Divide and Conquer Data

Advanced Methods for partitioning and sharding data - Latest developments

Jobin Augustine
Senior Support Engineer / DBA

PERCONA

Agenda

- Alternate Schools of scale-up
- Numbers everyone should know - 2019 Review
- New Improvements in hardware favouring sharding
  - New trends in SSDs and Non-Volatile memory
  - Hyper Convergence
- PostgreSQL Performance Numbers everyone should know
- PostgreSQL Partitioning
- PostgreSQL simple sharding and new improvements
- Advanced sharding options
- Externally shared systems.

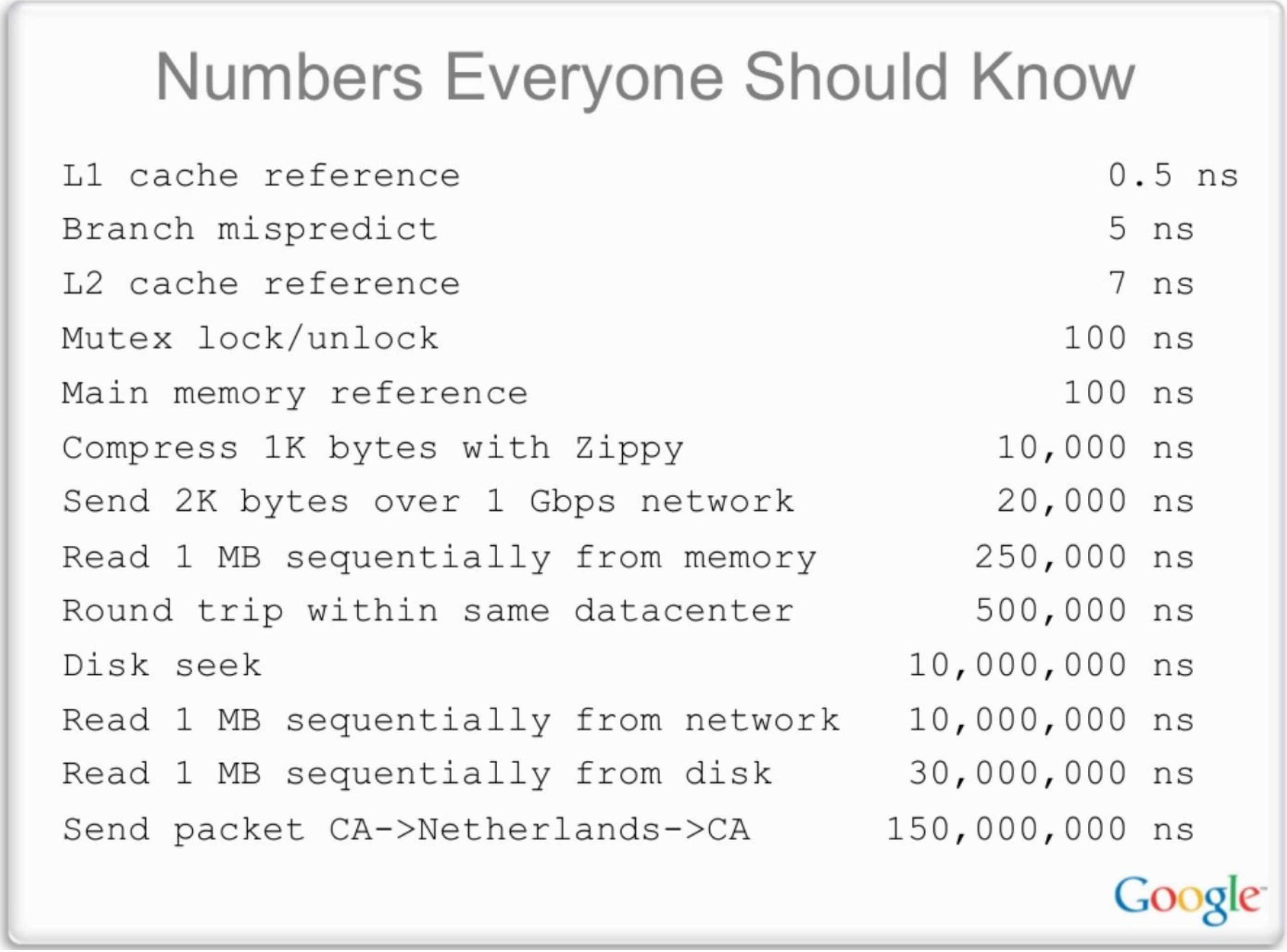
Alternate Schools

Alternate Schools of thoughts

1. Expensive Big monolithic systems
    - More memory, Processing, faster storage
2. Standbys and Reporting-Read Split
3. **Multi master cluster**
    a. Shared disk clusters
    b. Mutual replication clusters

Ever evolving Hardware

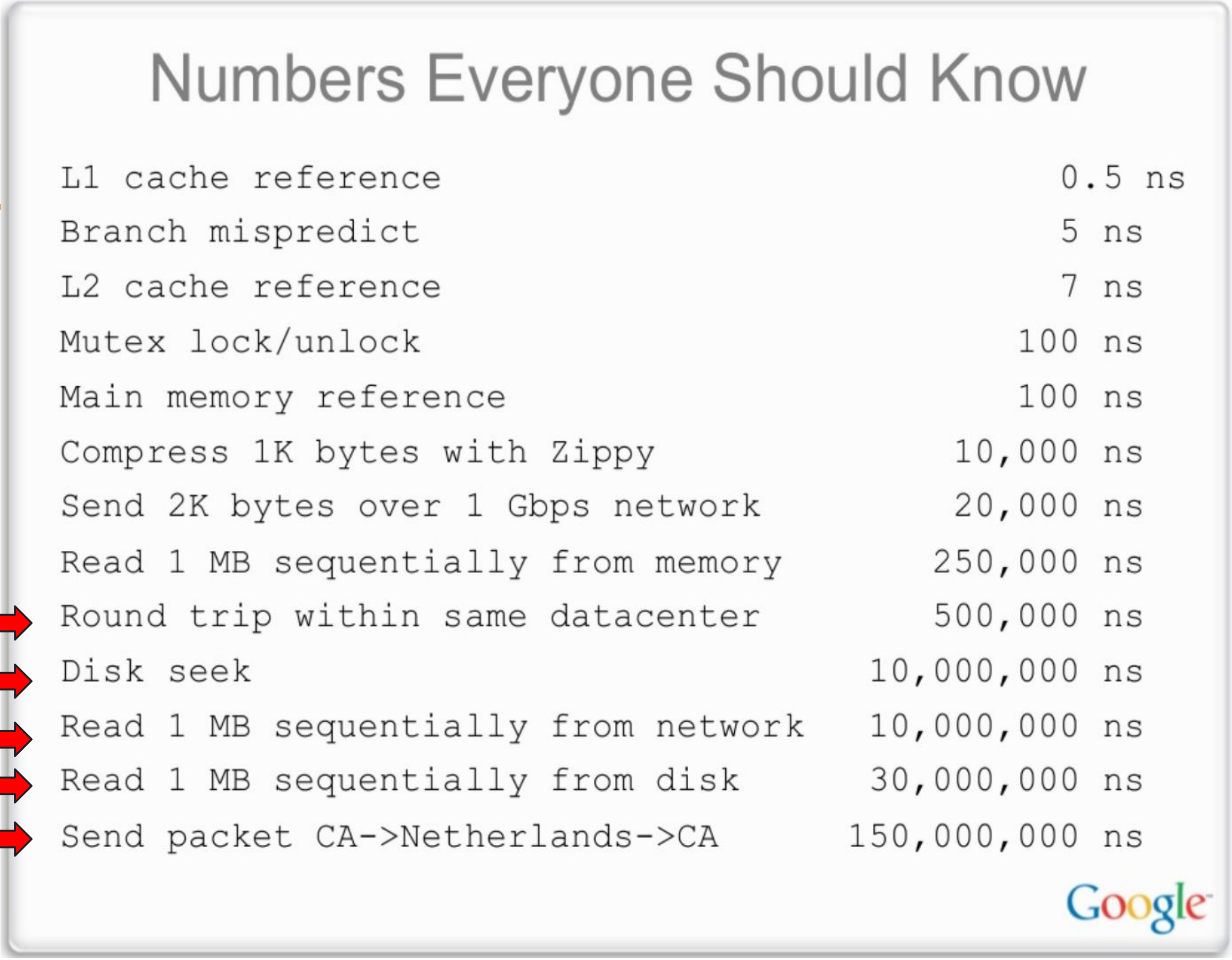
Important changes in hardware that affect the database design



Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Courtesy : Jeff Dean, Google



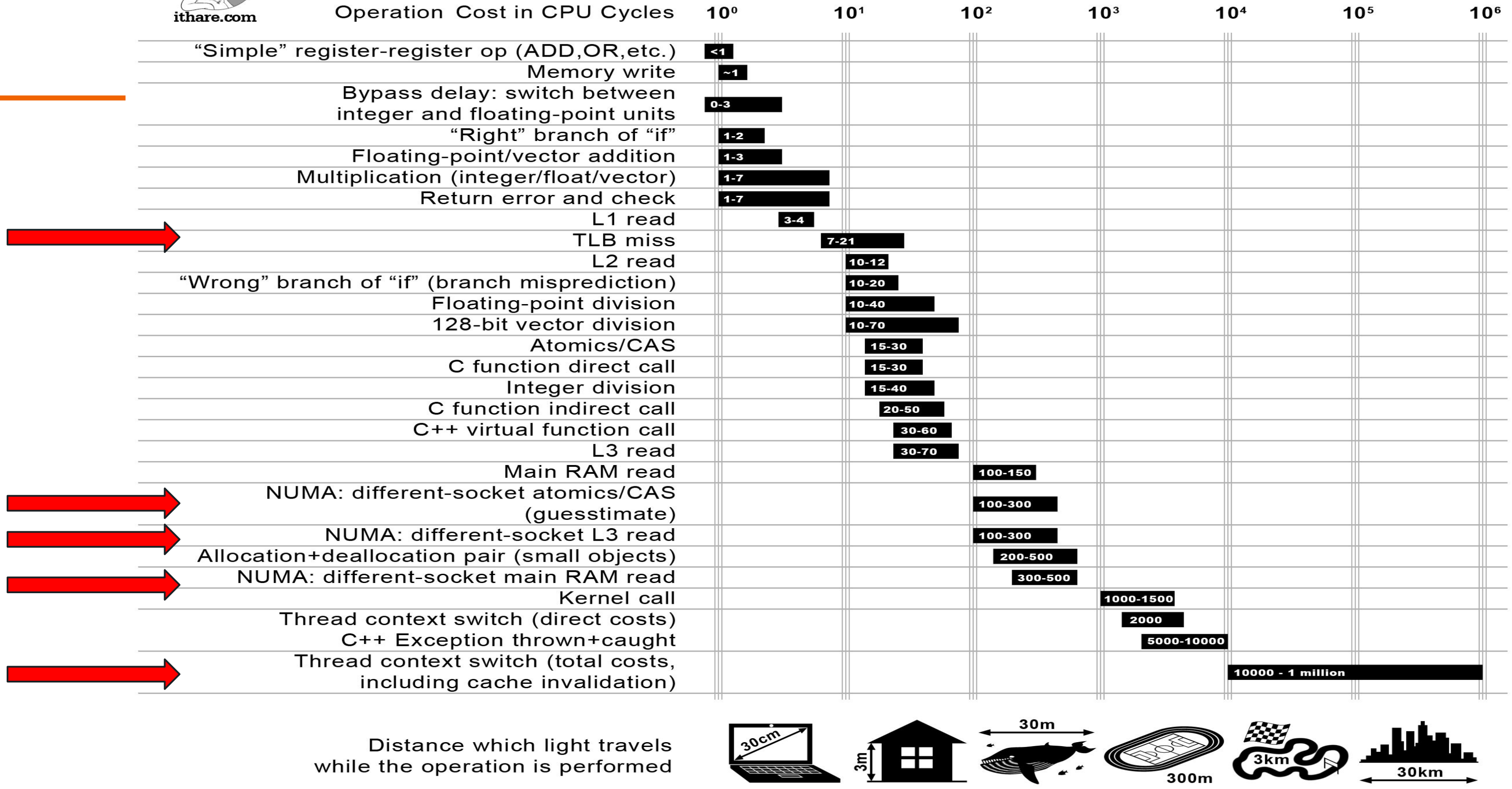
Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Google

ithare.com

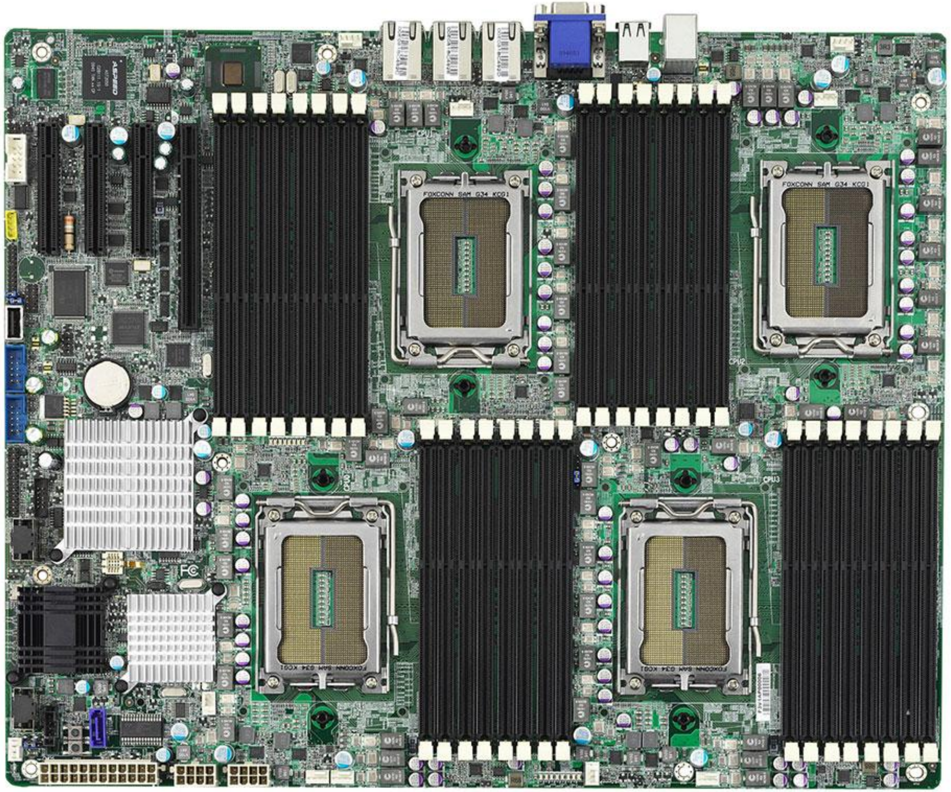
Not all CPU operations are created equal



Operation	Cost in CPU Cycles
"Simple" register-register op (ADD,OR,etc.)	<1
Memory write	~1
Bypass delay: switch between integer and floating-point units	0-3
"Right" branch of "if"	1-2
Floating-point/vector addition	1-3
Multiplication (integer/float/vector)	1-7
Return error and check	1-7
L1 read	3-4
TLB miss	7-21
L2 read	10-12
"Wrong" branch of "if" (branch misprediction)	10-20
Floating-point division	10-40
128-bit vector division	10-70
Atomics/CAS	15-30
C function direct call	15-30
Integer division	15-40
C function indirect call	20-50
C++ virtual function call	30-60
L3 read	30-70
Main RAM read	100-150
NUMA: different-socket atomics/CAS (guesstimate)	100-300
NUMA: different-socket L3 read	100-300
Allocation+deallocation pair (small objects)	200-500
NUMA: different-socket main RAM read	300-500
Kernel call	1000-1500
Thread context switch (direct costs)	2000
C++ Exception thrown+caught	5000-10000
Thread context switch (total costs, including cache invalidation)	10000 - 1 million

Distance which light travels while the operation is performed

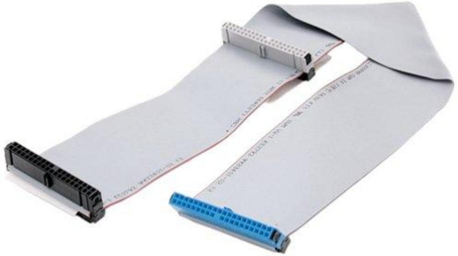
NUMA



Storage connectivity

- IDE - ATA (Parallel ATA)
- SATA

- HBA Cards



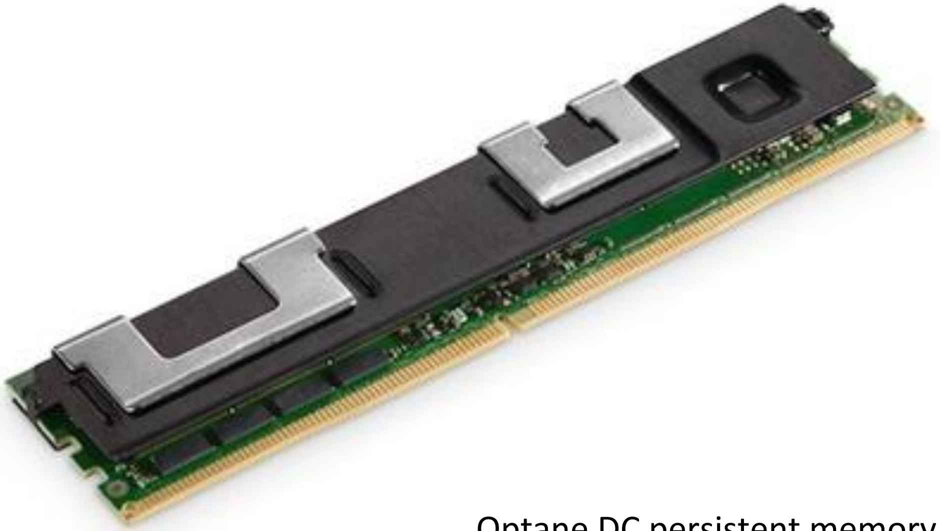
SCSI - *The SCSI standards define commands, protocols, electrical, optical and logical interfaces*

- Cables/Wires and their limitations of transporting data
- Laws of Physics and Noise

- 500k to I Million IOPs
- M.2 overcoming the Limitations of older interface

They can bring data closer to processing reducing latency

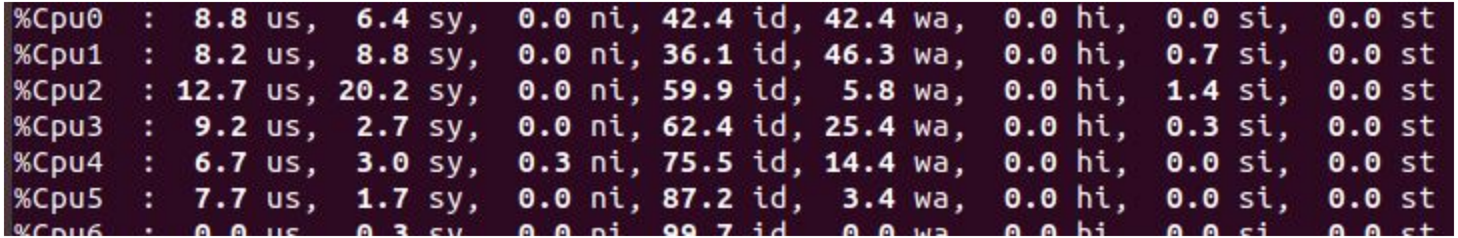
Persistent Memory over DIMM



Optane DC persistent memory

- byte-addressable
- persistent memory DIMMs
- DDR4 bus interface
- New Processors and new instruction set

Why Storage

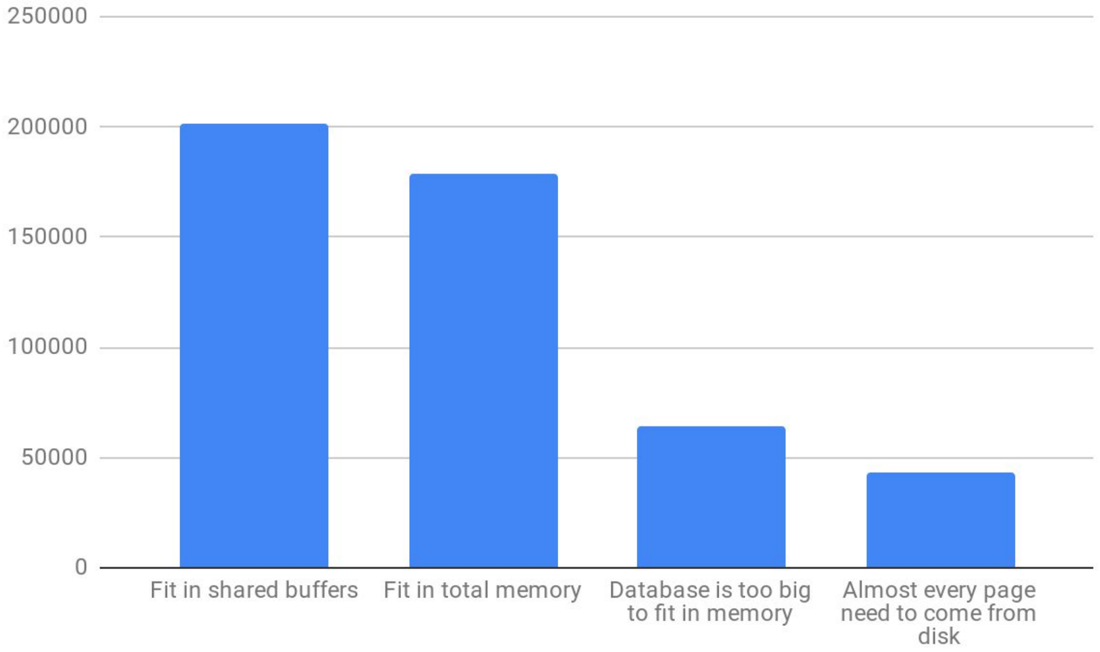


```
%Cpu0  :  8.8 us,  6.4 sy,  0.0 ni, 42.4 id, 42.4 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  8.2 us,  8.8 sy,  0.0 ni, 36.1 id, 46.3 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu2  : 12.7 us, 20.2 sy,  0.0 ni, 59.9 id,  5.8 wa,  0.0 hi,  1.4 si,  0.0 st
%Cpu3  :  9.2 us,  2.7 sy,  0.0 ni, 62.4 id, 25.4 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu4  :  6.7 us,  3.0 sy,  0.3 ni, 75.5 id, 14.4 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  :  7.7 us,  1.7 sy,  0.0 ni, 87.2 id,  3.4 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  :  0.0 us   0.3 sy   0.0 ni  99.7 id   0.0 wa   0.0 hi   0.0 si   0.0 st
```

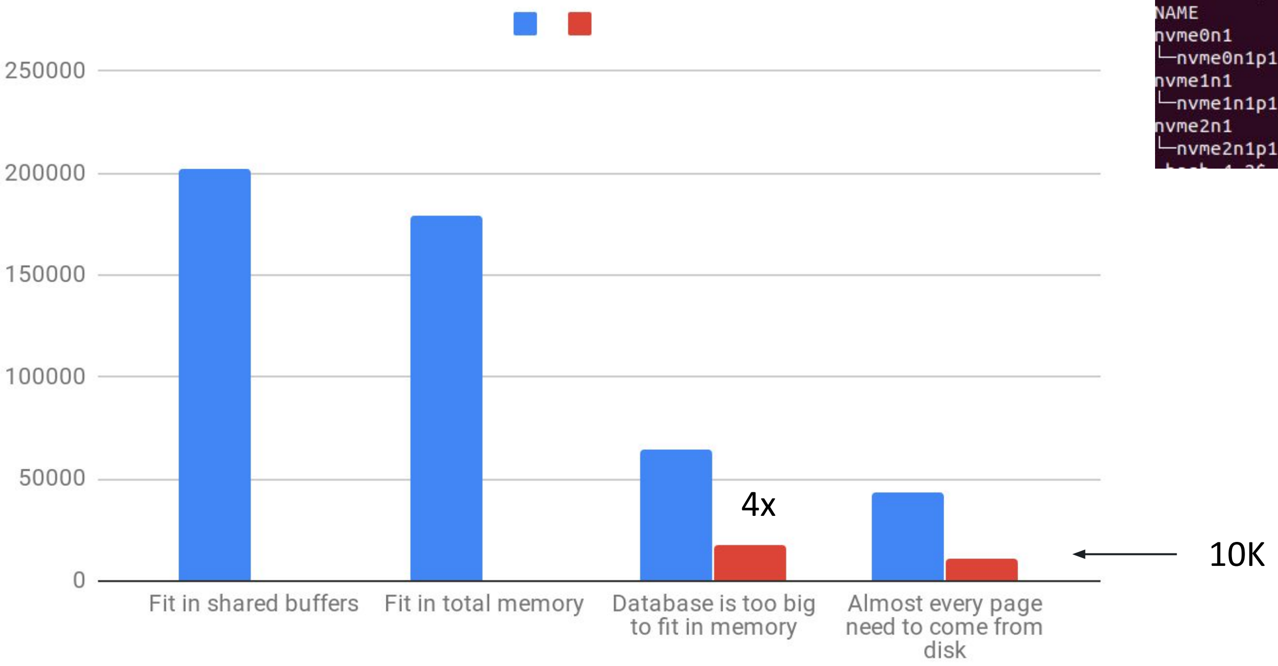
A database is all about persistently store data and retrive data

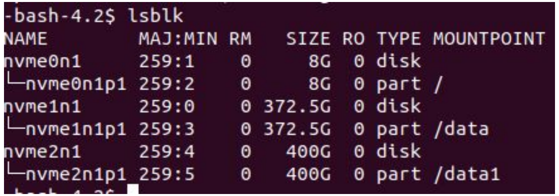
Database performance numbers

Single node NVMe



local vs remote storage





```
-bash-4.2$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
nvme0n1     259:1    0     8G  0 disk
└─nvme0n1p1 259:2    0     8G  0 part /
nvme1n1     259:0    0 372.5G  0 disk
└─nvme1n1p1 259:3    0 372.5G  0 part /data
nvme2n1     259:4    0   400G  0 disk
└─nvme2n1p1 259:5    0   400G  0 part /data1
```

Key Points

Separating storage and accessing the remote storage is getting as bad as accessing remote RAM

- **Storages is getting faster and faster today**
- **Local storage is becoming more important**
- Bigger memory is not efficient

Importance of Independent Computing Units

Partitioning

Getting maximum out of single node

Partitioning Advantages

Partition pruning

Added Advantages:

- Small Working-set of data
- Small indexes
- Vacuum benefits
- Retention policies
- Tablespaces and different disks

Impact on Vacuum

- **Typically vacuum kicks in when you have 20% dead tuples.**
    - 100 GB table can have 20GB dead tuples
- **lots of data it need to hold and process in maintenance_work_mem and complexity of indexes.**

Traditional Solutions:

- DBAs tweeks Autovacuum parameters the for aggressiveness.
- IO overhead of scanning the table and indexes more frequently

Impact on Memory

Handling bigger tables and associated bigger index requires more memory.

Undivided data = Bigger active data set.

Strategy of fitting active dataset into shared_buffers

Risk of falling from the cliff of bigger shared_buffers.

Partitioning automation : pg_partman

`SELECT create_parent(table_name …)`

- `Partitioning for older versions of pg.`
- `Currently supports native partitioning`
- `Adds and deletes partitioning`
- `Background worker for partition maintenance`

`pg_partmaint - Super Simple partition maintenance for native partitioning`

Simple Shards

Application level shards and postgres_fdw as a sharding solution

Application level shards

- Application awareness
- Avoid statement routing.
- Isolating unavailability.
- Application + DB scaling.

Sharding using Buit-in Features

Advancements in :

Postgres_fdw + Partitioning + Parallelism

- Individual partitions can be foreign tables

Postgres_fdw feature

- Predicate pushdown
- Aggregate pushdown
- Join pushdown
- Partition Wise join

Areas to improve

- Parallel execution
- smarter planner
- DMLs

```
CREATE FOREIGN TABLE [ IF NOT
EXISTS ] table_name
  PARTITION OF parent_table [ (
  { column_name [ WITH OPTIONS ]
[ column_constraint [ ... ] ]
    | table_constraint }
    [, ... ]
) ] partition_bound_spec
  SERVER server_name
[ OPTIONS ( option 'value' [, ...
] ) ]
```

Advanced Sharding

Extending PostgreSQL

Extensions for PostgreSQL

- Pg_shard and Citus data
- Timescale DB
- External databases and FDWs

pg_shard

- **Data is cut into small chunks and distributed into worker nodes**
  - Each table is splitted into many shards.
- **Worker nodes stores data.**
  - One shard of a table is one table in the worker node.
  - Automatically shard tables are named
- **Metadata server - coordinator node**
  - Holds repository about shards (only few MBs)
  - where we create extension and shard table.
  - Place to send queries
  - Queries are analyzed to find out the right shard.

Citus Extension

Implemented as an Extension

- Go deep into PostgreSQL extension API to override query planner
- Query will be planned for shards.
- Data load will get faster to shared cluster (millions of TPS is easy) due to parallel load
- OLAP Load and Roll-up tables

```
SELECT create_distributed_table(table_name,colum_name
```

Time Series Data

Architecture

- Past and Present
- Ledger

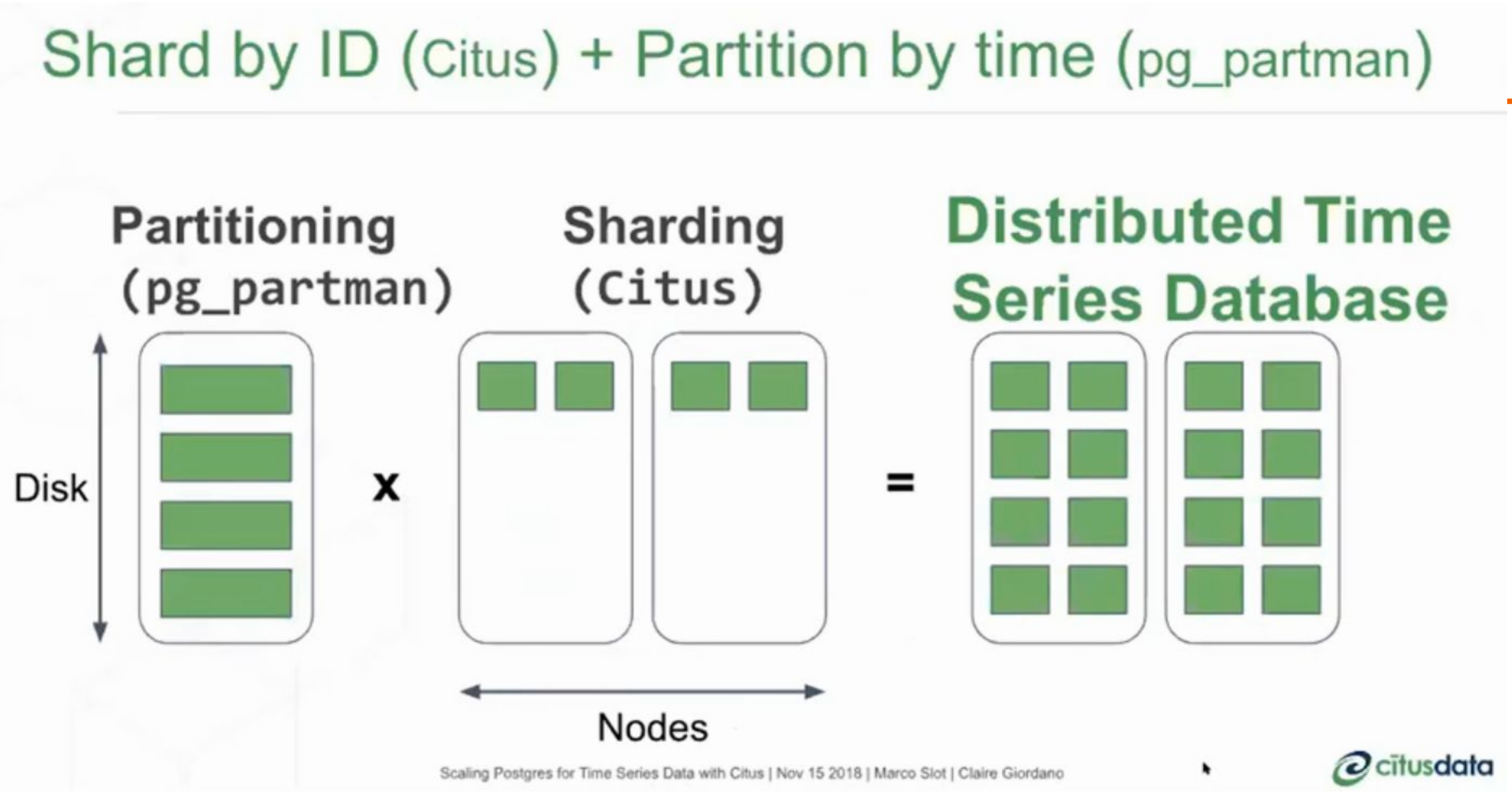
Applications

- Universally applicable
- IOT
- Monitoring
- Weather
- Satelite

Implication

- Large Volume of data
- Primary key cannot be timestamp in general.
  - *Need an secondary index - B-tree*

When you update a data, you are losing old data

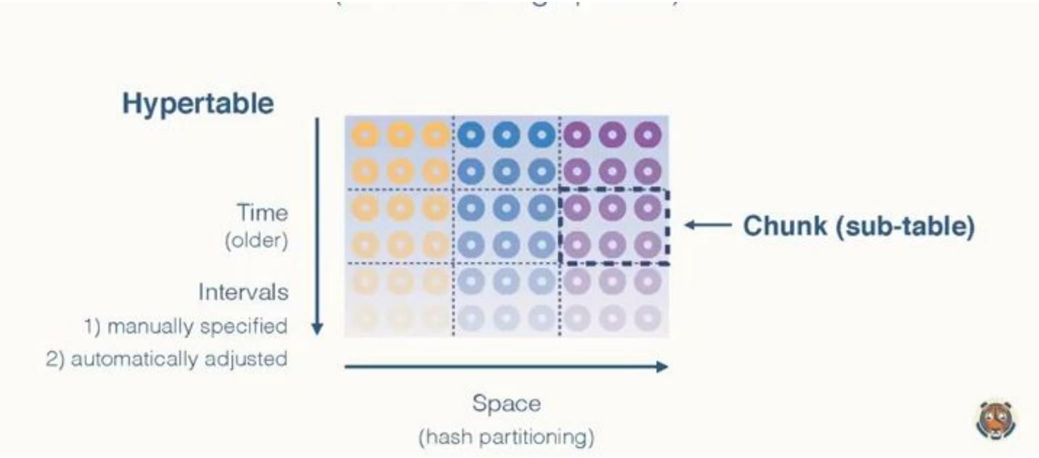


Shard by ID (Citus) + Partition by time (pg_partman)

Scaling Postgres for Time Series Data with Citus | Nov 15 2018 | Marco Slot | Claire Giordano

TimescaleDB

- Addresses many of the limitations of NoSQL databases.
- Full PostgreSQL and SQL features.
- Good Abstraction of underlying complexity and exposes table for application.
- High Insert performance
- Hypertable
- Right-size chunks
- Transparent disk addition
- Intelligent push down
- Custom UDFs

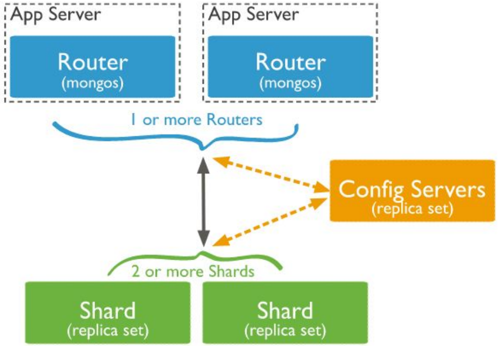


Externally Sharded data

External Data using FDWs

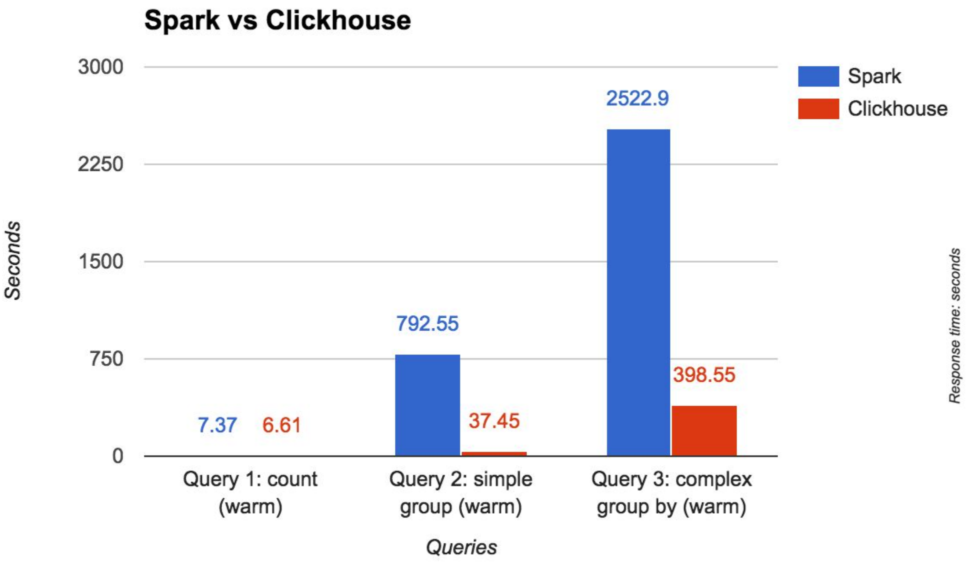
MongoDB and Mongo_fdw

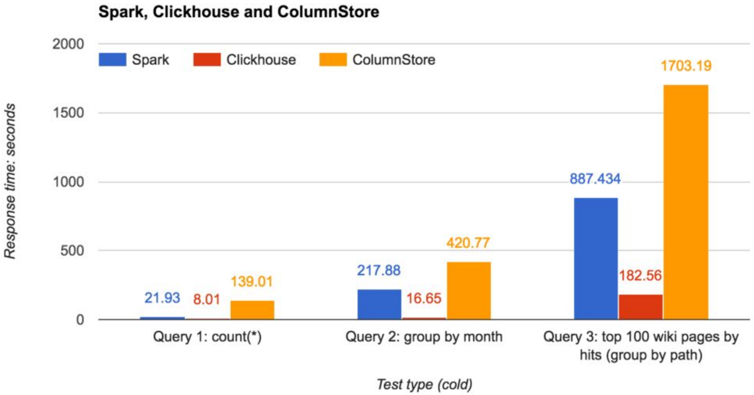
- NoSQL - Doc store
- Growing
- Designed for sharding



- Collections as Tables
- Full Capable SQL
- MongoDB sharded cluster as distributed “Storage engine”

Clickhouse db





https://clickhouse.yandex/

Clickhouse db

- Column Store
- Linearly scalable
- High compression
- SIMD instruction
- Distributed engine

clickhousedb_fdw

References

NVMe Performance : https://www.youtube.com/watch?v=ada_JMsQ3Gk&feature=youtu.be
Table Inheritance : http://evol-monkey.blogspot.com/2018/03/implementing-distributed-reporting.html
Built in sharding : https://www.pgconf.asia/JA/2017/wp-content/uploads/sites/2/2017/12/D2-B1.pdf
Storage Networking Industry Association : https://www.snia.org/
Intel Optane DC : https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html

Thank You

Join Us at Percona Live

Percona Live 2019 takes place in Austin, Texas from May 28-30, 2019 at the Hyatt Regency.

Percona Live provides an opportunity to network with peers and technology professionals. Mingle with all types of database community members: DBAs, developers, C-level executives and the latest database technology trend-setters.

Dedicated PostgreSQL Tutorials and Breakout Sessions in the PostgreSQL Track!

ADVANCED RATE TICKETS ON SALE!

https://www.percona.com/live/19/

PERCONA LIVE

Connect. Accelerate. Innovate.