9. Beyond Traditional RDBMS

- Mega Trends in Data Management
- NoSQL (Not Only SQL) Databases
- NewSQL Databases

9. Beyond Traditional RDBMS

- ☞ Mega Trends in Data Management
- ❖ NoSQL (Not Only SQL) Databases
- ❖ New SQL Databases

History of Data Management

- (Early-2000s) Business processing → Relational Database Management Systems (RDBMS)
  - E.g., Oracle, IBM DB2, Sybase, SQL Server, etc.
- (Mid-2000s) Internet blooming → low-cost RDBMS alternatives
  - Supported transactions, replication, recovery
  - Still must use custom middleware to scale out across multile machines
  - Memcache for caching queries
  - E.g., MySQL, PostgreSQL

History of Data Management

- ❖ (Late-2000s) Big Data → NoSQL

“… *the whole point of seeking alternatives is that you need to solve a problem that relational databases are a bad fit for* …”

- Eric Evans

- ❖ Class of non-relational data storage systems
  - Usually do not require a fixed table schema nor do they use the concept of joins
- ❖ Relax one or more of the ACID properties
  - Brewer′s CAP theorem

Challenge 1: Scaling Up

- Datasets are just too big
- Hundreds of thousands of visitors in a short-time span → a massive increase in traffic
  - Developers begin to front RDBMS with a read-only cache to offload a considerable amount of the read traffic
  - Memcache or integrate other caching mechanisms within the application (i.e., Ehcache)
    - In-memory indexes, distributing and replicating objects over multiple nodes
  - As datasets grow, the simple memcache/MySQL model (for lower-cost startups) started to become problematic.

Challenge 2: Availability

- A web-site is most likely to be unavailable when it is most needed → a huge volume of revenue loss

- Goal of web services today is to be as available as long as the network is on.
  - When some nodes crash or some communication links fail, the service still performs as expected
  - One desirable fault tolerance capability is to **survive a network partitioning into multiple parts**.
    - Distributed DBMSs (covered in the course) provides no solutions yet ...

Traditional Approach

Structured & Repeatable Analysis

Business Users

Determine what question to ask

IT

Structures the data to answer that question

Monthly sales reports
Profitability analysis
Customer surveys

Big Data Approach

Iterative & Exploratory Analysis

IT

Delivers a platform to enable creative discovery

Business

Explores what questions could be asked

Brand sentiment
Product strategy
Maximum asset utilization

Possible Solutions to Scalability

- Began to look at multi-node database solutions
  - Distributed Database Systems
    - Basic principles and implementation techniques have been covered in the course
  - More techniques
    - To be covered by the next few slides

Scaling RDBMS – Master/Slave

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to slaves
- Multi-Master replication

Scaling RDBMS - Partitioning

- Partition or sharding
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware (in contrast to DDB)
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Scaling RDBMS - NoSQL

- NoSQL systems are able to scale horizontally right out of the box:
  - Schemaless
  - Not ACID (i.e., eventual consistency)
  - Many are based on Google’s Big Table or Amazon’s Dynamo systems

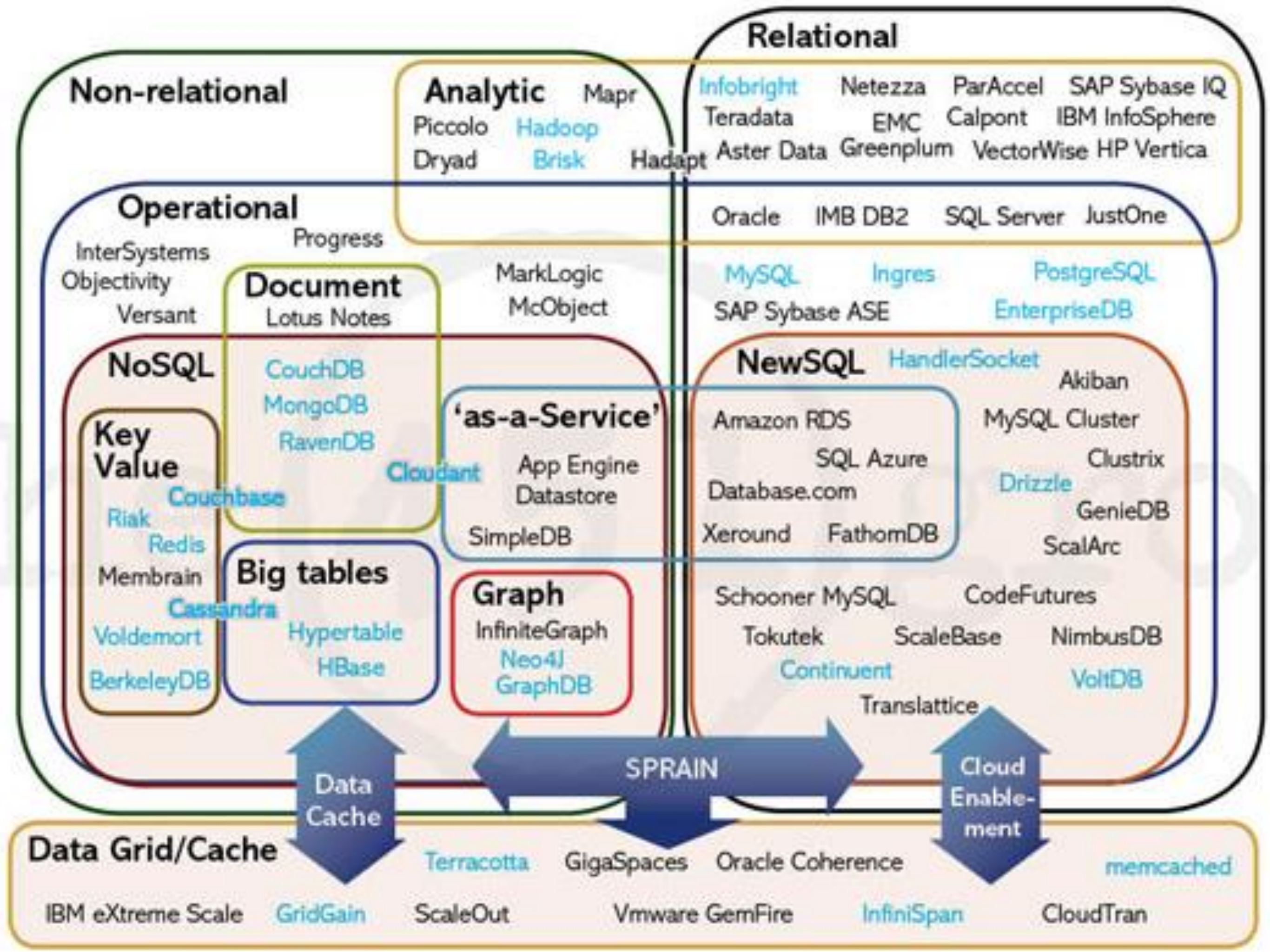
Scaling RDBMS - NewSQL

- (Early-2010s) Big Data → NewSQL
- New DBMSs that can scale across multiple machines natively and provide ACID guarantees
  - New Architectures
  - New MySQL storage enginees
  - Transaparent Clustering/Sharding

NewSQL Definition

- SQL as the primary interface
- ACID support for transactions
- Non-locking concurrency control
- Higher per-node performance
- Parallel, shared-nothing architecture

NoSQL, NewSQL, and Beyond (by 451 Group)



"NoSQL, NewSQL and Beyond: The answer to SPRAINed relational databases"

9. Beyond Traditional RDBMS

- ❖ Mega Trends in Data Management
- ☞ NoSQL (Not Only SQL) Databases
- ❖ New SQL Databases

NoSQL (Not Only SQL)

- INSERT only, no UPDATE/DELETE
- No JOIN, thereby reducing query time
  - This involves de-normalizing data
- Lack of SQL support
- Non-adherence to ACID (Atomicity, Consistency, Isolation and Durability) properties

Three Seeds of NoSQL

- BigTable (Google, 2006)
- Dynamo (Amazon, 2007)
  - Distributed key-value data store
- CAP Theorem (Eric A. Brewer)
  - BASE vs ACID

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm;
- Not a backlash/rebellion against RDBMS;
- SQL is a rich query language that cannot be rivaled by the current list of NoSQL (Not Only SQL) offerings.

Google's BigTable

- A distributed storage system for managing structured data.
- Designed to scale to a very large size
  - Petabytes of data across thousands of servers
- Used for many Google projects
  - Web indexing, Personalized Search, Google Earth, Google Analytics, Google Finance, …
- Flexible, high-performance solution for all of Google's products

Motivation for BigTable

- Lots of (semi-)structured data at Google
  - URLs:
    - Contents, crawl metadata, links, anchors, pagerank, …
  - Per-user data:
    - User preference settings, recent queries/search results, …
  - Geographic locations:
    - Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …
- Scale is large
  - Billions of URLs, many versions per page
  - Hundreds of millions of users, thousands of queries per sec
  - 100TB+ of satellite image data

Why Not Just Use Commercial DB?

- Scale is too large for most commercial databases
- Even if it weren't, cost would be very high
- Low-level storage optimizations help performance significantly
  - Building internally means system can be applied across many projects for low incremental cost
  - Much harder to do when running on top of a database layer

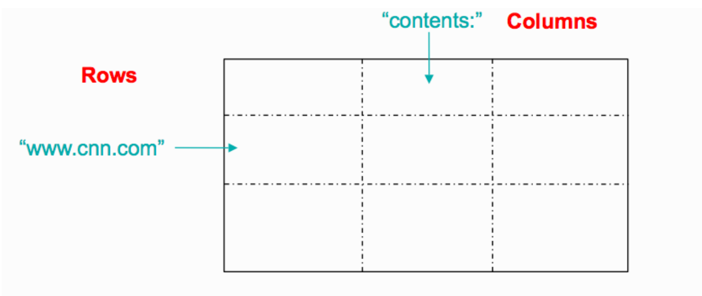
Google's Goals

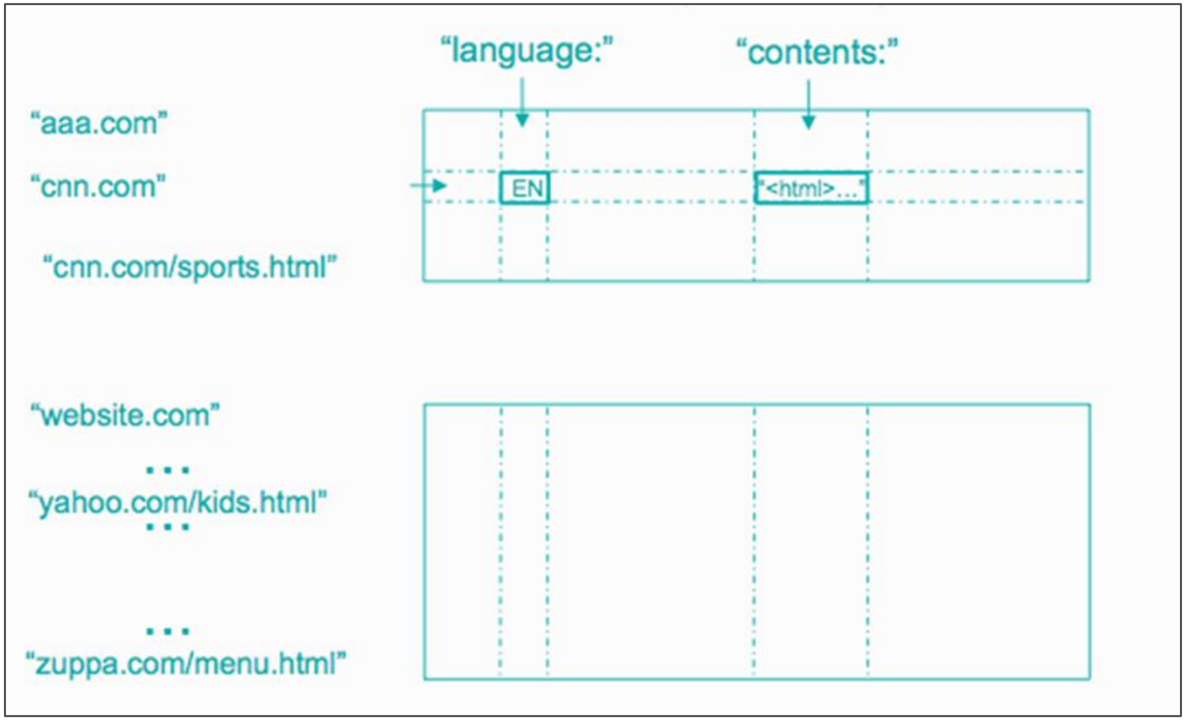
- Want asynchronous processes to be continuously updating different pieces of data
  - Want access to most current data at any time
- Need to support:
  - Very high read/write rates (millions of ops per second)
  - Efficient scans over all or interesting subsets of data
  - Efficient joins of large one-to-one and one-to-many datasets
- Often want to examine data changes over time
  - E.g. Contents of a web page over multiple crawls

Basic Data Model - BigTable

- A sparse, distributed, persistent, multi-dimensional sorted map

  (row, column, timestamp) → cell contents





- Accommodate a large collection of web pages and related information

Rows

- Use URLs as row keys
- Various aspects of a web page as column names
- Store contents of a web page in the contents: column under the timestamps when they were fetched.
- Name is an arbitrary string
  - Access to data in a row is atomic
  - Row creation is implicit upon storing data
- Rows ordered lexicographically
  - Rows close together lexicographically usually on one or a small number of machines

Rows (*cont.*)

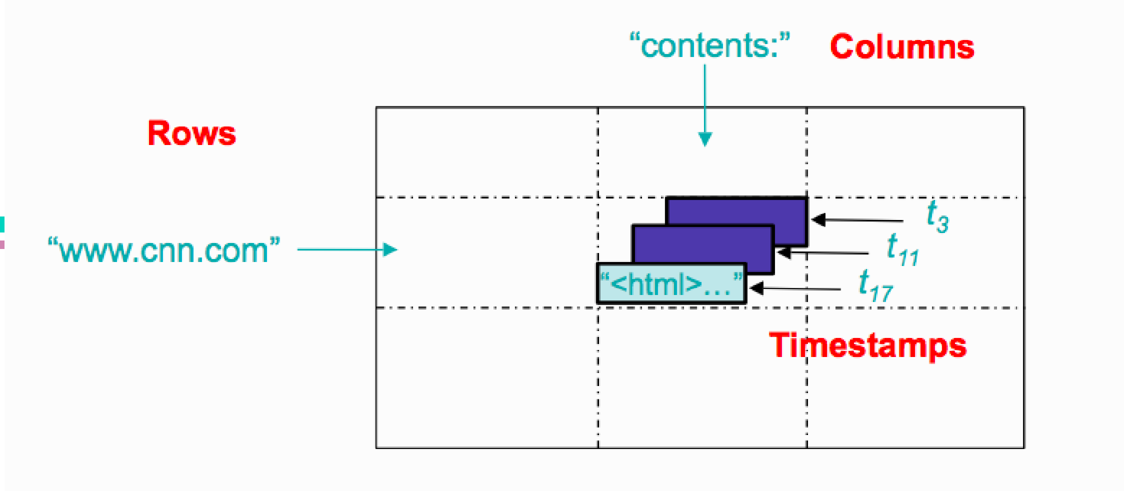
- Reads of short row ranges are efficient and typically require communication with a small number of machines.
- Can exploit this property by selecting row keys so they get good locality for data access.
- Example:

```
math.gatech.edu, math.uga.edu, phys.gatech.edu,
phys.uga.edu
```

vs

```
edu.gatech.math, edu.gatech.phys, edu.uga.math,
edu.uga.phys
```

Timestamps



- Used to store different versions of data in a cell
  - New writes default to current time, but timestamps for writes can also be set explicitly by clients
- Items in a cell are stored in decreasing timestamp order.
- Application specifies how many versions of data items are maintained in a cell.
  - Bigtable garbage collects obsolete versions.

BigTable API

- Implementation interfaces to
  - create and delete tables and column families
  - modify cluster, table, and column family metadata (e.g., access control rights)
  - write or delete values in Bigtable
  - look up values from individual rows
  - iterate over a subset of the data in a table
  - atomic R-M-W sequences on data stored in a single row key.

Advantages of BigTable

- Distributed multi-level map
- Fault-tolerant, persistent
- Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance

BigTables in Google's Applications

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

Application 1: Google Analytics

- Enable webmasters to analyze traffic patterns at their web sites. Statistics such as:
  - Number of unique visitors per day and the page views per URL per day
  - Percentage of users that made a purchase given that they earlier viewed a specific page.
- How?
  - A small JavaScript program that the webmaster embeds in their web pages.
  - Every time the page is visited, the program is executed.
  - Program records information about each request:
    - user identifier and the pages being fetched

Application 1: Google Analytics (*cont.*)

- Raw-Click BigTable (~ 200 TB)
  - A row for each end-user session.
  - Row name includes website's name and the time at which the session was created.
  - Clustering of sessions that visit the same web site in a sorted chronological order.
  - Compression factor: 6-7.
- Summary BigTable (~ 20 TB)
  - Stores predefined summaries for each web site.
  - Generated from the raw click table by periodically scheduled MapReduce jobs.
  - Each MapReduce job extracts recent session data from the raw click table.
  - Row name includes website's name and the column family is the aggregate summaries.
  - Compression factor is 2-3.

Application 2: Google Earth & Maps

- Move, view, and annotate satellite imagery at different resolution levels.
- One BigTable stores raw imagery (~ 70 TB):
  - Row name is a geographic segment. Names are chosen to ensure adjacent geographic segments are clustered together.
  - Column family maintains sources of data for each segment.
- There are different sets of tables for serving client data (e.g., index table).

Application 3: Personalized Search

- Records user queries and clicks across Google properties.
- Users browse their search histories and request for personalized search results based on their historical usage patterns.

Application 3: Personalized Search (*cont.*)

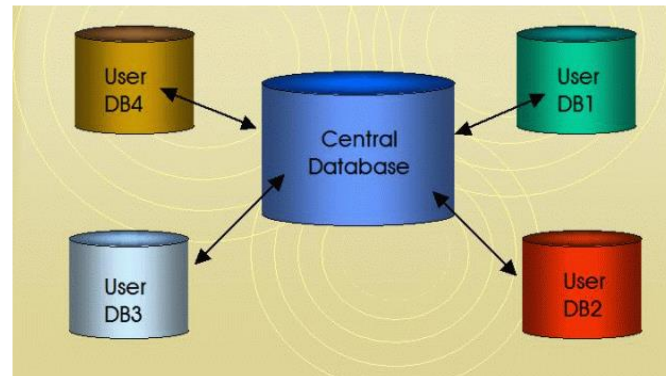
- One BigTable:
  - Row name is userid
  - A column family is reserved for each action type, e.g., web queries, clicks.
  - User profiles are generated using MapReduce.
    - These profiles personalize live search results.
  - Replicated geographically to reduce latency and increase availability.

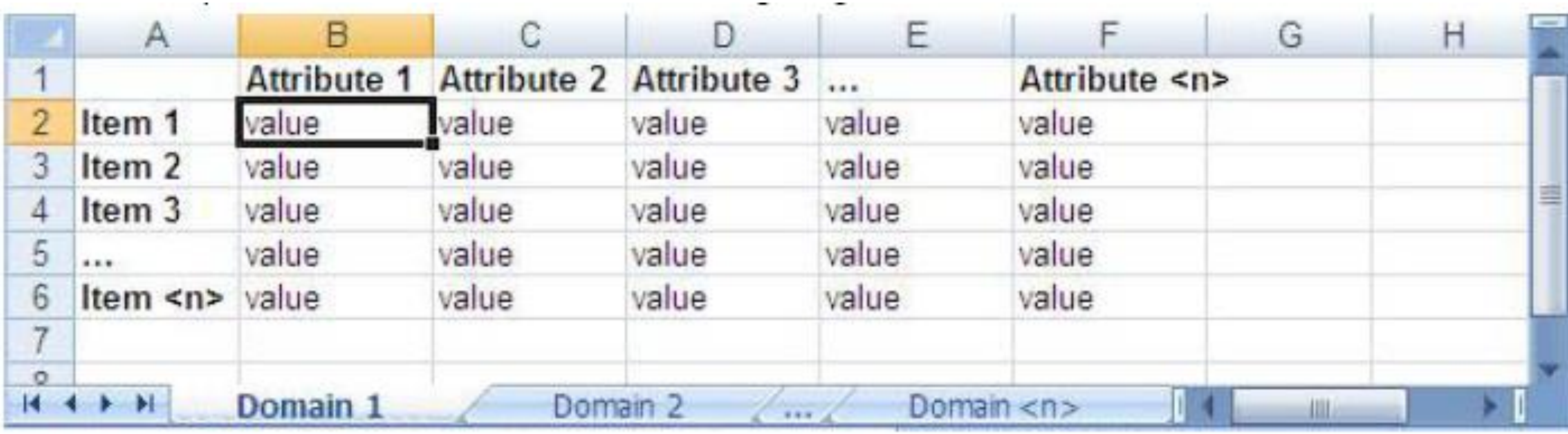
Amazon

- Huge Infrastructure
- Customer oriented business
- Reliability is key
- Guarantee Service Level Agreements
  - e.g., providing a response within 300ms for 99.9% of its requests for a peak client load of 500 requests per second.

Amazon’s Dynamo

- A distributed key-value storage system
  - Simple
  - Scale
  - Highly available





	A	B	C	D	E	F	G	H
1		Attribute 1	Attribute 2	Attribute 3	...	Attribute <n>		
2	Item 1	value	value	value	value	value		
3	Item 2	value	value	value	value	value		
4	Item 3	value	value	value	value	value		
5	...	value	value	value	value	value		
6	Item <n>	value	value	value	value	value		
7								

Domain 1 | Domain 2 | ... | Domain <n>

Requirements and Assumptions

- Query Model
  - simple read and write operations to a data item that is uniquely identified by a key.
- ACID Properties
  - Atomicity, Consistency, Isolation, Durability.
- Efficiency
  - latency requirements which are in general measured at the 99.9th percentile of the distribution.
- Other Assumptions
  - operation environment is assumed to be friendly and there are no security related requirements such as authentication and authorization.

Amazon SimpleDB

- A web service based on Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2)
- It stores, processes, and queries structured data in real time without operational complexity.
- It requires no schema, automatically indexes data, and provides a simple API for storage and access.
  - eliminating the administrative burden of data modeling, index maintenance, and performance tuning
- Developers gain access to its functionality within Amazon's computing environment, are able to scale instantly, and pay for what they use.

Features of SimpleDB

- Simple to use
- Flexible
- Scalable
- Fast
- Reliable
- Inexpensive
- Designed for use with other Amazon Web services

SimpleDB – Simple to Use

- Allowing users to quickly add data and easily retrieve or edit that data through a simple set of web service based API calls.

- Eliminating the complexity of maintaining and scaling users' operations.

SimpleDB - Flexible

- Unnecessary to pre-define all of the data formats one will need to store; simply add new attributes to the data set when needed, and the system will automatically index the data accordingly.

- Storing structured data without first defining a schema provides developers with greater flexibility when building applications.

SimpleDB - Scalable

- Allowing one to easily scale applications. Users can quickly create new domains as the data grows or your request throughput increases.

- Currently, users can store up to 10 GB per domain and can create up to 250 domains.

SimpleDB - Fast

- Providing quick, efficient storage and retrieval of data to support high performance web applications.

SimpleDB - Reliable

- The service runs within Amazon's high-availability data centers to provide strong and consistent performance.

- To prevent data from being lost or becoming unavailable, users’ fully indexed data is stored redundantly across multiple servers and data centers.

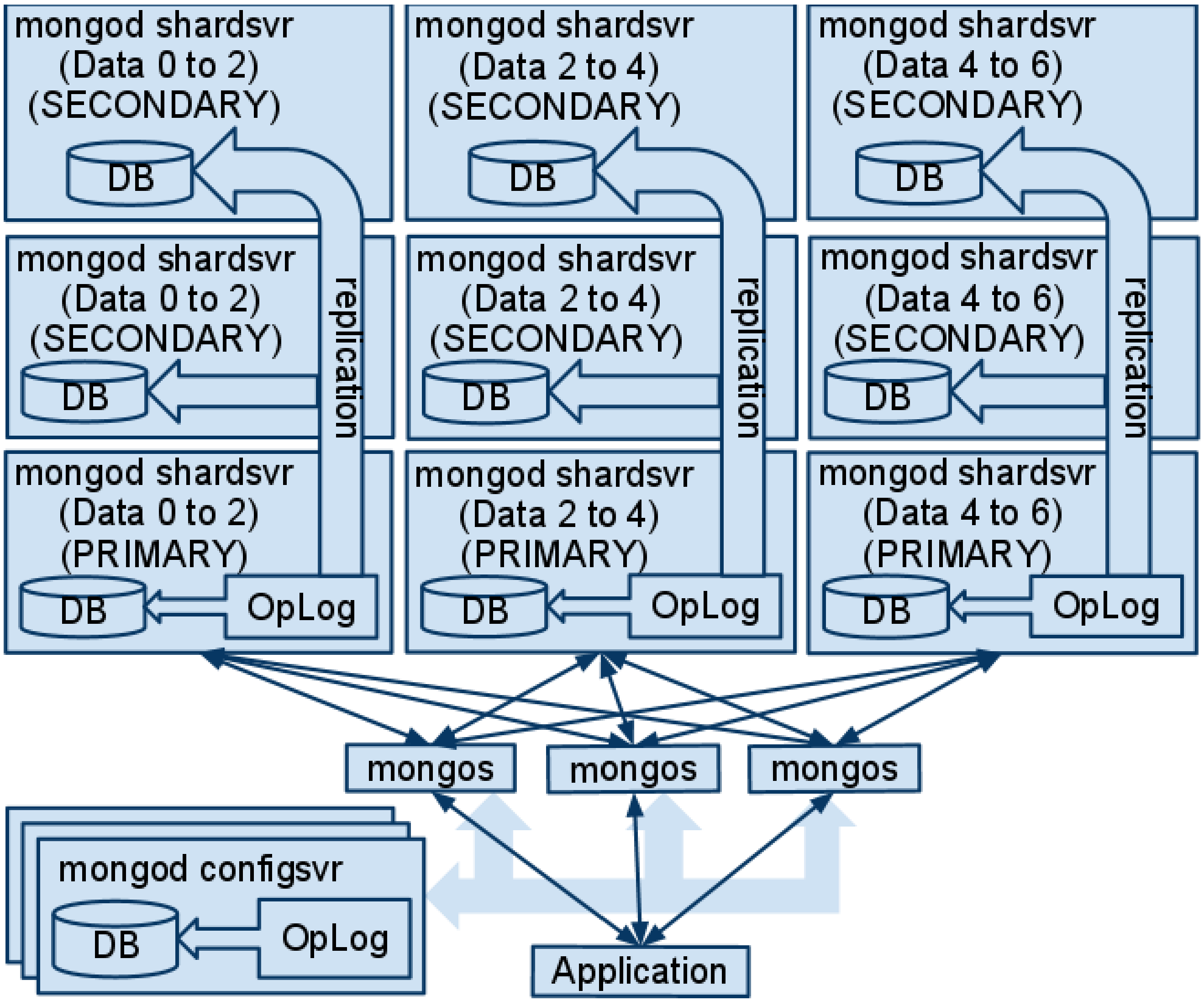
SimpleDB - Inexpensive

- Users pay only for resources they consume.
- Avoiding significant up-front expenditures traditionally required to obtain software licenses and purchase and maintain hardware, either in-house or hosted.
- Freeing users from many of the complexities of capacity planning, transforms large capital expenditures into much smaller operating costs, and eliminating the need to over-buy "safety net" capacity to handle periodic traffic spikes.

SimpleDB – Integration with other Amazon Web Services

- Integrating with other Amazon web services such as Amazon EC2 compute cloud and Amazon S3 storage.
  - E.g., developers can query the object metadata from within the application in Amazon EC2 and return pointers to the objects stored in Amazon S3.

MongoDB Architecture



- Easy to use
- Becoming more like a DBMS over time
- No transactions

9. Beyond Traditional RDBMS

- ❖ Mega Trends in Data Management
- ❖ NoSQL (Not Only SQL) Databases
- ☞ New SQL Databases

NewSQL Solutions

- NewSQL is a class of database systems that aim to provide the same scalable performance of NoSQL systems while still maintaining the ACID guarantees of traditional SQL relational databases.
  - When the application needs to handle very large datasets or a very large number of transactions
  - When ACID guarantees are required
  - When the application can significantly benefit from the use of the relational model and SQL

NewSQL Database Features

- SQL as the primary mechanism for application interaction.
- ACID support for transactions.
- A non-locking concurrency control mechanism so real-time reads will not conflict with writes.
- An architecture providing much higher per-node performance than available from traditional RDBMS solutions.
- A scale-out, shared-nothing architecture, capable of running on a large number of nodes without suffering bottlenecks.

Categorization of NewSQL Solutions

- New Architectures
- New MySQL Storage Engines
- Transparent Clustering/Sharding

New Achitectures

- Newly designed from scratch to achieve scalability and performance (operate in a distributed cluster shared-nothing nodes)
- One of the key considerations in improving the performance is making non-disk (memory) or new kinds of disks (flash/SSD) the primary data store
- Some (hopefully minor) changes to the code
- Data migration is needed
- Solutions can be software-only (VoltDB, NuoDB and Drizzle) or supported as an appliance (Clustrix, Translattice).
- Examples: VoltDB, NuoDB, Clustrix, Drizzle, Translattice, and VMware's SQLFire

New Database Eaxmple - VoltDB

- In-memory database
- ACID-compliant RDBMS
- Use a shared nothing architecture
- Written in Java and C++
- Supported operation systems:
  - Linus and Mac OS X
- Provides client libraries for
  - Java, C++, C#, PHP, Python and Node.js

ACID in VoltDB

- Atomicity: VoltDB defines a transaction as a stored procedure, which either succeeds or rolls back on failure
- Consistency: VoltDB enforces schema and datatype constraints in all database queries
- Isolation: VoltDB transactions are globally ordered and run to completion on all affected partitions without interleaving
- Durability: VoltDB provides replication of partitions, and periodic database snapshots combined with command logging to ensure high availability and database durability

New MySQL Storage Engines

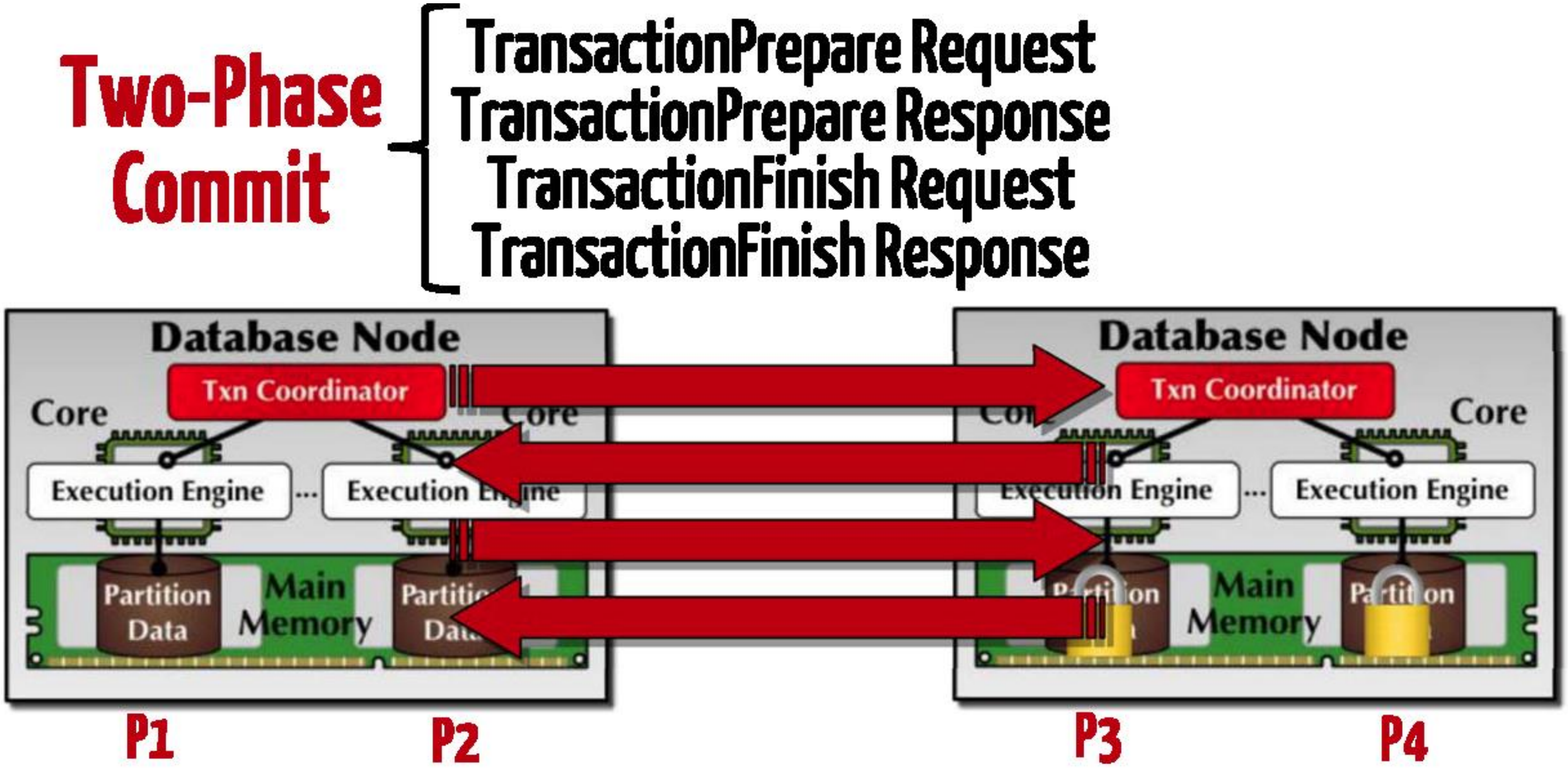
- MySQL is used extensively in OLTP
- To overcome MySQL's scalability problems, a set of storage engines are developed
- Use the same programming interface as MySQL but scale better
- Examples: Xeround, Akiban, MySQL NDB cluster, GenieDB, Tokutek, etc.
- The good part is the usage of the MySQL interface, but the downside is data migration from other databases (including old MySQL) is not supported.

Transparent Clustering/Sharding

- Provide a sharding middleware layer to automatically split databases across mutiple nodes
- Retain the OLTP databases in their original format, but provide a pluggable feature to cluster transparently for scalability.
- Provide a transparent sharding middleware layer to automatically split databases across mutiple nodes for scalability
- Both approaches allow reuse of existing skill sets and ecosystem, and avoid the need to rewrite code or perform any data migration.
- Examples: ScalArc, Schooner MySQL, dbShards and ScaleBase; and Continuent Tungsten

H-Store

- An in-memory distributed database system
- Support for arbitrary transactions

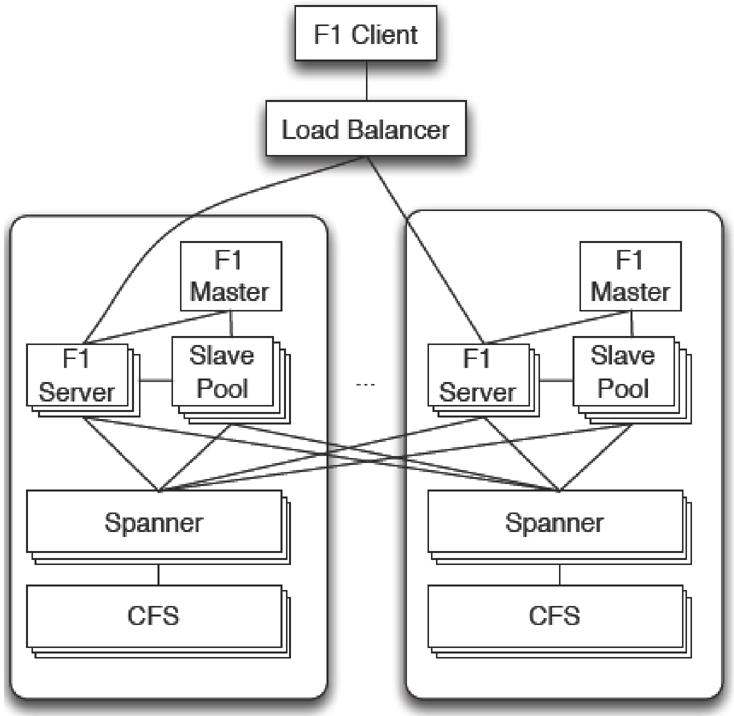


Google’s NewSQL Solution

- Google’s BigTable supports NoSQL and BASE
- Goolge also needs to support ACID
- Google’s F1 is a hybrid database that combines high availability, the scalability of NoSQL systems like Bigtable, and the consistency and usability of traditional SQL databases.

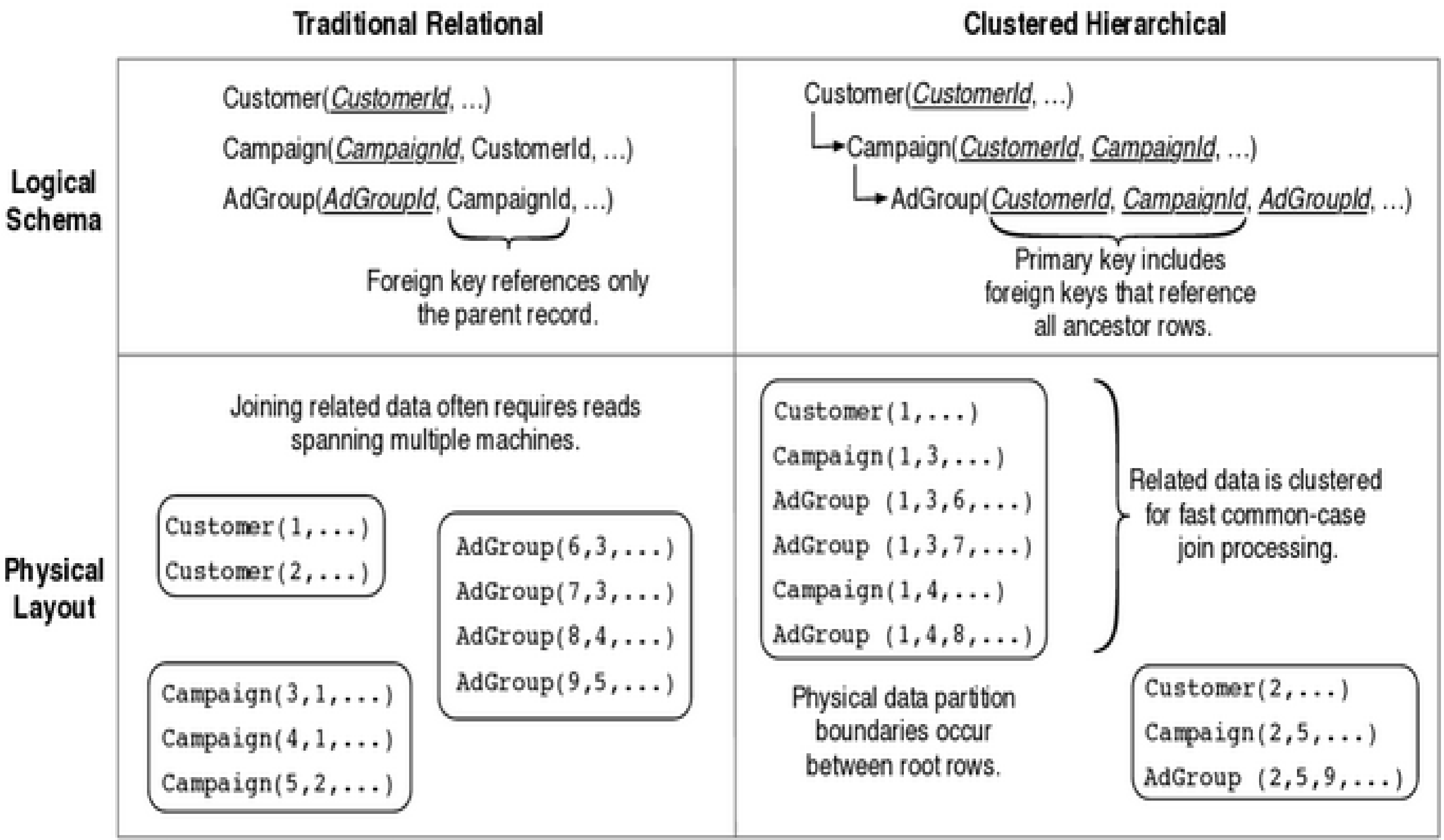
Google F1 and Spanner

- F1 is built on Spanner, which provides synchronous cross-data center replication and strong consistency.
  - Spanner: Google’s Globally-Distributed Database
  - Synchronous replication implies higher commit latency



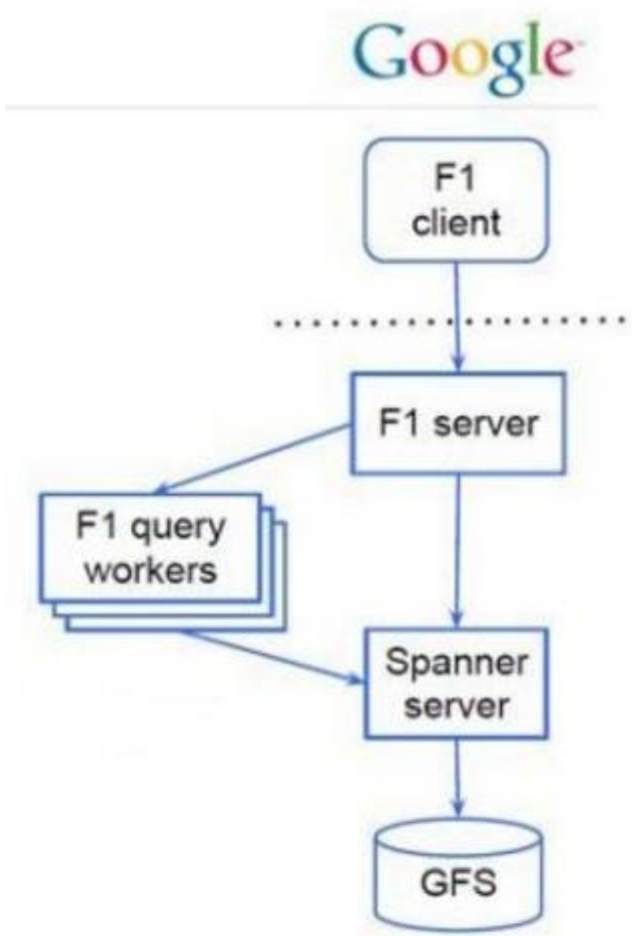
  - Spanner mitigates this latency by using a hierarchical schema model with structured data types and through smart application design.

F1's clustered hierarchical schema



Besides

- F1 also includes a fully functional distributed SQL query engine and automatic change tracking and publishing.



Google File System (GFS)

F1 supports three types of transactions

- Each F1 transaction consists of multiple reads, optionally followed by a single write that commits the transaction.
- F1 implements three types of transactions, all built on top of Spanner's transaction support
  - Snapshot transaction
  - Pessimistic transaction
  - Optimistic transaction

F1's Snapshot Transaction

- Read-only transaction with snapshot semantics
- Multiple client servers can see consistent views of the entire database at the same timestamp

F1’s Pessimistic Transaction

- The same as Spanner transactions
- Use a stateful communications protocol to require holding locks, so all requests in a single pessimistic transaction get directed to the same F1 server.
  - If the F1 server restarts, the pessimistic transaction aborts.
  - Reads in pessimistic transactions can request either shared or exclusive locks.

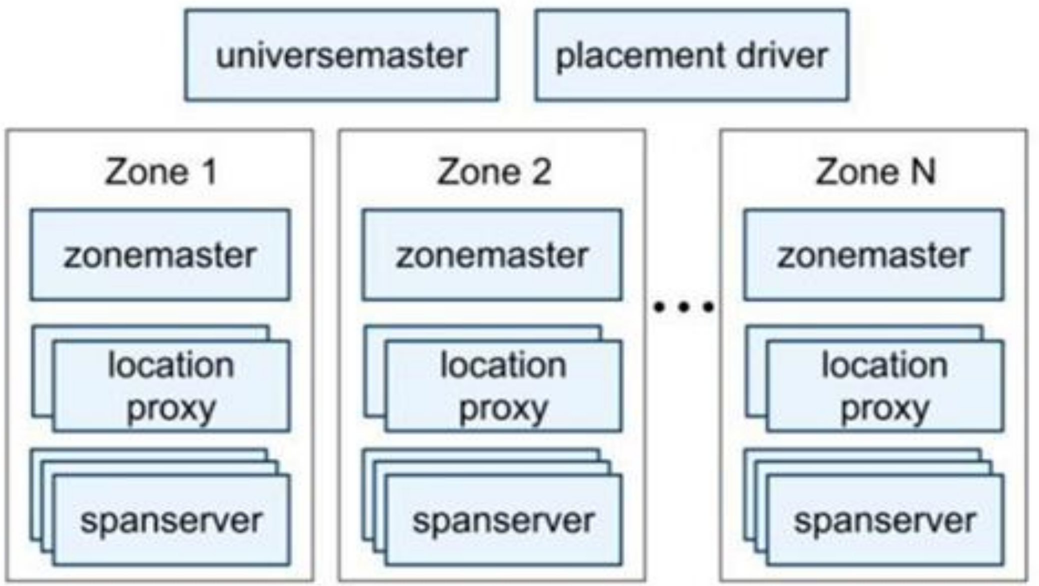
F1's Optimistic Transaction

- Consist of an arbitrarily long read phase, which never takes Spanner locks, and then a short write phase.
- To detect row-level conflicts, F1 returns with each row its last modification timestamp, which is stored in a hidden lock column in that row.
- The new commit timestamp is automatically written into the lock column whenever the corresponding data is updated (in either pessimistic or optimistic transactions).

F1's Optimistic Transaction (*cont.*)

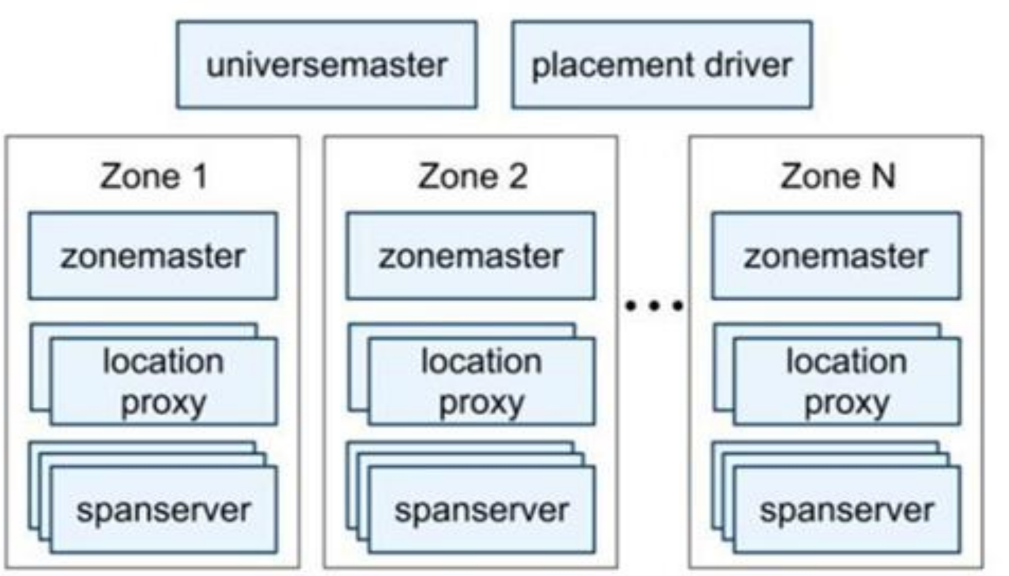
- The client library collects these timestamps, and passes them back to an F1 server with the write that commits the transaction.
- The F1 server creates a short-lived Spanner pessimistic transaction and re-reads the last modification timestamps for all read rows.
  - If any of the re-read timestamps differ from what was passed in by the client, there was a conflicting update, and F1 aborts the transaction;
  - Otherwise, F1 sends the writes on to Spanner to finish the commit.

Architecture of Spanner



- A zone has one zonemaster and hundreds of spanservers.
  - zonemaster assigns data to spanservers
  - Spanservers serve data to clients.
  - The per-zone location proxies are used by clients to locate the spanservers for the requested data.

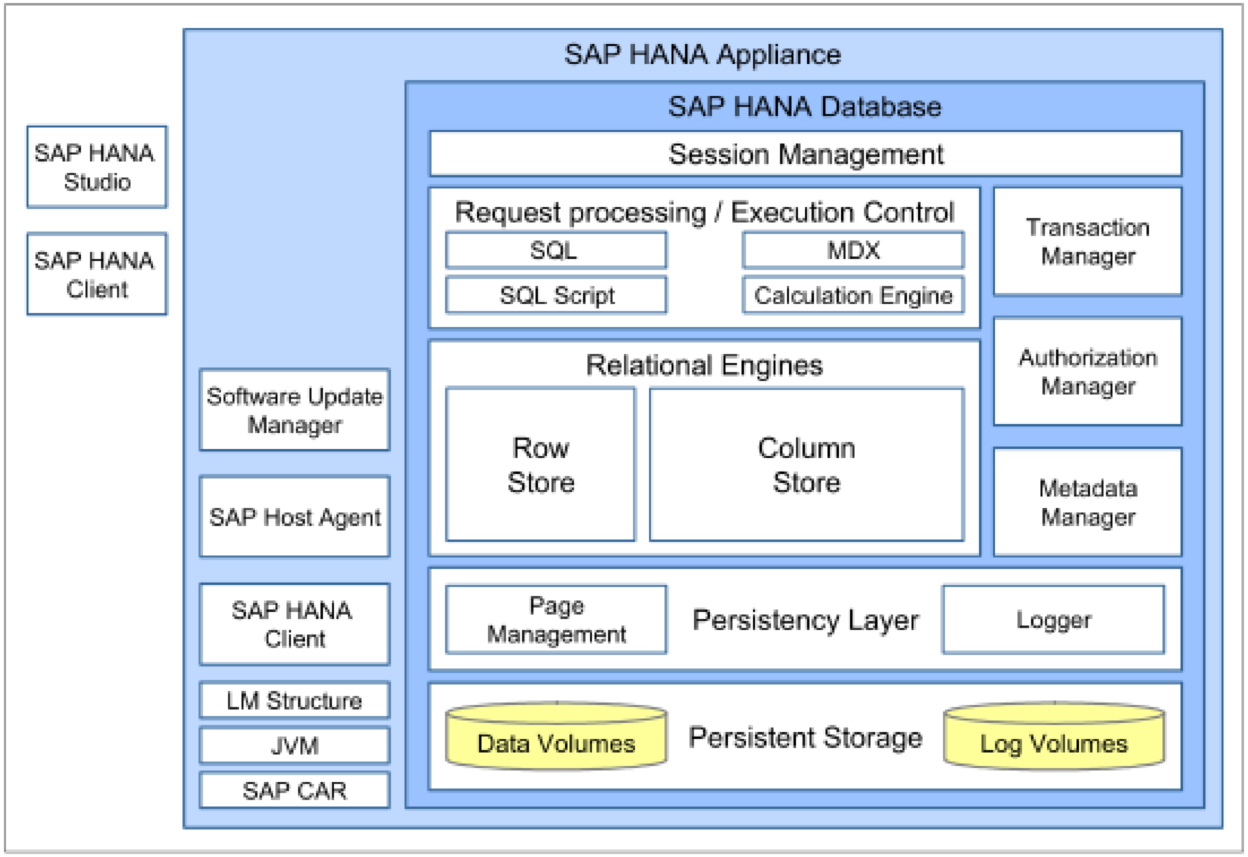
Architecture of Spanner (*cont.*)



- The universe master displays status information about all the zones for interactive debugging.
- The placement driver handles automated movement of data across zones on the timescale of minutes.
  - It periodically communicates with the spanservers to find data that needs to be moved, either to meet updated replication constraints or to balance load.

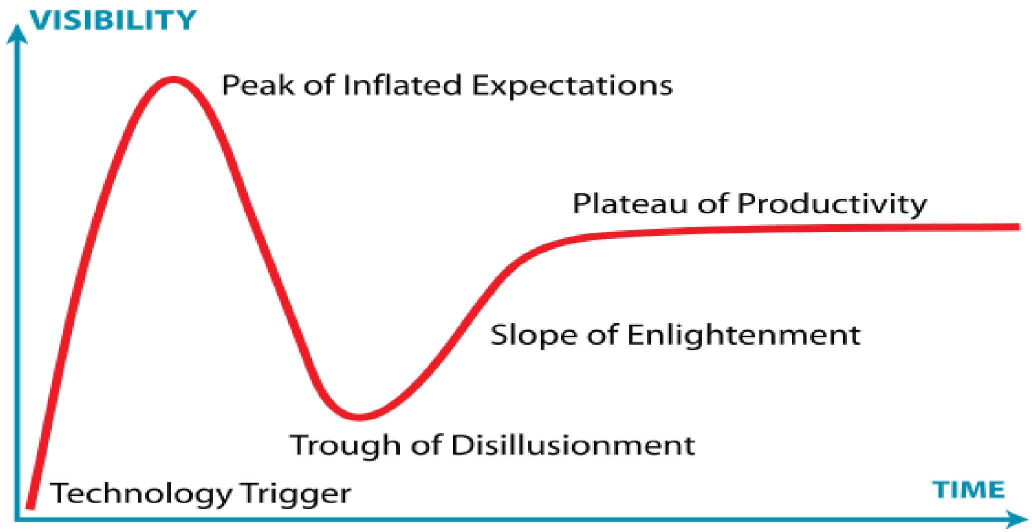
SAP Hana

- In-memory DB
- Combines row, column and object-oriented technology at the table level



Summary

- The most powerful technologies take a while to mature. But when they do, they can rapidly retire mainstays that are decades old.



Gartner Inc.'s **hype cycle**: a graphic representation of the maturity, adoption, and social application of specific technologies

Q & A