ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

Data Management in the Cloud

Tim Kraska

[Anology from IM 2/09 / Daniel Abadi]

Do you want milk?

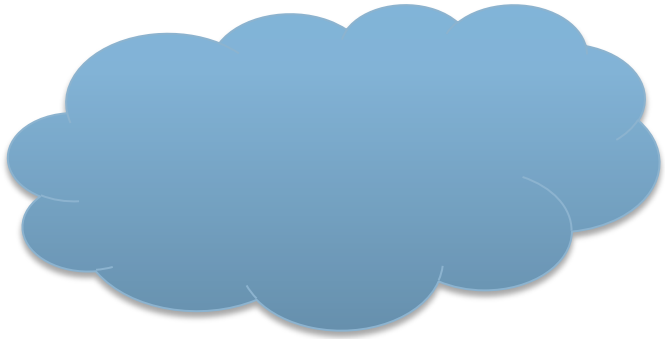
Buy a cow

- High upfront investment
- High maintenance cost
- Produces a fixed amount of milk
- Stepwise scaling

Buy bottled milk

- Pay-per-use
- Lower maintenance cost
- Linear scaling
- Fault-tolerant

Traditional Computing vs. Cloud Computing



Your computer is a cow

- High upfront investment
- High maintenance cost
- Fixed amount of resources
- Stepwise scaling

Cloud computing is bottled milk

- Pay-per-use
- Lower maintenance cost
- Linear scaling
- Fault-tolerant

Requirements for DM in the Cloud

- Scalability
  - response time independent of number of clients
- 100 percent read + write availability
  - no client is ever blocked under any circumstances
- Cost ($$$)
  - pay as you go along, no investment upfront
  - get cheaper every year, leverage new technology
- No administration
  - “outsource” patches, backups, fault tolerance

Consistency: Optimization goal, not constraint

Outline

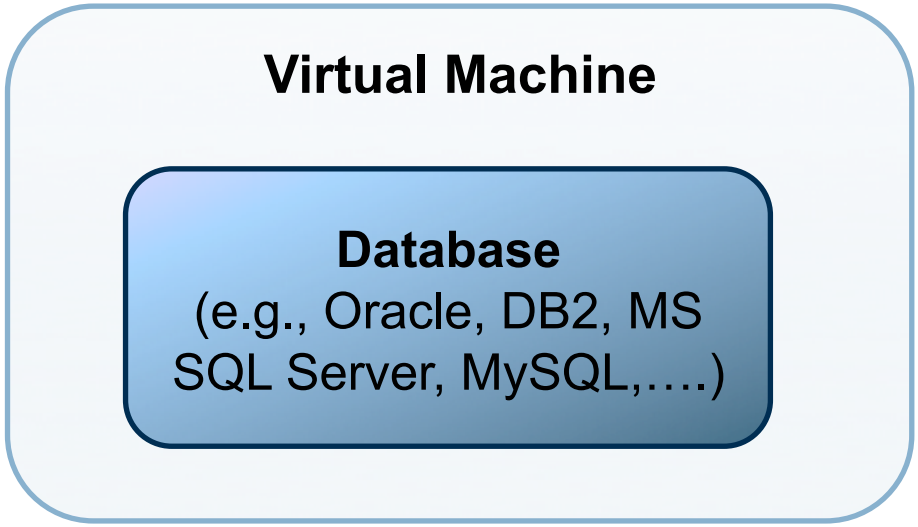
- Motivation
- Camp 1: Relational DB's in the Cloud
  - Amazon RDS
  - MS Azure
- Camp 2: New Cloud DB/Storage System
  - Amazon Dynamo
  - Google's MegaStore, BigTable, GFS
  - Building a DB applications without a DBMS
- Analytics in the Cloud: Hadoop
- What's next?

Why not use a standard relational DB?

- **Advantages**
  - Traditional databases are available
  - Proven to work well; many tools
  - People trained and confident with them

Camp 1: Install standard DBMS in the Cloud

- **Advantages**
  - Traditional databases are available
  - Proven to work well; many tools
  - People trained and confident with them

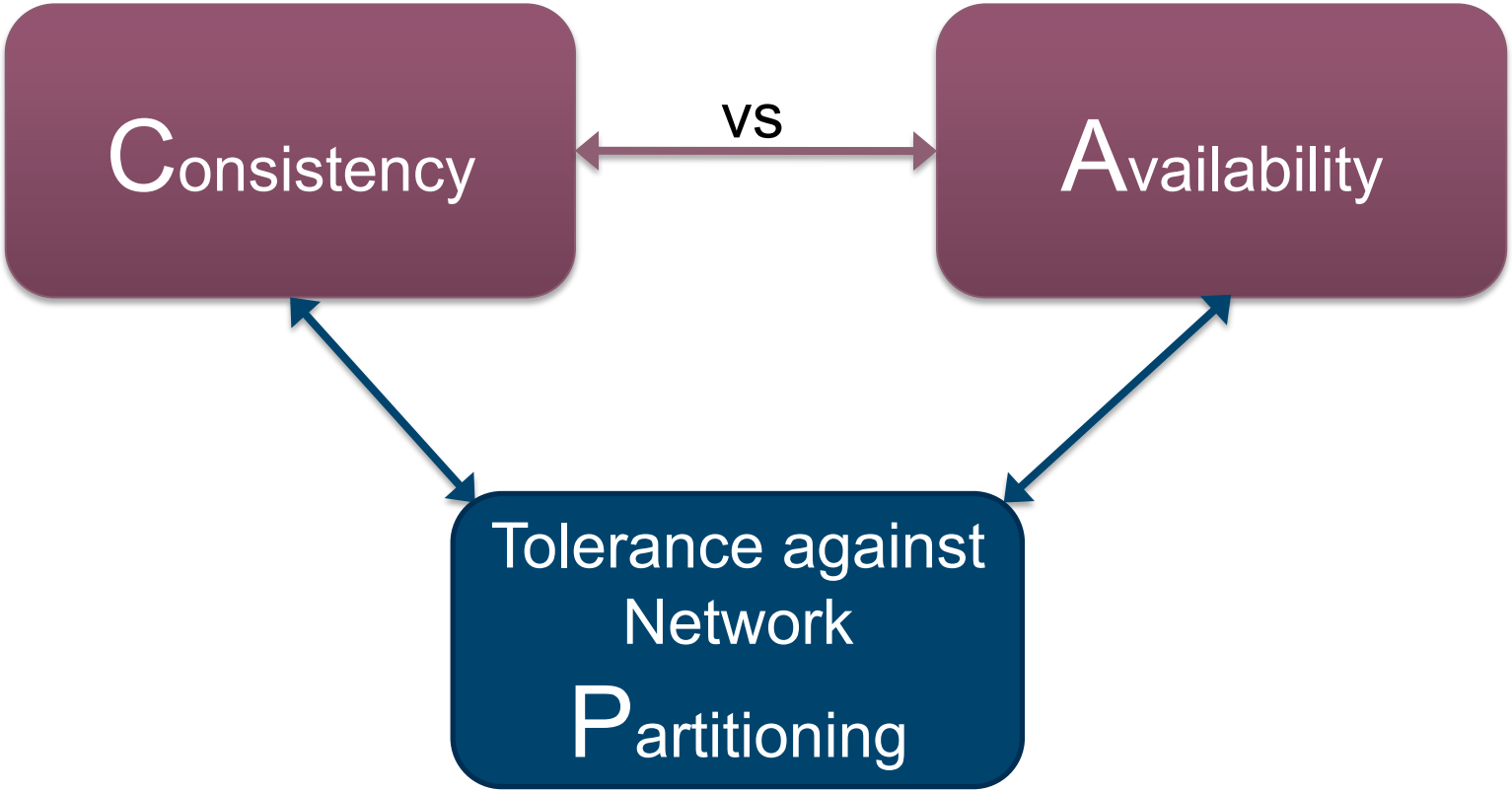


Camp 1: Install standard DBMS in the Cloud

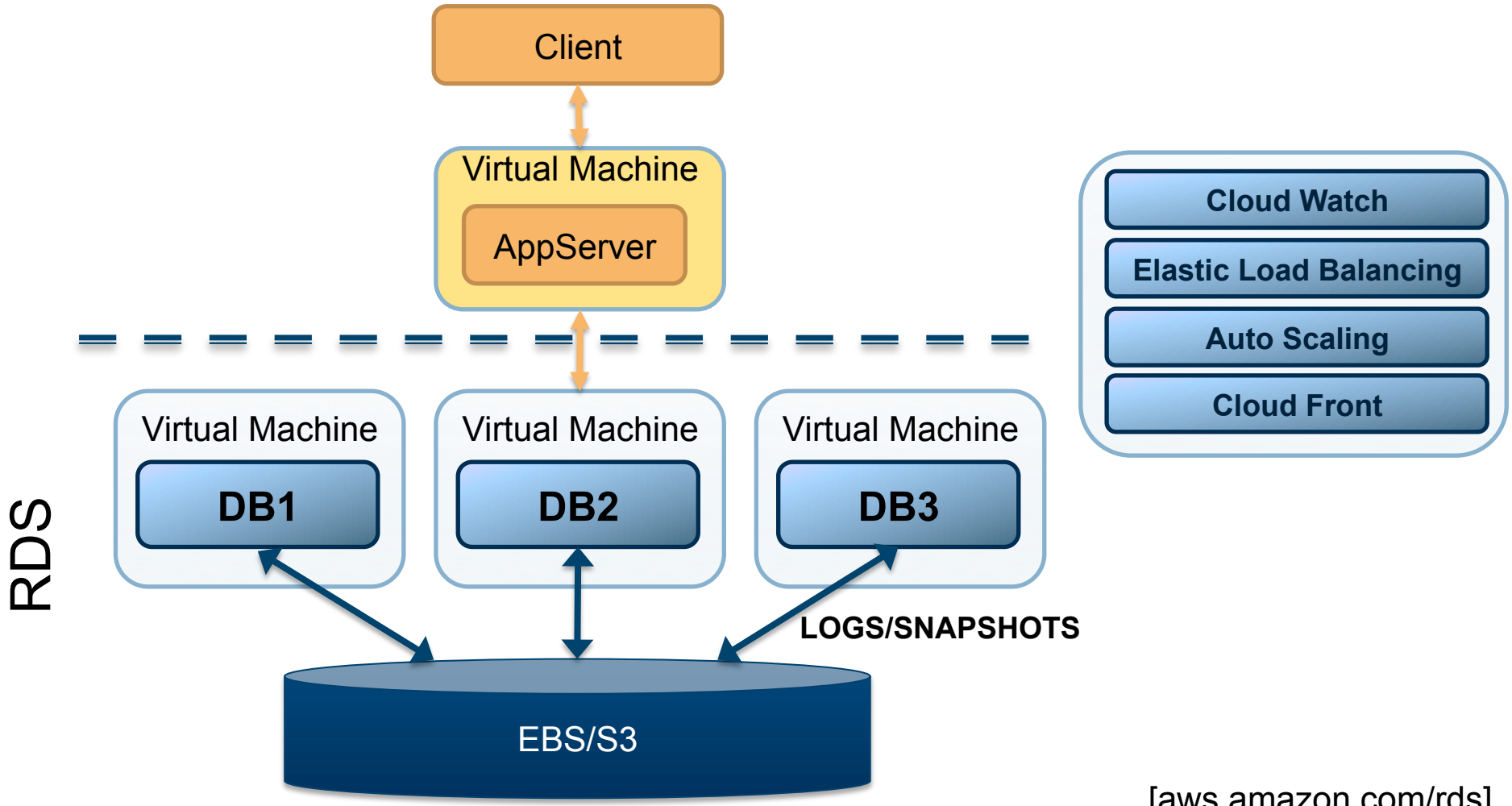
- **Advantages**
  - Traditional databases are available
  - Proven to work well; many tools
  - People trained and confident with them
- **Disadvantages**
  - Build for scale-up, not scale-out
  - Traditional DBMS solve the wrong problem anyway
    - Focus on throughput and consistency
    - SAP and Oracle misuse the DBMS already today
  - Traditional DBMS make the wrong assumptions
    - e.g., DBMS optimizers fail on virtualized hardware

vs

CAP-Theorem

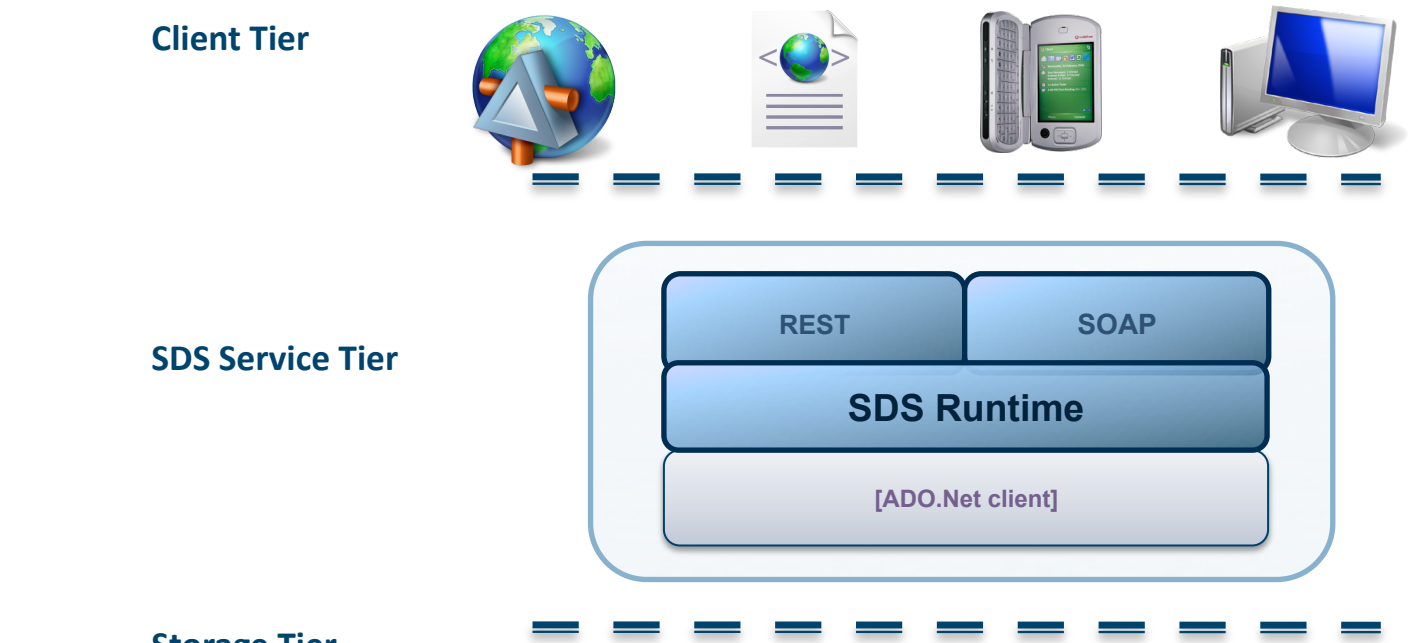

Consistency
vs
Availability
Tolerance against Network Partitioning

Amazon Relational Database Service (RDS)



[aws.amazon.com/rds]

MS Azure - Architecture



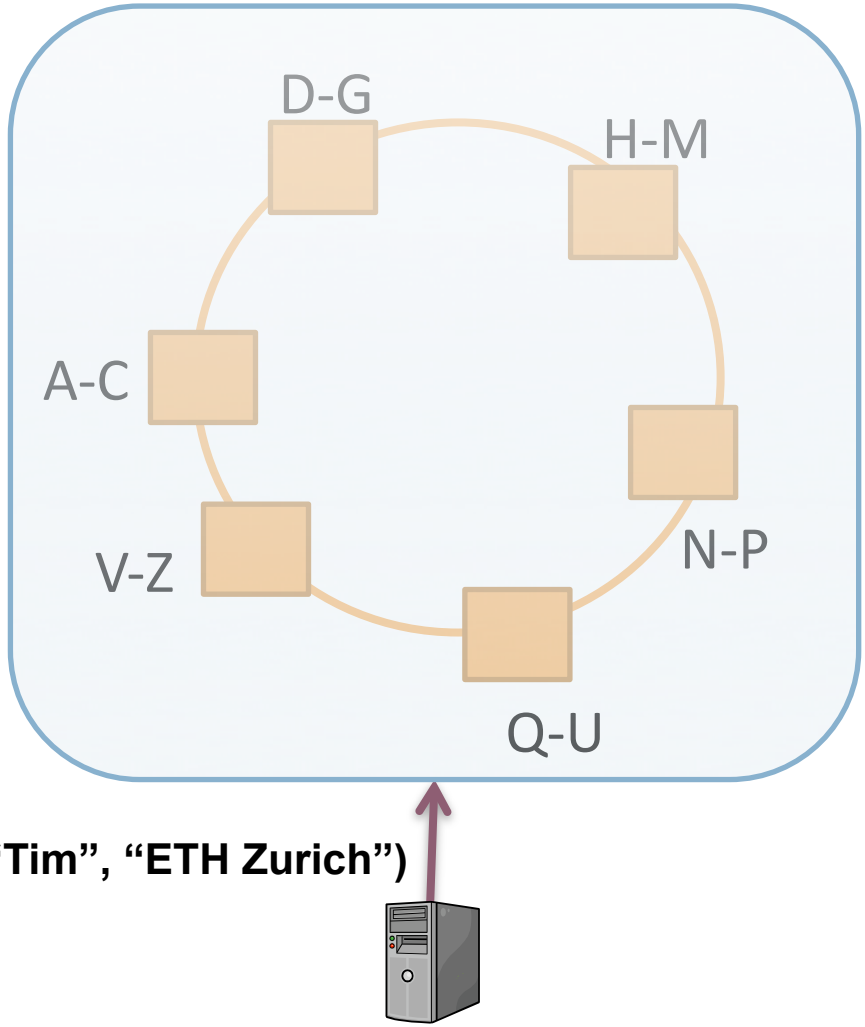
[www.microsoft.com/windowsazure/sqlazure]

Camp 2: Rethink the whole system architecture

- Rethink the whole system architecture
  - Do NOT use a traditional DBMS
  - Build a new systems tailored for cost efficiency, availability, and scalability and see consistency as an optimization goal

- Advantages and Disadvantages
  - Requires new breed of (immature) systems + tools
  - Solves the right problem and gets it right
  - Optimized for cost, availability, scalability,...
  - Leverages organization‘s investments in SOA

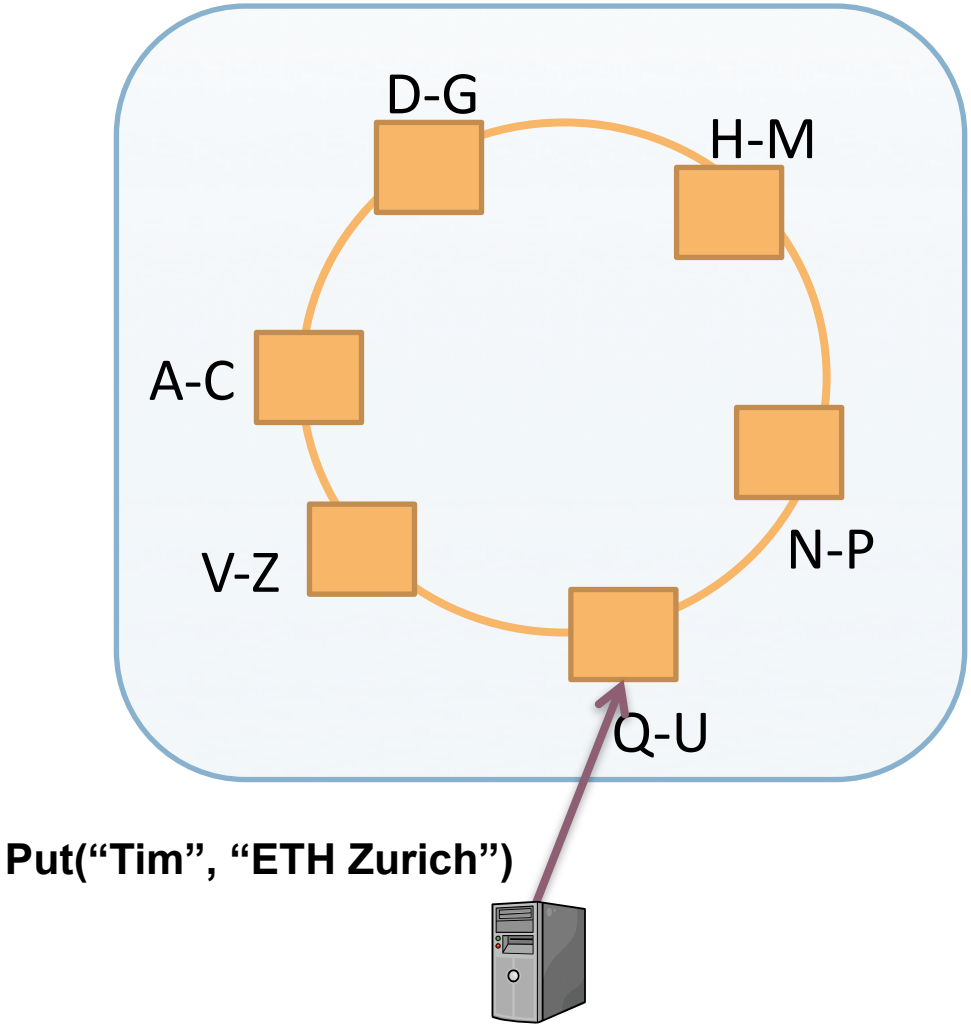
Key/Value-Store: Example Dynamo



- Data model: **Key->Value**
- Operations
  - **Put(Key, Value)**
  - **Get(Key)**
  - **NO-SQL!!!!**

[SOSP 07]

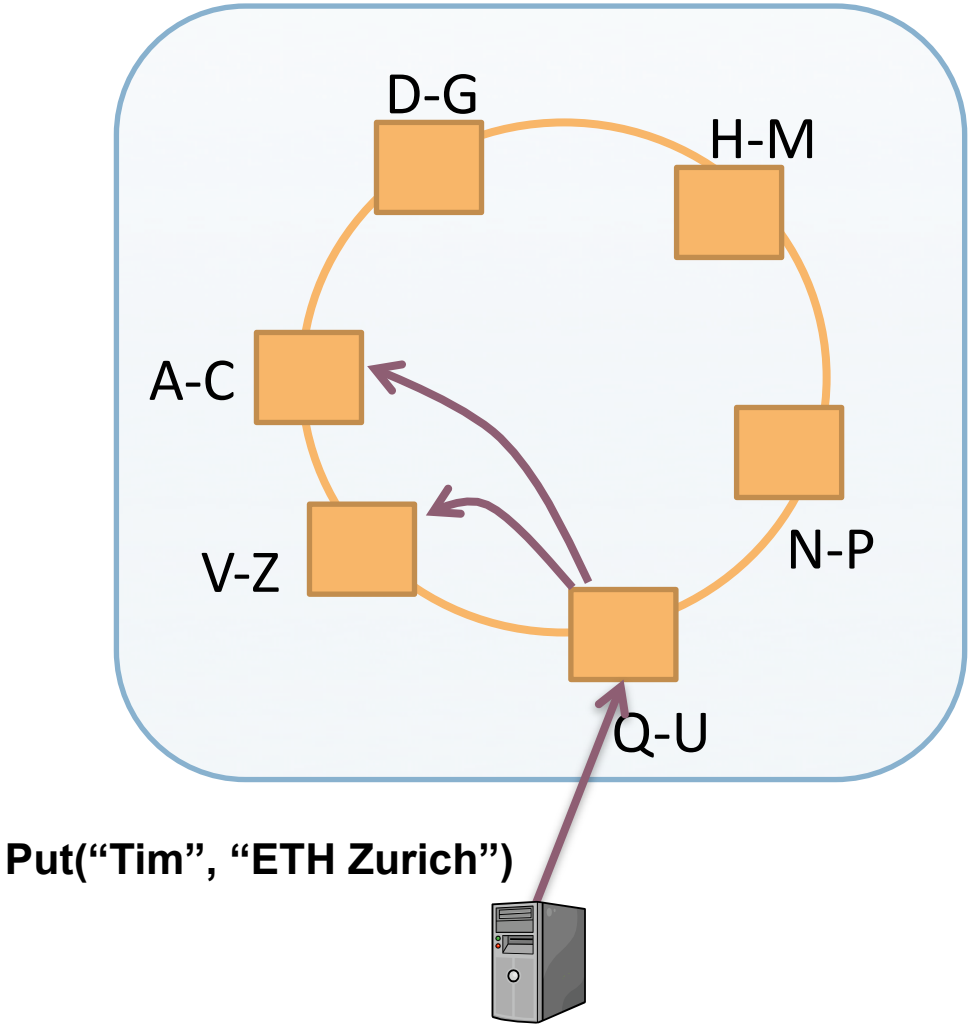
Key/Value-Store: Example Dynamo



- Data model: Key->Value
- Operations
  - Put(Key, Value)
  - Get(Key)
- Routing
  - Consistent Hashing
  - Partitioned Hash Table
- High Availability
  - Replication
  - Sloppy Quorum and hinted hand-over
  - Gossiping (Membership)
- Consistency
  - Vector Clocks
  - Merkle Trees (Anti-Entropy)

[SOSP 07]

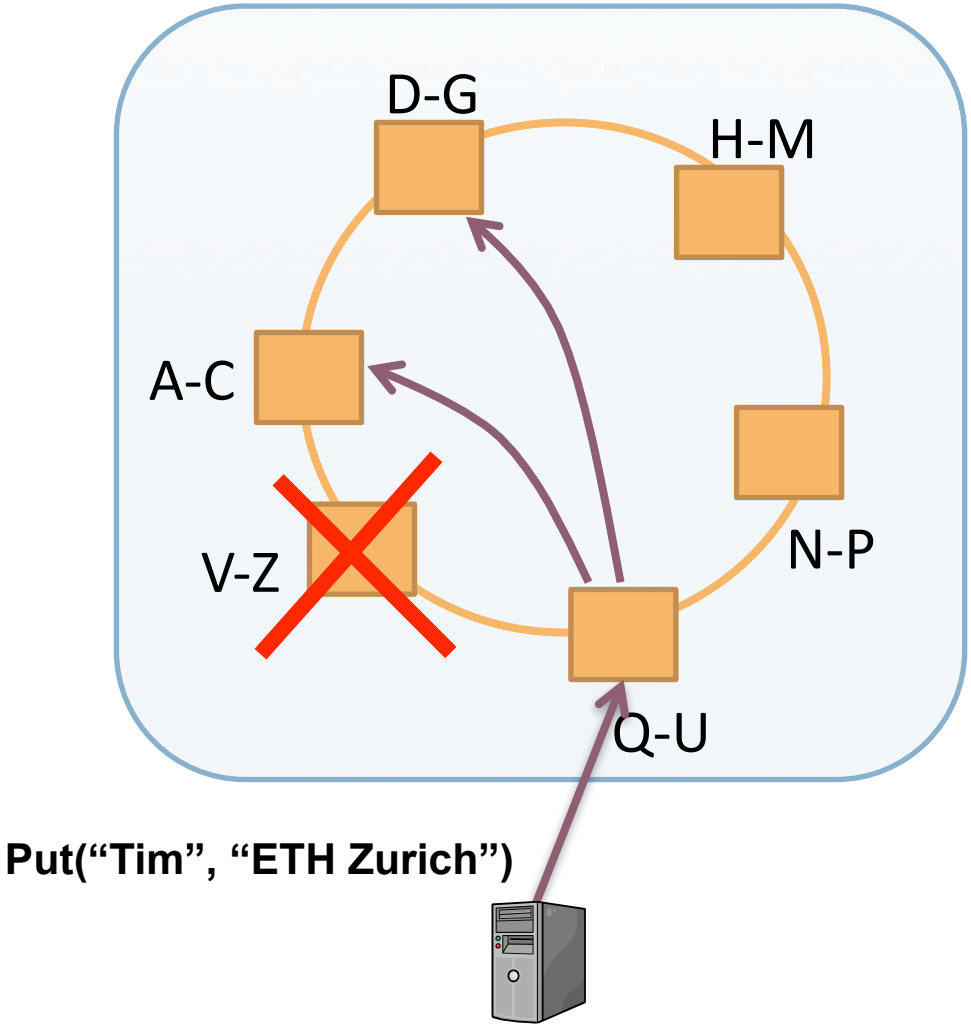
Key/Value-Store: Example Dynamo



- Data model: Key->Value
- Operations
  - Put(Key, Value)
  - Get(Key)
- Routing
  - Consistent Hashing
  - Partitioned Hash Table
- High Availability
  - Replication
  - Sloppy Quorum and hinted hand-over
  - Gossiping (Membership)
- Consistency
  - Vector Clocks
  - Merkle Trees (Anti-Entropy)

[SOSP 07]

Key/Value-Store: Example Dynamo



- Data model: Key->Value
- Operations
  - Put(Key, Value)
  - Get(Key)
- Routing
  - Consistent Hashing
  - Partitioned Hash Table
- High Availability
  - Replication
  - Sloppy Quorum and hinted hand-over
  - Gossiping (Membership)
- Consistency
  - Vector Clocks
  - Merkle Trees (Anti-Entropy)

[SOSP 07]

Google's Approach

Chubby
(Locking Service)

Scheduling

MegaStore
Transactions, Indexes, GQL

BigTable
Distributed Multi-Dimensional Map

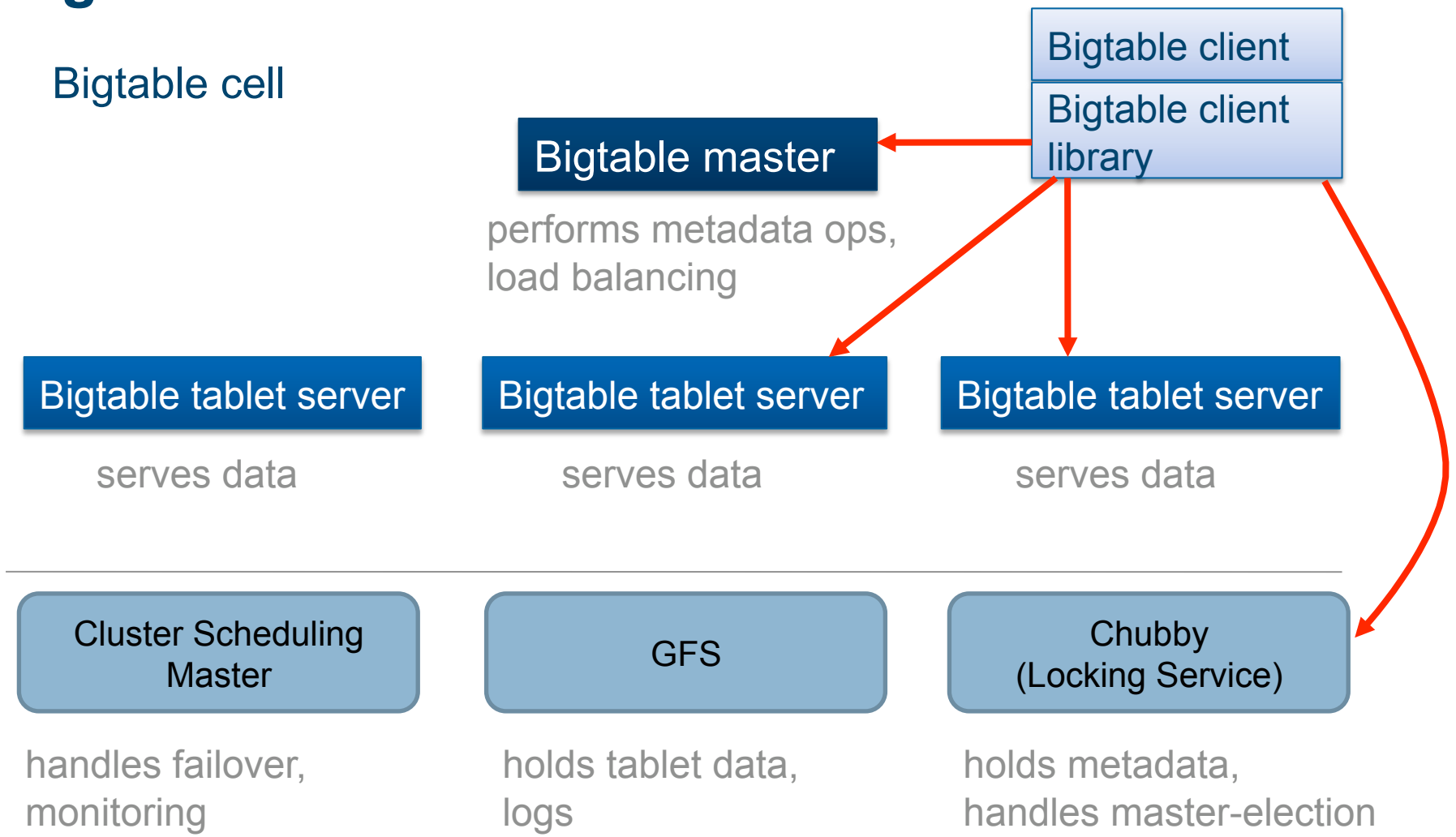
Google File System (GFS)
Distributed Storage (made for append-only)

BigTable

- Distributed multi-level map (again no SQL)
- 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

[SOSP 03]

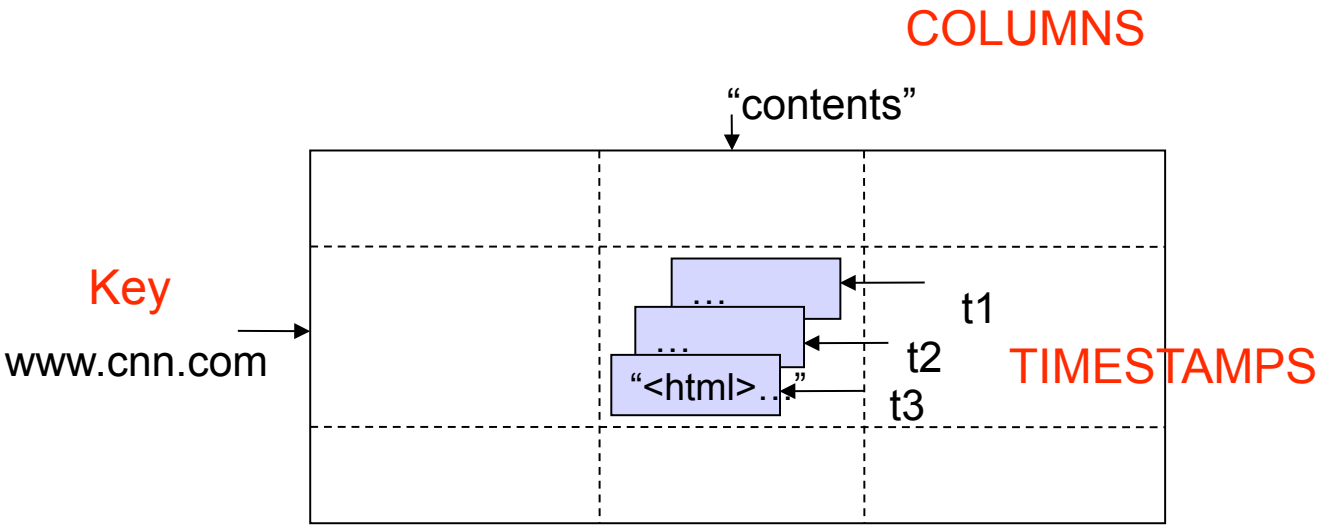
BigTable Architecture



[SOSP 03]

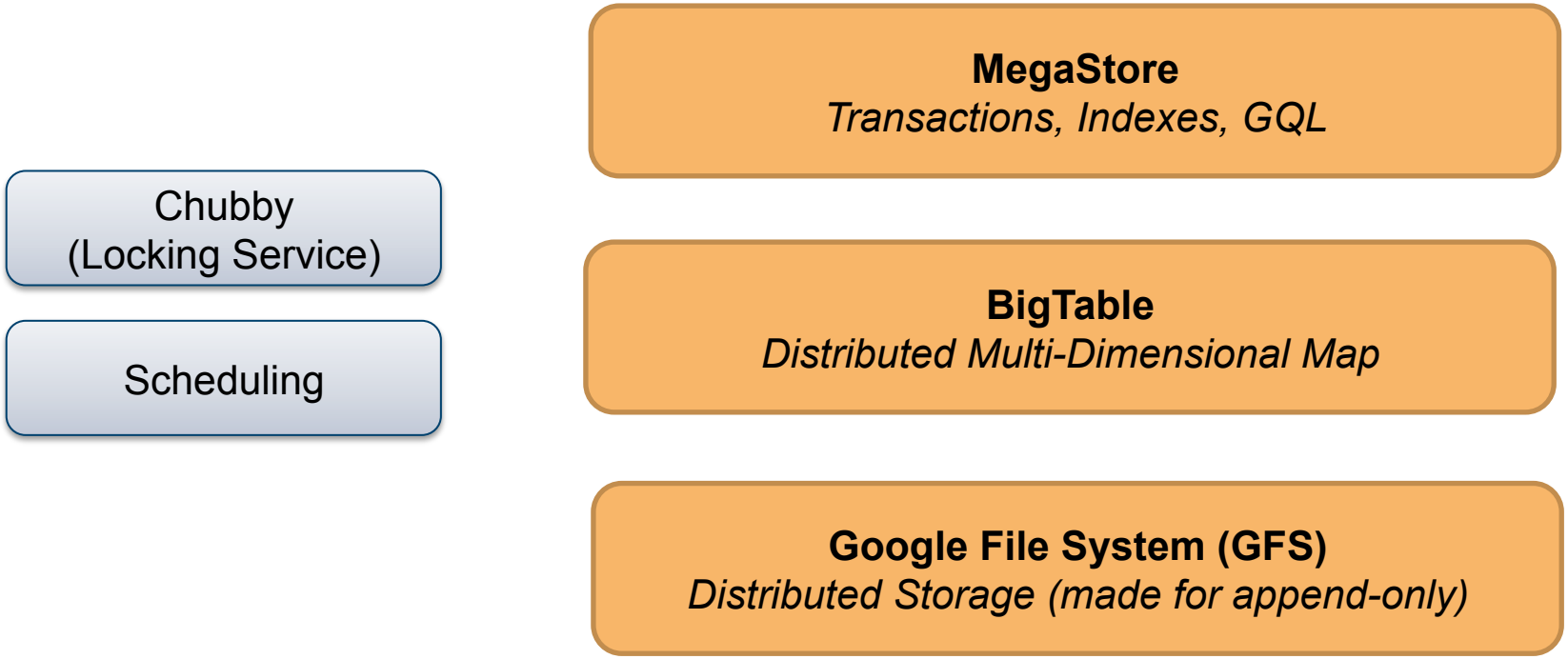
BigTable DataModel

- **Distributed** multi-dimensional sparse map
  (key, column, timestamp) → cell contents

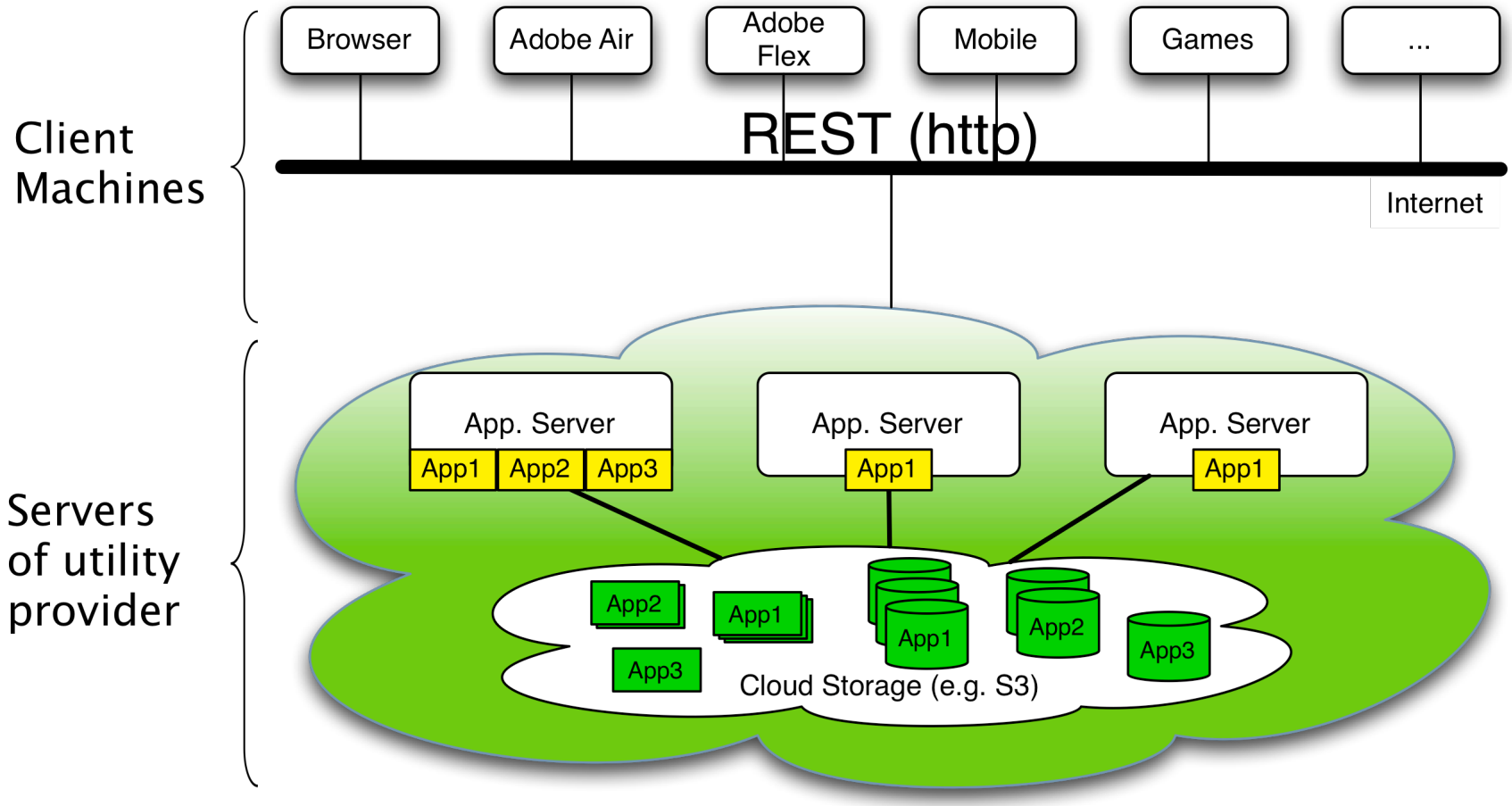


[SOSP 03]

Google's Approach



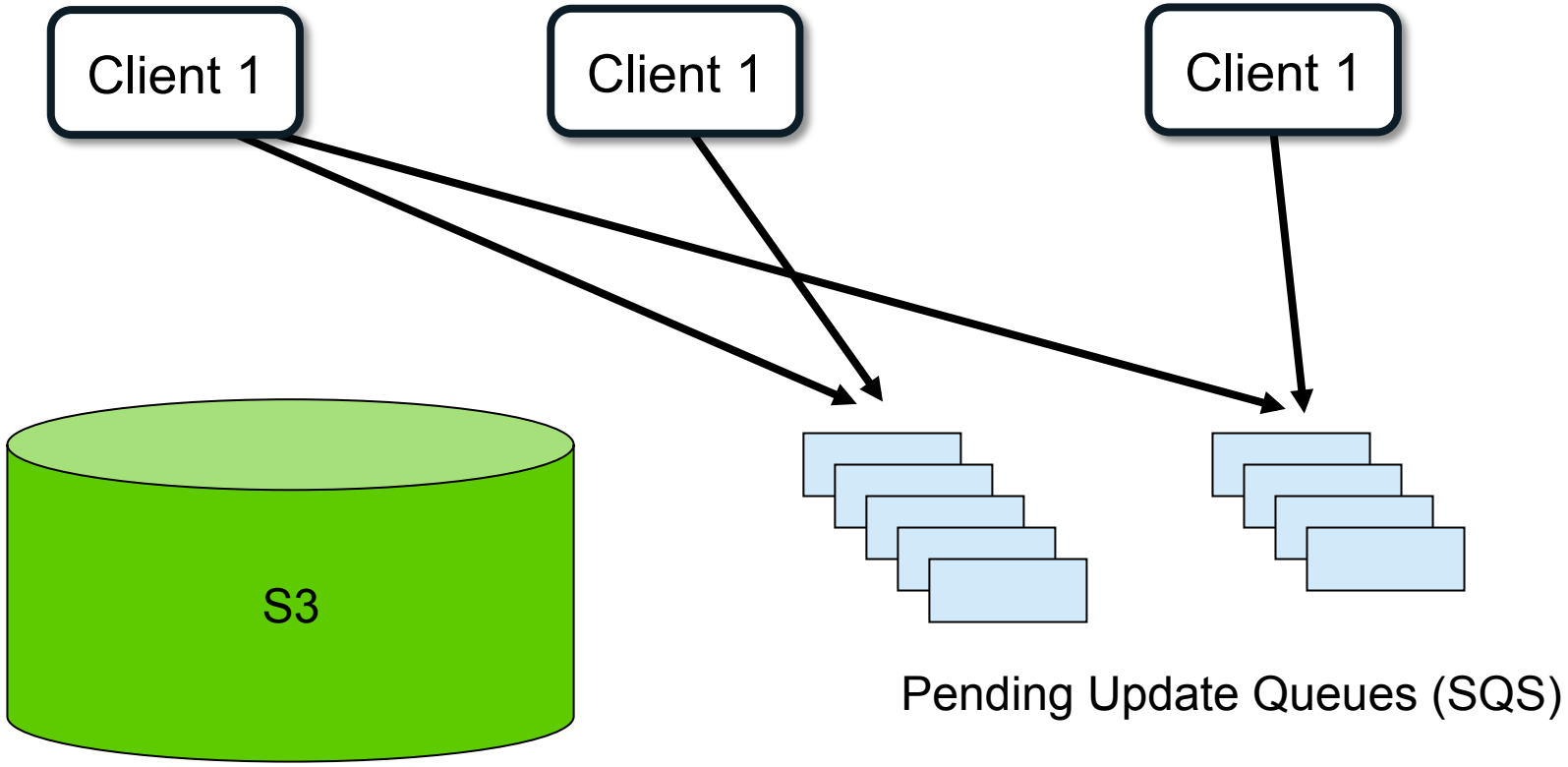
Bulding DB applications without a DBMS



[SIGMOD 08]

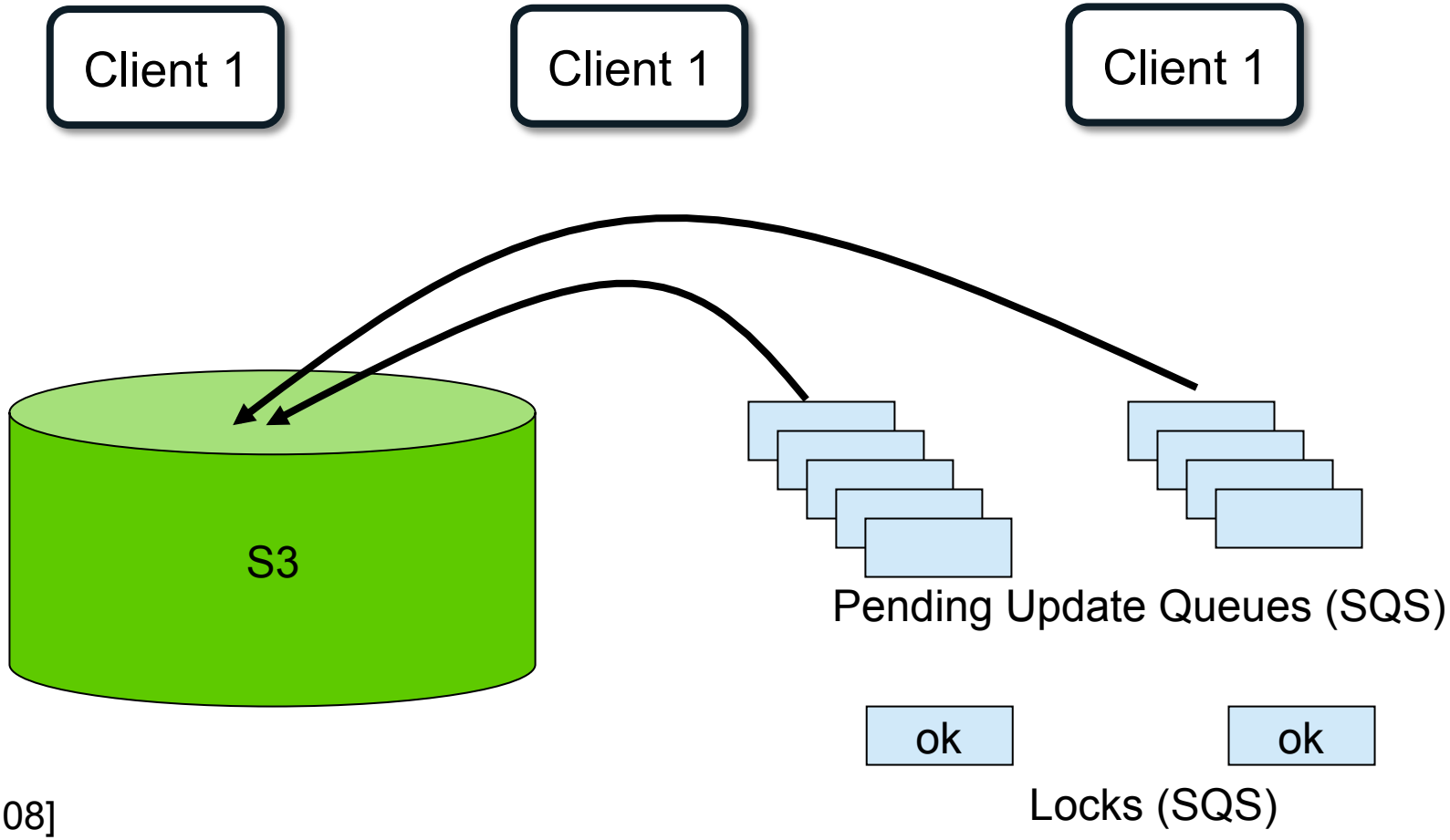
Concept commercialized in Sausalito (28msec, Inc.)

Step 1: Clients commit update records to pending update queues



[SIGMOD 08]

Step 2: Checkpointing propagates updates from SQS to S3



[SIGMOD 08]

Cloud Storage/DB Systems

- Commercial
  - Azure SQL, Store
  - Google MegaStore, BigTable
  - Amazon S3, SimpleDB, RDS, CloudFront
  - 28msec Sausalito
  - …
- OpenSource:
  - HBase (≈ *BigTable)*
  - CouchDB/Scalaris (≈ *Dynamo)*
  - Cassandra (≈ *Dynamo* + *BigTable Data-Model)*
  - Redis
  - Cloudy (ETHZ)
  - SCADS (Berkeley)
  - …

Outline

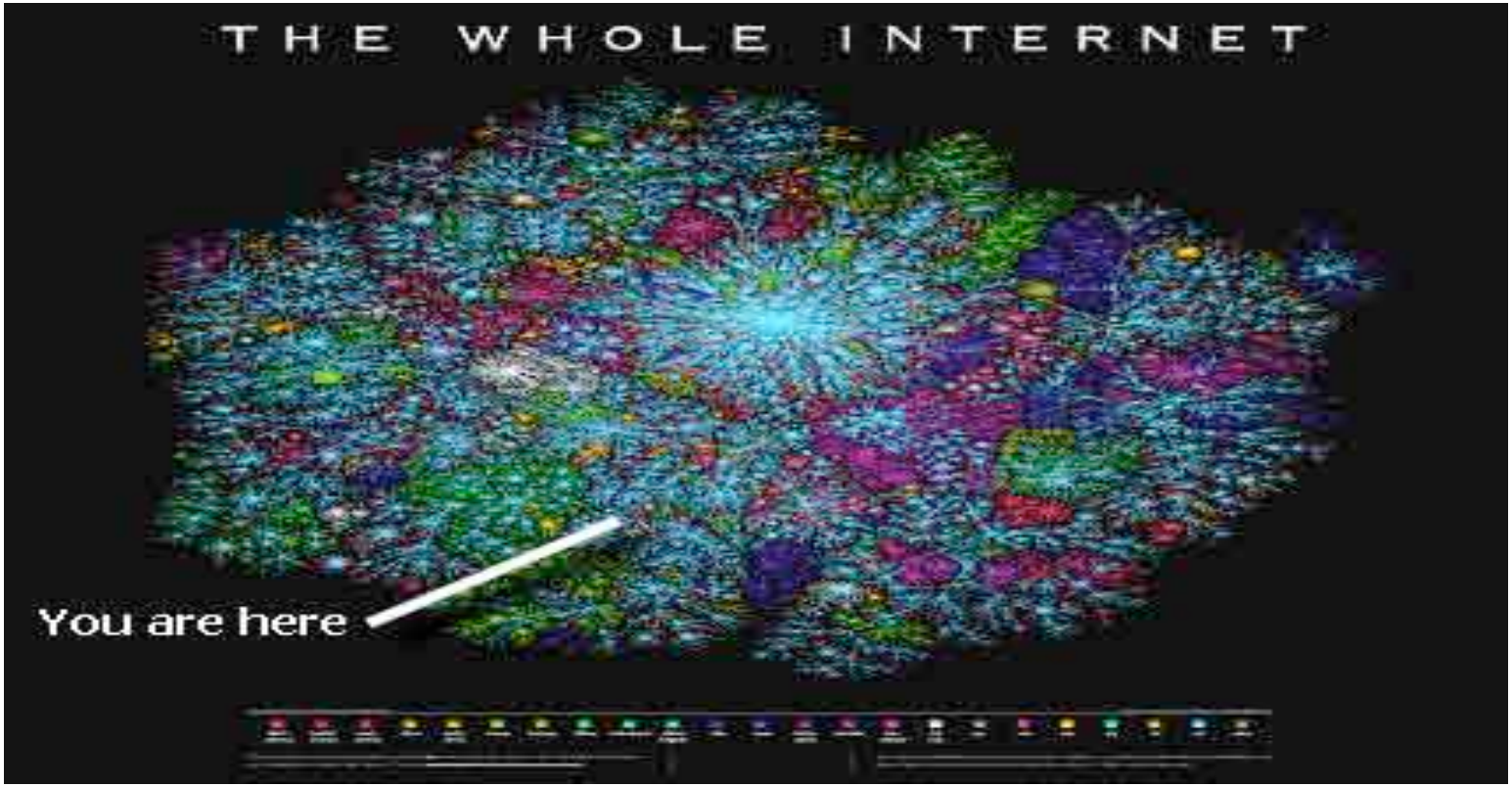
- Motivation
- Camp 1: Relational DB's in the Cloud
  - Amazon RDS
  - MS Azure
- Camp 2: New Cloud DB/Storage System
  - Amazon Dynamo
  - Google's MegaStore, BigTable, GFS
  - Building a DB applications without a DBMS
- Analytics in the Cloud: Hadoop
- What's next?

Why MapReduce?

Big Data

Potential Applications

- Web data analysis applications
  - Example: Google



Google: The Data Challenge

- Jeffrey Dean, Google Fellow, PACT'06 keynote speech:
  - 20+ billion web pages x 20KB = 400 TB
  - One computer can read 30-35 MB/sec from disk
    - ~ **4 months** to read the web
  - ~ 1,000 hard drives just to store the web
  - Even more to "do" something with the data
  - But: Same problem with 1,000 machines < **3 hours**

- MapReduce CACM'08 article:
  - 100,000 MapReduce jobs executed in Google every day
  - Total data processed > **20 PB of data per day**

Map/Reduce in a Nutshell

- Given:
  - a very large dataset
  - a well-defined computation task to be performed on elements of this dataset (preferably, in a parallel fashion on a large cluster)
- MapReduce programming model/abstraction/ framework:
  - Just express what you want to compute (map() & reduce())
  - Don’t worry about parallelization, fault tolerance, data distribution, load balancing (MapReduce takes care of these)
  - What changes from one application to another is the actual computation; the programming structure stays similar

[OSDI 04]

Map/Reduce in a Nutshell

- Here is the framework in simple terms:
  - Read lots of data
  - **Map**: extract something that you care about from each record (similar to map in functional programming)
  - Shuffle and sort
  - **Reduce**: aggregate, summarize, filter, or transform (similar to fold in functional programming)
  - Write the results
- One can use as many Maps and Reduces as required to model a given problem

[OSDI 04]

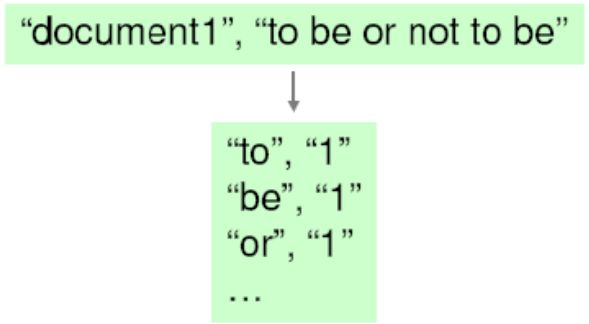
Map/Reduce Example: Count Word Occurrences in a Document

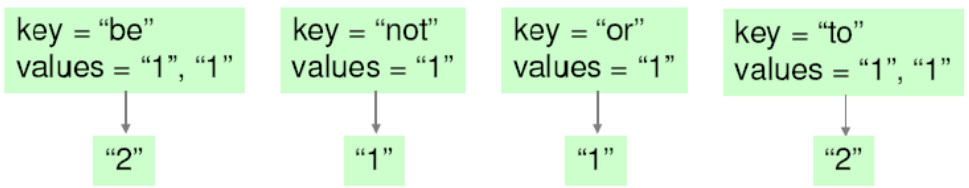
map(k1, v1) → list(k2, v2)

reduce(k2, list(v2)) → list(v2)

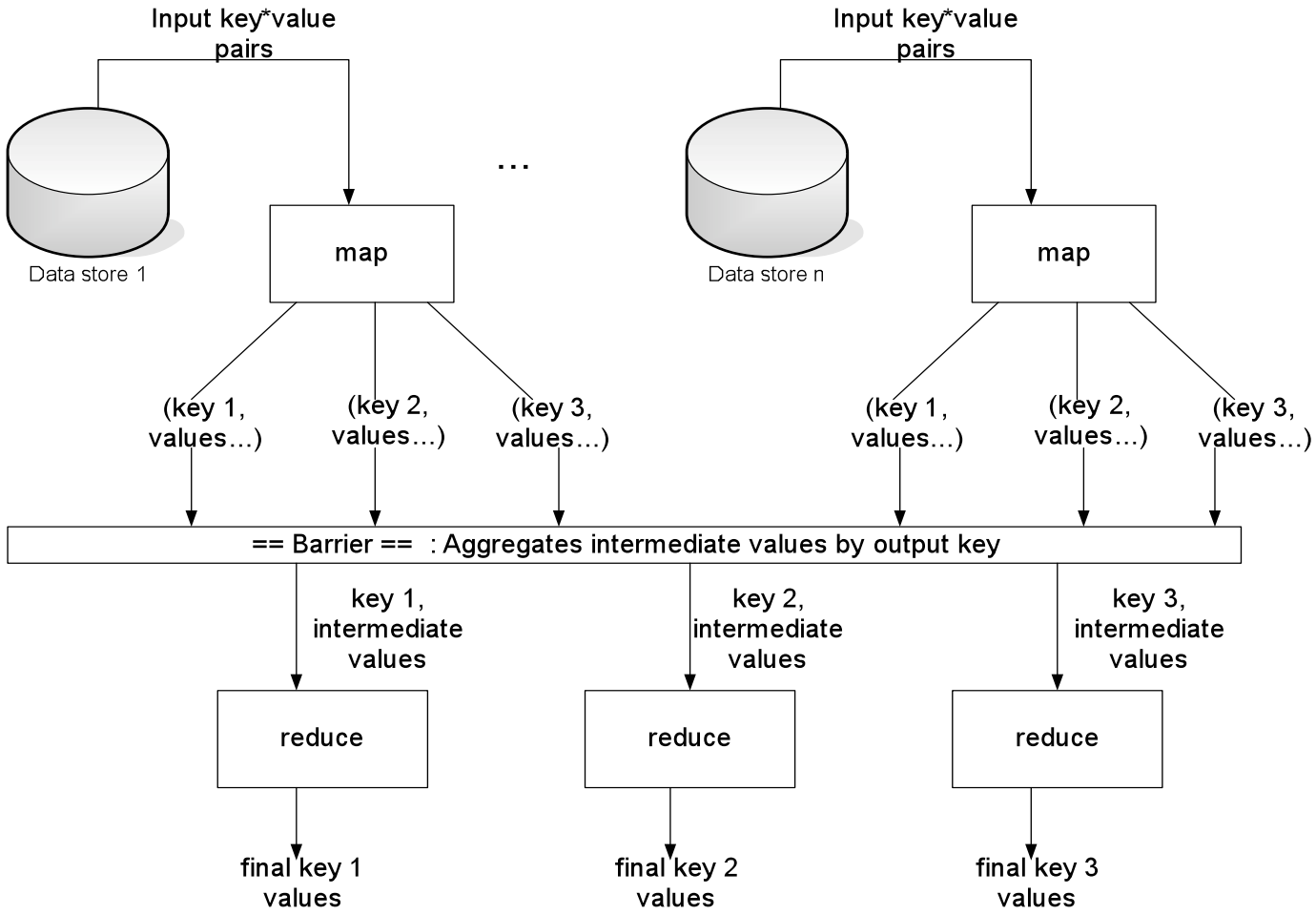
```
map (String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");
```

```
reduce (String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```



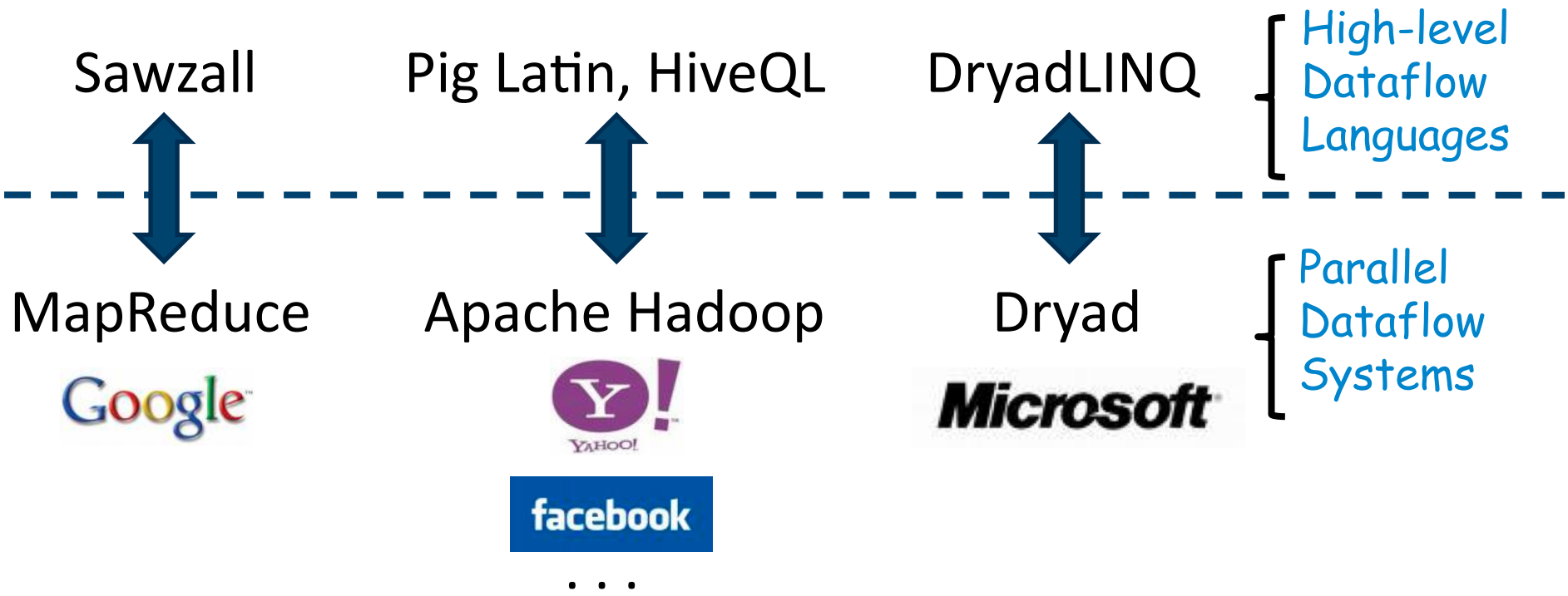


Google's MapReduce Model



[OSDI 04]

Parallel Dataflow Systems and Languages



- A high-level language provides:
  - more transparent program structure
  - easier program development and maintenance
  - automatic optimization opportunities

Outline

- Motivation
- Camp 1: Relational DB's in the Cloud
  - Amazon RDS
  - MS Azure
- Camp 2: New Cloud DB/Storage System
  - Amazon Dynamo
  - Google's MegaStore, BigTable, GFS
  - Building a DB applications without a DBMS
- Analytics in the Cloud: Hadoop
- What's next?

What's next???

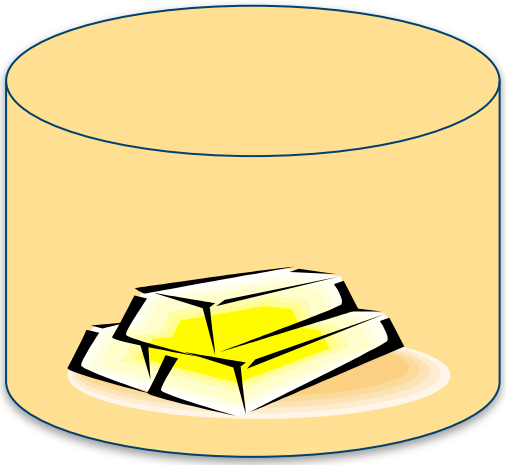
- Consistency (CAP Theorem)
  - How to balance consistency vs. cost vs. availability?
  - Does consistency limit the scalability?
  - How to scale transaction processing?
- Data model & languages
  - Is Key/Value really what we need?
  - Why do I need to implement a join by myself???
  - E.g. BOOM
- Other workload patterns
  - Streaming
  - Scientific
  - Data Mining
  - …
- Cost optimizations
  - New optimization goal: Cost (not performance)
  - SLA imply implicit cost
- Benchmarks
- Security (don't get me started)
- ….

What's next???

- Consistency (CAP Theorem)
  - **How to balance consistency vs. cost vs. availability?**
  - Does consistency limit the scalability?
  - How to scale transaction processing?
- Data model & languages
  - Is Key/Value really what we need?
  - Why do I need to implement a join by myself???
  - E.g. BOOM
- Other workload patterns
  - **Streaming**
  - Scientific
  - Data Mining
  - …
- Cost optimizations
  - New optimization goal: Cost (not performance)
  - SLA imply implicit cost
- Benchmarks
- Security (don't get me started)
- ….

Consistency Rationing

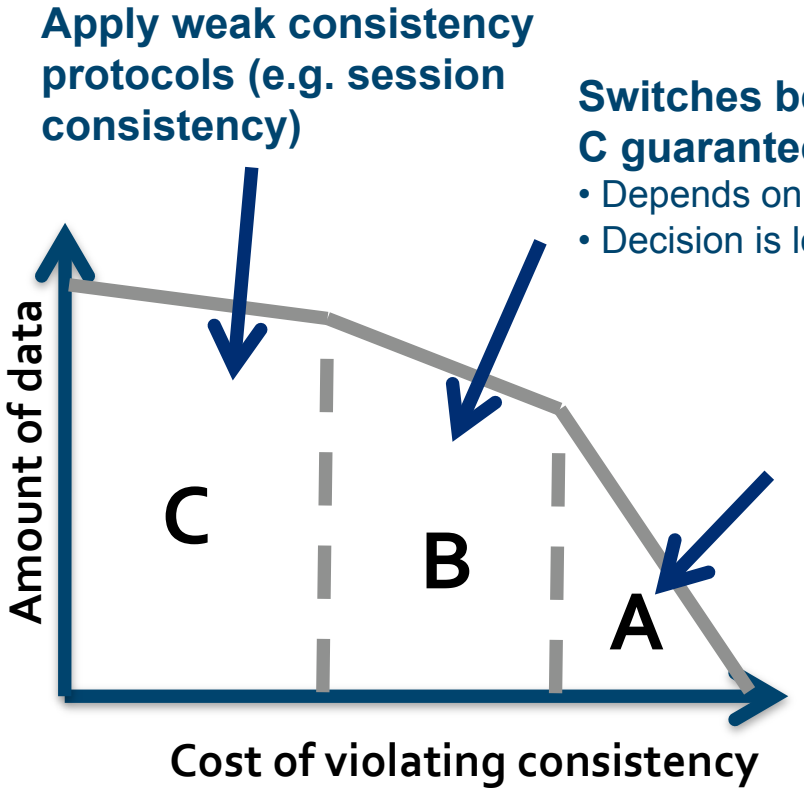
- ACID prevents scaling & availability (CAP theorem)!
- Strong consistency is expensive
- But not everything is worth gold!



- Idea: Handle data according to the cost of inconsistency
- Violating consistency is OK as long as it helps to reduce the overall cost

[VLDB 09]

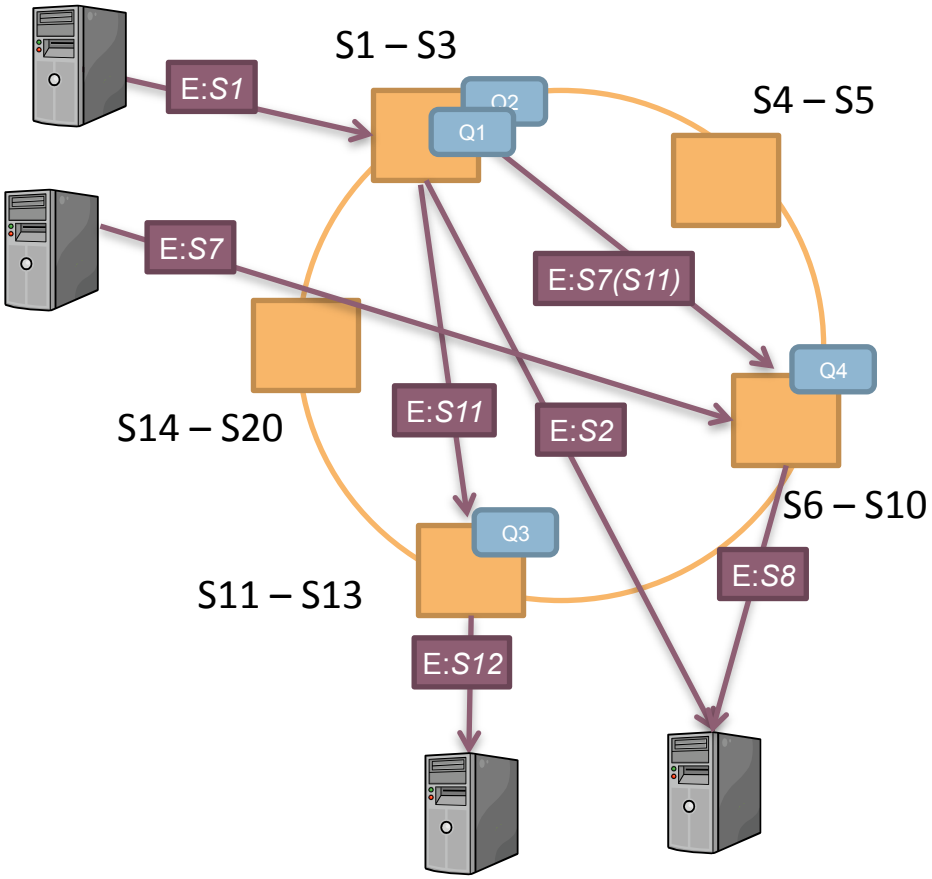
Consistency Rationing - Guarantees



- Consistency requirements per category instead of transaction level
- Different categories can mix in a single transaction

[VLDB 09]

Streaming in the Cloud - Smoky



Provide streaming as a service: No installation, maintenance etc.

- Use Cases
  - Server/workflow monitoring
  - message transformations
  - combining RSS streams
  - ...
- Leverage established cloud techniques
- Idea: **Storage ring: Key-> Data**

Stream ring: Event-type ->Query

Other (selected) ETH projects

- CloudBench
  - A benchmark for the cloud
- Barrelfish
  - new written-from-scratch OS kernel
  - targeted for multi- and many-core processor systems
- Rhizoma
  - constraint-based runtime system for distributed applications
  - self-hosting
- Concierge
  - blur the burden between mobile and cloud applications
- Xadoop
  - XQuery on Hadoop

Summary

- Cloud has changed the view on data management
  - Focus on scalability, availability and cost
  - Instead of consistency and performance at any price
  - E.g. No-SQL Movement
- As a result many Cloud Storage/DB Systems appeared on the marked place
- Although (or because) the cloud is economical driven, it provides many interesting research challenges

tim.kraska@inf.ethz.ch