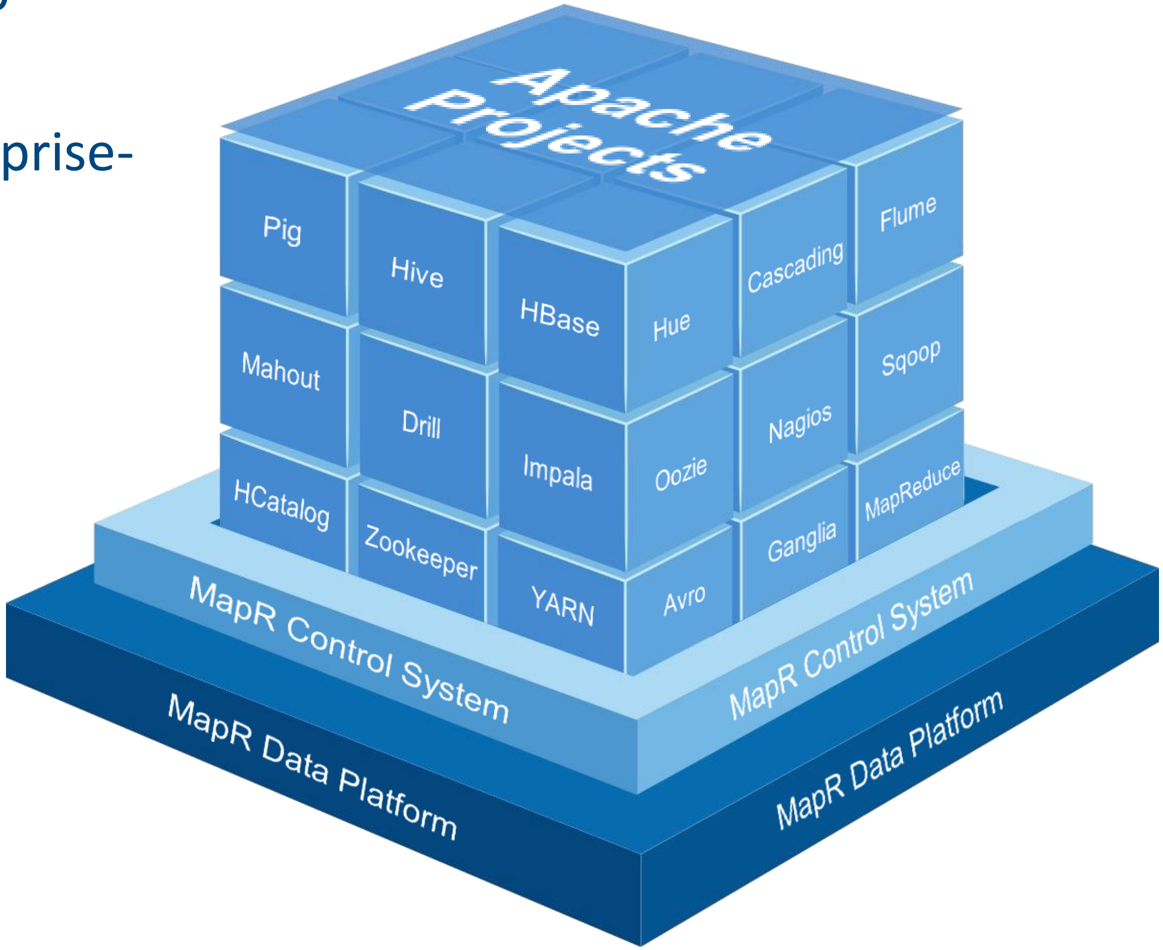
An Architectural Overview of MapR

M. C. Srivas, CTO/Founder

MapR Distribution for Apache Hadoop

- 100% Apache Hadoop
- With significant enterprise-grade enhancements
- Comprehensive management
- Industry-standard interfaces
- Higher performance



MapR: Lights Out Data Center Ready

Reliable Compute

Dependable Storage

- Automated stateful failover
- Automated re-replication
- Self-healing from HW and SW failures
- Load balancing
- Rolling upgrades
- No lost jobs or data
- 99999's of uptime

- Business continuity with snapshots and mirrors
- Recover to a point in time
- End-to-end check summing
- Strong consistency
- Built-in compression
- Mirror between two sites by RTO policy

MapR does MapReduce (fast)

TeraSort Record

1 TB in 54 seconds

1003 nodes

MinuteSort Record

1.5 TB in 59 seconds

2103 nodes

MapR does MapReduce (faster)

MAPR TECHNOLOGIES Google

TeraSort Record

1 TB in 54 seconds

1003 nodes

MinuteSort Record

1.65 ~~1.5~~ TB in 59 seconds

~~2103~~ nodes

300

The Cloud Leaders Pick MapR

Amazon EMR is the largest Hadoop provider in revenue and # of clusters

Google chose MapR to provide Hadoop on Google Compute Engine

Deploying OpenStack? MapR partnership with Canonical and Mirantis on OpenStack support.

How to make a cluster reliable?

1. Make the storage reliable
   - Recover from disk and node failures

How to make a cluster reliable?

1. Make the storage reliable
   - Recover from disk and node failures
2. Make services reliable
   - Services need to checkpoint their state rapidly
   - Restart failed service, possibly on another node
   - Move check-pointed state to restarted service, using (1) above

How to make a cluster reliable?

1. Make the storage reliable
   - Recover from disk and node failures
2. Make services reliable
   - Services need to checkpoint their state rapidly
   - Restart failed service, possibly on another node
   - Move check-pointed state to restarted service, using (1) above
3. Do it fast
   - Instant-on … (1) and (2) must happen very, very fast
   - Without maintenance windows
     - No compactions (eg, Cassandra, Apache HBase)
     - No "anti-entropy" that periodically wipes out the cluster (eg, Cassandra)

Reliability With Commodity Hardware

- No NVRAM

Reliability With Commodity Hardware

- No NVRAM

- Cannot assume special connectivity
  - no separate data paths for "online" vs. replica traffic

Reliability With Commodity Hardware

- No NVRAM

- Cannot assume special connectivity
  - no separate data paths for "online" vs. replica traffic

- Cannot even assume more than 1 drive per node
  - no RAID possible

Reliability With Commodity Hardware

- No NVRAM

- Cannot assume special connectivity
  - no separate data paths for "online" vs. replica traffic

- Cannot even assume more than 1 drive per node
  - no RAID possible

- Use replication, but …
  - cannot assume peers have equal drive sizes
  - drive on first machine is 10x larger than drive on other?

Reliability With Commodity Hardware

- No NVRAM

- Cannot assume special connectivity
  - no separate data paths for "online" vs. replica traffic

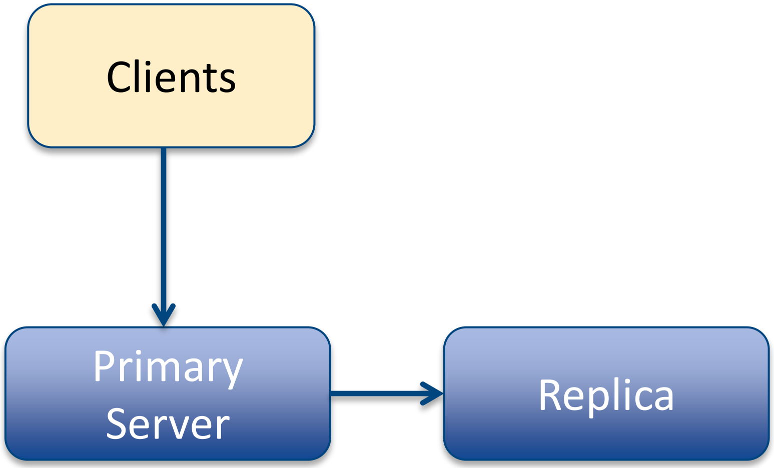
- Cannot even assume more than 1 drive per node
  - no RAID possible

- Use replication, but …
  - cannot assume peers have equal drive sizes
  - drive on first machine is 10x larger than drive on other?

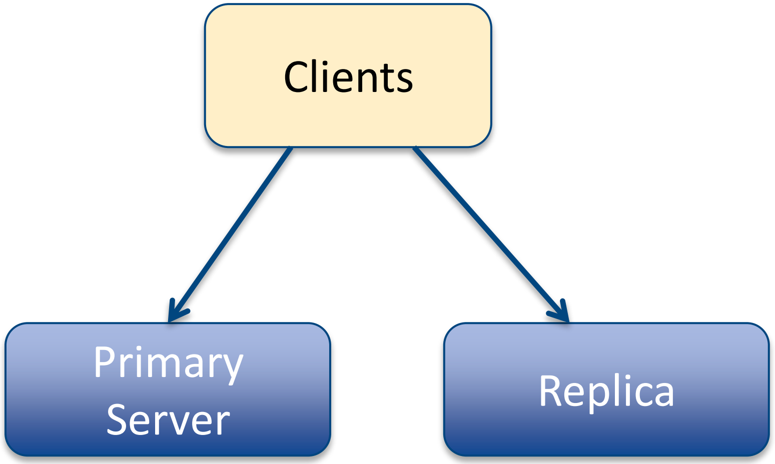
- No choice but to replicate for reliability

Reliability via Replication

- Replication is easy, right? All we have to do is send the same bits to the master and replica.



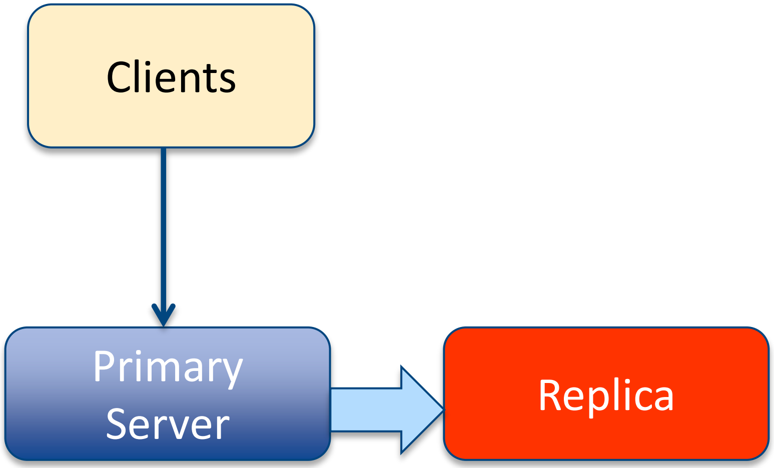
Normal replication, primary forwards



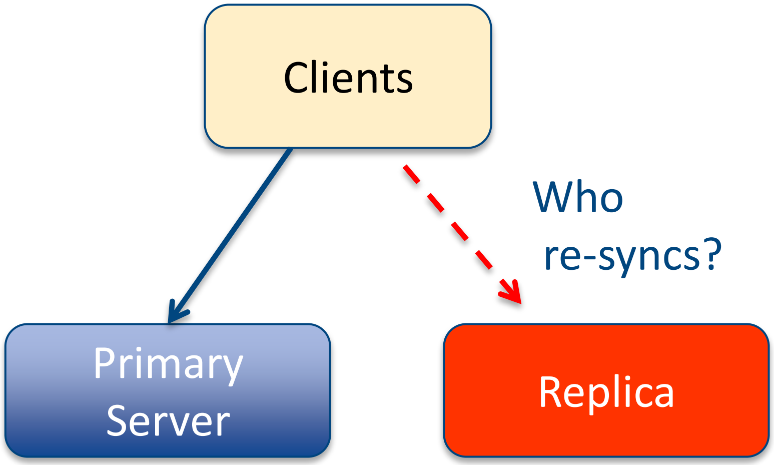
Cassandra-style replication

But crashes occur…

- When the replica comes back, it is stale
  - it must brought up-to-date
  - until then, exposed to failure



Primary re-syncs replica



Replica remains stale until "anti-entropy" process kicked off by administrator

Unless its Apache HDFS …

- HDFS solves the problem a third way

Unless its Apache HDFS …

- HDFS solves the problem a third way
- **Make everything read-only**
  - Nothing to re-sync

Unless its Apache HDFS …

- HDFS solves the problem a third way
- **Make everything read-only**
  - Nothing to re-sync
- Single writer, no reads allowed while writing

Unless its Apache HDFS …

- HDFS solves the problem a third way
- **Make everything read-only**
  - Nothing to re-sync
- Single writer, no reads allowed while writing
- File close is the transaction that allows readers to see data
  - unclosed files are lost
  - cannot write any further to closed file

Unless its Apache HDFS …

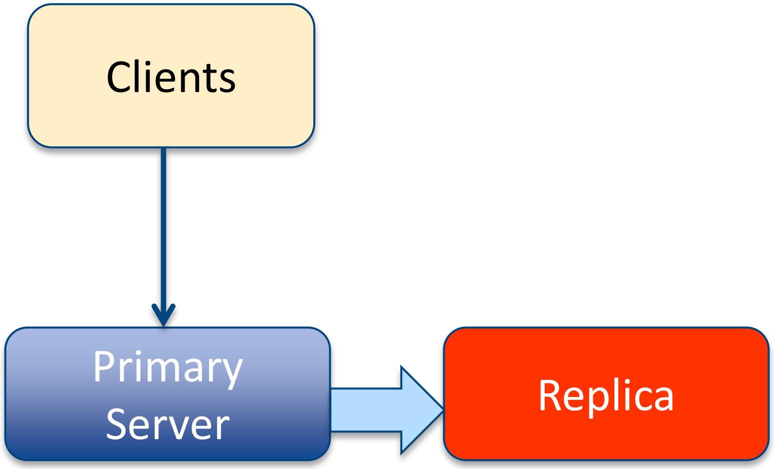
- HDFS solves the problem a third way
- **Make everything read-only**
  - Nothing to re-sync
- Single writer, no reads allowed while writing
- File close is the transaction that allows readers to see data
  - unclosed files are lost
  - cannot write any further to closed file
- Real-time not possible with HDFS
  - to make data visible, must close file immediately after writing
  - Too many files is a serious problem with HDFS (a well documented limitation)

Unless its Apache HDFS …

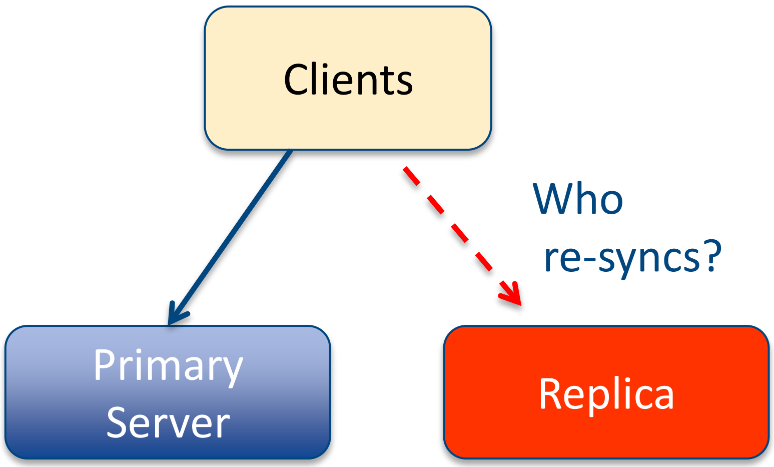
- HDFS solves the problem a third way
- **Make everything read-only**
  - Nothing to re-sync
- Single writer, no reads allowed while writing
- File close is the transaction that allows readers to see data
  - unclosed files are lost
  - cannot write any further to closed file
- Real-time not possible with HDFS
  - to make data visible, must close file immediately after writing
  - Too many files is a serious problem with HDFS (a well documented limitation)
- HDFS therefore cannot do NFS, ever
  - No “close” in NFS … can lose data any time

This is the 21st century…

- To support normal apps, need full read/write support
- Let's return to issue: resync the replica when it comes back



Primary re-syncs replica



Replica remains stale until "anti-entropy" process kicked off by administrator

How long to re-sync?

- 24 TB / server
  - @ 1000MB/s = 7 hours
  - practical terms, @ 200MB/s = 35 hours

How long to re-sync?

- 24 TB / server
  - @ 1000MB/s = 7 hours
  - practical terms, @ 200MB/s = 35 hours

- Did you say you want to do this online?

How long to re-sync?

- 24 TB / server
  - @ 1000MB/s = 7 hours
  - practical terms, @ 200MB/s = 35 hours

- Did you say you want to do this online?
  - throttle re-sync rate to 1/10th
  - 350 hours to re-sync (= 15 days)

How long to re-sync?

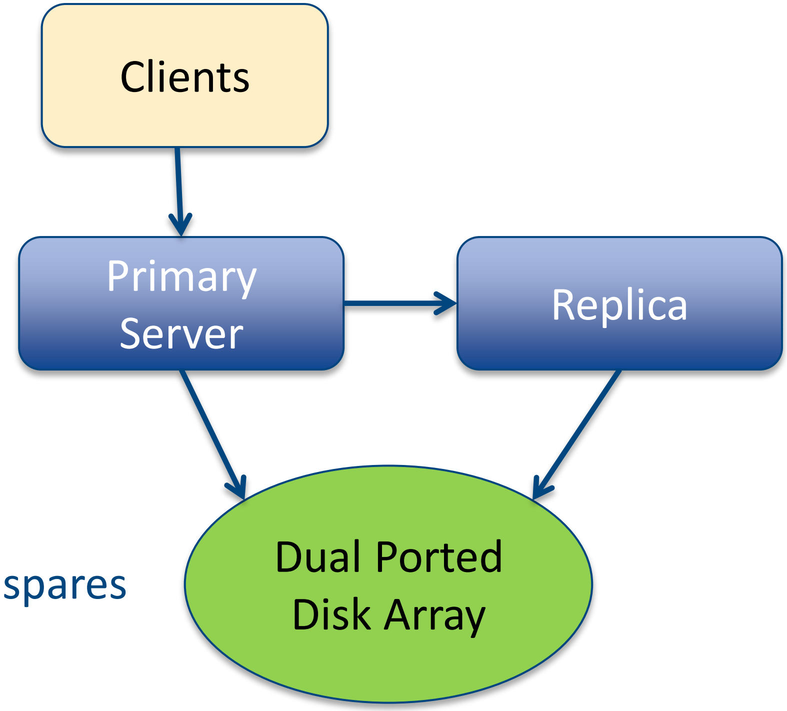
- 24 TB / server
  - @ 1000MB/s = 7 hours
  - practical terms, @ 200MB/s = 35 hours
- Did you say you want to do this online?
  - throttle re-sync rate to 1/10th
  - 350 hours to re-sync (= 15 days)
- What is your Mean Time To Data Loss (MTTDL)?

How long to re-sync?

- 24 TB / server
  - @ 1000MB/s = 7 hours
  - practical terms, @ 200MB/s = 35 hours
- Did you say you want to do this online?
  - throttle re-sync rate to 1/10th
  - 350 hours to re-sync (= 15 days)
- What is your Mean Time To Data Loss (MTTDL)?
  - how long before a double disk failure?
  - a triple disk failure?

Traditional solutions

- Use dual-ported disk to side-step this problem

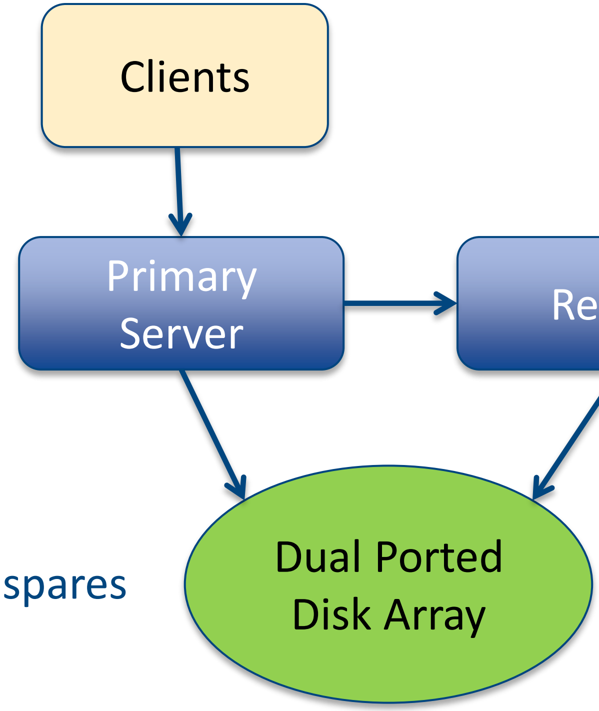


Servers use NVRAM

Raid-6 with idle spares

Traditional solutions

- Use dual-ported disk to side-step this problem



Servers use NVRAM

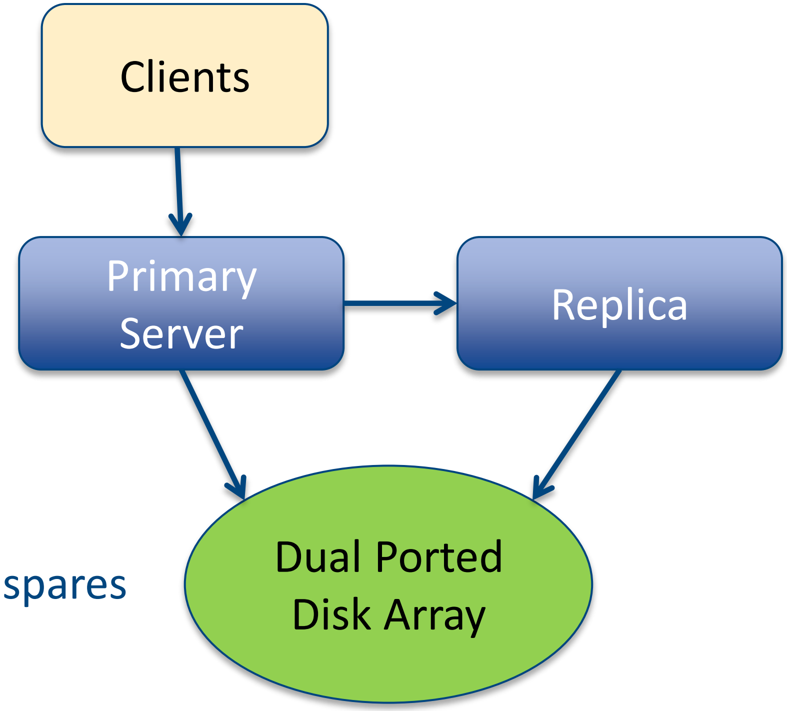
Raid-6 with idle spares

COMMODITY HARDWARE

LARGE SCALE CLUSTERING

Traditional solutions

- Use dual-ported disk to side-step this problem



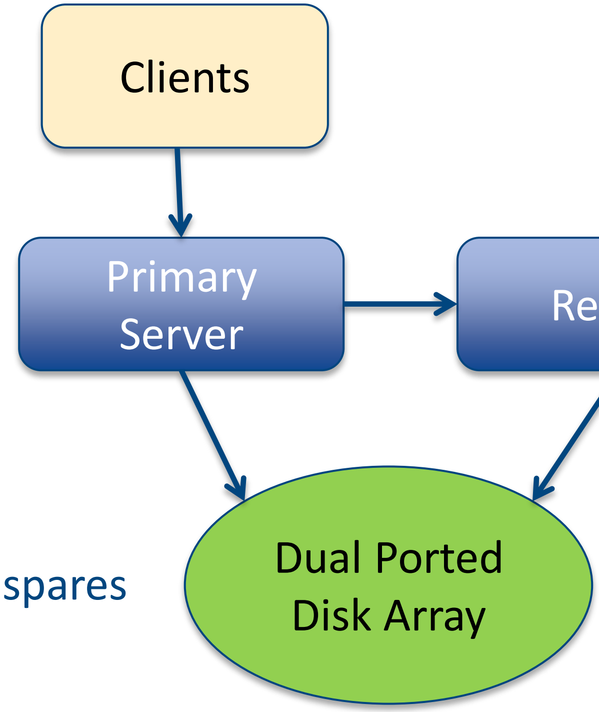
Servers use NVRAM

Raid-6 with idle spares

~~COMMODITY HARDWARE~~ ~~LARGE SCALE CLUSTERING~~

Traditional solutions

- Use dual-ported disk to side-step this problem



Servers use NVRAM

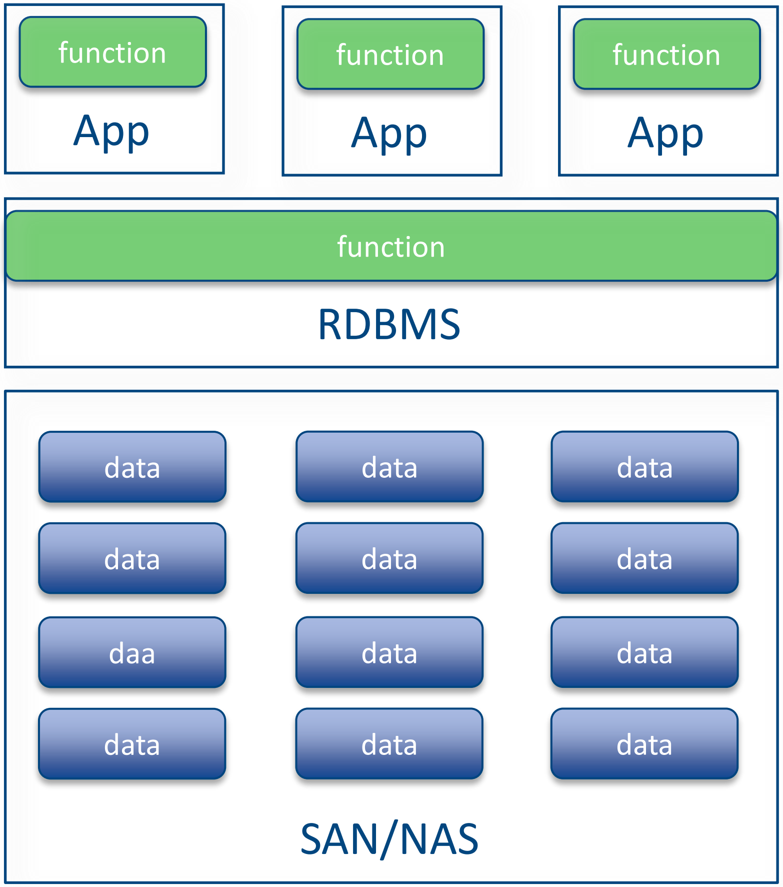
Raid-6 with idle spares

~~COMMODITY HARDWARE~~ ~~LARGE SCALE CLUSTERING~~

Large Purchase Contracts, 5-year spare-parts plan

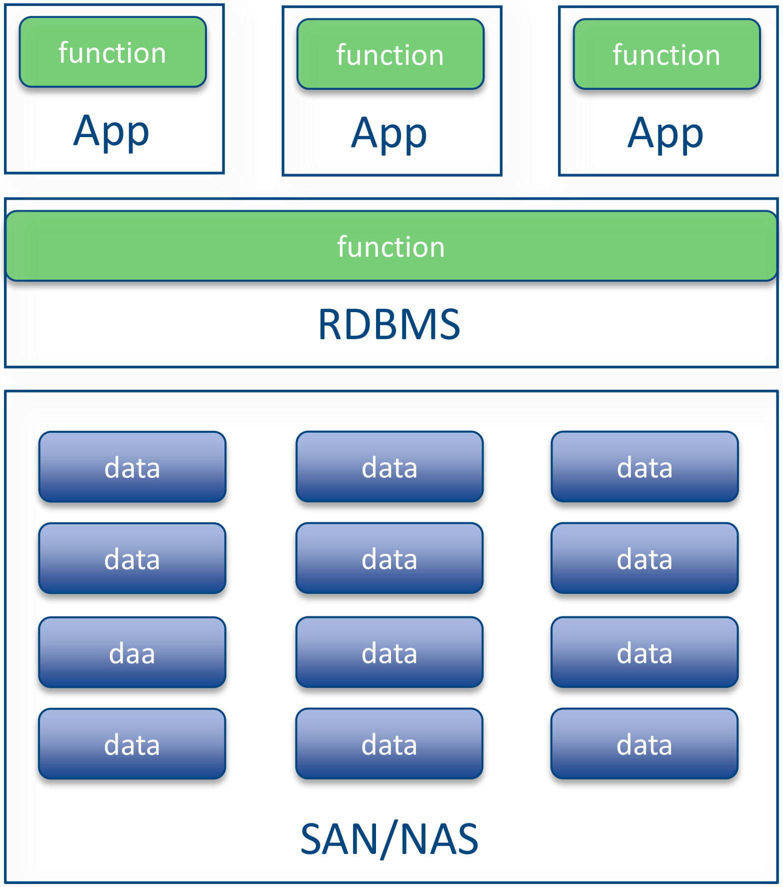
Forget Performance?

Traditional Architecture

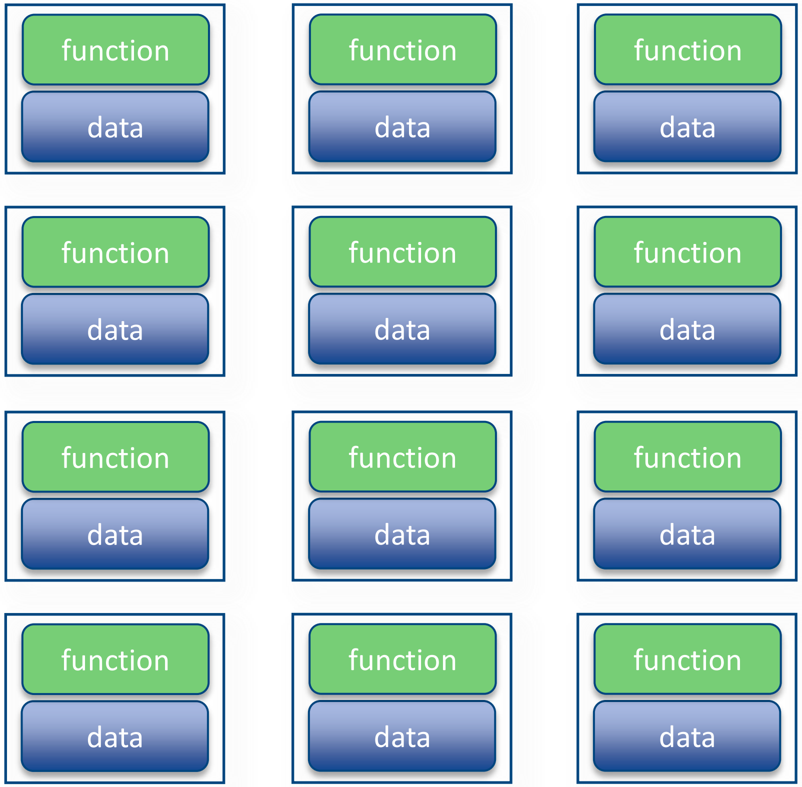


Forget Performance?

Traditional Architecture



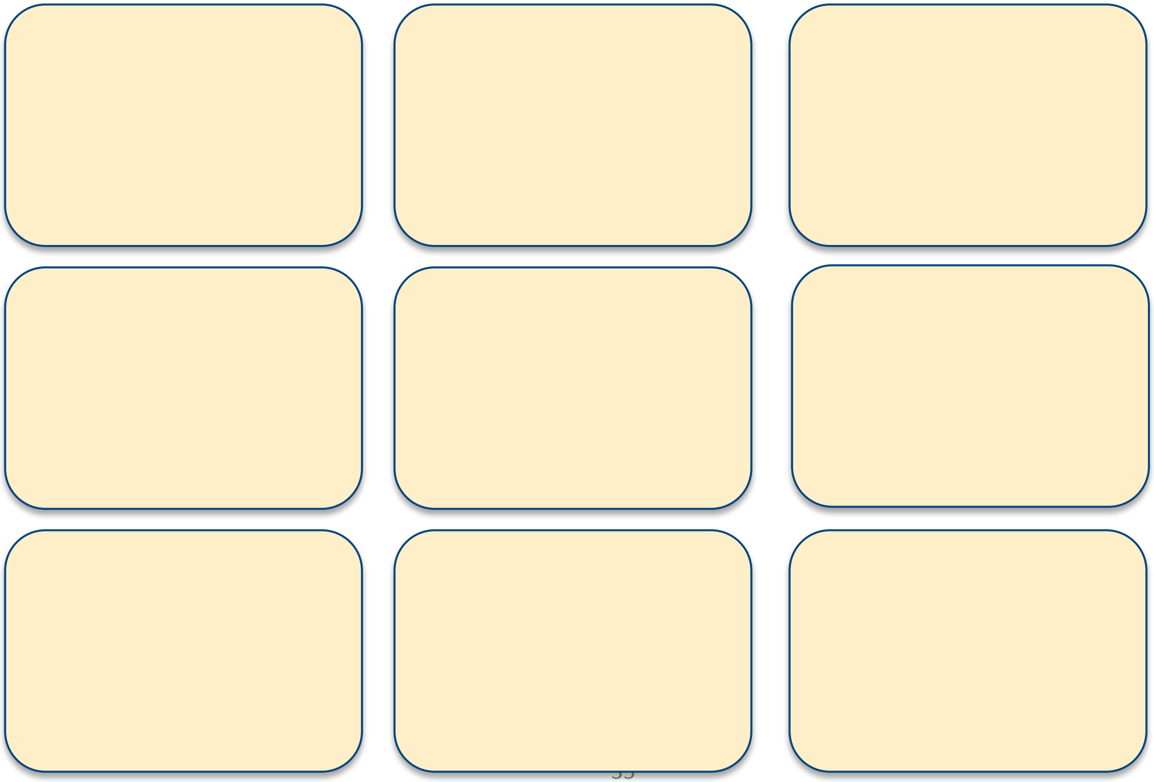
Hadoop



Geographically dispersed also?

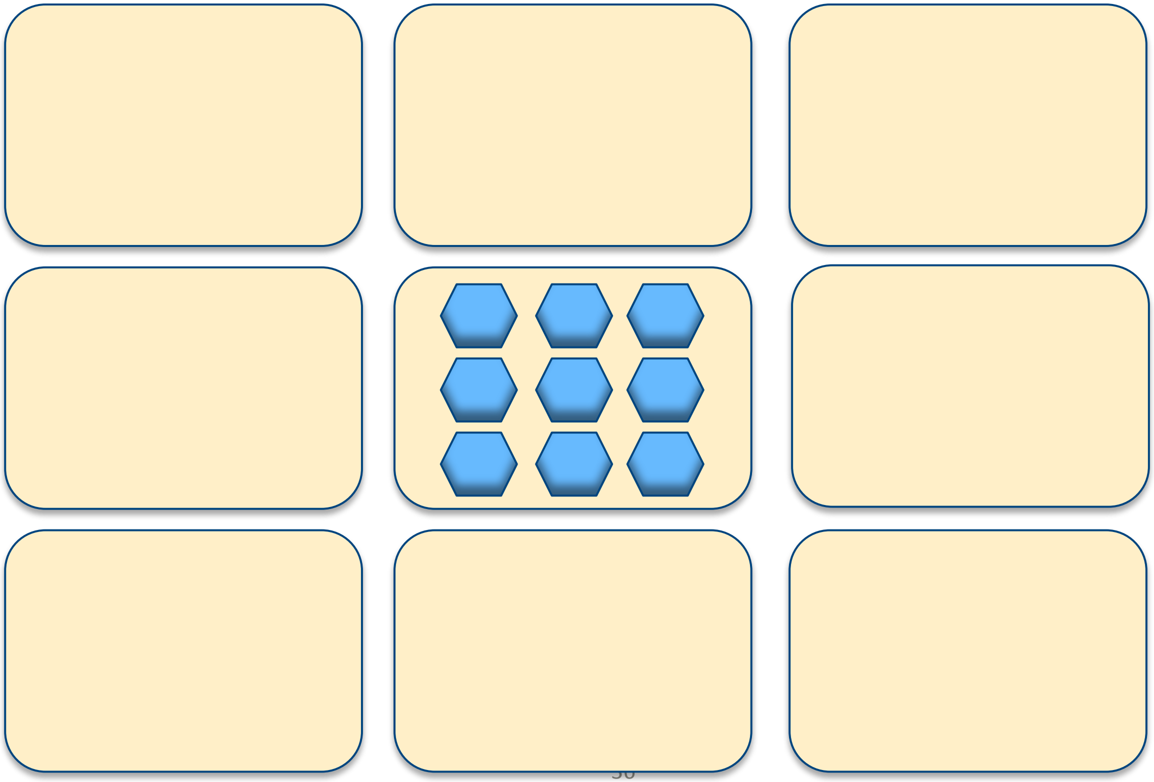
What MapR does

- Chop the data on each node to 1000's of pieces
  - not millions of pieces, only 1000's
  - pieces are called ***containers*** ⬢



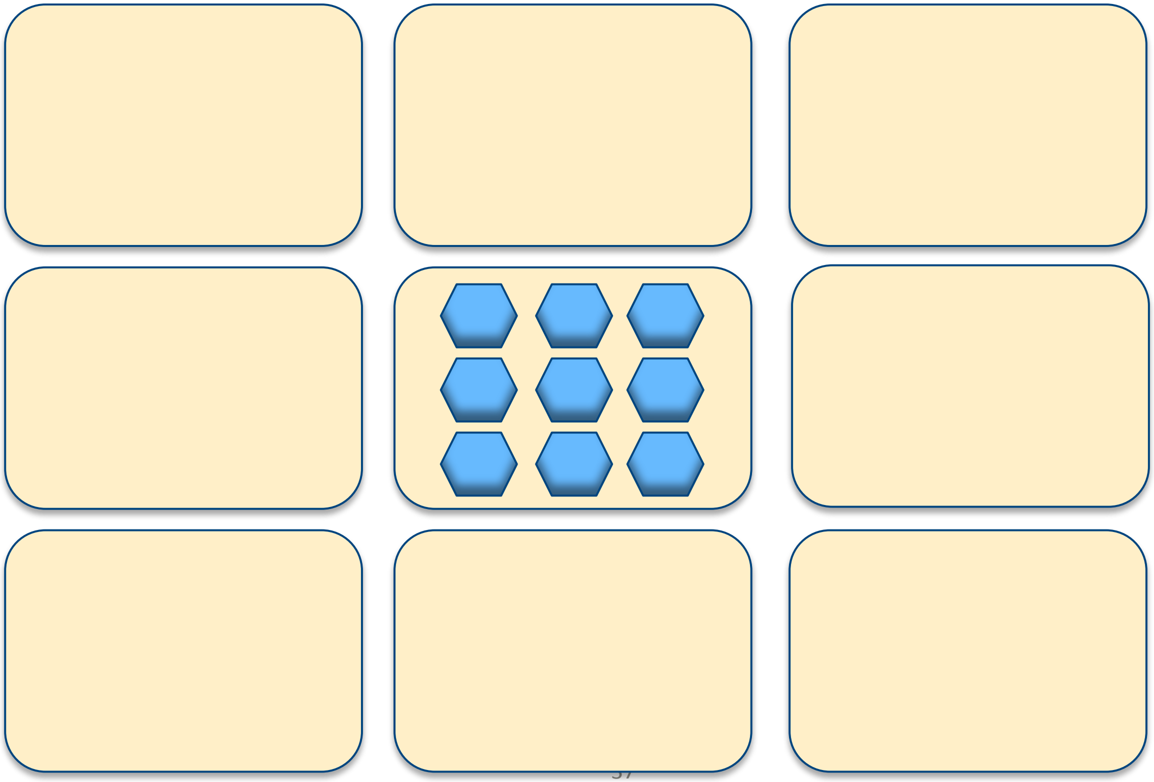
What MapR does

- Chop the data on each node to 1000's of pieces
  - not millions of pieces, only 1000's
  - pieces are called ***containers*** ⬢



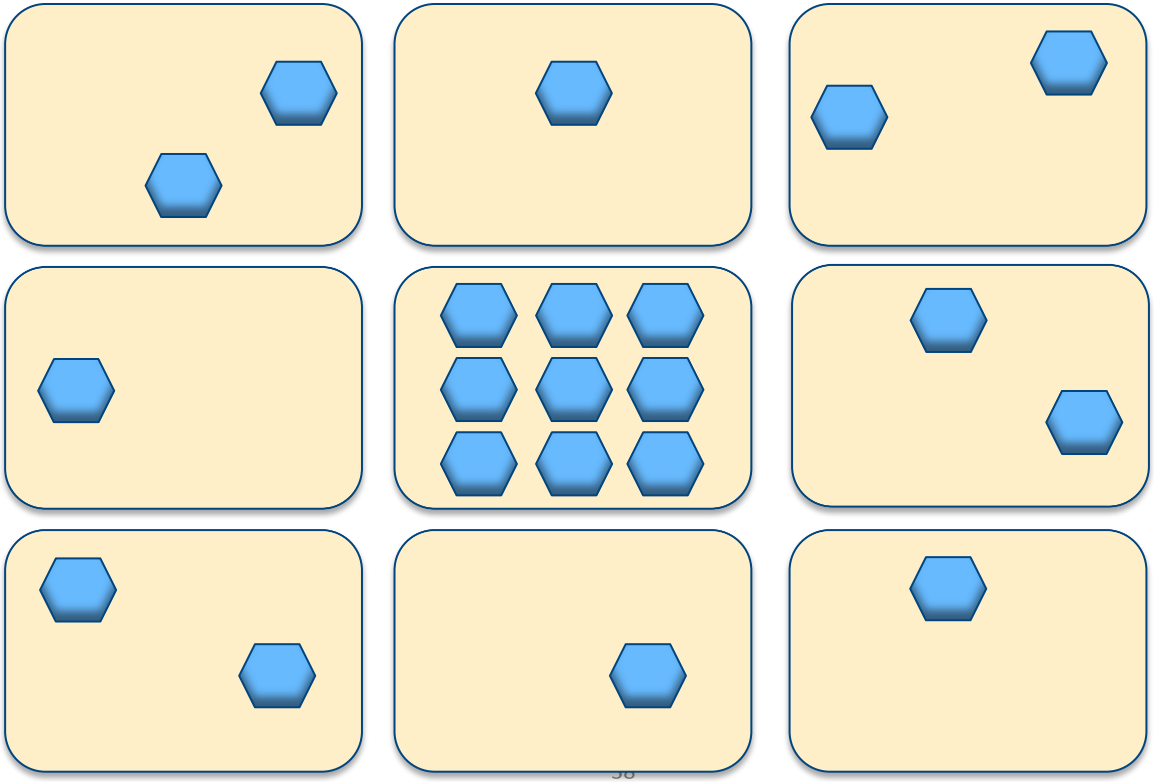
What MapR does

- Chop the data on each node to 1000's of pieces
  - not millions of pieces, only 1000's
  - pieces are called ***containers*** ⬢
- Spread replicas of each container across the cluster



What MapR does

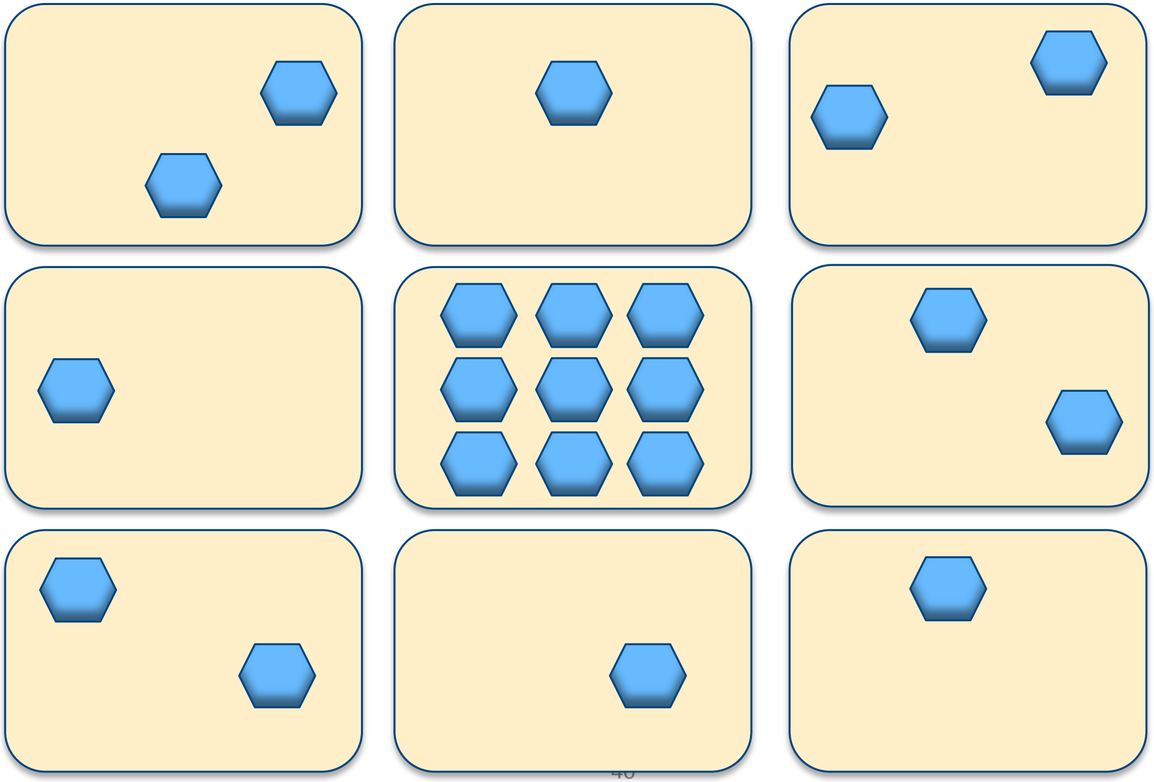
- Chop the data on each node to 1000's of pieces
  - not millions of pieces, only 1000's
  - pieces are called ***containers*** ⬢
- Spread replicas of each container across the cluster



Why does it improve things?

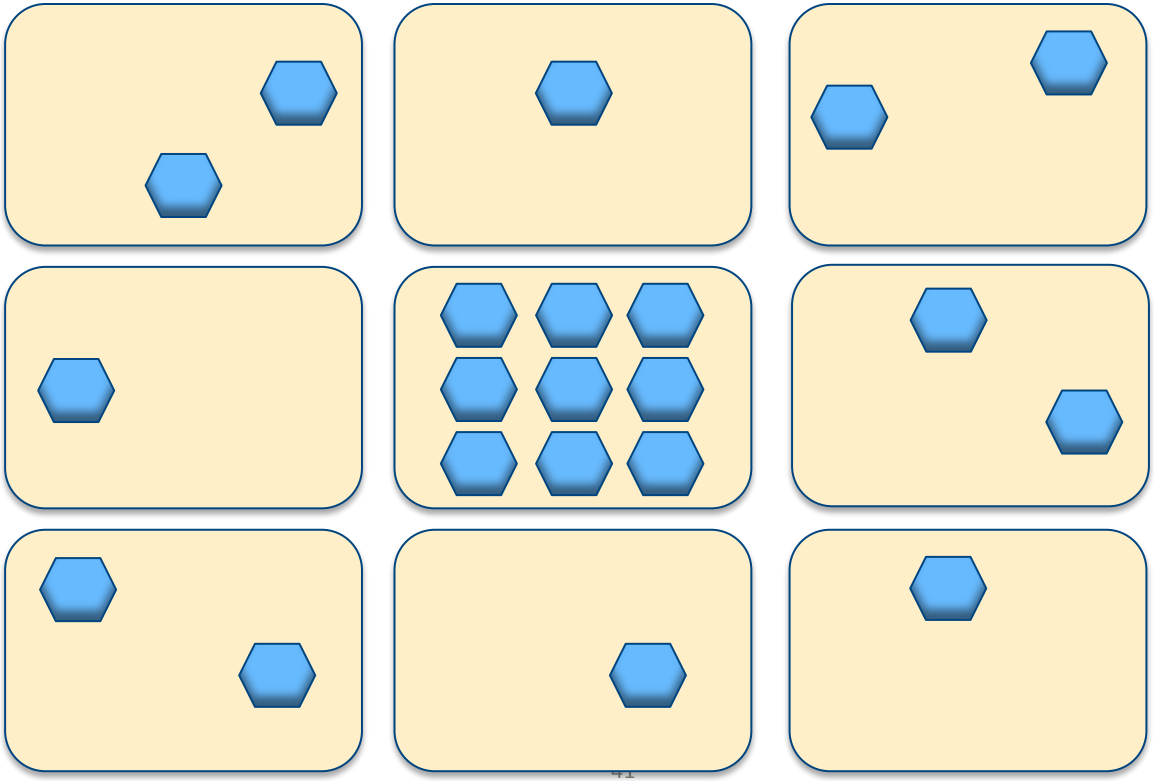
MapR Replication Example

- 100-node cluster
- each node holds 1/100th of every node's data



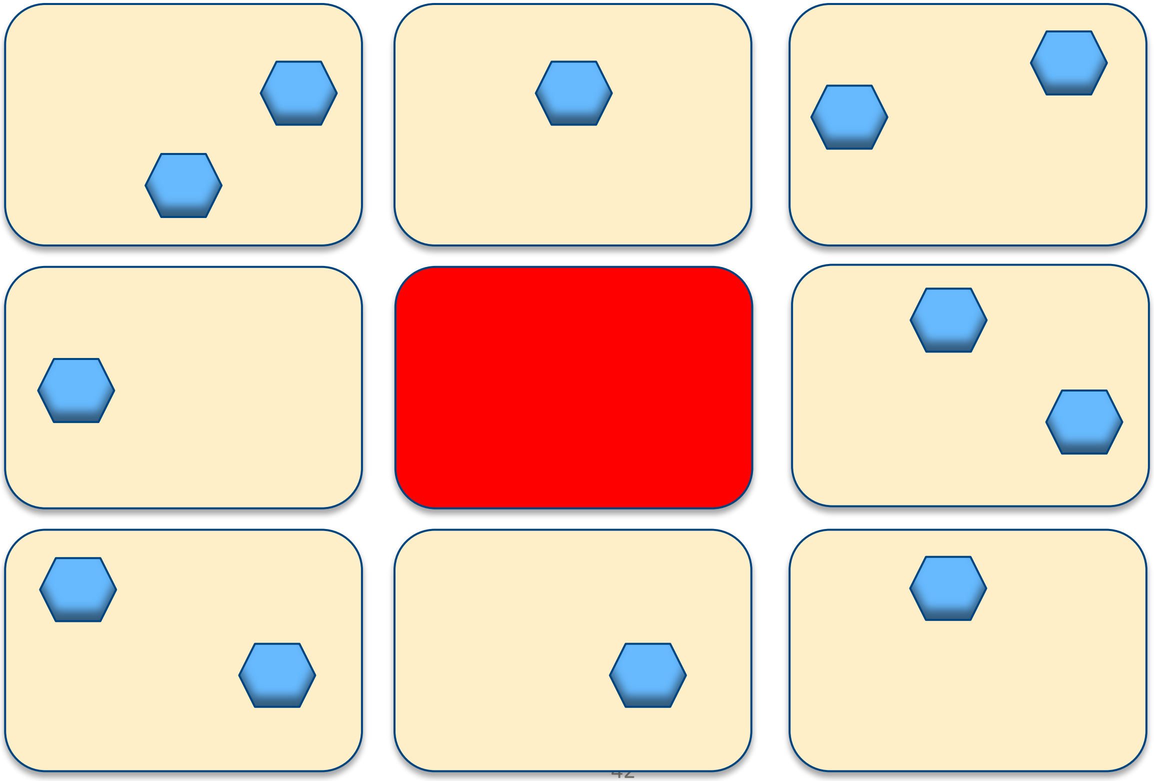
MapR Replication Example

- 100-node cluster
- each node holds 1/100th of every node's data
- when a server dies



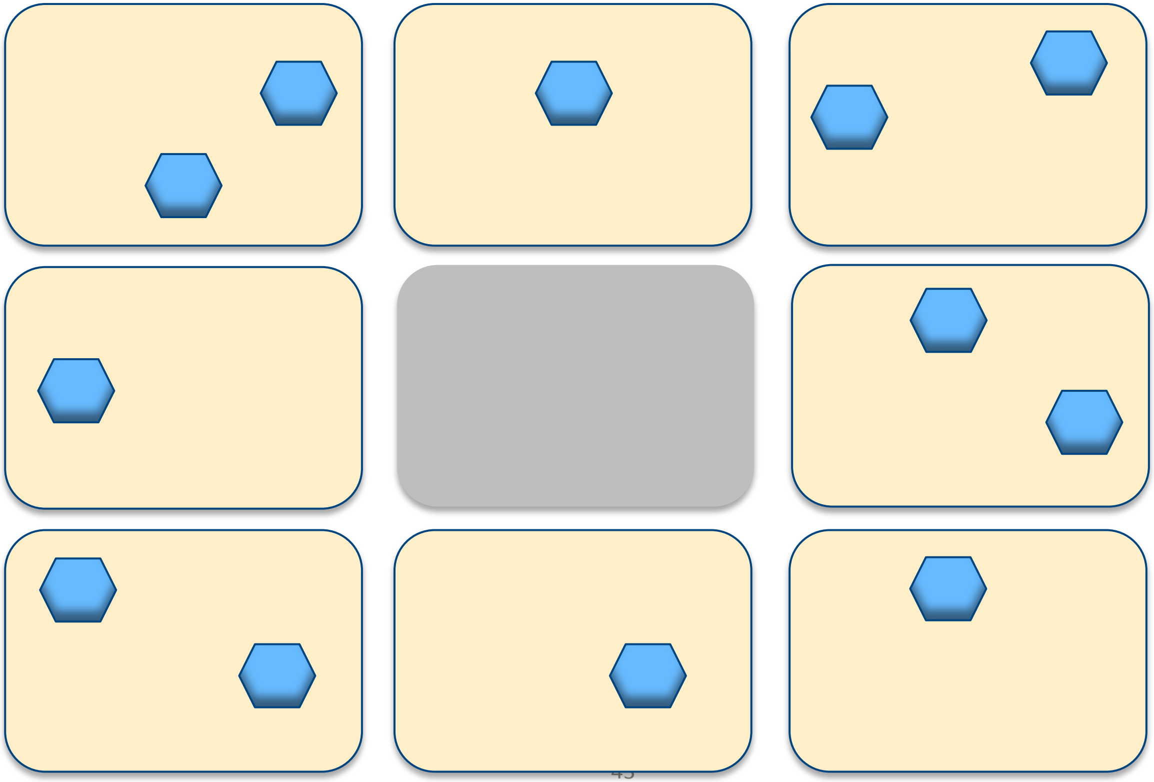
MapR Replication Example

- 100-node cluster
- each node holds 1/100th of every node's data
- when a server dies



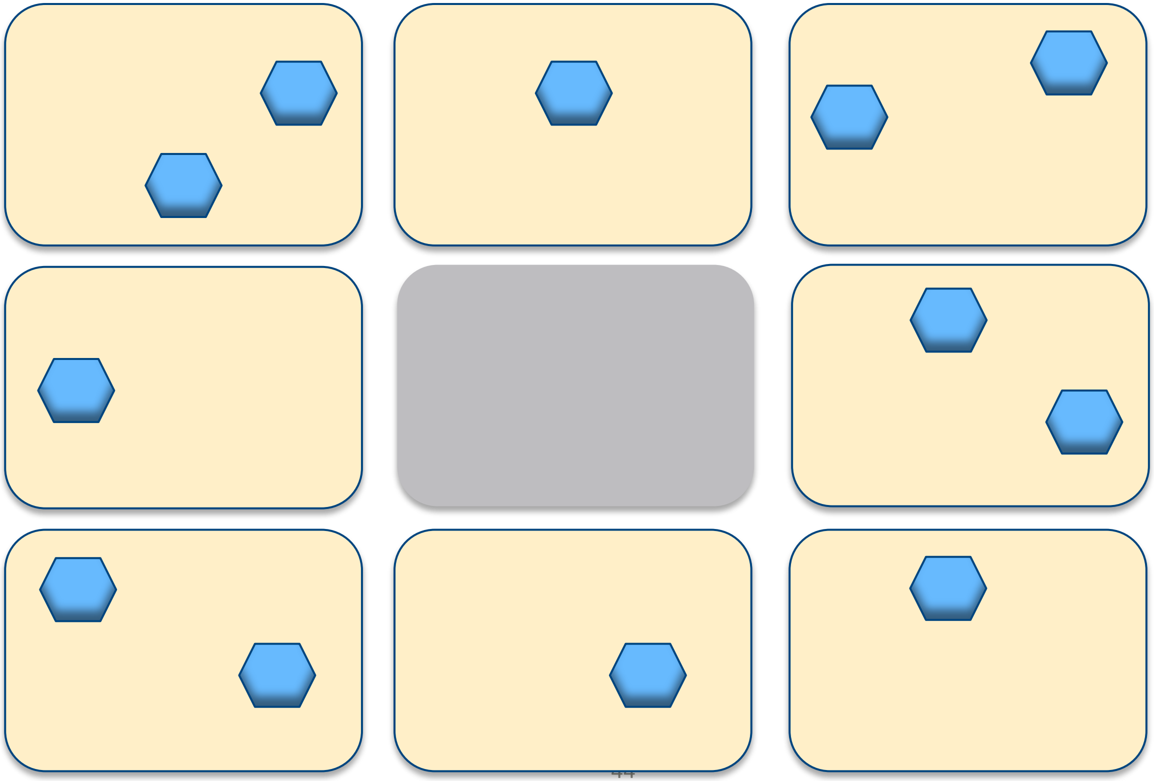
MapR Replication Example

- 100-node cluster
- each node holds 1/100th of every node's data
- when a server dies



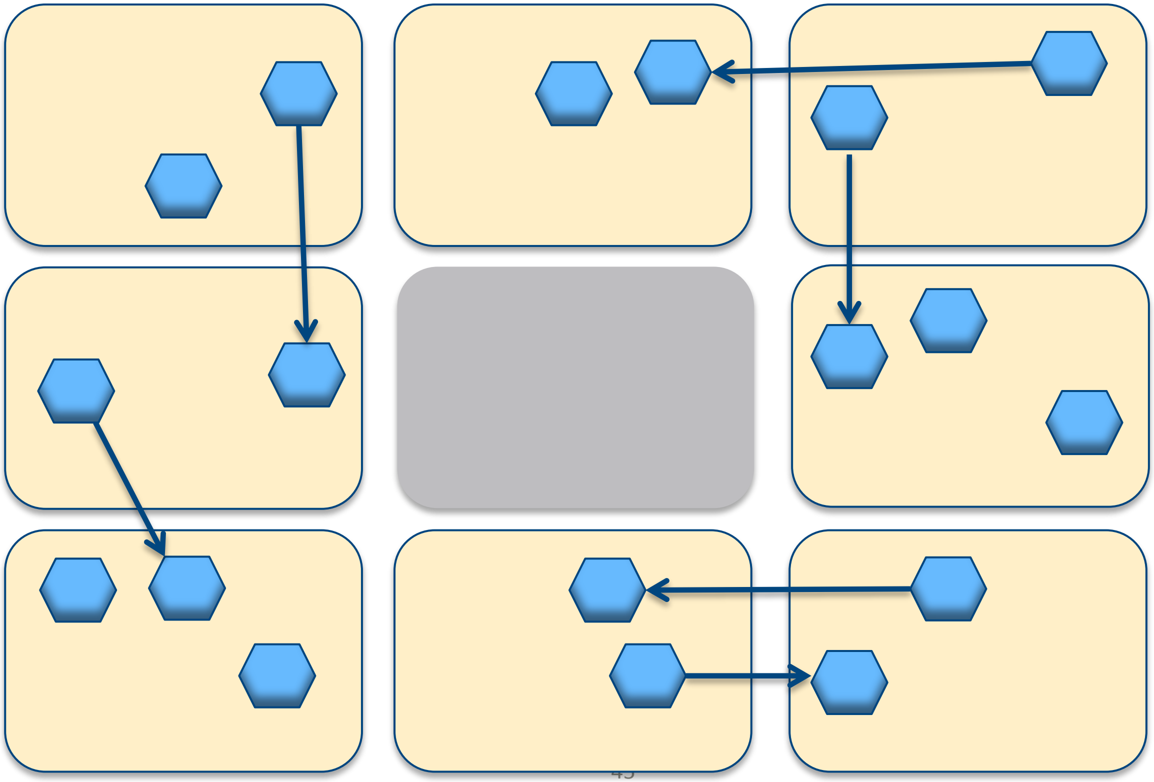
MapR Replication Example

- 100-node cluster
- each node holds 1/100th of every node's data
- when a server dies
- entire cluster resync's the dead node's data



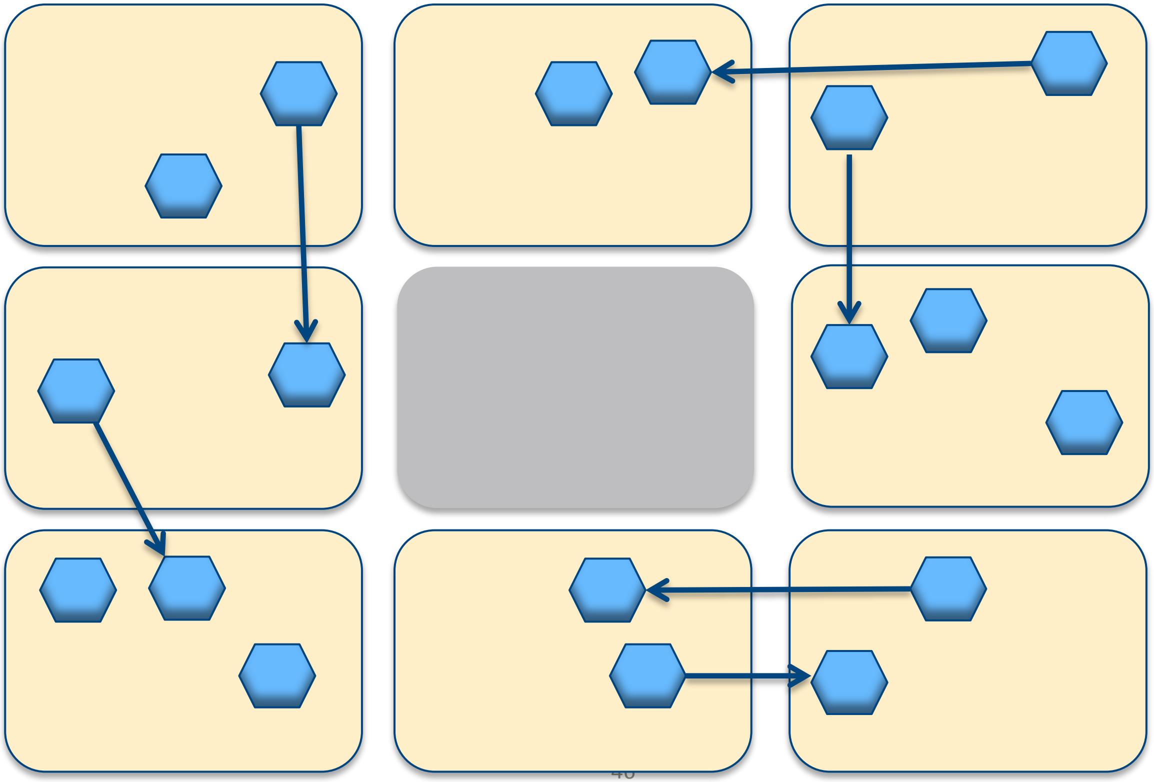
MapR Replication Example

- 100-node cluster
- each node holds 1/100th of every node's data
- when a server dies
- entire cluster resync's the dead node's data



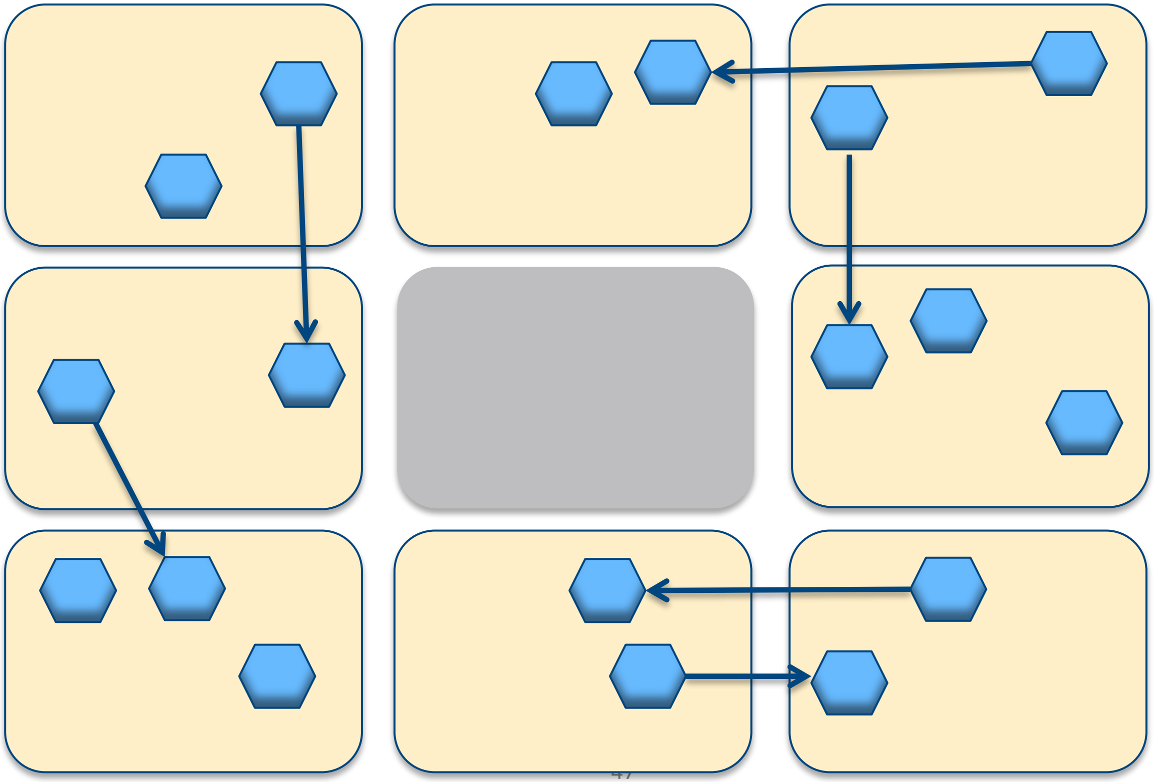
MapR Re-sync Speed

- 99 nodes re-sync'ing in parallel



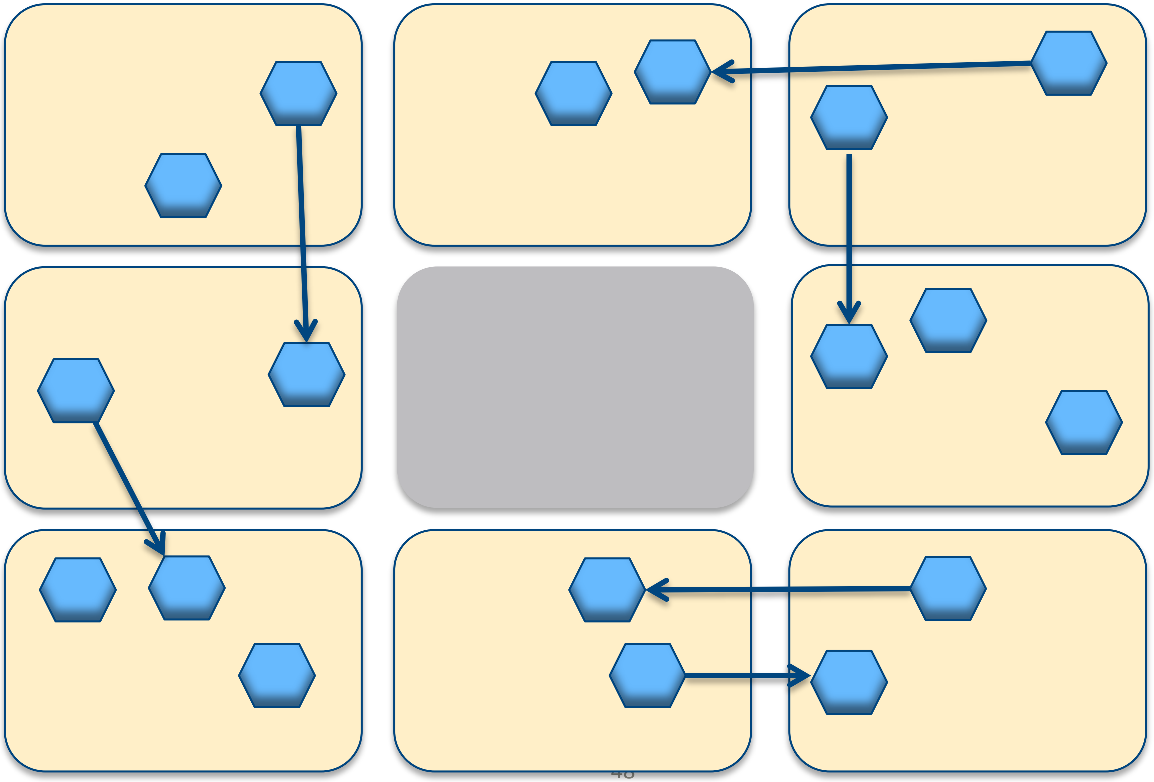
MapR Re-sync Speed

- 99 nodes re-sync'ing in parallel
  - 99x number of drives
  - 99x number of ethernet ports
  - 99x cpu's



MapR Re-sync Speed

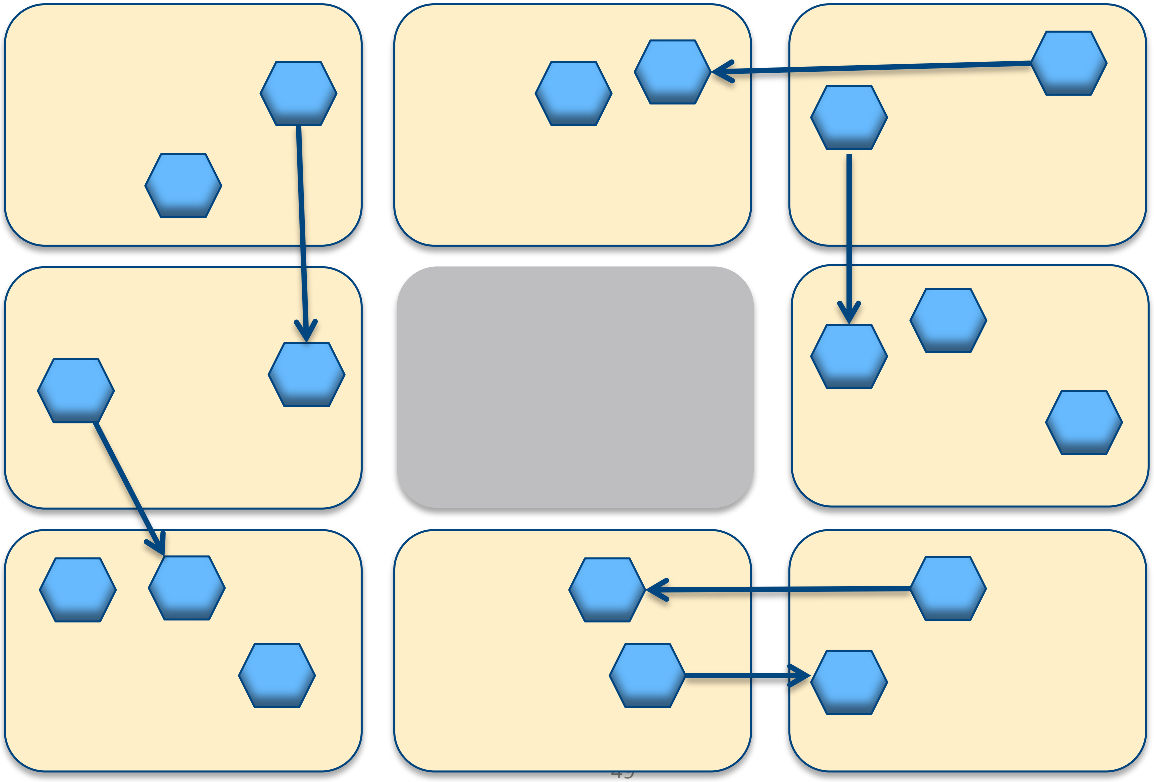
- 99 nodes re-sync'ing in parallel
  - 99x number of drives
  - 99x number of ethernet ports
  - 99x cpu's
- Each is resync'ing 1/100th of the data



MapR Re-sync Speed

- 99 nodes re-sync'ing in parallel
  - 99x number of drives
  - 99x number of ethernet ports
  - 99x cpu's
- Each is resync'ing 1/100th of the data

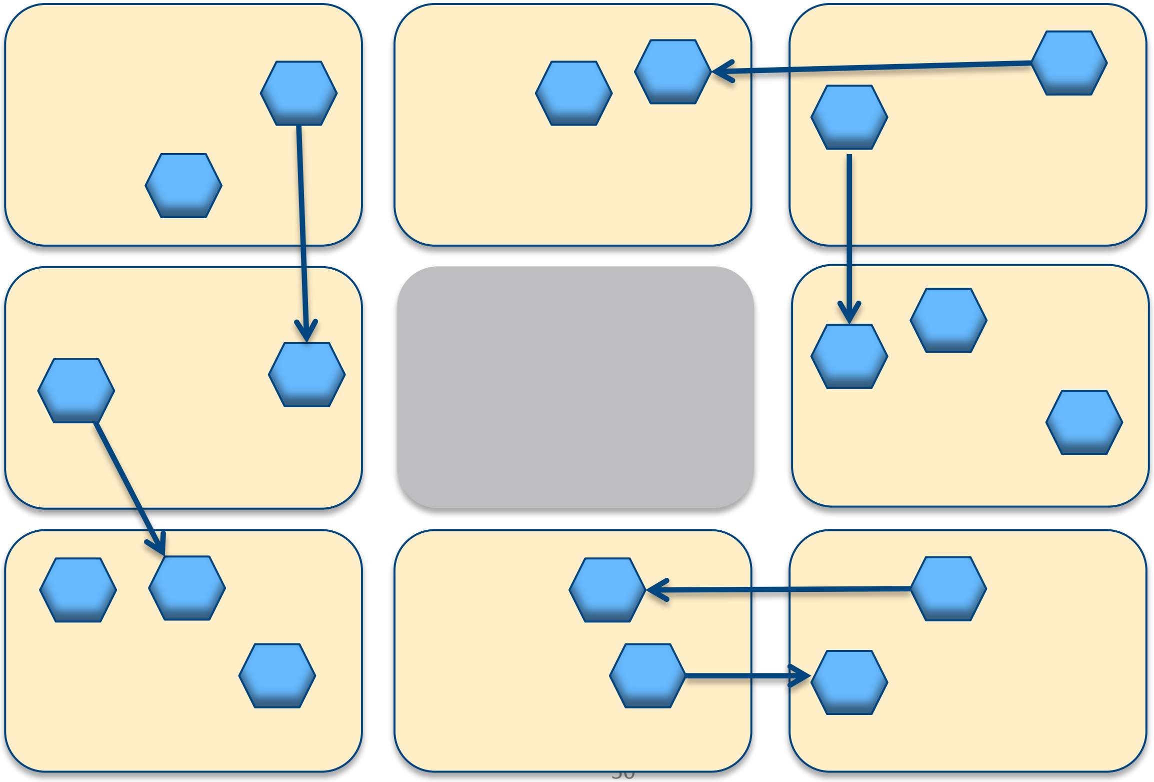
- Net speed up is about 100x
  - 350 hours vs. 3.5



MapR Re-sync Speed

- 99 nodes re-sync'ing in parallel
  - 99x number of drives
  - 99x number of ethernet ports
  - 99x cpu's
- Each is resync'ing 1/100th of the data

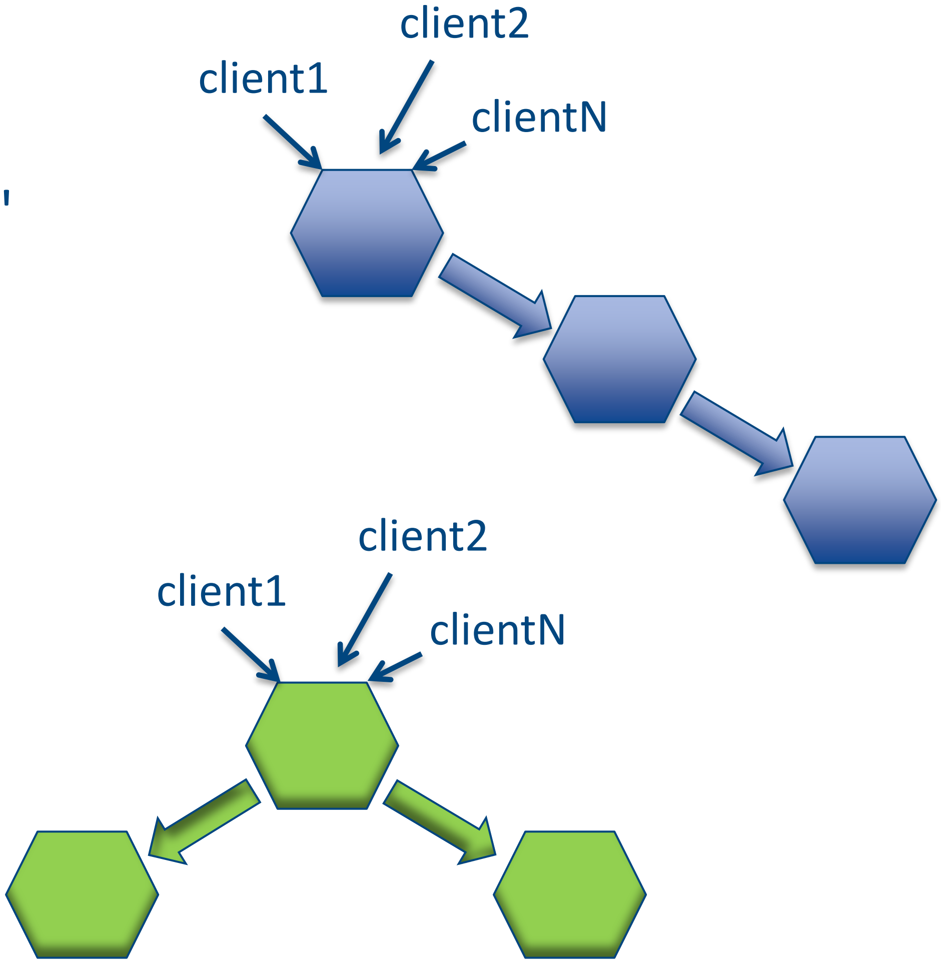
- Net speed up is about 100x
  - 350 hours vs. 3.5
- **MTTDL is 100x better**



Why is this so difficult?

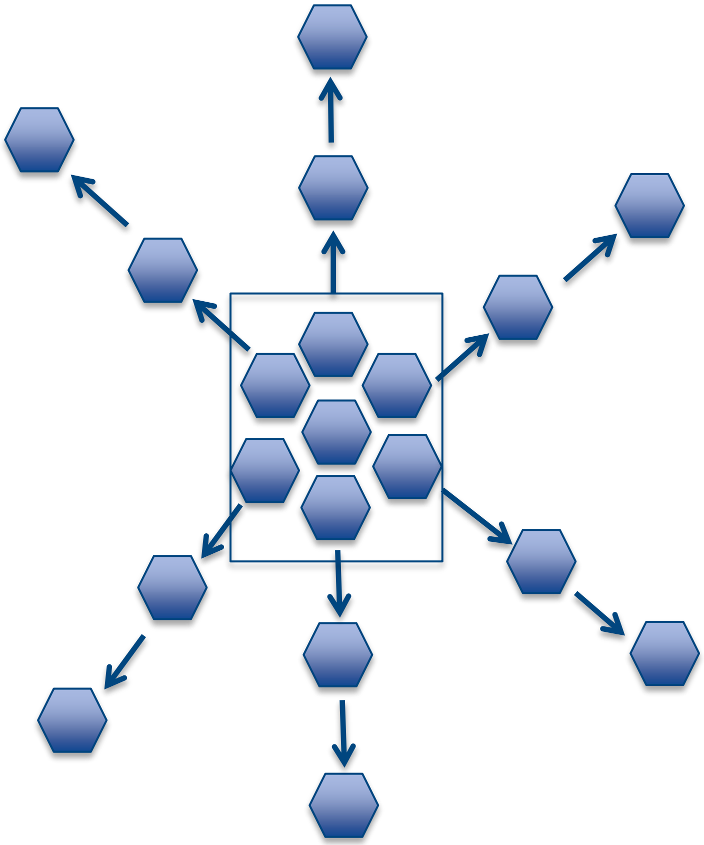
MapR's Read-write Replication

- Writes are synchronous
- Data is replicated in a "chain" fashion
  – utilizes full-duplex network
- Meta-data is replicated in a "star" manner
  – response time better



Container Balancing

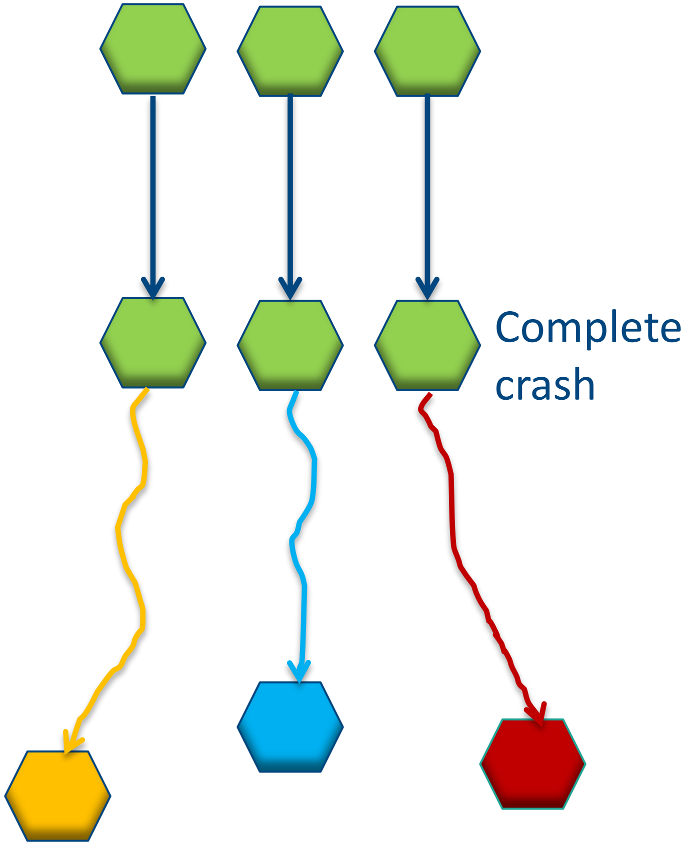
- Servers keep a bunch of containers "ready to go".
- Writes get distributed around the cluster.



- As data size increases, writes spread more, like dropping a pebble in a pond
- Larger pebbles spread the ripples farther
- Space balanced by moving idle containers

MapR Container Resync

- MapR is 100% random write
  - very tough problem
- On a complete crash, all replicas diverge from each other
- On recovery, which one should be master?

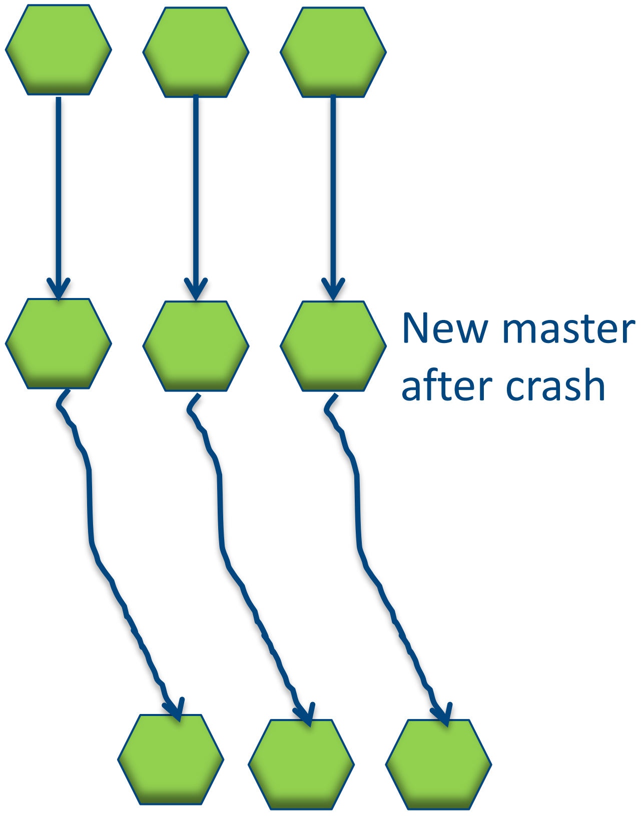


MapR Container Resync

- MapR can detect exactly where replicas diverged
  - even at 2000 MB/s update rate

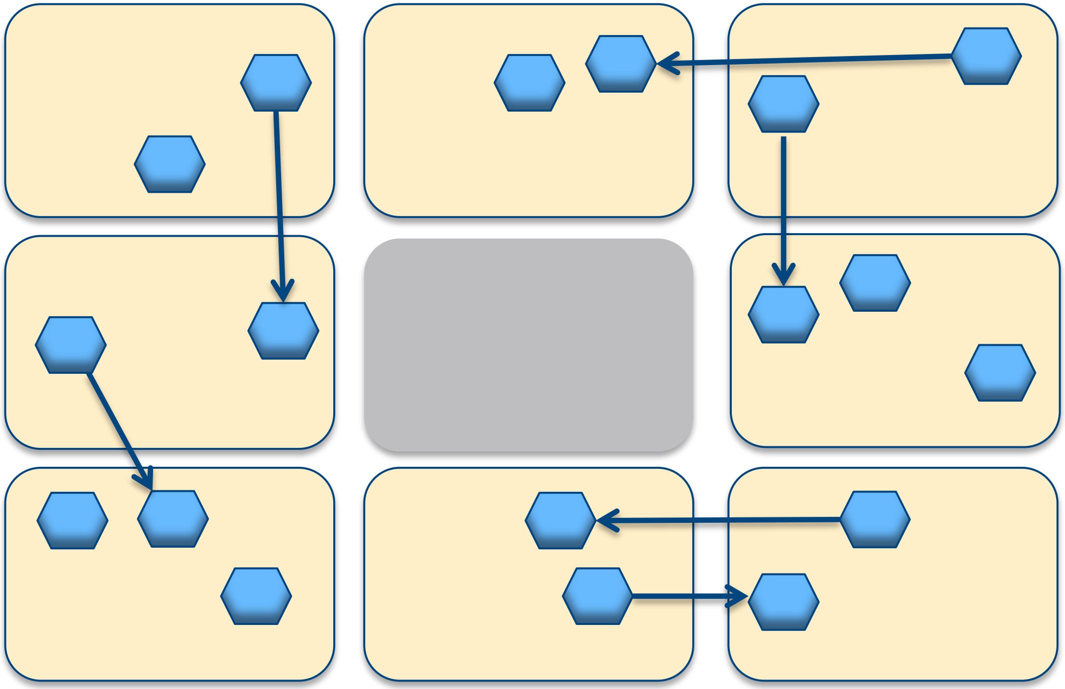
- Resync means
  - roll-back rest to divergence point
  - roll-forward to converge with chosen master

- Done while online
  - with very little impact on normal operations



MapR does Automatic Resync Throttling

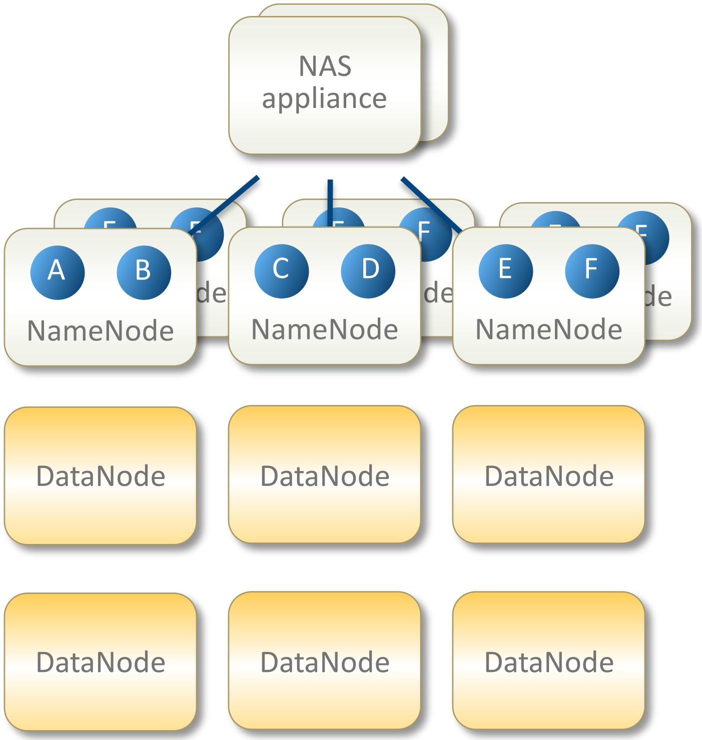
- Resync traffic is “secondary”
- Each node continuously measures RTT to all its peers
- More throttle to slower peers
  - Idle system runs at full speed
- All automatically



Where/how does MapR exploit this unique advantage?

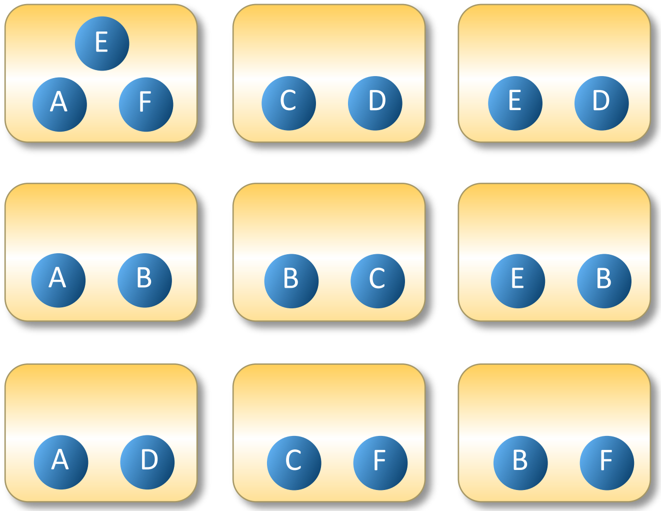
MapR's No-NameNode Architecture

HDFS Federation



- Multiple single points of failure
- Limited to 50-200 million files
- Performance bottleneck
- Commercial NAS required

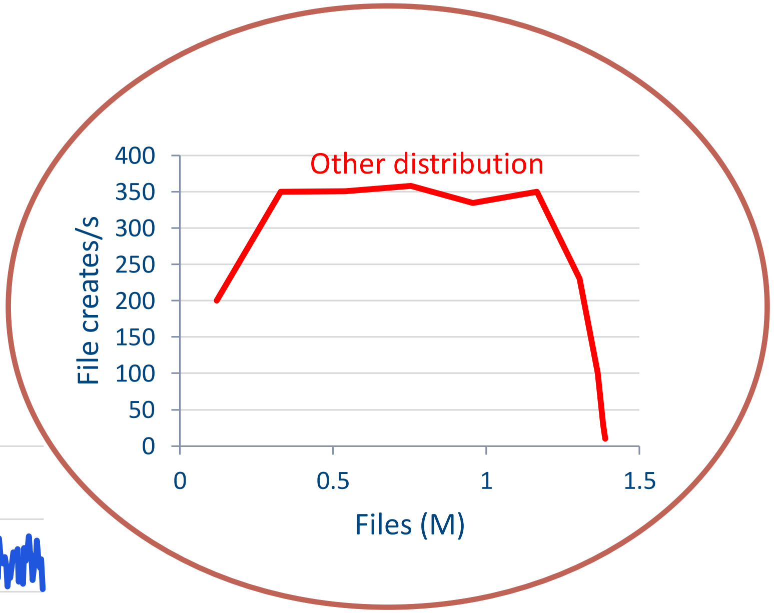
MapR (distributed metadata)

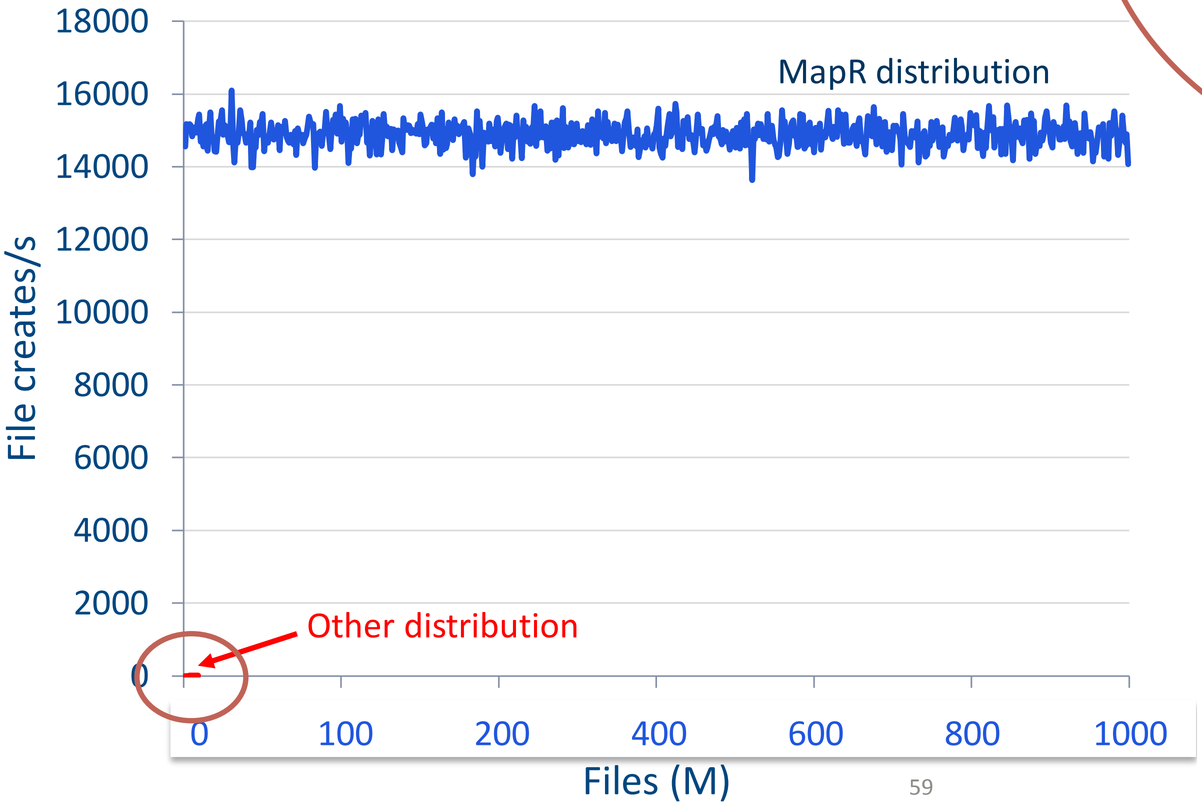


- HA w/ automatic failover
- Instant cluster restart
- Up to 1T files (> 5000x advantage)
- 10-20x higher performance
- 100% commodity hardware

Relative performance and scale

	MapR	Other	Advantage
Rate (creates/s)	14-16K	335-360	40x
Scale (files)	6B	1.3M	4615x



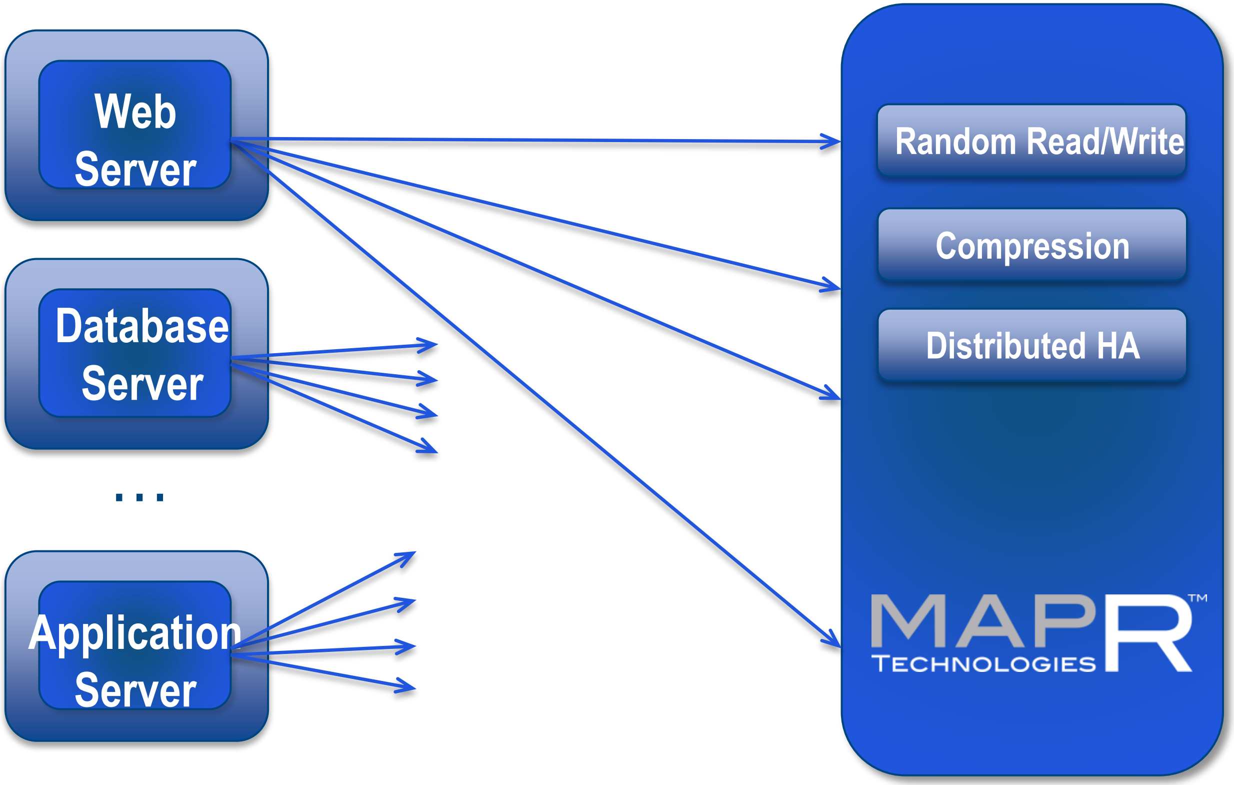


<u>Benchmark:</u> File creates (100B)

<u>Hardware</u>: 10 nodes, 2 x 4 cores, 24 GB RAM, 12 x 1 TB 7200 RPM

Where/how does MapR exploit this unique advantage?

MapR’s NFS allows Direct Deposit

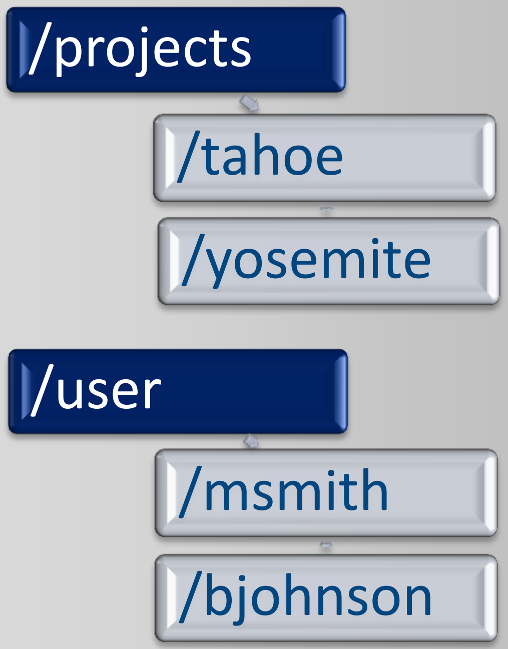


Connectors not needed

No extra scripts or clusters to deploy and maintain

Where/how does MapR exploit this unique advantage?

MapR Volumes



/projects

/tahoe

/yosemite

/user

/msmith

/bjohnson

100K volumes are OK, create as many as desired!

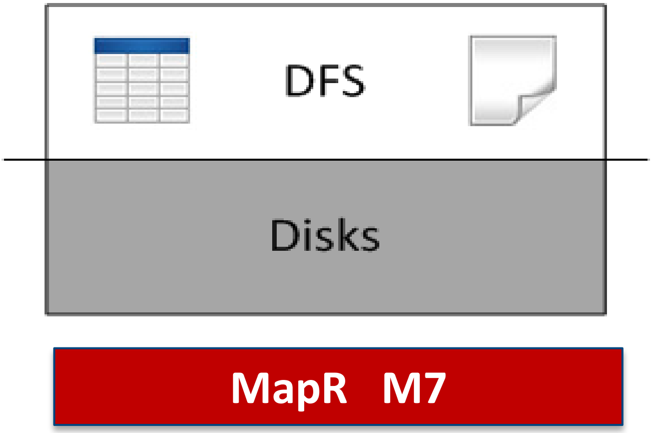
Volumes dramatically simplify the management of Big Data

- Replication factor
- Scheduled mirroring
- Scheduled snapshots
- Data placement control
- User access and tracking
- Administrative permissions

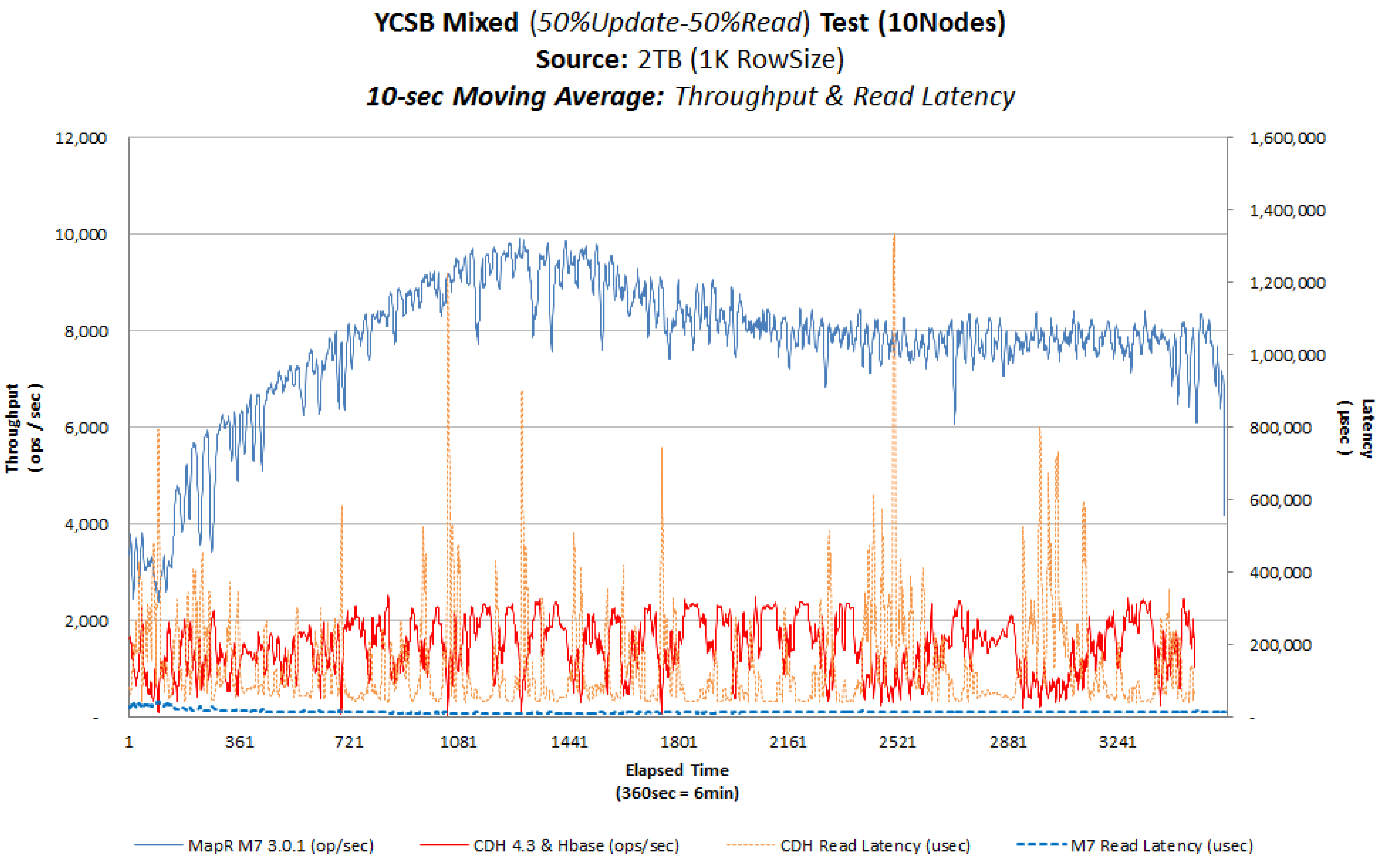
Where/how does MapR exploit this unique advantage?

M7 Tables

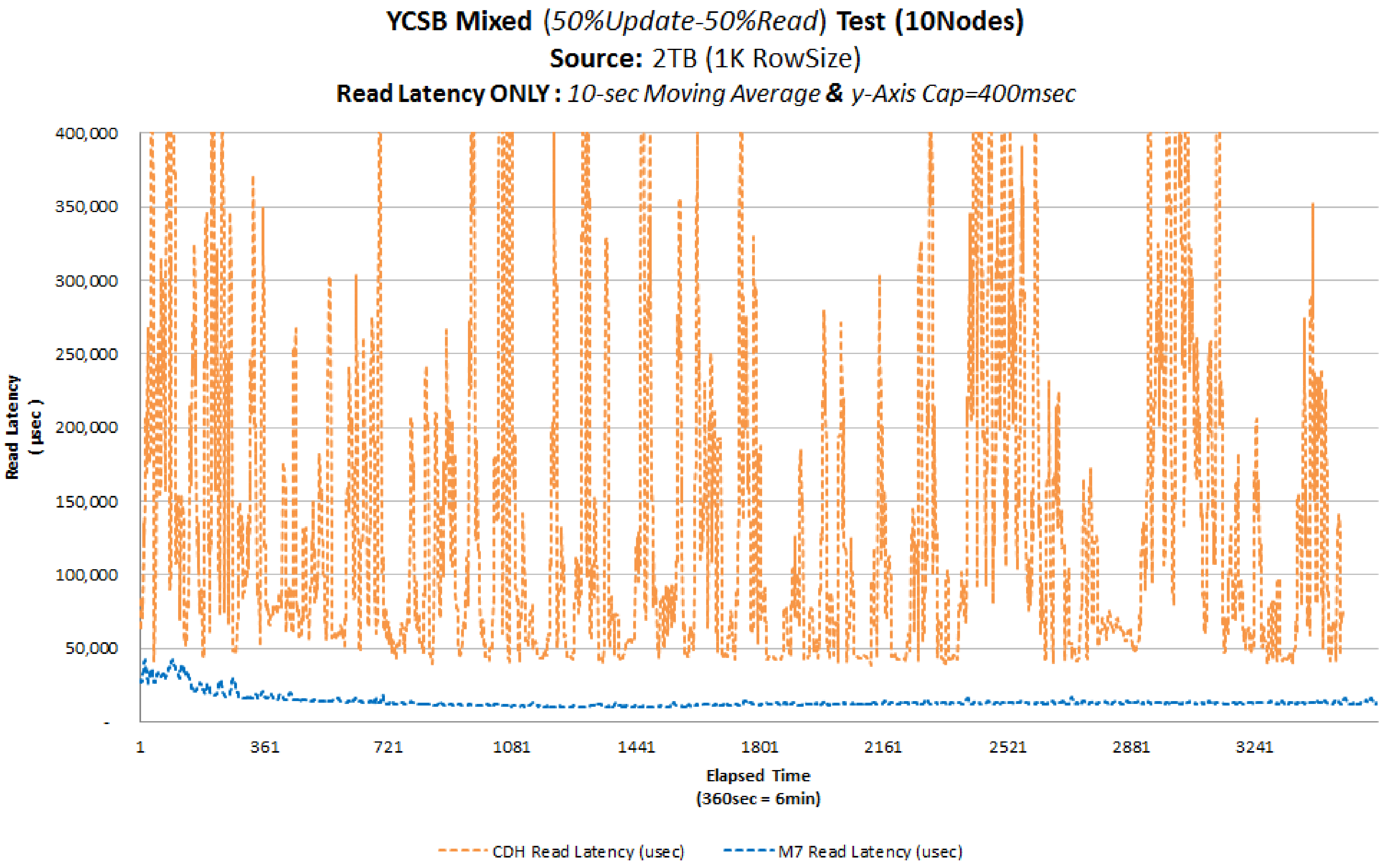
- M7 tables integrated into storage
  - always available on every node, zero admin
- Unlimited number of tables
  - Apache HBase is typically 10-20 tables (max 100)
- **No compactions**
- **Instant-On**
  - zero recovery time
- 5-10x better perf
- Consistent low latency
  - At 95%-ile and 99%-ile



M7 vs. CDH: 50-50 Mix (Reads)



M7 vs. CDH: 50-50 load (read latency)



Where/how does MapR exploit this unique advantage?

MapR makes Hadoop truly HA

- **ALL** Hadoop components are Highly Available, eg, YARN

MapR makes Hadoop truly HA

- **ALL** Hadoop components are Highly Available, eg, YARN
- ApplicationMaster (old JT) and TaskTracker record their state in MapR

MapR makes Hadoop truly HA

- **ALL** Hadoop components are Highly Available, eg, YARN

- ApplicationMaster (old JT) and TaskTracker record their state in MapR
- On node-failure, AM recovers its state from MapR
  - Works even if entire cluster restarted

MapR makes Hadoop truly HA

- **ALL** Hadoop components are Highly Available, eg, YARN

- ApplicationMaster (old JT) and TaskTracker record their state in MapR
- On node-failure, AM recovers its state from MapR
  - Works even if entire cluster restarted
- All jobs resume **from where they were**
  - Only from MapR

MapR makes Hadoop truly HA

- **ALL** Hadoop components are Highly Available, eg, YARN

- ApplicationMaster (old JT) and TaskTracker record their state in MapR
- On node-failure, AM recovers its state from MapR
  - Works even if entire cluster restarted
- All jobs resume **from where they were**
  - Only from MapR
- Allows pre-emption
  - MapR can pre-empt any job, without losing its progress
  - ExpressLane™ feature in MapR exploits it

Where/how can YOU exploit MapR's unique advantage?

- **ALL** your code can easily be scale-out HA

Where/how can YOU exploit MapR's unique advantage?

- **ALL** your code can easily be scale-out HA

- Save service-state in MapR
- Save data in MapR

Where/how can YOU exploit MapR's unique advantage?

- **ALL** your code can easily be scale-out HA

- Save service-state in MapR
- Save data in MapR
- Use Zookeeper to notice service failure

Where/how can YOU exploit MapR's unique advantage?

- **ALL** your code can easily be scale-out HA

- Save service-state in MapR
- Save data in MapR
- Use Zookeeper to notice service failure
- Restart anywhere, data+state will move there automatically

Where/how can YOU exploit MapR's unique advantage?

- **ALL** your code can easily be scale-out HA

- Save service-state in MapR
- Save data in MapR
- Use Zookeeper to notice service failure
- Restart anywhere, data+state will move there automatically
- That's what we did!

Where/how can YOU exploit MapR’s unique advantage?

- **ALL** your code can easily be scale-out HA

- Save service-state in MapR
- Save data in MapR
- Use Zookeeper to notice service failure
- Restart anywhere, data+state will move there automatically
- That’s what we did!
- Only from MapR: HA for Impala, Hive, Oozie, Storm, MySQL, SOLR/Lucene, Kafka, …

MapR: Unlimited Scale

# files, # tables	trillions
# rows per table	trillions
# data	1-10 Exabytes
# nodes	10,000+

Build cluster brick by brick, one node at a time

- Use commodity hardware at rock-bottom prices
- Get enterprise-class reliability: instant-restart, snapshots, mirrors, no-single-point-of-failure, …
- Export via NFS, ODBC, Hadoop and other std protocols