Dodging the *cost of scalability* in data analysis with *CPU efficiency*

Budapest Data Forum 2020
Dr. Hannes Mühleisen

About me

- Senior Researcher at Centrum Wiskunde & Informatica in Amsterdam
    - Database Architectures Group
- Teaching Big Data™ at University of Amsterdam
- Main Interest: Data Management for Data Science
- `@hfmuehleisen`
  `http/hannes.muehleisen.org`

Scalability! But at what COST?

Frank McSherry Michael Isard Derek G. Murray
Unaffiliated Unaffiliated* Unaffiliated†

Abstract

We offer a new metric for big data platforms, COST, or the Configuration that Outperforms a Single Thread. The COST of a given platform for a given problem is the hardware configuration required before the platform outperforms a competent single-threaded implementation. COST weighs a system's scalability against the overheads introduced by the system, and indicates the actual performance gains of the system, without rewarding systems that bring substantial but parallelizable overheads.

We survey measurements of data-parallel systems recently reported in SOSP and OSDI, and find that many systems have either a surprisingly large COST, often hundreds of cores, or simply underperform one thread for all of their reported configurations.

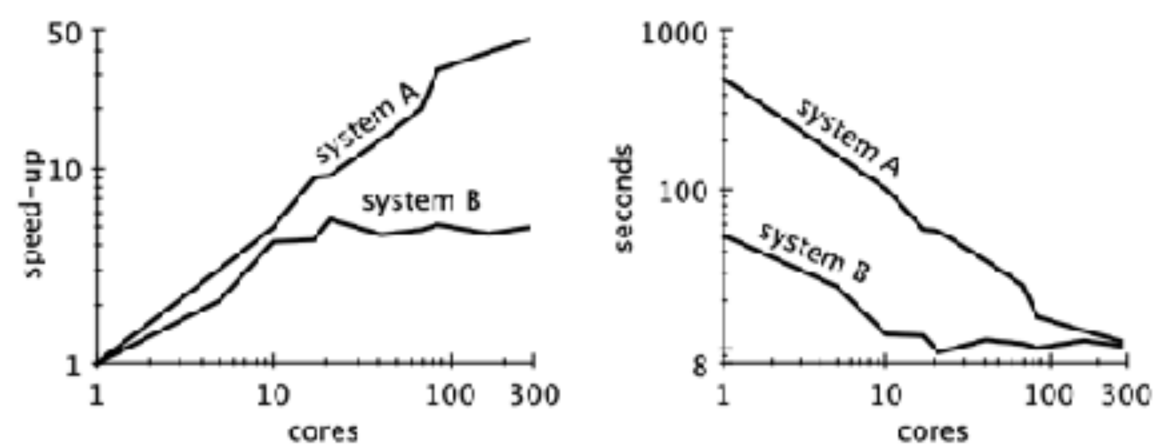


Figure 1: Scaling and performance measurements for a data-parallel algorithm, before (system A) and after (system B) a simple performance optimization. The unoptimized implementation "scales" far better, despite (or rather, because of) its poor performance.

While this may appear to be a contrived example, we will argue that many published big data systems more closely resemble system A than they resemble system B.

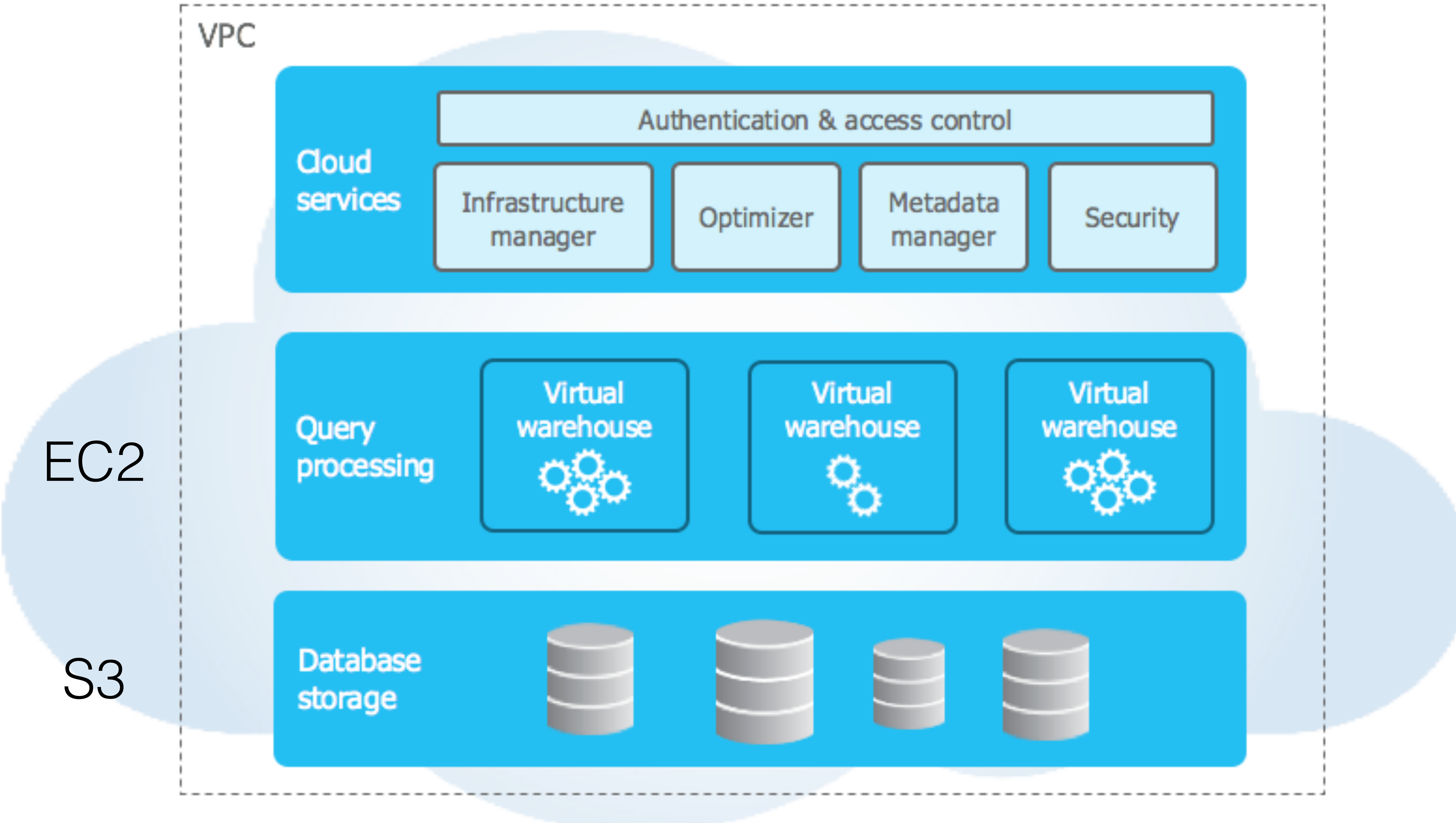
“Distributed Computing”

- Communication Overhead
- Coordination Overhead
- Intermediate Reshuffling Overhead
- Sensitivity to Group Cardinality Skew
- Complex failure modes
- Horrible debugging

“Disaggregated Storage”

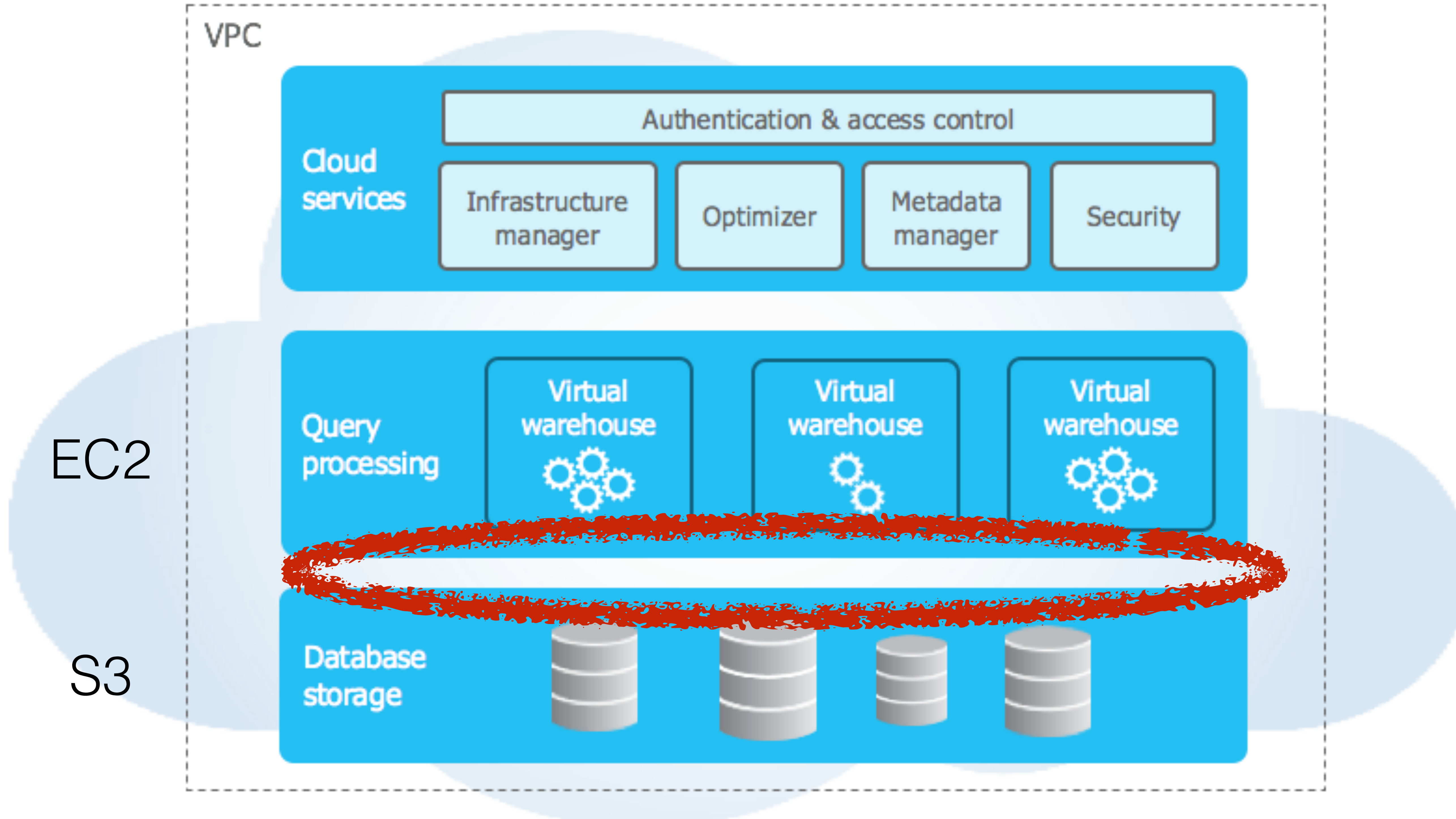
- Old & busted: Co-locate compuation and storage
    - Hadoop
- New & shiny: Separate storage and computation
    - Snowflake, Spark + EMR + S3, …
- Problem: Single-thread data read from S3 is slooow
    - ~ 20 MB/s
- Solution: Many threads, many VMS, many **$$$**

Snowflake Architecture



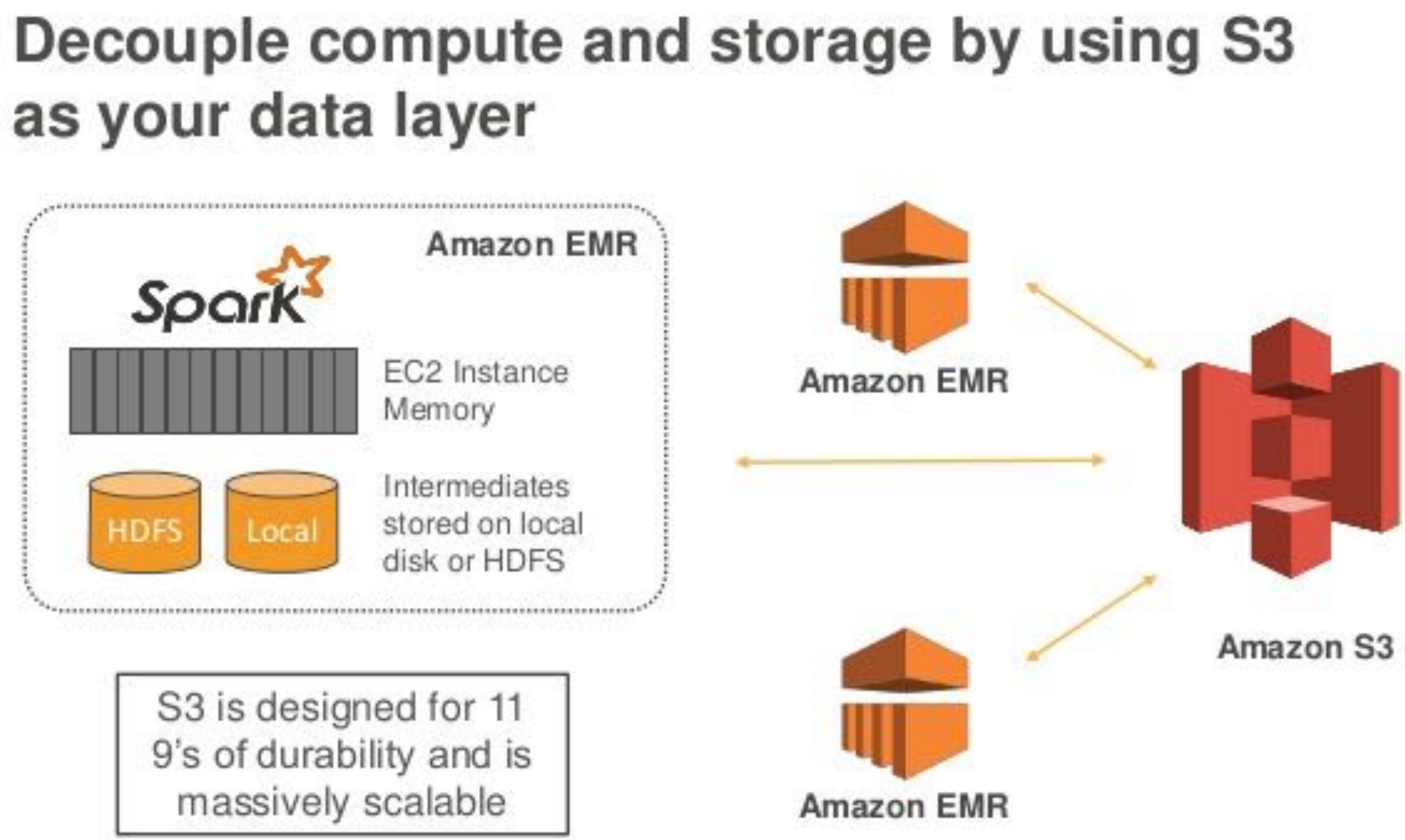
[Snowflake]

Snowflake Architecture



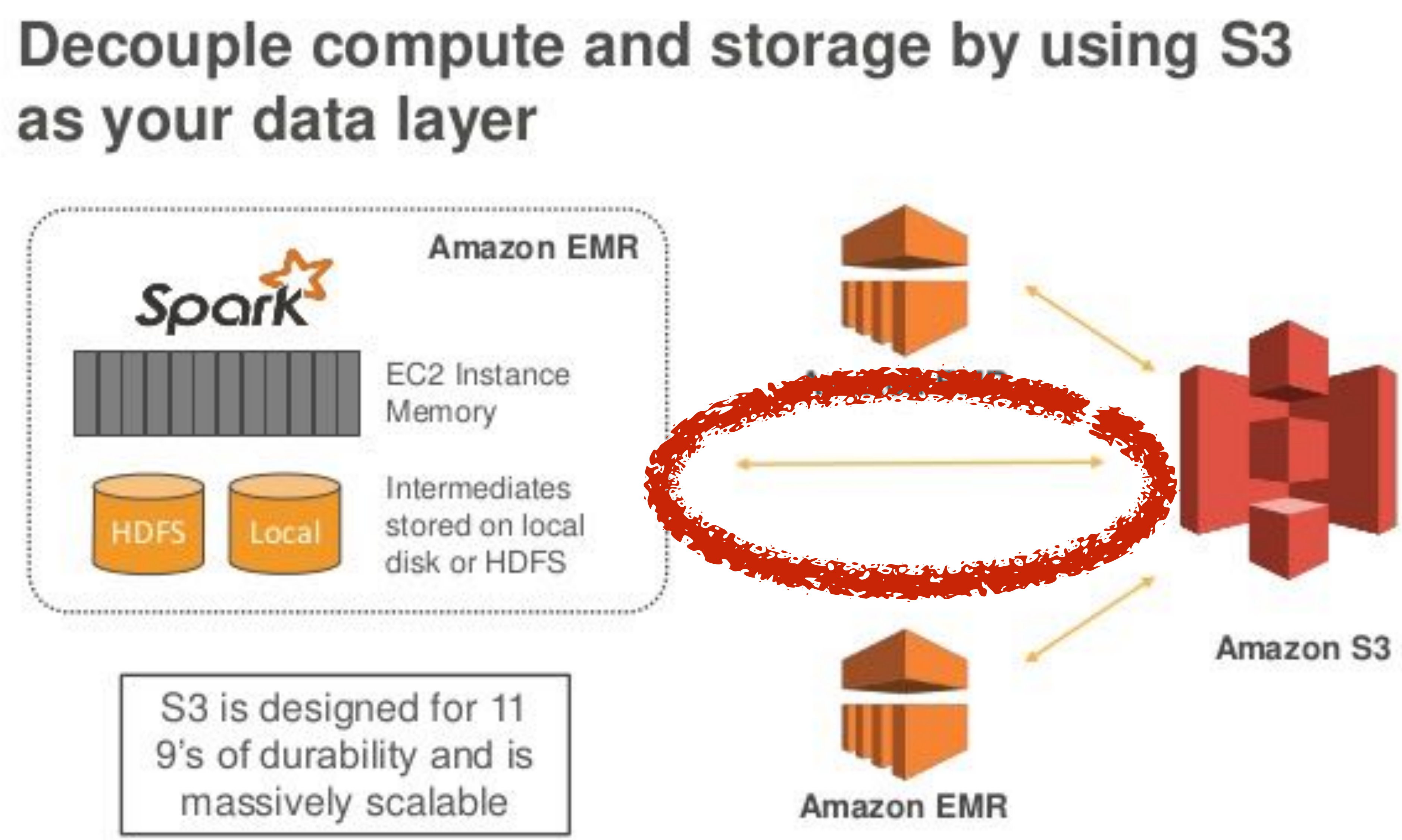
[Snowflake]

Spark on AWS Architecture



[AWS]

Spark on AWS Architecture



[AWS]

“Disaggregated Storage”

- Old & busted: Co-locate compuation and storage
  - Hadoop
- New & shiny: Separate storage and computation
  - Snowflake, Spark + EMR + S3, …
- Problem: Single-thread data read from S3 is slooow
  - ~ 20 MB/s
- Solution: Many threads, many VMS, many **$$$**

Back to Single Node

- Can have very fast IO with NVMe
- 8-core CPUs commonplace
- 64 GB RAM available in MacBooks
- Need Software! ~~Postgres~~? ~~MySQL~~? ~~Pandas~~? ~~R~~?
  - Too slow!

DuckDB

- **DuckDB**: The SQLite for Analytics
  - Fast vectorized analytical queries
  - In-process runtime, no server management
    - Fast data transfer
  - Single-file storage format
  - Simple installation `pip install duckdb`
  - C++11, Free and Open Source (MIT)

www.duckdb.org

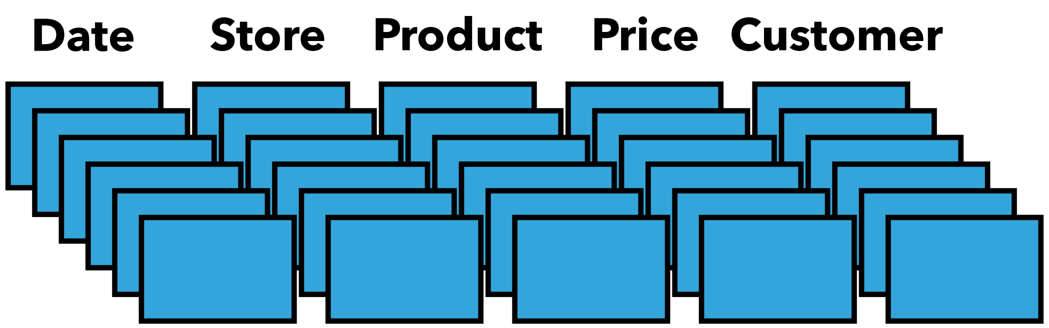
Last Sunday…

Hacker News new | threads | past | comments | ask | show | jobs | submit   hfmuehleisen (20) | logout

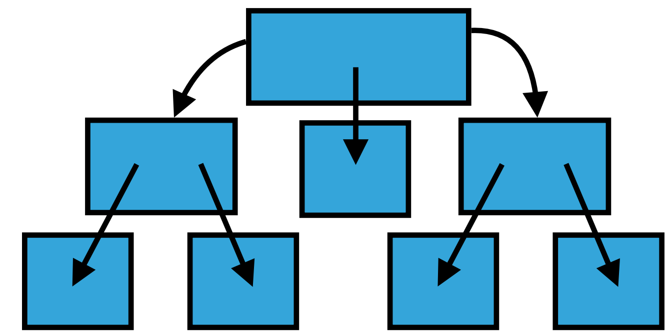
1. DuckDB – An embeddable SQL database like SQLite, but supports Postgres features (duckdb.org)
   335 points by pcr910303 8 hours ago | hide | 62 comments
2. ▲ Raspberry Pi – UASP, Trim, and Boot Performance via USB (jeffgeerling.com)
   98 points by geerlingguy 5 hours ago | hide | 11 comments
3. ▲ Base 65536 (github.com)
   38 points by leoh 3 hours ago | hide | 14 comments
4. ▲ Mouse found atop a 22,000-foot volcano, breaking world record (nationalgeographic.com)
   29 points by greenyoda 3 hours ago | hide | 4 comments
5. ▲ Windows Server vulnerability requires immediate attention (cisa.gov)
   362 points by ohjeez 14 hours ago | hide | 80 comments

DuckDB Internals

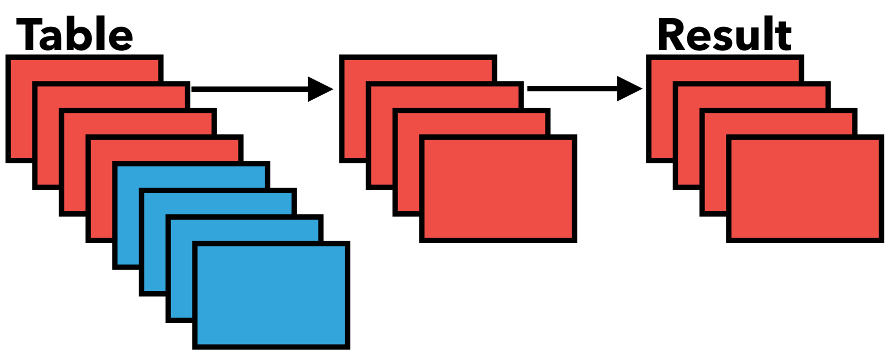
Column-Store



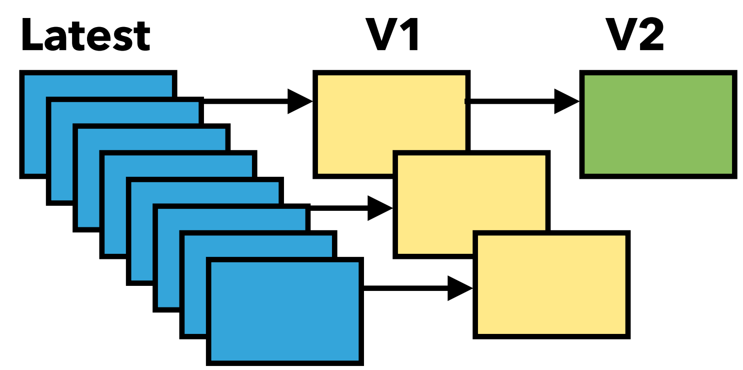
ART Index



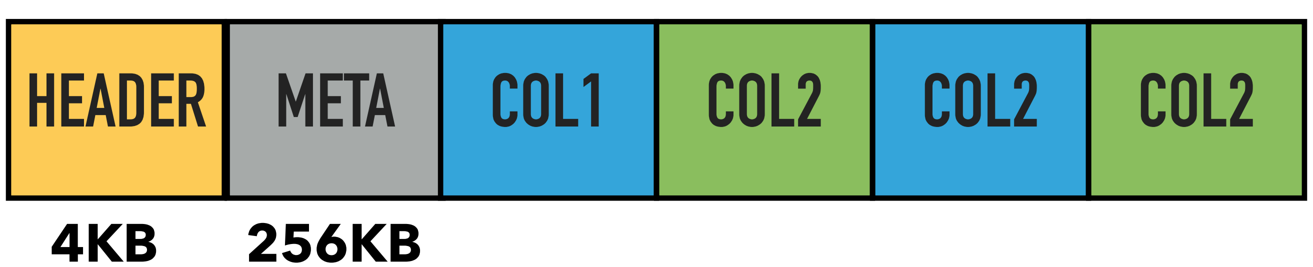
Vectorized Processing



MVCC



Single-File Storage

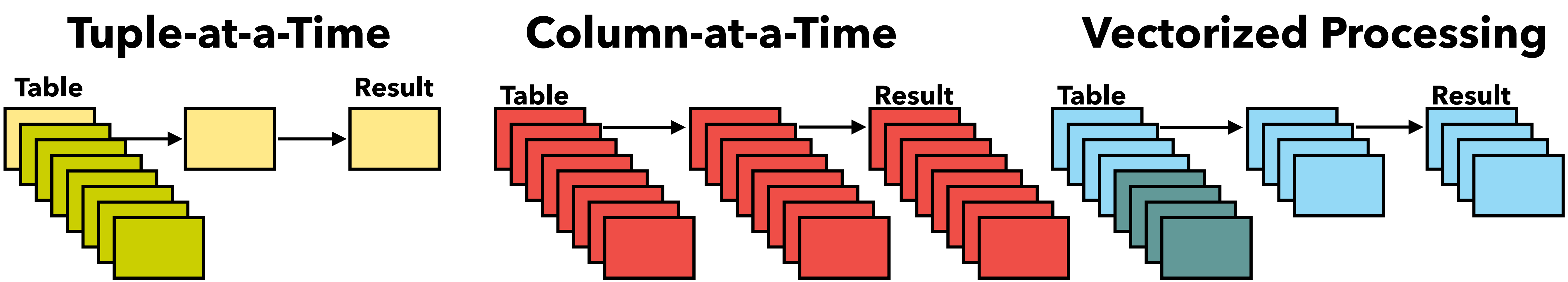


database.db

Parser

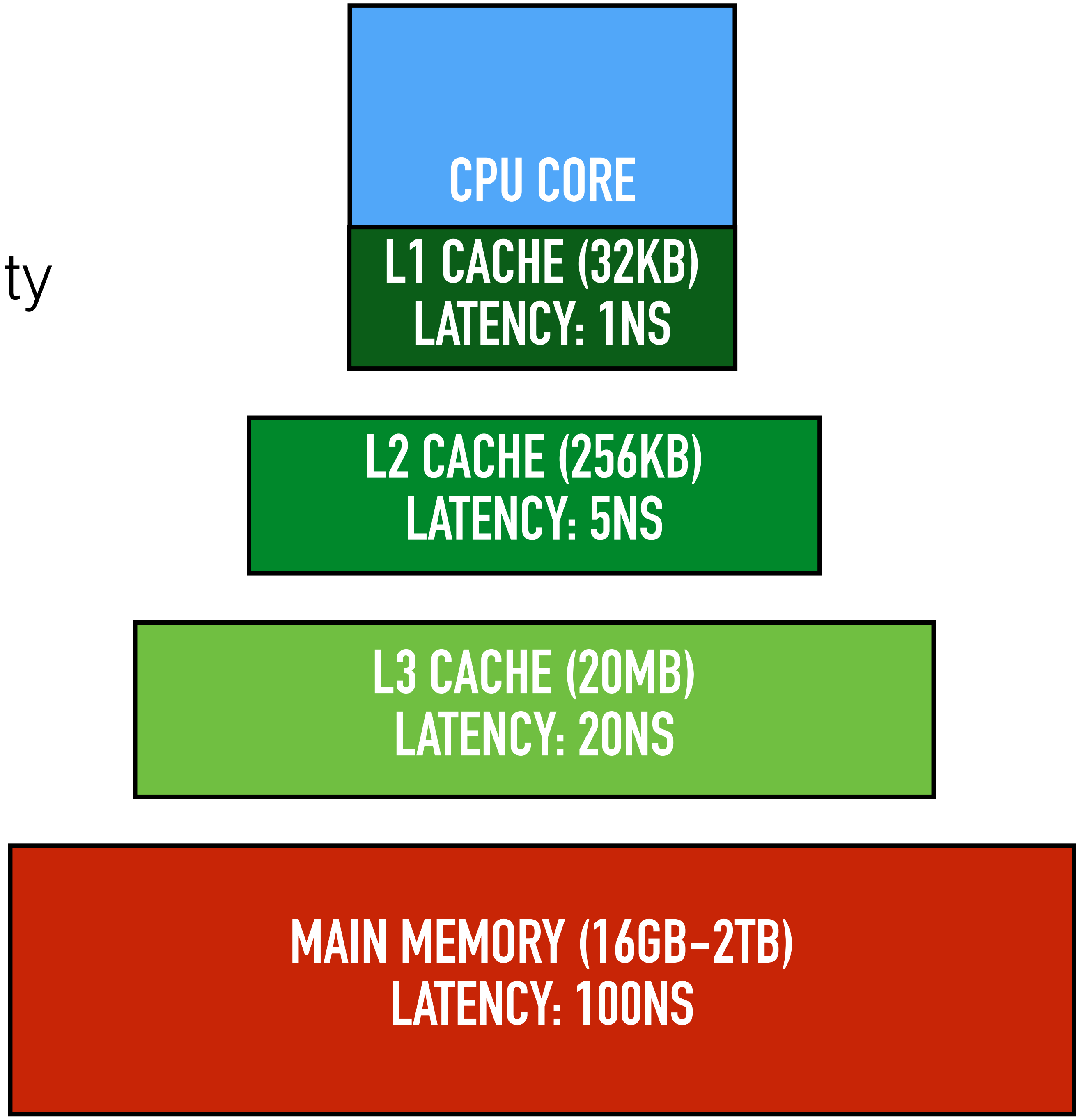
Query Execution Engine

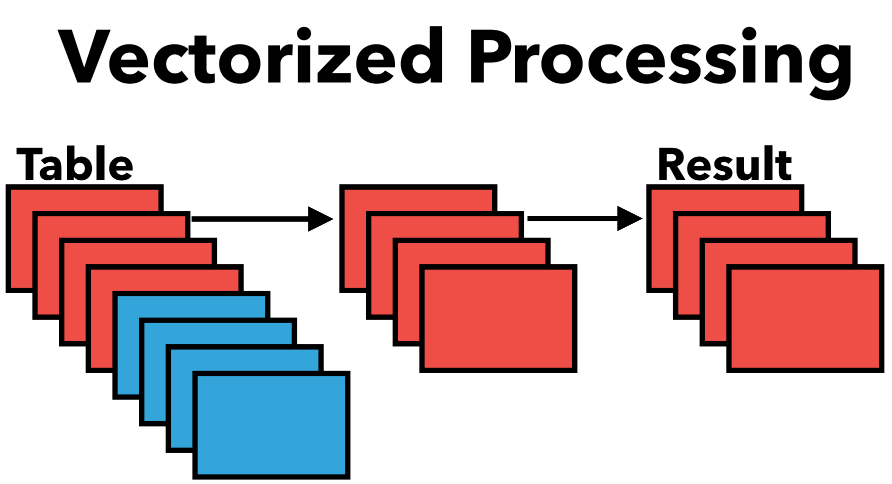
- SQLite/PostgreSQL/MySQL/...: Tuple-At-A-Time
- Pandas/NumPy.R: Column-at-a-time
- DuckDB: **Vectorized** Processing



Query Execution Engine

- DuckDB: Vectorized Processing
  - Optimized for CPU Cache locality
  - SIMD instructions, Pipelining
  - Small intermediates
    (ideally fit in L1 cache)





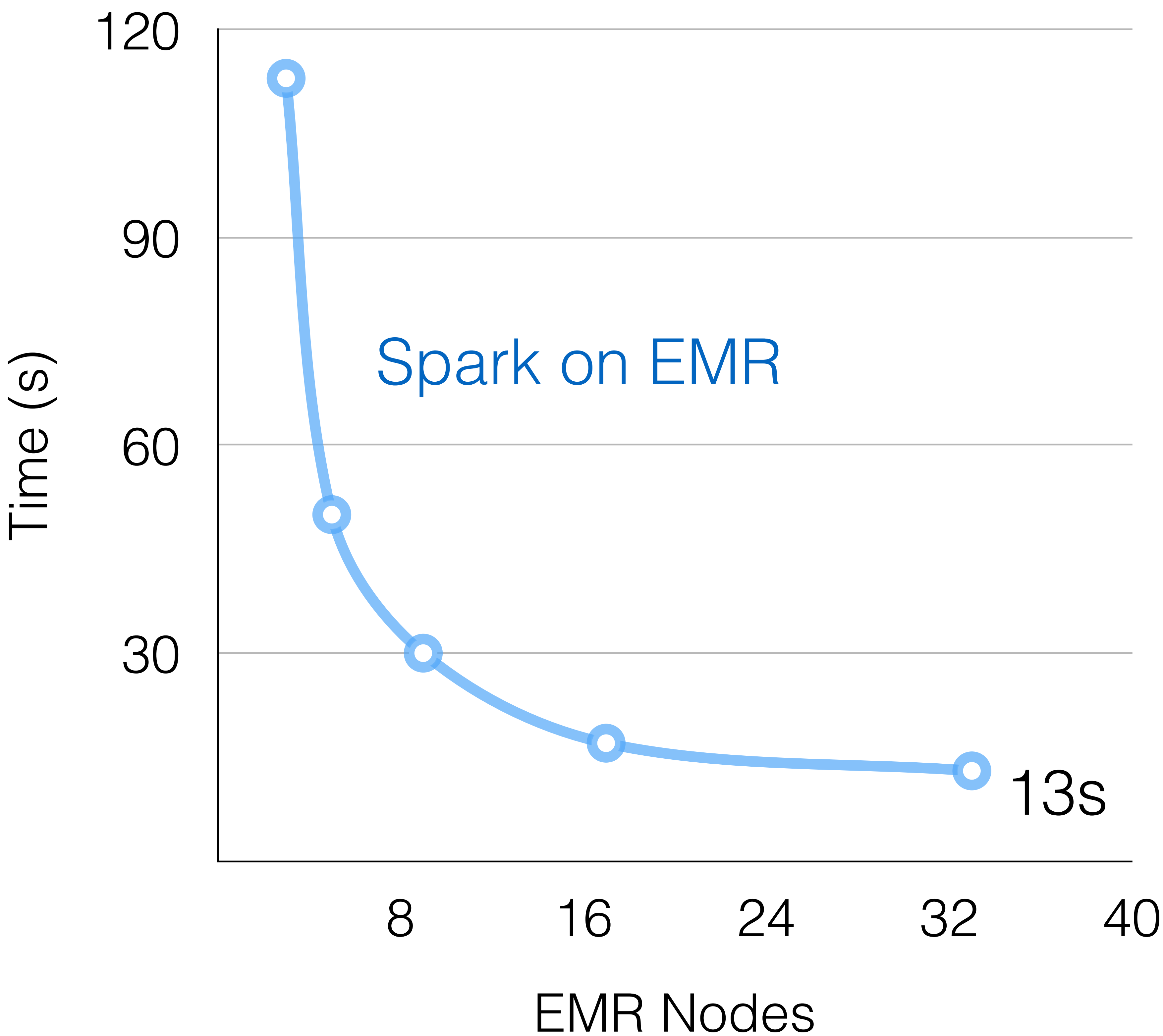
Science Time

- How many Amazon nodes does it take to beat a fairly efficient single-node implementation?
- Hardware: 8-Core Xeon / m5.2xlarge
- Software: Spark vs. DuckDB
- Data: TPC-H SF1000, lineitem table, ~220 GB
  - Converted to Parquet files with Spark
- Query:

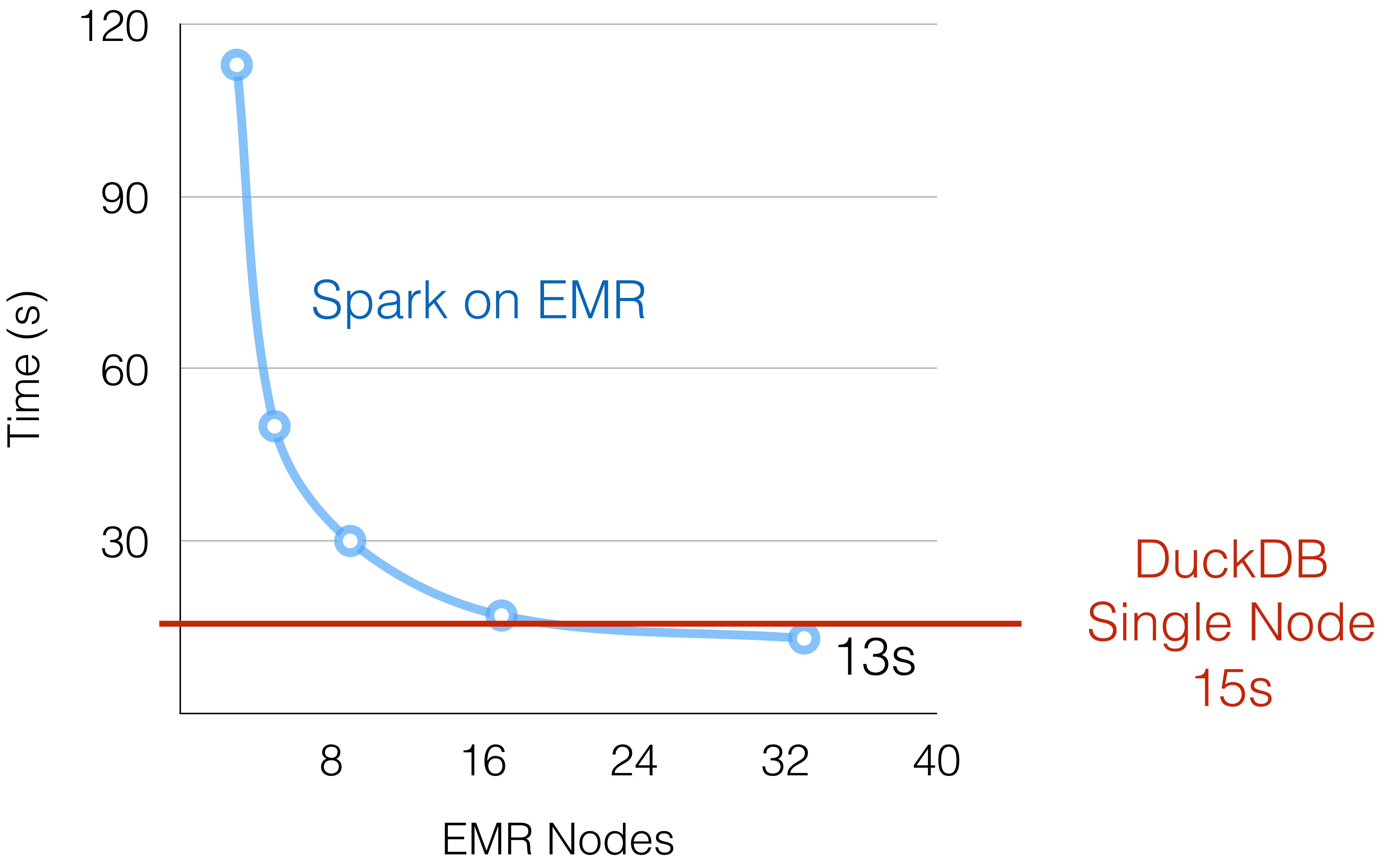
```
SELECT sum(l_extendedprice), sum(l_tax),
sum(l_discount) FROM lineitem
```

Best case for Spark!

Experiment Results



Experiment Results



Experiment Results

- It took **33** nodes for Spark to beat DuckDB on a single node
  - Mostly due to disaggregated storage
  - Best case, query trivially parallelizable
  - Think of the CO_2!

Conclusion

- Don’t write off single node just yet
- Efficient tools can stretch single node *far* into Big Data territory
- DuckDB is a novel, CPU-efficient data processing system
    - `www.duckdb.org`

DuckDB

@hfmuehleisen