Scalability

28-01-2016
Mathijs Kattenberg - mathijs.kattenberg@surfsara.nl
Jeroen Schot - jeroen.schot@surfsara.nl
Machiel Jansen - machiel.jansen@surfsara.nl

SURF SARA

Bottleneck Sources

- Fundamental system/application characteristics
- Application design
- Human effort/involvement

SURF SARA

Fundamental Bottlenecks

- Machine limits
- Latencies
- Sequential versus random processing
- Machine/network failures
- Algorithm characteristics

Machine Limits

Current system limits:

- ~128 CPU cores
- 2TB of RAM
- ~500 TB disk space

= expensive (cost does not scale linearly)

SURF SARA

Latencies

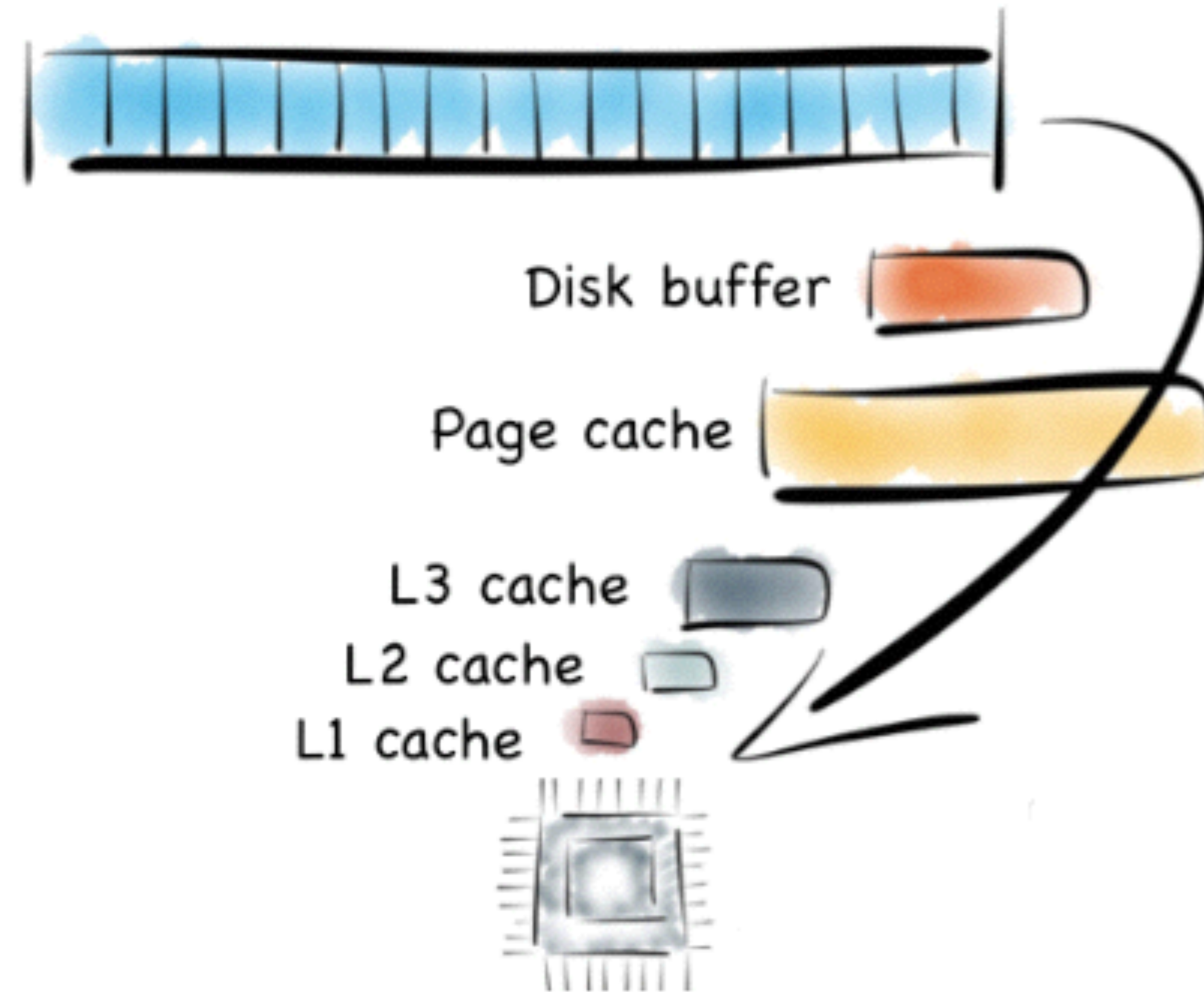
http://bit.ly/1aGoF5l

Table 2.2 Example Time Scale of System Latencies

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

SURF SARA

Sequential vs Random



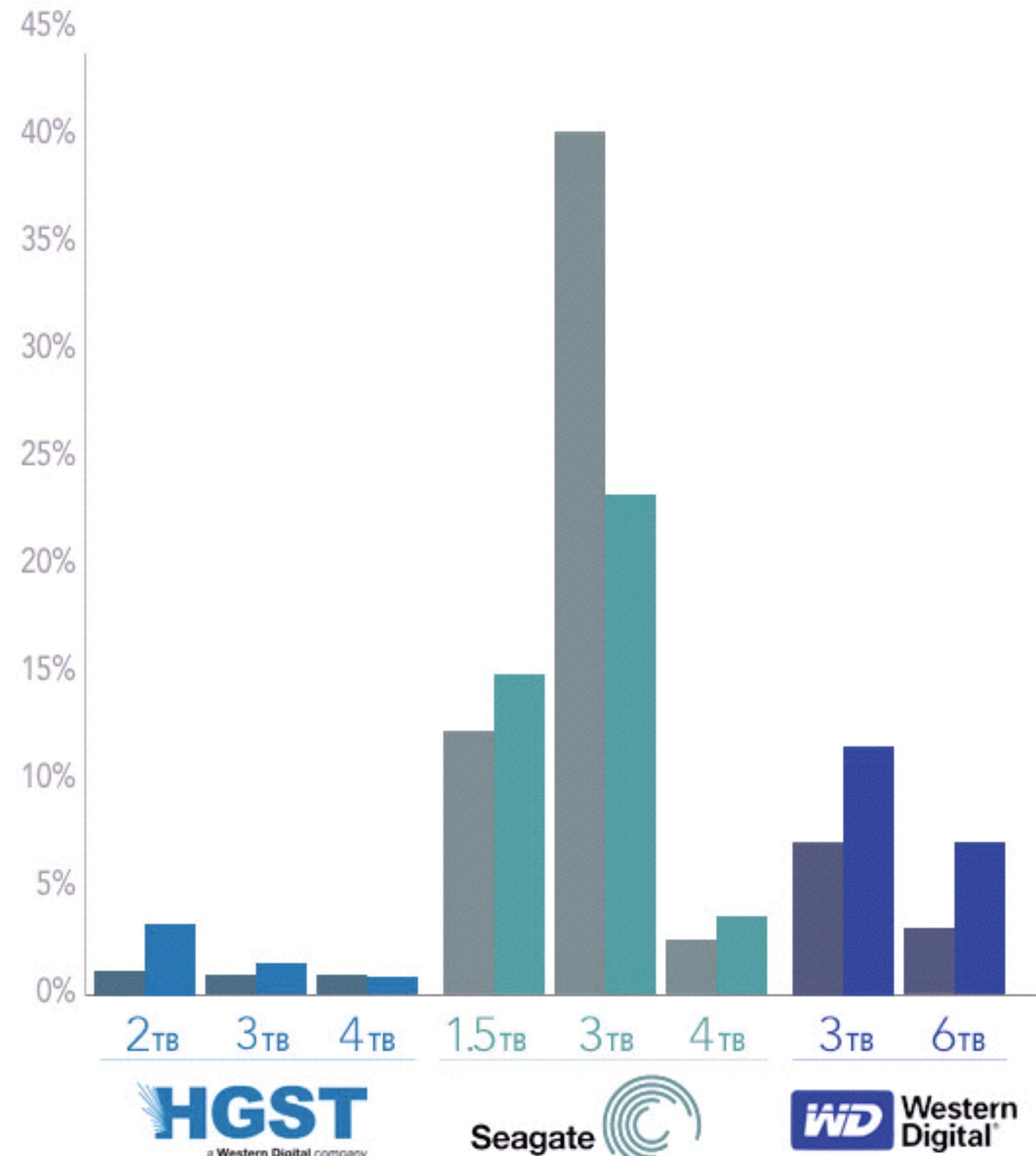
Courtesy Ben Stopford: http://bit.ly/1JNyBtx

SURF SARA

Failures

Backblaze cloud storage:

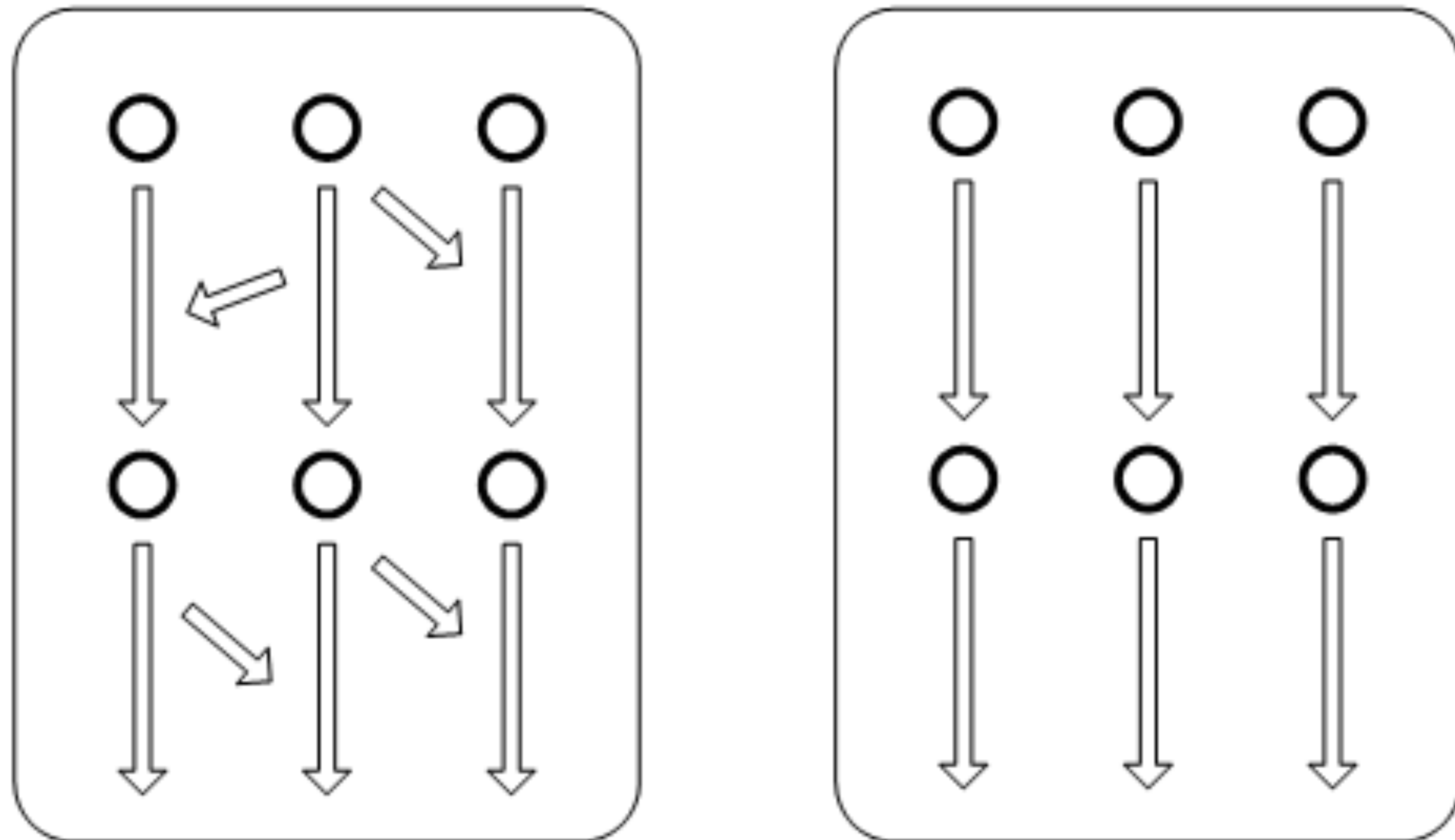
- ~ 46K disks
- http://bit.ly/1V5Gbq7

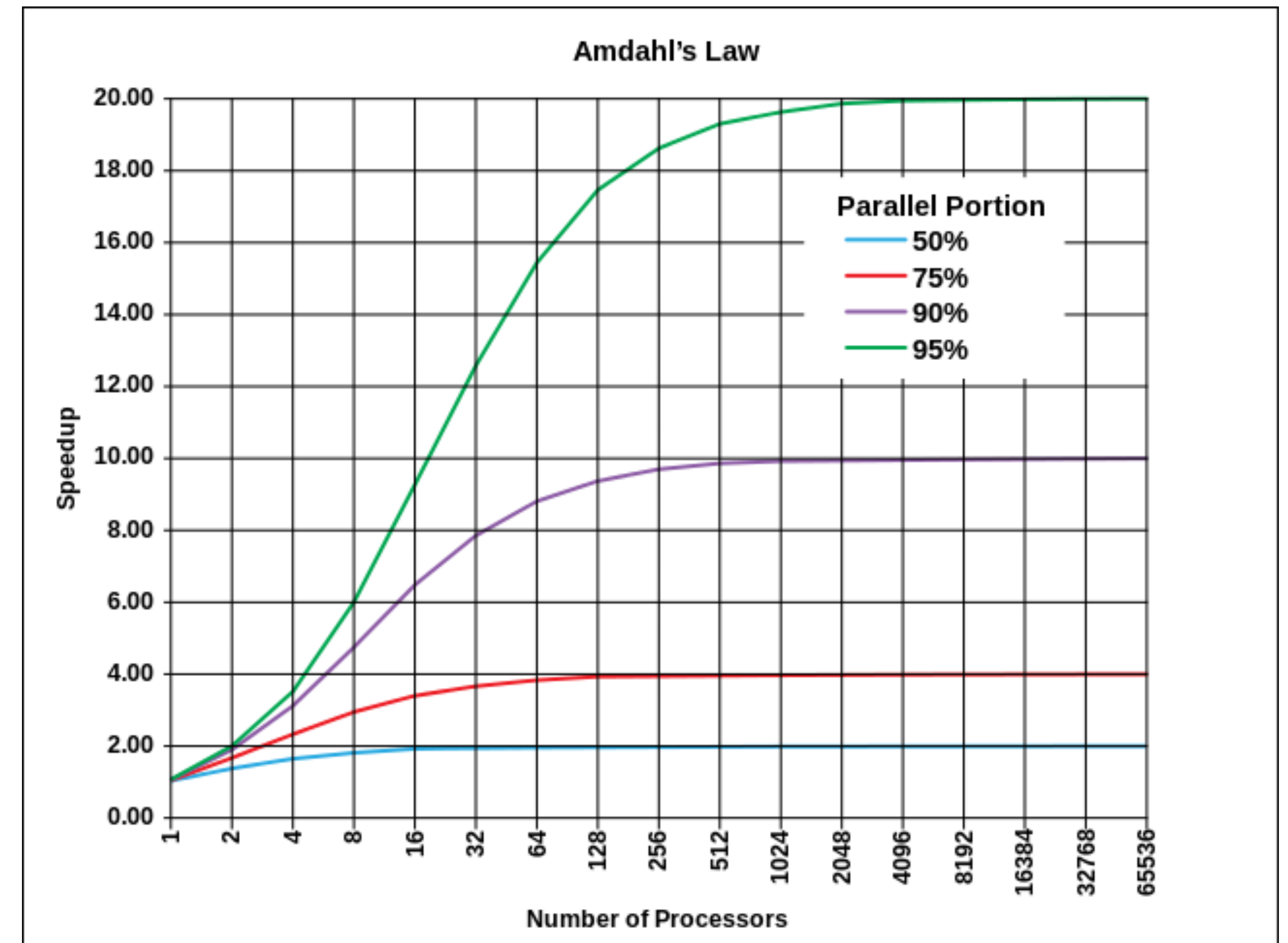


SURF SARA

Algorithm Characteristics

Fine grained, coarse grained, embarrassingly (data) parallel:





SURF SARA

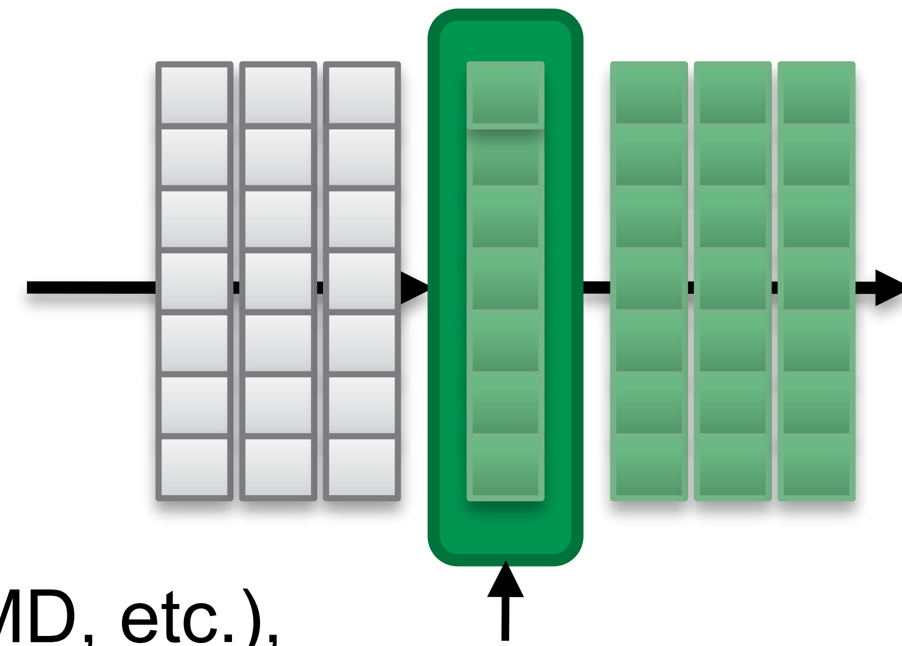
Algorithm Characteristics

mutable state + parallel processing = non-determinism

```
x = 0

async { x = x + 1}

async { x = x * 2}

# can give 0, 1, 2
```

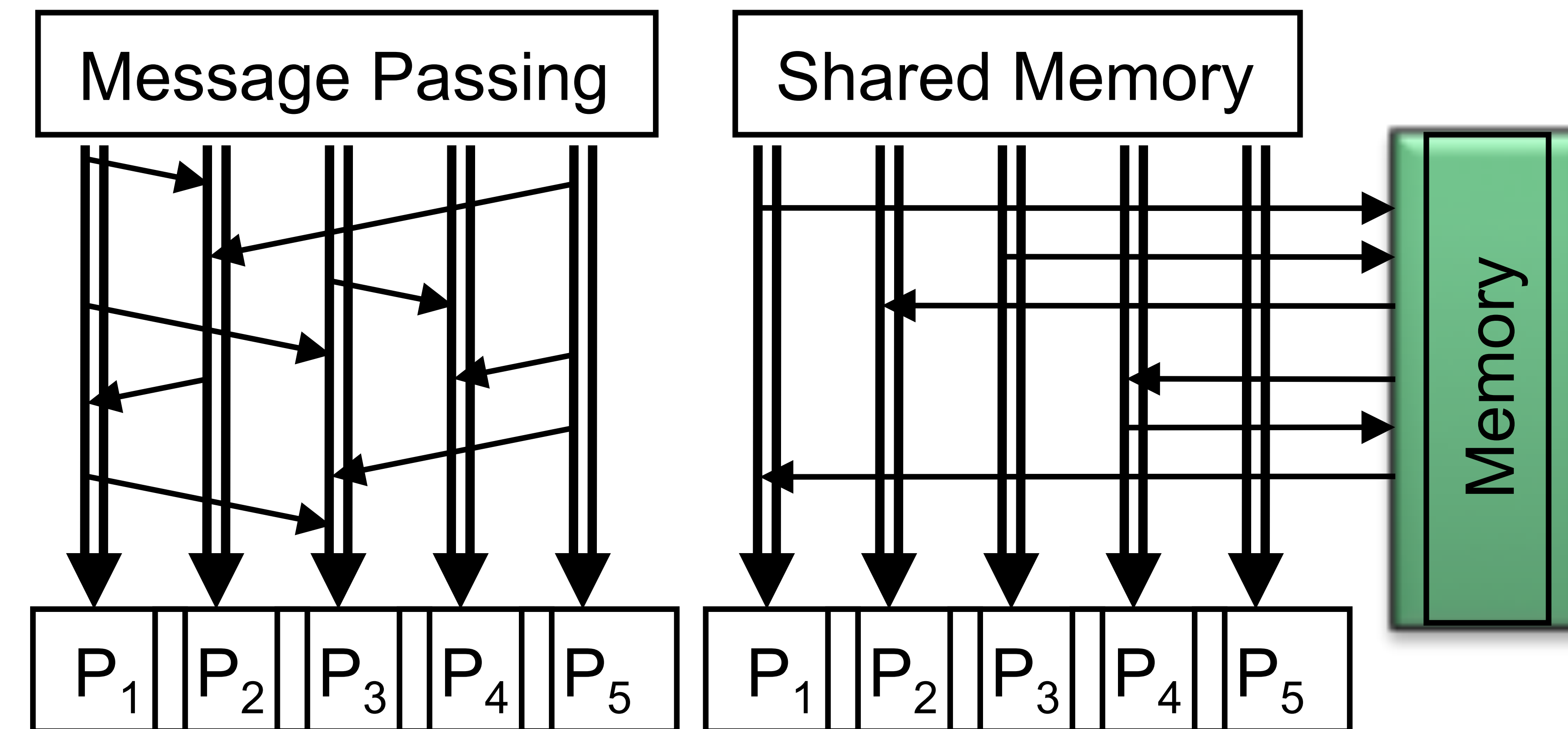
SURF SARA

scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …



Flynn's taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth UMA vs. NUMA, cache coherence

Different programming models

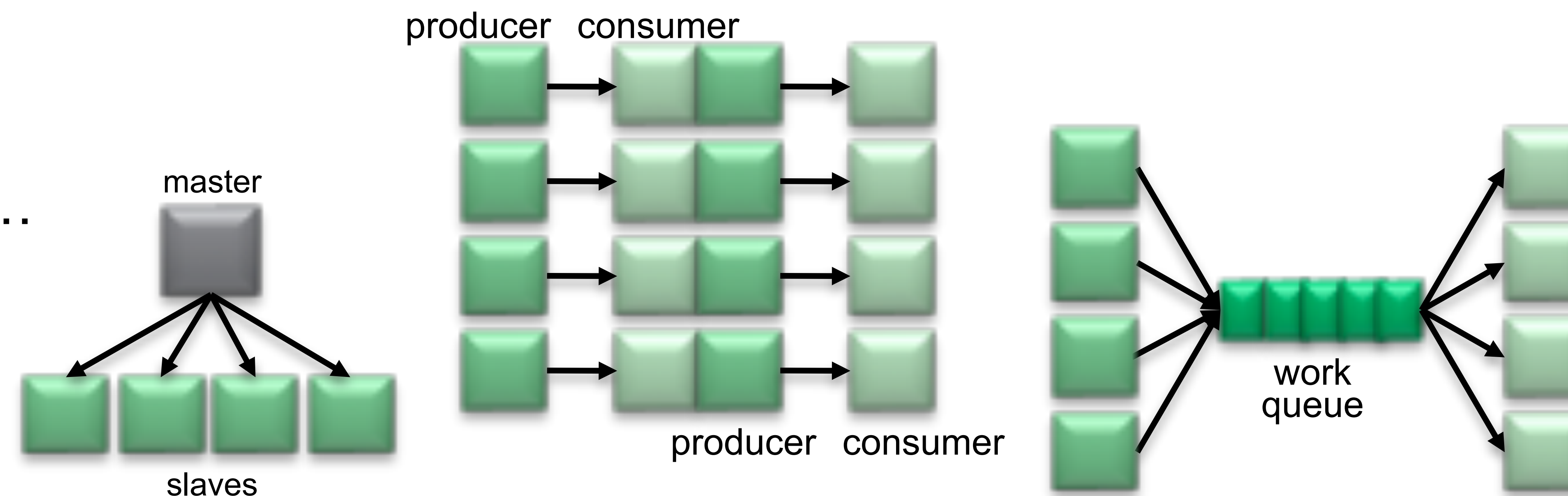


Common problems

livelock, deadlock, data starvation, priority inversion…
dining philosophers, sleeping barbers, cigarette smokers, …

Different programming constructs

mutexes, conditional variables, barriers, …
masters/slaves, producers/consumers, work queues, …



Programmer shoulders the burden of managing concurrency…

SURF SARA

```
inputFile = open('document.txt')

counts = {}
for line in inputFile:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

for word in counts:
    print word, counts[word]
```

SURF SARA

```
inputFile = open('document.txt')

counts = {}
for line in inputFile:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

for word in counts:
    print word, counts[word]
```

SURF SARA

```
inputFile = open('document.txt')

counts = {}
for line in inputFile:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

for word in counts:
    print word, counts[word]
```

SURF SARA

```
inputFile = open('document.txt')

counts = {}
for line in inputFile:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

for word in counts:
    print word, counts[word]
```

SURF SARA

Scalability: Design

- Data is growing faster than computing power and IO

  => distributed computing necessary

- Most standard applications cannot run in a distributed fashion

  => Applications need to be designed with scalability from the start

Scalability: Design

Idea: take a step back and consider:

- Work without mutable state
- Restrict the programming interface so that more can be done automatically.

Turns out: we can use ideas from functional programming

SURF SARA

Functional Programming

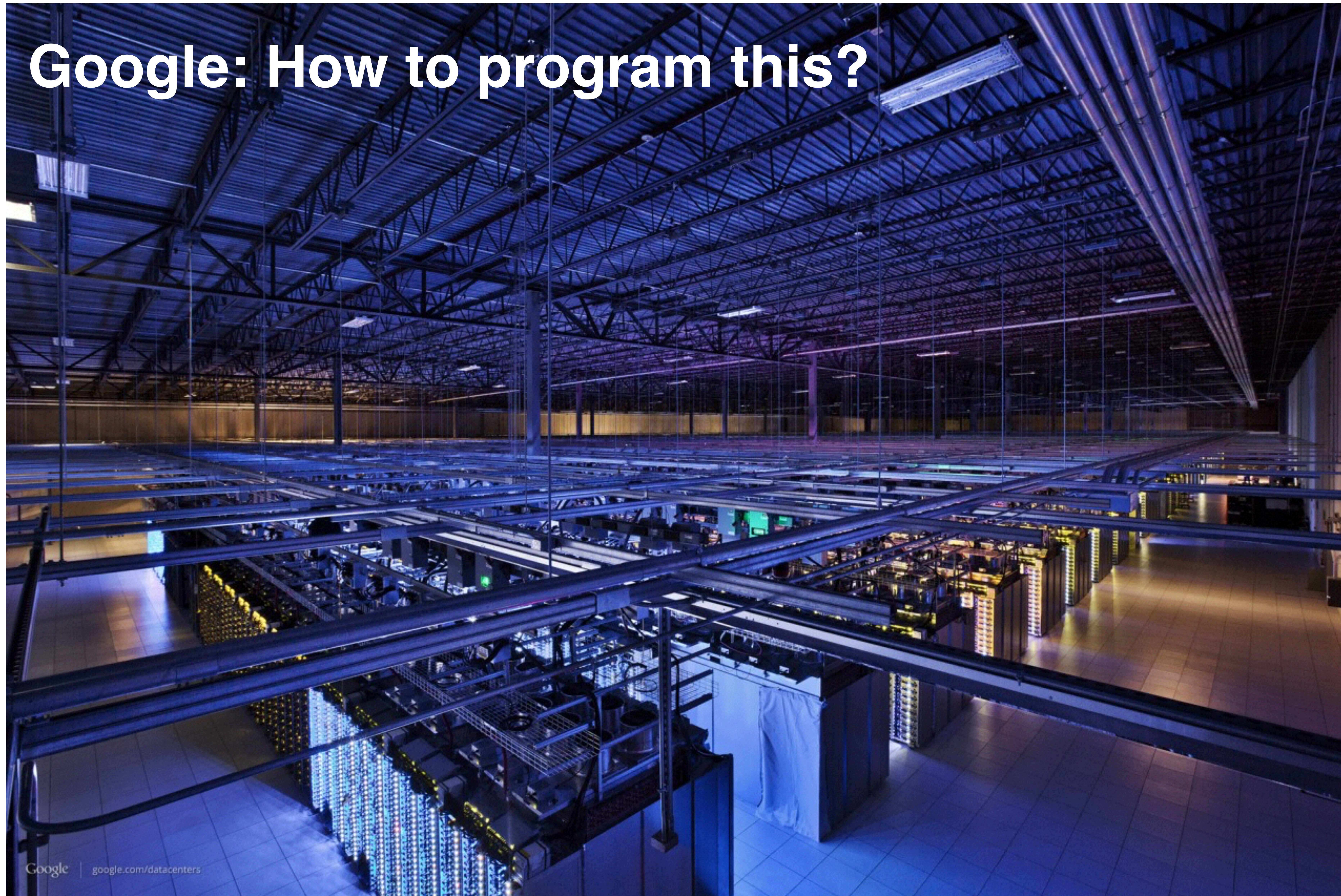
Restrict the programming interface so that the system can do more automatically. Use ideas from functional programming:

"Here is a function, apply it to all of the data"

- I do not care where it runs (the system should handle that)
- Feel free to run it twice on different nodes (no side effects!)

SURF SARA

Google: How to program this?



A Data-Parallel Approach

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

…@google.com, sanjay@google.com

…e, Inc.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the compo-

SURF SARA

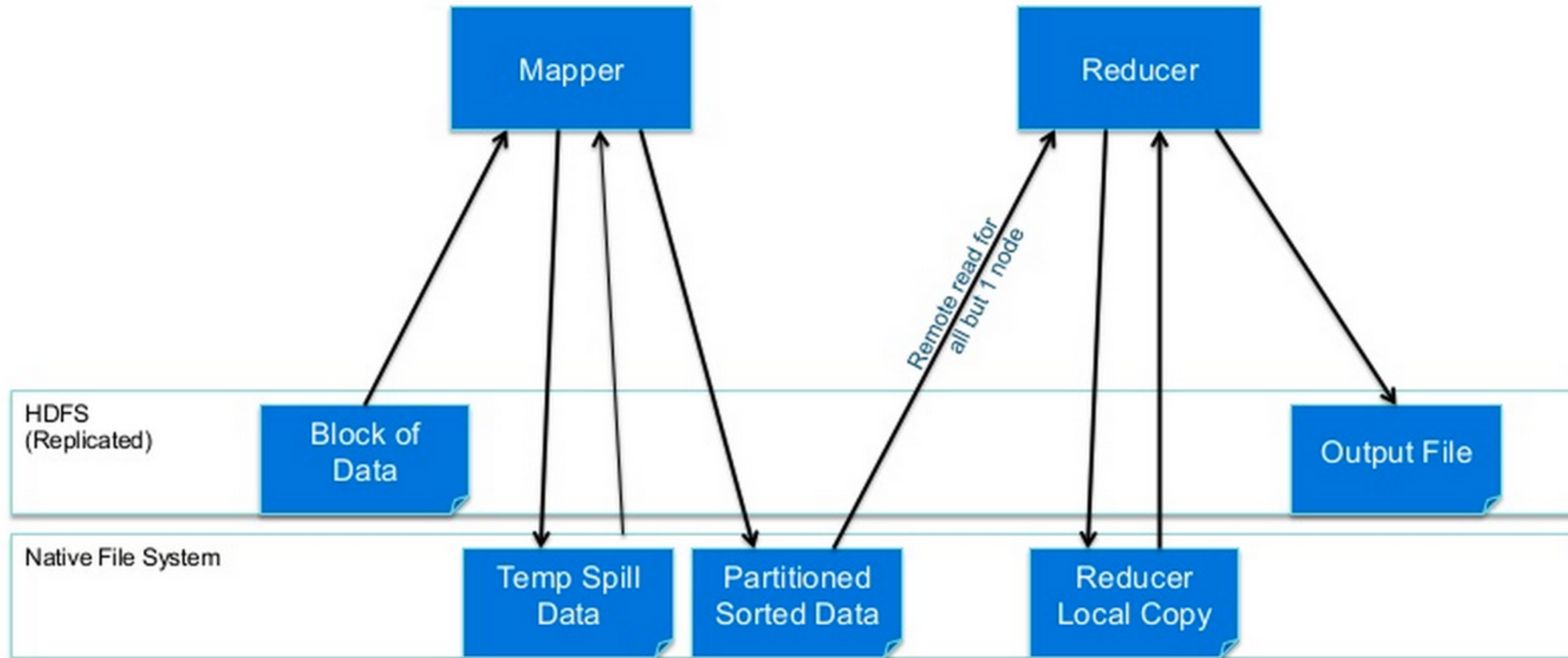
MapReduce

"A simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs."

Jeffrey Dean and Sanjay Ghemawat , "MapReduce: Simplified Data Processing on Large Clusters", Proceedings of the 6th OSDI Symposium, 2004

SURF SARA

MapReduce Basic High Level

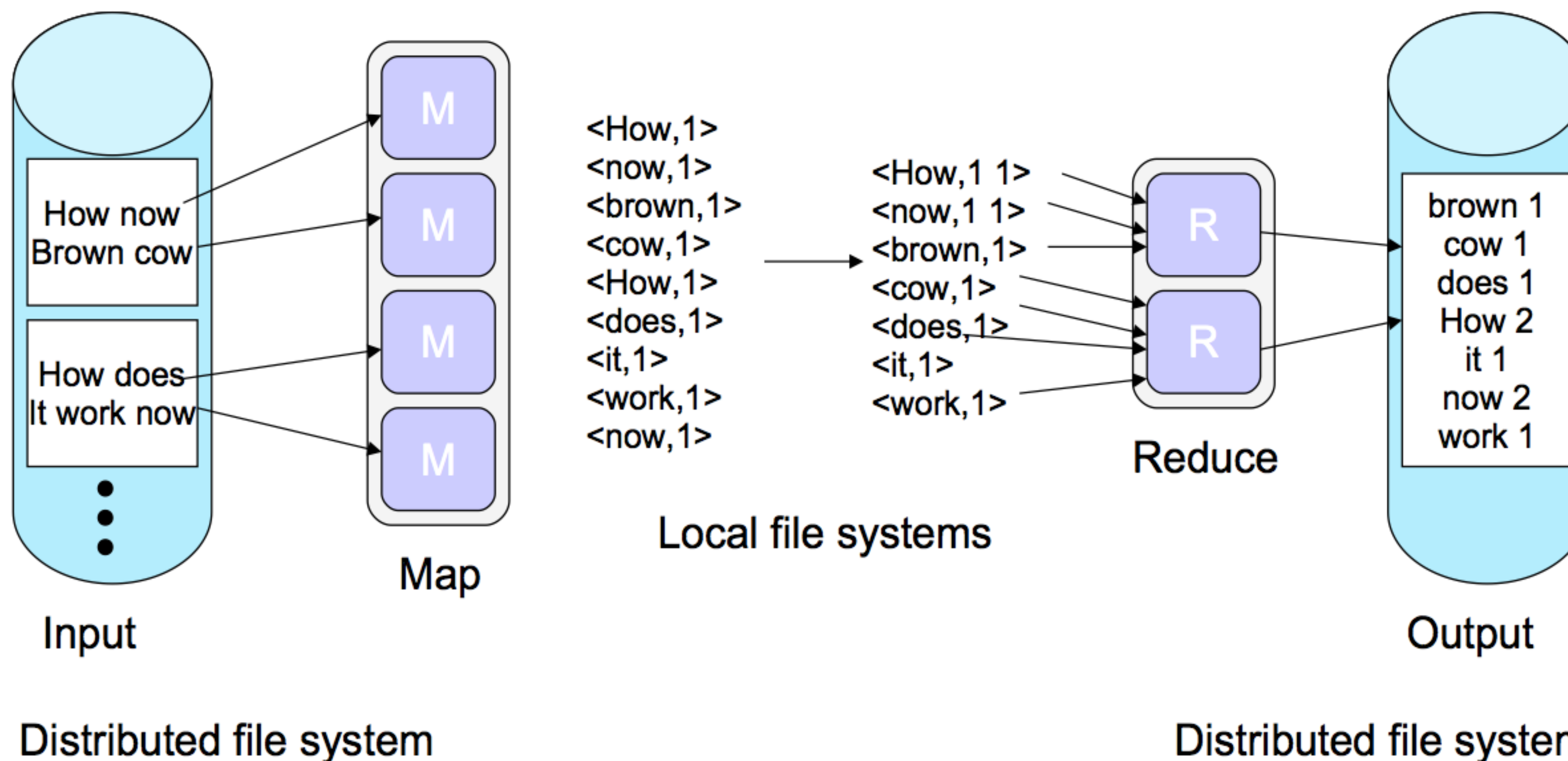


SURF SARA

MapReduce Programming Model

Map function: $(K_1, V_1) \longrightarrow list(K_2, V_2)$

Reduce function: $(K_2, list(V_2)) \longrightarrow list(K_3, V_3)$



SURF SARA

Date	Time	UserID	Activity	TimeSpent
01/01/2011	18:00	user1	load_page1	3s
01/01/2011	18:01	user1	load_page2	5s
01/01/2011	18:01	user2	load_page1	2s
01/01/2011	18:01	user3	load_page1	3s
01/01/2011	18:04	user4	load_page3	10s
01/01/2011	18:05	user1	load_page3	5s
01/01/2011	18:05	user3	load_page5	3s
01/01/2011	18:06	user4	load_page4	6s
01/01/2011	18:06	user1	purchase	5s
01/01/2011	18:10	user4	purchase	8s
01/01/2011	18:10	user1	confirm	9s
01/01/2011	18:10	user4	confirm	11s
01/01/2011	18:11	user1	load_page3	3s

Host1

```
"1" :  "01/01/2011  18:00  user1  load_page1  3s"
"2" :  "01/01/2011  18:01  user1  load_page2  5s"
"3" :  "01/01/2011  18:01  user2  load_page1  2s"
```

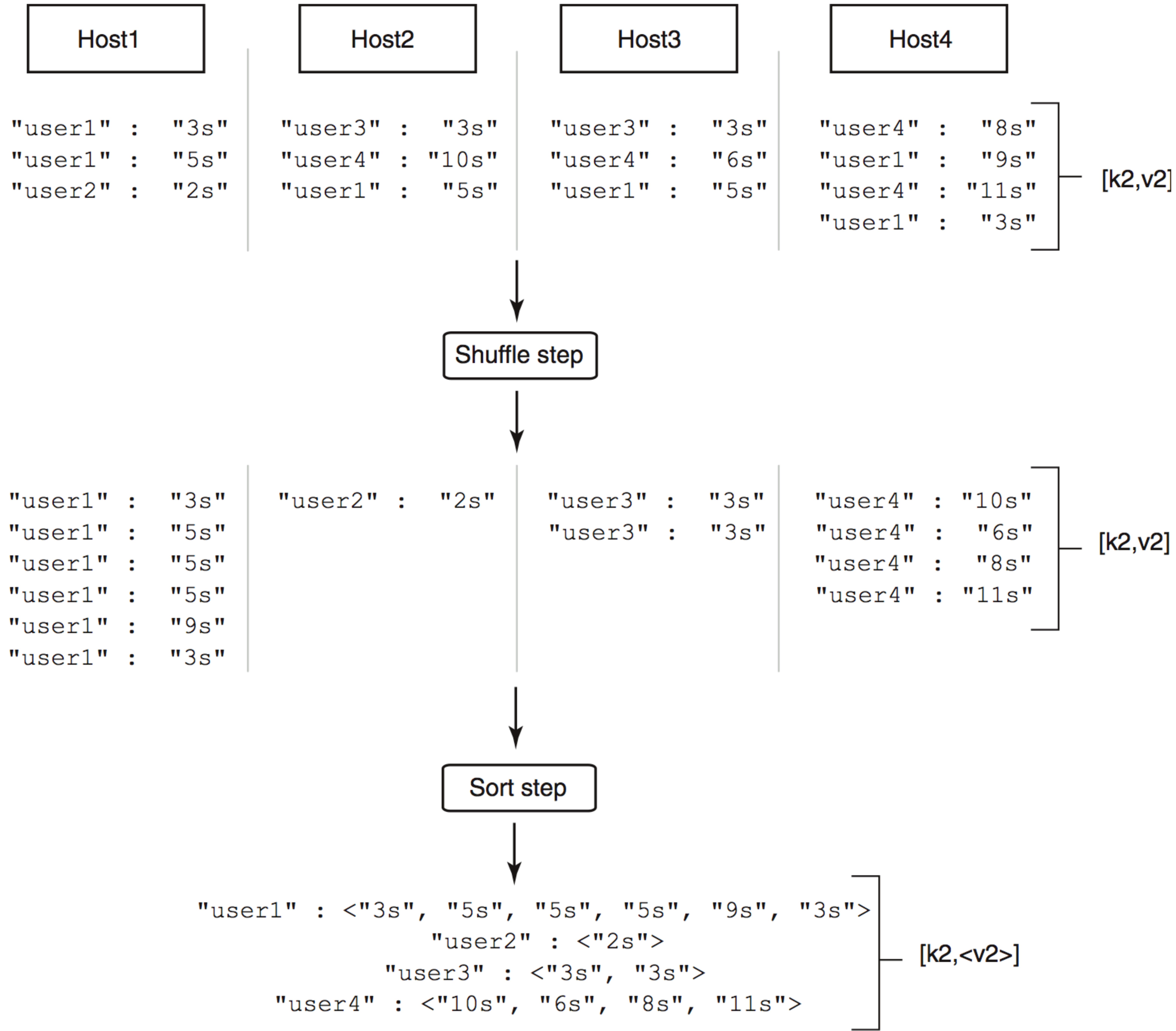
[k1,v1]

Do work

```
"user1" : "3s"
"user1" : "5s"
"user2" : "2s"
```

[k2,v2]

SURF SARA



Host1
Host2
Host3
Host4
"user1" : "3s"
"user1" : "5s"
"user2" : "2s"
"user3" : "3s"
"user4" : "10s"
"user1" : "5s"
"user3" : "3s"
"user4" : "6s"
"user1" : "5s"
"user4" : "8s"
"user1" : "9s"
"user4" : "11s"
"user1" : "3s"
[k2,v2]
Shuffle step
"user1" : "3s"
"user1" : "5s"
"user1" : "5s"
"user1" : "5s"
"user1" : "9s"
"user1" : "3s"
"user2" : "2s"
"user3" : "3s"
"user3" : "3s"
"user4" : "10s"
"user4" : "6s"
"user4" : "8s"
"user4" : "11s"
[k2,v2]
Sort step
"user1" : <"3s", "5s", "5s", "5s", "9s", "3s">
"user2" : <"2s">
"user3" : <"3s", "3s">
"user4" : <"10s", "6s", "8s", "11s">
[k2,<v2>]

Host3

```
"user1" : <"3s", "5s", "5s", "5s", "9s", "3s">
"user2" : <"2s">
"user3" : <"3s", "3s">
"user4" : <"10s", "6s", "8s", "11s">
```

[k2,v2]

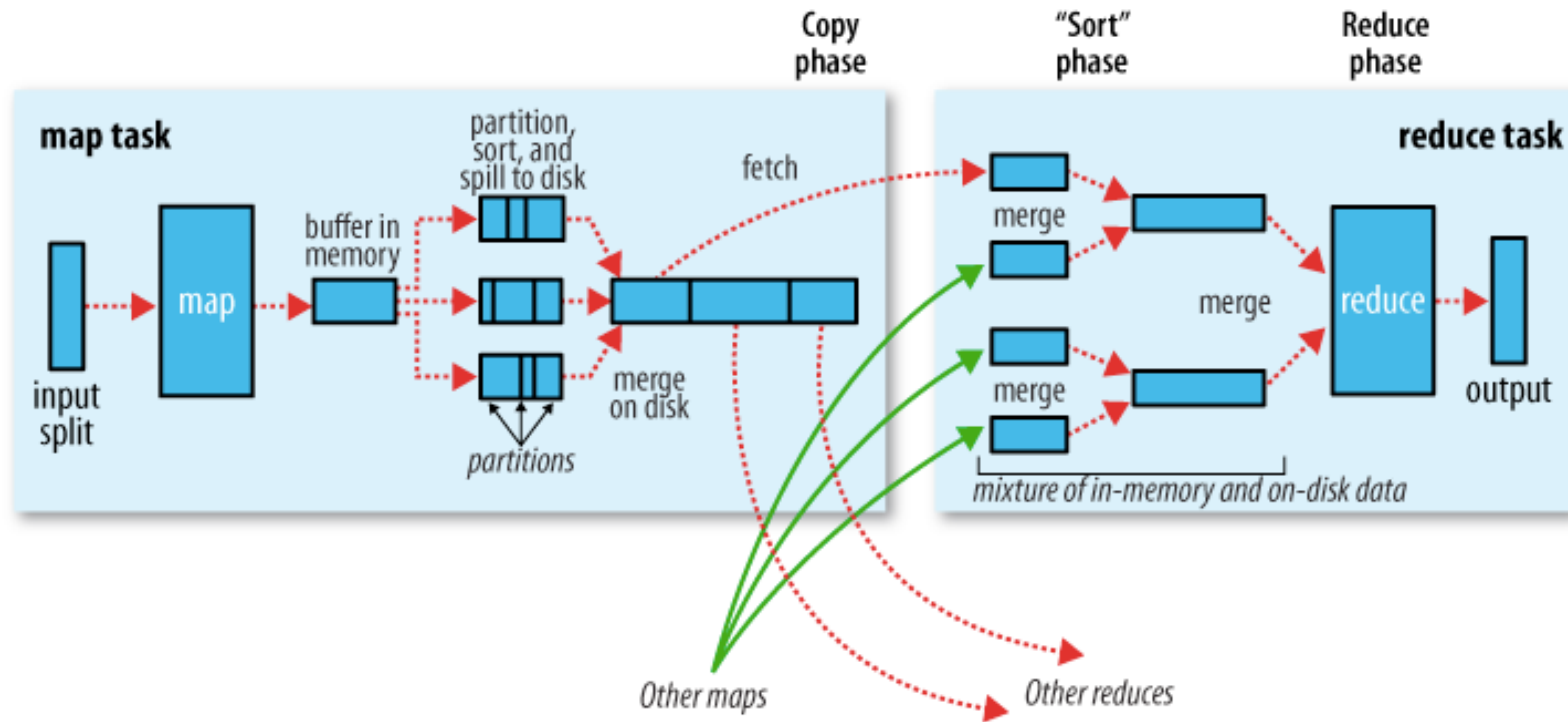
Aggregate work

```
"user1"  :  "30s"
"user2"  :  "2s"
"user3"  :  "6s"
"user4"  :  "35s"
```

[k3,<v3>]

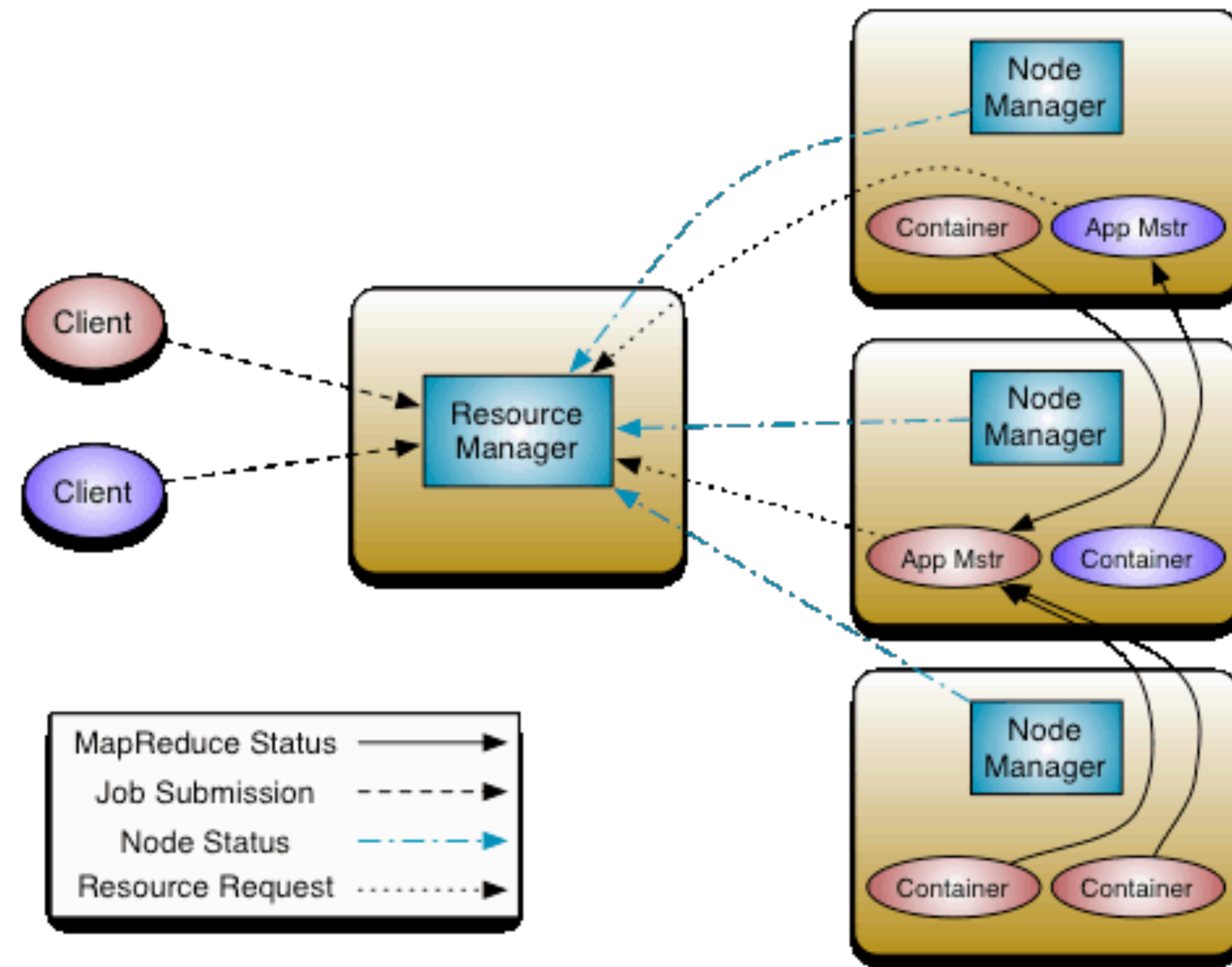
SURF SARA

Shuffle and Sort



SURF SARA

YARN: an Executing Application



SURF SARA

Hadoop at SURFsara

Hathi cluster:

- 197 nodes, 8 cores, 64GB RAM
- 1576 container slots
- 4 x {2,4} TB disks: ~ 2.3PB HDFS
- Hortonworks HDP 2.3 (Hadoop 2.7.1)
- Kerberos authentication
- YARN for {MapReduce, Spark, … }

SURF SARA

Hands-on: Notebooks

Jupyter notebooks:

- Browse to: http://hadws{1..28}.demouva.vm.surfsara.nl:8888
- e.g.: http://hadws1.demouva.vm.surfsara.nl:8888
- Password: spark@uvahpc