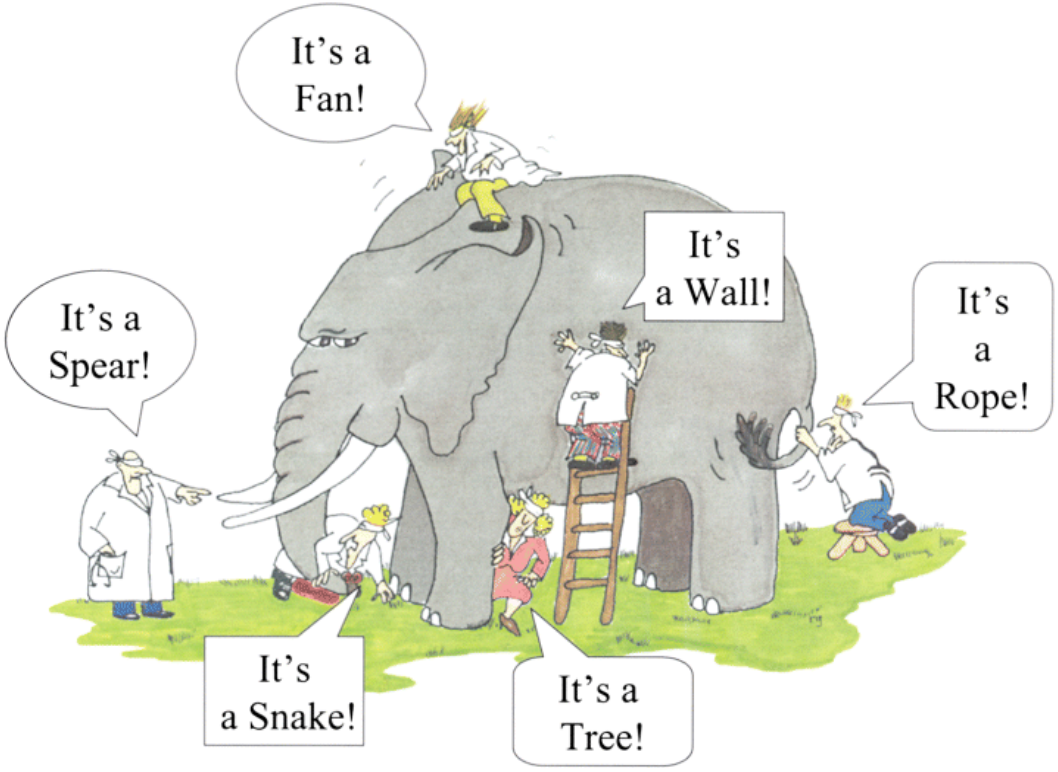
Blind Men and an Elephant: Perspectives on BigData



Chris Douglas, Carlo Curino

Microsoft Cloud and Information Services Lab

>whoami

Microsoft

Research Engineer, Cloud and Information Services Lab (CISL)

Apache

Chair, Apache Hadoop PMC

>id curino

Microsoft

Scientist, Cloud and Information Services Lab (CISL)

Apache

Committer, Apache Hadoop

[citation]

Goals and Disclaimers

Goal

- Understand the context (History)
- Familiarize with the ecosystem (Taxonomy and key insights)
- Learn to contribute (OSS—industry—Research interactions)

Classical Disclaimers

- Not exhaustive or unbiased
- Liberally borrowed slides from others
- If I talk about your system, I expect you to stick up for it

History

Those who fail to learn from history are still gainfully employed

BigData in the Database World

OLAP data warehouses

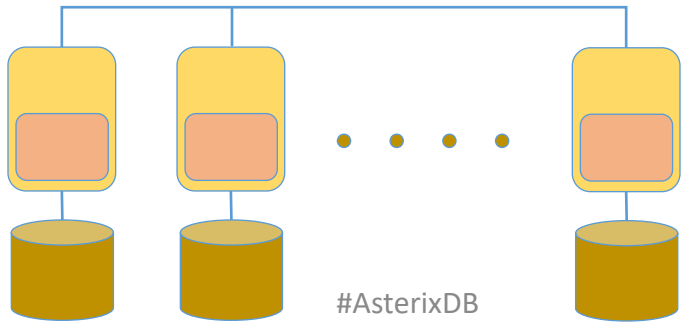
Store and query historical business data

1980's: Parallel database systems based on "shared-nothing" architectures (Gamma/GRACE, Teradata)

2000's: Netezza, Aster Data, DATAllegro, Greenplum, Vertica, ParAccel

OLTP

1980's: Tandem's NonStop SQL system



Slide courtesy of Michael Carey

Web Crawl

Data

- Bulk, append only
- Massive (multi-PiB)
- Unstructured

Query

- Days, weeks to process
- Mostly full scans
- Procedural computation

Hardware

- Thousands of machines
- Consumer grade

The origin

- **Purpose-built technology**
- Within large web companies
- Well targeted mission (process webcrawl)
- → scale and fault tolerance

Google leading the pack

Google File System + MapReduce (2003/2004)

Open-source and parallel efforts

Yahoo! adopts Hadoop ecosystem HDFS + MapReduce (2006)

Microsoft Scope/Cosmos (2008) (more than MapReduce)

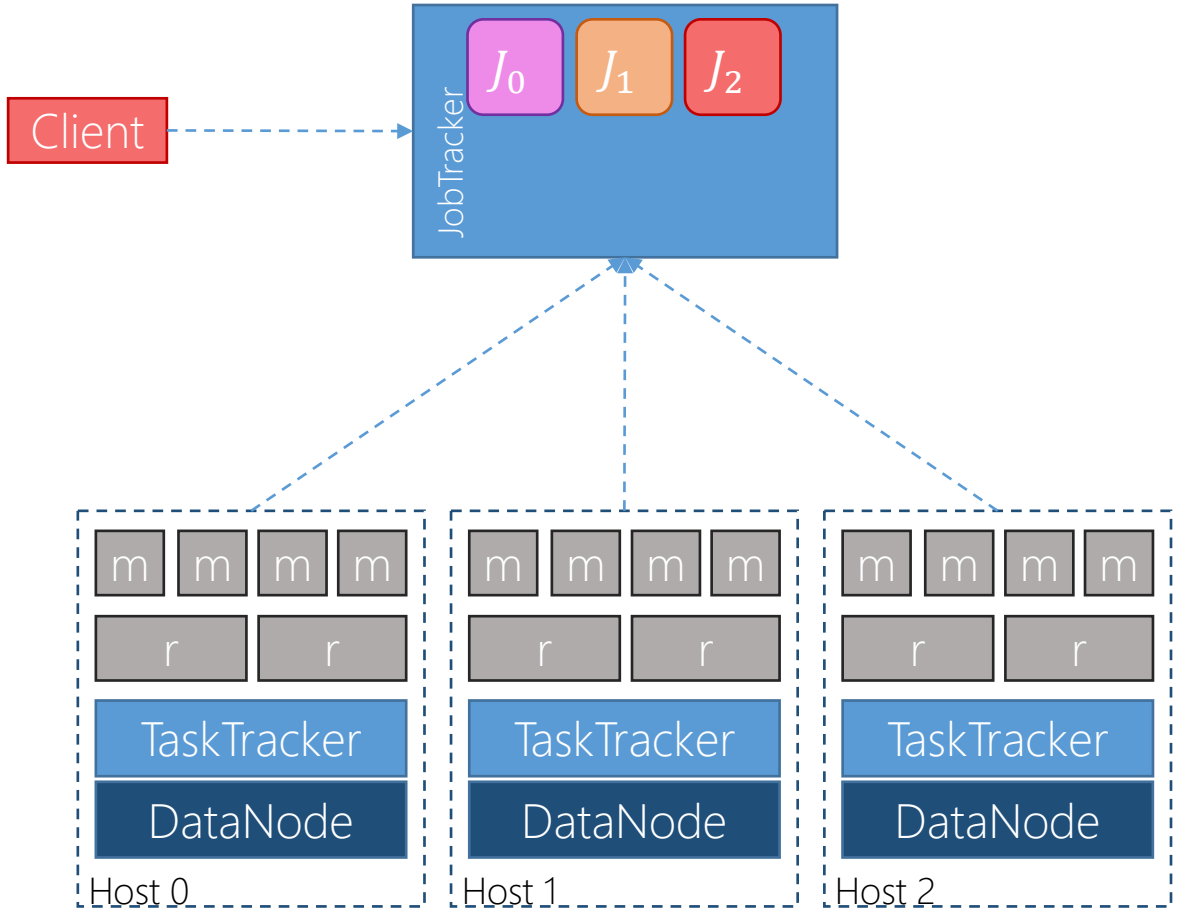
MapReduce (Hadoop 1.x)

Central Job/Resource coordinator

Workers collocated with data

Optimized for full scans

Automatic parallelism



Why are we talking about Hadoop?

Growth

Access, access, access…

All the data sit in the DFS

Trivial to use massive compute power

→ lots of new applications

But… everything has to be MapReduce

Cast any computation as map-only job

MPI, graph processing, streaming, *launching web-servers!?!*

Data Lake

Everybody wants BigData

- Insight from raw data is cool
- Outside MS and Google, BigData = Hadoop MapReduce
- Hadoop as catch-all BigData solution (and cluster manager)

Even for Small Data

- Queries over samples
- Exploratory analysis

Shortcomings of first-gen BigData

Programming model rigidity

- map/reduce operators not expressive enough
- High-level languages over MR limited

Efficiency

- Batch oriented, high latency
- Scale and fault tolerance over performance

Hadoop implementation limitations

- Map vs Reduce slots lead to *low cluster utilization* (~70%)
- Bottleneck for resource allocation
- Single points of failure

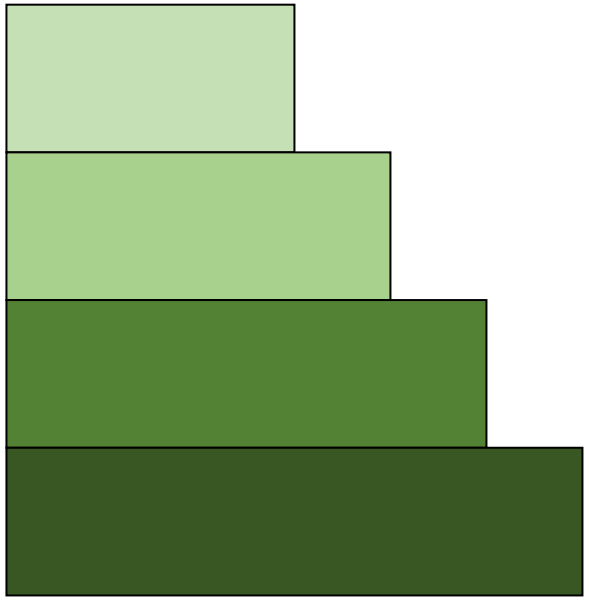
BigData next generation

We all agree it should be fixed

Papers, blog posts, startups… we get it

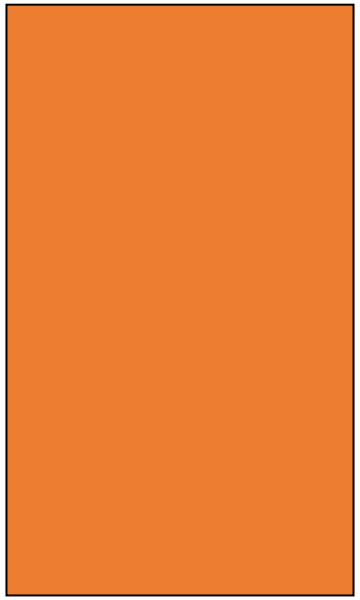
BigData second generation

Systems



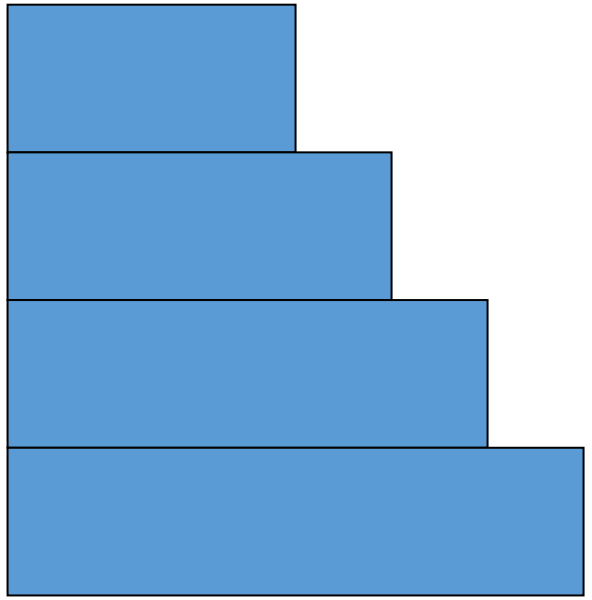
Pig, Hive, Cascading, Crunch, Sawzall, FlumeJava,…
Calcite, Summingbird, Corfu, …
Tez, REEF, Twill, Chronos, Marathon, …
YARN, Mesos, Omega, Borg, Corona, …

Database



Impala, SCOPE, Drill, Tajo, Presto, …

Hybrid



Spark, Asterix, Flink, …

Systems: Cluster OS

“The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise.” --Dijkstra

Systems: Cluster OS

Apache Hadoop YARN (2011, SoCC13)*

Request-based central scheduler

Apache Mesos (2009, NSDI11)*

Offer-based, two-level scheduler

Google Omega (2013, EuroSys13)*

Shared-state, two-level scheduler

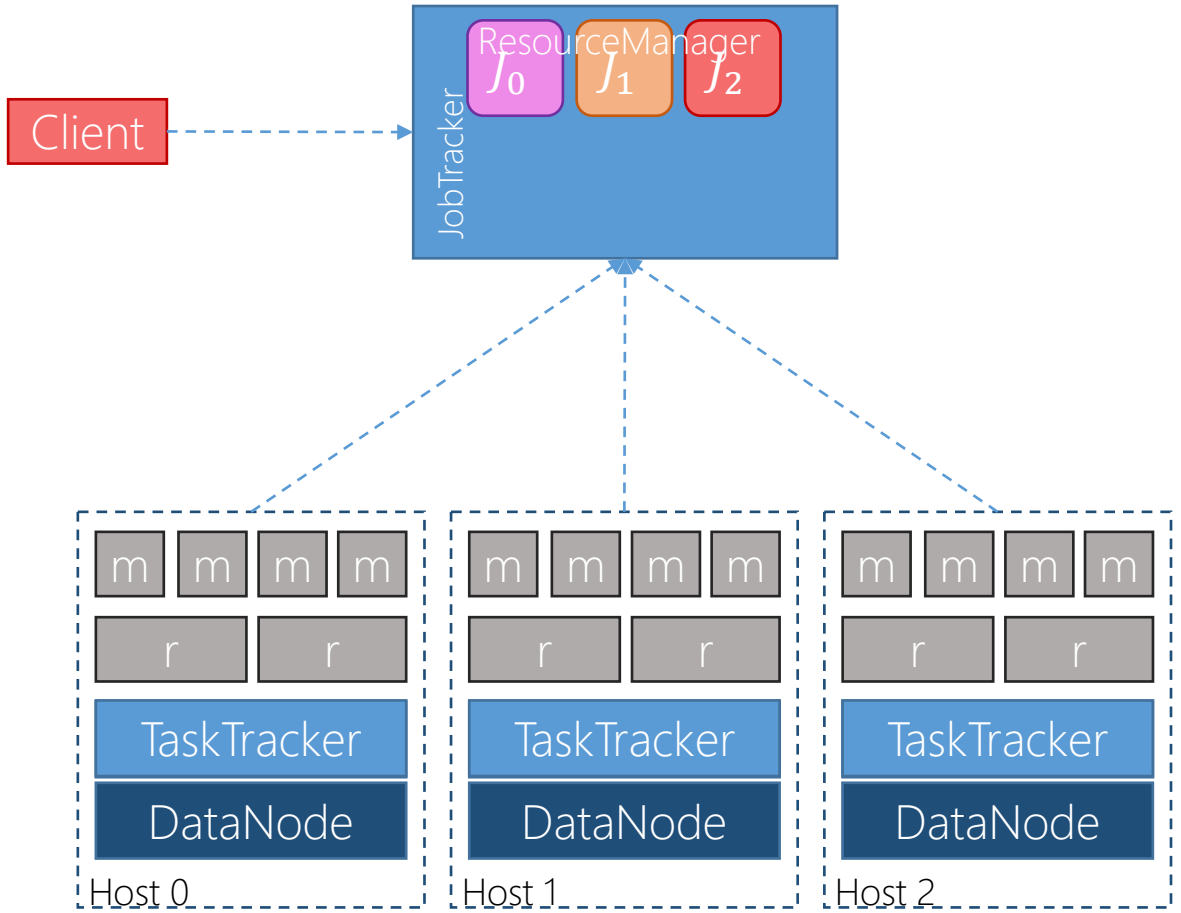
* all three won best-paper or best-student-paper

Apache Hadoop YARN

Central *ResourceManager (RM)*

Enforces sharing policies across tenants

Matches available rsrc to pending requests



YARN Container Lifecycle

Central *ResourceManager*

- Enforces sharing policies across tenants
- Matches available rsrc to pending requests

Users submit *Applications*

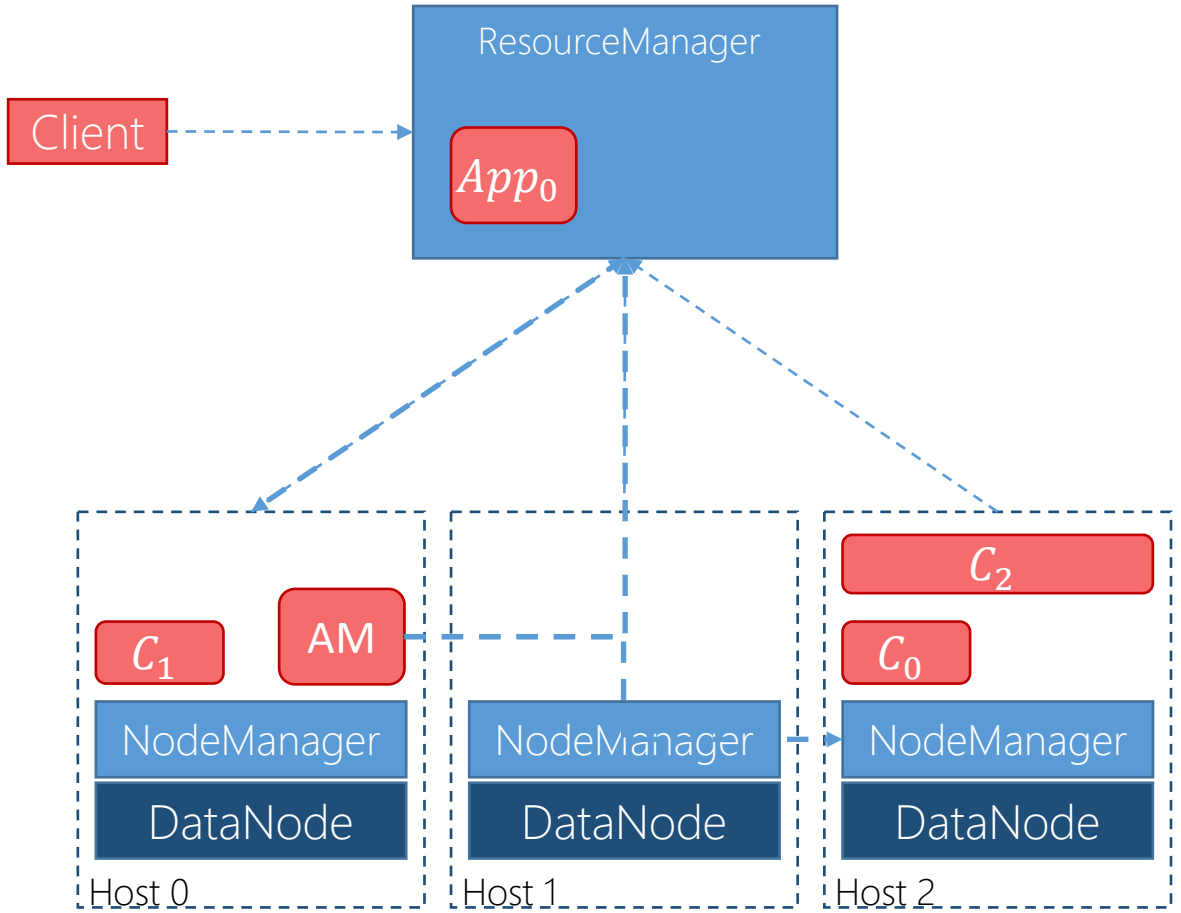
- ApplicationMaster (AM) control process

AMs request *Containers*

- ResourceRequest encodes constraints
- AM binds to a process desc on receipt

Containers started by *NodeManagers*

- Enforces isolation between processes
- Monitoring, reporting for containers



YARN ResourceRequest

Hard constraints

- Memory, CPU cores, [network], [disk]
- Affinity, anti-affinity

Node labels (opaque)

- e.g., Public-facing IP, processor architecture, OS

Soft constraints

- Locality

Why does this matter?

Flexibility, Performance and Availability

- Multiple Programming Models
- Central components do less → *scale better*
- Easier High-Availability (e.g., RM vs AM)

System	Jobs/Day	Tasks/Day	Cores pegged
Hadoop 1.0	77k	4M	3.2
YARN	125k (150k)	12M (15M)	6 (10)

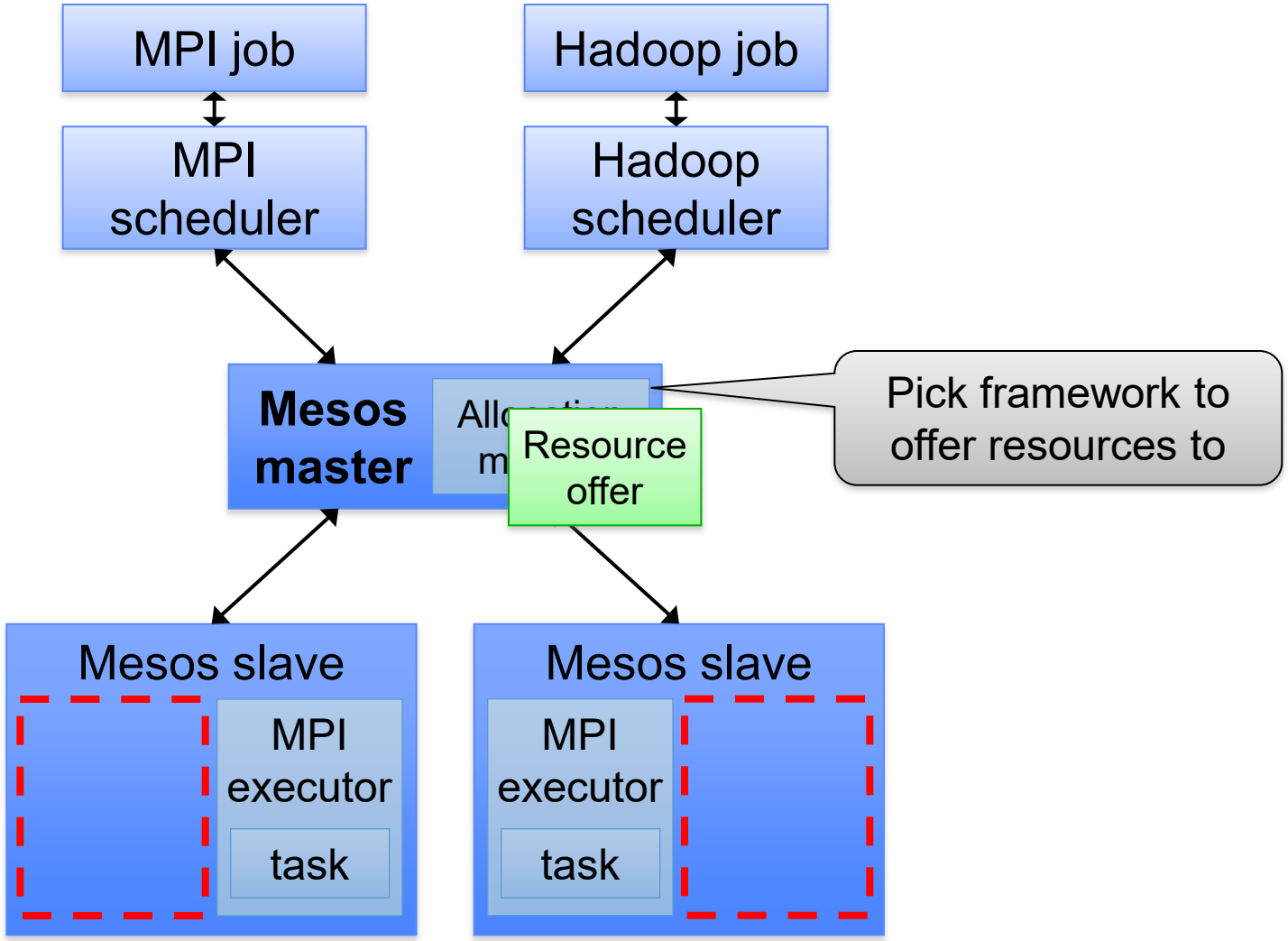
Anything else?

Anything else?

Maintenance, Upgrades, and Experimentation

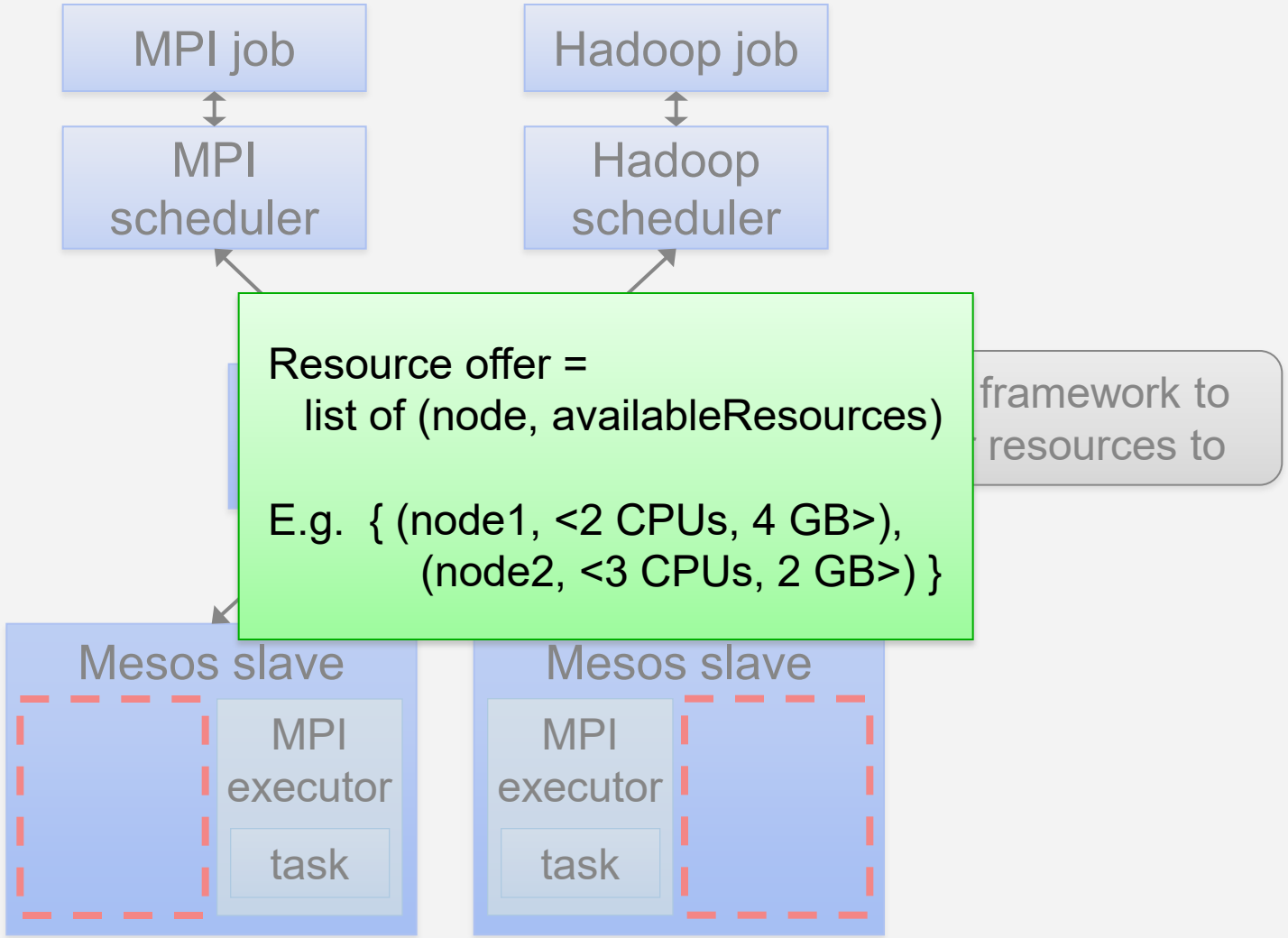
- Run with multiple framework versions (at one time)
- *Trying out a new idea is as simple as launching a job*

Mesos Architecture



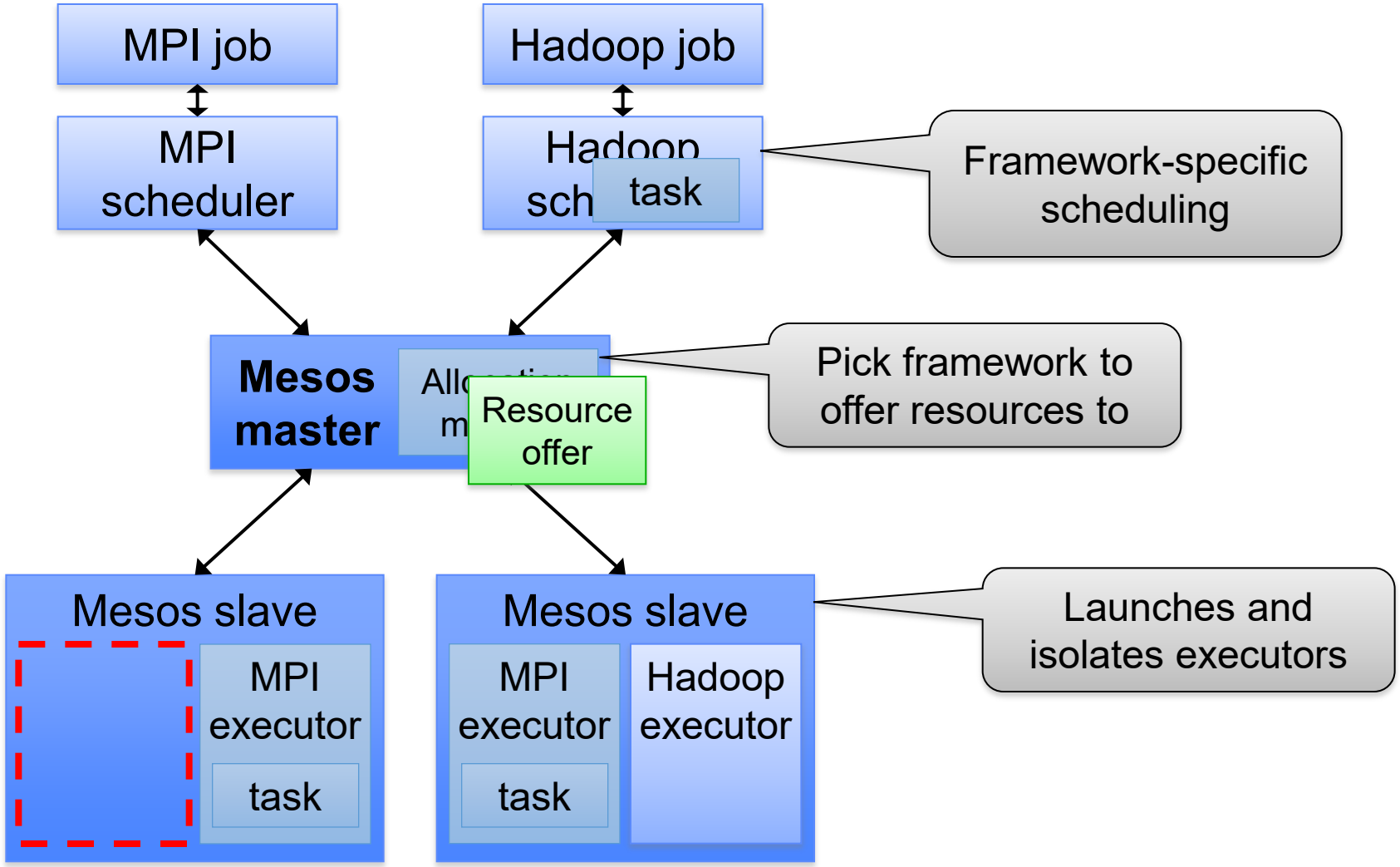
Slide courtesy of Mesosphere

Mesos Architecture



Slide courtesy of Mesosphere

Mesos Architecture



Slide courtesy of Mesosphere

Apache Mesos

Framework offers

- Arbitrates among services
- Services have their own models for tenants
- Supports authentication plugins (working on Kerberos)
- Preemption (“inverse offers”) in progress

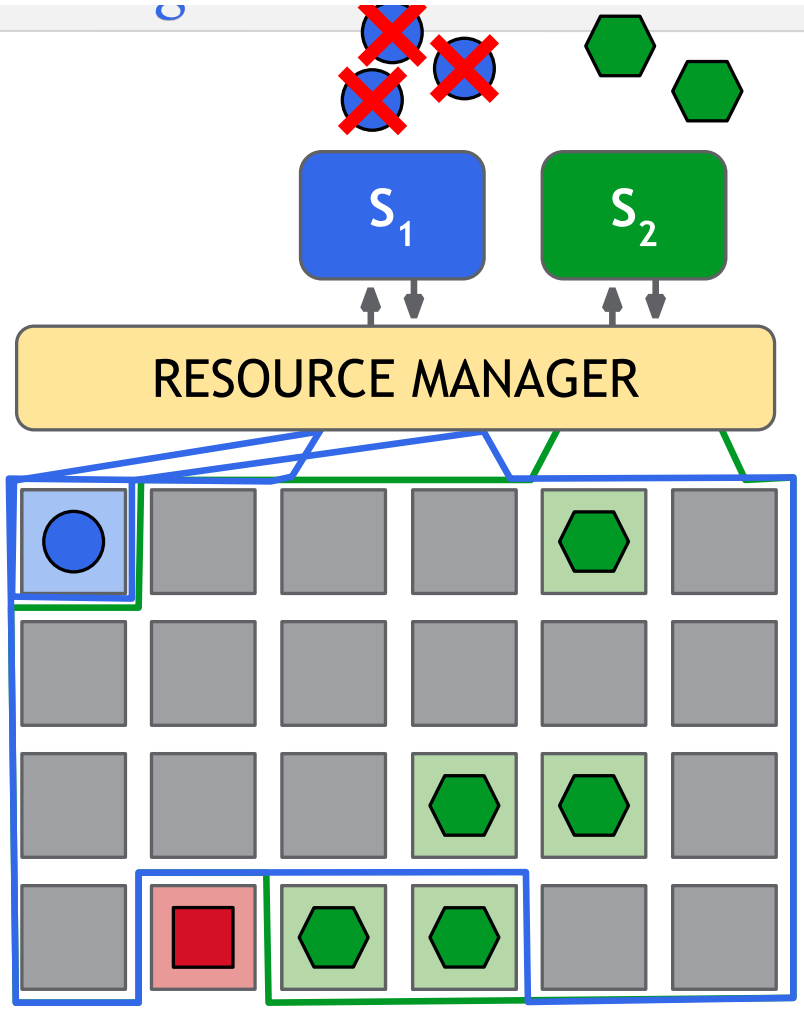
Supports resource isolation using cgroups

- CPU/memory/disk iops/disk usage

Scalable

- Largest single-cluster deployments in “high thousands” of nodes
- Tens of thousands of tasks (framework workers), tested to 50K

Omega critique of Mesos



1. Green receives offer of all available resources.
2. Blue's task finishes.
3. Blue receives tiny offer.
4. Blue cannot use it.

[repeat many times]

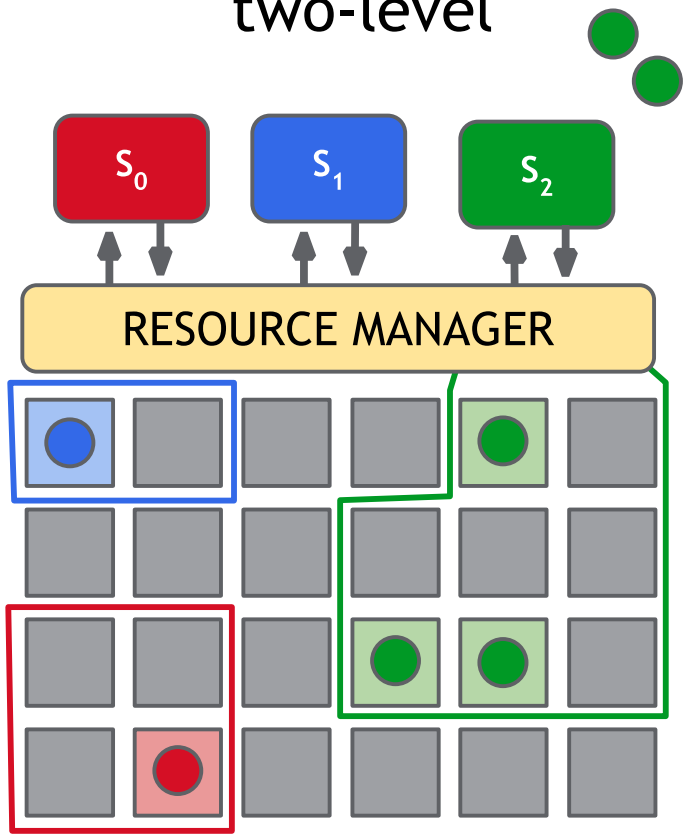
5. Green finishes scheduling.
6. Blue receives large offer.

By now, it has given up.

Slide courtesy of John Wilkes

Omega

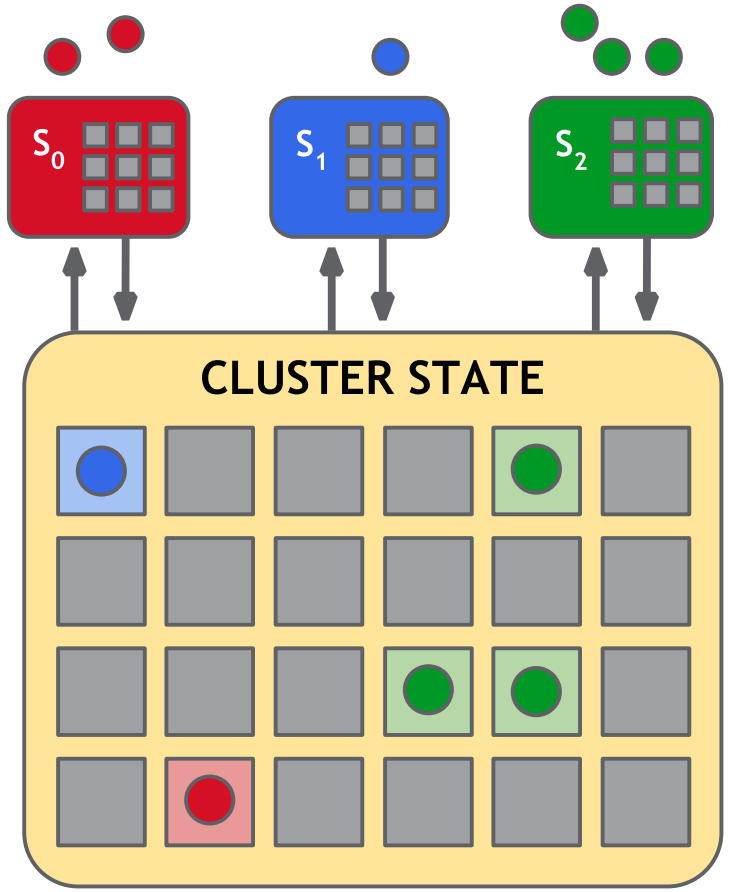
two-level



- hoarding
- information hiding

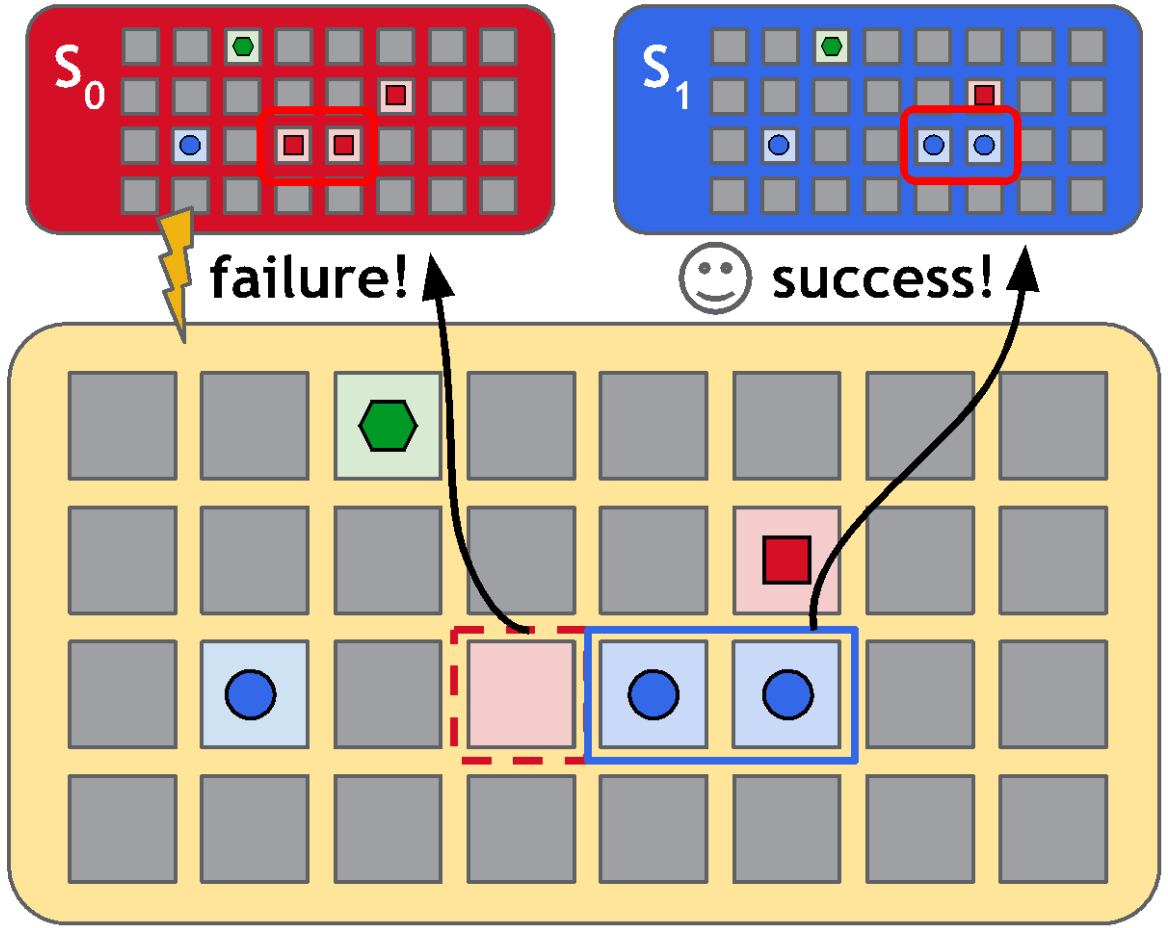
e.g. UCB Mesos [NSDI 2011]

shared-state



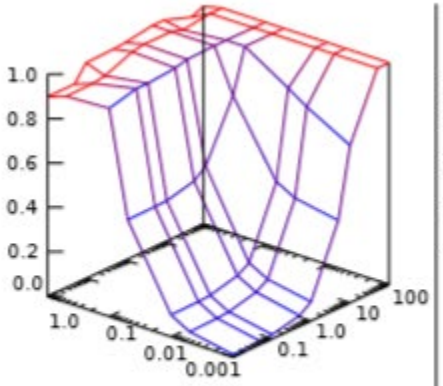
Slide courtesy of John Wilkes

Omega

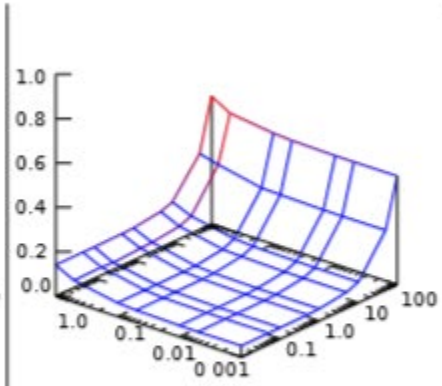


Slide courtesy of John Wilkes

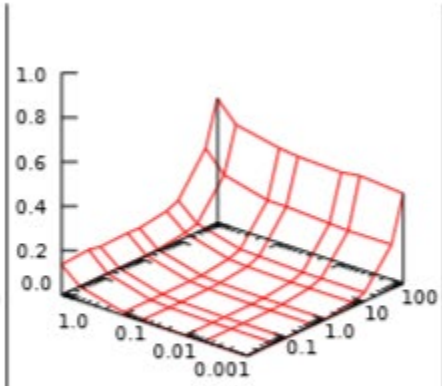
Simulation Results



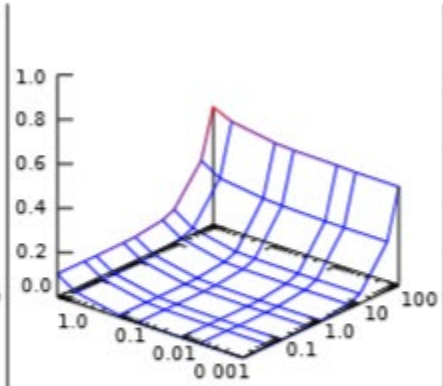
(a) Monolithic scheduler, single-path.



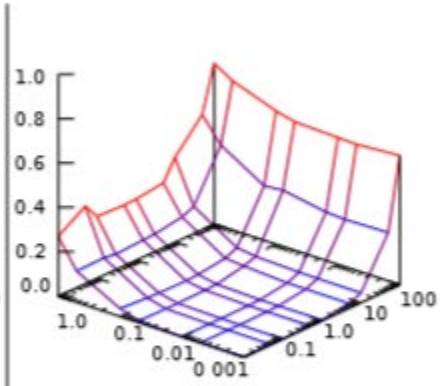
(b) Monolithic scheduler, multi-path.



(c) Two-level scheduling (Mesos).



(d) Shared-state (Omega).



(e) Shared-state, coarse, gang scheduling.

Summarizing

YARN

- Organized by *tenant/application*
- Enforces global invariants (user/job/queue)
- Request constraints matched against available resources

Mesos/Omega

- Organized by *framework/service*
- Multi-tenancy managed at second tier
- Priority managed cooperatively across frameworks

Systems: Libraries

“Although correct, it hurts me, for I don’t like to crack an egg with a sledgehammer, no matter how effective the sledgehammer is for doing so.” --Dijkstra

Building on the Cluster OS

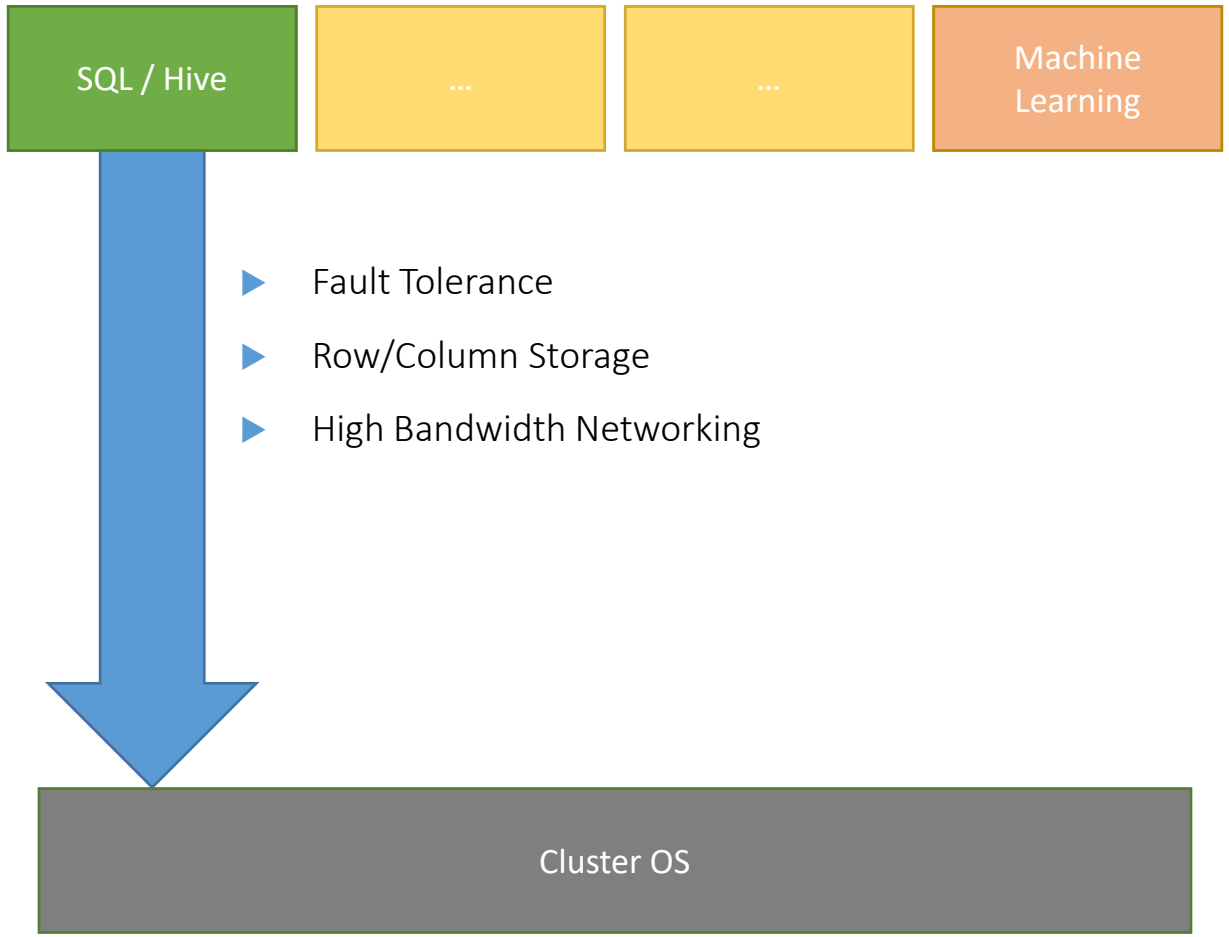
Repeated, necessary work

- Communication
- Configuration
- Data/Control flow
- Error handling/Fault tolerance
- Cross-platform

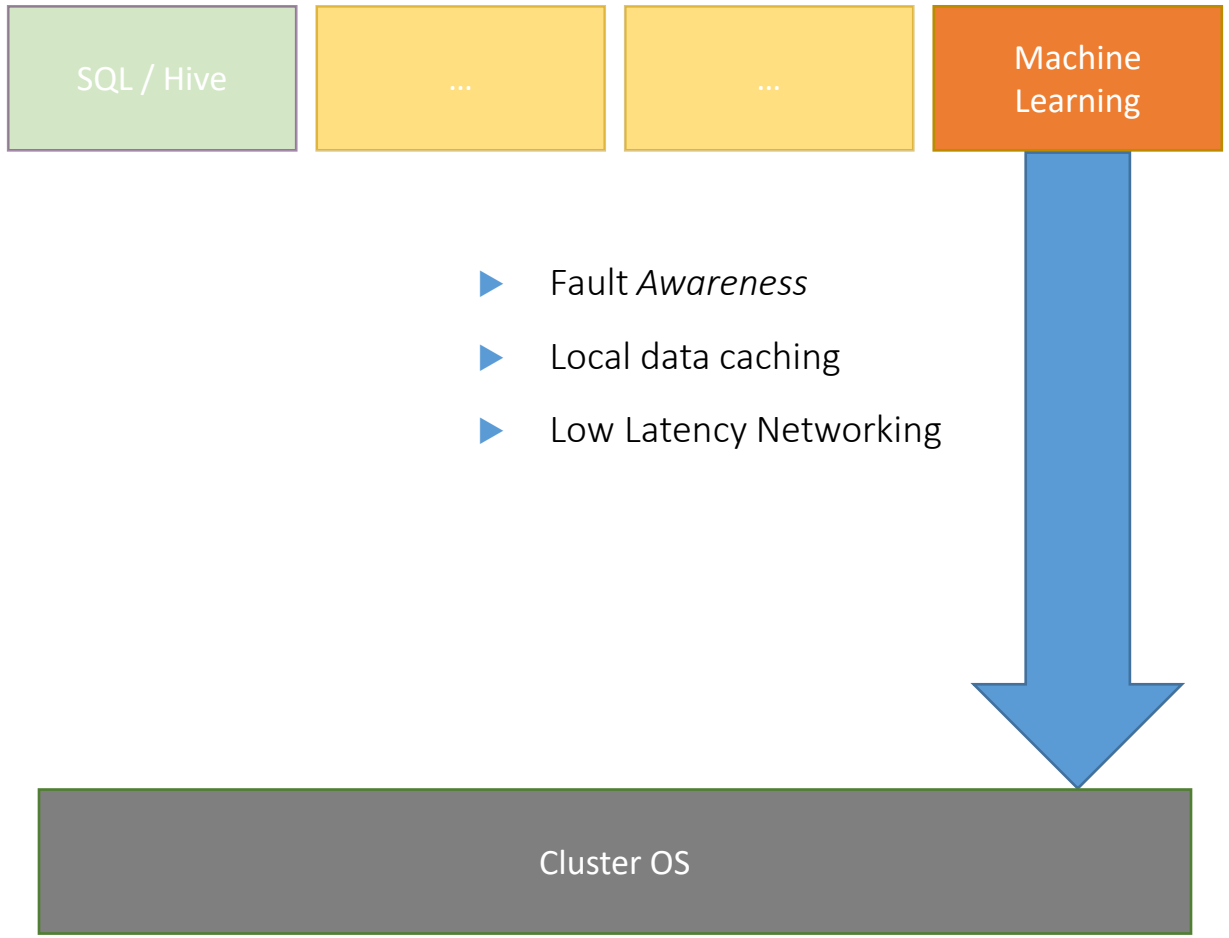
Common “better than Hadoop” tricks

- Amortize scheduling overhead
- Avoid (re)materializing data (pipelining, caching)

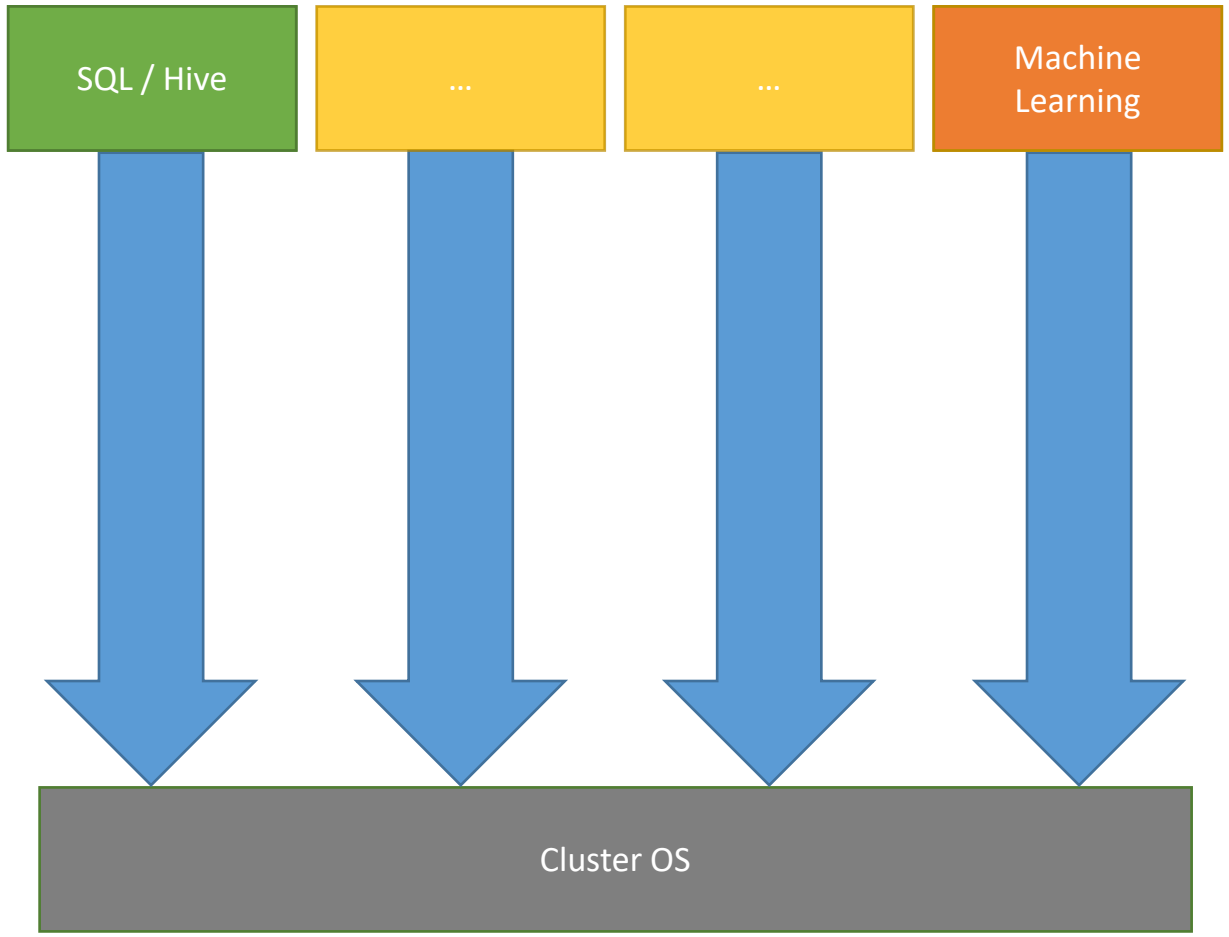
The Challenge



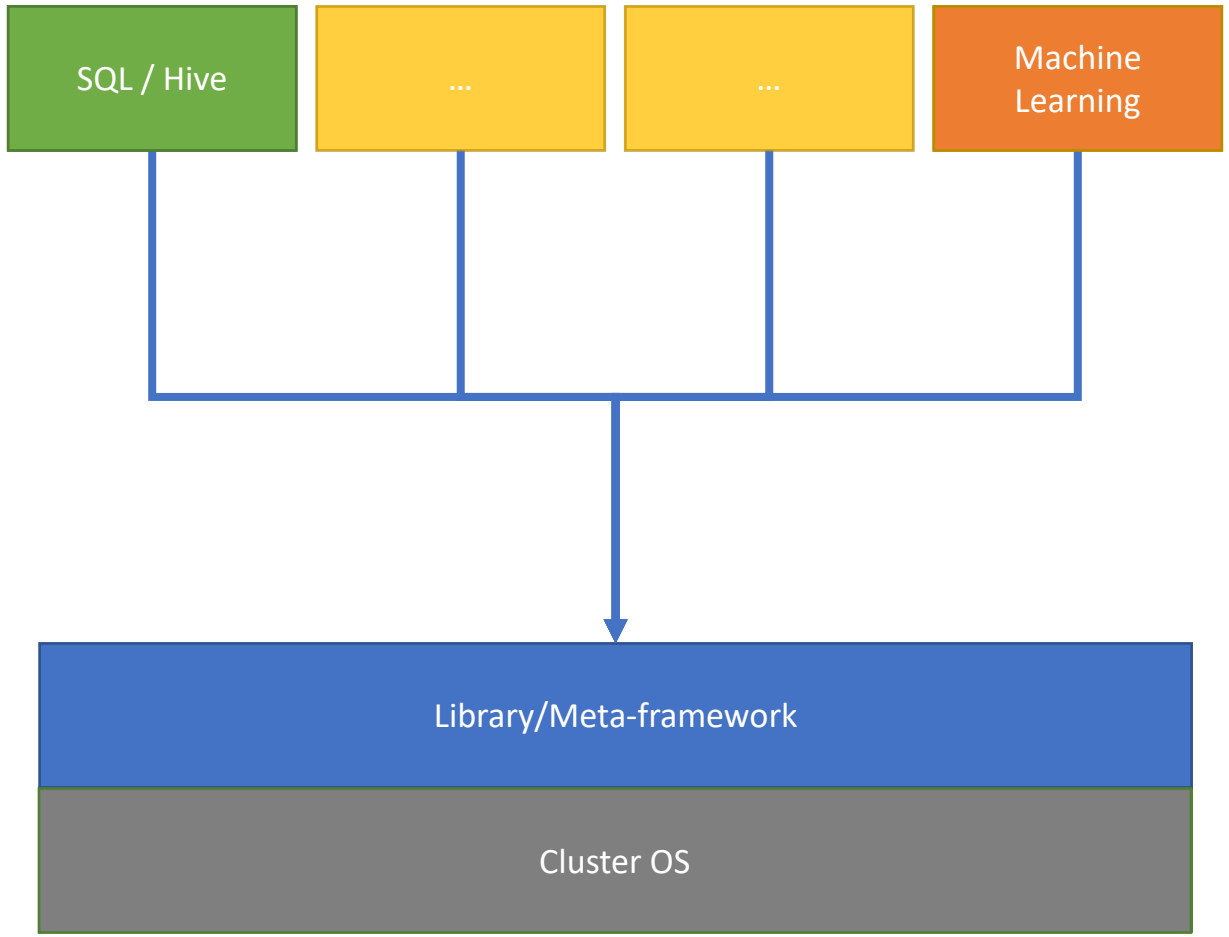
The Challenge



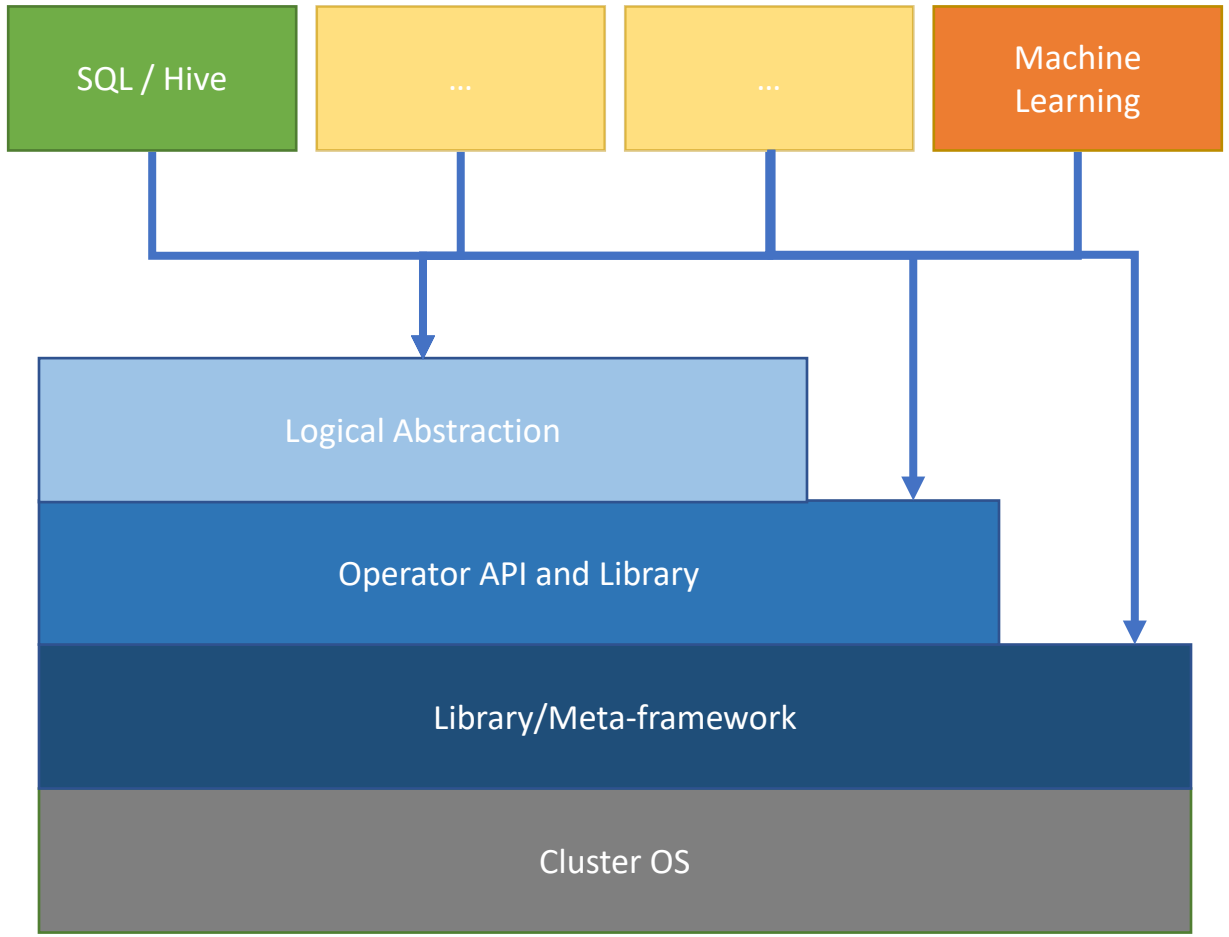
The Challenge



Library Stack



Library Stack



Systems: Libraries

Services: Apache Slider, Marathon, Chronos

Deploy existing applications

Apache Twill (incubating) (2013)

Thread-oriented abstraction on YARN

Apache REEF (incubating) (2014, VLDB14, SIGMOD15)

Event-driven control plane for decentralized dataflows

Apache Tez (2013, SIGMOD15)

Dryad-inspired DAG scheduler for Hive/Pig/Cascading

Services in shared clusters

YARN

Slider

- Start a service on YARN
- Lifecycle management
- Discovery

Oozie

- Workflow manager
- Submits workflows to YARN

Mesos

Marathon

- “init” for datacenter
- Long-running services

Chronos

- “cron” for datacenter

Apache Twill (incubating)

Familiar thread model (java.util.concurrent)

Features

Real-time logging

Command messages

Service Registry

Elastic Resource

```
public class HelloWorld {
  static Logger LOG = LoggerFactory.getLogger(HelloWorld.class);
  static class HW extends AbstractTwillRunnable {
      @Override
      public void run() {
        LOG.info("Hello World");
      }
  }
  public static void main(String[] args) throws Exception {
    YarnConfiguration conf = new YarnConfiguration();
    TwillRunnerService runner =
        new YarnTwillRunnerService(conf, "localhost:2181");
    runner.startAndWait();
    TwillController controller = runner.prepare(new HW()).start();
    Services.getCompletionFuture(controller).get();
  }
}
```

Apache REEF

Intuition

- Provide a "stdlib for BigData" distributed applications
- Modular design (application diversity)
- Support multiple resource managers (YARN, Mesos, processes, VMs)

Core REEF

- Control flow pattern centralized, event-driven "driver", and distributed "evaluators"
- Separate the notion of task from evaluator (enables container re-use)

REEF Services

- Data caching in evaluator, Naming and Communications, Coordination and Consistent Logging, Configuration validation, etc..

REEF

Client

Job1

Resource Manager

Scheduler

Injection-based checkable configuration

Event-based Control flow

User control flow logic

User data crunching logic

Retains State!

services

Task

Evaluator

NodeManager

Name-based

Driver

REEF RT

NodeManager

services

Task

Evaluator

Fault-detection

NodeManager

Libraries

Wake

- Networking abstraction
- Event-based programming/remoting
  - Static checking of event flows
  - Aggressive JVM event inlining
- Static subset of Rx

Tang

- Configuration as dependency injection

Corfu

- Currently deployable as a service using REEF
- Integration as a service to applications for distributed data structures

Apache REEF Summary

Control Flow is centralized in the Driver

Evaluator, Tasks configuration and launch

Error Handling is centralized in the Driver

All exceptions are forwarded to the Driver

All APIs are asynchronous

Caching / Checkpointing / Group communication

Example Apps Running on REEF

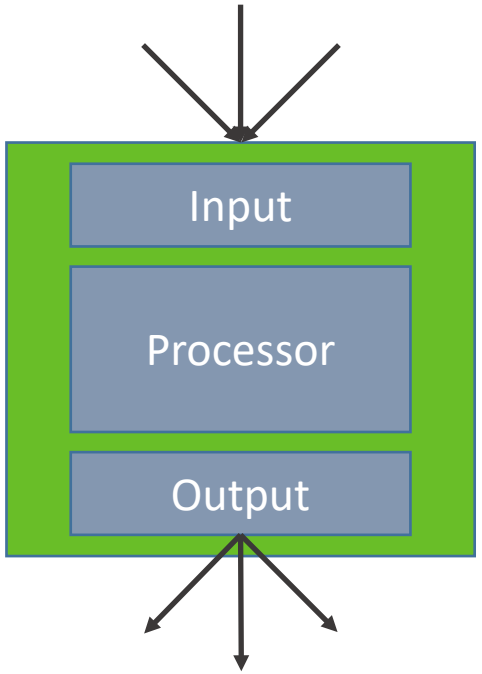
MR, Asynch Page Rank, ML regressions, PCA, distributed shell, Azure Streaming Analytics,…

Cross-platform, multi-language

YARN, Mesos

Java, .NET

Apache Tez



DAG API

- Defines the structure of the data processing and the relationship between producers and consumers
- Defines complex data flow pipelines using simple graph connection APIs. Tez expands the logical DAG at runtime

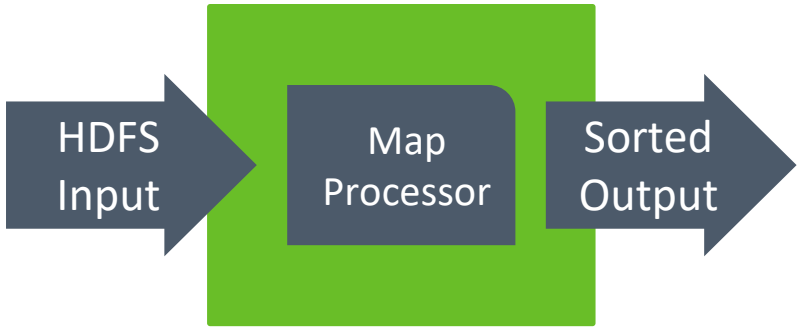
Runtime API

- Defines the interfaces using which the framework and app code interact with each other
- App code transforms data and moves it between tasks

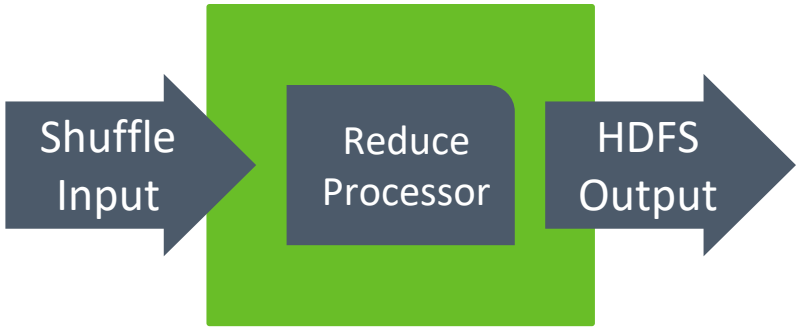
Slide courtesy of Hortonworks

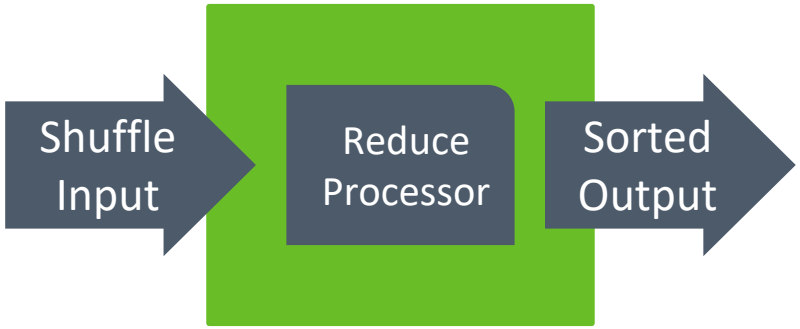
Library of Inputs and Outputs

Classical 'Map'



Classical 'Reduce'





Intermediate 'Reduce' for Map-Reduce-Reduce

- **What is built in?**
  - Hadoop InputFormat/OutputFormat
  - SortedGroupedPartitioned Key-Value Input/Output
  - UnsortedGroupedPartitioned Key-Value Input/Output
  - Key-Value Input/Output

Slide courtesy of Hortonworks

WordCount

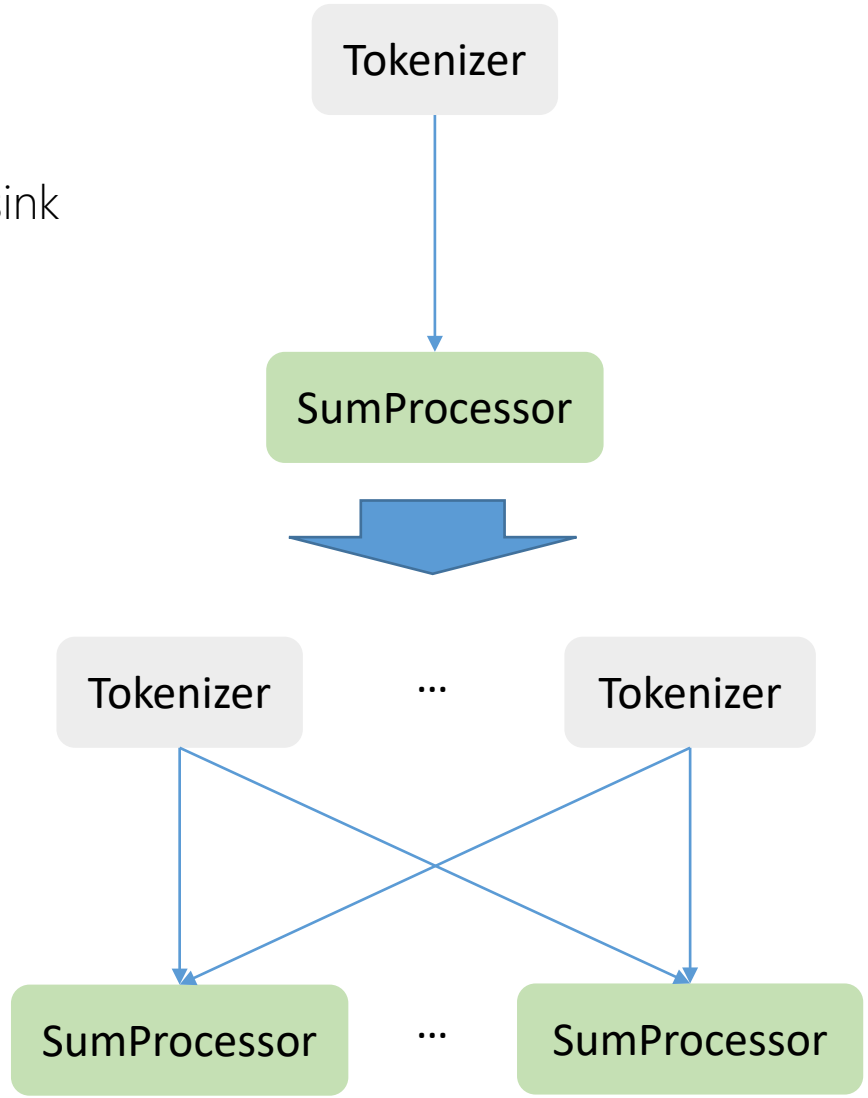
```
DataSourceDescriptor dataSource = /* specify input data */
DataSinkDescriptor dataSink =     /* specify output location */

// create vertices
Vertex tokenvertex = Vertex.create(...).addDataSource(INPUT, dataSource);
Vertex sumvertex   = Vertex.create(...).addDataSink(OUTPUT, dataSink);

// create edge
EdgeProperty edge = /* Inititialize Edge with Hash partitioner */

DAG dag = DAG.create("WordCount");
dag.addVertex(tokenizerVertex)
   .addVertex(summationVertex)
   .addEdge(Edge.create(tokenvertex, sumvertex, edge);
```

Define source/sink

Define vertices

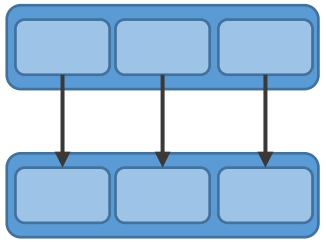
Define edges

Wireup



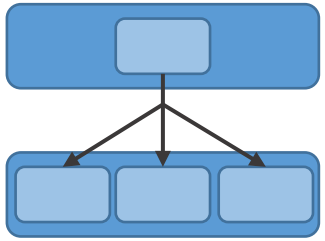
Slide courtesy of Hortonworks

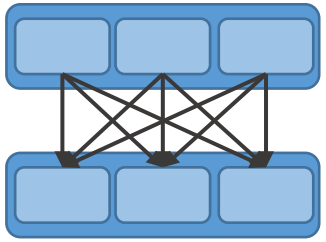
Edge Types



One-To-One
Data from producer *i* routes to consumer *i*

Broadcast
Data from producer routes to all consumer tasks





Scatter-Gather
Producers partition data into shards.
Shard *i* from all producers routes to consumer *i*

Slide courtesy of Hortonworks

Systems: libraries

Different surface from each library

- **Tez** DAG exec, storage libraries, coordination, YARN-only
- **REEF** stdlib, difficult plumbing for distrib apps, cross-platform
- **Slider** service discovery, gradually expose YARN surface
- **Marathon/Chronos** treat services in datacenter as daemons

Service as a Service

- Apache Myriad (incubating) project runs YARN as a Mesos service

MapReduce is a library

- In both YARN and Mesos, MapReduce is library code
- Though deployment is different

Systems: Applications

“So… what would you say you DO here?”

Apache Pig

Procedural dataflow language

- Earlier versions omit control flow, modularity features (loops, funcs, branches)
- Rich support for UDFs, Macros (DEFINE, IMPORT)
- Builtin Functions (load/stor, eval, relational, math, string, datetime, arithmetic)

Supports multiple execution engines

- Tez, MapReduce

Rule-based optimizer

- Safe transformations (push-up filter, push-down flatten, pruning, etc.)

Interactive mode

- "Grunt" shell for interactive analysis
- Explore samples of data

Sample PigLatin Queries

Query 1: Get the list of web pages visited by users whose age is between 20 and 29 years.

```
USERS = load 'users' as (uid, age);
USERS_20s = filter USERS by  age >= 20 and age <= 29;
PVs = load 'pages' as (url, uid, timestamp);
PVs_u20s  = join USERS_20s by uid, PVs by uid;
```

Query 2: PageRank

```
good_urls = FILTER urls BY pagerank > 0.2;
groups = GROUP good_urls BY category;
big_groups = FILTER groups BY COUNT(good_urls)>106;
output = FOREACH big_groups
         GENERATE category, AVG(good_urls.pagerank)
```

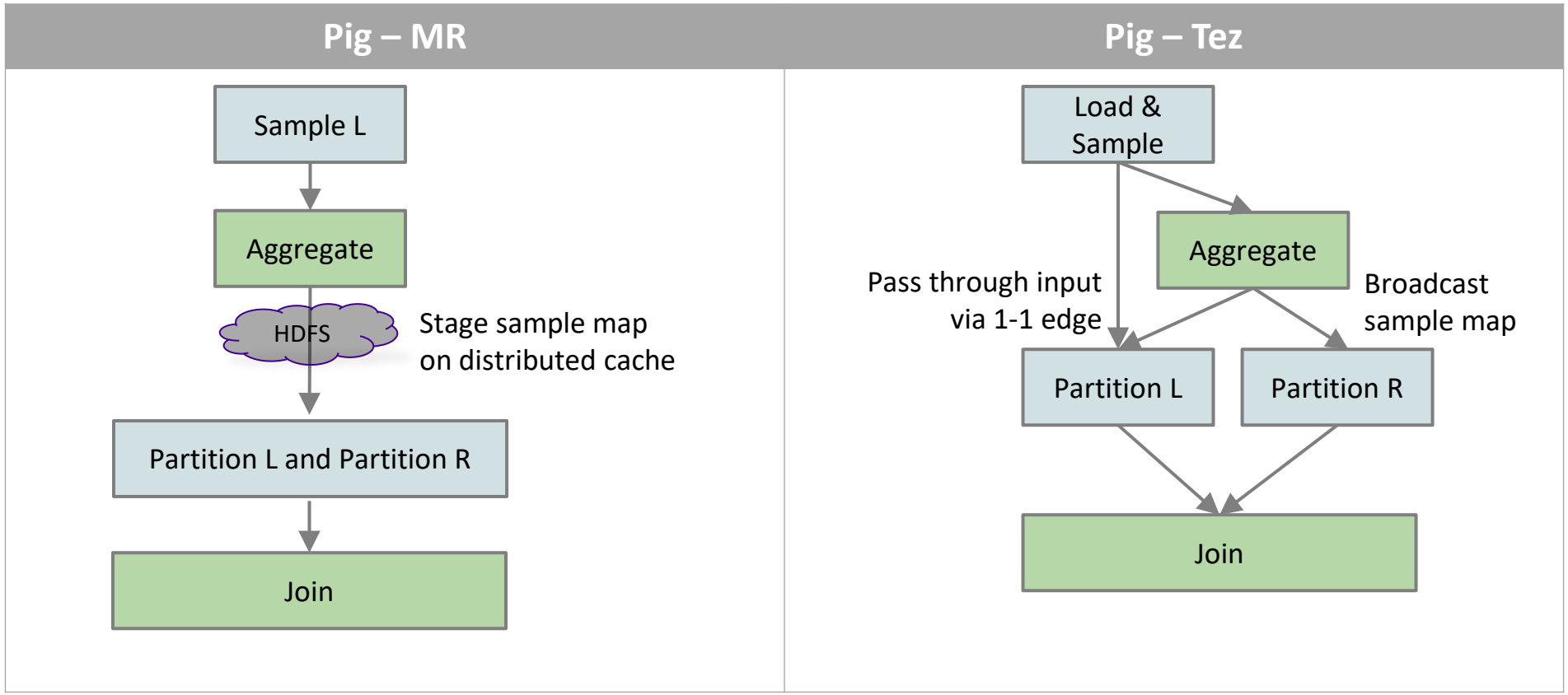
PigLatin

SQL

```
SELECT category, AVG(pagerank)
FROM urls WHERE pagerank > 0.2
GROUP BY category HAVING COUNT(*) > 106
```

Example: Pig Skewed Join

```
l = LOAD 'left' AS (x, y);
r = LOAD 'right' AS (x, z);
j = JOIN l BY x, r BY x
    USING 'skewed';
```



Slide courtesy of Hortonworks

Apache Hive

Users write a variant of SQL (HiveQL)

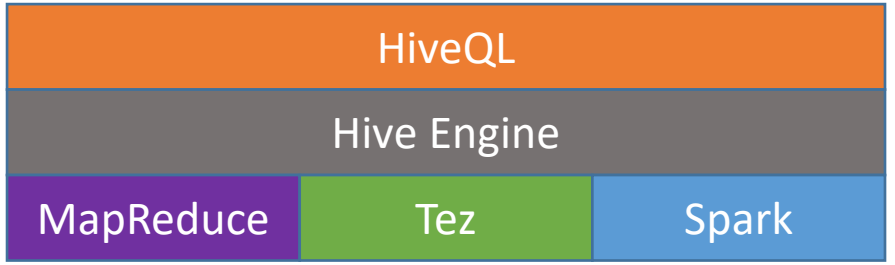
- Used by projects that may not use the Hive engine
- Interactive shell (Beeline)

HiveServer2 (metaserver)

- Stores schema, statistics; coordinates transactions
- Used by other frameworks (e.g., MapReduce, Pig)

ACID transactions

- INSERT, UPDATE, DELETE
- Only for bucketed tables in ORC format
- No BEGIN, COMMIT, ROLLBACK
- Snapshot-level isolation
- Streaming ingest



Stinger Initiative

Vectorized Query Execution

- Process blocks of rows at a time
- Very few function calls/conditional branches in inner loop
- Set of known types, subset of built-in func and UDFs

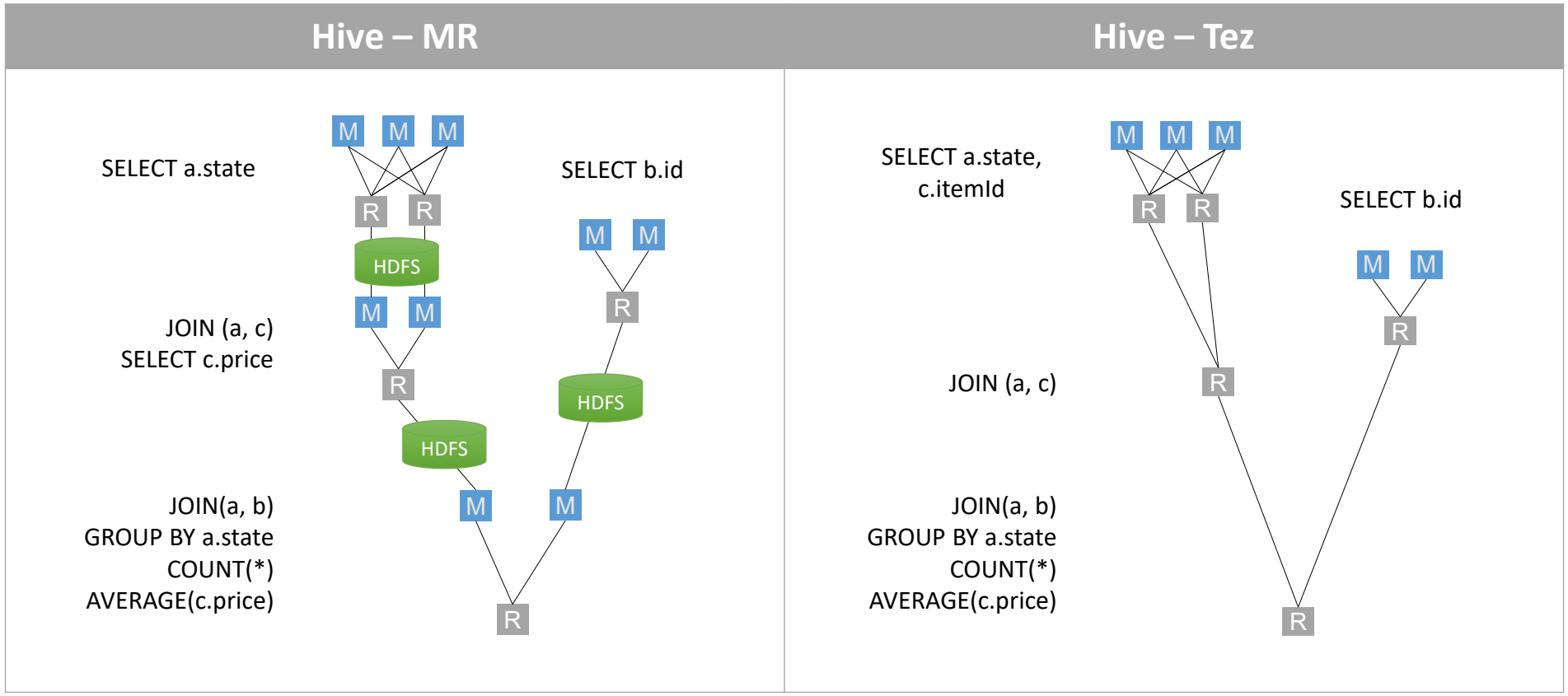
Optimized Record Columnar File (ORCFile)

- Replaces Hive RCFile
- Efficient integer transcoding, zigzag encoding, run-length encoding (RLE), etc.
- Support for push-down predicates using lightweight indexes
- Metadata statistics can be used to answer queries, avoid decompression
- Additional compression can significantly decrease storage costs

Hive-on-MR vs. Hive-on-Tez

```
SELECT a.x, AVERAGE(b.y) AS avg
  FROM a JOIN b ON (a.id = b.id) GROUP BY a
UNION SELECT x, AVERAGE(y) AS AVG
  FROM c GROUP BY x
ORDER BY AVG;
```

Tez avoids unneeded writes to HDFS



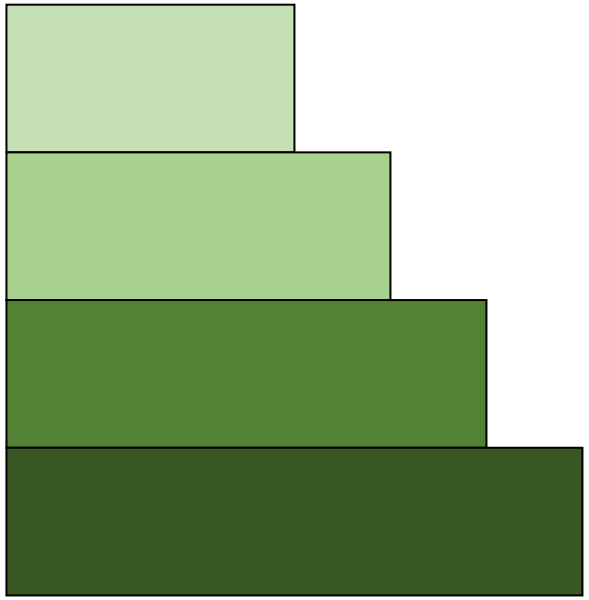
Slide courtesy of Hortonworks

Large application ecosystem

Dryad DAG computations
Storm stream processing
Giraph graph-processing, Bulk Synchronous Parallel (Pregel)
Flink (Stratosphere) parallel, iterative computations
Crunch library for MapReduce/Spark pipelines
Hama bulk-synchronous parallel
MRQL SQL-like query language over MapReduce, Hama, Spark
Spark interactive, in-memory, iterative
Drill low-latency, SQL query engine
Impala (Llama) scalable, interactive, SQL-like query

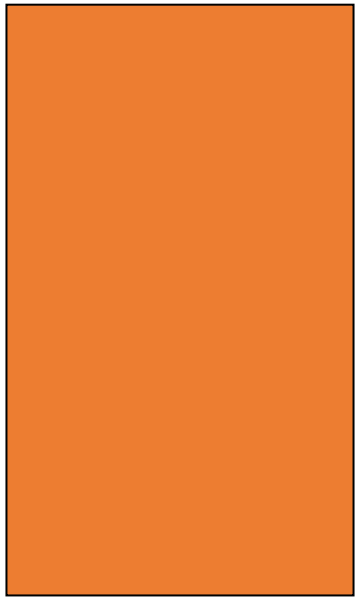
BigData second generation

Systems



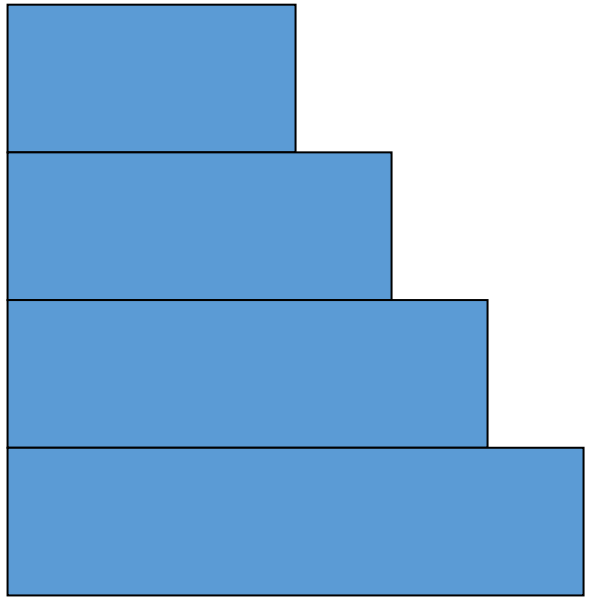
Pig, Hive, Cascading, Crunch, Sawzall, FlumeJava,...
Calcite, Summingbird, Corfu, ...
Tez, REEF, Twill, Chronos, Marathon, ...
YARN, Mesos, Omega, Borg, Corona, ...

Database



Impala, SCOPE, Drill, Tajo, Presto, ..

Hybrid



Spark, Asterix, Flink, ...

Database approach

“…and in the darkness bind them.”

Cloudera Impala

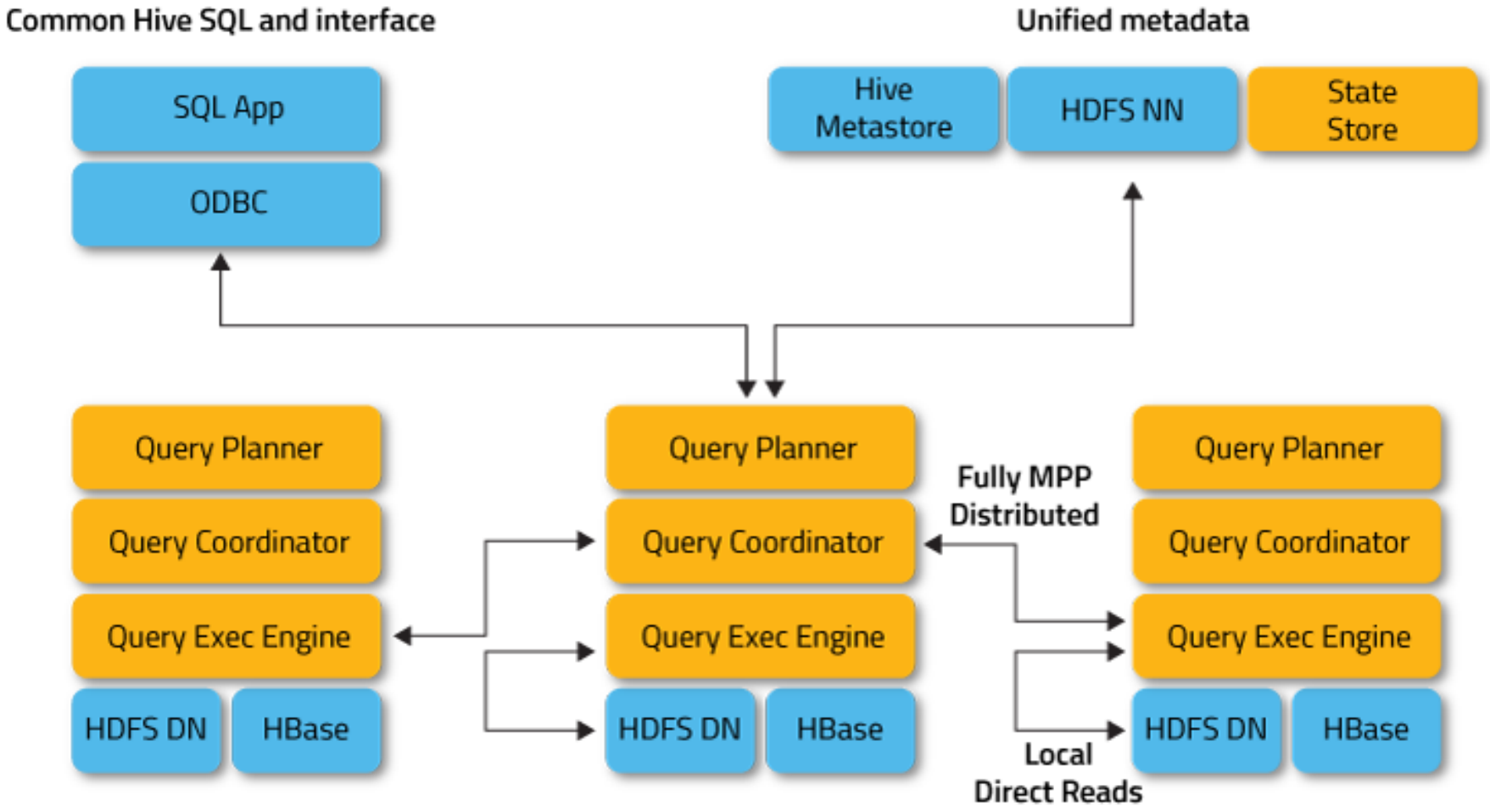
MPP Analytic Query Engine

- Inspired by Google Dremel
- Interactive workloads (lower latency, sometimes by orders of magnitude)
- Restricted SQL language (only fast operations)
- Support for scalar UDFs, user-defined aggregates (UDA)
- Support for low-level optimizations (e.g., code generation in LLVM)

Deep integration with Hadoop

- Data in HDFS/HBase, support common formats (Parquet, Avro, text, Seq, RC)
- Pin data in memory (HDFS feature)
- Daemons co-located with HDFS DataNodes
- Kerberos/LDAP authorization
- Apache Sentry (role-based auth) integration

Cloudera Impala Architecture



Cloudera Impala Performance

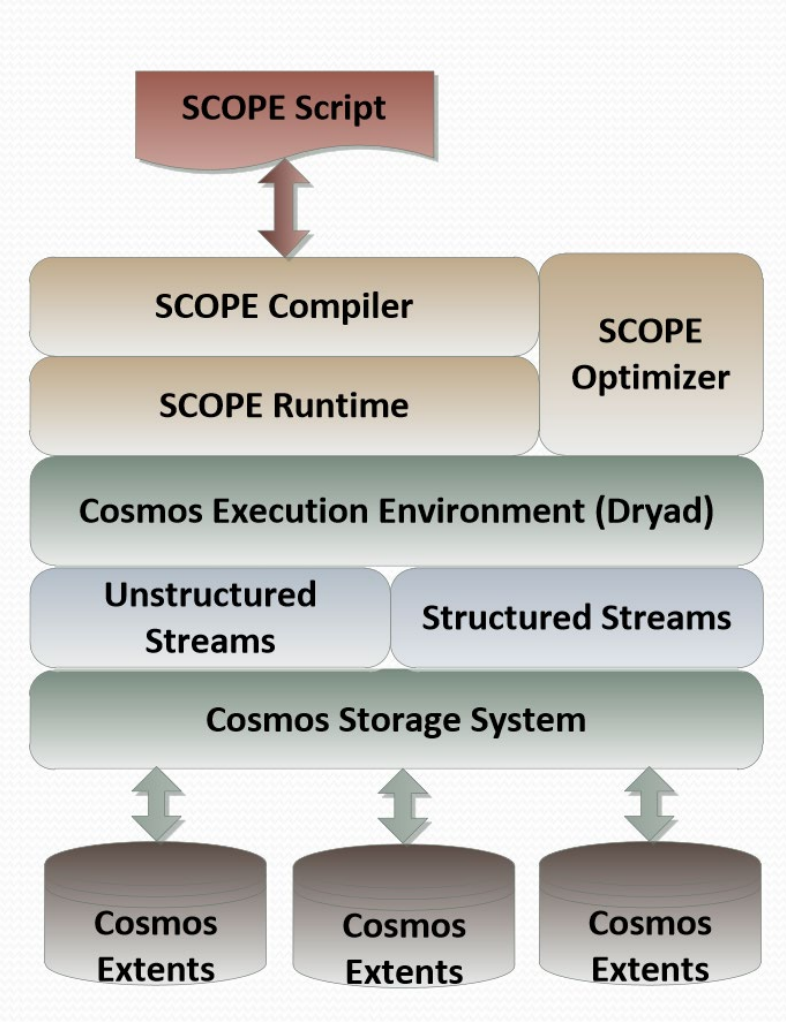
Key technical tricks

- Vertical integration (no MR)
- Intra-framework multi-tenancy
- YARN integration via Llama
- C++ and LLVM code generation
- Read-only data (no transactions)
- Lightweight fault-tolerance

Microsoft Scope / Apollo

Integrated vertical stack

- SQL-like language well integrated in .NET
- Powerful optimizer
- Dryad runtime
- Integrated scalable job scheduler
- Filesystem co-designed

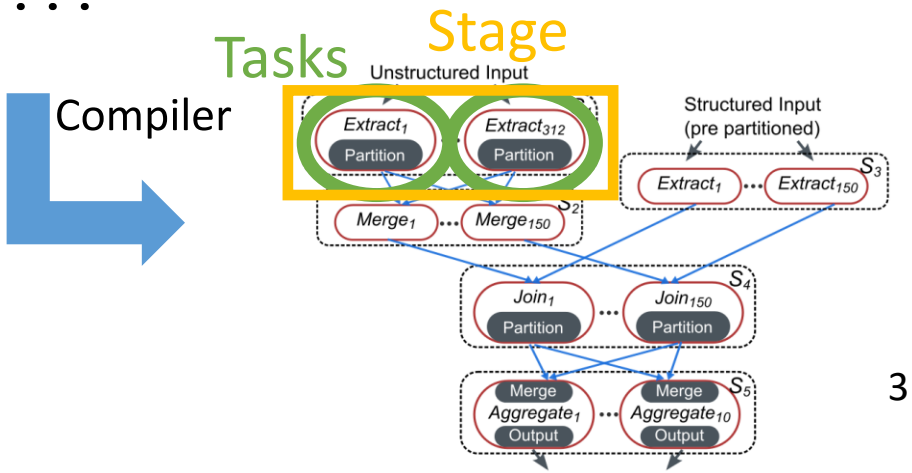


Cloud Scale Job

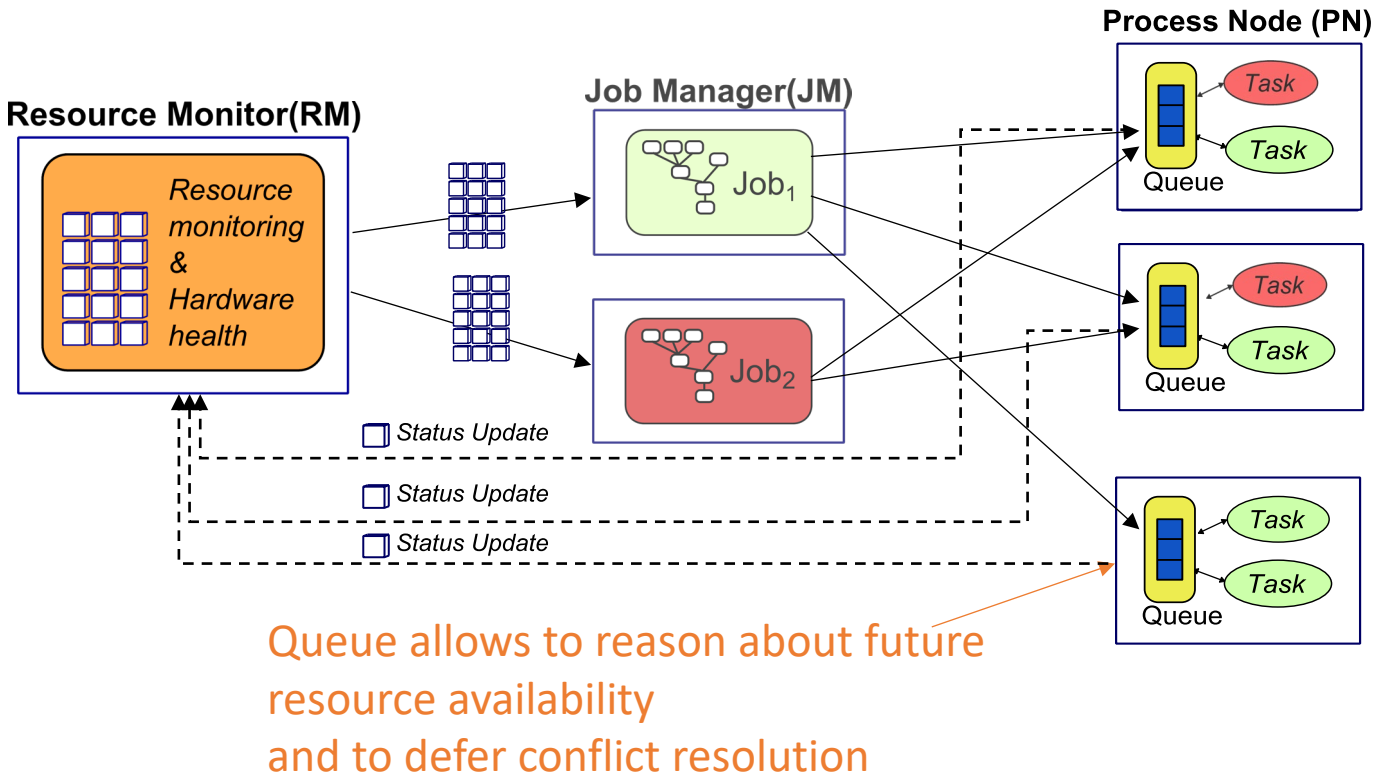
- The job query plan is represented as a DAG
- **Tasks** are the basic unit of computation
- Tasks are grouped in **Stages**
- Execution is driven by a scheduler

Job sample: SCOPE (VLDBJ, 2012)

```
SELECT
AVG(DateTime.Parse(latency))
AS E2ELatency,
market
FROM
QueryLatencies
GROUP BY market
...
```



Apollo Distributed Scheduler



Queue allows to reason about future resource availability
and to defer conflict resolution

Apollo Distributed Scheduler

Initial task placement

- Wait-time matrix
- Computes **expected wait time** for a given size

Speculative Execution

- Schedules duplicate exec given new information

Opportunistic back-filling

- Distributed scheduling requires static capacity allocation
- Opportunistically borrow unused resources (low-pri tasks)

→ *Memory*

↓ *CPU*

	4GB	8GB	12GB	16GB
2 core	0	10	10	16
4 core	0	10	10	16
6 core	5	10	10	16
8 core	10	15	15	25

Wait-time matrix (seconds)

Systems/Database

Systems

Pro

- OSS friendly (modular)
- Competitive (at each level)
- Agile (experimentation)

Con

- Prevents vertical integration
- Modularity often overstated

Database

Pro

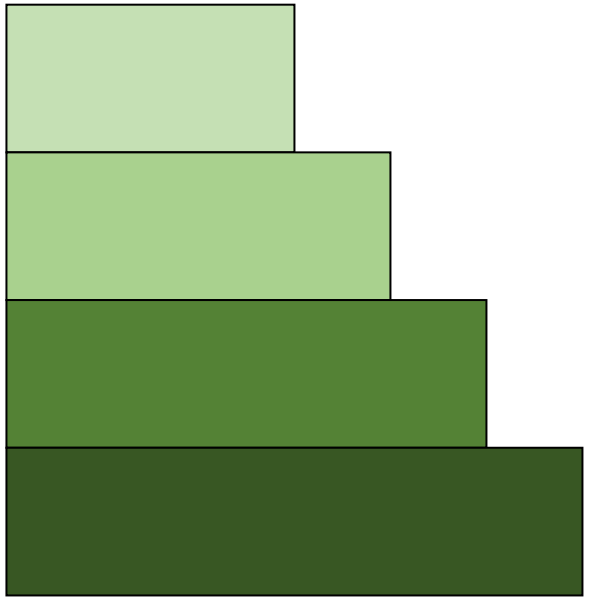
- Vertical integration
- Scalability
- System-wide changes possible

Con

- Rigid model
- Trade diversity for utilization

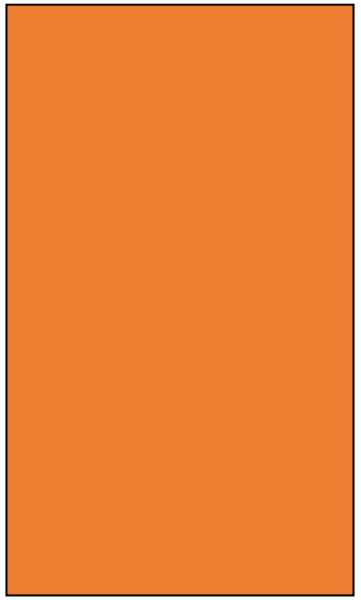
BigData second generation

Systems



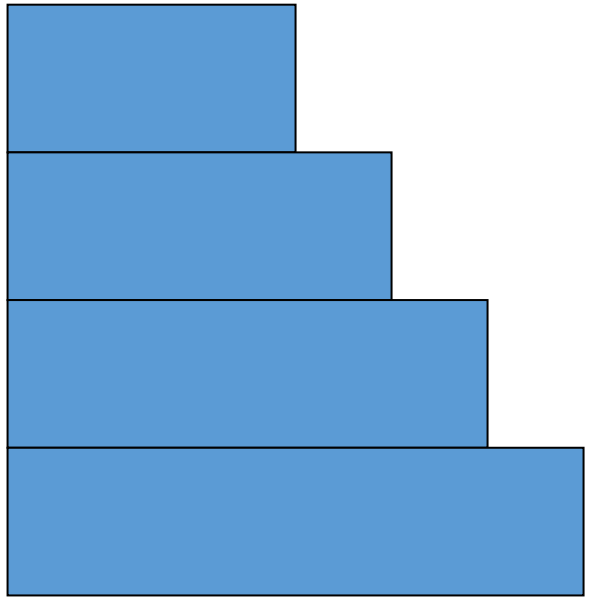
Pig, Hive, Cascading, Crunch, Sawzall, FlumeJava,…
Calcite, Summingbird, Corfu, …
Tez, REEF, Twill, Chronos, Marathon, …
YARN, Mesos, Omega, Borg, Corona, …

Database



Impala, SCOPE, Drill, Tajo, Presto, …

Hybrid

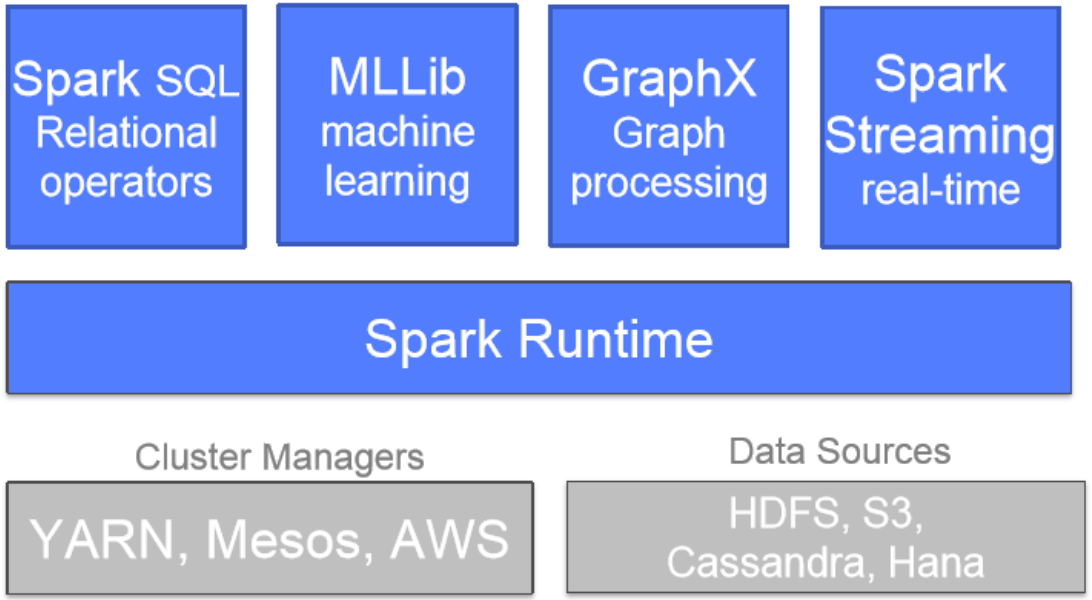


Spark, Asterix, Flink, …

Hybrid

Layered, like an onion

Apache Spark



Originally built for iterative workloads

- Machine learning jobs
- Cache data in reusable containers

Resilent Distributed Datasets (RDD)

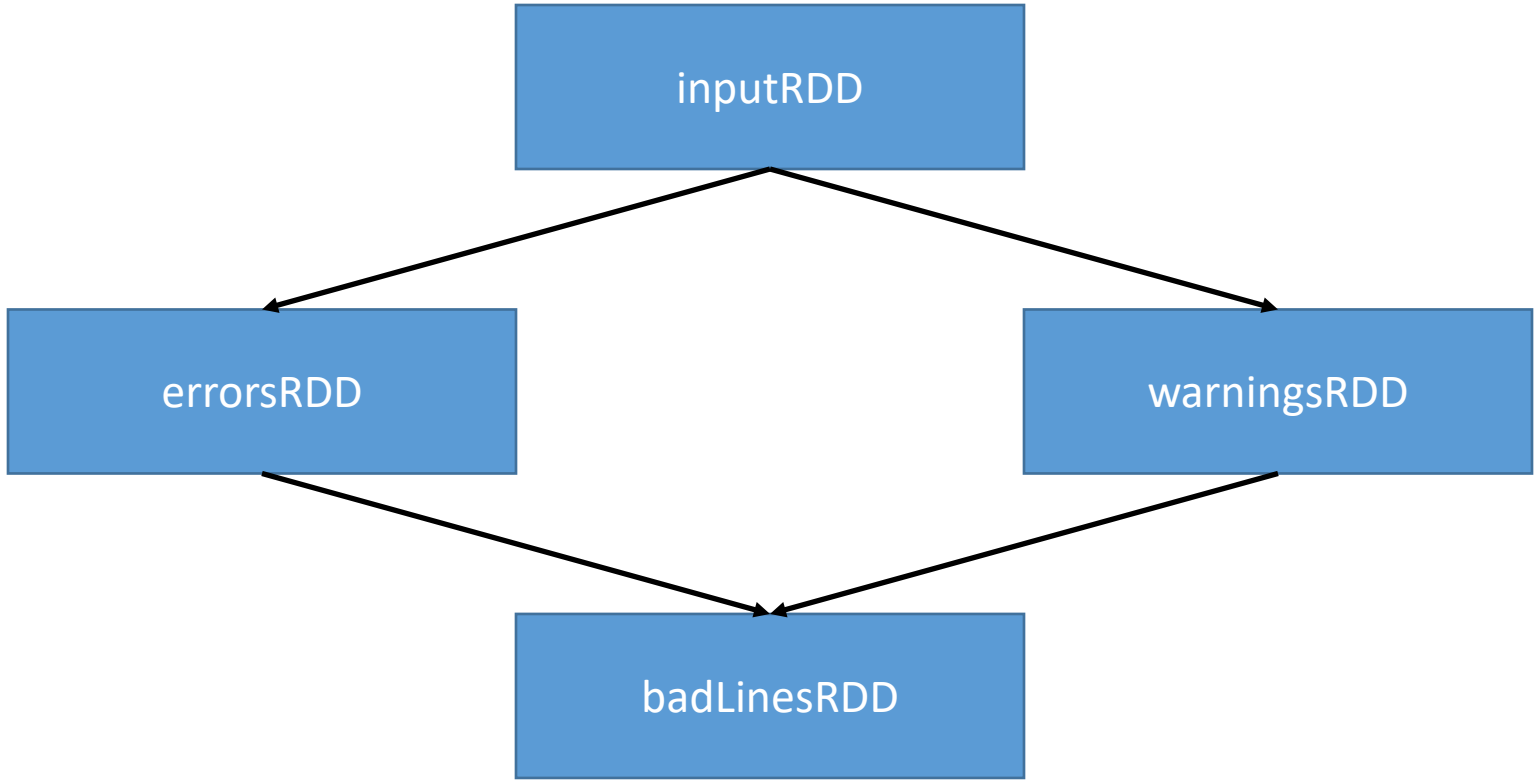
- Immutable, partitioned collection of objects
- Partitions, dependencies, compute, [partitioner], [locality]
- May have a schema (SchemaRDD/DataFrames)

DSL

- Originally in Scala, has python, Java bindings
- *Transformations* are closed over RDDs
- *Actions* force eval of transformations to return output

Example Transformation (Scala)

```
val inputRDD    = sc.textFile("log.txt")
val errorsRDD   = inputRDD.filter(lambda x: "error" in x)
val warningsRDD = inputRDD.filter(lambda x: "warning" in x)
val badLinesRDD = errorsRDD.union(warningsRDD)
```



Example Actions

```
// ...
val badLinesRDD = errorsRDD.union(warningsRDD)
println("Input had " + badLines.count() + " concerning lines")
println("Here are 10 examples: ")
badLines.take(10).foreach(println)
badLines.saveAsSequenceFile("/home/cdoug/badLines")
```

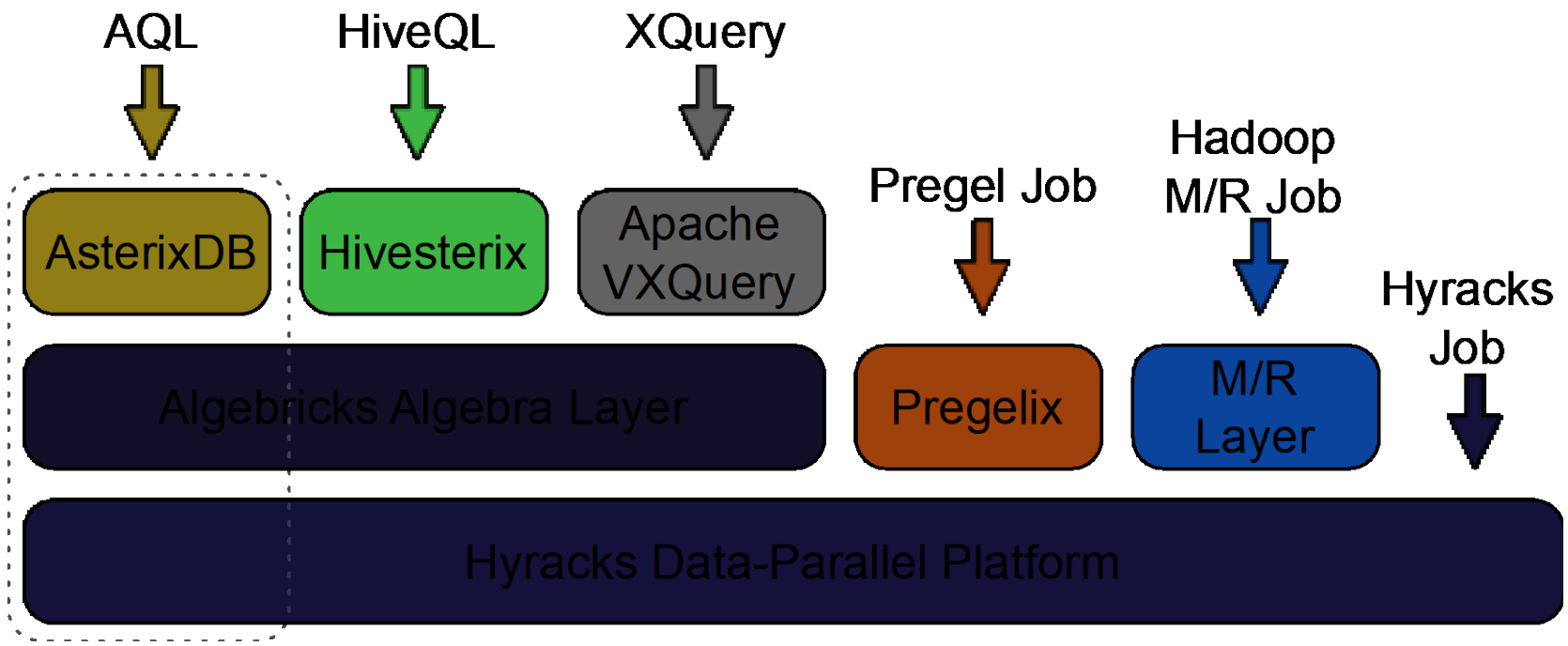
Action forces evaluation of RDD

- RDD not a dataset, but instructions for computing dataset
- May be persisted by user request or high-level framework
- Faults automatically recoverable from lineage
- Operations automatically chained/combined
- Partitioning encoded in framework

Mixing SparkSQL and MLlib

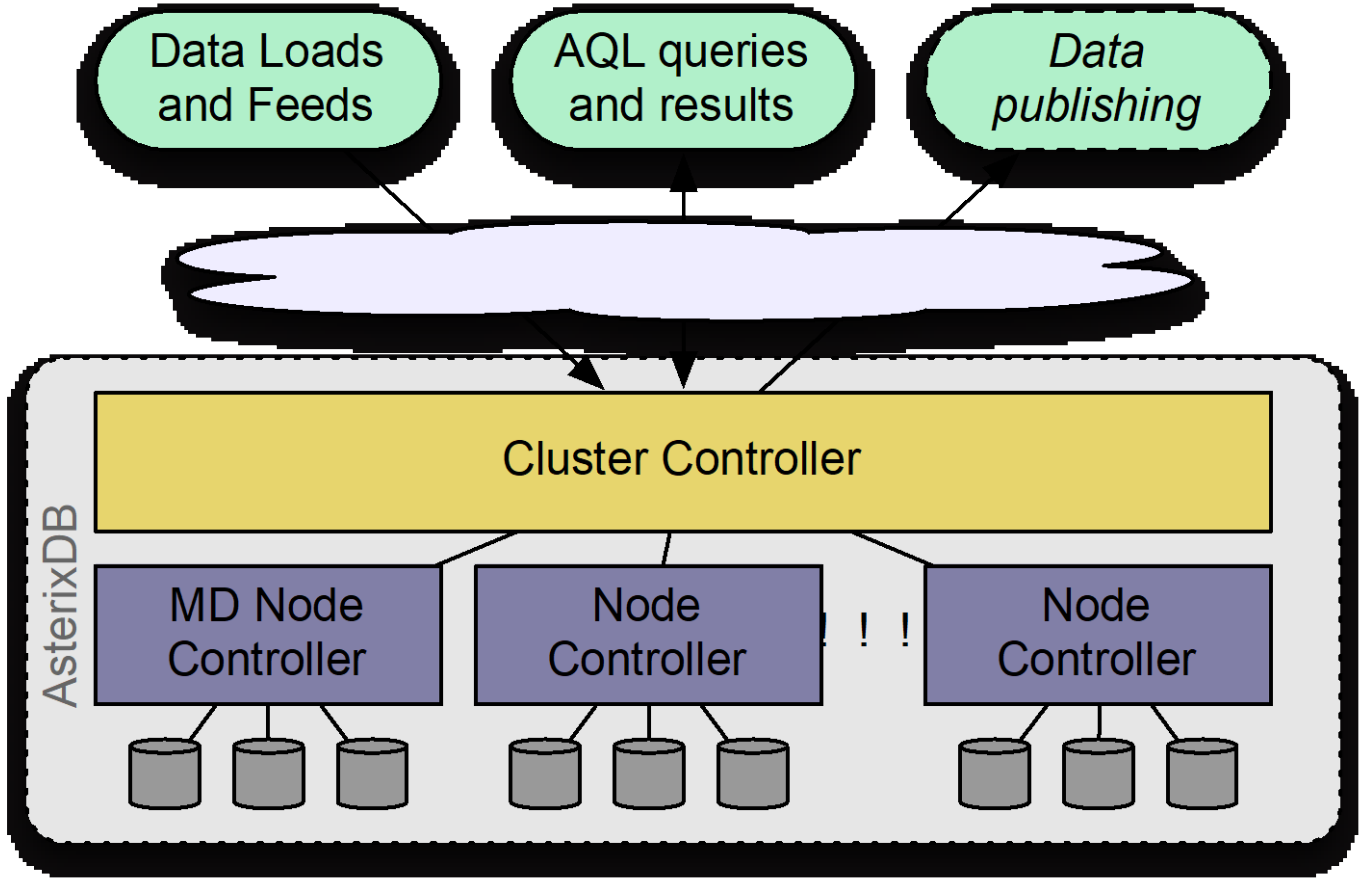
```
training_data_table = sql("""
    SELECT e.action, u.age, u.latitude, u.logitude
    FROM Users u
    JOIN Events e ON u.userId = e.userId""")

def featurize(u):
  LabeledPoint(u.action, [u.age, u.latitude, u.longitude])

// SQL results are RDDs so can be used directly in MLlib.
training_data = training_data_table.map(featurize)

model = new LogisticRegressionWithSGD.train(training_data)
```

Asterix Stack

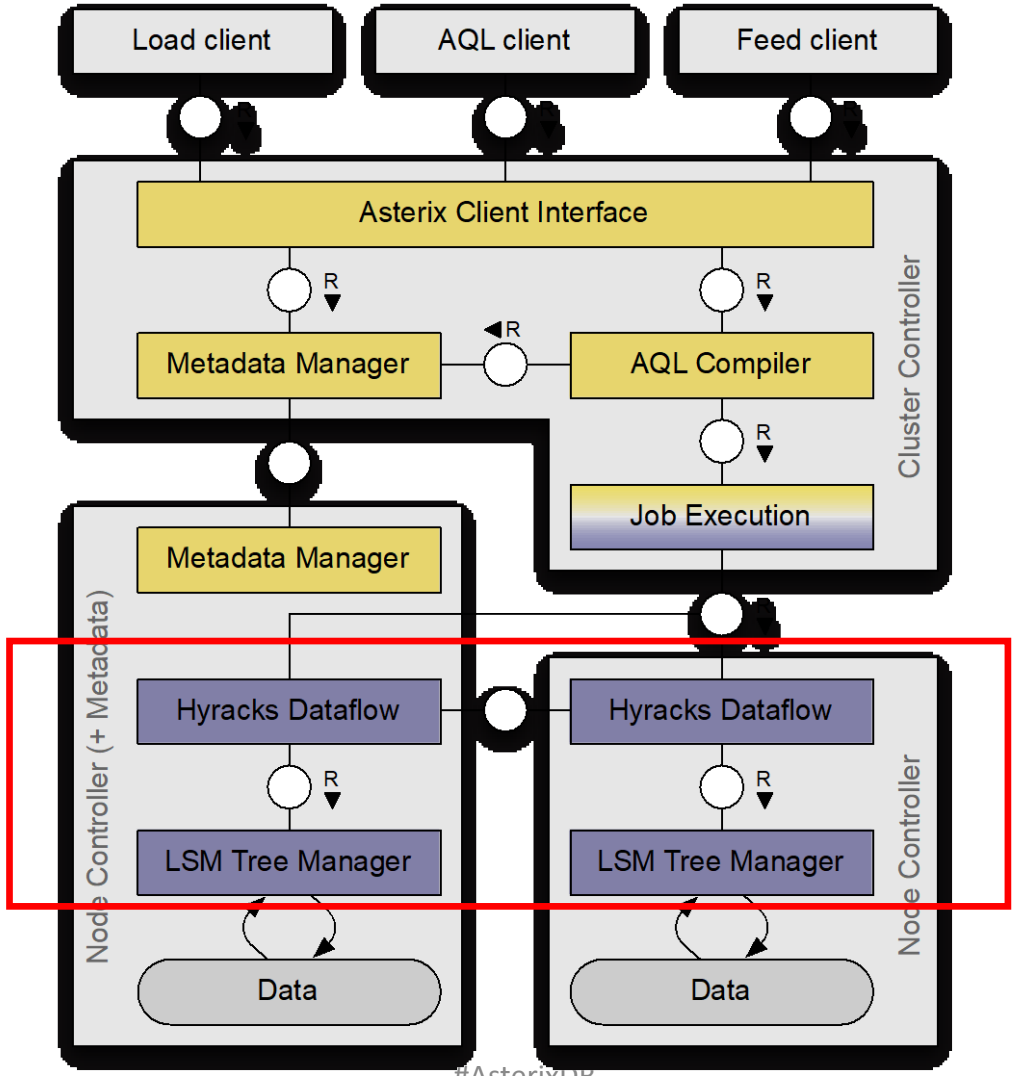


Slide courtesy of Asterix

AsterixDB System Overview



AsterixDB System Overview *(cont.)*

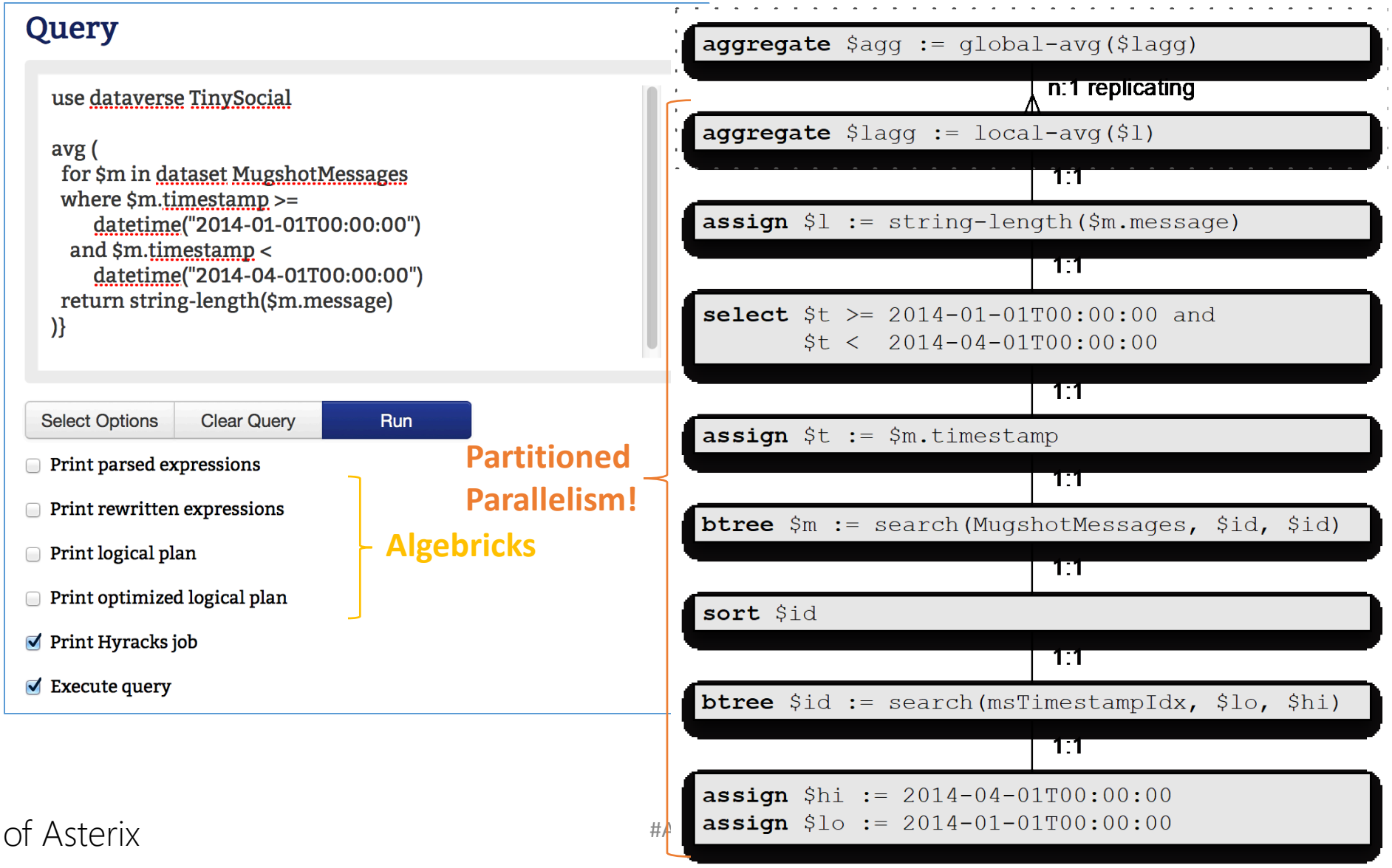


Slide courtesy of Asterix

Hyracks Runtime

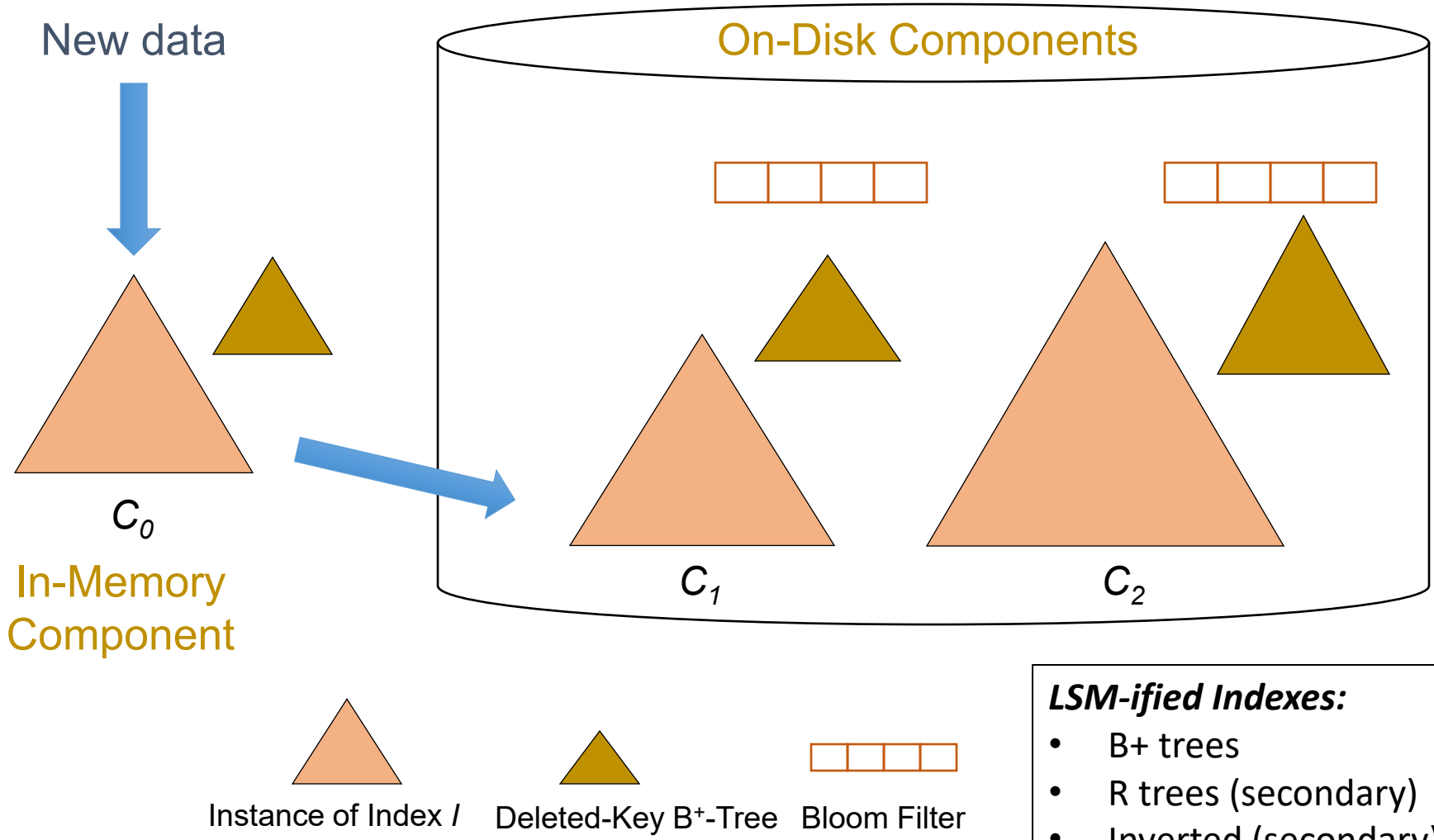
- Partitioned-parallel platform for data-intensive computing
- Job = dataflow DAG of operators and connectors
  - Operators consume and produce *partitions* of data
  - Connectors *route* (repartition) data between operators
- Hyracks *vs.* the "competition"
  - Based on time-tested parallel database principles
  - *vs.* Hadoop: More flexible model and less "pessimistic"
  - *vs.* Dryad: Supports data as a first-class citizen
- Scale-tested on a Yahoo! Labs cluster with 180 nodes (1440 cores, 720 disks)

Hyracks *(cont.)*



Slide courtesy of Asterix

LSM-Based Storage & Indexes



LSM-ified Indexes:

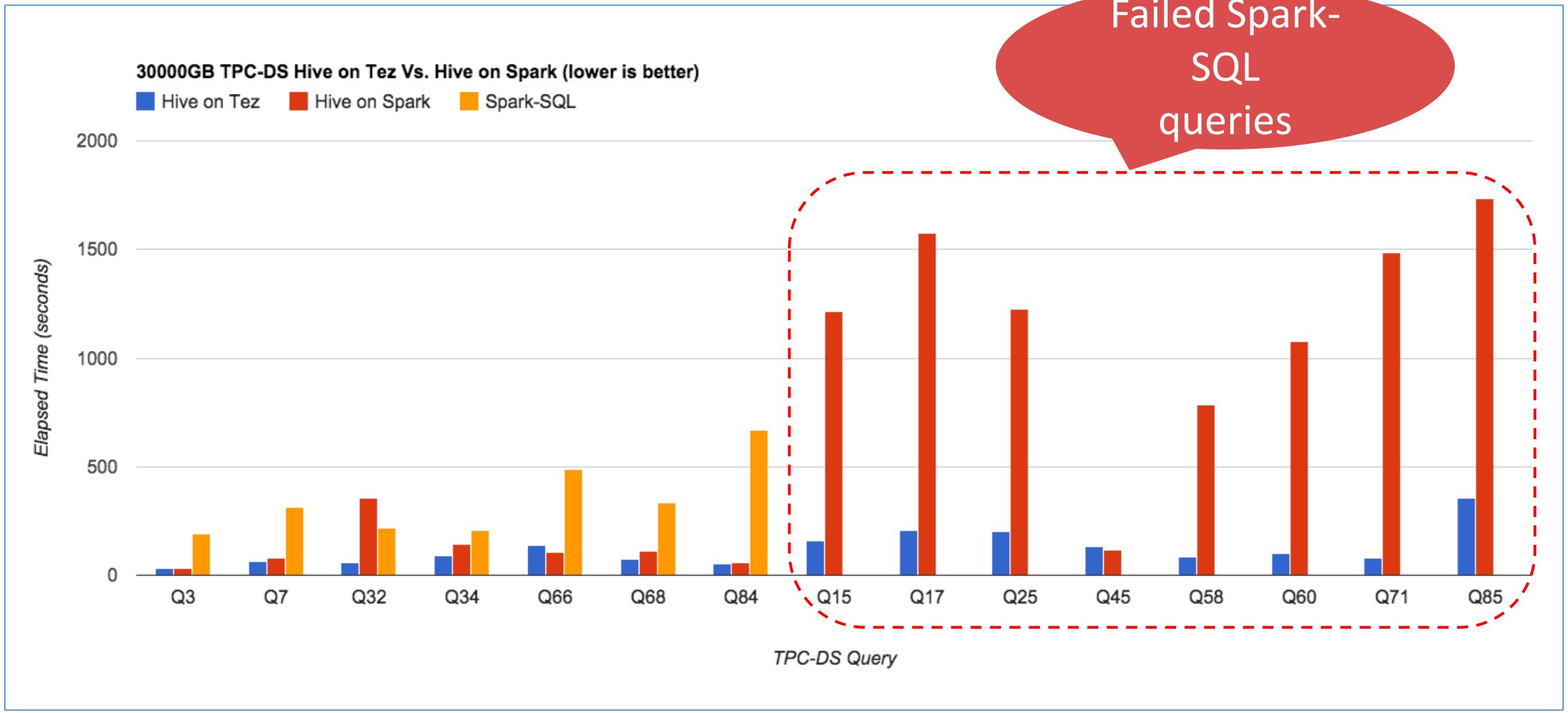
- B+ trees
- R trees (secondary)
- Inverted (secondary)

Slide courtesy of Asterix

Benchmarks and Performance

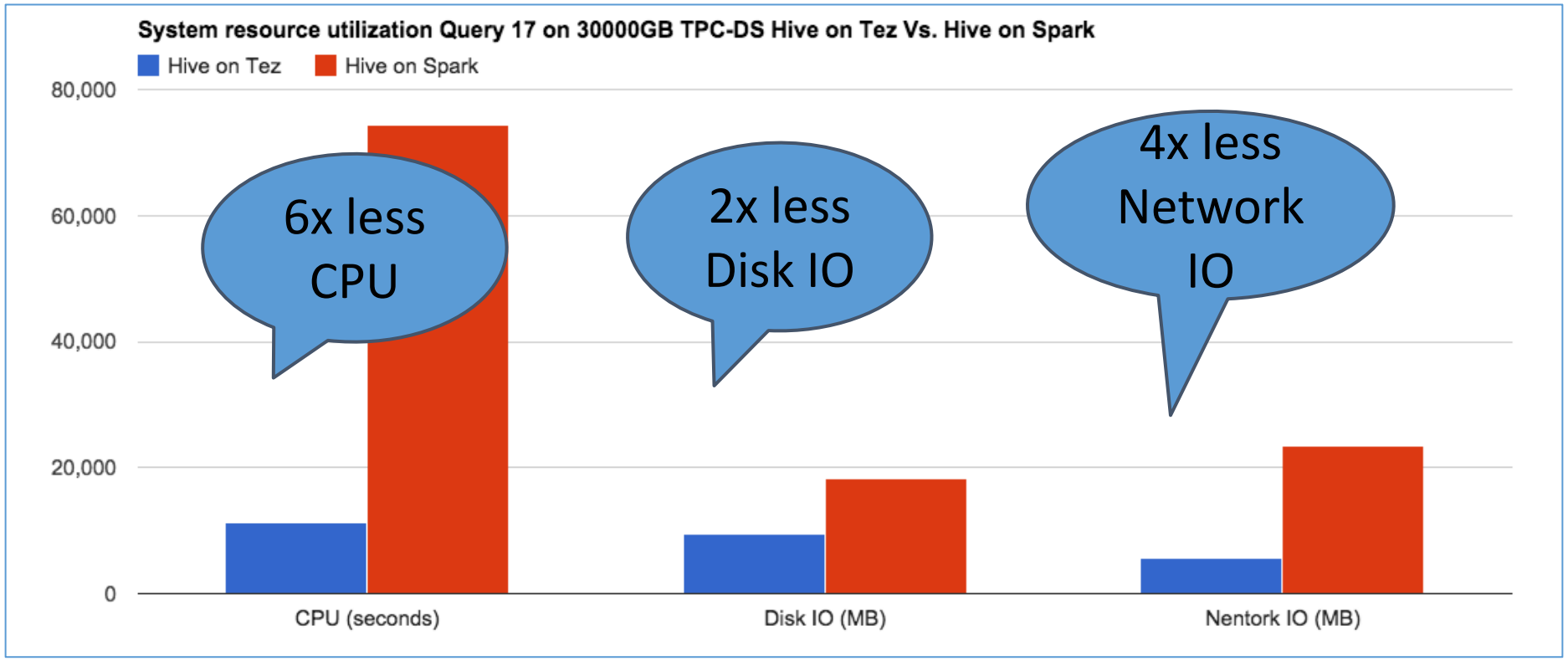
“What is best in life?”

Performance comparison : TPC-DS 30TB continued

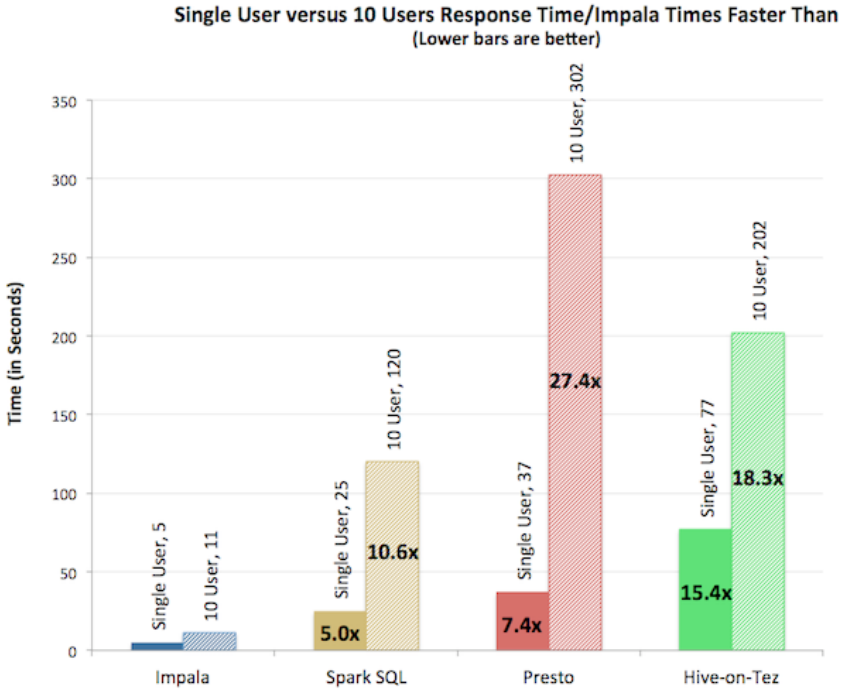


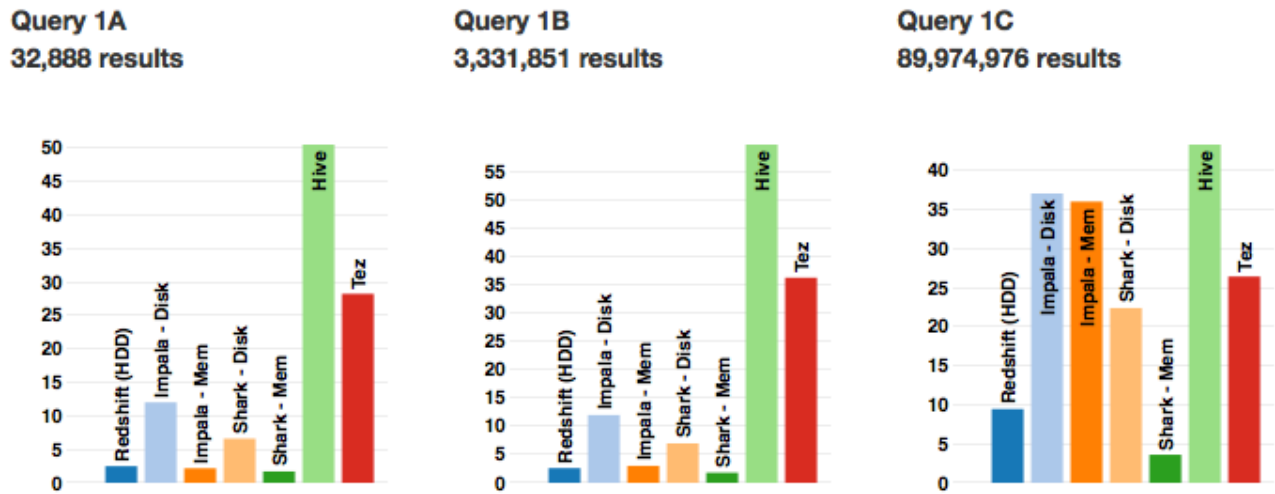
Why didn't Spark take Hive to sub-second?

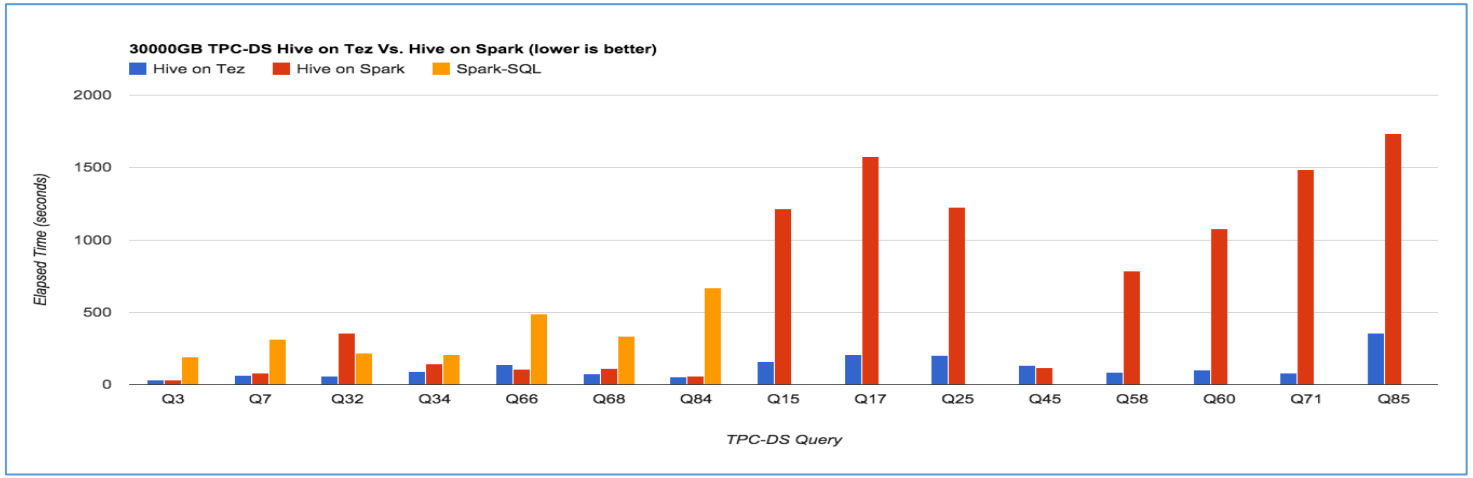
- Hive is CPU bound for most operations specially after the introduction of columnar file formats (do more with less)
- Spark consumes more CPU, Disk & Network IO than Tez for relatively large datasets
- Hive on Spark spends a lot of time translating from RDDs to Hive's "Row Containers"



Impala vs Spark vs Hive/Tez







OSS – Industry – Research

Best Frenemies Forever

Competition

Research

- Limited scope to demonstrate changes
- Low cost for testing prototypes

Industry

- Access to talent, tools beyond
- Ability to influence ecosystem
- Easy collaboration across companies

OSS

- “Safe harbor”
- IP protections (Apache Software Foundation)

Questions?

Microsoft Cloud and Information Services Lab (CISL)

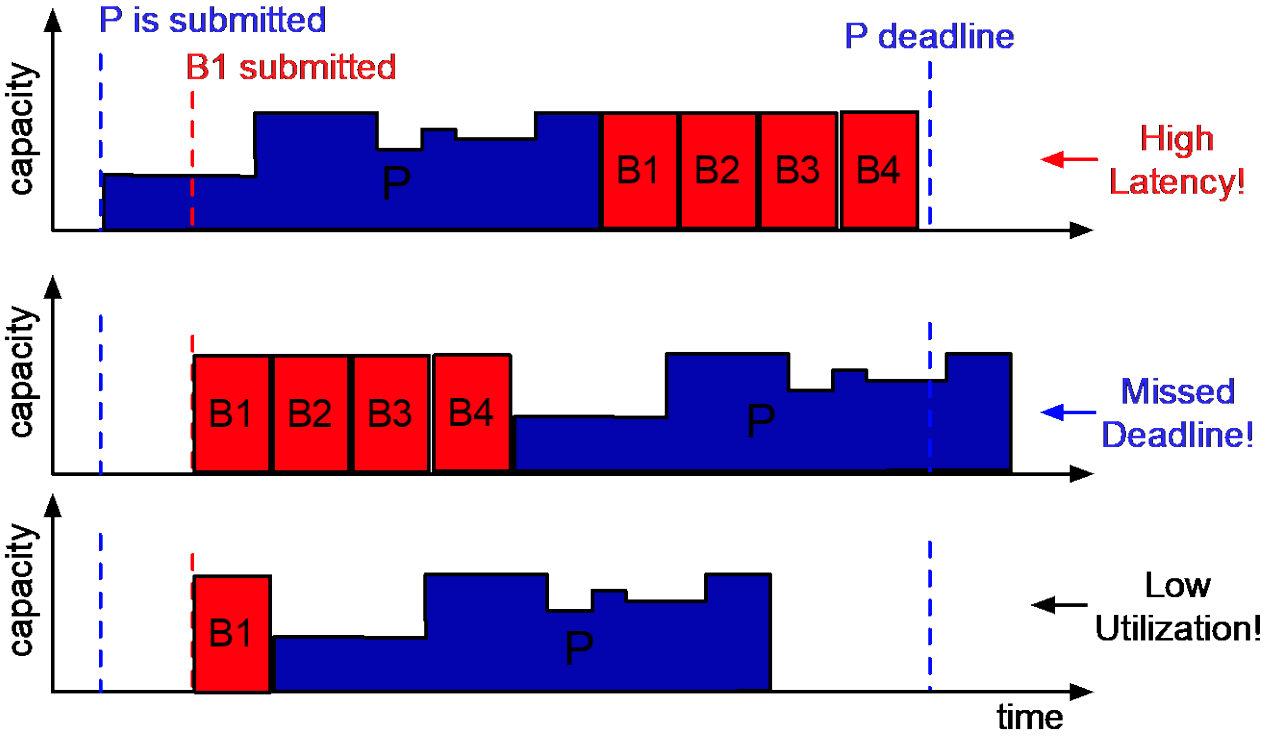
Systems Group

Apache Hadoop YARN: Yet Another Resource Negotiator (SoCC 2013)
Tango: Distributed data structures over a shared log (SOSP 2013)
Reservation-based Scheduling: If You're Late Don't Blame Us! (SoCC 2014)
Multi-resource packing for cluster schedulers (SIGCOMM 2014)
REEF: Retainable Evaluator Execution Framework (SIGMOD 2015)
WANalytics: Analytics for a Geo-Distributed Data-Intensive World (CIDR 2015)
Global analytics in the face of bandwidth and regulatory constraints (NSDI 2015)
Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters (tech report)

Chris Douglas

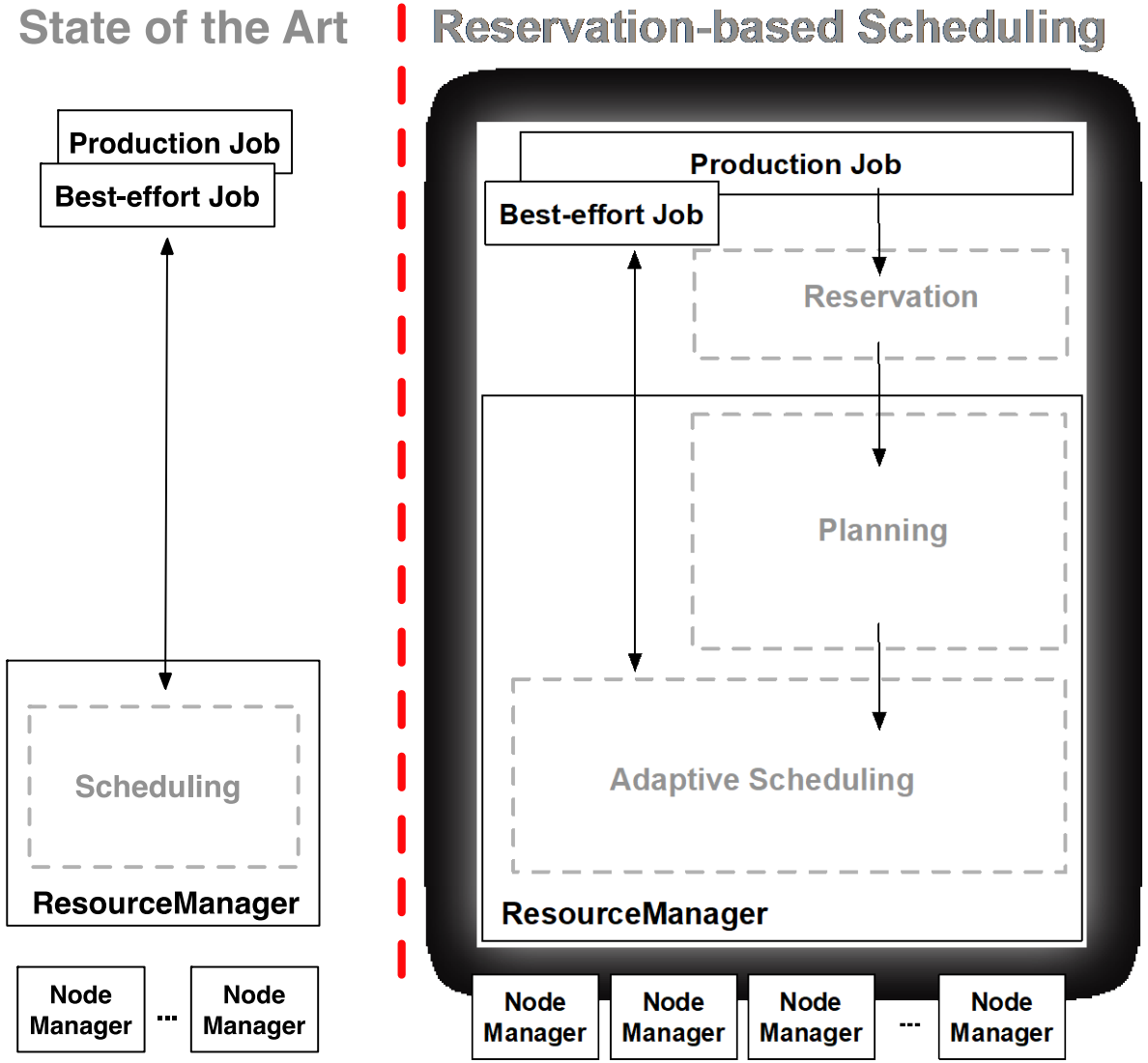
cdoug@microsoft.com, cdouglas@apache.org, @chris_douglas

What are my options...



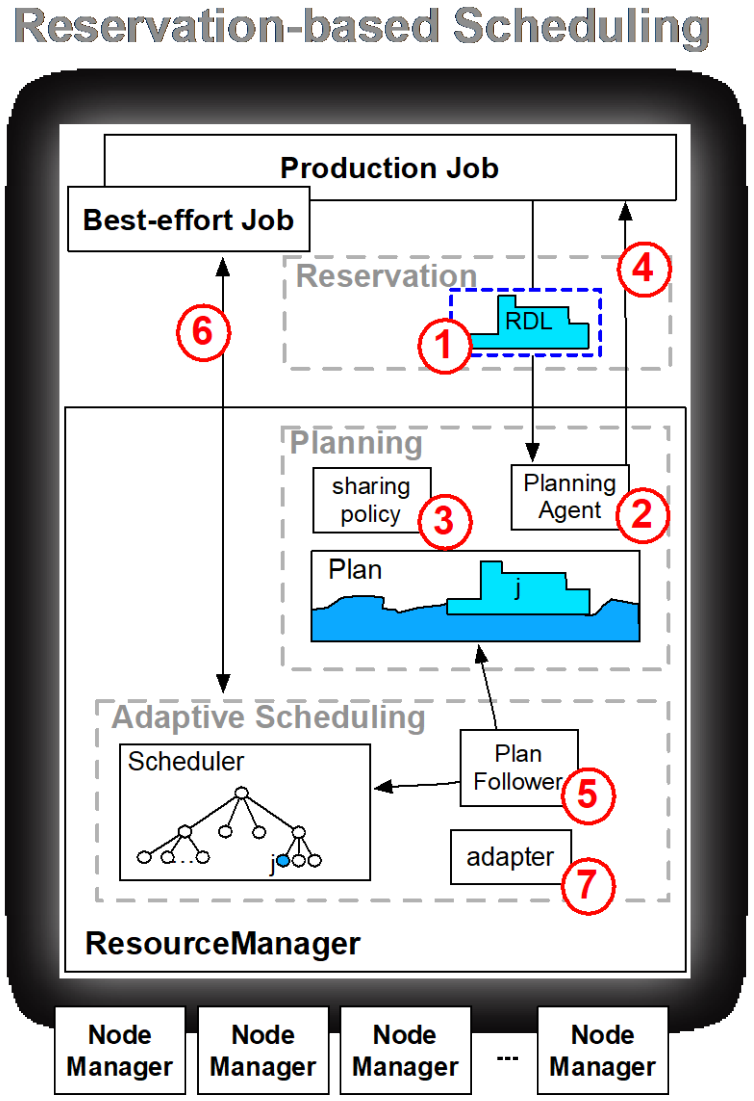
XXXX ffixme

Teach the RM about time



- Ideas:
- 1) Declaratively negotiate use of resources
- 2) Leverage the exposed flexibility to plan
- 3) Dynamically adapt scheduling invariants

Teach the RM about time: Architecture



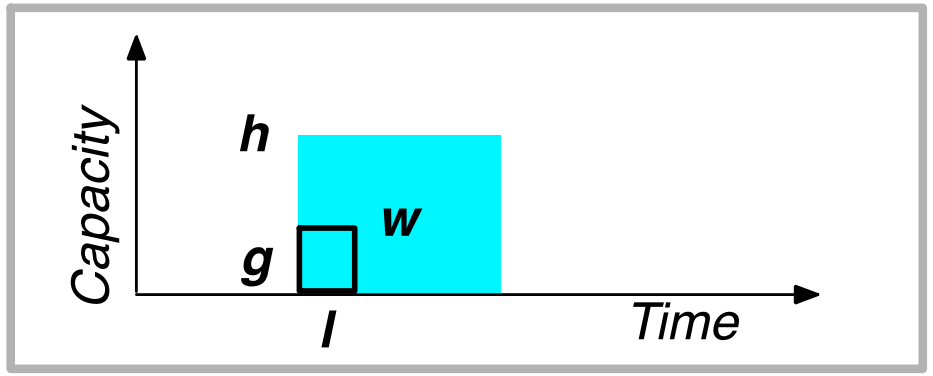
- Steps:
- 1. App formulates reservation request in RDL
- 2. Request is “placed” in the *Plan*
- 3. Allocation is validated against sharing policy
- 4. System commits to deliver resources on time
- 5. Plan is dynamically enacted
- 6. Jobs get (reserved) resources
- 7. System adapts to live conditions
-

(1) Reservation Definition Language (RDL)

a) atomic RDL expression

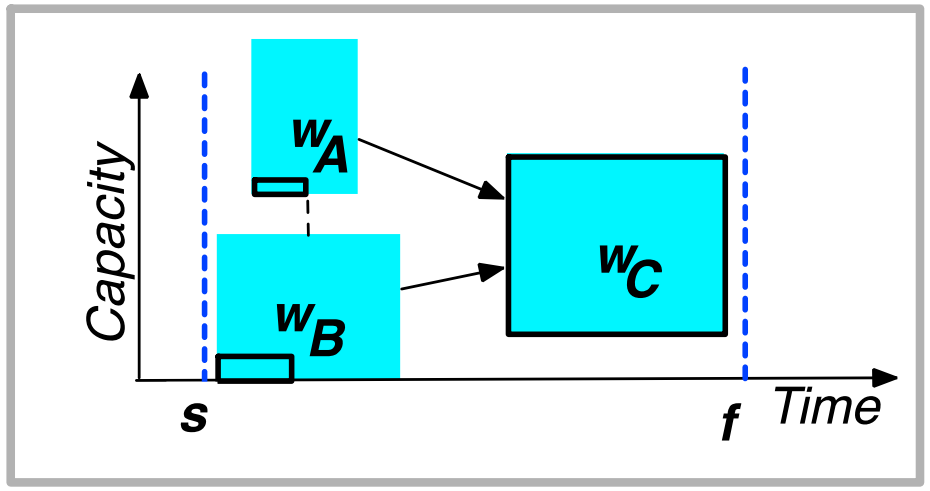
atom(b,g,h,l,w)

e.g., atom (<2GB,1core>, 1, 10, 1min, 10bundle/min)



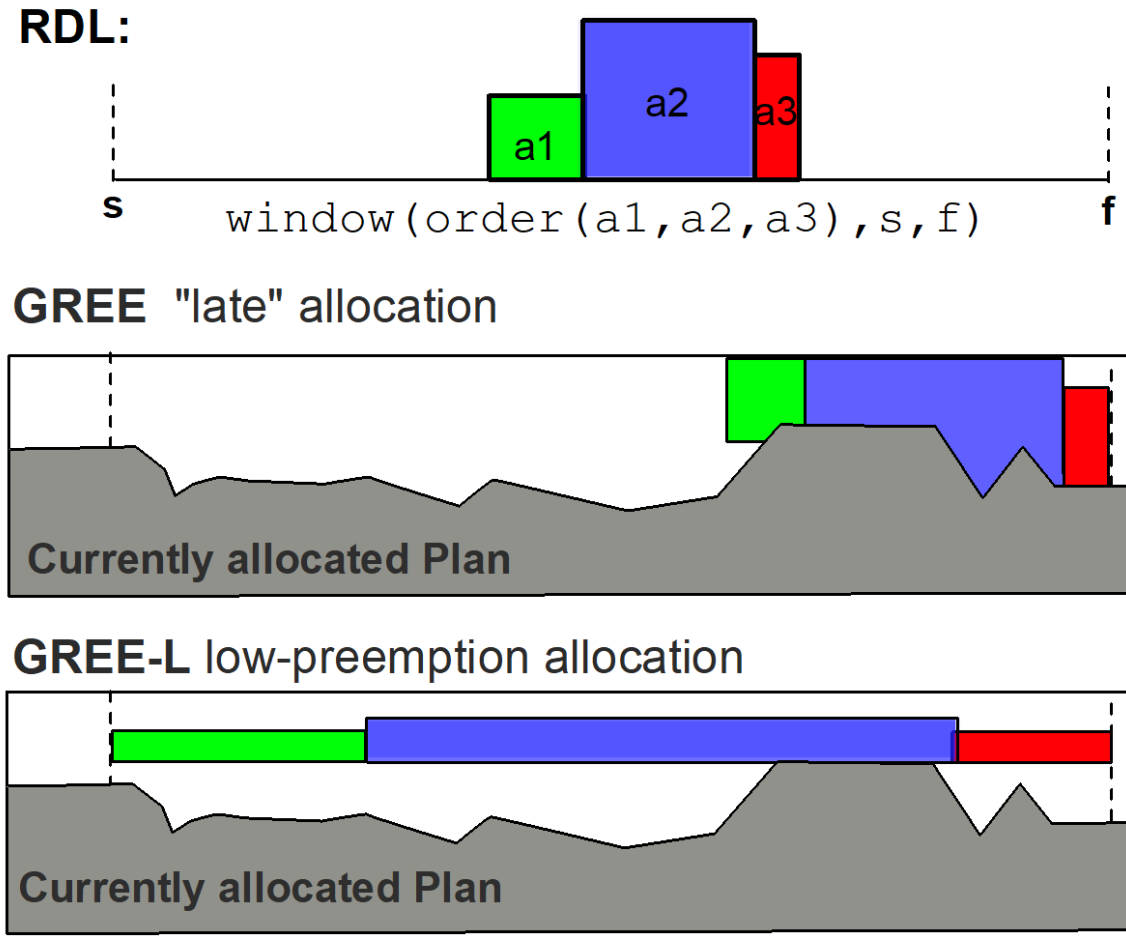
b) composite RDL expression

```
window(
 order(
   all(atom(b,gA,hA,lA,wA),
       atom(b,gB,hB,lB,wB)),
   atom(b,gC,hC,lC,wC)),
 s,f)
```



- *(simplified for OSS release)*

(2) Greedy Agents: intuition



- GREE
  - Allocate “late”, run “early”
- GREE-L
  - Allocate “flat”
  - Minimize preemption

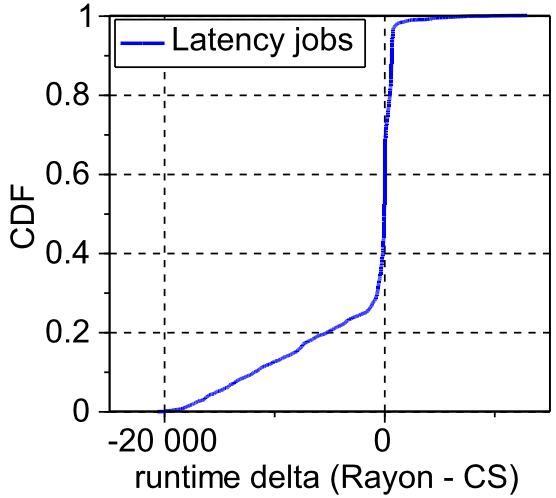
(3) User Quotas (trade-off flexibility to fairness)

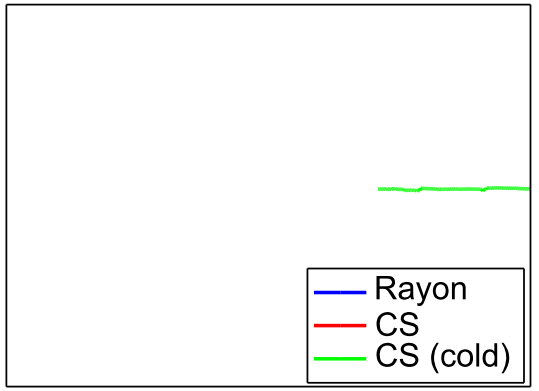
- Validation Policy: *CapacityOverTimePolicy*
- instantaneous max, running avg
- *e.g., user does not exceed instantaneous 30% allocation, and*
- *an average of 10% in any window of 24h*
-
- *O(n)* scanning the inventory min(alloc) – T to max(alloc) + T
-

(7) Adapting to changing conditions

- Coping with Misprediction (user)
  - budget for them in reservation by overallocating
  - continue in best-effort mode
  - re-negotiate to extend reservation
- Coping with Failures (system)
  - leave headroom in plan
  - monitor and re-plan (move jobs)
  - discard reservations (last accepted/least important)

Comparing against Hadoop CapacityScheduler



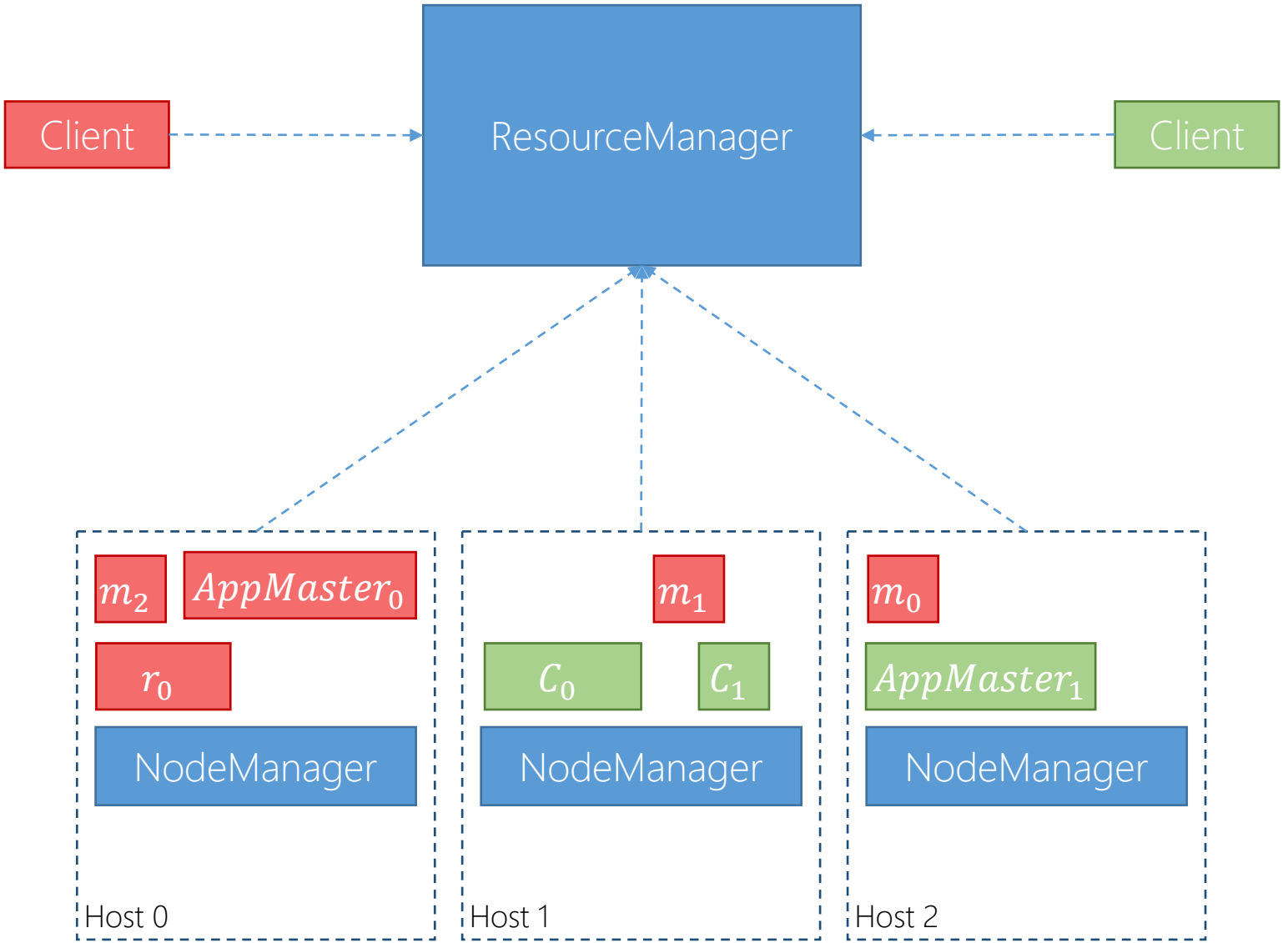


- Results
- meet all production job SLAs
- lowers best-effort jobs latency
- increased cluster utilization and throughput

Rayon Conclusion

- *Idea:* explicit treatment of time in resource management
- *Enables:* completion SLAs, low-latency for best-effort jobs, high ROI
- *End-to-end system effort:* in Apache Hadoop *2.6*

- Future Directions
- Hive/MR modeling (Perforator project)
- Giraph modeling (PreDICT: Adrian Popescu EPFL)
- Advanced agents for placement
- Dynamic pricing

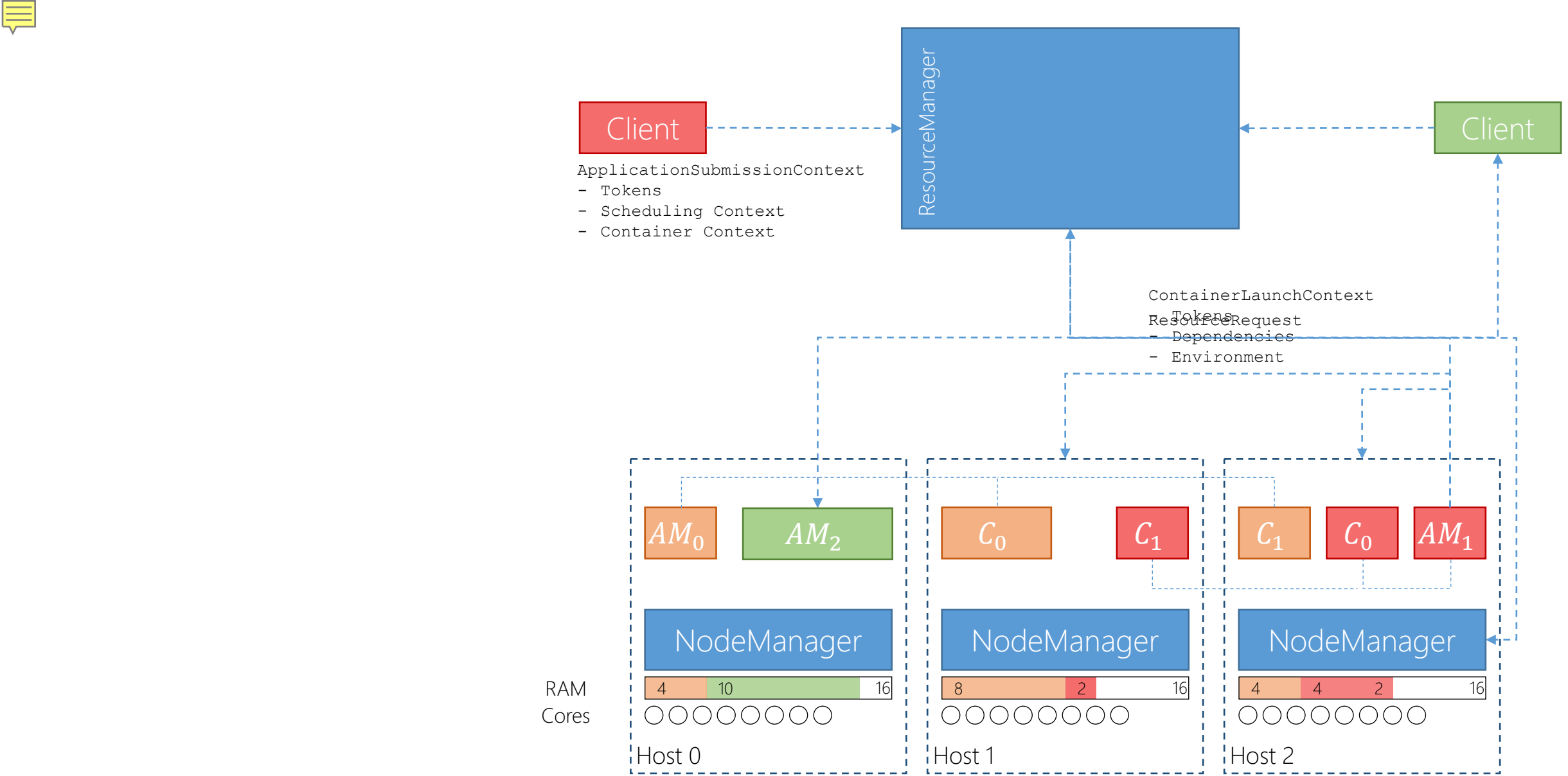

Client
ResourceManager
Client
m_2
$AppMaster_0$
r_0
NodeManager
Host 0
m_1
C_0
C_1
NodeManager
Host 1
m_0
$AppMaster_1$
NodeManager
Host 2

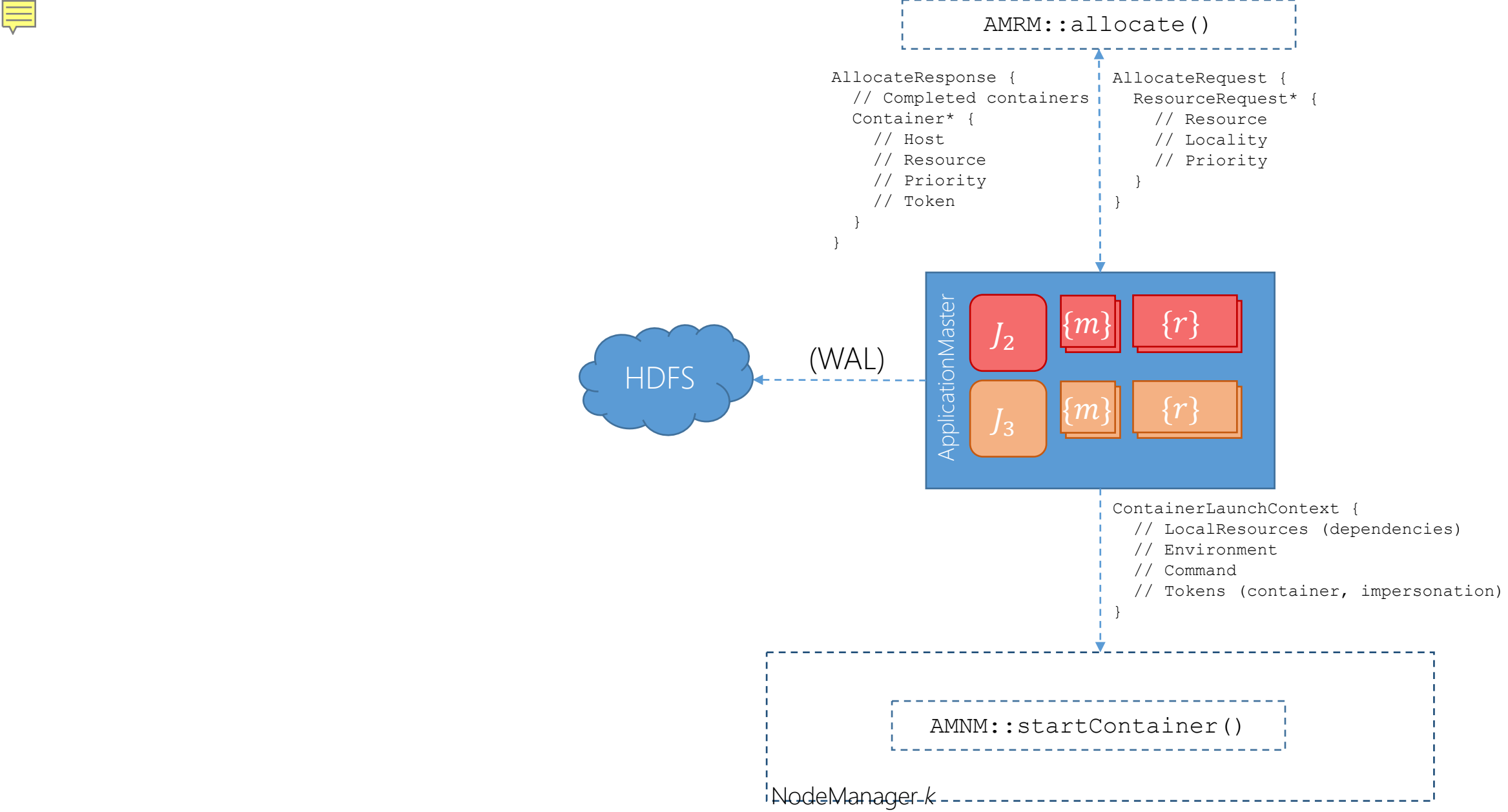
Queries

- Inline containers into the ApplicationMaster

Sessions

- Persistent connection to client
- Elastically allocate resources
- Retain resources across queries


Client
ResourceManager
Client
ApplicationSubmissionContext
- Tokens
- Scheduling Context
- Container Context
ContainerLaunchContext
- Tokens
- ResourceRequest
- Dependencies
- Environment
AM_0
AM_2
C_0
C_1
C_1
C_0
AM_1
NodeManager
NodeManager
NodeManager
RAM
Cores
4
10
16
8
2
16
4
4
2
16
Host 0
Host 1
Host 2


AMRM::allocate()
AllocateResponse {
// Completed containers
Container* {
// Host
// Resource
// Priority
// Token
}
}
AllocateRequest {
ResourceRequest* {
// Resource
// Locality
// Priority
}
}
HDFS
(WAL)
ApplicationMaster
J_2
$\{m\}$
$\{r\}$
J_3
$\{m\}$
$\{r\}$
ContainerLaunchContext {
// LocalResources (dependencies)
// Environment
// Command
// Tokens (container, impersonation)
}
AMNM::startContainer()
NodeManager k