Data Serving in the Cloud

Raghu Ramakrishnan
Chief Scientist, Audience and Cloud Computing

Brian Cooper
Adam Silberstein
Utkarsh Srivastava

Yahoo! Research

Joint work with the Sherpa team in Cloud Computing

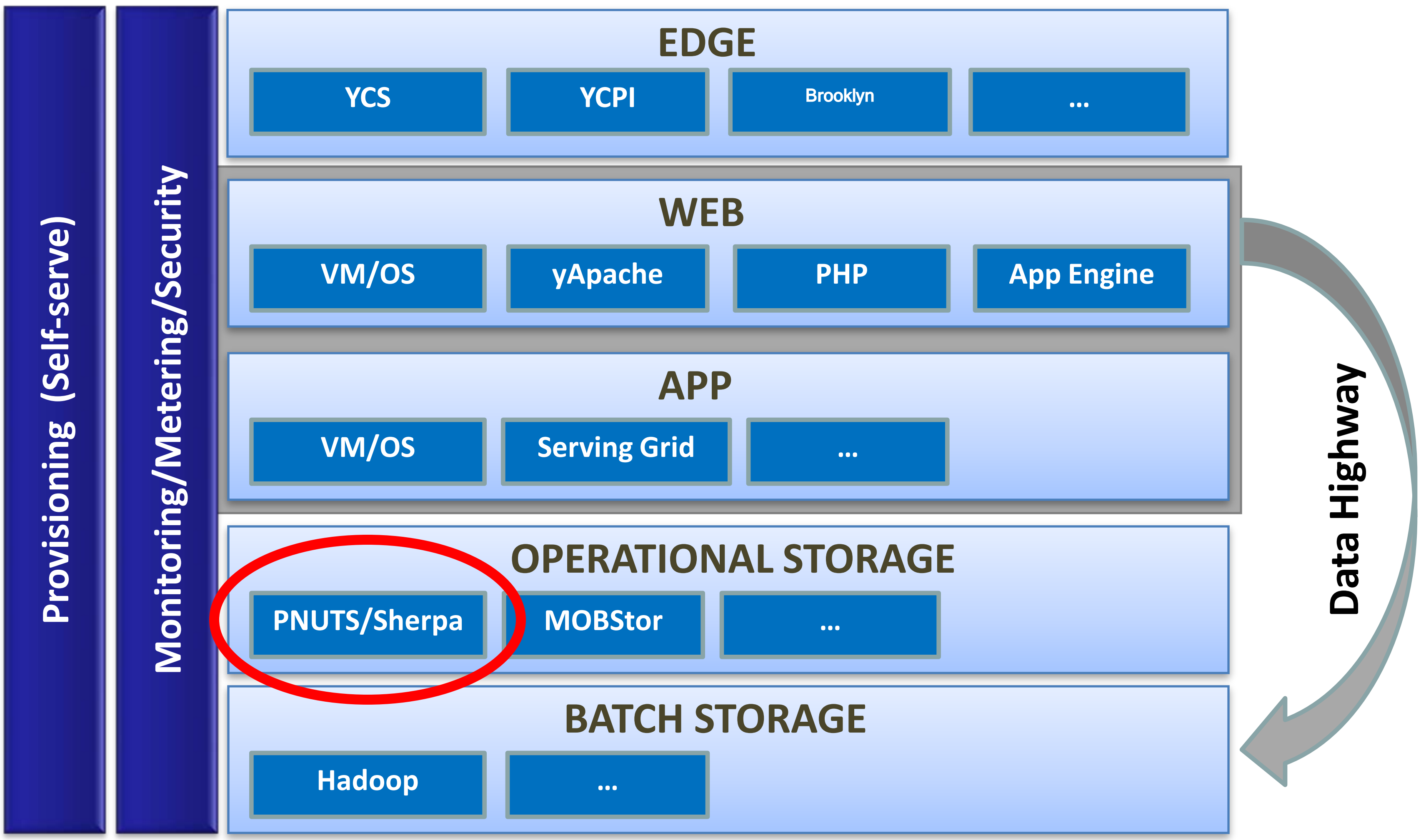
Outline

- Clouds
- Scalable serving—the new landscape
  - Very Large Scale Distributed systems (VLSD)
- Yahoo!'s PNUTS/Sherpa
- Comparison of several systems
  - Preview of upcoming Y! Cloud Serving (YCS) benchmark

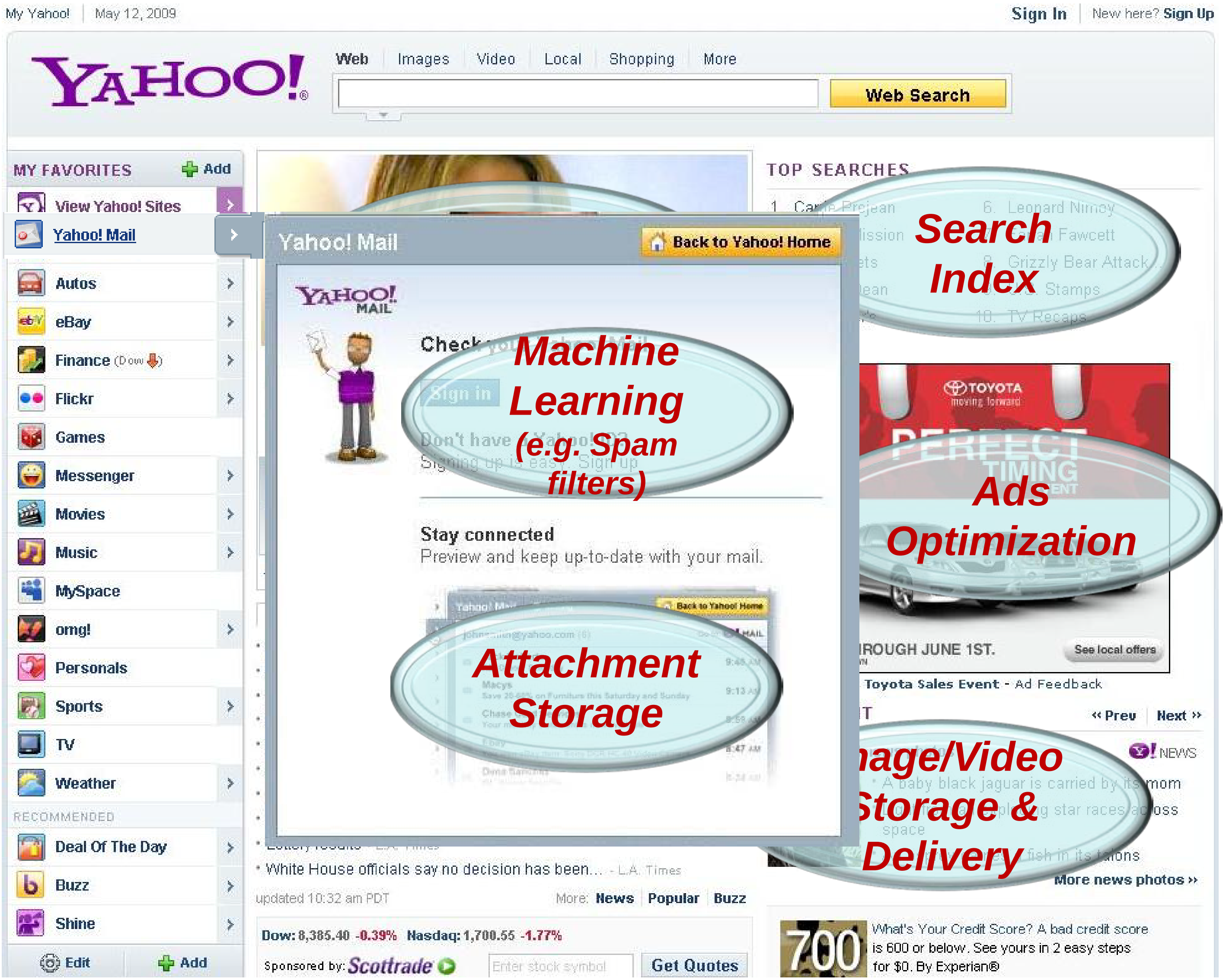
Types of Cloud Services

- Two kinds of cloud services:
  - Horizontal ("Platform") Cloud Services
    - Functionality enabling tenants to build applications or new services on top of the cloud
  - Functional Cloud Services
    - Functionality that is useful in and of itself to tenants. E.g., various SaaS instances, such as Saleforce.com; Google Analytics and Yahoo!'s IndexTools; Yahoo! properties aimed at end-users and small businesses, e.g., flickr, Groups, Mail, News, Shopping
    - Could be built on top of horizontal cloud services or from scratch
    - Yahoo! has been offering these for a long while (e.g., Mail for SMB, Groups, Flickr, BOSS, Ad exchanges, YQL)

Yahoo! Horizontal Cloud Stack



Cloud-Power @ Yahoo!



Yahoo!'s Cloud: Massive Scale, Geo-Footprint

- Massive user base and engagement
  - 500M+ unique users per month
  - Hundreds of petabyte of storage
  - Hundreds of billions of objects
  - Hundred of thousands of requests/sec
- Global
  - Tens of globally distributed data centers
  - Serving each region at low latencies
- Challenging Users
  - Downtime is not an option (outages cost $millions)
  - Very variable usage patterns

New in 2010!

- SIGMOD and SIGOPS are starting a new annual conference (co-located with SIGMOD in 2010):

ACM Symposium on Cloud Computing (SoCC)

PC Chairs: Surajit Chaudhuri & Mendel Rosenblum

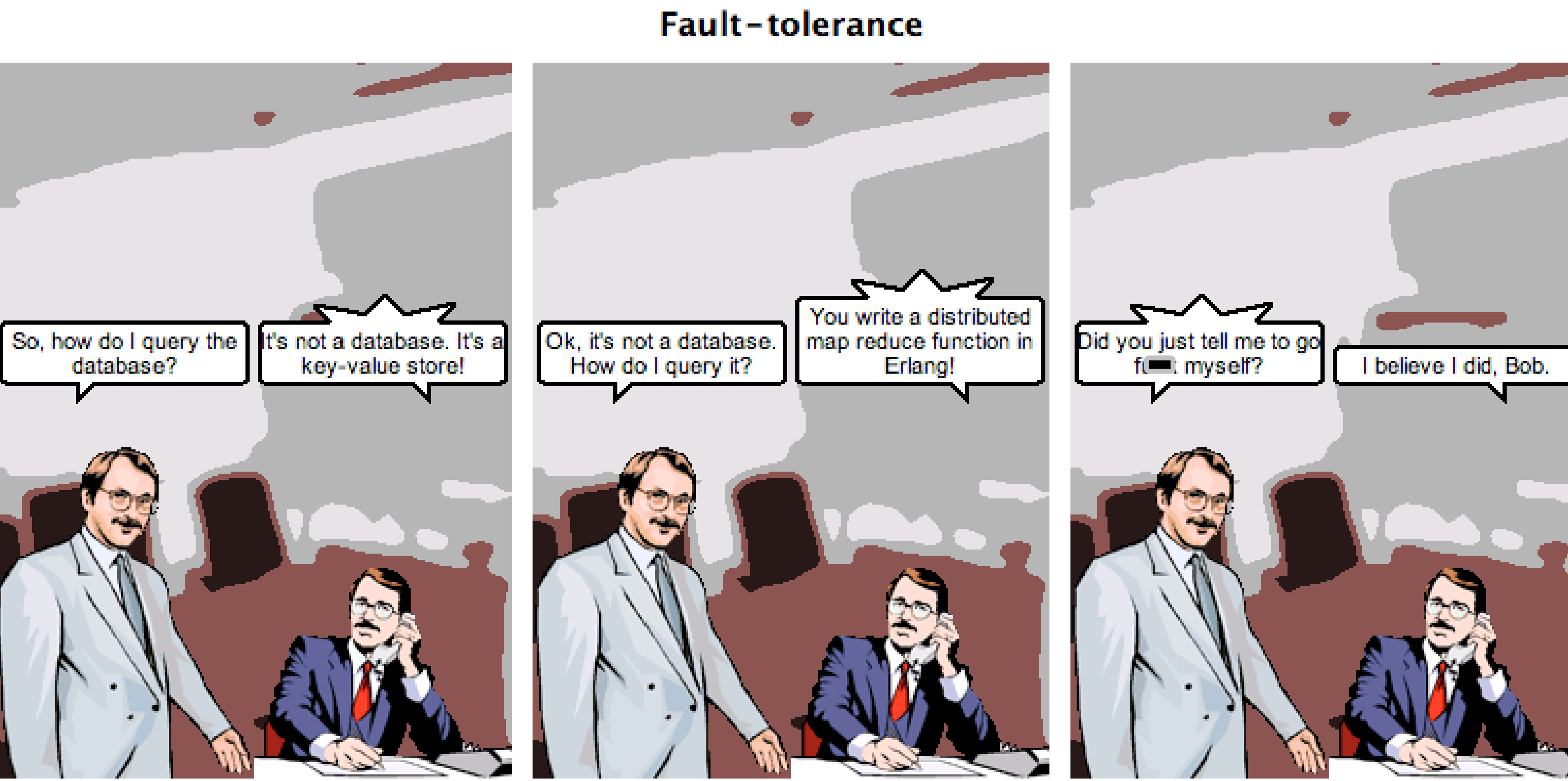
GC: Joe Hellerstein Treasurer: Brian Cooper

- Steering committee: Phil Bernstein, Ken Birman, Joe Hellerstein, John Ousterhout, Raghu Ramakrishnan, Doug Terry, John Wilkes

ACID or BASE? Litmus tests are colorful, but the picture is cloudy

VERY LARGE SCALE DISTRIBUTED (VLSD) DATA SERVING

Databases and Key-Value Stores



http://browsertoolkit.com/fault-tolerance.png

Web Data Management

Large data analysis (Hadoop)

- Warehousing
- Scan oriented workloads
- Focus on sequential disk I/O
- $ per cpu cycle

Structured record storage (PNUTS/Sherpa)

- CRUD
- Point lookups and short scans
- Index organized table and random I/Os
- $ per latency

Blob storage (MObStor)

- Object retrieval and streaming
- Scalable file storage
- $ per GB storage & bandwidth

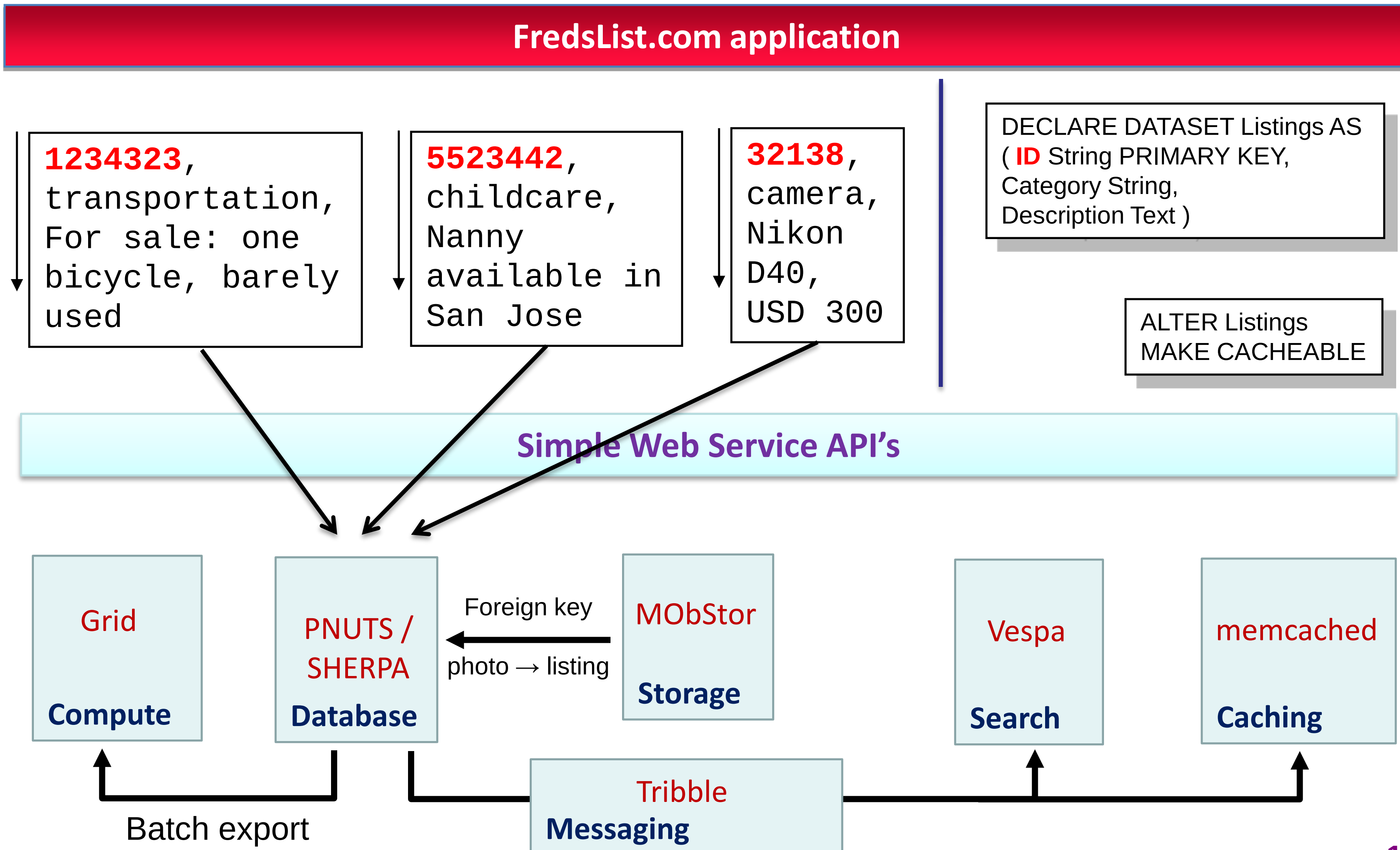
The World Has Changed

- Web serving applications need:
  - Scalability!
    - Preferably elastic
  - Flexible schemas
  - Geographic distribution
  - High availability
  - Reliable storage
- Web serving applications can do without:
  - Complicated queries
  - Strong transactions
    - But some form of consistency is still desirable

Typical Applications

- User logins and profiles
  - Including changes that must not be lost!
    - But single-record “transactions” suffice
- Events
  - Alerts (e.g., news, price changes)
  - Social network activity (e.g., user goes offline)
  - Ad clicks, article clicks
- Application-specific data
  - Postings in message board
  - Uploaded photos, tags
  - Shopping carts

Data Serving in the Y! Cloud



VLSD Data Serving Stores

- Must partition data across machines
  - How are partitions determined?
  - Can partitions be changed easily? (Affects elasticity)
  - How are read/update requests routed?
  - Range selections? Can requests span machines?
- Availability: What failures are handled?
  - With what semantic guarantees on data access?
- (How) Is data replicated?
  - Sync or async? Consistency model? Local or geo?
- How are updates made durable?
- How is data stored on a single machine?

The CAP Theorem

- You have to give up one of the following in a distributed system (Brewer, PODC 2000; Gilbert/Lynch, SIGACT News 2002):
  - Consistency of data
    - Think serializability
  - Availability
    - Pinging a live node should produce results
  - Partition tolerance
    - Live nodes should not be blocked by partitions

Approaches to CAP

- "BASE"
  - No ACID, use a single version of DB, reconcile later
- Defer transaction commit
  - Until partitions fixed and distr xact can run
- Eventual consistency (e.g., Amazon Dynamo)
  - Eventually, all copies of an object converge
- Restrict transactions (e.g., Sharded MySQL)
  - 1-M/c Xacts: Objects in xact are on the same machine
  - 1-Object Xacts: Xact can only read/write 1 object
- Object timelines (PNUTS)

http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

“I want a big, virtual database”

“What I want is a robust, high performance virtual relational database that runs transparently over a cluster, nodes dropping in and out of service at will, read-write replication and data migration all done automatically.

I want to be able to install a database on a server cloud and use it like it was all running on one machine.”

-- Greg Linden’s blog

18

Y! CCDI

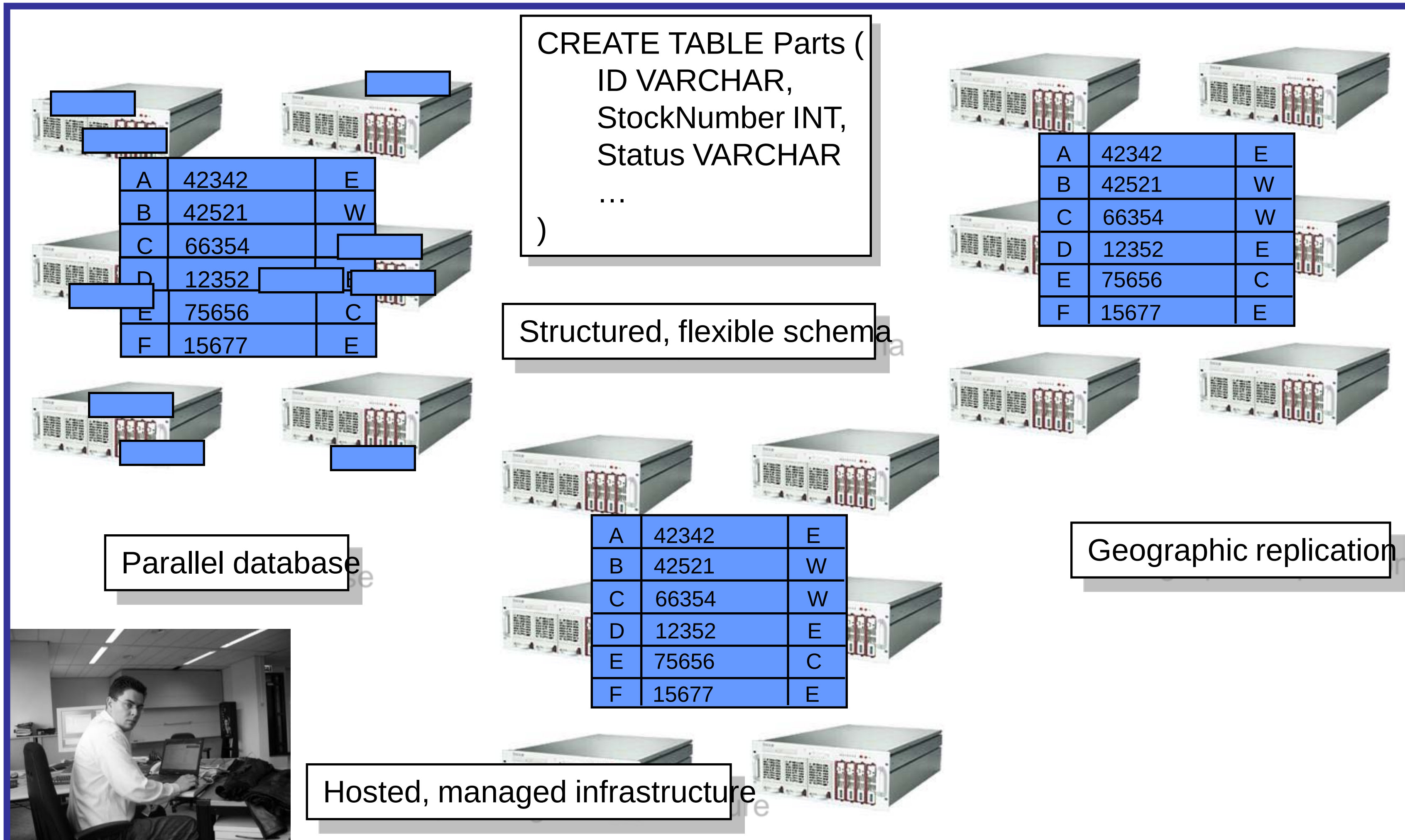
PNUTS / SHERPA

To Help You Scale Your Mountains of Data

Yahoo! Serving Storage Problem

- Small records – 100KB or less
- Structured records – Lots of fields, evolving
- Extreme data scale - Tens of TB
- Extreme request scale - Tens of thousands of requests/sec
- Low latency globally - 20+ datacenters worldwide
- High Availability - Outages cost $millions
- Variable usage patterns - Applications and users change

What is PNUTS/Sherpa?



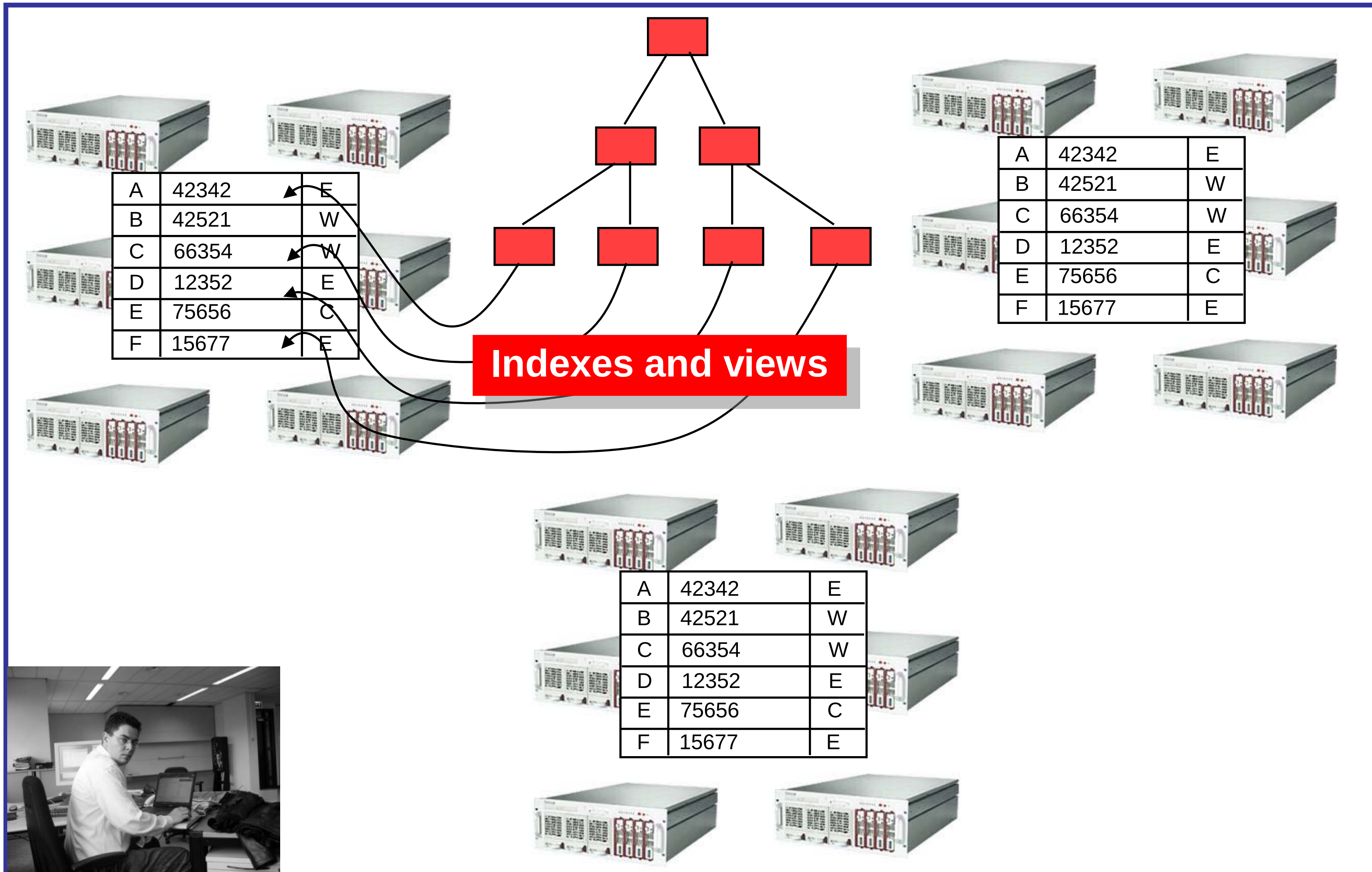

```
CREATE TABLE Parts (
    ID VARCHAR,
    StockNumber INT,
    Status VARCHAR
    …
)
```

A	42342	E
B	42521	W
C	66354	W
D	12352	E
E	75656	C
F	15677	E

A	42342	E
B	42521	W
C	66354	W
D	12352	E
E	75656	C
F	15677	E

A	42342	E
B	42521	W
C	66354	W
D	12352	E
E	75656	C
F	15677	E

What Will It Become?



Design Goals

Scalability
- Thousands of machines
- Easy to add capacity
- Restrict query language to avoid costly queries

Geographic replication
- Asynchronous replication around the globe
- Low-latency local access

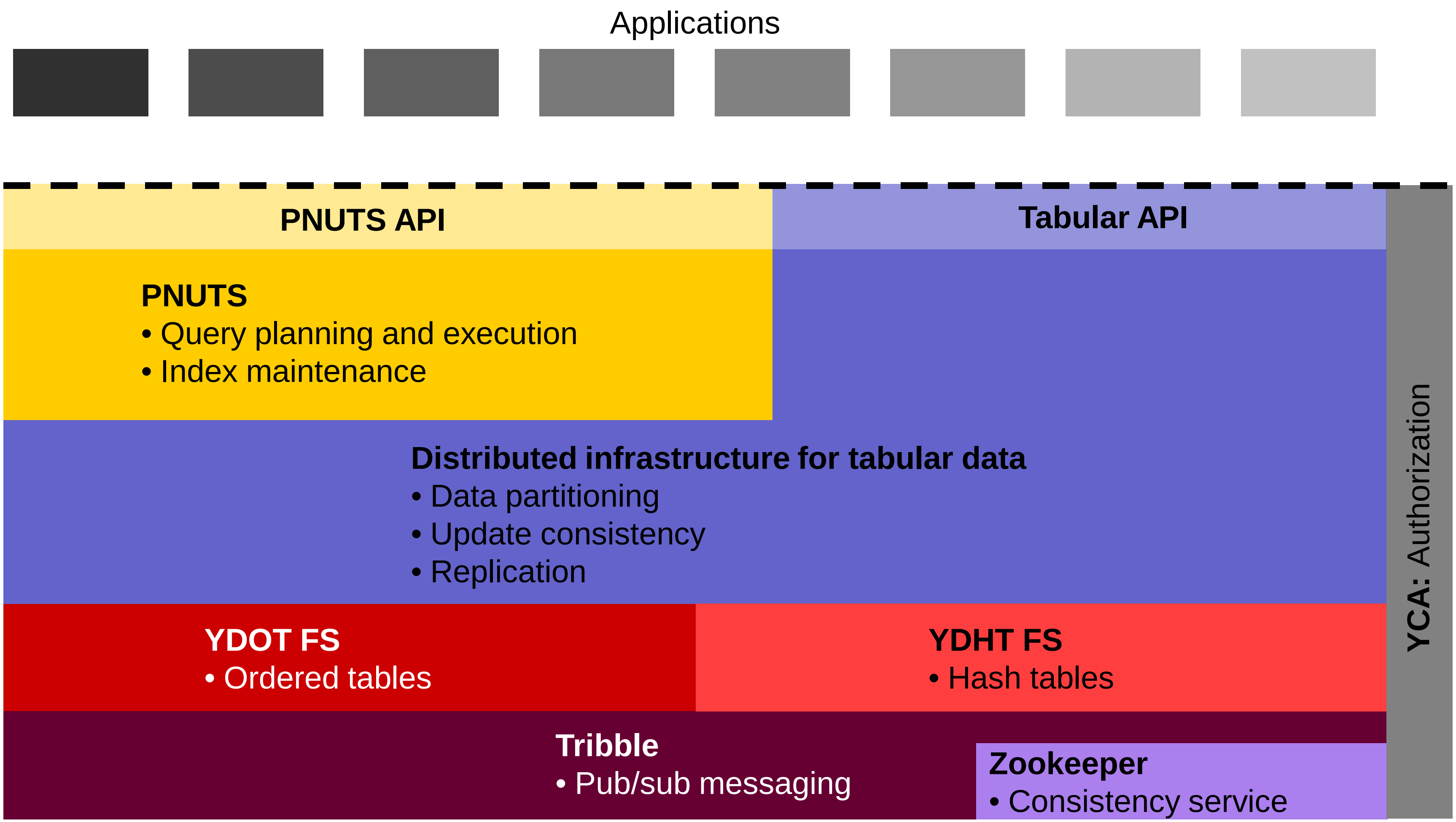
High availability and fault tolerance
- Automatically recover from failures
- Serve reads and writes despite failures

Consistency
- Per-record guarantees
- Timeline model
- Option to relax if needed

Multiple access paths
- Hash table, ordered table
- Primary, secondary access

Hosted service
- Applications plug and play
- Share operational cost

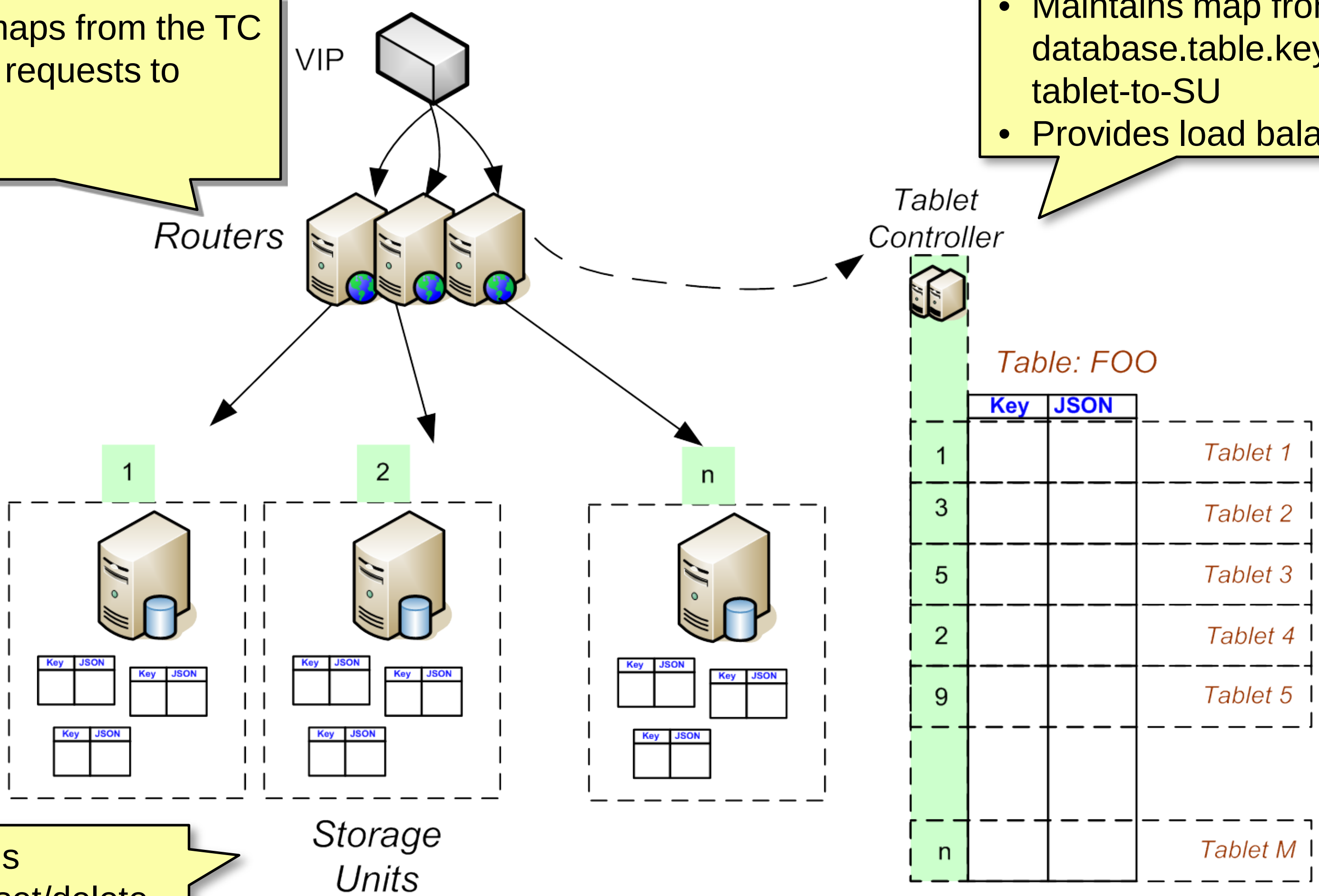
Technology Elements



PNUTS: Key Components

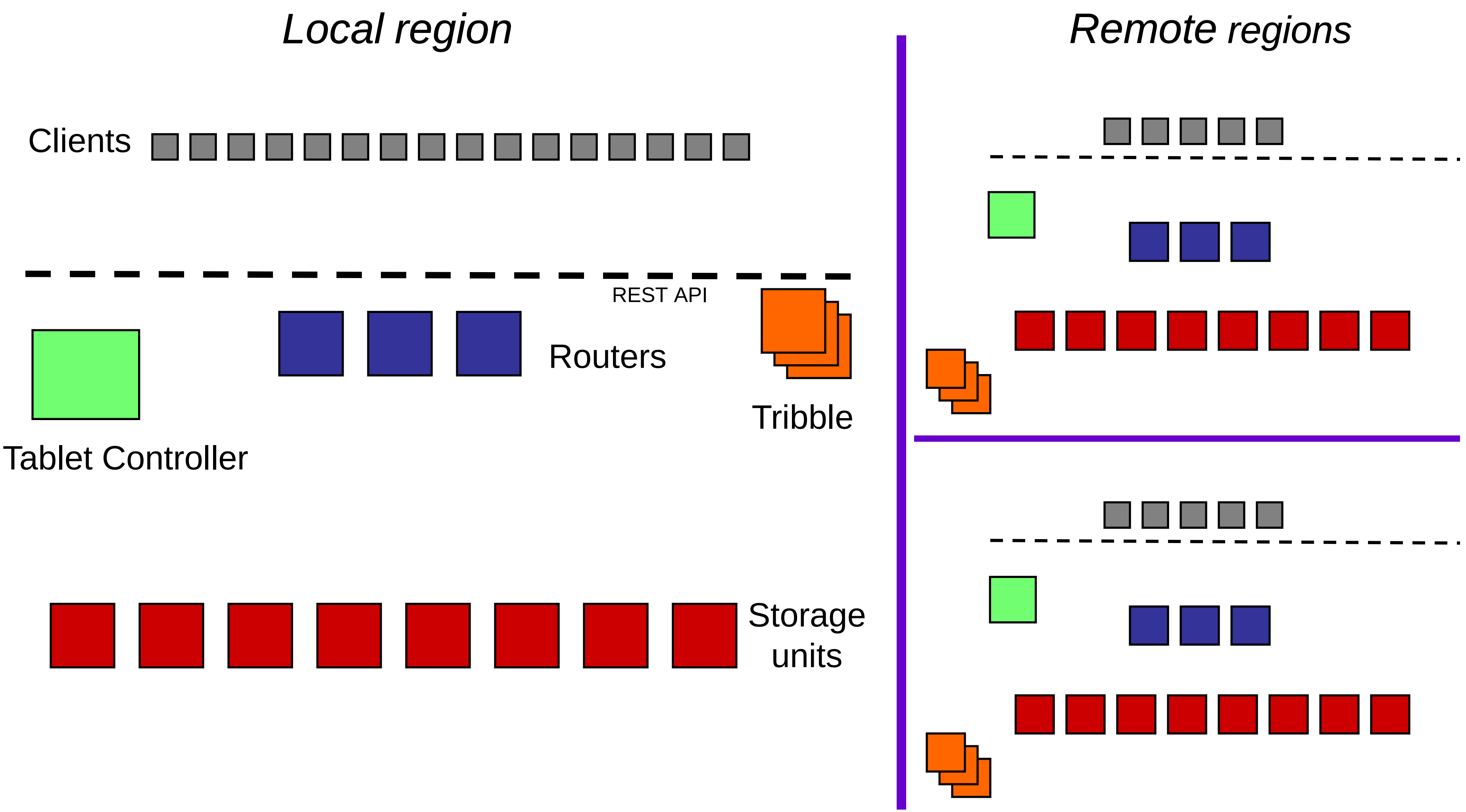
- Caches the maps from the TC
- Routes client requests to correct SU

- Maintains map from database.table.key-to-tablet-to-SU
- Provides load balancing



- Stores records
- Services get/set/delete requests

Detailed Architecture



DATA MODEL

Data Manipulation

- Per-record operations
  - Get
  - Set
  - Delete
- Multi-record operations
  - Multiget
  - Scan
  - Getrange
- Web service (RESTful) API

Tablets—Hash Table

	Name	Description	Price
0x0000	Grape	Grapes are good to eat	$12
	Lime	Limes are green	$9
	Apple	Apple is wisdom	$1
	Strawberry	Strawberry shortcake	$900
0x2AF3	Orange	Arrgh! Don’t get scurvy!	$2
	Avocado	But at what price?	$3
	Lemon	How much did you pay for this lemon?	$1
	Tomato	Is this a vegetable?	$14
0x911F	Banana	The perfect fruit	$2
0xFFFF	Kiwi	New Zealand	$8

Tablets—Ordered Table

	Name	*Description*	*Price*
A	Apple	Apple is wisdom	$1
	Avocado	But at what price?	$3
	Banana	The perfect fruit	$2
	Grape	Grapes are good to eat	$12
H	Kiwi	New Zealand	$8
	Lemon	How much did you pay for this lemon?	$1
	Lime	Limes are green	$9
	Orange	Arrgh! Don't get scurvy!	$2
Q	Strawberry	Strawberry shortcake	$900
	Tomato	Is this a vegetable?	$14
Z			

Flexible Schema

Posted date	*Listing id*	*Item*	*Price*	*Color*	*Condition*
6/1/07	424252	Couch	$570		Good
6/1/07	763245	Bike	$86		
6/3/07	211242	Car	$1123	Red	Fair
6/5/07	421133	Lamp	$15		

Primary vs. Secondary Access

Primary table

Posted date	Listing id	Item	Price
6/1/07	424252	Couch	$570
6/1/07	763245	Bike	$86
6/3/07	211242	Car	$1123
6/5/07	421133	Lamp	$15

Secondary index

Price	Posted date	Listing id
15	6/5/07	421133
86	6/1/07	763245
570	6/1/07	424252
1123	6/3/07	211242

Planned functionality

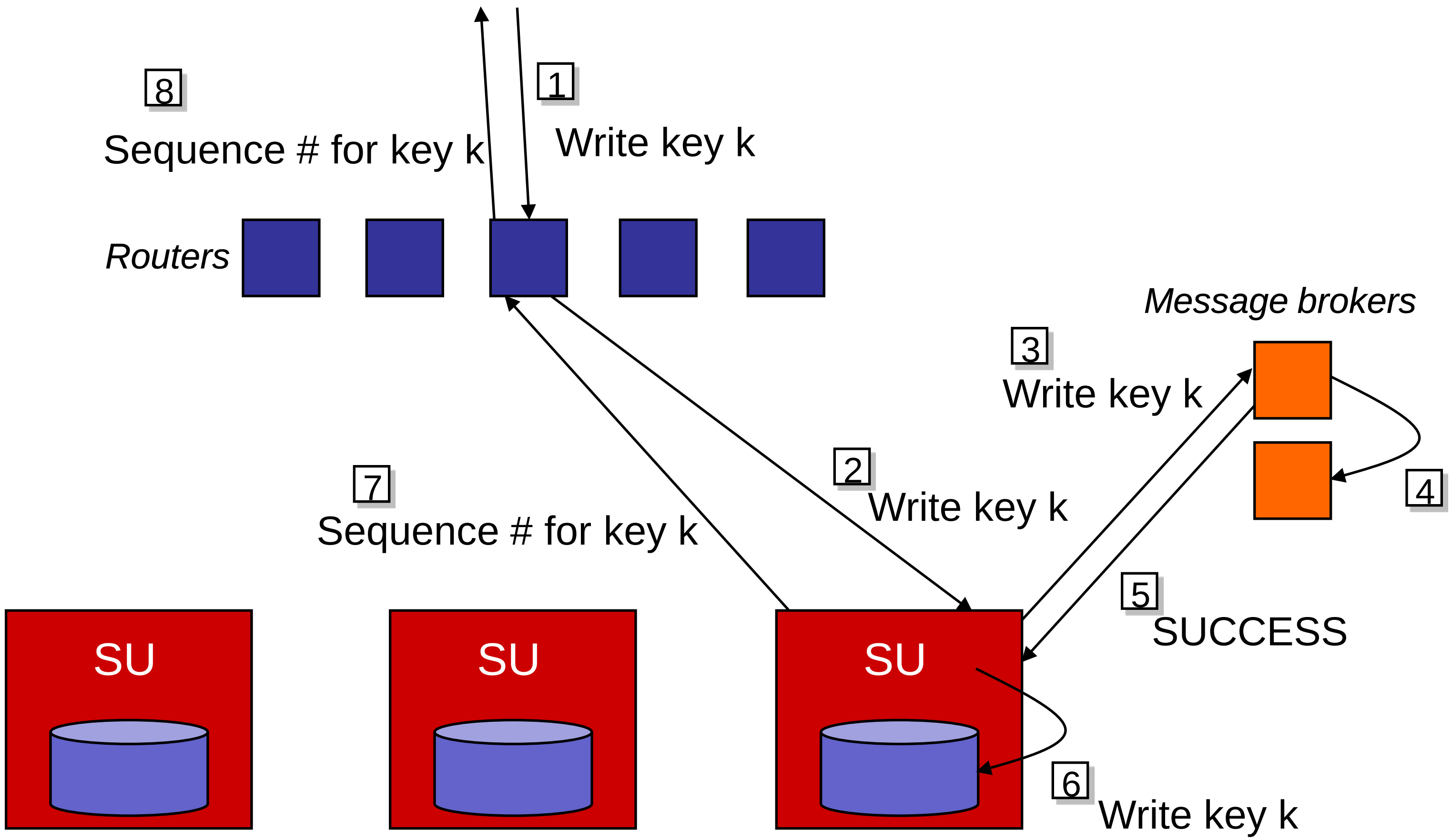
Index Maintenance

- How to have lots of interesting indexes and views, without killing performance?

- Solution: Asynchrony!
  - Indexes/views updated asynchronously when base table updated

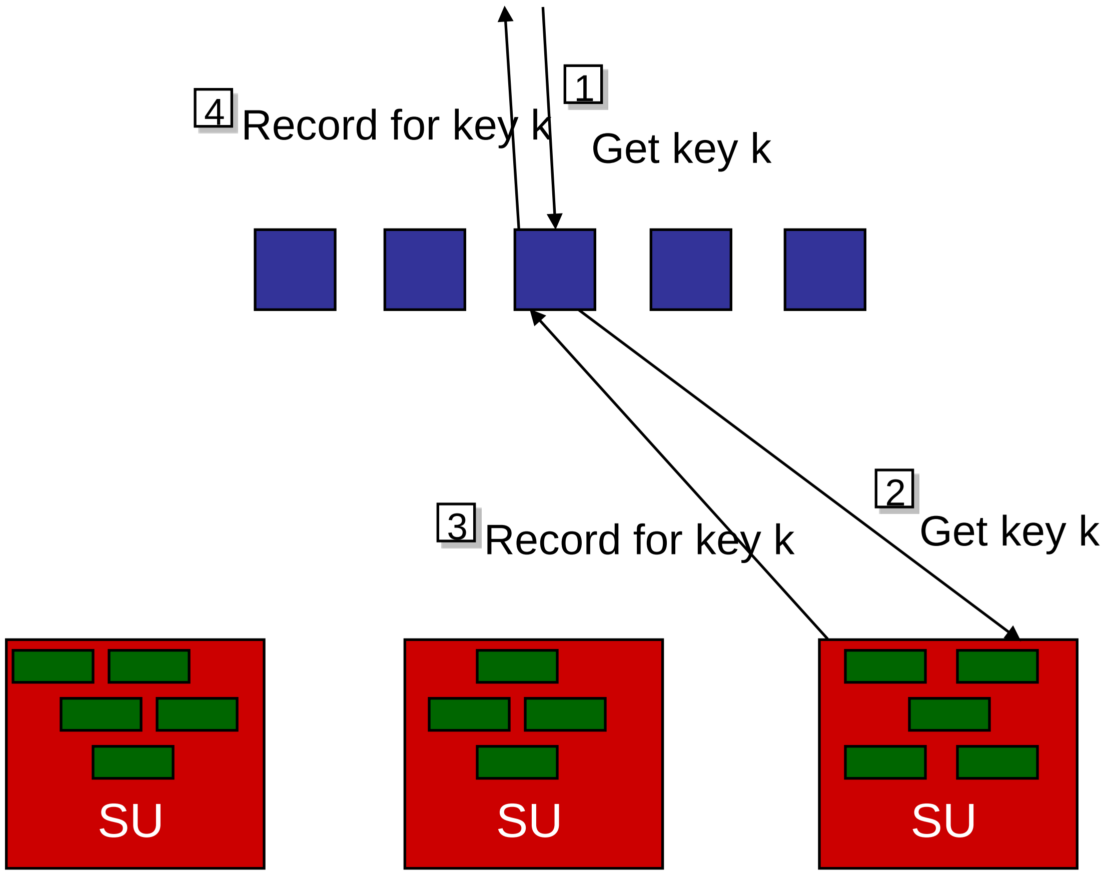
PROCESSING
READS & UPDATES

Updates

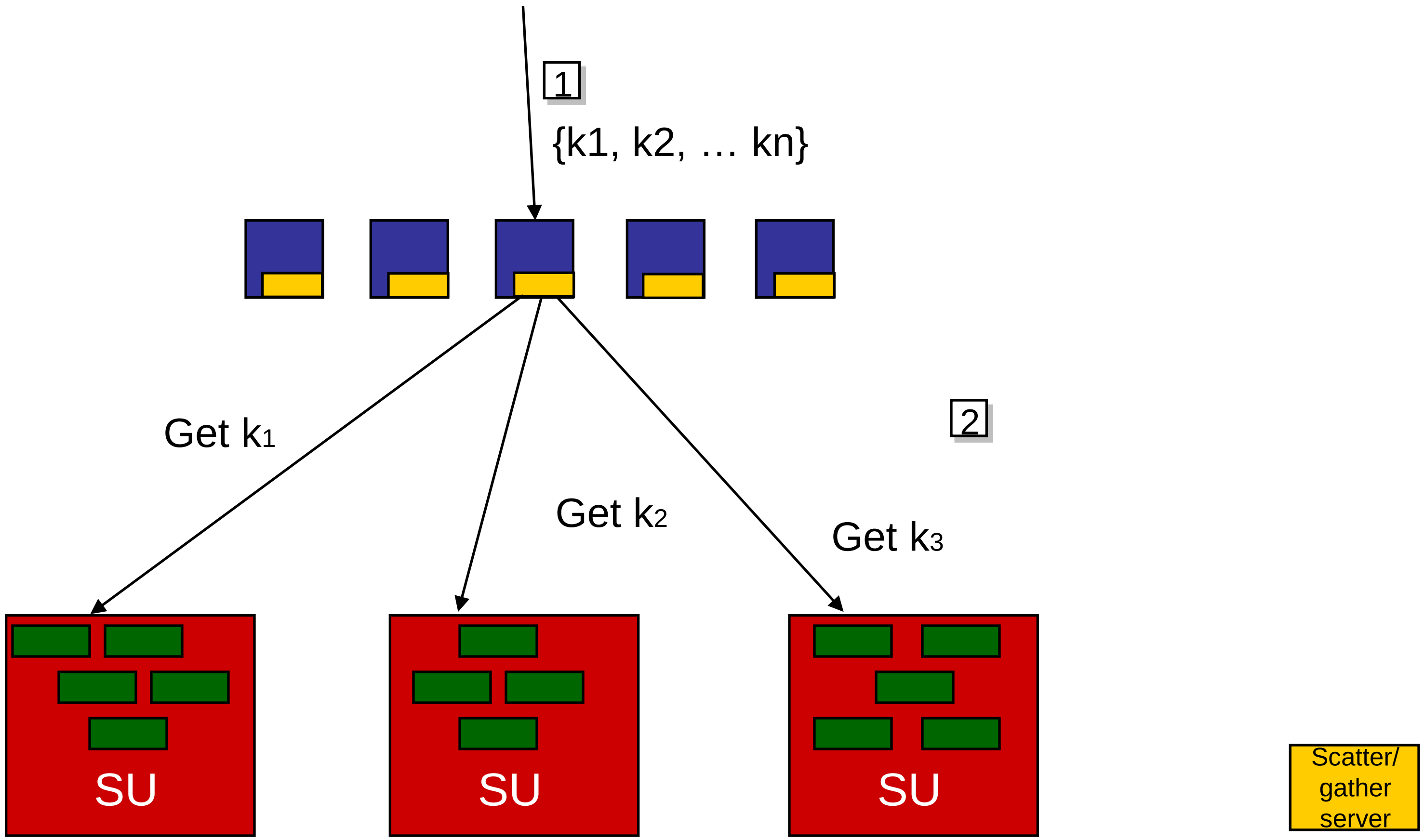


8
Sequence # for key k
1
Write key k
Routers
Message brokers
3
Write key k
2
Write key k
4
7
Sequence # for key k
5
SUCCESS
SU
SU
SU
6
Write key k

Accessing Data

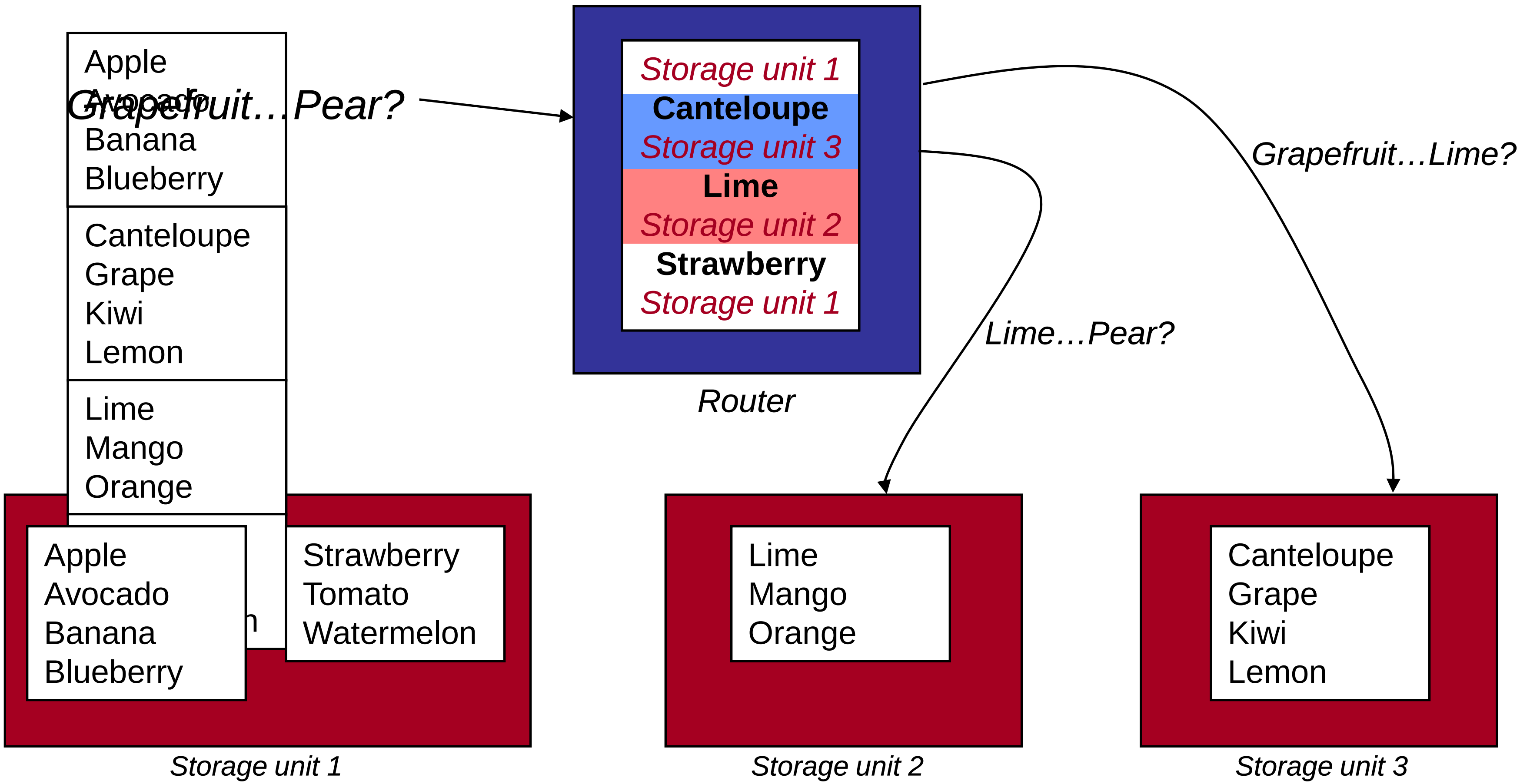


Bulk Read



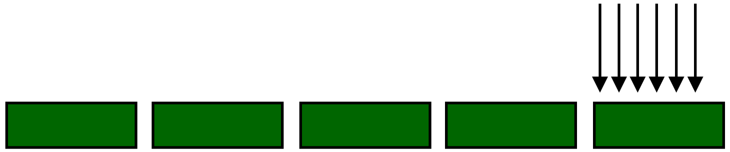
Range Queries in YDOT

- Clustered, ordered retrieval of records

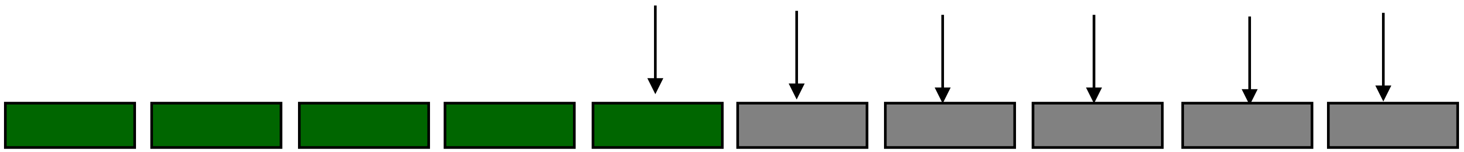


Bulk Load in YDOT

- YDOT bulk inserts can cause performance hotspots



- Solution: preallocate tablets



ASYNCHRONOUS REPLICATION AND CONSISTENCY

Asynchronous Replication

Consistency Model

- If copies are asynchronously updated, what can we say about stale copies?
  - – ACID guarantees require synchronous updts
  - – Eventual consistency: Copies can drift apart, but will eventually converge if the system is allowed to quiesce
    - To what value will copies converge?
    - Do systems ever “quiesce”?
  - – Is there any middle ground?

Example: Social Alice

West

(Alice logs on)

User	Status
Alice	Busy

User	Status
Alice	Busy

User	Status
Alice	???

East

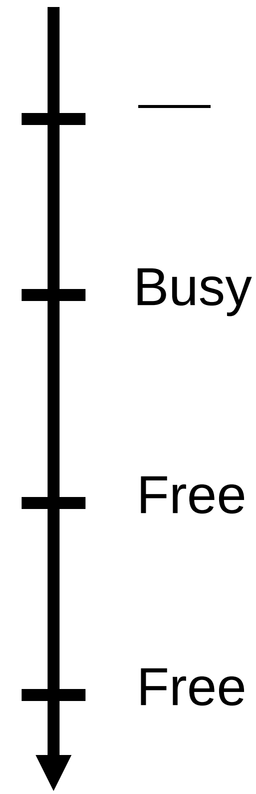
User	Status
Alice	____

(Network fault,
updt goes to East)

User	Status
Alice	Free

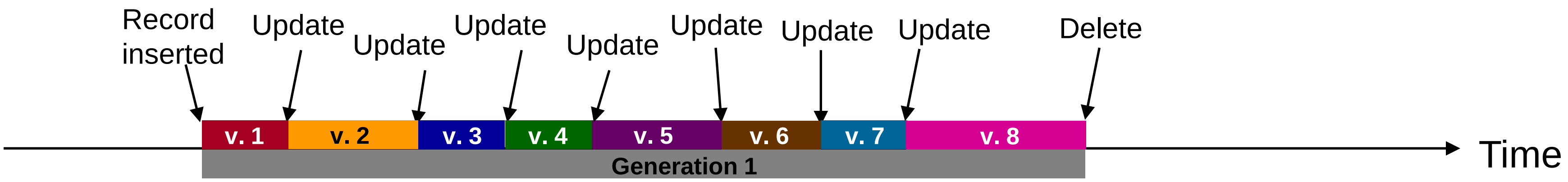
User	Status
Alice	???

Record Timeline



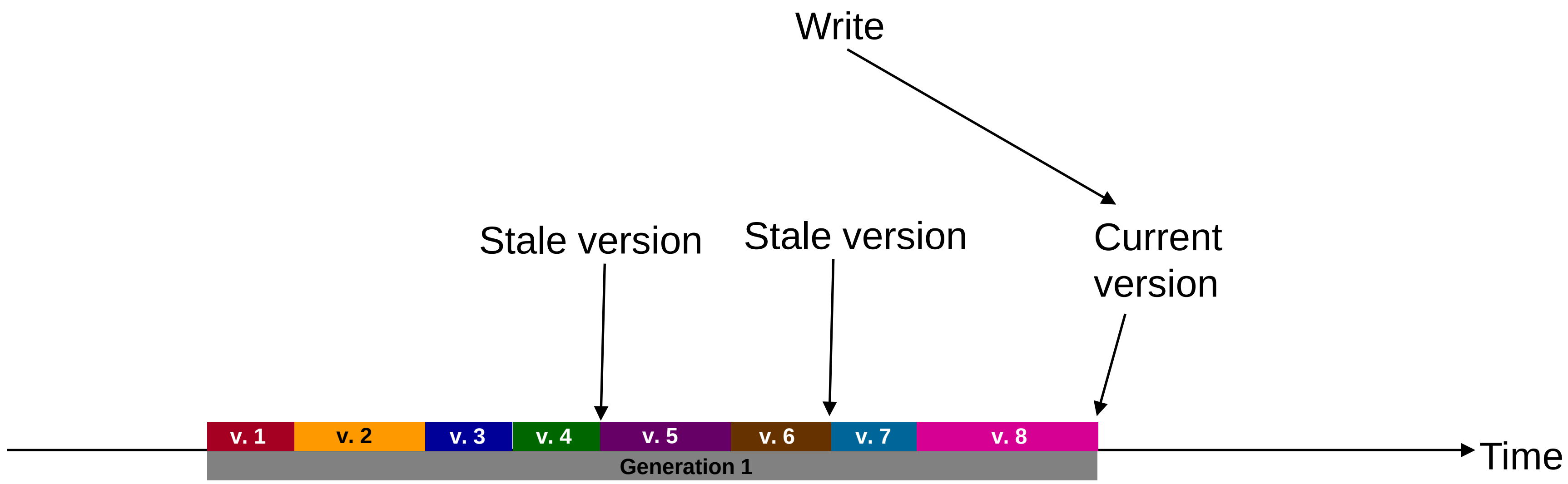
PNUTS Consistency Model

- Goal: Make it easier for applications to reason about updates and cope with asynchrony
- What happens to a record with primary key “Alice”?



As the record is updated, copies may get out of sync.

PNUTS Consistency Model



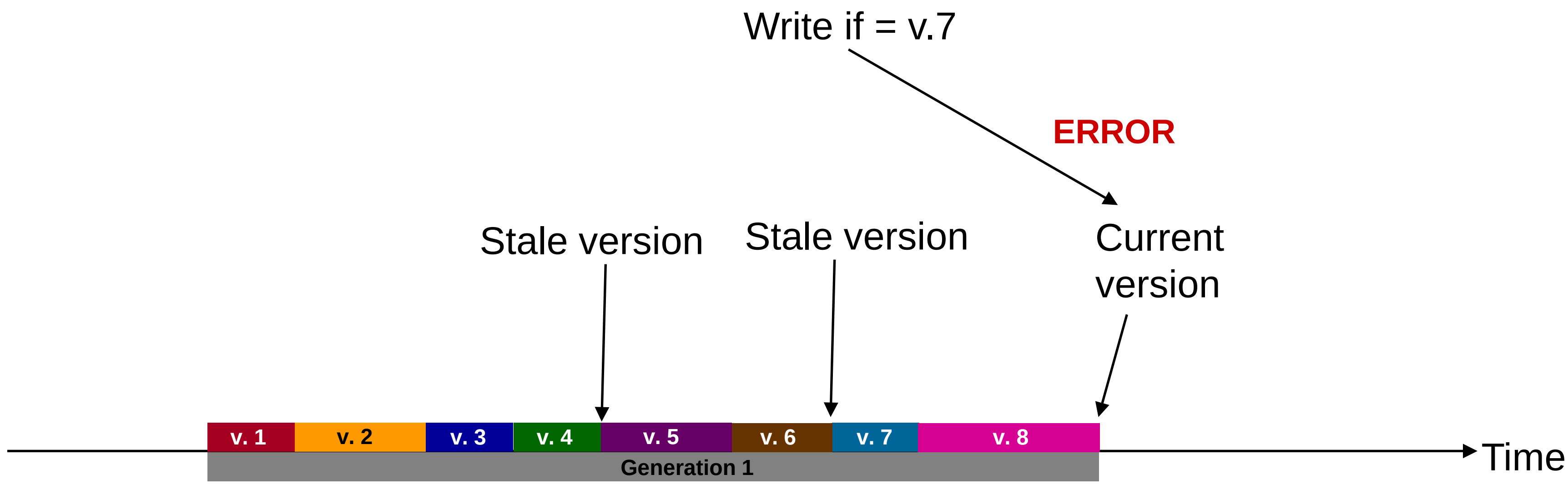
Achieved via per-record primary copy protocol

(To maximize availability, record masterships automaticlly transferred if site fails)

Can be selectively weakened to eventual consistency

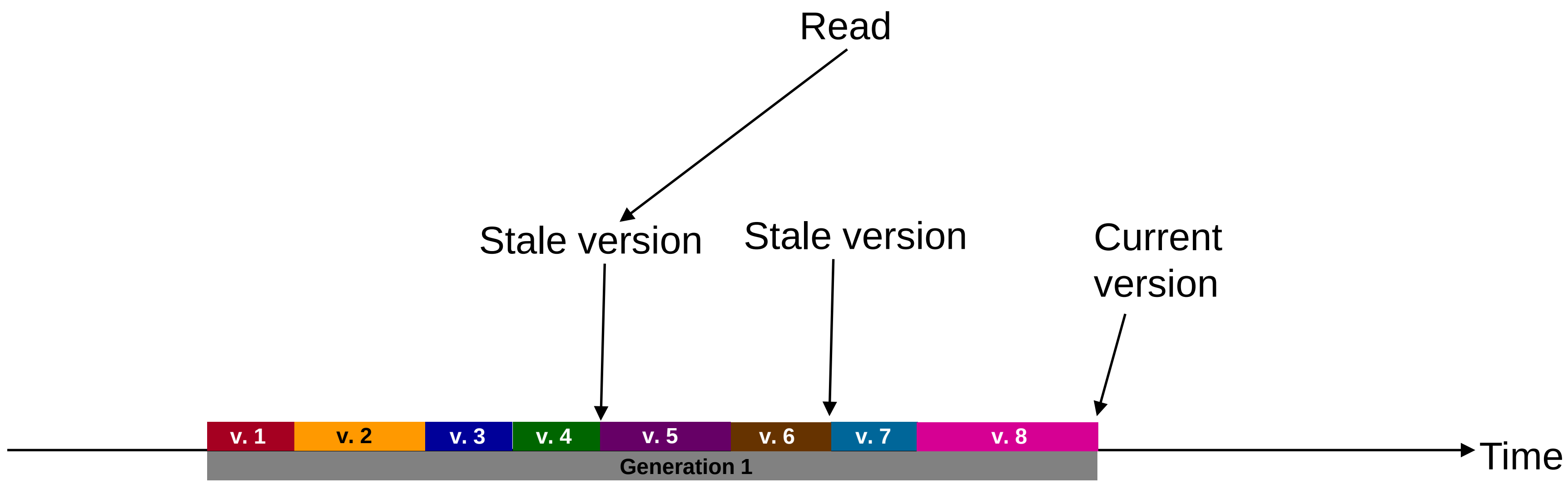
(local writes that are reconciled using version vectors)

PNUTS Consistency Model



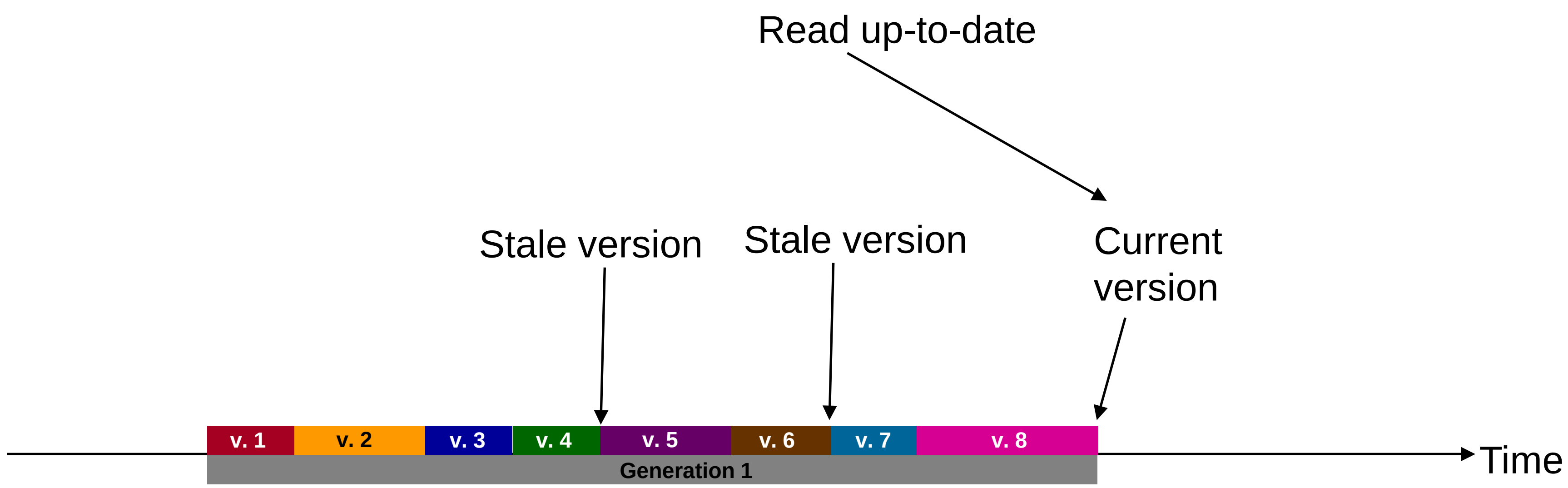
Test-and-set writes facilitate per-record transactions

PNUTS Consistency Model



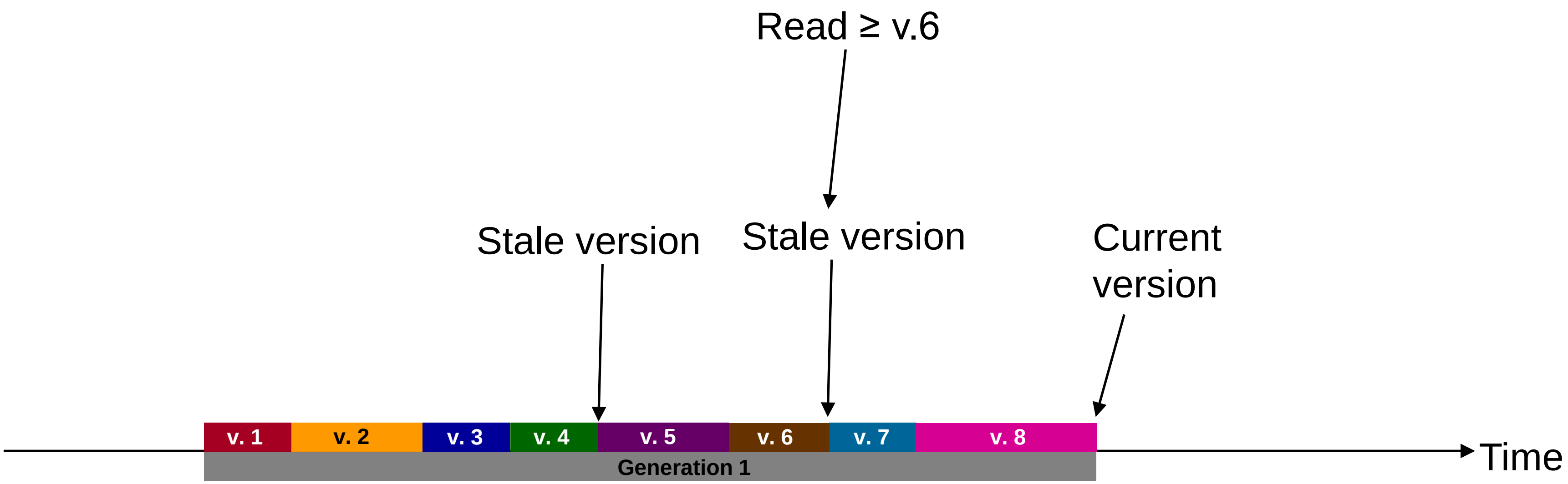
In general, reads are served using a local copy

PNUTS Consistency Model



But application can request and get current version

PNUTS Consistency Model



Or variations such as “read forward”—while copies may lag the master record, *every* copy goes through the same sequence of changes

OPERABILITY

Distribution

Data shuffling for load balancing

6/1/07			$70
6/1/07	256623	Car	$1123
6/2/07	636353	Bike	$86
6/5/07	662113	Chair	$10
6/7/07	121113	Lamp	$19
6/9/07	887734	Bike	$56
6/11/07	252111	Scooter	$18
6/11/07	116458	Hammer	$8000

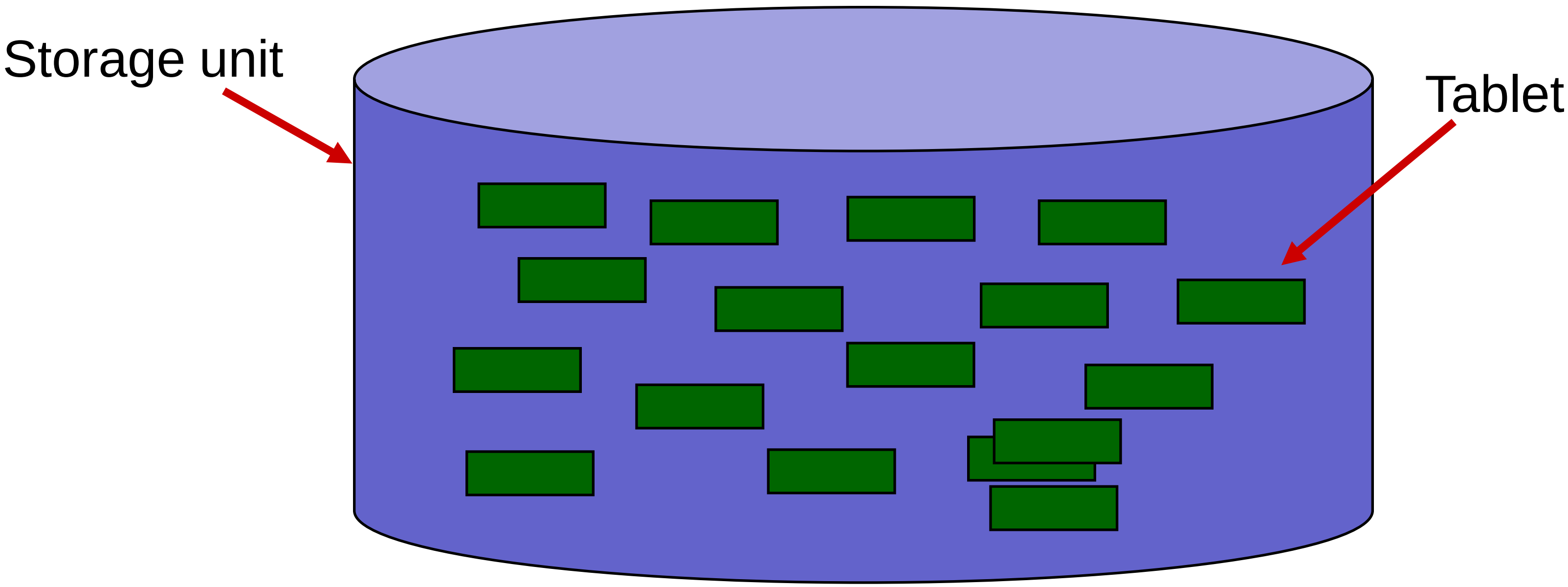
Server 1 *Server 2* *Server 3* *Server 4*

51

Tablet Splitting and Balancing

Each storage unit has many tablets (horizontal partitions of the table)

Storage unit may become a hotspot



Overfull tablets split

Tablets may grow over time

Shed load by moving tablets to other servers

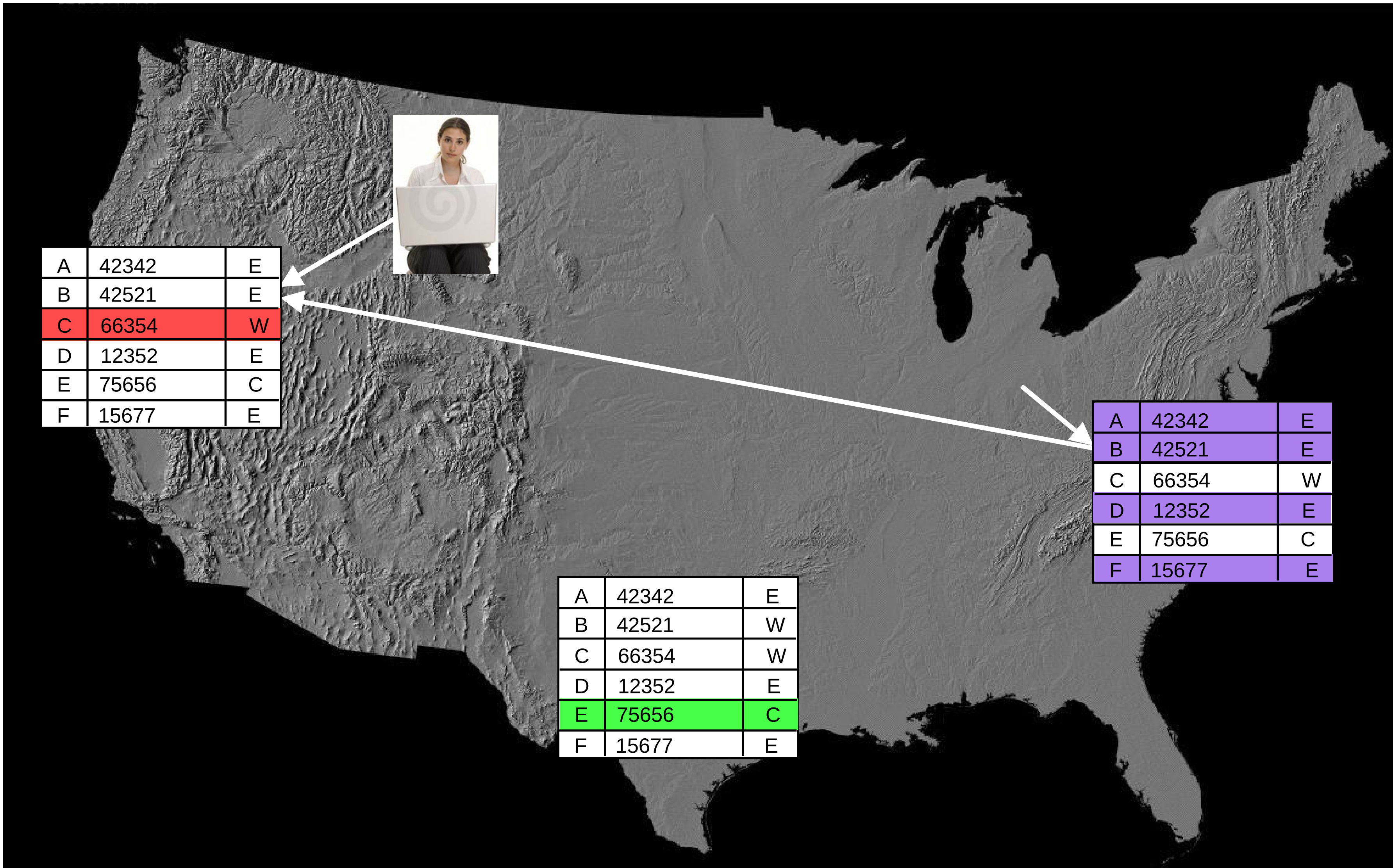
Consistency Techniques

- Per-record mastering
  - Each record is assigned a “master region”
    - May differ between records
  - Updates to the record forwarded to the master region
  - Ensures consistent ordering of updates

- Tablet-level mastering
  - Each tablet is assigned a “master region”
  - Inserts and deletes of records forwarded to the master region
  - Master region decides tablet splits

- These details are hidden from the application
  - Except for the latency impact!

Mastering

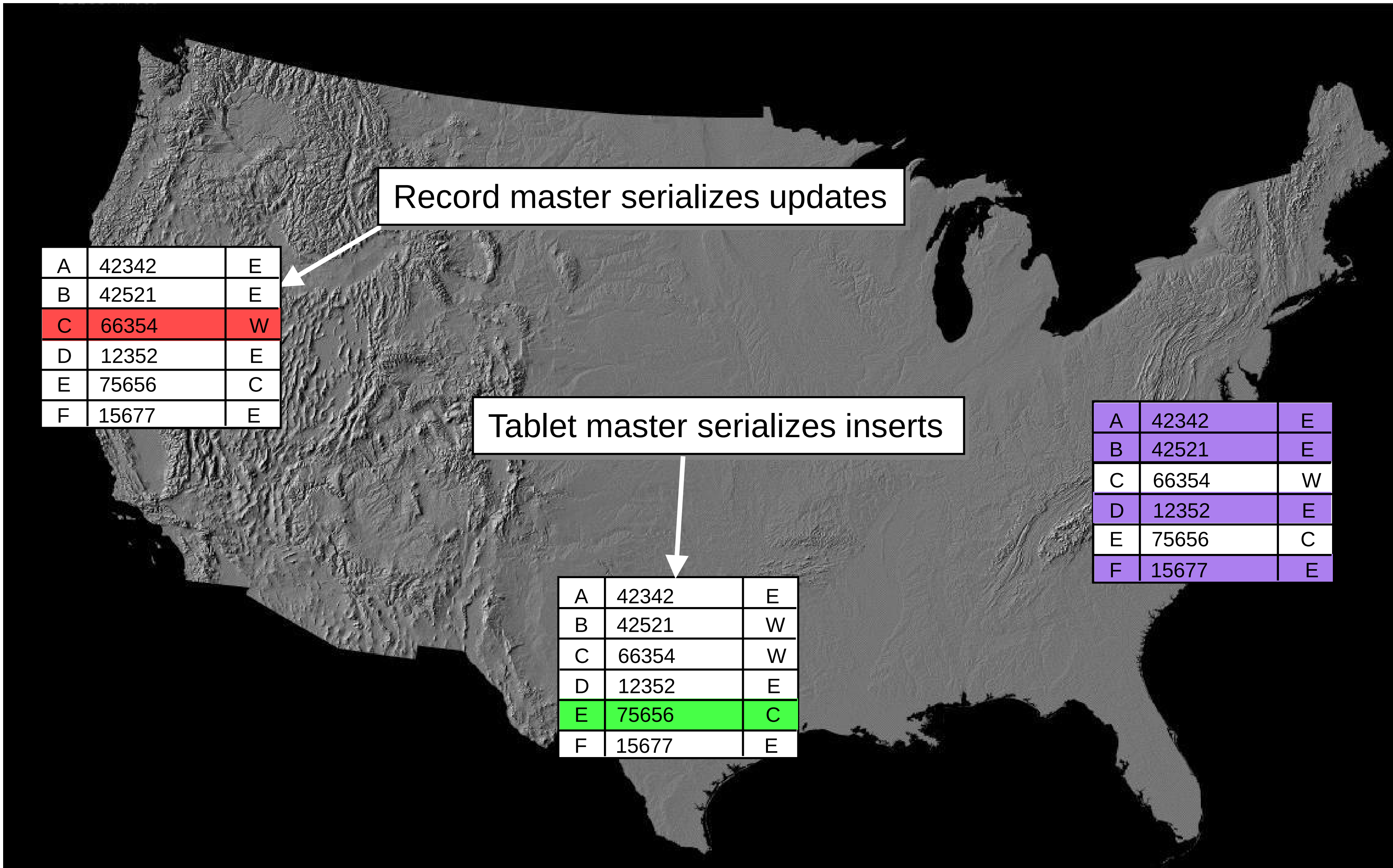


A	42342	E
B	42521	E
C	66354	W
D	12352	E
E	75656	C
F	15677	E

A	42342	E
B	42521	E
C	66354	W
D	12352	E
E	75656	C
F	15677	E

A	42342	E
B	42521	W
C	66354	W
D	12352	E
E	75656	C
F	15677	E

Record vs. Tablet Master

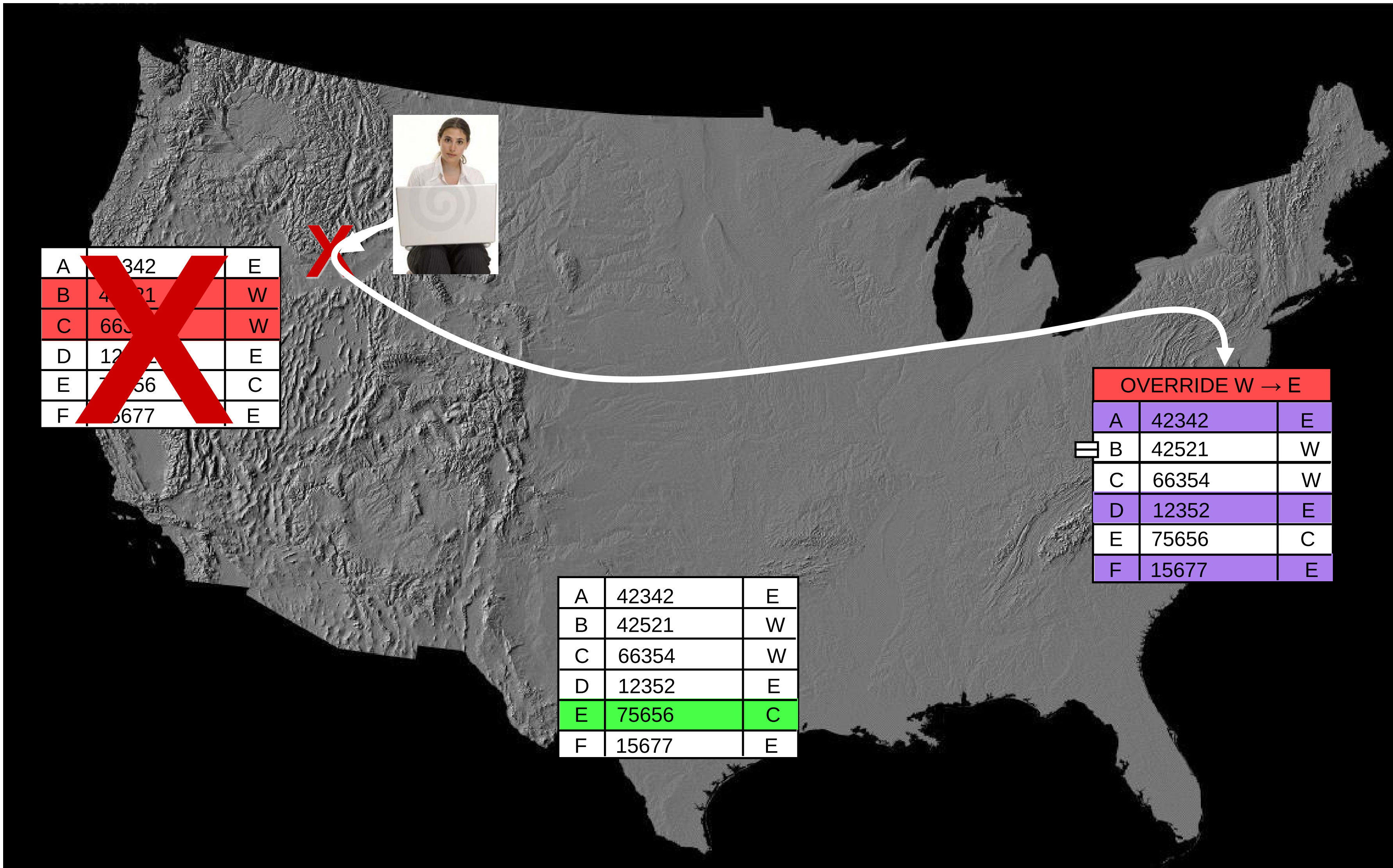


A	42342	E
B	42521	E
C	66354	W
D	12352	E
E	75656	C
F	15677	E

A	42342	E
B	42521	W
C	66354	W
D	12352	E
E	75656	C
F	15677	E

A	42342	E
B	42521	E
C	66354	W
D	12352	E
E	75656	C
F	15677	E

Coping With Failures



Further PNutty Reading

Efficient Bulk Insertion into a Distributed Ordered Table (SIGMOD 2008)
Adam Silberstein, Brian Cooper, Utkarsh Srivastava, Erik Vee,
Ramana Yerneni, Raghu Ramakrishnan

PNUTS: Yahoo!'s Hosted Data Serving Platform (VLDB 2008)
Brian Cooper, Raghu Ramakrishnan, Utkarsh Srivastava,
Adam Silberstein, Phil Bohannon, Hans-Arno Jacobsen,
Nick Puz, Daniel Weaver, Ramana Yerneni

Asynchronous View Maintenance for VLSD Databases (SIGMOD 2009)
Parag Agrawal, Adam Silberstein, Brian F. Cooper, Utkarsh Srivastava and
Raghu Ramakrishnan

Cloud Storage Design in a PNUTShell
Brian F. Cooper, Raghu Ramakrishnan, and Utkarsh Srivastava
Beautiful Data, O’Reilly Media, 2009

Adaptively Parallelizing Distributed Range Queries (VLDB 2009)
Ymir Vigfusson, Adam Silberstein, Brian Cooper, Rodrigo Fonseca

Green Apples and Red Apples

COMPARING SOME CLOUD SERVING STORES

Motivation

- Many “cloud DB” and “nosql” systems out there
  - PNUTS
  - BigTable
    - HBase, Hypertable, HTable
  - Azure
  - Cassandra
  - Megastore (behind Google AppEngine)
  - Amazon Web Services
    - S3, SimpleDB, EBS
  - And more: CouchDB, Voldemort, etc.

- How do they compare?
  - Feature tradeoffs
  - Performance tradeoffs
  - Not clear!

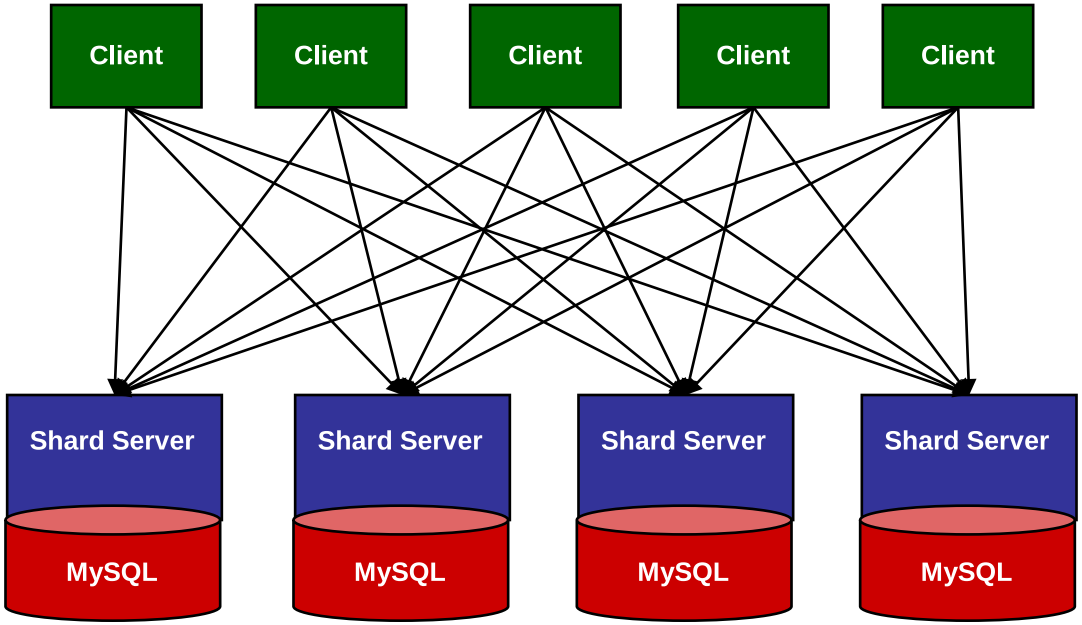
The Contestants

- Baseline: Sharded MySQL
  - Horizontally partition data among MySQL servers
- PNUTS/Sherpa
  - Yahoo!'s cloud database
- Cassandra
  - BigTable + Dynamo
- HBase
  - BigTable + Hadoop

SHARDED MYSQL

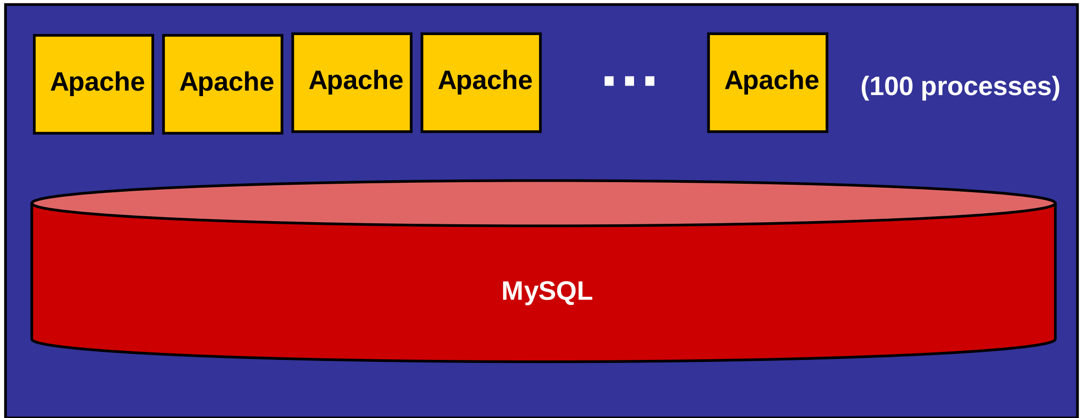
Architecture

- Our own implementation of sharding



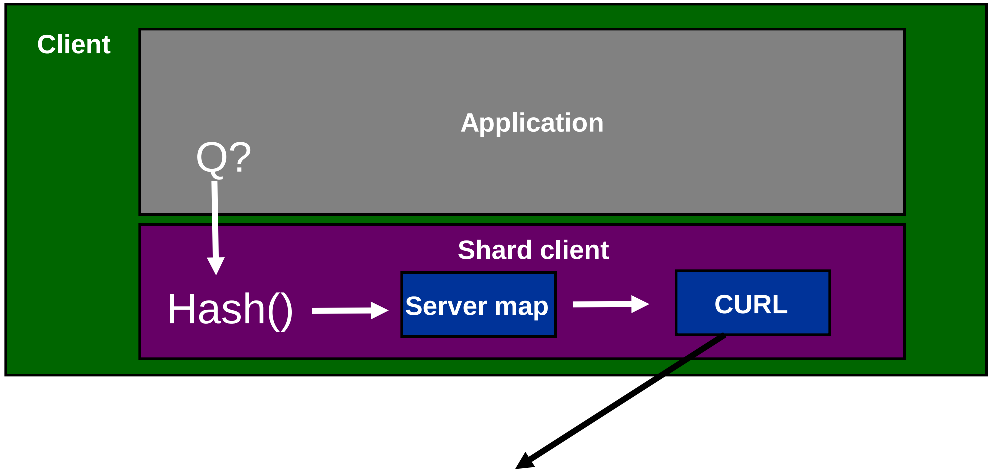
Shard Server

- Server is Apache + plugin + MySQL
  - MySQL schema: key varchar(255), value mediumtext
  - Flexible schema: value is blob of key/value pairs
- Why not direct to MySQL?
  - Flexible schema means an update is:
    - Read record from MySQL
    - Apply changes
    - Write record to MySQL
  - Shard server means the read is local
    - No need to pass whole record over network to change one field



Client

- Application plus shard client
- Shard client
  - Loads config file of servers
  - Hashes record key
  - Chooses server responsible for hash range
  - Forwards query to server



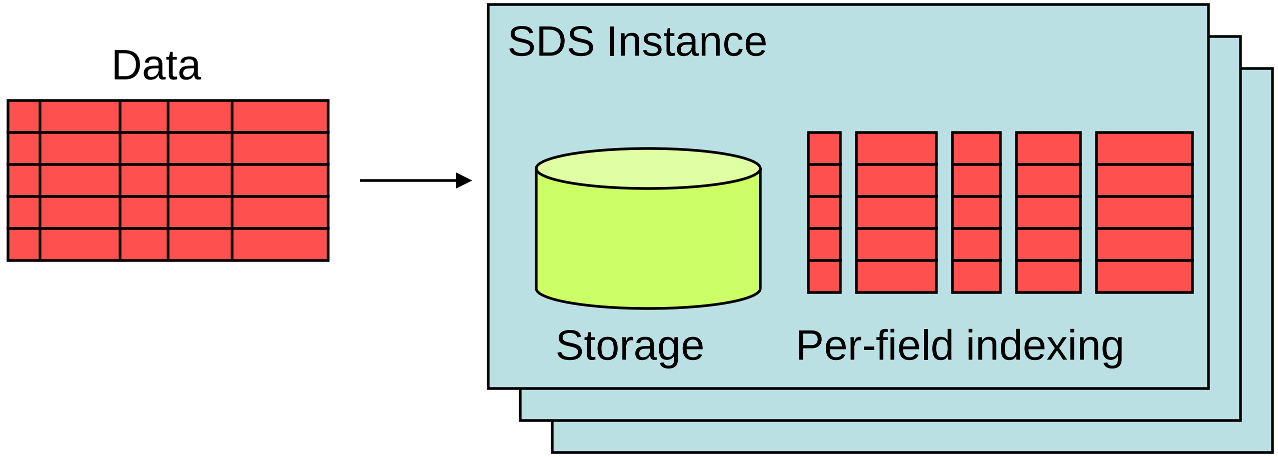
Pros and Cons

- Pros
  - Simple
  - “Infinitely” scalable
  - Low latency
  - Geo-replication

- Cons
  - Not elastic (Resharding is hard)
  - Poor support for load balancing
  - Failover? (Adds complexity)
  - Replication unreliable (Async log shipping)

Azure SDS

- Cloud of SQL Server instances
- App partitions data into instance-sized pieces
  - Transactions and queries within an instance



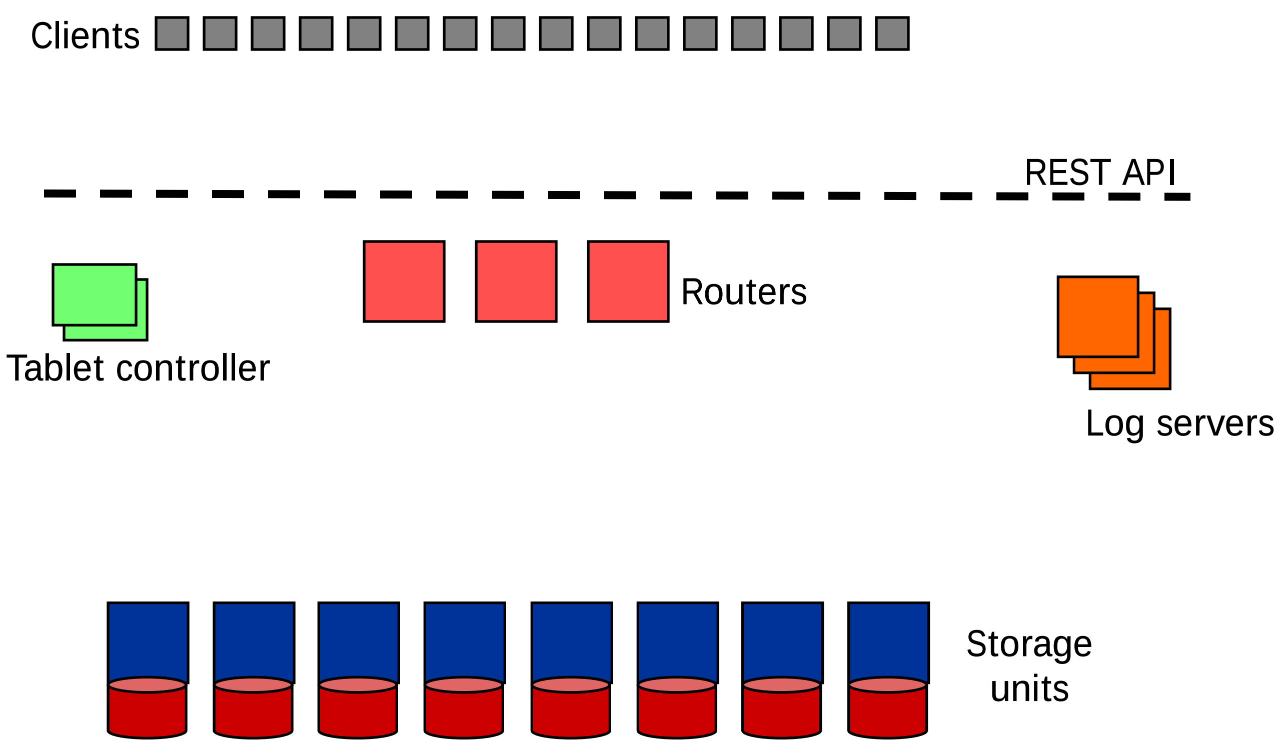
Google MegaStore

- Transactions across entity groups
  - Entity-group: hierarchically linked records
    - Ramakris
    - Ramakris.preferences
    - Ramakris.posts
    - Ramakris.posts.aug-24-09
  - Can transactionally update multiple records within an entity group
    - Records may be on different servers
    - Use Paxos to ensure ACID, deal with server failures
  - Can join records within an entity group
  - Reportedly, moving to ordered, async replication w/o ACID
- Other details
  - Built on top of BigTable
  - Supports schemas, column partitioning, some indexing

Phil Bernstein, http://perspectives.mvdirona.com/2008/07/10/GoogleMegastore.aspx

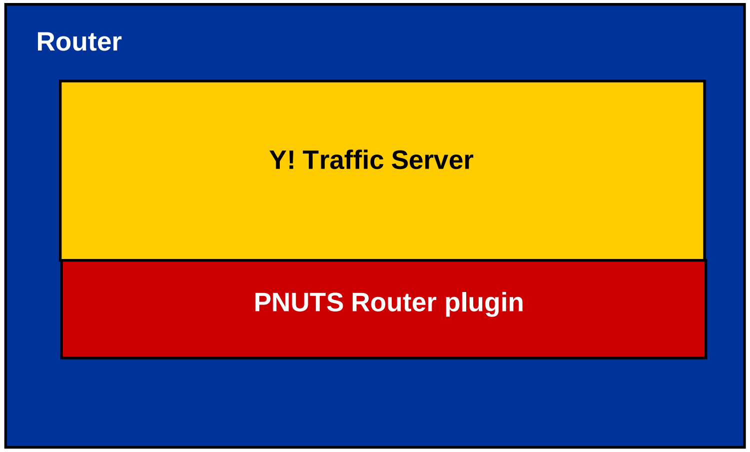
PNUTS

Architecture



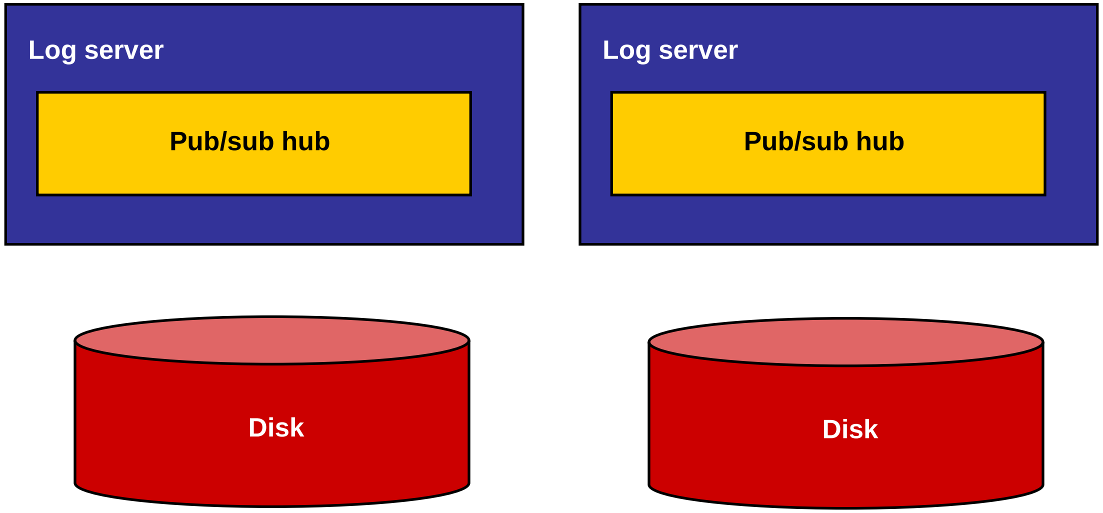
Routers

- Direct requests to storage unit
  - Decouple client from storage layer
    - Easier to move data, add/remove servers, etc.
  - Tradeoff: Some latency to get increased flexibility



Msg/Log Server

- Topic-based, reliable publish/subscribe
  – Provides reliable logging
  – Provides intra- and inter-datacenter replication



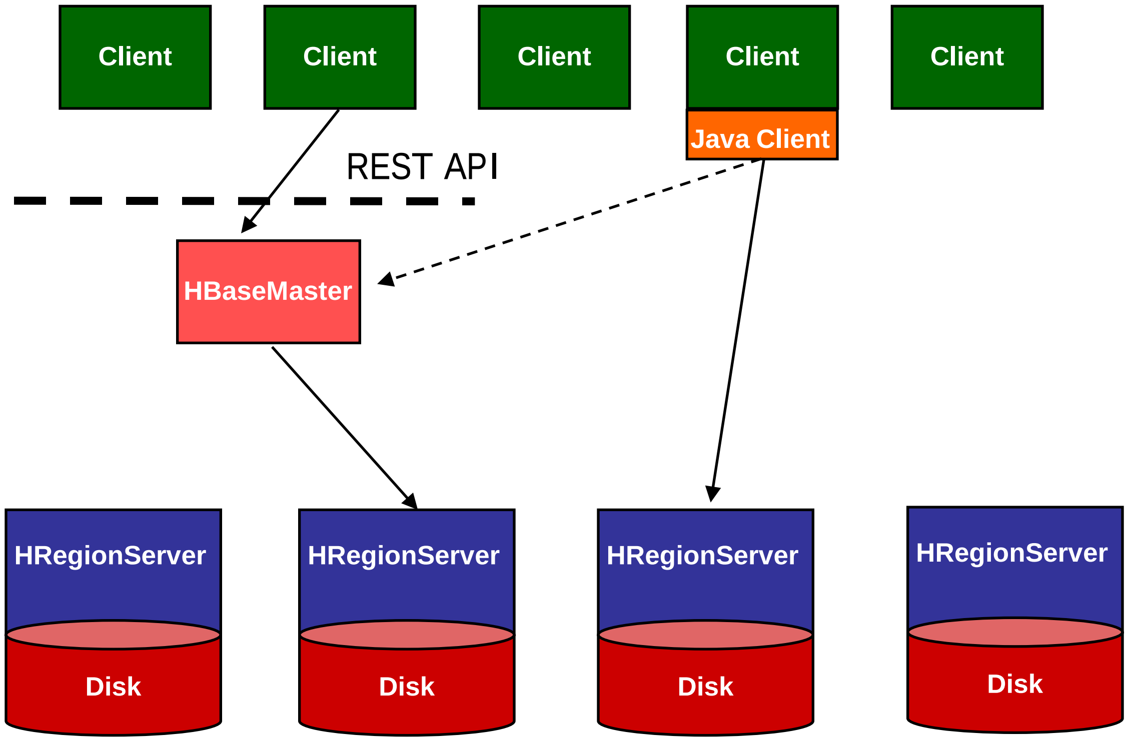
Pros and Cons

- Pros
  - Reliable geo-replication
  - Scalable consistency model
  - Elastic scaling
  - Easy load balancing

- Cons
  - System complexity relative to sharded MySQL to support geo-replication, consistency, etc.
  - Latency added by router

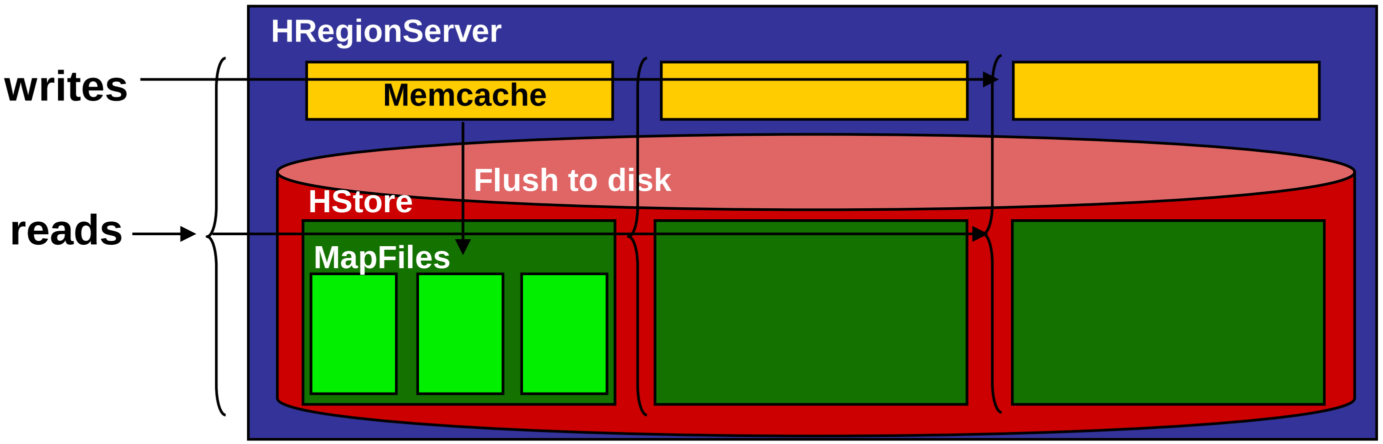
HBASE

Architecture



HRegion Server

- Records partitioned by column family into HStores
  – Each HStore contains many MapFiles
- All writes to HStore applied to single memcache
- Reads consult MapFiles and memcache
- Memcaches flushed as MapFiles (HDFS files) when full
- Compactions limit number of MapFiles



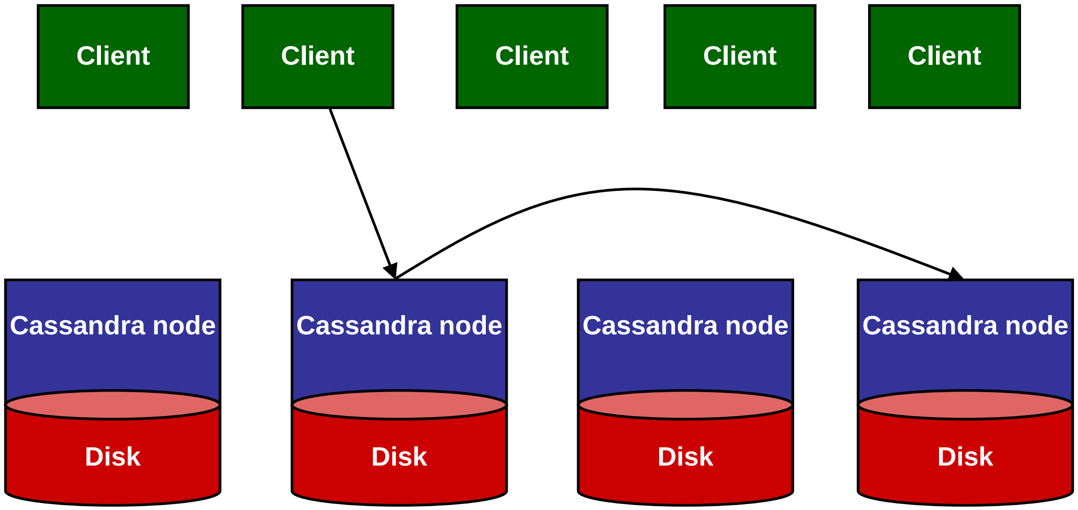
Pros and Cons

- Pros
  - Log-based storage for high write throughput
  - Elastic scaling
  - Easy load balancing
  - Column storage for OLAP workloads
- Cons
  - Writes not immediately persisted to disk
  - Reads cross multiple disk, memory locations
  - No geo-replication
  - Latency/bottleneck of HBaseMaster when using REST

CASSANDRA

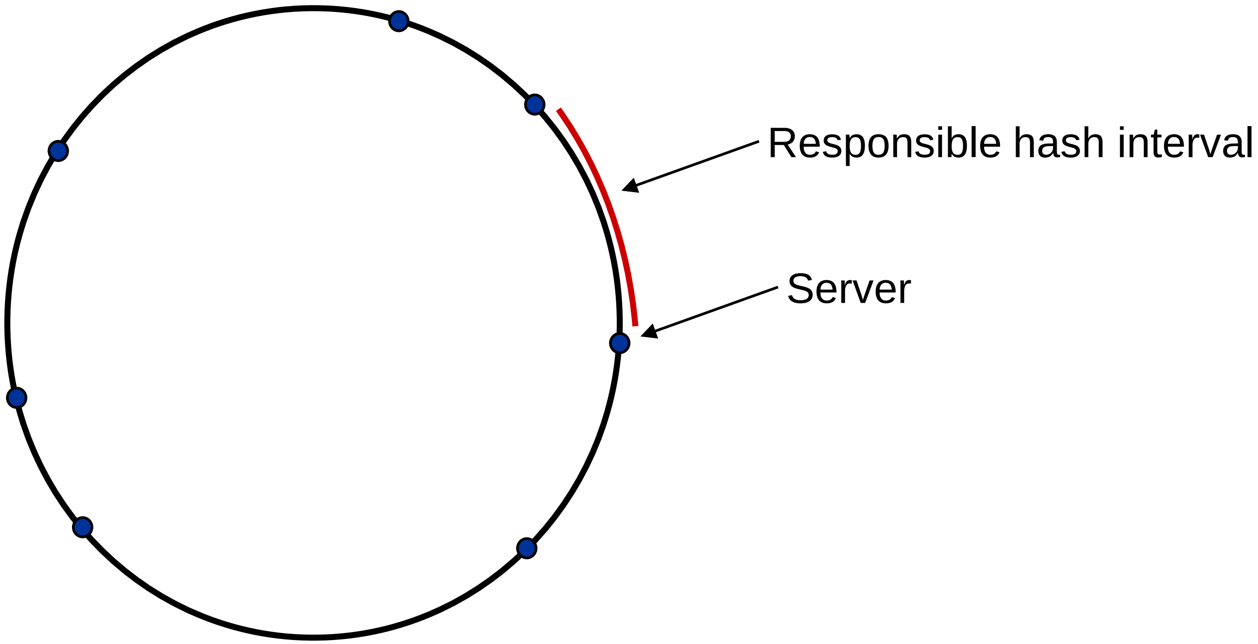
Architecture

- Facebook's storage system
  - BigTable data model
  - Dynamo partitioning and consistency model
  - Peer-to-peer architecture



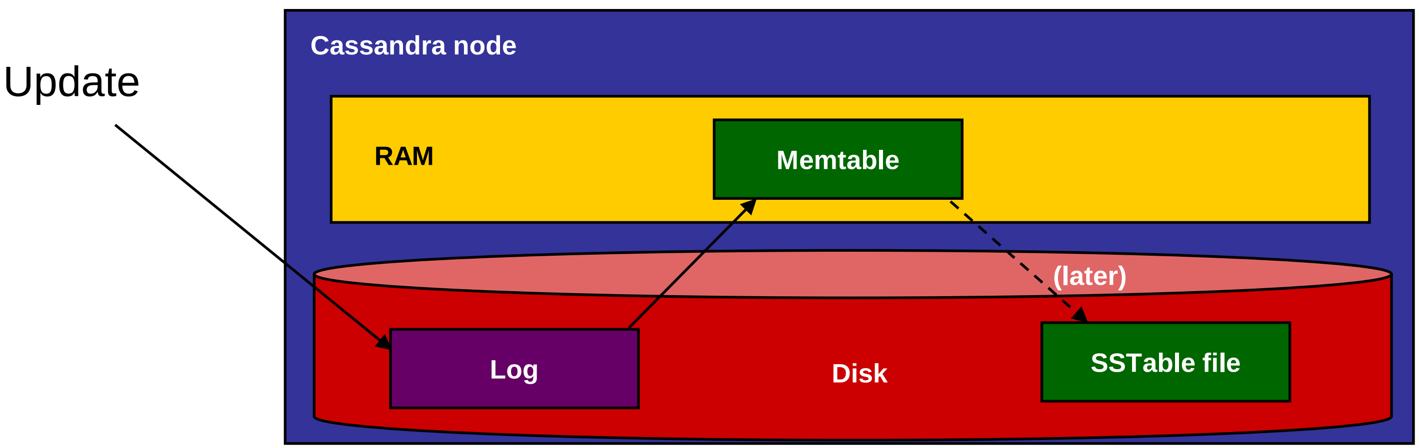
Routing

- Consistent hashing, like Dynamo or Chord
  - Server position = hash(serverid)
  - Content position = hash(contentid)
  - Server responsible for all content in a hash interval



Cassandra Server

- Writes go to log and memory table
- Periodically memory table merged with disk table



Pros and Cons

- Pros
  - Elastic scalability
  - Easy management
    - Peer-to-peer configuration
  - BigTable model is nice
    - Flexible schema, column groups for partitioning, versioning, etc.
  - Eventual consistency is scalable

- Cons
  - Eventual consistency is hard to program against
  - No built-in support for geo-replication
    - Gossip can work, but is not really optimized for, cross-datacenter
  - Load balancing?
    - Consistent hashing limits options
  - System complexity
    - P2P systems are complex; have complex corner cases

Cassandra Findings

- Tunable memtable size
  - Can have large memtable flushed less frequently, or small memtable flushed frequently
  - Tradeoff is throughput versus recovery time
    - Larger memtable will require fewer flushes, but will take a long time to recover after a failure
    - With 1GB memtable: 45 mins to 1 hour to restart
- Can turn off log flushing
  - Risk loss of durability
- Replication is still synchronous with the write
  - Durable if updates propagated to other servers that don’t fail

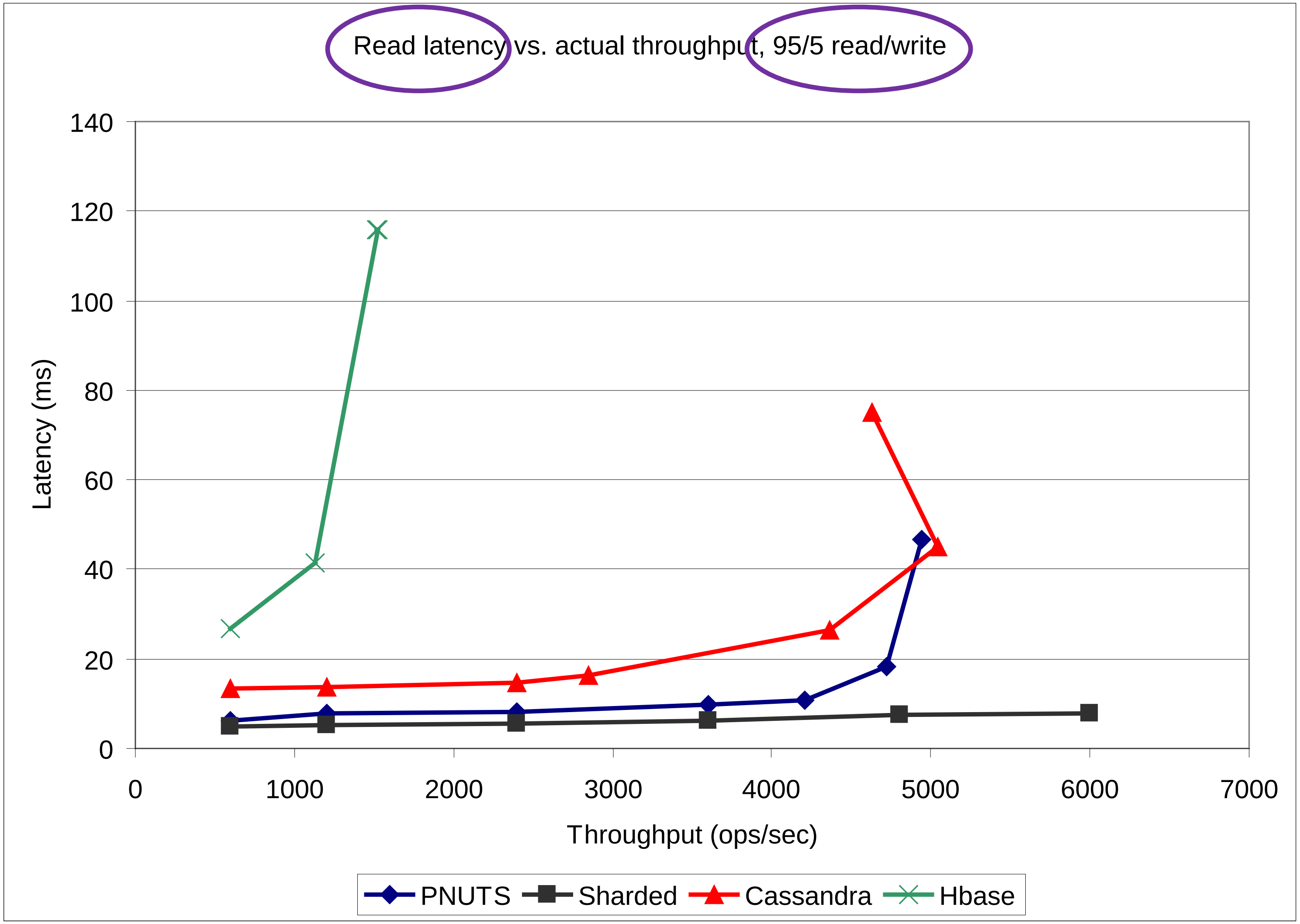
Thanks to Ryan Rawson & J.D. Cryans for advice on HBase configuration, and Jonathan Ellis on Cassandra

NUMBERS

Overview

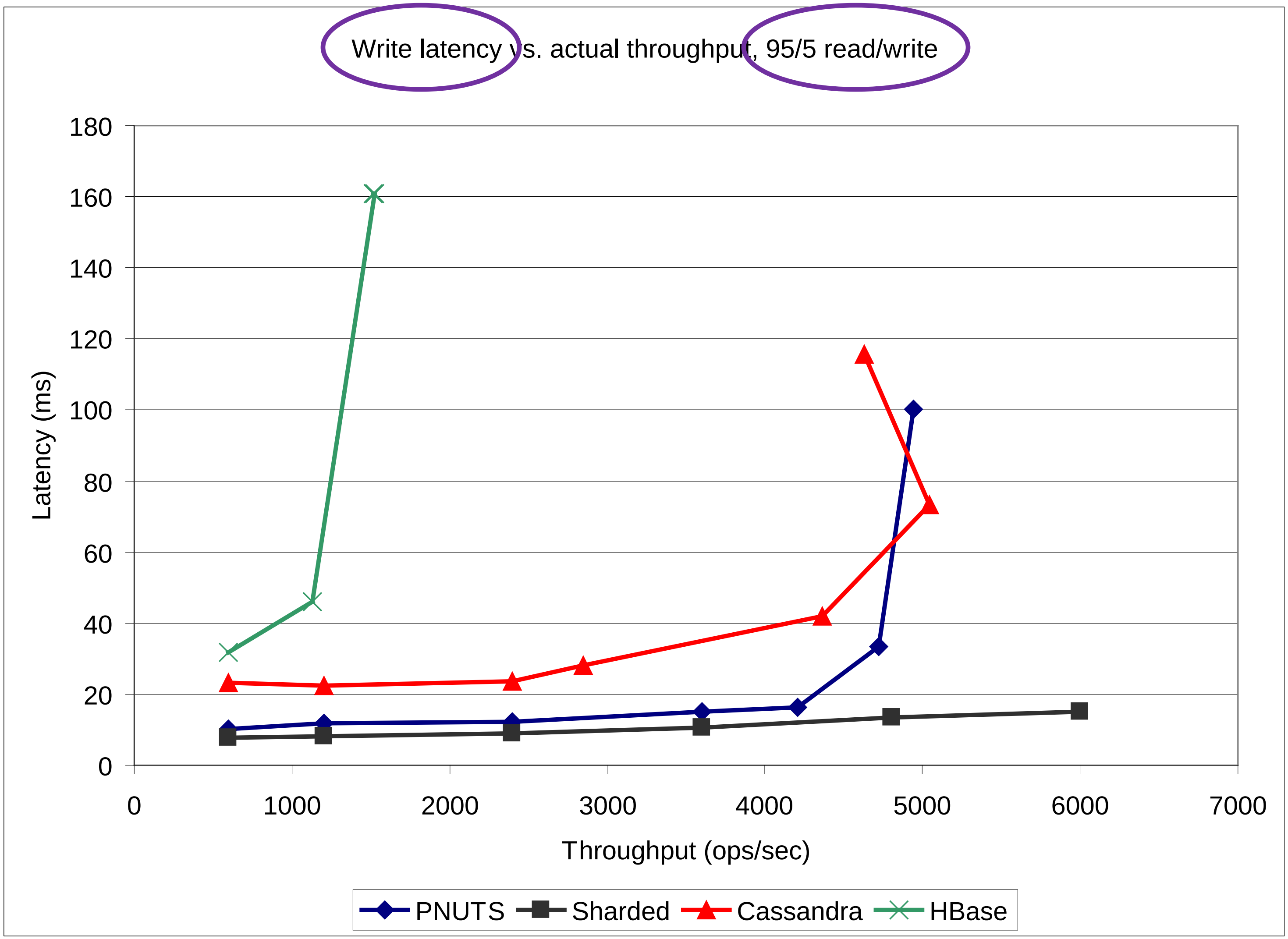
- Setup
  - Six server-class machines
    - 8 cores (2 x quadcore) 2.5 GHz CPUs, RHEL 4, Gigabit ethernet
    - 8 GB RAM
    - 6 x 146GB 15K RPM SAS drives in RAID 1+0
  - Plus extra machines for clients, routers, controllers, etc.
- Workloads
  - 120 million 1 KB records = 20 GB per server
  - Write heavy workload: 50/50 read/update
  - Read heavy workload: 95/5 read/update
- Metrics
  - Latency versus throughput curves
- Caveats
  - Write performance would be improved for PNUTS, Sharded ,and Cassandra with a dedicated log disk
  - We tuned each system as well as we knew how

Results



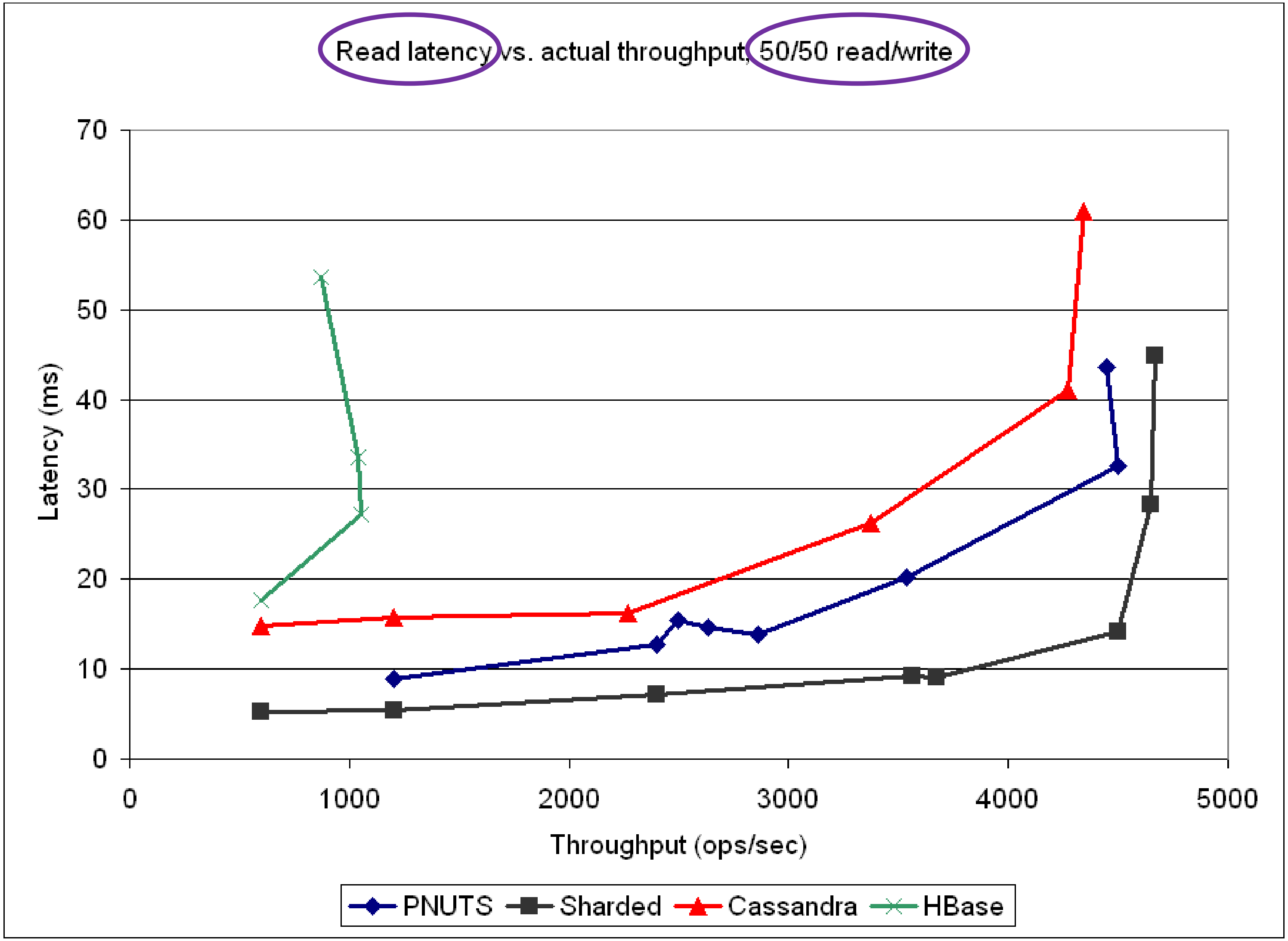
Read latency vs. actual throughput, 95/5 read/write
140
120
100
80
60
40
20
0
Latency (ms)
0
1000
2000
3000
4000
5000
6000
7000
Throughput (ops/sec)
PNUTS
Sharded
Cassandra
Hbase

Results



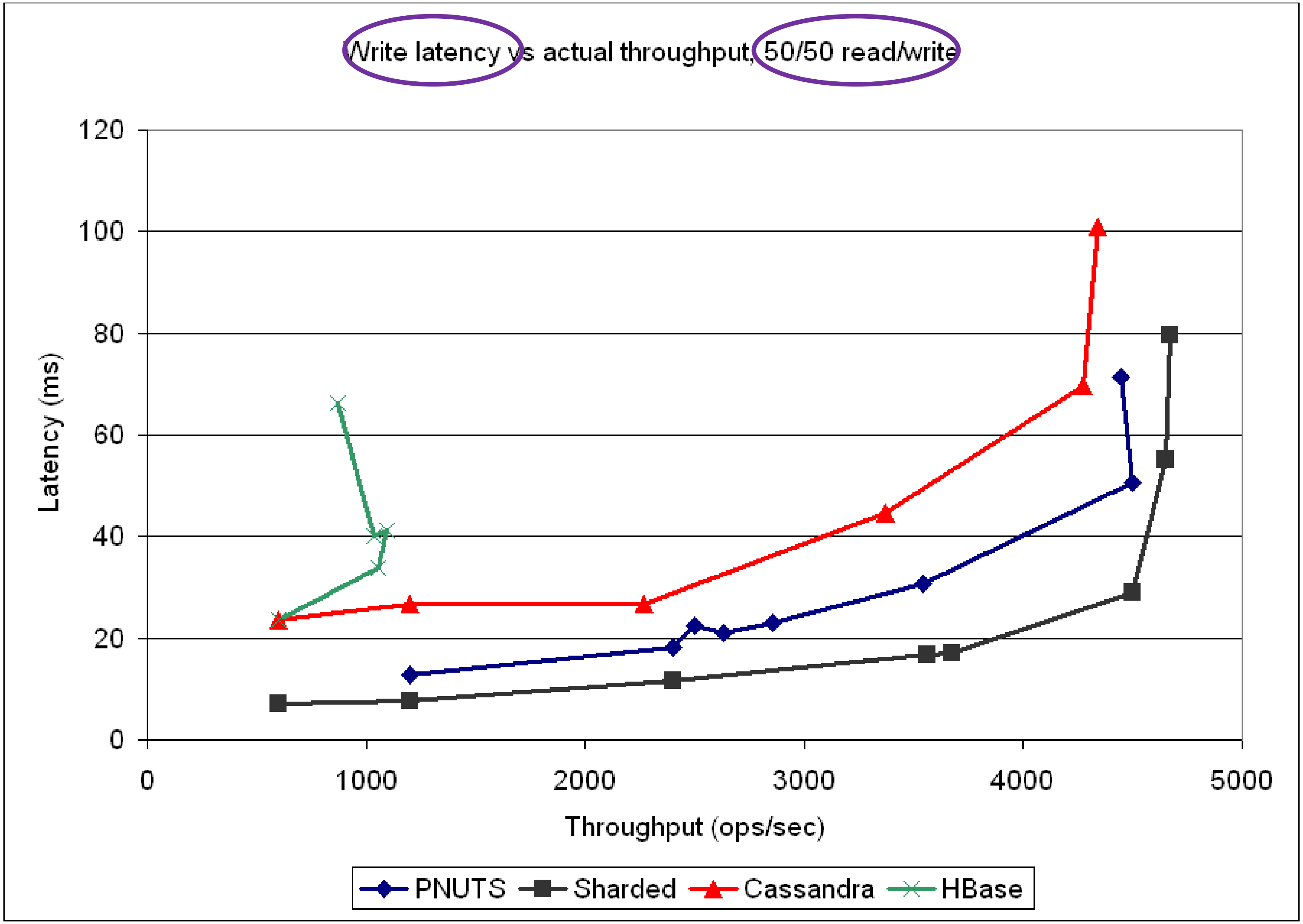
Write latency vs. actual throughput, 95/5 read/write
Latency (ms)
180
160
140
120
100
80
60
40
20
0
0
1000
2000
3000
4000
5000
6000
7000
Throughput (ops/sec)
PNUTS
Sharded
Cassandra
HBase

Results



Read latency vs. actual throughput, 50/50 read/write
70
60
50
40
30
20
10
0
Latency (ms)
0
1000
2000
3000
4000
5000
Throughput (ops/sec)
PNUTS
Sharded
Cassandra
HBase

Results



Write latency vs actual throughput, 50/50 read/write
120
100
80
60
40
20
0
Latency (ms)
0
1000
2000
3000
4000
5000
Throughput (ops/sec)
PNUTS
Sharded
Cassandra
HBase

Qualitative Comparison

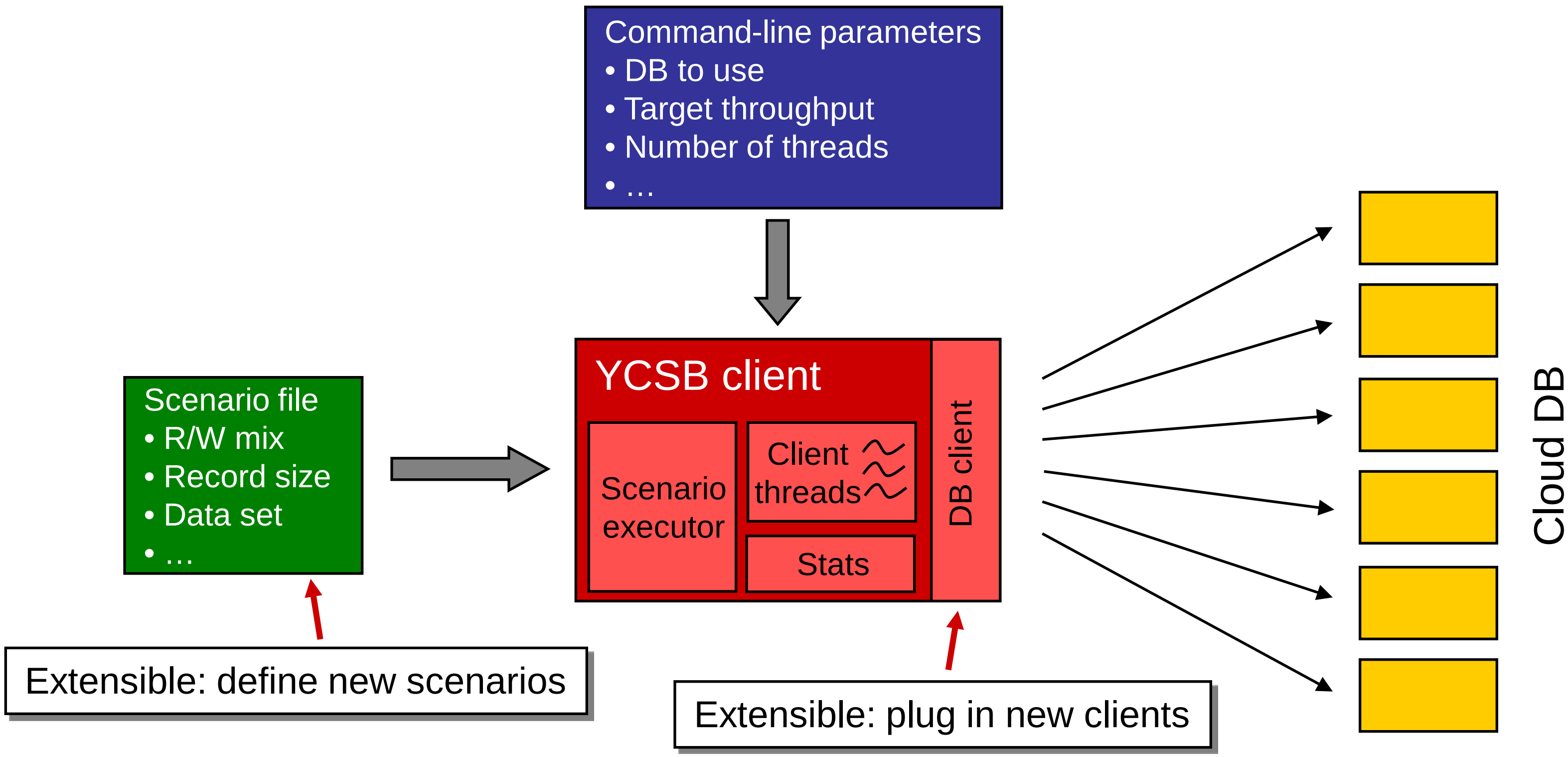
- Storage Layer
  - File Based: HBase, Cassandra
  - MySQL: PNUTS, Sharded MySQL
- Write Persistence
  - Writes committed synchronously to disk: PNUTS, Cassandra, Sharded MySQL
  - Writes flushed asynchronously to disk: HBase (current version)
- Read Pattern
  - Find record in MySQL (disk or buffer pool): PNUTS, Sharded MySQL
  - Find record and deltas in memory and on disk: HBase, Cassandra

Qualitative Comparison

- Replication (not yet utilized in benchmarks)
  - Intra-region: HBase, Cassandra
  - Inter- and intra-region: PNUTS
  - Inter- and intra-region: MySQL (but not guaranteed)
- Mapping record to server
  - Router: PNUTS, HBase (with REST API)
  - Client holds mapping: HBase (java library), Sharded MySQL
  - P2P: Cassandra

YCS Benchmark

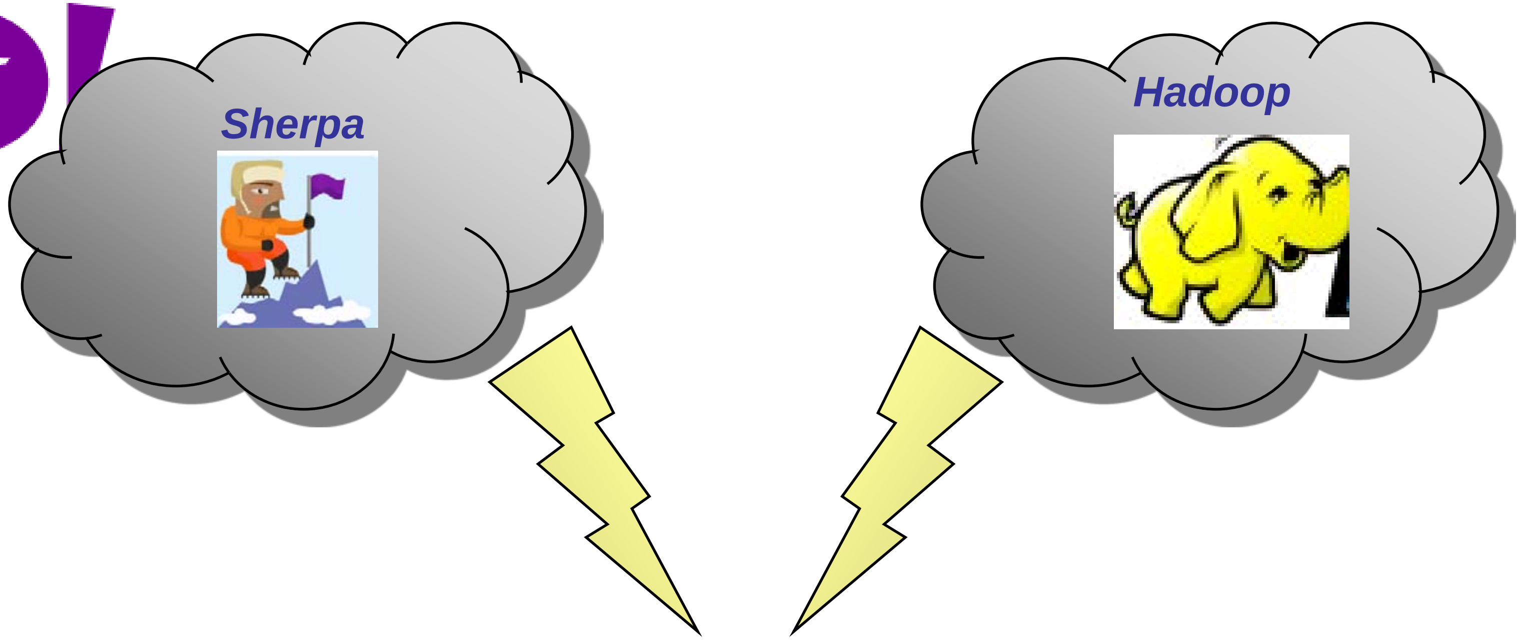
- Will distribute Java application, plus extensible benchmark suite
  - Many systems have Java APIs
  - Non-Java systems can support REST



Benchmark Tiers

- Tier 1: Cluster performance
  - Tests fundamental design independent of replication/scale-out/availability
- Tier 2: Replication
  - Local replication
  - Geographic replication
- Tier 3: Scale out
  - Scale out the number of machines by a factor of 10
  - Scale out the number of replicas
- Tier 4: Availability
  - Kill system components

If you're interested in this, please contact me or cooperb@yahoo-inc.com



Shadoop

Adam Silberstein and the Sherpa team in Y! Research and CCDI

Sherpa vs. HDFS

- Sherpa optimized for low-latency record-level access: ***B-trees***
- HDFS optimized for batch-oriented access: ***File system***
- At 50,000 feet the data stores appear similar
  - ➢ Is Sherpa a reasonable backend for Hadoop for Sherpa users?

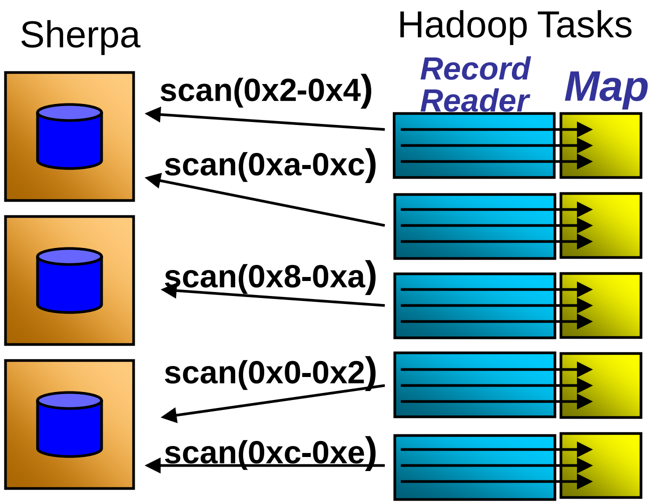
Sherpa HDFS

Shadoop Goals

1. Provide a batch-processing framework for Sherpa using Hadoop
2. Minimize/characterize penalty for reading/writing Sherpa vs. HDFS

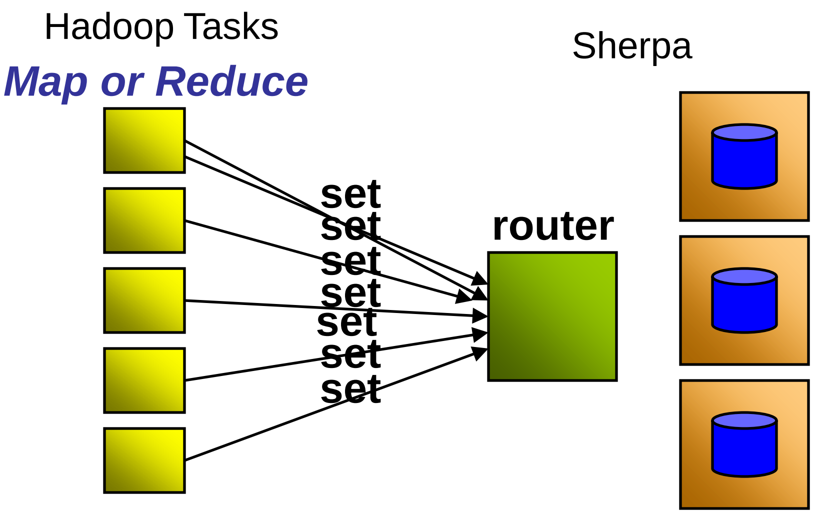
Building Shadoop

Input



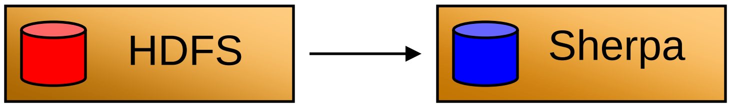
1. Split Sherpa table into hash ranges
2. Each Hadoop task assigned a range
3. Task uses Sherpa scan to retrieve records in range
4. Task feeds scan results and feeds records to map function

Output

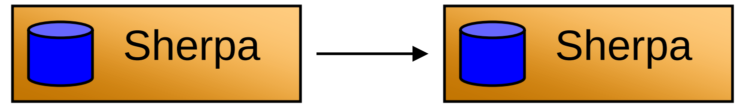


1. Call Sherpa set to write output
   1. No DOT range-clustering
   2. Record at a time insert

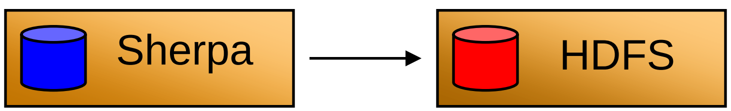
Use Cases



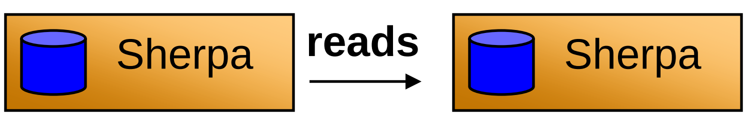
Bulk Load Sherpa Tables
Source data in HDFS
Map/Reduce, Pig
Output is servable content



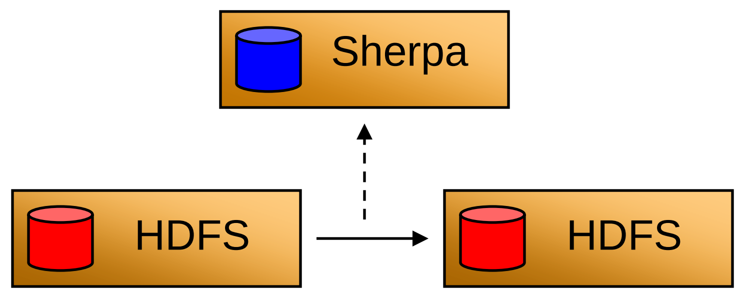
Migrate Sherpa Tables
Between farms, code versions
Map/Reduce, Pig
Output is servable content



Map/Reduce, Pig
Output is input to further analysis



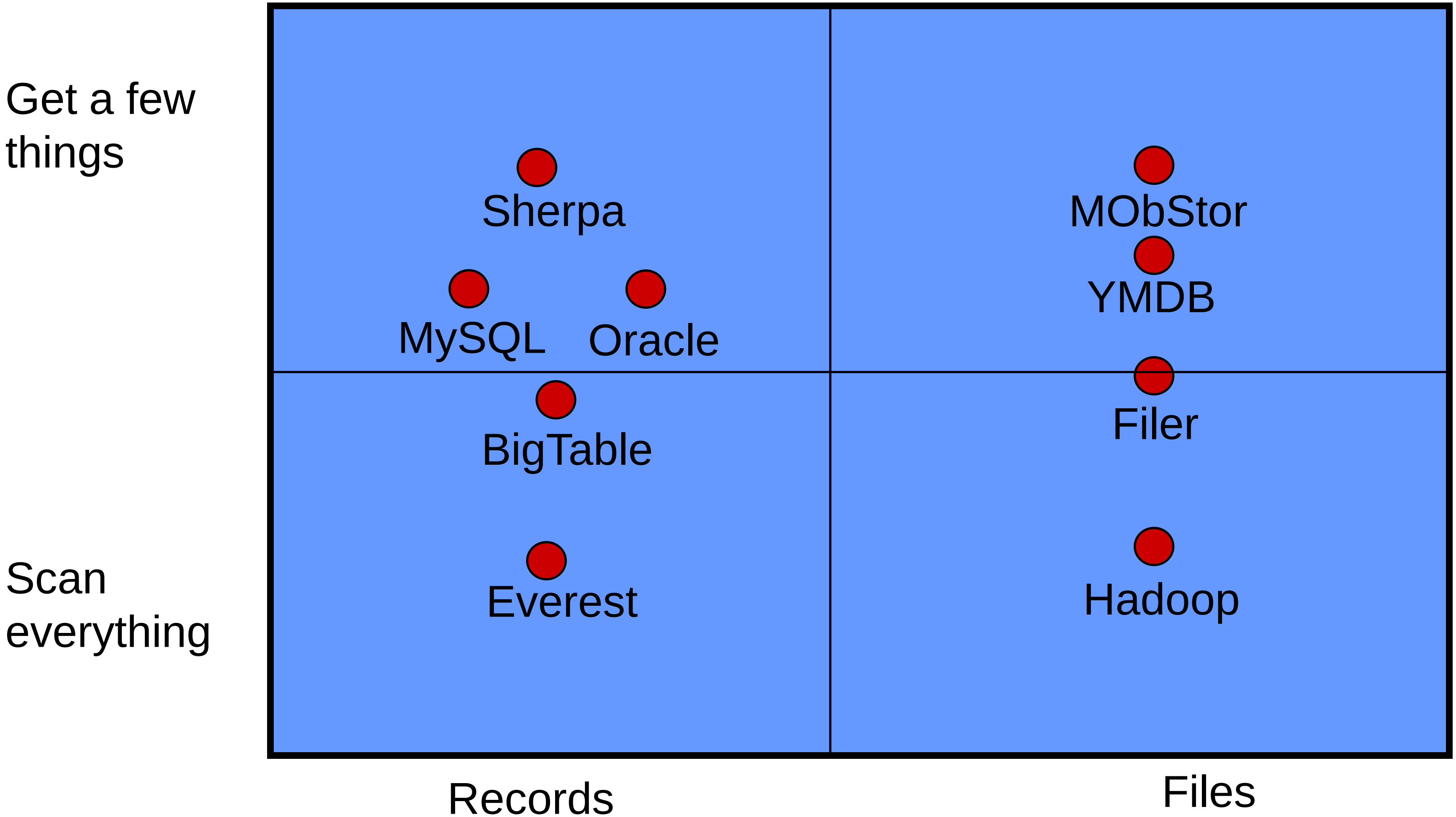
Validator Ops Tool
Ensure replicas are consistent



Sherpa as Hadoop "cache"
Standard Hadoop in HDFS,
using Sherpa as a shared cache

SYSTEMS
IN CONTEXT

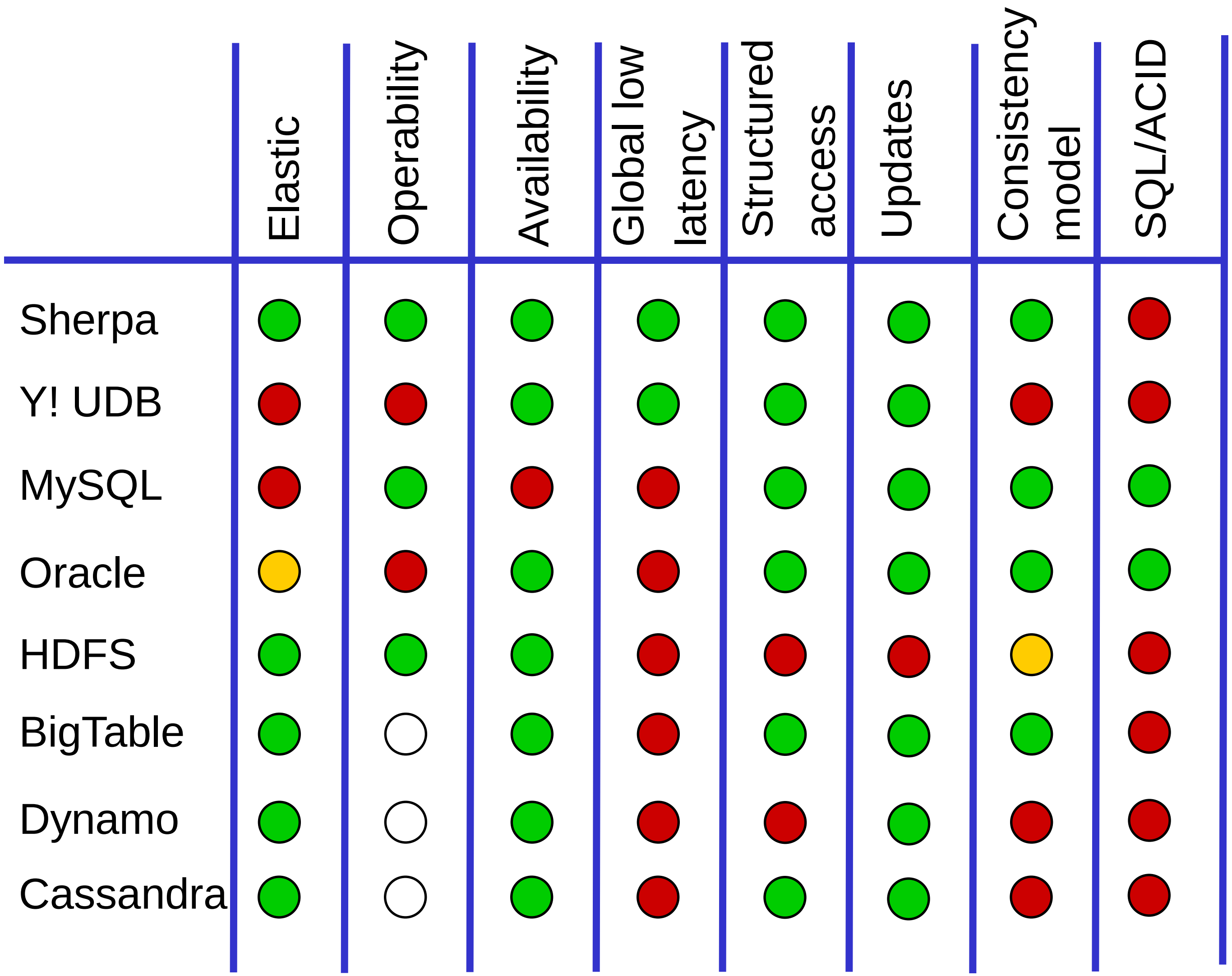
Application Design Space

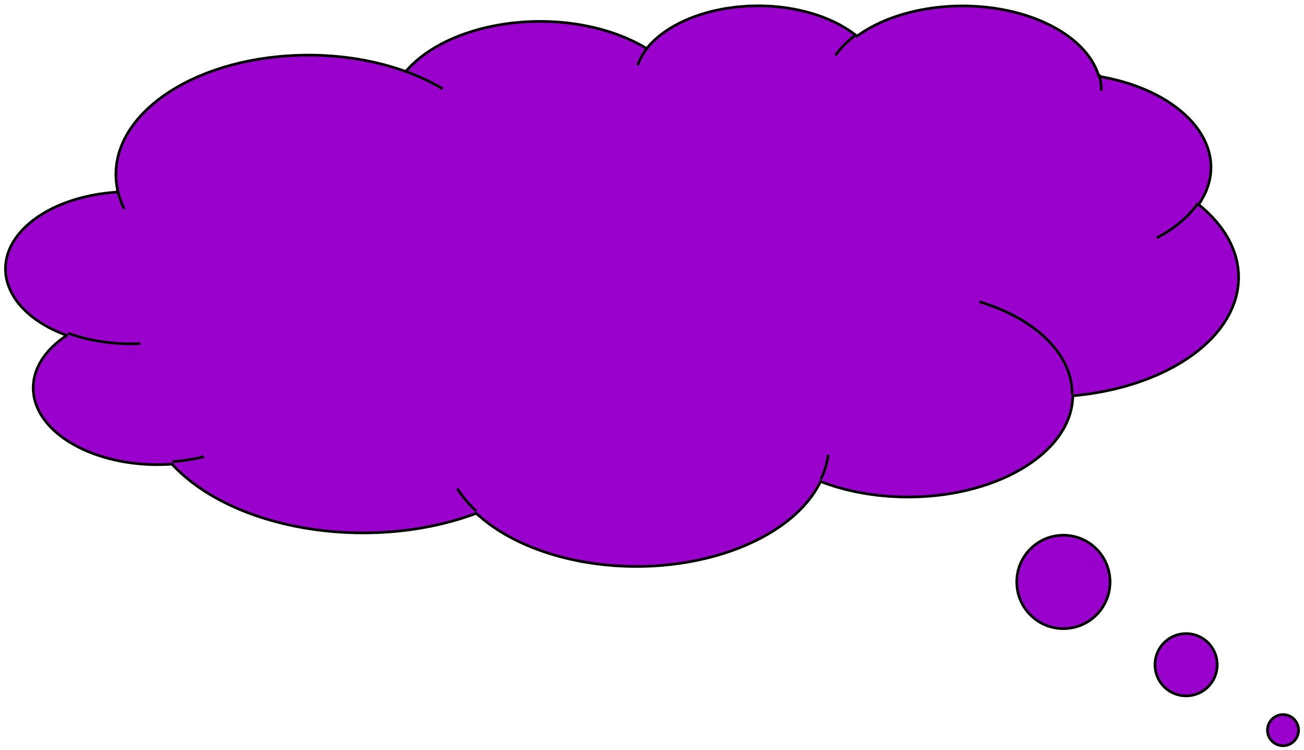


Comparison Matrix

	Partitioning			Availability		Replication			Storage	
	Hash/sort	Dynamic	Routing	Failures handled	During failure	Sync/async	Local/geo	Consistency	Durability	Reads/ writes
PNUTS	H+S	Y	Rtr	Colo+ server	Read+ write	Async	Local+ geo	Timeline + Eventual	Double WAL	Buffer pages
MySQL	H+S	N	Cli	Colo+ server	Read	Async	Local+ nearby	ACID	WAL	Buffer pages
HDFS	Other	Y	Rtr	Colo+ server	Read+ write	Sync	Local+ nearby	N/A (no updates)	Triple replication	Files
BigTable	S	Y	Rtr	Colo+ server	Read+ write	Sync	Local+ nearby	Multi- version	Triple replication	LSM/ SSTable
Dynamo	H	Y	P2P	Colo+ server	Read+ write	Async	Local+ nearby	Eventual	WAL	Buffer pages
Cassandra	H+S	Y	P2P	Colo+ server	Read+ write	Sync+ Async	Local+ nearby	Eventual	Triple WAL	LSM/ SSTable
Megastore	S	Y	Rtr	Colo+ server	Read+ write	Sync	Local+ nearby	ACID/ other	Triple replication	LSM/ SSTable
Azure	S	N	Cli	Server	Read+ write	Sync	Local	ACID	WAL	Buffer pages

Comparison Matrix





QUESTIONS?