Stochastic Multi-CAS

Filip Pizlo
Purdue University

Crazy Idea Talk

ISMM | 22 Oct 2007 | Montreal

Compare and Swap

```
If Target = Expected Value
Then
    Target := New Value
End
Return Old Value
```

Compare and Swap

- Compare and Swap (CAS) is essential for implementing interesting lock-free algorithms.
- But existing CAS implementations are quite constrained...

The Problem

- Hardware gives at best a 128-bit CAS. The bits must be contiguous in memory.
- Lock-free software CAS implementations can relax this constraint - but they do so by stealing bits.

Why should you care?

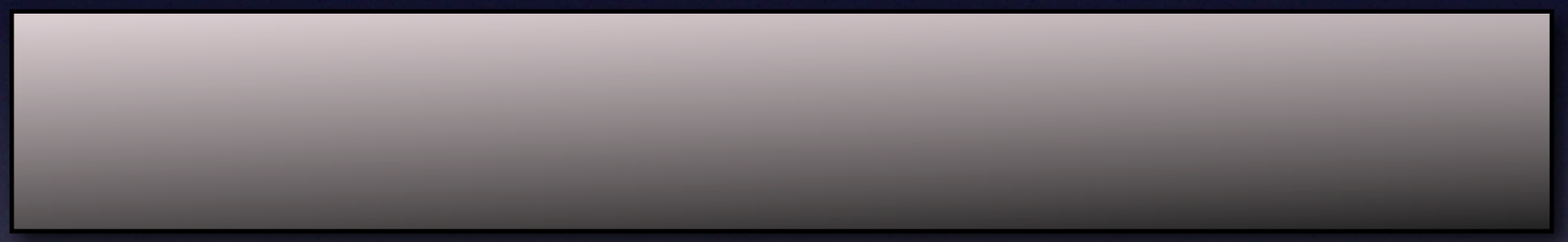
- Lots of algorithms can be easily made lock-free if we had practical multi-CAS.
- Example: concurrent GC.
- But bit stealing is intrusive.

- Stealing a bit is intrusive because:
  - vanilla C/C#/Java primitive types
- we require that the client software be designed with an a priori knowledge about bit stealing.

Harris approach

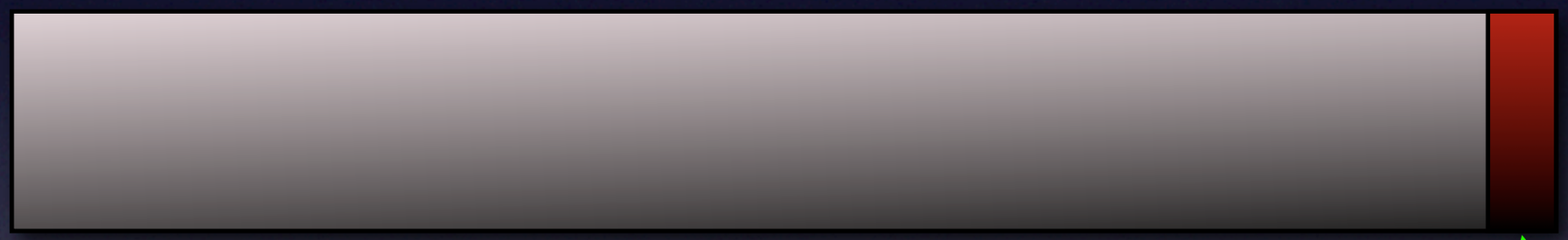
Harris approach

Hardware CAS-able word



Harris approach

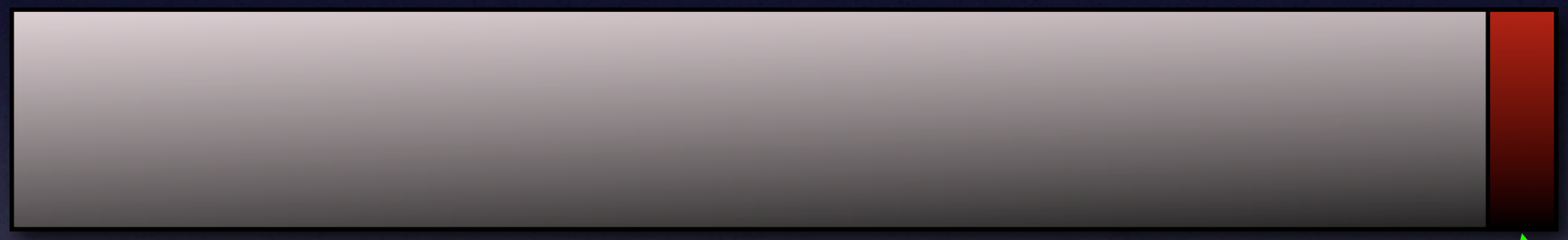
Hardware CAS-able word



Steal one bit

Harris approach

Hardware CAS-able word



Either payload
-or-
CASN control data

Steal one bit

Can we get 1 bit without stealing it?

Is there some way to cheat?

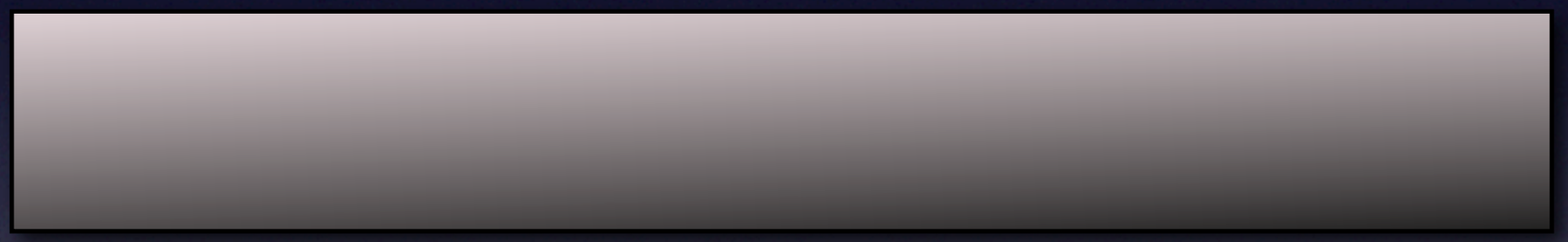
Yes!

Use a random number!

Stochastic approach

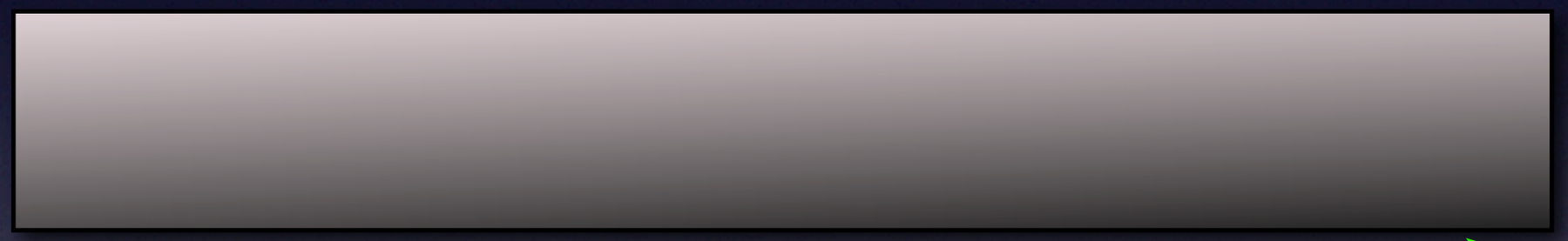
Stochastic approach

Hardware CAS-able word



Stochastic approach

Hardware CAS-able word



All bits available

for payload!

Stochastic approach

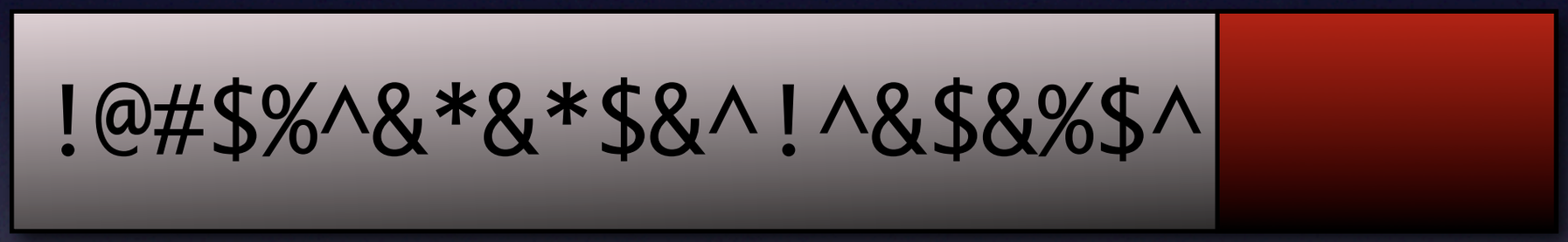
Hardware CAS-able word

!@#$%^&*&*$&^!^&$&%$^

When it comes to run multi-CAS, store random marker.

Stochastic approach

Hardware CAS-able word



When marker present, use remaining bits for multi-CAS control data.

Why is it good?

- Can multi-CAS any field (including primitive fields)
- The types don’t need to change
- Lock-free, performance need not be atrocious.

What is the challenge?

- Convincing people to use a stochastic algorithm.

Conclusion

- An implementation already exists:

http://homepage.mac.com/pizlo/smcas

Conclusion

- An implementation already exists:

http://homepage.mac.com/pizlo/smcas

CAS throughput: 92ns/64bit!

Conclusion

- An implementation already exists:

http://homepage.mac.com/pizlo/smcas

CAS throughput: 92ns/64bit!
(Upper bounds. It’s faster per-field for larger Multi-CAS operations.)

Conclusion

- An implementation already exists:

http://homepage.mac.com/pizlo/smcas

CAS throughput: 92ns/64bit!
(Upper bounds. It's faster per-field for larger Multi-CAS operations.)

Read barrier throughput: ~2ns/64bit!