A Scalable Ordering Primitive for Multicore Machines

Sanidhya Kashyap Changwoo Min Kangnyeon Kim Taesoo Kim

Era of multicore machines

Supermicro
Intel Xeon F

By Sue Smith / NewsFactc

PUBLISHED:
OCTOBER
24

S
en
en

Cavium™ Expands the ThunderX2 Server Ecosystem for Cloud and HPC Applications

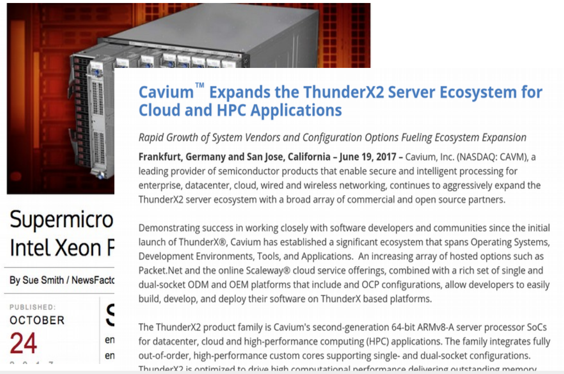
Rapid Growth of System Vendors and Configuration Options Fueling Ecosystem Expansion

Frankfurt, Germany and San Jose, California – June 19, 2017 – Cavium, Inc. (NASDAQ: CAVM), a leading provider of semiconductor products that enable secure and intelligent processing for enterprise, datacenter, cloud, wired and wireless networking, continues to aggressively expand the ThunderX2 server ecosystem with a broad array of commercial and open source partners.

Demonstrating success in working closely with software developers and communities since the initial launch of ThunderX®, Cavium has established a significant ecosystem that spans Operating Systems, Development Environments, Tools, and Applications. An increasing array of hosted options such as Packet.Net and the online Scaleway® cloud service offerings, combined with a rich set of single and dual-socket ODM and OEM platforms that include and OCP configurations, allow developers to easily build, develop, and deploy their software on ThunderX based platforms.

The ThunderX2 product family is Cavium's second-generation 64-bit ARMv8-A server processor SoCs for datacenter, cloud and high-performance computing (HPC) applications. The family integrates fully out-of-order, high-performance custom cores supporting single- and dual-socket configurations. ThunderX2 is optimized to drive high computational performance delivering outstanding memory bandwidth and memory capacity. The new line of ThunderX2 processors includes multiple workload optimized SKUs for both scale up and scale out applications and is fully compliant with ARMv8-A architecture specifications as well as the ARM Server Base System Architecture and ARM Server Base

Scope of multicore machines



Cavium™ Expands the ThunderX2 Server Ecosystem for Cloud and HPC Applications

Rapid Growth of System Vendors and Configuration Options Fueling Ecosystem Expansion

Frankfurt, Germany and San Jose, California – June 19, 2017 – Cavium, Inc. (NASDAQ: CAVM), a leading provider of semiconductor products that enable secure and intelligent processing for enterprise, datacenter, cloud, wired and wireless networking, continues to aggressively expand the ThunderX2 server ecosystem with a broad array of commercial and open source partners.

Demonstrating success in working closely with software developers and communities since the initial launch of ThunderX®, Cavium has established a significant ecosystem that spans Operating Systems, Development Environments, Tools, and Applications. An increasing array of hosted options such as Packet.Net and the online Scaleway® cloud service offerings, combined with a rich set of single and dual-socket ODM and OEM platforms that include and OCP configurations, allow developers to easily build, develop, and deploy their software on ThunderX based platforms.

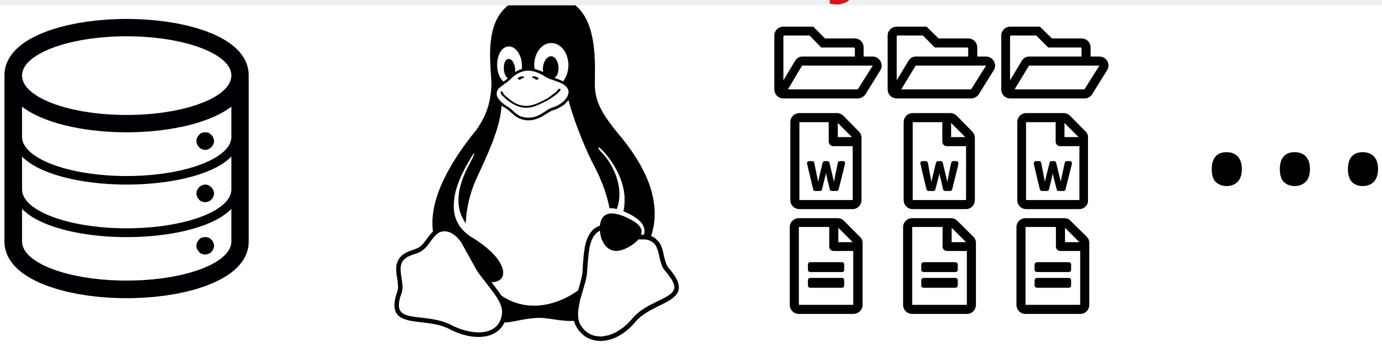
The ThunderX2 product family is Cavium's second-generation 64-bit ARMv8-A server processor SoCs for datacenter, cloud and high-performance computing (HPC) applications. The family integrates fully out-of-order, high-performance custom cores supporting single- and dual-socket configurations.

Huge hardware thread parallelism

How are operations executed correctly?

Ordering

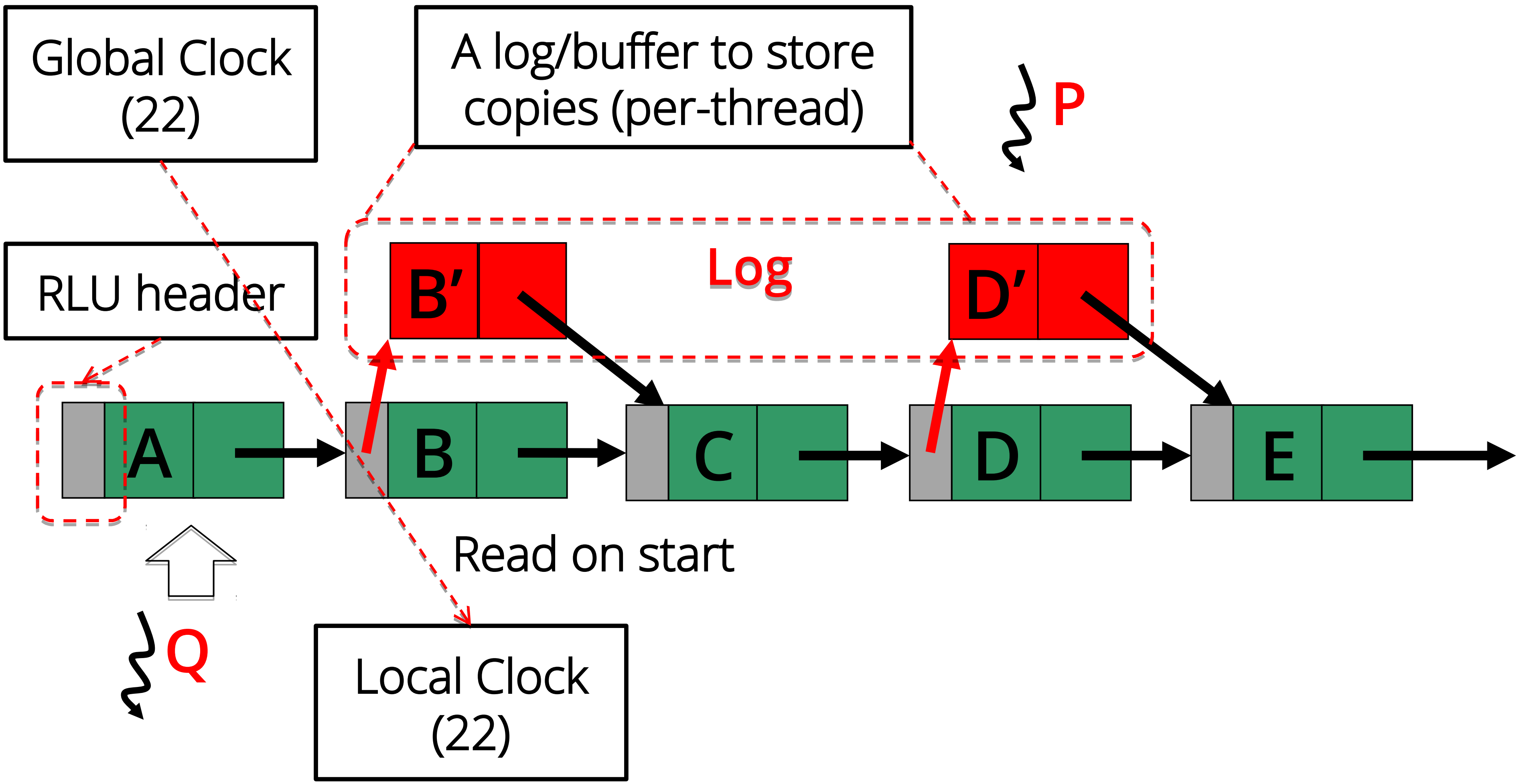
Becomes scalability bottleneck



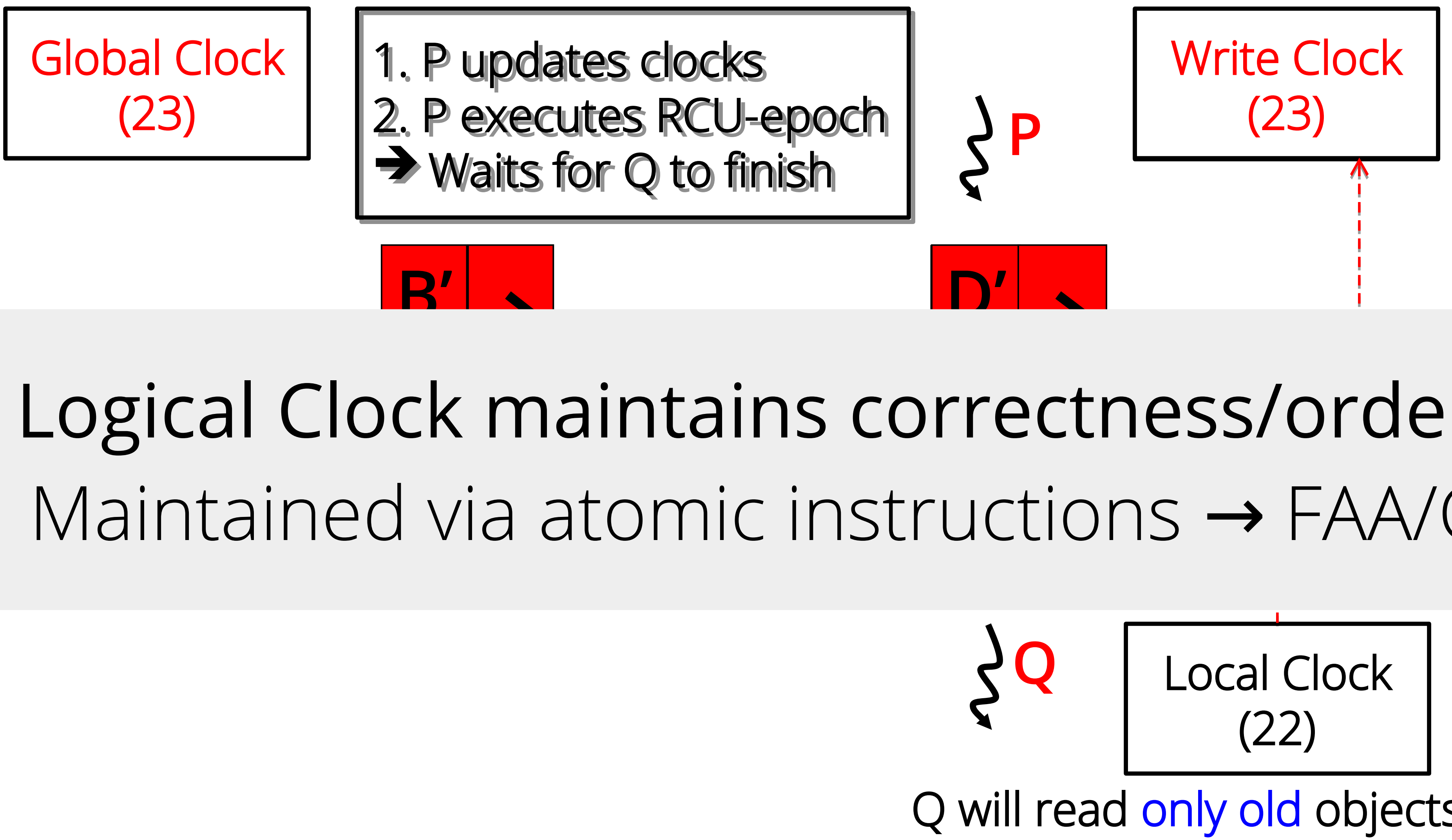
Example: Read Log Update (RLU)

- Extension of RCU
- Modifies objects in a thread's **local log**
- **Clock** maintains correct snapshot (old vs new)
- Frees objects via **epoch-based reclamation**

Read Log Update (RLU) operation



RLU commit operation

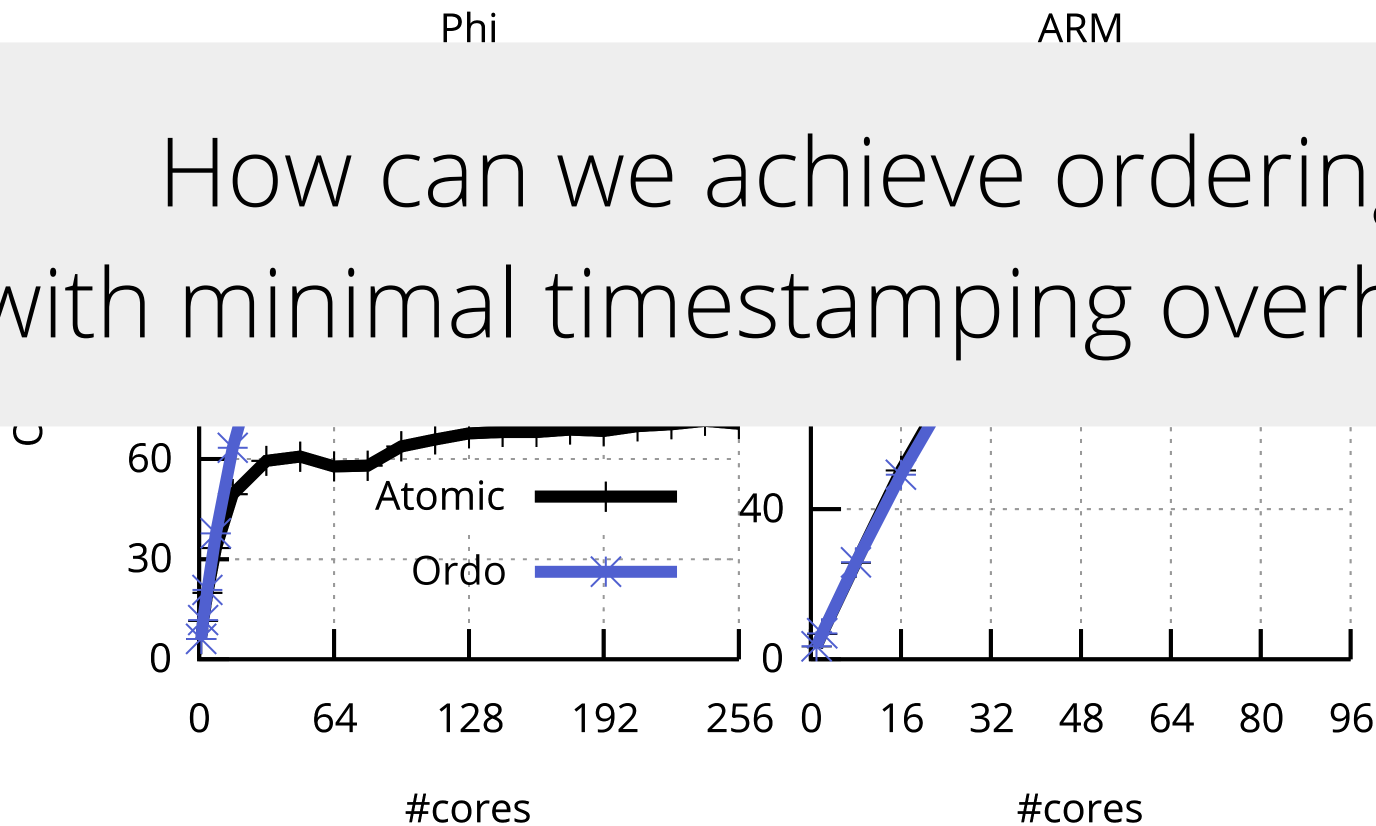


Logical Clock maintains correctness/ordering

Maintained via atomic instructions → FAA/CAS

Issue with logical clock

- RLU suffers from global clock contention
  - Cache-line contention due to atomic instructions
  - Possible to circumvent with our approach



How can we achieve ordering
with minimal timestamping overhead?

Our proposed ordering primitive: *Ordo*

- Exposes a monotonically increasing clock
  - Current hardware already provides
  - rdtscp (X86), cntvct (ARM), stick (Sparc)
- Relies on a per-core **invariant hardware clock**
  - *Monotonically increases with constant skew regardless of dynamic frequency and voltage scaling*

Challenges with Ordo

- Comparing two clocks
  - Clocks are not synchronized
  - Cores receive RESET signal at varying times
- Application:
  - Modifying algorithms to use Ordo
  - Able to compare between two timestamps

Embracing the invariant clocks

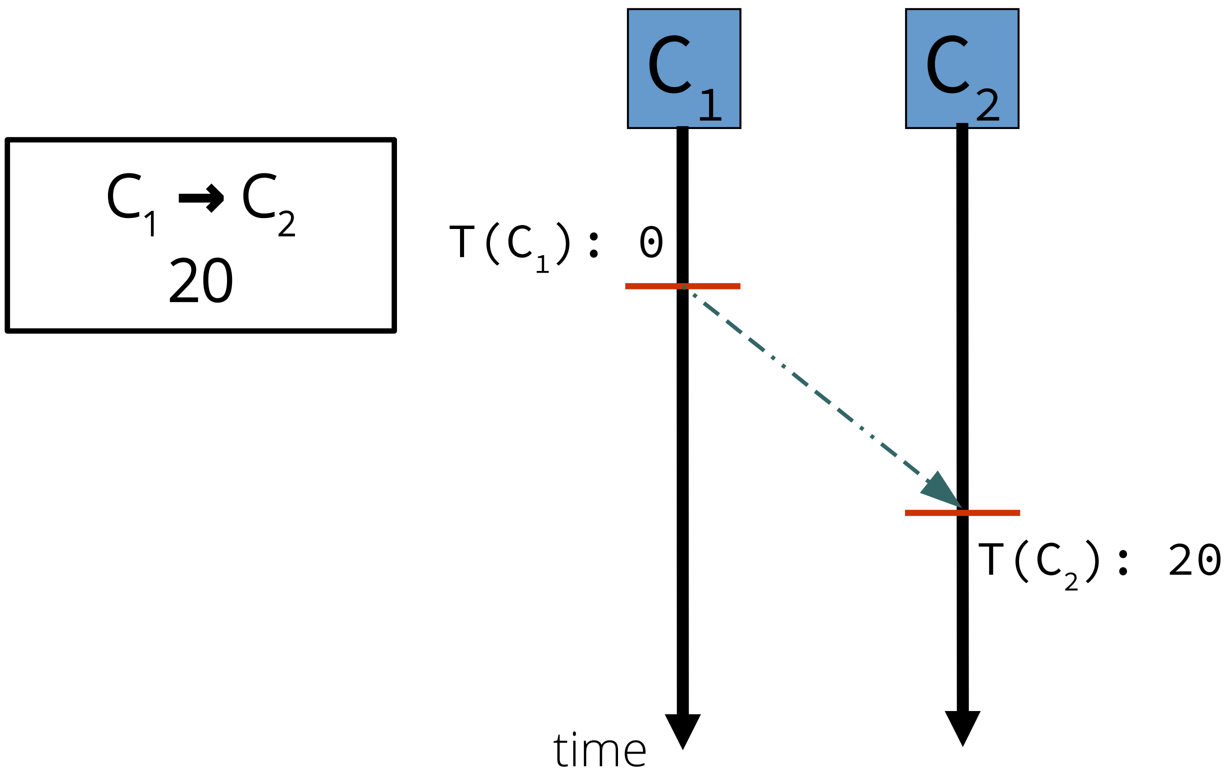
- Measure a global uncertainty window
  - Ensure a new timestamp once a window is over
  - Provides a notion of *globally synchronized clock*
- Measured offset MUST have the invariant:

  Measured offset is greater than the physical offset

  - Physical offset: offset due to RESET signal
  - Measured offset: physical offset + *one-way delay*

Calculating global uncertainty window: ORDO_BOUNDARY

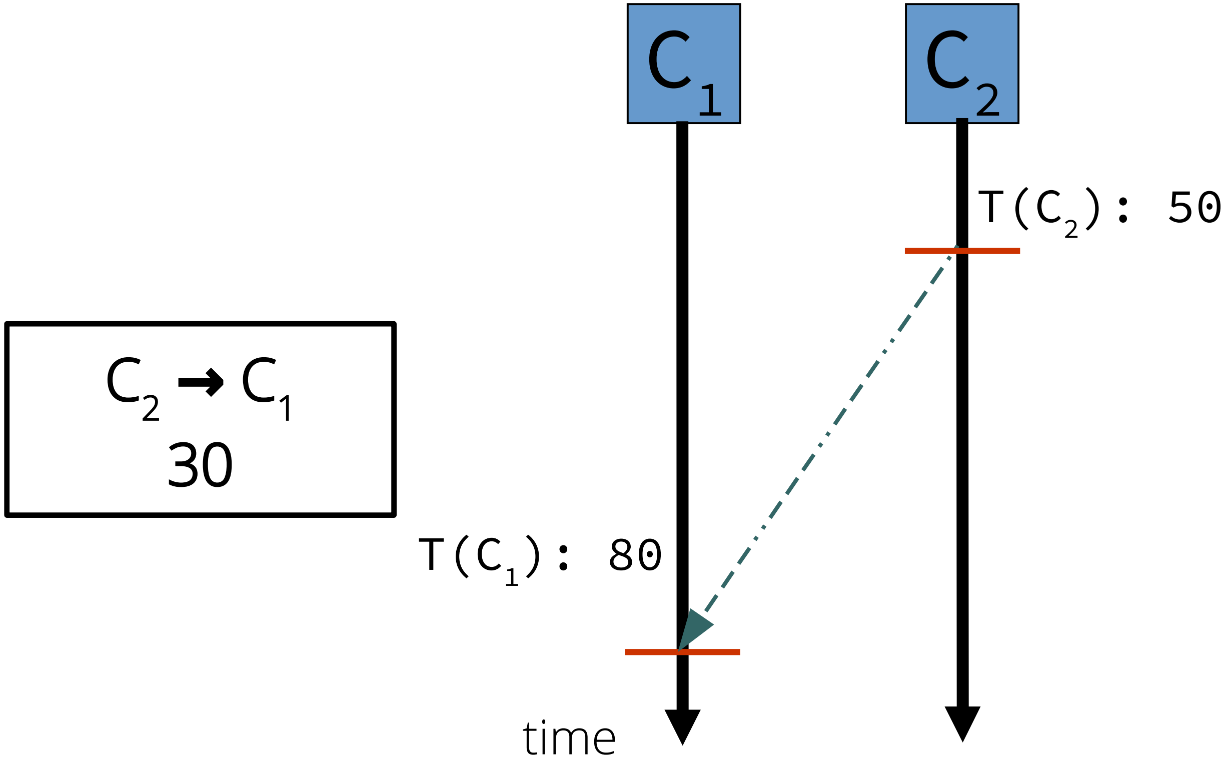
- Add one-way delay latency on each path



1) Calculate C_1 timestamp
2) Notify C_2 via memory
3) Get C_2 timestamp
4) Repeat steps 1-3 to get the minimum

Calculating global uncertainty window: ORDO_BOUNDARY

- Add one-way delay latency on each path

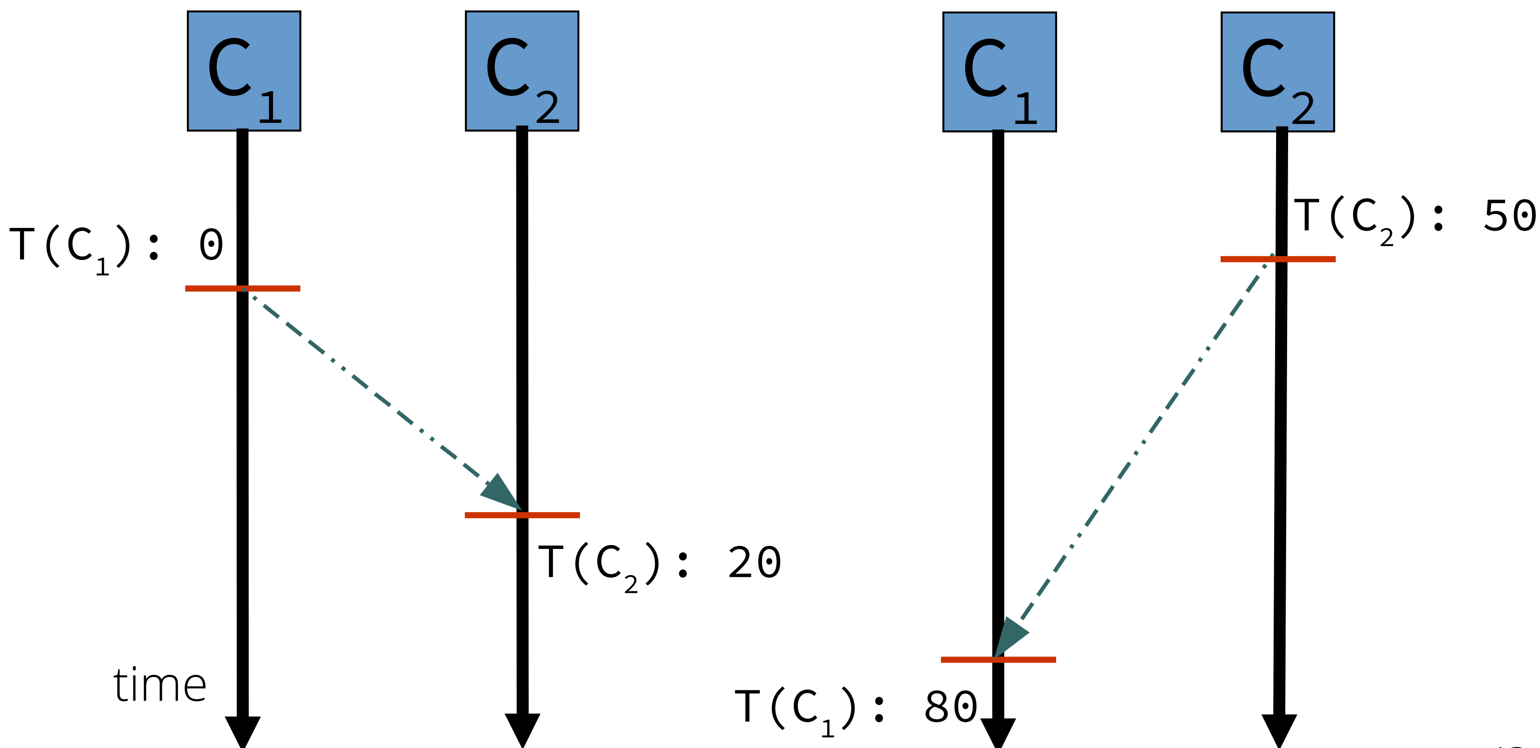


- Repeat prior steps in opposite direction
- Do not know which clock is ahead of the other

Calculating global uncertainty window: ORDO_BOUNDARY

- Repeat steps for each pair of cores from C_1 to C_n
- The maximum offset is the ORDO_BOUNDARY

$C_1 \rightarrow C_2$ 20
$C_2 \rightarrow C_1$ 30
$C_1 \longleftrightarrow C_2$ 30



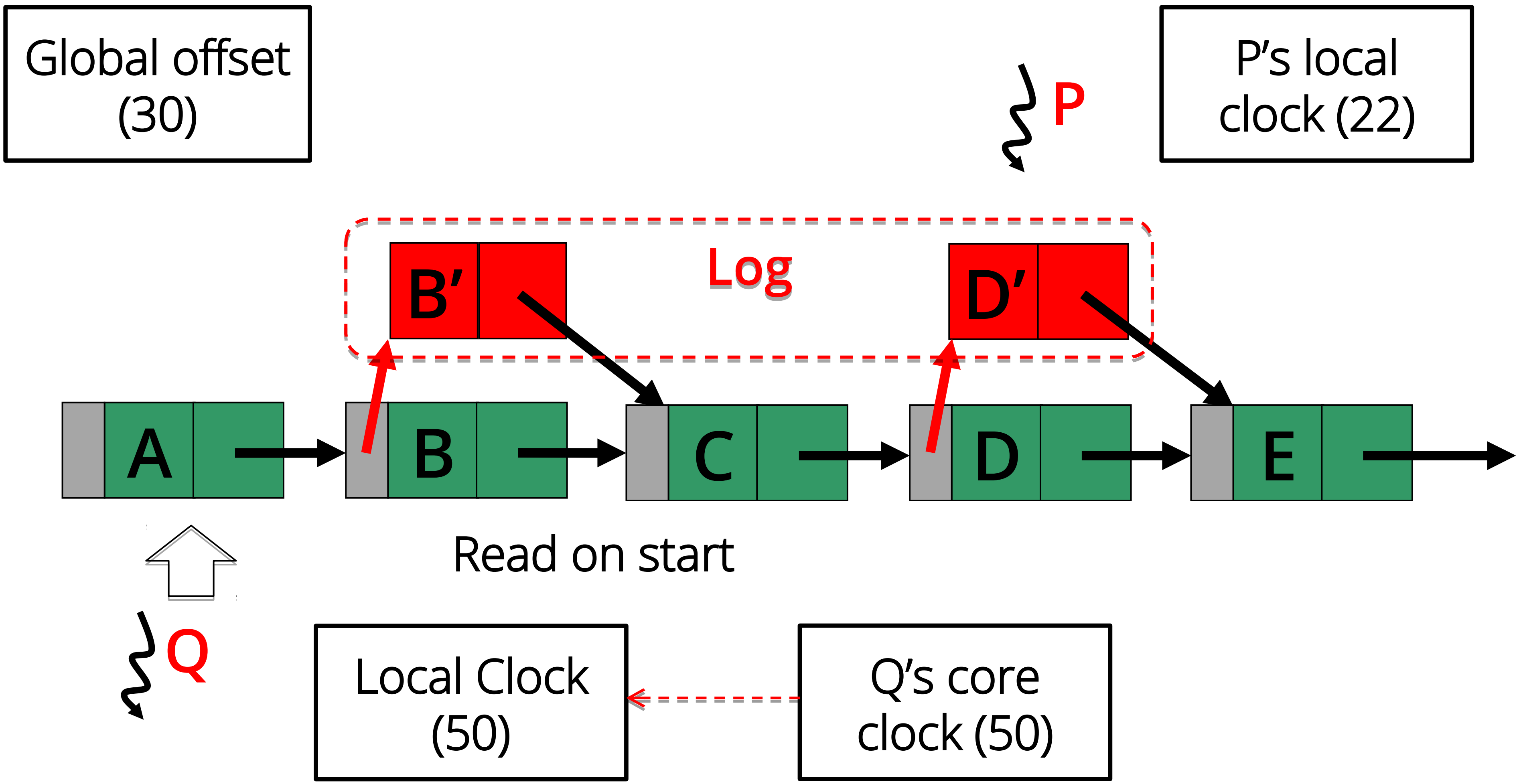
Ordo application

- Applicable to any timestamp-based algorithm
- Expose Ordo API for these algorithms
  - get_time(): Current hardware timestamp
  - cmp_time(t_1, t_2): Compare two timestamps with uncertainty, if $|t_1-t_2|$ < ORDO_BOUNDARY
  - new_time(t): Return t_{new} > (t + ORDO_BOUNDARY)
- **Catch:** Algorithms should handle uncertainty

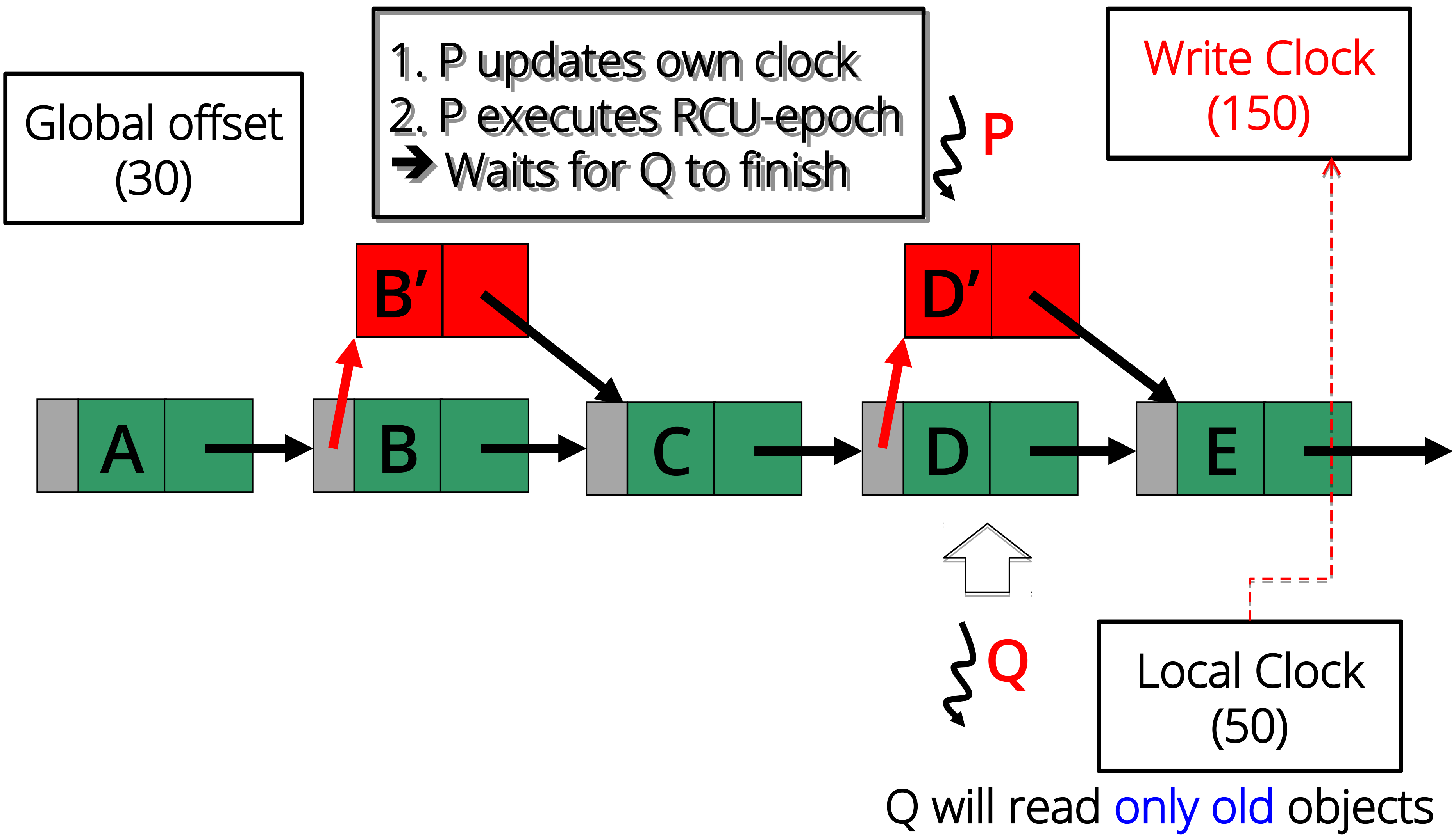
Algorithms with Ordo handling uncertainty

- Physical to logical timestamping:
  - Rely on `cmp_time()` to compare two timestamps
  - Either defer or revert if comparison is uncertain
  - Use `new_time()` to guarantee new time
- Physical timestamping:
  - Use `new_time()` to access the global clock

Read Log Update (RLU^{Ordo}) operation



RLU^{Ordo} commit operation



Algorithms modified with Ordo

- RLU
- Transactional Locking (TL2) in STM
- Database concurrency control: OCC, MVCC
- Oplog used in Linux forking functionality

See our paper

Evaluation

- Questions:
  - Measured global offset (ORDO_BOUNDARY)
  - Maximum scalability of Ordo
  - Ordo’s impact on algorithms
- Machines configuration:
  - 240 core, 8 socket Intel Xeon machine (Xeon)
  - 256 core, Intel Xeon Phi (Phi)
  - 96 core, 2 socket ARM machine (ARM)
  - 32 core, 8 socket AMD machine (AMD)

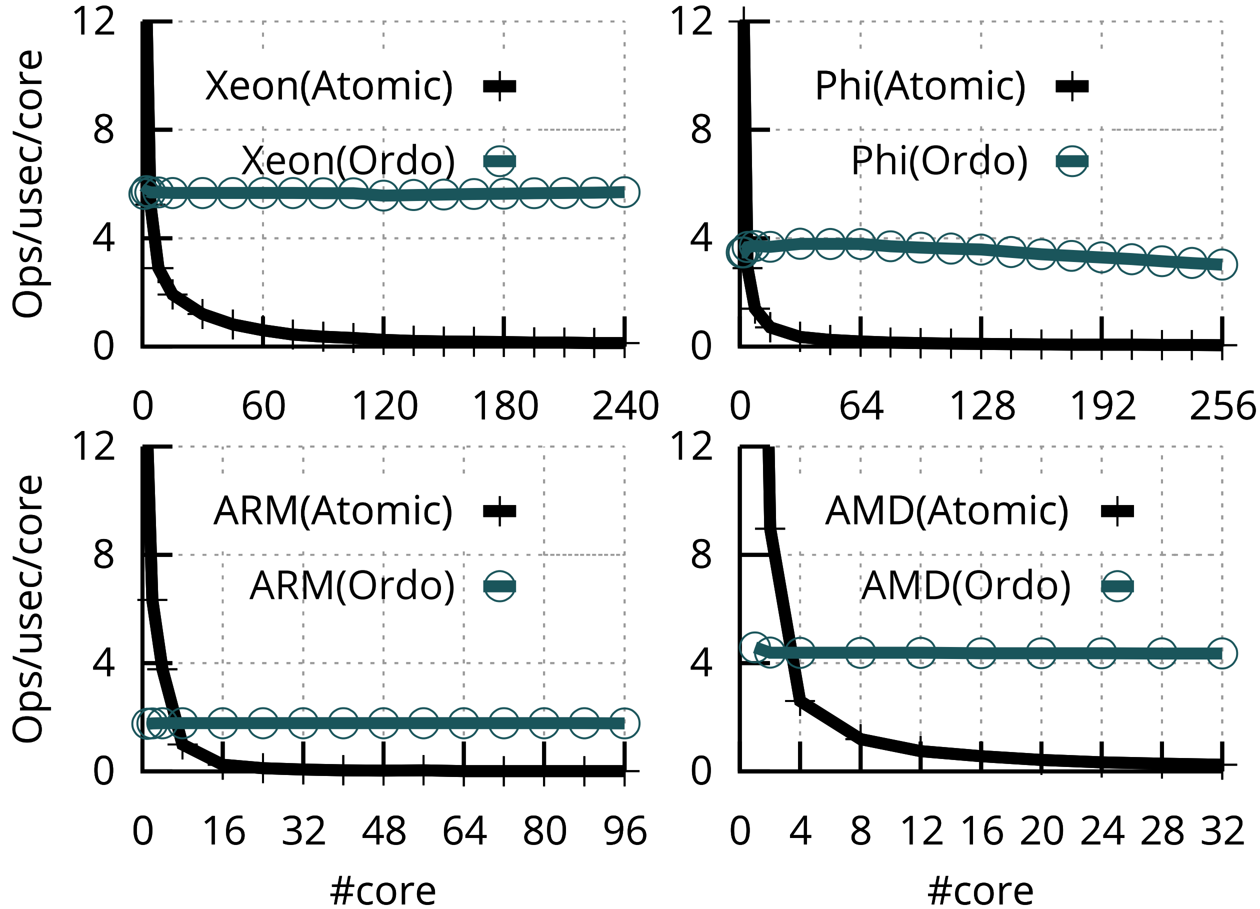
Offset between clocks

- Empirically measured offset after reboots
- ORDO_BOUNDARY is the maximum offset

Machine	Minimum (ns)	Maximum (ns)
Intel Xeon	70	276
Intel Xeon phi	90	270
ARM	100	1,100
AMD	93	203

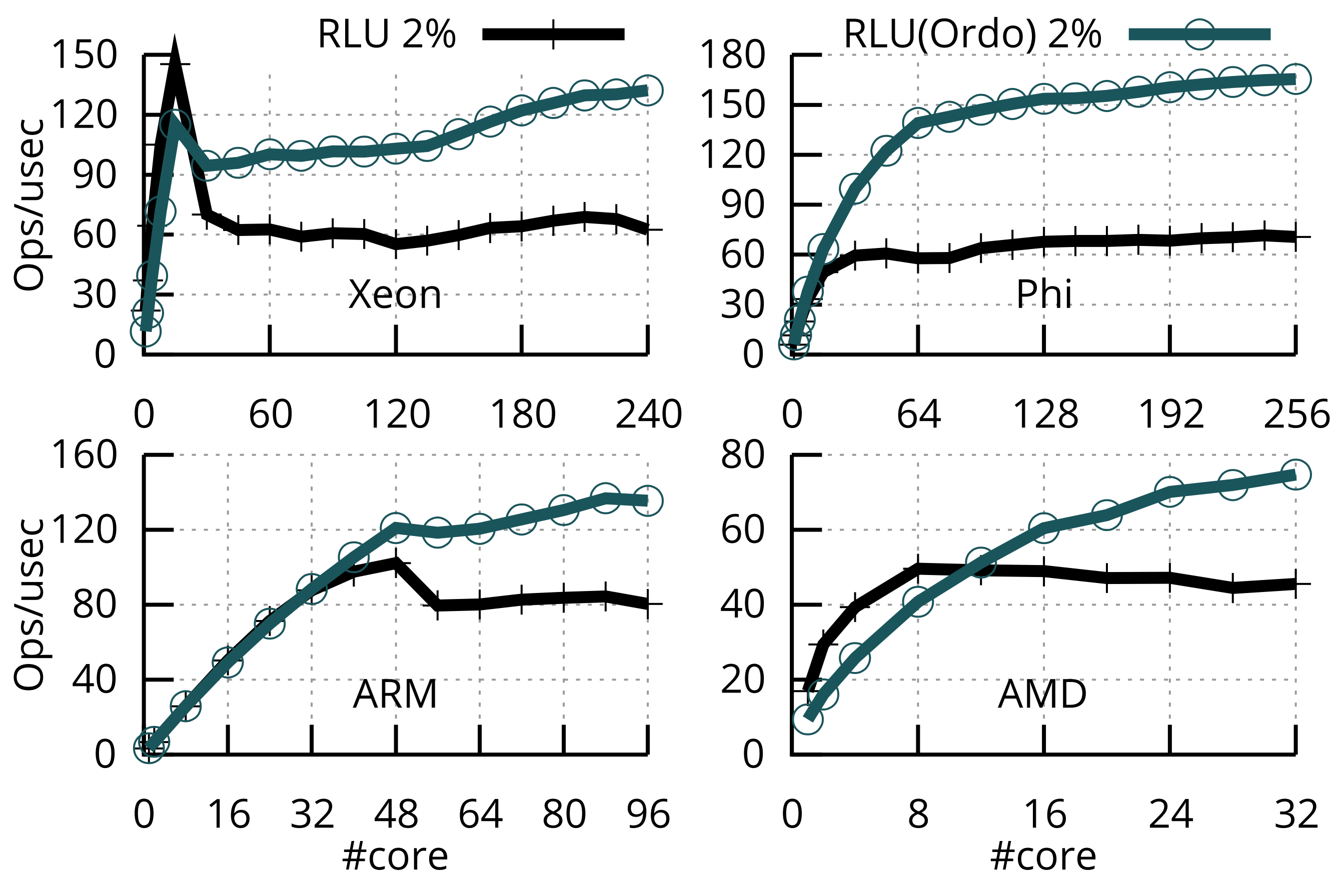
Timestamping with Ordo

- Ordo relies on hardware timestamping
- 17.4 – 285.5x faster than atomic increments



Scaling RLU with Ordo

- RLU^{Ordo} is 2.1x faster on an average
- Still suffers from object copy and its locking



Discussion and limitations

- Simplifies the design and understanding of algorithms
- Not a panacea
  - Applicable when clock is contentious
- No skew consideration
- Thread ID-based timestamp comparison has its limitation

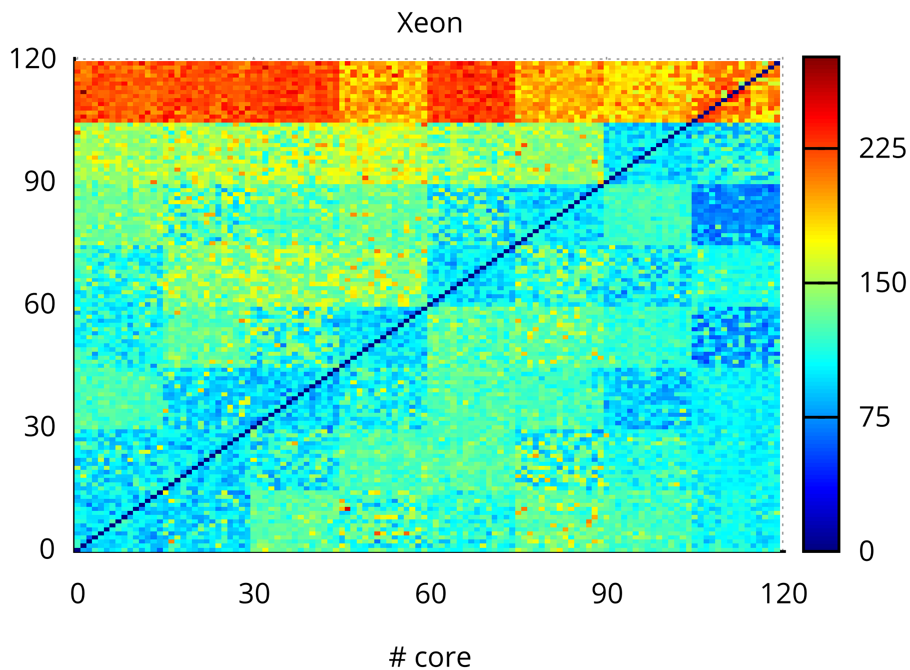
Conclusion

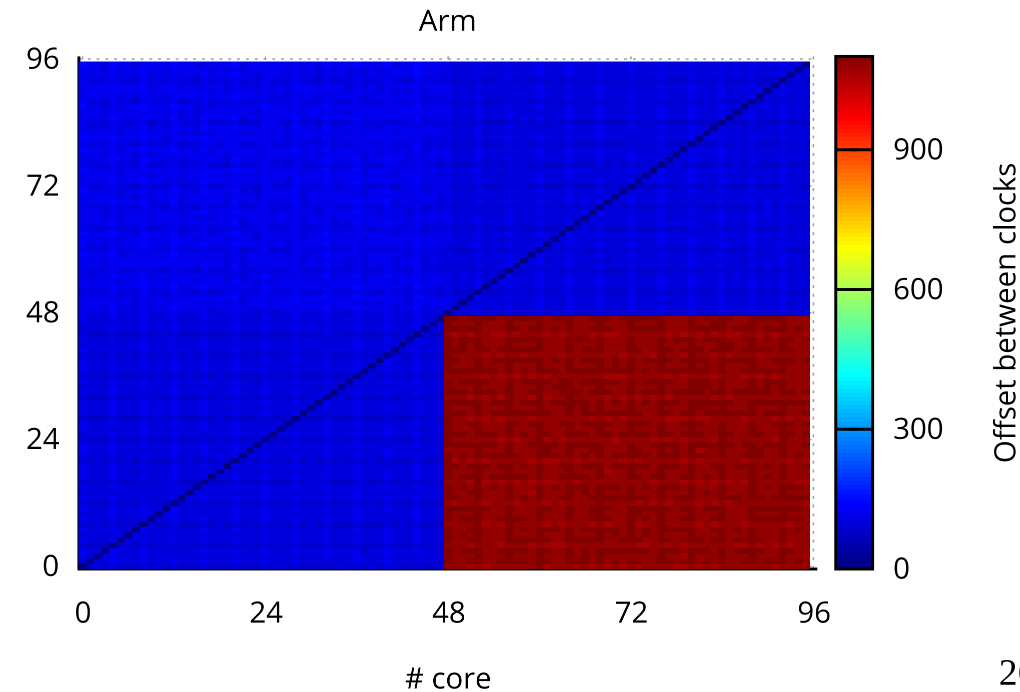
- Ordo is a scalable timestamping primitive
  - Relies on invariant hardware clocks
- Exposes time-based API to the user
- Applied Ordo to five concurrent algorithms
- Improves the scalability of algorithms by at most 39.7x across architectures

Backup Slides

Offset between clocks

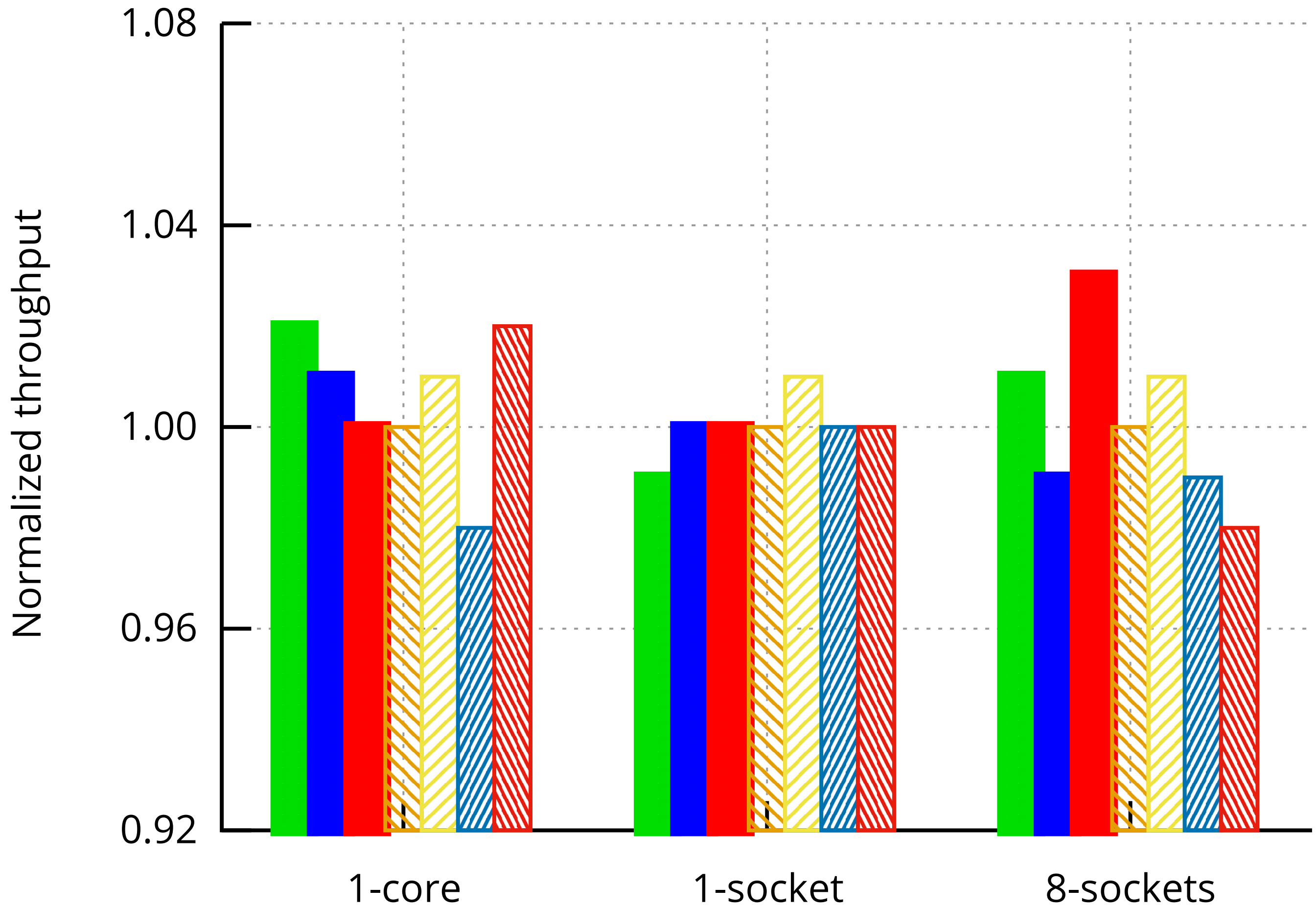
- Clocks are not synchronized
  - 8th socket in Xeon and 2nd socket in ARM
  - Results remain consistent even after reboots and measuring after a period of time





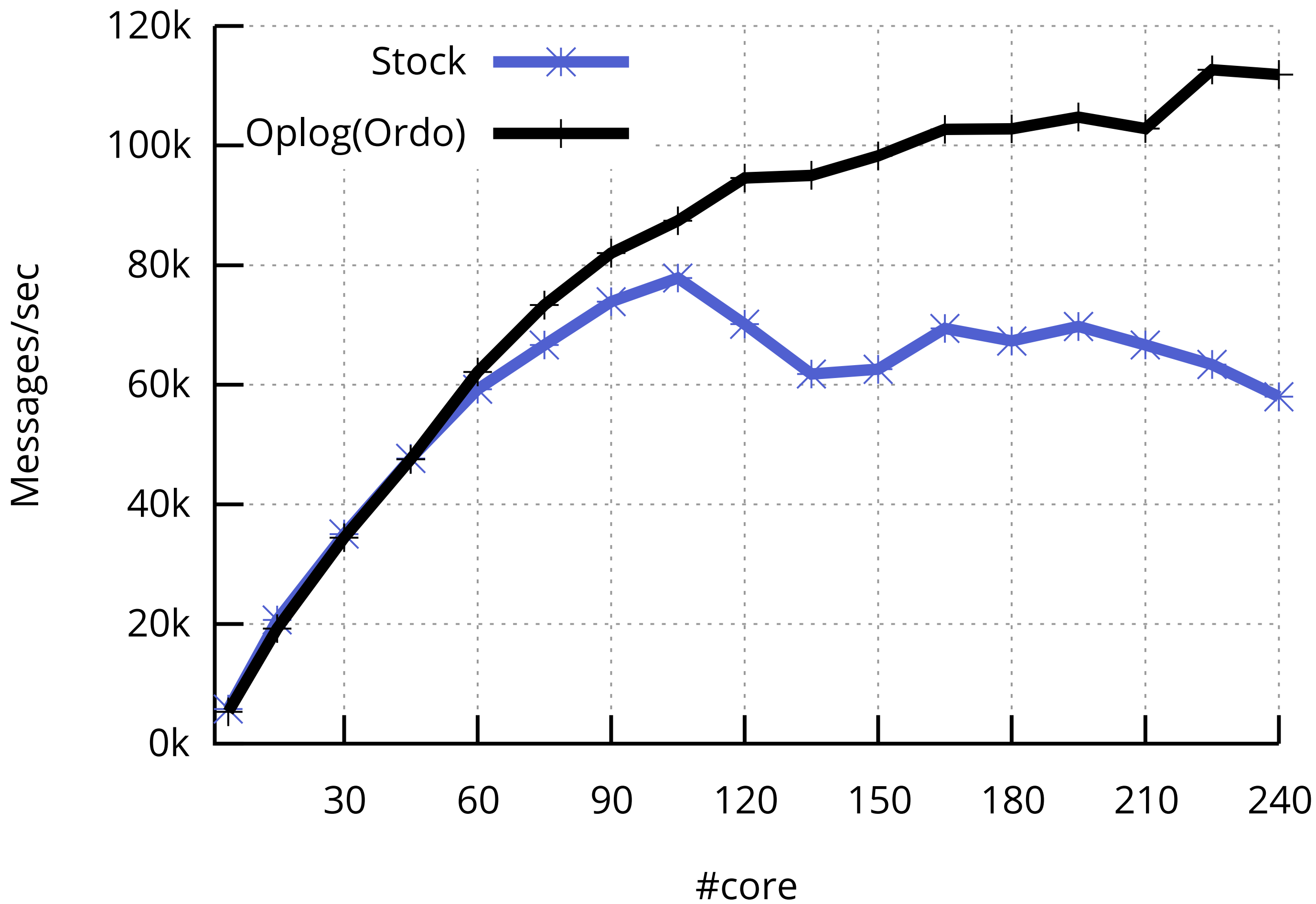
Sensitivity of ORDO_BOUNDARY

- Varying ORDO_BOUNDARY from 1/8x – 8x
- Cycles increases from 32.2–18K on Xeon machine



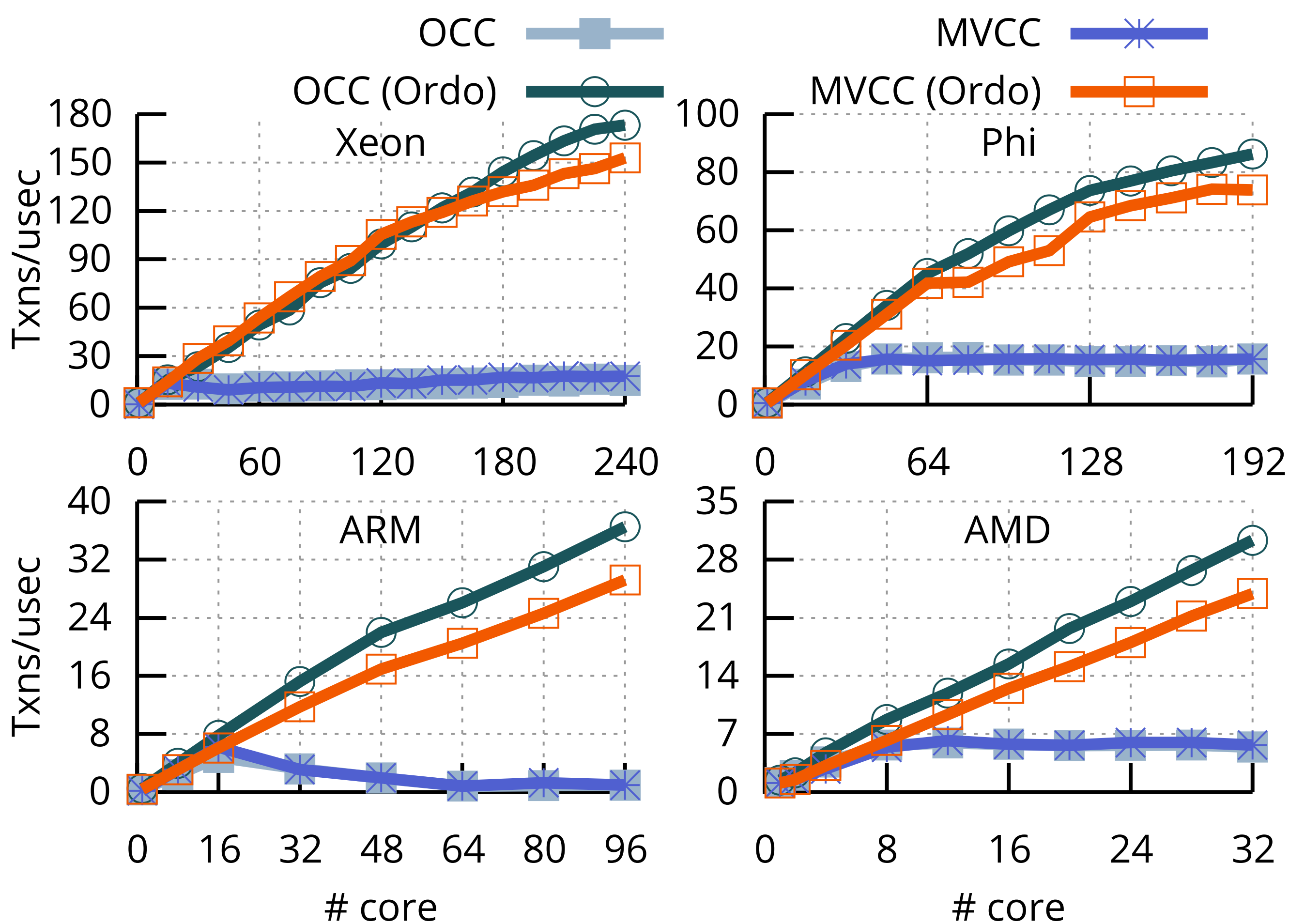
Physical timestamping: Oplog

- Improves Exim performance by 1.9x at 240 cores



Scaling database concurrency control

- Improves OCC and MVCC by 4.1–39.7x for read-only (YCSB)
- OCC^{Ordo} 1.24x faster than Tictoc and Silo (TPC-C)



Cannot use clock synchronization protocols

- No information on minimum bounds on message delivery between/among clocks
- Protocols introduce various errors
- Can lead to mis-synchronized clocks
  - Larger or smaller than the actual physical offset

Lead to incorrect implementation of concurrent algorithms