Citron: Distributed Range Lock Management with One-sided RDMA

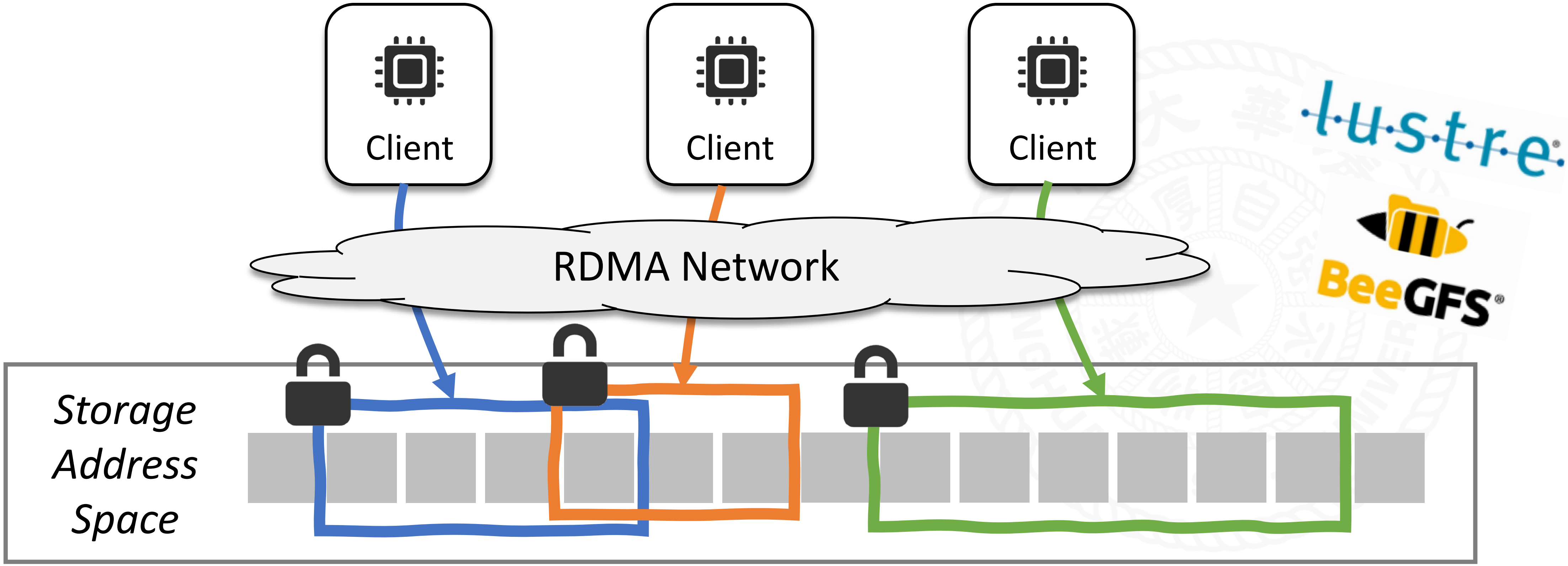
Jian Gao, Youyou Lu, Minhui Xie, Qing Wang, Jiwu Shu

Tsinghua University

FAST’23

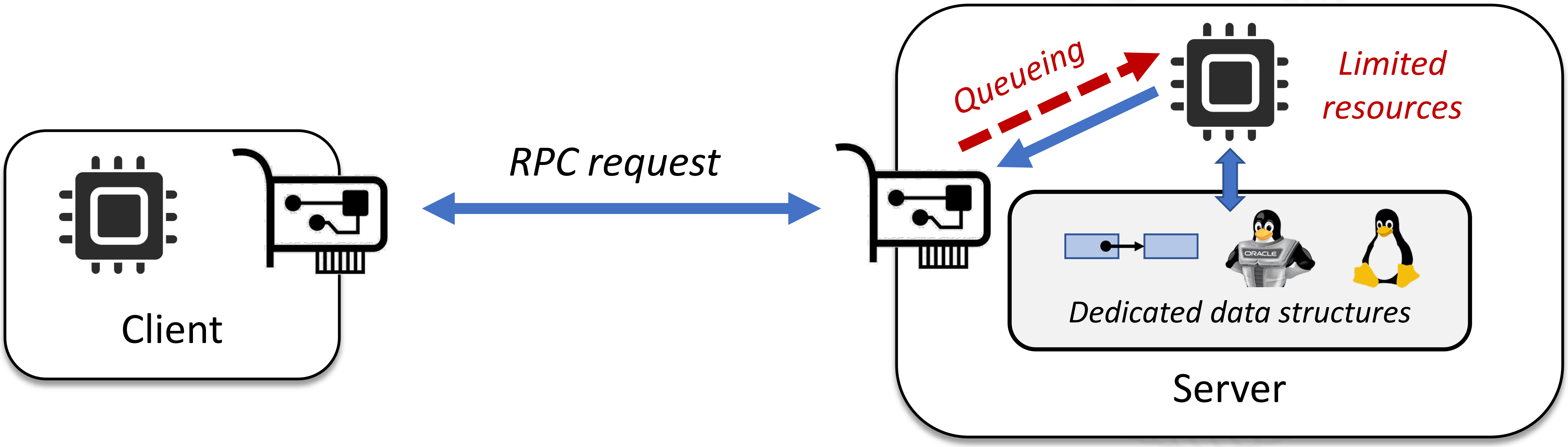
Range Locks

Concurrency safety for accessing **storage address space**



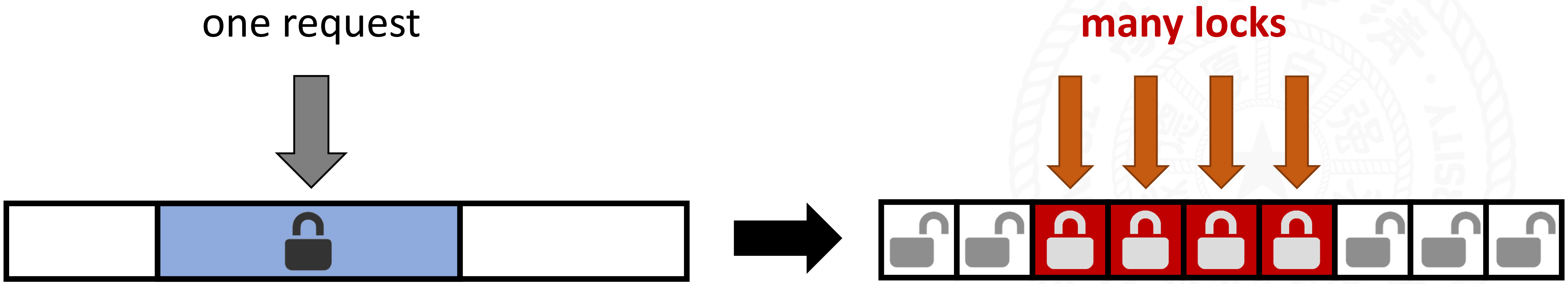
Previous Approaches

- **CPU-based** range lock manager with **two-sided RDMA-based** RPC
- **Rely fully on server-side CPUs -> CPU bottleneck**



One-sided Approach?

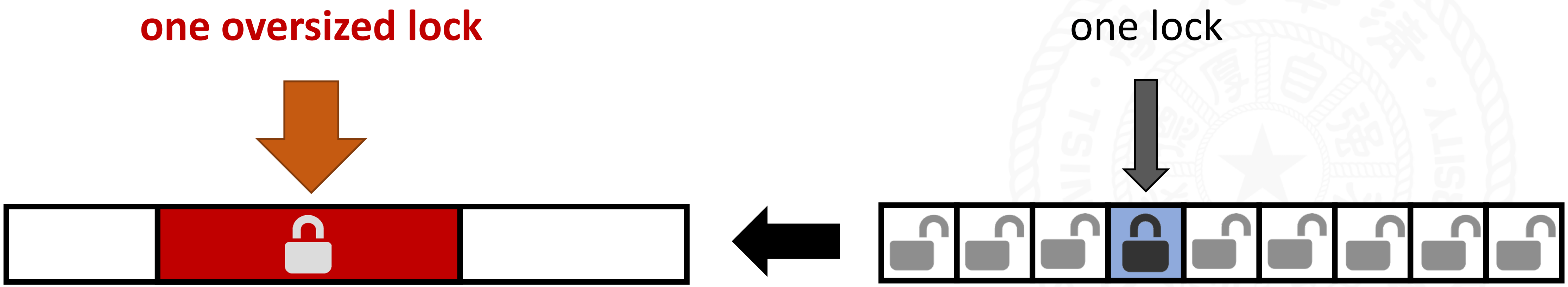
- Range lock = **mutex (already solved!)** * range size
- **Excessive network roundtrips -> high latencies**



* Dong Young Yoon, et al. Distributed Lock Management with RDMA: Decentralization without Starvation. In *SIGMOD'18*.

One-sided Approach? (Cont.)

- Combine multiple mutexes into a larger one
- **Resource waste for small ranges leads to low throughput**



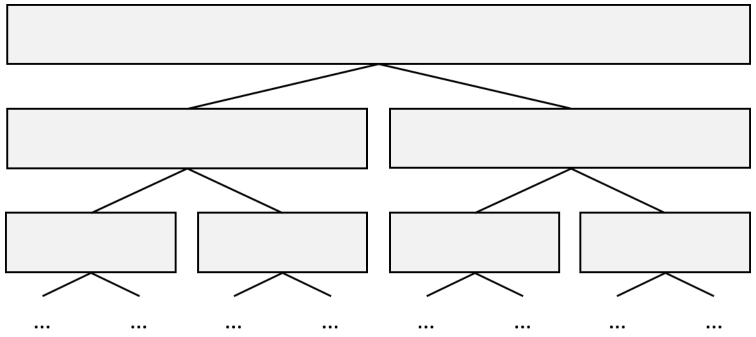
* Dong Young Yoon, et al. Distributed Lock Management with RDMA: Decentralization without Starvation. In *SIGMOD'18*.

Our One-sided Solution: **Citron**

effi***CI***ent dis***T***ributed ***R***ange l***O***ck ma***N***ager

The Big Picture

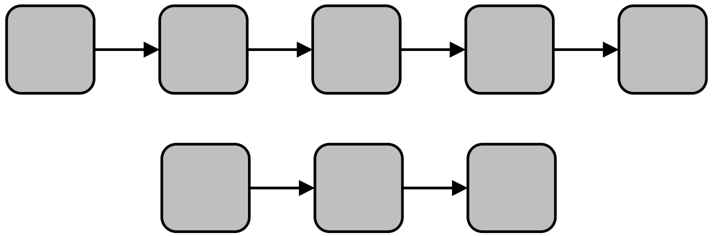
Citron



Data Structure

Segment Tree

Ranges to mutexes



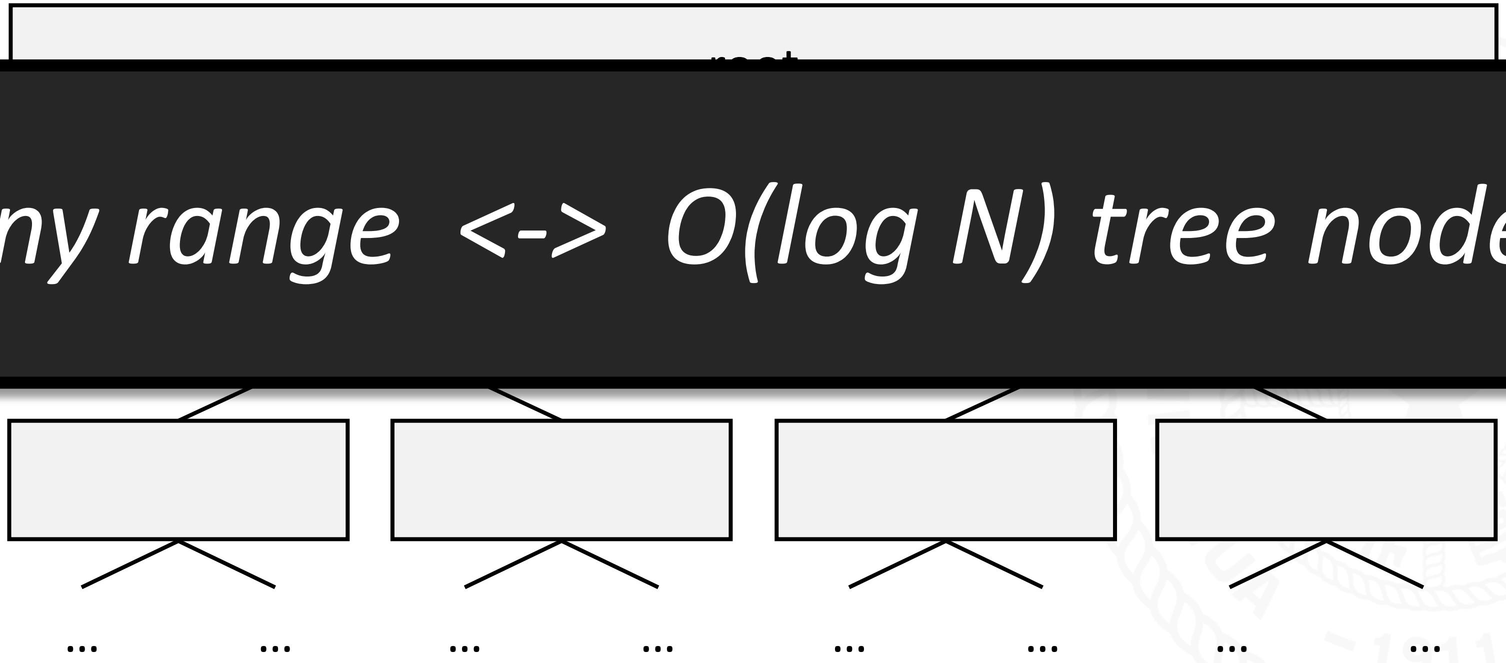
Lock Protocol

Extended atomics

Resolve conflicts

The Segment Tree

A perfect tree whose each node represents a range.



Any range <-> O(log N) tree nodes

Example: *degree* = 2

Map Ranges to Tree Nodes

Throughput-Optimal

+ Precise mapping
– **High latencies:** $O(\log N)$ nodes, **tens** in the worst case

Latency-Optimal

+ Lock only one node
– **Low throughput:** false conflicts

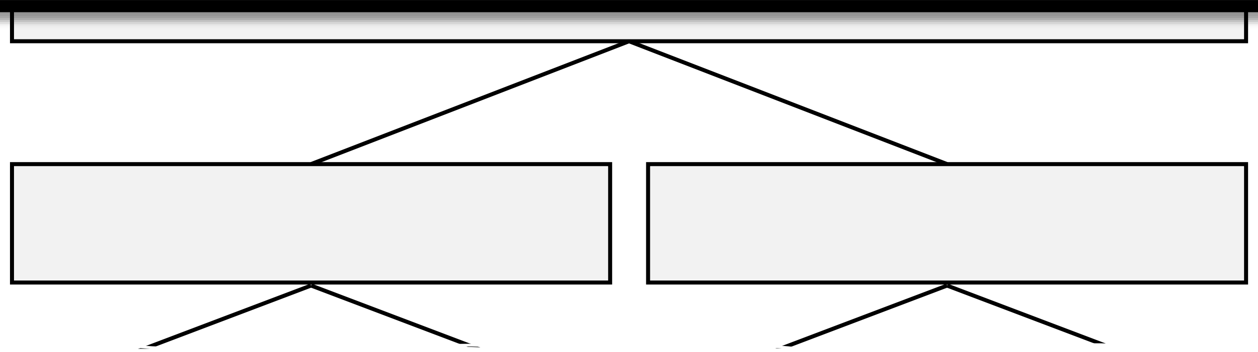
Citron's trade-off

Lock up to k nodes. $\boldsymbol{k}$ **= 2 by default.**

Selecting the Nodes Properly

Goal: minimize locked but unneeded range.

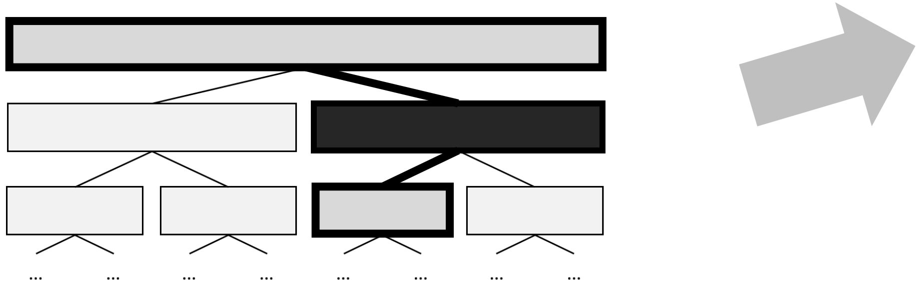
*k = 2 reduces false conflicts by **96%** than k = 1*

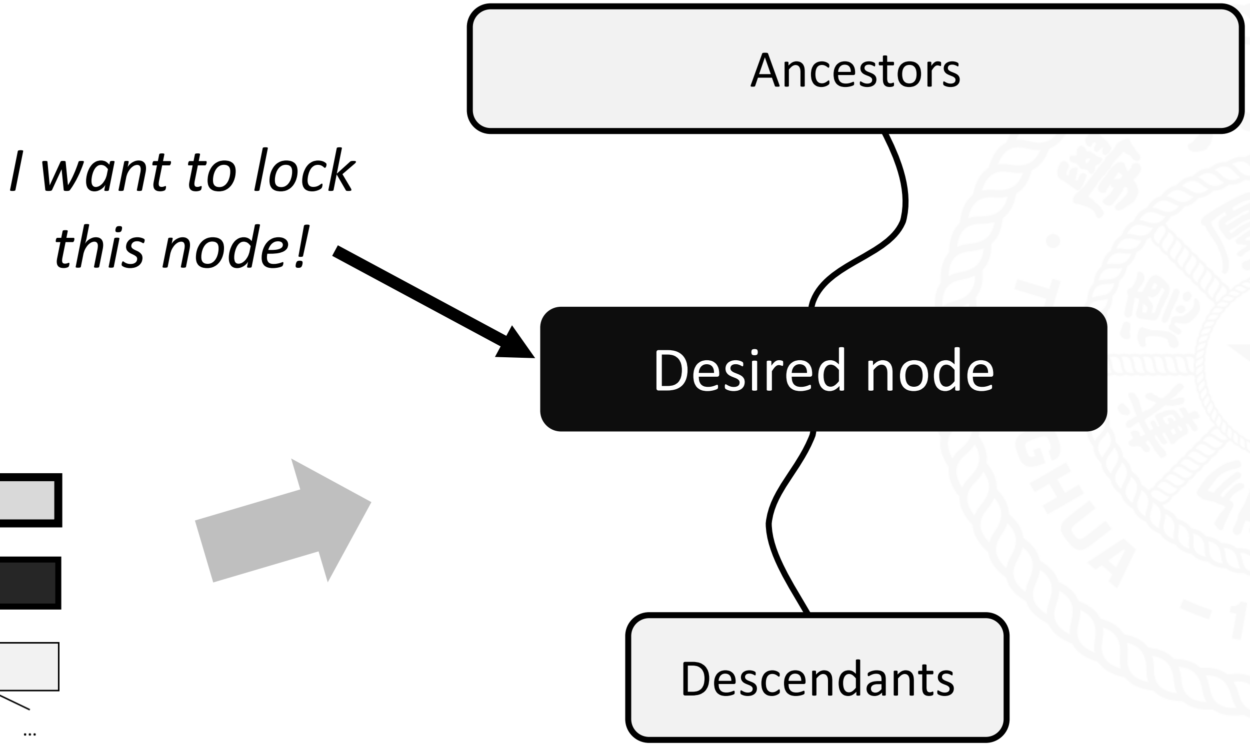


- **Value:** preciseness improvement

Lock Protocol - Theory

- **Observation:** Node X conflict only with **ancestors** and **descendants**

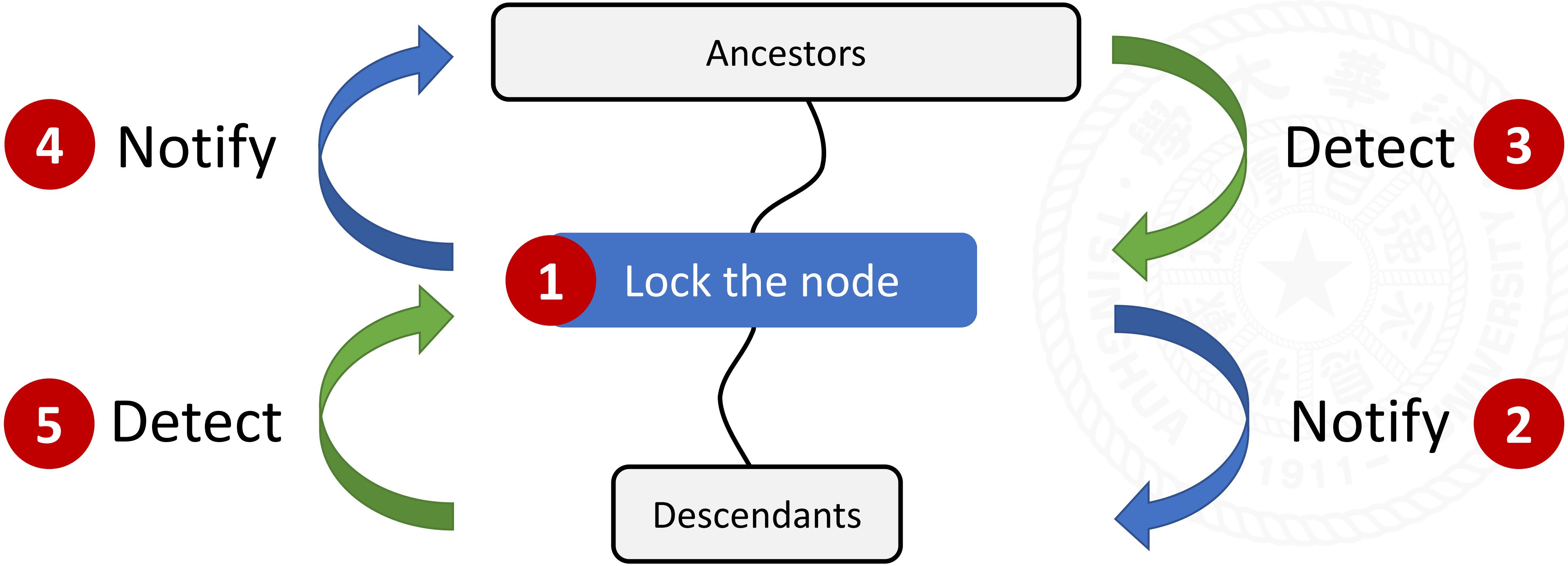




Notification
+
Detection

Lock Protocol - Practice, i.e., Tasks

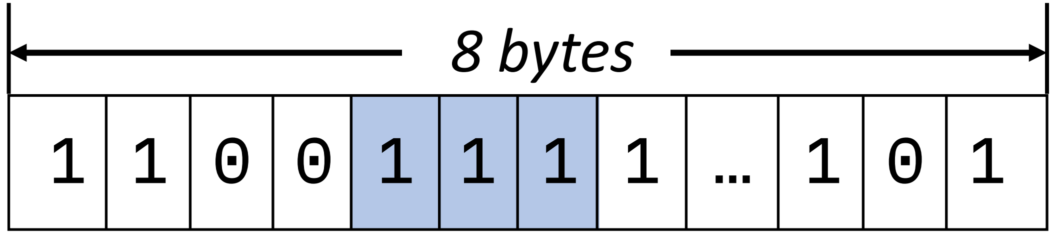
- **Observation:** Node X conflict only with **ancestors** and **descendants**



Lock Protocol - Tools

RDMA NICs nowadays support Extended Atomics.

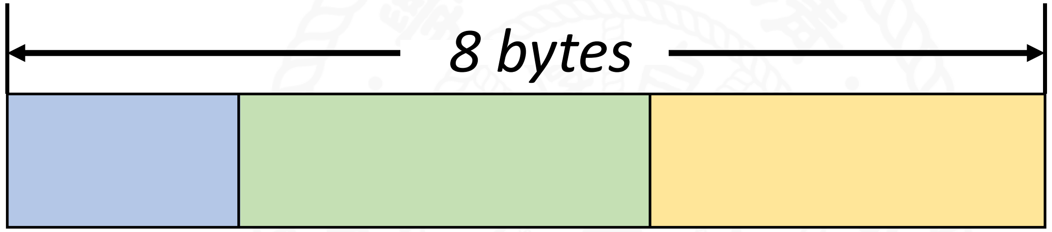
Ext-CAS



Manipulate **leaf nodes**

Set bit = locked, unset bit = unlocked

Ext-FAA



Manipulate **internal nodes**

Idea from Lamport's bakery

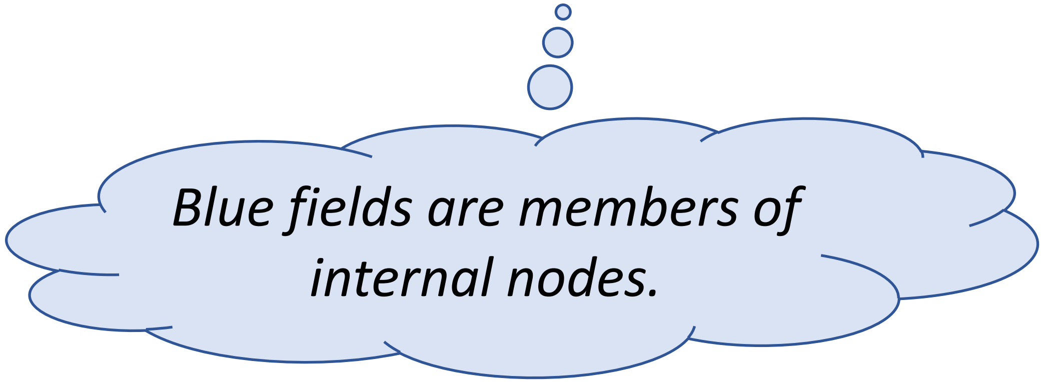
Task 1 - Lock the Desired Node

- **Leaf:** Ext-CAS -- lock desired bits
- **Internal:**
  - Enter bakery

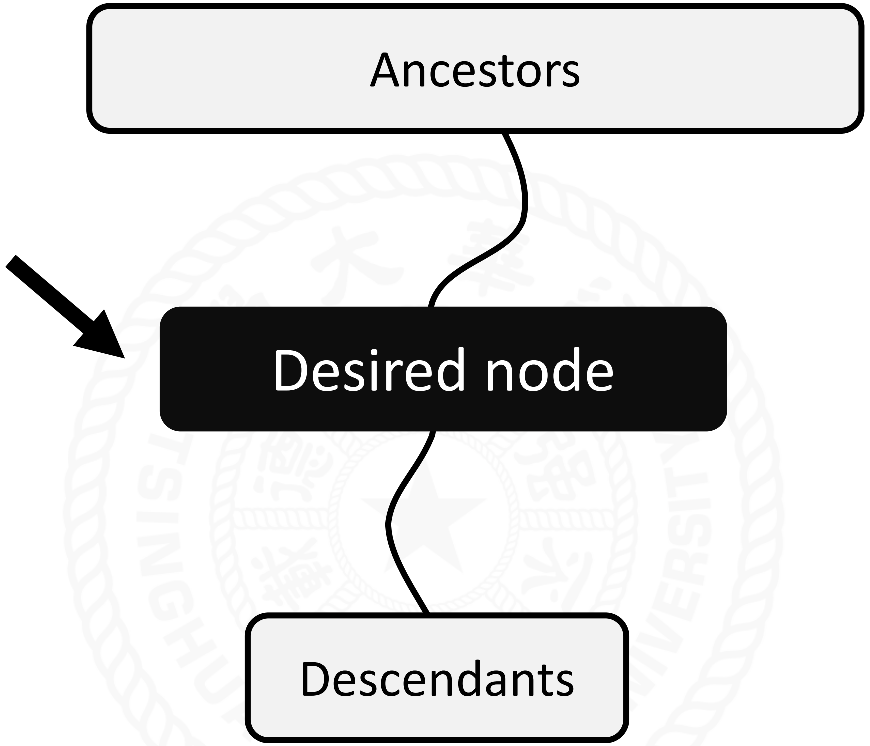
```
mypos = rear++;
```

  - Then wait for my turn

```
while (head != mypos);
```

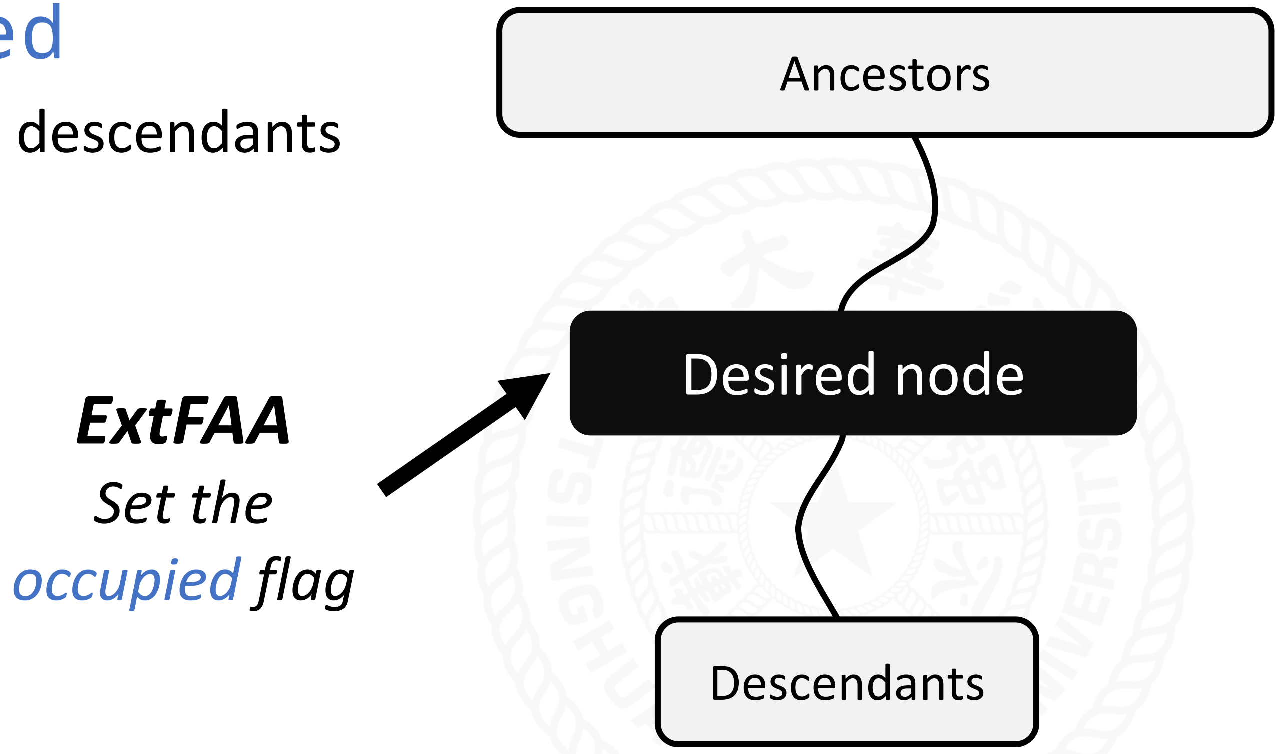


ExtCAS or ExtFAA+ Read



Task 2 - Notify Descendants

- A 1-bit flag called `occupied`
  - Blocks subsequent conflicts @ descendants
  - `occupied = 1;`



Task 3 - Detect Ancestors

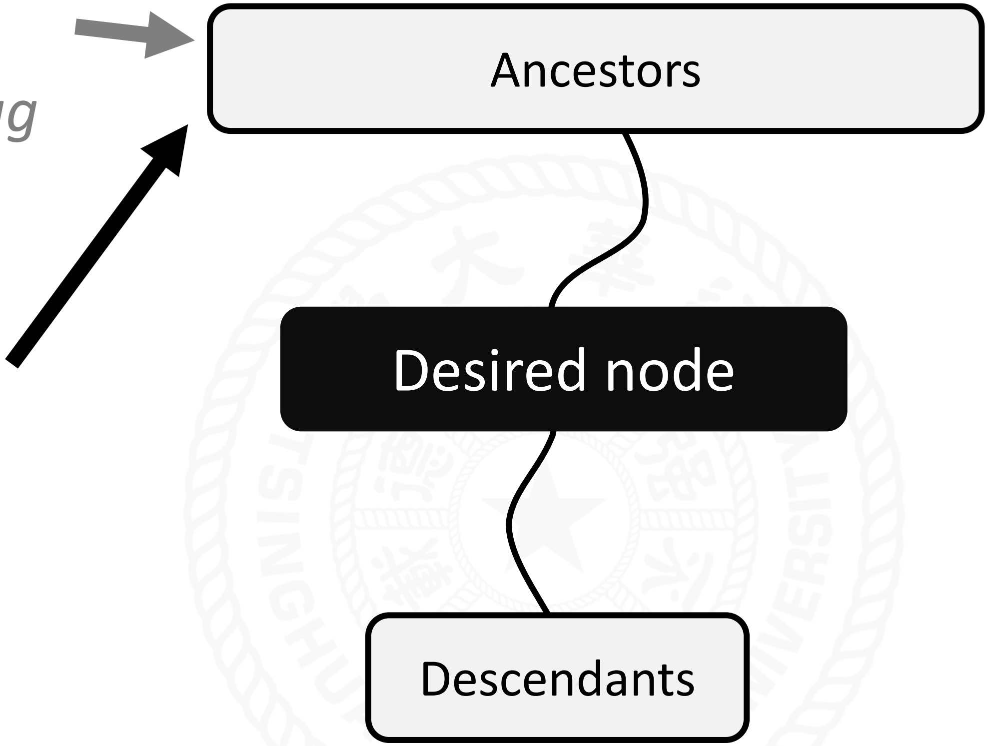
Another client's Task 2:

counterpart

ExtFAA

Set the occupied flag

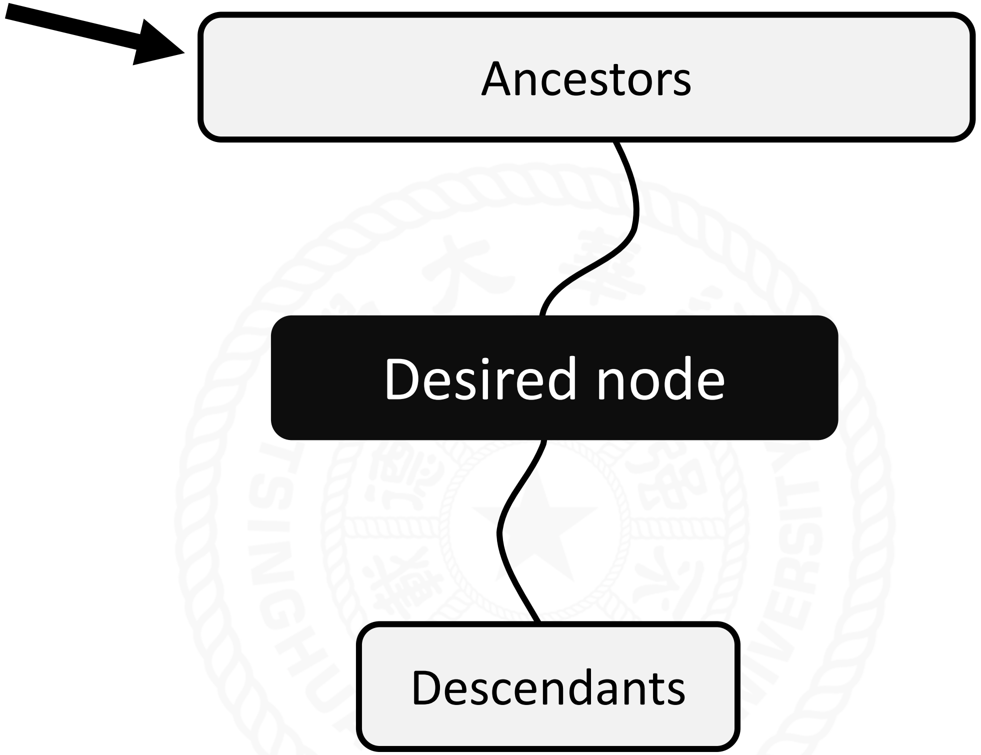
Read



- Read the `occupied` flag
- Wait for occupied ancestors
  - `while (ancestor.occupied);`

Task 4 - Notify Ancestors

ExtFAA
Increment d_rear



- Another pair of counters
  - `d_head, d_rear`
- Enter the “descendant queue” of each ancestor
  - `ancestor.d_rear++;`

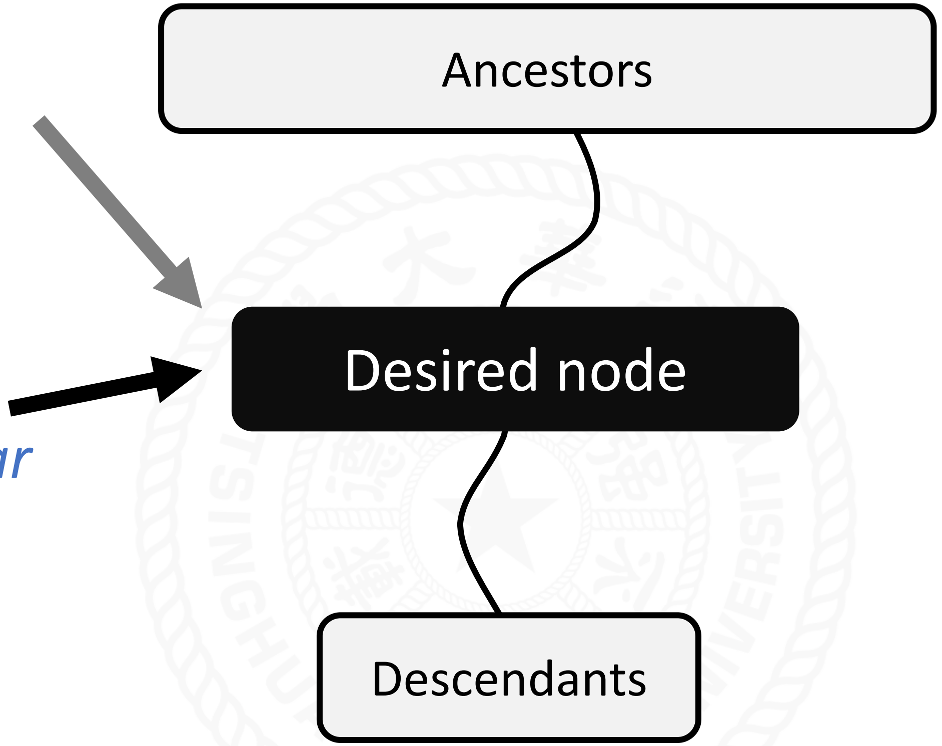
Task 5 - Detect Descendants

Another client's Task 4:

counterpart

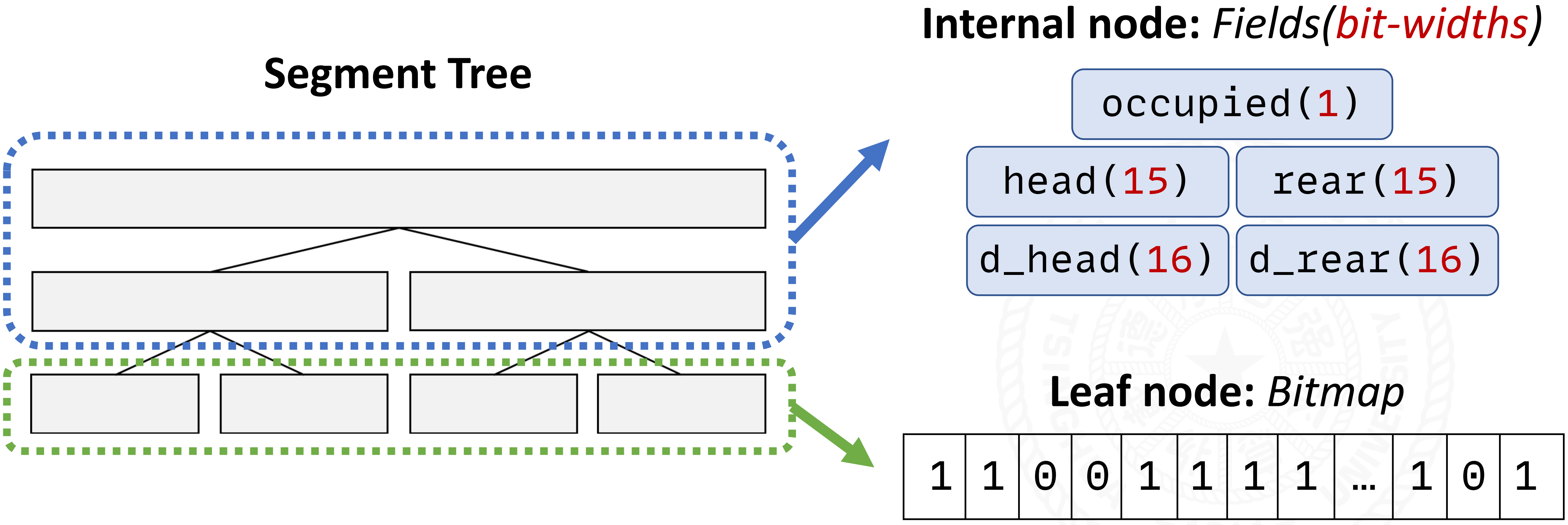
ExtFAA
Increment d_rear

Read
Poll d_head & d_rear



- Wait for my "descendant queue" to get emptied
  - `while (d_head != d_rear);`

Data Structure Summary



Lock Protocol Summary

Task	RDMA Verbs	Verbs' Targets
1. Lock the desired node	ExtCAS / ExtFAA + Read	Desired node
2. Notify descendants	ExtFAA	Desired node
3. Detect conflicts at ancestors	Read	Ancestors
4. Notify ancestors	ExtFAA	Ancestors
5. Detect conflicts at descendants	Read	Desired node

counterparts (2 and 3)

counterparts (4 and 5)

Unlock

Revert modifications done in lock acquisition.

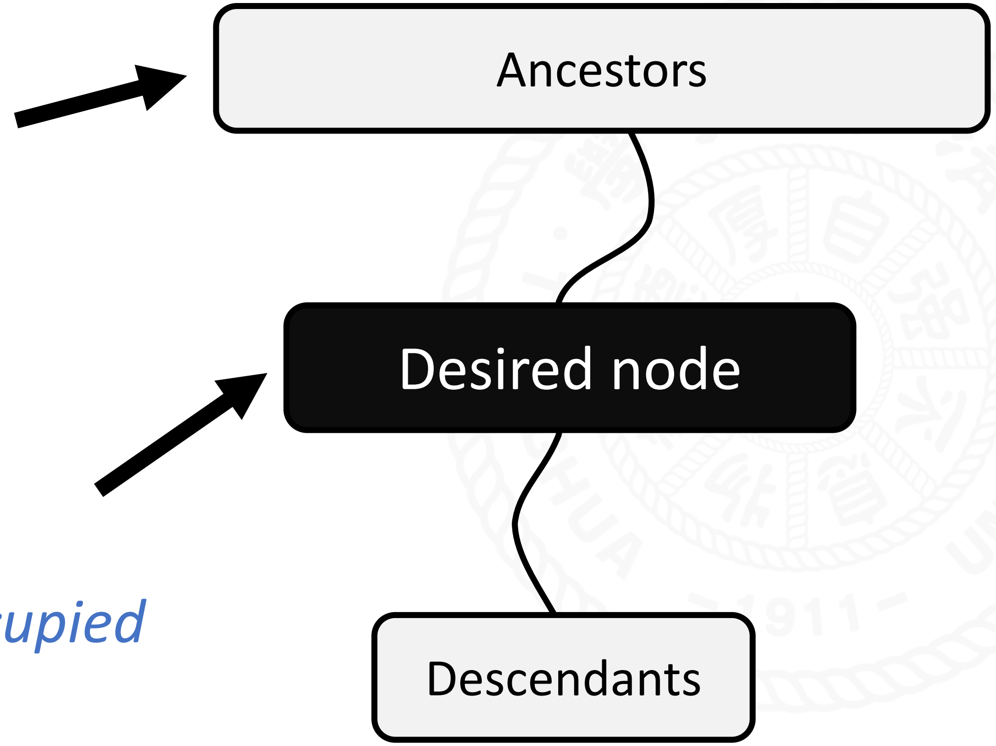
ExtFAA

Increment d_head

ExtCAS / ExtFAA

Clear the bitmap /

Increment head & clear occupied



Supportive Designs

Timing-based Sync (Task 4 & 5) to enforce protocol correctness

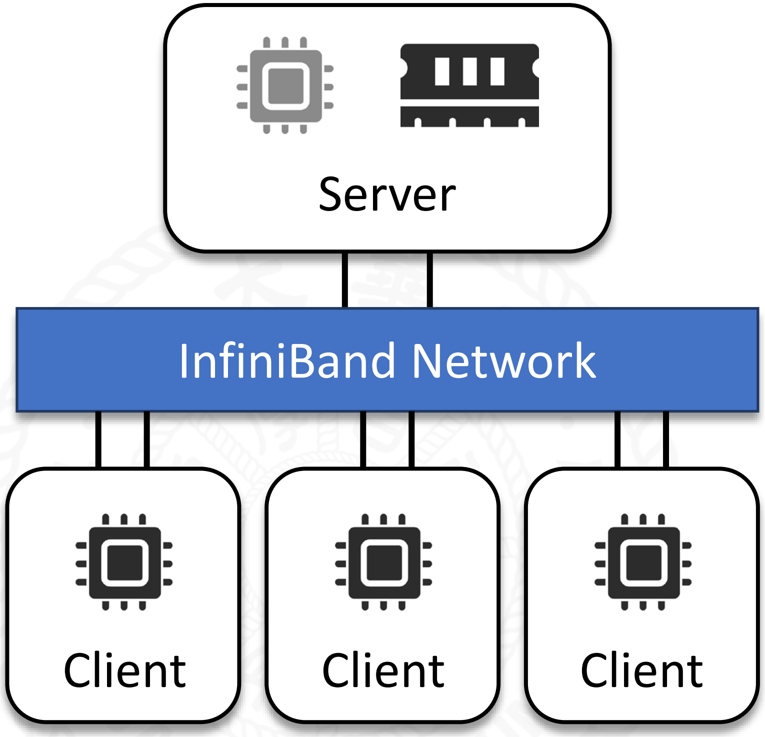
Strided Notification (Task 4) to reduce overheads

- Runtime scaling, parameter tuning, fast path, failure recovery, …
- *See our paper for more details!*

Experiment Setup

CPU	Intel Xeon Gold 5220 @ 2.20 GHz
Memory	256 GB
NIC	Mellanox ConnectX-6

- 3 clients + 1 server
- **Microbenchmark:** lock acquire & release
- **Application:** Filebench & IO500 & NPB BT-IO

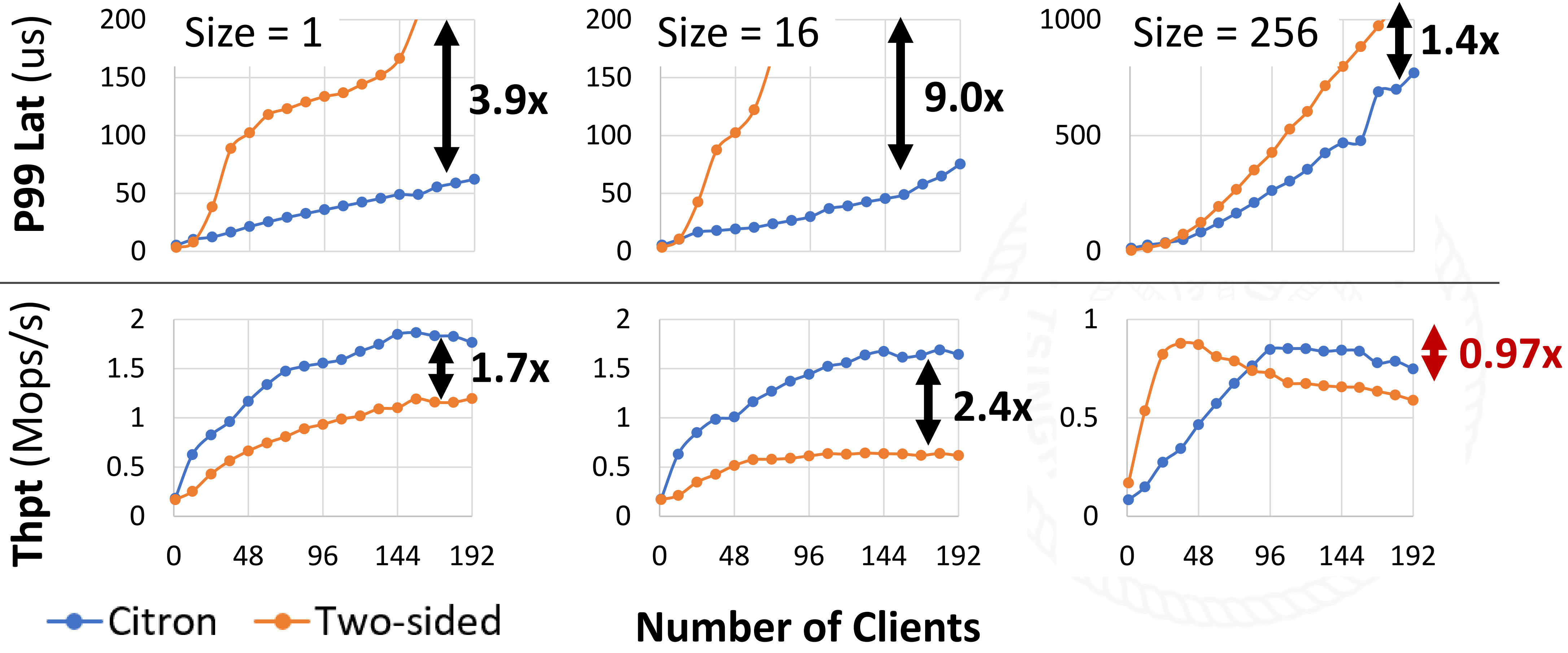


Baselines

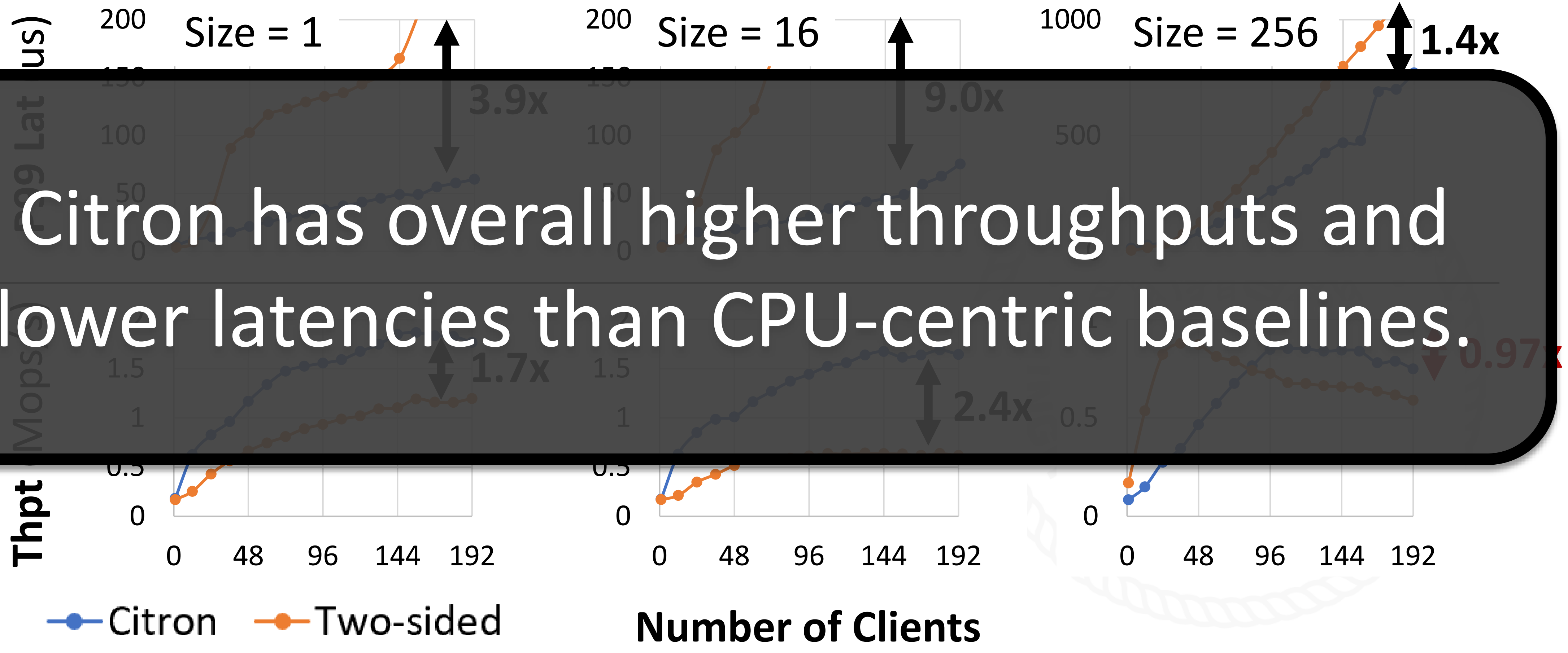
Type	Description
Two-sided RDMA with eRPC [2]	**Maple tree** from Oracle Linux UEK
	Interval tree from Lustre
	Linked list from [1]
One-sided RDMA	**Linked list** from [1] with pure RDMA
	Trivially mutex * range size

[1] Alex Kogan, et al. Scalable Range Locks for Scalable Address Spaces and Beyond. In *EuroSys'20*.
[2] Anuj Kalia, et al. Datacenter RPCs can be General and Fast. In *NSDI'19*.

Citron vs. Two-sided

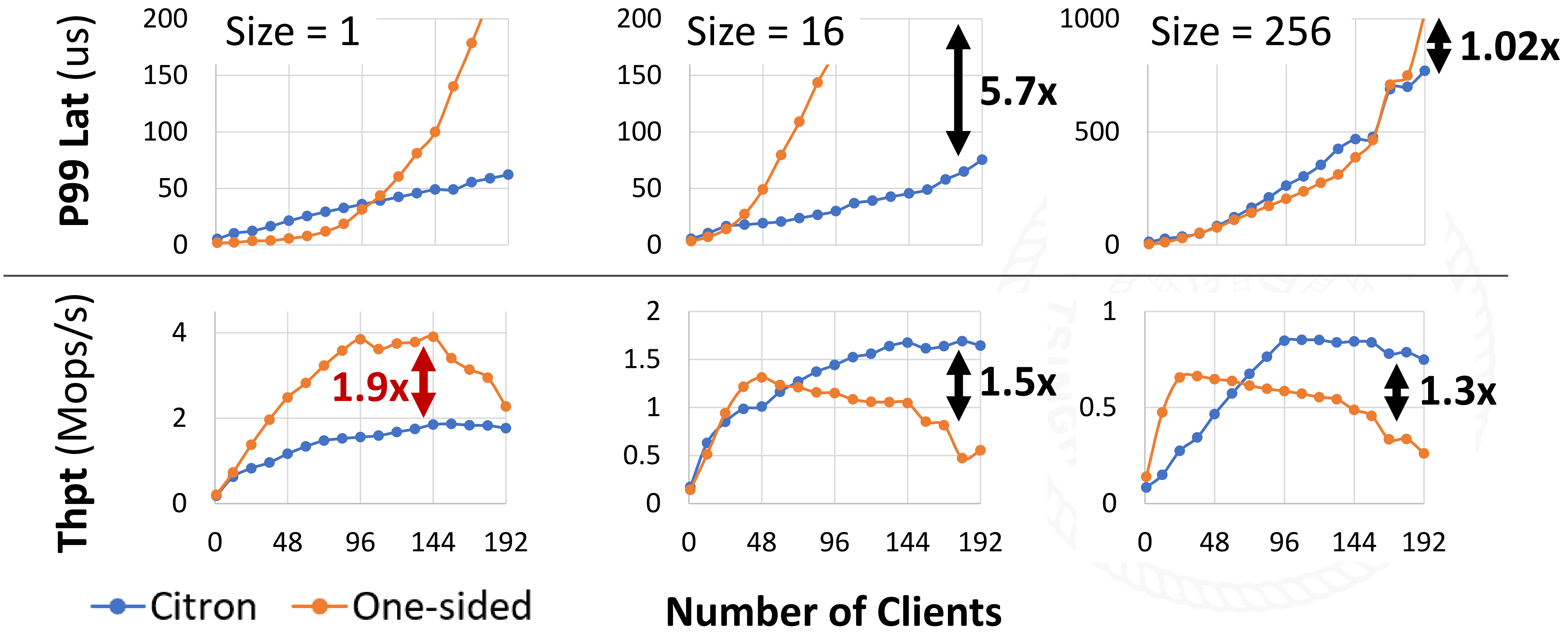


Citron vs. Two-sided

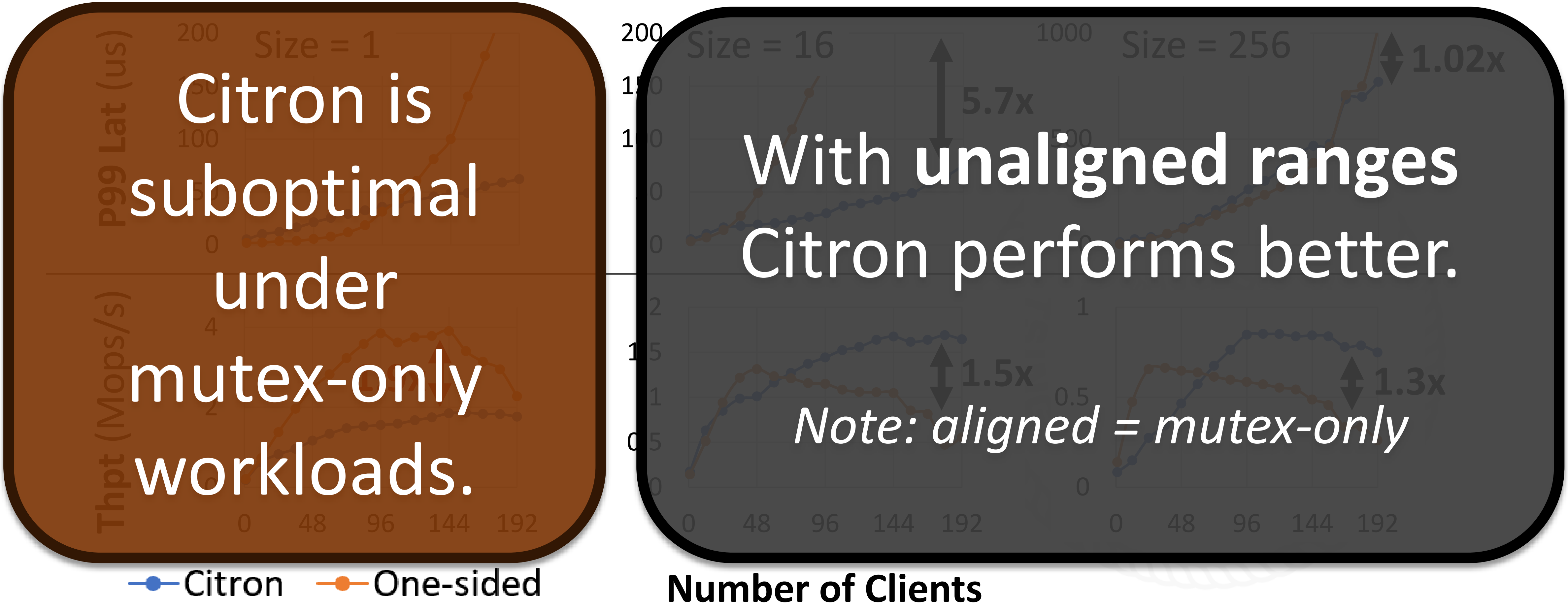


Citron has overall higher throughputs and lower latencies than CPU-centric baselines.

Citron vs. Trivial One-sided



Citron vs. Trivial One-sided



Citron is suboptimal under mutex-only workloads.

With **unaligned ranges** Citron performs better.

Note: aligned = mutex-only

Other Evaluation Results

- **P50 latencies** *similar*
- **P99 latencies** *orders-of-magnitude lower*

- Limited **false conflict rates** and **abort rates** (1e-5 ~ 1e-2)

- Quickly adapt to storage resource size growths (*sub-millisecond* level)

Conclusion

- We designed Citron, an efficient ***distributed range lock manager*** using ***only one-sided RDMA*** to acquire and release locks.
- Citron translates ranges into RDMA-friendly mutexes with a ***segment tree*** and use ***RDMA Extended Atomics*** to perform synchronization.
- Citron achieves overall higher performance than both two-sided and trivial one-sided baselines.

Thanks & Q/A

Citron: Distributed Range Lock Management
with One-sided RDMA

Jian Gao, Youyou Lu, Minhui Xie, Qing Wang, Jiwu Shu

Contact: *gaoj20@mails.tsinghua.edu.cn*