Advanced Data Management (CSCI 640/490)

Scalable Dataframes

Dr. David Koop

Recent History in Databases

- Early 2000s: Commercial DBs dominated, Open-source DBs missing features
- Mid 2000s: MySQL adopted by web companies
- Late 2000s: NoSQL dos scale horizontally out of the box
- Early 2010s: New DBMSs that can scale across multiple machines natively and provide ACID guarantees

[A. Pavlo]

NewSQL Definitions

- Stonebraker's Definition:
  - SQL as the primary interface
  - ACID support for transactions
  - Non-locking concurrency control
  - High per-node performance
  - Parallel, **shared-nothing** architecture (what about shared-disk?)
- Wikipedia (Pavlo): A class of modern relational DBMSs that provide the same scalable performance of NoSQL systems for OLTP workloads while still maintaining the ACID guarantees of a traditional DBMS.

[A. Pavlo]

NewSQL Positioning

[A. Pavlo]

Three Types of NewSQL Systems

- New Architectures
  - New codebase without architectural baggage of legacy systems
  - Examples: VoltDB, Spanner, Clustrix
- Transparent Sharding Middleware:
  - Transparent data sharding & query redirecting over cluster of single-node DBMSs
  - Examples: citusdata, ScaleArc (usually support MySQL/postgres wire)
- Database-as-a-Service:
  - Distributed architecture designed specifically for cloud-native deployment
  - Examples: xeround, GenieDB, FathomDB (usually based on MySQL)

[A. Pavlo]

What went wrong?

- Almost every NewSQL company from the last decade has closed, sold for scraps, or pivoted to other markets
- Why?
  - Selling an OLTP Database System is hard
  - Startup cost of a relational system is harder than NoSQL
  - Existing DBMS Systems (MySQL, postgresql) are Good
  - Cloud Disruption
    - Can't sell on-premises
    - Can't complete on cost with cloud vendors
  - Lack of Open Source

[A. Pavlo]

Conclusions

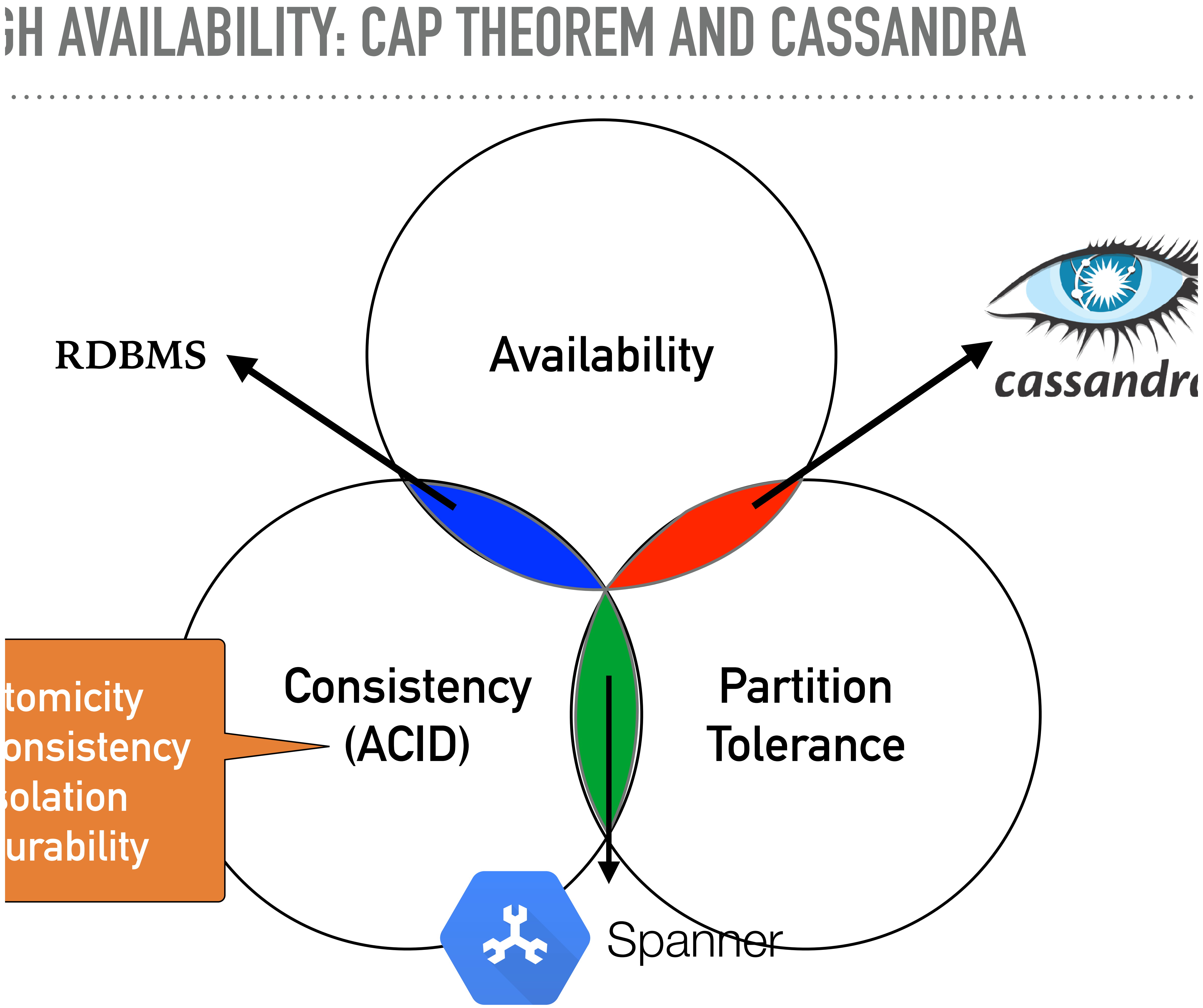
- NewSQL is dead
- Academic: the NewSQL movement was a success
- Business: a failure for those who embraced the NewSQL mantle
- Next?
  - You still need humans to design, configure, and optimize logical/physical aspects of a database
  - Humans are expensive
  - Automation is the future.

[A. Pavlo]

Spanner Overview

- Focus on scaling databases focused on OLTP (not OLAP)
- Since OLTP, focus is on sharding **rows**
- Tries to satisfy CAP (which is impossible per CAP Theorem) by not worrying about 100% availability
- External consistency using multi-version concurrency control through timestamps
- ACID is important
- Structured: universe with zones with zone masters and then spans with span masters
- SQL-like (updates allow SQL to be used with Spanner)

Spanner and the CAP Theorem



- Which type of system is Spanner?
  - C: consistency, which implies a single value for shared data
  - A: 100% availability, for both reads and updates
  - P: tolerance to network partitions
- Which two?
  - CA: close, but not totally available
  - So actually **CP**

External Consistency

- Traditional DB solution: **two-phase locking**—no writes while client reads
- "The system behaves as if all transactions were executed sequentially, even though Spanner actually runs them across multiple servers (and possibly in multiple datacenters) for higher performance and availability" [Google]
- Semantically indistinguishable from a single-machine database
- Uses multi-version concurrency control (MVCC) using **timestamps**
- Spanner uses **TrueTime** to generate monotonically increasing timestamps across all nodes of the system

Google Cloud Spanner

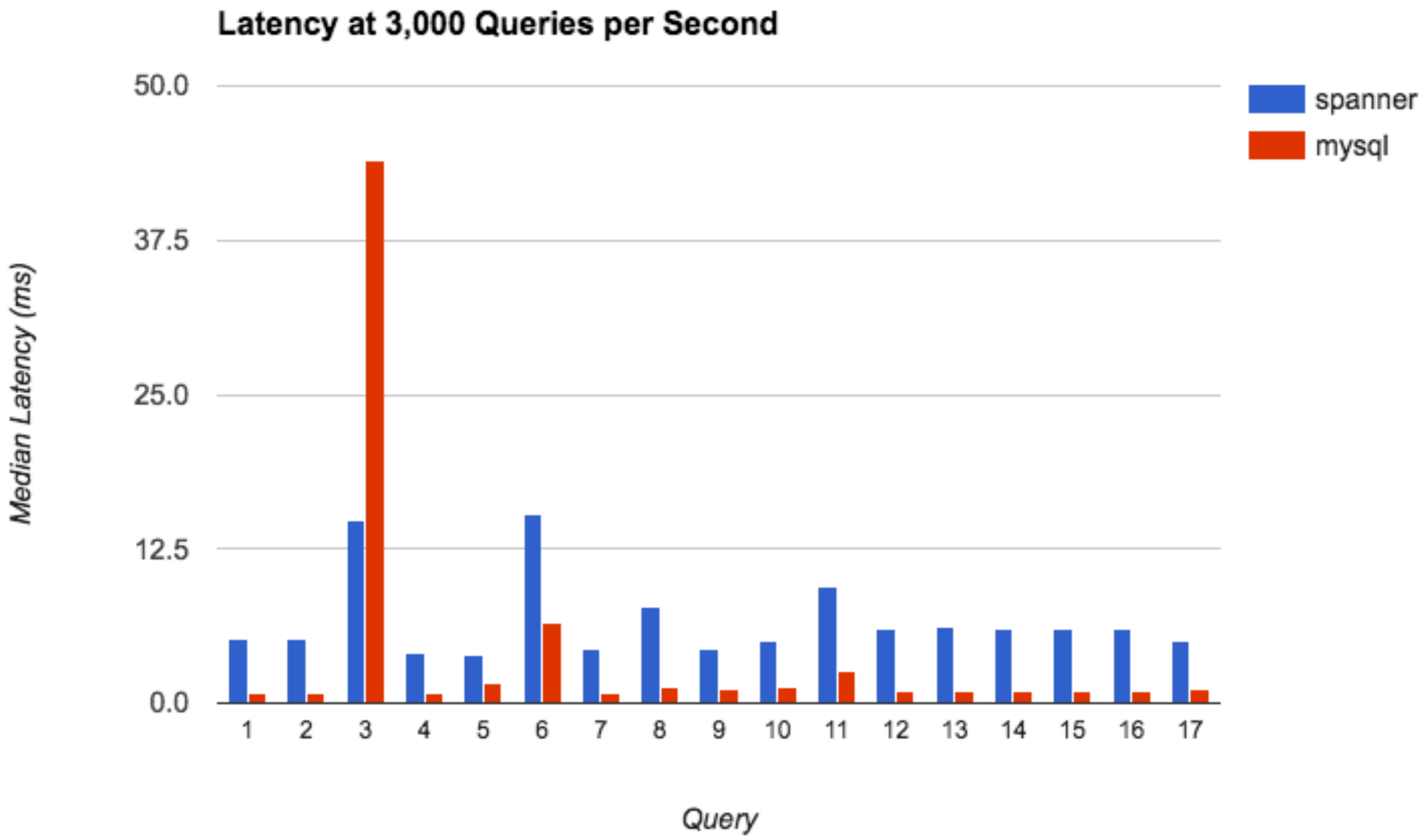
- https://cloud.google.com/spanner/
- Features:
  - Global Scale: thousands of nodes across regions / data centers
  - Fully Managed: replication and maintenance are automatic
  - Transactional Consistency: global transaction consistency
  - Relational Support: Schemas, ACID Transactions, SQL Queries
  - Security
  - Highly Available

More Recent Tests: Spanner vs. MySQL

	Frequency	Query
1	0.30%	INSERT INTO `terms` (`term`, `rank`, `set_id`, `last_modified`) VALUES (?,?,?,?),(?,?,?,?)
2	0.25%	INSERT INTO `terms` (`term`, `rank`, `set_id`, `last_modified`, `definition`) VALUES (?,?,?,?,?),(?,?,?,?,?),(?,?,?,?,?),...
3	4.22%	INSERT INTO `terms` (`term`,`rank`,`set_id`,`last_modified`) VALUES (?,?,?,?)
4	1.88%	INSERT INTO `terms` (`term`,`rank`,`set_id`,`last_modified`,`definition`) VALUES (?,?,?,?,?)
5	3.28%	SELECT * FROM `terms` WHERE (`is_deleted` = 0) AND (`set_id` IN (??)) AND (`rank` IN (0,1,2,3)) AND (`term` != '')
6	14.13%	SELECT `set_id`, COUNT(*) FROM `terms` WHERE (`is_deleted` = 0) AND (`set_id` = ?) GROUP BY `set_id`
7	12.56%	SELECT * FROM `terms` WHERE (`id` = ?)
8	0.49%	SELECT * FROM `terms` WHERE (`id` IN (??) AND `set_id` IN (??))
9	4.11%	SELECT `id`, `set_id` FROM `terms` WHERE (`set_id` = ?) LIMIT 20000
10	0.43%	SELECT `id`, `set_id` FROM `terms` WHERE (`set_id` IN (??)) LIMIT 20000
11	0.59%	SELECT * FROM `terms` WHERE (`id` IN (??))
12	36.76%	SELECT * FROM `terms` WHERE (`set_id` = ?)
13	0.61%	SELECT * FROM `terms` WHERE (`set_id` IN (??))
14	6.10%	UPDATE `terms` SET `definition`=?, `last_modified`=? WHERE `id`=? AND `set_id`=?
15	0.33%	UPDATE `terms` SET `is_deleted`=?, `last_modified`=? WHERE `id` IN (??) AND `set_id`=??
16	12.56%	UPDATE `terms` SET `rank`=?, `last_modified`=? WHERE `id`=? AND `set_id`=?
17	1.06%	UPDATE `terms` SET `word`=?, `last_modified`=? WHERE `id`=? AND `set_id`=?
18	0.32%	UPDATE `terms` SET `definition`=?, `word`=?, `last_modified`=? WHERE `id`=? AND `set_id`=?

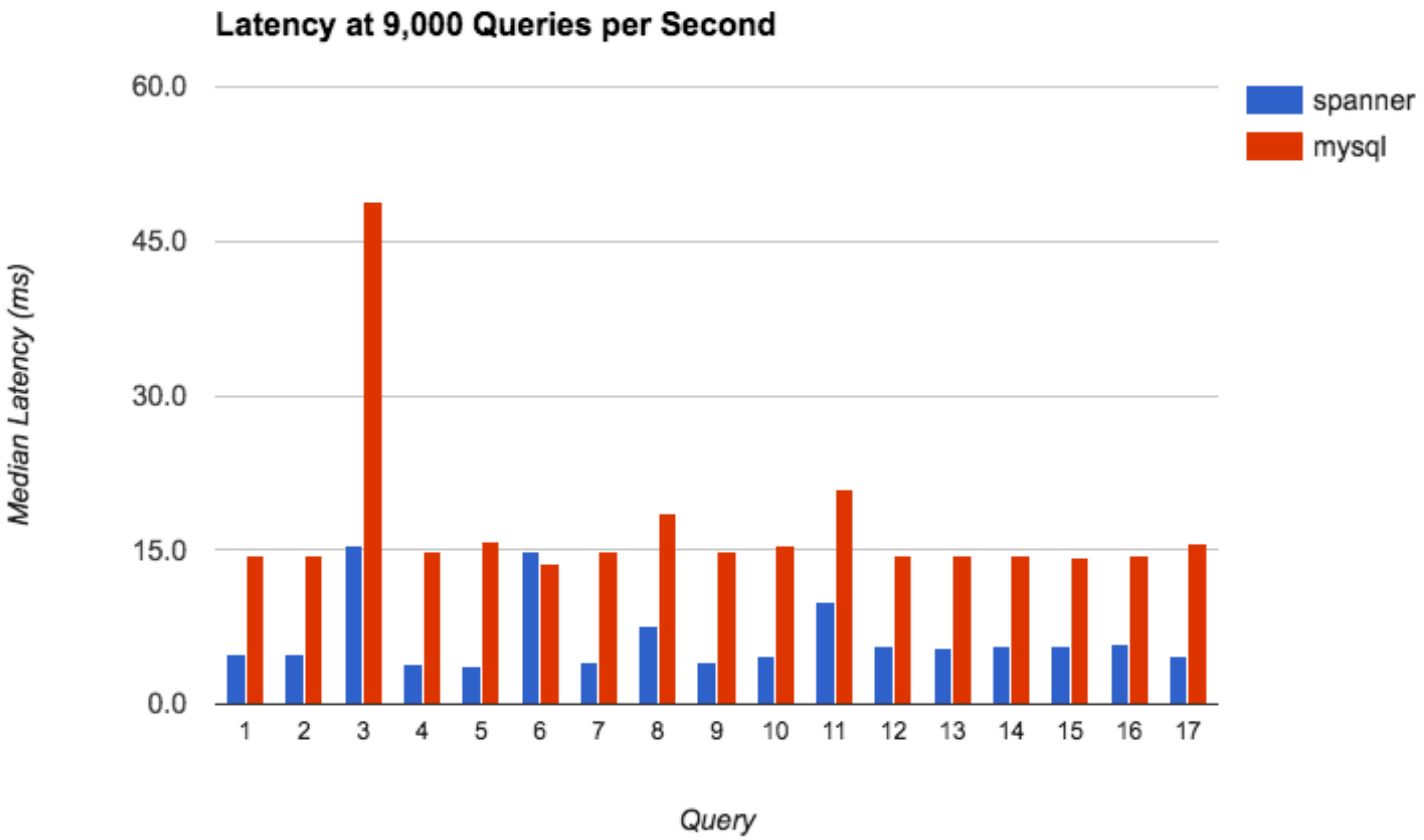
[P. Bakkum and D. Cepeda, 2017]

Latency: Spanner vs. MySQL



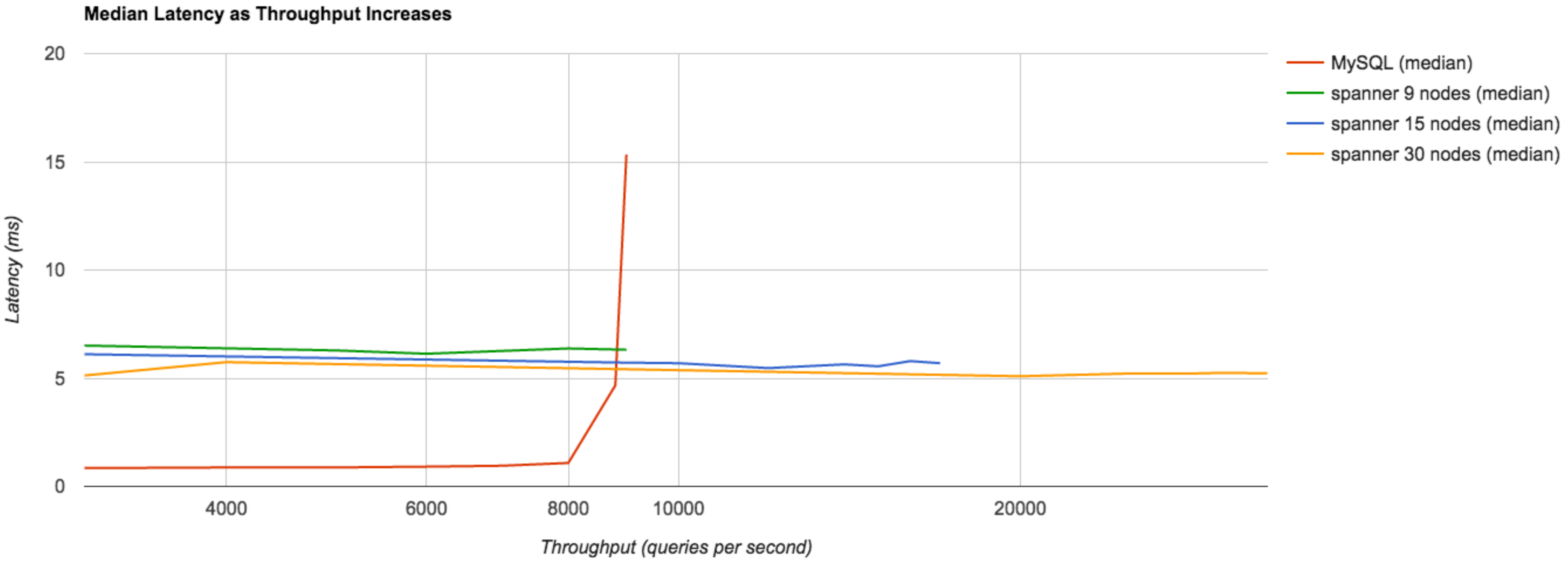
[P. Bakkum and D. Cepeda, 2017]

Latency: Spanner vs. MySQL



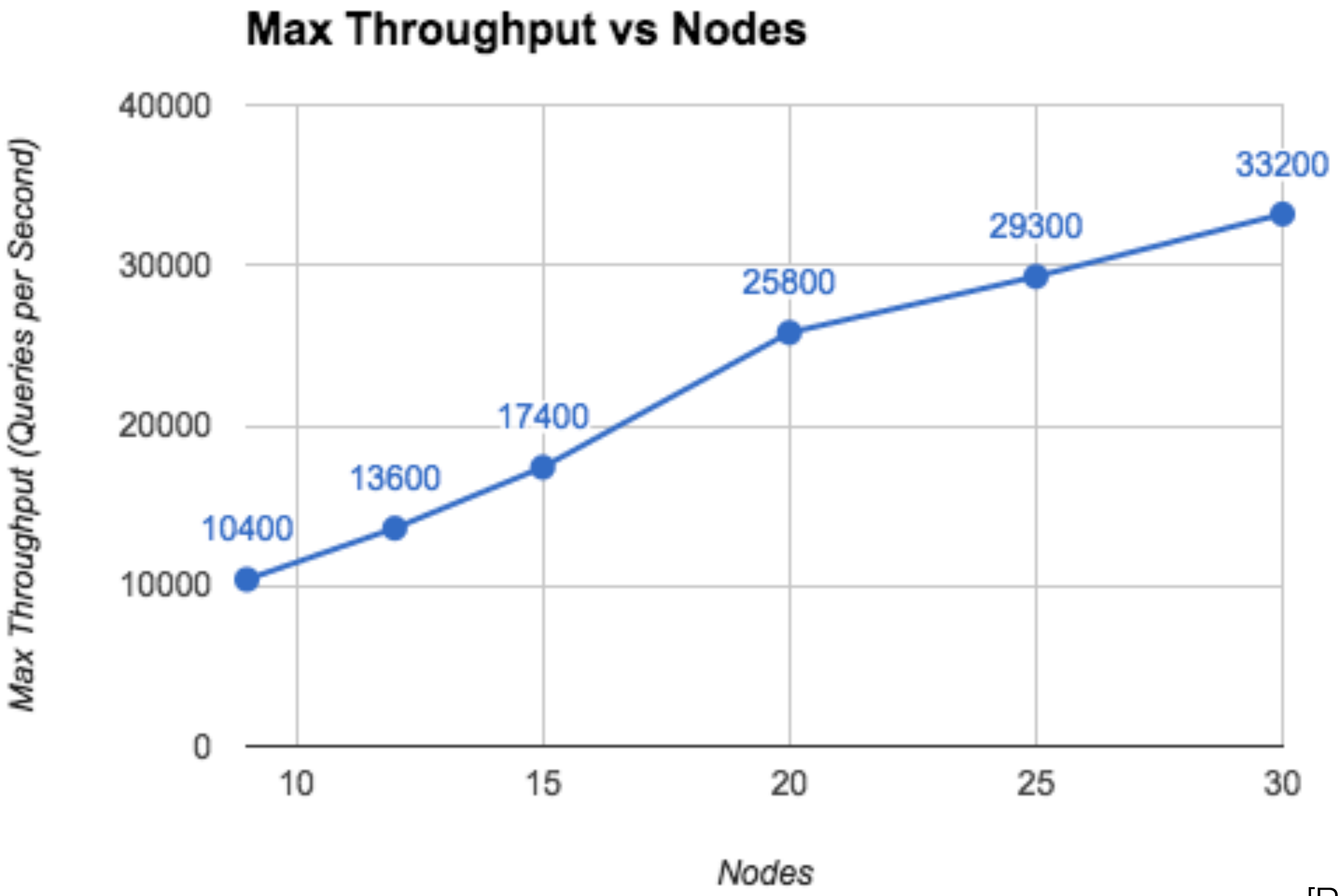
[P. Bakkum and D. Cepeda, 2017]

Throughput: Spanner vs. MySQL



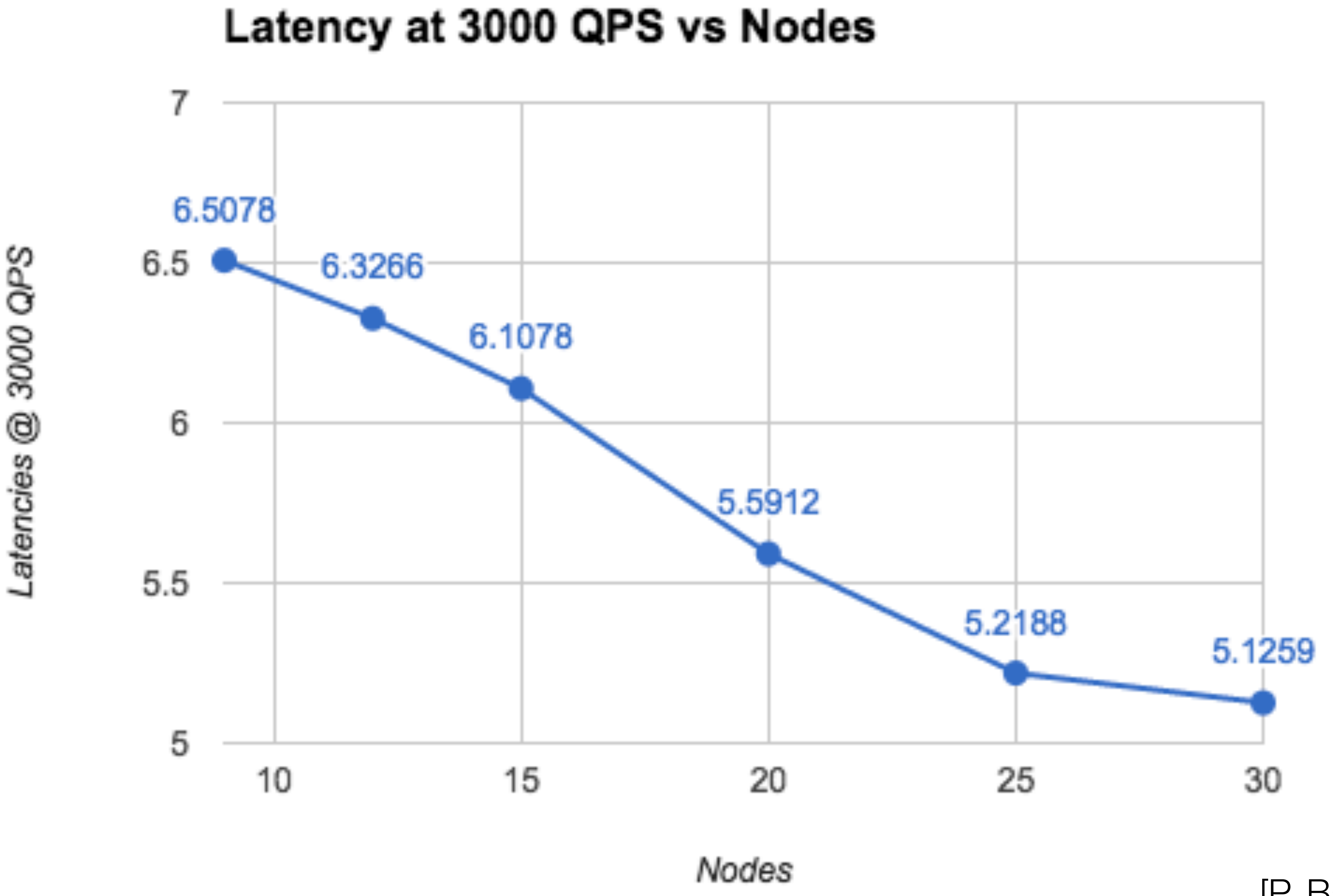
[P. Bakkum and D. Cepeda, 2017]

Max Throughput vs. Nodes



[P. Bakkum and D. Cepeda, 2017]

Spanner: Latency vs. Nodes



[P. Bakkum and D. Cepeda, 2017]

Assignment 4

- Work on Data Integration and Data Fusion
- Integrate artist datasets from different institutions (Met, NGA, AIC, CMA)
  - Integrate information based on ids and matching
- Record Matching:
  - Which artists are the same?
- Data Fusion:
  - Names
  - Dates
  - Nationalities

Scalable Dataframes

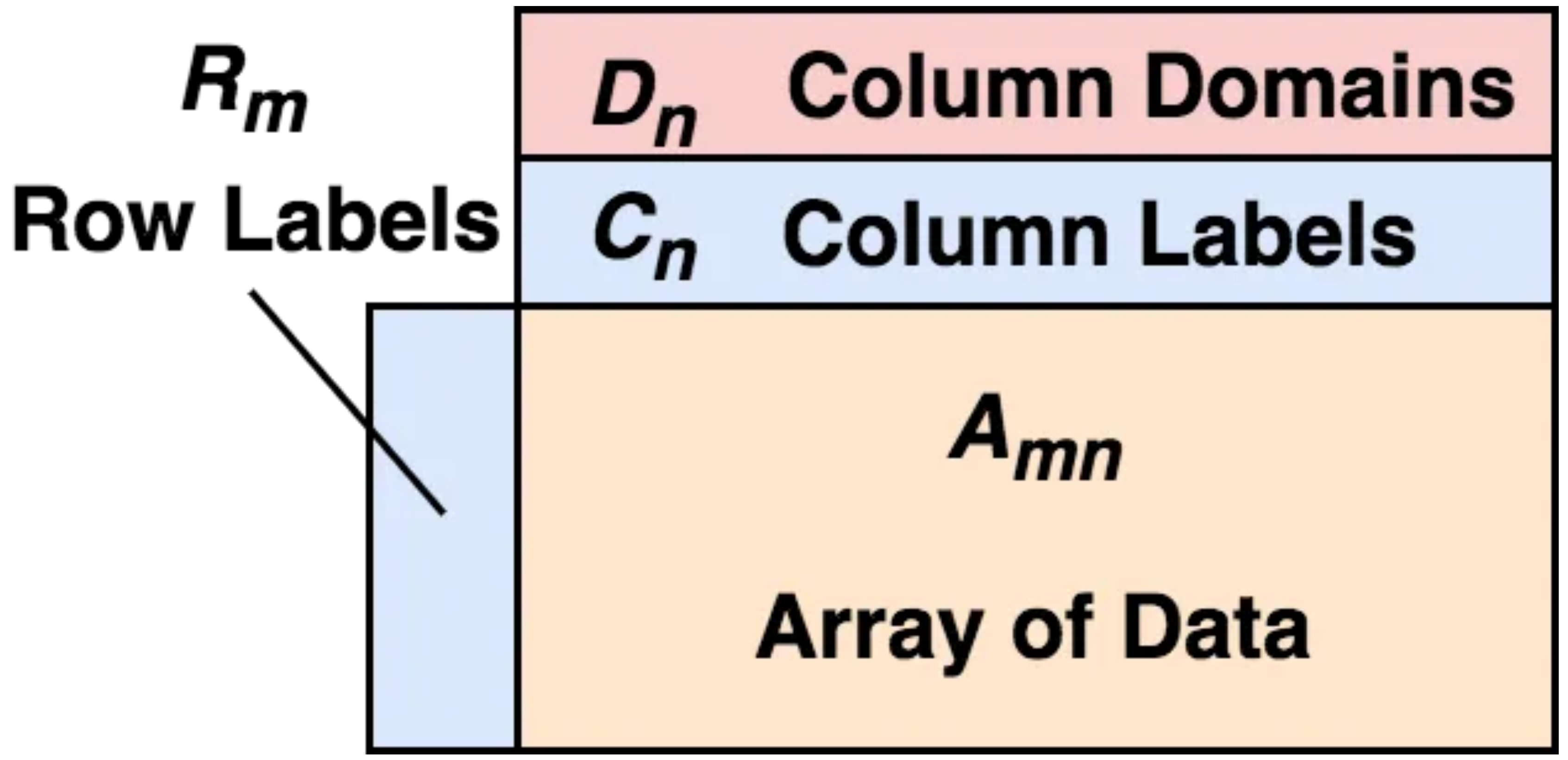
History of Dataframes

- Originally in *Statistical Models in S*, [J. M. Chambers & T. J. Hastie, 1992]
- R, open-source alternative to S, developed in 2000 (with dataframes)
- Pandas, 2009
- Spark, 2010 (resilient distributed dataset [RDD], Dataset API)

[D. Petersohn, 2022]

Formalizing Dataframes

- Combines parts of matrices, databases, and spreadsheets
- Ordered rows (unlike databases)
- Types can be inferred at runtime, not the same across all columns
- Lots of "intuitive" functions (600+)



[D. Petersohn, 2022]

Differences between Databases & Dataframes

Convenience	Entire query at once	Incremental + inspection
Flexible	Strict schema	Mixed types, R/C and data/metadata equiv.
Versatility	SFW or bust	600+ functions

[D. Petersohn, 2022]

Scaling Dataframes

- Solutions:
  - Spark
  - Dask
  - Polars
  - Vaex
  - Modin

Scaling up your pandas workflows with Modin

D. Petersohn

Ibis Overview

Blazing fast dataframes in Python with Polars

J. L. C. Rodríguez