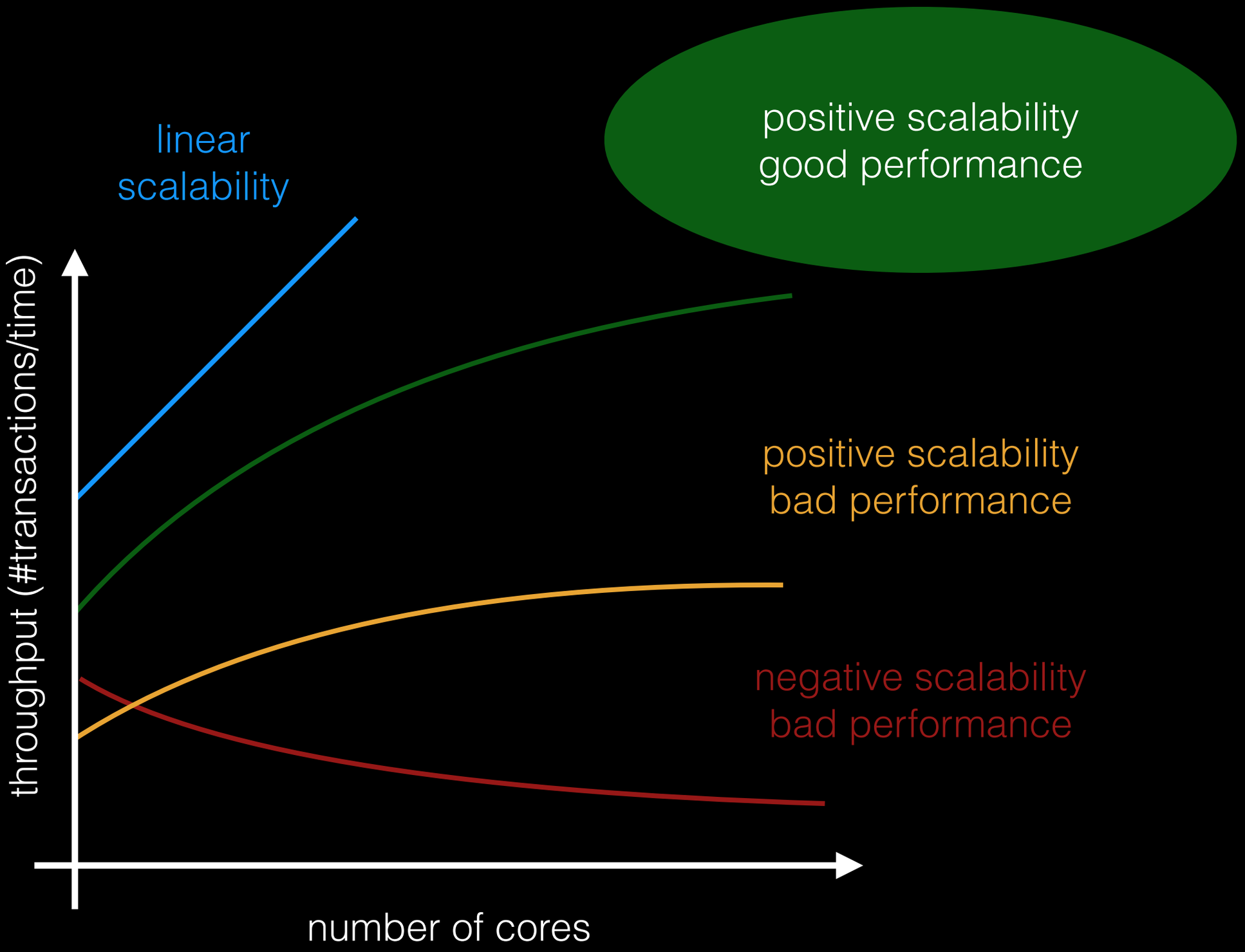
Towards Relaxing STM

Christoph M. Kirsch, **Michael Lippautz**
University of Salzburg

Euro-TM Workshop, April 2014

Problem


linear
scalability
positive scalability
good performance
positive scalability
bad performance
negative scalability
bad performance
throughput (#transactions/time)
number of cores

Scal Project

Collection of concurrent data structure implementations

Strict

- Treiber Stack
- Lock-based Queue
- Michael-Scott Queue
- Flat-Combining Queue
- Wait-free Queue
- **Timestamping Deque [SBG14]**
- Elimination-Diffraction Pool
- Rendezvousing Pool

Relaxed

- Random-Dequeue Queue
- Segmented Queue
- **k-FIFO Queues [PaCT13]**
- **k-Stack [POPL13]**
- **Distributed Queues [CF13]**

scal.cs.uni-salzburg.at

Observation

Lock-freedom does not imply scalability.

Fact

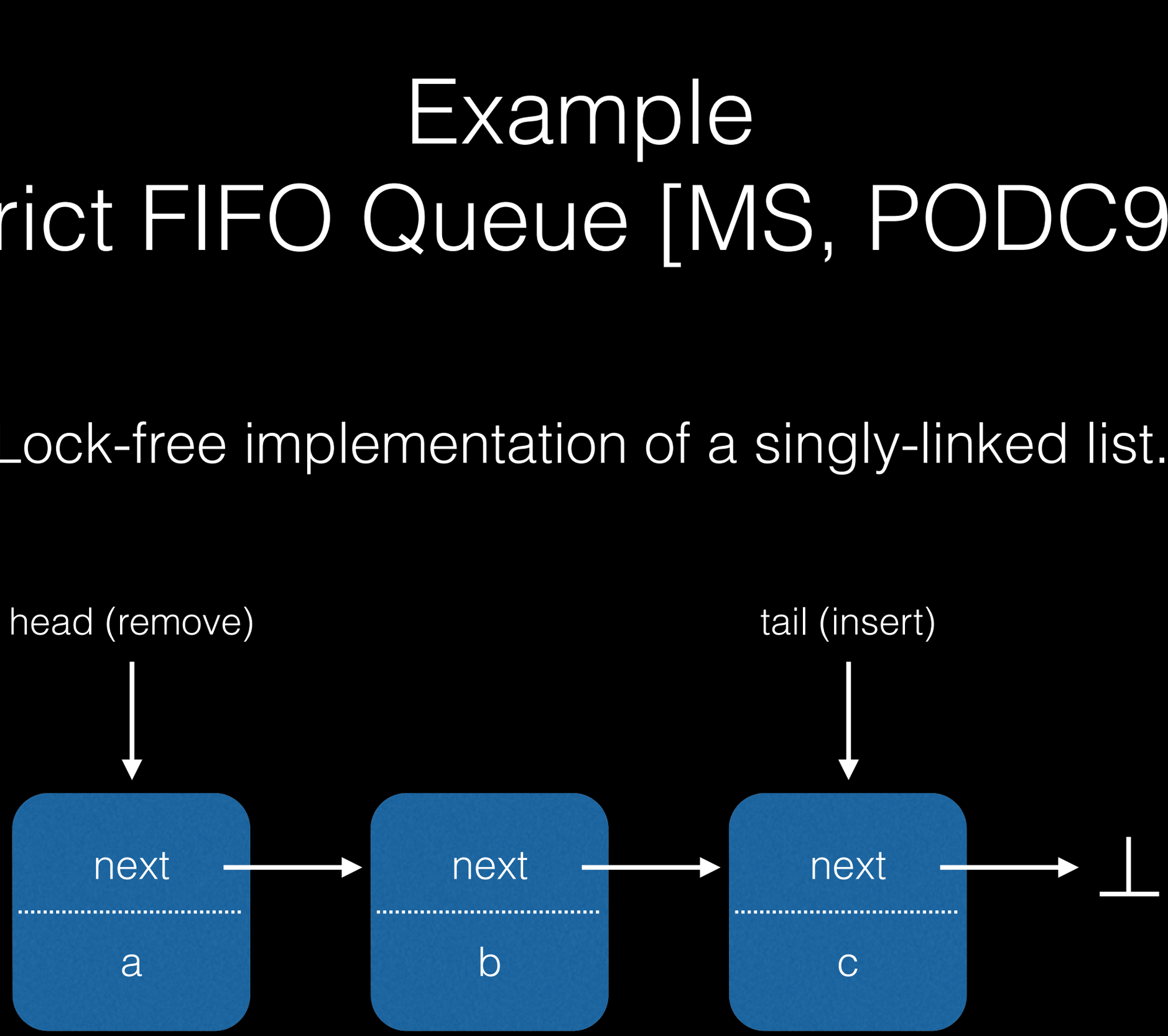
Semantics requires synchronization.
[Laws of Order, POPL11]

Claim

Relaxation is *a way* to get scalability.

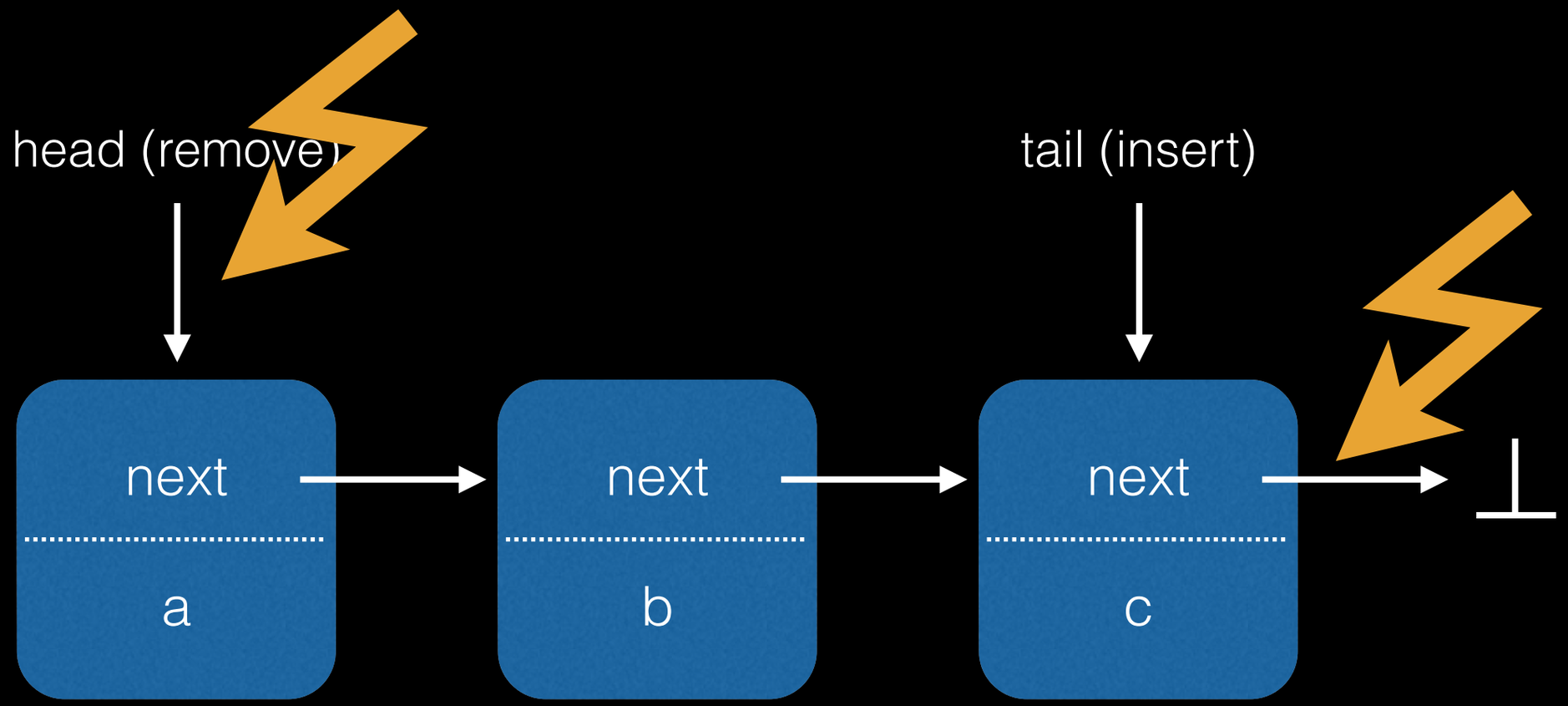
Example
Strict FIFO Queue [MS, PODC96]

Lock-free implementation of a singly-linked list.



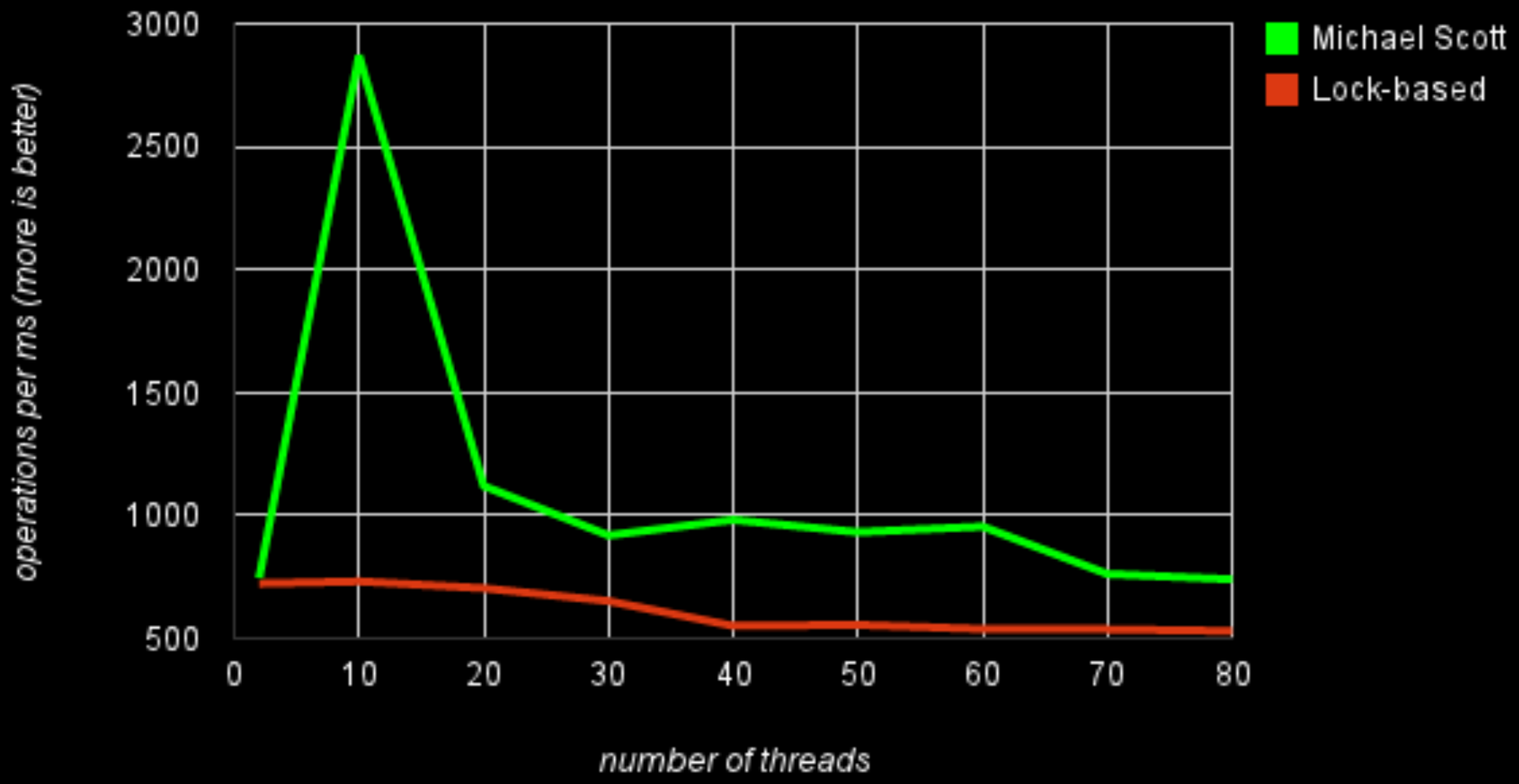
Example
Strict FIFO Queue [MS, PODC96]

Lock-free implementation of a singly-linked list.



Reality Check #1

Producer/Consumer Workload
on a 40-core (w/ 2 hyperthreads) machine



Relaxation
(Informally)

Happens on two levels

1. Weaken semantics of data structures by allowing more behavior in their specifications [POPL13]

   Example: Reordering elements (up to some constant k) in a queue or stack.

2. Use the freedom provided by the specification in the actual implementation [PaCT13, CF13, POPL13]

   Rule of thumb: Distribute contention across multiple memory locations.

Relaxation
(Informally)

Happens on two levels

1. Weaken semantics of data structures by allowing more behavior in their specifications [POPL13]

   Example: Reordering elements (up to some constant k) in a queue or stack.

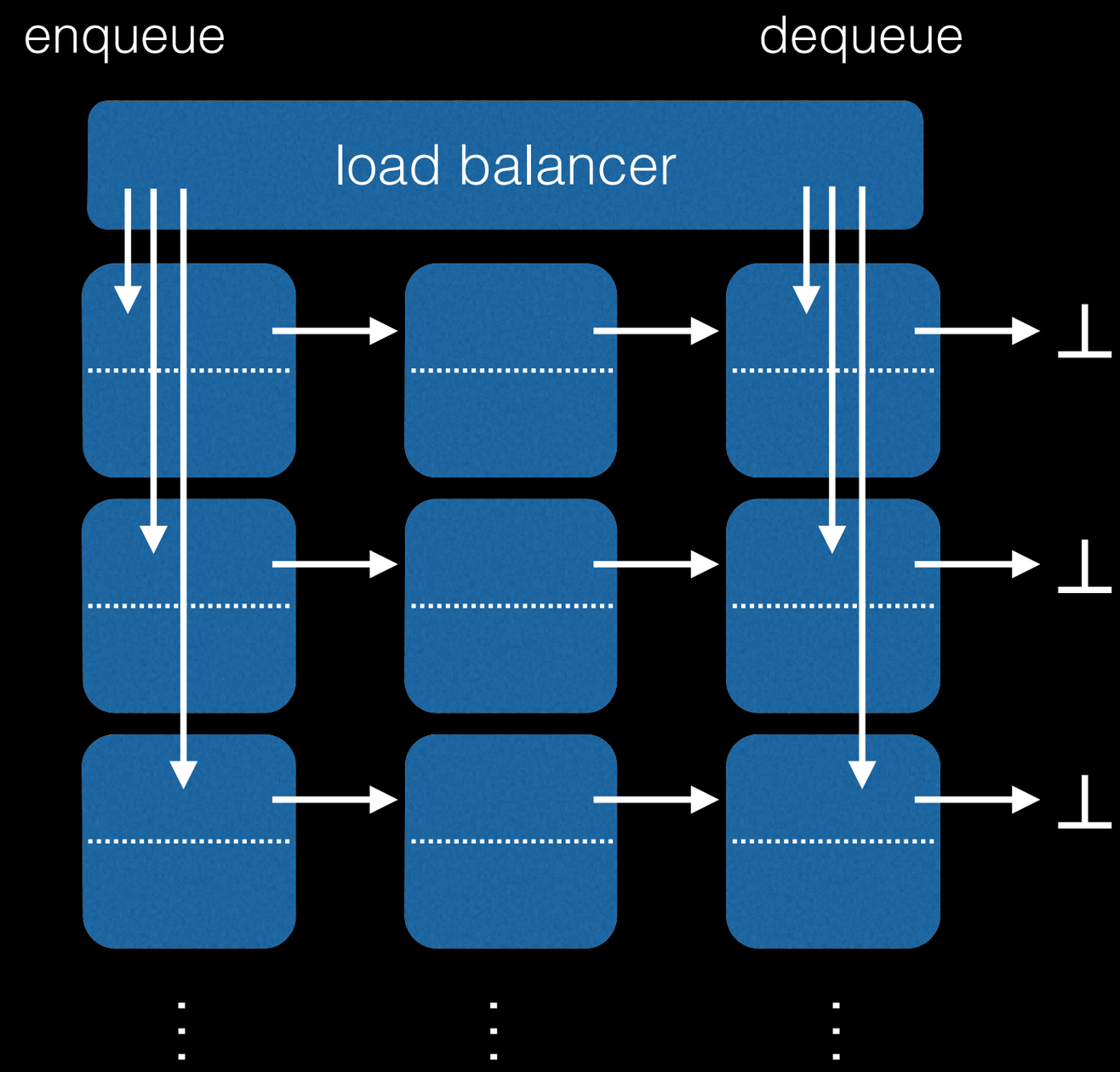
2. Use the freedom provided by the specification in the actual implementation [PaCT13, CF13, POPL13]

   Rule of thumb: Distribute contention across multiple memory locations.

Tradeoff: search time vs memory contention

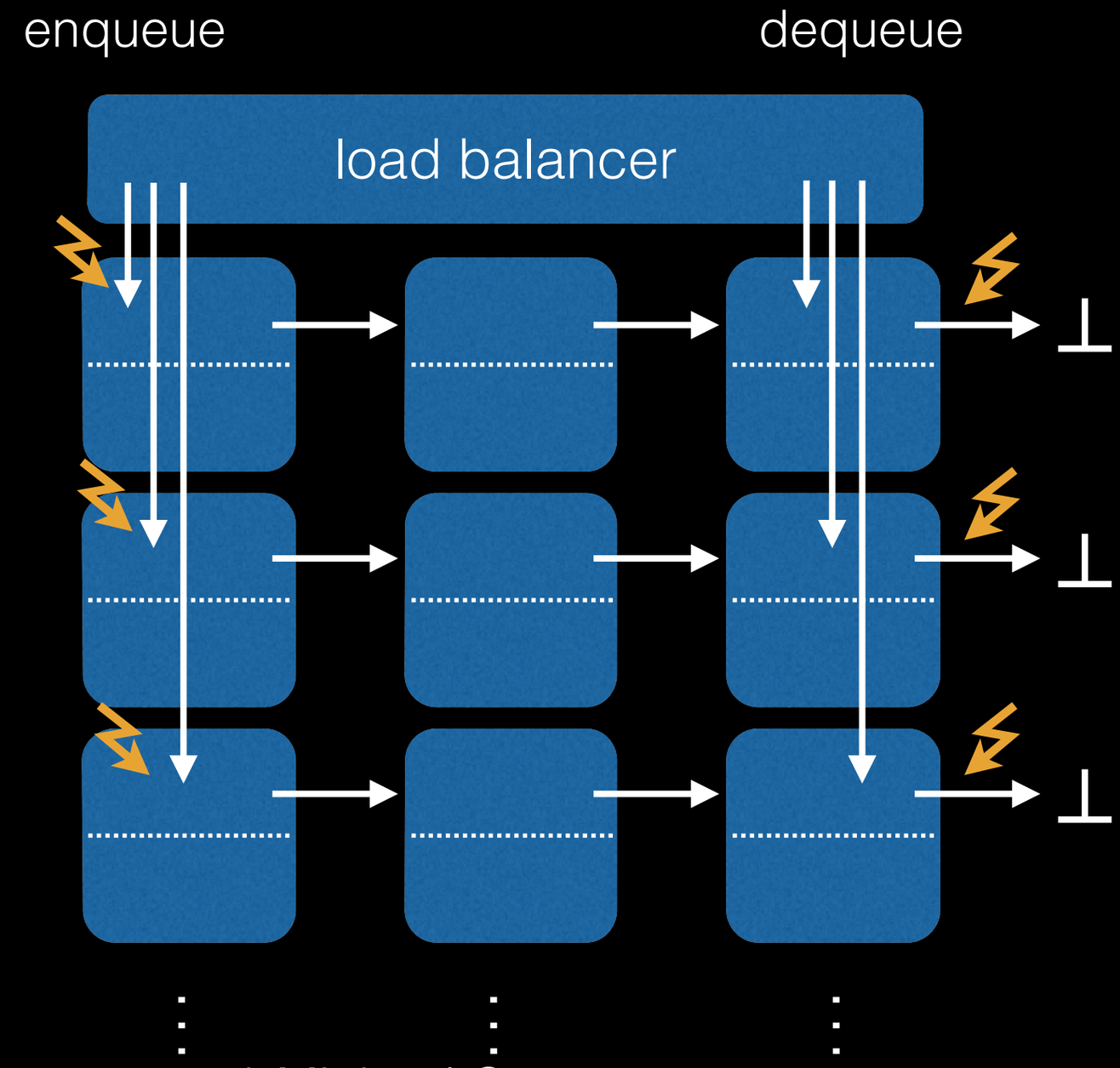
Example: Distributed Queues [CF13]

- k Michael-Scott queues
- Load balancer distributes contention (round-robin, random, LRU, …)
- Semantics is determined by k and the load balancer
- Emptiness check still strict

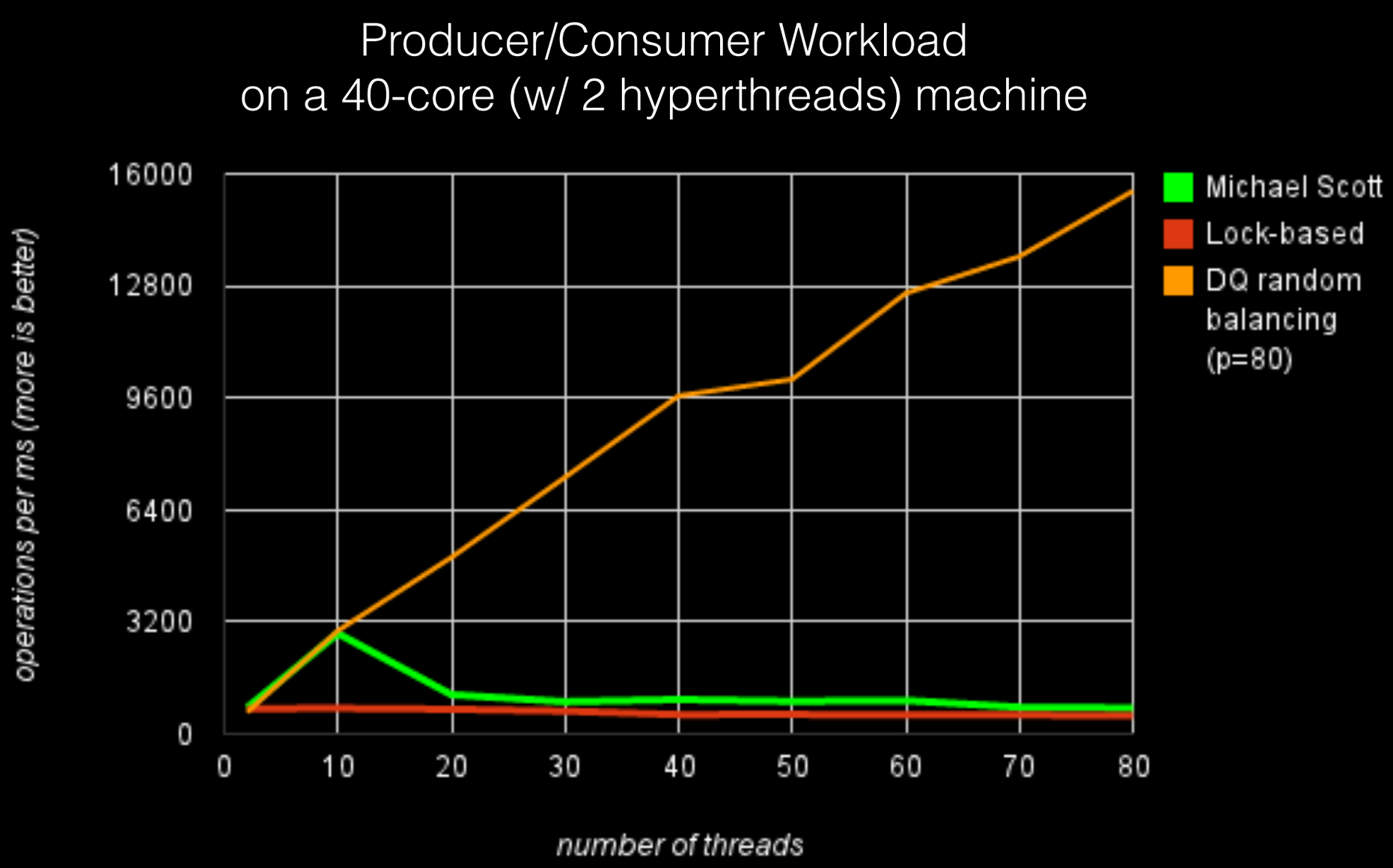


Example: Distributed Queues [CF13]

- k Michael-Scott queues
- Load balancer distributes contention (round-robin, random, LRU, …)
- Semantics is determined by k and the load balancer
- Emptiness check still strict



Reality Check #2



Observation

Lock-freedom does not imply scalability.

Fact

Semantics requires synchronization.
[Laws of Order, POPL11]

Claim

Relaxation is *a way* to get scalability.

Ideas for Relaxed TM

Similar to distributed queues:

- Create copies of memory locations
- Provide *efficient* load balancing between copies
- “Local” transactions on a single copy

In other words:

Consistency within a copy (read/write) and relaxed global semantics.

Example: Relaxed Stack via Relaxed TM

- *k* copies of top pointers
  (albeit logically just one)

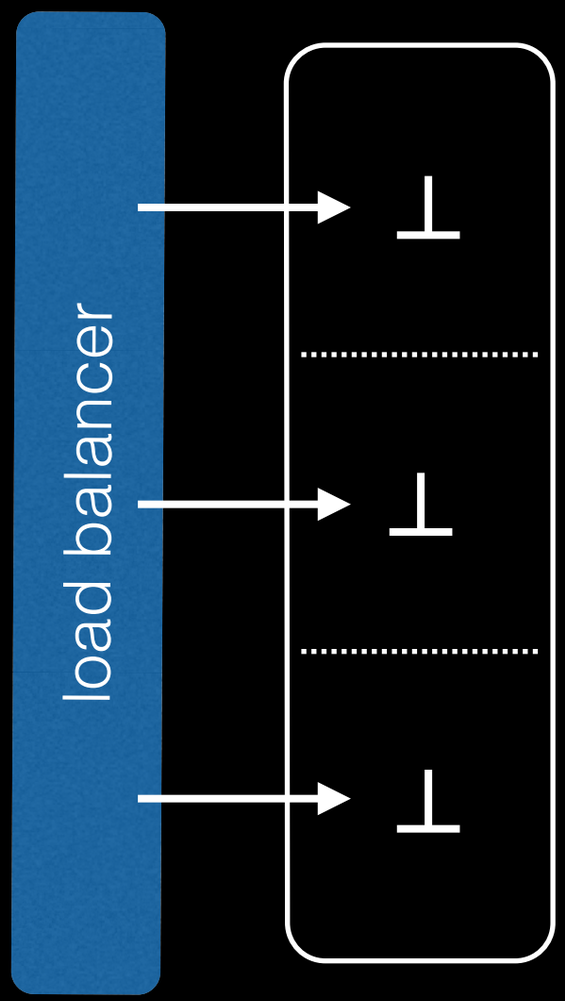
- Initially all copies of top are `null`

- Load balancing (example): Round-robin

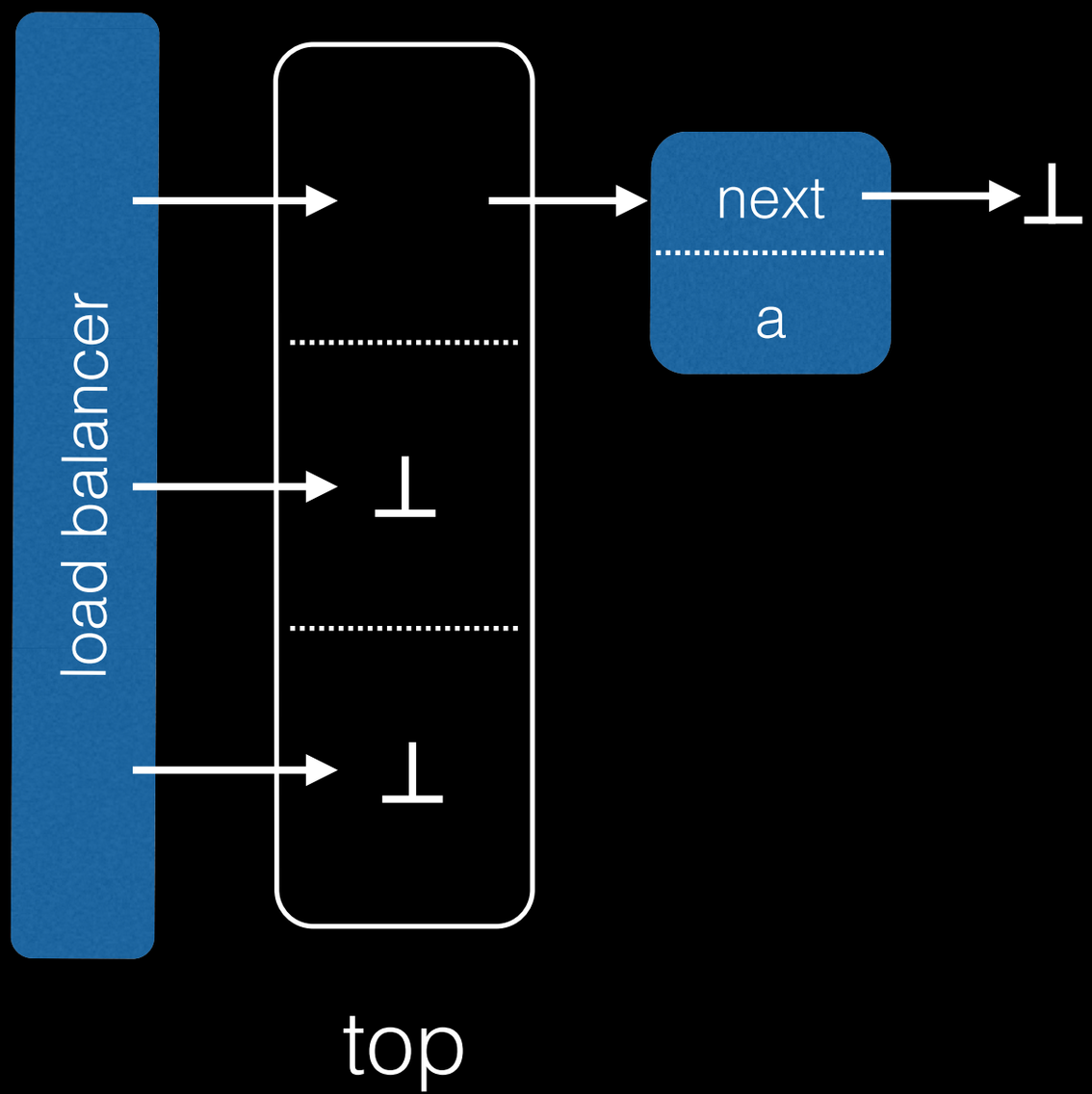
```
record Stack {
   Node top
}

record Node {
   Node next
   Data data
}
```

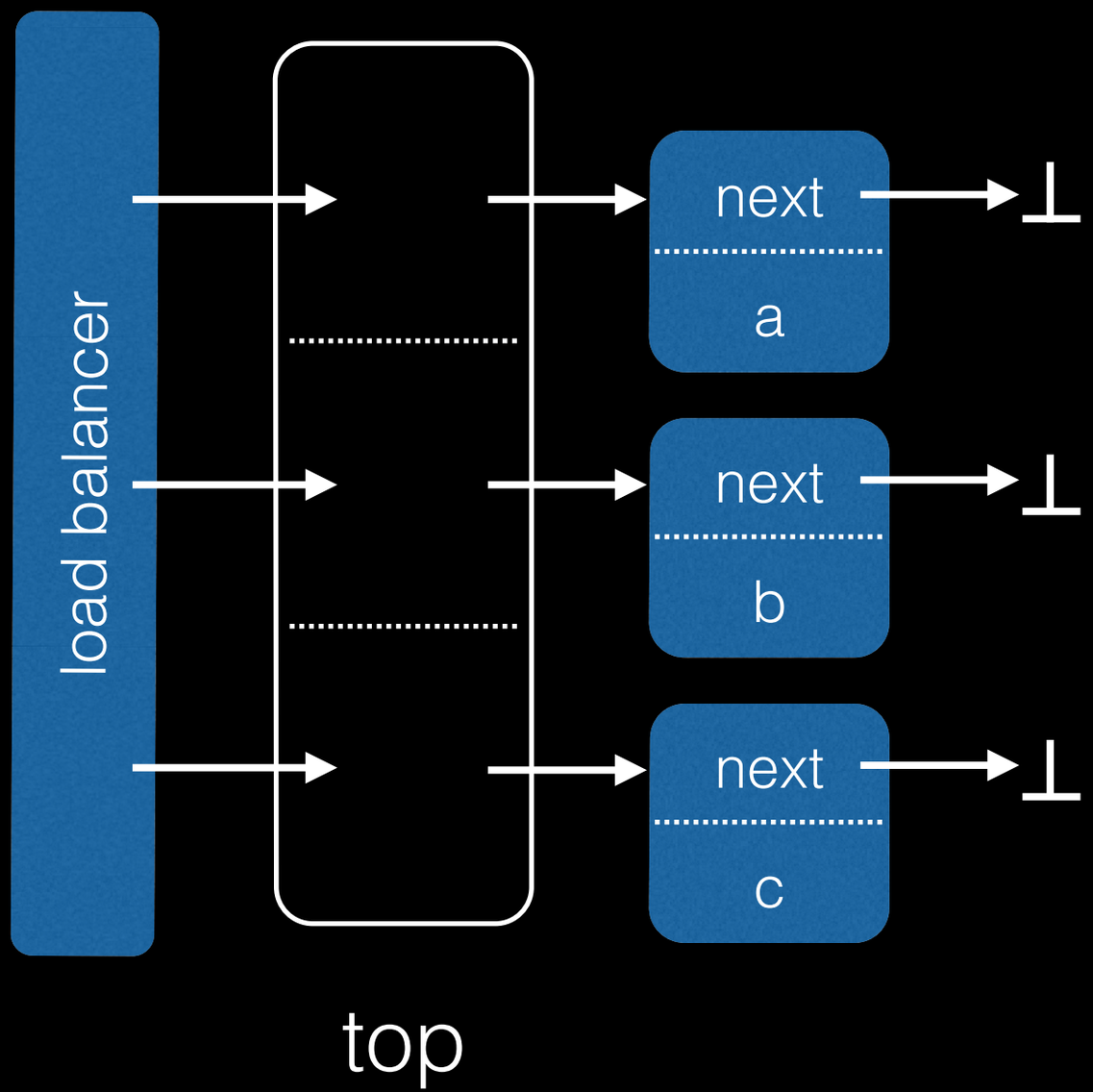
Example:
Relaxed Stack via Relaxed TM



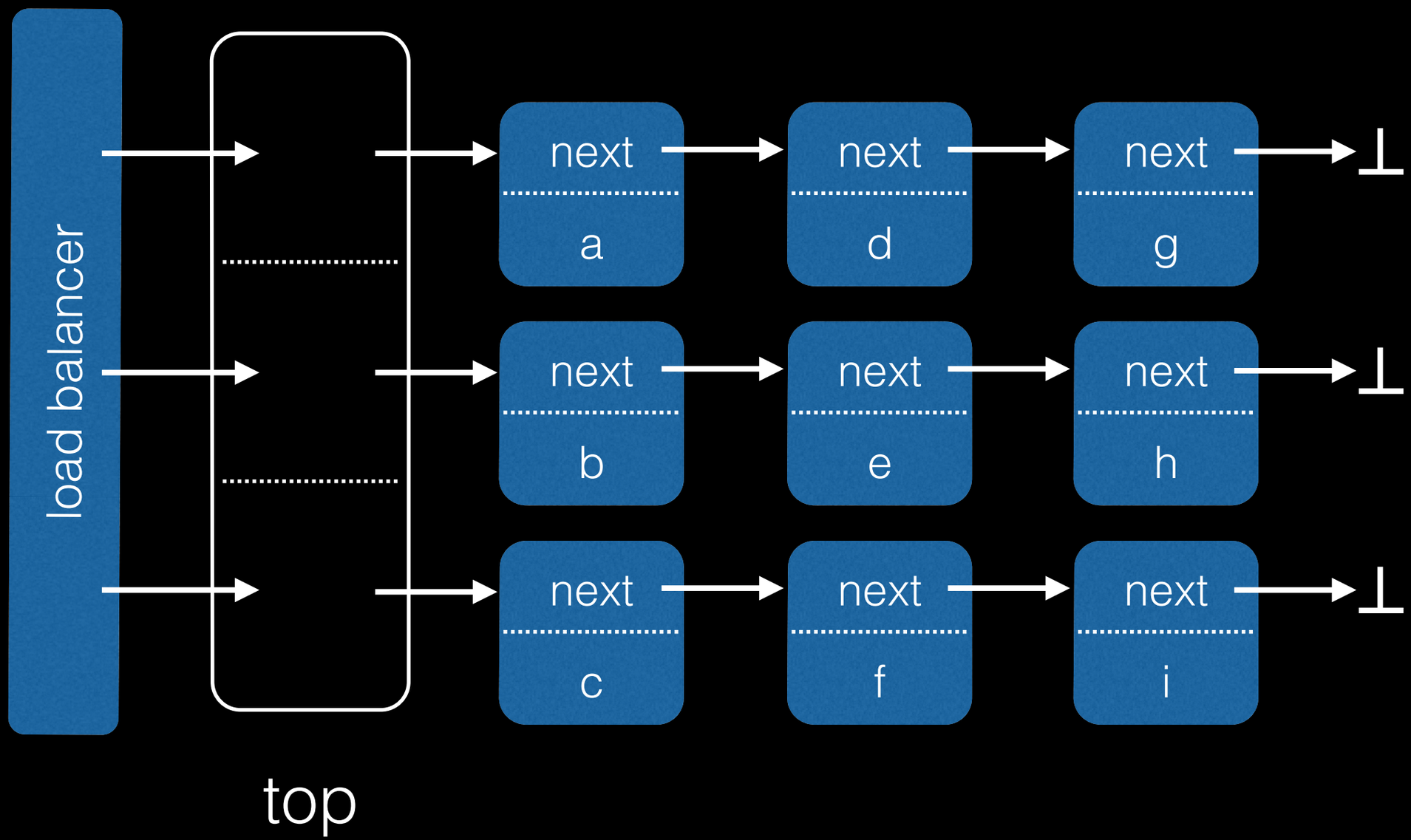
Example:
Relaxed Stack via Relaxed TM



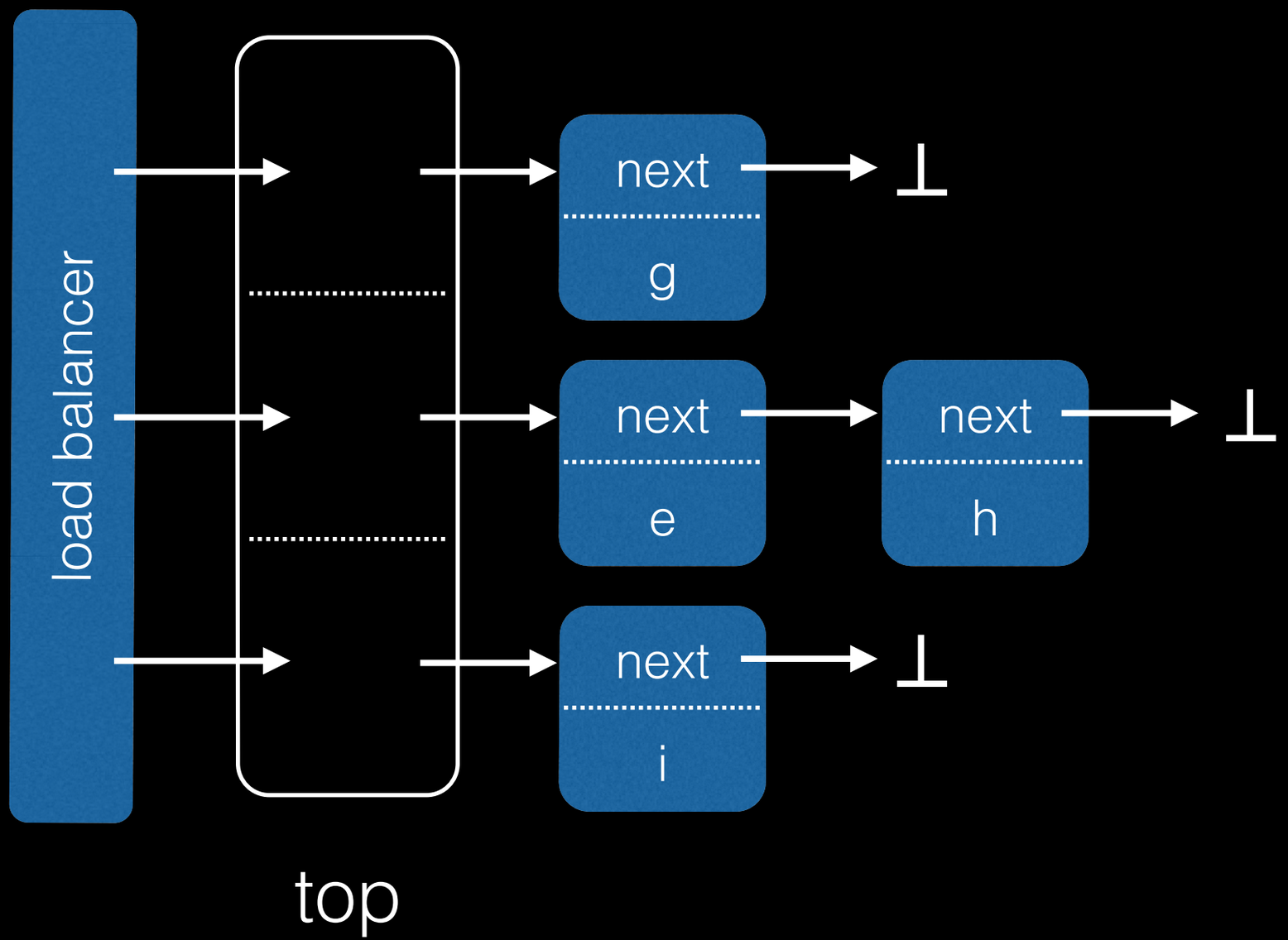
Example:
Relaxed Stack via Relaxed TM



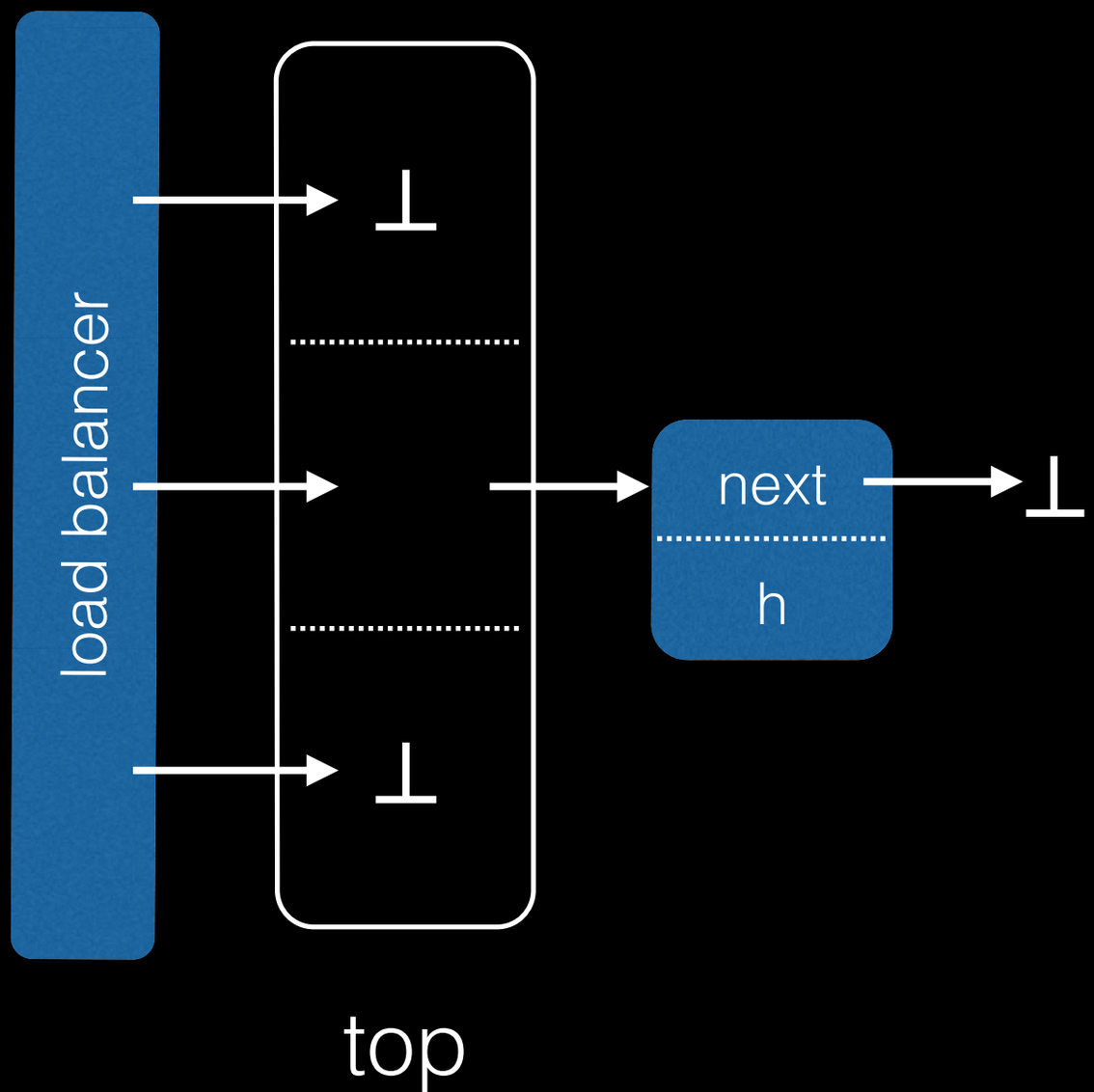
Example:
Relaxed Stack via Relaxed TM



Example:
Relaxed Stack via Relaxed TM

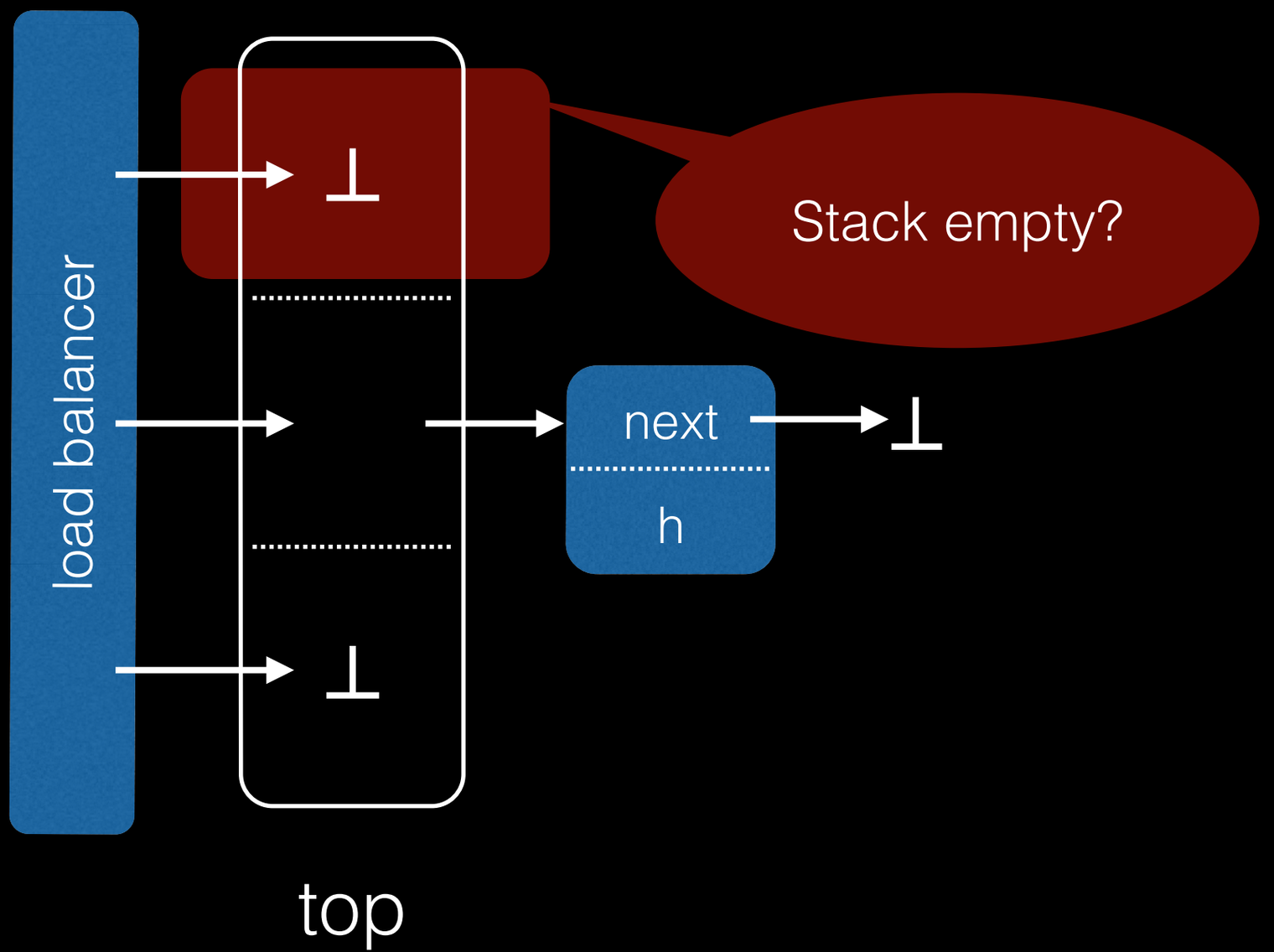


Example:
Relaxed Stack via Relaxed TM



top

Example:
Relaxed Stack via Relaxed TM



Combining Relaxed and Strict Operations

- Relaxed operations might not provide enough semantics

- Analogy: Strict emptiness check with distributed queues

- TM: Support for global read/write transactions

Potential Benefits

- Performance
- Scalability

Drawbacks

- Breaking abstraction (user needs to provide input)
- (Allowing relaxed behavior)

Thank you!

[Laws of Order, POPL11]

H. Attiya, R. Guerraoui, D. Hendler, P. Kuznetsov, M.M. Michael, and M. Vechev. Laws of order: Expensive synchronization in concurrent algorithms cannot be eliminated. In *Proc. Symposium on Principles of Programming Languages (POPL)*, pages 487–498. ACM, 2011.

[MS, PODC96]

M.M. Michael, and M.L. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In Proc. Symposium on Principles on Distributed Computing (PODC), pages 267-275. ACM, 1996.

[POPL13]

T.A. Henzinger, C.M. Kirsch, H. Payer, A. Sezgin, and A. Sokolova. Quantitative relaxation of concurrent data structures. In *Proc. Principles of Programming Languages (POPL)*. ACM, 2013.

[CF13]

A. Haas, T.A. Henzinger, C.M. Kirsch, M. Lippautz, H. Payer, A. Sezgin, and A. Sokolova. Distributed queues in shared memory: Multicore performance and scalability through quantitative relaxation. In *Proc. International Conference on Computing Frontiers (CF)*. ACM, 2013.

[PaCT13]

C.M. Kirsch, M. Lippautz, and H. Payer. Fast and scalable, lock-free k-fifo queues. In *Proc. International Conference on Parallel Computing Technologies (PaCT)*, LNCS, pages 208-223. Springer, 2013.

[SBG14]

A. Haas, M. Dodds, and C.M. Kirsch. *Fast Concurrent Data-Structures Through Explicit Timestamping*. In Technical report TR-2014-02, Department of Computer Sciences, University of Salzburg, 2014.