Operating Systems and The Cloud, Part II: Search => Cluster Apps => Scalable Machine Learning

David E. Culler

CS162 – Operating Systems and Systems Programming

Lecture 40

December 3, 2014

Proj: CP 2 today
Reviews in R&R
Mid3 12/15

The Data Center as a System

- **Clusters became THE architecture for large scale internet services**
  - Distribute disks, files, I/O, net, and compute over everything
  - Massive AND Incremental scalability
- **Search Engines the initial "Killer App"**
- **Multiple components as Cluster Apps**
  - Web crawl, Index, Search & Rank, Network, …
- **Global Layer as a Master/Worker pattern**
  - GFS, HDFS
- **Map Reduce framework address core of search on massive scale – and much more**
  - Indexing, log analysis, data querying
  - Collating, inverted indexes : map(k,v) => f(k,v),(k,v)
  - Filtering, Parsing, Validation
  - Sorting

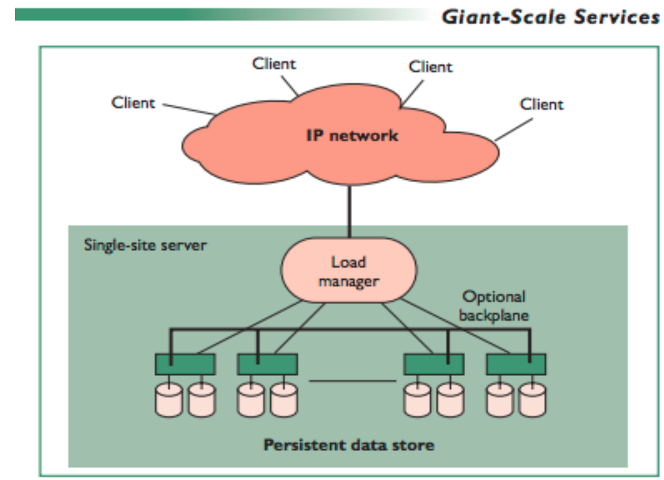
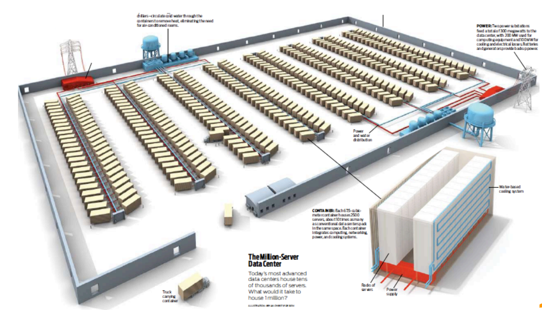


Figure 1. The basic model for giant-scale services. Clients connect via the Internet and then go through a load manager that hides down nodes and balances traffic.



[Lessons from Giant-Scale Services, Eric Brewer, IEEE Computer, Jul 2001](#)

Research …

research.google.com/archive/mapreduce-osdi04…

Home Prev Next 1

MapReduce:
Simplified Data Processing on Large Clusters

Jeff Dean, Sanjay Ghemawat
Google, Inc.

research.google.com/archive/mapreduce-osdi04-slides/index-auto-0031.html

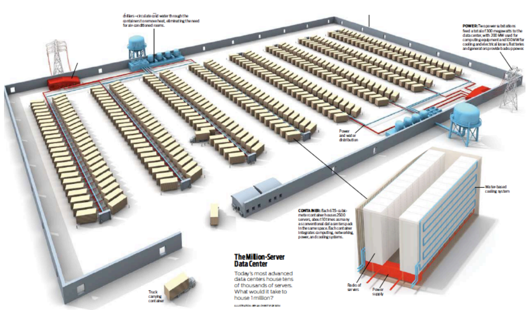
Home Prev Next

Related Work

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]

The Data Center as a System

- **Clusters became THE architecture for large scale internet services**
  - Distribute disks, files, I/O, net, and compute over everything
  - Massive AND Incremental scalability
- **Search Engines the initial "Killer App"**
- **Multiple components as Cluster Apps**
  - Web crawl, Index, Search & Rank, Network, …
- **Global Layer as a Master/Worker pattern**
  - GFS, HDFS
- **Map Reduce framework address core of search on massive scale – and much more**
  - Indexing, log analysis, data querying
  - Collating, inverted indexes : map(k,v) => f(k,v),(k,v)
  - Filtering, Parsing, Validation
  - Sorting,
  - Graph Processing (???) – page rank,
  - Cross-correlation (???)
  - Machine Learning (???)



Time Travel

- **It's not just storing it, it's what you do with the data**

AMPLab Unification Philosophy

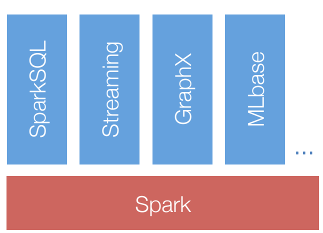
Don't specialize MapReduce – Generalize it!

Two additions to Hadoop MR can enable all the models shown earlier!

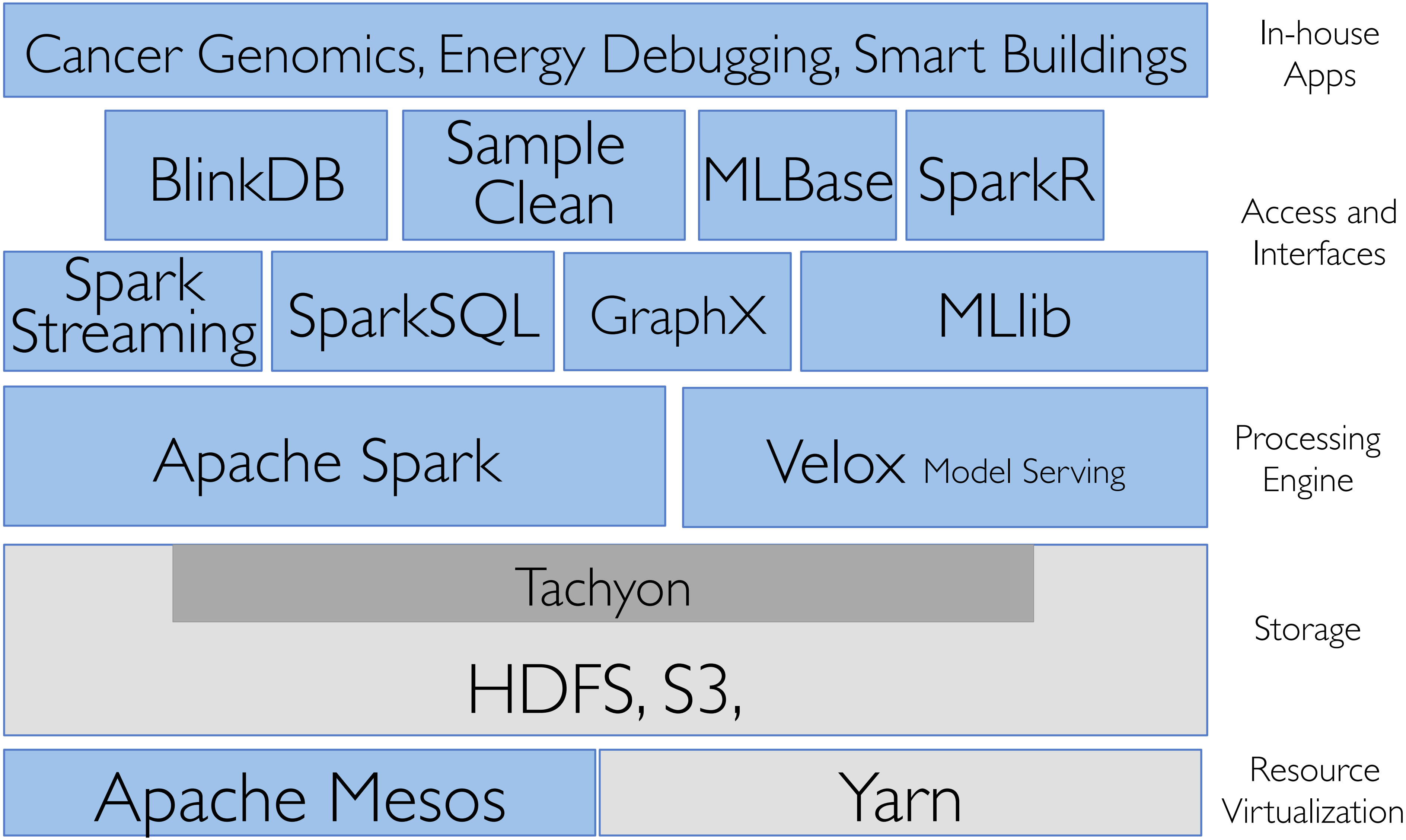
1. General Task DAGs
2. Data Sharing

For Users:

Fewer Systems to Use
Less Data Movement

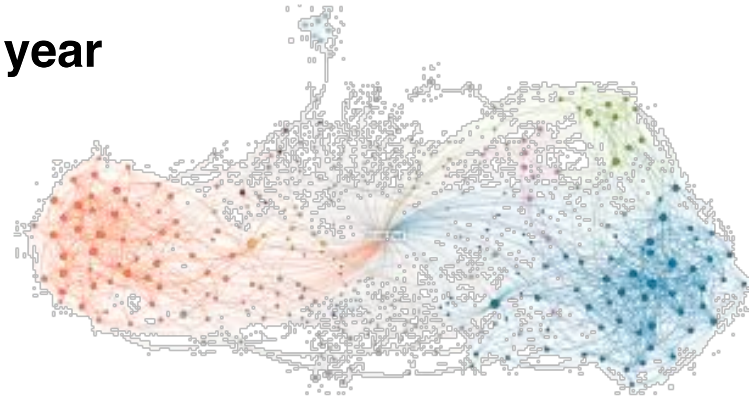


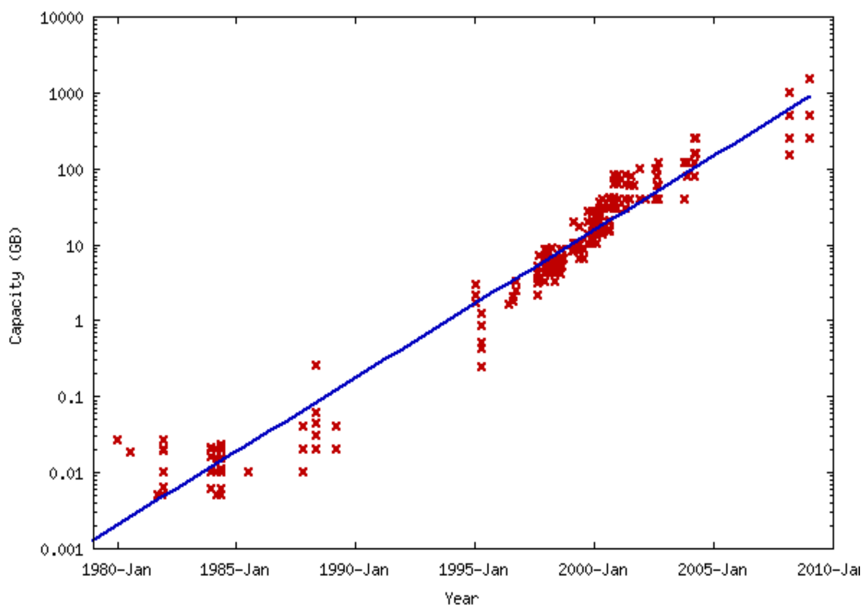
Berkeley Data Analytics Stack

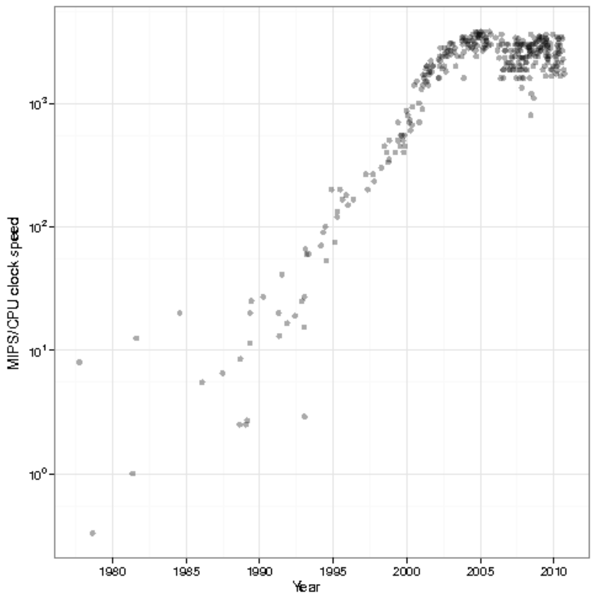


The Data Deluge

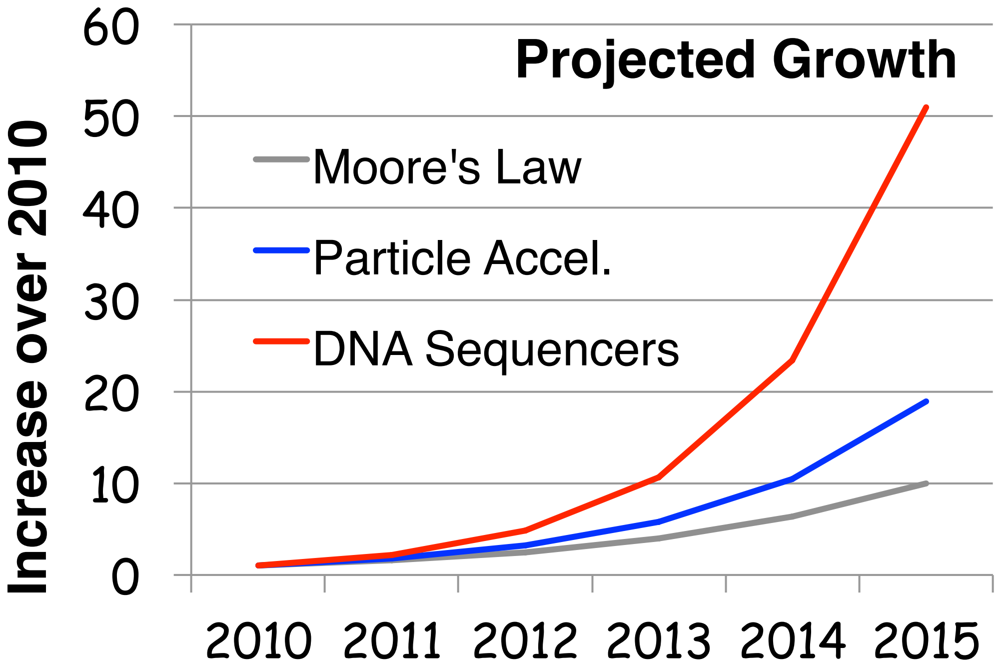
- Billions of users connected through the net
  - WWW, Facebook, twitter, cell phones, …
  - 80% of the data on FB was produced last year
- Clock Rates stalled
- Storage getting cheaper
  - Store more data!





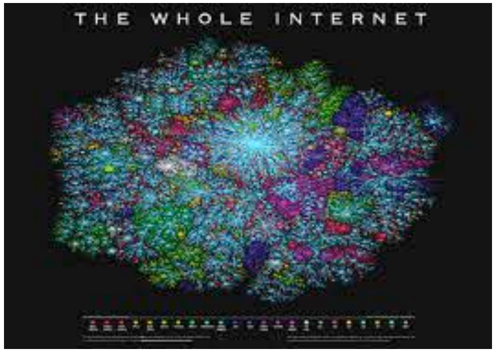


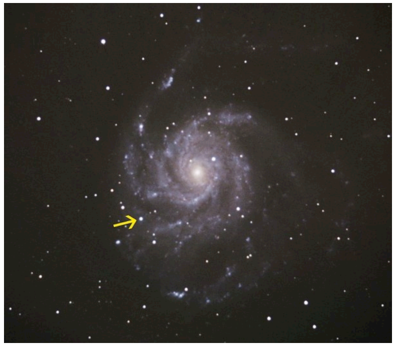
Data Grows Faster than Moore's Law



Complex Questions

- **Hard questions**
  - What is the impact on traffic and home prices of building a new ramp?
- **Detect real-time events**
  - Is there a cyber attack going on?
- **Open-ended questions**
  - How many supernovae happened last year?

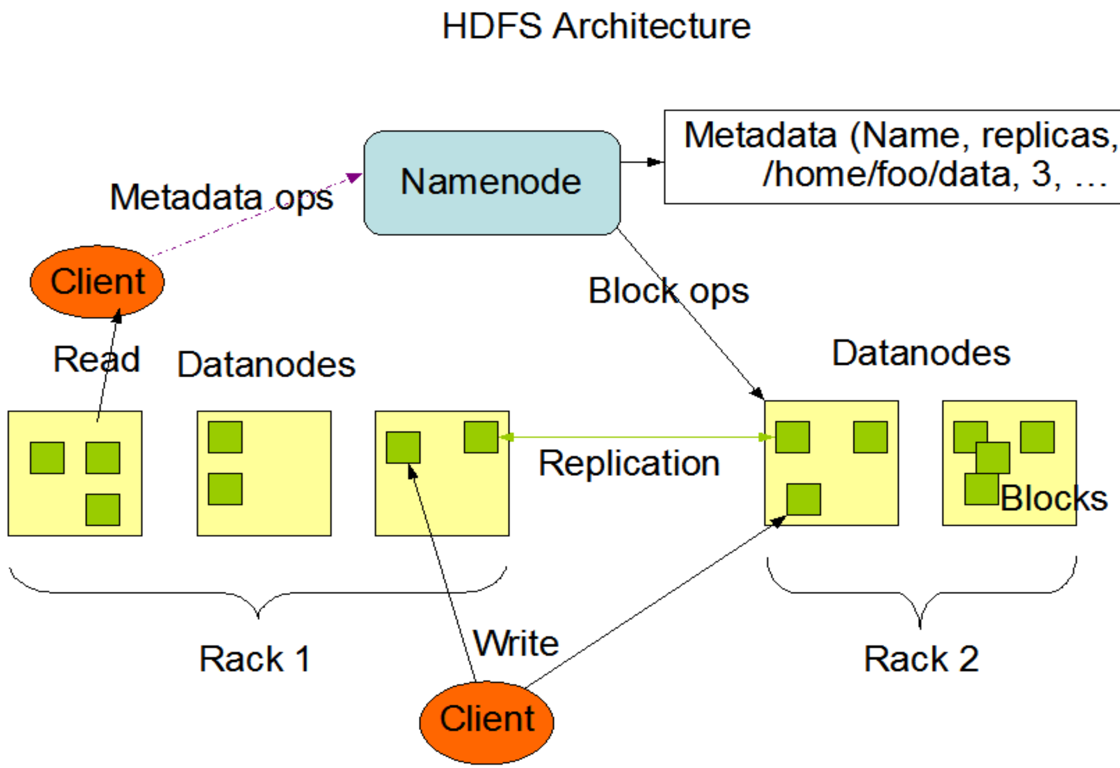




MapReduce Pros

- Distribution is completely transparent
  - Not a single line of distributed programming (ease, correctness)
- Automatic fault-tolerance
  - Determinism enables running failed tasks somewhere else again
  - Saved intermediate data enables just re-running failed reducers
- Automatic scaling
  - As operations as side-effect free, they can be distributed to any number of machines dynamically
- Automatic load-balancing
  - Move tasks and speculatively execute duplicate copies of slow tasks (*stragglers*)

HDFS – distributed file system



Master server
- manages file systems namespace
- Regulates client access
- Mapping to data nodes

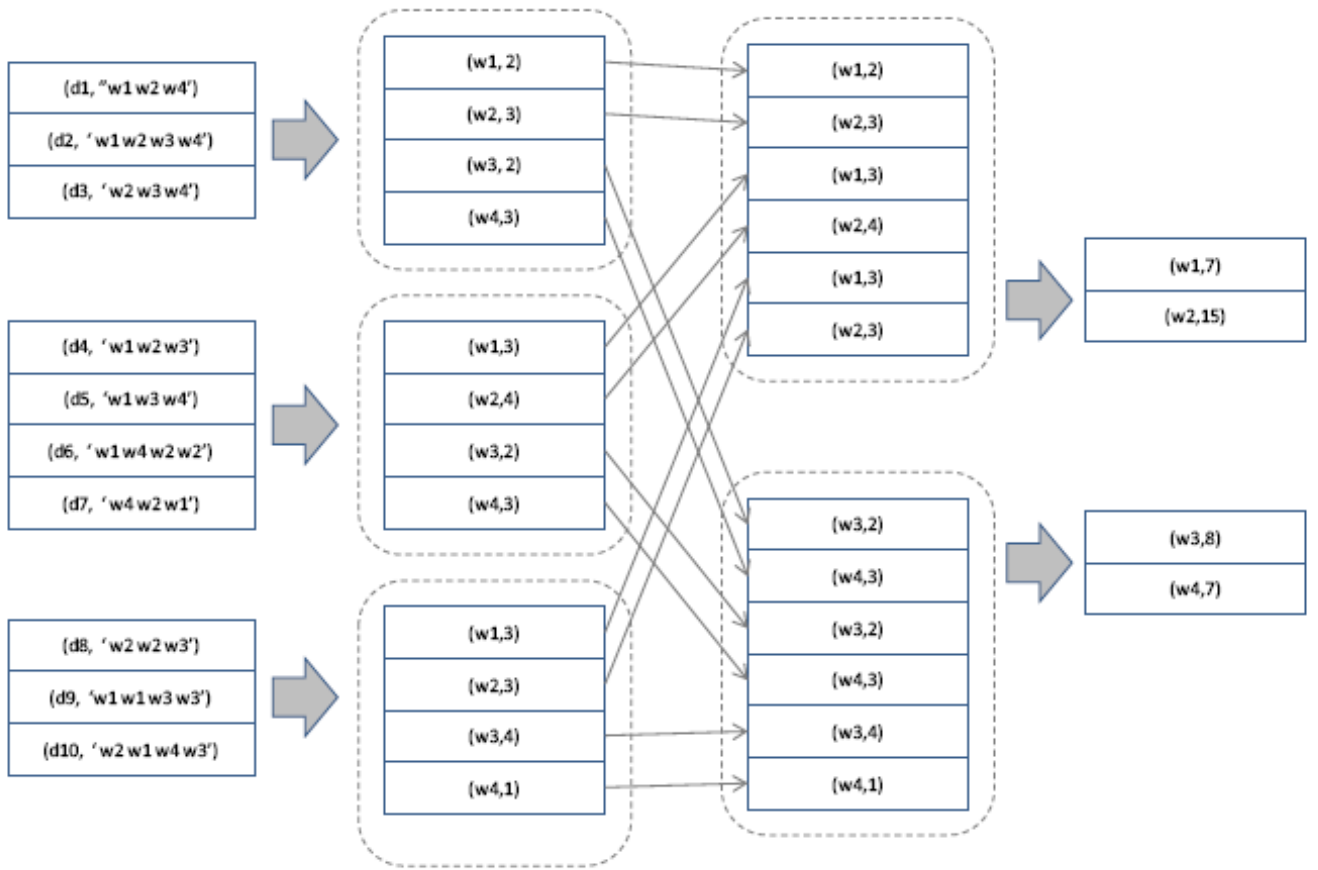
- **Blocks are distributed, with replicas, across nodes**
- **Name-node provides the index structure**
- **Client locates blocks via RPC to metadata**
- **Data nodes inform Namenode of failures through heartbeats**
- **Block locations made visible to MapReduce Framework**

MapReduce

- **both a *programming model* and a *clustered computing system***
  - **A specific pattern of computing over large distributed data sets**
  - **A system which takes a MapReduce-formulated problem and executes it on a large cluster**
  - **Hides implementation details, such as hardware failures, grouping and sorting, scheduling …**

Word-Count using MapReduce

Problem: determine the frequency of each word in a large document collection



M=3 mappers

R=2 reducers

$$(d_k, `w_1 \ldots w_n') \rightarrow [(w_i, c_i)]$$

$$(w_i, [c_i]) \rightarrow (w_i, \sum_i c_i)$$

General MapReduce Formulation

Map:

– Preprocesses a set of files to generate intermediate *key-value pairs, in parallel*

$$\text{Map:} \quad (k_1, v_1) \rightarrow [(k_2, v_2)]$$

Group:

– Partitions intermediate *key-value pairs* by unique key, generating a list of all associated values

» Shuffle so each key list is all on a node

Reduce:

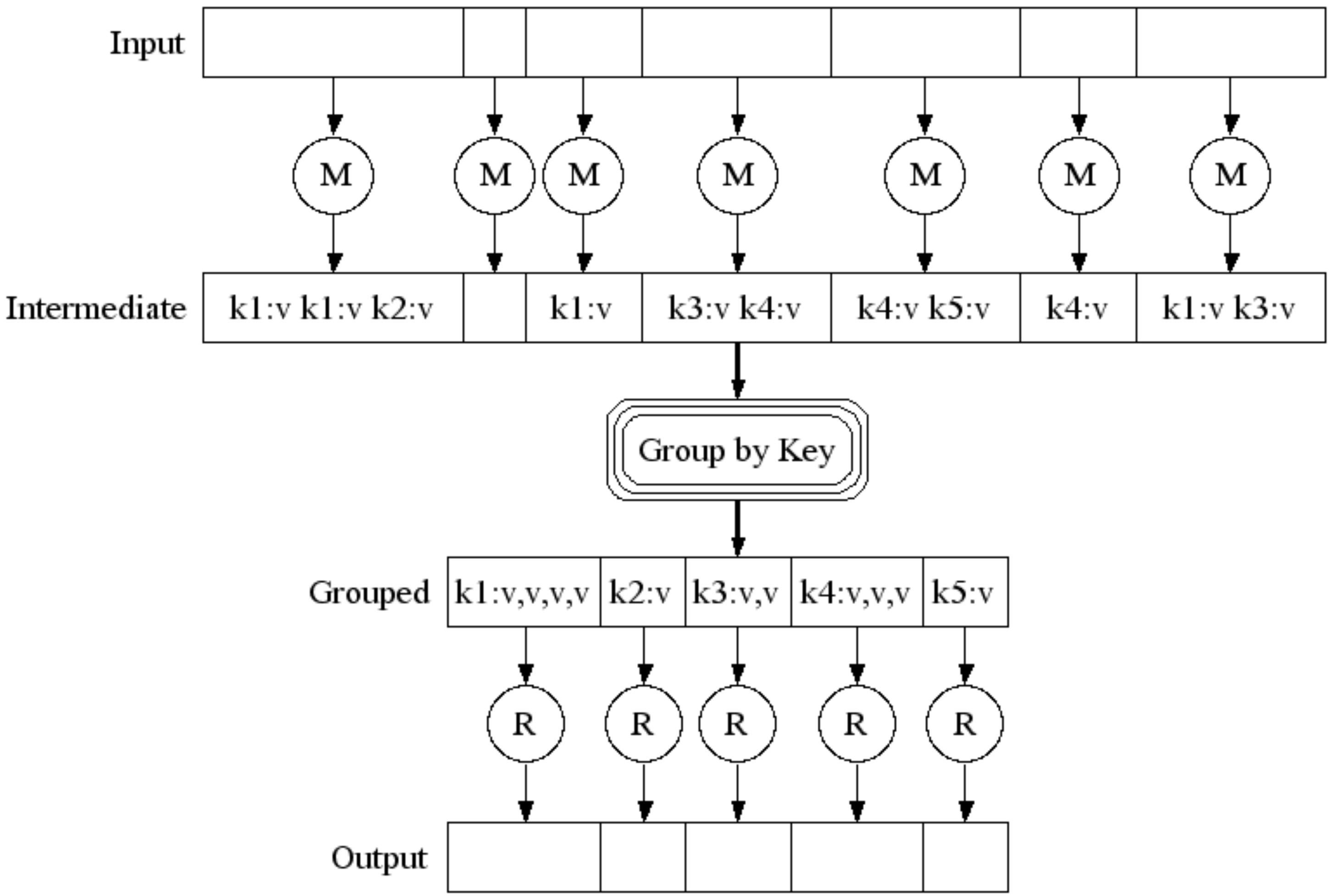
– For each key, iterate over value list

– Performs computation that requires context between iterations

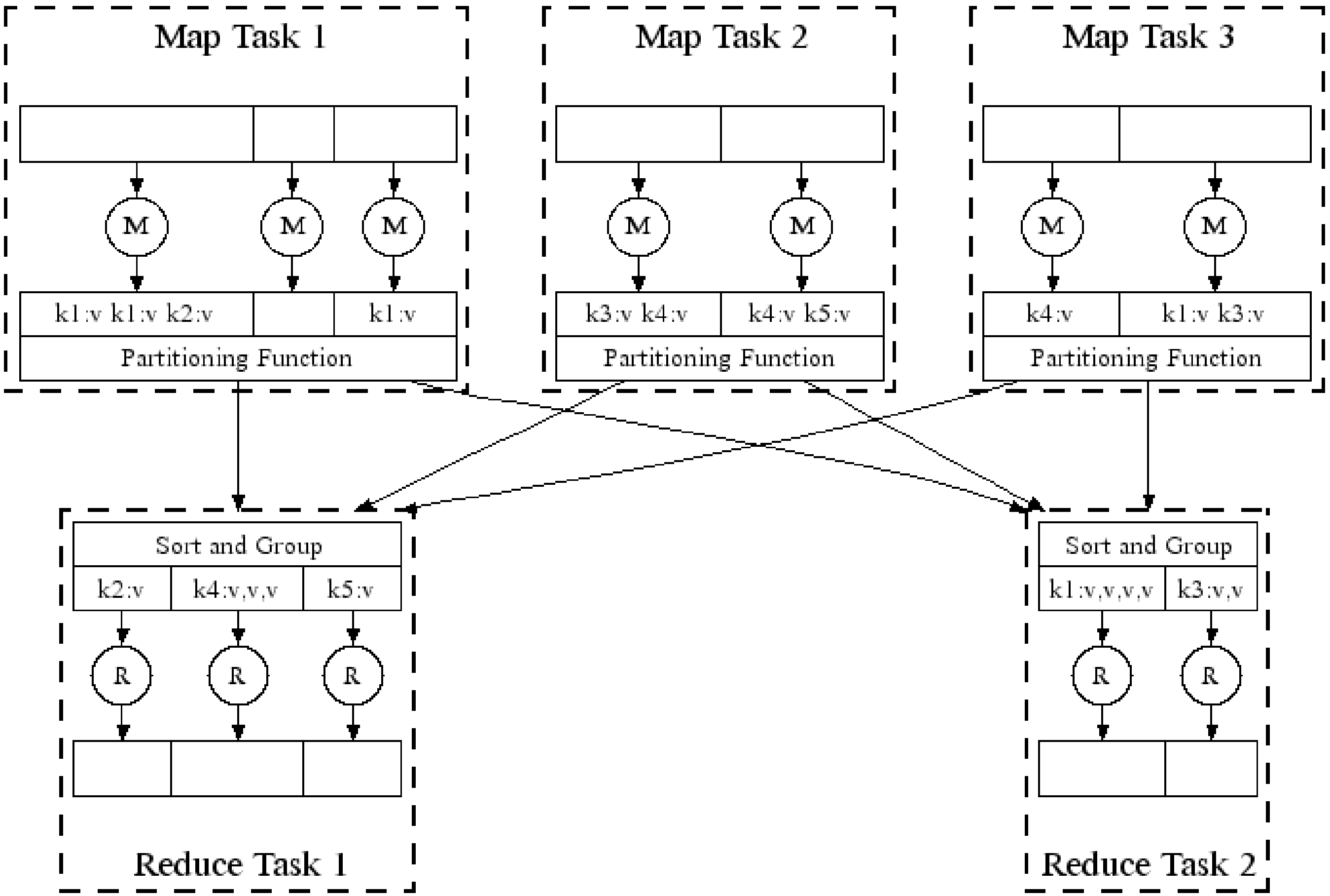
– Parallelizable amongst different keys, but not within one key

$$\text{Reduce:} \quad (k_2, [v_2]) \rightarrow (k_2, f([v_2]))$$

MapReduce Logical Execution



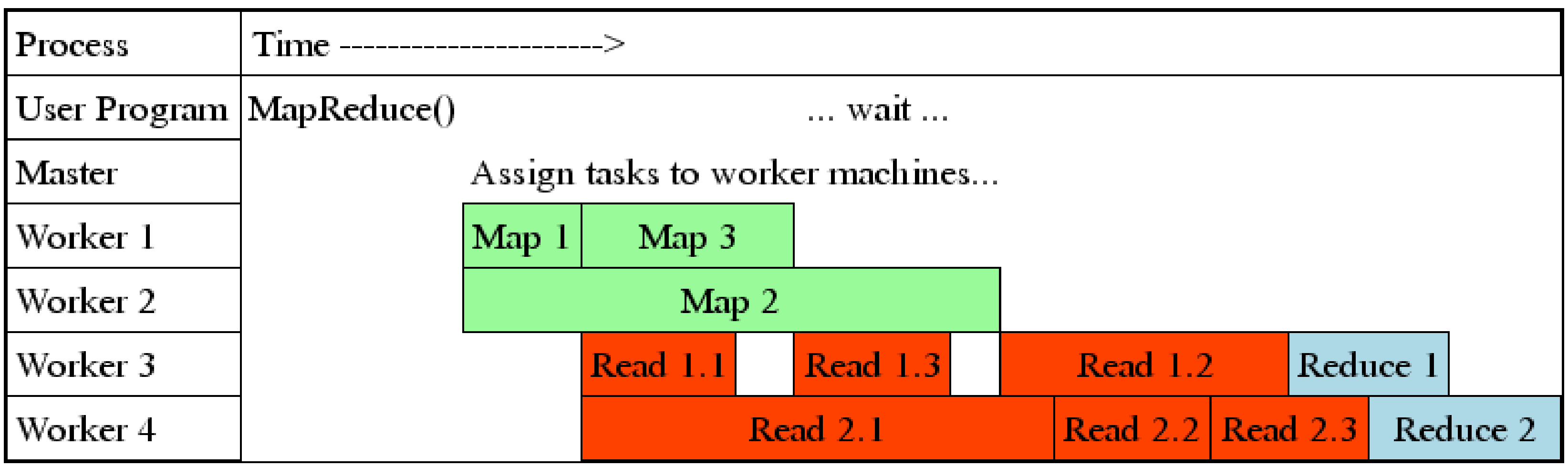
MapReduce Parallel Execution



Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

MapReduce Parallelization: Pipelining

- Fine grain tasks: many more map tasks than machines
  - Better dynamic load balancing
  - Minimizes time for fault recovery
  - Can pipeline the shuffling/grouping while maps are still running
- Example: 2000 machines -> 200,000 map + 5000 reduce tasks



Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

MR Runtime Execution Example

- The following slides illustrate an example run of MapReduce on a Google cluster
- A sample job from the indexing pipeline, processes ~900 GB of crawled pages

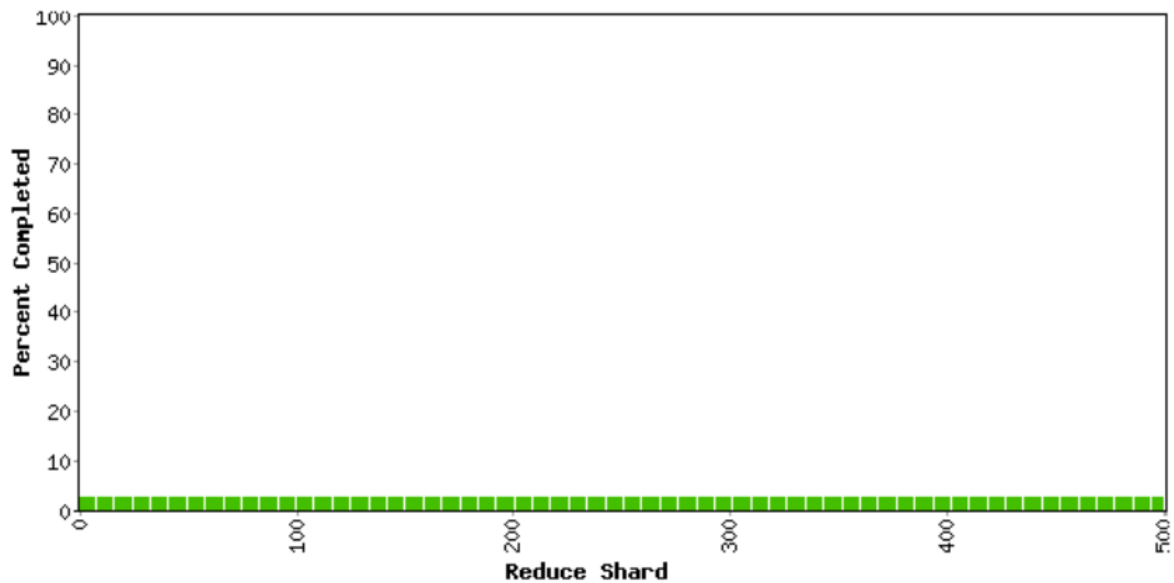
MR Runtime (1 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-outputs	506631

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

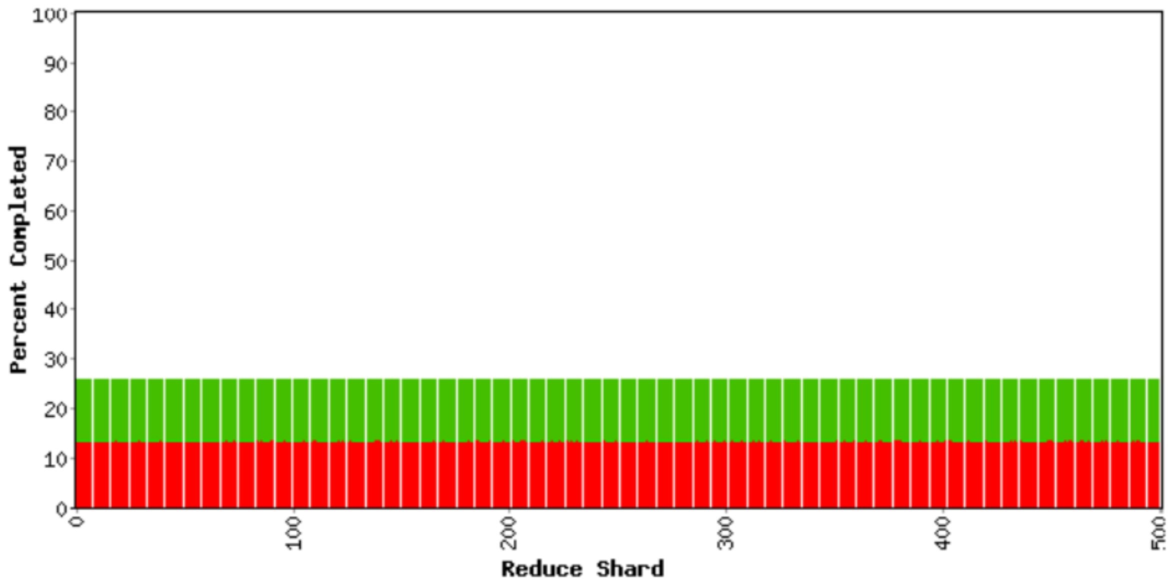
MR Runtime (2 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

MR Runtime (3 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0

Counters

Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

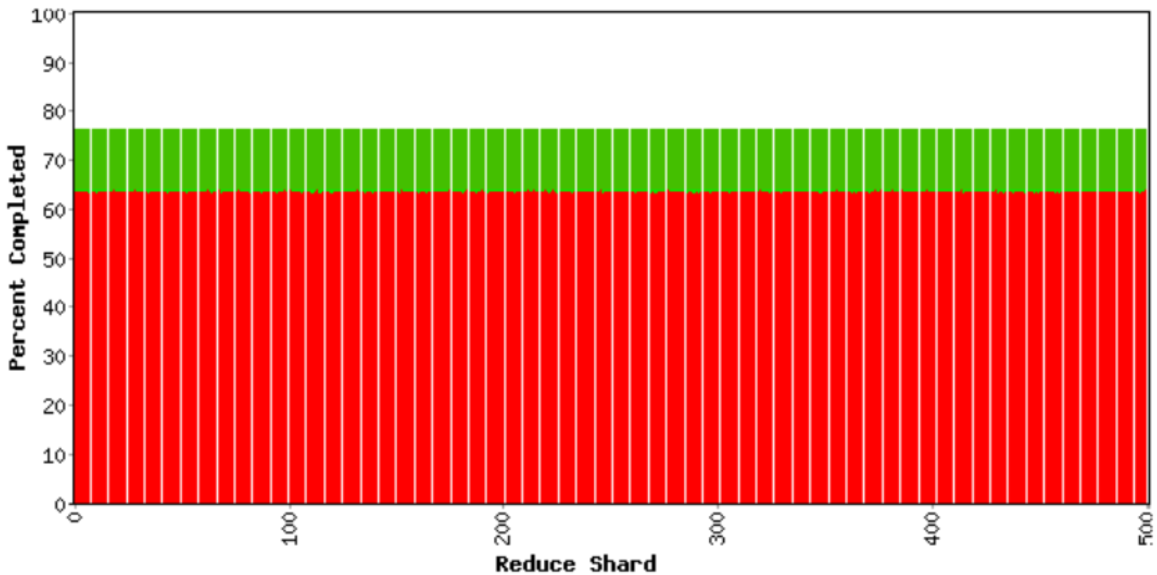
MR Runtime (4 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

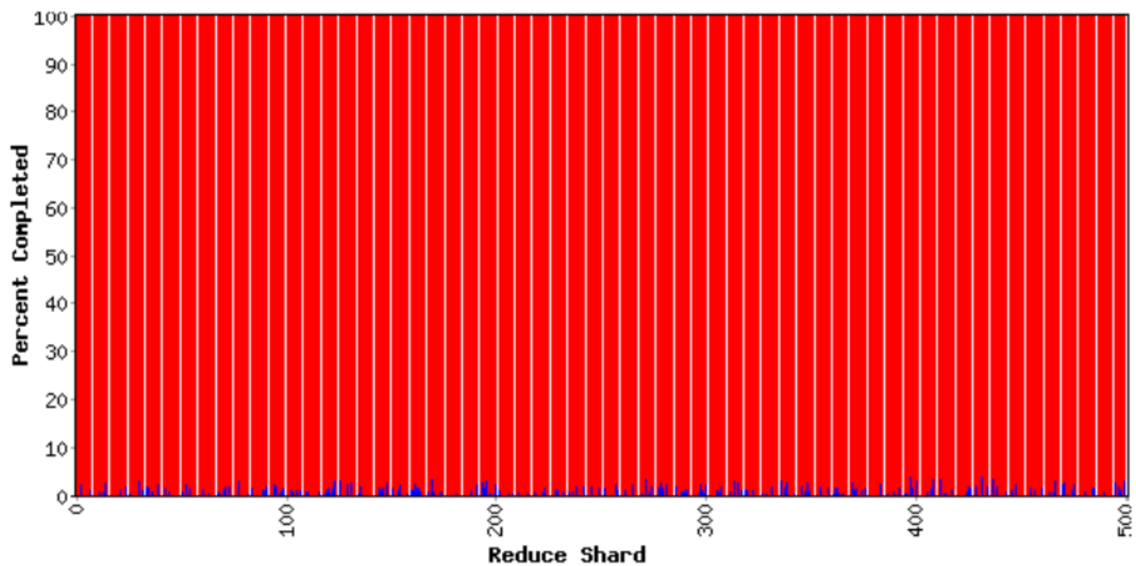
MR Runtime (5 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	10
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

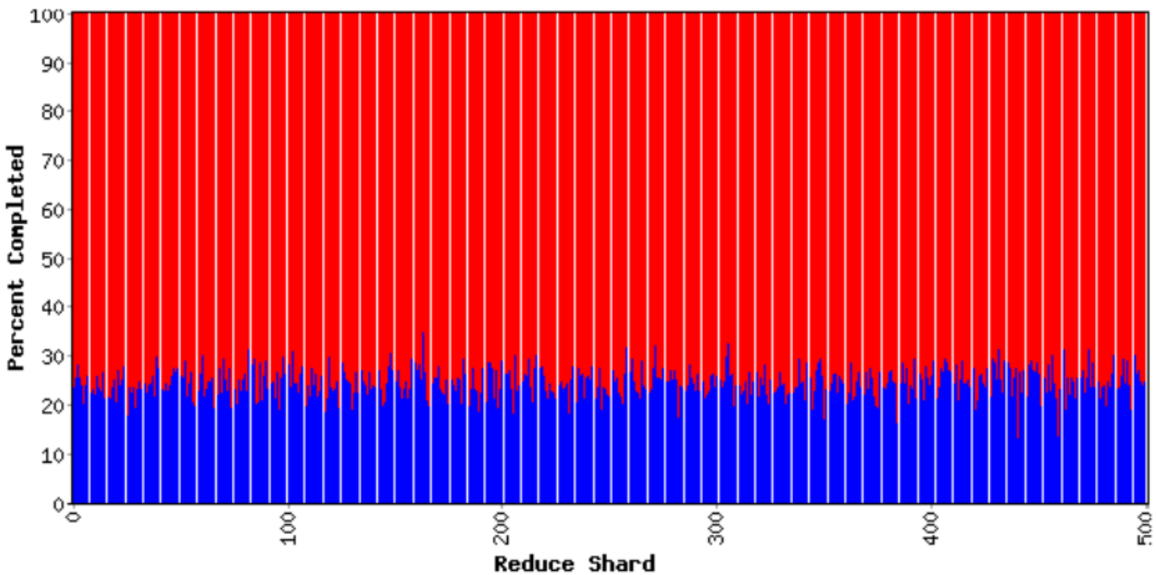
MR Runtime (6 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.1
Output (MB/s)	1238.8
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51738599
mr-merge-outputs	51738599

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

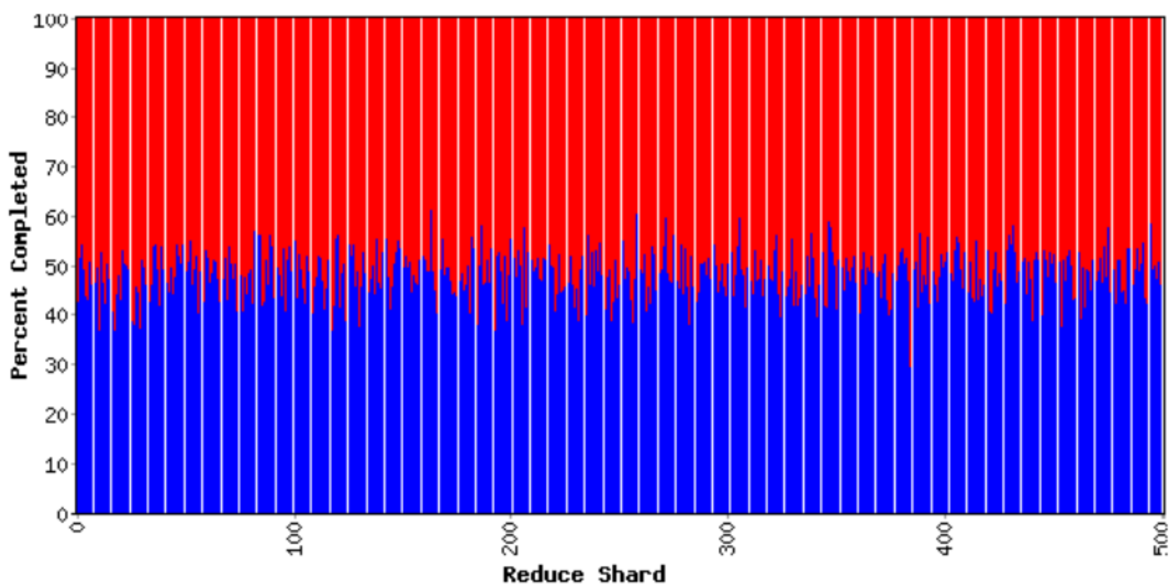
MR Runtime (7 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1225.1	
doc-index-hits	0	1
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51842100	
mr-merge-outputs	51842100	

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

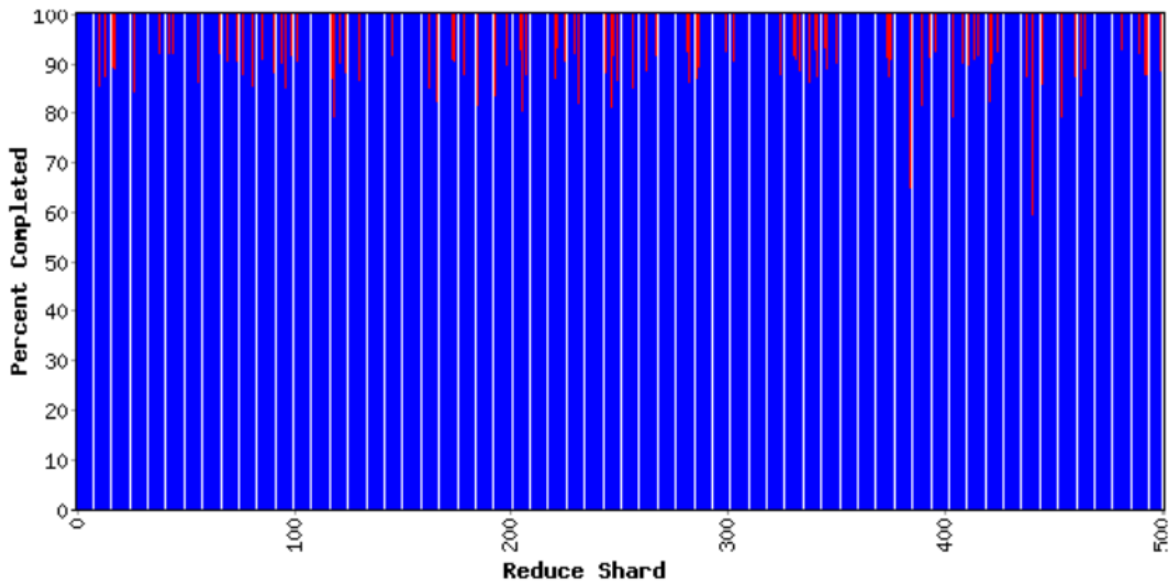
MR Runtime (8 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	849.5	
doc-index-hits	0	1
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	35083350	
mr-merge-outputs	35083350	

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

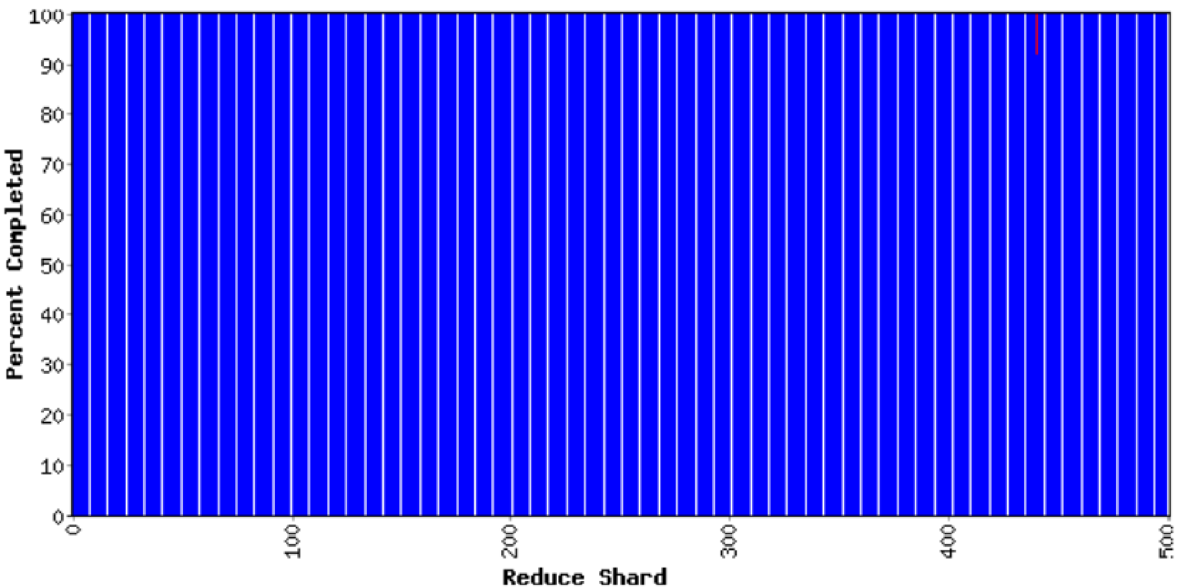
MR Runtime (9 of 9)

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

Shamelessly stolen from Jeff Dean's OSDI '04 presentation
http://labs.google.com/papers/mapreduce-osdi04-slides/index.html

Fault Tolerance vis re-execution

- **On Worker Failure:**
  - Detect via periodic heartbeats
  - Re-execute completed and in-progress *map* tasks
  - Re-execute in progress *reduce* tasks
  - Task completion committed through master
- **Master ???**

Admin

- Project 3
- Reviews during R&R
- Mid 3 in Final Exam Group 1 (12/15 8-11)
  - 10 Evans (currently)

MapReduce Cons

- **Restricted programming model**
  - Not always natural to express problems in this model
  - Low-level coding necessary
  - Little support for iterative jobs (lots of disk access)
  - High-latency (batch processing)

- **Addressed by follow-up research and Apache projects**
  - Pig and Hive for high-level coding
  - Spark for iterative and low-latency jobs

Big Data Ecosystem Evolution

MapReduce

Pregel Giraph

Dremel Drill Tez

Impala GraphLab

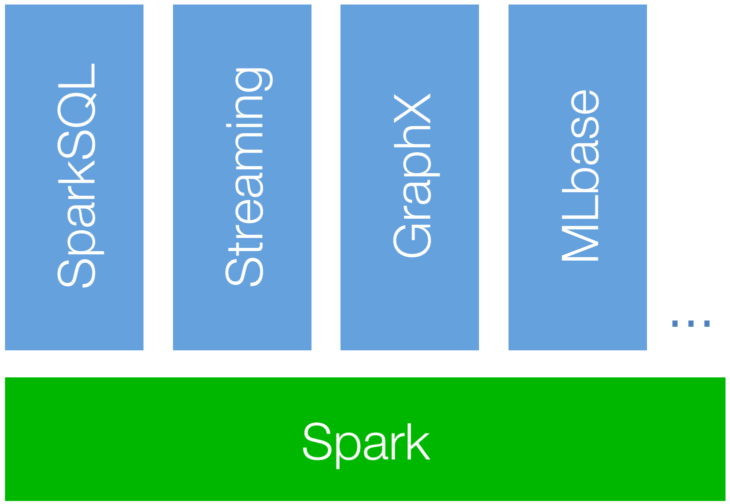
Storm S4 …

General batch processing

Specialized systems (iterative, interactive and streaming apps)

AMPLab Unification Philosophy

- Don't specialize MapReduce – Generalize it!
- Two additions to Hadoop MR can enable all the models shown earlier!
- 1. General Task DAGs
- 2. Data Sharing
- For Users:
- Fewer Systems to Use
- Less Data Movement

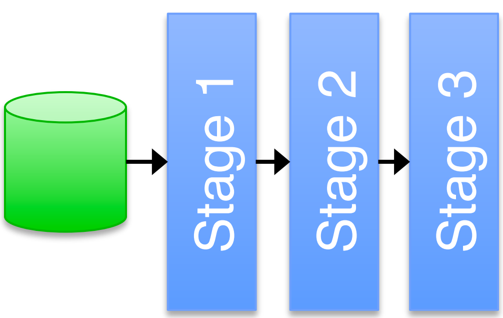


UCB / Apache Spark Motivation

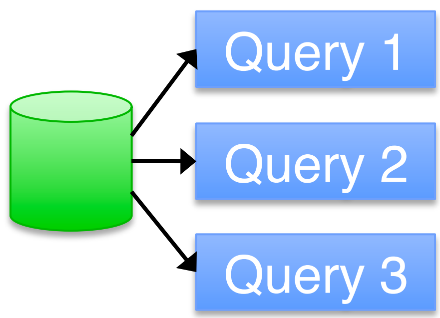
Spark
Lightning-Fast Cluster Computing

Complex jobs, interactive queries and online processing all need one thing that MR lacks:

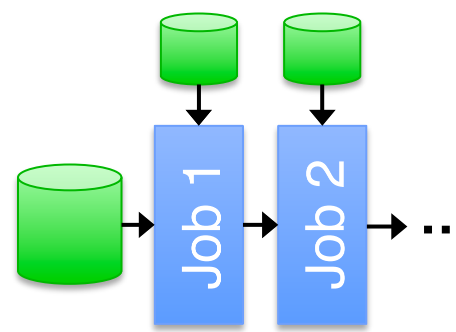
Efficient primitives for data sharing



Iterative job



Interactive mining



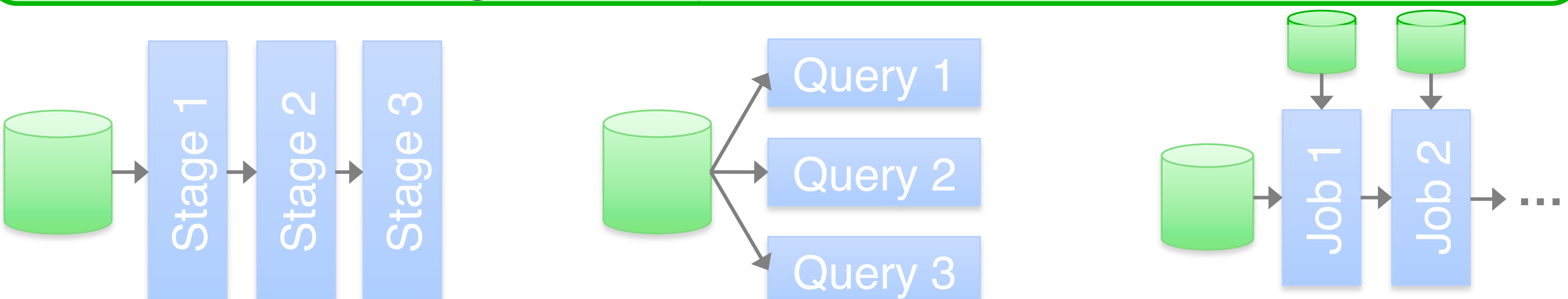
Stream processing

Spark Motivation

Complex jobs, interactive queries and online processing all need one thing that MR lacks:

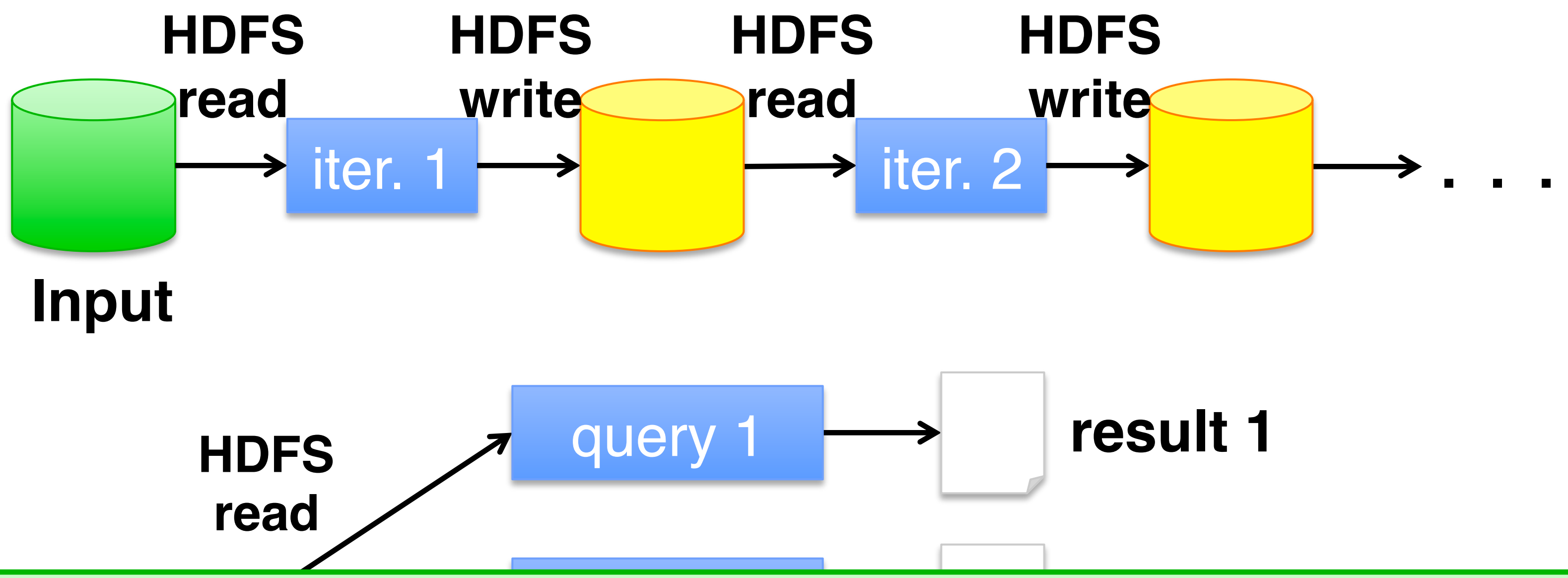
Efficient primitives for data sharing

Problem: in MR, the only way to share data across jobs is using stable storage (e.g. file system) ➔ slow!



Iterative job **Interactive mining** **Stream processing**

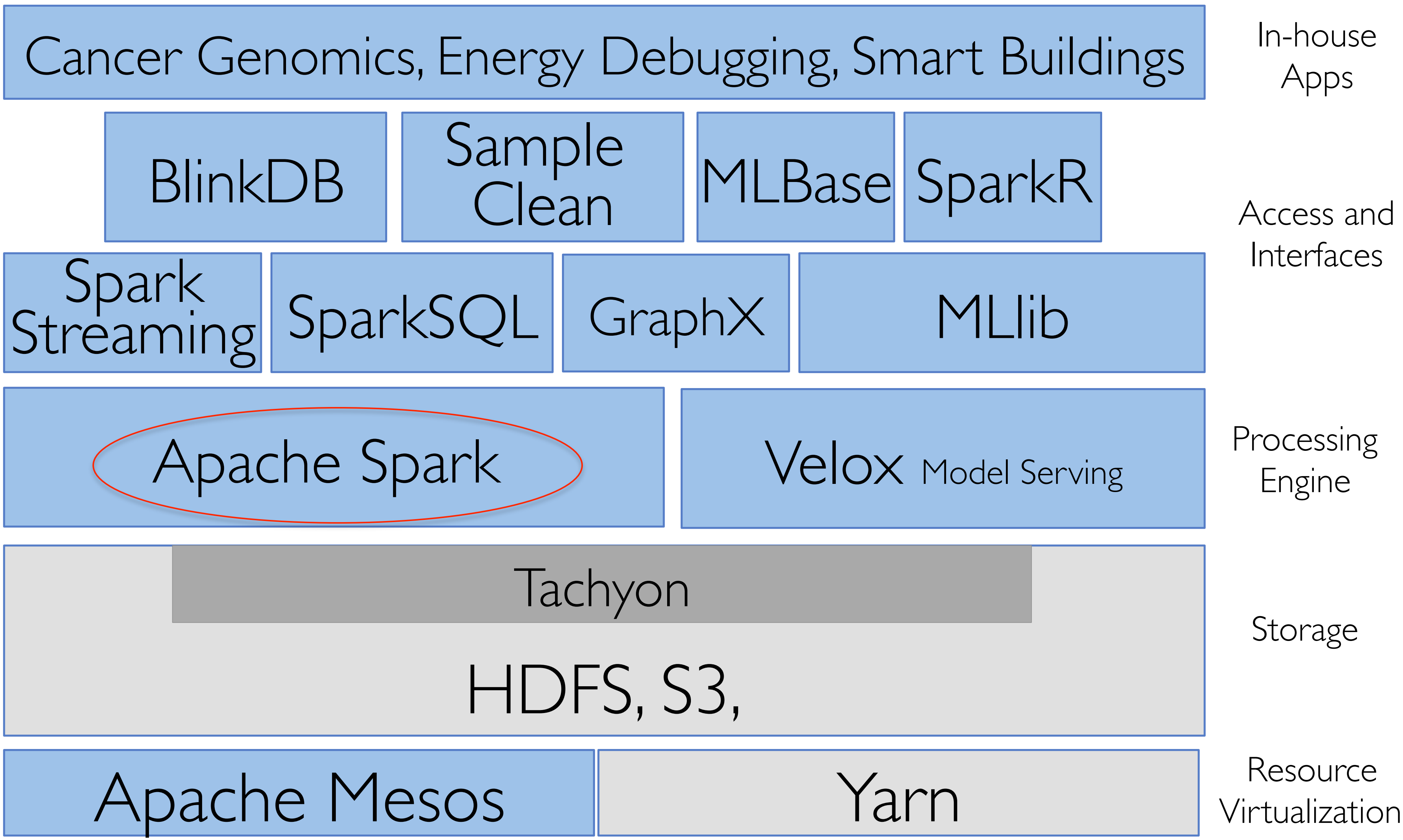
Examples



Opportunity: DRAM is getting cheaper ➔ use main memory for intermediate results instead of disks

. . .

Berkeley Data Analytics Stack (open source software)



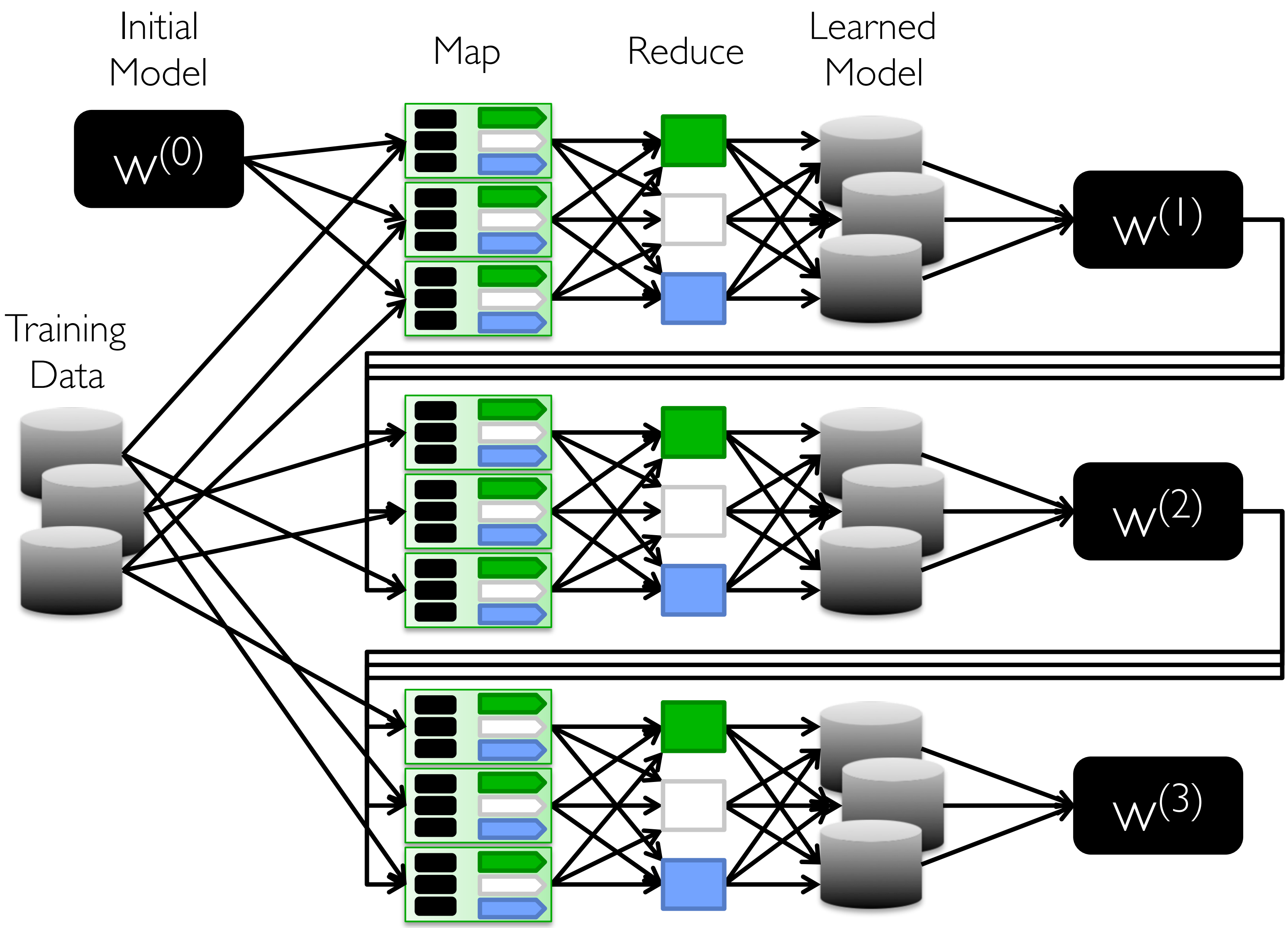
"It's only September but it's already clear that 2014 will be the year of Apache Spark"

-- Datanami, 9/15/14

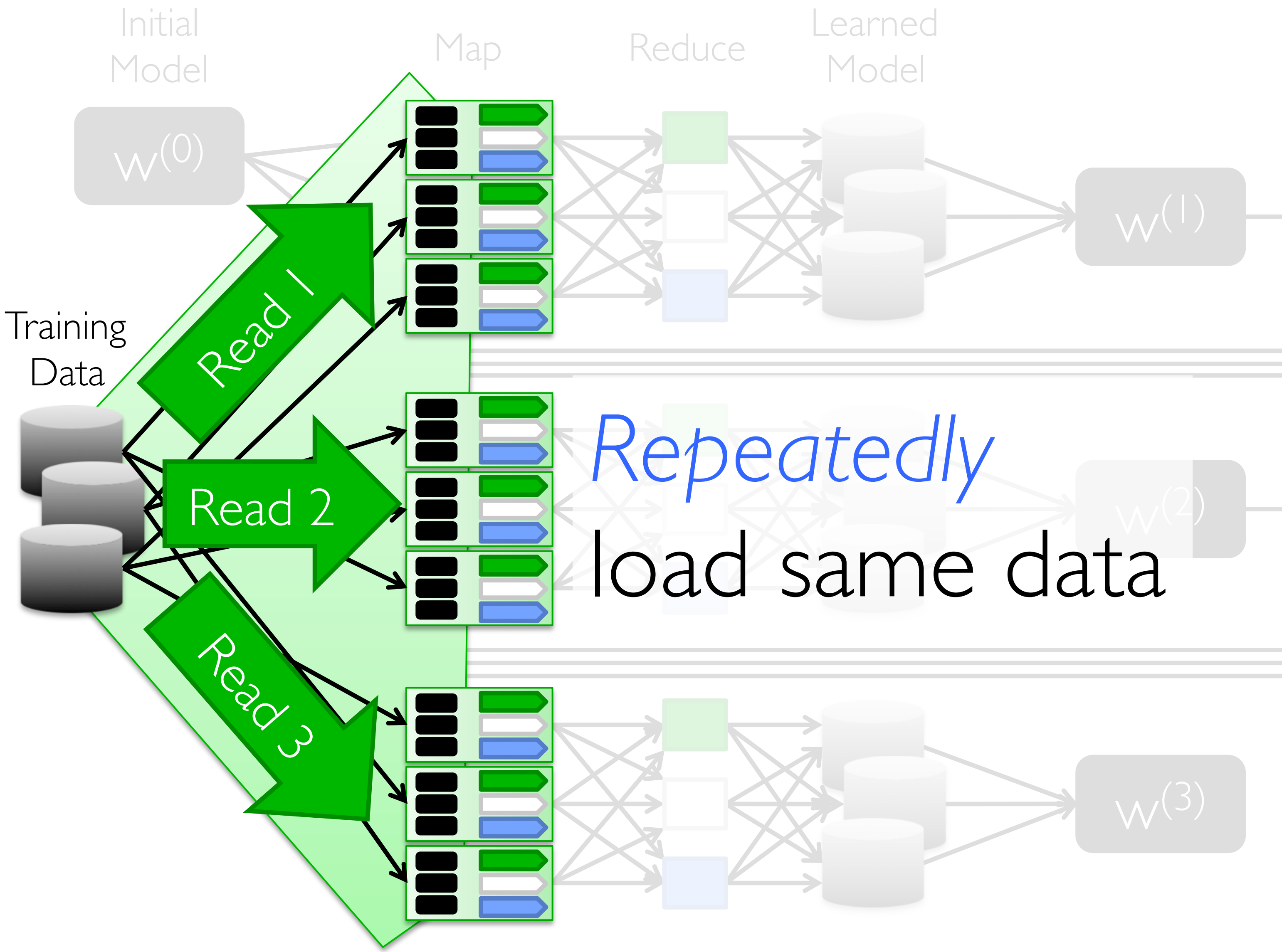
In-Memory Dataflow System

M. Zaharia, M. Choudhury, M. Franklin, I. Stoica, S. Shenker, "Spark: Cluster Computing with Working Sets, USENIX HotCloud, 2010.

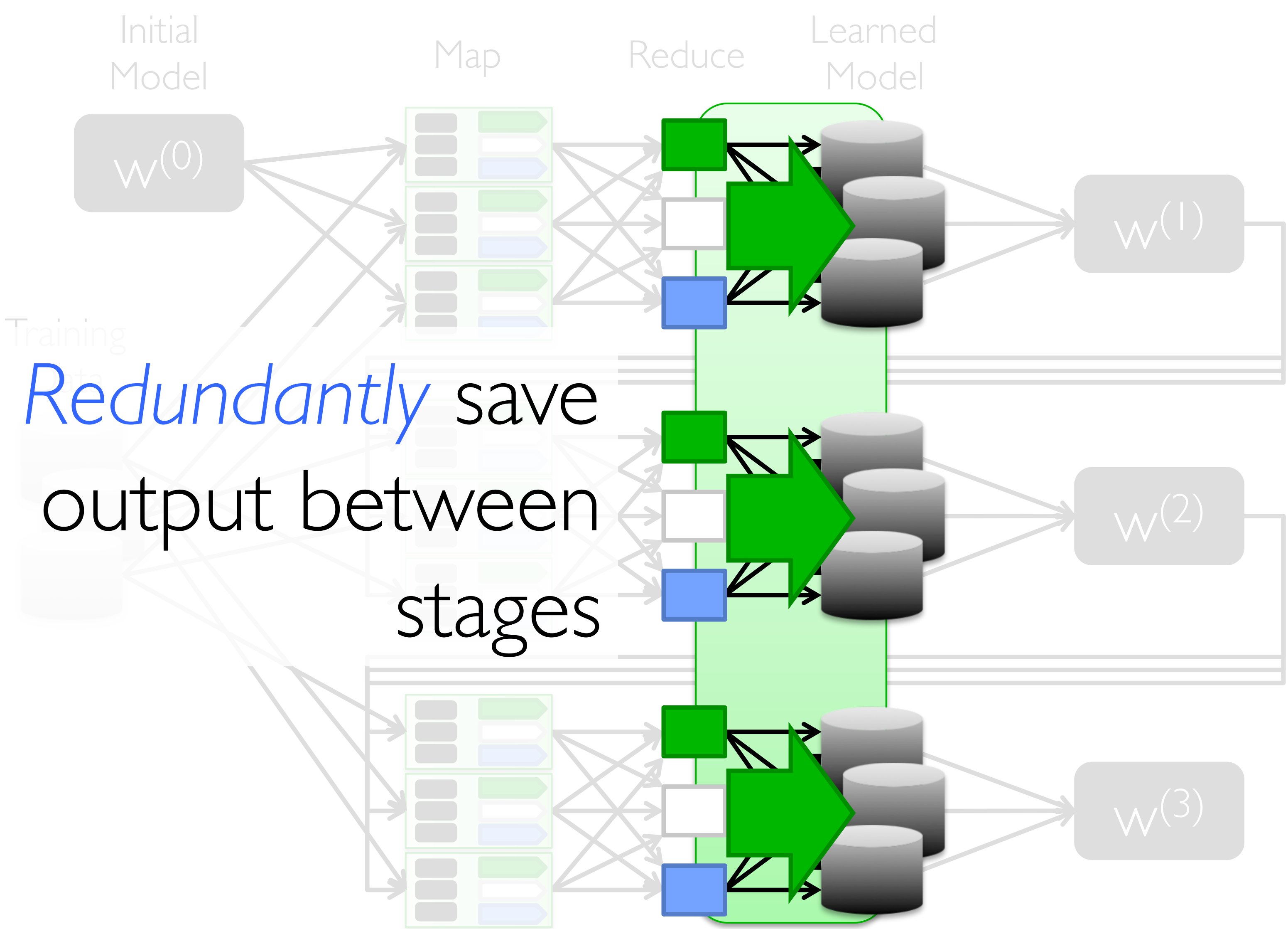
Iteration in Map-Reduce



Cost of Iteration in Map-Reduce

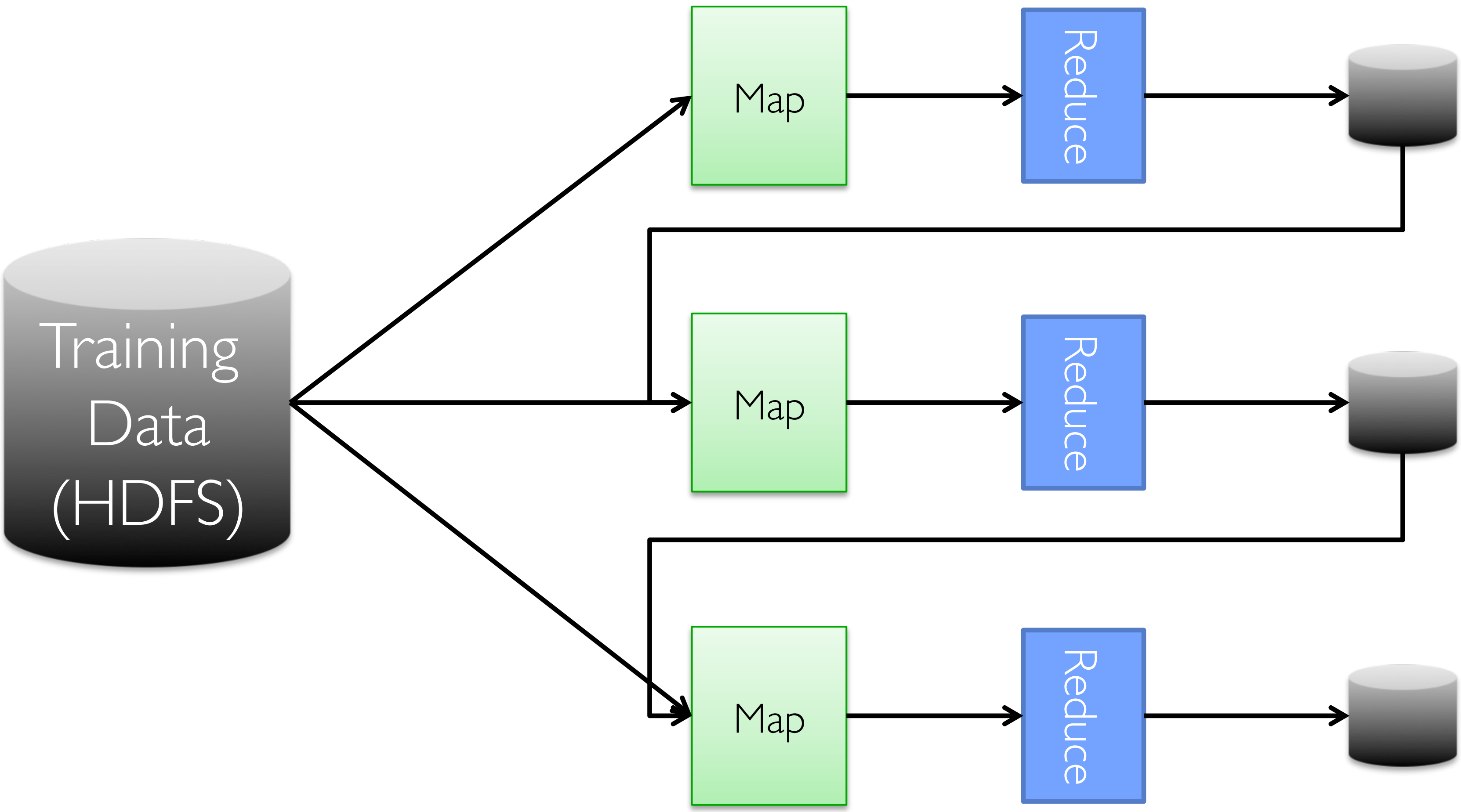


Cost of Iteration in Map-Reduce

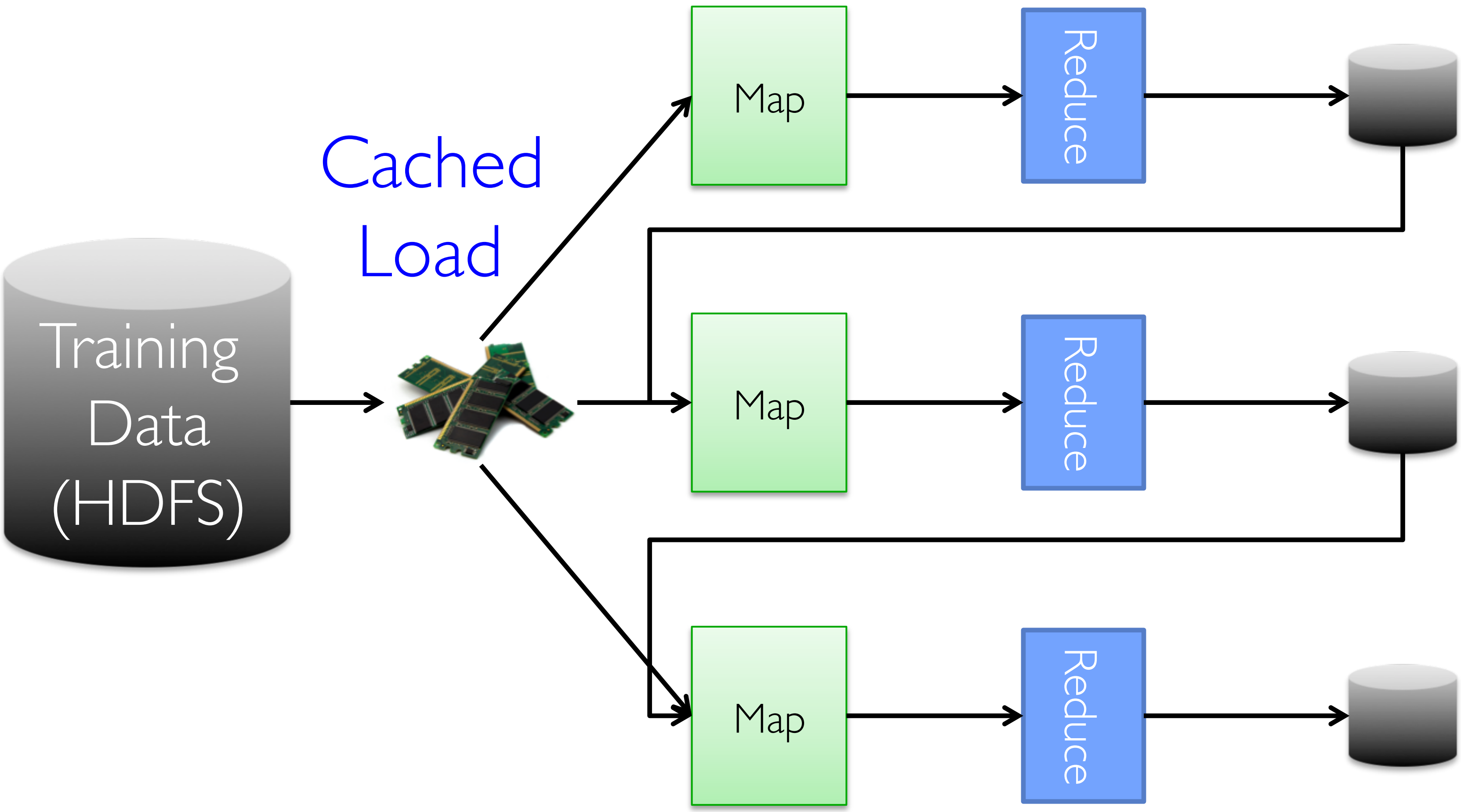


Redundantly save output between stages

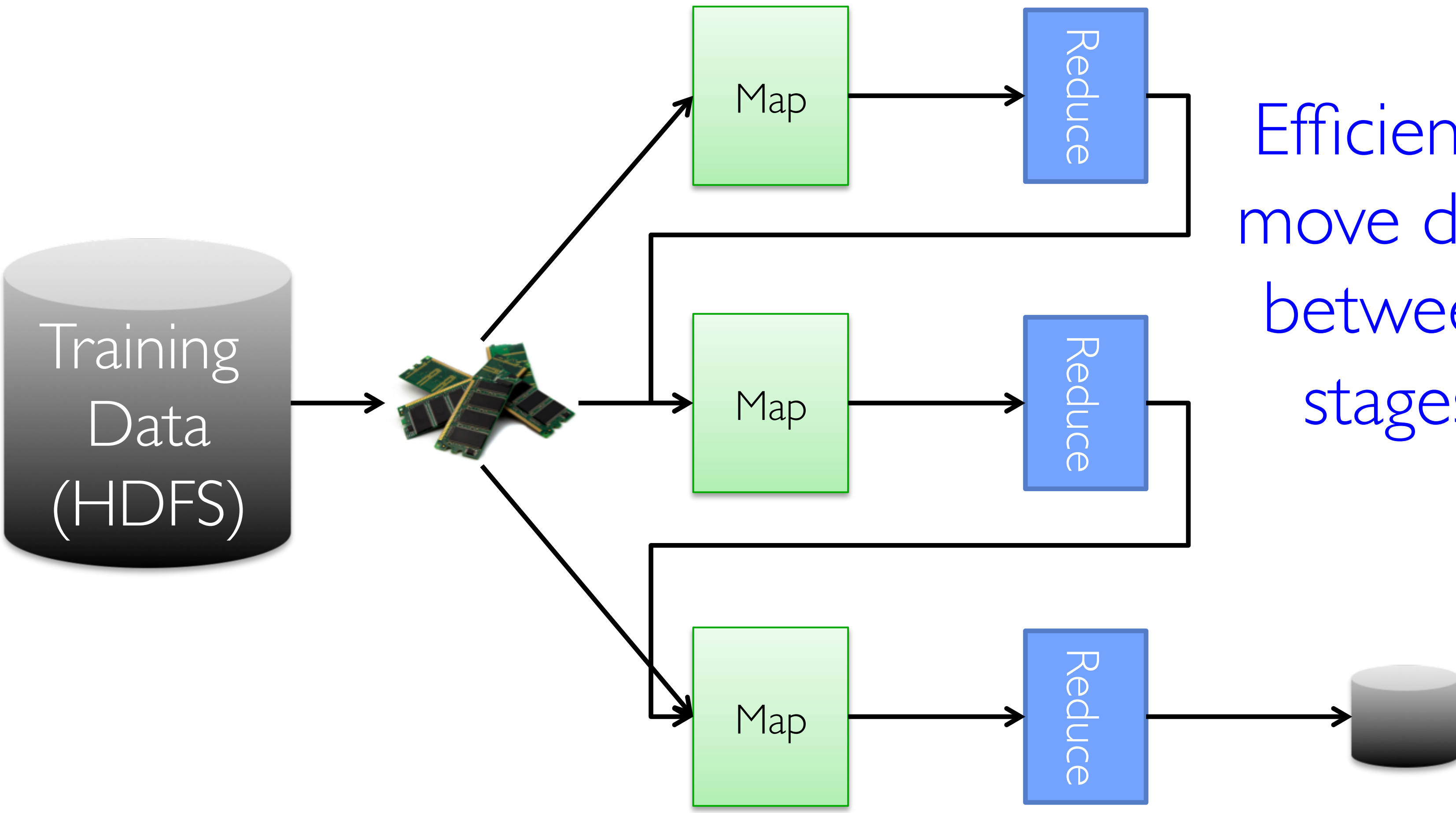
Dataflow View



Memory Opt. Dataflow



Memory Opt. Dataflow View



Efficiently move data between stages

Spark:10-100× faster than Hadoop MapReduce

Resilient Distributed Datasets (RDDs)

- **API: coarse-grained *transformations* (map, group-by, join, sort, filter, sample,…) on immutable collections**
- **Resilient Distributed Datasets (RDDs)**
  - **Collections of objects that can be stored in memory or disk across a cluster**
  - **Built via parallel transformations (map, filter, …)**
  - **Automatically rebuilt on failure**
- **Rich enough to capture many models:**
  - **Data flow models: MapReduce, Dryad, SQL, …**
  - **Specialized models: Pregel, Hama, …**

M. Zaharia, et al, Resilient Distributed Datasets: A fault-tolerant abstraction for in-memory cluster computing, NSDI 2012.

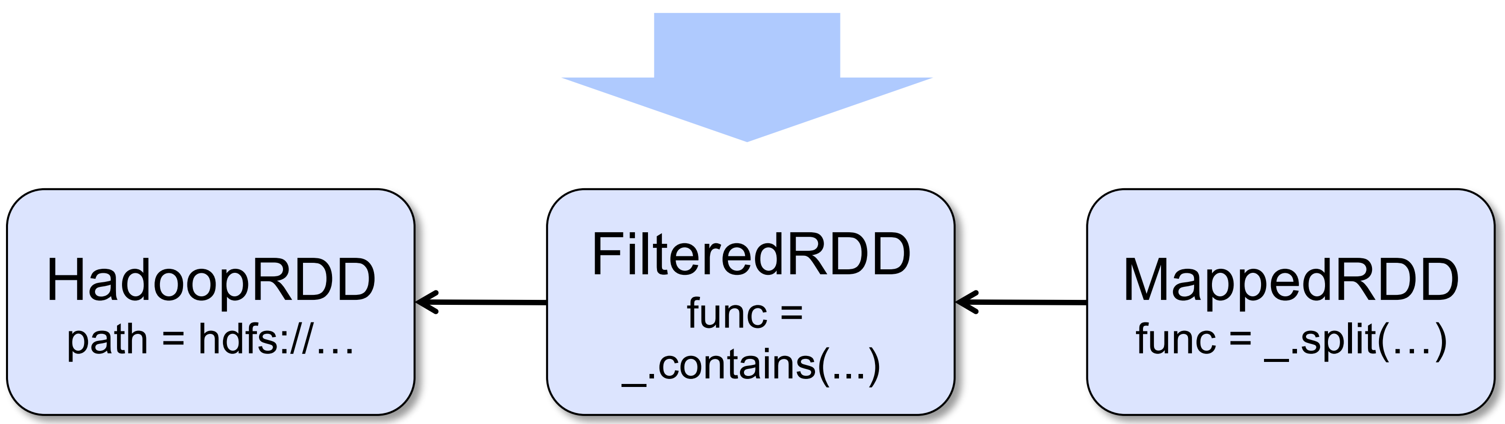
Fault Tolerance with RDDs

RDDs track the series of transformations used to build them (their *lineage*)

- Log one operation to apply to many elements
- No cost if nothing fails

Enables per-node recomputation of lost data

```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split('\t')(2))
```



Systems Research as Time Travel …