



Beyond the marketing and hype of big data

Why design choices and customer needs matter

Background

Me

- Software Architect
- Apache Drill Comitter and PMC, focused on building a new massively scalable query layer
- Previously
 - 4 years search startup, failed
 - 2 years ad startup, acq. by AOL
 - 8 years big data analytics, acq. by MS

Where I work

- MapR Technologies
- 150 person enterprise software startup in South Bay
- Hadoop and NoSQL for large enterprises
- Open source and proprietary combination
- Focused on performance and stability

Short version...

- Big data is big
- The internet and Hadoop make the kernel
- If you want to make big money by being a good engineer, be a systems engineer because that's where design matters (examples)
- But only when it focuses on customer problems in big markets

Big Data example

Climate change

“There is no question that climate change is happening; the only arguable point is what part humans are playing in it.”

–David Attenborough

If we accept climate change is occurring,
what do we do next?

What is the impact of this on farmers?

- Unpredictable growing seasons
- More frequent problems with drought, flooding and frost
- Crop damage and loss
- Farm bankruptcy

Is it possible to insure against this possibility?

How do you build actuarial tables?

- Soil samples
 - Extensive weather sensor and observations
 - Substantial historical data
-
- You can do it at a state level.
 - You can probably do it at a county level.
 - How do you do it at a farmer level?

Climate Corporation has done this 2006



- More than 50 years of public weather and NOAA data
- More than 2.5mm private and public sensors currently collecting daily, hourly and minute-level observations
- >150 billion soil samples
- >10 trillion data points in total
- The most sophisticated weather models ever created

Country-wide availability of farm-level
crop insurance

How could they do this?

- The granularity of data had to be sufficient to provide a meaningful model
 - Lesser granularity means hard to monetize
 - Sensor cost had to be low
- The cost for storing and analyzing this dataset must be relative small
 - Hardware costs
 - Software capabilities and costs
- Unclear if this was even possible until this decade

Big Data Movement

Analysis problems that combine:

- Extraordinary quantities of data
- Affordable storage
- Massively parallel processing

Growth of Big Data problems

Internet Driven

- Explosion in available observations
 - Larger than anything that has existed previously
- Money and monetization opportunities
- User behavior analysis and advertising

Leveraged in real world

- Reduction in cost of sensors
- Cost of connectedness
- Warehousing, fraud detection, retail analytics

How big is it before I call it big data?

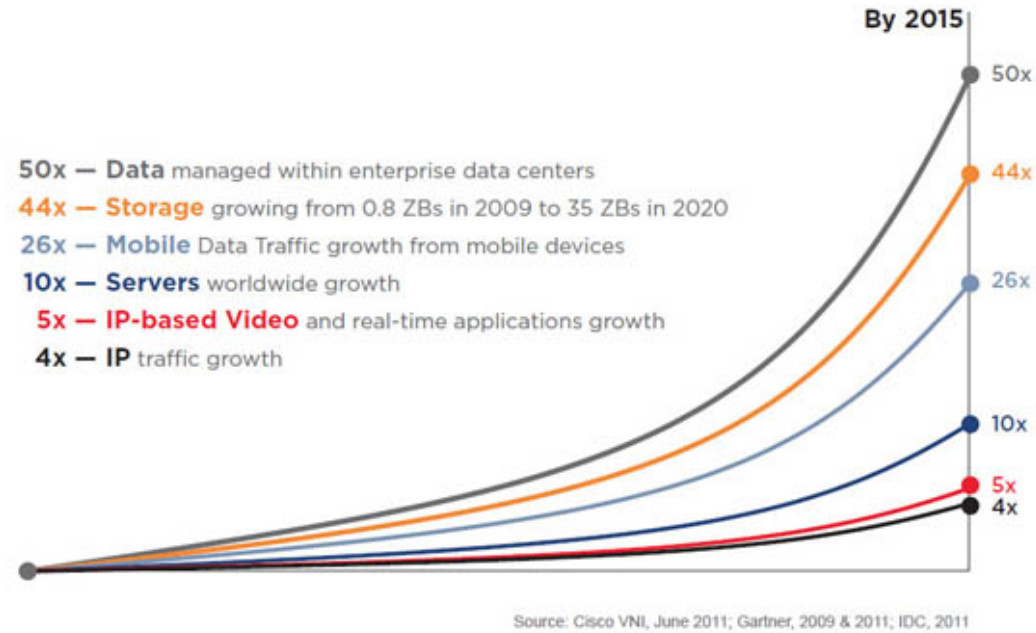
- 100s of terabytes to petabytes
 - Quantity of records, not just volume of bytes
- 100-1000s of nodes



Big Data Landscape

Solving Big Data Problems

- Scale comes first
 - Growth is outpacing everything
- Scale-out commodity hardware
- Expect failure and work around it
- No per byte pricing



NoSQL

- Not SQL => Not Only SQL
- Started by sharding systems like MySQL or Oracle and providing applications layer on top to manage partitioning of data
 - Huge loss of functionality
 - Administrative Overhead
- Then rose new systems that shared loss in functionality but managed shards and client direction
- They didn't have ACID, transactions, the SQL query languages, but they were able to scale (and that mattered more than anything else).
- Primarily focused on OLtP (small t) workloads, some simplistic analytical capabilities

NoSQL

A Lightweight SQL Db
Overview | Branding | Bugs | Blueprint

riak

MONETDB
open-source database system for high-performance applications

redis

M
MONETDB

mongoDB
name: "mongo" type: "DB"

SCALIEN

ZopeDB

Firebird
Innovative RDBMS software
That's going where you're going

4store

4store, an efficient, scalable and stable RDF database

TITAN

Cassandra

ORACLE
BERKELEY DB

V

Dynamo DB

HSQL database engine
hsqldb - 100% Java Database

Shoal

GIGASPACE

apache CouchDB
relax

CloudKit

HYPERTABLE INC

LUCIDDB

Neo4j
The World's Leading Graph Database

FlockDB

Persevere
about | getting started | documentation | database explorer

openneptune
Distributed Structured Data Storage

Vertex DB

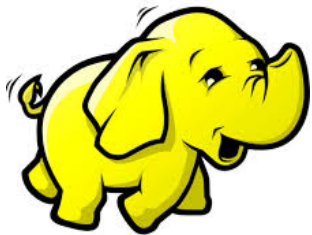
MariaDB
a backward compatible, drop-in replacement branch of the MySQL Database Server.

thruodb
Document-Oriented Database Service



Hadoop

- Ecosystem of loosely connected components
- Built on top of a shared distributed file system
- Primarily leverage a common parallel processing framework
- Initially focused on batch analytical workloads, then extended to interactive workloads and some OLtP use cases



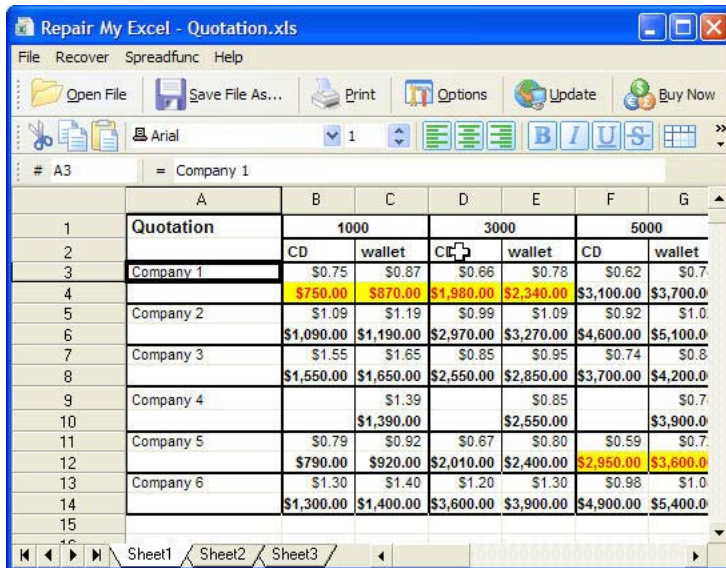
APACHE
HBASE



If Scale is core, how to keep things consistent

- If a node fails, what happens to latency
- Option 1: Fully consistent, some downtime
- Option 2: Full uptime, possibility of inconsistency
- Market:
 - Developers find it hard to think with inconsistent models
 - Amazon, Facebook, and most other companies have avoided inconsistency in all but the most
 - Fast failover is fast enough in most cases
 - Gaming, IM, some other places, data inconsistency or loss is better than high latency

Data no longer looks like Excel

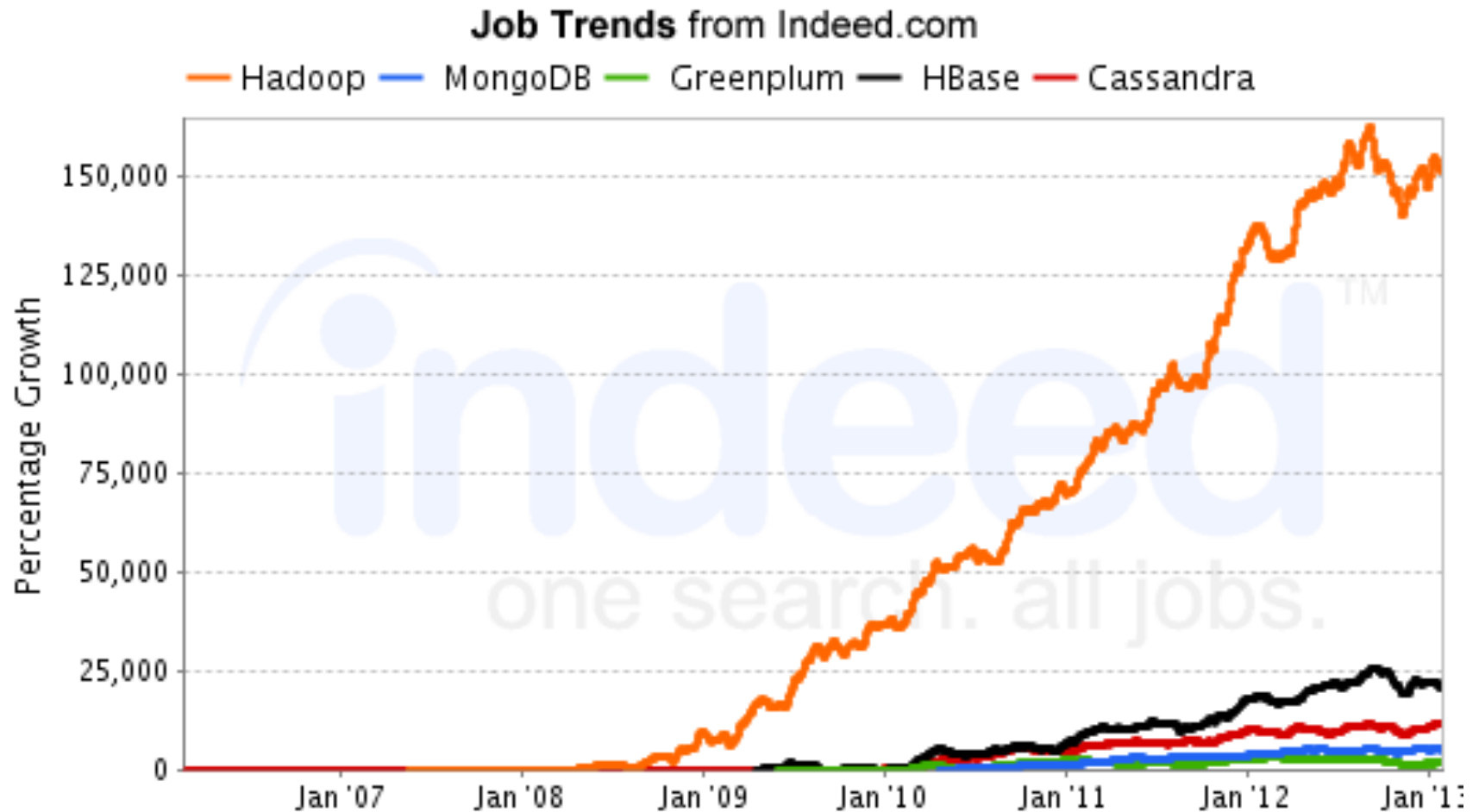


	A	B	C	D	E	F	G
1	Quotation	1000		3000		5000	
2		CD	wallet	CD	wallet	CD	wallet
3	Company 1	\$0.75	\$0.87	\$0.66	\$0.78	\$0.62	\$0.7
4		\$750.00	\$870.00	\$1,980.00	\$2,340.00	\$3,100.00	\$3,700.0
5	Company 2	\$1.09	\$1.19	\$0.99	\$1.09	\$0.92	\$1.0
6		\$1,090.00	\$1,190.00	\$2,970.00	\$3,270.00	\$4,600.00	\$5,100.0
7	Company 3	\$1.55	\$1.85	\$0.85	\$0.95	\$0.74	\$0.8
8		\$1,550.00	\$1,650.00	\$2,550.00	\$2,850.00	\$3,700.00	\$4,200.0
9	Company 4		\$1.39		\$0.85		\$0.7
10			\$1,390.00		\$2,550.00		\$3,900.0
11	Company 5	\$0.79	\$0.92	\$0.67	\$0.80	\$0.59	\$0.7
12		\$790.00	\$920.00	\$2,010.00	\$2,400.00	\$2,950.00	\$3,600.0
13	Company 6	\$1.30	\$1.40	\$1.20	\$1.30	\$0.98	\$1.0
14		\$1,300.00	\$1,400.00	\$3,600.00	\$3,900.00	\$4,900.00	\$5,400.0
15							



```
{  
  name: Jacques  
  spouse: {  
    name: Sarah  
  }  
  pets: [  
    { name: Robert, type: Dog},  
    { name: Charles, type: Ferret}  
  ]  
}
```

Hadoop Growth



When Technology Design Choices Matter

Consumer: Adoption is king, everything else doesn't matter

Twitter.com downtime in 2007		
Month	Uptime (%)	Downtime (d, h, m)
<u>December 2007</u>	97.51%	10 hours, 59 mins
<u>November 2007</u>	98.74%	9 hours, 4 mins
<u>October 2007</u>	99.02%	7 hours, 19 mins
<u>September 2007</u>	99.06%	6 hours, 44 mins
<u>August 2007</u>	98.47%	11 hours, 22 mins
<u>July 2007</u>	99.59%	3 hours, 2 mins
<u>June 2007</u>	99.08%	6 hours, 37 mins
<u>May 2007</u>	97.67%	17 hours, 9 mins
<u>April 2007</u>	99.11%	6 hours, 15 mins
<u>March 2007</u>	92.09%	2 day, 10 hours, 3 mins
<u>February 2007</u>	98.21%	6 hours, 22 mins
TOTAL	98.06%	5 days, 23 hours, 0 mins

Consumer and application success is driven by customer adoption rather than technical innovation

Enterprise: Useful Differentiated Tech

- The explosion of open source has brought solutions for most common problems
 - Makes it harder to commercialize a simple application
 - But, building business off pure open source plays haven't built many long term winners, Red Hat being the only clear example
- Opportunity is in true technical innovation and differentiated IP
- Leverage open source as much as possible
- Build a defensible technology offering

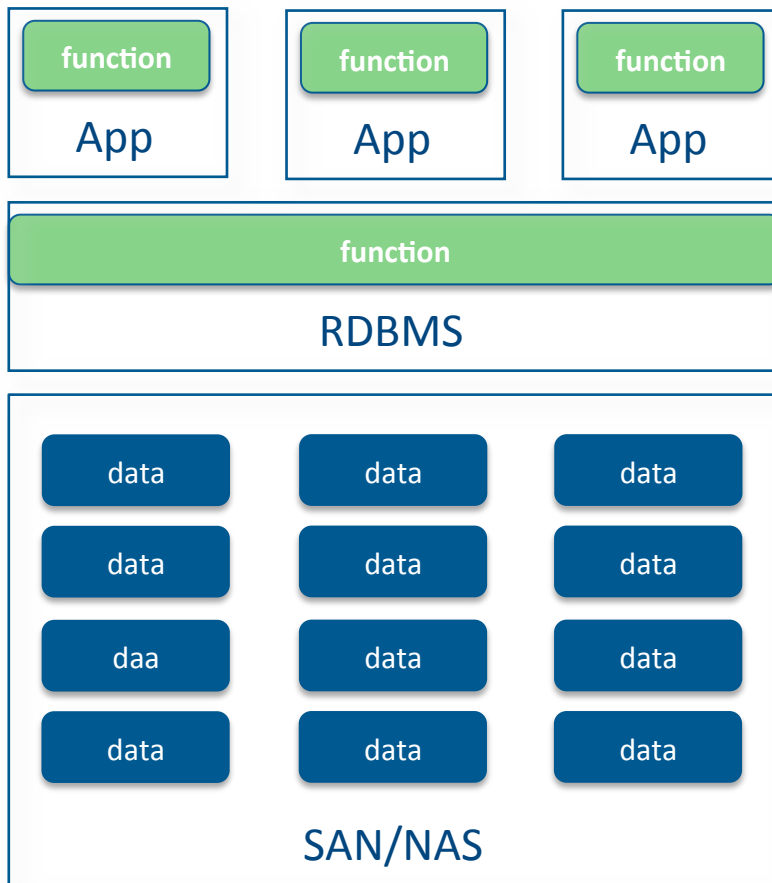
Technical innovation driven big successes
happen in systems and hardware engineering,
not application and consumer

Design making a difference:

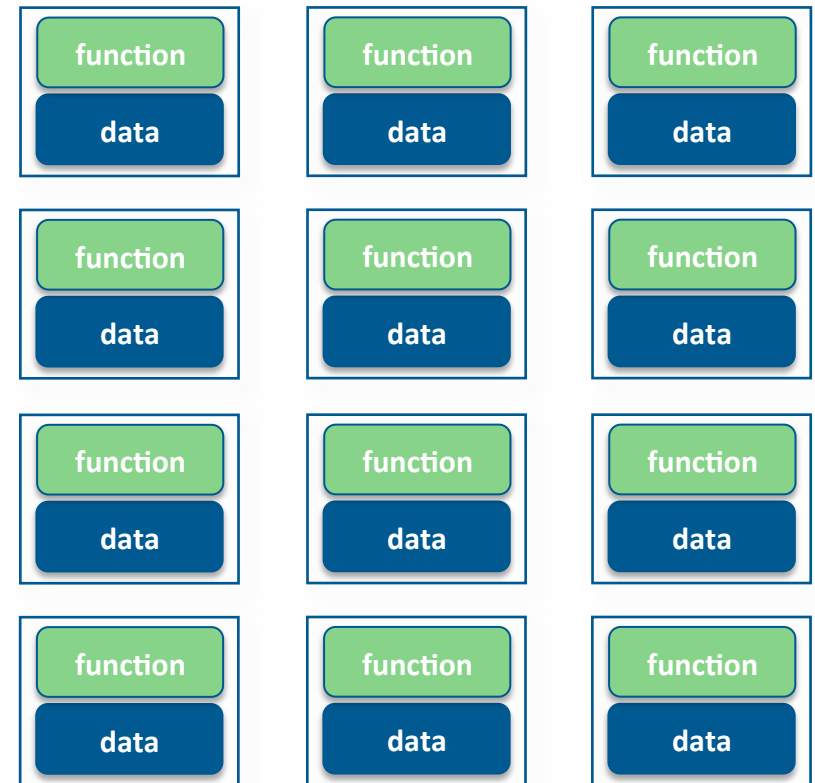
Example 1, distributed file systems in Hadoop

At 10 nodes, locality is nice. At 10,000 nodes, it's required

Traditional Architecture

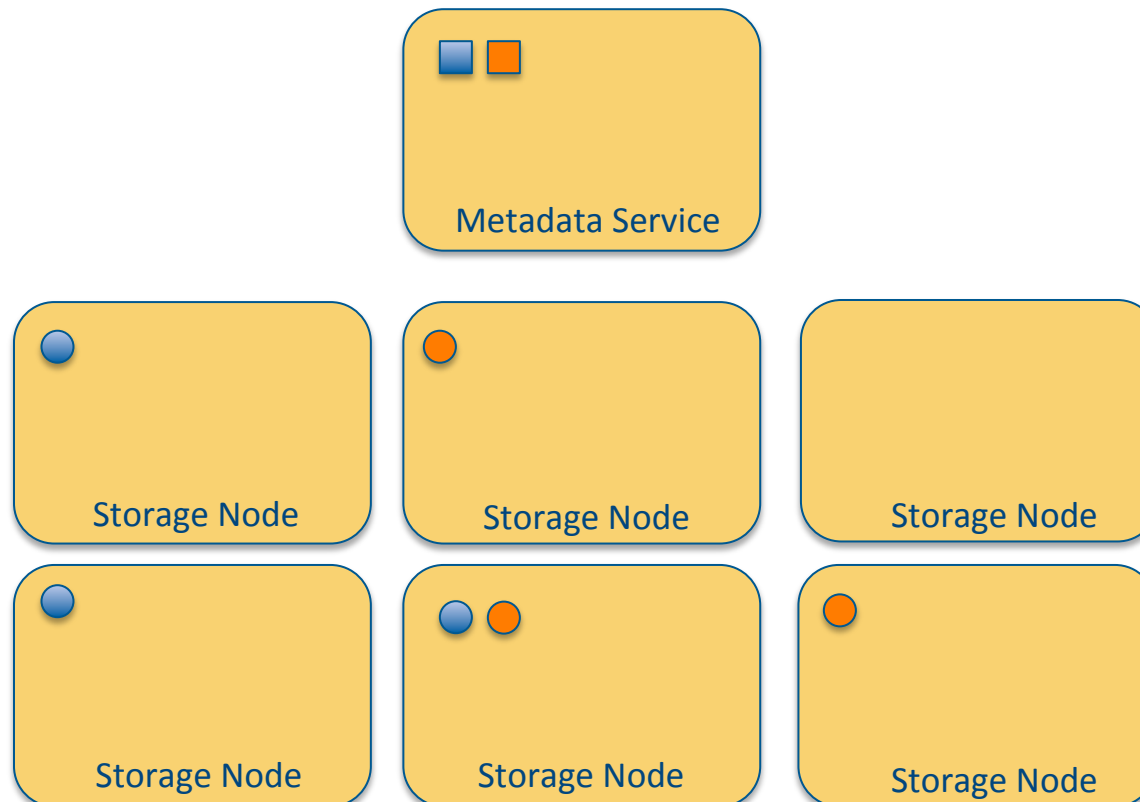


Hadoop



How do you distribute the data using DFS?

- Each file gets broken into a large number of chunks
- Chunks are distributed across the entire cluster
- Centralized metadata service

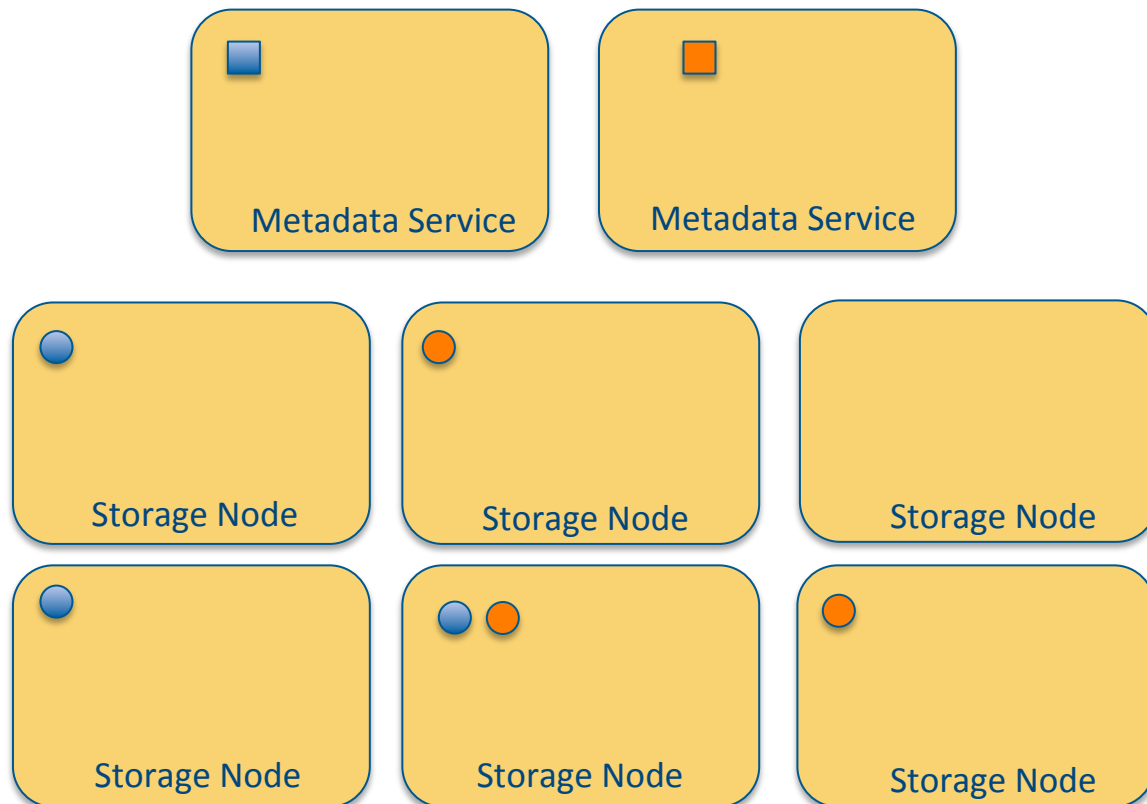


Works but creates externalities

- Centralized service limits scale
 - Largest traditional DFS clusters hover around 4000 nodes
- Number of files becomes a design concern for all applications
 - Applications running on top of DFS have to be designed to make file sizes larger
 - If they aren't large portions of cluster utilization are focused on concatenating small files into larger files
- Data recovery performance suffers due to unit of replication and system topography

One fix: federation

- Add multiple metadata service nodes that operate independently

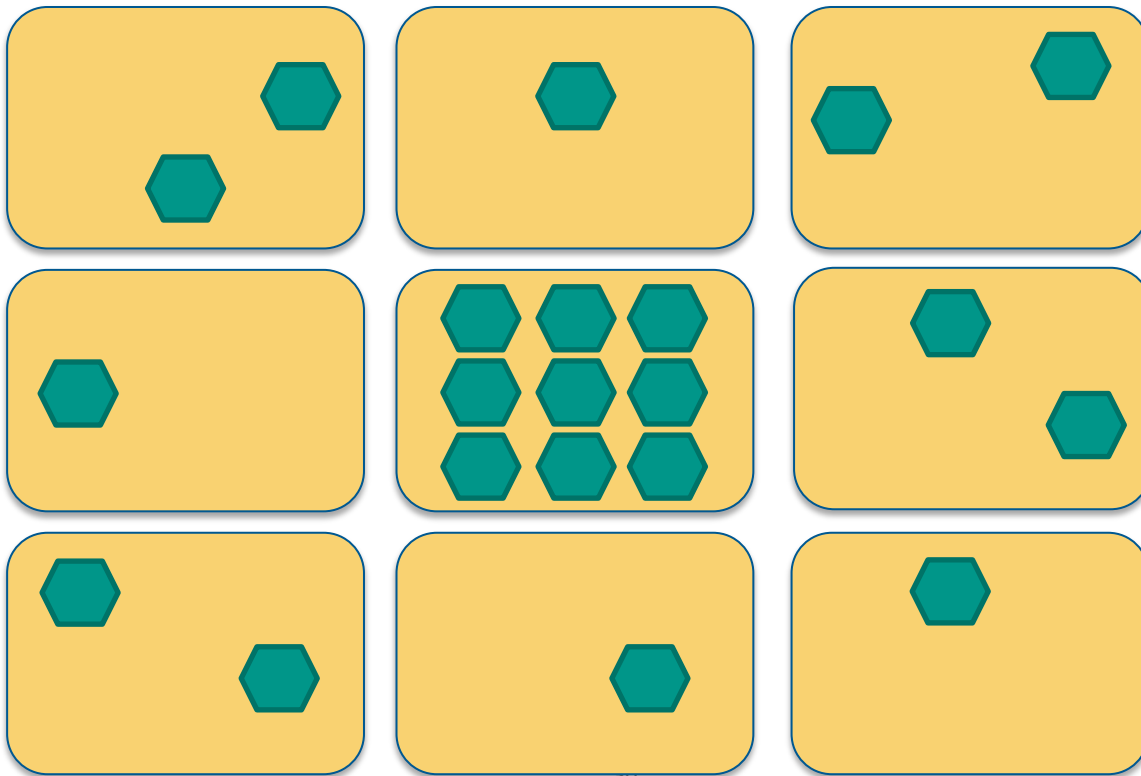


You're still fighting an uphill battle

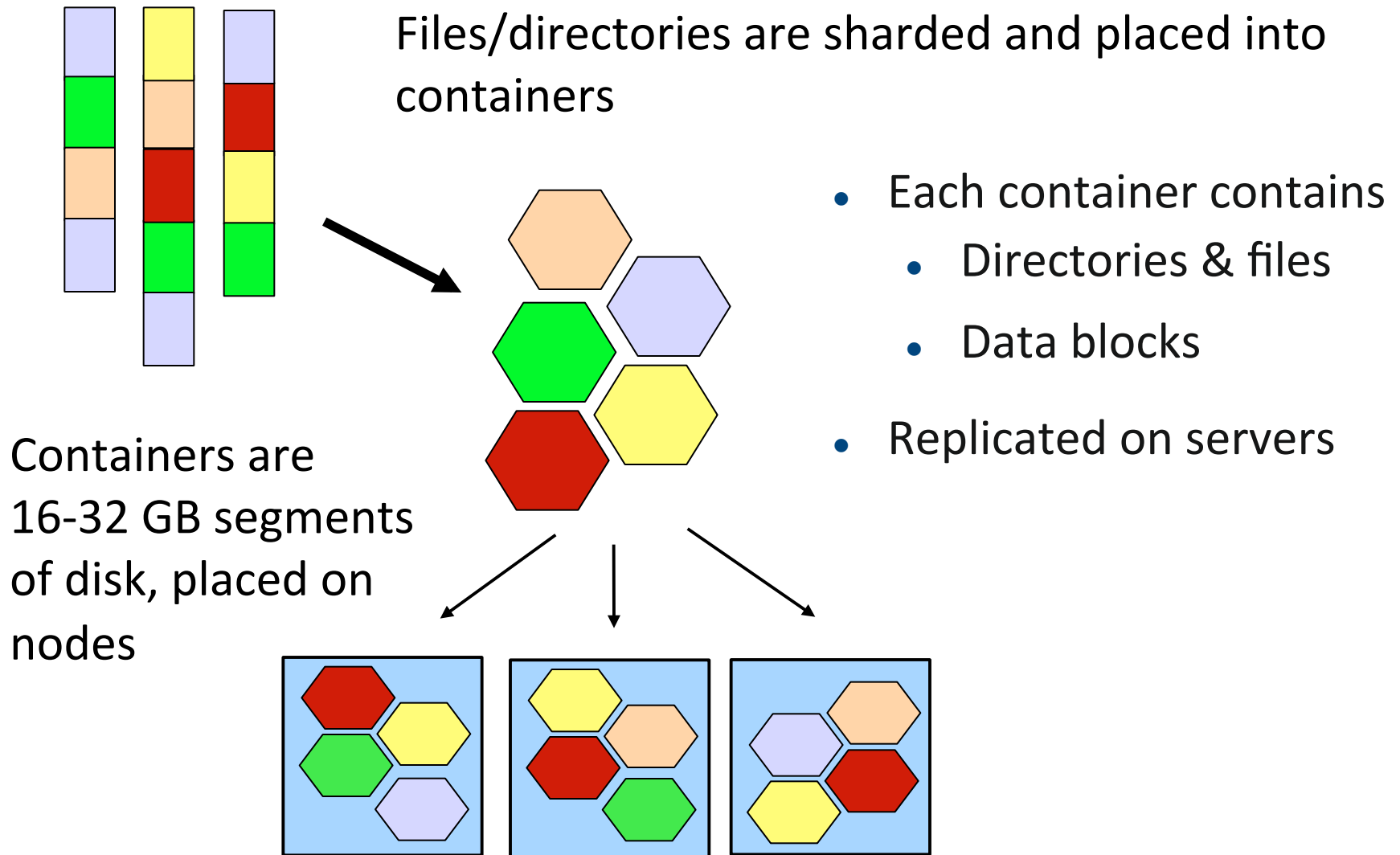
- Operational complexity continues to be challenging
 - Requirement to scale individual node types at different velocities
- Latency exists as multiple hops always required
- Maintaining high availability requires replication of each metadata service independently of data services
- Replication performance on node or disk failure continues to be problematic

Alternative approach: DFS2

- Choose different abstractions for different purposes
- Chop the data on each node to 1000s of pieces instead of millions
- Spread replicas of each container across the cluster



Store files inside large containers abstraction

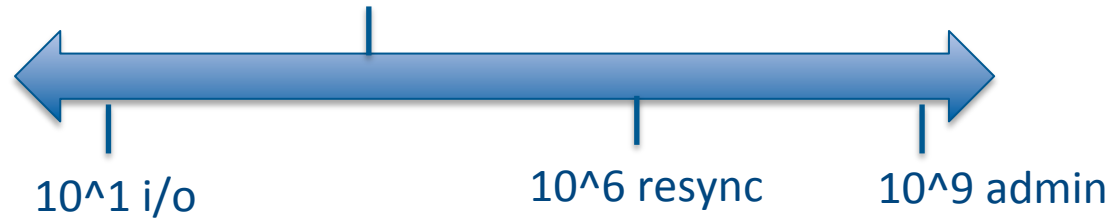


Pick the right abstraction levels

DFS1

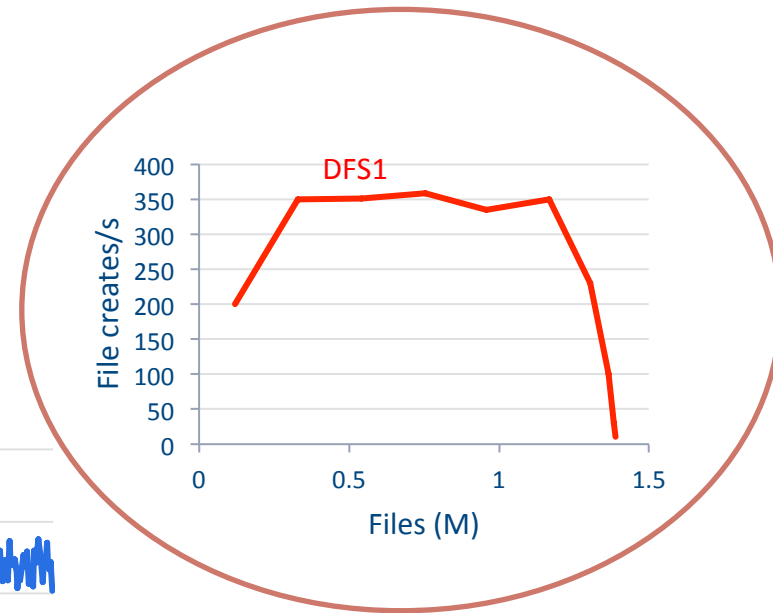
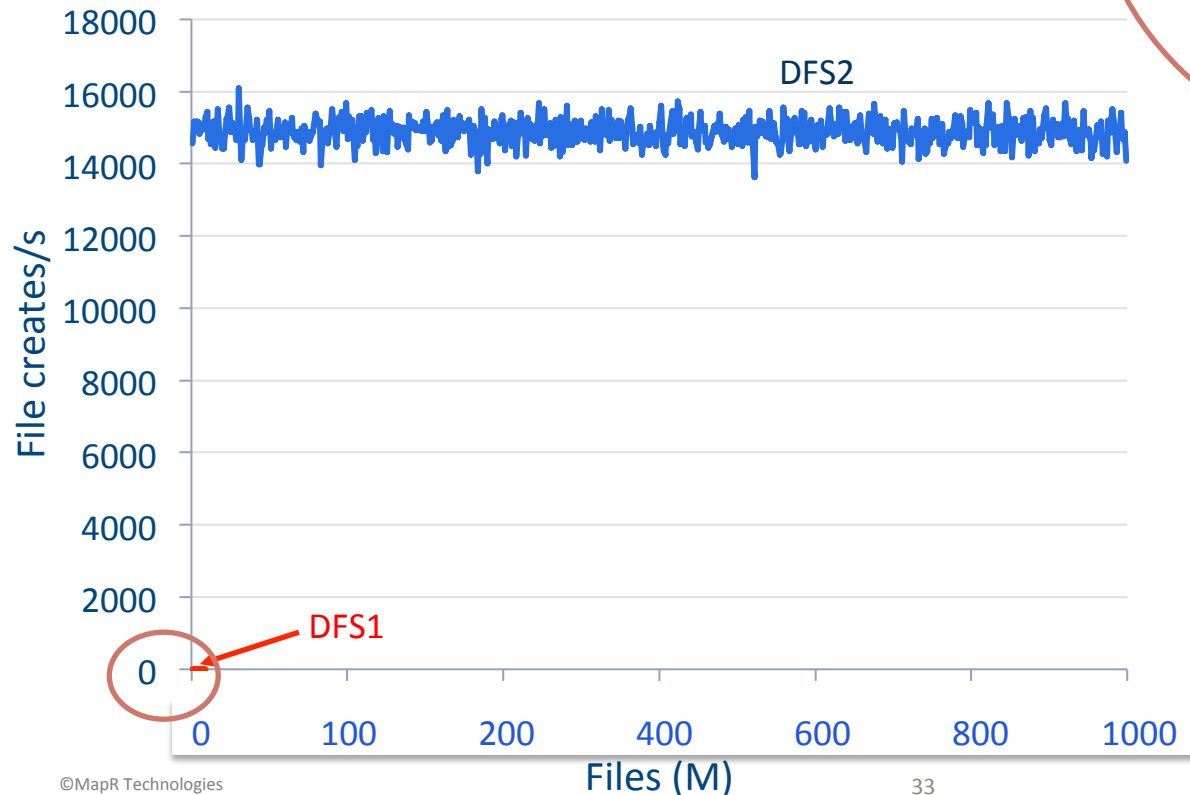
10^3 Everything

DFS2



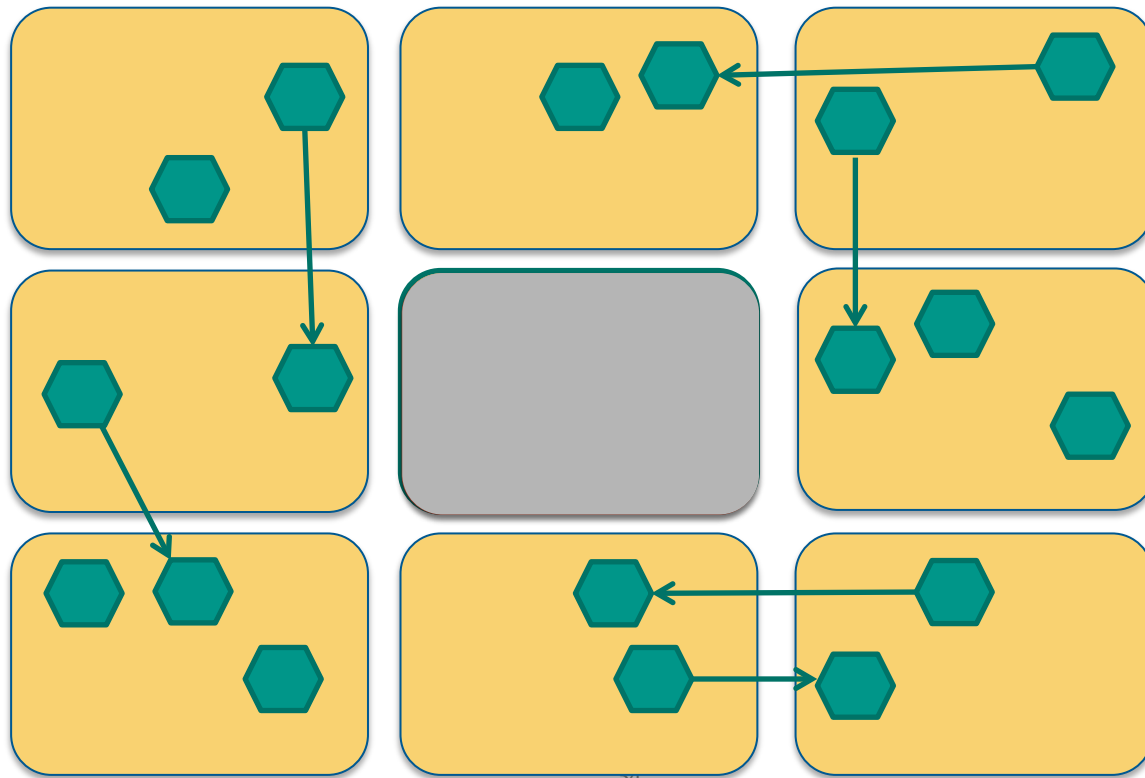
Relative Performance and Scale

	DFS1	DFS2
Rate (creates/s)	335-360	14-16K
Scale (files)	1.3M	6B



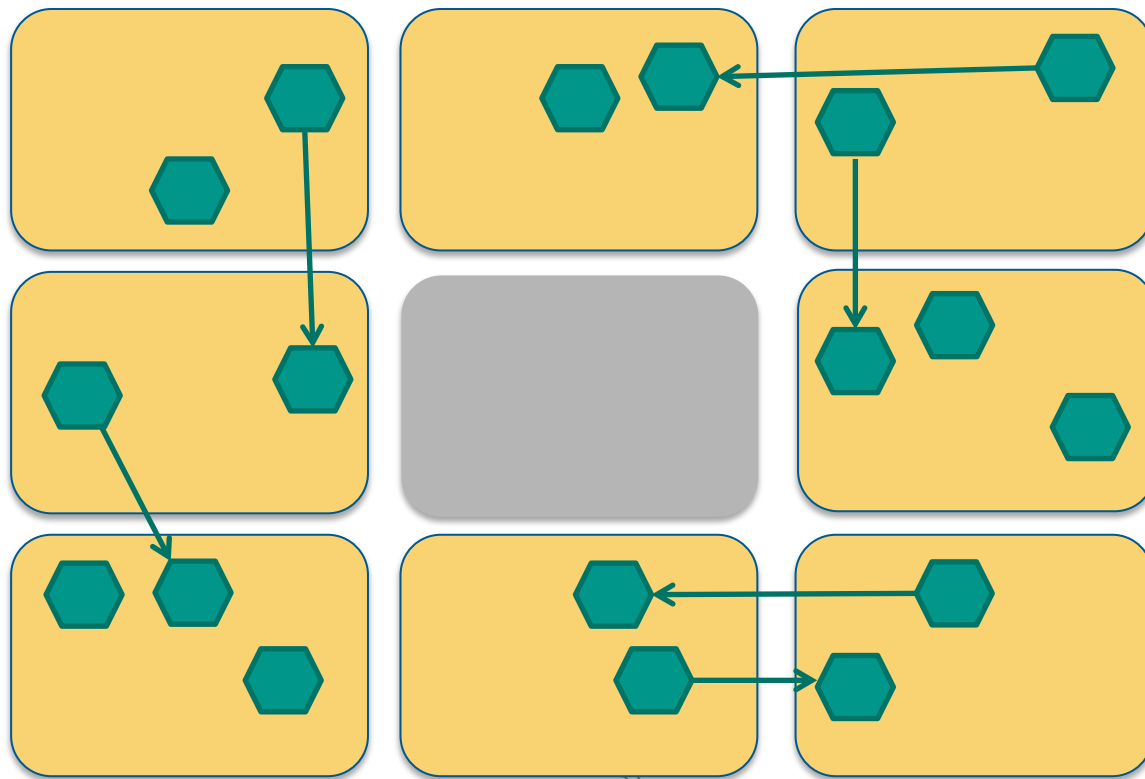
Replication Alternative: DFS2

- 100-node cluster
- Each node holds 1/100th of every node's data
- When a server dies, entire cluster re-syncs the dead node's data



DFS2 Re-sync Speed

- 99 nodes re-sync'ing in parallel
 - 99x number of drives
 - 99x number of Ethernet ports
 - 99x CPUs
- Net speed up is about 100x
 - 350 hours vs. 3.5
- MTDDL is **100x** better
- Each is re-sync'ing 1/100th of the data

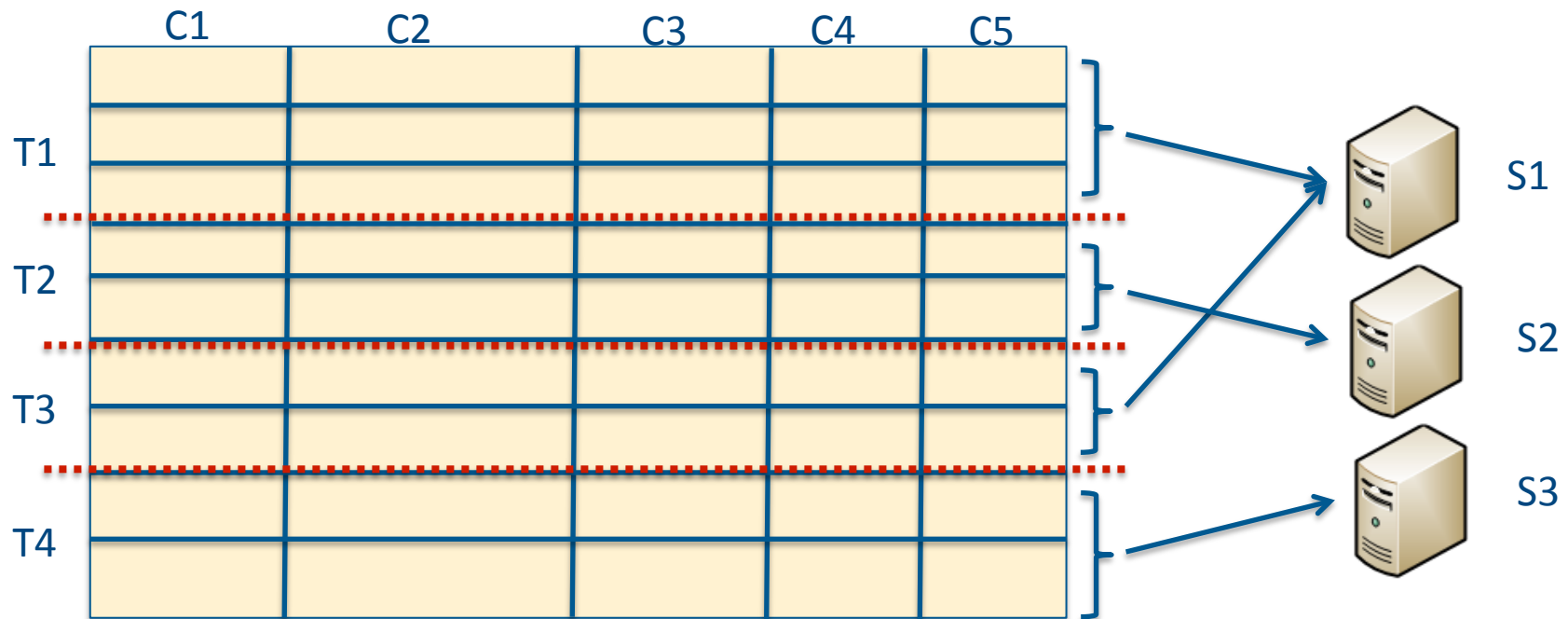


Design making a difference:

Example 2: NoSQL on Hadoop Latency

Range-based NoSQL solutions based on BigTable

- Tables are divided into key ranges (tablets)
- Each tablets is owned by one node



BigTable model is popular

- Strong consistency model
 - when a write returns, all readers will see same value
 - "eventually consistent" is often "eventually inconsistent"
- Scan works
 - does not broadcast
 - DHT ring-based NoSQL databases (eg, Cassandra, Riak) suffer on scans
- Scales automatically
 - Splits when regions become too large
 - Uses DFS to spread data, manage space
- Integrated with Hadoop processing paradigm
 - map-reduce is straightforward
- Log Structure merge is good way to manage disk workload
 - Minimize random IO

What is Log-structured merge-tree?

- Maintain log for recovery.
- Build in memory ordered representation.
- Dump to disk every so often
- Merge and rewrite every so often

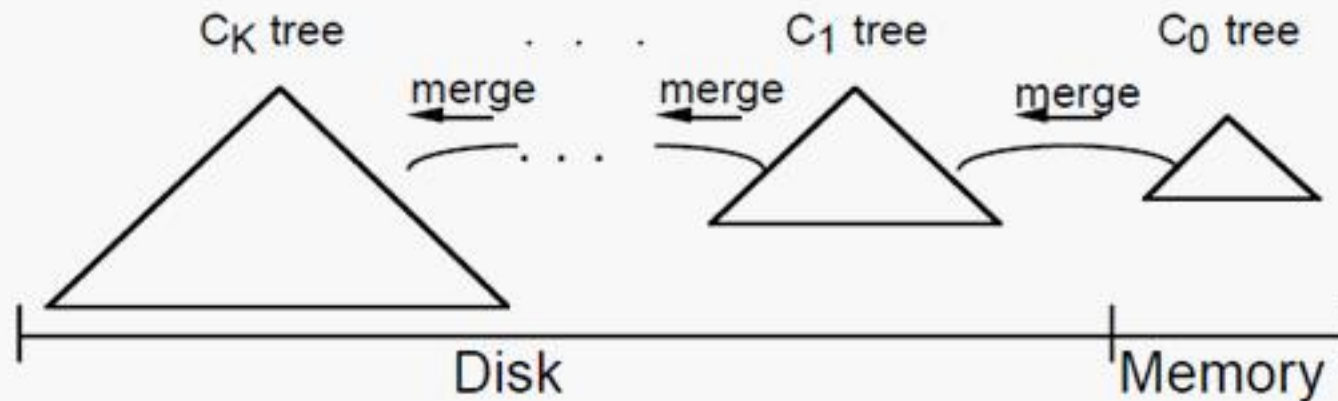


Figure 3.1. An LSM-tree of $K+1$ components

What is important in log-structured merge-trees

- Read amplification
 - Before merging, separate files that share the same key range must all be examined before a particular piece of data can be found
 - The total number of files read is the read amplification factor
 - Reduction can be done by merging a smaller set of files into larger files
- Write amplification
 - Writing the log file and then writing the first flush doubles the amount of data written to disk
 - Each initial remerge, and another write amplification factor

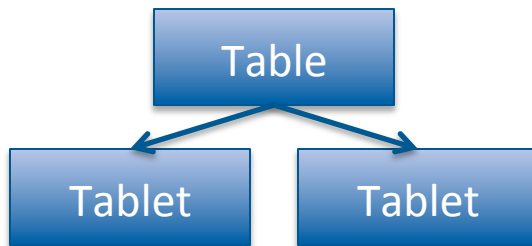
NoSQL1: Build a simple solution

- For each range, maintain a single LSM set associated with that data
- Periodically merge this LSM set to avoid excess read amplification
- Use LSM for all data structures, even metadata structures
- Build this solution on existing DFS abstractions
- Rely on implicit guarantees to ensure data locality
- Use a separate distributed system to manage cluster cohesion

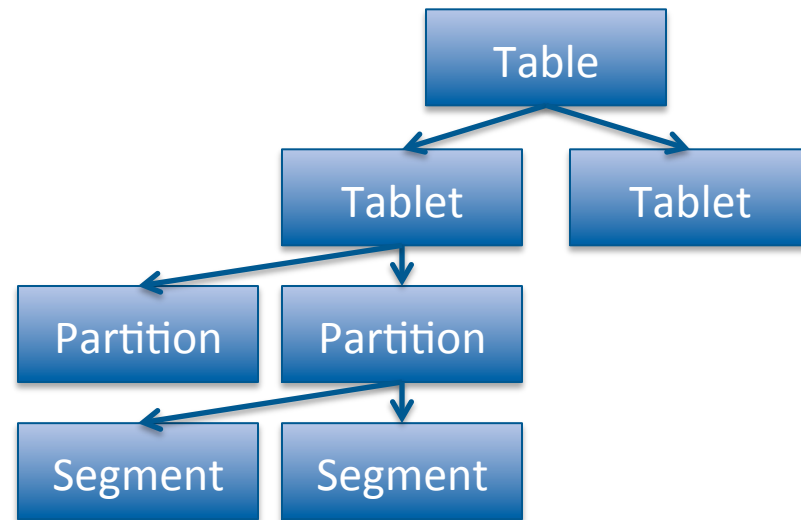
Challenges with this solution

- Three distributed systems make state and failure management exceedingly difficult to build/fix
- You either have to maintain an exceedingly large number of key ranges (problematic), or you have a huge files associated with each LSM set
- Large files drive increase read and write amplification
- Implicit locality guarantees result in substantial remote reads from DFS when responding to queries
- Multiple layers of abstraction increase resource consumption and overall response time

NoSQL2: Create additional levels of abstraction

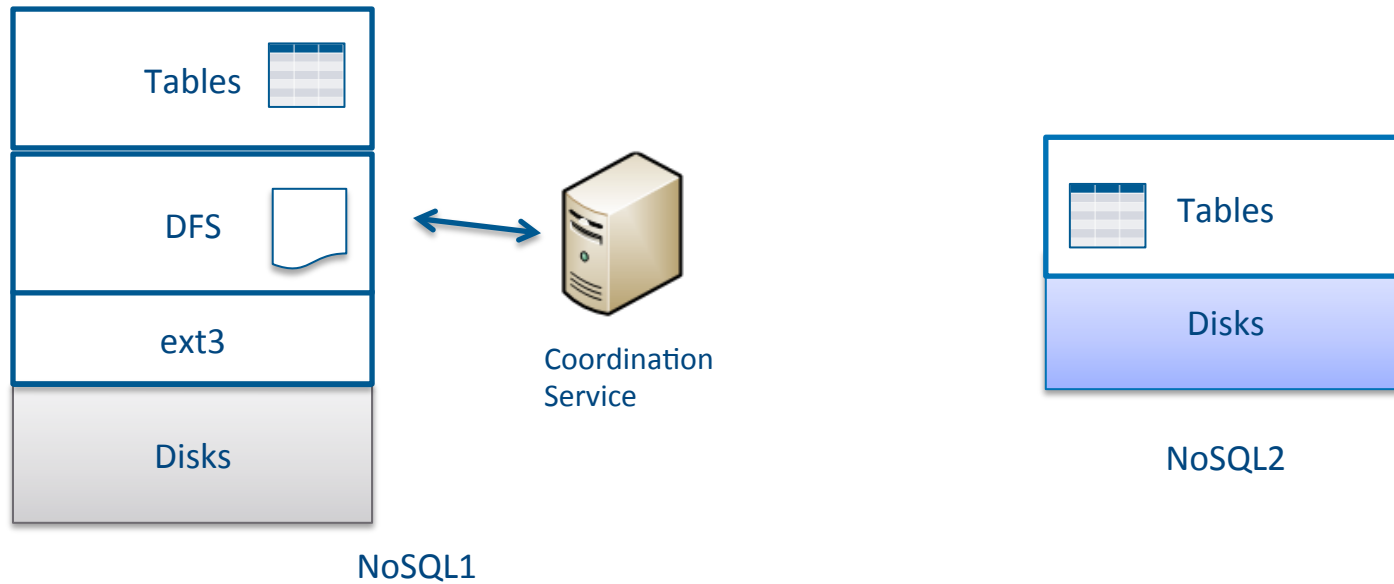


NoSQL1



NoSQL2

Simplify the stack

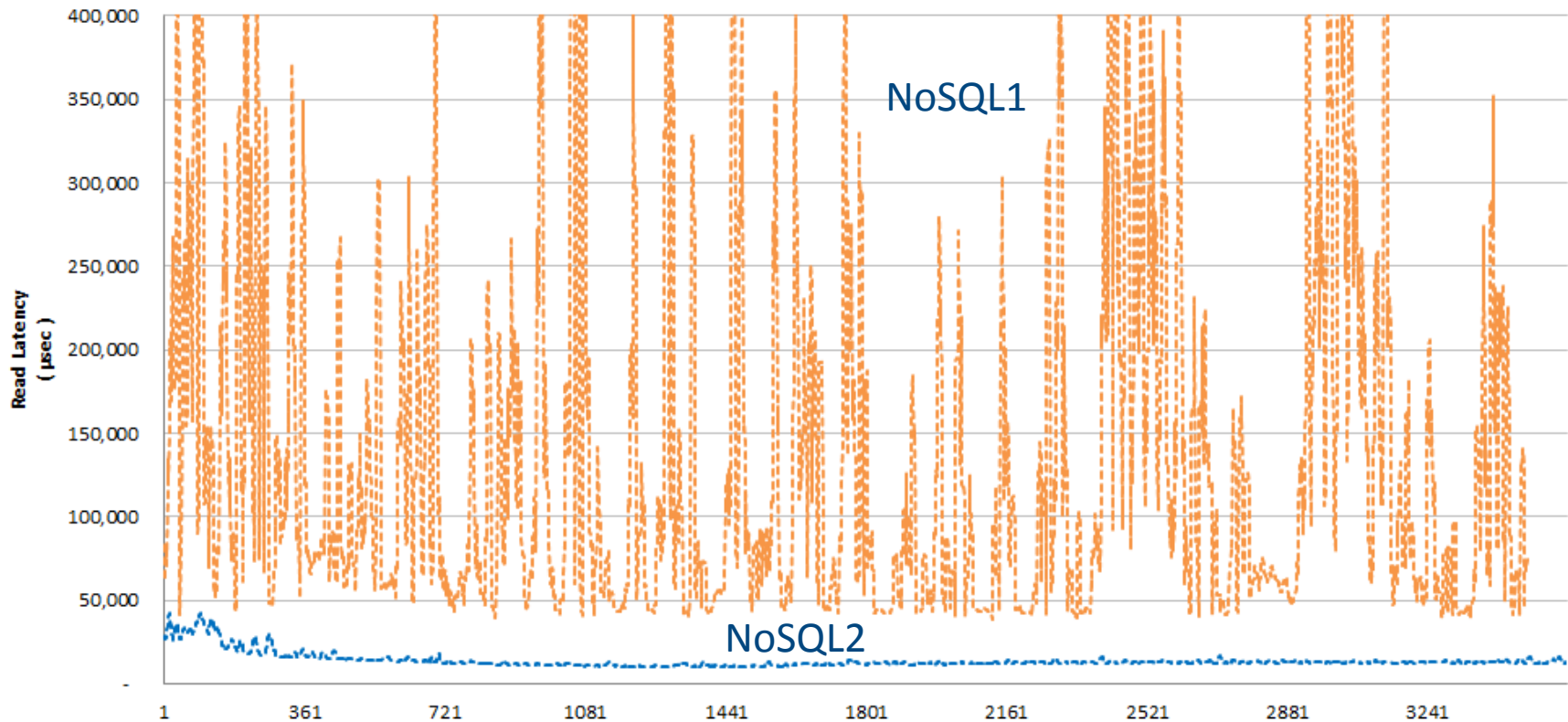


Outcome

YCSB Mixed (50%Update-50%Read) Test (10Nodes)

Source: 2TB (1K RowSize)

Read Latency ONLY : 10-sec Moving Average & y-Axis Cap=400msec



Design making a difference?

What I'm working on now... Apache Drill

Apache Drill Overview

Interactive analysis of Big Data using standard SQL

Fast

- Low latency queries
- Columnar execution
- Complement native interfaces and MapReduce/Hive/Pig

Apache Drill

Interactive queries
Data analyst
Reporting
100 ms-20 min

Open

- Community driven open source project
- Under Apache Software Foundation
- No vendor lock-in

Modern

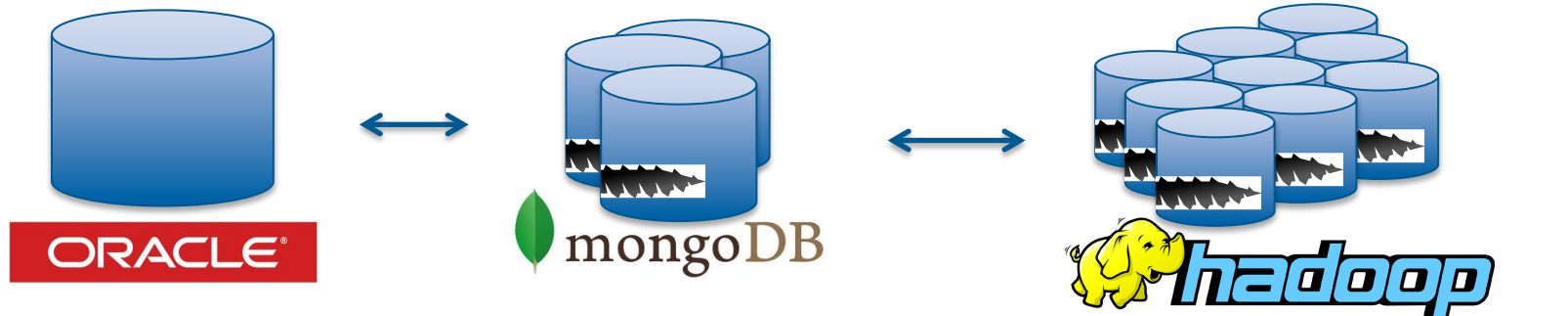
- Dynamic and static schemas
- Centralized schemas and self-describing data
- Nested/hierarchical data support
- Standard ANSI SQL:2003
- Strong extensibility

MapReduce
Hive
Pig

Data mining
Modeling
Large ETL
20 min-20 hr

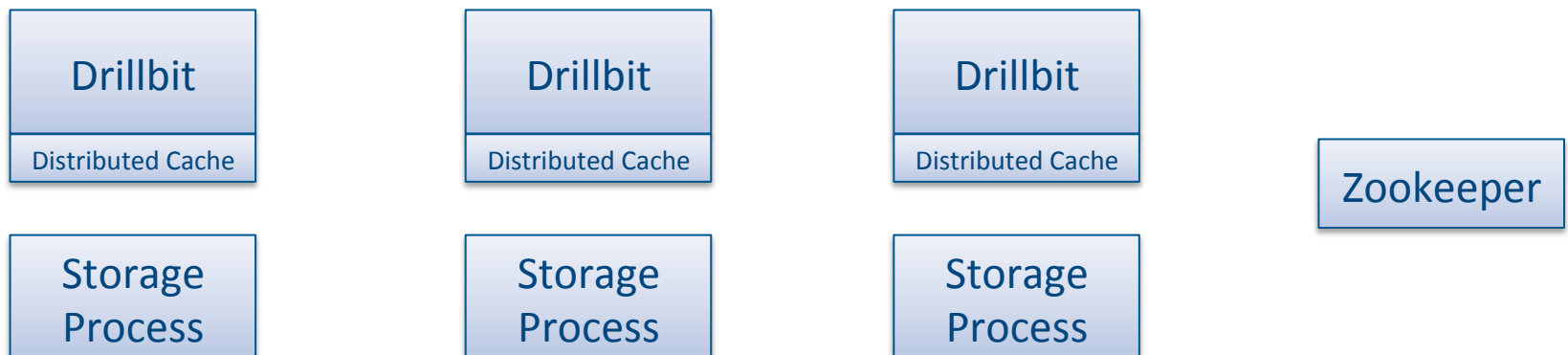
How Does It Work?

- Drillbits run on each node, designed to maximize data locality
- Drill includes a distributed execution environment built specifically for distributed query processing.
- Coordination, query planning, optimization, scheduling, and execution are distributed

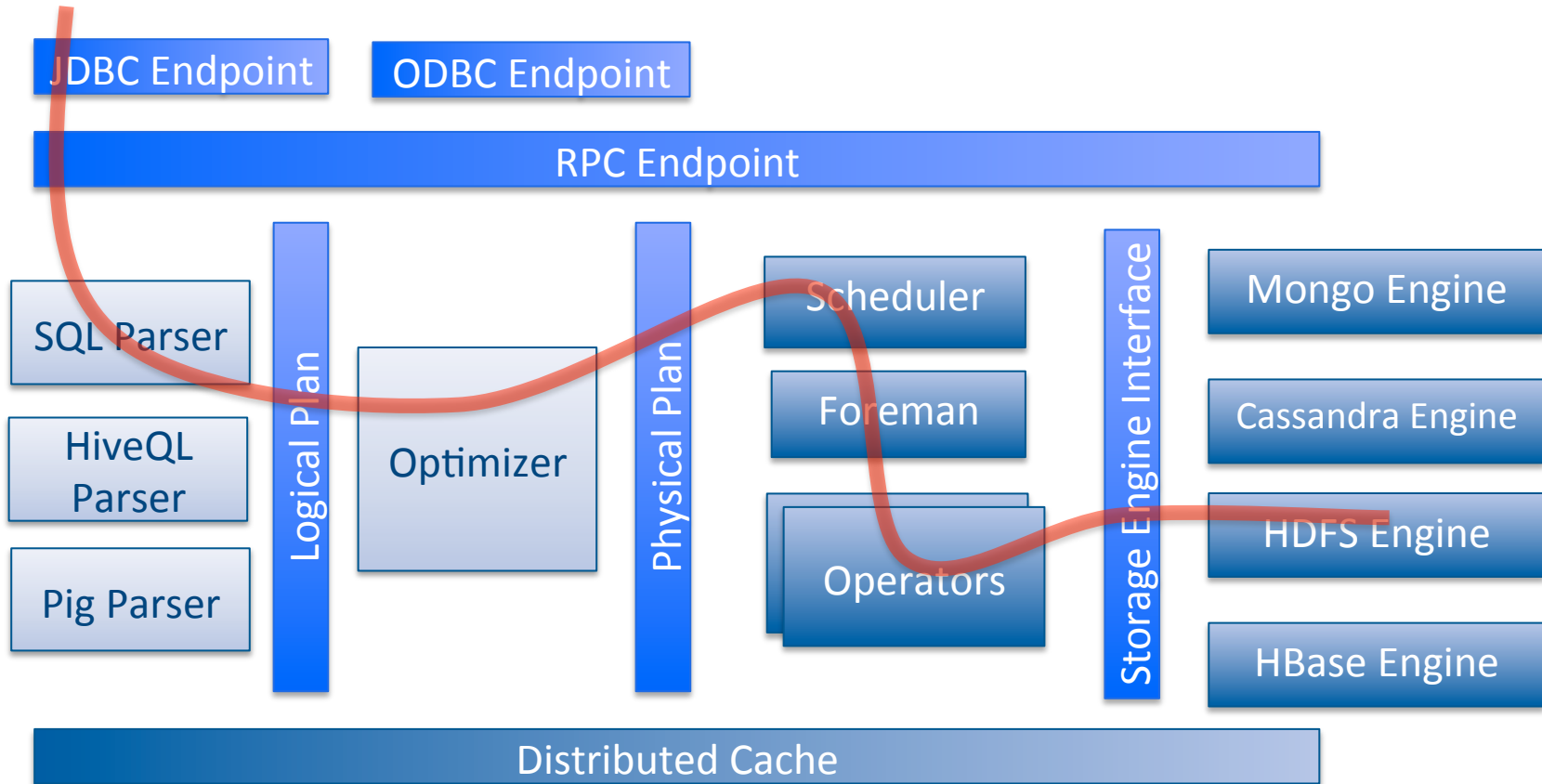


High Level Architecture

- Any Drillbit can act as endpoint for particular query.
- Zookeeper maintains ephemeral cluster membership information only
- Small distributed cache utilizing embedded Hazelcast maintains information about individual queue depth, cached query plans, metadata, locality information, etc.
- Originating Drillbit acts as foreman, driving all execution for a particular query, scheduling based on priority, queue depth and locality information.
- Drillbit data communication is pipelined streaming and avoids any serialization/deserialization



Drillbit Modules



Start with the core data path

- What types of problems plague big data Java applications
 - Large heaps and garbage collection
 - Pauses and cluster coordination are especially problematic
 - Overhead moving back and forth from native code
 - Excessive copying
- Focus on simple use cases: read and count, read and filter, read and aggregate, read and sort

Optimistic Execution

- With a short time horizon, failures infrequent
 - Don't spend energy and time creating boundaries and checkpoints to minimize recovery time
 - Rerun entire query in face of failure
- No barriers
- No persistence unless memory overflow
- Don't delay network IO

Why Not Leverage MapReduce?

- Scheduling Model
 - Coarse resource model reduces hardware utilization
 - Acquisition of resources typically takes 100's of millis to seconds
- Resource distribution and startup
 - JVM start up takes time
 - Memory management across JVMs inelegant
 - Even with reuse, jar copying takes time
- Barriers
 - All maps must complete before reduce can start
 - In chained jobs, one job must finish entirely before the next one can start
- Persistence and Recoverability
 - Data is persisted to disk between each barrier
 - Serialization and deserialization are required between execution phase

Comparison for Aggregate Query

MapReduce

- Full sort must occur
- Aggregations don't start until all data is sorted
- Assignments for reduce locations aren't made until map tasks partially completed

Drill

- Sort not always required
- Data pipelined between first and second fragments
- Aggregations start as soon as first collection of records are available
- Assignments are made at initial query time and data is streamed immediately to destination when available

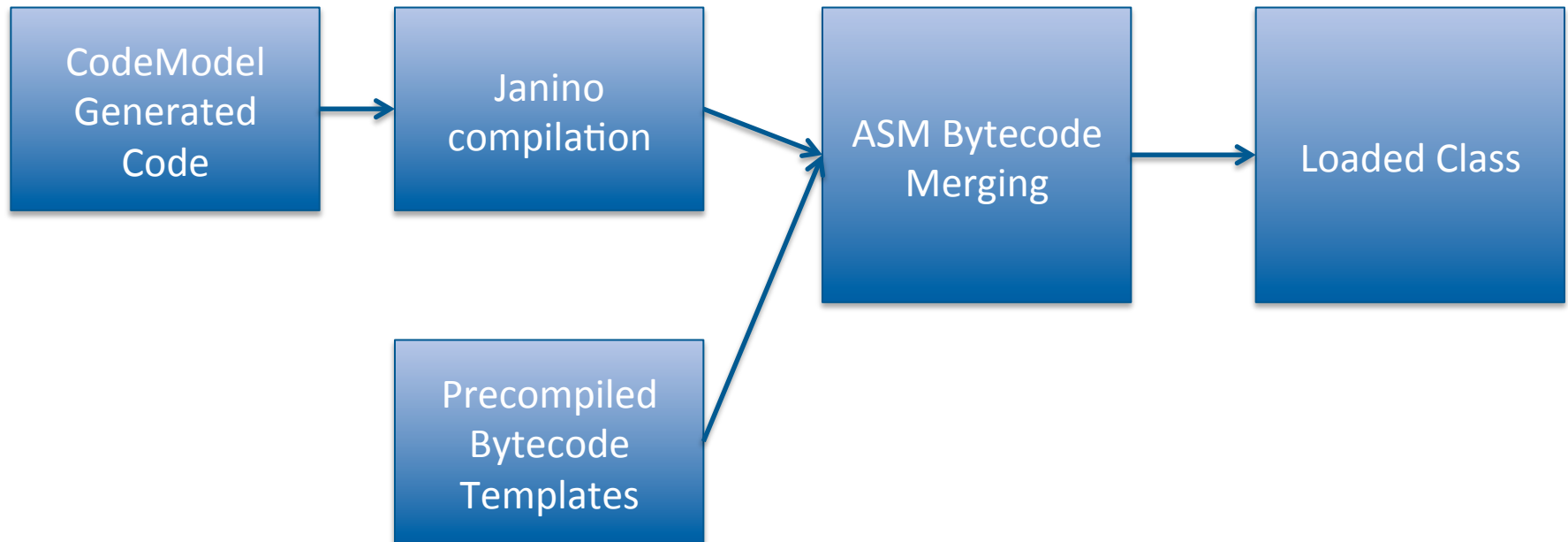
Runtime Compilation

- Give JIT help
- Avoid virtual method invocation
- Avoid heap allocation and object overhead
- Minimize memory overhead

First, a couple tools

- Janino
 - Java based Java compiler
 - Supports most of 1.5 except generics and annotations
 - Much faster than `javax.tools.JavaCompiler`
- CodeModel
 - Builder model for Java code generation
 - Simplifies generation code and maintenance
- ASM
 - Bytecode traversal and manipulation

Runtime Bytecode Merging



- CodeModel to generate runtime specific blocks
- Janino to generate runtime bytecode
- Precompiled bytecode templates
- Use ASM package to merge the two distinct classes into one runtime class

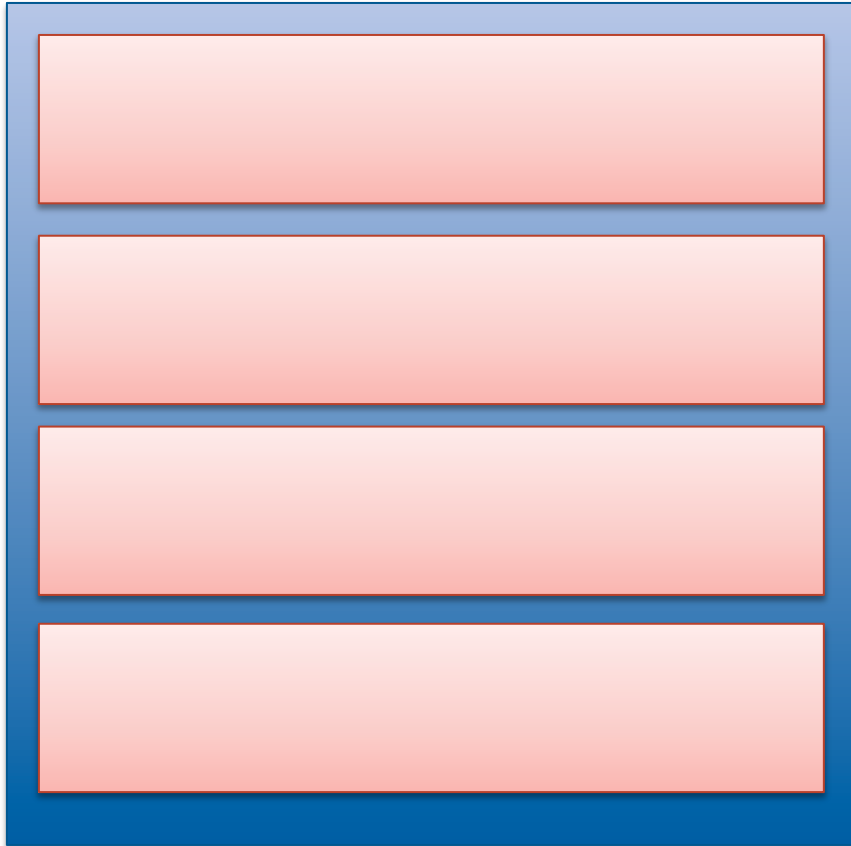
Runtime Function Compilation

- Function implementations are a combination of types, annotations and execution blocks
- Designed specifically designed to minimize execution overhead
- All expressions across an operator are compiled to a single method
- That method is typically by JIT inlined

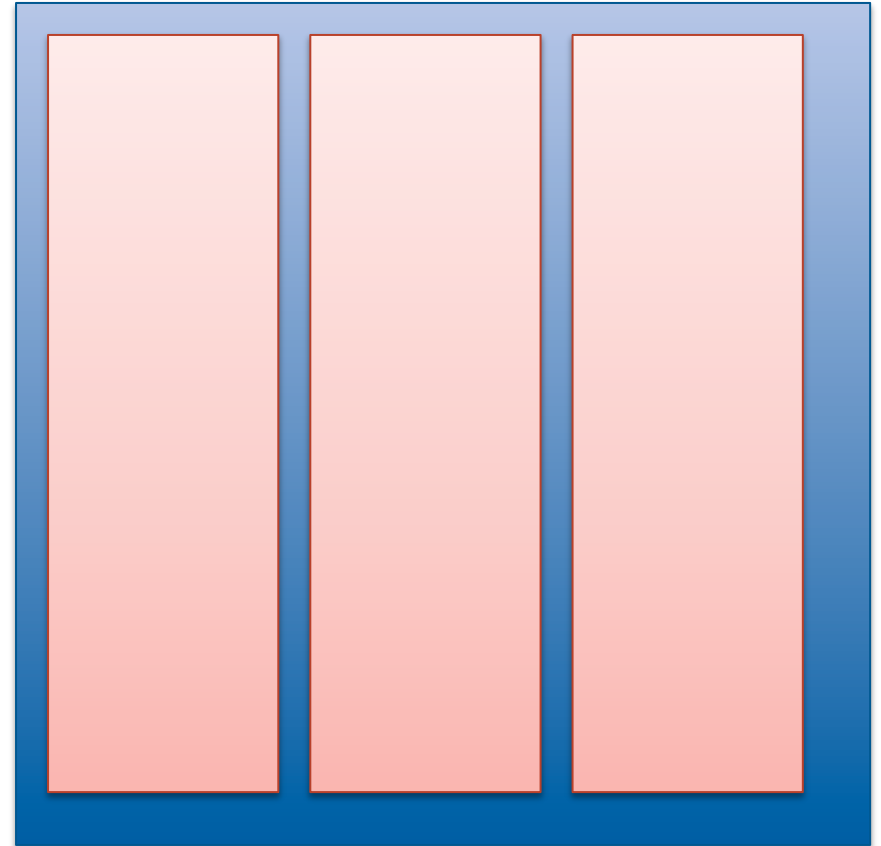
Runtime Operator Compilation

- Operator state broken into key stages
- Stages typically broken into individual methods
- Abstract signatures defined in precompiled bytecode templates
- Stage methods typically inlined by JVM
- Methods built using CodeModel and expression compilation

Record versus Columnar Representation



Record



Column

Data Format Example

Donut	Price	Icing
Bacon Maple Bar	2.19	[Maple Frosting, Bacon]
Portland Cream	1.79	[Chocolate]
The Loop	2.29	[Vanilla, Fruitloops]
Triple Chocolate Penetration	2.79	[Chocolate, Cocoa Puffs]

Record Encoding

Bacon Maple Bar, 2.19, Maple Frosting, Bacon, Portland Cream, 1.79, Chocolate
The Loop, 2.29, Vanilla, Fruitloops, Triple Chocolate Penetration, 2.79, Chocolate, Cocoa Puffs

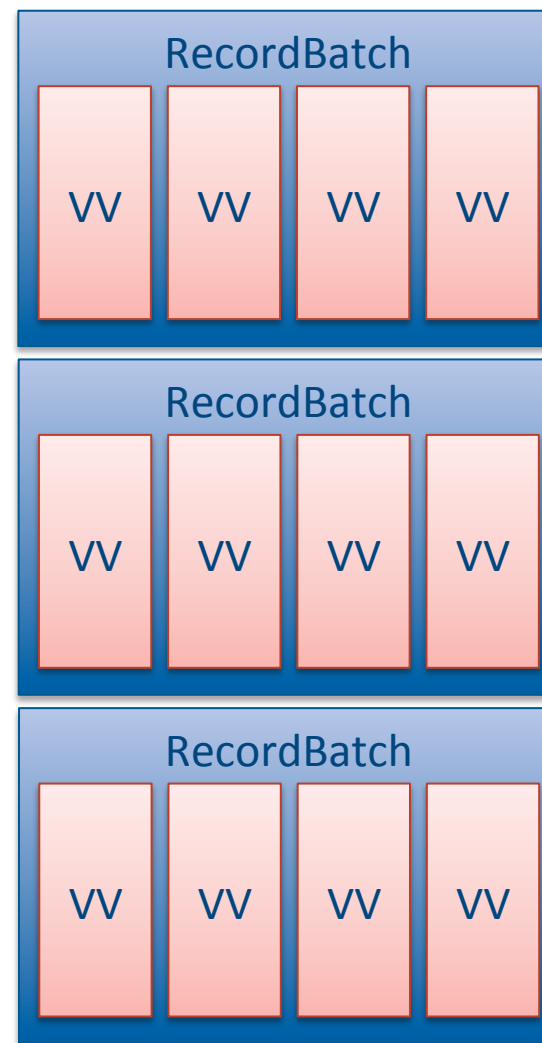
Columnar Encoding

Bacon Maple Bar, Portland Cream, The Loop, Triple Chocolate Penetration
2.19, 1.79, 2.29, 2.79

Maple Frosting, Bacon, Chocolate, Vanilla, Fruitloops, Chocolate, Cocoa Puffs

Record Batch

- Drill optimizes for BOTH columnar STORAGE and Execution
- Record Batch is unit of work for the query system
 - Operators always work on a batch of records
- All values associated with a particular collection of records
- Each record batch must have a single defined schema
 - Possibly includes fields that have embedded types if you have a heterogeneous field
- Record batches are pipelined between operators and nodes
- No more than 65k records
- Target single L2 cache (~256k)
- Operator reconfiguration is done at RecordBatch boundaries



Strengths of RecordBatch + ValueVectors

- RecordBatch clearly delineates low overhead/high performance space
 - Record-by-record, avoid method invocation
 - Batch-by-batch, trust JVM
- Avoid serialization/deserialization
- Off-heap means large memory footprint without GC woes
- Full specification combined with off-heap and batch-level execution allows C/C++ operators as necessary
- Random access: sort without copy or restructuring

Memory Design

- ValueVectors contain multiple individual off-heap buffers
- Leverage Netty's ByteBuf abstraction and native byte ordering
- Memory pooling based on Java implementation of Facebook's Jemalloc
 - Allocation/deallocation are managed via reference counting
- Strongly defined buffer ownership semantics

Serialization/Deserialization

- Data stays off heap in ValueVectors
- Heap maintains operational metadata
- Serialization
 - Small metadata protobuf combined with ordered list of direct buffers
 - Minimize context switching and copies using gathering writes
- Deserialization
 - Incoming buffer slicing minimizing data copying

Selection Vector

- Includes particular records from consideration by record batch index
- Avoids early copying of records after applying filtering
 - Maintains random accessibility
- All operators need to support SelectionVector access

Selection Vector	Donut	Price	Icing
0	Bacon Maple Bar	2.19	[Maple Frosting, Bacon]
3	~~Portland Cream~~	~~1.79~~	~~{Chocolate}~~
	~~The Loop~~	~~2.29~~	~~{Vanilla, Fruitloops}~~
	Triple Chocolate Penetration	2.79	[Chocolate, Cocoa Puffs]

Selection Vector Types

Selection Vector

- Identify records within a batch that should be considered for later operators
- Reduces copies when subsequent operators will already copy
- Currently removed before network transfer
 - May optimize based on selectivity in future

Hyper Vector

- Identify records across multiple batch that should be considered
- Enables pointer sort without data copies

Additional VV Strengths

- Vectorized Operations
 - Scans (partial Parquet already)
 - Null check elimination and word-sized operations
 - Implicit data type simplification
 - Filtering
 - Aggregations and other operators
- Fixed bit type
 - Enables late dictionary materialization
 - Enables direct bit-packed manipulation
- Reference redirection
 - Modeled after Lucene's object reuse pattern and AttributeSource concept
 - Simplifies runtime compilation and memory management

Example: Bitpacked Dictionary VarChar Sort

- Dataset:
 - Dictionary: [Rupert, Bill, Larry]
 - Values: [1,0,1,2,1,2,1,0]
- Normal Work:
 - Decompress & store: Bill, Rupert, Bill, Larry, Bill, Larry, Bill, Rupert
 - Sort: ~24 comparisons of variable width strings (requiring length lookup and check during comparisons)
- Optimized Work
 - Sort Dictionary: {Bill: 1, Larry: 2, Rupert: 0}
 - Sort bitpacked values
 - Work: max 3 string comparisons, ~24 comparisons of fixed-width dictionary bits
 - Data in 16 bits as opposed 368/736 for UTF8/16

What are the results?

- Time will tell, ping me in six months

Closing Thoughts

- As you build a company (or work for other companies)
 - Evaluate companies based on customer demand and market potential
 - Evaluate work based on technology interest
- Don't confuse the two and make sure you balance both

In Review

- Big data is big
- The internet and Hadoop make the kernel
- If you want to make big money by being a good engineer, be a systems engineer because that's where design matters
- But only when it focuses on customer problems in big markets