Local Linearizability

Ana Sokolova UNIVERSITY of SALZBURG

joint work with:

Andreas Haas UNIVERSITY of SALZBURG
Andreas Holzer UNIVERSITY OF TORONTO
Michael Lippautz UNIVERSITY of SALZBURG
Ali Sezgin UNIVERSITY OF CAMBRIDGE
Tom Henzinger IST AUSTRIA
Christoph Kirsch UNIVERSITY of SALZBURG
Hannes Payer Google
Helmut Veith TU WIEN

Concurrent Data Structures
Correctness and Performance

Semantics of concurrent data structures

t1: enq(2) deq(1)

t2: enq(1) deq(2)

e.g. pools, queues, stacks

- 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

there exists a sequential witness that preserves precedence

Linearizability [Herlihy,Wing '90]

t1:	enq(2)[2]	deq(1)[3]		
t2:	[1]enq(1)		deq(2)[4]	

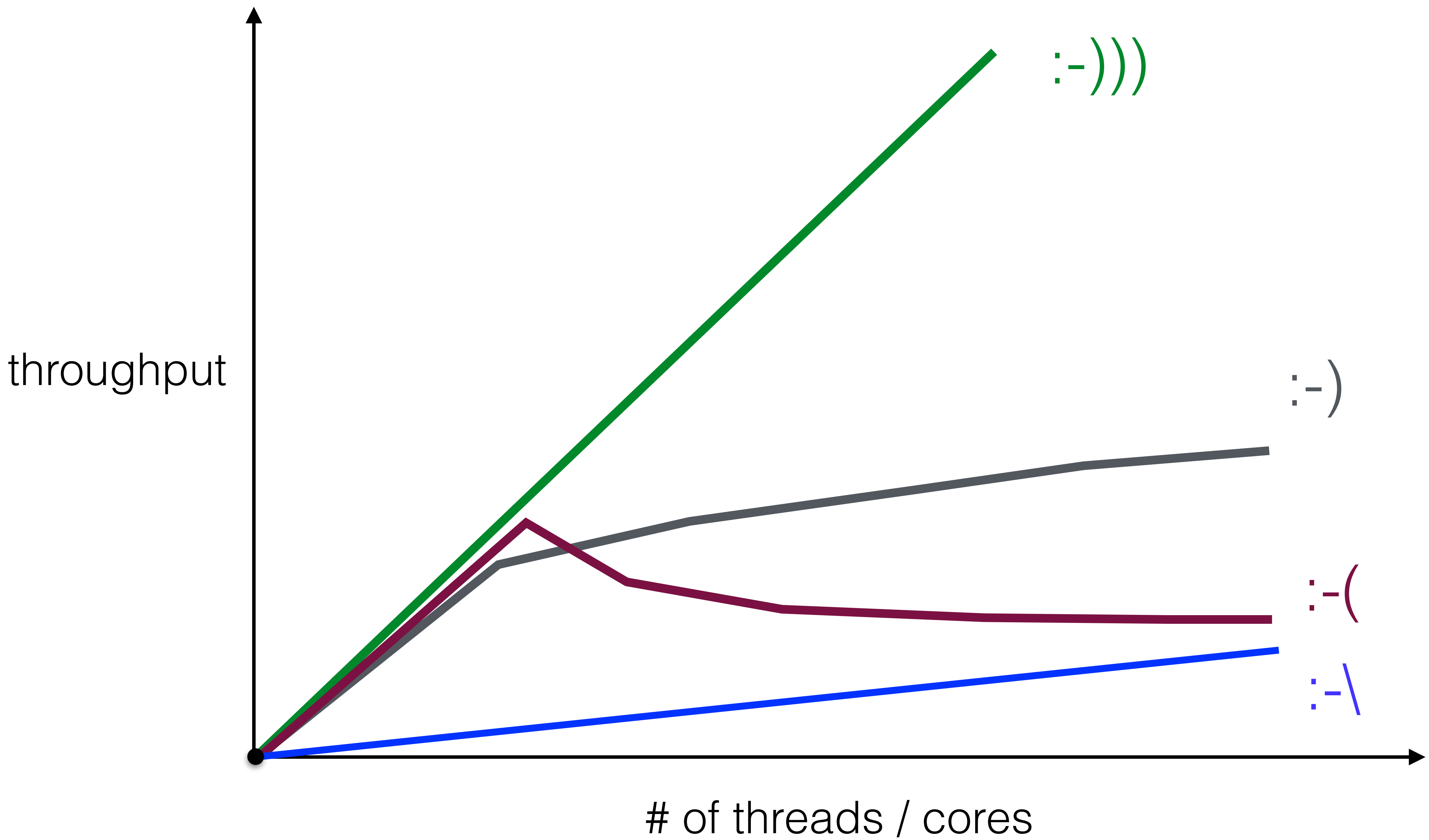
↓

Sequential Consistency [Lamport'79]

there exists a sequential witness that preserves per-thread precedence (program order)

t1:		[1]enq(1)	deq(2)[4]
t2:	deq(1)[2]		enq(2)[3]

Performance and scalability



Relaxations allow trading

correctness
for
performance

provide the potential for better-performing implementations

Relaxing the Semantics

not "sequentially correct"

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

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

Local linearizability
in this talk

for queues only
(feel free to ask for more)

too weak

Local Linearizability main idea

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

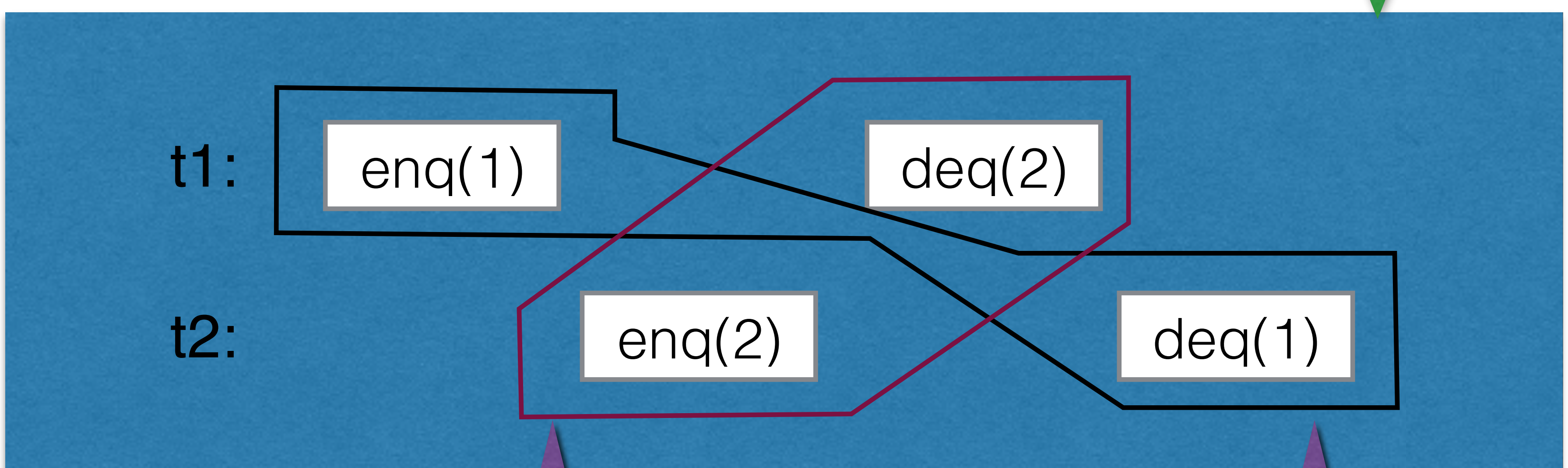
- Partition a history into a set of local histories
- Require linearizability per local history

no global witness

Local sequential consistency… is also possible

Local Linearizability (queue) example

(sequential) history not linearizable



t2-induced history, linearizable

t1-induced history, linearizable

locally linearizable

Local Linearizability (queue) definition

Queue signature $\Sigma = \{enq(x) \mid x \in V\} \cup \{deq(x) \mid x \in V\} \cup \{deq(empty)\}$

For a history **h** with n threads, we put

$$In_{\mathbf{h}}(i) = \{enq(x)^i \in \mathbf{h} \mid x \in V\}$$

in-methods of thread i enqueues performed by thread i

$$Out_{\mathbf{h}}(i) = \{deq(x)^j \in \mathbf{h} \mid enq(x)^i \in In_{\mathbf{h}}(i)\} \cup \{deq(empty)\}$$

out-methods of thread i dequeues (performed by any thread) corresponding to enqueues that are in-methods

h is locally linearizable iff every thread-induced history

$$\mathbf{h}_i = \mathbf{h} \mid (In_{\mathbf{h}}(i) \cup Out_{\mathbf{h}}(i))$$

is linearizable.

Generalizations of Local Linearizability

Signature Σ

For a history **h** with n threads, choose

$In_{\mathbf{h}}(i)$

$Out_{\mathbf{h}}(i)$

in-methods of thread i, methods that go in $\mathbf{h}_i$

out-methods of thread i, dependent methods on the methods in $In_{\mathbf{h}}(i)$ (performed by any thread)

by increasing the in-methods, LL gradually moves to linearizability

h is locally linearizable iff every thread-induced history

$\mathbf{h}_i = \mathbf{h} \mid (In_{\mathbf{h}}(i) \cup Out_{\mathbf{h}}(i))$

is linearizable.

Where do we stand?

In general

Linearizability

Local Linearizability

Sequential Consistency

Where do we stand?

For queues (and all pool-like data structures)

Linearizability

Local Linearizability

Sequential Consistency

Where do we stand?

C: For queues

Linearizability

Local Linearizability
& Pool-seq.cons.

Sequential Consistency

Properties

Local linearizability is compositional

like linearizability
unlike sequential consistency

h (over multiple objects) is locally linearizable
iff
each per-object subhistory of **h** is locally linearizable

Local linearizability is modular / "decompositional"

uses decomposition into smaller histories, by definition

may allow for modular verification

Verification (queue)

Queue sequential specification (axiomatic)

s is a legal queue sequence

iff

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

Queue linearizability (axiomatic)

h is queue linearizable

iff

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

precedence order

Verification (queue)

Queue sequential specification (axiomatic)

s is a legal queue sequence

iff

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

Queue local linearizability (axiomatic)

h is queue locally linearizable

iff

1. **h** is pool locally linearizable, and
2. $\text{enq}(x) <_{\mathbf{h}}^{i} \text{enq}(y) \;\wedge\; \text{deq}(y) \in \mathbf{h} \;\Rightarrow\; \text{deq}(x) \in \mathbf{h} \;\wedge\; \text{deq}(y) \not<_{\mathbf{h}} \text{deq}(x)$

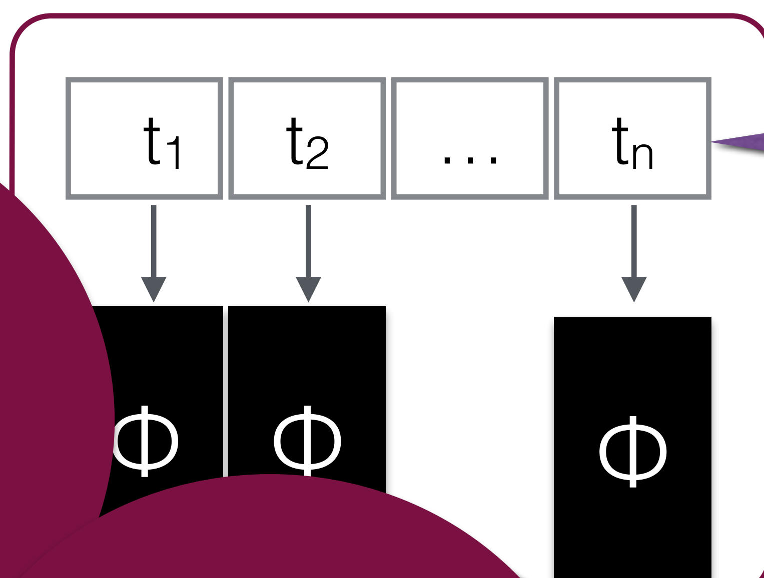
thread-local
precedence order

Generic Implementations

Your favorite linearizable data structure implementation

Φ

turns into a locally linearizable implementation by:



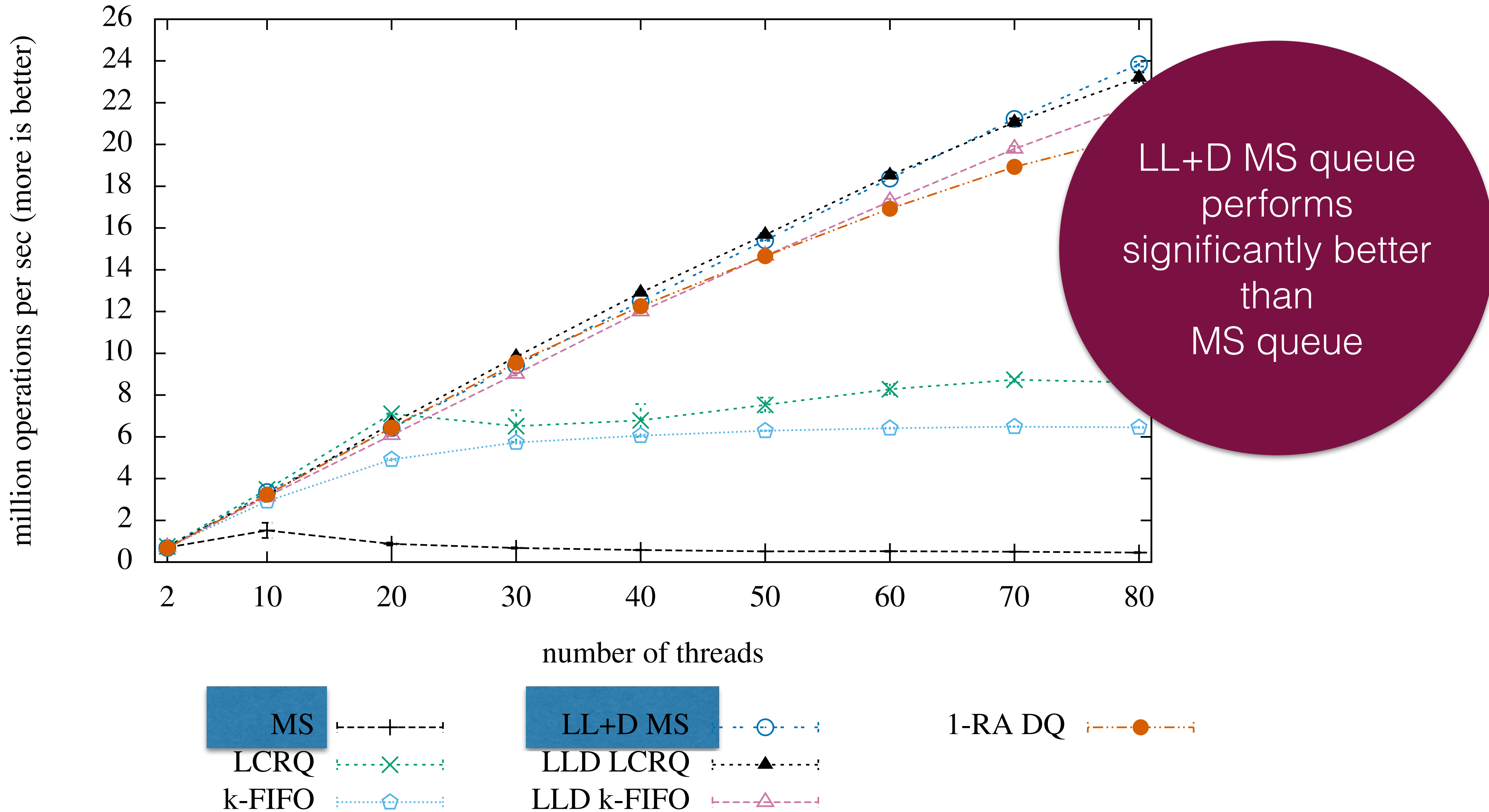
segment of dynamic size (n)

LLD Φ (locally linearizable)

LL+D Φ (also pool linearizable)

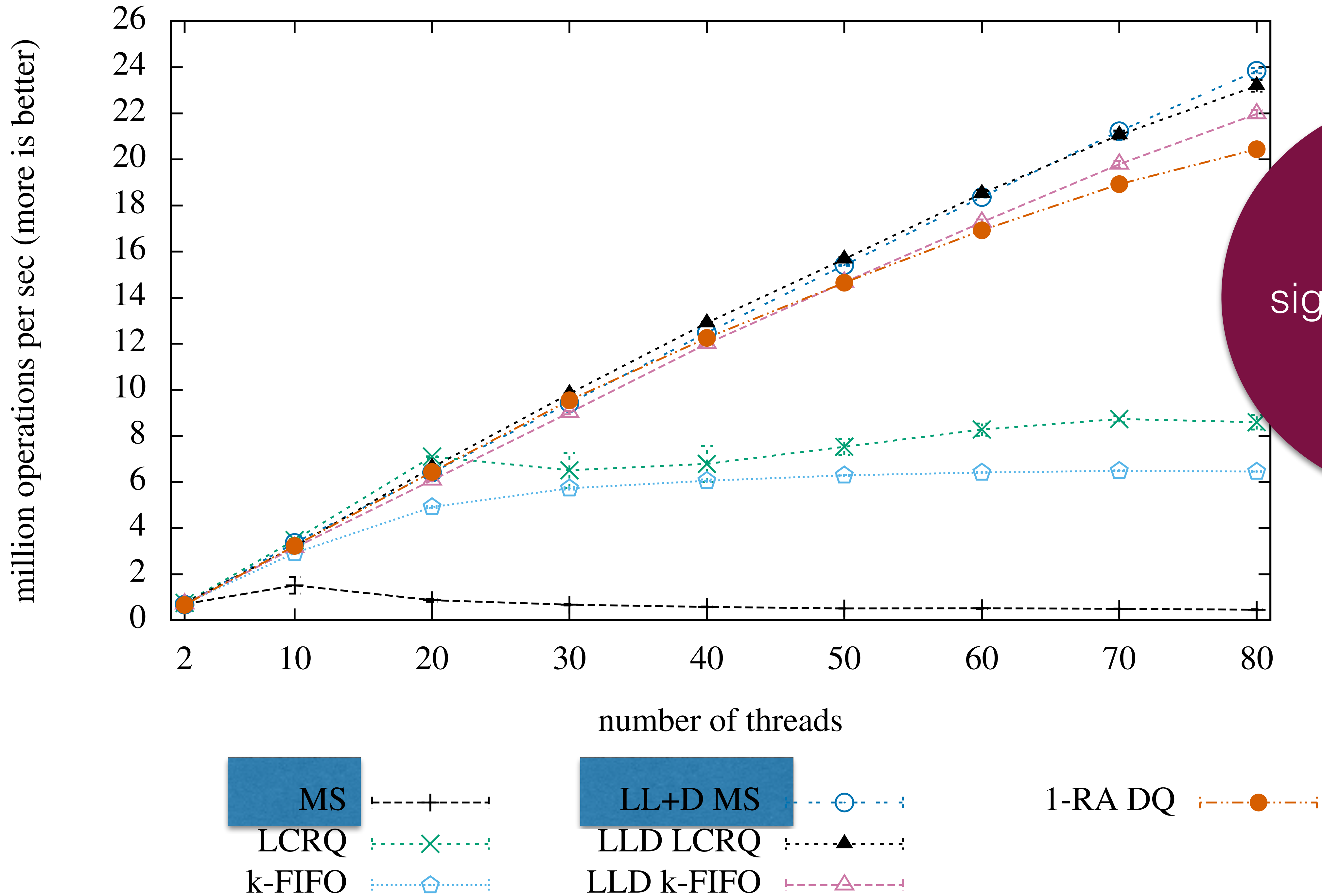
local inserts / global (randomly distributed) removes

Performance



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

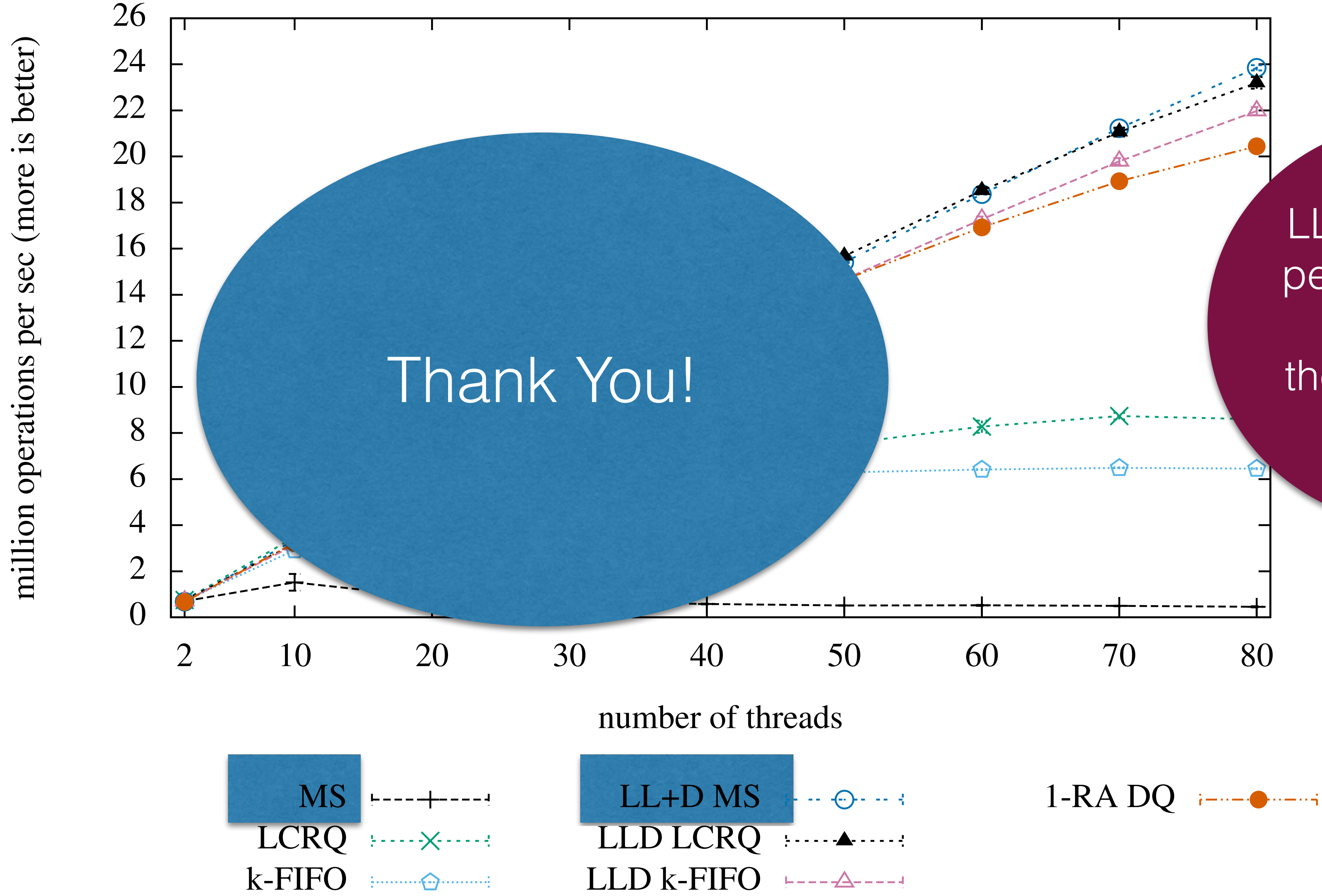
Performance



LLD Φ performs significantly better than Φ

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

Performance



Thank You!

LLD MS queue performs better than the best known pools

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