CS 61C: Great Ideas in Computer Architecture

Lecture 21: *Warehouse-Scale Computing, MapReduce, and Spark*

Instructor: Sagar Karandikar

sagark@eecs.berkeley.edu

http://inst.eecs.berkeley.edu/~cs61c

Berkeley EECS

ELECTRICAL ENGINEERING & COMPUTER SCIENCES

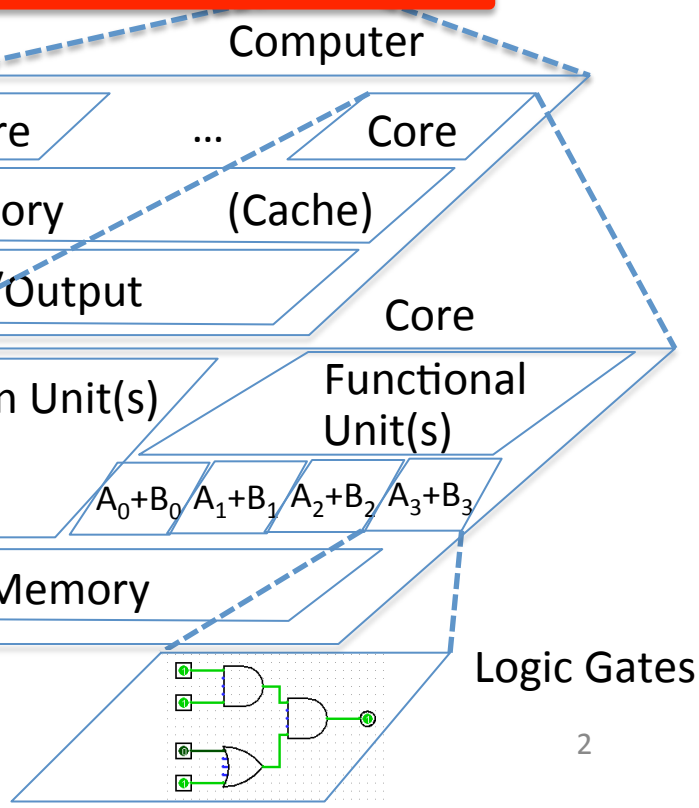
New-School Machine Structures (It's a bit more complicated!)

Software *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Asanovic"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Deep Learning for image classification
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance

Labs 11/13



In the news

- Google disclosed that it continuously uses enough electricity to power 200,000 homes, but it says that in doing so, it also makes the planet greener.
- Search cost per day (per person) same as running a 60-watt bulb for 3 hours

Urs Hoelzle, Google SVP
Co-author of today's reading

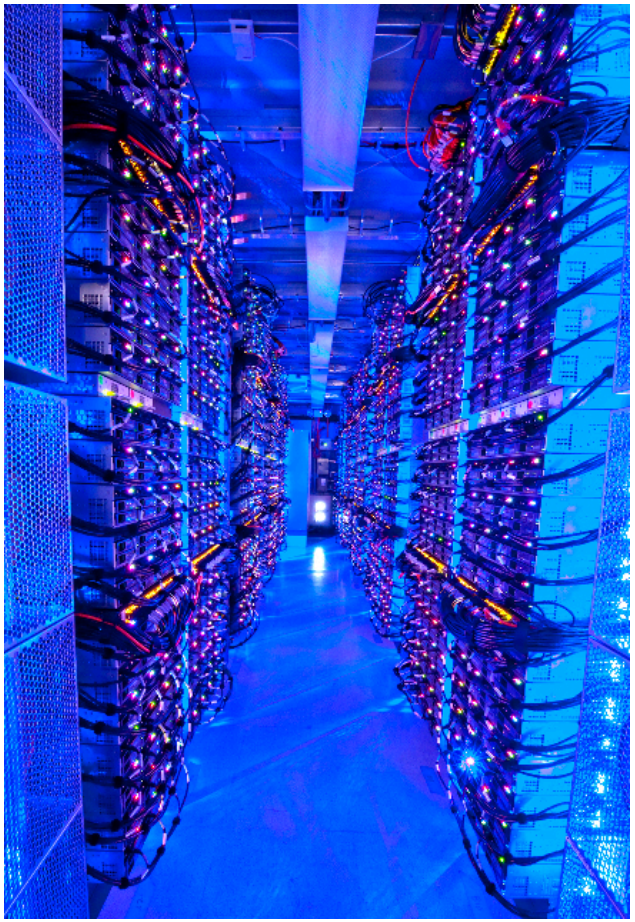
http://www.nytimes.com/2011/09/09/technology/google-details-and-defends-its-use-of-electricity.html

E.g., Google's Oregon WSC

Containers in WSCs

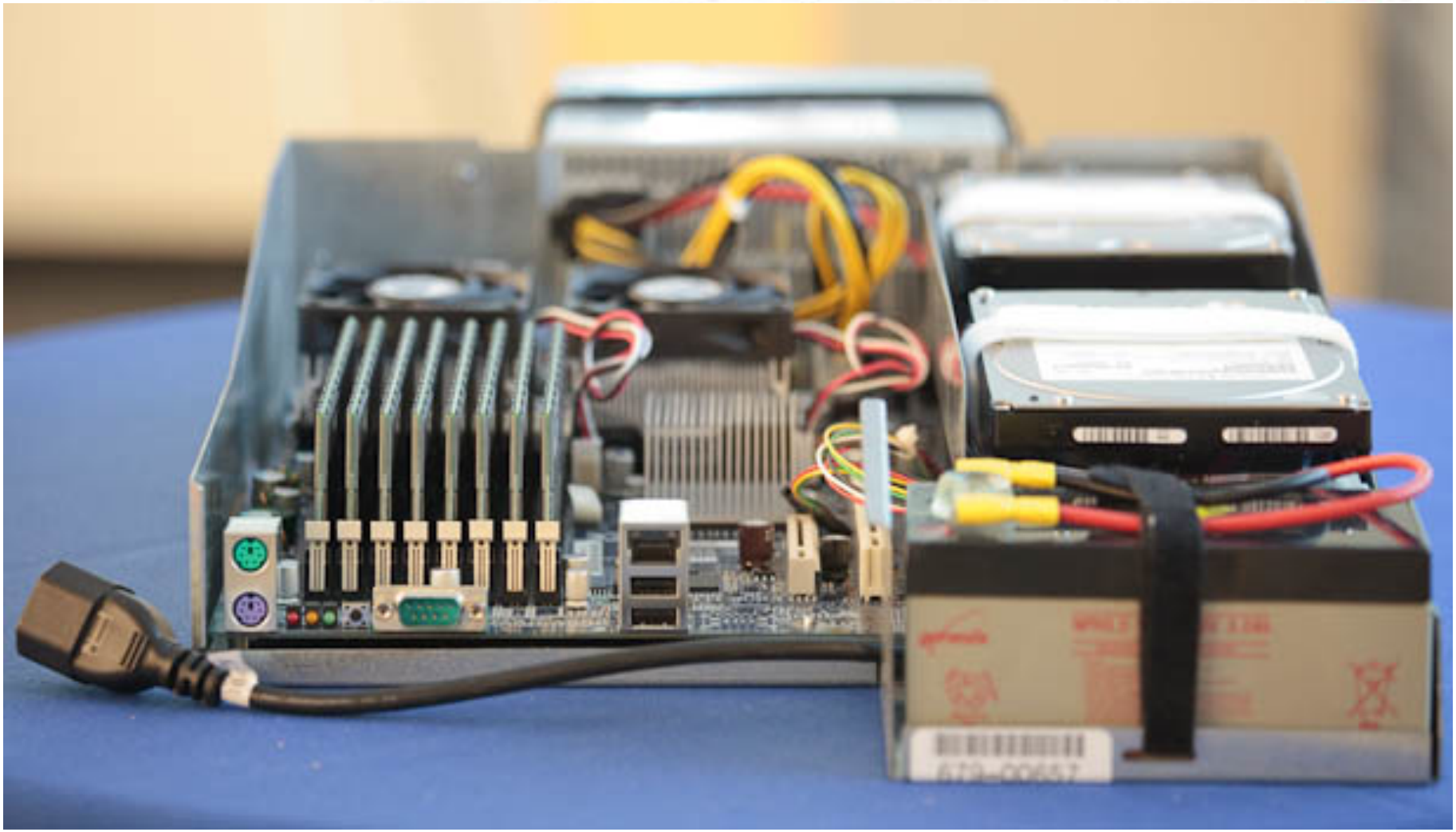
Inside WSC

Inside Container



Server, Rack, Array

Google Server Internals



Warehouse-Scale Computers

- Datacenter
  - Collection of 10,000 to 100,000 severs
  - Networks connecting them together
- ***Single gigantic*** machine
- Very large applications(Internet service): search, email, video sharing, social networking
- Very high availability
- "…WSCs are no less worthy of the expertise of computer systems architects than any other class of machines" Barroso and Hoelzle 2009

Unique to WSCs

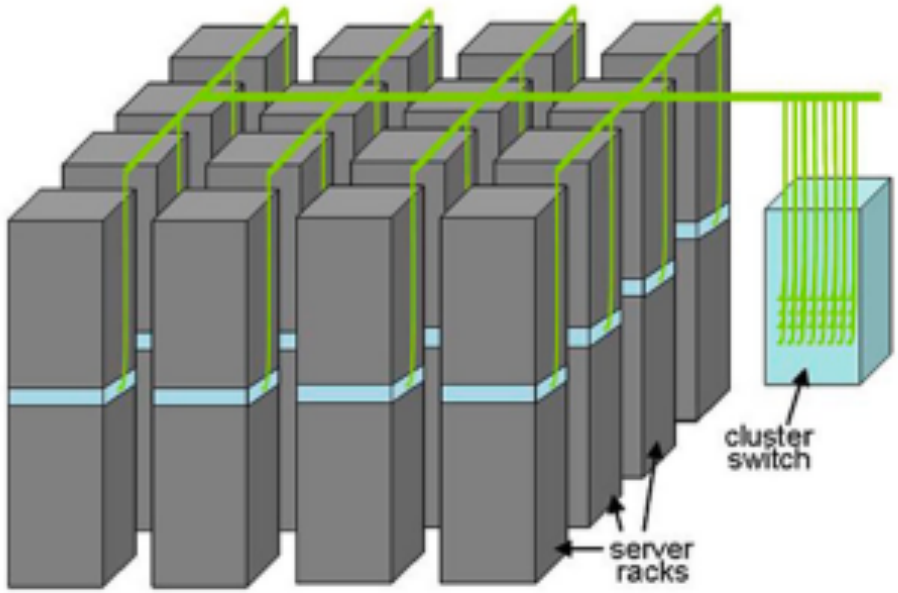
- Ample Parallelism
  - **Request-level Parallelism:** ex) Web search
  - **Data-level Parallelism:** ex) Image classifier training
- Scale and its Opportunities/Problems
  - **Scale of economy**: low per-unit cost
  - Cloud computing: rent computing power with low costs
  - **High # of failures**

    ex) 4 disks/server, annual failure rate: 4%
    → WSC of 50,000 servers: 1 disk fail/hour
- Operation Cost Count
  - Longer life time(>10 years)
  - **Cost of equipment purchases << cost of ownership**

WSC Architecture

1U Server:
8 cores,
16 GB DRAM,
4x1 TB disk

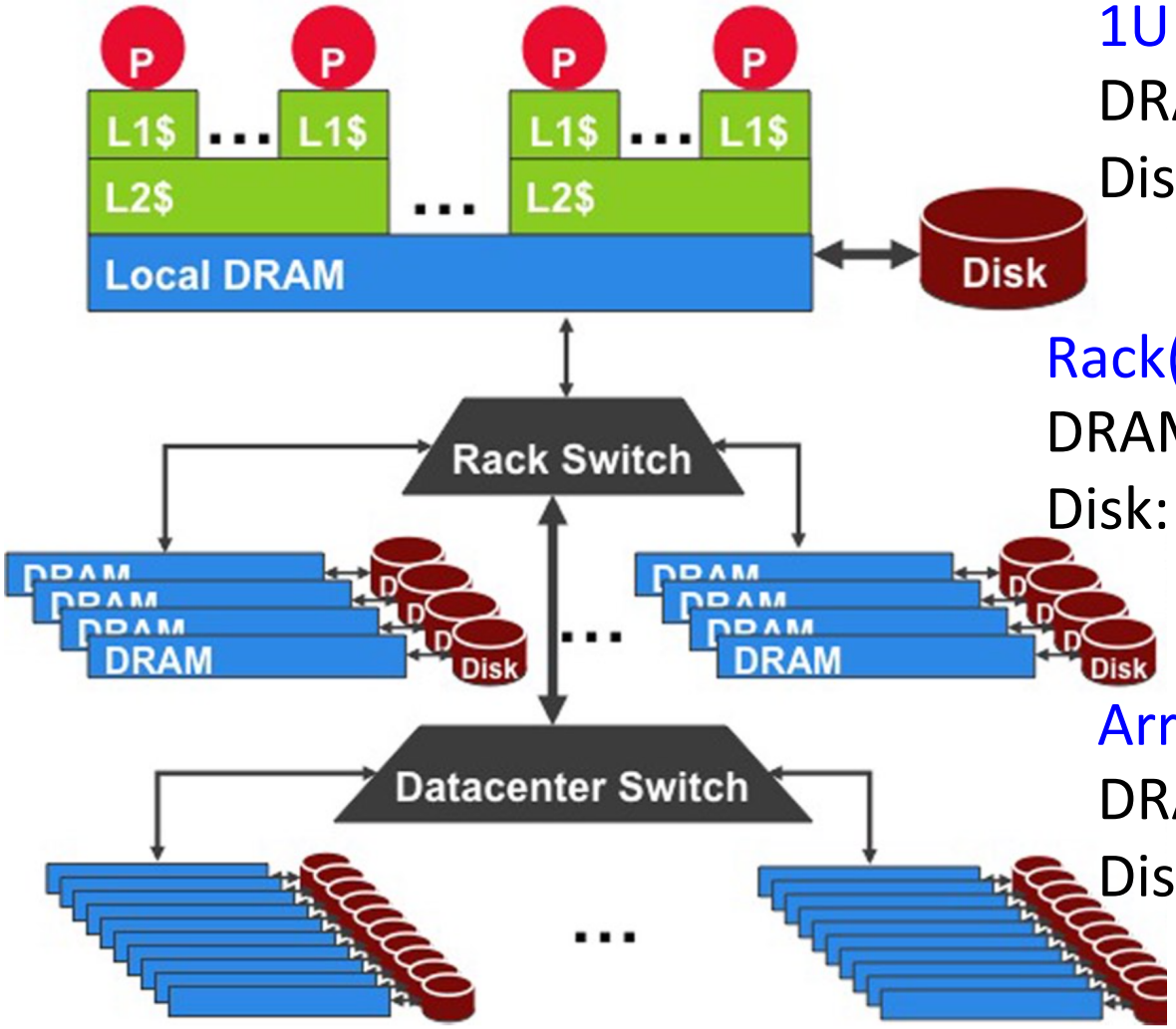
Rack:
40-80 severs,
Local Ethernet(1-10Gbps) switch
(30$/1Gbps/server)



Array(aka cluster):
16-32 racks
Expensive switch
(10X bandwidth → 100x cost)

WSC Storage Hierarchy

Lower latency to DRAM in another server than local disk
Higher bandwidth to local disk than to DRAM in another server

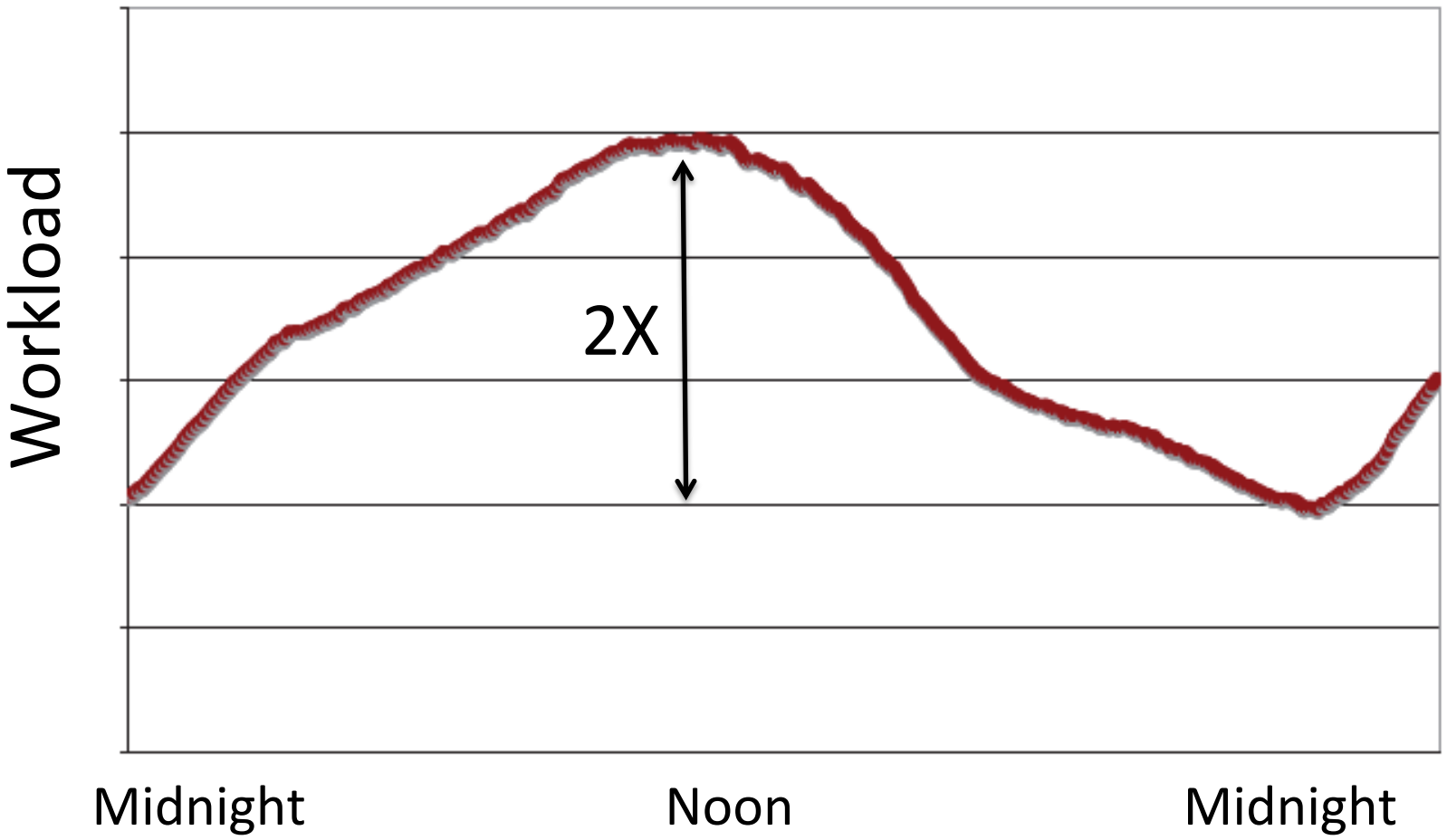


1U Server:
DRAM: 16GB, 100ns, 20GB/s
Disk: 2TB, 10ms, 200MB/s

Rack(80 severs):
DRAM: 1TB, 300us, 100MB/s
Disk: 160TB, 11ms, 100MB/s

Array(30 racks):
DRAM: 30TB, 500us, 10MB/s
Disk: 4.80PB, 12ms, 10MB/s

Workload Variation



- Online service: Peak usage 2X off-peak

Impact on WSC software

- ***Latency, bandwidth*** → Performance
  - Independent data set within an array
  - Locality of access within sever or rack
- ***High failure rate*** → Reliability, Availability
  - Preventing failures is expensive
  - Cope with failures gracefully
- ***Varying workloads*** → Availability
  - Scale up and down gracefully
- More challenging than software for single computers!

Power Usage Effectiveness

- Energy efficiency
  - Primary concern in the design of WSC
  - Important component of the total cost of ownership
- Power Usage Effectiveness (PUE):

$$\frac{\textbf{Total Building Power}}{\textbf{IT equipment Power}}$$

  - A power efficiency measure for WSC
  - Not considering efficiency of servers, networking gears
  - Perfection = 1.0
  - Google WSC's PUE = 1.2

PUE in the Wild (2007)

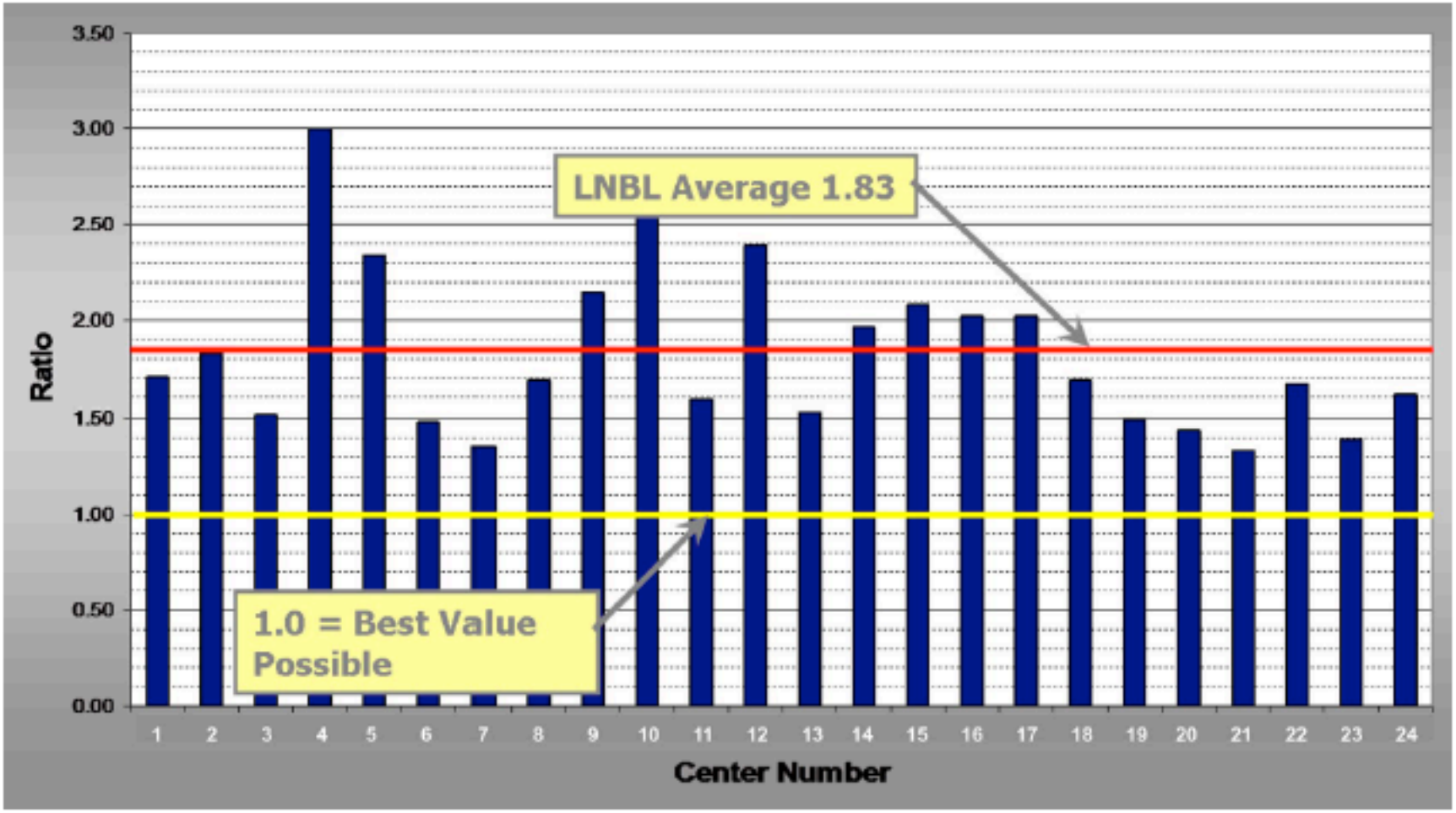


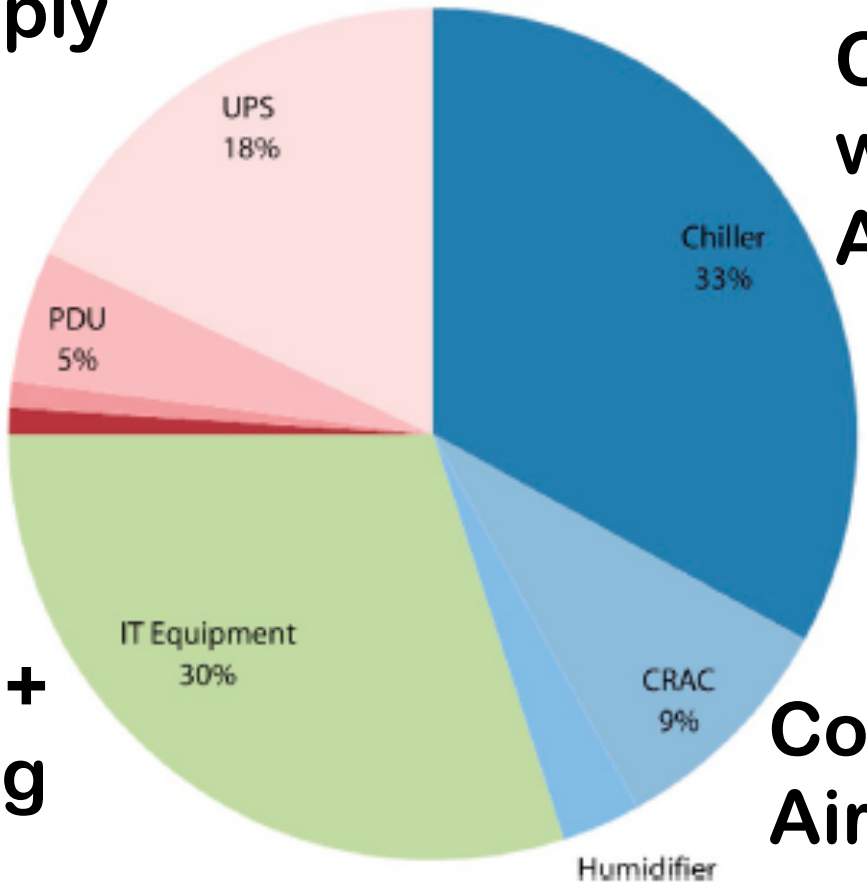
FIGURE 5.1: LBNL survey of the power usage efficiency of 24 datacenters, 2007 (Greenberg et al.)

High PUE: Where Does Power Go?

Uninterruptable Power Supply (battery)

Power Distribution Unit

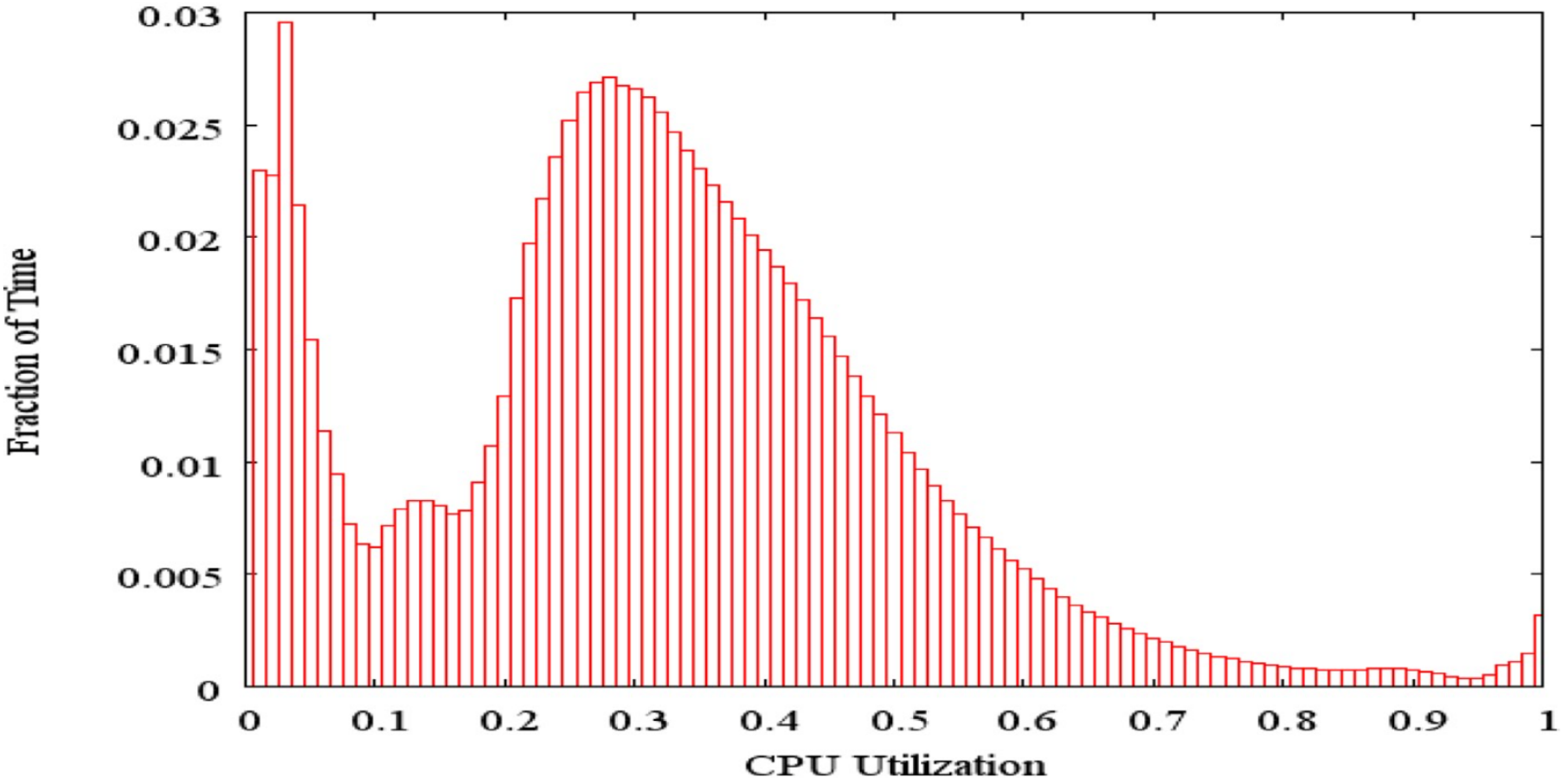
Severs + Networking



Cooling warm water from Air Conditioner

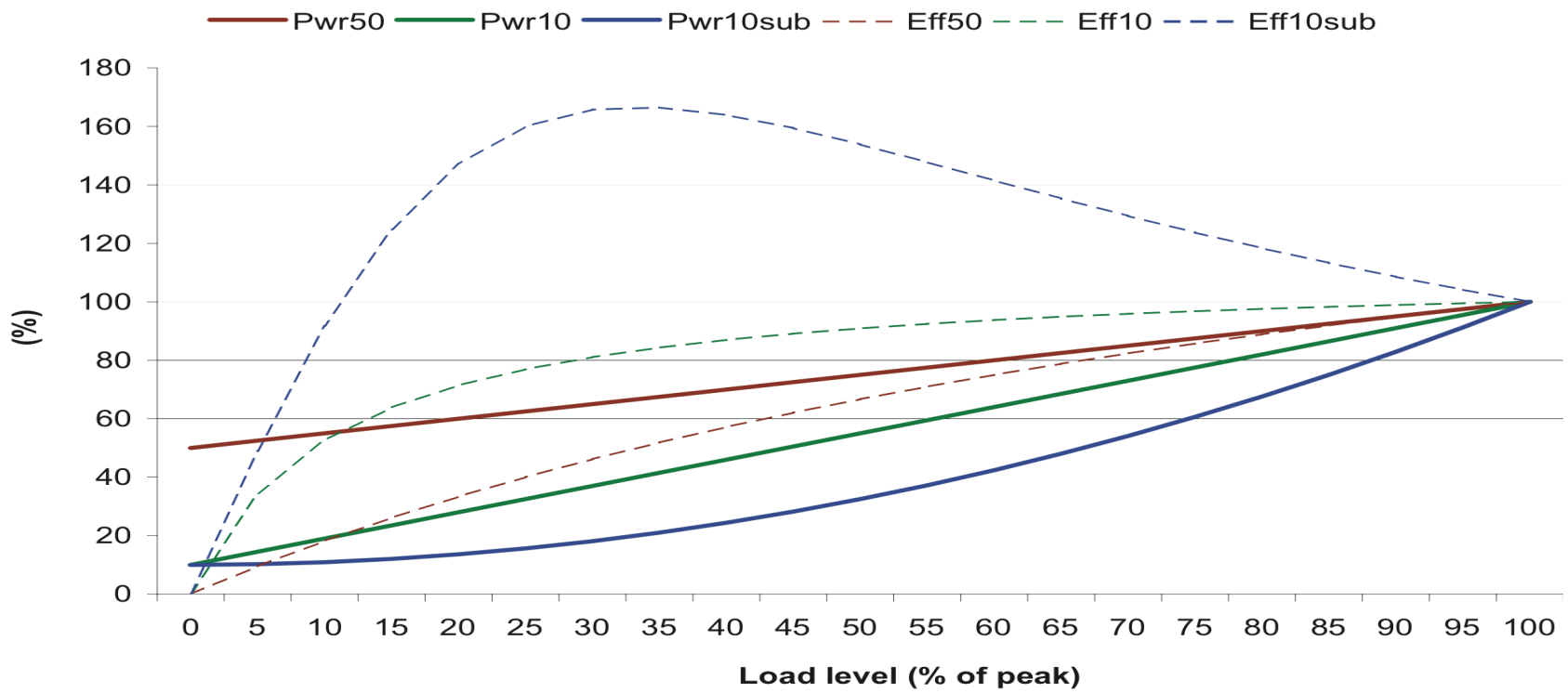
Computer Room Air Conditioner

Load Profile of WSCs



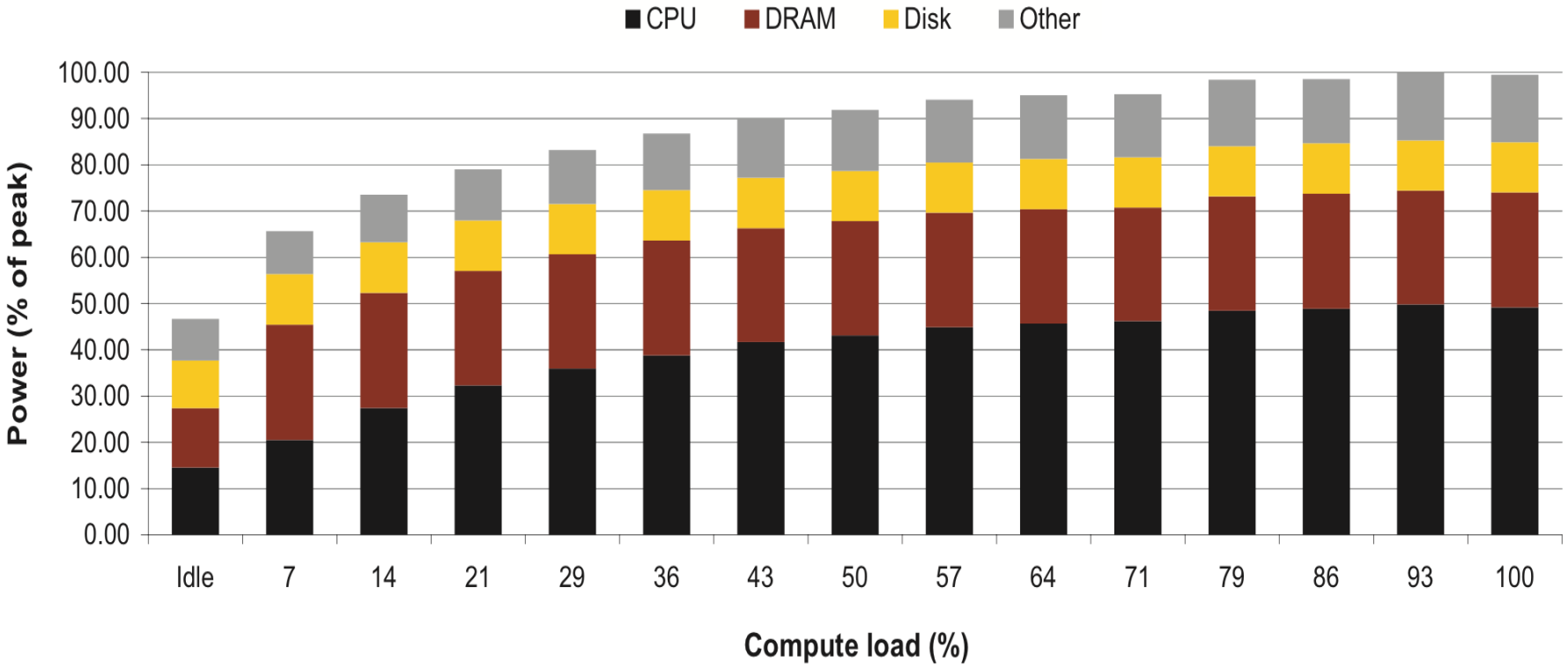
- Average CPU utilization of 5,000 Google severs
- Most servers in WSC idle or utilized ***20% to 60%***

Energy-Proportional Computing :Design Goal of WSC



- Energy = Power x Time, Efficiency = Computation / Energy
  → Efficiency α 1 / Power, 1 / Time
- Consumes almost no power when idle
- Gradually consume more power as the activity level increases

Cause of Poor Energy Proportionality



- CPU: 50% at peek, 30% at idle
- DRAM, disks, networking: 70% at idle!
- Need to improve the energy efficiency of peripherals

Cloud Computing: Scale of Economy

Instance	Per Hour	Ratio to small	Compute Units	Virtual Cores	Compute Unit / Core	Memory (GiB)	Disk (GiB)	Address
Standard Small	$0.065	1.0	1.0	1	1.00	1.7	160	32bit
Standard Large	$0.260	4.0	4.0	2	2.00	7.5	850	64bit
Standard Extra Large	$0.520	8.0	8.0	4	2.00	15.0	1680	64bit
High-Memory Extra Large	$0.460	5.9	6.5	2	3.25	17.1	420	64bit
High-Memory Double Extra Large	$0.920	11.8	13.0	4	3.25	34.2	850	64bit
High-Memory Quadruple Extra Large	$1.840	23.5	26.0	8	3.25	68.4	1680	64bit
High-CPU Medium	$0.165	2.0	5.0	2	2.50	1.7	350	32bit
High-CPU Extra Large	$0.660	8.0	20.0	8	2.50	7.0	1680	64bit

- March 2014 AWS Instances & Prices
- Closest computer in WSC example is Standard Extra
- At these low rates, Amazon EC2 can make money!
  - even if used only 50% of time
- Virtual Machine(VM) plays an important role

Agenda

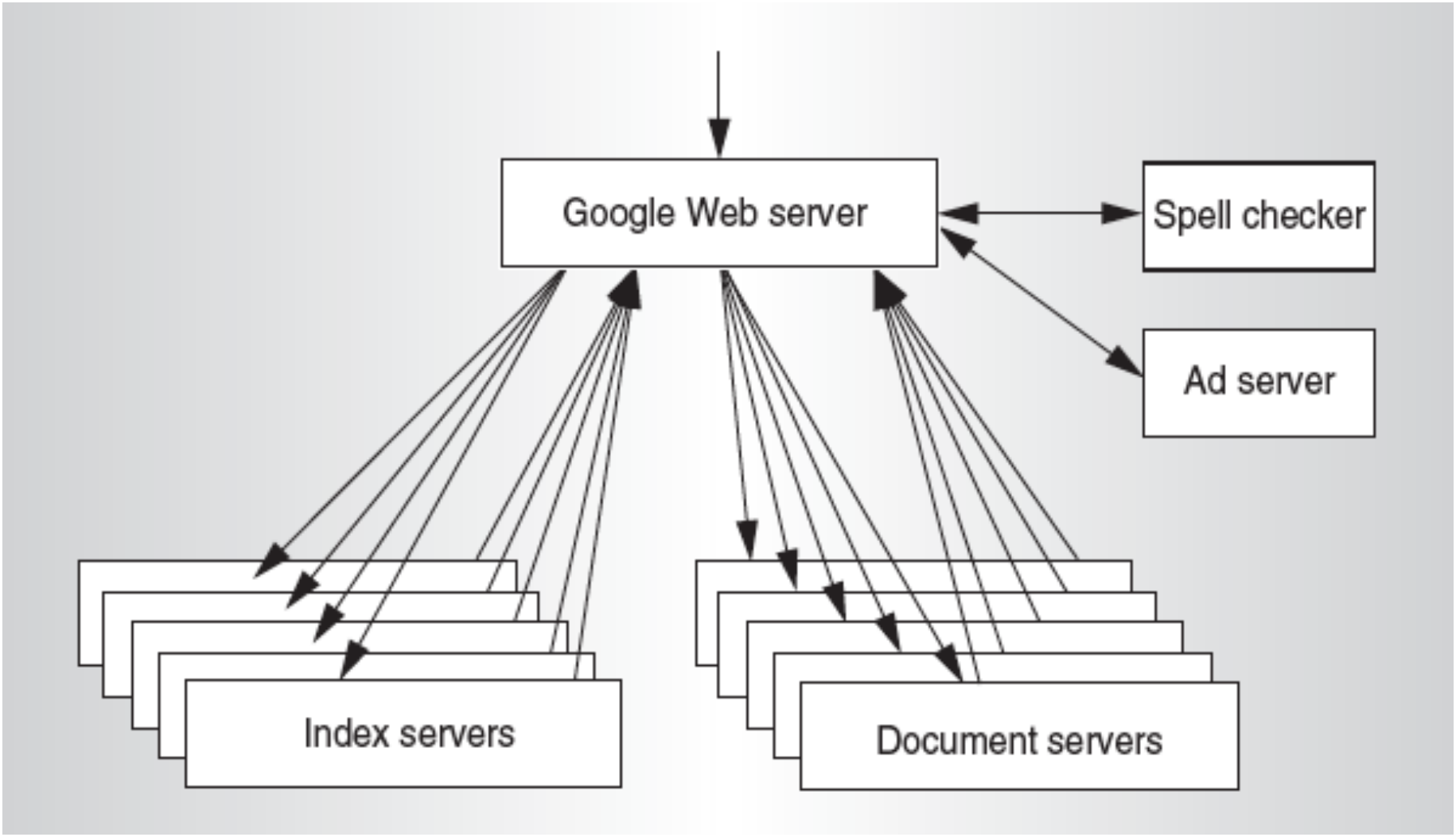
- Warehouse Scale Computing
- Request-level Parallelism

  e.g. Web search
- Administrivia & Clickers/Peer Instructions
- Data-level Parallelism
  - Hadoop, Spark
  - MapReduce

Request-Level Parallelism (RLP)

- Hundreds of thousands of requests per sec
  - Not your laptop or cell-phone, but popular Internet services like web search, social networking, …
  - Such requests are largely independent
    - Often involve read-mostly databases
    - Rarely involve read-write sharing or synchronization across requests
- Computation easily partitioned across different requests and even within a request

Google Query-Serving Architecture



Anatomy of a Web Search

- Google “Krste Asanovic”

krste asanovic

Web Videos Images Shopping News More Search tools

About 25,500 results (0.67 seconds)

Home Page for Krste Asanović
www.eecs.berkeley.edu/~krste/ University of California, Berkeley
I am a Professor in the Computer Science Division of the EECS Department at the University of California, Berkeley. My main research areas are computer ...

Publications
Krste Asanović, Publications by Year ... Year Anniversary ...

If you're a student interested …
Krste Asanović: Information for Prospective Students. If you're a ...

More results from berkeley.edu »

Krste Asanović | EECS at UC Berkeley
www.eecs.berkeley.edu/.../asanovic.ht... University of California, Berkeley
M. Maas, P. Reames, J. Morlan, K. **Asanović**, A. D. Joseph, and J. D. Kubiatowicz, "GPUs as an Opportunity for Offloading Garbage Collection," in Proceedings of ...

Krste Asanovic | ICSI
https://www.icsi.berkeley.edu/icsi/people/krste
Senior Researcher, Research Initiatives krste @ icsi.berkeley.edu. **Krste Asanović** received his bachelor's degree in electrical and information sciences from the ...

FireBox: A Hardware Building Block for 2020 Warehouse ...
https://www.usenix.org/conference/fast14/technical.../keynote USENIX
Krste Asanović, University of California, Berkeley ... **Krste Asanović** received a B.A. in Electrical and Information Sciences from Cambridge University in 1987 and ...

"The First 20 Years, the First 20 Chips," Krste Asanovic ...

www.youtube.com/watch?v=UgnphJf7qF0
Apr 12, 2012 - Uploaded by ICSIatBerkeley
Professor **Krste Asanovic**, leader of ICSI's Architecture Group, speaks about ICSI's work in computer ...

Stanford Seminar - Krste Asanović of UC Berkeley - YouTube

www.youtube.com/watch?v=vB0FC1DZqUM
Oct 16, 2014 - Uploaded by stanfordonline
"Instructions Sets Should Be Free: The Case for RISC-V" - **Krste Asanović** of UC Berkeley Colloquium on ...

Anatomy of a Web Search (1/3)

- Google “Krste Asanovic”
  - Direct request to “closest” Google WSC
  - Front-end load balancer directs request to one of many arrays (cluster of severs) within WSC
  - Within array, select one of many Goggle Web Servers(GWS) to handle the request and compose the response pages
  - GWS communicates with Index Servers to find documents that contains the search words, “Krste”, “Asanovic”
  - Return document list with associated relevance score

Anatomy of a Web Search (2/3)

- In parallel,
  - Ad system: run ad auction for bidders on search terms
  - Get videos of Krste Asanovic's talks
- Use docids (Document IDs) to access indexed documents
- Compose the page
  - Result document extracts (with keyword in context) ordered by relevance score
  - Sponsored links (along the top) and advertisements (along the sides)

Anatomy of a Web Search (3/3)

- Implementation strategy
  - Randomly distribute the entries
  - Make many copies of data(a.k.a. “replicas”)
  - Load balance requests across replicas
- ***Redundant copies*** of indices and documents
  - Breaks up search hot spots, e.g. “Taylor Swift”
  - Increases opportunities for ***request-level parallelism***
  - Makes the system more ***tolerant of failures***

Agenda

- Warehouse Scale Computing
- Request-level Parallelism
  - e.g. Web search
- Administrivia & Clickers/Peer Instruction
- Data-level Parallelism
  - Hadoop, Spark
  - MapReduce

Administrivia

- Project 3-2 Out
- HW5 Out – Performance Programming
- Guerrilla Section on Performance Programming on Thursday, 5-7pm, Woz

Clicker/Peer Instruction: Which Statement is True

- **A: Idle servers consume almost no power**
- **B: Disks will fail once in 20 years, so failure is not a problem of WSC**
- **C: The search requests of the same keyword from different users are dependent**
- **D: WSCs are not good at batch processing**
- **E: WSCs contain many copies of data**

Break

Agenda

- Warehouse Scale Computing
- Administrivia & Clickers/Peer Instructions
- Request-level Parallelism
  - e.g. Web search
- Data-level Parallelism
  - MapReduce
  - Hadoop, Spark

Data-Level Parallelism(DLP)

- SIMD
  - Supports data-level parallelism in a single machine
  - Additional instructions & hardware

  e.g. Matrix multiplication in memory

- DLP on WSC
  - Supports data-level parallelism across multiple machines
  - MapReduce & scalable file systems

  e.g. Training CNNs with images across multiple disks

What is MapReduce?

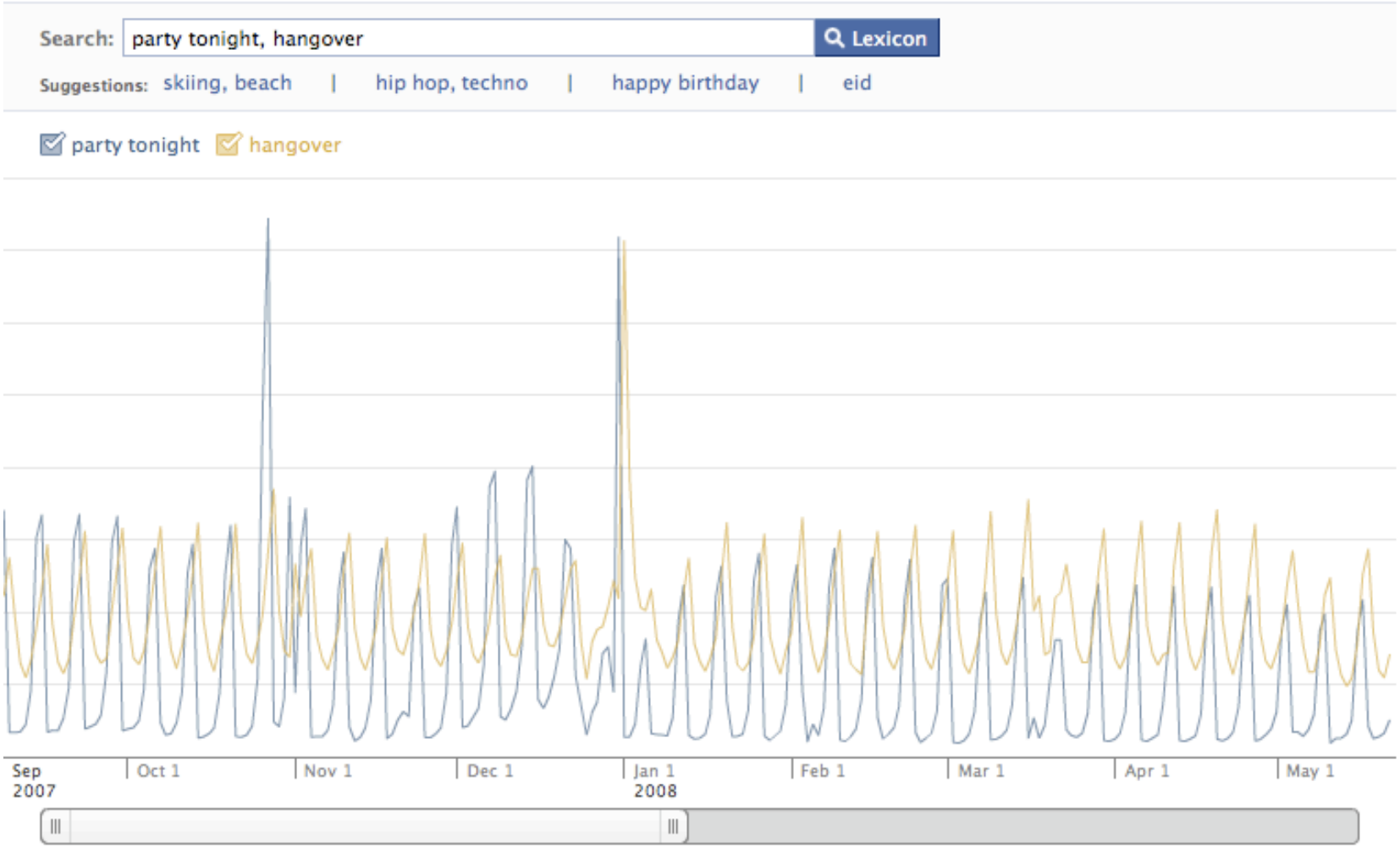
- Simple data-parallel ***programming model*** and ***implementation*** for processing large dataset
- Users specify the computation in terms of
  - a ***map*** function, and
  - a ***reduce*** function
- Underlying runtime system
  - Automatically ***parallelize*** the computation across large scale clusters of machines.
  - ***Handles*** machine ***failure***
  - ***Schedule*** inter-machine communication to make efficient use of the networks

Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *6th USENIX Symposium on Operating Systems Design and Implementation, 2004*.
(optional reading, linked on course homepage – a digestible CS paper at the 61C level)

What is MapReduce used for?

- At Google:
  - Index construction for Google Search
  - Article clustering for Google News
  - Statistical machine translation
  - For computing multi-layers street maps
- At Yahoo!:
  - "Web map" powering Yahoo! Search
  - Spam detection for Yahoo! Mail
- At Facebook:
  - Data mining
  - Ad optimization
  - Spam detection

Example: Facebook Lexicon

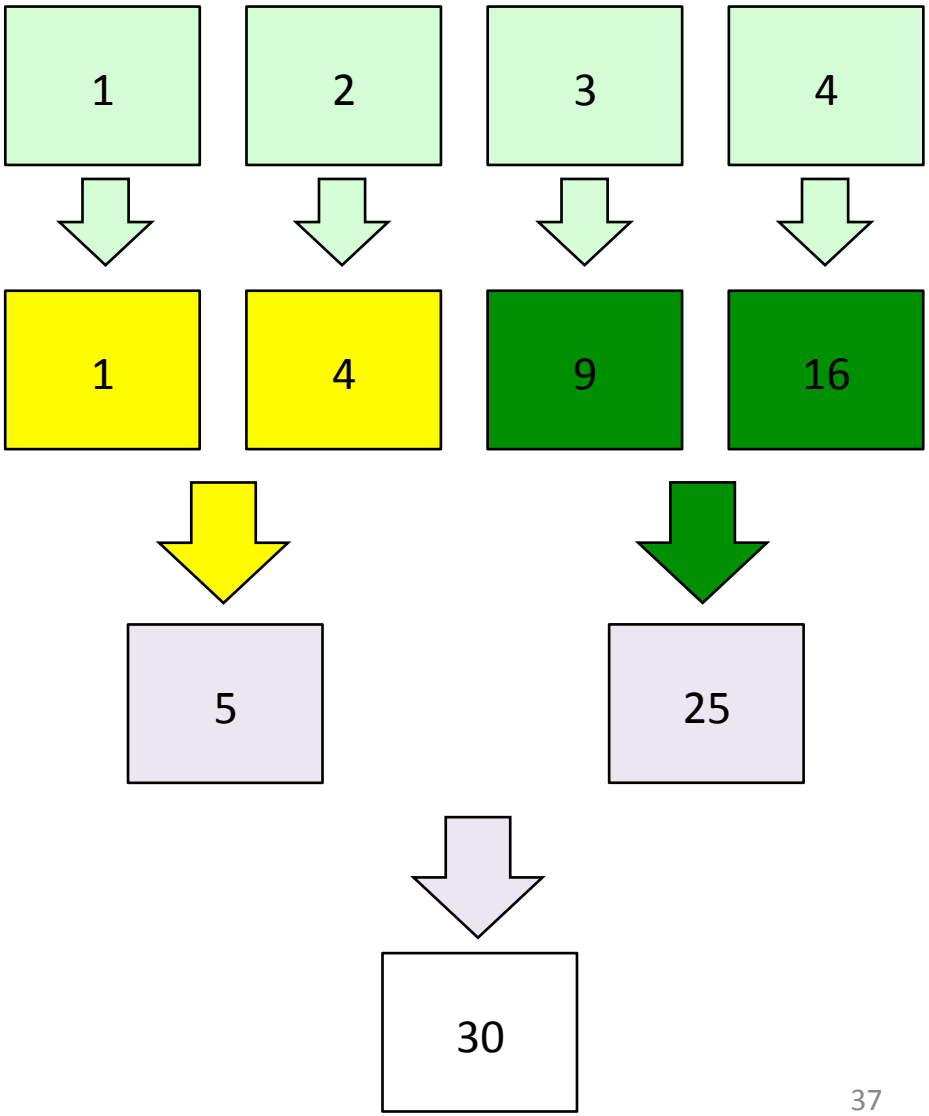


www.facebook.com/lexicon(no longer available)

Map & Reduce Functions in Python

- Calculate : $\sum_{n=1}^{4} n^2$

```
l = [1, 2, 3, 4]
def square(x):
    return x * x
def sum(x, y):
    return x + y
reduce(sum,
    map(square, l))
```



MapReduce Programming Model

- ***Map***: `(in_key, in_value) → list(interm_key, interm_val)`

```
map(in_key, in_val):
  // DO WORK HERE
  emit(interm_key,interm_val)
```

  - Slice data into "shards" or "splits" and distribute to workers
  - Compute set of intermediate key/value pairs

- ***Reduce***: `(interm_key, list(interm_value)) → list(out_value)`

```
reduce(interm_key, list(interm_val)):
  // DO WORK HERE
  emit(out_key, out_val)
```

  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)

MapReduce Word Count Example

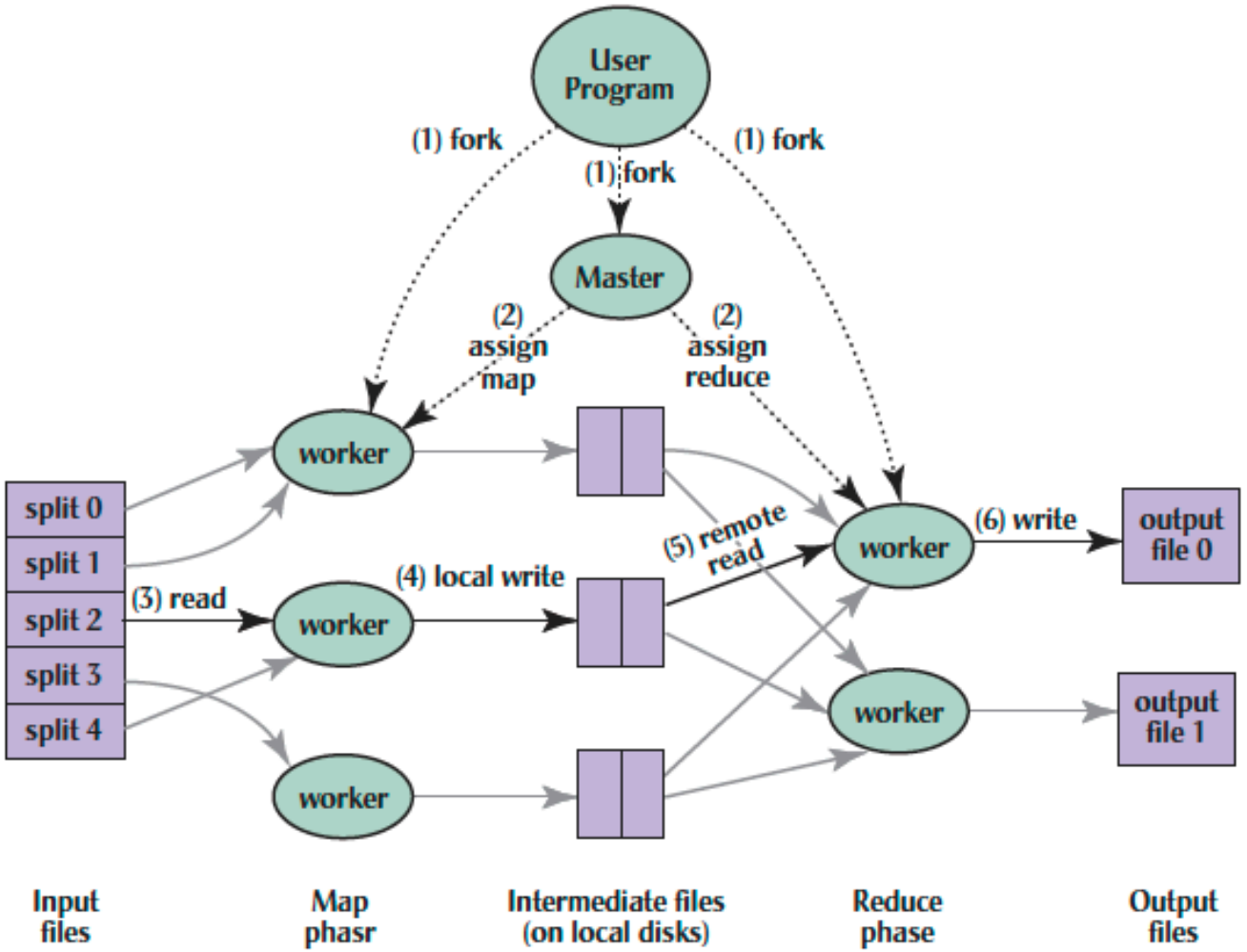
- ***Map*** phase: (doc name, doc contents) → list(word, count)

```
// "I do I learn"" → [("I",1),("do",1),("I",1),("learn",1)]
map(key, value):
  for each word w in value:
    emit(w, 1)
```

- ***Reduce*** phase: (word, list(count)) → (word, count_sum)

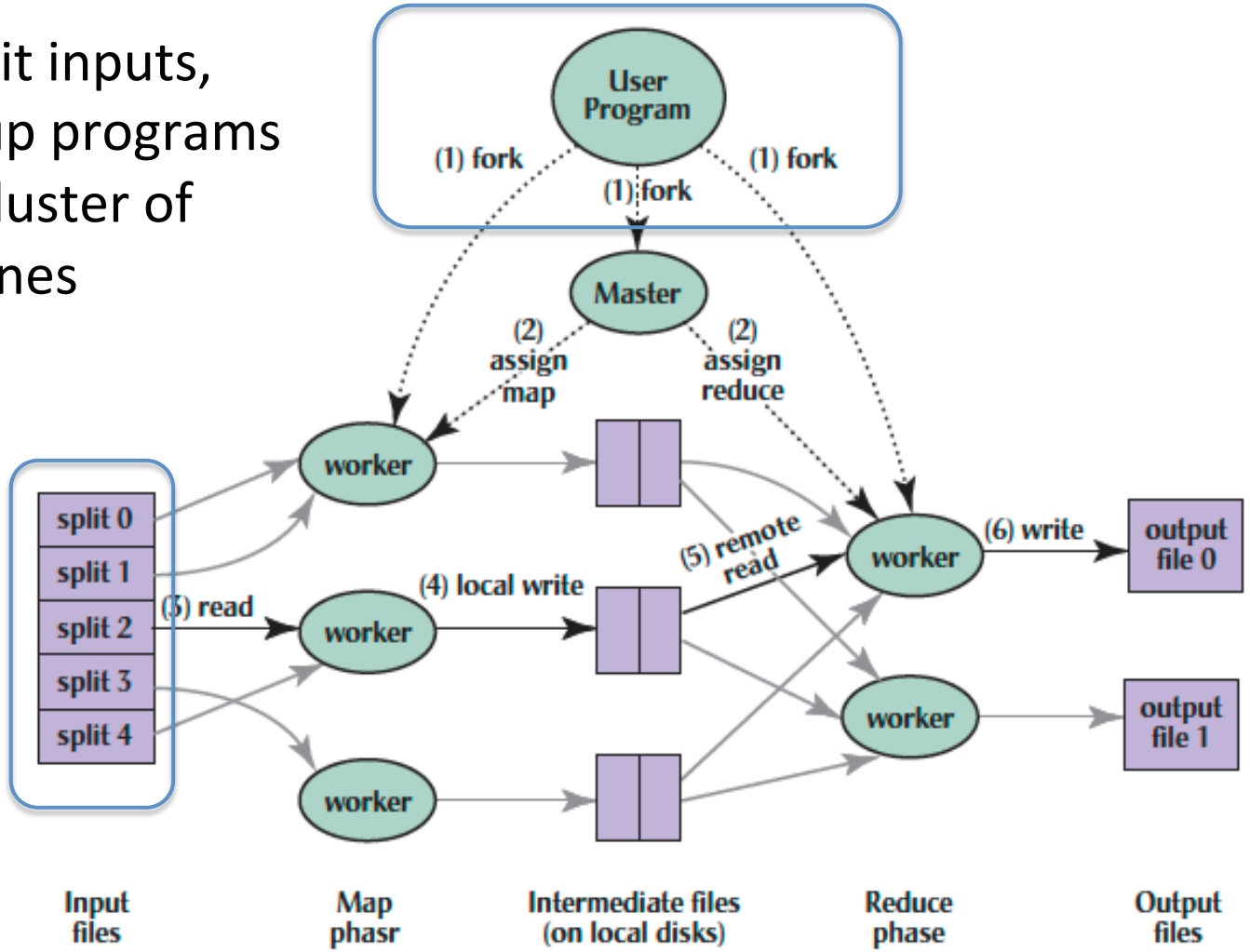
```
// ("I", [1,1]) → ("I",2)
reduce(key, values):
  result = 0
  for each v in values:
    result += v
  emit(key, result)
```

MapReduce Implementation



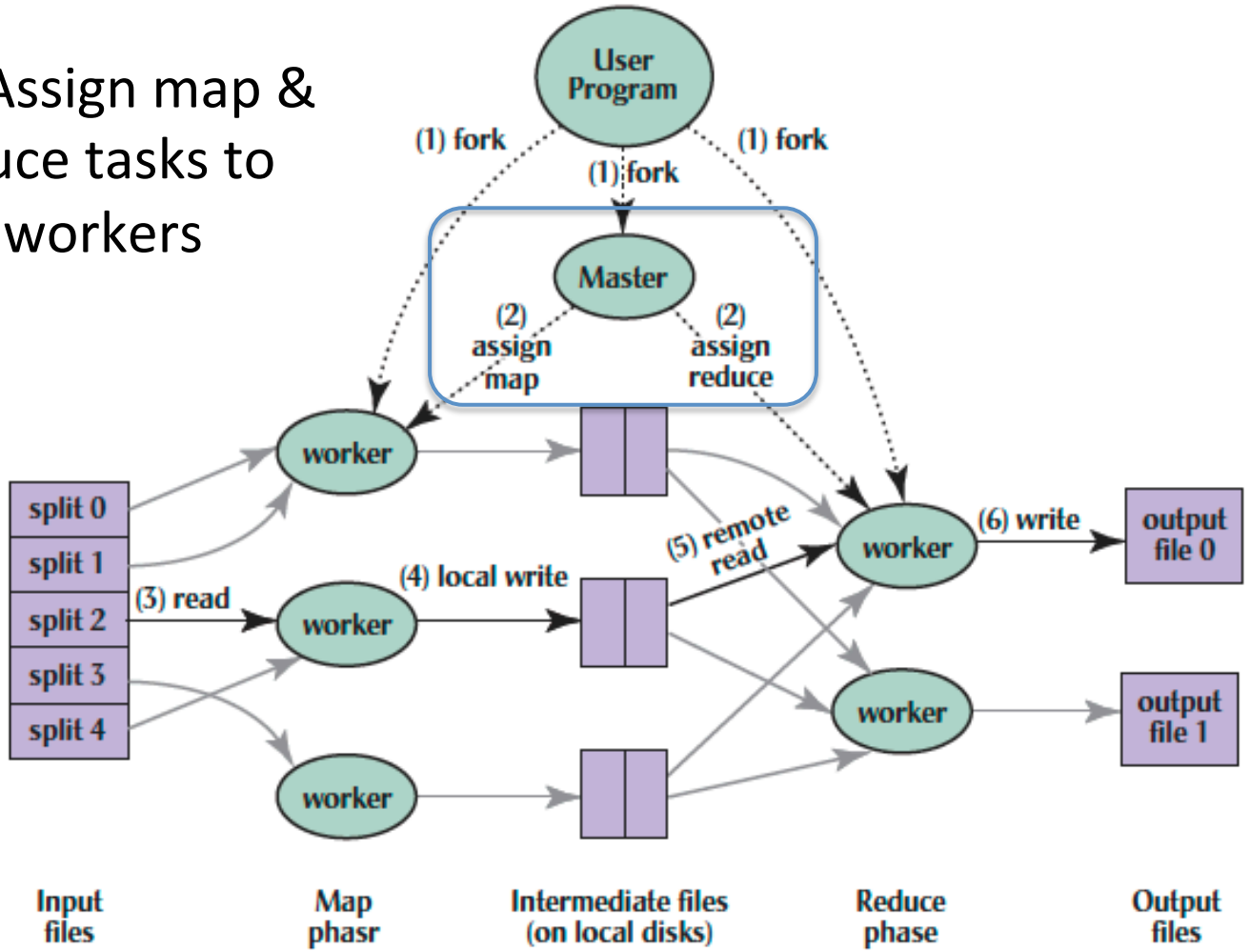
MapReduce Execution

(1) Split inputs, start up programs on a cluster of machines



MapReduce Execution

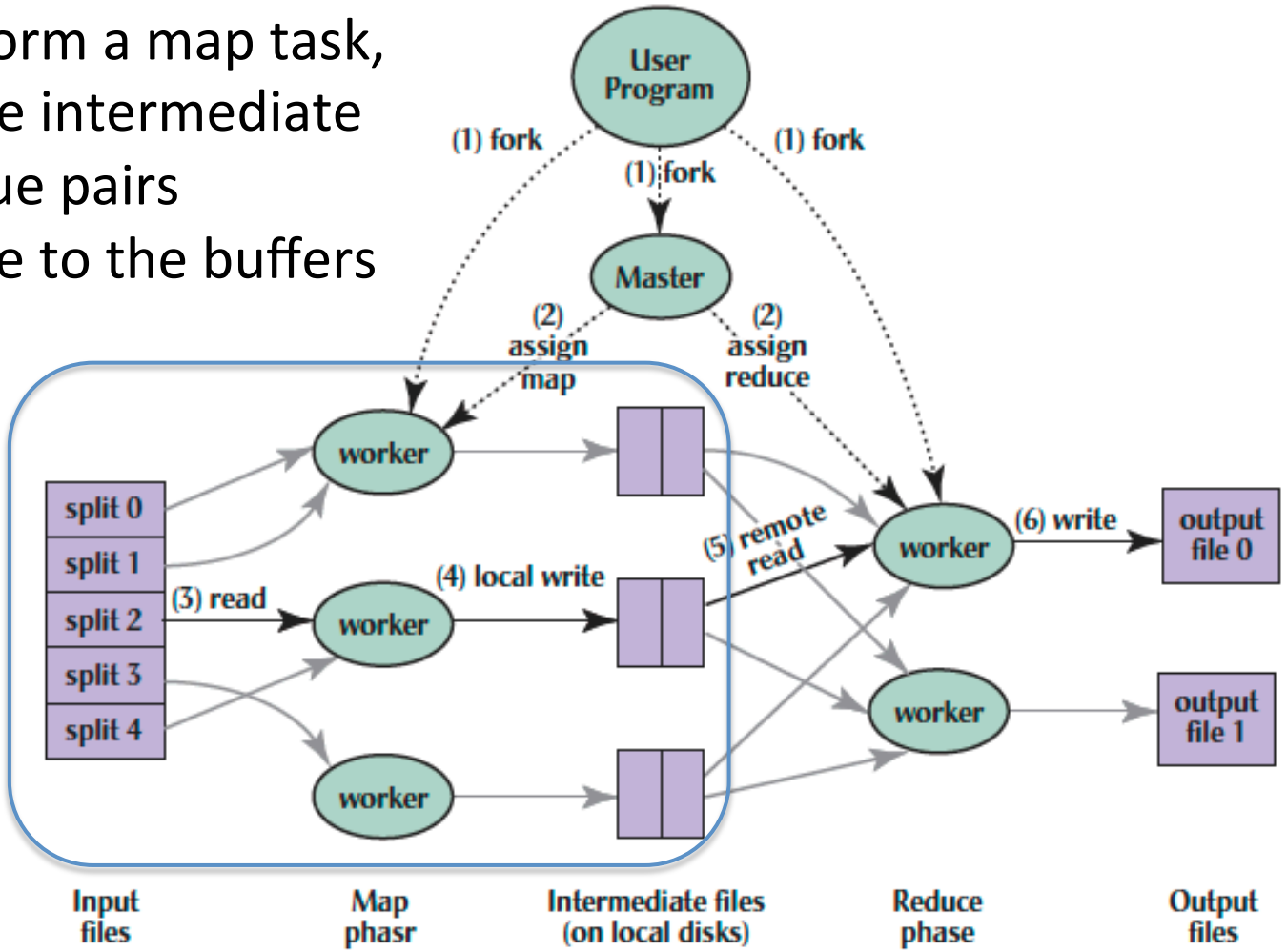
(2) Assign map & reduce tasks to idle workers



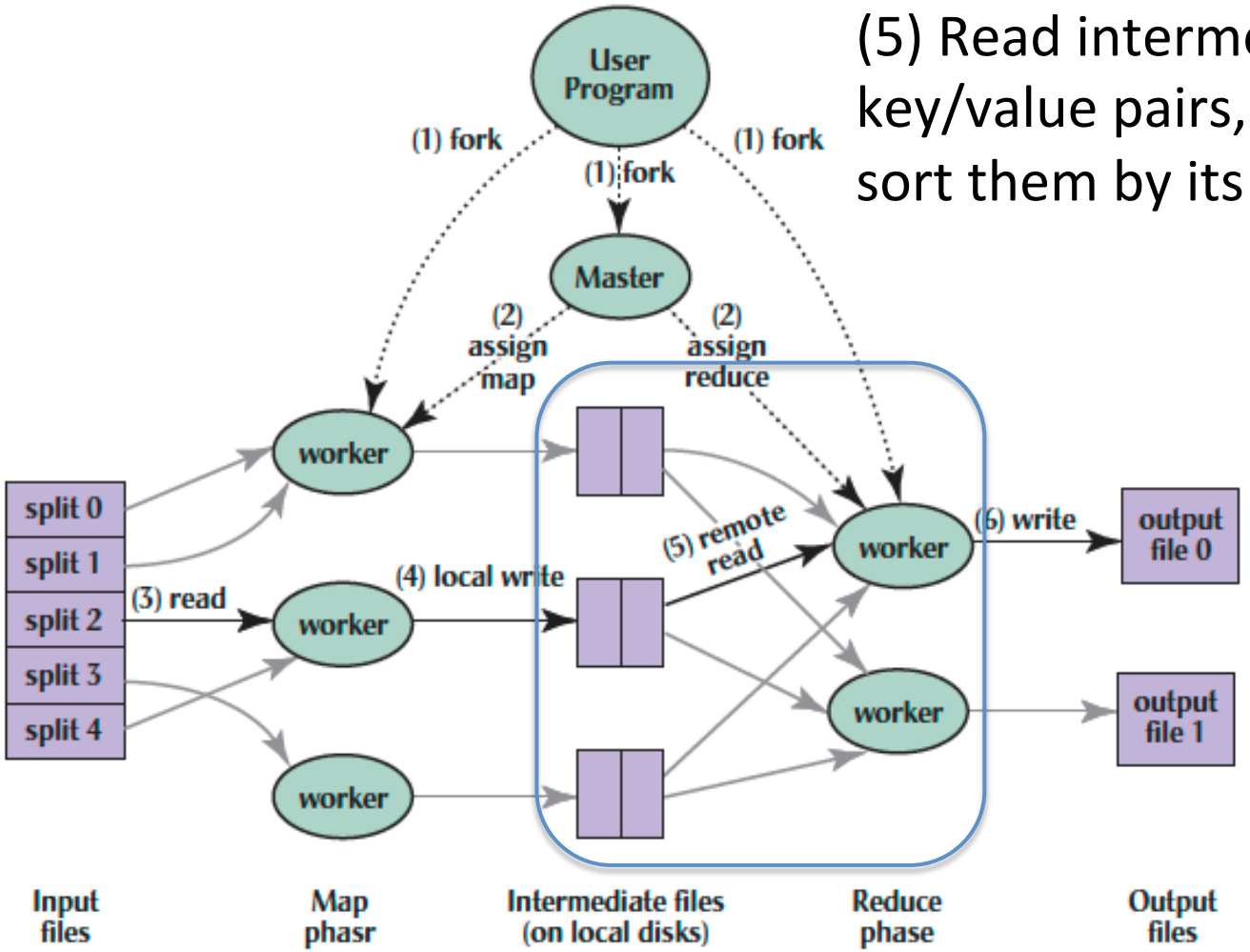
MapReduce Execution

(3) Perform a map task, generate intermediate key/value pairs

(4) Write to the buffers

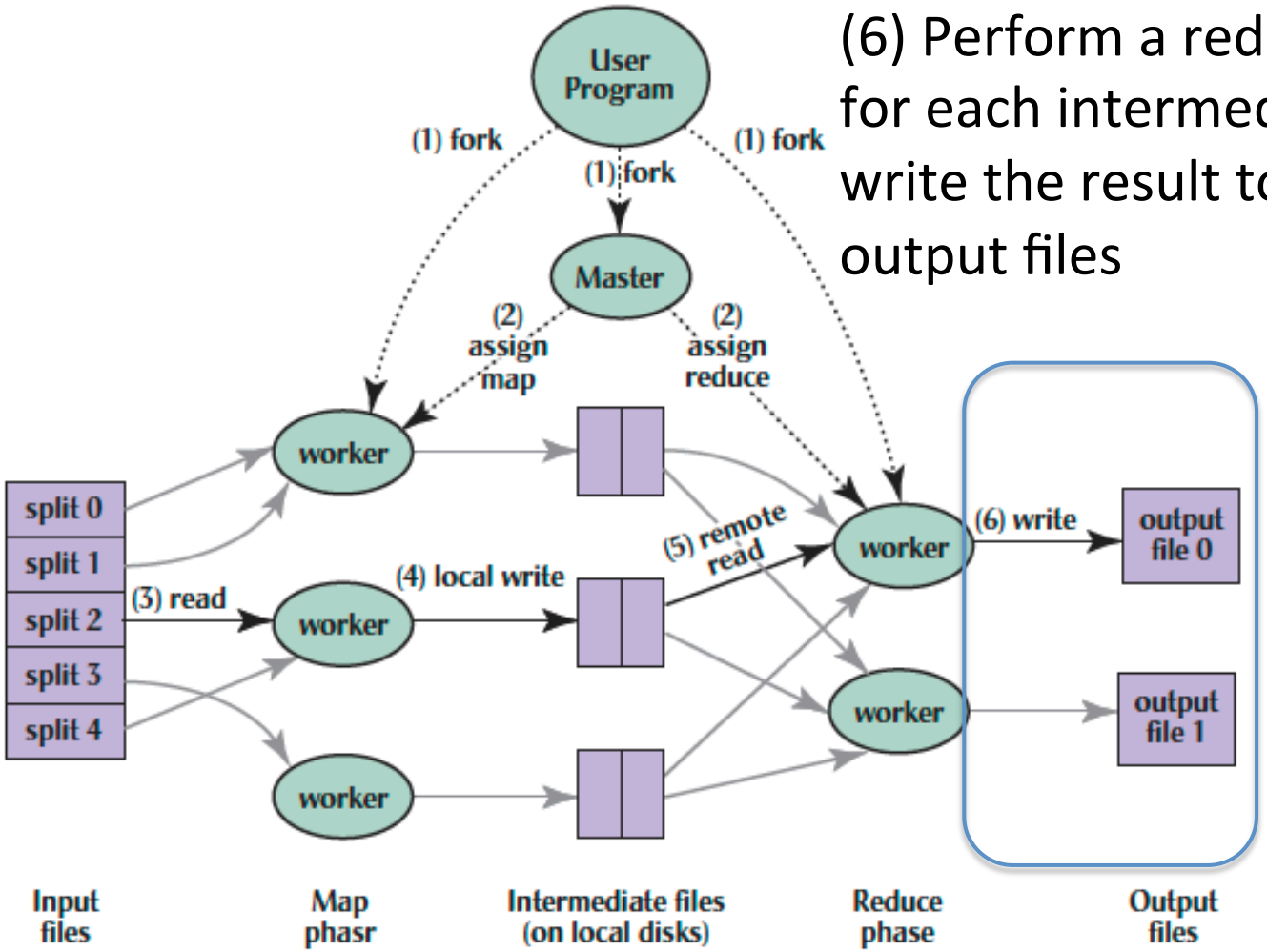


MapReduce Execution



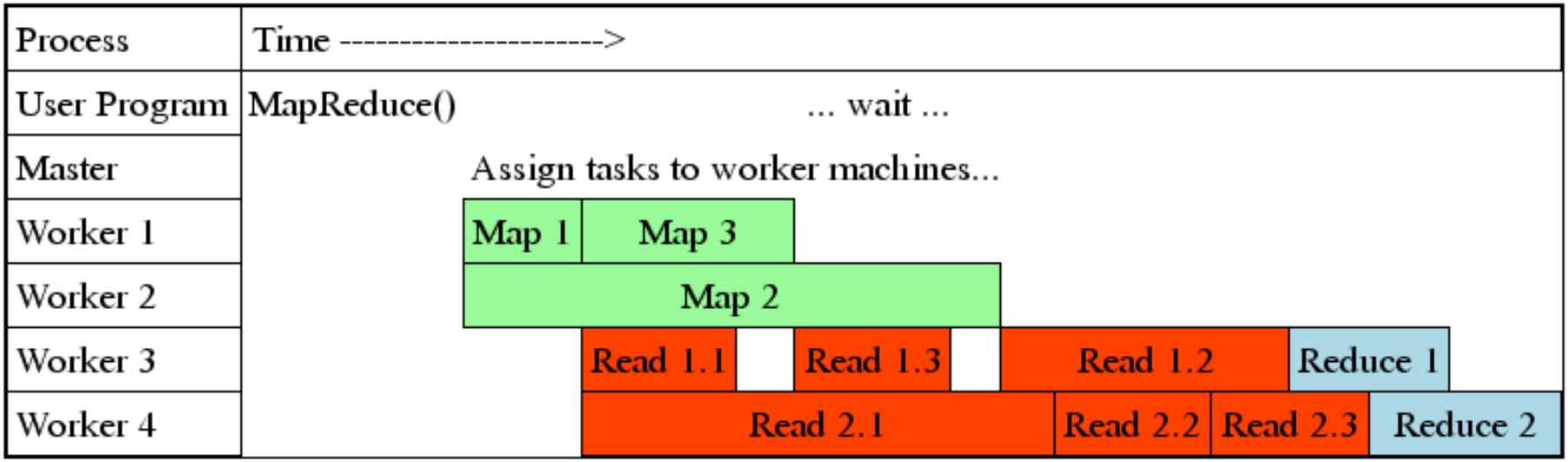
(5) Read intermediate key/value pairs, sort them by its key.

MapReduce Execution



(6) Perform a reduce task for each intermediate key, write the result to the output files

MapReduce Processing Time Line



- Master assigns map + reduce tasks to “worker” servers
- As soon as a map task finishes, worker server can be assigned a new map or reduce task
- Data shuffle begins as soon as a given Map finishes
- Reduce task begins as soon as all data shuffles finish
- To tolerate faults, reassign task if a worker server “dies”

Big Data Framework: Hadoop & Spark

- Apache Hadoop
  - Open-source MapReduce Framework
  - Hadoop Distributed File System (HDFS)
  - MapReduce Java APIs

- Apache Spark
  - Fast and general engine for large-scale data processing.
  - Originally developed in the AMP lab at UC Berkeley
  - Running on HDFS
  - Provides Java, Scala, Python APIs for
    - Database
    - Machine learning
    - Graph algorithm

WordCount in Hadoop's Java API

```
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(WCMap.class);
    conf.setCombinerClass(WCReduce.class);
    conf.setReducerClass(WCReduce.class);
    conf.setInputPath(new Path(args[0]));
    conf.setOutputPath(new Path(args[1]));
    JobClient.runJob(conf);
}

public class WCMap extends MapReduceBase implements Mapper {
    private static final IntWritable ONE = new IntWritable(1);
    public void map(WritableComparable key, Writable value,
                    OutputCollector output,
                    Reporter reporter) throws IOException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            output.collect(new Text(itr.next()), ONE);
        }
    }
}

public class WCReduce extends MapReduceBase implements Reducer {
    public void reduce(WritableComparable key, Iterator values,
                       OutputCollector output,
                       Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += ((IntWritable) values.next()).get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

Word Count in Spark's Python API

```
// RDD: primary abstraction of a distributed collection of items
file = sc.textFile("hdfs://…")
// Two kinds of operations:
// Actions: RDD → Value
// Transformations: RDD → RDD
// e.g. flatMap, Map, reduceByKey
file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
```

Summary

- Warehouse Scale Computers
  - New class of computers
  - Scalability, energy efficiency, high failure rate
- Request-level parallelism

  e.g. Web Search
- Data-level parallelism on a large dataset
  - MapReduce
  - Hadoop, Spark