Does Hardware Transactional Memory Change Everything?

Maurice Herlihy
Computer Science Dept
Brown University

IBM's new transactional memory: make-or-break time for multithreaded revolution

At Hot Chips last week, IBM talked about BlueGene/Q, the processor powering ...

by **Peter Bright** - Aug 31 2011, 5:15pm EDT

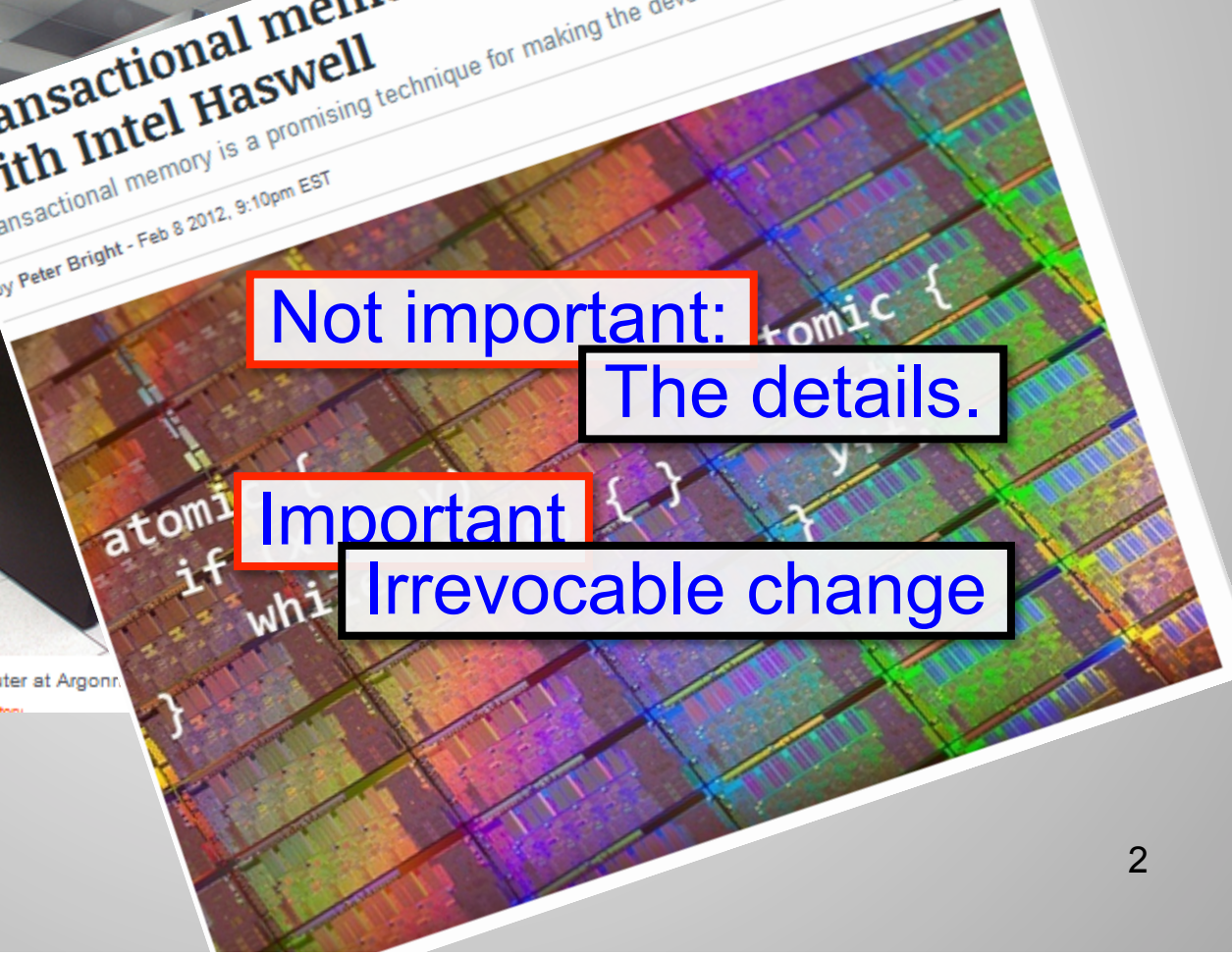
The Blue Gene / P supercomputer at Argonn

Transactional memory going mainstream with Intel Haswell

Transactional memory is a promising technique for making the development of ...

by **Peter Bright** - Feb 8 2012, 9:10pm EST

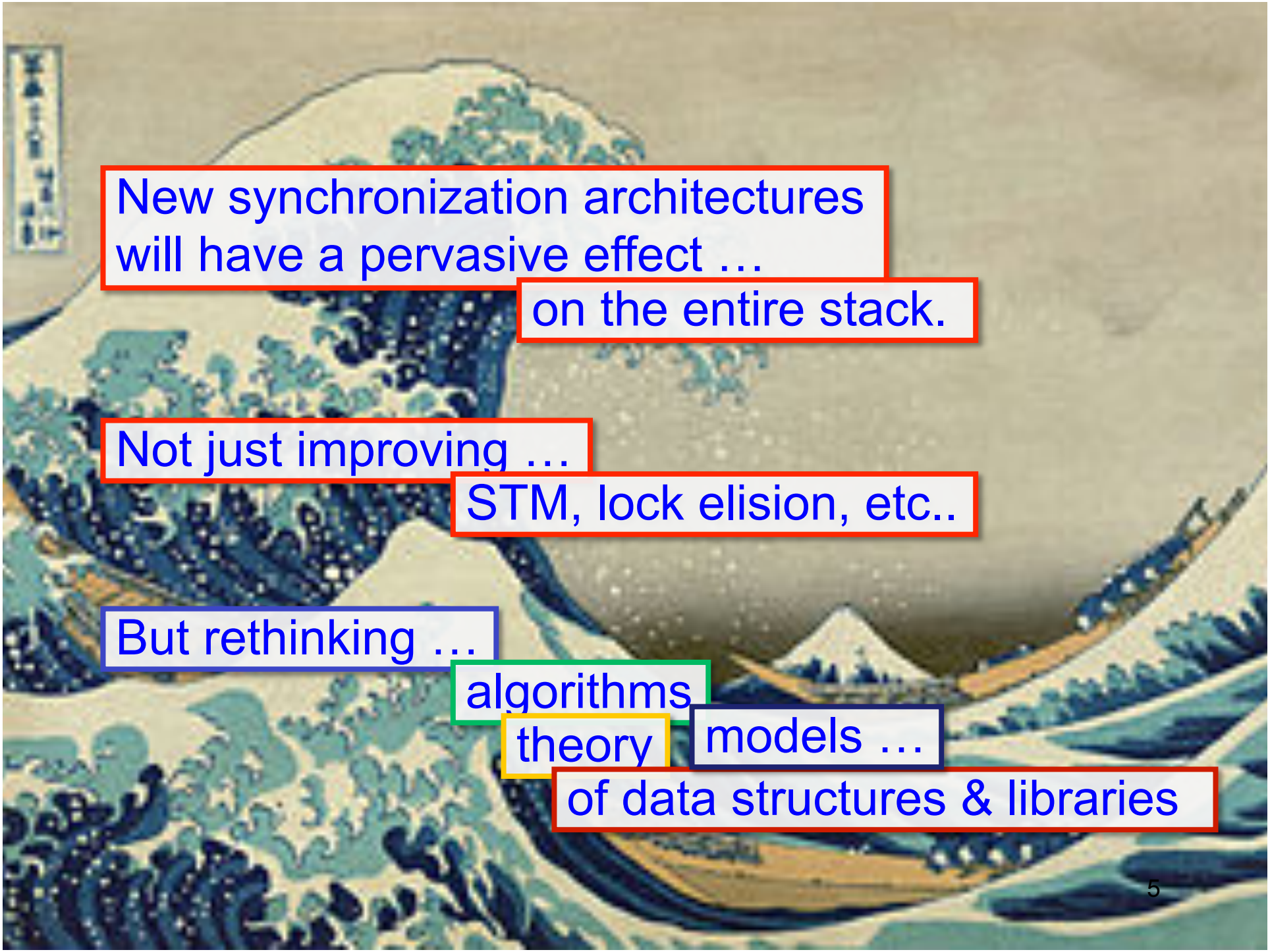
68



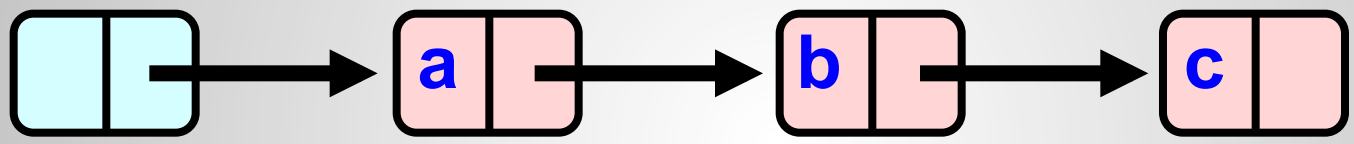

How to think about
Synchronization

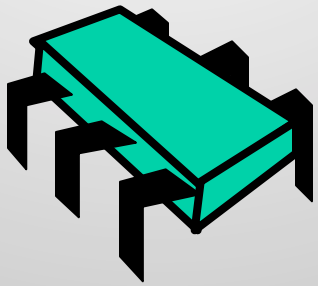
Amdal's Law

Poor synchronization ruins everything

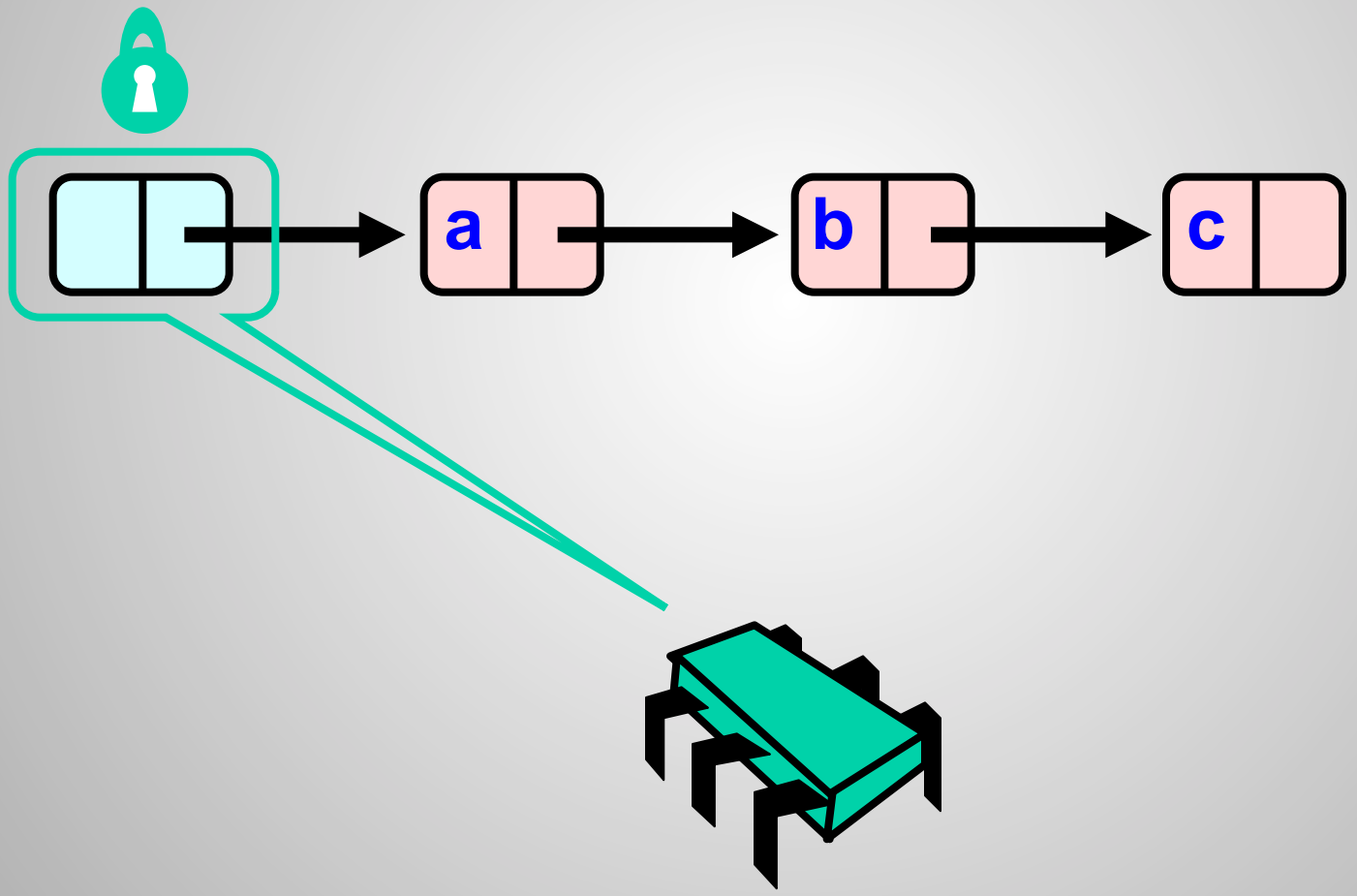

New synchronization architectures will have a pervasive effect …
on the entire stack.
Not just improving …
STM, lock elision, etc..
But rethinking …
algorithms
theory
models …
of data structures & libraries

Hand-over-Hand locking

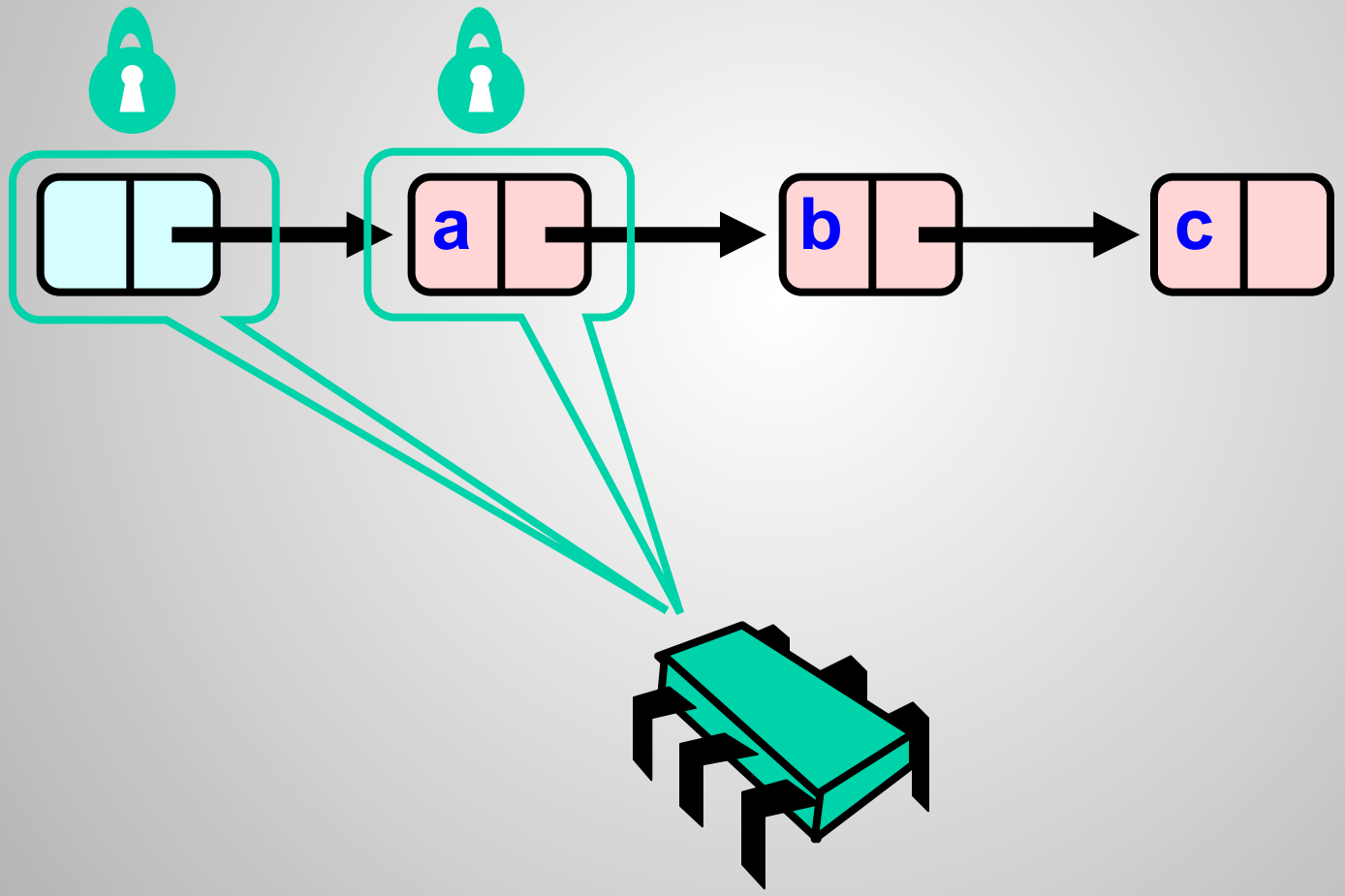




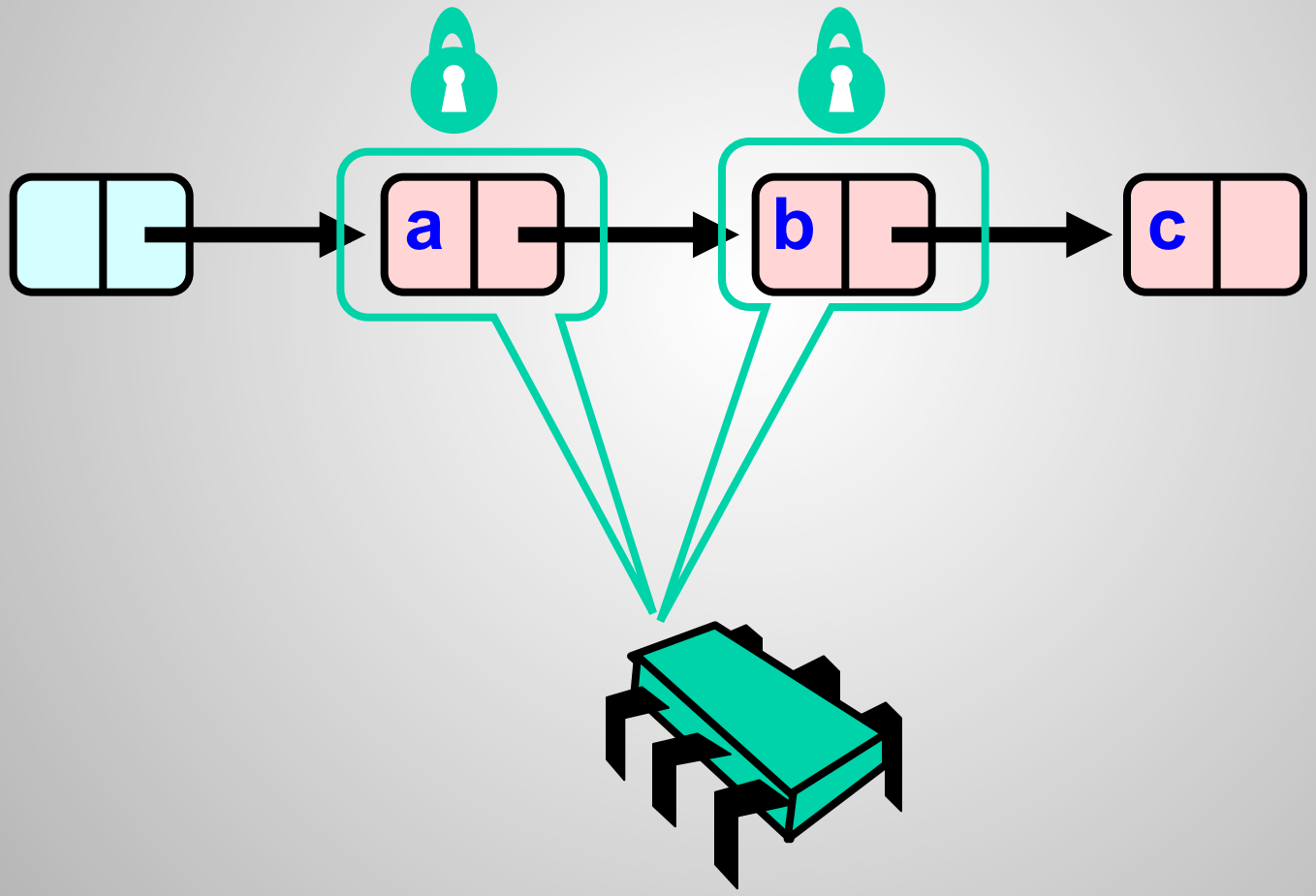
Hand-over-Hand locking



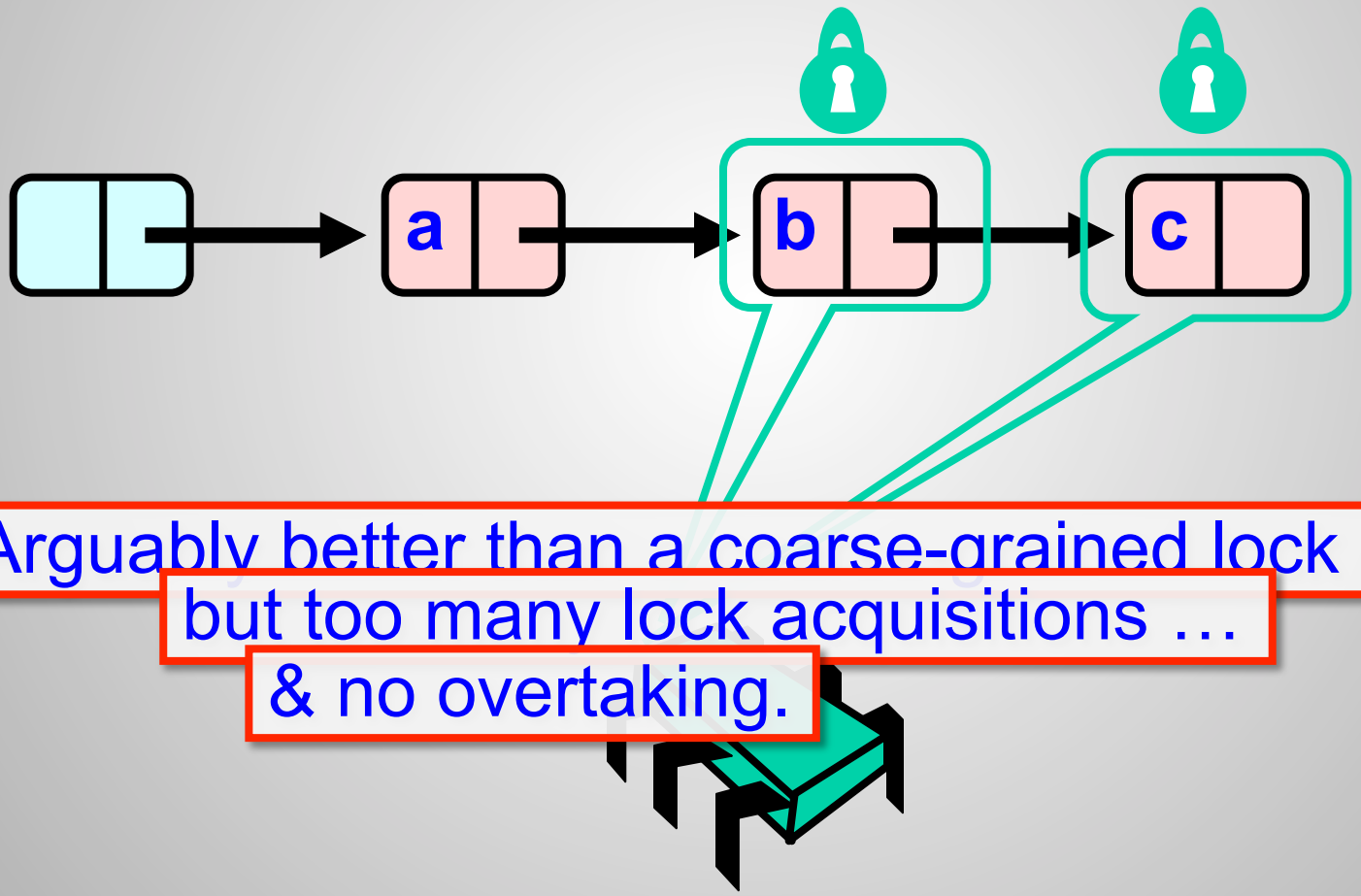
Hand-over-Hand locking



Hand-over-Hand locking



Hand-over-Hand locking

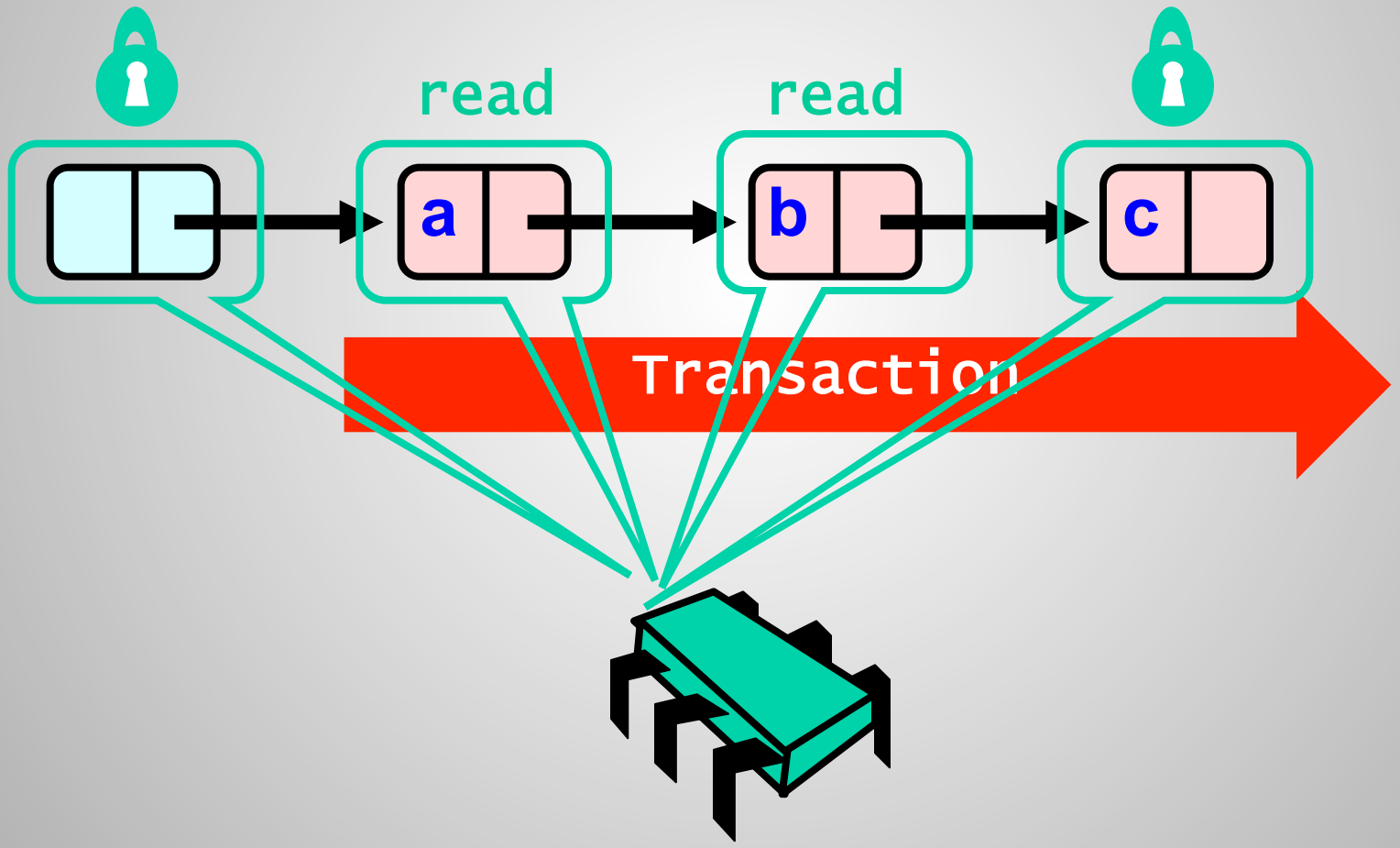


Arguably better than a coarse-grained lock …

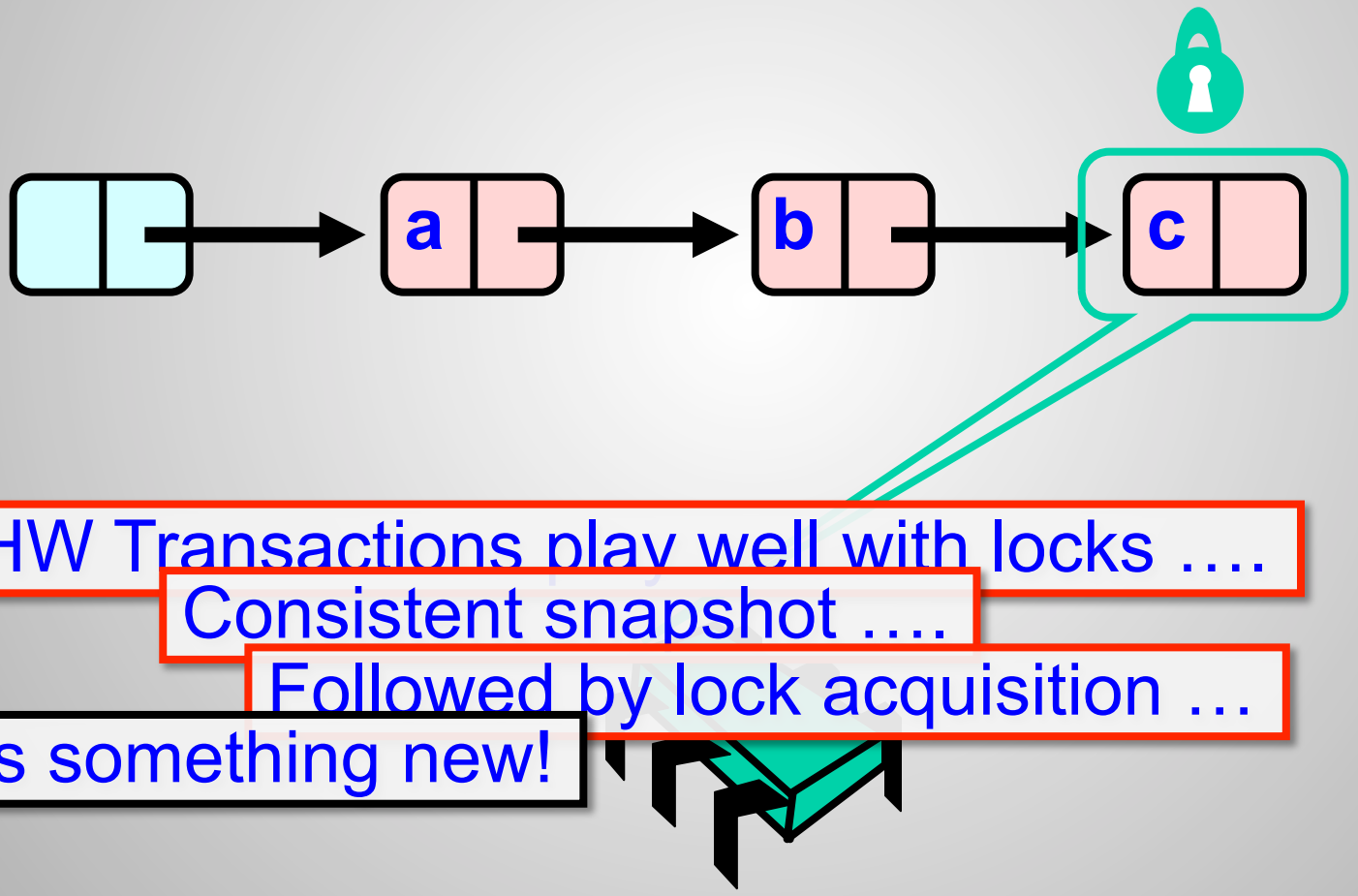
but too many lock acquisitions …

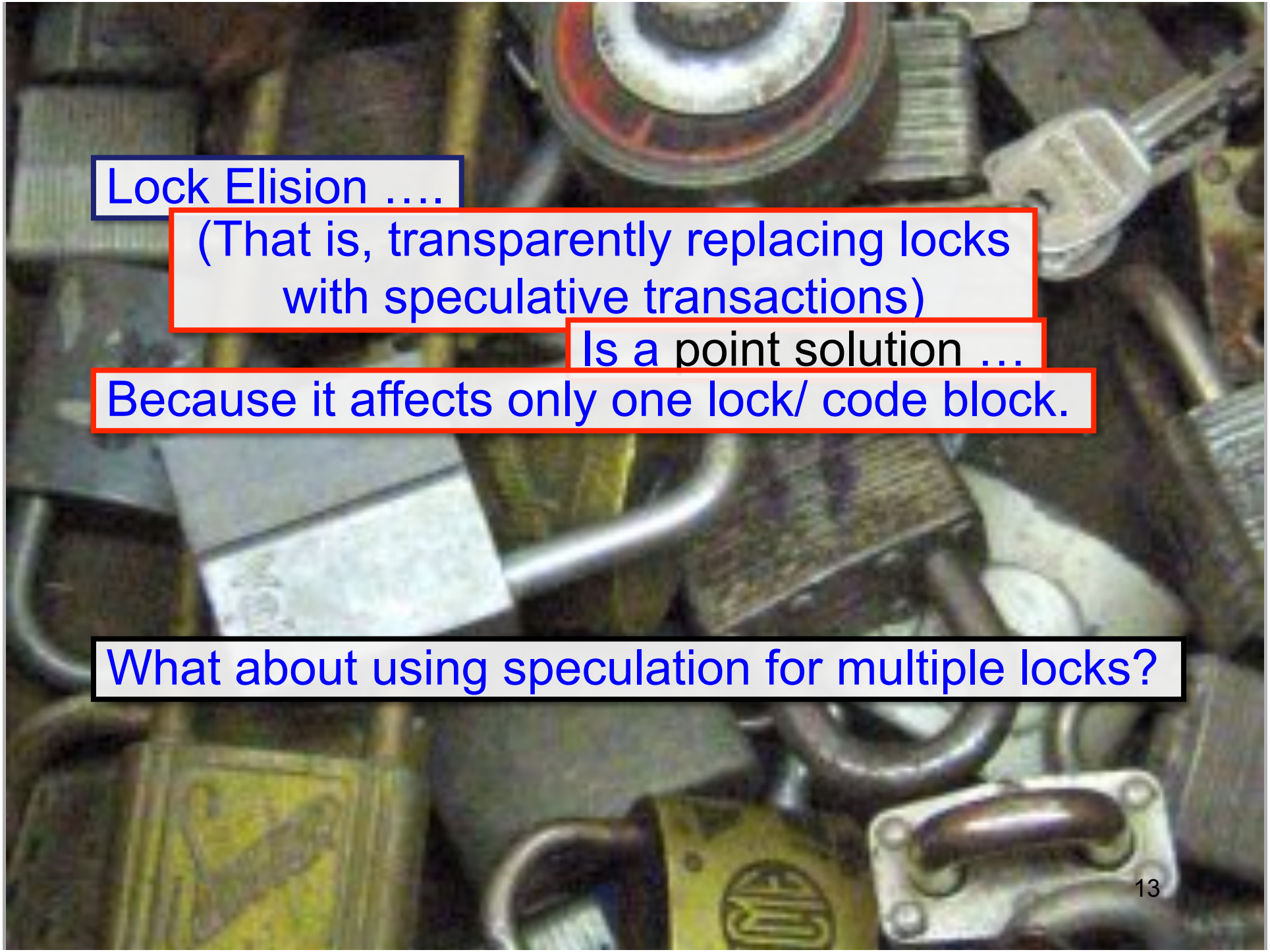
& no overtaking.

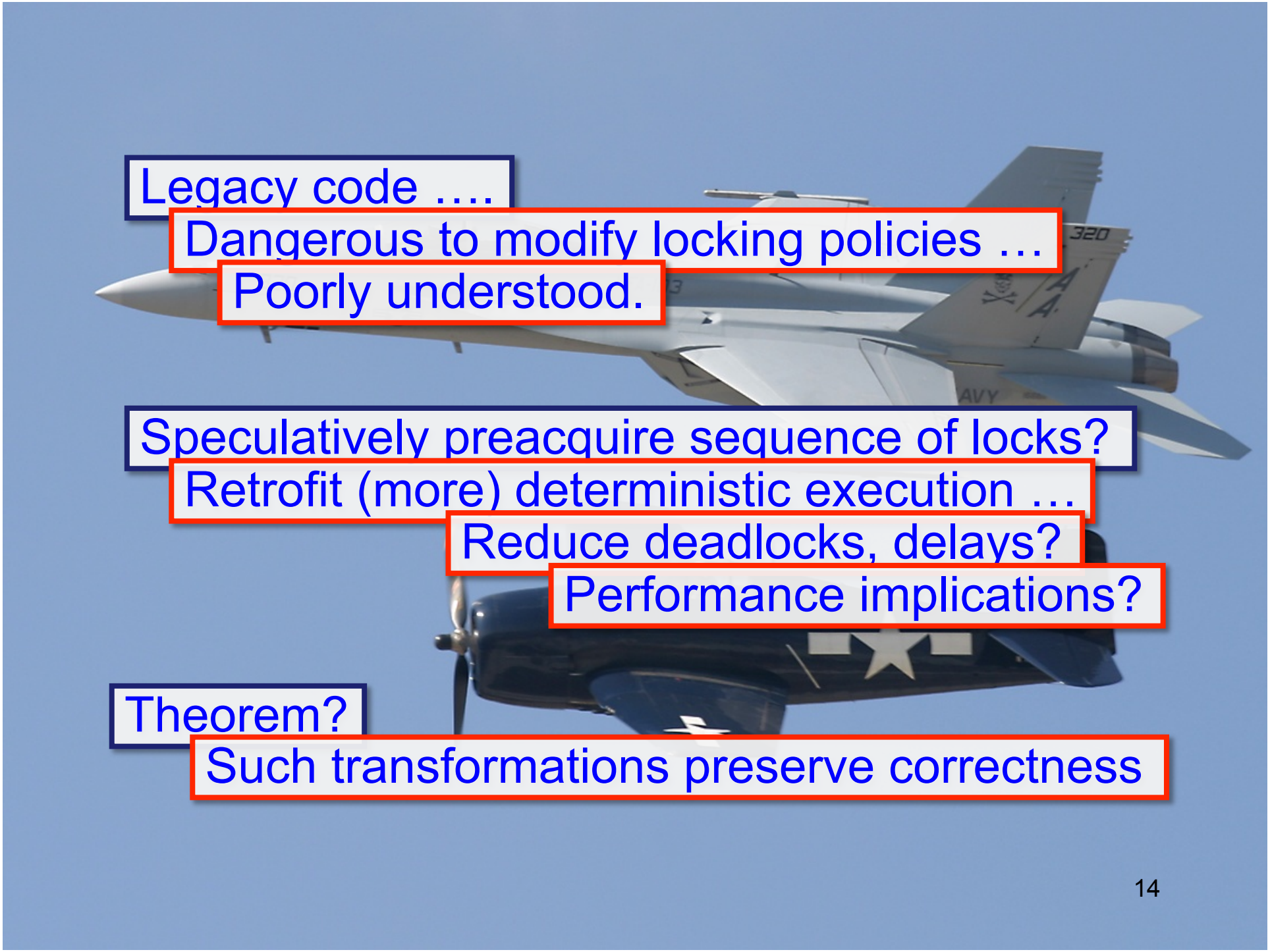
Transactional HoH locking

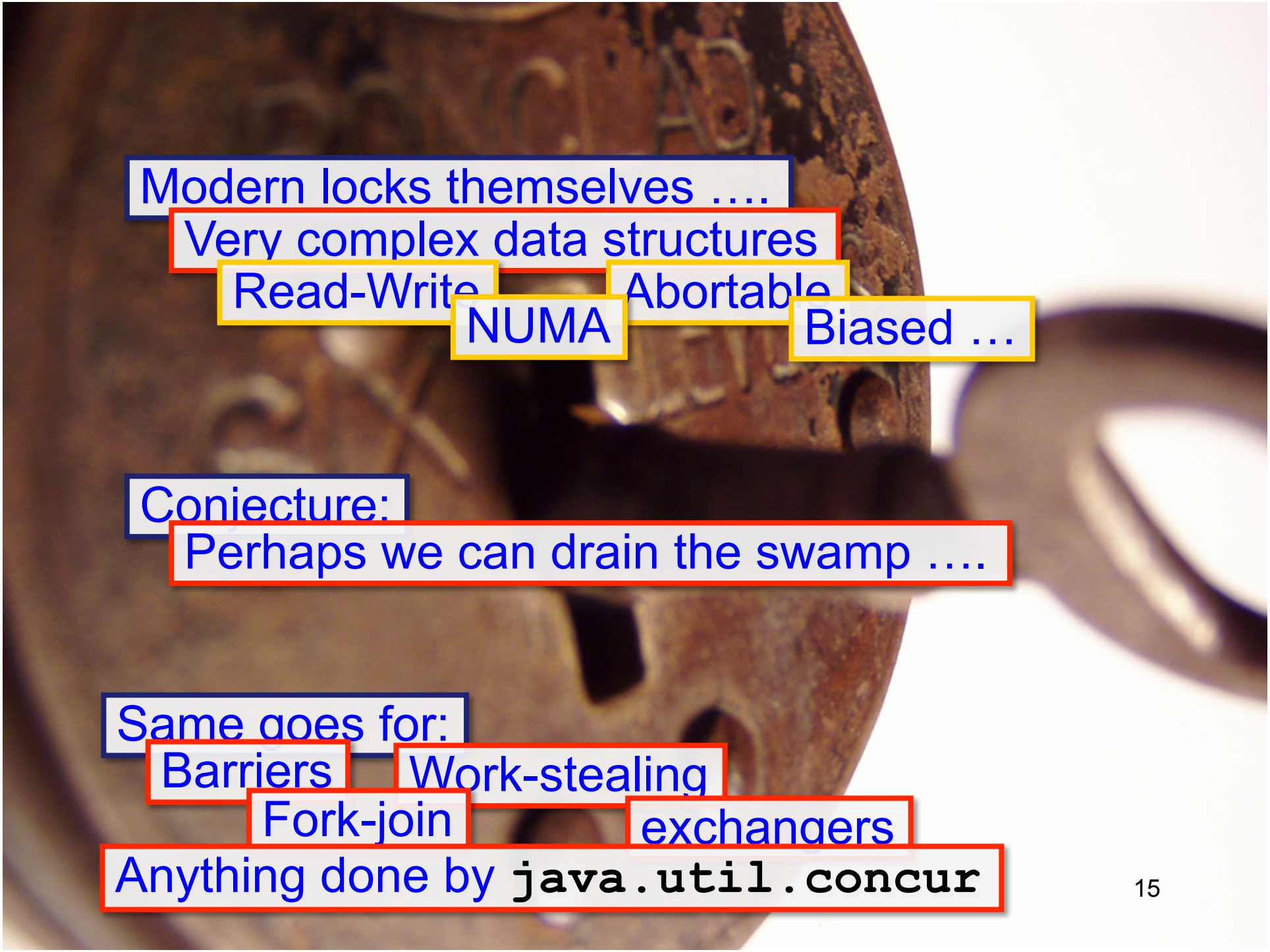


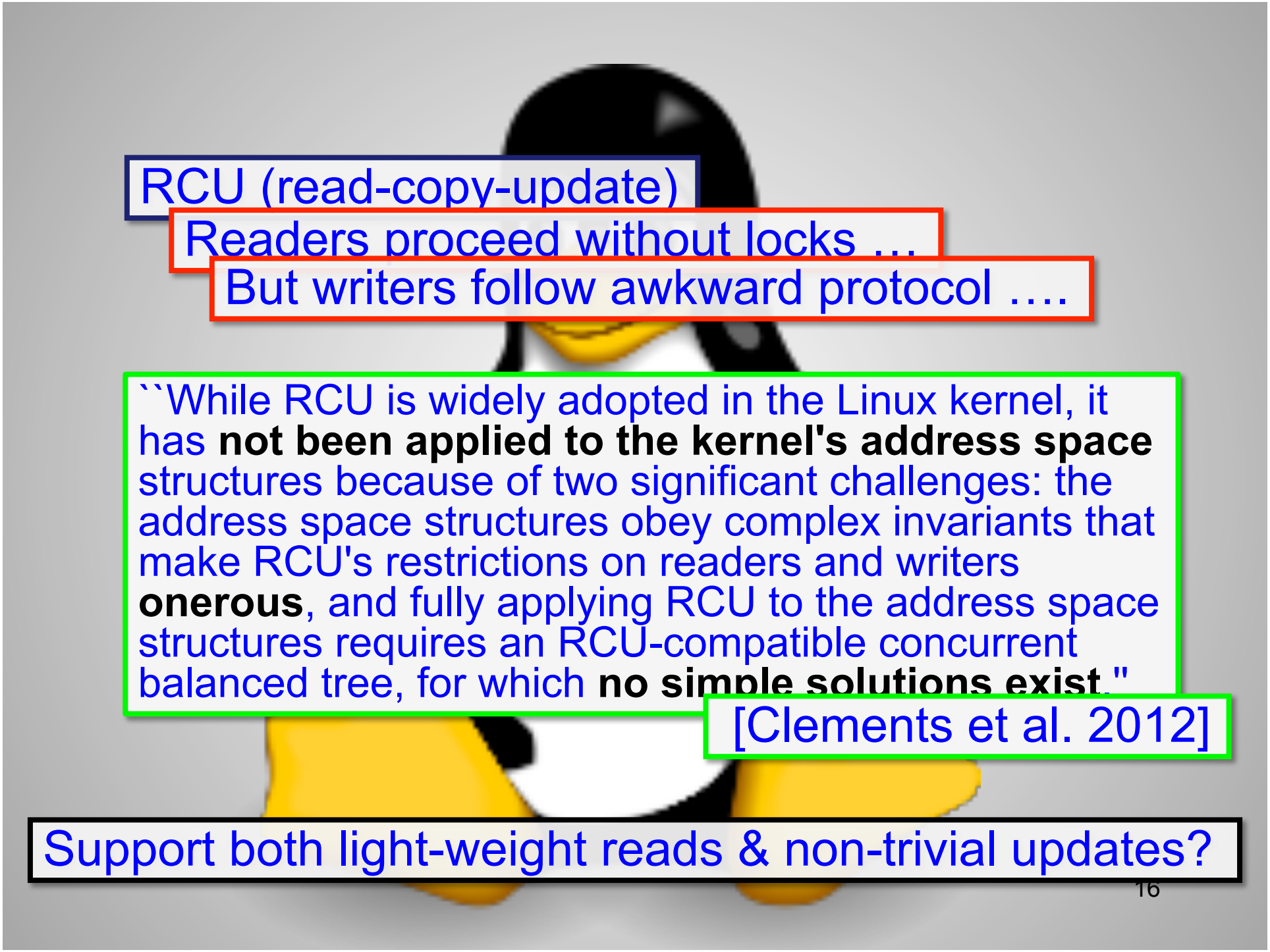
Transactional HoH locking




Lock Elision ….
(That is, transparently replacing locks with speculative transactions)
Is a point solution …
Because it affects only one lock/ code block.
What about using speculation for multiple locks?


Legacy code ….
Dangerous to modify locking policies …
Poorly understood.
Speculatively preacquire sequence of locks?
Retrofit (more) deterministic execution …
Reduce deadlocks, delays?
Performance implications?
Theorem?
Such transformations preserve correctness

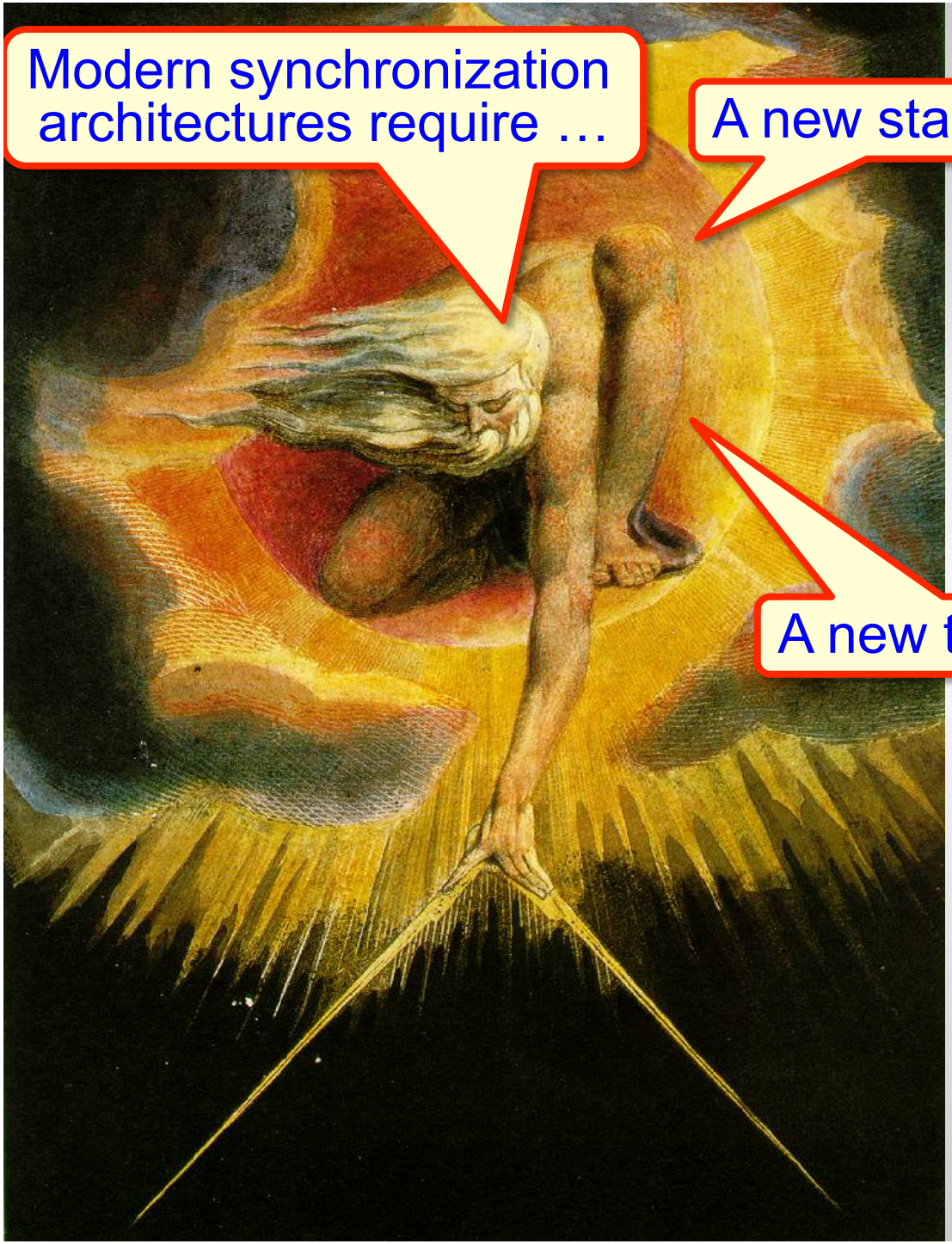

Modern locks themselves ….
Very complex data structures
Read-Write
Abortable
NUMA
Biased …
Conjecture:
Perhaps we can drain the swamp ….
Same goes for:
Barriers
Work-stealing
Fork-join
exchangers
Anything done by java.util.concur


RCU (read-copy-update)
Readers proceed without locks …
But writers follow awkward protocol ….
``While RCU is widely adopted in the Linux kernel, it has **not been applied to the kernel's address space** structures because of two significant challenges: the address space structures obey complex invariants that make RCU's restrictions on readers and writers **onerous**, and fully applying RCU to the address space structures requires an RCU-compatible concurrent balanced tree, for which **no simple solutions exist**."
[Clements et al. 2012]
Support both light-weight reads & non-trivial updates?


Memory Management?
Malloc/free vs lock-free data structures?
Reference counts?
Hazard pointers?
RCU?
Transactions can transform & improve these techniques?


Granularity?

(weasel words)



Data structures

Algorithms

Synchronization structures

OS, VM functions

progress

complexity

transformations