# CS 61C: Great Ideas in Computer Architecture (Machine Structures) *Warehouse-Scale Computing, MapReduce, and Spark*

Instructors:

John Wawrzynek & Vladimir Stojanovic

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

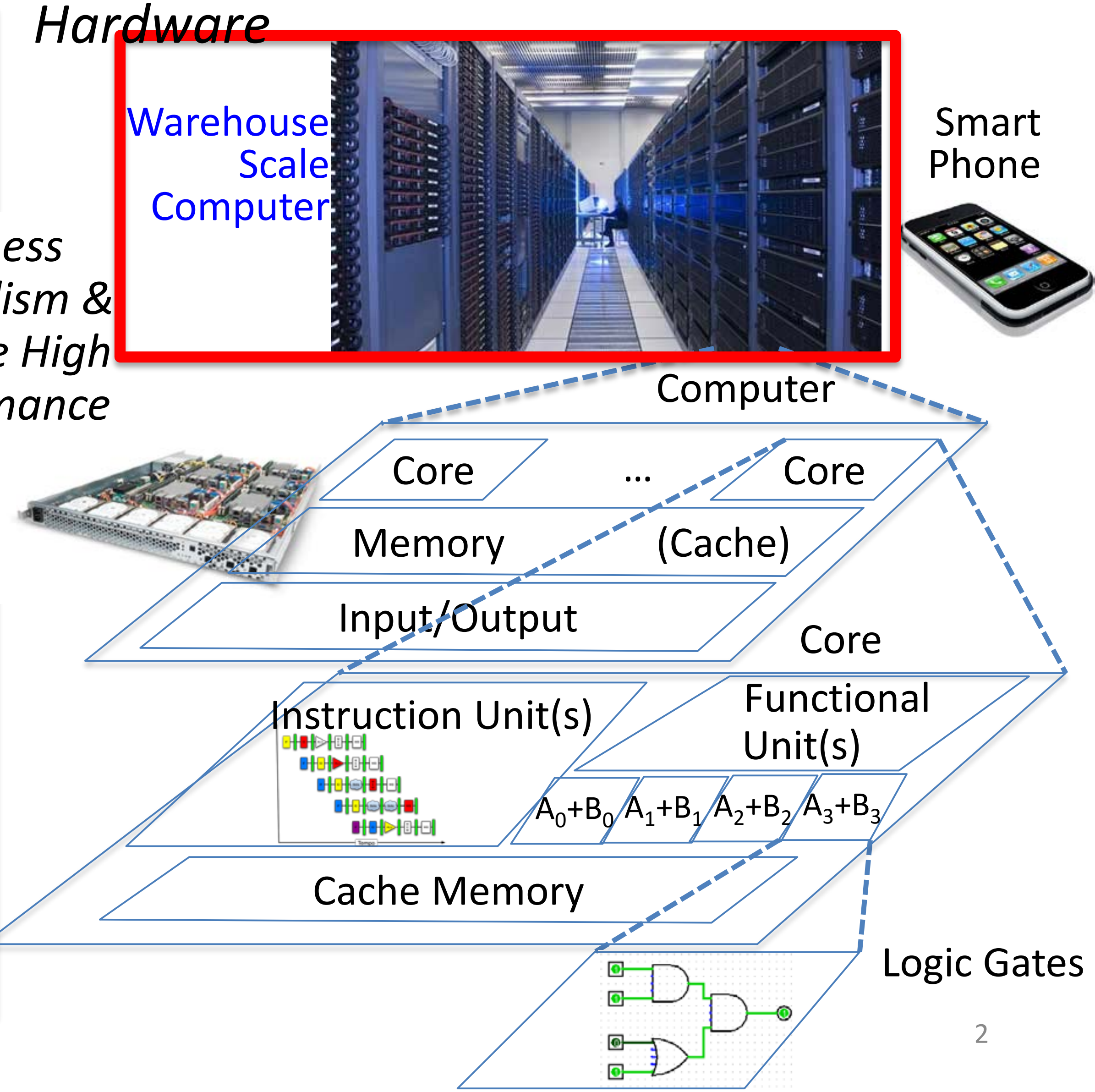
# New-School Machine Structures (It's a bit more complicated!)

*Software*

- Parallel Requests
  Assigned to computer
  e.g., Search "cats"
- 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

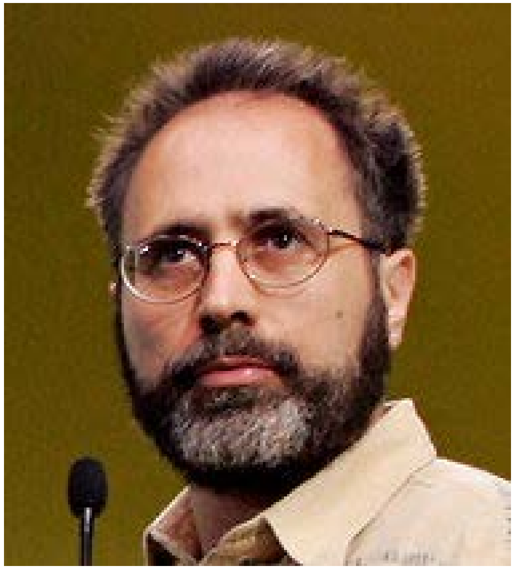
*Harness Parallelism & Achieve High Performance*

*Hardware*



# In the news

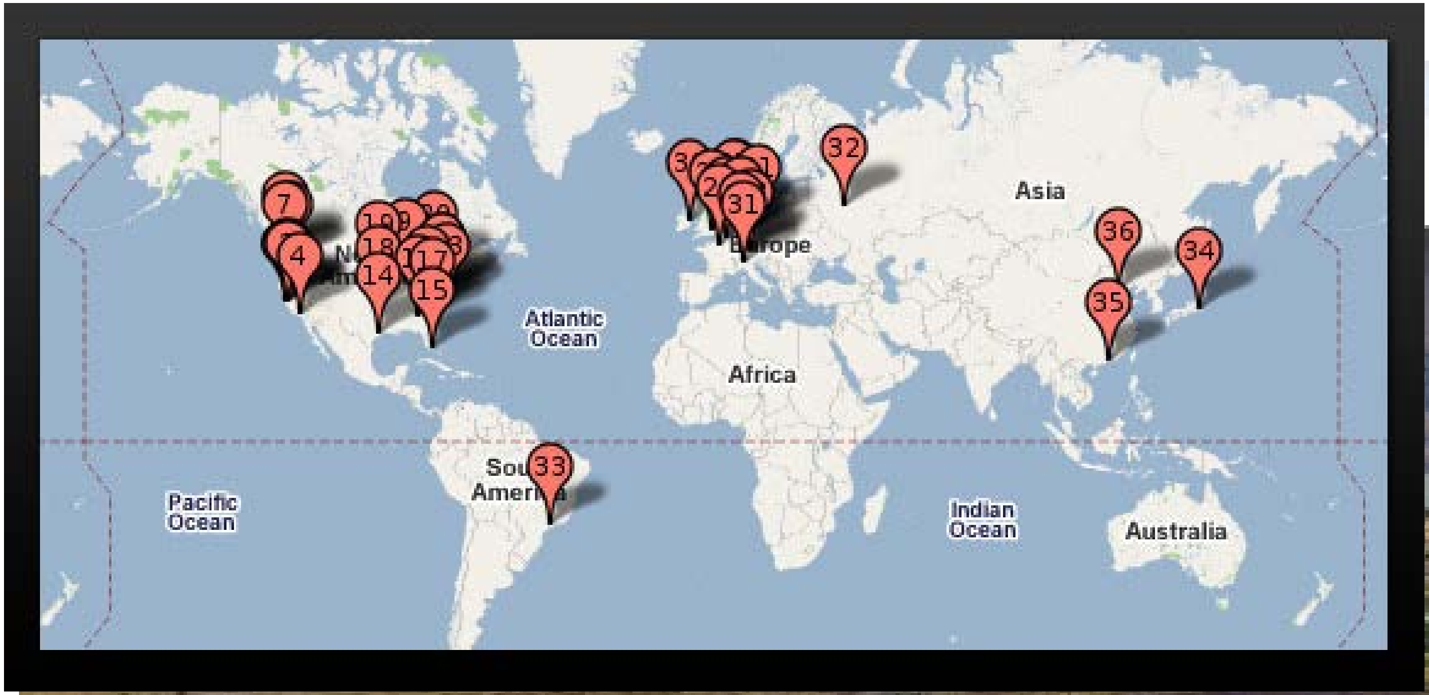
- Google disclosed that it continuously uses enough electricity to power 200,000 homes, but it says that in doing so, it also makes the planet greener.
- Average energy use per typical user per month is same as running a 60-watt bulb for 3 hours (180 watt-hours).



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

# Google's WSCs

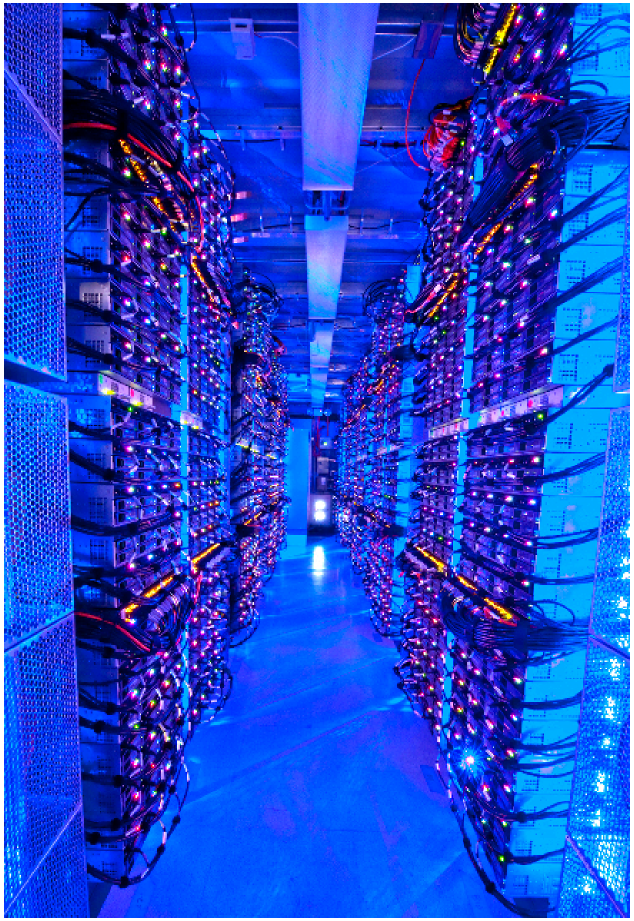


Ex: In Oregon

# Containers in WSCs

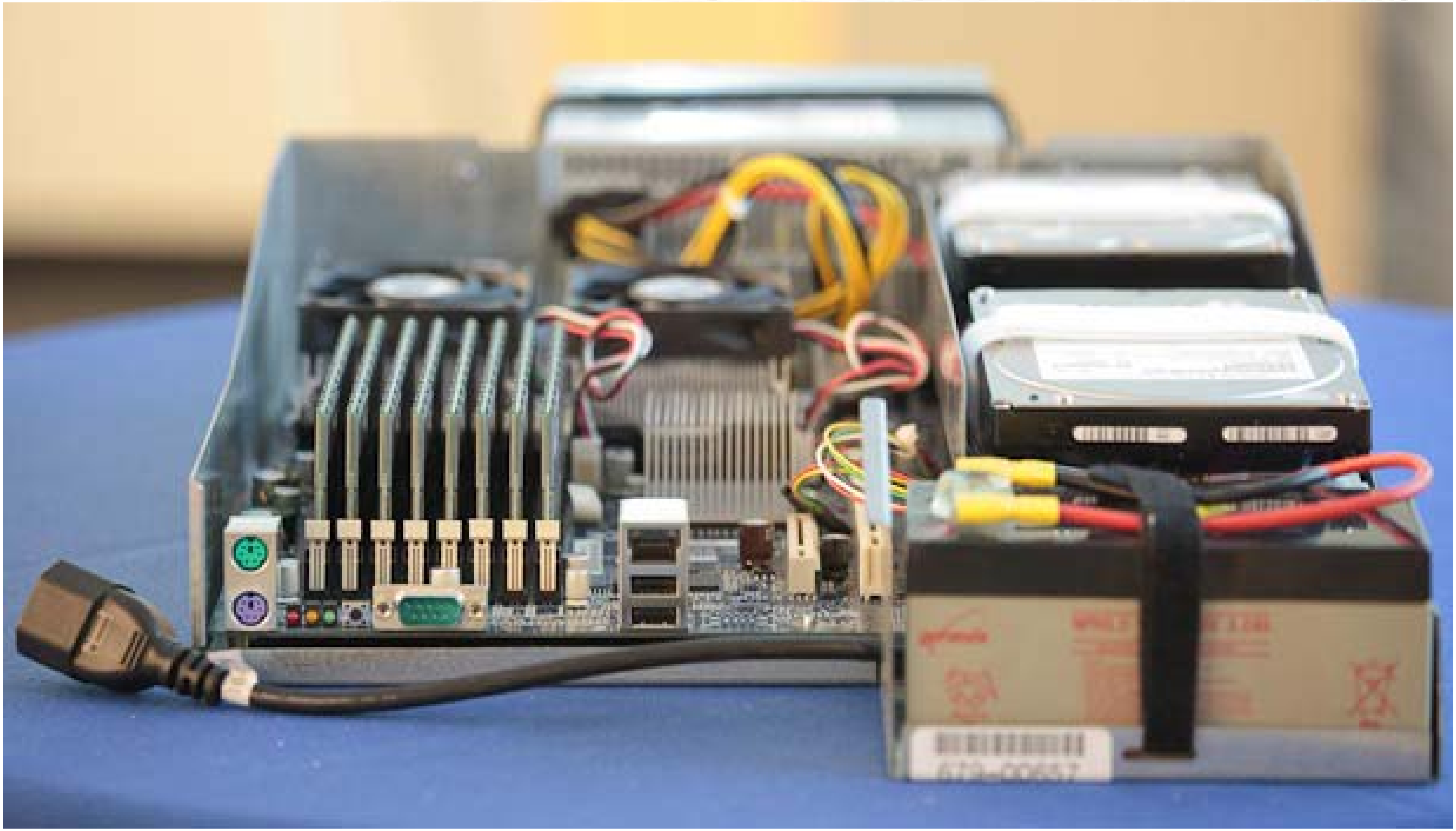
Inside WSC

Inside Container



# Server, Rack, Array

# Google Server Internals



# Warehouse-Scale Computers

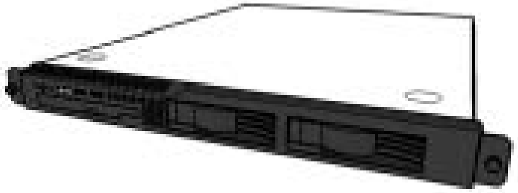
- Datacenter
  - Collection of 10,000 to 100,000 servers
  - Networks connecting them together
- ***Single gigantic*** machine
- Very large applications (Internet service): search, email, video sharing, social networking
- Very high availability
- "…WSCs are no less worthy of the expertise of computer 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 (ex: AWS)
  - **High # of failures**

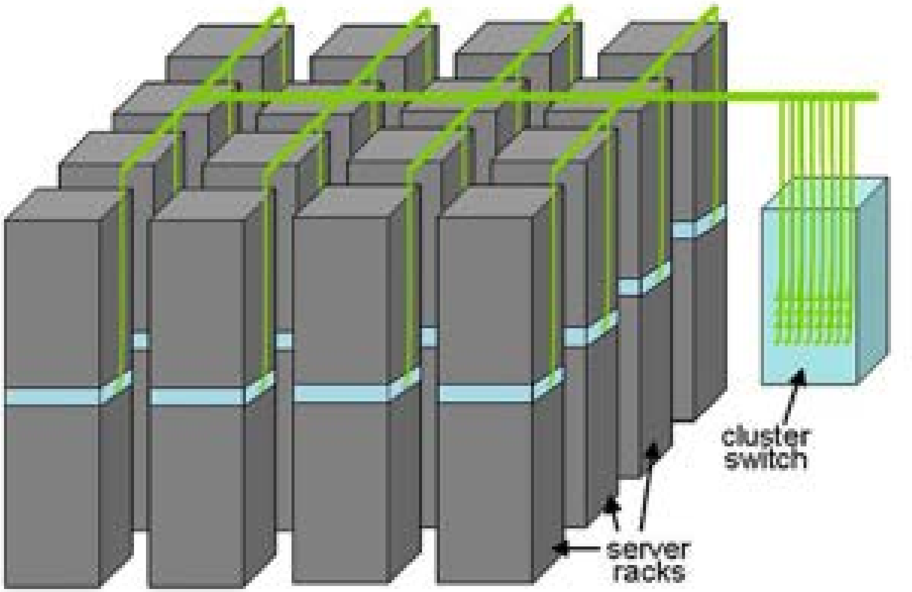
    ex: 4 disks/server, annual failure rate: 4%
    → WSC of 50,000 servers: 1 disk fail/hour
- Operation Cost Count
  - Longer life time (>10 years)
  - **Cost of equipment purchases << cost of ownership**

# WSC Architecture



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

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

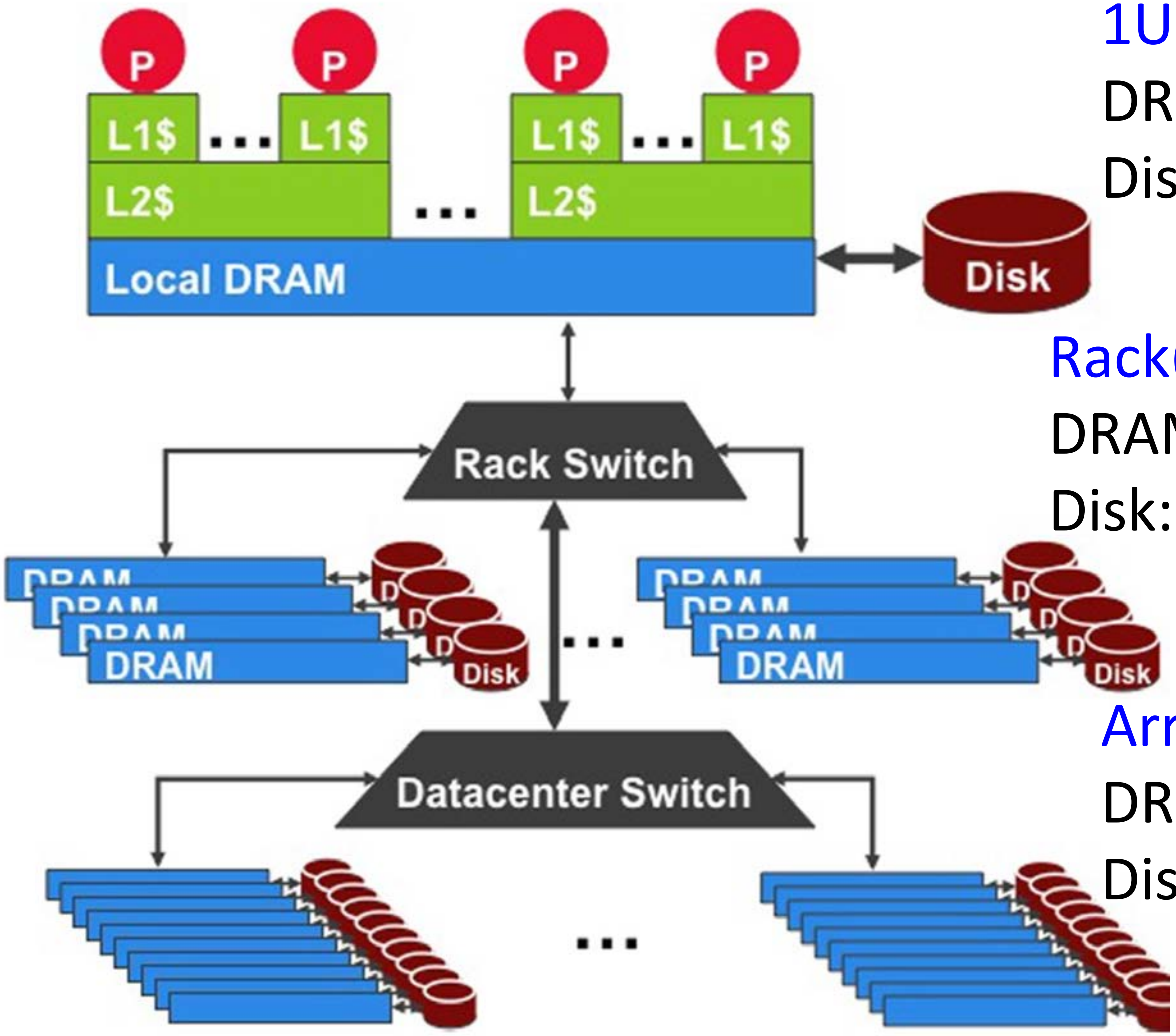


Array (aka cluster):
16-32 racks
Expensive switch
(10X bandwidth → 100x cost)

# 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

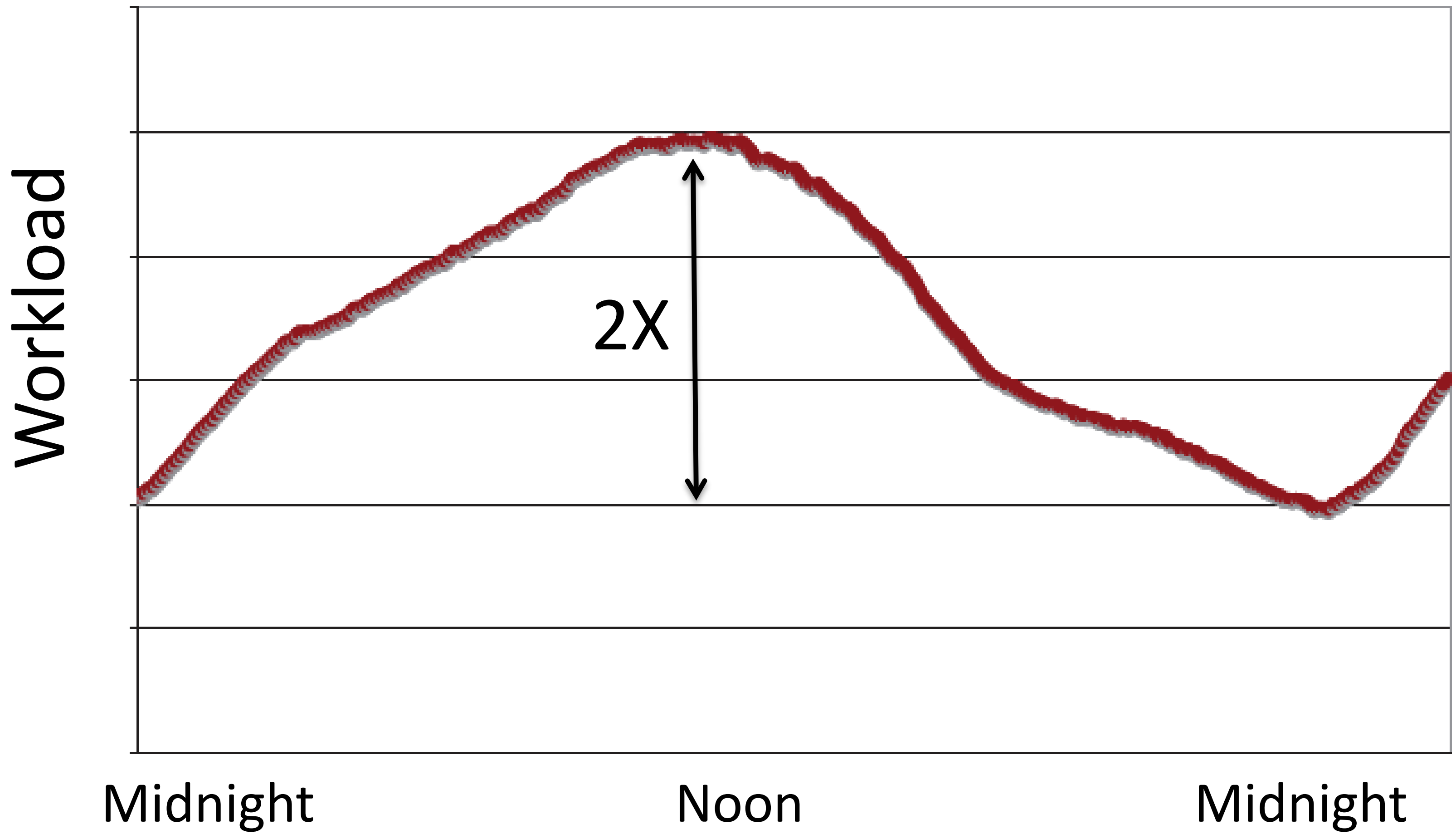
Disk: 160TB, 11ms, 100MB/s

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 server 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
  - Perfection = 1.0
  - Google WSC’s PUE = 1.2

# PUE in the Wild (2007)

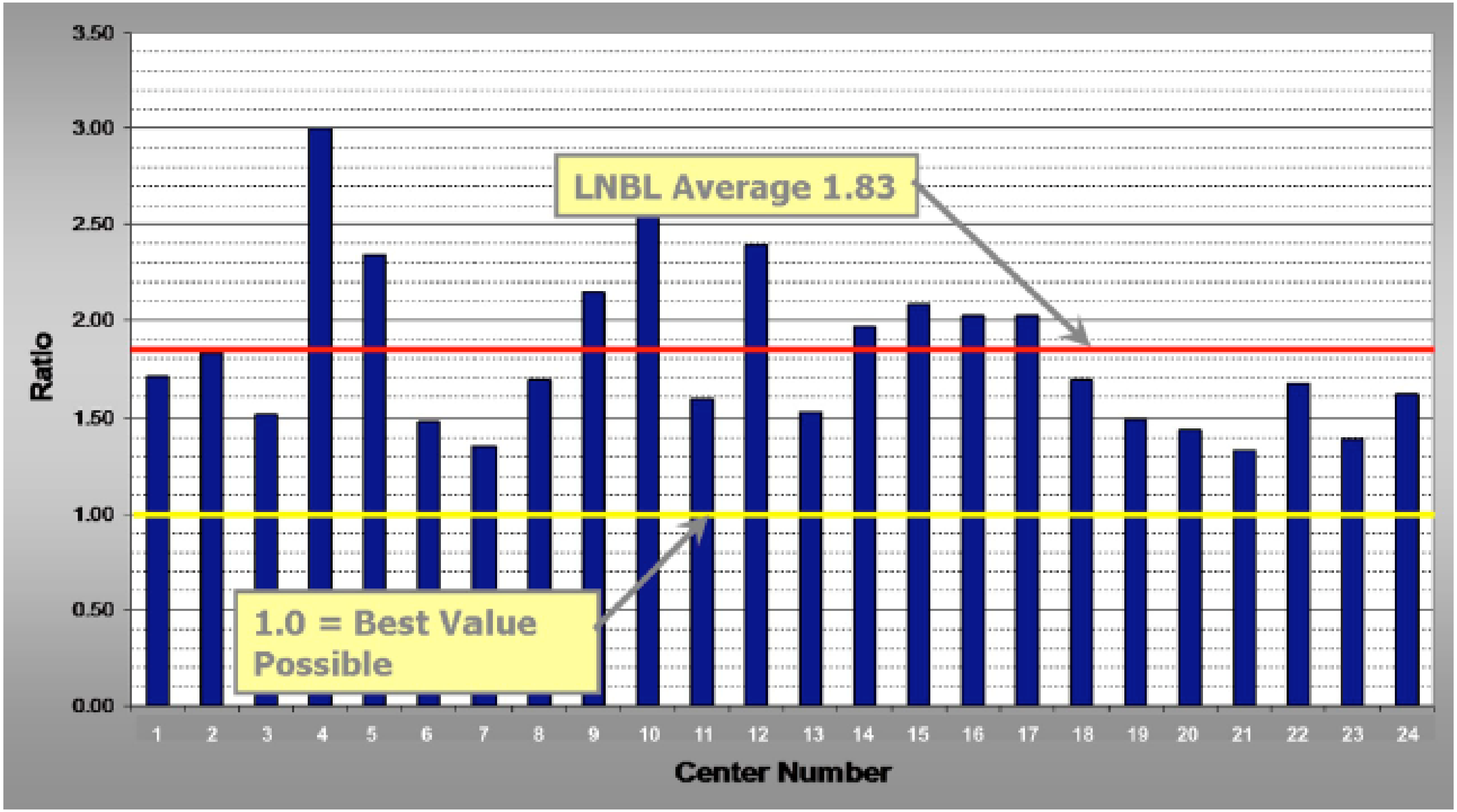
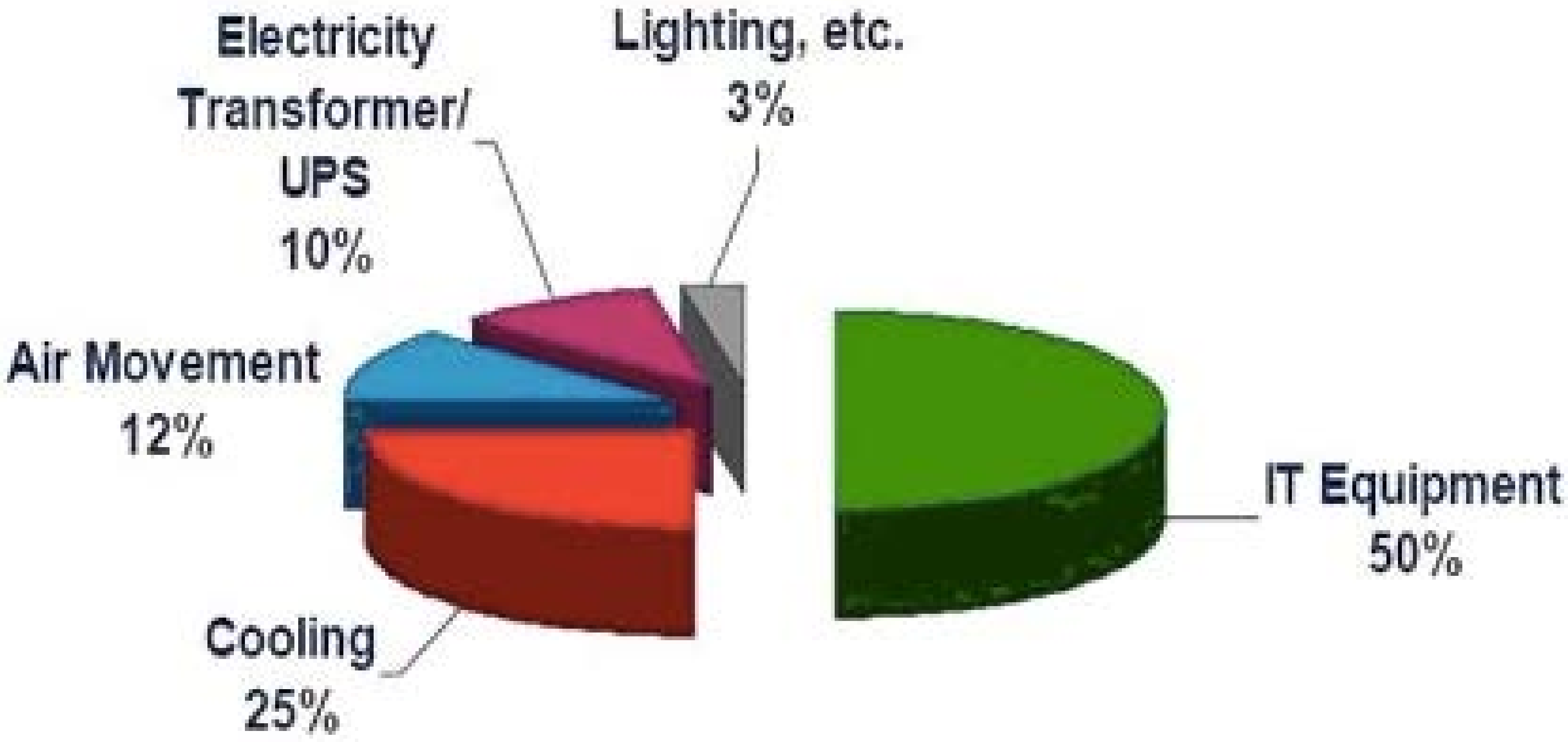
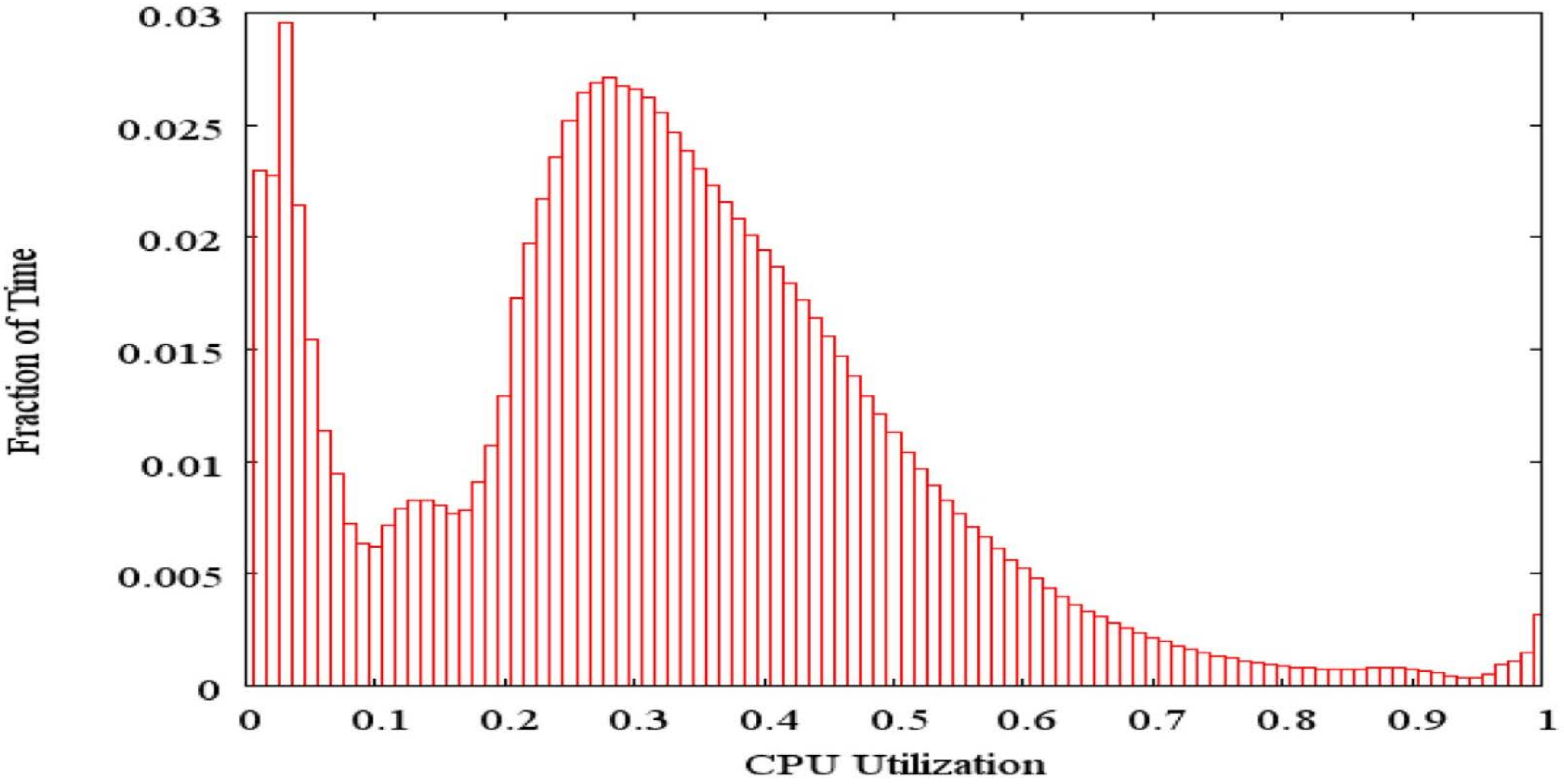


FIGURE 5.1: LBNL survey of the power usage efficiency of 24 datacenters, 2007 (Greenberg et al.)

# *Where Data Center Power Goes*



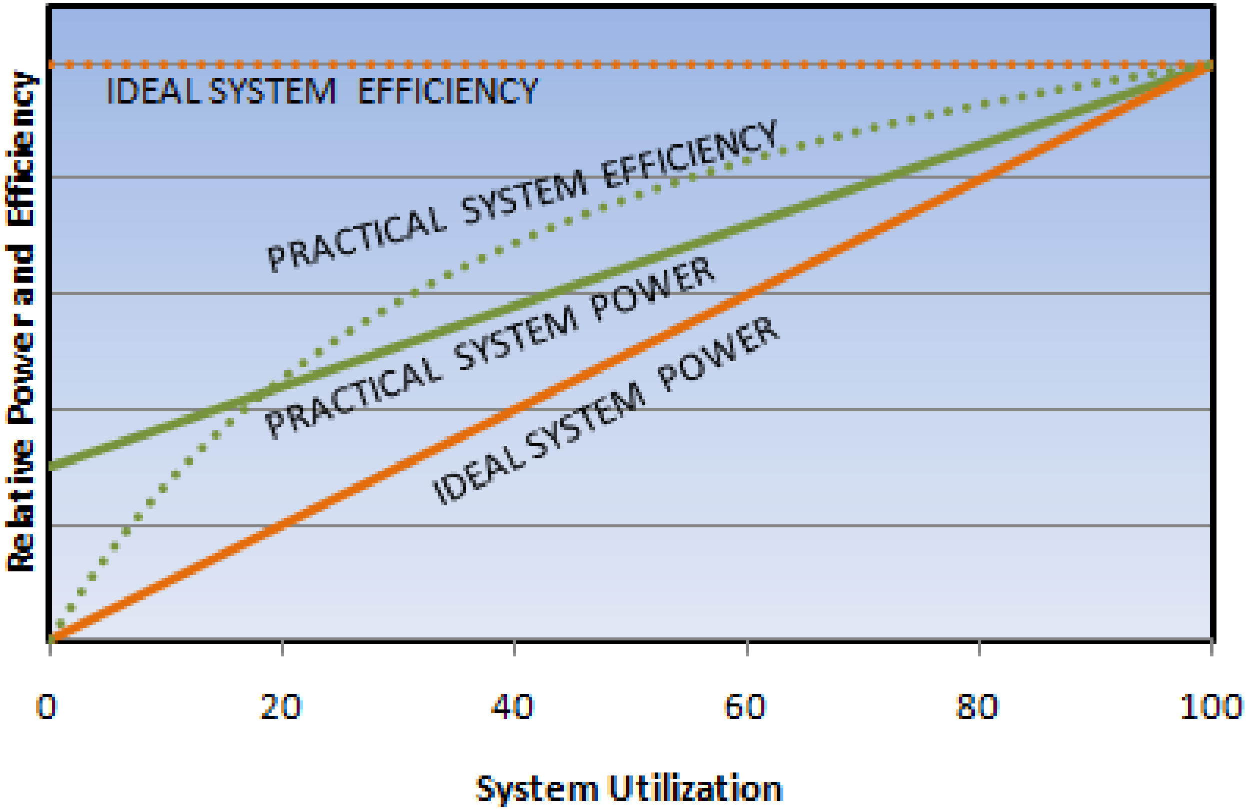
# Load Profile of WSCs



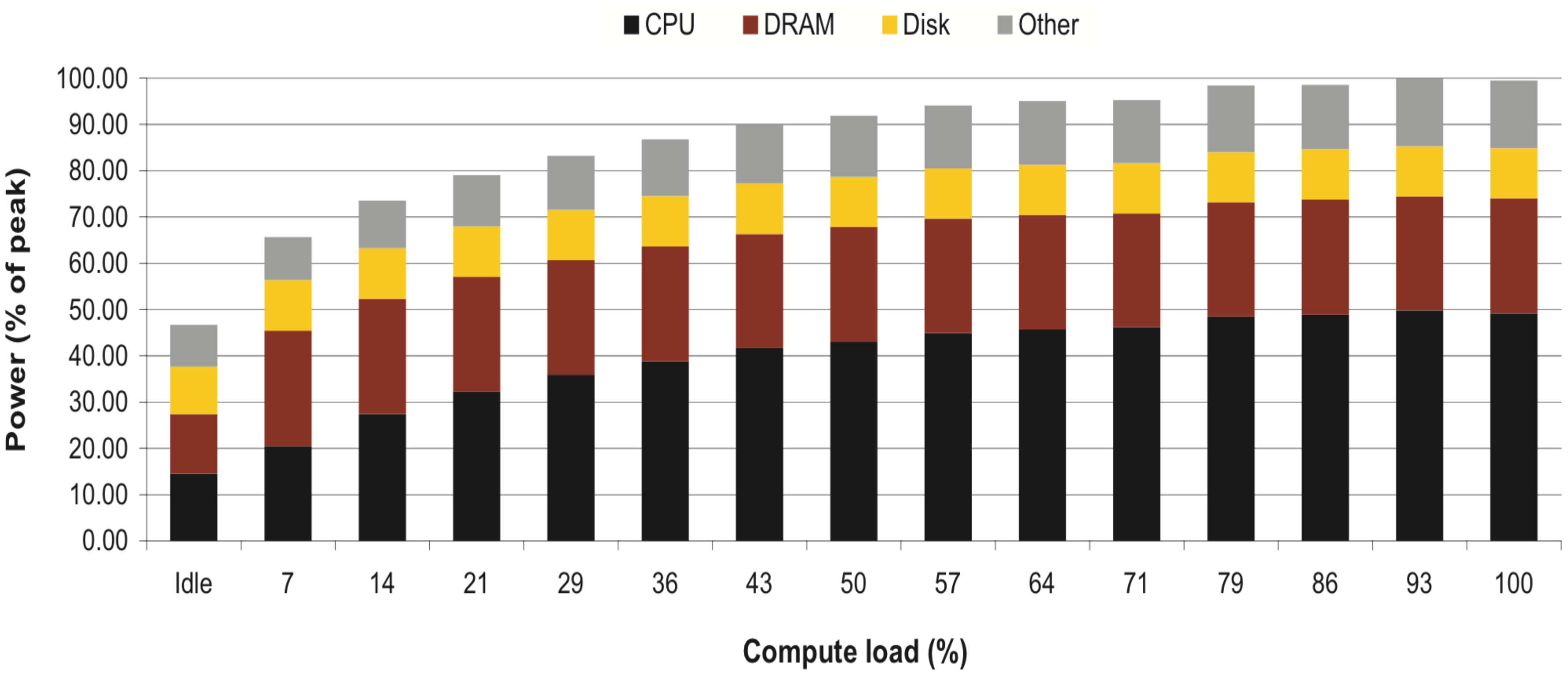
- Average CPU utilization of 5,000 Google servers, 6 month period
- Servers rarely idle or fully utilized, operating most of the time at ***10% to 50%*** of their maximum utilization

# Energy-Proportional Computing: Design Goal of WSC

- Energy = Power x Time, Efficiency = Computation / Energy
- Desire:
  - Consume almost no power when idle (“Doing nothing well”)
  - 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

# Agenda

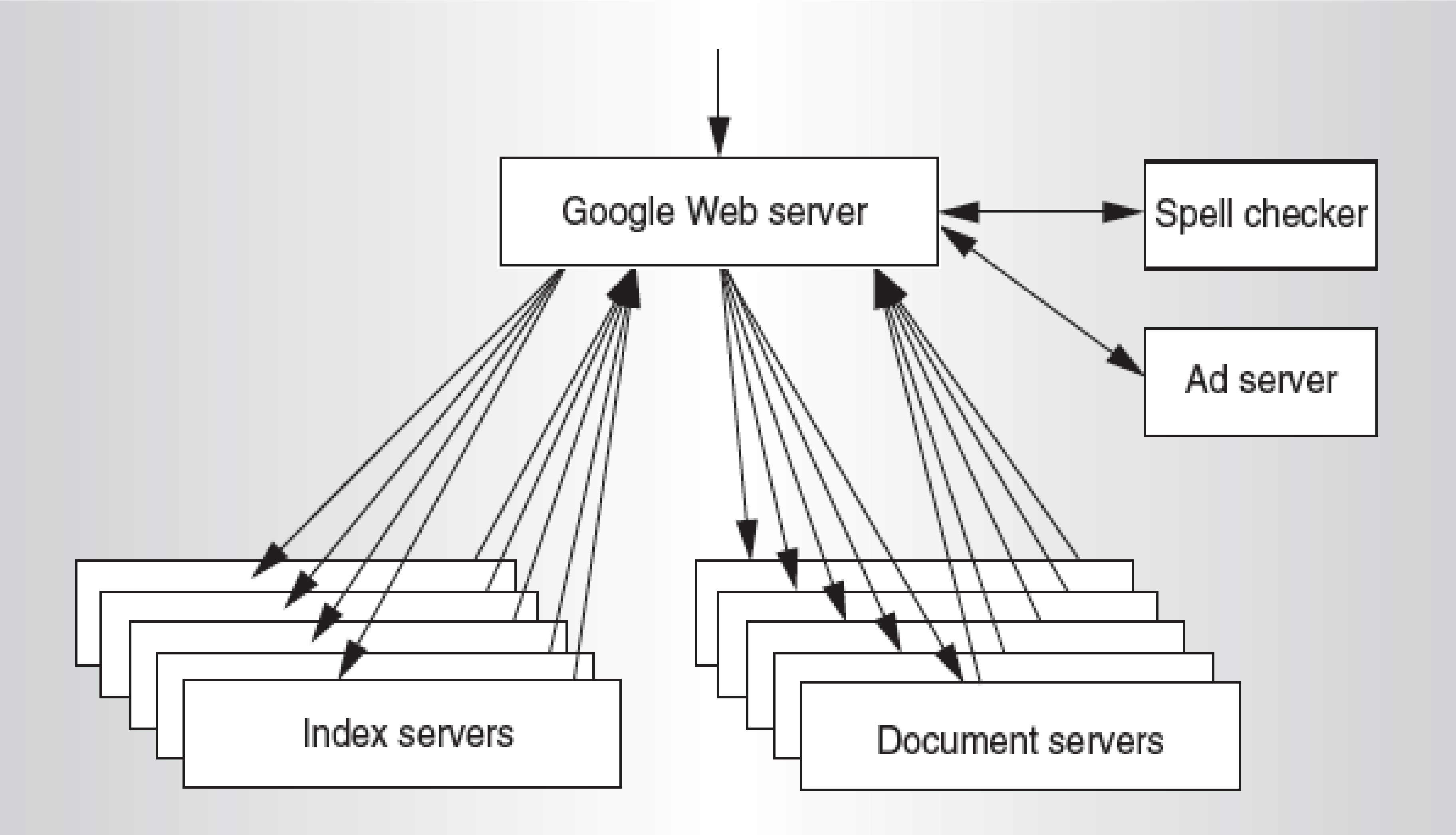
- Warehouse Scale Computing
- Request-level Parallelism

  e.g. Web search
- Administrivia & Clickers/Peer Instructions
- Data-level Parallelism
  - Hadoop, Spark
  - MapReduce
- (Bonus) Convolutional Neural Networks

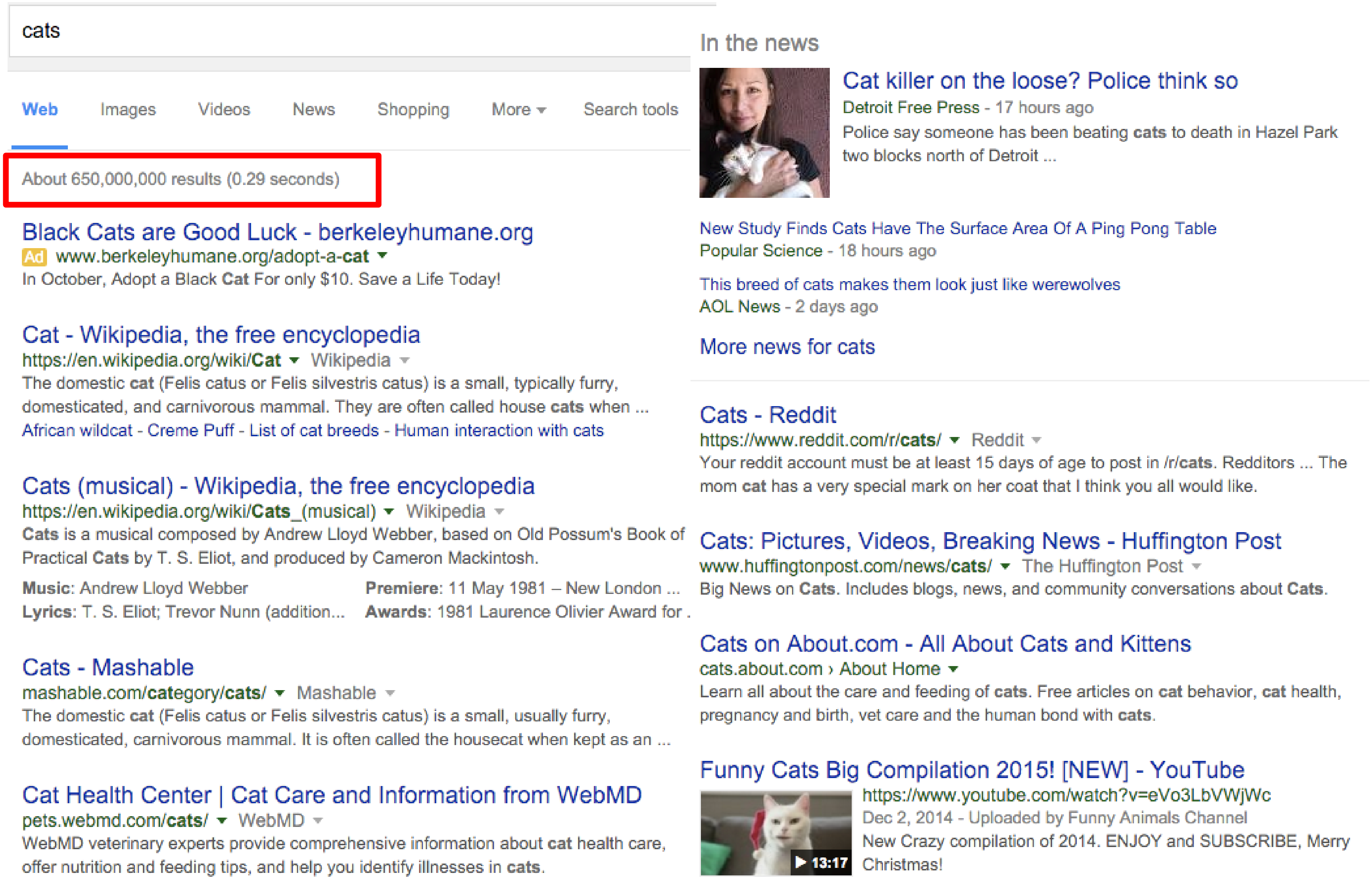
# Request-Level Parallelism (RLP)

- Hundreds of thousands of requests per sec.
  - 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



About 650,000,000 results (0.29 seconds)

Black Cats are Good Luck - berkeleyhumane.org
Ad www.berkeleyhumane.org/adopt-a-**cat** ▾
In October, Adopt a Black **Cat** For only $10. Save a Life Today!

Cat - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Cat** ▾ Wikipedia ▾
The domestic **cat** (Felis catus or Felis silvestris catus) is a small, typically furry, domesticated, and carnivorous mammal. They are often called house **cats** when ...
African wildcat - Creme Puff - List of cat breeds - Human interaction with cats

Cats (musical) - Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/**Cats**_(musical) ▾ Wikipedia ▾
**Cats** is a musical composed by Andrew Lloyd Webber, based on Old Possum's Book of Practical **Cats** by T. S. Eliot, and produced by Cameron Mackintosh.
**Music**: Andrew Lloyd Webber
**Lyrics**: T. S. Eliot; Trevor Nunn (addition...
**Premiere**: 11 May 1981 – New London ...
**Awards**: 1981 Laurence Olivier Award for .

Cats - Mashable
mashable.com/category/**cats**/ ▾ Mashable ▾
The domestic **cat** (Felis catus or Felis silvestris catus) is a small, usually furry, domesticated, carnivorous mammal. It is often called the housecat when kept as an ...

Cat Health Center | Cat Care and Information from WebMD
pets.webmd.com/**cats**/ ▾ WebMD ▾
WebMD veterinary experts provide comprehensive information about **cat** health care, offer nutrition and feeding tips, and help you identify illnesses in **cats**.

In the news

Cat killer on the loose? Police think so
Detroit Free Press - 17 hours ago
Police say someone has been beating **cats** to death in Hazel Park two blocks north of Detroit ...

New Study Finds Cats Have The Surface Area Of A Ping Pong Table
Popular Science - 18 hours ago

This breed of cats makes them look just like werewolves
AOL News - 2 days ago

More news for cats

Cats - Reddit
https://www.reddit.com/r/**cats**/ ▾ Reddit ▾
Your reddit account must be at least 15 days of age to post in /r/**cats**. Redditors ... The mom **cat** has a very special mark on her coat that I think you all would like.

Cats: Pictures, Videos, Breaking News - Huffington Post
www.huffingtonpost.com/news/**cats**/ ▾ The Huffington Post ▾
Big News on **Cats**. Includes blogs, news, and community conversations about **Cats**.

Cats on About.com - All About Cats and Kittens
cats.about.com › About Home ▾
Learn all about the care and feeding of **cats**. Free articles on **cat** behavior, **cat** health, pregnancy and birth, vet care and the human bond with **cats**.

Funny Cats Big Compilation 2015! [NEW] - YouTube
https://www.youtube.com/watch?v=eVo3LbVWjWc
Dec 2, 2014 - Uploaded by Funny Animals Channel
New Crazy compilation of 2014. ENJOY and SUBSCRIBE, Merry Christmas!

# Anatomy of a Web Search (1/3)

- Google “cats”
  - Direct request to “closest” Google WSC
  - Front-end load balancer directs request to one of many arrays (cluster of servers) 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 word, “cats”
  - 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
- 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 Instructions
- Data-level Parallelism
  - Hadoop, Spark
  - MapReduce
- Convolutional Neural Networks

# Clickers:

- What did you think of the midterm exam?

A: Very difficult

- 
- 
-

E: Too easy

# Administrivia

- Please go to their own lab this week for checkoff.
  - If you wait to get checked off in the Monday lab, there may be too many people for everyone to be checked off.
- We expect to have MT2 graded by Monday 11/23
- Project 4 out
  - Image processing performance optimization
  - Parallelization with SSE (DLP) & openMP (TLP)
  - due 11/18 (next Wednesday)

# 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: More than half of the power of WSCs goes into cooling.**
- **E: WSCs contain many copies of data.**

# Agenda

- Warehouse Scale Computing
- Administrivia & Clickers/Peer Instructions
- Request-level Parallelism

  e.g. Web search
- Data-level Parallelism
  - MapReduce
  - Hadoop, Spark
- (Bonus) Convolutional Neural Networks

# 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)

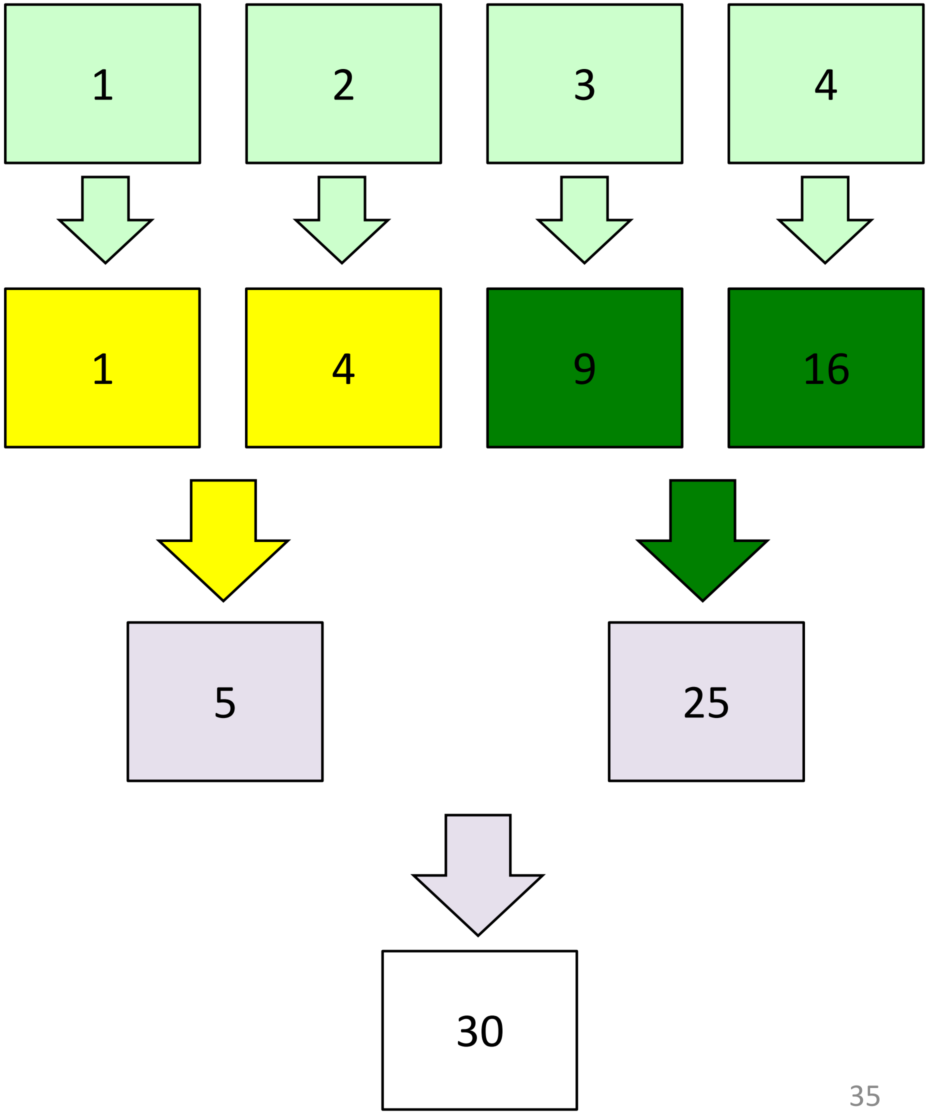
# What is MapReduce used for?

- 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

# Inspiration: Map & Reduce Functions, ex: Python

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

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



# 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

***Task of counting the number of occurrences of each word in a large collection of documents.***

User-written Map function reads the document data and parses out the words. For each word, it writes the (key, value) pair of (word, 1). That is, the word is treated as the intermediate key and the associated value of 1 means that we saw the word once.

***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)
```

# MapReduce Word Count Example

*Task of counting the number of occurrences of each word in a large collection of documents.*

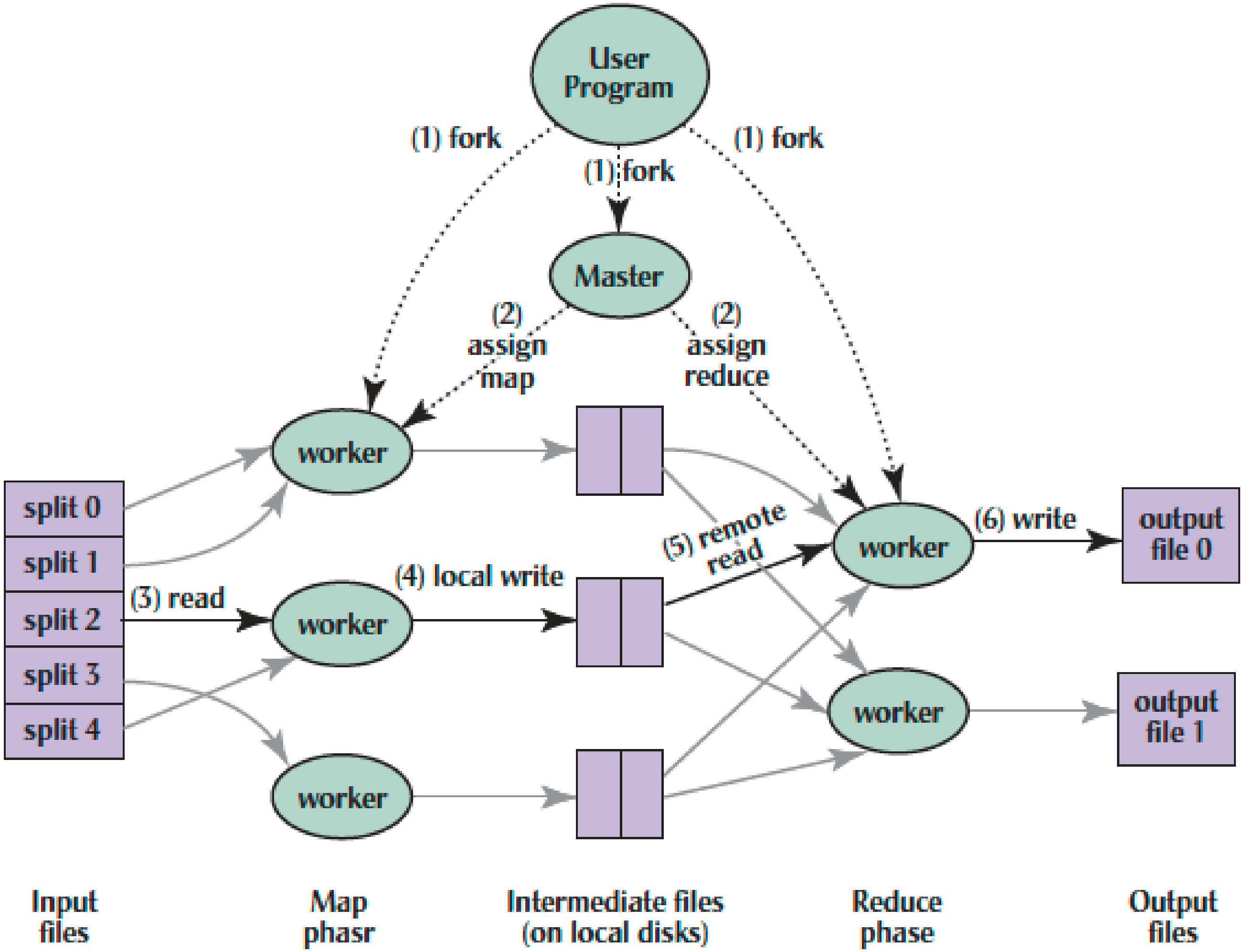
The intermediate data is then sorted by MapReduce by keys and the user’s Reduce function is called for each unique key. In this case, Reduce is called with a list of a "1" for each occurrence of the word that was parsed from the document. The function adds them up to generate a total word count for that word.

***Reduce*** phase: (word, list(counts)) → (word, count_sum)

```
// (“I”, [1,1]) → (“I”,2)
```

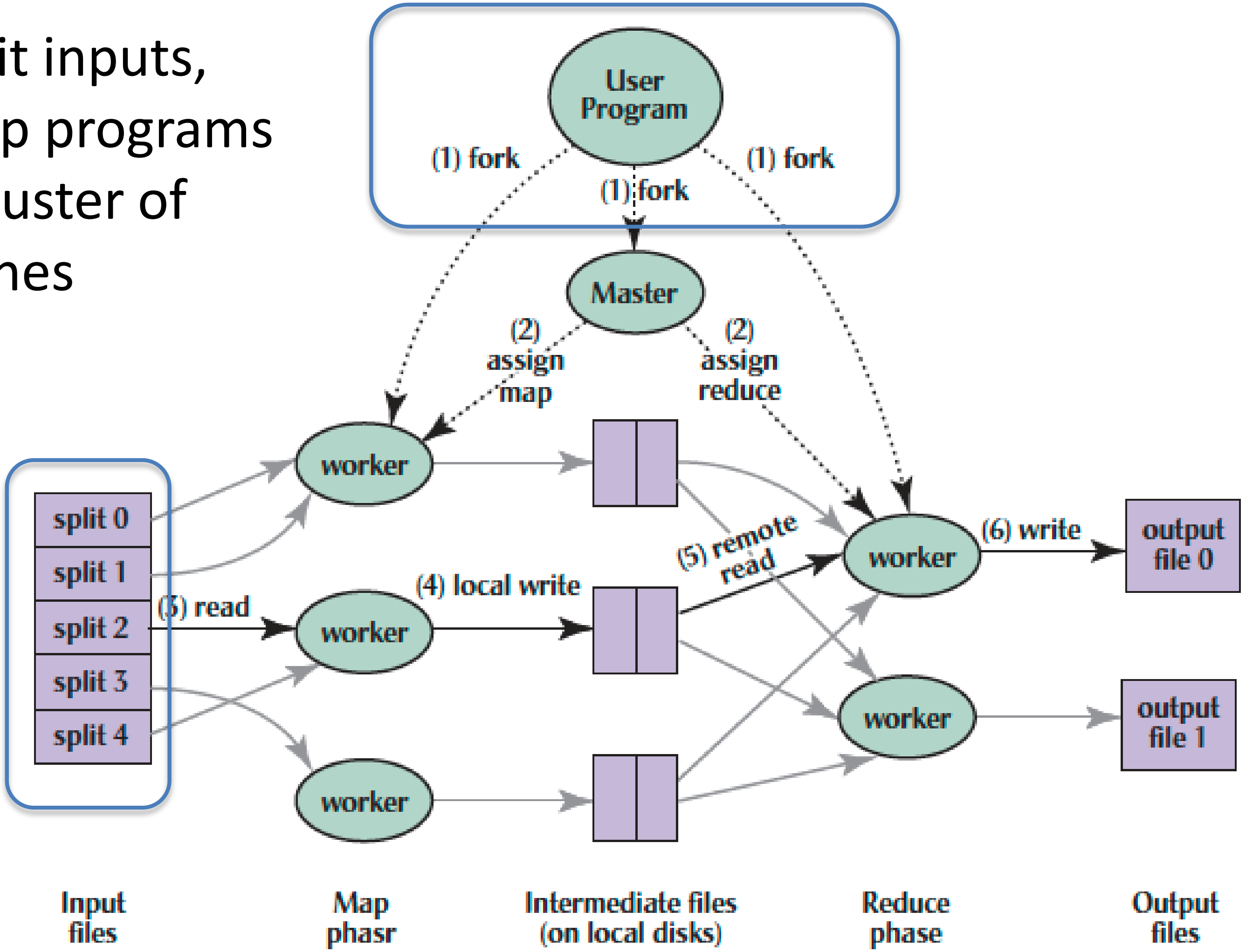
```
reduce(key, values):
  result = 0
  for each v in values:
    result += v
  emit(key, result)
```

# MapReduce Implementation



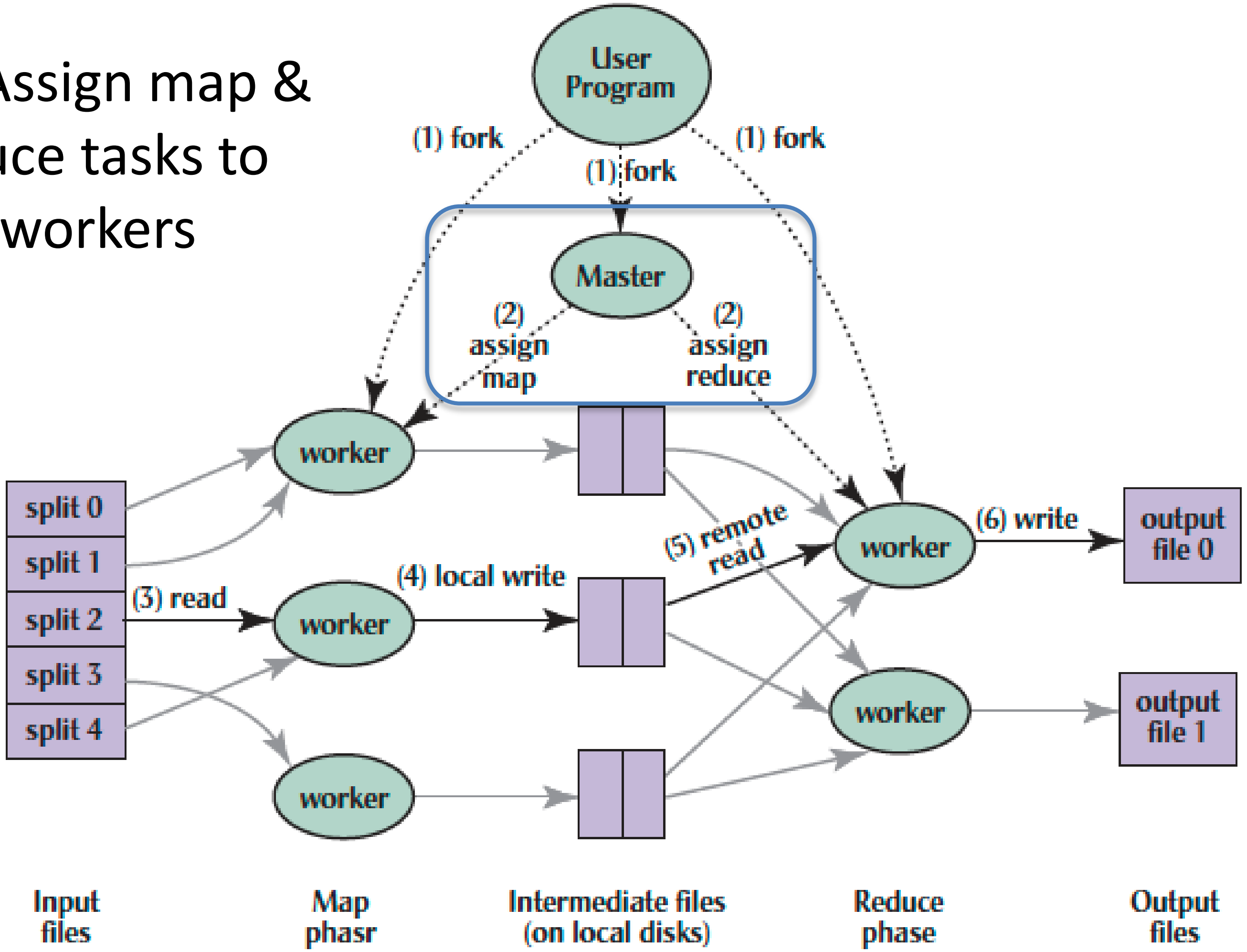
# MapReduce Execution

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



# MapReduce Execution

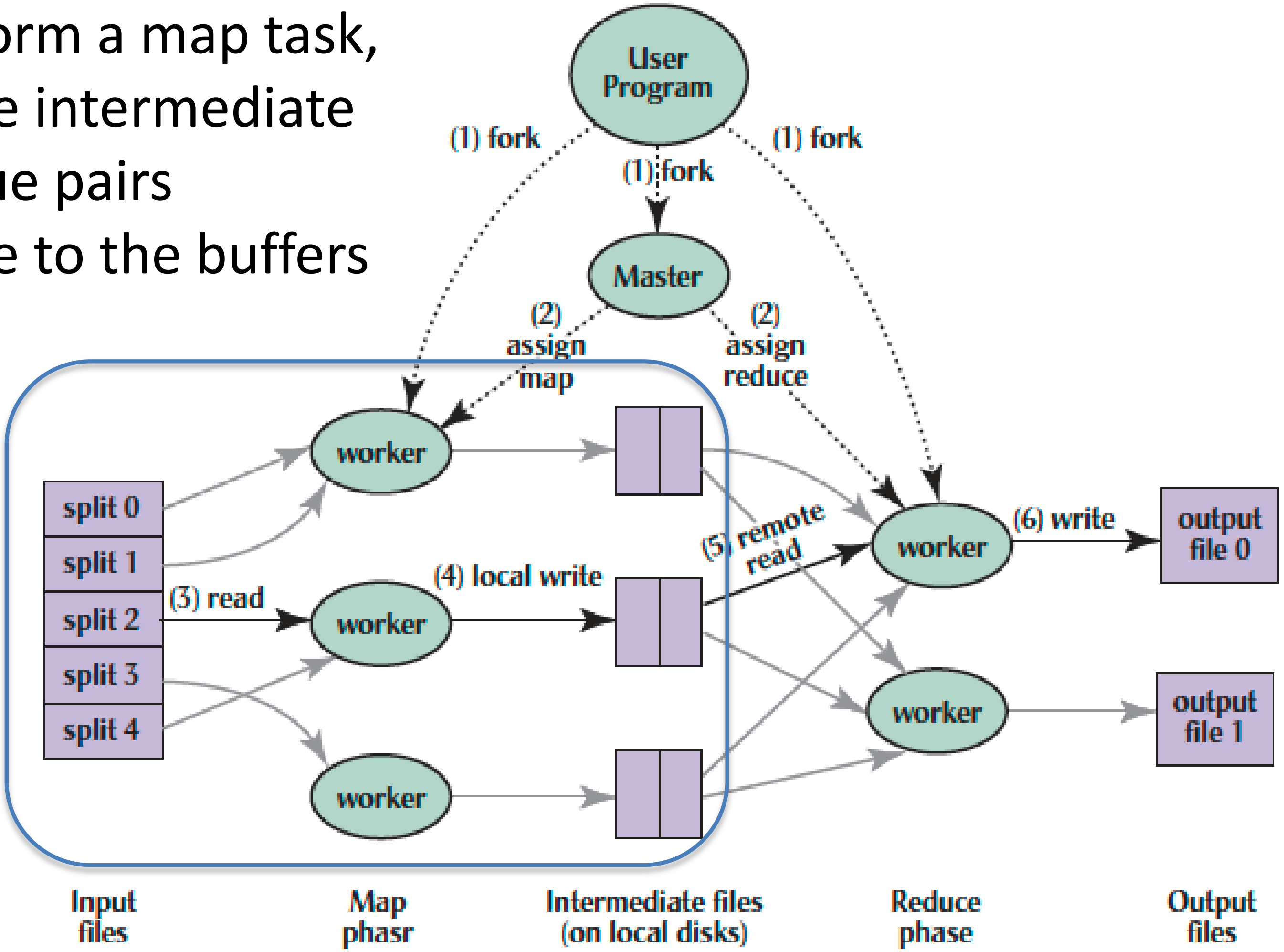
(2) Assign map & reduce tasks to idle workers



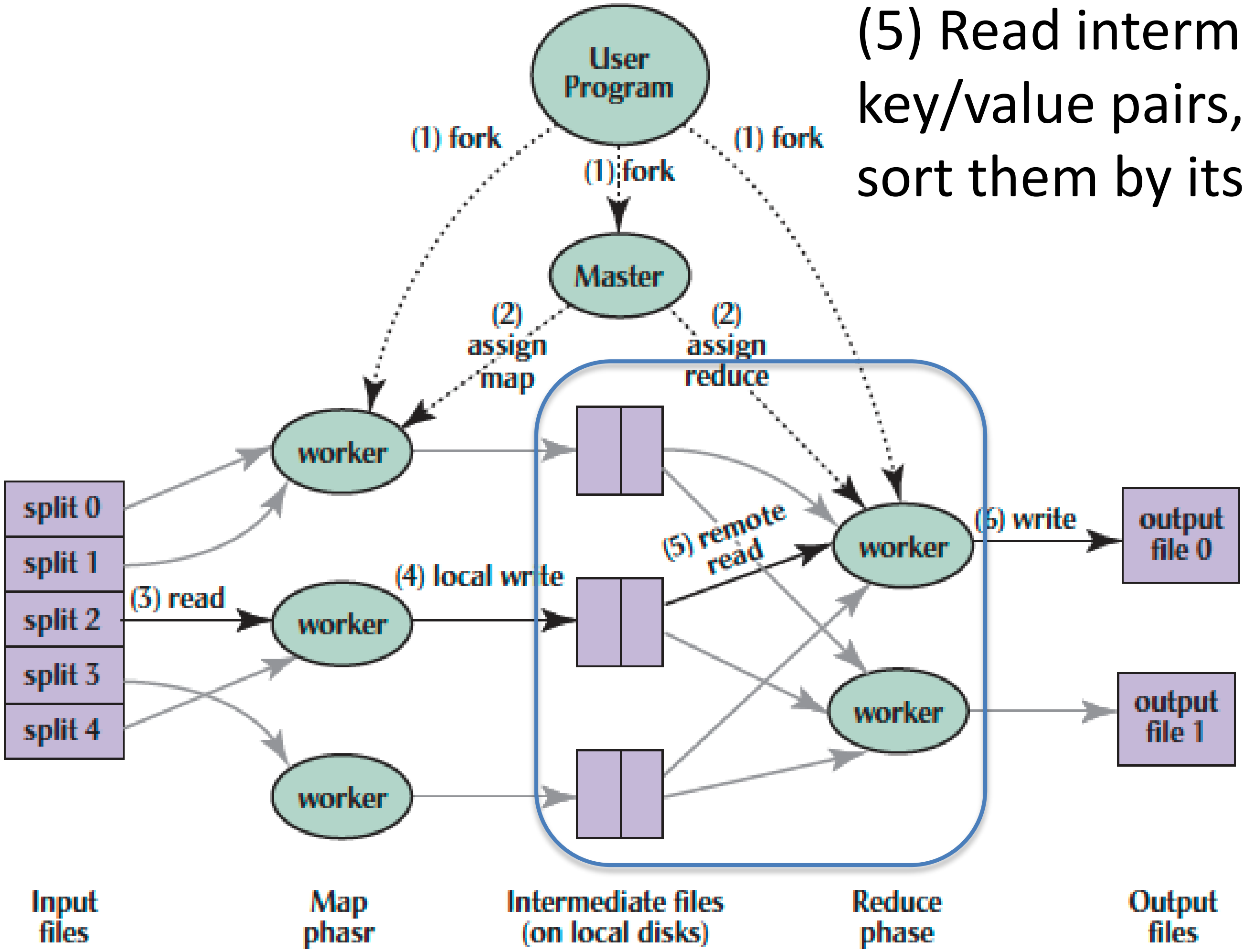
# MapReduce Execution

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

(4) Write to the buffers



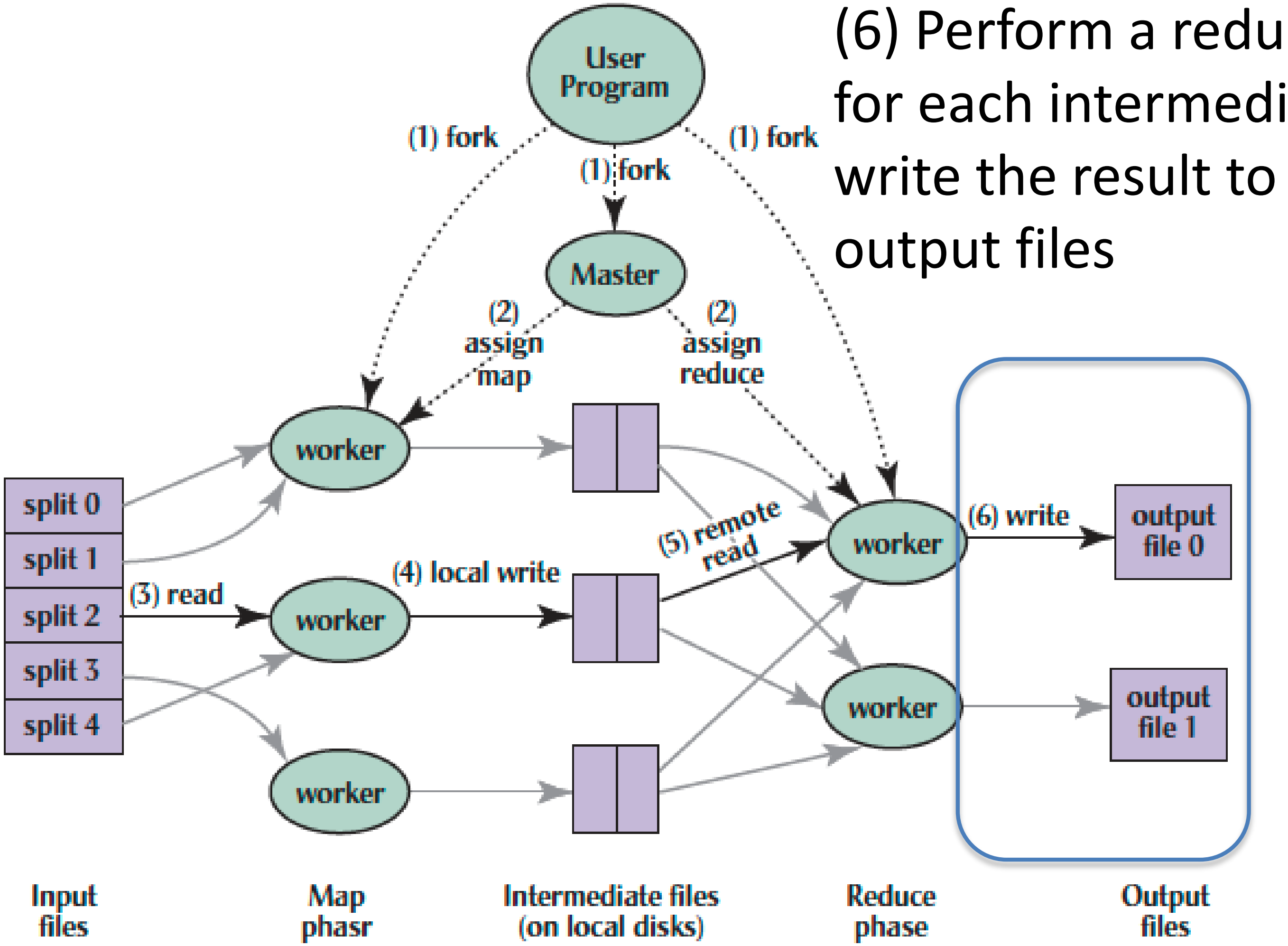
# MapReduce Execution



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

# MapReduce Execution

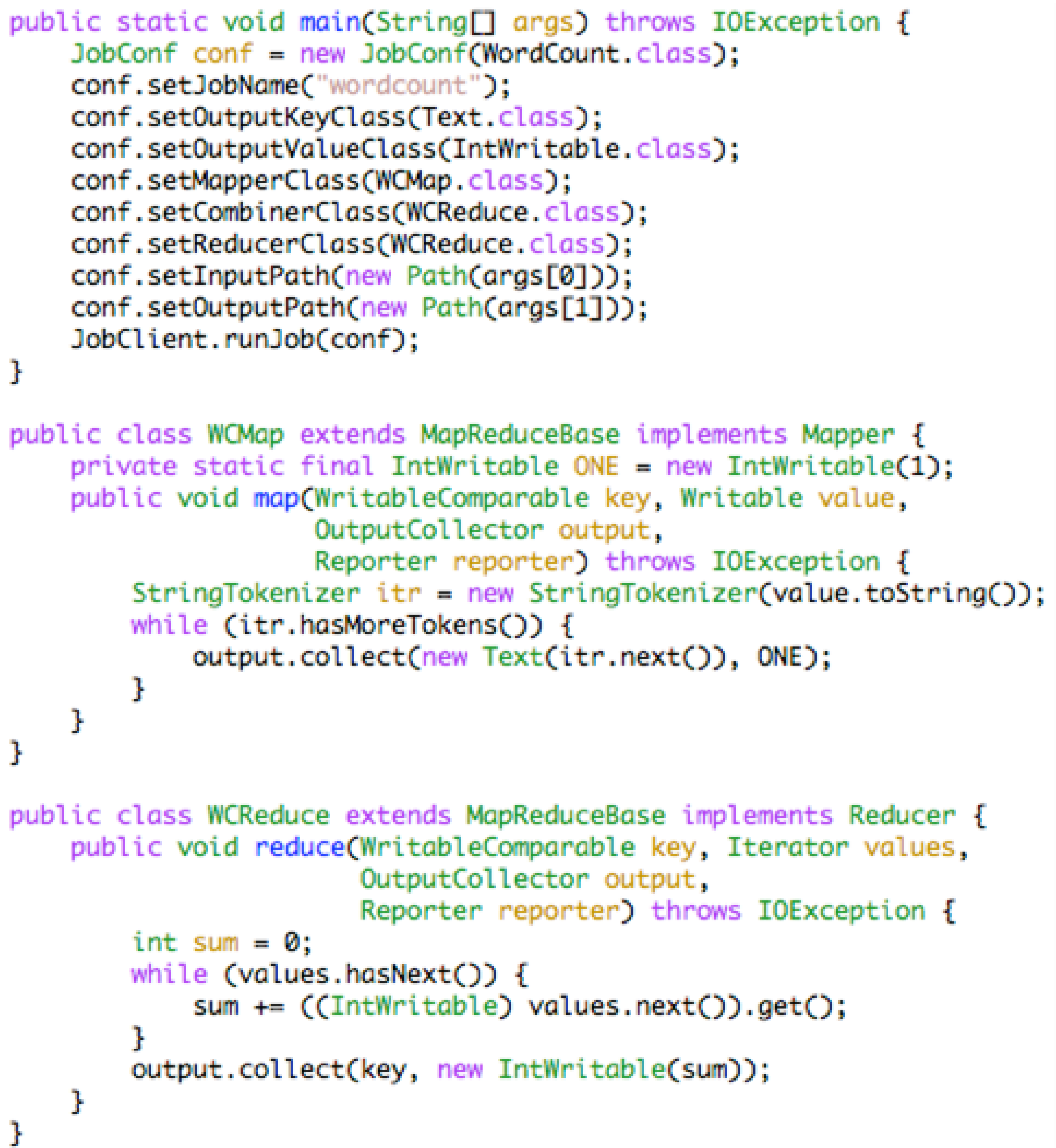
(6) Perform a reduce task for each intermediate key, write the result to the output files



# 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