

scal.cs.uni-
salzburg.at

multicore-scalable
concurrent data
structures

scalloc.cs.uni-
salzburg.at

multicore-scalable
concurrent allocator

selfie.cs.uni-
salzburg.at

self-referential systems
software for teaching

Scal, Scalloc, and Selfie

Christoph Kirsch, University of Salzburg, Austria

Joint Work

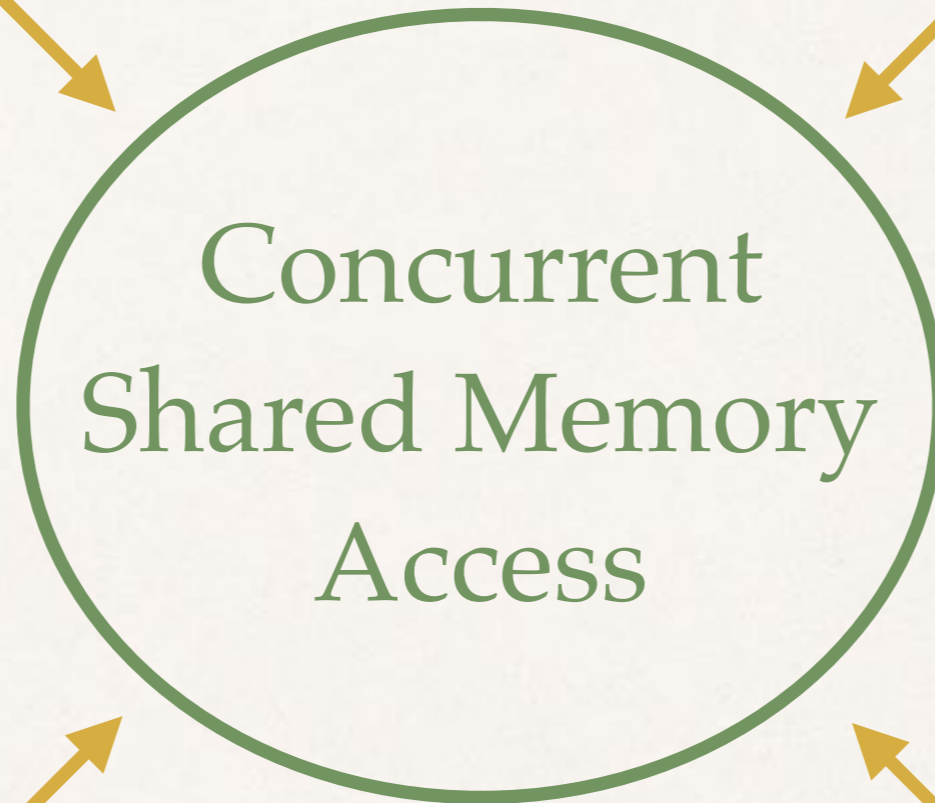
- ❖ Martin Aigner
- ❖ Christian Barthel
- ❖ Mike Dodds
- ❖ Andreas Haas
- ❖ Thomas Henzinger
- ❖ Andreas Holzer
- ❖ Thomas Hütter
- ❖ Michael Lippautz
- ❖ Alexander Miller
- ❖ Simone Oblasser
- ❖ Hannes Payer
- ❖ Mario Preishuber
- ❖ Ana Sokolova
- ❖ Ali Szegin

How do we exchange data among increasingly many cores on a shared memory machine such that performance still increases with the number of cores?

–The Multicore Scalability Challenge

Core 1

Core 2



Core 3

Core 4

~100 cores

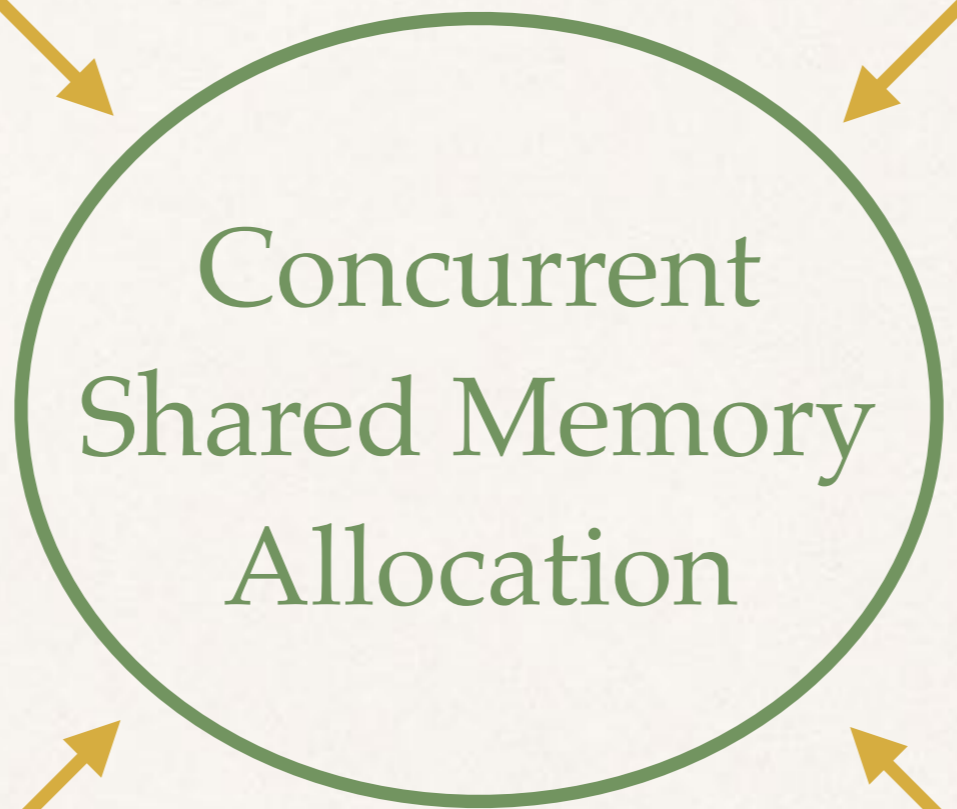
How do we allocate and deallocate shared memory with increasingly many cores such that performance increases with the number of cores while memory consumption stays low?

–Multicore Shared Memory Allocation Problem

~1TB memory

Core 1

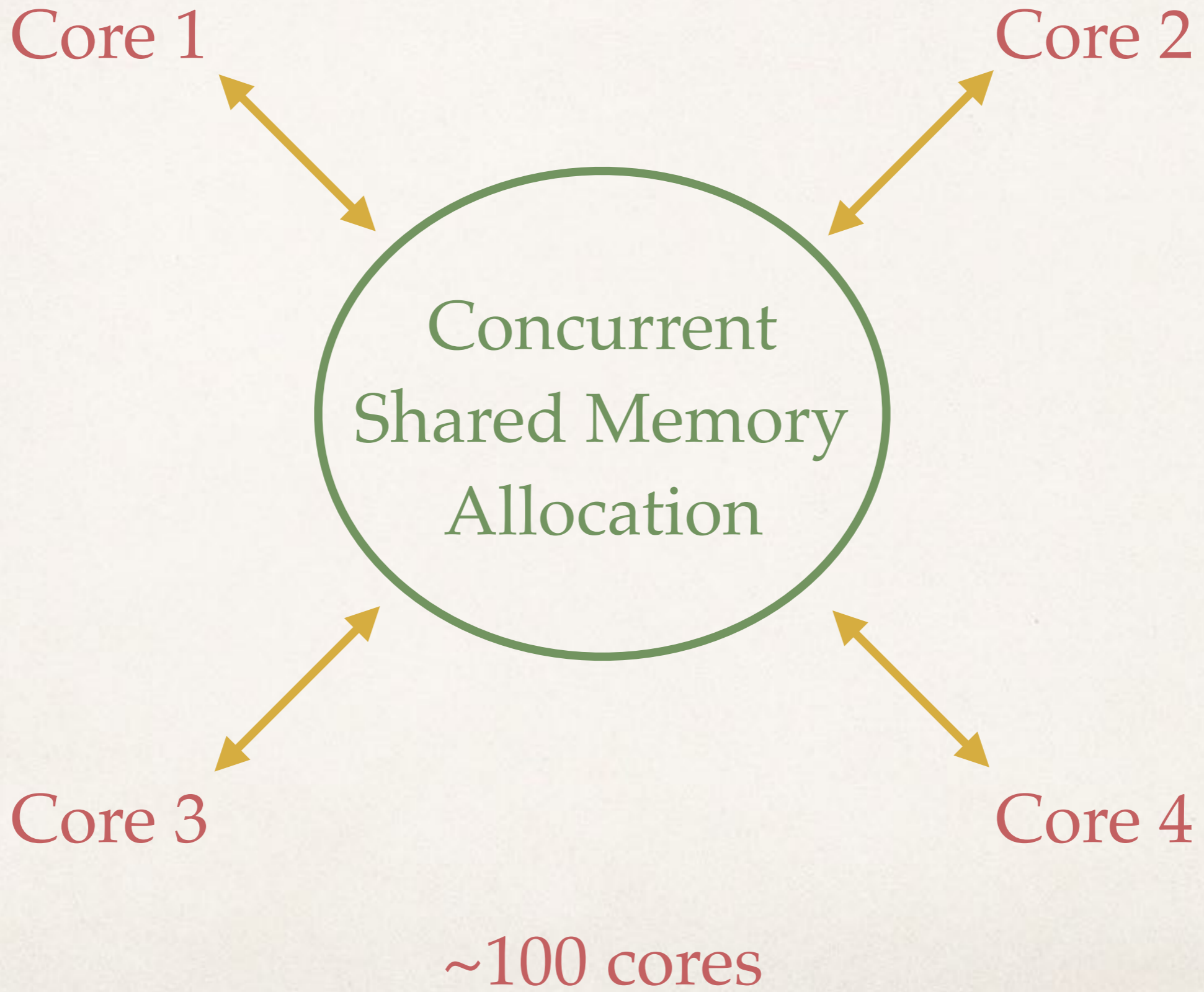
Core 2



Core 3

Core 4

~100 cores

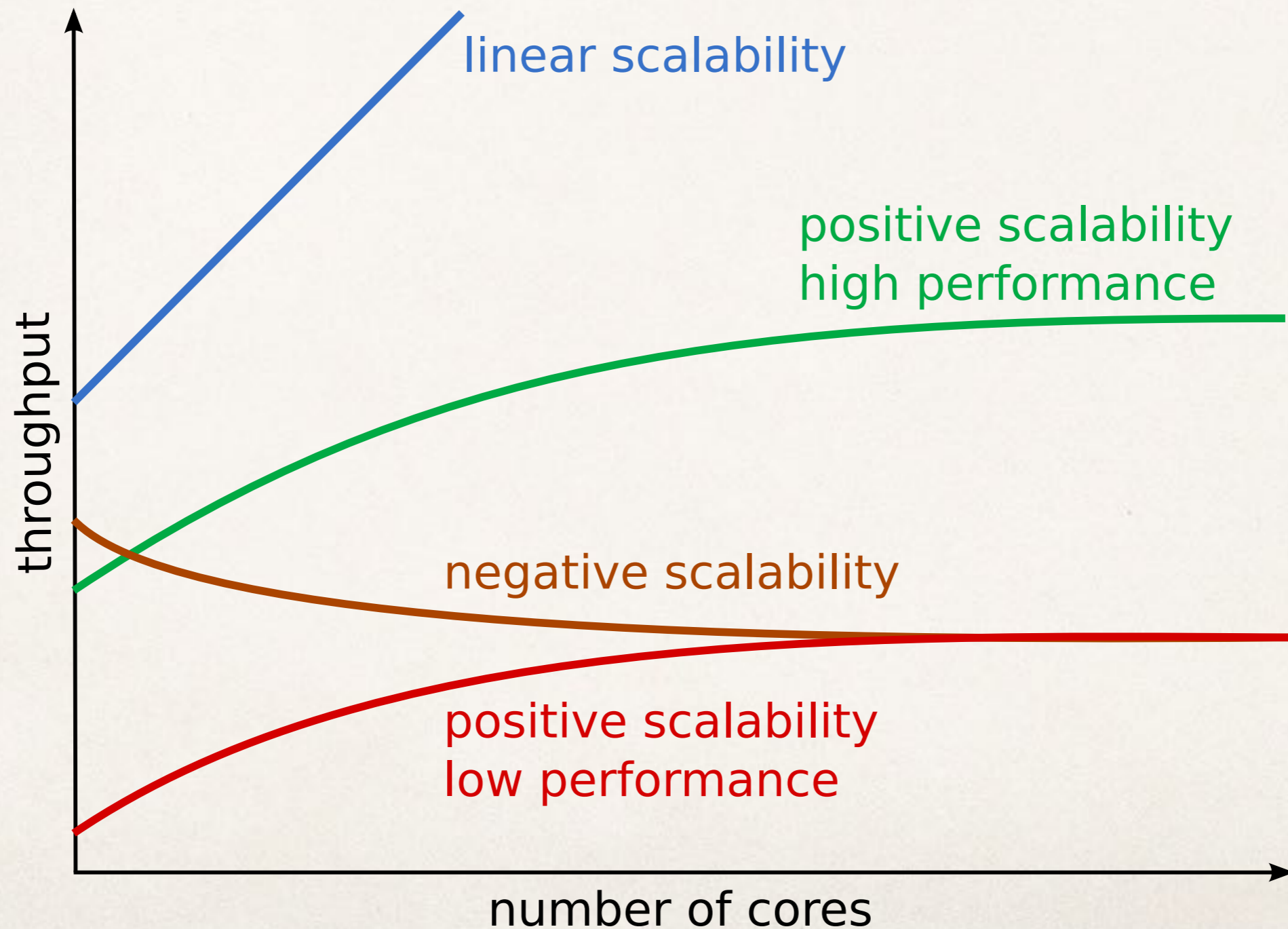


How do we teach computer science to students not necessarily majoring in computer science but who anyway code every day?

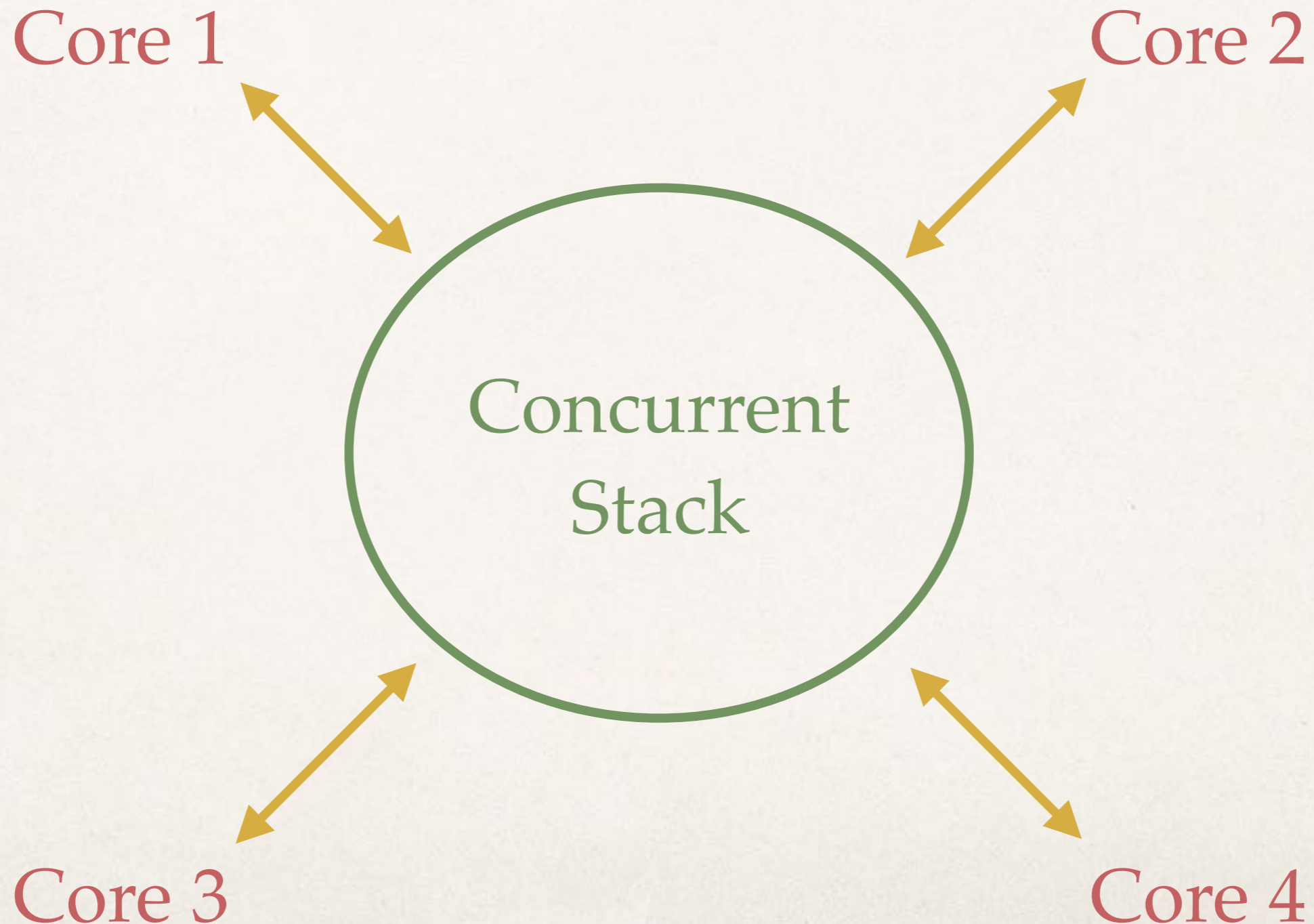
–The Computer Science Education Challenge



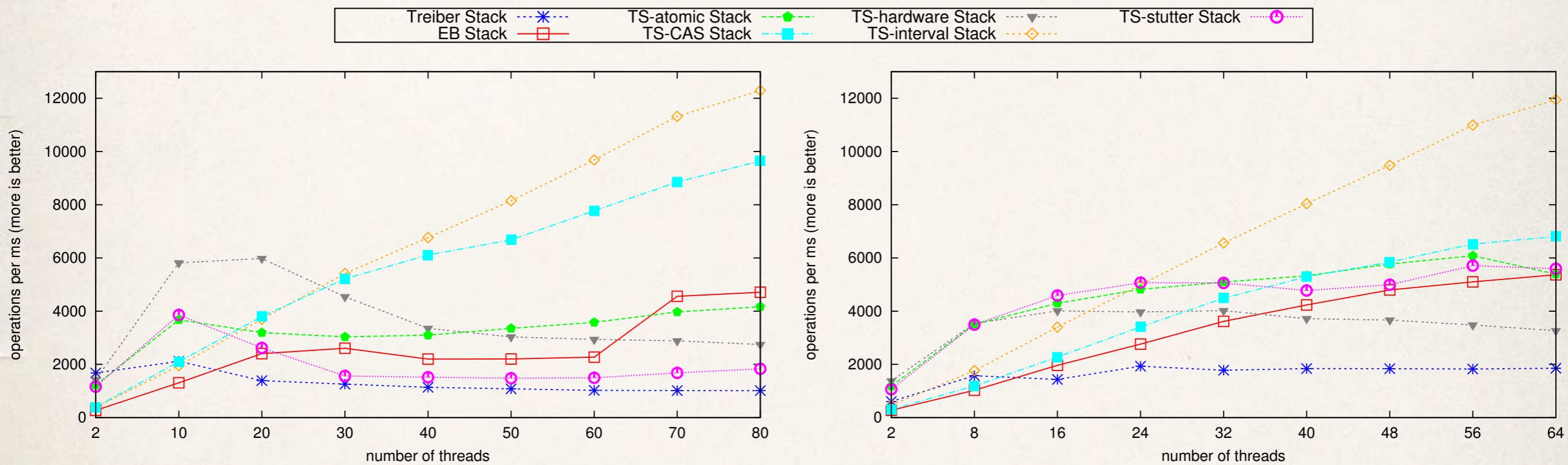
The Multicore Scalability Challenge



Timestamped (TS) Stack [POPL15]



Timestamped (TS) Stack [POPL15]



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

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

Elimination Through Shared Timestamps

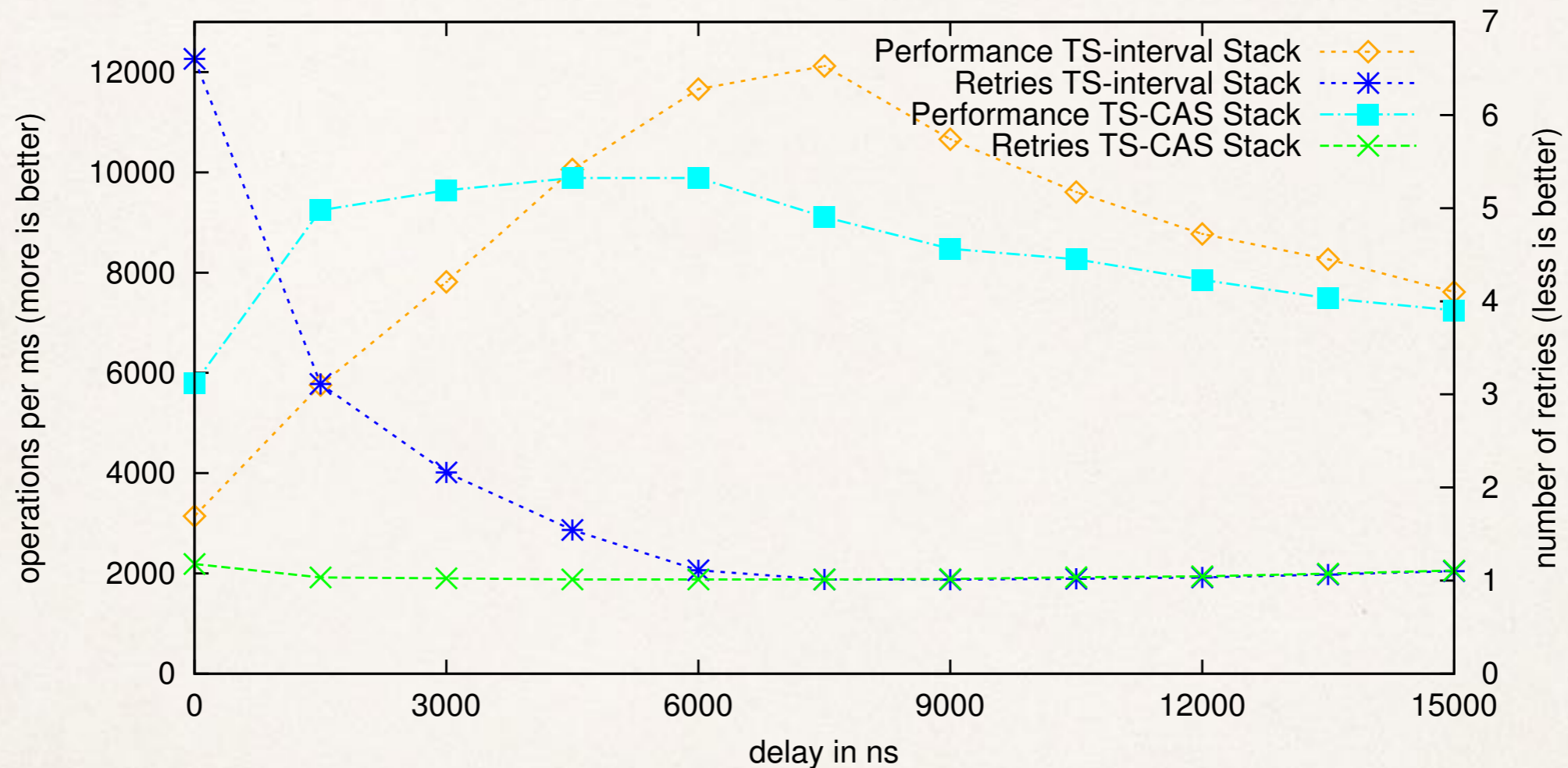
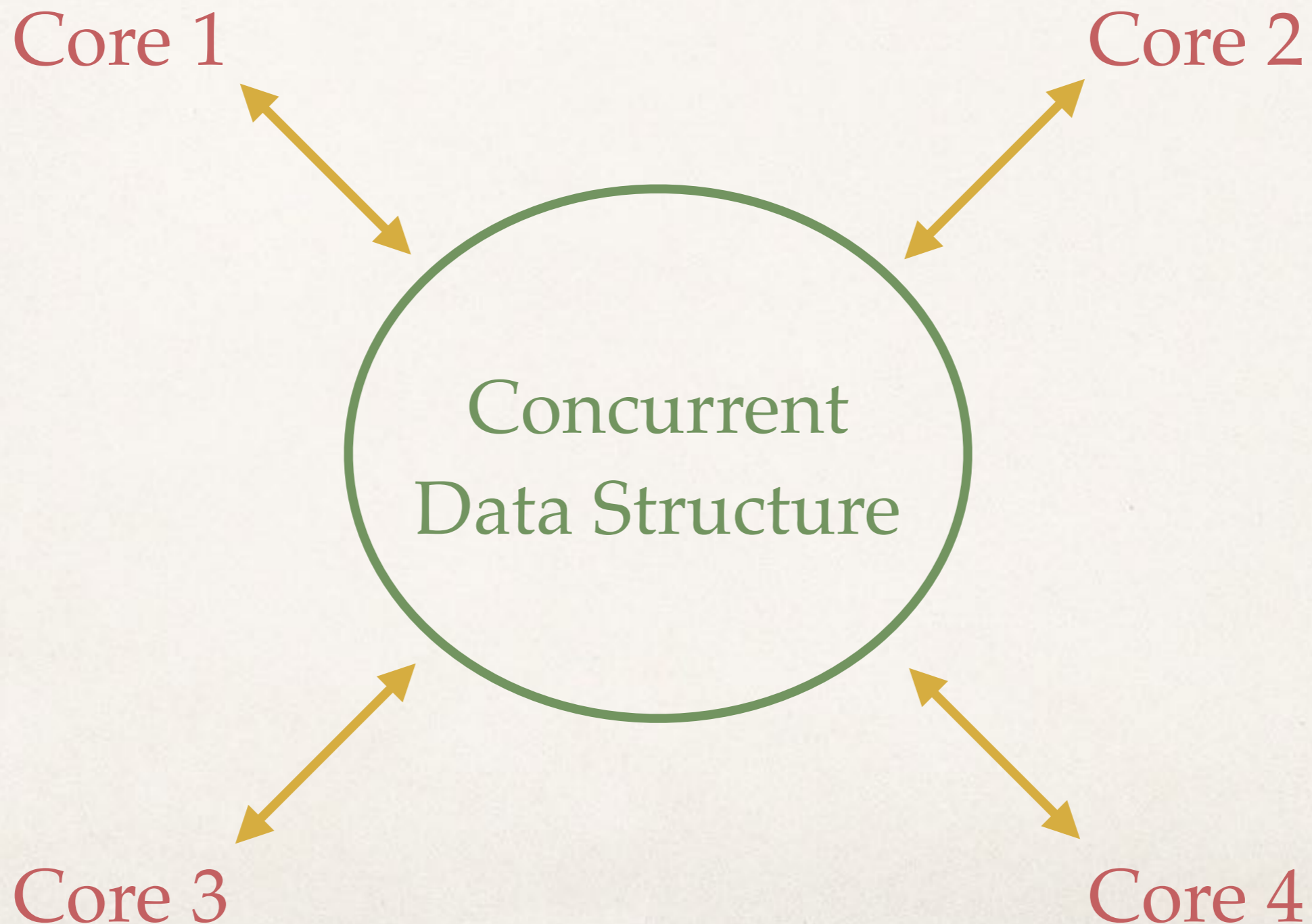


Figure 6: High-contention producer-consumer benchmark using TS-interval and TS-CAS timestamping with increasing delay on the 40-core machine, exercising 40 producers and 40 consumers.

The order in which concurrently pushed elements appear on the TS stack is only determined when they are popped off the TS stack, even if they had been on the TS stack for, say, a year.

–Showing that the TS Stack is a stack is hard hence POPL

Concurrent Data Structures: scal.cs.uni-salzburg.at [POPL13, CF13, POPL15, NETYS15]



Scal: A Benchmarking Suite for Concurrent Data Structures [NETYS15]

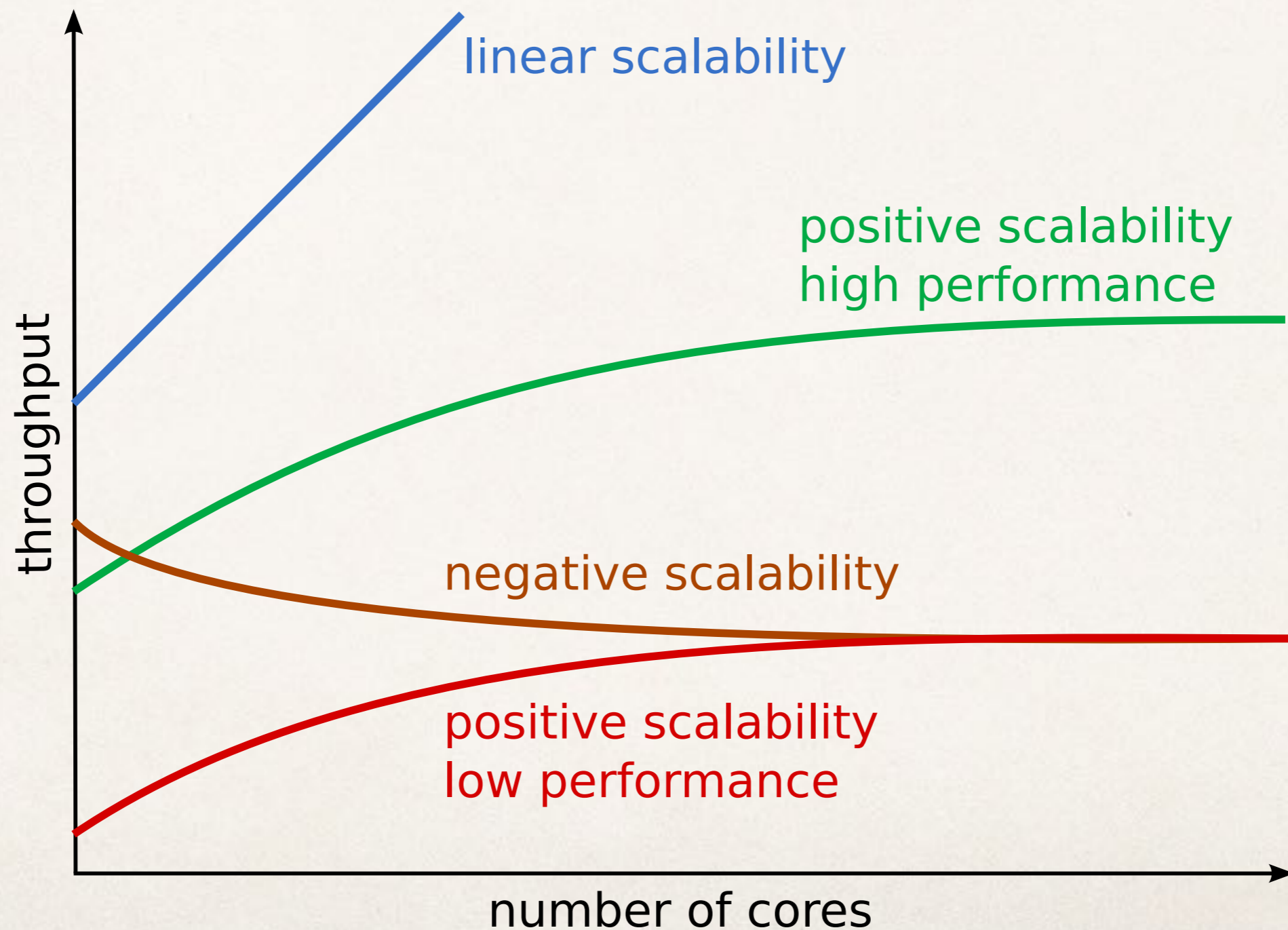
Name	Semantics	Year	Ref
Lock-based Singly-linked	strict queue	1968	[1]
Michael Scott (MS) Queue	strict queue	1996	[2]
Flat Combining Queue	strict queue	2010	[3]
Wait-free Queue	strict queue	2012	[4]
Linked Cyclic Ring Queue	strict queue	2013	[5]
Timestamped (TS) Queue	strict queue	2015	[6]
Cooperative TS Queue	strict queue	2015	[7]
Segment Queue	k-relaxed queue	2010	[8]
Random Dequeue (RD)	k-relaxed queue	2010	[8]
Bounded Size k-FIFO	k-relaxed queue, pool	2013	[9]
Unbounded Size k-FIFO	k-relaxed queue, pool	2013	[9]
b-RR Distributed Queue	k-relaxed queue, pool	2013	[10]
Least-Recently-Used (LRU)	k-relaxed queue, pool	2013	[10]
Locally Linearizable DQ	locally linearizable	2015	[11]
Locally Linearizable k-FIFO	locally linearizable	2015	[11]
Relaxed TS Queue	quiescently consistent	2015	[7]
Lock-based Singly-linked	strict stack	1968	[1]
Treiber Stack	strict stack	1986	[12]
Elimination-backoff Stack	strict stack	2004	[13]
Timestamped (TS) Stack	strict stack	2015	[6]
k-Stack	k-relaxed stack	2013	[14]
b-RR Distributed Stack (DS)	k-relaxed stack, pool	2013	[10]
Least-Recently-Used (LRU)	k-relaxed stack, pool	2013	[10]
Locally Linearizable DS	locally linearizable	2015	[11]
Locally Linearizable k-Stack	locally linearizable	2015	[11]
Timestamped (TS) Deque	strict deque	2015	[7]
d-RA DQ and DS	strict pool	2013	[10]



How do we allocate and deallocate shared memory with increasingly many cores such that performance increases with the number of cores while memory consumption stays low?

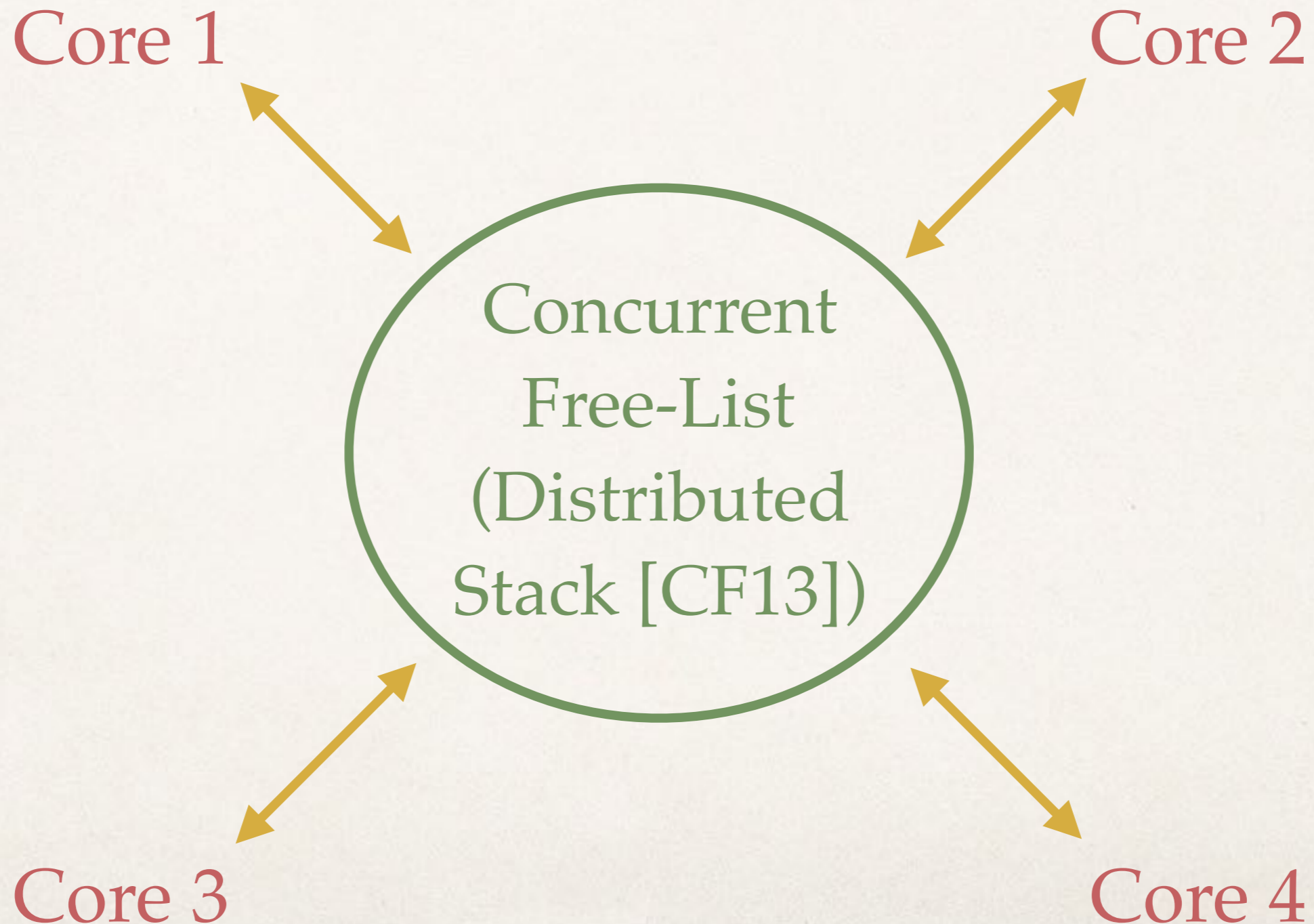
–Multicore Shared Memory Allocation Problem

Multicore Memory Allocation Problem



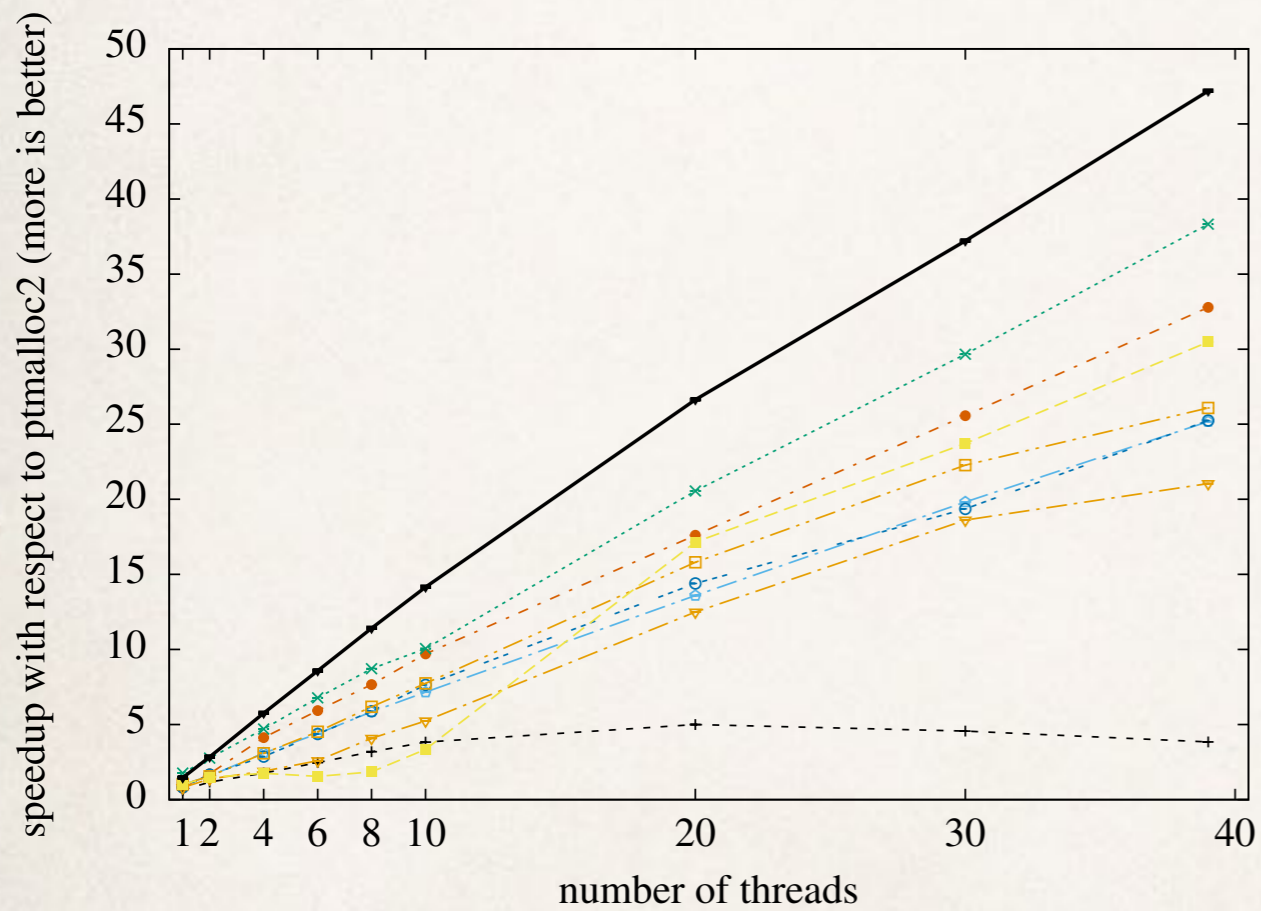
Scalloc: Concurrent Memory Allocator

scalloc.cs.uni-salzburg.at [OOPSLA15]

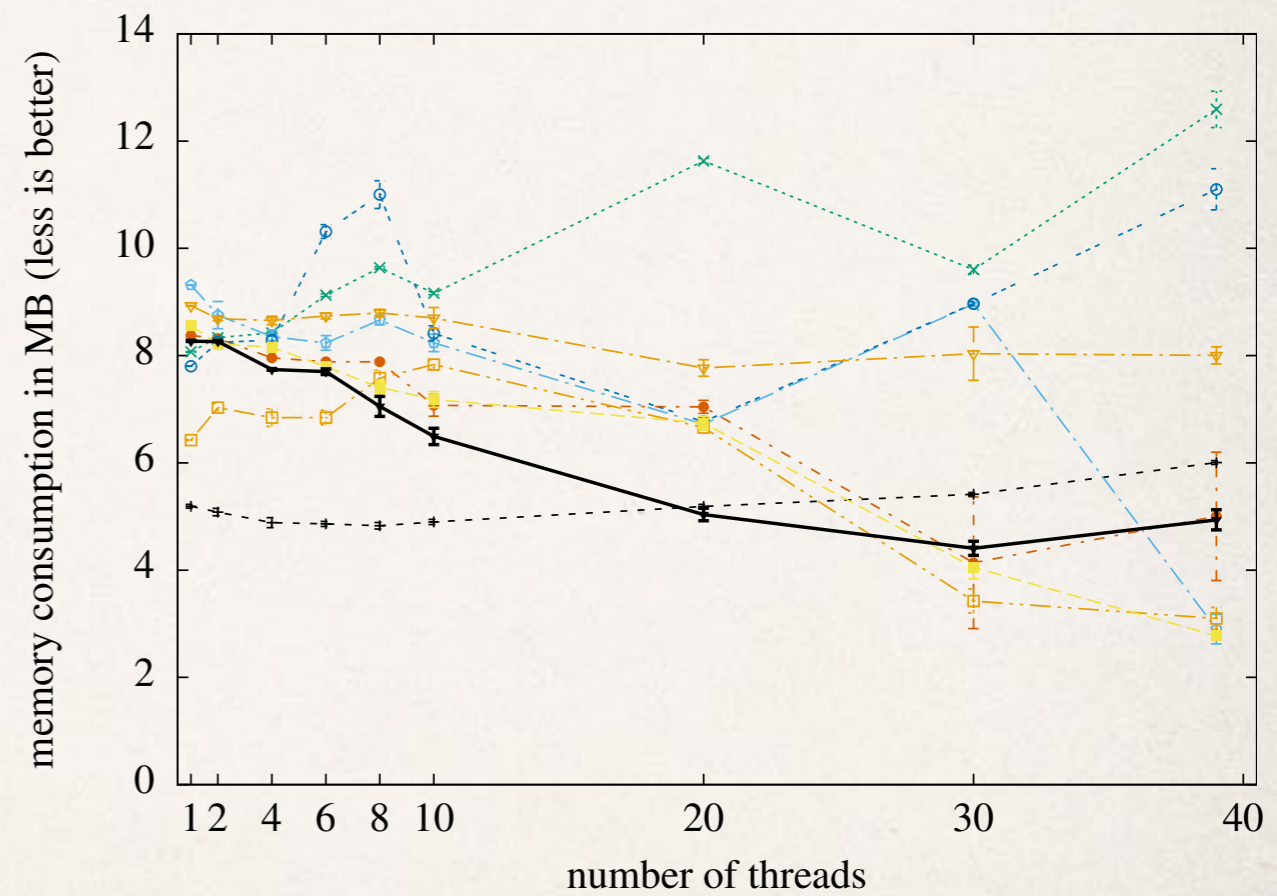




Local Allocation & Deallocation



(a) Speedup



(b) Memory consumption

Figure 6: Thread-local workload: Threadtest benchmark



“Remote” Deallocation

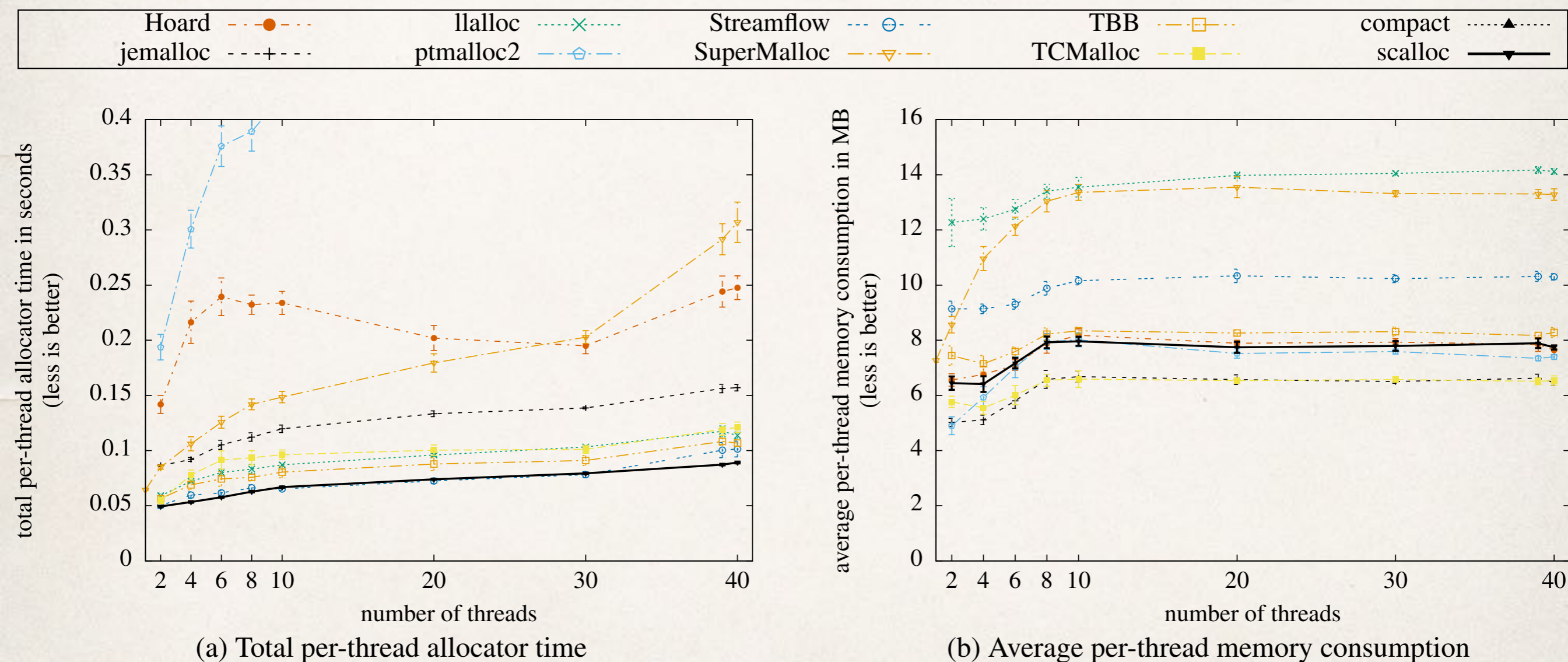
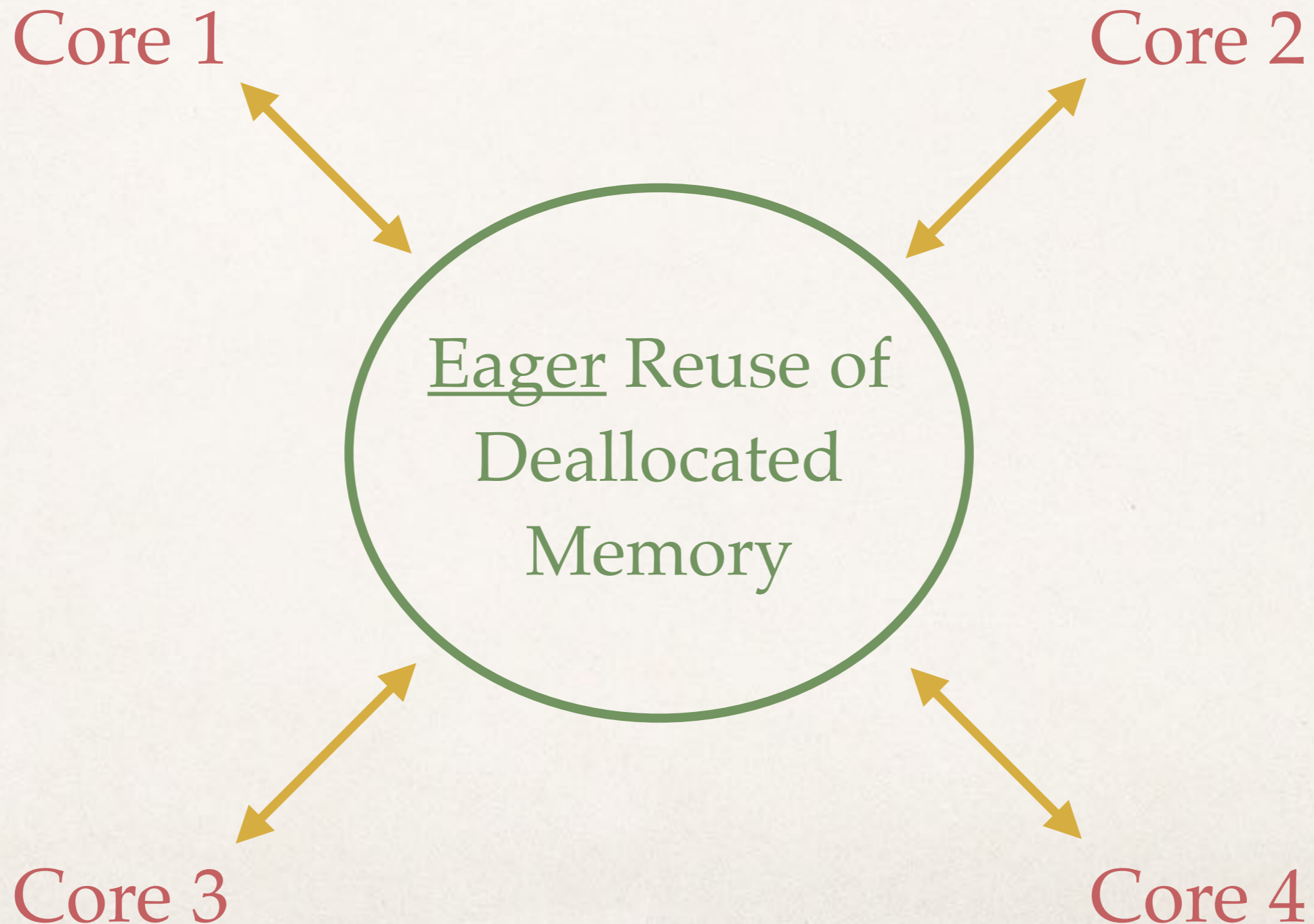


Figure 9: Temporal and spatial performance for the producer-consumer experiment

Scalloc: Concurrent Memory Allocator

scalloc.cs.uni-salzburg.at [OOPSLA15]



Virtual Spans: 64-bit Address Space

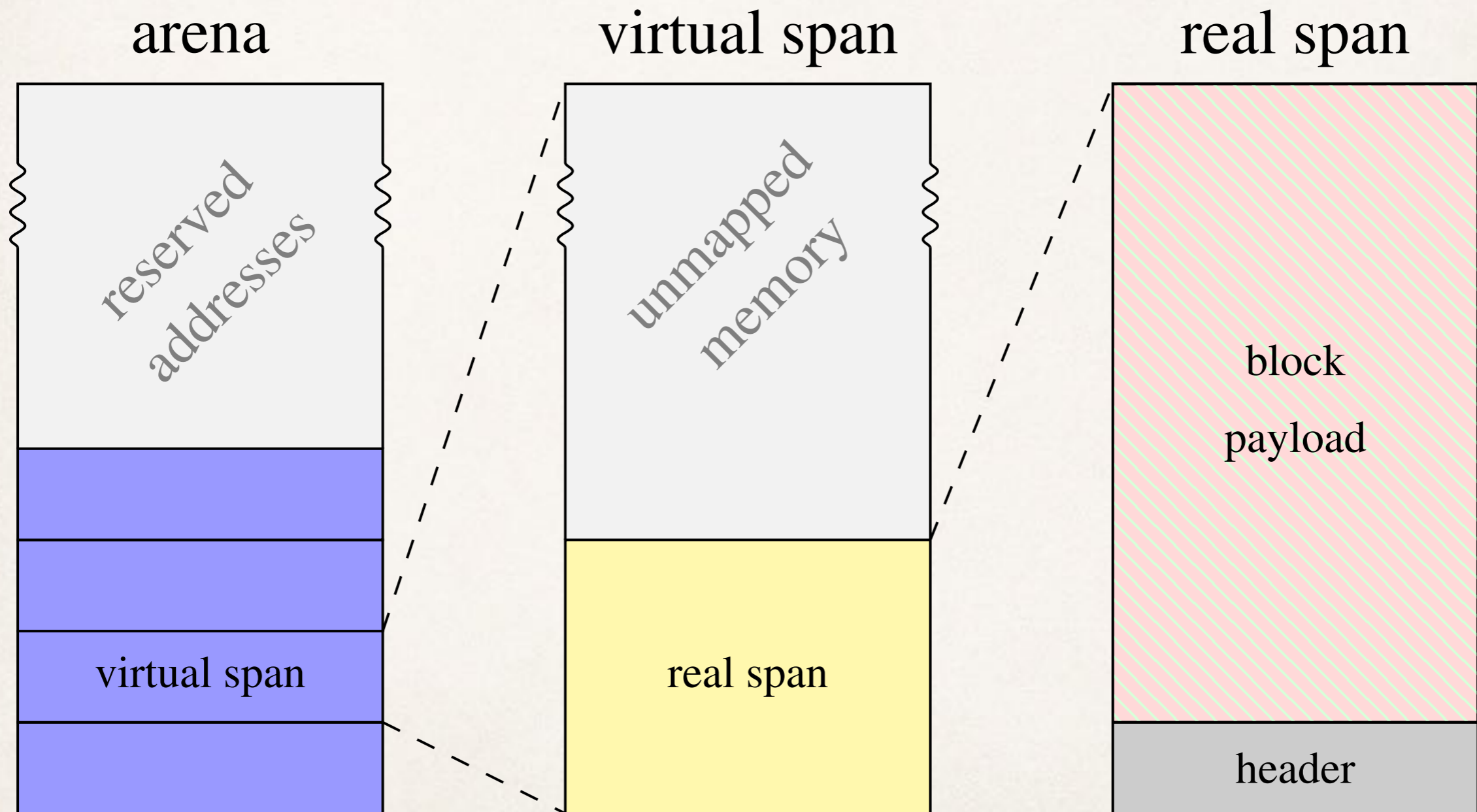


Figure 1: Structure of arena, virtual spans, and real spans

Object Size

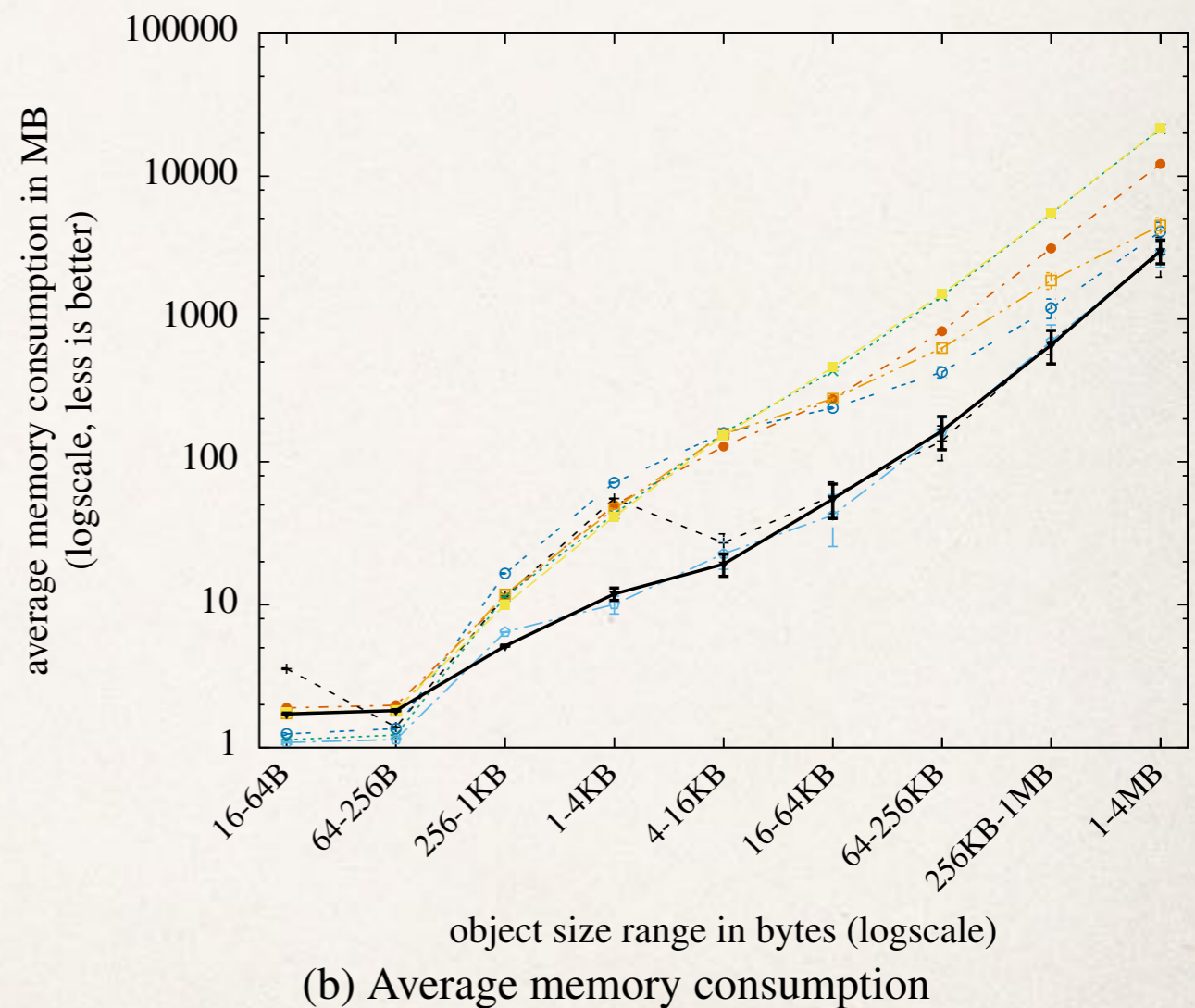
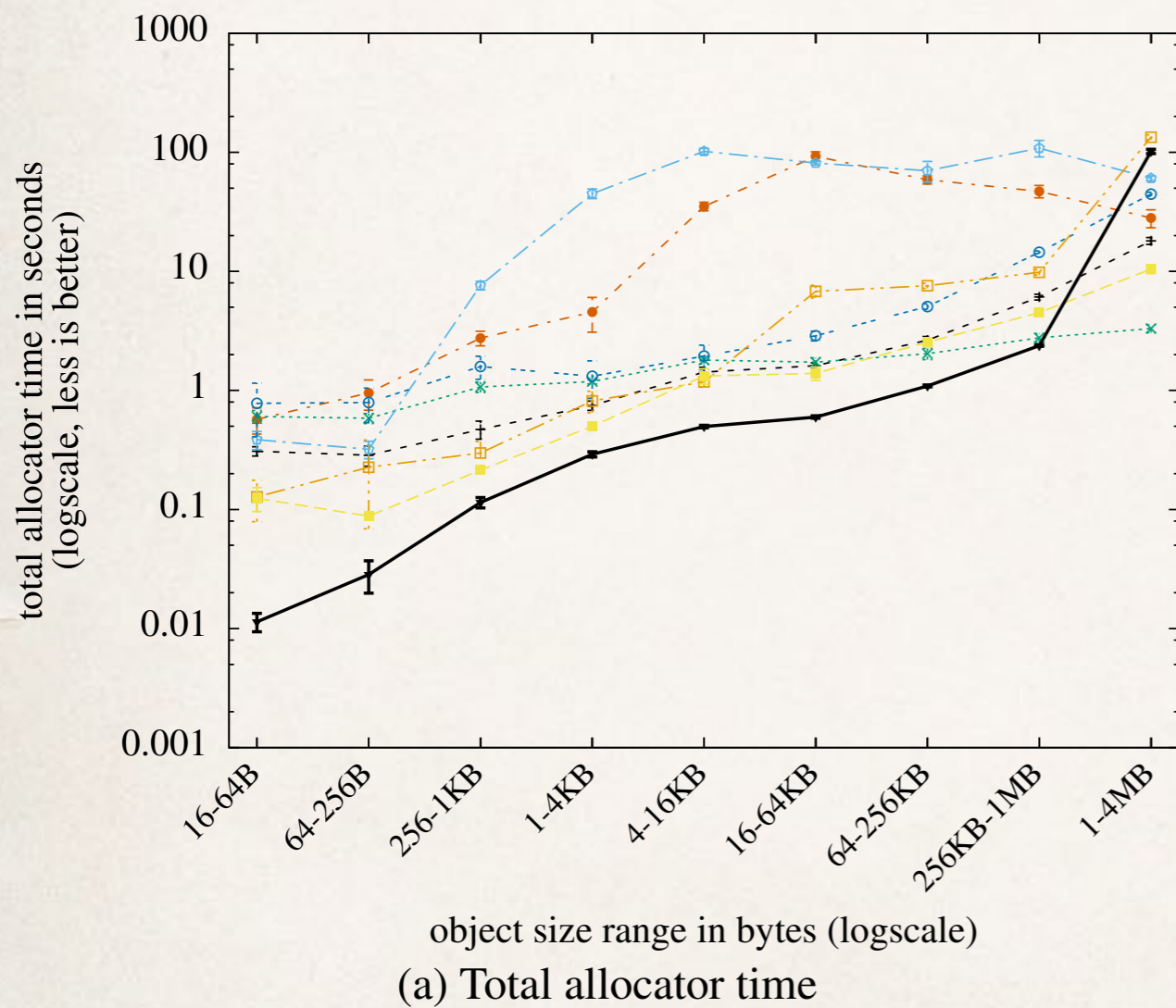


Figure 10: Temporal and spatial performance for the object-size robustness experiment at 40 threads

ACDC: Explorative Benchmarking Memory Management

acdc.cs.uni-salzburg.at [ISMM13,DLS14]

- ❖ configurable multicore-scalable mutator for mimicking virtually any allocation, deallocation, sharing, and access behavior
- ❖ written in C, tracks with minimal overhead:
 1. memory allocation time
 2. memory deallocation time
 3. memory consumption
 4. memory access time

Memory Access

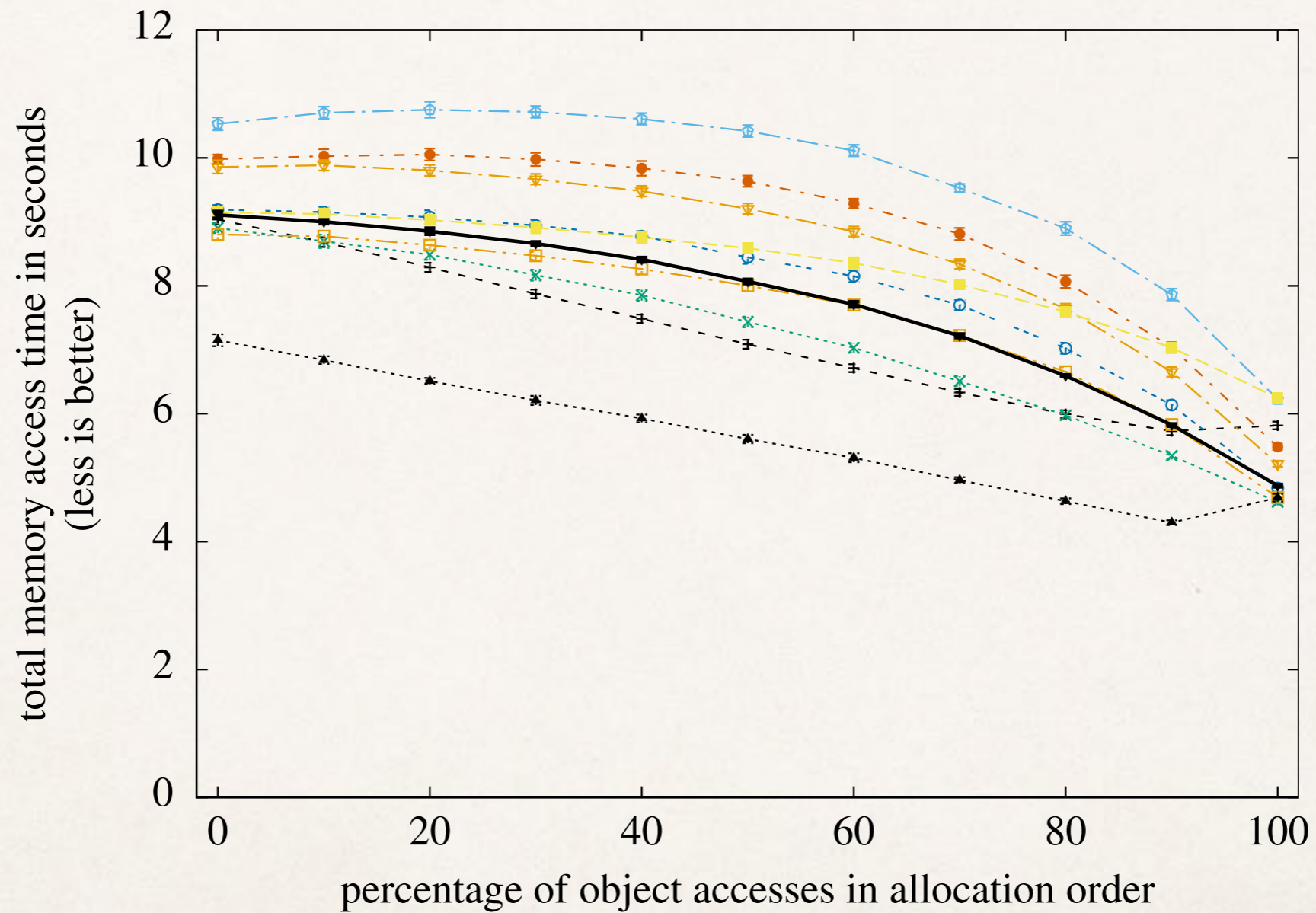


Figure 11: Memory access time for the locality experiment

How do we teach computer science to students not necessarily majoring in computer science but who anyway code every day?

–The Computer Science Education Challenge



Selfie: Teaching Computer Science

[selfie.cs.uni-salzburg.at]

- ❖ *Selfie* is a self-referential 7k-line C implementation (in a single file) of:
 1. a self-compiling compiler called *starc* that compiles a tiny subset of C called C Star (C*) to a tiny subset of MIPS32 called MIPSter,
 2. a self-executing emulator called *mipster* that executes MIPSter code including itself when compiled with *starc*,
 3. a self-hosting hypervisor called *hypster* that virtualizes *mipster* and can host all of *selfie* including itself, and
 4. a tiny C* library called *libcstar* utilized by all of *selfie*.

5 statements:
assignment
while
if
return
procedure()

```
int atoi(int *s) {  
    int i;  
    int n;  
    int c;  
  
    i = 0;  
    n = 0;  
    c = *(s+i);
```

no data structures,
just int and int*
and dereferencing:
the * operator

character literals
string literals

```
while (c != 0) {  
    n = n * 10 + c - '0';  
    if (n < 0)  
        return -1;
```

integer arithmetics
pointer arithmetics

```
    i = i + 1;  
    c = *(s+i);
```

no bitwise operators
no Boolean operators

```
return n;
```

library: exit, malloc, open, read, write

```
> make
```

```
cc -w -m32 -D'main(a,b)=main(a, char**argv)' selfie.c -o selfie
```

-bootstrapping selfie using standard C compiler


```
> ./selfie
```

```
./selfie: usage: selfie { -c { source } | -o binary | -s assembly  
| -l binary } [ ( -m | -d | -y | -min | -mob ) size ... ]
```

-selfie usage

```
> ./selfie -c selfie.c
```

```
./selfie: this is selfie's starc compiling selfie.c
```

```
./selfie: 176408 characters read in 7083 lines and 969 comments  
./selfie: with 97779(55.55%) characters in 28914 actual symbols  
./selfie: 261 global variables, 289 procedures, 450 string literals  
./selfie: 1958 calls, 723 assignments, 57 while, 572 if, 243 return  
./selfie: 121660 bytes generated with 28779 instructions and 6544  
bytes of data
```

-compiling selfie with selfie (takes seconds)


```
> ./selfie -c selfie.c -m 2 -c selfie.c
```

```
./selfie: this is selfie's starc compiling selfie.c
```

```
./selfie: this is selfie's mipster executing selfie.c with 2MB of  
physical memory
```

```
selfie.c: this is selfie's starc compiling selfie.c
```

```
selfie.c: exiting with exit code 0 and 1.05MB of mallocated memory
```

```
./selfie: this is selfie's mipster terminating selfie.c with exit code  
0 and 1.16MB of mapped memory
```

*–compiling selfie with selfie and then running that executable to
compile selfie again (takes ~6 minutes)*

```
> ./selfie -c selfie.c -o selfie1.m -m 2 -c selfie.c -o selfie2.m
```

```
./selfie: this is selfie's starc compiling selfie.c
```

```
./selfie: 121660 bytes with 28779 instructions and 6544 bytes of data  
written into selfie1.m
```

```
./selfie: this is selfie's mipster executing selfie1.m with 2MB of  
physical memory
```

```
selfie1.m: this is selfie's starc compiling selfie.c
```

```
selfie1.m: 121660 bytes with 28779 instructions and 6544 bytes of data  
written into selfie2.m
```

```
selfie1.m: exiting with exit code 0 and 1.05MB of mallocated memory
```

```
./selfie: this is selfie's mipster terminating selfie1.m with exit  
code 0 and 1.16MB of mapped memory
```

*–compiling selfie with selfie and generating an executable selfie1.m
that is then executed to compile selfie again generating another
executable selfie2.m (takes ~6 minutes)*


```
> ./selfie -c selfie.c -m 2 -c selfie.c -m 2 -c selfie.c
```

–compiling selfie with selfie and then running that executable to compile selfie again and then running that executable to compile selfie again (takes ~24 hours)

```
> ./selfie -c selfie.c -m 2 -c selfie.c (-y 2) -c selfie.c
```

*–compiling selfie with selfie and then running that executable to compile selfie again and then **hosting** that executable in a virtual machine to compile selfie again (takes ~12 minutes)*





Thank you!