NOSQL

BILL HOWE

UNIVERSITY OF WASHINGTON

teaching
datascience
.org

WHERE WE ARE

Data science

1. Data Preparation (at scale)
2. Analytics
3. Communication

Databases

- Key ideas: Relational algebra, physical/logical data independence

MapReduce

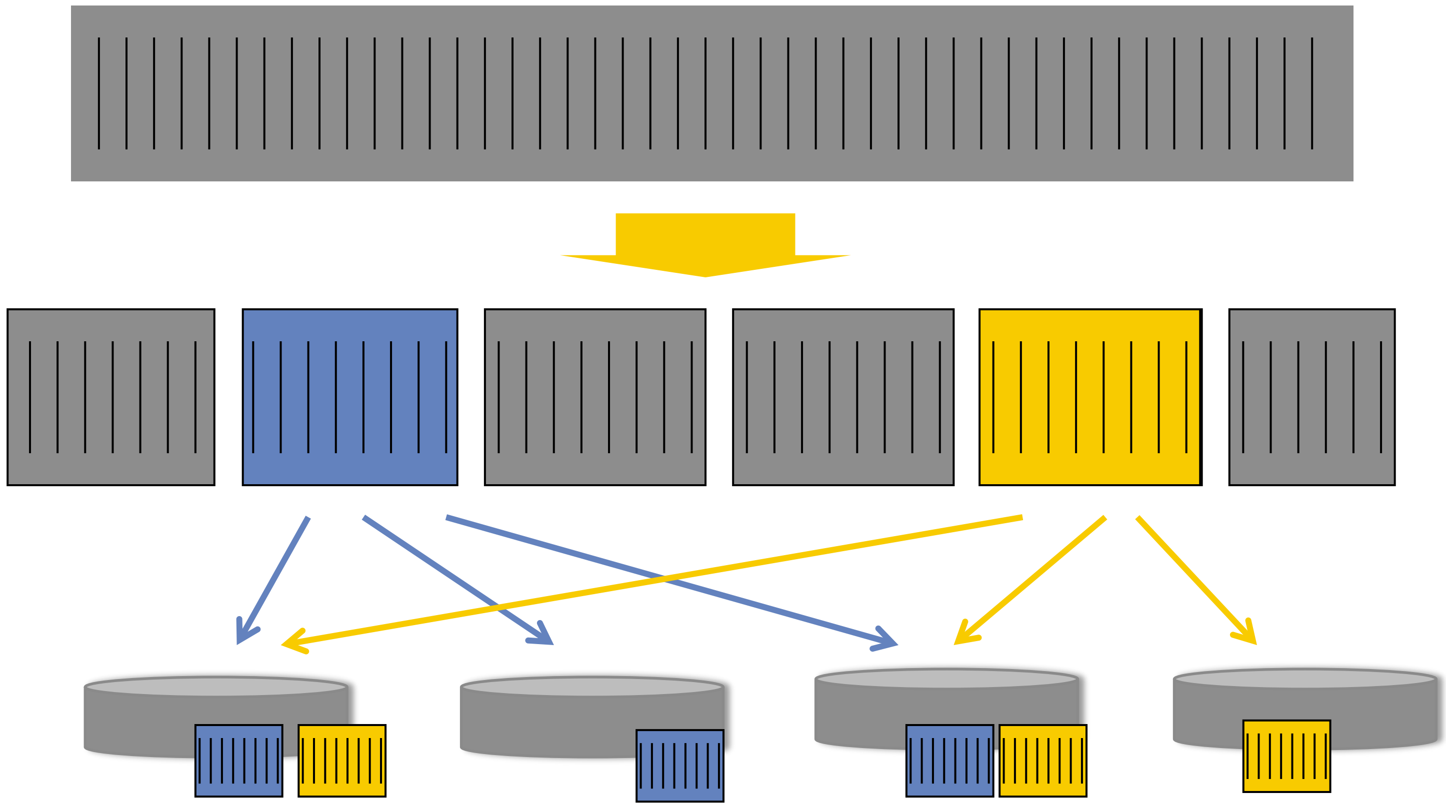
- Key ideas: Fault tolerance, no loading, direct programming on "in site" data

NOSQL AND RELATED SYSTEM, BY FEATURE

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	O	✔	✔	✔	✔	✔	✔	✔	tables	sql-like
2003	memcached	✔	✔	O	O	O	O	O	O	key-val	nosql
2004	MapReduce	✔	O	O	O	✔	O	O	O	key-val	batch
2005	CouchDB	✔	✔	✔	record	MR	O	✔	O	document	nosql
2006	BigTable/ Hbase	✔	✔	✔	record	compat. w/MR	/	O	O	ext. record	nosql
2007	MongoDB	✔	✔	✔	EC, record	O	O	O	O	document	nosql
2007	Dynamo	✔	✔	O	O	O	O	O	O	ext. record	nosql
2008	Pig	✔	O	O	O	✔	/	O	✔	tables	sql-like
2008	HIVE	✔	O	O	O	✔	✔	O	✔	tables	sql-like
2008	Cassandra	✔	✔	✔	EC, record	O	✔	✔	O	key-val	nosql
2009	Voldemort	✔	✔	O	EC, record	O	O	O	O	key-val	nosql
2009	Riak	✔	✔	✔	EC, record	MR	O			key-val	nosql
2010	Dremel	✔	O	O	O	/	✔	O	✔	tables	sql-like
2011	Megastore	✔	✔	✔	entity groups	O	/	O	/	tables	nosql
2011	Tenzing	✔	O	O	O	O	✔	✔	✔	tables	sql-like
2011	Spark/Shark	✔	O	O	O	✔	✔	O	✔	tables	sql-like
2012	Spanner	✔	✔	✔	✔	?	✔	✔	✔	tables	sql-like
2013	Impala	✔	O	O	O	✔	✔	O	✔	tables	sql-like

SCALE IS THE PRIMARY MOTIVATION

Year	System/ Paper	Scale to 1000s	Primary Index	Secondar y Indexes	Transactions	Joins/ Analytics	Integrity Constraint s	Views	Language/ Algebra	Data model	my label
1971	RDBMS	O	✔	✔	✔	✔	✔	✔	✔	tables	sql-like
2003	memcached	✔	✔	O	O	O	O	O	O	key-val	nosql
2004	MapReduce	✔	O	O	O	✔	O	O	O	key-val	batch
2005	CouchDB	✔	✔	✔	record	MR	O	✔	O	document	nosql
2006	BigTable (Hbase)	✔	✔	✔	record	compat. w/MR	/	O	O	ext. record	nosql
2007	MongoDB	✔	✔	✔	EC, record	O	O	O	O	document	nosql
2007	Dynamo	✔	✔	O	O	O	O	O	O	ext. record	nosql
2008	Pig	✔	O	O	O	✔	/	O	✔	tables	sql-like
2008	HIVE	✔	O	O	O	✔	✔	O	✔	tables	sql-like
2008	Cassandra	✔	✔	✔	EC, record	O	✔	✔	O	key-val	nosql
2009	Voldemort	✔	✔	O	EC, record	O	O	O	O	key-val	nosql
2009	Riak	✔	✔	✔	EC, record	MR	O			key-val	nosql
2010	Dremel	✔	O	O	O	/	✔	O	✔	tables	sql-like
2011	Megastore	✔	✔	✔	entity groups	O	/	O	/	tables	nosql
2011	Tenzing	✔	O	O	O	O	✔	✔	✔	tables	sql-like
2011	Spark/Shark	✔	O	O	O	✔	✔	O	✔	tables	sql-like
2012	Spanner	✔	✔	✔	✔	?	✔	✔	✔	tables	sql-like
2012	Accumulo	✔	✔	✔	record	compat. w/MR	/	O	O	ext. record	nosql
2013	Impala	✔	O	O	O	✔	✔	O	✔	tables	sql-like



1) We need to ensure high availability

2) We also want to support updates

EXAMPLE

User: Sue Friends: Joe, Kai, … Status: “Headed to new Bond flick” Wall: “…”, “…”	User: Kai Friends: Sue, … Status: “Done for tonight” Wall: “…”, “…”	User: Joe Friends: Sue, … Status: “I’m sleepy” Wall: “…”, “…”

Write: Update Sue’s status. Who sees the new status, and who sees the old one?

Databases: *“Everyone MUST see the same thing, either old or new, no matter how long it takes.”*

NoSQL: *“For large applications, we can’t afford to wait that long, and maybe it doesn’t matter anyway”*

EXAMPLE

Friends

Jim, Sue
Sue, Jim
Lin, Joe
Joe, Lin
Jim, Kai
Kai, Jim
Jim, Lin
Lin, Jim

Users

Jim
Sue
…

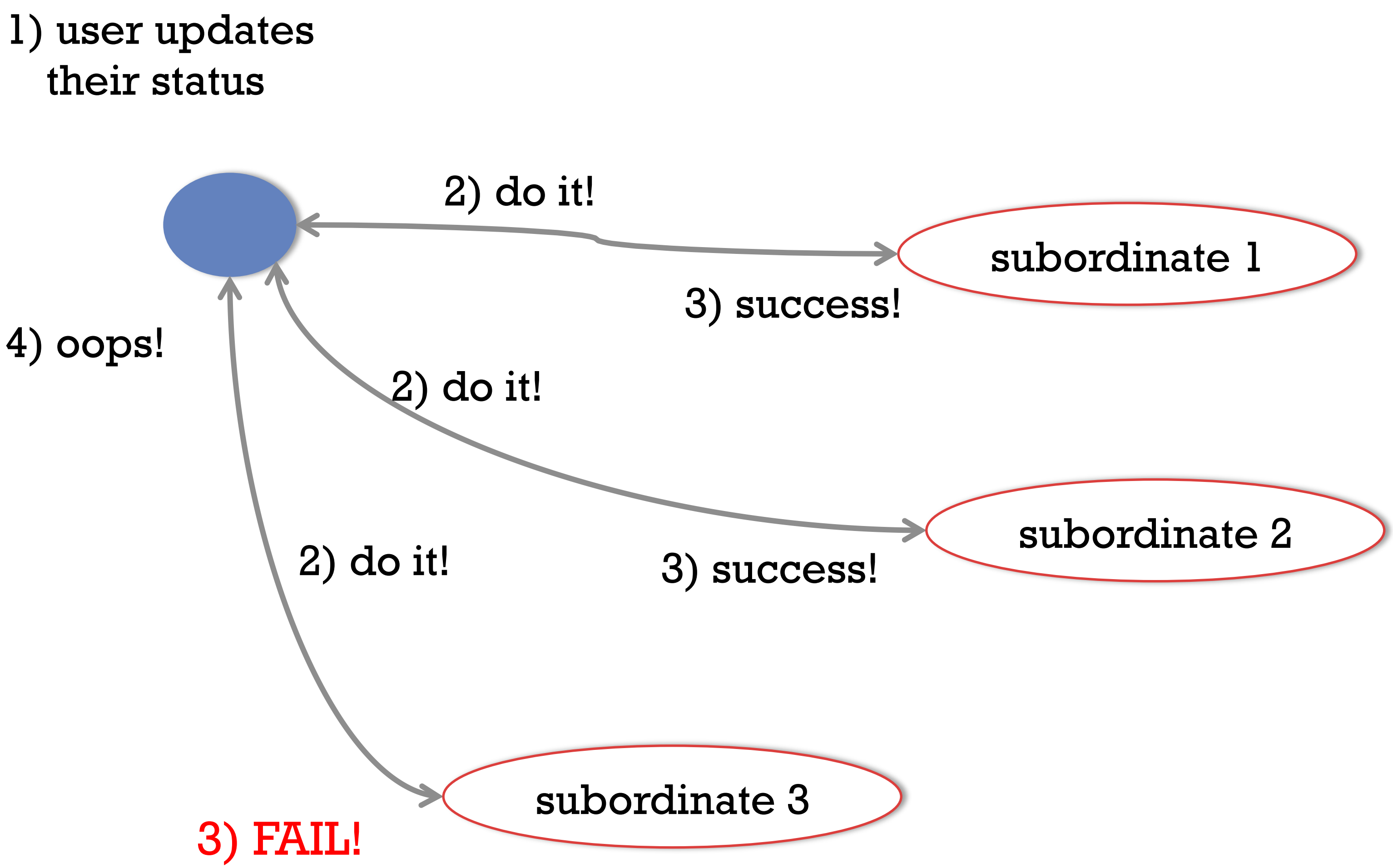
Posts

Sue: "headed to see new Bond flick"

Sue: "it was ok"

Kai: "I'm hungry"

TWO-PHASE COMMIT MOTIVATION



TWO-PHASE COMMIT

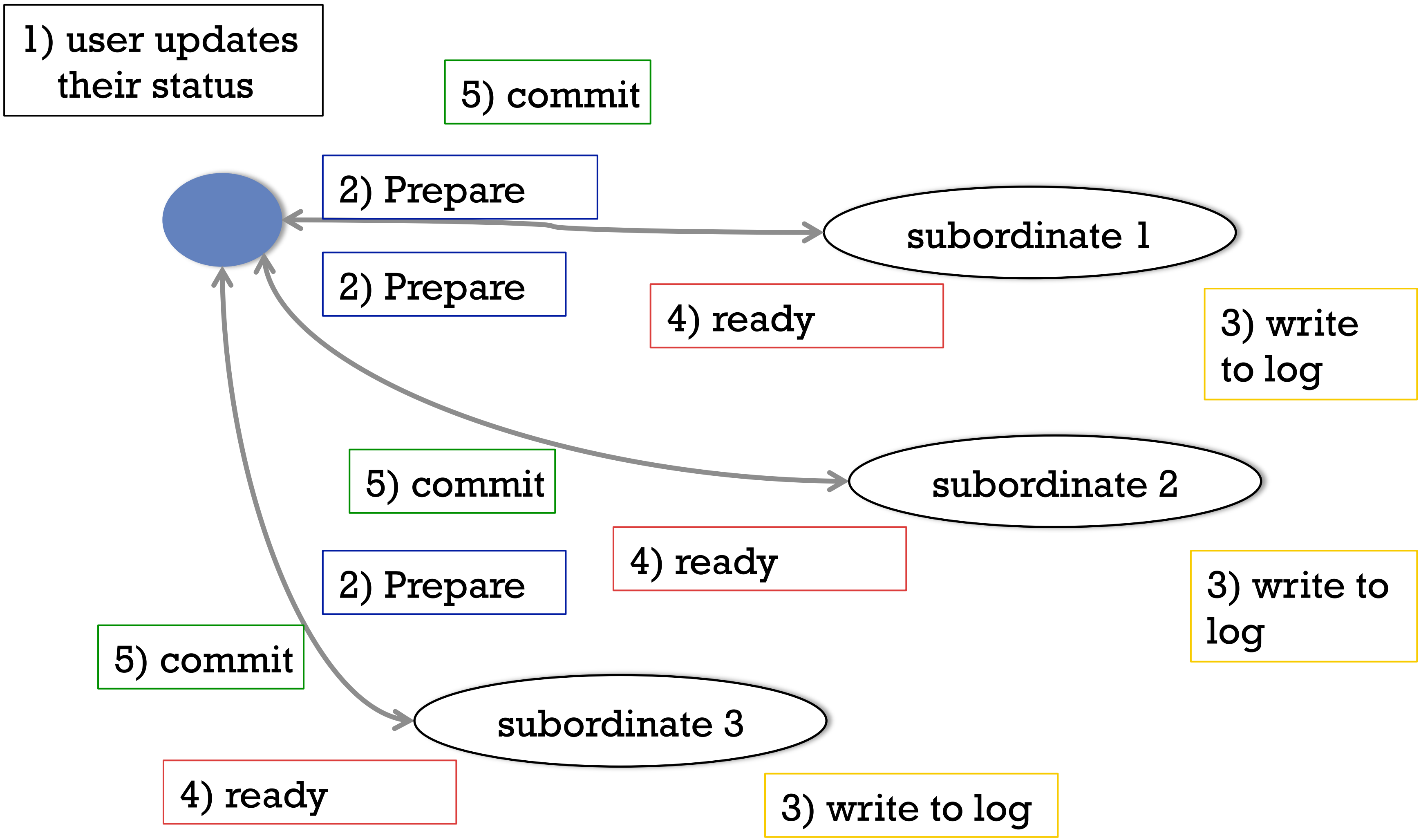
Phase 1:

- Coordinator Sends "Prepare to Commit"
- Subordinates make sure they can do so no matter what
  - Write the action to a log to tolerate failure
- Subordinates Reply "Ready to Commit"

Phase 2:

- If all subordinates ready, send "Commit"
- If anyone failed, send "Abort"

TWO-PHASE COMMIT



"EVENTUAL CONSISTENCY"

Write conflicts will eventually propagate throughout the system

- D. Terry et al., "Managing Update Conflicts in Bayou,a Weakly Connected Replicated Storage System", SOSP 1995

"We believe that applications must be aware that they may read weakly consistent data and also that their write operations may conflict with those of other users and applications."

"Moreover, applications must be revolved m the detection and resolution of conflicts since these naturally depend on the semantics of the application."

EVENTUAL CONSISTENCY

What the application sees in the meantime is sensitive to replication mechanics and difficult to predict

Contrast with RDBMS, Paxos: Immediate (or "strong") consistency, but there may be deadlocks

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
2003	memcached	✔	✔	O	O	O	O	O	O	key-val	nosql
2005	CouchDB	✔	✔	✔	record	MR	O	✔	O	document	nosql
2006	BigTable (Hbase)	✔	✔	✔	record	compat. w/MR	/	O	O	ext. record	nosql
2007	MongoDB	✔	✔	✔	EC, record	O	O	O	O	document	nosql
2007	Dynamo	✔	✔	O	O	O	O	O	O	key-val	nosql
2008	Cassandra	✔	✔	✔	EC, record	O	✔	✔	O	key-val	nosql
2009	Voldemort	✔	✔	O	EC, record	O	O	O	O	key-val	nosql
2009	Riak	✔	✔	✔	EC, record	MR	O			key-val	nosql
2011	Megastore	✔	✔	✔	entity groups	O	/	O	/	tables	nosql
2012	Accumulo	✔	✔	✔	record	compat. w/MR	/	O	O	ext. record	nosql
2012	Spanner	✔	✔	✔	✔	?	✔	✔	✔	tables	sql-like

CAP THEOREM [BREWER 2000, LYNCH 2002]

Consistency

- Do all applications see all the same data?

Availability

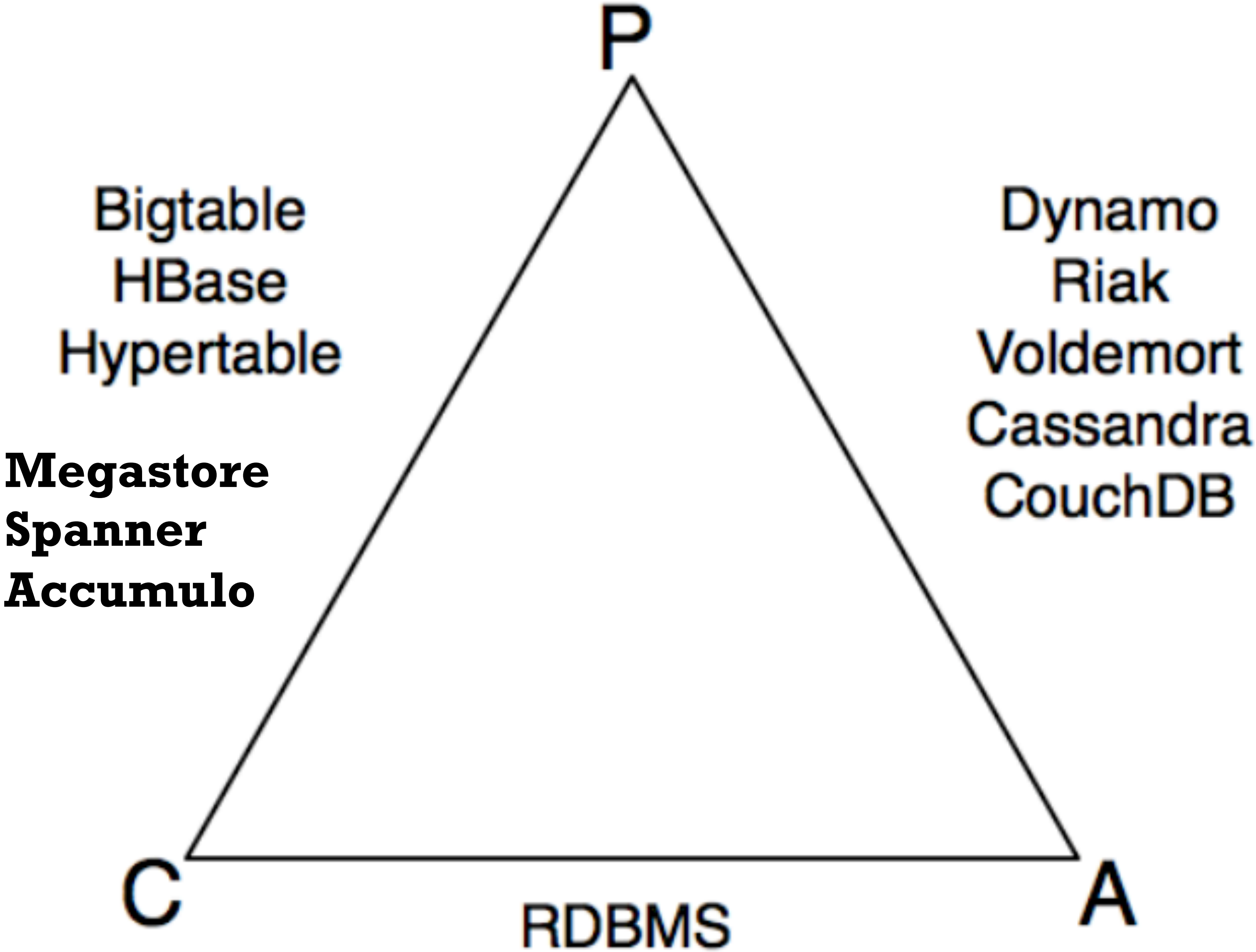
- If some nodes fail, does everything still work?

Partitioning

- If two sections of your system cannot talk to each other, can they make forward progress on their own?
  - If not, you sacrifice Availability
  - If so, you might have to sacrific Consistency – can't have everything

Conventional databases assume no partitioning – clusters were assumed to be small and local

NoSQL systems may sacrifice consistency



src: Shashank Tiwari

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	O	✔	✔	✔	✔	✔	✔	✔	**tables**	**sql-like**
2003	memcached	✔	✔	O	O	O	O	O	O	**key-val**	**nosql**
2004	MapReduce	✔	O	O	O	✔	O	O	O	**key-val**	**batch**
2005	CouchDB	✔	✔	✔	**record**	**MR**	O	✔	O	**document**	**nosql**
2006	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	/	O	O	**ext. record**	**nosql**
2007	MongoDB	✔	✔	✔	**EC, record**	O	O	O	O	**document**	**nosql**
2007	Dynamo	✔	✔	O	O	O	O	O	O	**key-val**	**nosql**
2008	Pig	✔	O	O	O	✔	/	O	✔	**tables**	**sql-like**
2008	HIVE	✔	O	O	O	✔	✔	O	✔	**tables**	**sql-like**
2008	Cassandra	✔	✔	✔	**EC, record**	O	✔	✔	O	**key-val**	**nosql**
2009	Voldemort	✔	✔	O	**EC, record**	O	O	O	O	**key-val**	**nosql**
2009	Riak	✔	✔	✔	**EC, record**	**MR**	O			**key-val**	**nosql**
2010	Dremel	✔	O	O	O	/	✔	O	✔	**tables**	**sql-like**
2011	Megastore	✔	✔	✔	**entity groups**	O	/	O	/	**tables**	**nosql**
2011	Tenzing	✔	O	O	O	O	✔	✔	✔	**tables**	**sql-like**
2011	Spark/Shark	✔	O	O	O	✔	✔	O	✔	**tables**	**sql-like**
2012	Spanner	✔	✔	✔	✔	**?**	✔	✔	✔	**tables**	**sql-like**
2012	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	/	O	O	**ext. record**	**nosql**
2013	Impala	✔	O	O	O	✔	✔	O	✔	**tables**	**sql-like**

Rick Cattel's clustering from
"Scalable SQL and NoSQL Data Stores"
SIGMOD Record, 2010

extensible record stores

document stores

key-value stores

TERMINOLOGY

Document = **nested values, extensible records (think XML or JSON)**

Extensible record = **families of attributes have a schema, but new attributes may be added**

Key-Value object = **a set of key-value pairs. No schema, no exposed nesting**

NOSQL FEATURES

Ability to horizontally scale "simple operation" throughput over many servers

- Simple = key lookups, read/write of 1 or few records

The ability to replicate and partition data over many servers

- Consider "sharding" and "horizontal partitioning" to be synonyms

A simple API – no query language

A weaker concurrency model than ACID transactions

Efficient use of distributed indexes and RAM for data storage

The ability to dynamically add new attributes to data records

Rick Cattel 2010

ACID V.S. BASE

ACID = Atomicity, Consistency, Isolation, and Durability

BASE = Basically Available, Soft state, Eventually consistent

Don't use "BASE" – it didn't stick.

Aside:

Consistency: "Any data written to the database must be valid according to all defined rules"

Rick Cattel 2010

MAJOR IMPACT SYSTEMS (RICK CATTEL)

- "Memcached demonstrated that in-memory indexes can be highly scalable, distributing and replicating objects over multiple nodes."
- "Dynamo pioneered the idea of [using] eventual consistency as a way to achieve higher availability and scalability: data fetched are not guaranteed to be up-to-date, but updates are guaranteed to be propagated to all nodes eventually."
- "BigTable demonstrated that persistent record storage could be scaled to thousands of nodes, a feat that most of the other systems aspire to."

Rick Cattel 2010

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language / Algebra	Data model	my label
1971	RDBMS	O	✔	✔	✔	✔	✔	✔	✔	tables	sql-like
2003	memcached	✔	✔	O	O	O	O	O	O	key-val	nosql
2004	MapReduce	✔	O	O	O	✔	O	O	O	key-val	batch
2005	CouchDB	✔	✔	✔	record	MR	O	✔	O	document	nosql
2006	BigTable (Hbase)	✔	✔	✔	record	compat. w/ MR	/	O	O	ext. record	nosql
2007	MongoDB	✔	✔	✔	EC, record	O	O	O	O	document	nosql
2007	Dynamo	✔	✔	O	O	O	O	O	O	key-val	nosql
2008	Pig	✔	O	O	O	✔	/	O	✔	tables	sql-like
2008	HIVE	✔	O	O	O	✔	✔	O	✔	tables	sql-like
2008	Cassandra	✔	✔	✔	EC, record	O	✔	✔	O	key-val	nosql
2009	Voldemort	✔	✔	O	EC, record	O	O	O	O	key-val	nosql
2009	Riak	✔	✔	✔	EC, record	MR	O			key-val	nosql
2010	Dremel	✔	O	O	O	/	✔	O	✔	tables	sql-like
2011	Megastore	✔	✔	✔	entity groups	O	/	O	/	tables	nosql
2011	Tenzing	✔	O	O	O	O	✔	✔	✔	tables	sql-like
2011	Spark/Shark	✔	O	O	O	✔	✔	O	✔	tables	sql-like
2012	Spanner	✔	✔	✔	✔	?	✔	✔	✔	tables	sql-like
2012	Accumulo	✔	✔	✔	record	compat. w/ MR	/	O	O	ext. record	nosql
2013	Impala	✔	O	O	O	✔	✔	O	✔	tables	sql-like

MEMCACHED

Main-memory caching service

- basic system: no persistence, replication, fault-tolerance
- Many extensions provide these features
- Ex: membrain, membase

Mature system, still in wide use

Important concept: *consistent hashing*

"REGULAR" HASHING

Assign M data keys to N servers

assign each key to server = k mod N

Example: N=3

key 0 -> server 0
key 1 -> server 1
key 2 -> server 2
key 3 -> server 0
key 4 -> server 1
…

data keys
k0 = 367
k1 = 452
k2 = 776
…

server 1 2 3 …. 64

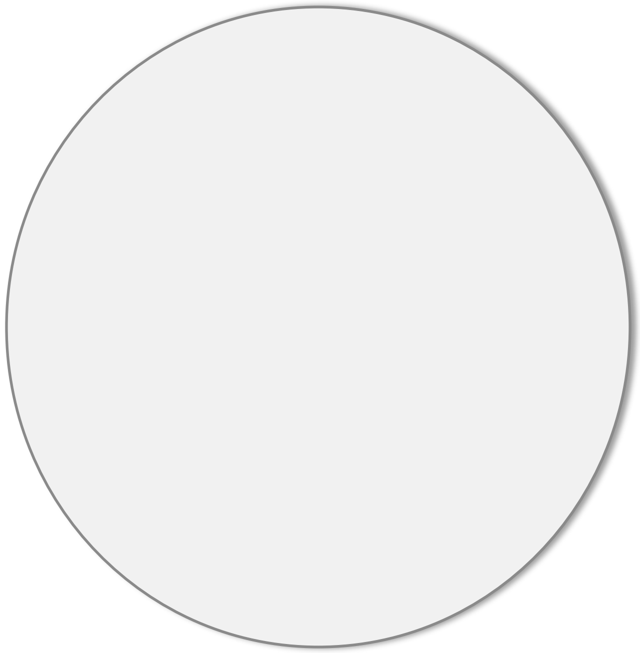
What happens if I increase the number of servers from N to 2N?

Every *existing key needs to be remapped, and we're screwed.*

CONSISTENT HASHING

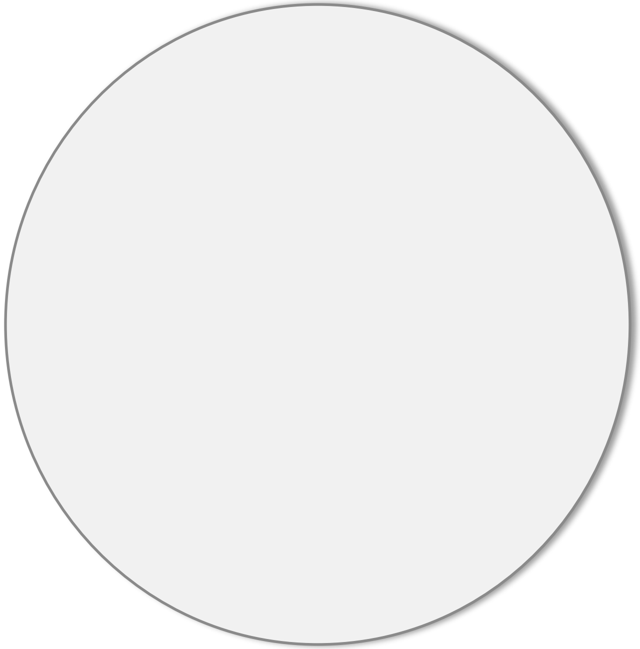
server id = 1
server id = 2
server id = 3
…

data key = 367
data key = 452
data key = 776
…



CONSISTENT HASHING: ROUTING

server id = 1
server id = 2
server id = 3
…



ROUNDUP

Hadoop**

Greenplum

Pig**

Dryad

DryadLINQ

HIVE**

Megastore

Dremel / BigQuery

Sawzall

Pregel

SciDB

MapReduce++

SCOPE

SimpleDB

Cassandra**

CouchDB**

MongoDB

SPARK

Twister

HaLoop

BigTable

Elastic MapReduce

S3

Google, Microsoft, Yahoo, Facebook, Amazon, Startup, ***University Research,*** **UW**

**Apache open source project

RICK CATTELL'S CLUSTERING

Key-Value Stores

- e.g., Memcached, Dynamo, Voldemort, Riak

Document Stores

- e.g., MongoDB, CouchDB

Extensible Record Store

- e.g., BigTable

MAJOR IMPACT SYSTEMS (RICK CATTEL)

- **Memcached** demonstrated that **in-memory indexes can be highly scalable**, distributing and replicating objects over multiple nodes.
- **Dynamo** pioneered the idea of [using] **eventual consistency as a way to achieve higher availability and scalability**: data fetched are not guaranteed to be up-to-date, but updates are guaranteed to be propagated to all nodes eventually.
- **BigTable** demonstrated that **persistent record storage could be scaled to thousands of nodes**, a feat that most of the other systems aspire to.

NOSQL FEATURES (RICK CATTEL)

Ability to horizontally scale "simple operation" throughput over many servers

- Simple = key lookups, read/write of 1 or few records

The ability to replicate and partition data over many servers

- Consider "sharding" and "horizontal partitioning" to be synonyms

A simple API

A weaker concurrency model than ACID transactions

Efficient use of distributed indexes and RAM for data storage

The ability to dynamically add new attributes to data records

A DIFFERENT (NON-DISJOINT) CLUSTERING

Parallel Relational Databases

- Teradata, Greenplum, Netezza, Aster Data Systems, Vertica, …
- (Not MySQL or PostgreSQL)

NoSQL systems

- "Key-value stores", "Document Stores" and "Extensible Record Stores"
- Cassandra, CouchDB, MongoDB, BigTable/Hbase/Accumulo

MapReduce-based systems

- Hadoop, Pig, HIVE

Cloud services

- DynamoDB, SimpleDB, S3, Megastore, BigQuery
- CouchBase,

ROUGH DECISION PROCEDURE

Parallel Relational Databases

- Schema?
- Complex queries?
- Transactions?
- High-performance?
    - indexing, views, cost-based optimization, ….
- Are you rich?
    - ~$20k-$125k / TB

ROUGH DECISION PROCEDURE

Hadoop (locally managed)

- Java?
- Large unstructured files?
- Relatively small number of tasks?
- System administration support?
- Need to / want to roll your own algorithms?

ROUGH DECISION PROCEDURE

Hadoop derivatives (Pig, HIVE)

- Same as Hadoop, with
- A high-level language
- Support for multi-step workflows

ROUGH DECISION PROCEDURE

NoSQL

- One object at a time manipulation?
- Lots of concurrent users/apps?
- Can you live without joins?
- System administration?
- Need interactive response times?
  - e.g., you're powering a web application

ROUGH DECISION PROCEDURE

Cloud Services

- No local hardware
- No local sys admin support
- Inconsistent workloads

ROADMAP

Intro

Decision procedures

Key-Value Store Example:

- memcached

Document Store Example:

- CouchDB

Extensible Record Store Example:

- Apache Hbase / Google BigTable

Discriminators

- Programming Model
- Consistency, Latency
- Cloud vs. Local

MEMCACHED

Main-memory caching service

- basic system: no persistence, replication, fault-tolerance
- Many extensions provide these features
- Ex: membrain, membase

Mature system, still in wide use

Important concept: *consistent hashing*

“REGULAR” HASHING

Example: N=3

Assign M data keys to N servers

key 0 -> server 0
key 1 -> server 1
key 2 -> server 2
key 3 -> server 0
key 4 -> server 1
…

assign each key to server = k mod N

data keys
k0 = 367
k1 = 452
k2 = 776
…

server 1 2 3 …. 64

What happens if I increase the number of servers from N to 2N?

Every existing key needs to be remapped, and we’re screwed.

ROADMAP

Intro

Decision procedures

Key-Value Store Example:

- memcached

Document Store Example:

- CouchDB

Extensible Record Store Example:

- Apache Hbase / Google BigTable

Discriminators

- Programming Model
- Consistency, Latency
- Cloud vs. Local

COUCHDB: DATA MODEL

Document-oriented

- Document = set of key/value pairs
- Ex:

```
{
  "Subject": "I like Plankton"
  "Author": "Rusty"
  "PostedDate": "5/23/2006"
  "Tags": ["plankton", "baseball", "decisions"]
  "Body": "I decided today that I don't like baseball. I like plankton."
}
```

COUCHDB: UPDATES

ACID

- Atomic, Consistent, Isolated, Durable

Lock-free concurrency

- Optimistic – attempts to edit dirty data fail

No transactions

- Transaction: Sequence of related updates
- Entire sequence considered ACID

COUCHDB: VIEWS

```
"_id":"_design/company",
"_rev":"12345",
"language": "javascript",
"views":
{
  "all": {
  "map": "function(doc) {if (doc.Type=='customer') emit(null,
doc) }"
  },
    "by_lastname": {
  "map": "function(doc) {if (doc.Type=='customer')
emit(doc.LastName, doc) }"
    },
  "total_purchases": {
  "map": "function(doc) {if (doc.Type=='purchase')
emit(doc.Customer, doc.Amount)}",
      "reduce": "function(keys, values) { return sum(values) }"
    }
  }
}
```

ROADMAP

Intro

Decision procedures

Key-Value Store Example:

- memcached

Document Store Example:

- CouchDB

Extensible Record Store Example:

- Apache Hbase / Google BigTable

Discriminators

- Programming Model
- Consistency, Latency
- Cloud vs. Local

GOOGLE BIGTABLE

OSDI paper in 2006

- Some overlap with the authors of the MapReduce paper

Complementary to MapReduce

- Recall: What is MapReduce *not* designed for?

DATA MODEL

“a sparse, distributed, persistent multi-dimensional sorted map”

$$(\text{row:string}, \text{column:string}, \text{time:int64}) \rightarrow \text{string}$$

ROWS

Data is sorted lexicographically by row key

Row key range broken into *tablets*

- Recall: What was Teradata's model of distribution?

A tablet is the unit of distribution and load balancing

COLUMN FAMILIES

Column names of the form *family:qualifier*

"family" is the basic unit of

- access control
- memory accounting
- disk accounting

Typically all columns in a family the same type

TIMESTAMPS

Each cell can be *versioned*

Each new version increments the timestamp

Policies:

- "keep only latest *n* versions"
- "keep only versions since time *t*"

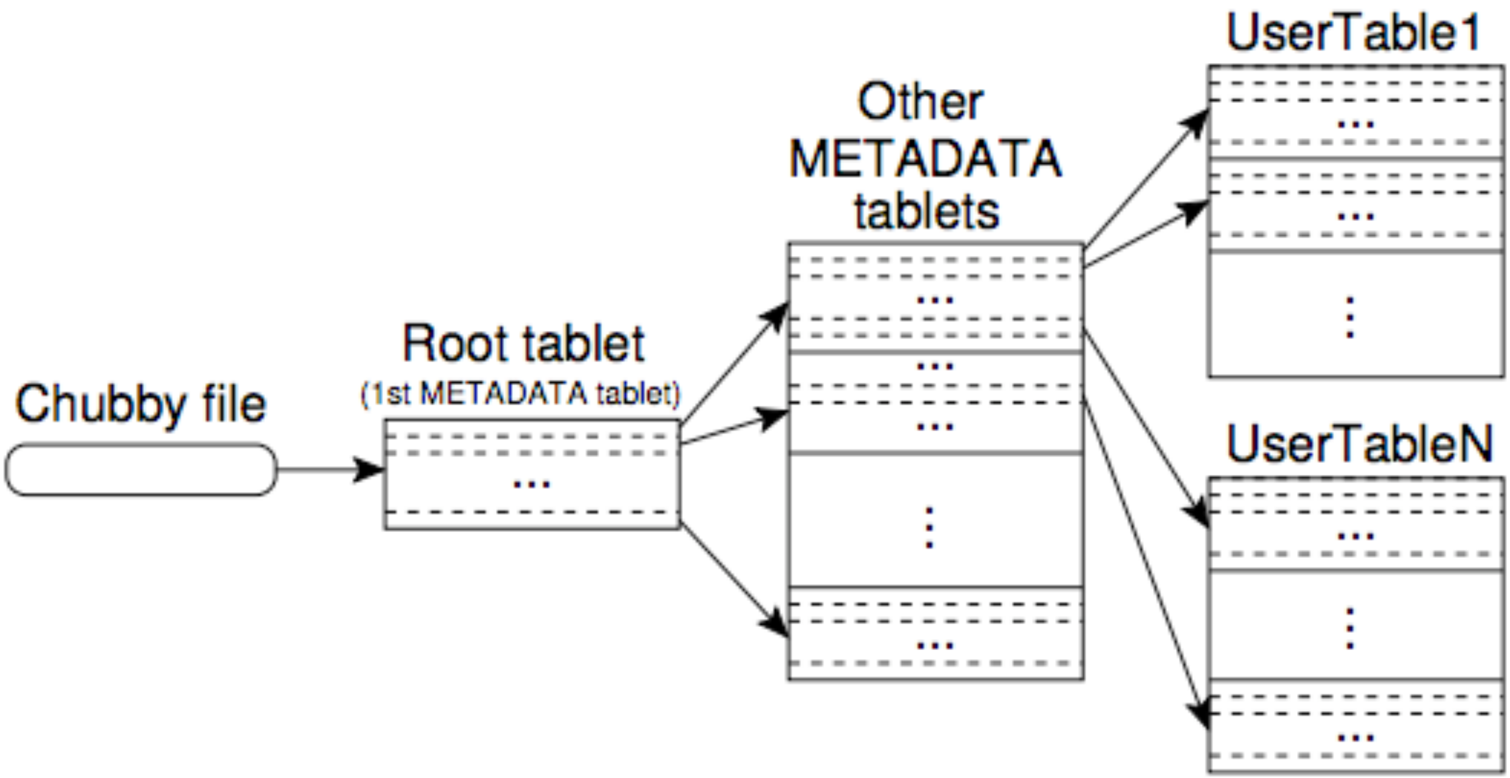
Master assigns tablets to tablet servers

Tablet server manages reads and writes from its tablets

Clients communicate directly with tablet server

Tablet server splits tablets that have grown too large.

TABLET LOCATION METADATA

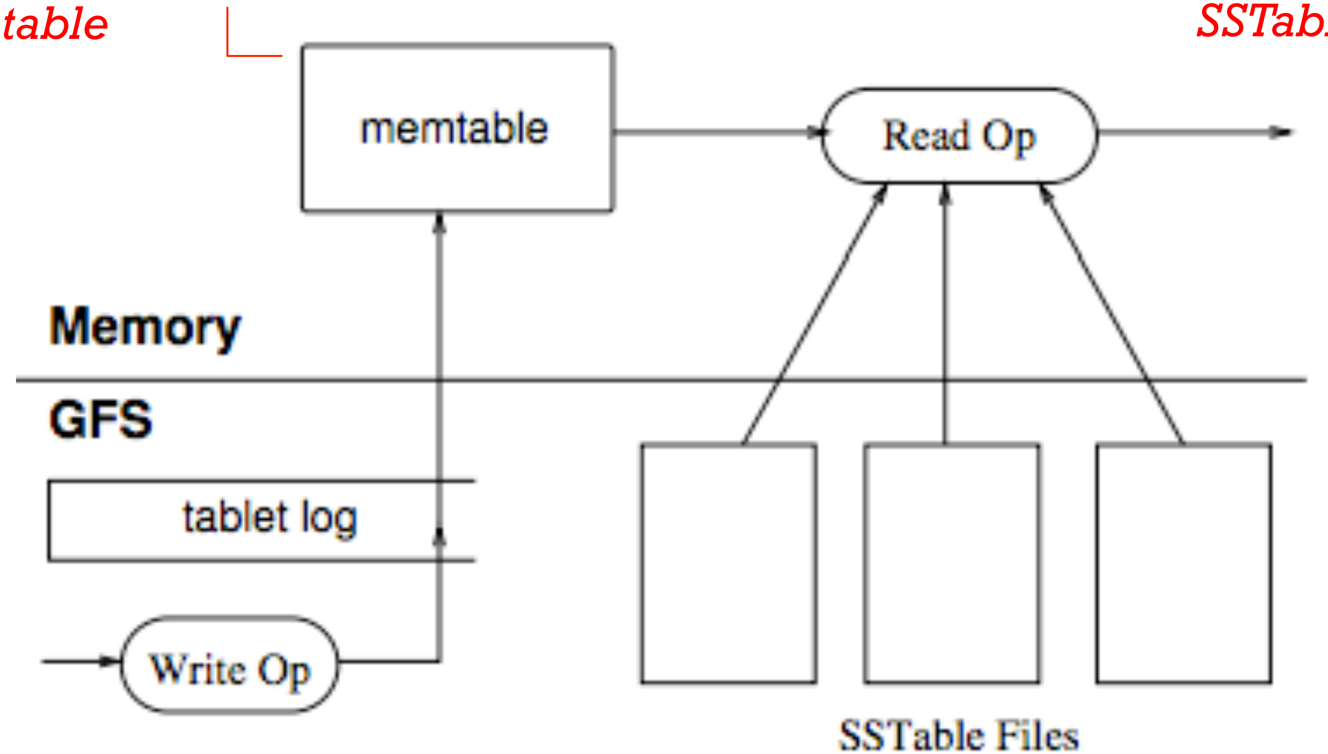


Chubby: distributed lock service

WRITE PROCESSING

minor compaction: write memtable buffer to a new SSTable

recent sequence of updates in memtable



major compaction: rewrite all SSTables into one SSTable; clean deletes

OTHER TRICKS

Compression

- specified by clients

Bloom filters

- fast set membership test for (row, column) pair
- reduces disk accesses during reads

Locality Groups

- Groups of column families frequently accessed together

Immutability

- SSTables (disk chunks) are immutable
- Only the memtable needs to support concurrent updates (via copy-on-write)

HBASE

Implementation of Google BigTable

Compatible with Hadoop

- TableInputFormat allows reading of BigTable data in the map phase
- One mapper per tablet
- Aside: Speculative Execution?

```
Table       (HBase table)
    Region       (Regions for the table)
         Store        (Store per ColumnFamily for each Region for the table)
              MemStore        (MemStore for each Store for each Region for the table)
              StoreFile       (StoreFiles for each Store for each Region for the table)
                    Block          (Blocks within a StoreFile within a Store for each Region for the table)
```

ROADMAP

Intro

Decision procedures

NoSQL example: CouchDB

Discriminators

- Programming Model
- Consistency, Latency
- Cloud vs. Local

JOINS

Ex: Show all comments by “Sue” on any blog post by “Jim”

Method 1:

- Lookup all blog posts by Jim
- For each post, lookup all comments and filter for “Sue”

Method 2:

- Lookup all comments by Sue
- For each comment, lookup all posts and filter for “Jim”

Method 3:

- Filter comments by Sue, filter posts by Jim,
- Sort all comments by blog id, sort all blogs by blog id
- Pull one from each list to find matches

PROGRAMMING MODELS: MY TERMINOLOGY

Lookup

- put/get objects by key only
- Ex: Cassandra

Filter

- Non-key access.
- No joins. Single-relation access only. May look like SQL.
- Ex: SimpleDB, Google Megastore

MapReduce

- Two functions define a distributed program: Map and Reduce

RA-like

- A set of operators akin to the Relational Algebra
- Ex: Pig, MS DryadLINQ

SQL-like

- Declarative language
- Includes joins
- Ex: HIVE, Google BigQuery

CLOUD SERVICES

Product	**Provider**	**Prog. Model**	**Storage Cost**	**Compute Cost**	**IO Cost**
Megastore	Google	Filter	$0.15 / GB / mo.	$0.10 / corehour	$.10 / GB in, $.12 / GB out
BigQuery	Google	SQL-like	Closed beta	Closed beta	Closed beta
Microsoft Table	Microsoft	Lookup	$0.15 / GB / mo.	$0.12 / hour and up	$.10 / GB in, $.15 / GB out
Elastic MapReduce	Amazon	MR, RA-like, SQL	$0.093 / GB / mo.	$0.10 / hour and up	$0.10 / GB in, $0.15 / GB out (1st GB free)
SimpleDB	Amazon	Filter	$0.093 / GB / mo.	1st 25 hours free, $0.14 after that	$0.10 / GB in, $0.15 / GB out (1st GB free)

http://escience.washington.edu/blog

CAP THEOREM [BREWER 2000, LYNCH 2002]

Consistency

- Do all applications see all the same data?

Availability

- If some nodes fail, does everything still work?

Partitioning

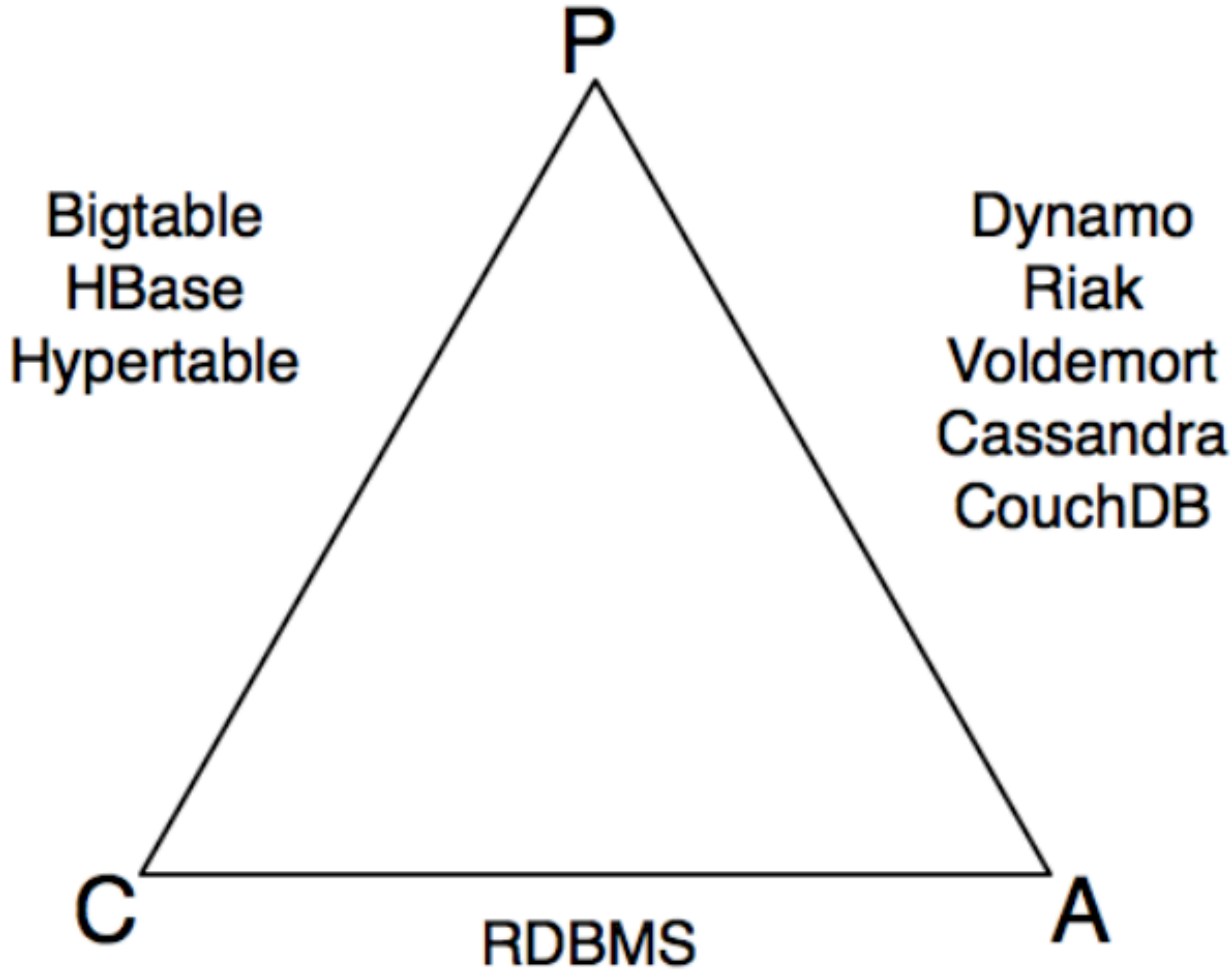
- If your nodes can't talk to each other, does everything still work?

CAP Theorem: Choose two, or sacrifice latency

Databases: Consistency, Availability

NoSQL: Availability, Partitioning

But: Some counterevidence – "NewSQL"



src: Shashank Tiwari

"EVENTUAL CONSISTENCY"

Write conflicts will eventually propagate throughout the system

- D. Terry et al., "Managing Update Conflicts in Bayou,a Weakly Connected Replicated Storage System", SOSP 1995

"We believe that applications must be aware that they may read weakly consistent data and also that their write operations may conflict with those of other users and applications."

"Moreover, applications must be revolved m the detection and resolution of conflicts since these naturally depend on the semantics of the application."

EVENTUAL CONSISTENCY

What the application sees in the meantime is sensitive to replication mechanics and difficult to predict

Contrast with RDBMS, Paxos: Immediate (or "strong") consistency, but there may be deadlocks

"What information consumes is rather obvious: it consumes the attention of its recipients. **Hence a wealth of information creates a poverty of attention**, and a need to allocate that attention efficiently among the overabundance of information sources that might consume it."

-- Herbert Simon

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	**O**	✔	✔	✔	✔	✔	✔	✔	**tables**	**sql-like**
2003	memcached	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2004	MapReduce	✔	**O**	**O**	**O**	✔	**O**	**O**	**O**	**key-val**	**batch**
2005	CouchDB	✔	✔	✔	**record**	**MR**	**O**	✔	**O**	**document**	**nosql**
2006	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**nosql**
2007	MongoDB	✔	✔	✔	**EC, record**	**O**	**O**	**O**	**O**	**document**	**nosql**
2007	Dynamo	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2008	Pig	✔	**O**	**O**	**O**	✔	**/**	**O**	✔	**tables**	**sql-like**
2008	HIVE	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**
2008	Cassandra	✔	✔	✔	**EC, record**	**O**	✔	✔	**O**	**key-val**	**nosql**
2009	Voldemort	✔	✔	**O**	**EC, record**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2009	Riak	✔	✔	✔	**EC, record**	**MR**	**O**			**key-val**	**nosql**
2010	Dremel	✔	**O**	**O**	**O**	**/**		**O**	✔	**tables**	**sql-like**
2011	Megastore	✔	✔	✔	**entity groups**	**O**	**/**	**O**	**/**	**tables**	**nosql**
2011	Tenzing	✔	**O**	**O**	**O**	**O**	✔	✔	✔	**tables**	**sql-like**
2011	Spark/Shark	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**
2012	Spanner	✔	✔	✔	✔	**?**	✔	✔	✔	**tables**	**sql-like**
2012	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**nosql**
2013	Impala	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**

DYNAMODB

Key features:

Service Level Agreement (SLN): at the

99th percentile, and not on mean/median/variance (otherwise, one penalizes the heavy users)

“Respond within 300ms for 99.9% of its requests”

DYNAMO (2)

Key features:

DHT with replication:

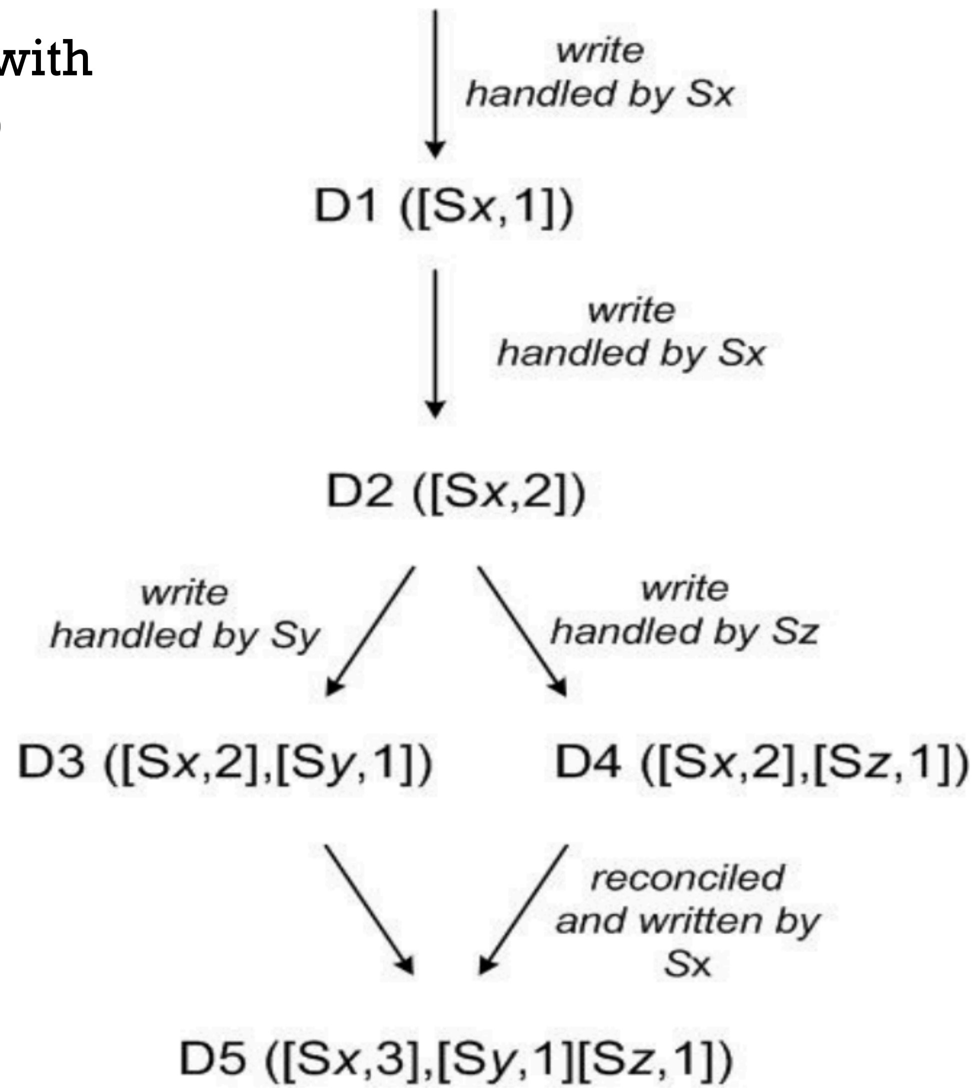
- Store value at k, k+1, …, k+N-1
- Eventual consistency through vector clocks

Reconciliation at read time:

- Writes never fail ("poor customer experience")
- Conflict resolution: "last write wins" or application specific

VECTOR CLOCKS

Each data item associated with a list of (server, timestamp) pairs indicating its version history.



VECTOR CLOCKS EXAMPLE

A client writes D1 at server SX:

D1 ([SX,1])

Another client reads D1, writes back D2; also handled by SX:

D2 ([SX,2]) (D1 garbage collected)

Another client reads D2, writes back D3; handled by server SY:

D3 ([SX,2], [SY,1])

Another client reads D2, writes back D4; handled by server SZ:

D4 ([SX,2], [SZ,1])

Another client reads D3, D4: CONFLICT !

Data 1	Data 2	Conflict?
$([S_x,3],[S_y,6])$	$([S_x,3],[S_z,2])$	
$([S_x,3])$	$([S_y,5])$	
$([S_x,3],[S_y,6])$	$([S_x,3],[S_y,6]),[S_z,2])$	
$([S_x,3],[S_y,10])$	$([S_x,3],[S_y,6]),[S_z,2])$	
$([S_x,3],[S_y,10])$	$([S_x,3],[S_y,20]),[S_z,2])$	

CONFIGURABLE CONSISTENCY

R = Minumum number of nodes that participate in a successful read

W = Minumum number of nodes that participate in a successful write

N = Replication factor

If R + W > N, you can claim consistency

But R + W < N means lower latency.

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	**O**	✔	✔	✔	✔	✔	✔	✔	**tables**	**sql-like**
2003	memcached	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2004	MapReduce	✔	**O**	**O**	**O**	✔	**O**	**O**	**O**	**key-val**	**batch**
2005	CouchDB	✔	✔	✔	**record**	**MR**	**O**	✔	**O**	**document**	**nosql**
2006	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**nosql**
2007	MongoDB	✔	✔	✔	**EC, record**	**O**	**O**	**O**	**O**	**document**	**nosql**
2007	Dynamo	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2008	Pig	✔	**O**	**O**	**O**	✔	**/**	**O**	✔	**tables**	**sql-like**
2008	HIVE	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**
2008	Cassandra	✔	✔	✔	**EC, record**	**O**	✔	✔	**O**	**key-val**	**nosql**
2009	Voldemort	✔	✔	**O**	**EC, record**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2009	Riak	✔	✔	✔	**EC, record**	**MR**	**O**			**key-val**	**nosql**
2010	Dremel	✔	**O**	**O**	**O**	**/**	✔	**O**	✔	**tables**	**sql-like**
2011	Megastore	✔	✔	✔	**entity groups**	**O**	**/**	**O**	**/**	**tables**	**nosql**
2011	Tenzing	✔	**O**	**O**	**O**	**O**	✔	✔	✔	**tables**	**sql-like**
2011	Spark/Shark	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**
2012	Spanner	✔	✔	✔	✔	**?**	✔	✔	✔	**tables**	**sql-like**
2012	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**nosql**
2013	Impala	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**

COUCHDB: DATA MODEL

Document-oriented

- Document = set of key/value pairs
- Ex:

```
{
  "Subject": "I like Plankton"
  "Author": "Rusty"
  "PostedDate": "5/23/2006"
  "Tags": ["plankton", "baseball", "decisions"]
  "Body": "I decided today that I don't like baseball. I
like plankton."
}
```

COUCHDB: UPDATES

Full Consistency within a document

Lock-free concurrency

- Optimistic – attempts to edit dirty data fail

No multi-row transactions

- Transaction: Sequence of related updates
- Entire sequence considered ACID

COUCHDB: VIEWS

```
"_id":"_design/company",
"_rev":"12345",
"language": "javascript",
"views":
{
 "all": {
  "map": "function(doc) {if (doc.Type=='customer’) emit(null, doc) }"
 },

 "by_lastname": {
  "map": "function(doc) {if (doc.Type=='customer’) emit(doc.LastName, doc) }"
  },

 "total_purchases": {
  "map": "function(doc) {
         if (doc.Type=='purchase’)
          emit(doc.Customer, doc.Amount)
        }",
  "reduce": "function(keys, values) { return sum(values) }"
  }
 }
}
```

COUCHDB "JOINS"

View Collation

```
function(doc) {
  if (doc.type == "post") {
    emit([doc._id, 0], doc);
  } else if (doc.type == "comment") {
    emit([doc.post, 1], doc);
  }
}
```

my_view?startkey=[“some_post_id"]&endkey;=[“some_post_id", 2]

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	O	✔	✔	✔	✔	✔	✔	✔	**tables**	**SQL-like**
2003	memcached	✔	✔	O	O	O	O	O	O	**key-val**	**lookup**
2004	MapReduce	✔	O	O	O	✔	O	O	O	**key-val**	**MR**
2005	CouchDB	✔	✔	✔	**record**	**MR**	O	✔	O	**document**	**filter/MR**
2006	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	/	O	O	**ext. record**	**filter/MR**
2007	MongoDB	✔	✔	✔	**EC, record**	O	O	O	O	**document**	**filter**
2007	Dynamo	✔	✔	O	O	O	O	O	O	**key-val**	**lookup**
2008	Pig	✔	O	O	O	✔	/	O	✔	**tables**	**RA-like**
2008	HIVE	✔	O	O	O	✔	✔	O	✔	**tables**	**SQL-like**
2008	Cassandra	✔	✔	✔	**EC, record**	O	✔	✔	O	**key-val**	**filter**
2009	Voldemort	✔	✔	O	**EC, record**	O	O	O	O	**key-val**	**lookup**
2009	Riak	✔	✔	✔	**EC, record**	**MR**	O			**key-val**	**filter**
2010	Dremel	✔	O	O	O	/	✔	O	✔	**tables**	**SQL-like**
2011	Megastore	✔	✔	✔	**entity groups**	O	/	O	/	**tables**	**filter**
2011	Tenzing	✔	O	O	O	O	✔	✔	✔	**tables**	**SQL-like**
2011	Spark/Shark	✔	O	O	O	✔	✔	O	✔	**tables**	**SQL-like**
2012	Spanner	✔	✔	✔	✔	?	✔	✔	✔	**tables**	**SQL-like**
2012	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	/	O	O	**ext. record**	**filter**
2013	Impala	✔	O	O	O	✔	✔	O	✔	**tables**	**SQL-like**

GOOGLE BIGTABLE

OSDI paper in 2006

- Some overlap with the authors of the MapReduce paper

Complementary to MapReduce

- Recall: What is MapReduce *not* designed for?

DATA MODEL

“a sparse, distributed, persistent multi-dimensional sorted map”

(row:string, column:string, time:int64) → string

ROWS

Data is sorted lexicographically by row key

Row key range broken into *tablets*

- Recall: What was Teradata's model of distribution?

A tablet is the unit of distribution and load balancing

COLUMN FAMILIES

Column names of the form *family:qualifier*

"family" is the basic unit of

- access control
- memory accounting
- disk accounting

Typically all columns in a family the same type

TIMESTAMPS

Each cell can be *versioned*

Each new version increments the timestamp

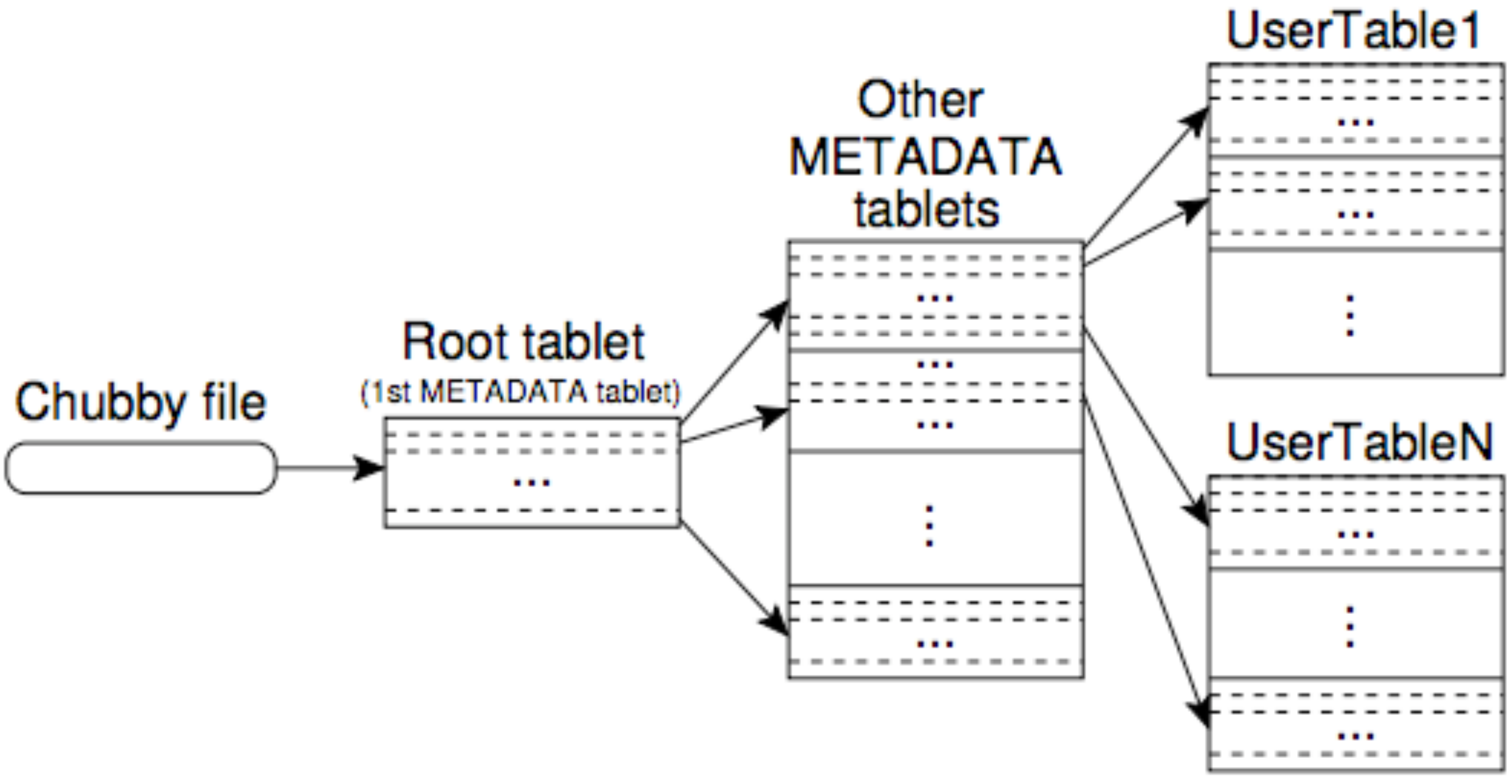
Policies:

- "keep only latest *n* versions"
- "keep only versions since time *t*"

TABLET MANAGEMENT

- **Master assigns tablets to tablet servers**
- **Tablet server manages reads and writes from its tablets**
- **Clients communicate directly with tablet server**
- **Tablet server splits tablets that have grown too large.**

TABLET LOCATION METADATA

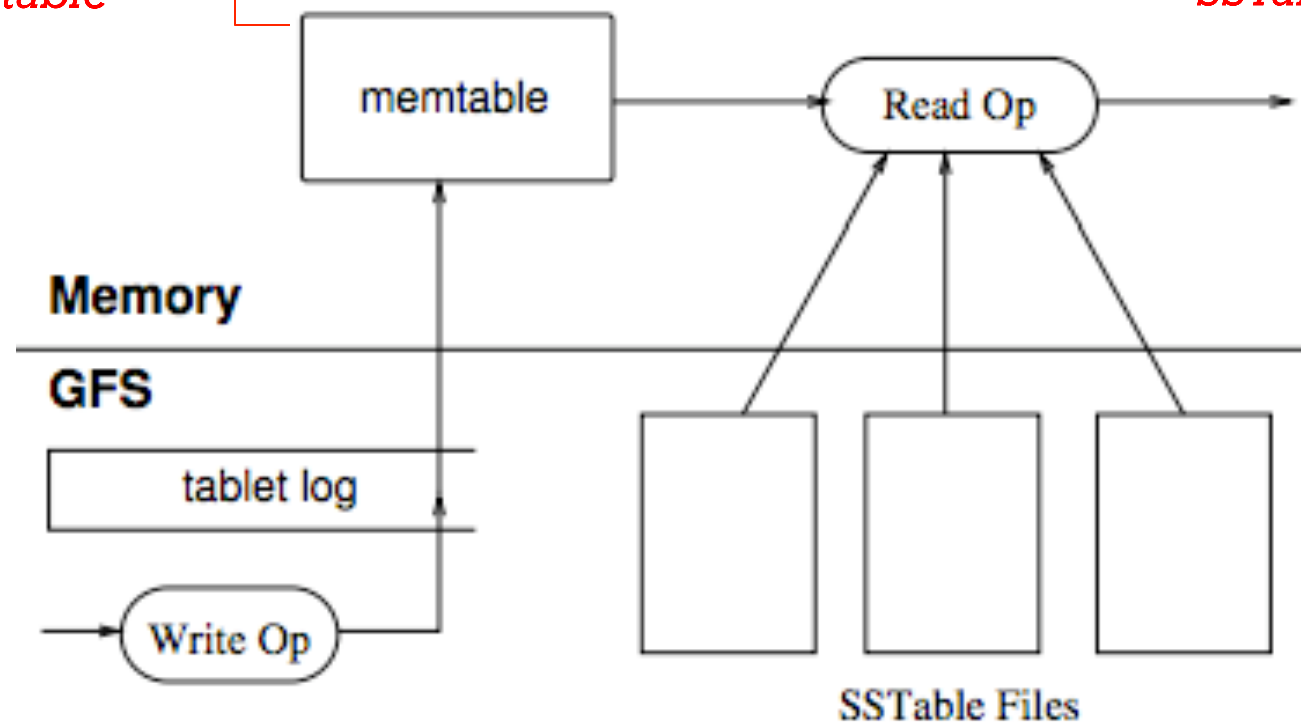


Chubby: distributed lock service

WRITE PROCESSING

minor compaction: write memtable buffer to a new SSTable

recent sequence of updates in memtable



major compaction: rewrite all SSTables into one SSTable; clean deletes

OTHER TRICKS

Compression

- specified by clients

Bloom filters

- fast set membership test for (row, column) pair
- reduces disk accesses during reads

Locality Groups

- Groups of column families frequently accessed together

Immutability

- SSTables (disk chunks) are immutable
- Only the memtable needs to support concurrent updates (via copy-on-write)

HBASE

Implementation of Google BigTable

Compatible with Hadoop

- TableInputFormat allows reading of BigTable data in the map phase
- One mapper per tablet

```
Table          (HBase table)
    Region         (Regions for the table)
        Store          (Store per ColumnFamily for each Region for the table)
            MemStore           (MemStore for each Store for each Region for the table)
            StoreFile          (StoreFiles for each Store for each Region for the table)
                Block              (Blocks within a StoreFile within a Store for each Region for the table)
```

MEGASTORE

Argues that loose consistency models complicate application programming

Synchronous replication

Full transactions within a partition

[Baker, Bond, Corbett, et al. 2011]

SPANNER

Even though many projects happily use Bigtable [9], we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: **those that have complex, evolving schemas**, **or** **those that want strong consistency in the presence of wide-area replication.**

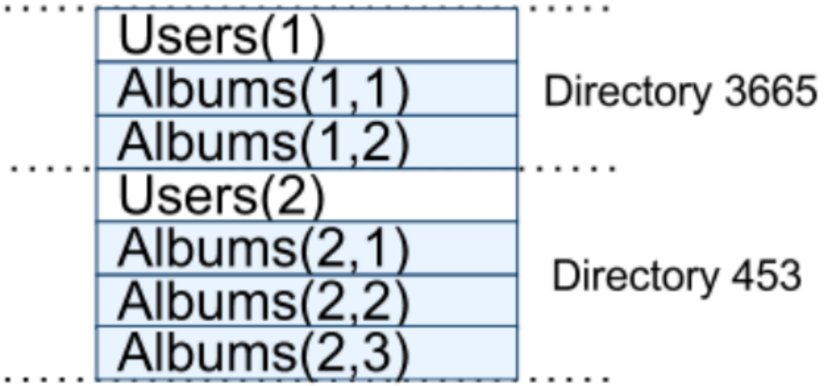
"We believe it is better to have application programmers deal with performance problems due to overuse of transactions as bottlenecks arise, rather than always coding around the lack of transactions."

"Although Spanner is scalable in the number of nodes, the node-local data structures have relatively poor performance on complex SQL queries, because they were designed for simple key-value accesses. Algorithms and data structures from DB literature could improve singlenode performance a great deal."

SPANNER DATA MODEL

Directory: set of contiguous keys with a shared prefix

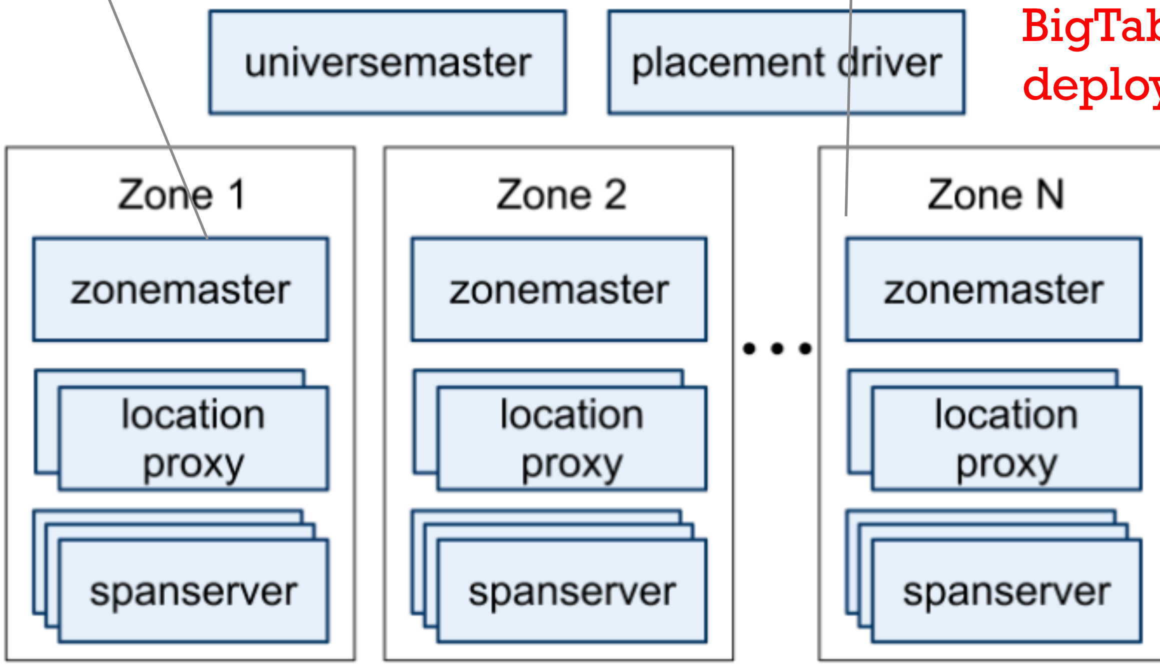
```
CREATE TABLE Users {
uid INT64 NOT NULL,
email STRING
} PRIMARY KEY (uid), DIRECTORY;

CREATE TABLE Albums {
uid INT64 NOT NULL,
aid INT64 NOT NULL,
name STRING
} PRIMARY KEY (uid, aid),
INTERLEAVE IN PARENT Users
```



Tables are interleaved to create a hierarchy mostly status info about zones

Moves directories for load balancing every few minutes

Zone ~ BigTable deployment

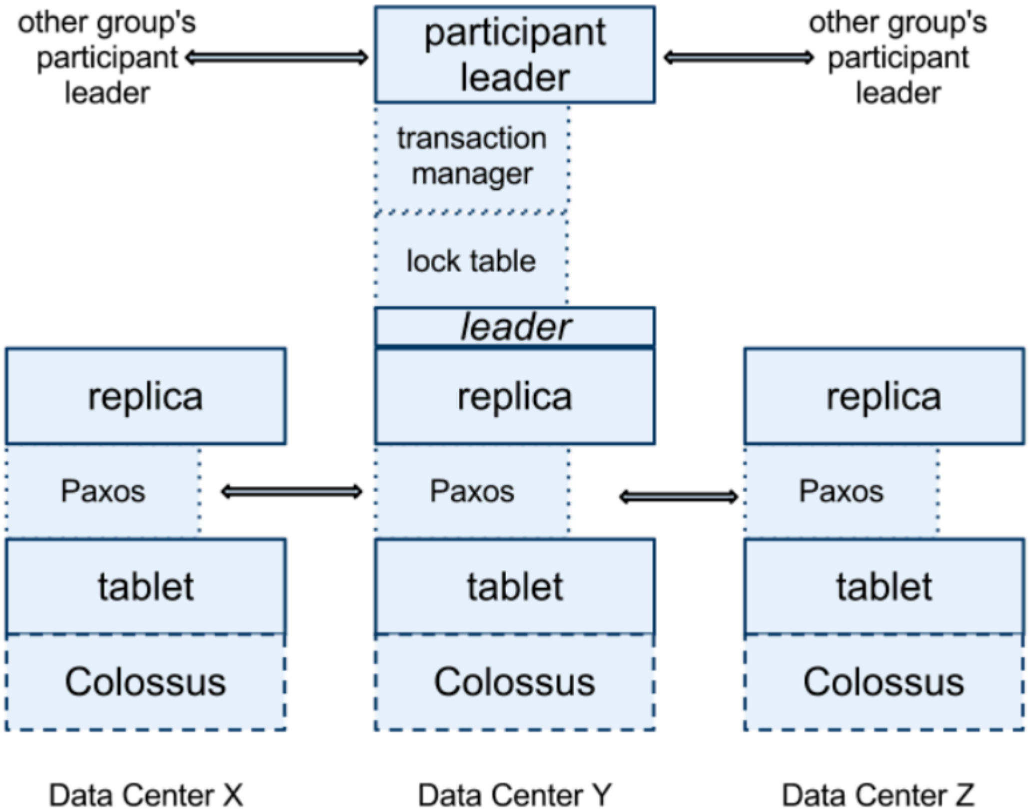


Assigns data to spanservers

Routes requests from clients

Serves data

SPANSERVER



2-phase commit across groups (when needed)

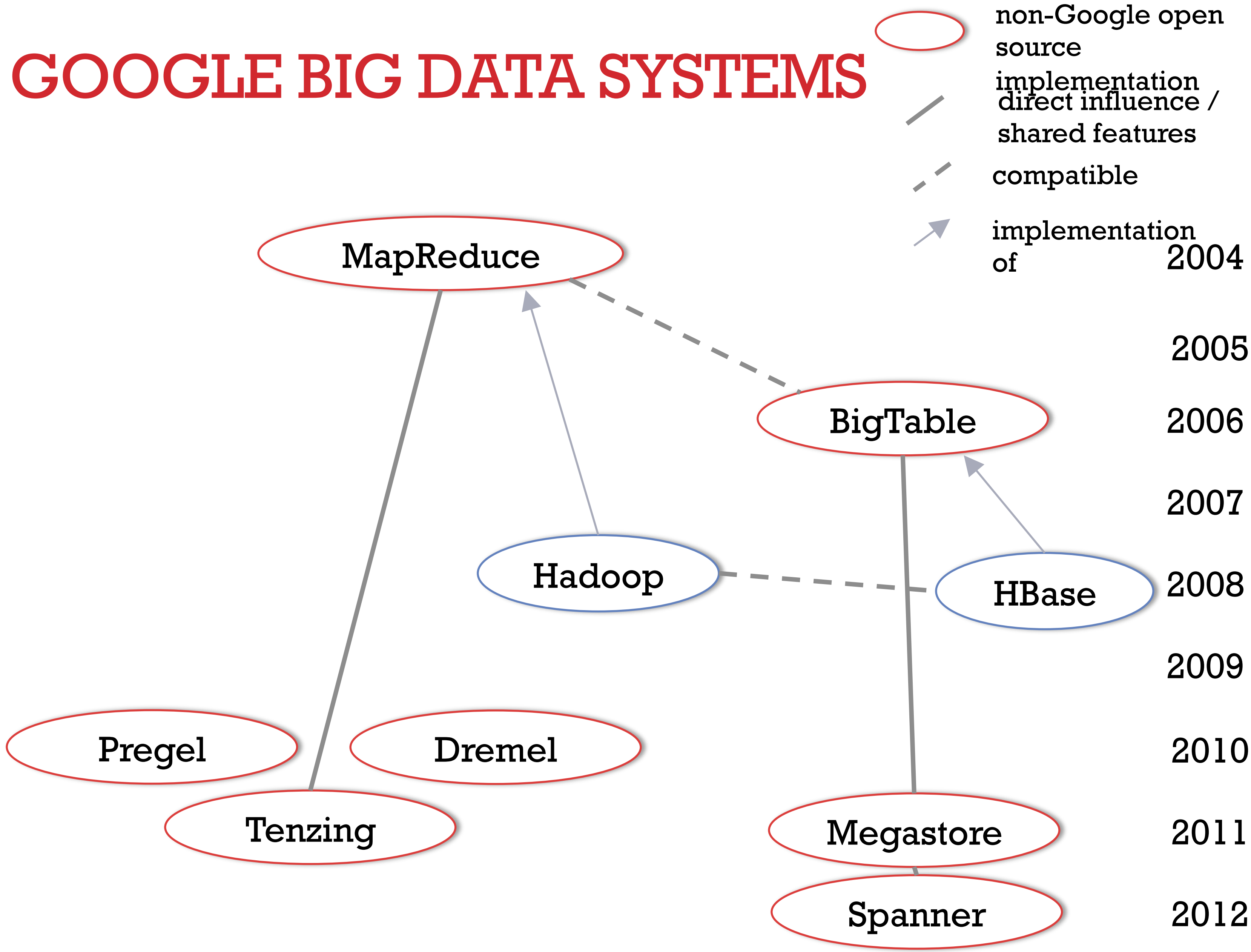
Paxos across replicas

Colossus successor to GFS

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	**O**	✔	✔	✔	✔	✔	✔	✔	**tables**	**sql-like**
2003	memcached	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2004	MapReduce	✔	**O**	**O**	**O**	✔	**O**	**O**	**O**	**key-val**	**batch**
2005	CouchDB	✔	✔	✔	**record**	**MR**	**O**	✔	**O**	**document**	**nosql**
2006	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**nosql**
2007	MongoDB	✔	✔	✔	**EC, record**	**O**	**O**	**O**	**O**	**document**	**nosql**
2007	Dynamo	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**ext. record**	**nosql**
2008	Pig	✔	**O**	**O**	**O**	✔	**/**	**O**	✔	**tables**	**sql-like**
2008	HIVE	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**
2008	Cassandra	✔	✔	✔	**EC, record**	**O**	✔	✔	**O**	**key-val**	**nosql**
2009	Voldemort	✔	✔	**O**	**EC, record**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2009	Riak	✔	✔	✔	**EC, record**	**MR**	**O**			**key-val**	**nosql**
2010	Dremel	✔	**O**	**O**	**O**	**/**	✔	**O**	✔	**tables**	**sql-like**
2011	Megastore	✔	✔	✔	**entity groups**	**O**	**/**	**O**	**/**	**tables**	**nosql**
2011	Tenzing	✔	**O**	**O**	**O**	✔	✔	✔	✔	**tables**	**sql-like**
2011	Spark/Shark	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**
2012	Spanner	✔	✔	✔	✔	**?**	✔	✔	✔	**tables**	**sql-like**
2012	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**nosql**
2013	Impala	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**

Google's Systems

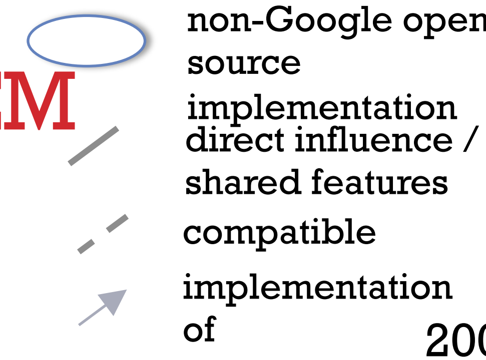
GOOGLE BIG DATA SYSTEMS

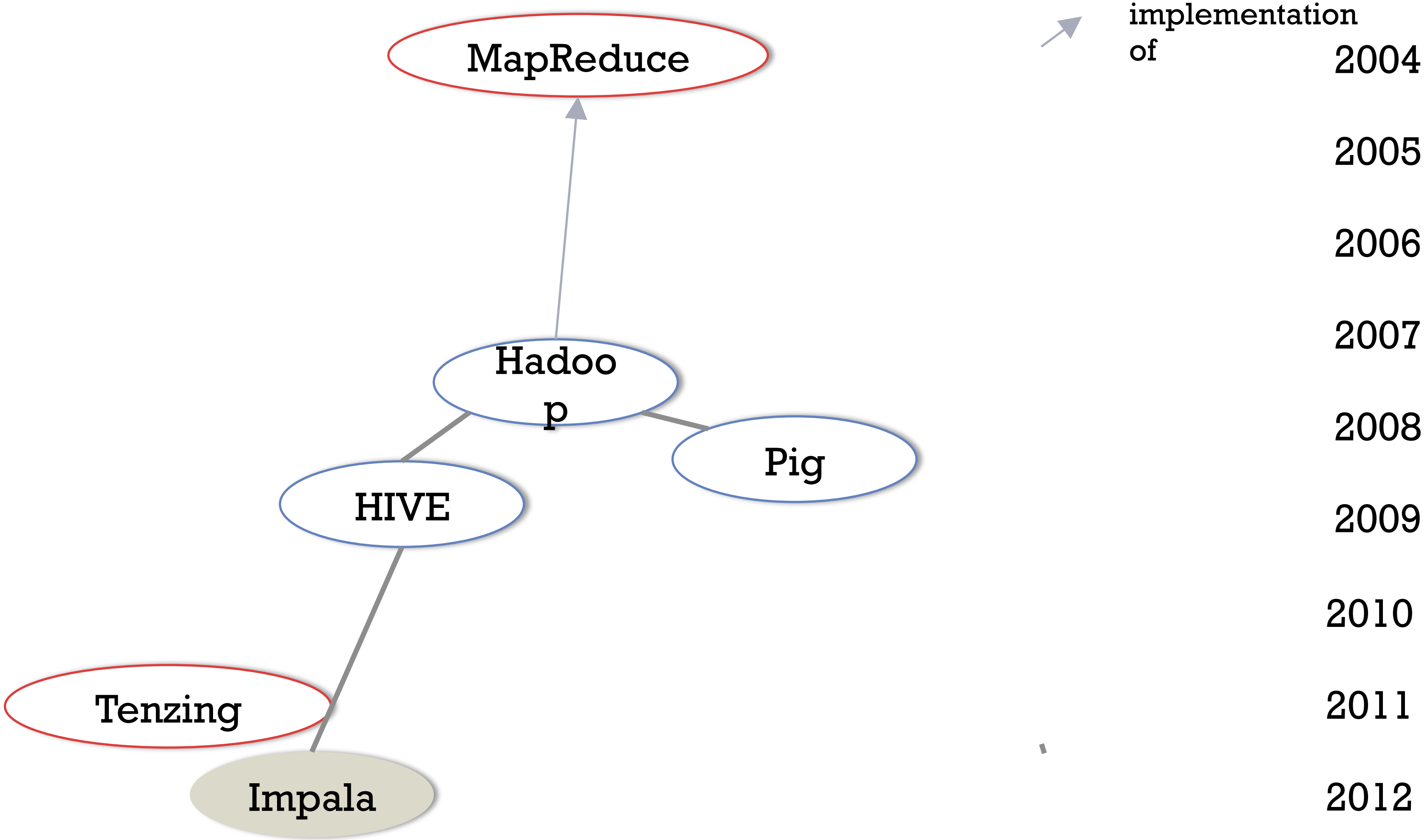


MAPREDUCED-BASED SYSTEMS

Year	*System/ Paper*	*Scale to 1000s*	*Primary Index*	*Secondary Indexes*	*Transactions*	*Joins/ Analytics*	*Integrity Constraints*	*Views*	*Language/ Algebra*	*Data model*	*my label*
1971	RDBMS	**O**	✔	✔	✔	✔	✔	✔	✔	**tables**	**sql-like**
2003	memcached	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2004	MapReduce	✔	**O**	**O**	**O**	✔	**O**	**O**	**O**	**key-val**	**batch**
2005	CouchDB	✔	✔	✔	**record**	**MR**	**O**	✔	**O**	**document**	**nosql**
2006	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**nosql**
2007	MongoDB	✔	✔	✔	**EC, record**	**O**	**O**	**O**	**O**	**document**	**nosql**
2007	Dynamo	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**ext. record**	**nosql**
2008	Pig	✔	**O**	**O**	**O**	✔	**/**	**O**	✔	**tables**	**sql-like**
2008	HIVE	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**
2008	Cassandra	✔	✔	✔	**EC, record**	**O**	✔	✔	**O**	**key-val**	**nosql**
2009	Voldemort	✔	✔	**O**	**EC, record**	**O**	**O**	**O**	**O**	**key-val**	**nosql**
2009	Riak	✔	✔	✔	**EC, record**	**MR**	**O**			**key-val**	**nosql**
2010	Dremel	✔	**O**	**O**	**O**	**/**	✔	**O**	✔	**tables**	**sql-like**
2011	Megastore	✔	✔	✔	**entity groups**	**O**	**/**	**O**	**/**	**tables**	**nosql**
2011	Tenzing	✔	**O**	**O**	**O**	✔	✔	✔	✔	**tables**	**sql-like**
2011	Spark/Shark	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**
2012	Spanner	✔	✔	✔	✔	**?**	✔	✔	✔	**tables**	**sql-like**
2012	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**nosql**
2013	Impala	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**sql-like**

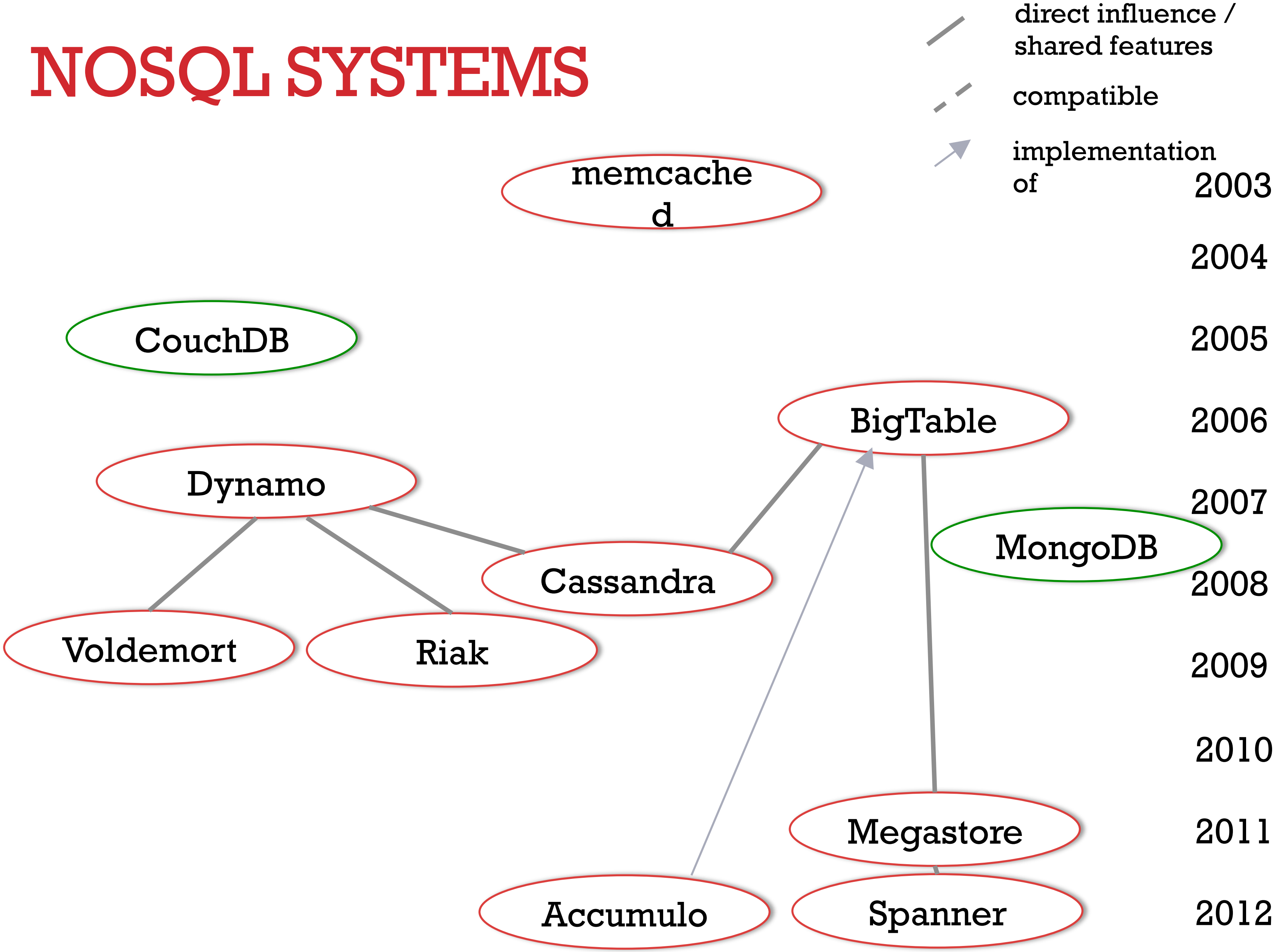
MAPREDUCED-BASED SYSTEM





Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	O	✔	✔	✔	✔	✔	✔	✔	**tables**	**sql-like**
2003	memcached	✔	✔	O	O	O	O	O	O	**key-val**	**nosql**
2004	MapReduce	✔	O	O	O	✔	O	O	O	**key-val**	**batch**
2005	CouchDB	✔	✔	✔	**record**	**MR**	O	✔	O	**document**	**nosql**
2006	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	/	O	O	**ext. record**	**nosql**
2007	MongoDB	✔	✔	✔	**EC, record**	O	O	O	O	**document**	**nosql**
2007	Dynamo	✔	✔	O	O	O	O	O	O	**ext. record**	**nosql**
2008	Pig	✔	O	O	O	✔	/	O	✔	**tables**	**sql-like**
2008	HIVE	✔	O	O	O	✔	✔	O	✔	**tables**	**sql-like**
2008	Cassandra	✔	✔	✔	**EC, record**	O	✔	✔	O	**key-val**	**nosql**
2009	Voldemort	✔	✔	O	**EC, record**	O	O	O	O	**key-val**	**nosql**
2009	Riak	✔	✔	✔	**EC, record**	**MR**	O			**key-val**	**nosql**
2010	Dremel	✔	O	O	O	/	✔	O	✔	**tables**	**sql-like**
2011	Megastore	✔	✔	✔	**entity groups**	O	/	O	/	**tables**	**nosql**
2011	Tenzing	✔	O	O	O	✔	✔	✔	✔	**tables**	**sql-like**
2011	Spark/Shark	✔	O	O	O	✔	✔	O	✔	**tables**	**sql-like**
2012	Spanner	✔	✔	✔	✔	?	✔	✔	✔	**tables**	**sql-like**
2012	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	/	O	O	**ext. record**	**nosql**
2013	Impala	✔	O	O	O	✔	✔	O	✔	**tables**	**sql-like**

NOSQL SYSTEMS



Year	source	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	many	RDBMS	O	✔	✔	✔	✔	✔	✔	✔	**tables**	**SQL-like**
2003	other	memcached	✔	✔	O	O	O	O	O	O	**key-val**	**lookup**
2004	Google	MapReduce	✔	O	O	O	✔	O	O	O	**key-val**	**MR**
2005	couchbase	CouchDB	✔	✔	✔	**record**	**MR**	O	✔	O	**document**	**filter/MR**
2006	Google	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	/	O	O	**ext. record**	**filter/MR**
2007	10gen	MongoDB	✔	✔	✔	**EC, record**	O	O	O	O	**document**	**filter**
2007	Amazon	Dynamo	✔	✔	O	O	O	O	O	O	**key-val**	**lookup**
2007	Amazon	SimpleDB	✔	✔	✔	O	O	O	O	O	**ext. record**	**filter**
2008	Yahoo	Pig	✔	O	O	O	✔	/	O	✔	**tables**	**RA-like**
2008	Facebook	HIVE	✔	O	O	O	✔	✔	O	✔	**tables**	**SQL-like**
2008	Facebook	Cassandra	✔	✔	✔	**EC, record**	O	✔	✔	O	**key-val**	**filter**
2009	other	Voldemort	✔	✔	O	**EC, record**	O	O	O	O	**key-val**	**lookup**
2009	basho	Riak	✔	✔	✔	**EC, record**	**MR**	O			**key-val**	**filter**
2010	Google	Dremel	✔	O	O	O	/	✔	O	✔	**tables**	**SQL-like**
2011	Google	Megastore	✔	✔	✔	**entity groups**	O	/	O	/	**tables**	**filter**
2011	Google	Tenzing	✔	O	O	O	✔	✔	✔	✔	**tables**	**SQL-like**
2011	Berkeley	Spark/Shark	✔	O	O	O	✔	✔	O	✔	**tables**	**SQL-like**
2012	Google	Spanner	✔	✔	✔	✔	?	✔	✔	✔	**tables**	**SQL-like**
2012	Accumulo	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	/	O	O	**ext. record**	**filter**
2013	Cloudera	Impala	✔	O	O	O	✔	✔	O	✔	**tables**	**SQL-like**

A lot of these systems give up joins!

JOINS

Ex: Show all comments by "Sue" on any blog post by "Jim"

Method 1:

- Lookup all blog posts by Jim
- For each post, lookup all comments and filter for "Sue"

Method 2:

- Lookup all comments by Sue
- For each comment, lookup all posts and filter for "Jim"

Method 3:

- Filter comments by Sue, filter posts by Jim,
- Sort all comments by blog id, sort all blogs by blog id
- Pull one from each list to find matches

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	**O**	✔	✔	✔	✔	✔	✔	✔	**tables**	**SQL-like**
2003	memcached	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**key-val**	**lookup**
2004	MapReduce	✔	**O**	**O**	**O**	✔	**O**	**O**	**O**	**key-val**	**MR**
2005	CouchDB	✔	✔	✔	**record**	**MR**	**O**	✔	**O**	**document**	**filter/MR**
2006	BigTable (Hbase)	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**filter/MR**
2007	MongoDB	✔	✔	✔	**EC, record**	**O**	**O**	**O**	**O**	**document**	**filter**
2007	Dynamo	✔	✔	**O**	**O**	**O**	**O**	**O**	**O**	**key-val**	**lookup**
2008	Pig	✔	**O**	**O**	**O**	✔	**/**	**O**	✔	**tables**	**RA-like**
2008	HIVE	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**SQL-like**
2008	Cassandra	✔	✔	✔	**EC, record**	**O**	✔	✔	**O**	**key-val**	**filter**
2009	Voldemort	✔	✔	**O**	**EC, record**	**O**	**O**	**O**	**O**	**key-val**	**lookup**
2009	Riak	✔	✔	✔	**EC, record**	**MR**	**O**			**key-val**	**filter**
2010	Dremel	✔	**O**	**O**	**O**	**/**	✔	**O**	✔	**tables**	**SQL-like**
2011	Megastore	✔	✔	✔	**entity groups**	**O**	**/**	**O**	**/**	**tables**	**filter**
2011	Tenzing	✔	**O**	**O**	**O**	**O**	✔	✔	✔	**tables**	**SQL-like**
2011	Spark/Shark	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**SQL-like**
2012	Spanner	✔	✔	✔	✔	**?**	✔	✔	✔	**tables**	**SQL-like**
2012	Accumulo	✔	✔	✔	**record**	**compat. w/MR**	**/**	**O**	**O**	**ext. record**	**filter**
2013	Impala	✔	**O**	**O**	**O**	✔	✔	**O**	✔	**tables**	**SQL-like**

NOSQL CRITICISM

Two value propositions

- Performance: "I started with MySQL, but had a hard time scaling it out in a distributed environment"
- Flexibility: "My data doesn't conform to a rigid schema"

Stonebraker CACM (blog 2)

NOSQL CRITICISM: FLEXIBILITY ARGUMENT

Who are the customers of NoSQL?

- Lots of startups

Very few enterprises. Why? most applications are traditional OLTP on structured data; a few other applications around the "edges", but considered less important

Stonebraker CACM (blog 2)

NOSQL CRITICISM: FLEXIBILITY ARGUMENT

No ACID Equals No Interest

- Screwing up mission-critical data is no-no-no

Low-level Query Language is Death

- Remember CODASYL?

NoSQL means NoStandards

- One (typical) large enterprise has 10,000 databases. These need accepted standards

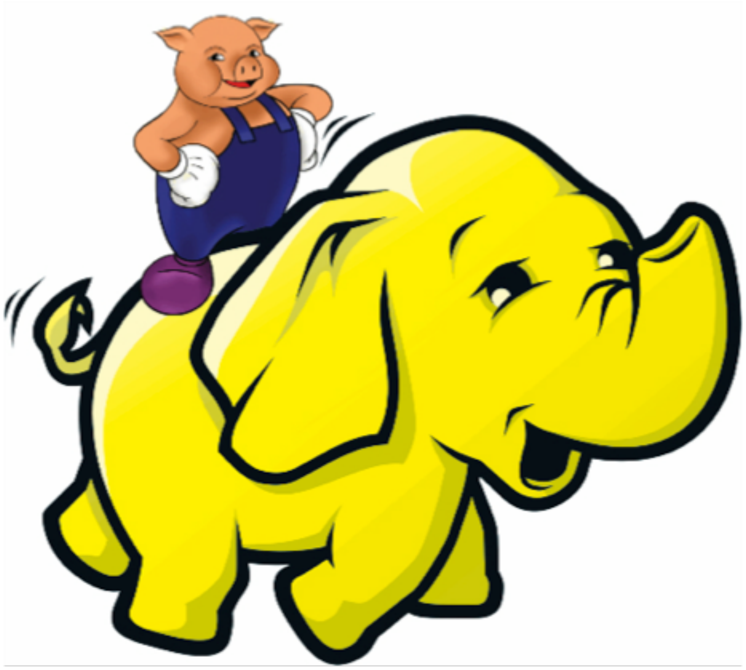
Stonebraker CACM (blog 2)

WHAT IS PIG?

An engine for executing programs on top of Hadoop

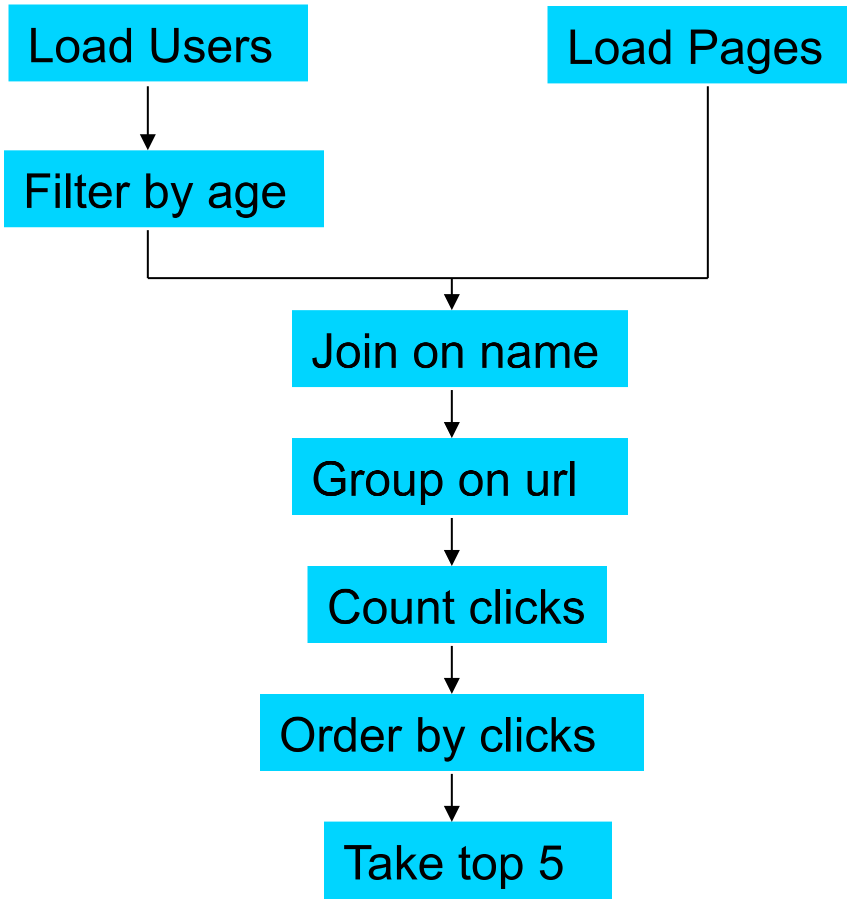
It provides a language, Pig Latin, to specify these programs

An Apache open source project http://pig.apache.org/



WHY USE PIG?

Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited sites by users aged 18 - 25.



IN MAPREDUCE

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
               if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
       FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
       FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
       FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
       FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
       FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
       FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
       FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
       FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
       FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

       JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

170 lines of code, 4 hours to write

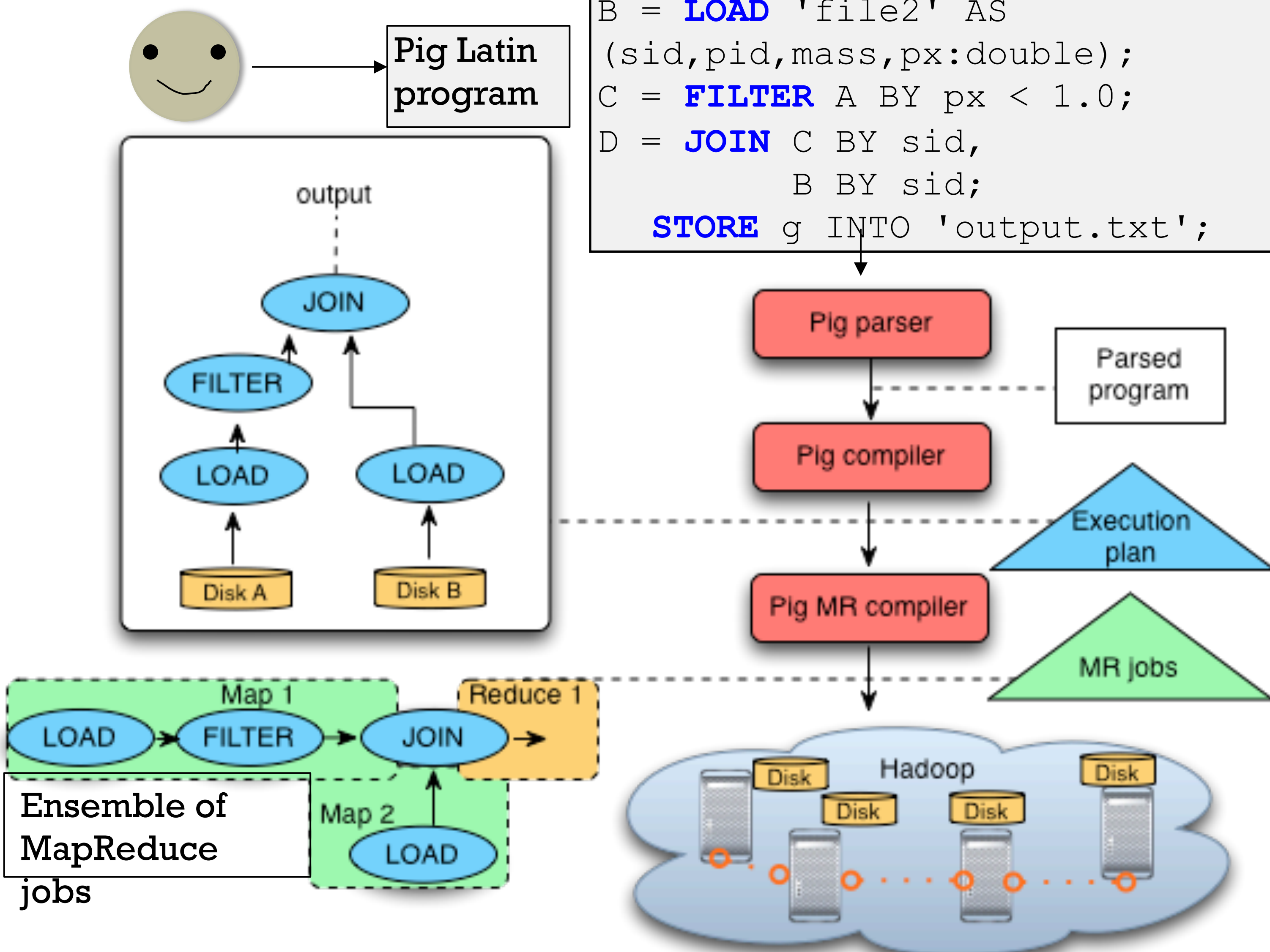
IN PIG LATIN

```
Users = load 'users' as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
       COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites';
```

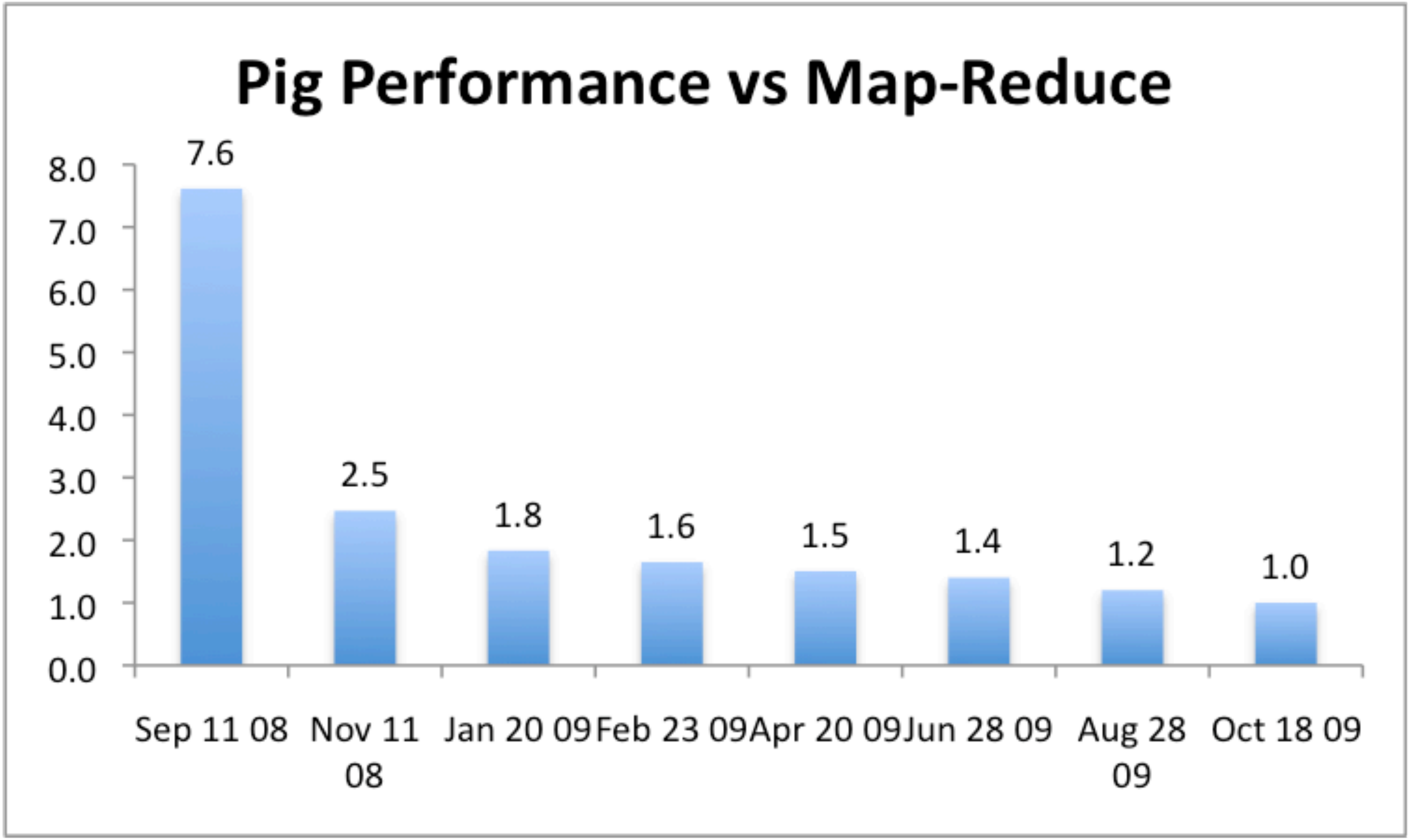
9 lines of code, 15 minutes to write

PIG SYSTEM OVERVIEW

```
A = LOAD 'file1' AS
(sid,pid,mass,px:double);
B = LOAD 'file2' AS
(sid,pid,mass,px:double);
C = FILTER A BY px < 1.0;
D = JOIN C BY sid,
         B BY sid;
   STORE g INTO 'output.txt';
```



BUT CAN IT FLY?



src: Olston

DATA MODEL

Atom: Integer, string, etc.

Tuple:

- Sequence of fields
- Each field of any type

Bag:

- A collection of tuples
- Not necessarily the same type
- Duplicates allowed

Map:

- String literal keys mapped to any type

Key distinction:
Allows Nesting

<1, {<2,3>,<4,6>,<5,7>}, ['apache':'search']>

f1:atom f2:bag f3:map

t = <1, {<2,3>,<4,6>,<5,7>}, ['apache':'search']>

Each field has a name, possibly derived from the operator f1:atom f2:bag f3:map

t = <1, {<2,3>,<4,6>,<5,7>}, ['apache':'search']>

expression	result
$0	1
f2	Bag {<2,3>,<4,6>,<5,7>}
f2.$0	Bag {<2>,<4>,<5>}
f3#'apache'	Atom "search"
sum(f2.$0)	2+4+5

LOAD

Input is assumed to be a bag (sequence of tuples)

Assumes that every dataset is a sequence of tuples

Specify a parsing function with "USING"

Specify a schema with "AS"

```
A = LOAD 'myfile.txt' USING PigStorage('\t') AS
(f1,f2,f3);
```

<1, 2, 3>
<4, 2, 1>
<8, 3, 4>
<4, 3, 3>
<7, 2, 5>
<8, 4, 3>

FILTER: GETTING RID OF DATA

Arbitrary boolean conditions

Regular expressions allowed

```
$0 matches apache
```

```
Y = FILTER A BY f1 == '8';
```

A =	<1, 2, 3>	Y =	<8, 3, 4>
	<4, 2, 1>		<8, 4, 3>
	<8, 3, 4>		
	<4, 3, 3>		
	<7, 2, 5>		
	<8, 4, 3>		

GROUP: GETTING DATA TOGETHER

```
X = GROUP A BY f1;
```

`A =`	<1, 2, 3>	`X =`	<1, {<1, 2, 3>}>
	<4, 2, 1>		<4, {<4, 2, 1>, <4, 3, 3>}>
	<8, 3, 4>		<7, {<7, 2, 5>}>
	<4, 3, 3>		<8, {<8, 3, 4>, <8, 4, 3>}>
	<7, 2, 5>		
	<8, 4, 3>		

first field will be named "group"

second field has name "A"

DISTINCT: GETTING RID OF DUPLICATES

```
Y = DISTINCT A
```

A = <1, 2, 3>
<1, 2, 3>
<8, 3, 4>

Y = <1, 2, 3>
<8, 3, 4>

Claim:

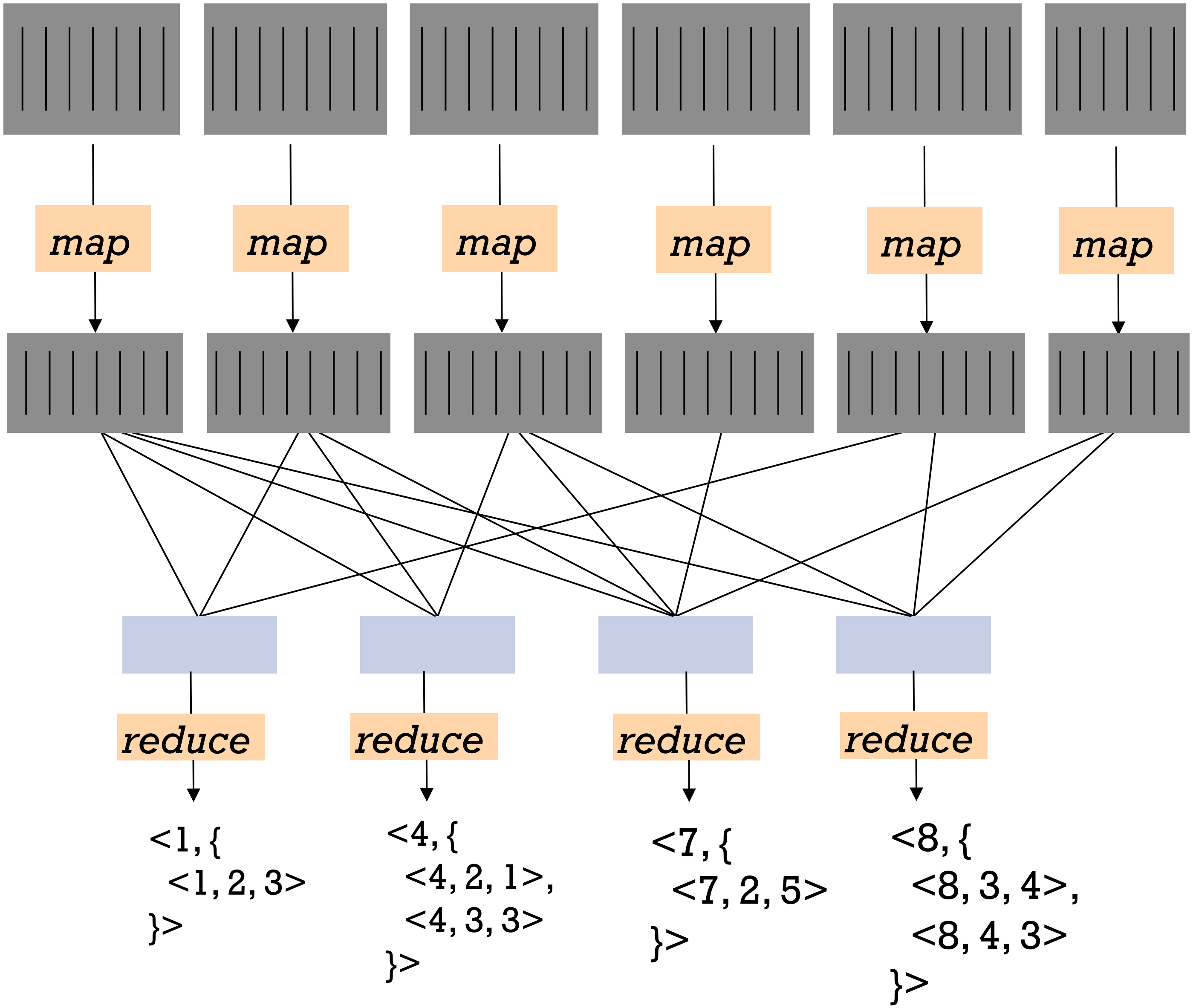
```
DISTINCT A == GROUP A BY (f1, f2, f3)
```

Not quite:

< <1, 2, 3>, {<1,2,3>} >
< <8, 3, 4>, {<8,3,4>} >

GROUP

```
GROUP A BY f1;
```



FOREACH: MANIPULATE EACH TUPLE

```
X = FOREACH A GENERATE f0, f1+f2;
```

You can call UDFs

You can manipulate nested objects

```
Y = GROUP A BY f1;
Z = FOREACH X GENERATE group, X.($1, $2);
```

```
A =  <1, 2, 3>      X =  <1, 5>      Z =  <1, {<2, 3>}>
     <4, 2, 1>           <4, 3>           <4, {<2, 1>, <3, 3>}>
     <8, 3, 4>           <8, 7>           <7, {<2, 5>}>
     <4, 3, 3>           <4, 6>           <8, {<3, 4>, <4, 3>}>
     <7, 2, 5>           <7, 7>
     <8, 4, 3>           <8, 7>
```

USING THE FLATTEN KEYWORD

```
Y = GROUP A BY f1;
Z = FOREACH X GENERATE group, FLATTEN(X);
```

I don’t like this, because FLATTEN has no well defined type. It’s “magic”

A =	<1, 2, 3>	X =	<1, 5>	Z =	<1, 2, 3>
	<4, 2, 1>		<4, 3>		<4, 2, 1>
	<8, 3, 4>		<8, 7>		<4, 3, 3>
	<4, 3, 3>		<4, 6>		<7, 2, 5>
	<7, 2, 5>		<7, 7>		<8, 3, 4>
	<8, 4, 3>		<8, 7>		<8, 4, 3>

IN MAPREDUCE

```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.jobcontrol.Job;
import org.apache.hadoop.mapred.jobcontrol.JobControl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class LoadAndFilterUsers extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }
    public static class Join extends MapReduceBase
        implements Reducer<Text, Text, Text, Text> {

        public void reduce(Text key,
                Iterator<Text> iter,
                OutputCollector<Text, Text> oc,
                Reporter reporter) throws IOException {
           // For each value, figure out which file it's from and
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
               if (value.charAt(0) == '1')
first.add(value.substring(1));
                else second.add(value.substring(1));
                reporter.setStatus("OK");
            }

            // Do the cross product and collect the values
            for (String s1 : first) {
                for (String s2 : second) {
                    String outval = key + "," + s1 + "," + s2;
                    oc.collect(null, new Text(outval));
                    reporter.setStatus("OK");
                }
            }
        }
    }
    public static class LoadJoined extends MapReduceBase
        implements Mapper<Text, Text, Text, LongWritable> {

        public void map(
                Text k,
                Text val,
                OutputCollector<Text, LongWritable> oc,
                Reporter reporter) throws IOException {
            // Find the url
            String line = val.toString();
            int firstComma = line.indexOf(',');
            int secondComma = line.indexOf(',', firstComma);
            String key = line.substring(firstComma, secondComma);
            // drop the rest of the record, I don't need it anymore,
            // just pass a 1 for the combiner/reducer to sum instead.
            Text outKey = new Text(key);
            oc.collect(outKey, new LongWritable(1L));
        }
    }
    public static class ReduceUrls extends MapReduceBase
        implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

        public void reduce(
                Text key,
                Iterator<LongWritable> iter,
                OutputCollector<WritableComparable, Writable> oc,
                Reporter reporter) throws IOException {
            // Add up all the values we see

            long sum = 0;
            while (iter.hasNext()) {
                sum += iter.next().get();
                reporter.setStatus("OK");
            }

            oc.collect(key, new LongWritable(sum));
        }
    }
    public static class LoadClicks extends MapReduceBase
        implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

        public void map(
                WritableComparable key,
                Writable val,
                OutputCollector<LongWritable, Text> oc,
                Reporter reporter) throws IOException {
            oc.collect((LongWritable)val, (Text)key);
        }
    }
    public static class LimitClicks extends MapReduceBase
        implements Reducer<LongWritable, Text, LongWritable, Text> {

        int count = 0;
        public void reduce(
            LongWritable key,
            Iterator<Text> iter,
            OutputCollector<LongWritable, Text> oc,
            Reporter reporter) throws IOException {

            // Only output the first 100 records
            while (count < 100 && iter.hasNext()) {
                oc.collect(key, iter.next());
                count++;
            }
        }
    }
    public static void main(String[] args) throws IOException {
        JobConf lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);
        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
            new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
            new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}
```

170 lines of code, 4 hours to write

IN PIG LATIN

```
Users = load 'users' as (name, age);
Fltrd = filter Users by
        age >= 18 and age <= 25;
Pages = load 'pages' as (user, url);
Jnd = join Fltrd by name, Pages by
user;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate group,
       COUNT(Jnd) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites';
```

9 lines of code, 15 minutes to write

COGROUP: GETTING DATA TOGETHER

```
C = COGROUP A BY f1, B BY $0;
```

A = <1, 2, 3>
<4, 2, 1>
<8, 3, 4>
<4, 3, 3>
<7, 2, 5>
<8, 4, 3>

B = <2, 4>
<8, 9>
<1, 3>
<2, 7>
<2, 9>
<4, 6>
<4, 9>

C = <1, {<1, 2, 3>}, {<1, 3>}>
<2, {}, {<2, 4>, <2, 7>, <2, 9>}>
<4, {<4, 2, 1>, <4, 3, 3>}, {<4, 6>,<4, 9>}>
<7, {<7, 2, 5>}, {}>
<8, {<8, 3, 4>, <8, 4, 3>}, {<8, 9>}>

JOIN: A SPECIAL CASE OF COGROUP

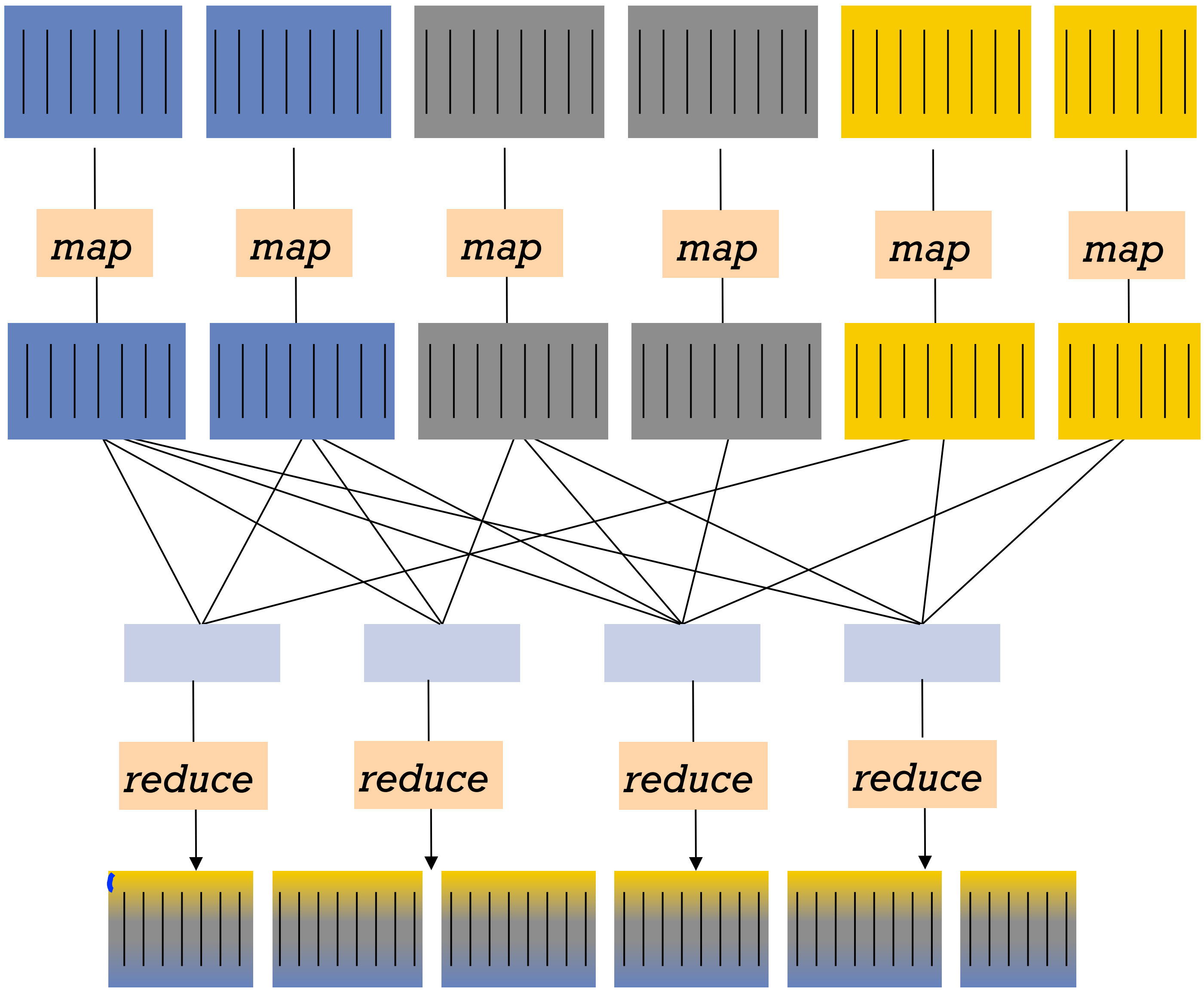
```
C = JOIN A BY $0, B BY $0;
```

A =	<1, 2, 3>	B =	<2, 4>	C =	<1, 2, 3, 1, 3>
	<4, 2, 1>		<8, 9>		<4, 2, 1, 4, 6>
	<8, 3, 4>		<1, 3>		<4, 2, 1, 4, 9>
	<4, 3, 3>		<2, 7>		<4, 3, 3, 4, 6>
	<7, 2, 5>		<2, 9>		<4, 3, 3, 4, 9>
	<8, 4, 3>		<4, 6>		<8, 3, 4, 8, 9>
			<4, 9>		<8, 4, 3, 8, 9>

COGROUP and JOIN can both on multiple datasets

- Think about why this works

JOIN MULTIPLE RELATIONS



THREE SPECIAL JOIN ALGORITHMS

Replicated

- One table is very small, one is big
- Replicate the small one

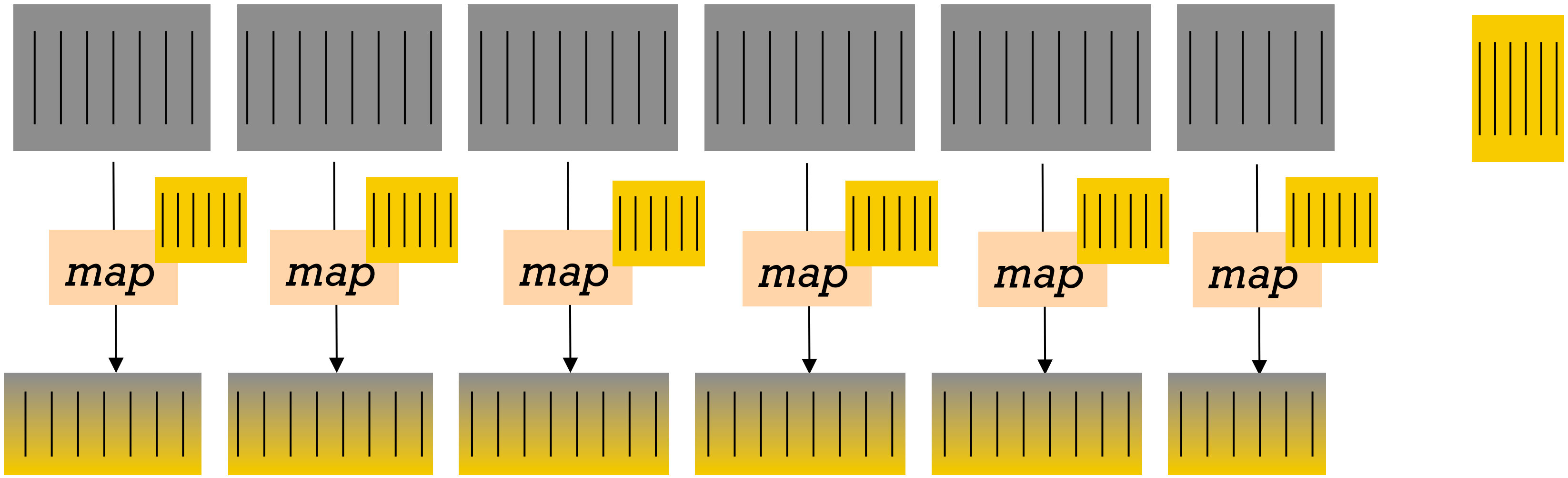
Skewed

- When one join key is associated with most of the data, we're in trouble
- handle these cases differently

Merge

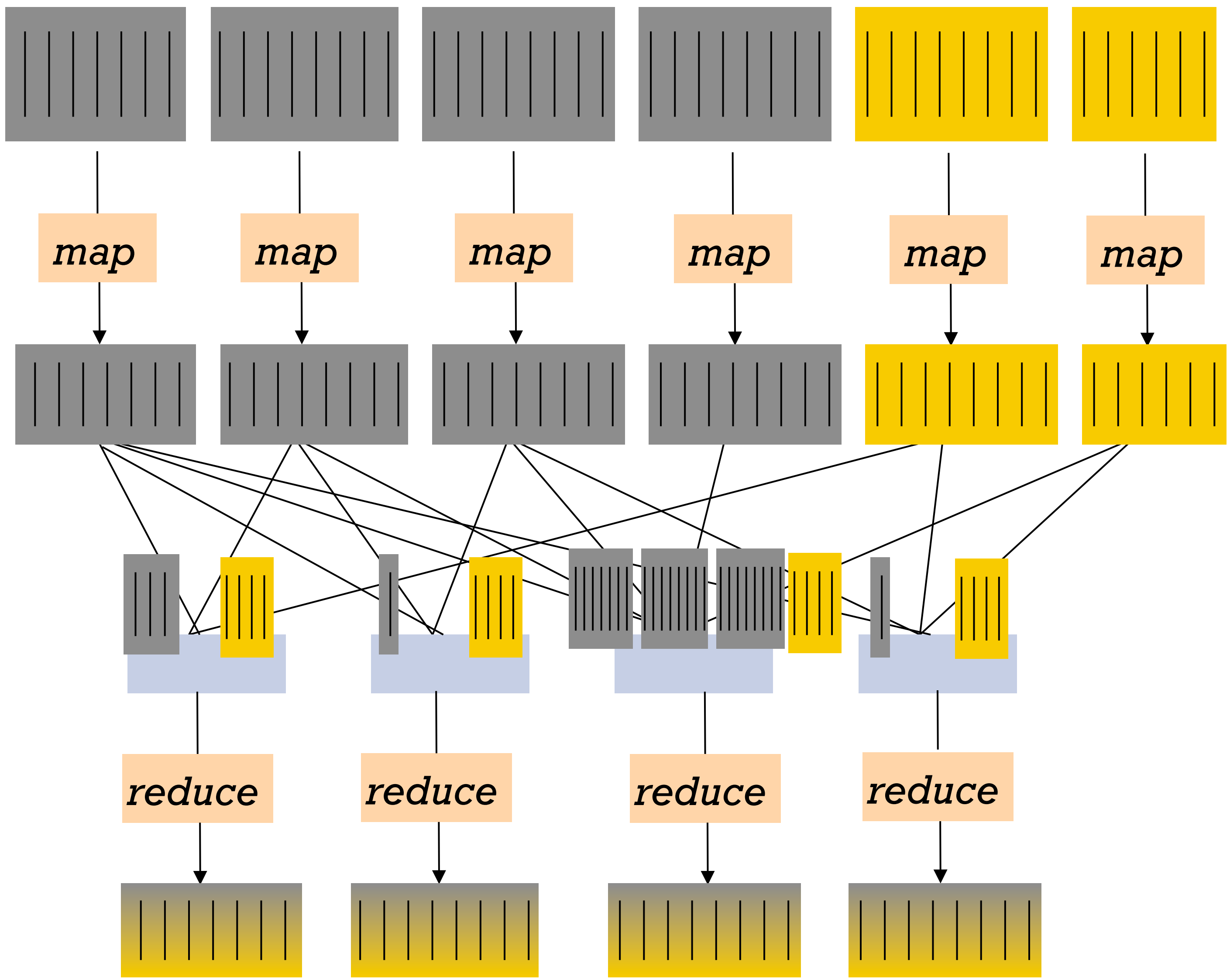
- If the two datasets are already grouped/sorted on the correct attribute, we can compute the join in the Map phase

REPLICATED JOIN

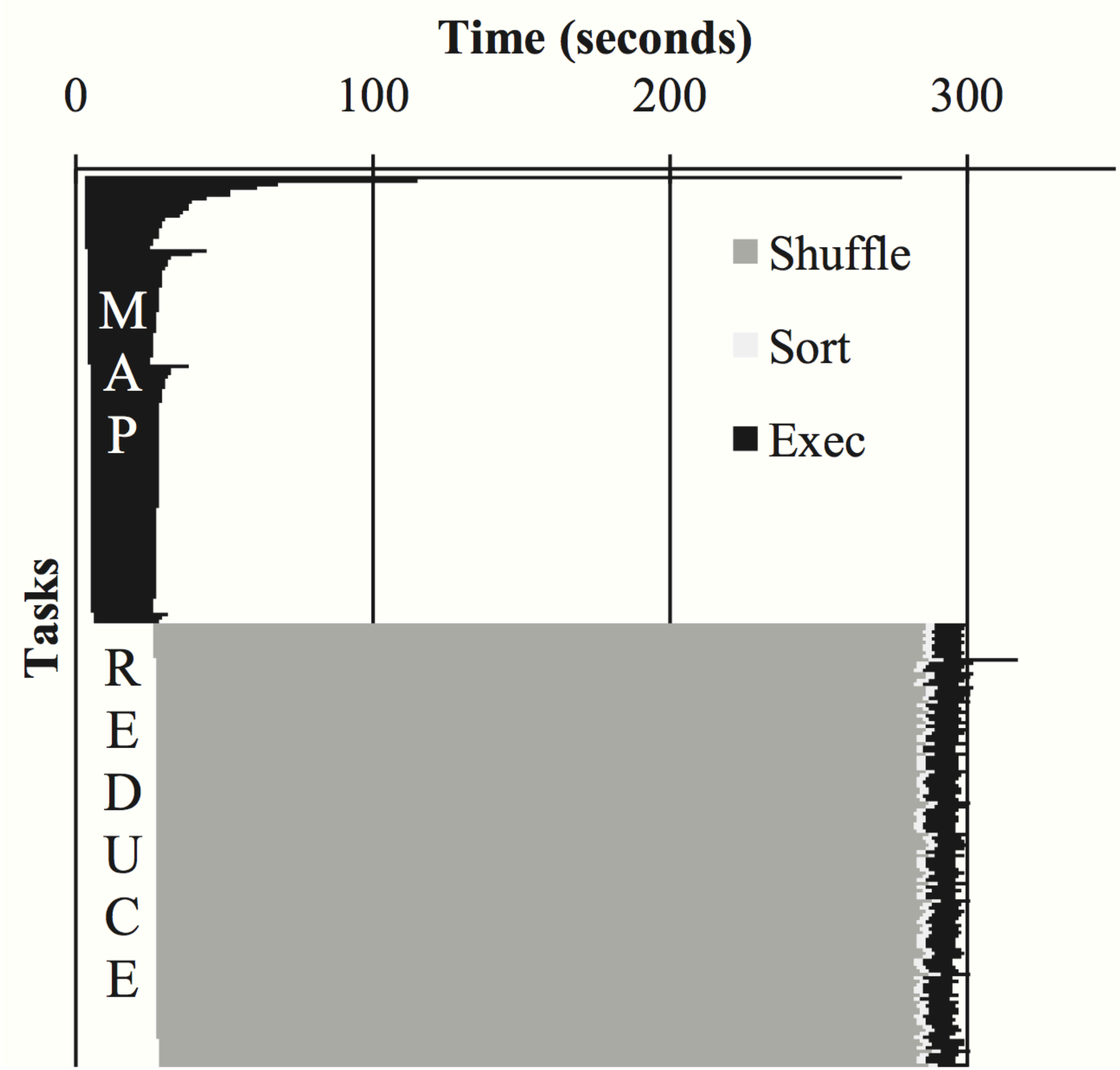


- Each mapper pulls a copy of the small relation from HDFS
- Small relation must fit in main memory
- You'll see this called "Broadcast Join" as well

SKEW JOIN



THE PROBLEM OF SKEW



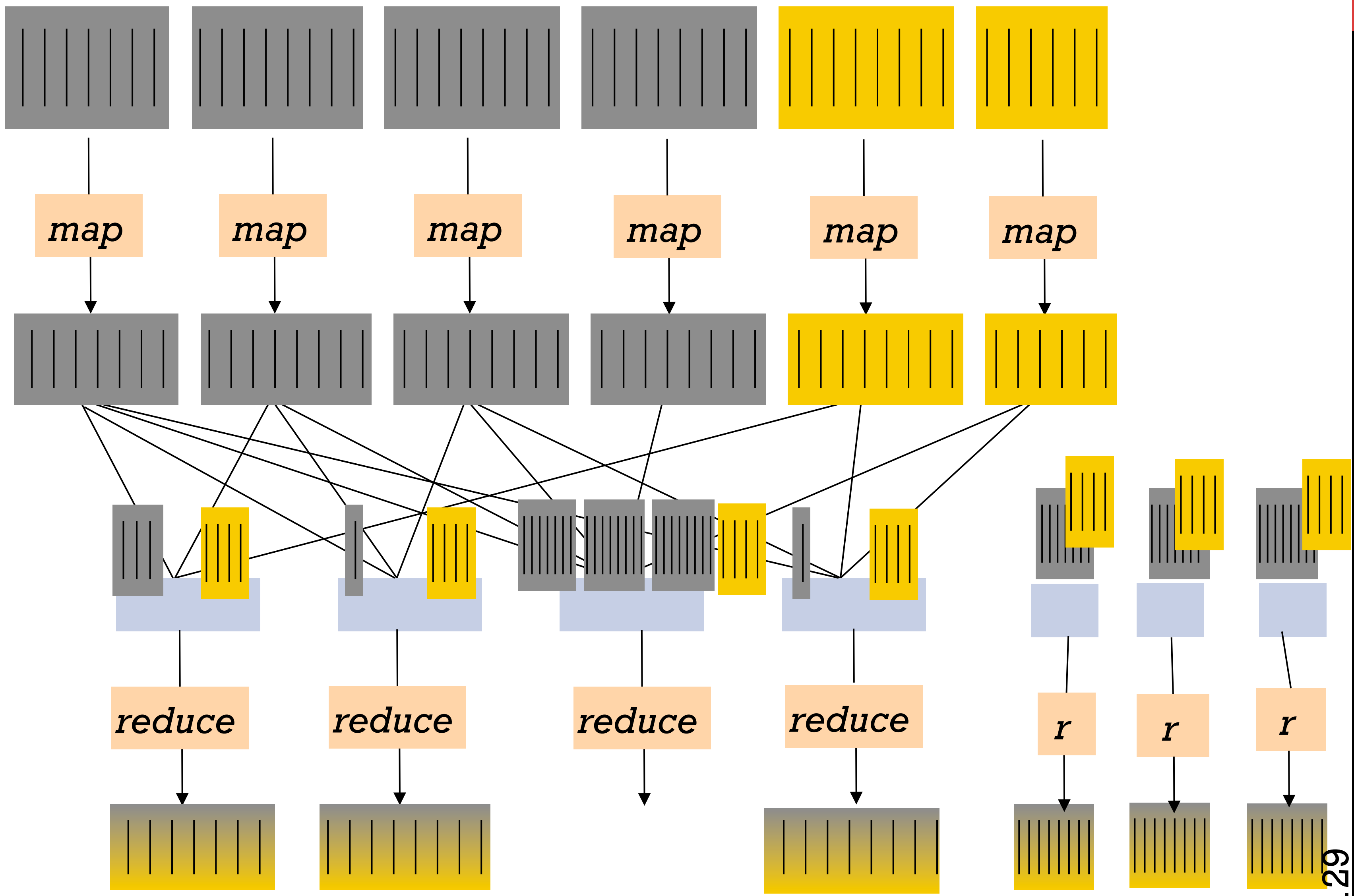
One task takes 5X longer than the average! Very little benefit to parallelism

If someone asks you "What is the biggest performance problem with MapReduce in practice?"

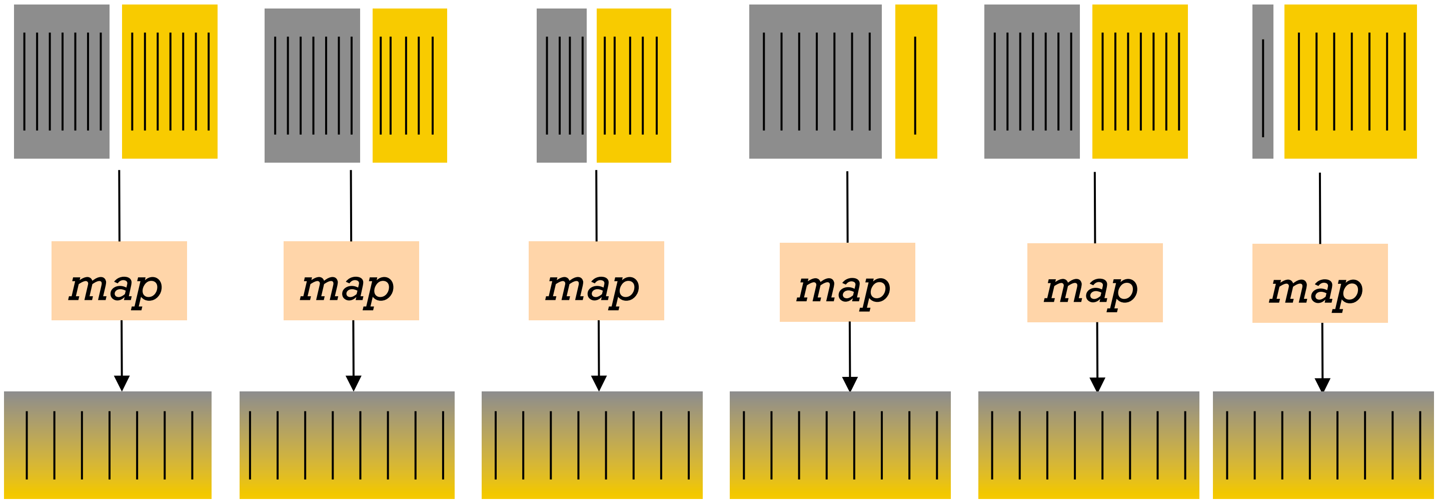
Say: "Stragglers!"

Kwon 2012

SKEW JOIN



MERGE JOIN



- Each mapper already has local access to the records from both relations, grouped and sorted by the join key.
- Just read in both relations from disk, in order.

WHY COGROUP AND NOT JOIN?

JOIN is a two-step process

- Create groups with shared keys
- Produce joined tuples

COGROUP only performs the first step

- You might do different processing on the groups
- e.g., count the tuples

```
join_result = JOIN A BY $0, B BY $0;

temp = COGROUP A BY $0, B BY $0;

join_result = FOREACH temp GENERATE
              FLATTEN(results), FLATTEN(revenue);
```

OTHER COMMANDS

- **STORE**
- **UNION**
- **CROSS**
- **DUMP**
- **ORDER**

STORE bagName INTO 'myoutput.txt'
USING some_func();

EXAMPLE

LOAD

```
A = LOAD 'traffic.dat' AS (ip, time, url);
B = GROUP A BY ip;

C = FOREACH B GENERATE group AS ip,
COUNT(A);

D = FILTER C BY ip IS '192.168.0.1';
      OR ip IS '192.168.0.0';

STORE D INTO 'local_traffic.dat';
```

Adapted from slides by Oliver Kennedy, U of Buffalo

EXAMPLE

```
A = LOAD 'traffic.dat' AS (ip, time, url);

B = GROUP A BY ip;

C = FOREACH B GENERATE group AS ip,
COUNT(A);

D = FILTER C BY ip IS '192.168.0.1';
      OR ip IS '192.168.0.0';

STORE D INTO 'local_traffic.dat';
```

LOAD

GROUP

Adapted from slides by Oliver Kennedy, U of Buffalo

EXAMPLE

```
A = LOAD 'traffic.dat' AS (ip, time, url);
B = GROUP A BY ip;

C = FOREACH B GENERATE group AS ip,
COUNT(A);

D = FILTER C BY ip IS '192.168.0.1';
      OR ip IS '192.168.0.0';

STORE D INTO 'local_traffic.dat';
```

LOAD

GROUP

FOREACH

Adapted from slides by Oliver Kennedy, U of Buffalo

EXAMPLE

```
A = LOAD 'traffic.dat' AS (ip, time, url);
B = GROUP A BY ip;

C = FOREACH B GENERATE group AS ip,
COUNT(A);

D = FILTER C BY ip IS '192.168.0.1';
      OR ip IS '192.168.0.0';

STORE D INTO 'local_traffic.dat';
```

LOAD

GROUP

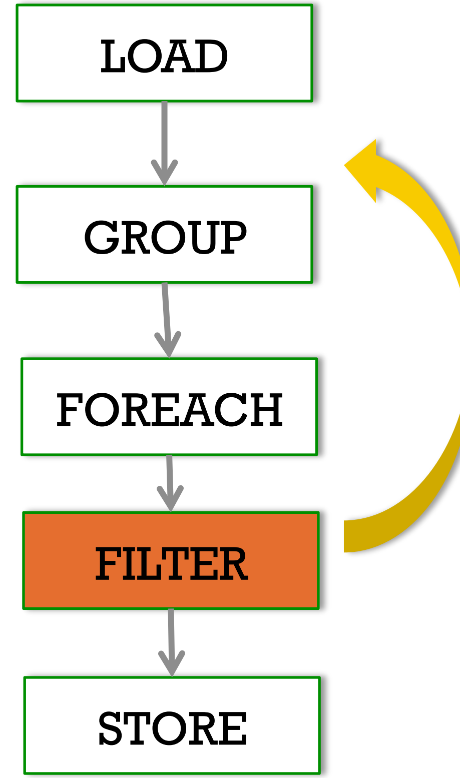
FOREACH

FILTER

Adapted from slides by Oliver Kennedy, U of Buffalo

EXAMPLE

```
A = LOAD 'traffic.dat' AS (ip, time, url);
B = GROUP A BY ip;
C = FOREACH B GENERATE group AS ip,
        COUNT(A);
D = FILTER C BY ip IS '192.168.0.1'
        OR ip IS '192.168.0.0';
STORE D INTO 'local_traffic.dat';
```

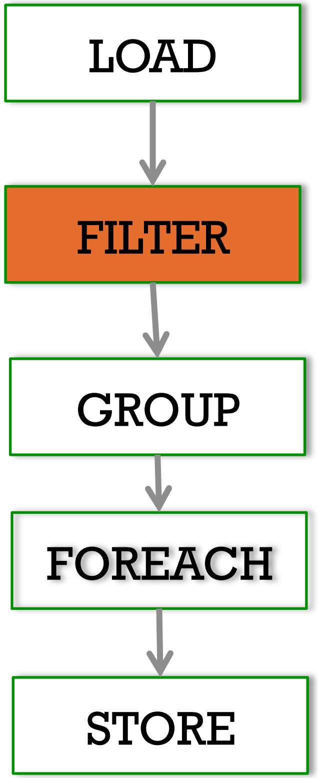


Algebraic Optimization!

Adapted from slides by Oliver Kennedy, U of Buffalo

EXAMPLE

```
A = LOAD 'traffic.dat' AS (ip, time, url);
B = GROUP A BY ip;
C = FOREACH B GENERATE group AS ip,
       COUNT(A);
D = FILTER C BY ip IS '192.168.0.1'
       OR ip IS '192.168.0.0';
STORE D INTO 'local_traffic.dat';
```

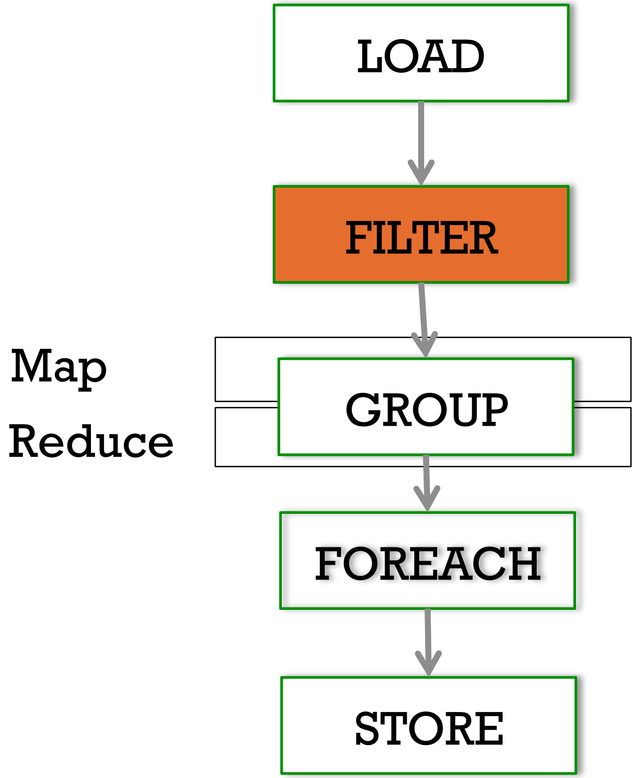


Lazy Evaluation:
No work is done
until STORE

Adapted from slides by Oliver Kennedy, U of Buffalo

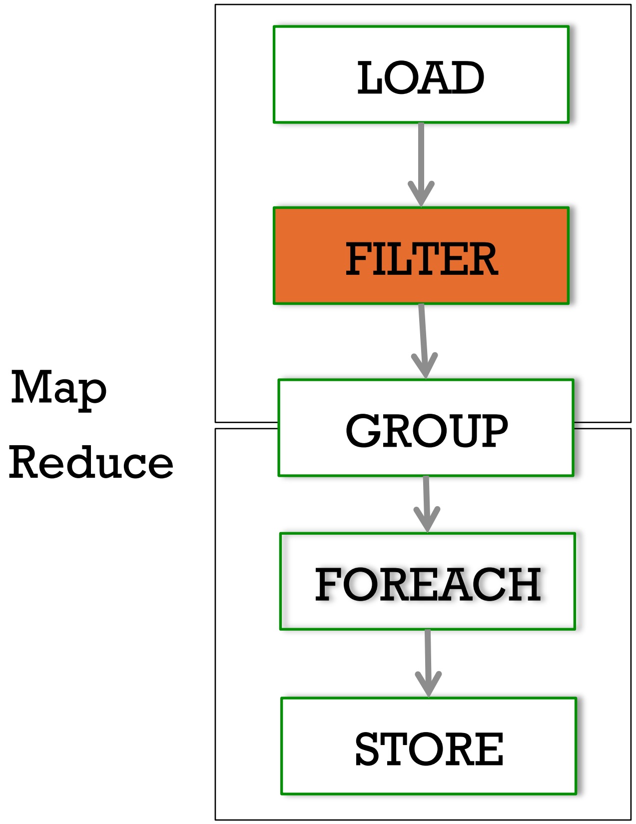
EXAMPLE

Create a MR job for each COGROUP



Adapted from slides by Oliver Kennedy, U of Buffalo

EXAMPLE



1) Create a MR job for each COGROUP

2) Add other commands where possible

Certain commands require their own MR job (e.g., ORDER)

Adapted from slides by Oliver Kennedy, U of Buffalo

REVIEW

NoSQL

- "NoSchema", "NoTransactions", "NoLanguage"
- A "reboot" of data systems focusing on just high-throughput reads and writes
- But: A clear trend towards re-introducing schemas, languages, transactions at full scale
    - Google's Spanner system, for example

Pig

- An RA-like language layer on Hadoop
- But not a pure relational data model
- "Schema-on-Read" rather than "Schema-on-write"

SLIDES CAN BE FOUND AT:

TEACHINGDATASCIENCE.ORG