METIS Spring School, Agadir, Morocco, May 2015

Design of Concurrent and Distributed Data Structures

Christoph Kirsch
University of Salzburg

Joint work with M. Dodds, A. Haas, T.A. Henzinger, A. Holzer, M. Lippautz, H. Payer, A. Szegin, A. Sokolova, and H. Veith.

Concurrent Data Structures

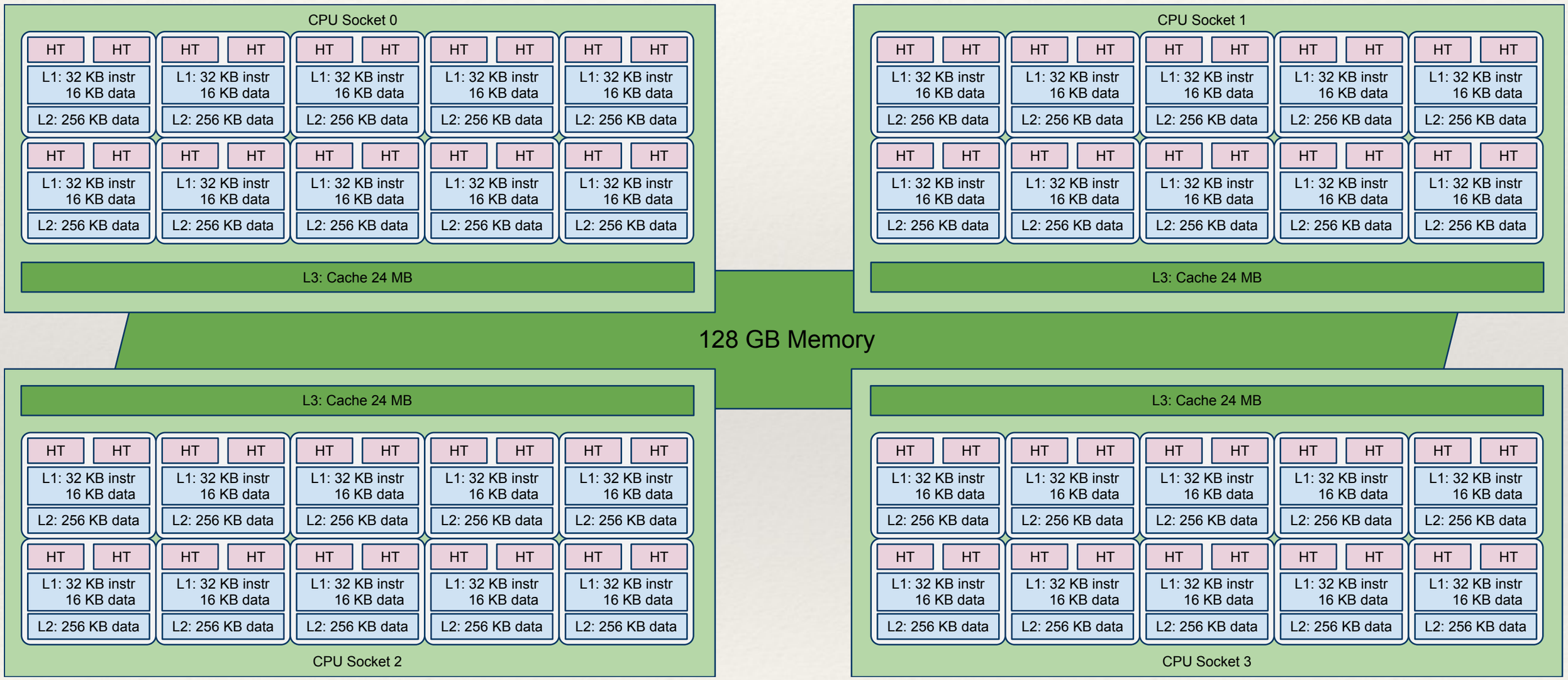
- We are interested in designing and implementing concurrent data structures that are fast and scale on multicore hardware.

- A concurrent data structure is typically shared by a dynamic number of threads that operate on the data structure concurrently and even in parallel in shared memory.

Properties

- Semantics:
  - Sequential specification (queues, stacks, pools, …)
  - Consistency condition (linearizability, quiescent consistency, …)
- Progress:
  - Deadlock (locks)
  - Liveness (lock freedom, wait freedom)
- Performance:
  - Throughput (data structure operations per second)
  - Scalability (throughput in the number of threads, cores, processors)

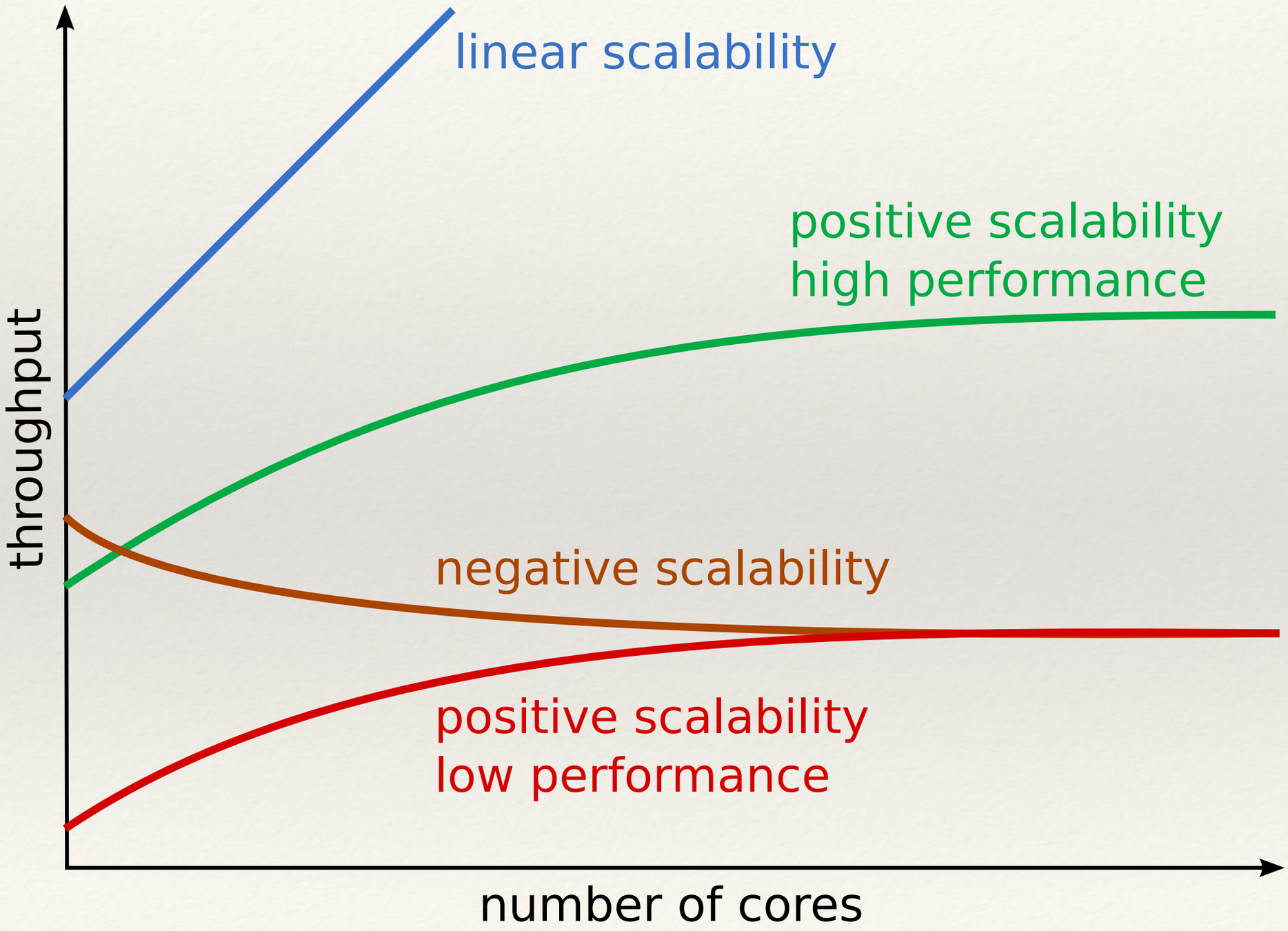
4 Processors w/ 10 Cores each w/ 2 Hyperthreads each



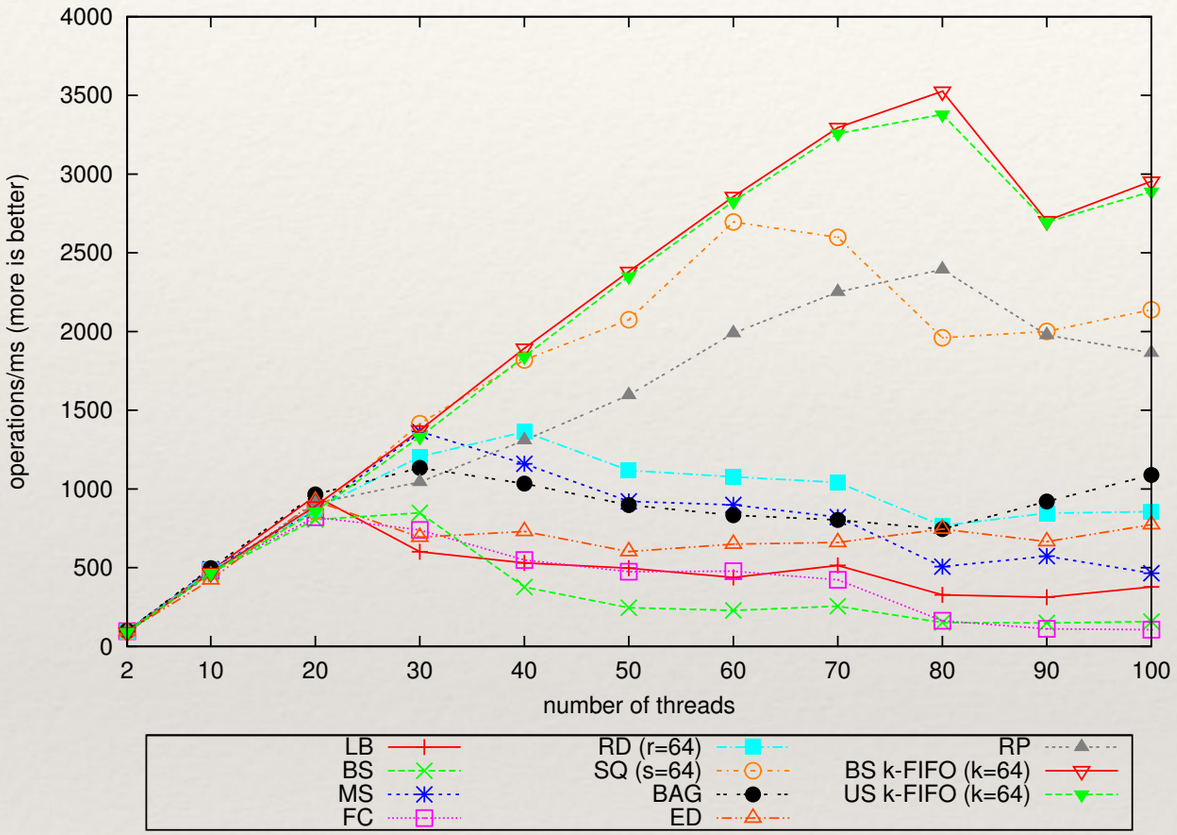
Shared Memory

- Address space versus memory content
- Memory access latency: uniform and non-uniform
- Temporal locality: cache size
- Spatial locality: line size
- False sharing: line size

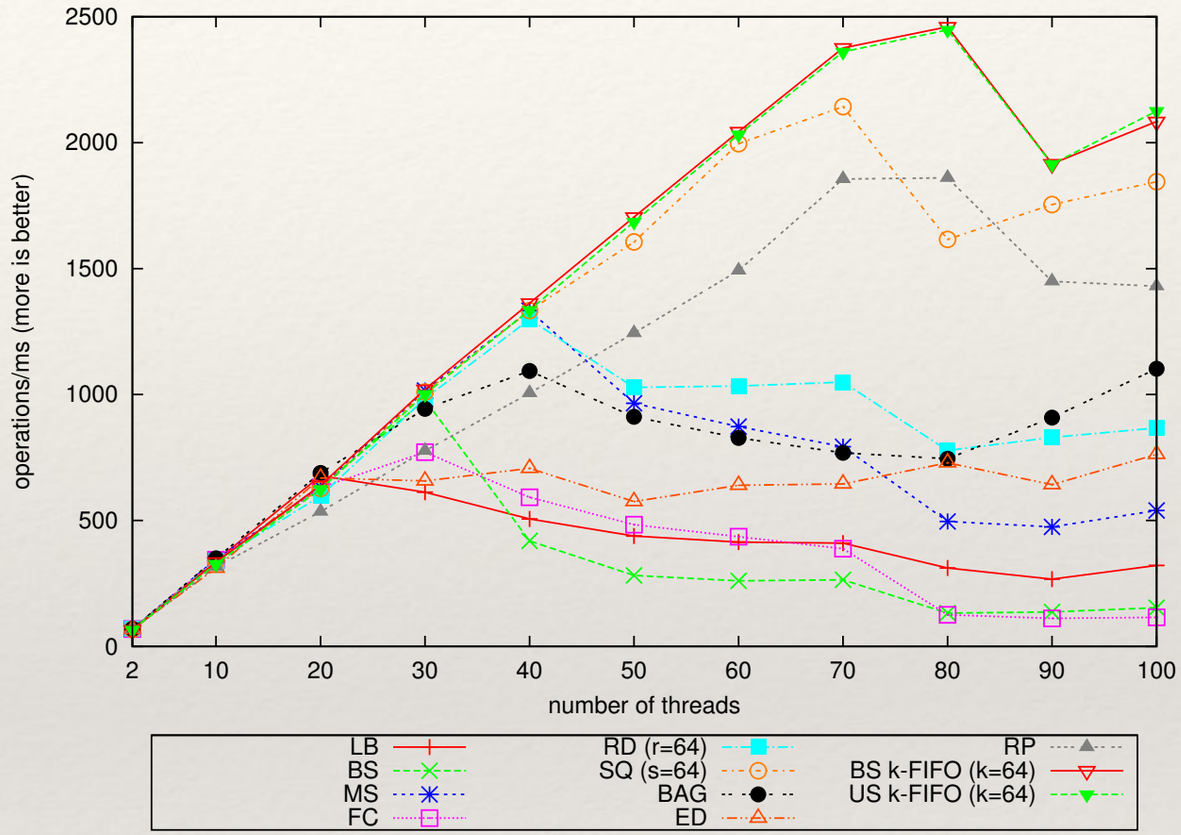
Multicore Scalability



Example



(c) Medium contention ($c = 7000$, $i = 0$)



(d) Low contention ($c = 10000$, $i = 0$)

Fig. 2. Performance and scalablity of producer/consumer microbenchmarks with an increasing number of threads

False Sharing

```
int x;
int y;


x = 0;
y = 0;


if (fork() != 0)
  while (x < 1000000) x = x + 1; // parent
else
  while (y < 1000000) y = y + 1; // child
```

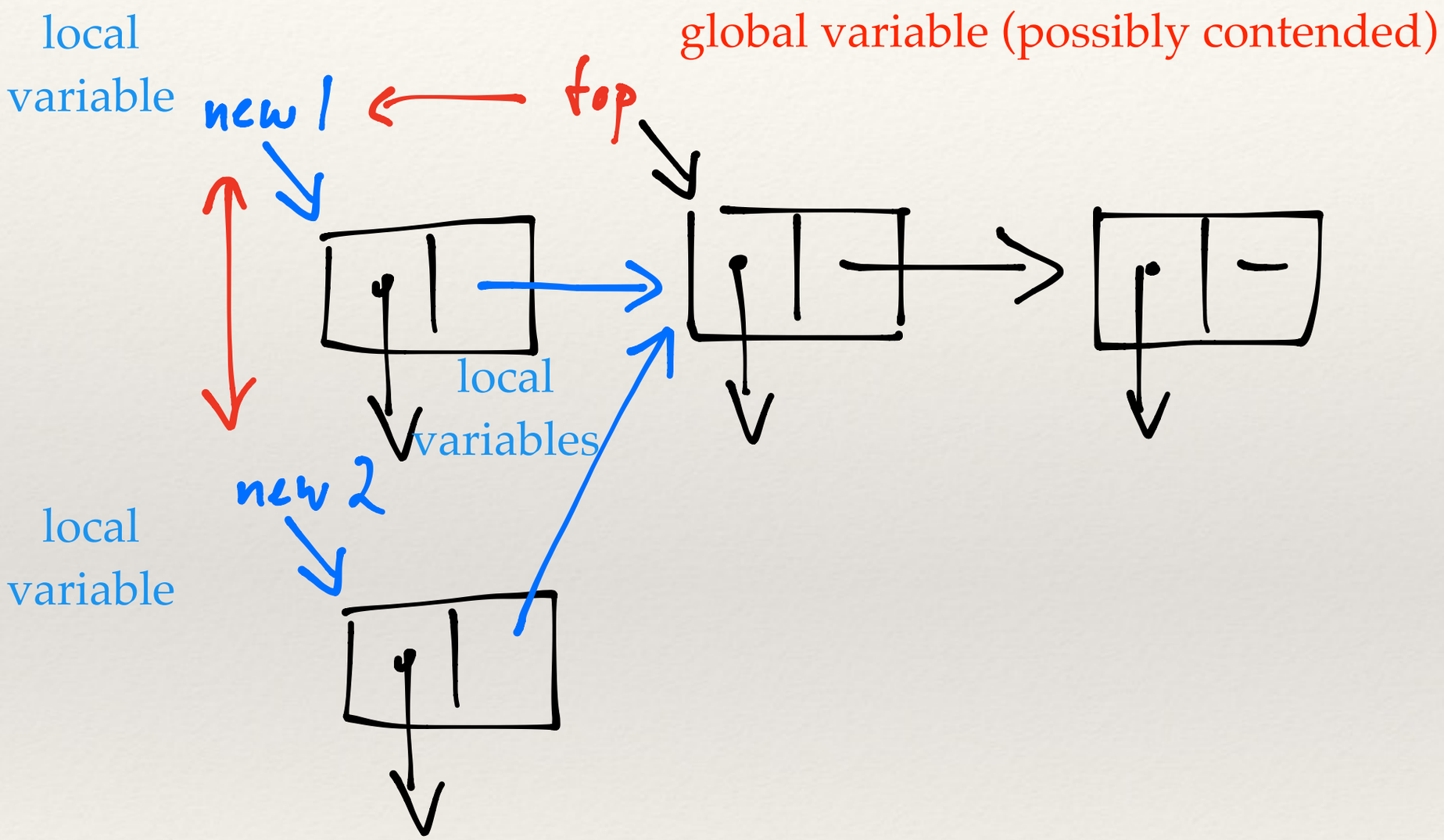
A Concurrent Variable

```
int x;

x = 14;          ADDI R1,R0,14

                 STW R1,R28,-4

fork();          BSR 0,0,fork:

x = x + 14;      LDW R1,R28,-4      LDW R1,R28,-4

                 ADDI R1,R1,14      ADDI R1,R1,14

                 STW R1,R28,-4      STW R1,R28,-4


x?
```

How about a lock?

How about atomic fetch and add?

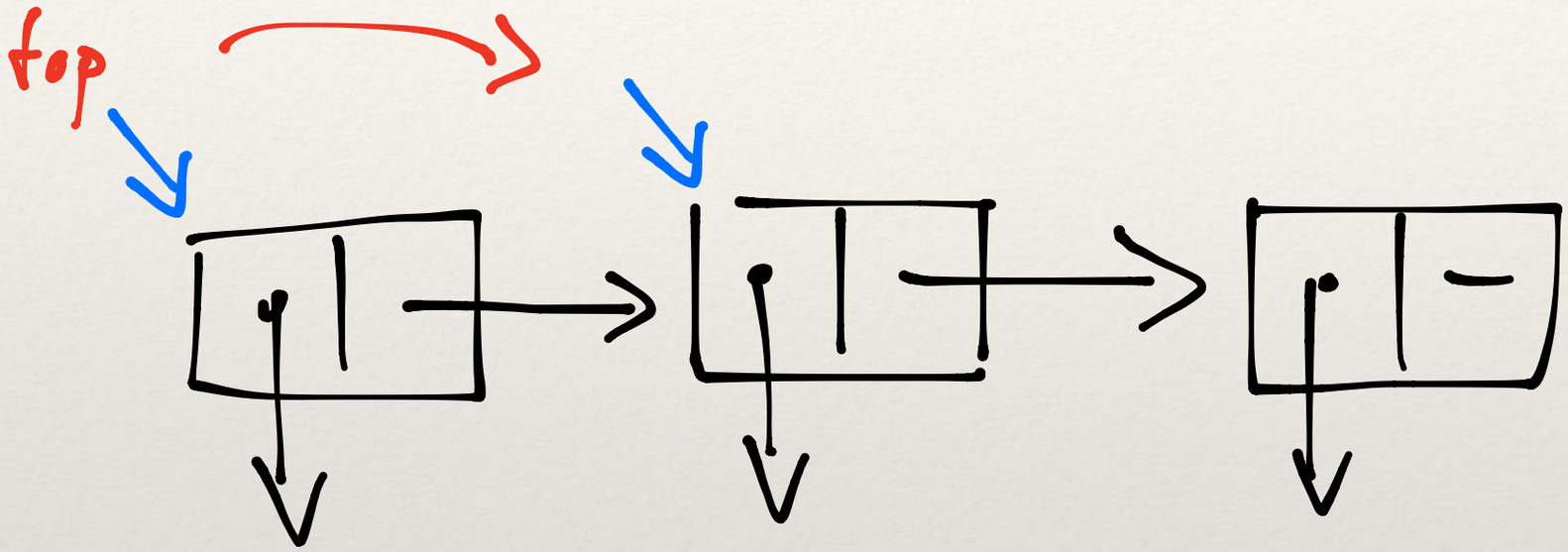
A Concurrent Stack: Push



How about a lock?

How about atomic compare and swap?

A Concurrent Stack: Pop



What about memory reuse?

ABA Problem

- stack: top -> A -> B -> C
- thread 1: attempts to pop A but is suspended before setting top to B yet remembering B (and A) anyway
- thread 2: pops A, pops B, pushes A again
- stack: top -> A -> C
- thread 1: resumes and pops A but sets top to B
- stack: top -> B (-> C)

ABA Solutions

- Principled problem: finitely many names (memory addresses) for describing infinite state space

- Pointer tagging: efficient but makes ABA only unlikely
- Garbage collection: can be used to solve ABA but requires attention
- Hazard pointers: solve ABA but takes time and space

Strict vs. Relaxed Semantics

- Strict semantics:
  - Locks versus no locks
  - Lock-freedom versus wait-freedom
  - Time versus atomicity
  - Time versus space
- Relaxed semantics:
  - sequential specification
  - consistency condition

Multicore Scalability is a Concurrent Semantics and Memory Layout & Search Problem

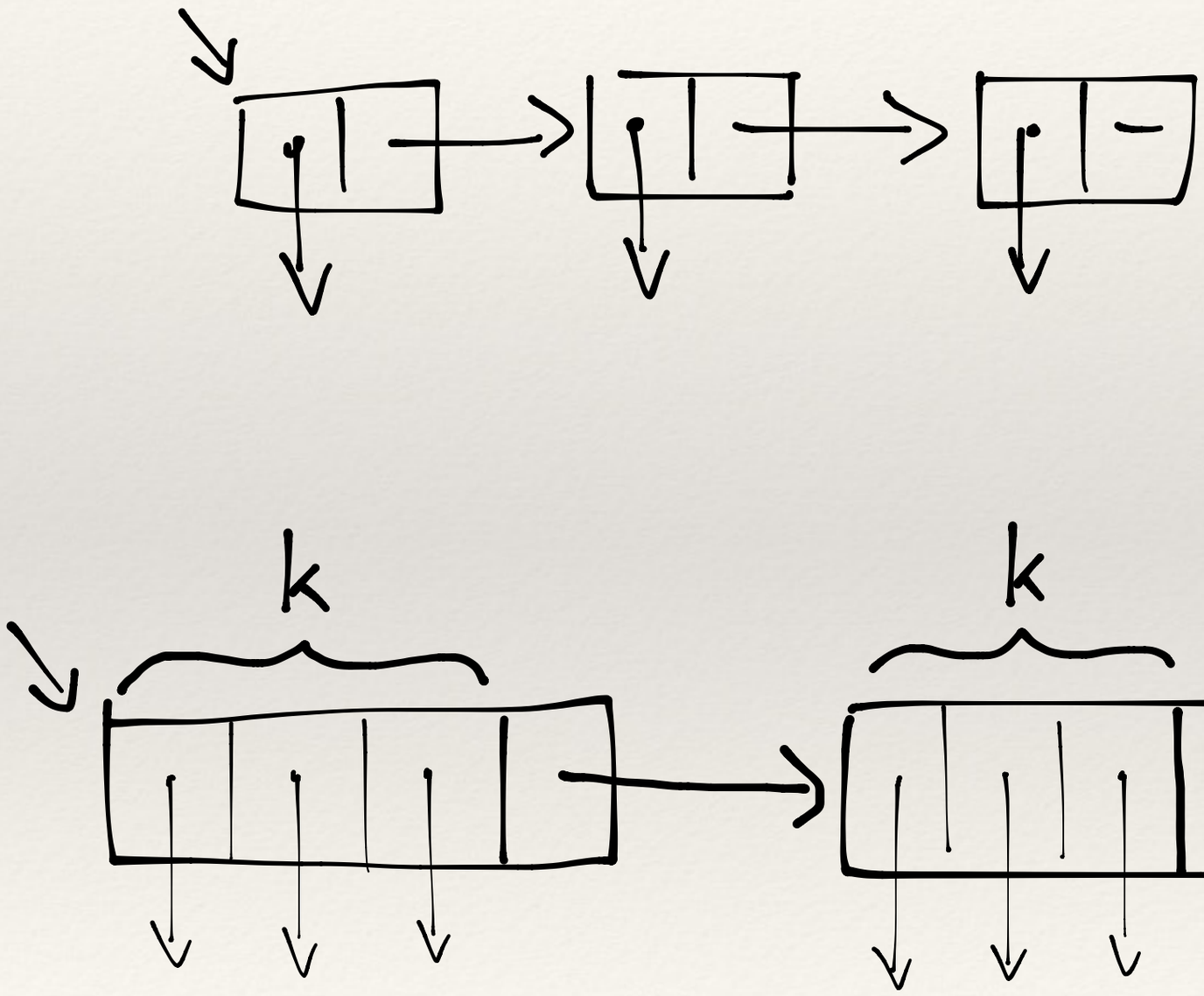
scal.cs.uni-salzburg.at

- Scal is a collection of concurrent data structures designed by us (underlined) and others, plus a benchmarking framework:
- Treiber Stack [IBM86]
  - Timestamped Stack [POPL15], k-Stack (relaxed) [POPL13]
- Michael-Scott Queue [PODC96]
- LCRQ [PPoPP13], Segment Queue (relaxed) [OPODIS10]
  - Timestamped Queue [POPL15], Distributed Queue (relaxed) [CF13], k-FIFO Queue (relaxed) [PaCT13]
  - Timestamped Deque [POPL15]
- …

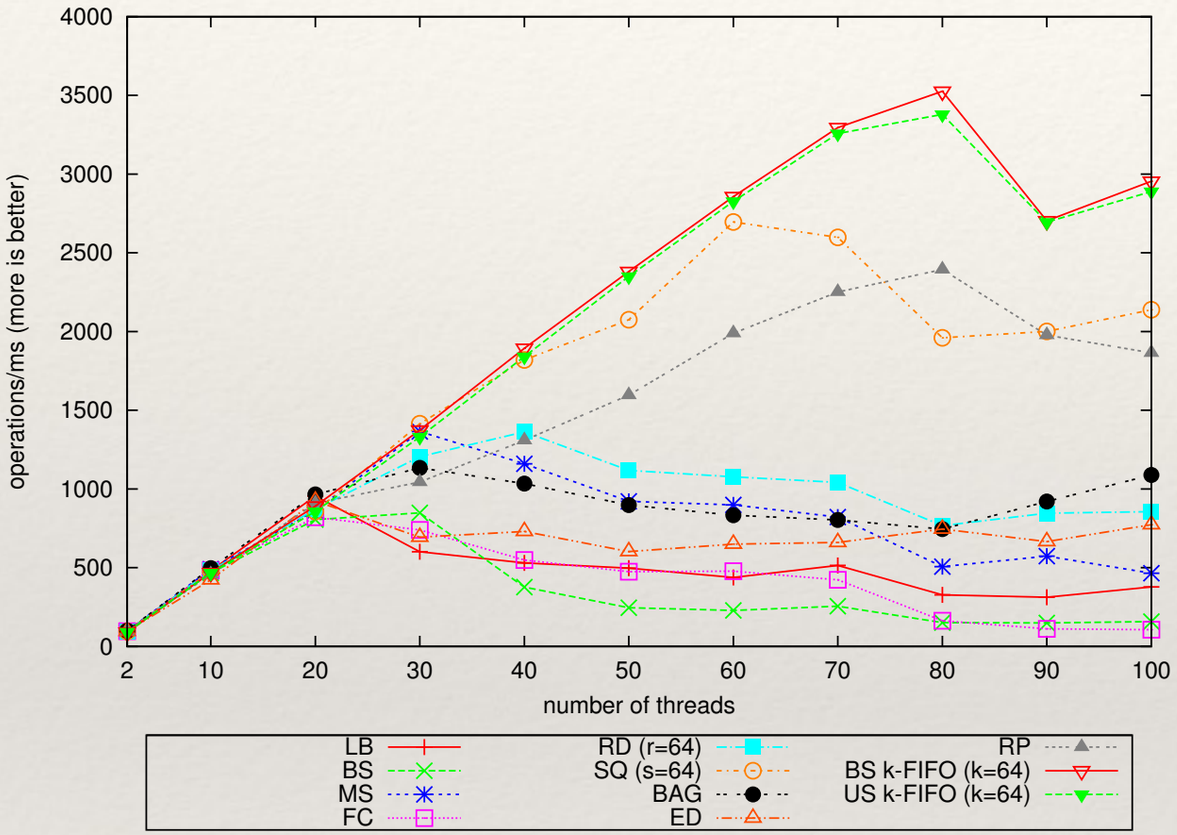
Treiber Stack [IBM86]

Michael-Scott Queue [PODC96]

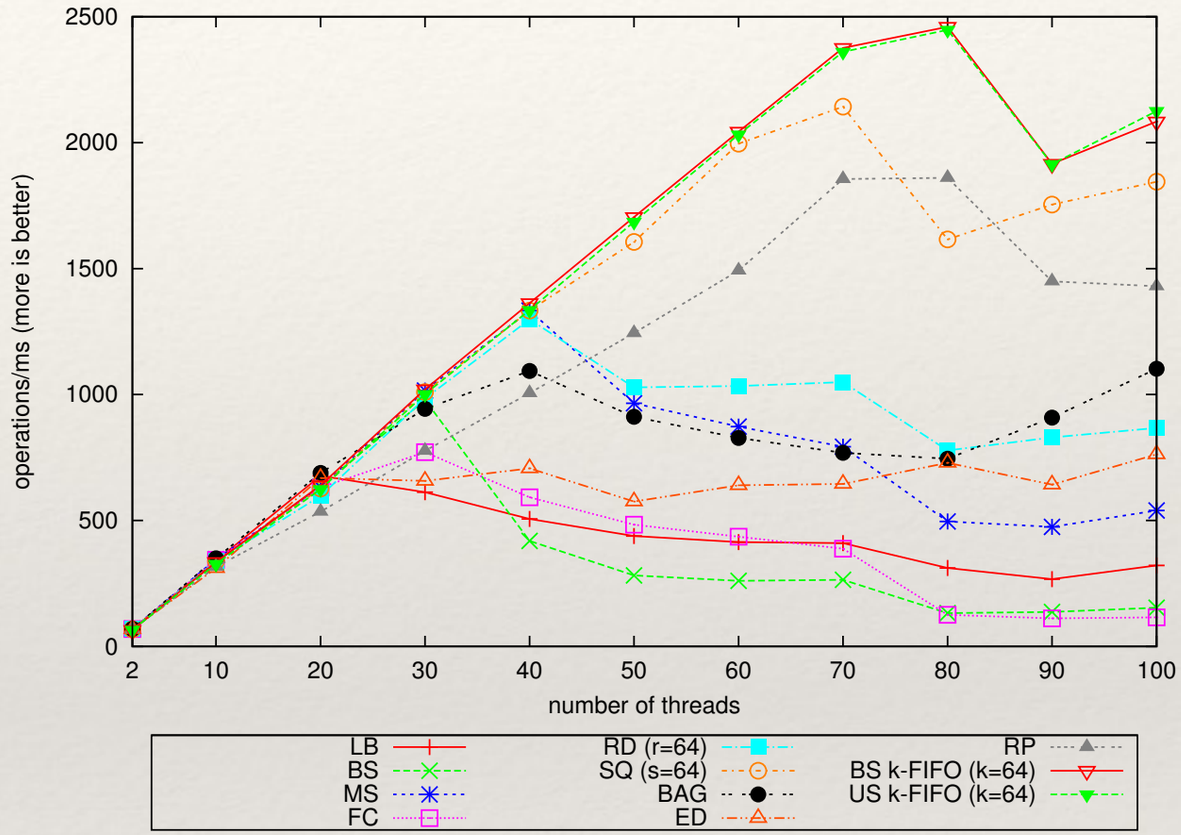
Segmentation



k-FIFO Queue [PaCT13]



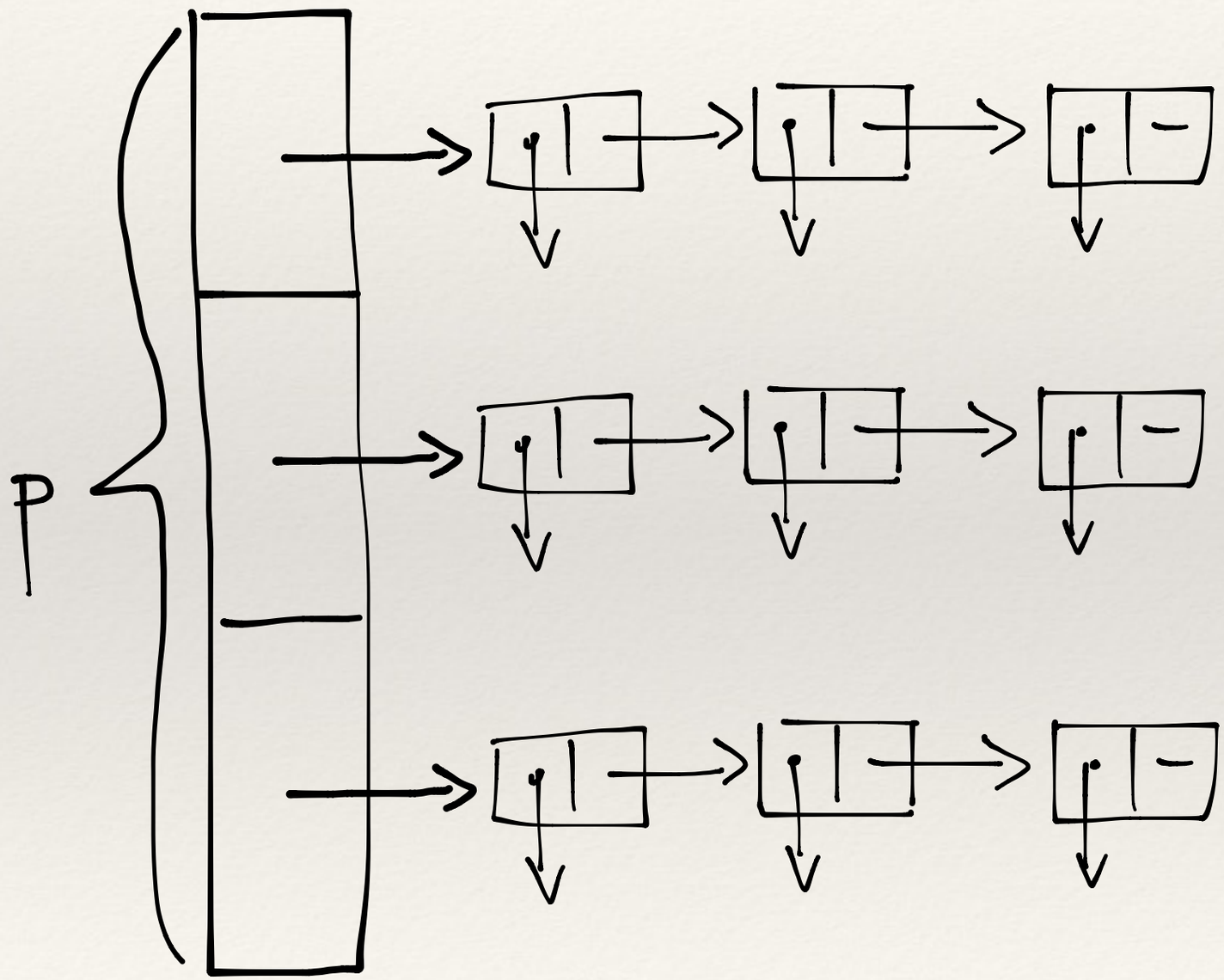
(c) Medium contention ($c = 7000$, $i = 0$)



(d) Low contention ($c = 10000$, $i = 0$)

Fig. 2. Performance and scalablity of producer/consumer microbenchmarks with an increasing number of threads

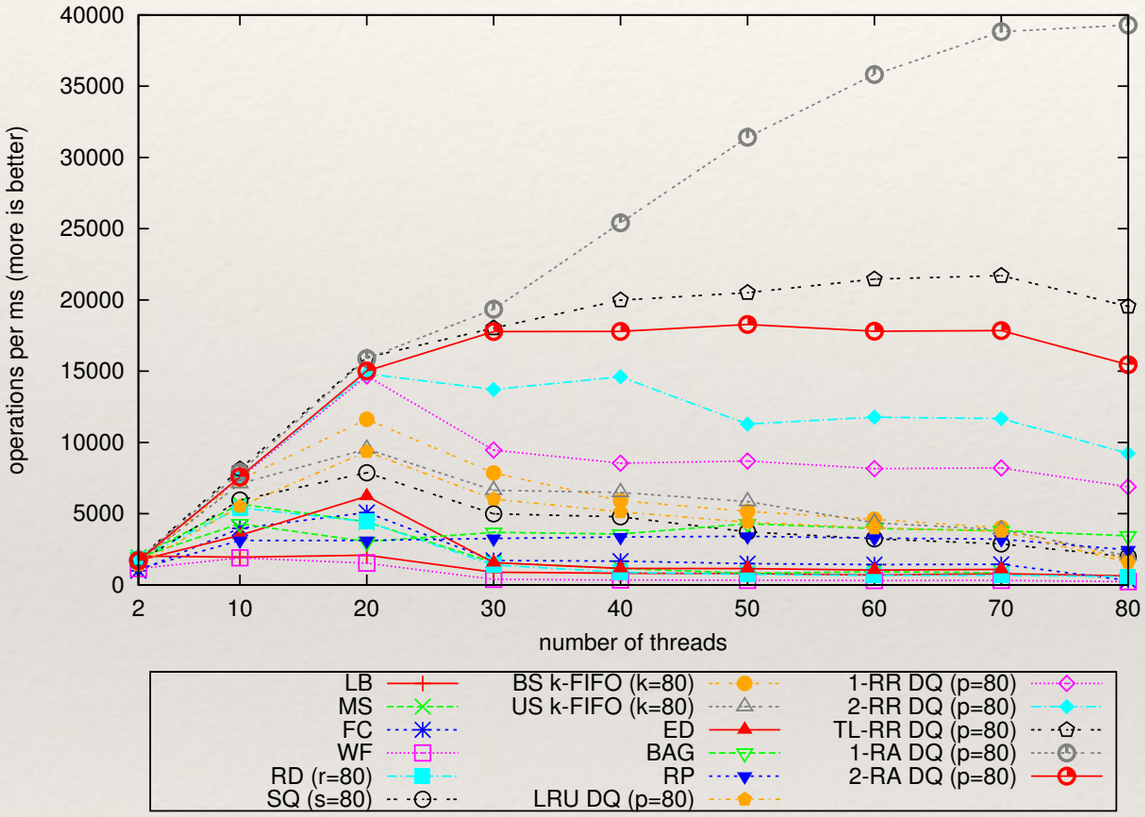
Distribution



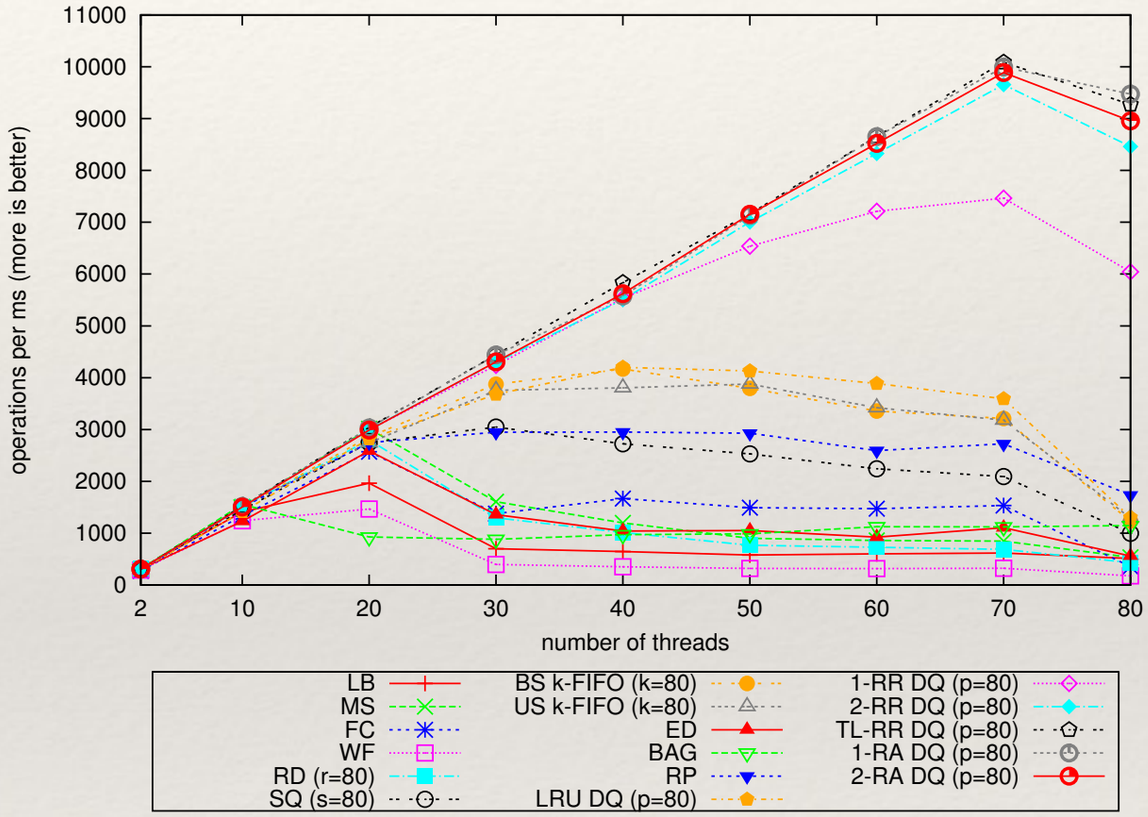
Load Balancers

- 1-RR: one round-robin counter
- 2-RR: two round-robin counters (enqueue, dequeue)
- TL-RR: thread-local round-robin counters
- LRU: least-recently-used queue
- 1-RA: random queue
- 2-RA: shorter of two random queues for enqueue, longer of two random queues for dequeue

Distributed Queues [CF13]



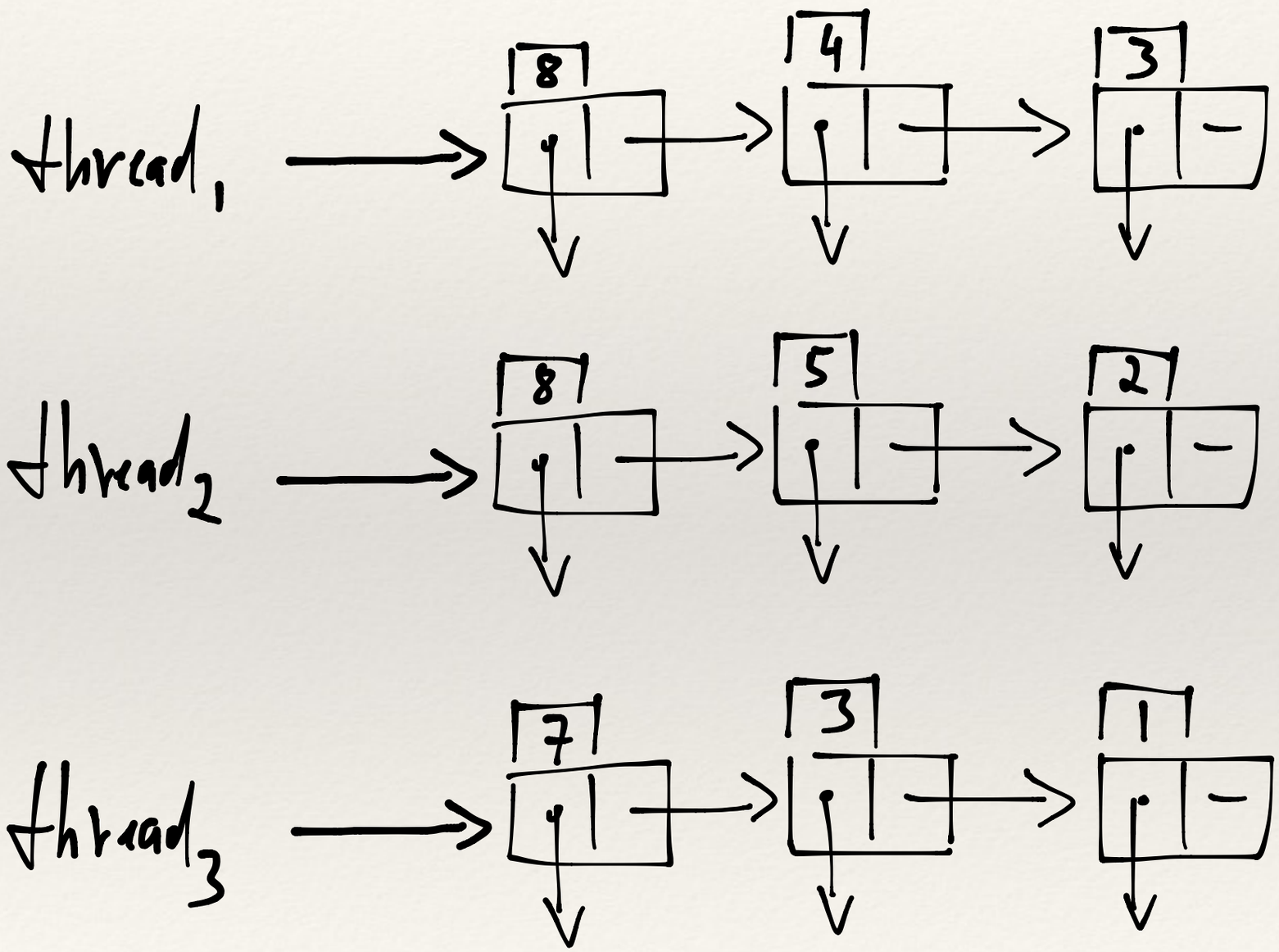
(a) High contention producer-consumer microbenchmark ($c = 250$)



(b) Low contention producer-consumer microbenchmark ($c = 2000$)

Figure 1: Performance and scalability of producer-consumer microbenchmarks with an increasing number of threads on a 40-core (2 hyper-threads per core) server machine

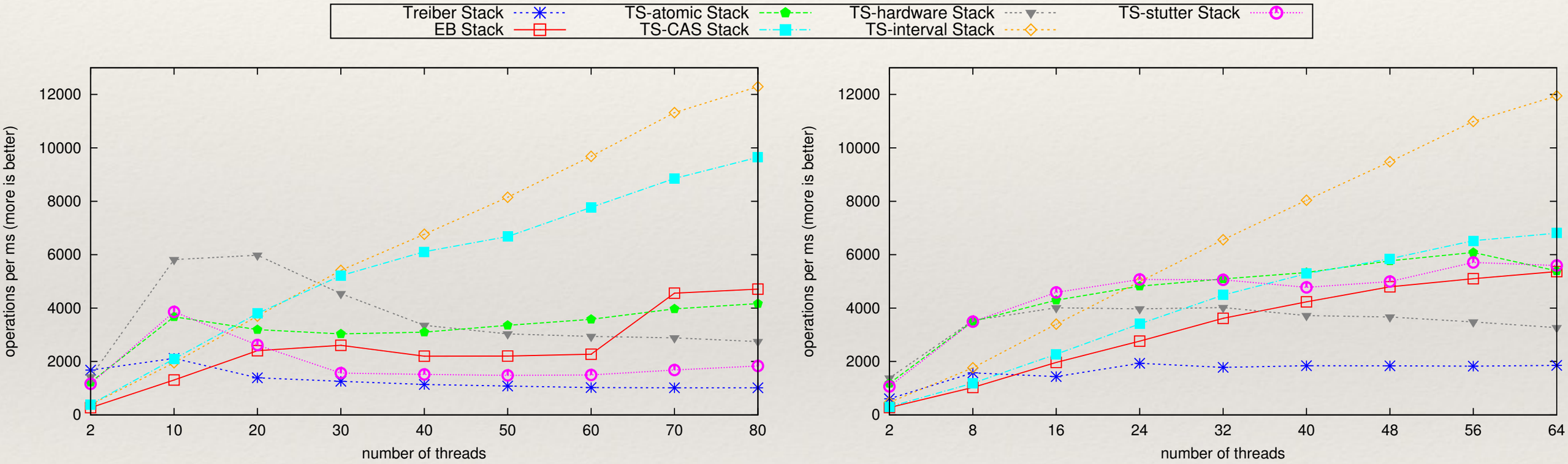
Timestamping



Time Sources

- TS-atomic: fetch and increment counter
- TS- stutter: stuttering counter (Lamport clock)
- TS-hardware: hardware clock
- TS-interval: interval hardware clock
- TS-CAS: compare-and-swap interval counter

Timestamped (TS) Stack [POPL15]



(a) Producer-consumer benchmark, 40-core machine.

(b) Producer-consumer benchmark, 64-core machine.

Timestamping

- TS stack: fastest concurrent stack
- TS queue: slower than LCRQ but faster than MS
- TS deque: fastest deque but correctness proof missing
- …

Thank you