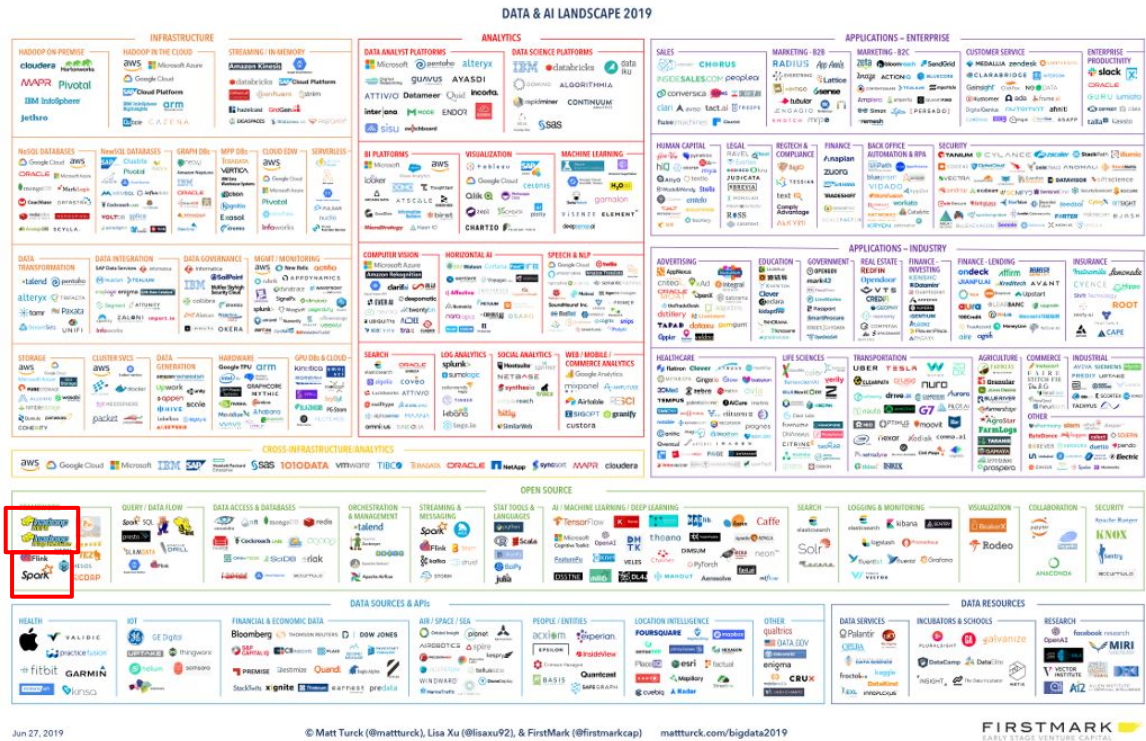
Databases and Big Data

Other Systems

Recap

- What did we cover?

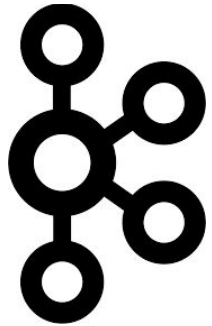


Recap

- Many choices
- But only few big ideas
- We have covered important ones
  - Distributed storage
  - Distributed processing: batch, iterative, interactive, streaming, SQL

Today

- Few more important ones
  - Data lake
  - Streaming systems
  - BigTable
  - Global-scale SQL databases



Database Evolution

- Ideal: one database for all your needs

Inventory

How we like to think of data in the organization

Database Evolution

- Reality:
  - Multiple databases everywhere

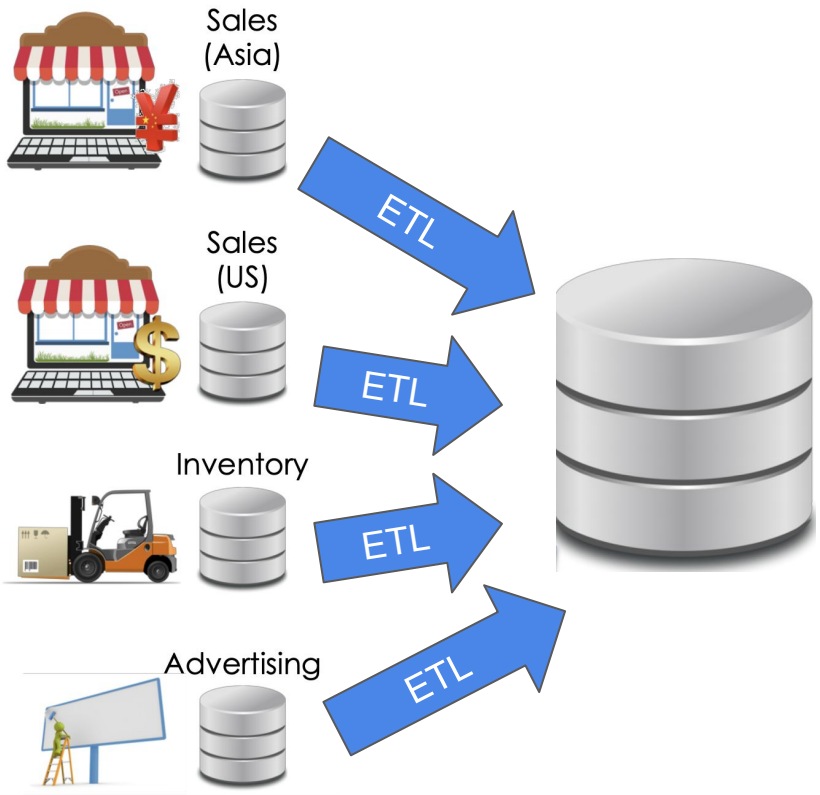
Database Evolution

- Reality
  - Each databases serve different purpose
  - Each is mission critical
  - Each has different format

People want **clean**, **consolidated**, **historical snapshots** of the data

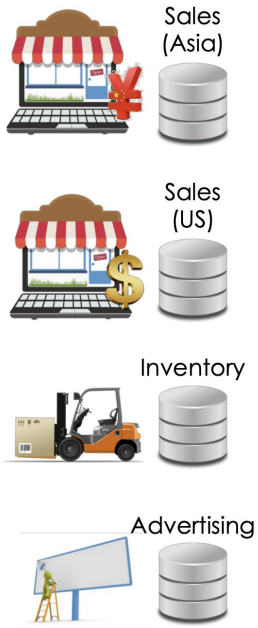
Database Evolution

- **Data warehouse**
  - Consolidated, cleaned, historical data from multiple databases
  - Periodically ETL data in
    - Extract from multiple databases
    - Transform to standard schemas
    - Load to the warehouse (a RDBMS)



Database Evolution

- OLTP:
  - Online Transaction Processing
- OLAP
  - Online Analytical Processing



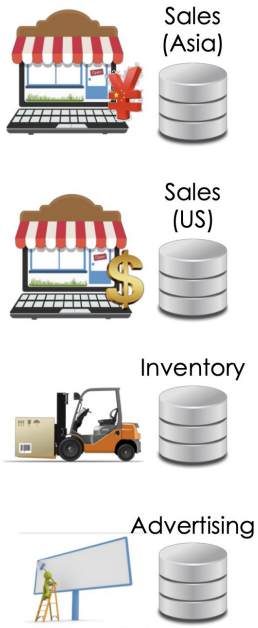
OLTPs

OLAP

Database Evolution

- OLTP:
  - Short-lived read and writes queries
  - Transaction is a must!
- OLAP
  - Long-running queries
- Most OLAP databases are closed-source

OLAP dB are expensive, designed to run queries efficiently

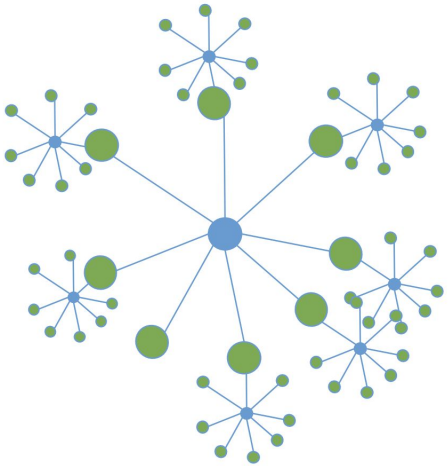


OLTPs

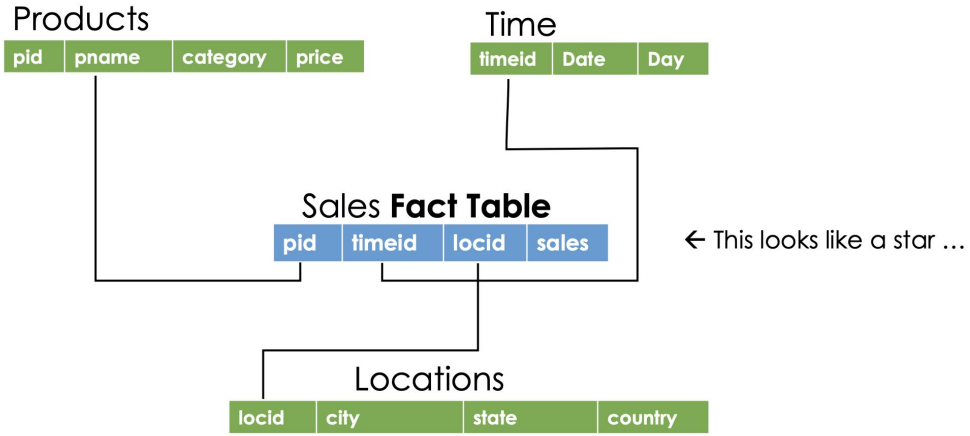
OLAP

OLAP

- Key features
  - Highly normalized tables



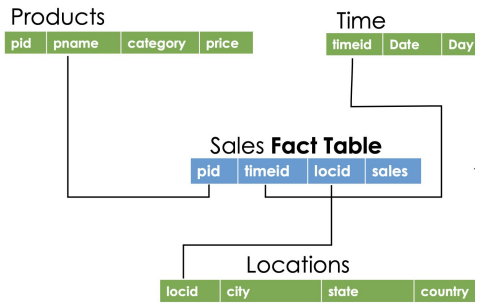
← This looks like a snowflake …



← This looks like a star …

OLAP

- Common schema



Sales **Fact Table**

pid	timeid	locid	sales
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
12	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35
11	2	2	22
11	3	2	10
12	1	2	26

Locations

locid	city	state	country
1	Omaha	Nebraska	USA
2	Seoul		Korea
5	Richmond	Virginia	USA

Products

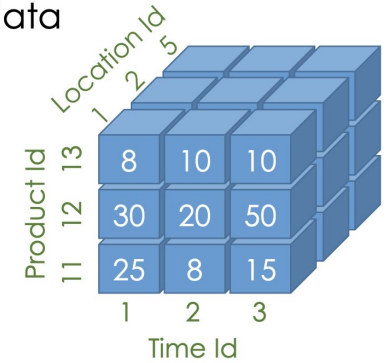
pid	pname	category	price
11	Corn	Food	25
12	Galaxy 1	Phones	18
13	Peanuts	Food	2

Time

timeid	Date	Day
1	3/30/16	Wed.
2	3/31/16	Thu.
3	4/1/16	Fri.

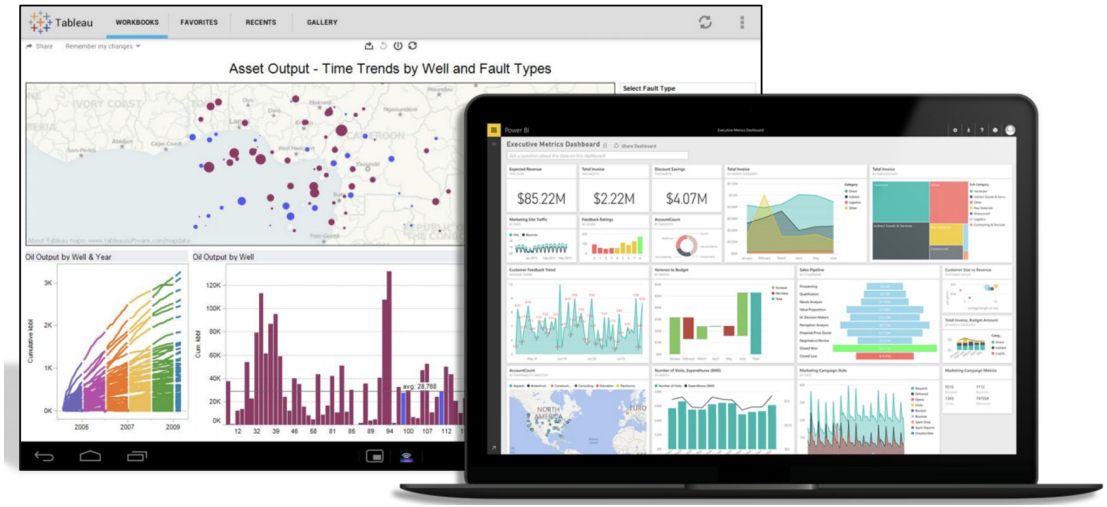
Dimension Tables

➢ Multidimensional "Cube" of data



OLAP

- Common queries:
  - Reporting and Business Intelligence (BI)
  - Drag-and-drop instead of SQL
  - Generate SQL automatically

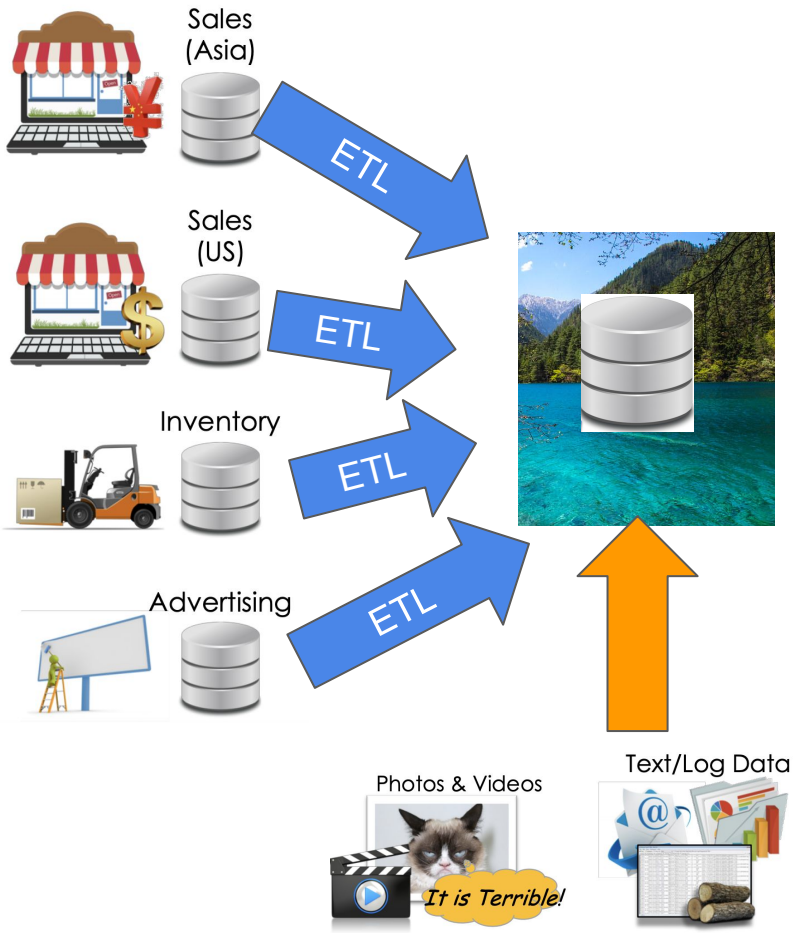


Database Evolution

- Data warehouse
  - How to support non-relational data
- Do we really need to force schema during ETL?

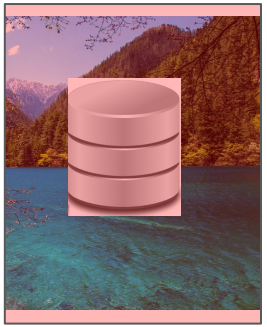
Enter Data Lake

- Data lake:
  - Lazy answer to these problems
  - Just dump it!
- Collect now, figure out later:
  - Whoever uses it later need to do more work:
    - E.g. infer the schema, clean the data, etc.



Data Lake

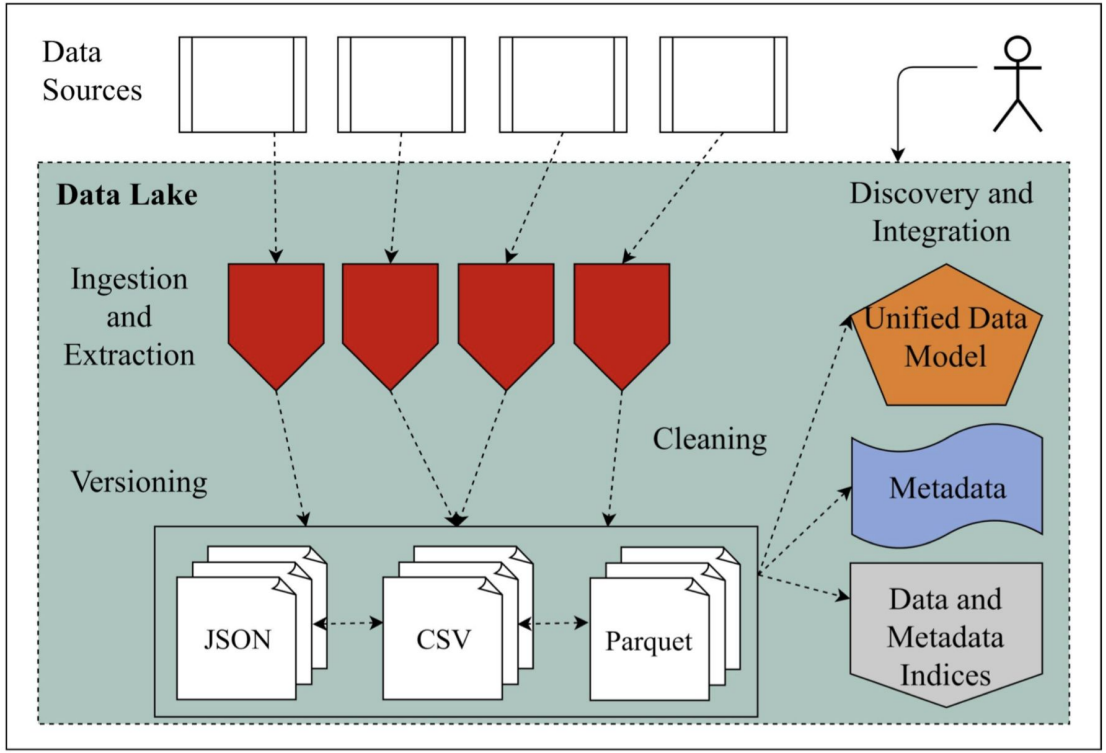
- The dark side
  - Cultural shift: *Curate* → *Save Everything*
    - More noise than signal
  - Missing context:
    - E.g.: given this file hdfs://.../xyz.csv_json.txt
      - Who created it?
      - What does it contain?
  - Dirty data:
    - Missing information, inconsistency



Data Lake

- What's in the lake

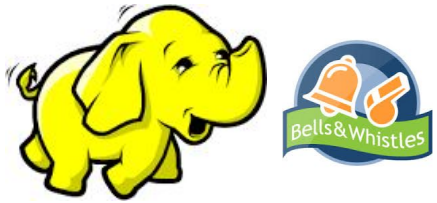
  just need to be able to dump things in and be able to retrieve things out

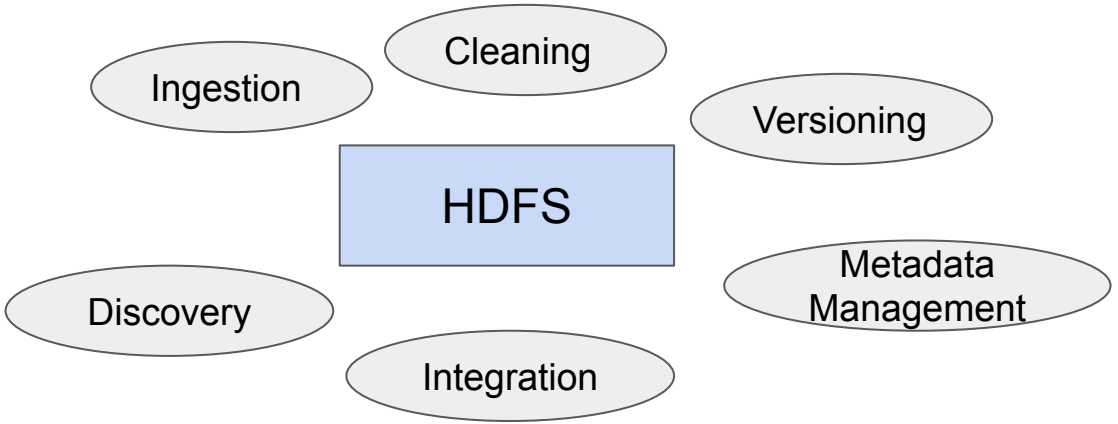


Data Lake

- A simple definition
  - HDFS plus many bells and whistles

No matter how good data lake is, still up to analyst to do a lot of cleaning.
At the core, Data lake is just HDFS with some functionalities on top of it.





Stream Systems

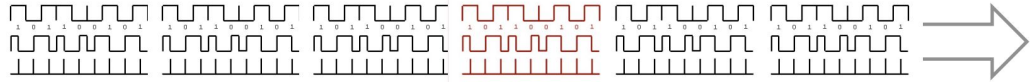
Stream Processing

- Recall Spark Streaming

Near real-time processing of continuous data

Fraud detection in bank transactions

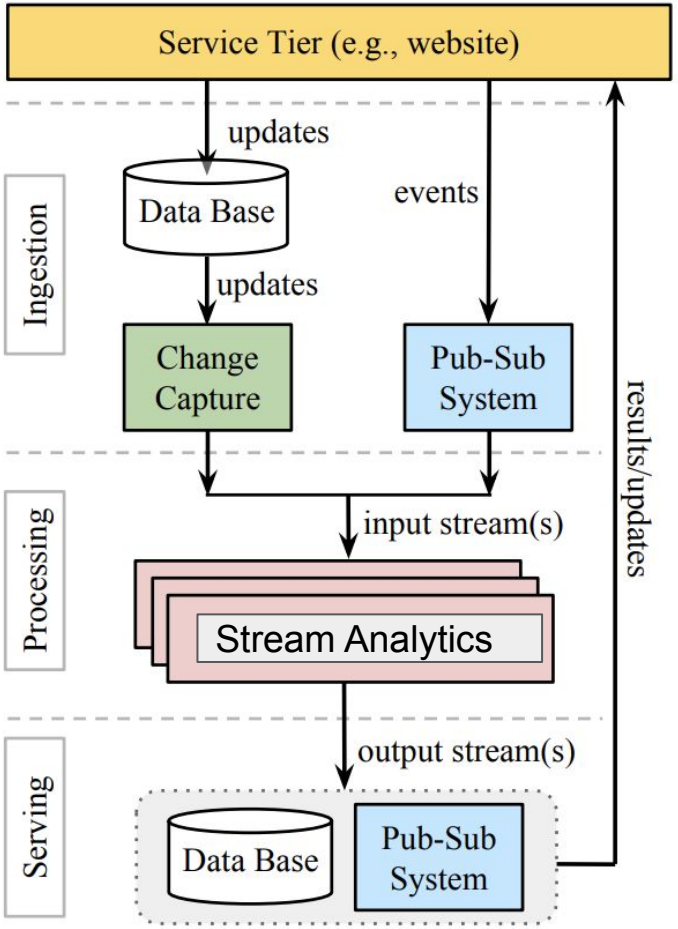
Anomalies in sensor data



Cat videos in tweets

Stream System

- Process data:
  - Near-real time

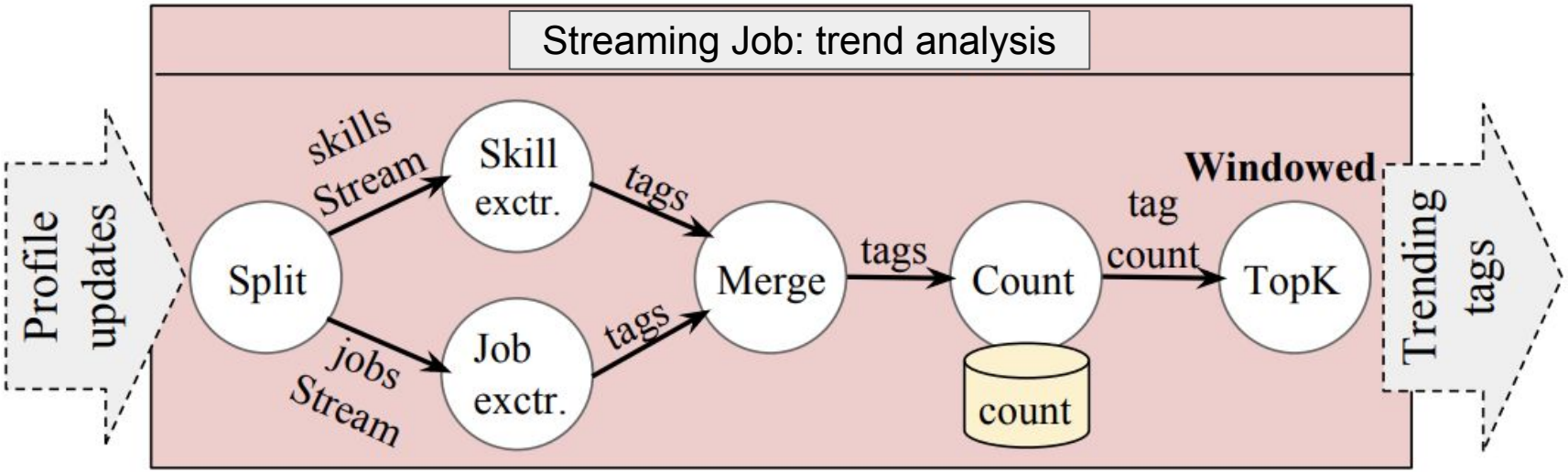


Stream System

TO implement TopK, need to keep some states which is the list of events you have seen so far. Update the local variable (state) whenever events come in.

- Example: trend analysis at LinkedIn

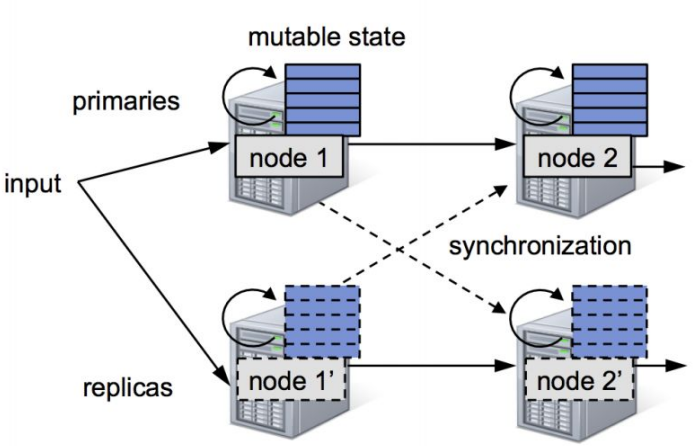
E.g. count is also another state.
*In Spark, we don't keep **states**, just throw everything away after we run*



Stream System

- Most systems (stateful):
  - Nodes connected in a graph
  - Data arrives in stream
  - Processed in one node, then sent out to the other nodes
  - Fault tolerance via replication

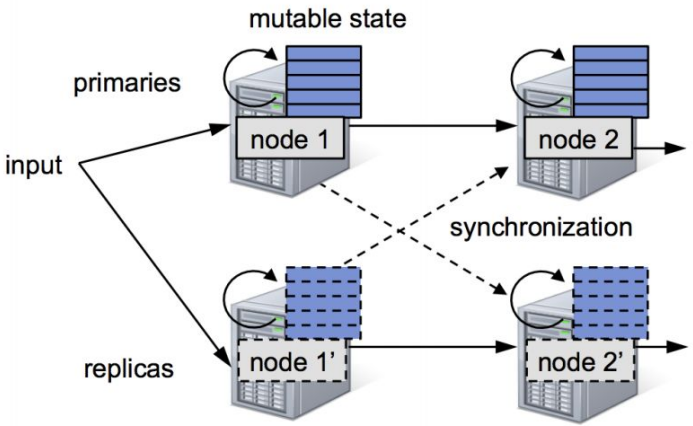
    expensive, have to do alot of replication to ensure data is not lost



Stream System

- Apache Storm:
  - Replay event if not processed
  - ***At least once*** semantics
    - Not good!
  - Mutable states lost if node fails
    - Need to be checkpointed

*possible that event will be **processed more than once.** (e.g. fraud detection where **correctness** is very important, cannot afford to process events more than once)*



Stream System

- Apache Flink
  - Designed for stream processing from the ground up
  - **Exactly-one processing**

If node fail and get rebooted, the event will not be replayed.

January 8, 2019

Alibaba Acquires Apache Flink Backer data Artisans

Alex Woodie

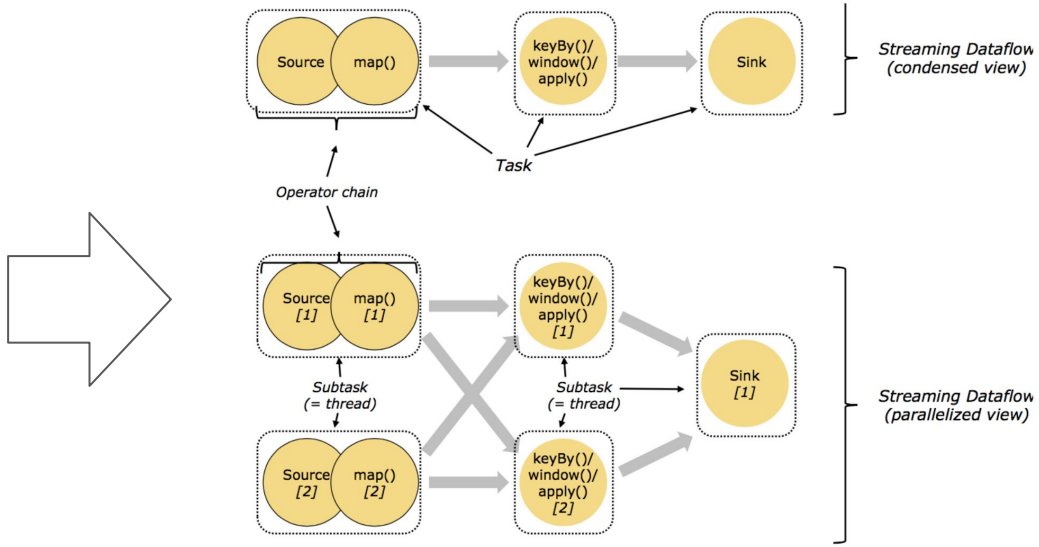
Chinese Internet giant Alibaba Group has acquired data Artisans, the German company behind the Apache Flink data processing framework. Alibaba is one of the biggest users of Flink, and plans to continue developing and Flink and contributing enhancements back to the open source project.

Apache Flink originated from the Stratosphere research project at the Technical University of

Stream System

- Flink
  - Stateful

```
...
socketStream = env.socketTextStream(...)
wordsStream = socketStream.
      flatMap(lambda x: x.split(“\\s+”)).
      map(lambda x: (x,1))
keyValPair = wordsStream.keyBy(0).
      timeWindow(1)
countPair = keyValPair.sum(1)
print(countPair)
env.execute()
```

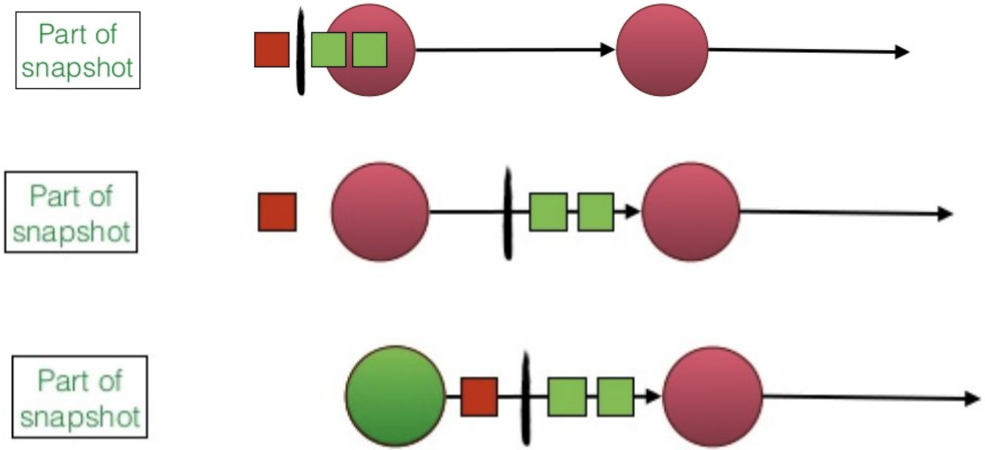


Stream System

Idea:
barrier just pass through, when pass through nodes, it takes the snapshot.

- Flink
  - Distributed snapshot

Barriers flow through the topology in line with data.

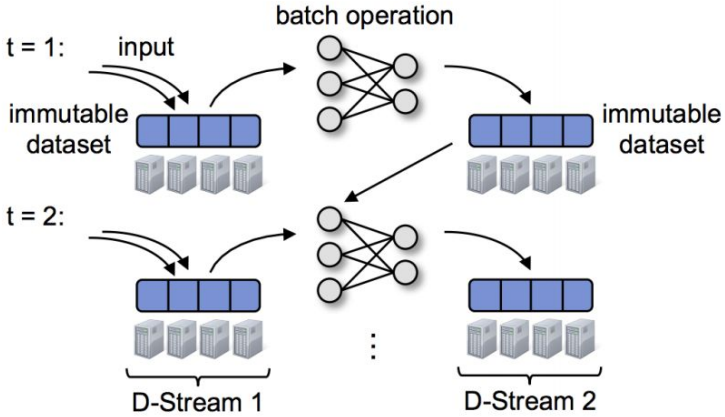


*These barriers are injected into the data stream and **flow** with the records as part of the data stream. Barriers never overtake records, they flow **strictly in line**. A barrier separates the records in the data stream into the set of records that goes into the current snapshot, and the records that go into the next snapshot. Each barrier carries the ID of the snapshot whose records it pushed in front of it. Barriers **do not interrupt the flow of the stream** and are hence very lightweight. Multiple barriers from different snapshots can be in the stream at the same time, which means that various snapshots may happen **concurrently**.*

Stream System

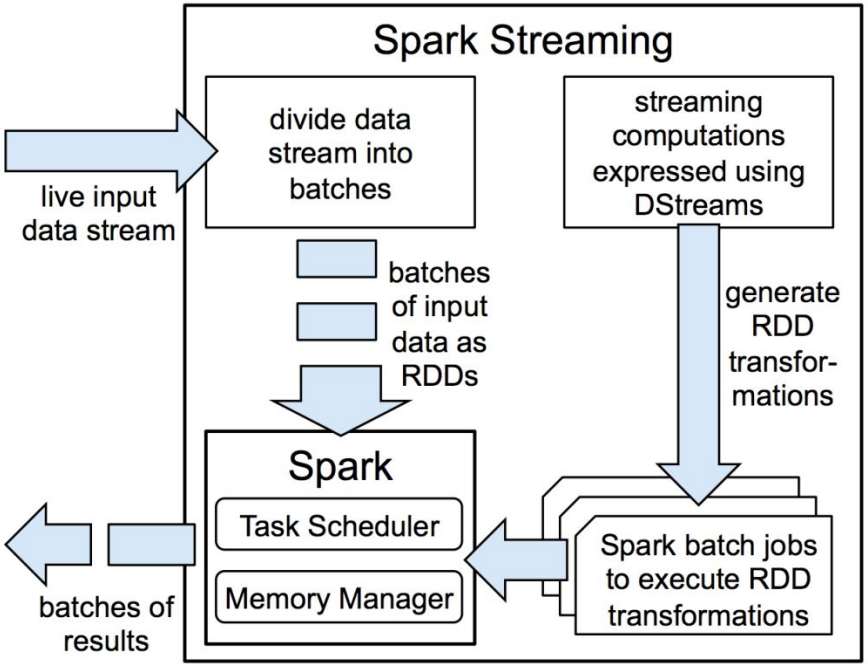
- Spark Streaming
  - Stateless
    - States are immutable RDDs
  - Turn stream into small batches
  - Run Spark on small batches
  - Fault tolerance inherited from Spark

***Spark** is designed for **batch** processing but it also supports **stream** processing (just change stream into small batches)*
- Every 1s data => turn into rdd => feed into Spark => get the output



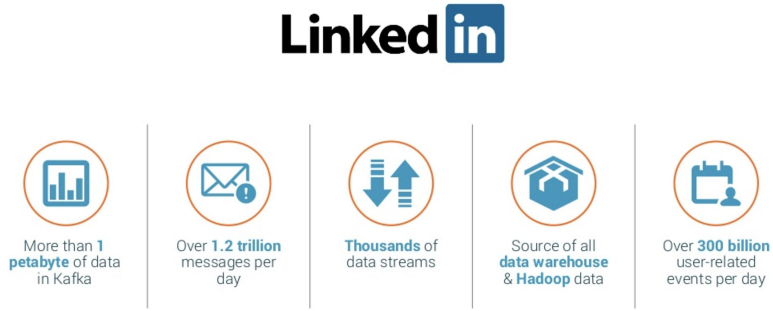
Stream System

- Spark Streaming
  - Architecture



Message System

- A variant of streaming systems
- Apache Kafka:
  - Support LinkedIn workload



How LinkedIn customizes Apache Kafka for 7 trillion messages per day

Jon Lee October 8, 2019

Share Tweet Share

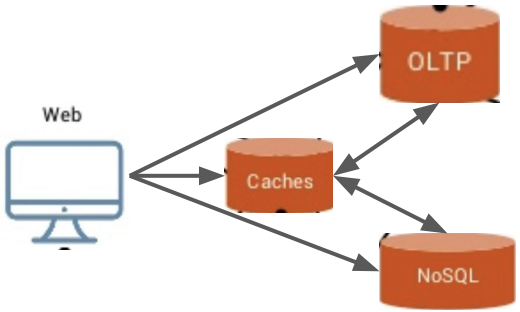
Co-authors: Jon Lee and Wesley Wu

Apache Kafka is a core part of our infrastructure at LinkedIn. It was originally

Why Kafka?

- Your application starts out like this

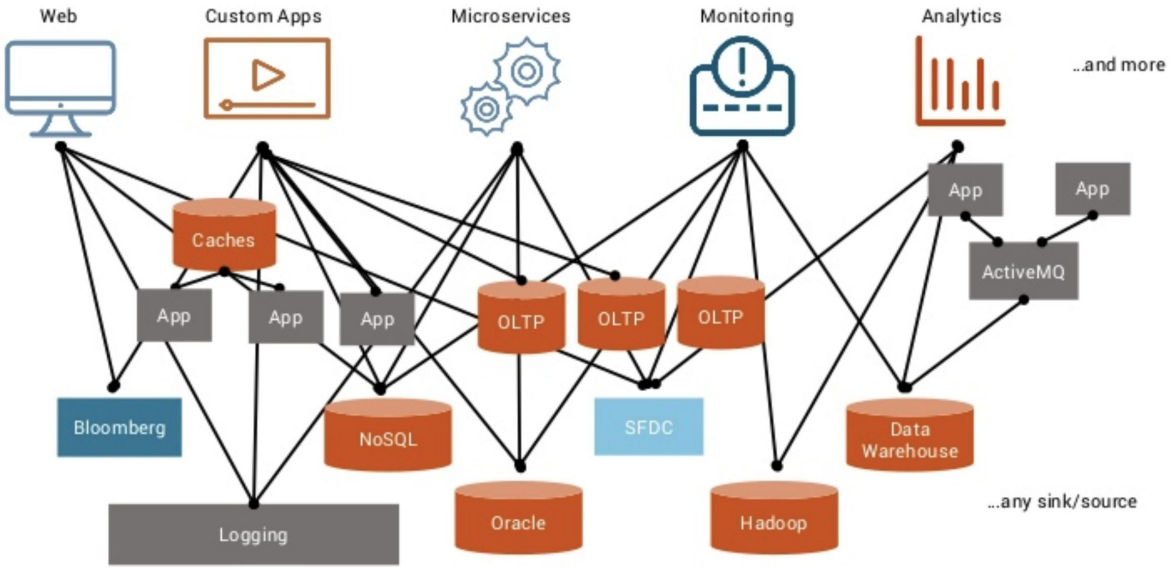
- Then maybe this



Why Kafka?

- And before you know it:

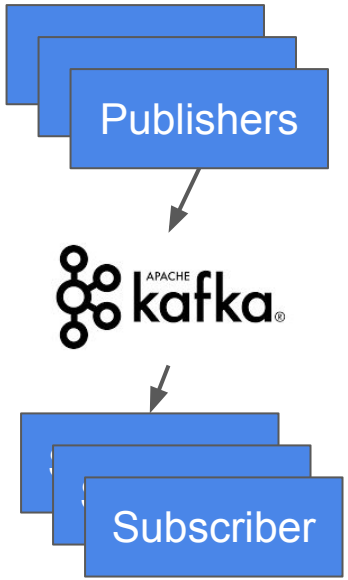
Kafka helps get out of this mess

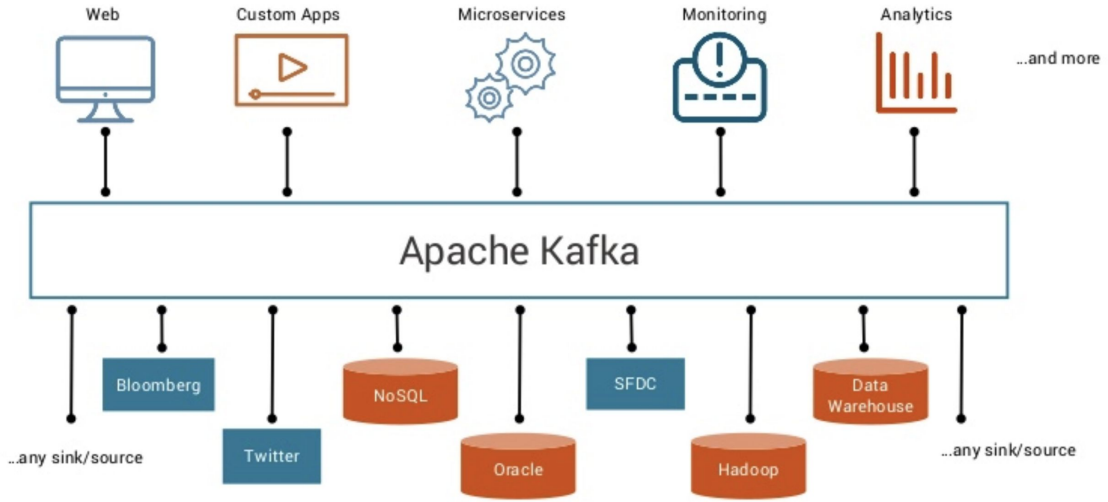


Why Kafka

*All inputs go into **Kafka**, whoever wanna use any of these events will just pull from Kafka.*
- Don't need to keep track of everything now, just push to Kafka

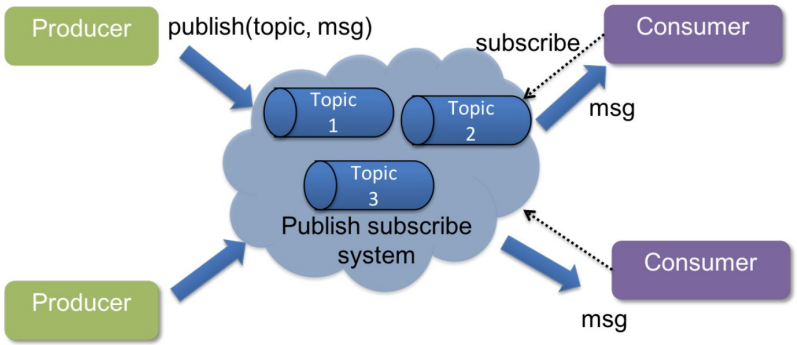
- Very specialized streaming task: message bus
  - Mass transit of messages/events between subsystems





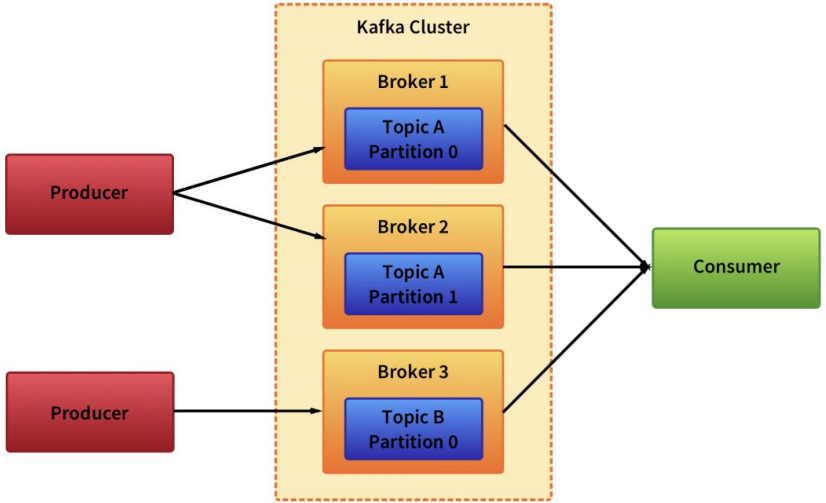
Kafka

- Architecture:
  - Producer publishes messages to a topic
  - Consumer subscribes to messages from a specific topic
  - Kafka cluster handles many topics



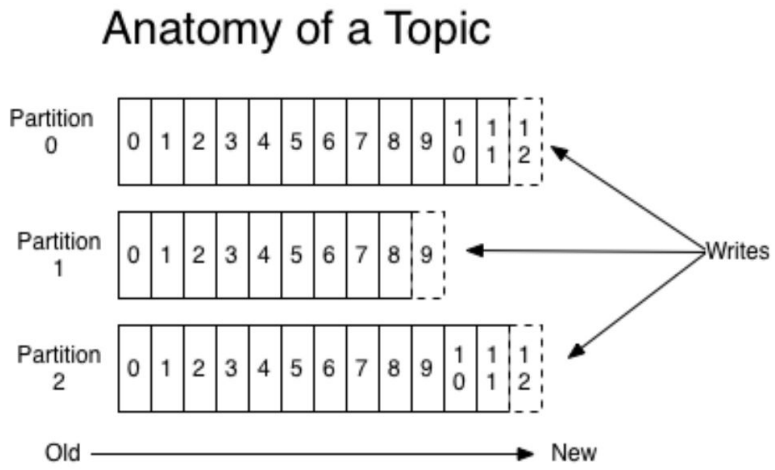
Kafka

- Architecture
  - Cluster consists of broker nodes
  - Each broker has > 1 partitions
  - Each topic split among multiple partitions
  - Producer split messages across partitions



Kafka

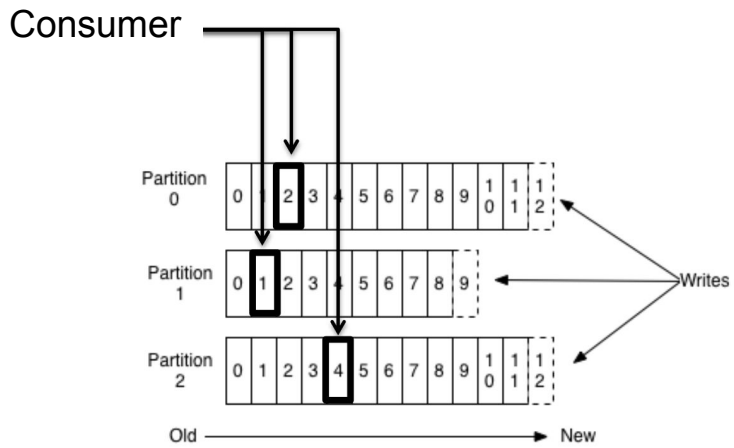
- A partition
  - A append-only log
  - Messages are ordered within the partition
  - **Not across partitions**



Kafka

ordered in partition but not across partitions.

- Consumer
  - Remember offset
    - <partitionID, offset>
  - So that it can leave and come back
  - 2 consumers of the same topic:
    - Have same event order per partition
    - But may have different order across partitions



Consumer 1: (P0, 2), (P1,1), (P2,4), (P2,5), (P2,6), (P1,2),...

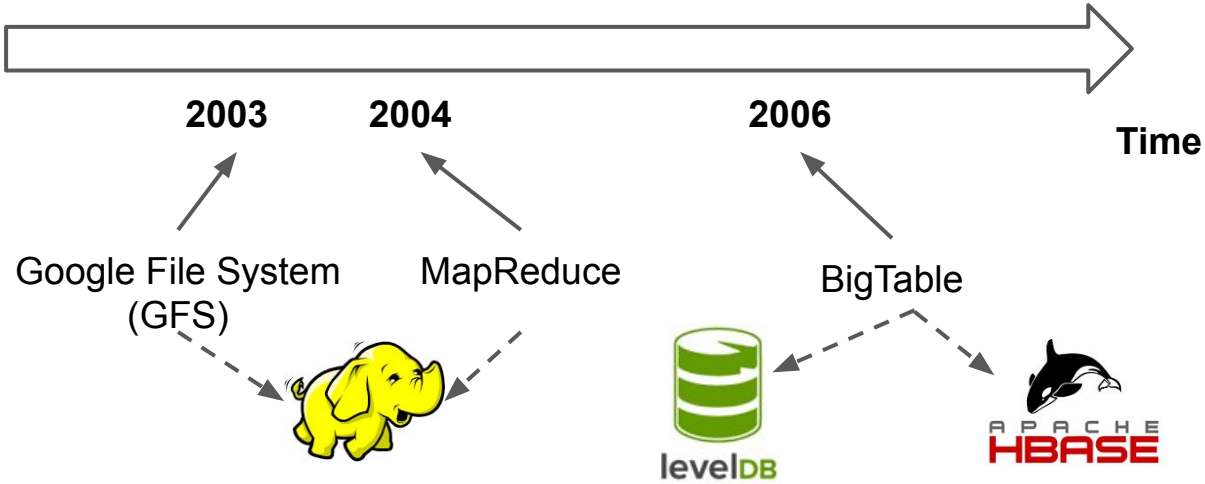
Consumer 2: (P1,2), (P1,3), (P1,4), (P0,2), (P2,5), (P0,3),...

Big Table

- It’s Google again!
  - 2006

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.



Big Table

GFS -> store data in binary block
- But sometimes you want structure in data. e.g. web ranking, need to crawl all the webs and store somewhere.
- Want to organise the data in some sort of structure.

- Why?
  - Wanted a storage system for semi-structured:
    - URL: content, metadata, rank, etc.
    - Map: road, images, point of interest, etc.
    - User: preference, recent queries, etc.
  - At scale:
    - Petabytes of data
    - Billions of objects

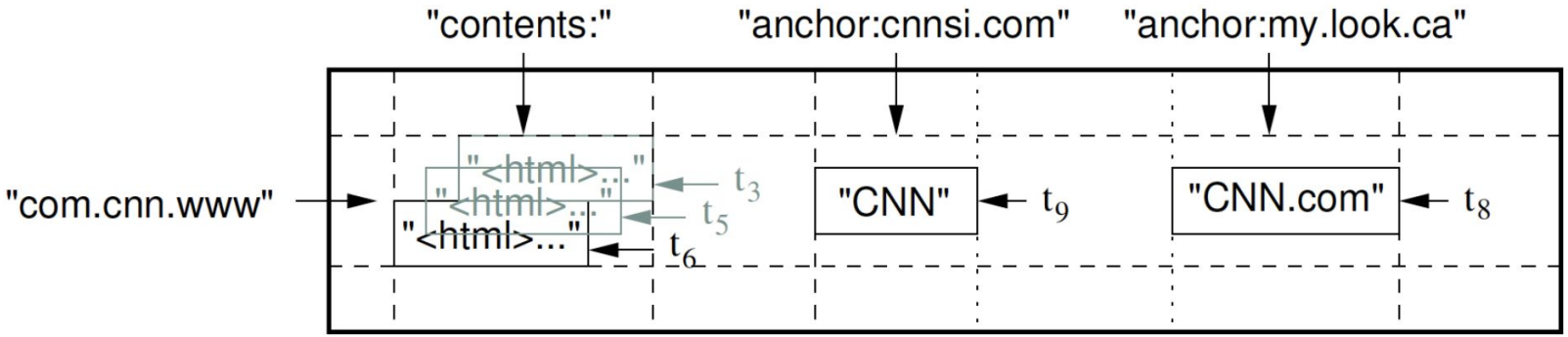
Why isn't GFS sufficient?

Big Table

- Is a distributed, sparse, multi-dimensional, sorted map

key is a tuple (row, col, timestamp)
value is binary

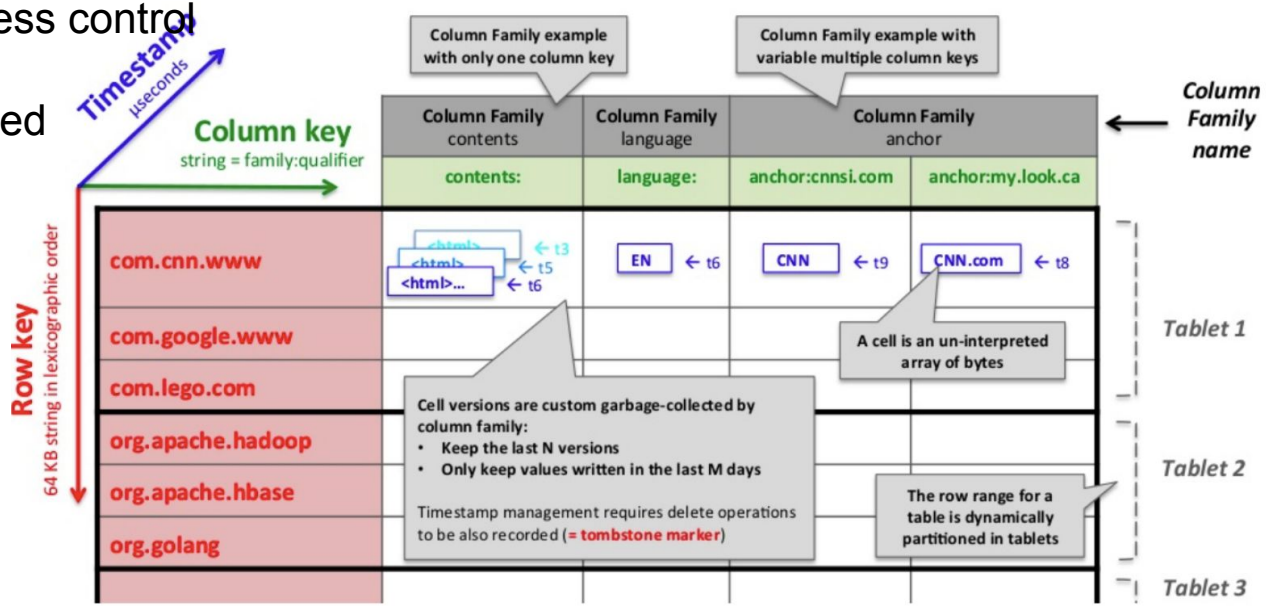
(row, column, timestamp) → cell content



Big Table

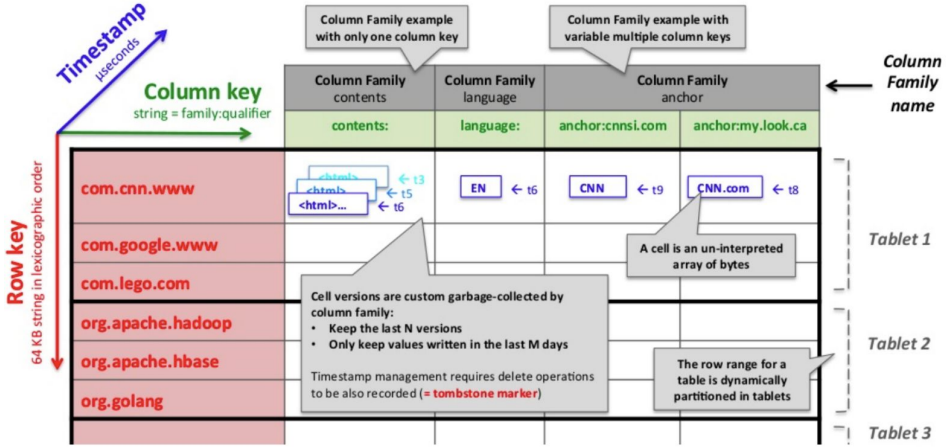
e.g. anchor column family

- Data model
  - Rows sorted lexicographically
  - New column can be added anytime
  - Column family:
    - Group of multiple columns
    - Basic unit for access control
    - Similar types
    - Can be compressed



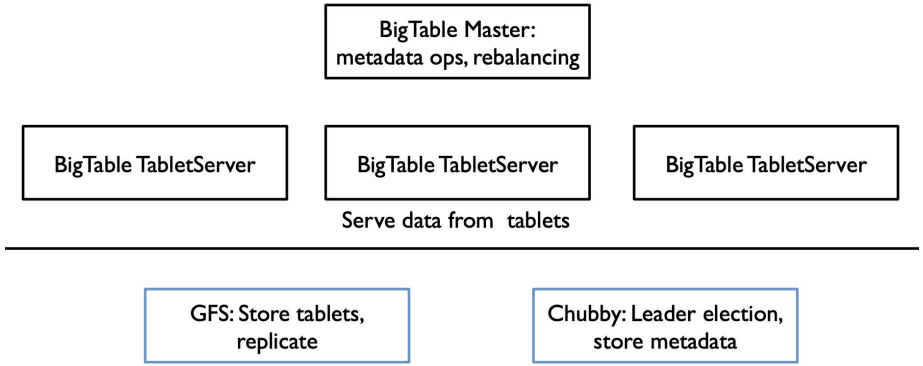
Big Table

- Tablet: *for one given row key, know exactly where tablet is.*
  - Group of multiple rows
  - Basic unit of distribution and load balancing:
    - Can be moved around
  - Rows in a tablet are sorted
  - ~200MB per tablet



Big Table

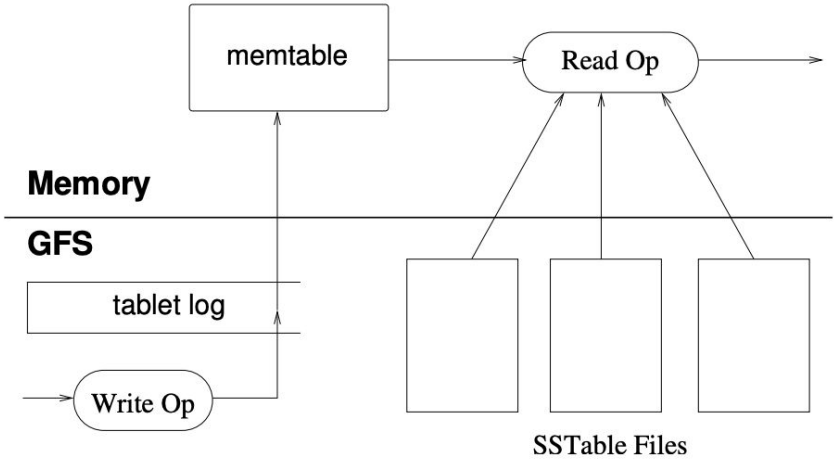
- System architecture
  - Similar to GFS
- Master:
  - Mapping row key → tablet server
  - Single point of failures
- Atomic write
  - Recall atomicity in week 8



Big Table

- Write operation
  - Entire row at a time
  - Client → tablet server
  - Atomicity via WAL
- How it works:
  - Write to commit log (GFS)
  - Buffer in memory
  - Regularly flushed to SSTables

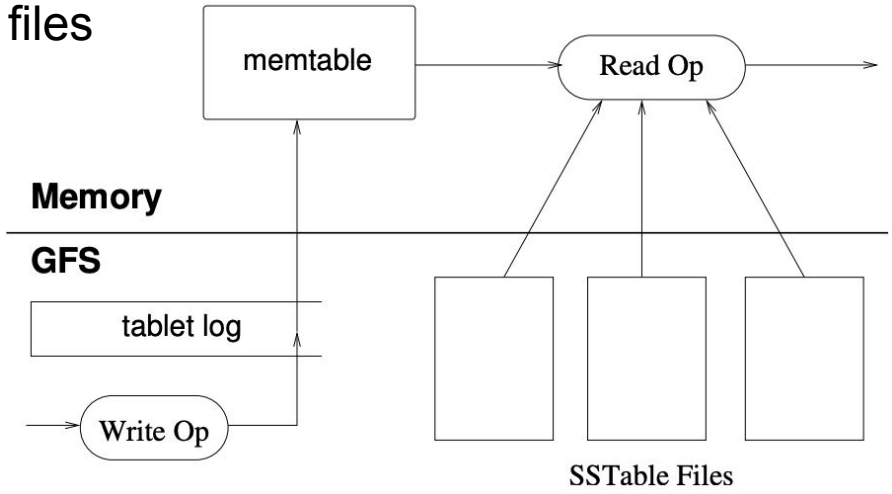
```
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");

// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```



Big Table

write operation cached in memory.
All values in Big Table are Strings

- Write operation
  - memtable and SSTables are sorted
  - Regular compaction on SSTables files
  - This design inspires LevelDB
- Read request:
  - Check on memtable
  - If not exist, read from SSTable



Big Table

- In production

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

Big Table

- Both GFS and Big Table show values of
  - Deep understanding of application workloads
  - Use the workloads to guide design
  - Make hard trade-off to simplify design
    - E.g. master-slave architecture
  - Simple designs scale well.

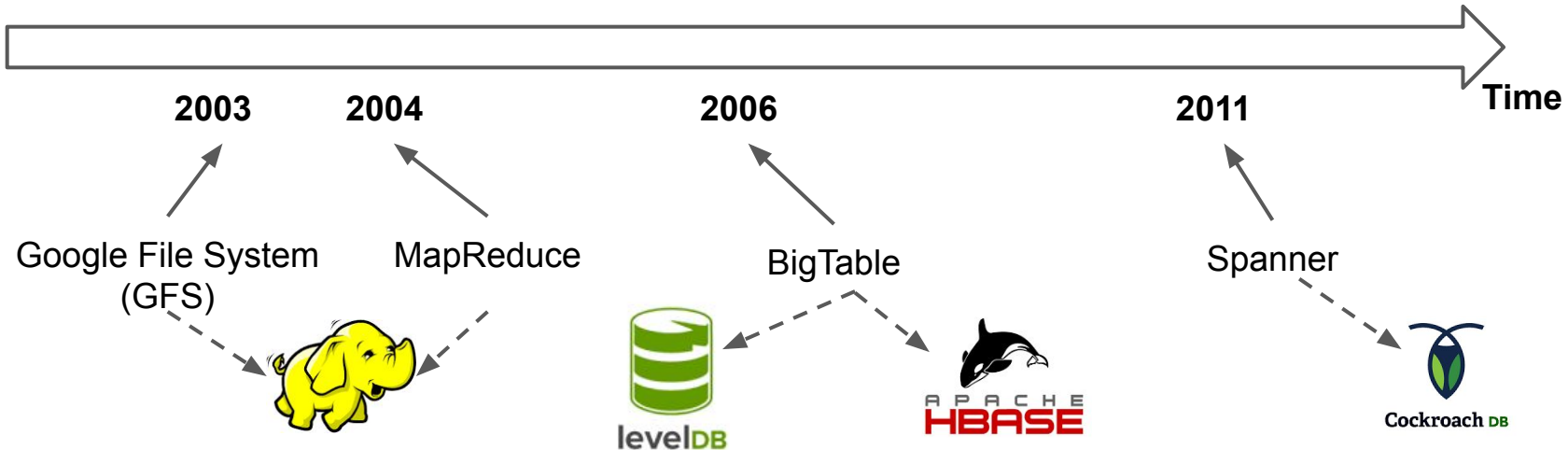
Spanner

- Google again
- Why another system?

Spanner: Google's Globally-Distributed Database

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford

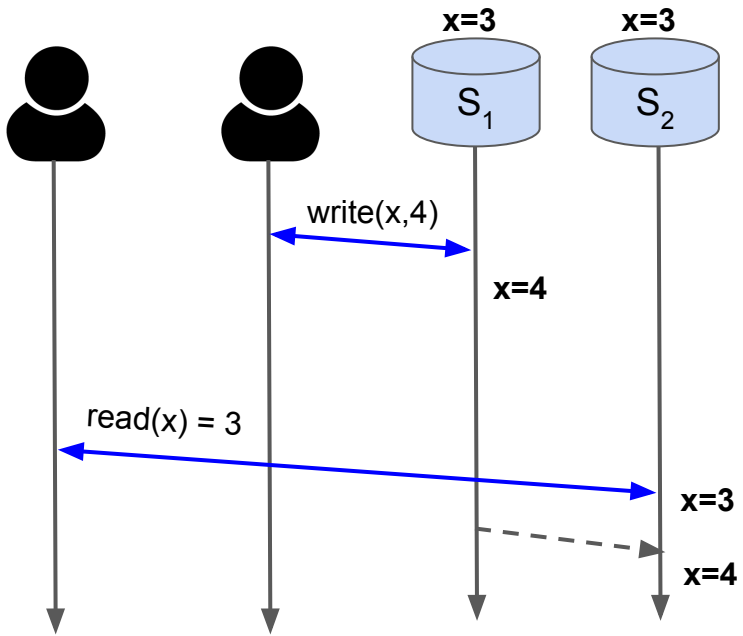
"We wanted to make it easy for developers to build their applications"



Spanner

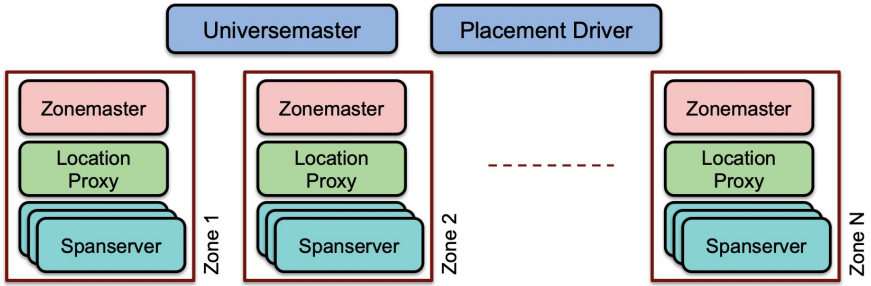
Takes a while for x=4 to be propagated to other replicas.
*The read becomes **stale** when the read is reading the replica (x=3)*

- Problems with BigTable:
  - Atomicity per row:
    - Does not extend to multi-row update
  - Eventual consistency:
    - Tablets are replicated over S_1, S_2
    - Write to S_1 at time t, Read from S_2 at time t+1
    - No guarantee that Read see the latest write



Spanner

- Spanner = Big Table with:
  - SQL support
  - ACID transactions
  - No problems with replicas
- Architecture:
  - Tables chopped into tablets
  - Tablets stored in Spanservers
  - 1000s Spanservers per zone

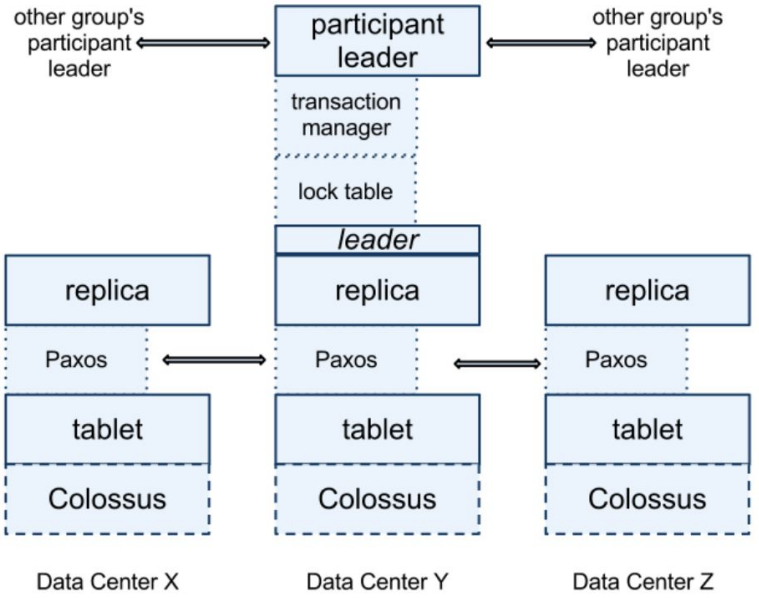


***Master** have to keep track of more information*
(now need to keep track of spanservers)

Spanner

*- Write **successful** only when all the replicas **agree** that write has succeeded.*
- Complicated because some replicas may fail during the write.
- Don't need to know the protocol, just know its solved by this.

- Replication:
  - Each tablet replicated over 3 data centers
  - Kept consistent by a **consensus protocol**
    - Solve inconsistency problem
    - Example: Paxos
    - Blockchain/Bitcoin built around variants of Paxos
    - More in your Distributed System course



Spanner

- Introduce one important technology
  - TrueTime!
- Having all servers agree on the same time is good:
  - You can have very fast transactions!
- Problem about global time
  - You **cannot** have *perfectly synchronized time* across many servers
    - Because of clock drift

Spanner

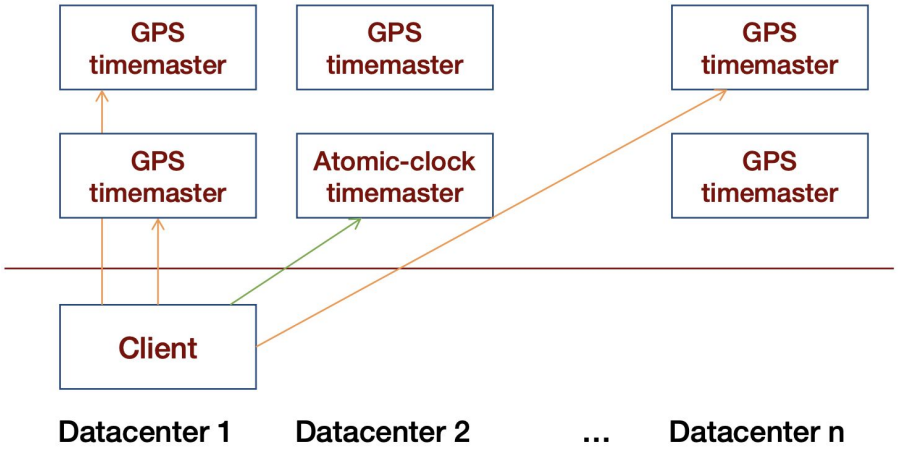
- How to minimize the differences between clocks:
  - NTP: too slow
    - Around 100ms-500ms
- Google implement their own protocol for time:
  - Each data center has GPS receivers + atomic clocks
  - Servers syncs frequently with the clocks
  - Monitor servers for clock drift
    - Evict bad ones

Spanner

- TrueTime *between any 2 servers in the world*
  - 7ms uncertainty interval
  - Simple APIs

Method	Returns
$TT.now()$	$TTinterval$: $[earliest, latest]$
$TT.after(t)$	true if t has definitely passed
$TT.before(t)$	true if t has definitely not arrived

Table 1: TrueTime API. The argument t is of type $TTstamp$.



Summary

- Big Data gives rise to many innovative systems
  - Distributed in the cloud
- Storage: GFS, BigTable, Data lakes
- General data processing: MapReduce, Spark
- Relational DB: Spanner
- Stream: Storm, SparkStreaming, Flink, Kafka
- Many more not covered in this course

Summary

- Recurring themes:
  - Master-slave architecture:
    - Master stores important metadata. Single point of failure
    - Slaves do all the work
  - Scalability and fault tolerance are (relatively) easy when data is immutable
  - But many systems require mutable states: stream processing, relational databases, etc.
    - Replication causes inconsistency
  - No one-size-fit-all systems:
    - Batch, interactive, stream, graph, ML, etc.
    - Each design comes from deep understanding of new workloads.