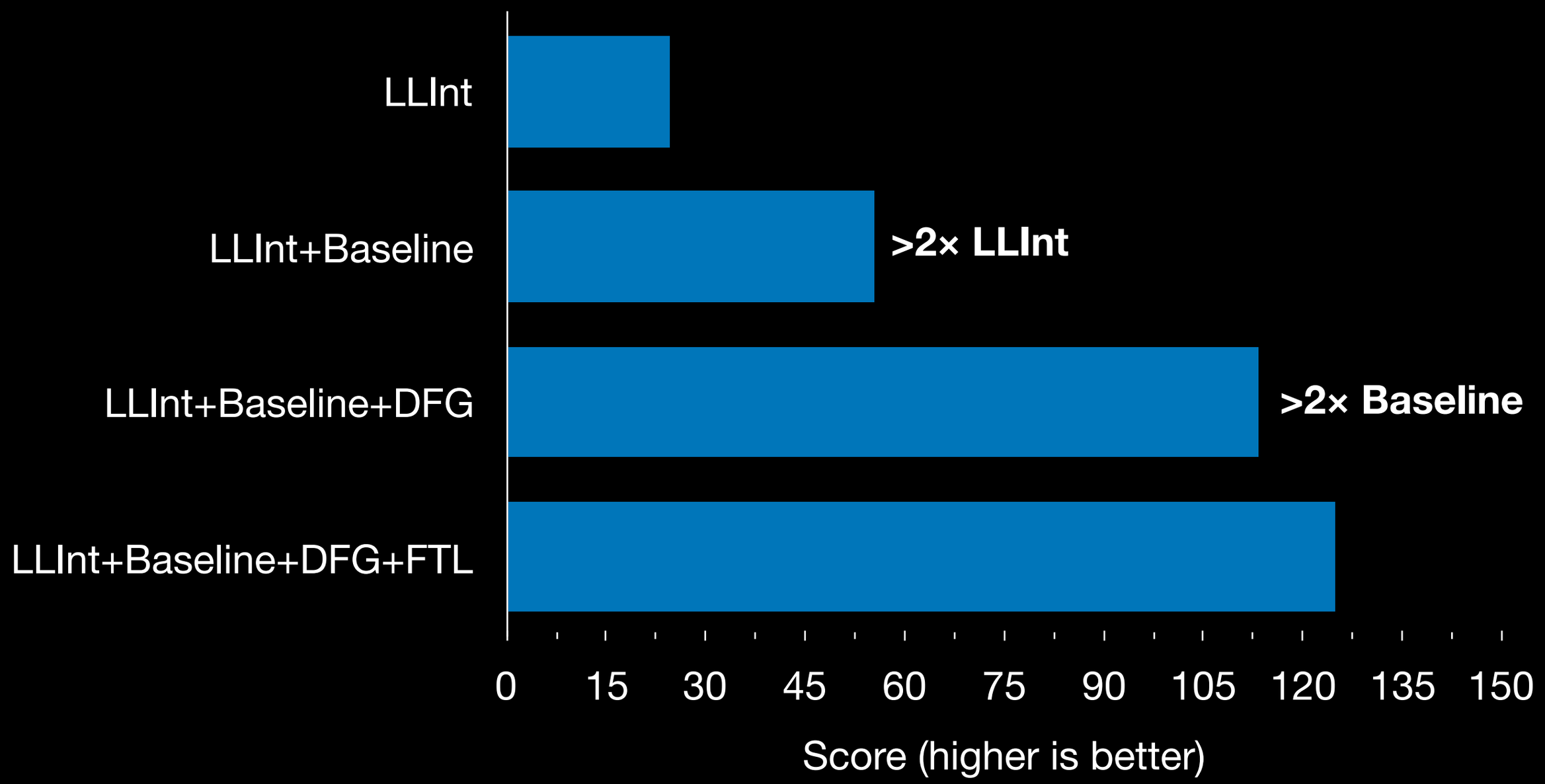
JetStream 2 Score

on my computer one day



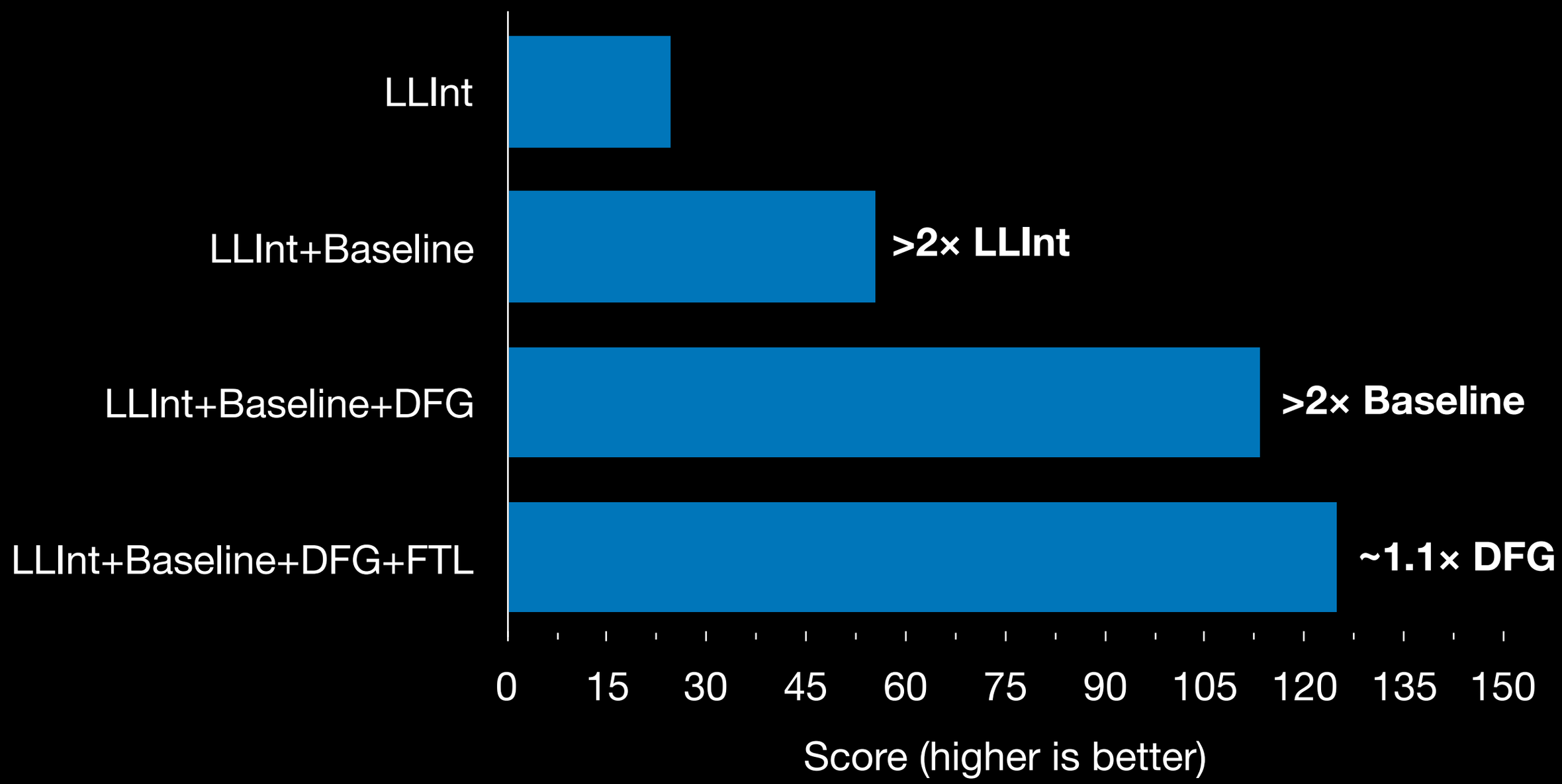
JetStream 2 Score

on my computer one day



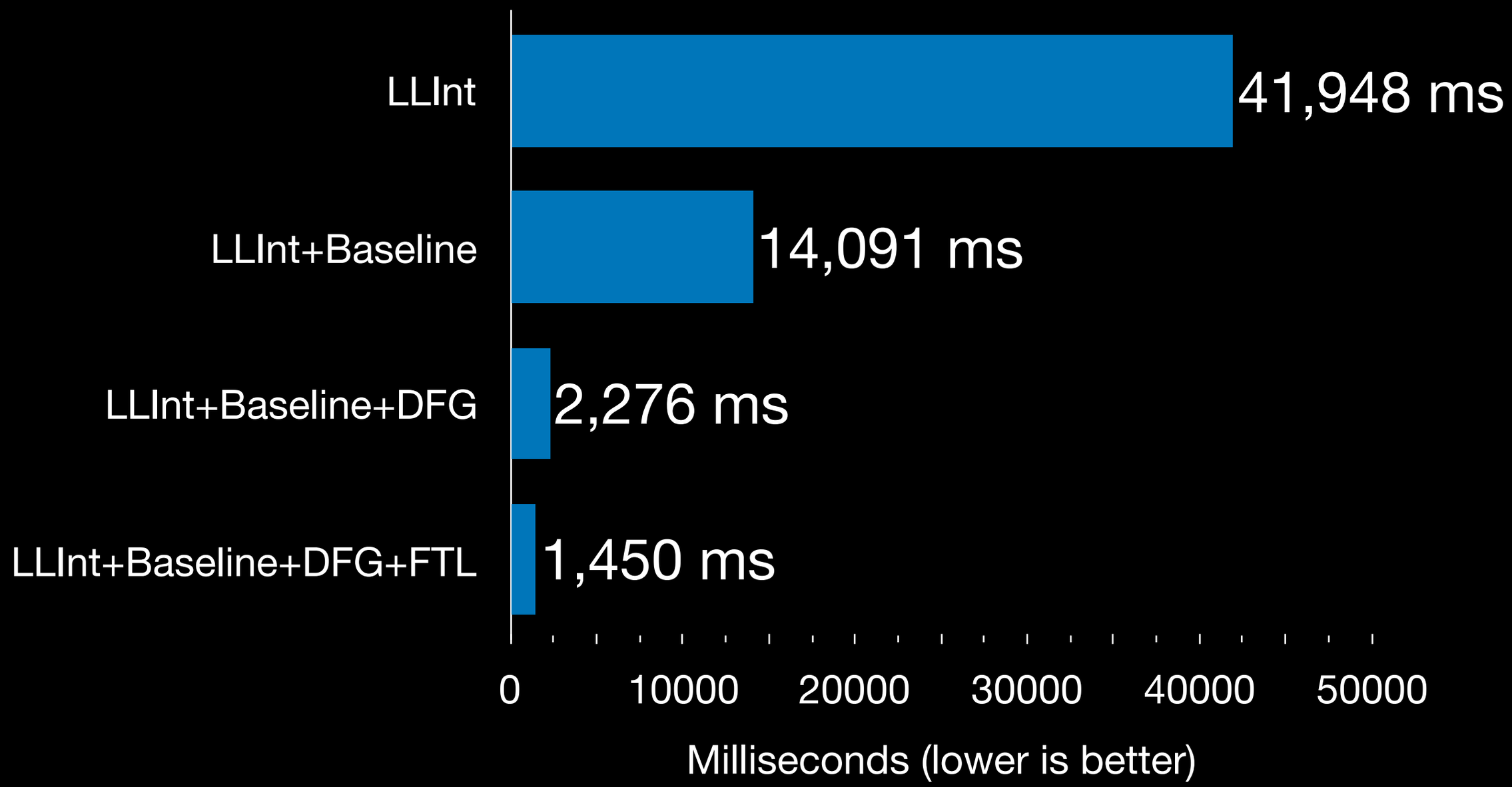
JetStream 2 Score

on my computer one day



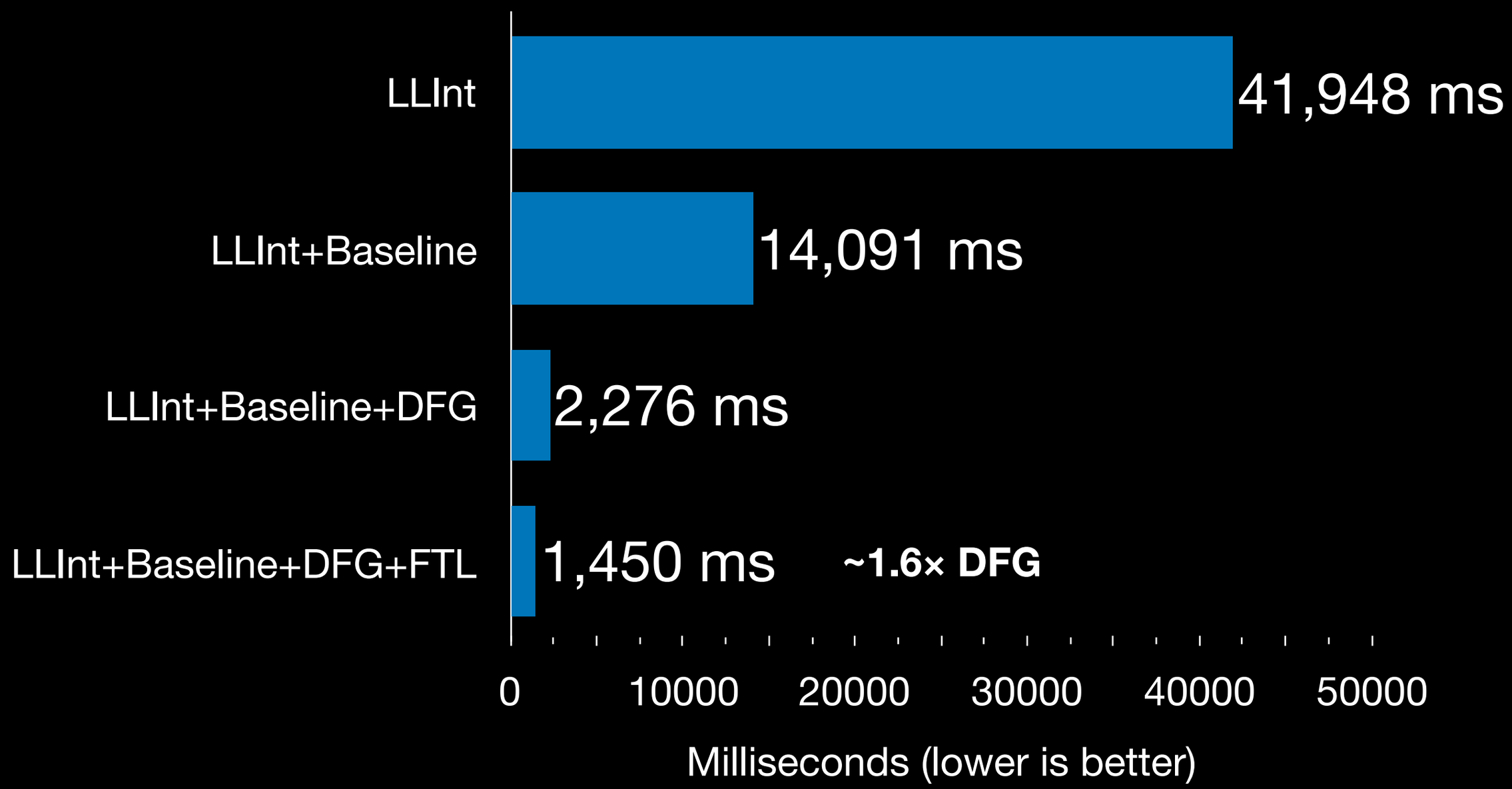
JetStream 2
“gaussian-blur”

on my computer one day



JetStream 2 “gaussian-blur”

on my computer one day



$$
\begin{aligned}
\text{Execution Time} = {} & (3.97\ \text{ns}) \times (\text{Bytecodes in LLInt}) \\
& + (1.71\ \text{ns}) \times (\text{Bytecodes in Baseline}) \\
& + (0.349\ \text{ns}) \times (\text{Bytecodes in DFG}) \\
& + (0.225\ \text{ns}) \times (\text{Bytecodes in FTL})
\end{aligned}
$$

Agenda

- Speculation Overview
- JavaScriptCore Overview
- Speculation
  - Bytecode (Common IR)
  - Control
  - Profiling
  - Compilation
  - OSR (On Stack Replacement)

Common IR

- Frame of reference for profiling
- Frame of reference for OSR

Bytecode

JSC Bytecode

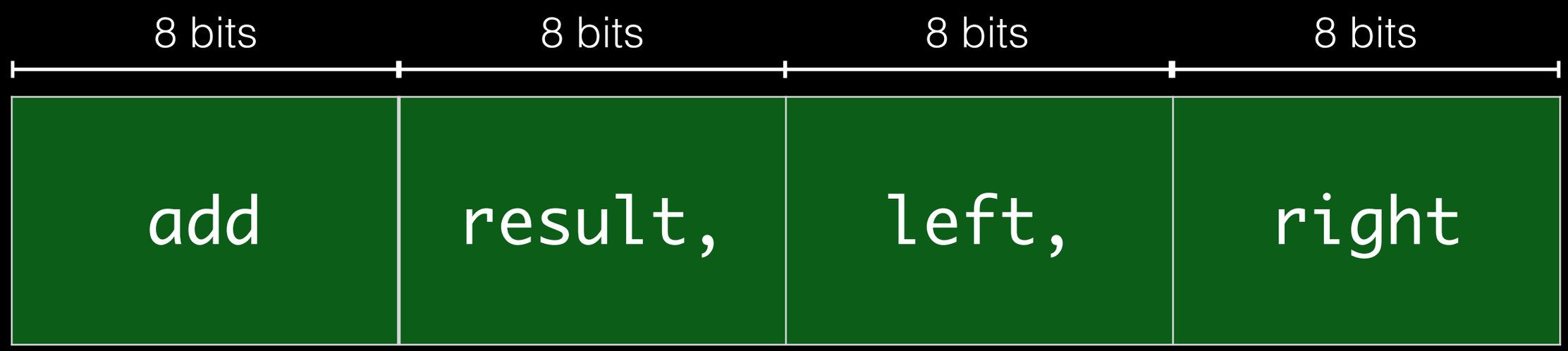
- Register-based
- Compact
- Untyped
- High-level
- Directly interpretable
- Transformable

Register-based

```
add     result,    left,     right
```

```
result = left + right
```

Compact




```
result = left + right
```

Untyped

add	result,	left,	right

result = left + right

High-level

add	result,	left,	right

result = left + right

Directly Interpretable

add	result,	left,	right

```
result = left + right
```

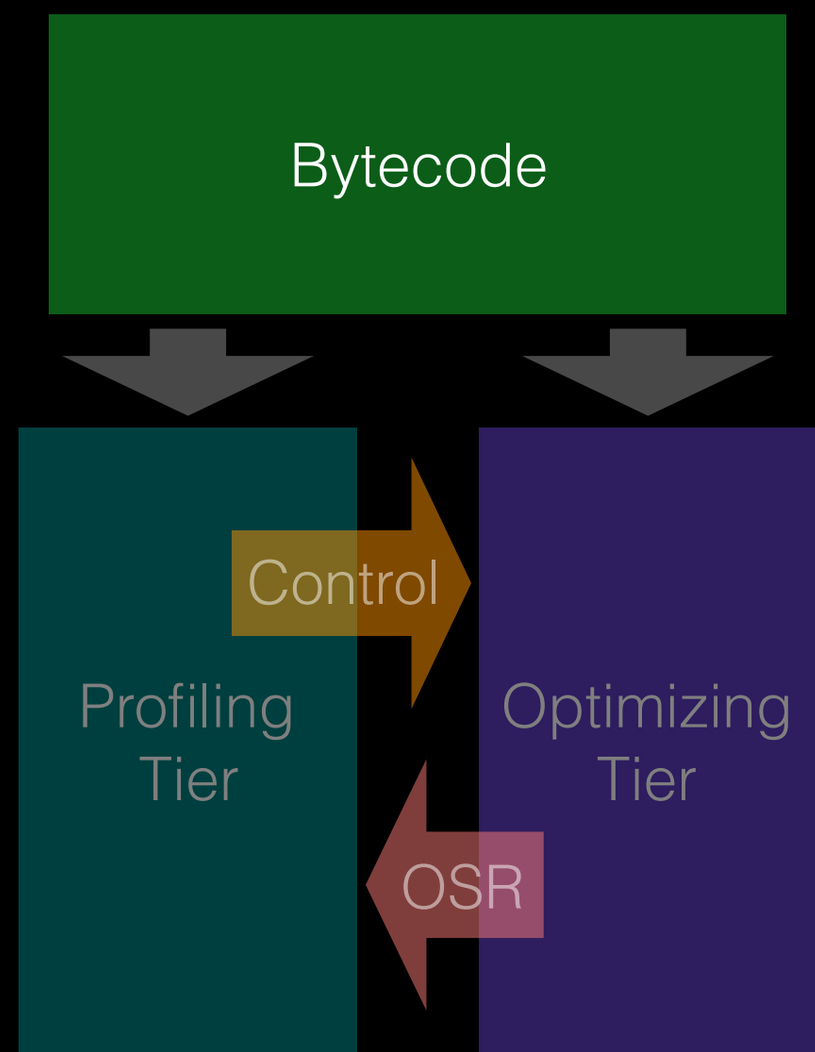
Transformable

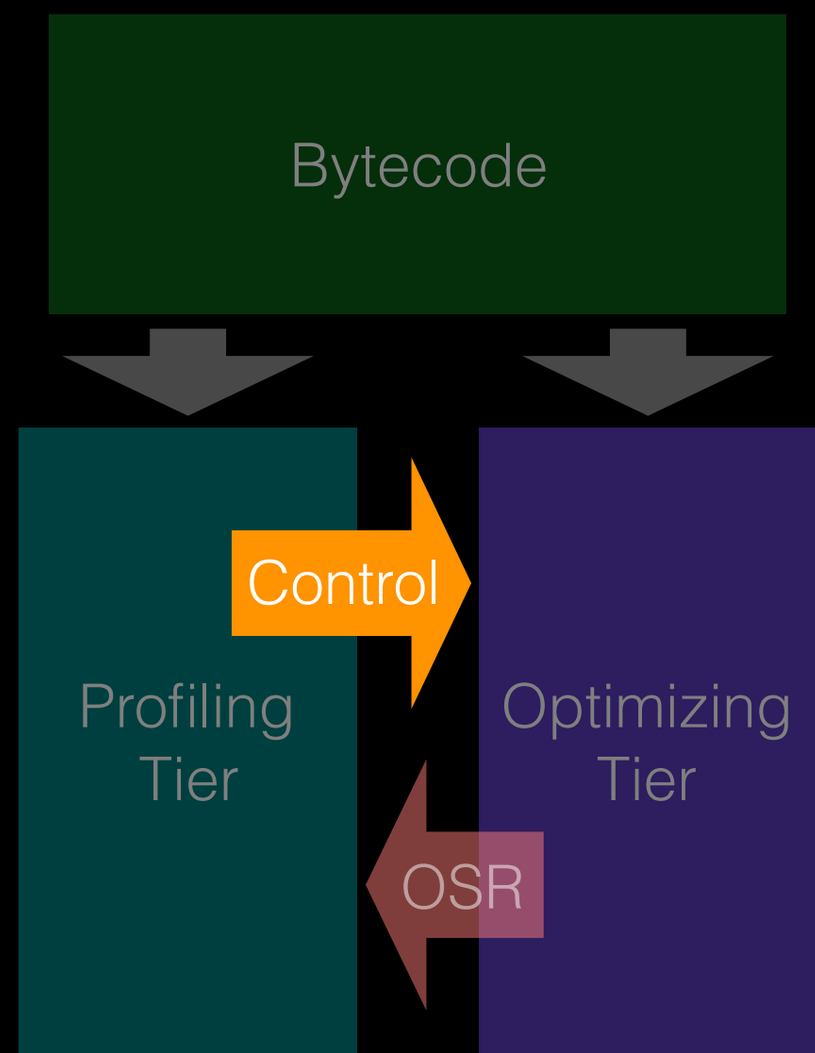
add	result,	left,	right

result = left + right

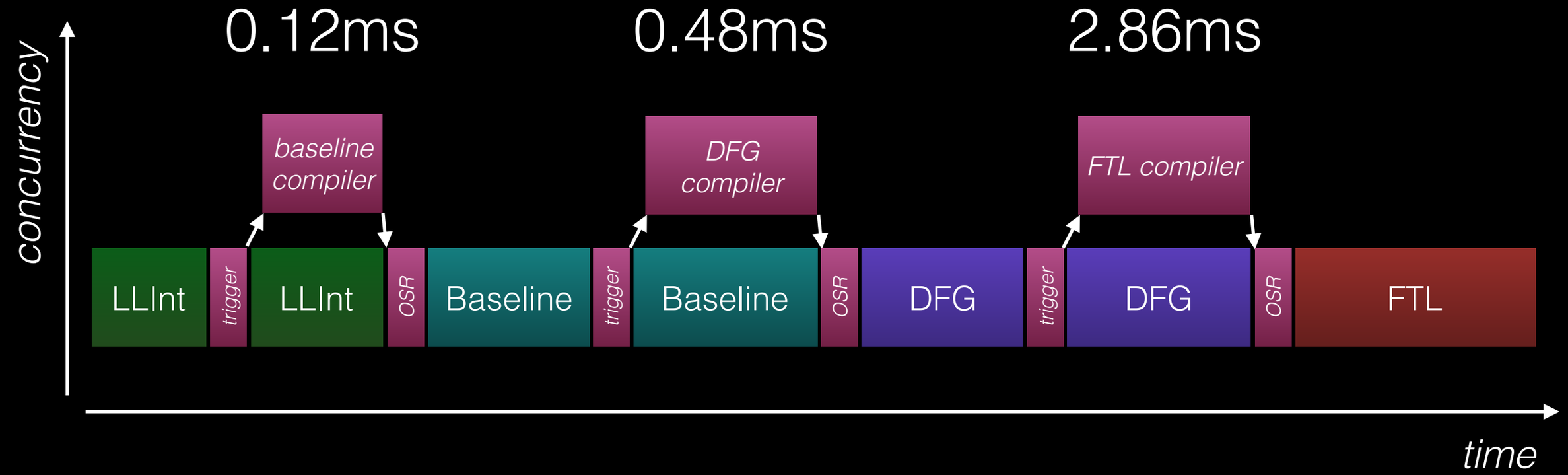
JSC Bytecode

- Register-based
- Compact
- Untyped
- High-level
- Directly interpretable
- Transformable


Bytecode
Control
Profiling Tier
Optimizing Tier
OSR


Bytecode
Control
Profiling Tier
Optimizing Tier
OSR

```
"use strict";

let result = 0;
for (let i = 0; i < 10000000; ++i) {
    let o = {f: i};
    result += o.f;
}

print(result);
```



Control

- Execution Counting
- Exit Counting
- Recompilation

Execution Counting

Case	Execution Count Increment Amount
Function Call	15
Loop Back Edge	1

Execution Count Thresholds for Tier-up

Tier-up Case	Required Count for Tier-up
LLInt → Baseline	500
Baseline → DFG	1000
DFG → FTL	100000

Execution Count Thresholds for Tier-up

Tier-up Case	Required Count for Tier-up			
LLInt → Baseline	Was Optimized?	Don't Know	Yes	No
	Count	500	250	2000
Baseline → DFG	1000			
DFG → FTL	100000			

Execution Count Thresholds for Tier-up

Tier-up Case	Required Count for Tier-up			
LLInt → Baseline	Was Optimized?	Don't Know	Yes	No
	Count	500	250	2000
Baseline → DFG	$1000 \times (0.825914 + 0.061504 \sqrt{S + 1.02406}) \times 2^R \times M/(M-U)$			
DFG → FTL	100000			

Execution Count Thresholds for Tier-up

Tier-up Case	Required Count for Tier-up			
LLInt → Baseline	Was Optimized?	Don't Know	Yes	No
	Count	500	250	2000
Baseline → DFG	$1000 \times (0.825914 + 0.061504\sqrt{S + 1.02406}) \times 2^R \times M/(M-U)$			
DFG → FTL	100000			

Execution Count Thresholds for Tier-up

Tier-up Case	Required Count for Tier-up			
LLInt → Baseline	Was Optimized?	Don't Know	Yes	No
	Count	500	250	2000
Baseline → DFG	$1000 \times (0.825914 + 0.061504 \sqrt{S + 1.02406}) \times 2^R \times M/(M-U)$			
DFG → FTL	100000			

Execution Count Thresholds for Tier-up

Tier-up Case	Required Count for Tier-up			
LLInt → Baseline	Was Optimized?	Don't Know	Yes	No
	Count	500	250	2000
Baseline → DFG	$1000 \times (0.825914 + 0.061504 \sqrt{S + 1.02406}) \times 2^R \times M/(M-U)$			
DFG → FTL	100000			

Execution Count Thresholds for Tier-up

Tier-up Case	Required Count for Tier-up			
LLInt → Baseline	Was Optimized?	Don't Know	Yes	No
	Count	500	250	2000
Baseline → DFG	$1000 \times (0.825914 + 0.061504\sqrt{S + 1.02406}) \times 2^R \times M/(M-U)$			
DFG → FTL	$100000 \times (0.825914 + 0.061504\sqrt{S + 1.02406}) \times 2^R \times M/(M-U)$			

Exit Count Thresholds for Jettison

Exit Case	Required Count for Jettison
Normal Exit	100×2^R
Exit that gets stuck in a loop	5×2^R

Jettison and Recompile

Another Example:

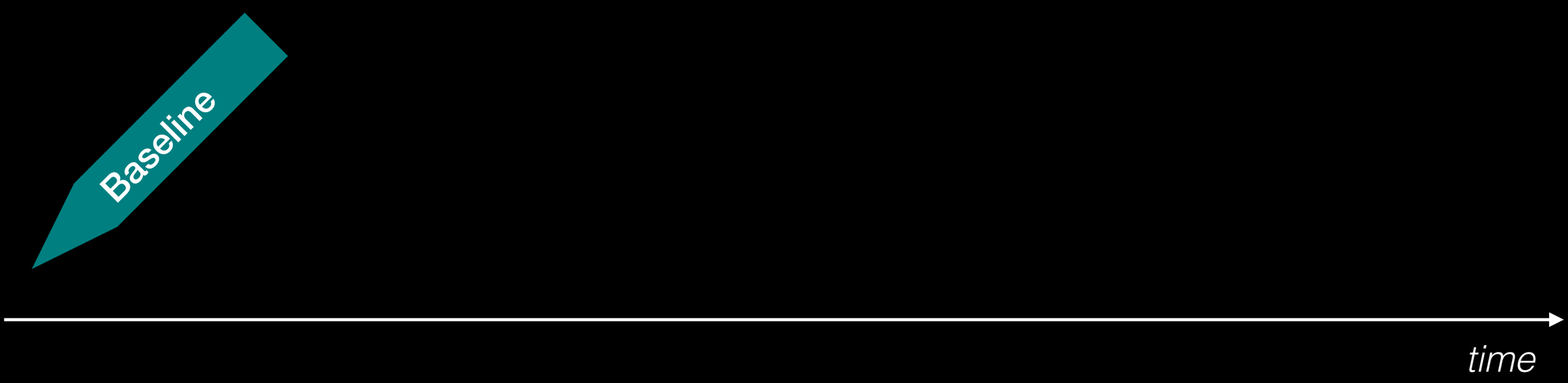
`_handlePropertyAccessExpression#D5n0Sd`

_handlePropertyAccessExpression#D5n0Sd

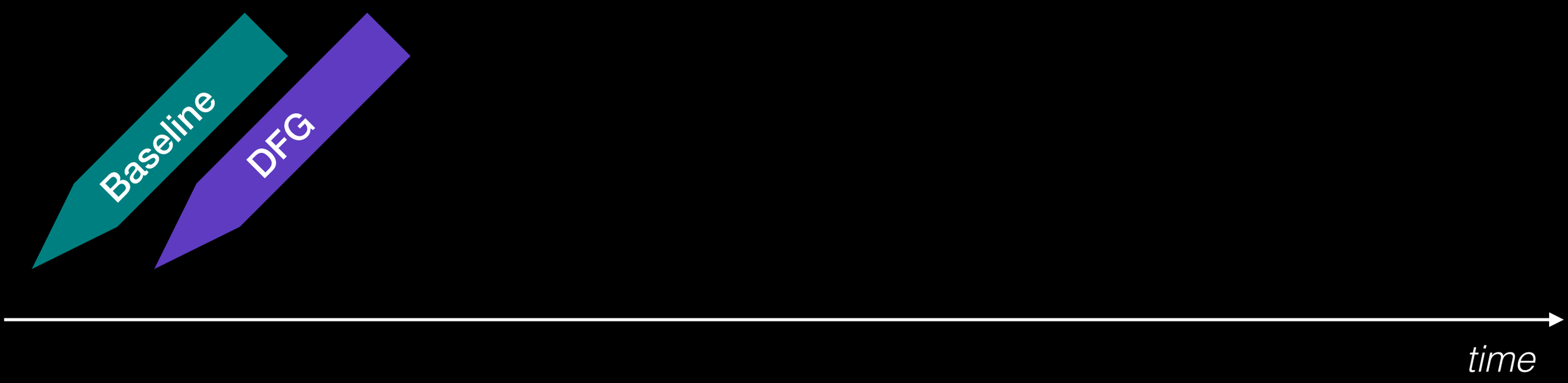
```
function (result, node)
{
    result.possibleGetOverloads = node.possibleGetOverloads;
    result.possibleSetOverloads = node.possibleSetOverloads;
    result.possibleAndOverloads = node.possibleAndOverloads;
    result.baseType = Node.visit(node.baseType, this);
    result.callForGet = Node.visit(node.callForGet, this);
    result.resultTypeForGet = Node.visit(node.resultTypeForGet, this);
    result.callForAnd = Node.visit(node.callForAnd, this);
    result.resultTypeForAnd = Node.visit(node.resultTypeForAnd, this);
    result.callForSet = Node.visit(node.callForSet, this);
    result.errorForSet = node.errorForSet;
    result.updateCalls();
}
```

_handlePropertyAccessExpression#D5n0Sd

_handlePropertyAccessExpression#D5n0Sd

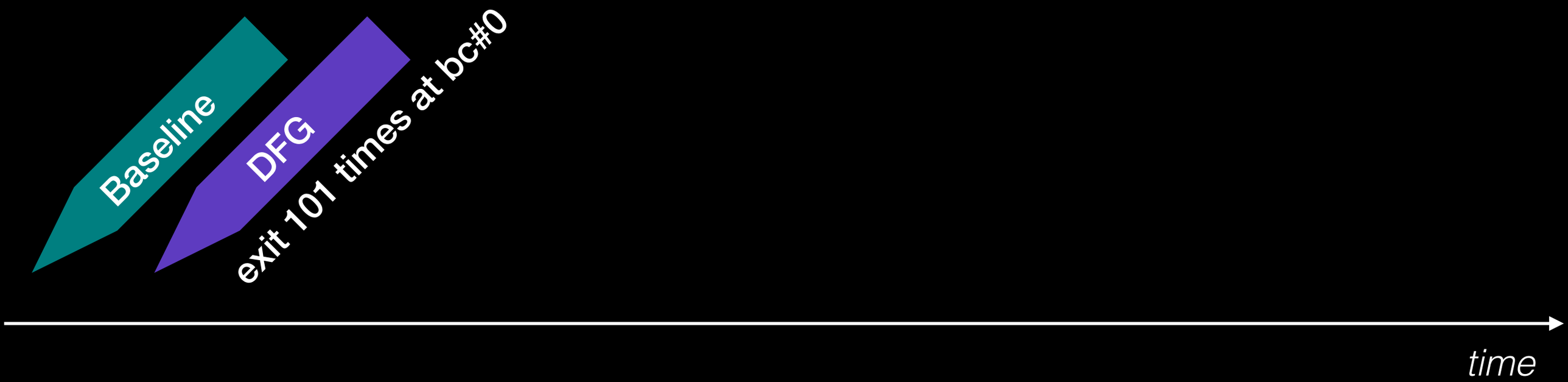


_handlePropertyAccessExpression#D5n0Sd



_handlePropertyAccessExpression#D5n0Sd

```
function (result, node)
{
    checkType(this, τ)
    …
}
```



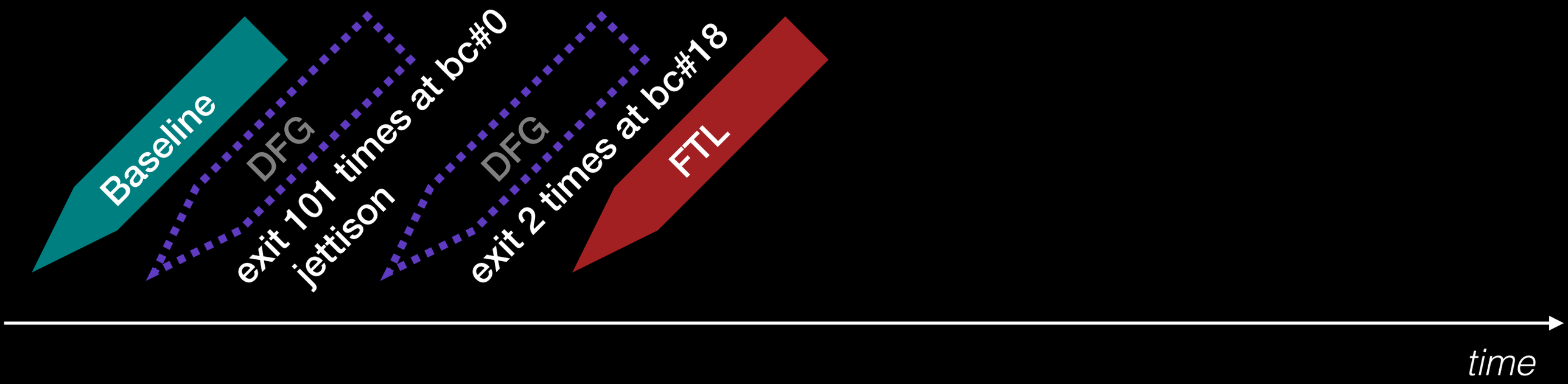

_handlePropertyAccessExpression#D5n0Sd
Baseline
DFG
exit 101 times at bc#0
jettison
time

_handlePropertyAccessExpression#D5n0Sd

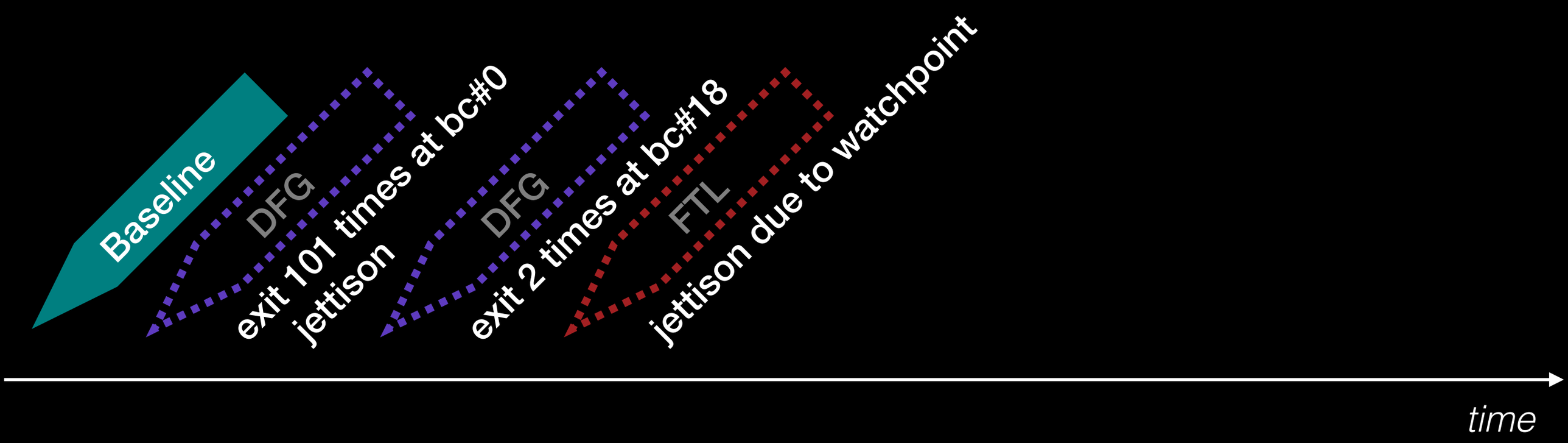
_handlePropertyAccessExpression#D5n0Sd

```
function (result, node)
{
    …
    checkType(result, σ)
    …
}
```

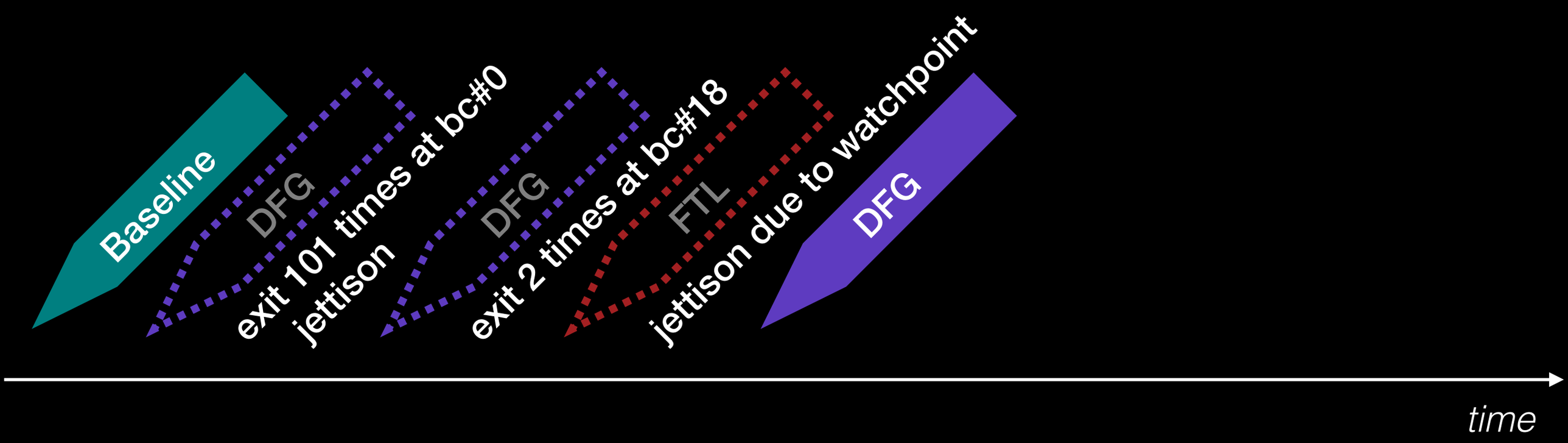
_handlePropertyAccessExpression#D5n0Sd



_handlePropertyAccessExpression#D5n0Sd

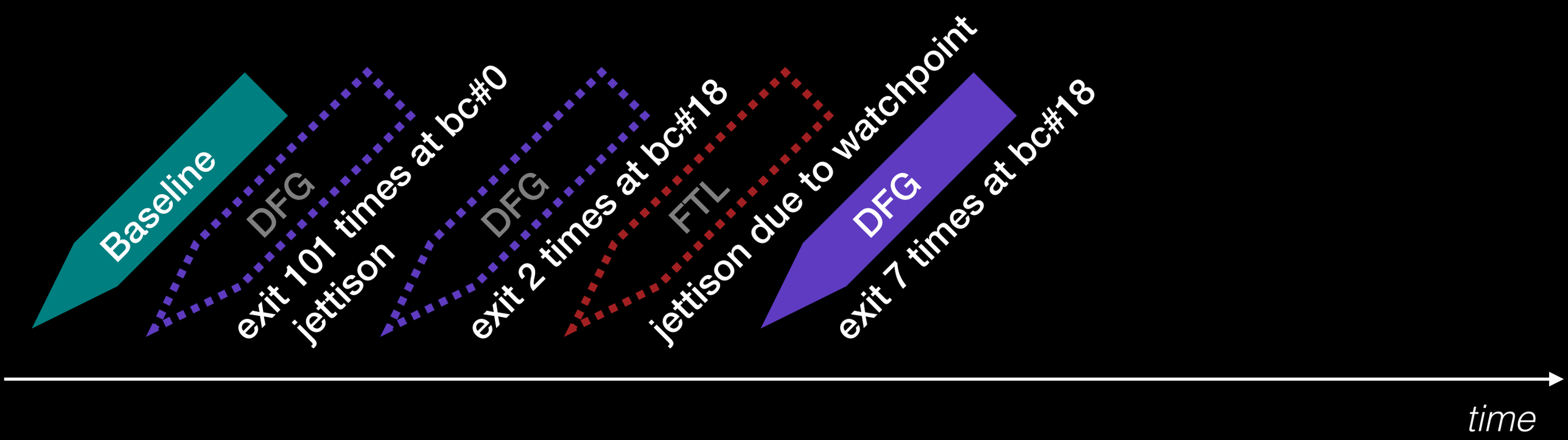


_handlePropertyAccessExpression#D5n0Sd

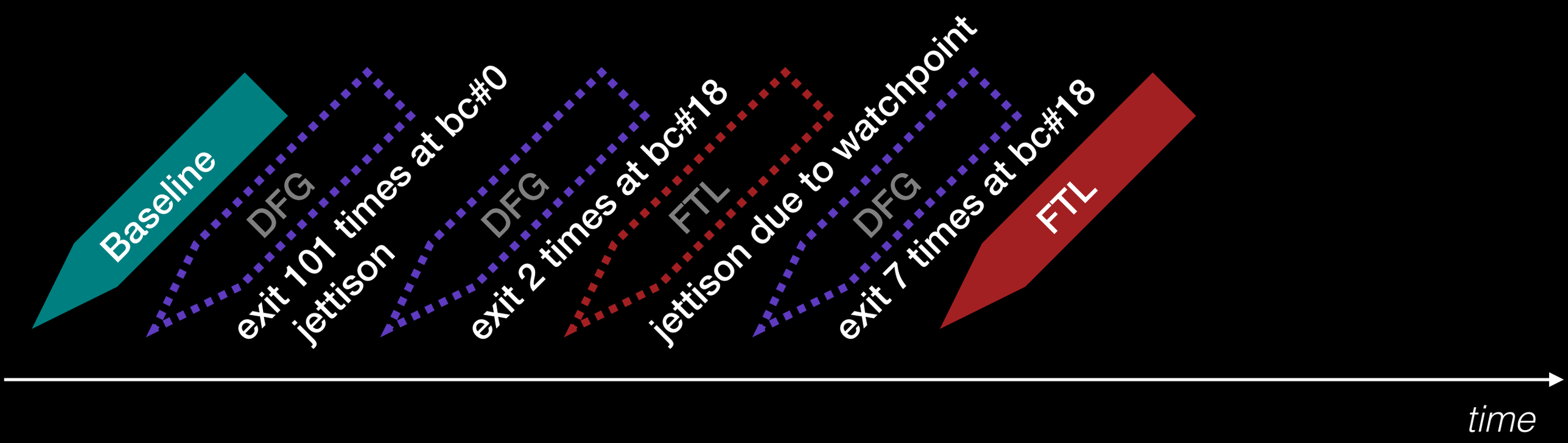


_handlePropertyAccessExpression#D5n0Sd

```
function (result, node)
{
    …
    checkType(result, σ)
    …
}
```

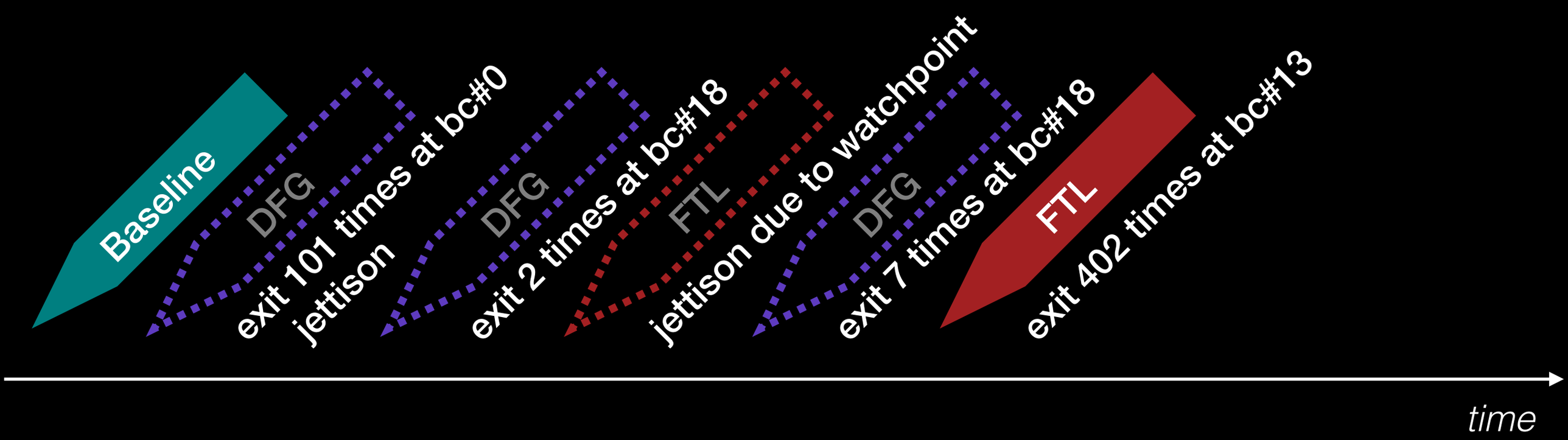


_handlePropertyAccessExpression#D5n0Sd

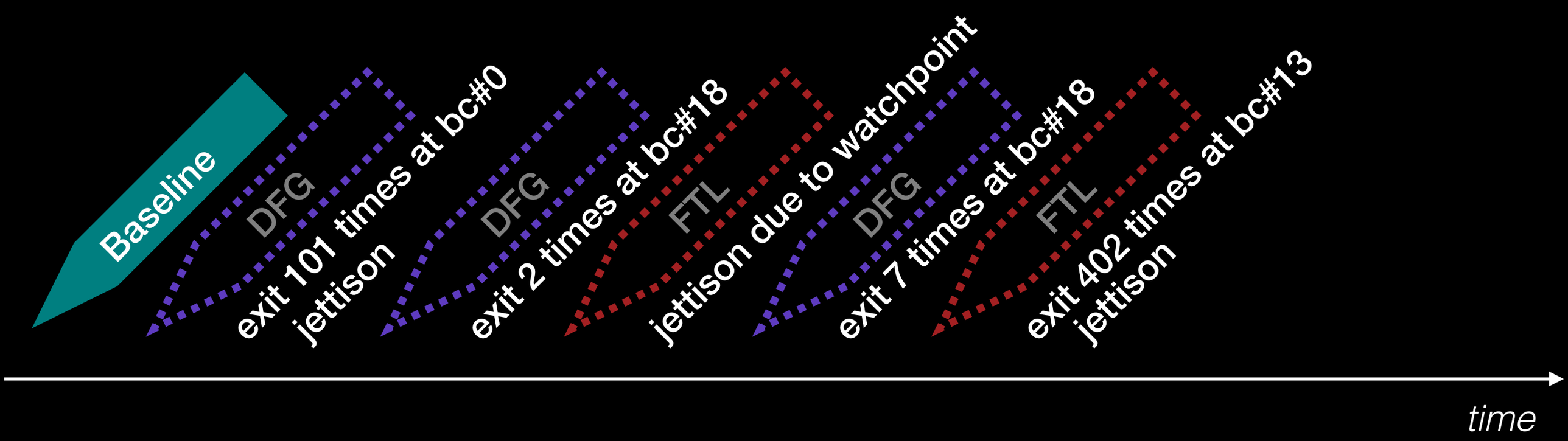


_handlePropertyAccessExpression#D5n0Sd

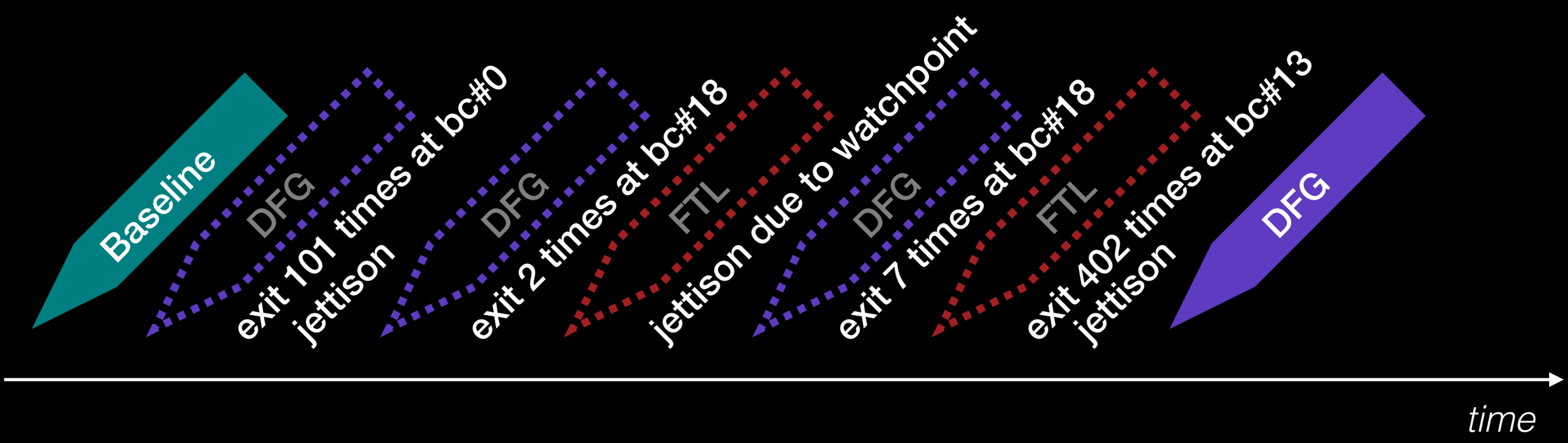
```
function (result, node)
{
    …
    checkType(node, υ)
    …
}
```



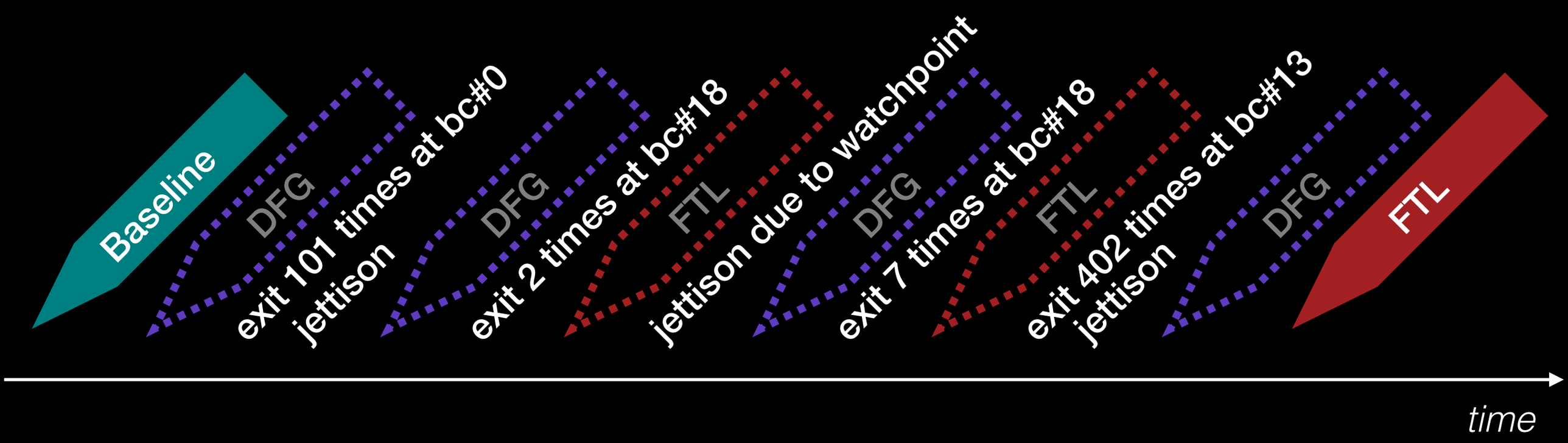
_handlePropertyAccessExpression#D5n0Sd



_handlePropertyAccessExpression#D5n0Sd

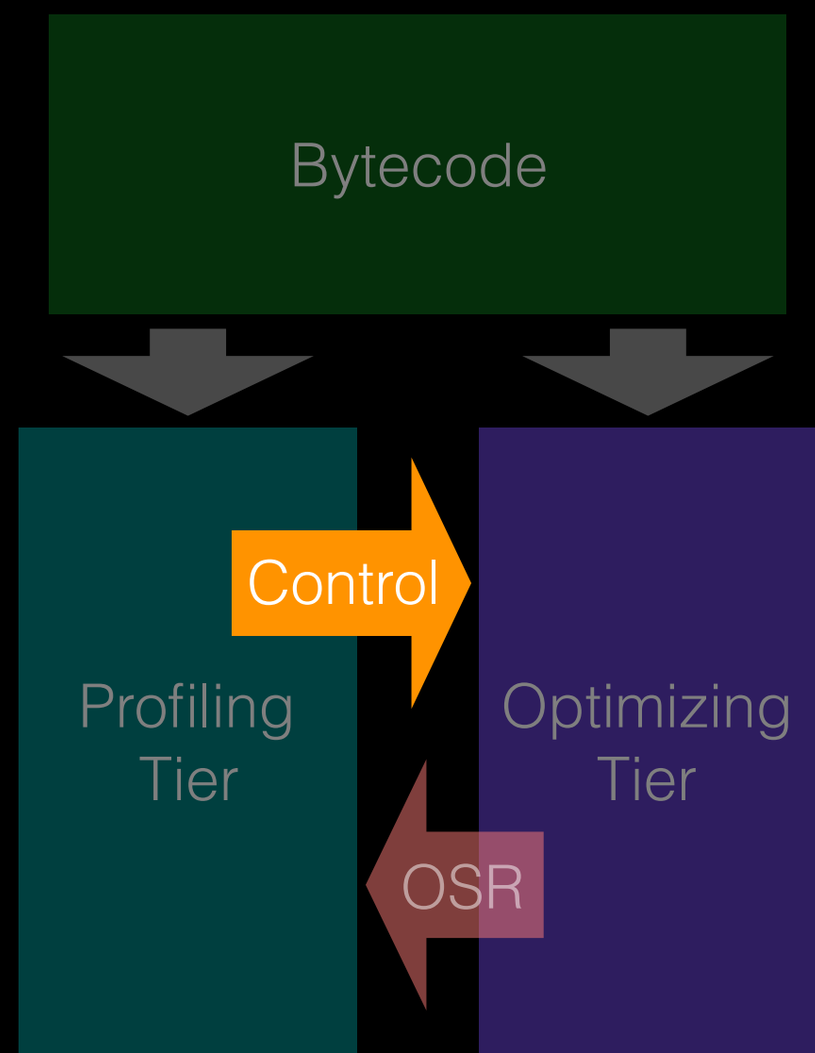


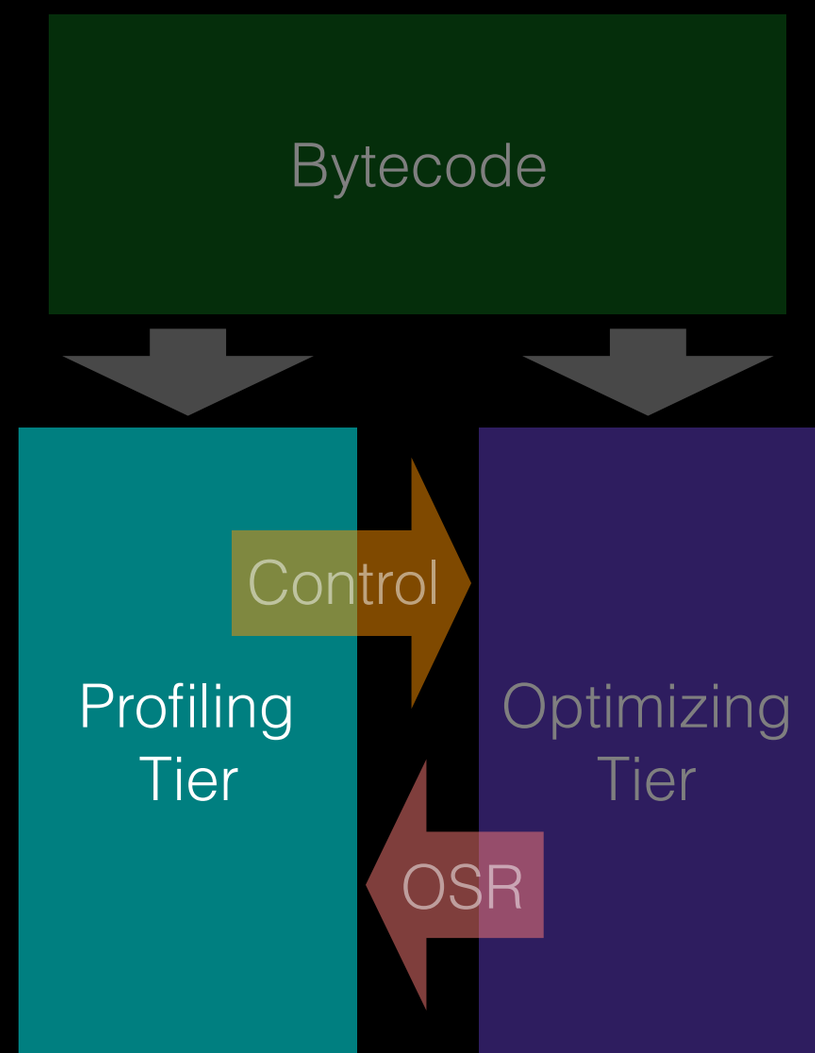
_handlePropertyAccessExpression#D5n0Sd



Control

- Execution Counting
- Exit Counting
- Recompilation

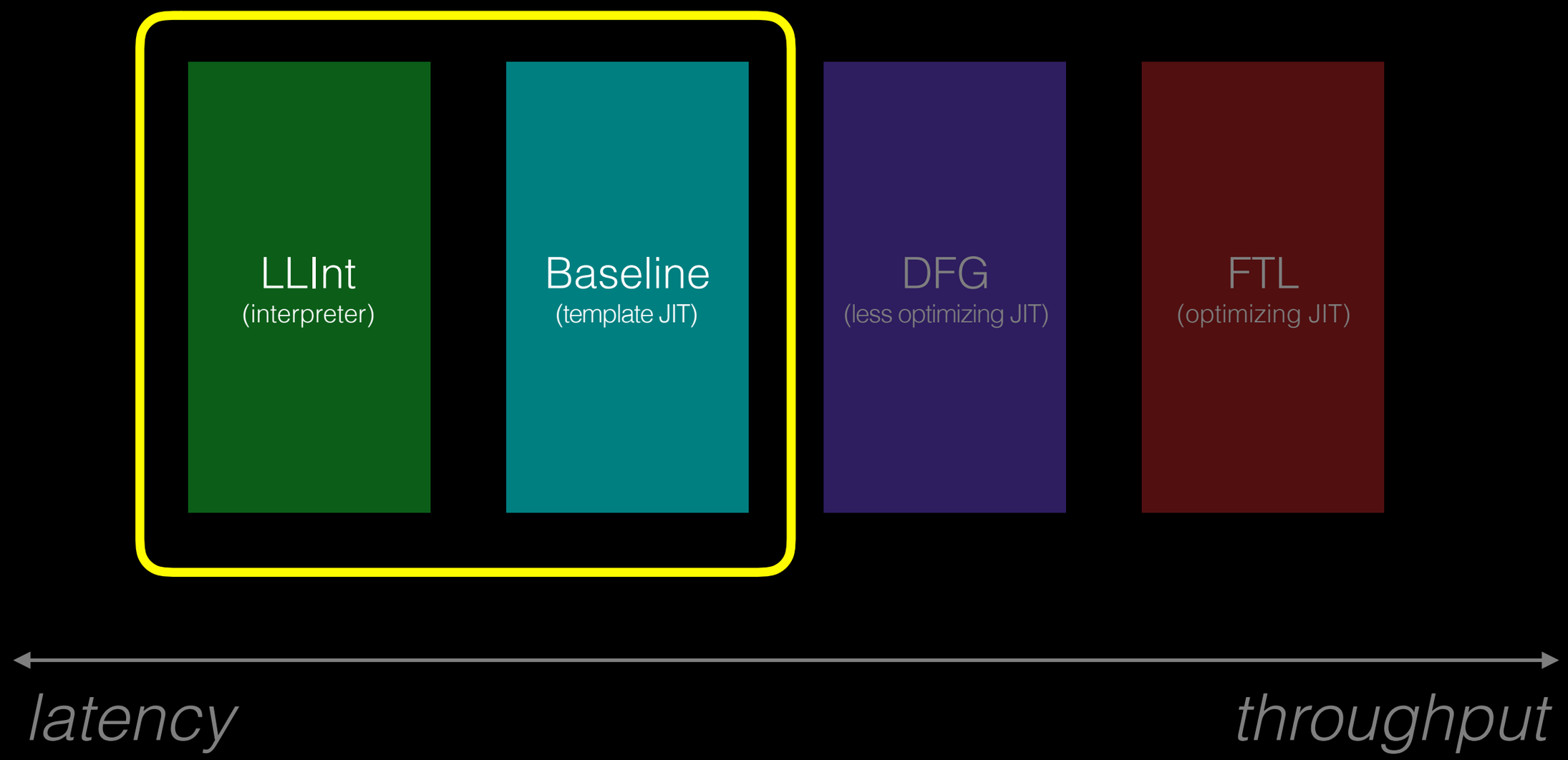

Bytecode
Control
Profiling Tier
Optimizing Tier
OSR


Bytecode
Control
Profiling Tier
Optimizing Tier
OSR

Profiling Tier

- Non-speculative execution engine(s)
- Profiling

Profiling


LLInt
(interpreter)
Baseline
(template JIT)
DFG
(less optimizing JIT)
FTL
(optimizing JIT)
latency
throughput

Low Level Interpreter

```
macro llintJumpTrueOrFalseOp(name, op, conditionOp)
    llintOpWithJump(op_%name%, op, macro (size, get, jump, dispatch)
        get(condition, t1)
        loadConstantOrVariable(size, t1, t0)
        btqnz t0, ~0xf, .slow
        conditionOp(t0, .target)
        dispatch()

    .target:
        jump(target)

    .slow:
        callSlowPath(_llint_slow_path_%name%)
        nextInstruction()
    end)
end
```

Low Level Interpreter

```
macro llintJumpTrueOrFalseOp(name, op, conditionOp)
    llintOpWithJump(op_%name%, op, macro (size, get, jump, dispatch)
        get(condition, t1)
        loadConstantOrVariable(size, t1, t0)
        btqnz t0, ~0xf, .slow
        conditionOp(t0, .target)
        dispatch()

    .target:
        jump(target)

    .slow:
        callSlowPath(_llint_slow_path_%name%)
        nextInstruction()
    end)
end
```

Baseline JIT

```
[   7] add                    loc6, arg1, arg2
        0x2f8084601a65: mov 0x30(%rbp), %rsi
        0x2f8084601a69: mov 0x38(%rbp), %rdx
        0x2f8084601a6d: cmp %r14, %rsi
        0x2f8084601a70: jb 0x2f8084601af2
        0x2f8084601a76: cmp %r14, %rdx
        0x2f8084601a79: jb 0x2f8084601af2
        0x2f8084601a7f: mov %esi, %eax
        0x2f8084601a81: add %edx, %eax
        0x2f8084601a83: jo 0x2f8084601af2
        0x2f8084601a89: or %r14, %rax
        0x2f8084601a8c: mov %rax, -0x38(%rbp)
```

Profiling Goals

- Cheap
- Useful

Useful Profiling

- Speculation is a bet.
- Profiling makes it a value bet.

Winning in the Average

$$\text{Expected Value of Bet} = p \times B - (1 - p) \times C$$

Variable	Meaning
p	Probability of Winning
B	Benefit of winning (positive)
C	Cost of losing (positive)

Winning in the Average

Good bet iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning
p	Probability of Winning
B	Benefit of winning (positive)
C	Cost of losing (positive)

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	
C	Time Lost by Speculation Failure	

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	
C	Time Lost by Speculation Failure	

Execution Time = (3.97 ns) × (Bytecodes in LLInt)

+ (1.71 ns) × (Bytecodes in Baseline)

+ (0.349 ns) × (Bytecodes in DFG)

+ (0.225 ns) × (Bytecodes in FTL)

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	
C	Time Lost by Speculation Failure	

Execution Time = (3.97 ns) × (Bytecodes in LLInt)
+ (1.71 ns) × (Bytecodes in Baseline)
+ (0.349 ns) × (Bytecodes in DFG)
+ (0.225 ns) × (Bytecodes in FTL)

$B < 1.71 - 0.225 = 1.48$ ns

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	
C	Time Lost by Speculation Failure	

Execution Time = (3.97 ns) × (Bytecodes in LLInt)

+ (1.71 ns) × (Bytecodes in Baseline)

+ (0.349 ns) × (Bytecodes in DFG)

+ (0.225 ns) × (Bytecodes in FTL)

B < 1.71 - 0.225 = 1.48 ns

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	
C	Time Lost by Speculation Failure	

Execution Time = (3.97 ns) × (Bytecodes in LLInt)

+ (1.71 ns) × (Bytecodes in Baseline)

+ (0.349 ns) × (Bytecodes in DFG)

+ (0.225 ns) × (Bytecodes in FTL)

B < 1.71 - 0.225 = 1.48 ns

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	< 1.48 ns
C	Time Lost by Speculation Failure	

Execution Time = (3.97 ns) × (Bytecodes in LLInt)
+ (1.71 ns) × (Bytecodes in Baseline)
+ (0.349 ns) × (Bytecodes in DFG)
+ (0.225 ns) × (Bytecodes in FTL)

B < 1.71 - 0.225 = 1.48 ns

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	< 1.48 ns
C	Time Lost by Speculation Failure	

C = (OSR exit cost) + (Bytecodes before reentry) × B + (Recompile Cost) / (exits to recompile)

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	< 1.48 ns
C	Time Lost by Speculation Failure	DFG ~ 2499 ns

C = (OSR exit cost) + (Bytecodes before reentry) × B + (Recompile Cost) / (exits to recompile)

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	
B	Time Saved by Speculation	< 1.48 ns
C	Time Lost by Speculation Failure	DFG ~ 2499 ns FTL ~ 9998 ns

C = (OSR exit cost) + (Bytecodes before reentry) × B + (Recompile Cost) / (exits to recompile)

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	**Good speculation iff $p > 0.9994$**
B	Time Saved by Speculation	< 1.48 ns
C	Time Lost by Speculation Failure	DFG ~ 2499 ns FTL ~ 9998 ns

C = (OSR exit cost) + (Bytecodes before reentry) × B + (Recompile Cost) / (exits to recompile)

Winning at Speculation

Good speculation iff $p \times B - (1 - p) \times C > 0$

Variable	Meaning	Likely Value
p	Probability of Winning	**Good speculation iff p ~ 1**
B	Time Saved by Speculation	< 1.48 ns
C	Time Lost by Speculation Failure	DFG ~ 2499 ns FTL ~ 9998 ns

C = (OSR exit cost) + (Bytecodes before reentry) × B + (Recompile Cost) / (exits to recompile)

Winning at Speculation

Only speculate if we believe that we will win every time.

Winning at Speculation

Profiling should record counterexamples to useful speculations.

Winning at Speculation

Profiling should run for a *long* time.

Winning at Speculation

Don’t stress when speculation fails, unless it fails in the average.

Profiling Sources in JSC

- Case Flags
- Case Counts
- Value Profiling
- Inline Caches
- Watchpoints
- Exit Flags

Profiling Sources in JSC

- Case Flags — *branch speculation*
- Case Counts — *branch speculation*
- Value Profiling — *type inference of values*
- Inline Caches — *type inference of object structure*
- Watchpoints — *heap speculation*
- Exit Flags — *speculation backoff*

Case Flags

Case flag = tells if a counterexample to a speculation ever happened.

Case Flags

```
class StructureStubInfo {
    …

    ALWAYS_INLINE bool considerCaching(
        CodeBlock* codeBlock, Structure* structure)
    {
        // We never cache non-cells.
        if (!structure) {
            sawNonCell = true;
            return false;
        }

        …
```

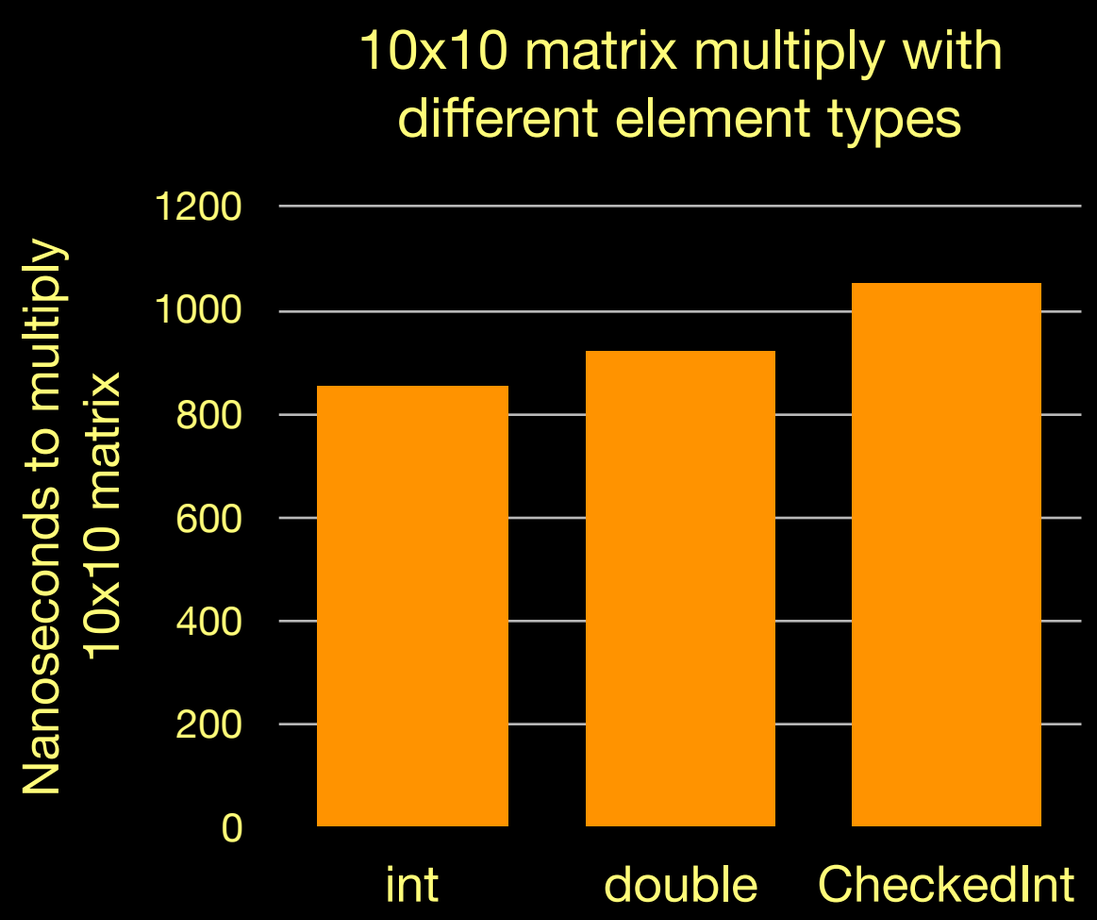
Case Flags

```
void ArithProfile::emitSetDouble(CCallHelpers& jit) const
{
    if (shouldEmitSetDouble())
        jit.or32(
            CCallHelpers::TrustedImm32(
                ArithProfile::Int32Overflow |
                ArithProfile::Int52Overflow |
                ArithProfile::NegZeroDouble |
                ArithProfile::NonNegZeroDouble),
            CCallHelpers::AbsoluteAddress(addressOfBits()));
}
```

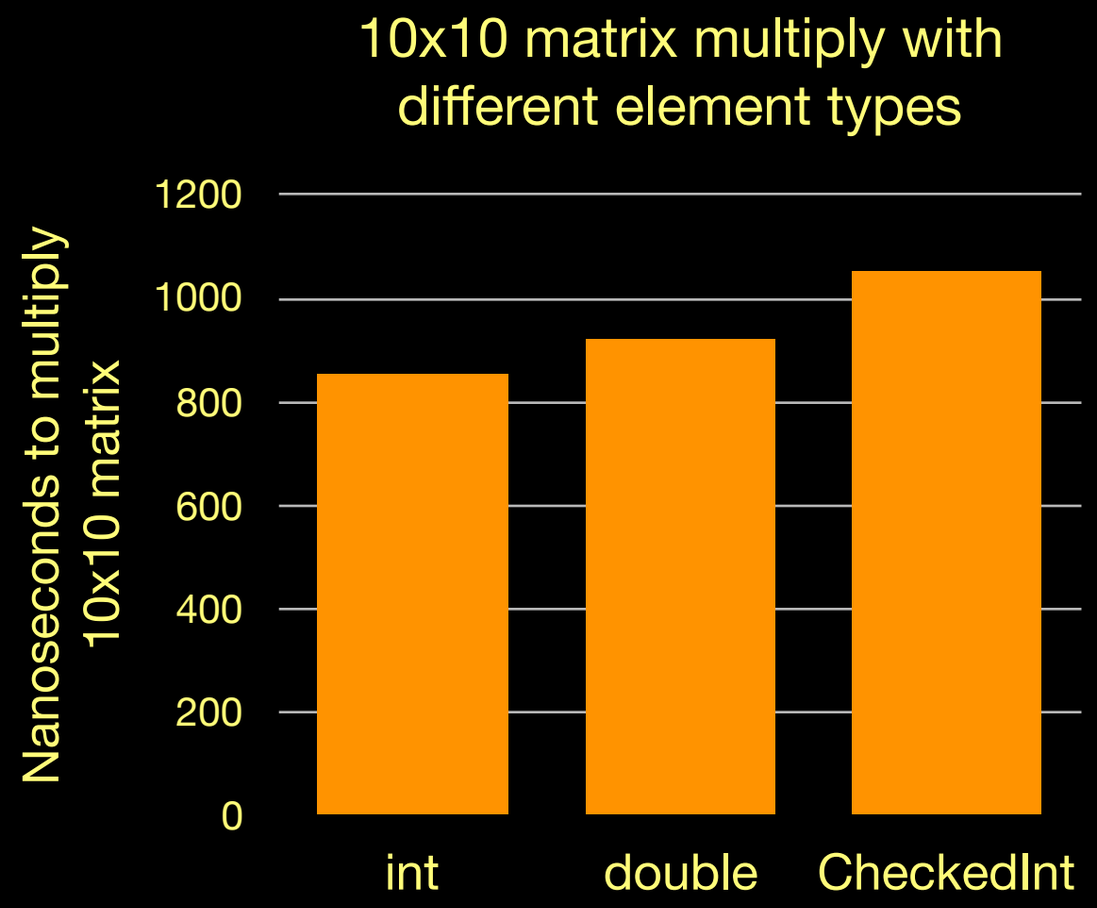
Why infer int32?

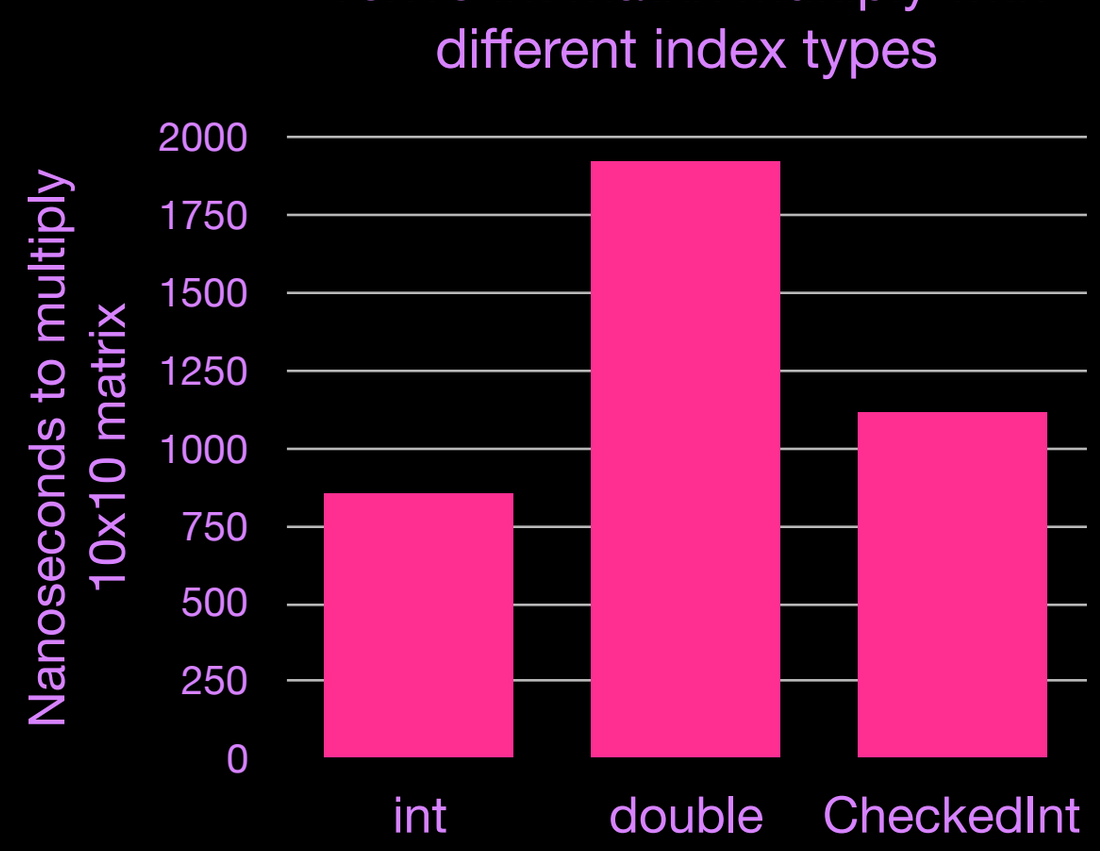
```
template<typename T, typename U>
void multiply(Mat<T>& result,
              const Mat<T>& left,
              const Mat<T>& right)
{
   for (U resultColumn = result.numColumns(); resultColumn--;) {
        for (U resultRow = result.numRows(); resultRow--;) {
            T& resultCell = result.at(resultRow, resultColumn);
            resultCell = T();
            for (U i = left.numColumns(); i--;) {
                resultCell +=
                    left.at(resultRow, i) *
                    right.at(i, resultColumn);
            }
        }
    }
}
```

```
template<typename T, typename U>
void multiply(Mat<T>& result,
              const Mat<T>& left,
              const Mat<T>& right)
{
   for (U resultColumn = result.numColumns(); resultColumn--;) {
        for (U resultRow = result.numRows(); resultRow--;) {
            T& resultCell = result.at(resultRow, resultColumn);
            resultCell = T();
            for (U i = left.numColumns(); i--;) {
                resultCell +=
                    left.at(resultRow, i) *
                    right.at(i, resultColumn);
            }
        }
    }
}
```

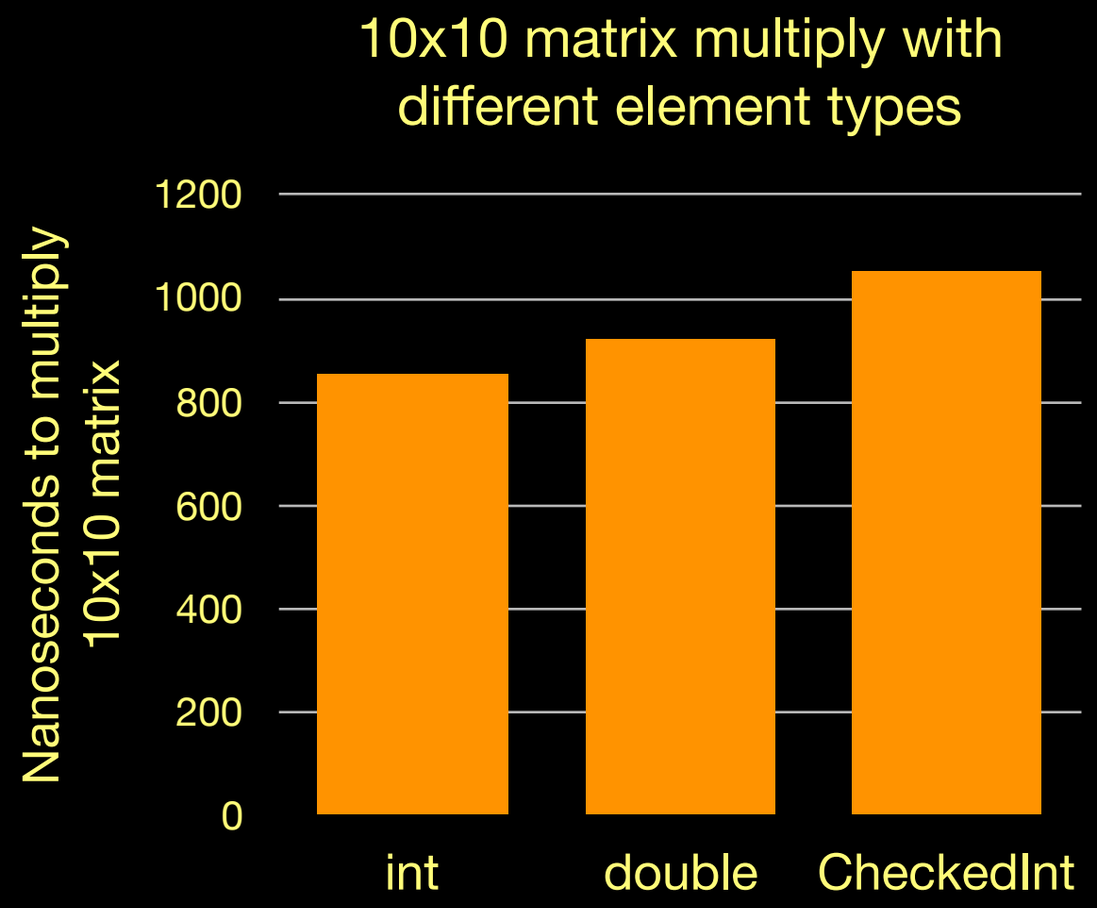


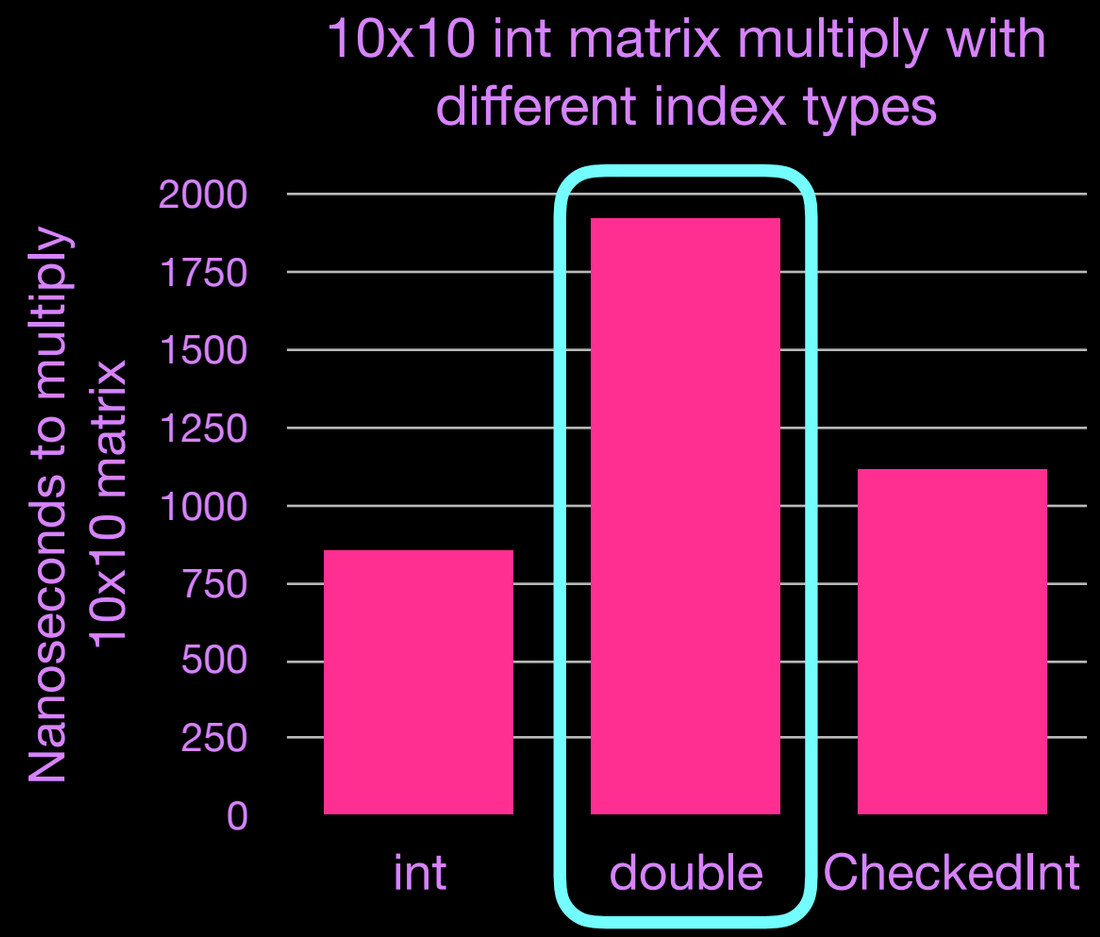
```
template<typename T, typename U>
void multiply(Mat<T>& result,
              const Mat<T>& left,
              const Mat<T>& right)
{
   for (U resultColumn = result.numColumns(); resultColumn--;) {
        for (U resultRow = result.numRows(); resultRow--;) {
            T& resultCell = result.at(resultRow, resultColumn);
            resultCell = T();
            for (U i = left.numColumns(); i--;) {
                resultCell +=
                    left.at(resultRow, i) *
                    right.at(i, resultColumn);
            }
        }
    }
}
```





```
template<typename T, typename U>
void multiply(Mat<T>& result,
              const Mat<T>& left,
              const Mat<T>& right)
{
   for (U resultColumn = result.numColumns(); resultColumn--;) {
        for (U resultRow = result.numRows(); resultRow--;) {
            T& resultCell = result.at(resultRow, resultColumn);
            resultCell = T();
            for (U i = left.numColumns(); i--;) {
                resultCell +=
                    left.at(resultRow, i) *
                    right.at(i, resultColumn);
            }
        }
    }
}
```





Use Int32 whenever possible to avoid future double→int conversions

Case Flags Example: Add

Profiling Tier

```
int32_t left = …;
int32_t right = …;
ArithProfile* profile = …;
int32_t intResult;
JSValue result;
if (UNLIKELY(addOverflowed(
        left, right,
        &intResult))) {
    result = jsNumber(
        double(left) +
        double(right));
    profile->setObservedInt32Overflow();
} else
    result = jsNumber(intResult);
```

Case Flags Example: Add

Profiling Tier

```
int32_t left = …;
int32_t right = …;
ArithProfile* profile = …;
int32_t intResult;
JSValue result;
if (UNLIKELY(addOverflowed(
        left, right,
        &intResult))) {
    result = jsNumber(
        double(left) +
        double(right));
    profile->setObservedInt32Overflow();
} else
    result = jsNumber(intResult);
```

Case Flags Example: Add

Profiling Tier

```
int32_t left = …;
int32_t right = …;
ArithProfile* profile = …;
int32_t intResult;
JSValue result;
if (UNLIKELY(addOverflowed(
        left, right,
        &intResult))) {
    result = jsNumber(
        double(left) +
        double(right));
    profile->setObservedInt32Overflow();
} else
    result = jsNumber(intResult);
```

Case Flags Example: Add

Profiling Tier

```
int32_t left = …;
int32_t right = …;
ArithProfile* profile = …;
int32_t intResult;
JSValue result;
if (UNLIKELY(addOverflowed(
        left, right,
        &intResult))) {
    result = jsNumber(
        double(left) +
        double(right));
    profile->setObservedInt32Overflow();
} else
    result = jsNumber(intResult);
```

Case Flags Example: Add

Profiling Tier	Optimizing Tier

`int32_t left = …;`
`int32_t right = …;`
`ArithProfile* profile = …;`
`int32_t intResult;`
`JSValue result;`
`if (UNLIKELY(addOverflowed(`
`  left, right,`
`  &intResult))) {`
`  result = jsNumber(`
`  double(left) +`
`  double(right));`
`  profile->setObservedInt32Overflow();`
`} else`
`  result = jsNumber(intResult);`
	`// if !profile->didObserveInt32Overflow()`
	
	`int32_t left = …;`
	`int32_t right = …;`
	`int32_t result;`
	`speculate(!addOverflowed(`
	`  left, right,`
	`  &result));`

Case Flags Example: Add

Profiling Tier	Optimizing Tier

`int32_t left = …;`
`int32_t right = …;`
`ArithProfile* profile = …;`
`int32_t intResult;`
`JSValue result;`
`if (UNLIKELY(addOverflowed(`
`  left, right,`
`  &intResult))) {`
`  result = jsNumber(`
`  double(left) +`
`  double(right));`
`  profile->setObservedInt32Overflow();`
`} else`
`  result = jsNumber(intResult);`
	`// if profile->didObserveInt32Overflow()`
	
	`double left =…;`
	`double right = …;`
	`double result = left + right;`

Case Counts

```
RareCaseProfile* rareCaseProfile = 0;
if (shouldEmitProfiling()) {
    rareCaseProfile =
        m_codeBlock->addRareCaseProfile(m_bytecodeOffset);
}
…
if (shouldEmitProfiling()) {
    add32(
        TrustedImm32(1),
        AbsoluteAddress(&rareCaseProfile->m_counter));
}
```

Rare Case Count Thresholds

Case	Count Threshold	Action
`this` **conversion**	10	Assume `this` is exotic.
arithmetic slow path	20	Assume integer math overflowed to double.

Profiling Sources in JSC

- Case Flags — *branch speculation*
- Case Counts — *branch speculation*
- Value Profiling — *type inference of values*
- Inline Caches — *type inference of object structure*
- Watchpoints — *heap speculation*
- Exit Flags — *speculation backoff*

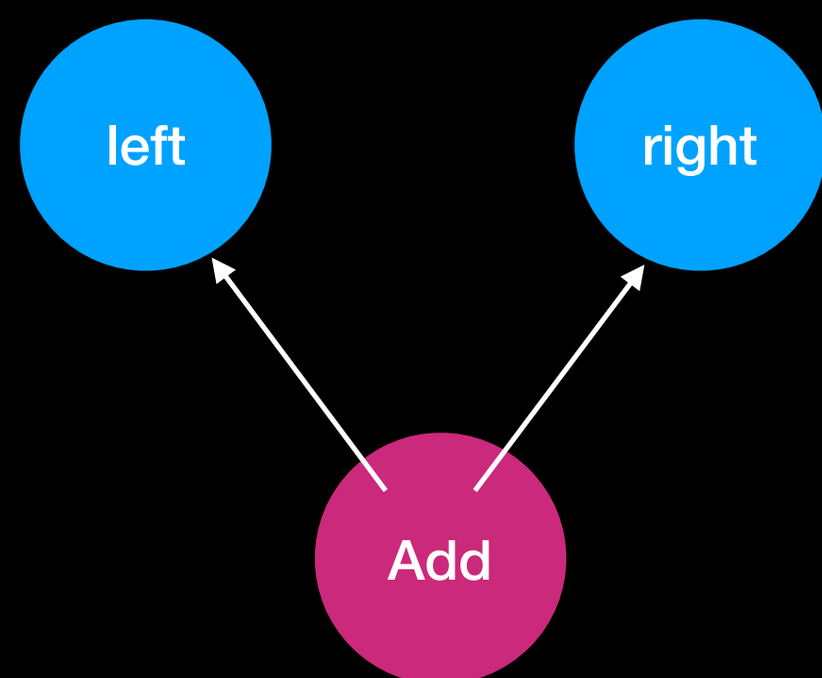
Value Profiling

```
macro valueProfile(op, metadata, value)
    storeq value, %op%::Metadata::profile.m_buckets[metadata]
end
```

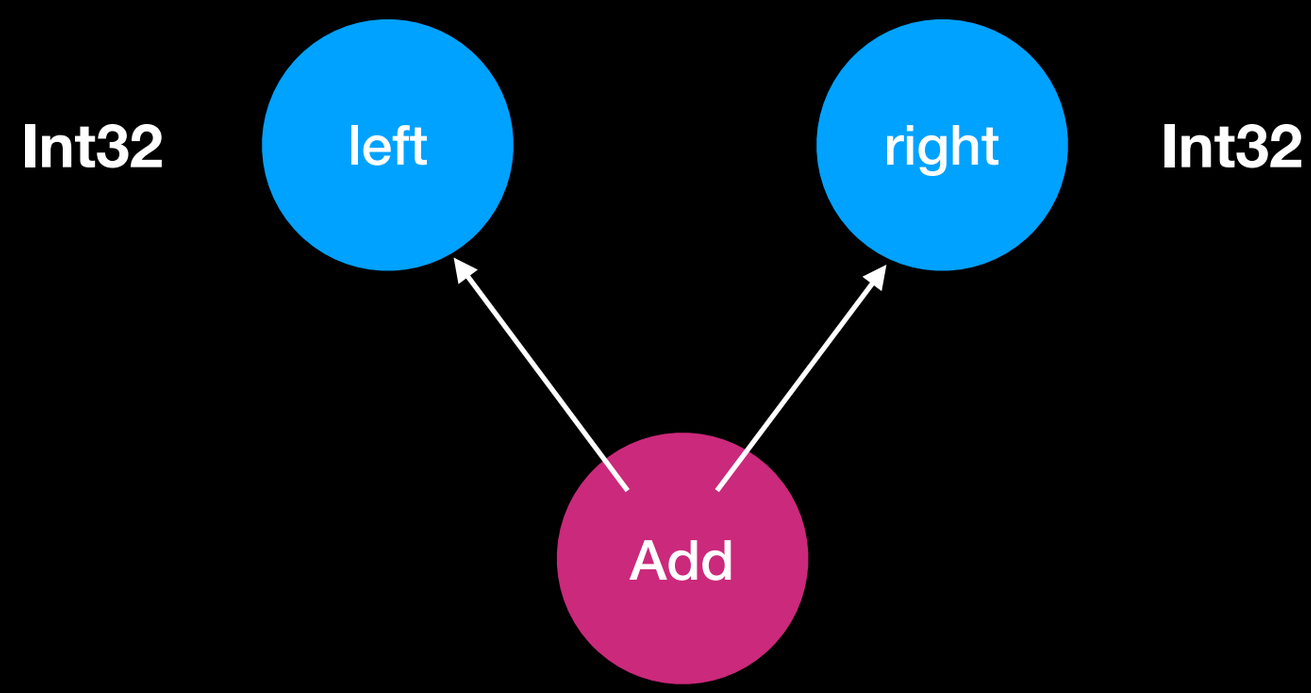
Value Profiling Idea

- Use static analysis whenever possible.
- Value profiling fills in the blanks:
  - loads
  - calls
  - etc

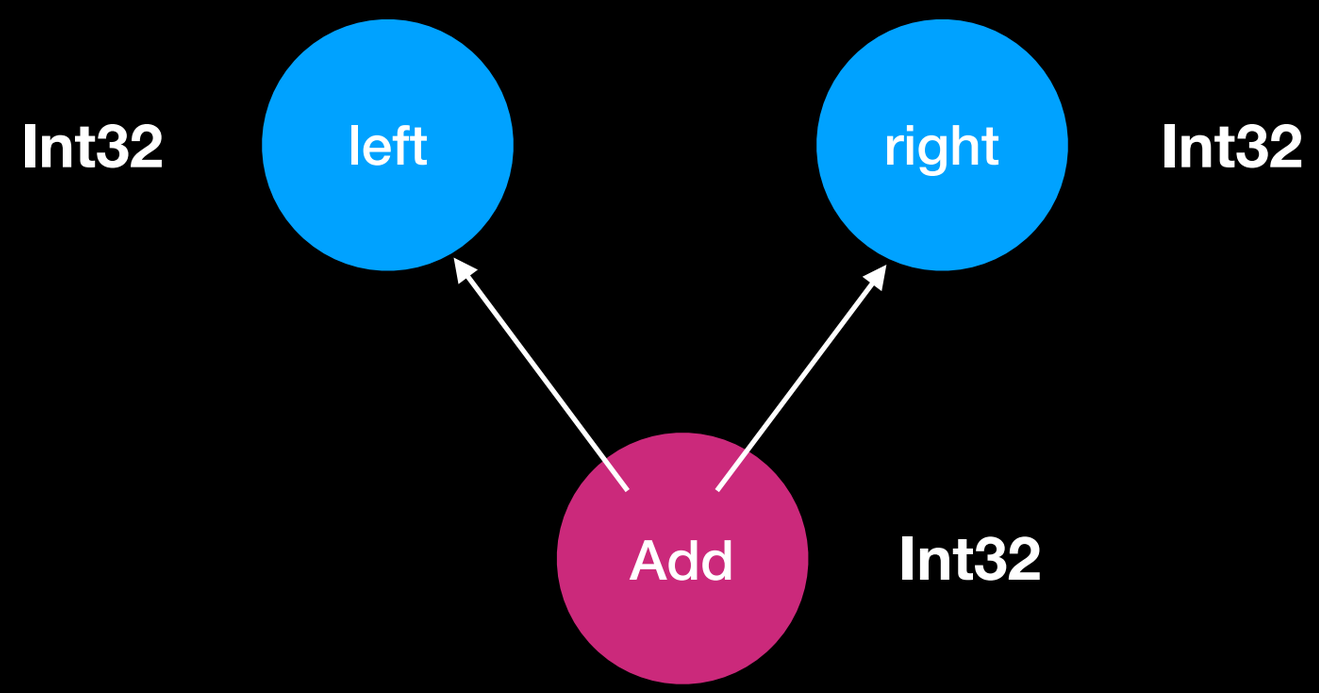
Prediction Propagation



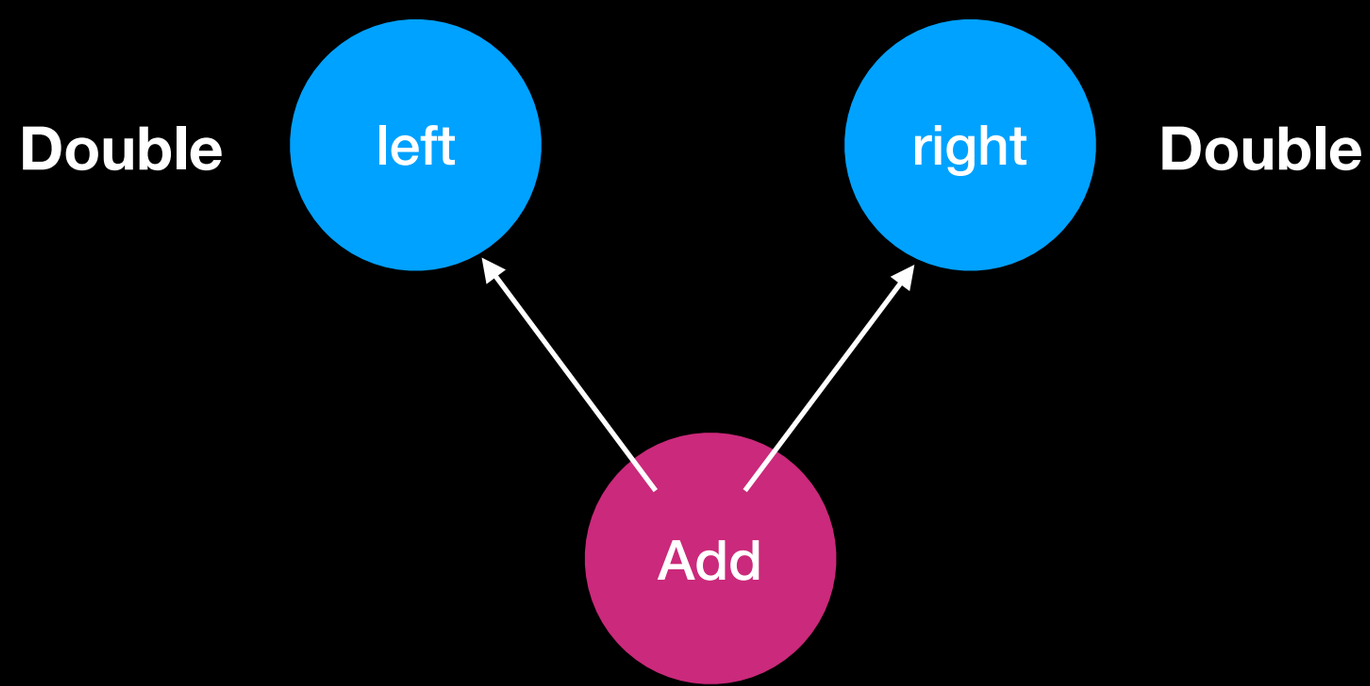
Prediction Propagation



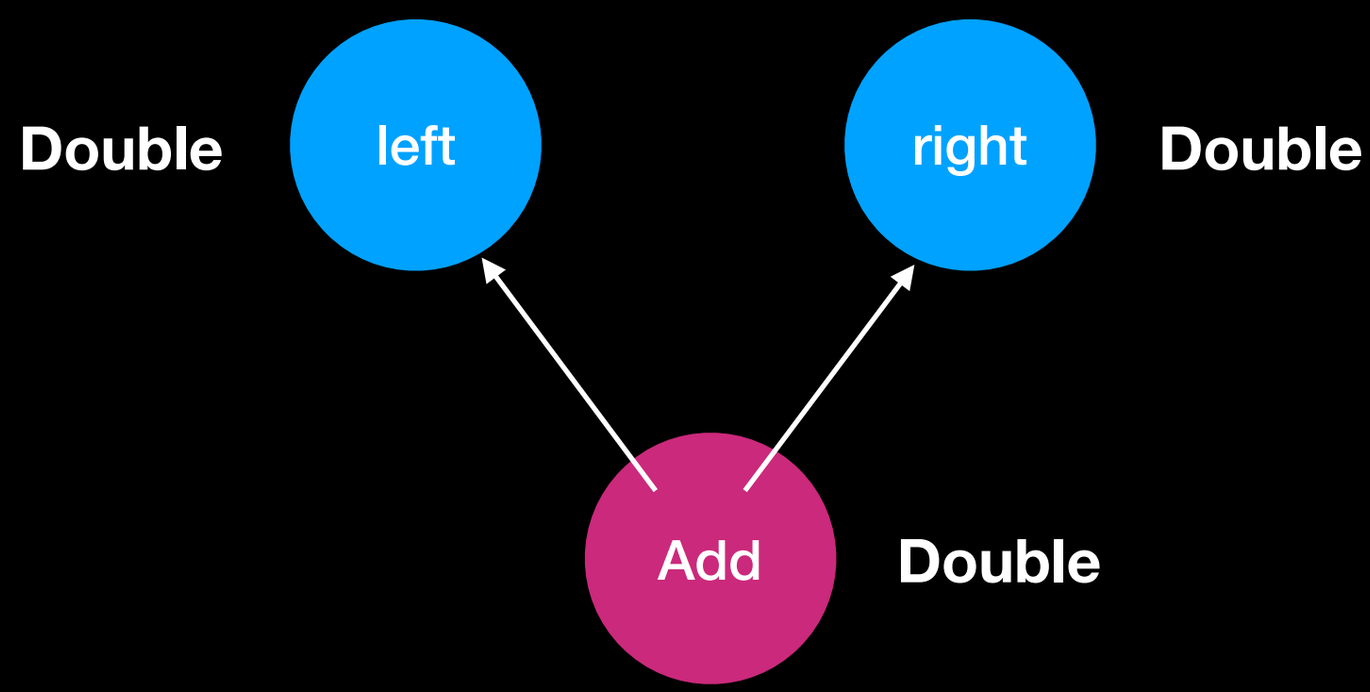
Prediction Propagation



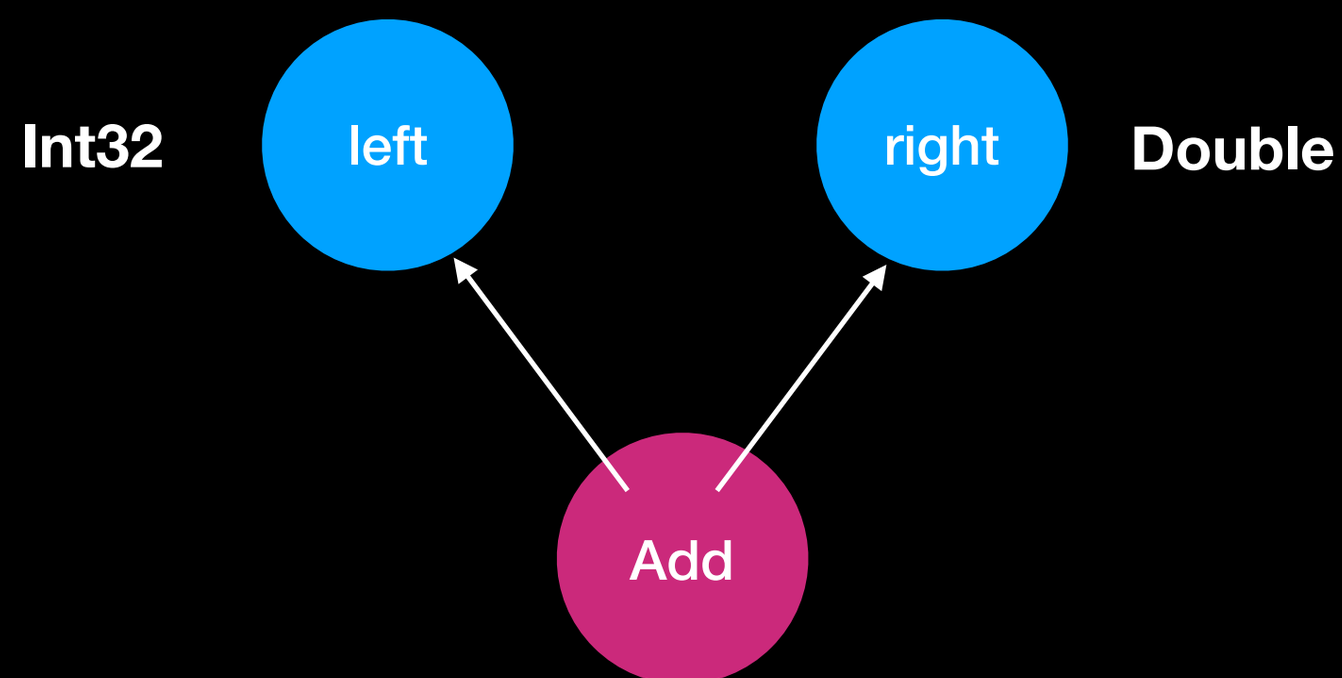
Prediction Propagation



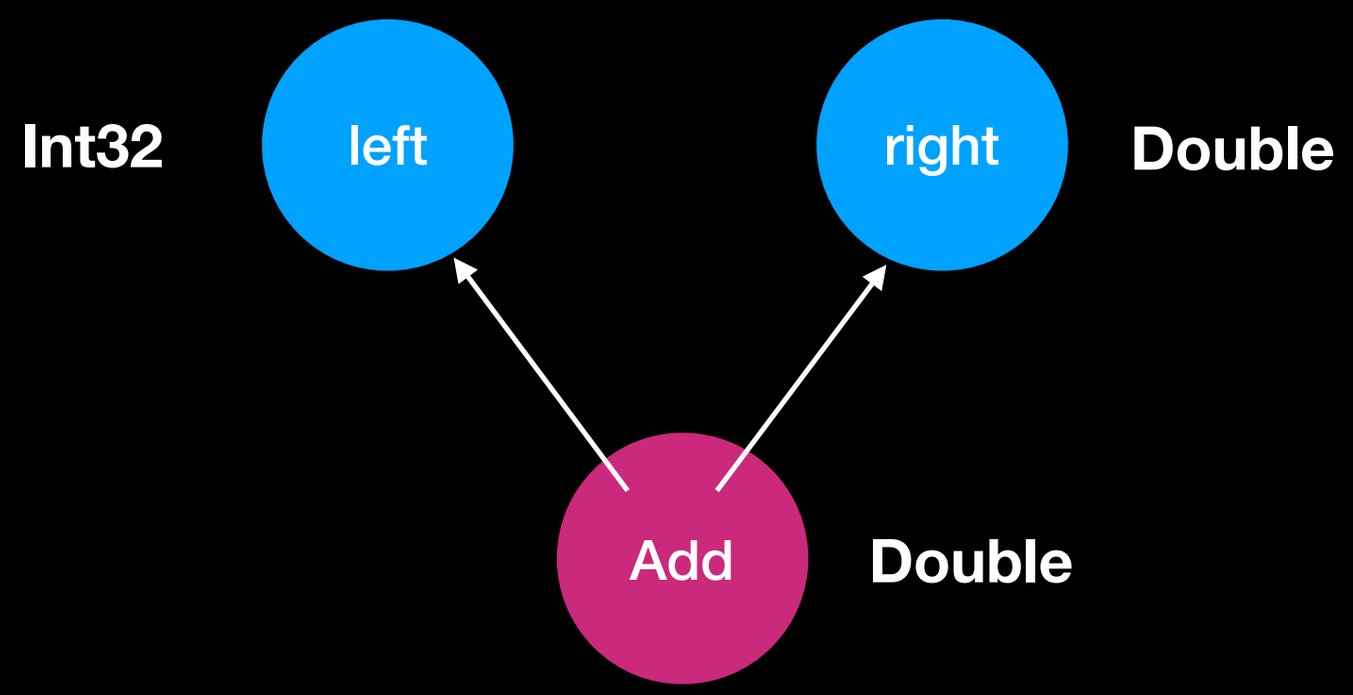
Prediction Propagation



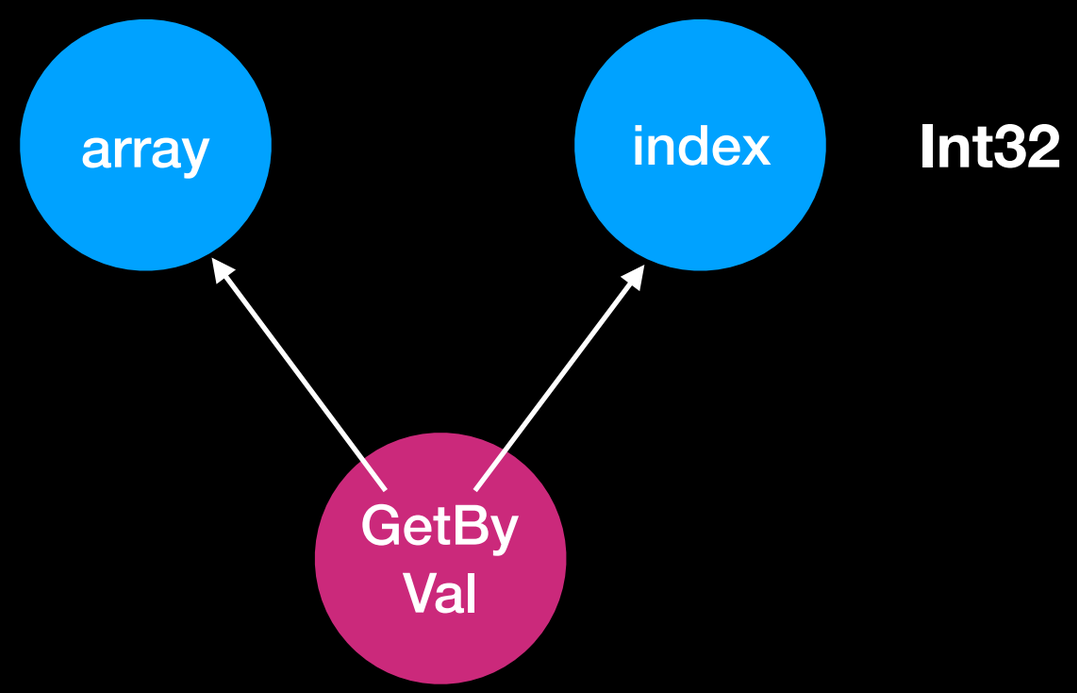
Prediction Propagation



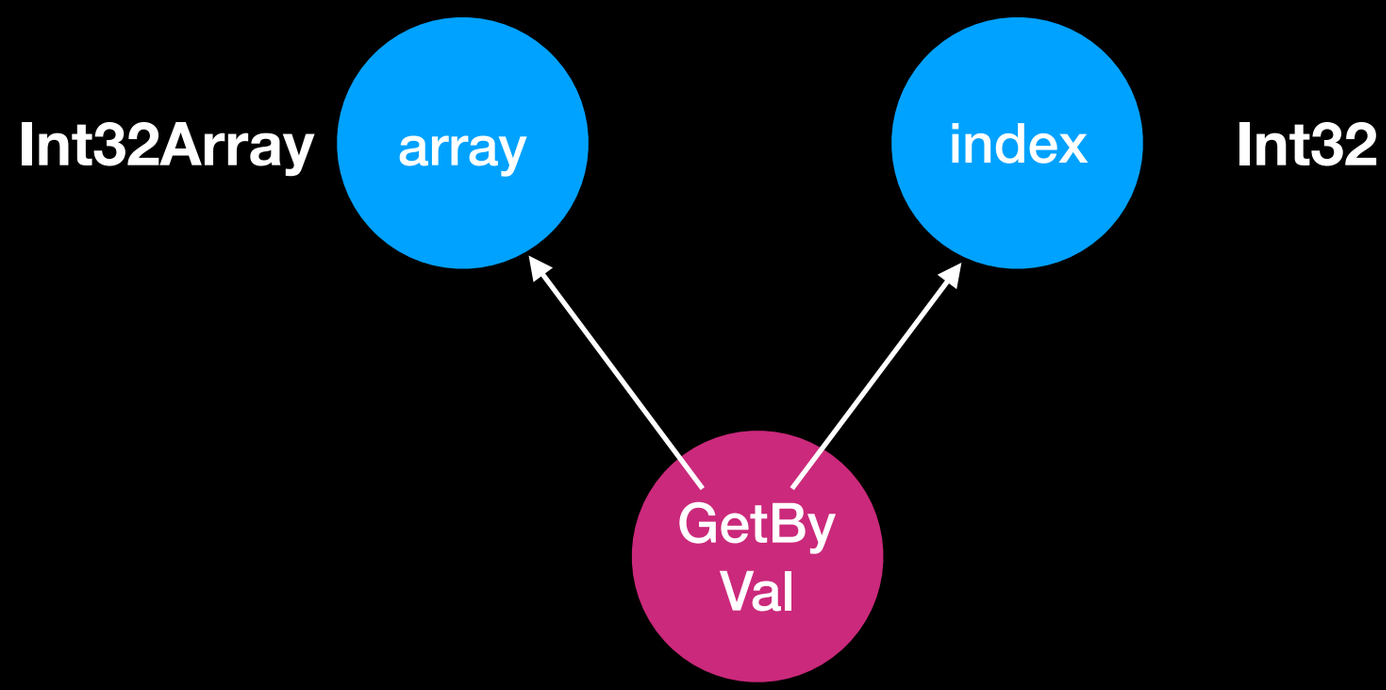
Prediction Propagation



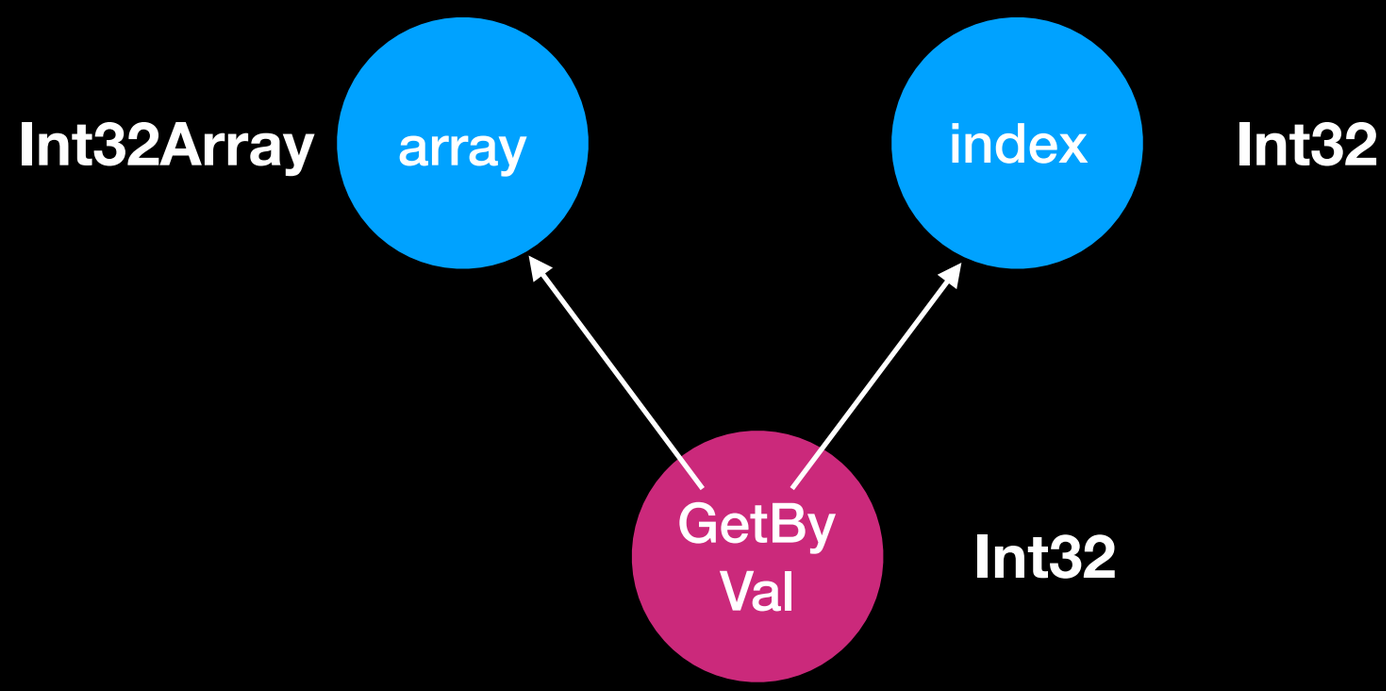
Prediction Propagation



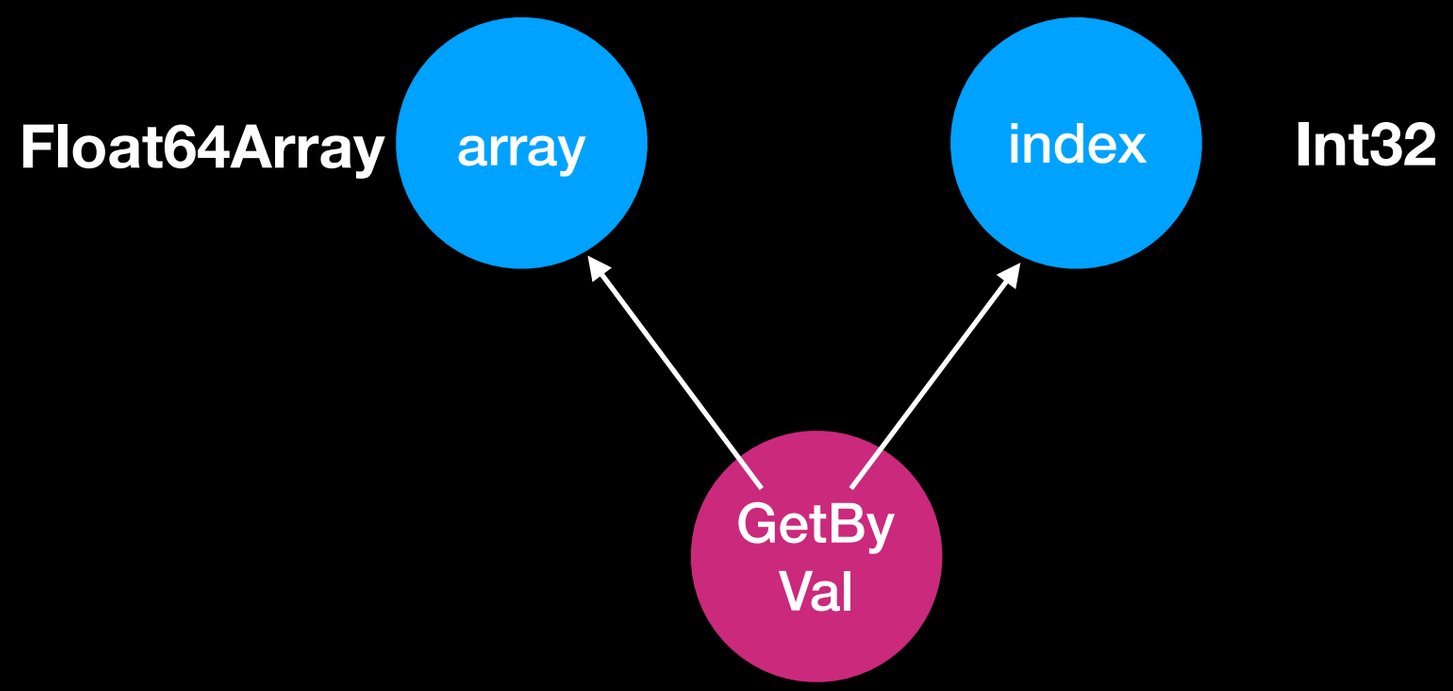
Prediction Propagation



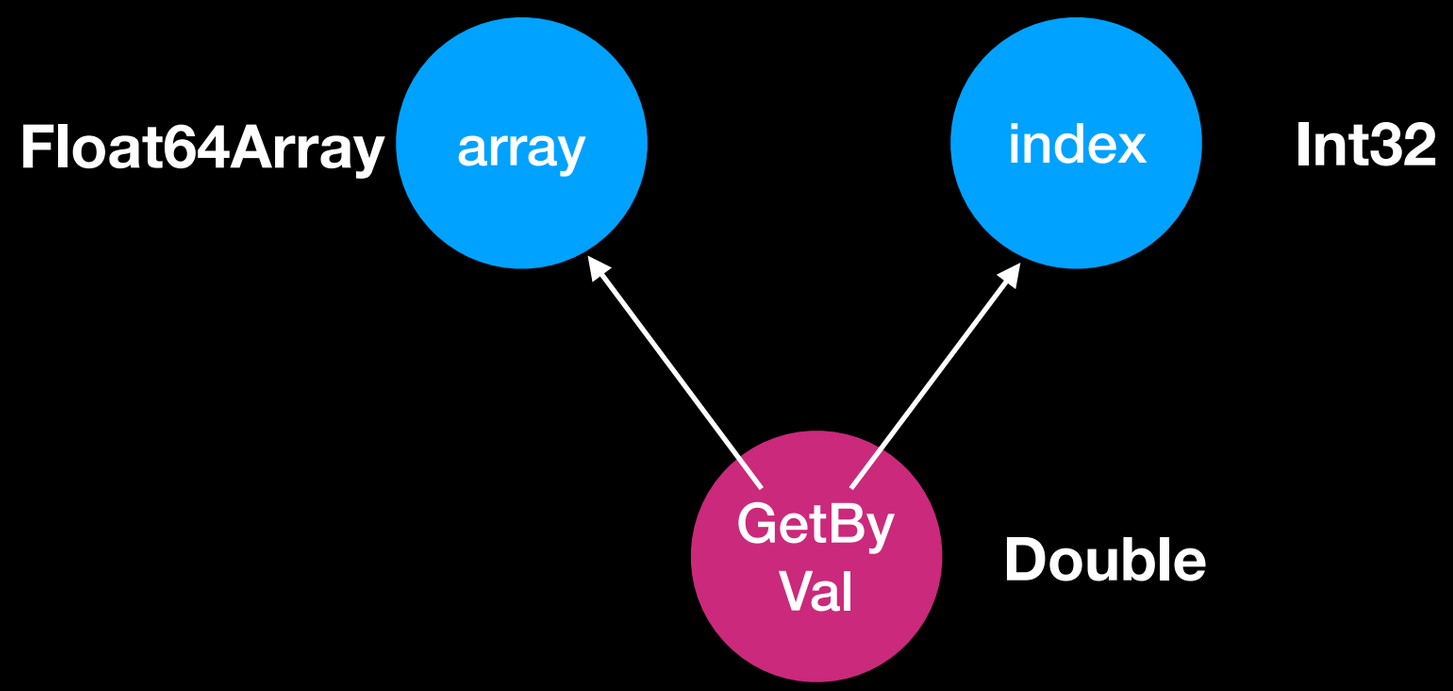
Prediction Propagation



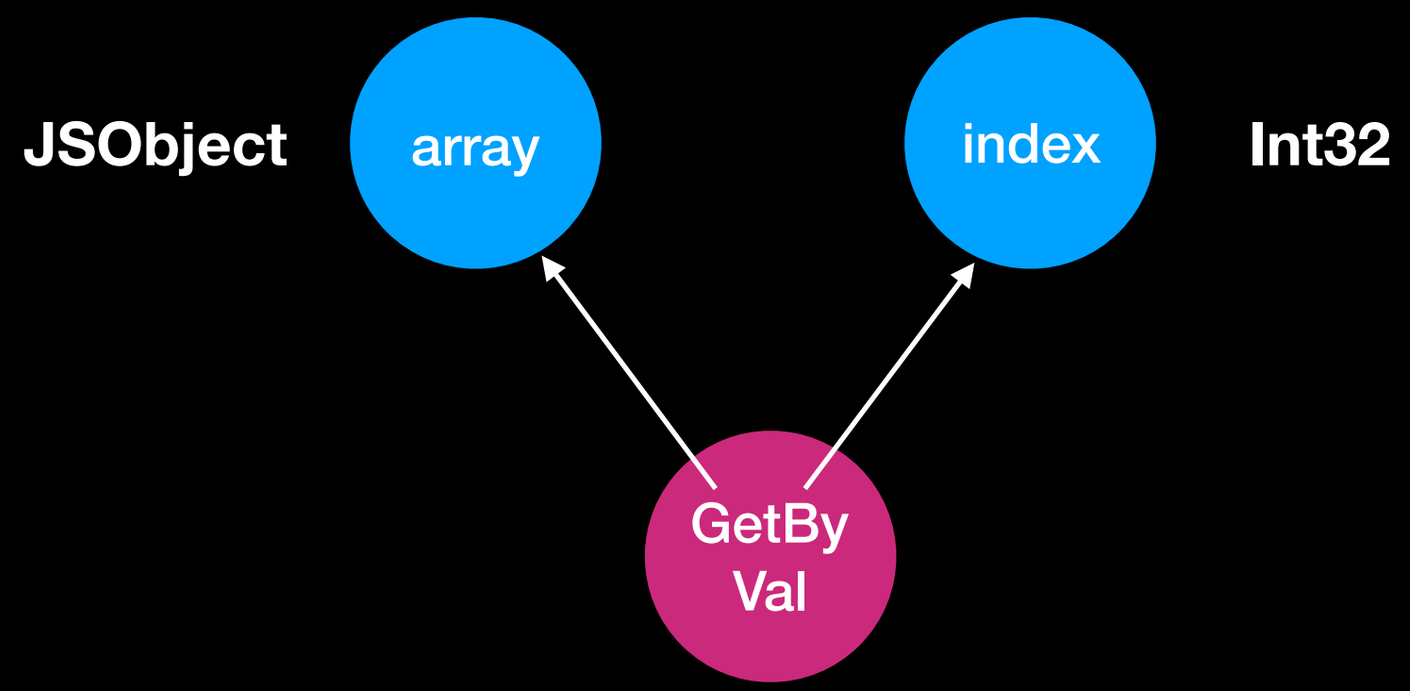
Prediction Propagation



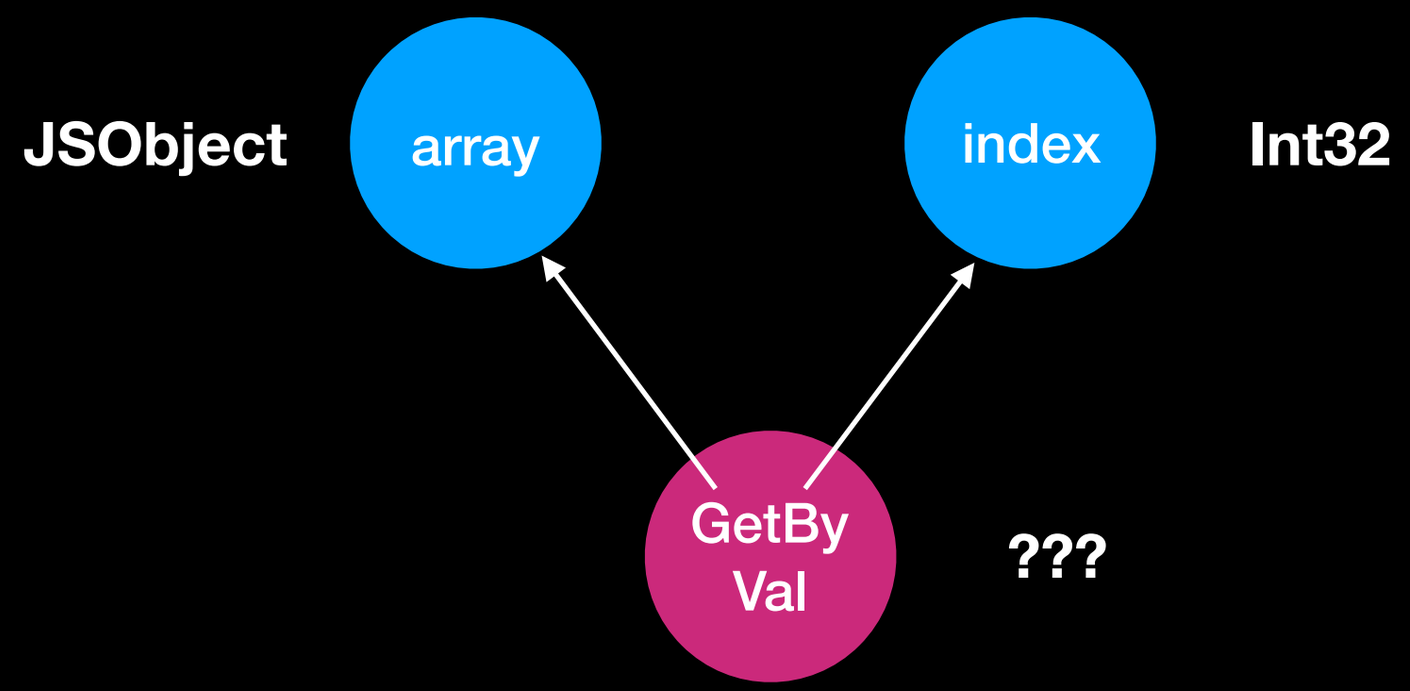
Prediction Propagation



Prediction Propagation



Prediction Propagation



Value Profiling Example

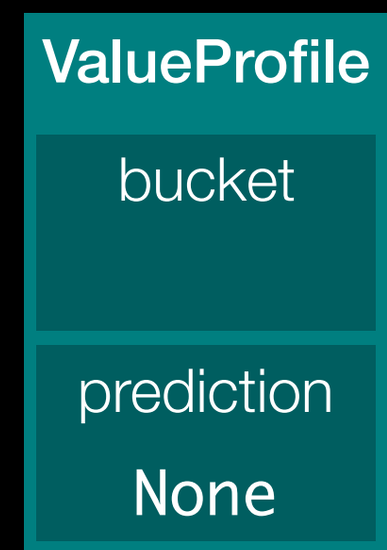
```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```

Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```

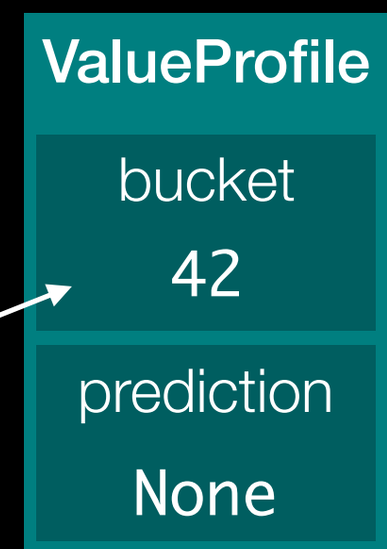
Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```



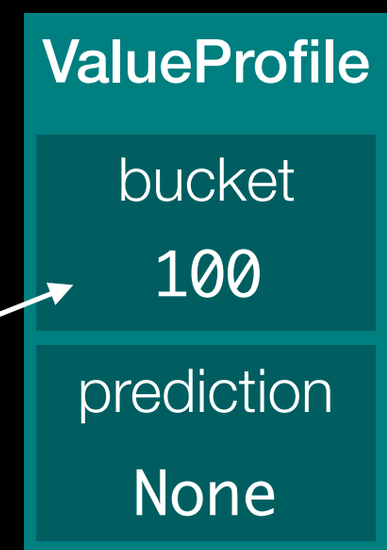
Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```



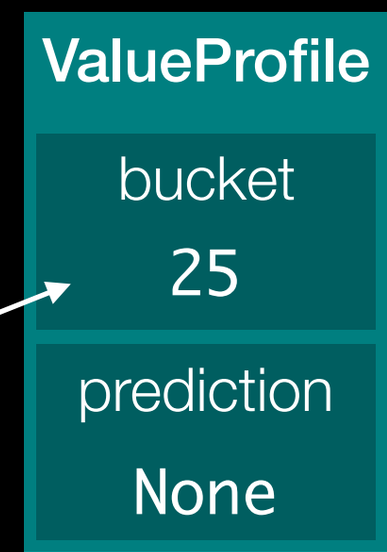
Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```



Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```



Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```

Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```

ValueProfile

bucket
7
prediction
None

```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```

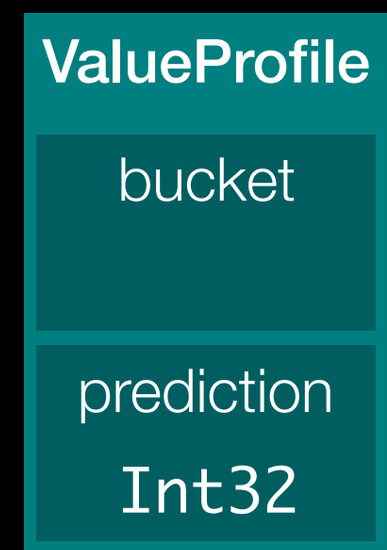
ValueProfile

bucket
7
prediction
Int32

```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling Example

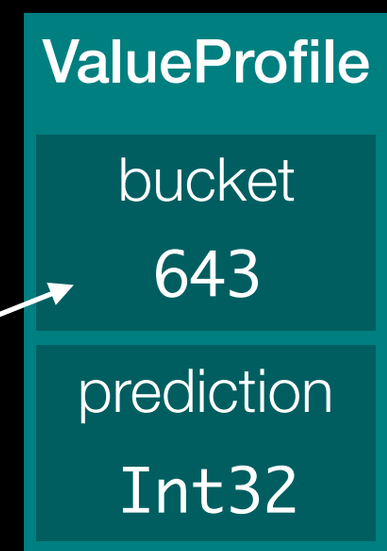
```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```




```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling Example

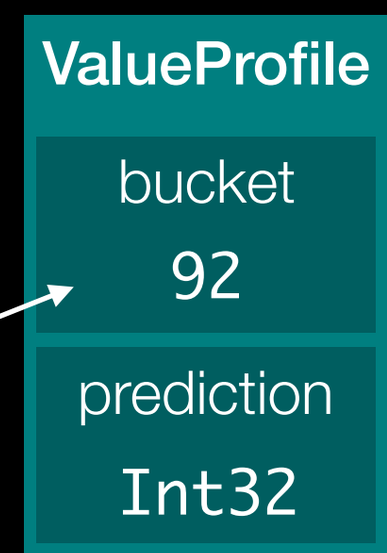
```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```




```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling Example

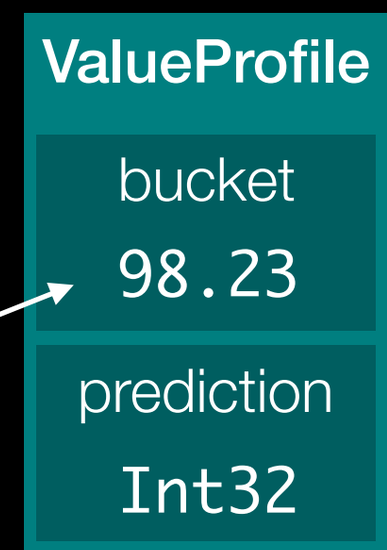
```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```




```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```




```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```

ValueProfile

bucket
98.23
prediction
Int32

```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```

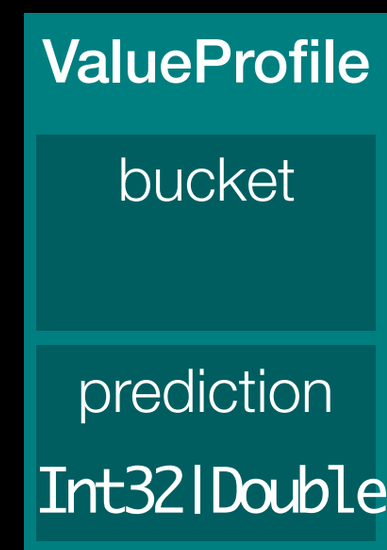
ValueProfile

bucket
98.23
prediction
Int32|Double

```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling Example

```
llintOpWithMetadata(
    op_get_by_val, OpGetByVal,
    macro (size, get, dispatch, metadata, return)
        macro finishGetByVal(result, scratch)
            get(dst, scratch)
            storeq result, [cfr, scratch, 8]
            valueProfile(OpGetByVal, t5, result)
            dispatch()
        end

        …
```




```
CodeBlock::updateAllPredictions()
  ValueProfile::computeUpdatedPrediction()
```

Value Profiling

- Combined with prediction propagation
- Provides *predicted* type inference

Speculated Types

FinalObject	Array	FunctionWithDefault HasInstance	FunctionWithNon DefaultHasInstance	Int8Array
Int16Array	Int32Array	Uint8Array	Uint8Clamped Array	Uint16Array
Uint32Array	Float32Array	Float64Array	DirectArguments	Scoped Arguments
StringObject	RegExpObject	MapObject	SetObject	WeakMapObject
WeakSetObject	ProxyObject	DerivedArray	ObjectOther	StringIdent
StringVar	Symbol	CellOther	BoolInt32	NonBoolInt32
Int52Only	AnyIntAsDouble	NonIntAsDouble	DoublePureNaN	Double ImpureNaN
Boolean	Other	Empty	BigInt	DataViewObject

Speculated Types

FinalObject	Array	FunctionWithDefault HasInstance	FunctionWithNon DefaultHasInstance	Int8Array
Int16Array	Int32Array	Uint8Array	Uint8Clamped Array	Uint16Array
Uint32Array	Float32Array	Float64Array	DirectArguments	Scoped Arguments
StringObject	RegExpObject	MapObject	SetObject	WeakMapObject
WeakSetObject	ProxyObject	DerivedArray	ObjectOther	StringIdent
StringVar	Symbol	CellOther	BoolInt32	NonBoolInt32
Int52Only	AnyIntAsDouble	NonIntAsDouble	DoublePureNaN	Double ImpureNaN
Boolean	Other	Empty	BigInt	DataViewObject

Speculated Types

FinalObject	Array	FunctionWithDefault HasInstance	FunctionWithNon DefaultHasInstance	Int8Array
Int16Array	Int32Array	Uint8Array	Uint8Clamped Array	Uint16Array
Uint32Array	Float32Array	Float64Array	DirectArguments	Scoped Arguments
StringObject	RegExpObject	MapObject	SetObject	WeakMapObject
WeakSetObject	ProxyObject	DerivedArray	ObjectOther	StringIdent
StringVar	Symbol	CellOther	BoolInt32	NonBoolInt32
Int52Only	AnyIntAsDouble	NonIntAsDouble	DoublePureNaN	Double ImpureNaN
Boolean	Other	Empty	BigInt	DataViewObject

Speculated Types

FinalObject	Array	FunctionWithDefault HasInstance	FunctionWithNon DefaultHasInstance	Int8Array
Int16Array	Int32Array	Uint8Array	Uint8Clamped Array	Uint16Array
Uint32Array	Float32Array	Float64Array	DirectArguments	Scoped Arguments
StringObject	RegExpObject	MapObject	SetObject	WeakMapObject
WeakSetObject	ProxyObject	DerivedArray	ObjectOther	StringIdent
StringVar	Symbol	CellOther	BoolInt32	NonBoolInt32
Int52Only	AnyIntAsDouble	NonIntAsDouble	DoublePureNaN	Double ImpureNaN
Boolean	Other	Empty	BigInt	DataViewObject

Speculated Types

FinalObject	Array	FunctionWithDefault HasInstance	FunctionWithNon DefaultHasInstance	Int8Array
Int16Array	Int32Array	Uint8Array	Uint8Clamped Array	Uint16Array
Uint32Array	Float32Array	Float64Array	DirectArguments	Scoped Arguments
StringObject	RegExpObject	MapObject	SetObject	WeakMapObject
WeakSetObject	ProxyObject	DerivedArray	ObjectOther	StringIdent
StringVar	Symbol	CellOther	BoolInt32	NonBoolInt32
Int52Only	AnyIntAsDouble	NonIntAsDouble	DoublePureNaN	Double ImpureNaN
Boolean	Other	Empty	BigInt	DataViewObject

Speculating On Speculated Type

```
CompareEq(Boolean:@left, Boolean:@right)
CompareEq(Int32:@left, Int32:@right)
CompareEq(Int32:BooleanToNumber(Boolean:@left), Int32:@right)
CompareEq(Int32:BooleanToNumber(Untyped:@left), Int32:@right)
CompareEq(Int32:@left, Int32:BooleanToNumber(Boolean:@right))
CompareEq(Int32:@left, Int32:BooleanToNumber(Untyped:@right))
CompareEq(Int52Rep:@left, Int52Rep:@right)
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Int52:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Number:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(NotCell:@right))
CompareEq(DoubleRep:DoubleRep(RealNumber:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:…, DoubleRep:…)
CompareEq(StringIdent:@left, StringIdent:@right)
CompareEq(String:@left, String:@right)
CompareEq(Symbol:@left, Symbol:@right)
CompareEq(Object:@left, Object:@right)
CompareEq(Other:@left, Untyped:@right)
CompareEq(Untyped:@left, Other:@right)
CompareEq(Object:@left, ObjectOrOther:@right)
CompareEq(ObjectOrOther:@left, Object:@right)
CompareEq(Untyped:@left, Untyped:@right)
```

Speculating On Speculated Type

```
CompareEq(Boolean:@left, Boolean:@right)
CompareEq(Int32:@left, Int32:@right)
CompareEq(Int32:BooleanToNumber(Boolean:@left), Int32:@right)
CompareEq(Int32:BooleanToNumber(Untyped:@left), Int32:@right)
CompareEq(Int32:@left, Int32:BooleanToNumber(Boolean:@right))
CompareEq(Int32:@left, Int32:BooleanToNumber(Untyped:@right))
CompareEq(Int52Rep:@left, Int52Rep:@right)
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Int52:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Number:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(NotCell:@right))
CompareEq(DoubleRep:DoubleRep(RealNumber:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:…, DoubleRep:…)
CompareEq(StringIdent:@left, StringIdent:@right)
CompareEq(String:@left, String:@right)
CompareEq(Symbol:@left, Symbol:@right)
CompareEq(Object:@left, Object:@right)
CompareEq(Other:@left, Untyped:@right)
CompareEq(Untyped:@left, Other:@right)
CompareEq(Object:@left, ObjectOrOther:@right)
CompareEq(ObjectOrOther:@left, Object:@right)
CompareEq(Untyped:@left, Untyped:@right)
```

Speculating On Speculated Type

```
CompareEq(Boolean:@left, Boolean:@right)
CompareEq(Int32:@left, Int32:@right)
CompareEq(Int32:BooleanToNumber(Boolean:@left), Int32:@right)
CompareEq(Int32:BooleanToNumber(Untyped:@left), Int32:@right)
CompareEq(Int32:@left, Int32:BooleanToNumber(Boolean:@right))
CompareEq(Int32:@left, Int32:BooleanToNumber(Untyped:@right))
CompareEq(Int52Rep:@left, Int52Rep:@right)
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Int52:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Number:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(NotCell:@right))
CompareEq(DoubleRep:DoubleRep(RealNumber:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:…, DoubleRep:…)
CompareEq(StringIdent:@left, StringIdent:@right)
CompareEq(String:@left, String:@right)
CompareEq(Symbol:@left, Symbol:@right)
CompareEq(Object:@left, Object:@right)
CompareEq(Other:@left, Untyped:@right)
CompareEq(Untyped:@left, Other:@right)
CompareEq(Object:@left, ObjectOrOther:@right)
CompareEq(ObjectOrOther:@left, Object:@right)
CompareEq(Untyped:@left, Untyped:@right)
```

Speculating On Speculated Type

```
CompareEq(Boolean:@left, Boolean:@right)
CompareEq(Int32:@left, Int32:@right)
CompareEq(Int32:BooleanToNumber(Boolean:@left), Int32:@right)
CompareEq(Int32:BooleanToNumber(Untyped:@left), Int32:@right)
CompareEq(Int32:@left, Int32:BooleanToNumber(Boolean:@right))
CompareEq(Int32:@left, Int32:BooleanToNumber(Untyped:@right))
CompareEq(Int52Rep:@left, Int52Rep:@right)
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Int52:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Number:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(NotCell:@right))
CompareEq(DoubleRep:DoubleRep(RealNumber:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:…, DoubleRep:…)
CompareEq(StringIdent:@left, StringIdent:@right)
CompareEq(String:@left, String:@right)
CompareEq(Symbol:@left, Symbol:@right)
CompareEq(Object:@left, Object:@right)
CompareEq(Other:@left, Untyped:@right)
CompareEq(Untyped:@left, Other:@right)
CompareEq(Object:@left, ObjectOrOther:@right)
CompareEq(ObjectOrOther:@left, Object:@right)
CompareEq(Untyped:@left, Untyped:@right)
```

Speculating On Speculated Type

```
CompareEq(Boolean:@left, Boolean:@right)
CompareEq(Int32:@left, Int32:@right)
CompareEq(Int32:BooleanToNumber(Boolean:@left), Int32:@right)
CompareEq(Int32:BooleanToNumber(Untyped:@left), Int32:@right)
CompareEq(Int32:@left, Int32:BooleanToNumber(Boolean:@right))
CompareEq(Int32:@left, Int32:BooleanToNumber(Untyped:@right))
CompareEq(Int52Rep:@left, Int52Rep:@right)
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Int52:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Number:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(NotCell:@right))
CompareEq(DoubleRep:DoubleRep(RealNumber:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:…, DoubleRep:…)
CompareEq(StringIdent:@left, StringIdent:@right)
CompareEq(String:@left, String:@right)
CompareEq(Symbol:@left, Symbol:@right)
CompareEq(Object:@left, Object:@right)
CompareEq(Other:@left, Untyped:@right)
CompareEq(Untyped:@left, Other:@right)
CompareEq(Object:@left, ObjectOrOther:@right)
CompareEq(ObjectOrOther:@left, Object:@right)
CompareEq(Untyped:@left, Untyped:@right)
```

Speculating On Speculated Type

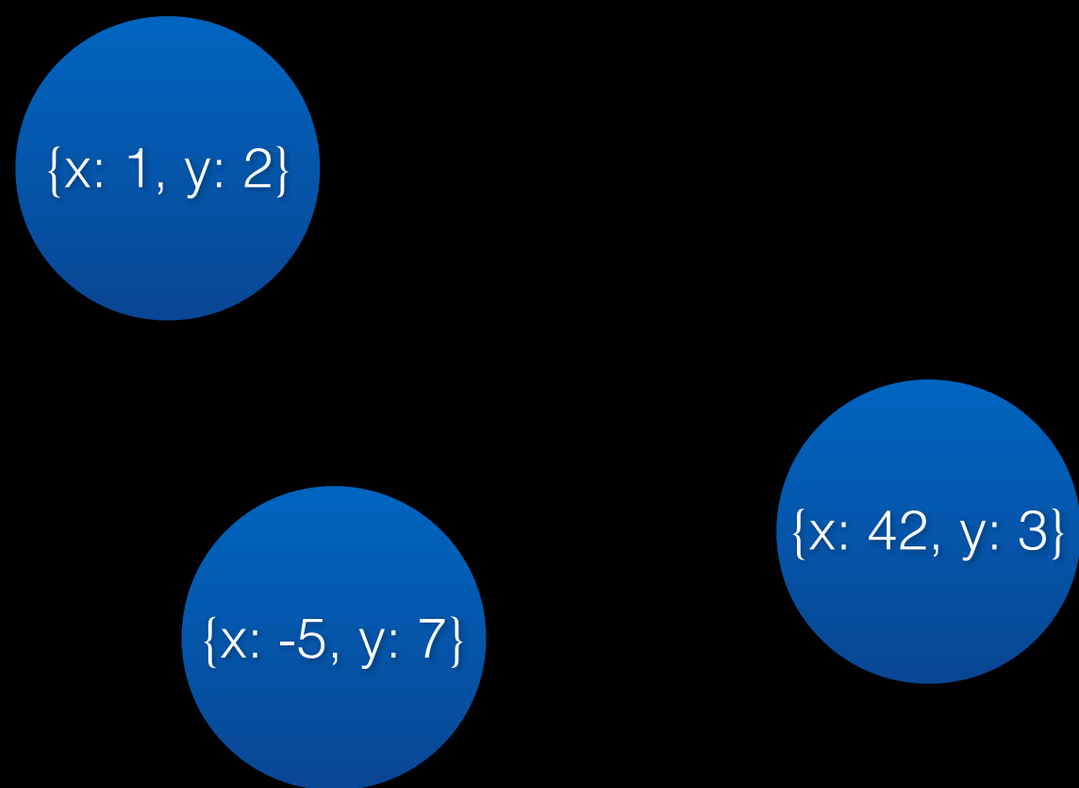
```
CompareEq(Boolean:@left, Boolean:@right)
CompareEq(Int32:@left, Int32:@right)
CompareEq(Int32:BooleanToNumber(Boolean:@left), Int32:@right)
CompareEq(Int32:BooleanToNumber(Untyped:@left), Int32:@right)
CompareEq(Int32:@left, Int32:BooleanToNumber(Boolean:@right))
CompareEq(Int32:@left, Int32:BooleanToNumber(Untyped:@right))
CompareEq(Int52Rep:@left, Int52Rep:@right)
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Int52:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Number:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(NotCell:@right))
CompareEq(DoubleRep:DoubleRep(RealNumber:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:…, DoubleRep:…)
CompareEq(StringIdent:@left, StringIdent:@right)
CompareEq(String:@left, String:@right)
CompareEq(Symbol:@left, Symbol:@right)
CompareEq(Object:@left, Object:@right)
CompareEq(Other:@left, Untyped:@right)
CompareEq(Untyped:@left, Other:@right)
CompareEq(Object:@left, ObjectOrOther:@right)
CompareEq(ObjectOrOther:@left, Object:@right)
CompareEq(Untyped:@left, Untyped:@right)
```

Speculating On Speculated Type

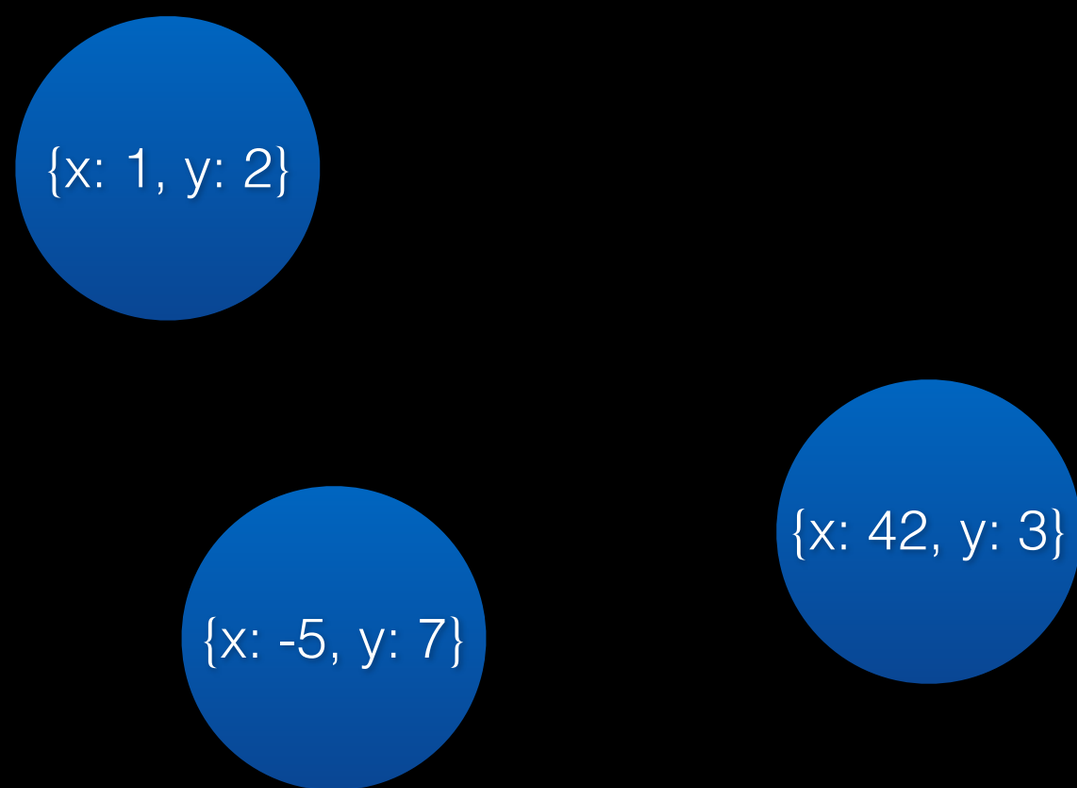
```
CompareEq(Boolean:@left, Boolean:@right)
CompareEq(Int32:@left, Int32:@right)
CompareEq(Int32:BooleanToNumber(Boolean:@left), Int32:@right)
CompareEq(Int32:BooleanToNumber(Untyped:@left), Int32:@right)
CompareEq(Int32:@left, Int32:BooleanToNumber(Boolean:@right))
CompareEq(Int32:@left, Int32:BooleanToNumber(Untyped:@right))
CompareEq(Int52Rep:@left, Int52Rep:@right)
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Int52:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(Number:@right))
CompareEq(DoubleRep:DoubleRep(Int52:@left), DoubleRep:DoubleRep(NotCell:@right))
CompareEq(DoubleRep:DoubleRep(RealNumber:@left), DoubleRep:DoubleRep(RealNumber:@right))
CompareEq(DoubleRep:…, DoubleRep:…)
CompareEq(StringIdent:@left, StringIdent:@right)
CompareEq(String:@left, String:@right)
CompareEq(Symbol:@left, Symbol:@right)
CompareEq(Object:@left, Object:@right)
CompareEq(Other:@left, Untyped:@right)
CompareEq(Untyped:@left, Other:@right)
CompareEq(Object:@left, ObjectOrOther:@right)
CompareEq(ObjectOrOther:@left, Object:@right)
CompareEq(Untyped:@left, Untyped:@right)
```

Profiling Sources in JSC

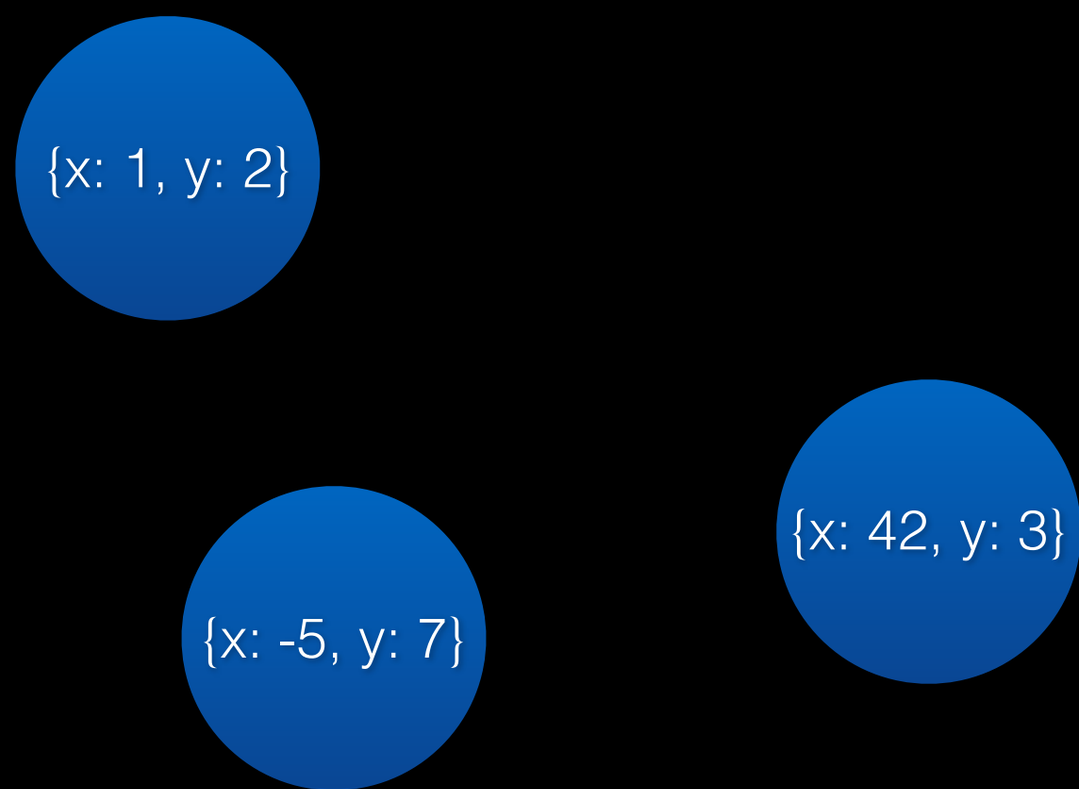
- Case Flags — *branch speculation*
- Case Counts — *branch speculation*
- Value Profiling — *type inference of values*
- Inline Caches — *type inference of object structure*
- Watchpoints — *heap speculation*
- Exit Flags — *speculation backoff*

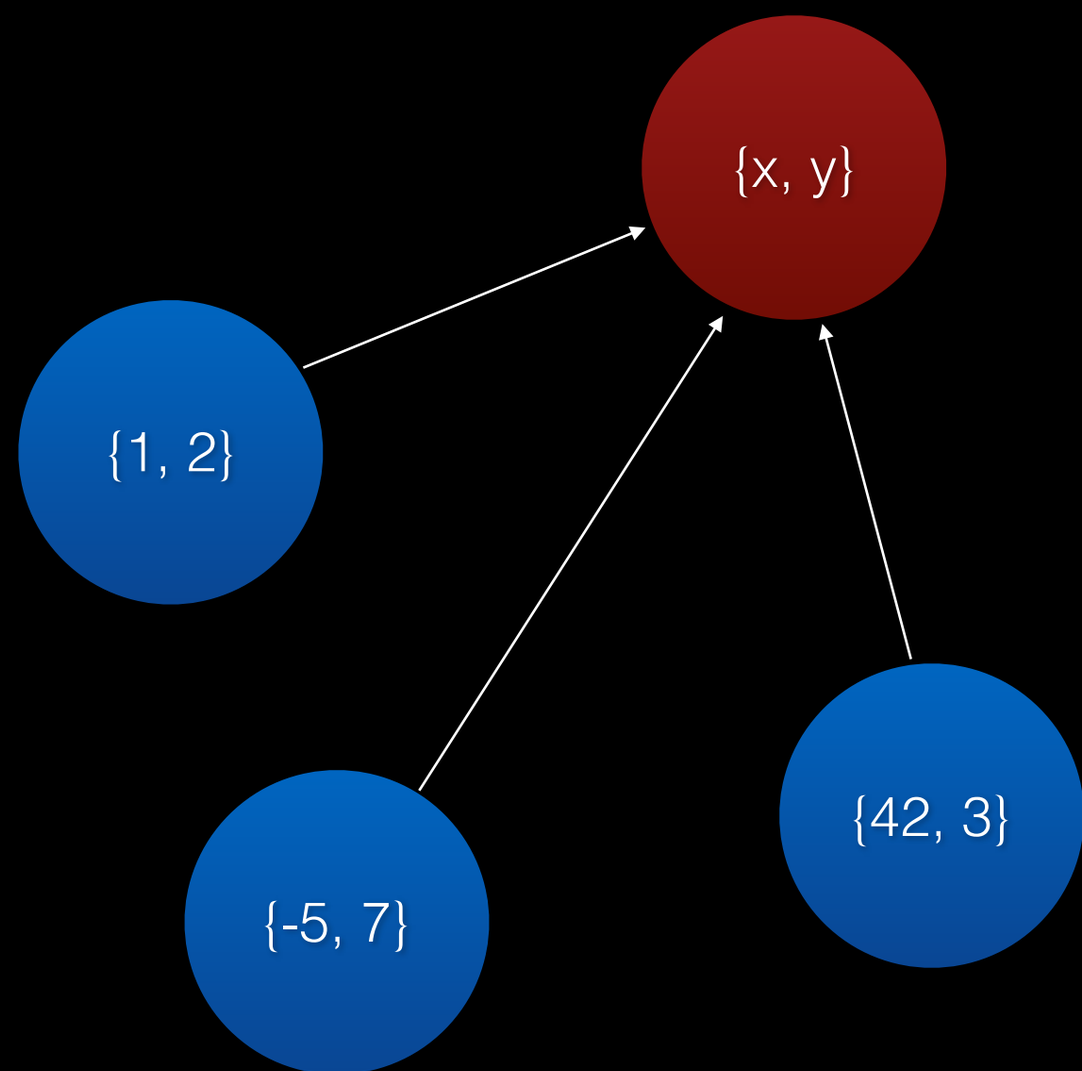

{x: 1, y: 2}
{x: -5, y: 7}
{x: 42, y: 3}

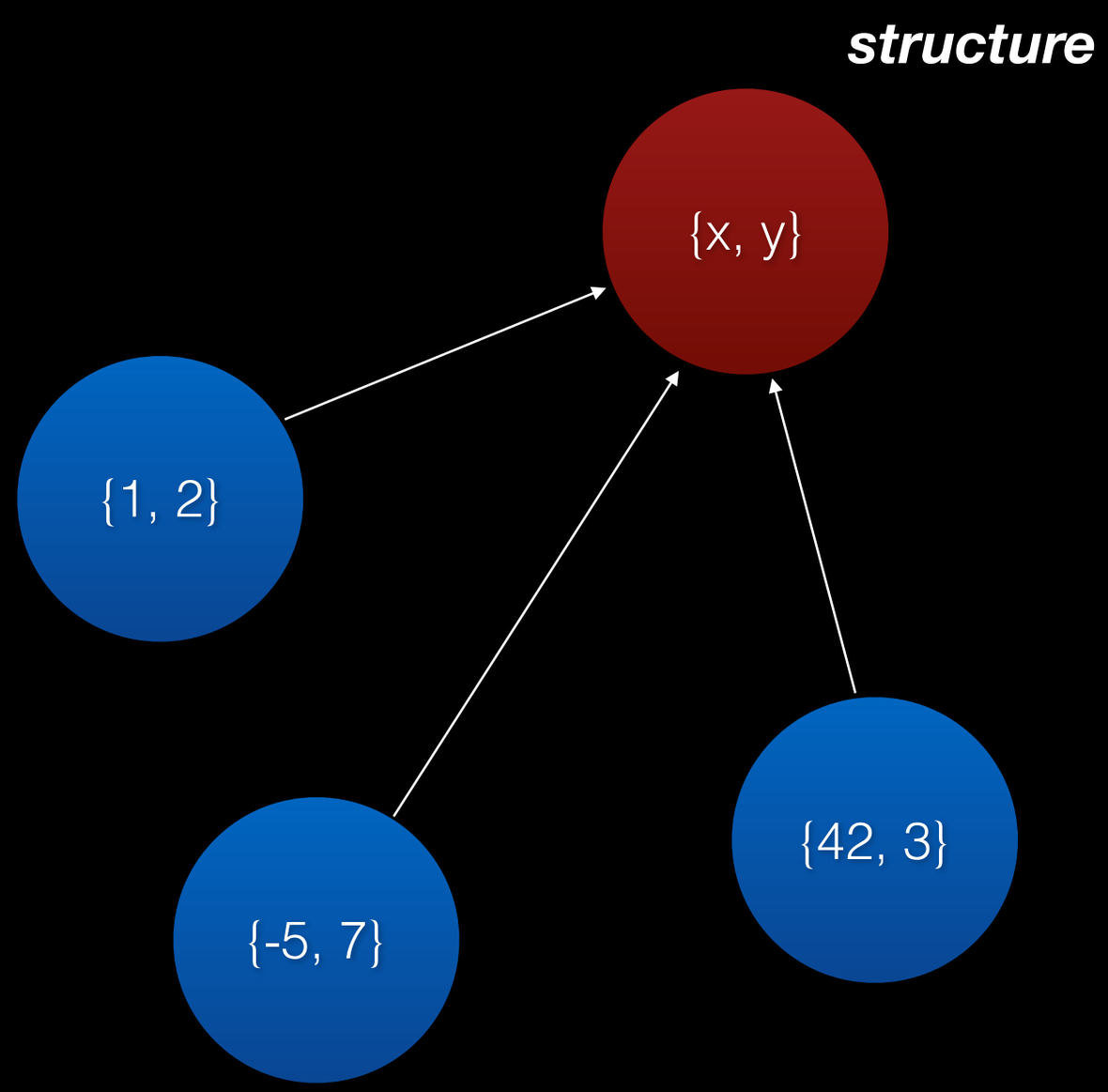
```
var x = o.x;
```



```
o.x = x;
```

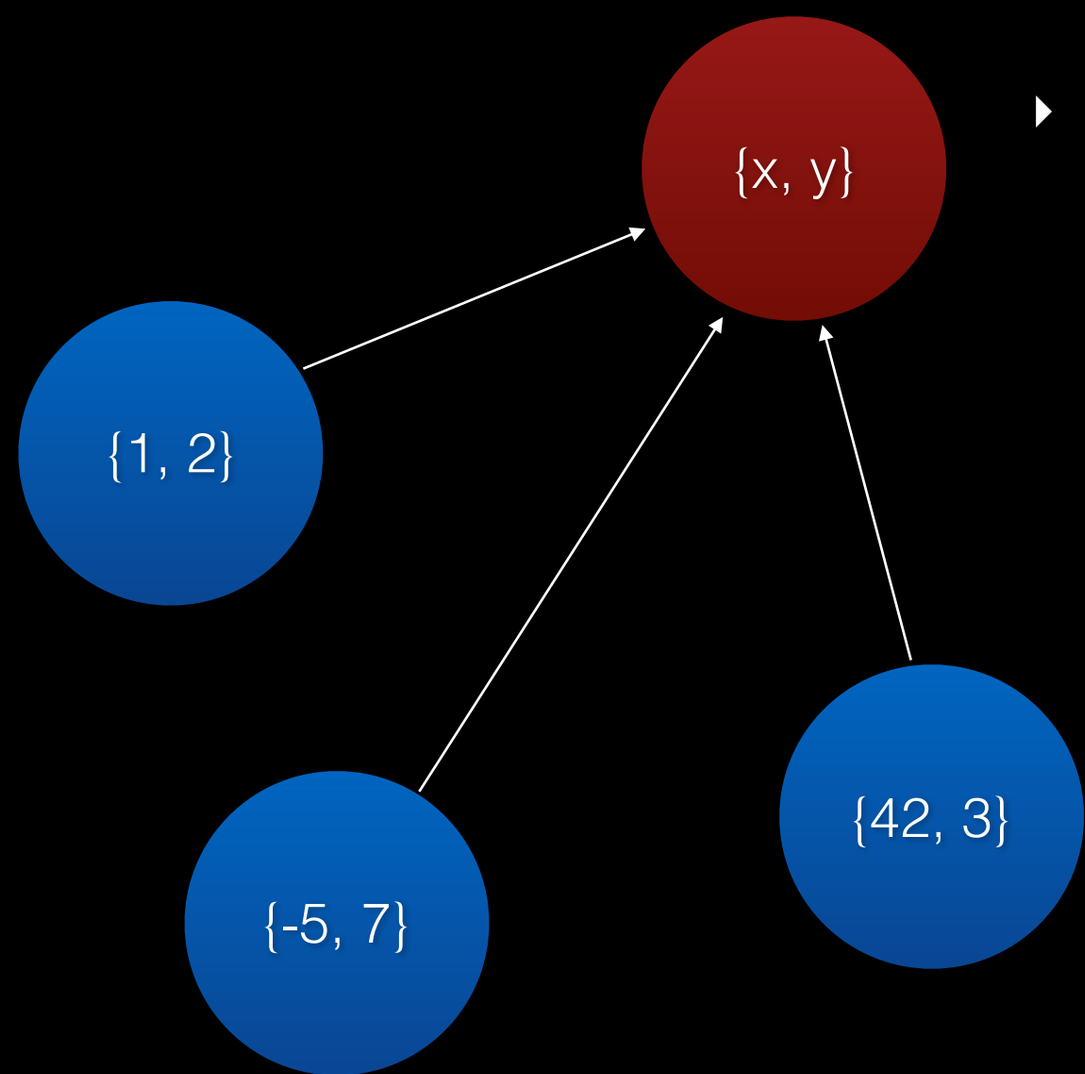


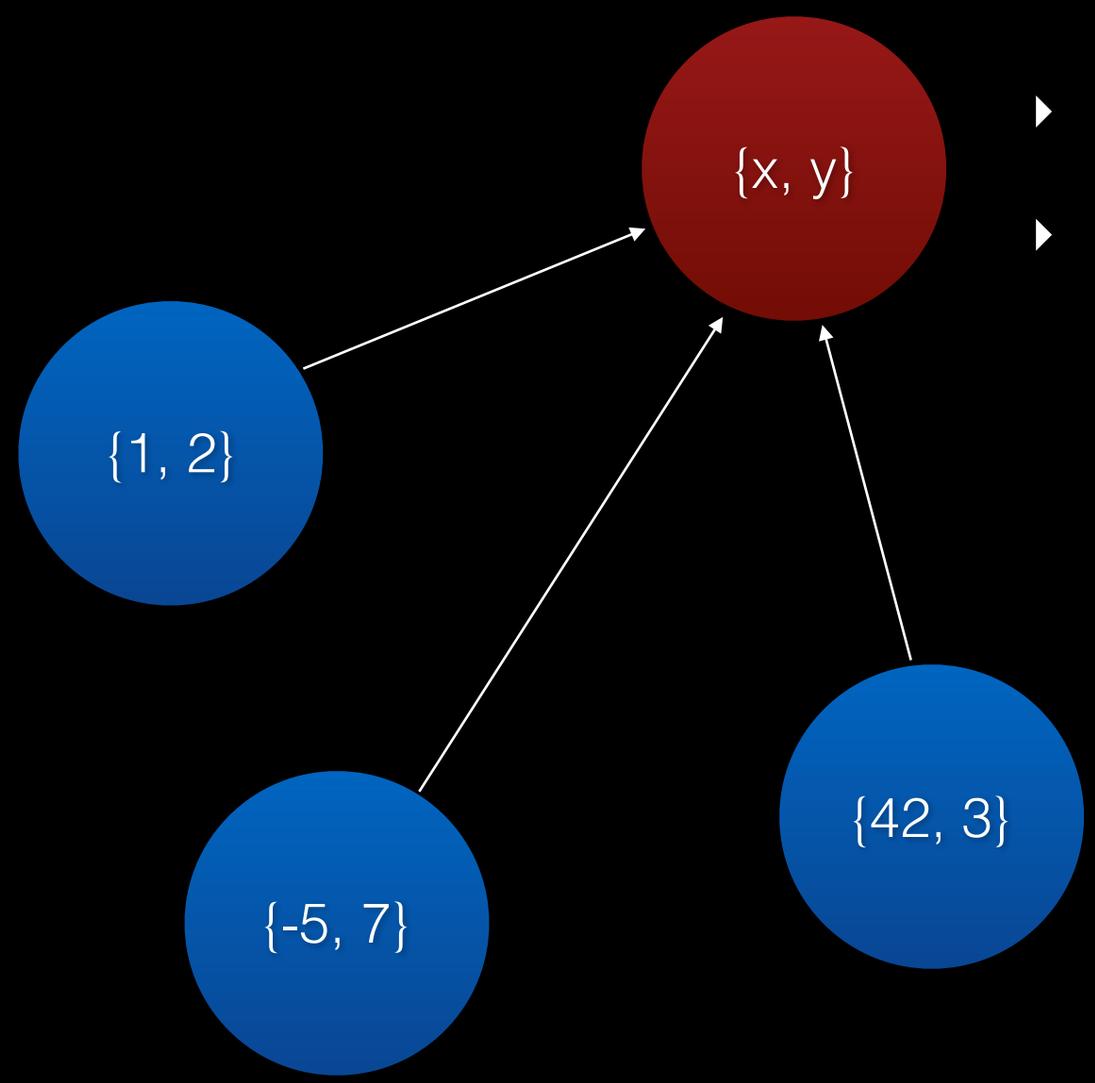

{x, y}
{1, 2}
{-5, 7}
{42, 3}


structure
{x, y}
{1, 2}
{-5, 7}
{42, 3}

structure

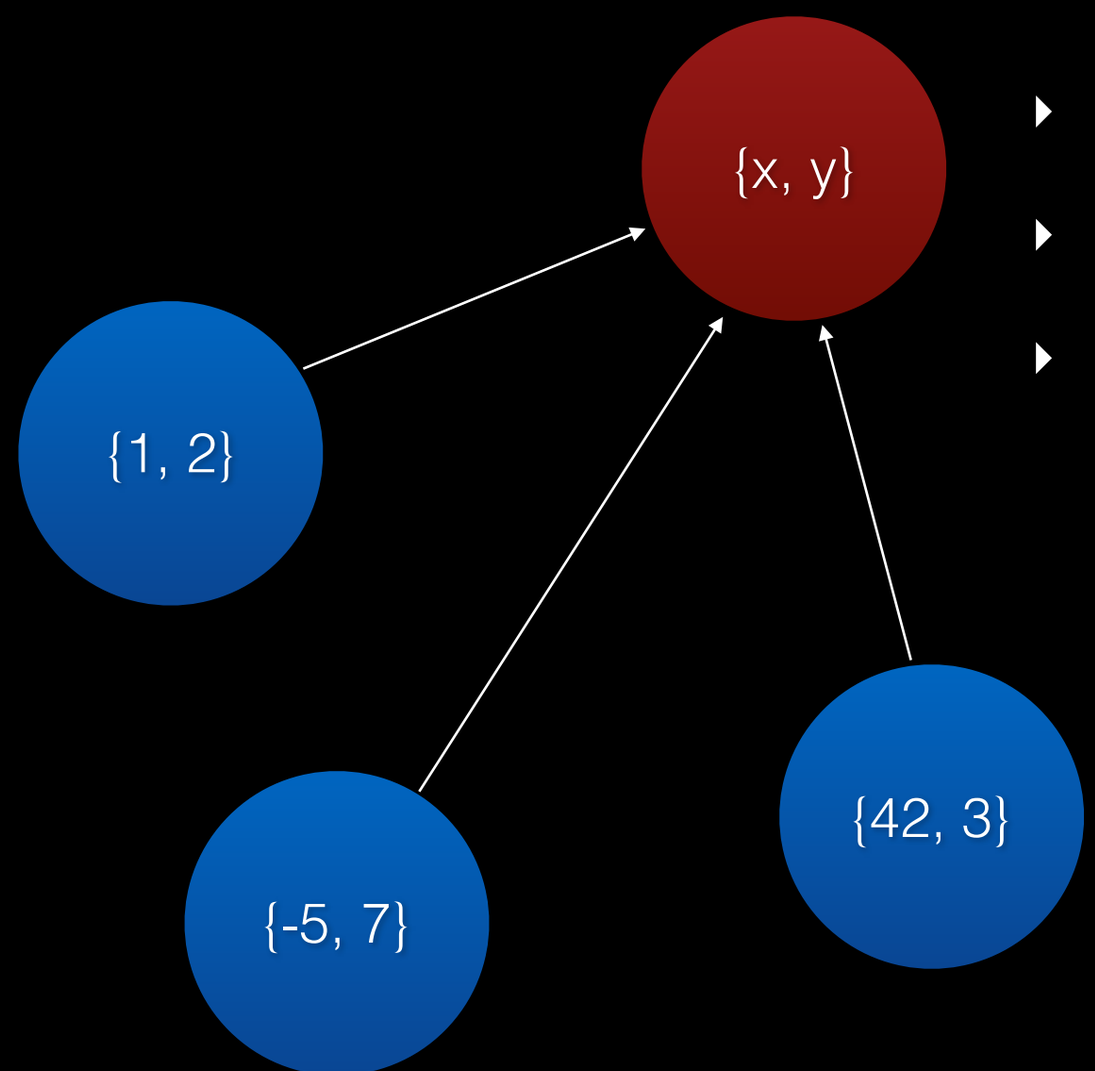
- hashtable





structure

- hashtable
- maps name to offset



structure

- hashtable
- maps name to offset
- hash-consed

Fast JSObject

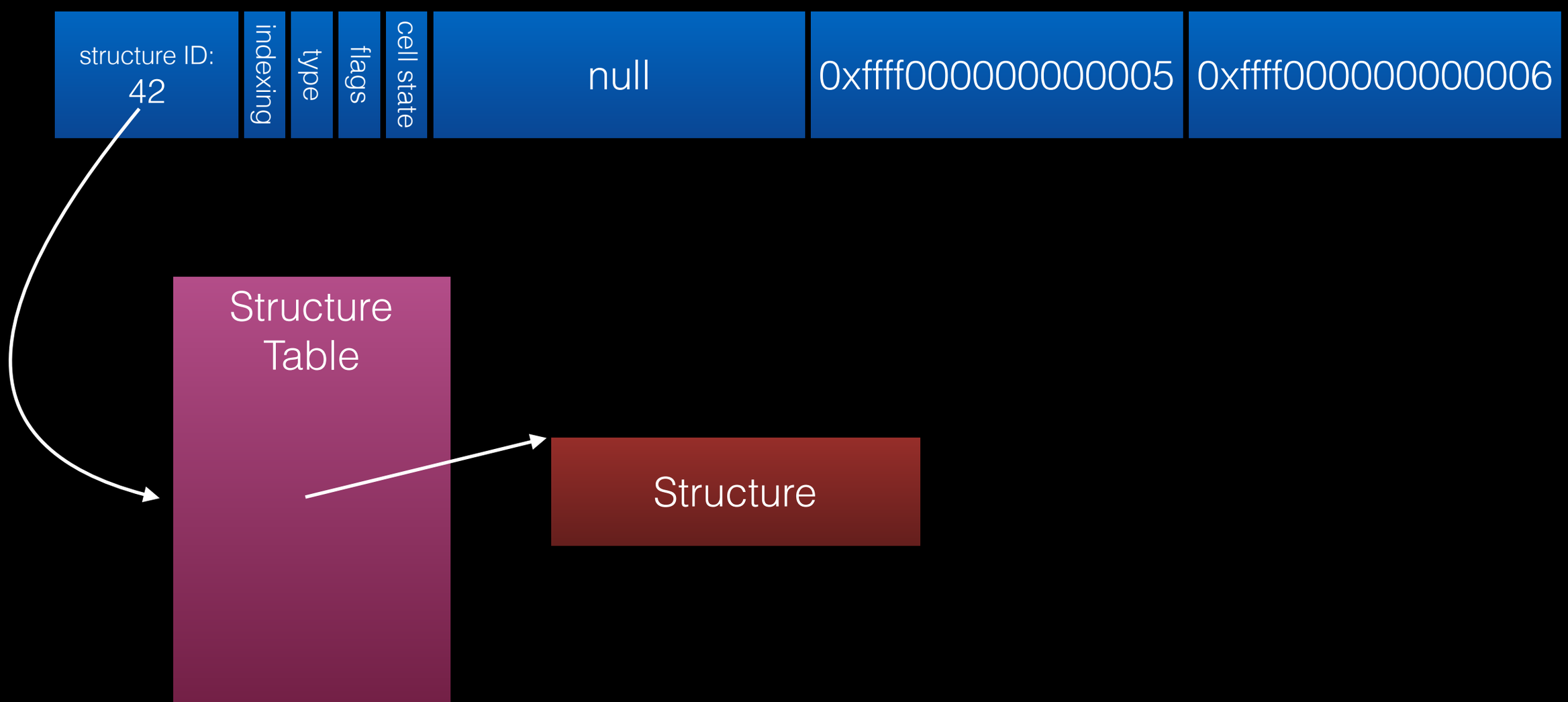
var o = {f: 5, g: 6};

Fast JSObject

var o = {f: 5, g: 6};

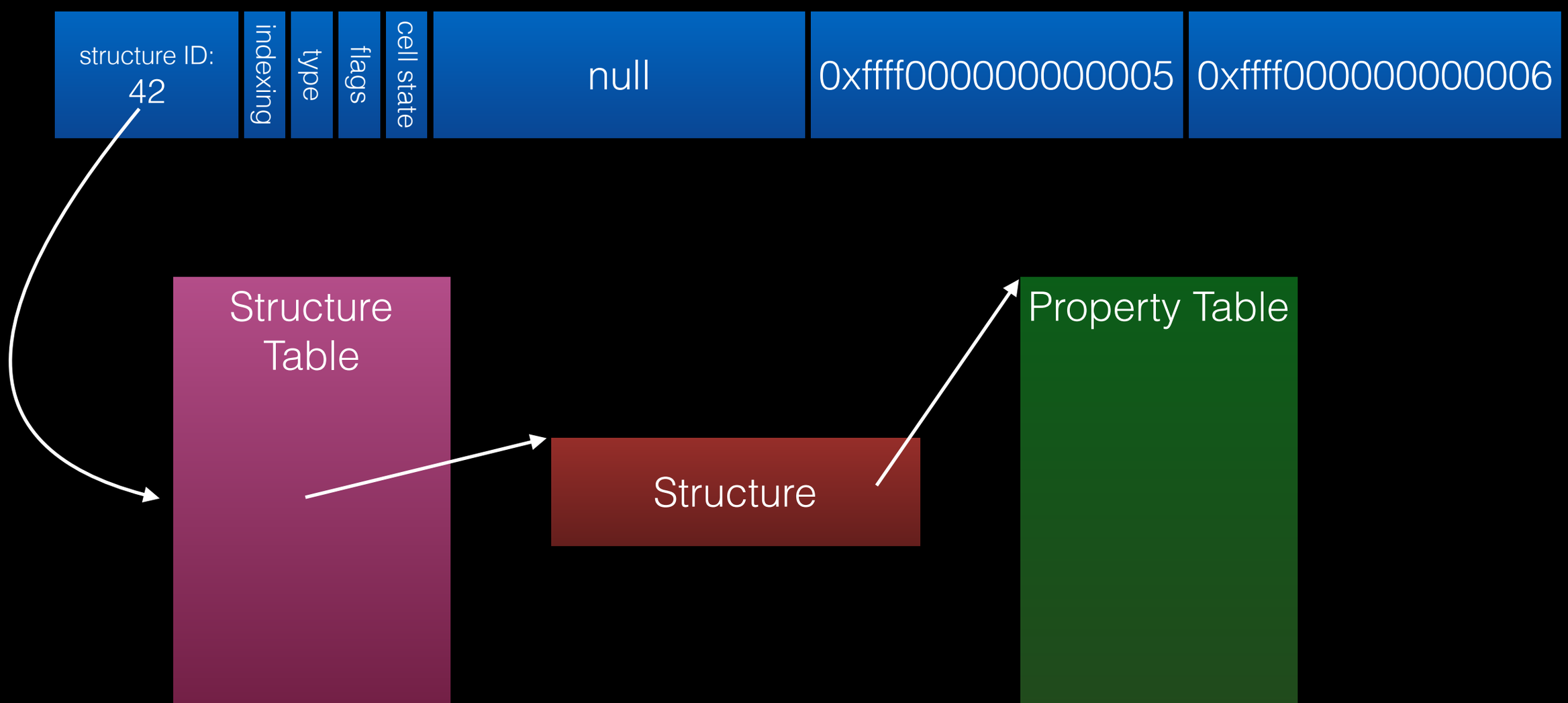
Fast JSObject

var o = {f: 5, g: 6};



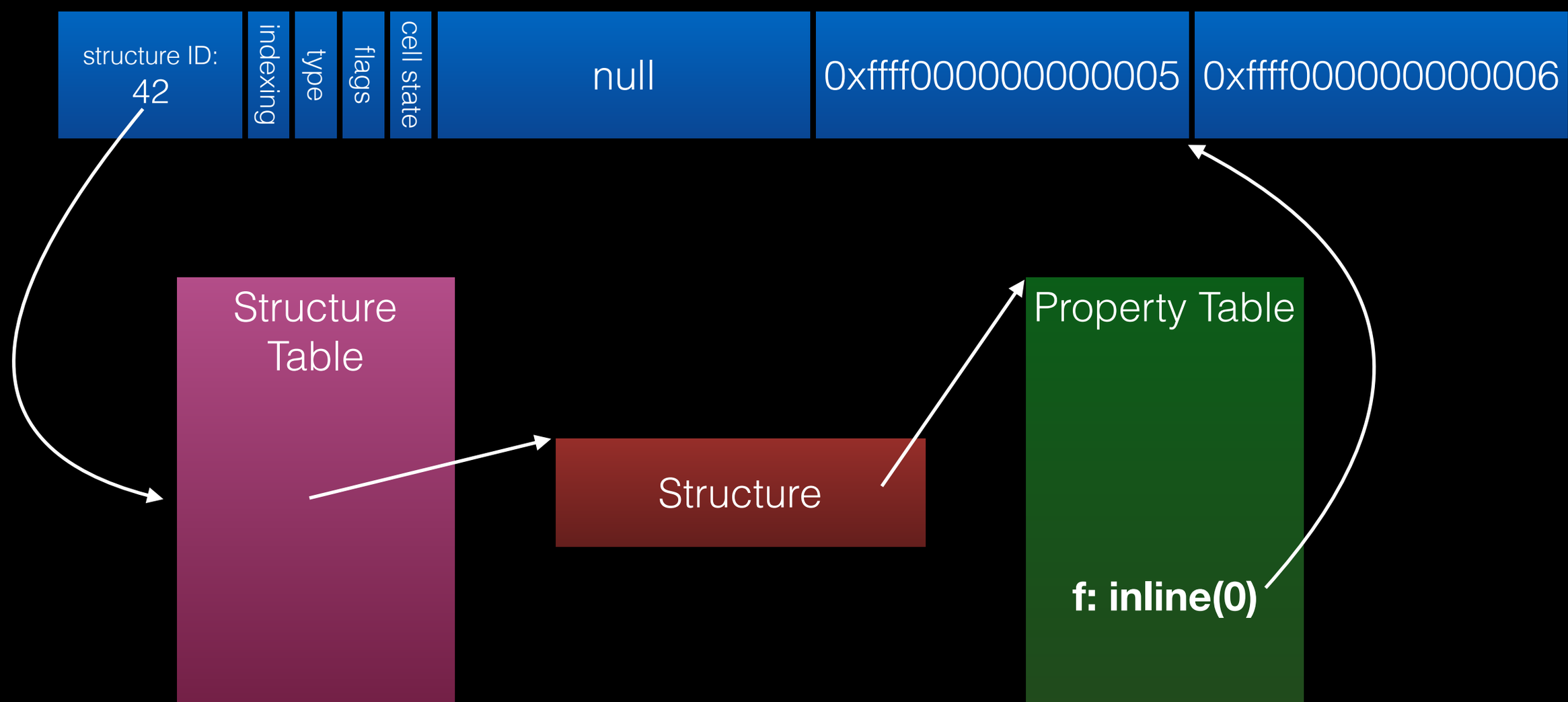
Fast JSObject

var o = {f: 5, g: 6};



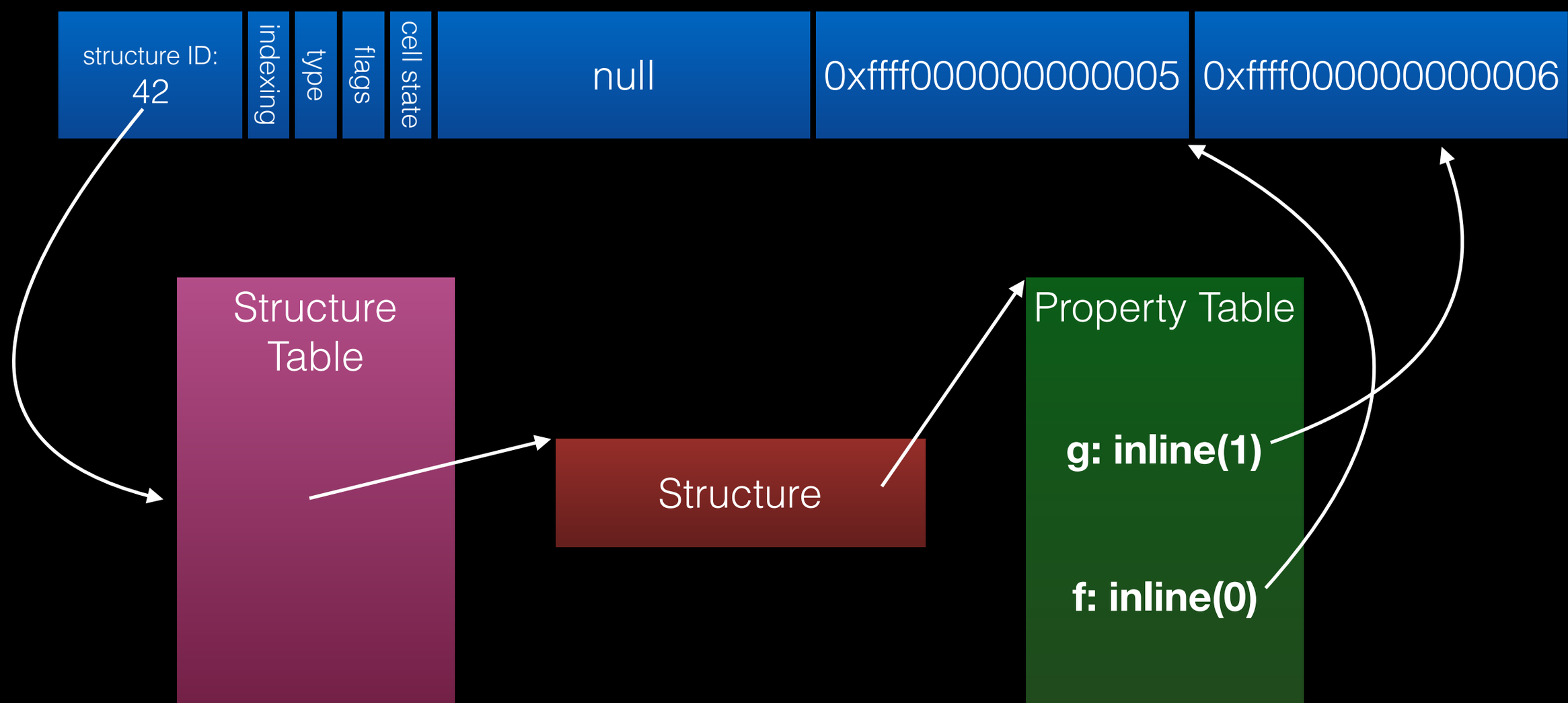
Fast JSObject

var o = {f: 5, g: 6};

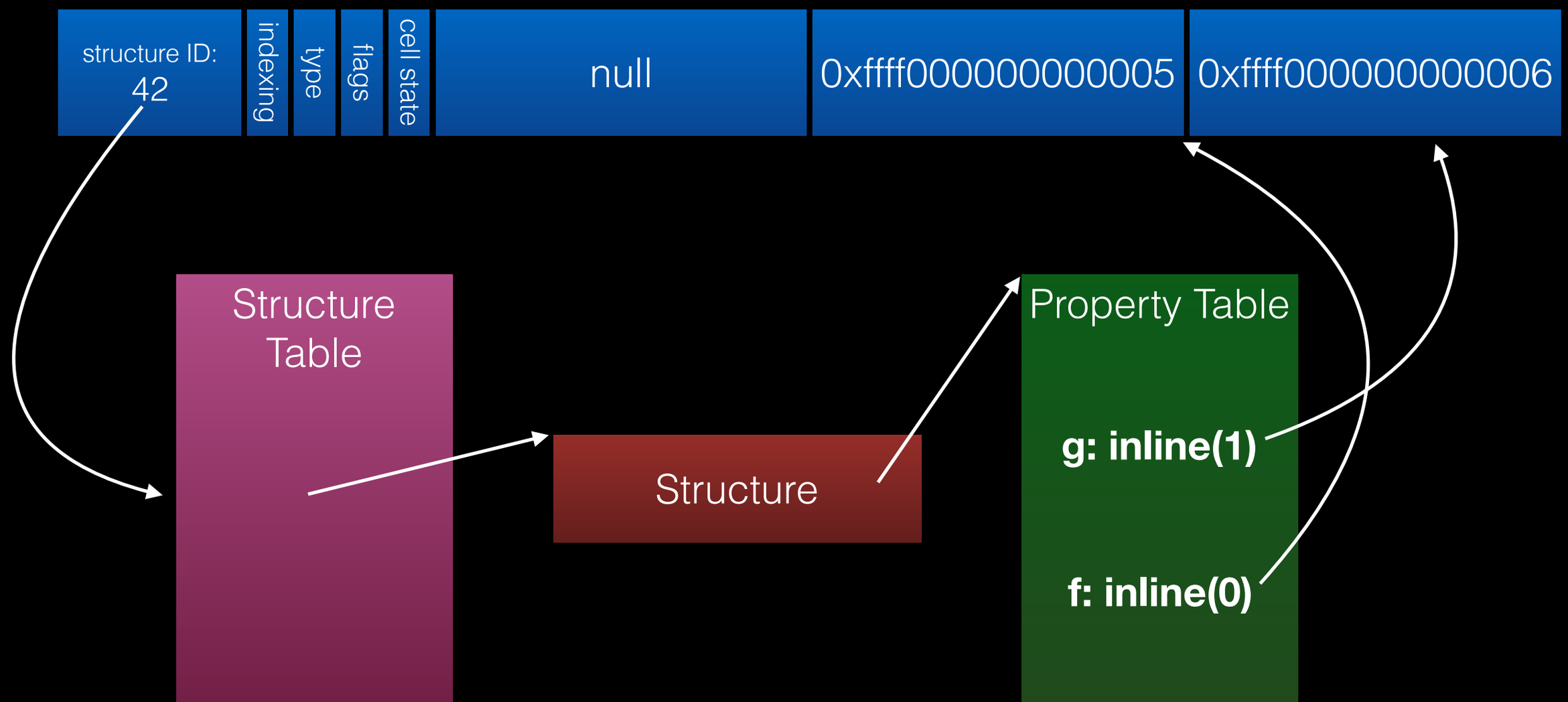


Fast JSObject

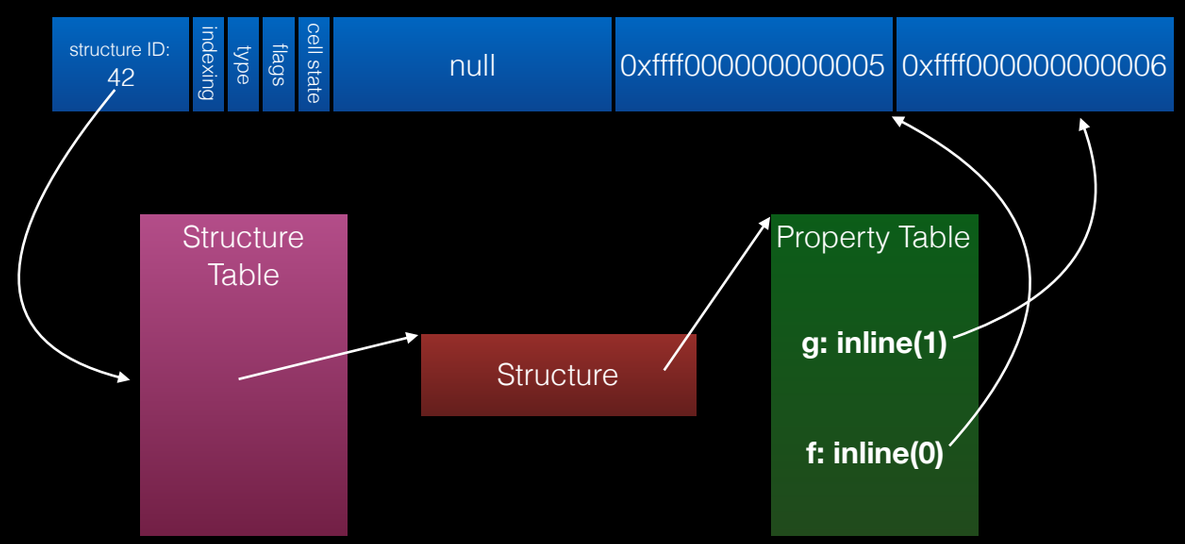
var o = {f: 5, g: 6};



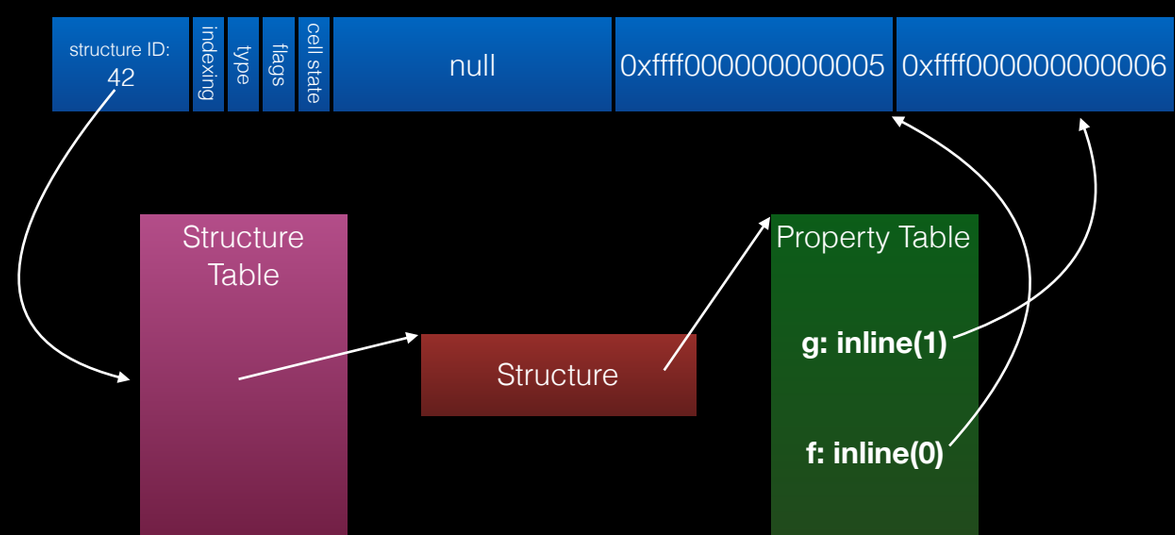
var o = {f: 5, g: 6};


structure ID:
42
indexing
type
flags
cell state
null
0xffff000000000005
0xffff000000000006
Structure Table
Structure
Property Table
g: inline(1)
f: inline(0)

var o = {f: 5, g: 6};

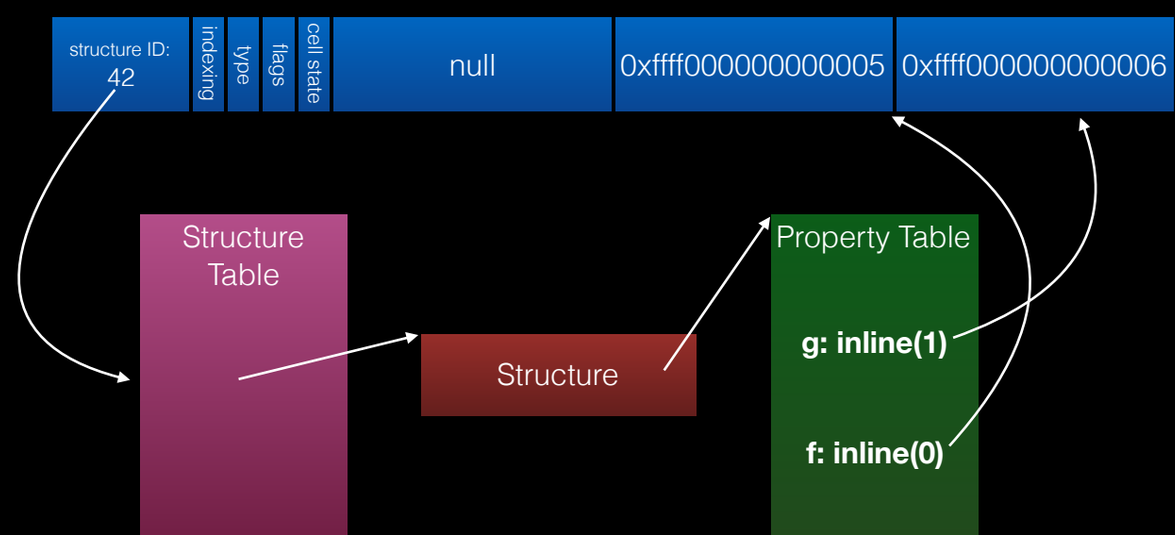

structure ID:
42
indexing
type
flags
cell state
null
0xffff000000000005
0xffff000000000006
Structure Table
Structure
Property Table
g: inline(1)
f: inline(0)

var o = {f: 5, g: 6};




```
var v = o.f;
```

var o = {f: 5, g: 6};

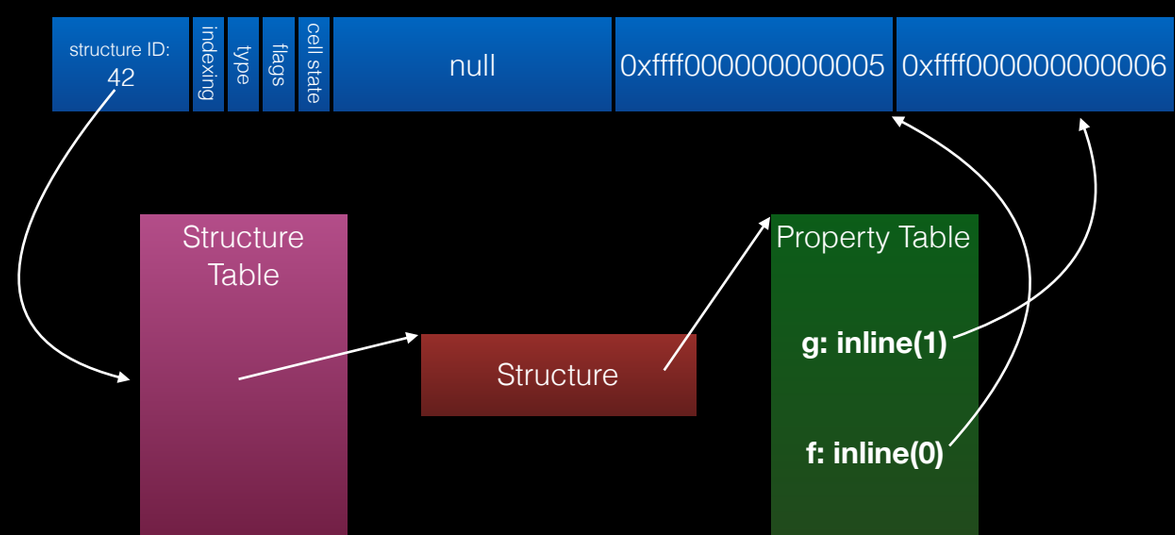



```
var v = o.f;
```

```
if (o->structureID == 42)
    v = o->inlineStorage[0]
else
    v = slowGet(o, “f”)
```

“Inline Cache”

var o = {f: 5, g: 6};




```
var v = o.f;
```

```
if (o->structureID == 42)
    v = o->inlineStorage[0]
else
    v = slowGet(o, “f”)
```

Interpreter Inline Cache

```
get_by_id <result>, <base>, <properyName>
```

Interpreter Inline Cache

```
get_by_id <result>, <base>, <properyName>,
    <cachedStructureID>, <cachedOffset>
```

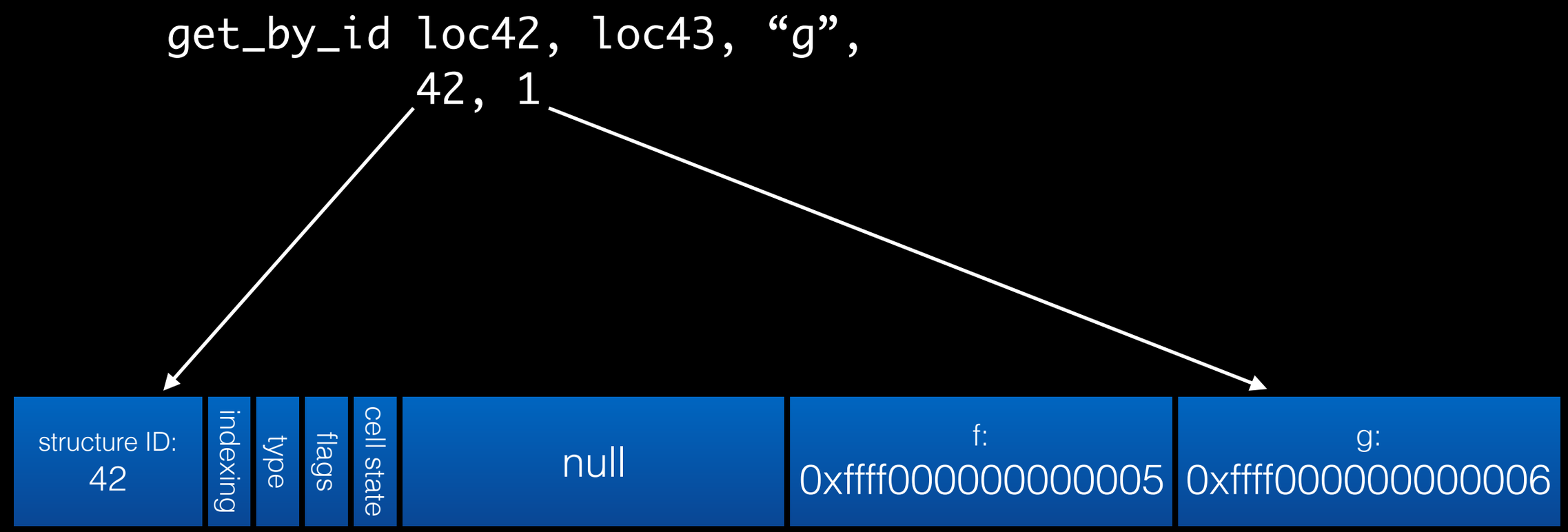
Interpreter Inline Cache

```
get_by_id loc42, loc43, “g”,
    0, 0
```

Interpreter Inline Cache

```
get_by_id loc42, loc43, “g”,
0, 0
```

Interpreter Inline Cache



JIT Inline Cache

```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: jmp 0x46f8c30ba2c
0x46f8c30b9c2: o16 nop %cs:0x200(%rax,%rax)
0x46f8c30b9d1: nop (%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

JIT Inline Cache

```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: jmp 0x46f8c30ba2c
0x46f8c30b9c2: o16 nop %cs:0x200(%rax,%rax)
0x46f8c30b9d1: nop (%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

JIT Inline Cache

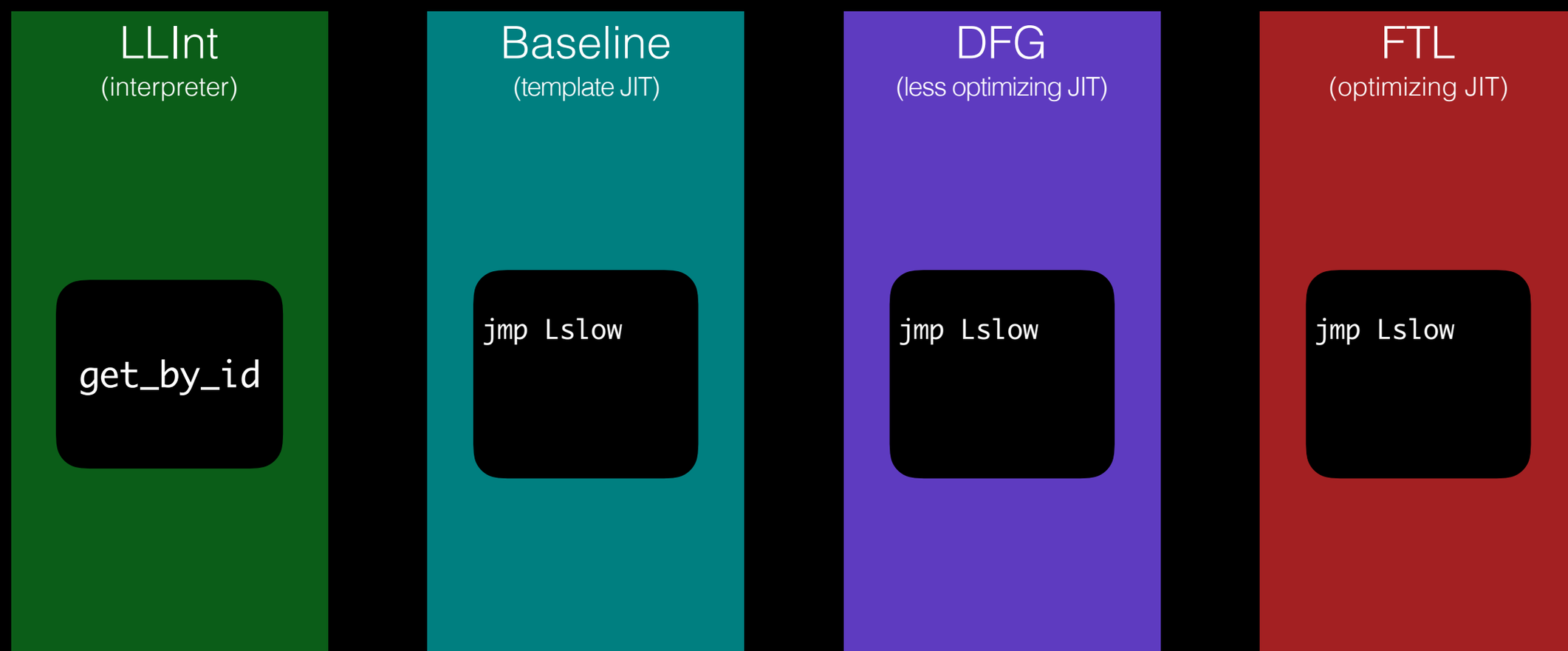
```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: jmp 0x46f8c30ba2c
0x46f8c30b9c2: o16 nop %cs:0x200(%rax,%rax)
0x46f8c30b9d1: nop (%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

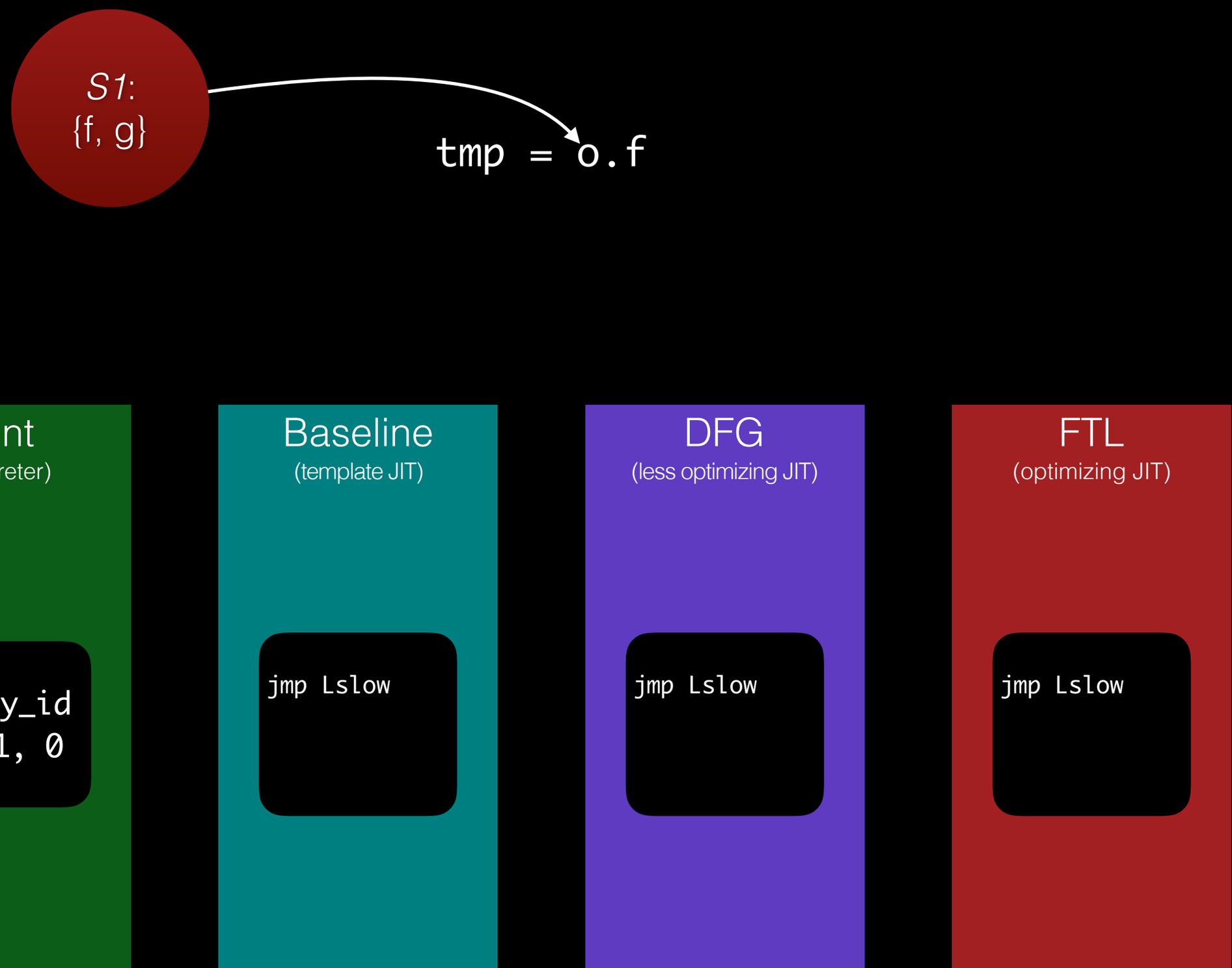
JIT Inline Cache

```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: cmp $0x125, (%rax)
0x46f8c30b9c3: jnz 0x46f8c30ba2c
0x46f8c30b9c9: mov 0x18(%rax), %rax
0x46f8c30b9cd: nop 0x200(%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

Inline caches implicitly collect profiling information.


S1:
{f, g}
tmp = o.f


LLInt
(interpreter)
get_by_id
Baseline
(template JIT)
jmp Lslow
DFG
(less optimizing JIT)
jmp Lslow
FTL
(optimizing JIT)
jmp Lslow


S1:
{f, g}
tmp = o.f
LLInt
(interpreter)
get_by_id
…, S1, 0
Baseline
(template JIT)
jmp Lslow
DFG
(less optimizing JIT)
jmp Lslow
FTL
(optimizing JIT)
jmp Lslow

LLInt
(interpreter)

```
get_by_id
…, S1, 0
```

Baseline
(template JIT)

```
cmp S1,
    (%rax)
jnz Lslow
mov 10(%rax),
    %rax
```

DFG
(less optimizing JIT)

```
jmp Lslow
```

FTL
(optimizing JIT)

```
jmp Lslow
```

LLInt
(interpreter)

```
get_by_id
…, S1, 0
```

Baseline
(template JIT)

```
cmp S1,
    (%rax)
jnz Lslow
mov 10(%rax),
    %rax
```

DFG
(less optimizing JIT)

```
cmp S1,
    (%rax)
jnz Lslow
mov 10(%rax),
    %rax
```

FTL
(optimizing JIT)

```
jmp Lslow
```

LLInt
(interpreter)

```
get_by_id
…, S1, 0
```

Baseline
(template JIT)

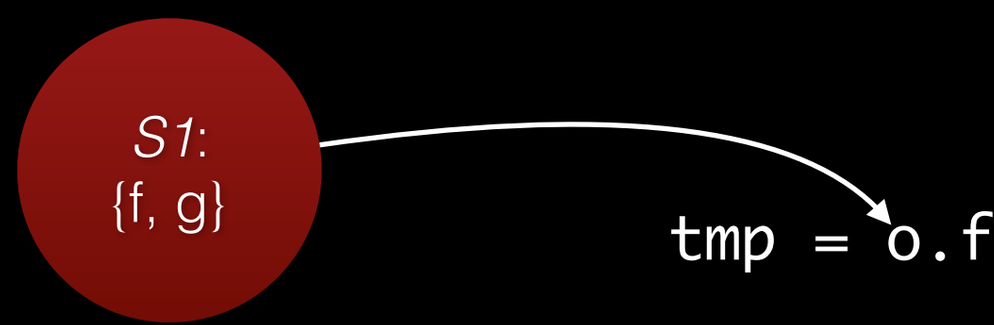
```
cmp S1,
    (%rax)
jnz Lslow
mov 10(%rax),
    %rax
```

DFG
(less optimizing JIT)

```
cmp S1,
    (%rax)
jnz Lslow
mov 10(%rax),
    %rax
```

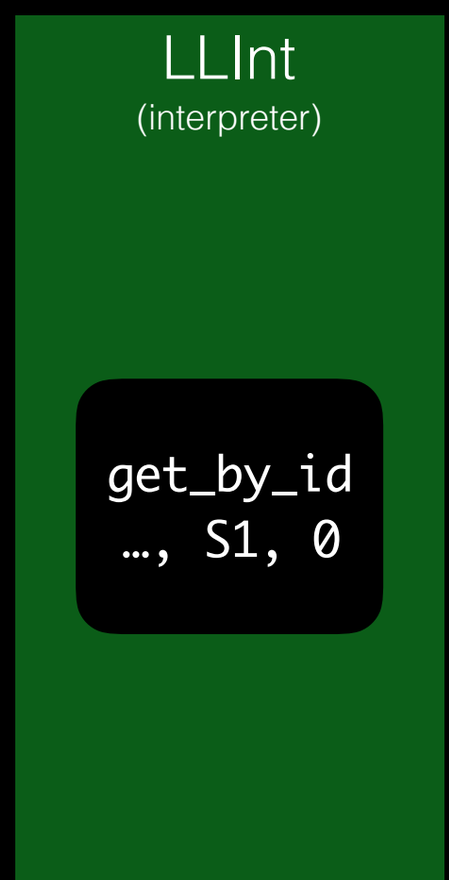
FTL
(optimizing JIT)

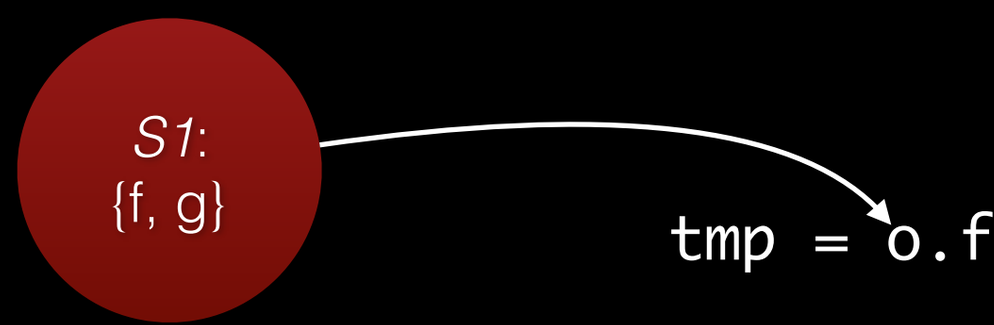
```
cmp S1,
    (%rax)
jnz Lslow
mov 10(%rax),
    %rax
```

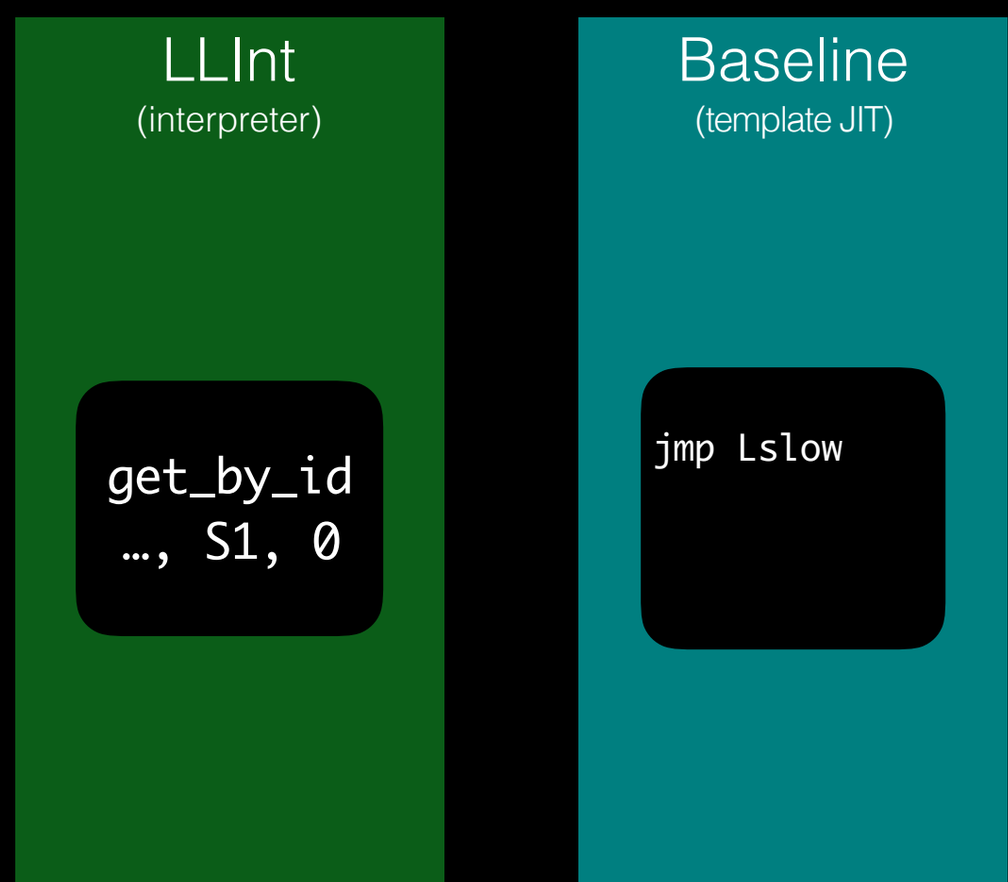

S1:
{f, g}
tmp = o.f


LLInt
(interpreter)
get_by_id

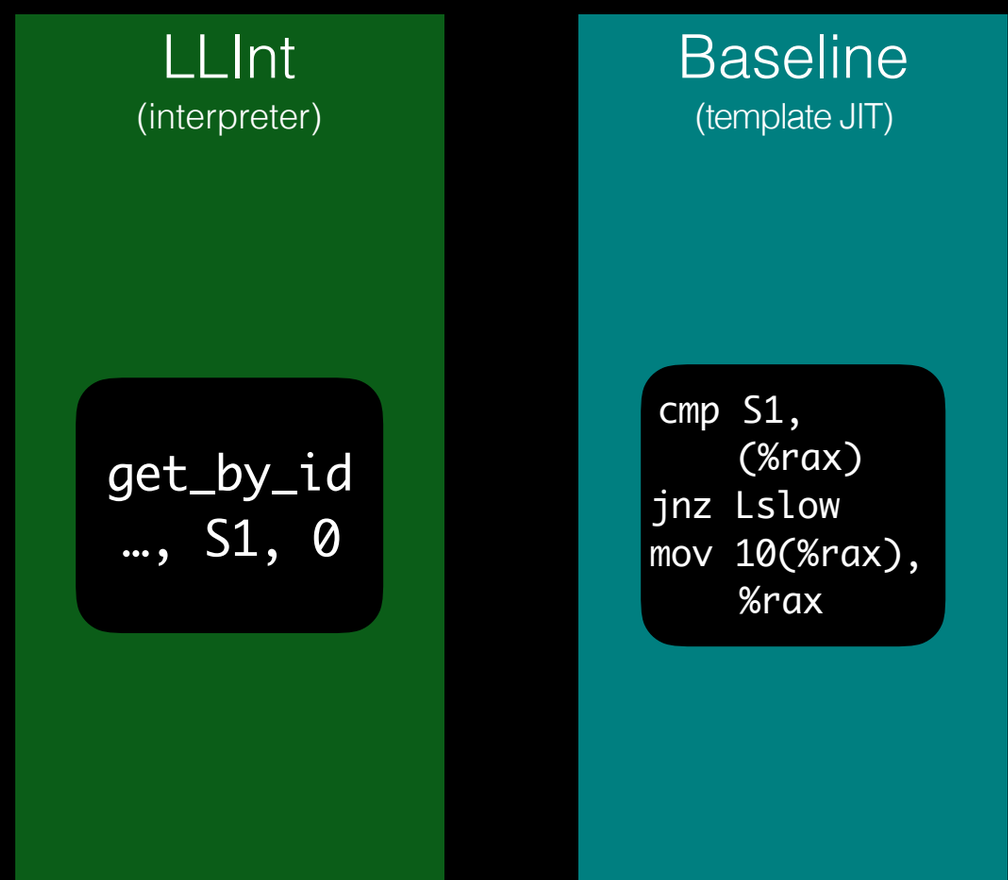

S1:
{f, g}
tmp = o.f


LLInt
(interpreter)
get_by_id
…, S1, 0


S1:
{f, g}
tmp = o.f


LLInt
(interpreter)
get_by_id
…, S1, 0
Baseline
(template JIT)
jmp Lslow


S1:
{f, g}
tmp = o.f


LLInt
(interpreter)
get_by_id
…, S1, 0
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax

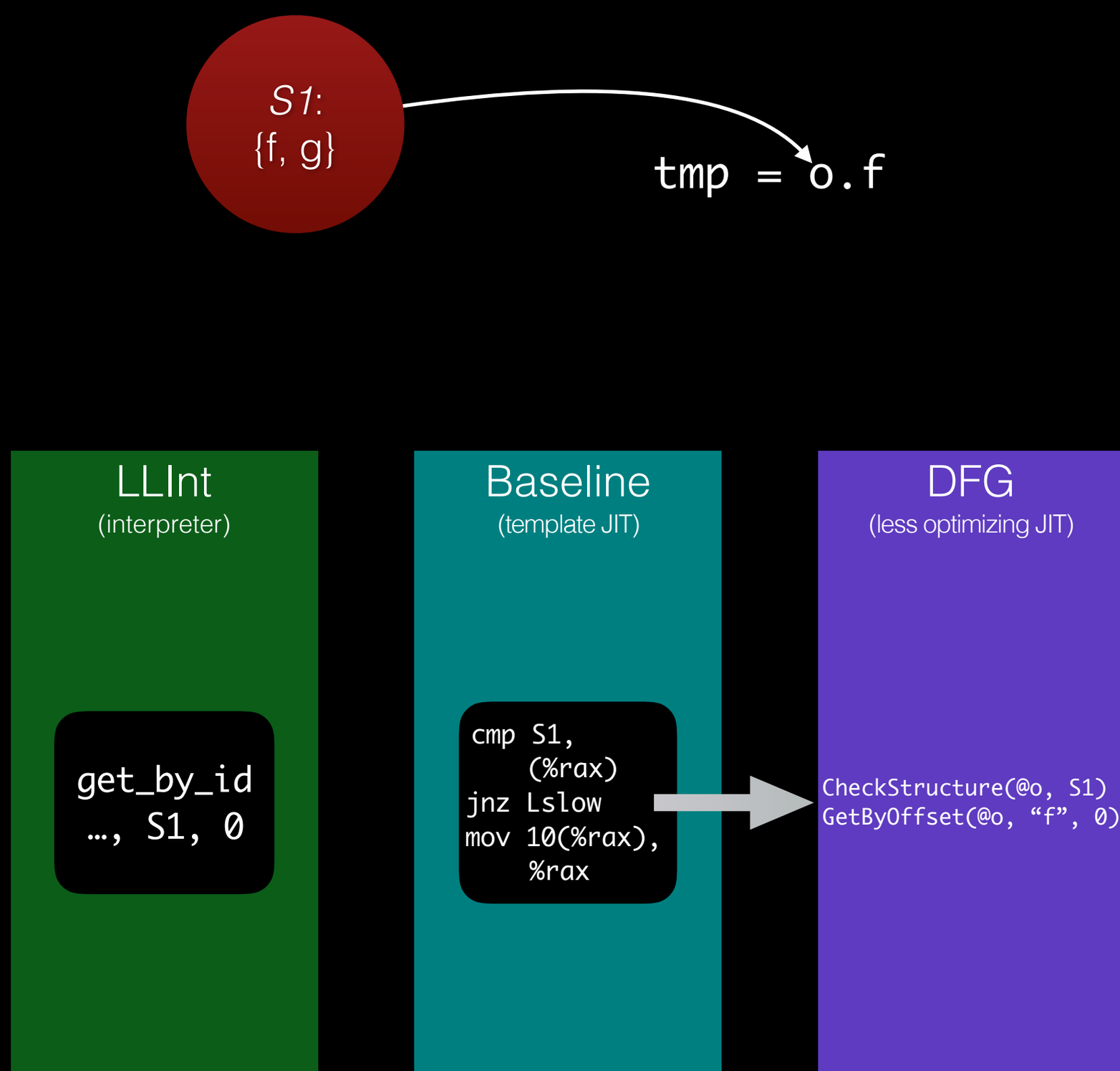
LLInt
(interpreter)

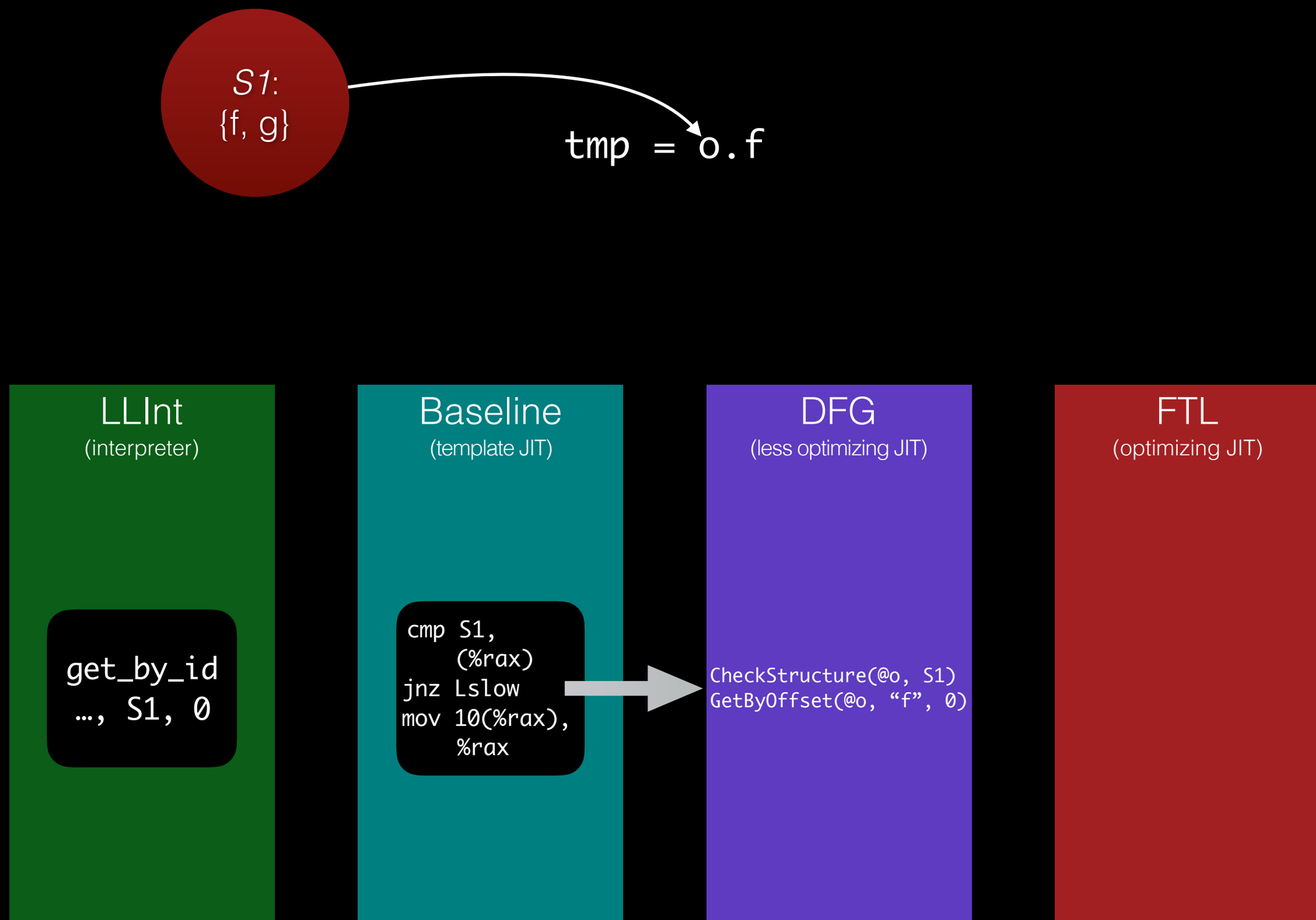
```
get_by_id
…, S1, 0
```

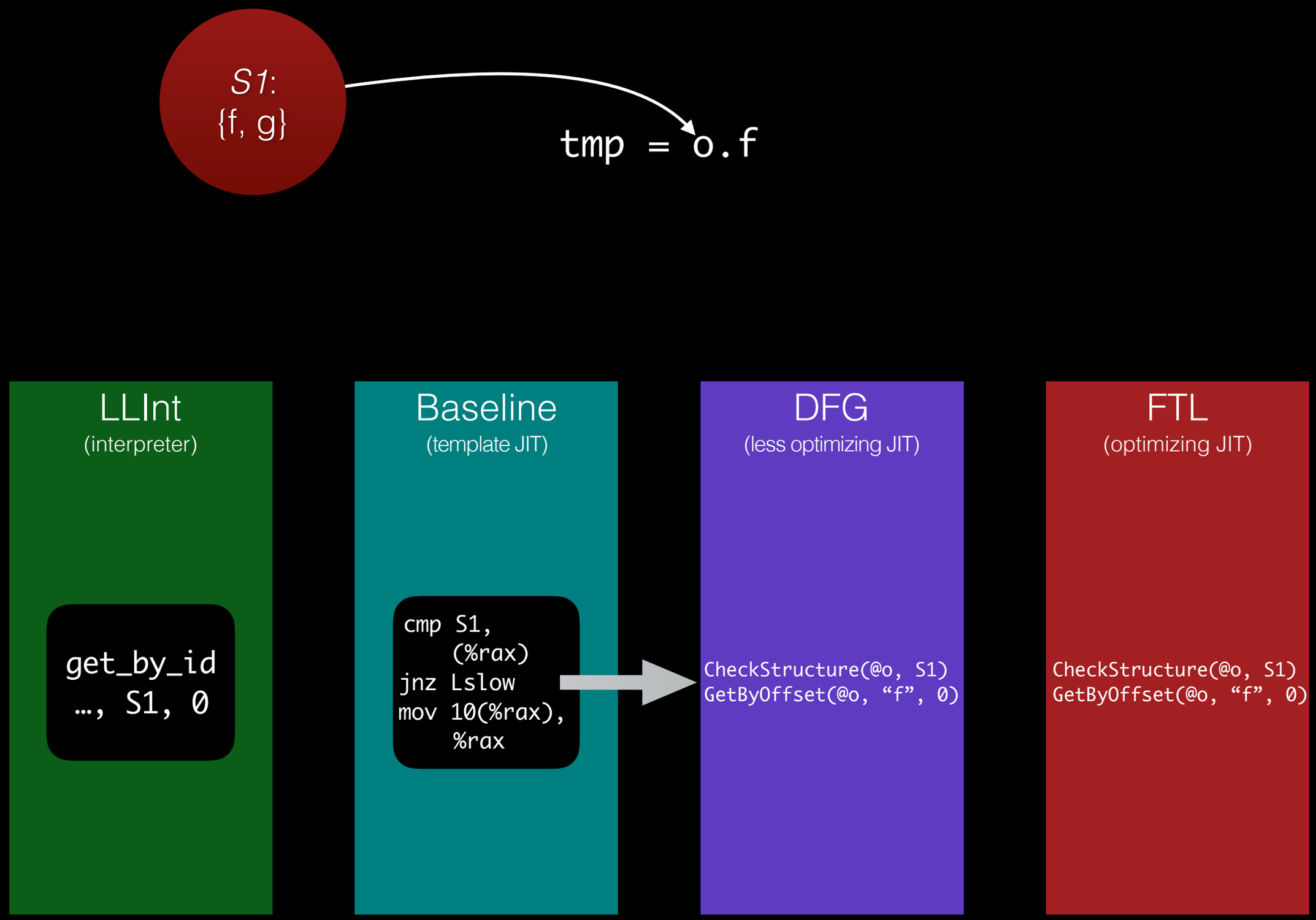
Baseline
(template JIT)

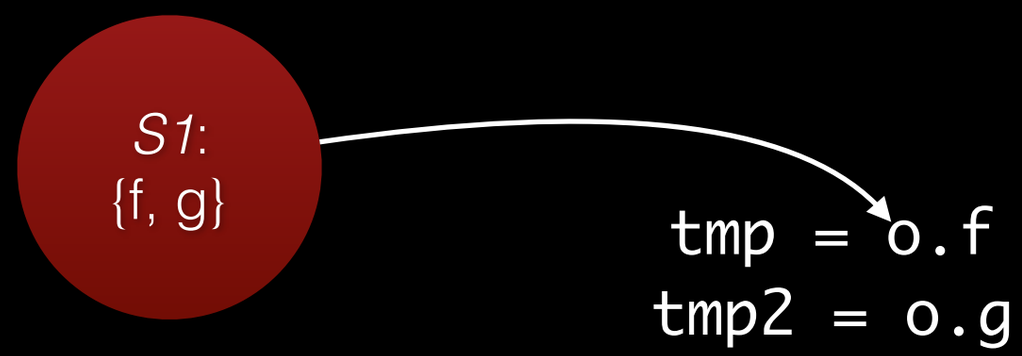
```
cmp S1,
    (%rax)
jnz Lslow
mov 10(%rax),
    %rax
```

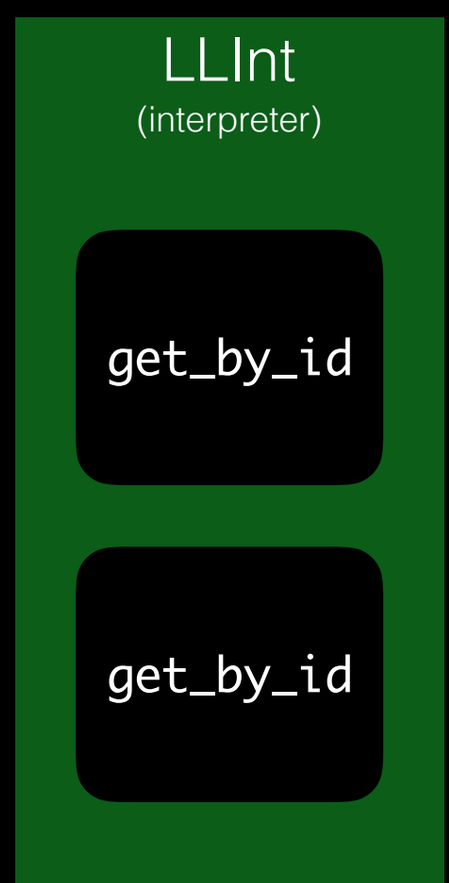
DFG
(less optimizing JIT)

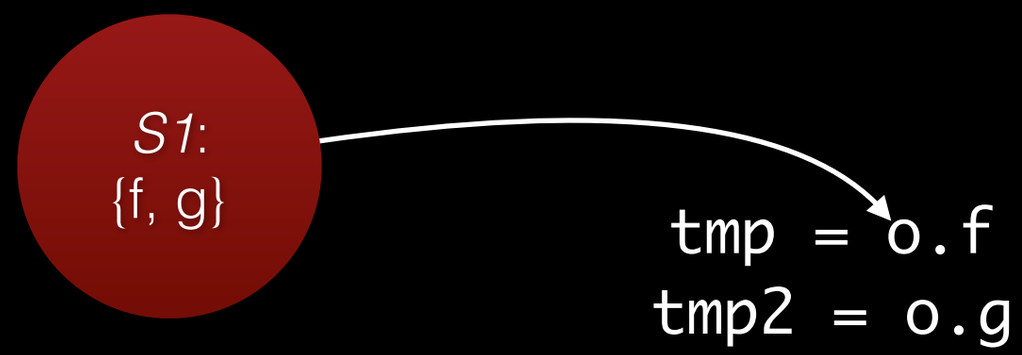

S1:
{f, g}
tmp = o.f
LLInt
(interpreter)
get_by_id
…, S1, 0
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)

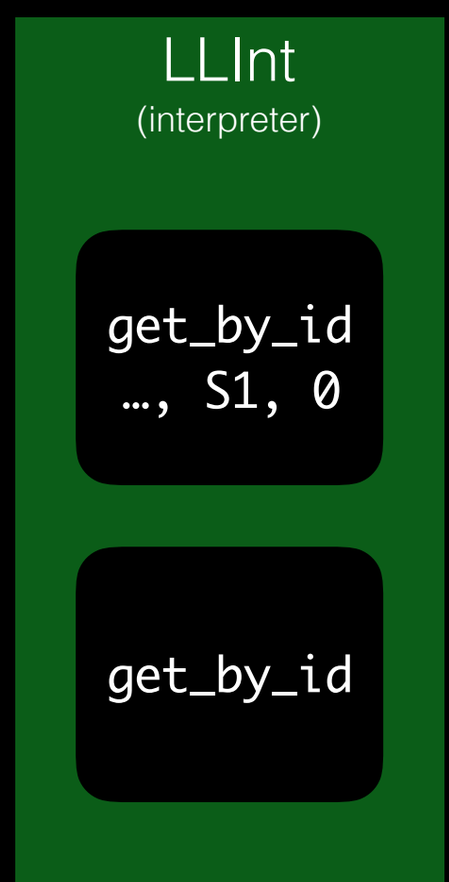

S1:
{f, g}
tmp = o.f
LLInt
(interpreter)
get_by_id
…, S1, 0
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
FTL
(optimizing JIT)

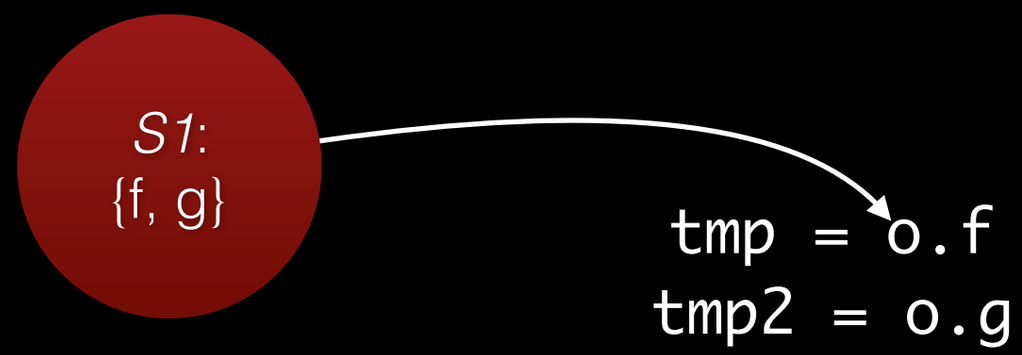

S1:
{f, g}
tmp = o.f
LLInt
(interpreter)
get_by_id
…, S1, 0
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
FTL
(optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)

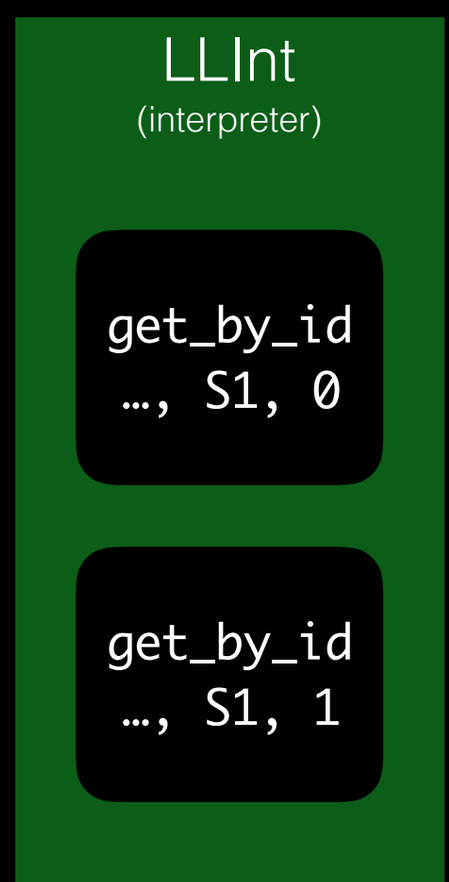

S1:
{f, g}
tmp = o.f
tmp2 = o.g

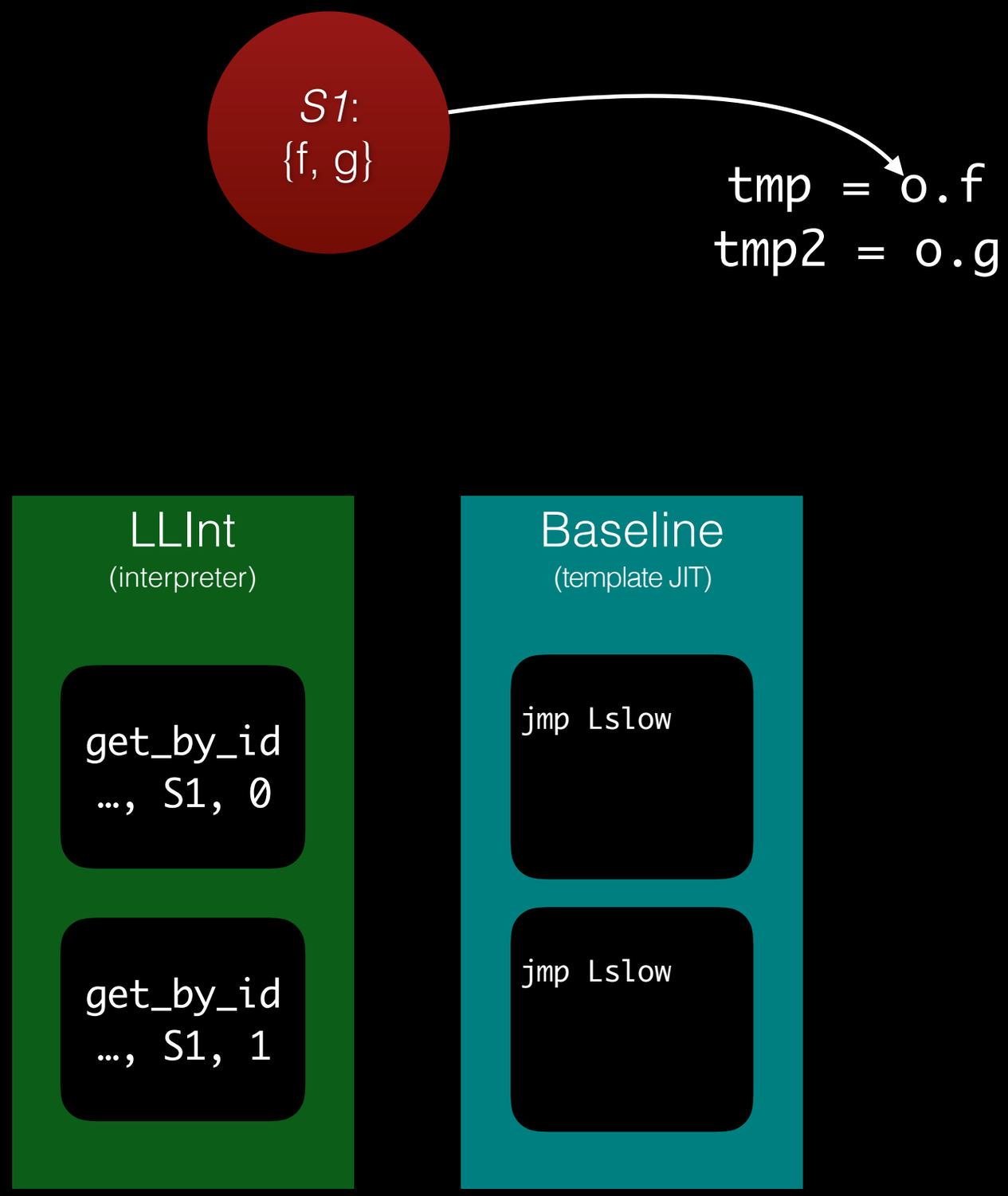

LLInt
(interpreter)
get_by_id
get_by_id

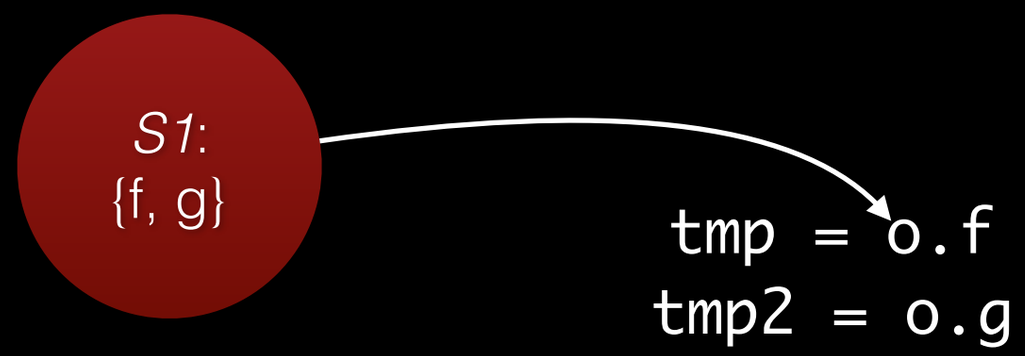

S1:
{f, g}
tmp = o.f
tmp2 = o.g

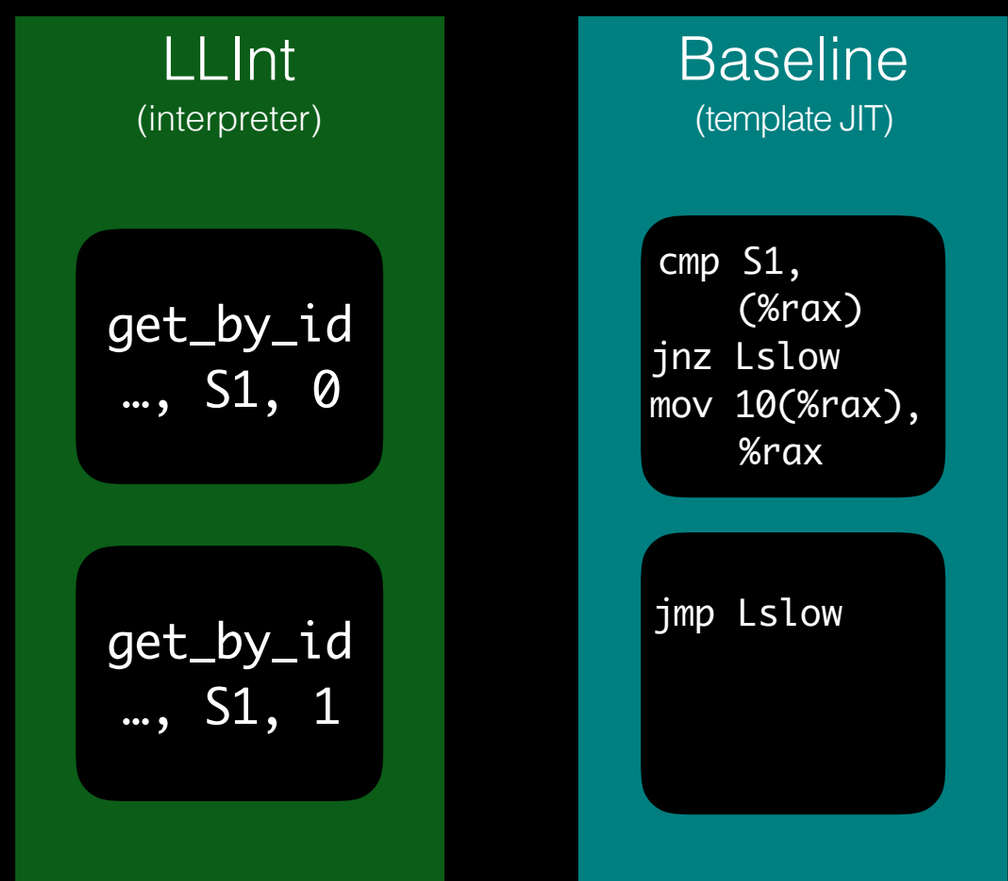

LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id

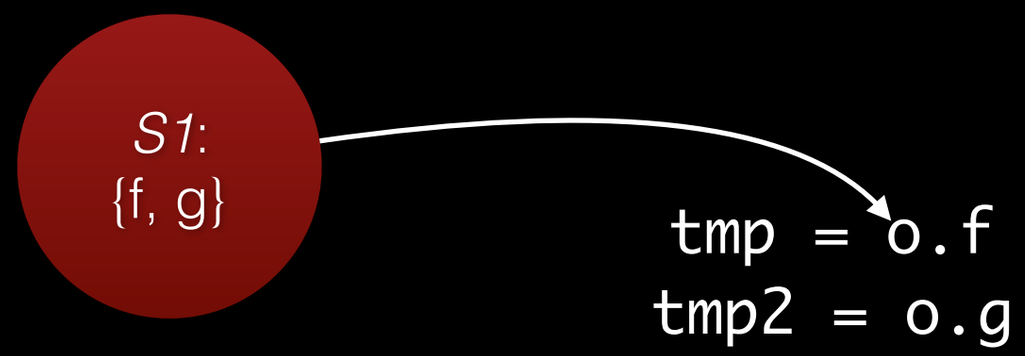

S1:
{f, g}
tmp = o.f
tmp2 = o.g

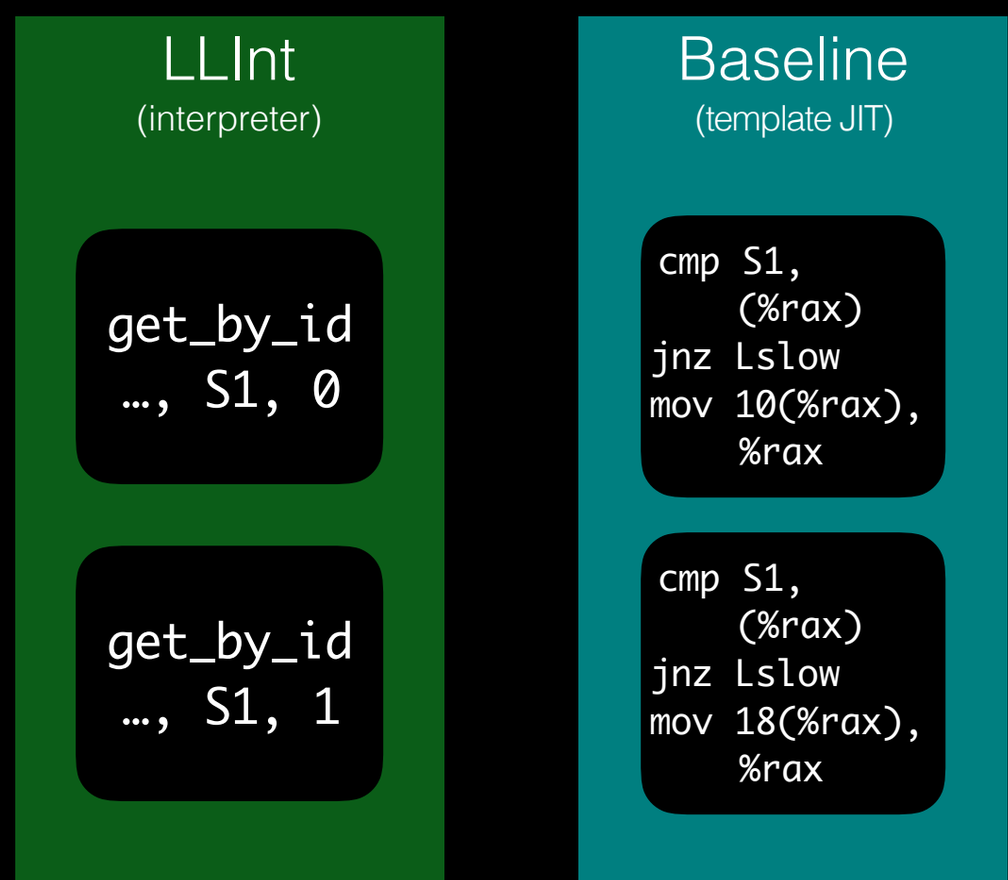

LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1

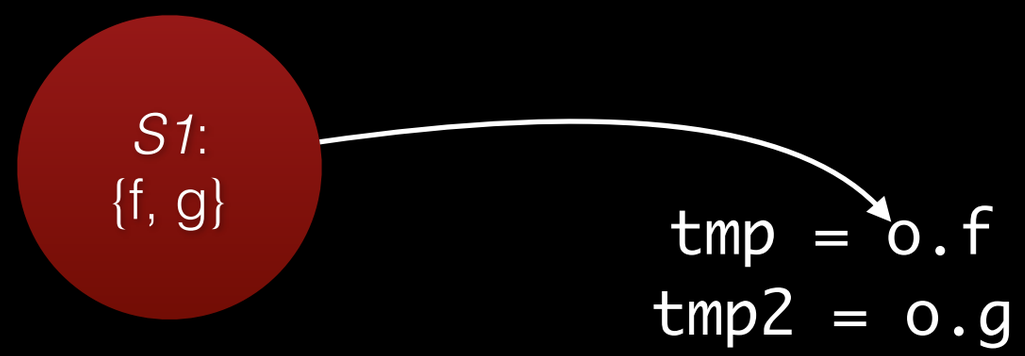

S1:
{f, g}
tmp = o.f
tmp2 = o.g
LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
jmp Lslow
jmp Lslow


S1:
{f, g}
tmp = o.f
tmp2 = o.g


LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
jmp Lslow


S1:
{f, g}
tmp = o.f
tmp2 = o.g


LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
cmp S1,
(%rax)
jnz Lslow
mov 18(%rax),
%rax



LLInt
(interpreter)

```
get_by_id
…, S1, 0
```

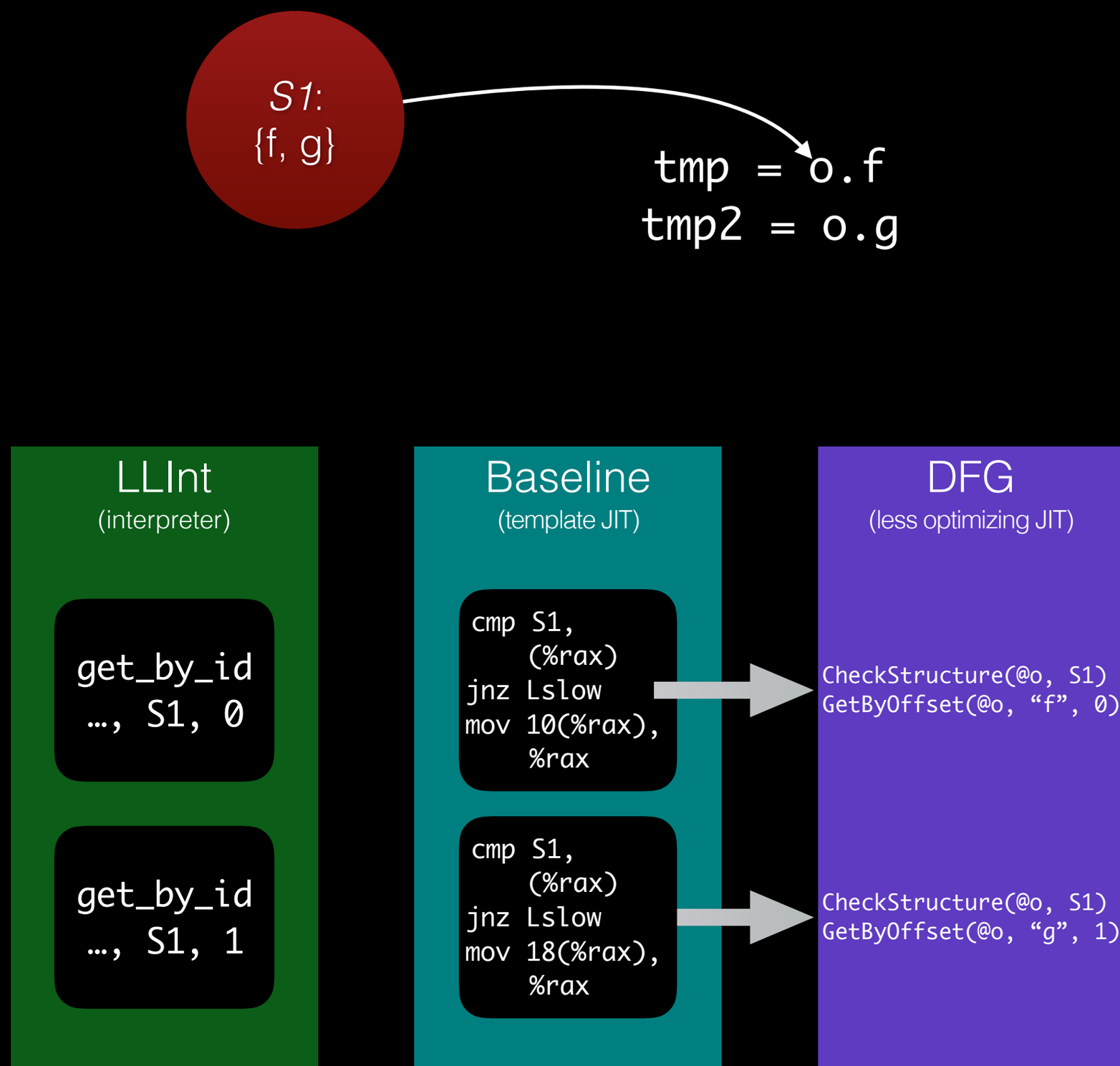
```
get_by_id
…, S1, 1
```

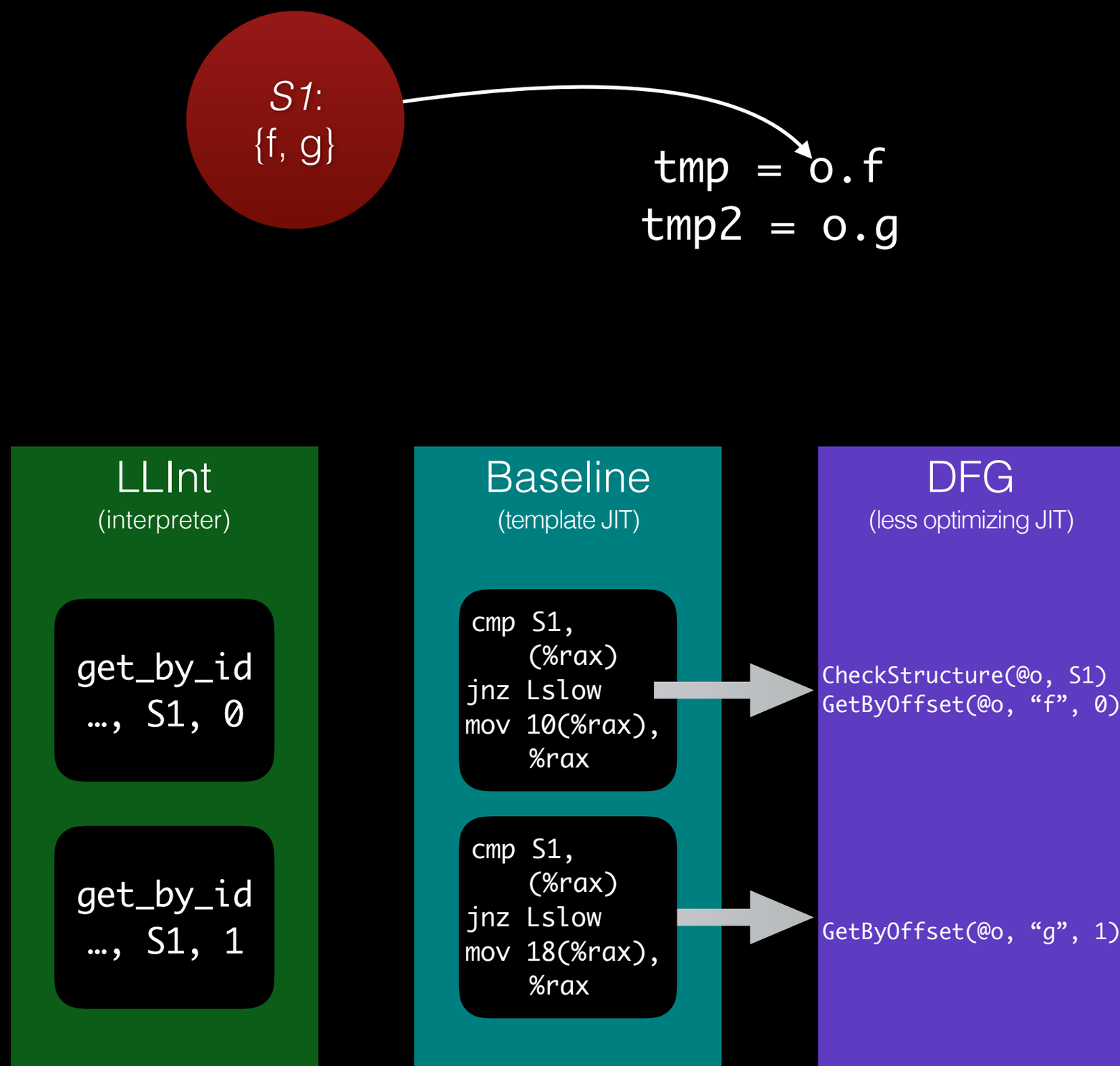
Baseline
(template JIT)

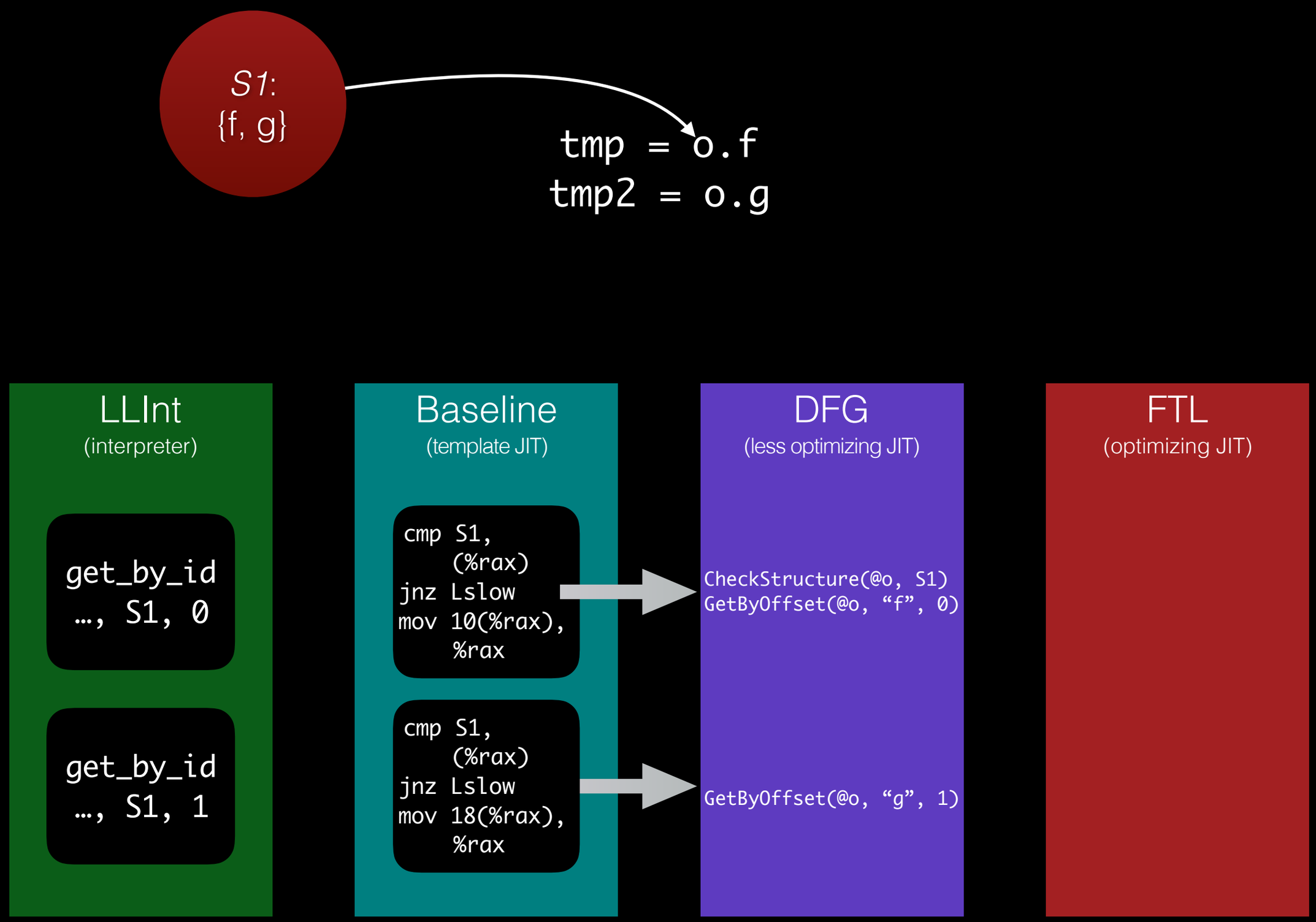
```
cmp S1,
    (%rax)
jnz Lslow
mov 10(%rax),
    %rax
```

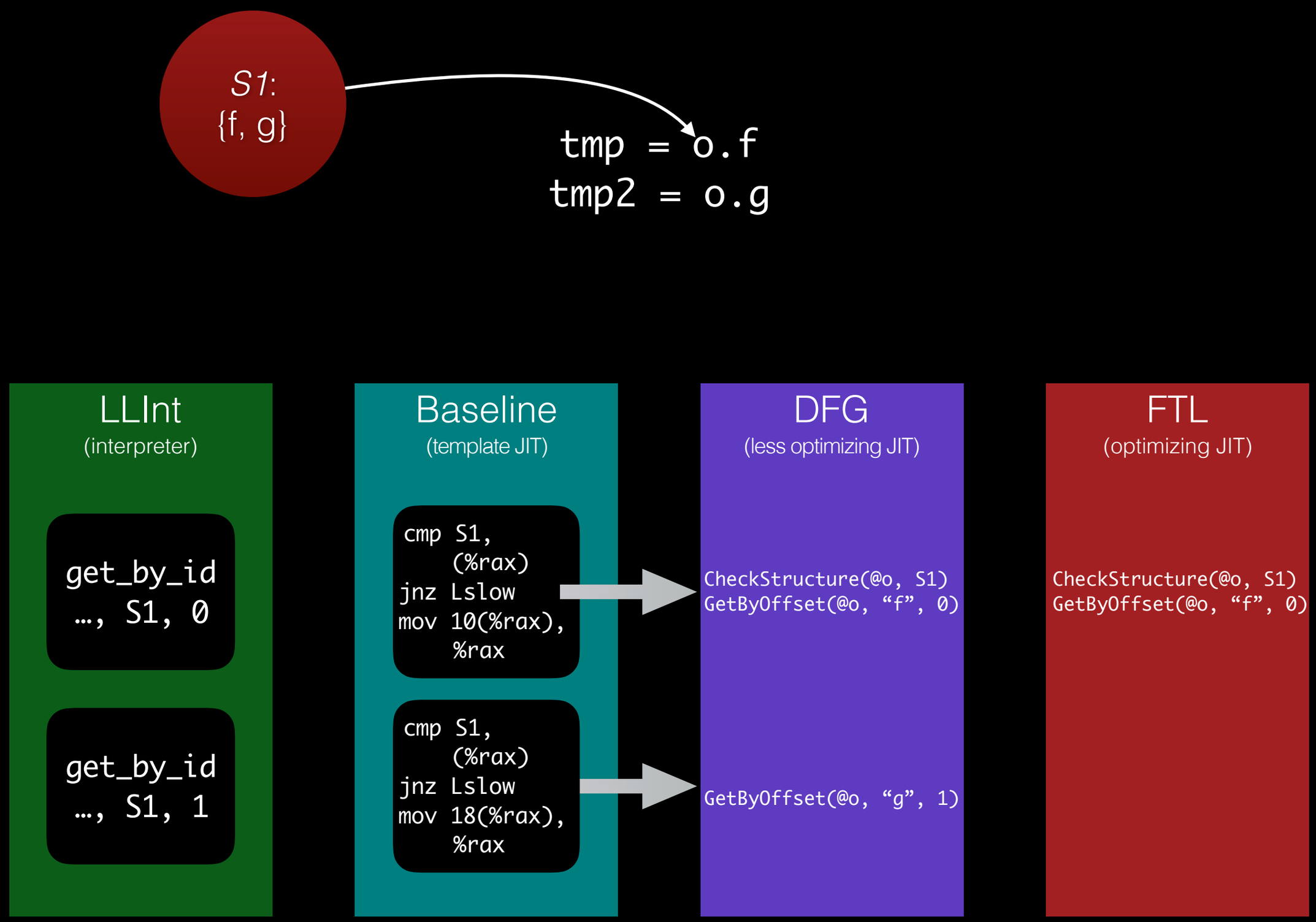
```
cmp S1,
    (%rax)
jnz Lslow
mov 18(%rax),
    %rax
```

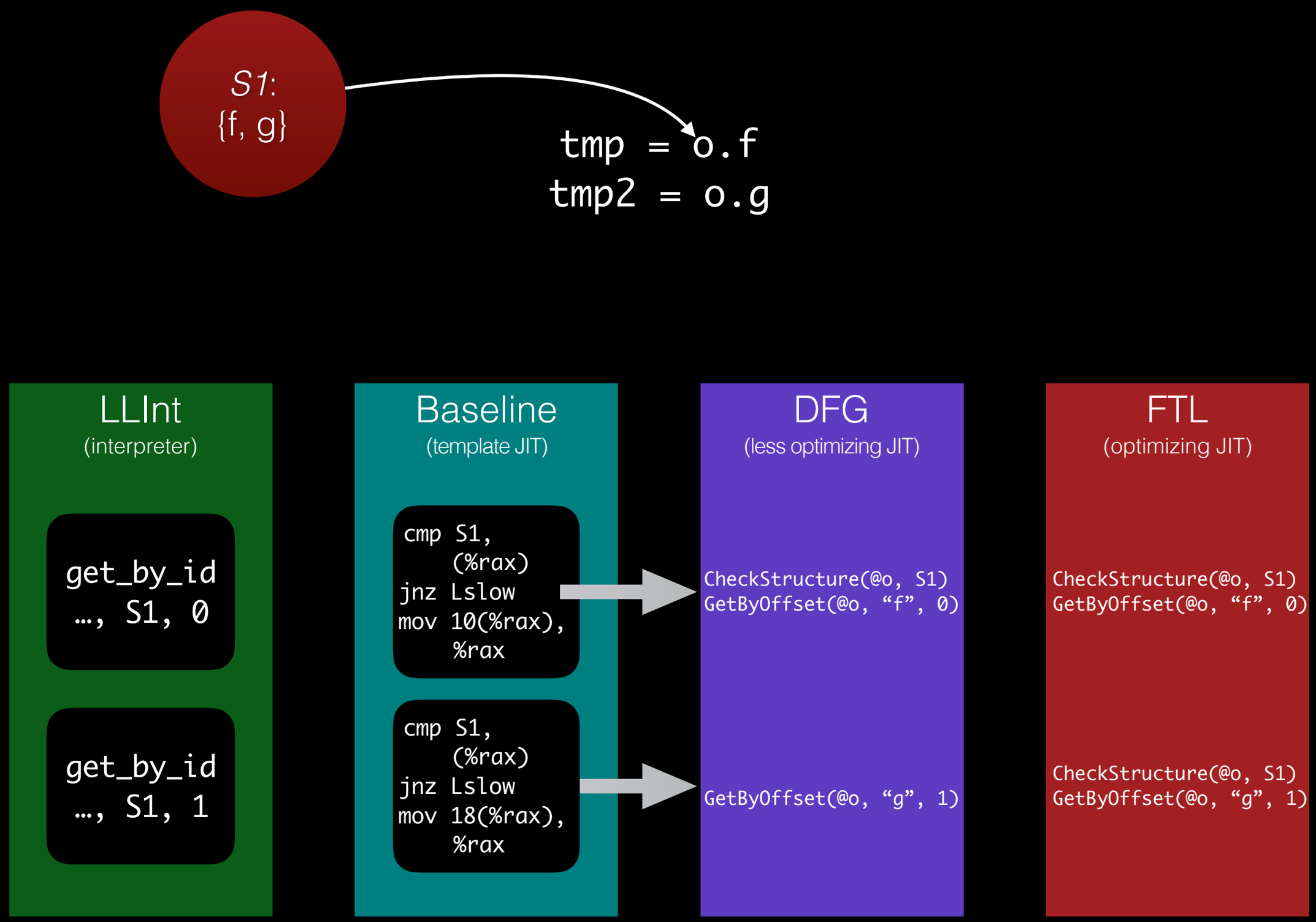
DFG
(less optimizing JIT)

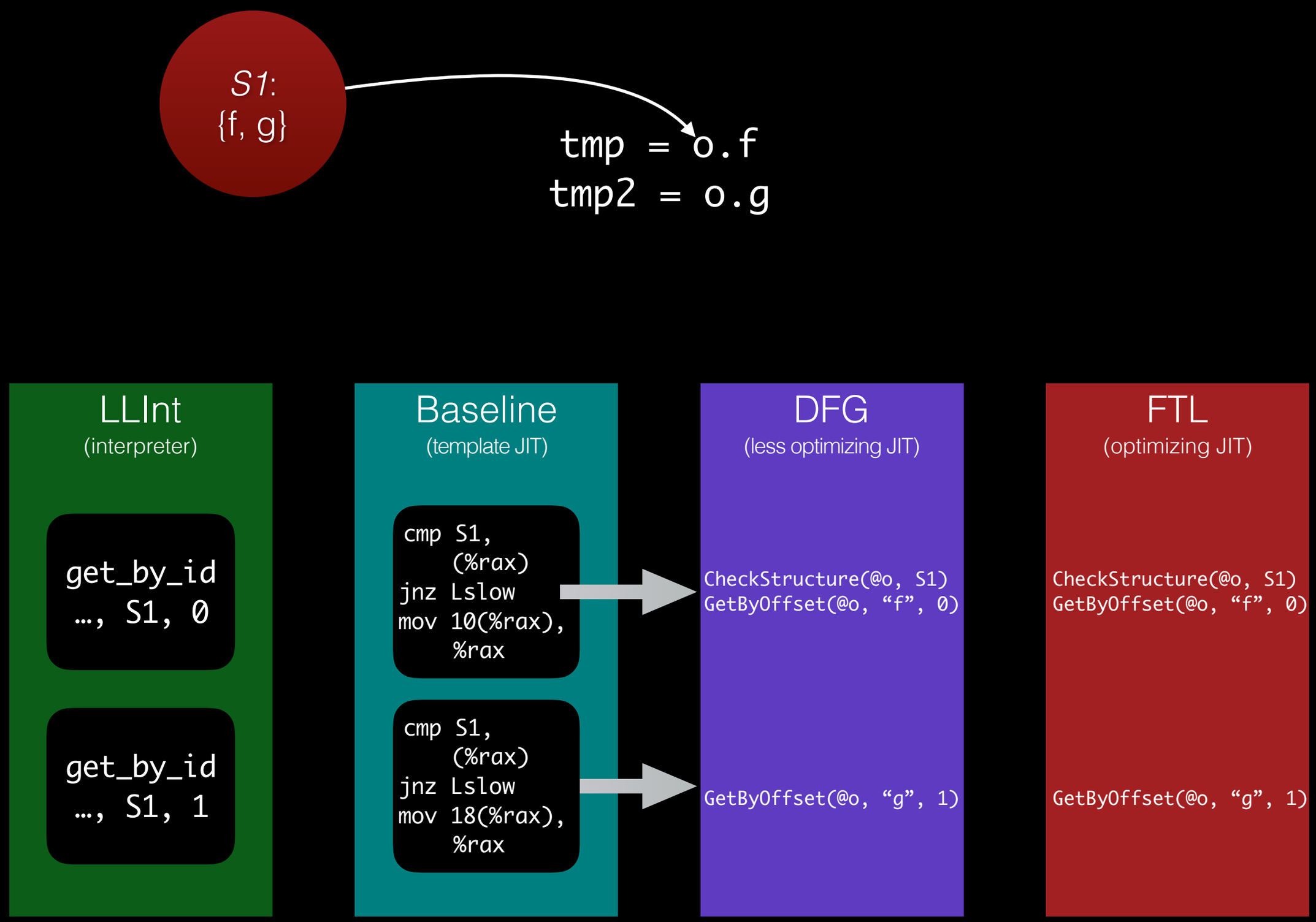

S1:
{f, g}
tmp = o.f
tmp2 = o.g
LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
cmp S1,
(%rax)
jnz Lslow
mov 18(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
CheckStructure(@o, S1)
GetByOffset(@o, “g”, 1)


S1:
{f, g}
tmp = o.f
tmp2 = o.g
LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
cmp S1,
(%rax)
jnz Lslow
mov 18(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
GetByOffset(@o, “g”, 1)

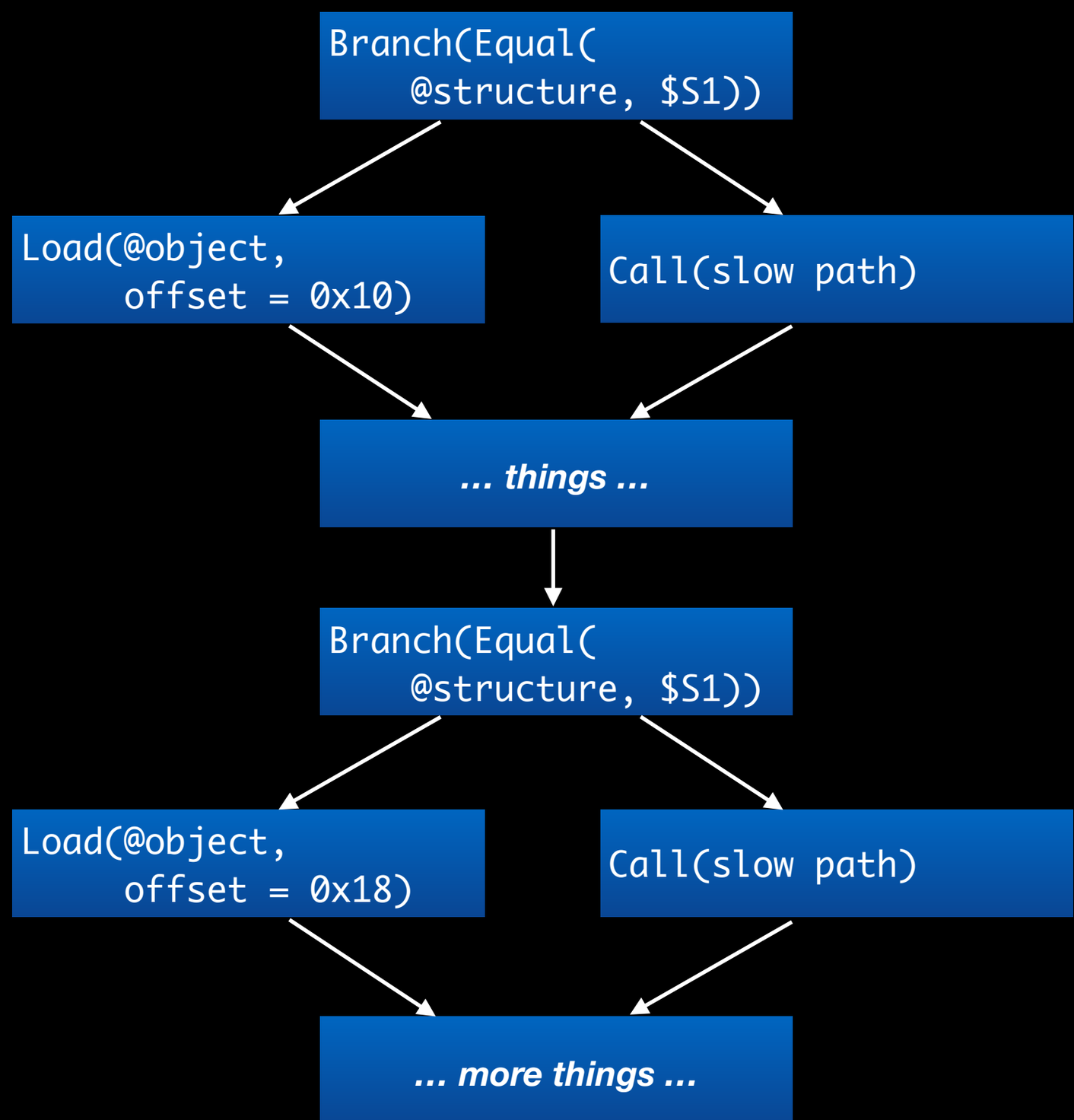

S1:
{f, g}
tmp = o.f
tmp2 = o.g
LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
cmp S1,
(%rax)
jnz Lslow
mov 18(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
GetByOffset(@o, “g”, 1)
FTL
(optimizing JIT)


S1:
{f, g}
tmp = o.f
tmp2 = o.g
LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
cmp S1,
(%rax)
jnz Lslow
mov 18(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
GetByOffset(@o, “g”, 1)
FTL
(optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)

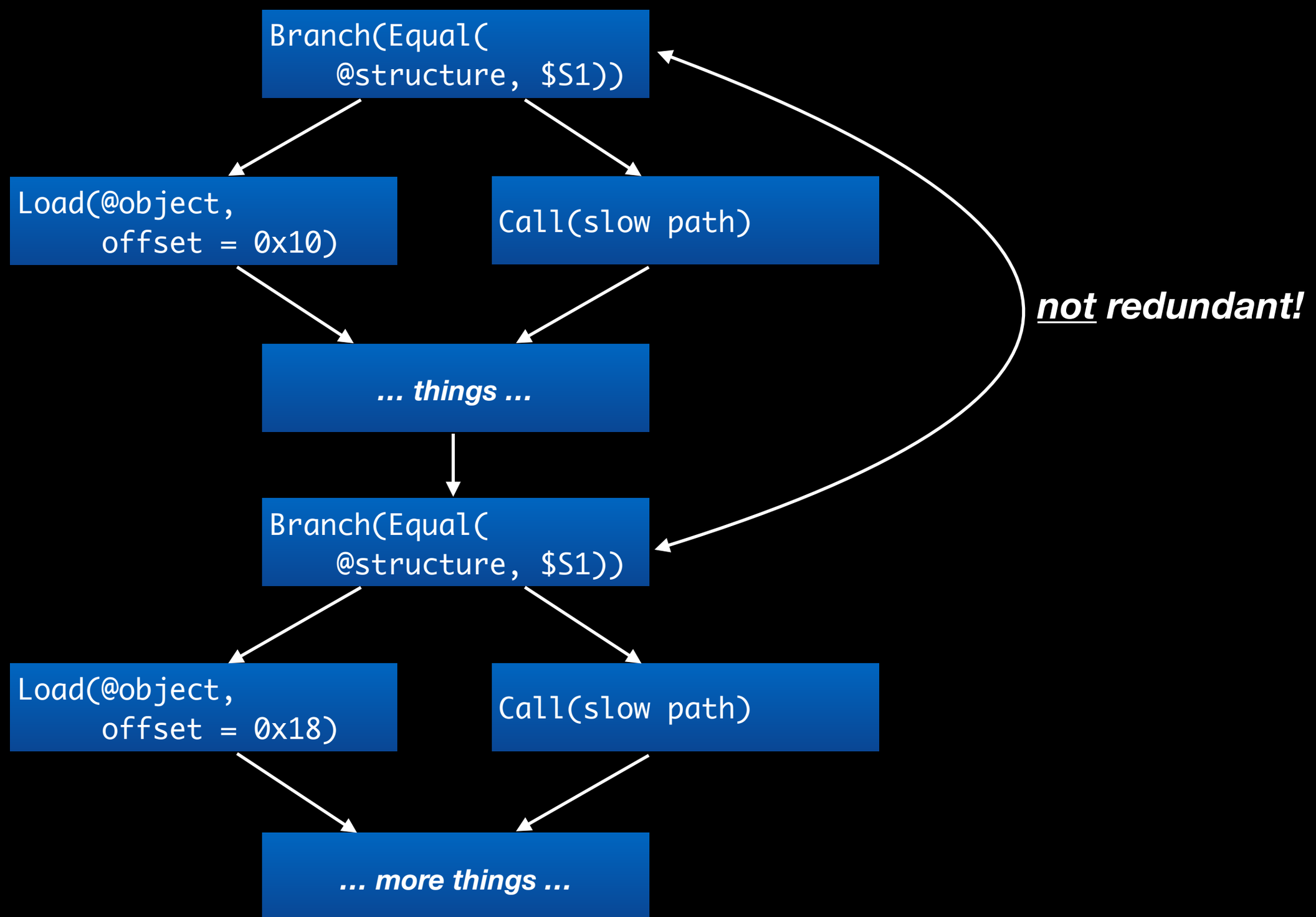

S1:
{f, g}
tmp = o.f
tmp2 = o.g
LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
cmp S1,
(%rax)
jnz Lslow
mov 18(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
GetByOffset(@o, “g”, 1)
FTL
(optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
CheckStructure(@o, S1)
GetByOffset(@o, “g”, 1)


S1:
{f, g}
tmp = o.f
tmp2 = o.g
LLInt
(interpreter)
get_by_id
…, S1, 0
get_by_id
…, S1, 1
Baseline
(template JIT)
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
cmp S1,
(%rax)
jnz Lslow
mov 18(%rax),
%rax
DFG
(less optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
GetByOffset(@o, “g”, 1)
FTL
(optimizing JIT)
CheckStructure(@o, S1)
GetByOffset(@o, “f”, 0)
GetByOffset(@o, “g”, 1)

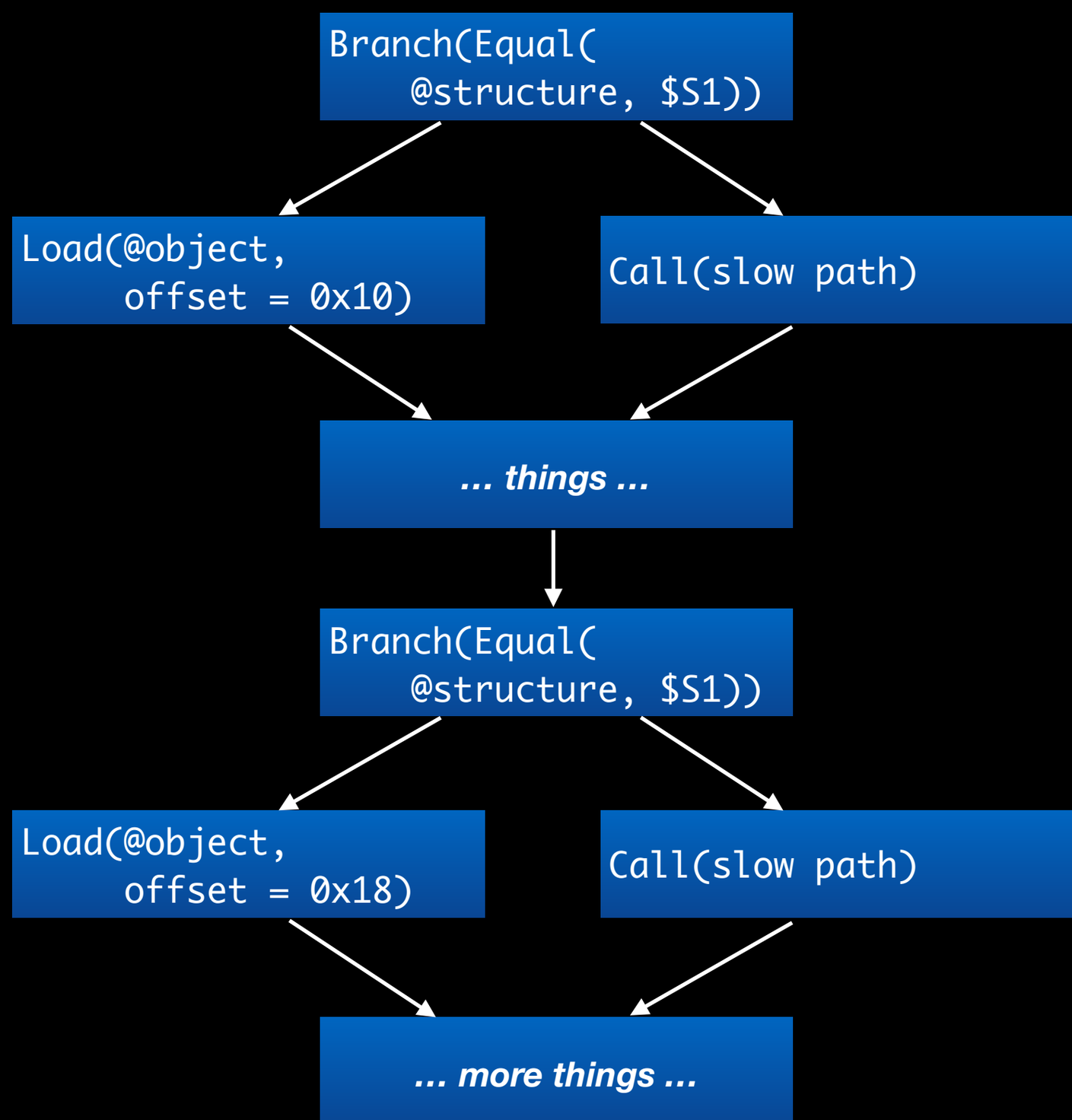
Inline Cache Control Flow



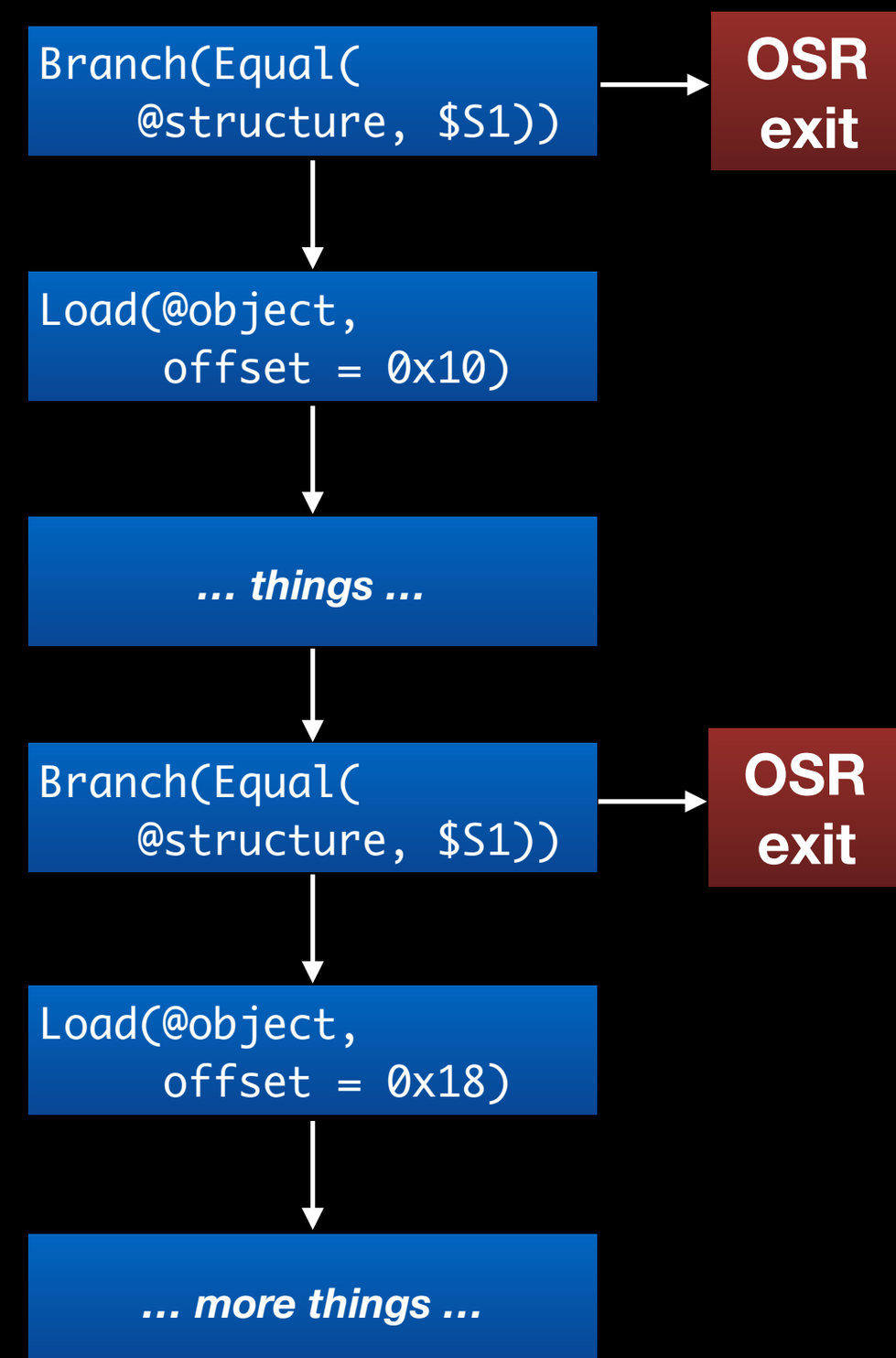
Inline Cache Control Flow



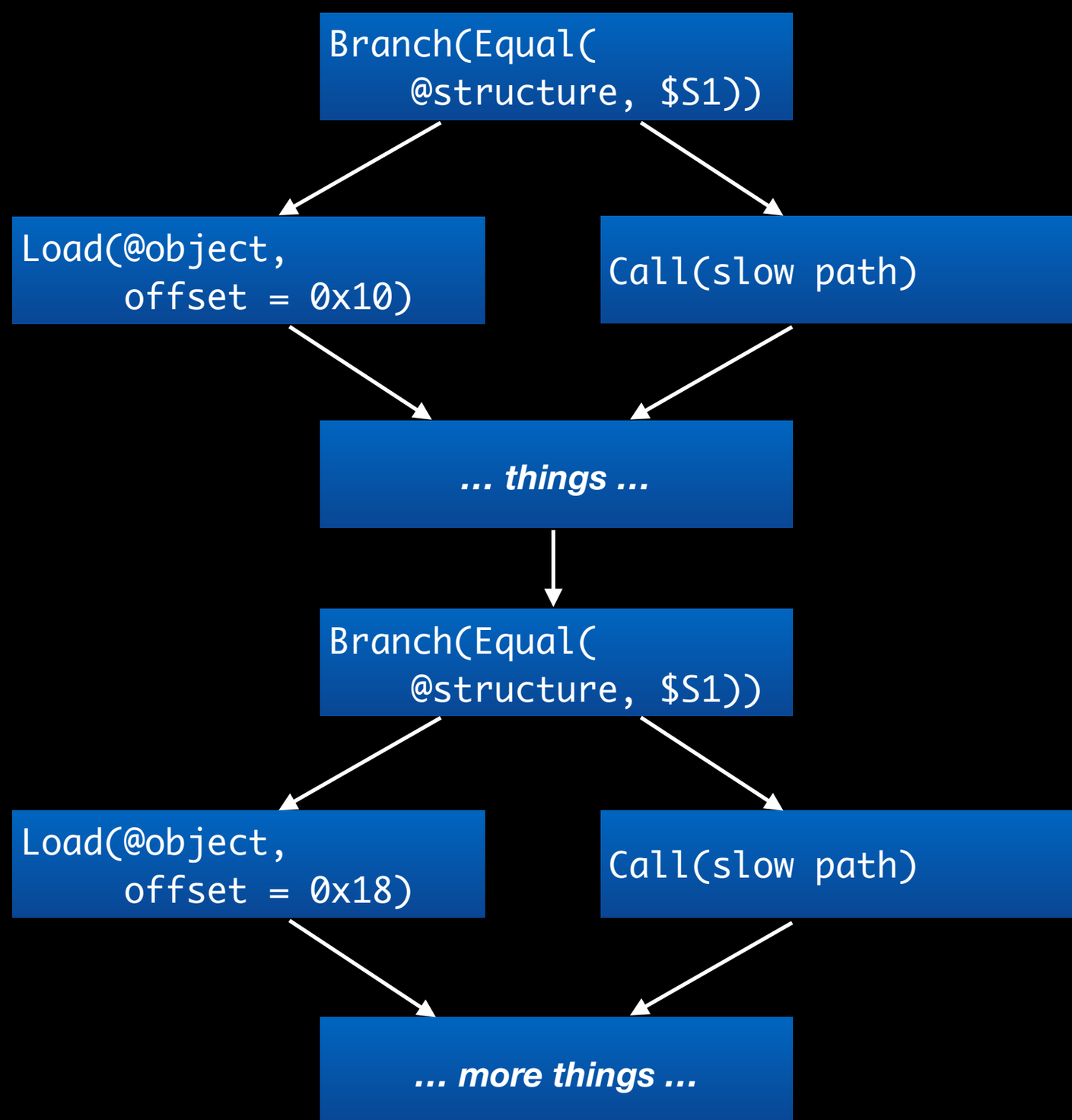
Inline Cache Control Flow



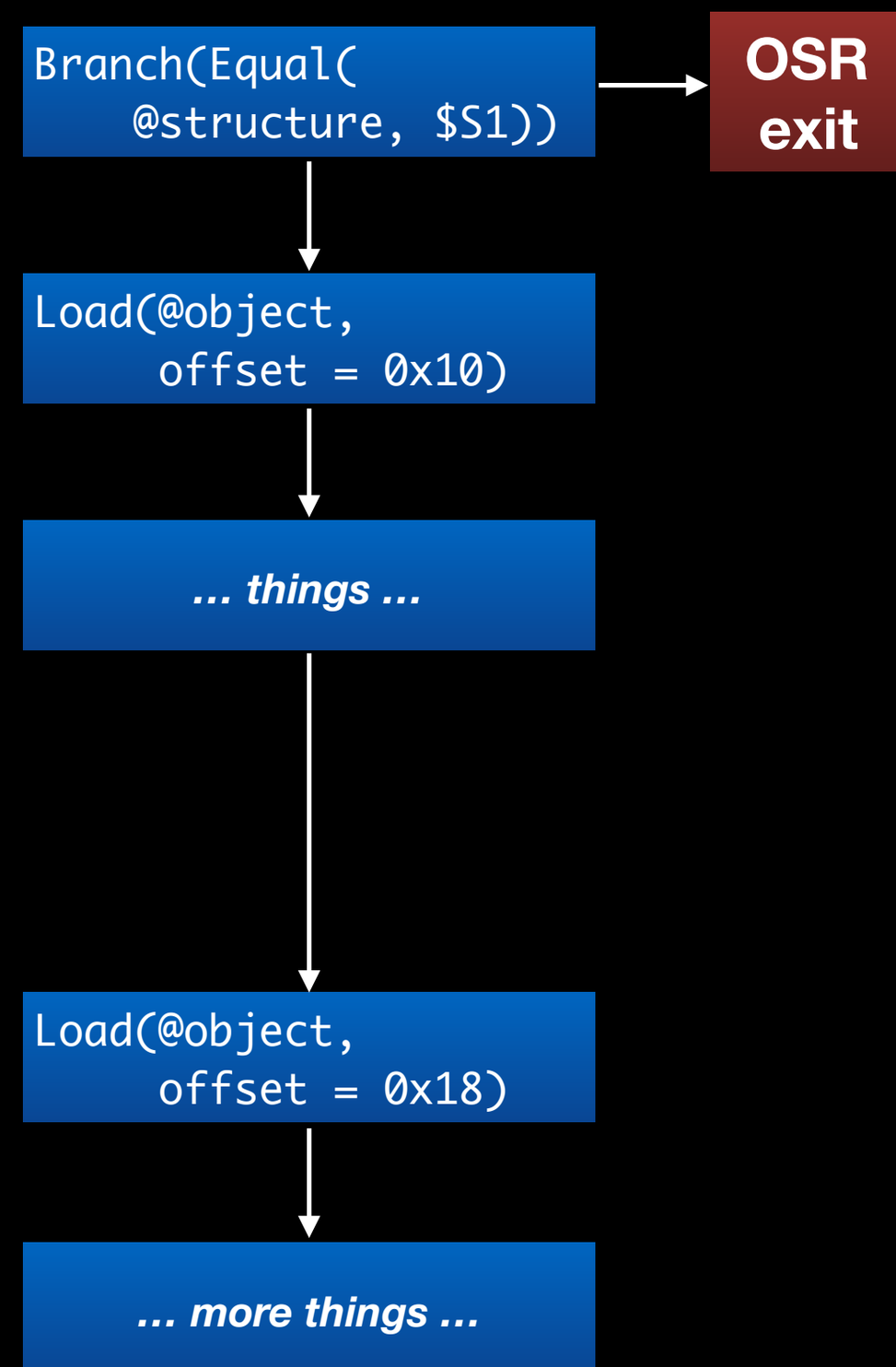
Inlined with OSR exits



Inline Cache Control Flow



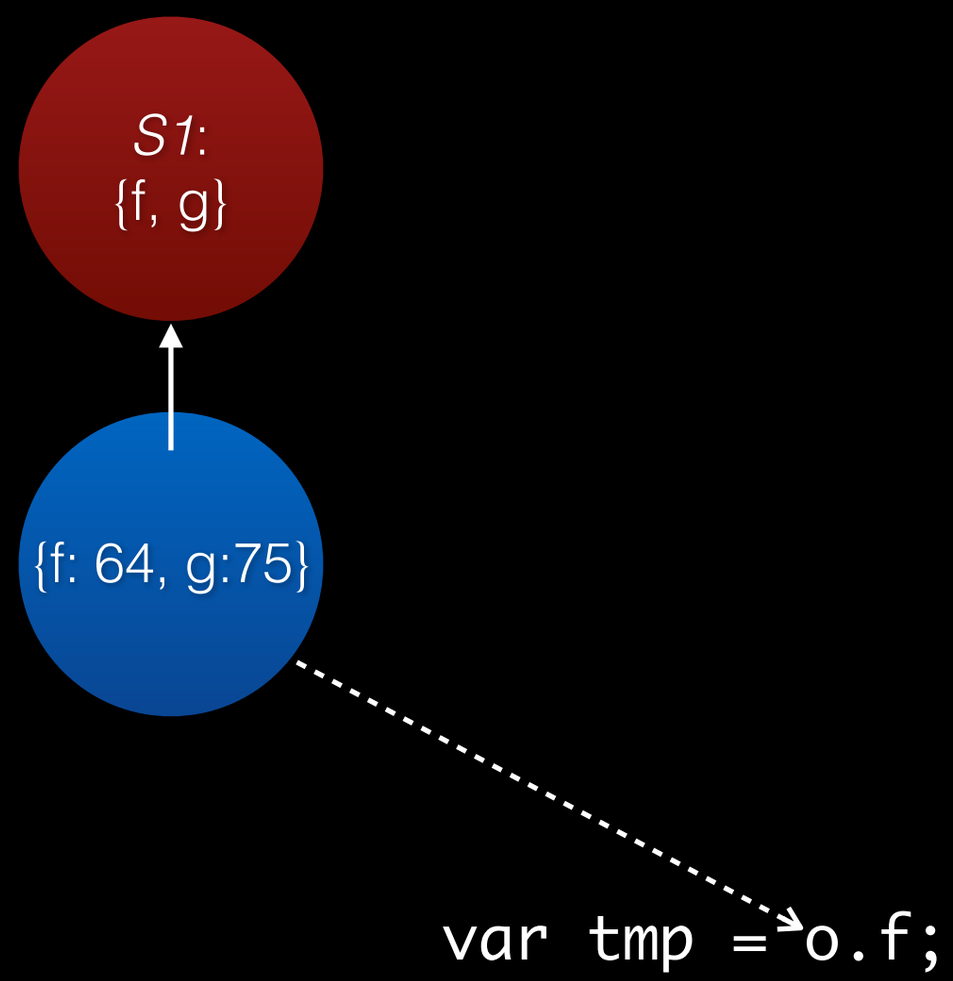
Inlined with OSR exits



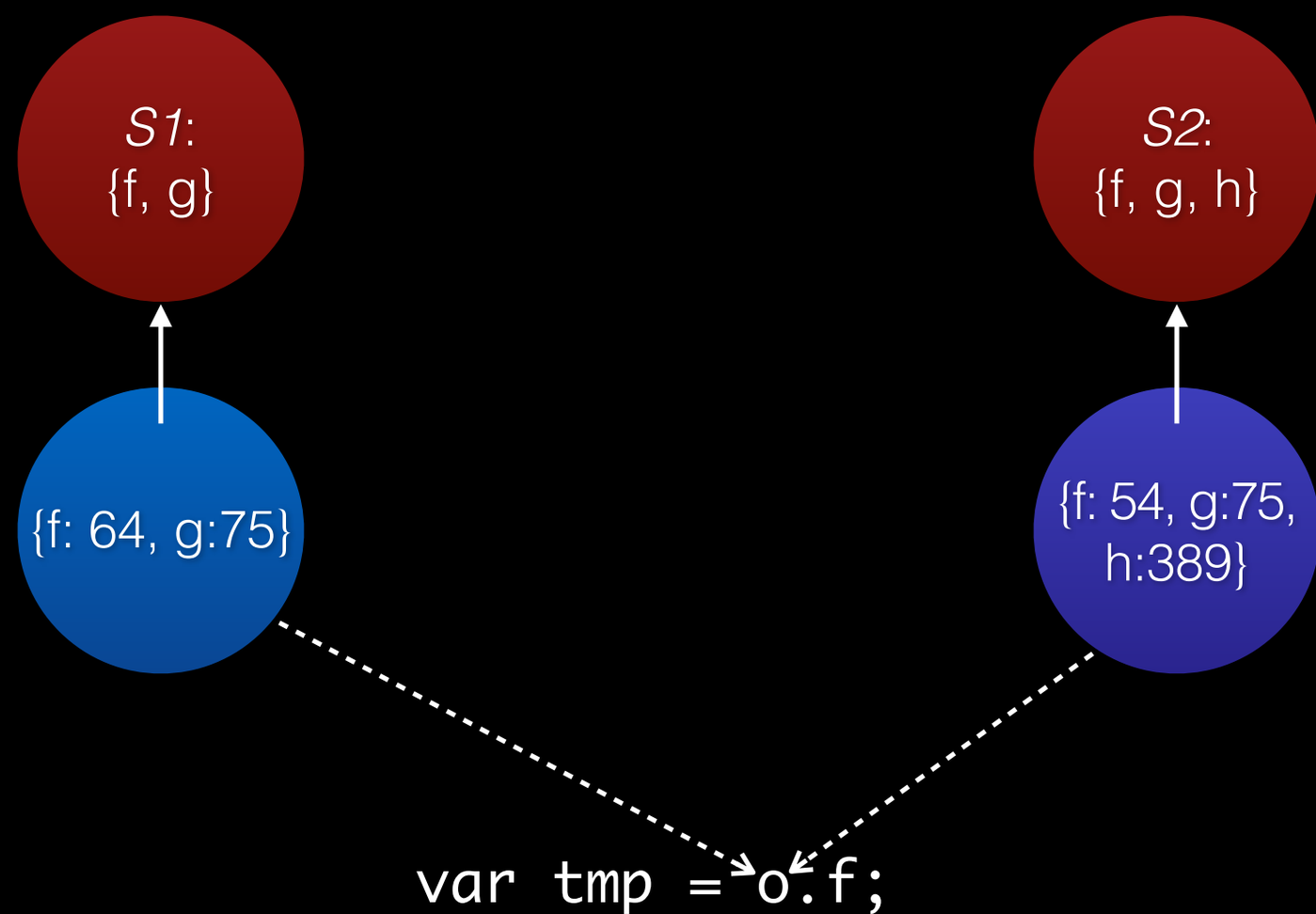
Minimorphic IC Inlining

```
var tmp = o.f;
```

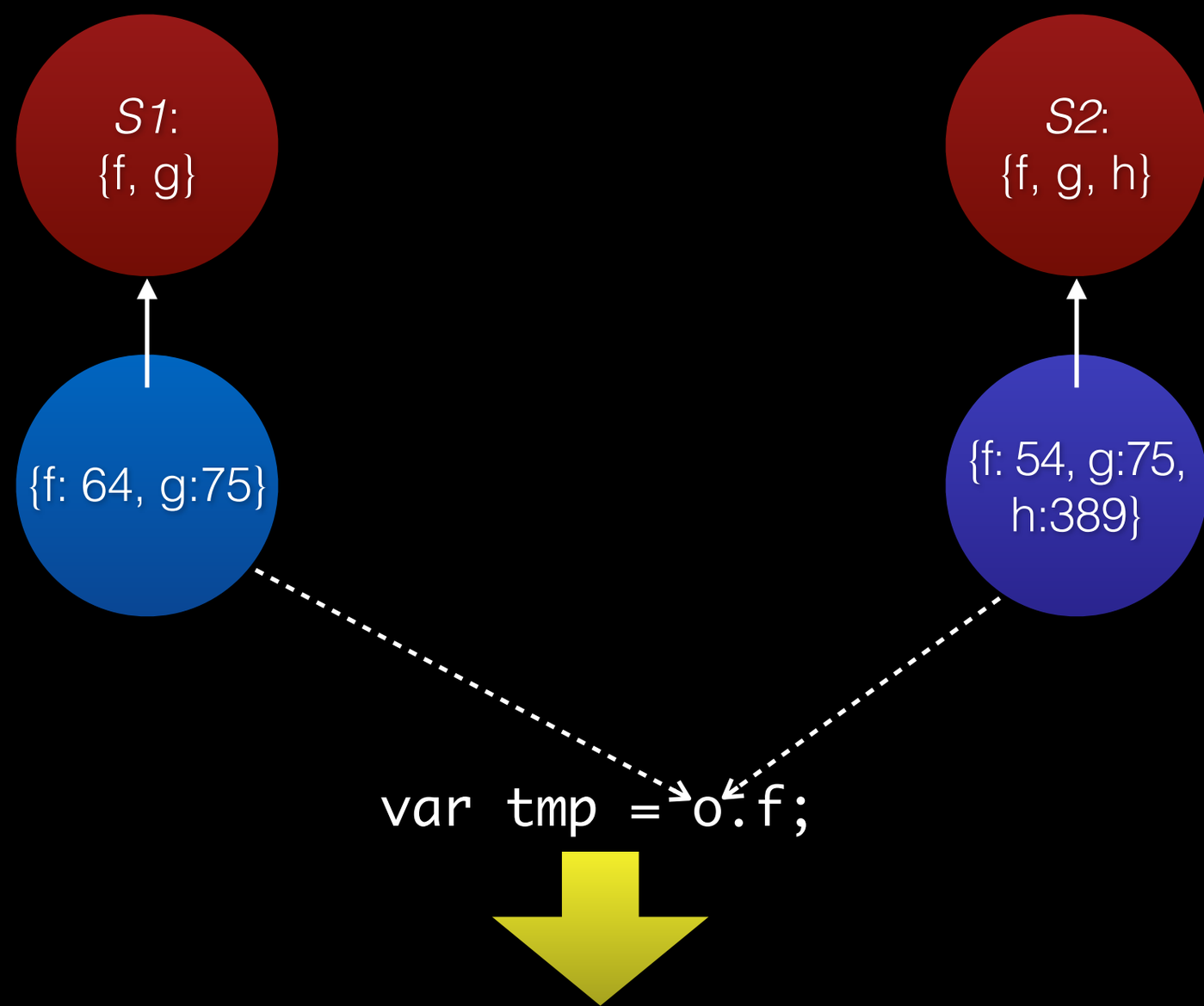
Minimorphic IC Inlining



Minimorphic IC Inlining



Minimorphic IC Inlining

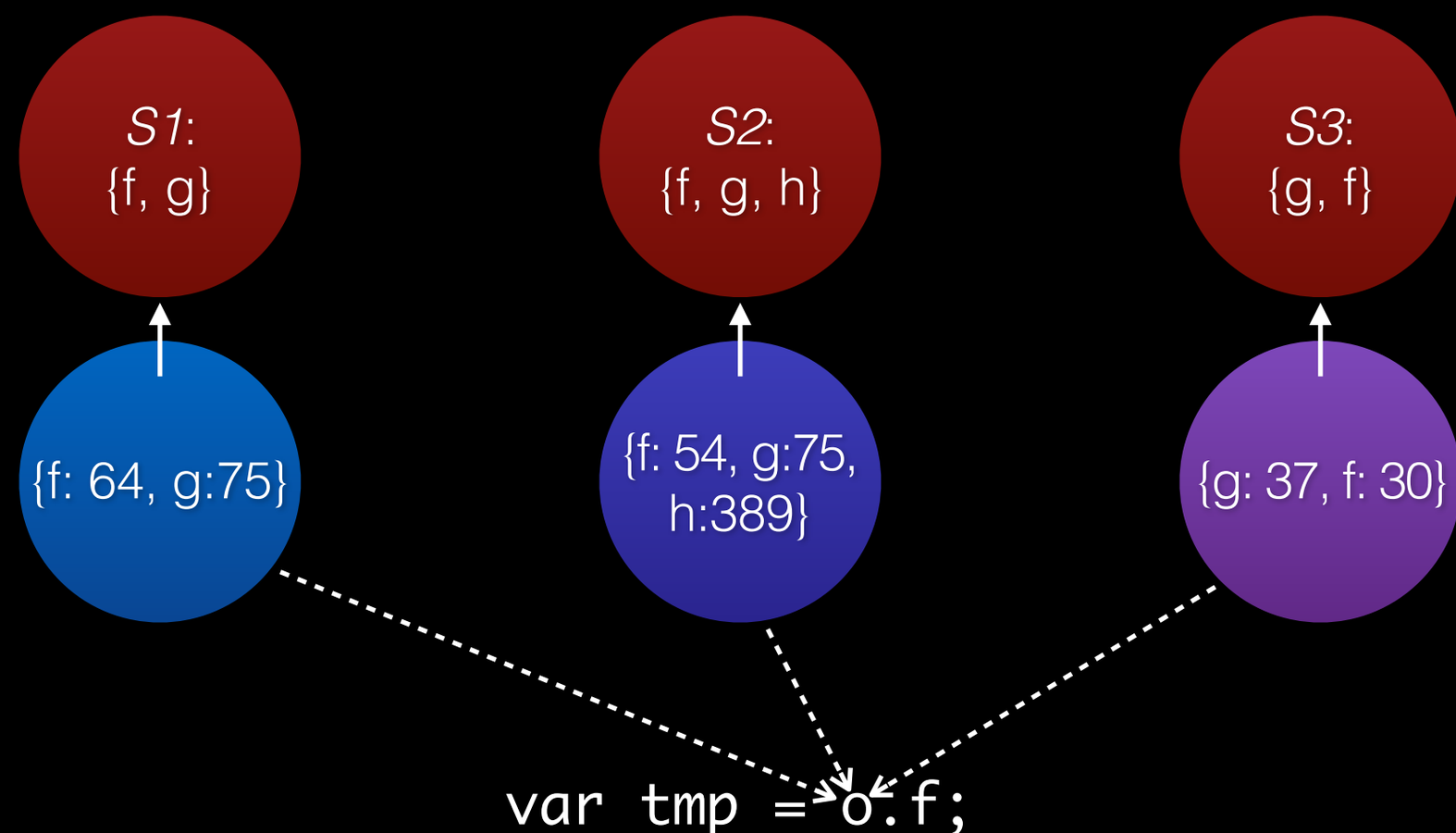



```
CheckStructure(@o, [S1, S2])
GetByOffset(@o, “f”, 0)
```

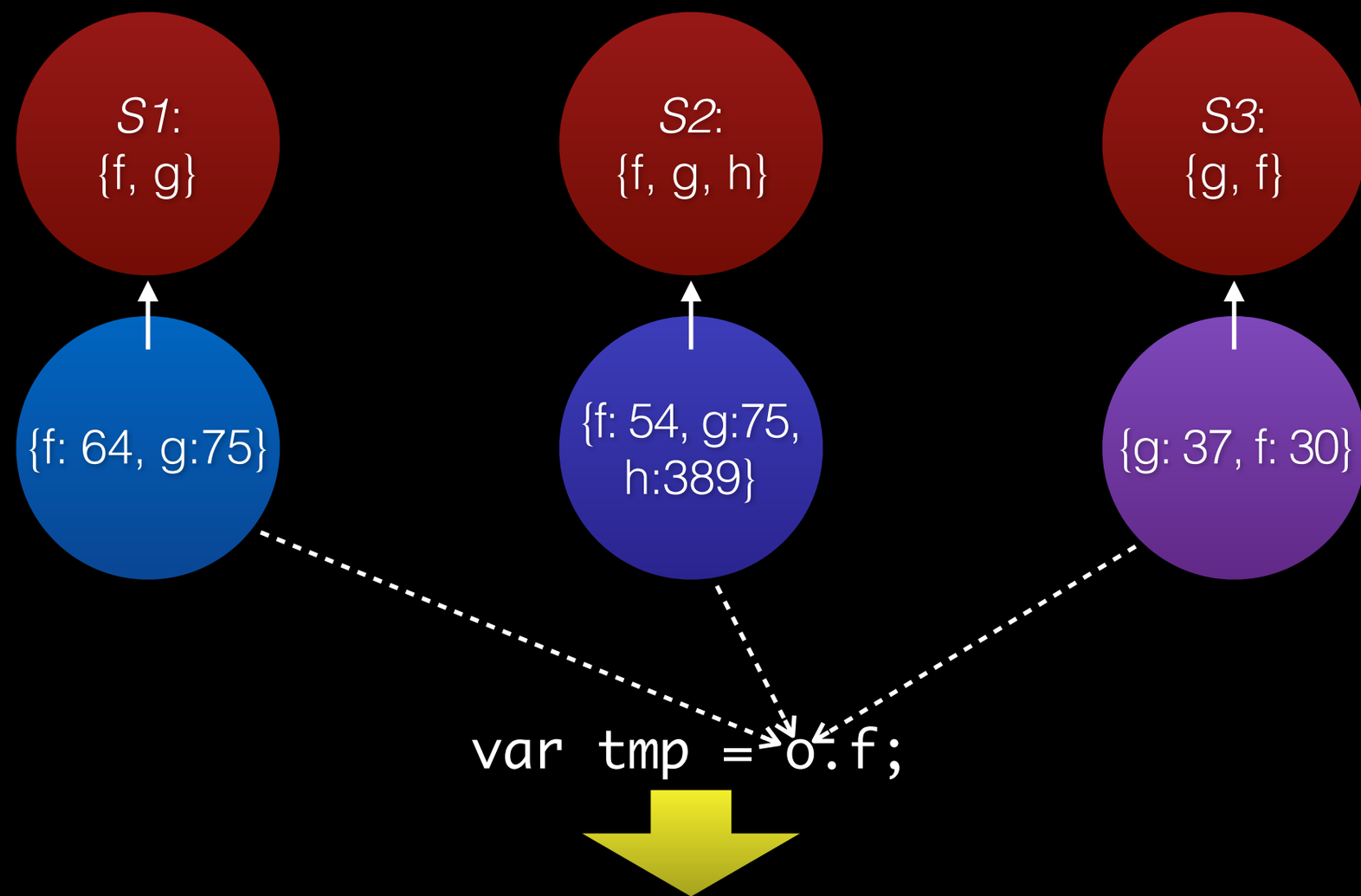
Polymorphic IC Inlining

```
var tmp = o.f;
```

Polymorphic IC Inlining

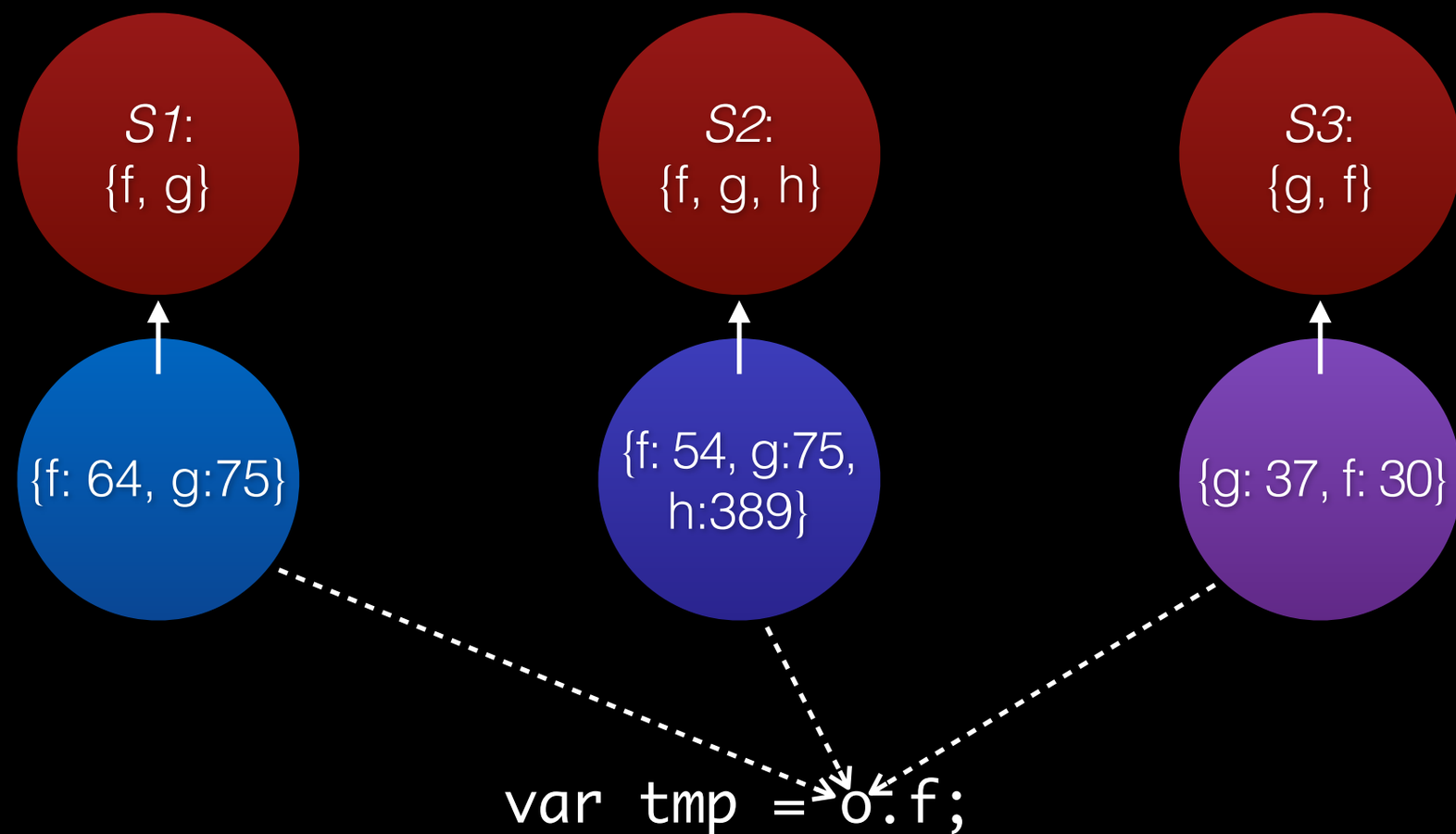


Polymorphic IC Inlining



DFG IR: `MultiGetByOffset(@o, “f”, [S1, S2] => 0, [S3] => 1)`

Polymorphic IC Inlining

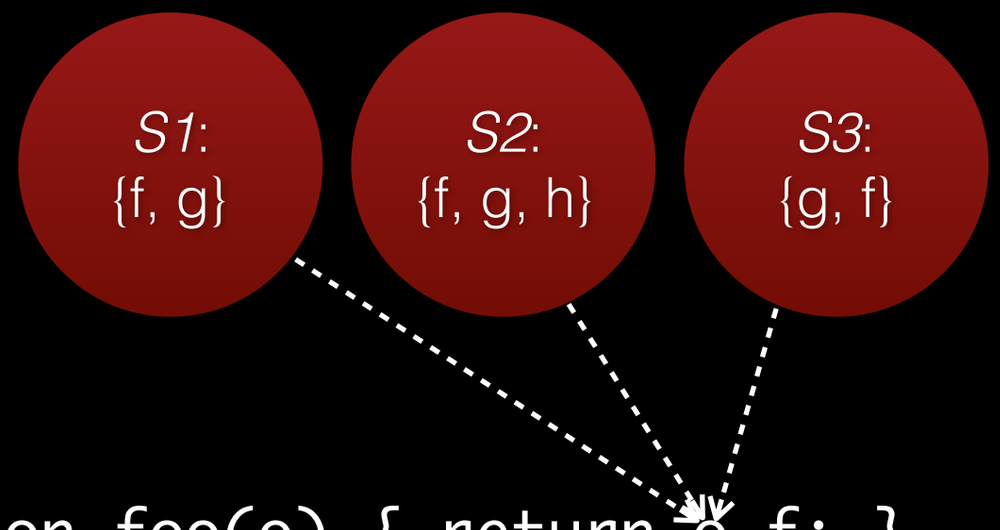


DFG IR: `MultiGetByOffset(@o, “f”, [S1, S2] => 0, [S3] => 1)`

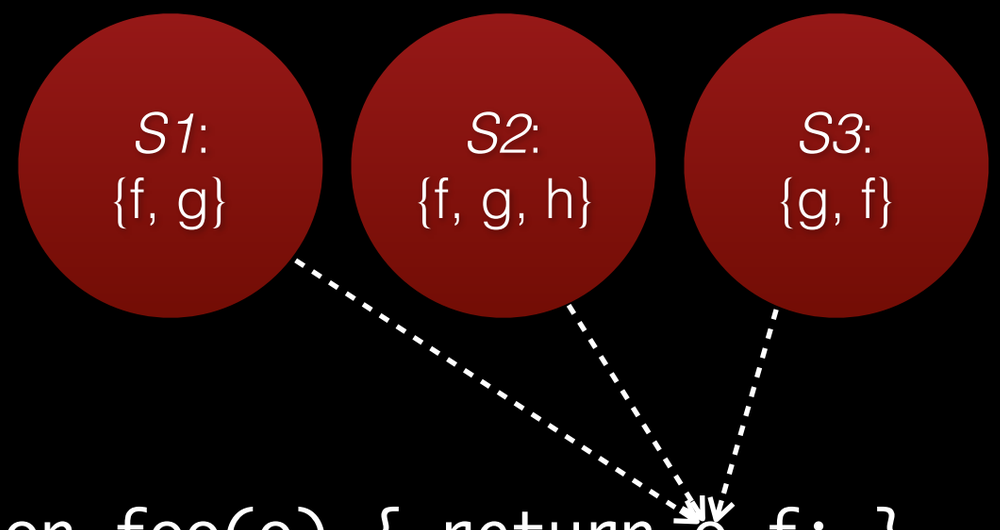
B3 IR:

```
if (o->structureID == S1 || o->structureID == S2)
    result = o->inlineStorage[0]
else
    result = o->inlineStorage[1]
```

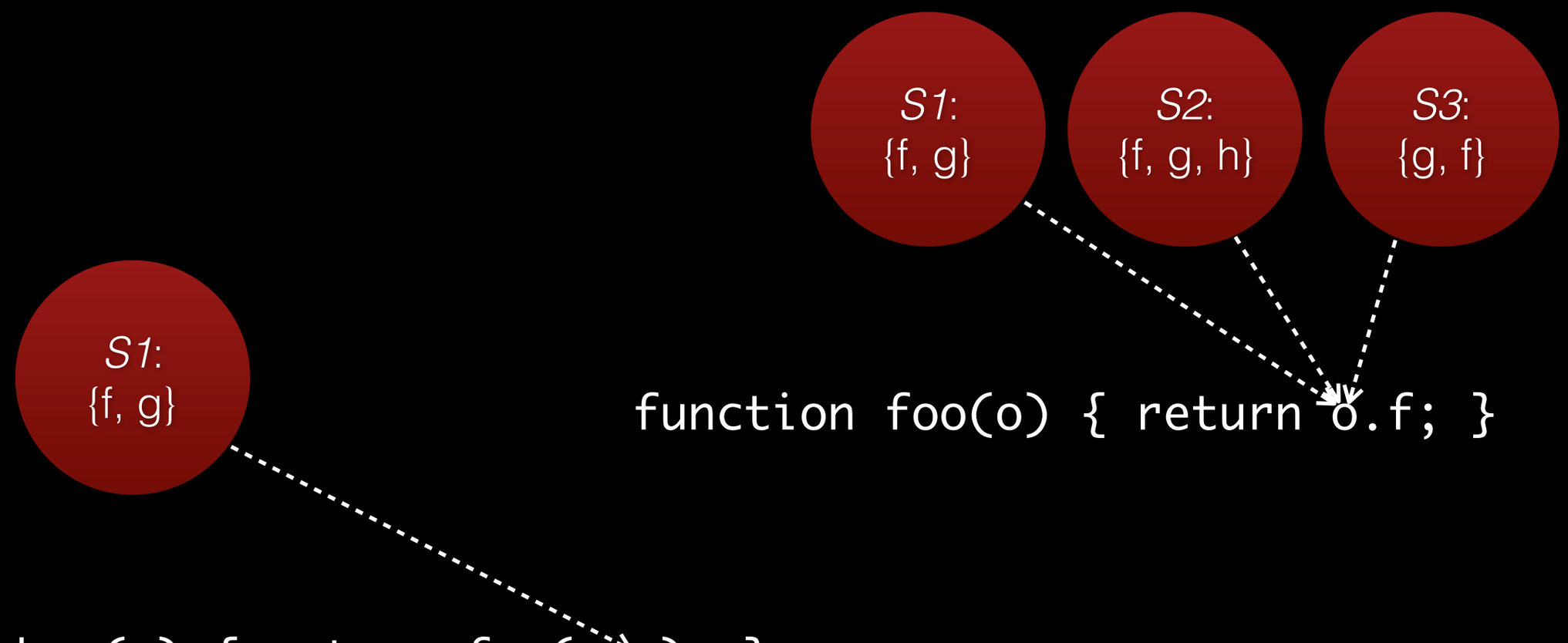
```
function foo(o) { return o.f; }
```

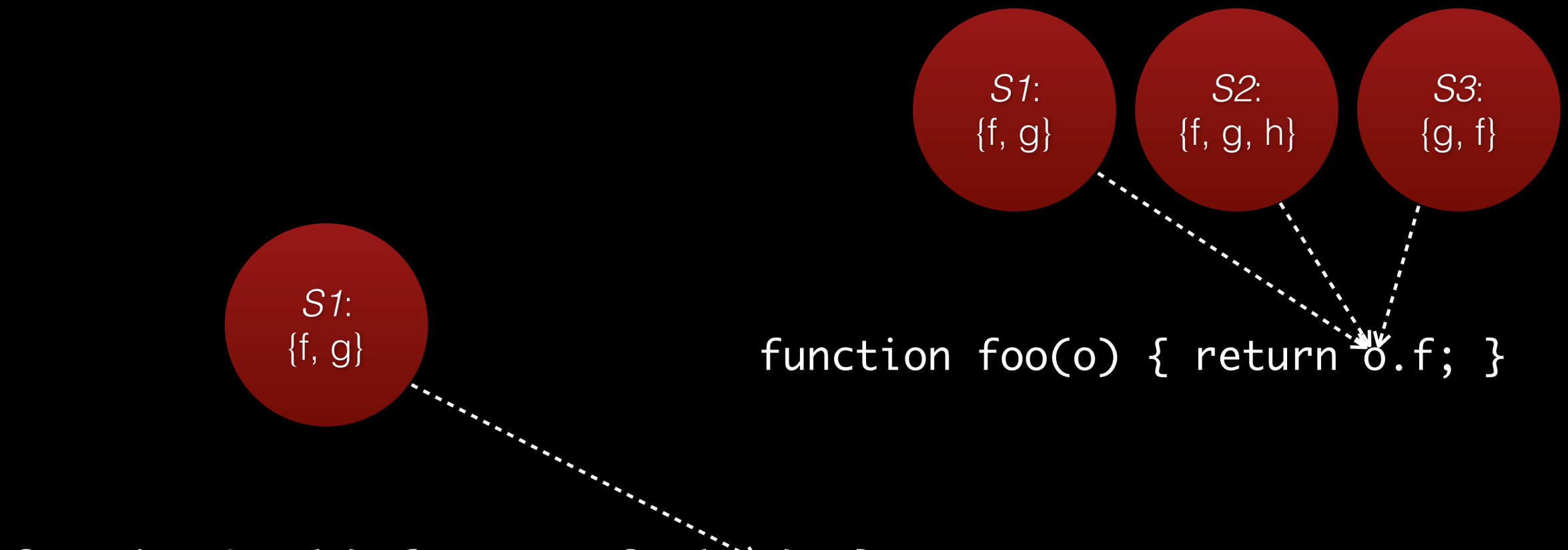



```
function foo(o) { return o.f; }
```

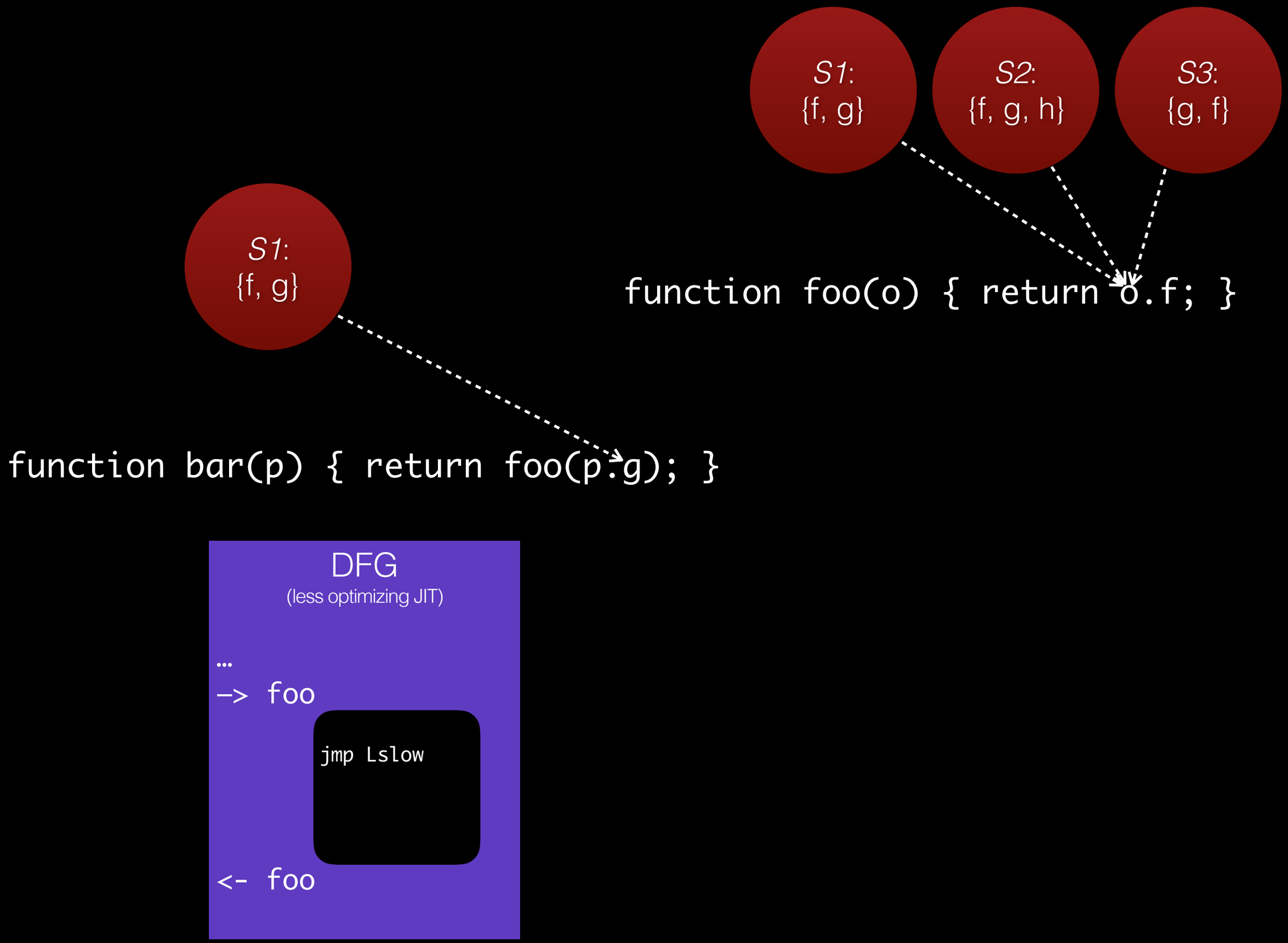


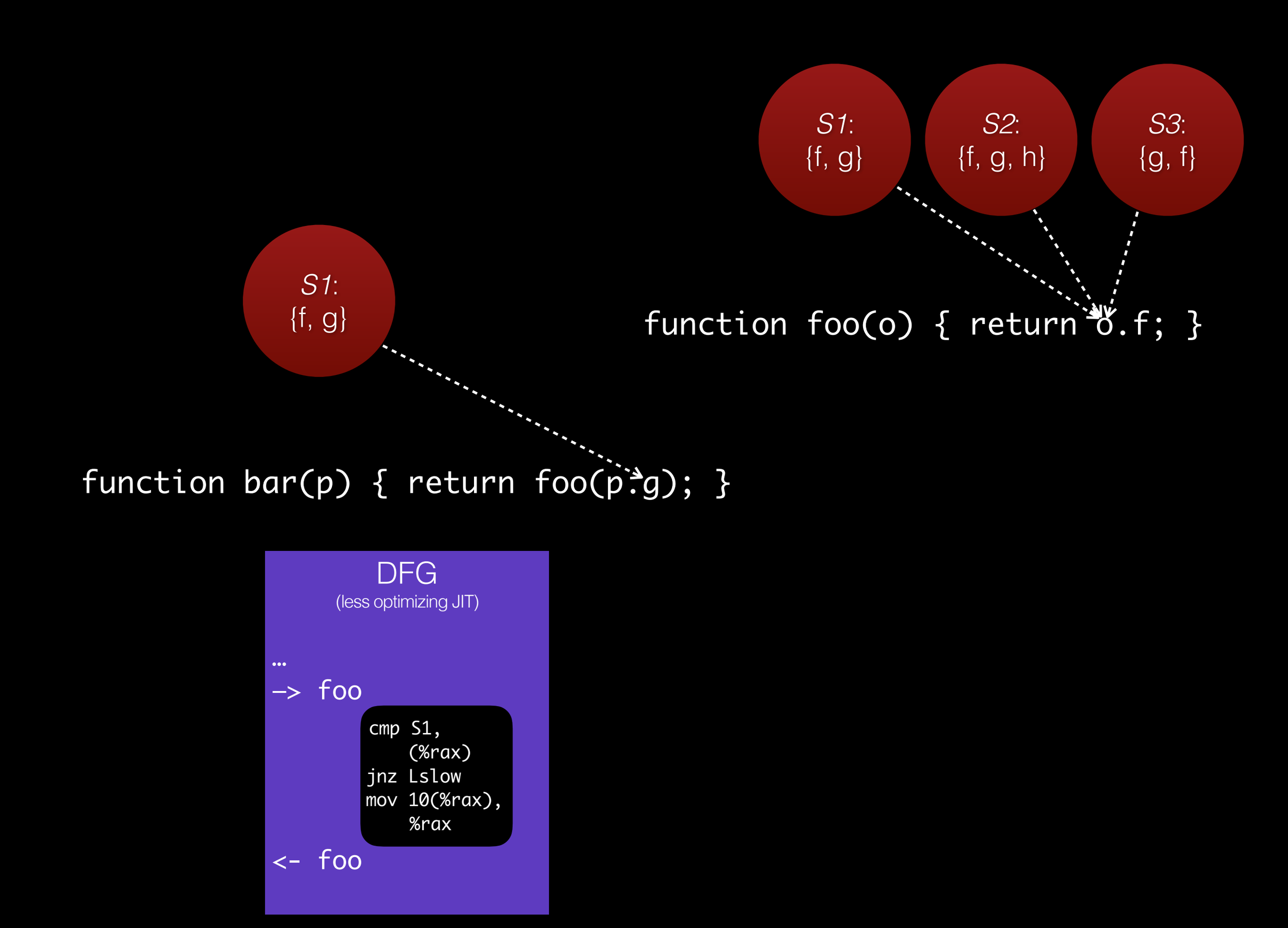

```
function foo(o) { return o.f; }
function bar(p) { return foo(p.g); }
```

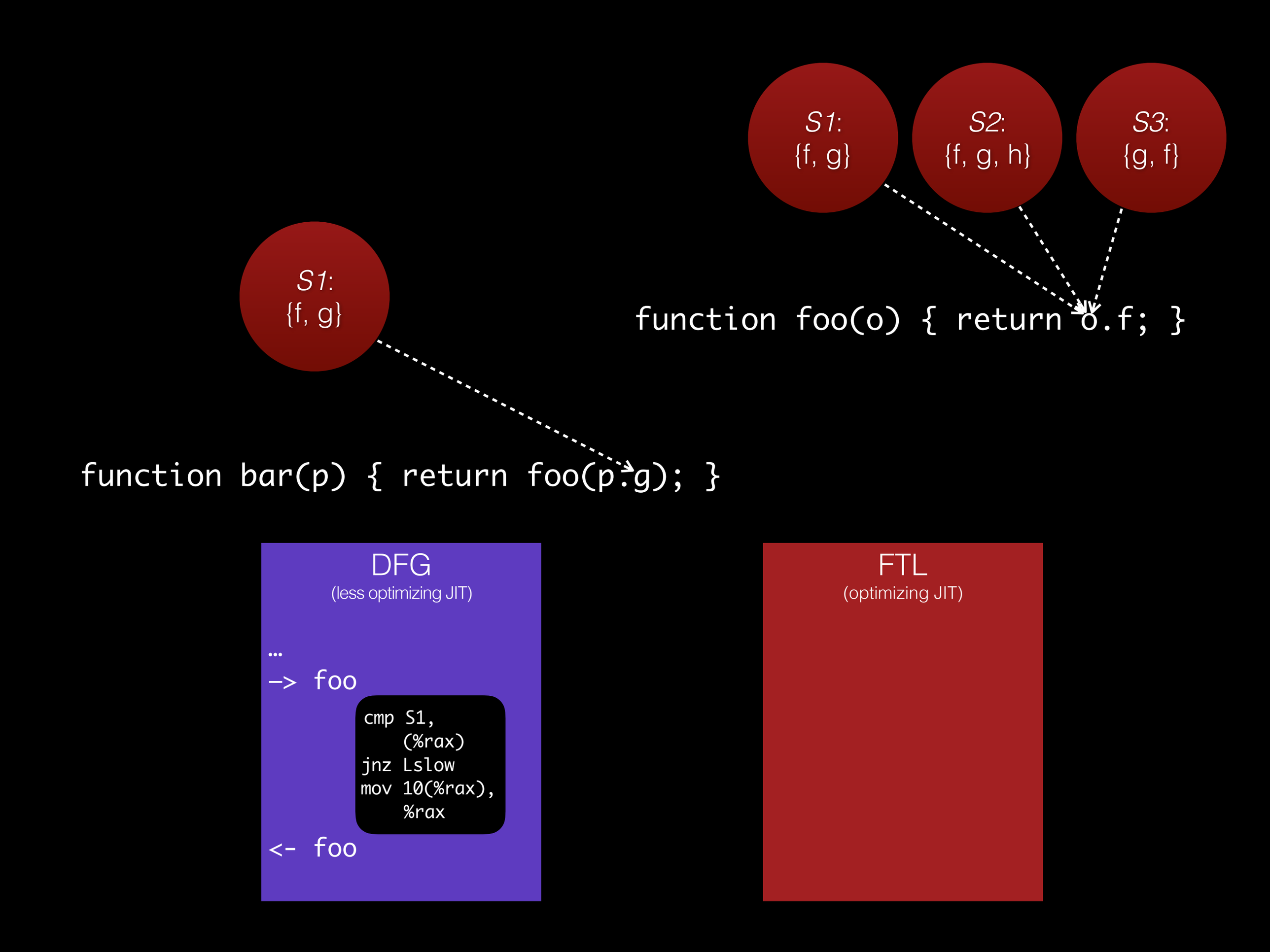

S1:
{f, g}
S2:
{f, g, h}
S3:
{g, f}
S1:
{f, g}
function foo(o) { return o.f; }
function bar(p) { return foo(p.g); }

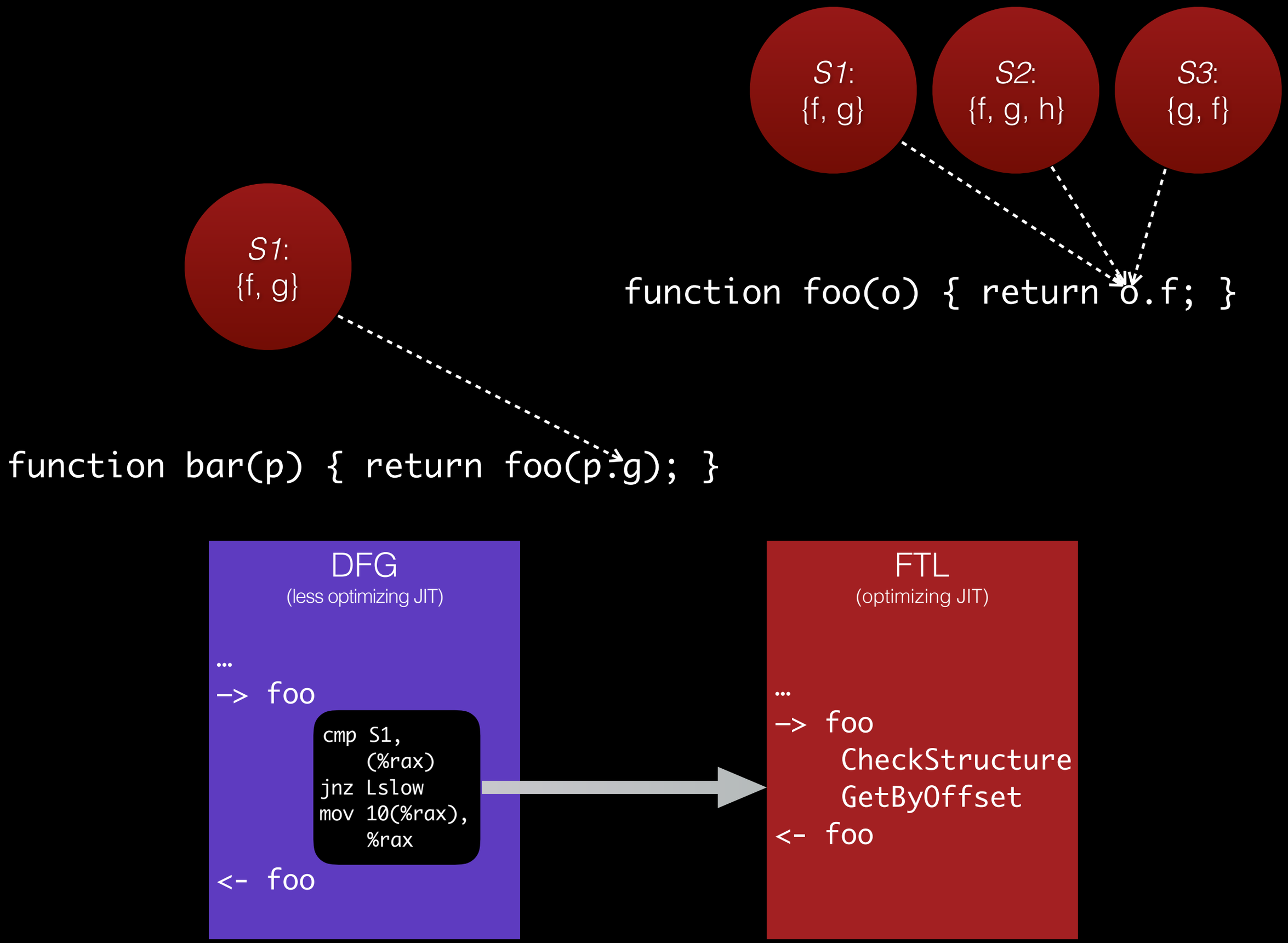


DFG
(less optimizing JIT)


S1:
{f, g}
S2:
{f, g, h}
S3:
{g, f}
function foo(o) { return o.f; }
S1:
{f, g}
function bar(p) { return foo(p.g); }
DFG
(less optimizing JIT)
...
-> foo
jmp Lslow
<- foo


S1:
{f, g}
S2:
{f, g, h}
S3:
{g, f}
function foo(o) { return o.f; }
S1:
{f, g}
function bar(p) { return foo(p.g); }
DFG
(less optimizing JIT)
…
-> foo
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
<- foo


S1:
{f, g}
S2:
{f, g, h}
S3:
{g, f}
function foo(o) { return o.f; }
S1:
{f, g}
function bar(p) { return foo(p.g); }
DFG
(less optimizing JIT)
…
-> foo
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
<- foo
FTL
(optimizing JIT)


S1:
{f, g}
S2:
{f, g, h}
S3:
{g, f}
function foo(o) { return o.f; }
S1:
{f, g}
function bar(p) { return foo(p.g); }
DFG
(less optimizing JIT)
…
-> foo
cmp S1,
(%rax)
jnz Lslow
mov 10(%rax),
%rax
<- foo
FTL
(optimizing JIT)
…
-> foo
CheckStructure
GetByOffset
<- foo

Inline Caches

- Great optimization
- Implicitly provides profiling data
- Polyvariant

Profiling Sources in JSC

- Case Flags — *branch speculation*
- Case Counts — *branch speculation*
- Value Profiling — *type inference of values*
- Inline Caches — *type inference of object structure*
- Watchpoints — *heap speculation*
- Exit Flags — *speculation backoff*

Watchpoints

```
Math.pow(42, 2)
```

Watchpoints

```
Math.pow(42, 2)
```

```
resolve_scope
get_from_scope
```

Watchpoints

```
Math.pow(42, 2)
```

```
resolve_scope
get_from_scope
get_by_id
```

Watchpoints

```
Math.pow(42, 2)
```

```
resolve_scope
get_from_scope
get_by_id
call
```

Watchpoints

```
Math.pow(42, 2)

resolve_scope
get_from_scope
get_by_id
call
```

Watchpoints

powfunc(42, 2)

const(*powfunc*)
call

Watchpoints

powfunc(42, 2)

Math = “wat”;

const(*powfunc*)

call

Watchpoints

```
Math.pow(42, 2)
```

```
resolve_scope Math = “wat”;
get_from_scope
get_by_id
call
```

Watchpoints Example #2

```
Strength.REQUIRED        = new Strength(0, "required");
Strength.STONG_PREFERRED = new Strength(1, "strongPreferred");
Strength.PREFERRED       = new Strength(2, "preferred");
Strength.STRONG_DEFAULT  = new Strength(3, "strongDefault");
Strength.NORMAL          = new Strength(4, "normal");
Strength.WEAK_DEFAULT    = new Strength(5, "weakDefault");
Strength.WEAKEST         = new Strength(6, "weakest");
```

Source: deltablue benchmark

Watchpoints Example #3

```
AST.prototype.typeCheck = function (typeFlow) {
    switch(this.nodeType) {
        case TypeScript.NodeType.Error:
        case TypeScript.NodeType.EmptyExpr: {
            this.type = typeFlow.anyType;
            break;
        …
```

Source: typescript compiler

Watchpoints Example #3

```
AST.prototype.typeCheck = function (typeFlow) {
    switch(this.nodeType) {
        case TypeScript.NodeType.Error:
        case TypeScript.NodeType.EmptyExpr: {
            this.type = typeFlow.anyType;
            break;
        …
```

Source: typescript compiler

Watchpoints

- Object Property Conditions (equality, presence, absence, etc)
  - *relies on structures and ICs*
- Lots of exotic watchpoints

Profiling Sources in JSC

- Case Flags — *branch speculation*
- Case Counts — *branch speculation*
- Value Profiling — *type inference of values*
- Inline Caches — *type inference of object structure*
- Watchpoints — *heap speculation*
- Exit Flags — *speculation backoff*

Exit Flags

Profiling	Speculation

`bool Graph::`
`canOptimizeStringObjectAccess(`
`const CodeOrigin& codeOrigin)`
`{`
`  if (hasExitSite(`
`  codeOrigin,`
`  NotStringObject))`
`  return false;`
 	
`  …`
	`void LowerDFGToB3::`
	`speculateStringObjectForStructureID`
	`(Edge edge, LValue structureID)`
	`{`
	`  …`
	
	`  speculate(`
	`  NotStringObject,`
	`  noValue(), 0,`
	`  m_out.notEqual(…));`
	`}`

Exit Flags

Profiling	Speculation

`bool Graph::`
`canOptimizeStringObjectAccess(`
`const CodeOrigin& codeOrigin)`
`{`
`  if (hasExitSite(`
`  codeOrigin,`
`  NotStringObject))`
`  return false;`
 	
`  …`
	`void LowerDFGToB3::`
	`speculateStringObjectForStructureID`
	`(Edge edge, LValue structureID)`
	`{`
	`  …`
	
	`  speculate(`
	`  NotStringObject,`
	`  noValue(), 0,`
	`  m_out.notEqual(…));`
	`}`

Exit Flags

Profiling	Speculation

`bool Graph::`
`canOptimizeStringObjectAccess(`
`const CodeOrigin& codeOrigin)`
`{`
`  if (hasExitSite(`
`  codeOrigin,`
`  NotStringObject))`
`  return false;`
 	
`  …`
	`void LowerDFGToB3::`
	`speculateStringObjectForStructureID`
	`(Edge edge, LValue structureID)`
	`{`
	`  …`
	
	`  speculate(`
	`  NotStringObject,`
	`  noValue(), 0,`
	`  m_out.notEqual(…));`
	`}`

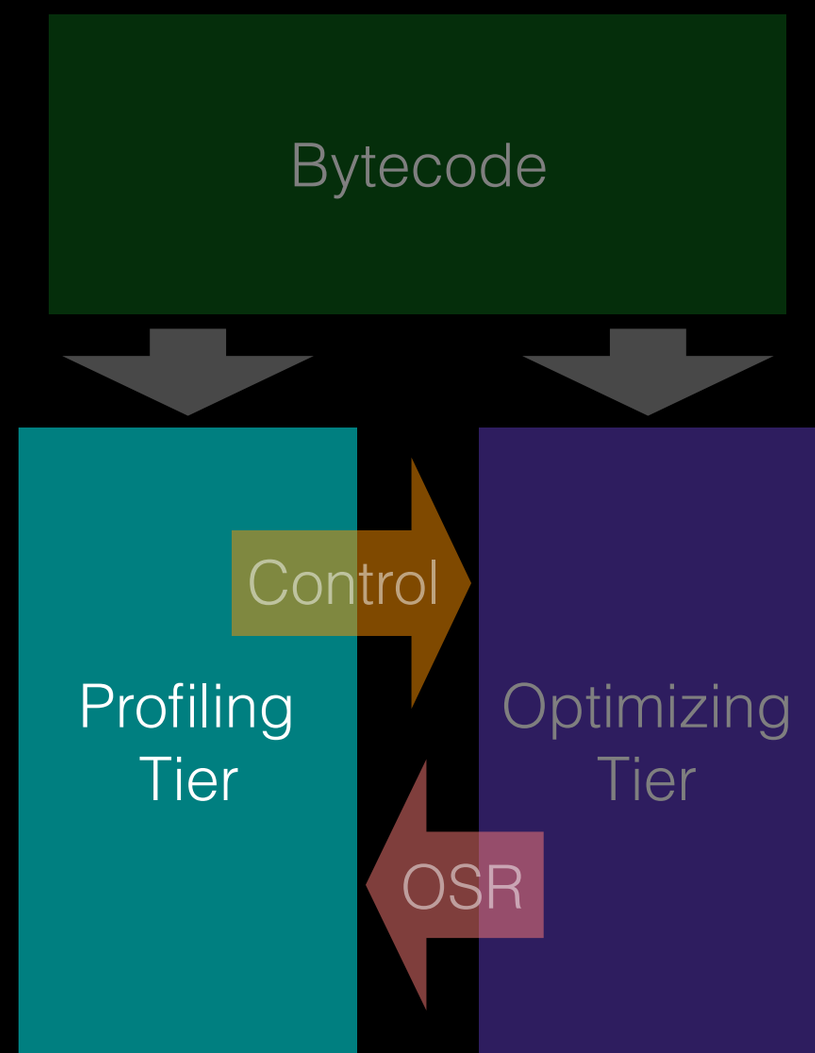
Exit Flags

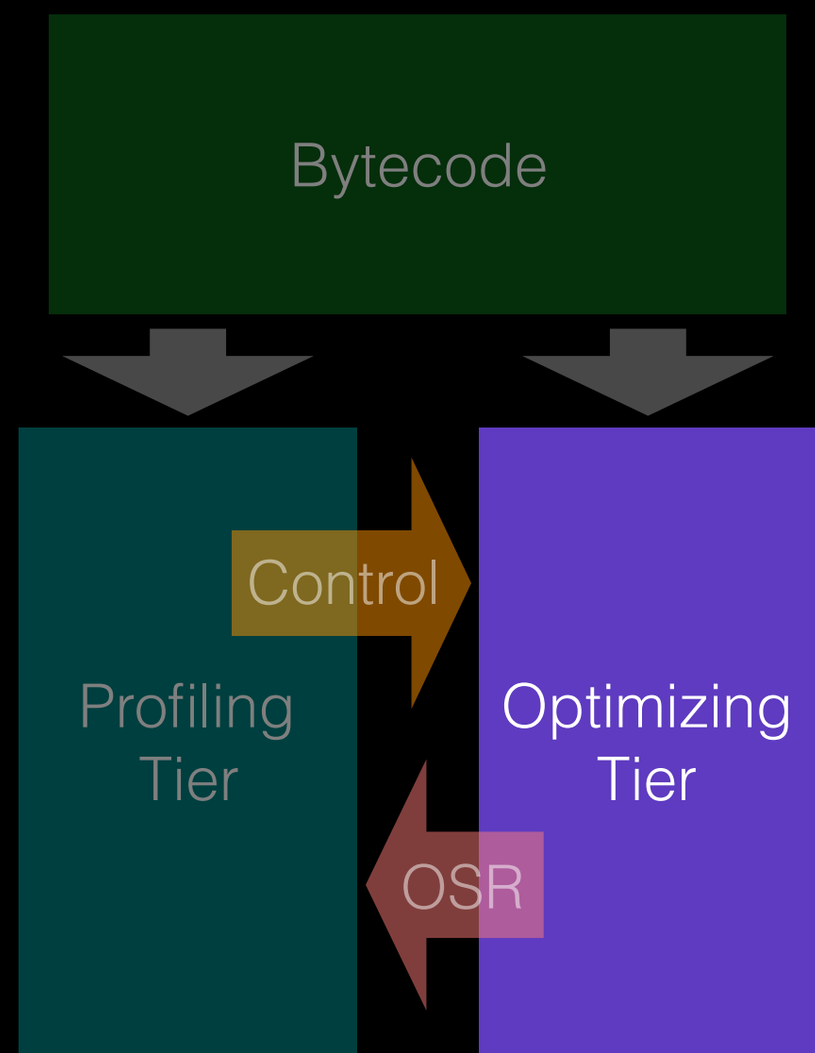
Profiling	Speculation

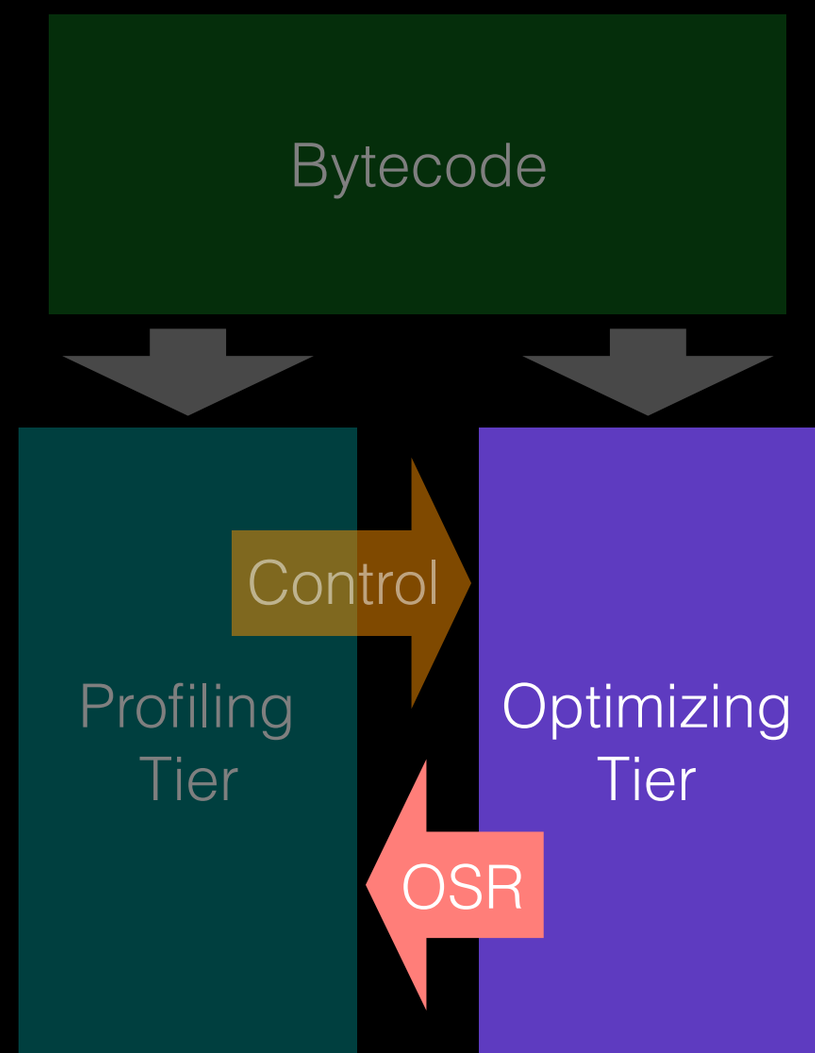
`bool Graph::`
`canOptimizeStringObjectAccess(`
`const CodeOrigin& codeOrigin)`
`{`
`  if (hasExitSite(`
`  codeOrigin,`
`  NotStringObject))`
`  return false;`
 	
`  …`
	`void LowerDFGToB3::`
	`speculateStringObjectForStructureID`
	`(Edge edge, LValue structureID)`
	`{`
	`  …`
	
	`  speculate(`
	`  NotStringObject,`
	`  noValue(), 0,`
	`  m_out.notEqual(…));`
	`}`

Profiling Sources in JSC

- Case Flags — *branch speculation*
- Case Counts — *branch speculation*
- Value Profiling — *type inference of values*
- Inline Caches — *type inference of object structure*
- Watchpoints — *heap speculation*
- Exit Flags — *speculation backoff*


Bytecode
Control
Profiling Tier
Optimizing Tier
OSR


Bytecode
Control
Profiling Tier
Optimizing Tier
OSR


Bytecode
Control
Profiling
Tier
Optimizing
Tier
OSR

DFG IR

Source

```
function foo(a, b)
{
    return a + b;
}
```

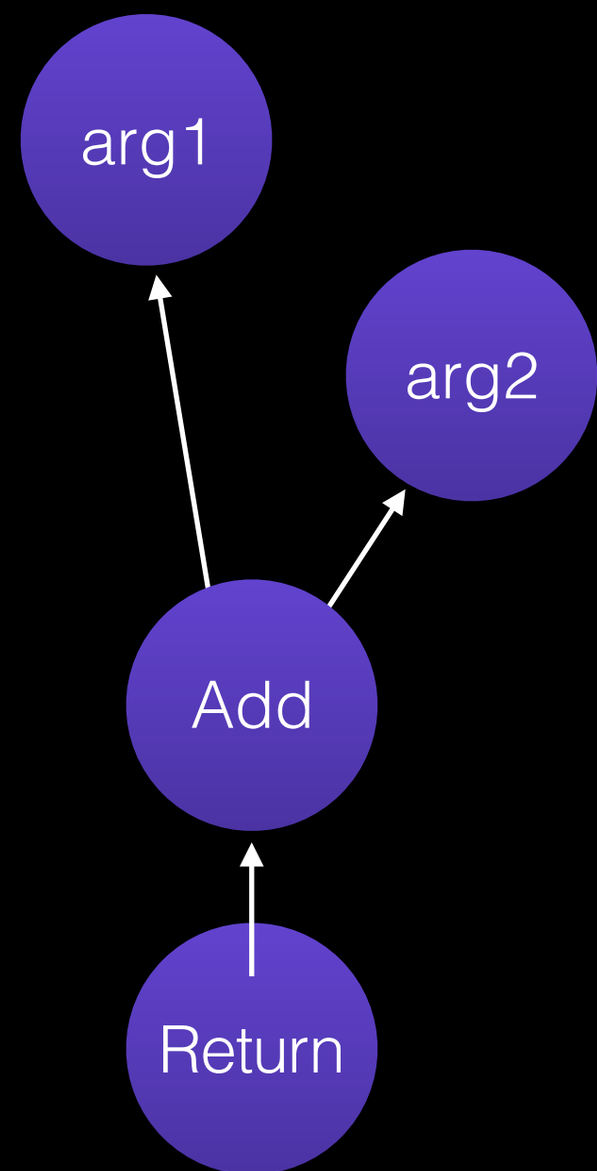
Bytecode

```
[    0] enter
[    1] get_scope          loc3
[    3] mov                loc4, loc3
[    6] check_traps
[    7] add                loc6, arg1, arg2
[   12] ret                loc6
```

Bytecode

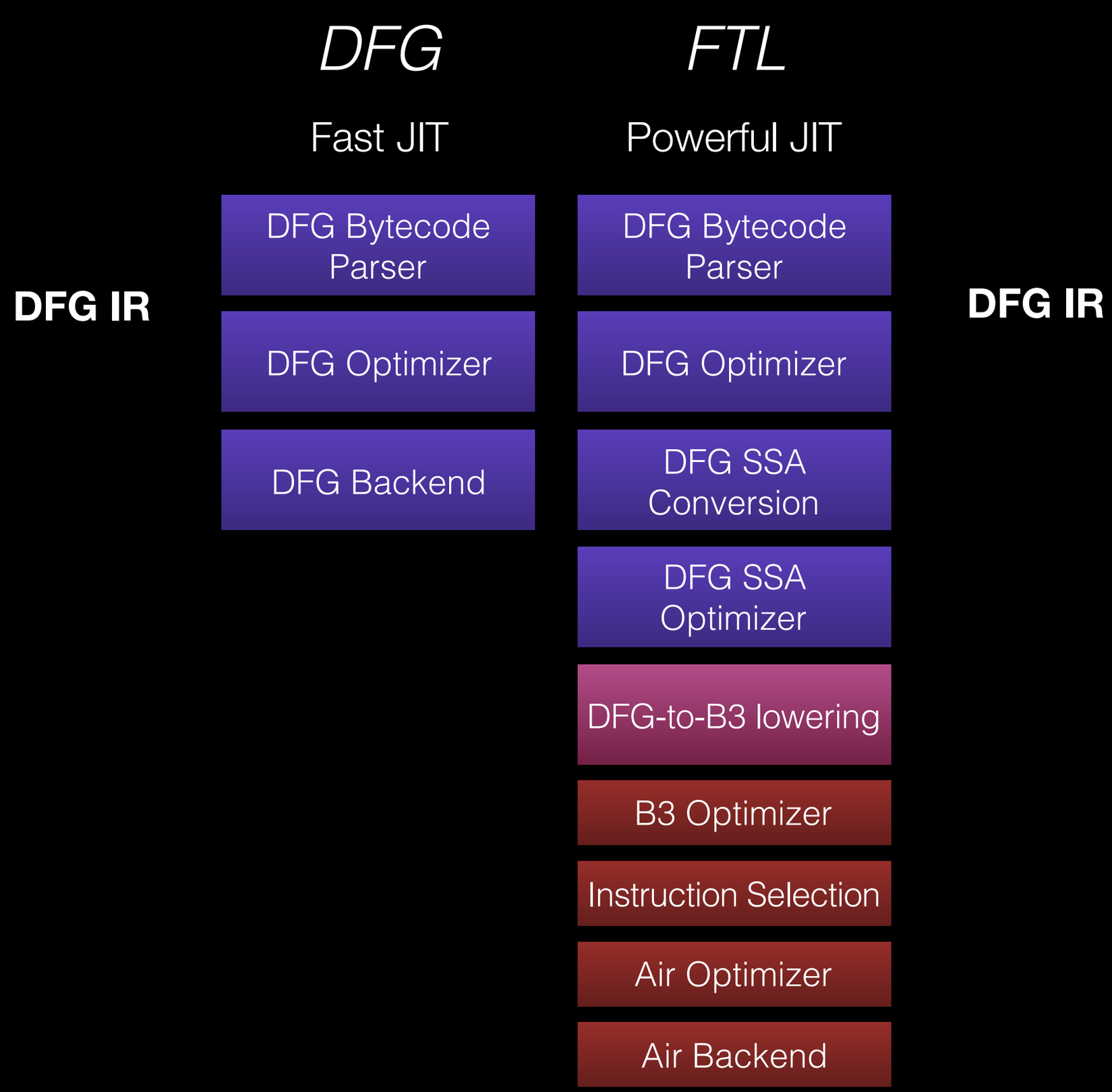
```
[    0] enter
[    1] get_scope          loc3
[    3] mov                loc4, loc3
[    6] check_traps
[    7] add                loc6, arg1, arg2
[   12] ret                loc6
```

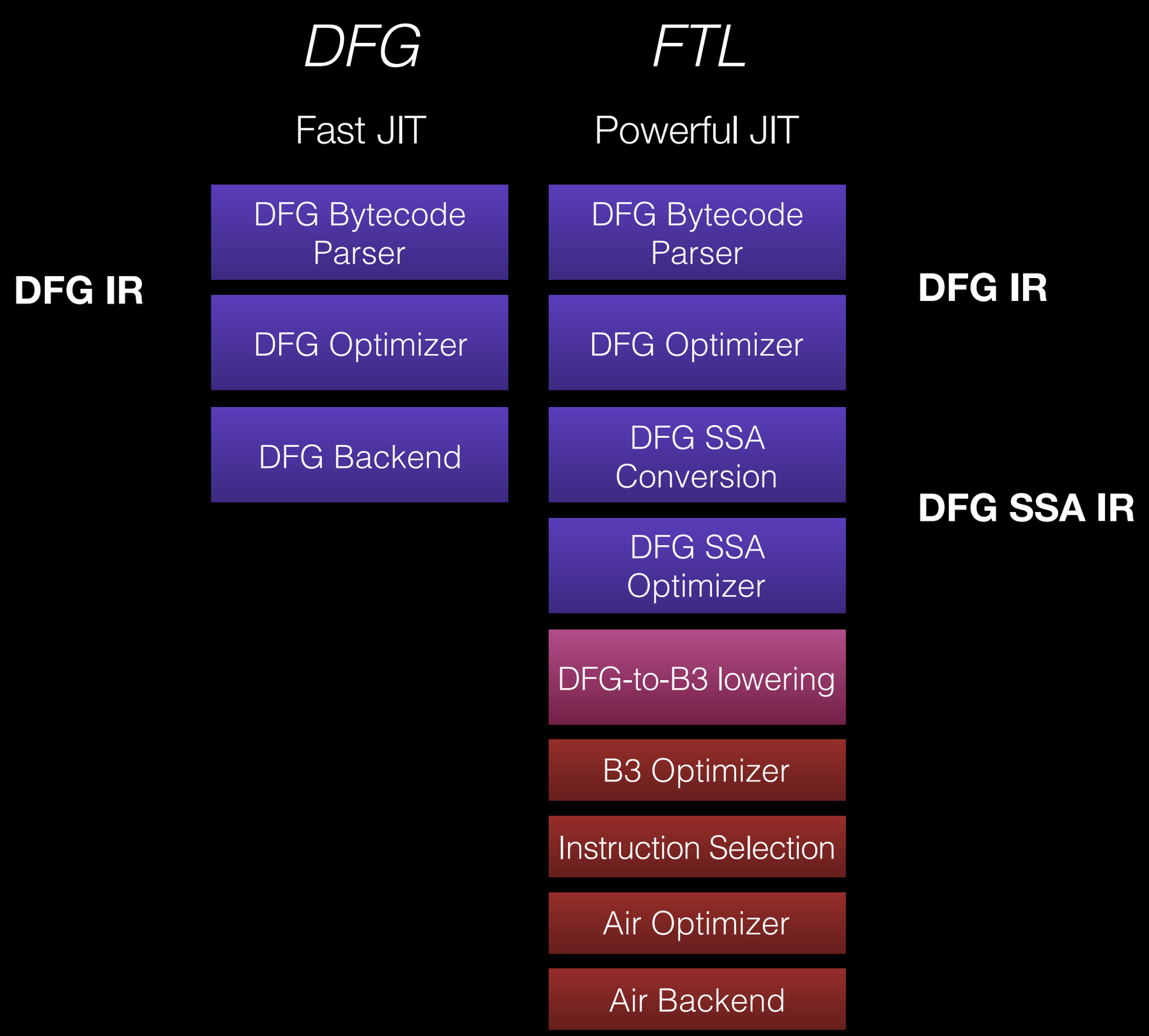
```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

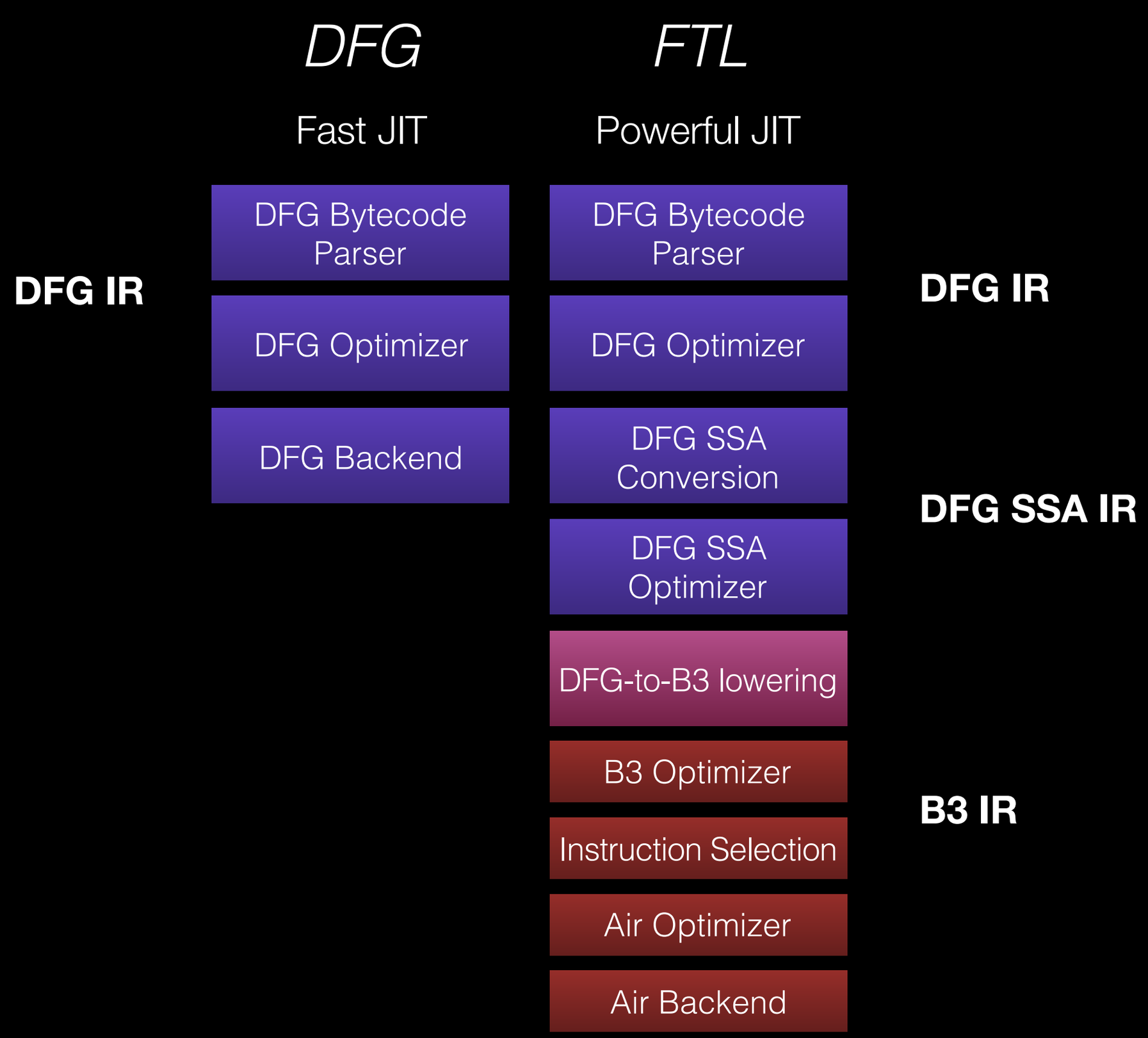


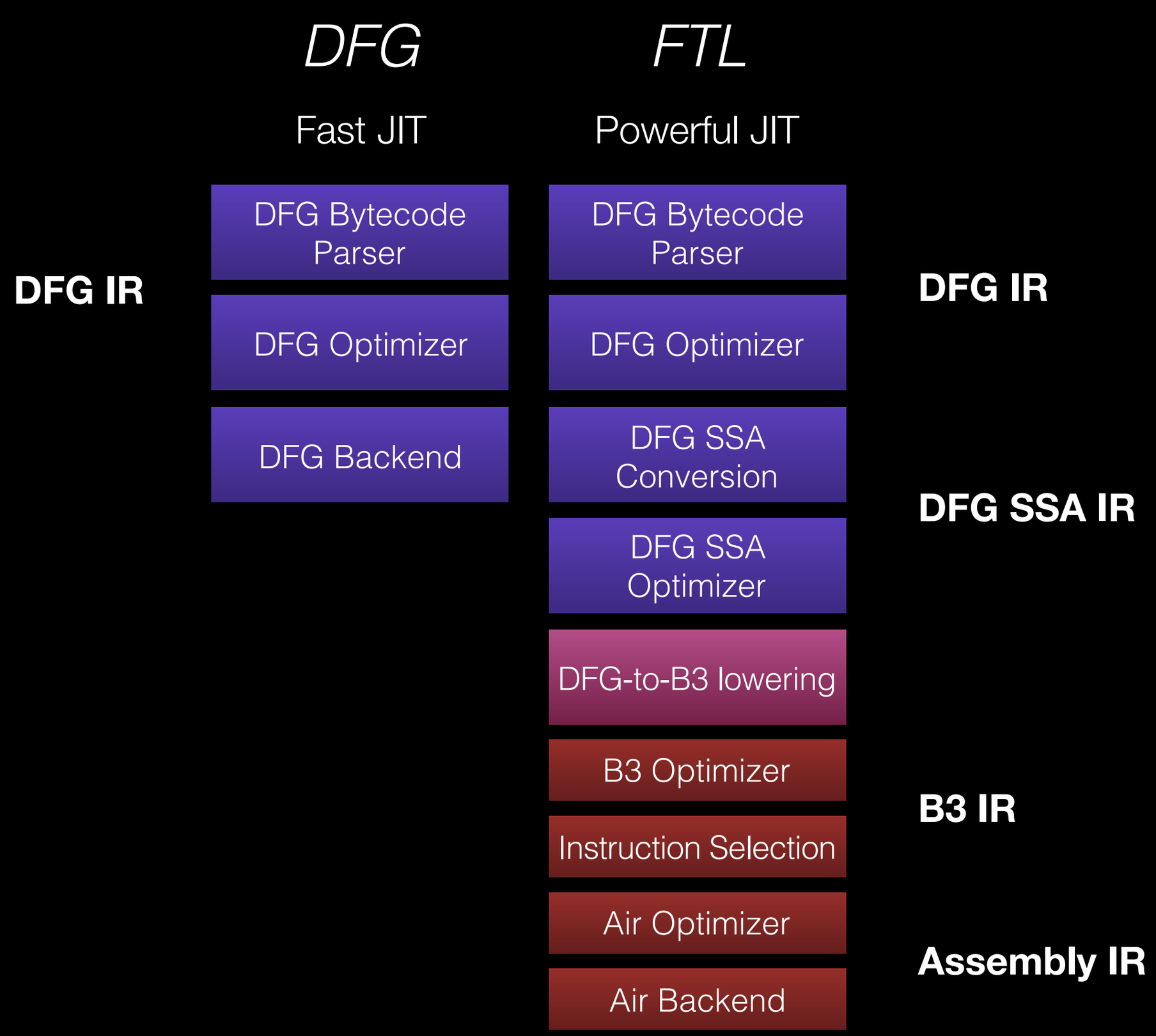
DFG	*FTL*
Fast JIT	Powerful JIT
DFG Bytecode Parser	DFG Bytecode Parser
DFG Optimizer	DFG Optimizer
DFG Backend	DFG SSA Conversion
	DFG SSA Optimizer
	DFG-to-B3 lowering
	B3 Optimizer
	Instruction Selection
	Air Optimizer
	Air Backend

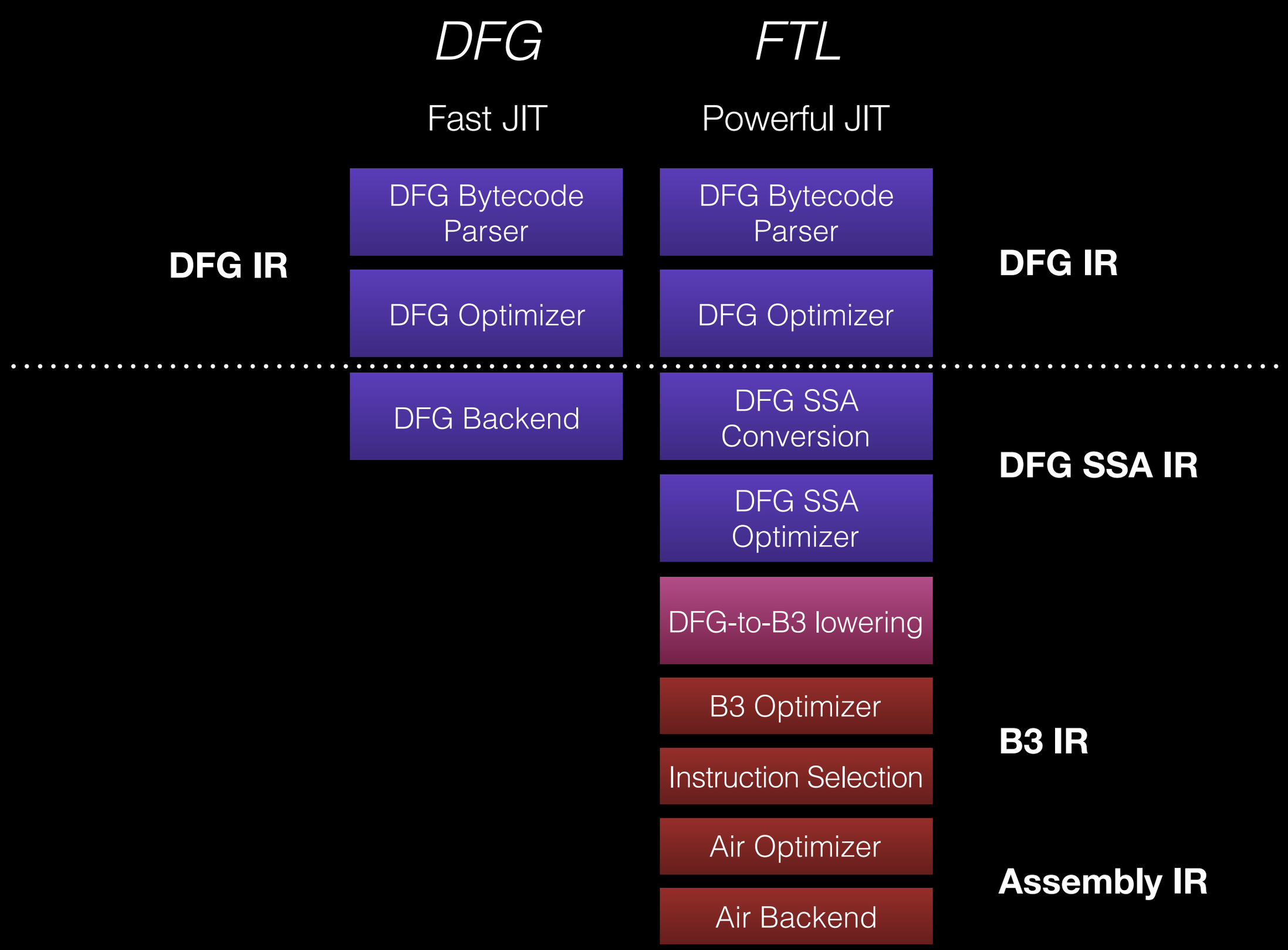
	DFG	*FTL*
	Fast JIT	Powerful JIT
DFG IR	DFG Bytecode Parser	DFG Bytecode Parser
	DFG Optimizer	DFG Optimizer
	DFG Backend	DFG SSA Conversion
		DFG SSA Optimizer
		DFG-to-B3 lowering
		B3 Optimizer
		Instruction Selection
		Air Optimizer
		Air Backend

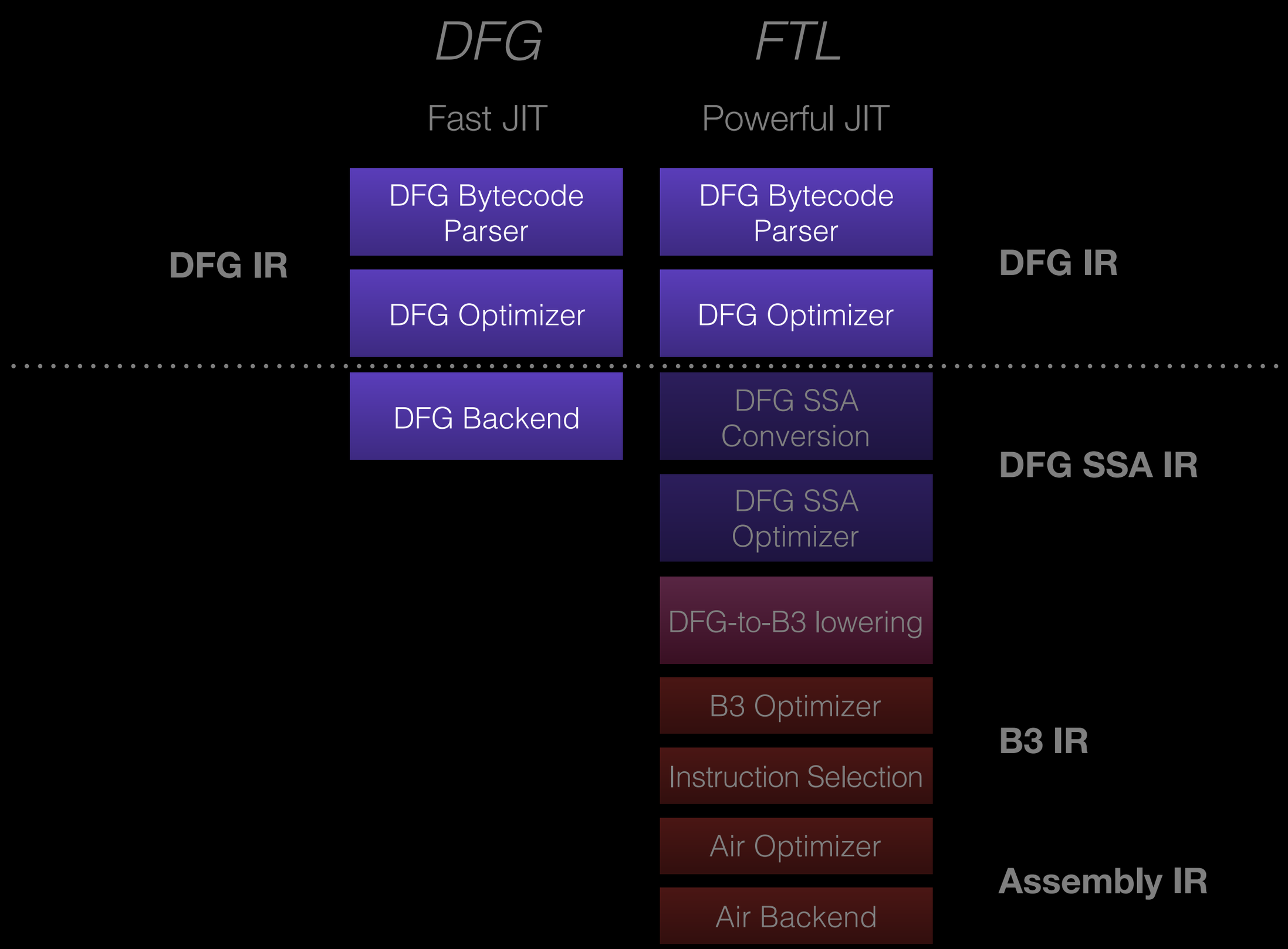

DFG
FTL
Fast JIT
Powerful JIT
DFG IR
DFG Bytecode Parser
DFG Optimizer
DFG Backend
DFG Bytecode Parser
DFG Optimizer
DFG SSA Conversion
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
Instruction Selection
Air Optimizer
Air Backend
DFG IR


DFG
FTL
Fast JIT
Powerful JIT
DFG IR
DFG Bytecode Parser
DFG Optimizer
DFG Backend
DFG Bytecode Parser
DFG Optimizer
DFG SSA Conversion
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
Instruction Selection
Air Optimizer
Air Backend
DFG IR
DFG SSA IR


DFG
FTL
Fast JIT
Powerful JIT
DFG IR
DFG Bytecode Parser
DFG Optimizer
DFG Backend
DFG Bytecode Parser
DFG Optimizer
DFG SSA Conversion
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
Instruction Selection
Air Optimizer
Air Backend
DFG IR
DFG SSA IR
B3 IR


DFG
FTL
Fast JIT
Powerful JIT
DFG IR
DFG Bytecode Parser
DFG Optimizer
DFG Backend
DFG Bytecode Parser
DFG Optimizer
DFG SSA Conversion
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
Instruction Selection
Air Optimizer
Air Backend
DFG IR
DFG SSA IR
B3 IR
Assembly IR


DFG
FTL
Fast JIT
Powerful JIT
DFG Bytecode Parser
DFG Bytecode Parser
DFG IR
DFG IR
DFG Optimizer
DFG Optimizer
DFG Backend
DFG SSA Conversion
DFG SSA IR
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
B3 IR
Instruction Selection
Air Optimizer
Assembly IR
Air Backend


DFG
FTL
Fast JIT
Powerful JIT
DFG Bytecode Parser
DFG Bytecode Parser
DFG IR
DFG IR
DFG Optimizer
DFG Optimizer
DFG Backend
DFG SSA Conversion
DFG SSA IR
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
B3 IR
Instruction Selection
Air Optimizer
Assembly IR
Air Backend

DFG Goal

Remove lots of type checks quickly.

DFG Goals

- Speculation
- Static Analysis
- Fast Compilation

DFG Goals

- Speculation
- Static Analysis
- Fast Compilation

DFG IR

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

DFG IR

profiling

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

DFG IR

speculation

profiling

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

DFG IR

speculation

profiling

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

OSR

OSR flattens control flow

OSR is *hard*

```
int foo(int* ptr)
{
    int w, x, y, z;

    w = … // lots of stuff


    x = is_ok(ptr) ? *ptr : slow_path(ptr);

    y = … // lots of stuff

    z = is_ok(ptr) ? *ptr : slow_path(ptr);

    return w + x + y + z;
}
```

```
int foo(int* ptr)
{
    int w, x, y, z;

    w = … // lots of stuff

    if (!is_ok(ptr))
        return foo_base1(ptr, w);
    x = *ptr;

    y = … // lots of stuff

    z = *ptr;

    return w + x + y + z;
}
```

```
int foo(int* ptr)
{
    int w, x, y, z;

    w = … // lots of stuff

    if (!is_ok(ptr))
        return foo_base1(ptr, w);
    x = *ptr;

    y = … // lots of stuff

    z = *ptr;

    return w + x + y + z;
}
```

- Must know where to exit.
- Must know what is live-at-exit.

```
[  42] add loc7, loc4, loc8
```

live after: loc3, loc4, loc7

```
[  42] add loc7, loc4, loc8
```

live after: loc3, loc4, loc7

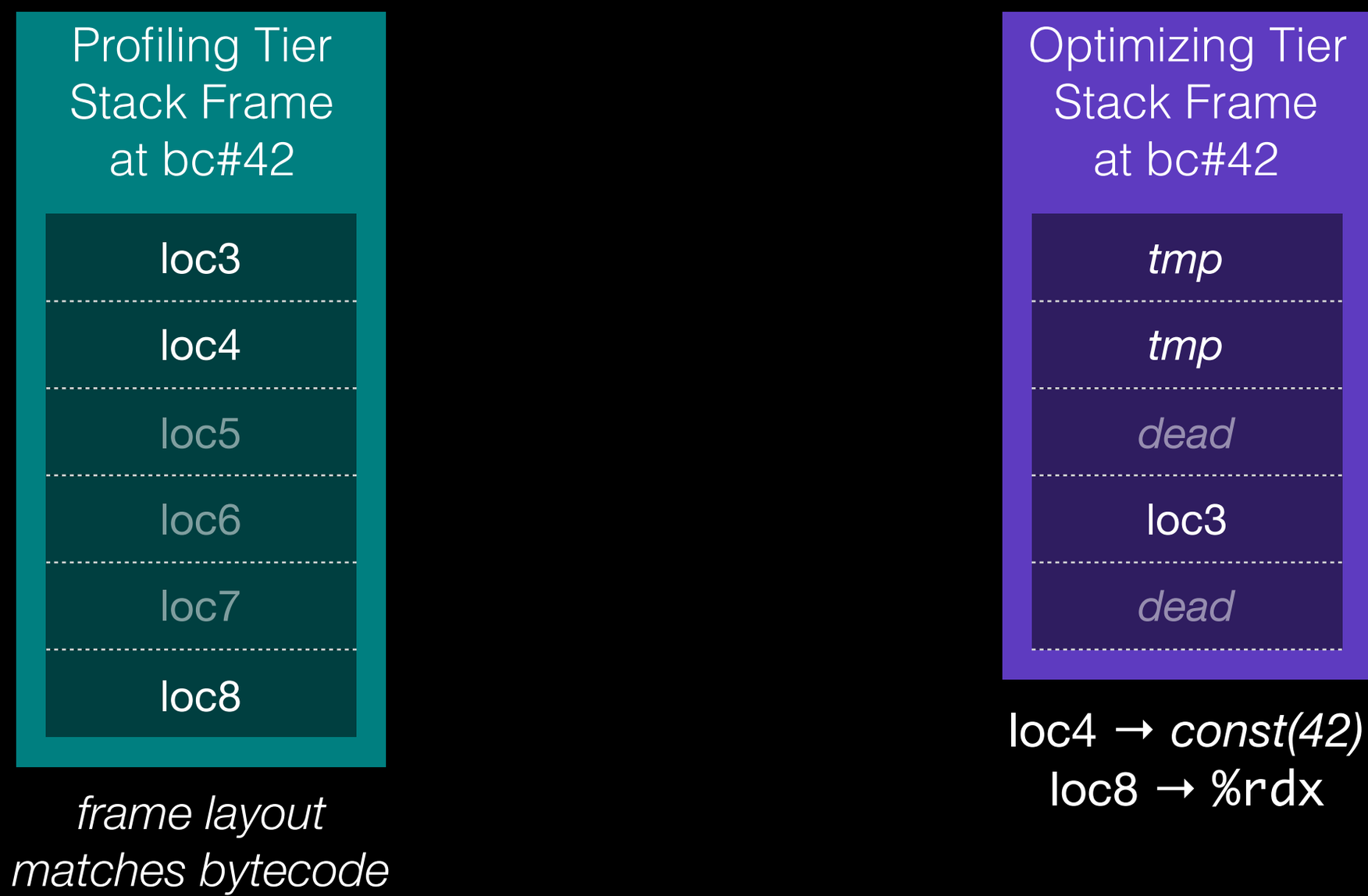
```
[  42] add loc7, loc4, loc8
```

live after: loc3, loc4, loc7

frame layout matches bytecode

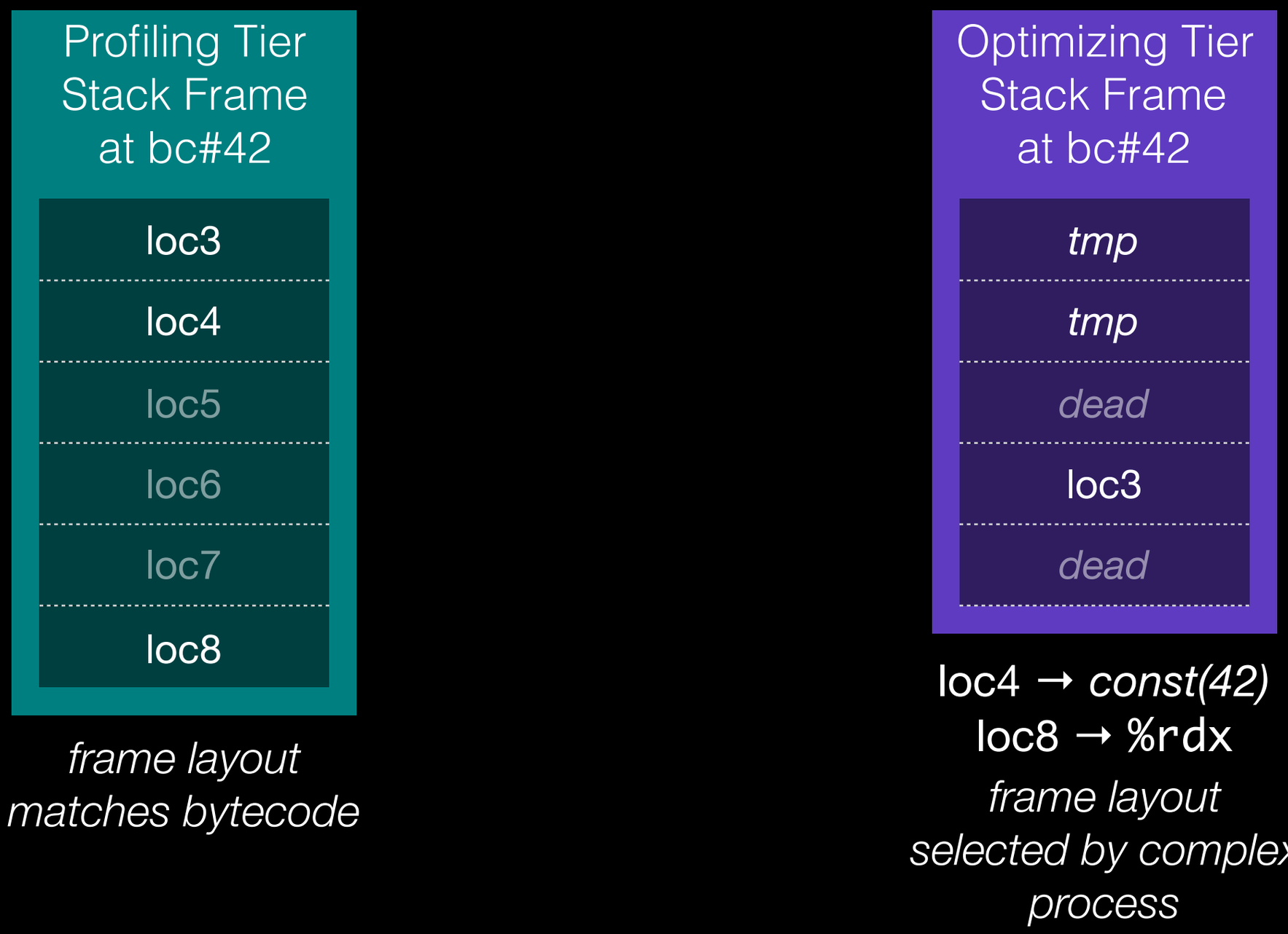
`[  42] add loc7, loc4, loc8`

live after: loc3, loc4, loc7



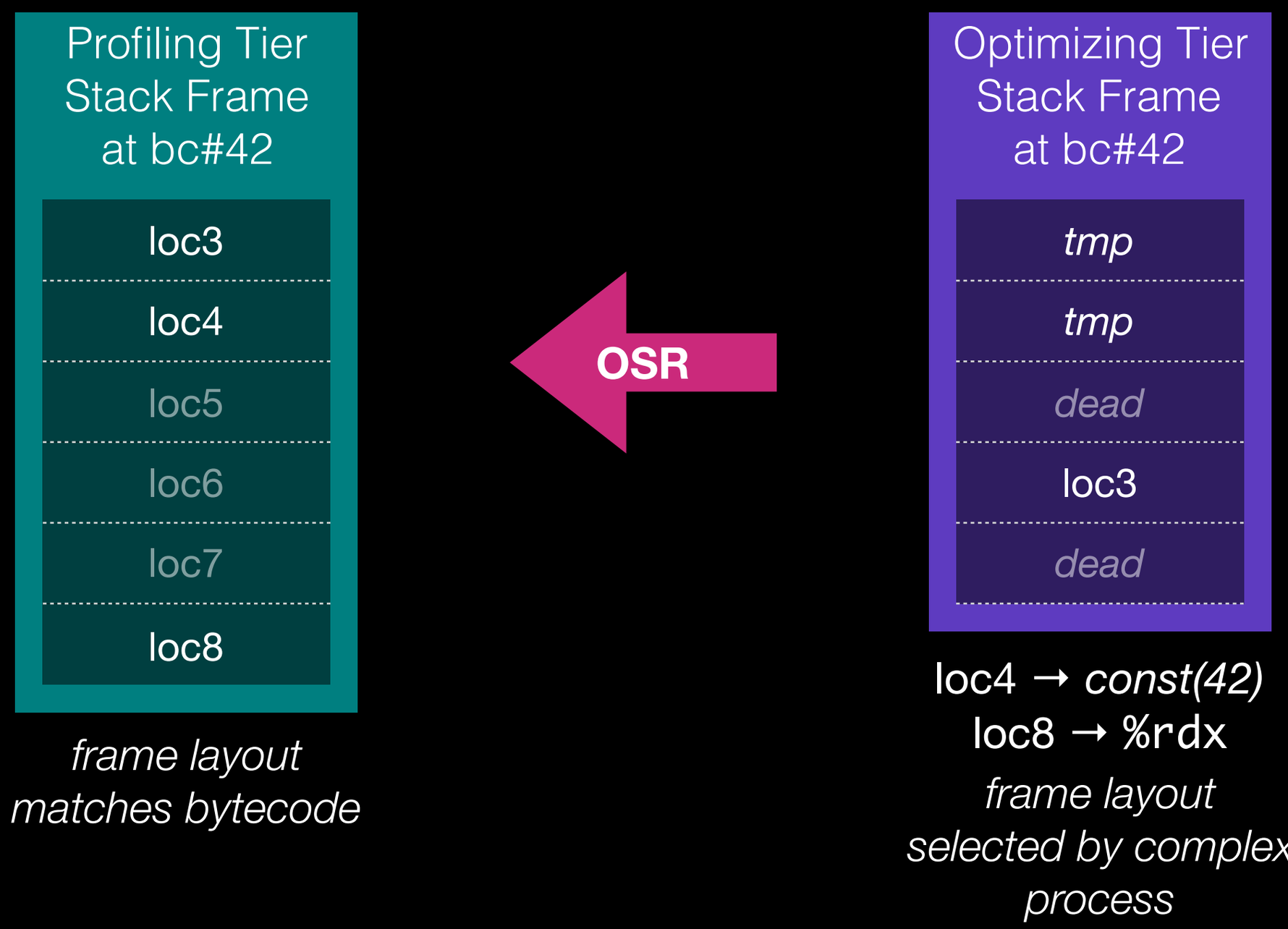
`[  42] add loc7, loc4, loc8`

live after: loc3, loc4, loc7



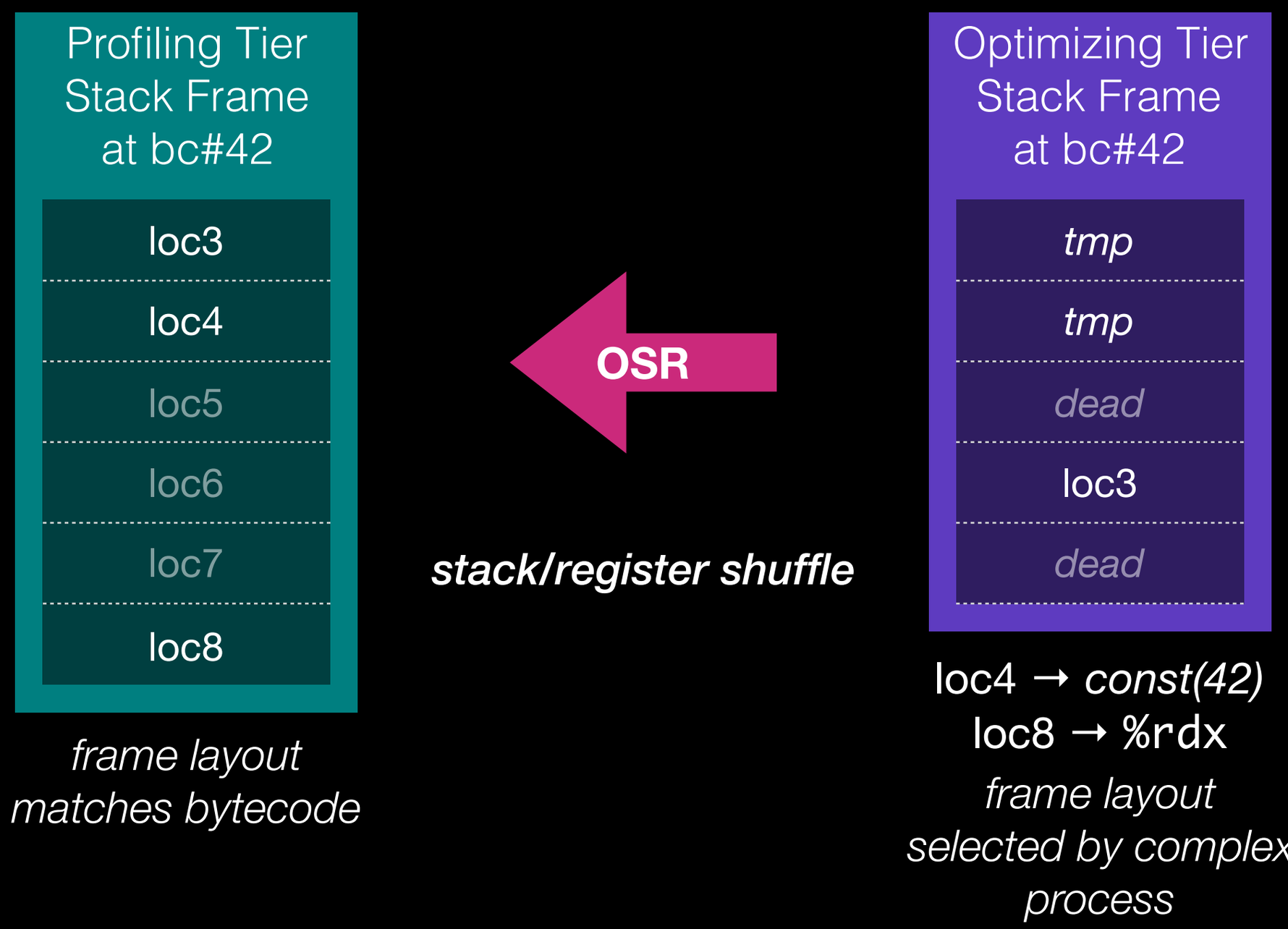
`[  42] add loc7, loc4, loc8`

live after: loc3, loc4, loc7



`[  42] add loc7, loc4, loc8`

live after: loc3, loc4, loc7



How?

Leverage Bytecode→SSA Conversion

```
[  42] add loc7, loc4, loc8
```

live after: loc3, loc4, loc7

[42] add loc7, loc4, loc8

live after: loc3, loc4, loc7

```
case op_add: {
    VirtualRegister result = instruction->result();
    VirtualRegister left   = instruction->left();
    VirtualRegister right  = instruction->right();

    stackMap[result] = createAdd(
        stackMap[left], stackMap[right]);
    break;
}
```

[42] add loc7, loc4, loc8

live after: loc3, loc4, loc7

```
case op_add: {
    VirtualRegister result = instruction->result();
    VirtualRegister left   = instruction->left();
    VirtualRegister right  = instruction->right();

    stackMap[result] = createAdd(
        stackMap[left], stackMap[right]);
    break;
}
```

stackMap *before bc#42*

Virtual Register	Value
loc3	GetScope
loc4	JSConstant(42)
loc5	*dead*
loc6	*dead*
loc7	*dead*
loc8	GetByOffset

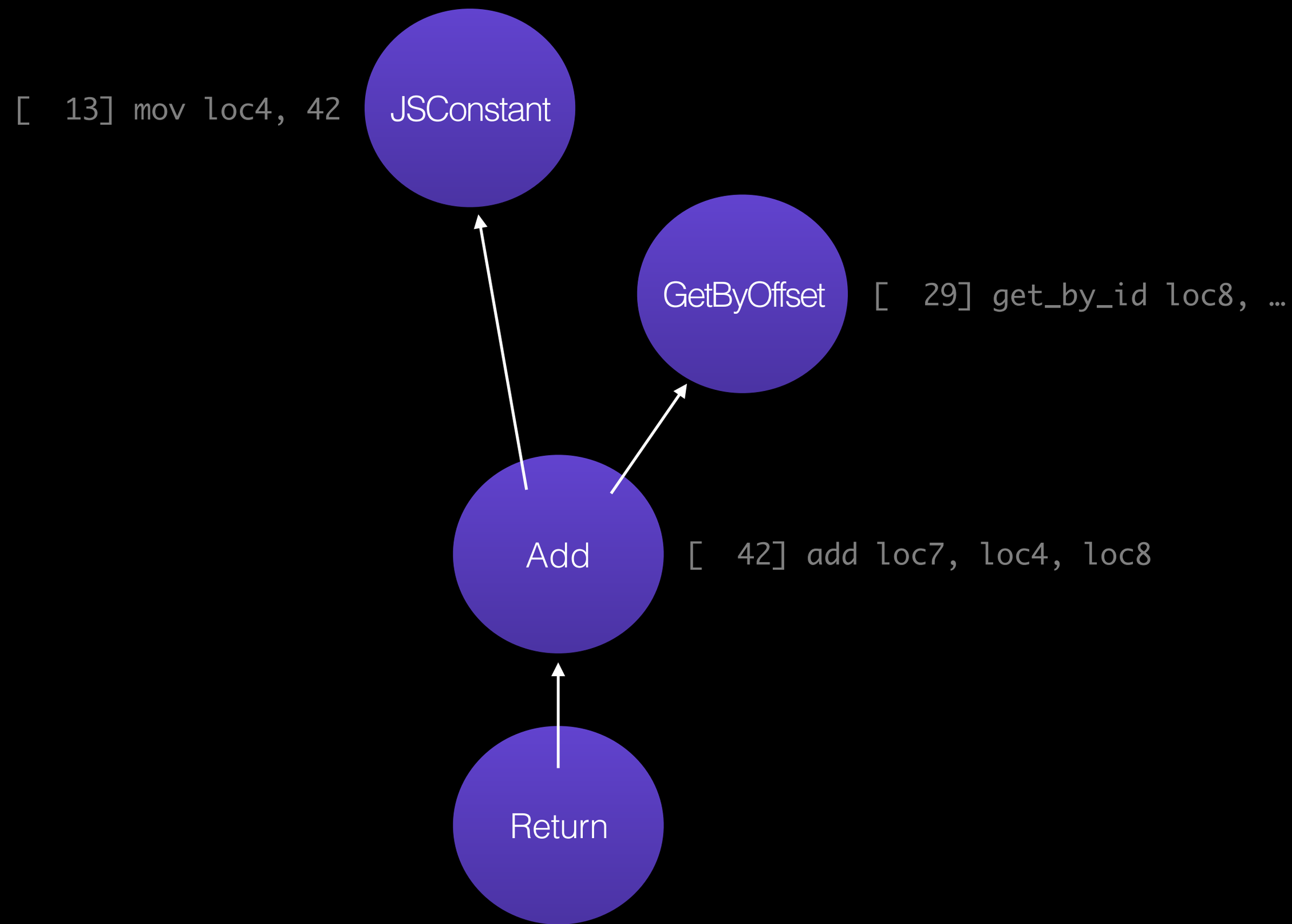
[42] add loc7, loc4, loc8

live after: loc3, loc4, loc7

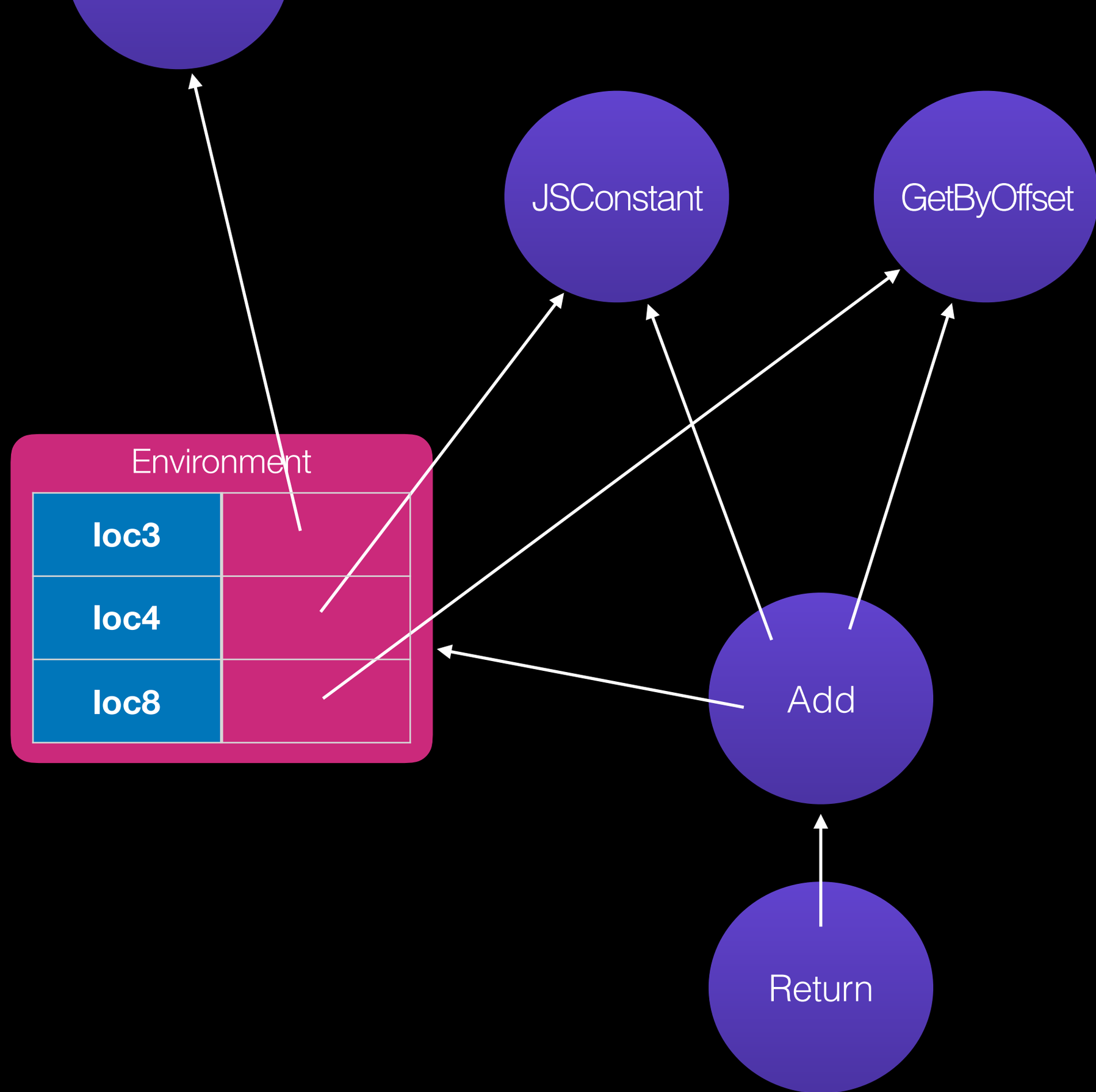
```
case op_add: {
    VirtualRegister result = instruction->result();
    VirtualRegister left   = instruction->left();
    VirtualRegister right  = instruction->right();

    stackMap[result] = createAdd(
        stackMap[left], stackMap[right]);
    break;
}
```

stackMap *after bc#42*

Virtual Register	Value
loc3	GetScope
loc4	JSConstant(42)
loc5	*dead*
loc6	*dead*
loc7	Add
loc8	*dead*


[13] mov loc4, 42
JSConstant
GetByOffset
[29] get_by_id loc8, …
Add
[42] add loc7, loc4, loc8
Return

```
case op_add: {
    VirtualRegister result = instruction->result();
    VirtualRegister left   = instruction->left();
    VirtualRegister right  = instruction->right();

    Map<VirtualRegister, Value*> environment;
    for (VirtualRegister reg : liveNow())
        environment[reg] = stackMap[reg];

    stackMap[reg] = createAdd(
        stackMap[left], stackMap[right],
        environment);
    break;
}
```


JSConstant
GetByOffset
Environment
loc3
loc4
loc8
Add
Return

Exit Environment

Exit Environment

- The obvious solution.

Exit Environment

- The obvious solution.
- Super widespread.

Exit Environment

- The obvious solution.
- Super widespread.
- *But it's awful for JavaScript!*

Exit Frequency	Environments Work?

Exit Frequency	Environments Work?
Seldom *(like inlined calls in Java)*	**Yes, they work great!** O(live variables) cost is incurred seldom, so it's not a big deal.

Exit Frequency	Environments Work?
Seldom *(like inlined calls in Java)*	**Yes, they work great!** O(live variables) cost is incurred seldom, so it's not a big deal.
Multiple Exits Per Bytecode Instruction *(like JavaScript)*	**Not really.** O(live variables) per instruction is a lot of: - data flow edges - memory

Observation:

environments hardly change between instructions.

Use delta encoding!

Use imperative delta encoding!

DFG IR

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

```
[   7] add                    loc6, arg1, arg2
```

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```

```
[    7] add                    loc6, arg1, arg2
```

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```

```
[    7] add                    loc6, arg1, arg2
```

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```

```
[    7] add                    loc6, arg1, arg2
```

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```

```
[    7] add                    loc6, arg1, arg2
```

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```

```
[   7] add                    loc6, arg1, arg2
```

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```

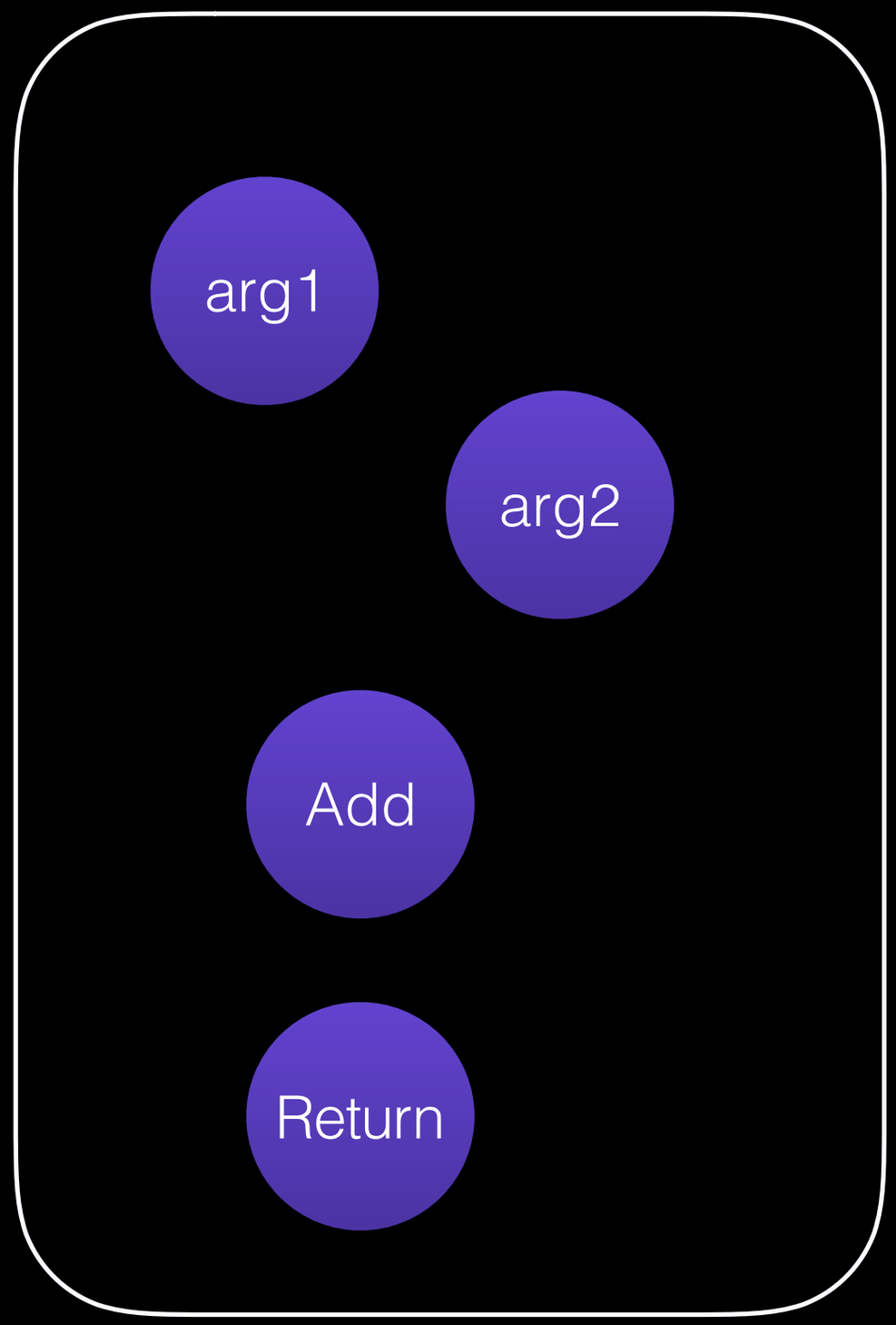
```
[   7] add                    loc6, arg1, arg2
```

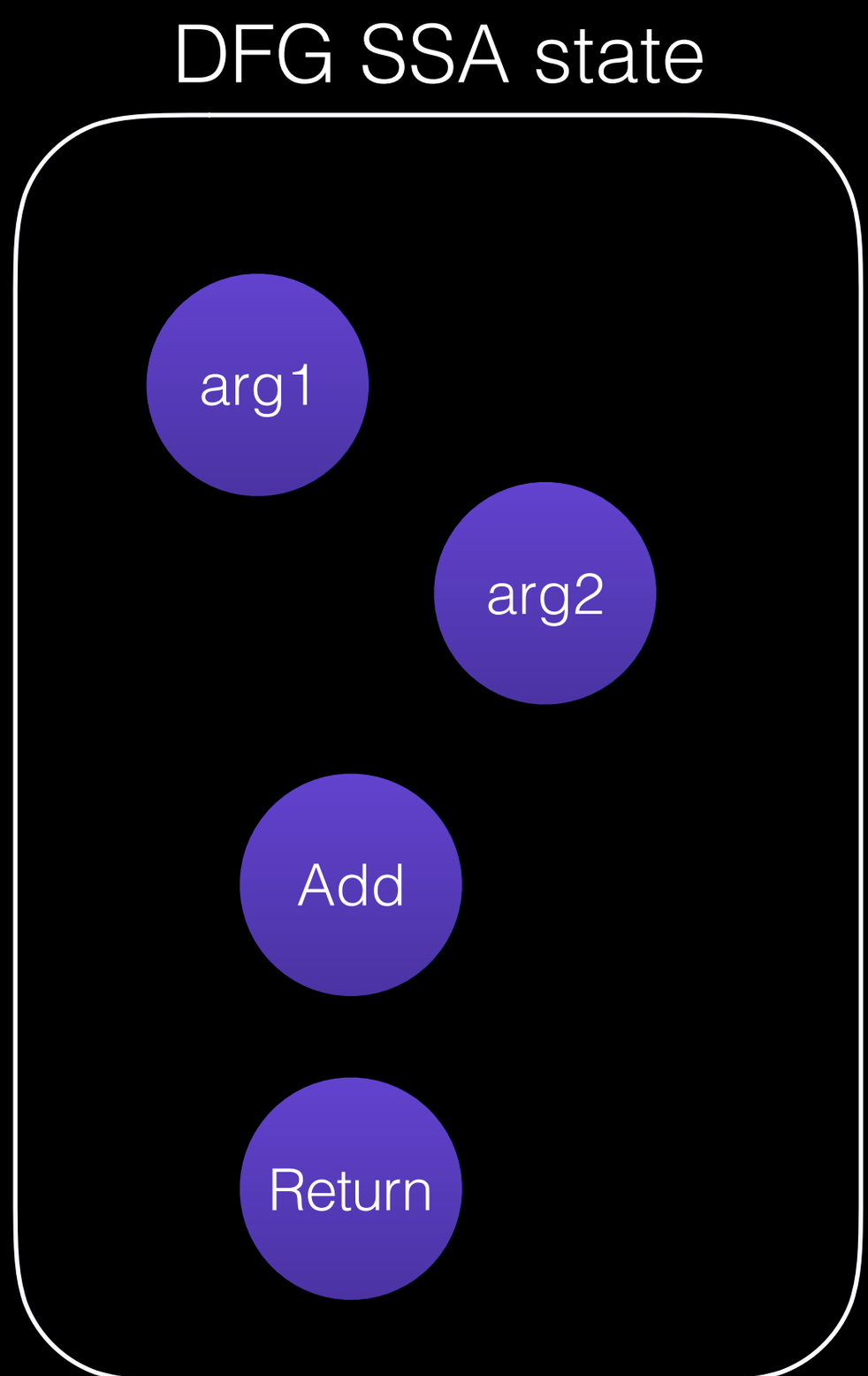
```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```

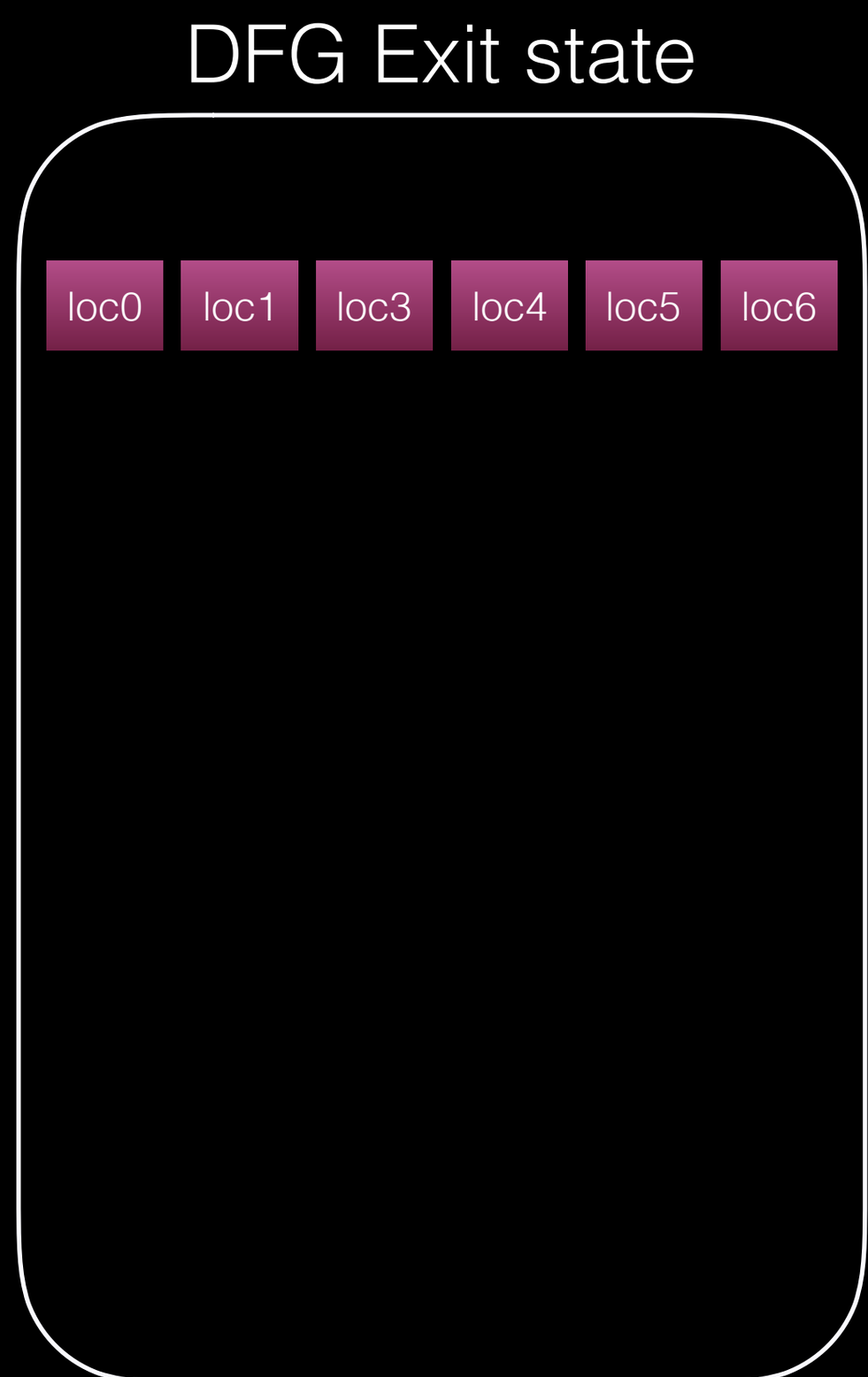
```
[   7] add                    loc6, arg1, arg2
```

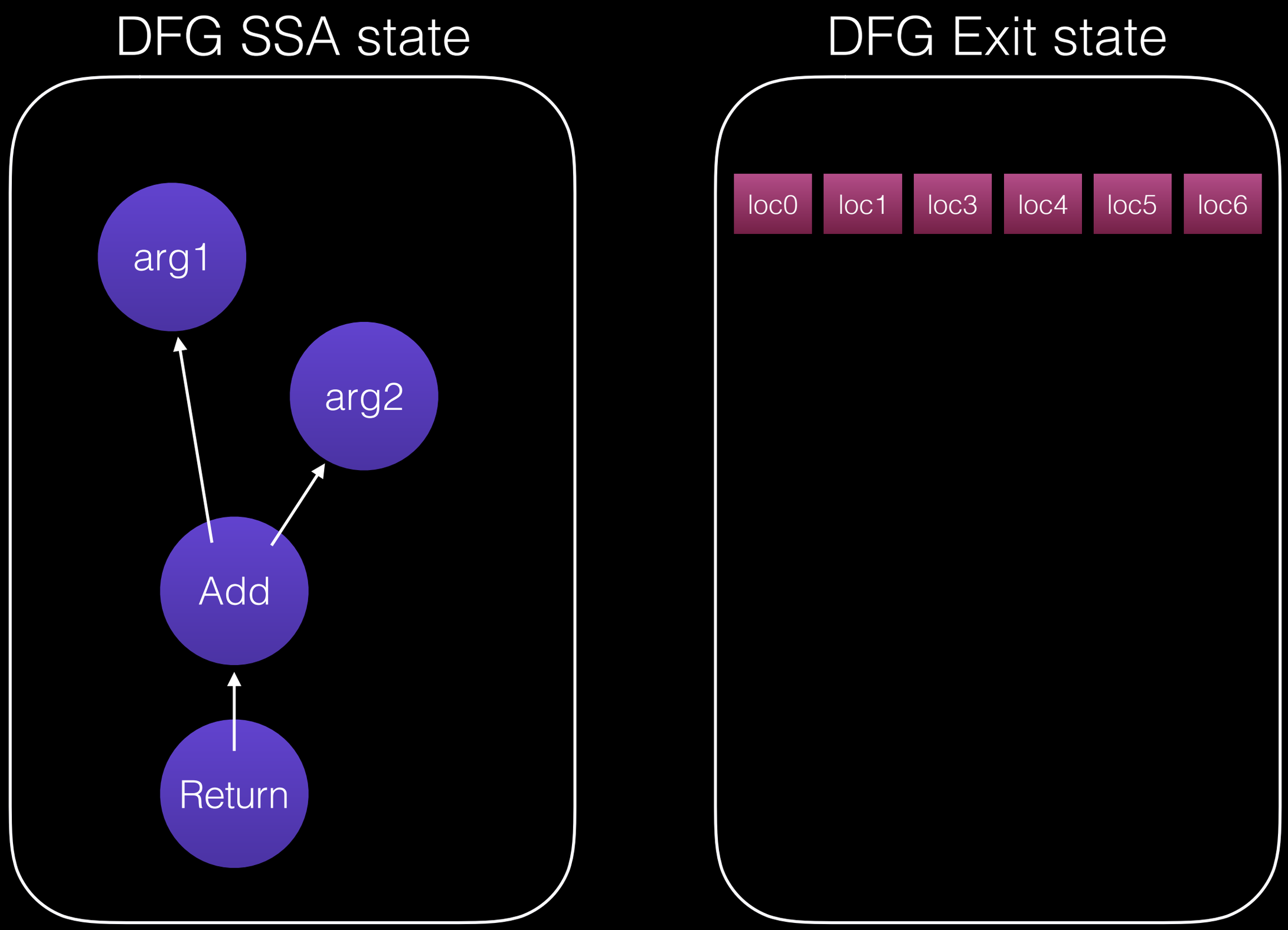
```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```

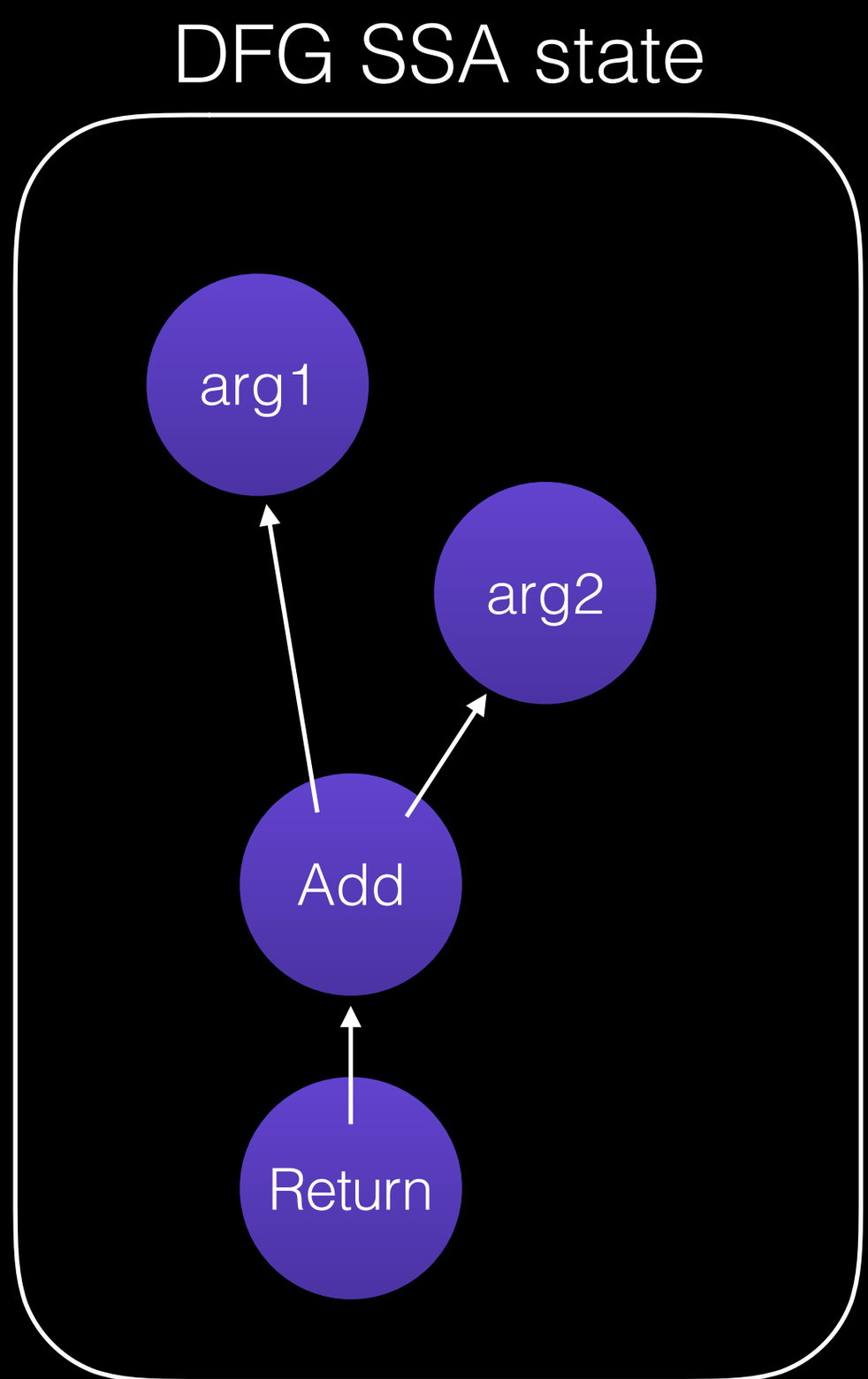
DFG SSA state

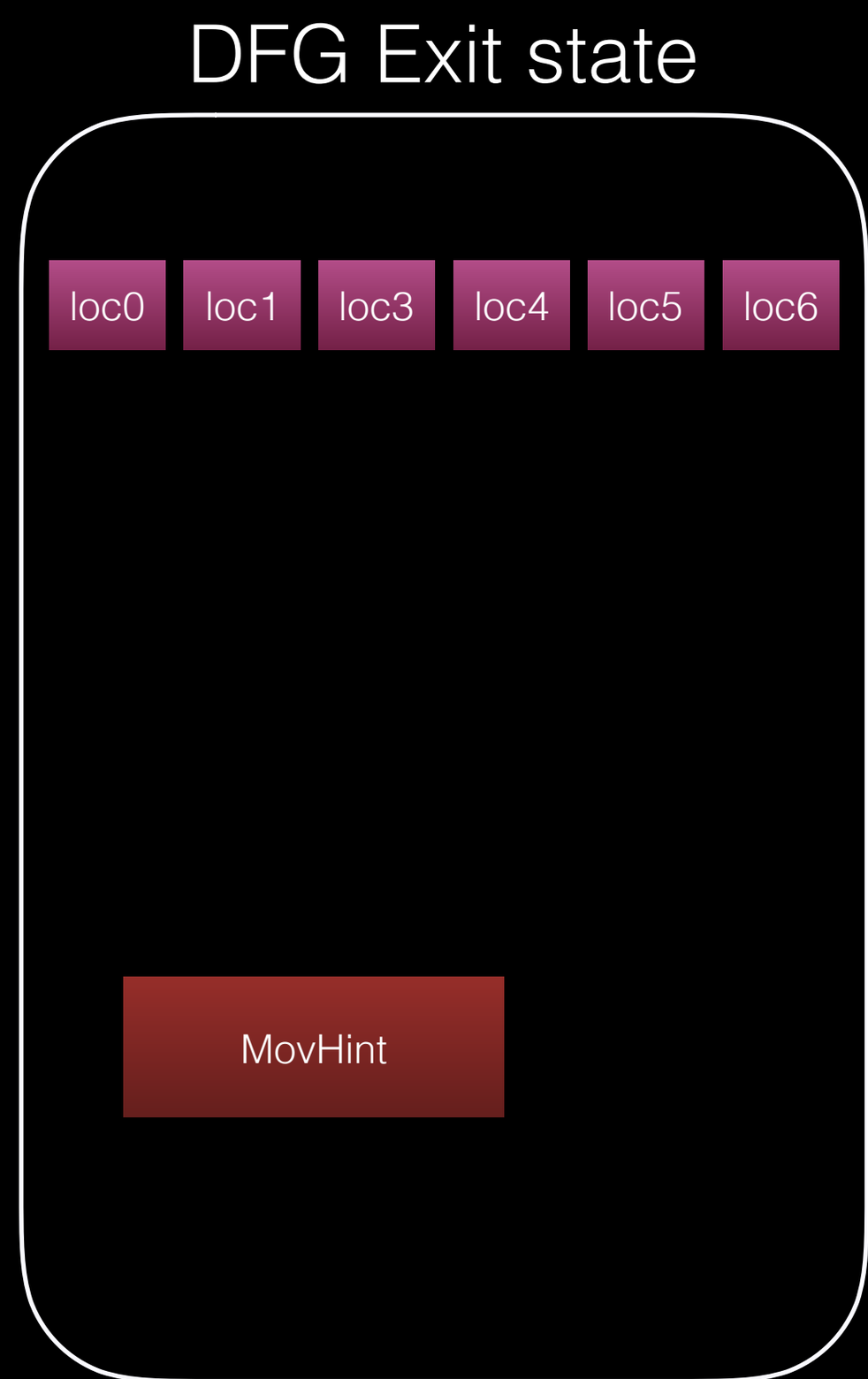


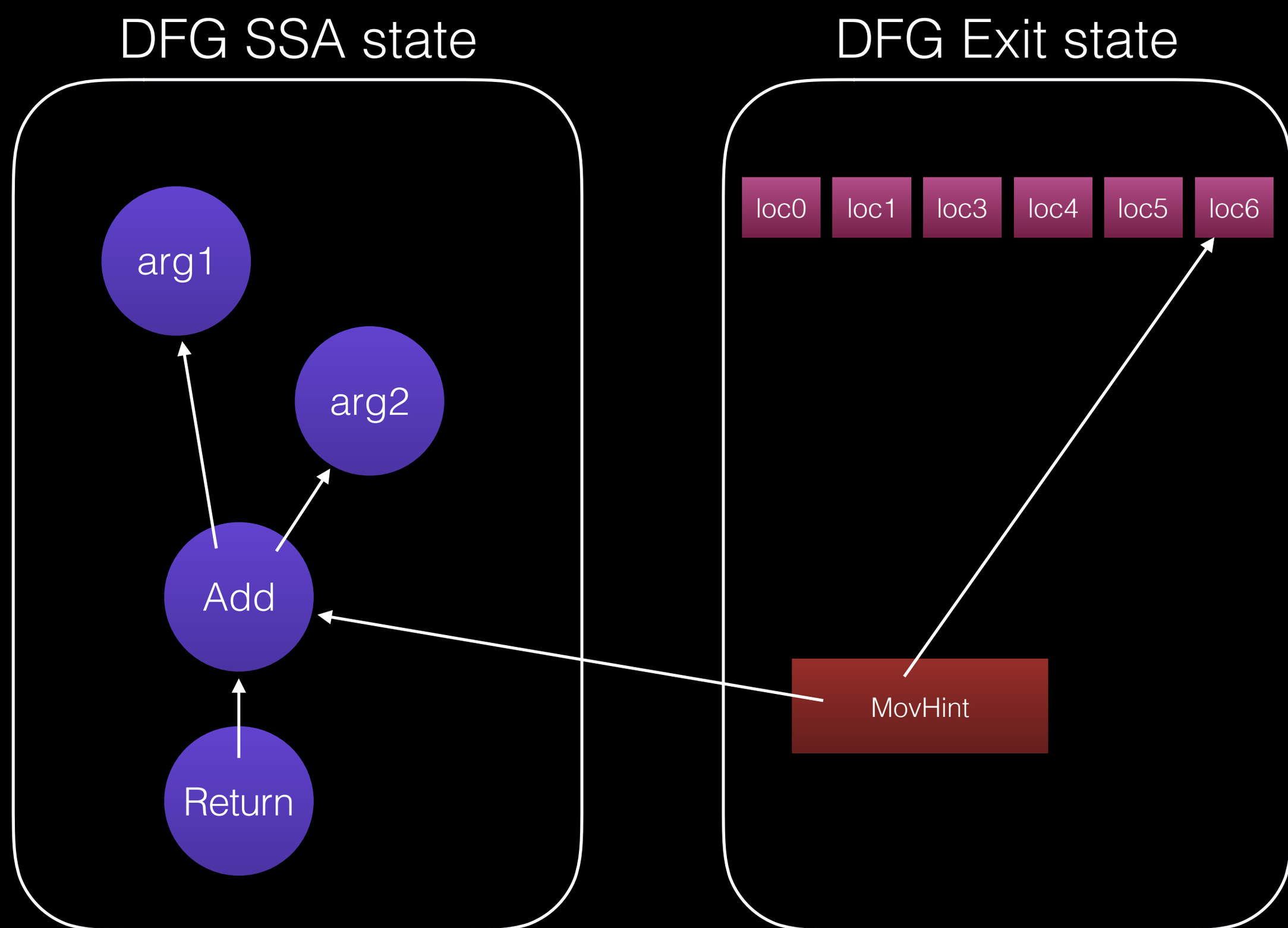

DFG SSA state
arg1
arg2
Add
Return

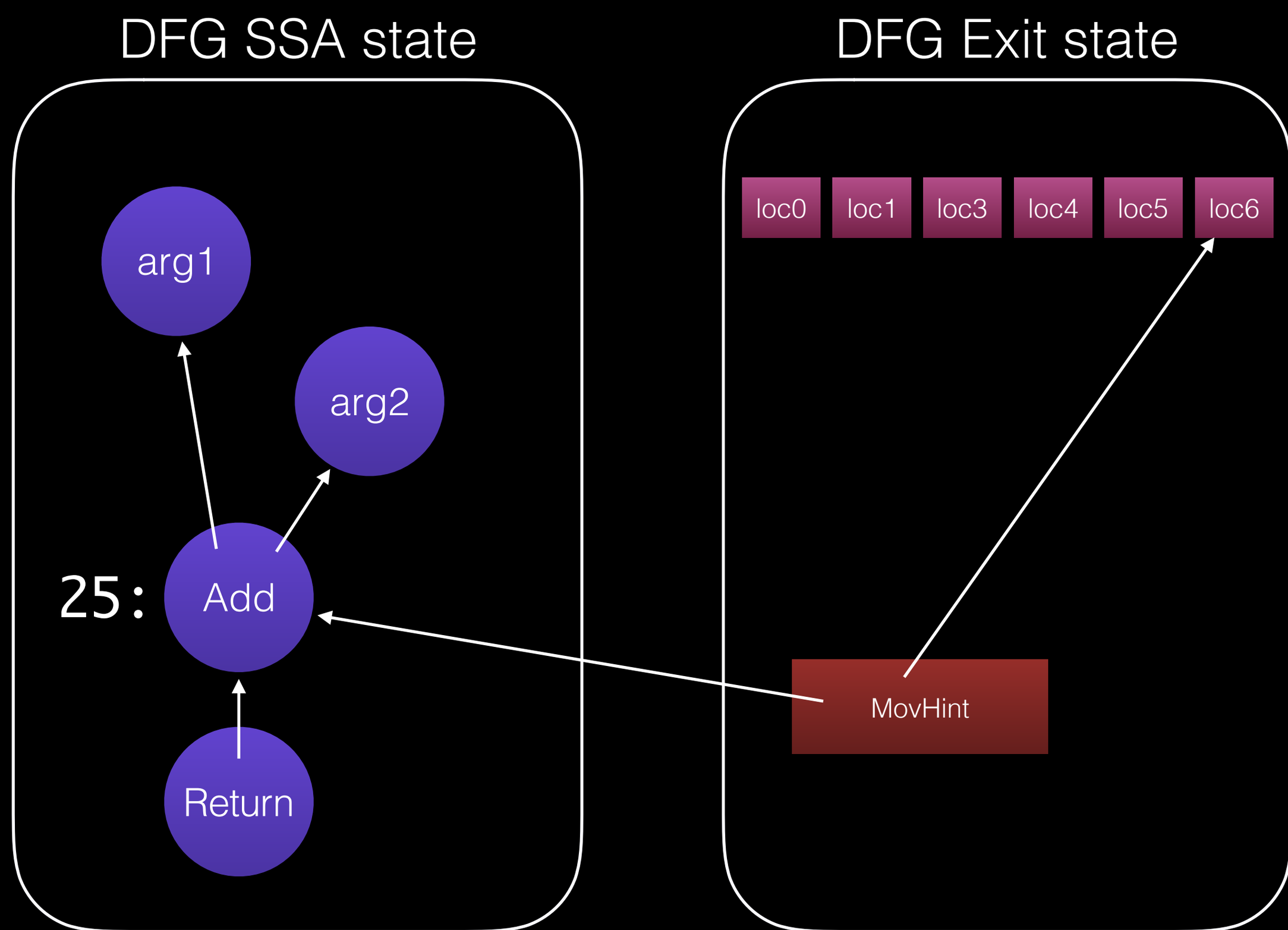

DFG Exit state
loc0
loc1
loc3
loc4
loc5
loc6

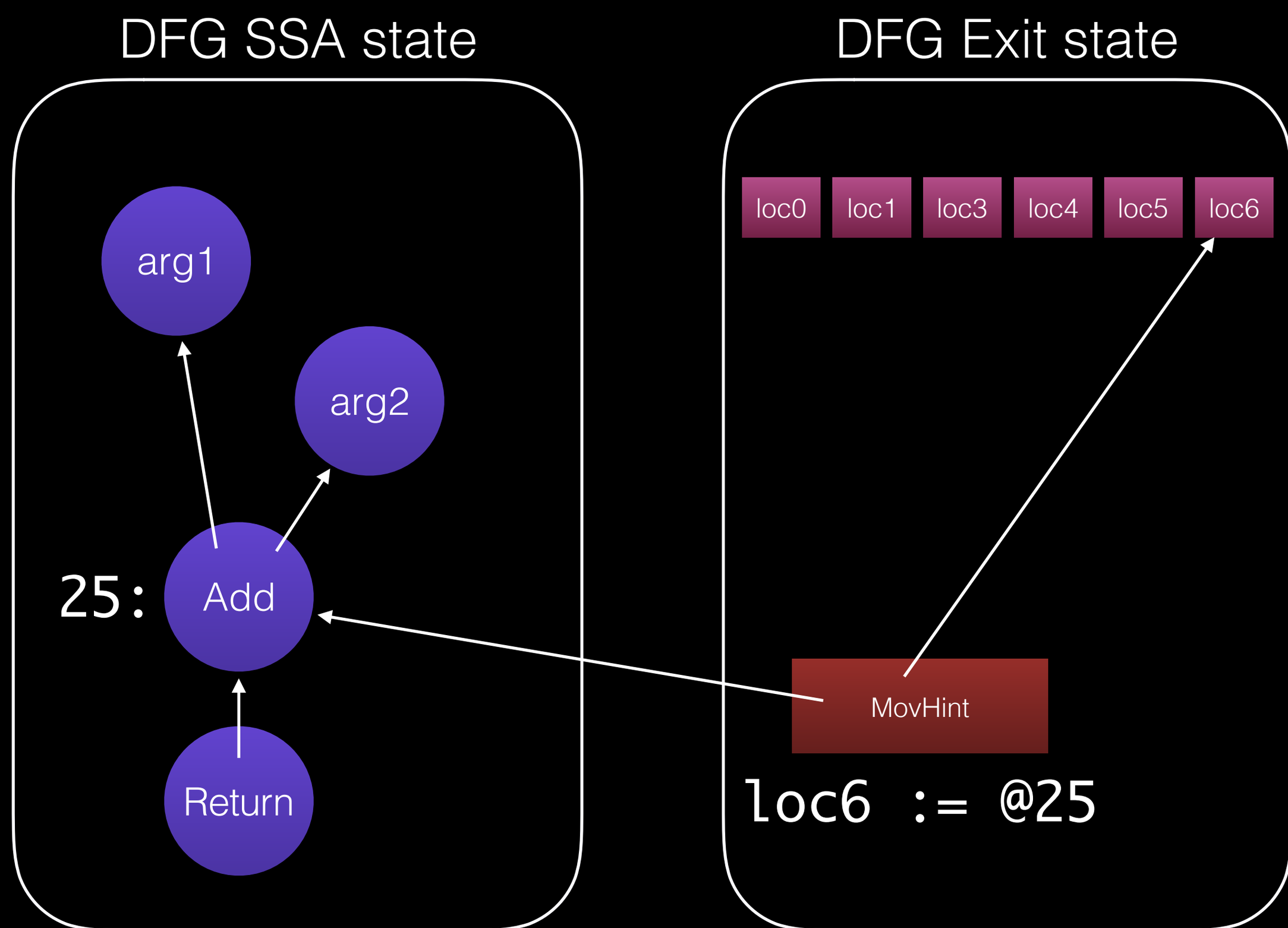

DFG SSA state
arg1
arg2
Add
Return
DFG Exit state
loc0
loc1
loc3
loc4
loc5
loc6

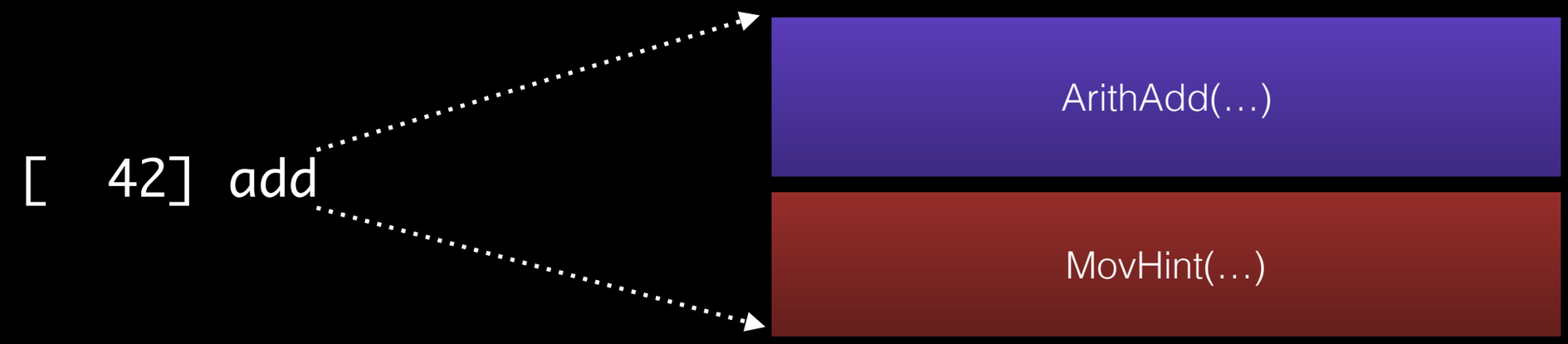

DFG SSA state
arg1
arg2
Add
Return

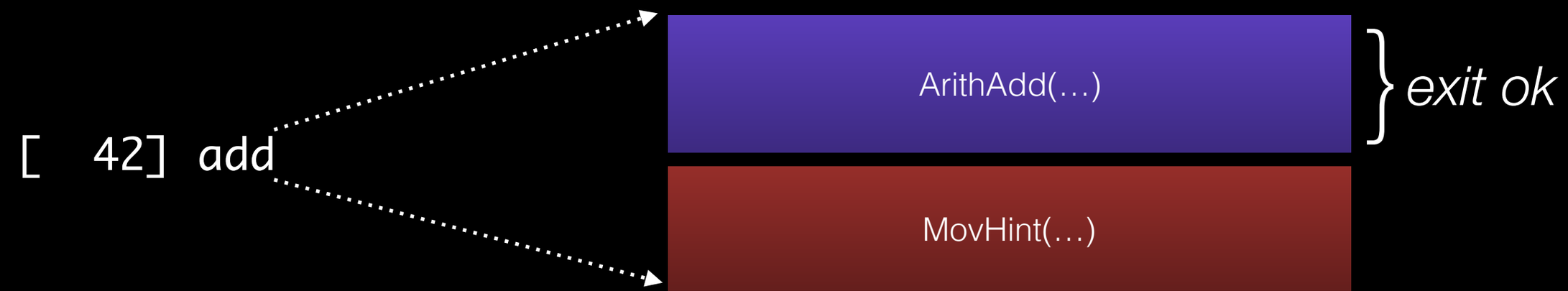

DFG Exit state
loc0
loc1
loc3
loc4
loc5
loc6
MovHint

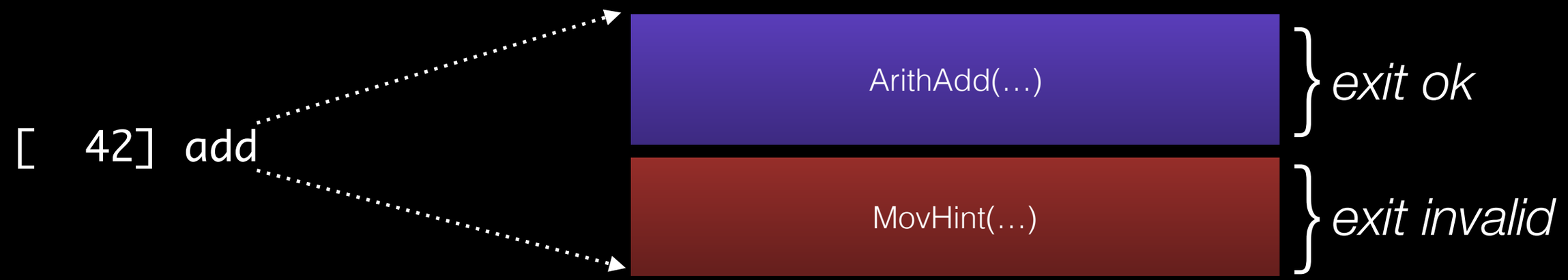

DFG SSA state
arg1
arg2
Add
Return
DFG Exit state
loc0
loc1
loc3
loc4
loc5
loc6
MovHint

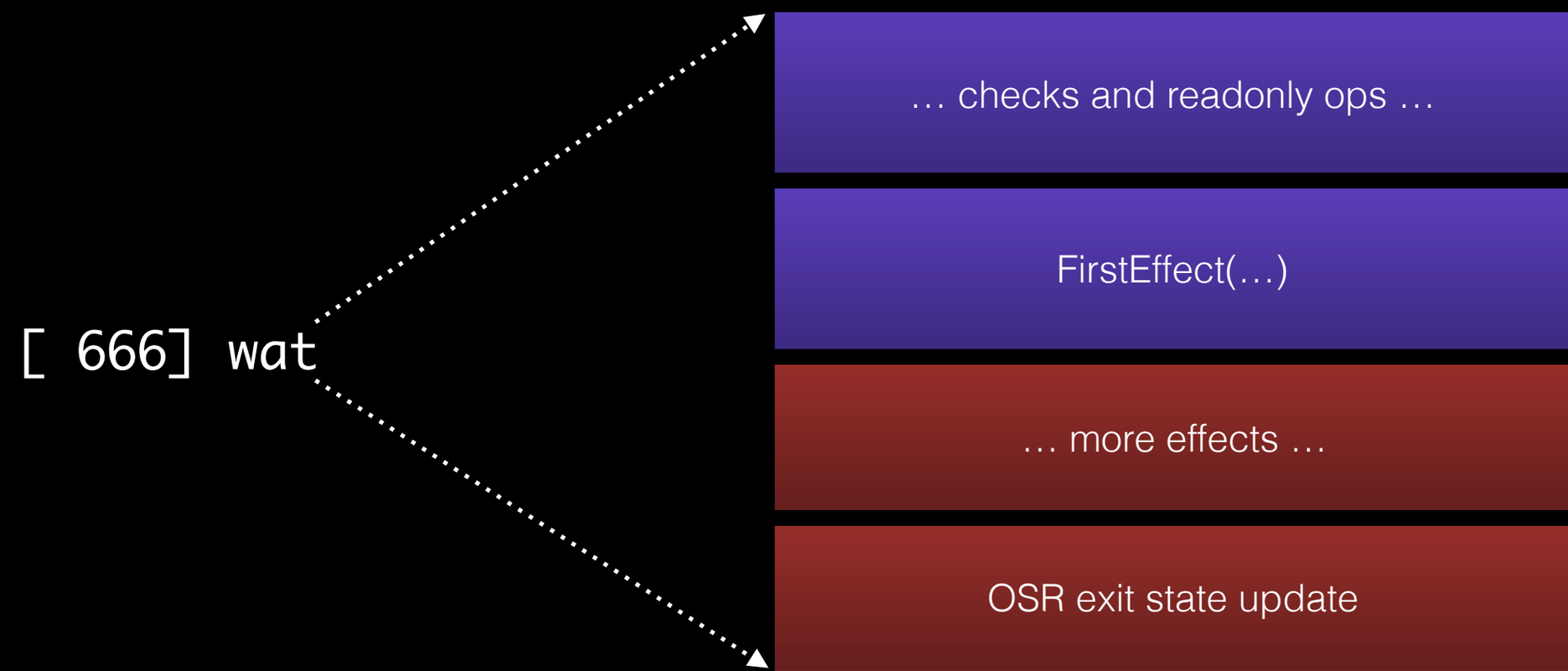

DFG SSA state
arg1
arg2
25:
Add
Return
DFG Exit state
loc0
loc1
loc3
loc4
loc5
loc6
MovHint

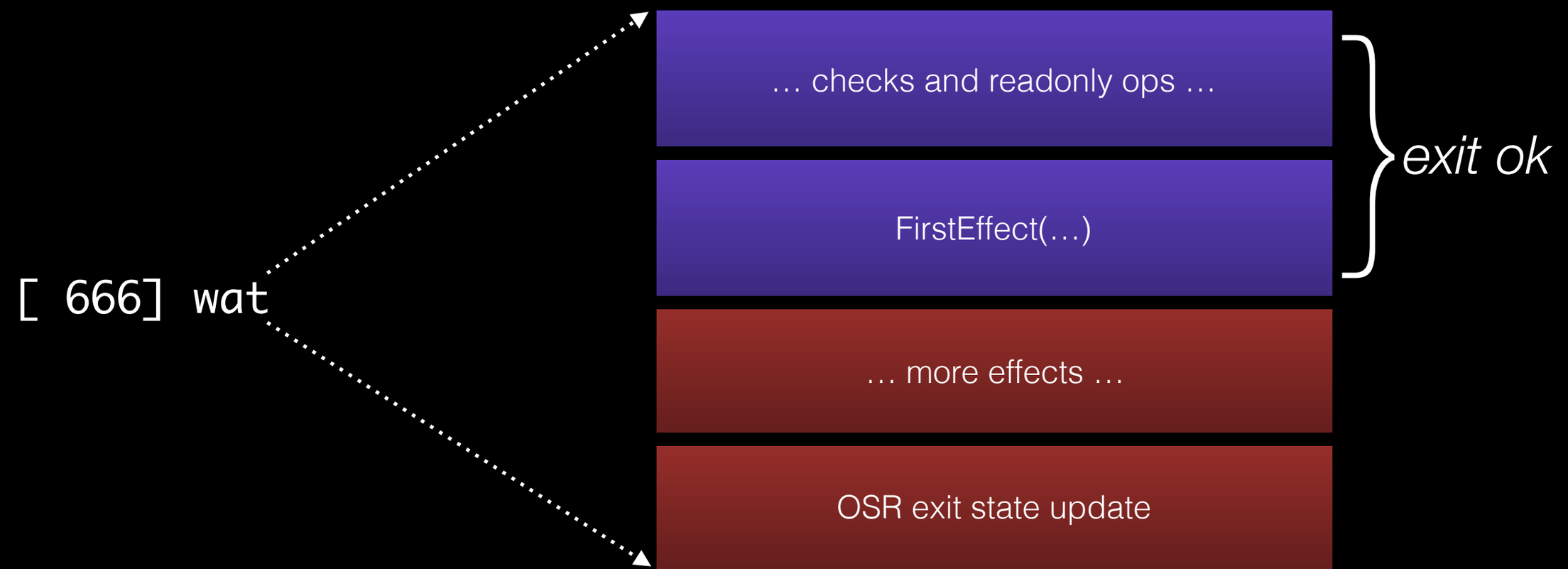

DFG SSA state
arg1
arg2
25:
Add
Return
DFG Exit state
loc0
loc1
loc3
loc4
loc5
loc6
MovHint
loc6 := @25

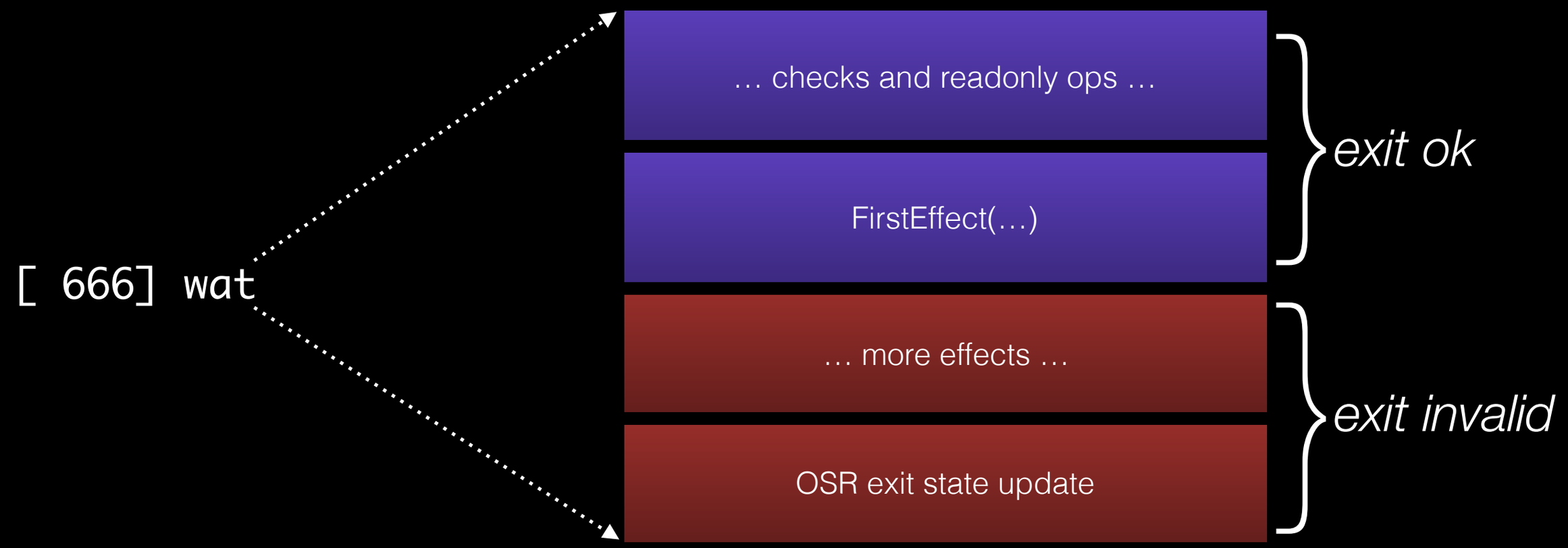

[42] add
ArithAdd(…)
MovHint(…)

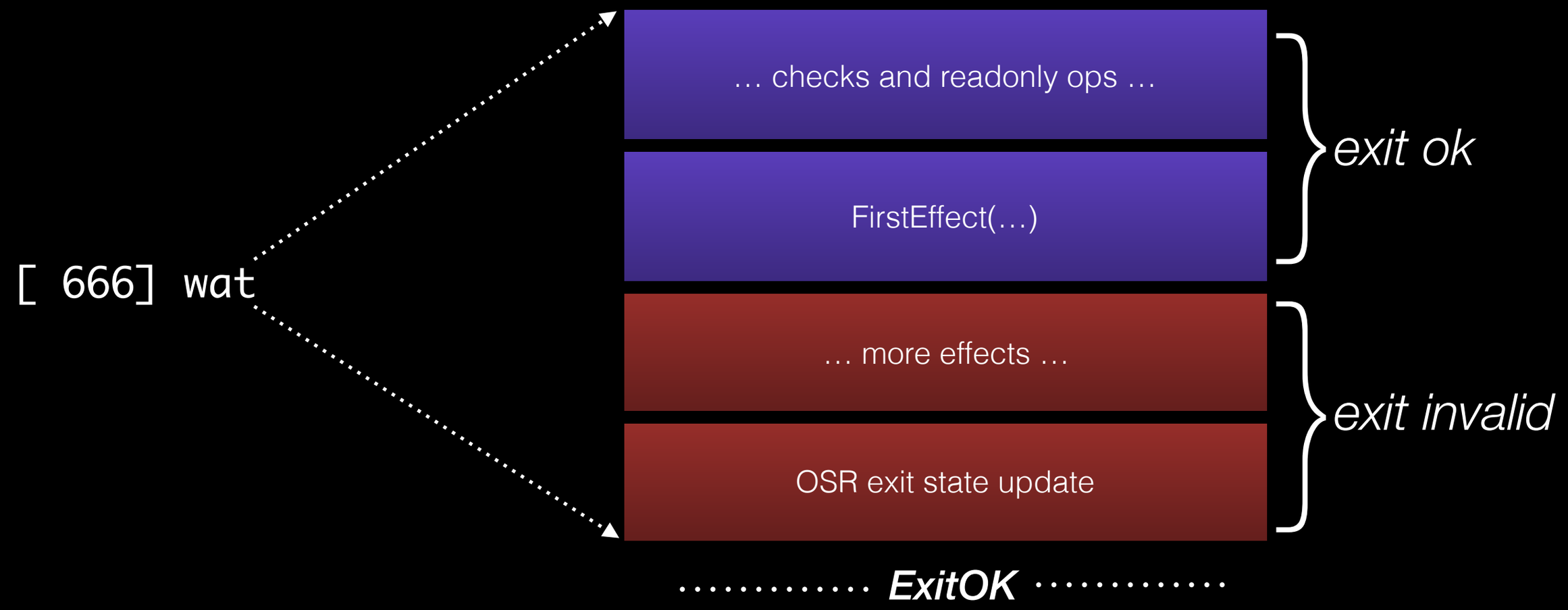

[42] add
ArithAdd(…)
exit ok
MovHint(…)

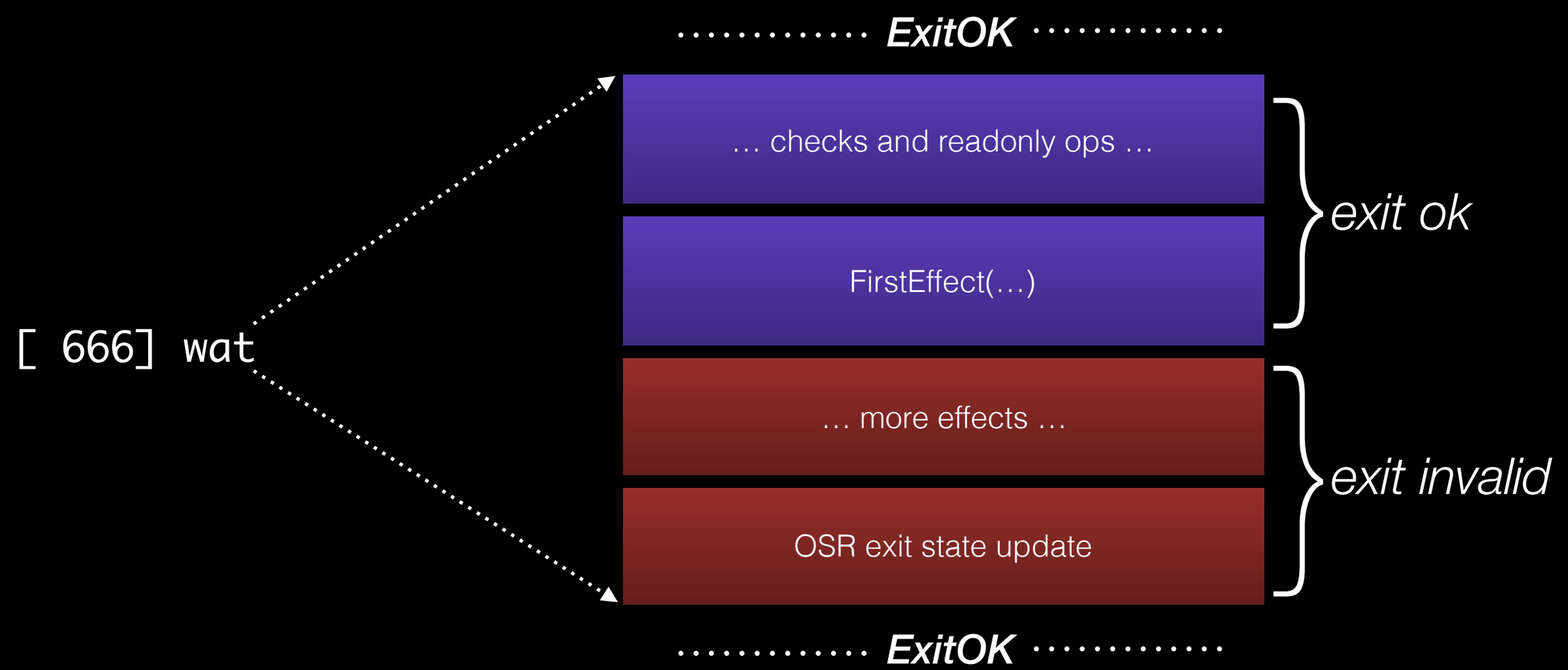

[42] add
ArithAdd(…)
exit ok
MovHint(…)
exit invalid

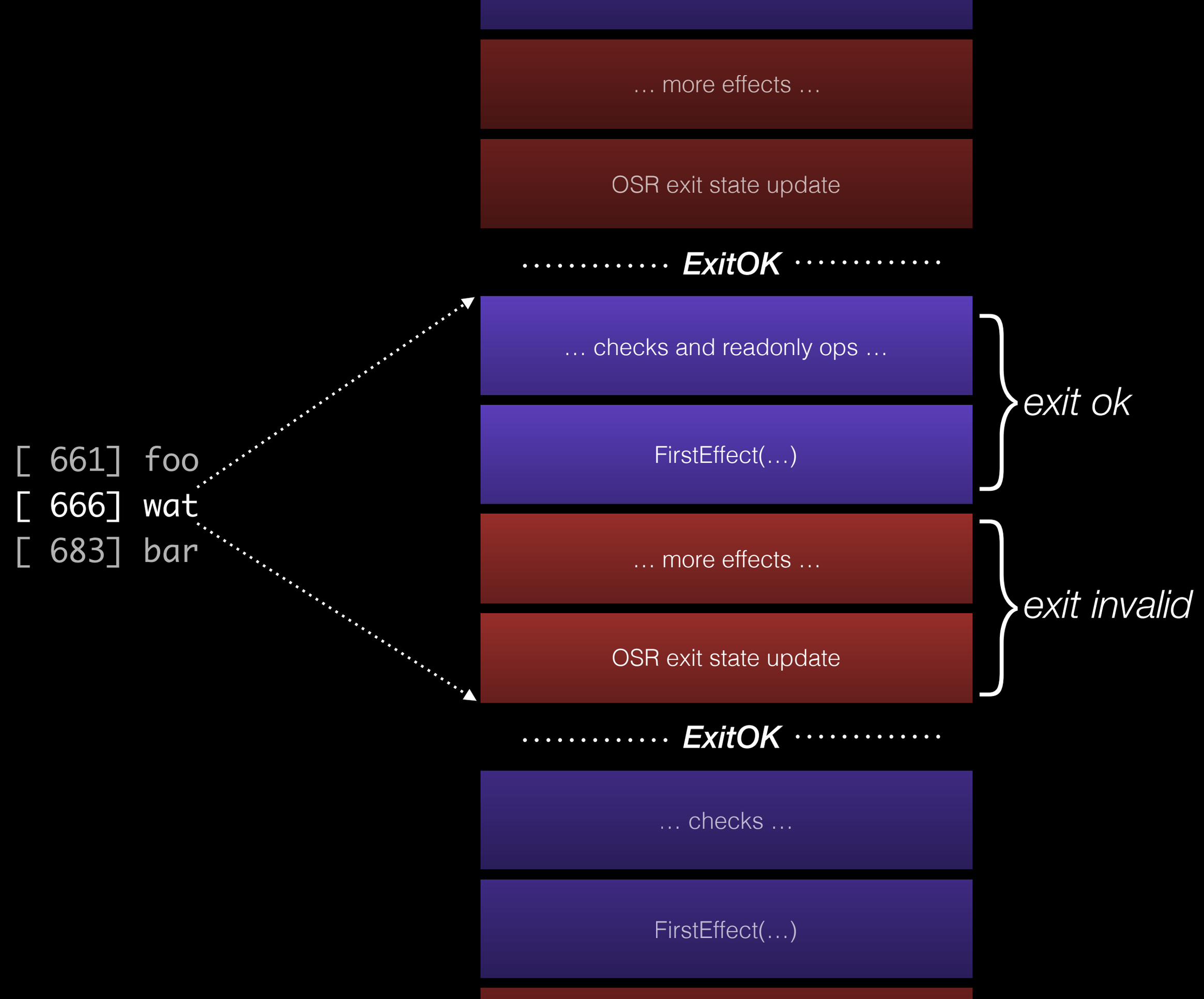

[666] wat
… checks and readonly ops …
FirstEffect(…)
… more effects …
OSR exit state update


[666] wat
… checks and readonly ops …
FirstEffect(…)
… more effects …
OSR exit state update
exit ok


[666] wat
… checks and readonly ops …
FirstEffect(…)
… more effects …
OSR exit state update
exit ok
exit invalid


[666] wat
… checks and readonly ops …
FirstEffect(…)
… more effects …
OSR exit state update
exit ok
exit invalid
ExitOK


ExitOK
… checks and readonly ops …
FirstEffect(…)
exit ok
[666] wat
… more effects …
OSR exit state update
exit invalid
ExitOK


… more effects …
OSR exit state update
ExitOK
… checks and readonly ops …
FirstEffect(…)
exit ok
… more effects …
OSR exit state update
exit invalid
ExitOK
… checks …
FirstEffect(…)
[661] foo
[666] wat
[683] bar

Watchpoints
+
InvalidationPoint

```
function foo() {
    return Math.pow(2, 3);
}
```

```
function foo() {
    return Math.pow(2, 3);
}
```

Profiling Tier Version

slow lookup of Math.pow

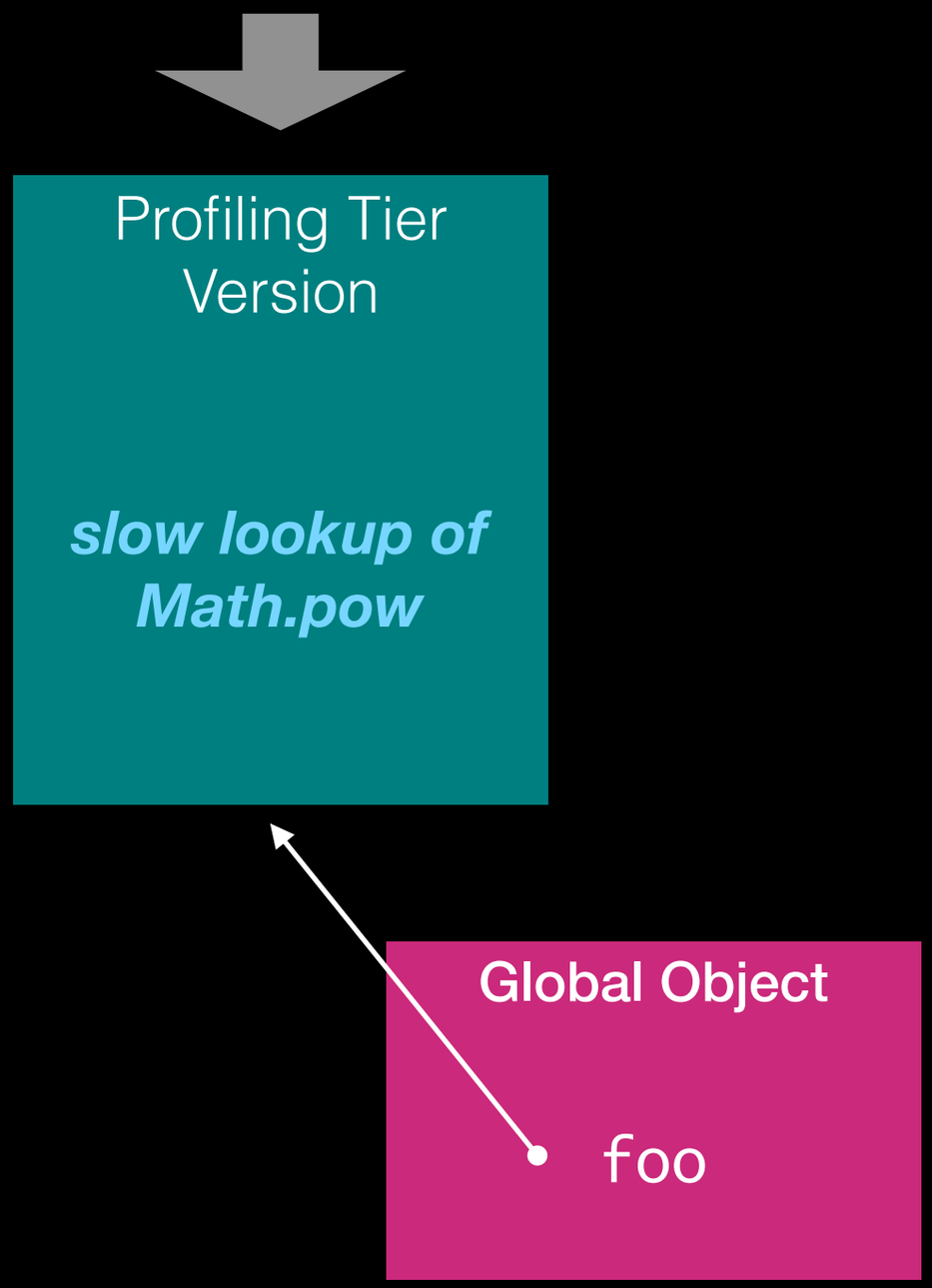
```
function foo() {
    return Math.pow(2, 3);
}
```

Profiling Tier Version

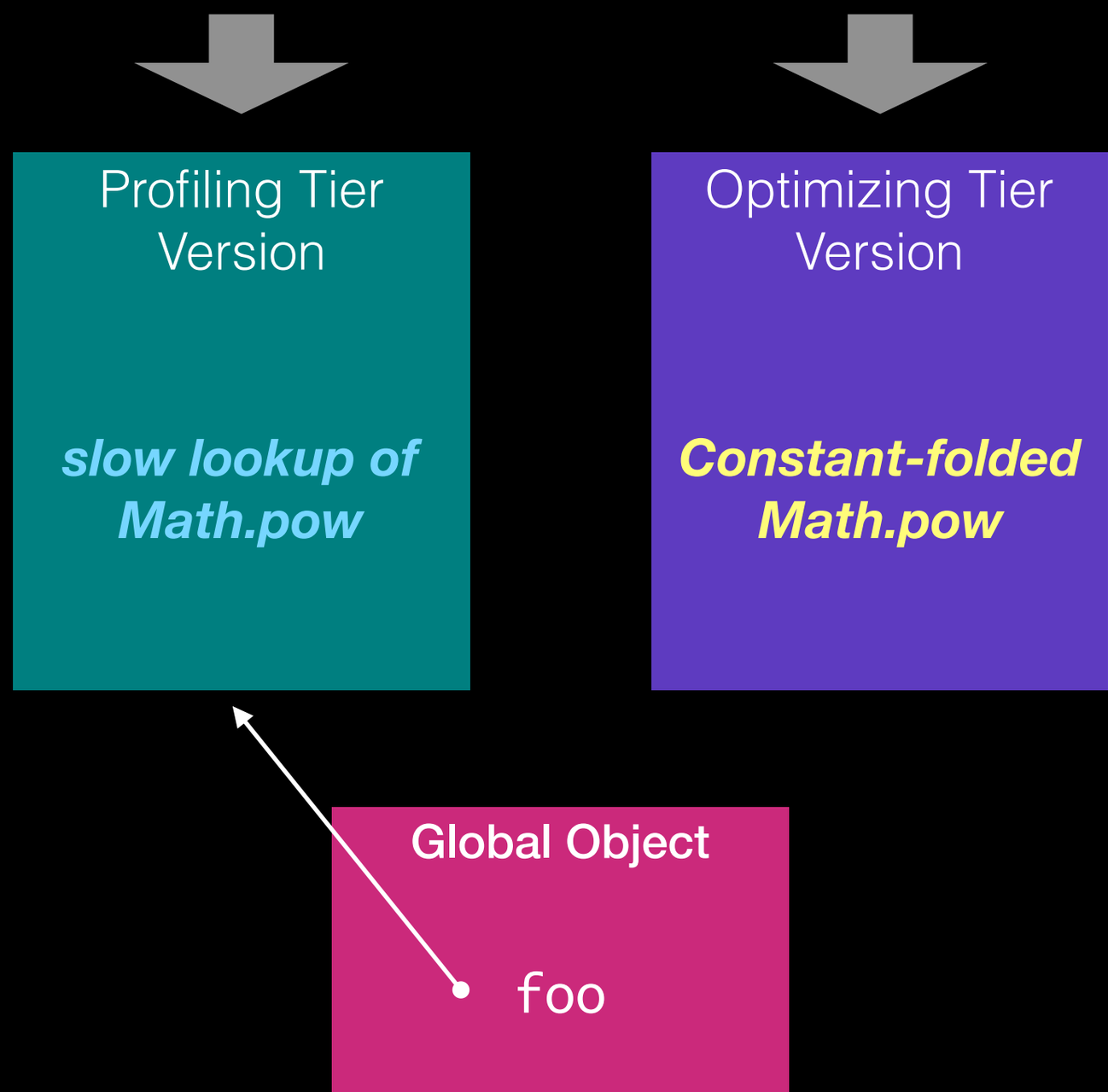
slow lookup of Math.pow

Global Object

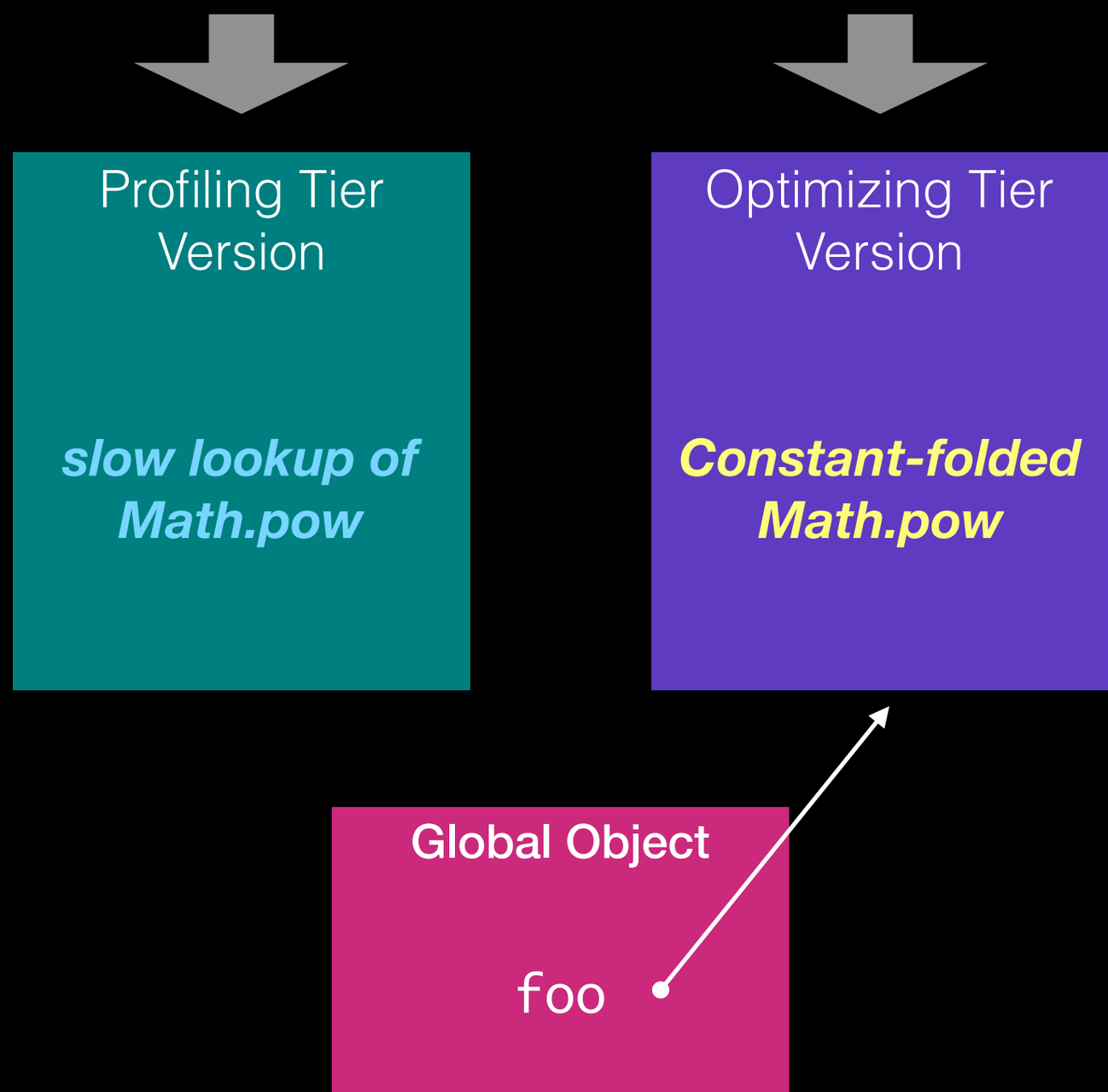
```
function foo() {
    return Math.pow(2, 3);
}
```



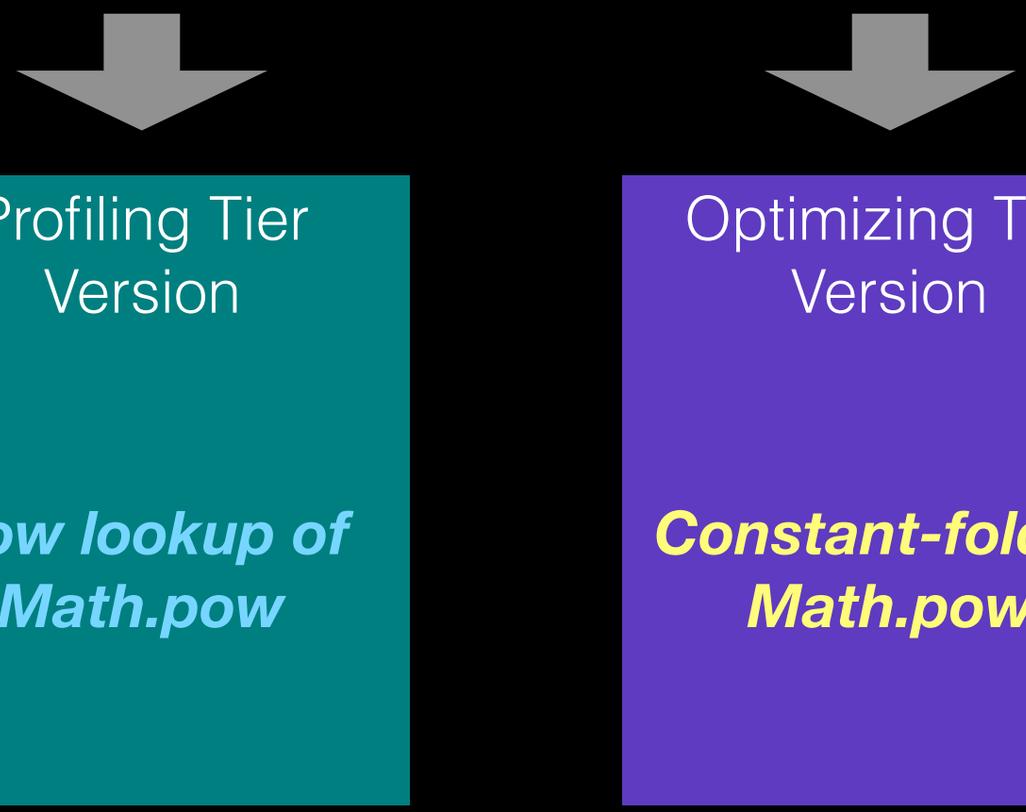
```
function foo() {
    return Math.pow(2, 3);
}
```



```
function foo() {
    return Math.pow(2, 3);
}
```



```
function foo() {
    return Math.pow(2, 3);
}
```




```
Math.pow = “hahaha”;
```

```
function foo() {
    return Math.pow(2, 3);
}
```

Profiling Tier Version

slow lookup of Math.pow


```
Math.pow = “hahaha”;
```

```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

Profiling Tier Version

slow lookup of Math.pow

```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

Profiling Tier Version

slow lookup of Math.pow

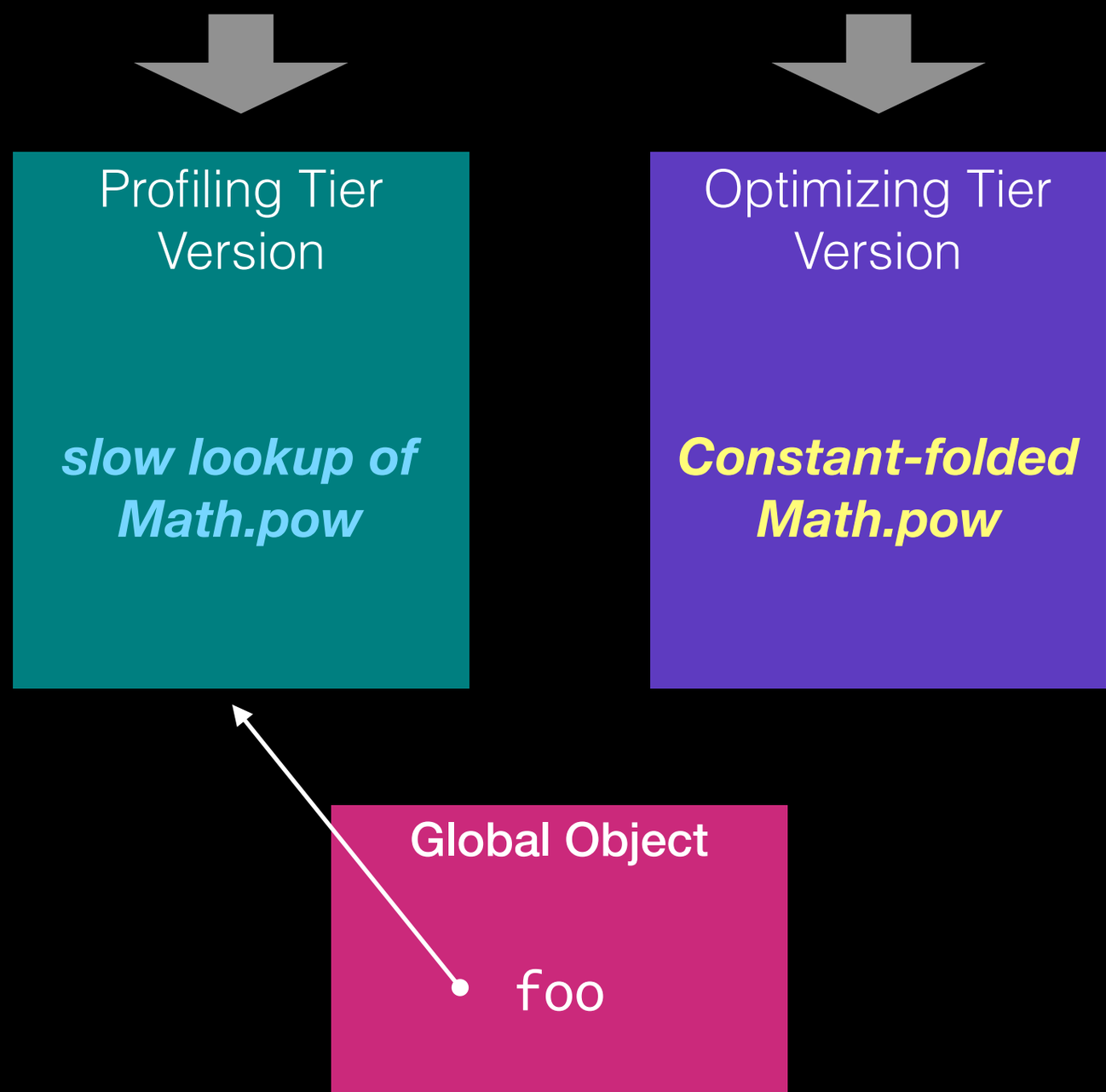
Global Object

```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

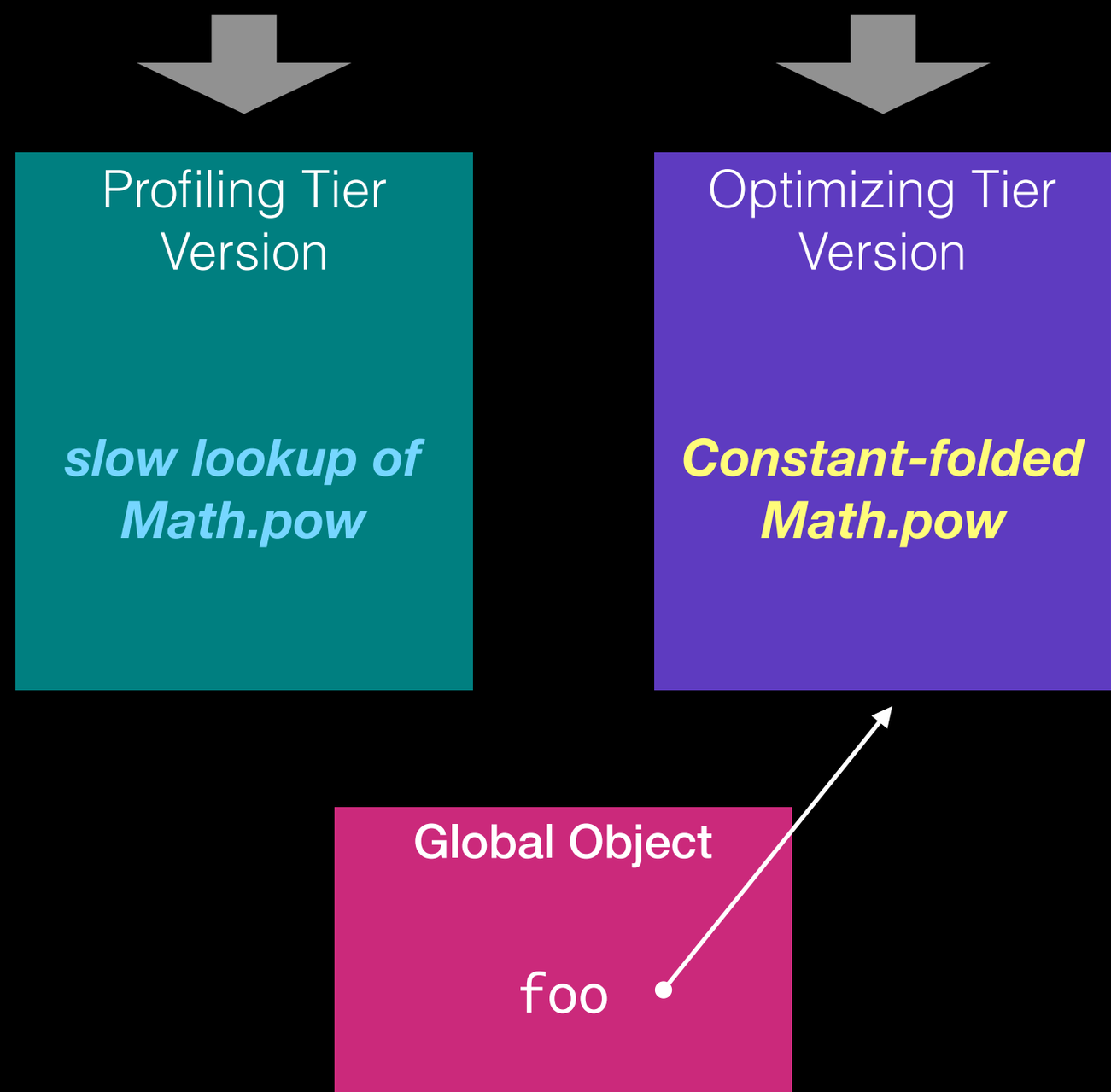
Profiling Tier Version

slow lookup of Math.pow

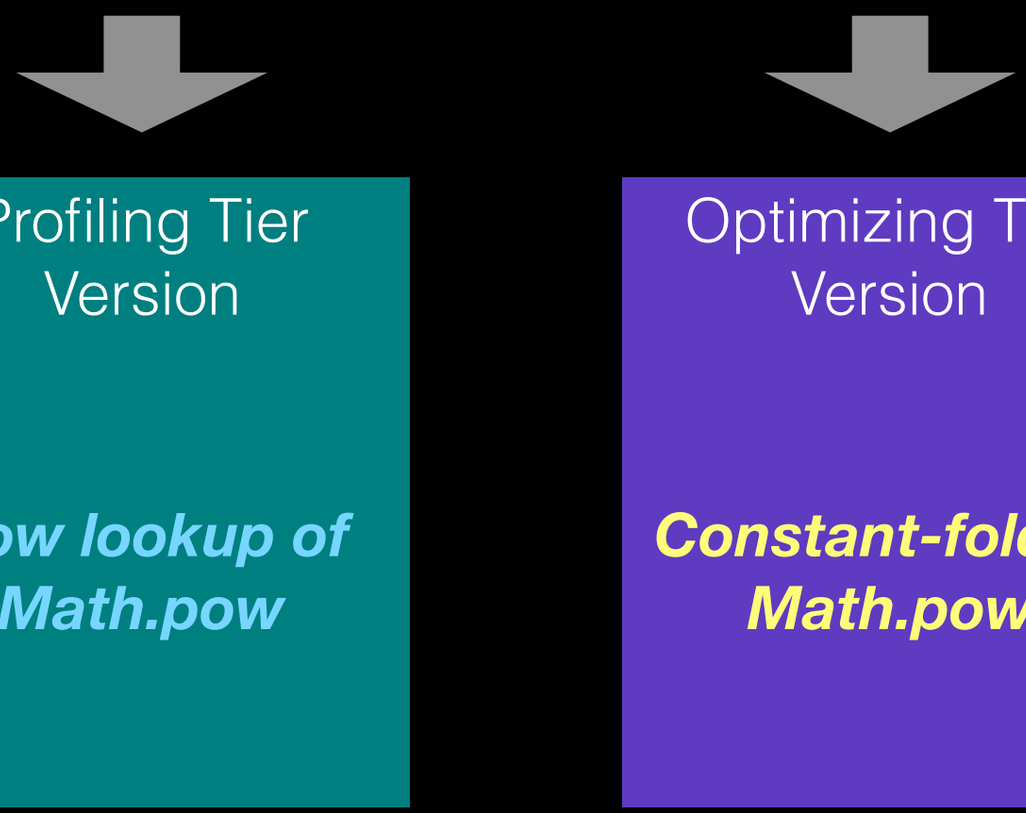
```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```



```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

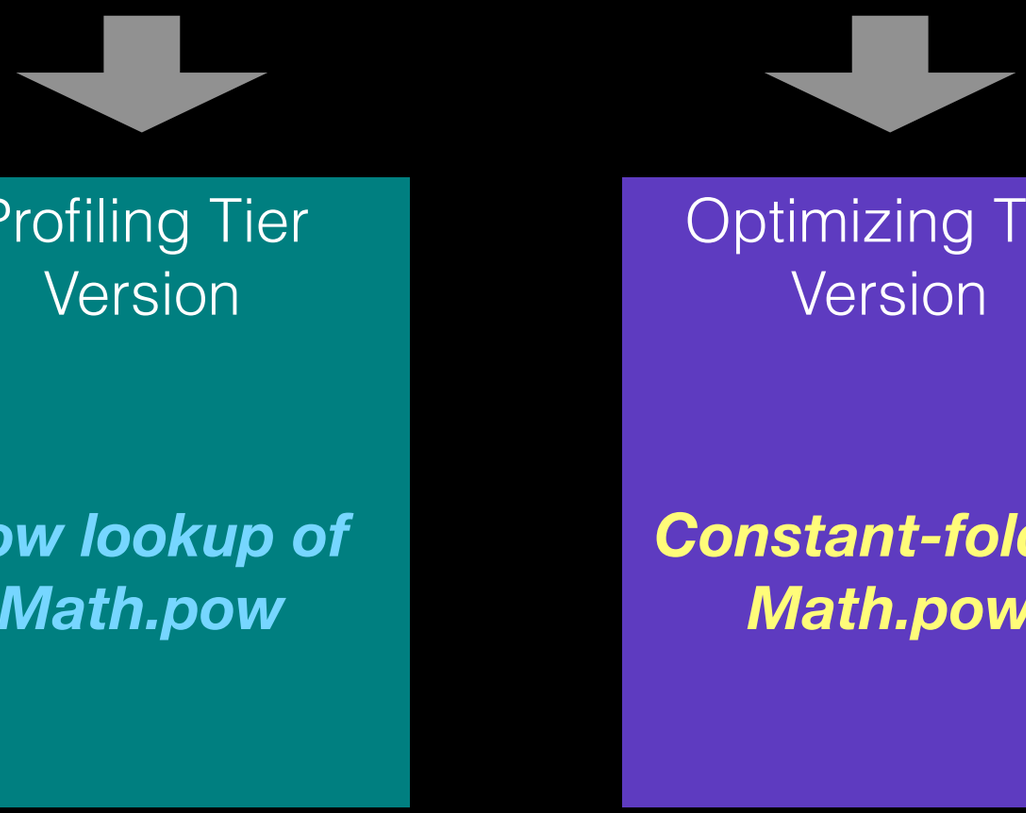


```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```



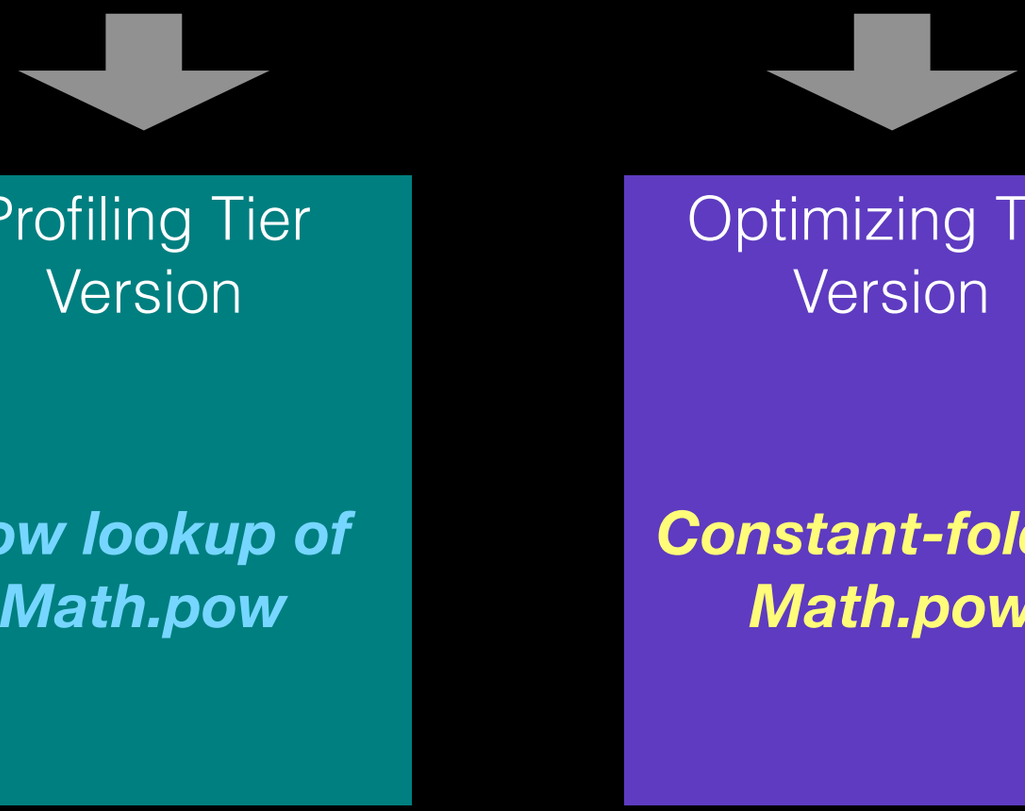

```
function bar() {
    if (p)
        Math = 0;
}
```

```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

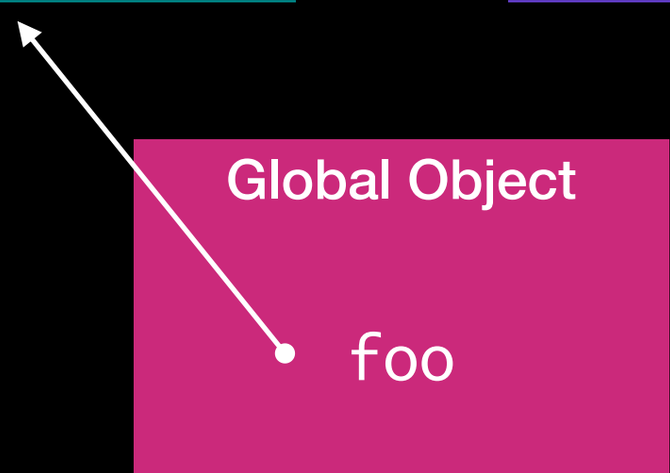



```
function bar() {
    if (p)
        Math = 0;
}
```

```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

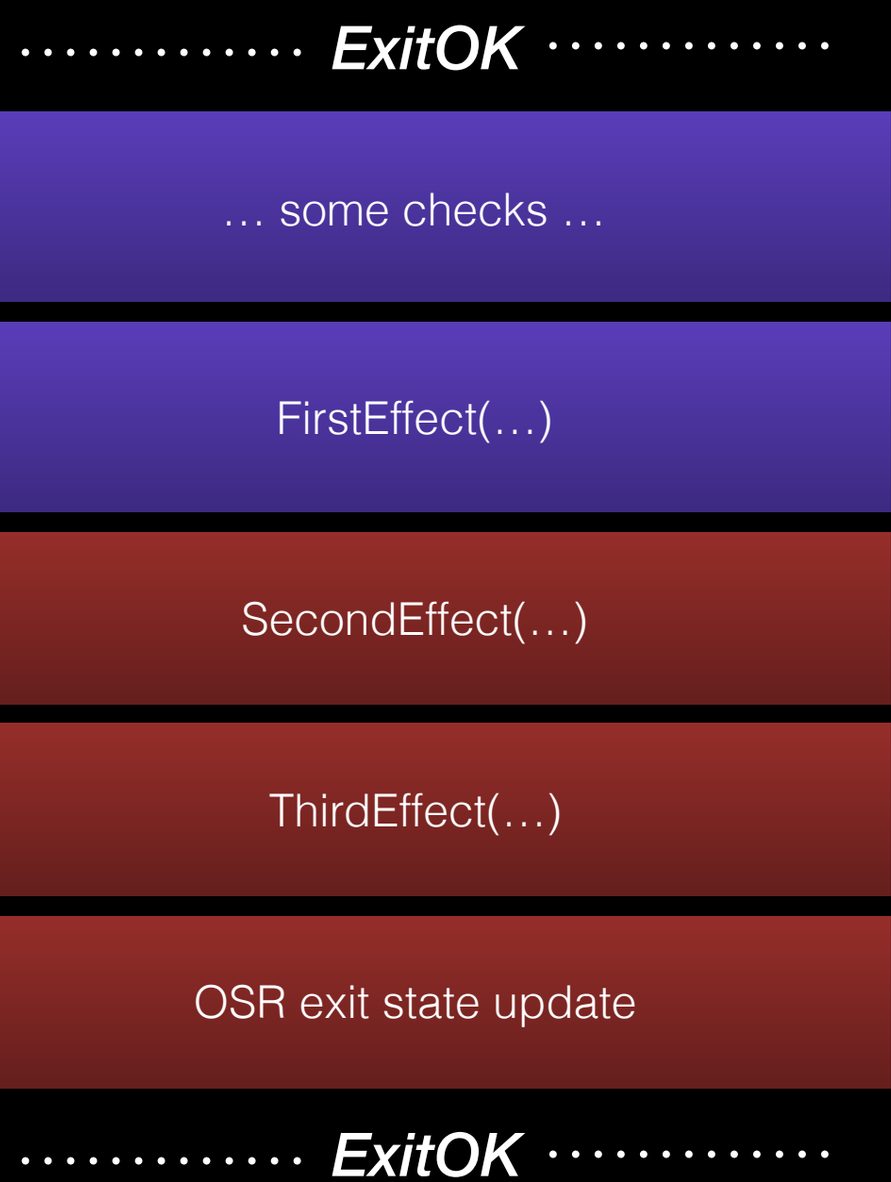


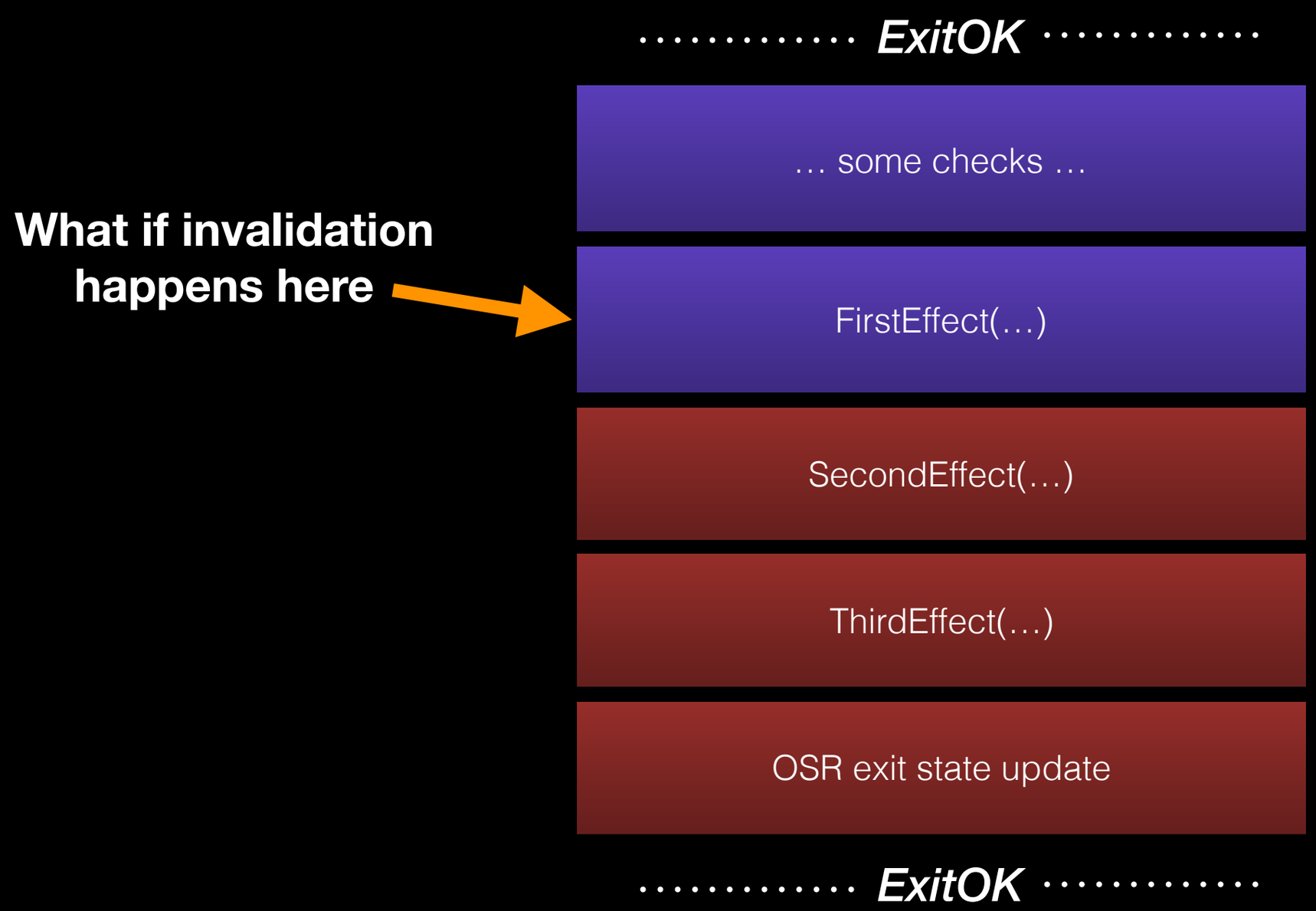

```
function bar() {
    if (p)
        Math = 0;
}
```

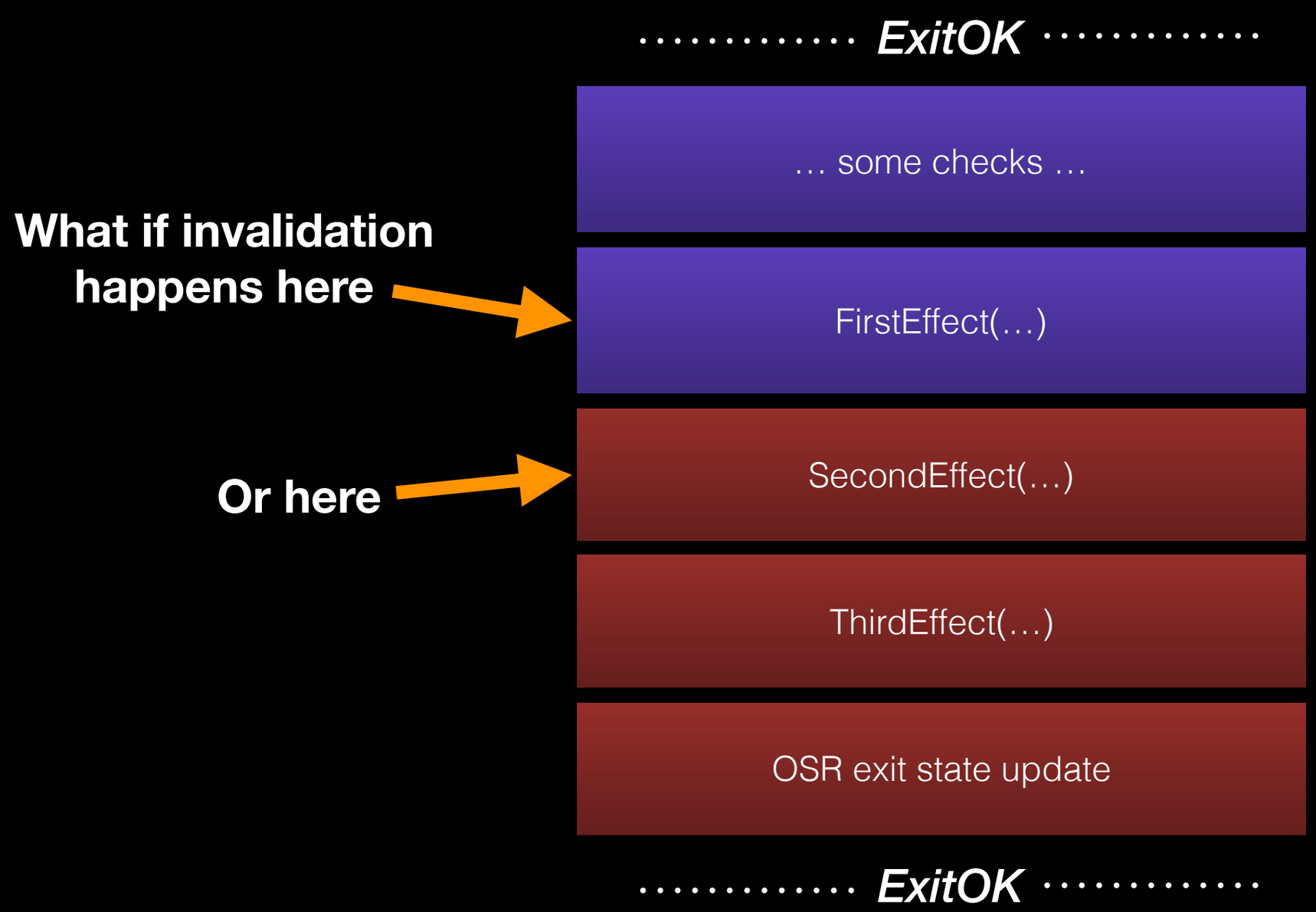


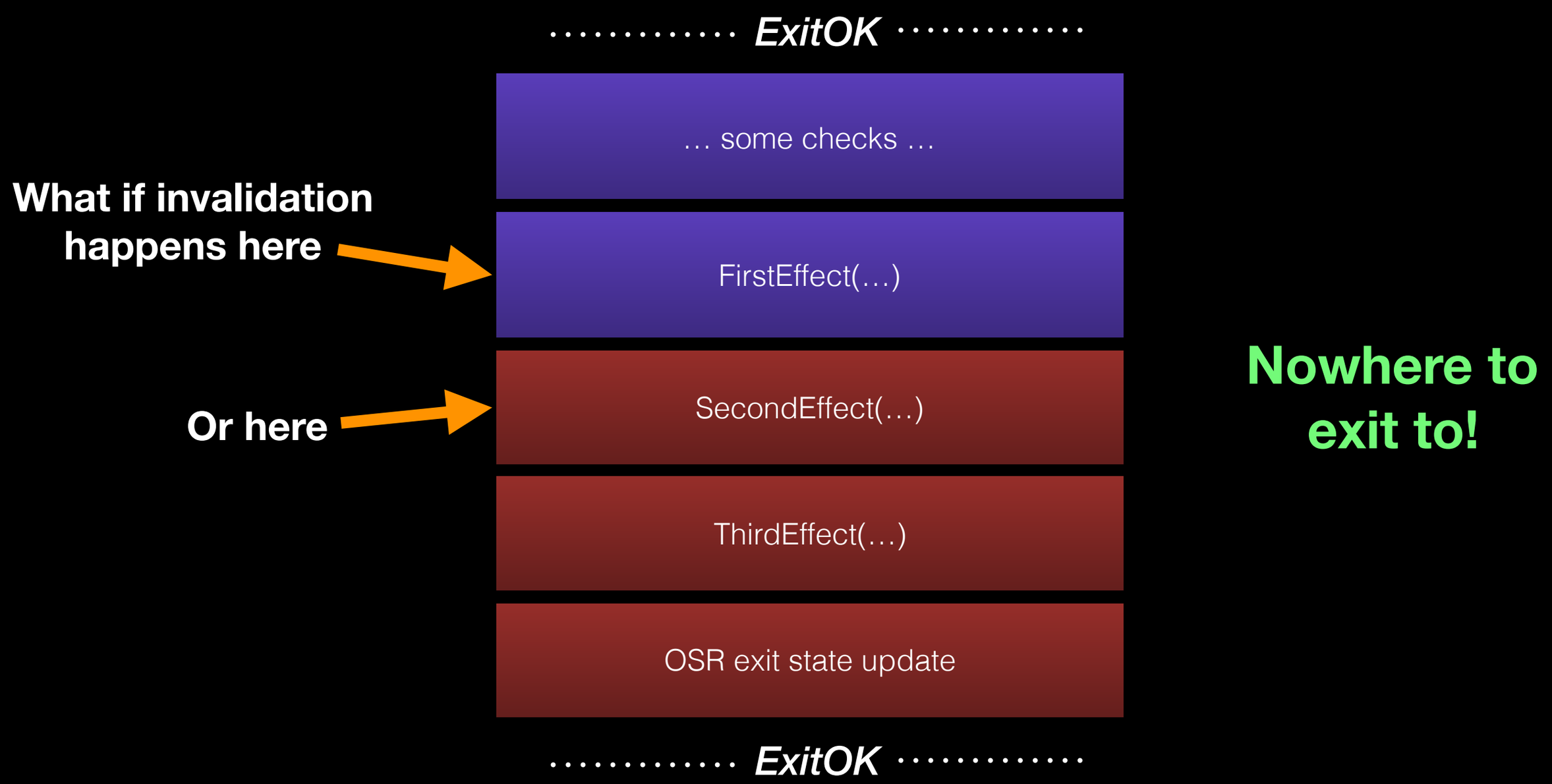
Invalidation Idea

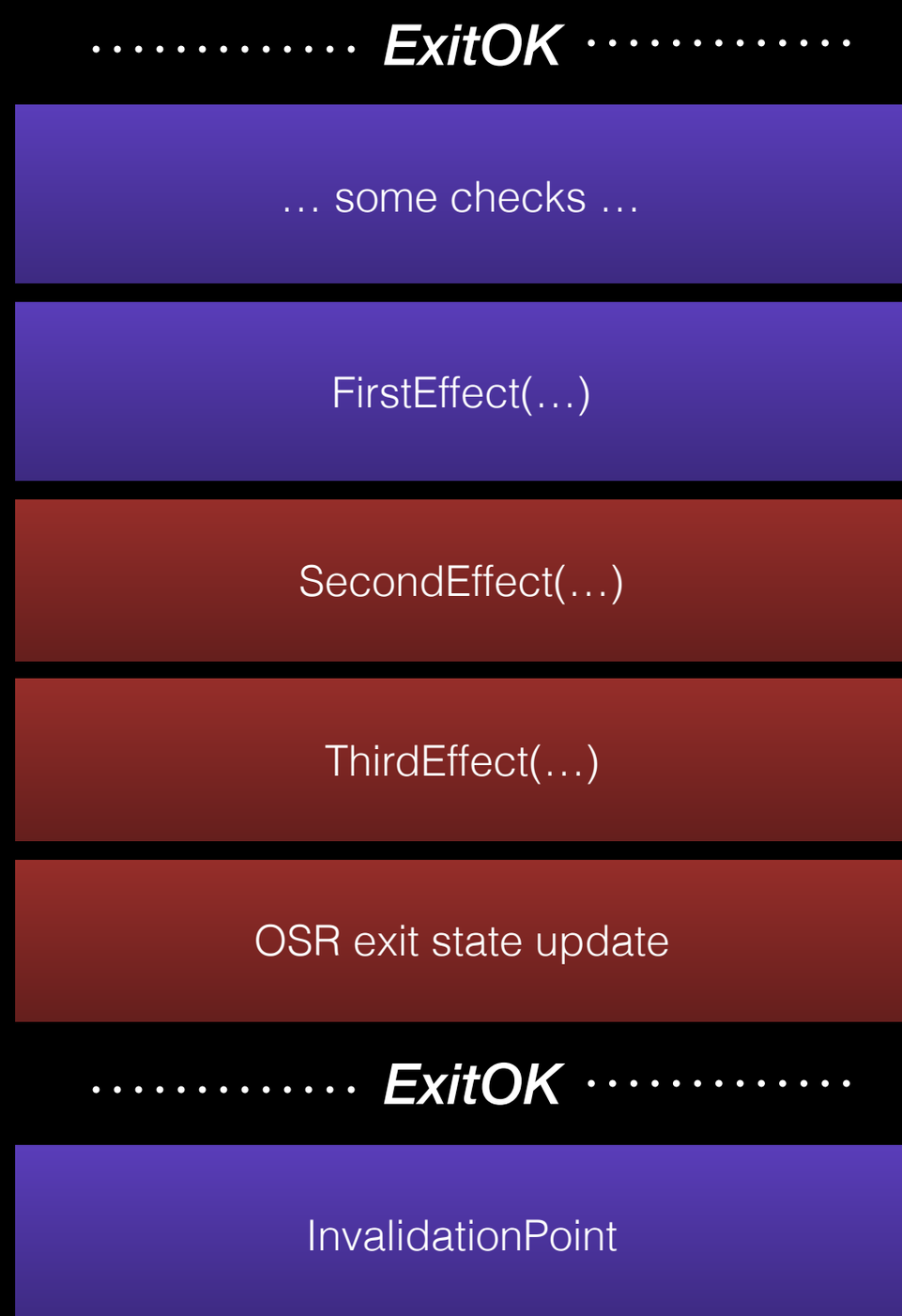
- Walk the stack.
- Repoint return pointers to OSR exit.
- *Widespread idea.*
- *Doesn’t work with DFG IR.*


ExitOK
… some checks …
FirstEffect(…)
SecondEffect(…)
ThirdEffect(…)
OSR exit state update
ExitOK


ExitOK
… some checks …
What if invalidation happens here
FirstEffect(…)
SecondEffect(…)
ThirdEffect(…)
OSR exit state update
ExitOK


ExitOK
… some checks …
What if invalidation happens here
FirstEffect(…)
Or here
SecondEffect(…)
ThirdEffect(…)
OSR exit state update
ExitOK

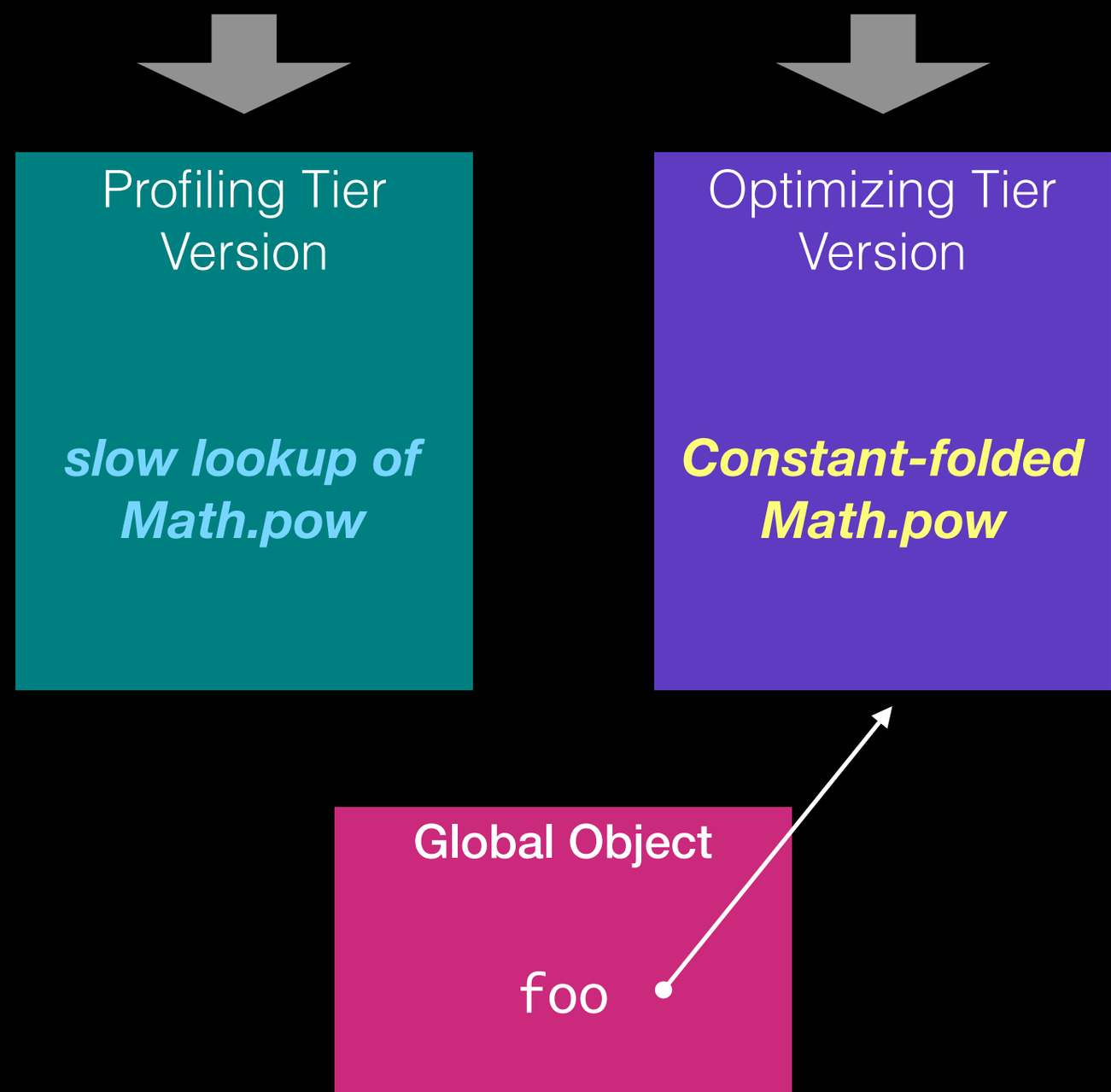

ExitOK
… some checks …
What if invalidation happens here
FirstEffect(…)
Or here
SecondEffect(…)
Nowhere to exit to!
ThirdEffect(…)
OSR exit state update
ExitOK


ExitOK
… some checks …
FirstEffect(…)
SecondEffect(…)
ThirdEffect(…)
OSR exit state update
ExitOK
InvalidationPoint

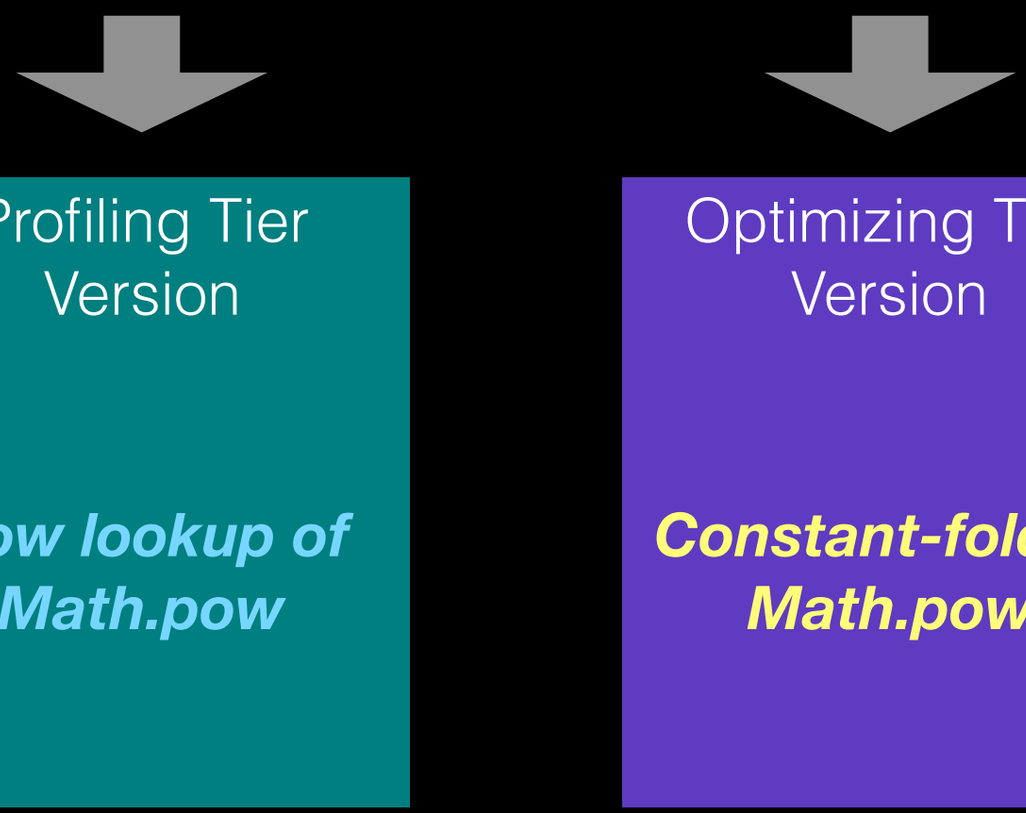
InvalidationPoint

- Deferred invalidation in case an in-progress effect has nowhere to exit.
- Emits no code.

```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

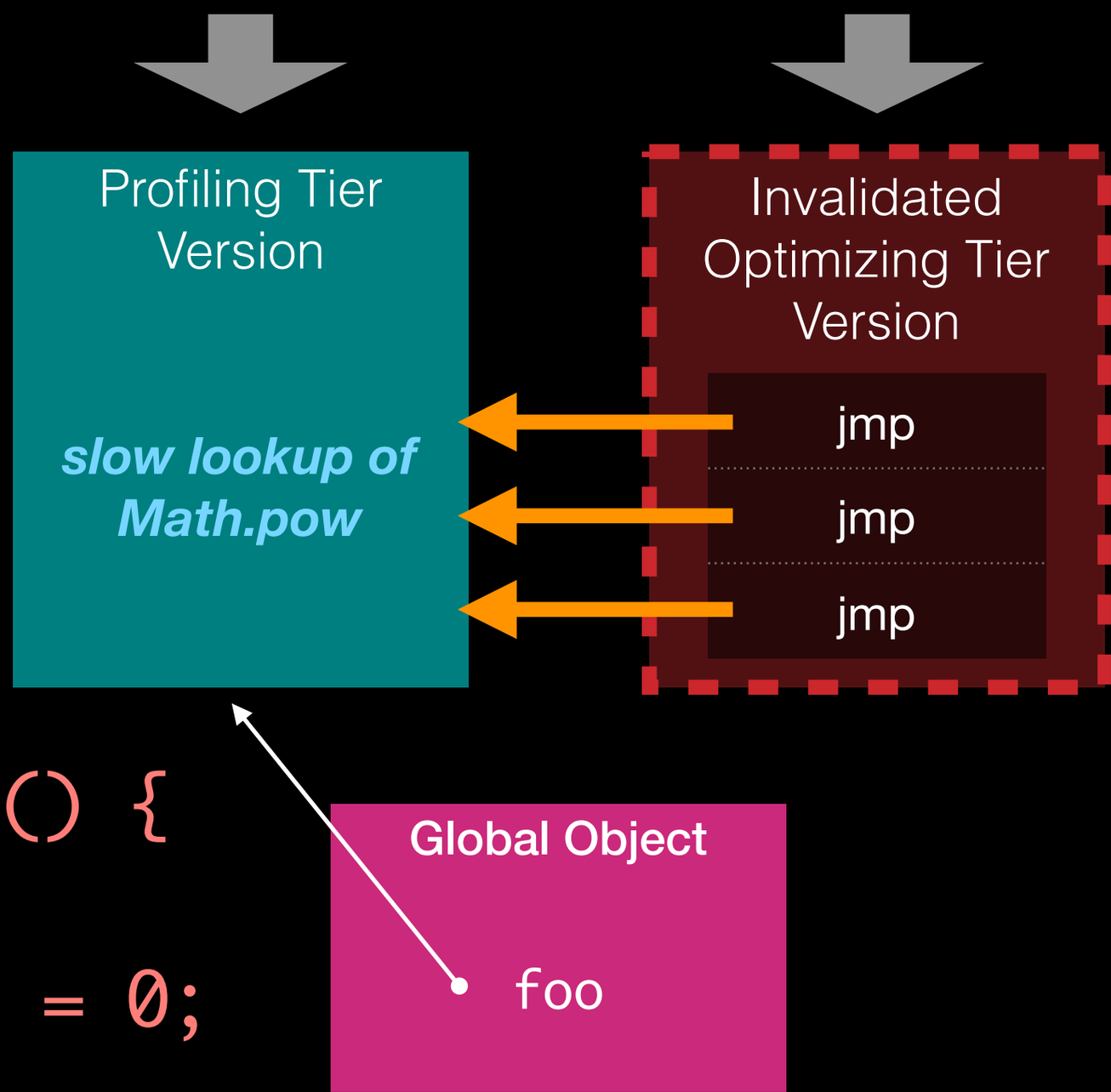


```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```




```
function bar() {
    if (p)
        Math = 0;
}
```

```
function foo() {
    bar();
    return Math.pow(2, 3);
}
```

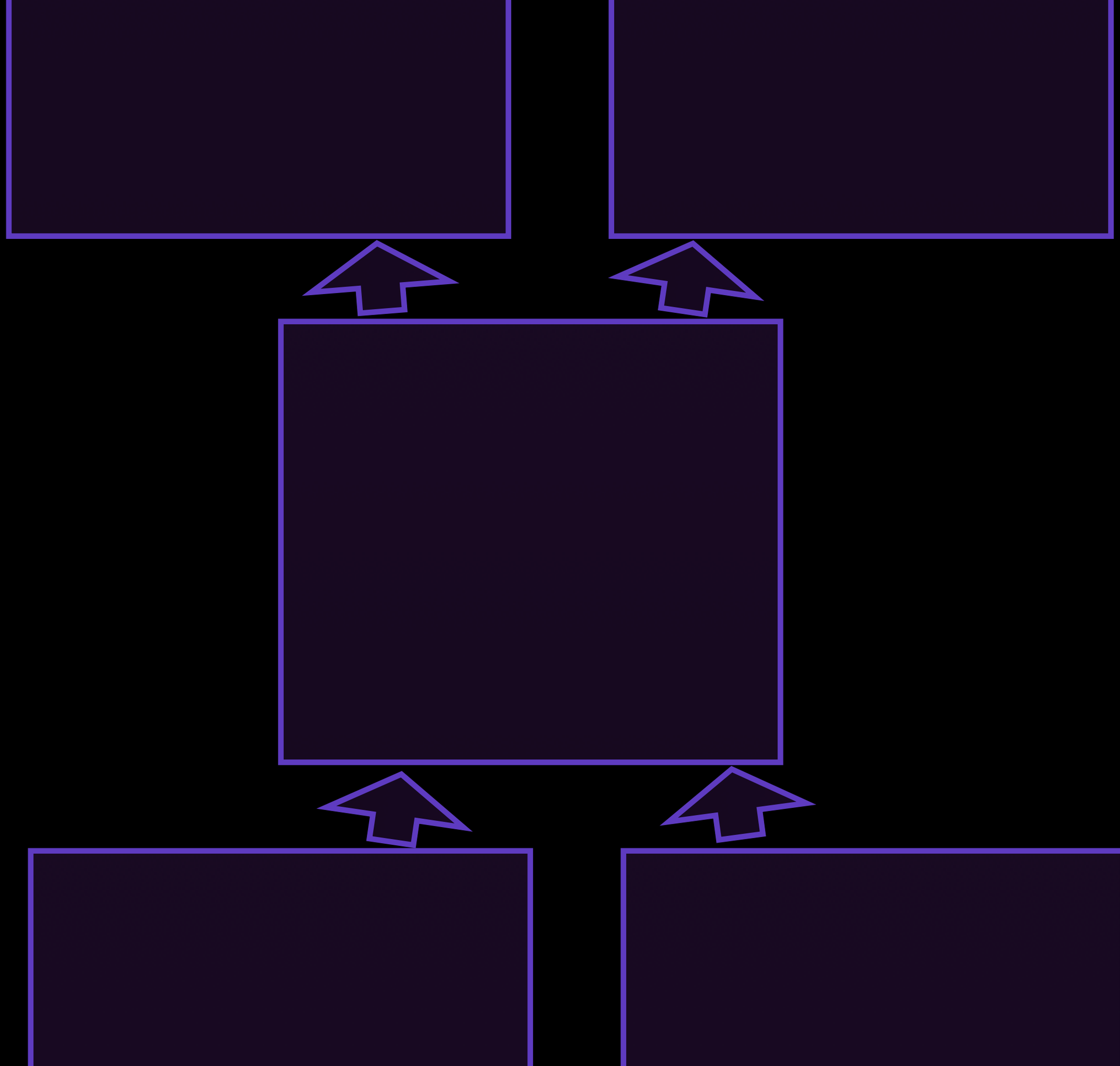


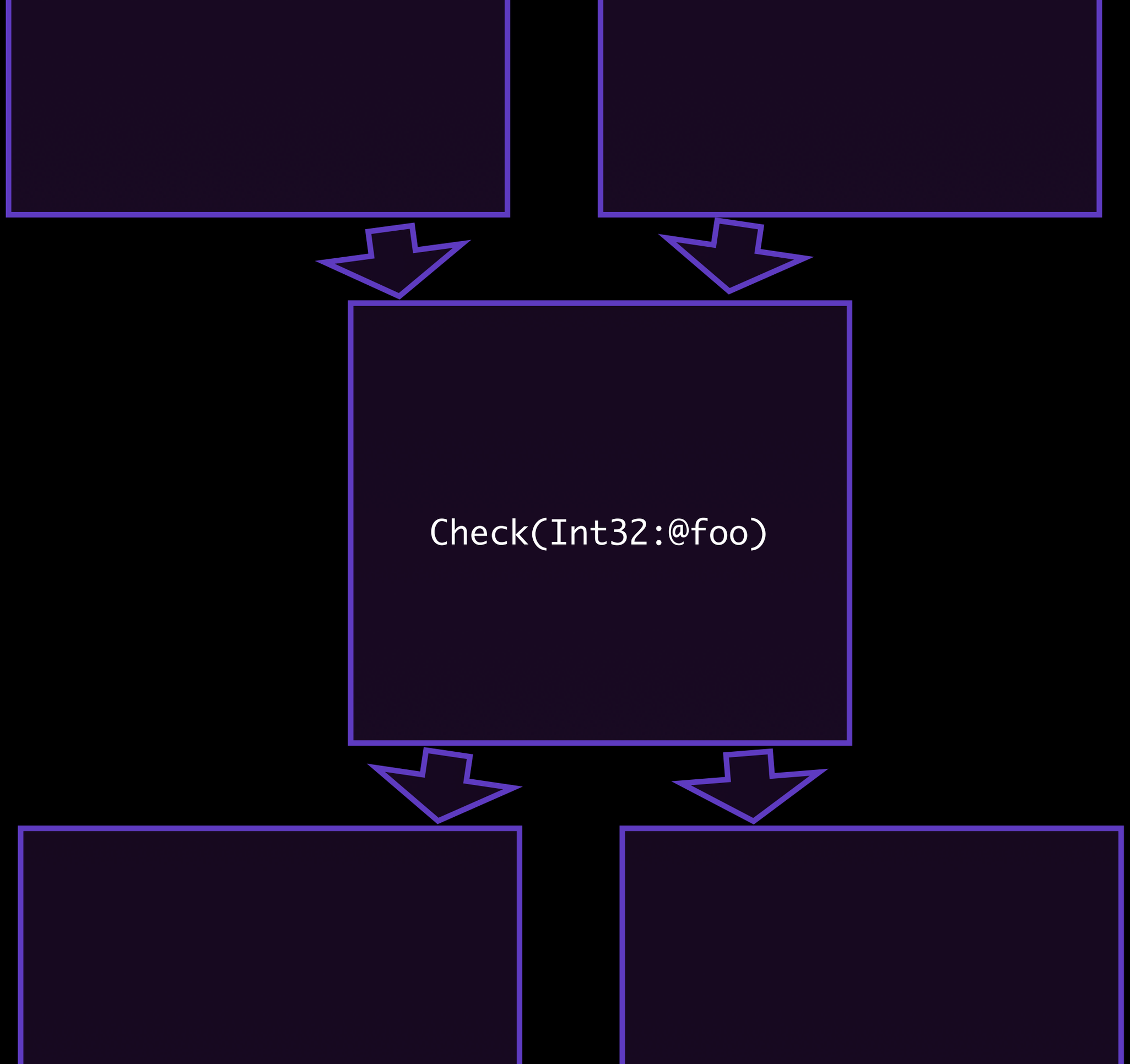

```
function bar() {
    if (p)
        Math = 0;
}
```

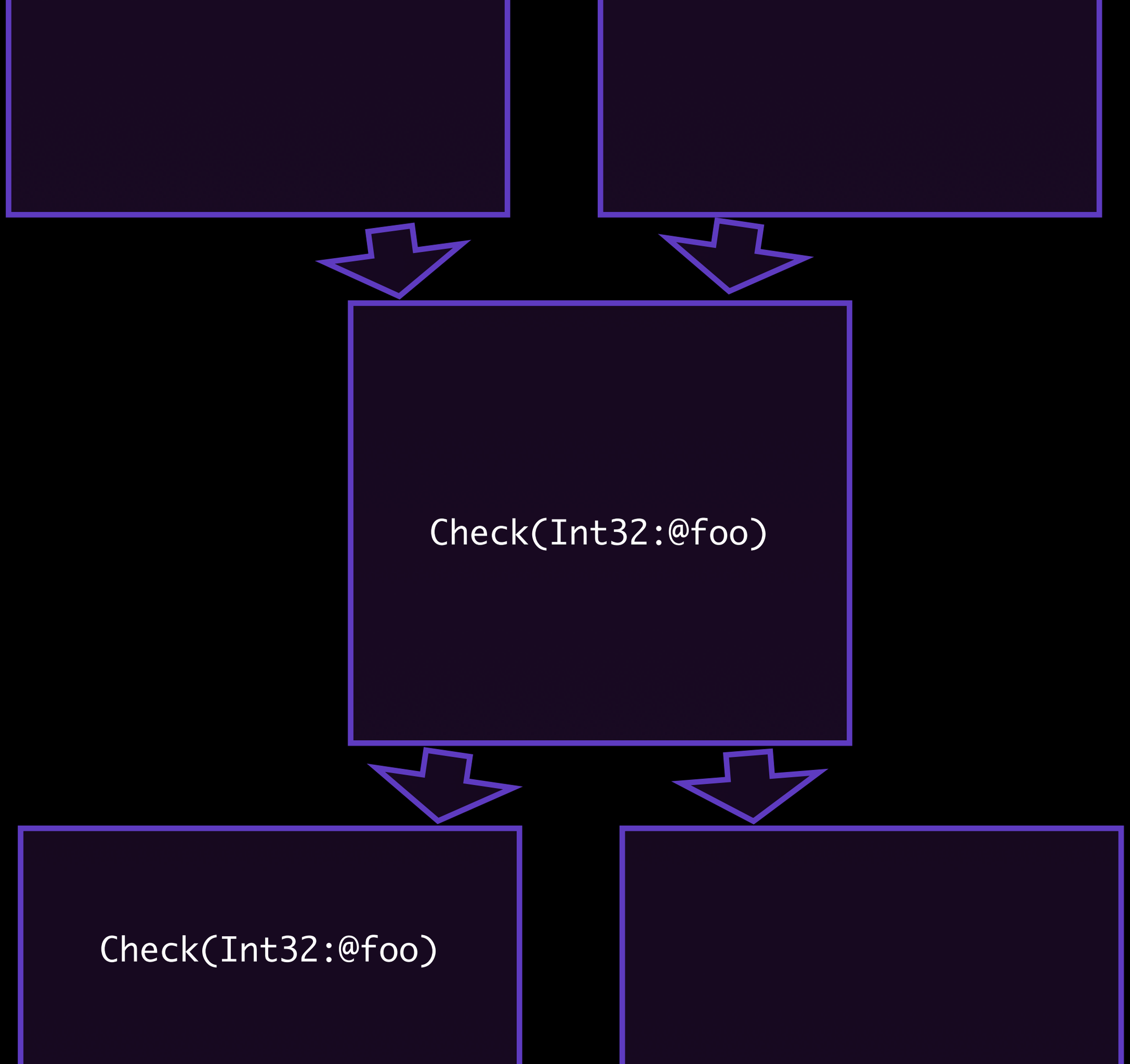
DFG Goals

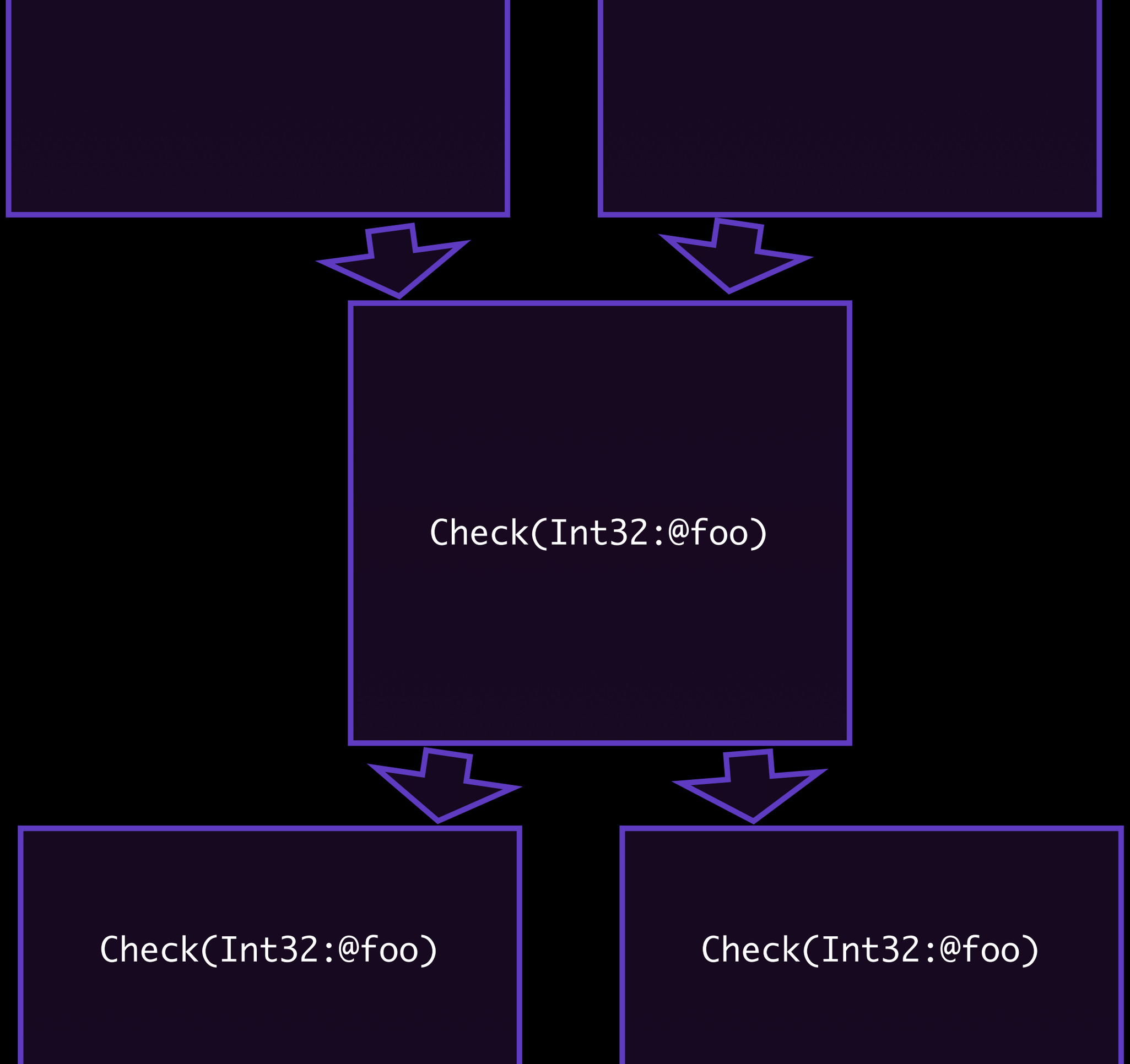
- Speculation
- Static Analysis
- Fast Compilation

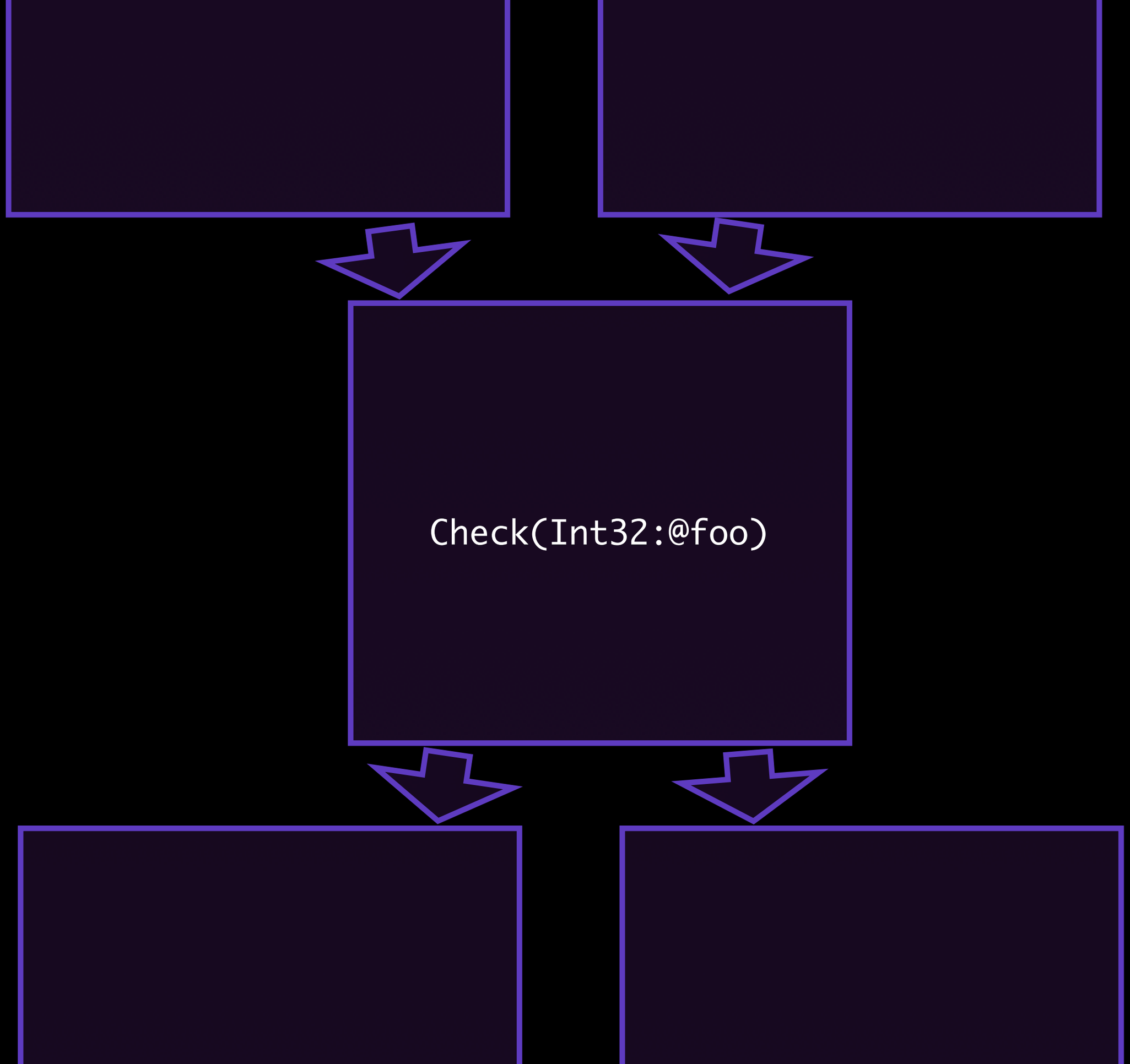
Remove type checks

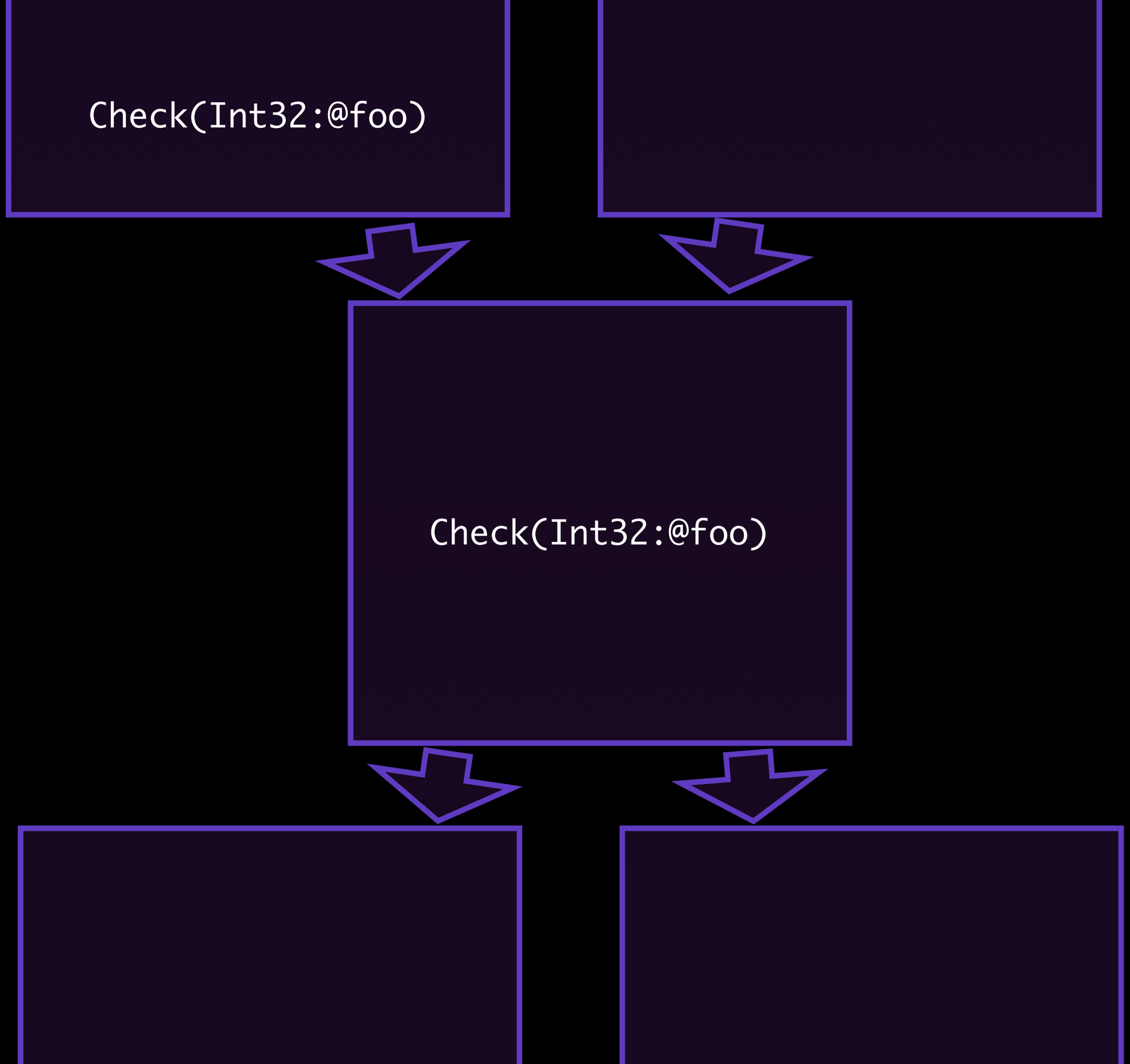


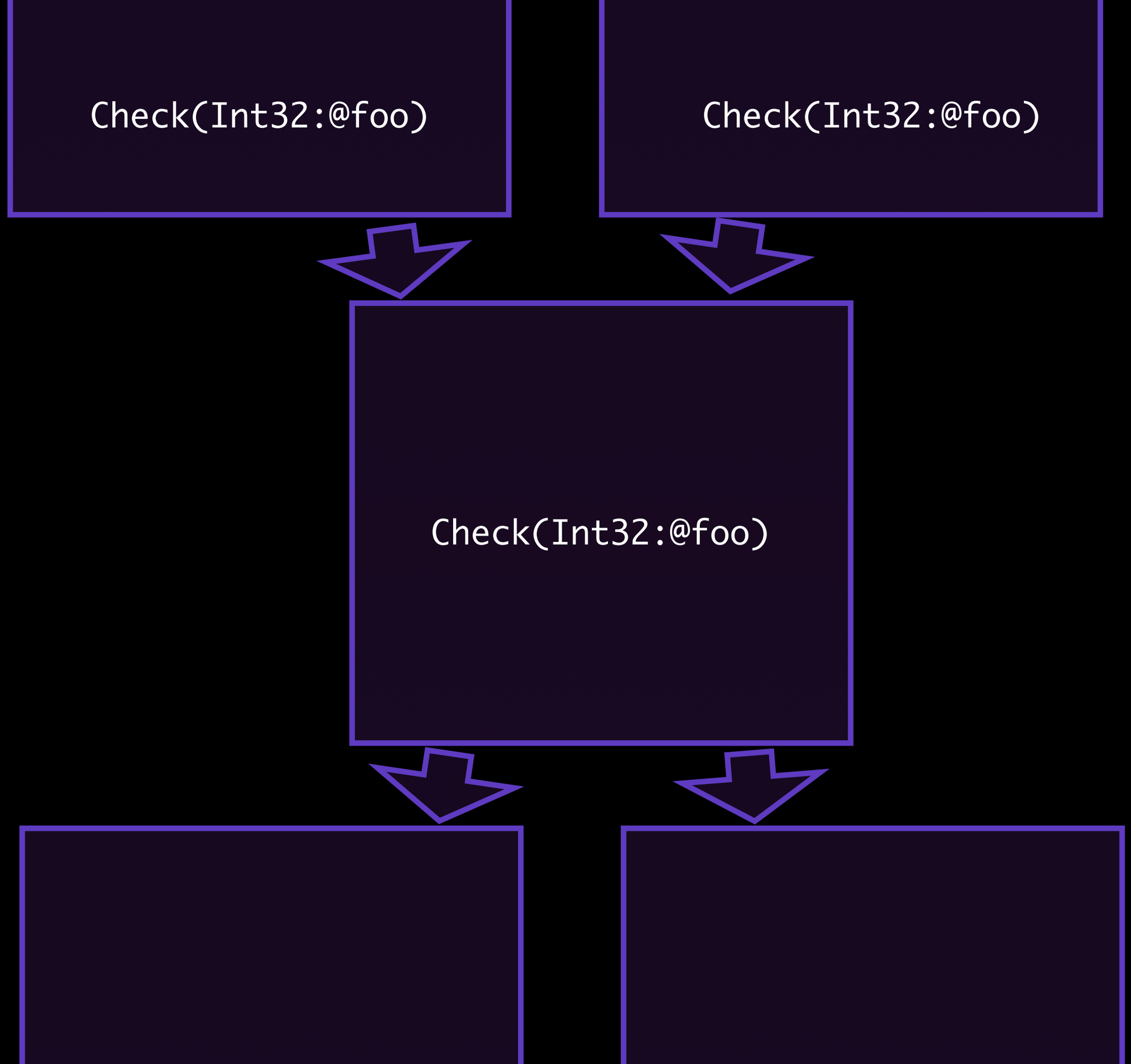

Check(Int32:@foo)

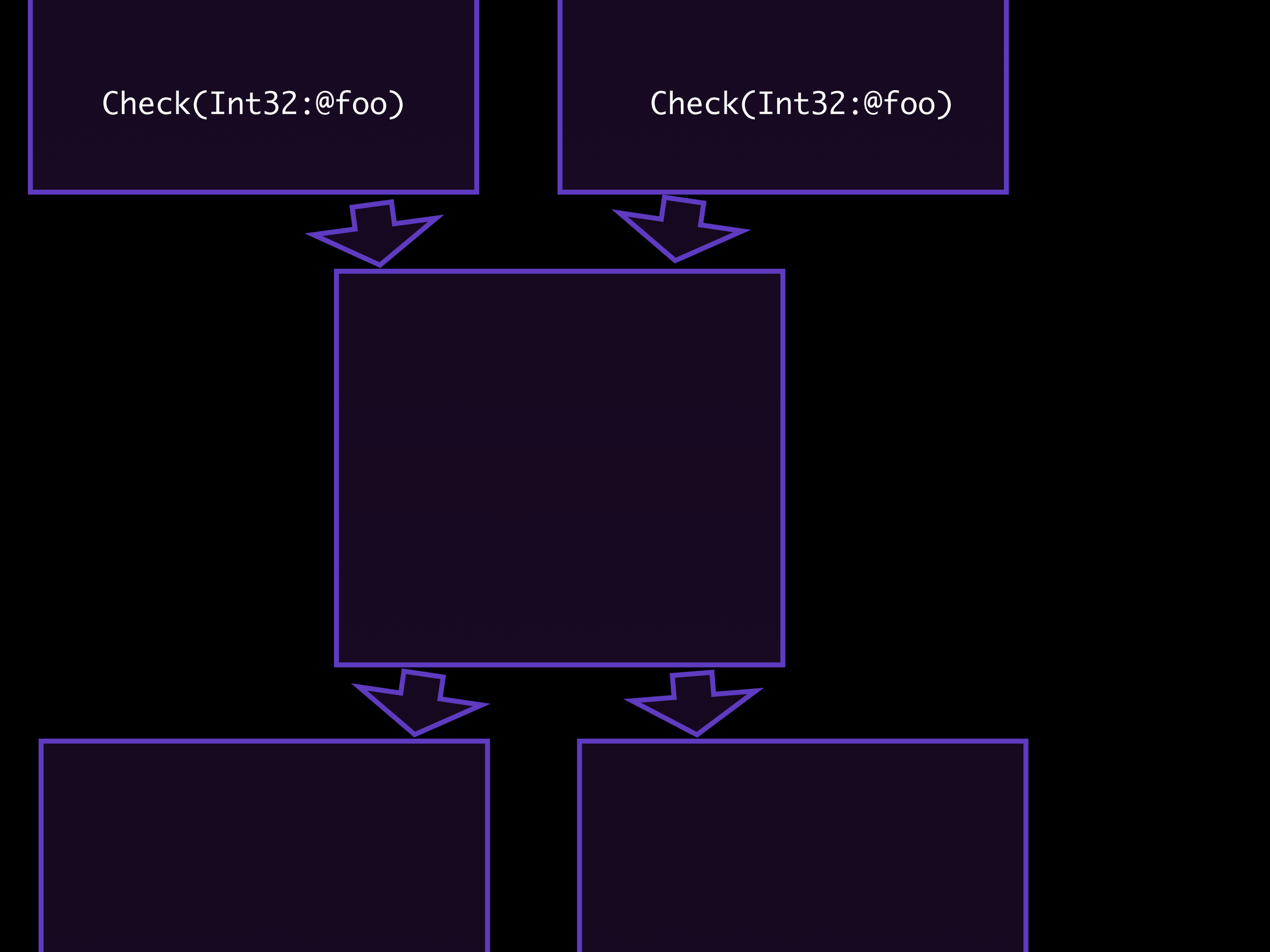

Check(Int32:@foo)
Check(Int32:@foo)


Check(Int32:@foo)
Check(Int32:@foo)
Check(Int32:@foo)


Check(Int32:@foo)


Check(Int32:@foo)
Check(Int32:@foo)


Check(Int32:@foo)
Check(Int32:@foo)
Check(Int32:@foo)


Check(Int32:@foo)
Check(Int32:@foo)

Abstract Interpreter

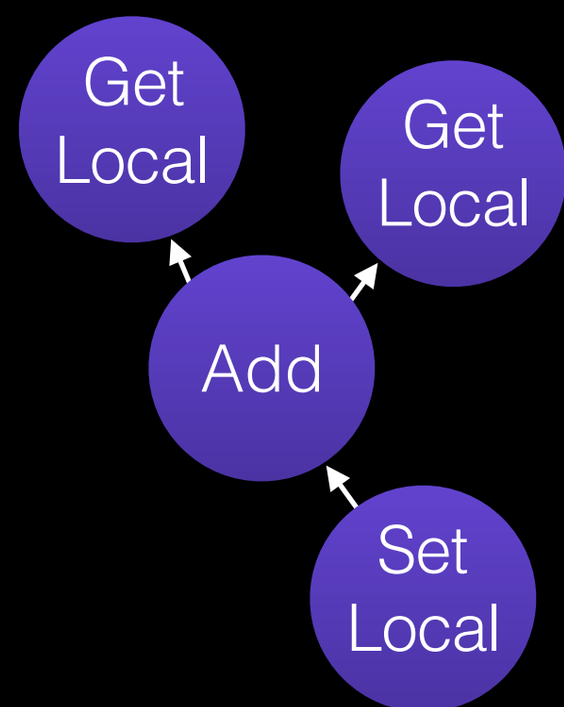
- “Global” (whole compilation unit)
- Flow sensitive
- Tracks:
  - variable type
  - object structure
  - indexing type
  - constants

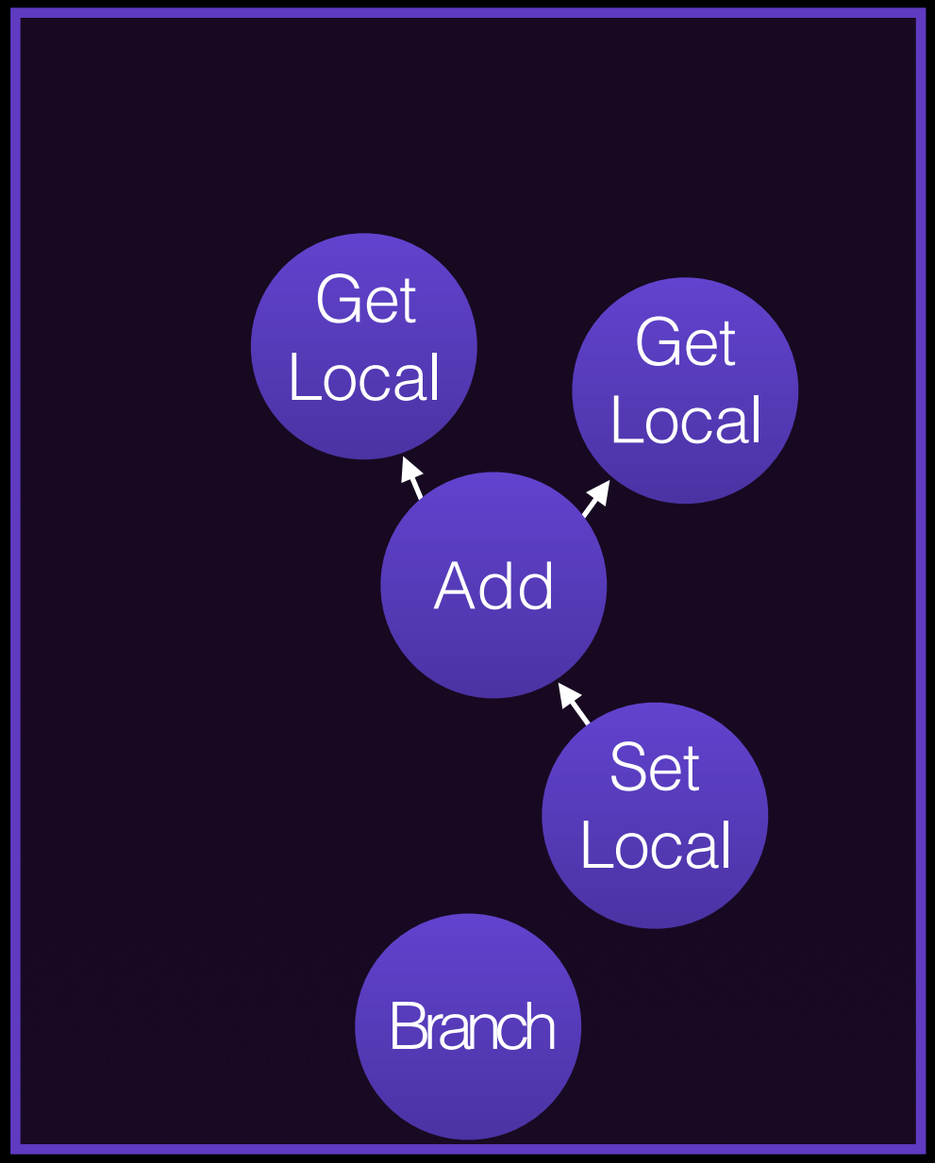
DFG Goals

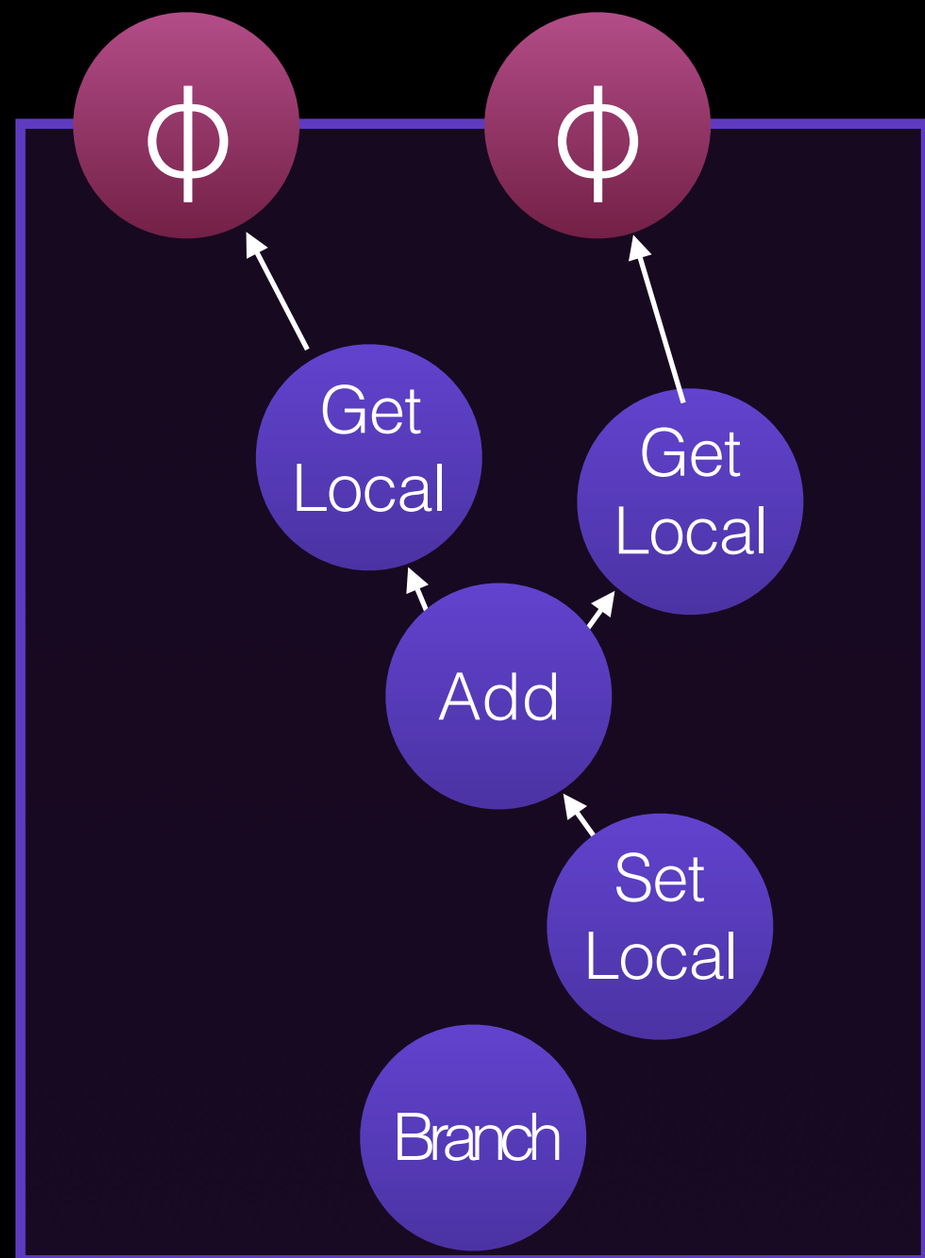
- Speculation
- Static Analysis
- Fast Compilation

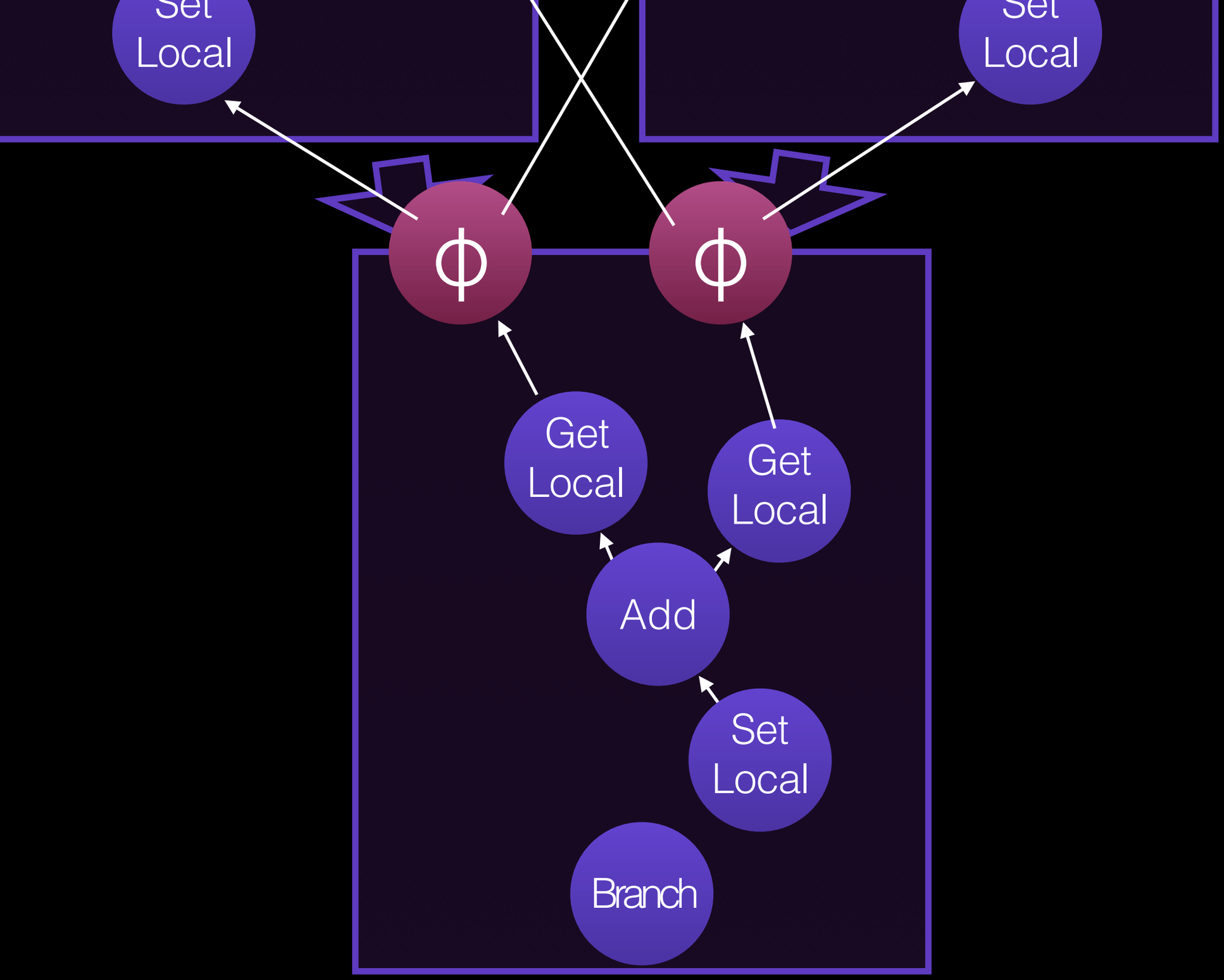
Fast Compile

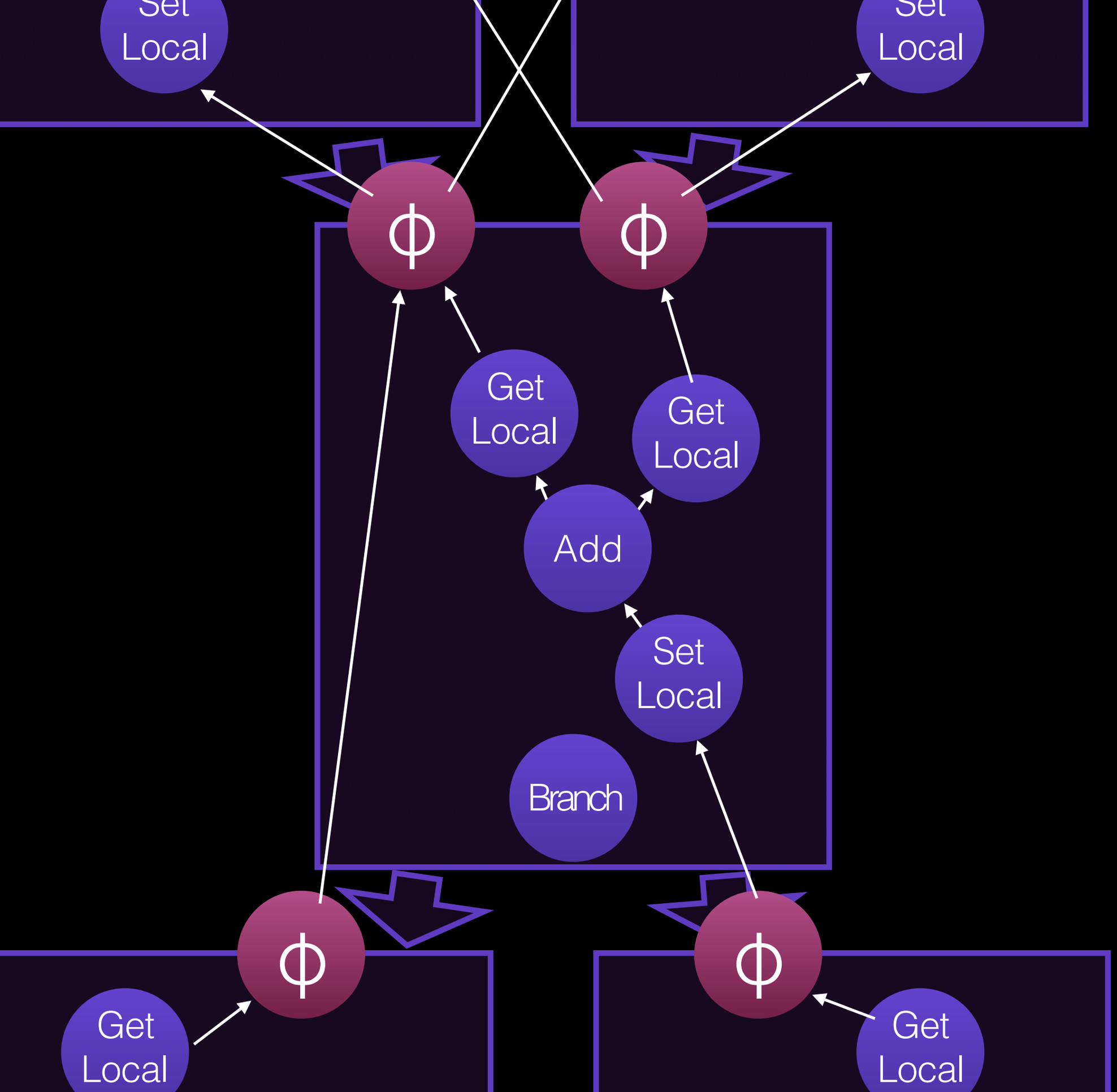
- Emphasis on block-locality.
- Template code generation.

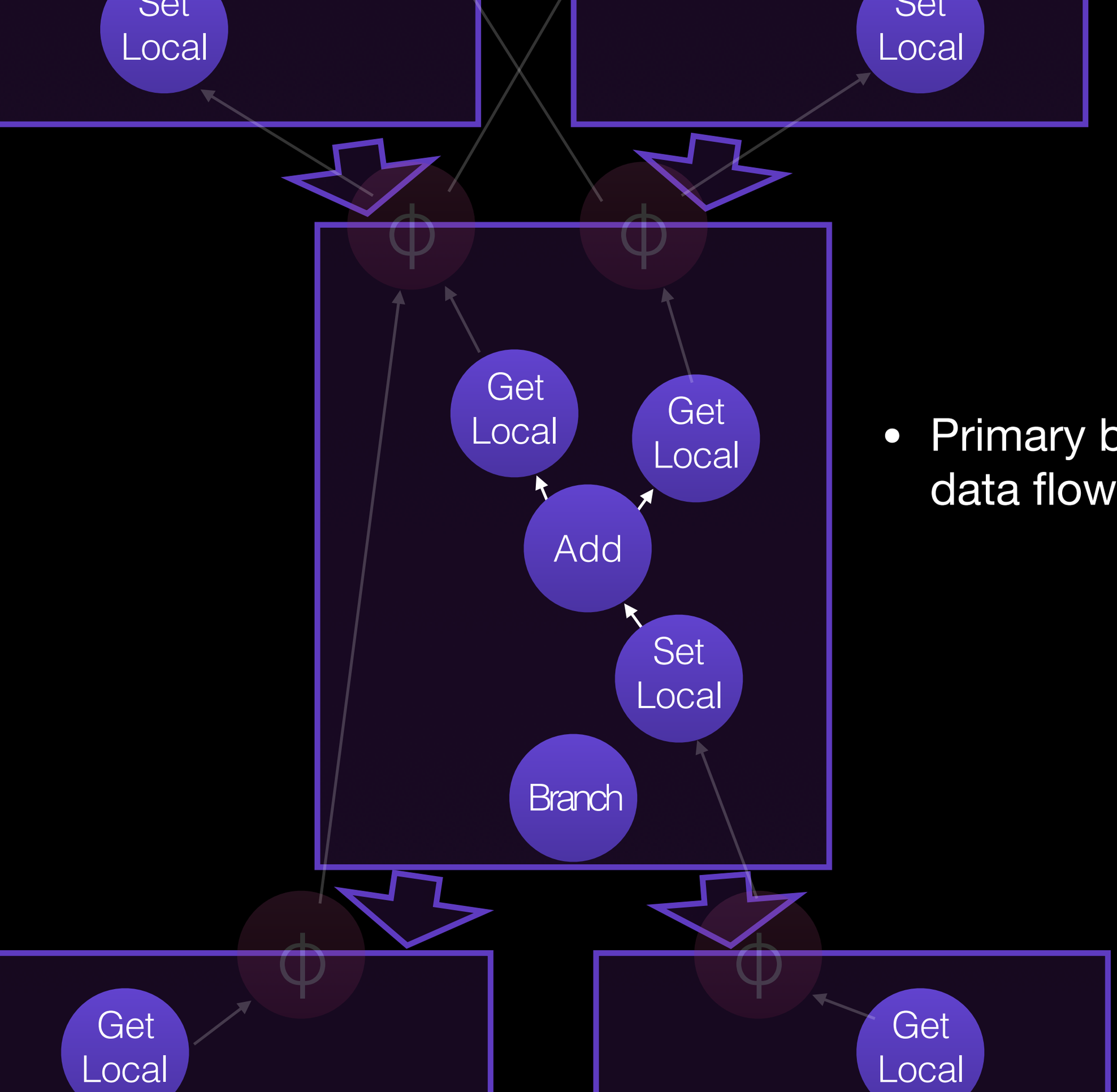

Get
Local
Get
Local
Add
Set
Local


Get
Local
Get
Local
Add
Set
Local
Branch

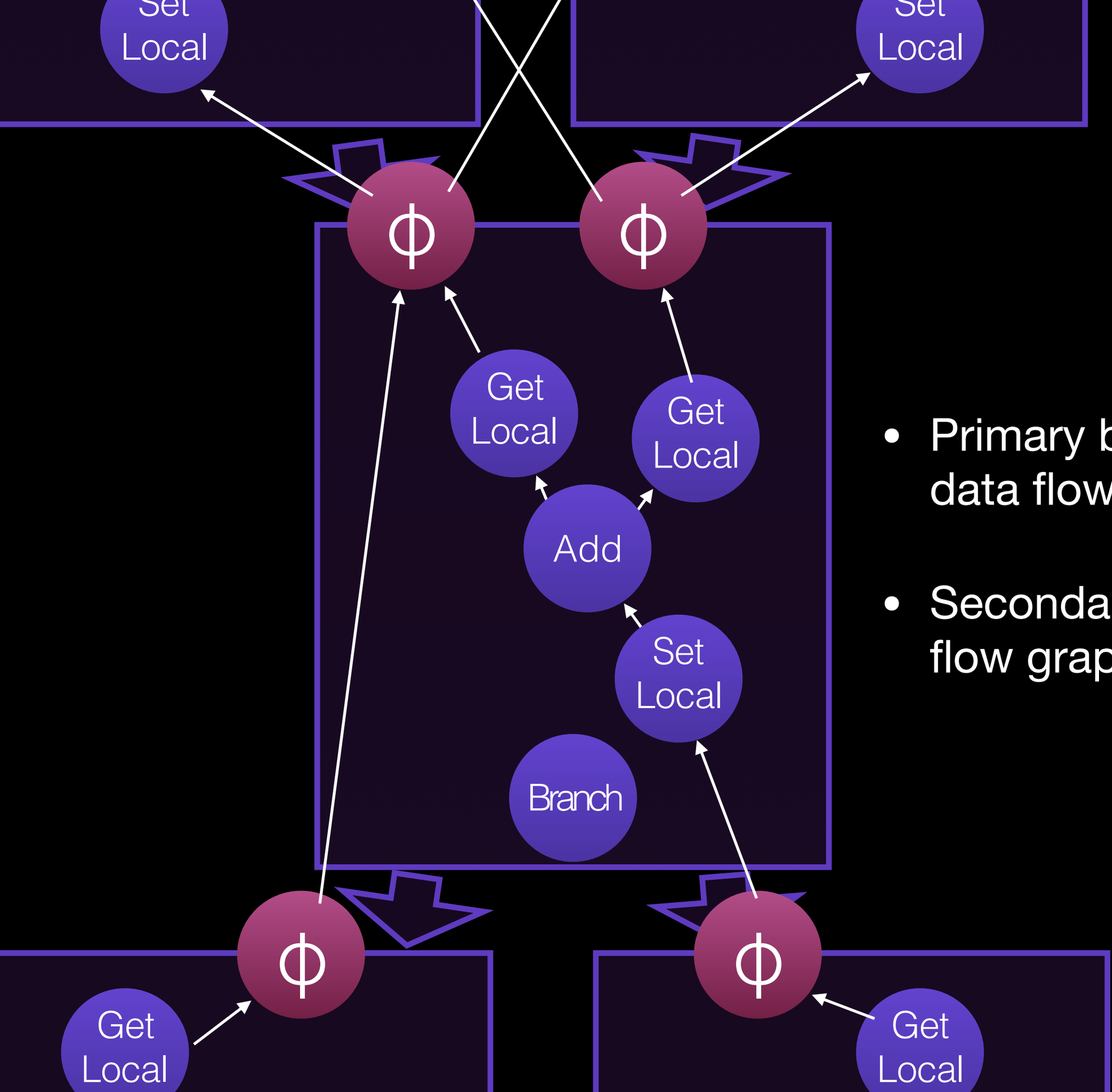

ϕ
ϕ
Get Local
Get Local
Add
Set Local
Branch


Set
Local
Set
Local
ϕ
ϕ
Get
Local
Get
Local
Add
Set
Local
Branch


Set
Local
Set
Local
ϕ
ϕ
Get
Local
Get
Local
Add
Set
Local
Branch
ϕ
ϕ
Get
Local
Get
Local



- Primary block-local data flow graph.



- Primary block-local data flow graph.
- Secondary global data flow graph.

DFG Template Codegen

```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

DFG Template Codegen

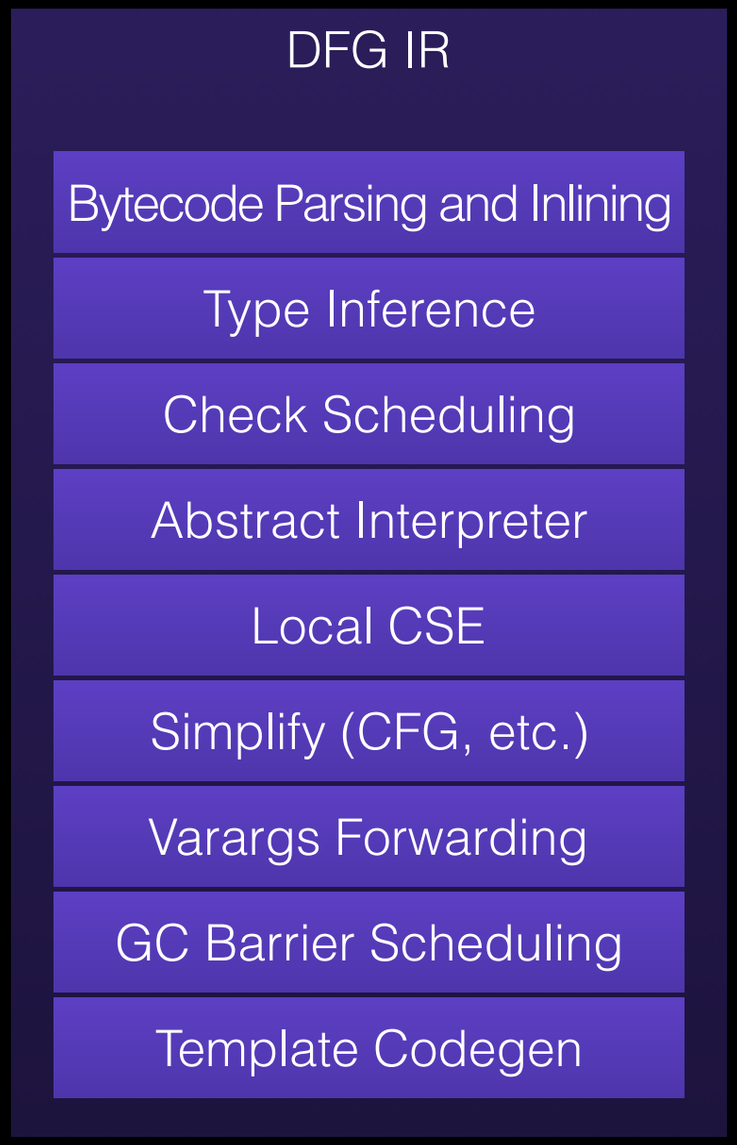
```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

DFG Template Codegen

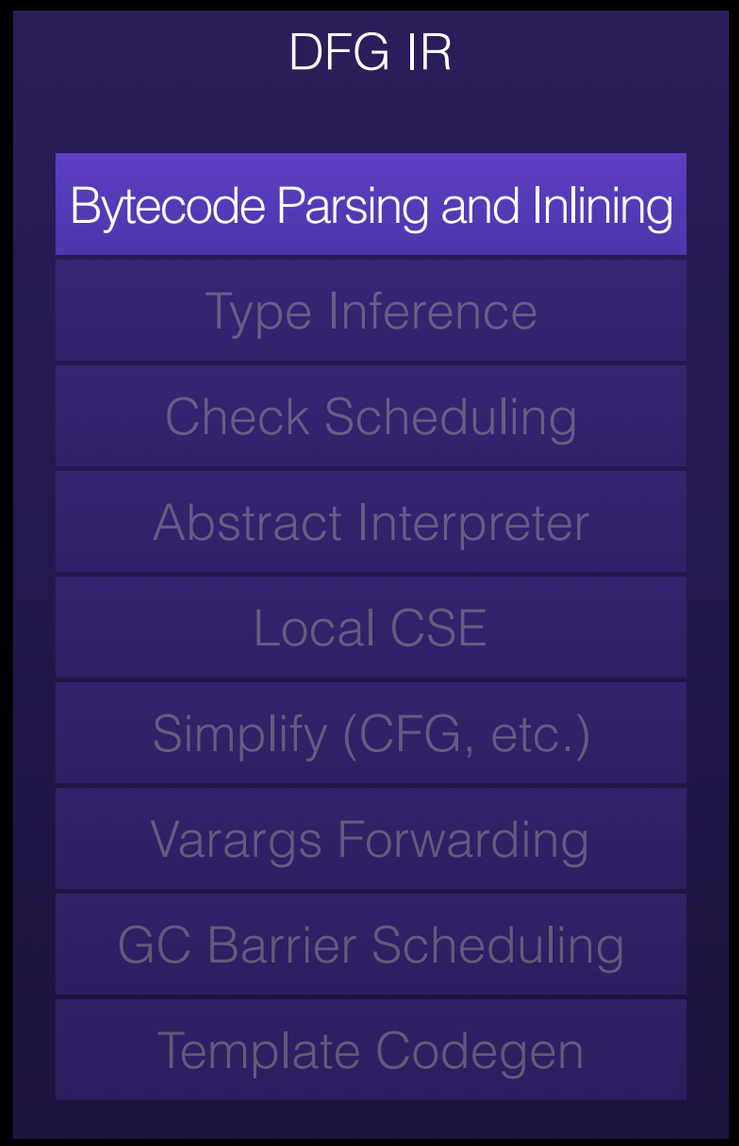
```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28:  Return(Untyped:@25, W:SideState, Exits, bc#12)
```

```
add %esi, %eax
jo Lexit
```

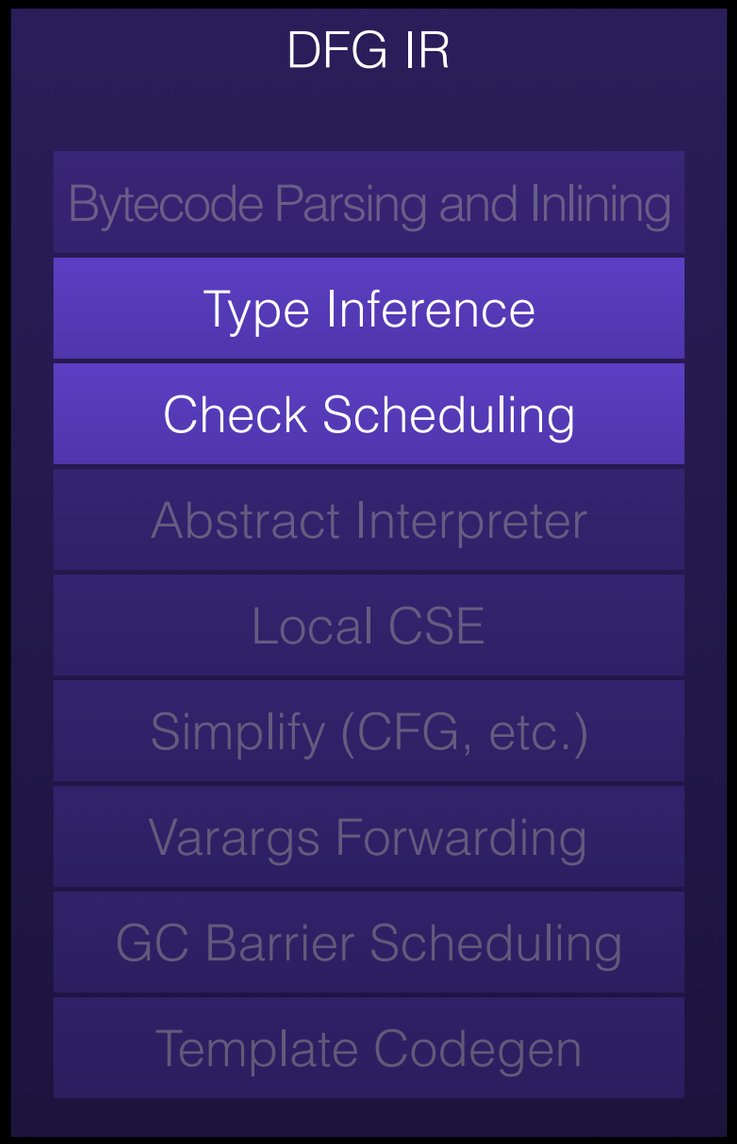
DFG optimization pipeline



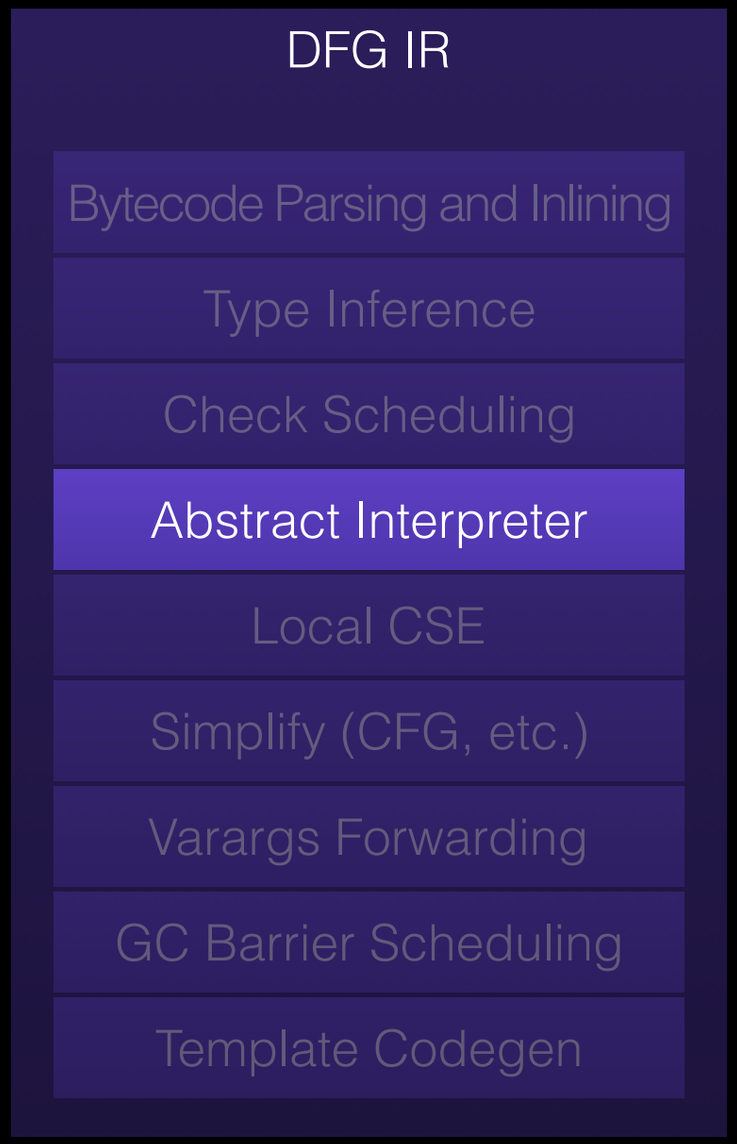
DFG optimization pipeline



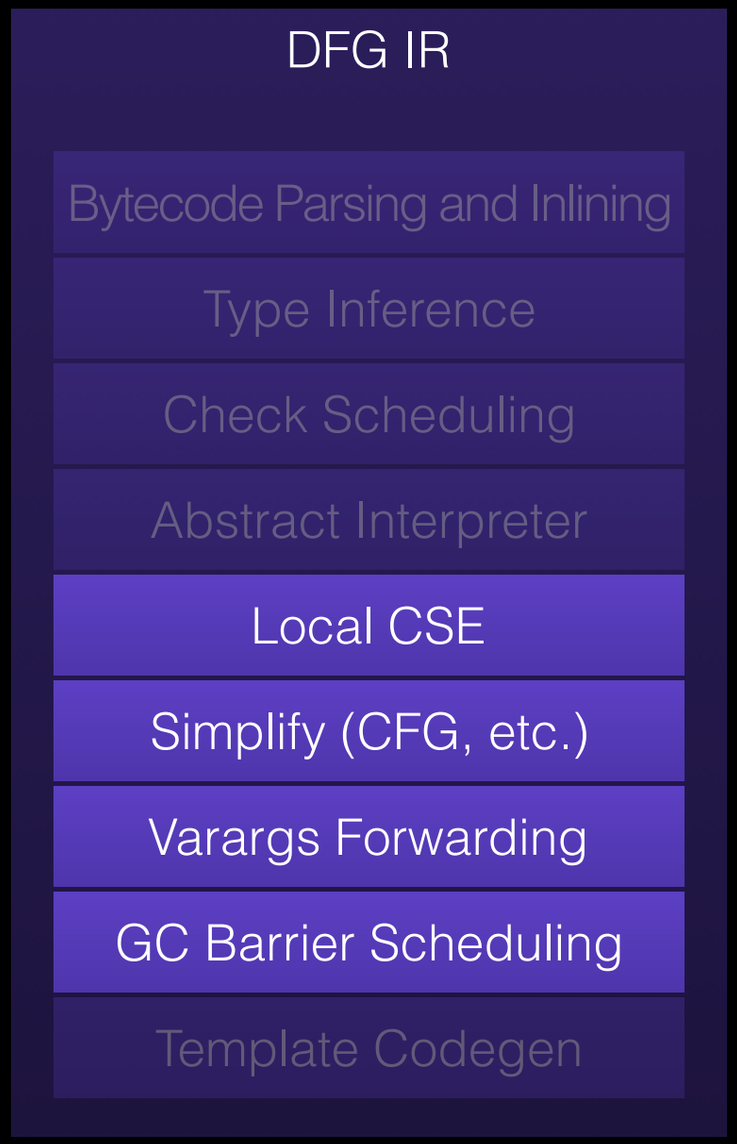
DFG optimization pipeline



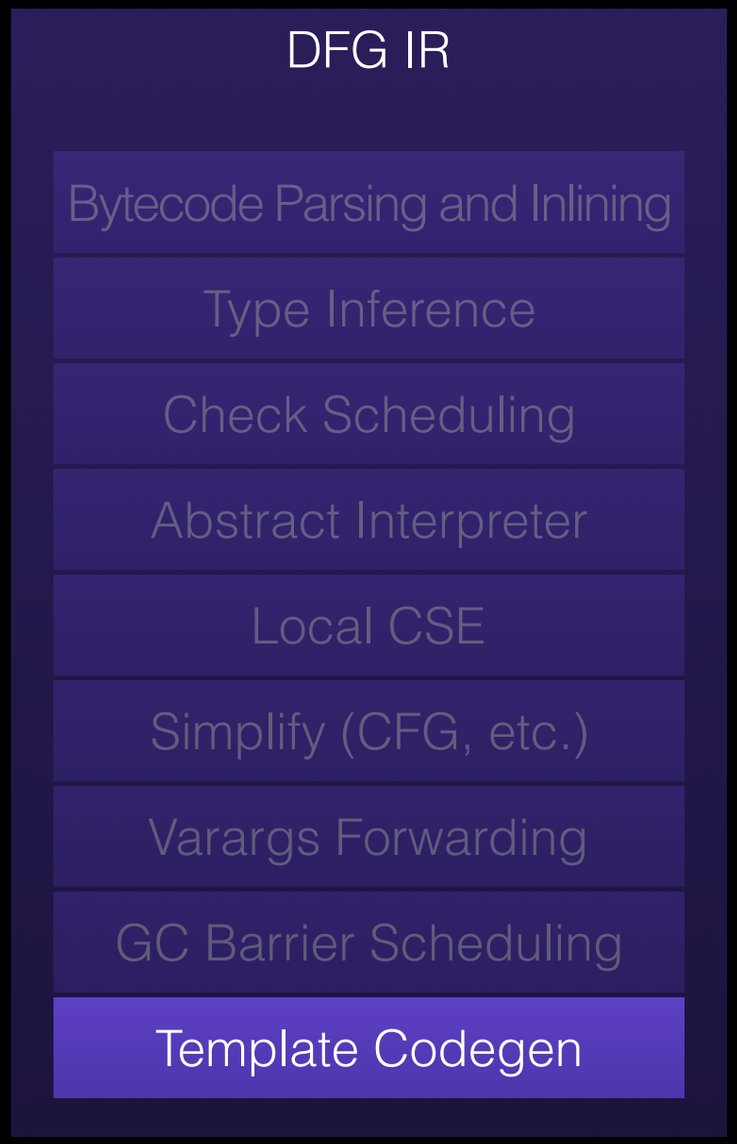
DFG optimization pipeline



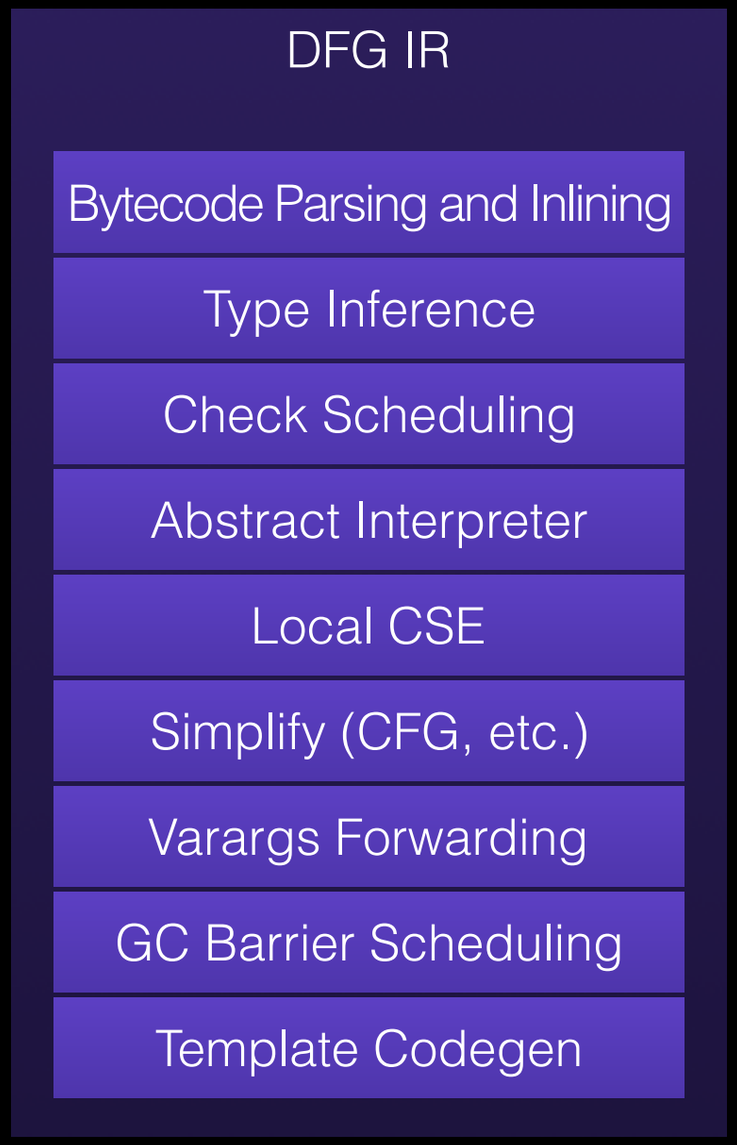
DFG optimization pipeline



DFG optimization pipeline

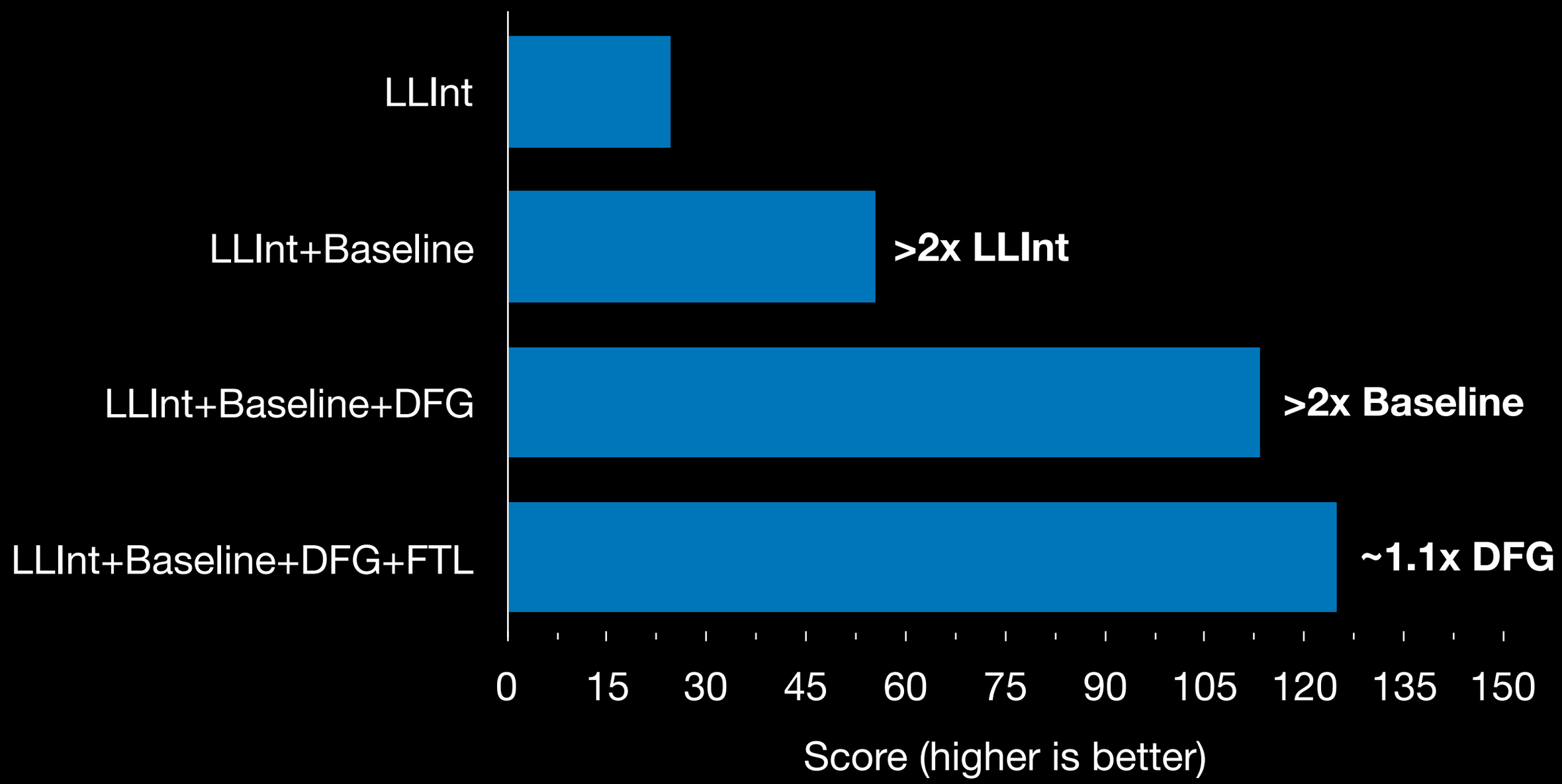


DFG optimization pipeline



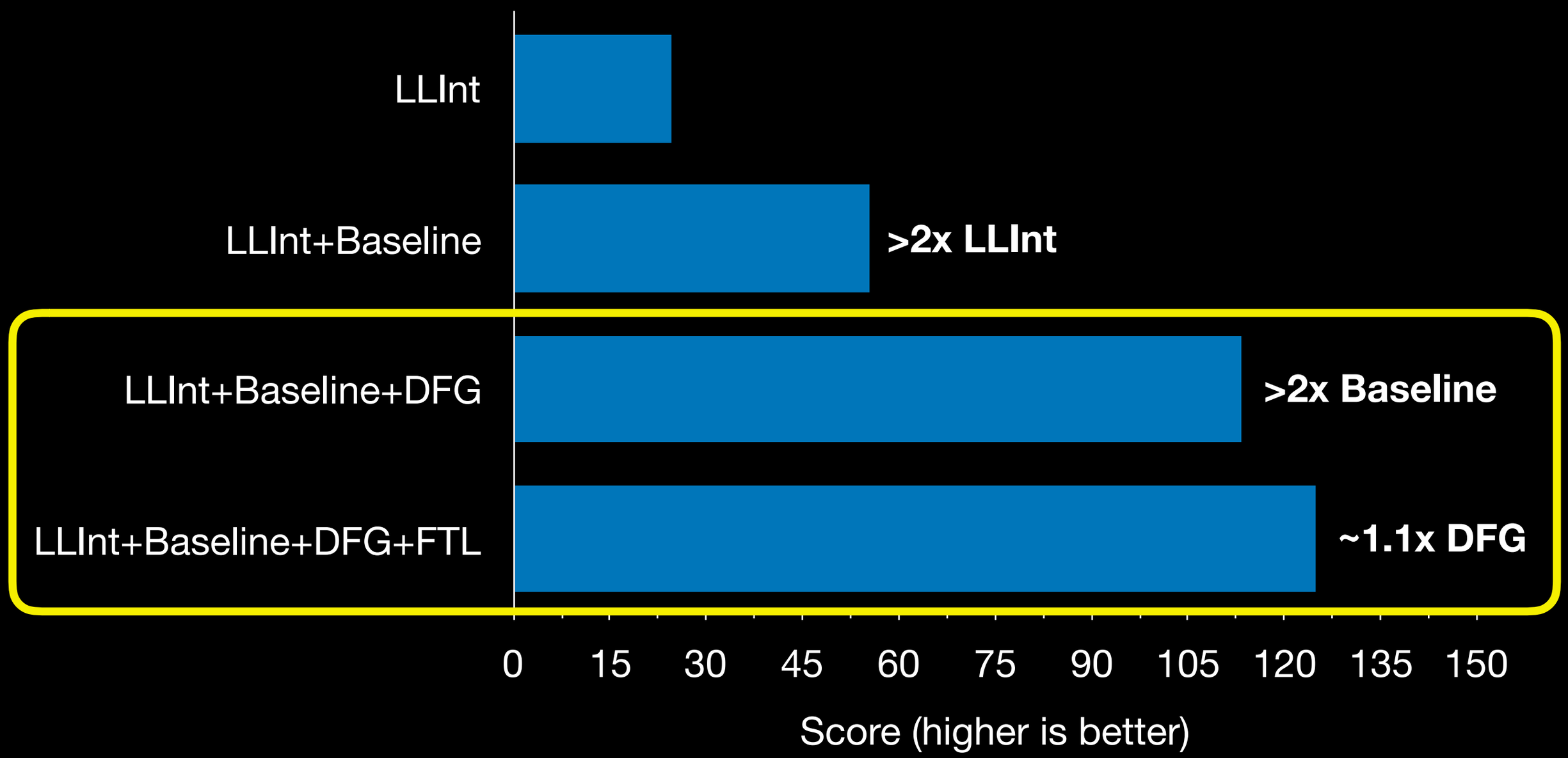
JetStream 2 Score

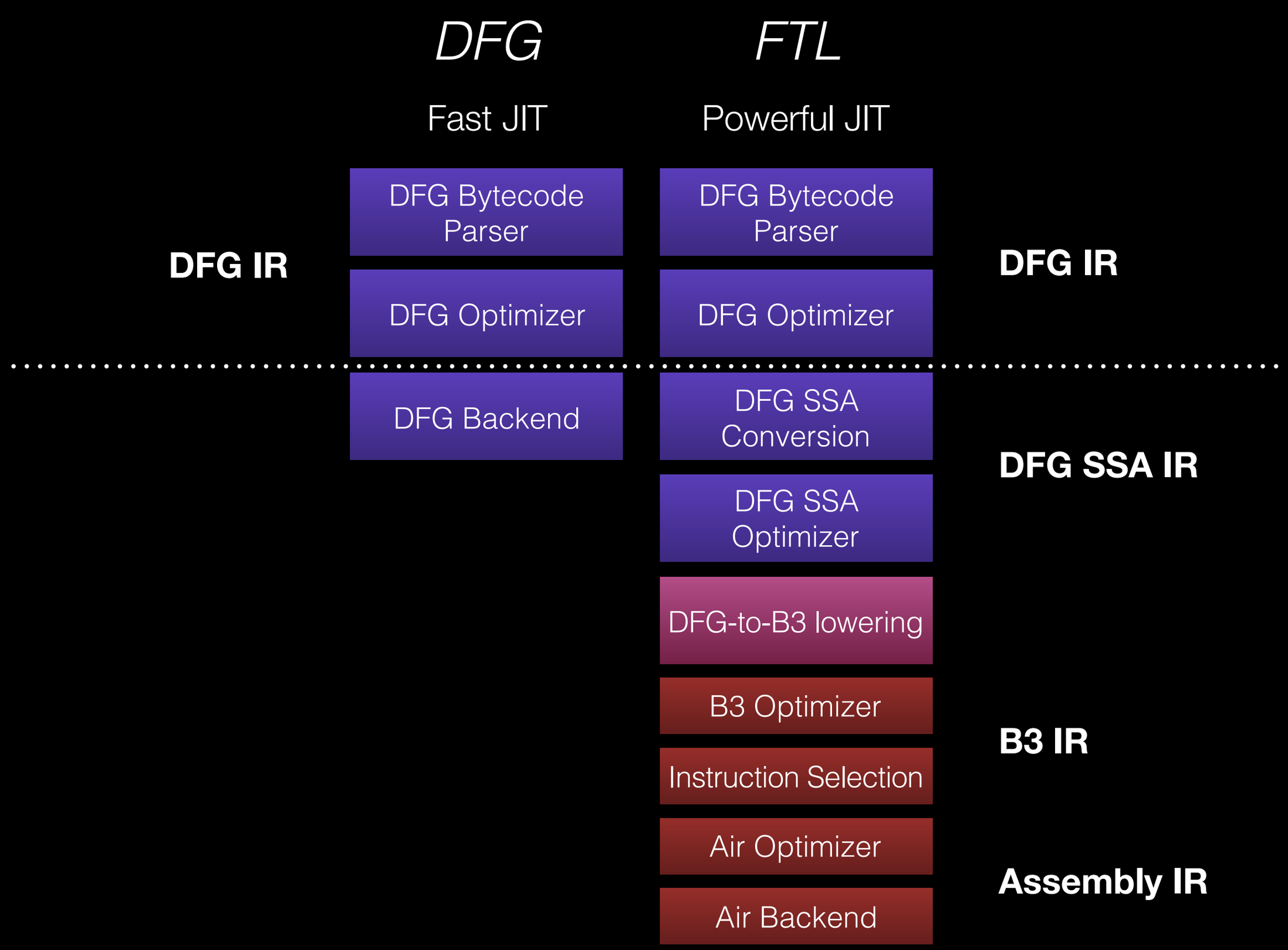
on my computer one day

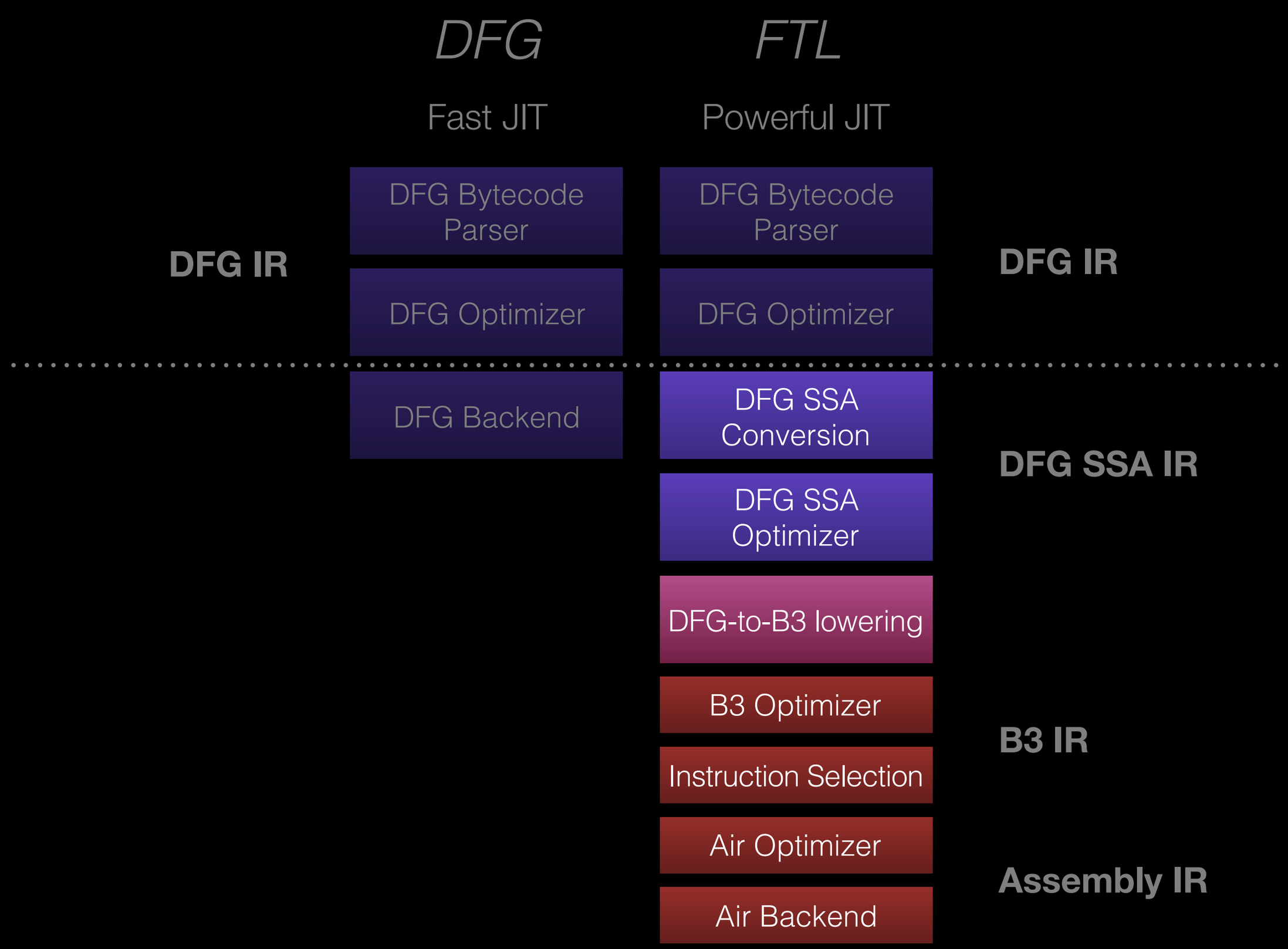


JetStream 2 Score

on my computer one day




DFG
FTL
Fast JIT
Powerful JIT
DFG Bytecode Parser
DFG Bytecode Parser
DFG IR
DFG IR
DFG Optimizer
DFG Optimizer
DFG Backend
DFG SSA Conversion
DFG SSA IR
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
B3 IR
Instruction Selection
Air Optimizer
Assembly IR
Air Backend


DFG
FTL
Fast JIT
Powerful JIT
DFG Bytecode Parser
DFG Bytecode Parser
DFG IR
DFG IR
DFG Optimizer
DFG Optimizer
DFG Backend
DFG SSA Conversion
DFG SSA IR
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
B3 IR
Instruction Selection
Air Optimizer
Assembly IR
Air Backend

FTL Goal

All the optimizations.

FTL IRs

IR	Style	Example
Bytecode	High Level Load/Store	`bitor dst, left, right`
DFG	Medium Level Exotic SSA	`dst: BitOr(Int32:@left,` `Int32:@right, …)`
B3	Low Level Normal SSA	`Int32 @dst =` `BitOr(@left, @right)`
Air	Architectural CISC	`Or32 %src, %dest`

FTL IRs

IR	Style	Example
Bytecode	High Level Load/Store	`bitor dst, left, right`
DFG	Medium Level Exotic SSA	`dst: BitOr(Int32:@left,` `Int32:@right, …)`
B3	Low Level Normal SSA	`Int32 @dst =` `BitOr(@left, @right)`
Air	Architectural CISC	`Or32 %src, %dest`

FTL IRs

IR	Style	Example
Bytecode	High Level Load/Store	`bitor dst, left, right`
DFG	Medium Level Exotic SSA	`dst: BitOr(Int32:@left,` `Int32:@right, …)`
B3	Low Level Normal SSA	`Int32 @dst =` `BitOr(@left, @right)`
Air	Architectural CISC	`Or32 %src, %dest`

FTL IRs

IR	Style	Example
Bytecode	High Level Load/Store	`bitor dst, left, right`
DFG	Medium Level Exotic SSA	`dst: BitOr(Int32:@left,` `Int32:@right, …)`
B3	Low Level Normal SSA	`Int32 @dst =` `BitOr(@left, @right)`
Air	Architectural CISC	`Or32 %src, %dest`

FTL IRs

IR	Style	Example
Bytecode	High Level Load/Store	`bitor dst, left, right`
DFG	Medium Level Exotic SSA	`dst: BitOr(Int32:@left,` `Int32:@right, …)`
B3	Low Level Normal SSA	`Int32 @dst =` `BitOr(@left, @right)`
Air	Architectural CISC	`Or32 %src, %dest`

FTL optimization pipeline

DFG IR	B3 IR	Air
Bytecode Parsing and Inlining	Double-to-Float	Simplify CFG
Type Inference	Simplify (folding, CFG, etc)	Macro Lowering
Check Scheduling	LICM	DCE
Simplify (CFG etc)	Global CSE	Graph Coloring Reg Alloc
Abstract Interpretation	Switch Inference	Spill CSE
Global CSE	Tail Duplication	Graph Coloring Stack Alloc
Escape Analysis	Path Constants	Report Used Registers
LICM	Macro Lowering	Fix Partial Register Stalls
Integer Range Optimization	Legalization	Lower Multiple Entrypoints
GC Barrier Scheduling	Constant Motion	Select Block Order
Lower to B3 IR	Lower to Air (isel)	Emit Machine Code

FTL optimization pipeline

DFG IR	B3 IR	Air
Bytecode Parsing and Inlining	Double-to-Float	Simplify CFG
Type Inference	Simplify (folding, CFG, etc)	Macro Lowering
Check Scheduling	LICM	DCE
Simplify (CFG etc)	Global CSE	Graph Coloring Reg Alloc
Abstract Interpretation	Switch Inference	Spill CSE
Global CSE	Tail Duplication	Graph Coloring Stack Alloc
Escape Analysis	Path Constants	Report Used Registers
LICM	Macro Lowering	Fix Partial Register Stalls
Integer Range Optimization	Legalization	Lower Multiple Entrypoints
GC Barrier Scheduling	Constant Motion	Select Block Order
Lower to B3 IR	Lower to Air (isel)	Emit Machine Code

FTL optimization pipeline

DFG IR	B3 IR	Air
Bytecode Parsing and Inlining	Double-to-Float	Simplify CFG
Type Inference	Simplify (folding, CFG, etc)	Macro Lowering
Check Scheduling	LICM	DCE
Simplify (CFG etc)	Global CSE	Graph Coloring Reg Alloc
Abstract Interpretation	Switch Inference	Spill CSE
Global CSE	Tail Duplication	Graph Coloring Stack Alloc
Escape Analysis	Path Constants	Report Used Registers
LICM	Macro Lowering	Fix Partial Register Stalls
Integer Range Optimization	Legalization	Lower Multiple Entrypoints
GC Barrier Scheduling	Constant Motion	Select Block Order
Lower to B3 IR	Lower to Air (isel)	Emit Machine Code

FTL optimization pipeline

DFG IR	B3 IR	Air
Bytecode Parsing and Inlining	Double-to-Float	Simplify CFG
Type Inference	Simplify (folding, CFG, etc)	Macro Lowering
Check Scheduling	LICM	DCE
Simplify (CFG etc)	Global CSE	Graph Coloring Reg Alloc
Abstract Interpretation	Switch Inference	Spill CSE
Global CSE	Tail Duplication	Graph Coloring Stack Alloc
Escape Analysis	Path Constants	Report Used Registers
LICM	Macro Lowering	Fix Partial Register Stalls
Integer Range Optimization	Legalization	Lower Multiple Entrypoints
GC Barrier Scheduling	Constant Motion	Select Block Order
Lower to B3 IR	Lower to Air (isel)	Emit Machine Code

FTL optimization pipeline

DFG IR	B3 IR	Air
Bytecode Parsing and Inlining	Double-to-Float	Simplify CFG
Type Inference	Simplify (folding, CFG, etc)	Macro Lowering
Check Scheduling	LICM	DCE
Simplify (CFG etc)	Global CSE	Graph Coloring Reg Alloc
Abstract Interpretation	Switch Inference	Spill CSE
Global CSE	Tail Duplication	Graph Coloring Stack Alloc
Escape Analysis	Path Constants	Report Used Registers
LICM	Macro Lowering	Fix Partial Register Stalls
Integer Range Optimization	Legalization	Lower Multiple Entrypoints
GC Barrier Scheduling	Constant Motion	Select Block Order
Lower to B3 IR	Lower to Air (isel)	Emit Machine Code

FTL optimization pipeline

DFG IR	B3 IR	Air
Bytecode Parsing and Inlining	Double-to-Float	Simplify CFG
Type Inference	Simplify (folding, CFG, etc)	Macro Lowering
Check Scheduling	LICM	DCE
Simplify (CFG etc)	Global CSE	Graph Coloring Reg Alloc
Abstract Interpretation	Switch Inference	Spill CSE
Global CSE	Tail Duplication	Graph Coloring Stack Alloc
Escape Analysis	Path Constants	Report Used Registers
LICM	Macro Lowering	Fix Partial Register Stalls
Integer Range Optimization	Legalization	Lower Multiple Entrypoints
GC Barrier Scheduling	Constant Motion	Select Block Order
Lower to B3 IR	Lower to Air (isel)	Emit Machine Code

FTL optimization pipeline

DFG IR	B3 IR	Air
Bytecode Parsing and Inlining	Double-to-Float	Simplify CFG
Type Inference	Simplify (folding, CFG, etc)	Macro Lowering
Check Scheduling	LICM	DCE
Simplify (CFG etc)	Global CSE	Graph Coloring Reg Alloc
Abstract Interpretation	Switch Inference	Spill CSE
Global CSE	Tail Duplication	Graph Coloring Stack Alloc
Escape Analysis	Path Constants	Report Used Registers
LICM	Macro Lowering	Fix Partial Register Stalls
Integer Range Optimization	Legalization	Lower Multiple Entrypoints
GC Barrier Scheduling	Constant Motion	Select Block Order
Lower to B3 IR	Lower to Air (isel)	Emit Machine Code

FTL optimization pipeline

DFG IR	B3 IR	Air
Bytecode Parsing and Inlining	Double-to-Float	Simplify CFG
Type Inference	Simplify (folding, CFG, etc)	Macro Lowering
Check Scheduling	LICM	DCE
Simplify (CFG etc)	Global CSE	Graph Coloring Reg Alloc
Abstract Interpretation	Switch Inference	Spill CSE
Global CSE	Tail Duplication	Graph Coloring Stack Alloc
Escape Analysis	Path Constants	Report Used Registers
LICM	Macro Lowering	Fix Partial Register Stalls
Integer Range Optimization	Legalization	Lower Multiple Entrypoints
GC Barrier Scheduling	Constant Motion	Select Block Order
Lower to B3 IR	Lower to Air (isel)	Emit Machine Code

Source

```
function foo(a, b, c)
{
    return a + b + c;
}
```

Bytecode

```
[    0] enter
[    1] get_scope          loc3
[    3] mov                loc4, loc3
[    6] check_traps
[    7] add                loc6, arg1, arg2
[   12] add                loc6, loc6, arg3
[   17] ret                loc6
```

DFG IR

```
24:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
25:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
26:  ArithAdd(Int32:@24, Int32:@25, CheckOverflow, Exits, bc#7)
27:  MovHint(Untyped:@26, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
29:  GetLocal(Untyped:@3, arg3(D<Int32>/FlushedInt32), R:Stack(8), bc#12)
30:  ArithAdd(Int32:@26, Int32:@29, CheckOverflow, Exits, bc#12)
31:  MovHint(Untyped:@30, loc6, W:SideState, ClobbersExit, bc#12, ExitInvalid)
33:  Return(Untyped:@3, W:SideState, Exits, bc#17)
```

DFG IR

```
24:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
25:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
26:  ArithAdd(Int32:@24, Int32:@25, CheckOverflow, Exits, bc#7)
27:  MovHint(Untyped:@26, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
29:  GetLocal(Untyped:@3, arg3(D<Int32>/FlushedInt32), R:Stack(8), bc#12)
30:  ArithAdd(Int32:@26, Int32:@29, CheckOverflow, Exits, bc#12)
31:  MovHint(Untyped:@30, loc6, W:SideState, ClobbersExit, bc#12, ExitInvalid)
33:  Return(Untyped:@3, W:SideState, Exits, bc#17)
```

B3 IR

```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
Int32 @46 = CheckAdd(@44:WarmAny, @45:WarmAny, @44:ColdAny, generator = 0x1052c5d70,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@30)
Int64 @47 = ZExt32(@46, DFG:@32)
Int64 @48 = Add(@47, $-281474976710656(@13), DFG:@32)
Void @49 = Return(@48, Terminal, DFG:@32)
```

B3 IR

```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
Int32 @46 = CheckAdd(@44:WarmAny, @45:WarmAny, @44:ColdAny, generator = 0x1052c5d70,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@30)
Int64 @47 = ZExt32(@46, DFG:@32)
Int64 @48 = Add(@47, $-281474976710656(@13), DFG:@32)
Void @49 = Return(@48, Terminal, DFG:@32)
```

B3 IR

```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
Int32 @46 = CheckAdd(@44:WarmAny, @45:WarmAny, @44:ColdAny, generator = 0x1052c5d70,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@30)
Int64 @47 = ZExt32(@46, DFG:@32)
Int64 @48 = Add(@47, $-281474976710656(@13), DFG:@32)
Void @49 = Return(@48, Terminal, DFG:@32)
```

```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
Int32 @46 = CheckAdd(@44:WarmAny, @45:WarmAny, @44:ColdAny, generator = 0x1052c5d70,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@30)
Int64 @47 = ZExt32(@46, DFG:@32)
Int64 @48 = Add(@47, $-281474976710656(@13), DFG:@32)
Void @49 = Return(@48, Terminal, DFG:@32)
```

```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
Int32 @46 = CheckAdd(@44:WarmAny, @45:WarmAny, @44:ColdAny, generator = 0x1052c5d70,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@30)
Int64 @47 = ZExt32(@46, DFG:@32)
Int64 @48 = Add(@47, $-281474976710656(@13), DFG:@32)
Void @49 = Return(@48, Terminal, DFG:@32)
```

```
26:  ArithAdd(Int32:@24, Int32:@25, CheckOverflow, Exits, bc#7)
27:  MovHint(Untyped:@26, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
30:  ArithAdd(Int32:@26, Int32:@29, CheckOverflow, Exits, bc#12)
```

```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
Int32 @46 = CheckAdd(@44:WarmAny, @45:WarmAny, @44:ColdAny, generator = 0x1052c5d70,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@30)
Int64 @47 = ZExt32(@46, DFG:@32)
Int64 @48 = Add(@47, $-281474976710656(@13), DFG:@32)
Void @49 = Return(@48, Terminal, DFG:@32)
```

```
26:  ArithAdd(Int32:@24, Int32:@25, CheckOverflow, Exits, bc#7)
27:  MovHint(Untyped:@26, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
30:  ArithAdd(Int32:@26, Int32:@29, CheckOverflow, Exits, bc#12)
```

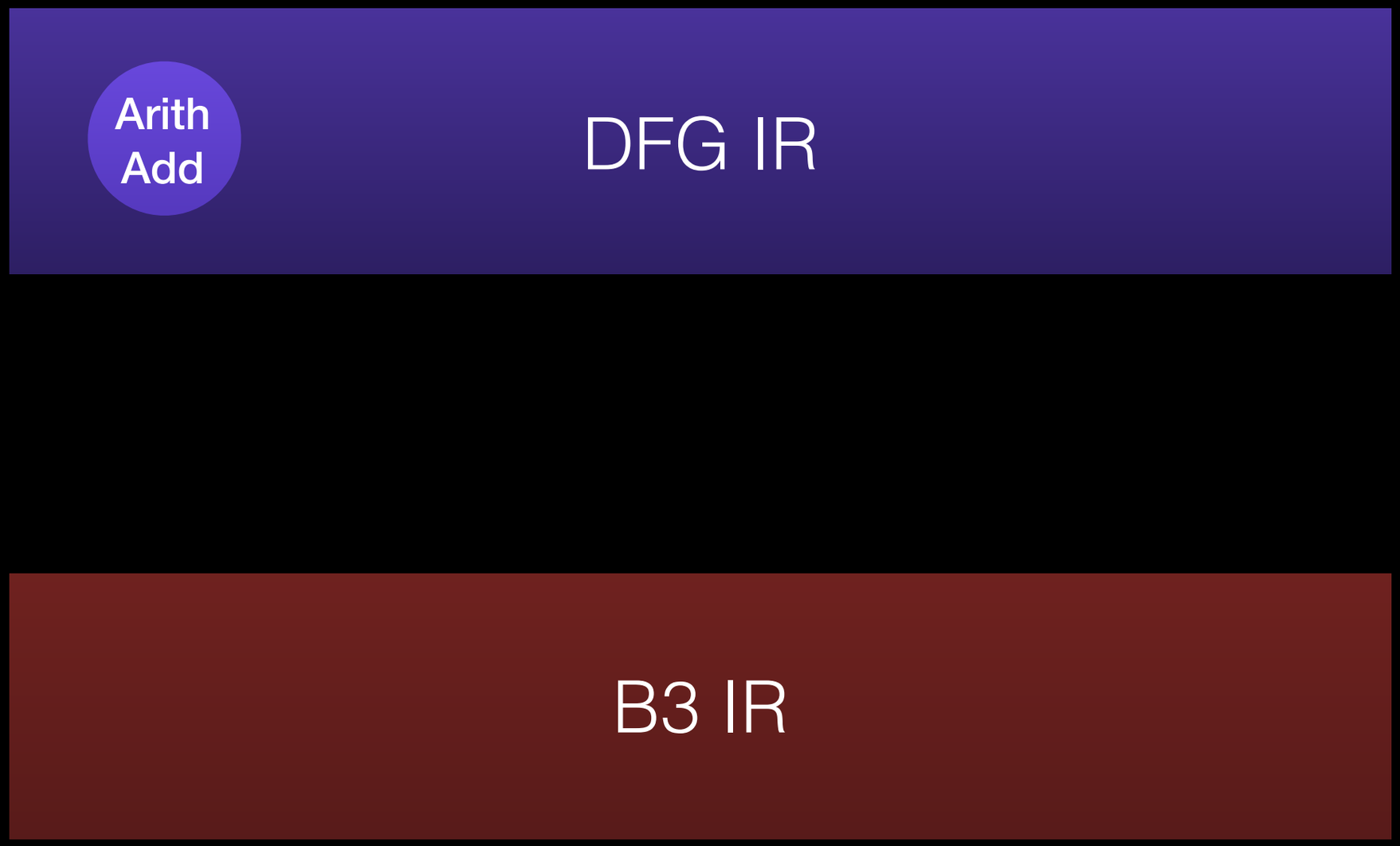
```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
Int32 @46 = CheckAdd(@44:WarmAny, @45:WarmAny, @44:ColdAny, generator = 0x1052c5d70,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@30)
Int64 @47 = ZExt32(@46, DFG:@32)
Int64 @48 = Add(@47, $-281474976710656(@13), DFG:@32)
Void @49 = Return(@48, Terminal, DFG:@32)
```

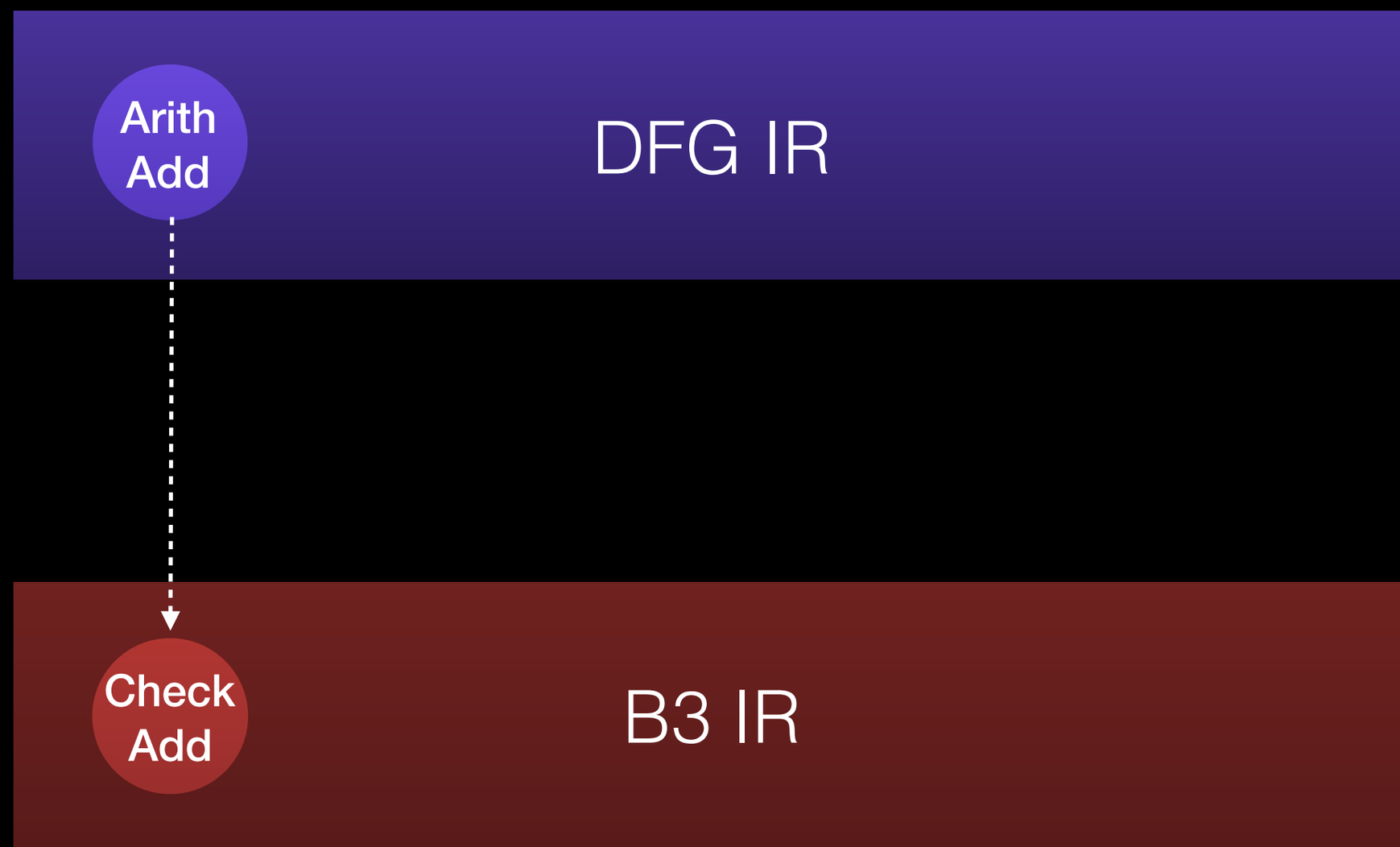
```
26:  ArithAdd(Int32:@24, Int32:@25, CheckOverflow, Exits, bc#7)
27:  MovHint(Untyped:@26, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
30:  ArithAdd(Int32:@26, Int32:@29, CheckOverflow, Exits, bc#12)
```

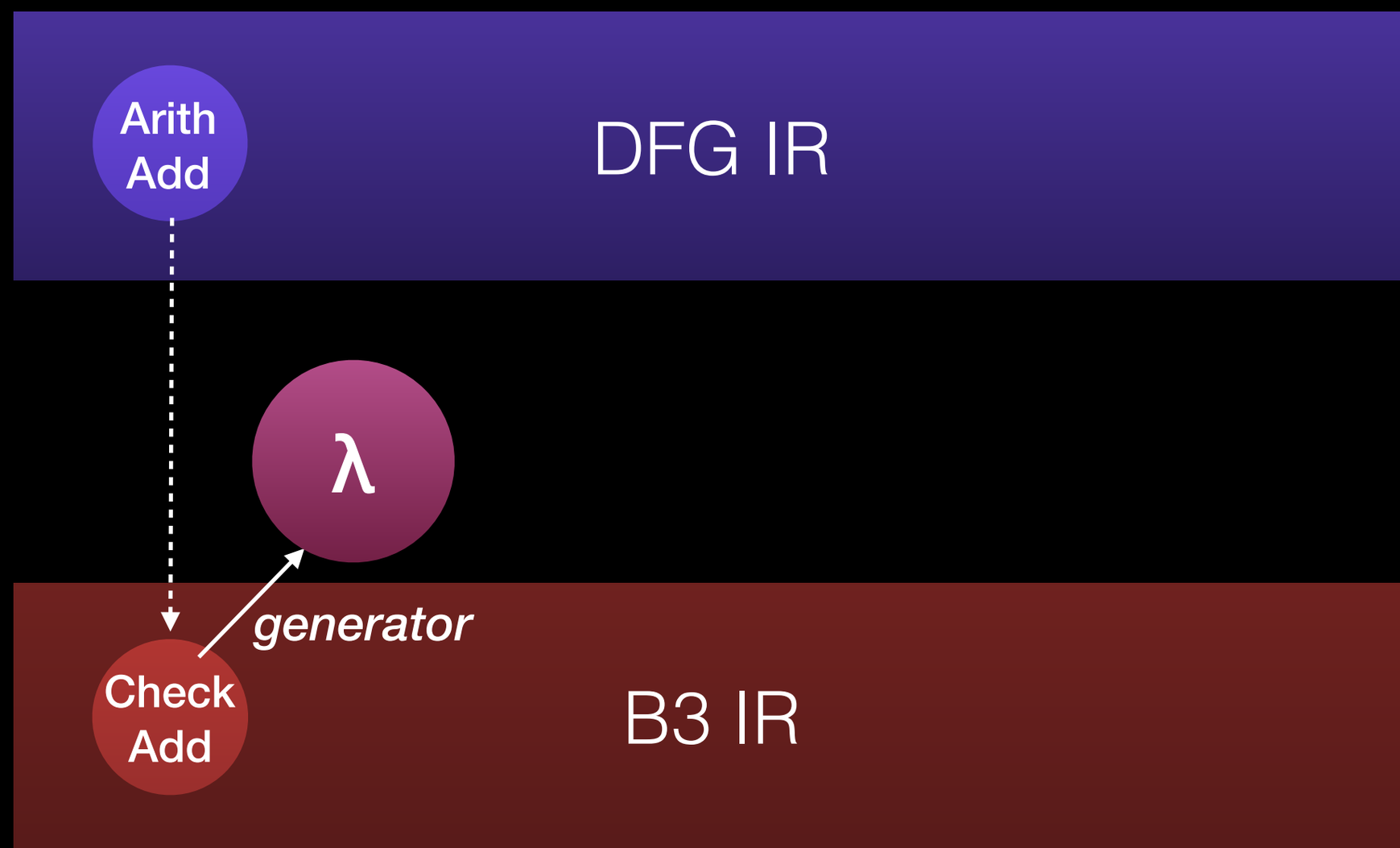
```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
Int32 @46 = CheckAdd(@44:WarmAny, @45:WarmAny, @44:ColdAny, generator = 0x1052c5d70,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@30)
Int64 @47 = ZExt32(@46, DFG:@32)
Int64 @48 = Add(@47, $-281474976710656(@13), DFG:@32)
Void @49 = Return(@48, Terminal, DFG:@32)
```

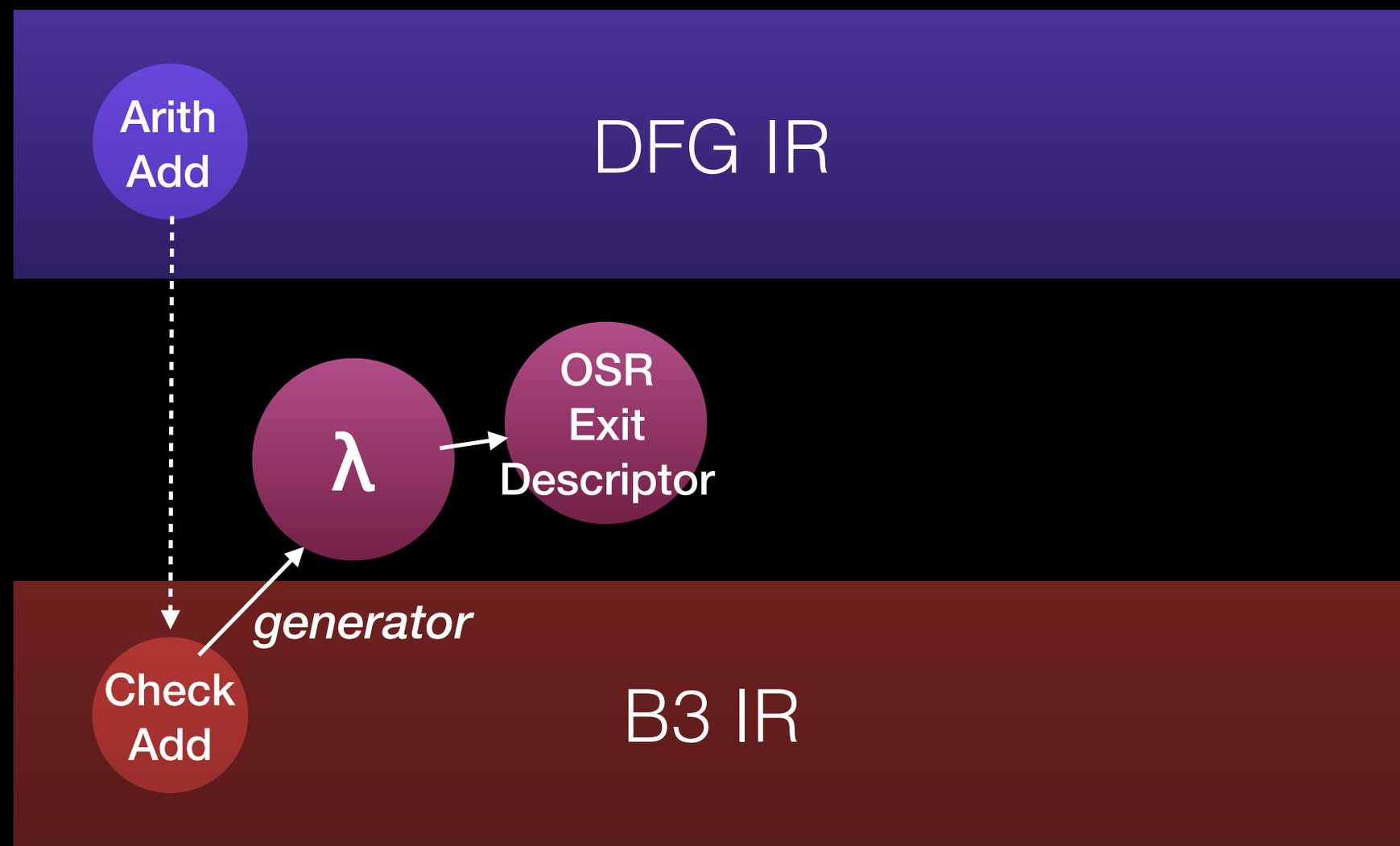
DFG IR

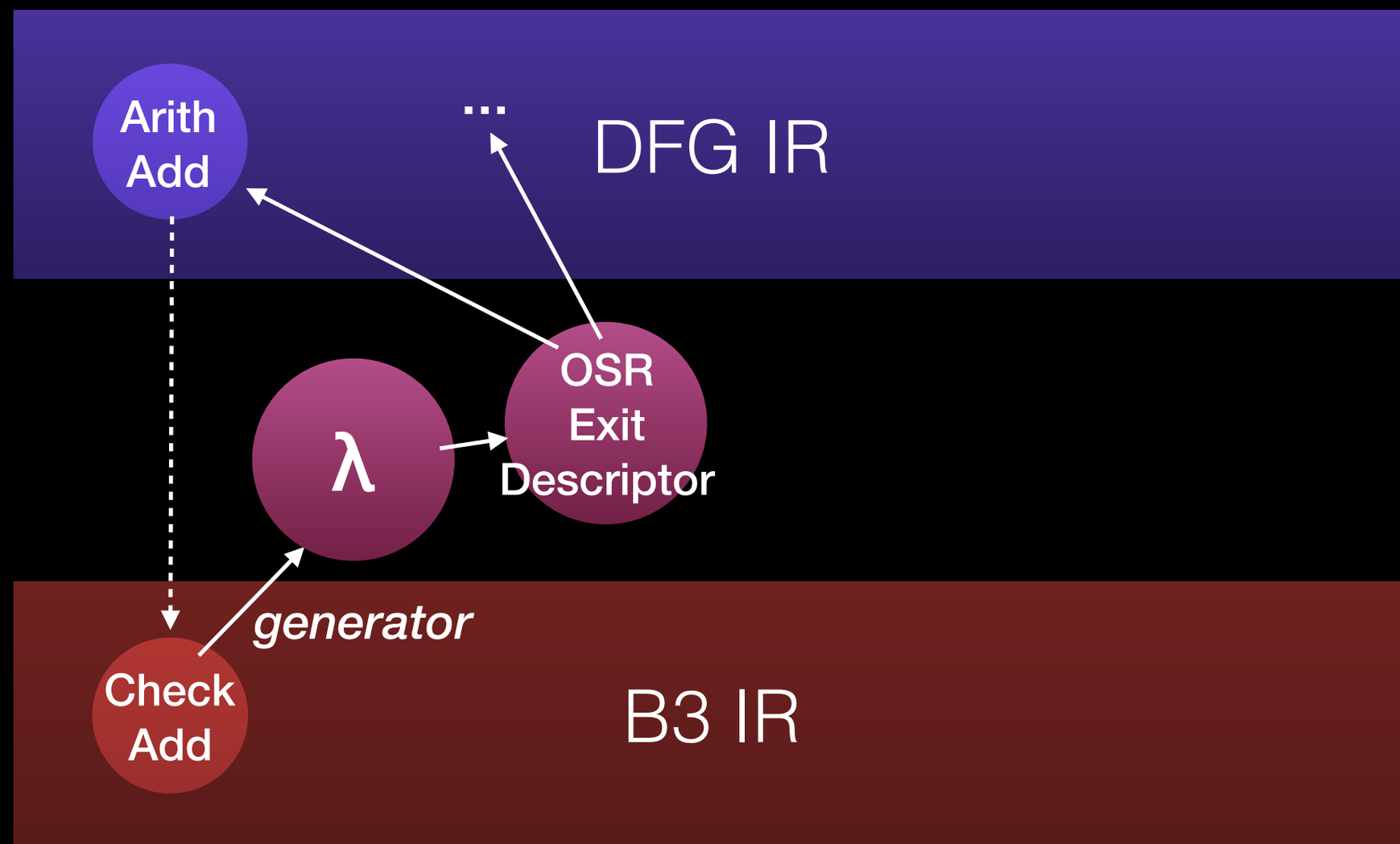
B3 IR


Arith
Add
DFG IR
B3 IR


Arith
Add
DFG IR
Check
Add
B3 IR


Arith
Add
DFG IR
λ
generator
Check
Add
B3 IR


Arith
Add
DFG IR
λ
OSR
Exit
Descriptor
generator
Check
Add
B3 IR


Arith
Add
...
DFG IR
OSR
Exit
Descriptor
λ
generator
Check
Add
B3 IR

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
         generator = 0x…)
```

Air backend

```
Patch &BranchAdd32 Overflow, %left, %right, %dst,
      %arg0, %arg1, %arg2, …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
         generator = 0x…)
```

Air backend

```
Patch &BranchAdd32 Overflow, %left, %right, %dst,
      %arg0, %arg1, %arg2, …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
        generator = 0x…)
```

Air backend

```
Patch &BranchAdd32 Overflow, %left, %right, %dst,
      %arg0, %arg1, %arg2, …,
        generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
        generator = 0x…)
```

Air backend

```
Patch &BranchAdd32 Overflow, %left, %right, %dst,
      %arg0, %arg1, %arg2, …,
        generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
        generator = 0x…)
```

Air backend

```
Patch &BranchAdd32 Overflow, %left, %right, %dst,
      %rcx , %r11 , %rax , …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

%rcx

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
        generator = 0x…)
```

Air backend

```
Patch &BranchAdd32 Overflow, %left, %right, %dst,
      %rcx , %r11 , %rax , …,
        generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

%rcx %r11

```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
        generator = 0x…)
```

Air backend

```
Patch &BranchAdd32 Overflow, %left, %right, %dst,
      %rcx , %r11 , %rax , …,
        generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

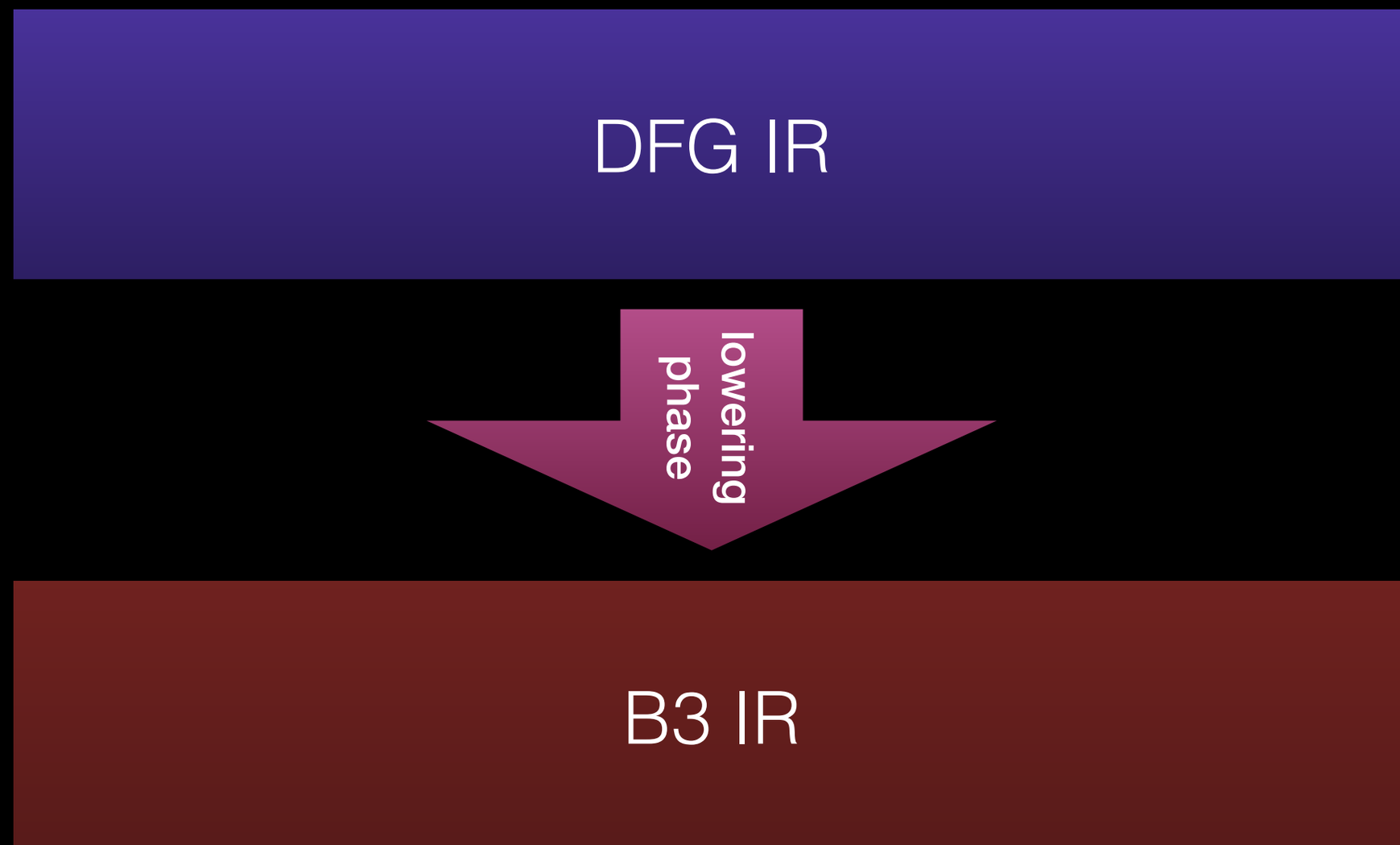
%rax %rcx %r11

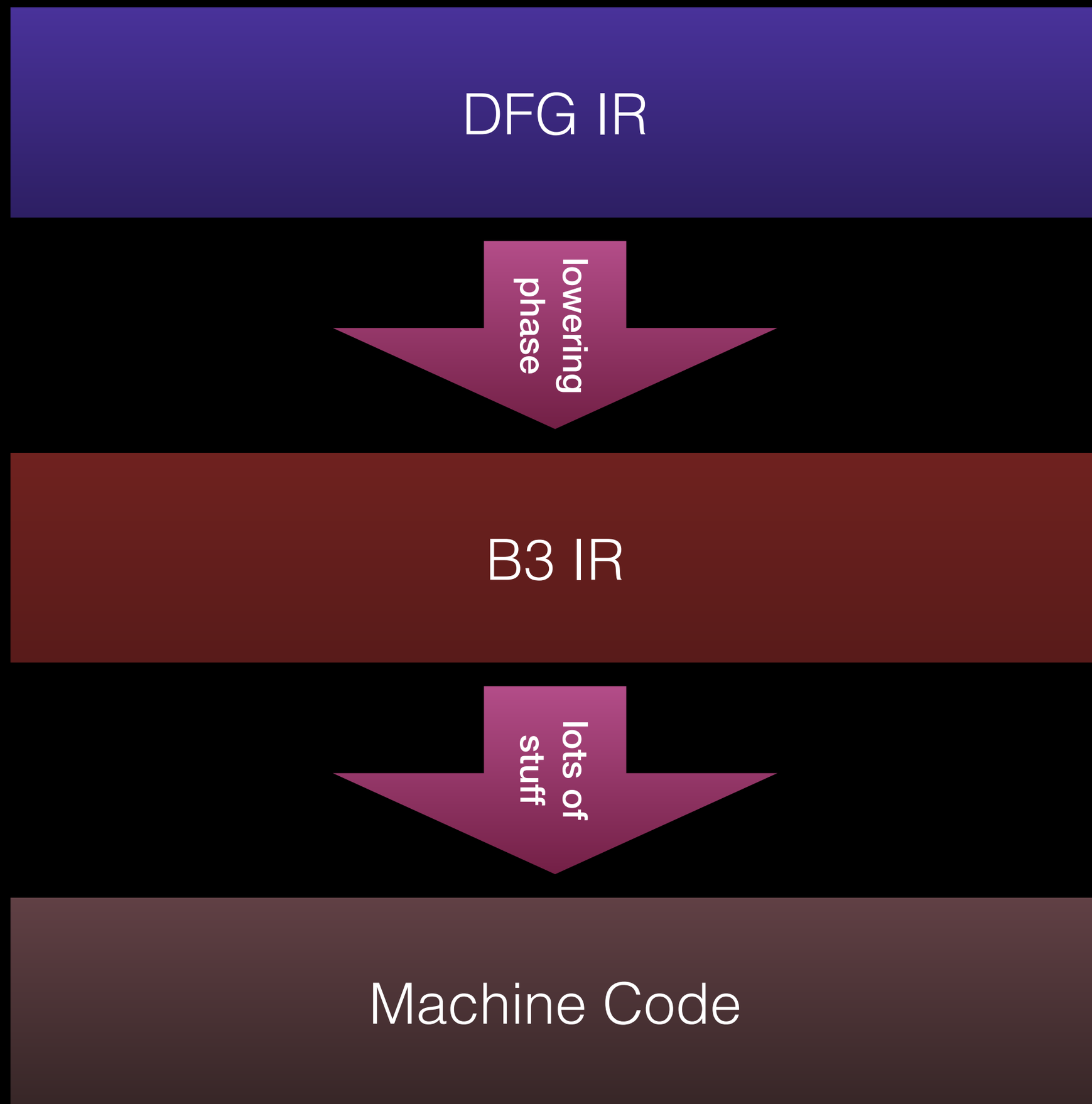
```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
        generator = 0x…)
```

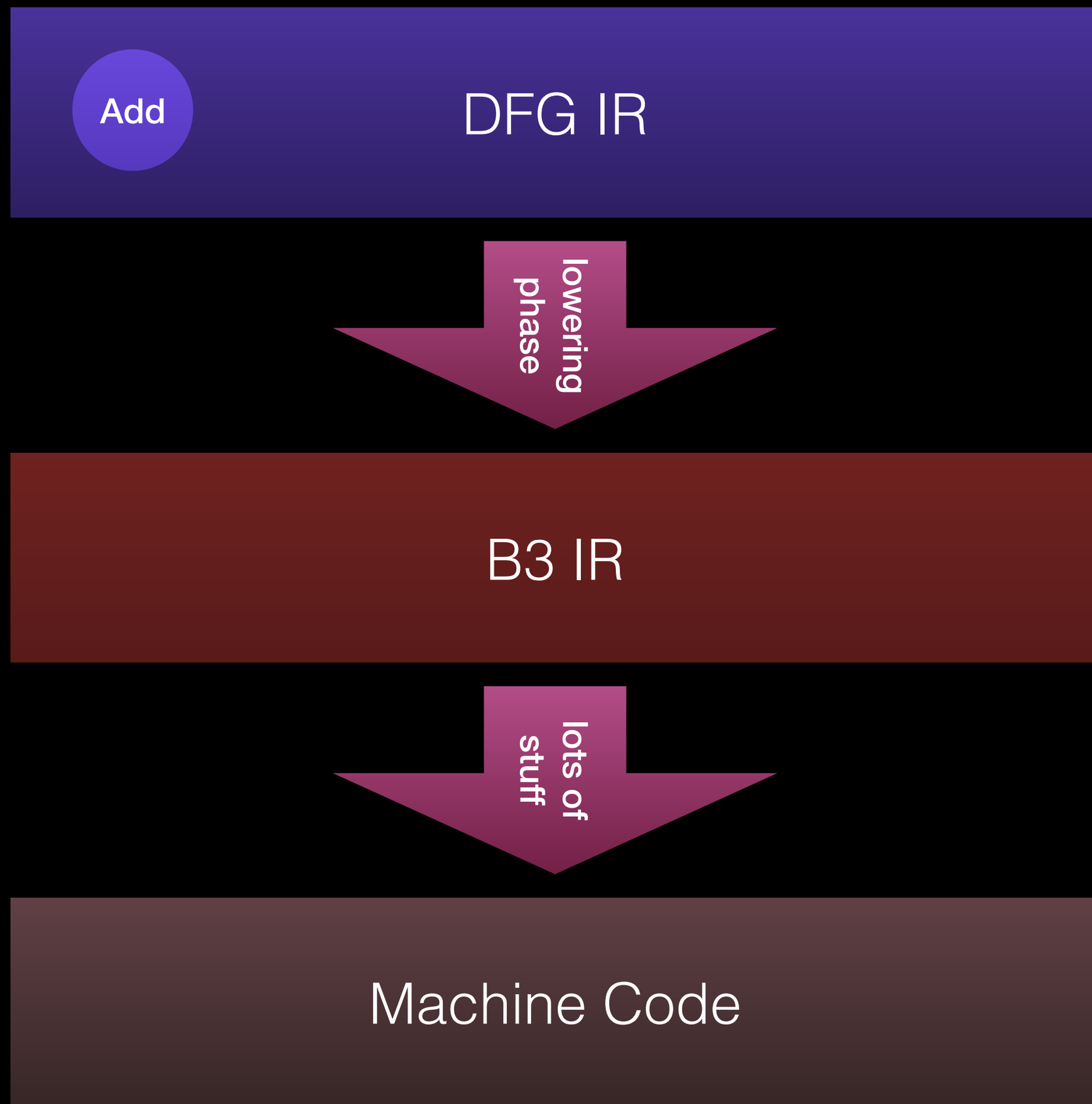
Air backend

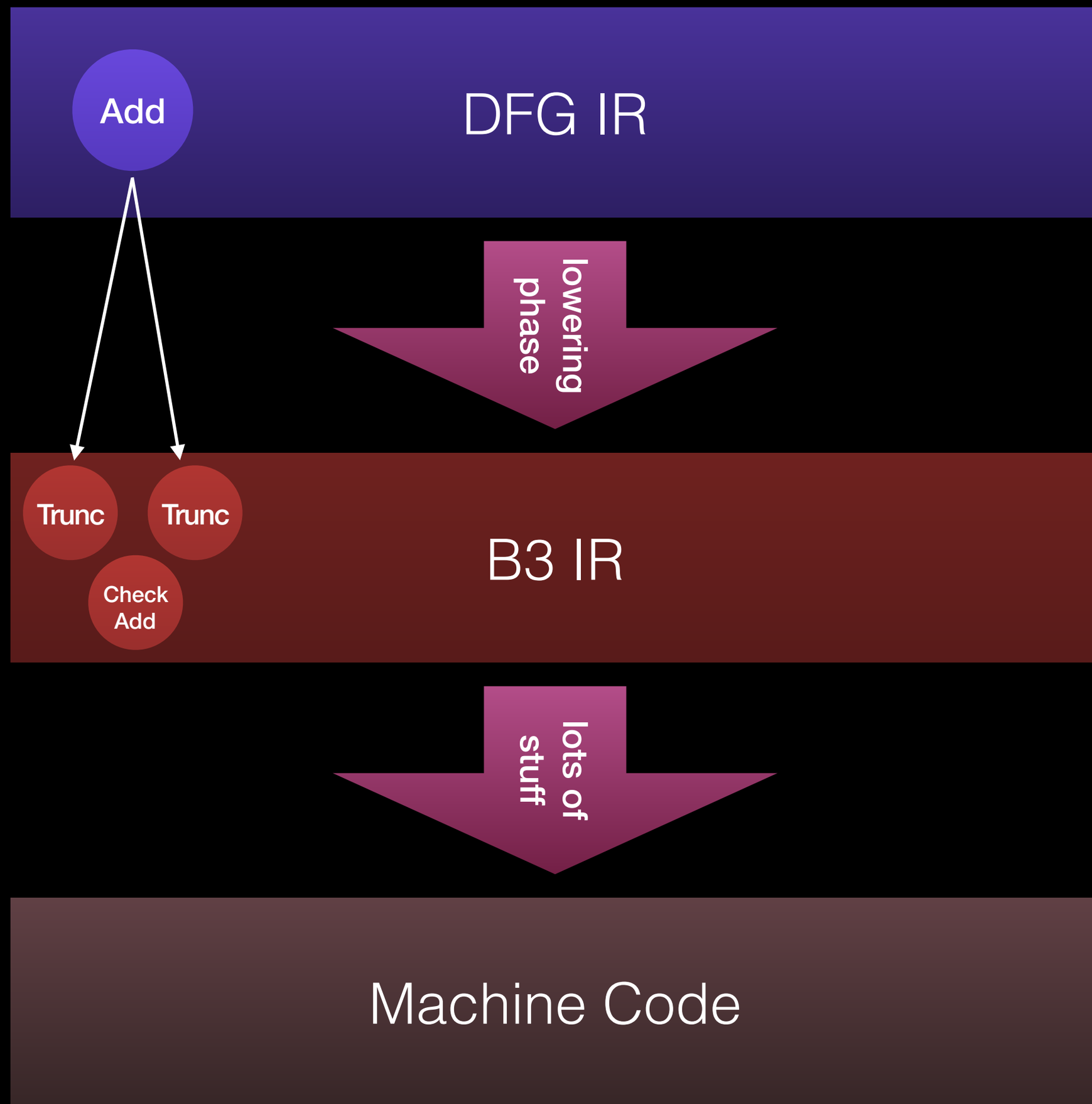
```
Patch &BranchAdd32 Overflow, %left, %right, %dst,
      %rcx , %r11 , %rax , …,
        generator = 0x…)
```

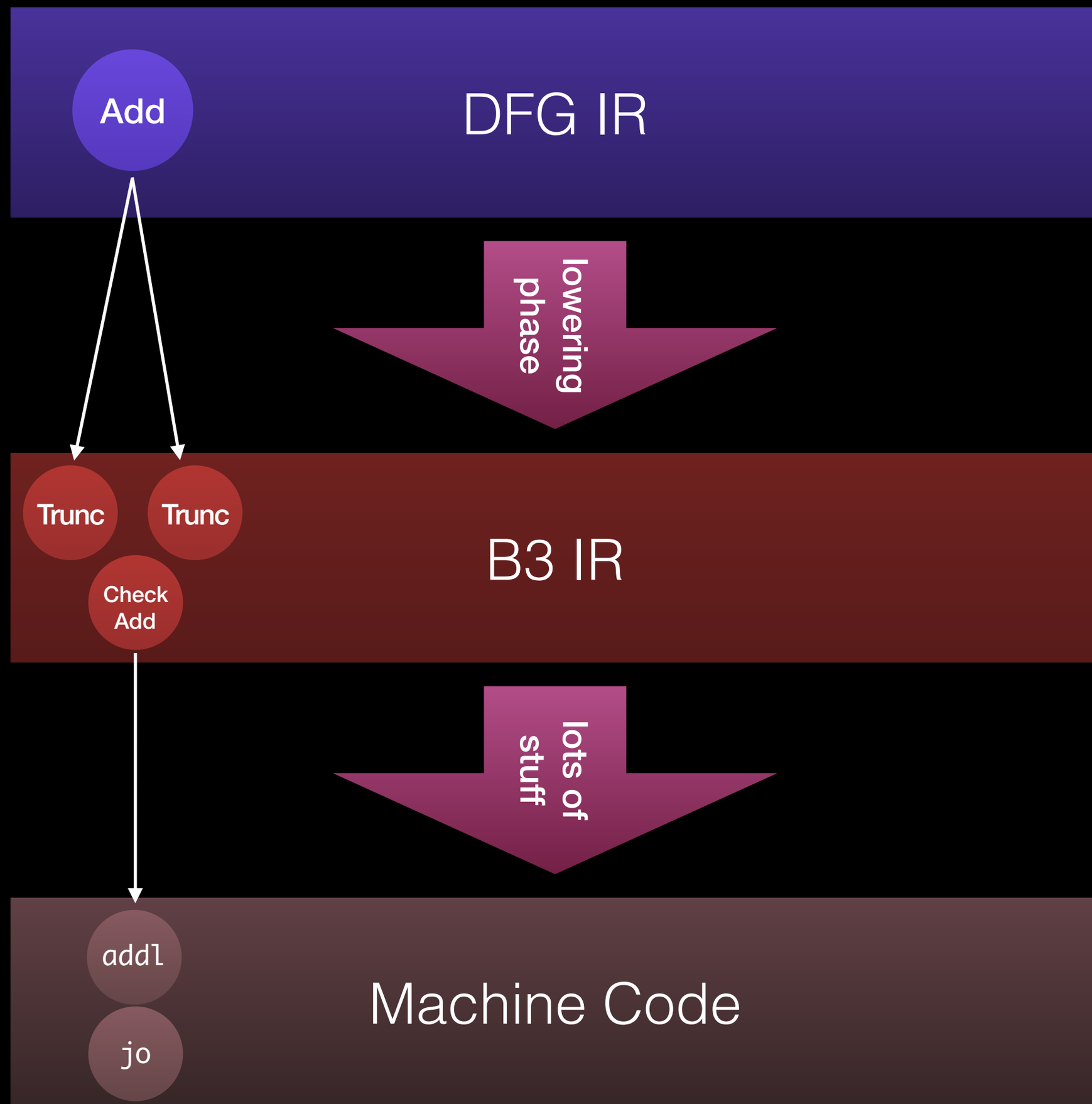
DFG IR

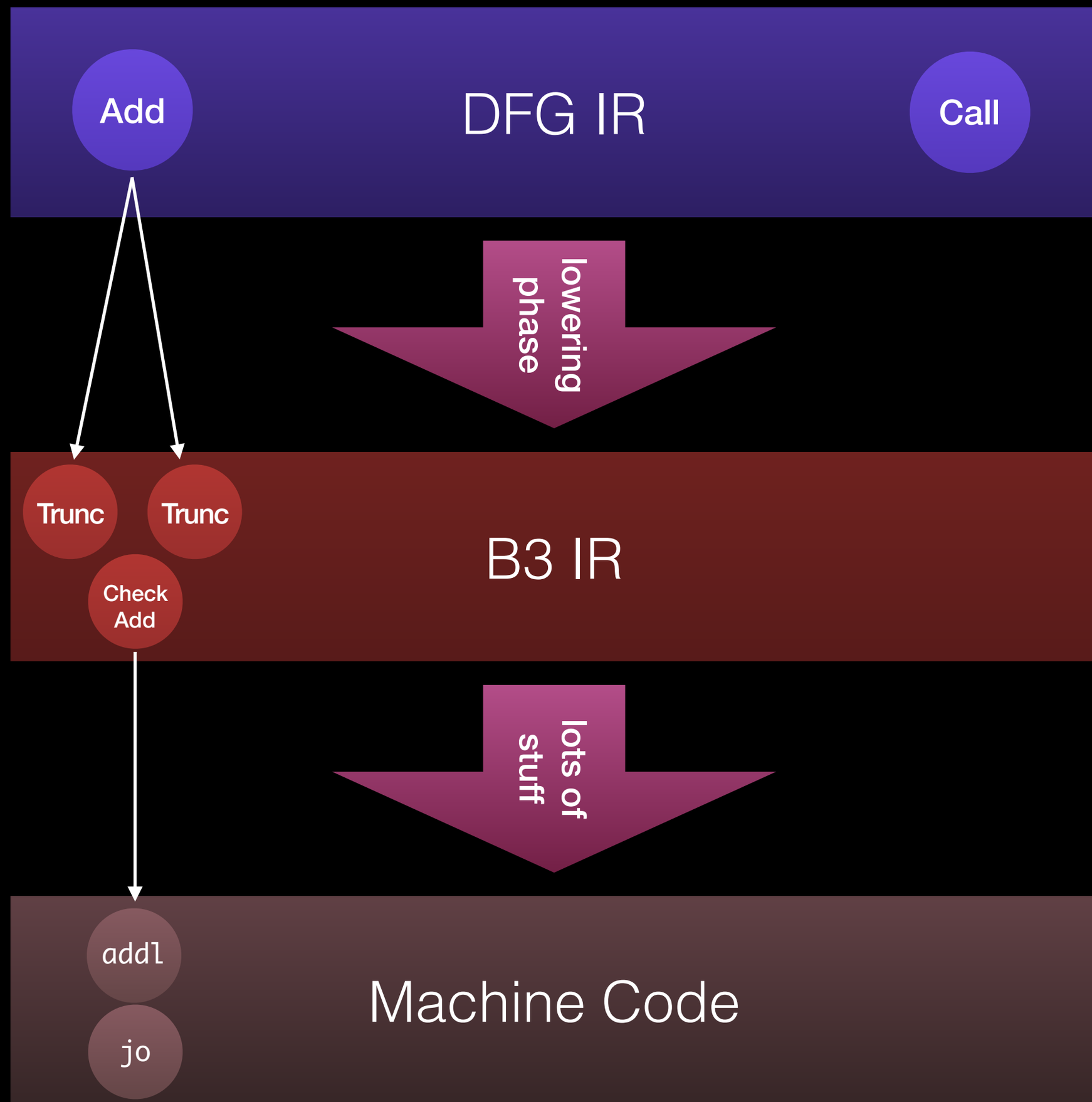

DFG IR
lowering phase
B3 IR

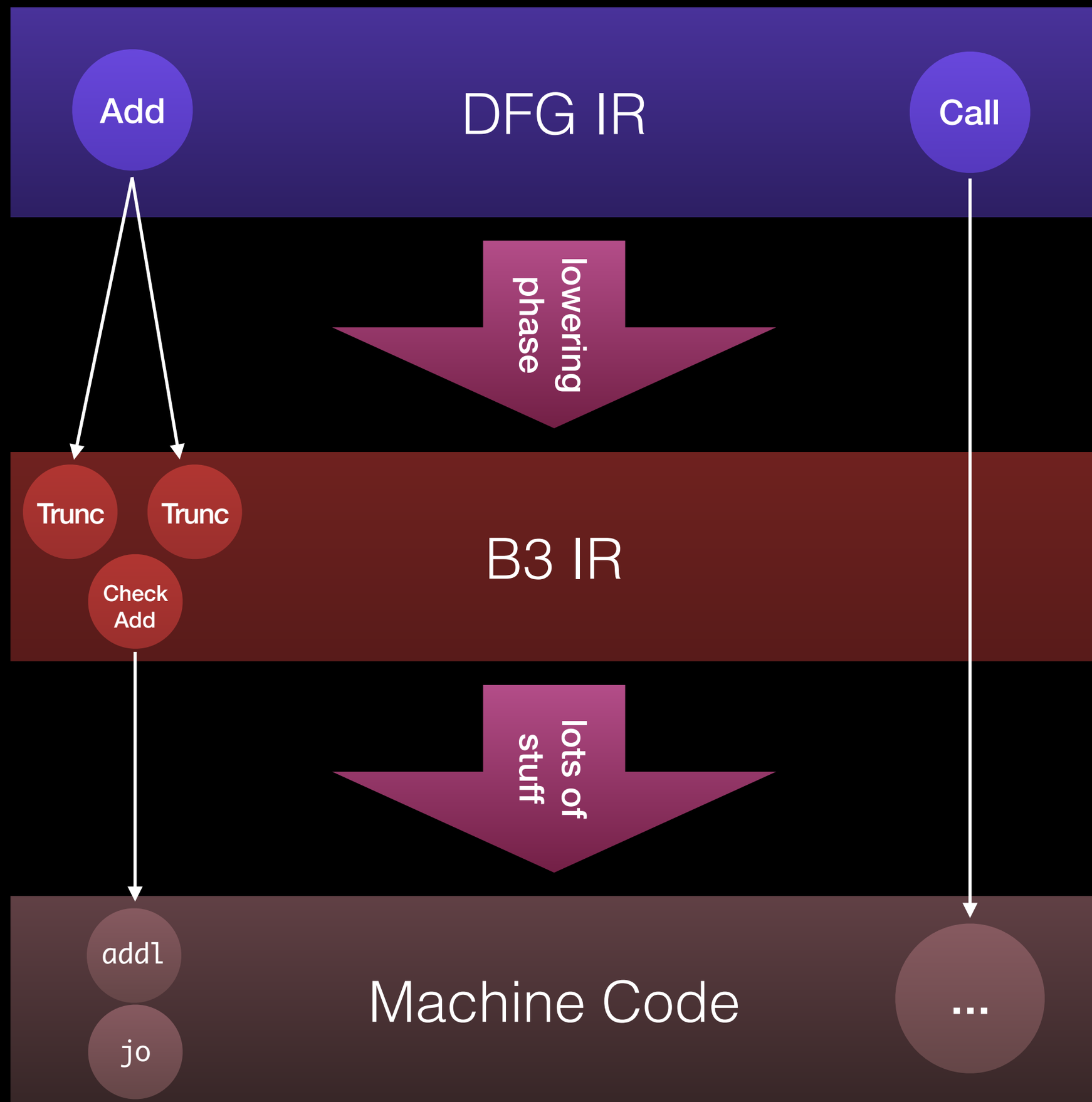

DFG IR
lowering phase
B3 IR
lots of stuff
Machine Code

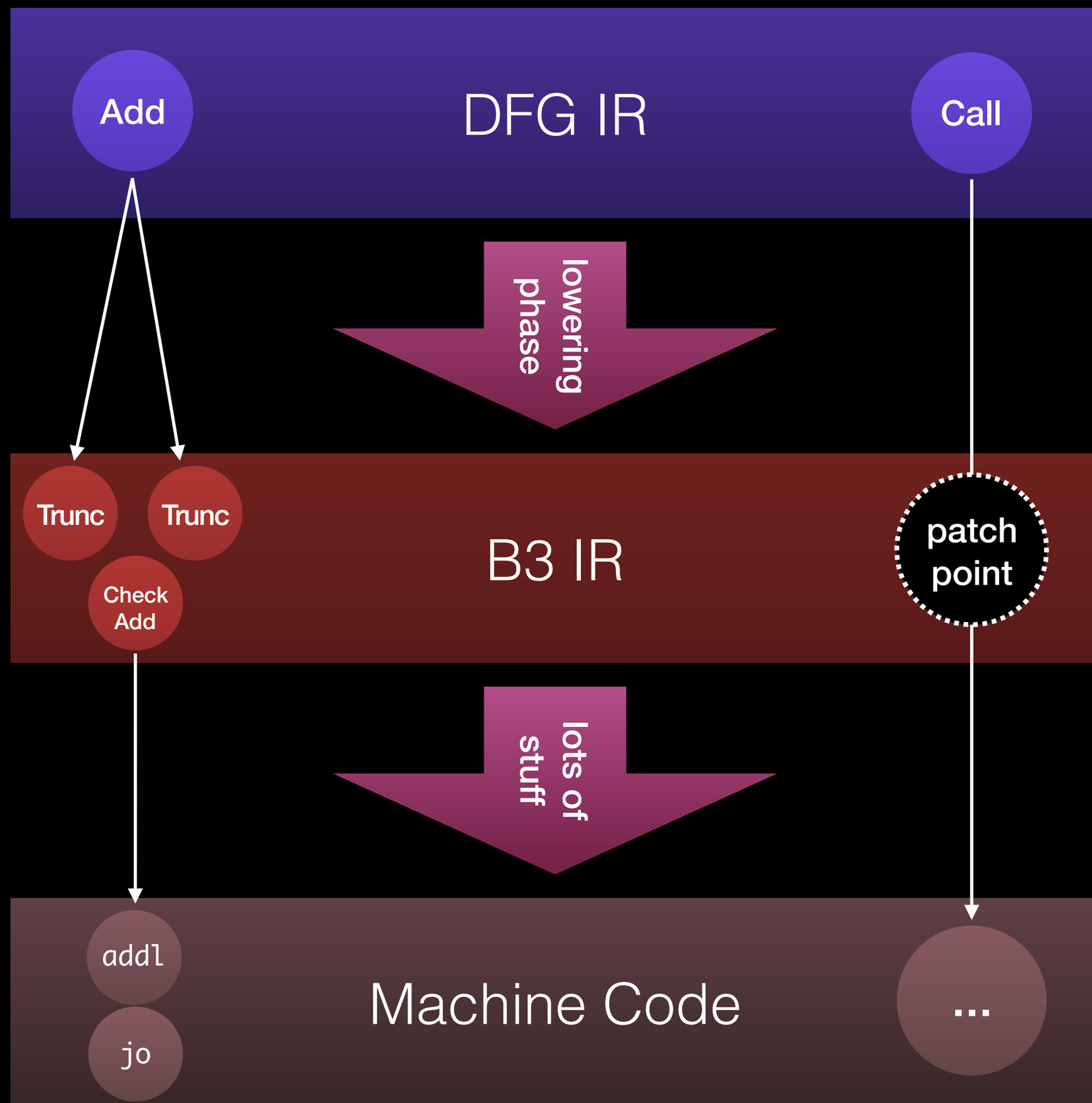

Add
DFG IR
lowering phase
B3 IR
lots of stuff
Machine Code

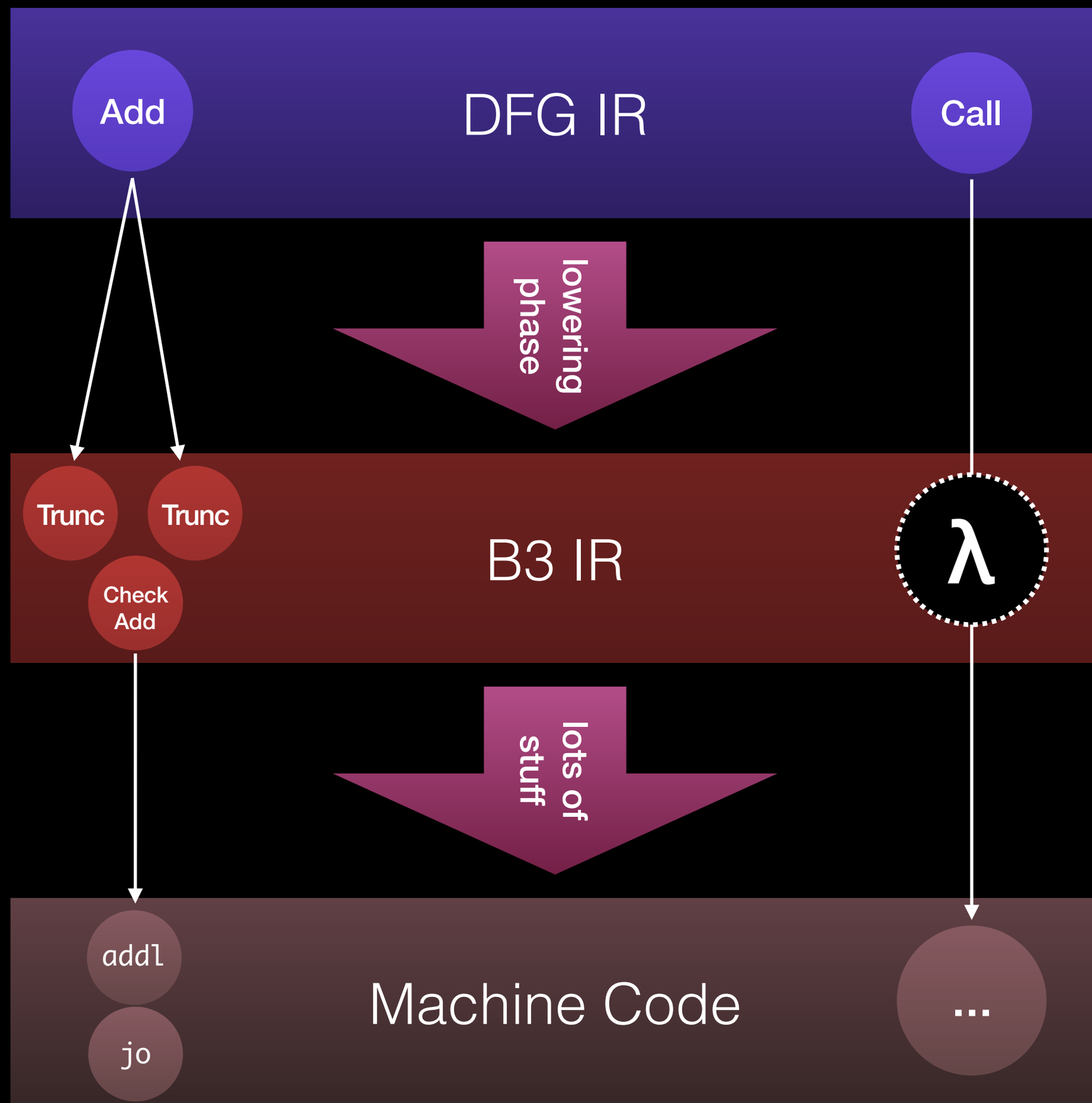

Add
DFG IR
lowering phase
Trunc
Trunc
Check Add
B3 IR
lots of stuff
Machine Code


Add
DFG IR
lowering phase
Trunc
Trunc
Check Add
B3 IR
lots of stuff
addl
jo
Machine Code


Add
DFG IR
Call
lowering phase
Trunc
Trunc
Check Add
B3 IR
lots of stuff
addl
jo
Machine Code


DFG IR
Add
Call
lowering phase
B3 IR
Trunc
Trunc
Check Add
lots of stuff
Machine Code
addl
jo
...


DFG IR
Add
Call
lowering phase
B3 IR
Trunc
Trunc
Check Add
patch point
lots of stuff
Machine Code
addl
jo
...


DFG IR
Add
Call
lowering phase
B3 IR
Trunc
Trunc
Check Add
λ
lots of stuff
Machine Code
addl
jo
...

```
inline void x86_cpuid()
{
    intptr_t a = 0, b, c, d;
    asm volatile(
        "cpuid"
        : "+a"(a), "=b"(b), "=c"(c), "=d"(d)
        :
        : "memory");
}
```

```
if (MacroAssemblerARM64::
    supportsDoubleToInt32ConversionUsingJavaScriptSemantics()) {
    PatchpointValue* patchpoint = m_out.patchpoint(Int32);
    patchpoint->appendSomeRegister(doubleValue);
    patchpoint->setGenerator(
        [=] (CCallHelpers& jit,
             const StackmapGenerationParams& params) {
            jit.convertDoubleToInt32UsingJavaScriptSemantics(
                params[1].fpr(), params[0].gpr());
        });
    patchpoint->effects = Effects::none();
    return patchpoint;
}
```

```
if (MacroAssemblerARM64::
    supportsDoubleToInt32ConversionUsingJavaScriptSemantics()) {
    PatchpointValue* patchpoint = m_out.patchpoint(Int32);
    patchpoint->appendSomeRegister(doubleValue);
    patchpoint->setGenerator(
        [=] (CCallHelpers& jit,
             const StackmapGenerationParams& params) {
            jit.convertDoubleToInt32UsingJavaScriptSemantics(
                params[1].fpr(), params[0].gpr());
        });
    patchpoint->effects = Effects::none();
    return patchpoint;
}
```

```
if (MacroAssemblerARM64::
    supportsDoubleToInt32ConversionUsingJavaScriptSemantics()) {
    PatchpointValue* patchpoint = m_out.patchpoint(Int32);
    patchpoint->appendSomeRegister(doubleValue);
    patchpoint->setGenerator(
        [=] (CCallHelpers& jit,
             const StackmapGenerationParams& params) {
            jit.convertDoubleToInt32UsingJavaScriptSemantics(
                params[1].fpr(), params[0].gpr());
        });
    patchpoint->effects = Effects::none();
    return patchpoint;
}
```

```
if (MacroAssemblerARM64::
    supportsDoubleToInt32ConversionUsingJavaScriptSemantics()) {
    PatchpointValue* patchpoint = m_out.patchpoint(Int32);
    patchpoint->appendSomeRegister(doubleValue);
    patchpoint->setGenerator(
        [=] (CCallHelpers& jit,
             const StackmapGenerationParams& params) {
            jit.convertDoubleToInt32UsingJavaScriptSemantics(
                params[1].fpr(), params[0].gpr());
        });
    patchpoint->effects = Effects::none();
    return patchpoint;
}
```

```
if (MacroAssemblerARM64::
    supportsDoubleToInt32ConversionUsingJavaScriptSemantics()) {
    PatchpointValue* patchpoint = m_out.patchpoint(Int32);
    patchpoint->appendSomeRegister(doubleValue);
    patchpoint->setGenerator(
        [=] (CCallHelpers& jit,
             const StackmapGenerationParams& params) {
            jit.convertDoubleToInt32UsingJavaScriptSemantics(
                params[1].fpr(), params[0].gpr());
        });
    patchpoint->effects = Effects::none();
    return patchpoint;
}
```

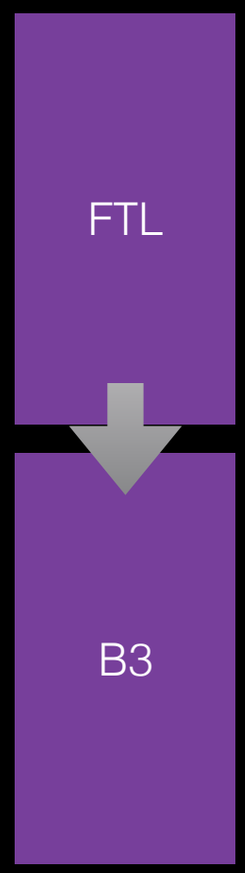
```
if (MacroAssemblerARM64::
    supportsDoubleToInt32ConversionUsingJavaScriptSemantics()) {
    PatchpointValue* patchpoint = m_out.patchpoint(Int32);
    patchpoint->appendSomeRegister(doubleValue);
    patchpoint->setGenerator(
        [=] (CCallHelpers& jit,
             const StackmapGenerationParams& params) {
            jit.convertDoubleToInt32UsingJavaScriptSemantics(
                params[1].fpr(), params[0].gpr());
        });
    patchpoint->effects = Effects::none();
    return patchpoint;
}
```

```
if (MacroAssemblerARM64::
    supportsDoubleToInt32ConversionUsingJavaScriptSemantics()) {
    PatchpointValue* patchpoint = m_out.patchpoint(Int32);
    patchpoint->appendSomeRegister(doubleValue);
    patchpoint->setGenerator(
        [=] (CCallHelpers& jit,
             const StackmapGenerationParams& params) {
            jit.convertDoubleToInt32UsingJavaScriptSemantics(
                params[1].fpr(), params[0].gpr());
        });
    patchpoint->effects = Effects::none();
    return patchpoint;
}
```

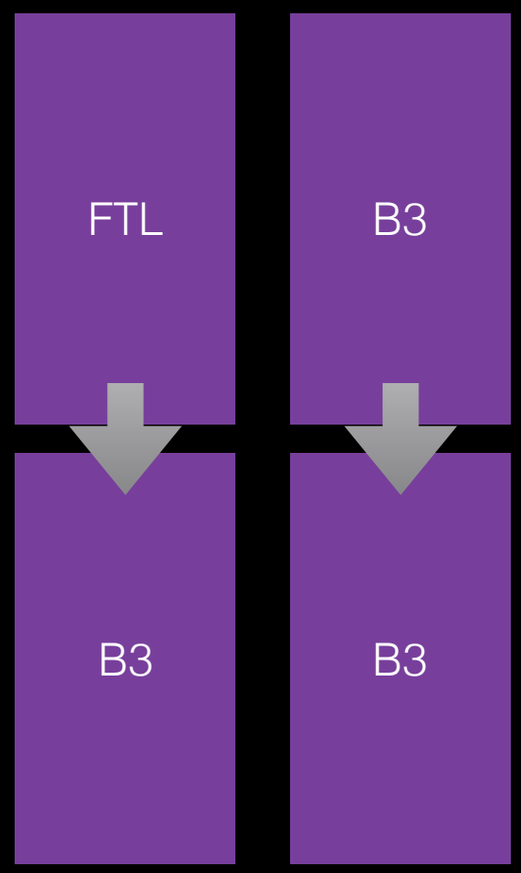
Patchpoint Use Cases

- Polymorphic inline caches
- Calls with interesting calling conventions
- Lazy slow paths
- Interesting instructions

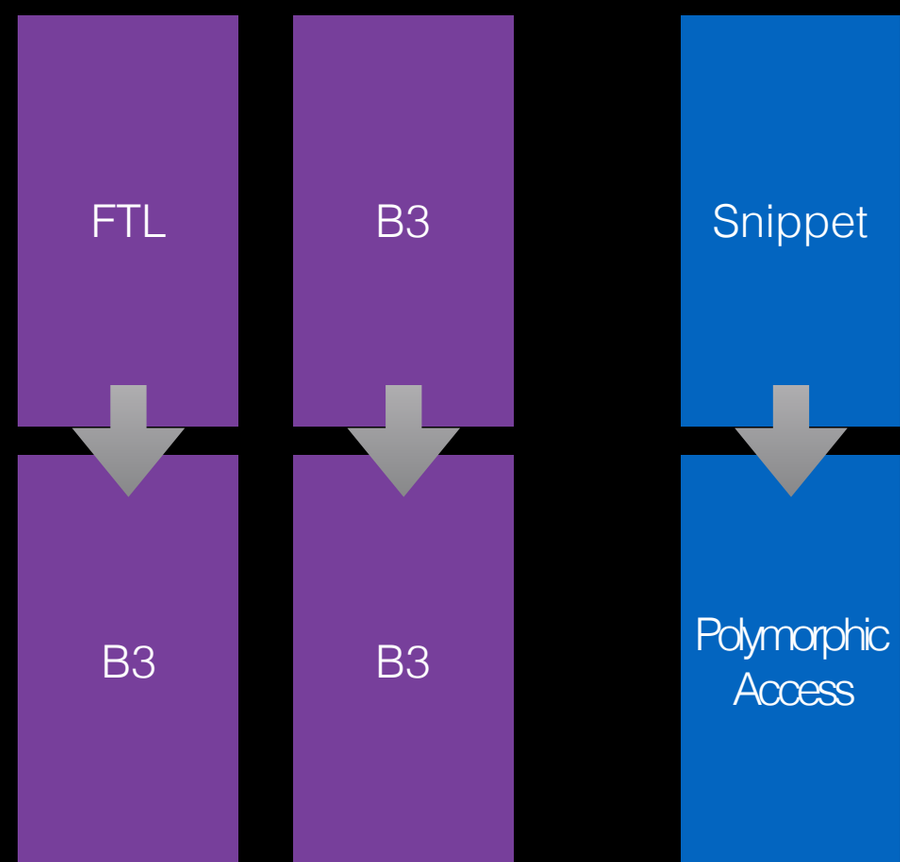
Patchpoint Use Cases



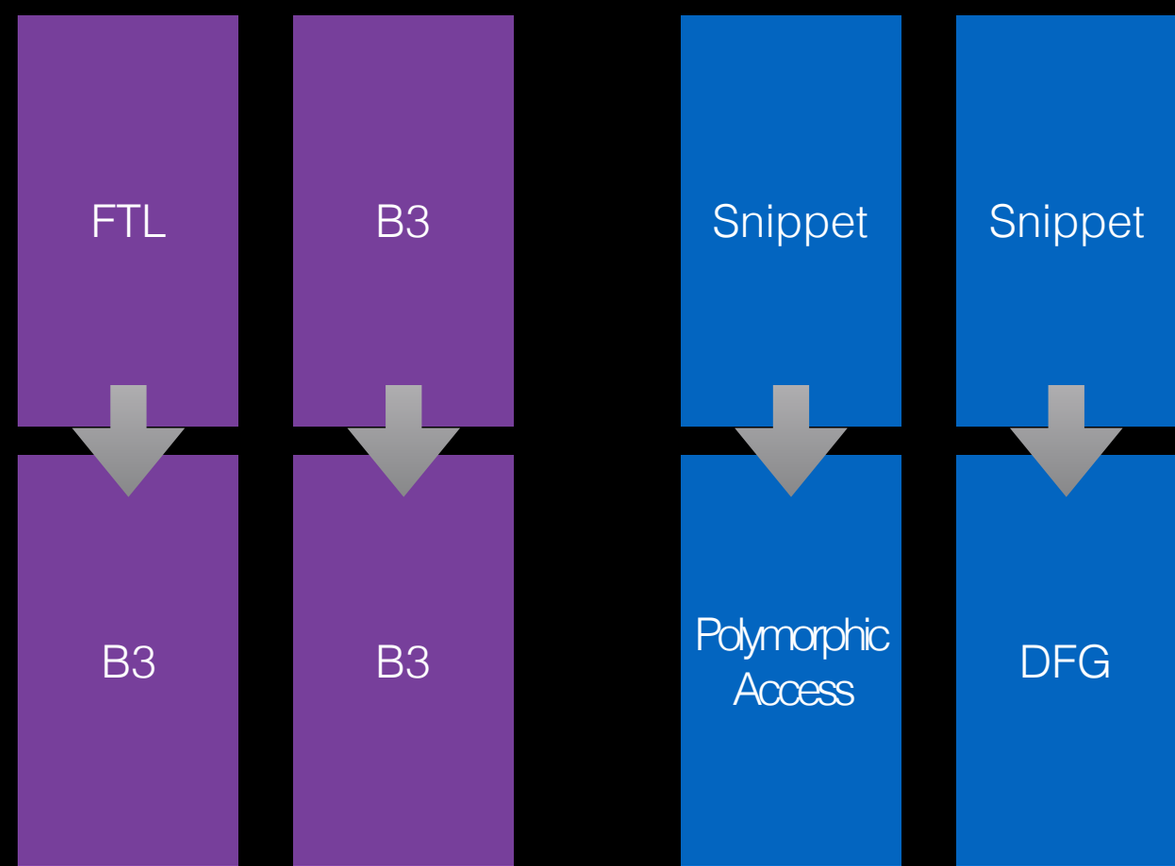
Patchpoint Use Cases



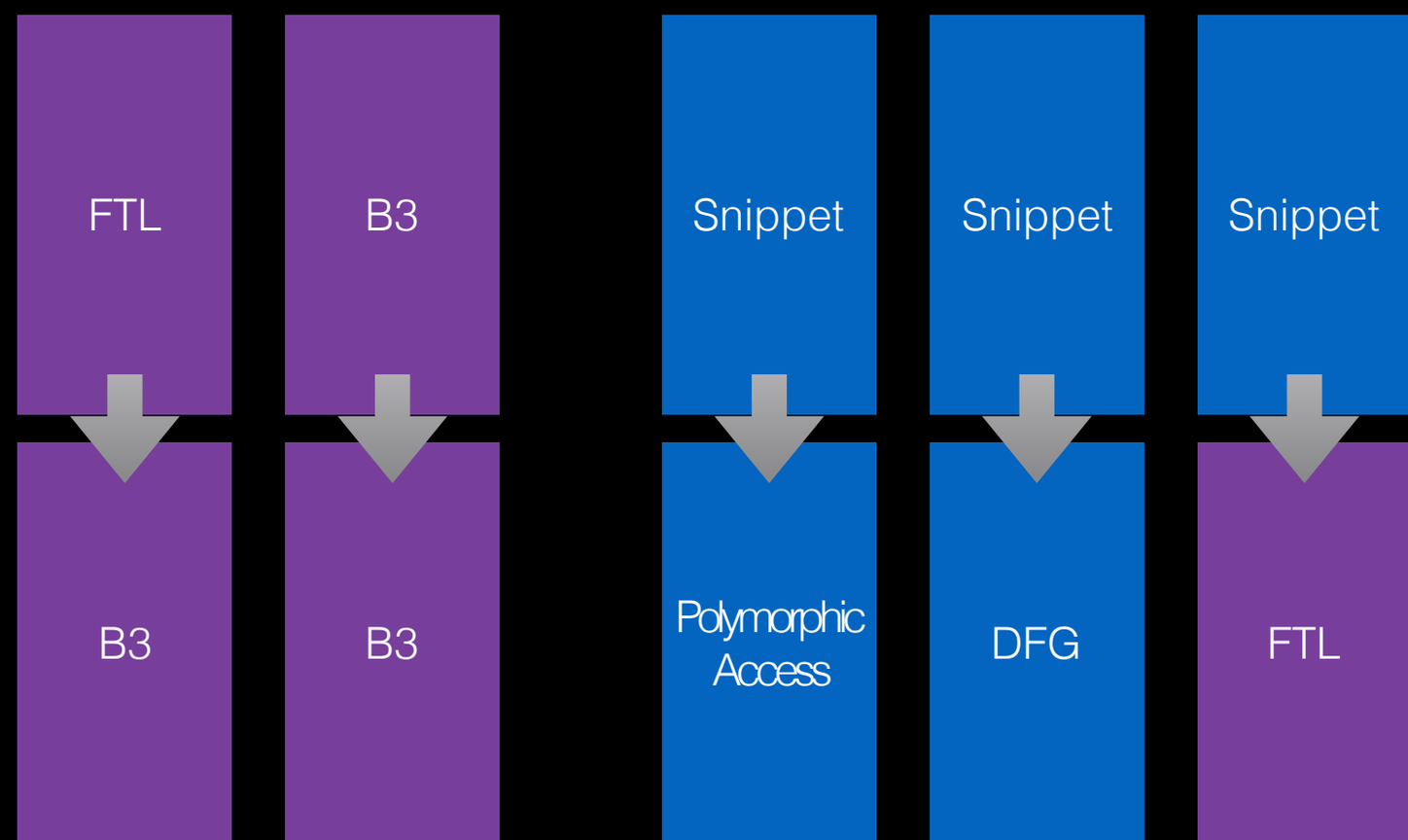
Patchpoint Use Cases

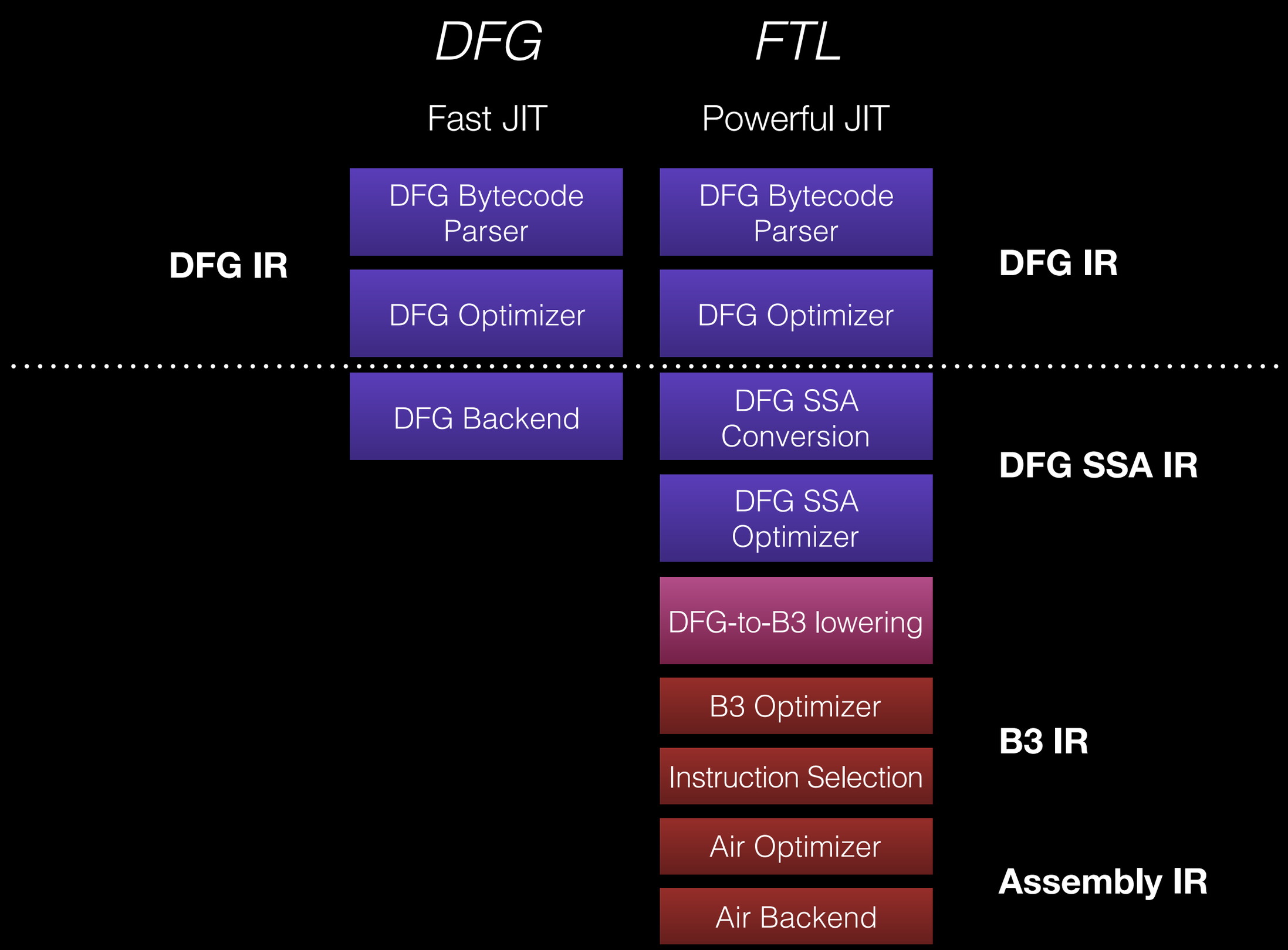


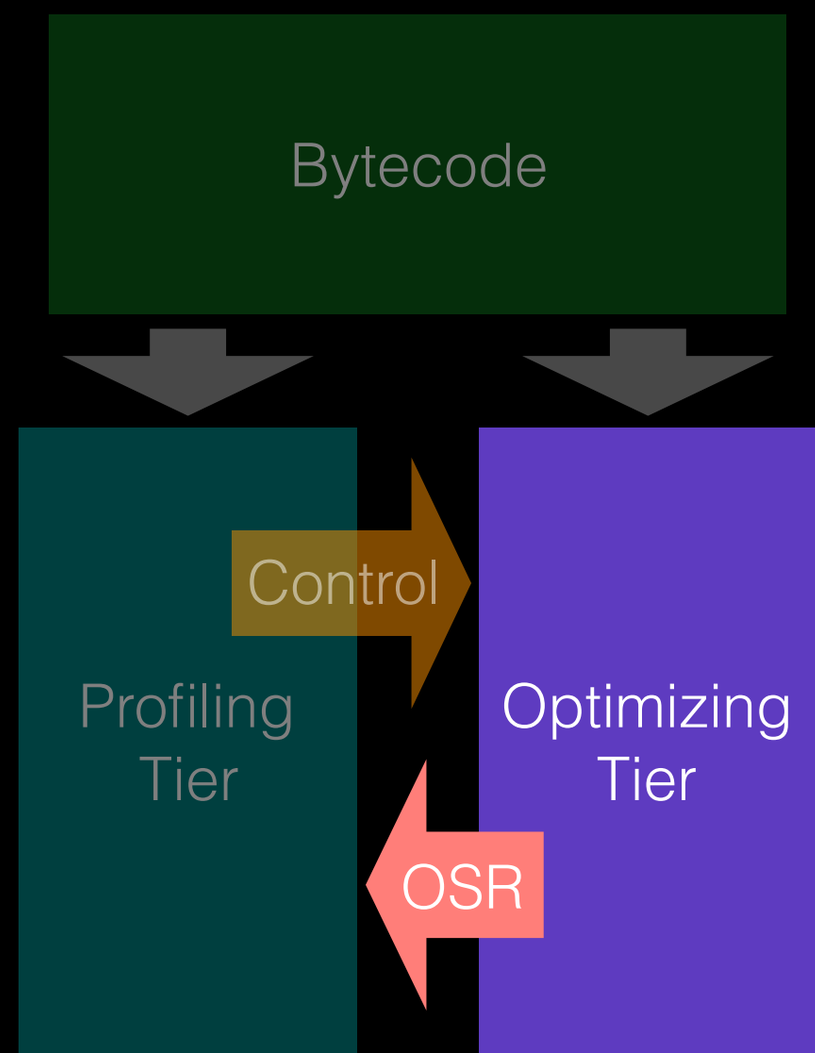
Patchpoint Use Cases

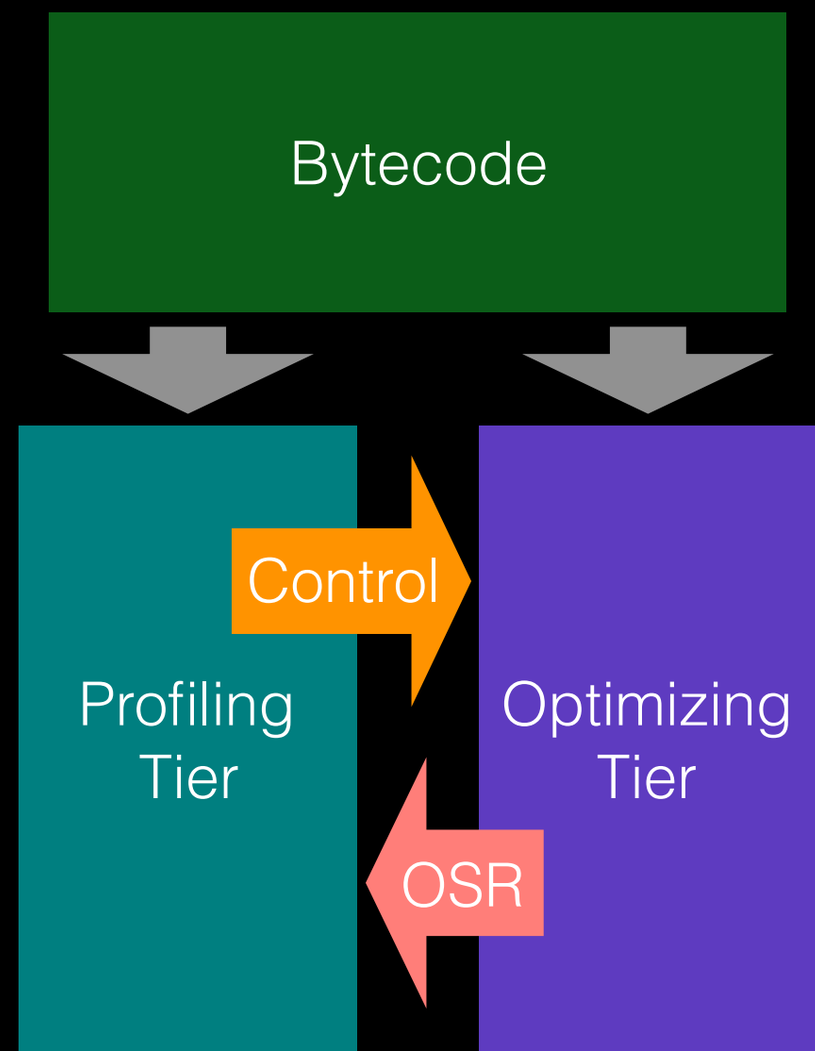


Patchpoint Use Cases




DFG
FTL
Fast JIT
Powerful JIT
DFG Bytecode Parser
DFG Bytecode Parser
DFG IR
DFG IR
DFG Optimizer
DFG Optimizer
DFG Backend
DFG SSA Conversion
DFG SSA IR
DFG SSA Optimizer
DFG-to-B3 lowering
B3 Optimizer
B3 IR
Instruction Selection
Air Optimizer
Assembly IR
Air Backend


Bytecode
Control
Profiling
Tier
Optimizing
Tier
OSR


Bytecode
Control
Profiling Tier
Optimizing Tier
OSR

Speculation in JSC