All About JavaScriptCore's Many Compilers

Filip Pizlo
Apple Inc.

webkit.org

https://svn.webkit.org/repository/webkit/trunk

JavaScriptCore.framework

Safari

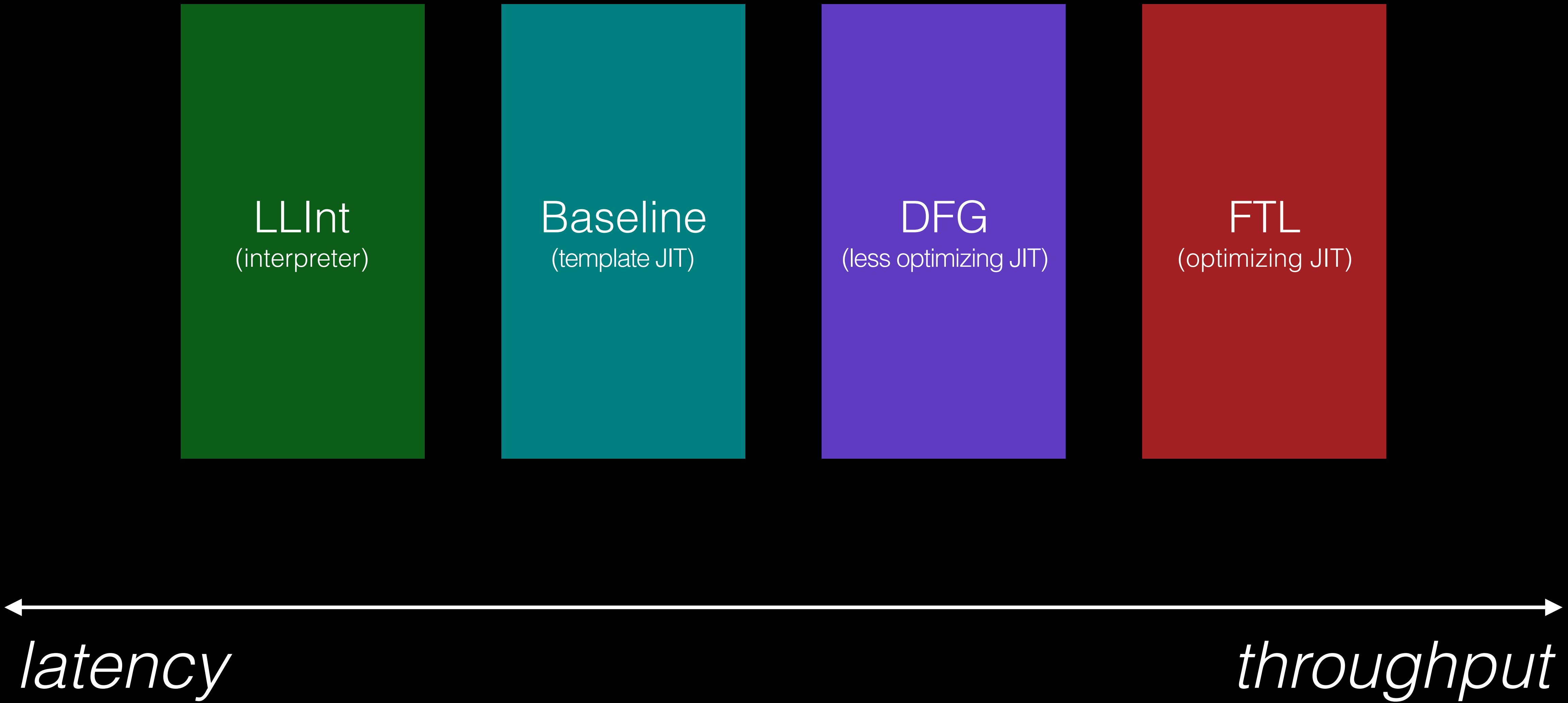
Agenda

- High Level Overview
- Template JITing
- Optimized JITing
  - DFG
  - FTL
  - BBQ
  - OMG

Agenda

- High Level Overview
- Template JITing
- Optimized JITing
  - DFG
  - FTL
  - BBQ
  - OMG

Four Tiers



LLInt
(interpreter)

Baseline
(template JIT)

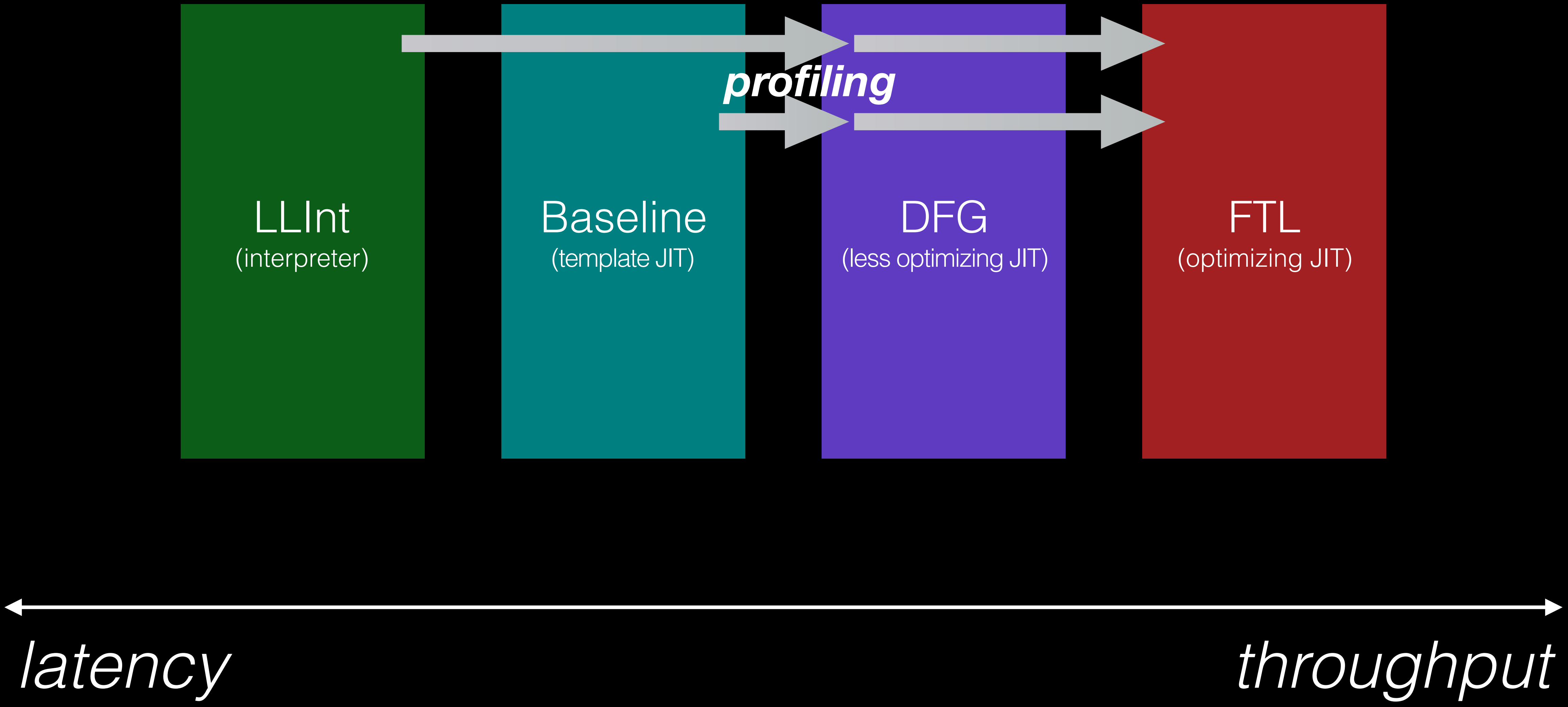
DFG
(less optimizing JIT)

FTL
(optimizing JIT)

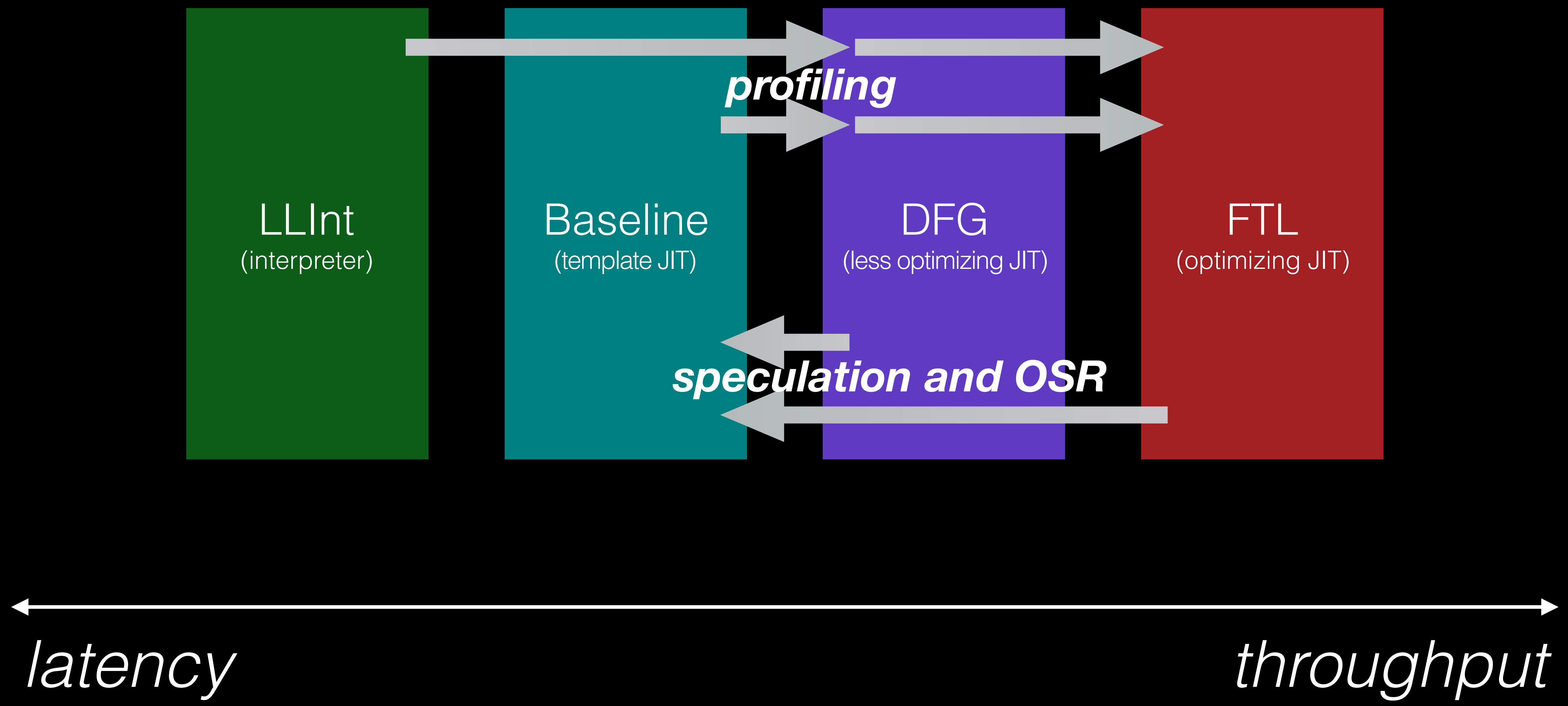
latency

throughput

Four Tiers


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

Four Tiers



LLInt
(interpreter)

Baseline
(template JIT)

DFG
(less optimizing JIT)

FTL
(optimizing JIT)

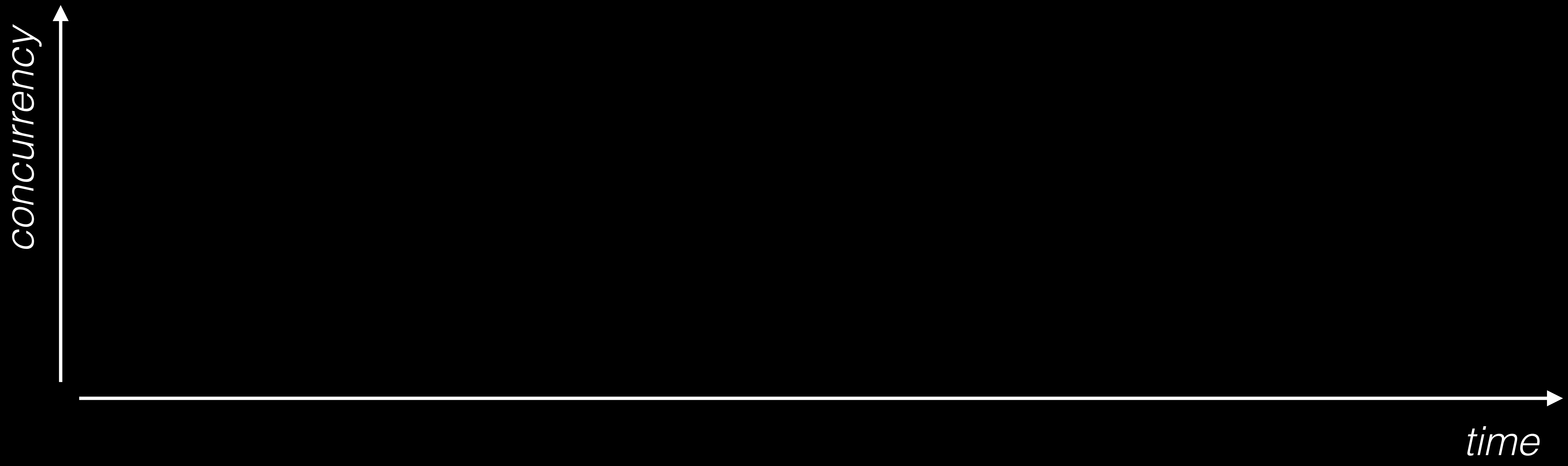
profiling

speculation and OSR

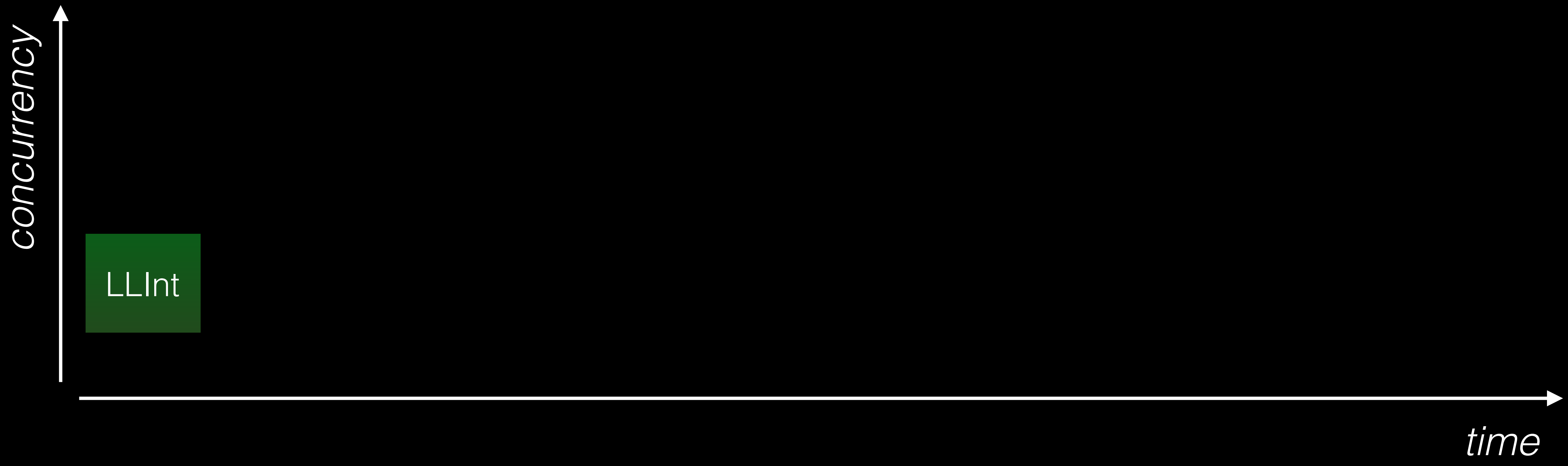
latency

throughput

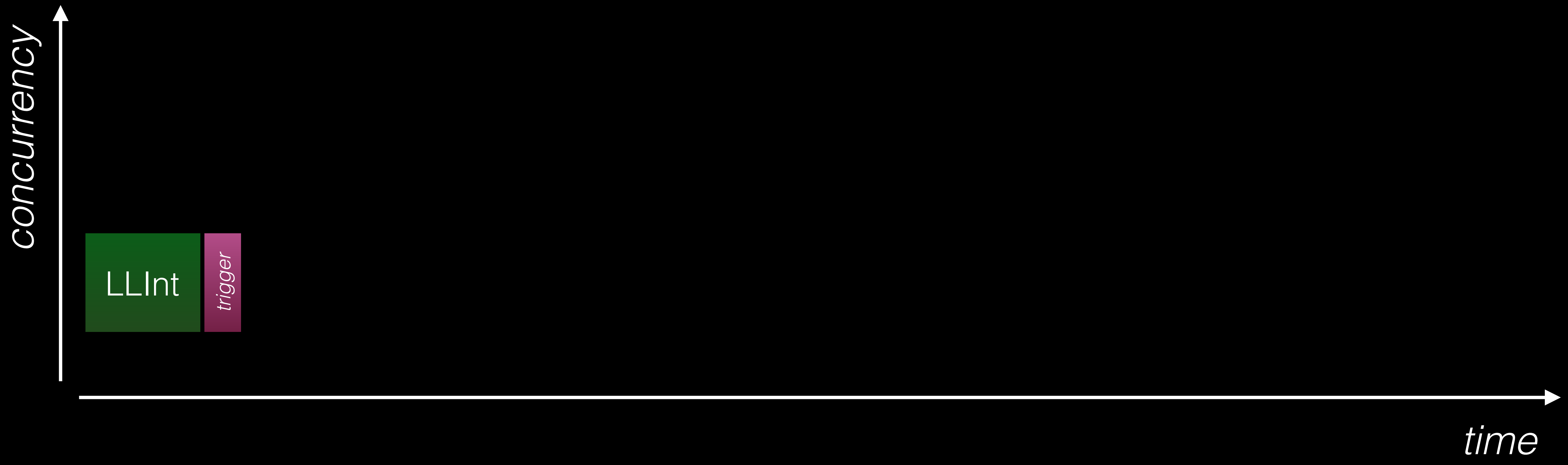
```
"use strict";

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

print(result);
```



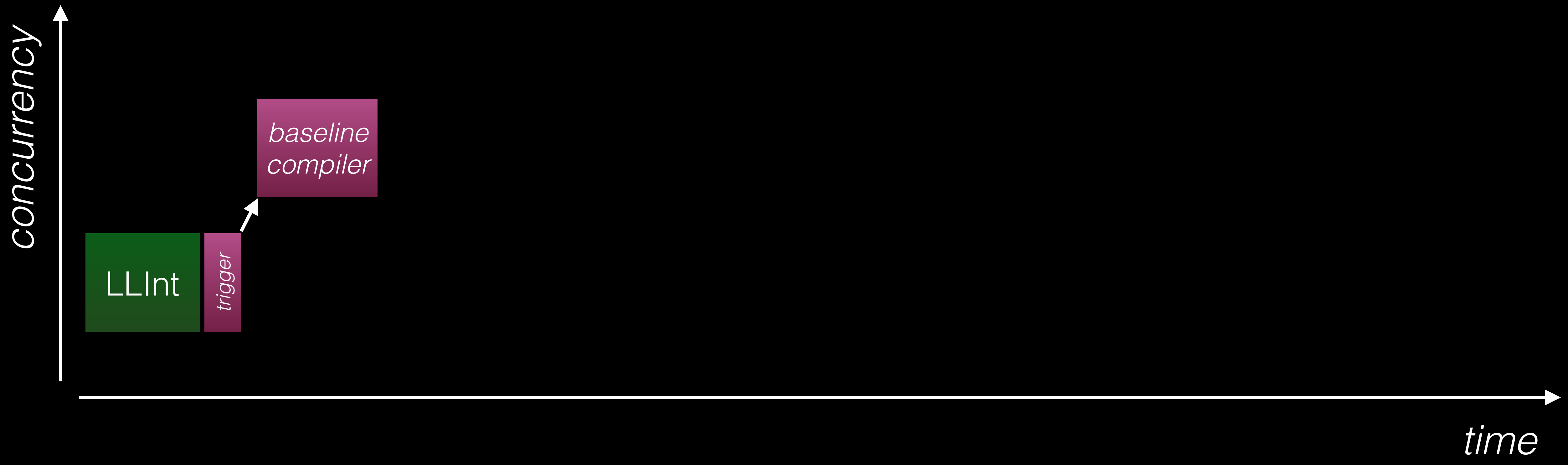
```
"use strict";

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

print(result);
```



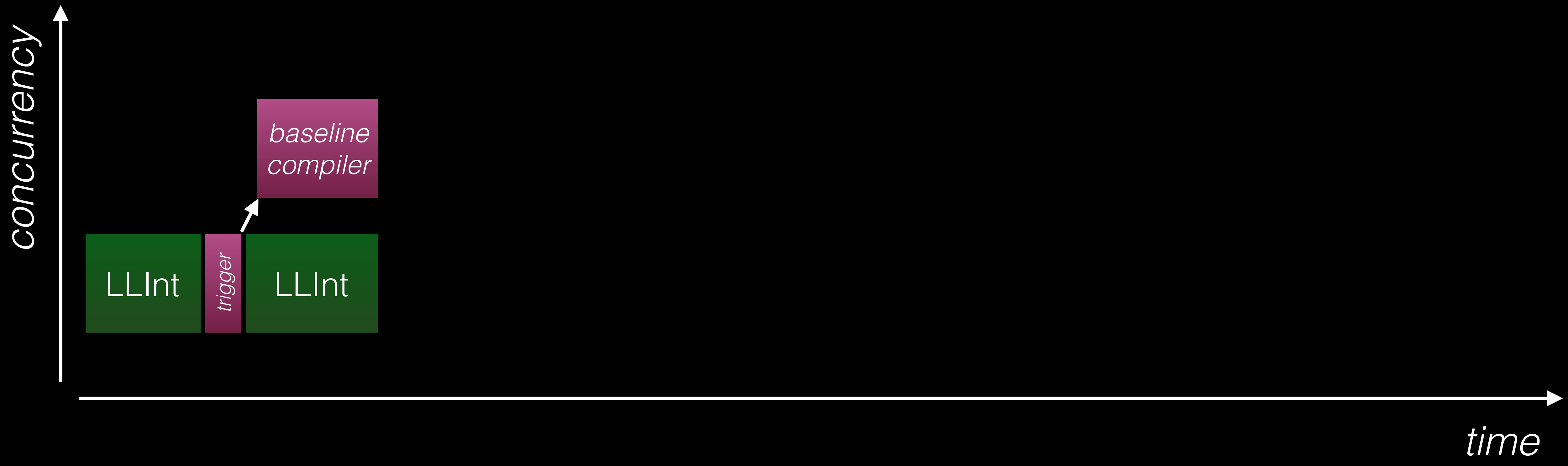
```
"use strict";

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

print(result);
```



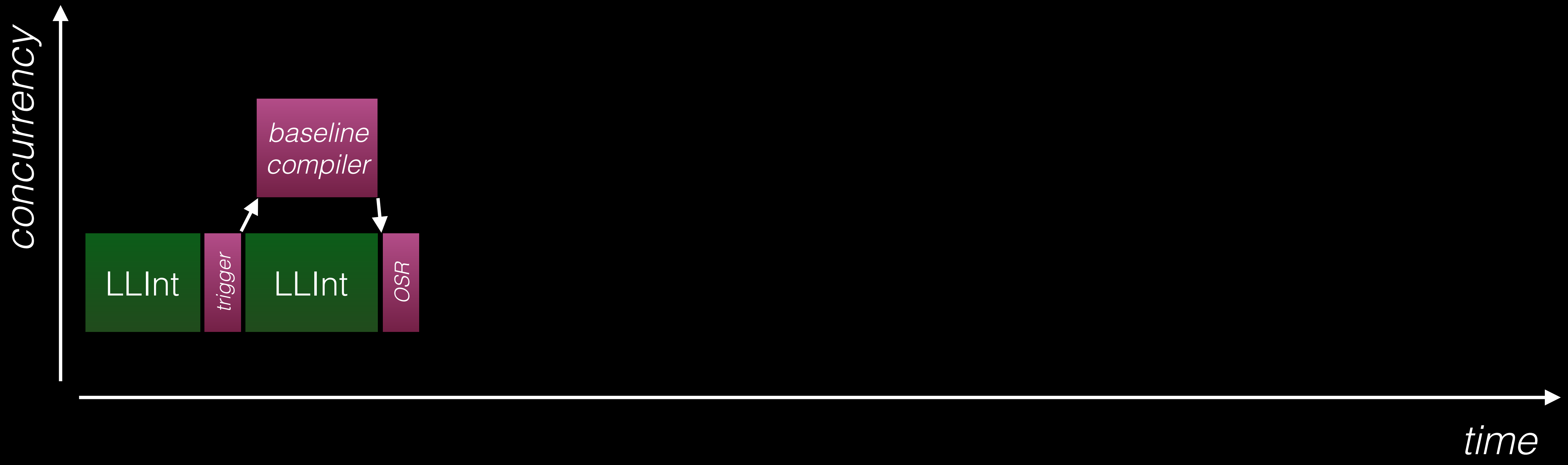
```
"use strict";

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

print(result);
```



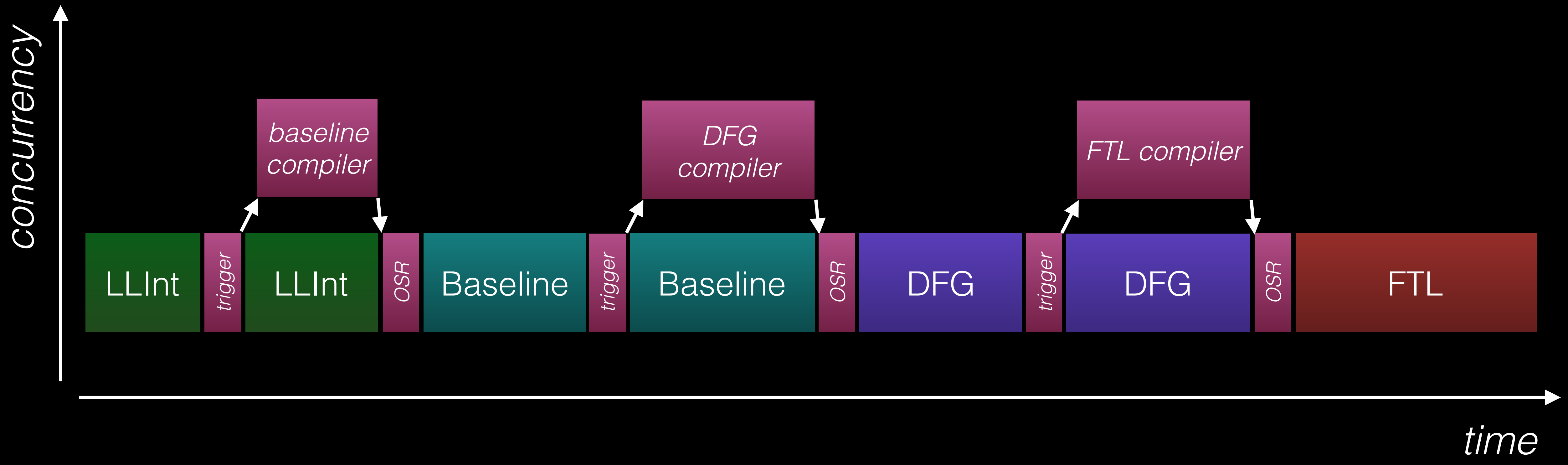
```
"use strict";

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);
```

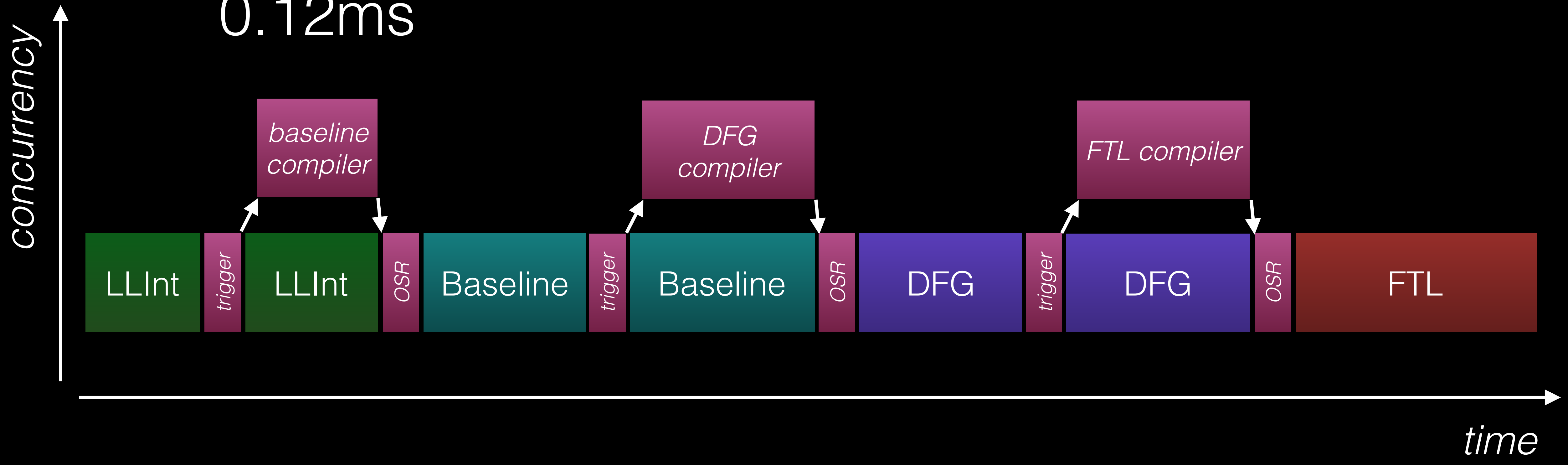


```
"use strict";

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

print(result);
```

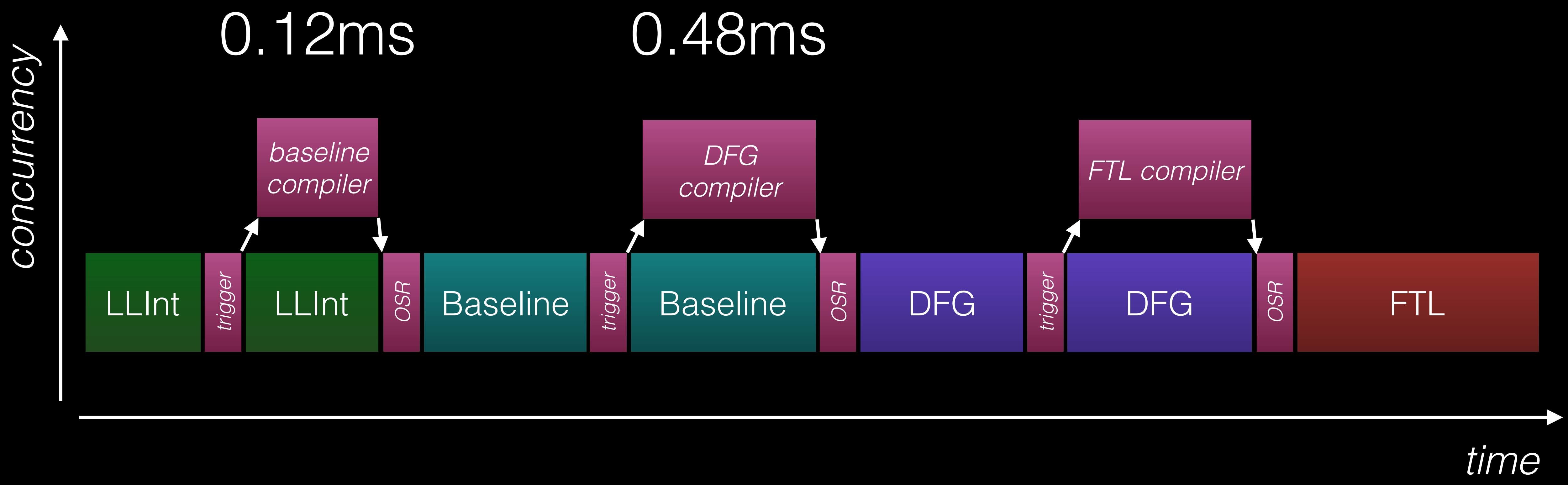


```
"use strict";

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

print(result);
```

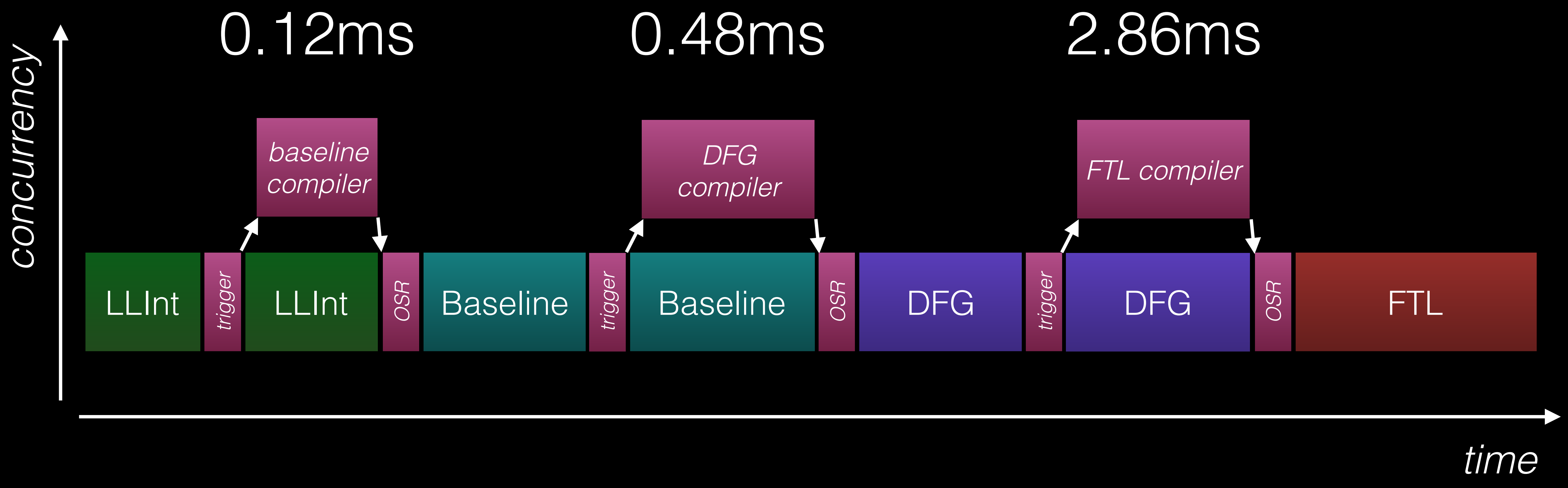
0.12ms



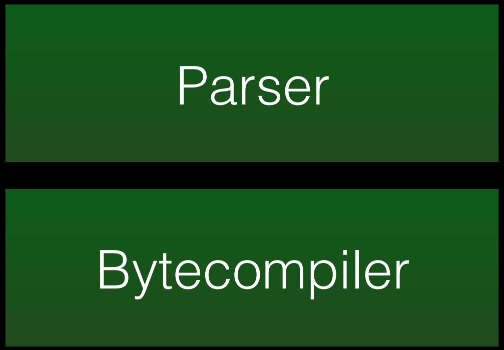
```
"use strict";

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);
```

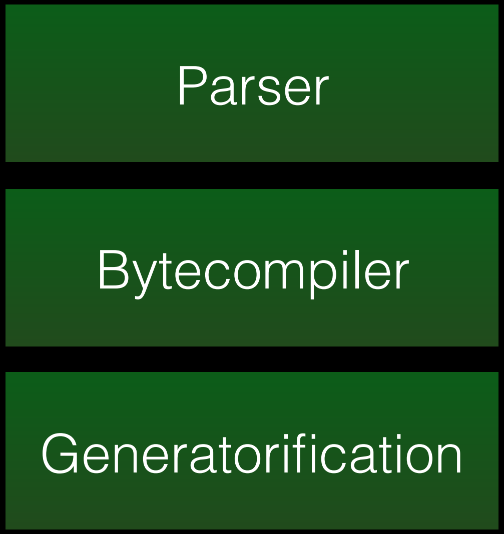


Parser



Parser

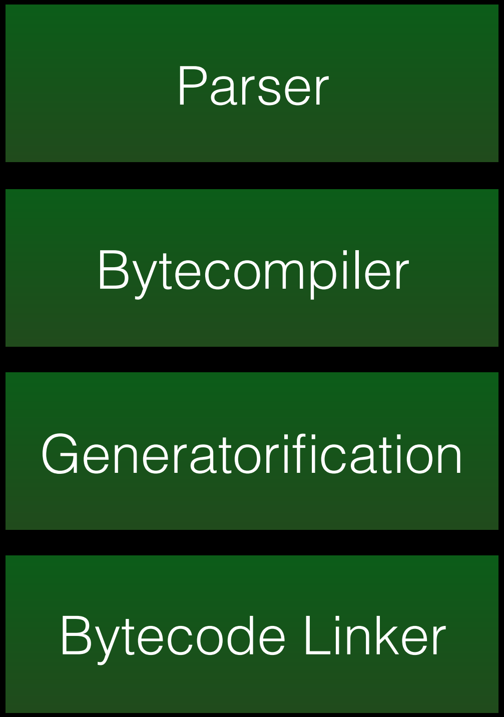
Bytecompiler



Parser

Bytecompiler

Generatorification

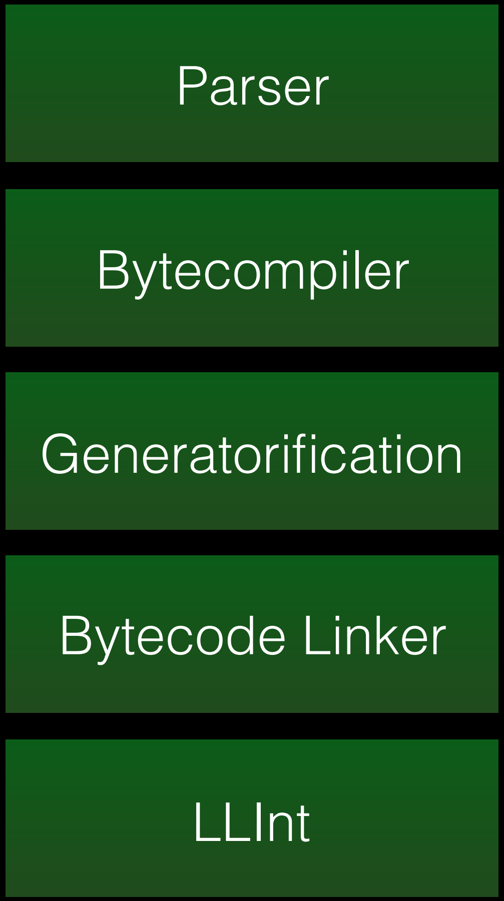


Parser

Bytecompiler

Generatorification

Bytecode Linker



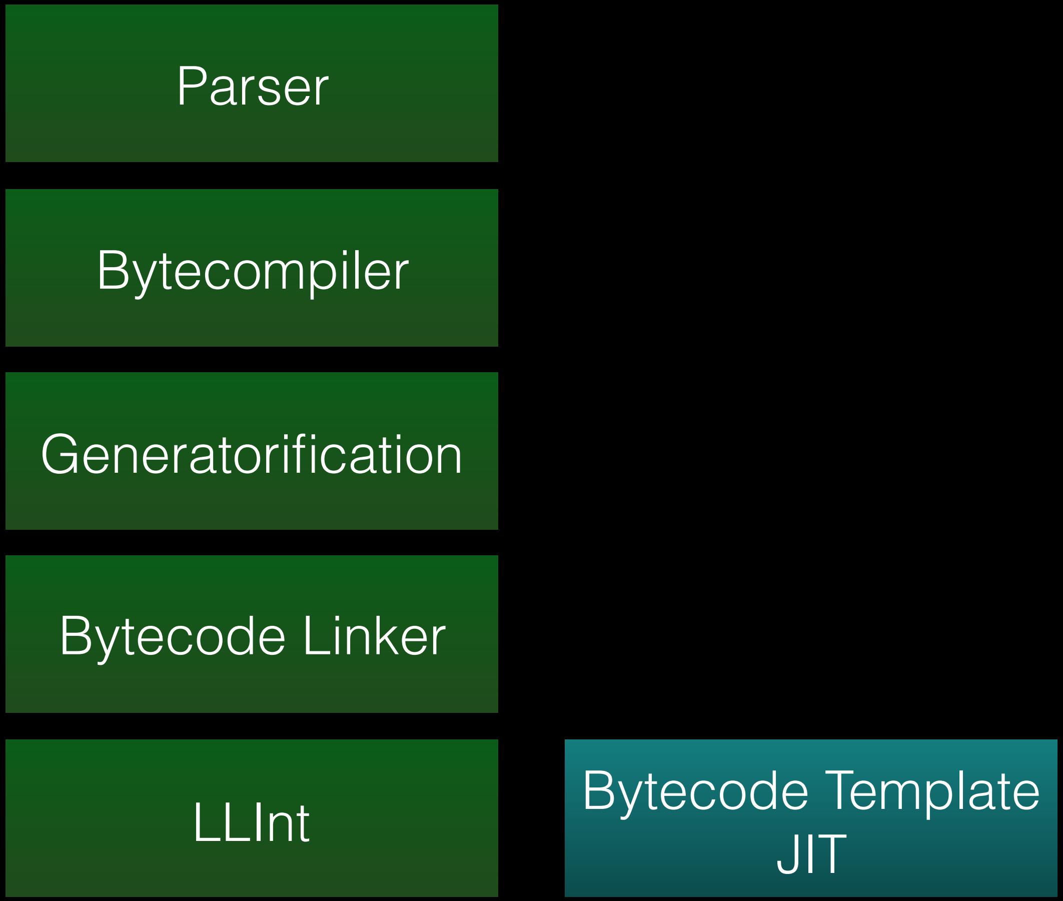
Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt



Parser

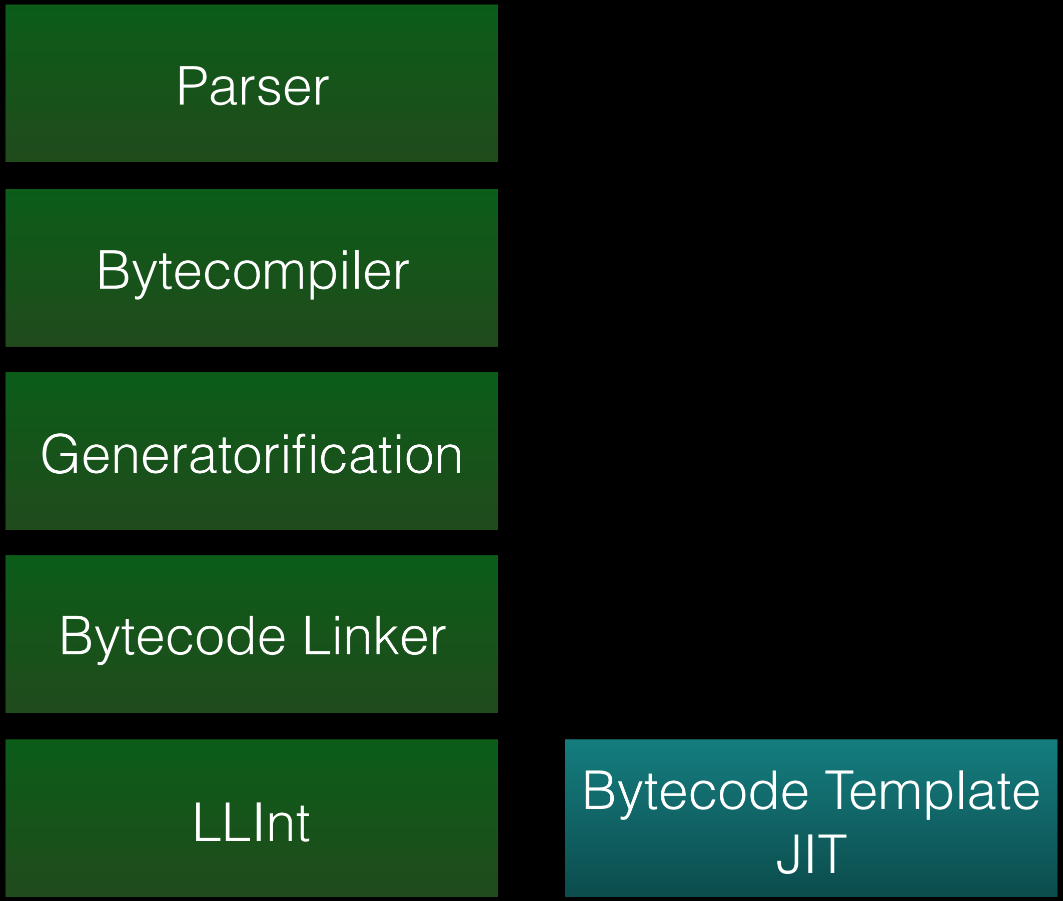
Bytecompiler

Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT



Parser

Bytecompiler

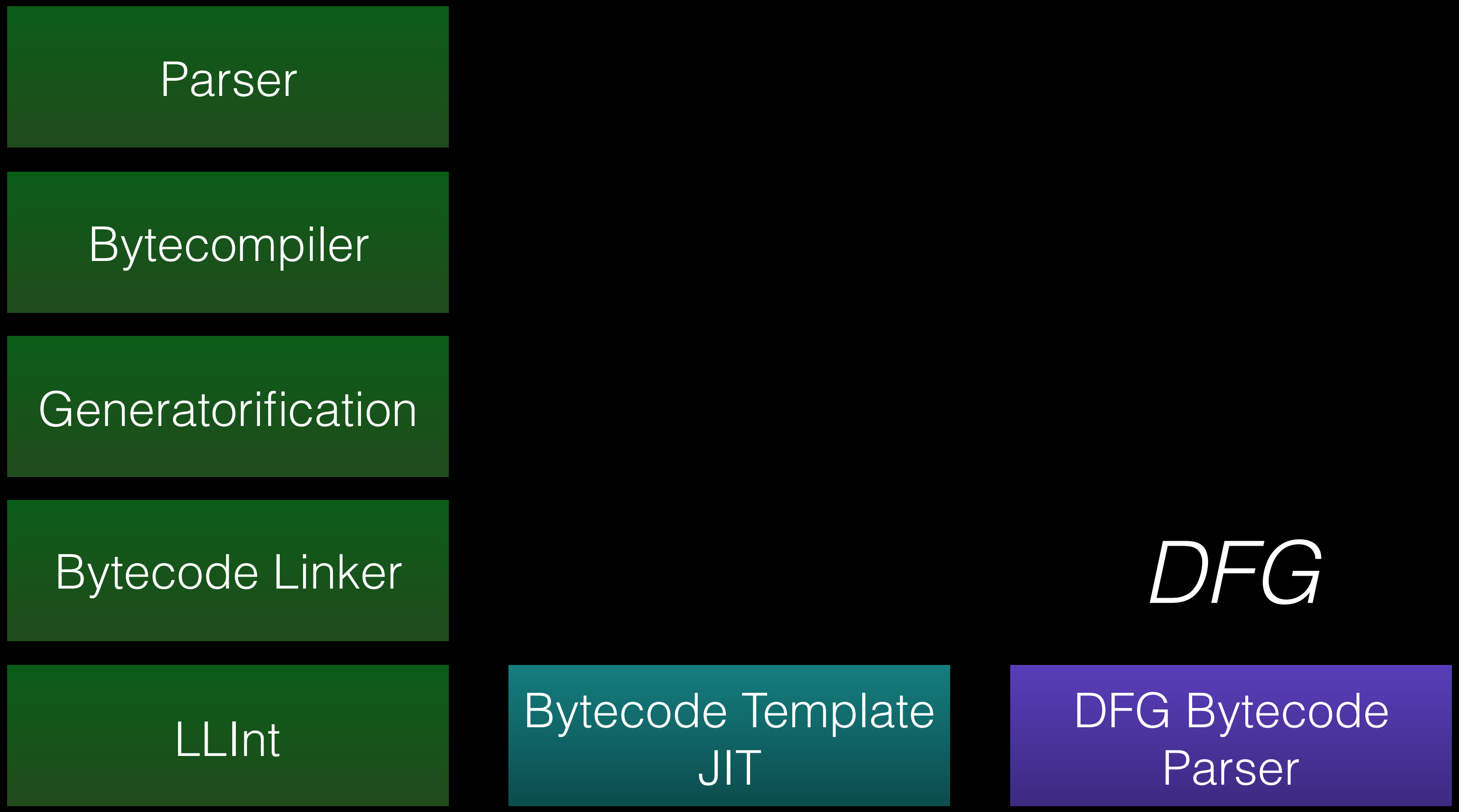
Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT

DFG



DFG

Parser

Bytecompiler

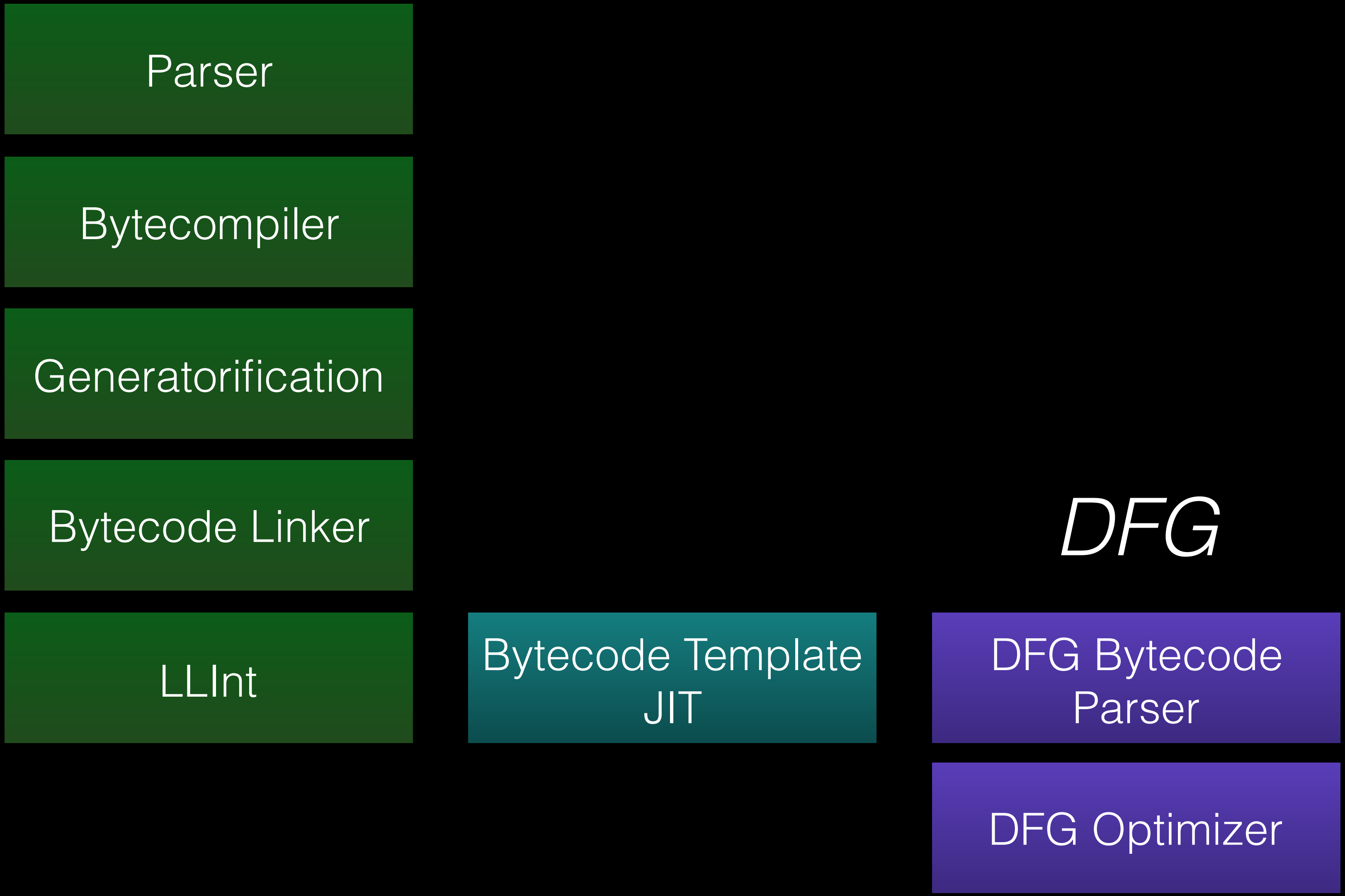
Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT

DFG Bytecode Parser



Parser

Bytecompiler

Generatorification

Bytecode Linker

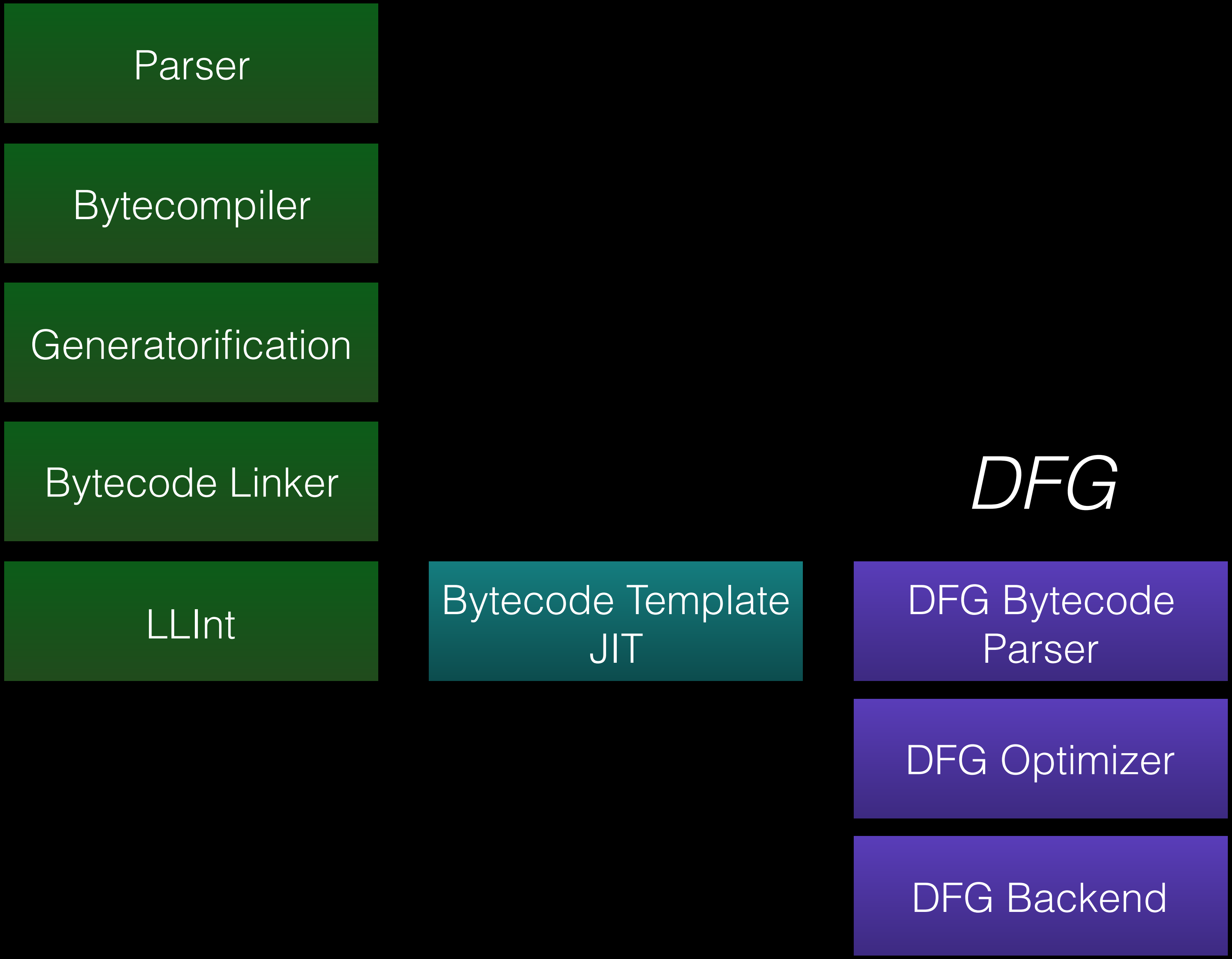
LLInt

Bytecode Template JIT

DFG

DFG Bytecode Parser

DFG Optimizer



Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt

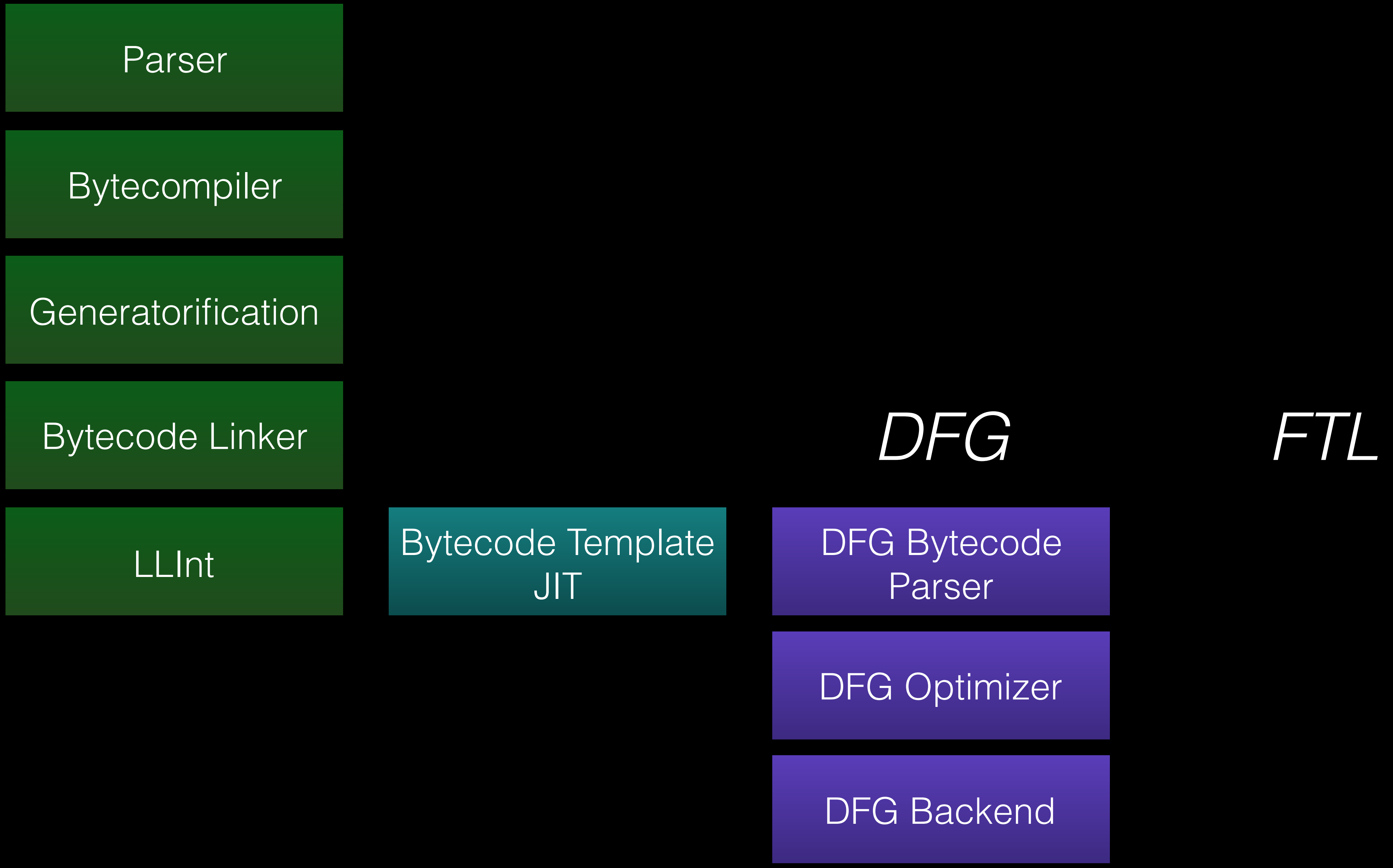
Bytecode Template JIT

DFG

DFG Bytecode Parser

DFG Optimizer

DFG Backend



Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT

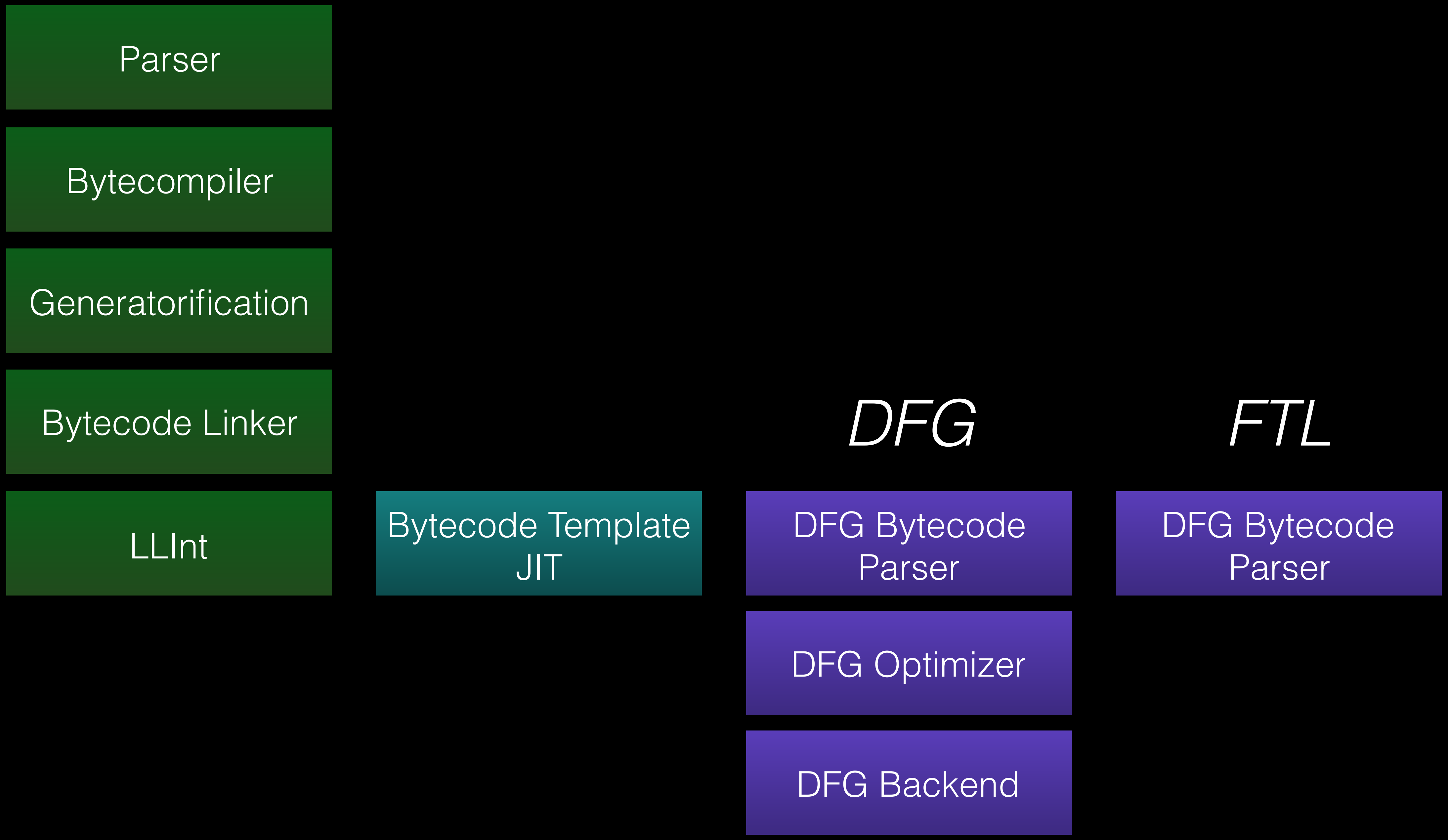
DFG

DFG Bytecode Parser

DFG Optimizer

DFG Backend

FTL



Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT

DFG

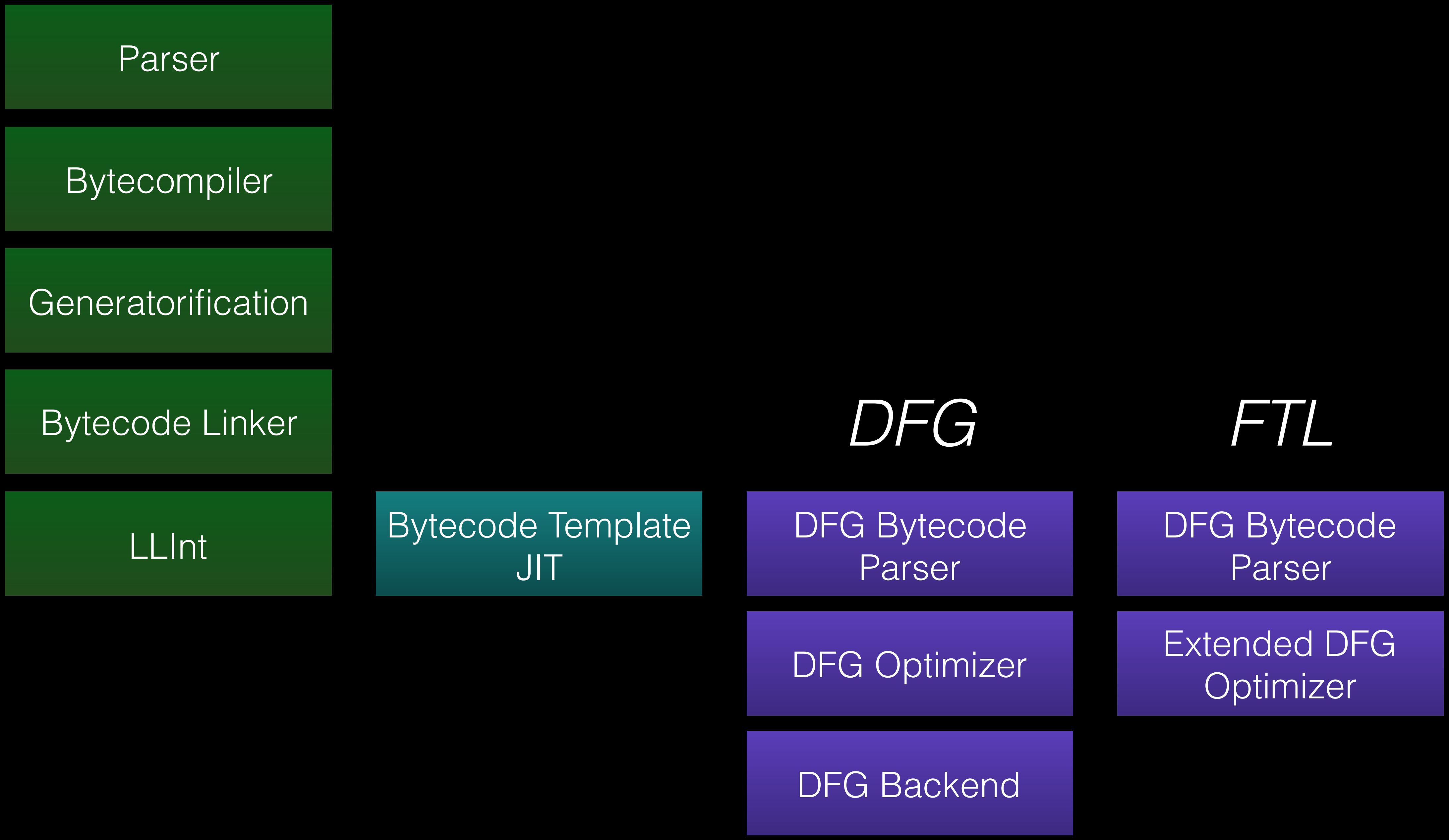
DFG Bytecode Parser

DFG Optimizer

DFG Backend

FTL

DFG Bytecode Parser



Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT

DFG

DFG Bytecode Parser

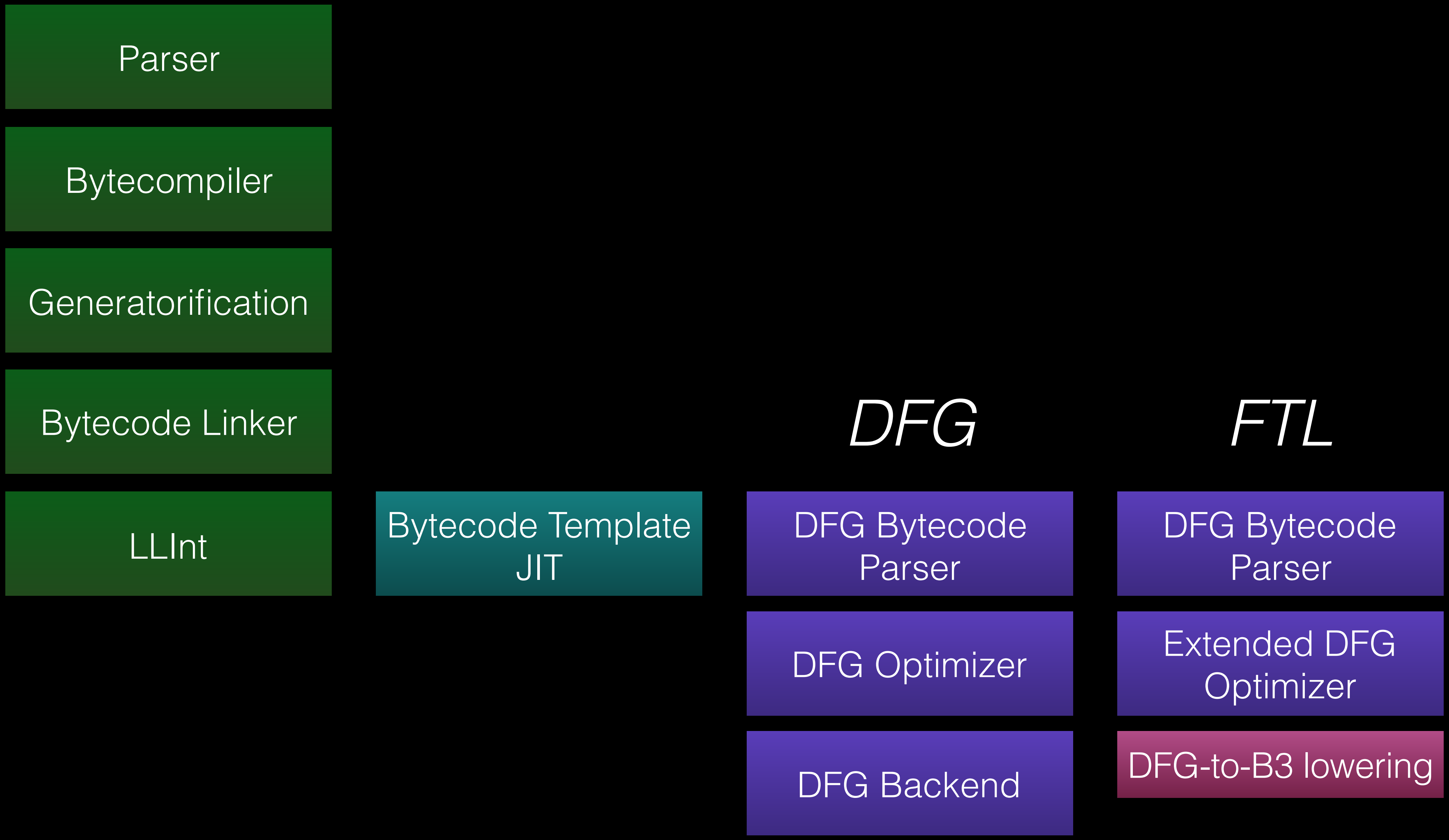
DFG Optimizer

DFG Backend

FTL

DFG Bytecode Parser

Extended DFG Optimizer



Parser

Bytecompiler

Generatorification

Bytecode Linker

LLInt

Bytecode Template JIT

DFG

DFG Bytecode Parser

DFG Optimizer

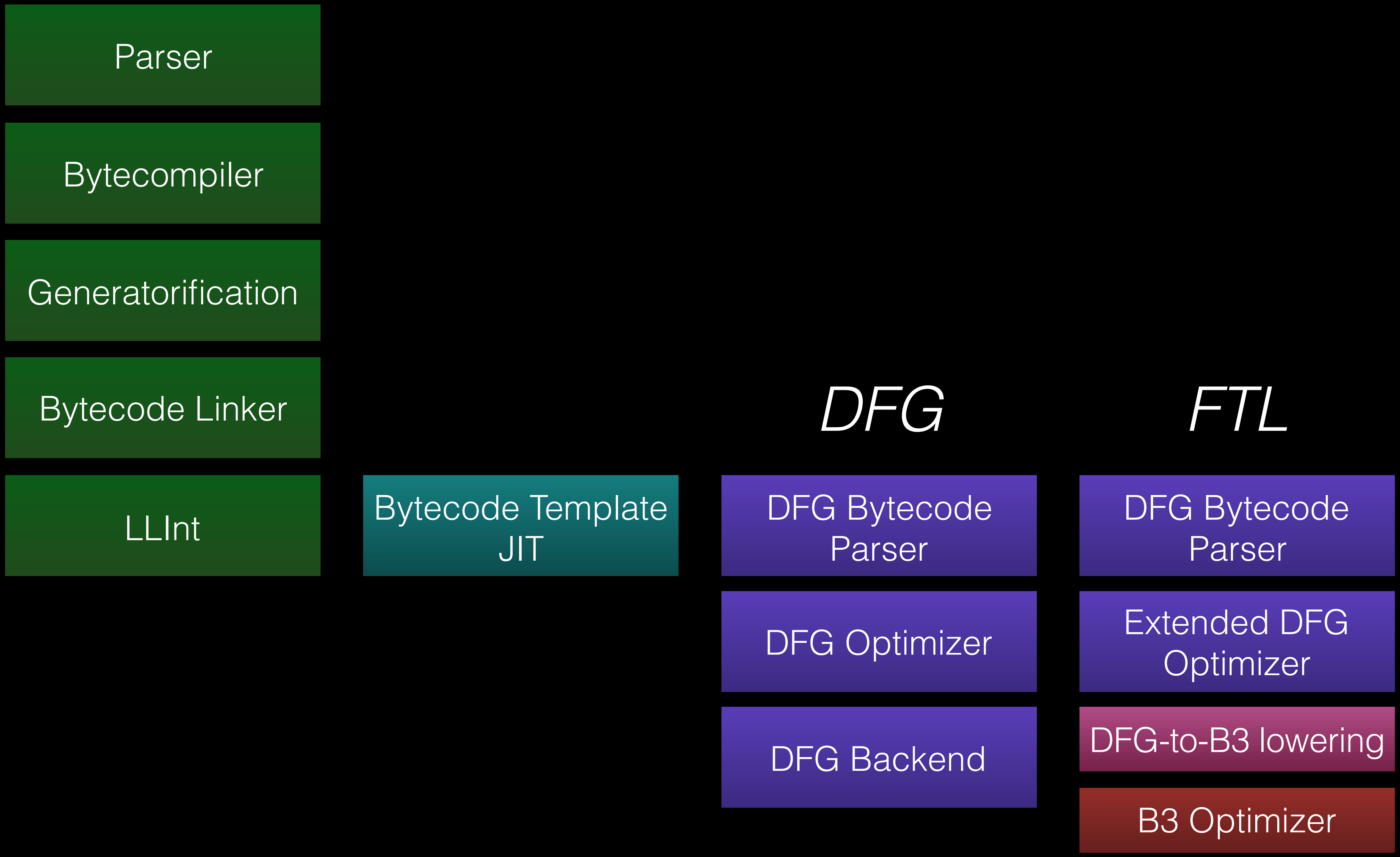
DFG Backend

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

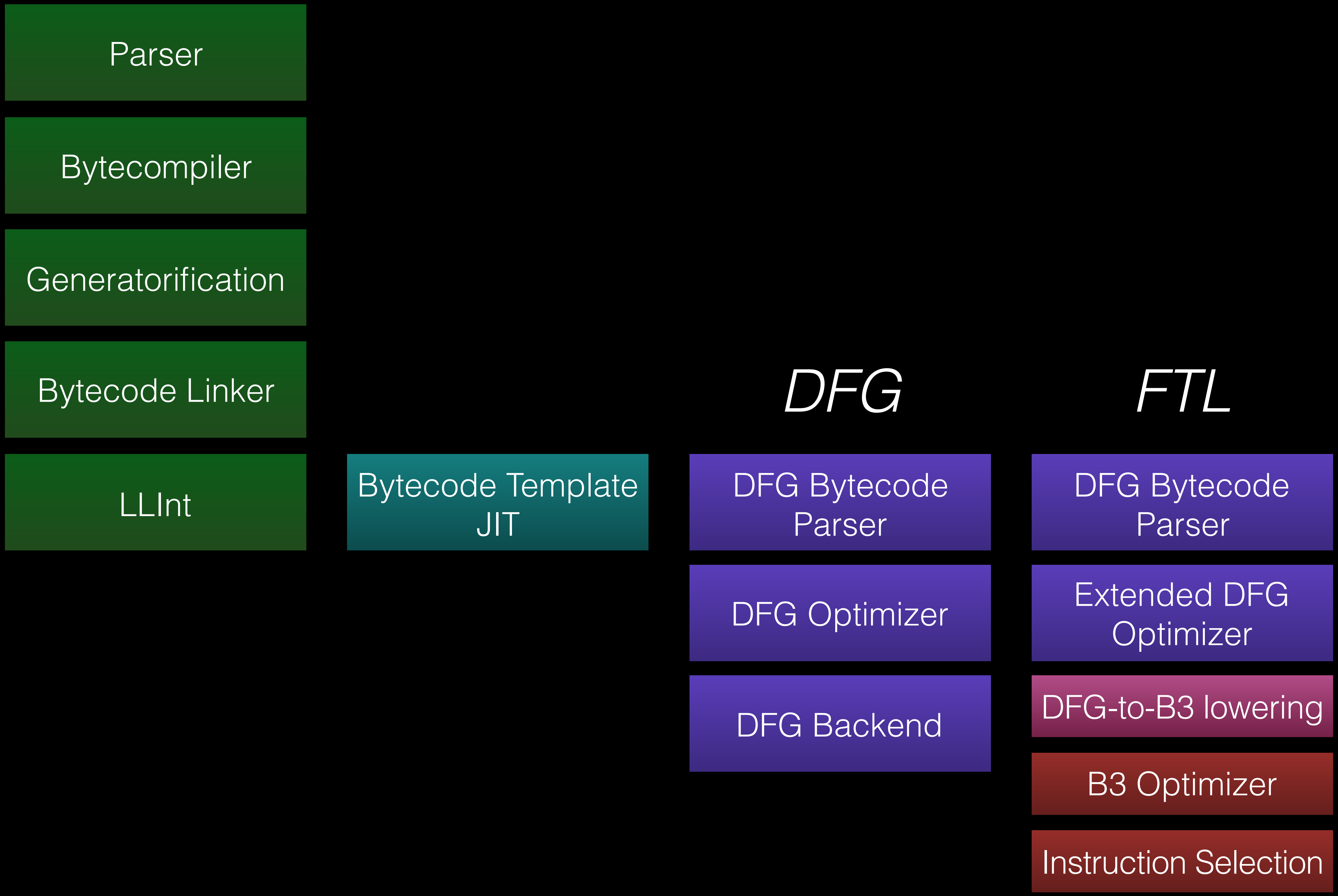
FTL

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

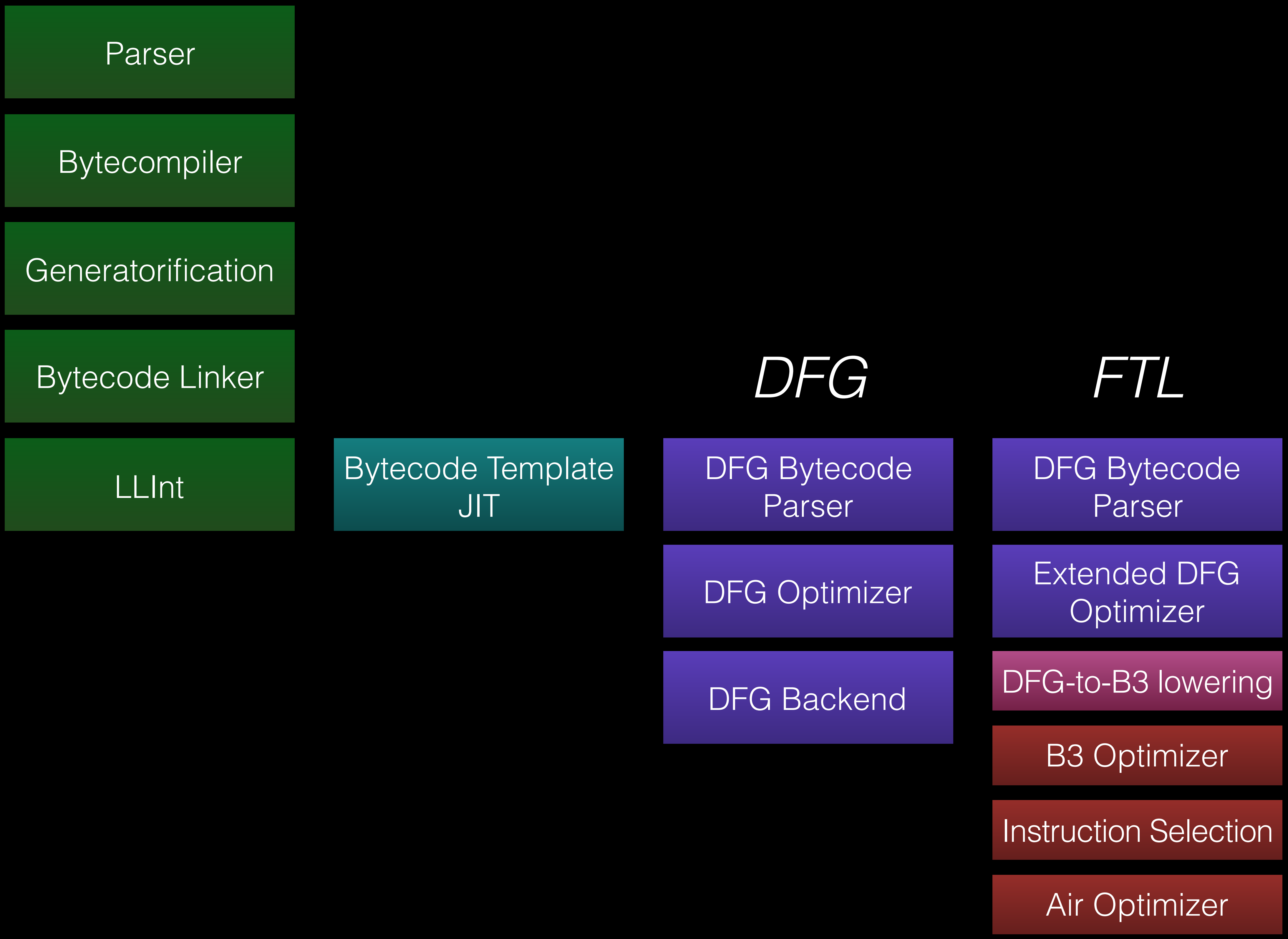
DFG Bytecode Parser

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

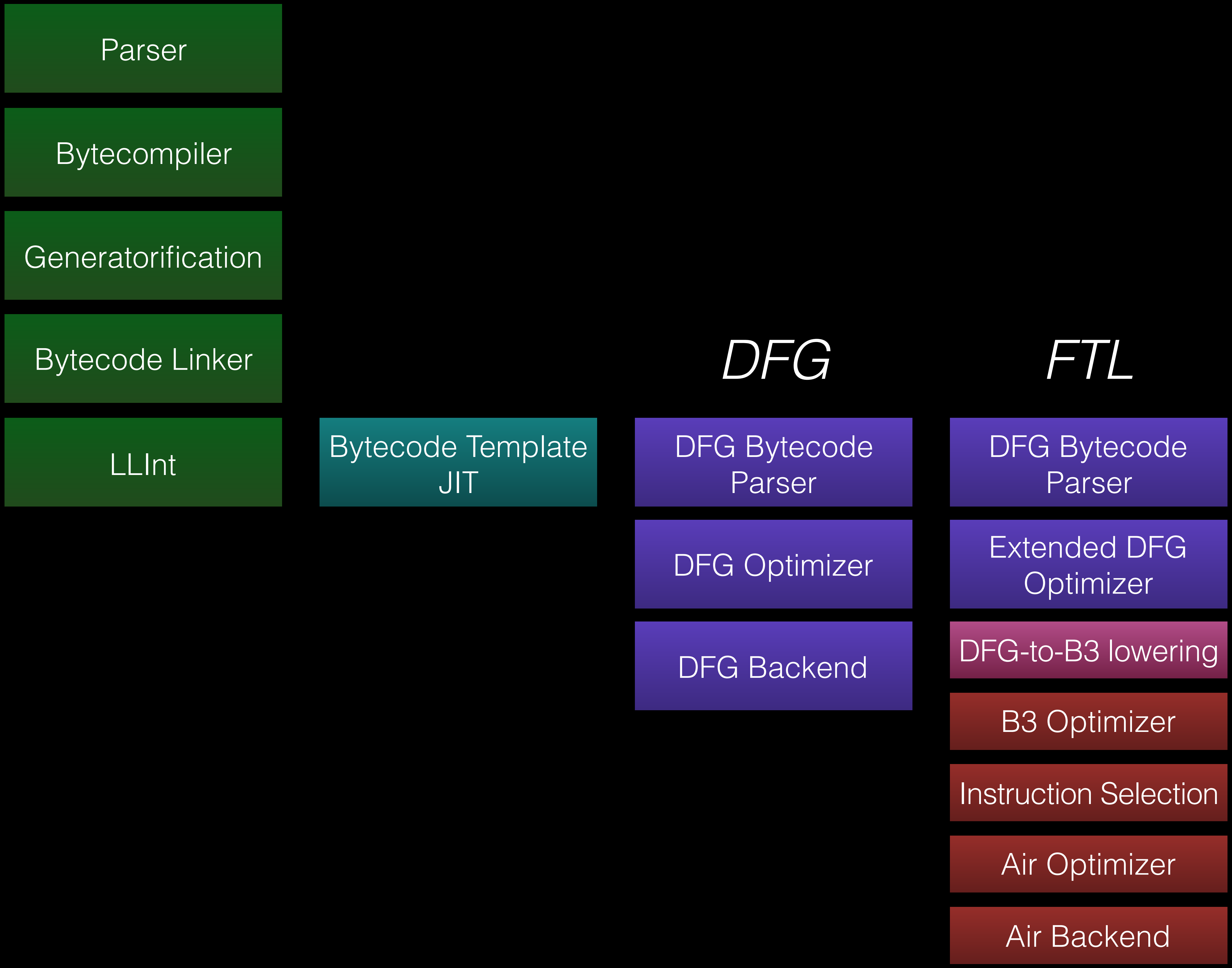
Extended DFG Optimizer

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

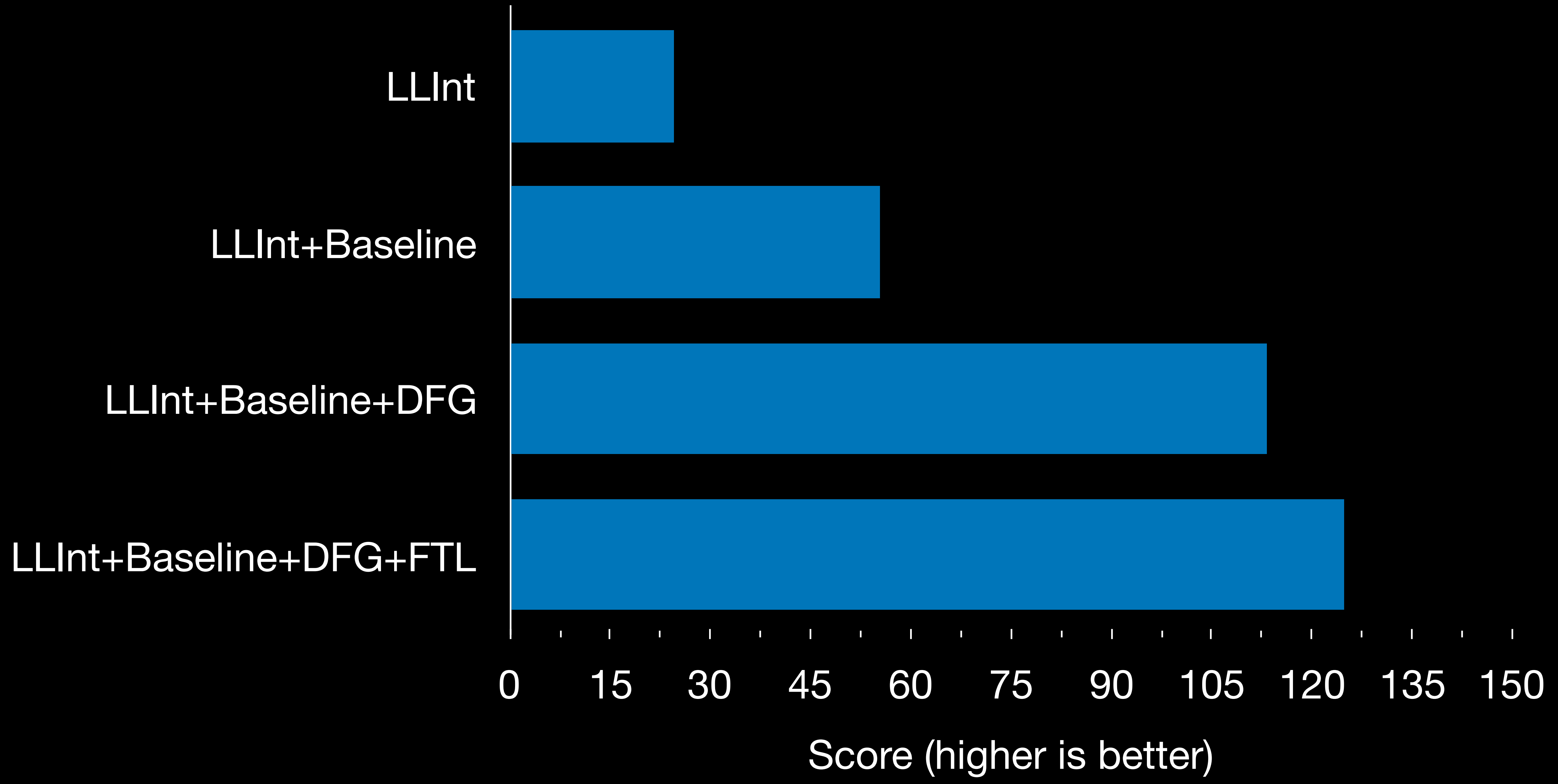
Instruction Selection

Air Optimizer

Air Backend

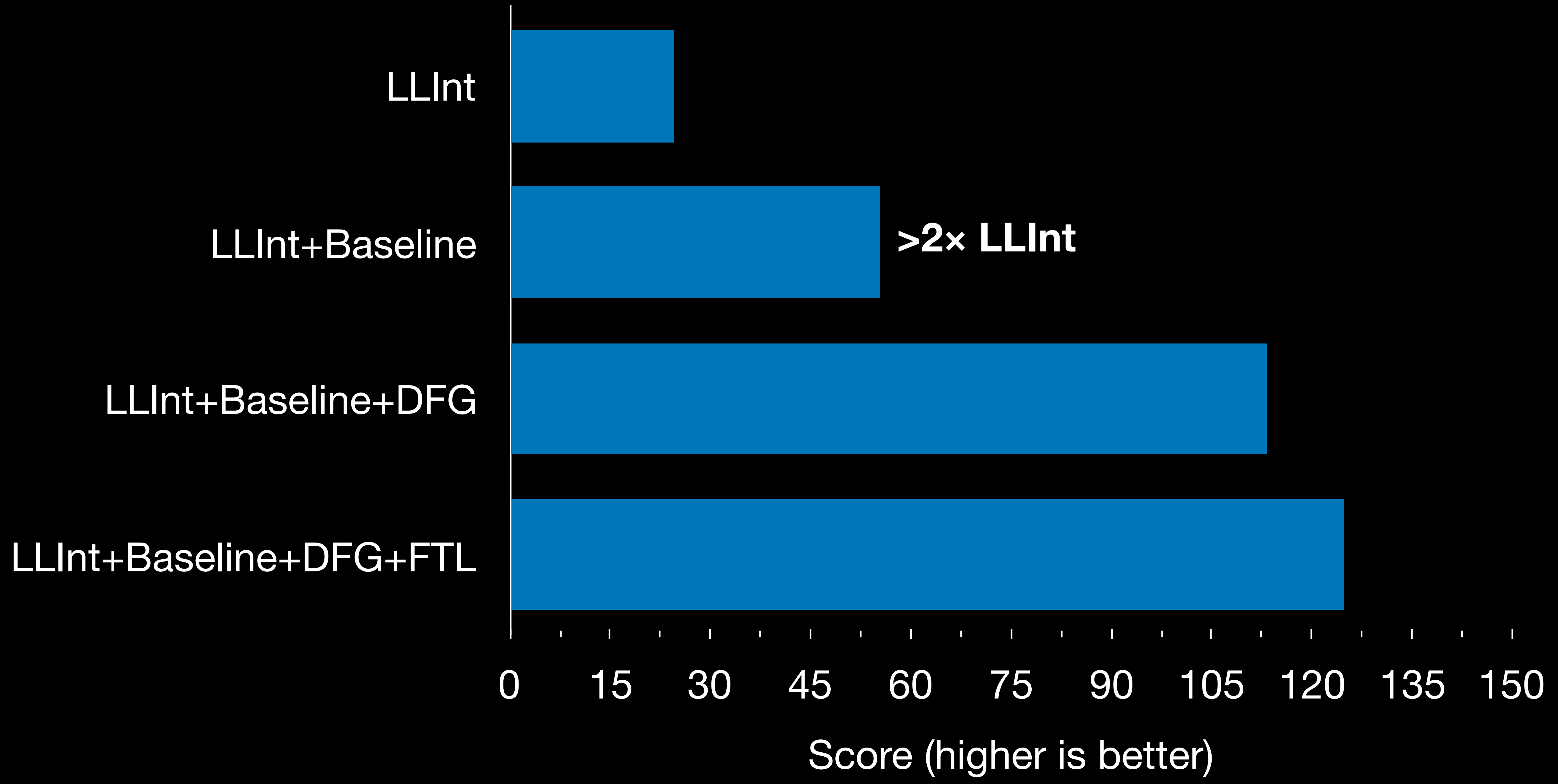
JetStream 2 Score

on my computer one day



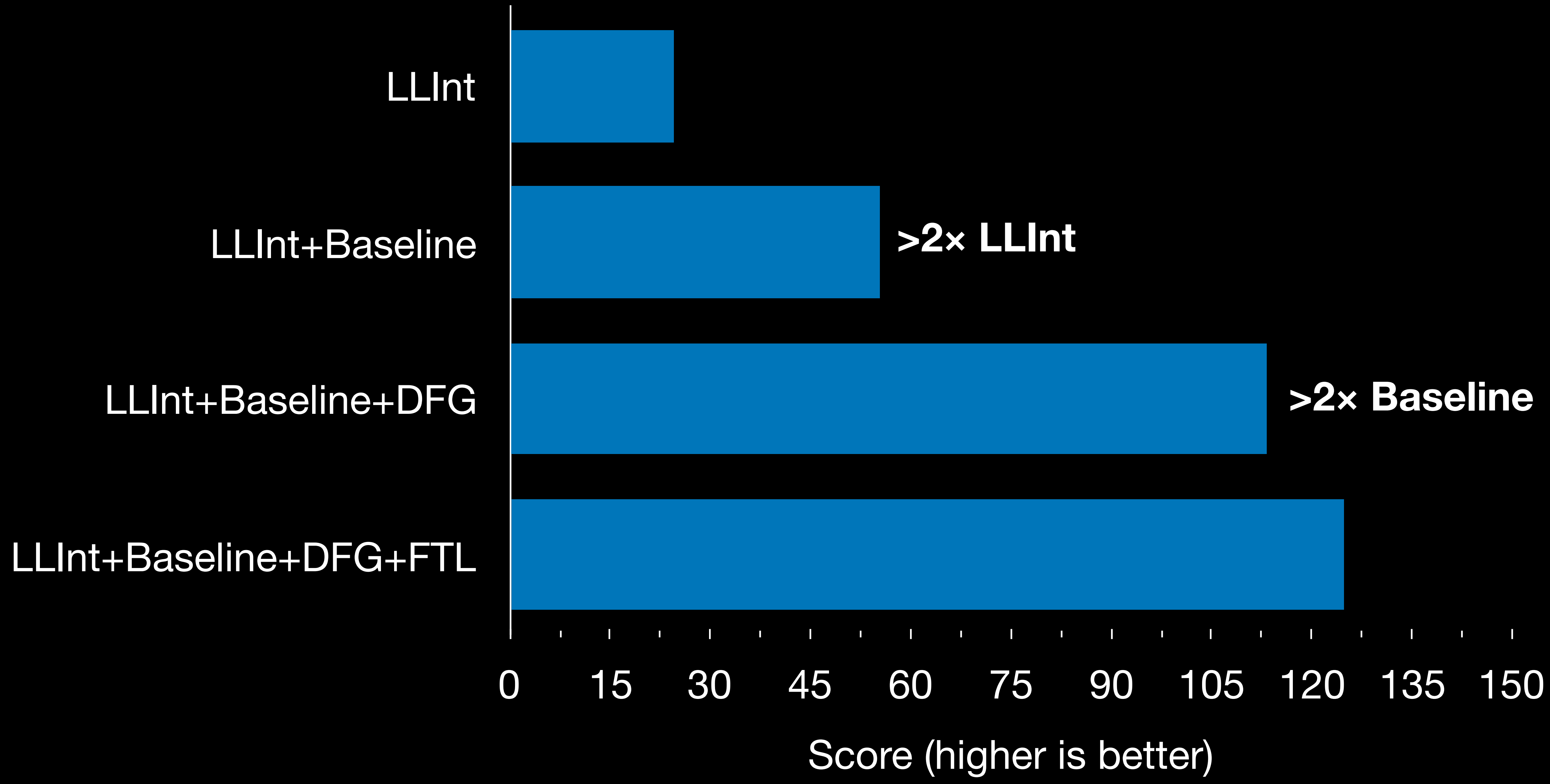
JetStream 2 Score

on my computer one day



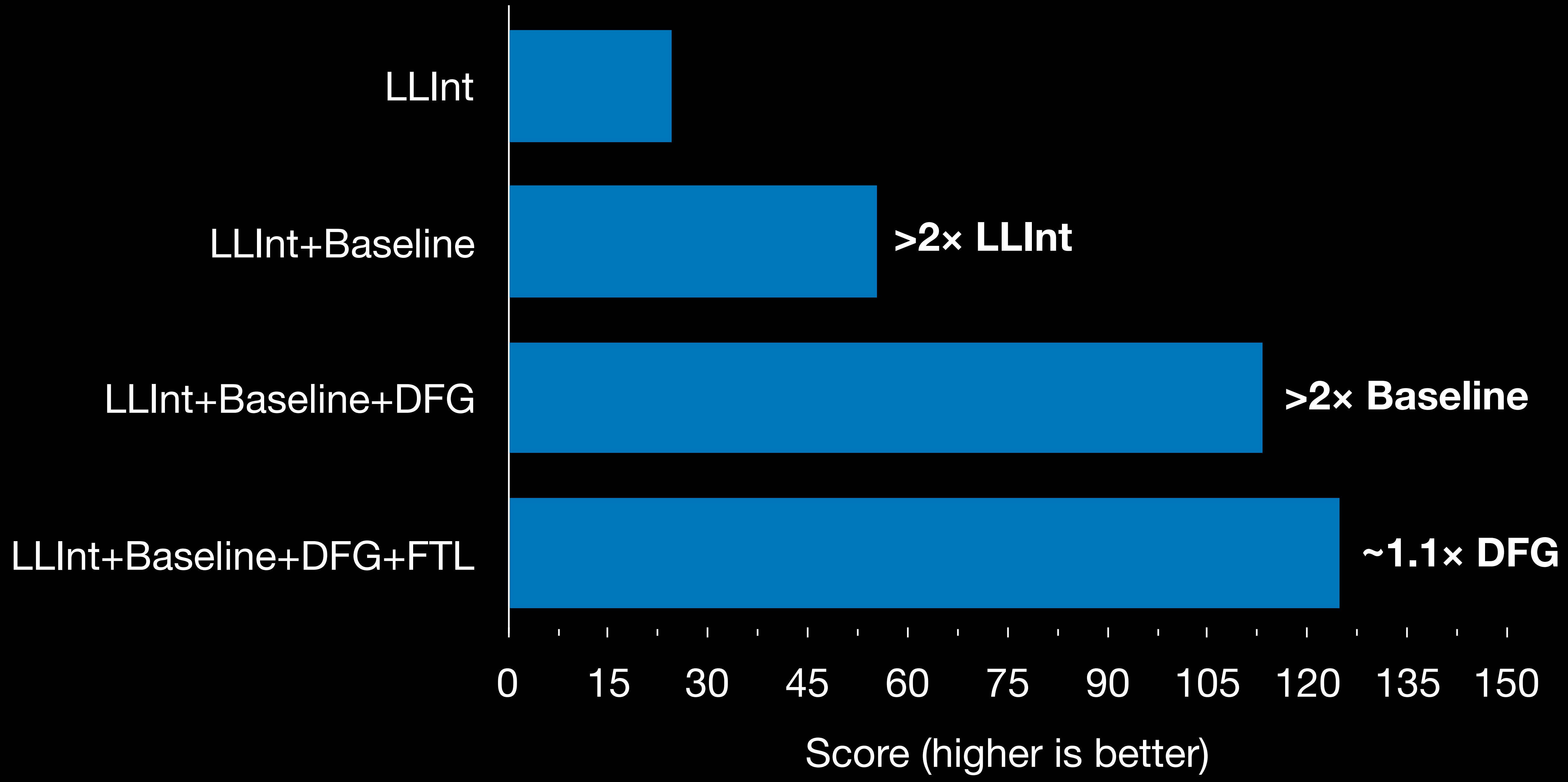
JetStream 2 Score

on my computer one day



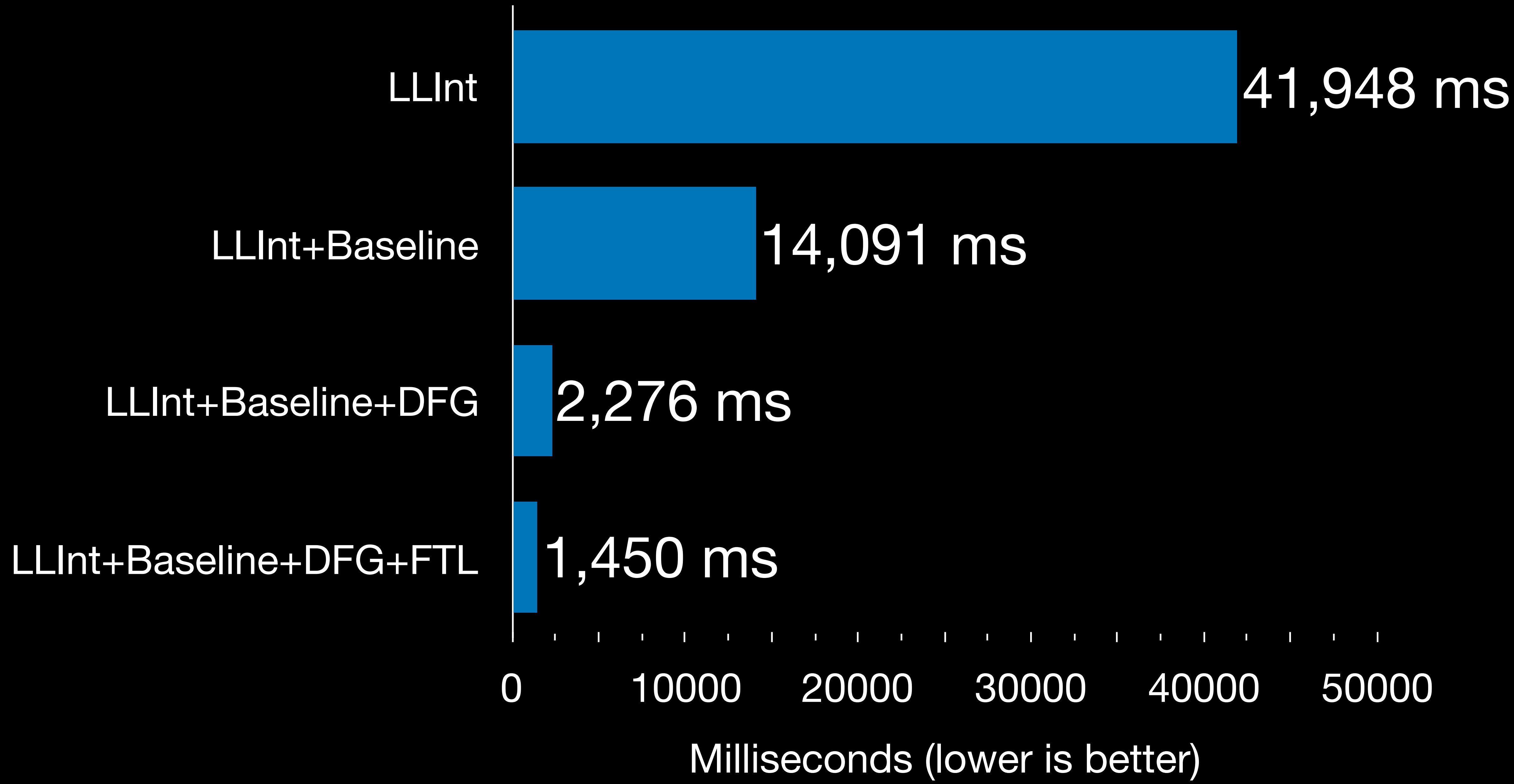
JetStream 2 Score

on my computer one day



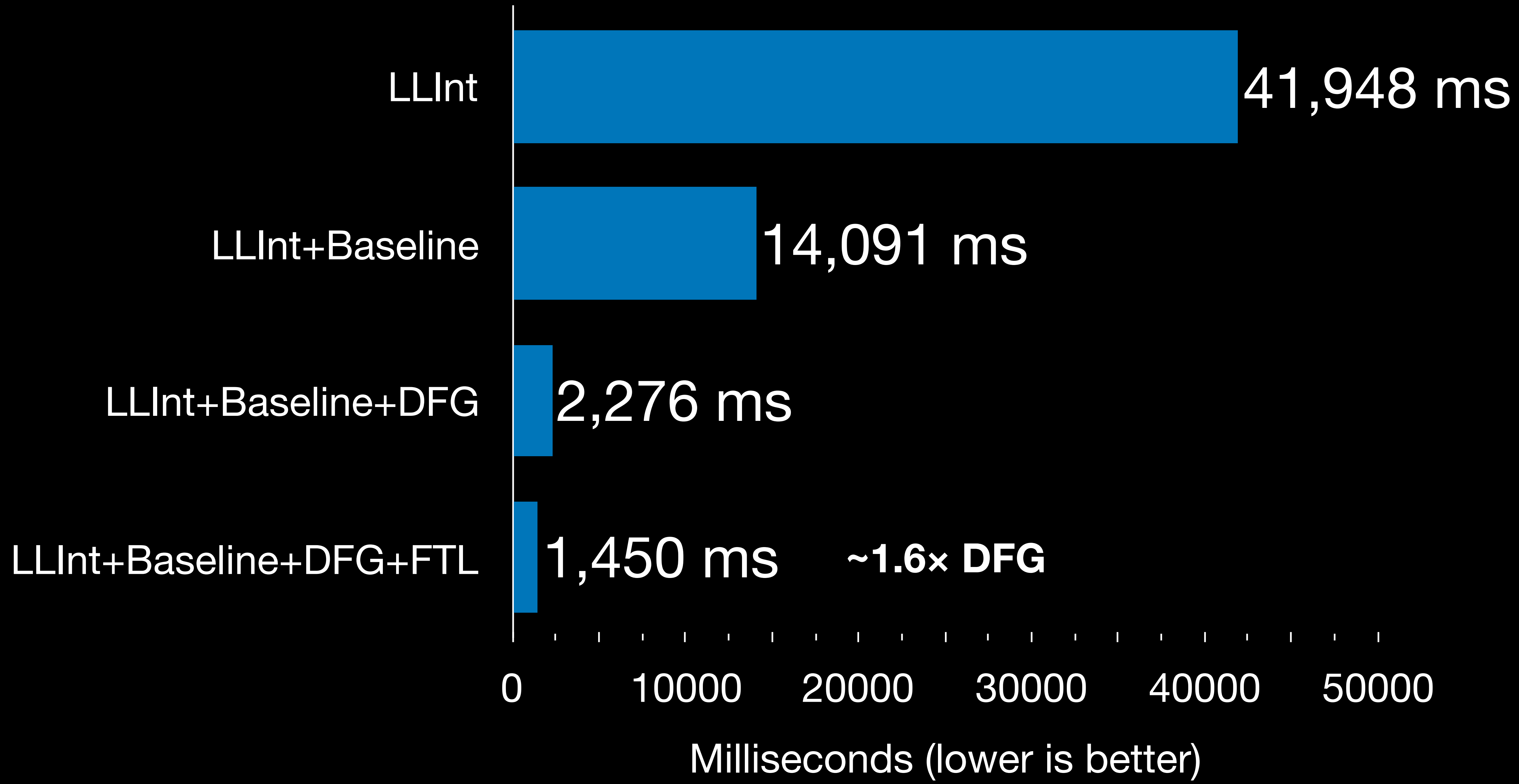
JetStream 2 “gaussian-blur”

on my computer one day



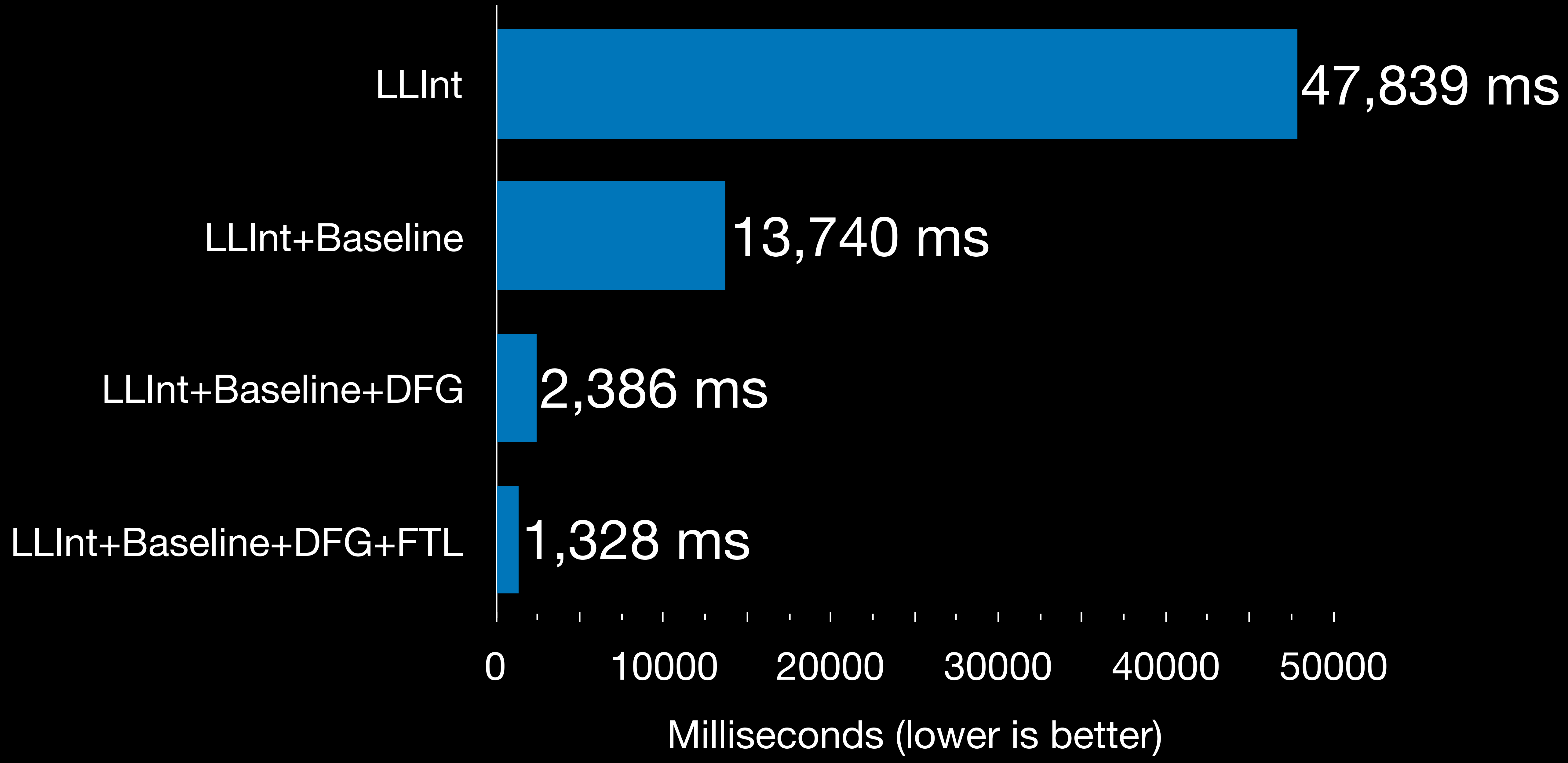
JetStream 2 “gaussian-blur”

on my computer one day



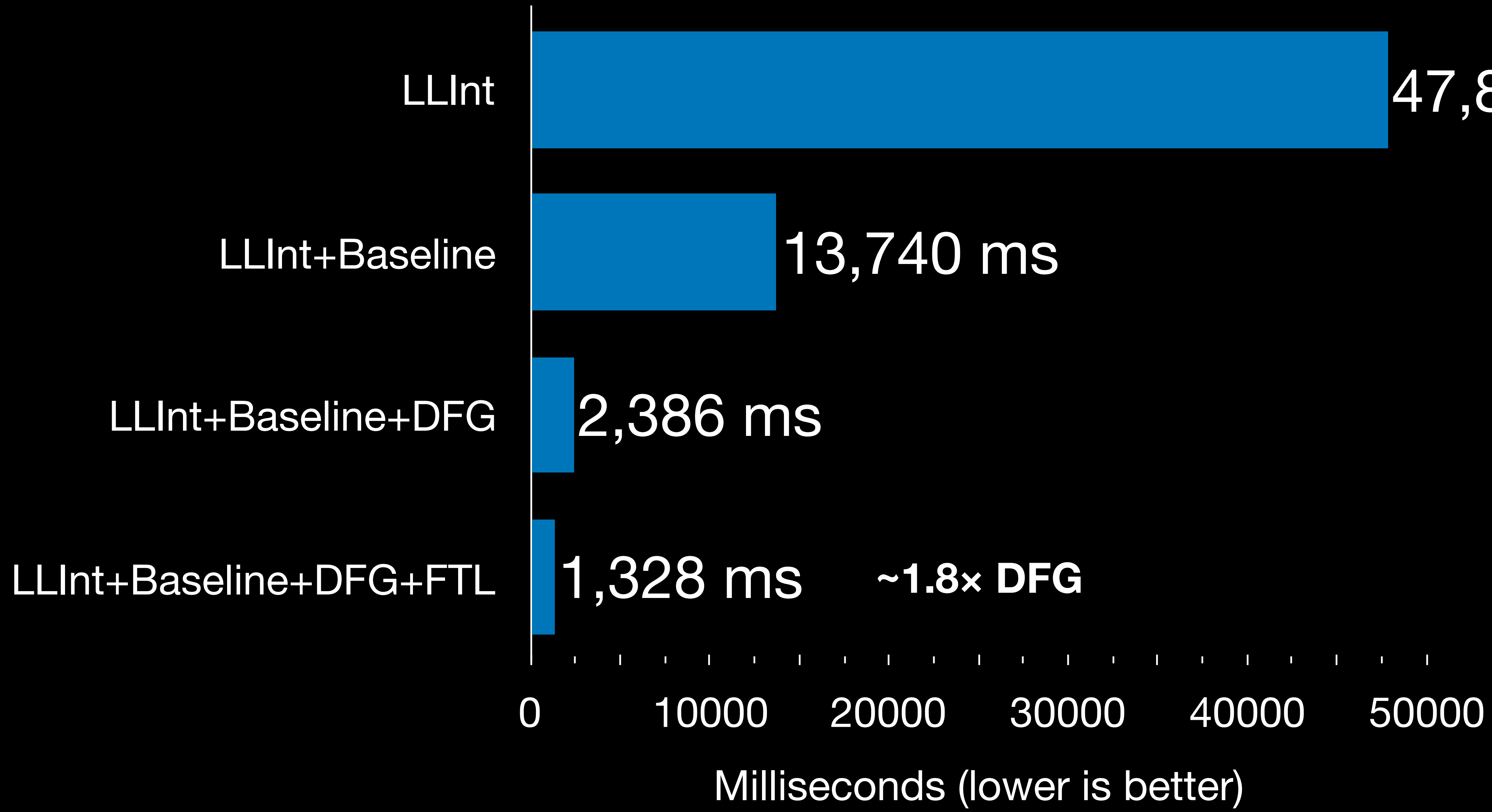
JetStream 2 "raytrace"

on my computer one day

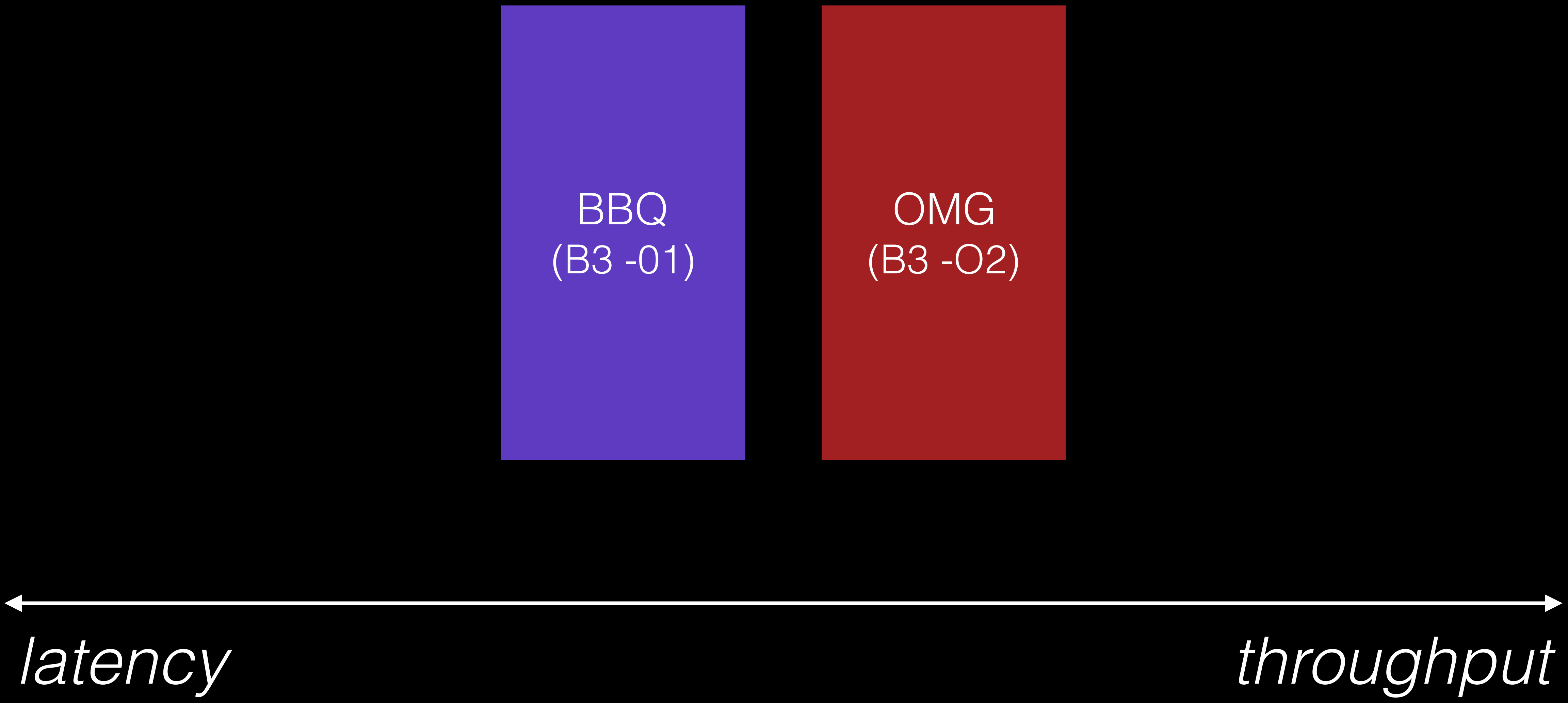


JetStream 2 “raytrace”

on my computer one day



Two WebAssembly Tiers



BBQ
(B3 -01)

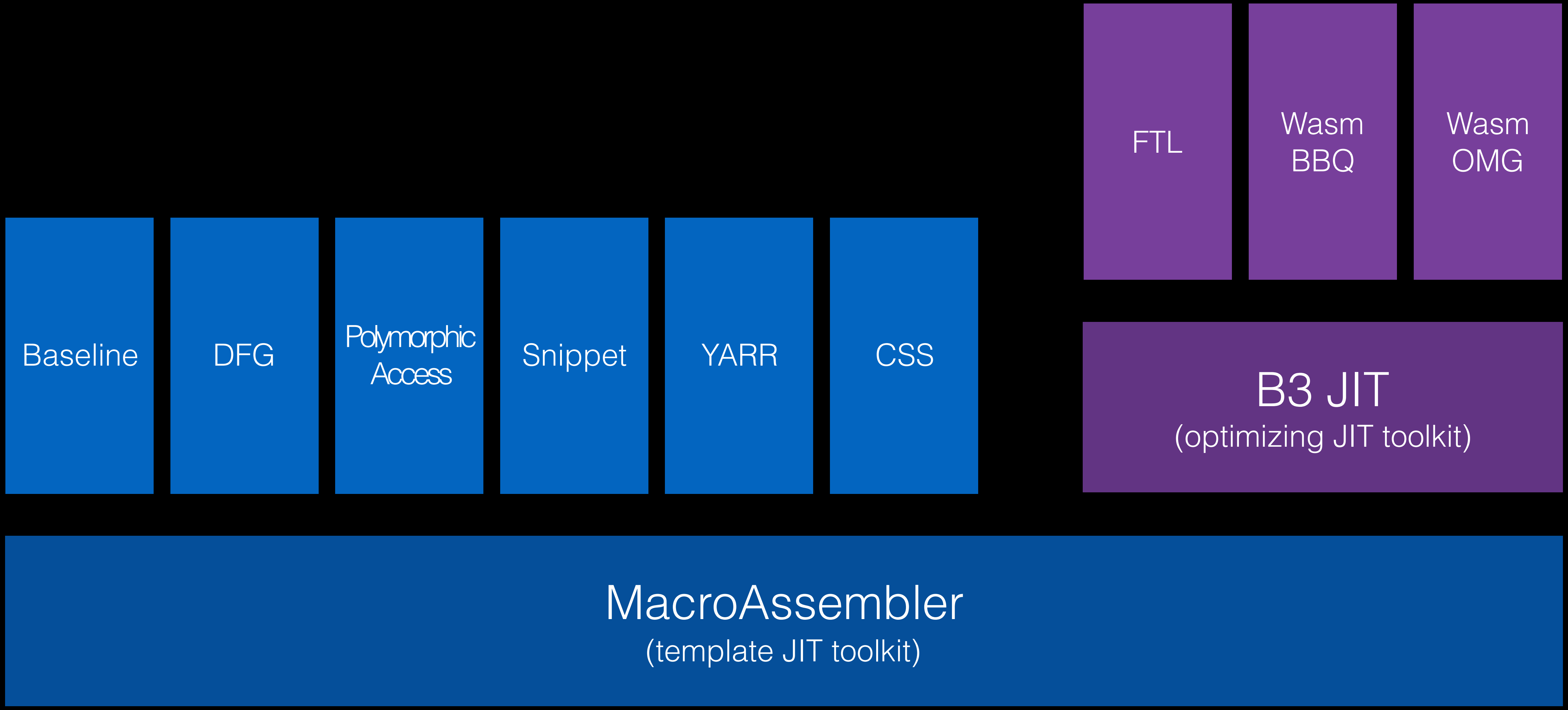
OMG
(B3 -O2)

latency

throughput

~9 JIT compilers

JavaScript execution engines:	LLInt (interpreter)	Baseline (template JIT)	DFG (template JIT)	FTL (B3 JIT)	Polymorphic Access (template JIT)	Snippet (template JIT)
WebAssembly execution engines:	Wasm BBQ (B3 JIT)	Wasm OMG (B3 JIT)				
Bonus JITs:	YARR (template regexp JIT)	CSS (template JIT)				



FTL

Wasm BBQ

Wasm OMG

Baseline

DFG

Polymorphic Access

Snippet

YARR

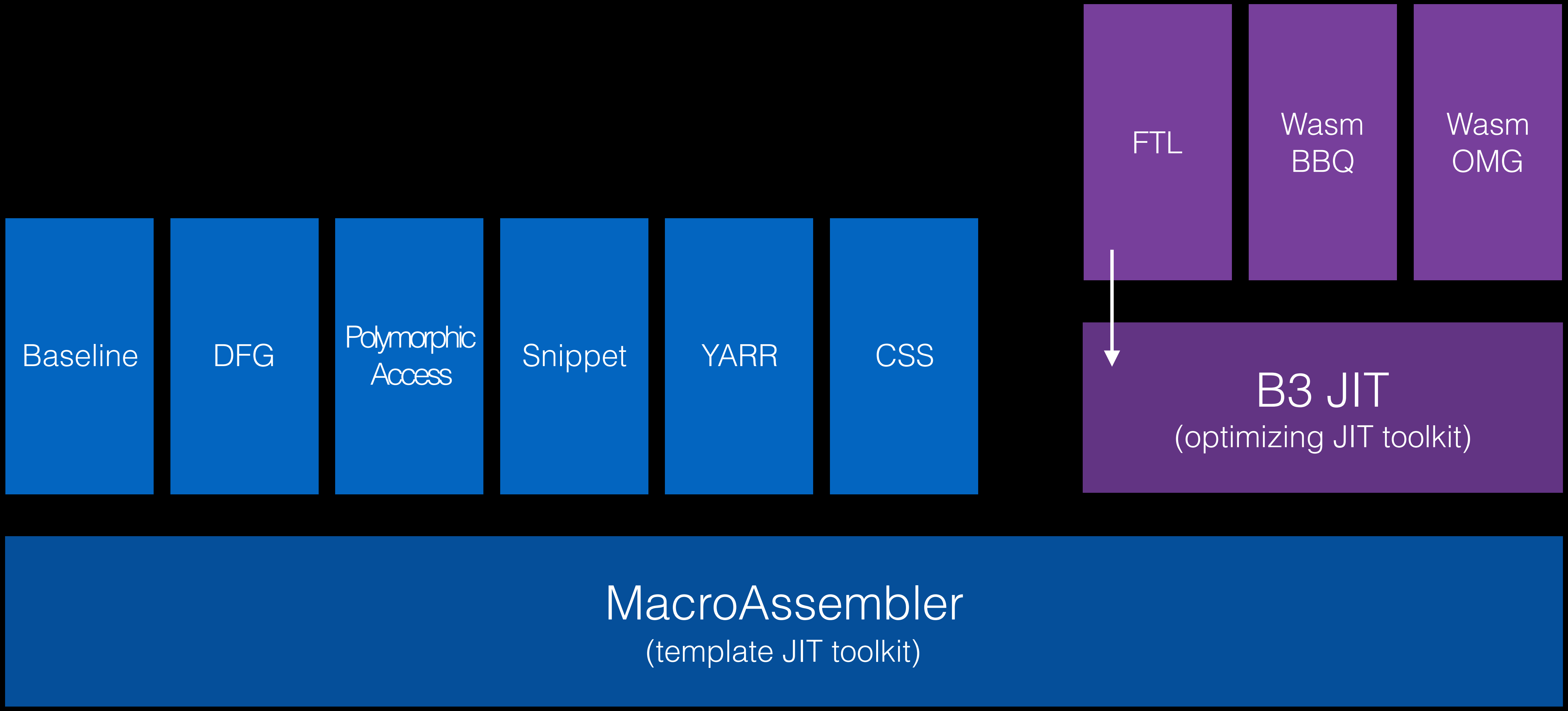
CSS

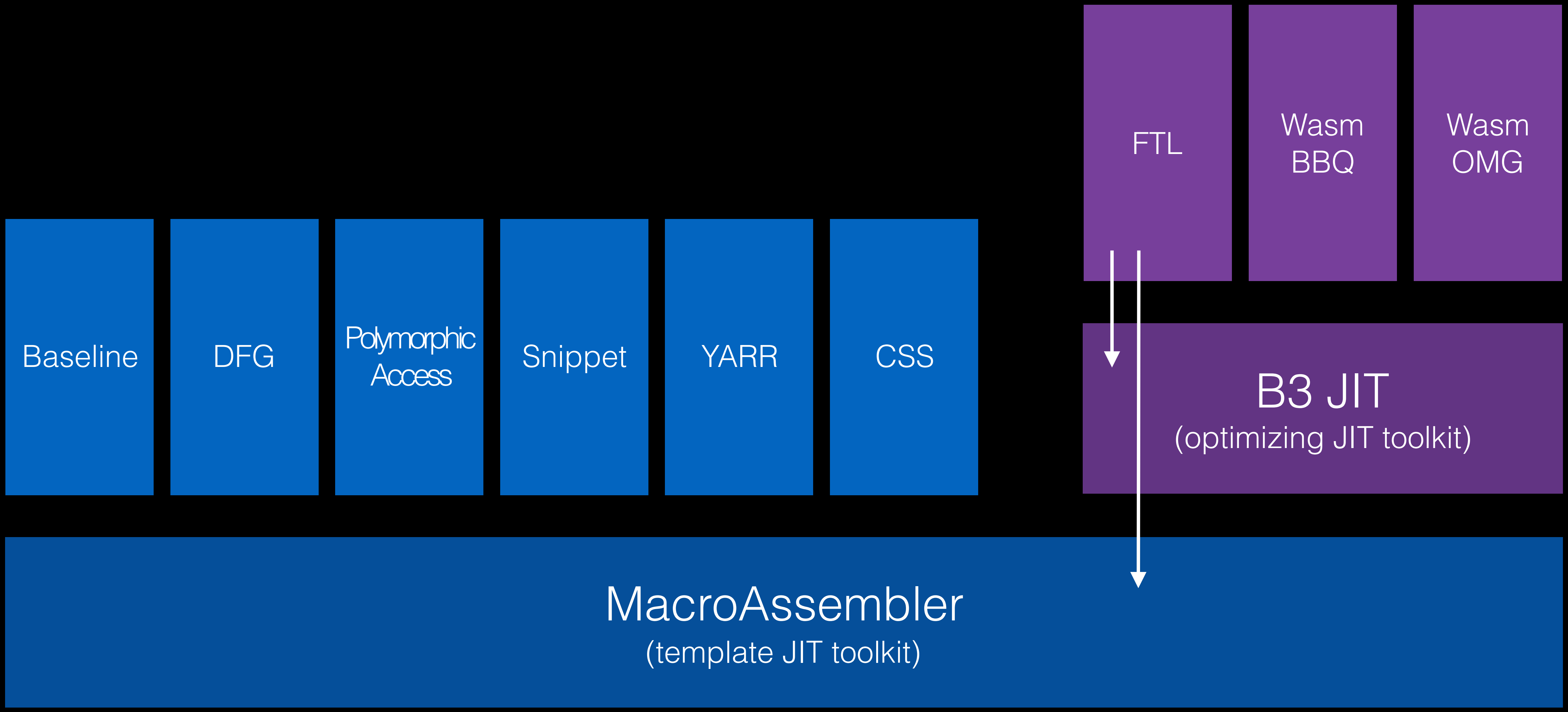
B3 JIT

(optimizing JIT toolkit)

MacroAssembler

(template JIT toolkit)


FTL
Wasm BBQ
Wasm OMG
Baseline
DFG
Polymorphic Access
Snippet
YARR
CSS
B3 JIT
(optimizing JIT toolkit)
MacroAssembler
(template JIT toolkit)



Baseline

DFG

Polymorphic Access

Snippet

YARR

CSS

FTL

Wasm BBQ

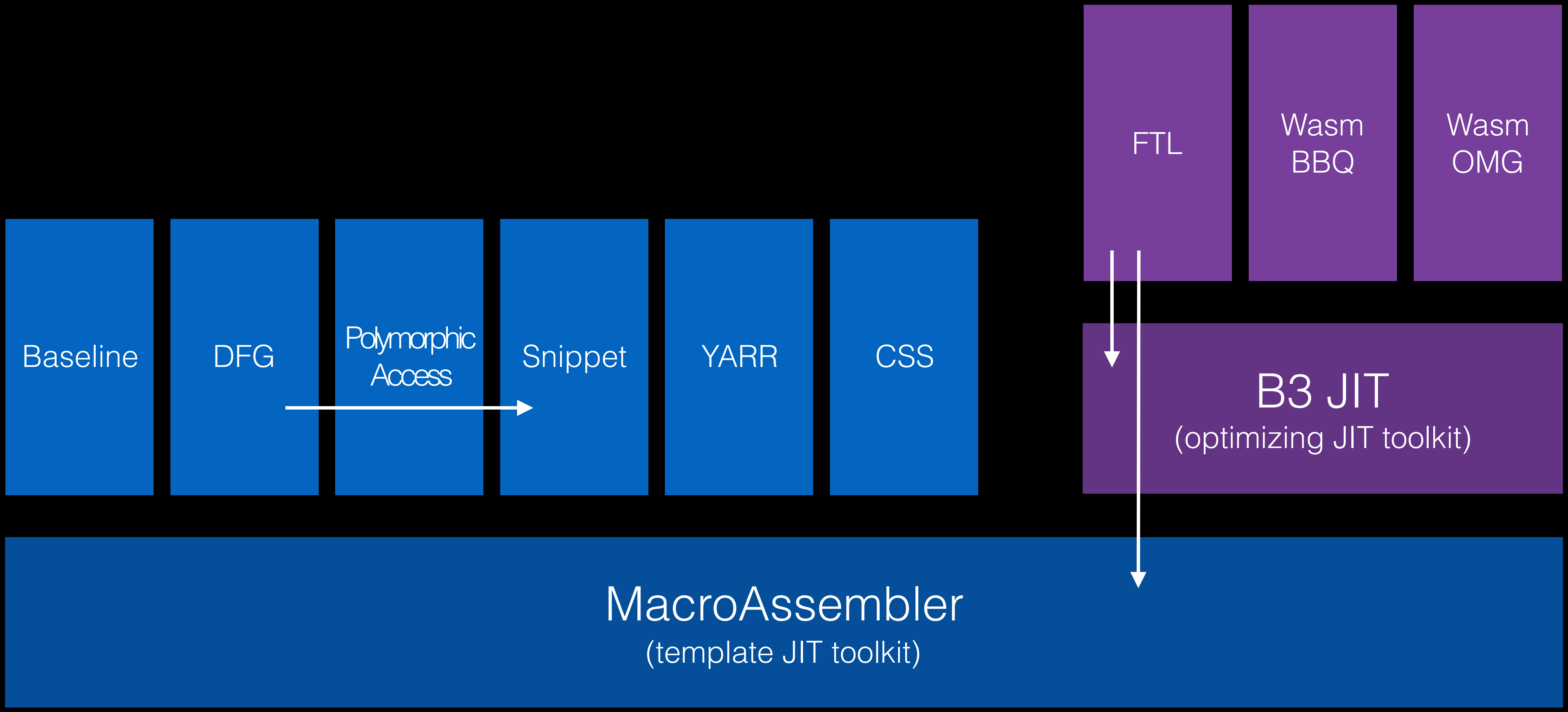
Wasm OMG

B3 JIT

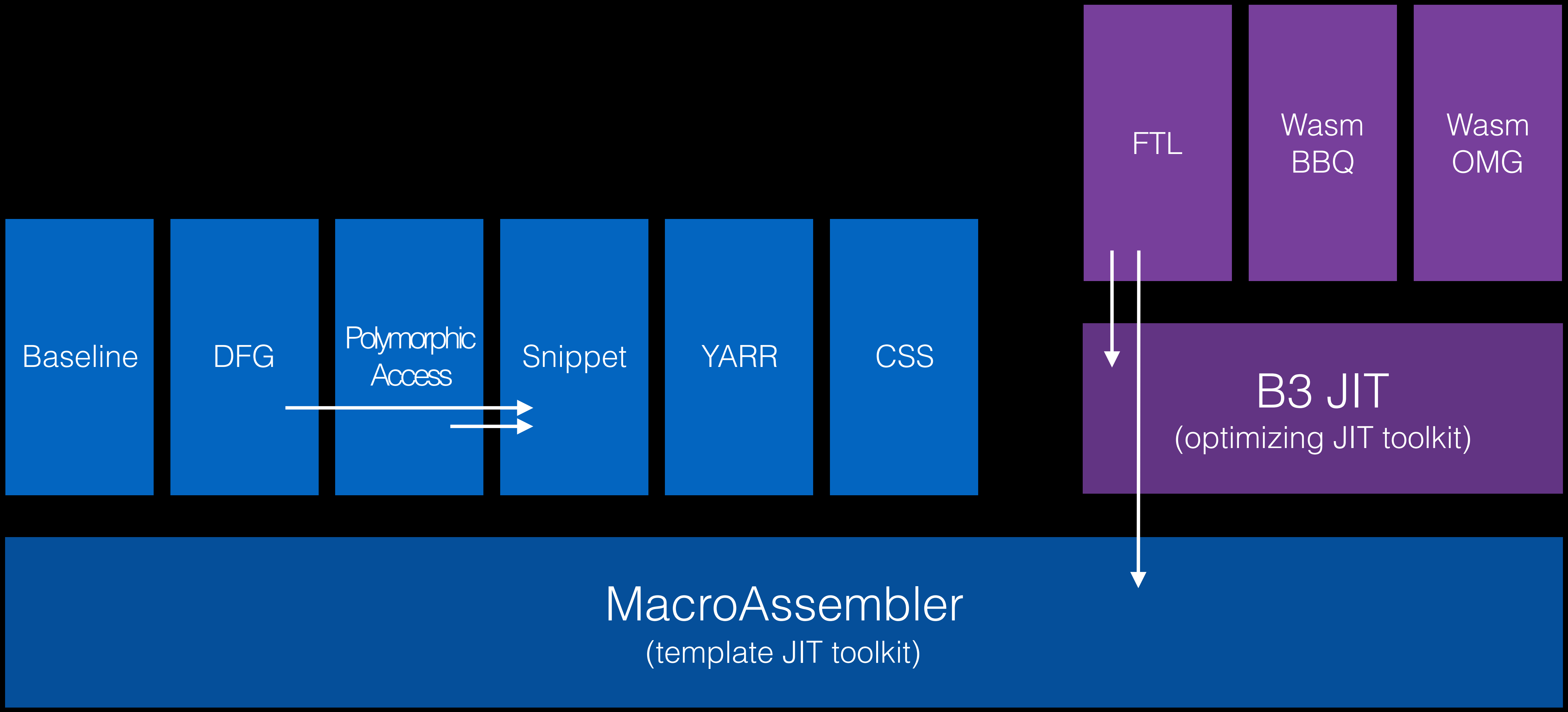
(optimizing JIT toolkit)

MacroAssembler

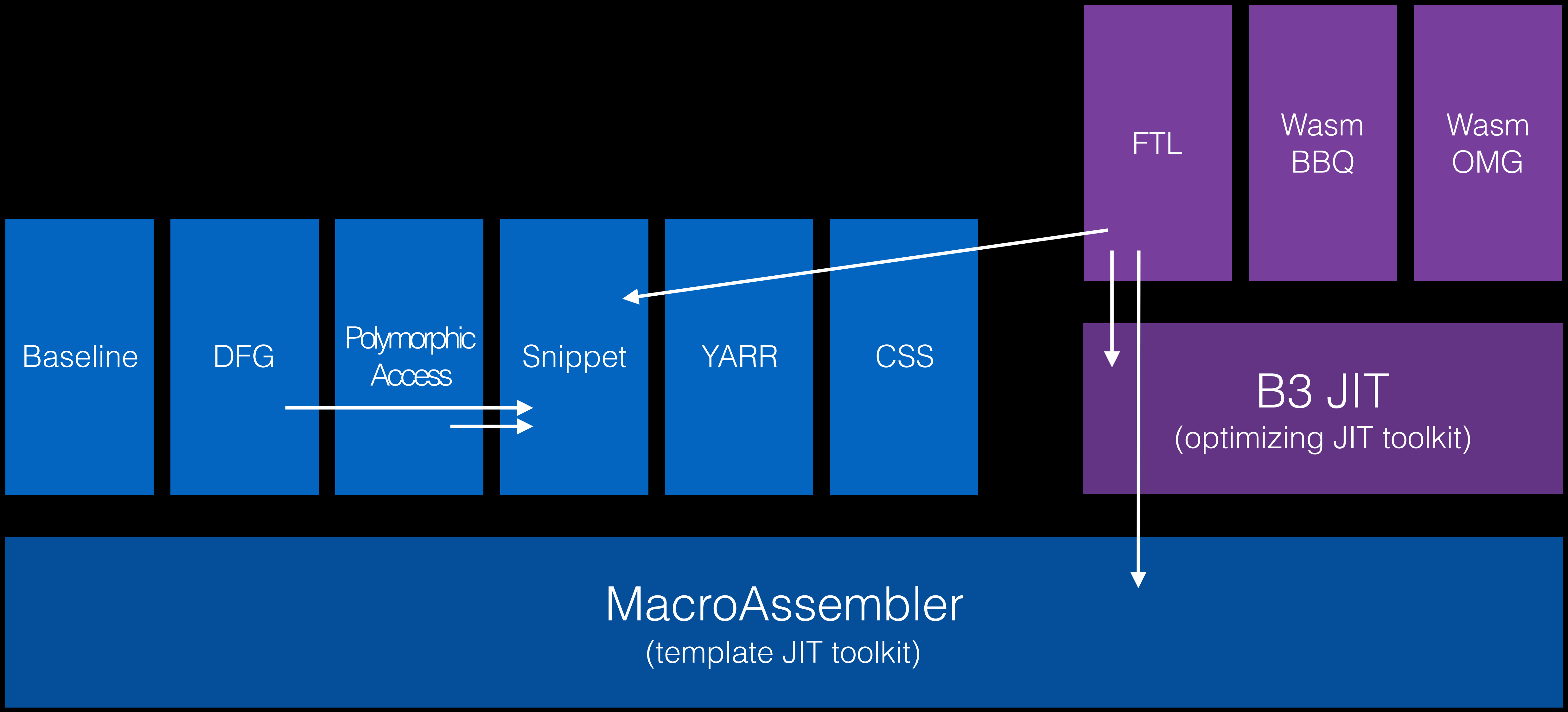
(template JIT toolkit)



FTL
Wasm BBQ
Wasm OMG
Baseline
DFG
Polymorphic Access
Snippet
YARR
CSS
B3 JIT
(optimizing JIT toolkit)
MacroAssembler
(template JIT toolkit)



FTL
Wasm BBQ
Wasm OMG
Baseline
DFG
Polymorphic Access
Snippet
YARR
CSS
B3 JIT
(optimizing JIT toolkit)
MacroAssembler
(template JIT toolkit)


FTL
Wasm BBQ
Wasm OMG
Baseline
DFG
Polymorphic Access
Snippet
YARR
CSS
B3 JIT
(optimizing JIT toolkit)
MacroAssembler
(template JIT toolkit)

JIT-friendly VM

- Conservative-on-the-stack GC
- Consistent, mostly C-like ABI

Agenda

- High Level Overview
- Template JITing
- Optimized JITing
  - DFG
  - FTL
  - BBQ
  - OMG

Template JIT

- Goal: decent throughput with low latency.

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

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

Template JIT

- Portable assembly meta-programming.

```
[   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)
```

```
[   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)
```

```
[   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, %rax
      0x2f8084601a79: jb 0x2f8084601af2
      0x2f8084601a7f: mov %esi, %eax
      0x2f8084601a81: add %edx, %eax
      0x2f8084601a83: jo 0x2f8084601af2
      0x2f8084601a89: or %r14, %rax
      0x2f8084601a8c: mov %rax, -0x38(%rbp)
```

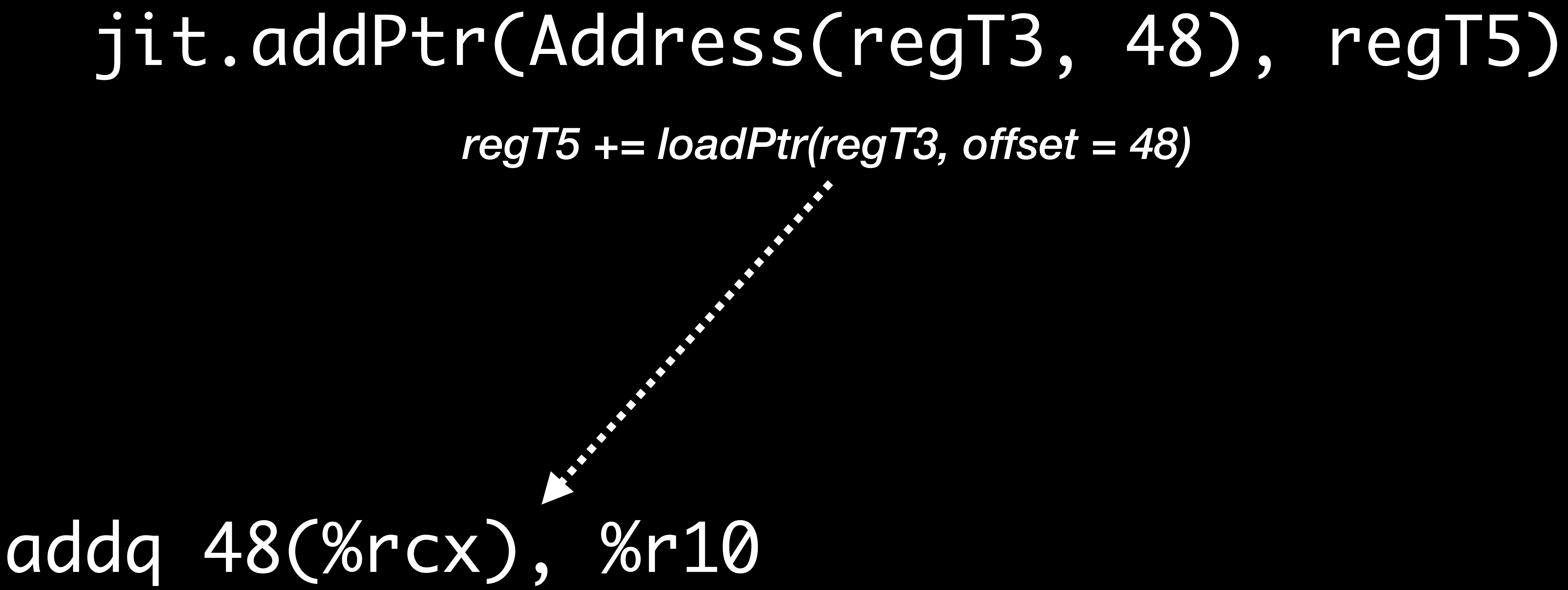
```
if (!m_leftOperand.isConstInt32())
    state.slowPathJumps.append(
        jit.branchIfNotInt32(m_left));
```

```
[   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)
```

```
if (!m_leftOperand.isConstInt32())
    state.slowPathJumps.append(
        jit.branchIfNotInt32(m_left));

jit.branchAdd32(
    Overflow, var.payloadGPR(),
    Imm32(constValue), scratch);
```

```
jit.addPtr(Address(regT3, 48), regT5)
```

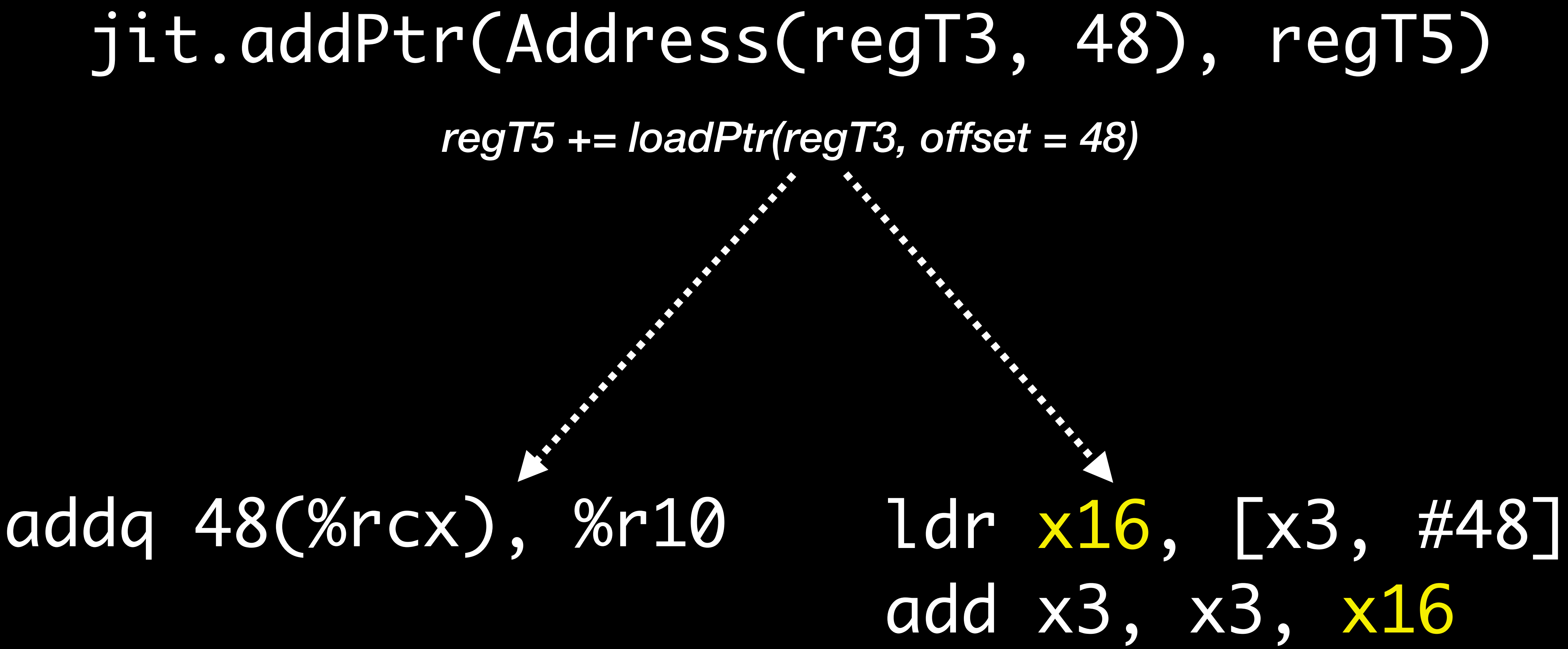
regT5 += loadPtr(regT3, offset = 48)



```
jit.addPtr(Address(regT3, 48), regT5)
```

regT5 += loadPtr(regT3, offset = 48)

```
addq 48(%rcx), %r10
```



```
jit.addPtr(Address(regT3, 48), regT5)
```

regT5 += loadPtr(regT3, offset = 48)

```
addq 48(%rcx), %r10
```

```
ldr x16, [x3, #48]
add x3, x3, x16
```

```
jit.emitFunctionPrologue()
jit.setupArguments<decltype(
    operationReallocateButterflyToGrowPropertyStorage)>(
        baseGPR,
        CCallHelpers::TrustedImm32(newSize / sizeof(JSValue)));

CCallHelpers::Call operationCall = jit.call(OperationPtrTag);

jit.addLinkTask([=] (LinkBuffer& linkBuffer) {
    linkBuffer.link(
        operationCall,
        FunctionPtr<OperationPtrTag>(
            operationReallocateButterflyToGrowPropertyStorage));
});
```

```
jit.setupArguments<decltype(
    operationReallocateButterflyToGrowPropertyStorage)>(
        baseGPR,
        CCallHelpers::TrustedImm32(newSize / sizeof(JSValue)));

CCallHelpers::Call operationCall = jit.call(OperationPtrTag);

jit.addLinkTask([=] (LinkBuffer& linkBuffer) {
    linkBuffer.link(
        operationCall,
        FunctionPtr<OperationPtrTag>(
            operationReallocateButterflyToGrowPropertyStorage));
});
```

```
jit.setupArguments<decltype(
    operationReallocateButterflyToGrowPropertyStorage)>(
        baseGPR,
        CCallHelpers::TrustedImm32(newSize / sizeof(JSValue)));

CCallHelpers::Call operationCall = jit.call(OperationPtrTag);

jit.addLinkTask([=] (LinkBuffer& linkBuffer) {
    linkBuffer.link(
        operationCall,
        FunctionPtr<OperationPtrTag>(
            operationReallocateButterflyToGrowPropertyStorage));
});
```

```
jit.setupArguments<decltype(
    operationReallocateButterflyToGrowPropertyStorage)>(
        baseGPR,
        CCallHelpers::TrustedImm32(newSize / sizeof(JSValue)));

CCallHelpers::Call operationCall = jit.call(OperationPtrTag);

jit.addLinkTask([=] (LinkBuffer& linkBuffer) {
    linkBuffer.link(
        operationCall,
        FunctionPtr<OperationPtrTag>(
            operationReallocateButterflyToGrowPropertyStorage));
});
```

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

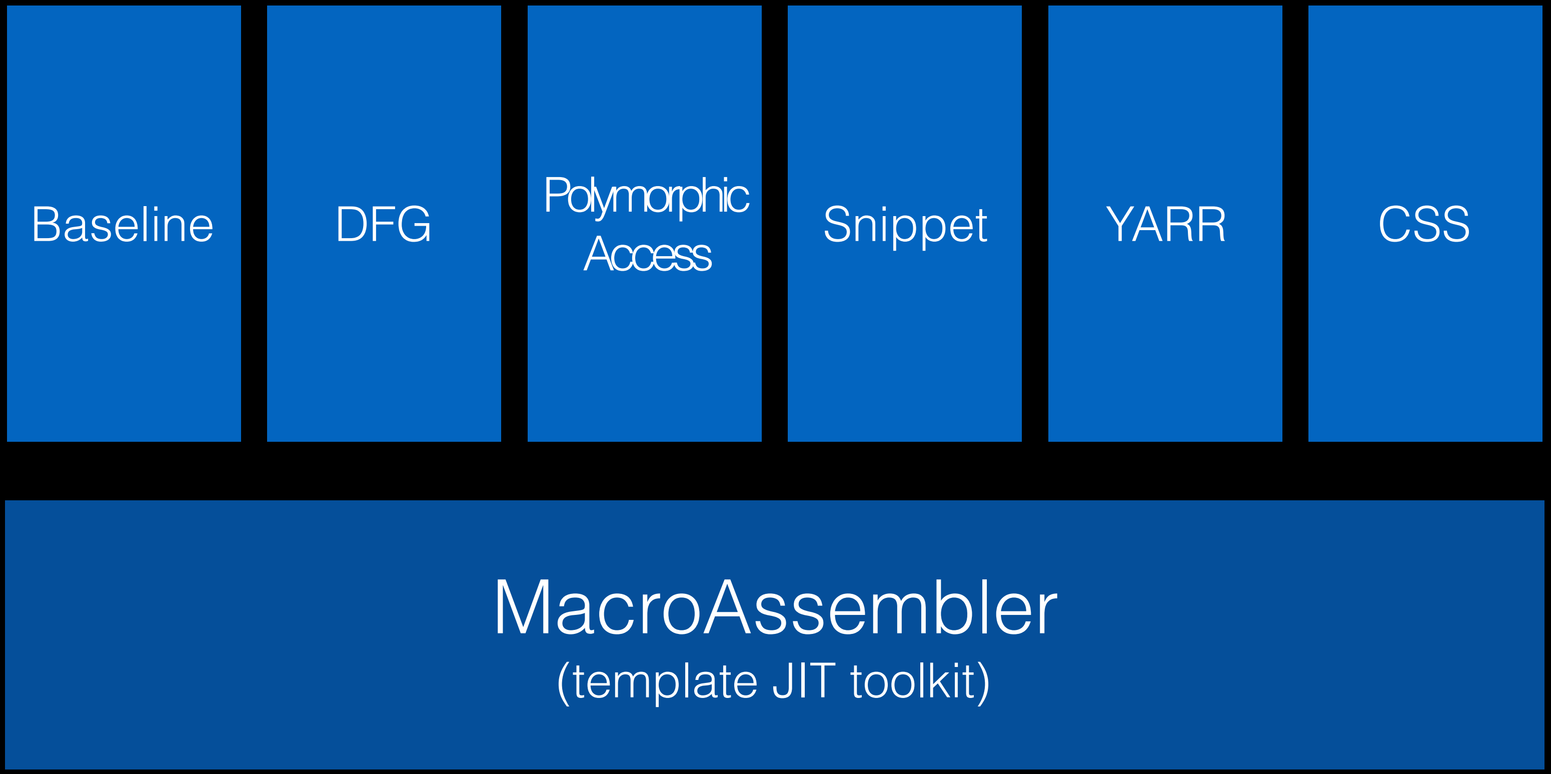
```
0x46f8c30b9b0: mov 0x30(%rbp), %rax
0x46f8c30b9b4: test %rax, %r15
0x46f8c30b9b7: jnz 0x46f8c30ba2c
0x46f8c30b9bd: jmp 0x46f8c30ba2c
0x46f8c30b9c2: o16 nop %cs:0x200(%rax,%rax)
0x46f8c30b9d1: nop (%rax)
0x46f8c30b9d4: mov %rax, -0x38(%rbp)
```

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)
```



Baseline

DFG

Polymorphic Access

Snippet

YARR

CSS

MacroAssembler

(template JIT toolkit)

Agenda

- High Level Overview
- Template JITing
- Optimized JITing
  - DFG
  - FTL
  - BBQ
  - OMG

Agenda

- High Level Overview
- Template JITing
- Optimized JITing
  - DFG
  - FTL
  - BBQ
  - OMG

DFG IR

Source

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

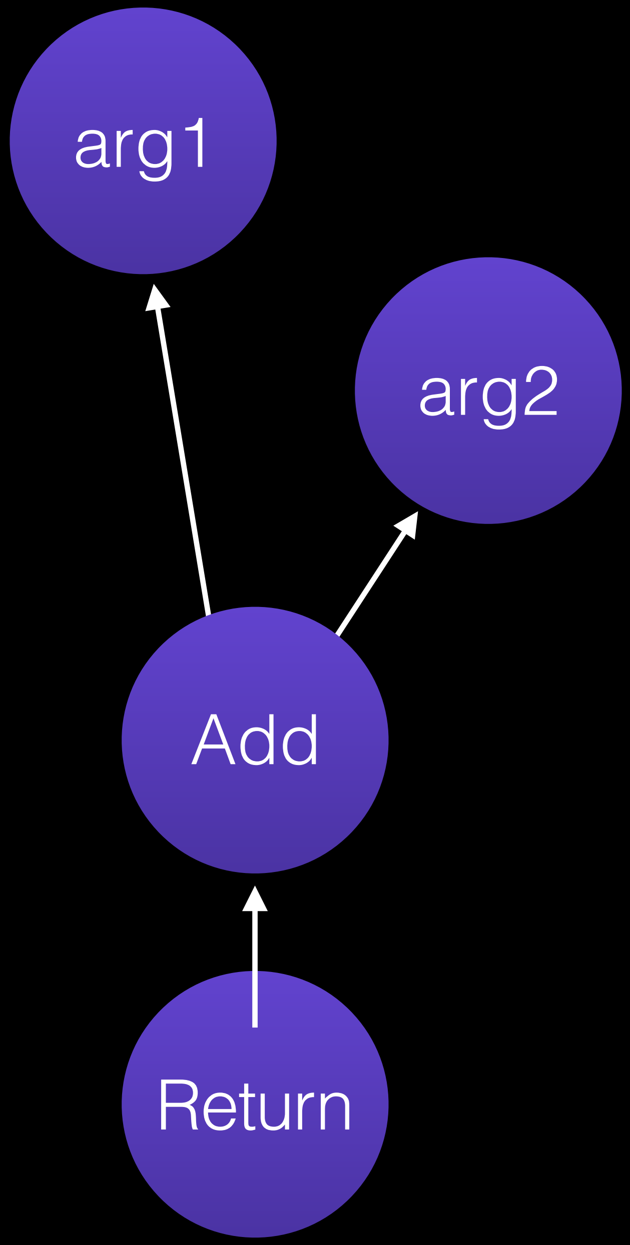
Bytecode

```
[   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)
```

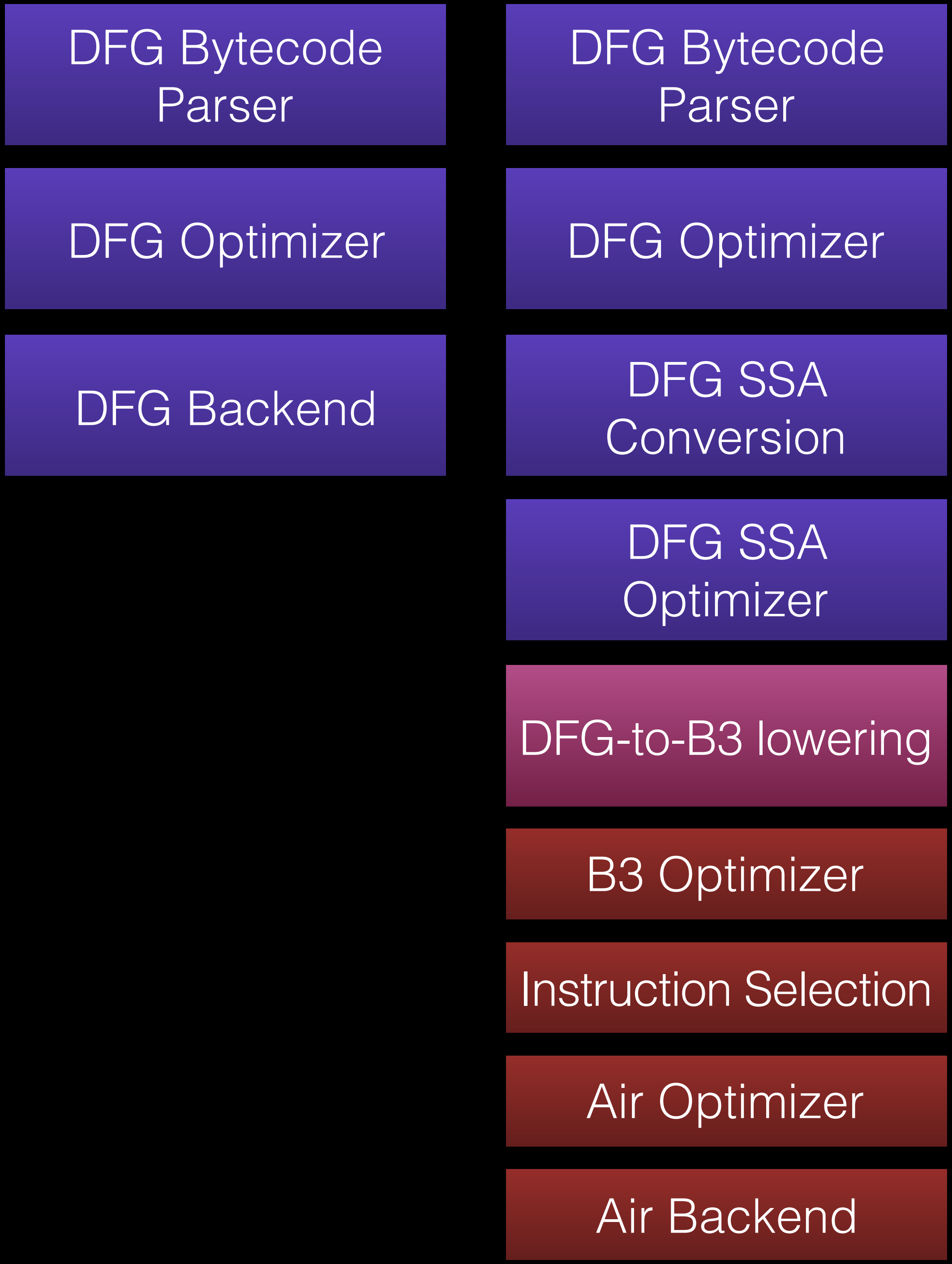


DFG

Fast JIT

FTL

Powerful JIT



DFG Bytecode Parser

DFG Optimizer

DFG Backend

DFG Bytecode Parser

DFG Optimizer

DFG SSA Conversion

DFG SSA Optimizer

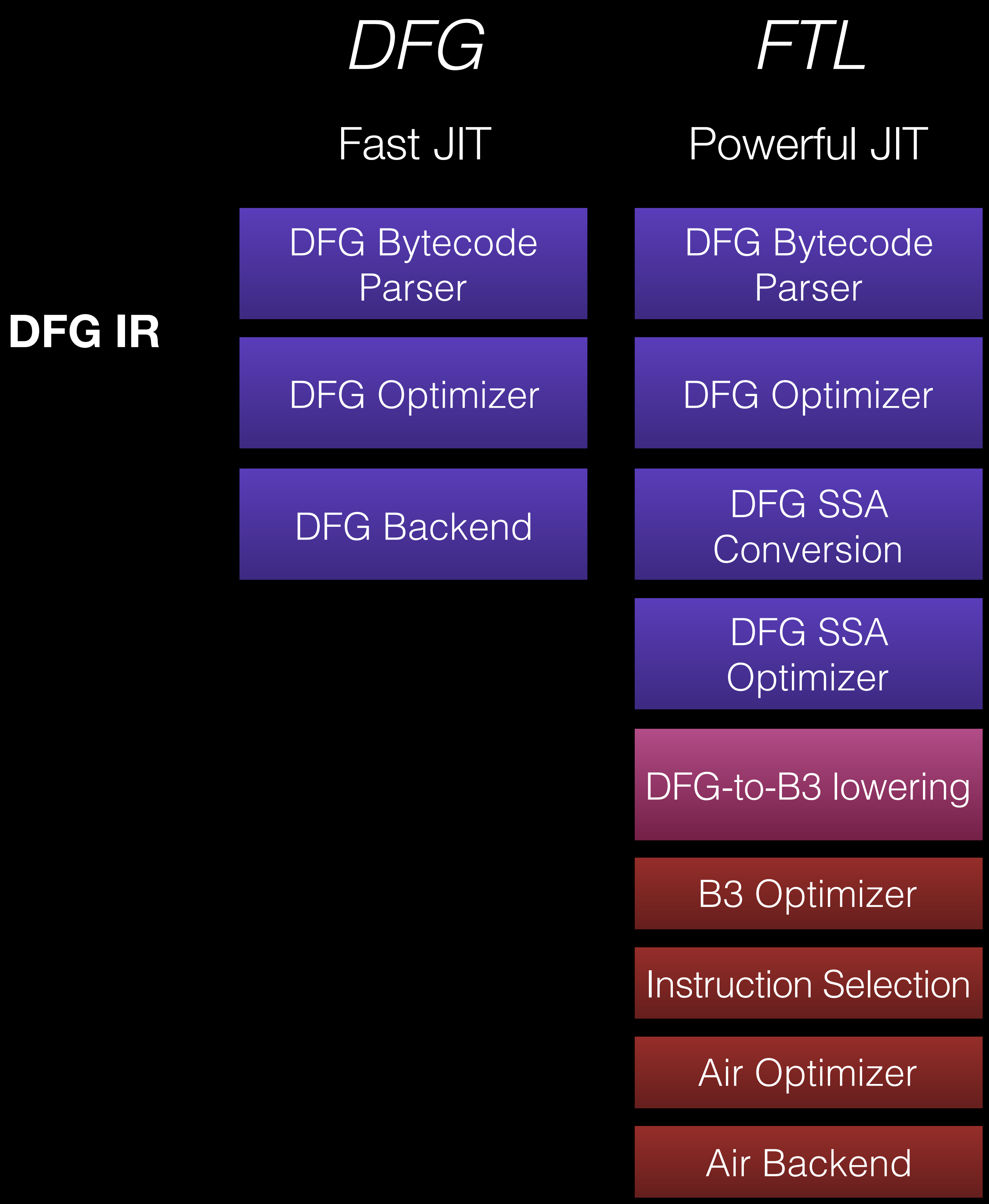
DFG-to-B3 lowering

B3 Optimizer

Instruction Selection

Air Optimizer

Air Backend



DFG

Fast JIT

FTL

Powerful JIT

DFG IR

DFG Bytecode Parser

DFG Optimizer

DFG Backend

DFG Bytecode Parser

DFG Optimizer

DFG SSA Conversion

DFG SSA Optimizer

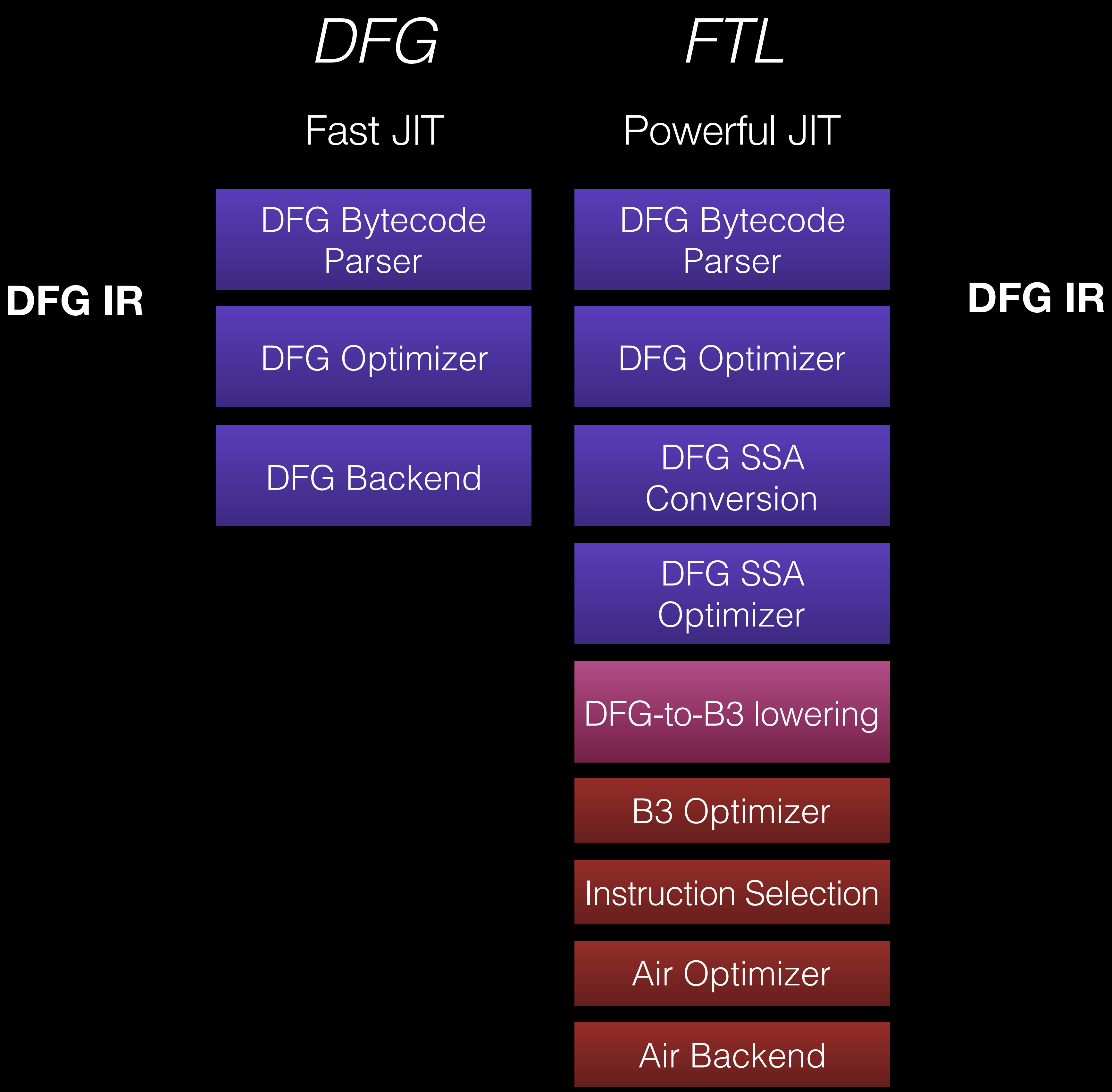
DFG-to-B3 lowering

B3 Optimizer

Instruction Selection

Air Optimizer

Air Backend



DFG

Fast JIT

FTL

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

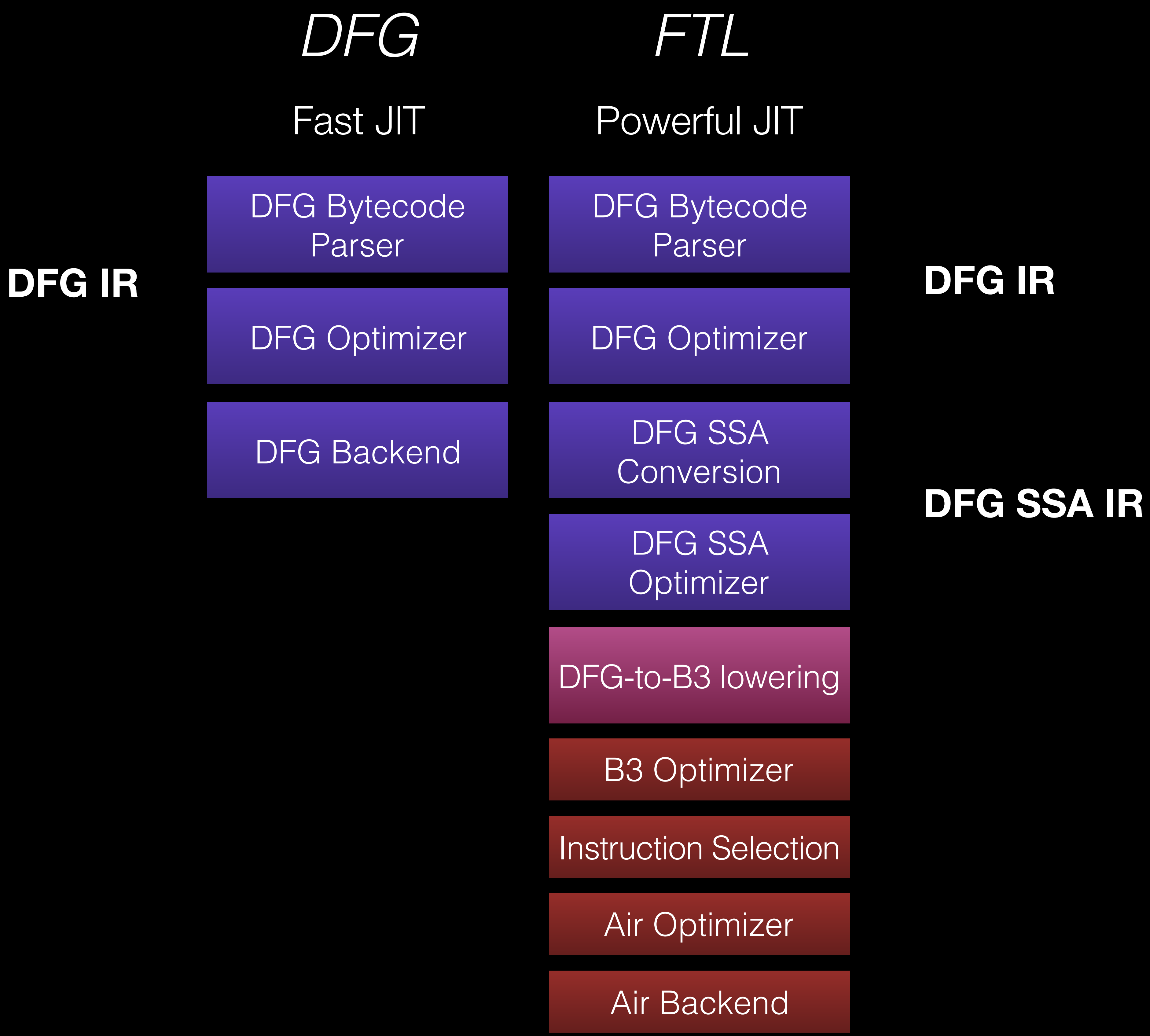
B3 Optimizer

Instruction Selection

Air Optimizer

Air Backend

DFG IR



DFG

Fast JIT

DFG Bytecode Parser

DFG Optimizer

DFG Backend

FTL

Powerful JIT

DFG Bytecode Parser

DFG Optimizer

DFG SSA Conversion

DFG SSA Optimizer

DFG-to-B3 lowering

B3 Optimizer

Instruction Selection

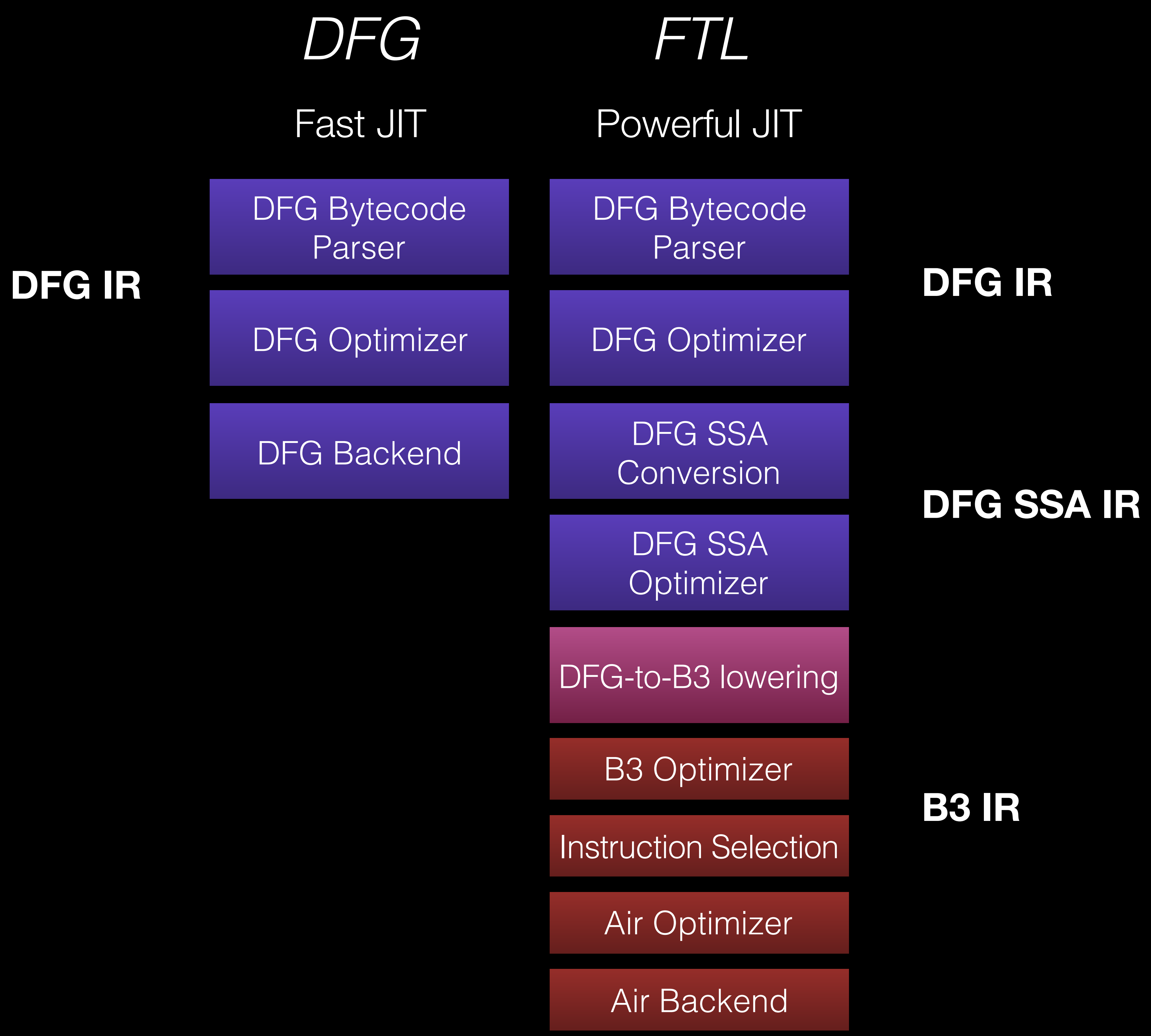
Air Optimizer

Air Backend

DFG IR

DFG IR

DFG SSA IR



DFG

Fast JIT

- DFG Bytecode Parser
- DFG Optimizer
- DFG Backend

DFG IR

FTL

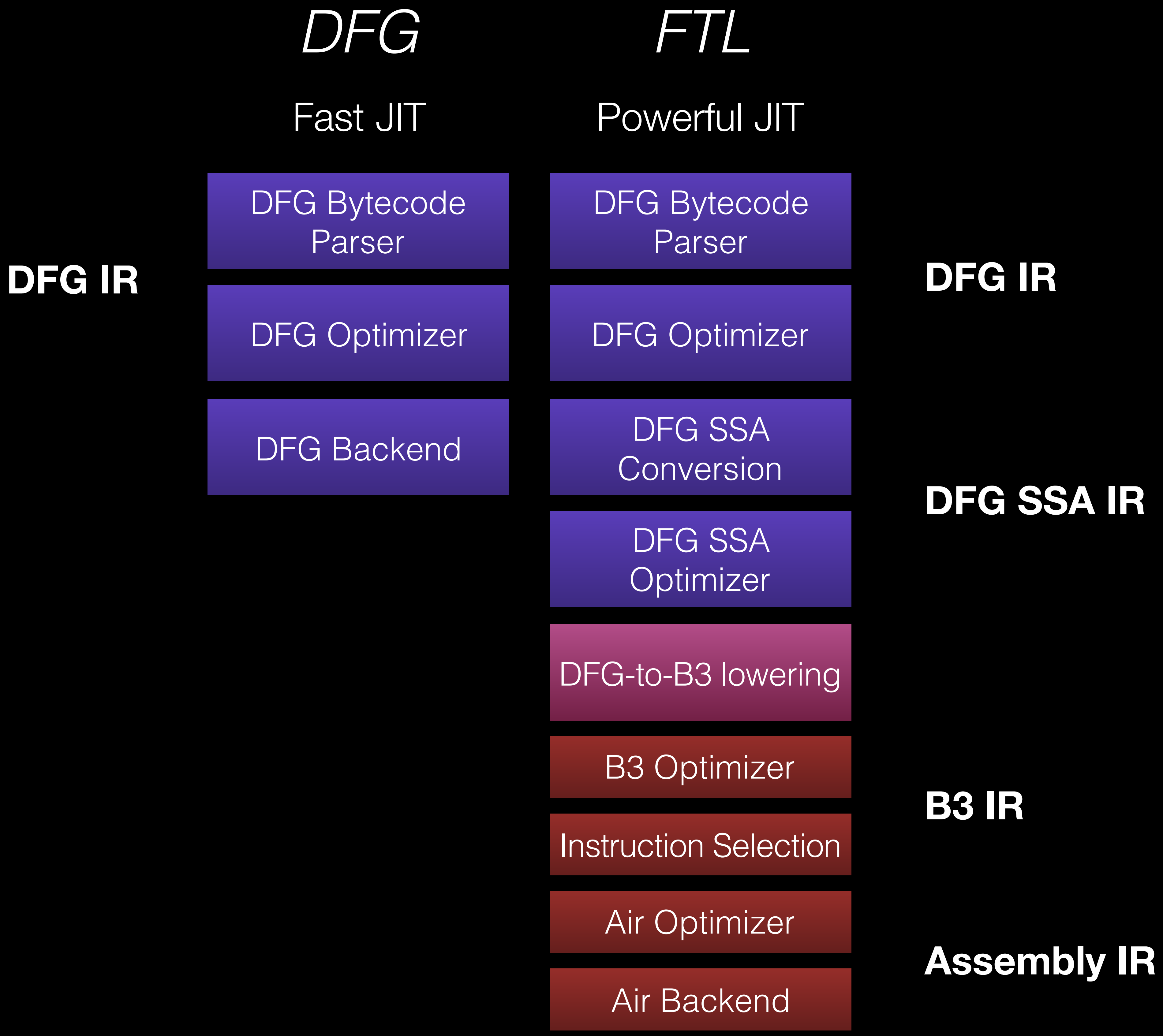
Powerful JIT

- 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

Fast JIT

FTL

Powerful JIT

DFG IR

DFG Bytecode Parser

DFG Optimizer

DFG Backend

DFG Bytecode Parser

DFG Optimizer

DFG IR

DFG SSA Conversion

DFG SSA Optimizer

DFG SSA IR

DFG-to-B3 lowering

B3 Optimizer

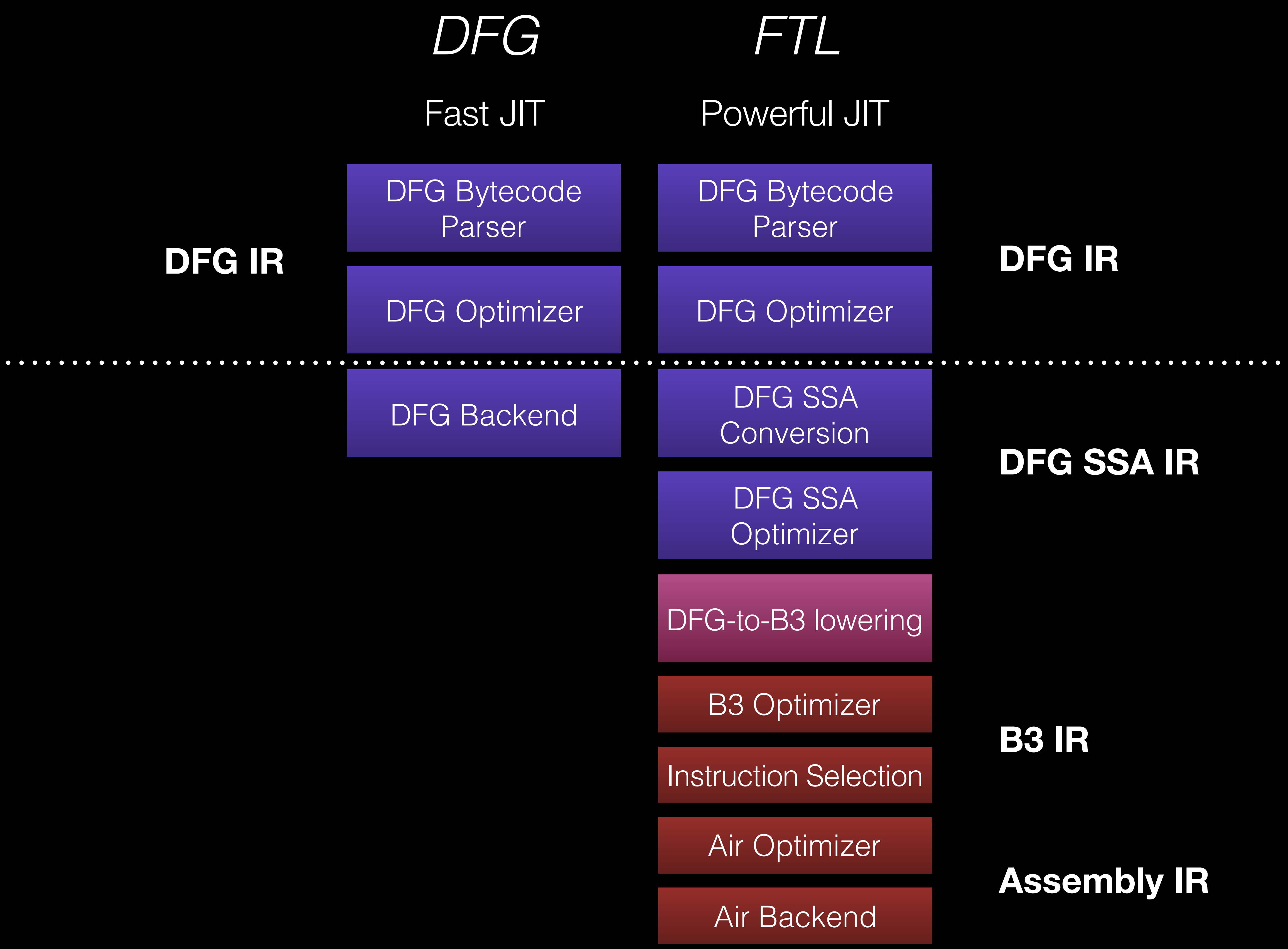
Instruction Selection

B3 IR

Air Optimizer

Air Backend

Assembly IR



DFG

Fast JIT

FTL

Powerful JIT

DFG IR

DFG Bytecode Parser

DFG Optimizer

DFG Backend

DFG Bytecode Parser

DFG Optimizer

DFG IR

DFG SSA Conversion

DFG SSA Optimizer

DFG SSA IR

DFG-to-B3 lowering

B3 Optimizer

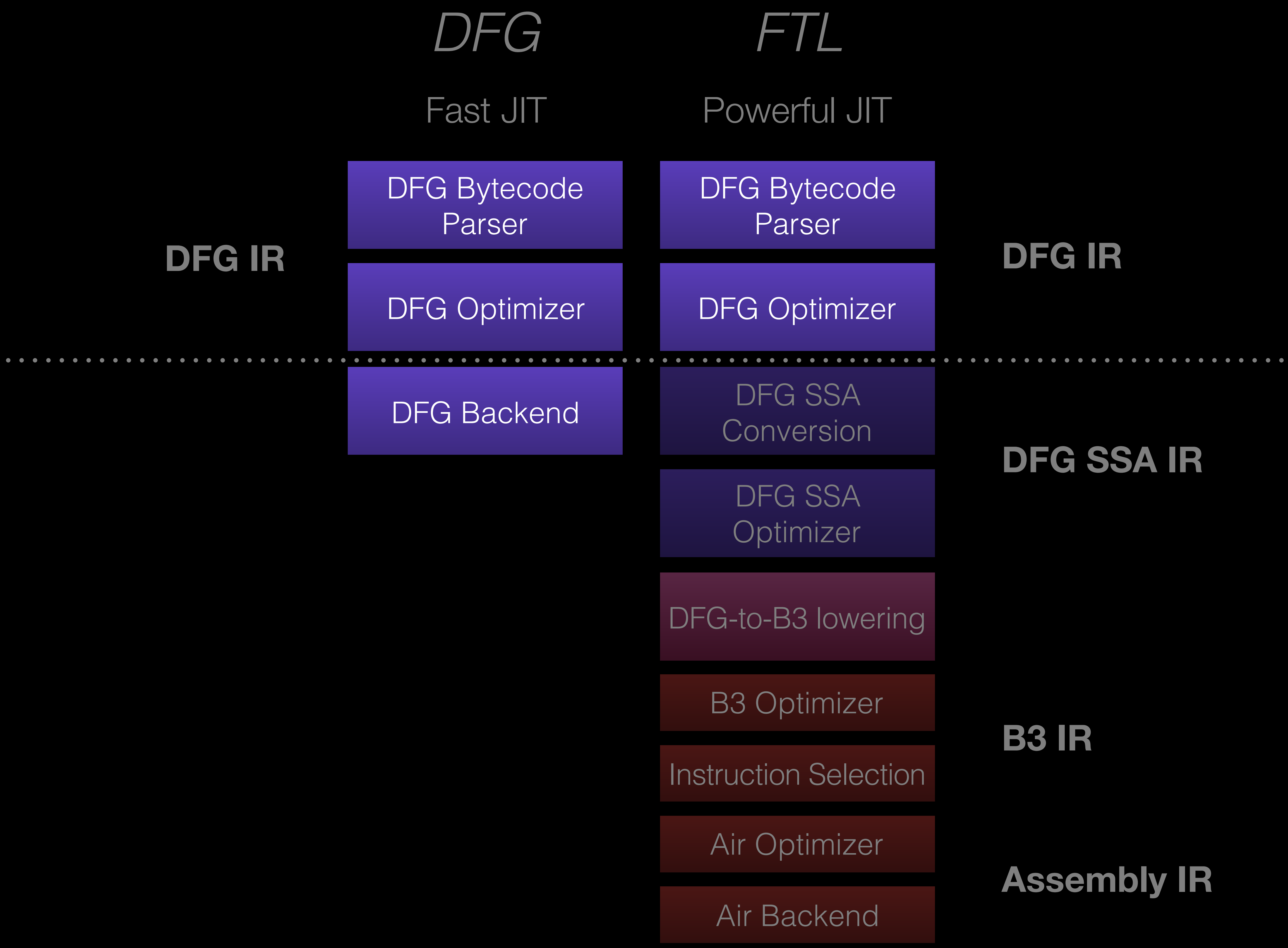
Instruction Selection

B3 IR

Air Optimizer

Air Backend

Assembly IR



DFG

Fast JIT

DFG Bytecode Parser

DFG Optimizer

DFG Backend

FTL

Powerful JIT

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 IR

DFG SSA IR

B3 IR

Assembly IR

DFG Goal

Remove lots of type checks quickly.

DFG Goals

- Speculation
- Static Analysis
- Fast Compilation

DFG Goals

- Speculation
- Static Analysis
- Fast Compilation

DFG IR

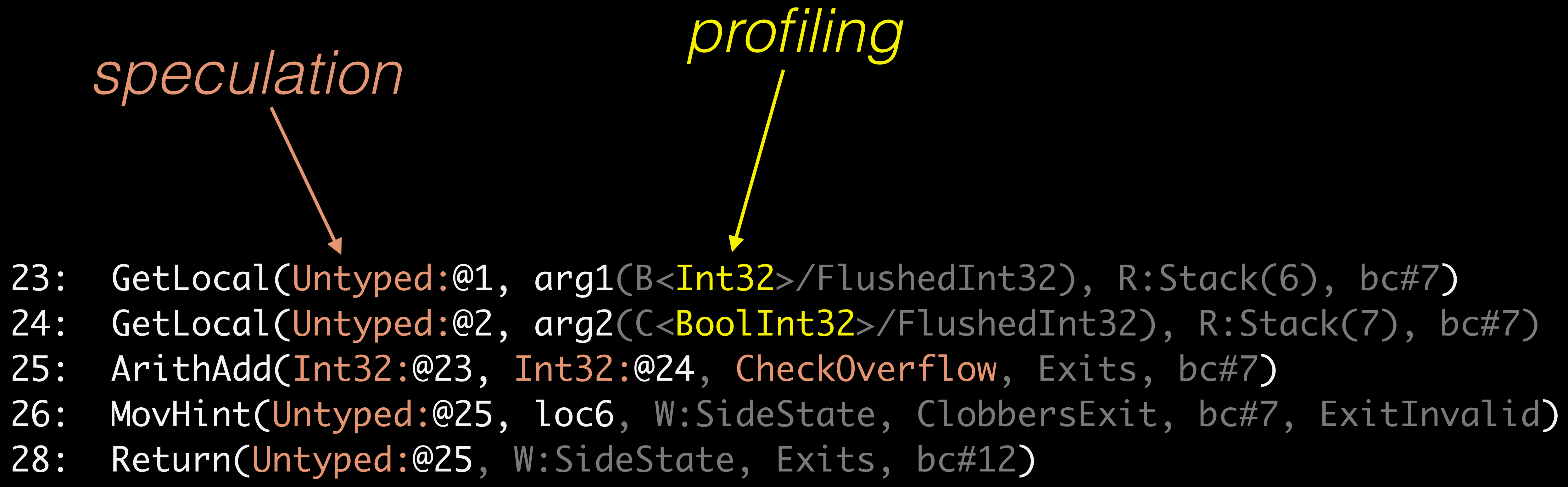
```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
28: 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



```
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

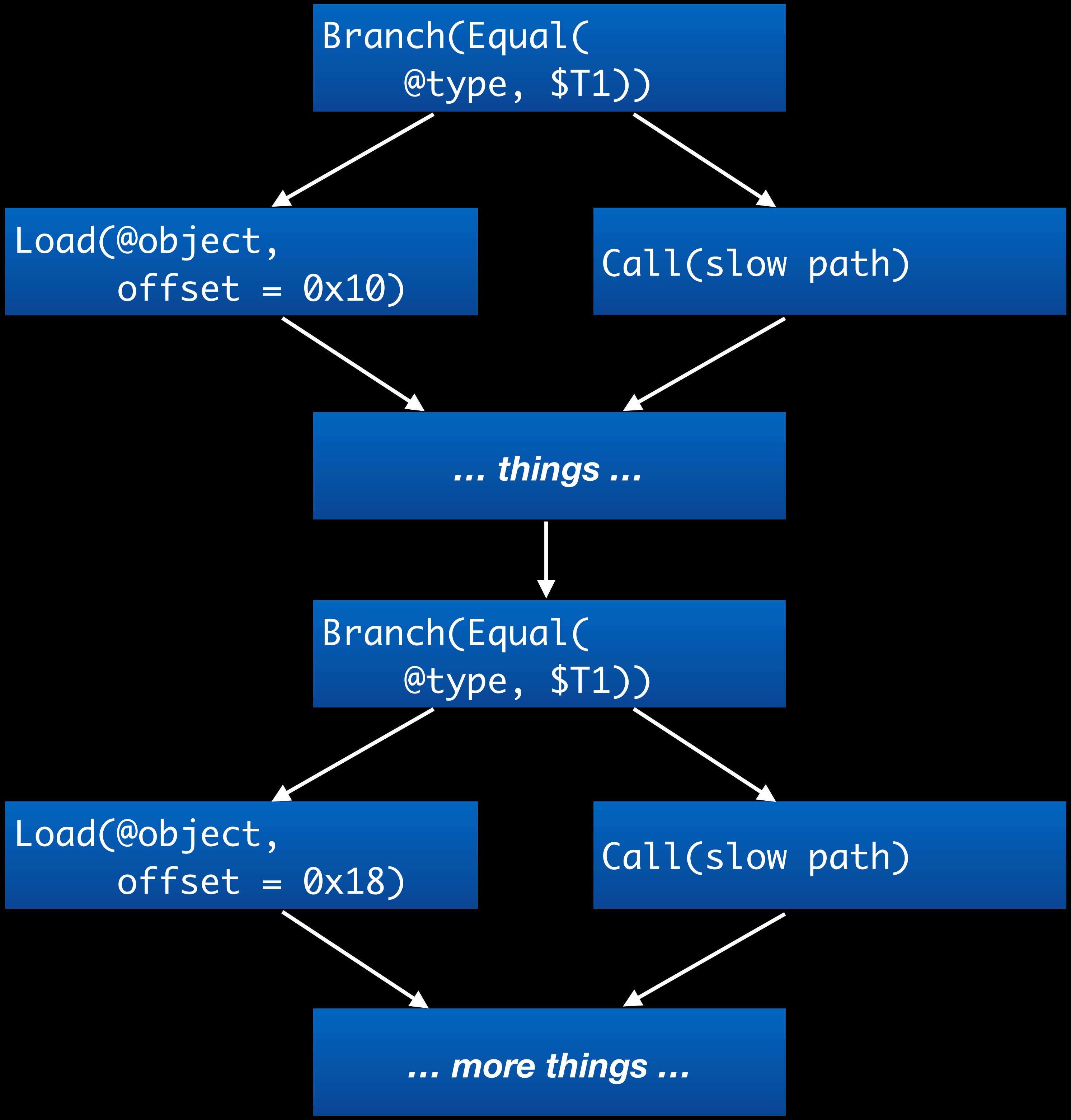
speculation

```
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

Why OSR?

Type Checks w/o OSR



```
Branch(Equal(
    @type, $T1))
```

```
Load(@object,
     offset = 0x10)
```

```
Call(slow path)
```

… things …

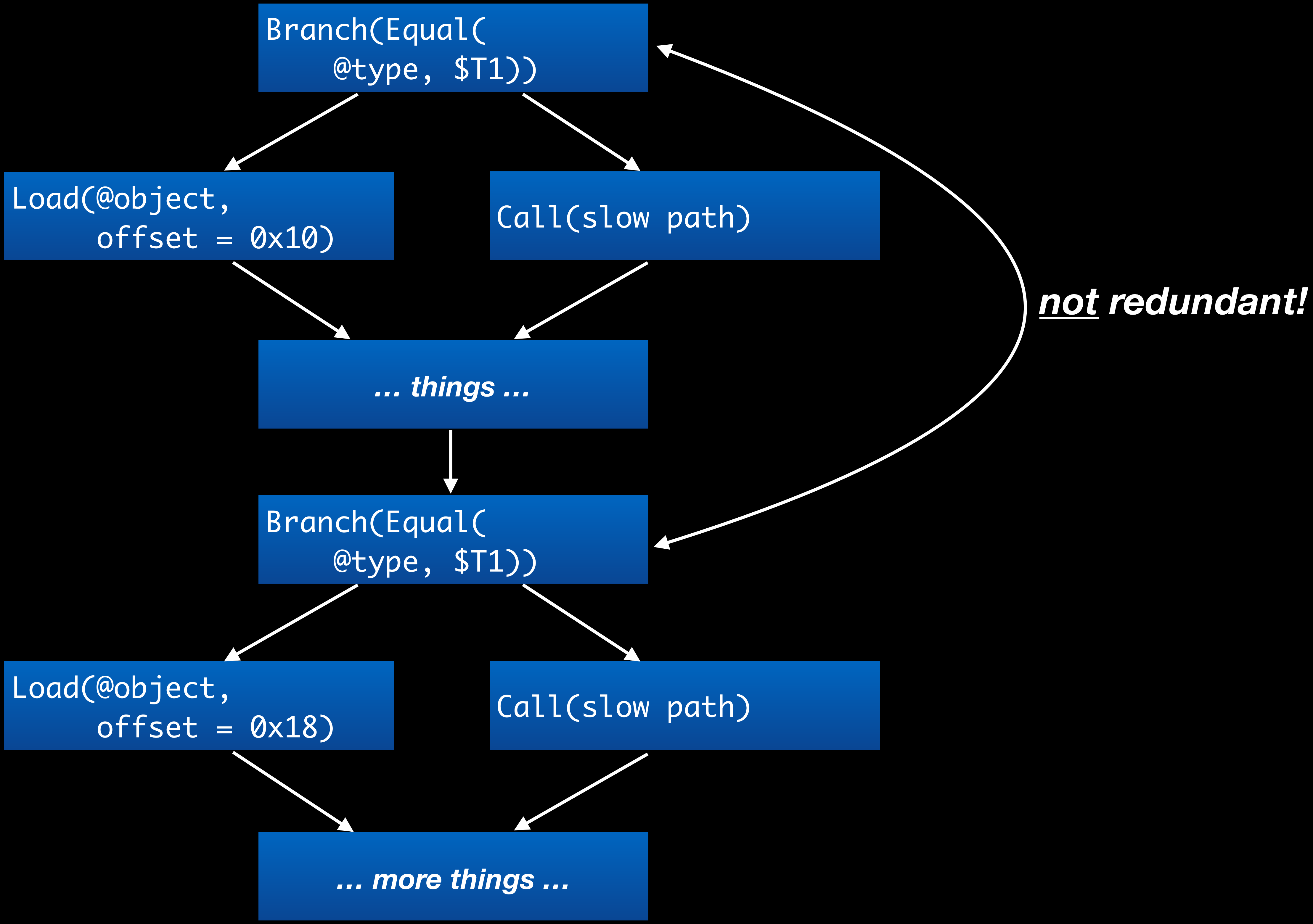
```
Branch(Equal(
    @type, $T1))
```

```
Load(@object,
     offset = 0x18)
```

```
Call(slow path)
```

… more things …

Type Checks w/o OSR



```
Branch(Equal(
    @type, $T1))
```

```
Load(@object,
    offset = 0x10)
```

```
Call(slow path)
```

… things …

```
Branch(Equal(
    @type, $T1))
```

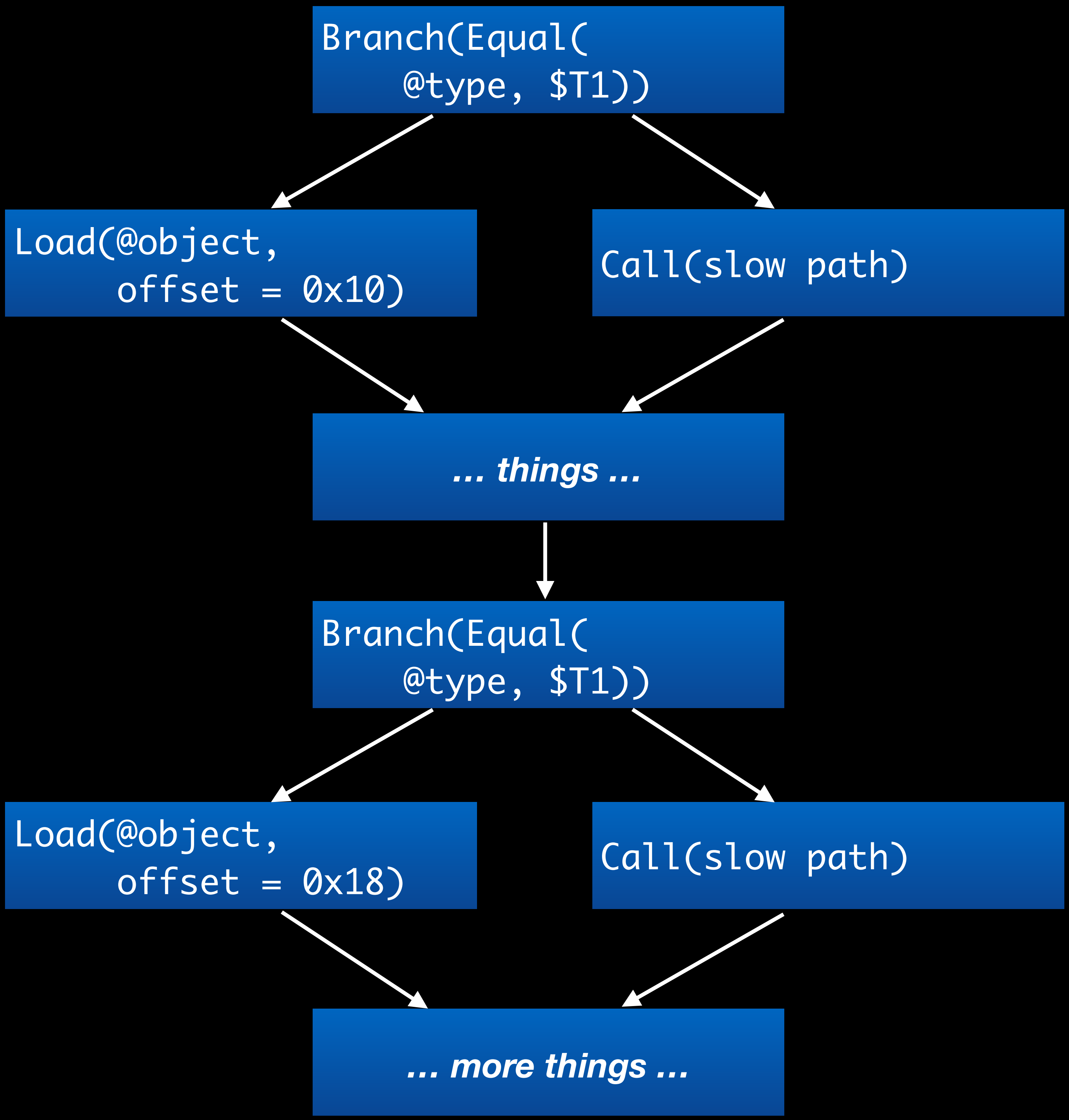
```
Load(@object,
    offset = 0x18)
```

```
Call(slow path)
```

… more things …

not redundant!

Type Checks w/o OSR



Branch(Equal(
@type, $T1))

Load(@object,
offset = 0x10)

Call(slow path)

... things ...

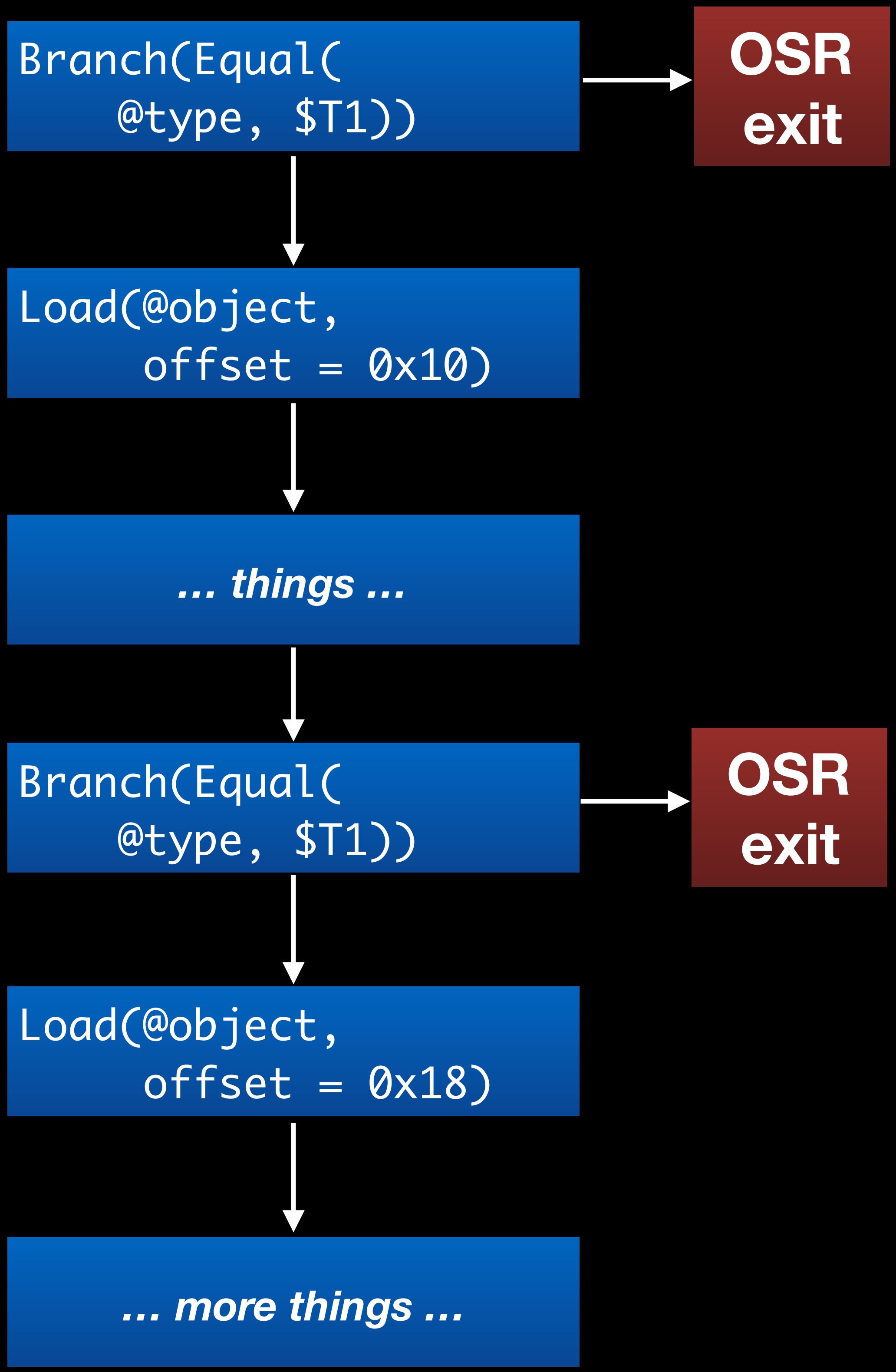
Branch(Equal(
@type, $T1))

Load(@object,
offset = 0x18)

Call(slow path)

... more things ...

Type Checks with OSR



Branch(Equal(
@type, $T1))

OSR exit

Load(@object,
offset = 0x10)

... things ...

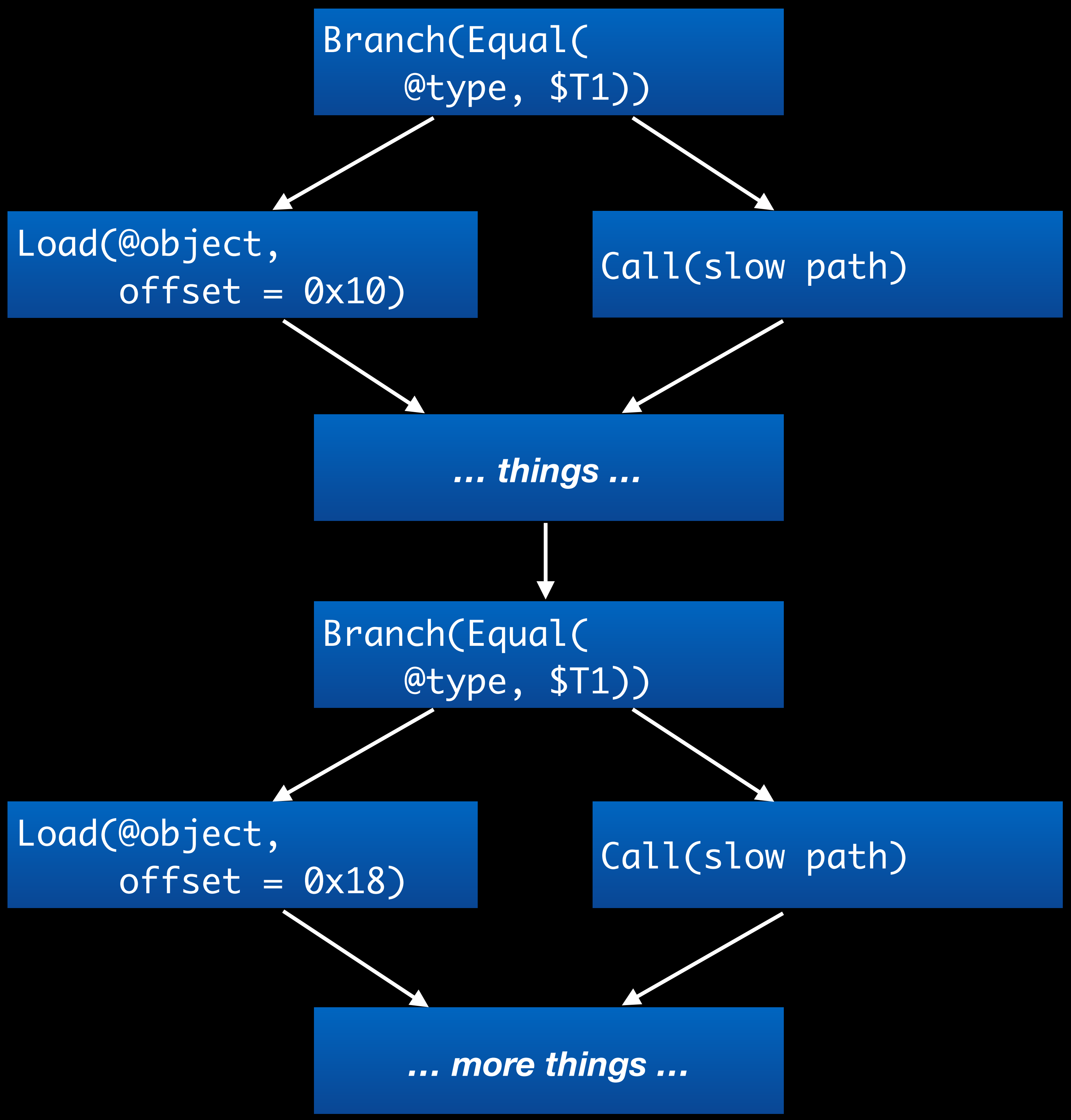
Branch(Equal(
@type, $T1))

OSR exit

Load(@object,
offset = 0x18)

... more things ...

Type Checks w/o OSR



```
Branch(Equal(
    @type, $T1))
```

```
Load(@object,
    offset = 0x10)
```

```
Call(slow path)
```

… things …

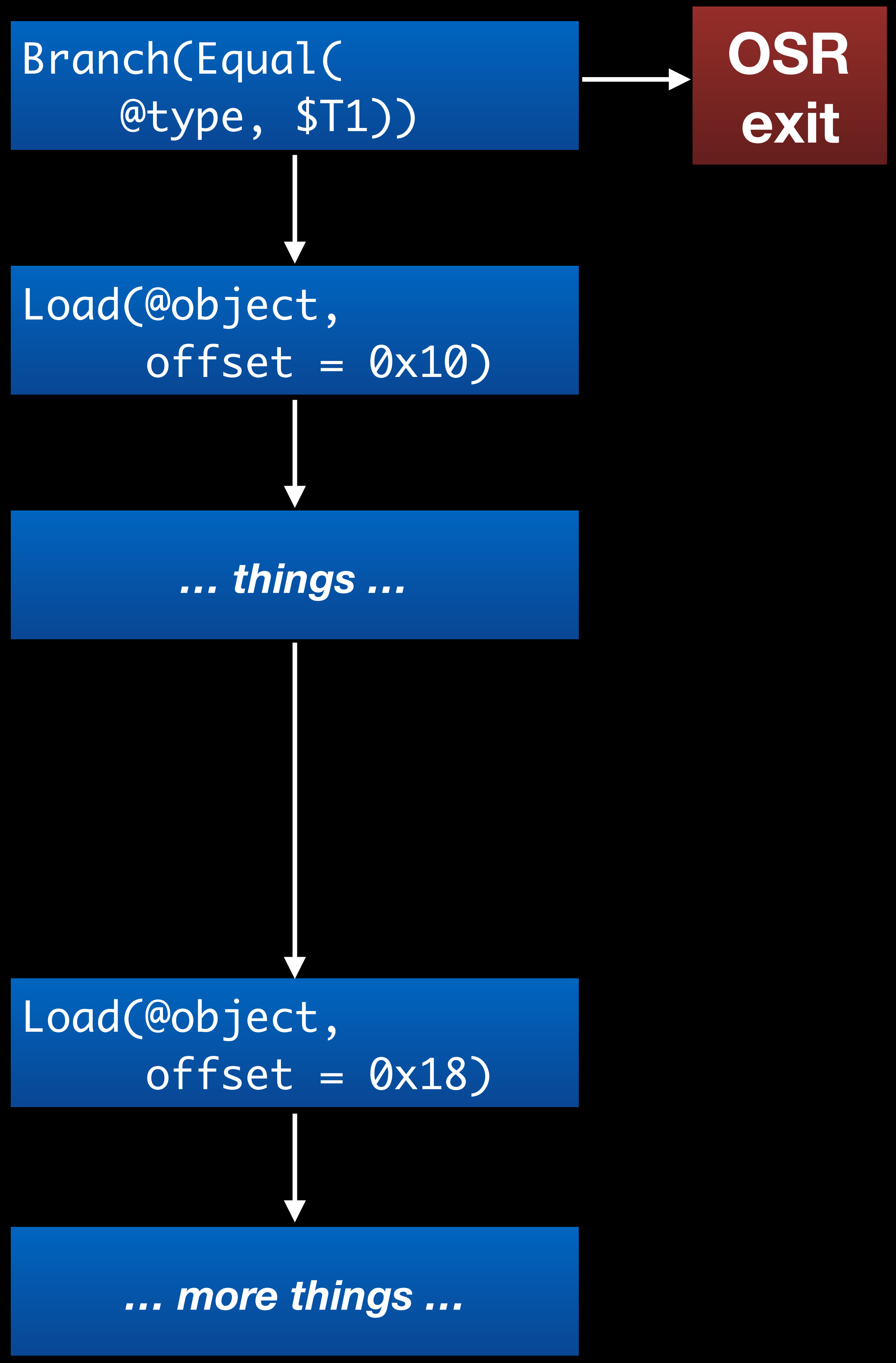
```
Branch(Equal(
    @type, $T1))
```

```
Load(@object,
    offset = 0x18)
```

```
Call(slow path)
```

… more things …

Type Checks with OSR



```
Branch(Equal(
    @type, $T1))
```

OSR exit

```
Load(@object,
    offset = 0x10)
```

… things …

```
Load(@object,
    offset = 0x18)
```

… more things …

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;
}
```

```c
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;
}
```

OSR IR Goals

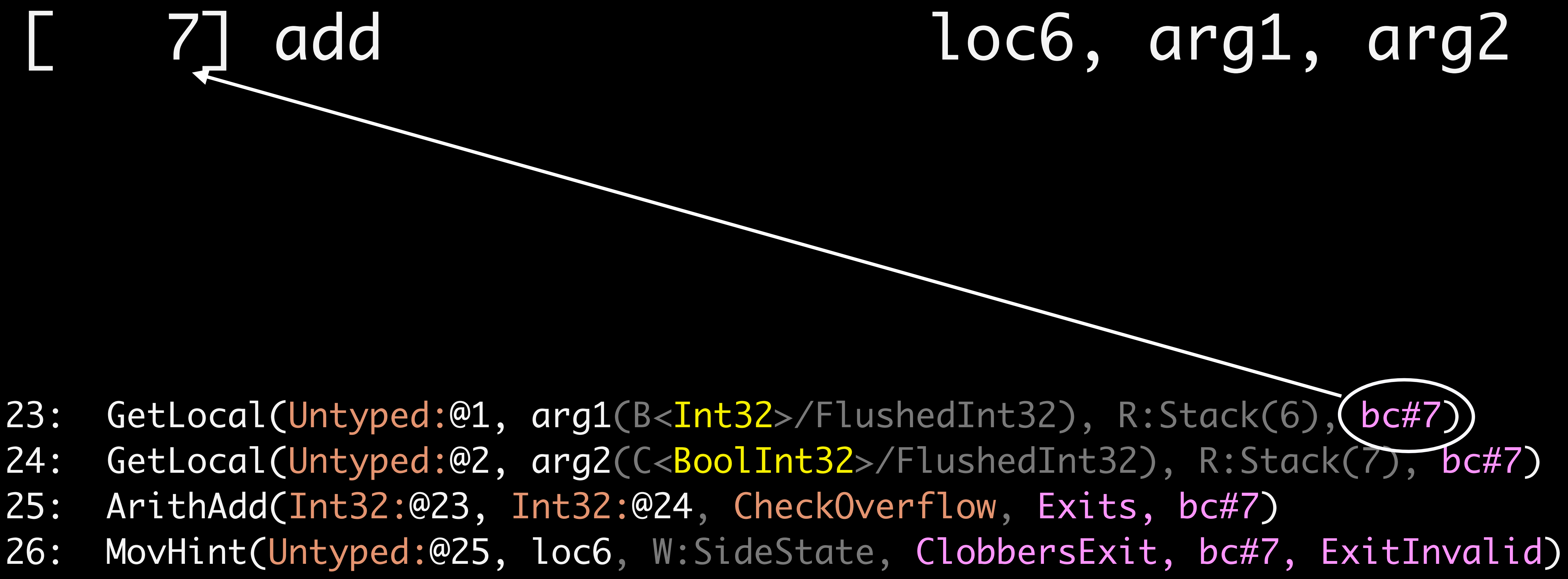
- Must know where to exit.
- Must know what is live-at-exit.
- Must be malleable.

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)
```

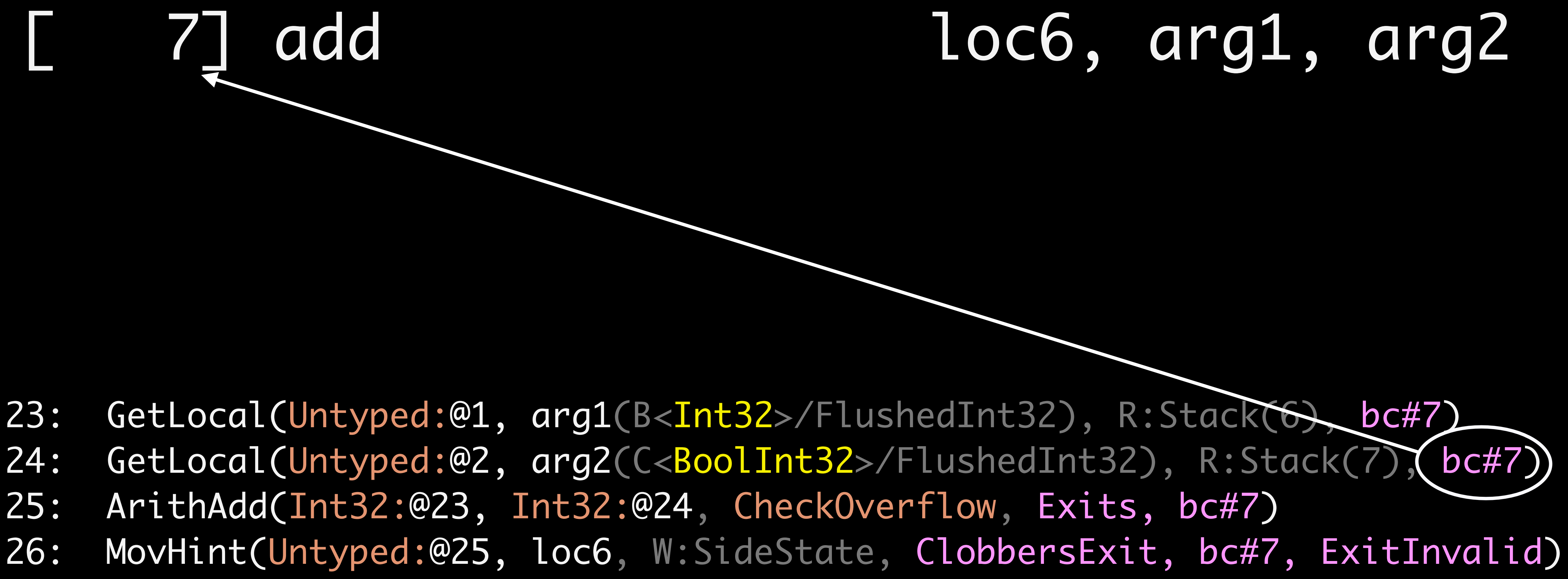
```
[   7] add               loc6, arg1, arg2
```

```
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)
```



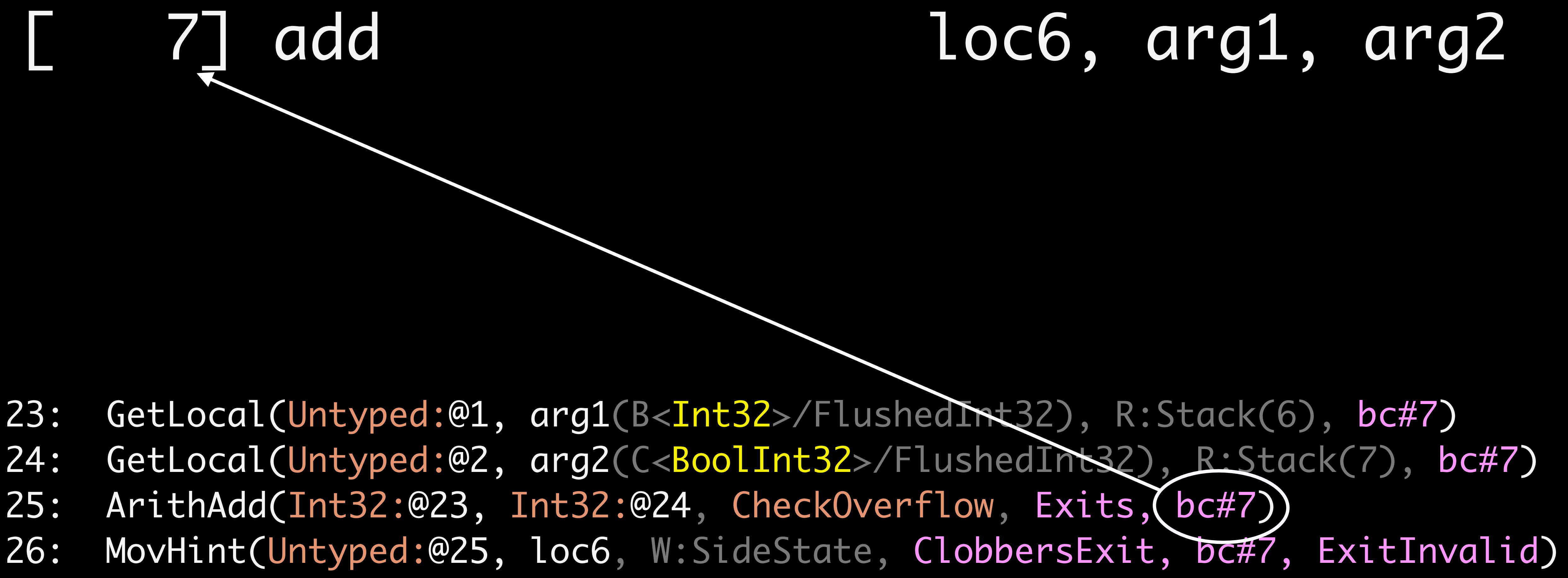
```
[   7] add                          loc6, arg1, arg2
```

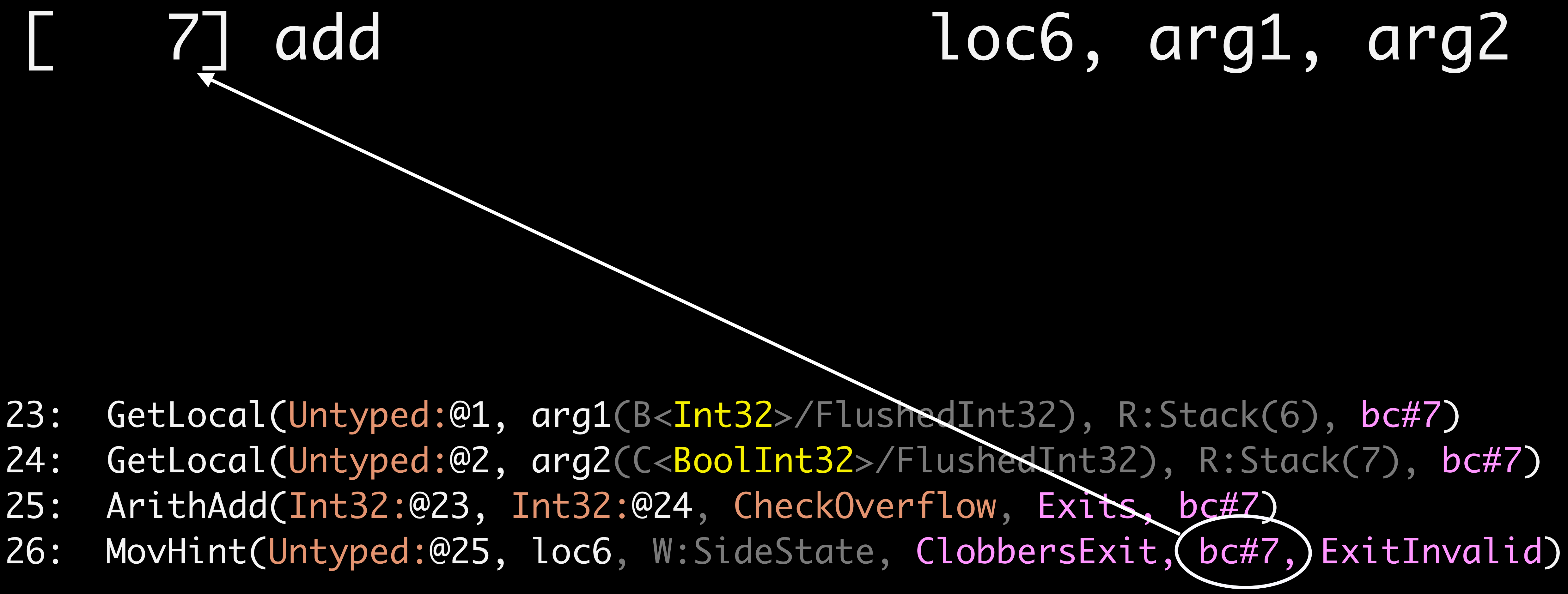
```
23:  GetLocal(Untyped:@1, arg1(B<Int32>/FlushedInt32), R:Stack(6), bc#7)
24:  GetLocal(Untyped:@2, arg2(C<BoolInt32>/FlushedInt32), R:Stack(7), bc#7)
25:  ArithAdd(Int32:@23, Int32:@24, CheckOverflow, Exits, bc#7)
26:  MovHint(Untyped:@25, loc6, W:SideState, ClobbersExit, bc#7, ExitInvalid)
```



```
[   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)
```

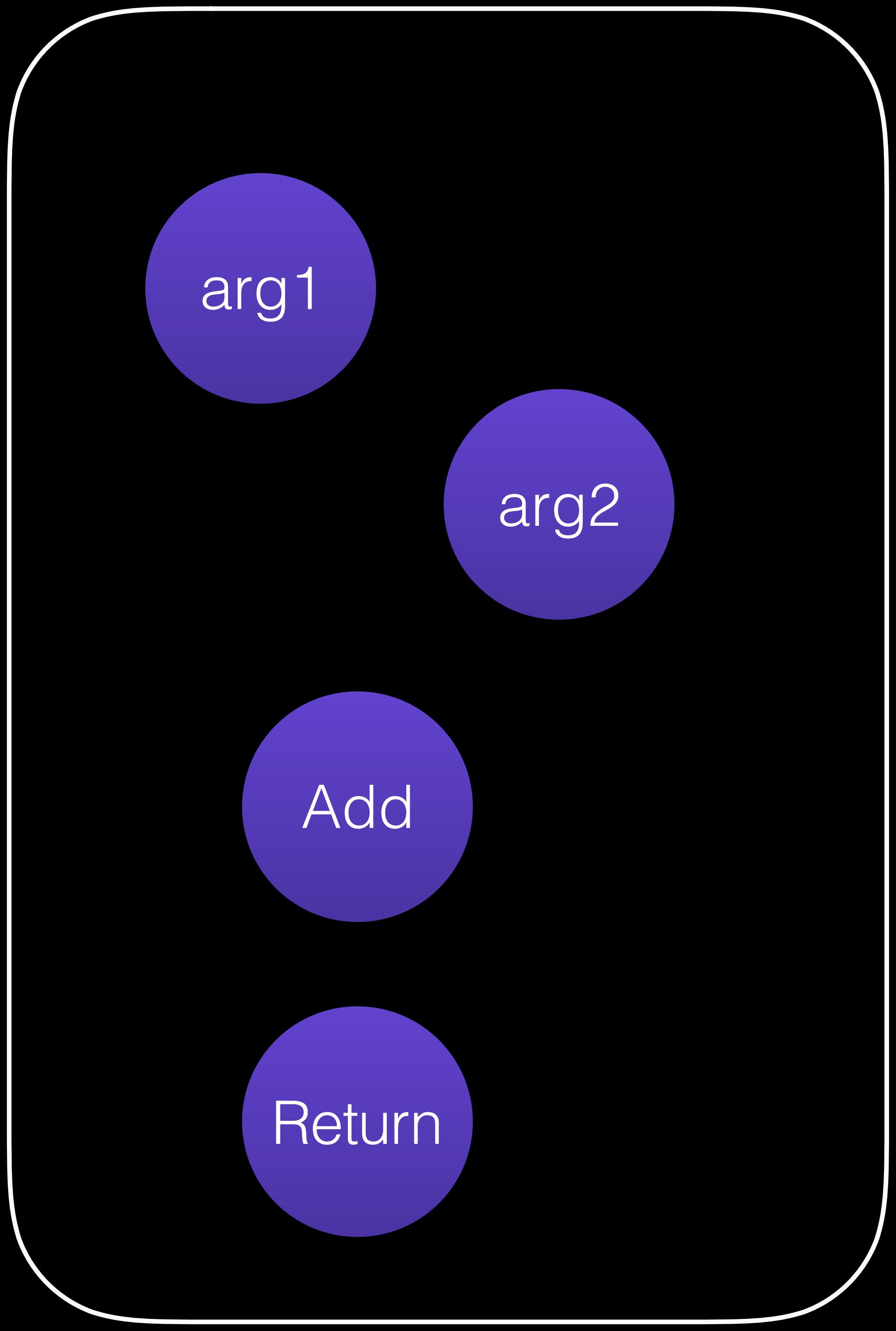
```
[   7] add                          loc6, arg1, arg2
```

```
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)
```

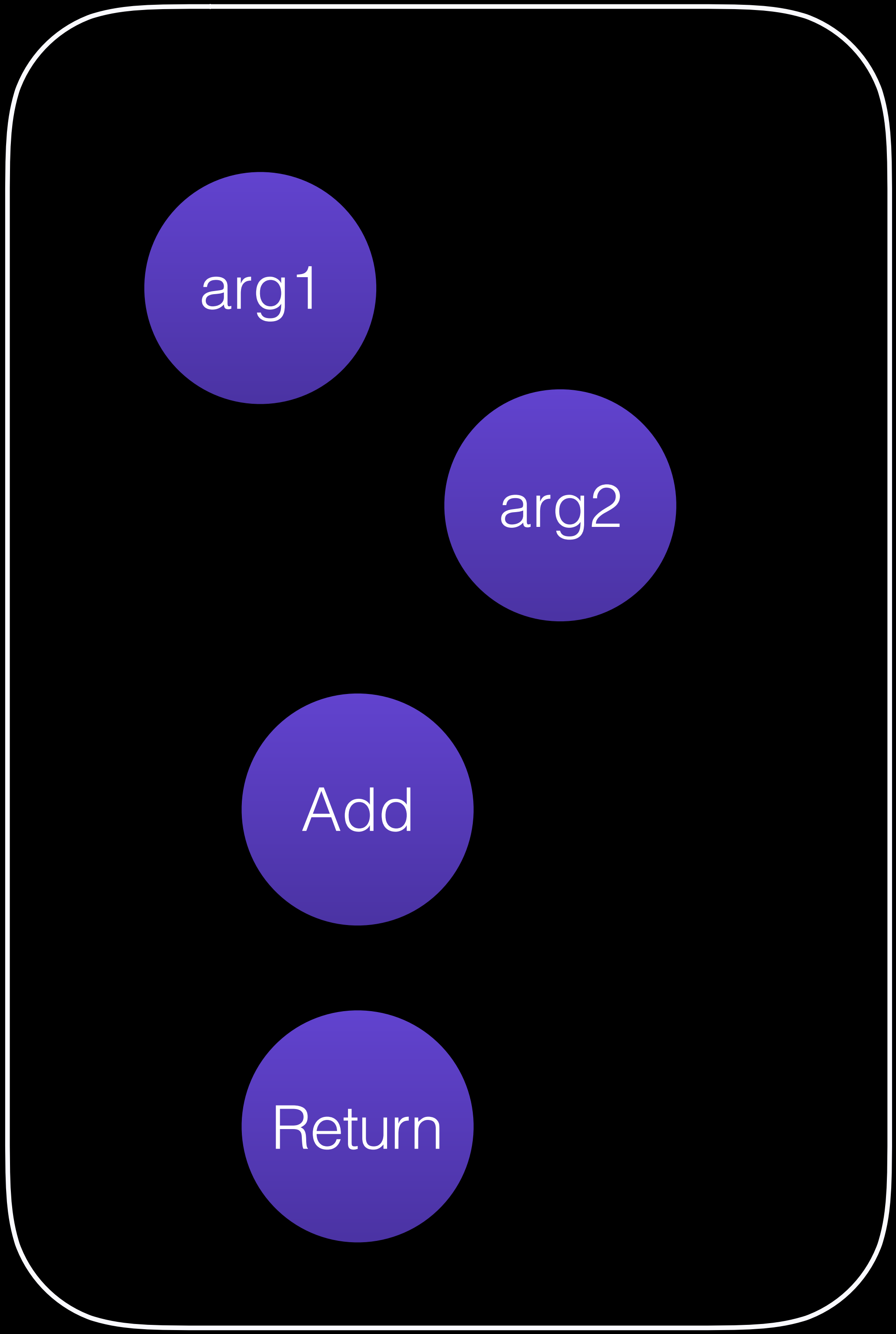
```
[   7] add                         loc6, arg1, arg2
```

```
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)
```

DFG SSA state



DFG SSA state



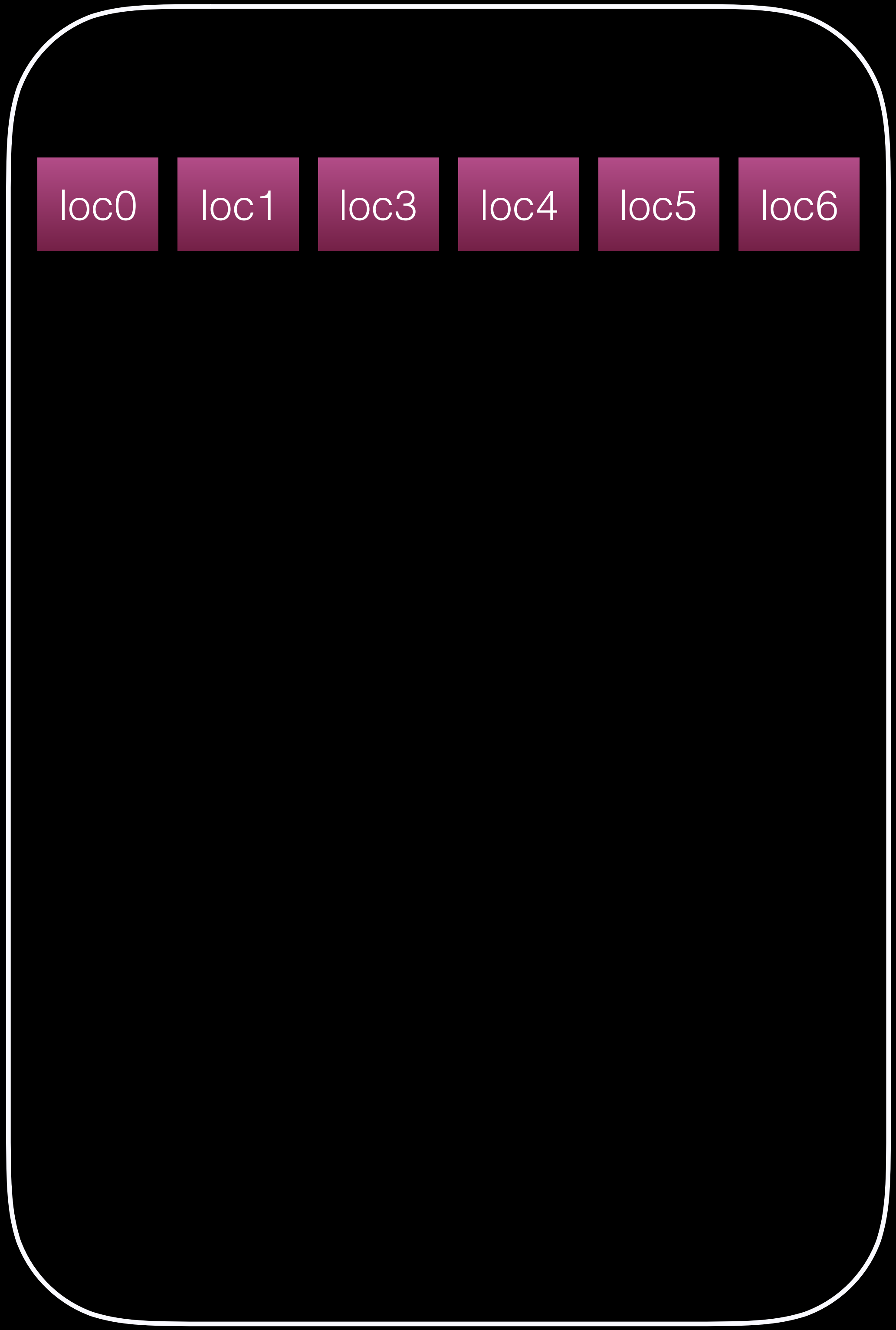
arg1

arg2

Add

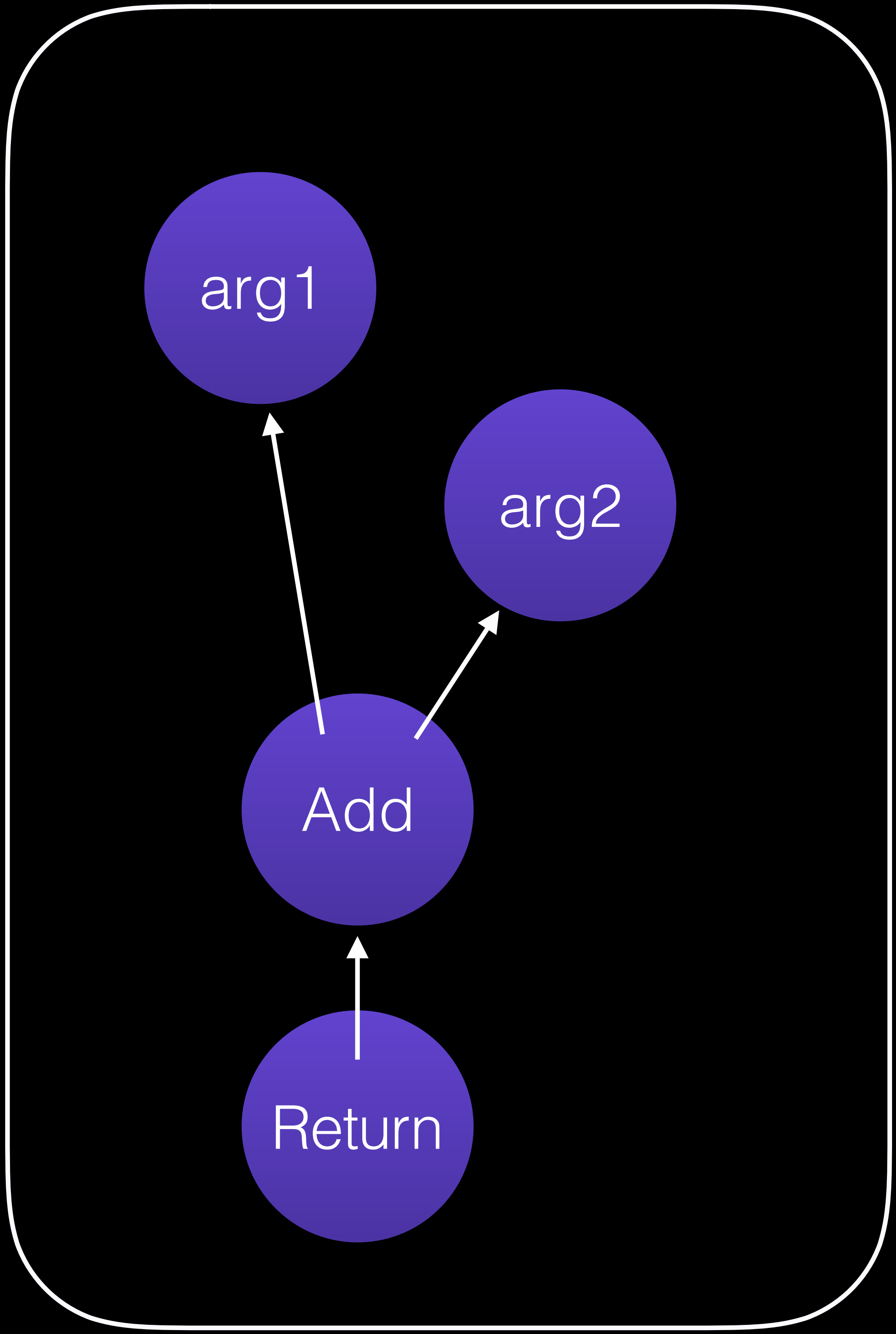
Return

DFG Exit state

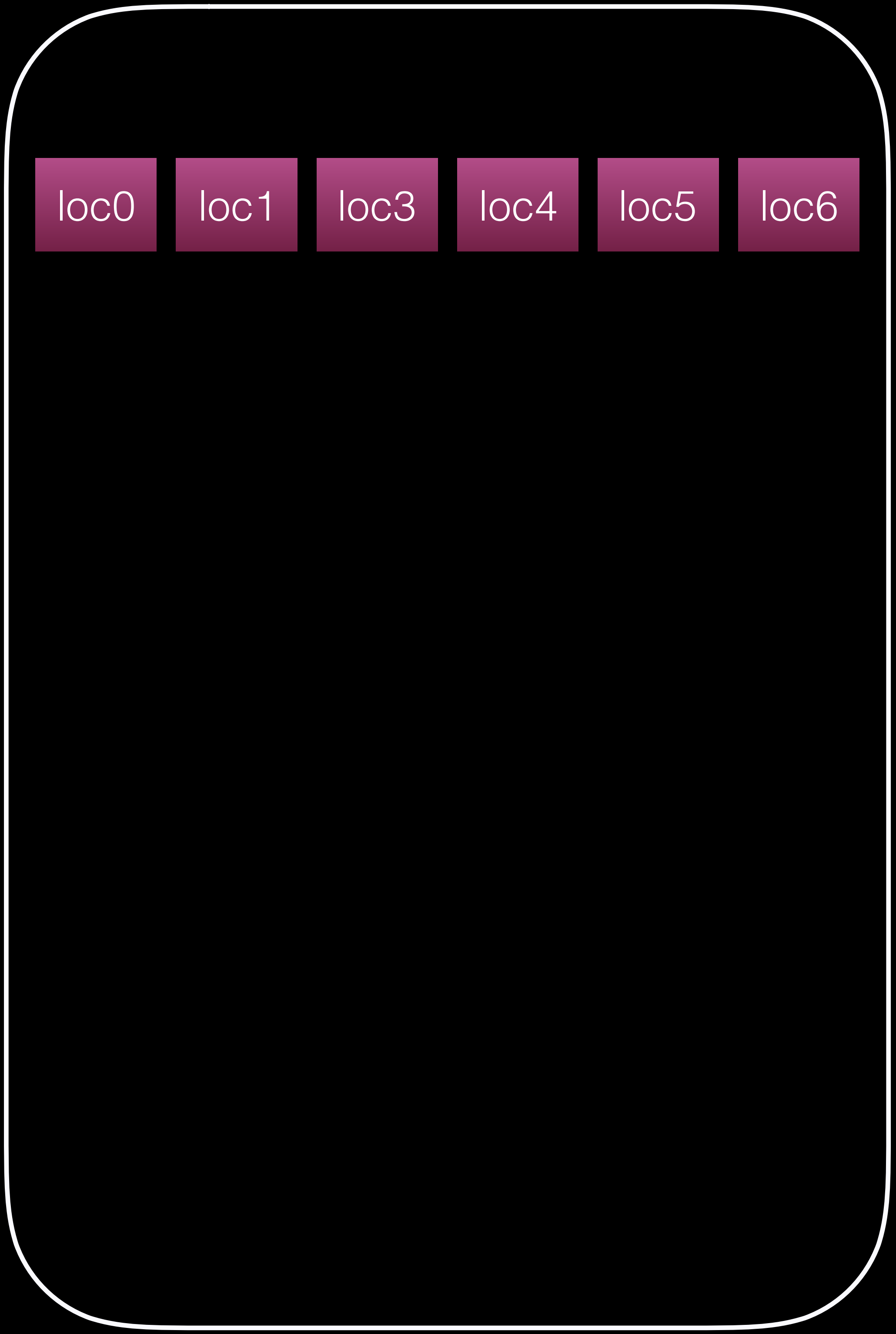


loc0	loc1	loc3	loc4	loc5	loc6

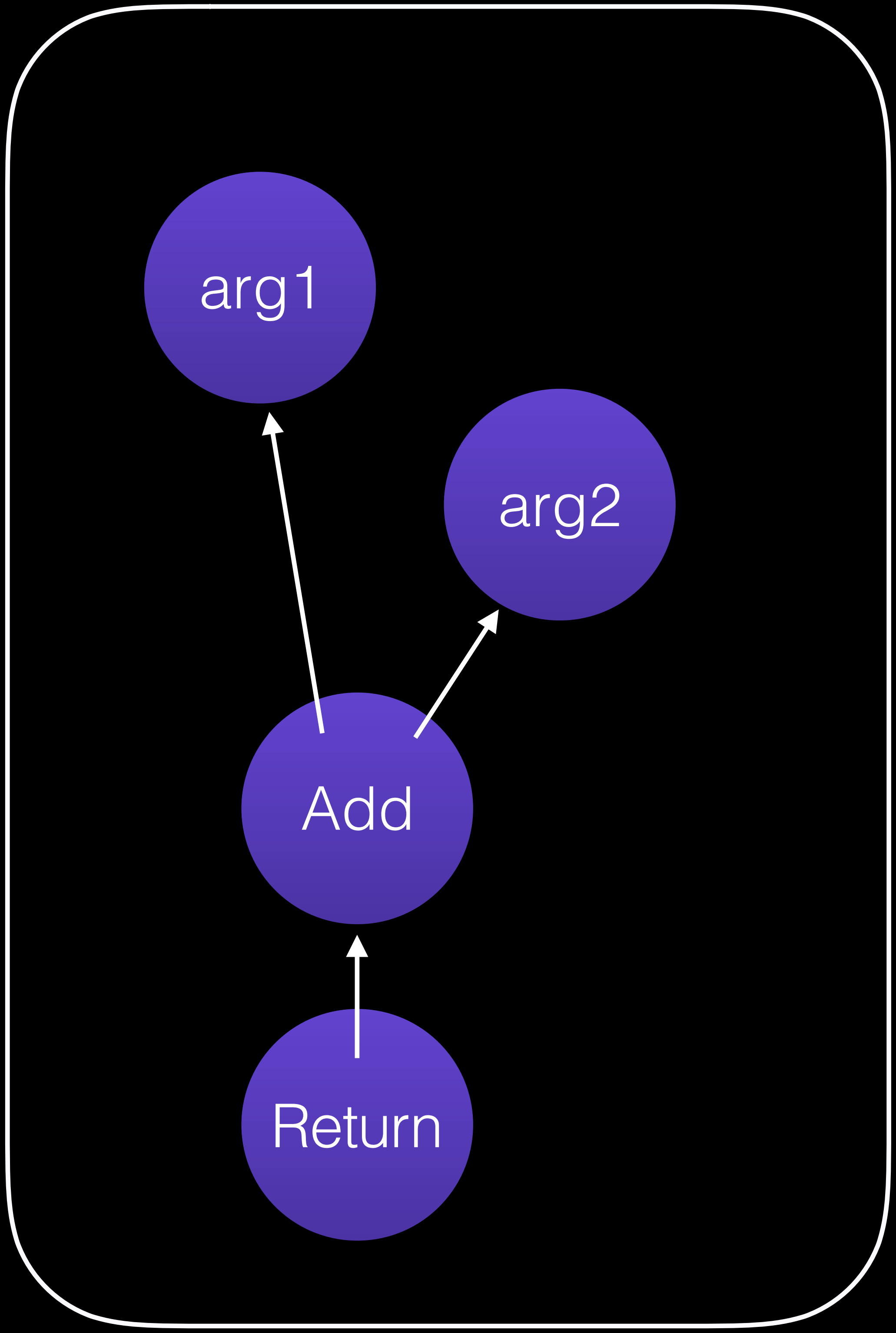
DFG SSA state



DFG Exit state



DFG SSA state



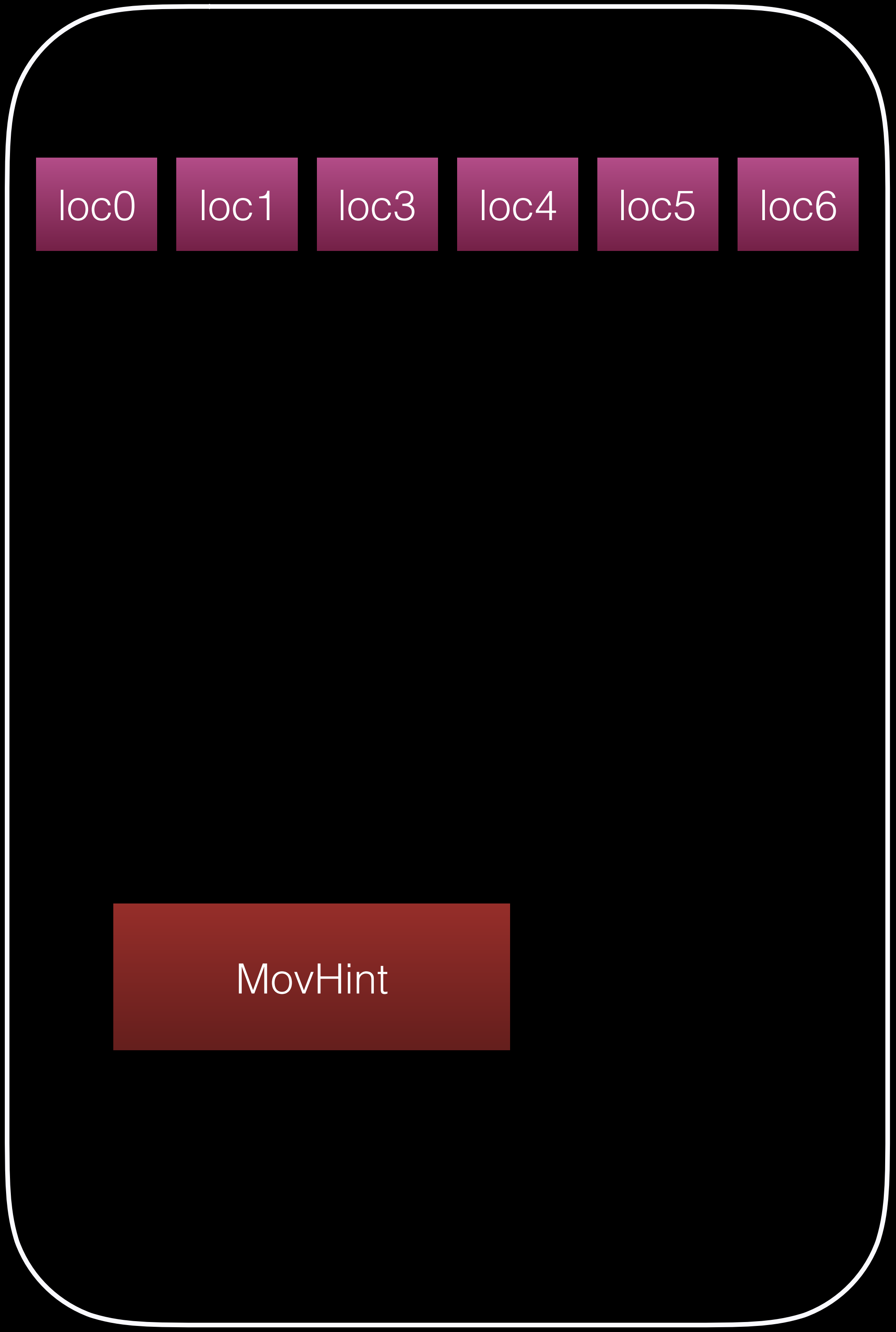
arg1

arg2

Add

Return

DFG Exit state

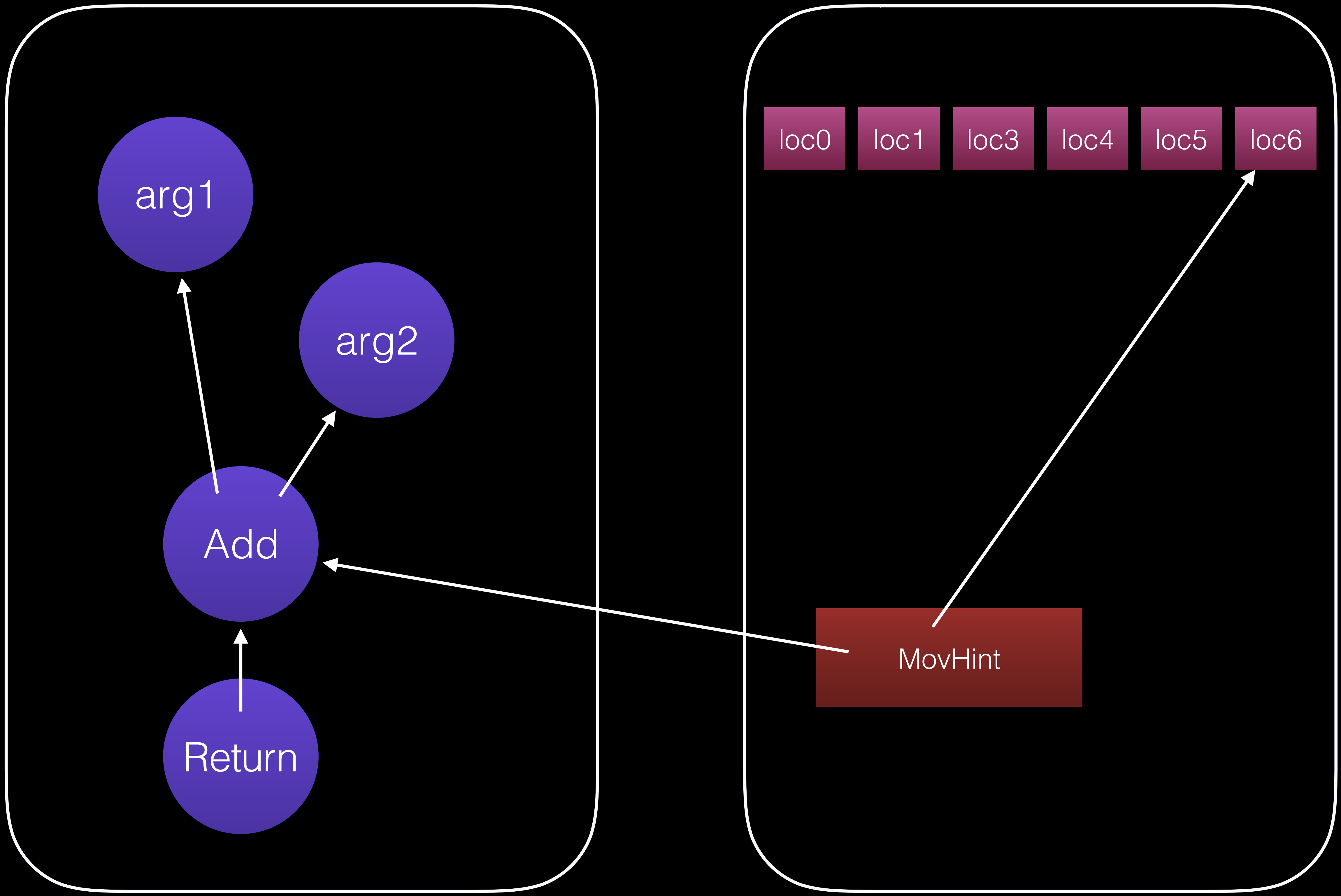


loc0 loc1 loc3 loc4 loc5 loc6

MovHint

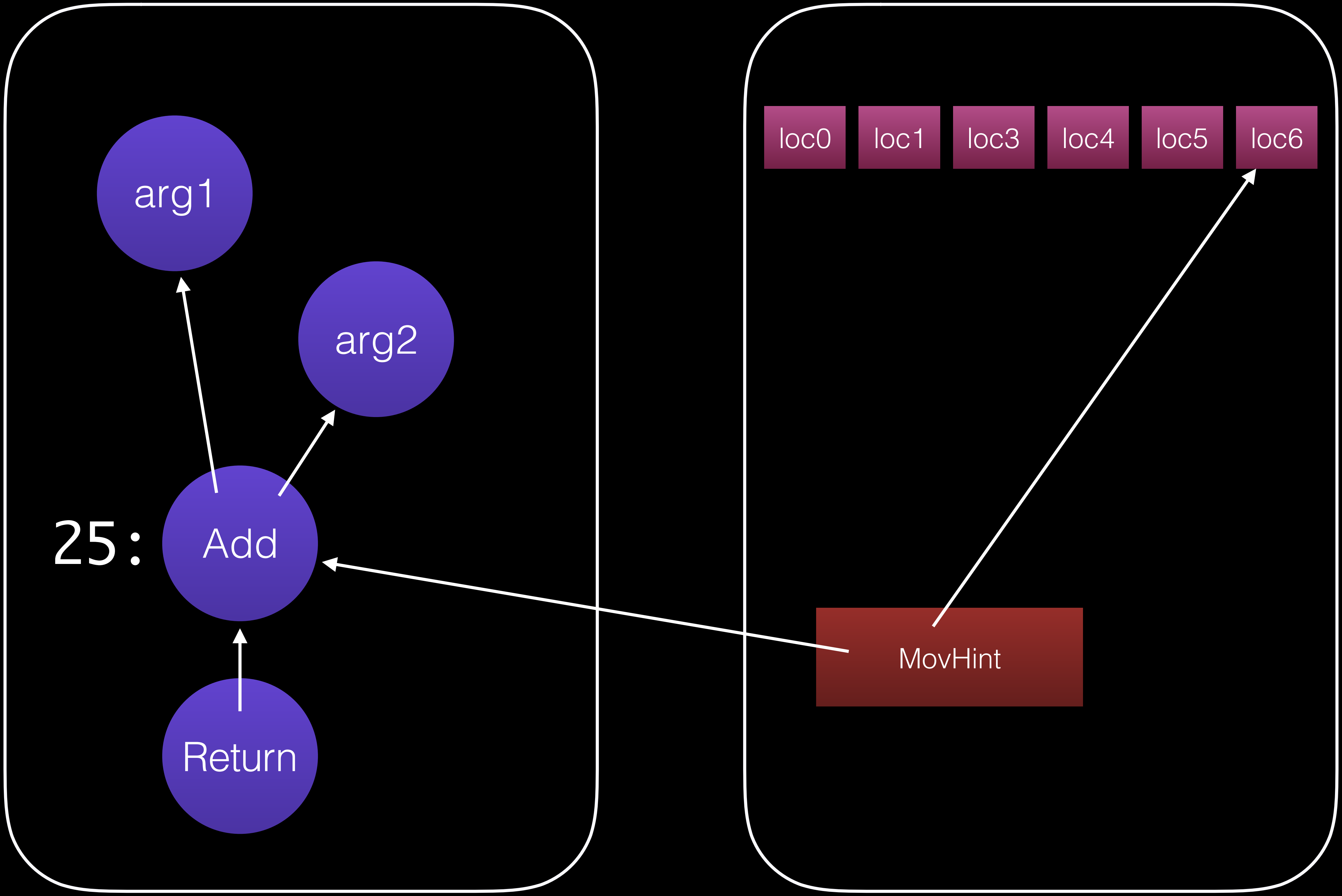
DFG SSA state

DFG Exit state



DFG SSA state

DFG Exit state



arg1

arg2

25: Add

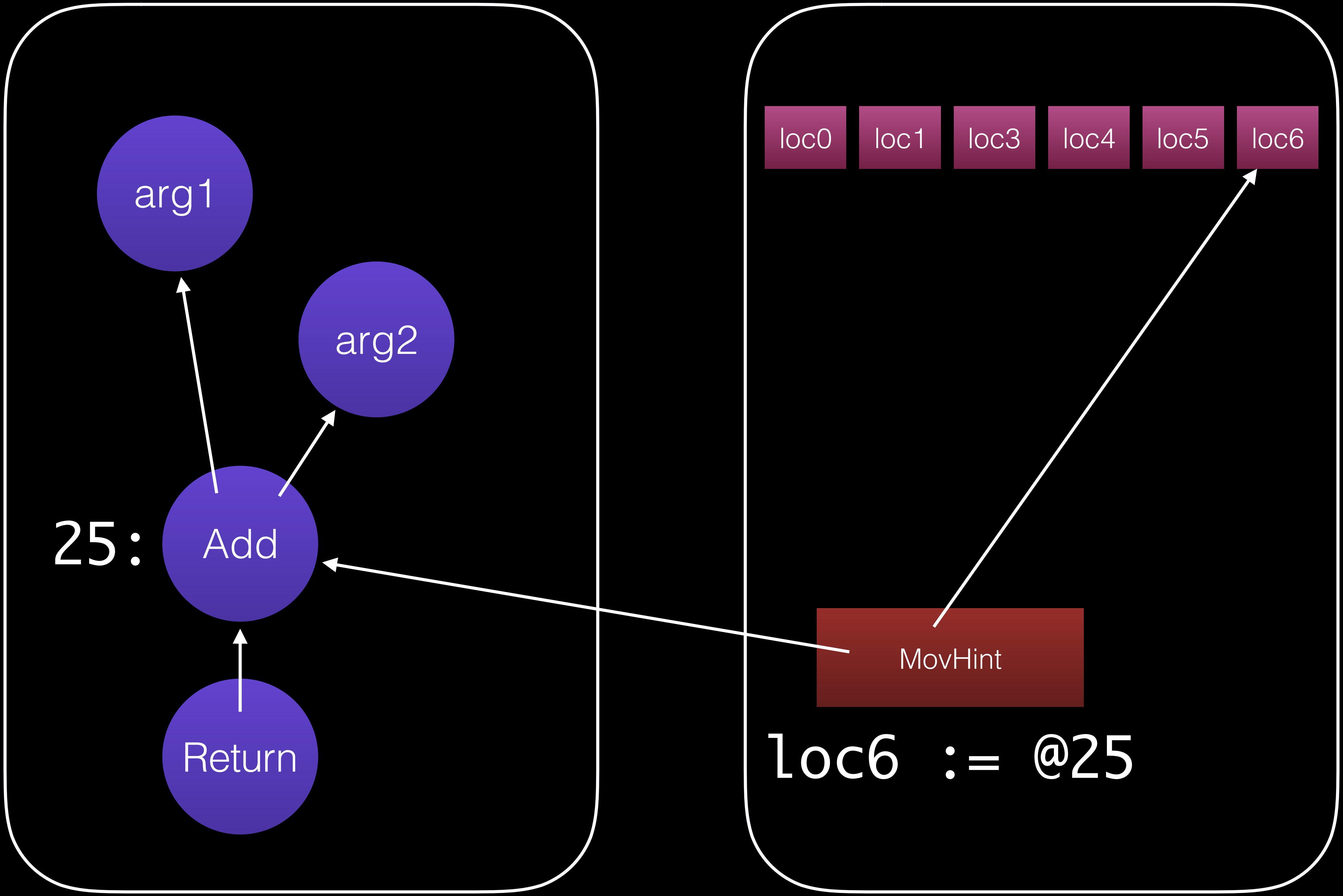
Return

loc0 loc1 loc3 loc4 loc5 loc6

MovHint

DFG SSA state

DFG Exit state



arg1

arg2

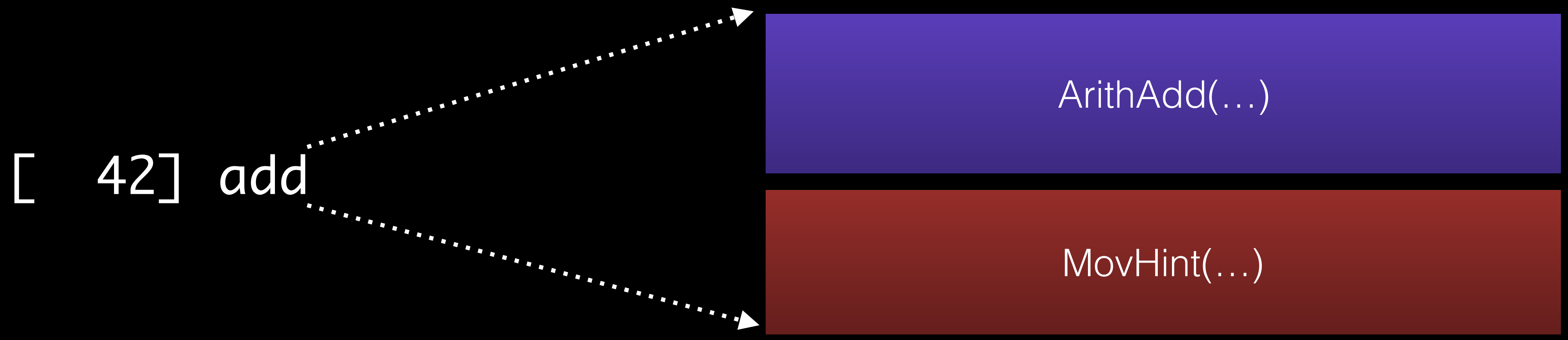
25: Add

Return

loc0 loc1 loc3 loc4 loc5 loc6

MovHint

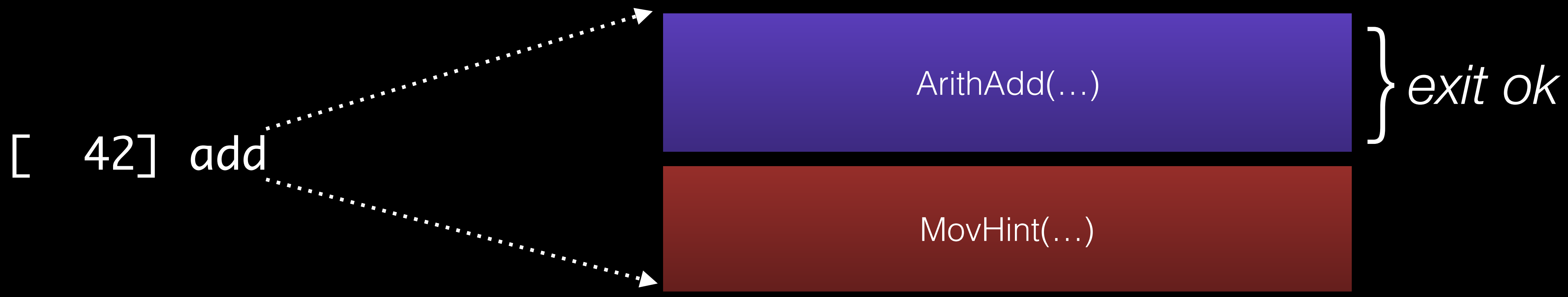
`loc6 := @25`



```
[  42] add
```

ArithAdd(…)

MovHint(…)

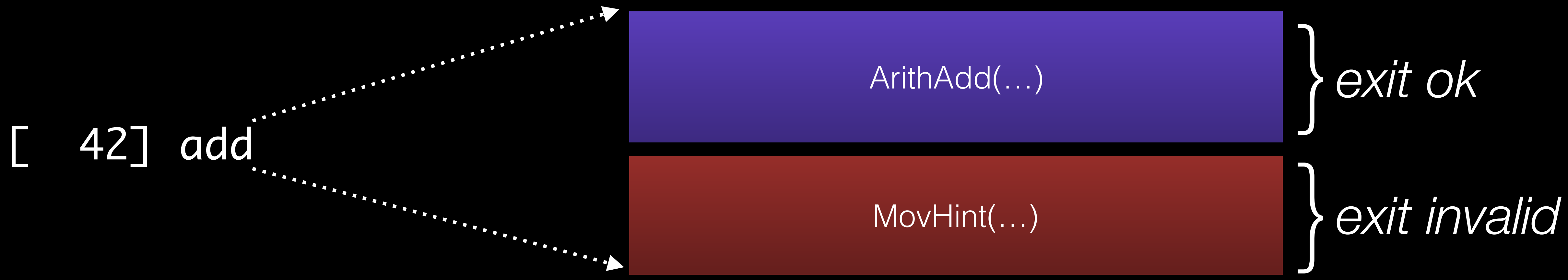


```
[  42] add
```

ArithAdd(…)

exit ok

MovHint(…)



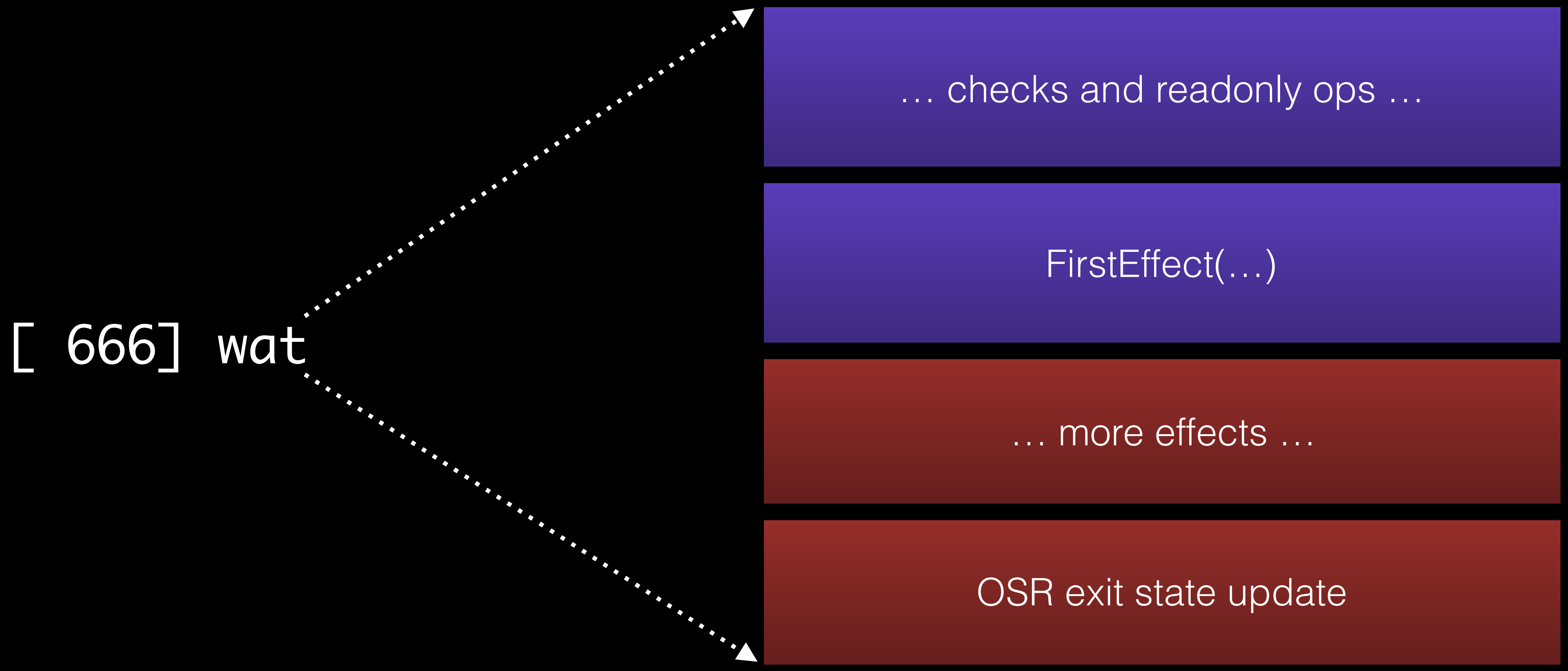
[42] add

ArithAdd(…)

exit ok

MovHint(…)

exit invalid



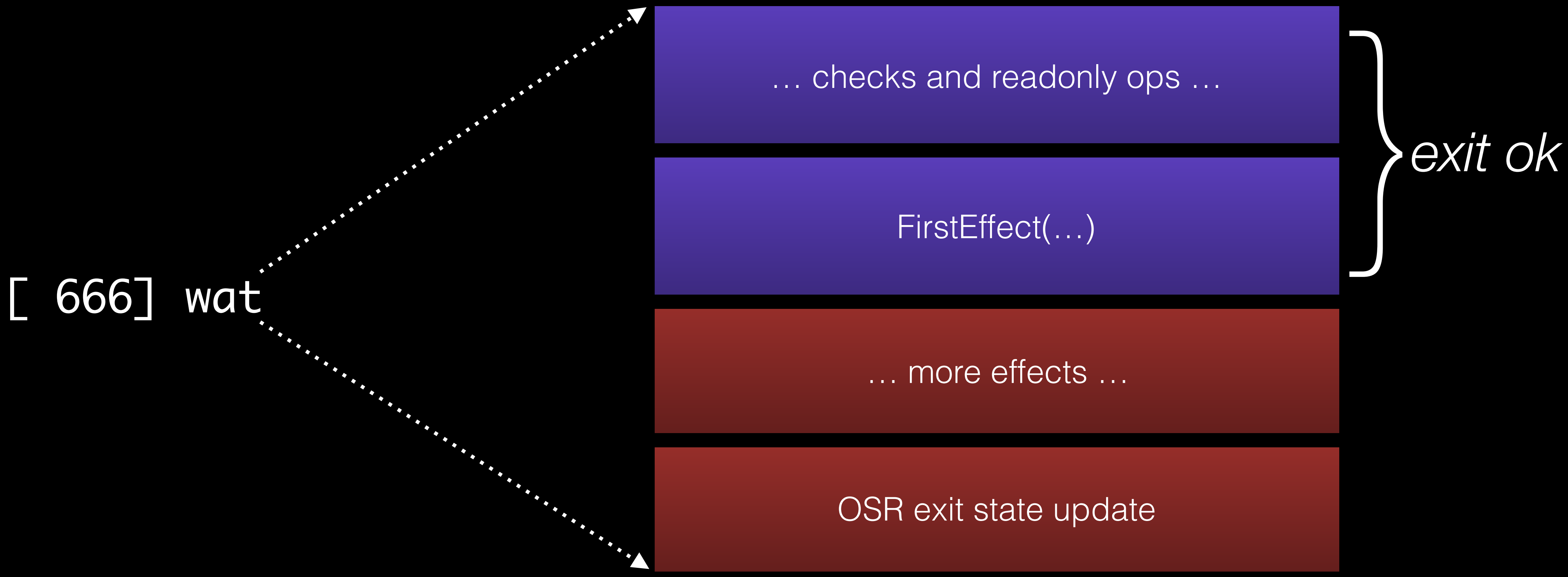
[666] wat

… checks and readonly ops …

FirstEffect(…)

… more effects …

OSR exit state update



[666] wat

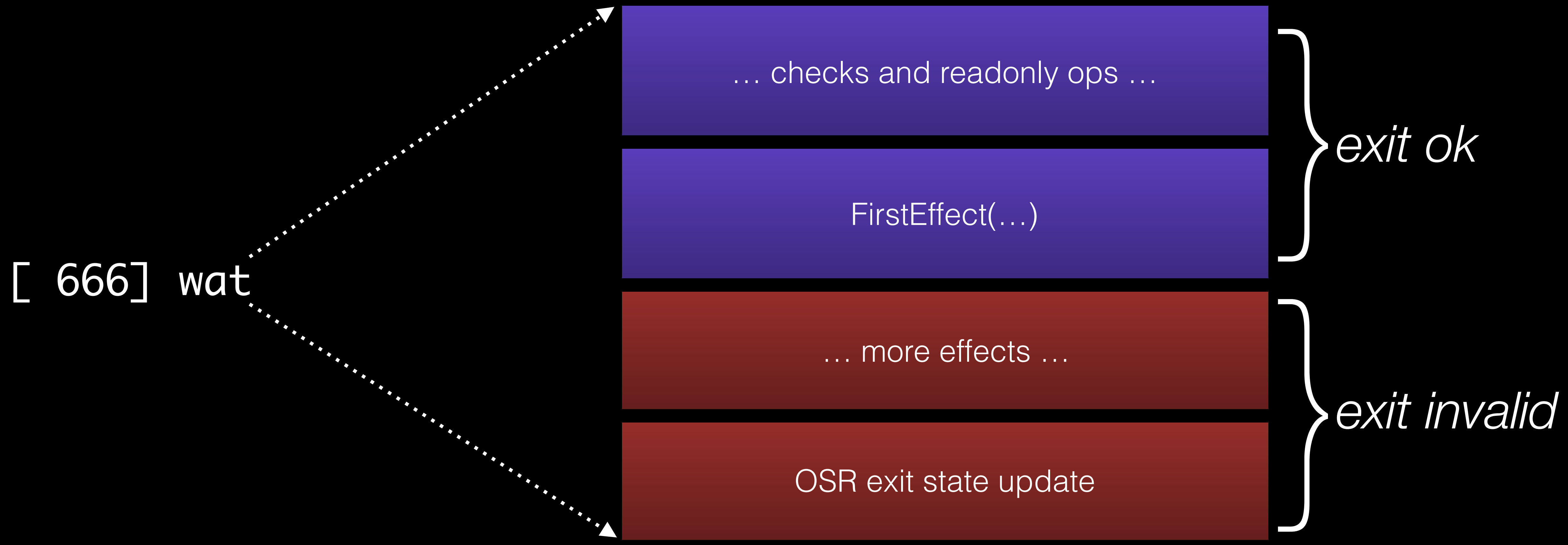
… checks and readonly ops …

FirstEffect(…)

exit ok

… more effects …

OSR exit state update



[666] wat

… checks and readonly ops …

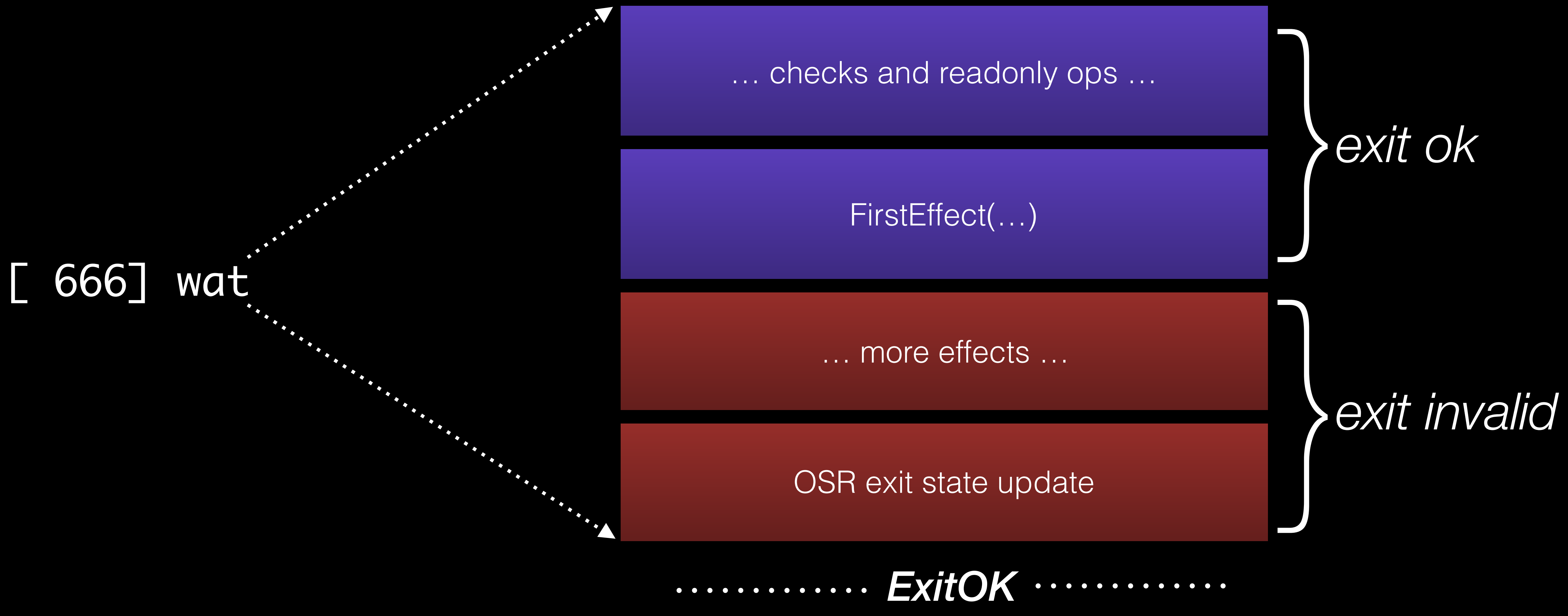
FirstEffect(…)

… more effects …

OSR exit state update

exit ok

exit invalid



[666] wat

… checks and readonly ops …

FirstEffect(…)

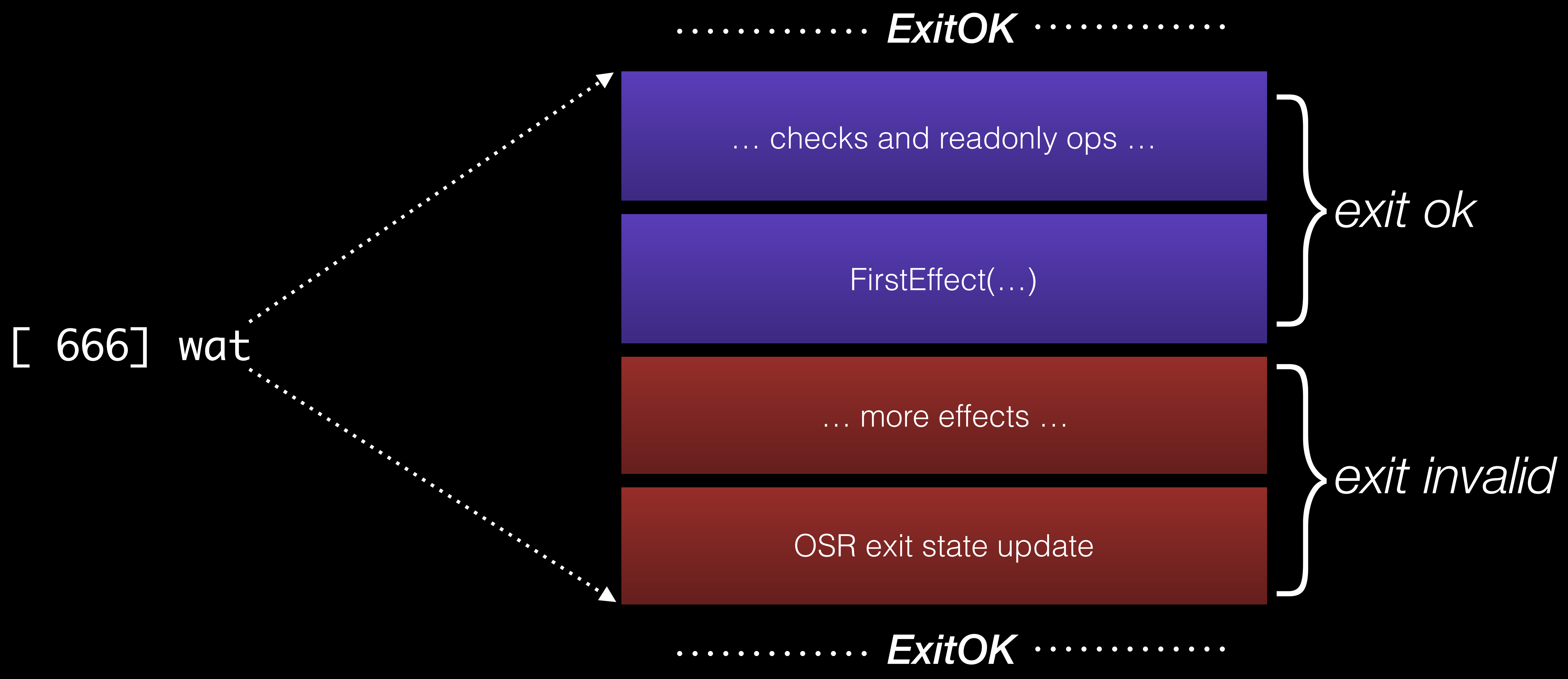
… more effects …

OSR exit state update

exit ok

exit invalid

ExitOK



ExitOK

[666] wat

… checks and readonly ops …

FirstEffect(…)

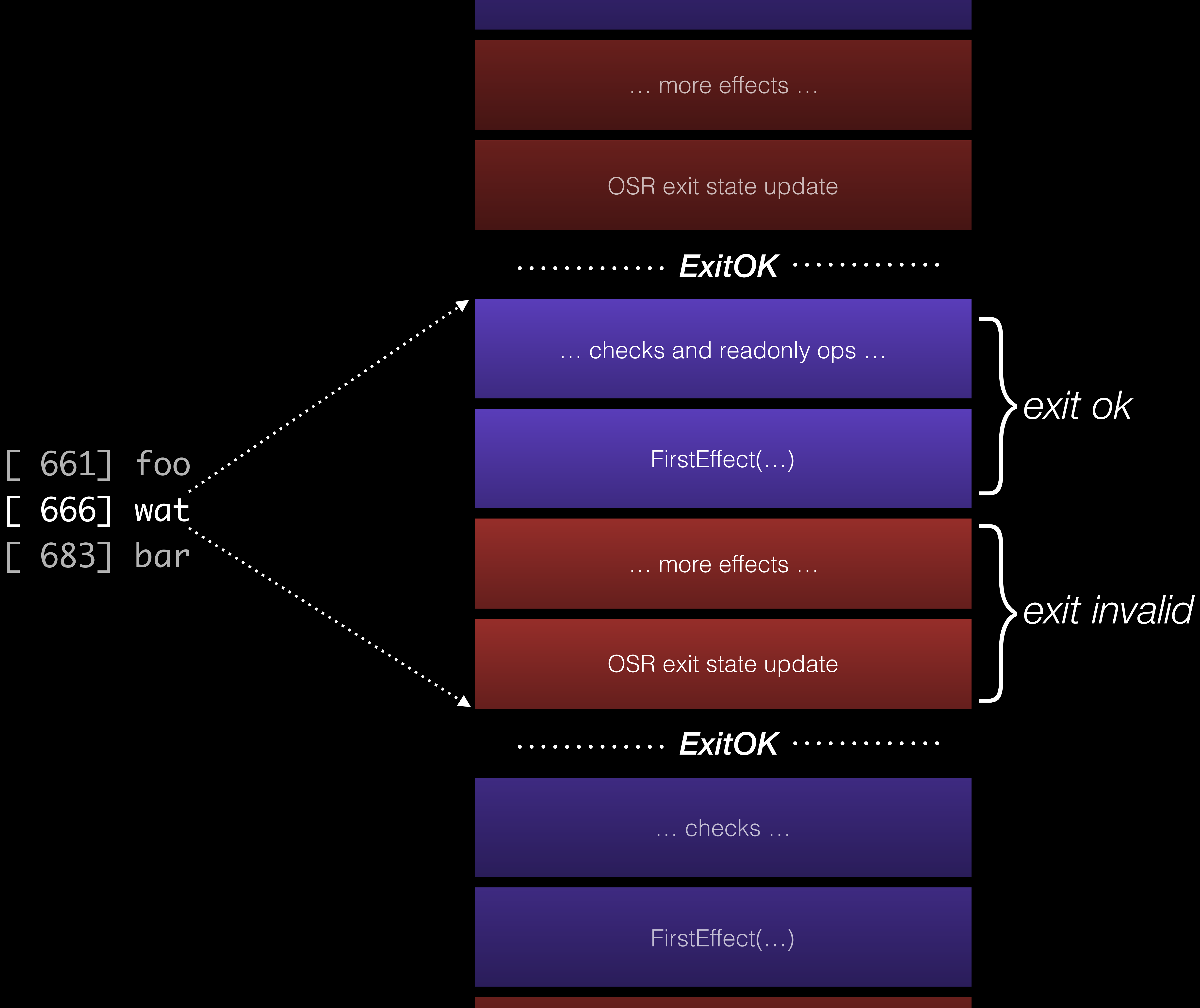
exit ok

… more effects …

OSR exit state update

exit invalid

ExitOK



```
[ 661] foo
[ 666] wat
[ 683] bar
```

… more effects …

OSR exit state update

ExitOK

… checks and readonly ops …

FirstEffect(…)

exit ok

… more effects …

OSR exit state update

exit invalid

ExitOK

… checks …

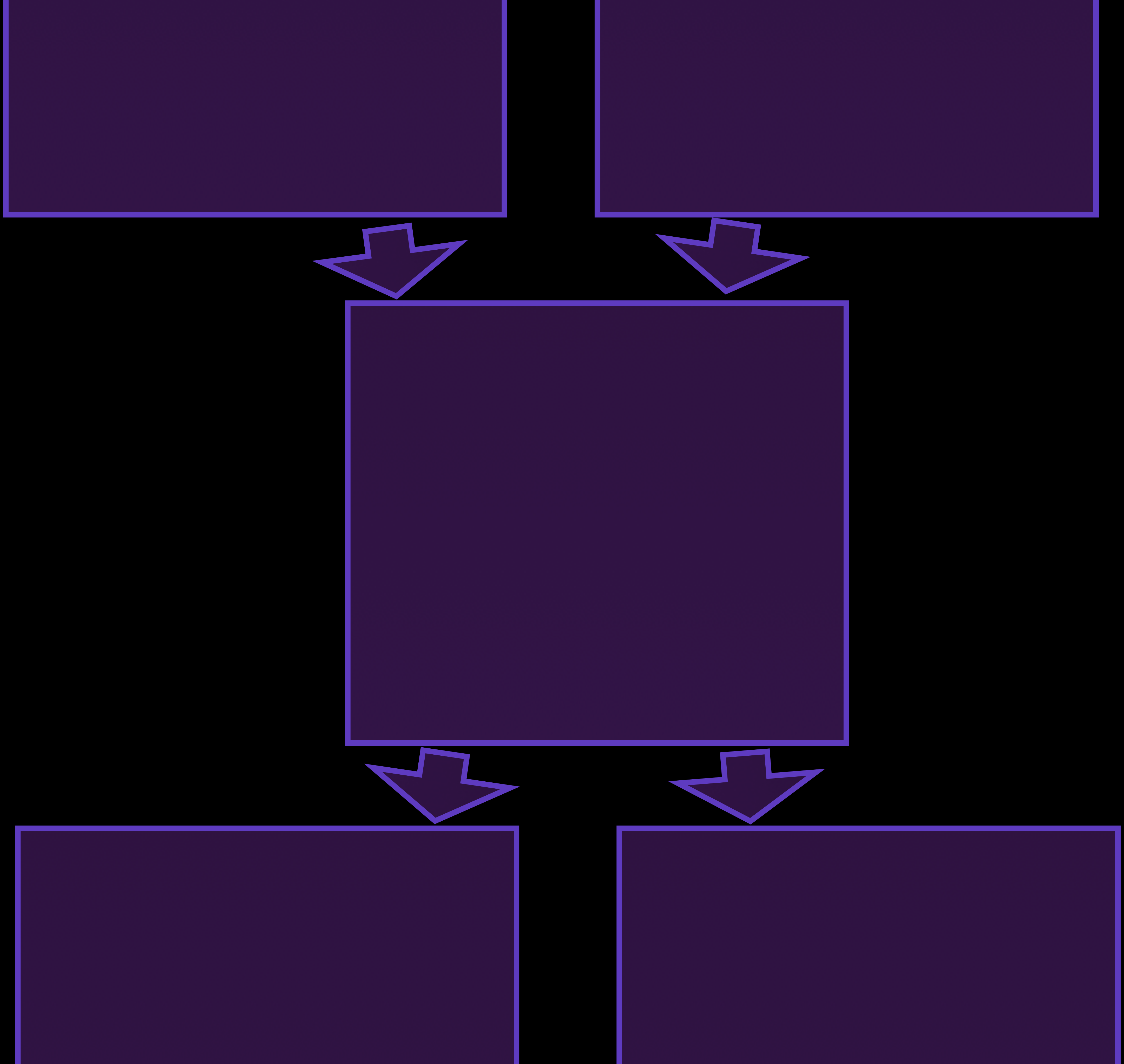
FirstEffect(…)

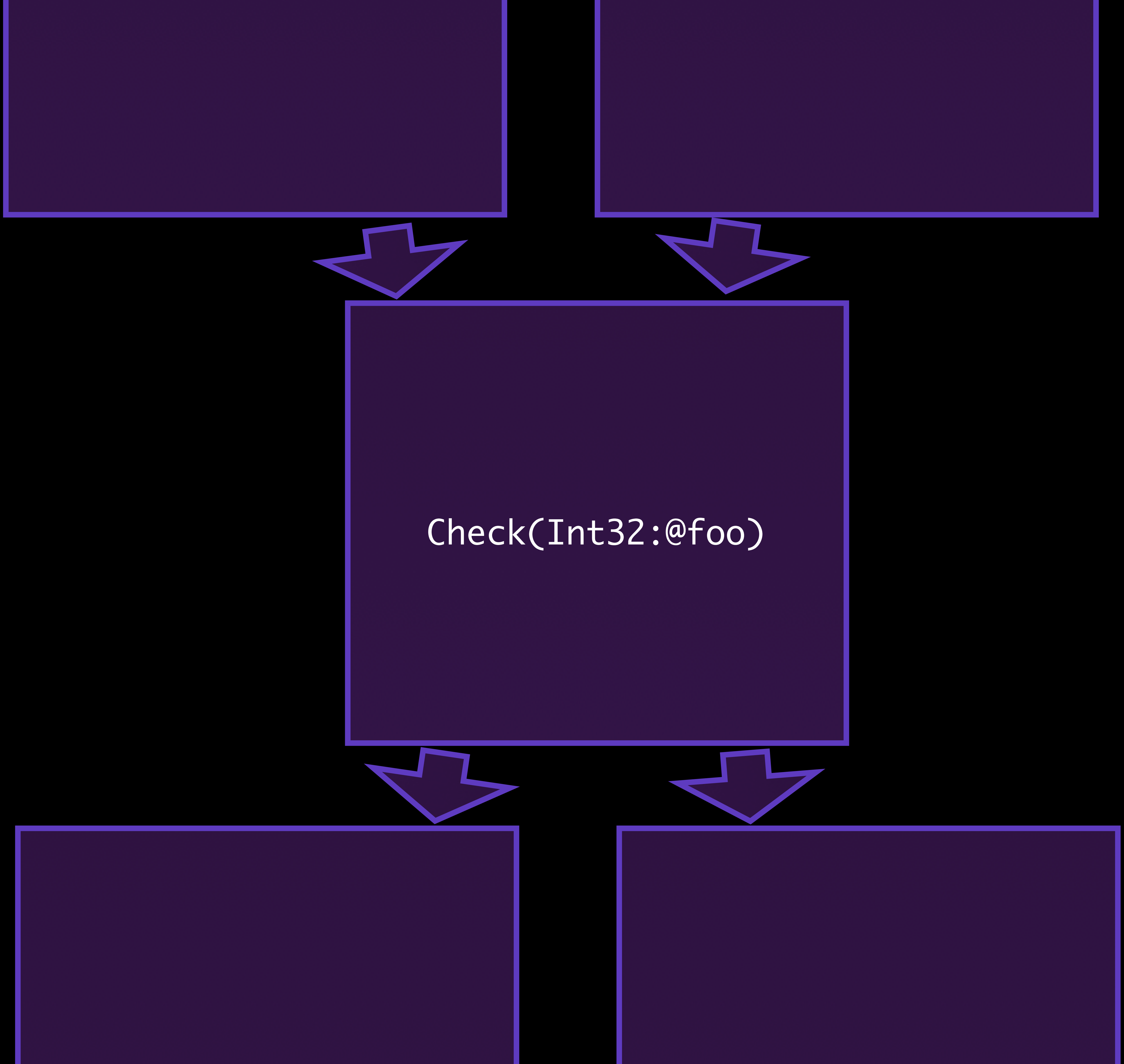
Watchpoints
+
InvalidationPoint

DFG Goals

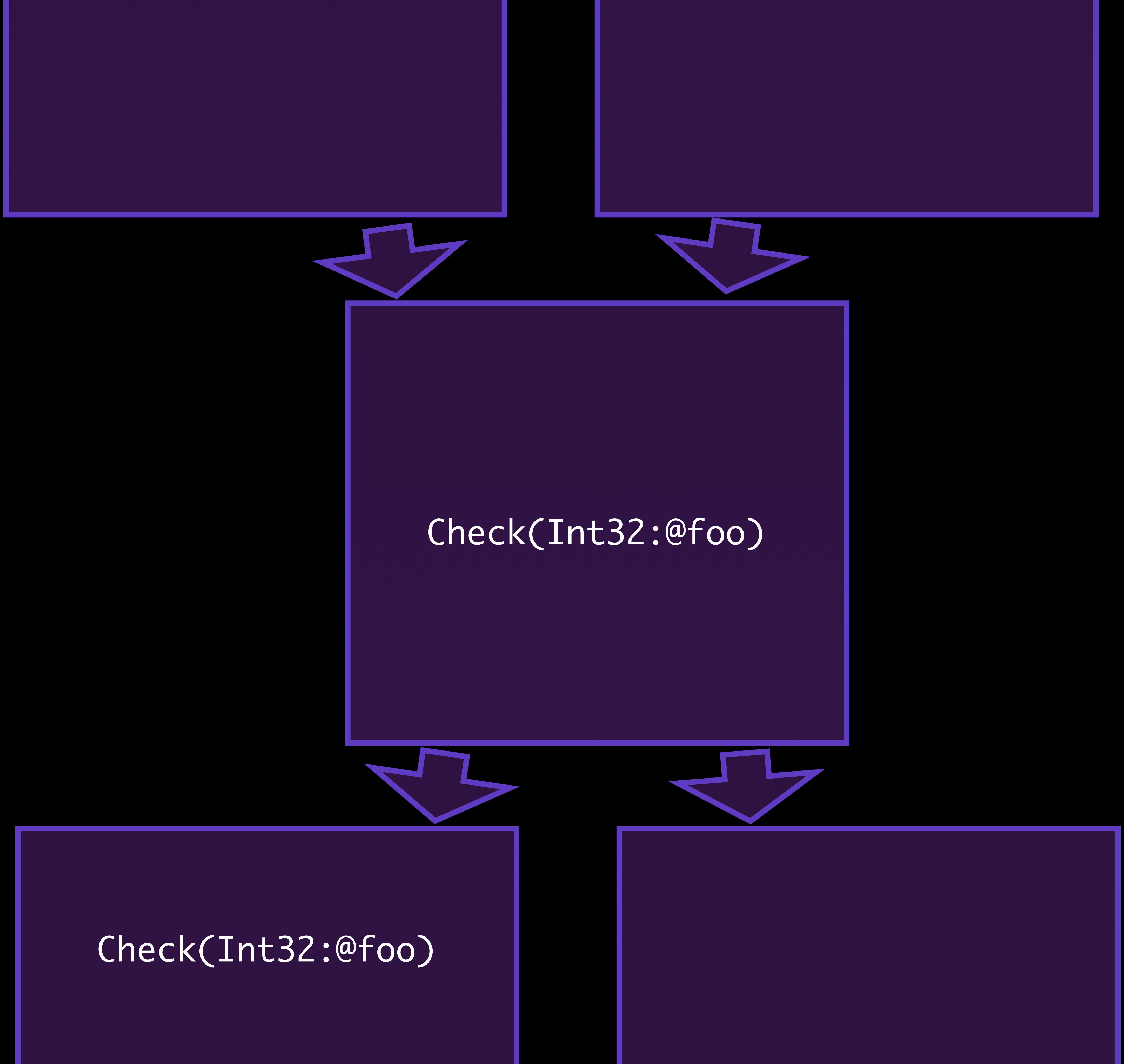
- Speculation
- Static Analysis
- Fast Compilation

Remove type checks



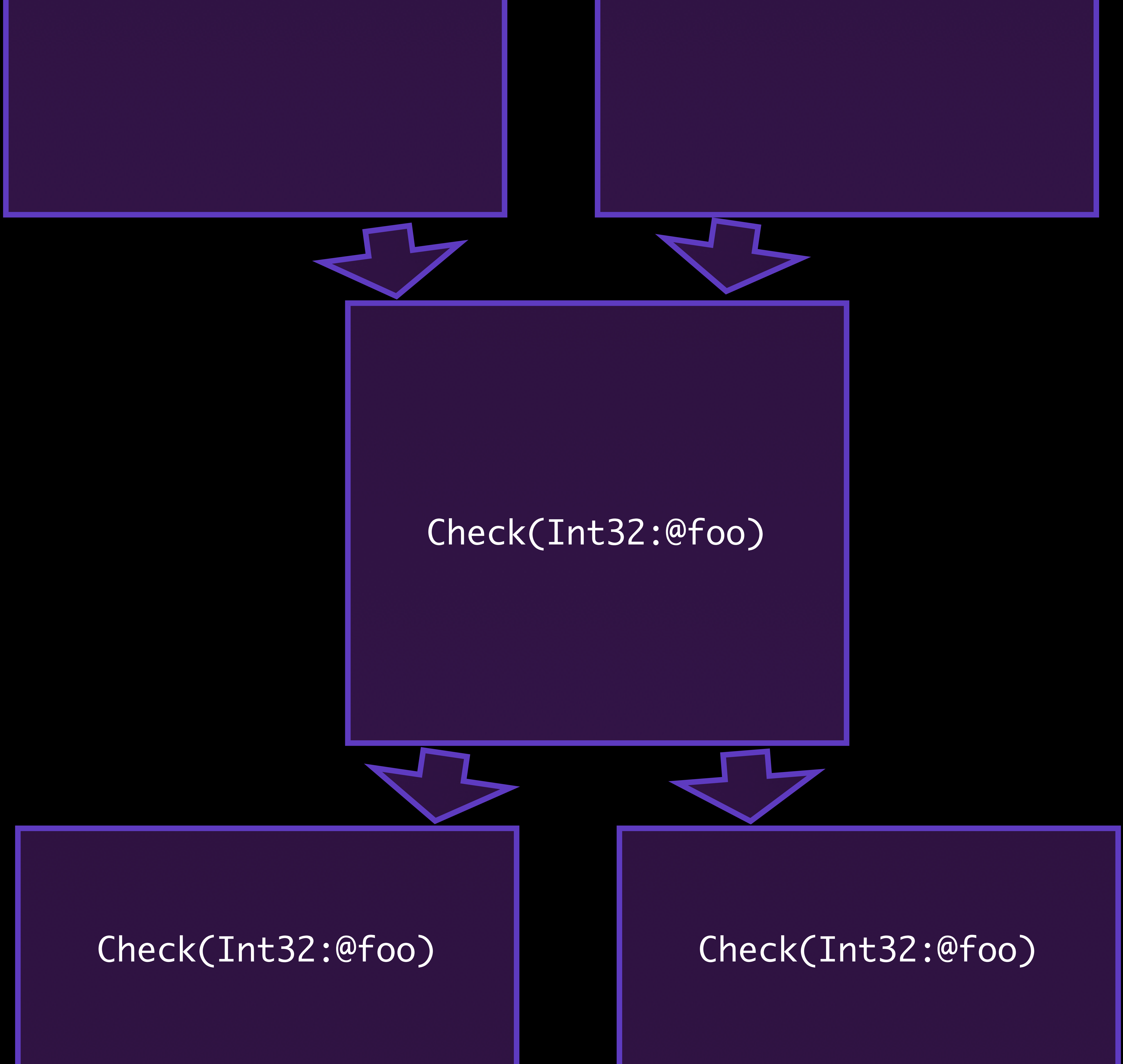


Check(Int32:@foo)



Check(Int32:@foo)

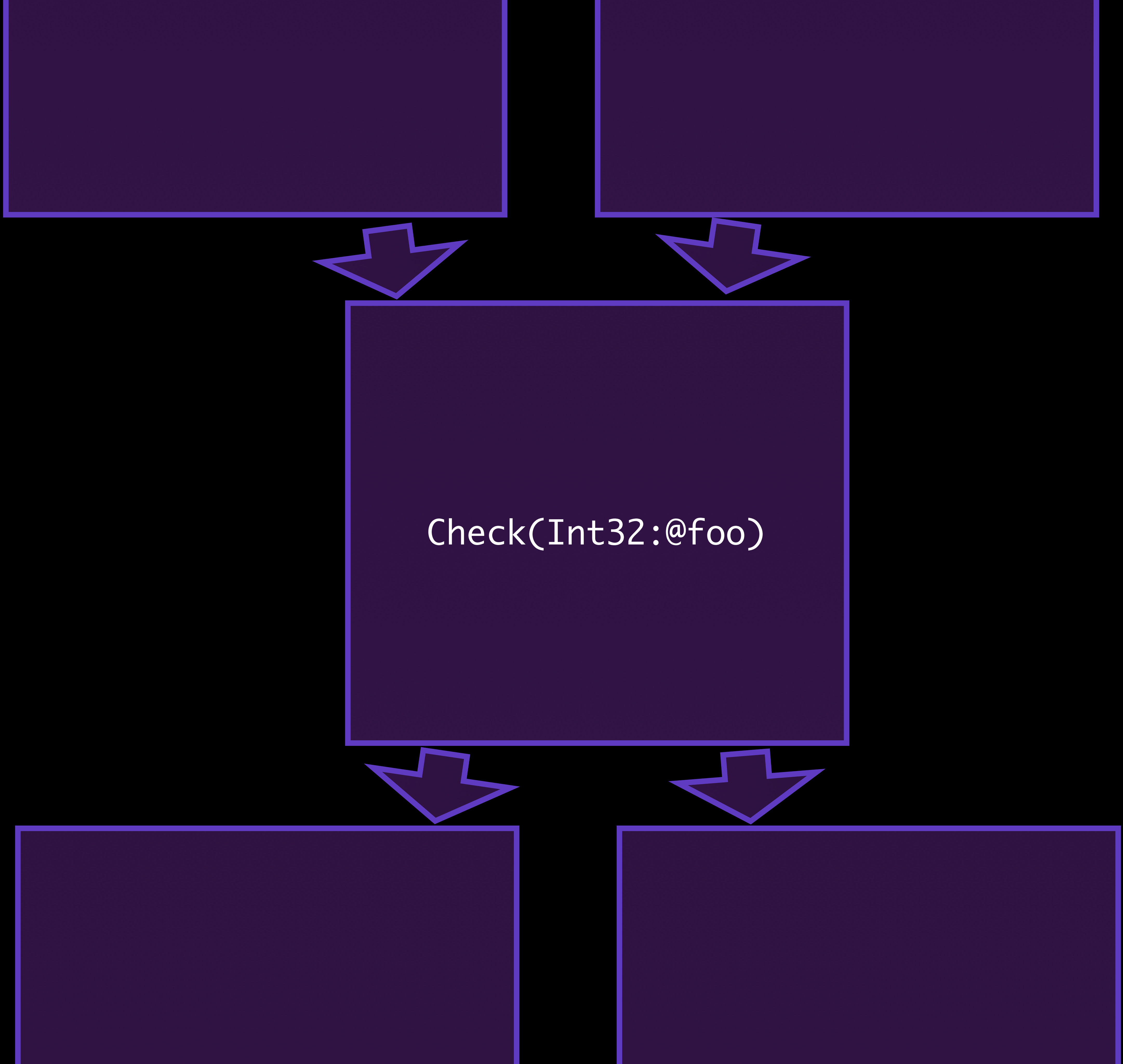
Check(Int32:@foo)



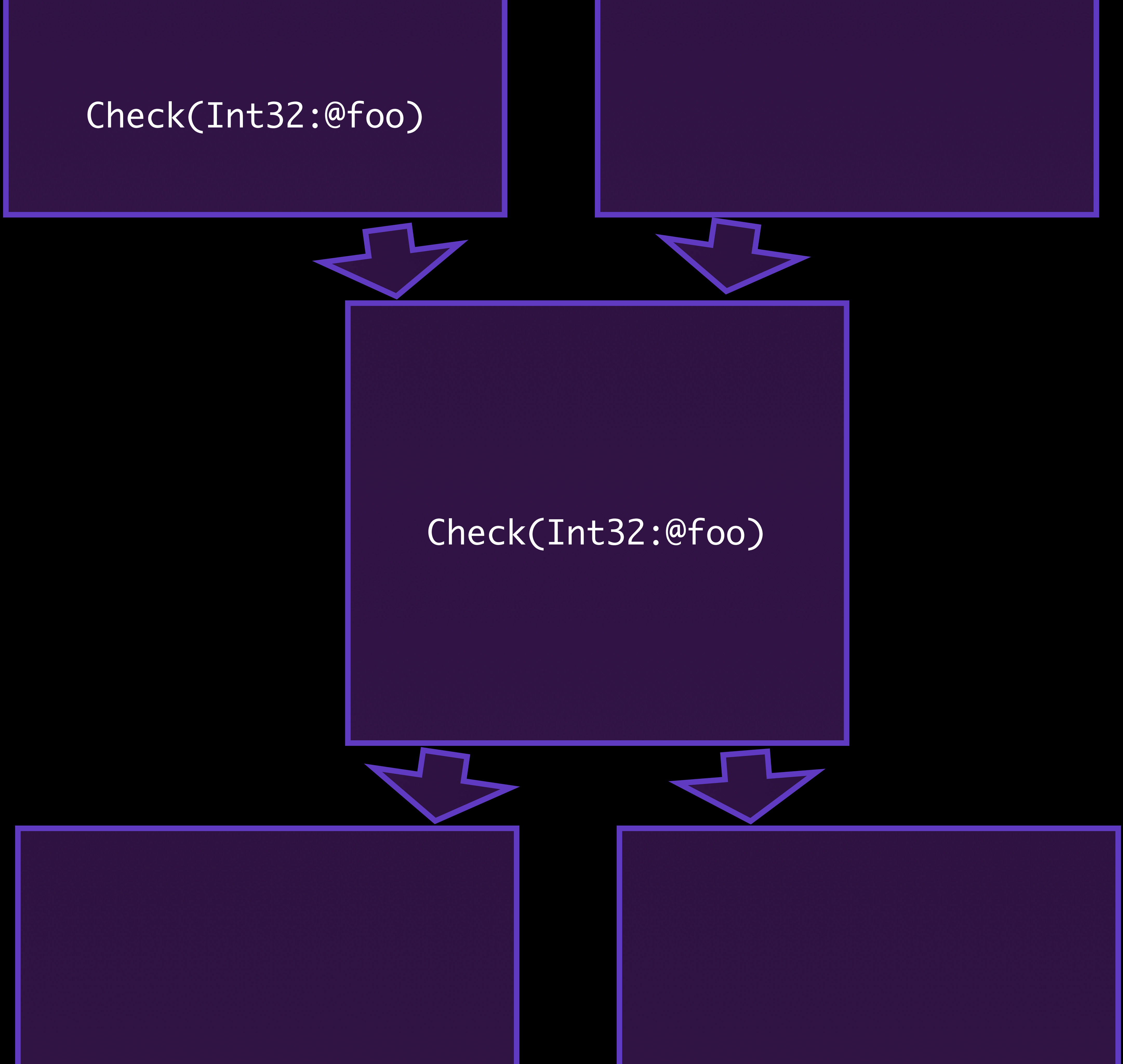
Check(Int32:@foo)

Check(Int32:@foo)

Check(Int32:@foo)

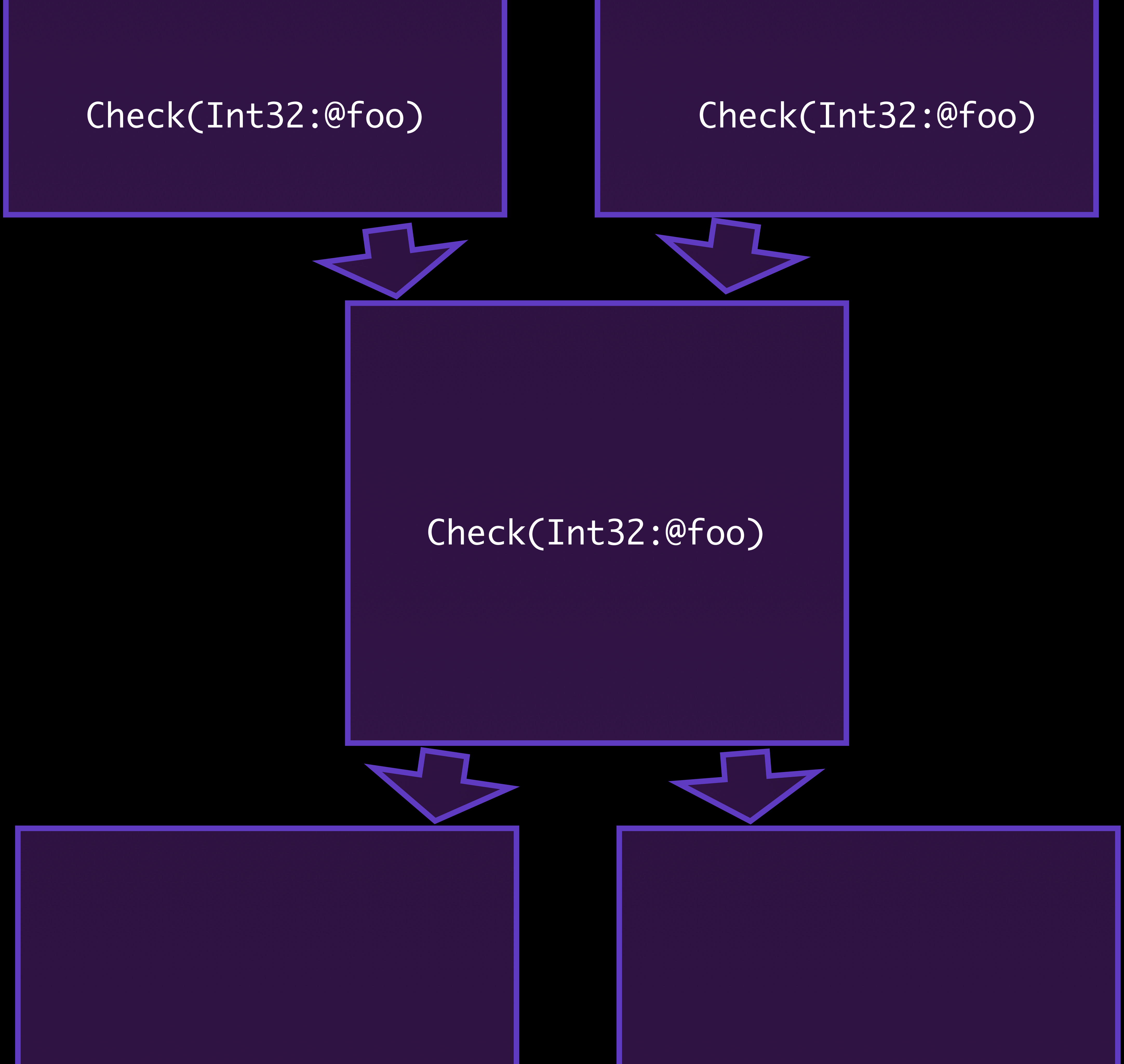


Check(Int32:@foo)



Check(Int32:@foo)

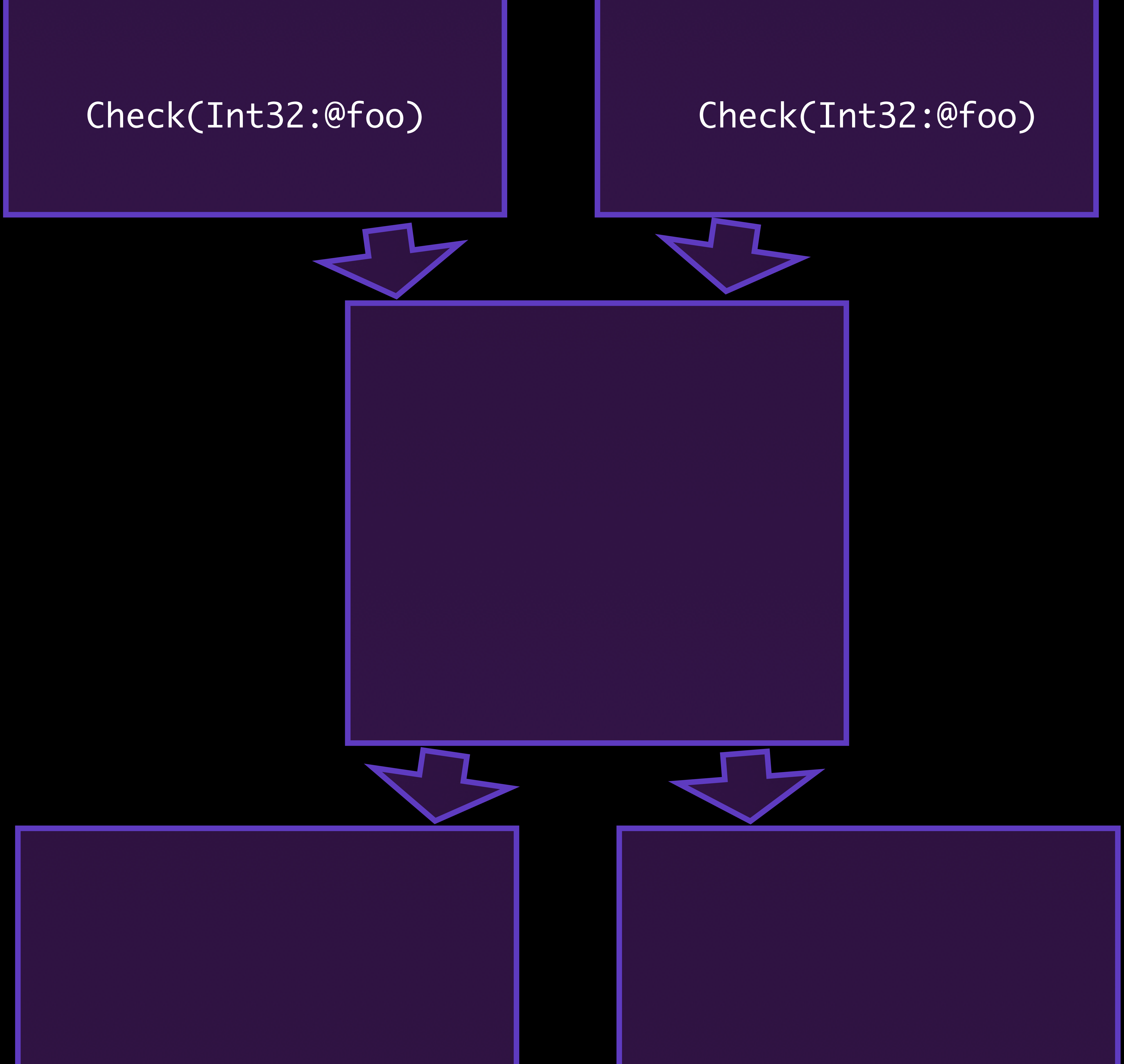
Check(Int32:@foo)



Check(Int32:@foo)

Check(Int32:@foo)

Check(Int32:@foo)



Check(Int32:@foo)

Check(Int32:@foo)

Abstract Interpreter

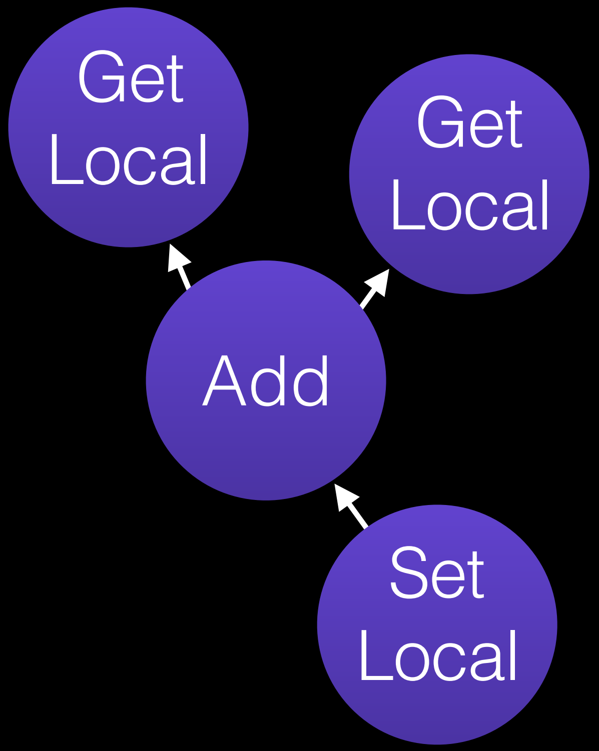
- “Global” (whole compilation unit)
- Flow sensitive
- Tracks:
    - variable type
    - object structure
    - indexing type
    - constants

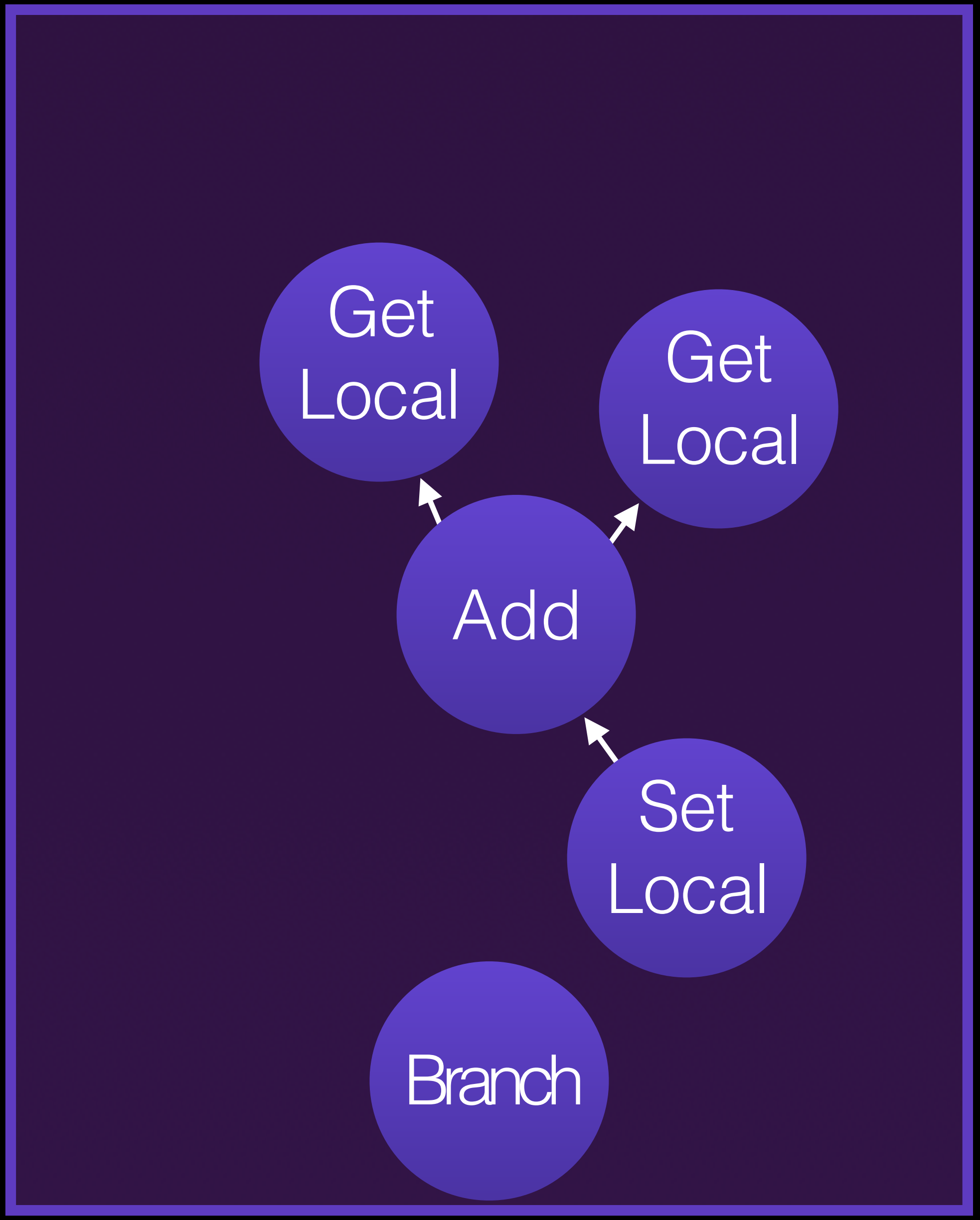
DFG Goals

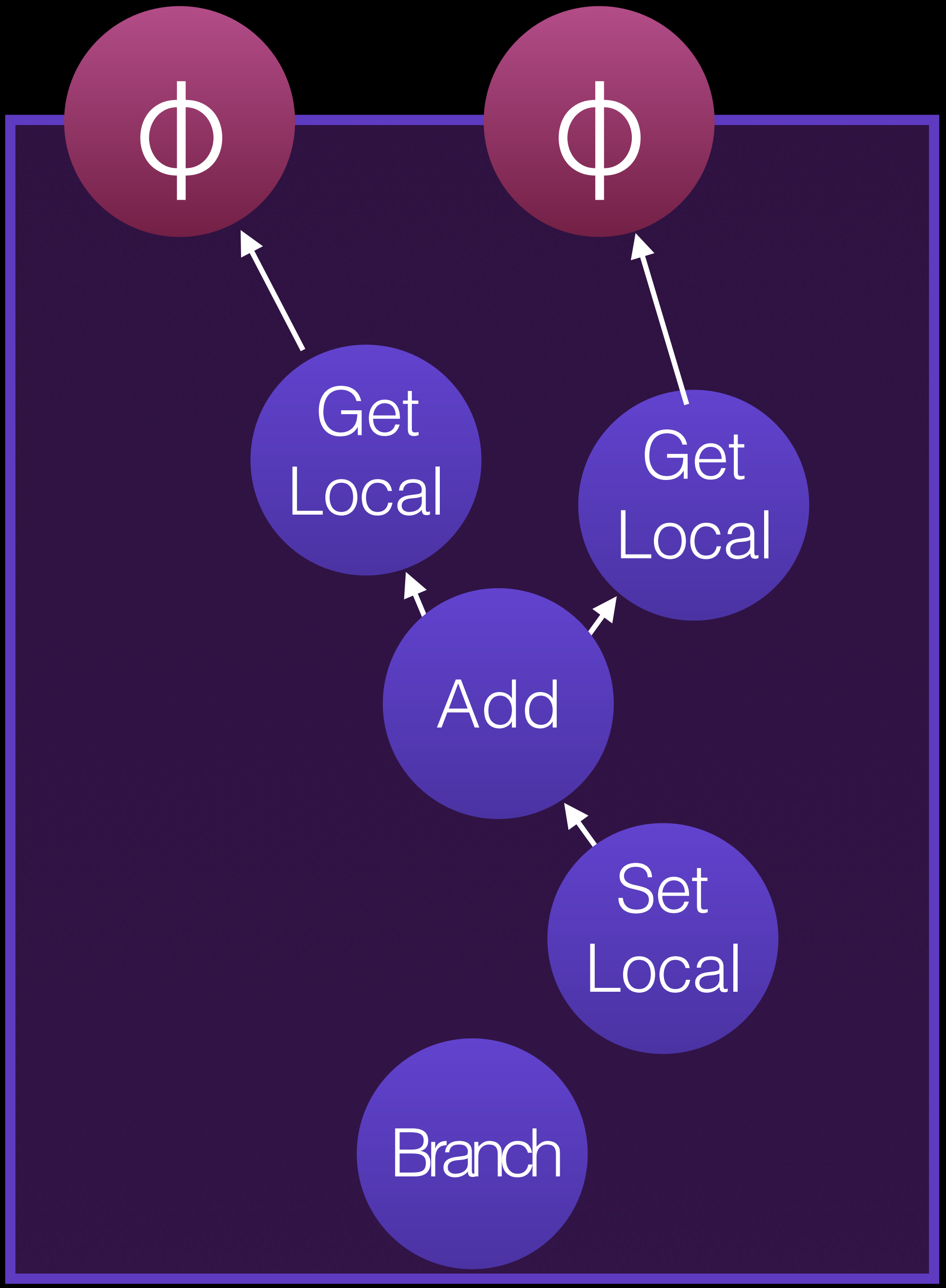
- Speculation
- Static Analysis
- Fast Compilation

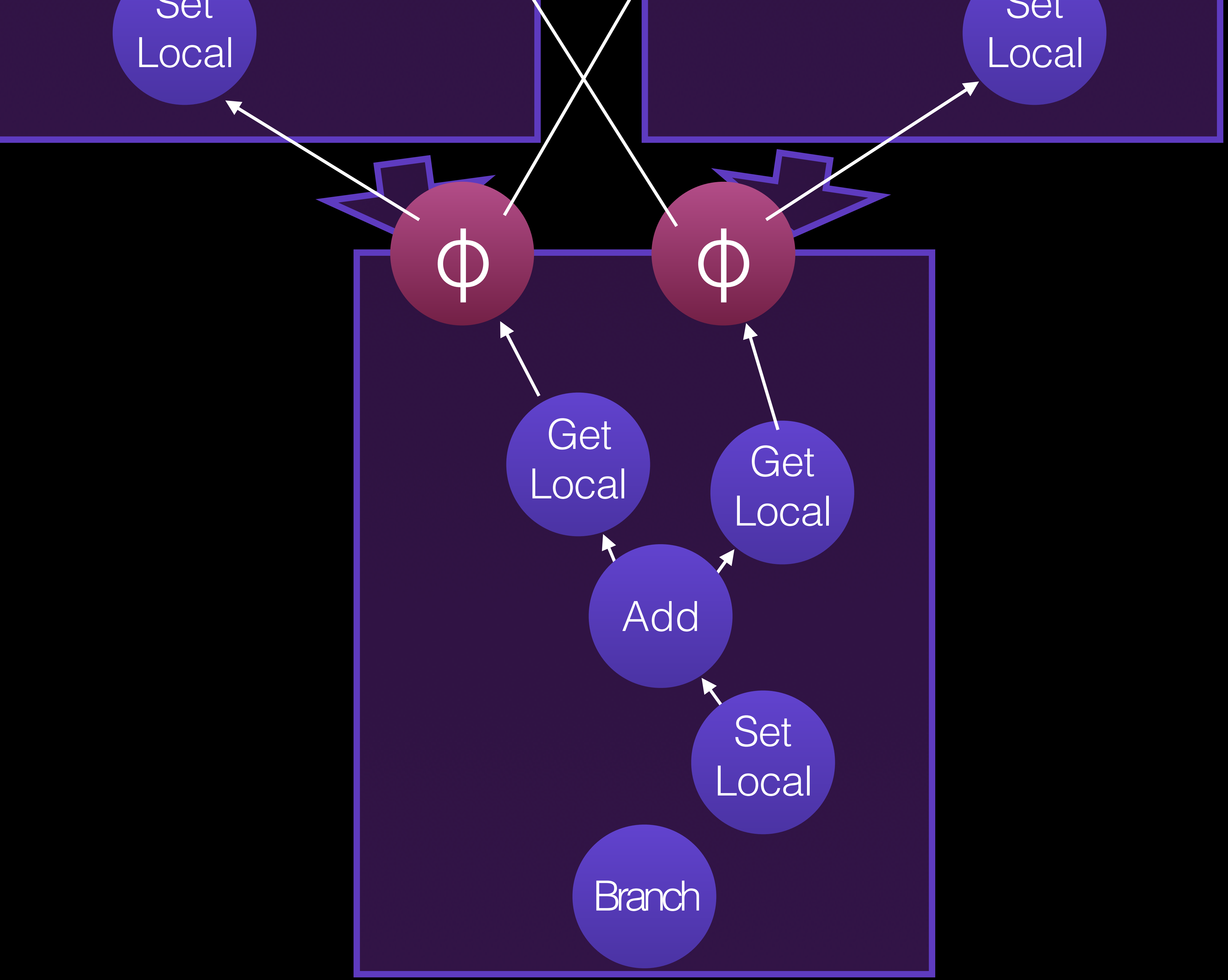
Fast Compile

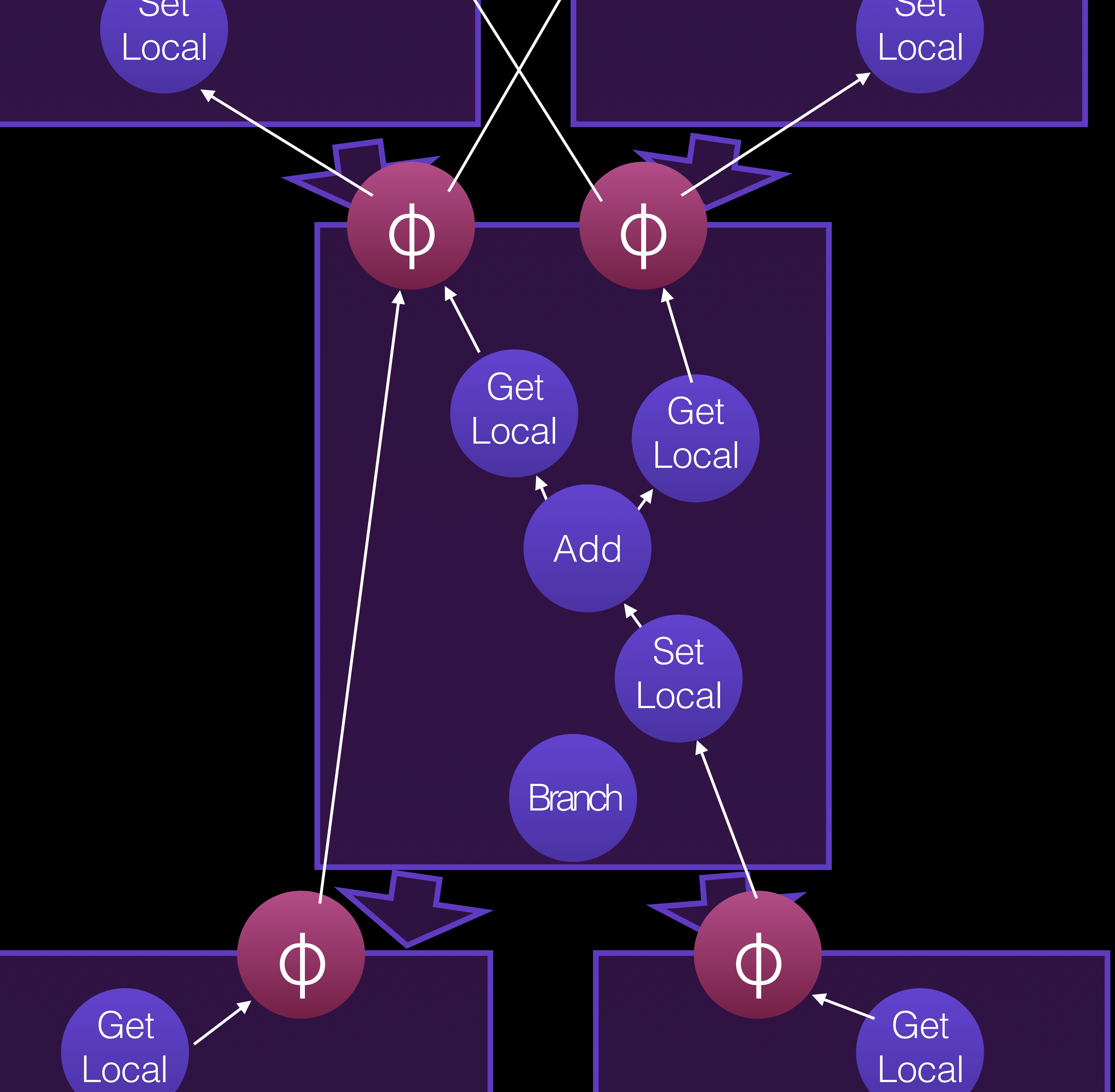
- Emphasis on block-locality.
- Template code generation.

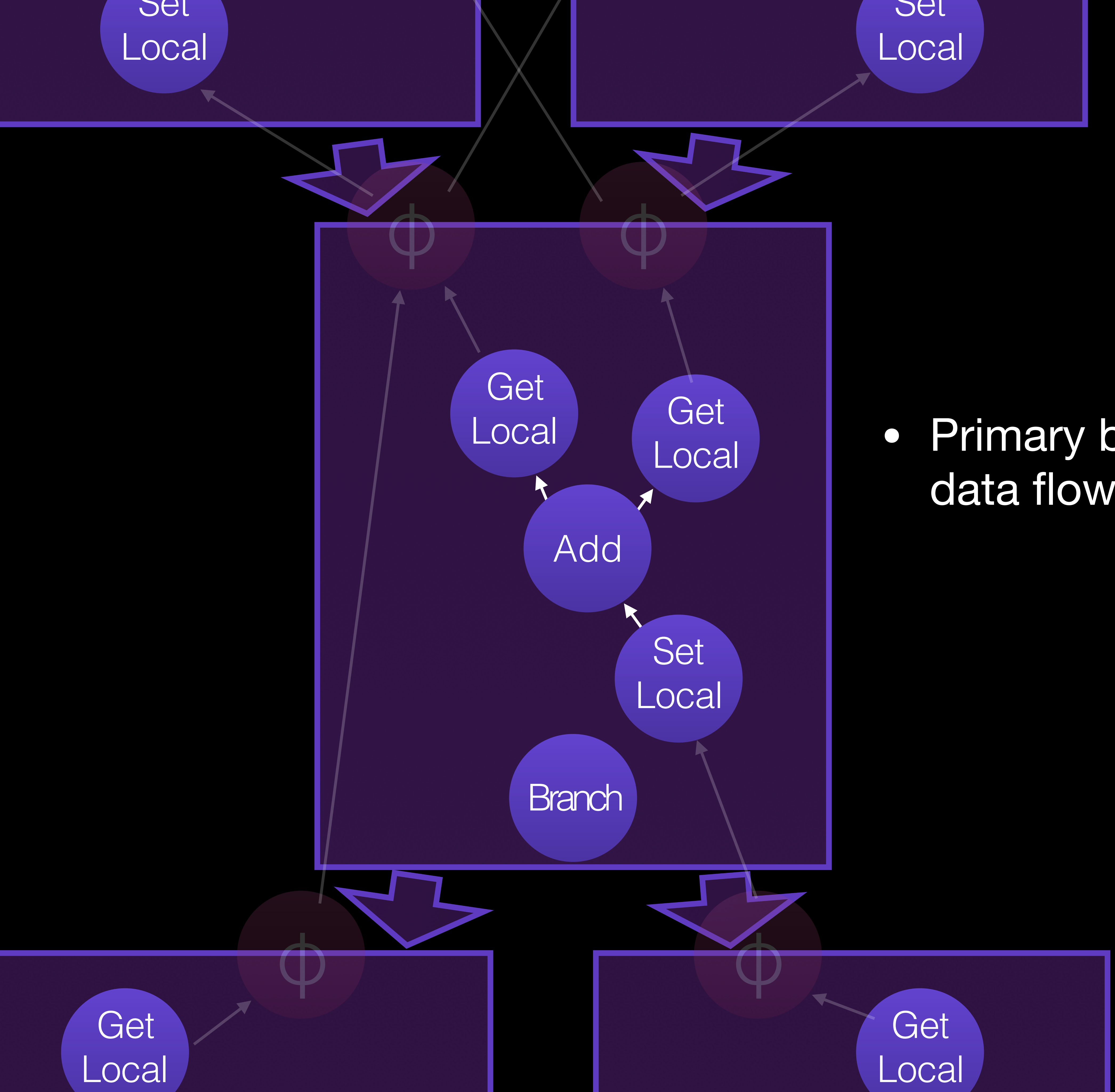

Get Local
Get Local
Add
Set Local


Get Local
Get Local
Add
Set Local
Branch

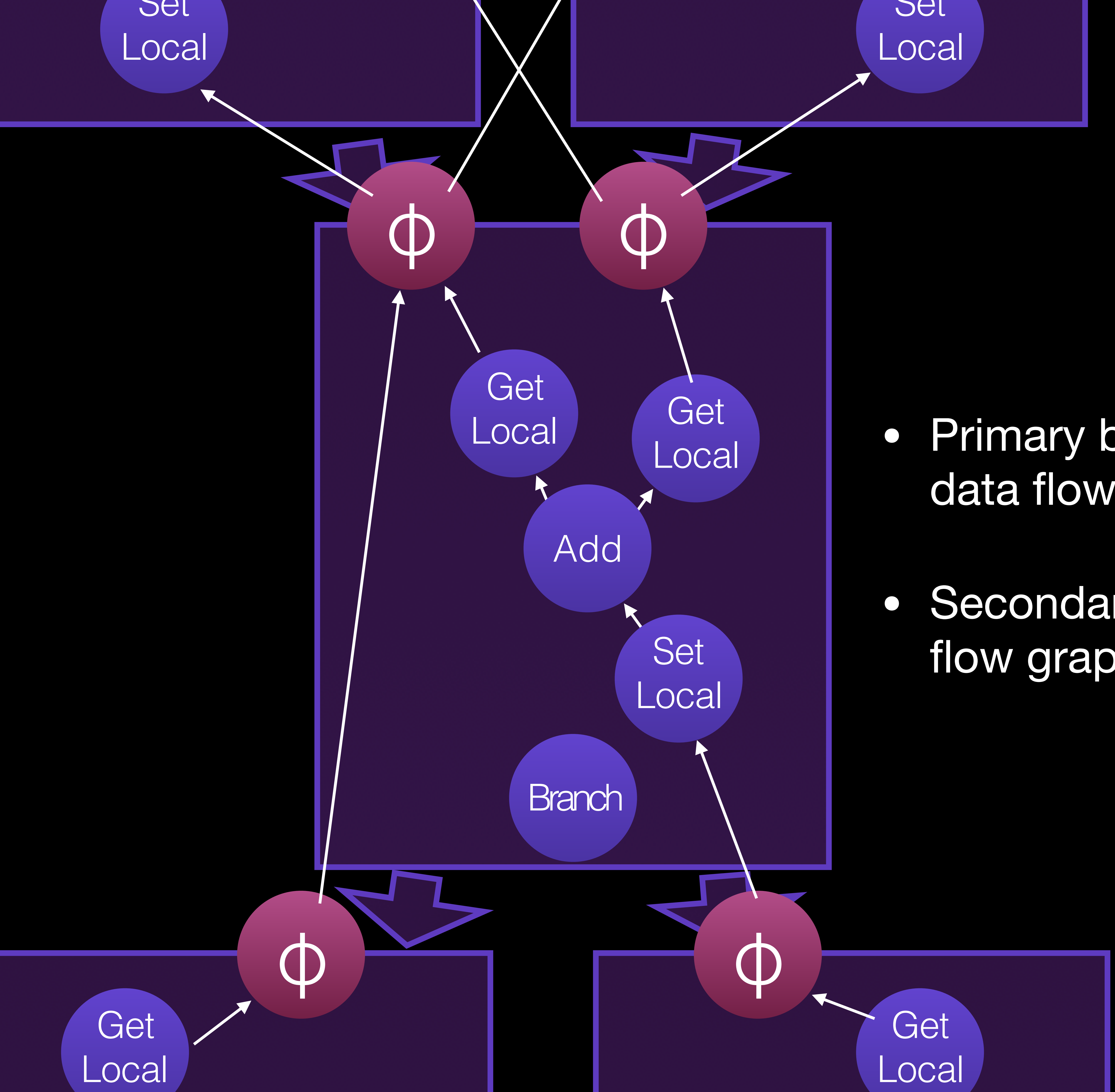

ϕ
ϕ
Get Local
Get Local
Add
Set Local
Branch


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.

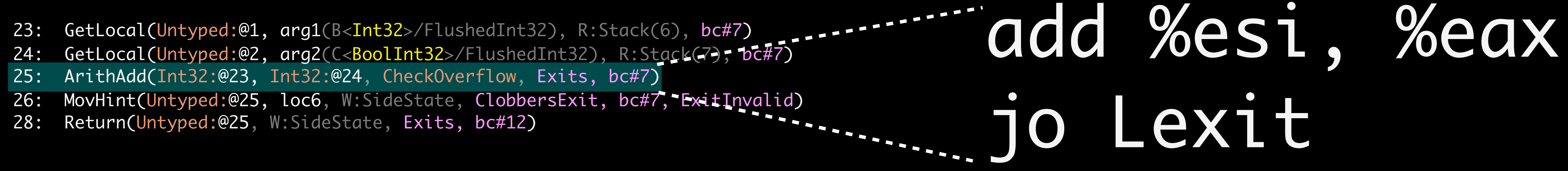
DFG Template Codegen

```
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)
```

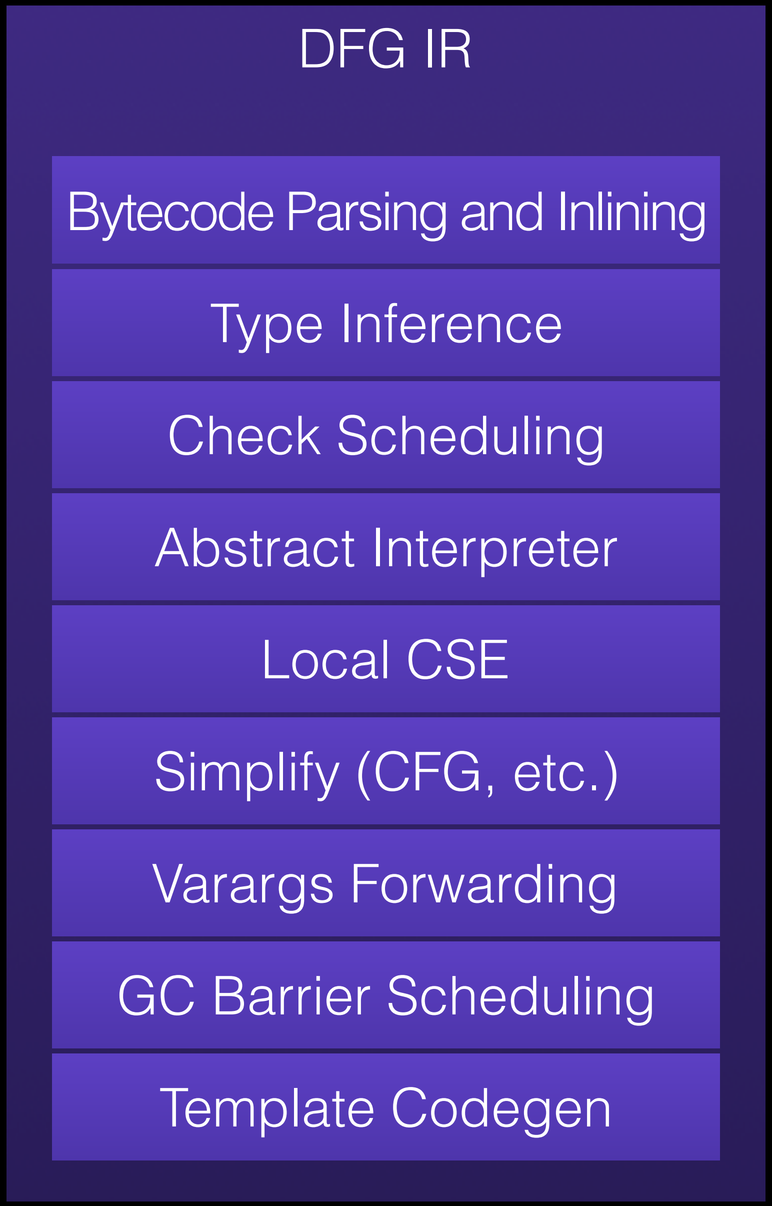
DFG Template Codegen



```
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 IR

Bytecode Parsing and Inlining

Type Inference

Check Scheduling

Abstract Interpreter

Local CSE

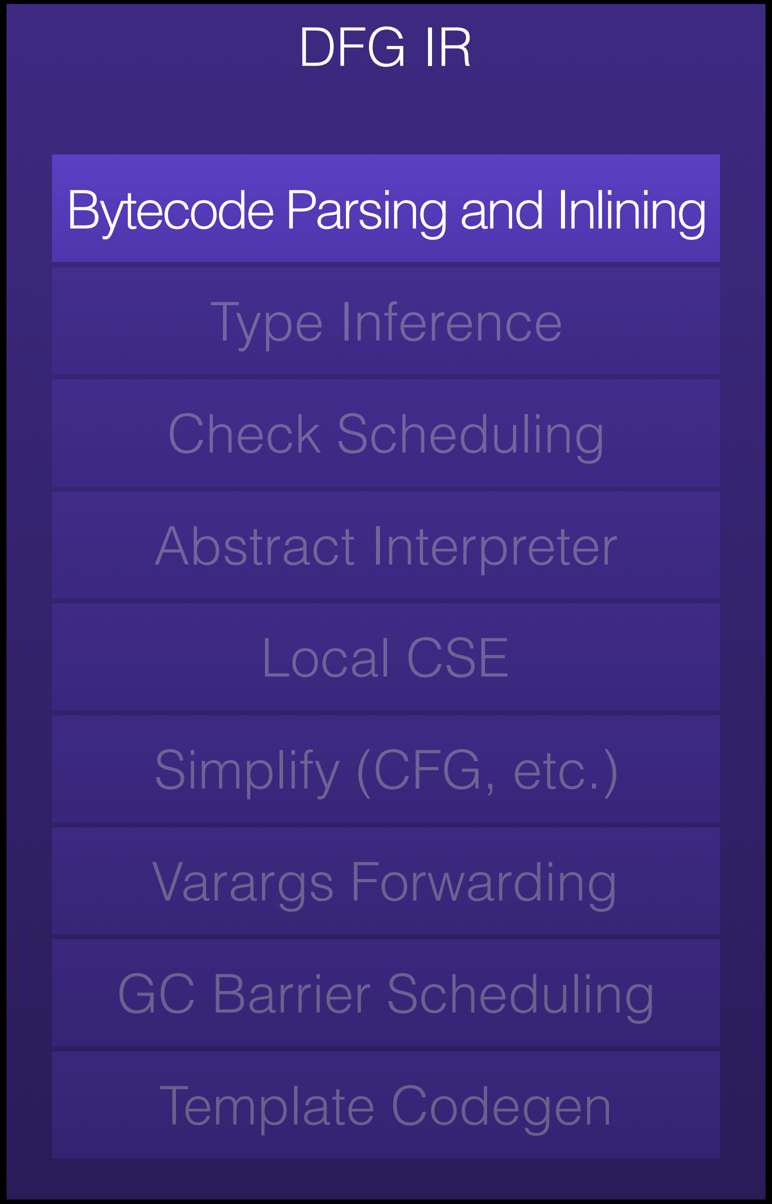
Simplify (CFG, etc.)

Varargs Forwarding

GC Barrier Scheduling

Template Codegen

DFG optimization pipeline



DFG IR

Bytecode Parsing and Inlining

Type Inference

Check Scheduling

Abstract Interpreter

Local CSE

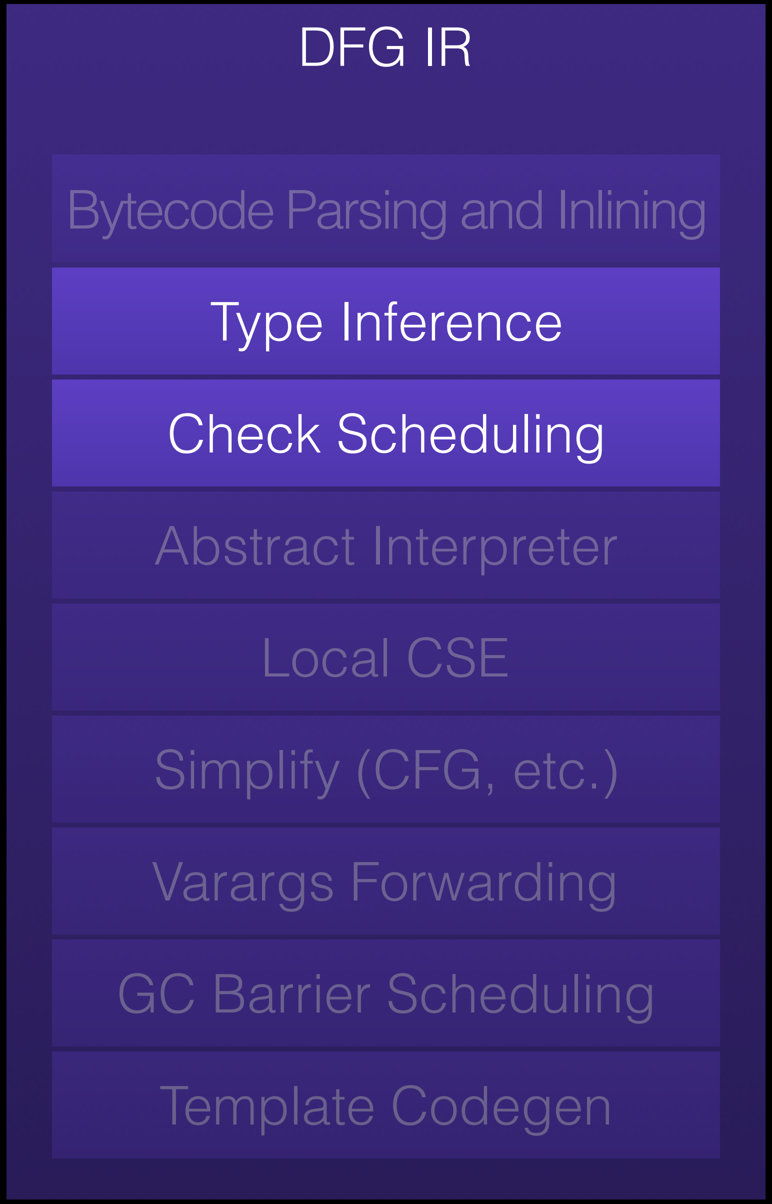
Simplify (CFG, etc.)

Varargs Forwarding

GC Barrier Scheduling

Template Codegen

DFG optimization pipeline



DFG IR

Bytecode Parsing and Inlining

Type Inference

Check Scheduling

Abstract Interpreter

Local CSE

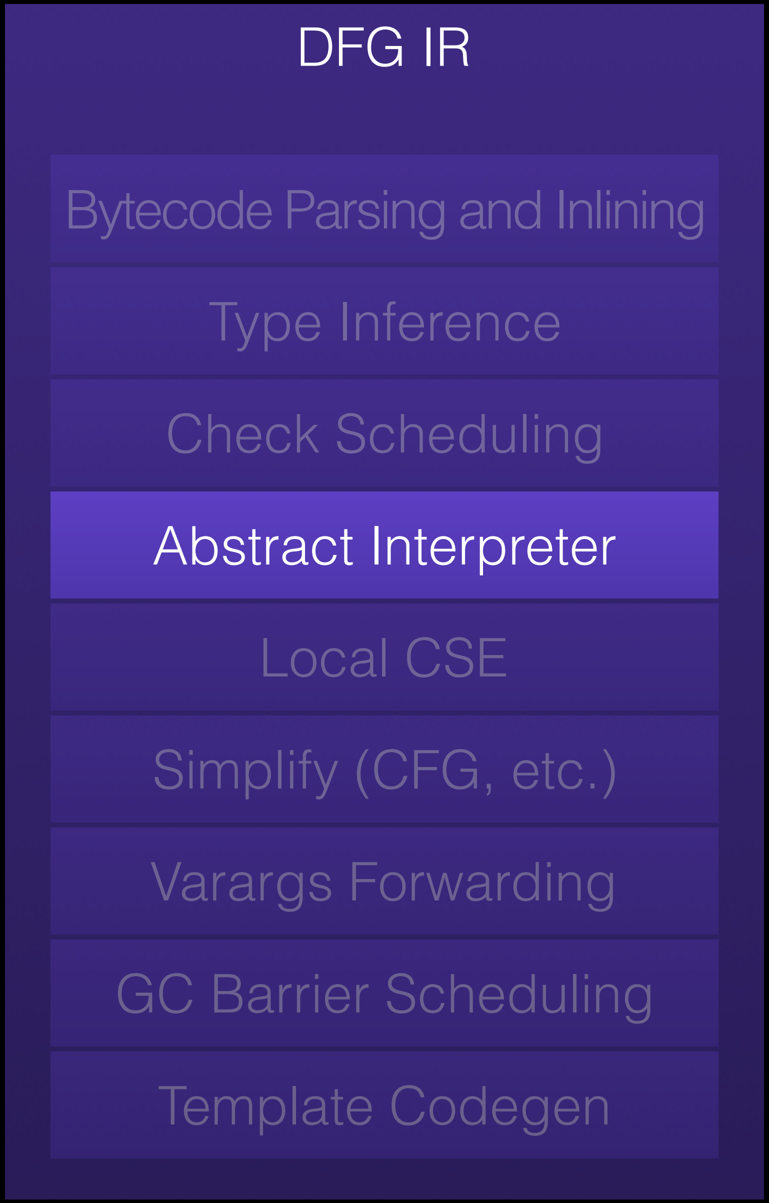
Simplify (CFG, etc.)

Varargs Forwarding

GC Barrier Scheduling

Template Codegen

DFG optimization pipeline



DFG IR

Bytecode Parsing and Inlining

Type Inference

Check Scheduling

Abstract Interpreter

Local CSE

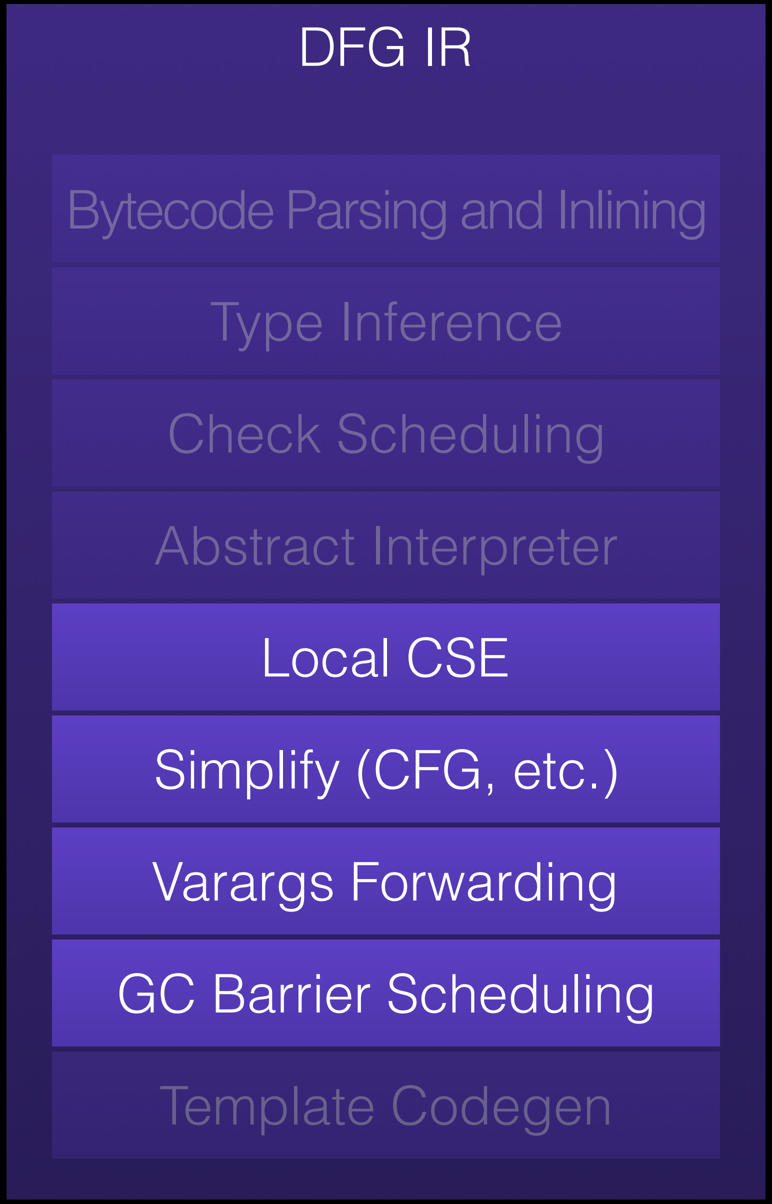
Simplify (CFG, etc.)

Varargs Forwarding

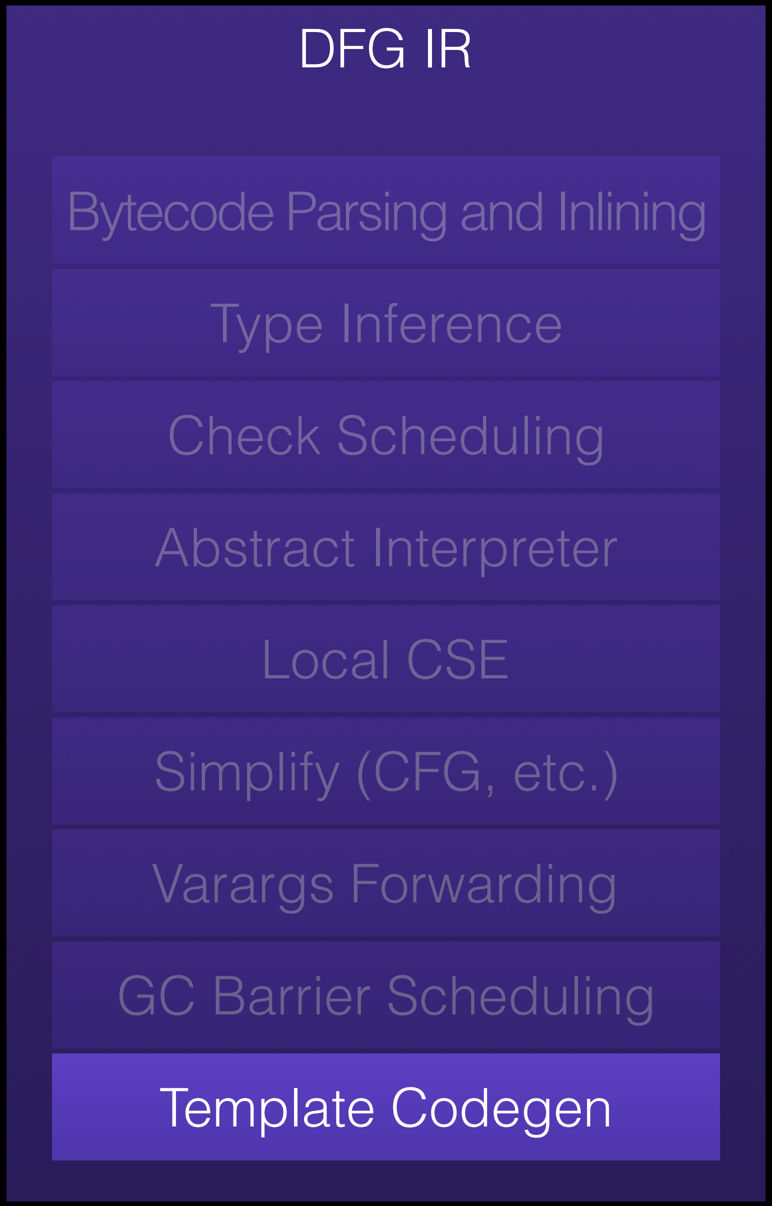
GC Barrier Scheduling

Template Codegen

DFG optimization pipeline



DFG optimization pipeline



DFG IR

Bytecode Parsing and Inlining

Type Inference

Check Scheduling

Abstract Interpreter

Local CSE

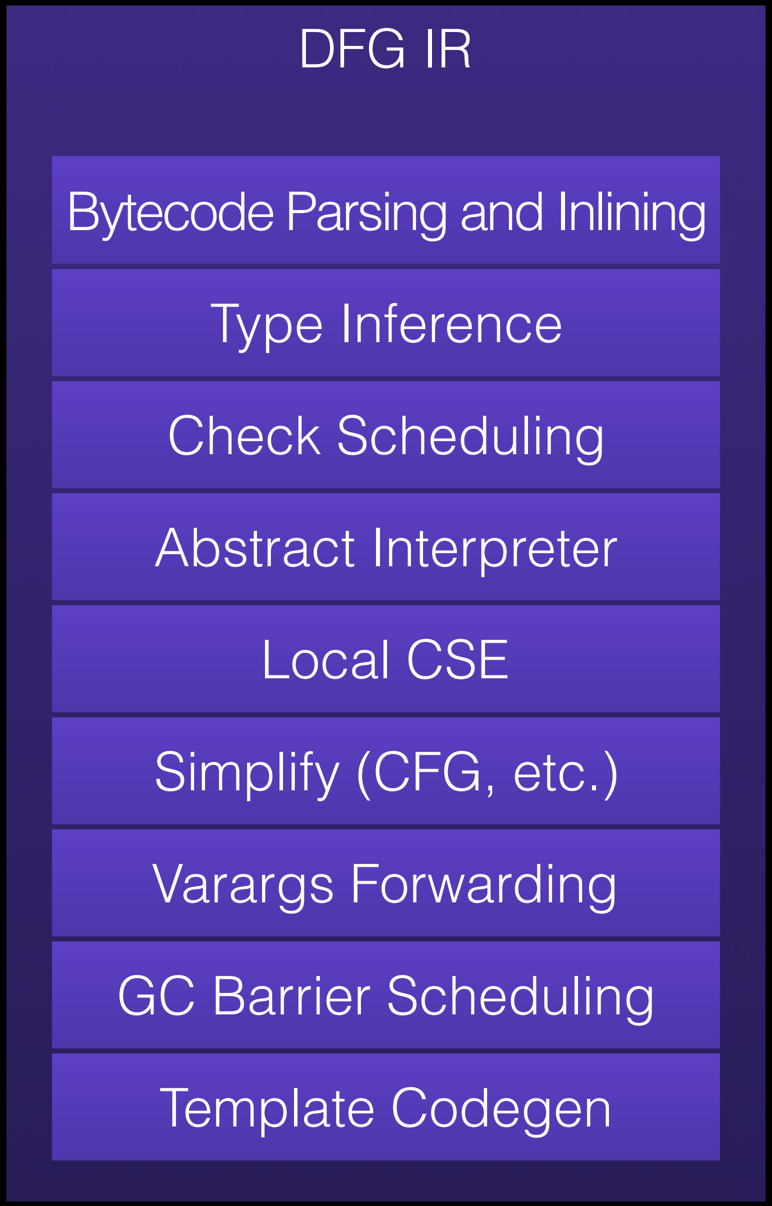
Simplify (CFG, etc.)

Varargs Forwarding

GC Barrier Scheduling

Template Codegen

DFG optimization pipeline



DFG IR

Bytecode Parsing and Inlining

Type Inference

Check Scheduling

Abstract Interpreter

Local CSE

Simplify (CFG, etc.)

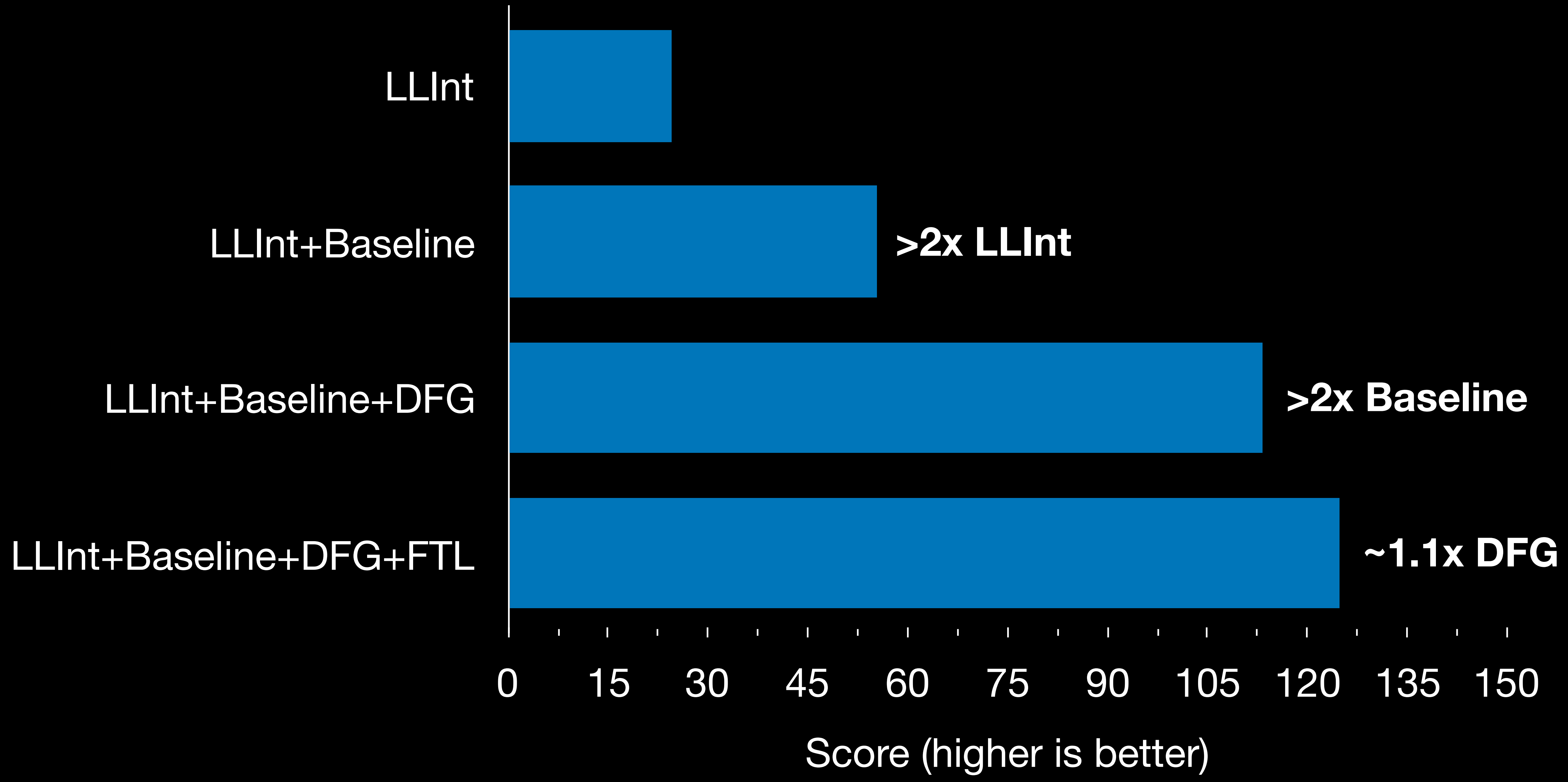
Varargs Forwarding

GC Barrier Scheduling

Template Codegen

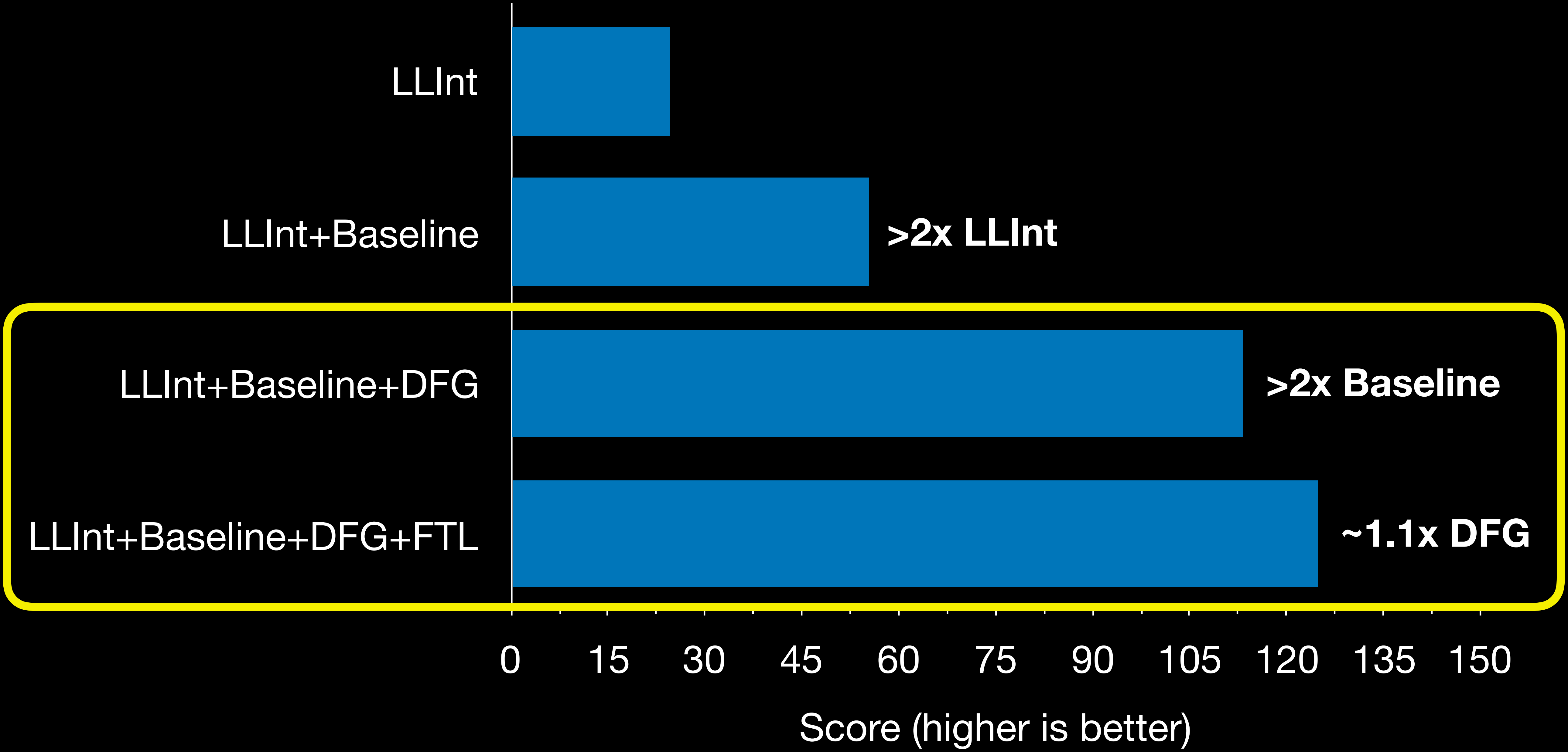
JetStream 2 Score

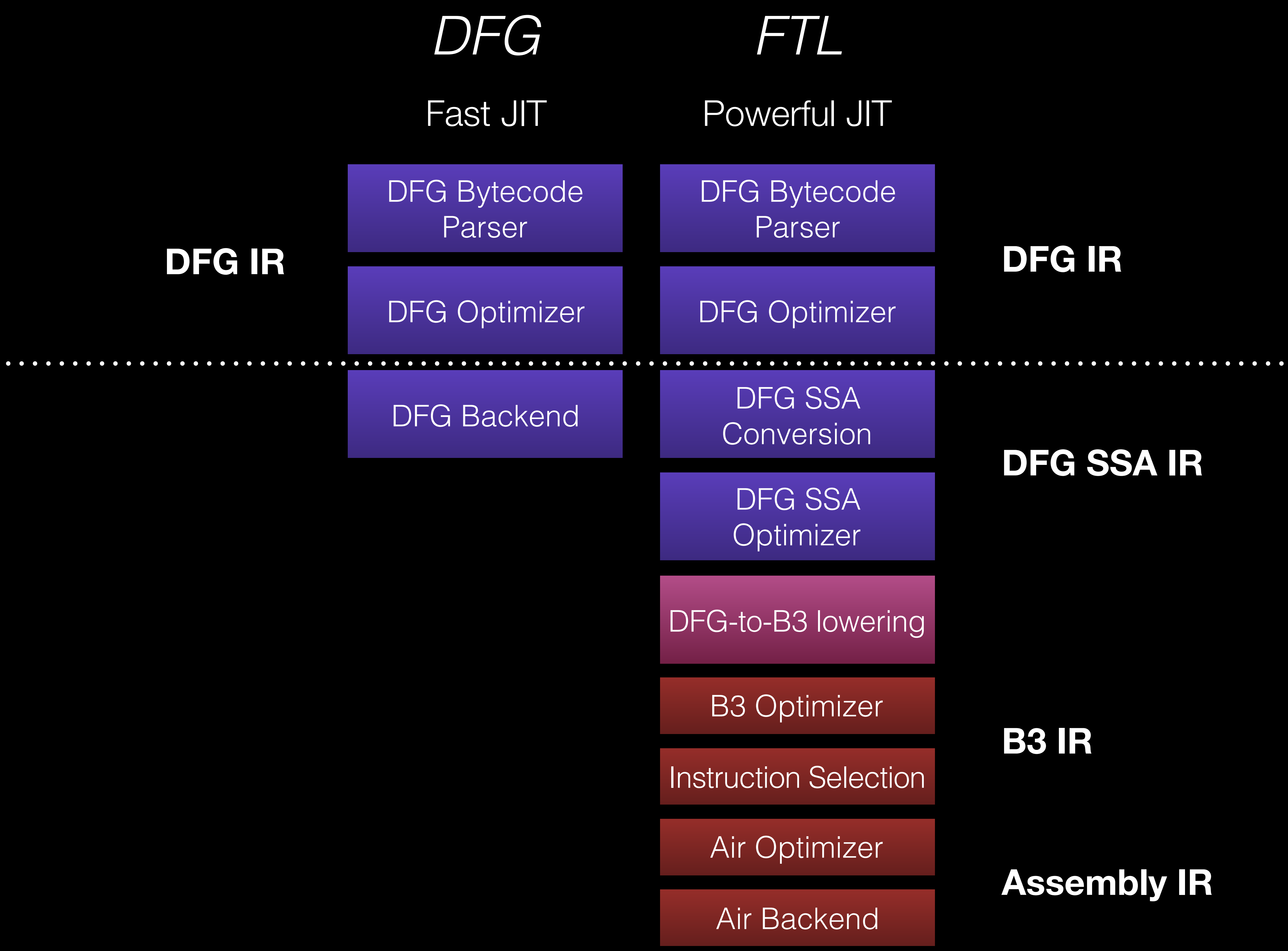
on my computer one day



JetStream 2 Score

on my computer one day





DFG

Fast JIT

DFG Bytecode Parser

DFG Optimizer

DFG Backend

FTL

Powerful JIT

DFG Bytecode Parser

DFG Optimizer

DFG SSA Conversion

DFG SSA Optimizer

DFG-to-B3 lowering

B3 Optimizer

Instruction Selection

Air Optimizer

Air Backend

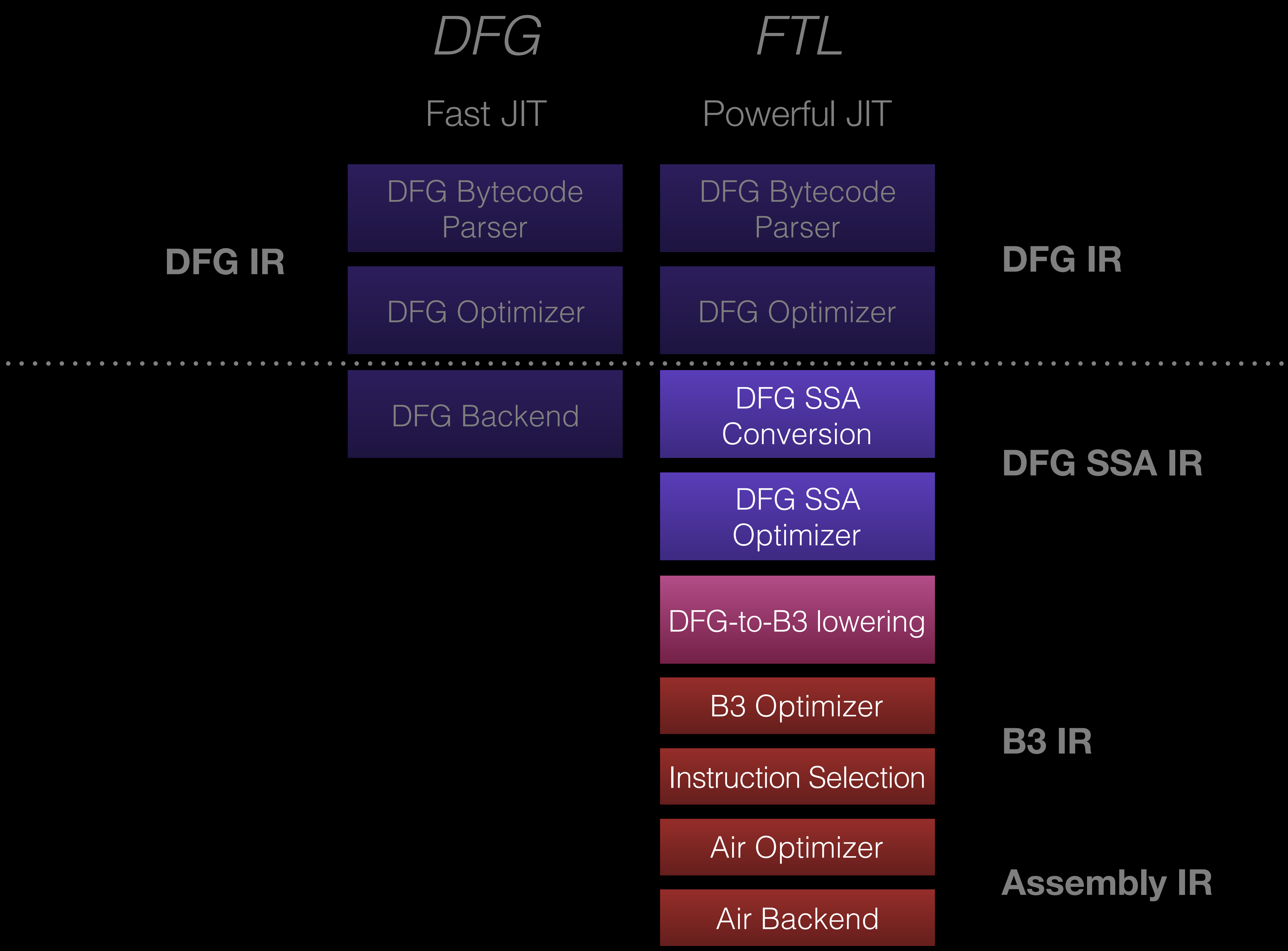
DFG IR

DFG IR

DFG SSA IR

B3 IR

Assembly IR



DFG

Fast JIT

DFG Bytecode Parser

DFG Optimizer

DFG Backend

FTL

Powerful JIT

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 IR

DFG SSA IR

B3 IR

Assembly IR

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

- Bytecode Parsing and Inlining
- Type Inference
- Check Scheduling
- Simplify (CFG etc)
- Abstract Interpretation
- Global CSE
- Escape Analysis
- LICM
- Integer Range Optimization
- GC Barrier Scheduling
- Lower to B3 IR

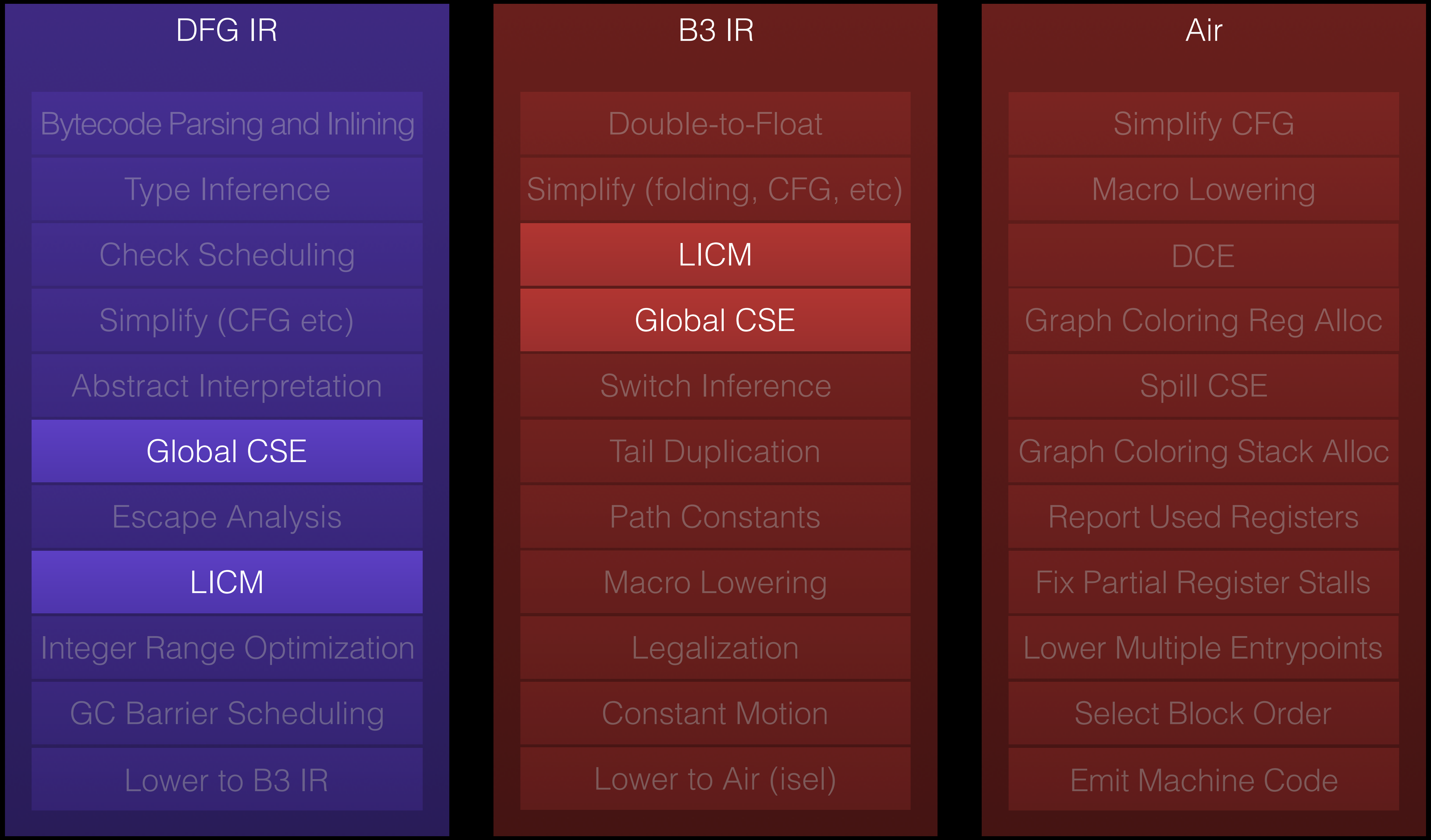
B3 IR

- Double-to-Float
- Simplify (folding, CFG, etc)
- LICM
- Global CSE
- Switch Inference
- Tail Duplication
- Path Constants
- Macro Lowering
- Legalization
- Constant Motion
- Lower to Air (isel)

Air

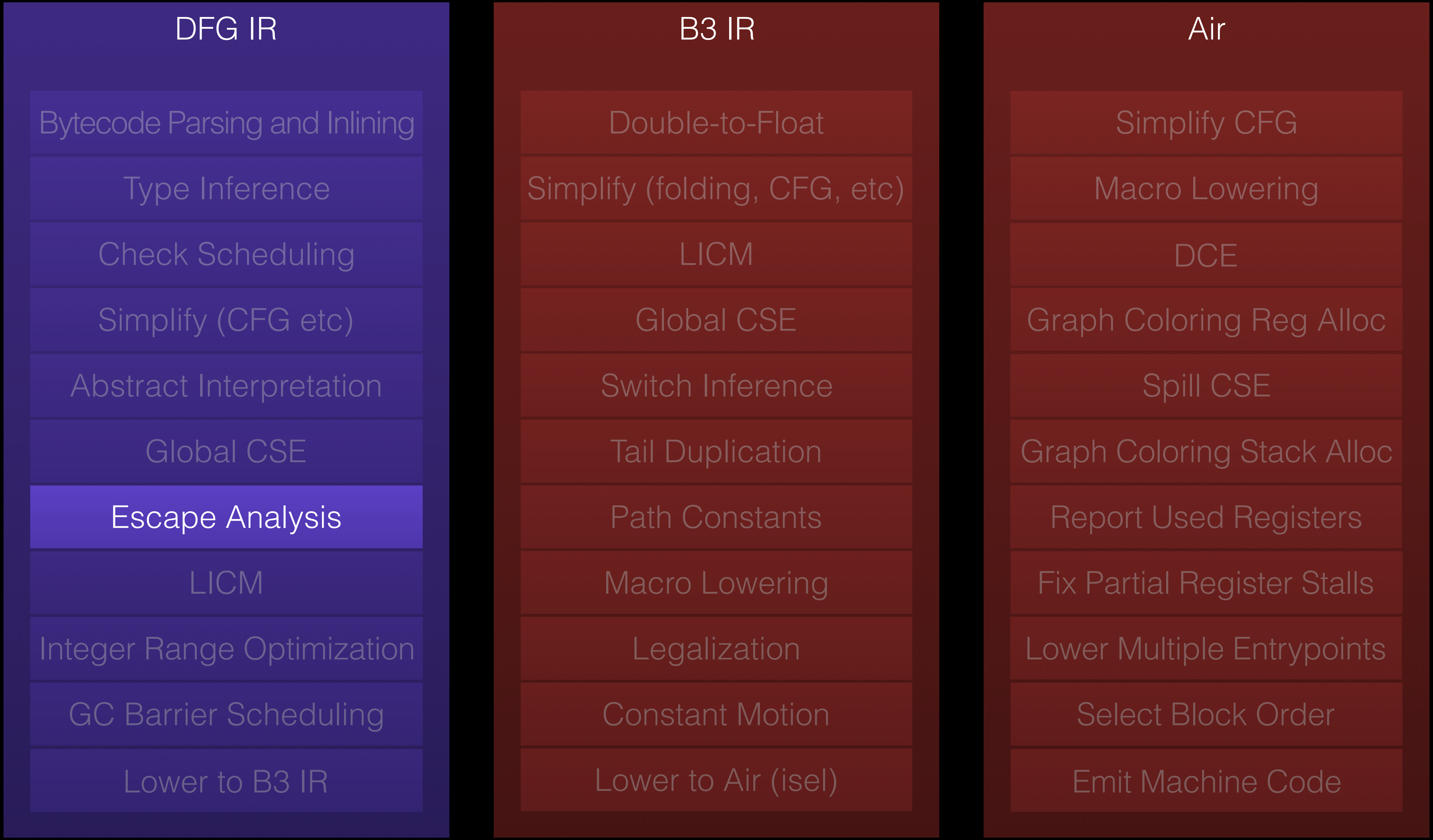
- Simplify CFG
- Macro Lowering
- DCE
- Graph Coloring Reg Alloc
- Spill CSE
- Graph Coloring Stack Alloc
- Report Used Registers
- Fix Partial Register Stalls
- Lower Multiple Entrypoints
- Select Block Order
- Emit Machine Code

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

- Bytecode Parsing and Inlining
- Type Inference
- Check Scheduling
- Simplify (CFG etc)
- Abstract Interpretation
- Global CSE
- Escape Analysis
- LICM
- Integer Range Optimization
- GC Barrier Scheduling
- Lower to B3 IR

B3 IR

- Double-to-Float
- Simplify (folding, CFG, etc)
- LICM
- Global CSE
- Switch Inference
- **Tail Duplication**
- Path Constants
- Macro Lowering
- Legalization
- Constant Motion
- Lower to Air (isel)

Air

- Simplify CFG
- Macro Lowering
- DCE
- Graph Coloring Reg Alloc
- Spill CSE
- Graph Coloring Stack Alloc
- Report Used Registers
- Fix Partial Register Stalls
- Lower Multiple Entrypoints
- Select Block Order
- 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

- Bytecode Parsing and Inlining
- Type Inference
- Check Scheduling
- Simplify (CFG etc)
- Abstract Interpretation
- Global CSE
- Escape Analysis
- LICM
- Integer Range Optimization
- GC Barrier Scheduling
- Lower to B3 IR

B3 IR

- Double-to-Float
- Simplify (folding, CFG, etc)
- LICM
- Global CSE
- Switch Inference
- Tail Duplication
- Path Constants
- Macro Lowering
- Legalization
- Constant Motion
- Lower to Air (isel)

Air

- Simplify CFG
- Macro Lowering
- DCE
- **Graph Coloring Reg Alloc**
- Spill CSE
- Graph Coloring Stack Alloc
- Report Used Registers
- Fix Partial Register Stalls
- Lower Multiple Entrypoints
- Select Block Order
- 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)
```

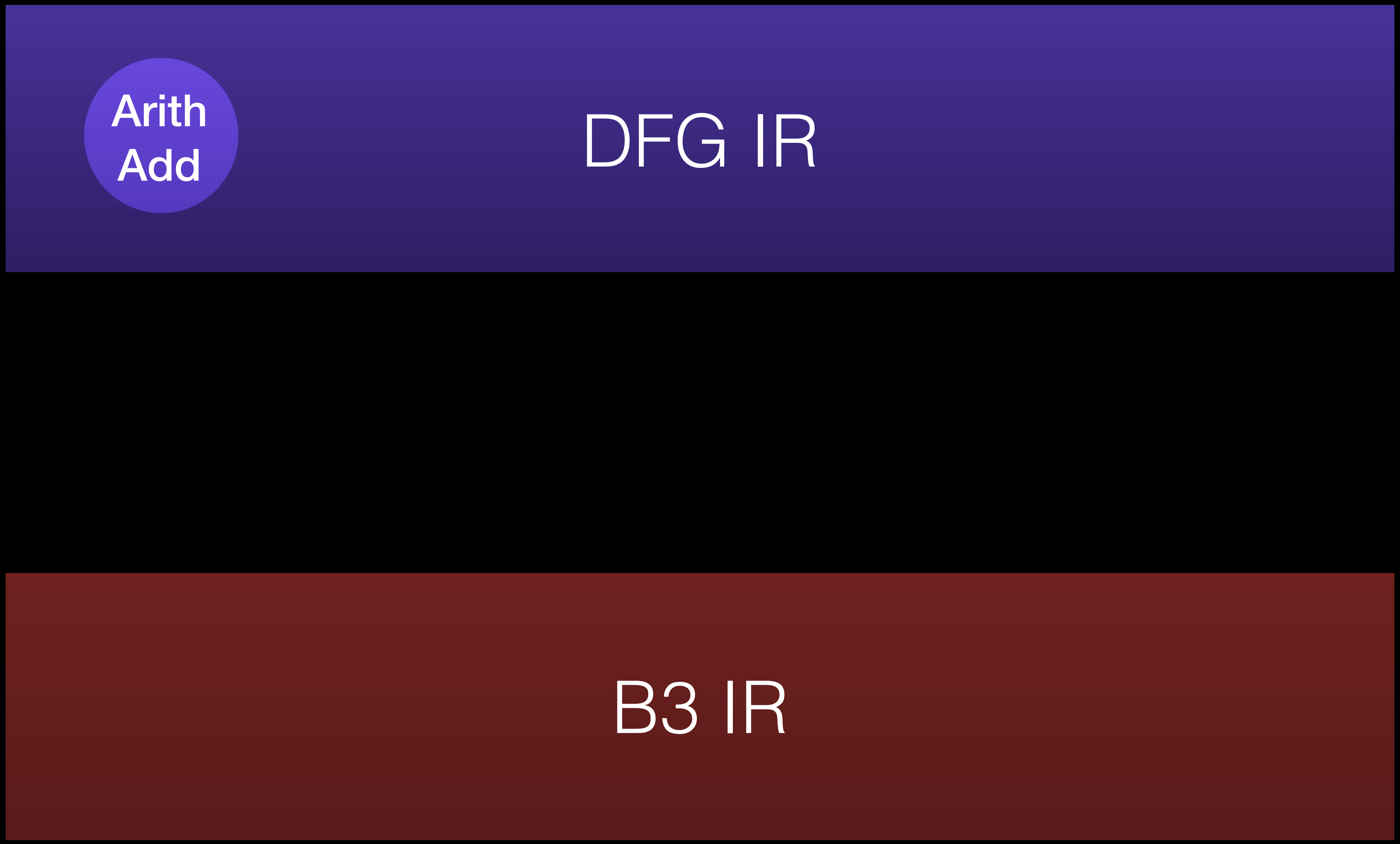
```
Int32 @42 = Trunc(@32, DFG:@26)
Int32 @43 = Trunc(@27, DFG:@26)
Int32 @44 = CheckAdd(@42:WarmAny, @43:WarmAny, generator = 0x1052c5cd0,
                     earlyClobbered = [], lateClobbered = [], usedRegisters = [],
                     ExitsSideways|Reads:Top, DFG:@26)
Int32 @45 = Trunc(@22, DFG:@30)
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

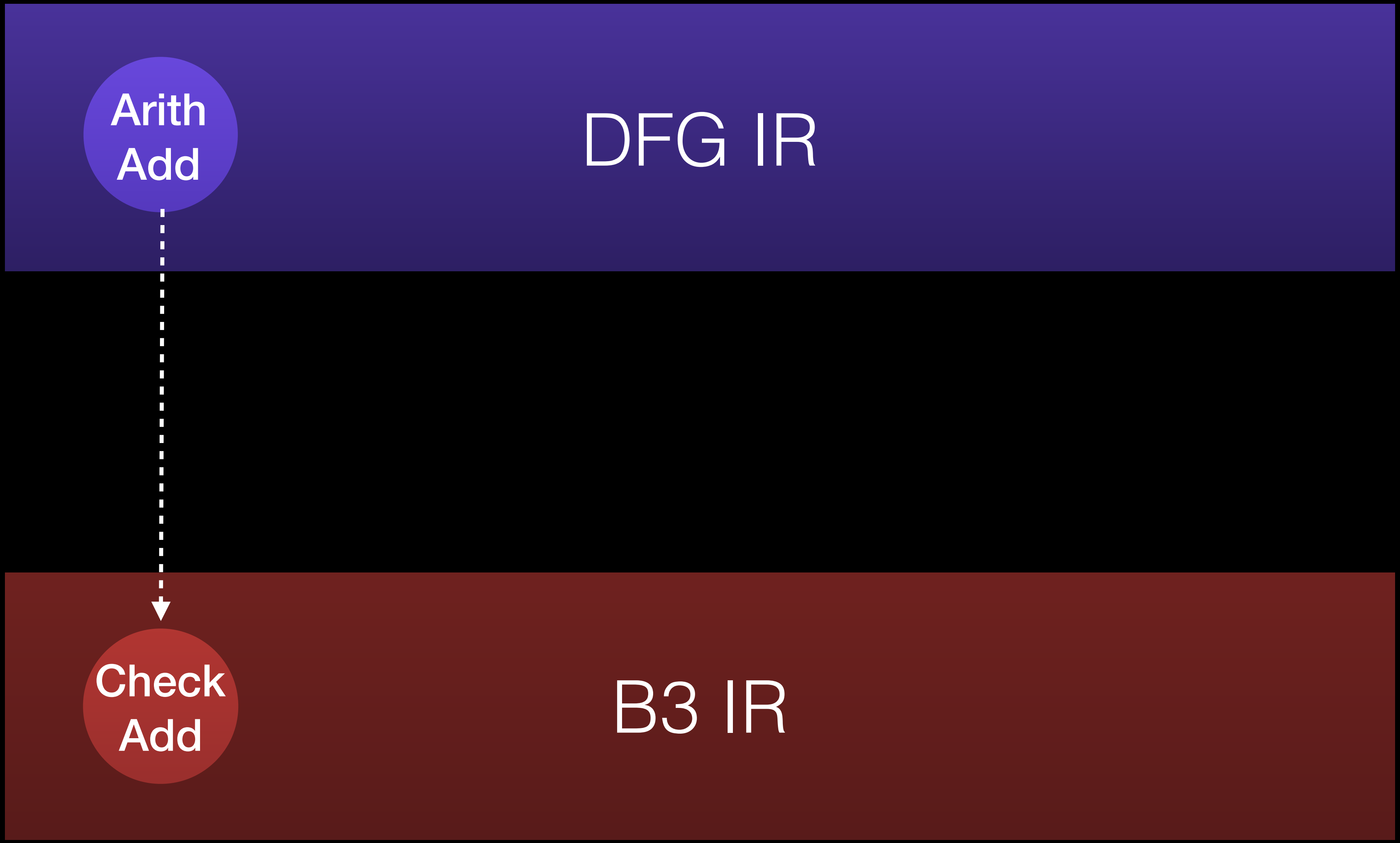
B3 IR



Arith Add

DFG IR

B3 IR

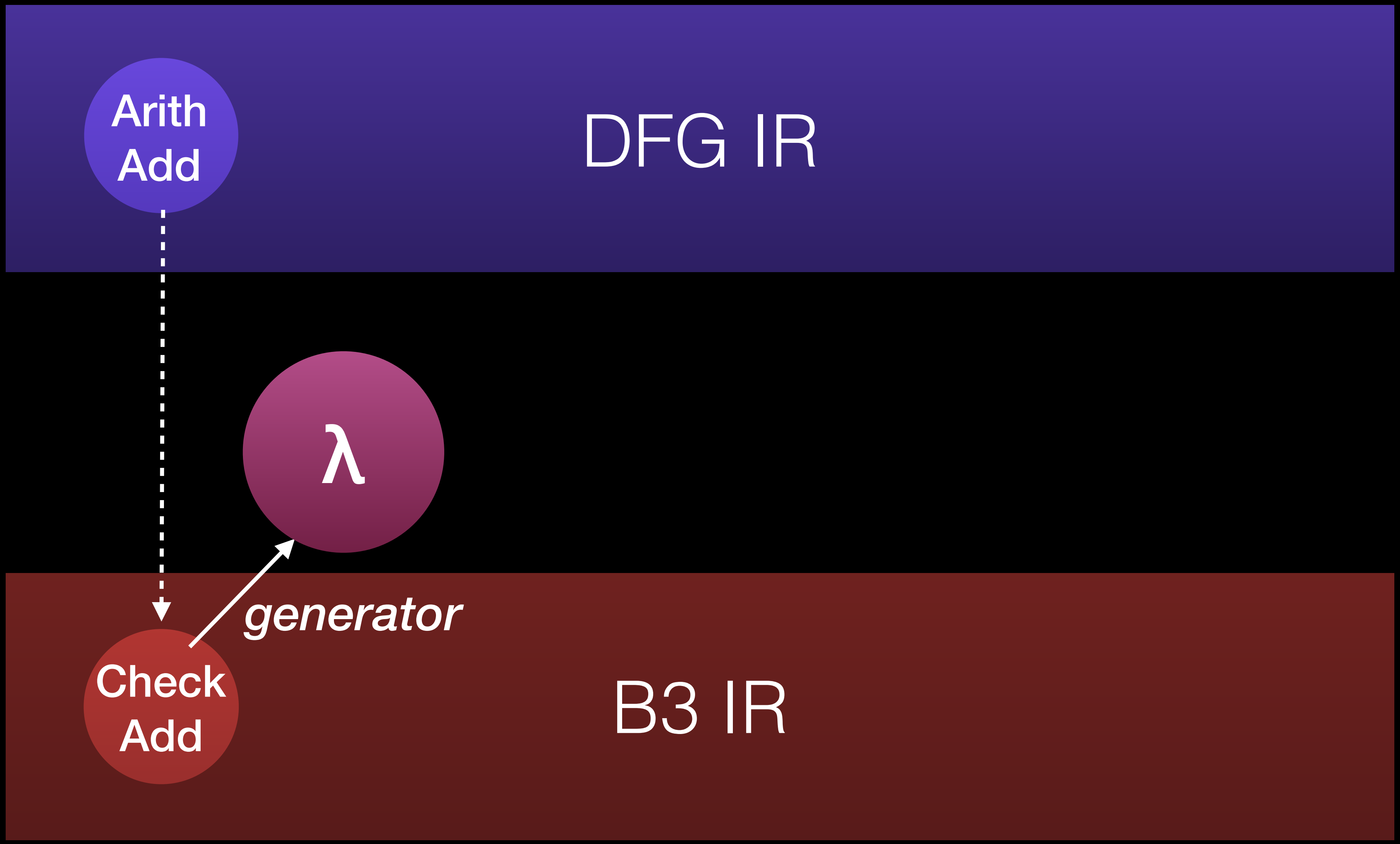


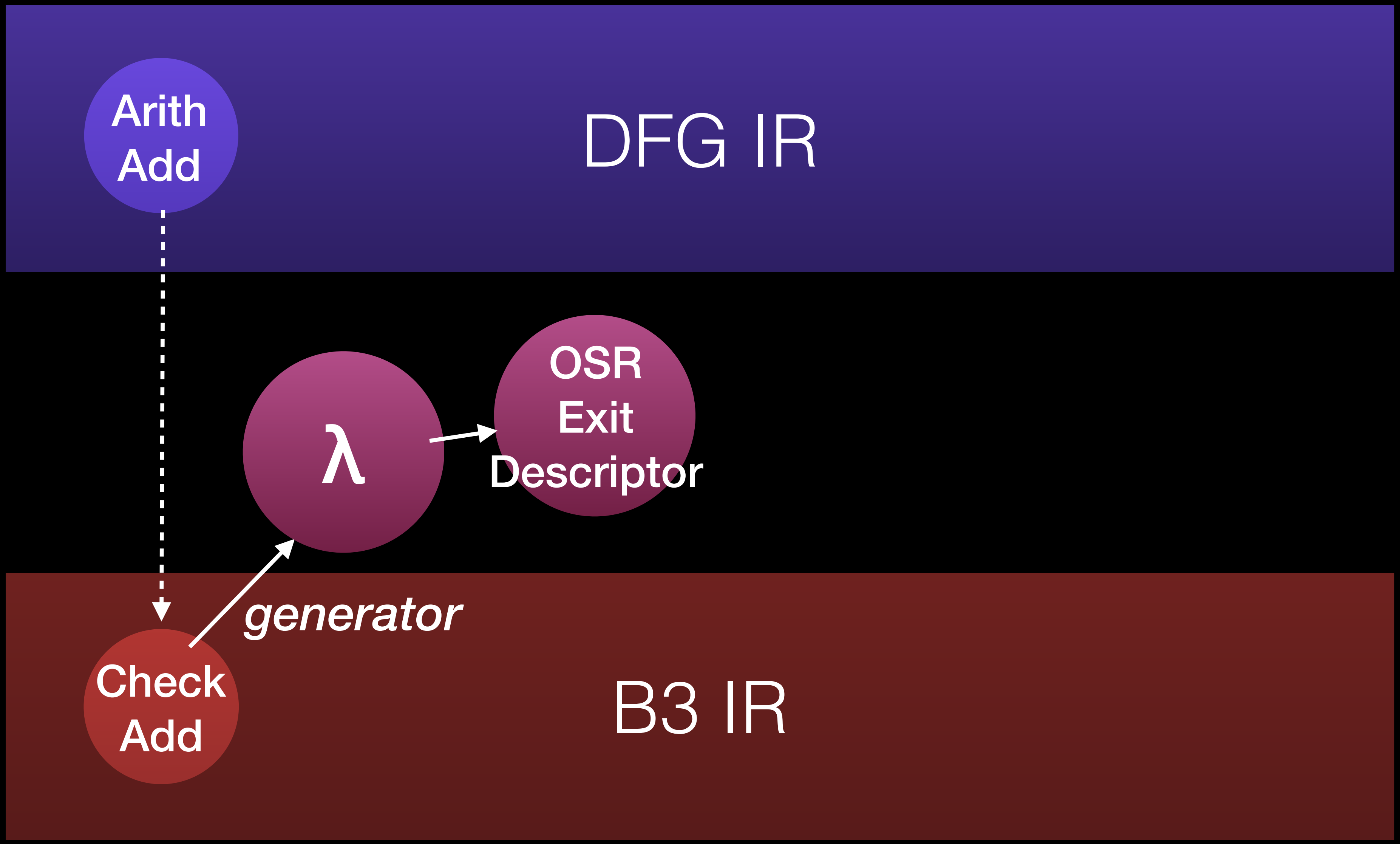
Arith Add

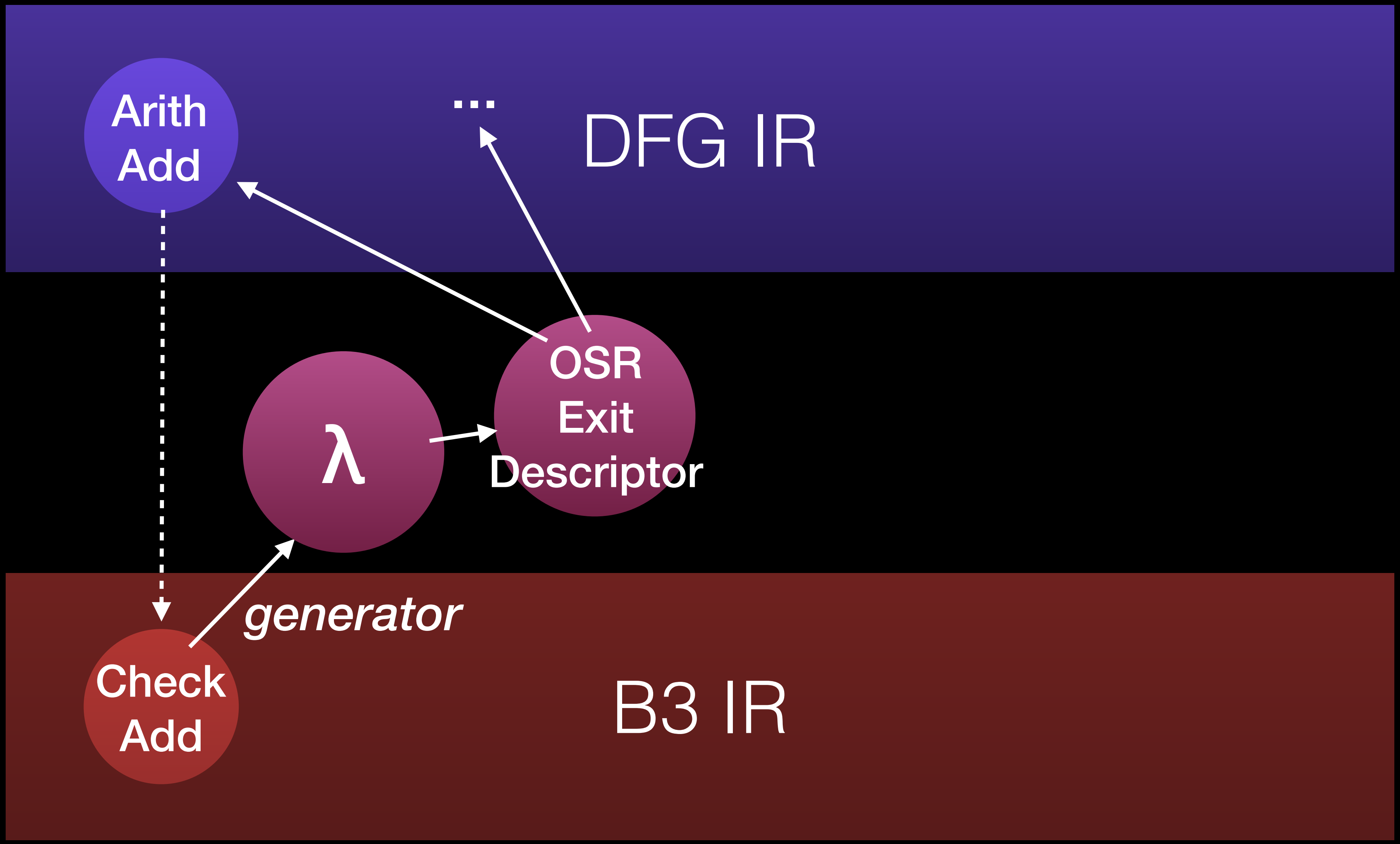
DFG IR

Check Add

B3 IR


Arith Add
DFG IR
λ
generator
Check Add
B3 IR


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


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

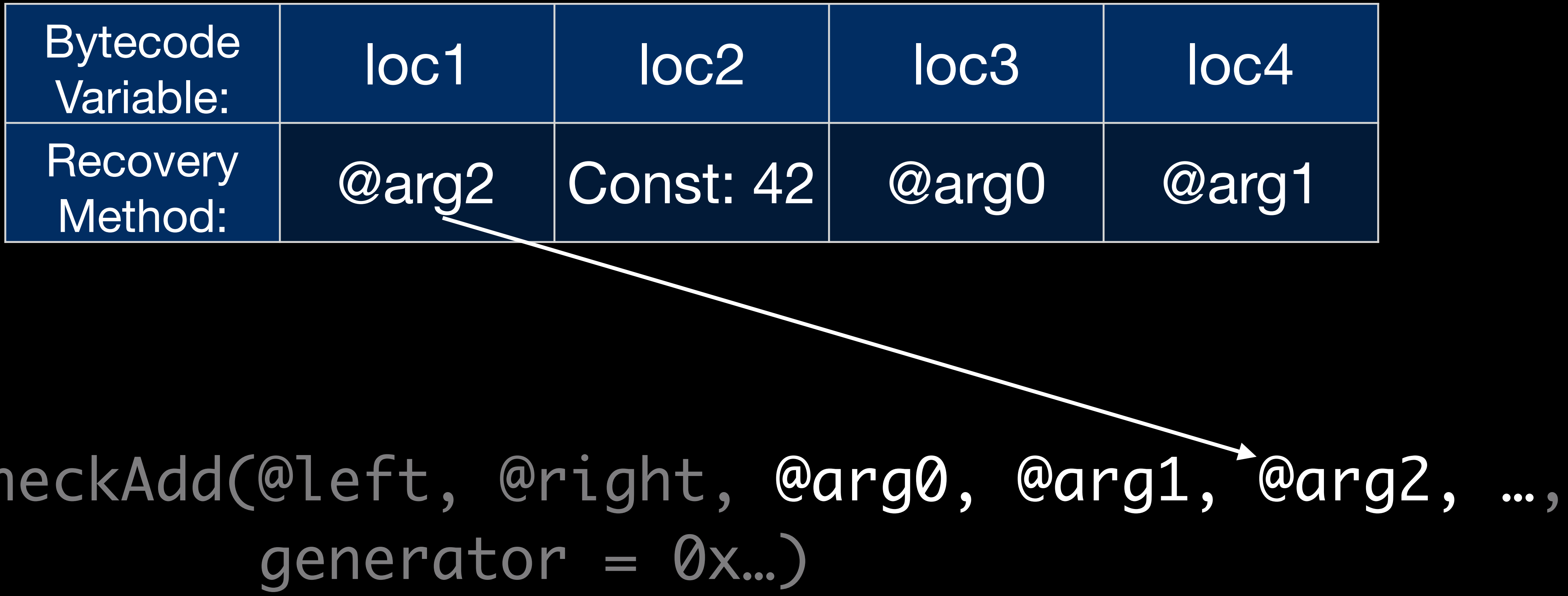
```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
         generator = 0x…)
```

JSC::FTL::OSRExitDescriptor

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

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

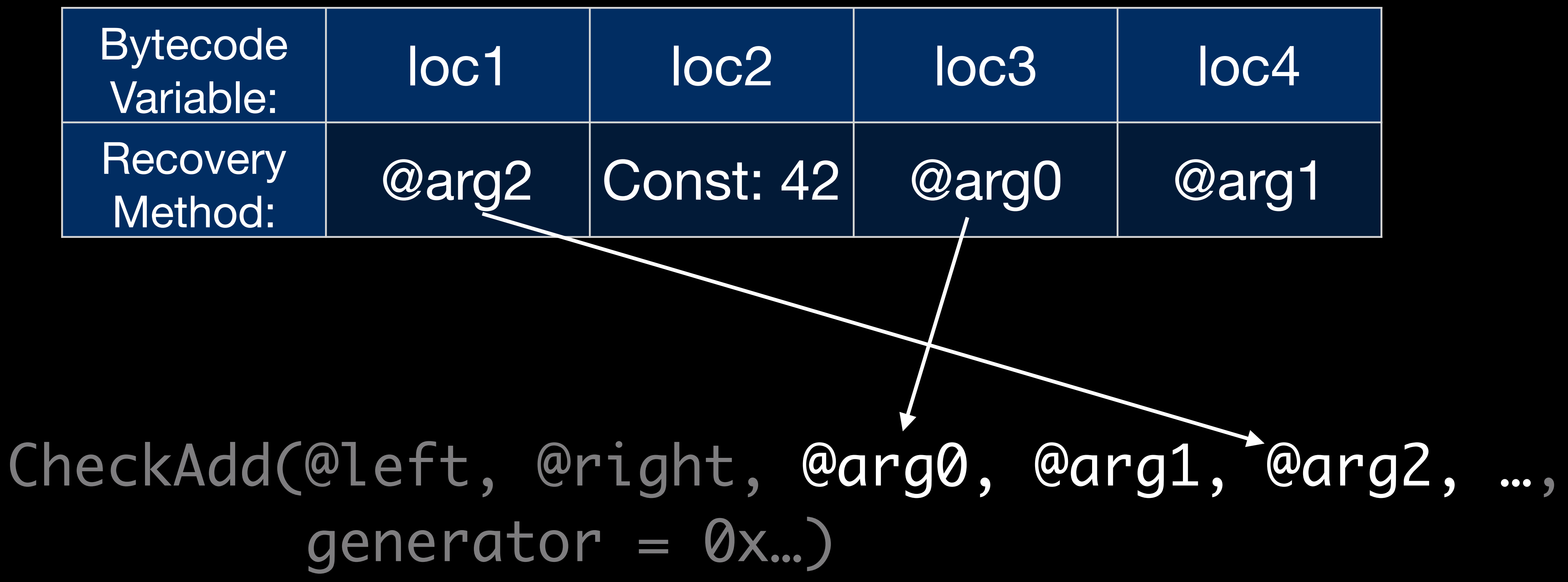
JSC::FTL::OSRExitDescriptor



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

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

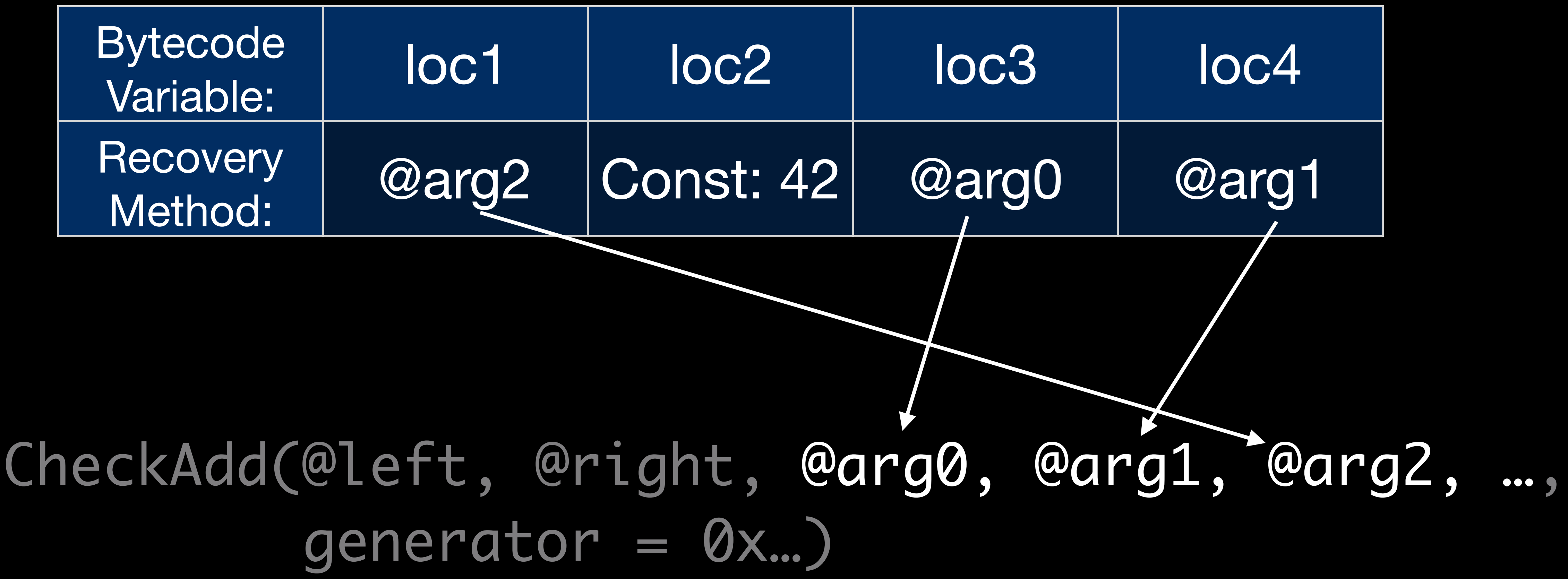
JSC::FTL::OSRExitDescriptor



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

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

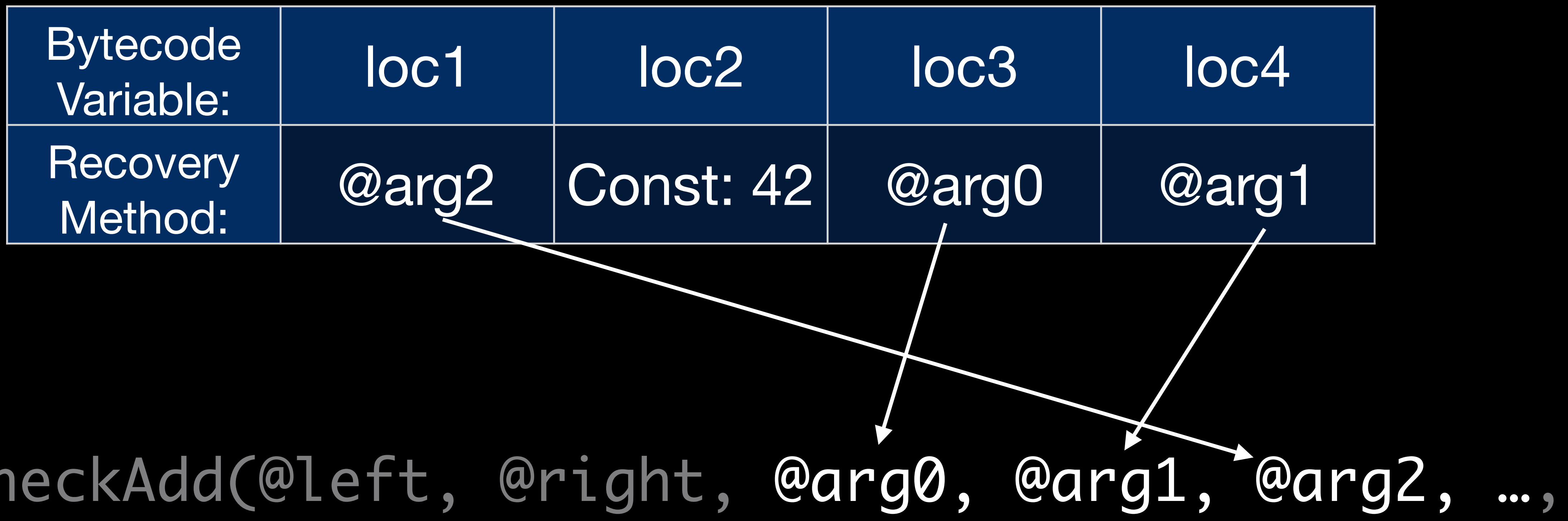
JSC::FTL::OSRExitDescriptor



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

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

JSC::FTL::OSRExitDescriptor



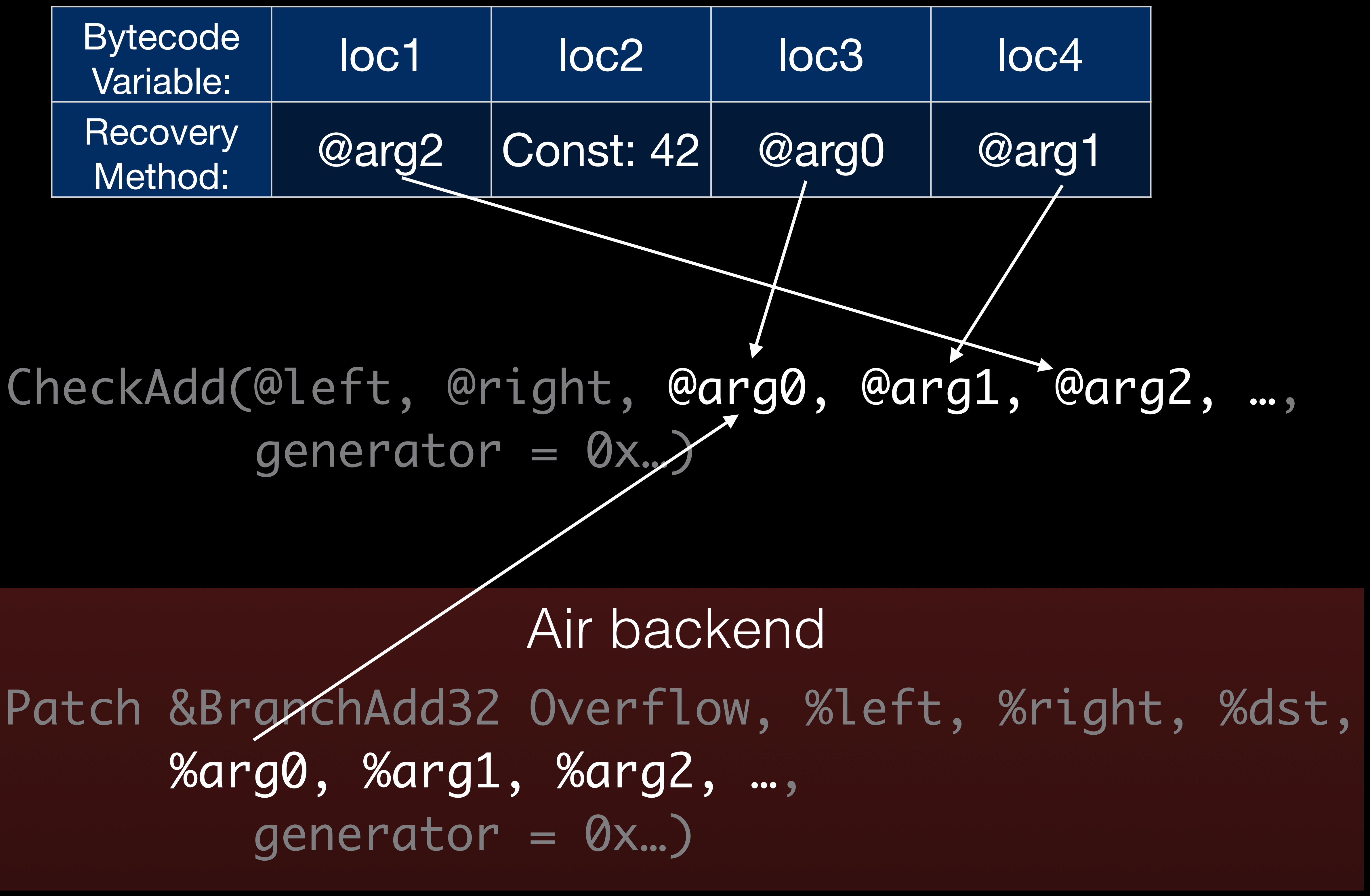
Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

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

Air backend

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

JSC::FTL::OSRExitDescriptor



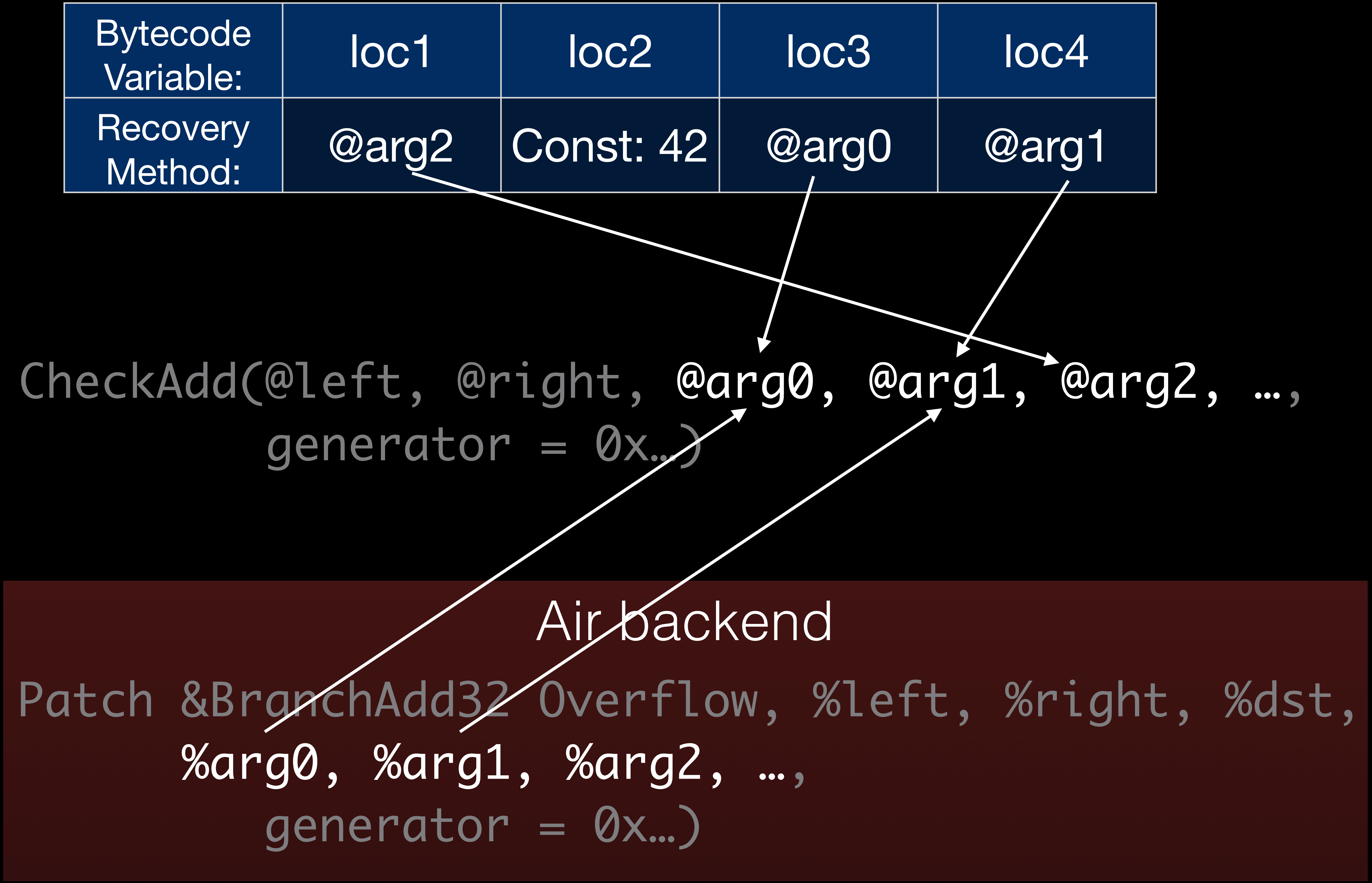
Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
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

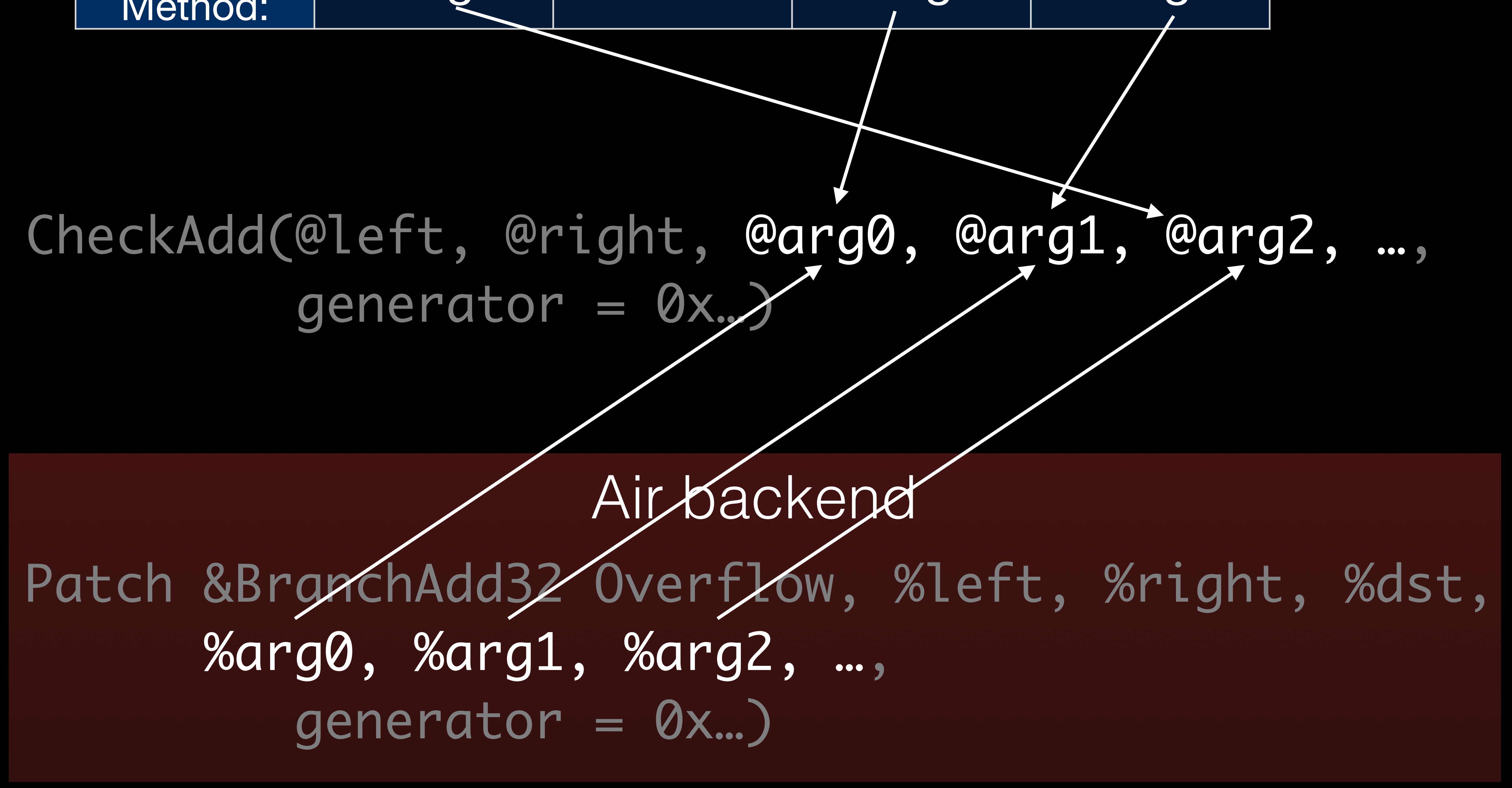
```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
        generator = 0x…)
```

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

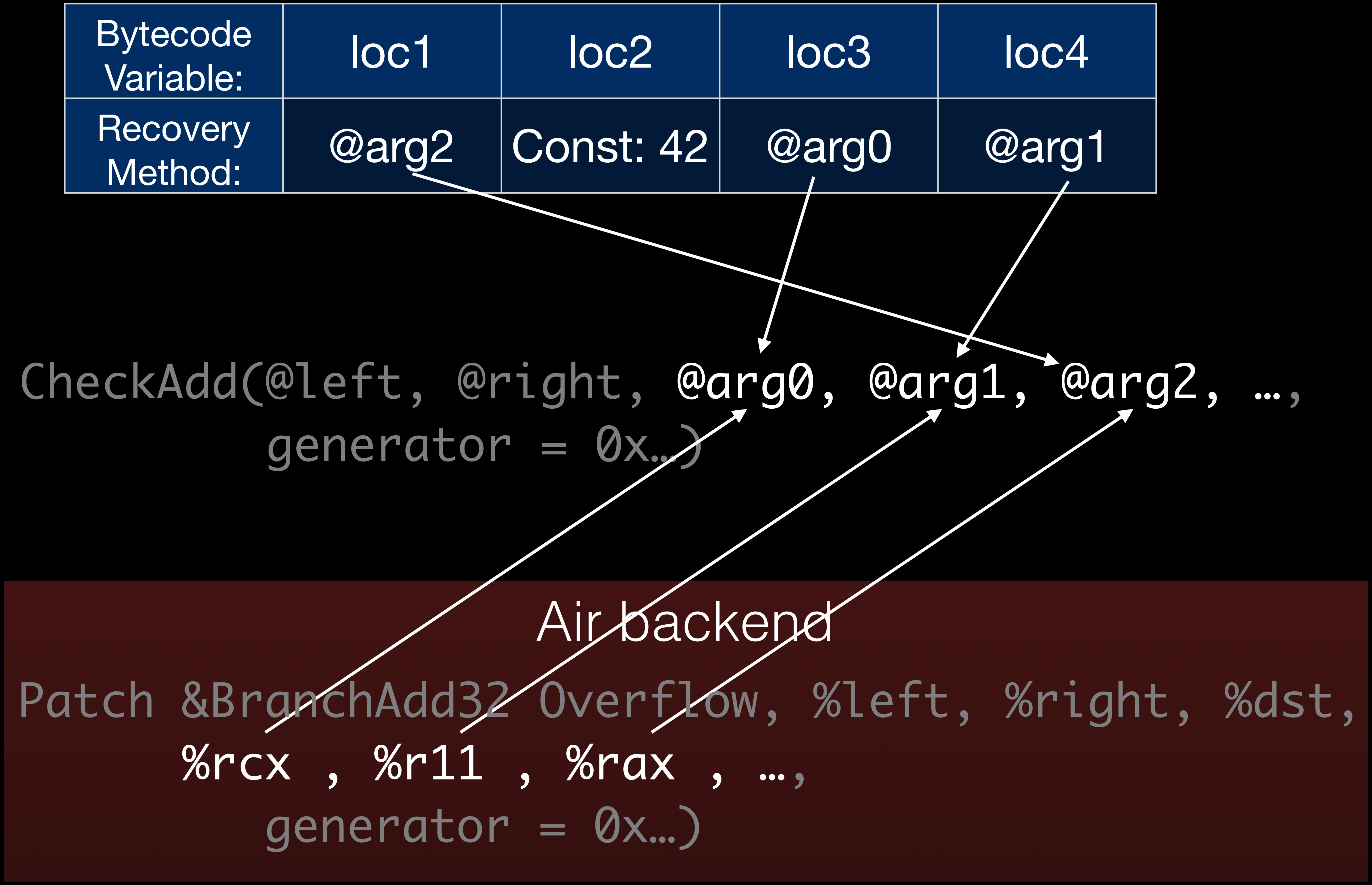


```
CheckAdd(@left, @right, @arg0, @arg1, @arg2, …,
        generator = 0x…)
```

Air backend

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

JSC::FTL::OSRExitDescriptor



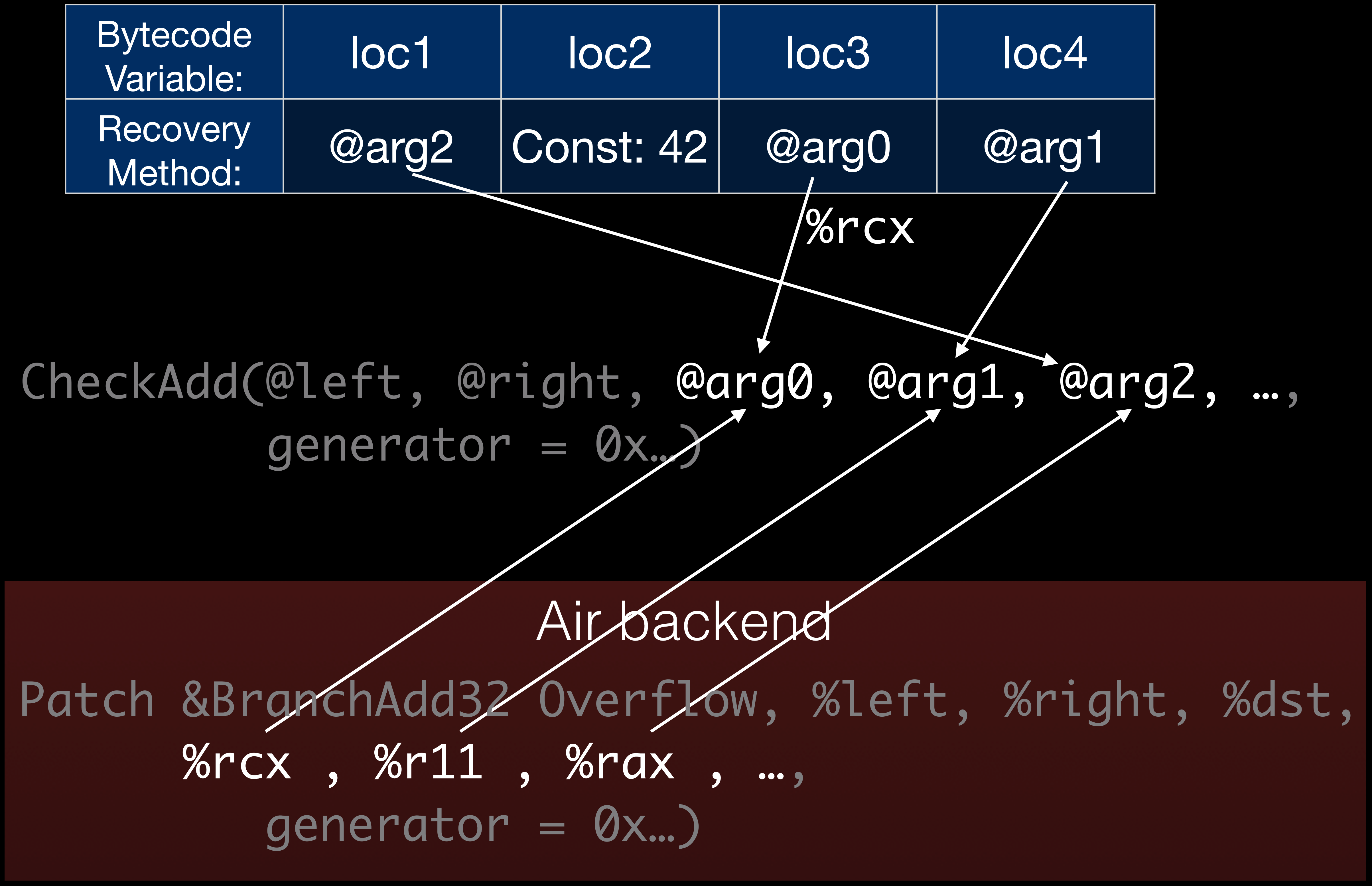
Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

```
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

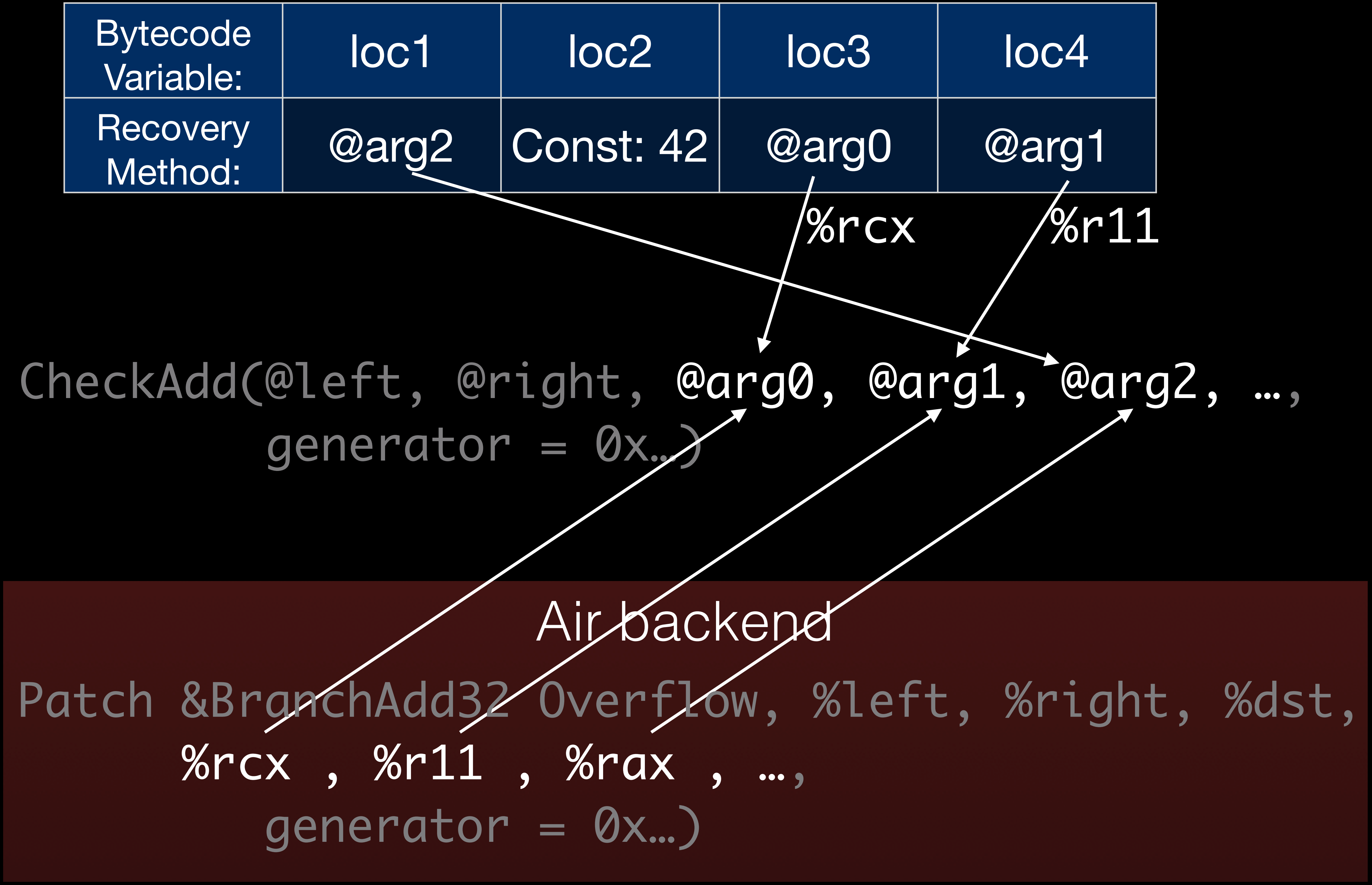
%rcx

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

Air backend

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

JSC::FTL::OSRExitDescriptor



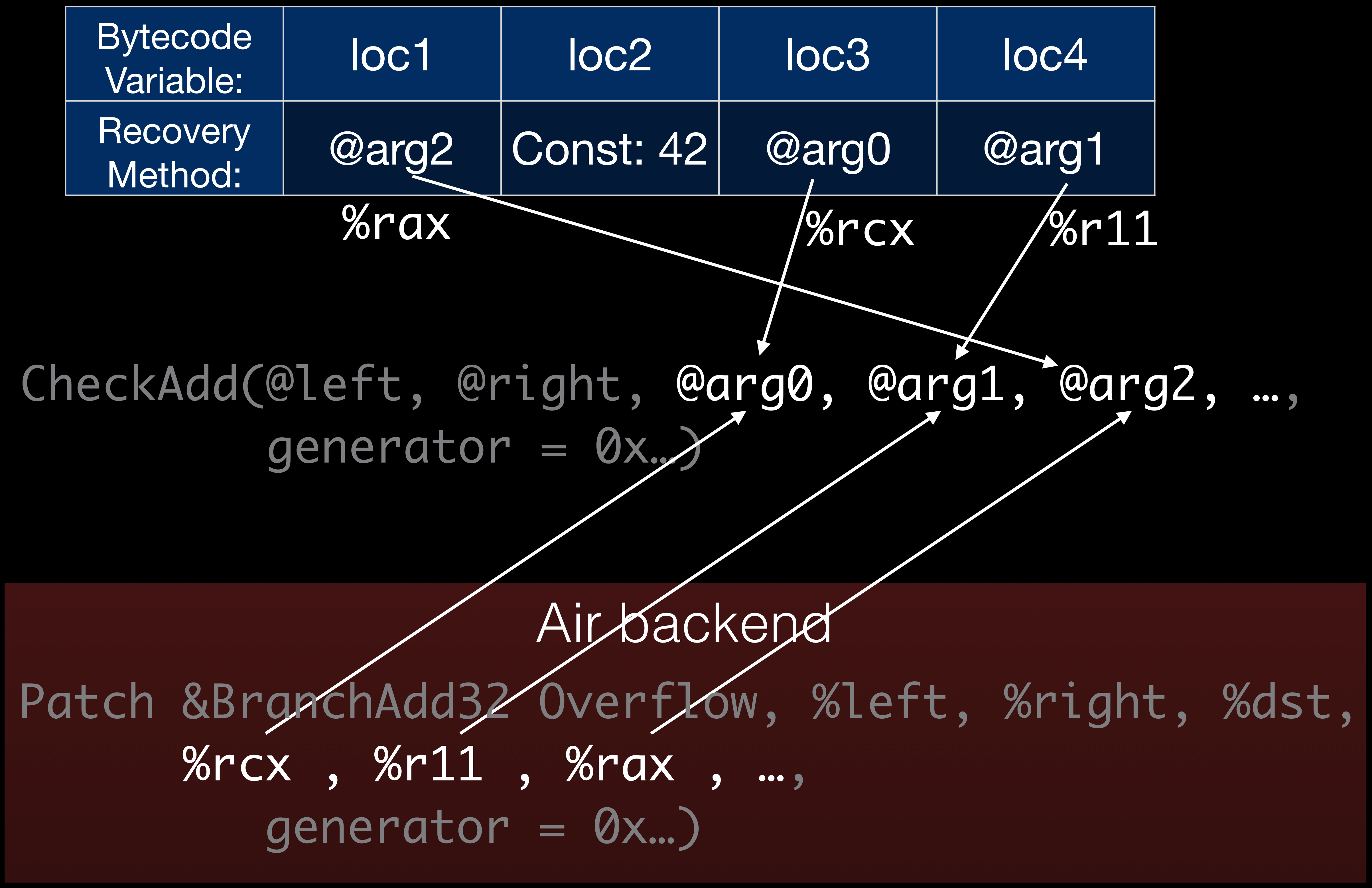
Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

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

Air backend

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

JSC::FTL::OSRExitDescriptor



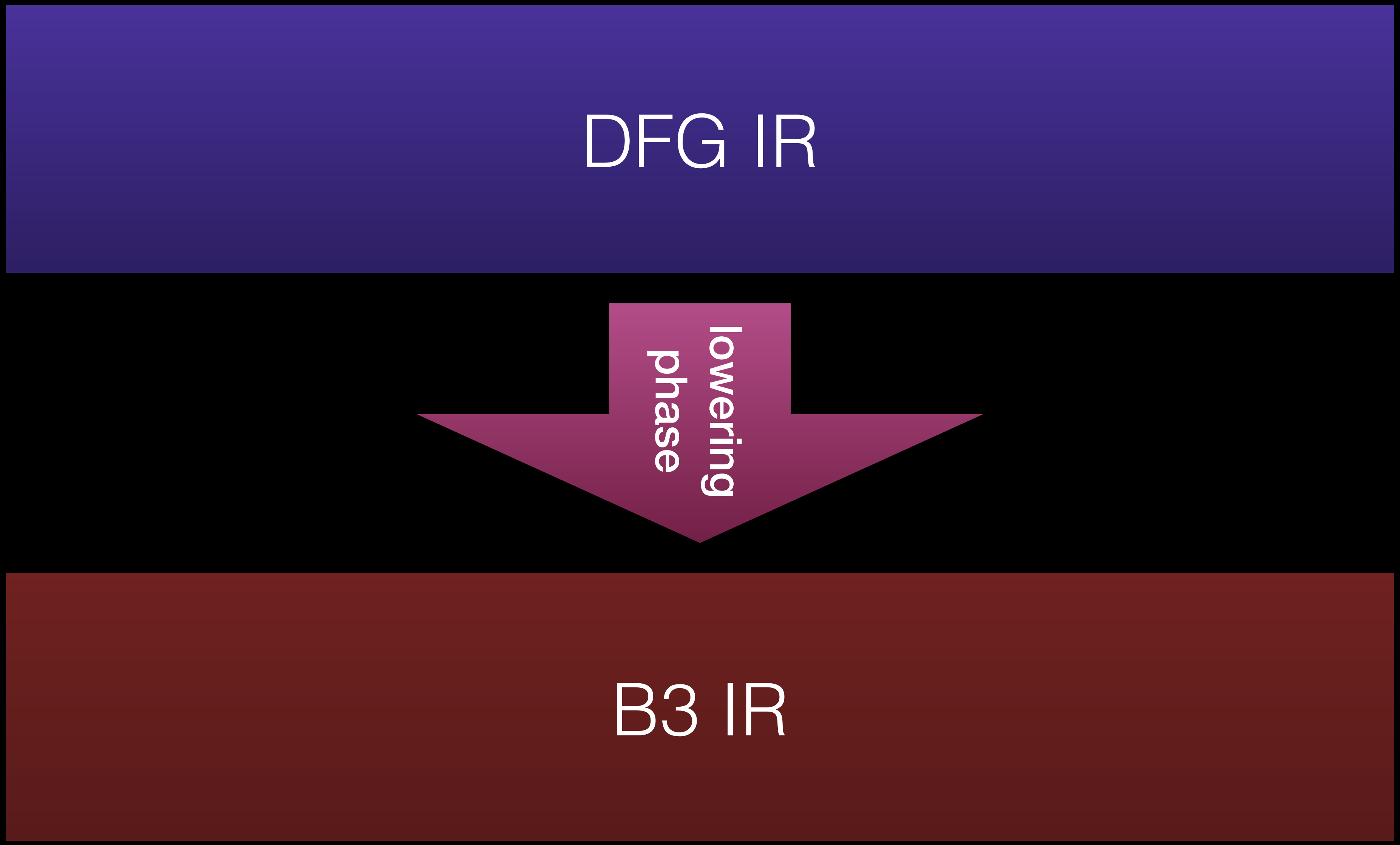
Bytecode Variable:	loc1	loc2	loc3	loc4
Recovery Method:	@arg2	Const: 42	@arg0	@arg1

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

Air backend

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

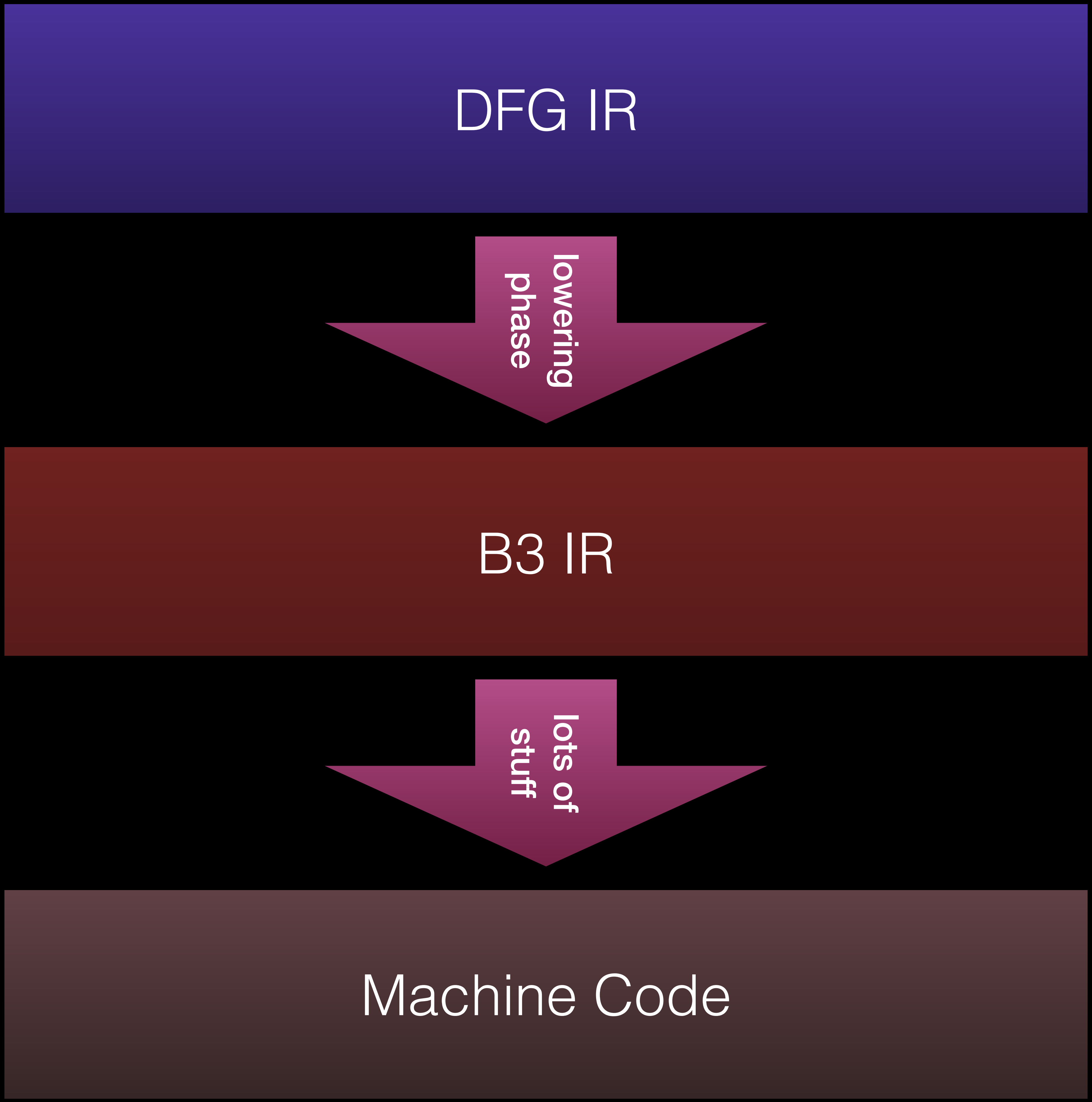
DFG IR



DFG IR

lowering phase

B3 IR



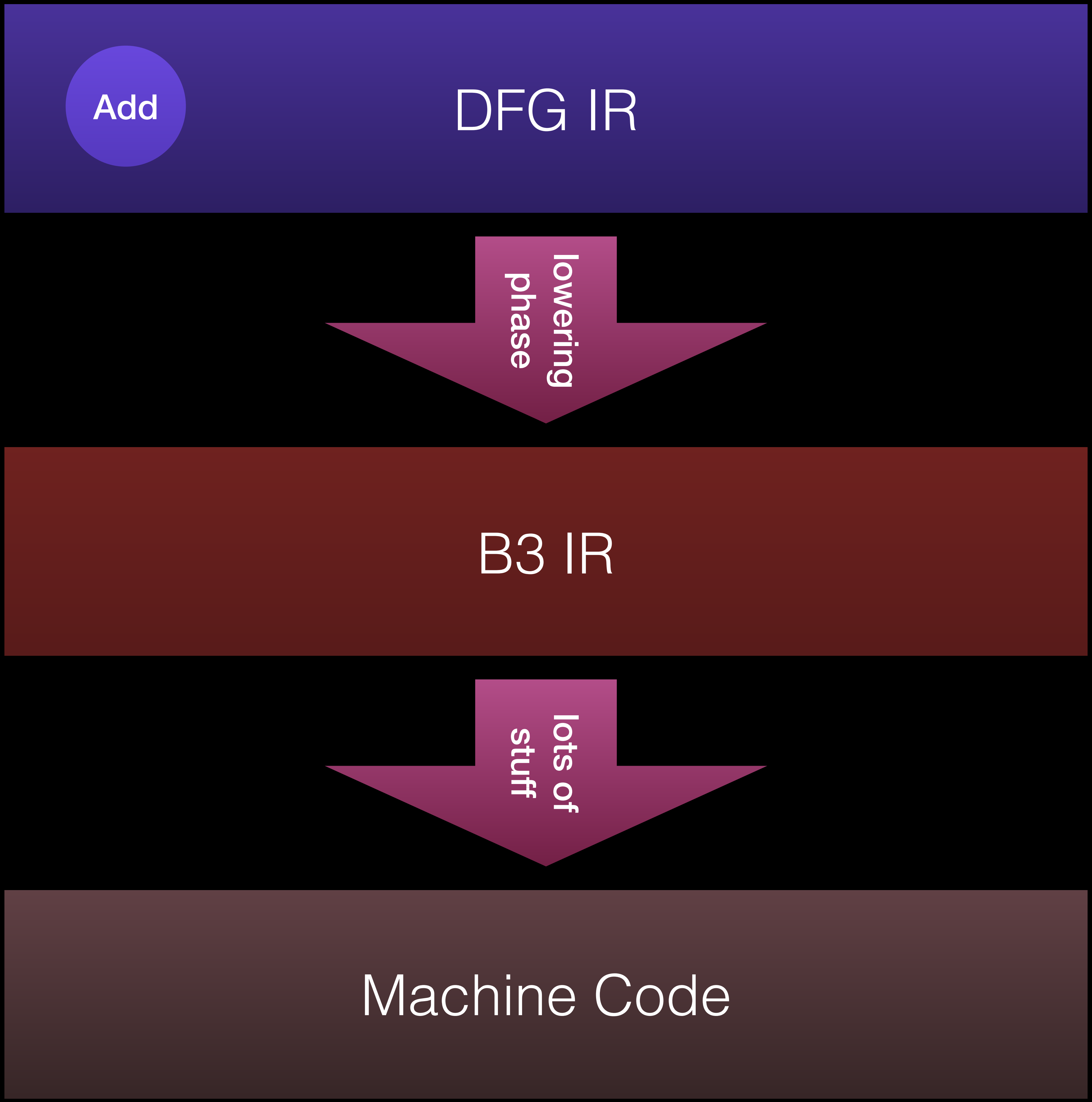
DFG IR

lowering phase

B3 IR

lots of stuff

Machine Code



Add

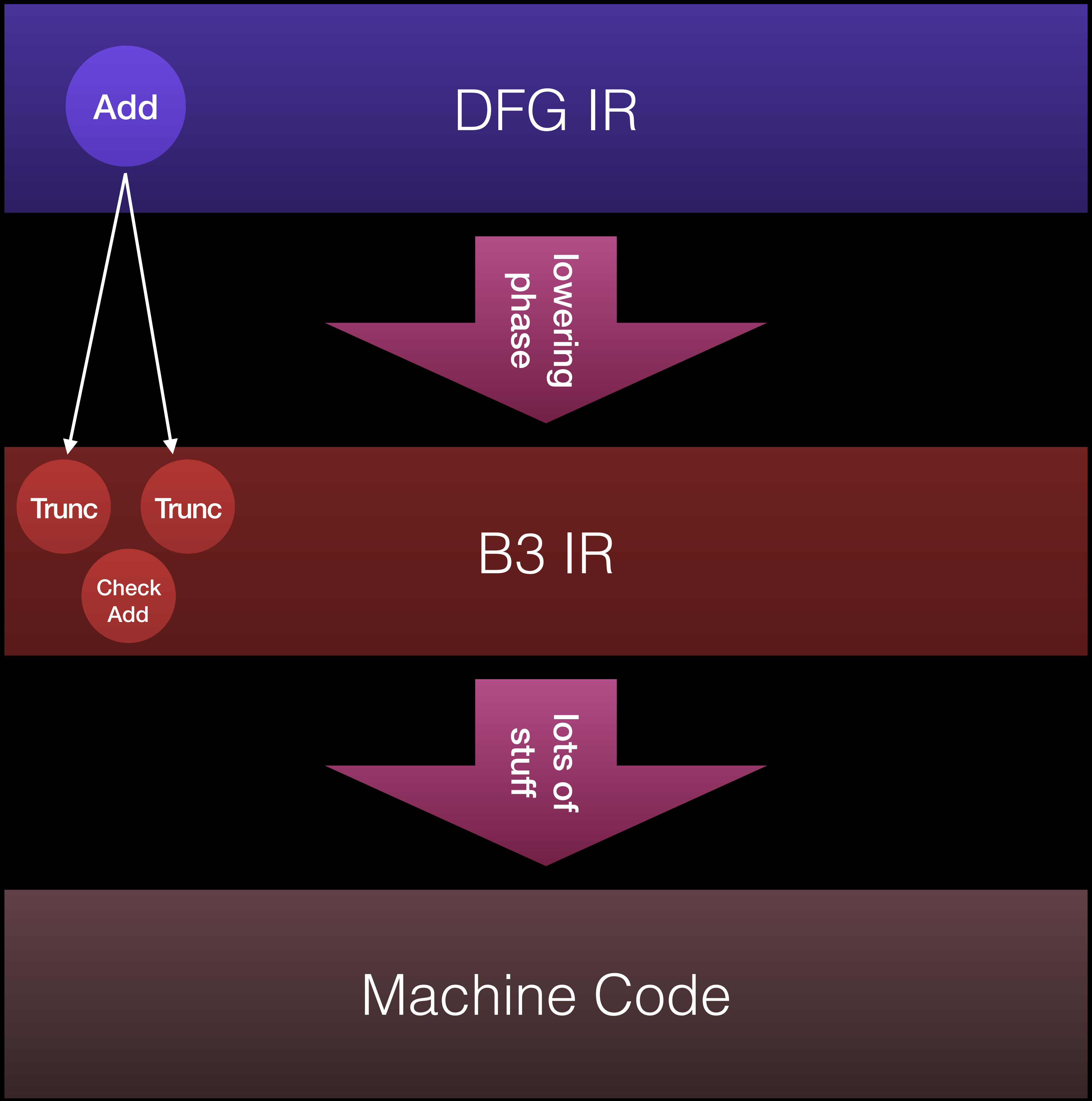
DFG IR

lowering phase

B3 IR

lots of stuff

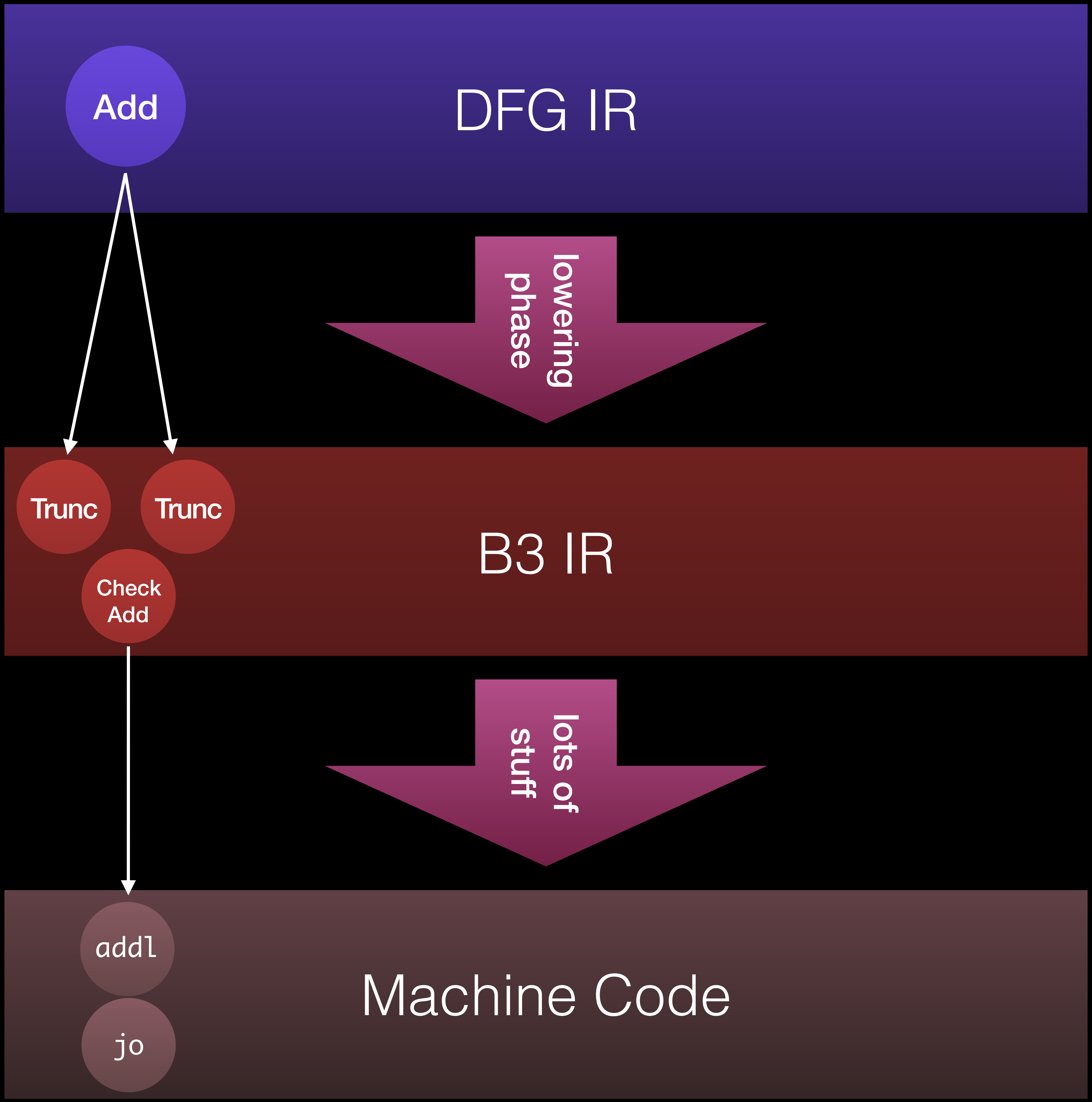
Machine Code



DFG IR

B3 IR

Machine Code



Add

DFG IR

lowering phase

Trunc

Trunc

Check Add

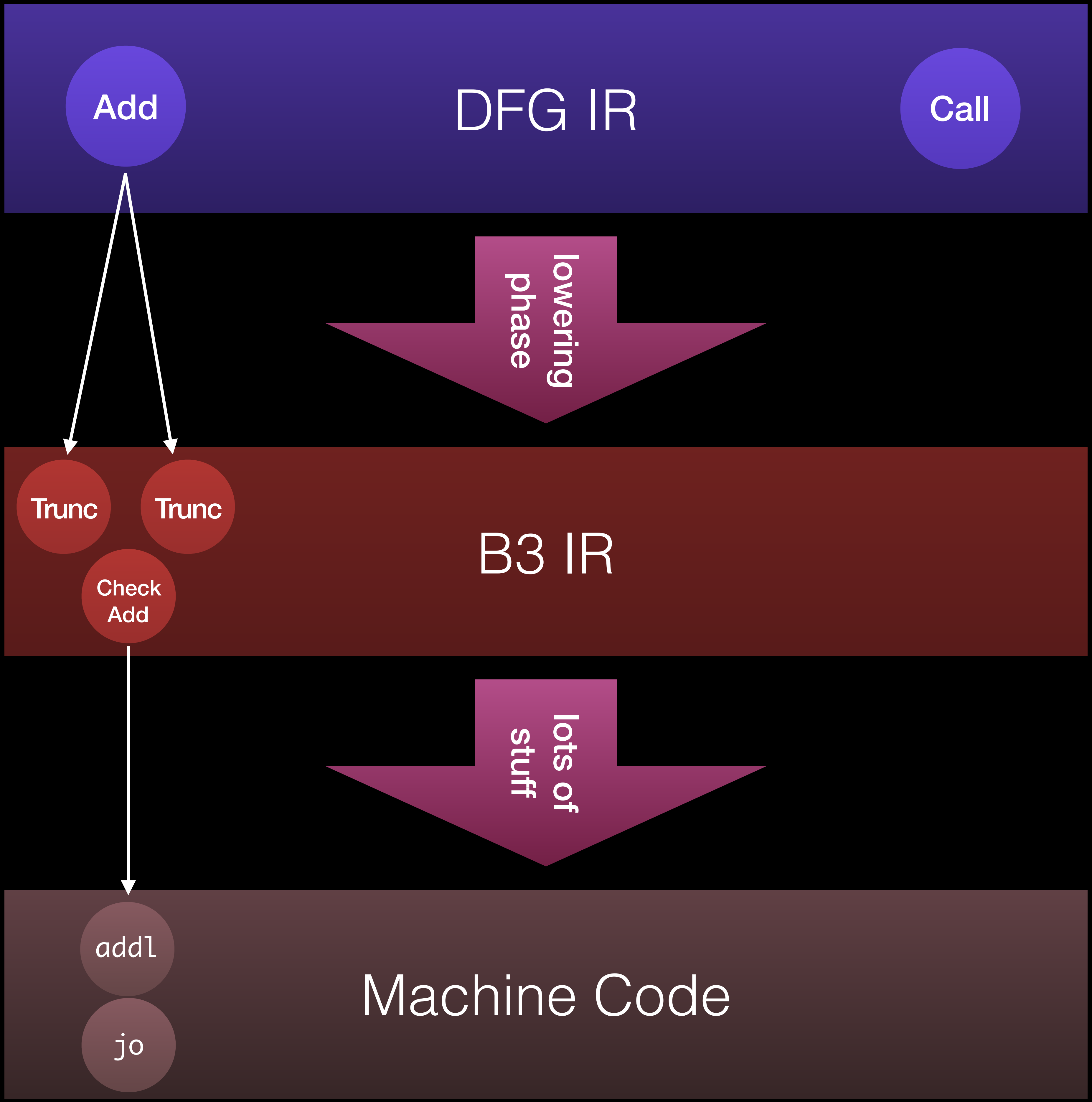
B3 IR

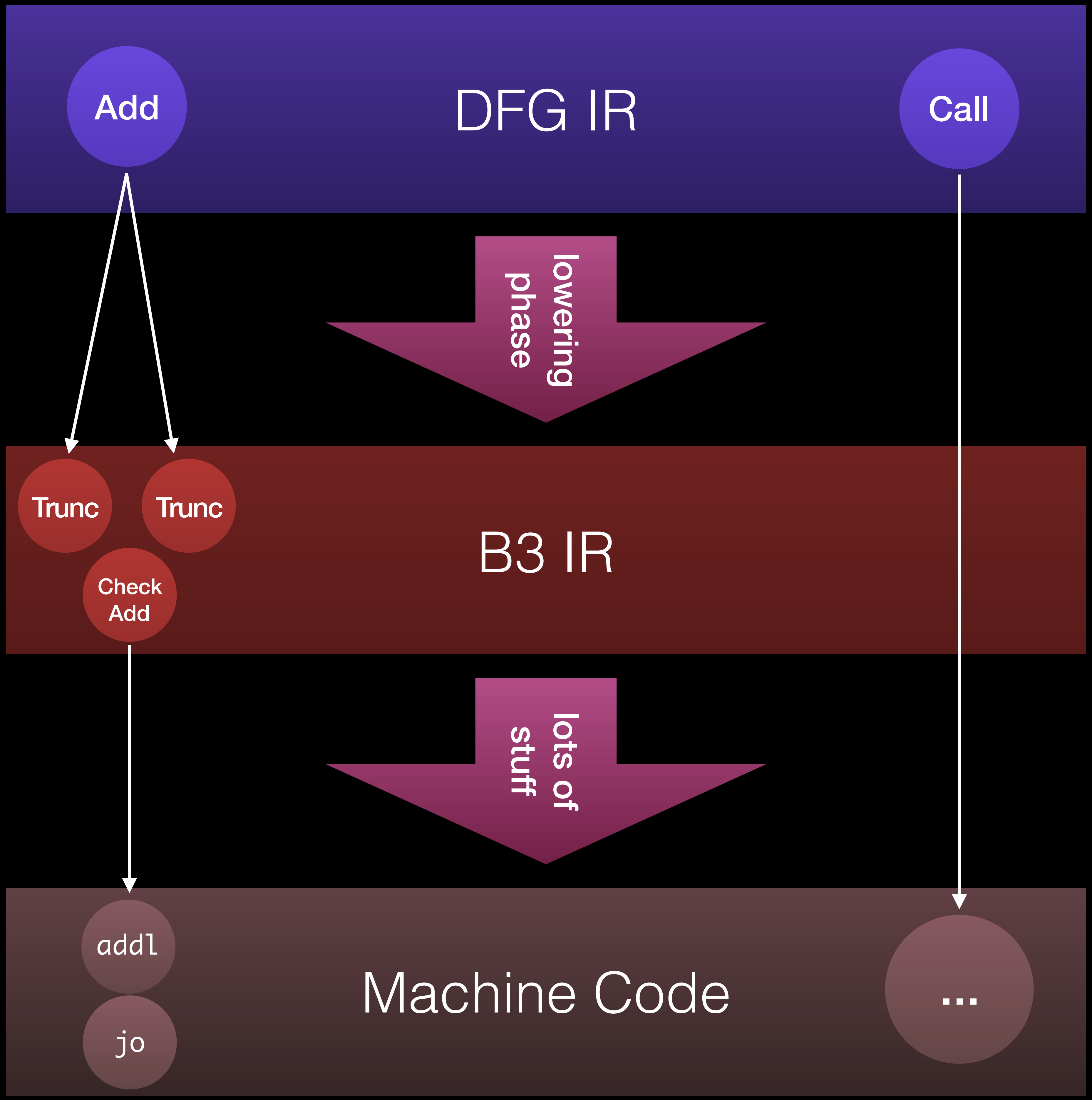
lots of stuff

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

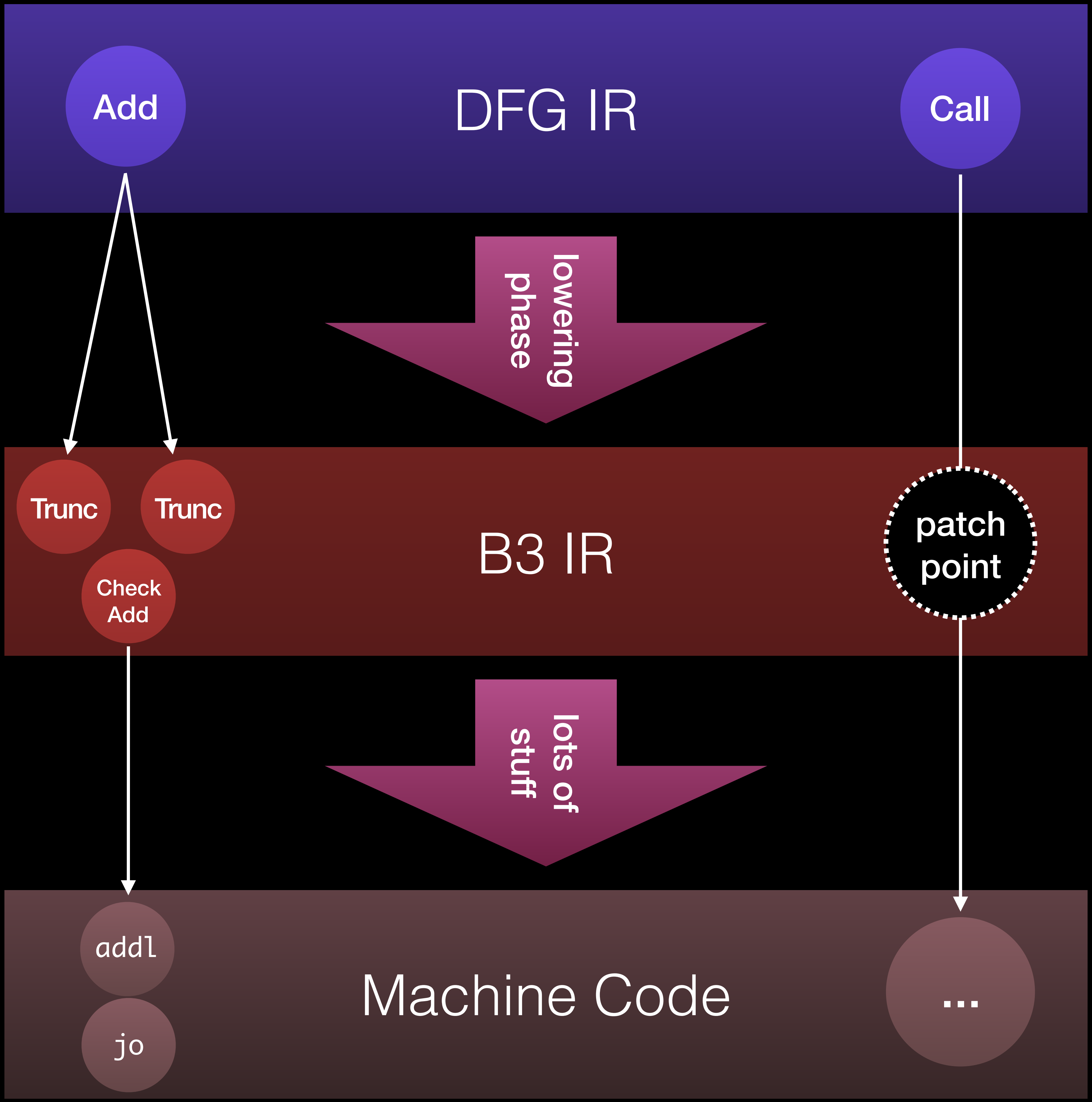
lots of stuff

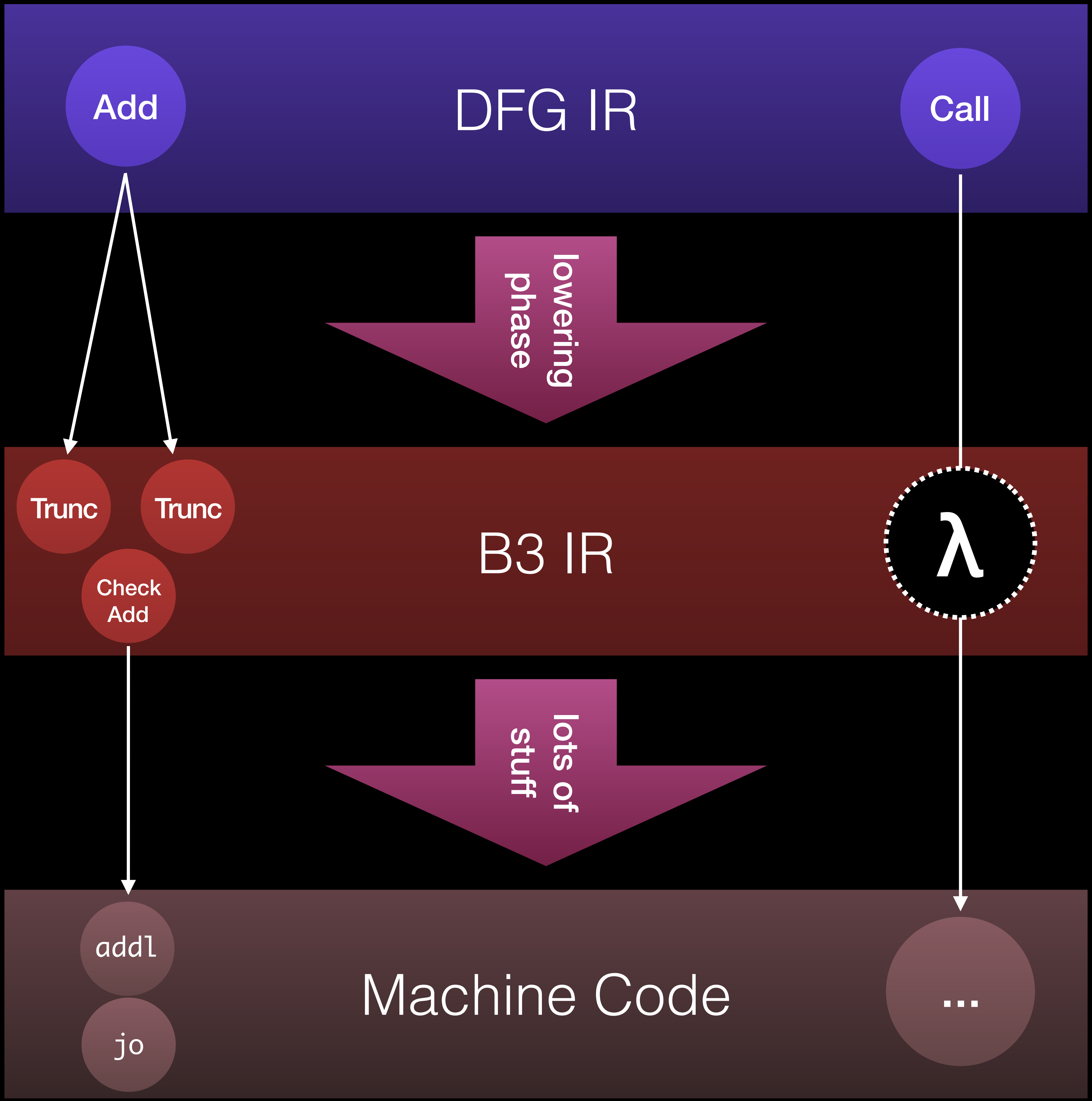
Machine Code

addl

jo

...


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



Add

DFG IR

Call

lowering phase

Trunc

Trunc

Check Add

B3 IR

λ

lots of stuff

addl

jo

Machine Code

...

```
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;
}
```

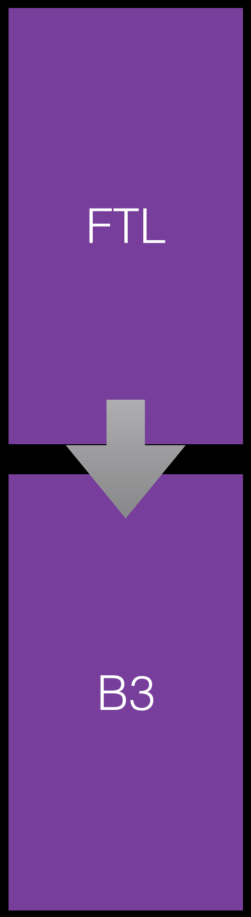
```
if (MacroAssemblerARM64::
    supportsDoubleToInt32ConversionUsingJavaScriptSemantics()) {
    PatchpointValue* patchpoint = m_out.patchpoint(Int32);
    patchpoint->appendSomeRegister(doubleValue);
    patchpoint->setGenerator(
        [=] (CCallHelpers& jit,
             const StackmapGenerationParams& 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;
}
```

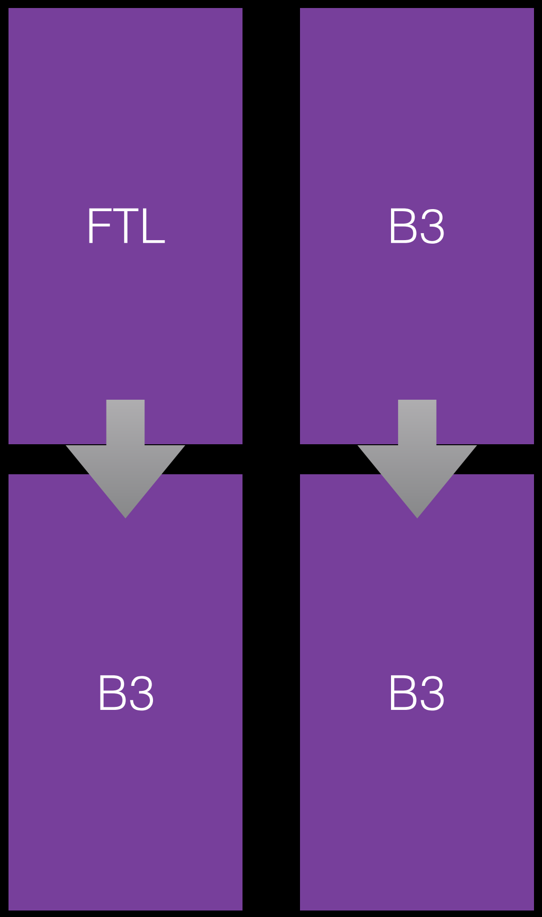
Patchpoint Use Cases

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

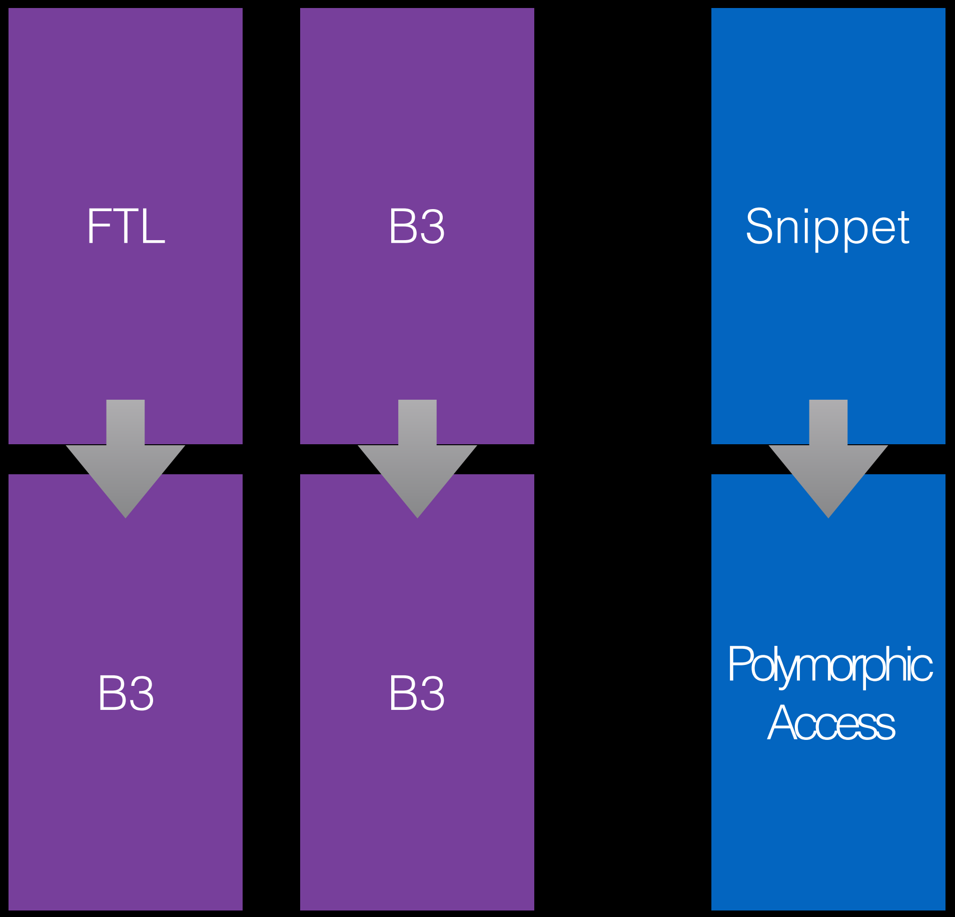
Patchpoint Use Cases



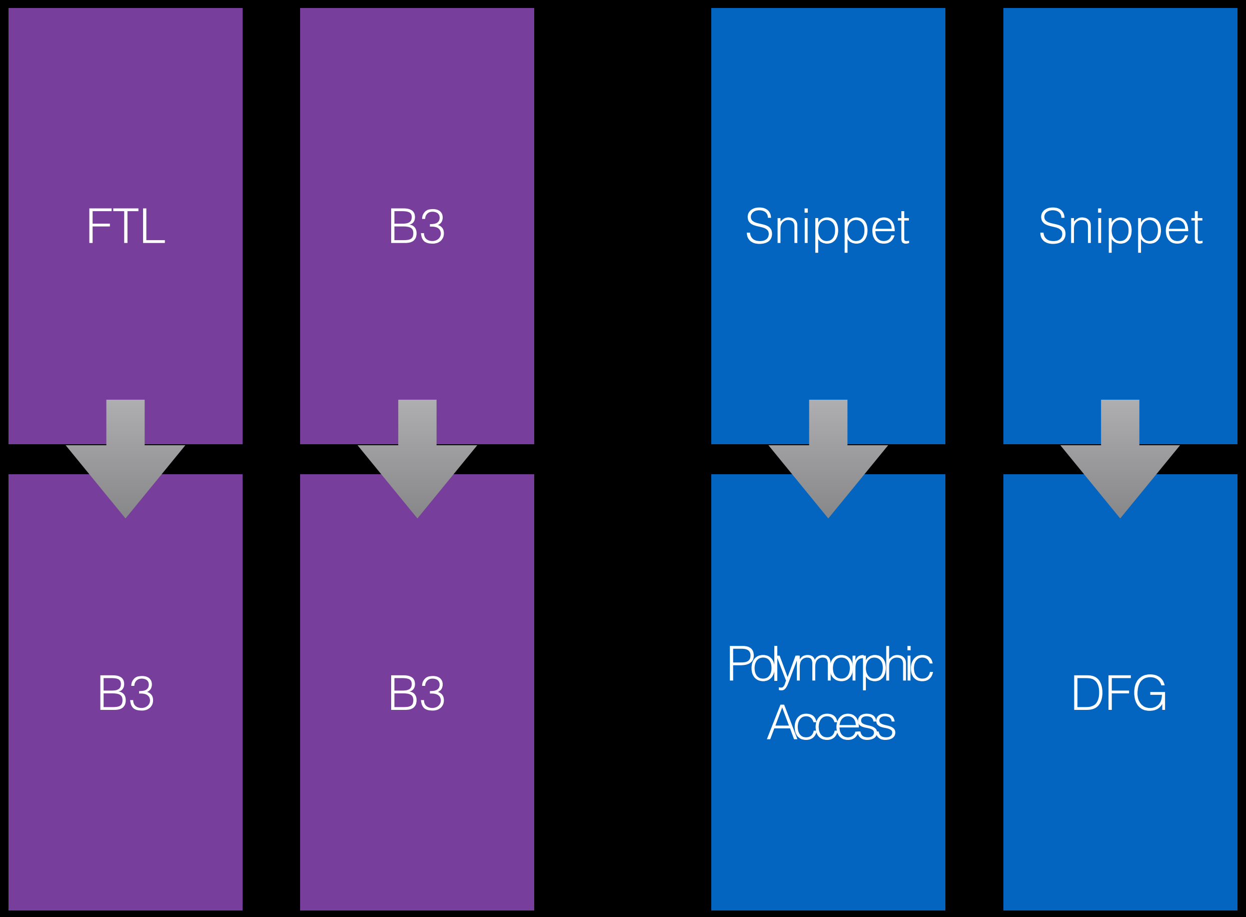
Patchpoint Use Cases



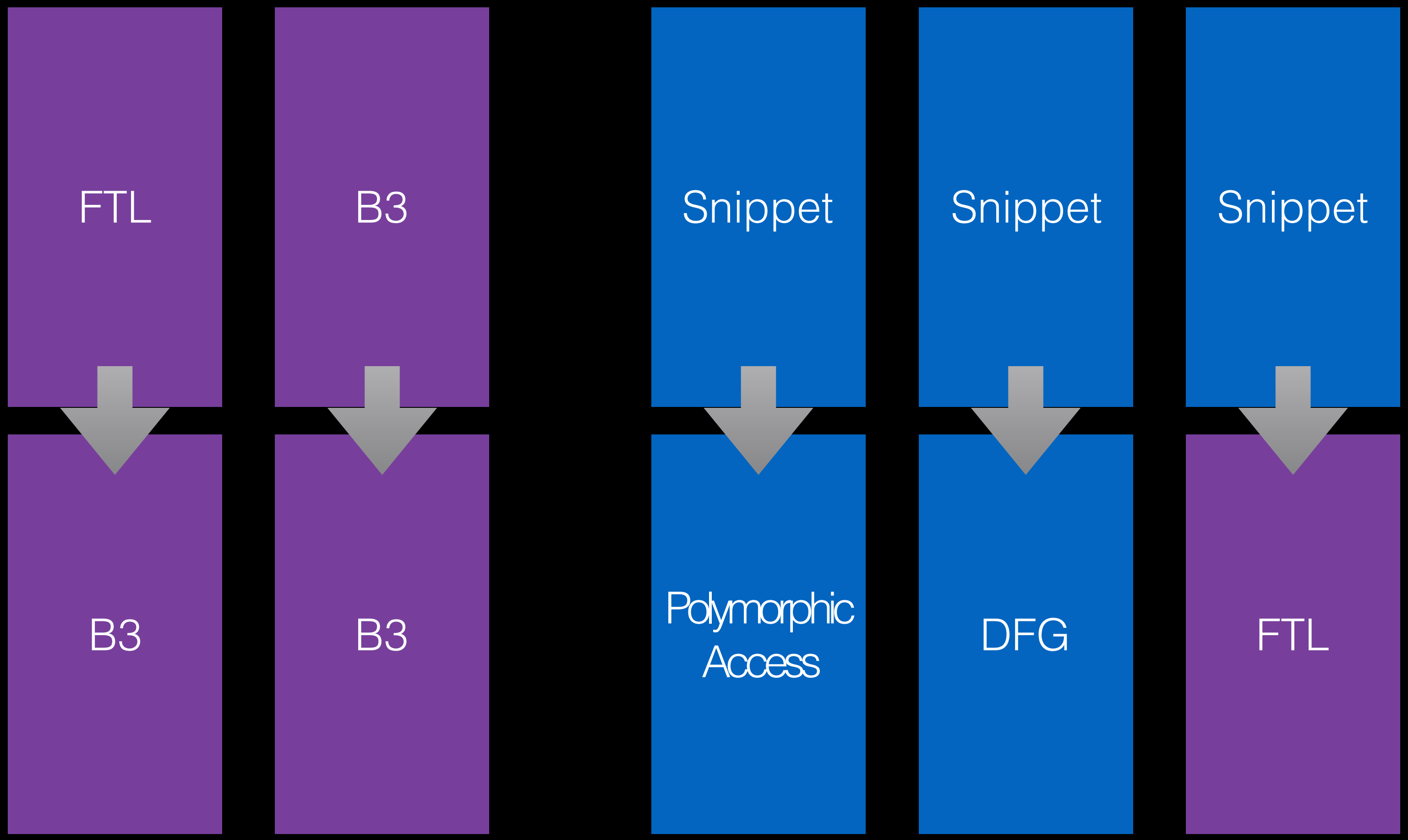
Patchpoint Use Cases

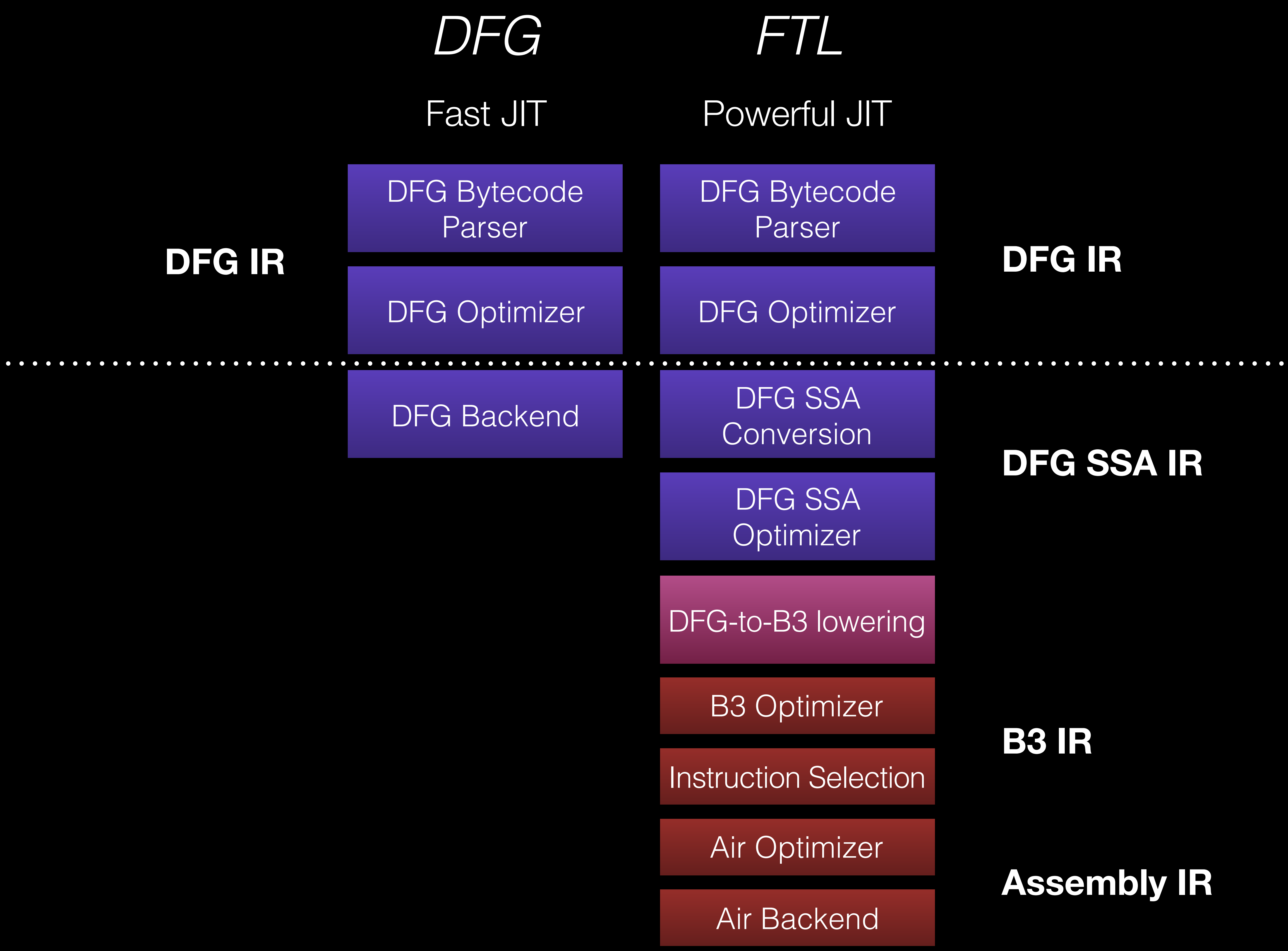


Patchpoint Use Cases



Patchpoint Use Cases





DFG

Fast JIT

FTL

Powerful JIT

DFG IR

DFG Bytecode Parser

DFG Optimizer

DFG Backend

DFG Bytecode Parser

DFG Optimizer

DFG IR

DFG SSA Conversion

DFG SSA Optimizer

DFG SSA IR

DFG-to-B3 lowering

B3 Optimizer

Instruction Selection

B3 IR

Air Optimizer

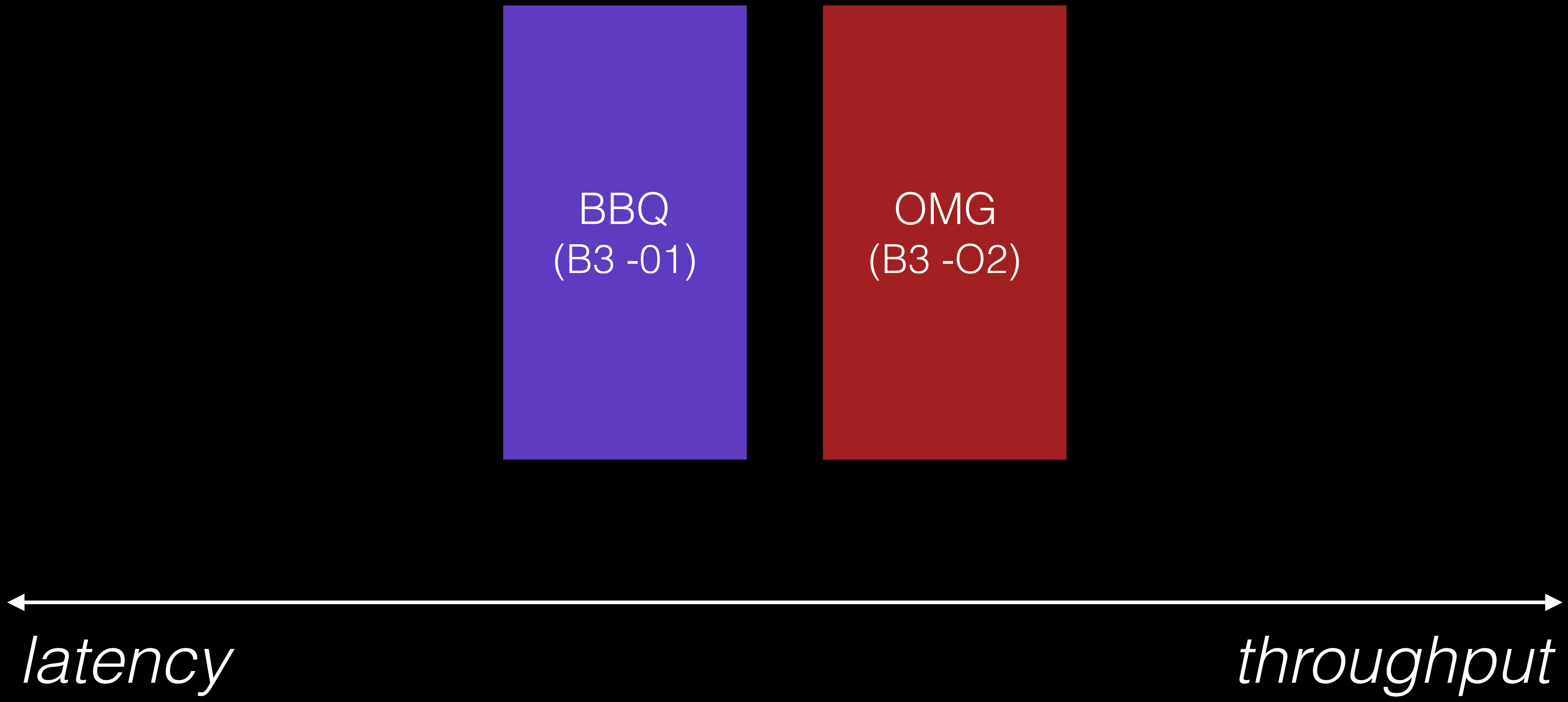
Air Backend

Assembly IR

Agenda

- High Level Overview
- Template JITing
- Optimized JITing
    - DFG
    - FTL
    - BBQ
    - OMG

Two WebAssembly Tiers



BBQ
(B3 -01)

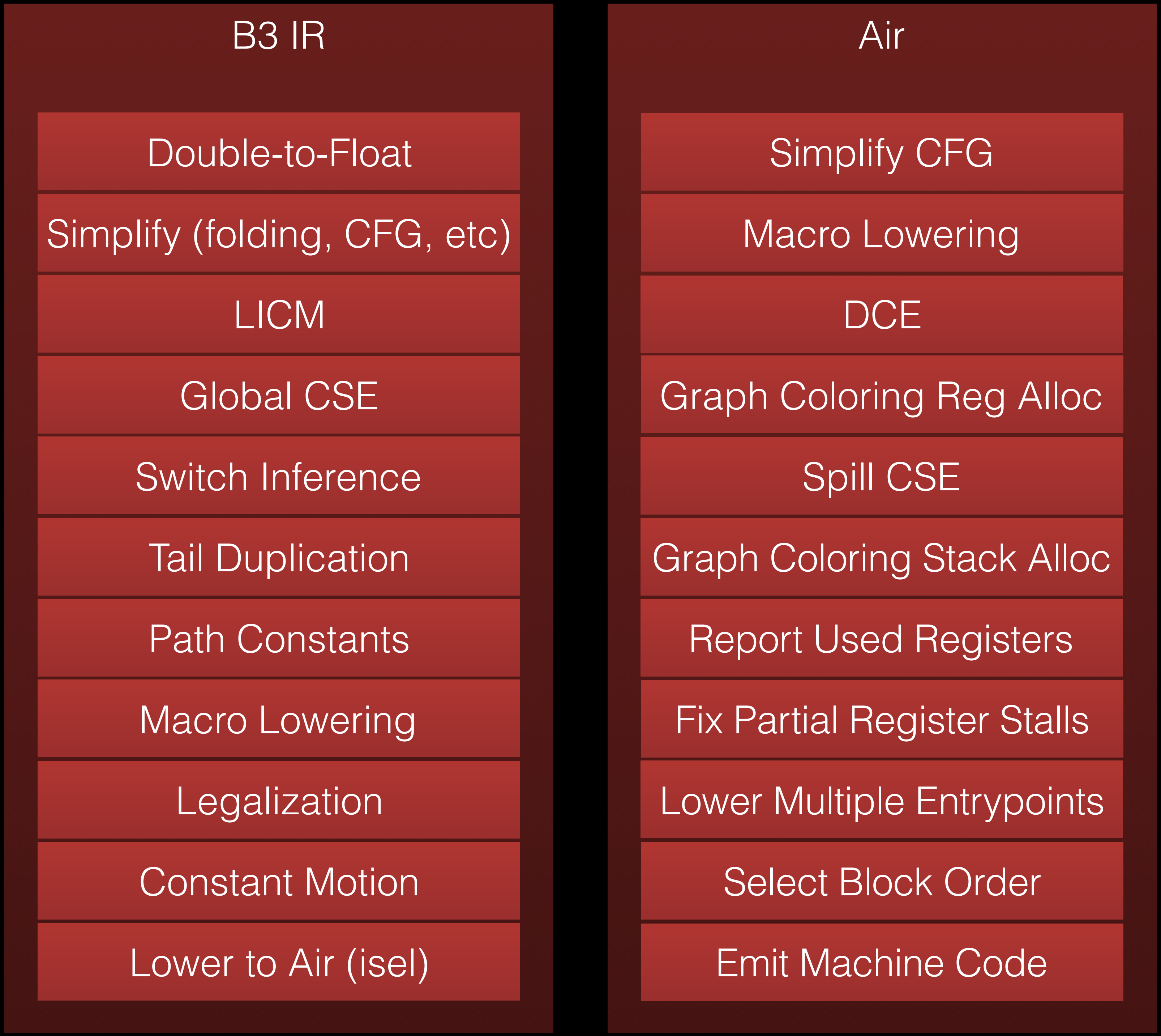
OMG
(B3 -O2)

latency

throughput

OMG

Powerful JIT



B3 IR

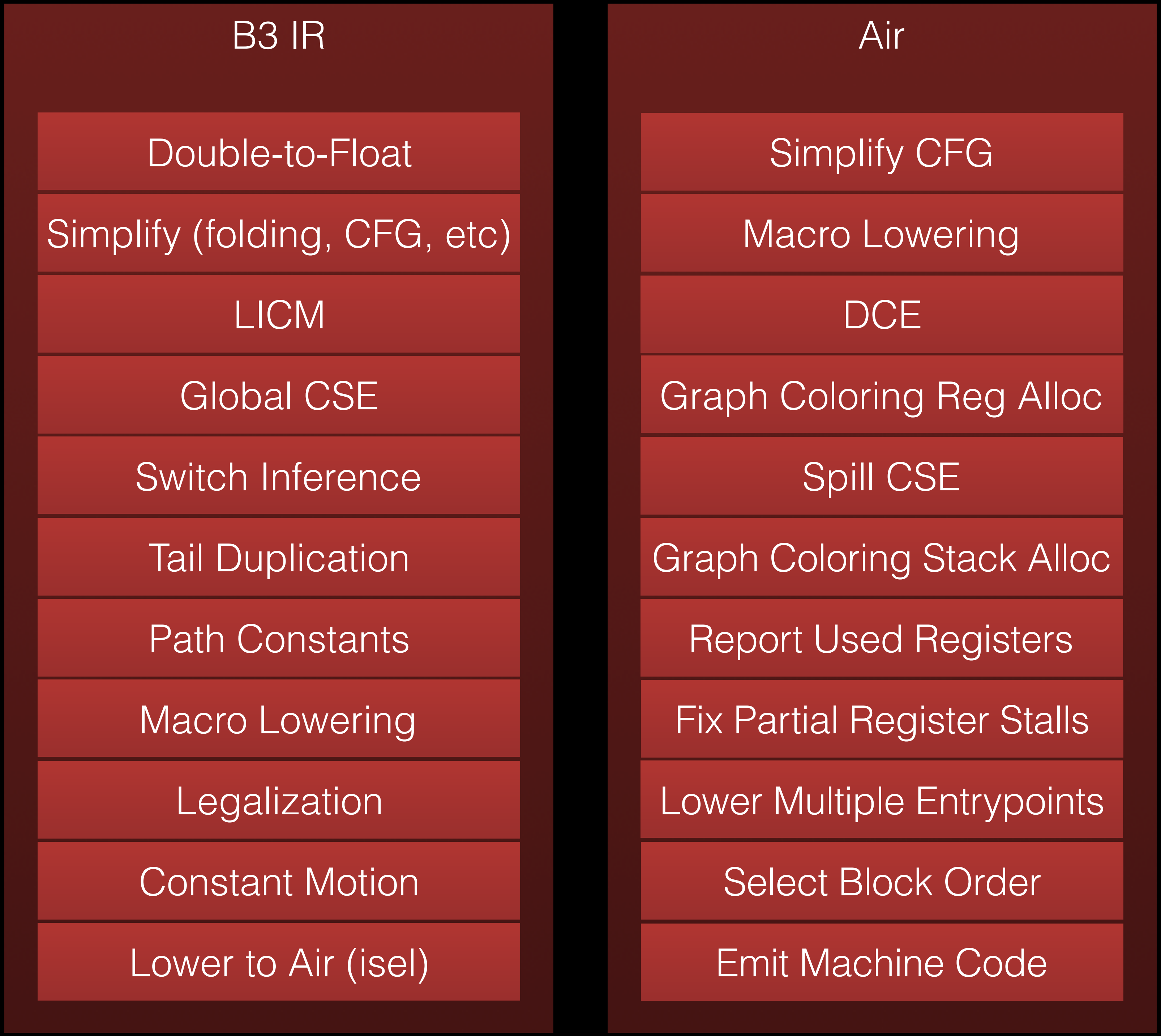
- Double-to-Float
- Simplify (folding, CFG, etc)
- LICM
- Global CSE
- Switch Inference
- Tail Duplication
- Path Constants
- Macro Lowering
- Legalization
- Constant Motion
- Lower to Air (isel)

Air

- Simplify CFG
- Macro Lowering
- DCE
- Graph Coloring Reg Alloc
- Spill CSE
- Graph Coloring Stack Alloc
- Report Used Registers
- Fix Partial Register Stalls
- Lower Multiple Entrypoints
- Select Block Order
- Emit Machine Code

BBQ

Fast JIT



B3 IR

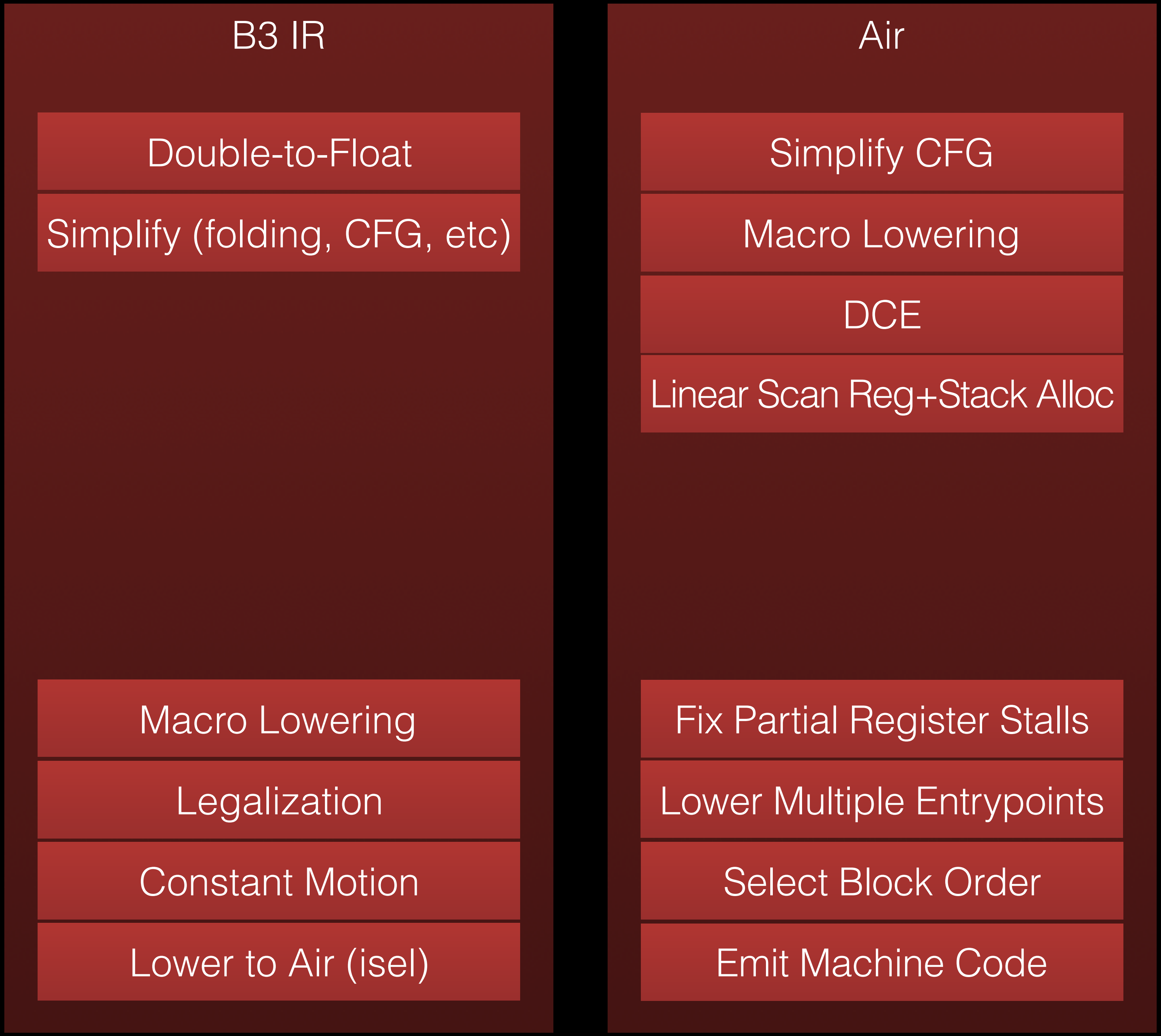
- Double-to-Float
- Simplify (folding, CFG, etc)
- LICM
- Global CSE
- Switch Inference
- Tail Duplication
- Path Constants
- Macro Lowering
- Legalization
- Constant Motion
- Lower to Air (isel)

Air

- Simplify CFG
- Macro Lowering
- DCE
- Graph Coloring Reg Alloc
- Spill CSE
- Graph Coloring Stack Alloc
- Report Used Registers
- Fix Partial Register Stalls
- Lower Multiple Entrypoints
- Select Block Order
- Emit Machine Code

BBQ

Fast JIT



B3 IR

Double-to-Float

Simplify (folding, CFG, etc)

Macro Lowering

Legalization

Constant Motion

Lower to Air (isel)

Air

Simplify CFG

Macro Lowering

DCE

Linear Scan Reg+Stack Alloc

Fix Partial Register Stalls

Lower Multiple Entrypoints

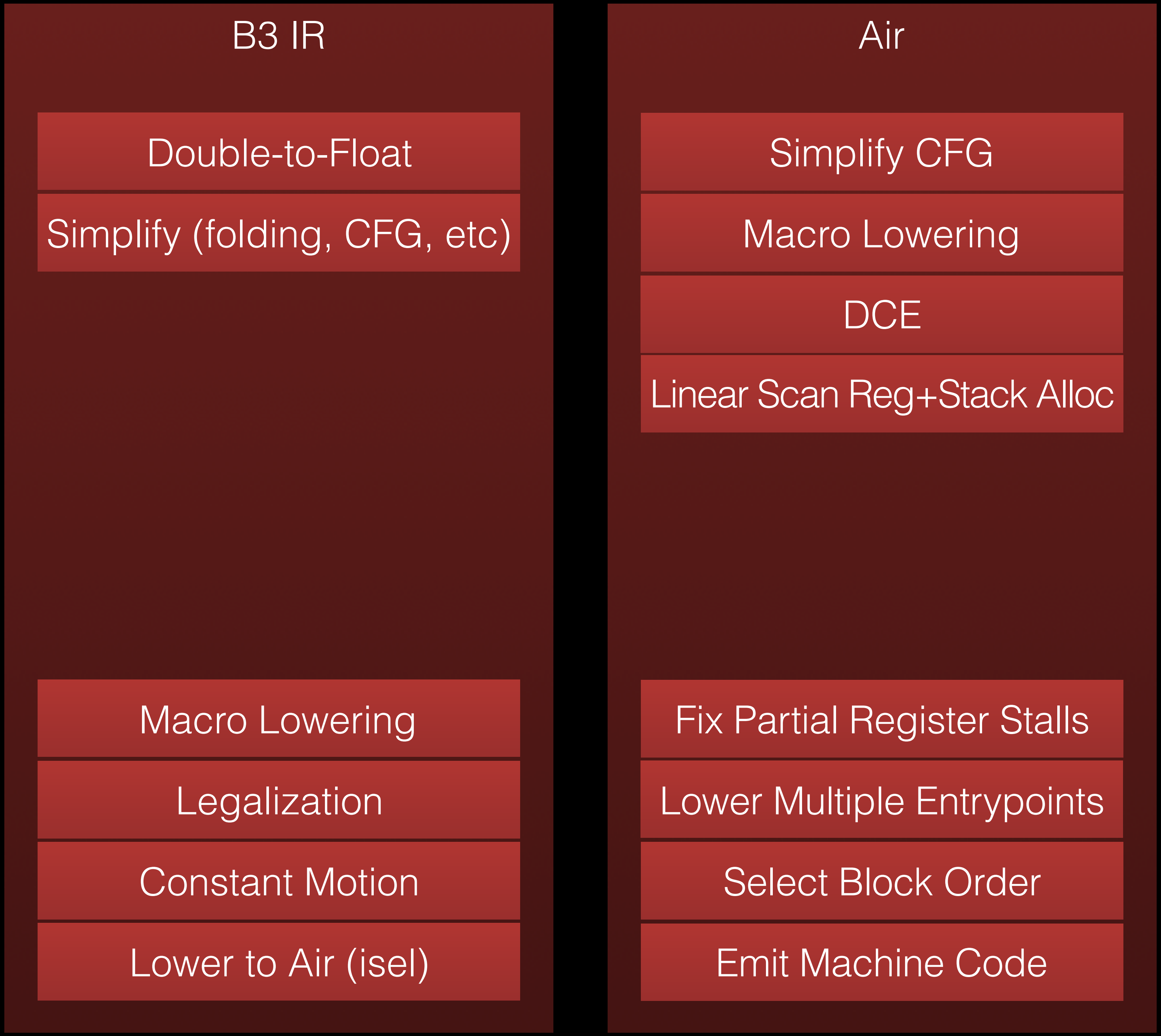
Select Block Order

Emit Machine Code

BBQ

Fast JIT

5× faster compile than OMG



B3 IR

Double-to-Float

Simplify (folding, CFG, etc)

Macro Lowering

Legalization

Constant Motion

Lower to Air (isel)

Air

Simplify CFG

Macro Lowering

DCE

Linear Scan Reg+Stack Alloc

Fix Partial Register Stalls

Lower Multiple Entrypoints

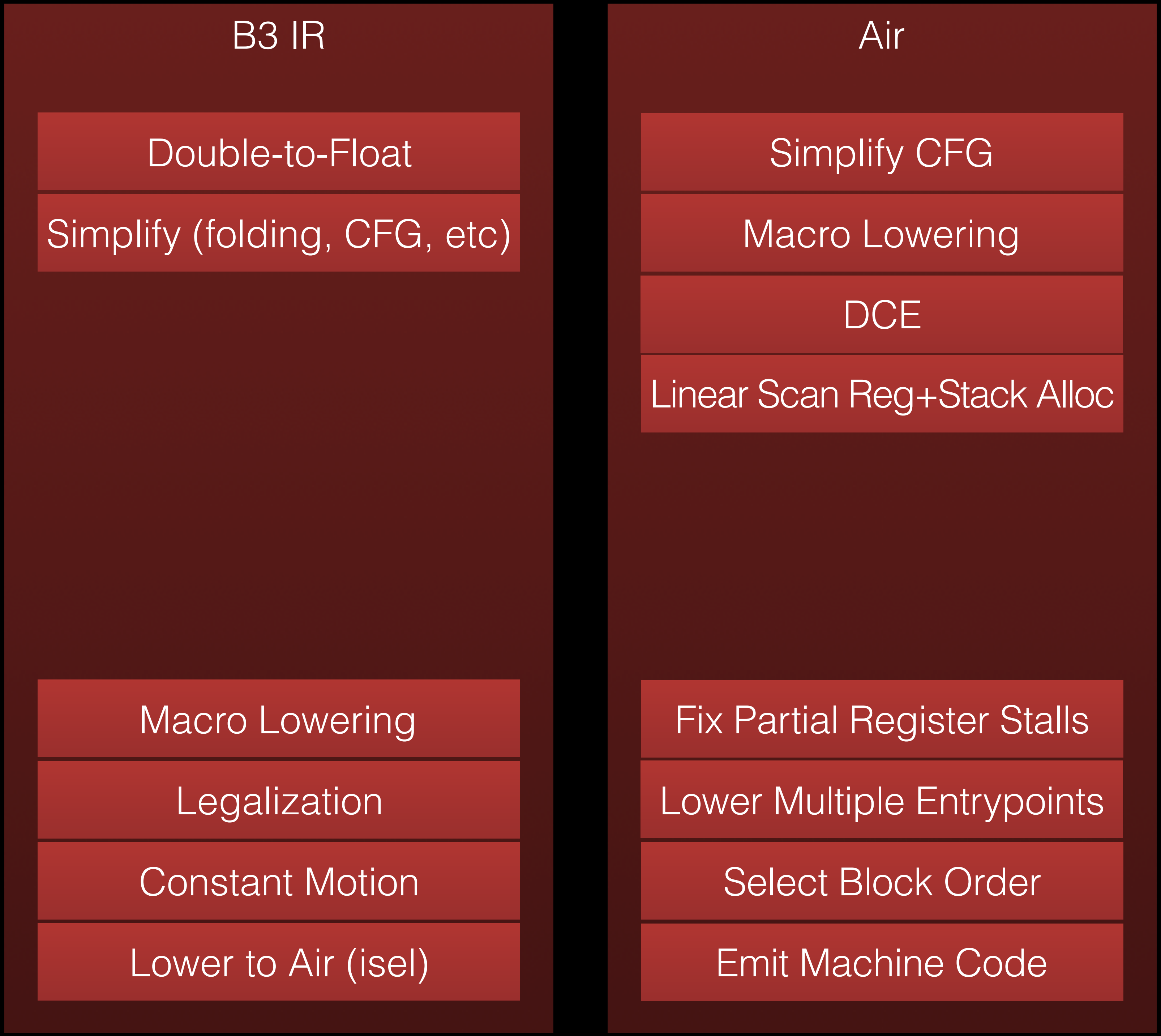
Select Block Order

Emit Machine Code

BBQ

Fast JIT

5× faster compile than OMG
2× slower execution than OMG



B3 IR

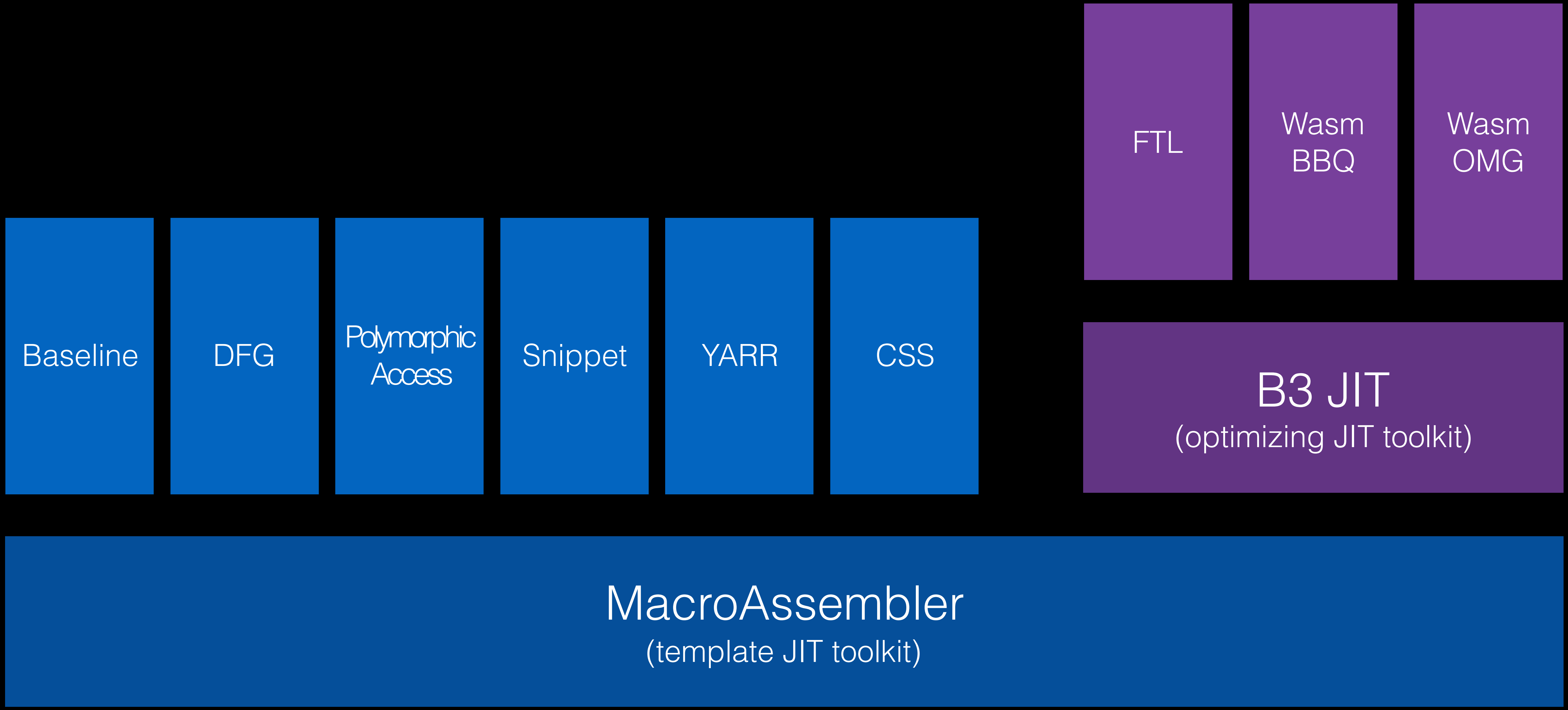
- Double-to-Float
- Simplify (folding, CFG, etc)
- Macro Lowering
- Legalization
- Constant Motion
- Lower to Air (isel)

Air

- Simplify CFG
- Macro Lowering
- DCE
- Linear Scan Reg+Stack Alloc
- Fix Partial Register Stalls
- Lower Multiple Entrypoints
- Select Block Order
- Emit Machine Code

Agenda

- High Level Overview
- Template JITing
- Optimized JITing
  - DFG
  - FTL
  - BBQ
  - OMG



FTL

Wasm BBQ

Wasm OMG

Baseline

DFG

Polymorphic Access

Snippet

YARR

CSS

B3 JIT
(optimizing JIT toolkit)

MacroAssembler
(template JIT toolkit)