RiSE
Rigorous Systems Engineering

RiSE
Rigorous Systems Engineering

Semantics of Concurrent Data Structures

Ana Sokolova UNIVERSITY of SALZBURG

AVM, 25.9.2018

Concurrent data structures

correctness and performance

structure and power

via semantic relaxations

* New results enabling

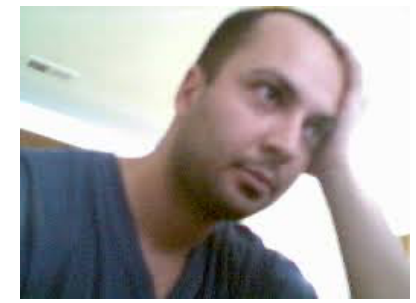
verifying linearizability

Concurrent Data Structures Correctness and Relaxations

Hannes Payer
Google

Tom Henzinger
IST AUSTRIA

Christoph Kirsch
UNIVERSITY of SALZBURG



Ali Sezgin UNIVERSITY OF CAMBRIDGE

Andreas Haas Google

Michael Lippautz

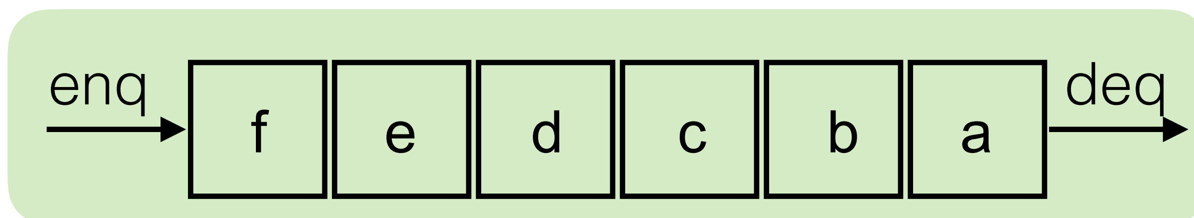
Andreas Holzer
Google

Helmut Veith TU WIEN

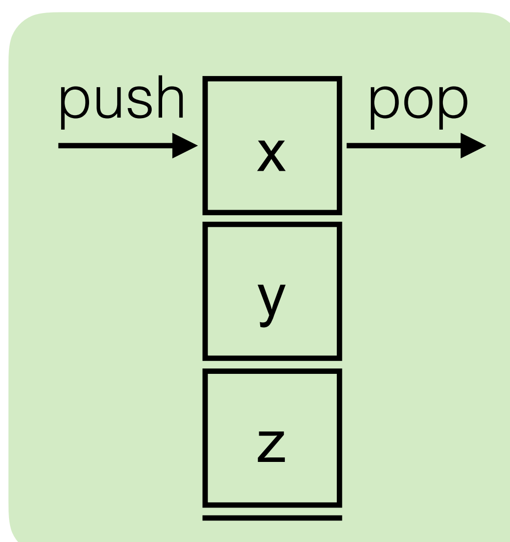
RiSE
Rigorous Systems Engineering

Data structures

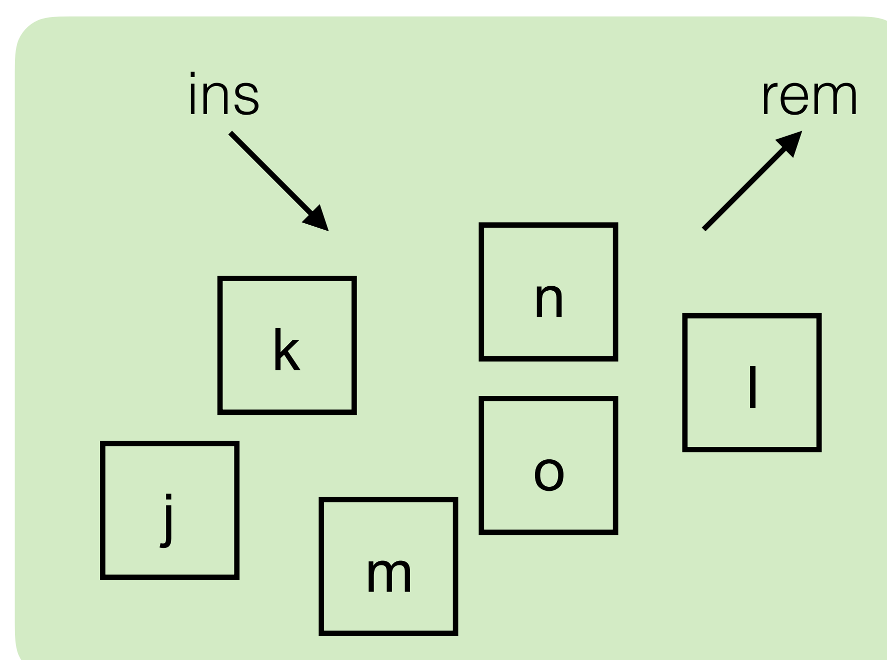
- Queue FIFO



- Stack LIFO

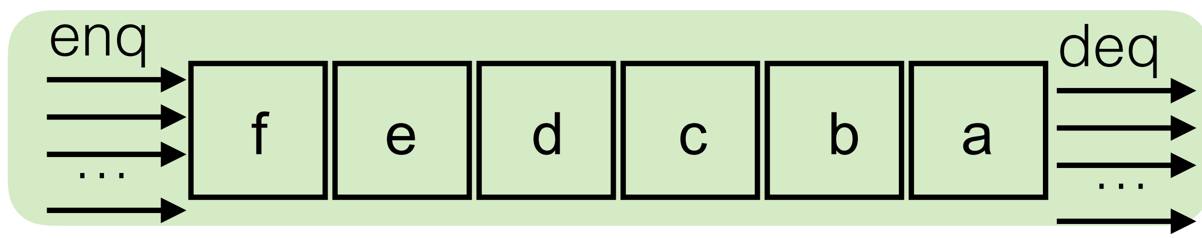


- Pool unordered

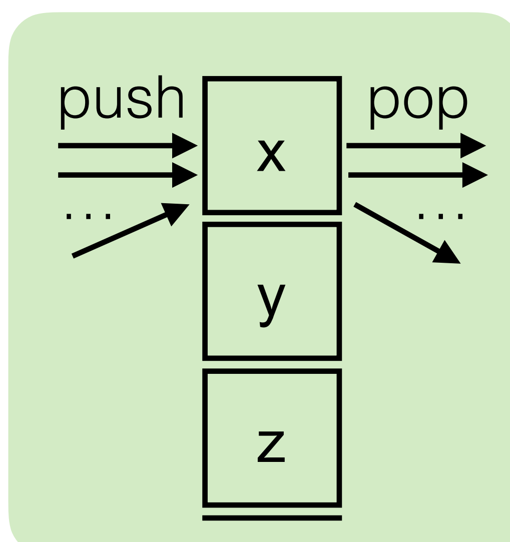


Concurrent data structures

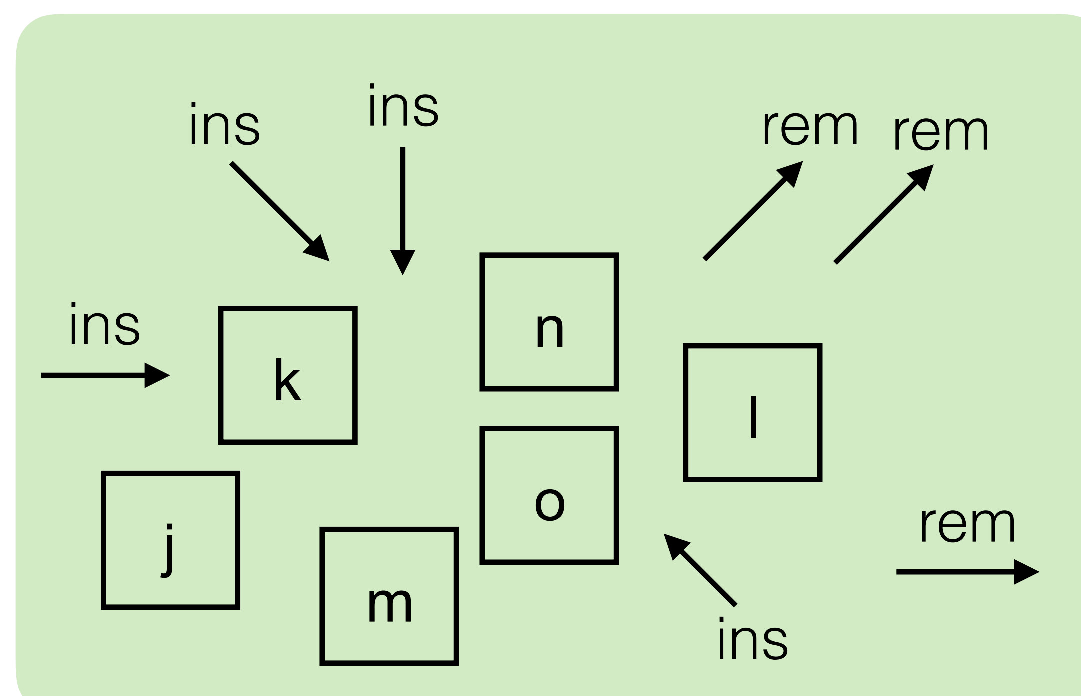
- Queue FIFO



- Stack LIFO



- Pool unordered



Semantics of concurrent data structures

t1: enq(2) deq(1)
t2: enq(1) deq(2)

e.g. queues

- Sequential specification = set of legal sequences

e.g. queue legal sequence
enq(1)enq(2)deq(1)deq(2)

- Consistency condition = e.g. linearizability / sequential consistency

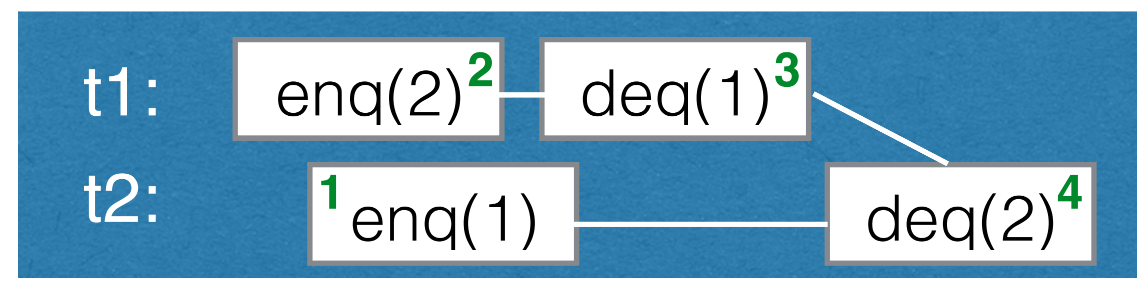
e.g. the concurrent history above is a linearizable queue concurrent history

Consistency conditions

A history is … wrt a sequential specification iff

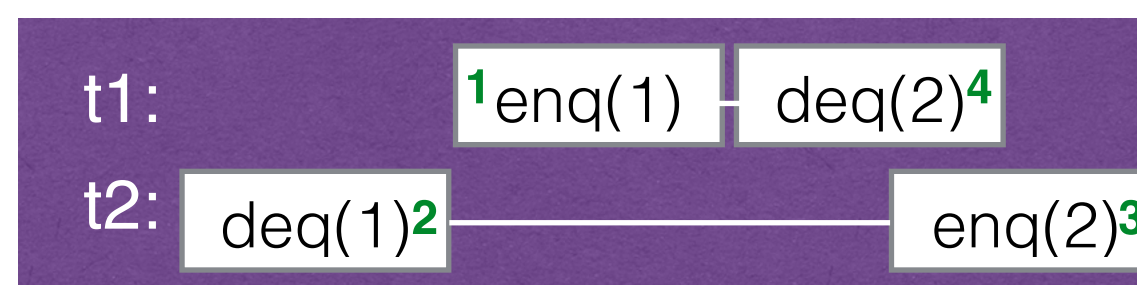
there exists a legal sequence that preserves precedence order

Linearizability [Herlihy,Wing '90]



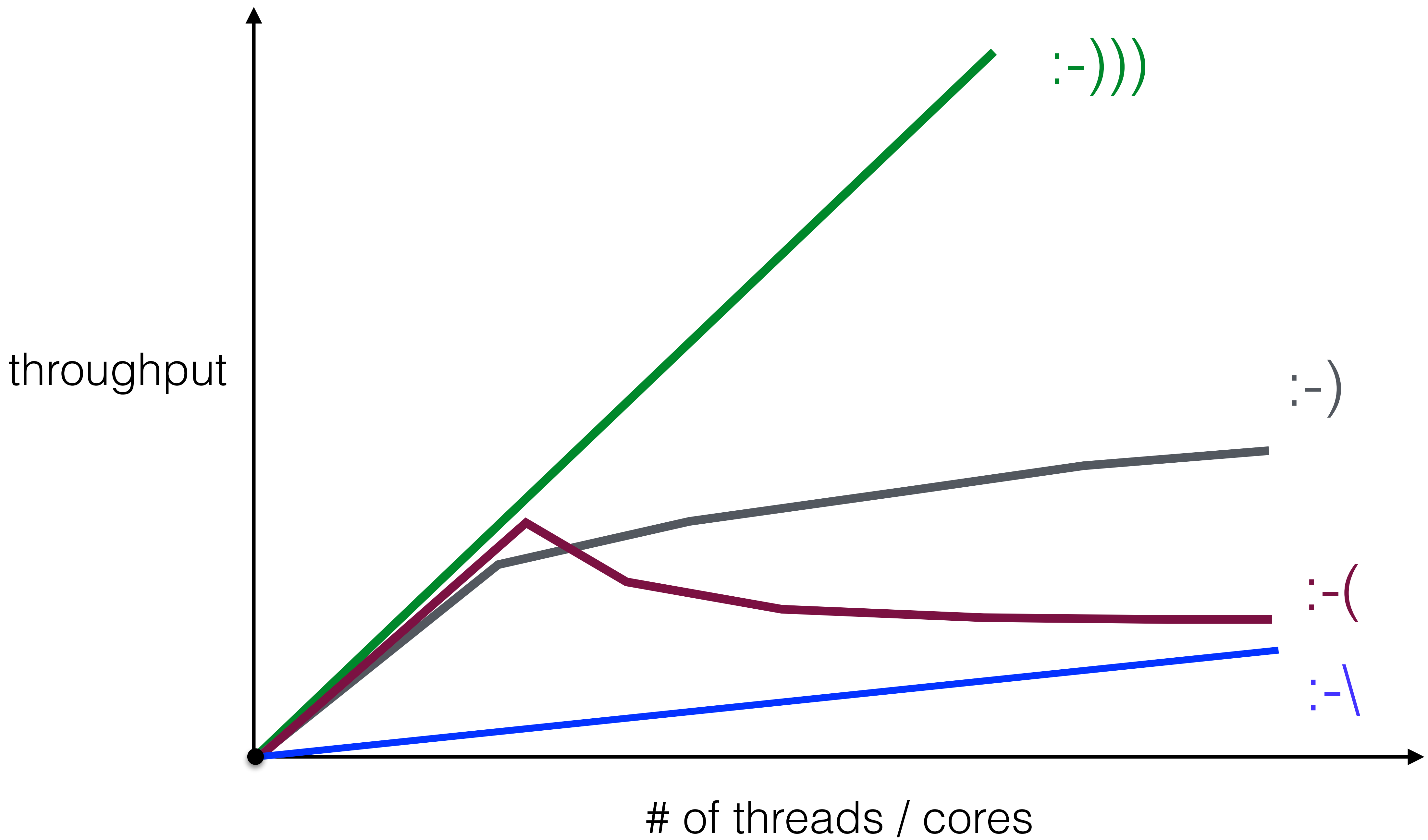
consistency is about extending partial orders to total orders

Sequential Consistency [Lamport'79]



there exists a legal sequence that preserves per-thread precedence (program order)

Performance and scalability



Relaxations allow trading

correctness
for
performance

provide the potential
for better-performing
implementations

Relaxing the Semantics

Quantitative relaxations
Henzinger, Kirsch, Payer, Sezgin,S. POPL13

- Sequential specification = set of legal sequences
- Consistency condition = e.g. linearizability / sequential consistency

Local linearizability
Haas, Henzinger, Holzer,…, S, Veith CONCUR16

Relaxing the Sequential Specification

Quantitative Relaxations (POPL13)

Goal

Stack - incorrect behavior

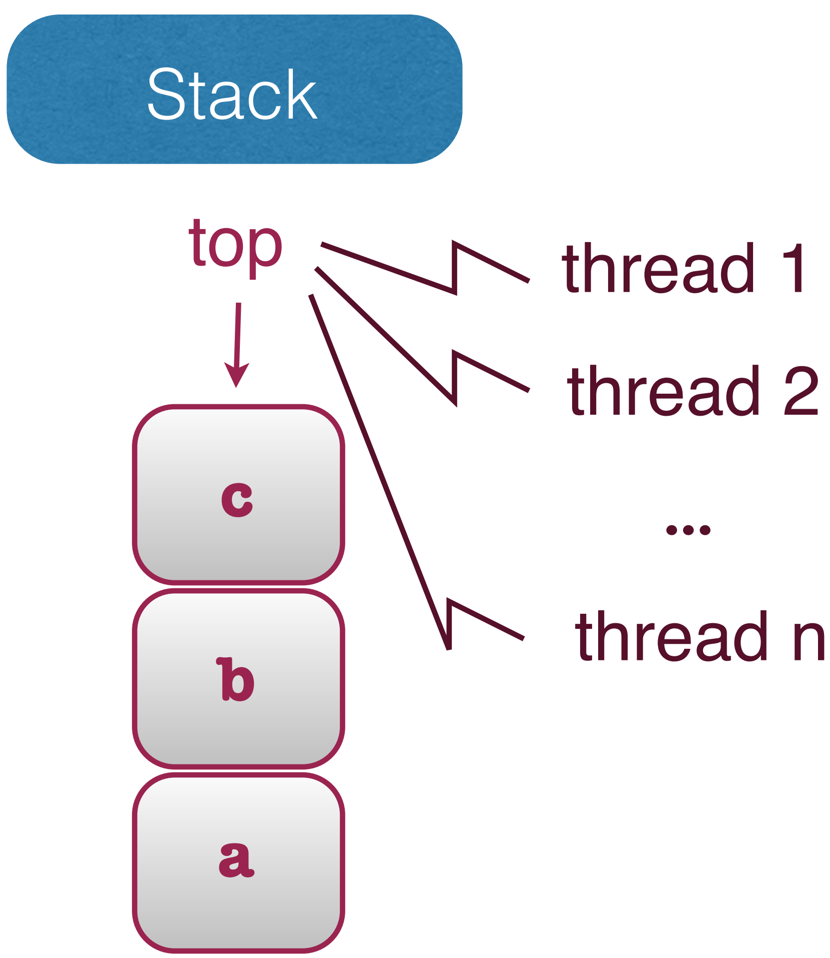
push(a)push(b)push(c)pop(a)pop(b)

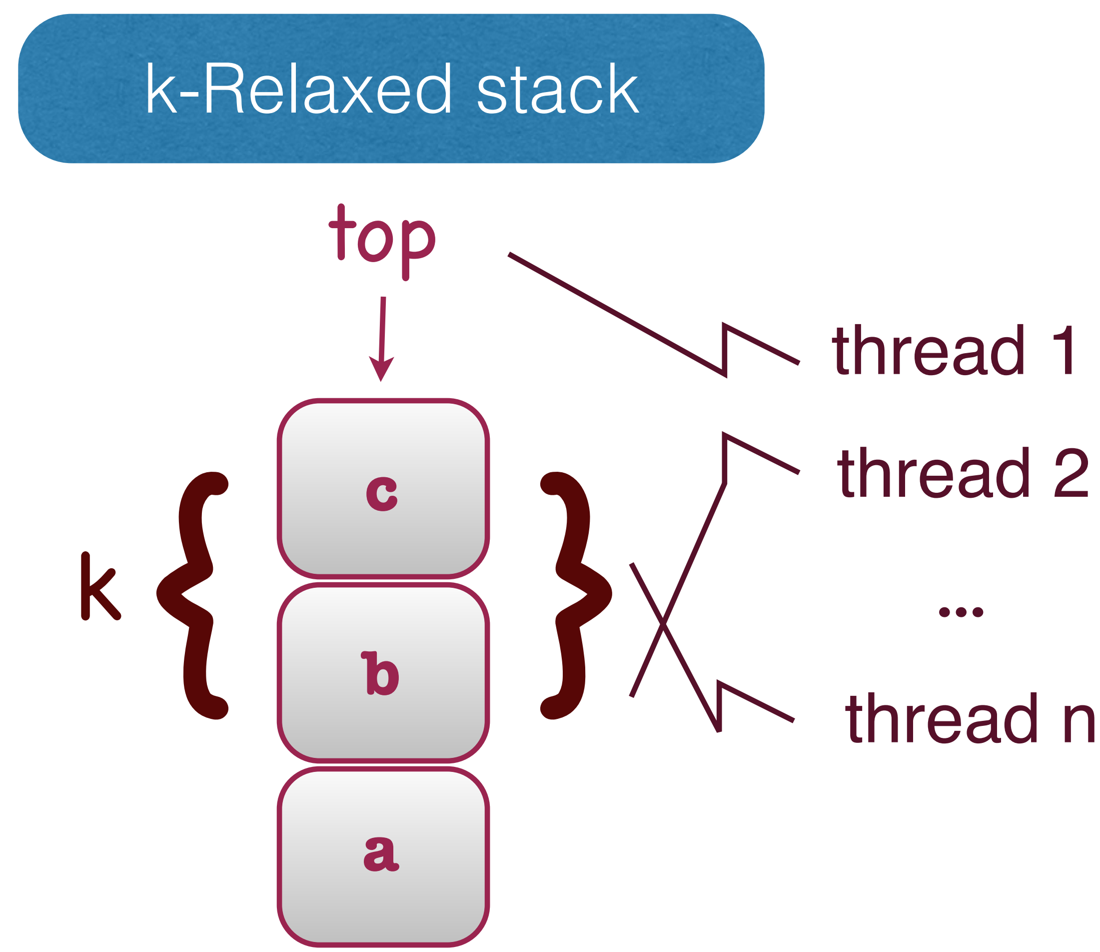
- trade correctness for performance
- in a controlled way with quantitative bounds

measure the error from correct behaviour

correct in a relaxed stack
... 2-relaxed? 3-relaxed?

How can relaxing help?





We have got

- Framework
- Generic examples
- Concrete relaxation examples
- Efficient concurrent implementations

for semantic relaxations

out-of-order / stuttering

stacks, queues, priority queues,.. / CAS, shared counter

of relaxation instances

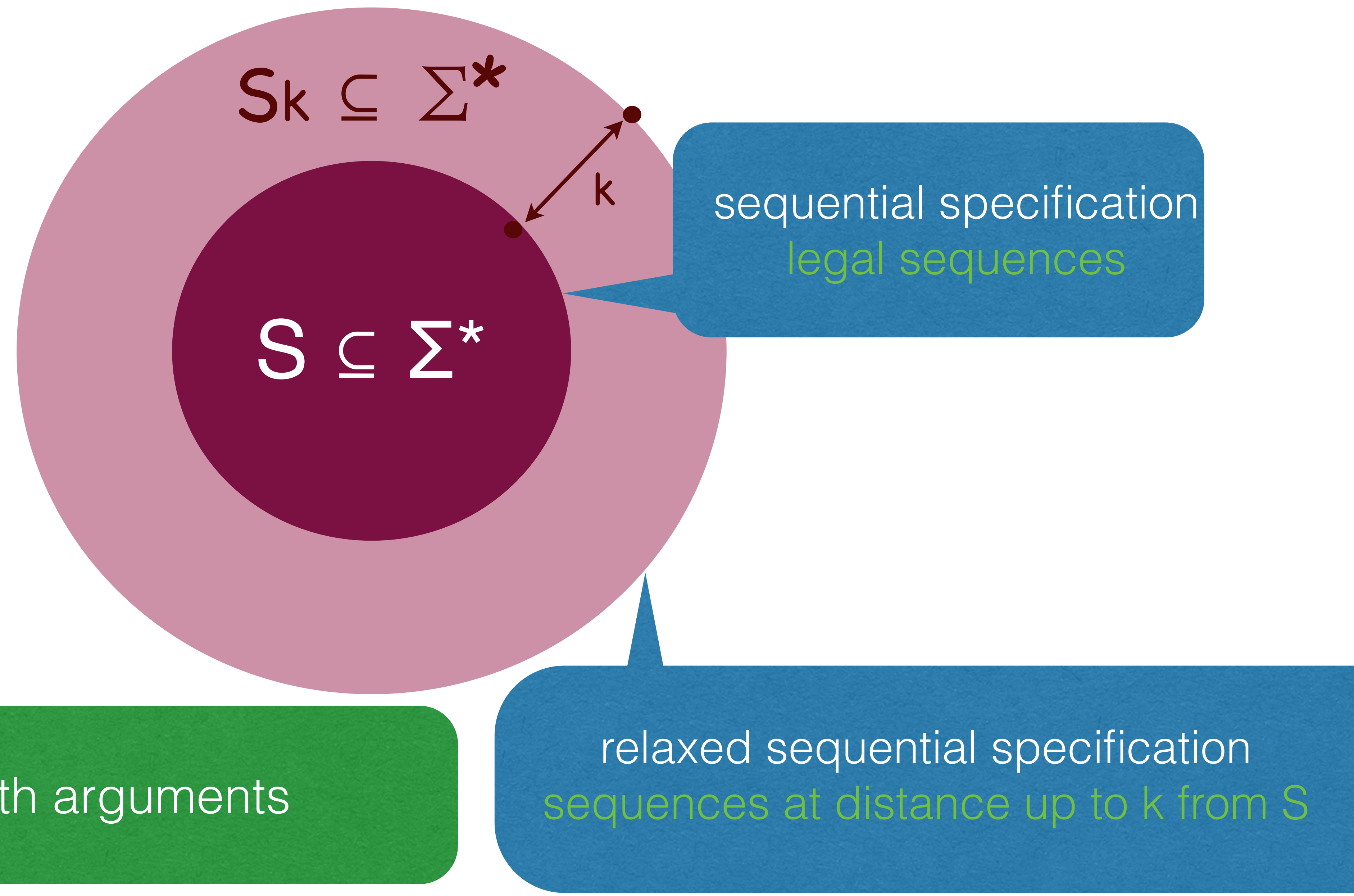
The big picture

$S \subseteq \Sigma^*$

sequential specification
legal sequences

Σ - methods with arguments

The big picture



Σ - methods with arguments

Relaxing the Consistency Condition

Local Linearizability (CONCUR16)

Local Linearizability main idea

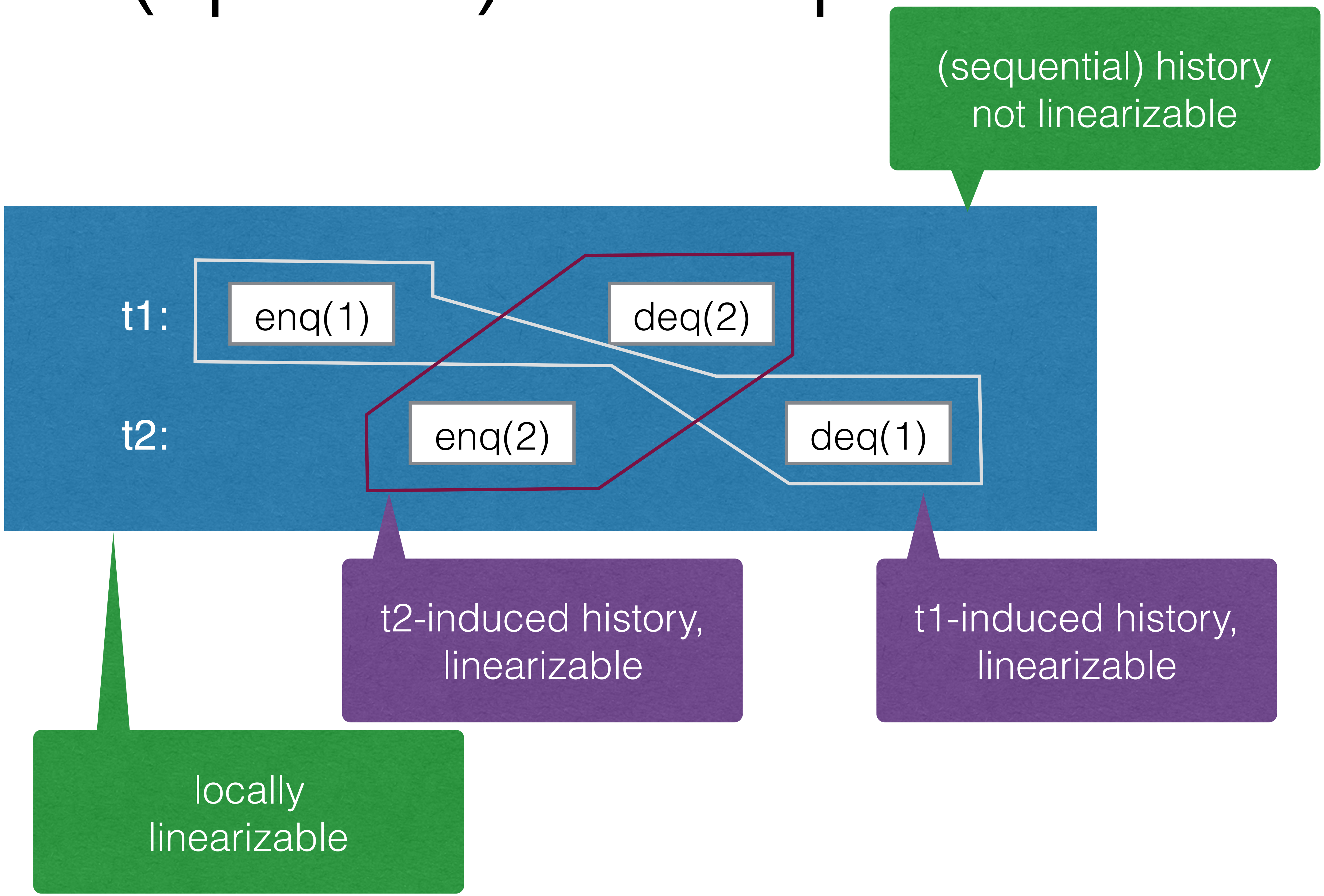
Already present in some shared-memory consistency conditions (not in our form of choice)

- Partition a history into a set of local histories
- Require linearizability per local history

no global witness

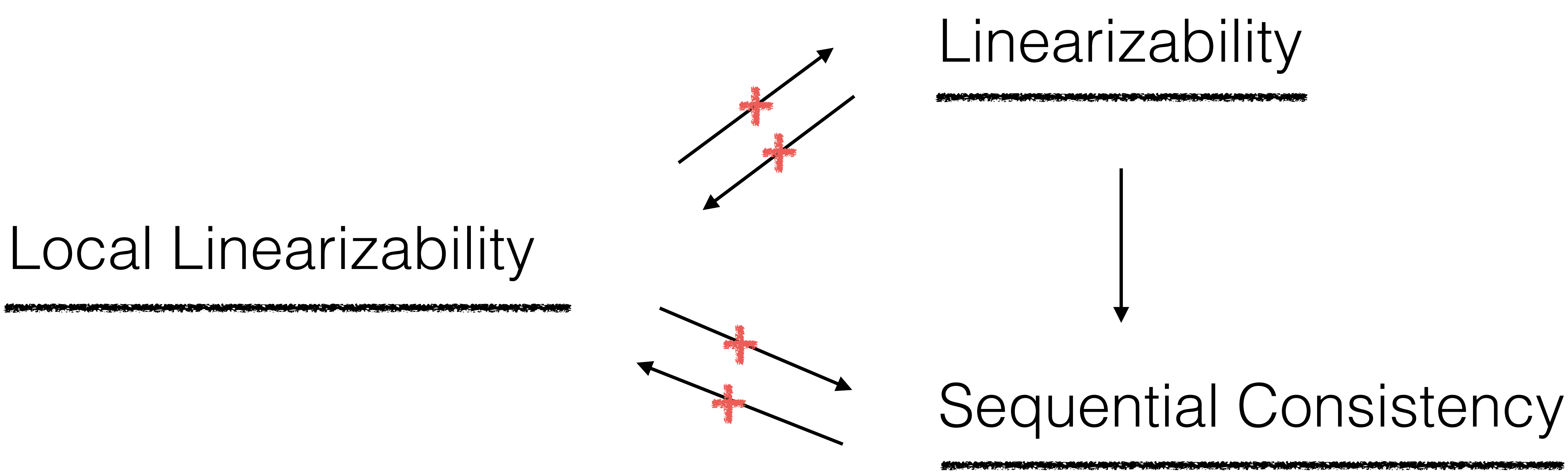
Local sequential consistency… is also possible

Local Linearizability (queue) example



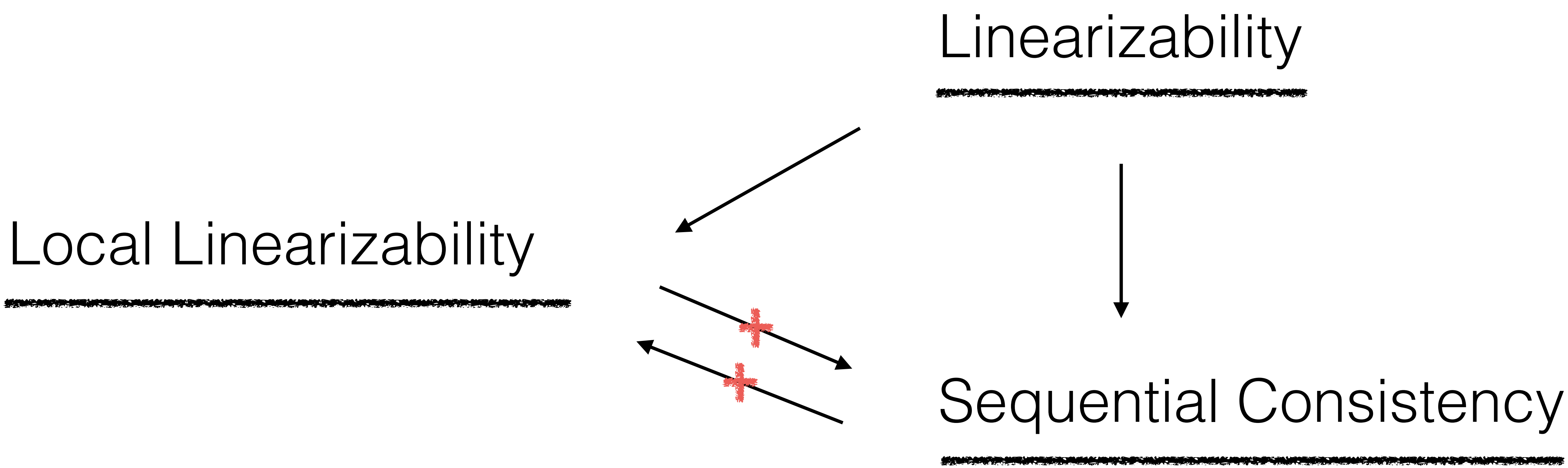
Where do we stand?

In general



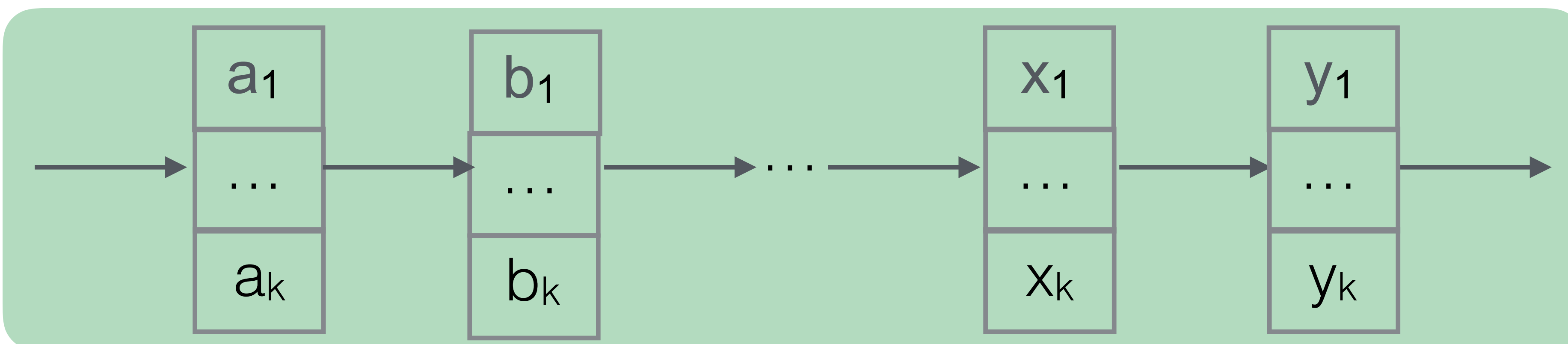
Where do we stand?

For queues (and most container-type data structures)



Lead to scalable implementations

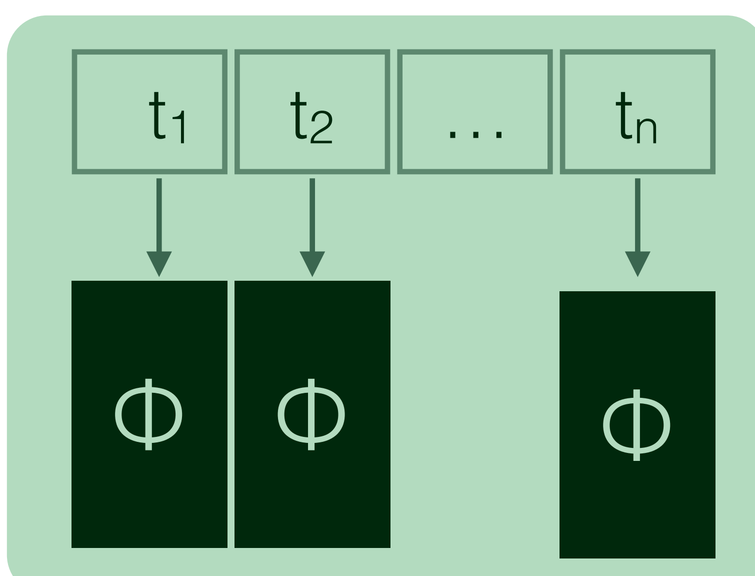
e.g. k-FIFO, k-Stack



k-out-of-order queue

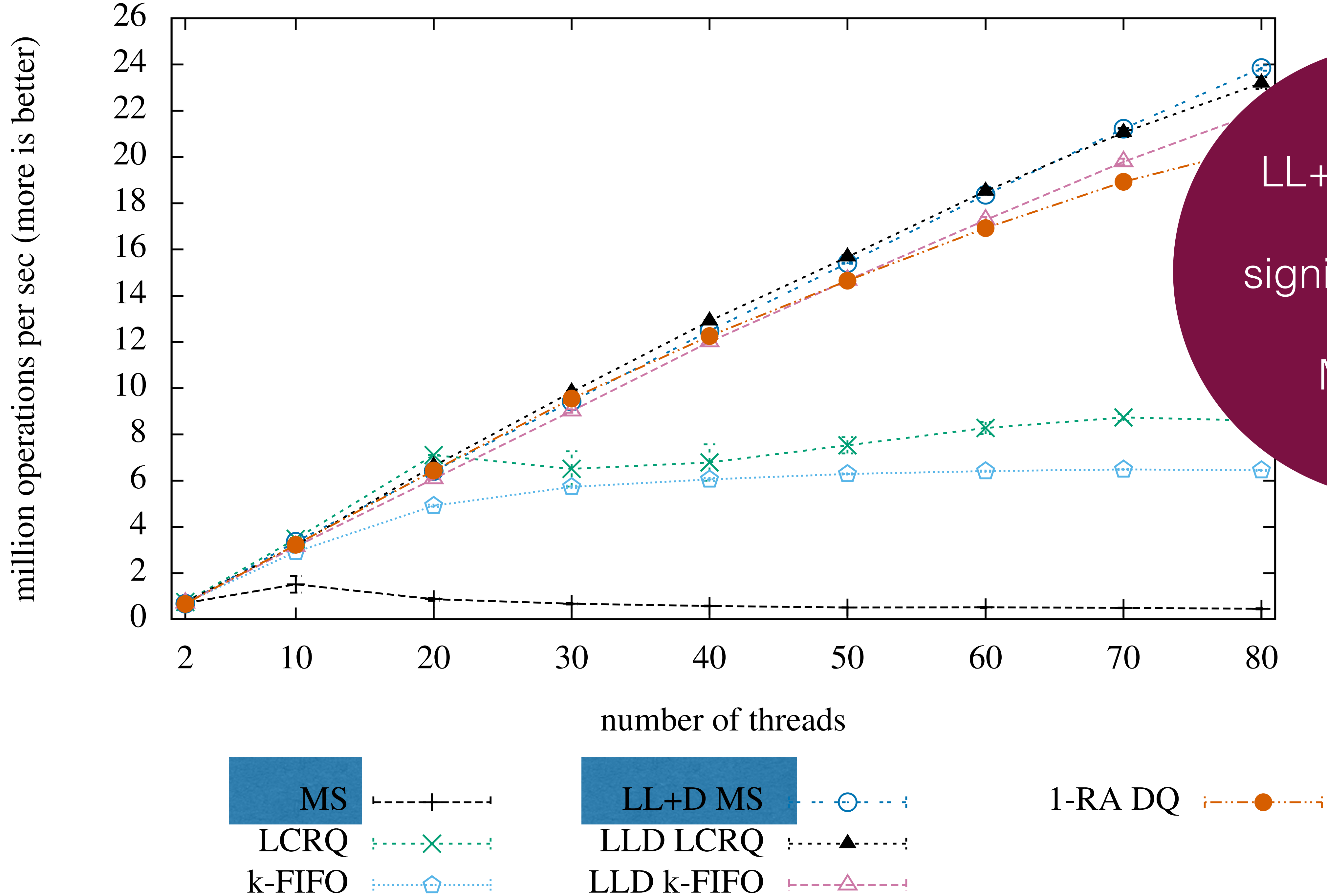
locally linearizable distributed implementation

LLD Φ
LL+D Φ



local inserts / global removes

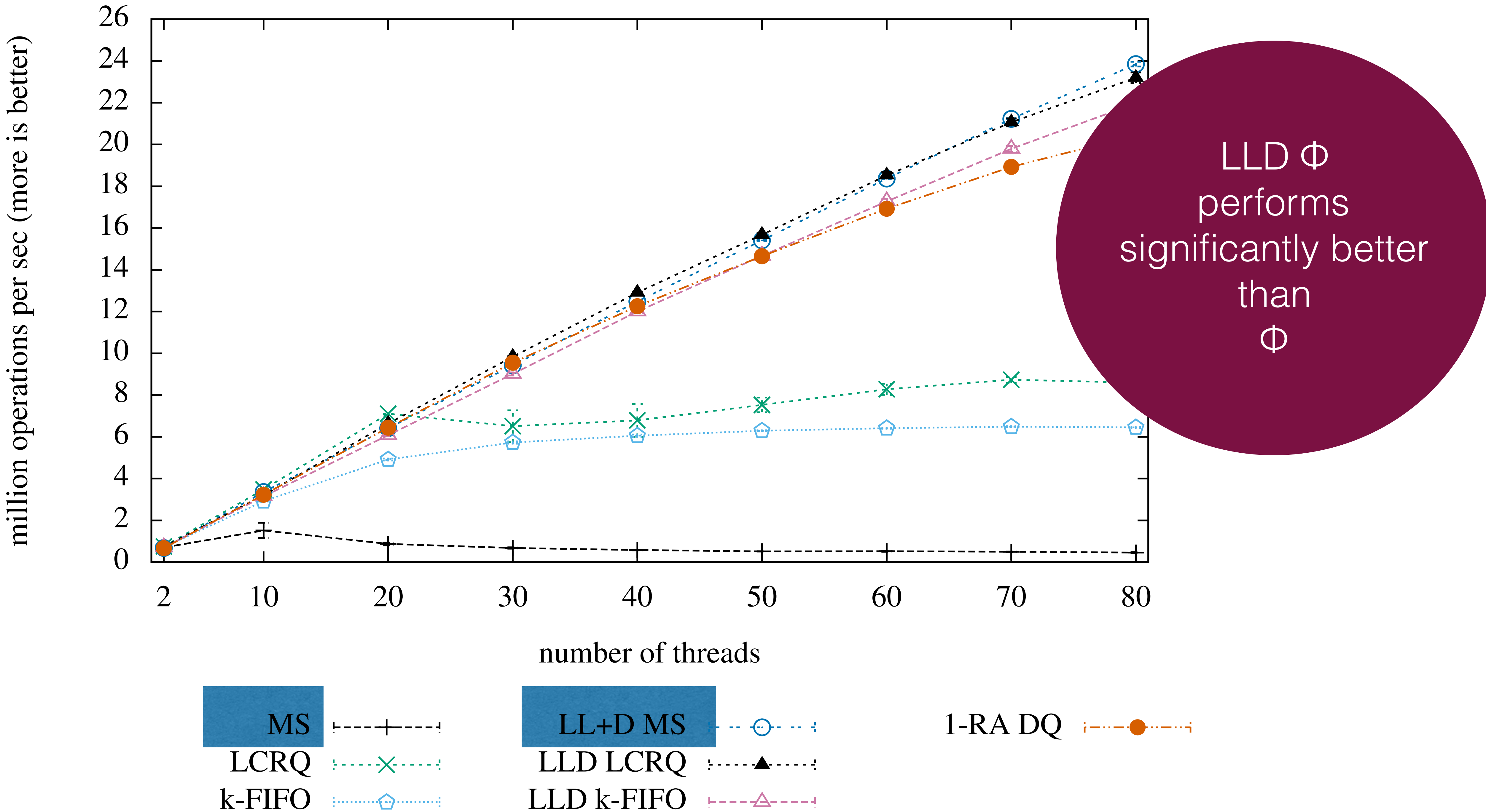
Performance



(a) Queues, LL queues, and "queue-like" pools

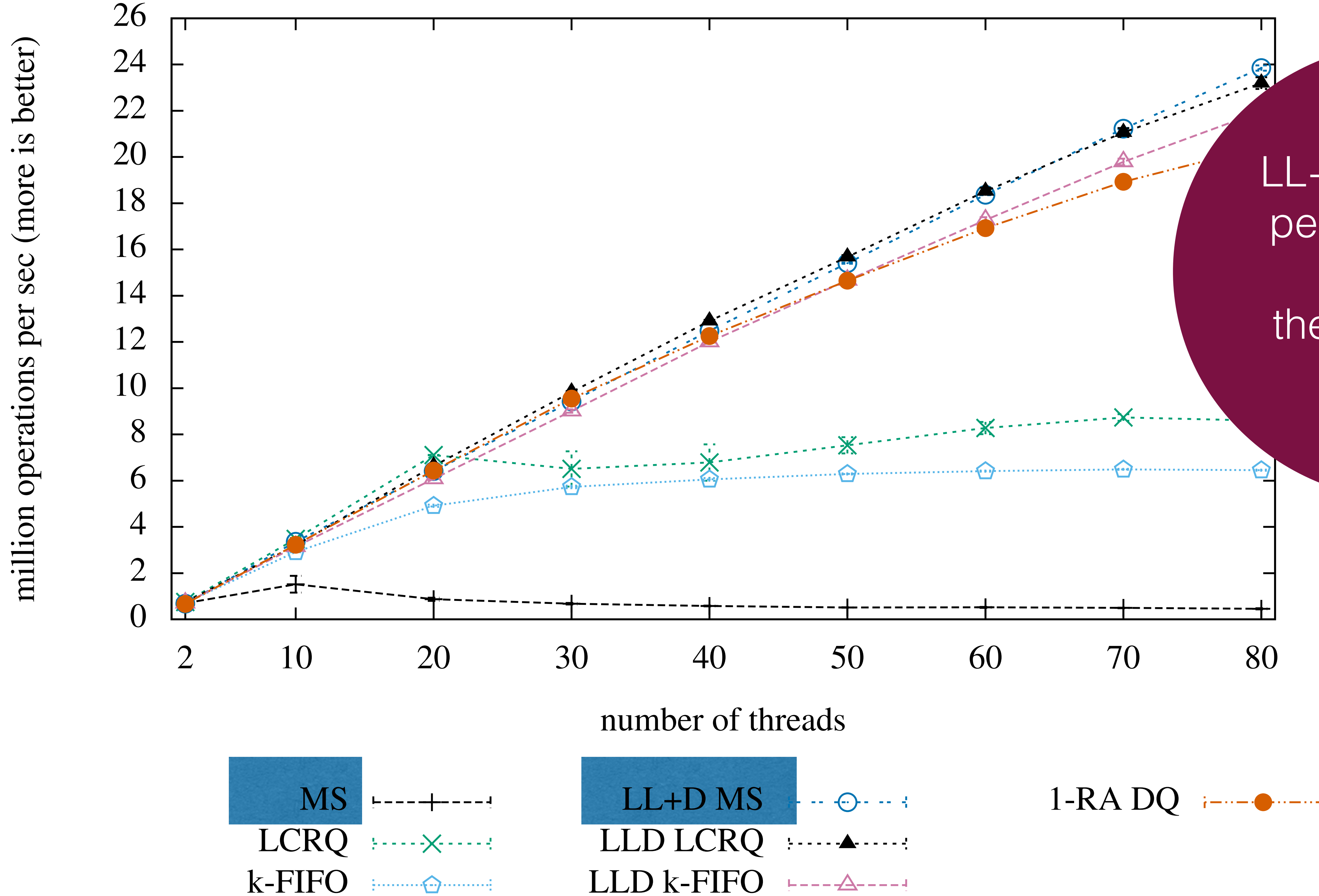
LL+D MS queue performs significantly better than MS queue

Performance



(a) Queues, LL queues, and "queue-like" pools

Performance



(a) Queues, LL queues, and “queue-like” pools

LL+D MS queue performs better than the best known pools scal.cs.uni-salzburg.at

Scal

High-Performance Multicore-Scalable Computing

We study the design, implementation, performance, and scalability of concurrent objects on multicore systems by analyzing the apparent trade-off between adherence to concurrent data structure semantics and scalability.

Thank You !

scal.cs.uni-salzburg.at

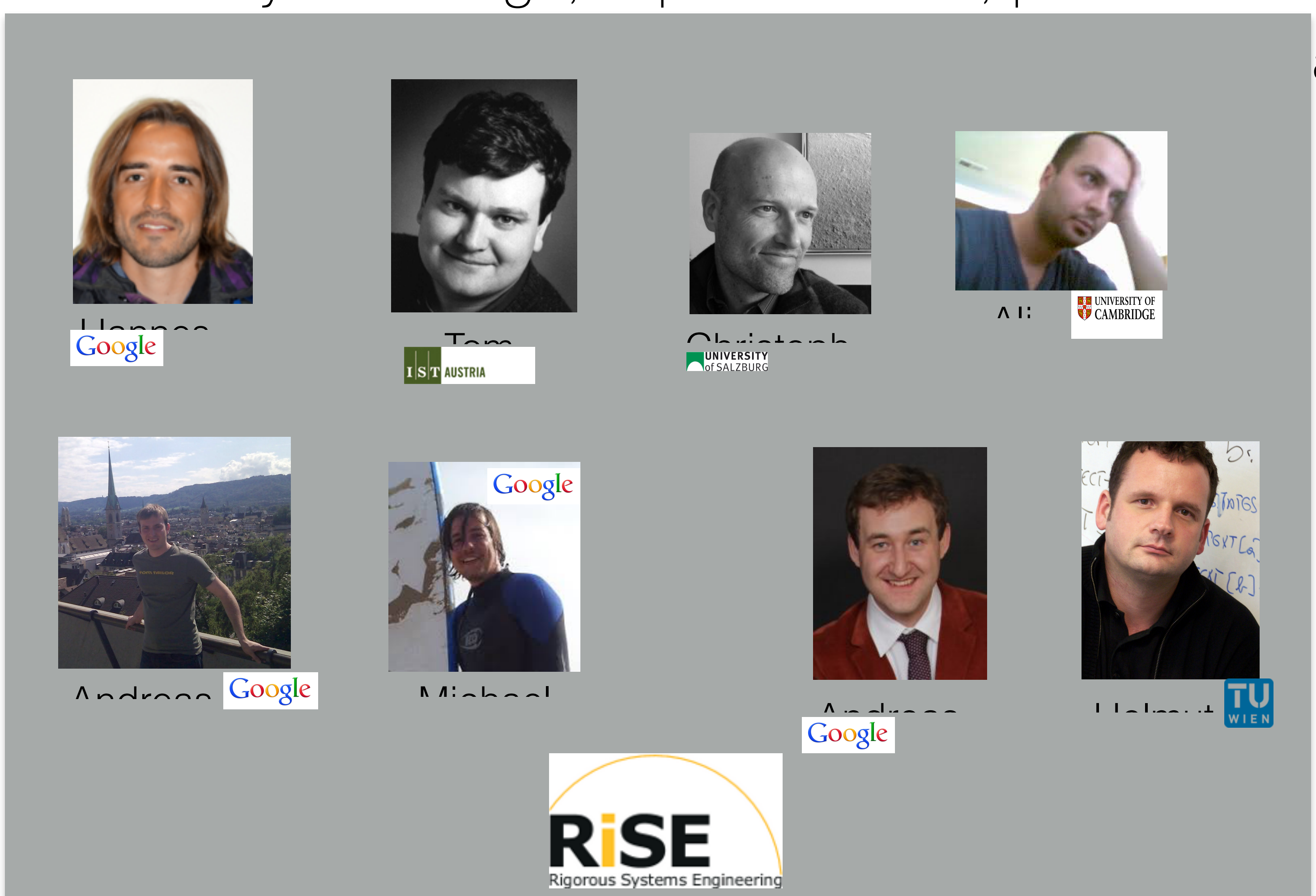
Scal

High-Performance Multicore-Scalable Computing

We study the design, implementation, performance, and scalability of co

arent trade-off between

and scalability.



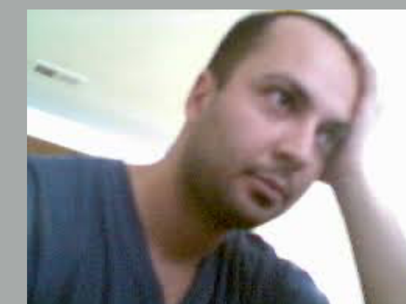
Thank You !

Concurrent Data Structures
Correctness and Performance

Hannes Payer
Google

Tom Henzinger
IST AUSTRIA

Christoph Kirsch
UNIVERSITY of SALZBURG



Ali Sezgin
UNIVERSITY OF CAMBRIDGE

Andreas Haas
Google

Michael Lippautz

Andreas Holzer
Google

Helmut Veith
TU WIEN

RiSE
Rigorous Systems Engineering

Thank You !

Linearizability via Order Extension Theorems

joint work with

Harald Woracek

foundational results
for
verifying linearizability

Inspiration

As well as
Reducing Linearizability to
State Reachability
[Bouajjani, Emmi, Enea, Hamza]
ICALP15 + …

Queue sequential specification (axiomatic)

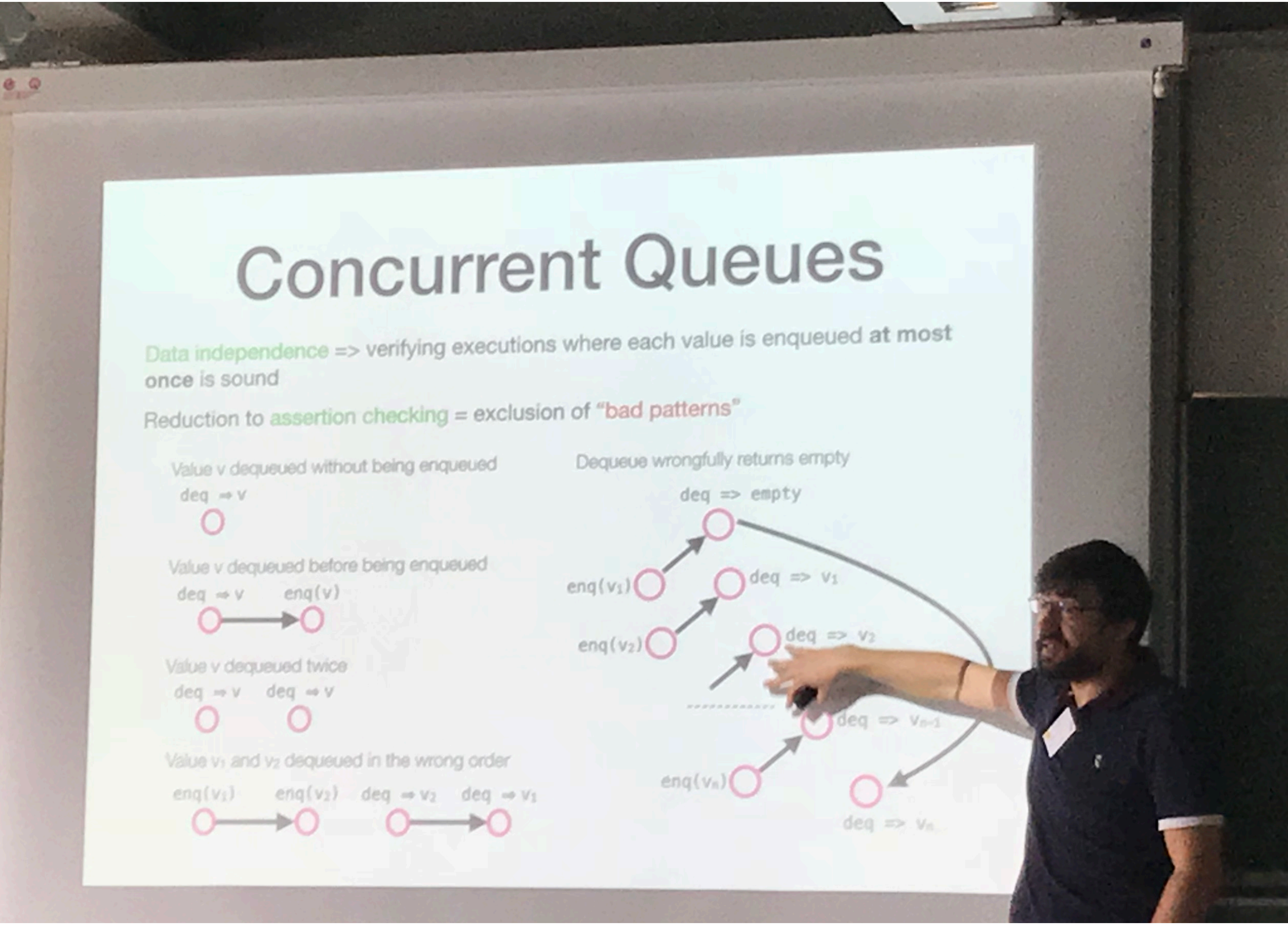
s is a legal queue sequence
iff
1. **s** is a legal pool sequence, and
2. enq(x) $<_{\mathbf{s}}$ enq(y) $\wedge$ deq(y) $\in$ **s** $\Rightarrow$ deq(x) $\in$ **s** $\wedge$ deq(x) $<_{\mathbf{s}}$ deq(y)

Queue linearizability (axiomatic)

Henzinger, Sezgin, Vafeiadis CONCUR13

h is queue linearizable
iff
1. **h** is pool linearizable, and
2. enq(x) $<_{\mathbf{h}}$ enq(y) $\wedge$ deq(y) $\in$ **h** $\Rightarrow$ deq(x) $\in$ **h** $\wedge$ deq(y) $\not<_{\mathbf{h}}$ deq(x)

precedence order


Concurrent Queues
Data independence => verifying executions where each value is enqueued at most once is sound
Reduction to assertion checking = exclusion of "bad patterns"
Value v dequeued without being enqueued
deq ⇒ v
Value v dequeued before being enqueued
deq ⇒ v
enq(v)
Value v dequeued twice
deq ⇒ v
deq ⇒ v
Value v_1 and v_2 dequeued in the wrong order
enq(v_1)
enq(v_2)
deq ⇒ v_2
deq ⇒ v_1
Dequeue wrongfully returns empty
deq => empty
enq(v_1)
deq => v_1
enq(v_2)
deq => v_2
deq => v_{n-1}
enq(v_n)
deq => v_n

Problems (stack)

Stack sequential specification (axiomatic)

s is a legal stack sequence

iff

1. **s** is a legal pool sequence, and
2. push(x) $<_s$ push(y) $<_s$ pop(x) $\Rightarrow$ pop(y) $\in$ **s** $\wedge$ pop(y) $<_s$ pop(x)

Stack linearizability (axiomatic)

h is stack linearizable

iff

1. **h** is pool linearizable, and
2. push(x) $<_h$ push(y) $<_h$ pop(x) $\Rightarrow$ pop(y) $\in$ **h** $\wedge$ pop(x) $\not<_h$ pop(y)

???

Problems (stack)

Stack sequential specification (axiomatic)

s is a legal stack sequence

iff

1. **s** is a legal pool sequence, and
2. push(x) $<_{\mathbf{s}}$ push(y) $<_{\mathbf{s}}$ pop(x) $\Rightarrow$ pop(y) $\in$ **s** $\wedge$ pop(y) $<_{\mathbf{s}}$ pop(x)

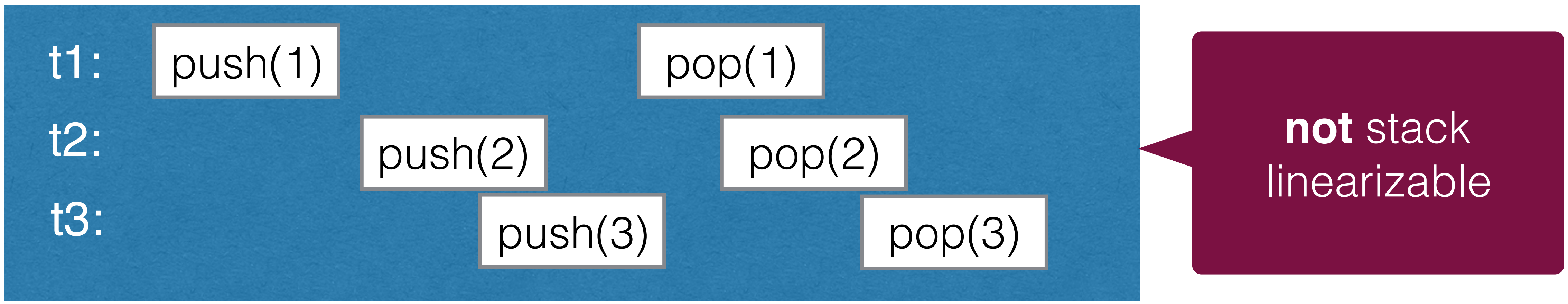
Stack linearizability (axiomatic)

~~**h** is stack linearizable~~

~~iff~~

1. ~~**h** is pool linearizable, and~~
2. ~~push(x) $<_{\mathbf{h}}$ push(y) $<_{\mathbf{h}}$ pop(x)~~ $\Rightarrow$ pop(y) $\in$ **h** $\wedge$ pop(x) $\not<_{\mathbf{h}}$ pop(y)

Problems (stack)



Stack linearizability (axiomatic)

h is stack linearizable

iff

1. **h** is pool linearizable, and
2. $push(x) <_{\mathbf{h}} push(y) <_{\mathbf{h}} pop(x) \Rightarrow pop(y) \in \mathbf{h} \wedge pop(x) \not<_{\mathbf{h}} pop(y)$

Linearizability verification

Data structure

- signature Σ - set of method calls including data values
- sequential specification $S \subseteq \Sigma^*$, prefix closed

identify sequences with total orders

Sequential specification via violations

Extract a set of violations V, relations on Σ, such that **s** ∈ S iff **s** has no violations

it is easy to find a large CV,
but difficult to find a small representative

$\mathcal{P}(\mathbf{s}) \cap V = \varnothing$

Linearizability verification

Find a set of violations CV such that: every interval order with no CV violations extends to a total order with no V violations.

we build CV iteratively from V

legal sequence

concurrent history

It works for

- Pool without empty removals
- Queue without empty removals
- Priority queue without empty removals
- Pool
- Queue
- Priority que

Thank You !

But not yet for Stack:
infinite CV violations
without clear
inductive structure

Exploring the space of
data structures
as well as new ideas
for problematic cases