NETYS Conference, Agadir, Morocco, May 2015

Scalloc: From Relaxed Concurrent Data Structures to the Fastest Multicore-Scalable Memory Allocator

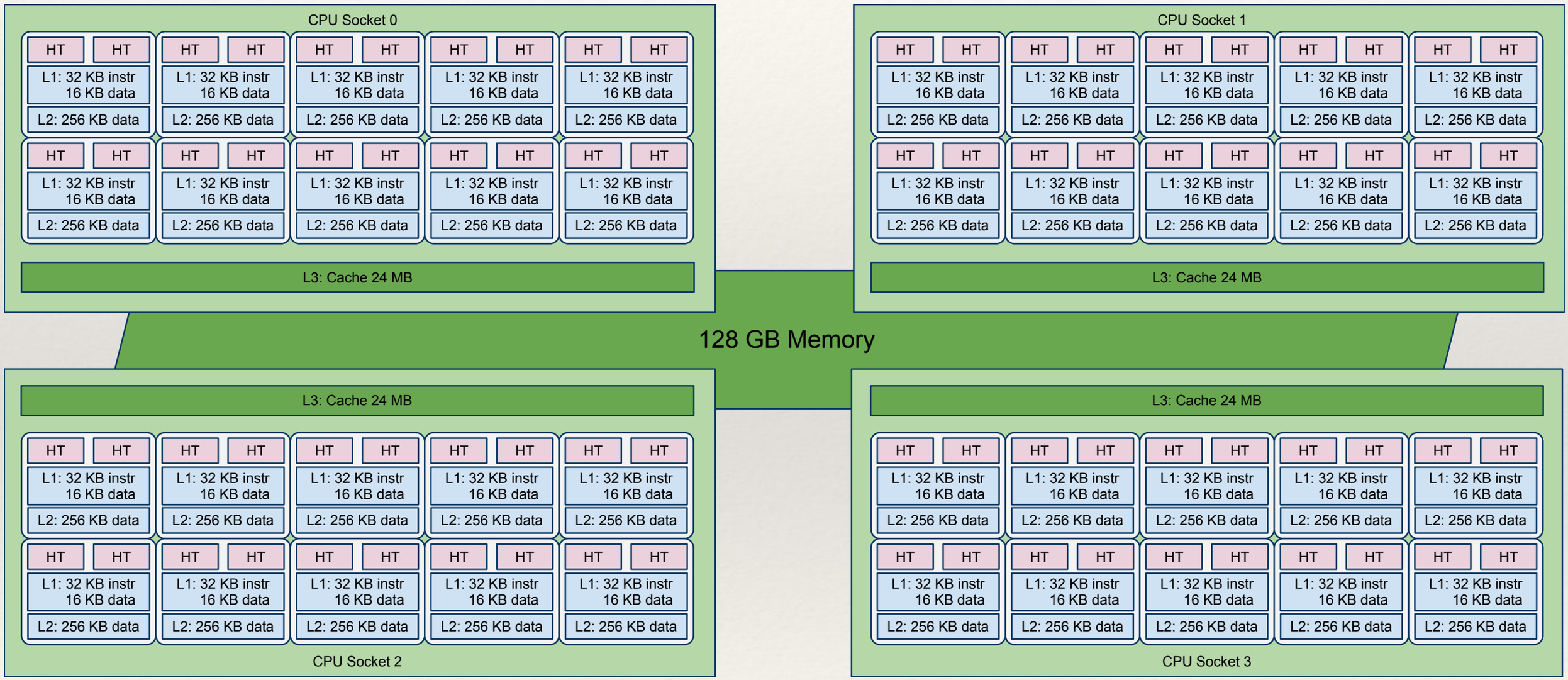
Christoph Kirsch
University of Salzburg

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

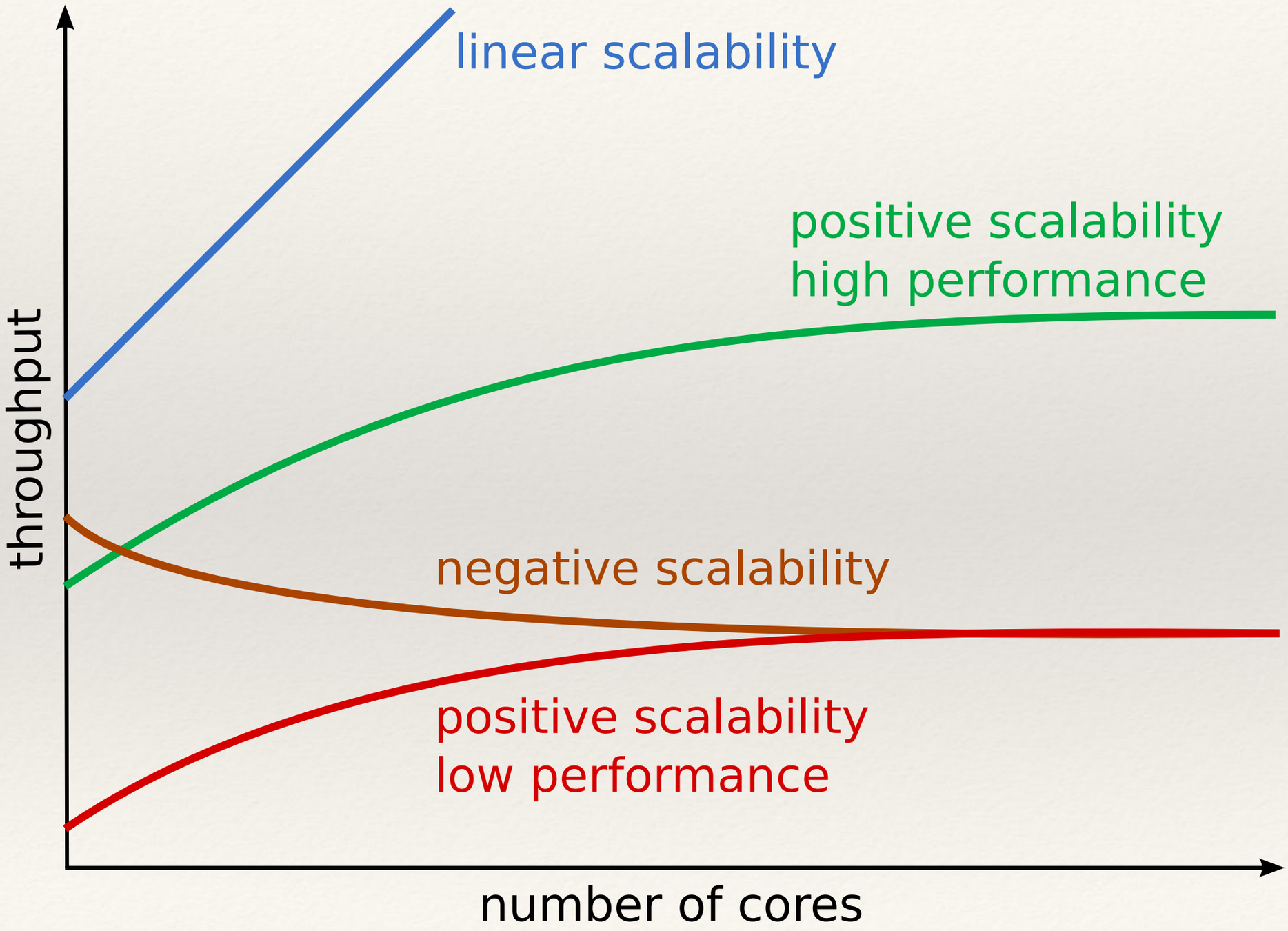
Infrastructural Software

- We are interested in designing and implementing concurrent data structures that are fast and scale on multicore hardware.
- We then apply the best designs in other infrastructural software such as a memory allocator.

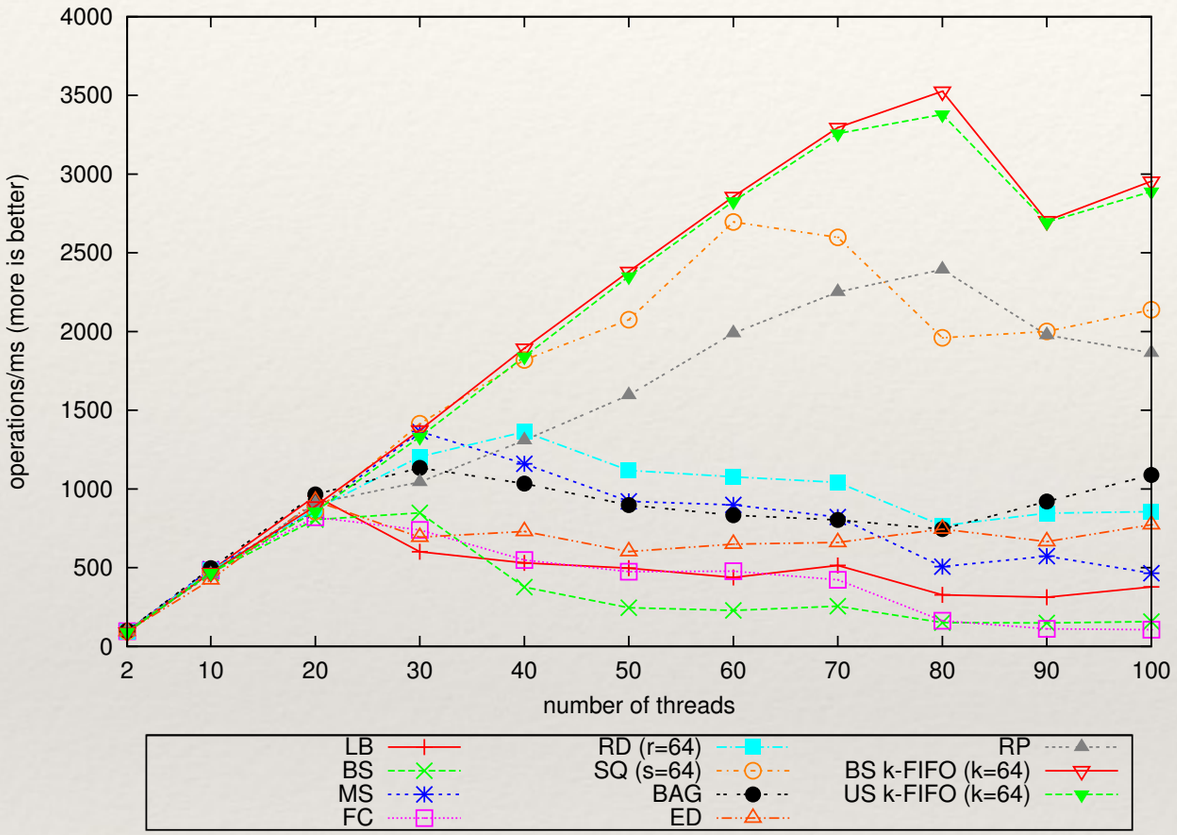
4 Processors w/ 10 Cores each w/ 2 Hyperthreads each



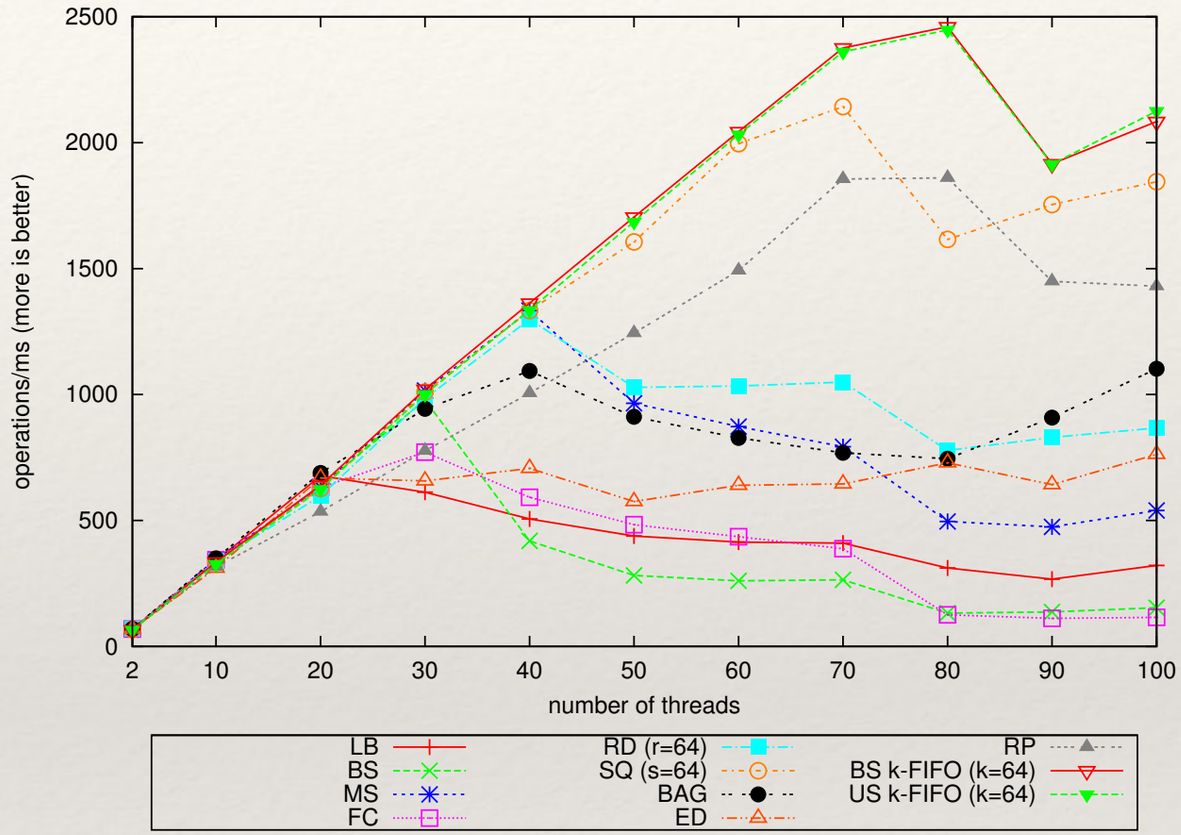
Multicore Scalability



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

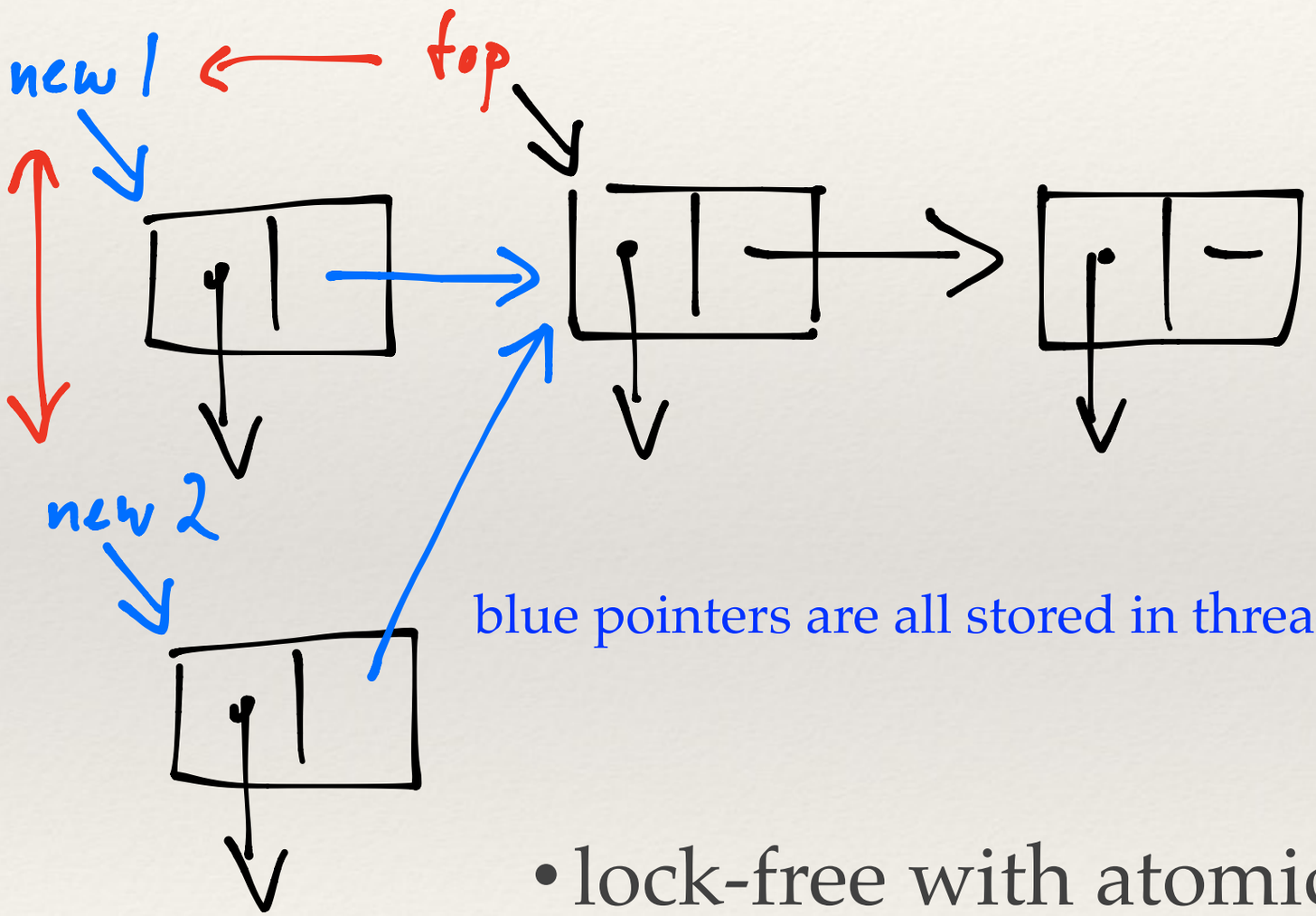
Multicore Scalability is a Concurrent Semantics and Memory Layout & Search Problem

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]

top pointer is global variable (single point of contention)

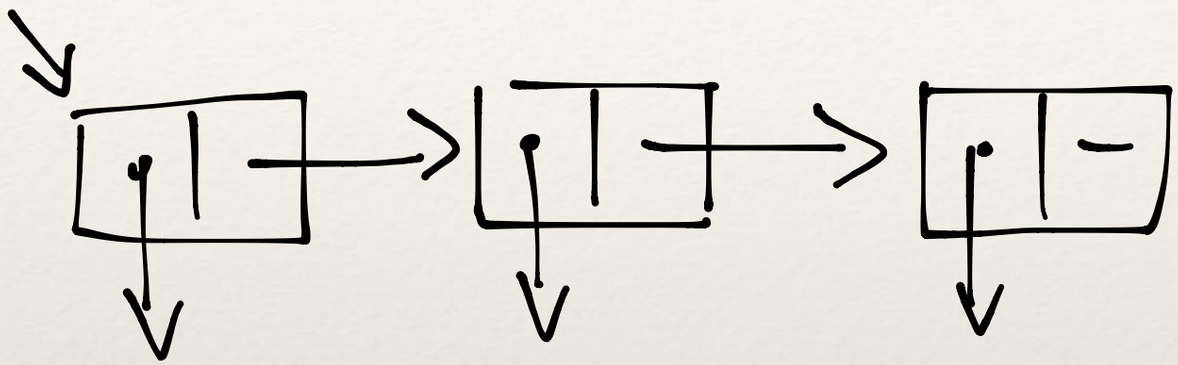


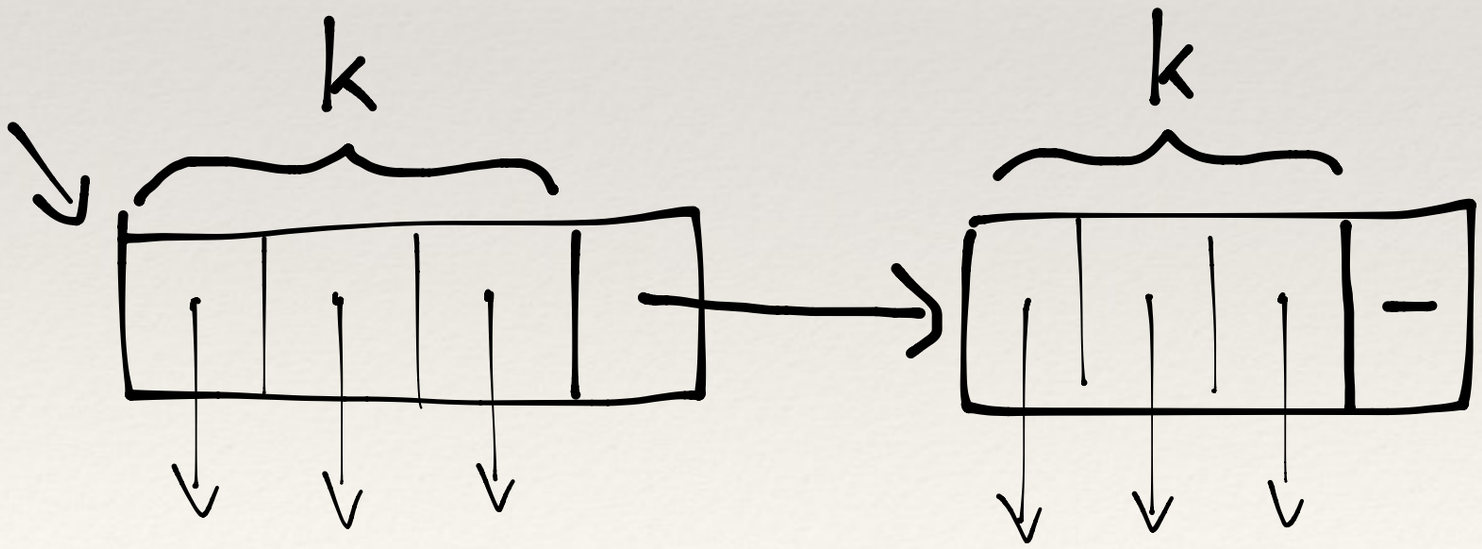
blue pointers are all stored in thread-local variables

- lock-free with atomic compare and swap
- failing threads retry, possibly indefinitely

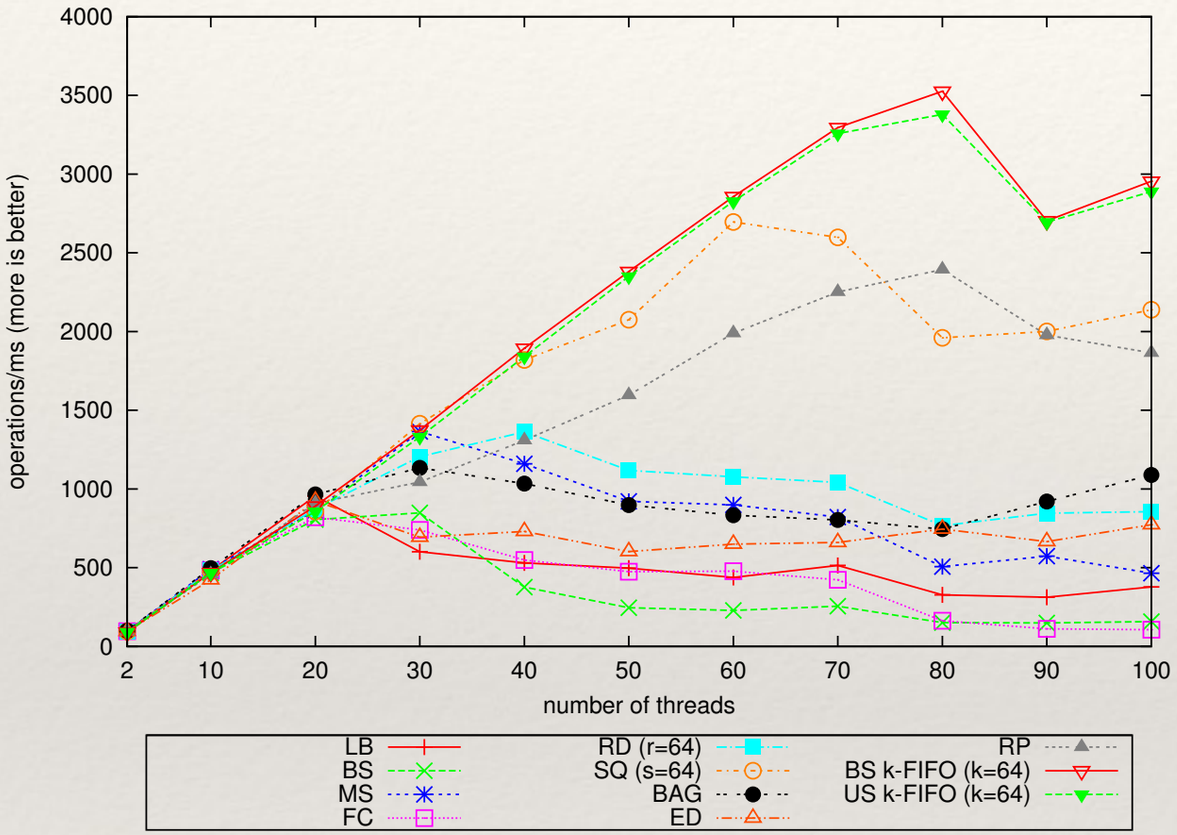
Michael-Scott Queue [PODC96]

Segmentation

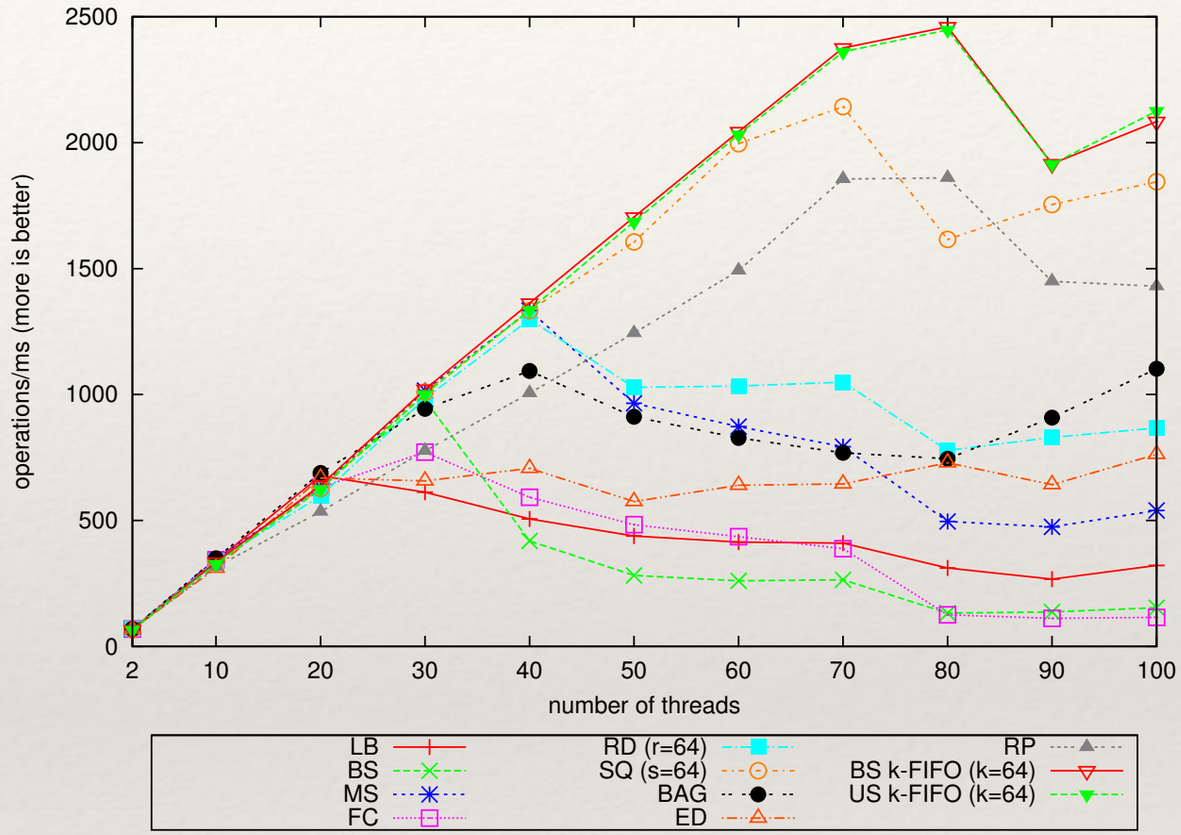




k-FIFO Queue [PaCT13]



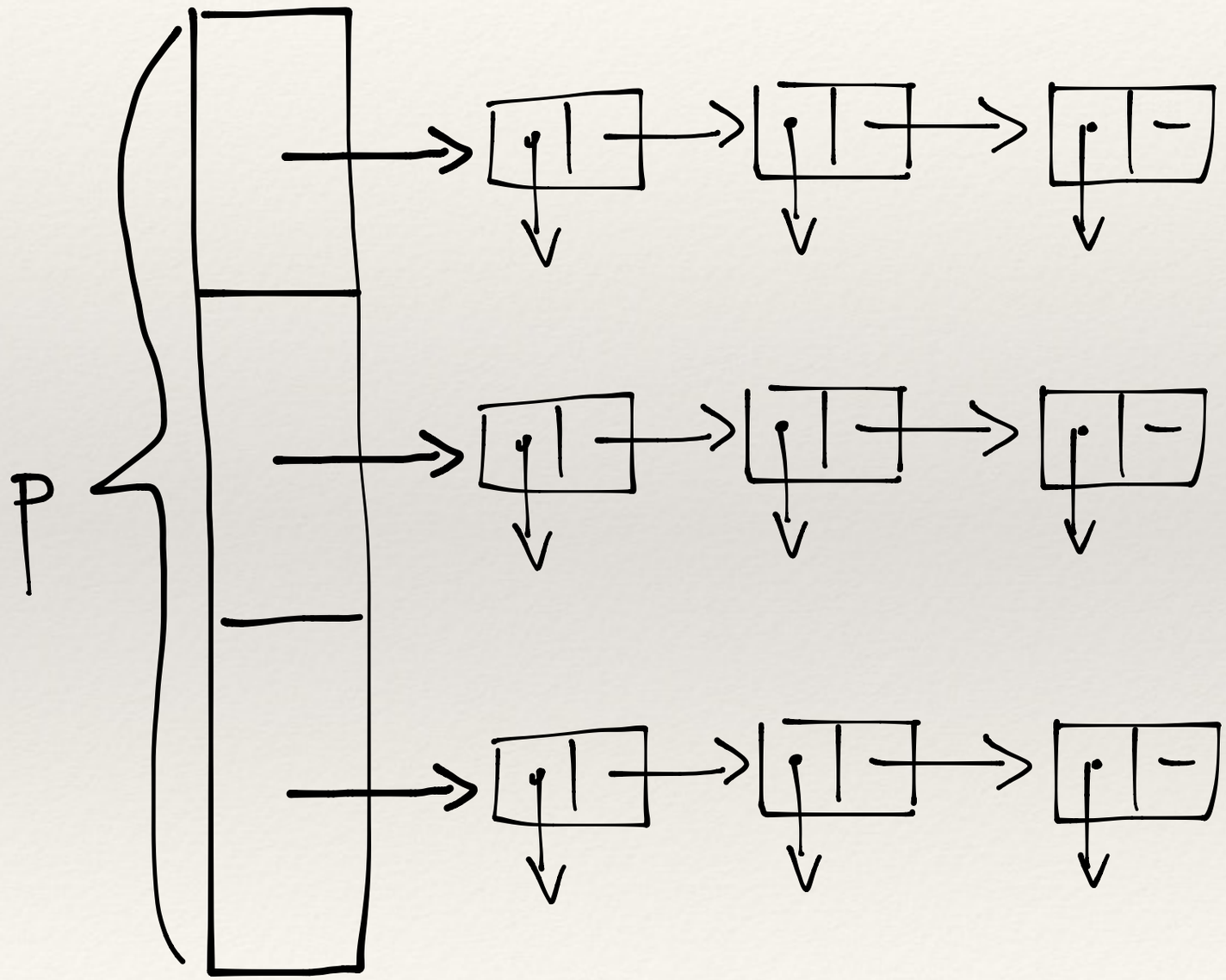
(c) Medium contention ($c = 7000$, $i = 0$)



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

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

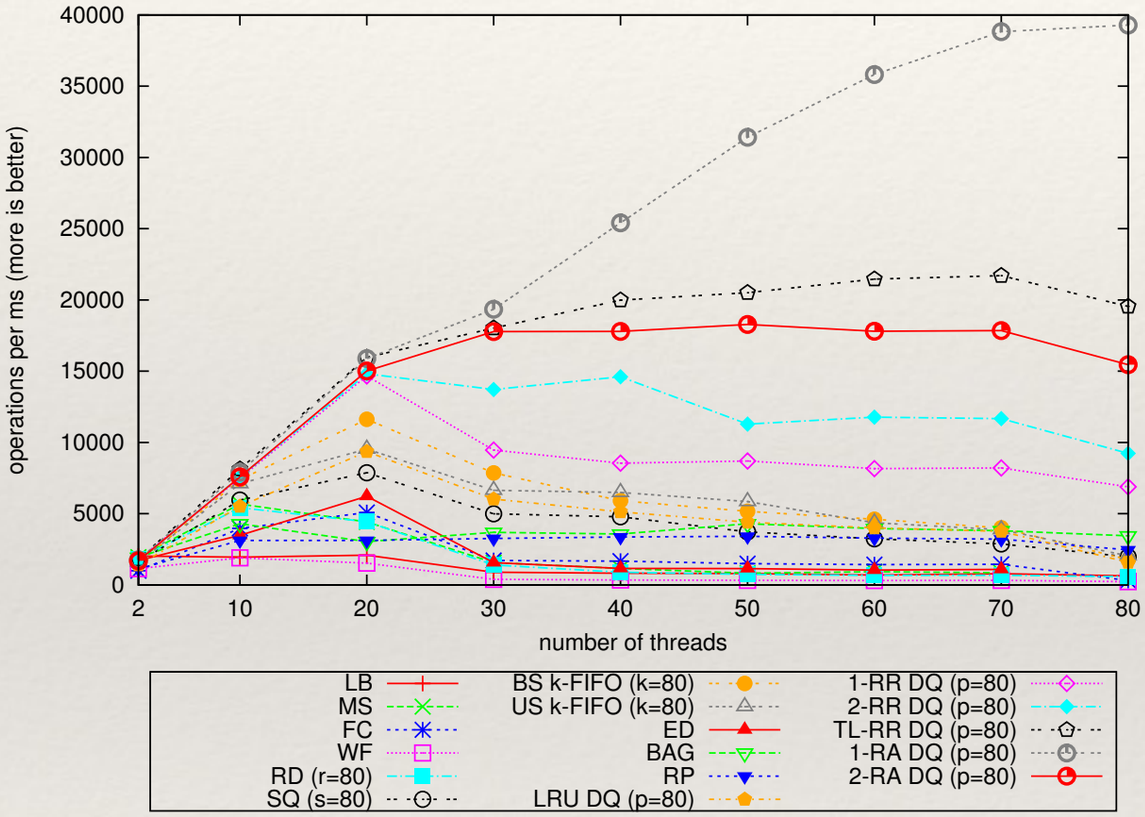
Distribution



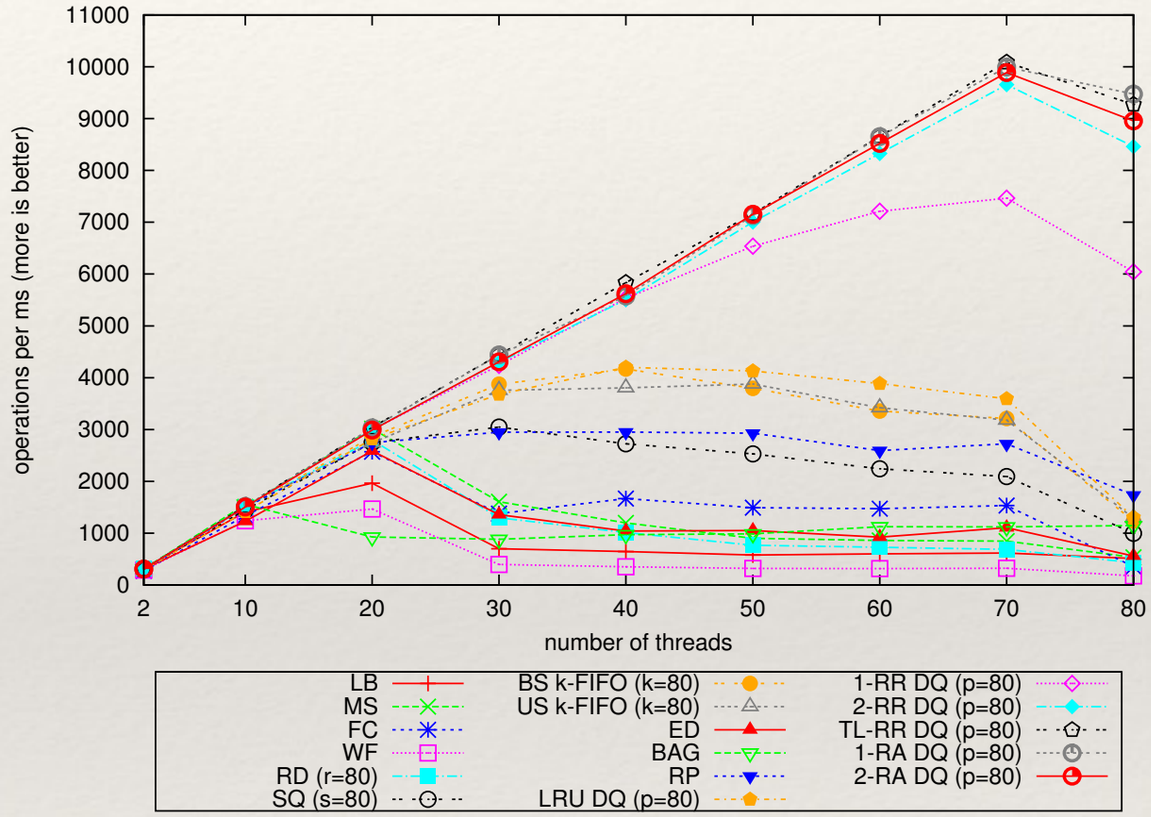
Load Balancers

- 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]



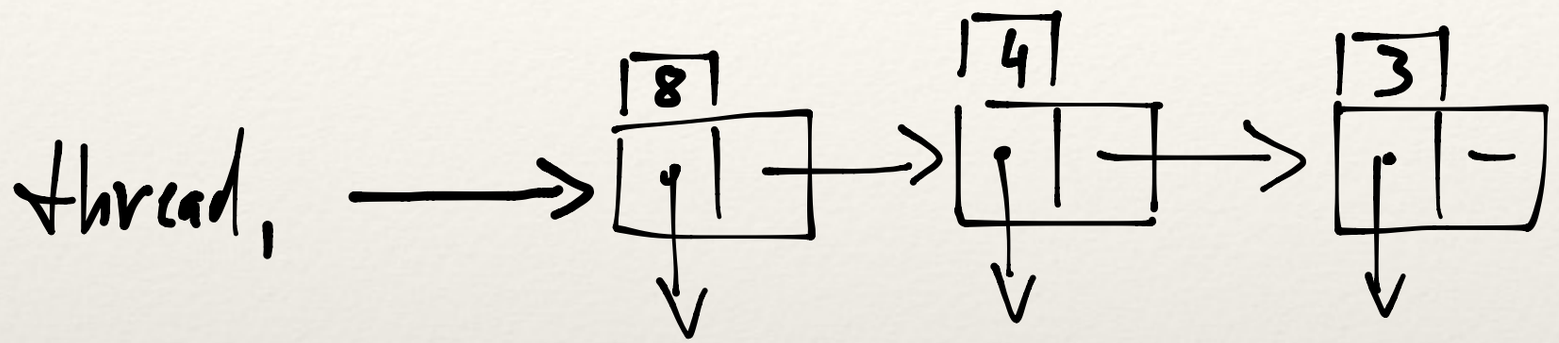
(a) High contention producer-consumer microbenchmark ($c = 250$)



(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

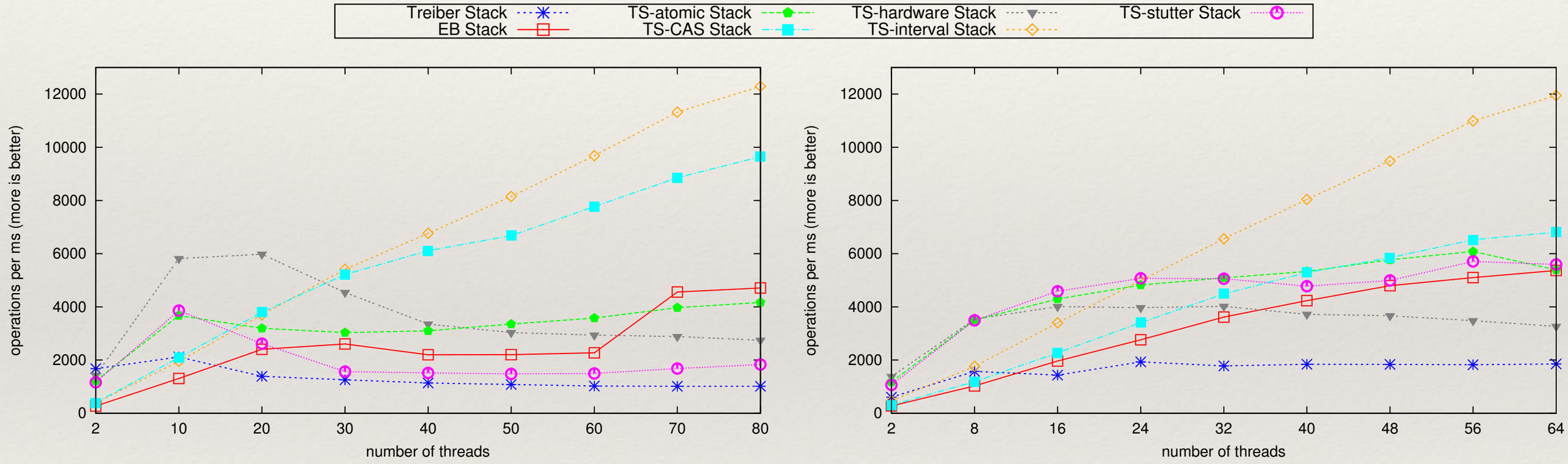
Timestamping



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
- …

scalloc.cs.uni-salzburg.at

- Scalloc is a fast, multi-threaded, multicore-scalable memory allocator with low memory consumption based on three new ideas (and many known ones):

1. Virtual spans (on demand paging)
2. Span-pool backend (lock-free distributed stack)
3. Constant-time frontend (locked doubly-linked list)

Challenges

- Allocation performance
- Deallocation performance
- Access performance
- Fragmentation
- Performance robustness
- Temporal and spatial scalability

Faster than others
while
using less memory

acdc.cs.uni-salzburg.at

- ACDC: configurable multi-threaded benchmark for memory management
- Simulates periodic allocation behavior (AC) and permanent memory (DC) based on models of real applications
- Simulates memory access
- Fully scalable on multicore hardware
- Available for C, C++, Java, JavaScript

Mutator

Allocator

Operating System

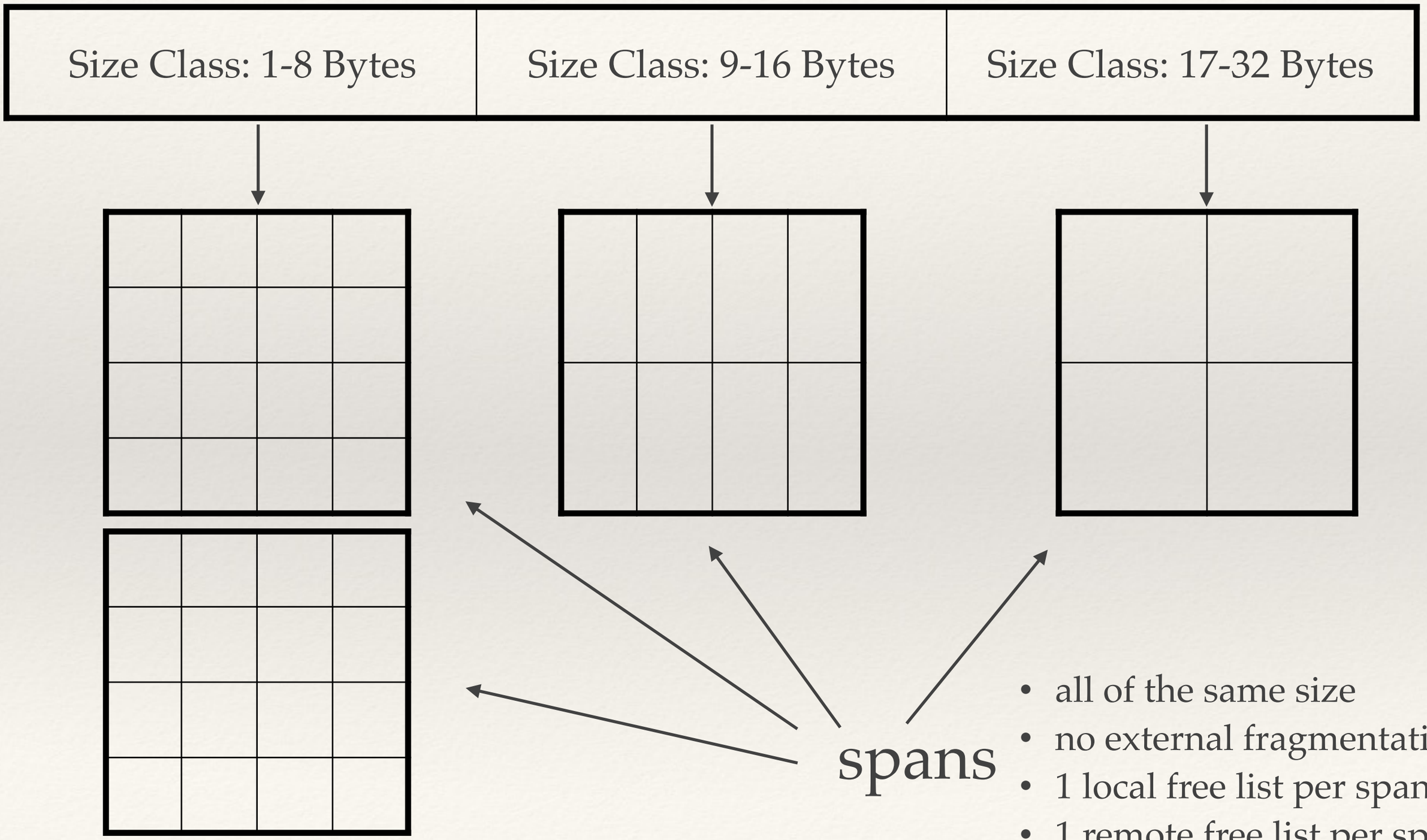
Old Idea:

Thread-local Allocation Buffers

TLABs: Good and Bad

- Fast allocation path
- Fast local deallocation path
- But potentially slow remote deallocation path
- And potentially high fragmentation

Old Idea: Size Classes



- all of the same size
- no external fragmentation
- 1 local free list per span
- 1 remote free list per span

Size Classes: Good and Bad

- No external fragmentation
- High locality
- But internal fragmentation
- Upper bound on object size
- Traditionally solved by using a different allocator for big objects

New Idea: Virtual Spans

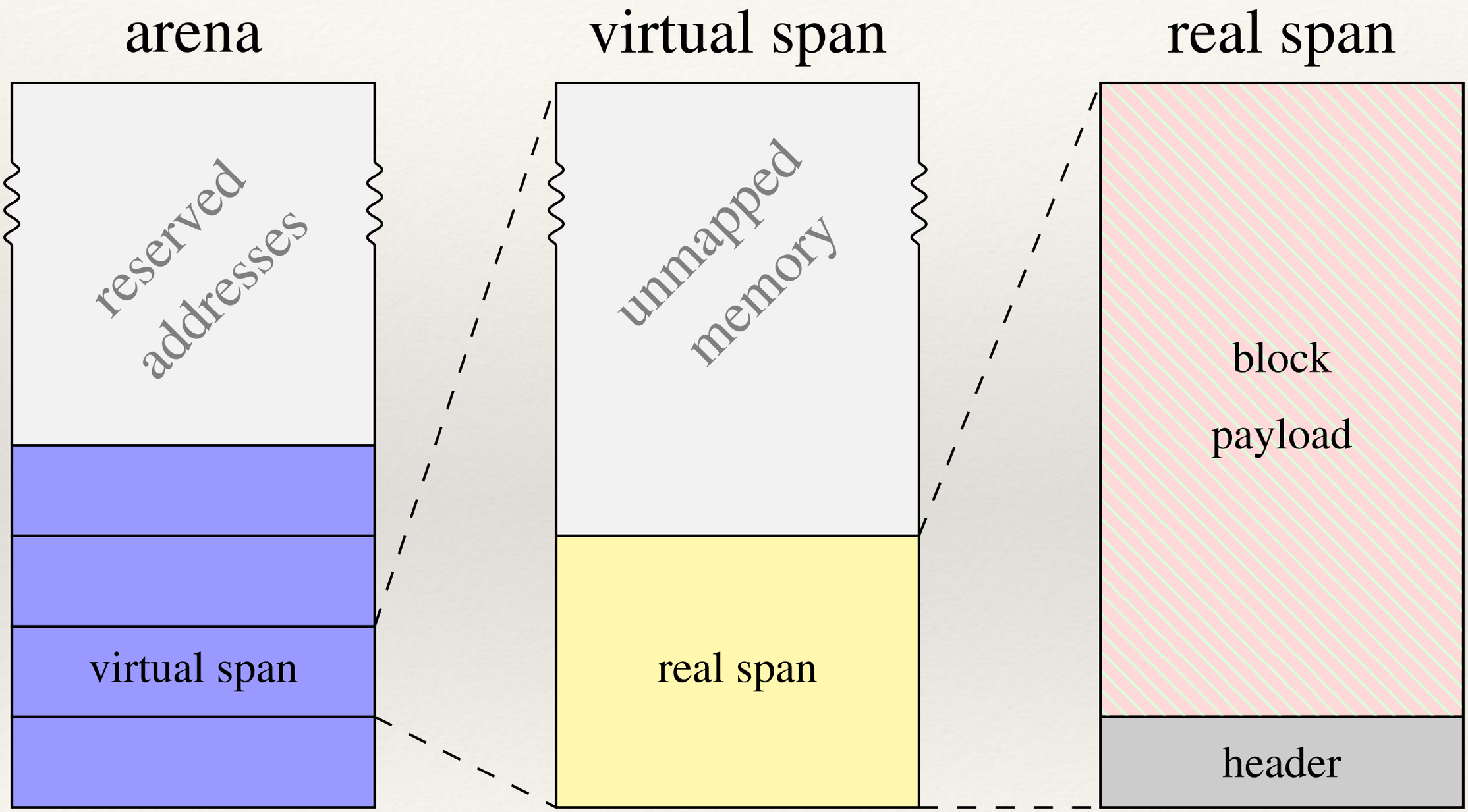


Figure 1: Structure of arena, virtual spans, and real spans

Virtual Spans: Good and Bad

- No external fragmentation
- High locality
- But internal fragmentation yet most only virtual
- Upper bound on object size larger than with real spans
- No hybrid allocator necessary, huge objects (>1MB) are mmapped
- But virtual memory (32TB) limits allocatable memory

New Idea: Global Span Pool

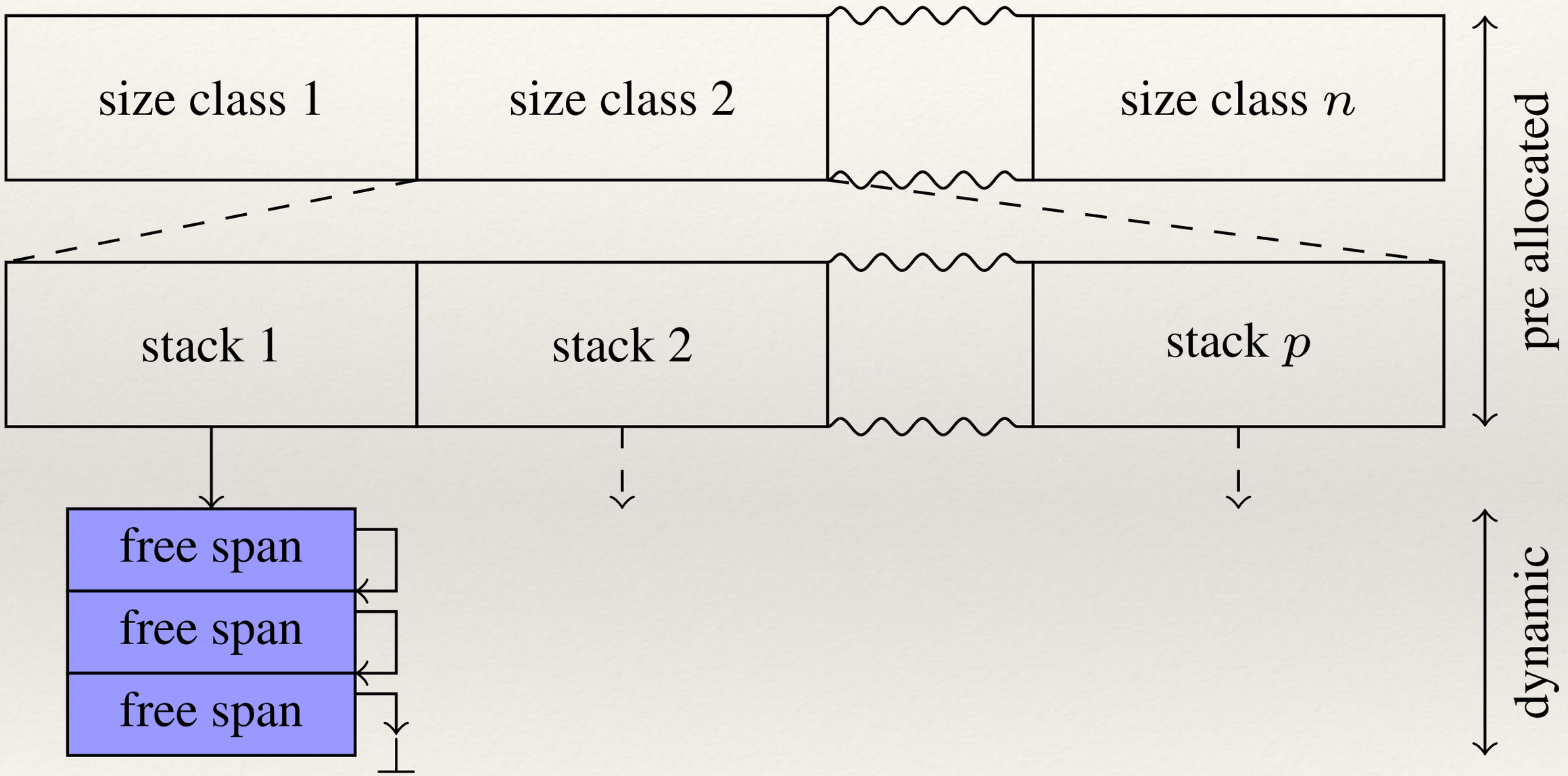


Figure 2: Span pool layout

New Idea: Constant-Time Frontend

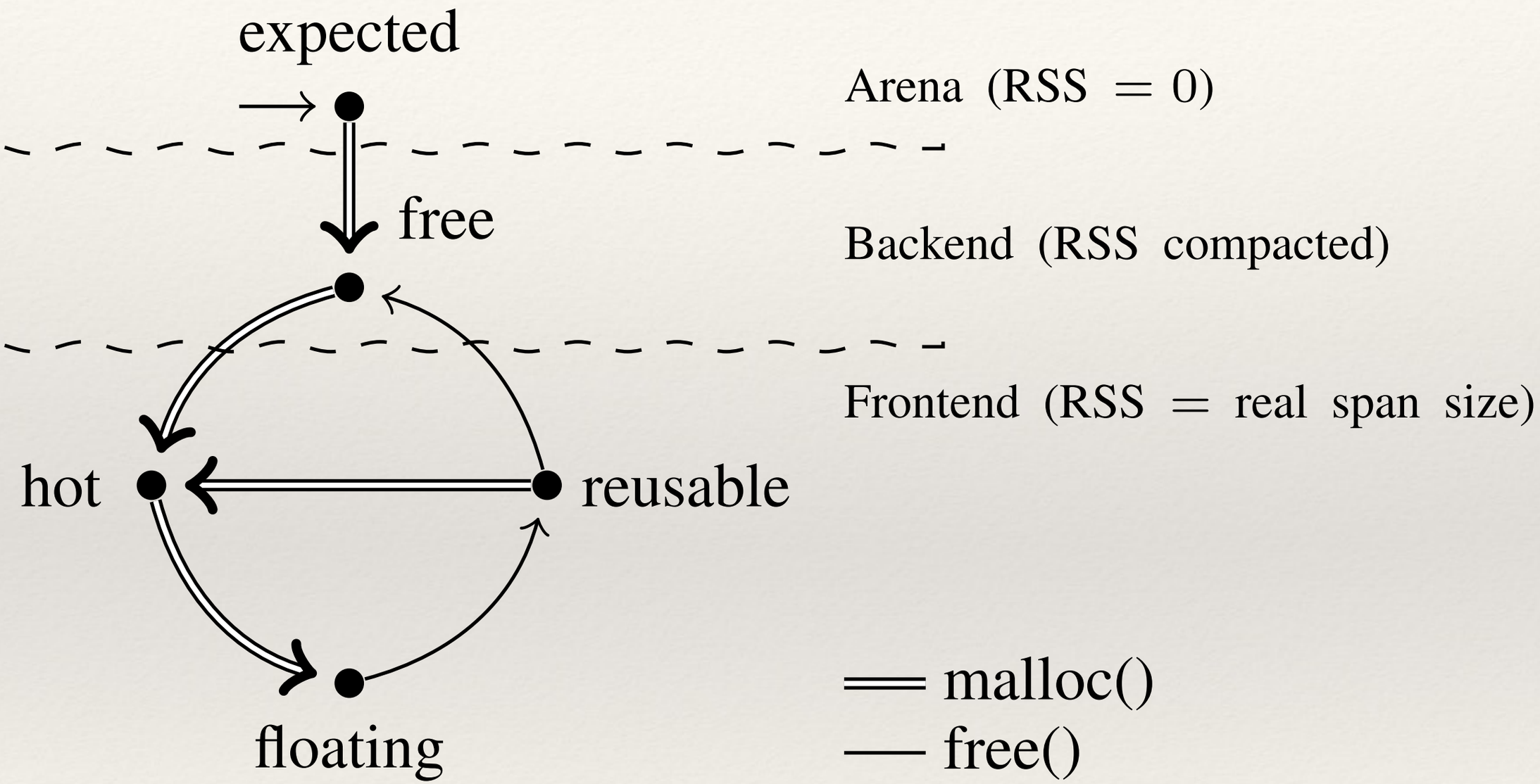
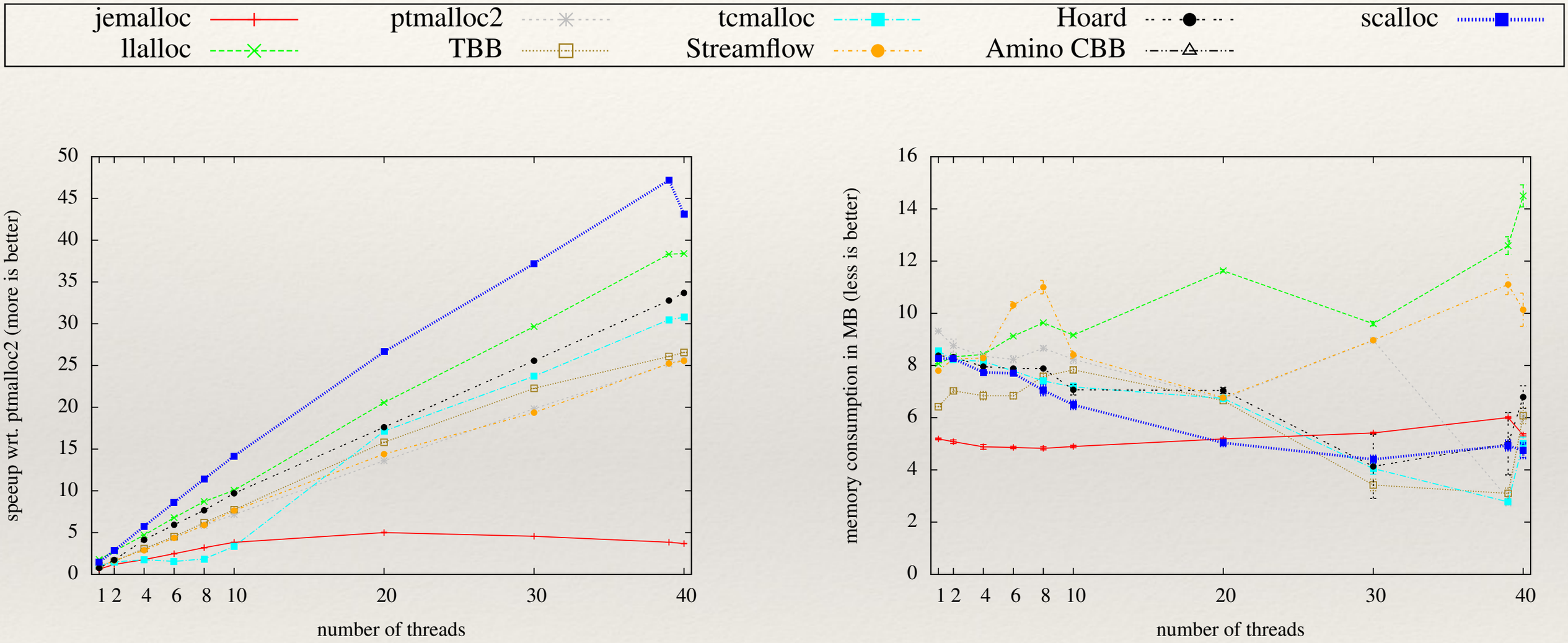


Figure 3: Life cycle of a span

Local Allocation & Deallocation

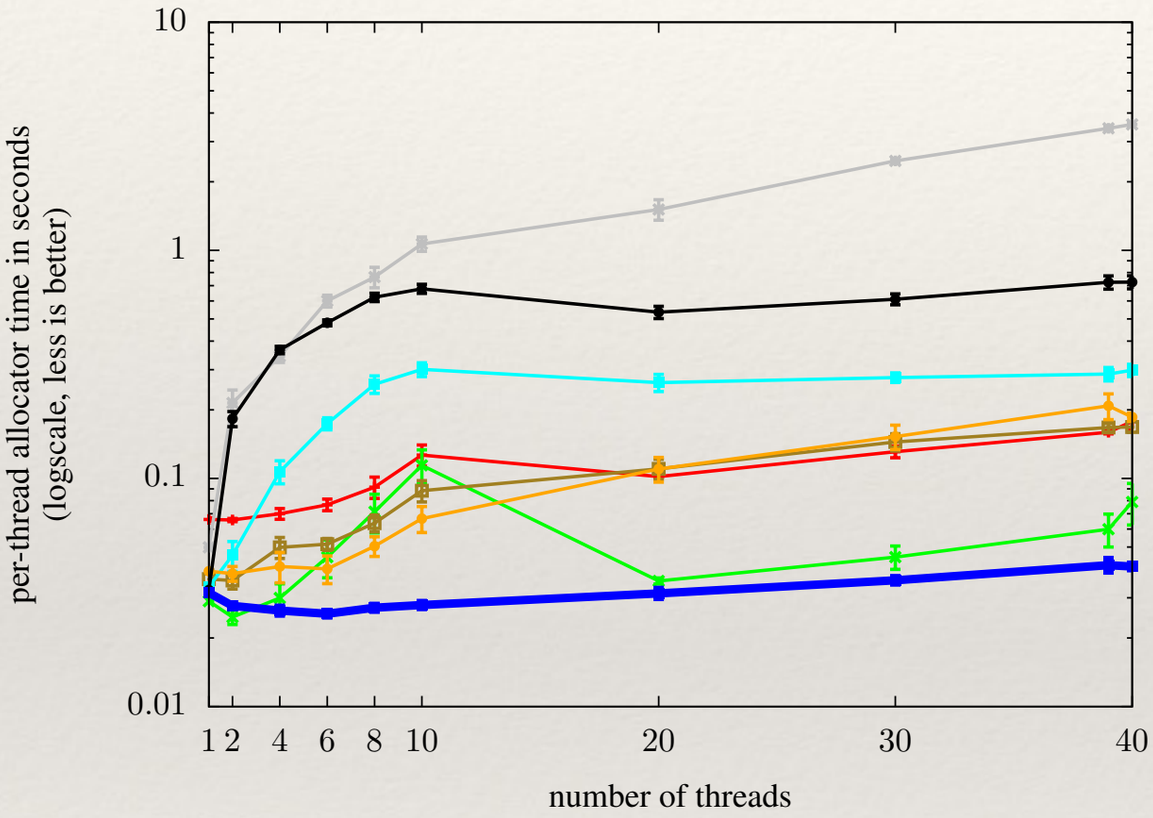


(a) Speedup

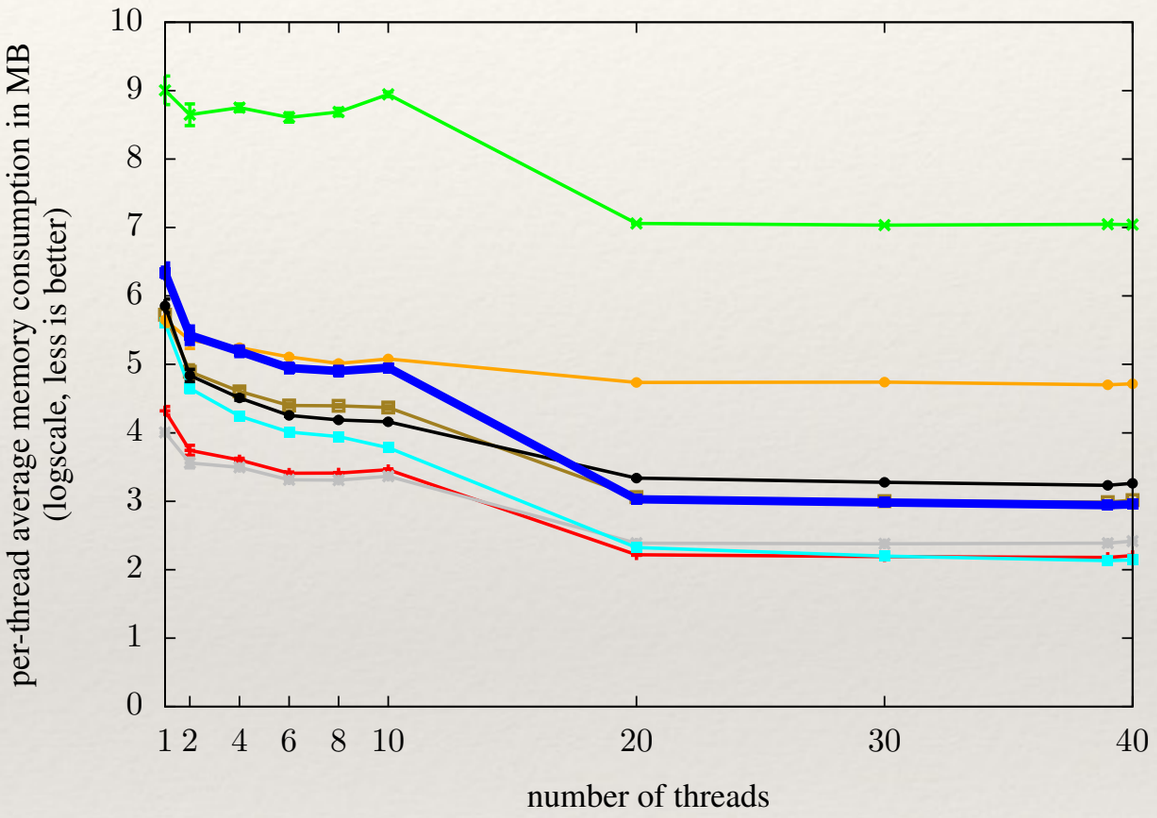
(b) Memory consumption

Figure 4: Threadtest benchmark

Remote Deallocation



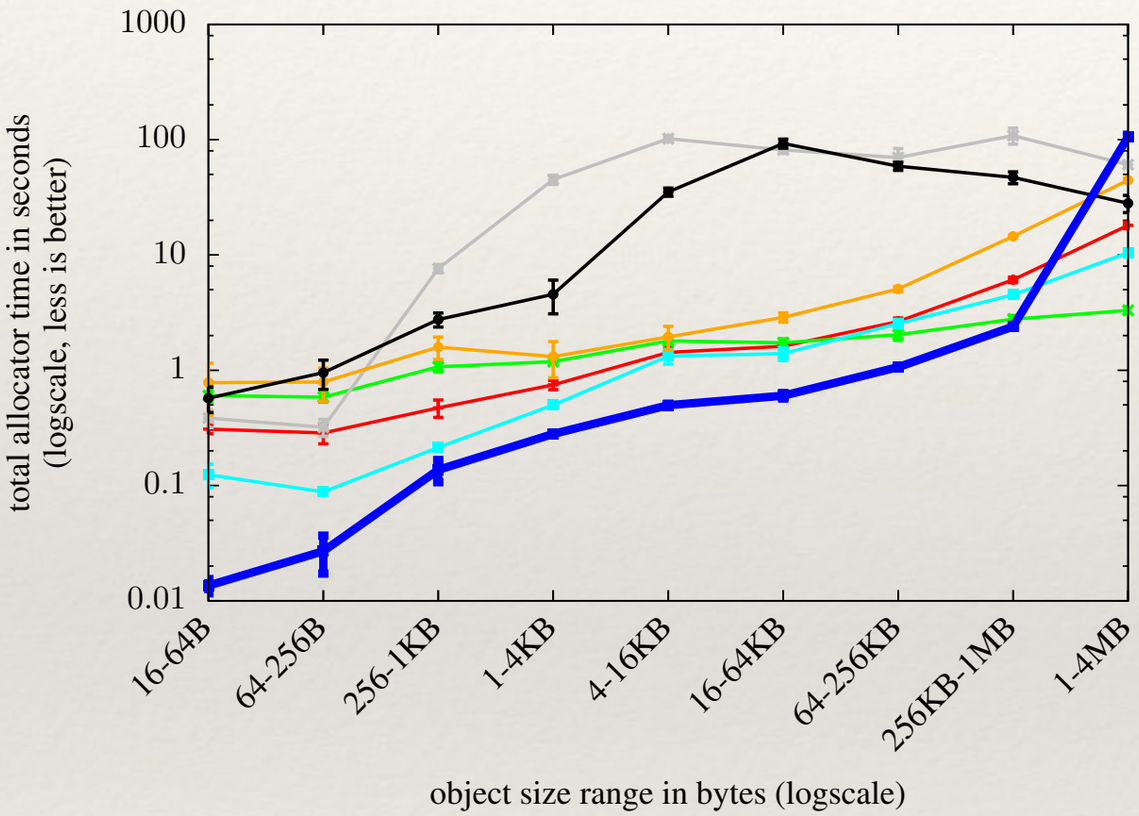
(a) Per-thread allocator time



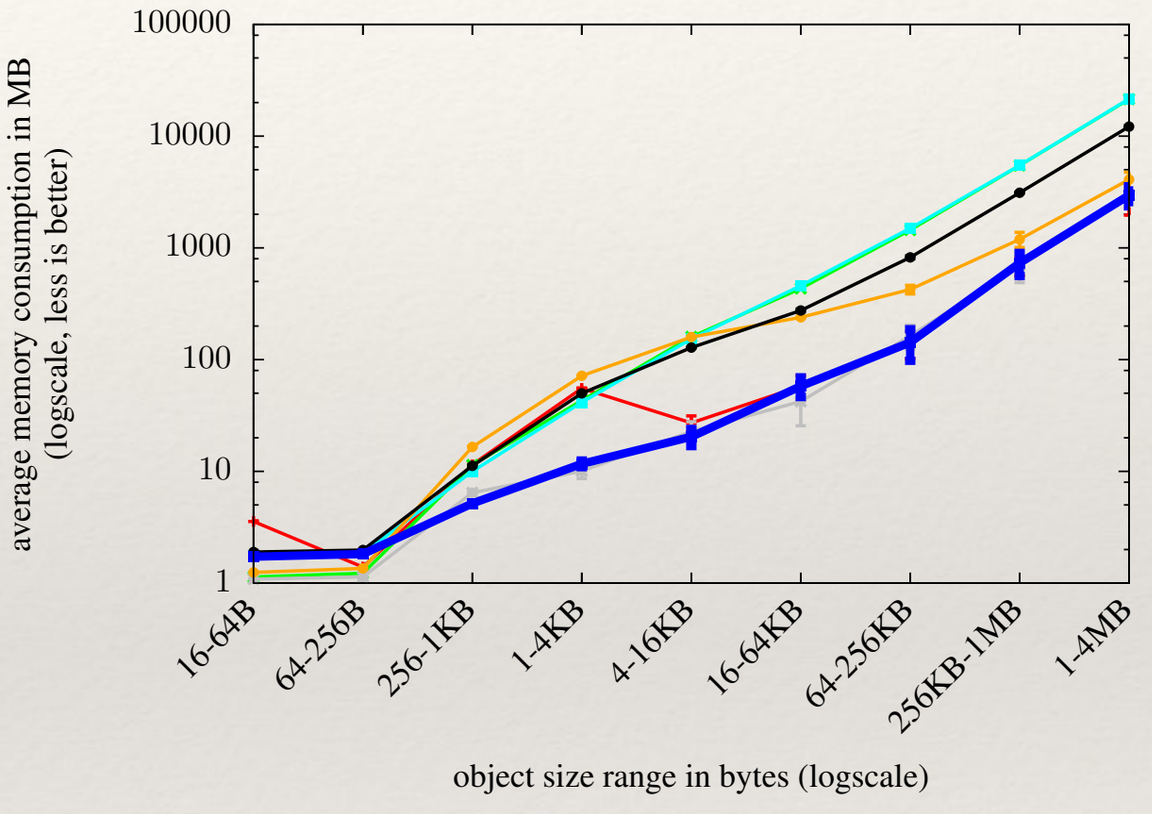
(b) Per-thread average memory consumption

Figure 9: Temporal and spatial performance for the remote-free/blowup robustness experiment

Object Size



(a) Total allocator time



(b) Average memory consumption

Figure 7: Temporal and spatial performance for the object-size robustness experiment

Memory Access

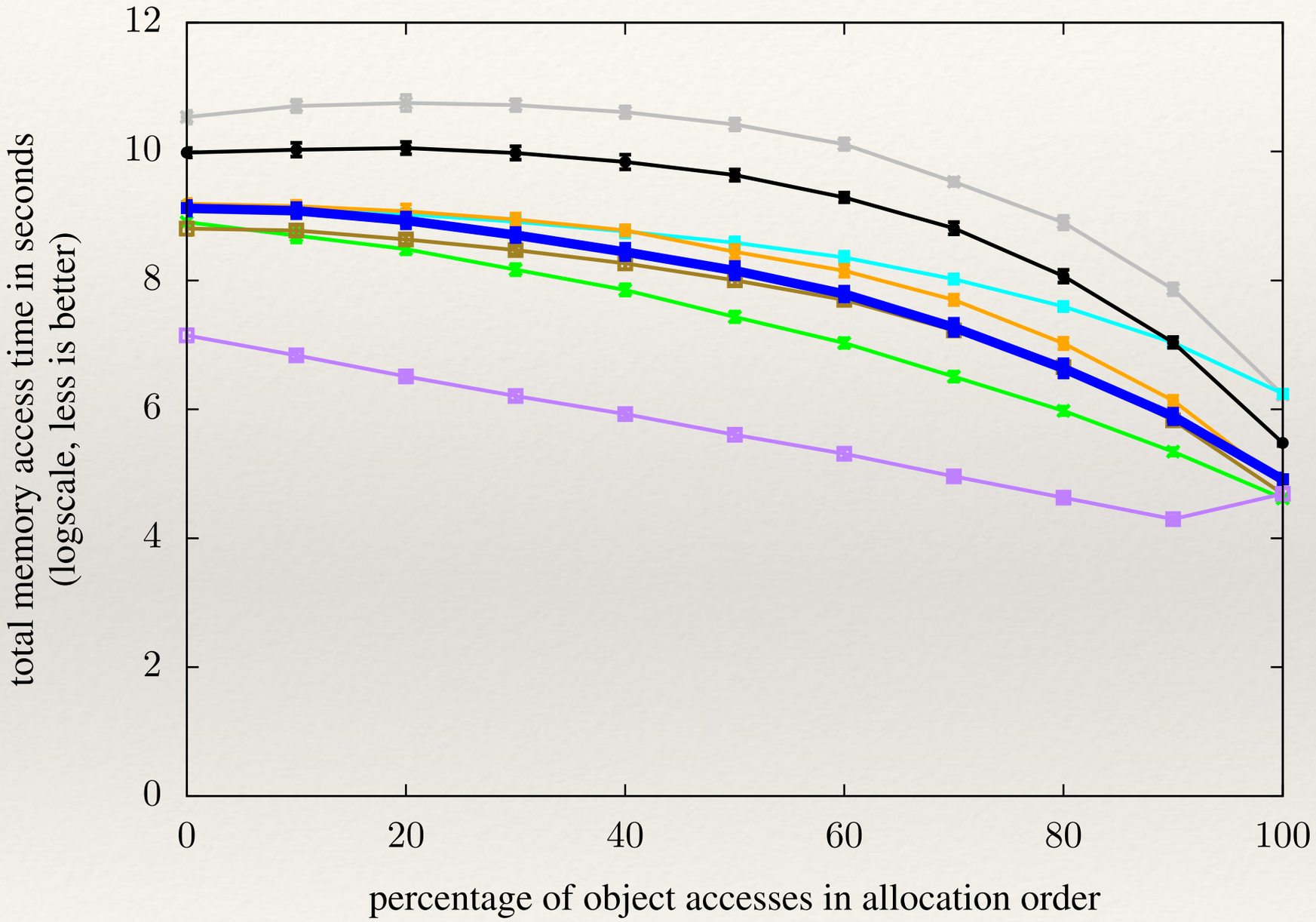


Figure 8: Memory access time for the locality experiment

Thank you