TECHNISCHE UNIVERSITÄT DARMSTADT

Rethinking Distributed Databases for Modern Networks

Carsten Binnig

DFG

HUAWEI

Mellanox TECHNOLOGIES

ORACLE

In the past …

Network Communication was evil: Must be **avoided at all cost**

	DDR3 -1600	1Gb Eth.	Net/RAM
Latency (µs)	0.1	100	1000
Throughput (GB/s)	51.2 (4 channels)	0.125	~400

Distributed DBMS Mantra: Data-Locality first!

- **Complex partitioning schemes** to leverage data-locality
- Complex **communication avoiding schemes** (e.g. semi-join reducers, relaxed consistency protocols)

BUT modern networks …

make it possible to achieve

network bandwidth

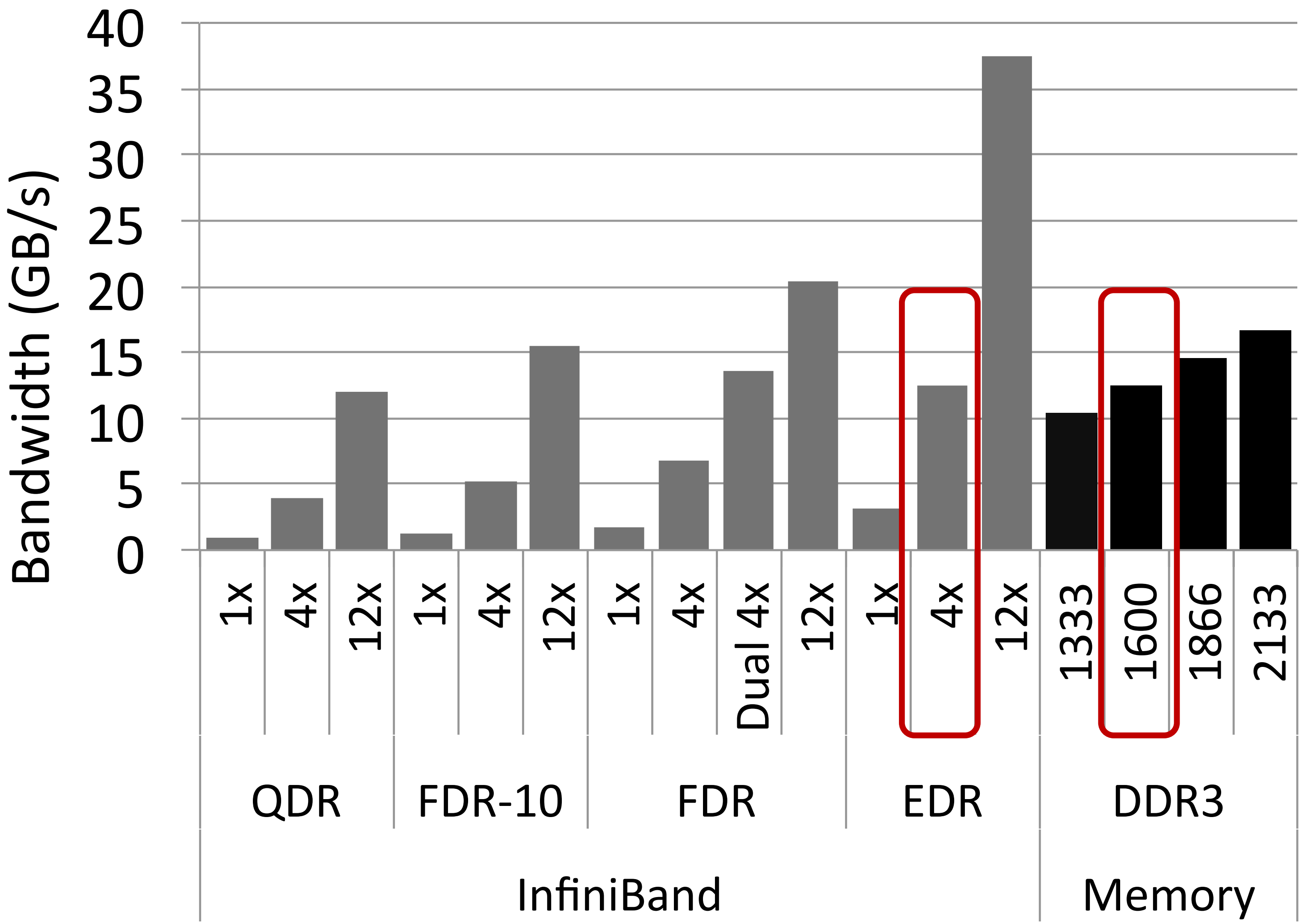
similar to the main

memory bandwidth

and it does no longer

ruin your budget

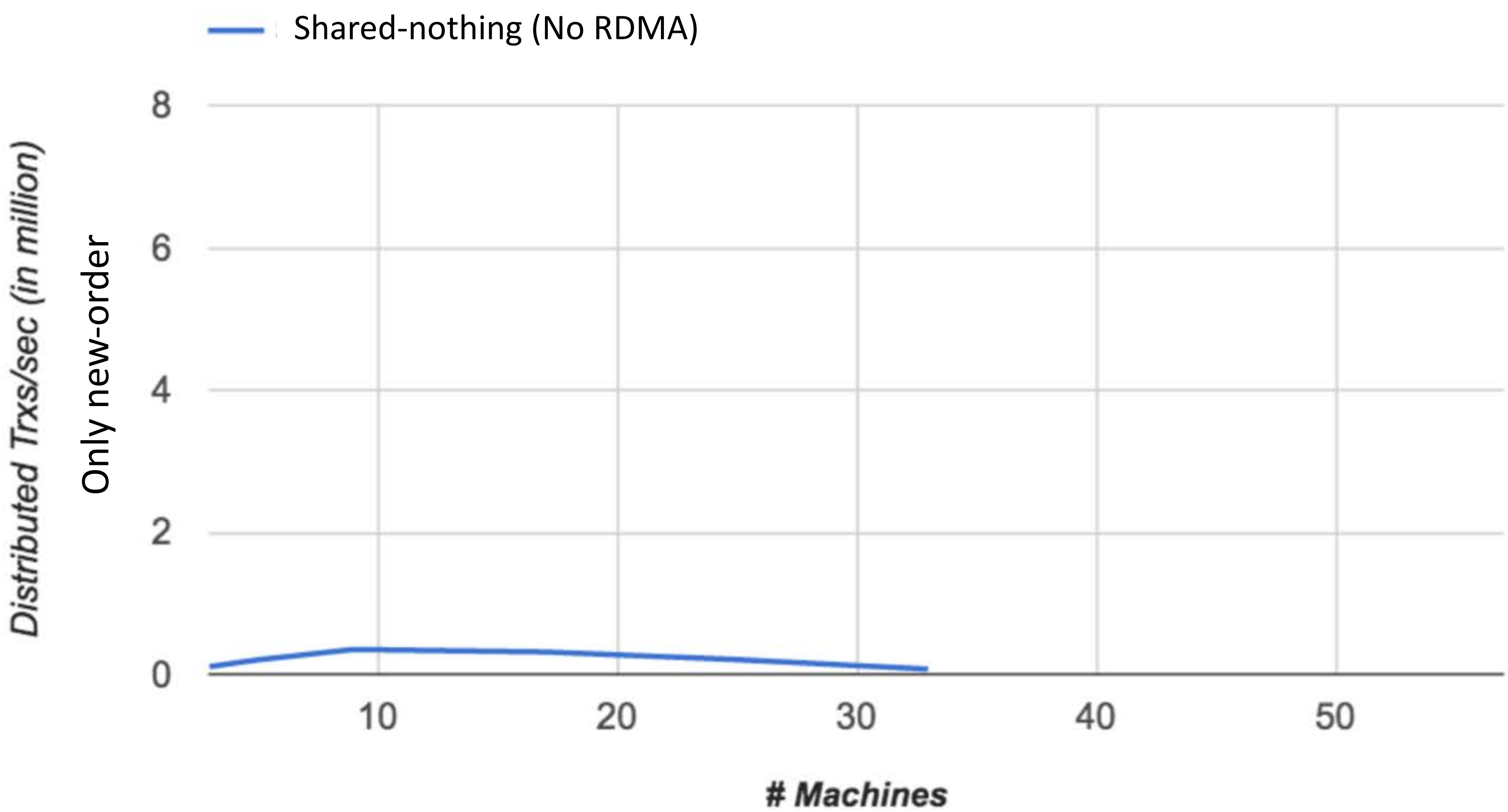
Distributed Systems are getting more balanced!



Distributed DBMSs:
Just **Upgrade Network?**

OLTP: Scale-out Experiment on TPC-C

Binnig et al.: The End of A Myth: Distributed Transactions Can Scale. VLDB 2017

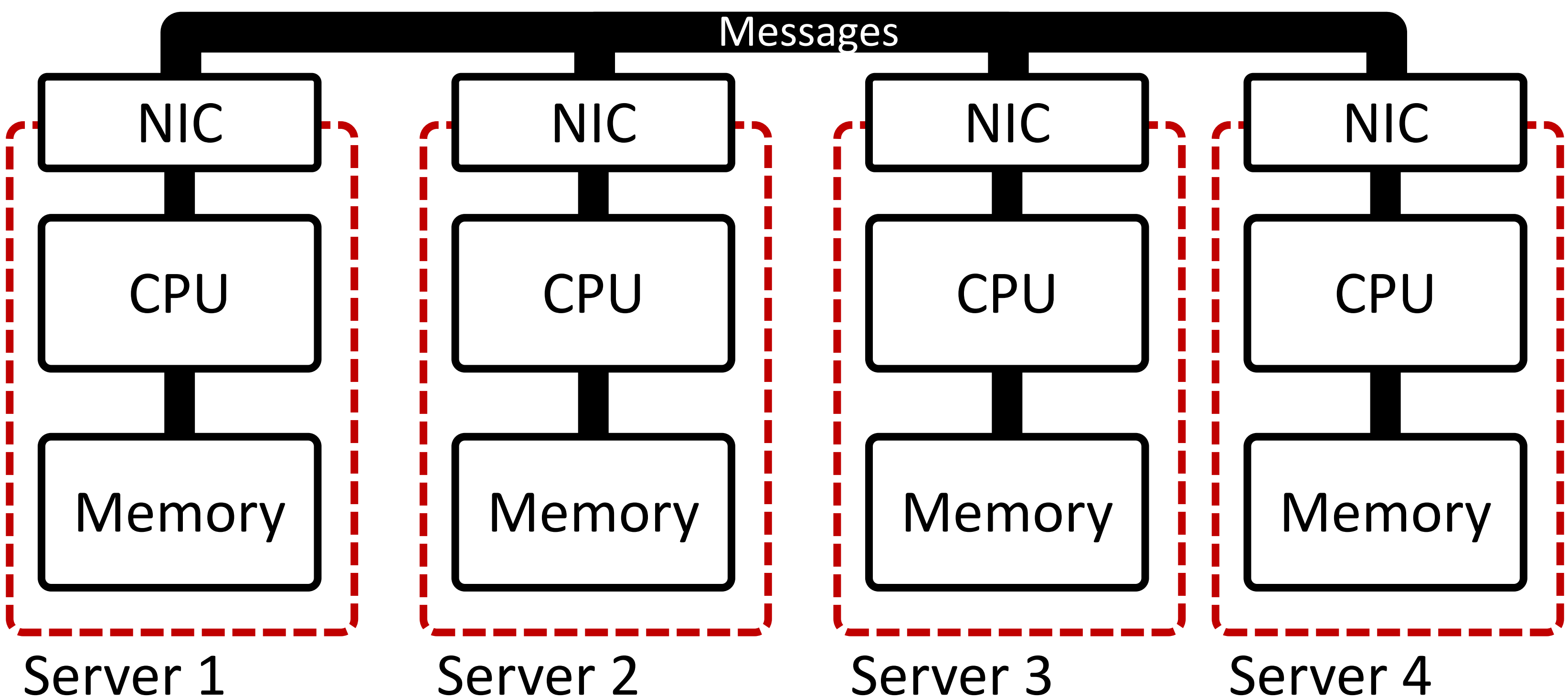


Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM


VV 1882

How do we redesign DBMSs?

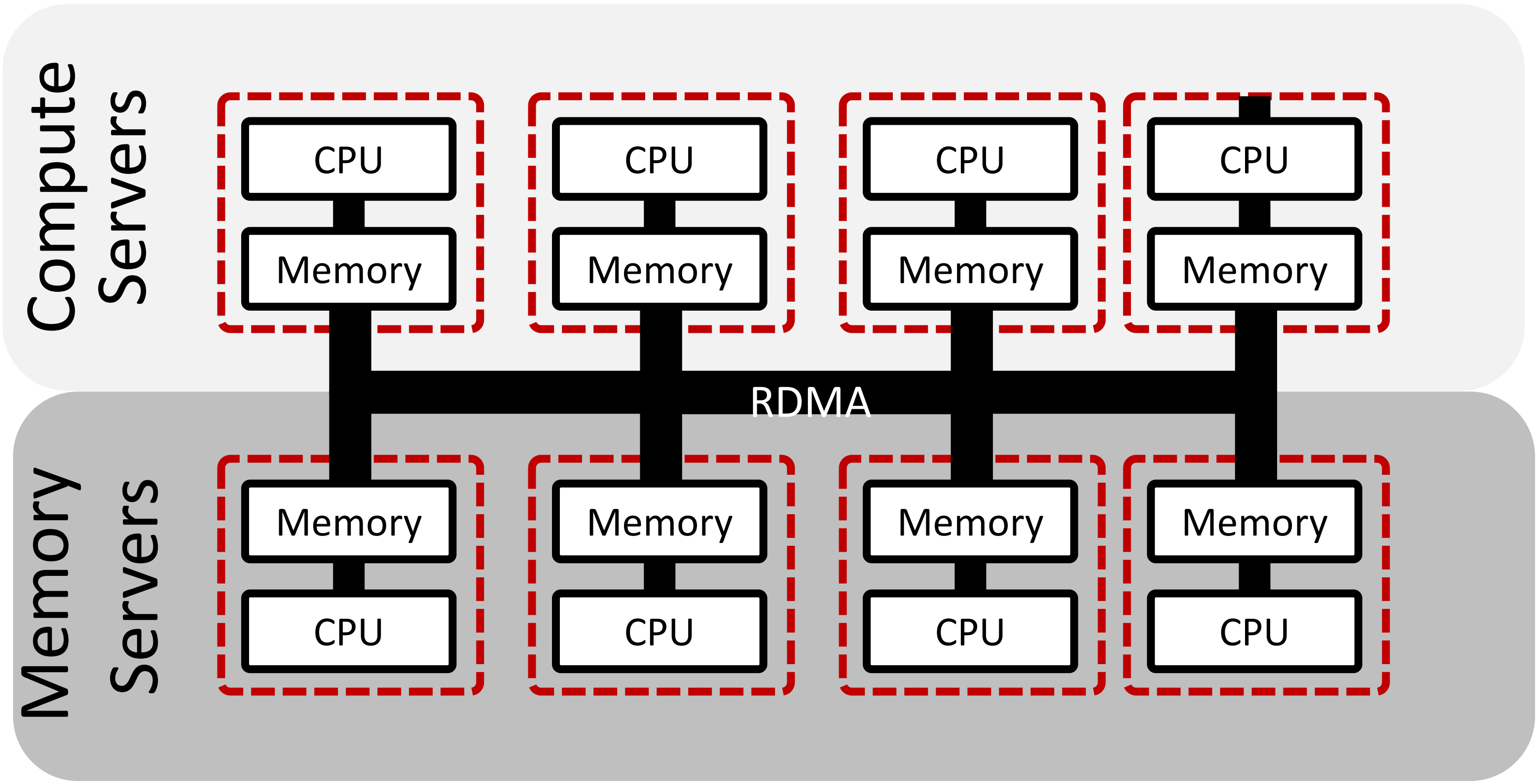
Classical DBMSs: Shared-Nothing



Problems of shared nothing

- Message Passing between nodes using IP stack (IPoIB)
- Bottlenecks due to load imbalance / skew

The Network-Attached-Memory Database Architecture (NAM-DB)

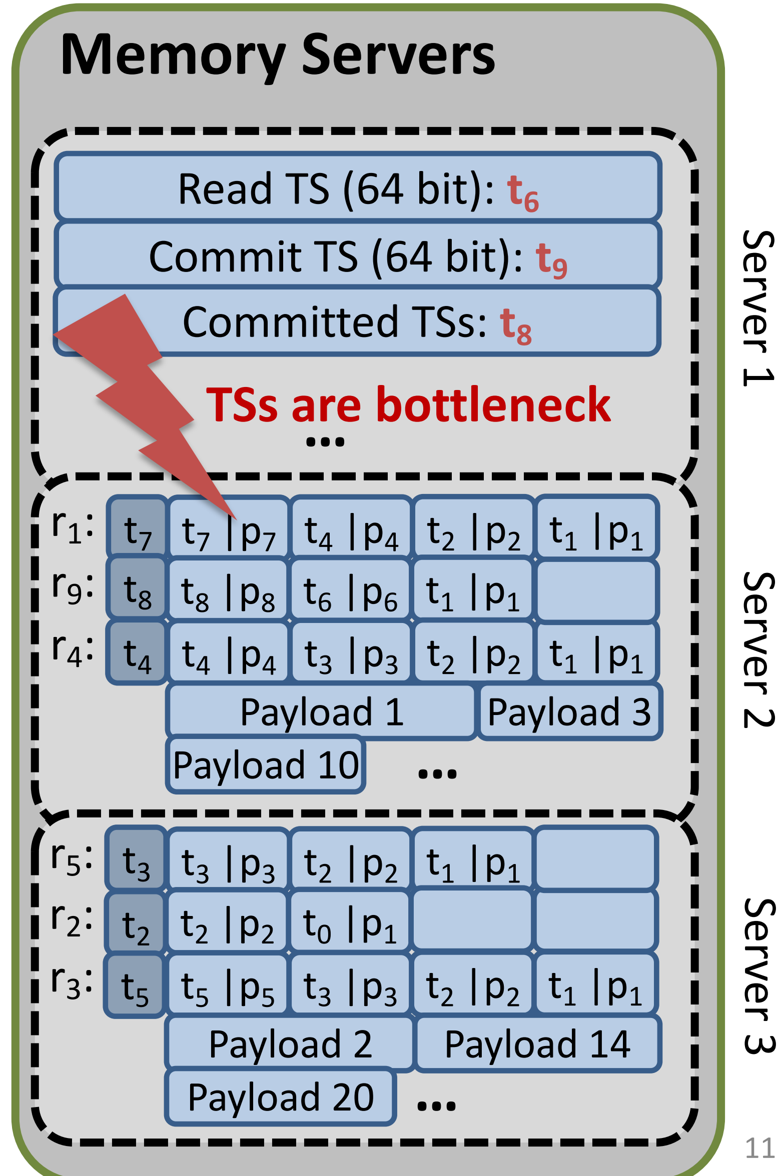


- Use **RDMA for ALL communication**
- **Separate state and compute** -> scalability & load balancing

NAM-DB: Naïve OLTP Protocol
(based on Generalized SI)

Client (Compute Server)

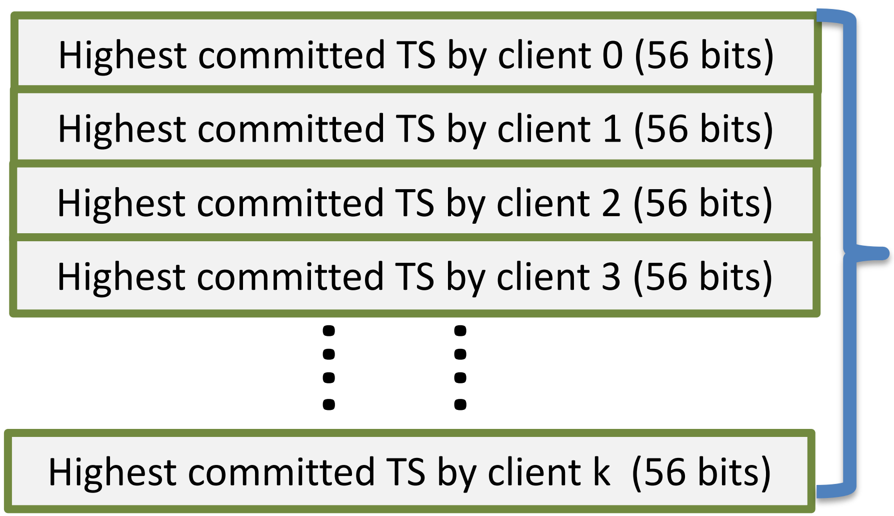
1) **RDMA-READ** "Read TS"
2) **RDMA-READ** *n* (version/pointers)-pairs
3) **RDMA-READ** payload according to "Read TS" (abort if version is no longer available)
4) **RDMA-Atomic-Increment** of "Commit TS"
5) For every record in write-set
   a) **RDMA-Compare-And-Swap** (64bit) the read record version to "Commit TS". If fails, roll-back all changed records.
   b) **RDMA-Write** payload
   c) **RDMA-Write** to install new version by simply replacing n (record/version) pairs
5) **RDMA-Send** "Commit TS" to append "Commit TS" to "Committed TSs

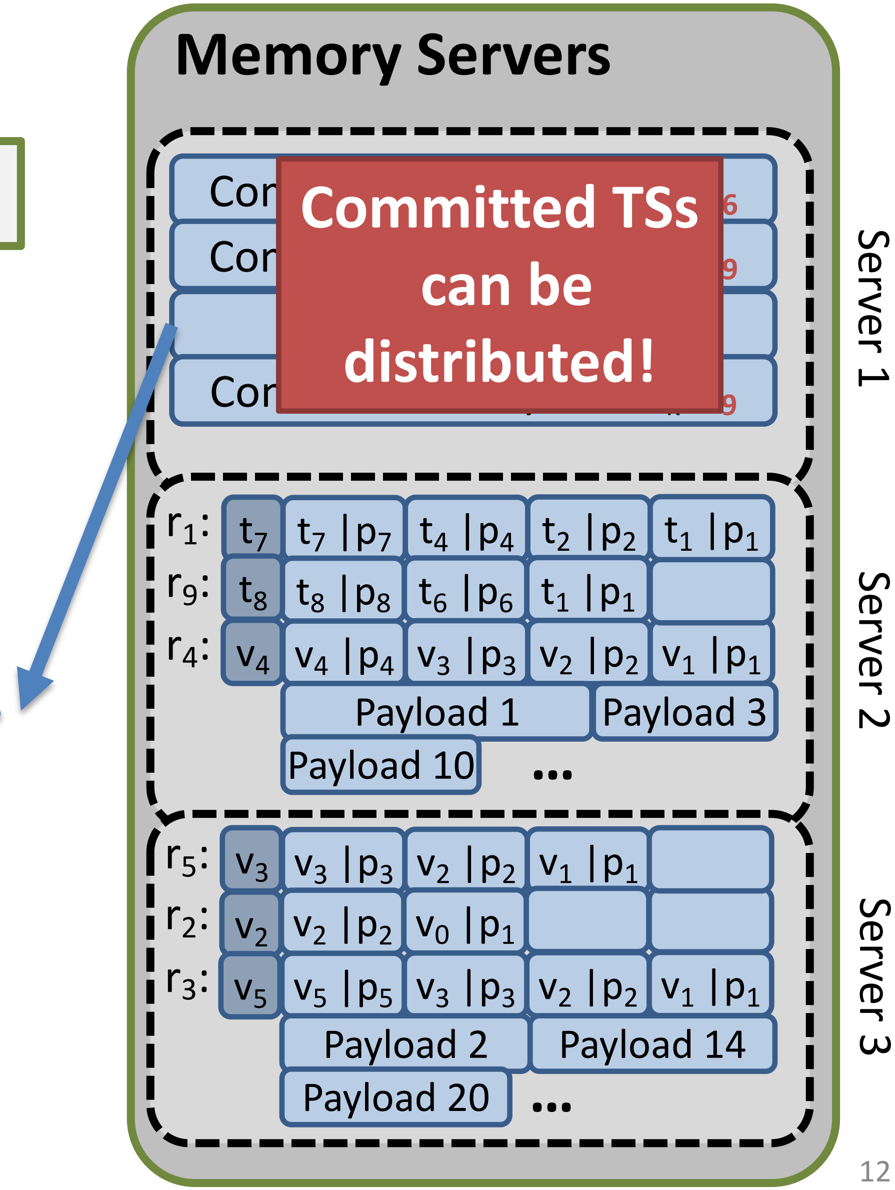


Alternative: Timestamp Vectors

Commit Timestamps

Read Timestamps (vector)





Example: Record and TS Vector

Read-TS

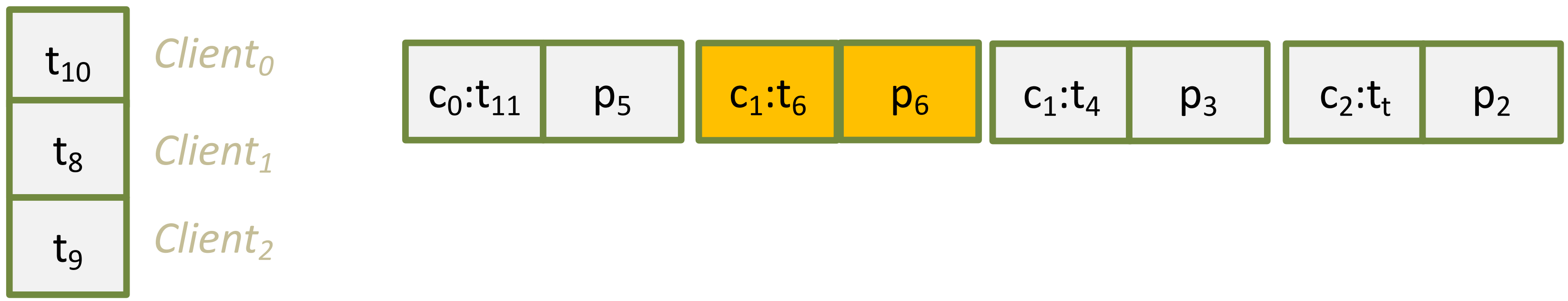
t_{10}	*Client$_0$*
t_8	*Client$_1$*
t_9	*Client$_2$*

Record (Multiple versions)

c_0:t_{11}	p_5	c_1:t_6	p_6	c_1:t_4	p_3	c_2:t_t	p_2

Example: Record and TS Vector

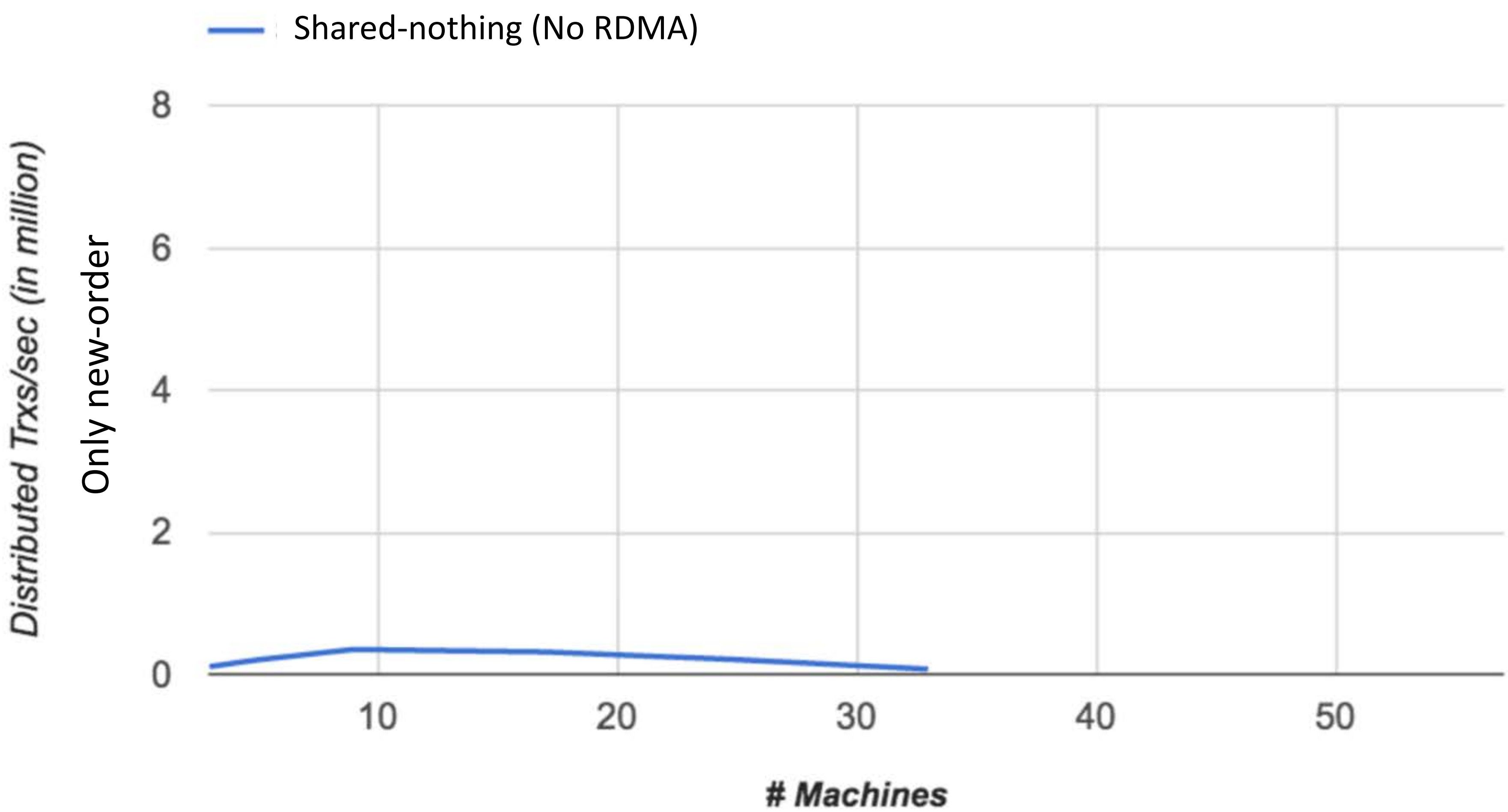
Read-TS **Record** (Multiple versions)



- **Similar to vector-clocks** but not really the same (Read-TS is a vector, a version consist of a single TS)
- **Can still guarantee SI not only generalized SI**
- Avoids problems with **long-running transactions and stale-reads**

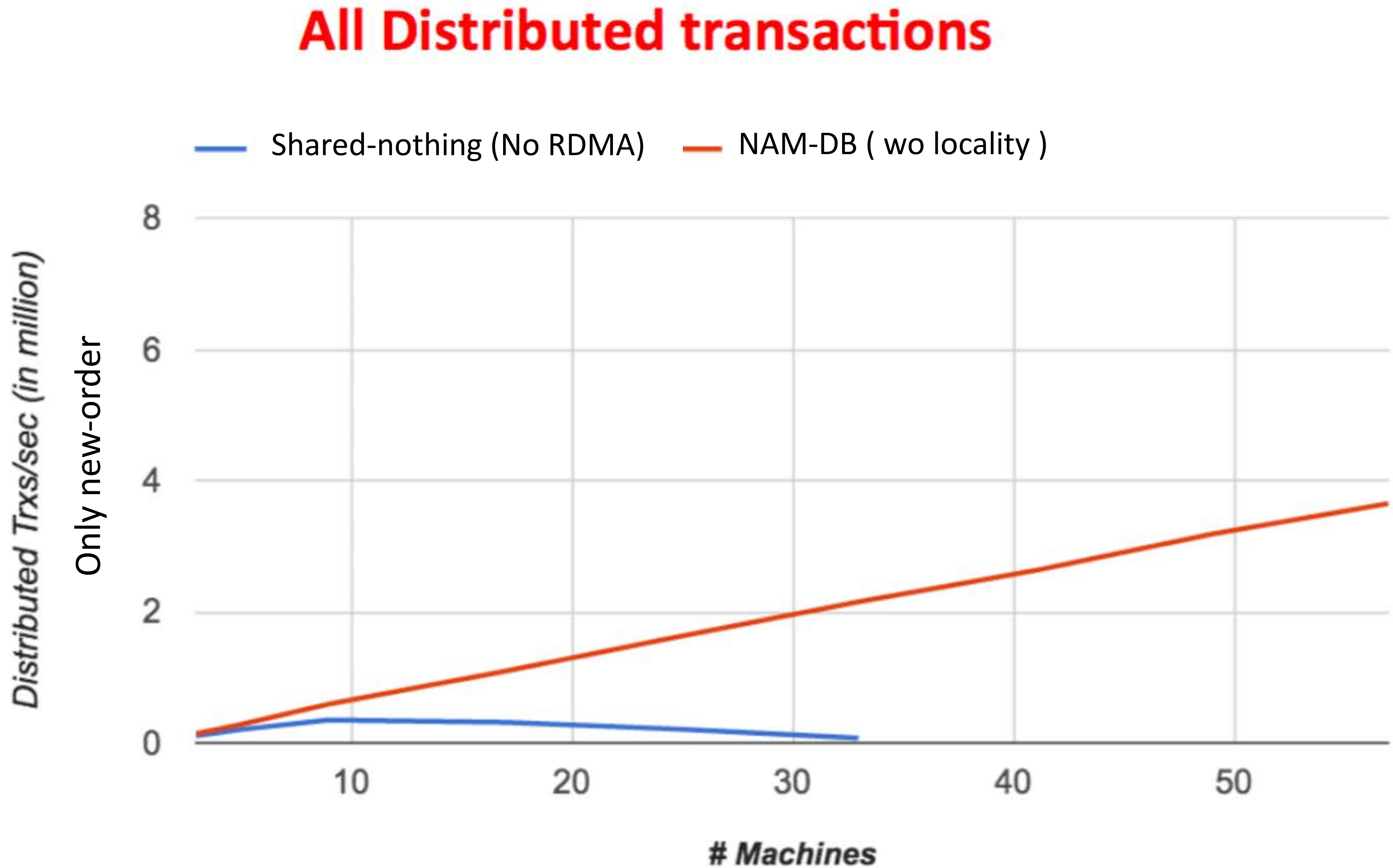
OLTP: Scale-out Experiment on TPC-C

Binnig et al.: The End of A Myth: Distributed Transactions Can Scale. VLDB 2017



Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM

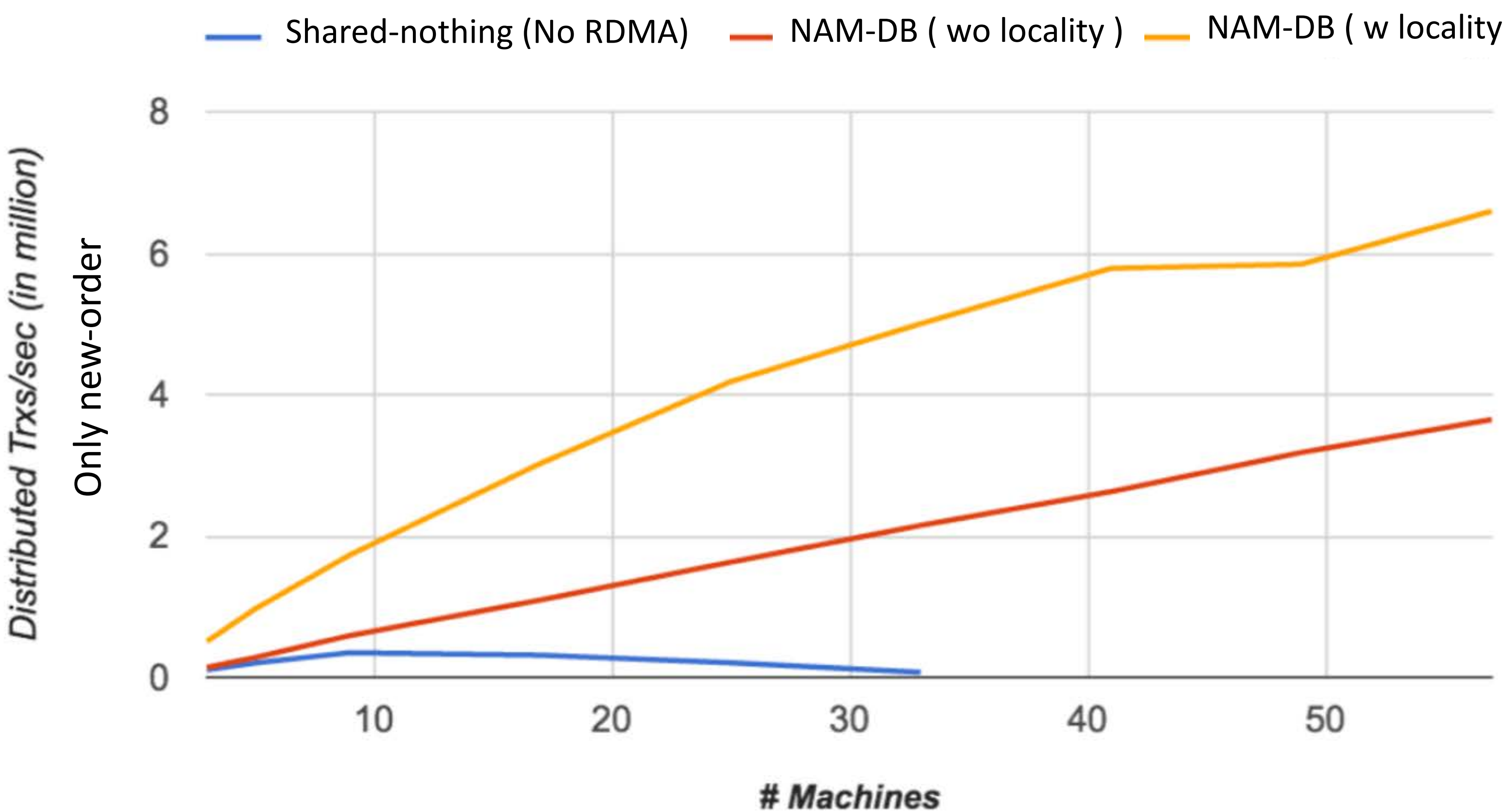
OLTP: Scale-out Experiment on TPC-C



Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM

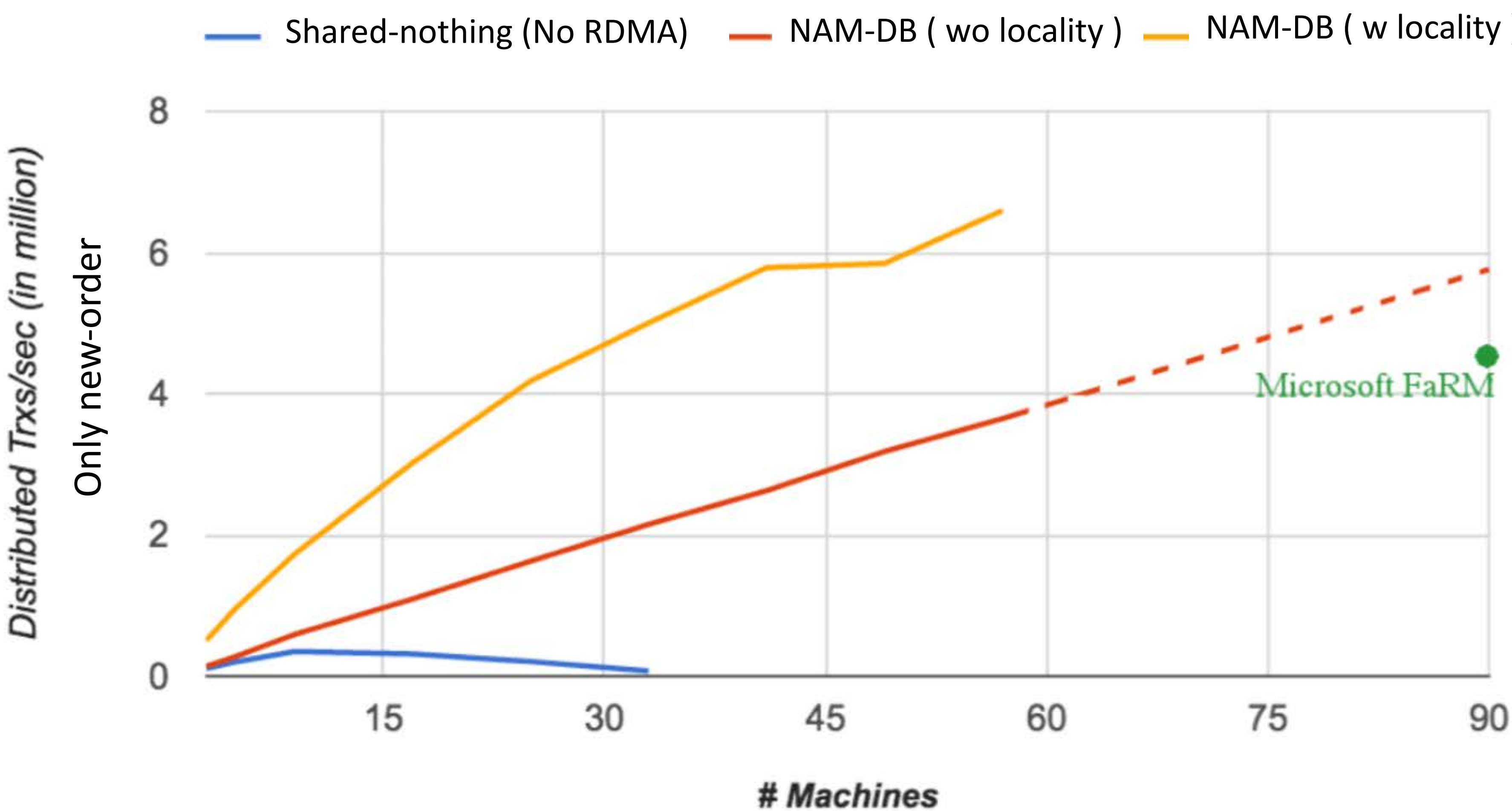
OLTP: Scale-out Experiment on TPC-C

90% local transactions, 10% distributed



Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM

OLTP: Scale-out Experiment on TPC-C



Workload: standard TPC-C, with 50 warehouses per server.
27 machines of type: Two Xeon E7-4820 processors (each with 8 cores), 128 GB RAM
28 machines of type: Two Xeon E5-2660 processors (each with 8 cores), 256 GB RAM
FaRM: From the paper "No compromises: distributed transactions with consistency, availability, and performance"

Many (Important) Details Left Out

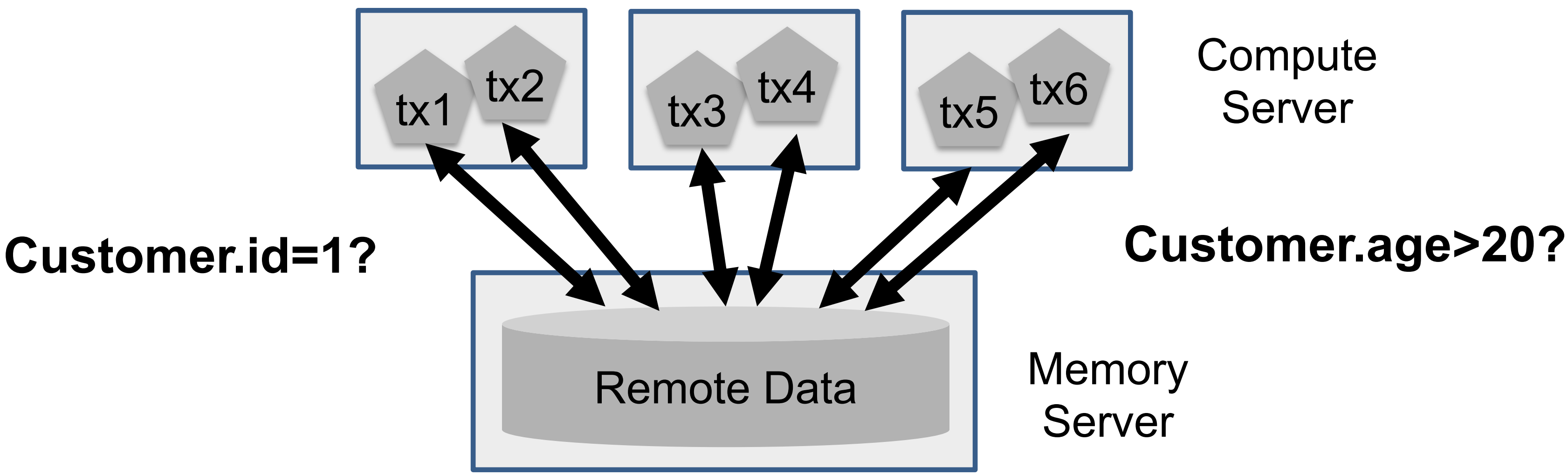
How to find records? *(see next slides)*

Fault-Tolerance, availability, and durability *(NVM, replication and additional checks to undo-transactions of failed clients)*

Many, many possible optimizations *(caches in compute server, extend RDMA verbs by programmable NICs)*

NAM-DB: Remote Table Access

How to enable efficient access of remote tables (key and range lookups) on memory servers?

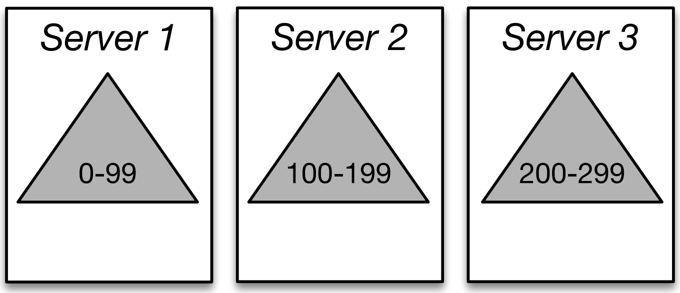


Key Question: How to design of tree-based indexes (i.e., B-tree like indexes) for RDMA?

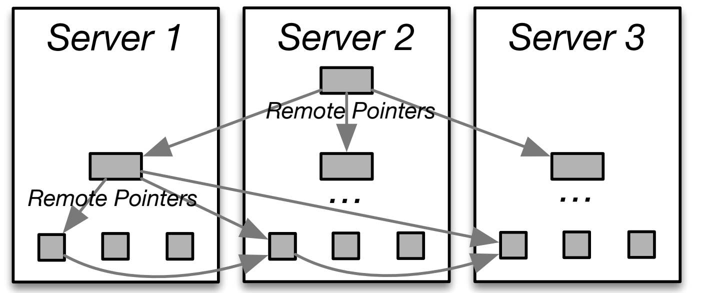
NAM-DB: Remote Indexes

Ziegler et al.: Designing Distributed Tree-based Indexes for RDMA. SIGMOD'19

Index Distribution: How to distribute remote indexes across memory servers?



Coarse-grained Distribution



Fine-grained Distribution

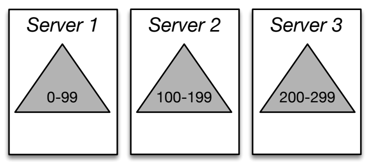
Index Access: How to implement index accesses from compute servers?

- **One-Sided RDMA:** Memory-based (READ / WRITE)
- **Two-Sided RDMA:** RPC-based (SEND / RCV)

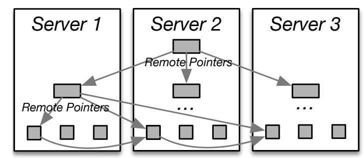
NAM-DB: Index Design Space

The "Design Matrix" for RDMA-based Indexes:

Index Distribution



Coarse-grained Distribution

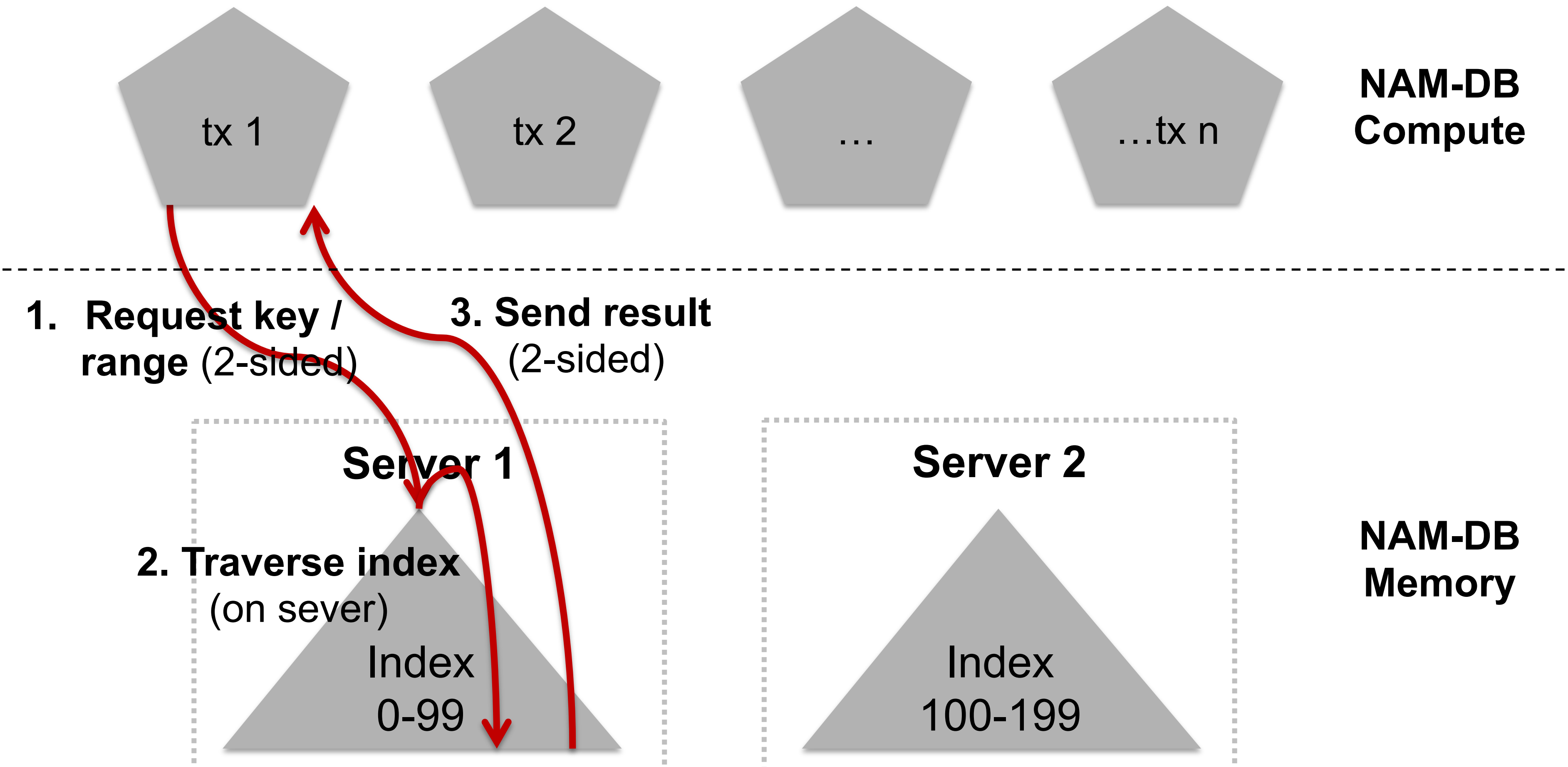


Fine-grained Distribution

Index Access	Coarse-grained Distribution	Fine-grained Distribution
Two-Sided	✓	No benefits over one-sided*
One-Sided	Strictly worse than two-sided	✓

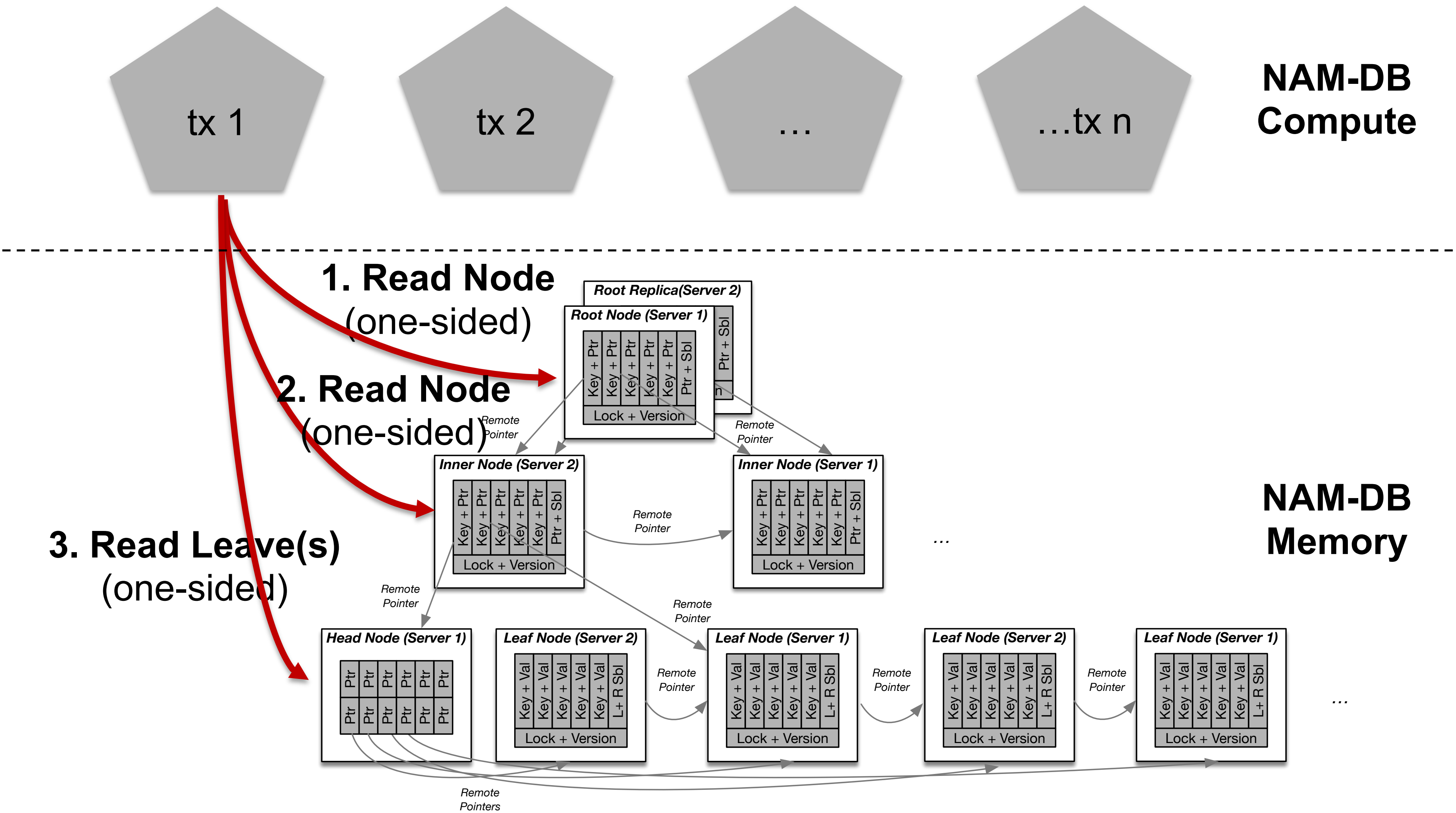
*Assuming that each RDMA access needs to visit a different server

Design 1: Coarse-Grained / 2-sided



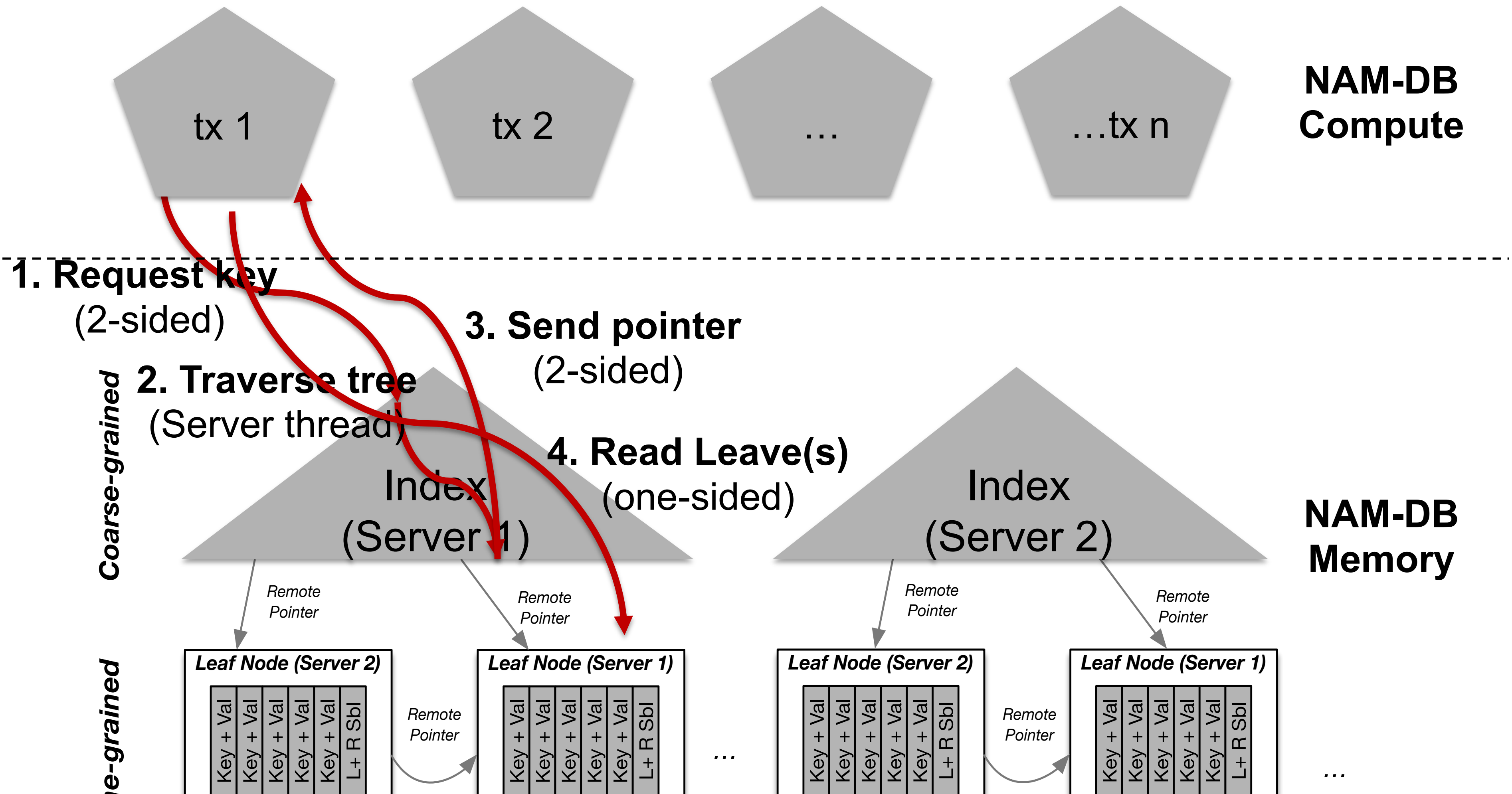
Only one roundtrip BUT sensitive to skewness

Design 2: Fine-grained / 1-sided



Multiple roundtrips BUT better load balancing

Design 3: Hybrid (Fine/Coarse)



One roundtrip for index traversal +
Multiple reads of data for **better load balancing**

NAM-DB: Evaluation (Indexes)

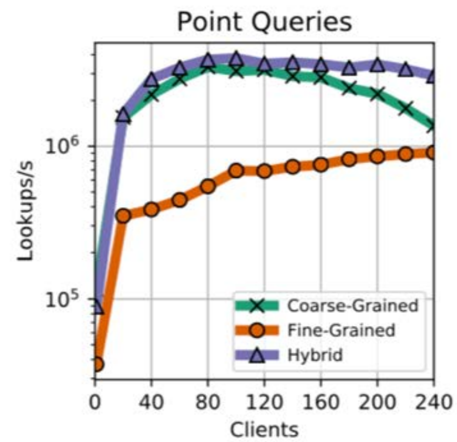
Index Workloads:

Workload	Point Queries	Range Queries (sel=s)	Inserts
A	100%		
B		100%	
C	95%		5%
D	50%		50%

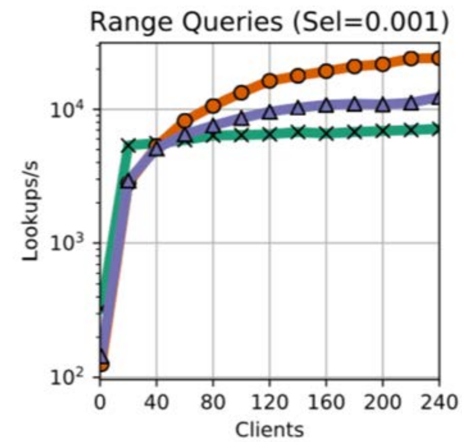
Setup:

- 4 Memory Servers
- 6 Compute Servers
- No co-location
- Data 100M unique keys

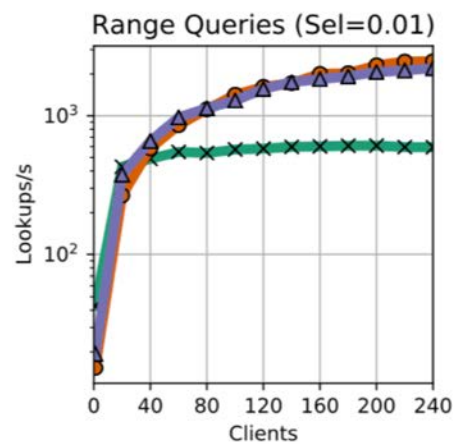
Throughput (Workload A+B, Skewed):



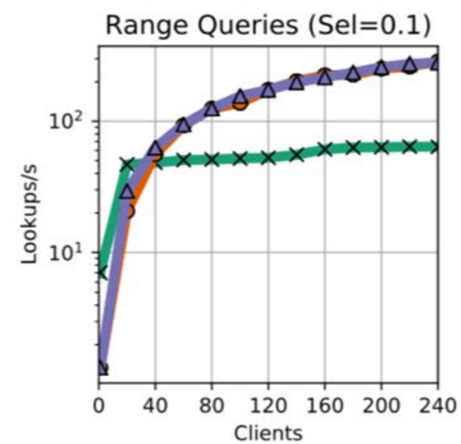
(a) Point Query



(b) Range Query (sel=0.001)



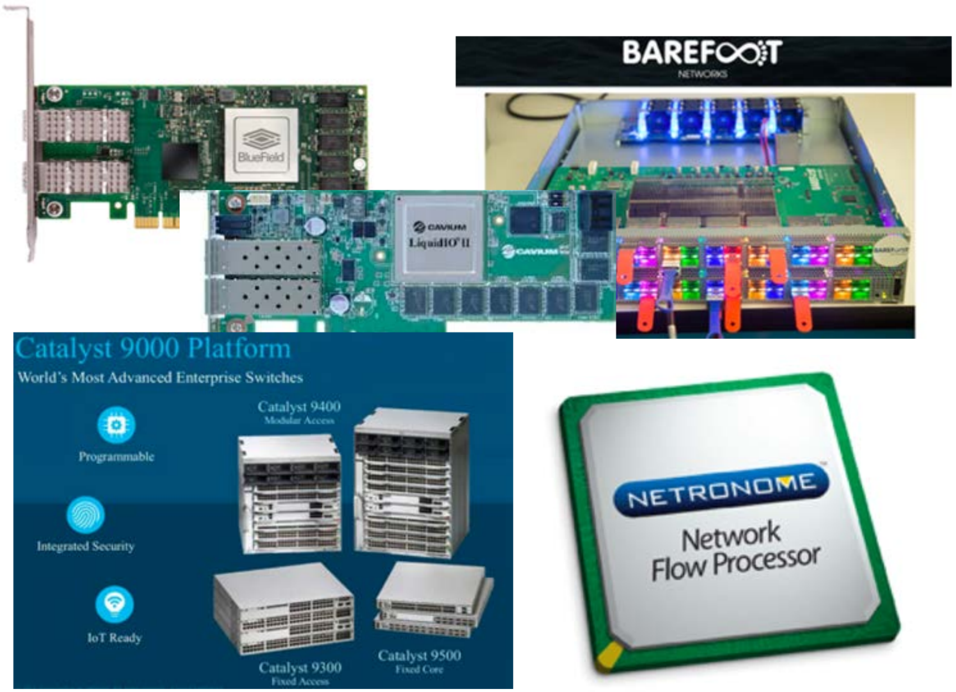
(c) Range Query (sel=0.01)



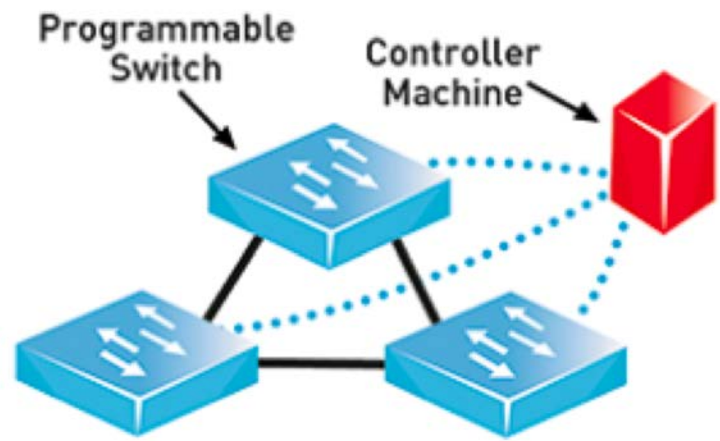
(d) Range Query (sel=0.1)

Networks are becoming smart

Smart NICs & Switches



Software-defined-Networking

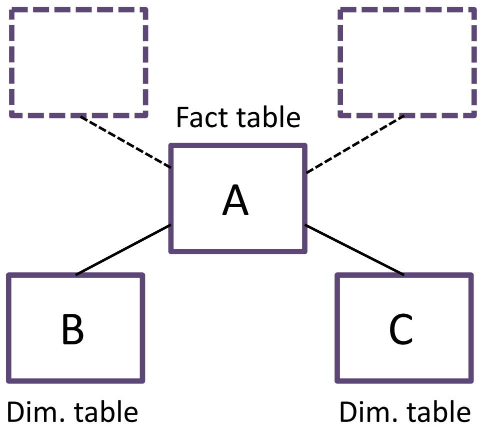


Use **network to offload computation** from a distribute DBMS
→ **In-Network-Processing** (INP) of SQL operators?

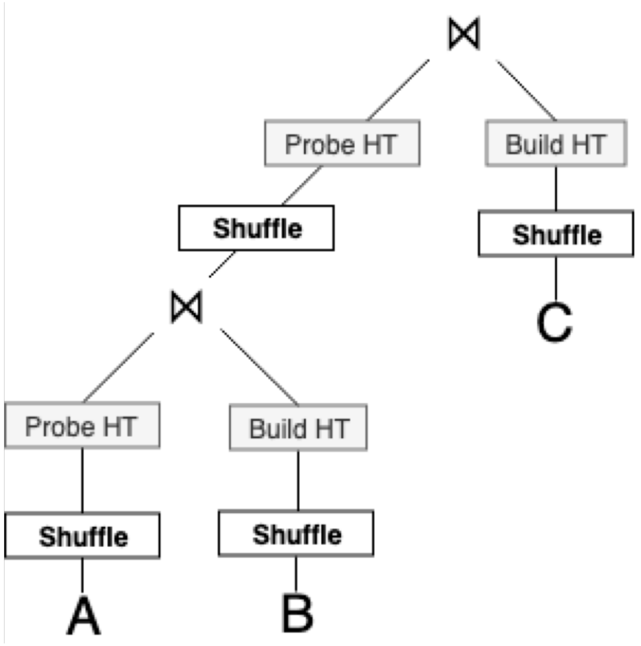
A motivating example

- **Data warehousing scenario:** star schema
- Fact table **A not co-partitioned** with dimensions B and C

Star Schema:



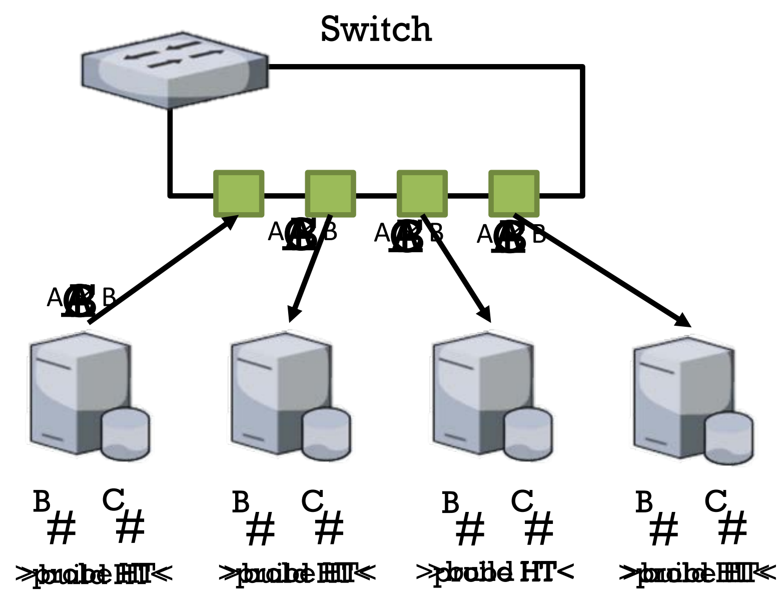
SELECT * FROM A JOIN B JOIN C



Traditional Distributed Execution

Steps:

1. Shuffle table B & build HT
2. Shuffle table A & probe HT of B
3. Shuffle table C & build HT
4. Shuffle intermediate A ⋈ B & probe HT of C



Observation:

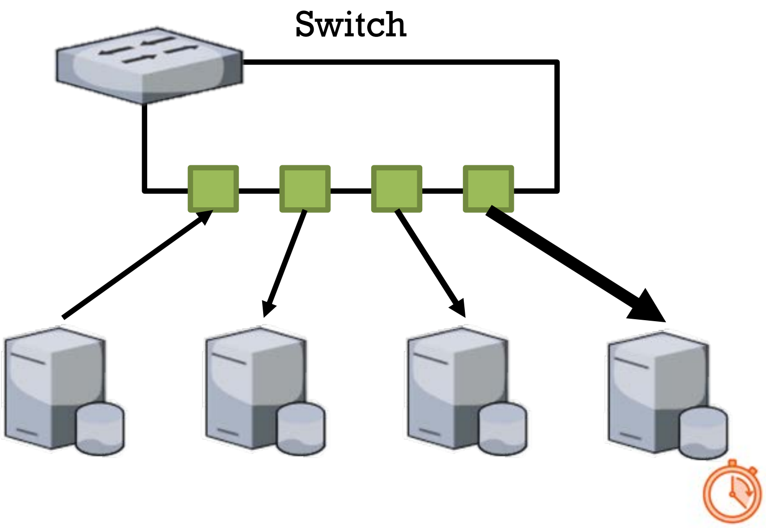
Re-shuffling of large fact table A is expensive

Moreover, skew is a problem

Non-uniform foreign-key distribution → **network skew**

Network link to one node is getting **congested**

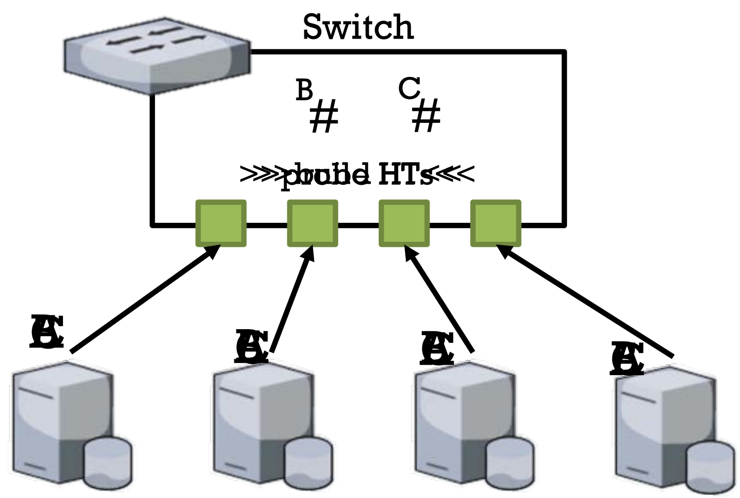
- Increased shuffling time
- Increased processing time on straggler



Case for In-network Processing

Steps:

1. Send table B and C to switch and build HT in switch
2. Stream fact table A through switch & probe HTs



Take away:

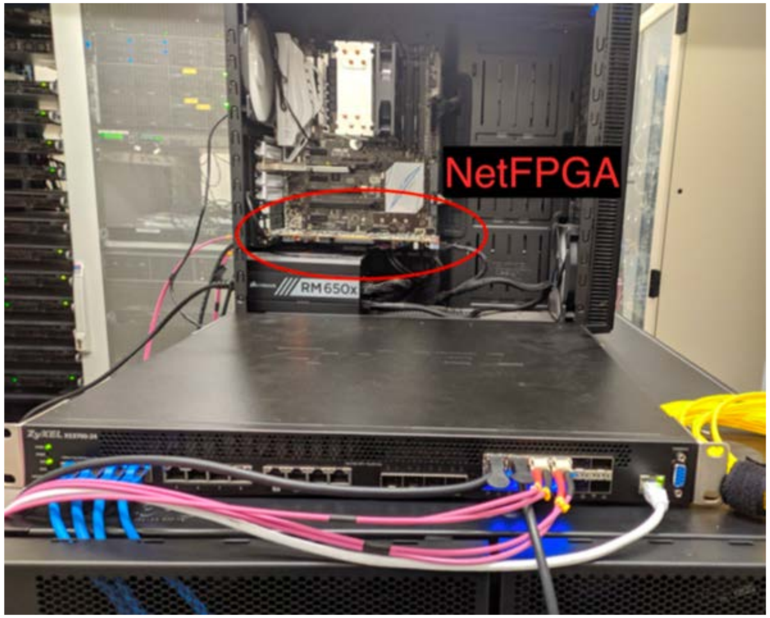
- Avoids re-shuffling of large fact table A
- Not sensitive to skew

Custom switch prototype

Current **P4 switches** (e.g., Barefoot Tofino) have **many limitations**

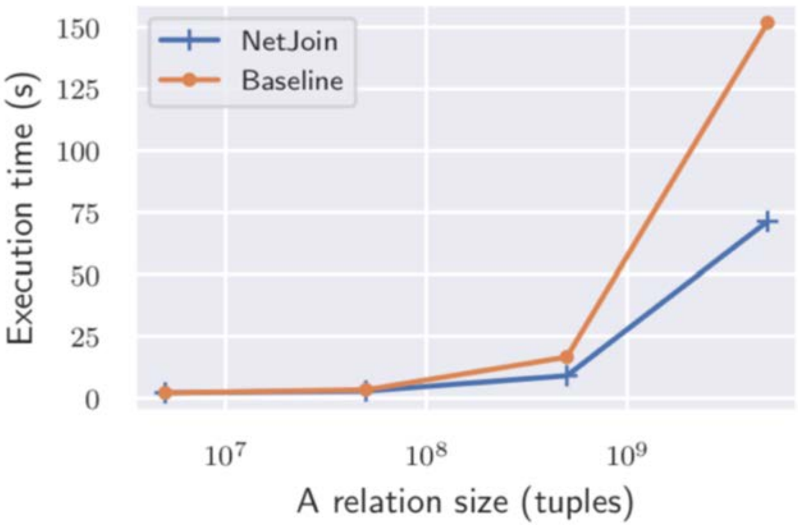
Our own **prototype switch:**

- FPGA chosen as processing unit
- Based on network focused FPGA dev board (NetFPGA SUME)
- 2 x 4GB DDR3 memory @ 800MHz

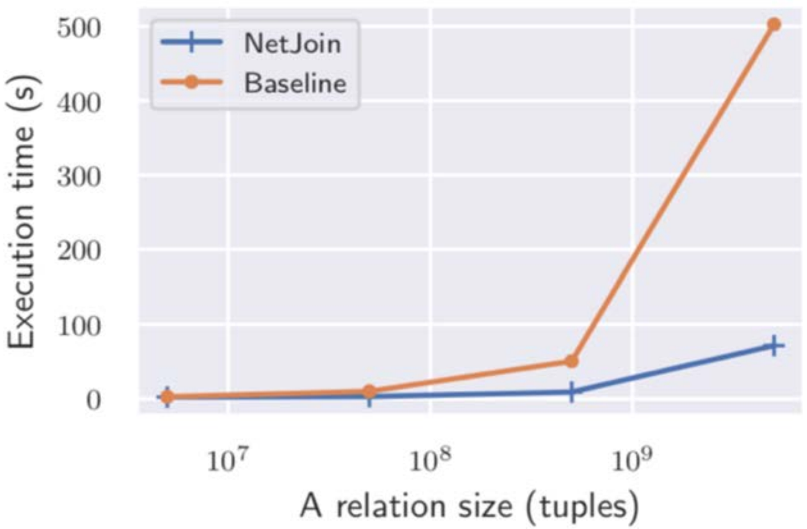


INP-Join: Experimental Evaluation

Query: A ⋈ B ⋈ C ⋈ D
Data: A: 5 × 10^6 to 5 × 10^9 tuples - B, C & D: 50 × 10^6 tuples



Without Skew



With Skew

Conclusions and Future Work

The next generation of **high-speed networks** requires us to **rethink distributed database systems**

Network-Attached Memory (NAM) as a general distributed architecture to take advantage of fast networks

Other workloads: Streaming, ML, Graphs, …

Networks are not only getting **“faster”** but also **“smarter”**

Collaborators