Distributed Systems Reasoning

Pipeline & Batch Systems (Part 1)
Orchestration and Serving (Part 2)

John Looney, Production Engineer, Facebook Dublin

These slides: https://tinyurl.com/srecon-dist-2019

- Sit at the front
- When you can add more colour, do so!
- Speak up

11:00 - Part 1

11:45 - Part 2

12:30 - Lunch

Pipeline & Batch Systems (Part 1)

In which our heroes will:

- Learn about Orchestration (placement of data/servers) and Locking
- Understand how to choose between batch data storage technologies
- Understand how to build a 1000+ node filesystem and database
- Read and critique a design document for a 'recommendation engine'

Orchestration: Finding, Ordering, Sharding

We often need to describe;

- data stores & inputs
- units of processing (servers, pipeline stages)

And describe how..

- ..data enters the system
- ..the system breaks data into parts
- ..those smaller parts are processed
- ..the processors communicate
- ..we know processing is done

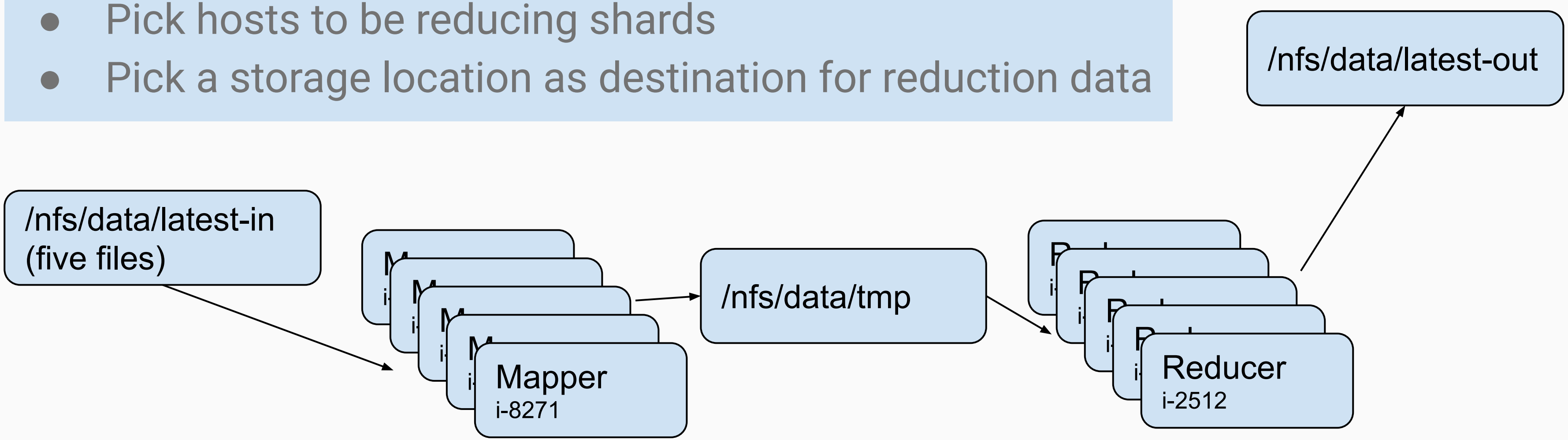
What tech we'll discuss today...

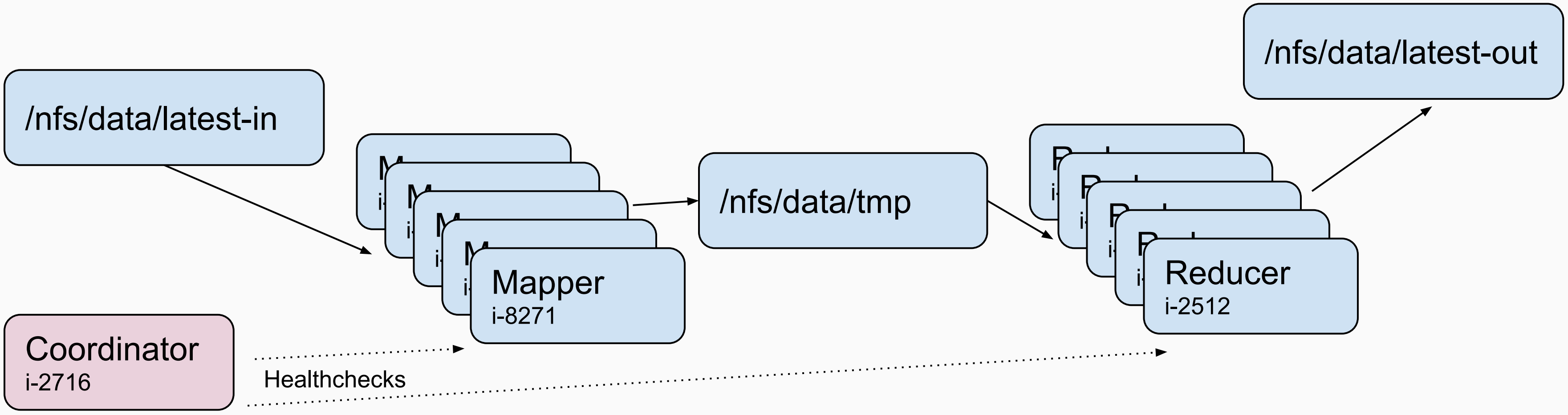
- Terraform
- Zookeeper, etcd
- Kafka, Pubsub, SQS
- Apache Spark, Storm
- DNS, Consul
- Mesos, Kubernetes, AWS CE

(no, you don't need to know what they are, you can read up on them tomorrow)

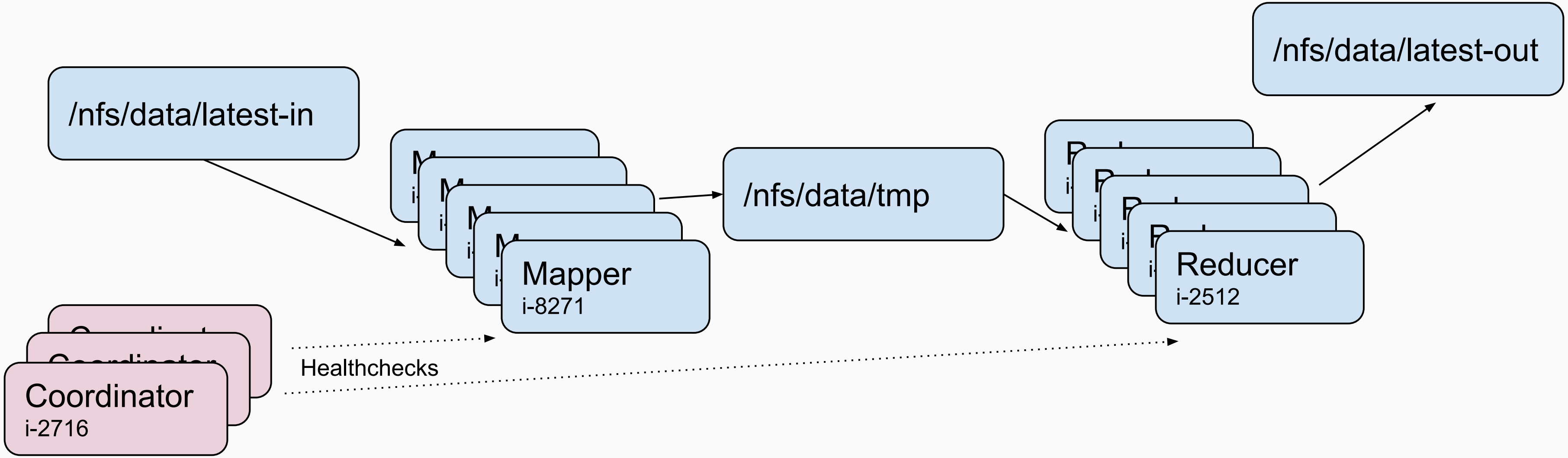
Old School 'prescription'

- Pick a host to be a primary
- Pick hosts to be mapping shards
- Pick hosts to be reducing shards
- Pick a storage location as destination for reduction data

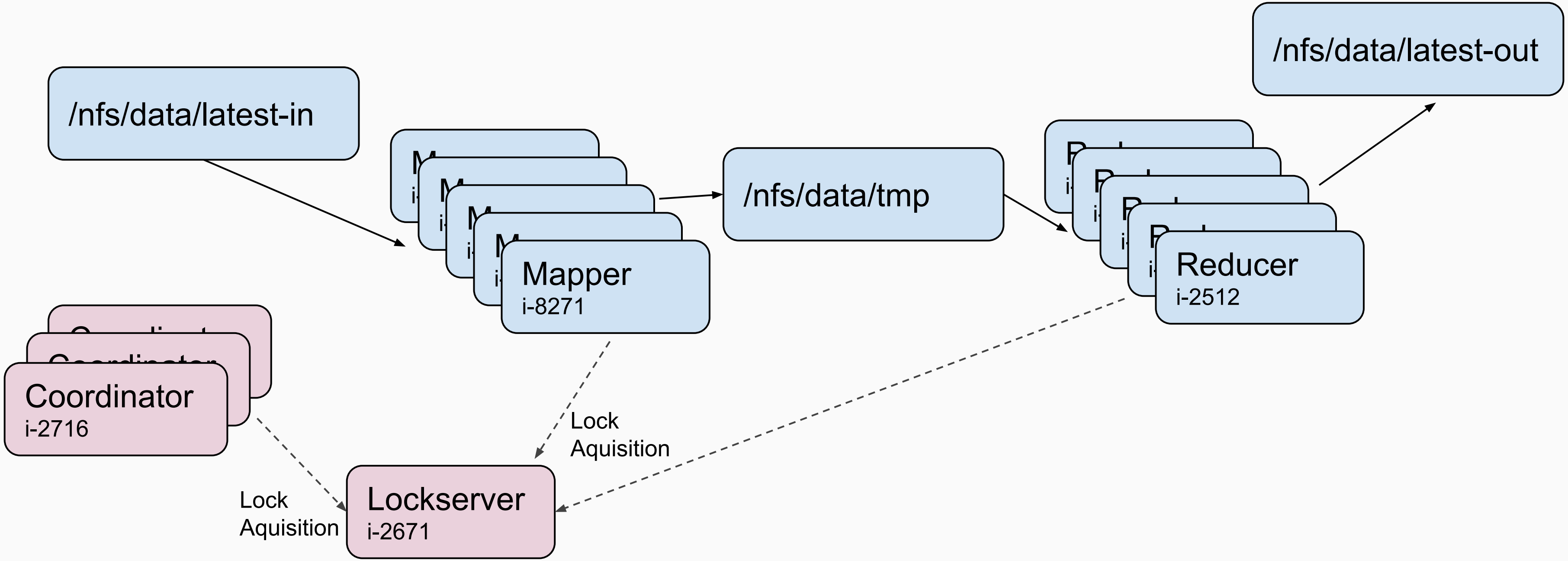




Let's make this Dynamically Scaled!



Let's make this resilient!



Let's make this even MORE resilient!

Making Reliability Worse: Failover

- What if both primaries are OK, just can't do network ?
- What if primaries are OK, but can't do heartbeats ?
- What if the standby primary takes over...and messes up ?
- What if the standby primary takes over, kills the old primary, but it's running old software ?

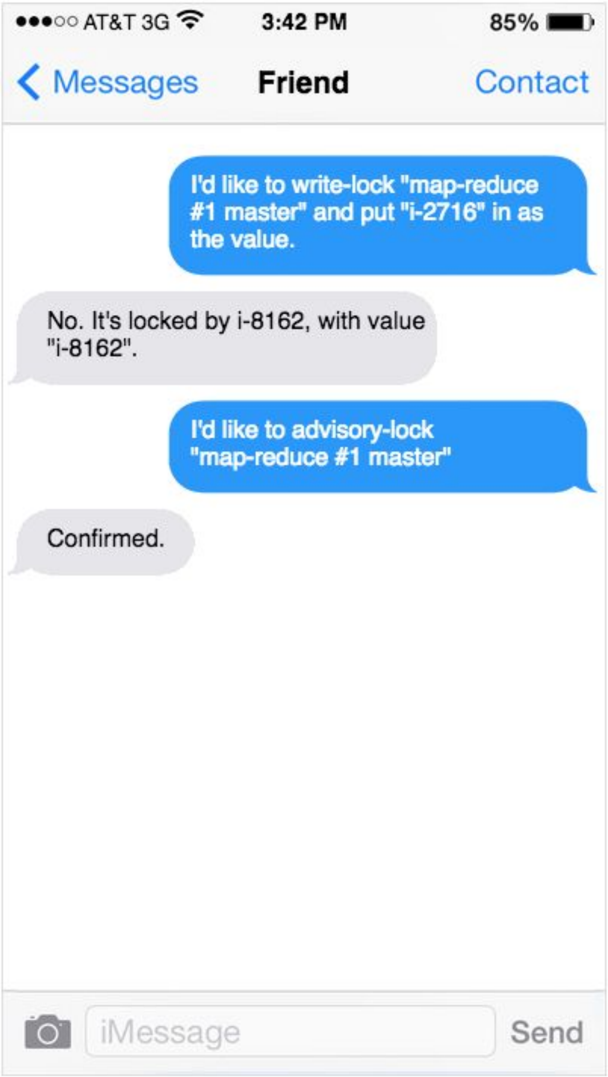
https://pxhere.com/en/photo/1370218

Image: Allan Alfjo, CC BY 2.0

Lockservers; locks

Forget failover, outsource it to a Lockserver!

- Write locks; change the "locked" value
- Advisory locks; subscribe for updates



I'd like to write-lock "map-reduce #1 master" and put "i-2716" in as the value.

No. It's locked by i-8162, with value "i-8162".

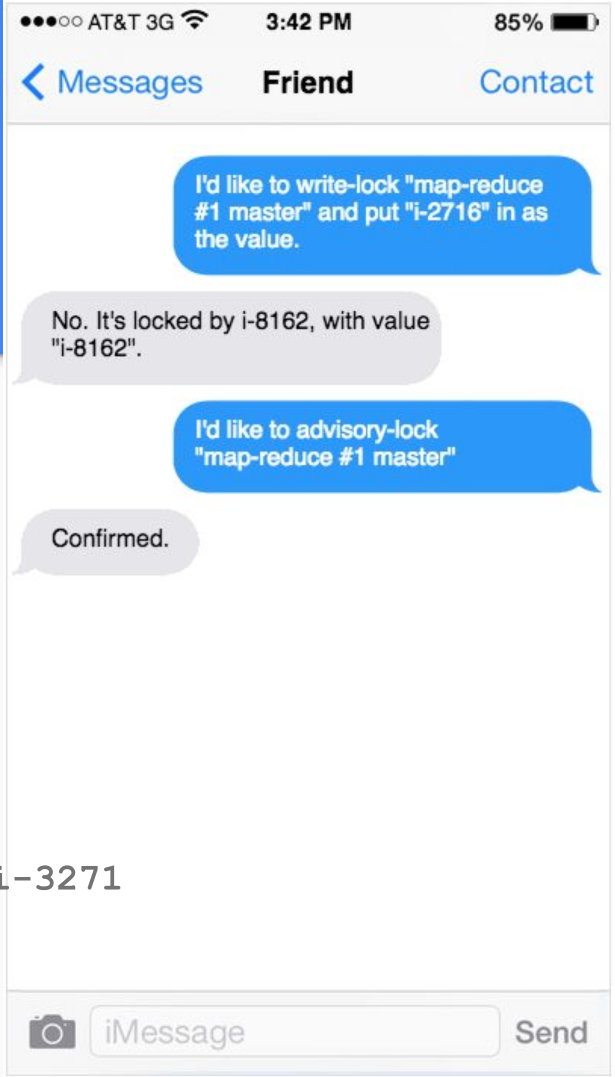
I'd like to advisory-lock "map-reduce #1 master"

Confirmed.

Lockservers; discovery

```
$ curl primary.1.mapreduce.lockserver
HTTP/1.1 301 Moved Permanently
Content-Type: text/html
Location: https://i-3271:10001/
```

```
$ dig srv primary._mapreduce.example.org
_primary._mapreduce.example.org. 29 IN SRV 10 10 10001 i-3271
```



I'd like to write-lock "map-reduce #1 master" and put "i-2716" in as the value.

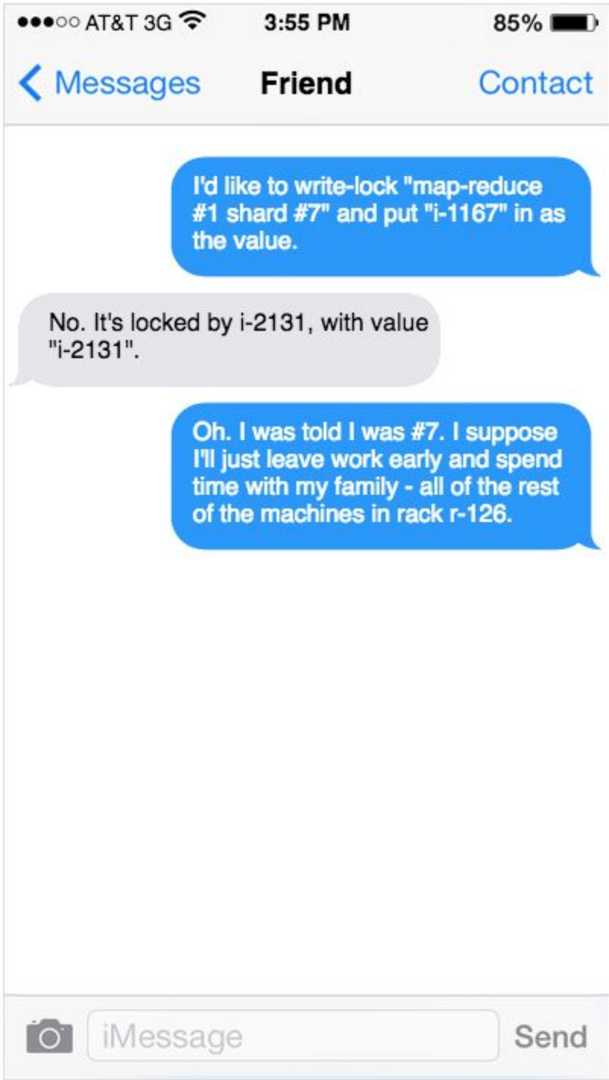
No. It's locked by i-8162, with value "i-8162".

I'd like to advisory-lock "map-reduce #1 master"

Confirmed.

Lockservers; discovery

It's not just for primaries: the secondaries can use lockservers for check-in too!



I'd like to write-lock "map-reduce #1 shard #7" and put "i-1167" in as the value.

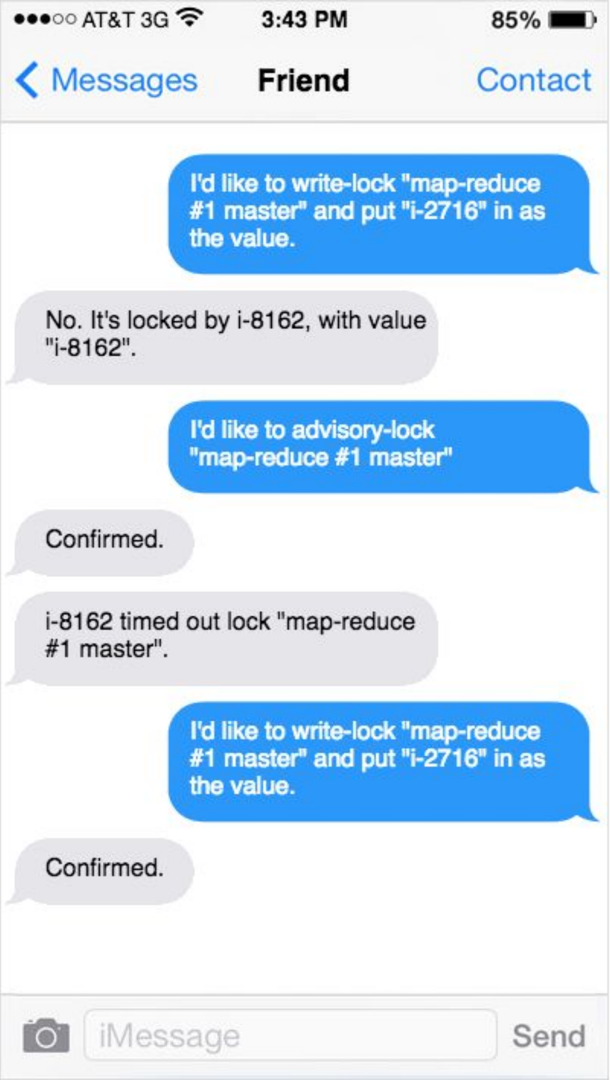
No. It's locked by i-2131, with value "i-2131".

Oh. I was told I was #7. I suppose I'll just leave work early and spend time with my family - all of the rest of the machines in rack r-126.

Lockservers; failover

The King is dead!

Long live the King!



I'd like to write-lock "map-reduce #1 master" and put "i-2716" in as the value.

No. It's locked by i-8162, with value "i-8162".

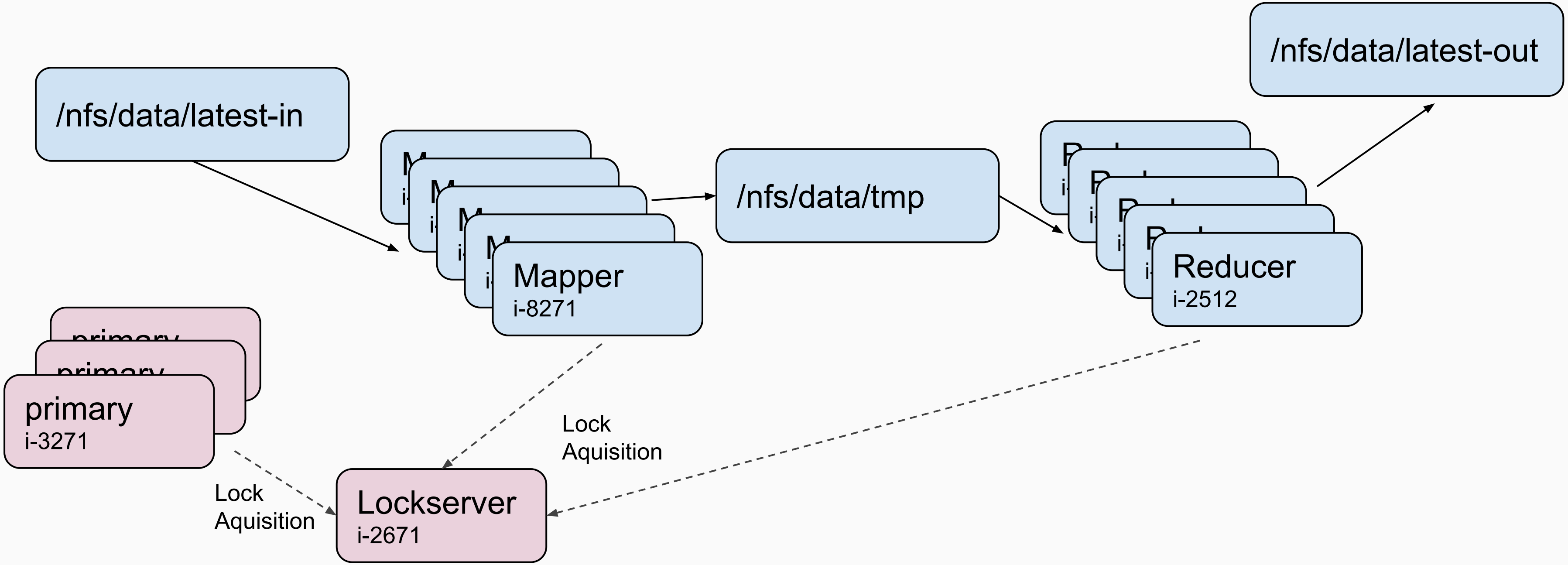
I'd like to advisory-lock "map-reduce #1 master"

Confirmed.

i-8162 timed out lock "map-reduce #1 master".

I'd like to write-lock "map-reduce #1 master" and put "i-2716" in as the value.

Confirmed.



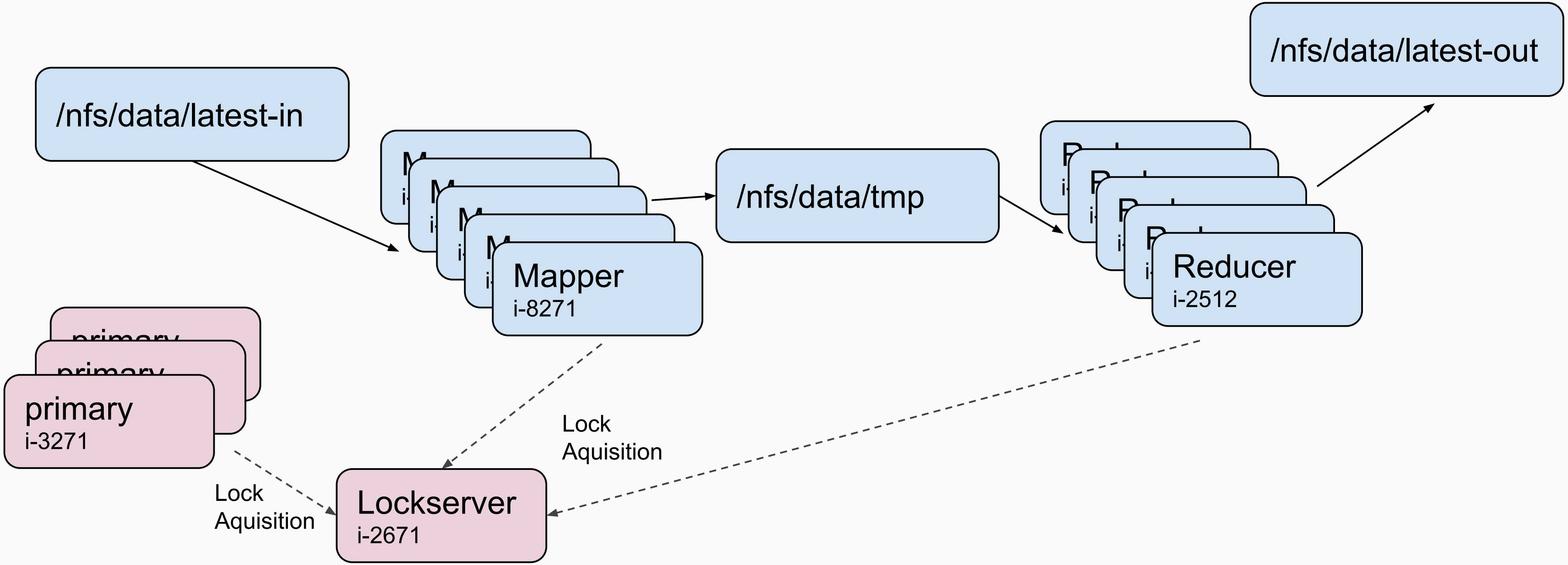
What happens when a Mapper can't talk to a primary anymore ?

Clients; self-resolution

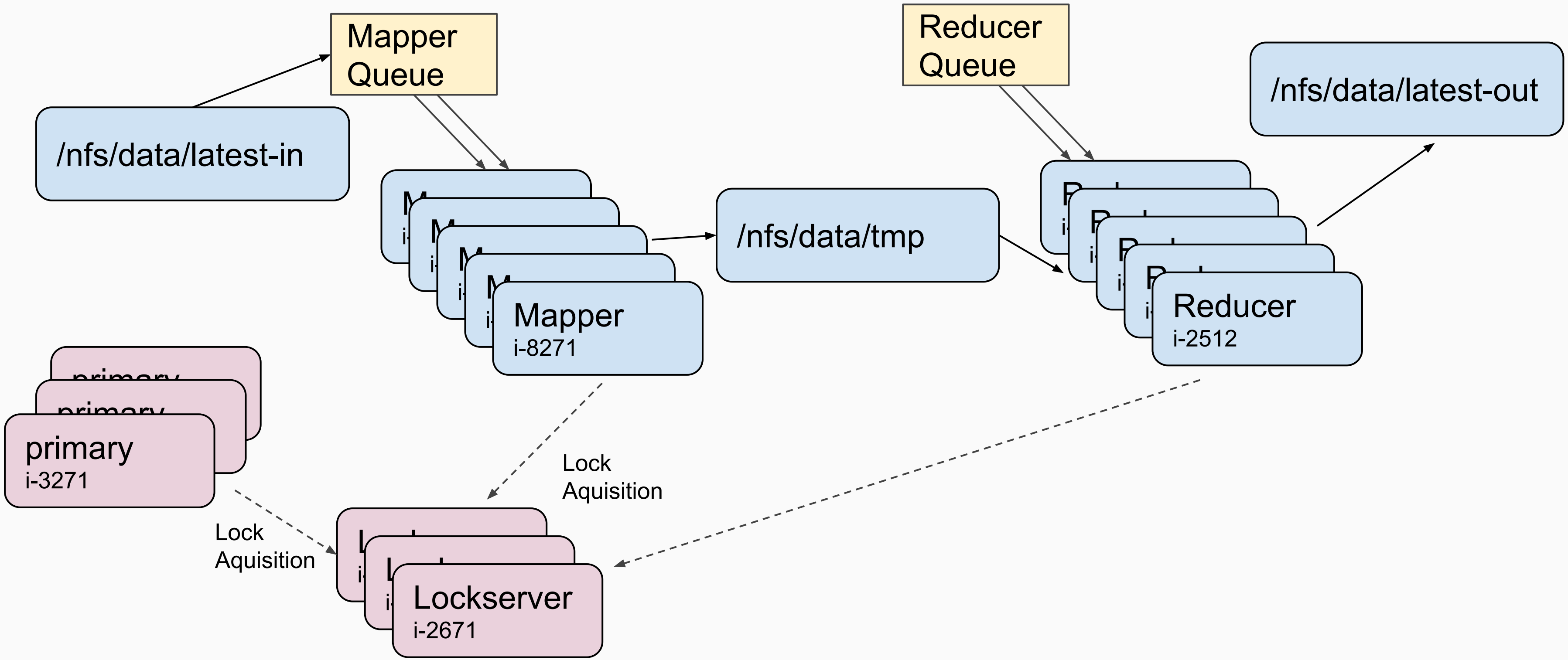
```
$ echo $SHARD_TYPE
mapper
$ echo $SHARD_NUMBER
1
$ dig +short srv ${SHARD_TYPE}.${SHARD_NUMBER}._mapreduce.example.org
_mapper.1._mapreduce.example.org. 29 IN SRV 10 10 10002 i-1238
$ hostname
I-1231
$ if [ "$(dig +short srv ${SHARD_TYPE}.${SHARD_NUMBER}._mapreduce.example.org|
   cut -f8 -d' ')" != $(hostname) ] ; then
 reboot -q
fi
```

So, what lockserver ?

- Zookeeper (old, complex)
- Cheap & Nasty hacks, like locking a row in a database
- Npm lockserver.js
- Clustered Redis
- Etcd (the new hotness)
- Consul (complete solution)



What happens when the lockserver falls over ?



Let's just add replicas! Though...they need to come to a consensus.

Let's talk about Consensus

Given a set processes, each chooses an initial value:

- All non-faulty processes **eventually** decide on a value
- A majority of processes decide on the **same** value
- The decision must have **proposed by one of the processes**

These three properties are referred to as **termination**, **agreement** and **validity**

Consensus Challenges

- Is it broken, or is it slow ?
- Is it unresponsive, or was a message lost en-route ?
- ‘Impossibility of Distributed Consensus with One Faulty Process’
  - Cannot be 100% sure of system's initial state
  - In an asynchronous system, ordering matters for changing *unsure* state to *sure*
  - In any attempt (round) at consensus, things may be *undecided*
  - *Undecided* last time does not guarantee *decided* this time

Consensus; Requirements

- Given multiple servers, each can propose a value for the log entry
- All will agreed on a single value
- Only one value is chosen
- A server is not told a value is 'chosen' unless it definitely has been
- A value has to be chosen within a timeout
- All servers will be told about the value chosen, eventually

Consensus; Raft

- There is 1 Leader, N-1 followers
- Changes to Log are sent to Leader
- If there is no Leader, an election is called
    - Each Follower asks all others to follow
- Heartbeats (~100ms) from a Leader postpones new elections
- Odd numbers of followers are most efficient

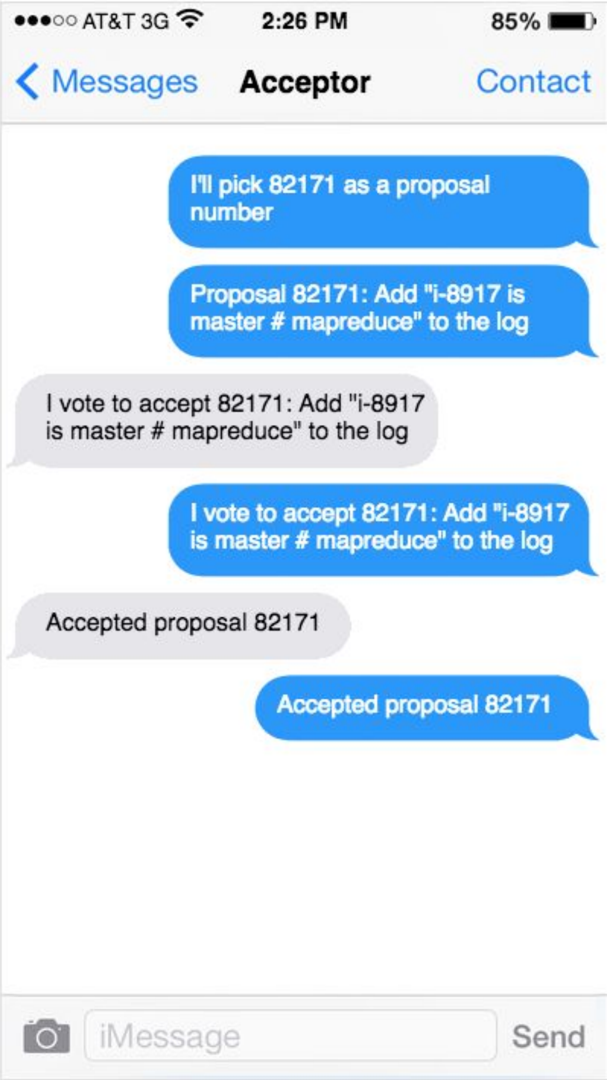
Consensus; Paxos

- All servers can Propose and Accept changes
- Complex proposal system, where each node can propose a change
  - If a majority accept, any subsequent proposal that conflicts is dropped
  - Must increment & persist proposal numbers
- Once a proposal is made, nodes ***broadcast*** if they Accept

More detail: https://www.microsoft.com/en-us/research/publication/paxos-made-simple/

Consensus; Paxos

- There are similarities to Raft, if every log addition was an election.
- Slower than raft, but multi-primary
- Multi-Paxos can use leader-election to make things go faster (just one proposer at a time, until Leader dies)



Acceptor

I'll pick 82171 as a proposal number

Proposal 82171: Add "i-8917 is master # mapreduce" to the log

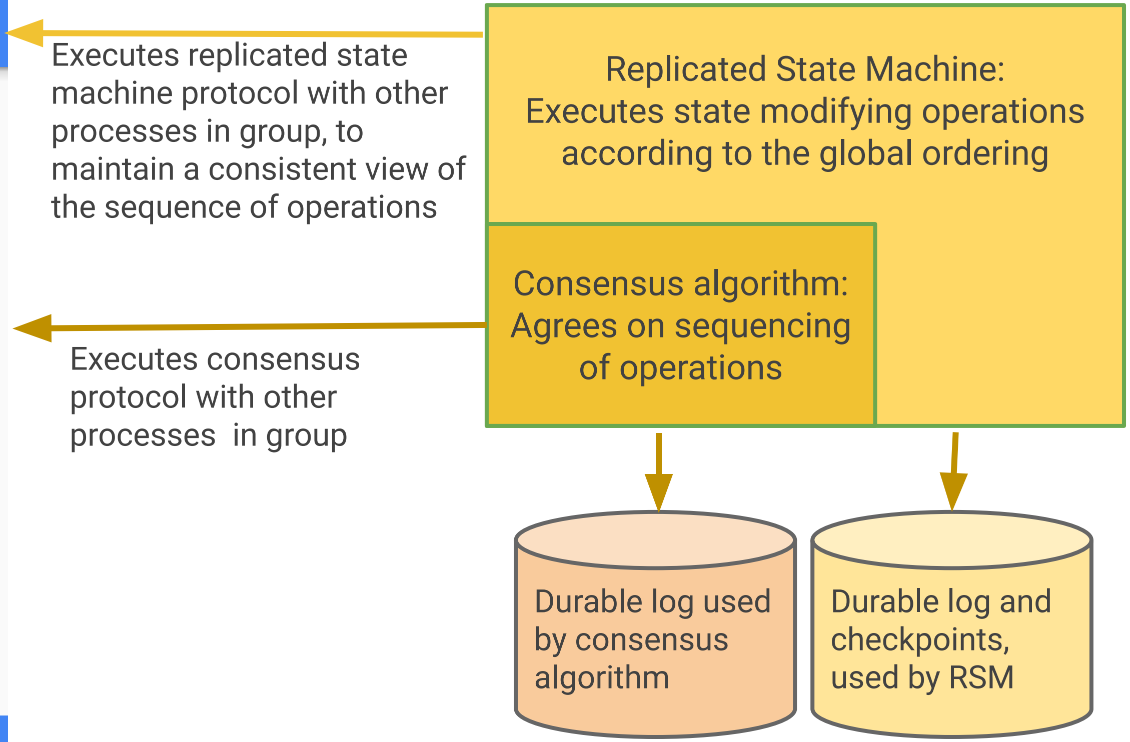
I vote to accept 82171: Add "i-8917 is master # mapreduce" to the log

I vote to accept 82171: Add "i-8917 is master # mapreduce" to the log

Accepted proposal 82171

Accepted proposal 82171

Replicated State Machine



Executes replicated state machine protocol with other processes in group, to maintain a consistent view of the sequence of operations

Replicated State Machine: Executes state modifying operations according to the global ordering

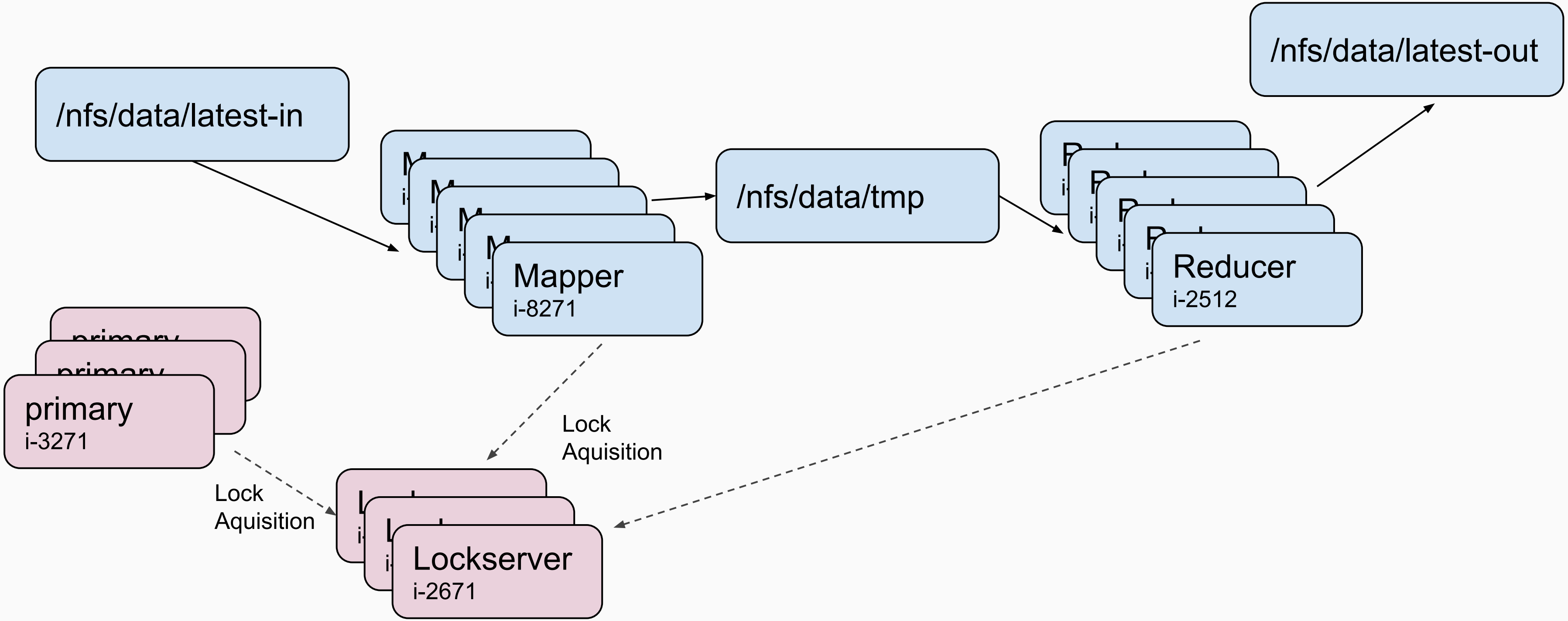
Consensus algorithm: Agrees on sequencing of operations

Executes consensus protocol with other processes in group

Durable log used by consensus algorithm

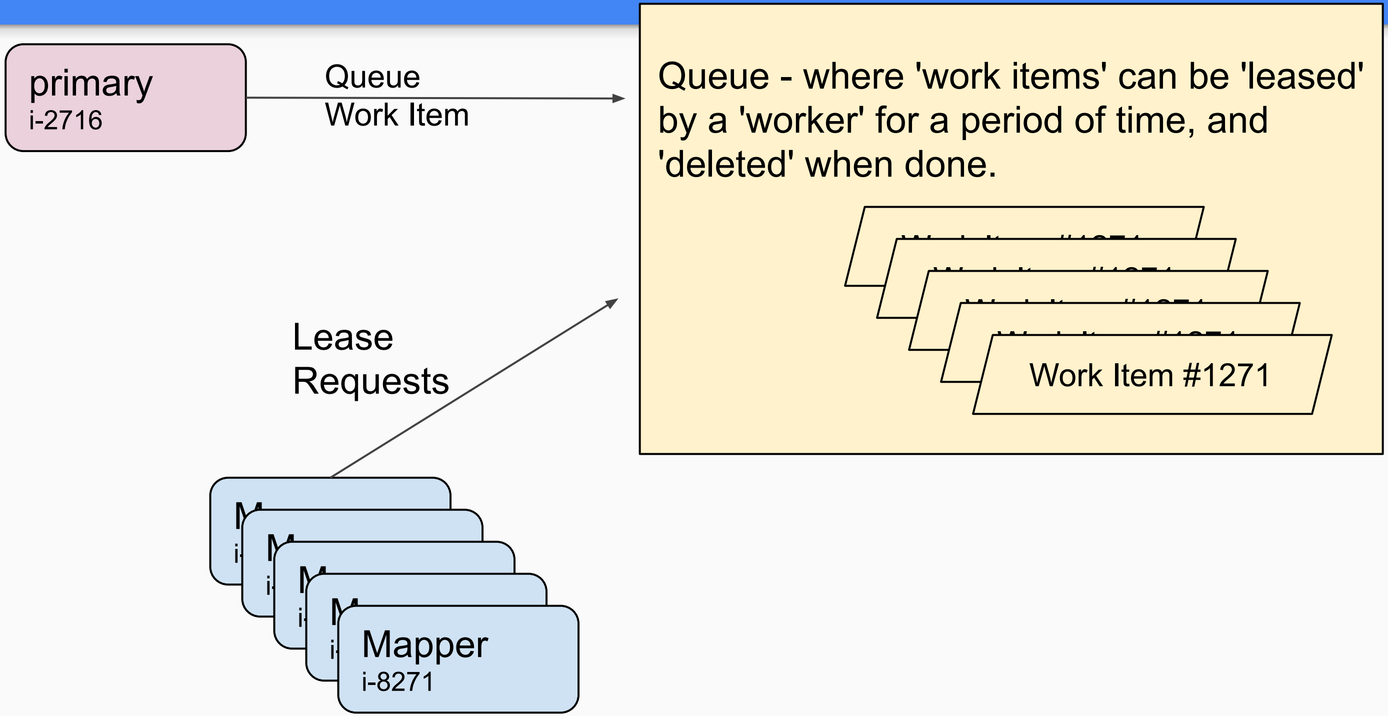
Durable log and checkpoints, used by RSM

Like a Finite State Machine, only...replicated



We need a better way to map input files to workers

Queues



Queue - where 'work items' can be 'leased' by a 'worker' for a period of time, and 'deleted' when done.

Spot the Difference!

Queue *noun*
[kyo͞o/]

Where 'work items' can be 'leased' by a 'worker' for a period of time, and 'deleted' when done.

'Queue'

'Work Item'

'Leased'

'Worker'

'Deleted'

Spot the Difference!

Lockserver *noun*
[lok **sur**-ver]

Where 'locks' can be 'locked' by a 'client' for a period of time, and 'released' when done.

'Lockserver'

'Locks'

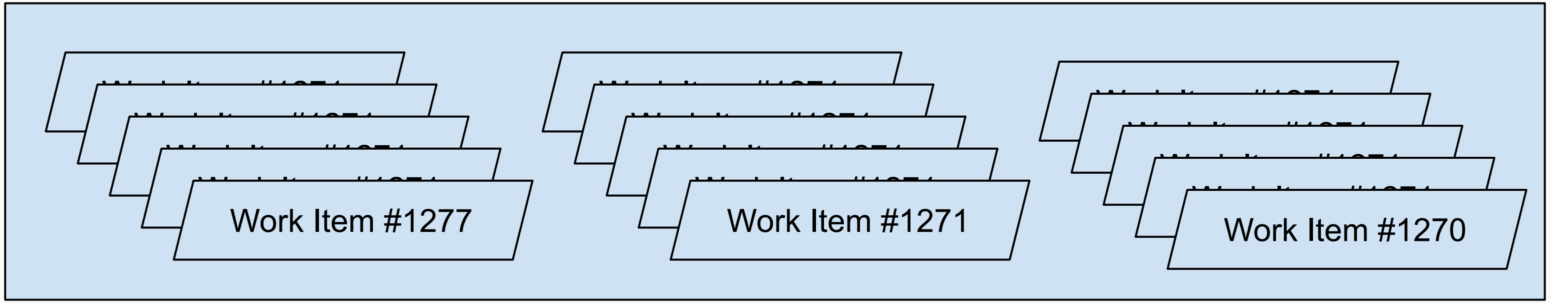
'Locked'

'Client'

'Released'

Challenges of Scaling RSMs

- Batching - not fine-grained, longer latency
- Sharding - one shard can be slower - jitter/unordering
- Pipelining - extra resource tracking, some jitter



Work Item #1277

Work Item #1271

Work Item #1270

Unordered Queues: At Most Once

Queue gives each task, to exactly one worker, exactly once

- Worker fails, task is lost.

Unordered Queues: At Least Once

Queue gives each task to a worker, requests ack before timeout

- Worker #1 times out, task is given to Worker #2
- Worker #1 succeeded eventually, but wasn't reachable for a while
- Task is processed twice

Unordered Queues: Probably Exactly Once

Queue gives each task to a worker, says don't submit after timeout

- All Workers have a synchronised clock
- Worker #1 times out, task is given to Worker #2
- Worker #1 succeeded eventually, but wasn't reachable for a while
- Worker #1 notices that it's past the timeout, so drops the task
- Task is processed twice, saved once

Unordered Queues: Someone Else's Problem

Queue gives a task to multiple consumers, tells them to work it out

- Worker #1, #2 and #3 are given a task
- Worker #2 hits up a lockserver to lock the task
- Worker #2 times out. Lockserver expires the lock.
- Worker #1 grabs the lock, does the job, commits it.
- Worker #2 comes back, realises it's lost the lock, drops the job

Ordered Queues: Pain And Suffering

- Makes no sense if you have multiple producers
- If you have multiple consumers, processing times can differ
- Ordered Queues can't be internally sharded without locking
- Properly implemented, they should have a deduplication key

Turns out, an ACID database table is best for ordered queues :(

Queues: PubSub & SQS

- Both provide AtLeastOnce semantics, maybe even ProbablyExactlyOnce
- SQS is one-queue-per-api call, PubSub 'subscribes' to multiple topics
- push/pull: SQS is pull, PubSub is both
- PubSub is like SNS/SQS/Kinesis in one
- SQS has 'FIFO' - ordering - if you want (300 qps max)
- SQS cleans up after 14 days, PubSub after 7

https://cloud.google.com/pubsub/docs/overview
https://aws.amazon.com/sqs/

Queues: Kafka, LogDevice, Kinesis

- Far more than a queue, more like a 'streaming log'
- Can be completely persistent, if you want
- Can mimic SQS or PubSub semantics
- Can also be basis for a stream-processing platform

https://code.fb.com/core-data/logdevice-a-distributed-data-store-for-logs/
https://kafka.apache.org/intro

Data Storage: CAP Theorem

- Consistency, Availability, Partition Tolerance (pick two)
    - Really 'sequential consistency' vs. 'high availability'
- We can kinda defeat PT it with Timing + Last Write Wins (see Spanner)
- We can kinda defeat Consistency with VectorClocks
- We can also defeat Availability with pre-prepared partitions

Data Storage

ACID

"All things to all people"

- **Atomicity**
  Transactions are 'all or nothing'
- **Consistency** (ugh)
  Refers to the application, not the DB
- **Isolation**
  Transactions don't step on toes
- **Durability**
  "Whatever you are having yourself"

Data Storage

BASE

"You call that a database?"

- **Basically Available**
  Mostly
- **Soft State**
  Snapshots aren't helpful
- **Eventually Consistent**
  If it doesn't make sense, just wait

Data Storage; B-Trees vs. LSM

B-Trees

- Great for many small reads
- Good for updating-in-place
- Good for fast insertions
- Great for heavy use of indexes
- Described as OLTP

Oracle, MySQL, Postgres, NTFS

Log Structure Merging

- More suitable for scanning
- Underlying storage is just logs
- Random writes -> sequential writes
- Can be setup as 'Columnar'
- Occasional 'compactions'

Bigtable, Cassandra, HBase, Lucene, MyRocksDB

Data Storage; Weak vs Strong Isolation

Weak

- No Dirty Reads
- No Dirty Writes
- Snapshot Isolation
- Atomic Writes
- Explicit Locking
- Conflict Resolution

Strong

- Literally Serial Execution
- Two-Phase Locking
  - Per-Row locks
  - Predicate Locks
  - Index-Range Locks
- Serializable Snapshot Isolation
  - MVCC visibility
  - Abort-on-tripwire
- XA Transactions

Data Storage: Data Loss

How do we lose data ?

- Disk loss
- Machine loss
- Switch loss
- Cluster loss
- Software bugs
- Security compromise
- Physics
- Chemistry

Data Storage: Data Loss

How do we lose data ?

- Disk loss
- Machine loss
- Switch loss
- Cluster loss
- Software bugs
- Security compromise
- Physics
- Chemistry

How do we avoid data loss ?

- Replication
- Replication + healthchecks
- Availability Zones
- Availability Zones
- Separate Backups
- Offsite Backups
- Background checksumming
- Scanning for correctable errors

Data Storage: Data Formats

- Columnar vs. Row
- Document vs. Cell Based
- Relational vs. NoSQL vs. Graph

Data Storage: Data Formats

- **Columnar vs. Row**
- Document vs. Cell Based
- Relational vs. NoSQL vs. Graph

Row

If gathering most of a row in every record

Finding a needle in a haystack

Column

Scanning in all of one or two columns.

Aggregations, etc.

Data Storage: Data Formats

- Columnar vs. Row
- **Document vs. Cell Based**
- Relational vs. NoSQL vs. Graph

Cell

Simple datatypes, with a fixed schema
Everyone is familiar with it from Excel to Oracle
Schema statically enforced on write

Document

Complex Datatypes, with looser schemas, like JSON, BSON, ProtocolBuffers, Avro etc.
Metadata is extracted from the Document.
Common in NoSQL, exotic in Relational DBs
Schema dynamically inferred on read

Data Storage: Database Types

- Columnar vs. Row
- Document vs. Cell Based
- **Relational vs. NoSQL vs. Graph**

Relational

Great for many-many relationships
Weak at scaling writes
The default between 1990-2015

NoSQL

Great at storing 'child records' next to a parent
Weak at pulling out single-fields
Riak, Cassandra, Bigtable, Spanner, Dynamo

Graph

Stores vertices (data) and edges (relationships)
Queried declaratively, easy to optimise queries
Neo4J, Oracle, SAP Hana

Data Storage: SQL vs. GraphQL

```
SELECT p.ProductName
FROM Product AS p
JOIN ProductCategory pc ON (p.CategoryID = pc.CategoryID AND pc.CategoryName = "Dairy Products")
JOIN ProductCategory pc1 ON (p.CategoryID = pc1.CategoryID
JOIN ProductCategory pc2 ON (pc2.ParentID = pc2.CategoryID AND pc2.CategoryName = "Dairy Products")
JOIN ProductCategory pc3 ON (p.CategoryID = pc3.CategoryID
JOIN ProductCategory pc4 ON (pc3.ParentID = pc4.CategoryID)
JOIN ProductCategory pc5 ON (pc4.ParentID = pc5.CategoryID AND pc5.CategoryName = "Dairy Products");
```

```
MATCH (p:Product)-[:CATEGORY]->(l:ProductCategory)-[:PARENT*0..]-(:ProductCategory {name:"Dairy Products"})
RETURN p.name
```

Taken from https://neo4j.com/developer/guide-sql-to-cypher/

Datacenter / Cluster Filesystems

Style one: Shared-Disk filesystems

- RedHat GFS2, IBM GPFS
- Designed for 'availability'
- Building block of 1990s style STONITH
- 'Block-level' access
  - SANs are usually block-level access

Design Review Time! (Optional)

1. Organise in Groups of 4
2. Read "Fast Recommendation Builder" Design; https://tinyurl.com/srecon-dist-2019-design1
3. Make notes/improvements to the Design
4. Argue!

Break Time!

https://goo.gl/msPjCx

Serving Systems (Part 2)

In which our heroes will discover the joy of working with...

- Cluster Filesystems
- Eventually Consistent Datastores
- Load Balancers
- Caches

Datacenter / Cluster Filesystems

Style two: Distributed Filesystems

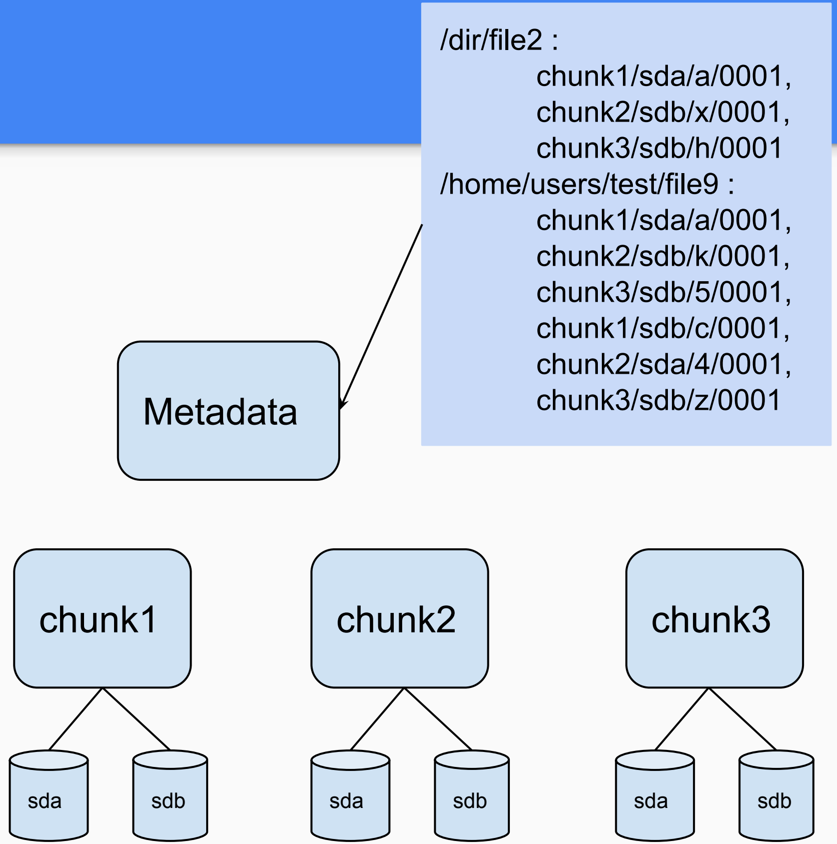
- Ceph, Hadoop HDFS RedHat Gluster, Google Colossus, Facebook WarmStorage
- Optimised for throughput
- Usually file-level access
- Features may include:
    - Load/Fault domain rebalancing, Scalability, Node-Failure Recovery

Evolution of Cluster Filesystems

Simple case (HDFS, Google File System)

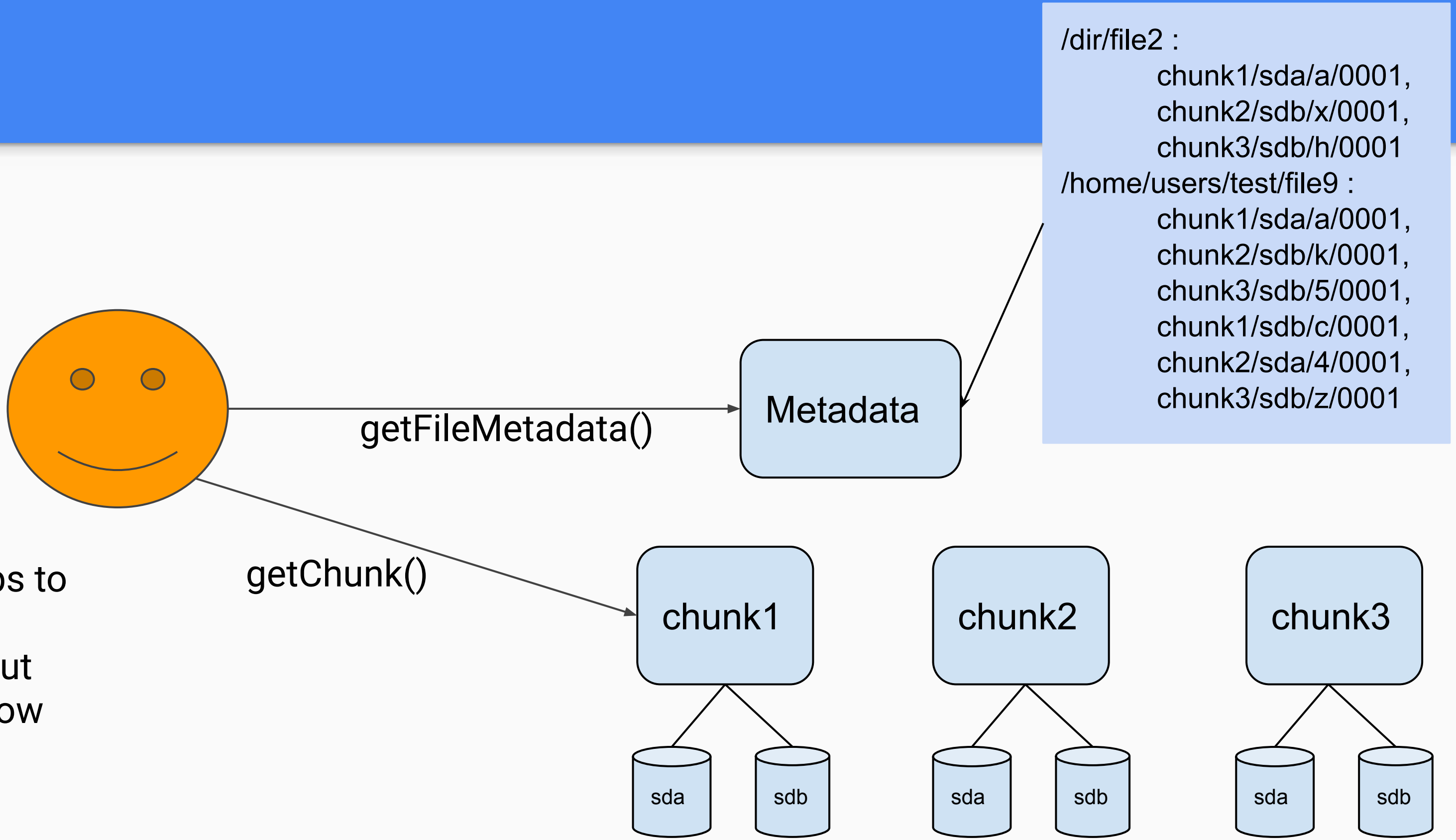
- Chunk servers store large data chunks
- Each server has multiple volumes
- Metadata server maps a filename (namespace) to a series of chunks
- Trivial to store files multiple times for 'redundancy' or read-throughput (think RAID1)

What Bottlenecks can you think of ?



```
/dir/file2 :
          chunk1/sda/a/0001,
          chunk2/sdb/x/0001,
          chunk3/sdb/h/0001
/home/users/test/file9 :
          chunk1/sda/a/0001,
          chunk2/sdb/k/0001,
          chunk3/sdb/5/0001,
          chunk1/sdb/c/0001,
          chunk2/sda/4/0001,
          chunk3/sdb/z/0001
```

Evolution of Cluster Filesystems

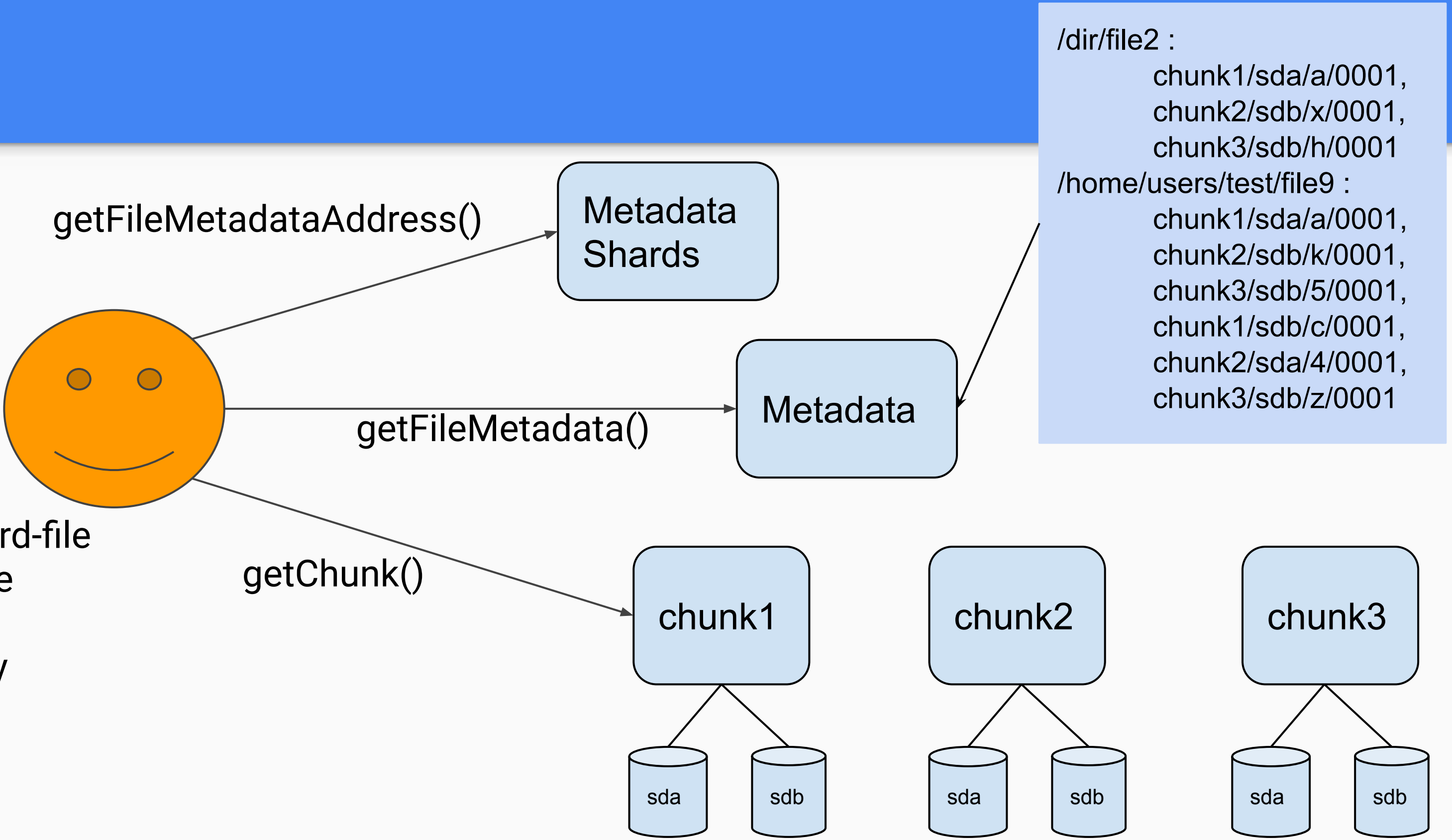


```
/dir/file2 :
          chunk1/sda/a/0001,
          chunk2/sdb/x/0001,
          chunk3/sdb/h/0001
/home/users/test/file9 :
          chunk1/sda/a/0001,
          chunk2/sdb/k/0001,
          chunk3/sdb/5/0001,
          chunk1/sdb/c/0001,
          chunk2/sda/4/0001,
          chunk3/sdb/z/0001
```

Getting Data:

- Need to do lookups to Metadata server
- Can be sharded, but clients need to know which shard to hit

Evolution of Cluster Filesystems



/dir/file2 :
chunk1/sda/a/0001,
chunk2/sdb/x/0001,
chunk3/sdb/h/0001
/home/users/test/file9 :
chunk1/sda/a/0001,
chunk2/sdb/k/0001,
chunk3/sdb/5/0001,
chunk1/sdb/c/0001,
chunk2/sda/4/0001,
chunk3/sdb/z/0001

getFileMetadataAddress()

Metadata Shards

getFileMetadata()

Metadata

getChunk()

chunk1

chunk2

chunk3

sda

sdb

sda

sdb

sda

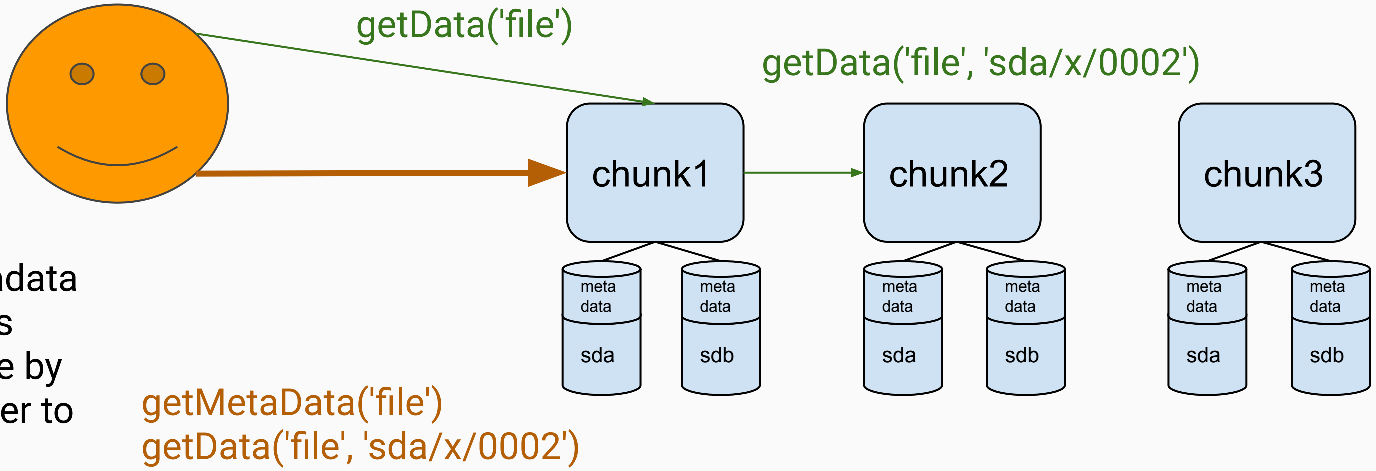
sdb

Getting Shards

- Maybe from a shard-file
- Maybe static in the client
- Maybe a discovery system

Bottleneck ideas ?

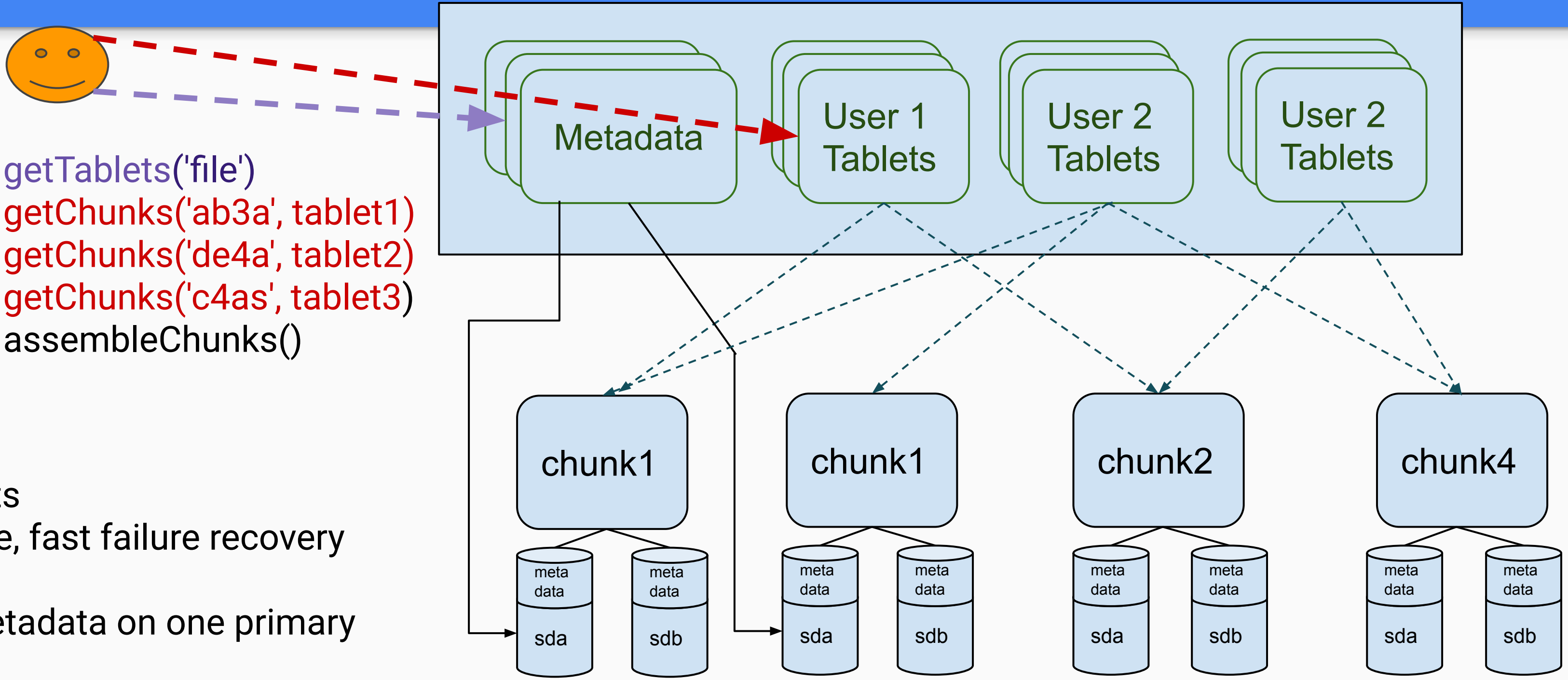
Evolution of Cluster Filesystems



Distribute Metadata Too

- Each disk has metadata for itself, and others
- Save round-trip time by allowing chunkserver to proxy data
- Reduce hotspots for even load

Evolution of Cluster Filesystems



1) getTablets('file')
2) getChunks('ab3a', tablet1)
getChunks('de4a', tablet2)
getChunks('c4as', tablet3)
3) assembleChunks()

Small tablets

- nimble, fast failure recovery

Big tablets

- All metadata on one primary

Useful Distributed DB/Cluster patterns

Allow latency sensitive systems to query multiple shares concurrently, and choose the winner

- Good: Send RPCs to all three replicas all the time
- Great: Send RPCs to two replicas when latency goes over 30ms and load is under 80%

Useful Distributed DB/Cluster patterns

Replicate 'hot' data multiple times

- Good: notice file 'xayzz' is accessed a lot, replication goes 3->12
- Great: notice that at files in /data/europe are accessed frequently between 09:00 and 12:00 UTC.
  Schedule a replication job at 08:30 and prune them at 12:30.

Useful Distributed DB/Cluster patterns

Partition disparate workloads

- A single filesystem be low-latency, high-throughput at massive scale ?
- Pin tablets to 'low-latency' machines or 'high-throughput' machines with Quotas

Eventually Consistent Datastores

What's the problem ?

- We can't tell when a node will come back
- We can't tell when a netsplit will end
- We can't tell if a node got a message or not
- We are in a hurry, and can't wait all day for confirmation

How do we get 'consistent' ?

- **Statement based replication**
- Write-Ahead-Log replication
- Logical Log Replication

How do we get 'consistent' ?

- Statement based replication
- **Write-Ahead-Log replication**
- Logical Log Replication

How do we get 'consistent' ?

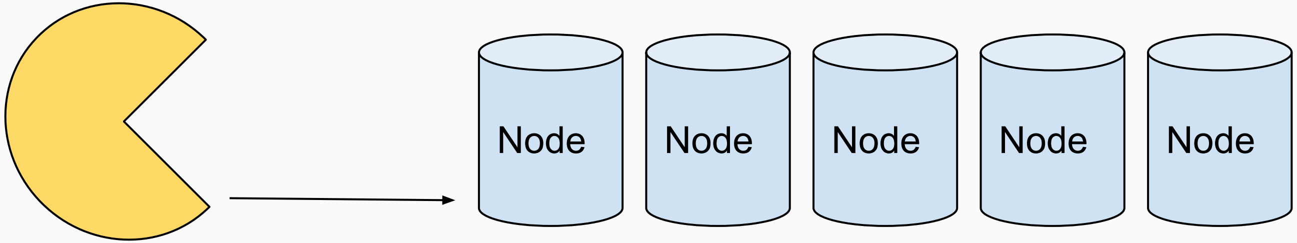
- Statement based replication
- Write-Ahead-Log replication
- **Logical Log Replication**

Cassandra & Tunable Consistency

- Choose how many nodes must take writes
- Choose how many nodes must ack writes

Cassandra & Tunable Consistency

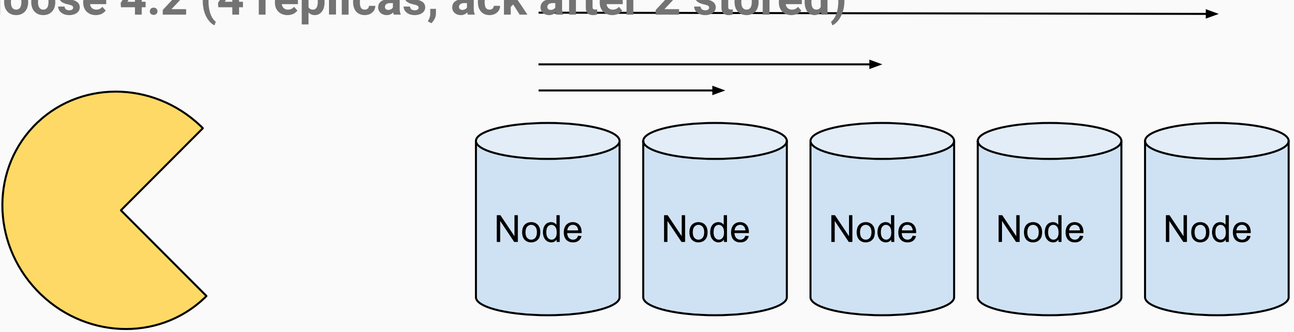
- Choose how many nodes must take writes
- Choose how many nodes must ack writes
- **Let's choose 4:2 (4 replicas, ack after 2 stored)**



T=0 - Client sends data to a cassandra node

Cassandra & Tunable Consistency

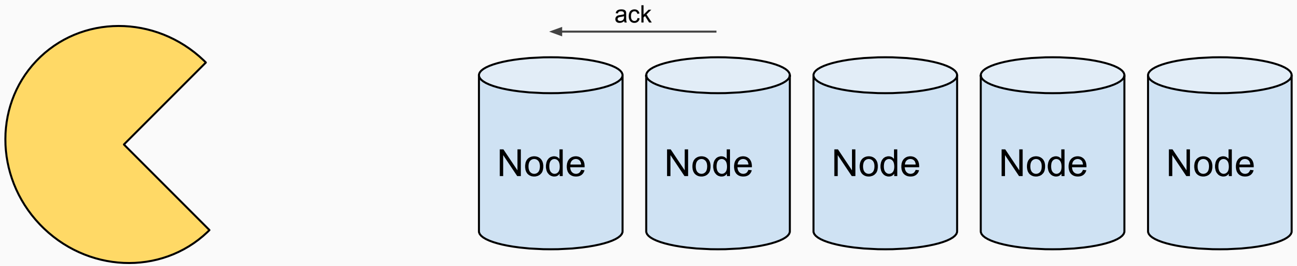
- Choose how many nodes must take writes
- Choose how many nodes must ack writes
- **Let's choose 4:2 (4 replicas, ack after 2 stored)**



t=1 Node sends data to other nodes

Cassandra & Tunable Consistency

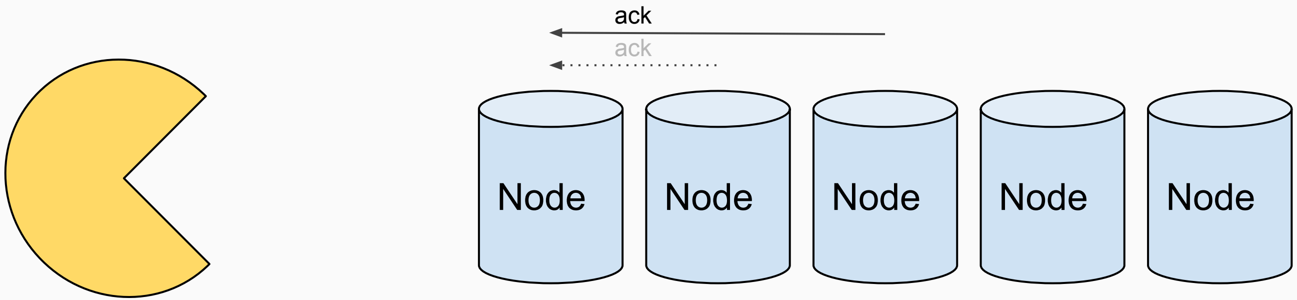
- Choose how many nodes must take writes
- Choose how many nodes must ack writes
- **Let's choose 4:2 (4 replicas, ack after 2 stored)**



t=3 1 node responds with 'ack'

Cassandra & Tunable Consistency

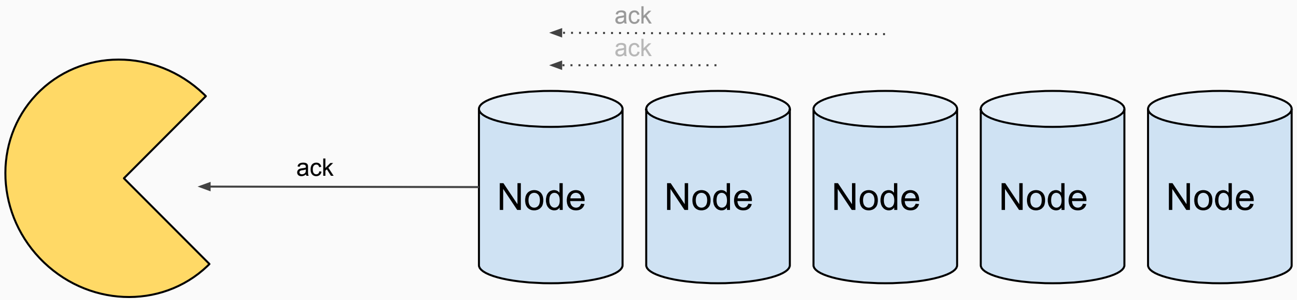
- Choose how many nodes must take writes
- Choose how many nodes must ack writes
- **Let's choose 4:2 (4 replicas, ack after 2 stored)**



t=4 a second node responds with 'ack'

Cassandra & Tunable Consistency

- Choose how many nodes must take writes
- Choose how many nodes must ack writes
- **Let's choose 4:2 (4 replicas, ack after 2 stored)**

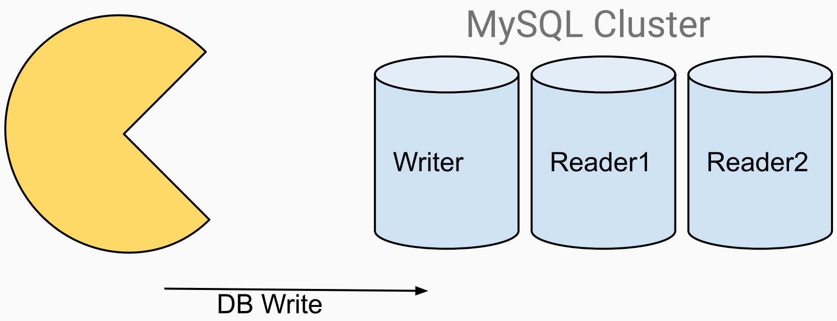


t=5 the client-facing node responds with 'ack', without waiting for other two nodes to ack.

Consistency problem #1: Replication Lag

1 Writer + X Readers

- Writer sends Binlogs to Readers
- Readers mutate their database
- Replication lag is ~5ms

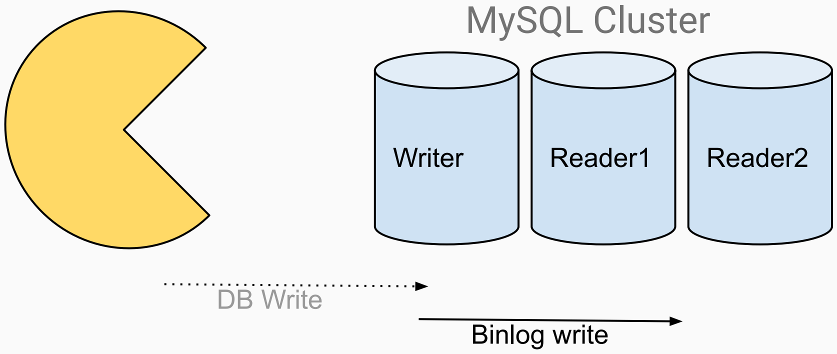


t=0ms
Client sends data to the Writer

Consistency problem #1: Replication Lag

1 Writer + X Readers

- Writer sends Binlogs to Readers
- Readers mutate their database
- Replication lag is ~5ms



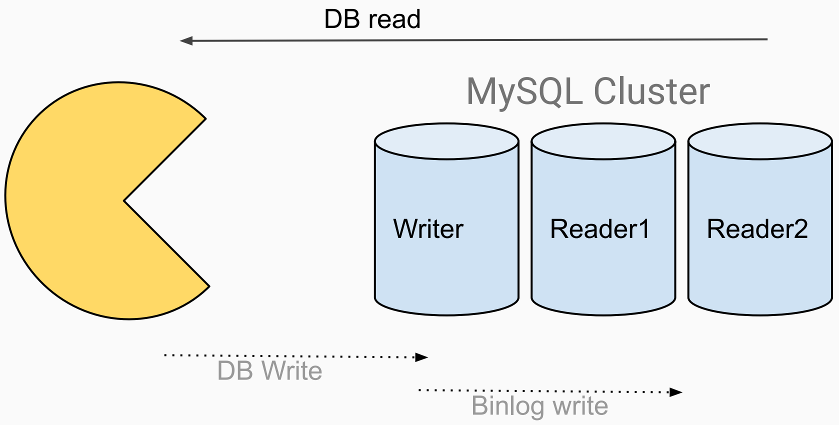
t=5ms
Writer's binlog makes it to Reader1

Consistency problem #1:

Replication Lag

1 Writer + X Readers

- Writer sends Binlogs to Readers
- Readers mutate their database
- Replication lag is ~5ms
- **Client reads old data, joined with other data**
    - ○ **Reads from Reader2**



t=6ms
Client reads back from Reader2

Consistency problem #2: Causality Violations

- Comments and Posts are stored on different partitions in a database
- A Post is created. Someone comments on the post.
- The comments are replicated to all shards of the partition
- One shard of the Post DB was slow
- A user read their list of comments, and the app threw a 500 because it couldn't join the comment with the missing post.

Consistency problem #2:
Causality Violations

- Comments and Posts are stored on different partitions in a database
- A Post is created. Someone comments on the post.
- The comments are replicated to all shards of the partition
- One shard of the Post DB was slow
- A user read their list of comments, and the app threw a 500 because it couldn't join the comment with the missing post.

Solution: Keep comments to a post in the same partition

Consistency problem #3:

Global split-brain

- We need data living in multiple continents
- We get regular net-splits
- During net-splits, we continue to accept writes
- After net-splits, try work out what the database should be

Consistency problem #3:

Global split-brain

Netsplit happens

1. A moderator in the US marks a post as 'unacceptable' with a reason
2. A moderator in the EU marks a post as 'illegal' with a reason
3. The EU appserver sets the 'last updated by' as the EU moderator
4. The US appserver sets the 'last updated by' as the US moderator

Netsplit finishes

What should we do with the post & 'last updated by' ?

Consistency problem #3:
Global split-brain

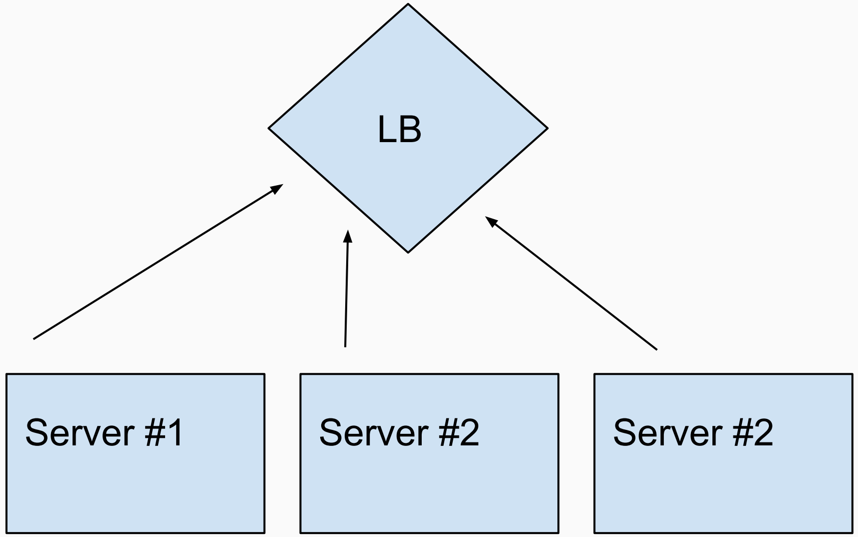
Some options...

- Last Write Wins
  - Variants like 'based on userID, not date' or 'based on webserver IP'
- Notify both Admins of the conflict, and hold changes
- Force writes through one writer
- Partition by post ID, with forced-writer
- Transactions
- Dedicated "conflict handler"
  - On read, or on write
- Operational Transformations instead of 'updates'

Handling Scaling; Sharding & Partitioning

- Share data, and the load it attracts over more nodes
- Reduce hotspots where possible
- Round-Robin inbound items of data is naïve
- More partitions (shards) == more fanout
- More replicas == more bandwidth & reliability

Load Balancing; What's The Point ?

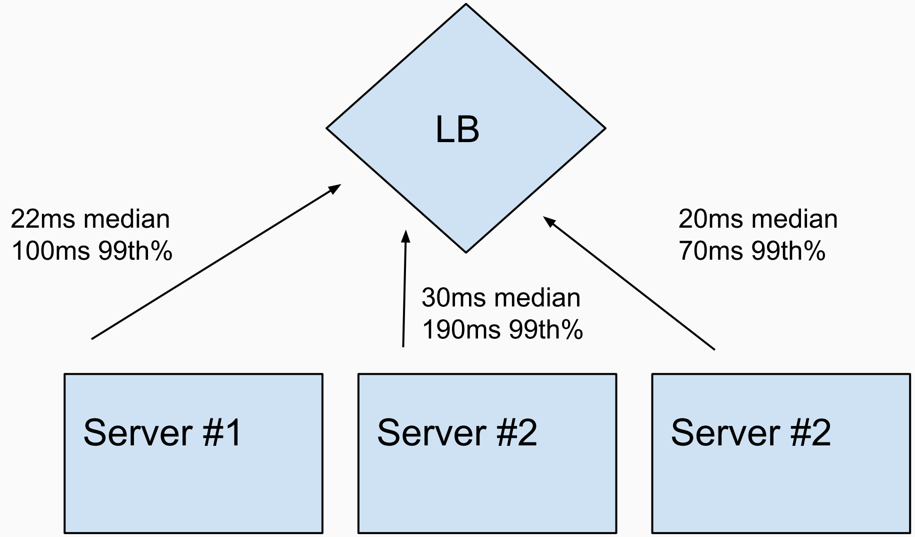


Spread the load, evenly.

Make good use of all nodes.

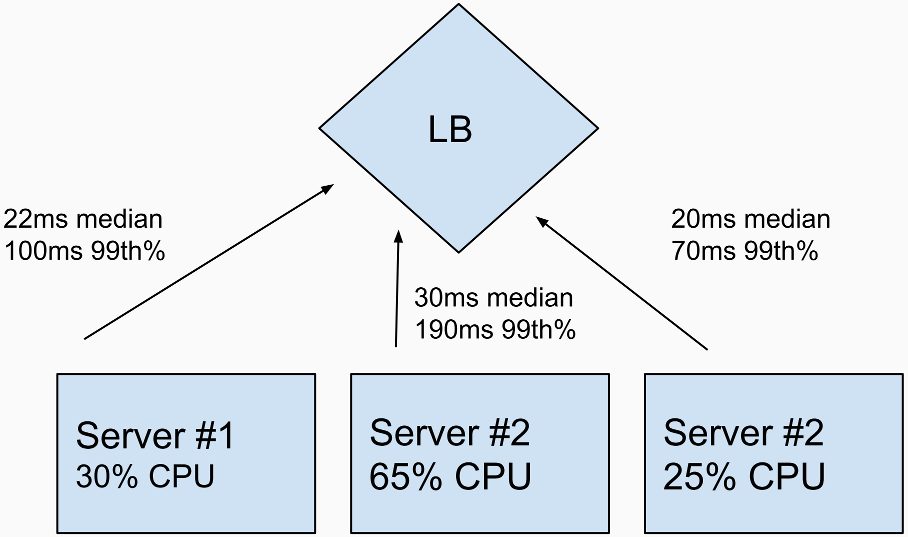
Spot broken nodes.

Load Balancing; Which node ?



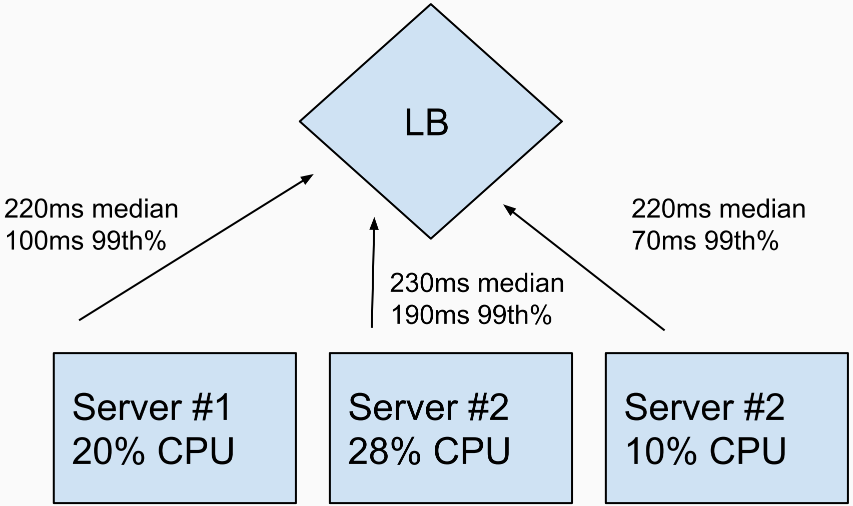
How do we choose the next destination ?

Load Balancing; Spreading Load



Why do servers respond differently to requests ?

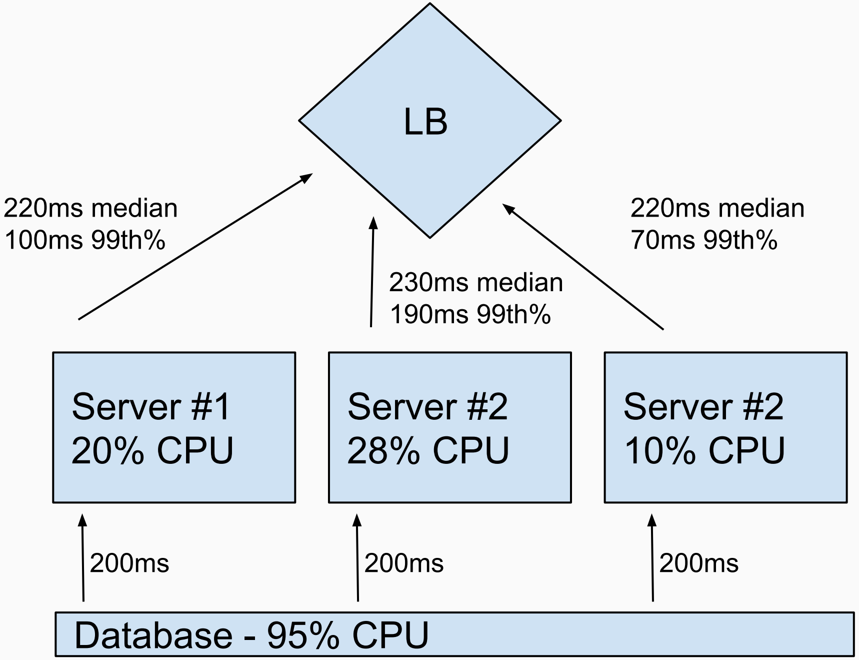
Load Balancing; What's The Point ?



What changed ?

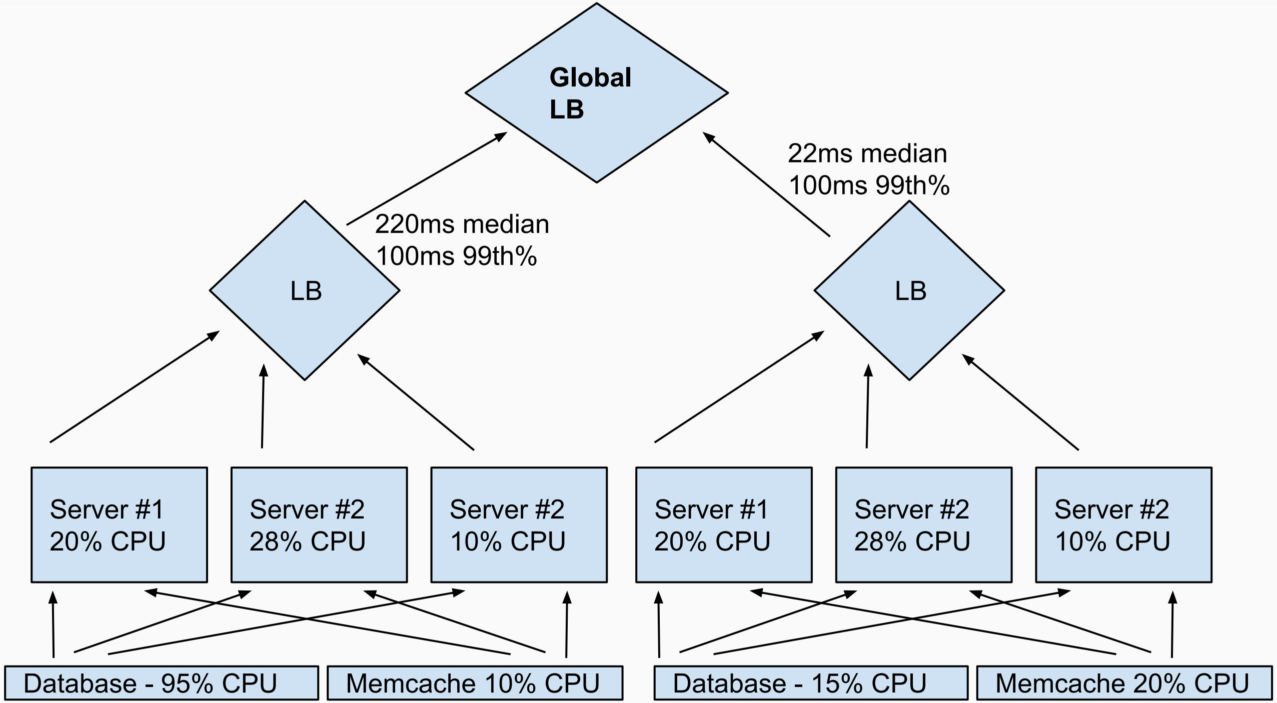
Seems CPU was such a good proxy here...

Load Balancing; Troubleshooting Time!



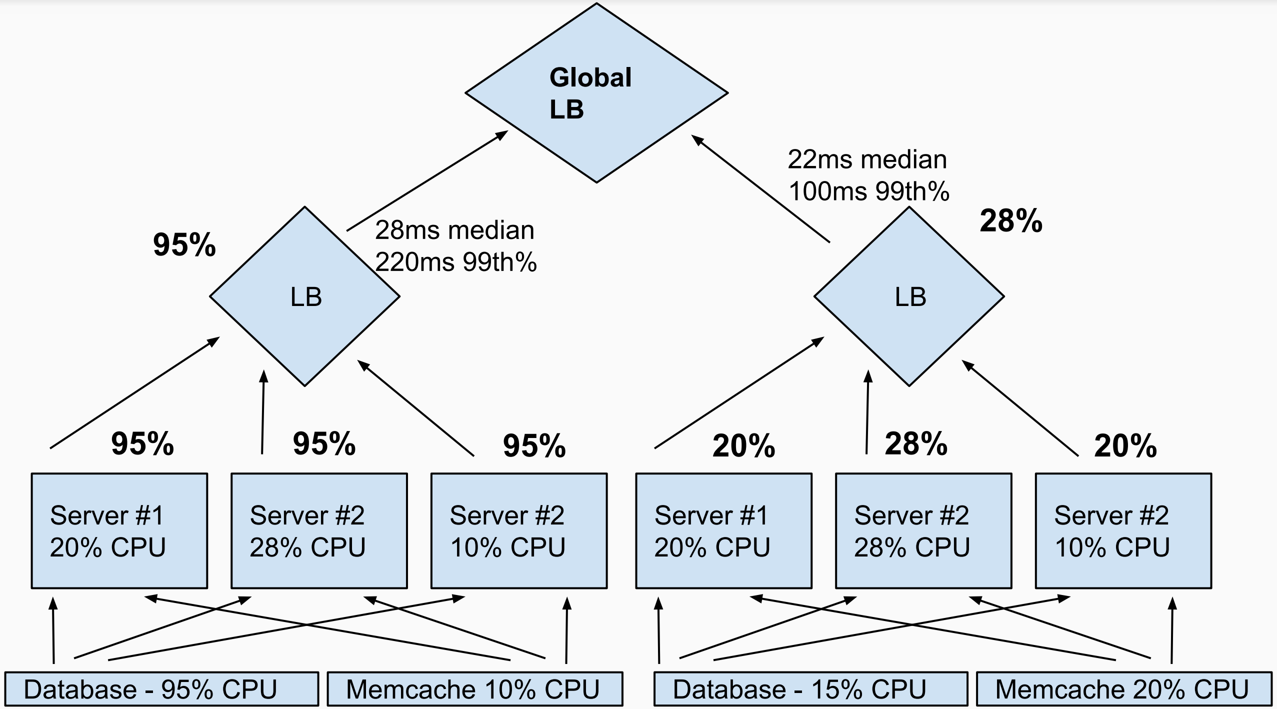
Ah. Slow database. What are you going to do ?

Load Balancing; Going Global



Ah. Slow database. What are you going to do ?

Load Balancing; Going Global



Global load balancers can't just go on response times to or CPU of the last node in the chain

A backend could report the max of many metrics, or any of it's children's metrics.

Load Balancing; Common Failure Modes

- Thundering Herd & Lukewarm Caches
- Death Ray of Doom
- Dirty Deeds, Done Dirt Cheap
- Deep Healthchecking

Load Balancing; Common Software

- AWS ELB (L3)
  - Dumb packet switcher, HTTP1.x only
- Front-End Proxies
  - Nginx
  - Apache etc.
- Full L4 balancers:
  - Good for routing URLs around
  - Maybe some protocol-specific magic

Load Balancing; Layer-4 balancers

AWS ALB (L4)

- More even connection balancing than ELB
- Can route to ECS services as well as ip:ports
- Very basic control over balancing choices

HAProxy

- Good variable/state exporting
- User Space & rock solid

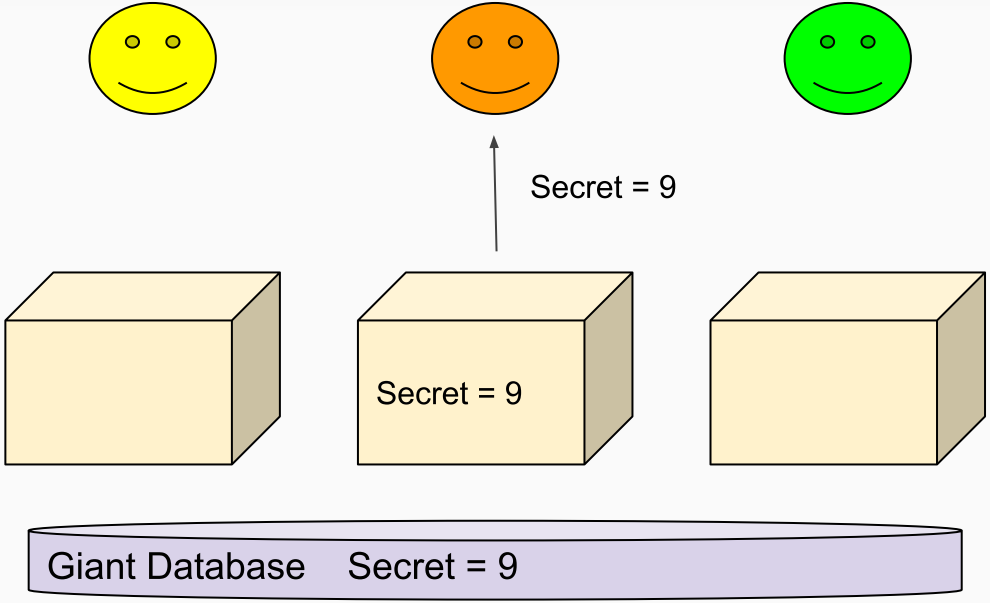
IPVS

- Linux Kernel-Space load balancing
- Simple, high-throughput forwarding
- No SSL termination etc.
- Supports VS-DR (Direct Routing)
- Supports UDP & VRRP

Caches; Overview

- Trade-off a storage resource for cpu, network or memory saving
- Usually at every layer of the stack
    - Caches compound
- The choice of eviction algorithm dictates how they behave under-stress
    - First In, First Out
    - Last In, First Out
    - Least Recently Used
    - Time-Aware
    - Least-Frequent, Recently Used

Caches; Distributed Coherence

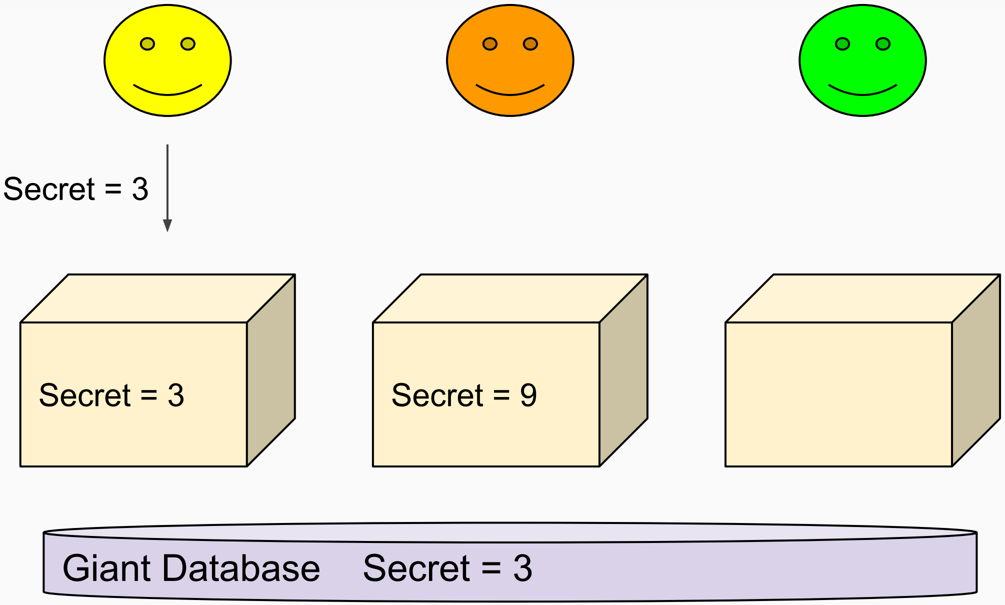


Secret = 9

Secret = 9

Giant Database Secret = 9

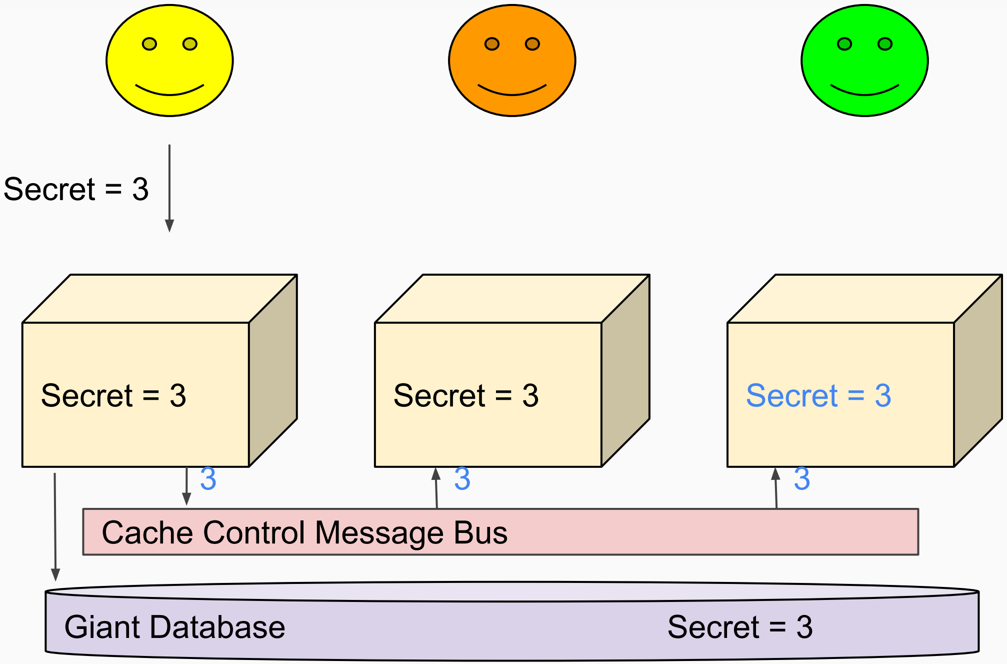
Caches; Distributed Coherence



When Yellow or Green will get back a different answer for the same value!

Critical if you are doing transactions where one item depends on the previous one!

Cache Snooping



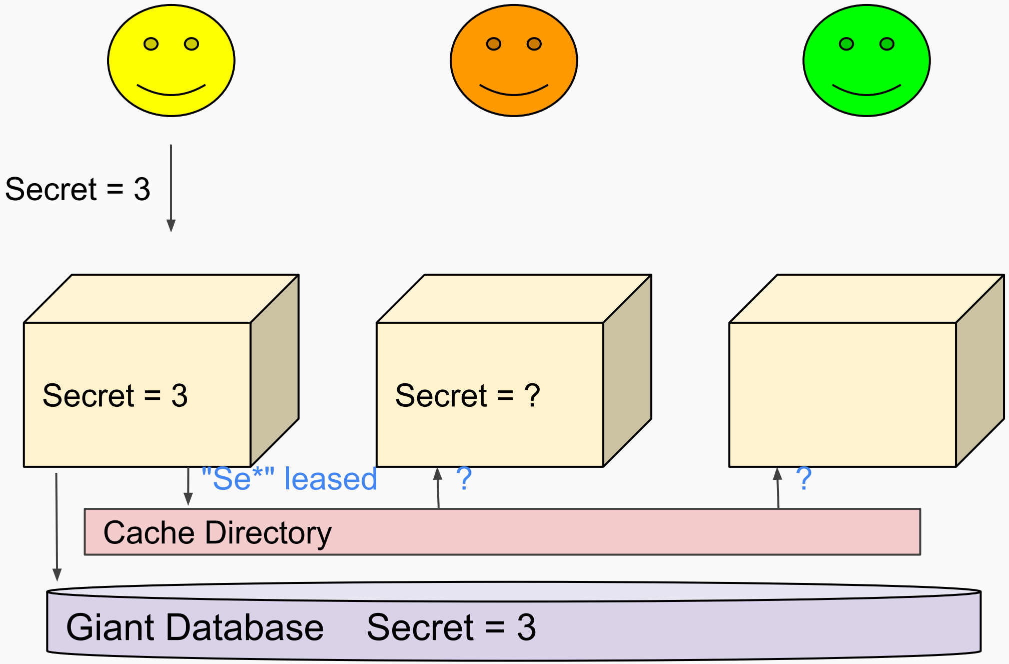
A message queue that all caches read from, to get updated important values can be useful.

If it's not already cached, it might not be set from the queue, as Green's cache has done. Choose from:

- Write Invalidate
- Write Update (as seen here)

Scalability depends on frequency of writes. Partitioning is key.

Cache Directories



A directory of cache leases is kept

Caches that want to write to the cache get a 'lease' on a subset of the dataset. Only they can write to the dataset.

Always 'Write-Invalidate'

Cache Capacity Planning

How do you choose a cache size ?

- Single-level caches are easy
  - load test them, decide on cost of cache vs scaled service
- Multi-level caches are sums of multiple curves
  - each layer load-tested
- It's never acceptable to guess, unless the cache doesn't matter
- Test your cold-caches!
  - Ensure you load-shed until they warm up

Design Review Time! (Optional)

1. Organise in Groups of 4
2. Make a copy of the "Fast Recommendation Service" design doc at https://tinyurl.com/srecon-dist-2019-design2
3. Make notes/improvements to the Design
4. Argue!