Cloud Computing and Big Data Processing

Kathy Yelick

With slides from Shivaram Venkataraman and Matei Zaharia

and from

Cloud Computing, Big Data

Big data, Hardware, Software, Business

Data, Data, Data

"…Storage space must be used efficiently to store indices and, optionally, the documents themselves. The indexing system must process hundreds of gigabytes of data efficiently…"

The Anatomy of a Large-Scale Hypertextual Web Search Engine

Sergey Brin and Lawrence Page

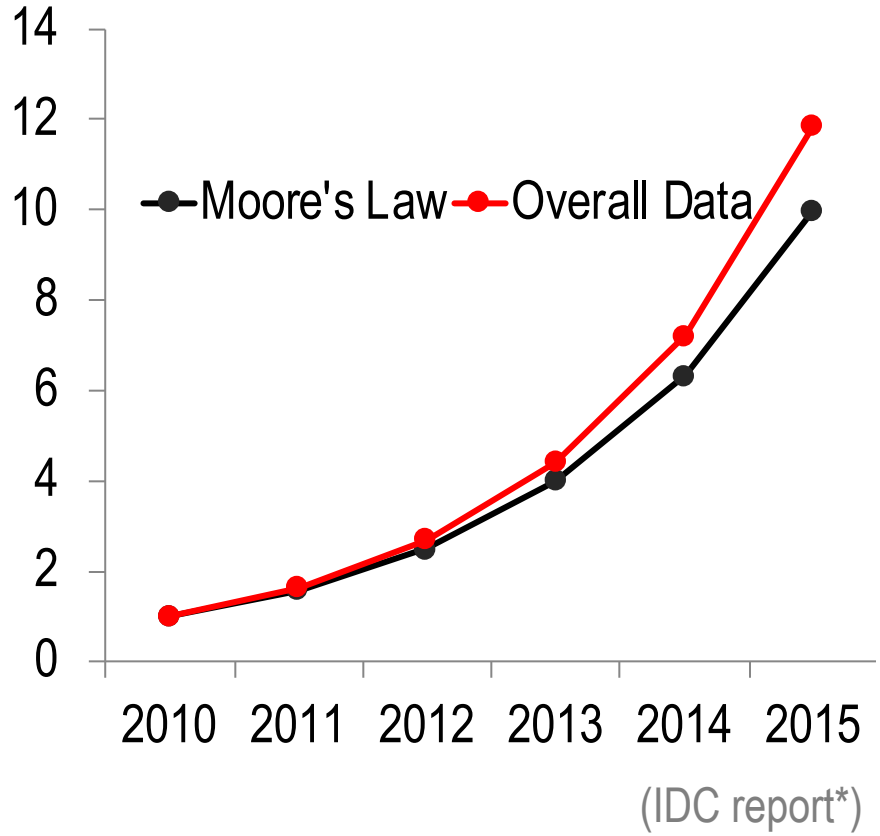
http://infolab.stanford.edu/~backrub/google.html

Datacenter Evolution

Facebook's daily logs:
60 TB

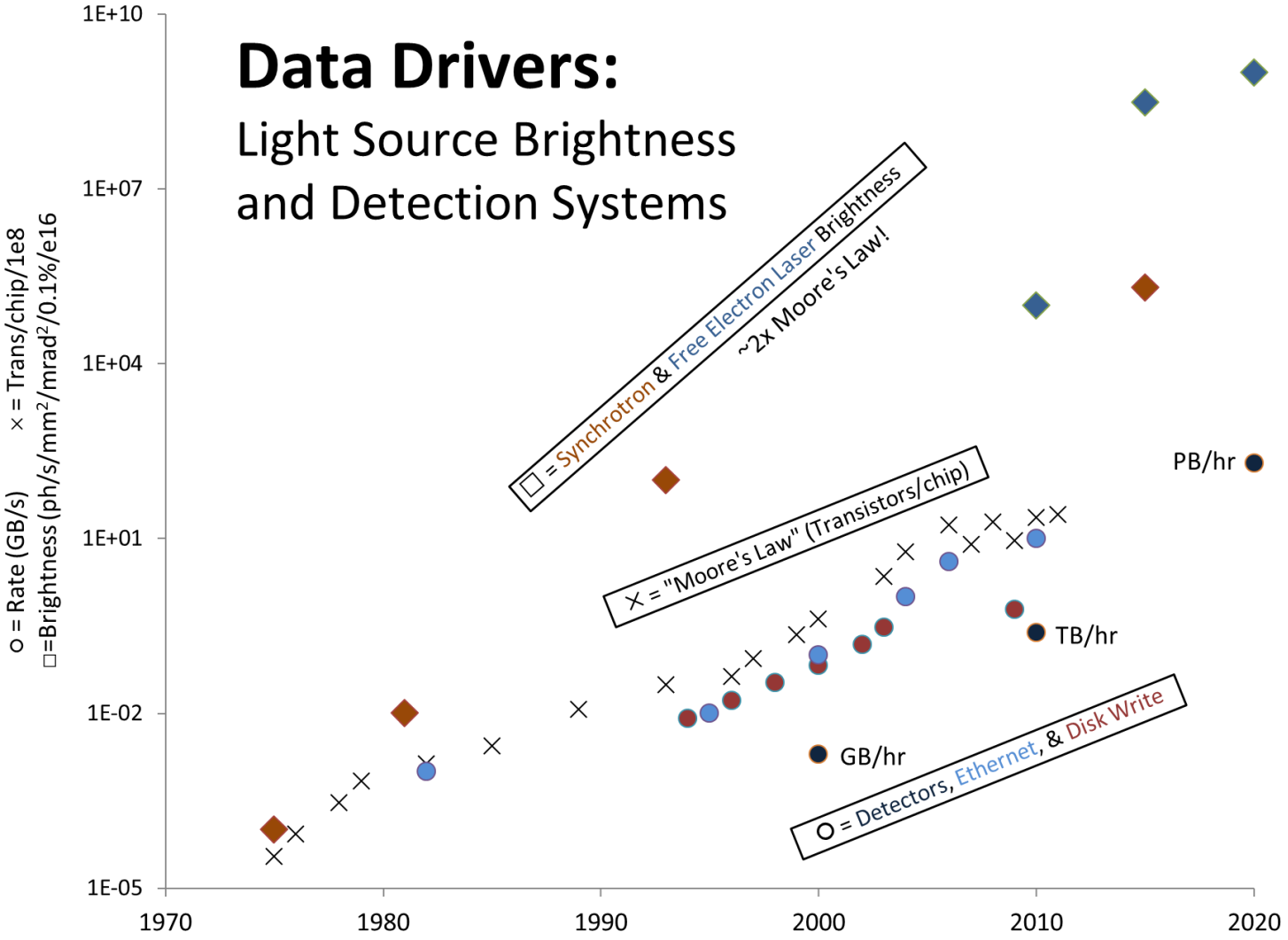
1000 genomes project:
200 TB

Google web index:
10+ PB

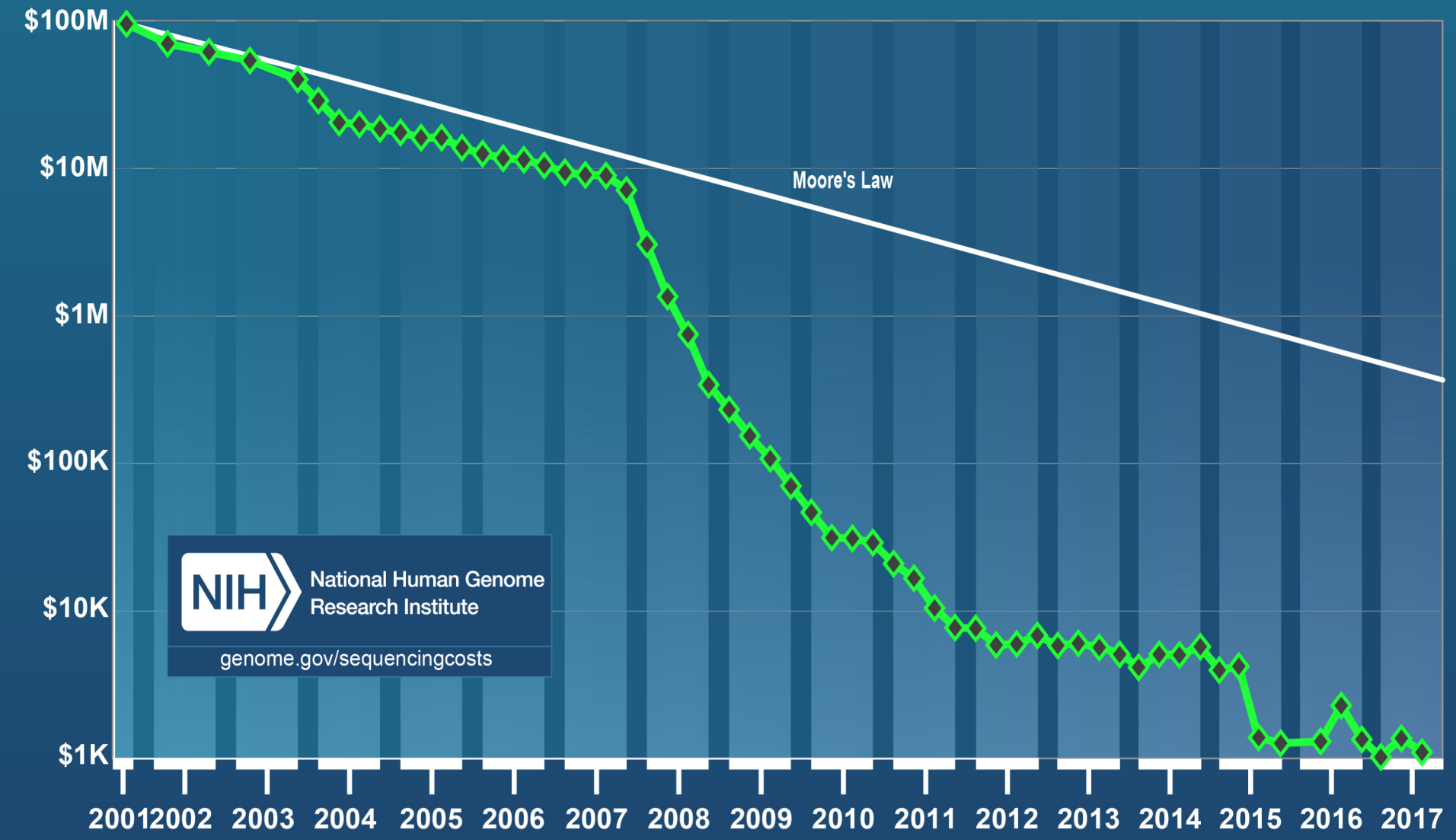


Data from Ion Stoica

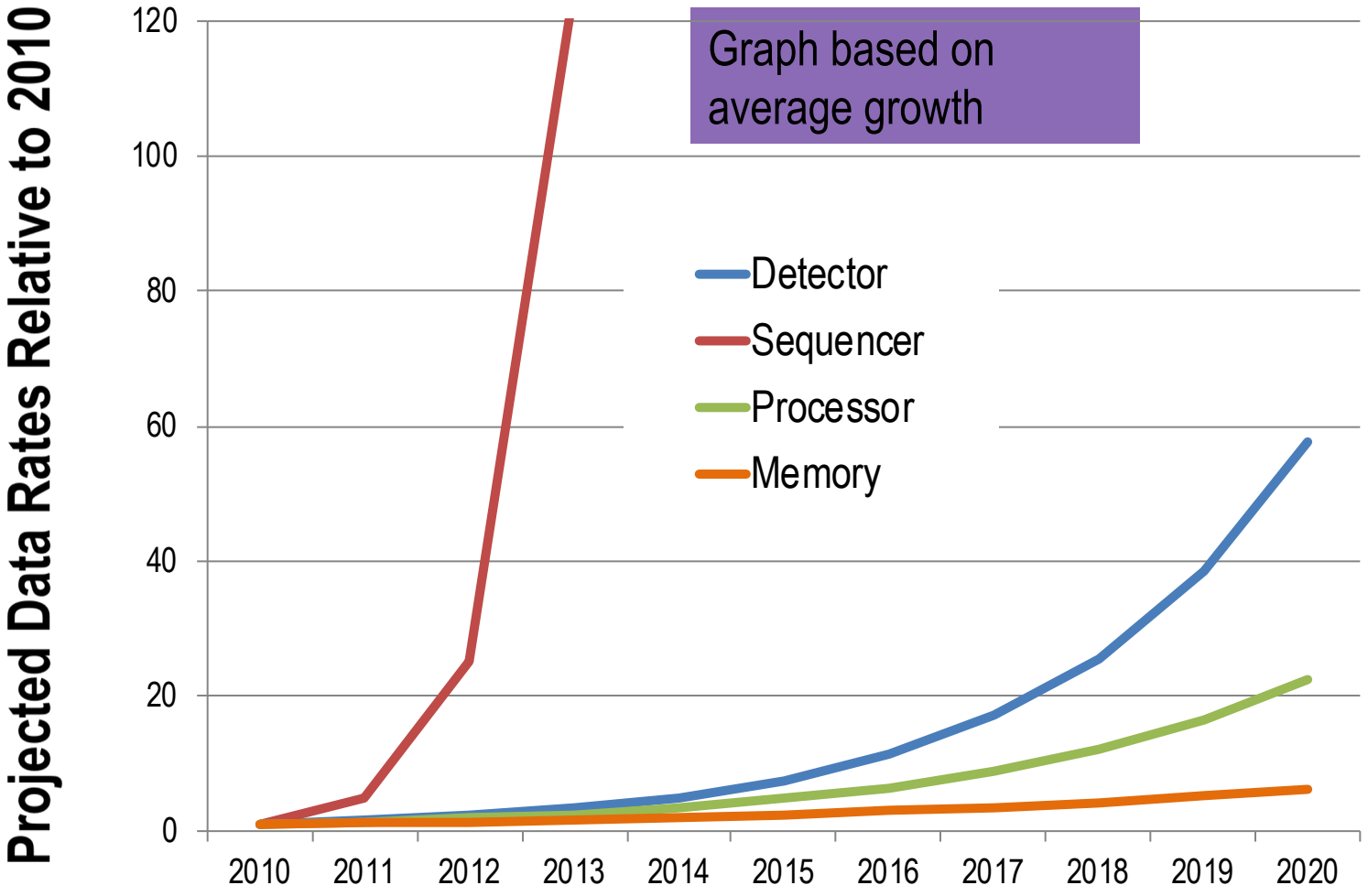
Data in Science: Light Sources data rates – Faster Than Moore's Law



Cost per Genome

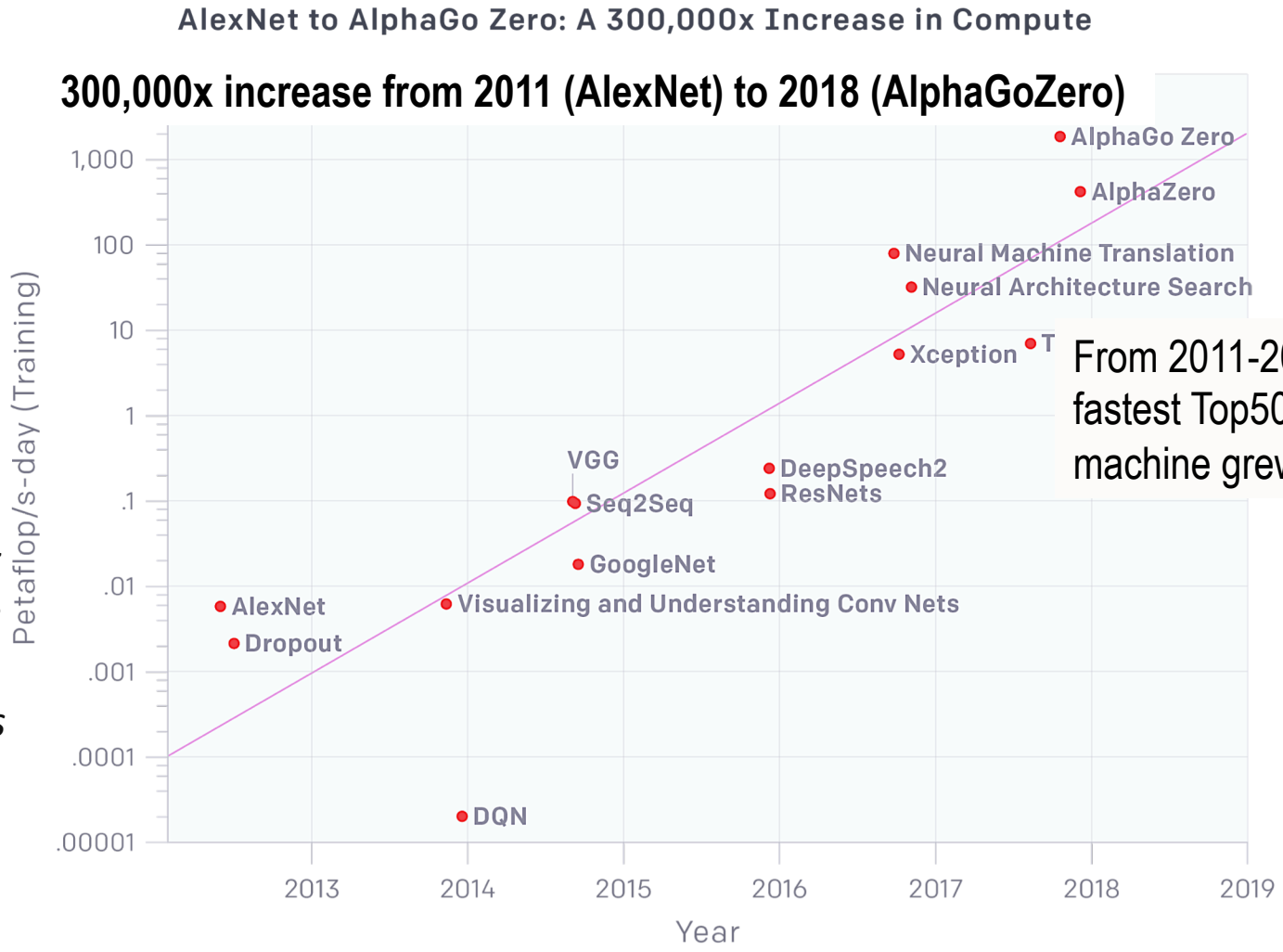


Data Growth is Outpacing Computing Growth



Growth in Computing for Machine Learning

A petaflop/s-day = 10^{15} neural net operations per second for one day, ~= 10^{20} operations



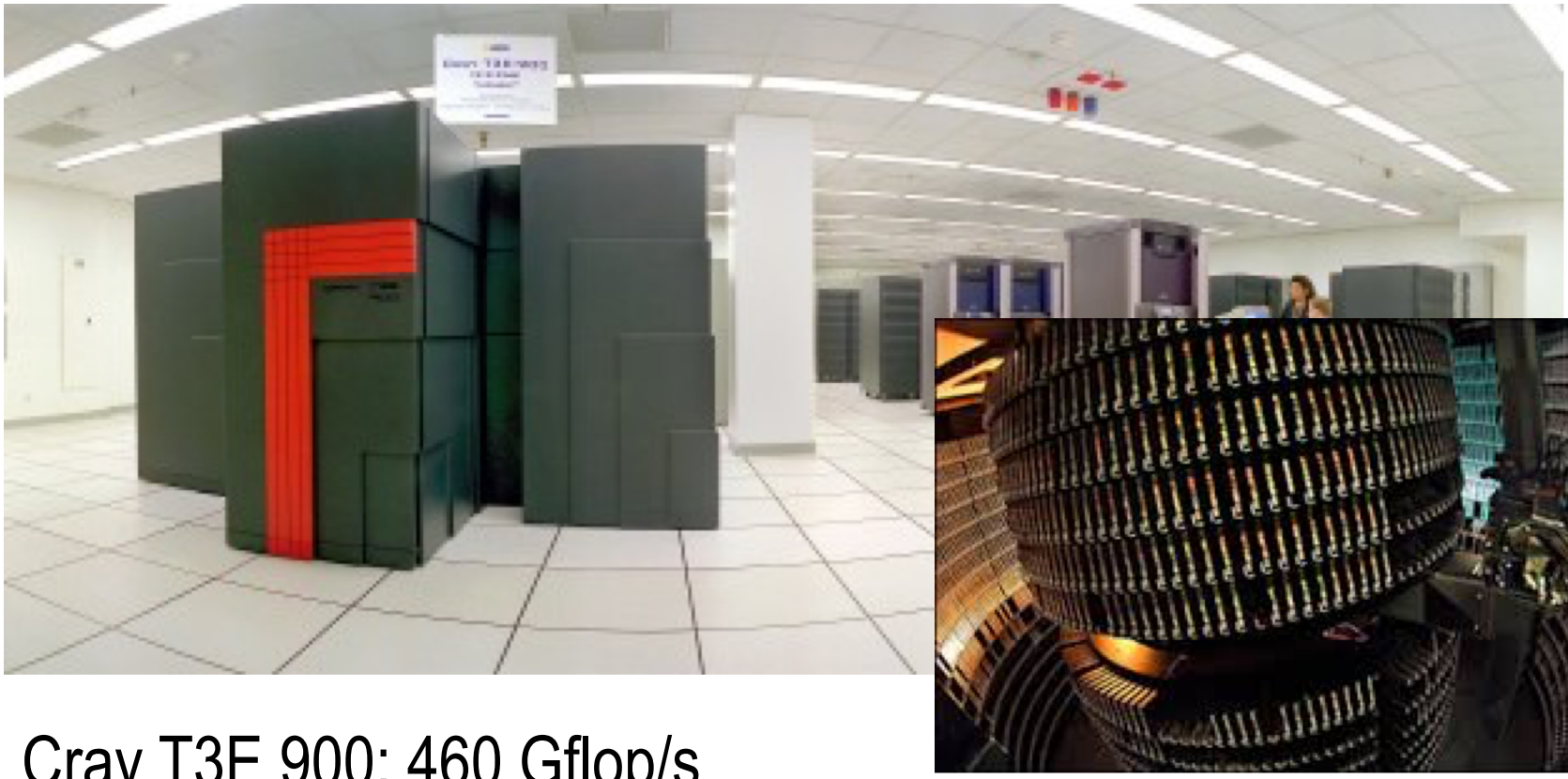
From 2011-2017 the fastest Top500 machine grew < 10x

https://blog.openai.com/ai-and-compute/

Hardware

Google 1997

NERSC 1996

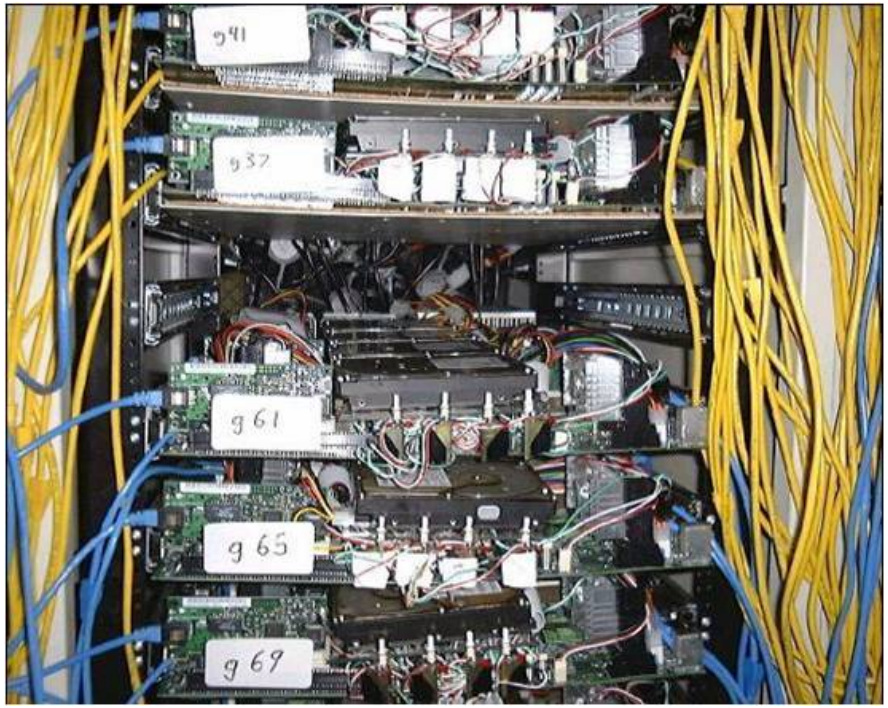


Cray T3E 900: 460 Gflop/s

850 GB of disk

106 TB Data archive

Google 2001



Commodity CPUs

Lots of disks

Low bandwidth network

Cheap !

NERSC 2001: Seaborg

2001 5 Tflop/s 3.3 TB upgraded in 2002 to 10 Tflop/s, 6.6 TB

Supercomputer speeds have increased by 1000 X almost every decade

CDC 6600 in 1964,
Megaflops

Cray 2 in 1985
Gigaflops

ASCI Red (Intel)in 1998
Teraflops

IBM Roadrunner in 2008
Petaflops

Datacenter Evolution

Google data centers in The Dalles, Oregon

- 200K SF + 164K sf in (3 buildings total)
- $1.2 B investment in site
- 175 people employed on site
- 70 Megawatts (when it was 200K SF)

Google in the Netherlands: 30MW Solar Farm

Supercomputing Center Evolution

NERSC Center in Berkeley

- 27K sf of compute center space
- ~250 people in building
- 12.5 MW today
- Upgrading to 25MW

Datacenter Design

Goals

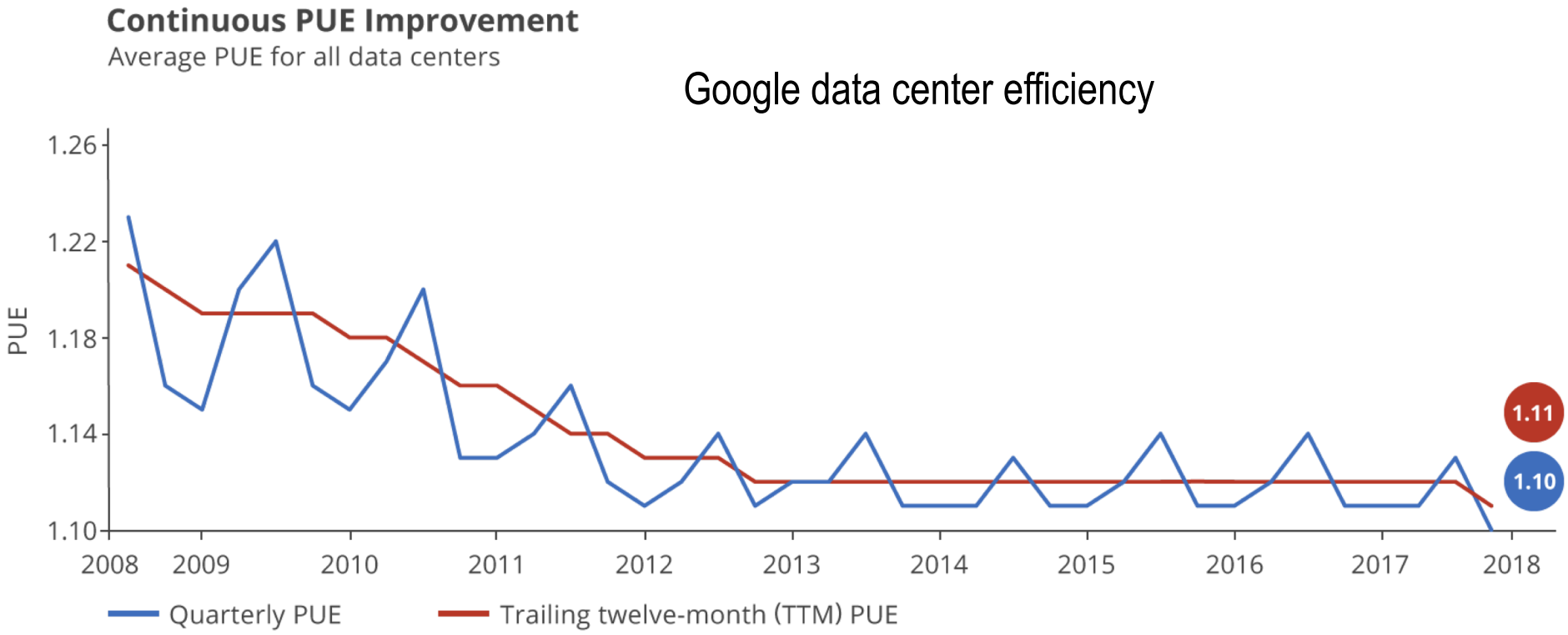
Power usage effectiveness (PUE)

Cost-efficiency

Custom machine design

Open Compute Project
(Facebook)

Energy Efficiency



Google data center efficiency

Data center average about 1.7, NERSC also under 1.1

https://www.google.com/about/datacenters/efficiency/internal/

Datacenter Evolution

Capacity:

~10000 machines

Bandwidth:

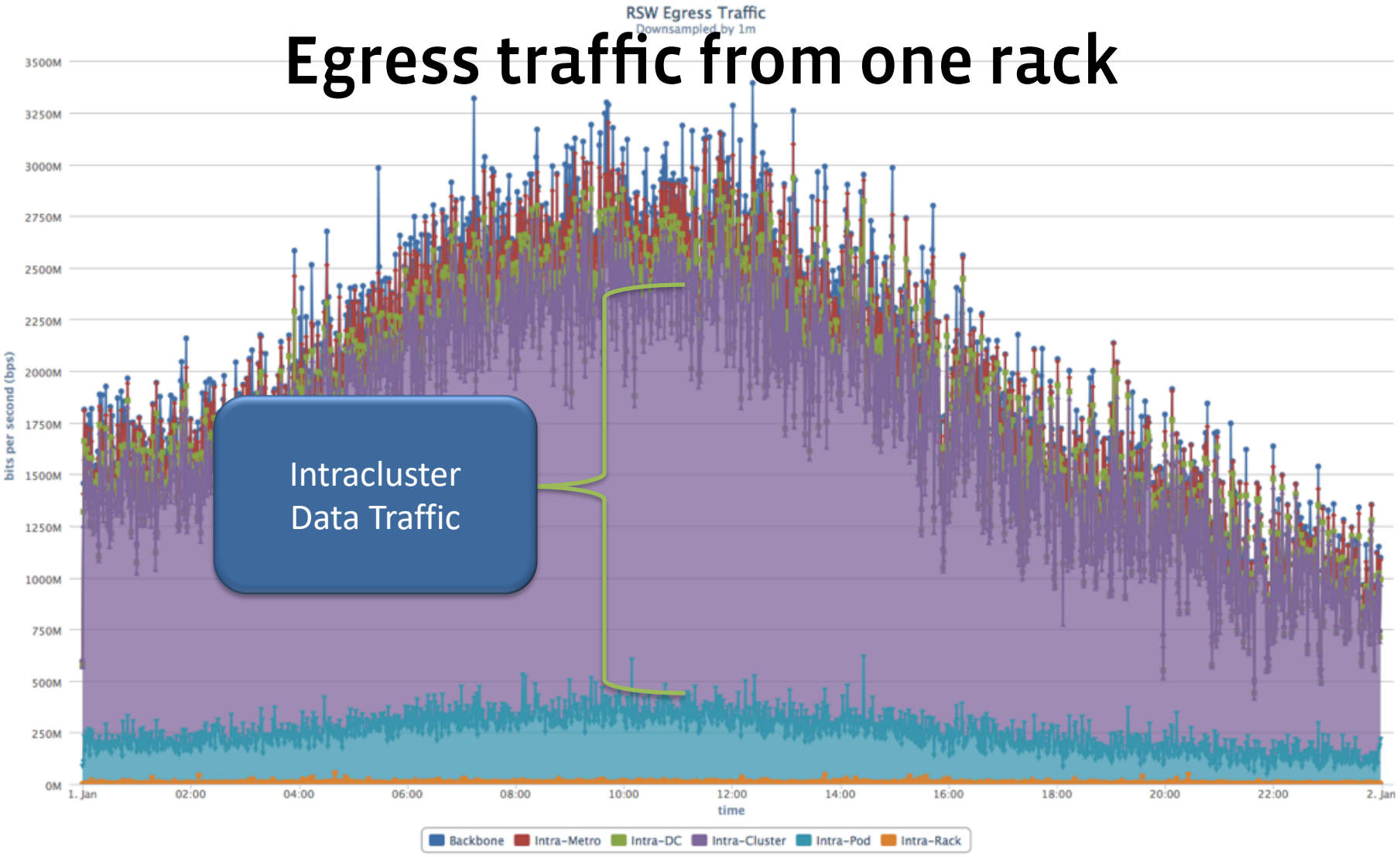
12-24 disks per node

Latency:

256GB RAM cache

Majority of Facebook Traffic is Intra-Cluster

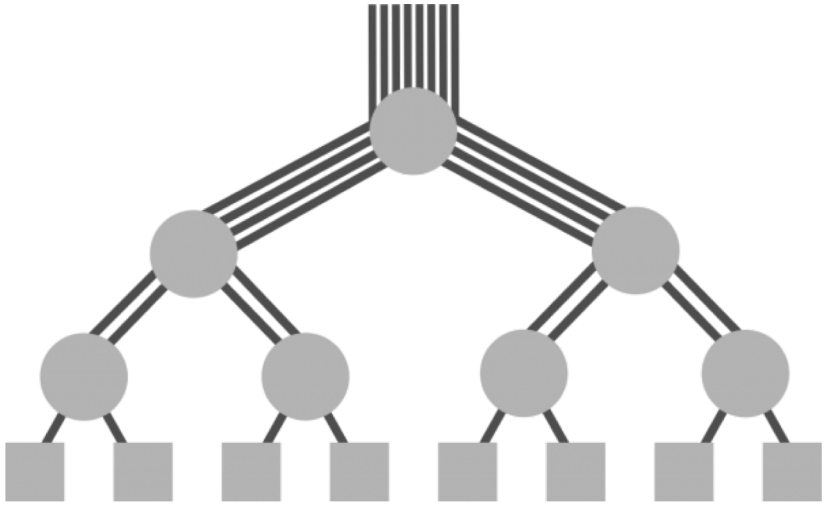
(Nathan Farrington, Facebook Inc., Presented at OIC2013)



Datacenter Networking

Initially tree topology

Over subscribed links



Fat tree, Bcube, VL2 etc.

Lots of research to get full bisection bandwidth

Datacenters → Cloud Computing

Above the Clouds: A Berkeley View of Cloud Computing

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz,
Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia
(Comments should be addressed to abovetheclouds@cs.berkeley.edu)

UC Berkeley Reliable Adaptive Distributed Systems Laboratory *
http://radlab.cs.berkeley.edu/

"…long-held dream of computing as a utility…"

From Mid 2006

Rent virtual computers in the “Cloud”

On-demand machines, spot pricing

Demand and Spot Pricing

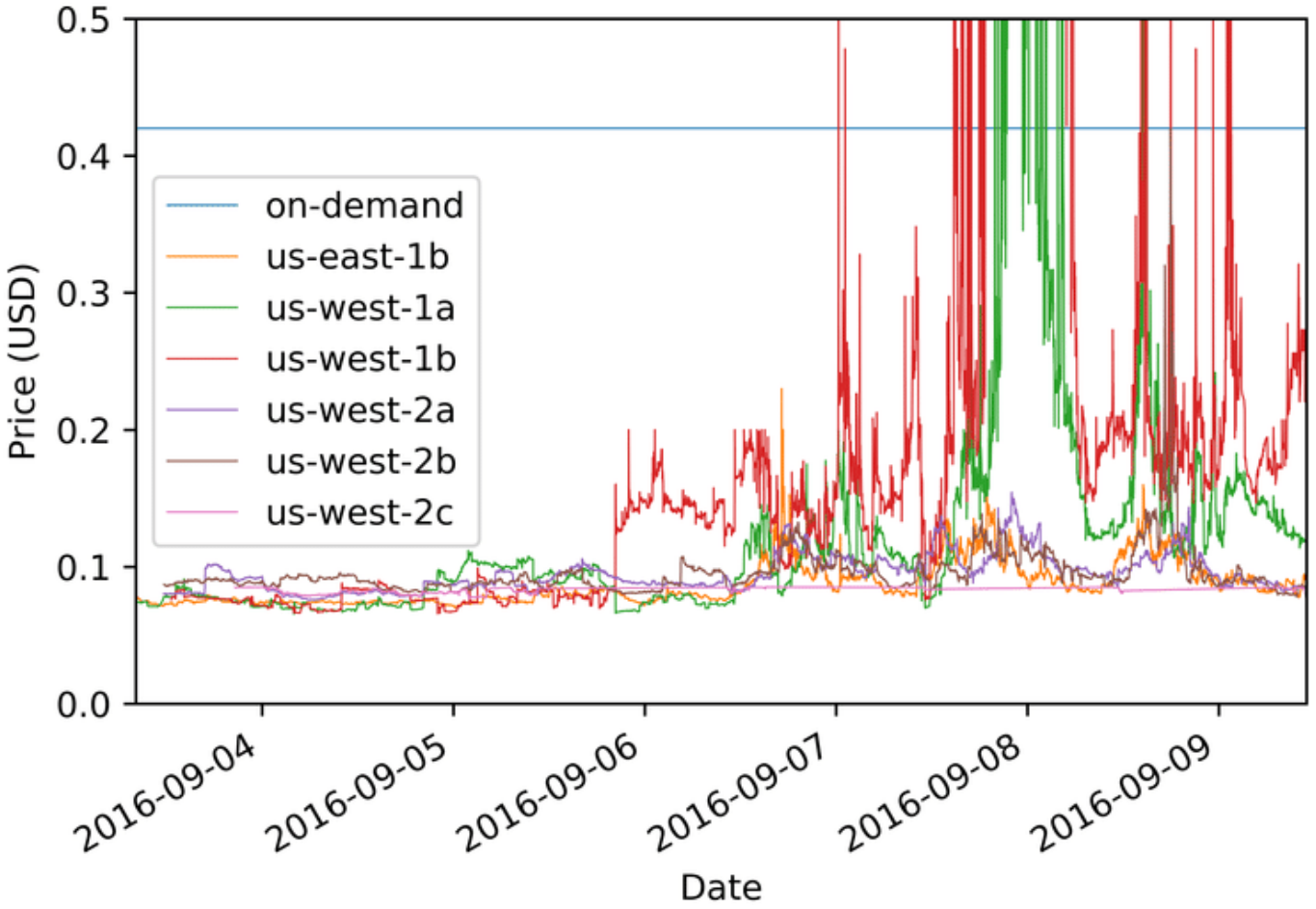


Image from: Baughman, Matthew & Haas, Christian & Wolski, Rich & Foster, Ian & Chard, Kyle. (2018). Predicting Amazon Spot Prices with LSTM Networks. 1-7. 10.1145/3217880.3217881.

Amazon EC2 (2014)

Machine	Memory (GB)	Compute Units (ECU)	Local Storage (GB)	Cost / hour
t1.micro	0.615	1	0	$0.02
m1.xlarge	15	8	1680	$0.48
cc2.8xlarge	60.5	88 (Xeon 2670)	3360	$2.40

1 ECU = CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor

Amazon EC2 (2015)

Machine	Memory (GB)	Compute Units (ECU)	Local Storage (GB)	Cost / hour
t2.micro	~~0.615~~ 1	1	0	$0.013
r3.xlarge	~~15~~ 30	~~8~~ 13	~~1680~~ 80(SSD)	$0.35
r3.8xlarge	~~60.5~~ 244	~~88~~ 104 (32 Ivy Bridge)	~~3360~~ 640(SSD)	$2.80

1 ECU = CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor

Amazon EC2 (2016)

Machine	Memory (GB)	Compute Units (ECU)	Local Storage (GB)	Cost / hour
t2.nano	0.5	1	0	$0.006
t2.micro	1	1	0	$0.013
r3.8xlarge	~~60.5~~ 244	~~88~~ 104 (32 Ivy Bridge)	~~3360~~ 640(SSD)	$2.80
x1 (TBA)	2 TB	4 * Xeon E7	?	?

Amazon EC2 (2017)

Machine	Memory (GB)	Compute Units (ECU)	Local Storage (GB)	Cost / hour
t2.nano	0.5	1	0	$0.006
r3.8xlarge	244	104 (32 Ivy Bridge)	640 (SSD)	~~$2.80~~ $2.66
x1.32xlarge	2 TB	4 * Xeon E7	3.8 TB (SSD)	$13.338
p2.16xlarge	732 GB	16 Nvidia K80 GPUs	0	$14.40

Amazon EC2 (2018)

Machine	Memory (GB)	Compute Units (ECU)	Local Storage (GB)	Cost / hour
t2.nano	0.5	1	0	$0.006
r3.8xlarge	244	104 (32 Ivy Bridge)	640 (SSD)	~~$2.66~~ $2.37
x1.32xlarge	2 TB	4 * Xeon E7	3.8 TB (SSD)	$13.338
p2.16xlarge	732 GB	16 Nvidia K80 GPUs	0	$14.40

Hardware

Hopper vs. Datacenter

	Hopper	Datacenter[2]
Nodes	6384	1000s to 10000s
CPUs (per node)	2x12 cores	~2x6 cores
Memory (per node)	32-64GB	~48-128GB
Storage (overall)	~4 PB	120-480 PB
Interconnect	~ 66.4 Gbps	~10Gbps

[2]http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/

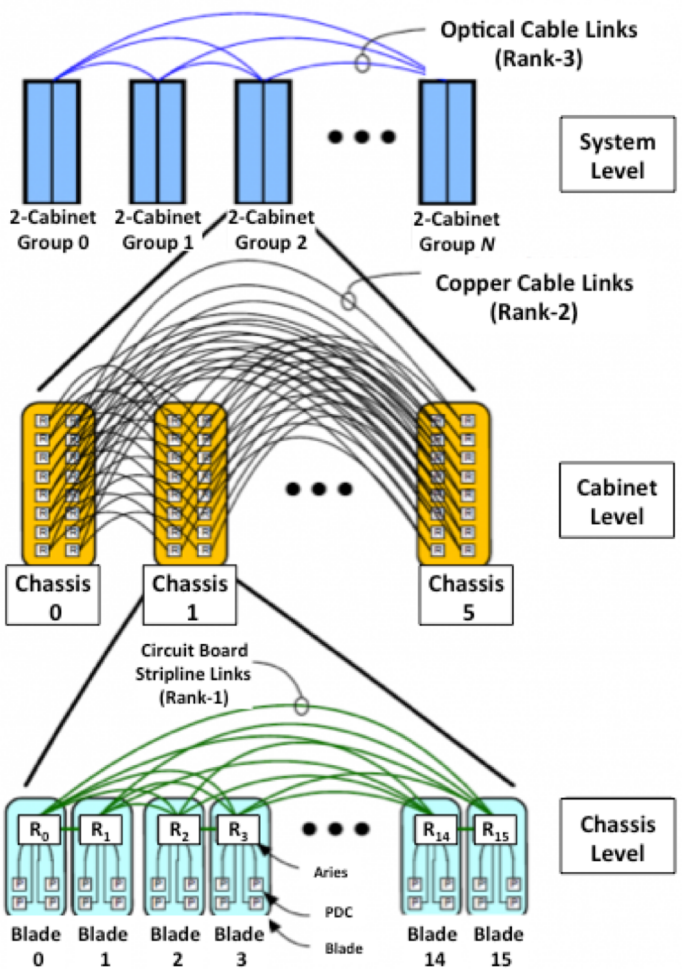
~~Hopper~~ Cori Phase 1 vs. Datacenter

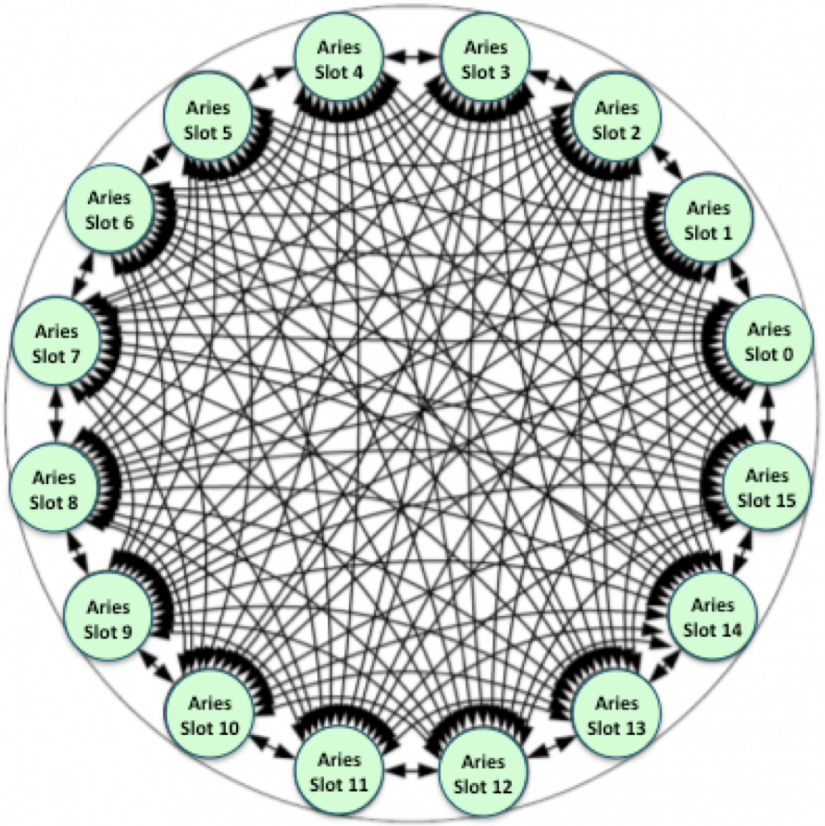
	~~Hopper~~ Cori Phase 1	Datacenter[2]
Nodes	~~6384~~ 2399	1000s to 10000s
CPUs (per node)	~~2x12~~ 2x16 cores 1.2 Tf/s	~2x6 cores
Memory (per node)	~~32-64GB~~ 128 GB	~48-128GB
Storage (overall)	~~~4 PB~~ ~30PB	120-480 PB
Interconnect	~ 66.4 Gbps	~10Gbps

[2]http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/

Cori's Cray Aries Interconnect

45.0 TB/s of Global Bandwidth (all to all)





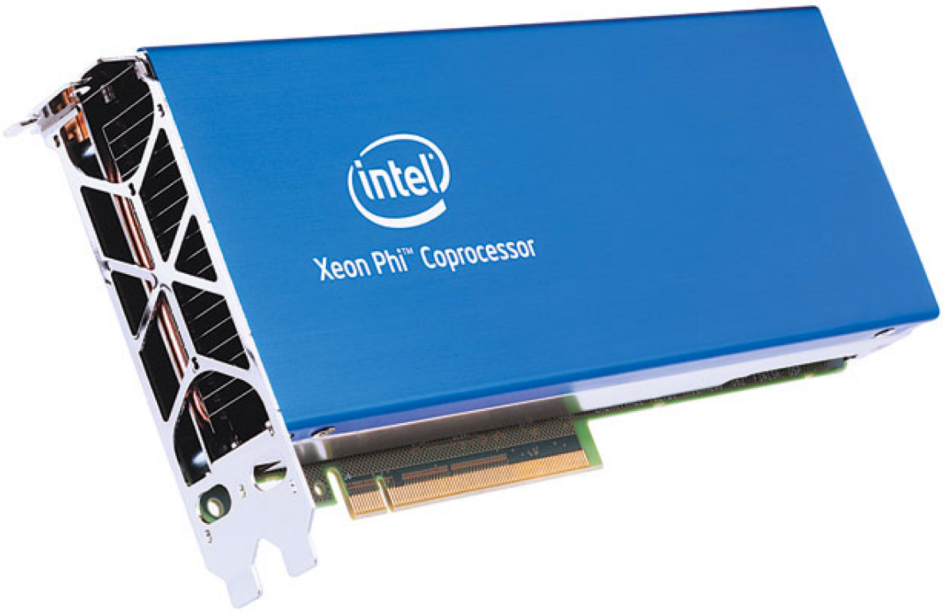
Intra-chassis network connects 16 network interfaces (Aries ASICs) to every other. This is done through the backplane and requires no cables.

Cori: Intel Xeon Phi

Many Core
Integrated Architecture

60 cores per node
(68 in Cori)

Vector Processing



~~Hopper~~ Cori Phase 2 vs. Datacenter

	~~Hopper~~ Cori Phase 1	Datacenter[2]
Nodes	~~6384 2388~~ 9688	1000s to 10000s
CPUs (per node)	~~2x12~~ 1x68 cores ~~1.2 Tf/s.~~ 3 Tf/s	~2x6 cores
Memory (per node)	~~32-64~~128 96 GB	~48-128GB
Storage (overall)	~~~4 PB~~ ~30PB +1.8 PB SSD	120-480 PB
Interconnect	~ 66.4 Gbps	~10Gbps

[2]http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/

OAK RIDGE National Laboratory

Summit (#1 machine) System Overview

summit

System Performance

- Peak performance of 200 petaflops for modeling & simulation
- Peak of 3.3 ExaOps for data analytics and artificial intelligence

Each node has

- 2 IBM POWER9 processors
- 6 NVIDIA Tesla V100 GPUs
- 608 GB of fast memory
- 1.6 TB of NVMe memory

The system includes

- 4608 nodes
- Dual-rail Mellanox EDR InfiniBand network
- 250 PB IBM Spectrum Scale file system transferring data at 2.5 TB/s

Summary of Traditional Differences

(both are changing)

Cloud	HPC
Focus on storage	Focus on computing (flop/s)
Cheap(est) commodity component	High end components (some specialization)
Commodity networks	High performance networks
Pay as you go	Purchased for mission; pay in non-fungible “hours”
< 50% utilization	> 90% utilization
On-demand access	Large jobs wait in queues
On-node disks (air cooled)	Separate storage (compute is liquid cooled)

Outage in Dublin Knocks Amazon, Microsoft Data Centers Offline

By: Rich Miller

August 7th, 2011

557

Like

Dallas-Fort Worth Data Center Update

78

Filed in
on July 9th, 200

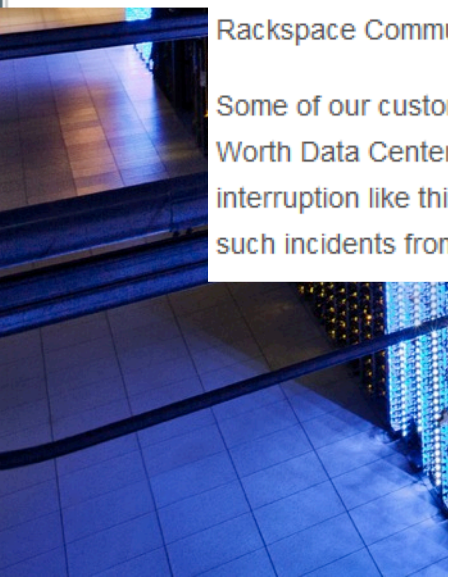
Tweet 0

A lightning s
for **Amazon**
many sites
Microsoft's

Message from R
July 9, 2009

Rackspace Commu

Some of our custo
Worth Data Cente
interruption like thi
such incidents fro



Official Gmail Blog

News, tips and tricks from Google's Gmail team and friends.

Sign U

Mor

Entire Site

Posted:

Posted

Amazon EC2 and Amazon RDS Service Disruption

Gmail's
people
problem

and we'nctionality to all affected services, we would like to share more details with our customers about the events
a list of our efforts to restore the services, and what we are doing to prevent this sort of issue from happening agai

The Joys of Real Hardware

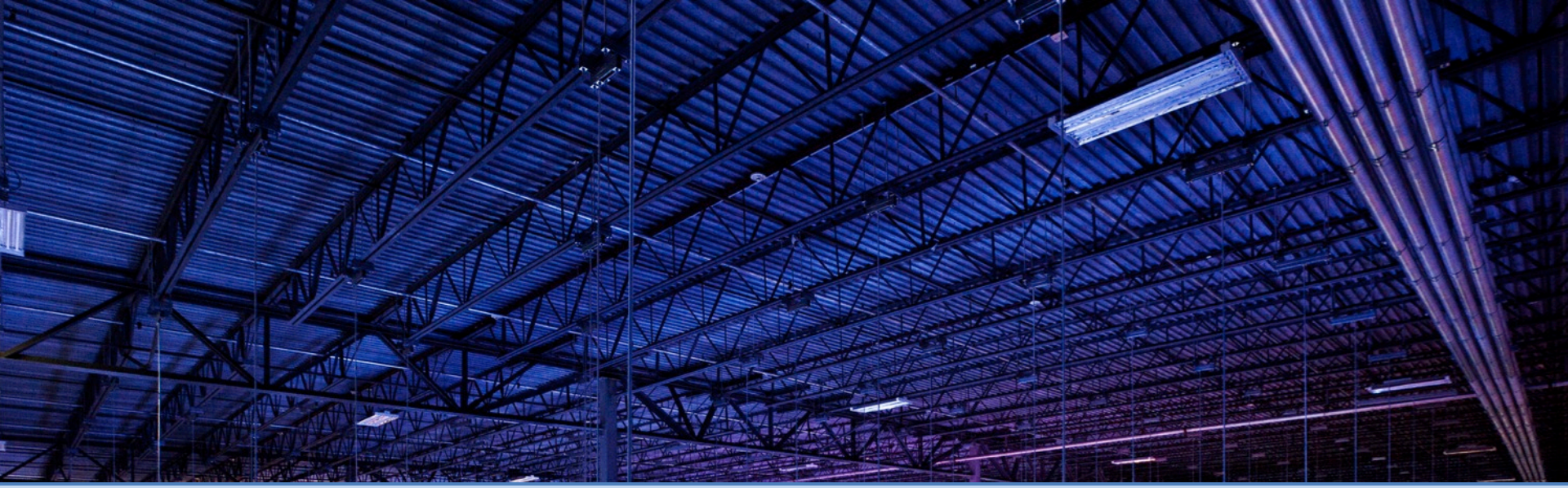
Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

Jeff Dean @ Google

- https://youtu.be/4WlOnlRncK0



How do we program this ?

Software

Open MPI

MPICH

MVAPICH

HPC: Programming Models

Message Passing Models (MPI)

Fine-grained messages + computation

Hard to deal with disk locality, failures, stragglers

1 server fails every 3 years →

10K nodes see 10 faults/day

Exascale research: Fault Tolerant MPI (FTMPI)

Checkpointing-based techniques

Data Programming Models

"Data" Parallel Models (loosely coupled)

Restrict the programming interface

Automatically handle failures, locality etc.

"Here's an operation, run it on all of the data"

- I don't care *where* it runs (you schedule that)
- In fact, feel free to run it *retry* on different nodes

MapReduce

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

ce is a programming model and an associ-
entation for processing and generating large
ers specify a *map* function that processes a
ir to generate a set of intermediate key/value
educe function that merges all intermediate
ated with the same intermediate key. Many
sks are expressible in this model, as shown

given day, etc. Most such computations a
ally straightforward. However, the input da
large and the computations have to be distri
hundreds or thousands of machines in orde
a reasonable amount of time. The issues of
allelize the computation, distribute the data
failures conspire to obscure the original sir
tation with large amounts of complex code
these issues.

Google 2004

Build search index
Compute PageRank

Hadoop: Open-source
at Yahoo, Facebook

MapReduce Programming Model

Data type: Each record is (key, value)

Map function:

$$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$$

Reduce function:

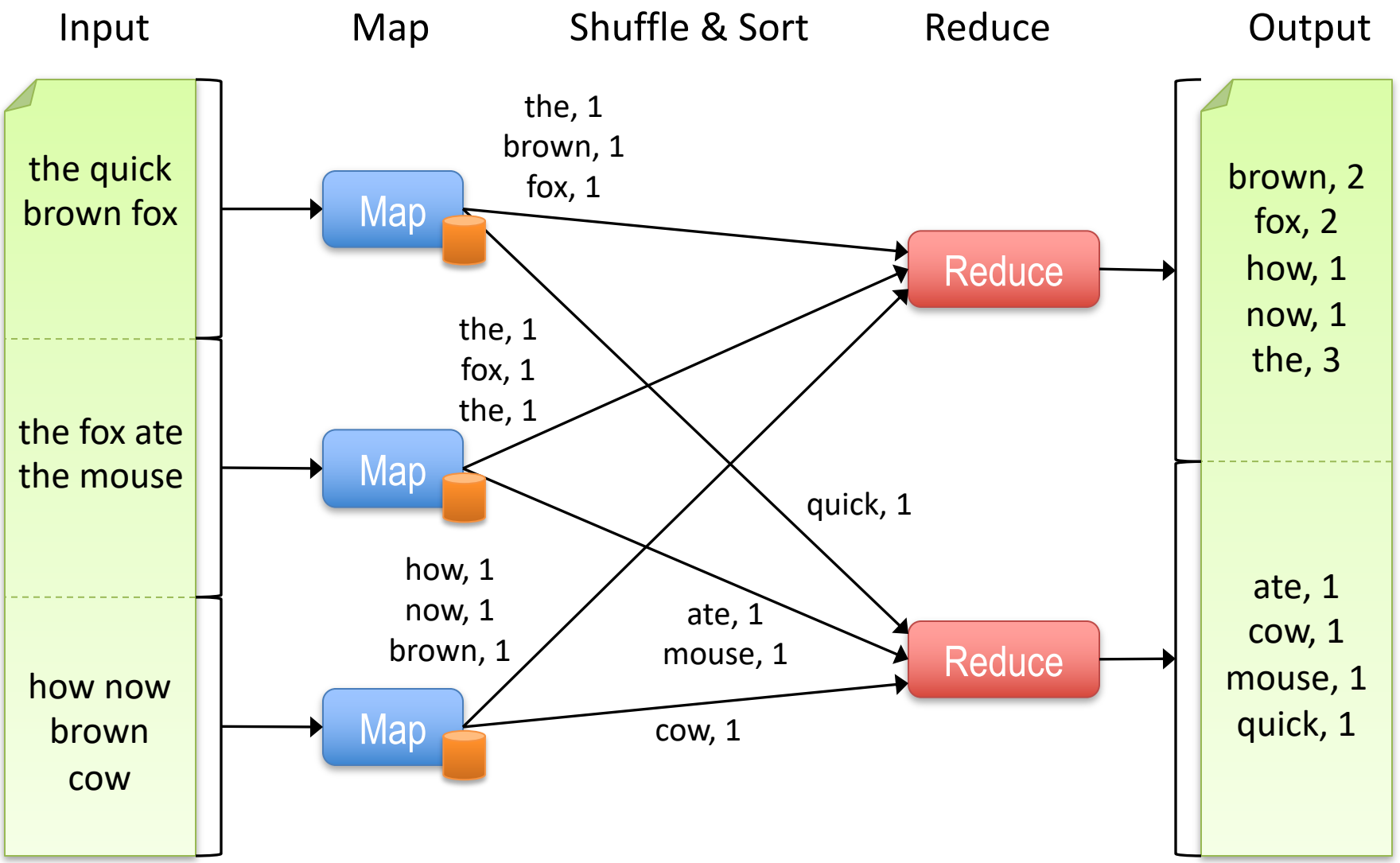
$$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$$

Example: Word Count

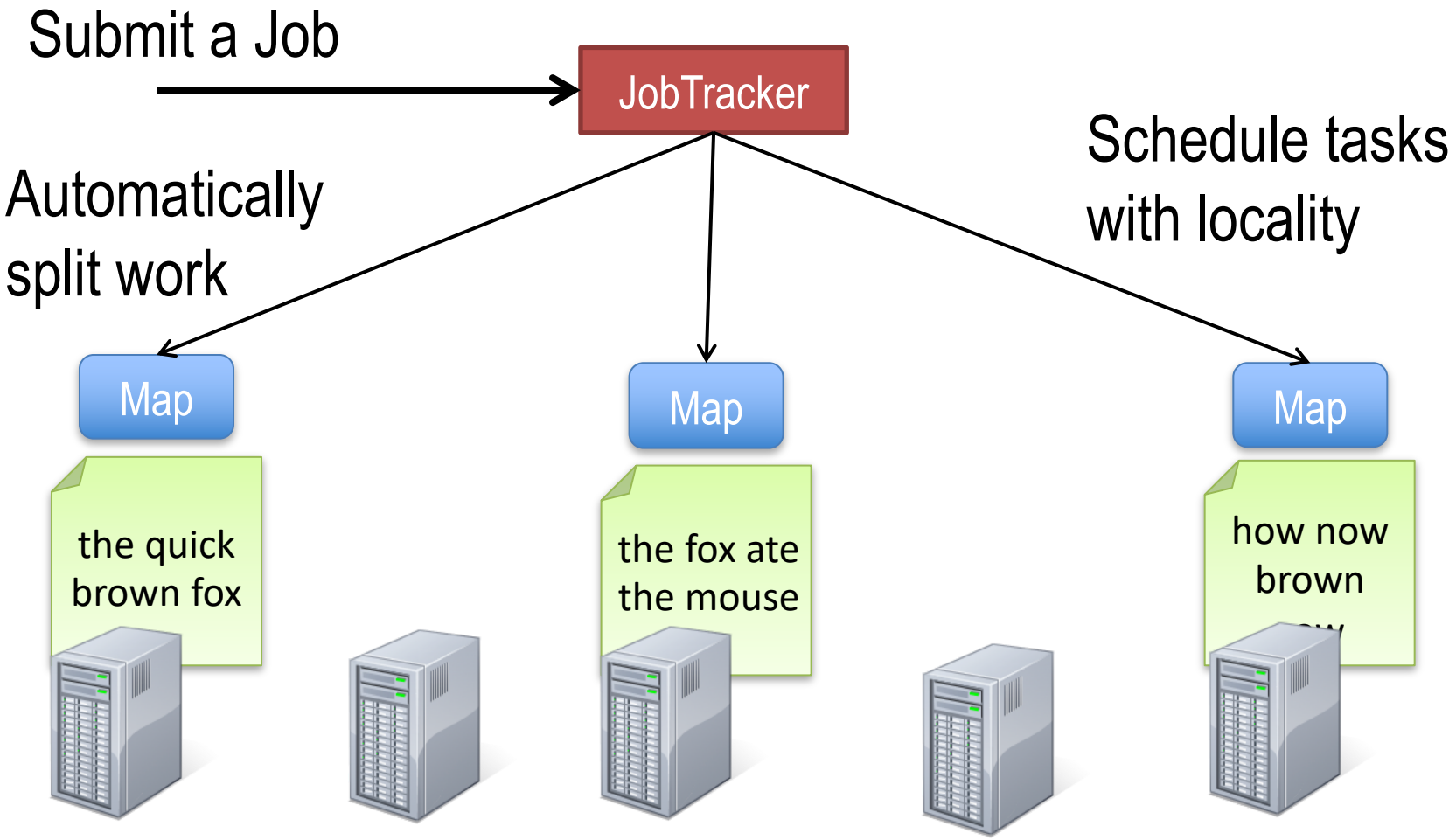
```
def mapper(line):
    for word in line.split():
        output(word, 1)


def reducer(key, values):
    output(key, sum(values))
```

Word Count Execution



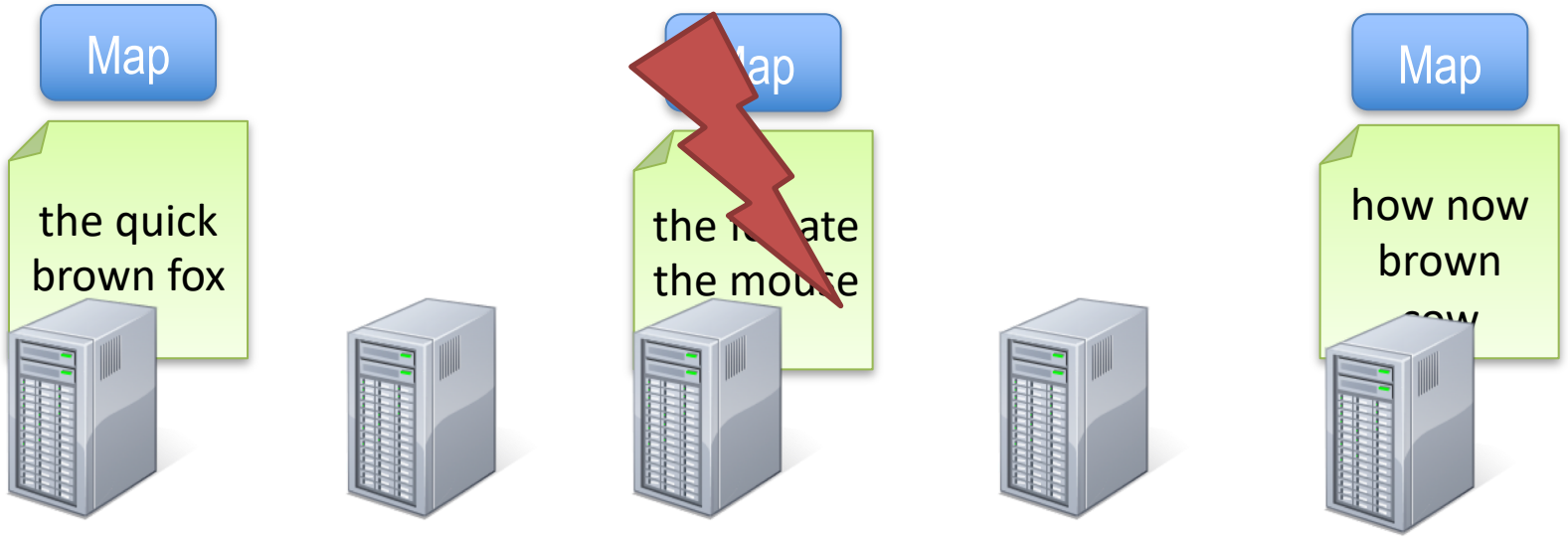
Word Count Execution



Fault Recovery

If a task crashes:

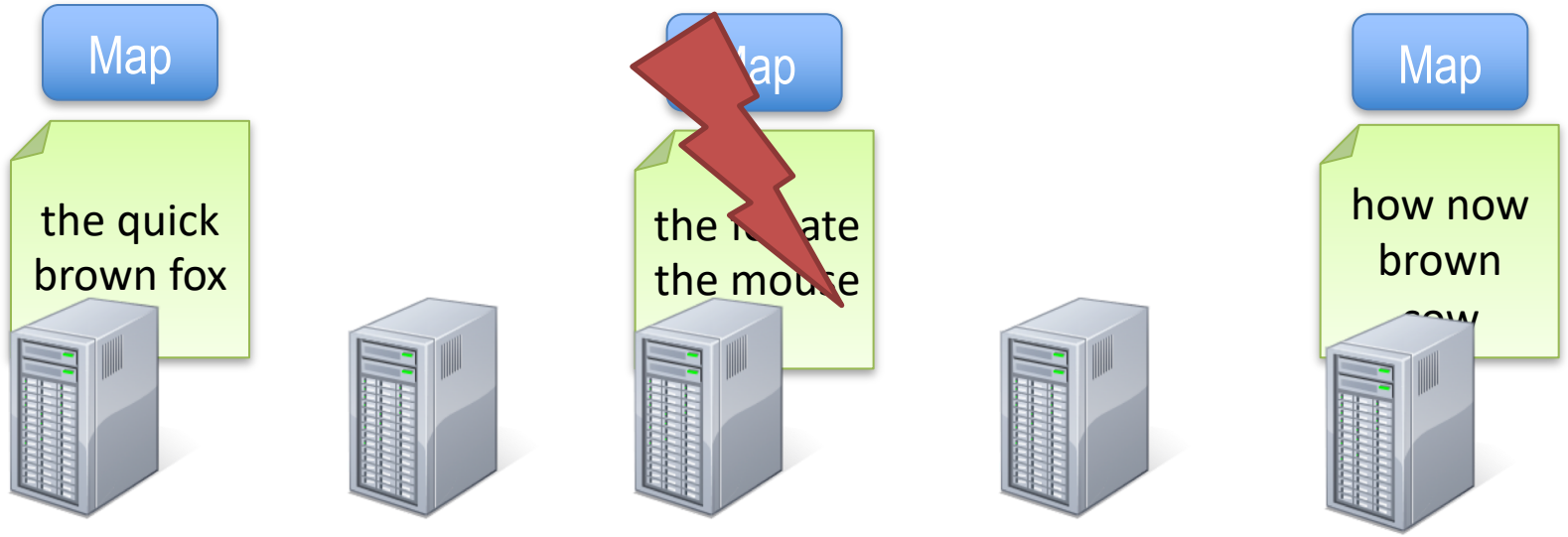
- Retry on another node
- If the same task repeatedly fails, end the job



Fault Recovery

If a task crashes:

- Retry on another node
- If the same task repeatedly fails, end the job



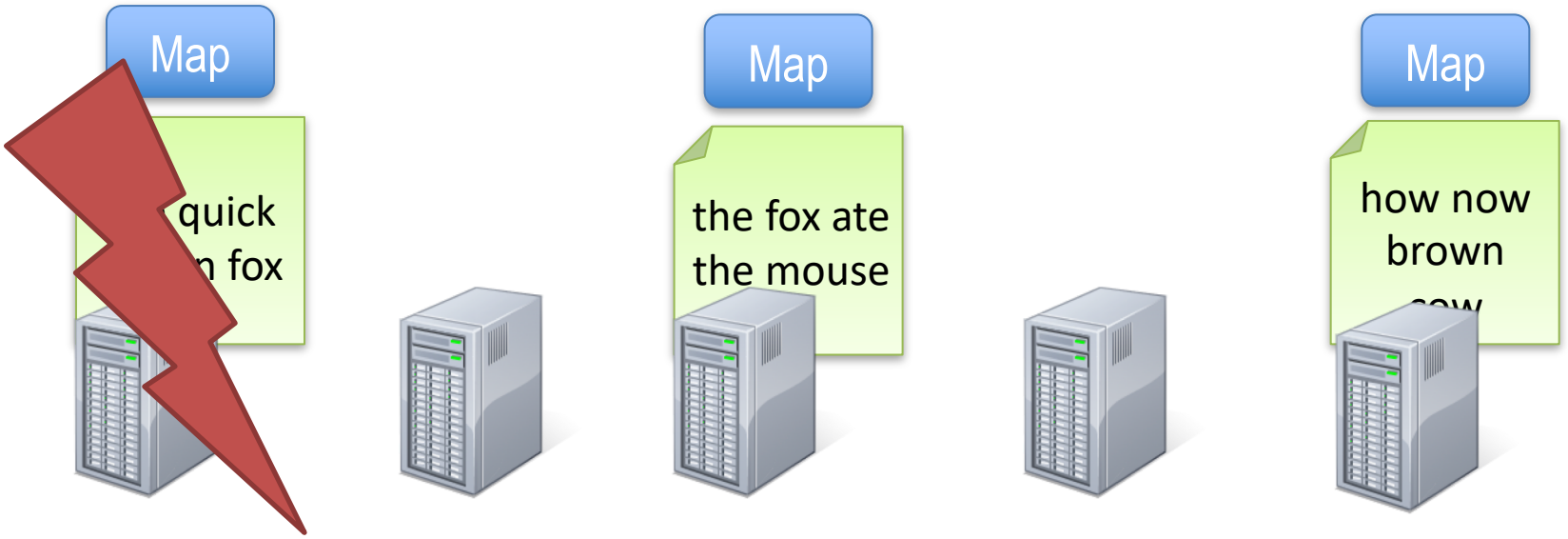
Requires user code to be **deterministic**

Fault Recovery

If a node crashes:

- Relaunch its current tasks on other nodes

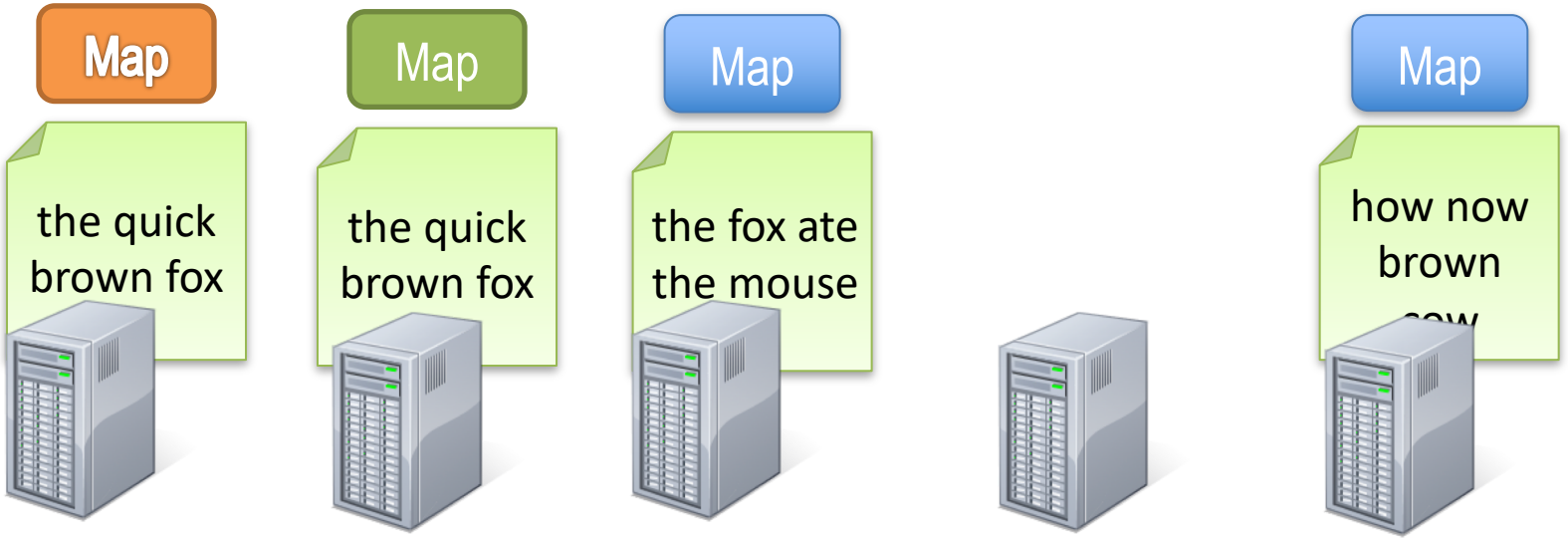
  What about task inputs ? File system replication



Fault Recovery

If a task is going slowly (straggler):

- Launch second copy of task on another node
- Take the output of whichever finishes first



MPI

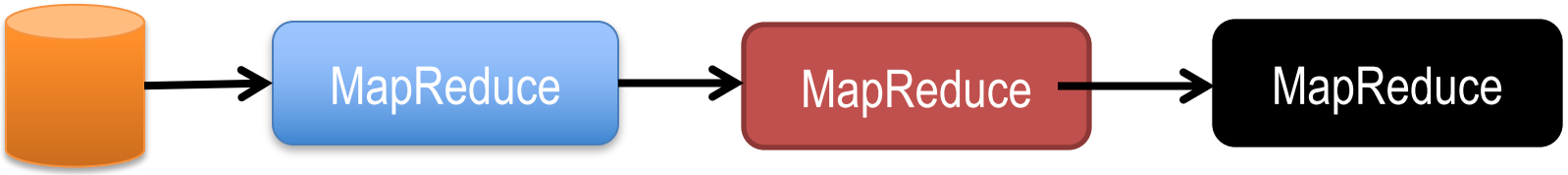
- Parallel process model
- Fine grain control
- Focus on High Performance

MapReduce

- High level data-parallel
- Automate locality, data transfers
- Focus on fault tolerance

When an Abstraction is Useful…

People want to compose it!



Most real applications require multiple MR steps

- Google indexing pipeline: 21 steps
- Analytics queries (e.g. sessions, top K): 2-5 steps
- Iterative algorithms (e.g. PageRank): 10's of steps

Programmability

Multi-step jobs create spaghetti code

- 21 MR steps → 21 mapper and reducer classes

Lots of boilerplate wrapper code per step

API doesn't provide type safety

Performance

MR only provides one pass of computation

- Must write out data to file system in-between

Expensive for apps that need to *reuse* data

- Multi-step algorithms (e.g. PageRank)
- Interactive data mining

Spark

Programmability: clean, functional API

- Parallel transformations on collections
- 5-10x less code than MR
- Available in Scala, Java, Python and R

Performance

- In-memory computing primitives
- Optimization across operators

Spark Programmability

Google MapReduce WordCount:

```
#include "mapreduce/mapreduce.h"

// User's map function
class SplitWords: public Mapper {
  public:
  virtual void Map(const MapInput& input)
  {
    const string& text = input.value();
    const int n = text.size();
    for (int i = 0; i < n; ) {
      // Skip past leading whitespace
      while (i < n && isspace(text[i]))
        i++;
      // Find word end
      int start = i;
      while (i < n && !isspace(text[i]))
        i++;
      if (start < i)
        Emit(text.substr(
            start,i-start),"1");
    }
  }
};

REGISTER_MAPPER(SplitWords);
```

```
// User's reduce function
class Sum: public Reducer {
  public:
  virtual void Reduce(ReduceInput* input)
  {
    // Iterate over all entries with the
    // same key and add the values
    int64 value = 0;
    while (!input->done()) {
      value += StringToInt(
                  input->value());
      input->NextValue();
    }
    // Emit sum for input->key()
    Emit(IntToString(value));
  }
};

REGISTER_REDUCER(Sum);
```

```
int main(int argc, char** argv) {
  ParseCommandLineFlags(argc, argv);
  MapReduceSpecification spec;
  for (int i = 1; i < argc; i++) {
    MapReduceInput* in= spec.add_input();
    in->set_format("text");
    in->set_filepattern(argv[i]);
    in->set_mapper_class("SplitWords");
  }

  // Specify the output files
  MapReduceOutput* out = spec.output();
  out->set_filebase("/gfs/test/freq");
  out->set_num_tasks(100);
  out->set_format("text");
  out->set_reducer_class("Sum");

  // Do partial sums within map
  out->set_combiner_class("Sum");

  // Tuning parameters
  spec.set_machines(2000);
  spec.set_map_megabytes(100);
  spec.set_reduce_megabytes(100);

  // Now run it
  MapReduceResult result;
  if (!MapReduce(spec, &result)) abort();
  return 0;
}
```

Spark Programmability

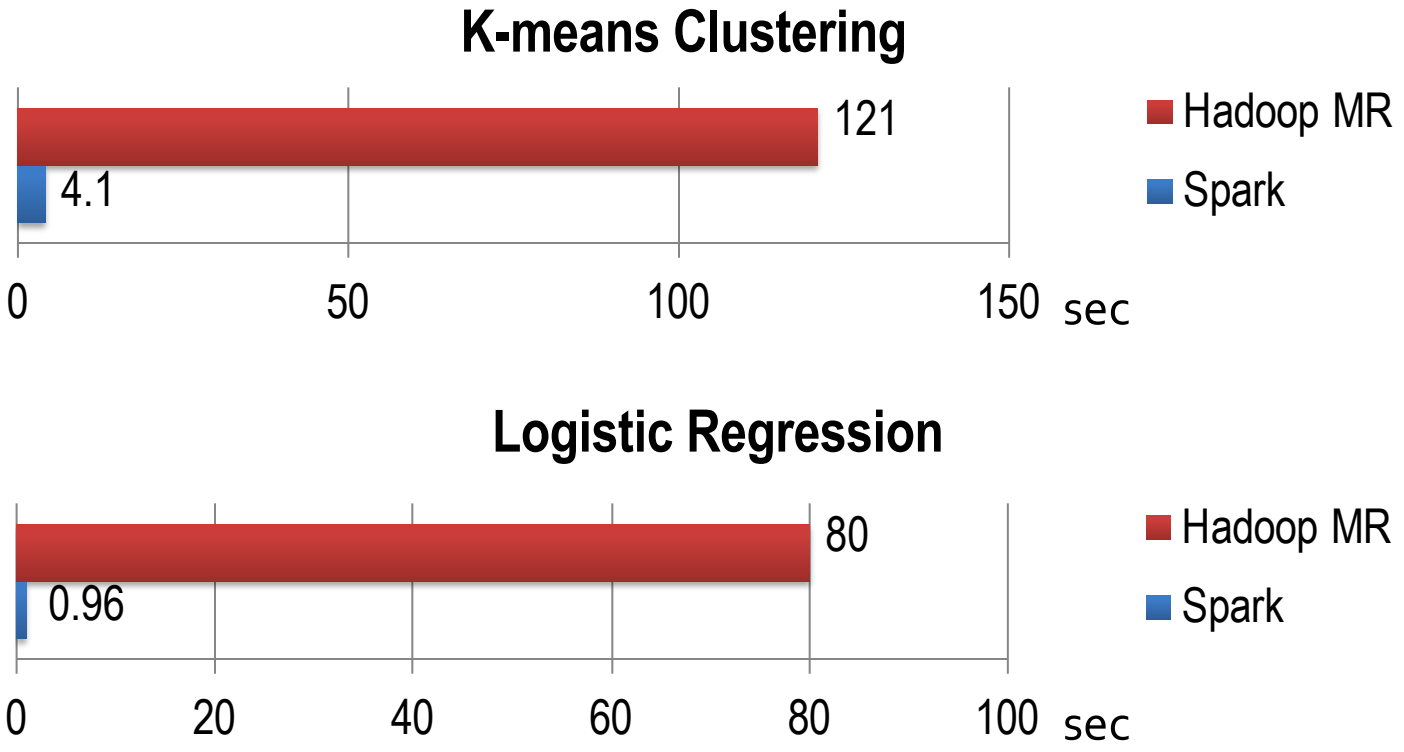
Spark WordCount:

```
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)

counts.save("out.txt")
```

Spark Performance

Iterative algorithms:



Spark Concepts

- Resilient distributed datasets (RDDs)
  - Immutable, partitioned collections of objects
  - May be cached in memory for fast reuse
- Operations on RDDs
  - *Transformations* (build RDDs)
  - *Actions* (compute results)
- Restricted shared variables
  - Broadcast, accumulators

Example: Log Mining

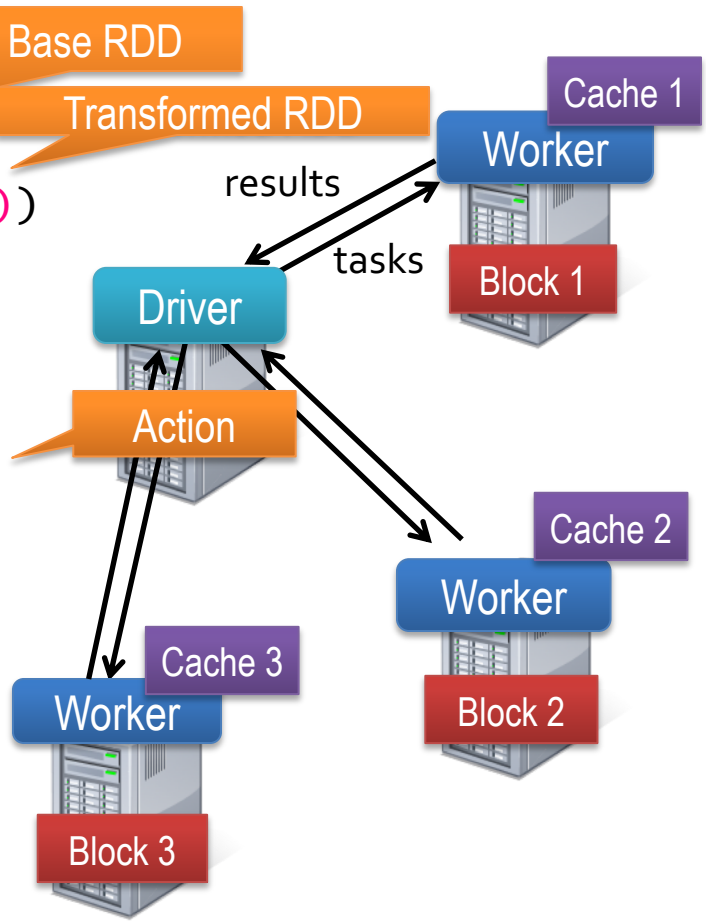
Find error messages present in log files interactively

(Example: HTTP server logs)

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
messages.cache()

messages.filter(_.contains("foo")).count
messages.filter(_.contains("bar")).count
. . .
```

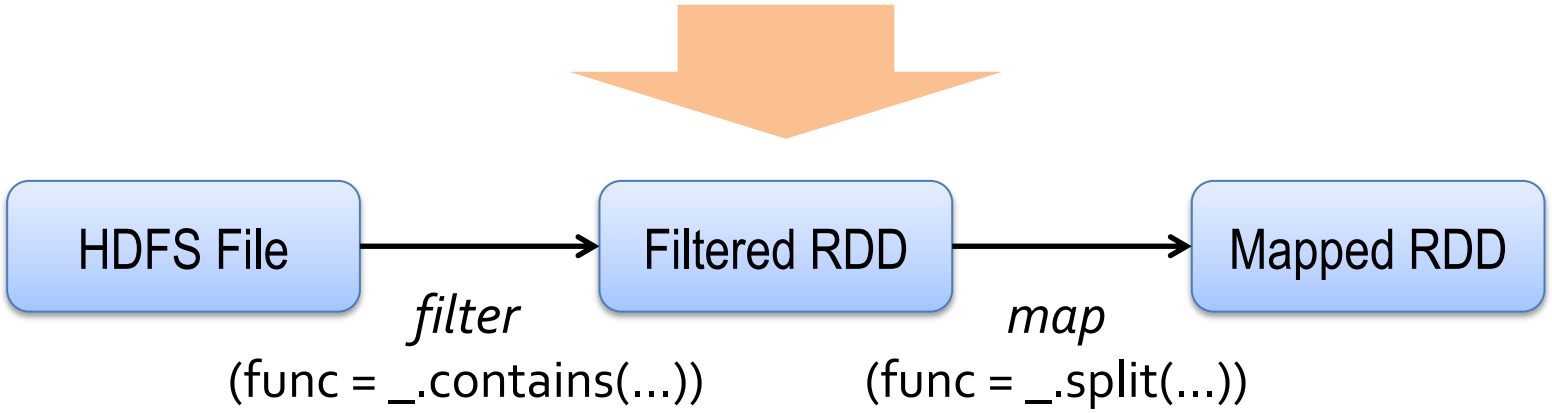
Result: search 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)



Fault Recovery

RDDs track *lineage* information that can be used to efficiently reconstruct lost partitions

Ex:
```
messages = textFile(...).filter(_.startsWith("ERROR"))
                        .map(_.split('\t')(2))
```



Pi in Java SPARK

```
List<Integer> l =
    new ArrayList<>(NUM_SAMPLES);
for (int i = 0; i < NUM_SAMPLES; i++) {
  l.add(i);
}
long count = sc.parallelize(l).filter(i -> {
            double x = Math.random();
            double y = Math.random();
            return x*x + y*y < 1; }).count();
System.out.println("Pi is roughly " +
                   4.0 * count / NUM_SAMPLES);
```

Other RDD Operations

Transformations (define a new RDD)	map filter sample groupByKey reduceByKey cogroup	flatMap union join cross mapValues ...
Actions (output a result)	collect reduce take fold	count saveAsTextFile saveAsHadoopFile ...

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains("error");
  }
}).count();
```

Python

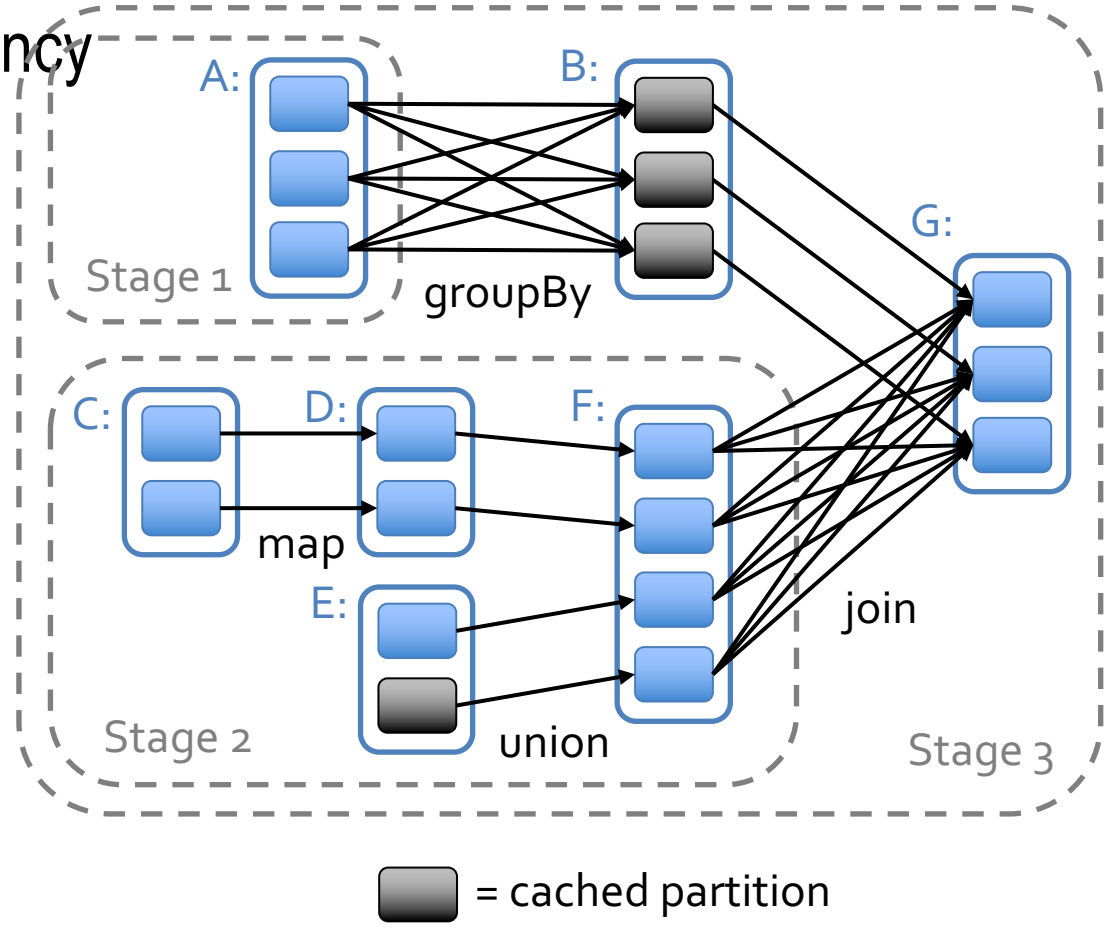
```
lines = sc.textFile(...)
lines.filter(lambda x: "error" in x).count()
```

R

```
lines ← textFile(sc, ...)
filter(lines, function(x) grepl("error", x))
```

Job Scheduler

- Captures RDD dependency graph
- Pipelines functions into “stages”
- Cache-aware for data reuse & locality
- Partitioning-aware to avoid shuffles



Higher-Level Abstractions

SparkStreaming: API for streaming data

GraphX: Graph processing model

MLLib: Machine learning library

SparkSQL: SQL queries, DataFrames

TensorFlowOnSpark: Deep learning

…

Need for DataFrames

Structured Data Processing

Read in CSV, JSON, JDBC etc.

Data source for Machine Learning

```
glm(a ~ b + c, data = df)
```

Functional transformations not intuitive

Example: Column Average

```
peopleRDD <- textFile(sc, “people.txt”)

lines <- flatMap(peopleRDD,
                 function(line) {
                   strsplit(line, ", ")
                 })
```

Column Average using RDD

```
peopleRDD <- textFile(sc, “people.txt”)

lines <- flatMap(peopleRDD,
                 function(line) {
                   strsplit(line, ", ")
                 })

ageInt <- lapply(lines,
                 function(line) {
                   as.numeric(line[2])
                 })

sum <- reduce(ageInt, function(x, y) {x+y})
avg <- sum / count(peopleRDD)
```

Column Average using DataFrames

```
# JSON File contains two columns age, name
df <- jsonFile(“people.json”)

avg <- select(df, avg(df$age))
```

Scientific Computing on Spark

Computation

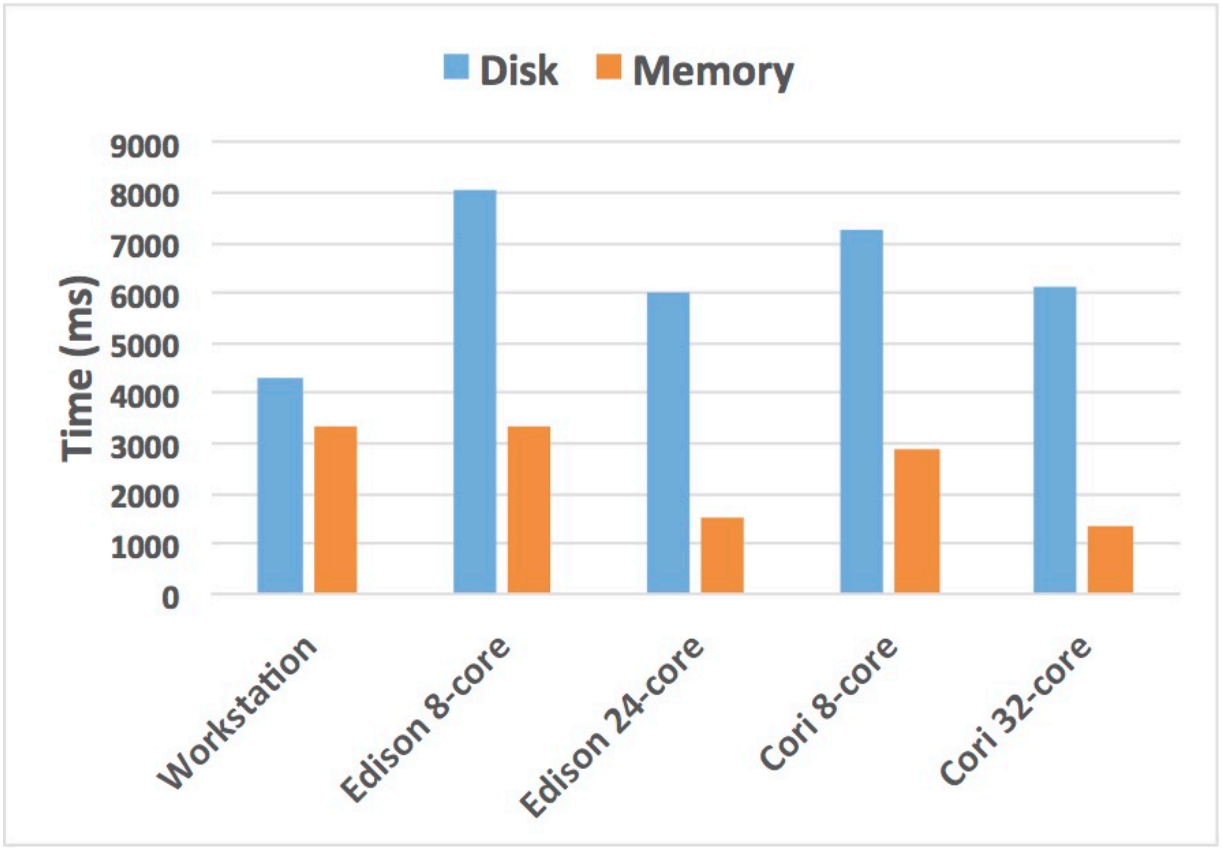
- Efficient native operations using JNI
- Use BLAS / OpenMP per-node

Communication

- Limited programming model
- Shuffle could add latency, bandwidth

Supercomputer Node 2x Slower Than Workstation

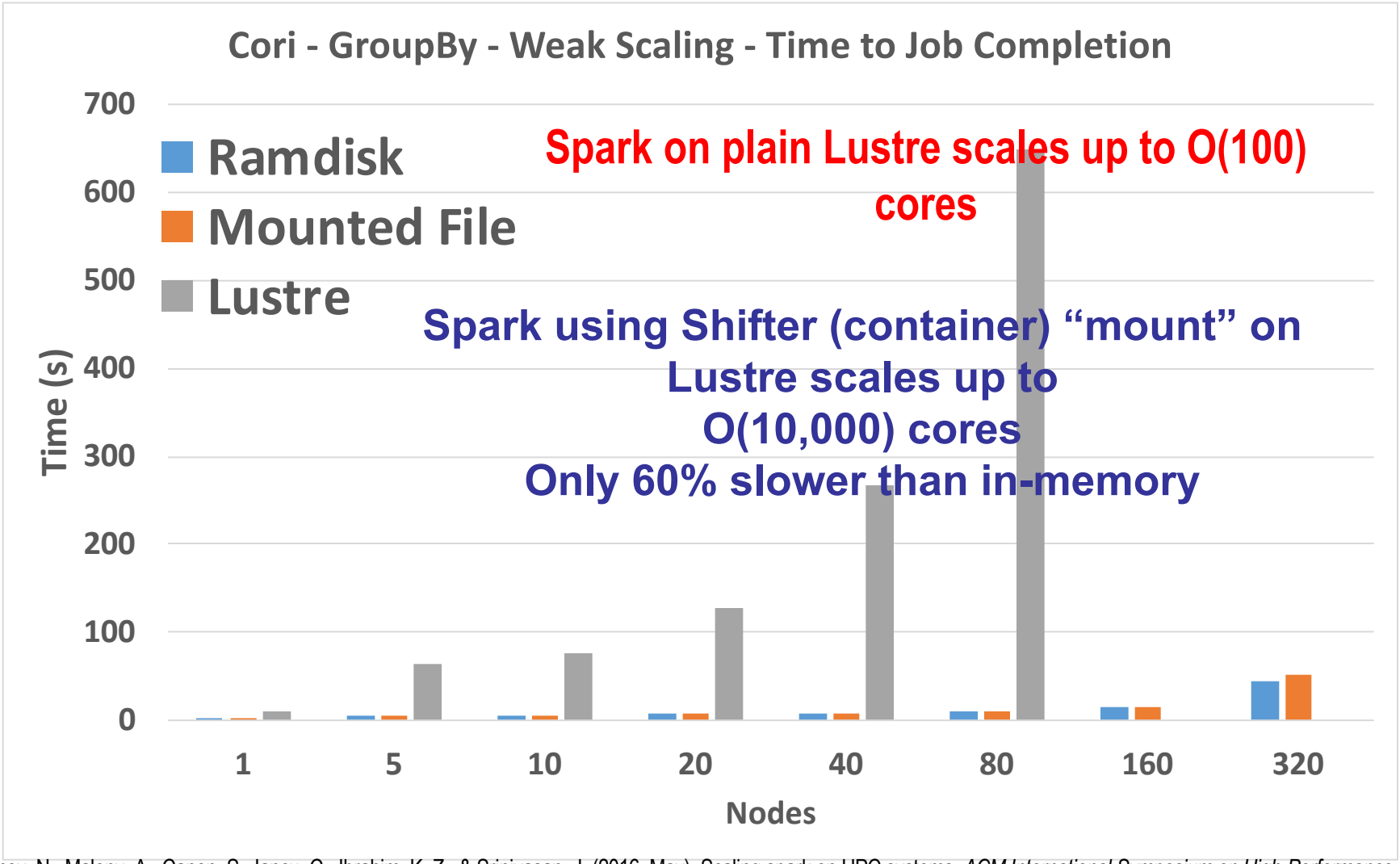
Spark SQL Big Data Benchmark



I/O is the problem: Supercomputer matches workstation when data is cached

Chaimov, N., Malony, A., Canon, S., Iancu, C., Ibrahim, K. Z., & Srinivasan, J. (2016, May). Scaling spark on HPC systems. *ACM International Symposium on High-Performance Parallel and Distributed Computing*

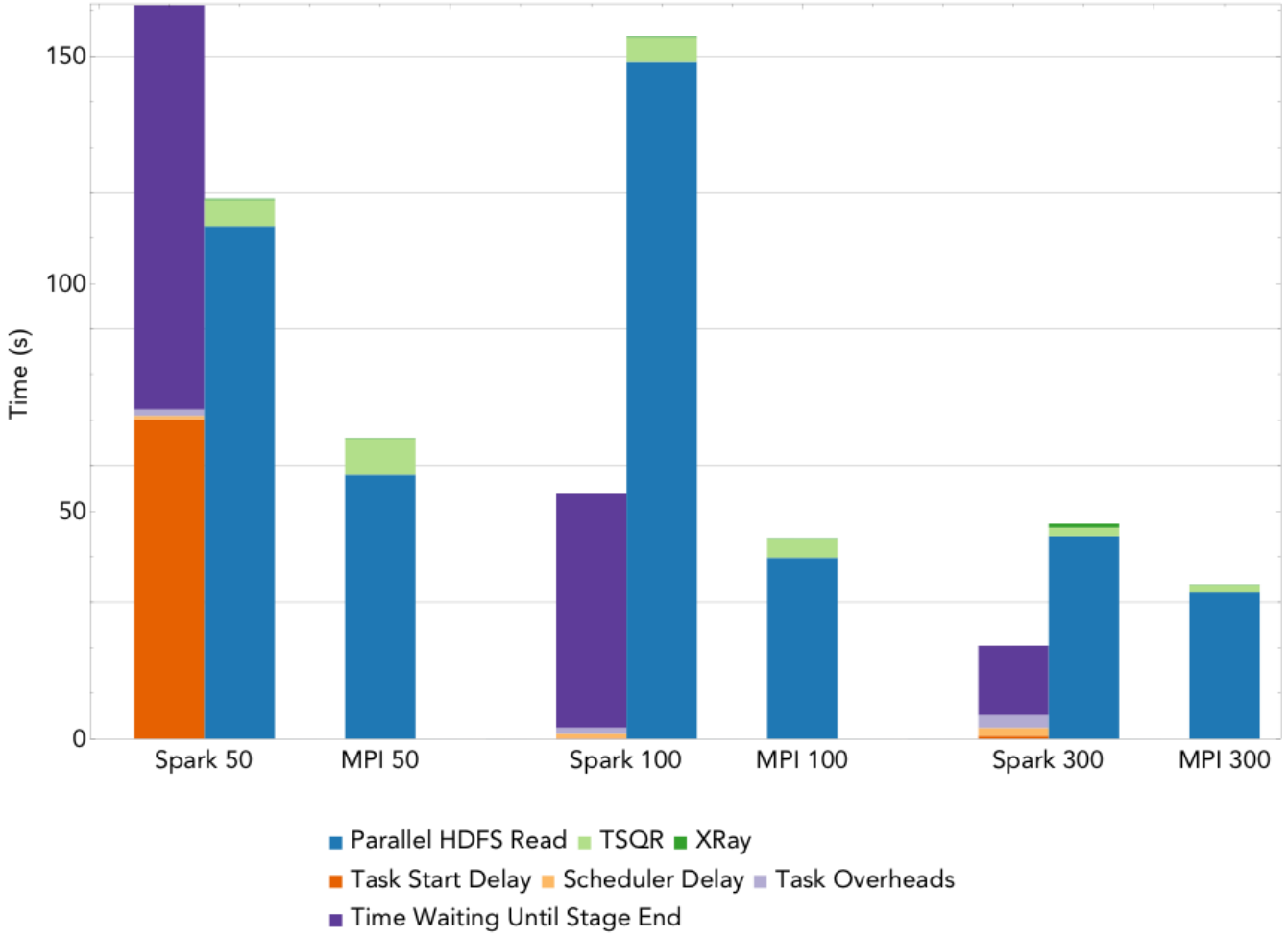
Scalability with Shifter



Chaimov, N., Malony, A., Canon, S., Iancu, C., Ibrahim, K. Z., & Srinivasan, J. (2016, May). Scaling spark on HPC systems. *ACM International Symposium on High-Performance Parallel and Distributed Computing*

Scientific Computing on SPARK

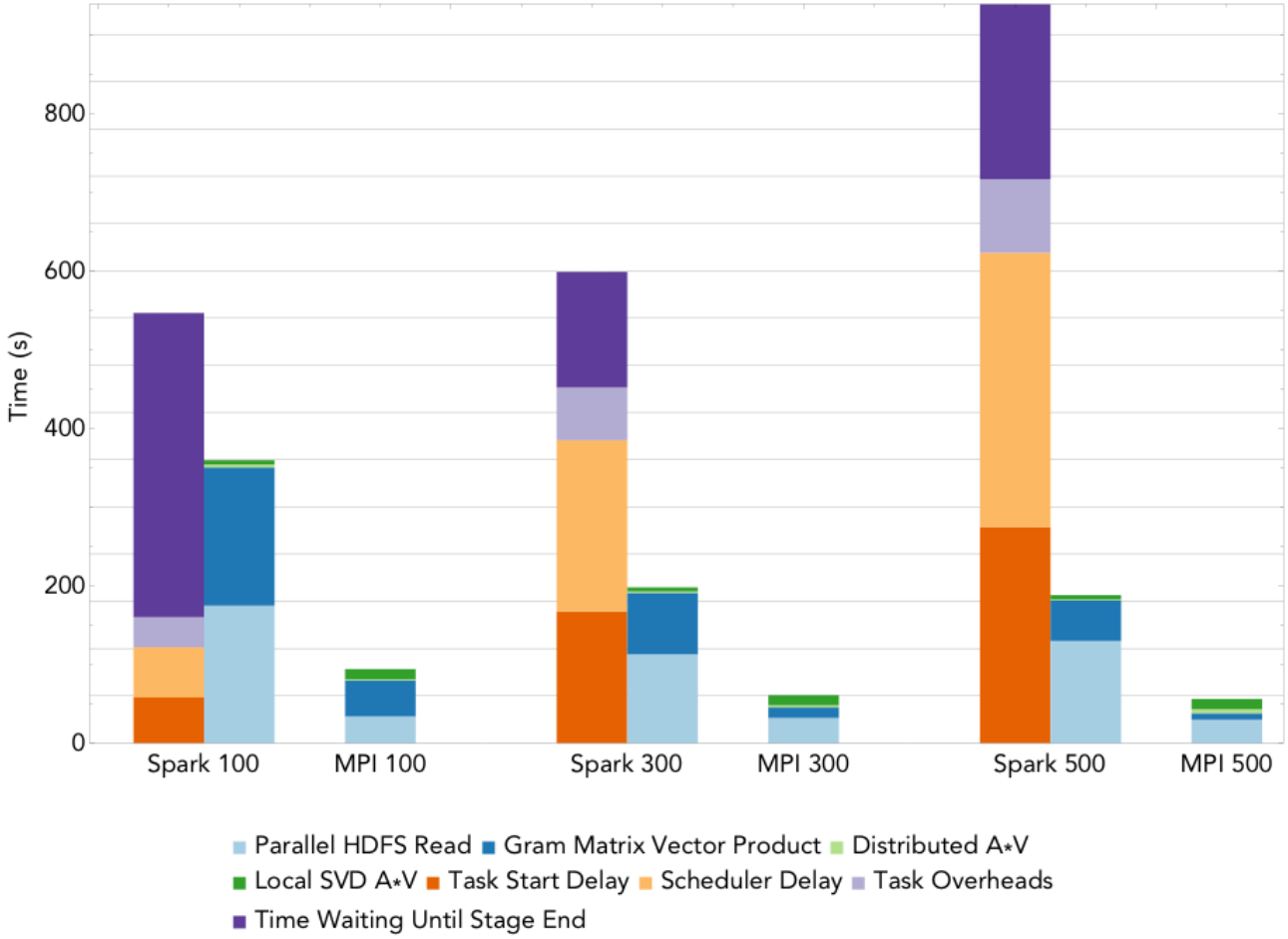
Running time for a rank 10 approximation using NMF on the 1.6TB Daya Bay matrix



Gittens, A., Devarakonda, A., Racah, E., Ringenburg, M., Gerhardt, L., Kottalam, J., ... & Sharma, P. (2016, December). Matrix factorizations at scale: A comparison of scientific data analytics in Spark and C+ MPI using three case studies. In *Big Data (Big Data)*.

Scientific Computing on SPARK

Running time for PCA on the 2.2TB Ocean matrix on Cori



Gittens, A., Devarakonda, A., Racah, E., Ringenburg, M., Gerhardt, L., Kottalam, J., ... & Sharma, P. (2016, December). Matrix factorizations at scale: A comparison of scientific data analytics in Spark and C+ MPI using three case studies. In *Big Data (Big Data)*.

Scientific Computing on Spark

Large Scale Machine Learning (AMPLab, Aspire)

- KeystoneML: Framework for ML
- Improved Algorithms

Spark on HPC systems (NERSC, Cray, AMPLab)

- Real world applications
- Profiling, hardware customizations

Spark Adoption

Open source Apache Project, > 400 contributors

Packaged by Cloudera, Hortonworks

Databricks: Spark as a cloud service

Unified Platform for Big Data Applications

Conclusion

Commodity clusters used for big data

Key challenges: Fault tolerance, stragglers

HPC systems used for simulation

Emerging challenges: Faults and variability

Cloud programming: MapReduce and Spark

- Simplify programming
- Handle faults automatically