NeVer Mind networking: Using shared non-volatile memory in scale-out software

Stanko Novakovic*, Paolo Faraboschi, Kimberly Keeton, Rob Schreiber, Edouard Bugnion*, Babak Falsafi *

EPFL, Switzerland * HP Labs, USA

Multiple non-volatile memory technologies

Magnetic disk

- Block-based storage, access via programmed IO or DMA (e.g. ATA)

Flash (PCIe-attached – via NVMe)

- Block-based storage, access via queue-pairs (i.e. SQ, CQ) and DMA

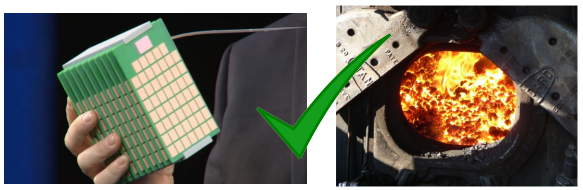
Non-volatile Random Access Memory (NVRAM)

- Persistent with similar performance characteristics to DRAM
- Byte-addressable, direct access via LD/ST
- Examples: Resistive RAM (RRAM), Phase-change memory (PCM)

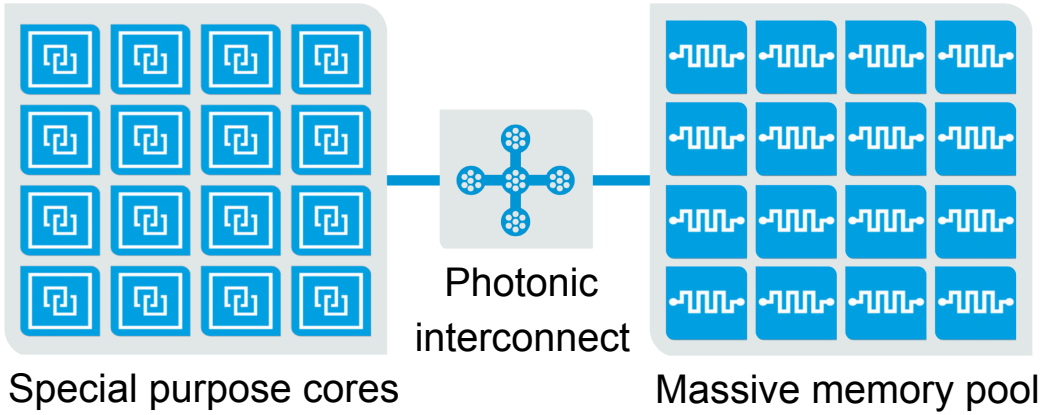
Rack-scale systems with shared non-volatile memory

	Shared disk architecture	Shared NVMe namespaces	TheMachine/ Firebox
Cache coher.	No	No	No
Interface	PIO, DMA	QP-based	LD/ST
Technology	Magnetic disk	Flash	NVRAM

1. **Direct shared access**
2. **Latency: small factor of DRAM**
3. **Bandwidth: DDR bandwidth**



The Machine from HP Labs



Petabytes of byte-addressable NVRAM based on memristor technology

- Shared across thousands of cores via photonic interconnect

High-performance, high-capacity rack-scale computing

Outline

Introduction to shared NVRAM

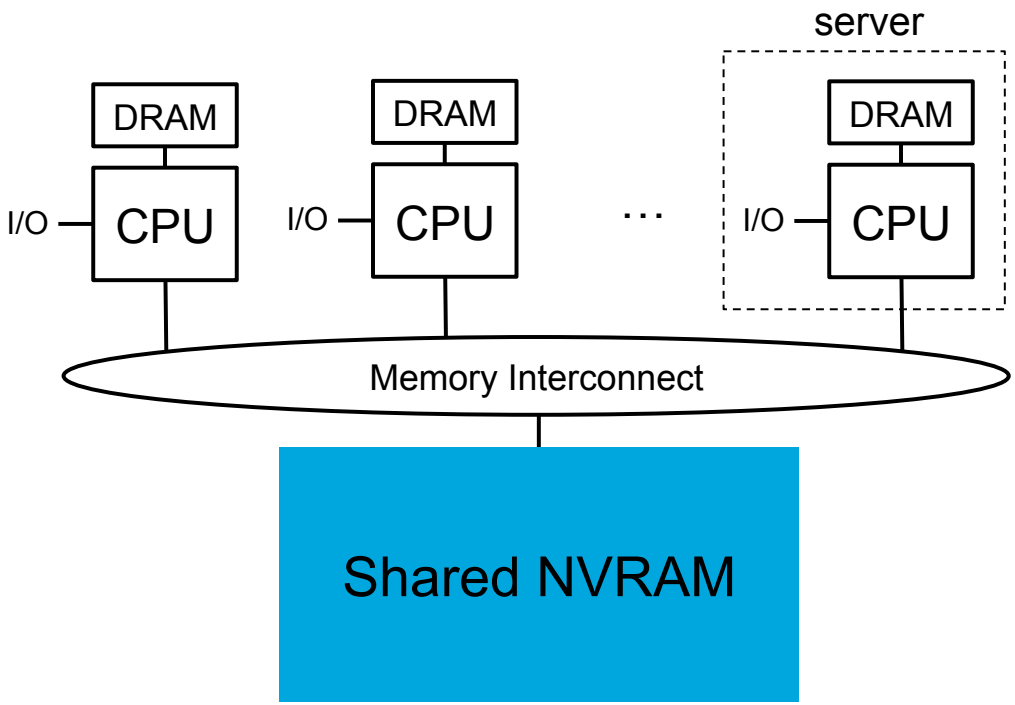
→ Shared NVRAM model

Hash Table

Graph Processing System

Conclusion

Shared NVRAM model



Servers

- Do not share DRAM
- Share NVRAM
  * but not CC *

Shared NVRAM

- Used as a heap
- Latency: 4-5x of DRAM
- Bandwidth: DDR rate

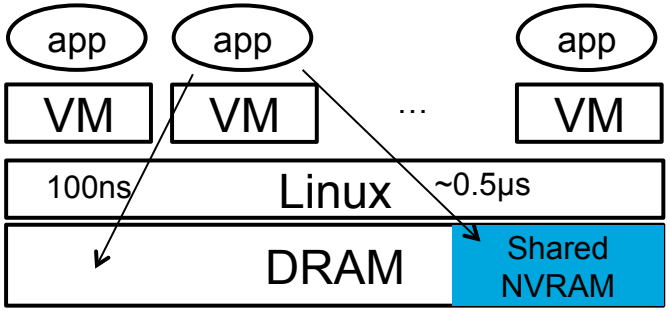
Can datacenter software benefit from such architecture?

Experimental shared NVRAM platform

Kernel Virtual Machines (KVM)

Fraction of each guest's address space is shared

User code instrumented to delay NVRAM accesses



Ttrans = lat + data_size/bandwidth

~0.5µs

DDR3/QPI
(unchanged)

Using shared NVRAM in datacenter software systems

Data serving

- Distributed hash tables, graph stores

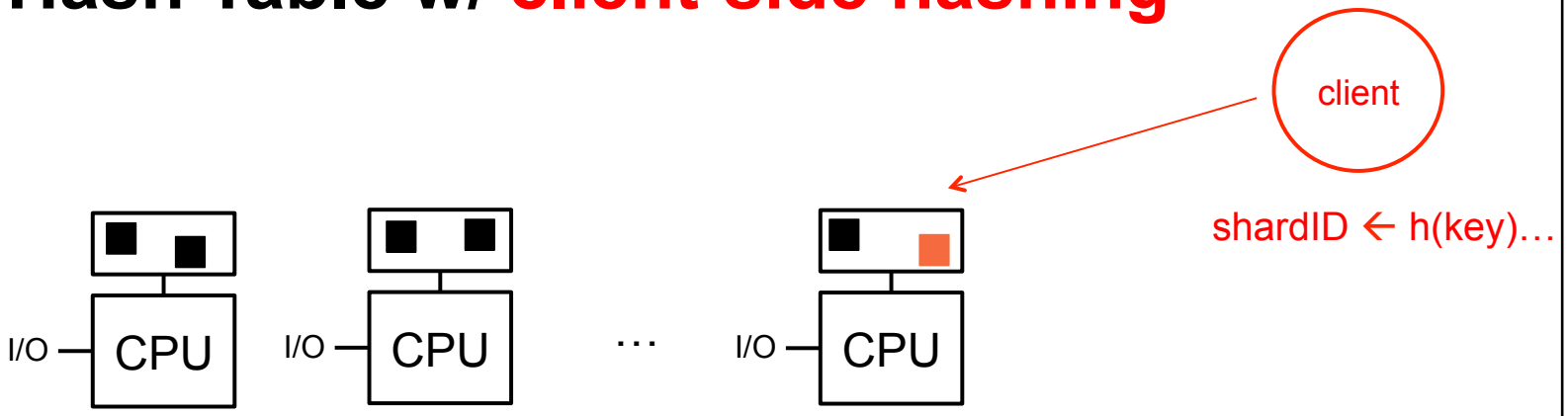
→ **Share data items via NVRAM**

Data processing (a.k.a. analytics)

- MapReduce, distributed graph processing, etc.

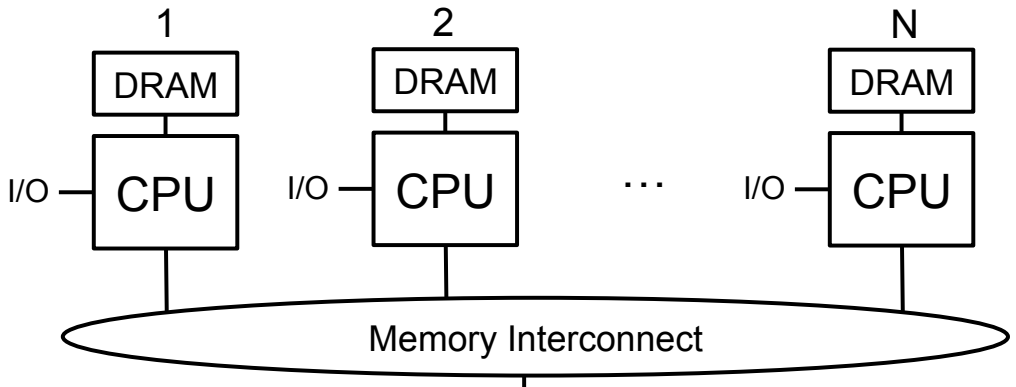
→ **Use shared NVRAM as communication medium**

Today: Hash Table w/ client-side hashing

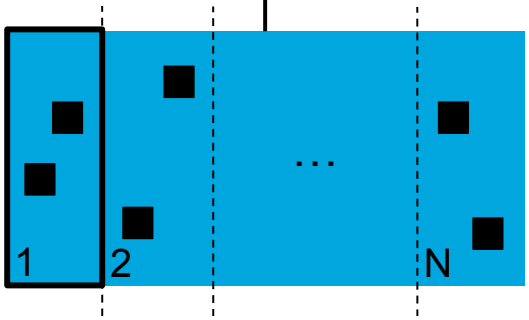


This work: CREW Hash Table for shared NVRAM

Each server owns write permission to one block and read permission to whole NVRAM



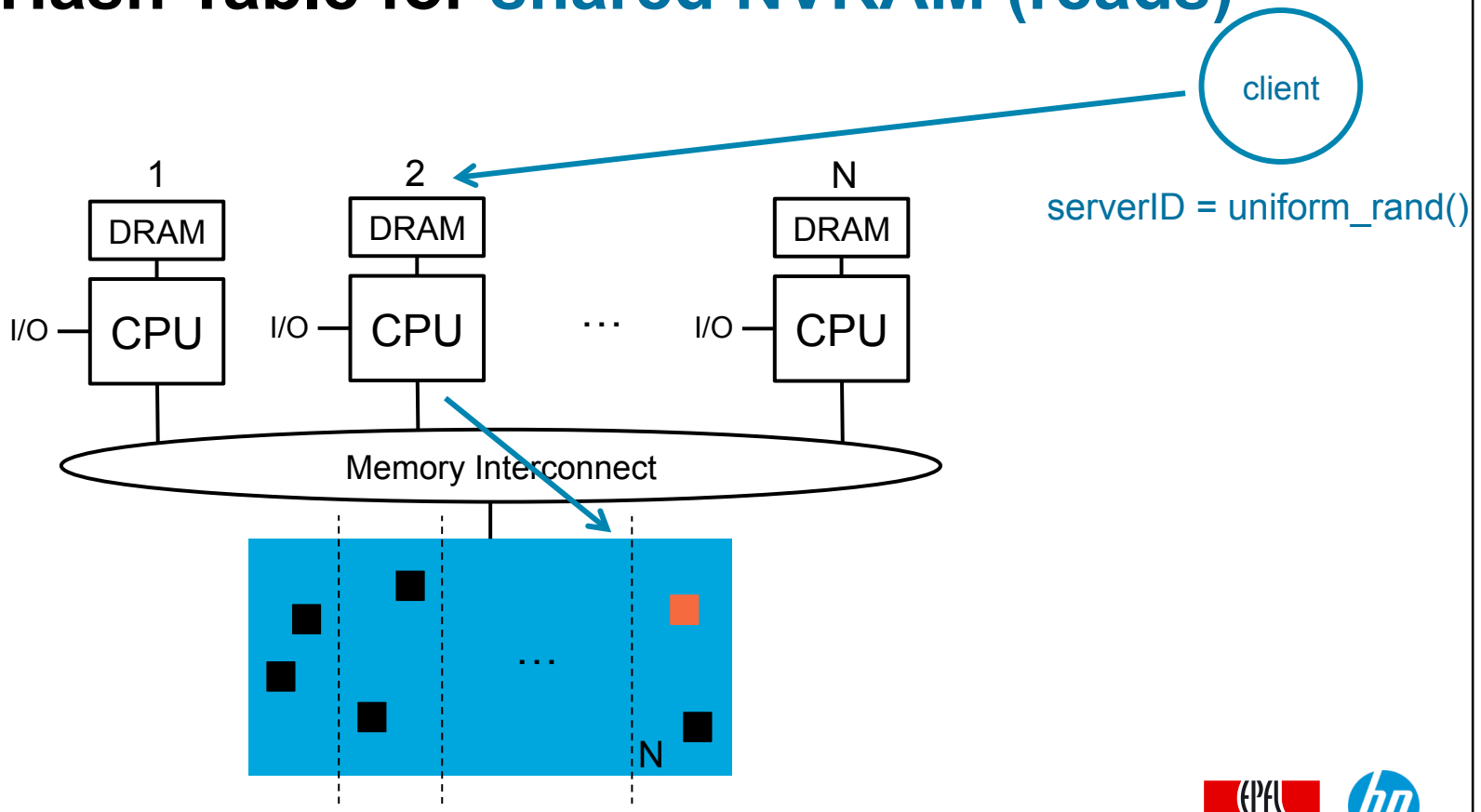
Permisssions:
Read-Write = {1}
Read-Only = {1,2,..N}



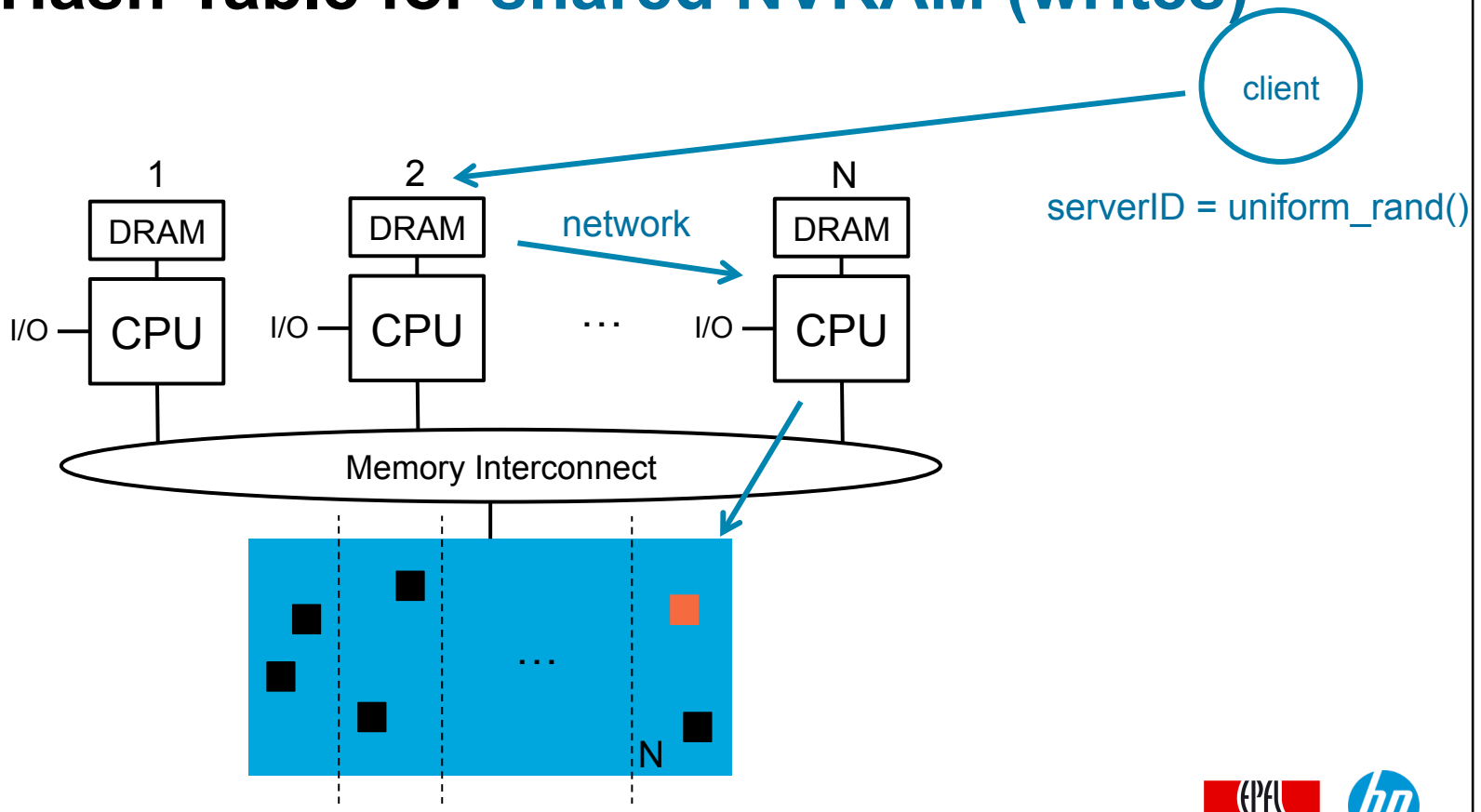
Data stored in NVRAM

 CREW – Concurrent Read Exclusive Write

CREW Hash Table for shared NVRAM (reads)



CREW Hash Table for shared NVRAM (writes)



Hash table for shared NVRAM prototype

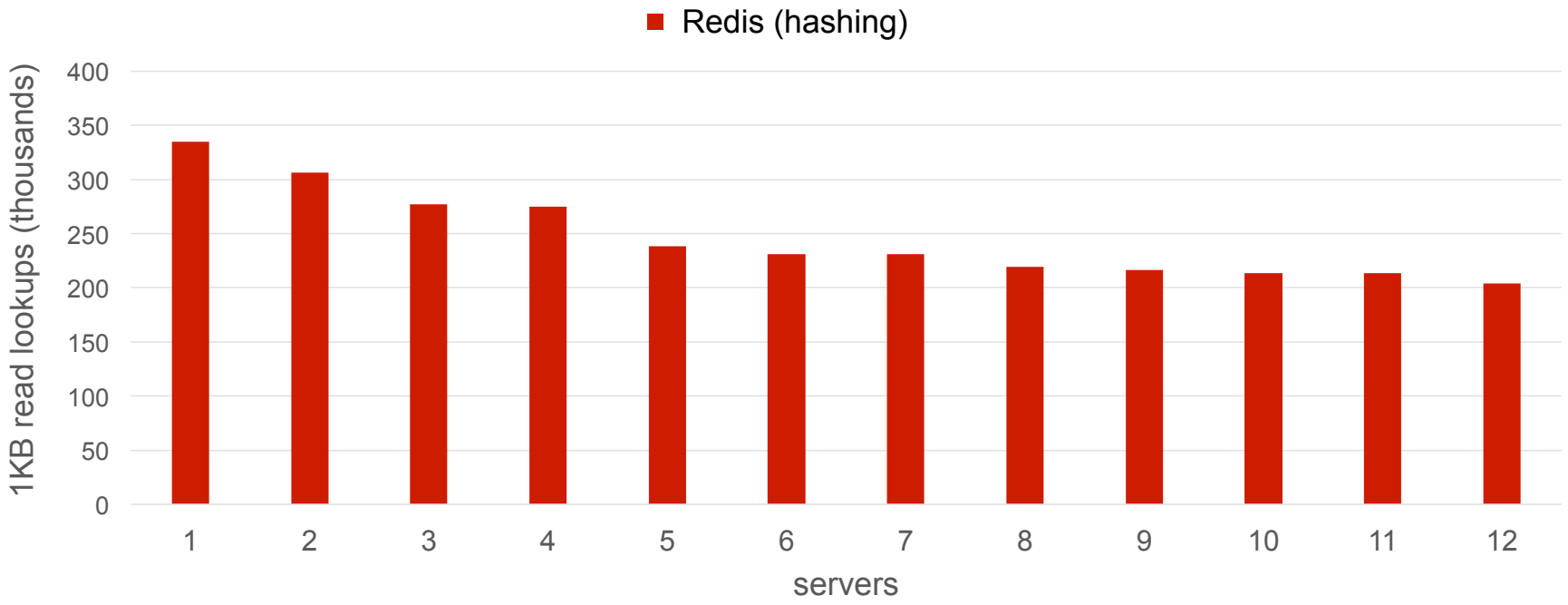
Based on Redis KVS

- Each server runs separate instance of Redis
- Shared read-only access to data items

YCSB clients run on separate servers

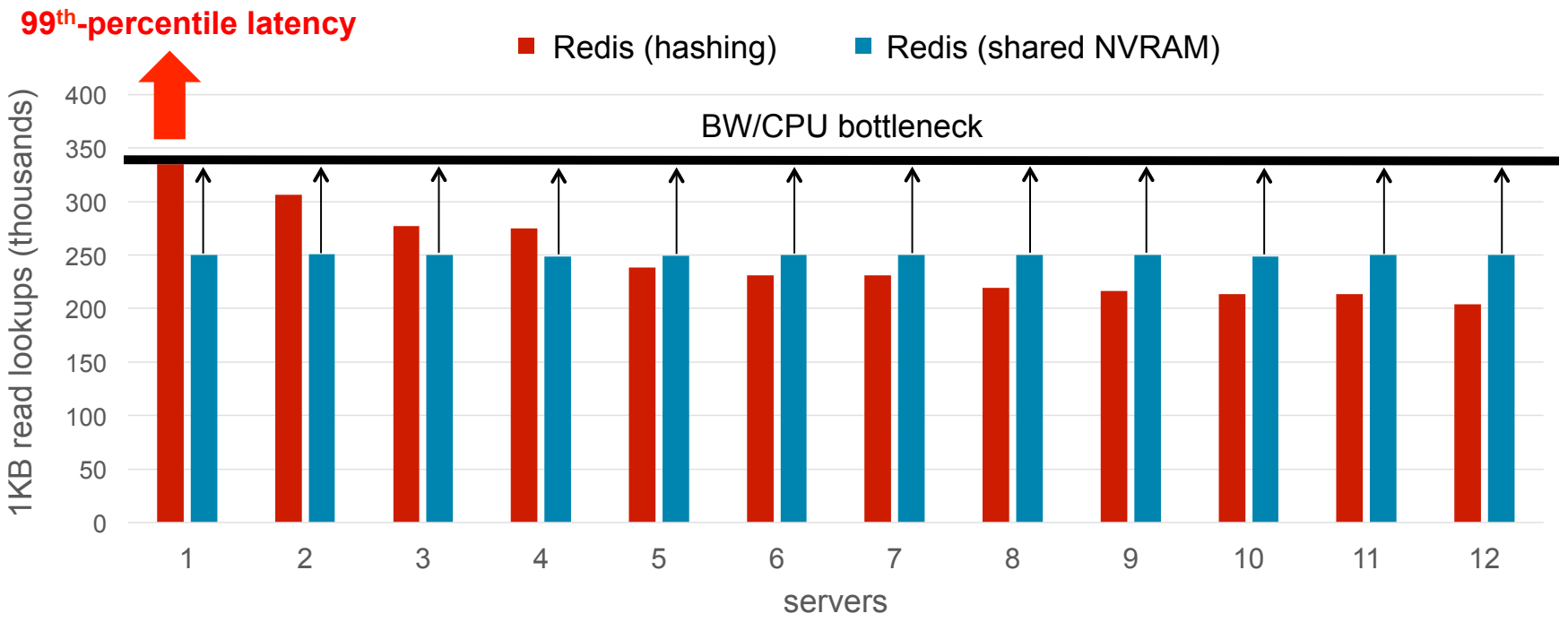
- Configured to compute hash or pick server in round-robin fashion

Per-server network utilization (0.99 Zipf workload)



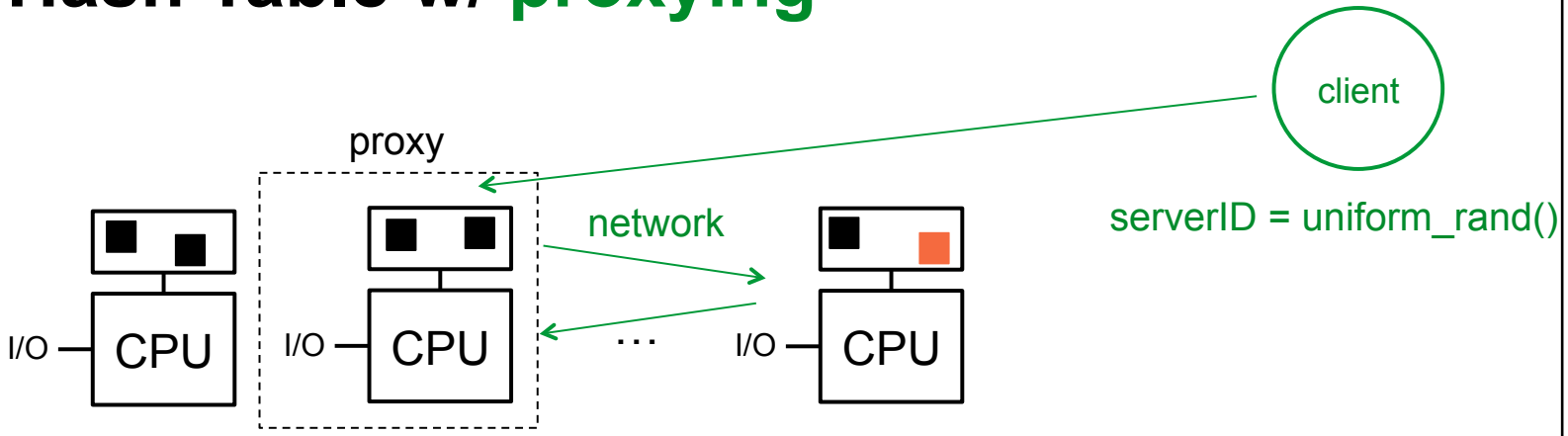
Consistent hashing on client side can cause load imbalance

What does uniform utilization buy us?

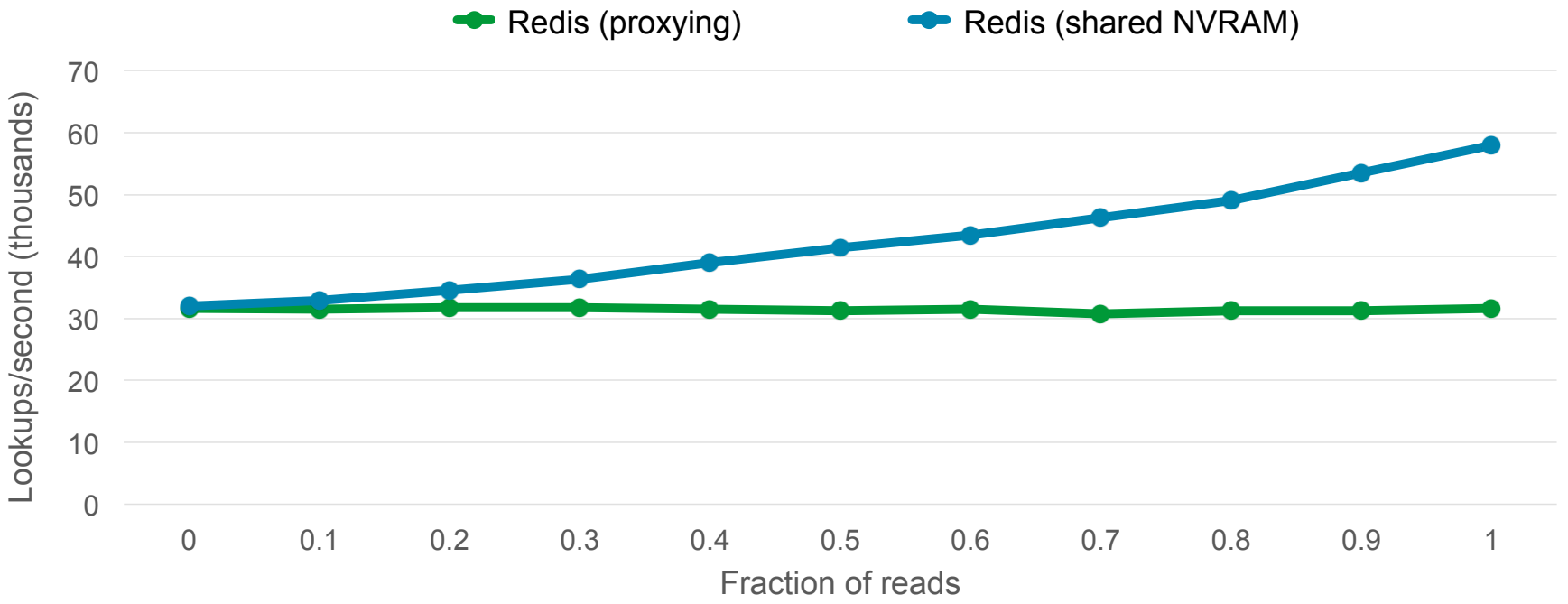


Higher throughput due to shared access w/o violating SLA

Today: Hash Table w/ proxying



HT w/ proxying vs. HT w/ shared access



~2x improvement in performance for read-only workload

Data serving recap

Looked at two hash table (KVS) designs

- with client-side hashing and proxying

KVS that uses shared NVRAM allows for shared read-only access

- Better load balancing over hashing
- Lower end-to-end latency over proxying

Outline

Introduction to shared NVRAM

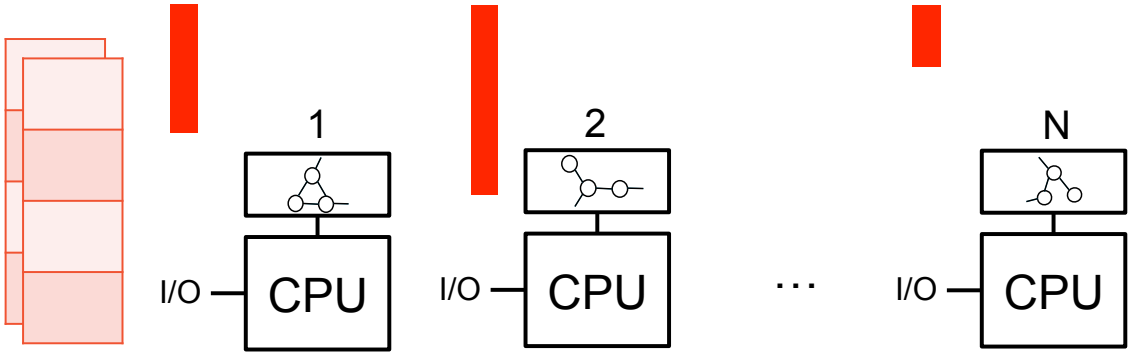
Shared NVRAM model

Hash Table

→ Graph Processing System

Conclusion

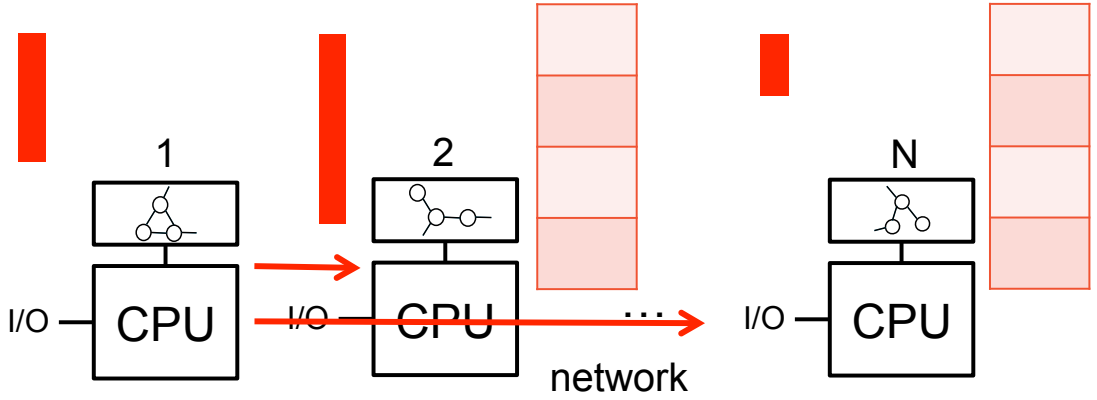
Bulk Synchronous Parallel (BSP) on scale-out



Compute

Buffered vertex updates

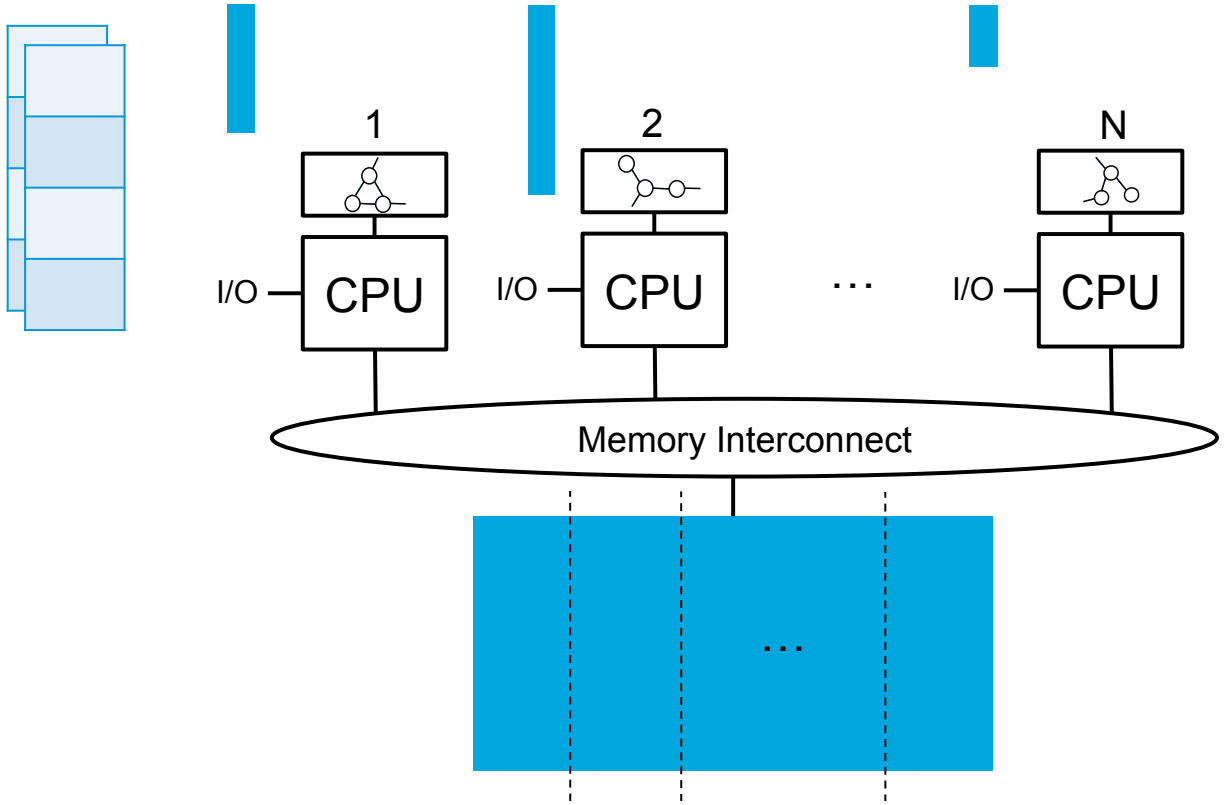
BSP graph processing on scale-out



Compute

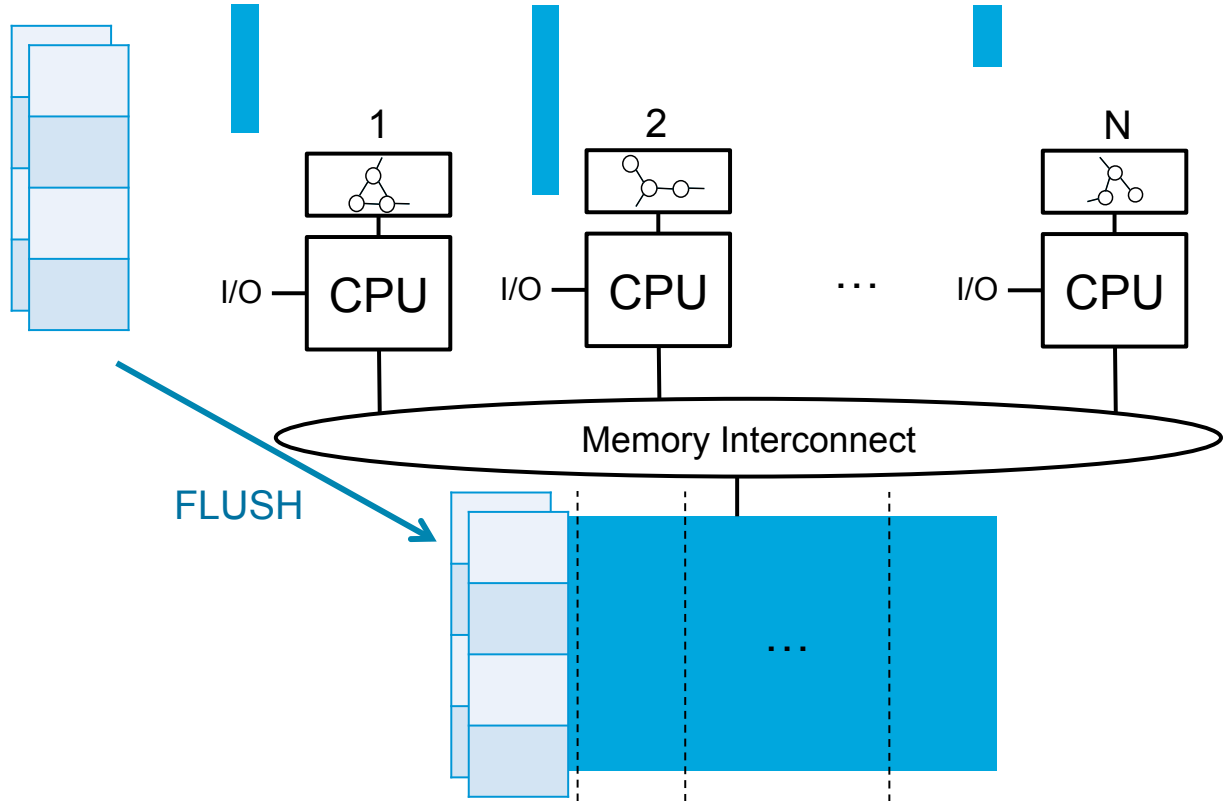
Communicate

BSP graph processing on shared NVRAM



Compute

BSP graph processing on shared NVRAM

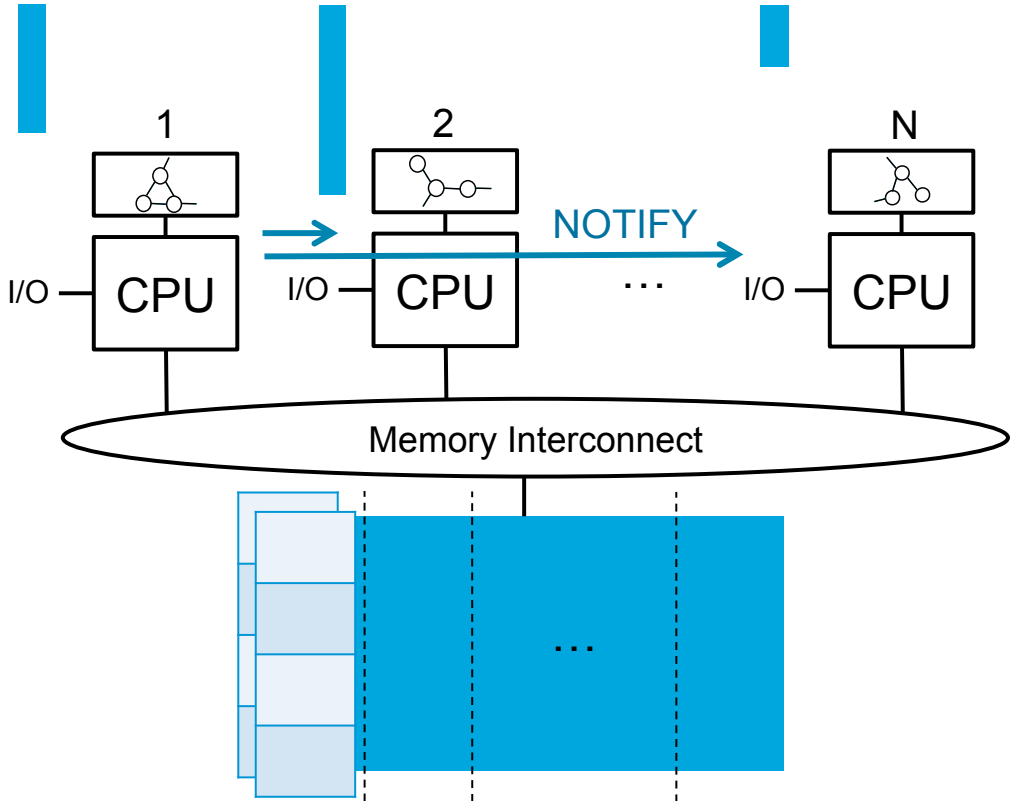


Compute

Communicate

DRAM → NVRAM

BSP graph processing on shared NVRAM

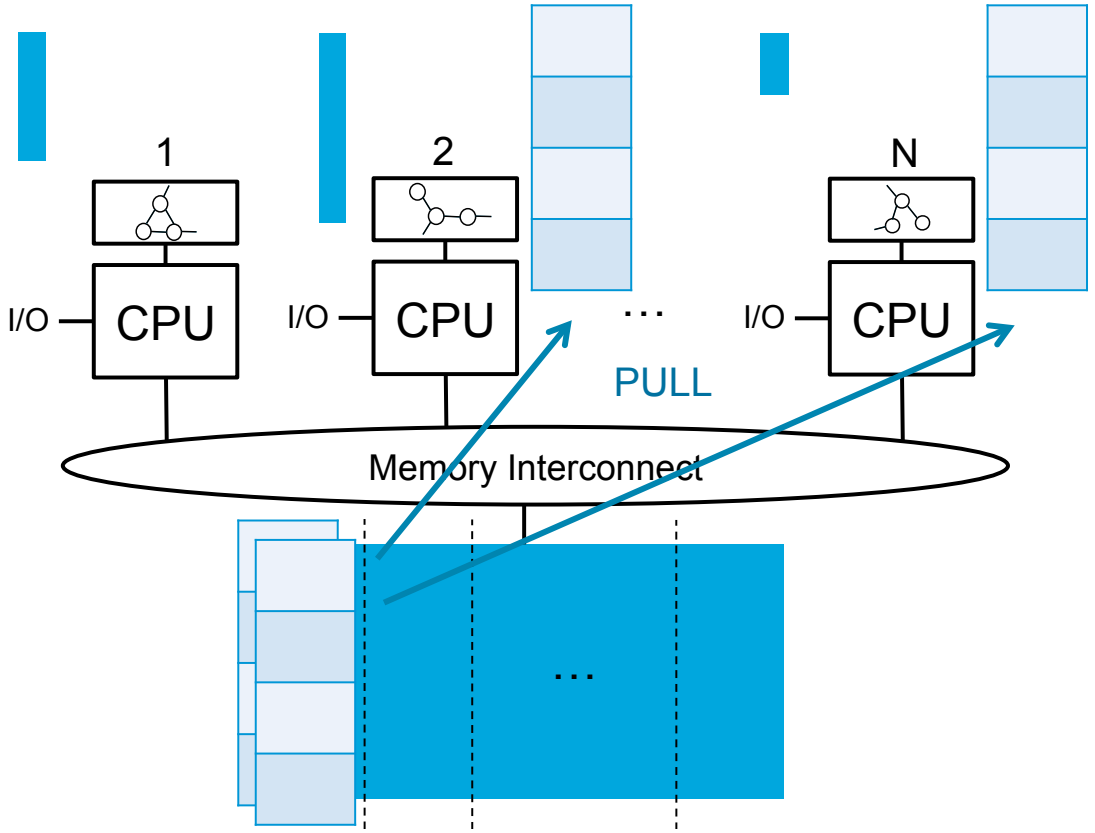


Compute

Communicate

- DRAM → NVRAM
- NOTIFY

BSP graph processing on shared NVRAM



Compute

Communicate

- DRAM → NVRAM
- NOTIFY
- NVRAM → DRAM

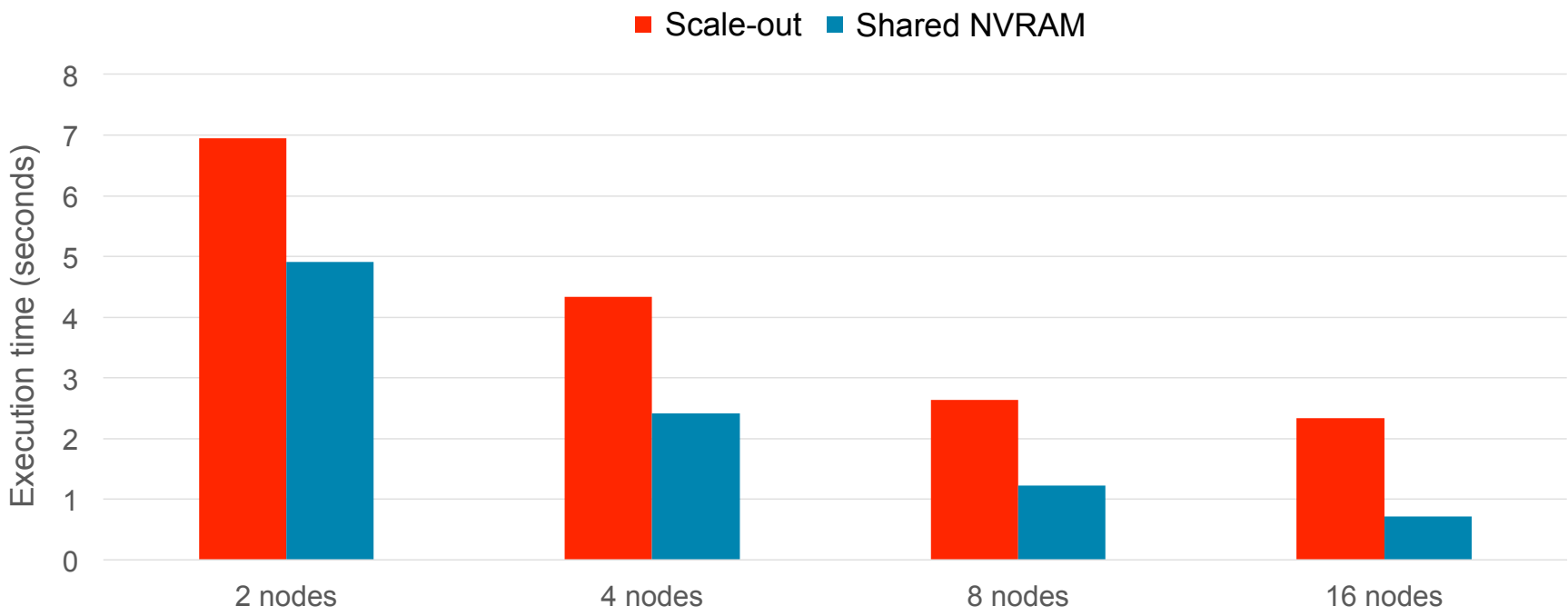
BSP graph processing for shared NVRAM prototype

Shared NVRAM BSP framework implemented in C++ from scratch

Algorithms implemented as compute kernels

Compare BSP over TCP/IP and BSP over NVRAM

PageRank on Twitter graph



Accelerating shuffle phase helps improve overall execution time

Related work

Concurrent Read Exclusive Write (FARM NSDI'14, MICA NSDI'14)

Graph update aggregation (Pregel SIGMOD'10)

Shared disks and shared NVMe namespaces

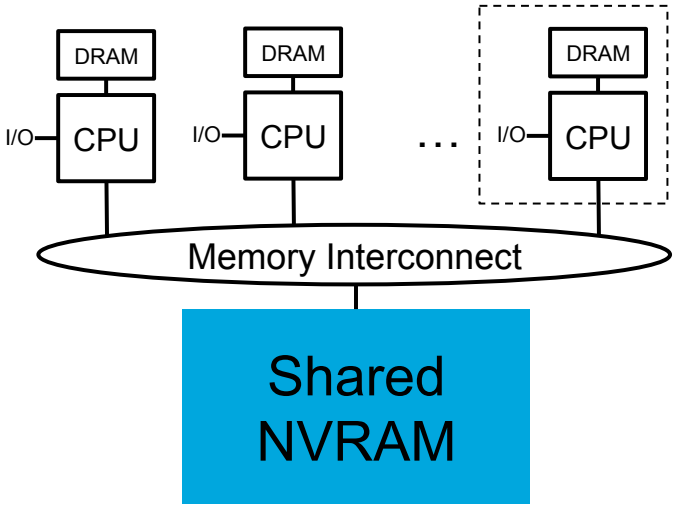
Conclusion

Rack-scale systems w/ shared NVRAM

- Direct access, byte-addressable

This talk: how to make use of this arch.

- In common DC apps
- Studied KVS and graph processing



Thank you