USENIX ATC 2016

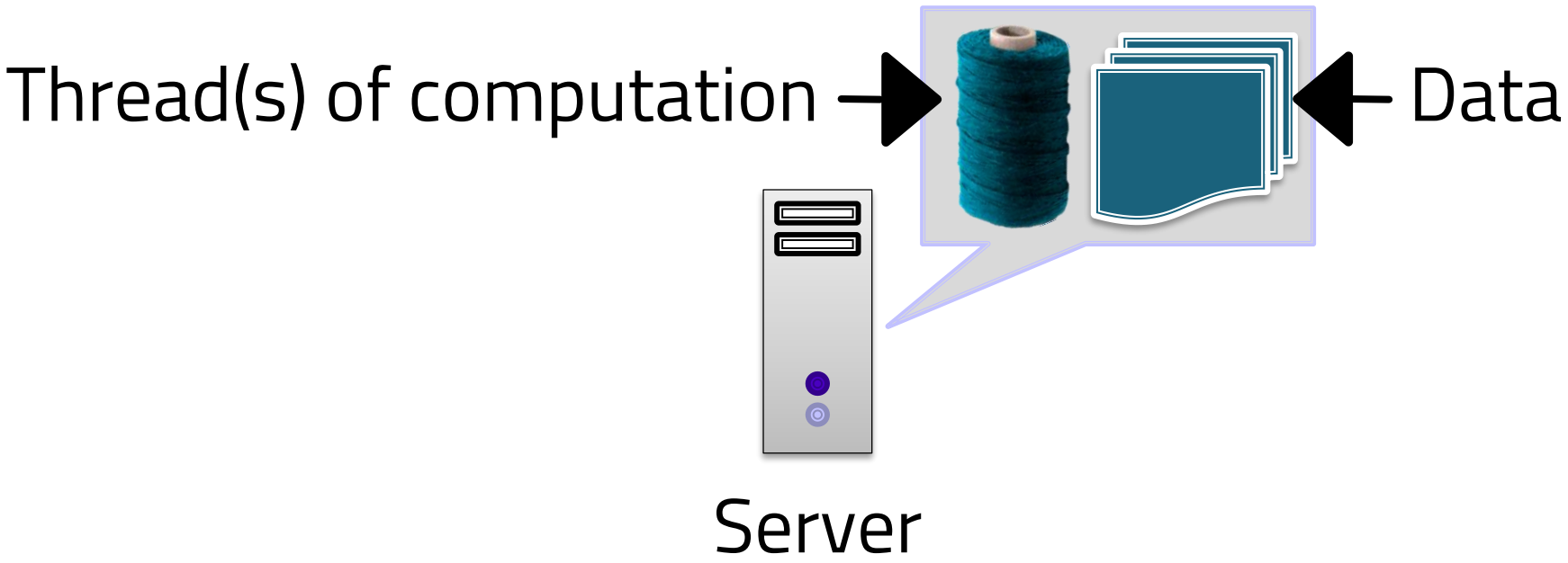
Balancing CPU and Network in the Cell Distributed B-Tree Store

Christopher Mitchell, Kate Montgomery, Lamont Nelson, Siddhartha Sen*, Jinyang Li

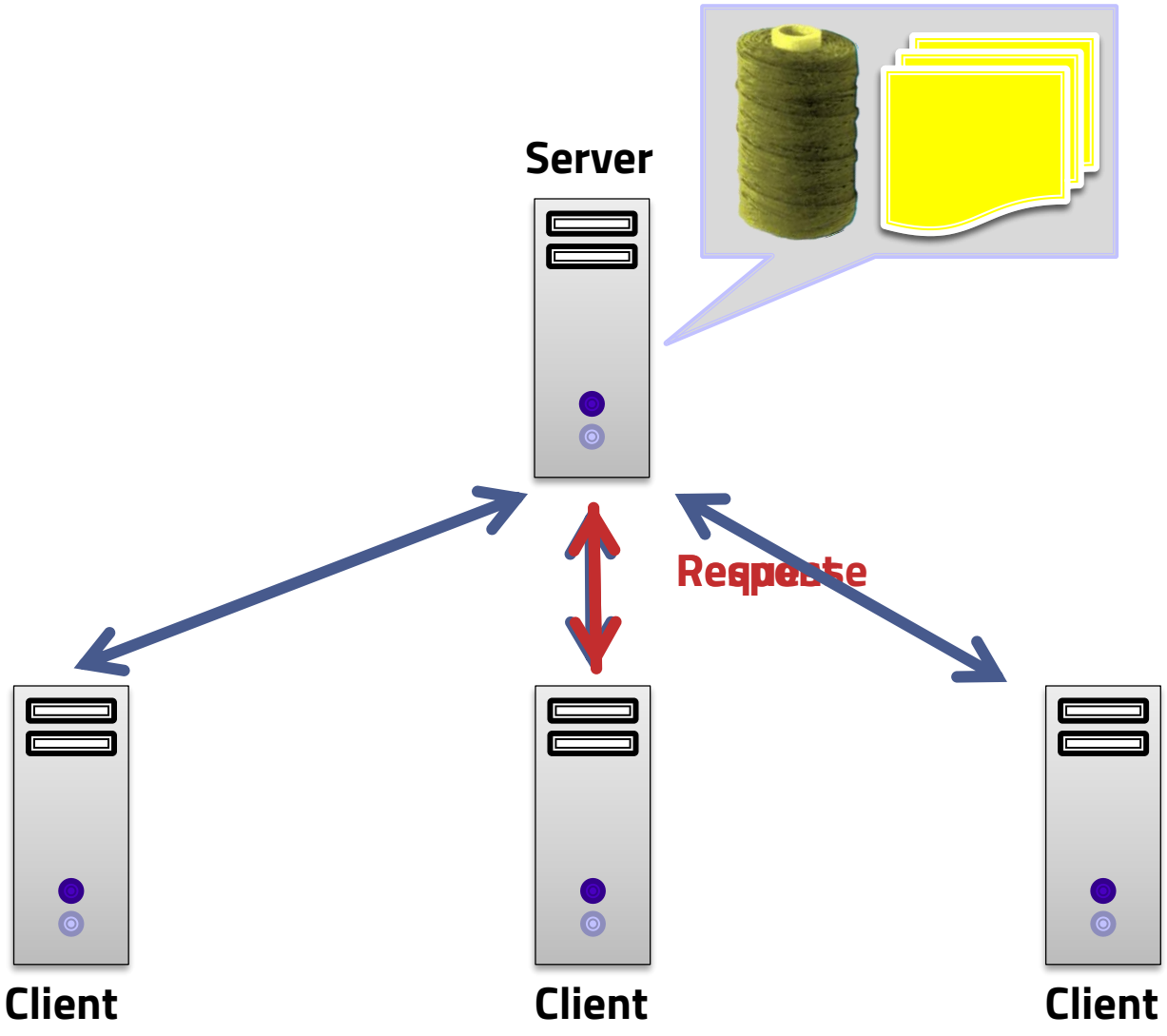
New York University, *Microsoft Research

June 23, 2016

Traditional Client-Server System…



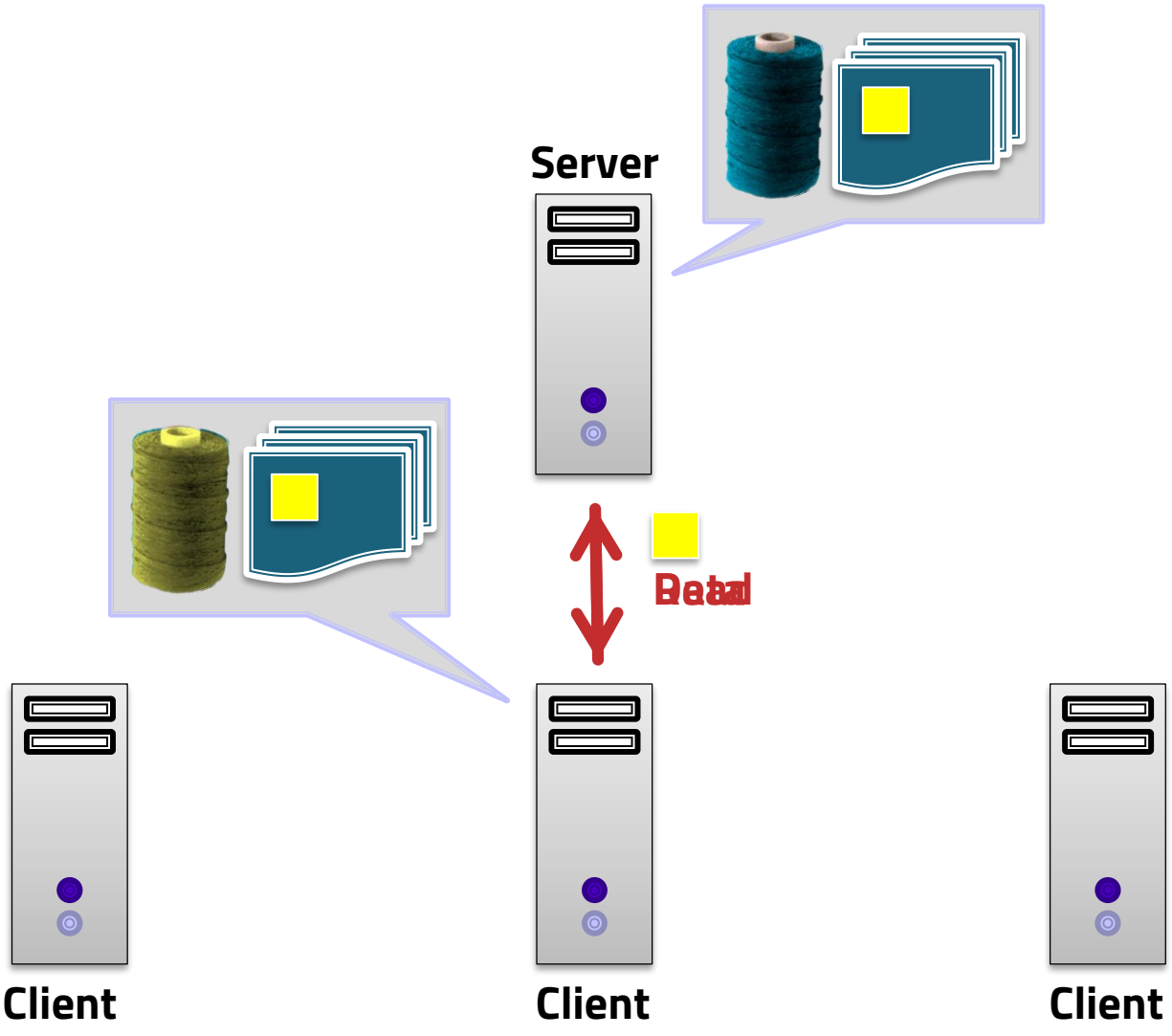
...Maintains Locality of Data and Computation



Problem: Server CPU-Bound

- Load spikes saturate server CPUs
- Options
  1. Over-provision server CPUs (wasteful)
  2. Spin up extra servers during spike (slow)
- Solution: Relax locality by processing requests at client?
  - Clients fetch the required server state

RDMA Enables Client-Side Processing

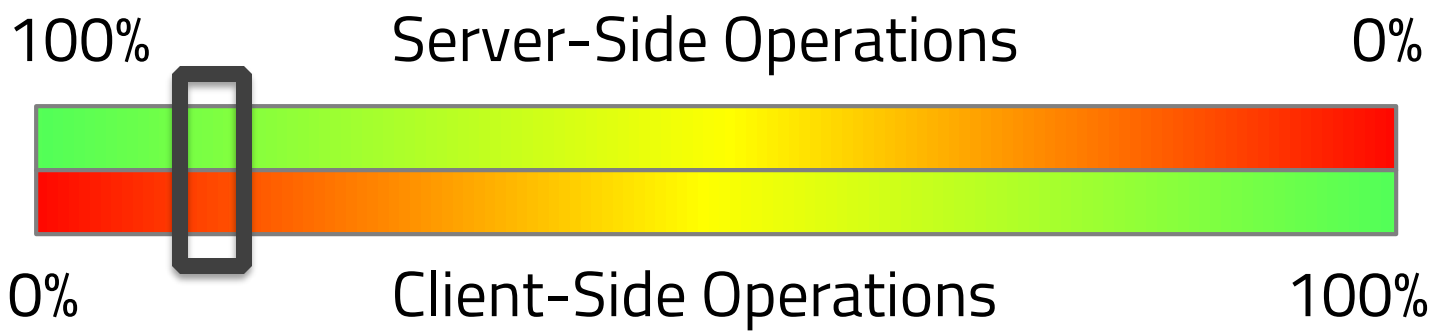


Choosing Client-Side vs. Server-Side Processing

- Server CPU bottleneck -> use client-side
- NIC bottleneck -> use server-side
  - (If you have excess server CPU, just use it)

Selectively Relaxed Locality

- Combining client-side and server-side operations
- **Insight:** *Selectively* relaxing locality improves load balancing



Cell: A Distributed B-Tree Store

1. Distributed, sorted, RDMA-enabled store
2. Selectively relaxed locality to improve load balancing and CPU efficiency.
3. Dynamic locality selector

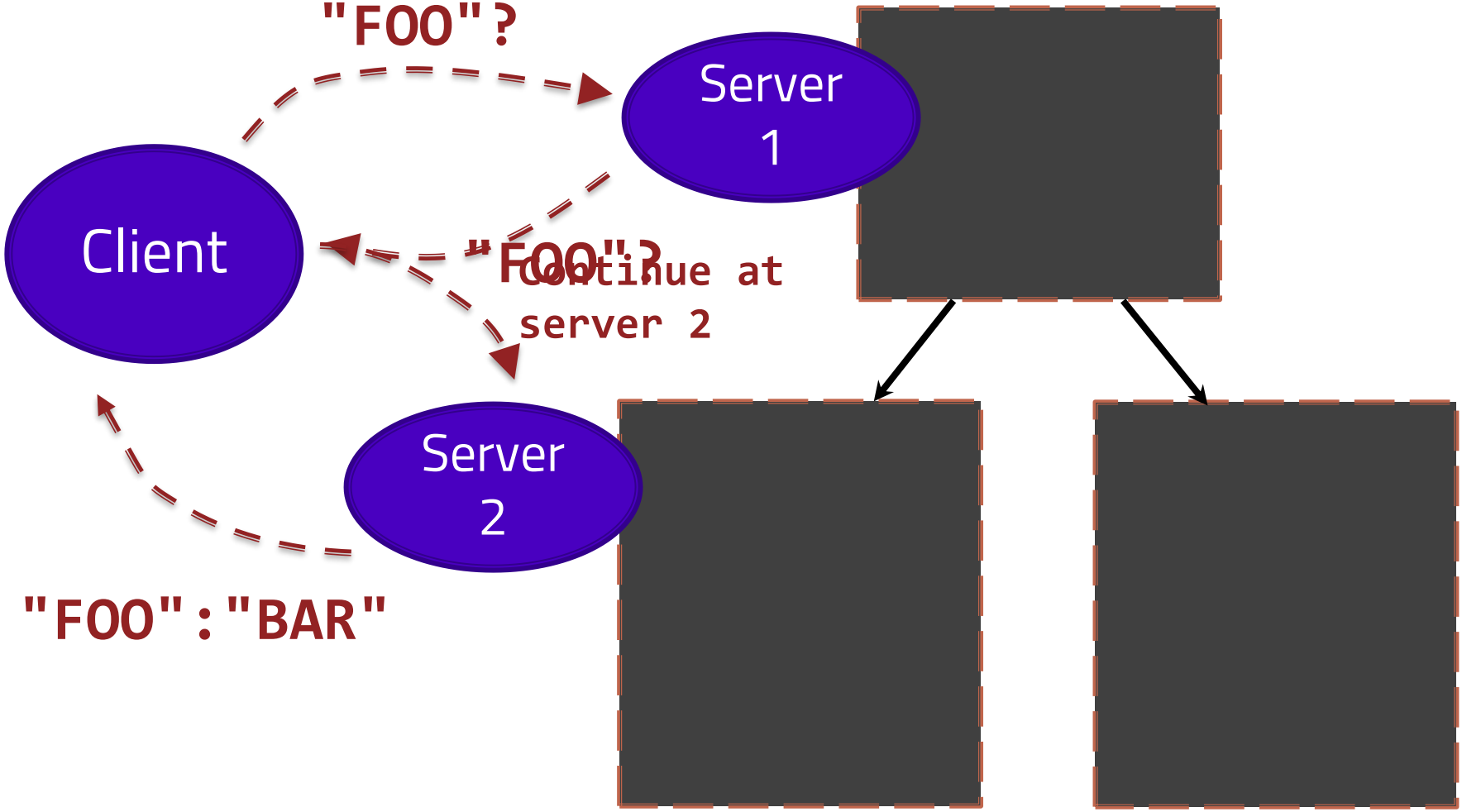
Outline

- Motivation: Selectively Relaxed Locality
- Cell Distributed B-tree
- Evaluation
- Related Work

Outline

- Motivation: Selectively Relaxed Locality
- Cell: Balancing Server-Side & Client-Side Search
  1. Making Client-Side and Server-Side Operations Efficient
  2. Ensuring Correctness During Operations
  3. Choosing Client-Side or Server-Side Search
- Evaluation
- Related Work

Today's Distributed Sorted Stores



Today's Sorted Stores

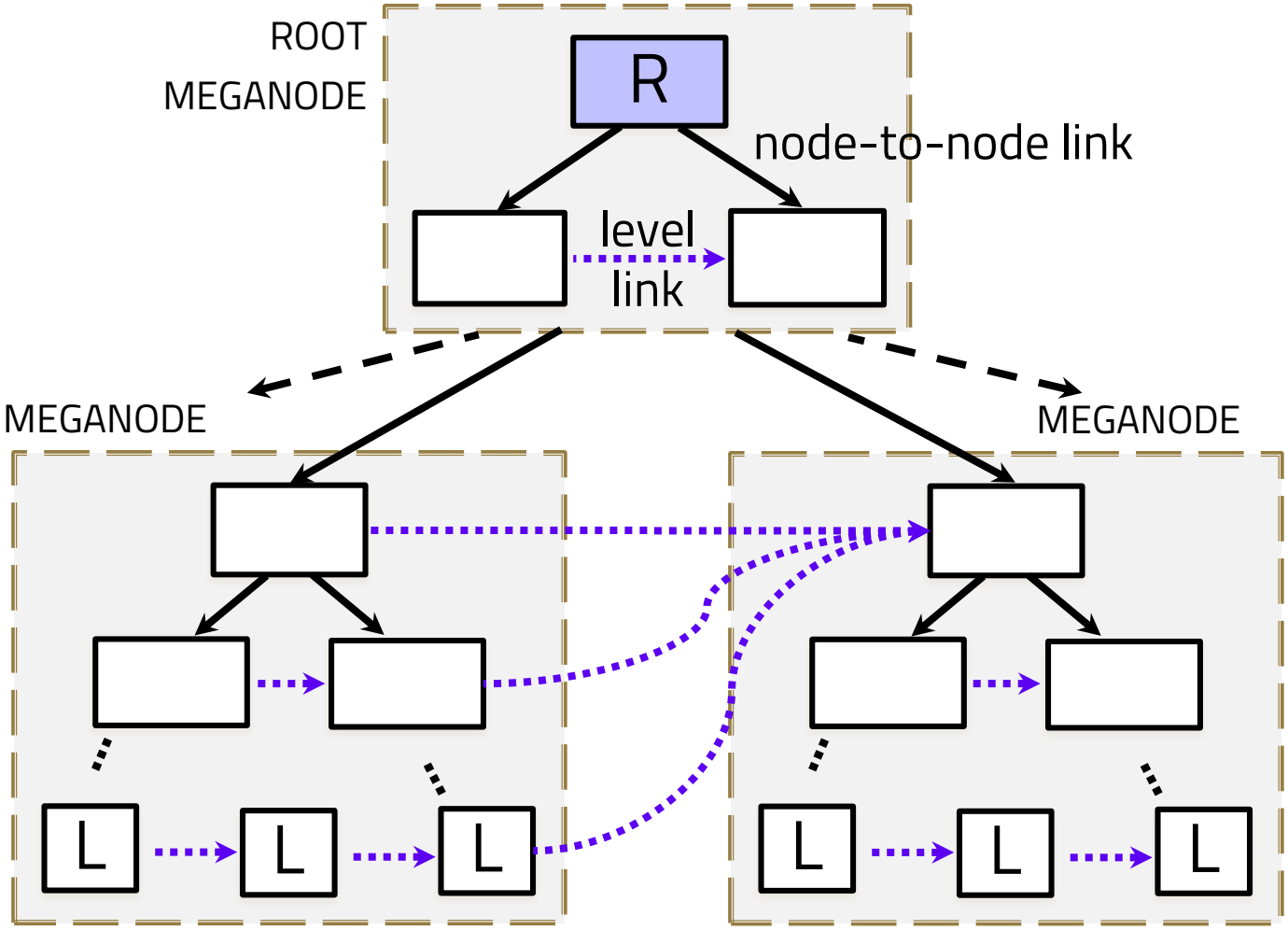
- Optimized for Ethernet
  - Data-computation locality heavily emphasized
  - BigTable: 128MB blocks, 3 RTs per operation
  - Large, opaque B-trees inside each B-tree node
- Great for server-side operations, bad for client-side operations
  - Bounded by server CPUs
  - Shouldn't ship large nodes via RDMA

How can RDMA help?

- **Selectively relax data-computation locality**

- B-link tree of (accessible) B-link trees
- Traverse tree by 1KB “lean” nodes
  - Client-side processing
- Traverse tree by 64MB “fat” nodes
  - Server-side processing

Cell B-Link Tree

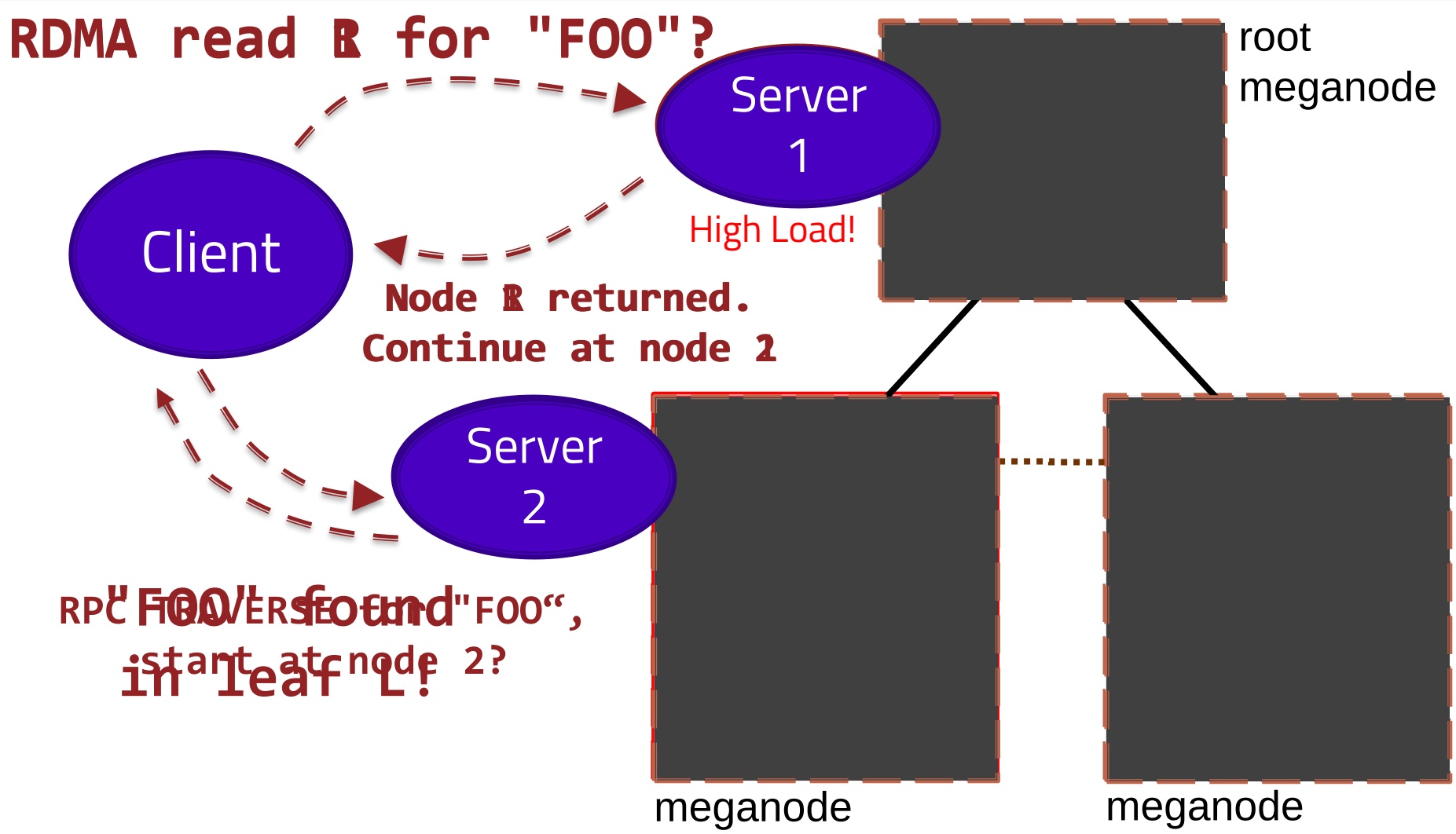


Design Choice 1: Client-Side and Server-Side Reads

Client-Side and Server-Side	Server-Side Only

• **Search**
  • Server-side: traverse fat nodes (meganodes) when server CPU is plentiful
  • Client-side: traverse slim nodes when server CPU is bottleneck
• **Scan**
	• **Insert**
	  • Node splits
	  • Meganode splits
	• **Delete**
	  • No rebalancing
	  • No distributed locks

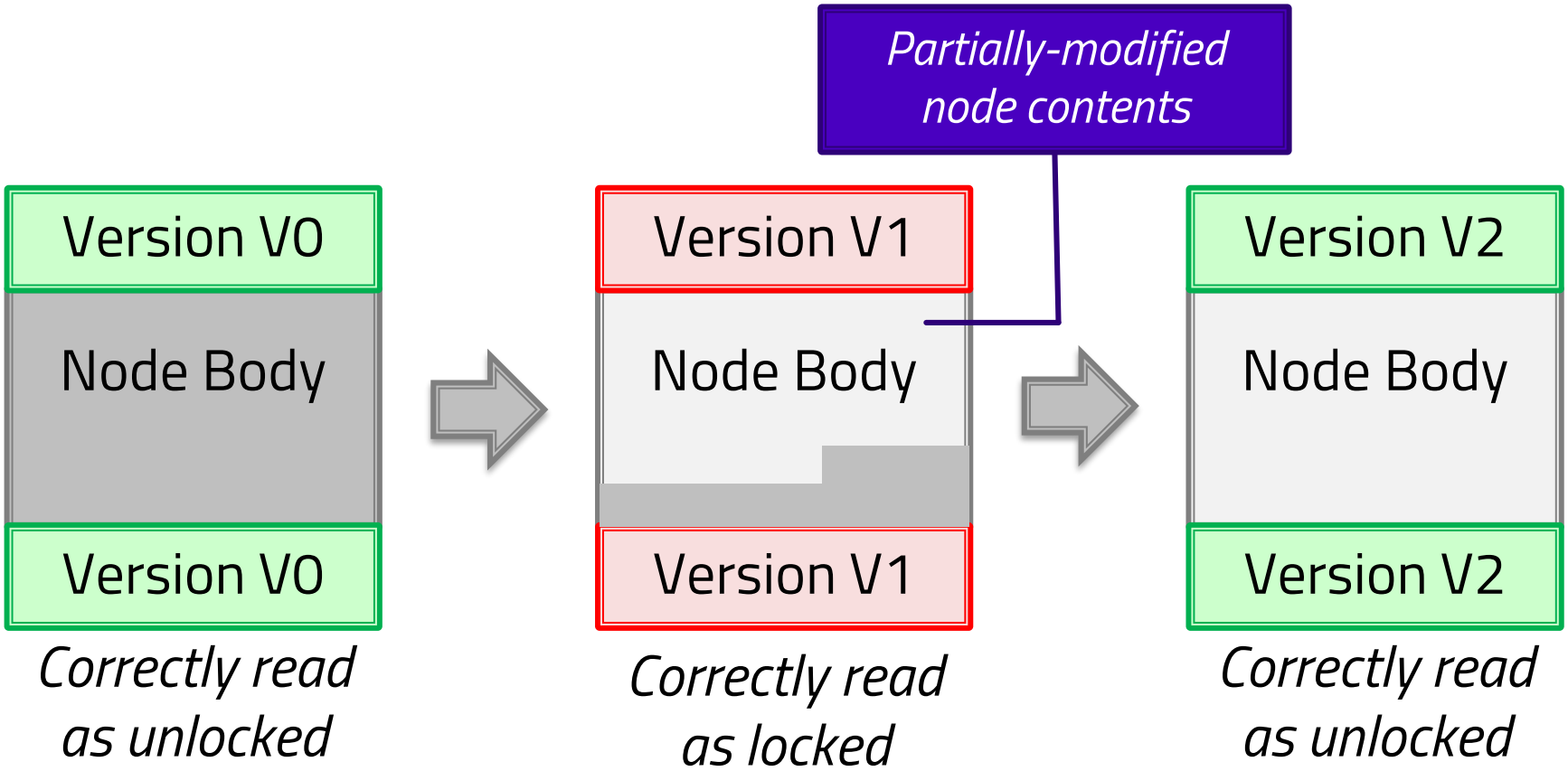
Cell's Sorted Store In Action



Using Client-Side and Server-Side Operations Together

- Writes: server-side only
- Reads: client-side or server-side
  - Server side: B-Link tree offers lock-free reads
  - Client side: lock-free reads… *if they're atomic*

Design Choice 2: Make Reads Atomic

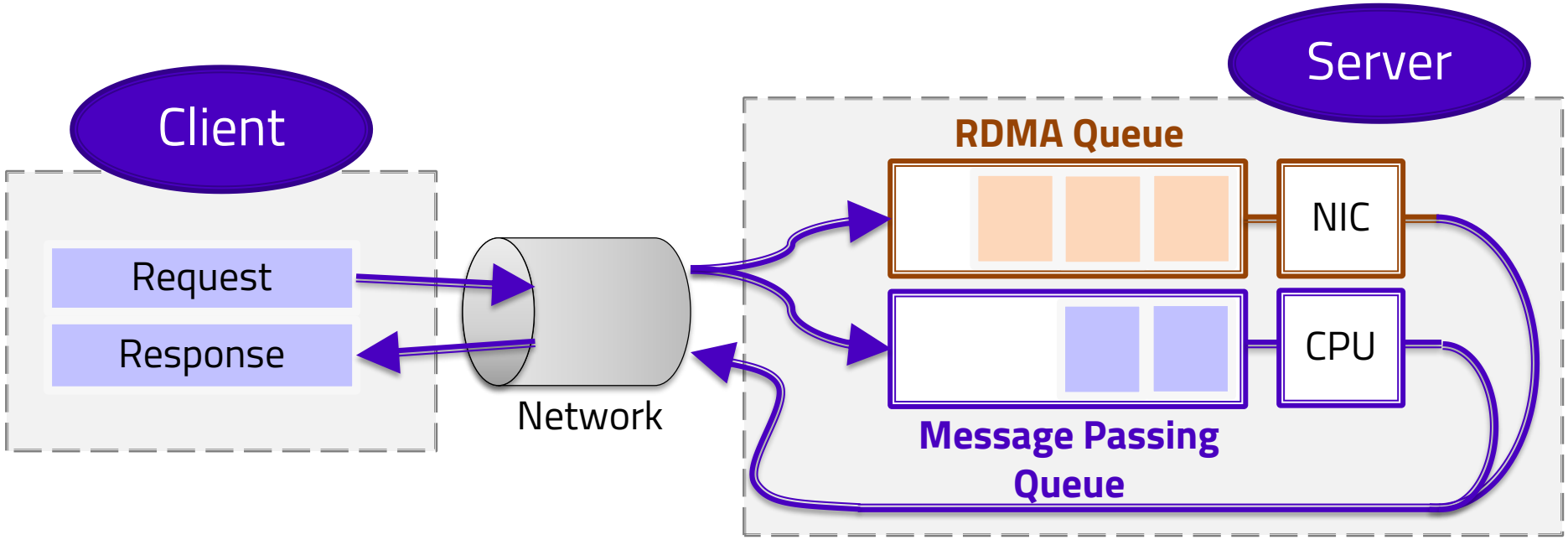


Choosing Between Client-Side and Server-Side Operations

- Naïve: pick lowest latency
- Suboptimal! Keep NIC and CPUs occupied.
- Potential pitfalls
  - Properly weighting operations
  - Extremely short transient conditions, outliers -> moving average
  - Stale measurements -> exploration

Design Choice 3: Client-Side Locality Selector

- Clients select client- or server-side search
- Queuing theory model
  - Select server "queue" currently least full



Outline

- Motivation: Selectively Relaxed Locality
- Cell: Balancing Server-Side & Client-Side Search
- Evaluation
- Related Work

Implementation

- C++, 16K LOC
- Infiniband with TCP-like connection mode
- Cell clients: Connection-sharing

Evaluation Questions

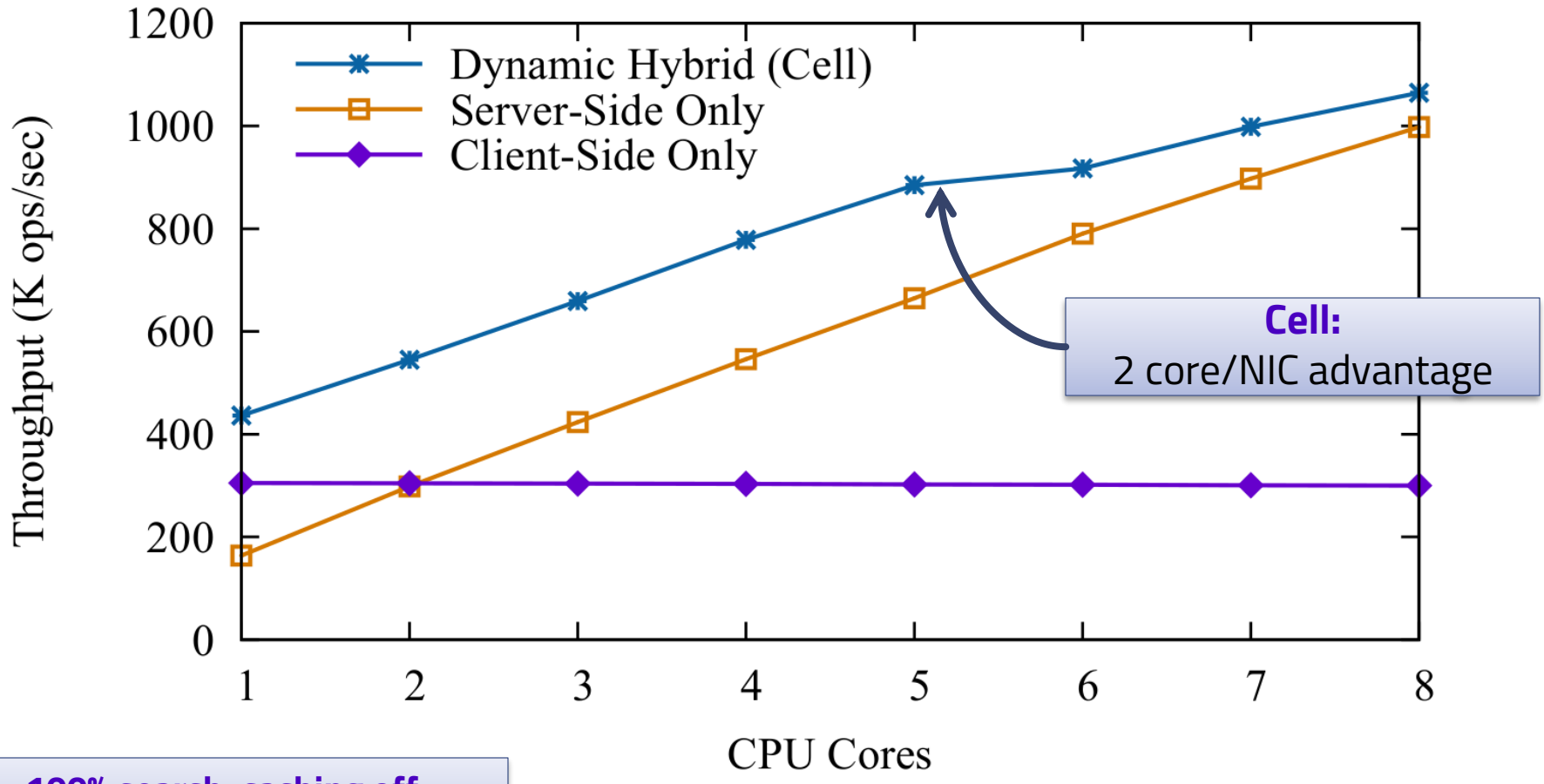
1. Can selectively relaxed locality save CPUs?
2. Do these techniques scale?
3. Can selectively relaxed locality handle load spikes?

Selectively Relaxed Locality is Fast

16 Server x 2-Core Throughput	Advantage over Server-Side	CPU Savings
5.31M OPS/SECOND	>170% VS. STRICT	2 CORES PER NIC

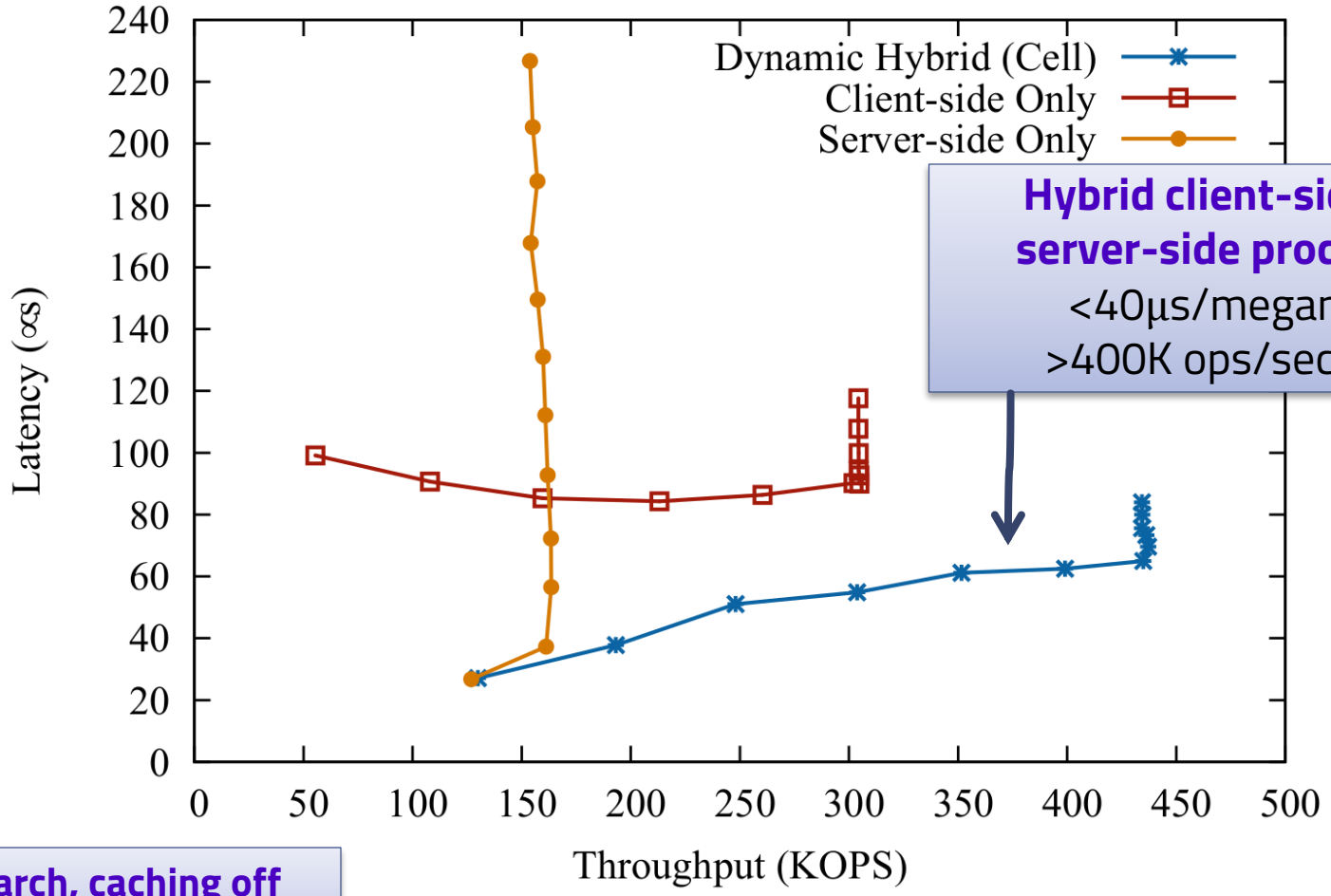
100% search, caching on
8-64 byte keys
64-256 byte values
2 cores per 16 servers

Selectively Relaxed Locality Scales to Many CPU Cores



100% search, caching off
1 meganode/serv, 1-8 cores

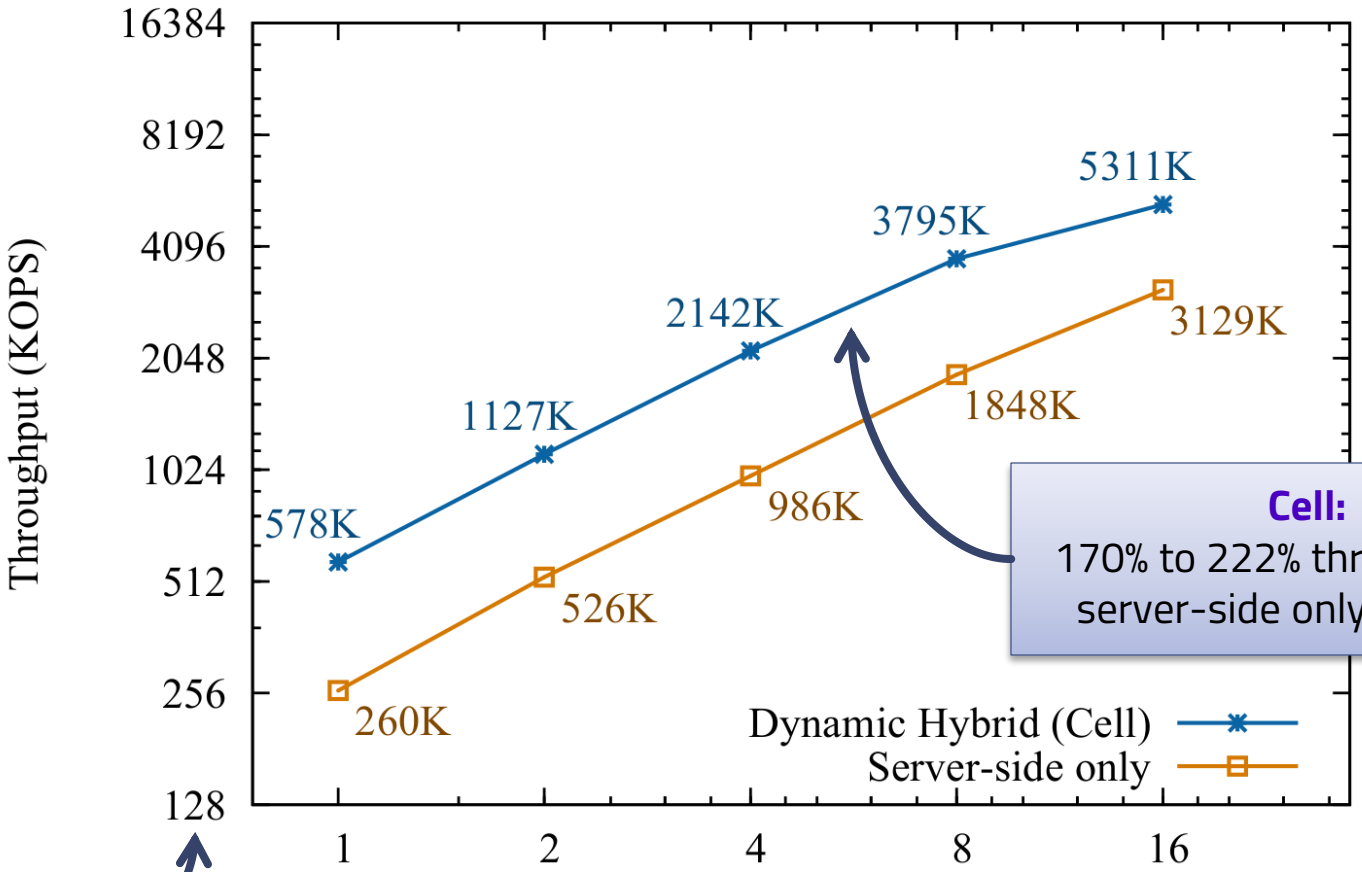
Selectively Relaxed Locality is Faster than Client-Side or Server-Side Alone



Hybrid client-side and server-side processing
<40µs/meganode
>400K ops/sec/core

100% search, caching off
1 meganode, 1 core

Selectively Relaxed Locality Scales to Many Servers

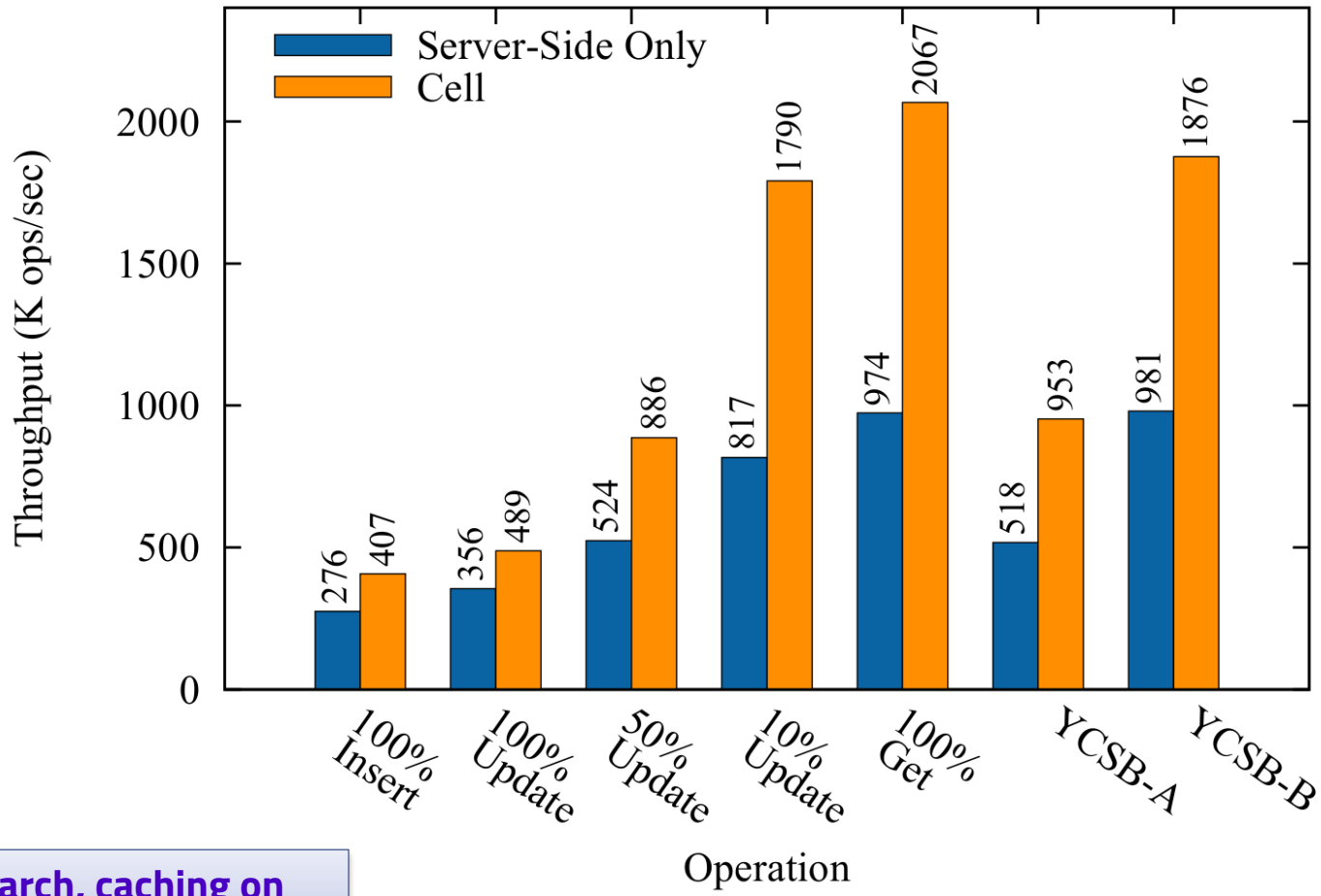


Cell:
170% to 222% throughput of server-side only searches

Log scale

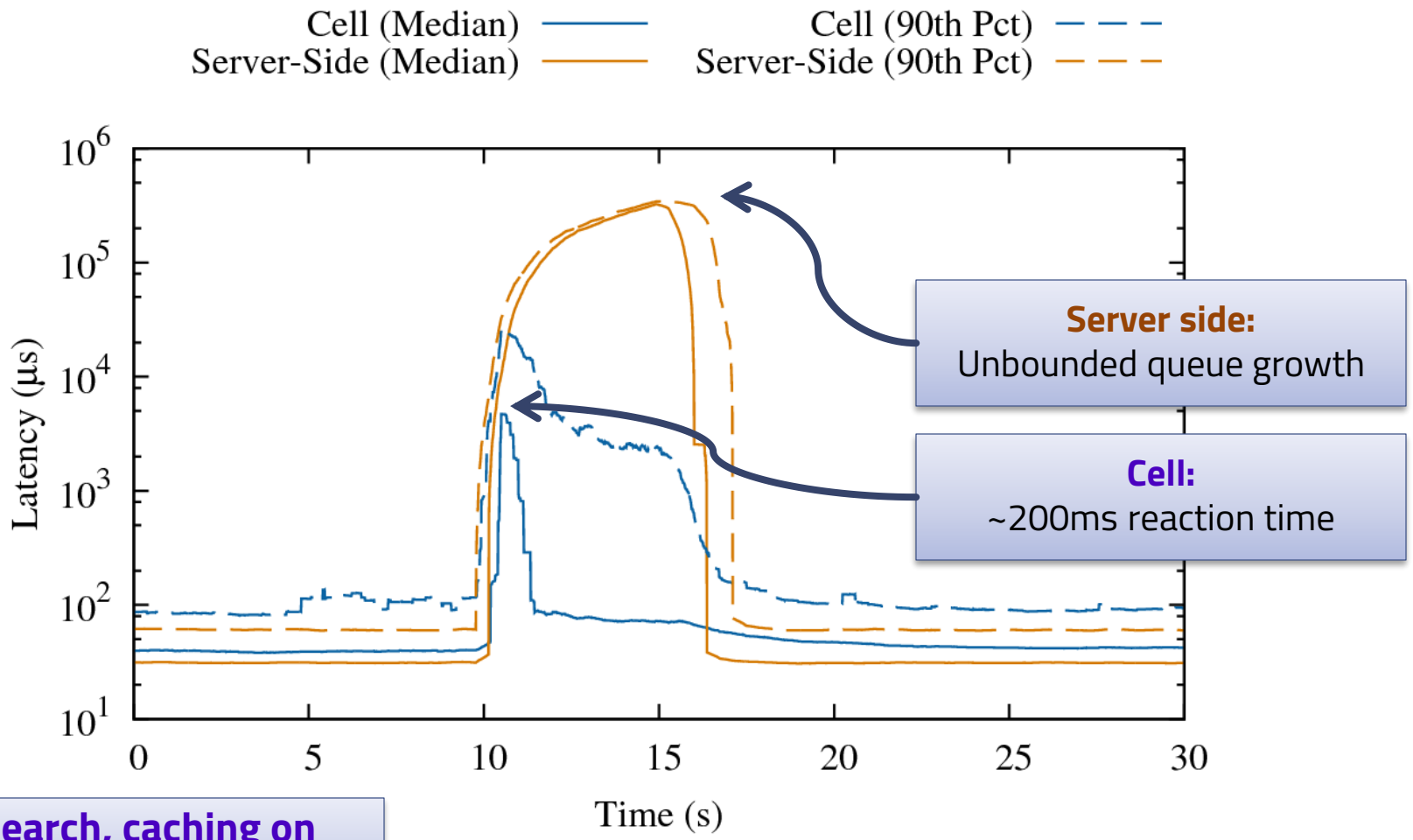
100% search, caching on
31 meganode/serv, 2 cores

Mixed Workloads



100% search, caching on
2 cores per 4 servers

Selectively-Relaxed Locality Handles Load Spikes



100% search, caching on
31 meganodes, 2 cores

Outline

- Motivation: Selectively Relaxed Locality
- Cell: Balancing Server-Side & Client-Side Search
- Evaluation
- Related Work

Related Work

- RDMA for faster message passing:
  - MPI, Memcached, Hbase, Hadoop, PVFS, NFS
  - Recent: HERD, FaRM
- In-memory K-V and sorted stores
  - FaRM: Similar to DSM, includes K-V store app
  - H-Store, VoltDB, Masstree, Silo
- Distributed B-trees
  - Sagiv's B-link tree: Johnson & Colbrook, Boxwood

Lessons & Conclusion

- Tomorrow's datacenters will include RDMA-capable, ultra-low latency networks

- New system architectures:
  1. Selectively-relaxed locality for load balancing and CPU efficiency
  2. Self-verifying data structures make this practical
  3. Locality-relaxation techniques work at scale

Thank you! Any questions?

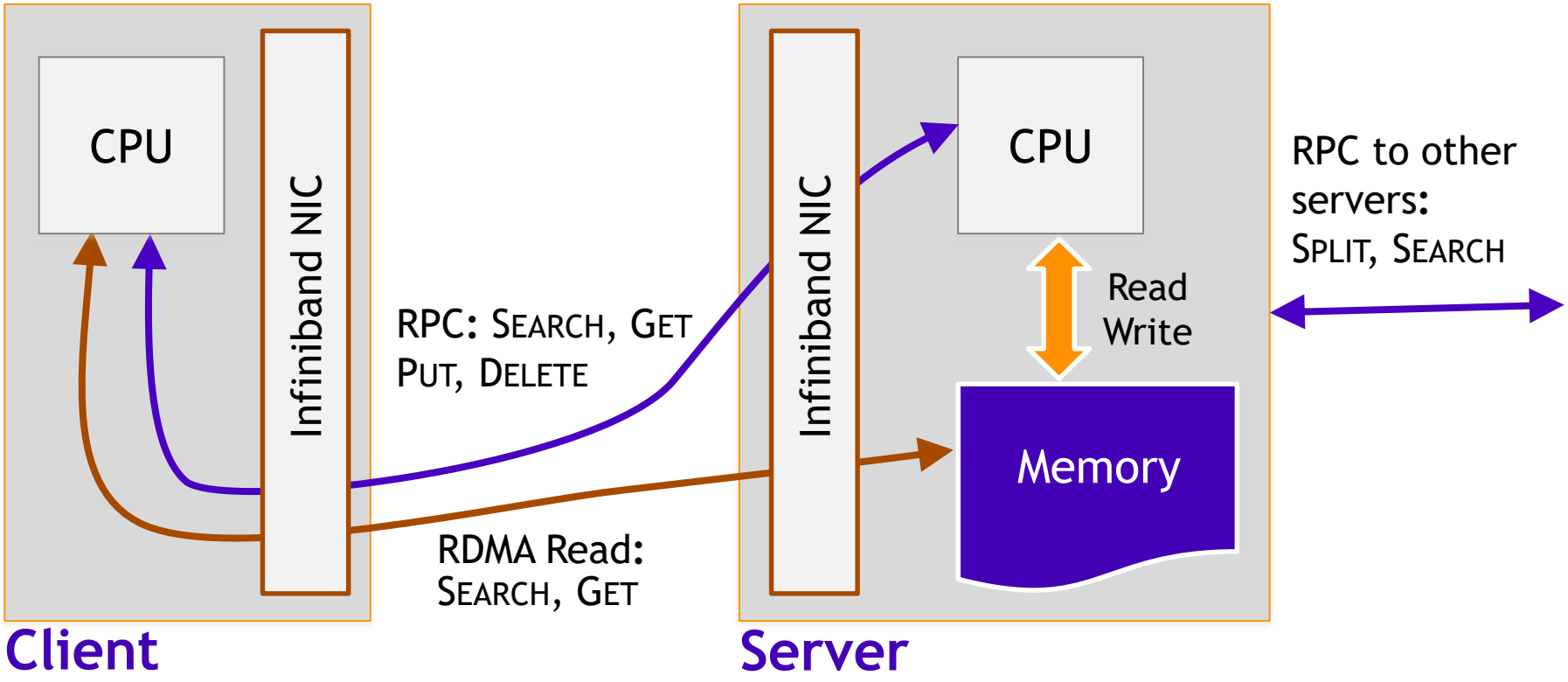
References

- MPI: Liu 2003, Liu 2004, Shipman 2006,
- Memcached: Stuedi 2012, Nishtala 2013, Jose 2011, Jose 2012
- Hbase: Huang 2012
- Hadoop: Lu 2013
- PVFS: Wu 2003
- NFS: Gibson 2008
- HERD: Kalia 2014
- FaRM: Dragojevic 2014, Dragojevic 2015
- H-Store: Kallman 2008
- VoltDB: Unknown, 2010
- Masstree: Mao 2012
- Silo: Tu 2013
- Sagiv's B-link tree: Lehman 1981, Sagiv 1986
- Johnson & Colbrook: Johnson 1992
- Boxwood: MacCormick 2004

Excised Slides

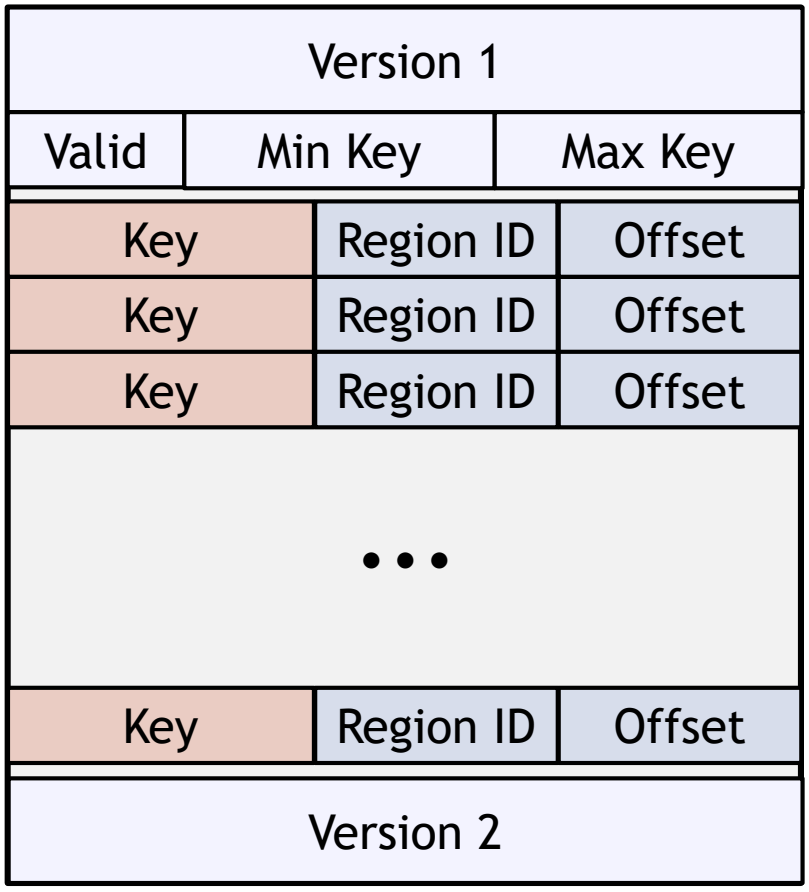
Potentially-useful extra slides

Cell's System Architecture

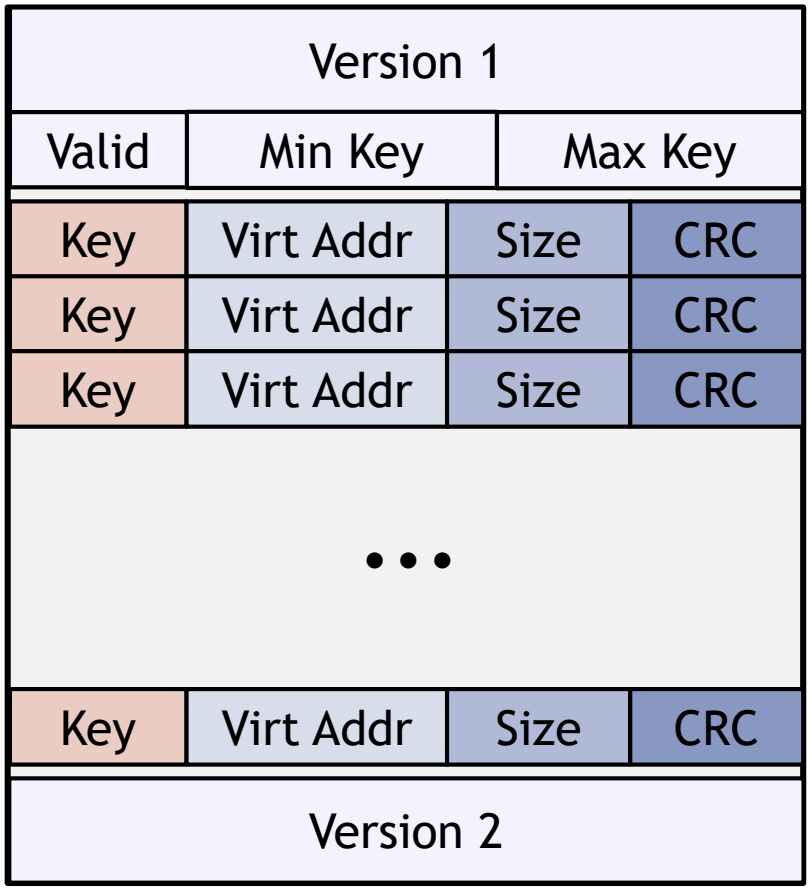


Small Node Structure

Internal Node



Leaf Node



JSQ Details

$$\frac{q_s}{T_s} < m \cdot \frac{q_r}{T_r}$$

- q_s = Server-side search queue length
- T_s = Server-side service capacity
- q_r = RDMA search queue length
- T_r = RDMA service capacity
- m = RDMA traversals per meganode