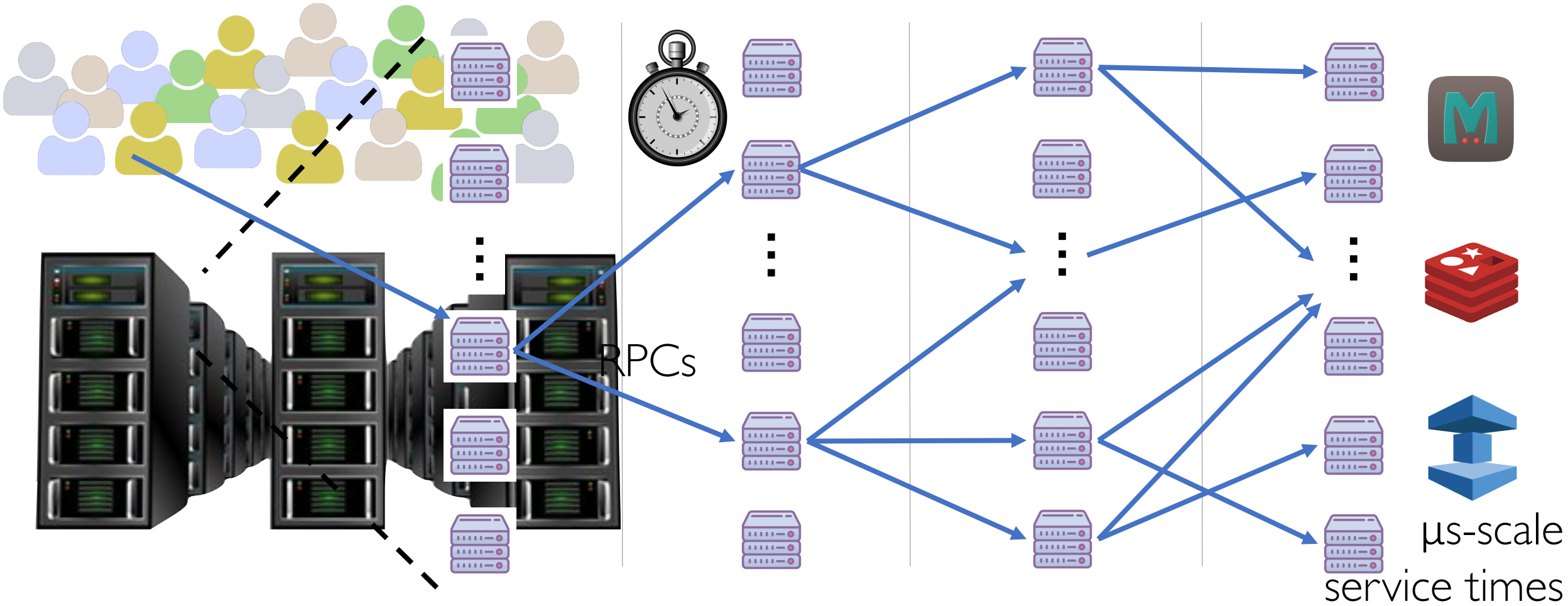
RPCValet: NI-Driven Tail-Aware Balancing of µs-Scale RPCs

Alexandros Daglis
Georgia Tech

Mark Sutherland, Babak Falsafi
EcoCloud, EPFL

Latency-Sensitive Online Services



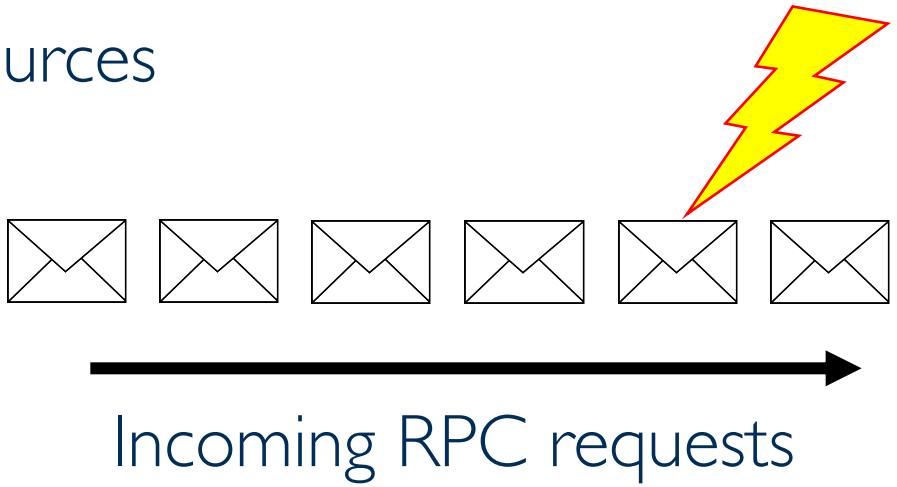
µs-scale RPCs exacerbate tail latency challenge

Sources of Tail Latency

Tail latency has many sources:

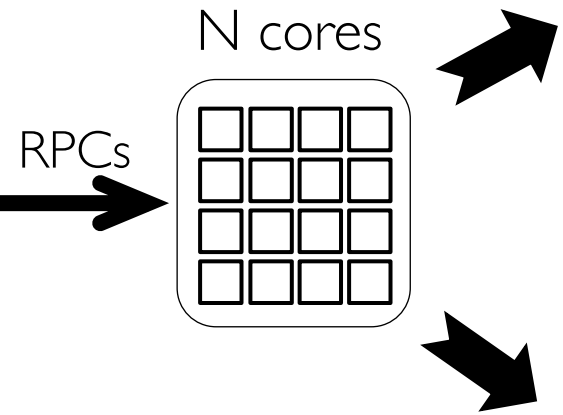
- Software events: interrupts, context switches
- Hardware-related events: cache/TLB misses, page faults, interference, …
- Queuing

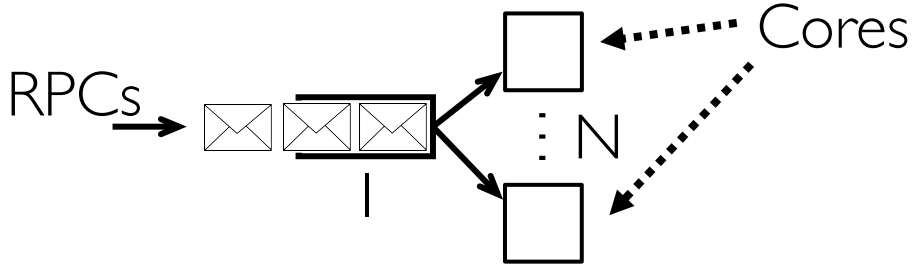
Queuing amplifies effect of ALL other sources



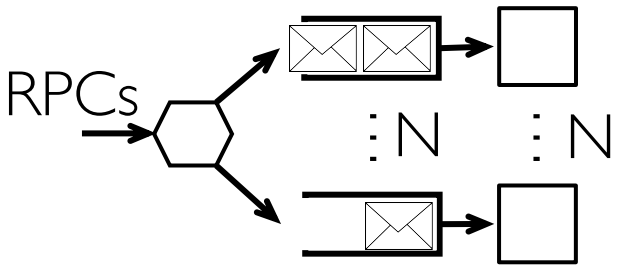
Queuing: prime tail latency optimization target

Queuing Implications on Manycore Servers

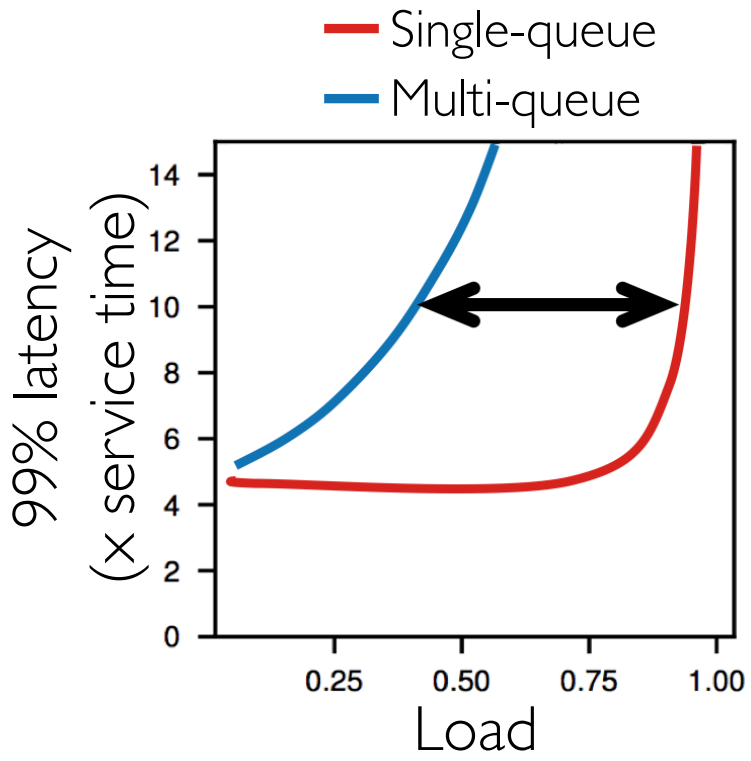




Single-queue



Multi-queue

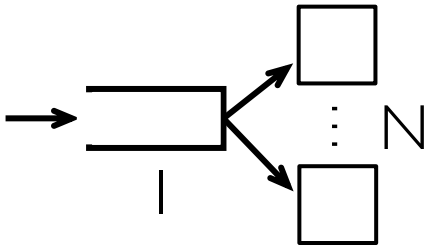


→ load imbalance that hurts tail latency

Single-queue: the best FCFS queuing system (in theory)

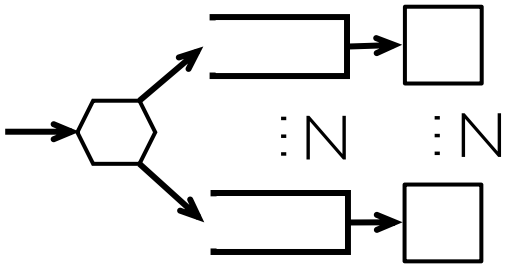
From Theory to Practice

Single-queue



✓ Best load balancing
✗ Synchronization overhead

Multi-queue



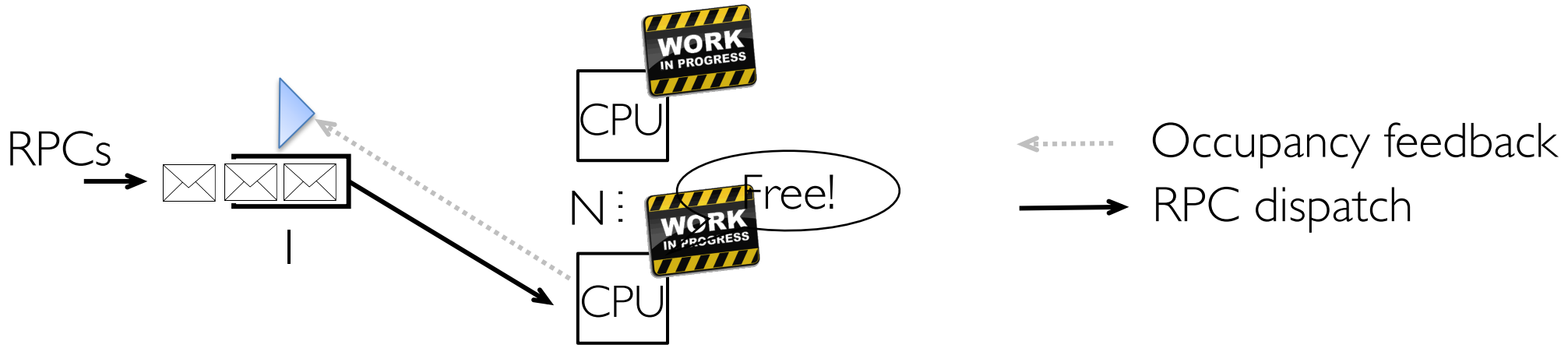
✗ Vulnerable to load imbalance
✓ Synchronization-free

Sync required for single-queue comparable to runtime of µs-scale RPCs

Goal: load balancing without synchronization overhead

RPCValet in a Nutshell

- Leverage integrated NI ( ) to monitor real-time per-core load
- NI-core coordination in 10s of ns
- Keep RPCs in *single* queue & dynamically *push* first RPC to first available core



Single-queue & sync-free load balancing

Outline

Overview

Background

RPCValet Design & Implementation

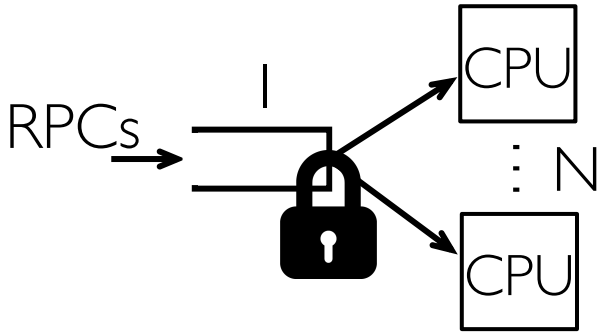
Evaluation

Conclusion

Single-Queue Load Distribution

Common load distribution implementation

- E.g., Linux **`poll`**, **`libevent`**'s locked event queues



RPCs arrive in single queue

- Cores pull RPCs in FIFO order

Queue is shared resource: need synchronization

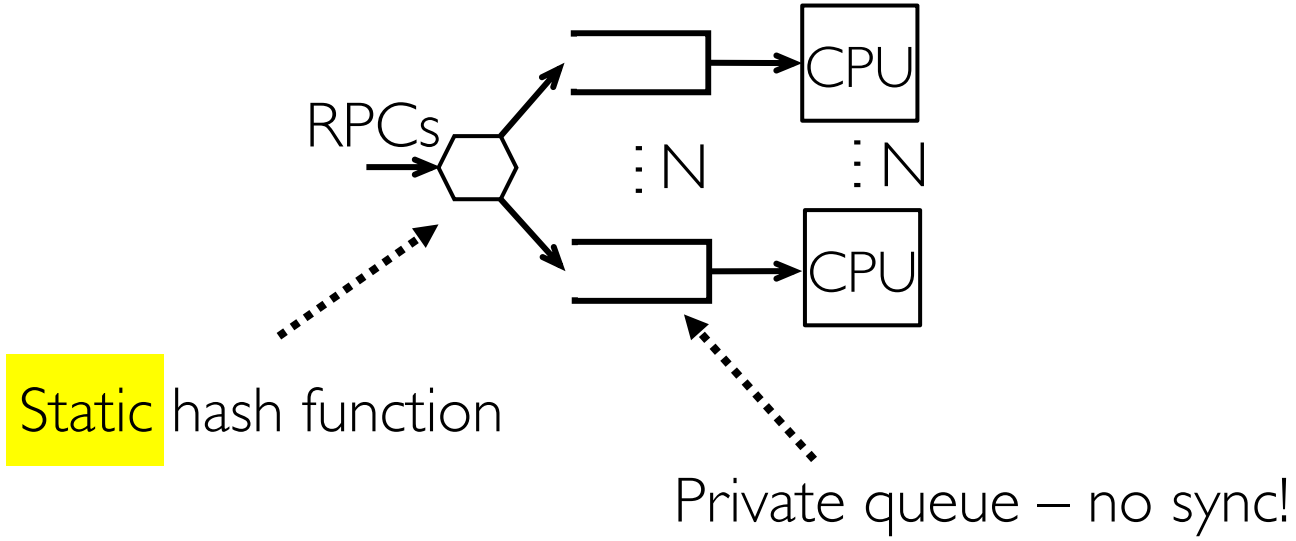
- Minor concern for typical RPCs (ms runtimes)
- Significant overhead for µs-scale RPCs

Sync overhead hurts fine-grained RPCs

Multi-Queue Load Distribution

Receive Side Scaling: Hardware support for multi-queue load distribution

- Leveraged by dataplanes (e.g., IX [Belay'14], Arrakis [Peter'14])



Distribution based on static decisions ≠ Balancing

From Load Distribution to Load Balancing

Need **dynamic** load dispatch decisions

- Rebalancing via work stealing helps, but still significant cost for µs-scale RPCs
- E.g., ZygOS [Prekas'17] >30% perf. gap from single-queue system for Memcached

Insight: leverage integrated NI for rapid feedback

On-chip NI facilitates dynamic load-balancing decisions

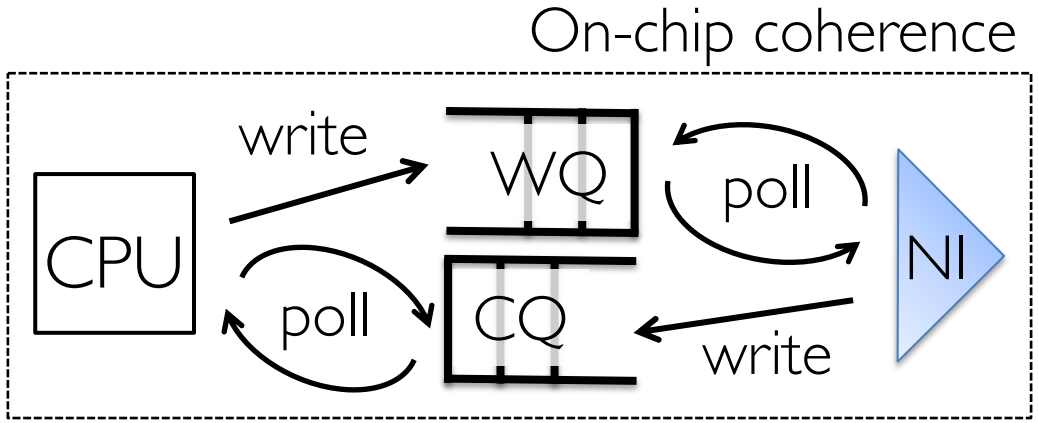
Base Architecture: Scale-Out NUMA

Architecture for rapid remote memory access [Novakovic'14]

Lean user-level, hardware-terminated protocol & integrated NI

RDMA-like hardware-software interface

- New requests in Work Queue (WQ)
- Replies in Completion Queue (CQ)



Basic primitives: one-sided reads/writes

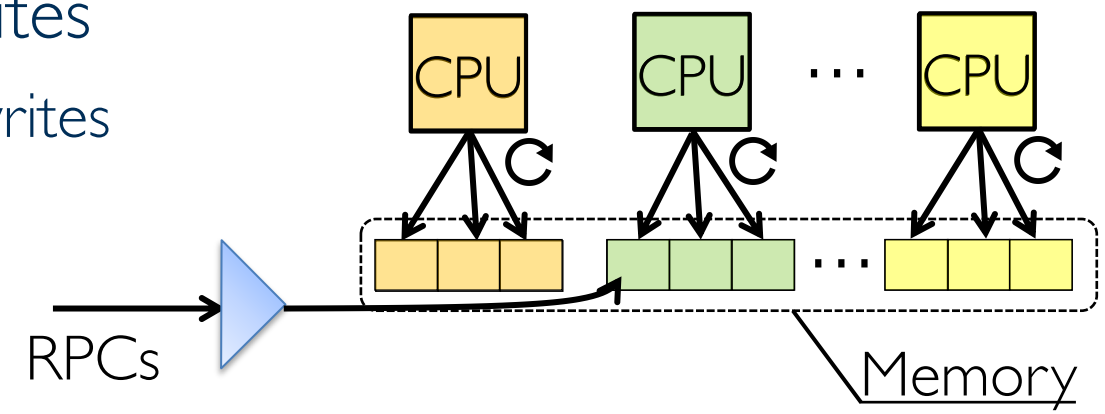
- Messaging emulated over one-sided writes

Lack of native messaging roadblock for RPC balancing

Limitation of Emulated Messaging

HERD [Kalia'14]: Fast RPCs over RDMA writes

- Write RPCs in remote memory w/ one-sided writes
- Cores poll on all possible RPC arrival locations
- Sync-free



Problem: Message arrival location dictates RPC-to-core assignment

→ multi-queue system by design

One-sided writes → multi-queue system → imbalance

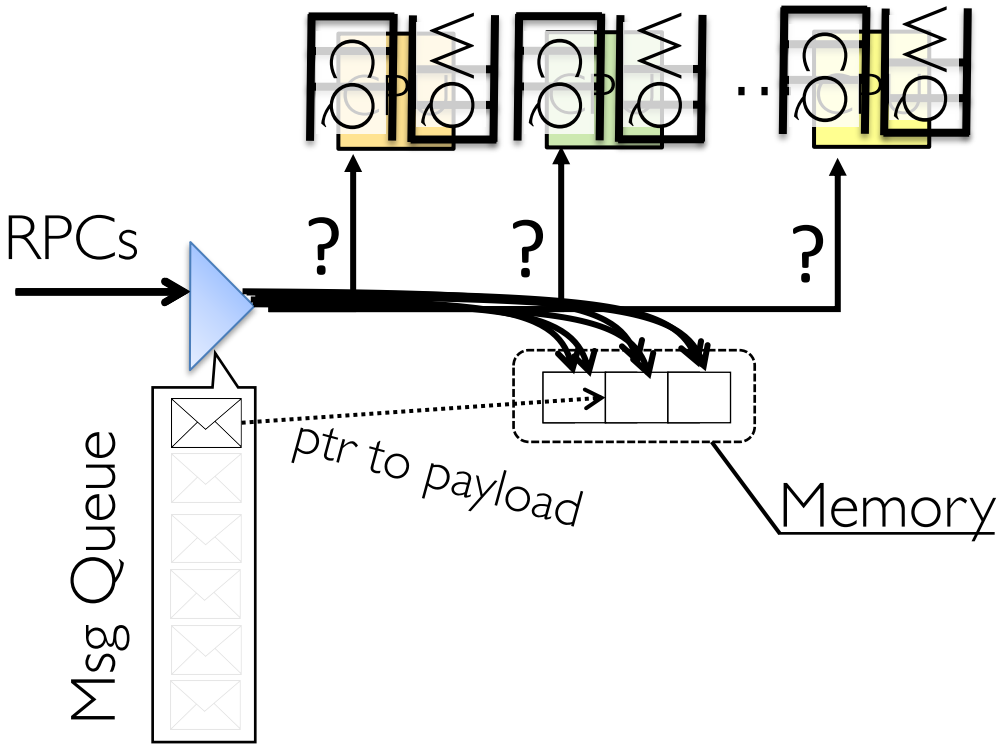
Enabling Single-Queue Load Balancing

Decouple RPC arrival from assignment to core

- Order arrival metadata, not RPC payload

Dispatch RPCs to cores in order

- *Push* instead of pull – no sync required
- When? To which core?



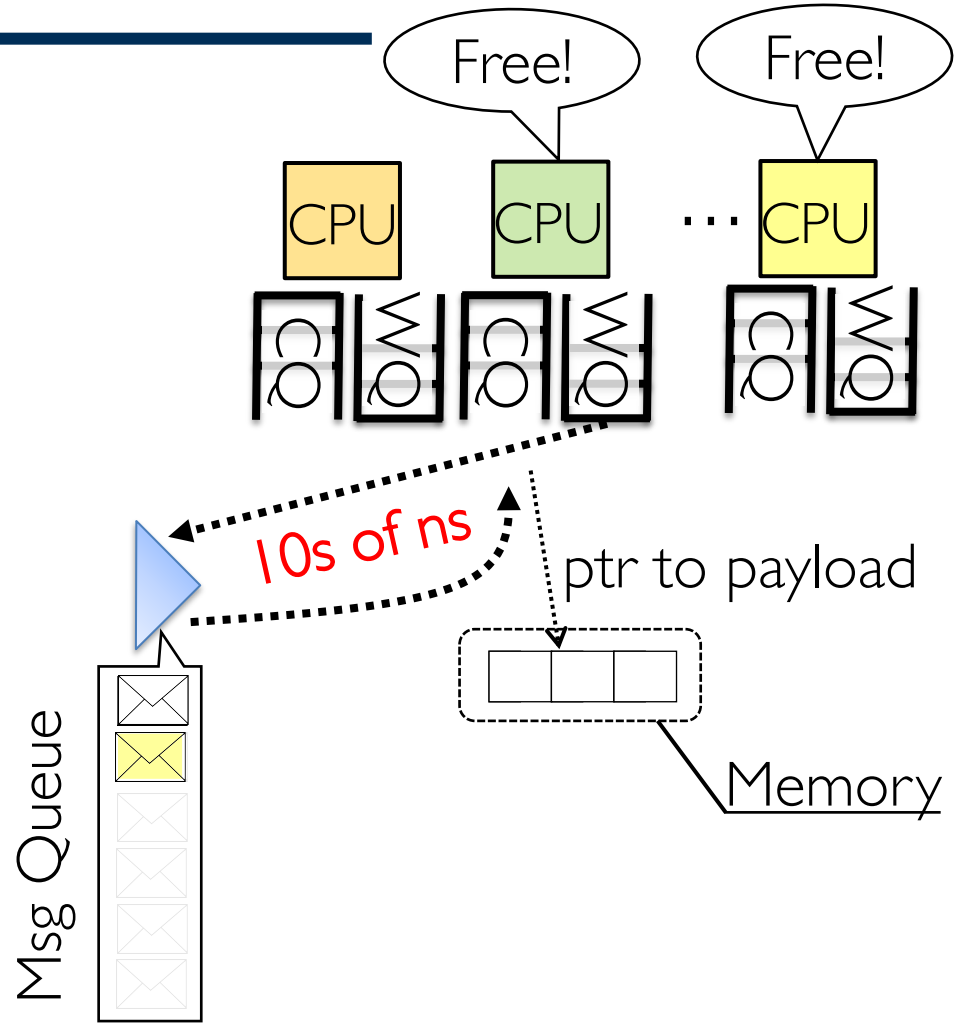
Enabling Single-Queue Load Balancing

Dispatch to 1st available core → true single-queue

- Cores self-signal availability via special msg in WQ

Integrated NI makes simple greedy dispatch viable

- On-chip message propagation << RPC service time
  → Execution bubbles sufficiently small



RPCValet: sync-free, single-queue load balancing

Balancing Policy vs. Throughput

Georgia Tech

NI dispatch stage has to sustain peak throughput

- Need to sustain max service rate, not line rate
- For 500ns RPCs & 64-core chip → 1 dispatch decision / 8ns

Trivial for RPCValet's greedy dispatch policy

- Read a 64-bit bitmap, pick available entry

Could implement more sophisticated dispatch policies

- Constraint: perform decision in 8ns or pipeline logic

RPCValet applicability not limited to greedy dispatch

Outline

Overview

Background

RPCValet Design & Implementation

Evaluation

Conclusion

Methodology

Cycle-accurate simulation of 16-core chip

Poisson arrivals & emulated RPC service time distributions

- Service time: mean 1µs & increasing variance (fixed, uni, exp, GEV)
- HERD and Masstree Key-Value stores (in the paper)

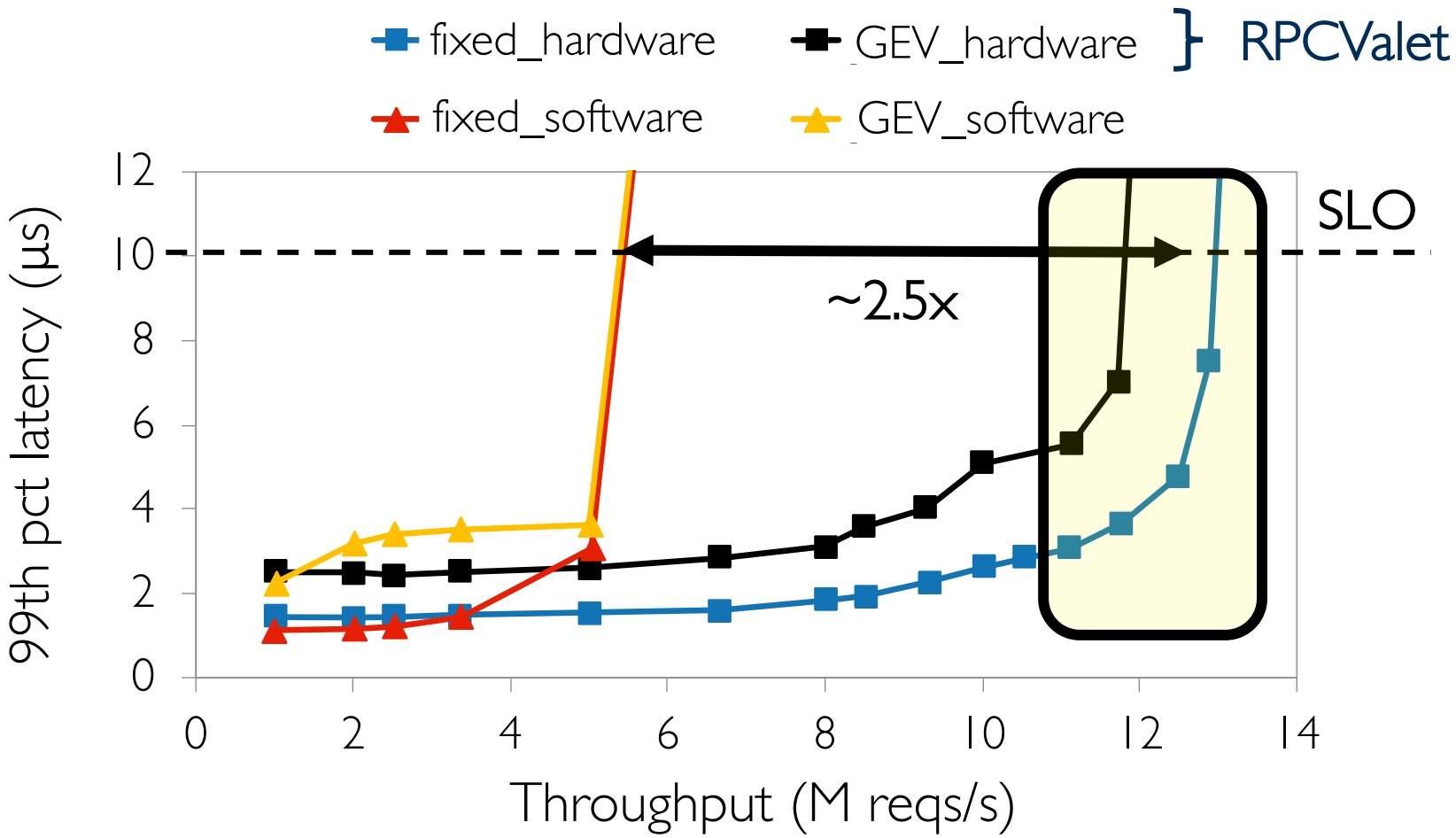
Metric: throughput under SLO (target: 10µs 99th pct latency)

- Server-side latency measurements

Configurations:

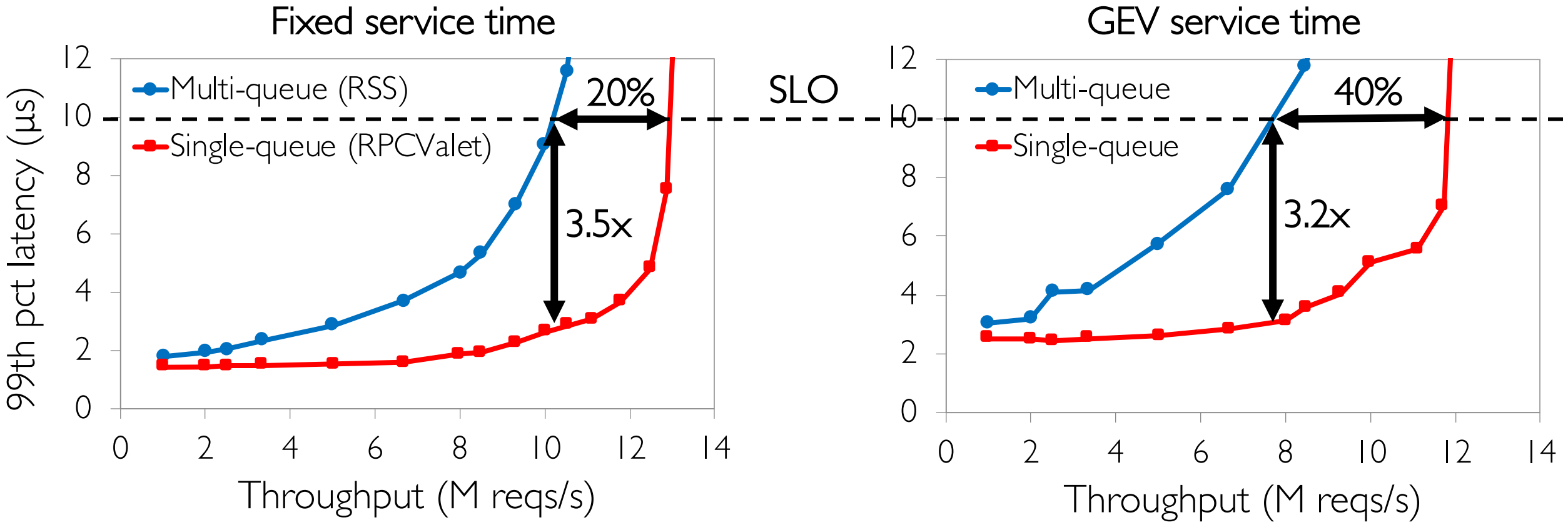
- Single-queue system w/ software synchronization (MCS queue lock)
- Hardware-dispatched multi-queue system (Receive Side Scaling – RSS)
- Hardware-dispatched single-queue system (RPCValet)

Single-Queue: Hardware vs. Software



Synchronization overhead severely hurts µs-scale RPCs

Hardware: Multi-Queue vs. Single-Queue



Up to 3.5x lower tail latency & 1.4x throughput under SLO

Conclusion

Georgia Tech

µs-scale RPCs exacerbate queuing-related tail latency challenge

Single-queue systems avoid load imbalance but require synchronization

RPCValet: sync-free single-queue load balancing

- Leverage NI integration for rapid dynamic dispatch decisions
- Up to 40% higher throughput under SLO vs. RSS
- Up to 3.5x lower tail latency at medium load vs. RSS

Thanks! Questions?