

Distributed Systems

CS6421

Intro to Distributed Systems and the Cloud

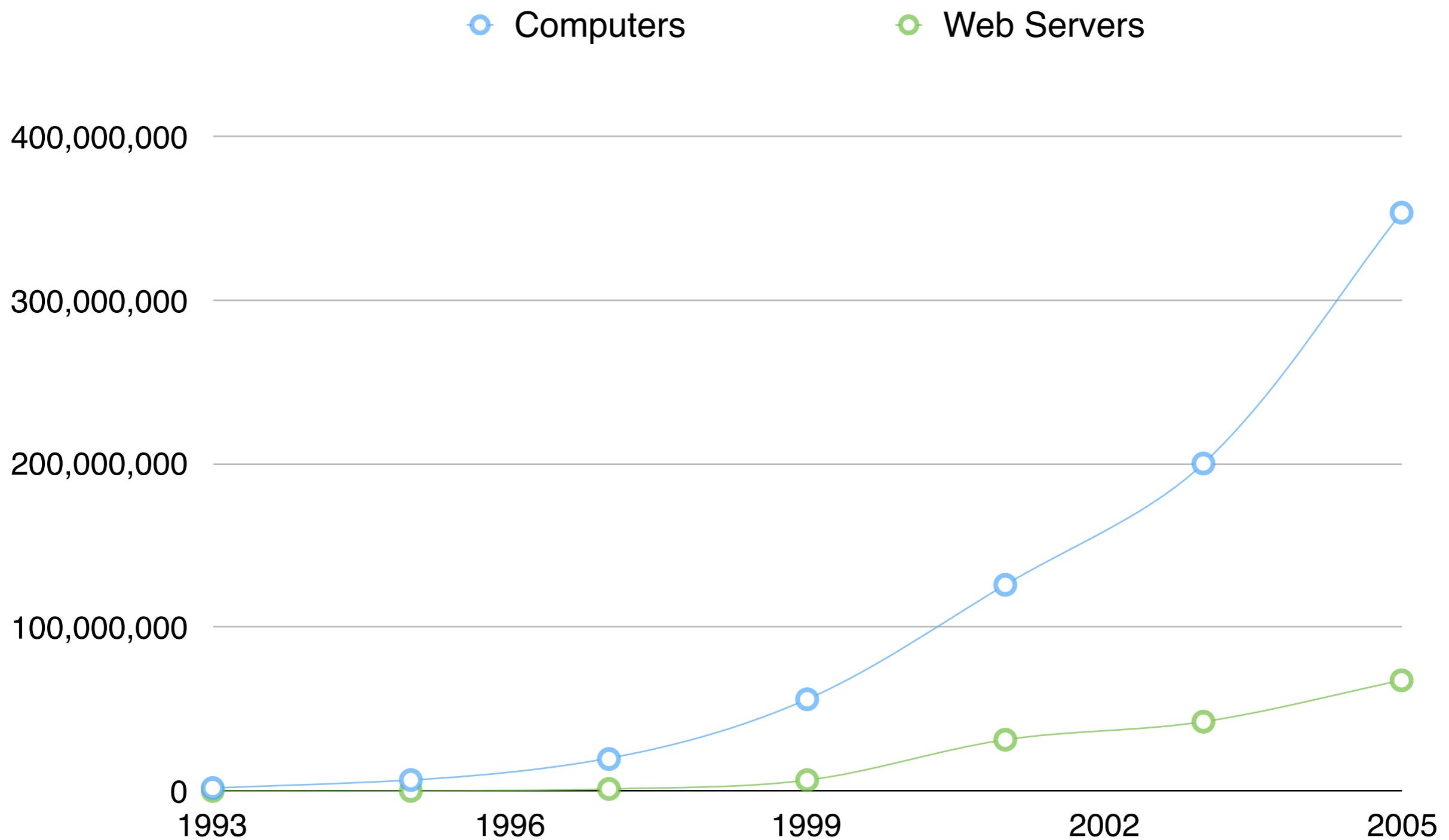
Prof. Tim Wood

Tim Wood

I teach: Software Engineering,
Operating Systems, Sr. Design
I like: distributed systems,
networks, building cool things



Lots of computers!



Lots of problems!

How to interconnect them?

How to write programs for them?

How to secure them?

How to manage them?

How to get good performance out of them?

How to efficiently use resources?

This Course

CSci 6421 Distributed Systems

Prof. Tim Wood - timwood@gwu.edu

Prerequisites:

- CSci 6212 Algorithms (or undergrad algorithms course)
- an undergraduate operating systems course
- strong programming abilities, comfort with multiple languages

<https://gwdistsys18.github.io/>

Be ready to code

We will be writing code both in class and for your homework assignments

- This course will be a lot of work!
- You only learn by doing!

You should be comfortable in at least one of these programming languages:

- C, Python, or Java
- and you should be ready and eager to learn the others!
- and you must understand the basics of the Linux command line

Be ready to participate

This class will be **active** and **interactive**

- I will not be lecturing for very long each period

You **must** participate in class

- Ask questions, answer questions
- Do the in-class exercises
- Even if you don't speak english well, that's fine. I will be patient!

You **may not** use your laptop except for times when I tell you to

- If a slide has a blue bar at the bottom, you may not use it
- If a slide has a green bar, you may use your laptop



Semester Outline

- Topic 1 Intro to Distributed Systems and the Cloud
- Topic 2 Network Programming (Sockets)
- Topic 3 Servers (EC2, VMs, Containers)
- Topic 4 Storage (S3, EBS)
- Topic 5 Networks (SDN, NFV)
- Topic 6 Scalable Web Services (nginx, memcache, RDS)
- Topic 9 Edge and Serverless Computing (AWS Lambda)
- Topic 7 Batched Big Data (Hadoop)
- Topic 8 Streaming Big Data (Storm)
- Topic 10 Fault Tolerance, Ordering, and Consistency (DynamoDB)

(Not necessarily in this order)

Course Outcomes

You will learn to:

Use important technologies common in today's distributed systems

- Hadoop, Amazon's cloud, etc.

Design coordination protocols that are efficient and correct despite network delays and failures

Implement and evaluate the performance of these systems

What *is* a cloud?

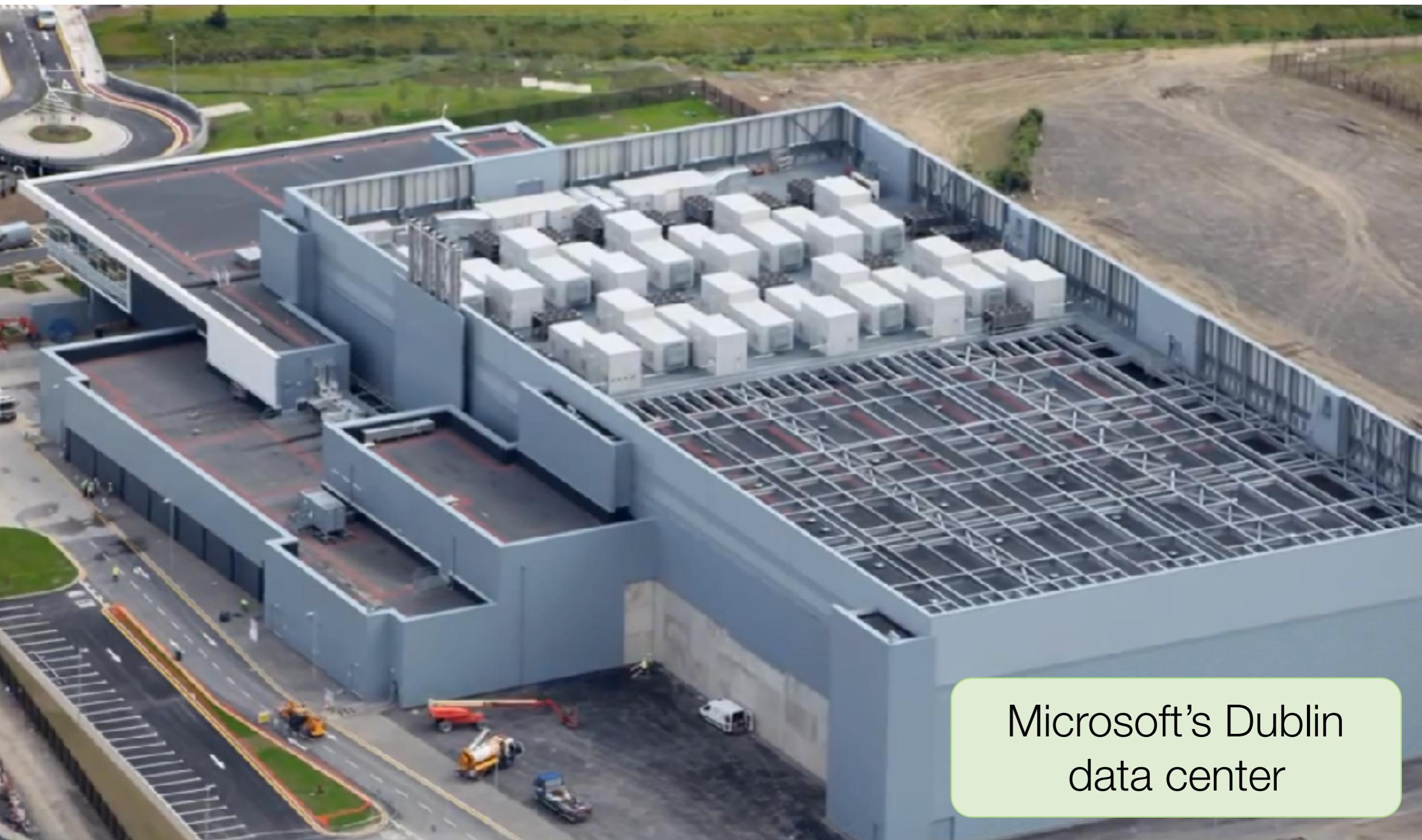
<spoiler alert>

It's not in the sky

it's not made of water droplets

</spoiler alert>

Some big buildings...



Microsoft's Dublin
data center

...and servers...

Giant warehouses

- The size of 10 football fields
- 10s of thousands of servers
- Petabytes of storage



...interconnected...

Level(3)SM
COMMUNICATIONS



...around the world...



Undersea Cables

- Connect all continents except Antarctica
- First deployed in 1850s



<http://www.cyprusupdates.com>

...that break a lot.



Truck crash shuts down Amazon data center (May 2010)

Lightning causes Amazon outages (2009 and 2011)



Comcast down after hunter shoots cable (2008)



Anchor hits underwater Internet cable (Feb 2012)

Or if you're really unlucky...

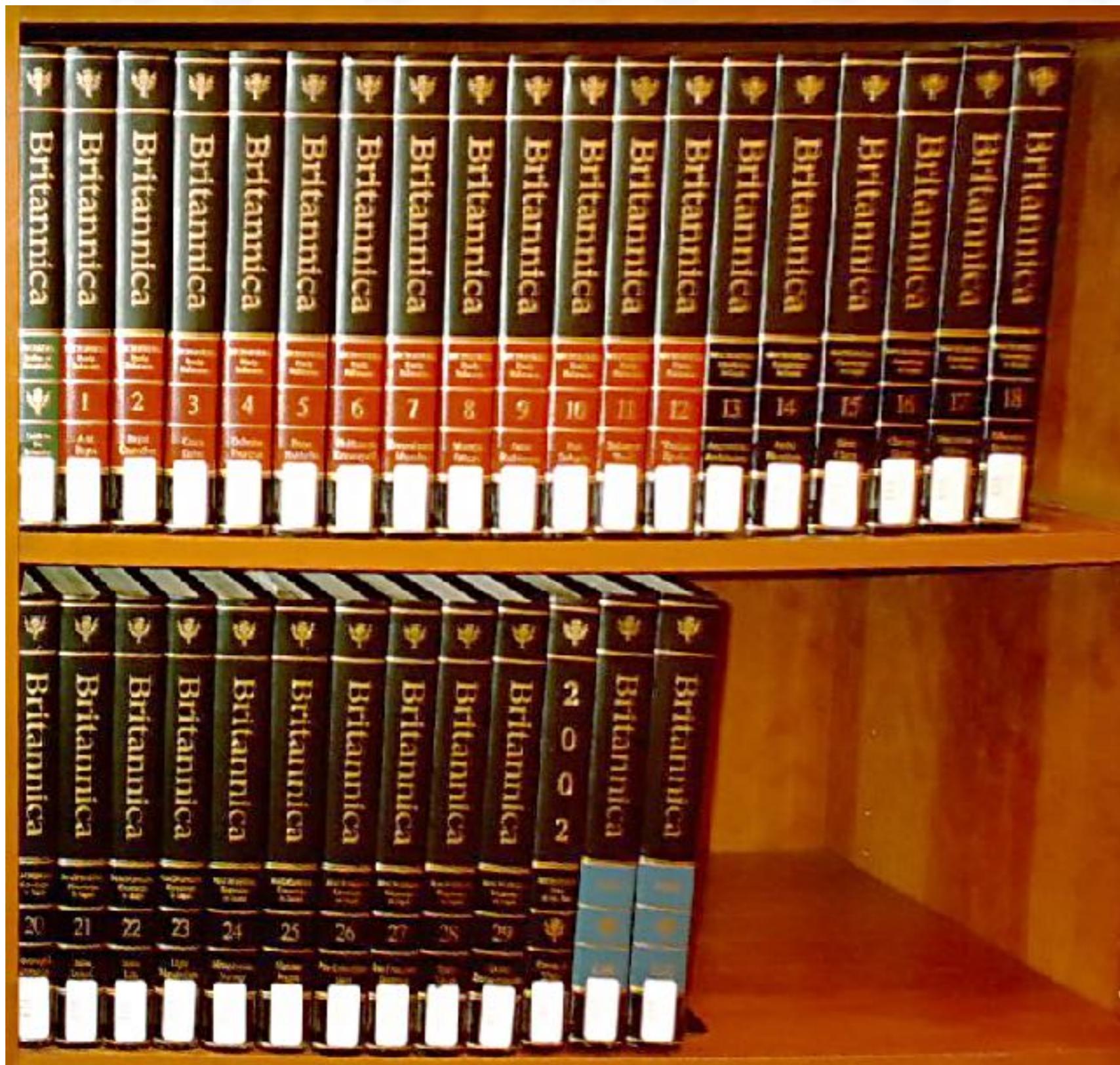


vs



Why do we need all of
this physical
infrastructure?

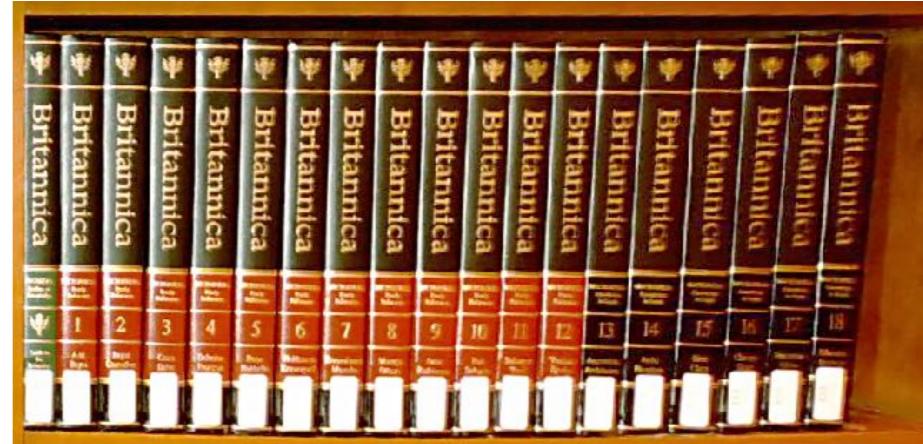
What is this???



Encyclopedias

Encyclopedia Britannica

- 40,000+ articles
- 32 hard bound volumes (32,640 pages)



Microsoft Encarta

- 60,000+ articles
- 1 CD-ROM (**700 MB**)



Wikipedia

- 5,512,202 articles (in English)
- More than **5 TB** of text (about 7,500 CDs)



Mega whats?

700MB vs 5TB

Mega	Million	$1024 \times 1024 =$ $\sim 1,000,000$
Giga	Billion	$1024 \times 1024 \times 1024 =$ $\sim 1,000,000,000$
Tera	Trillion	$1024 \times 1024 \times 1024 \times 1024 =$ $\sim 1,000,000,000,000$

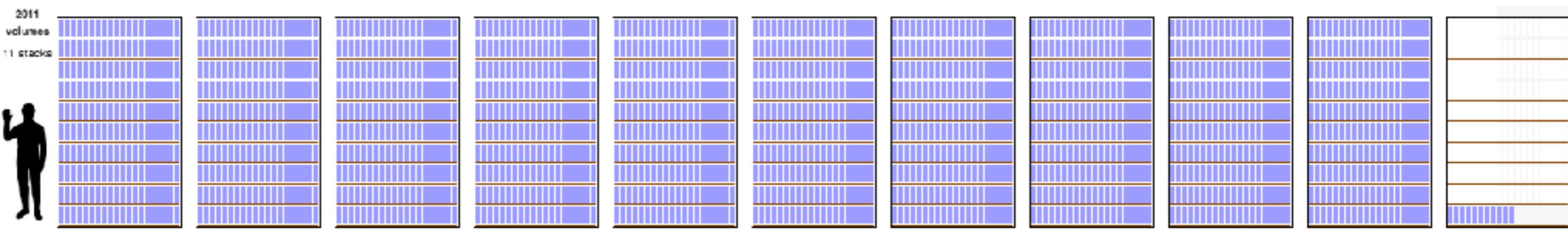
200 songs vs 1.4 million songs

Encyclopedias

Wikipedia... in print

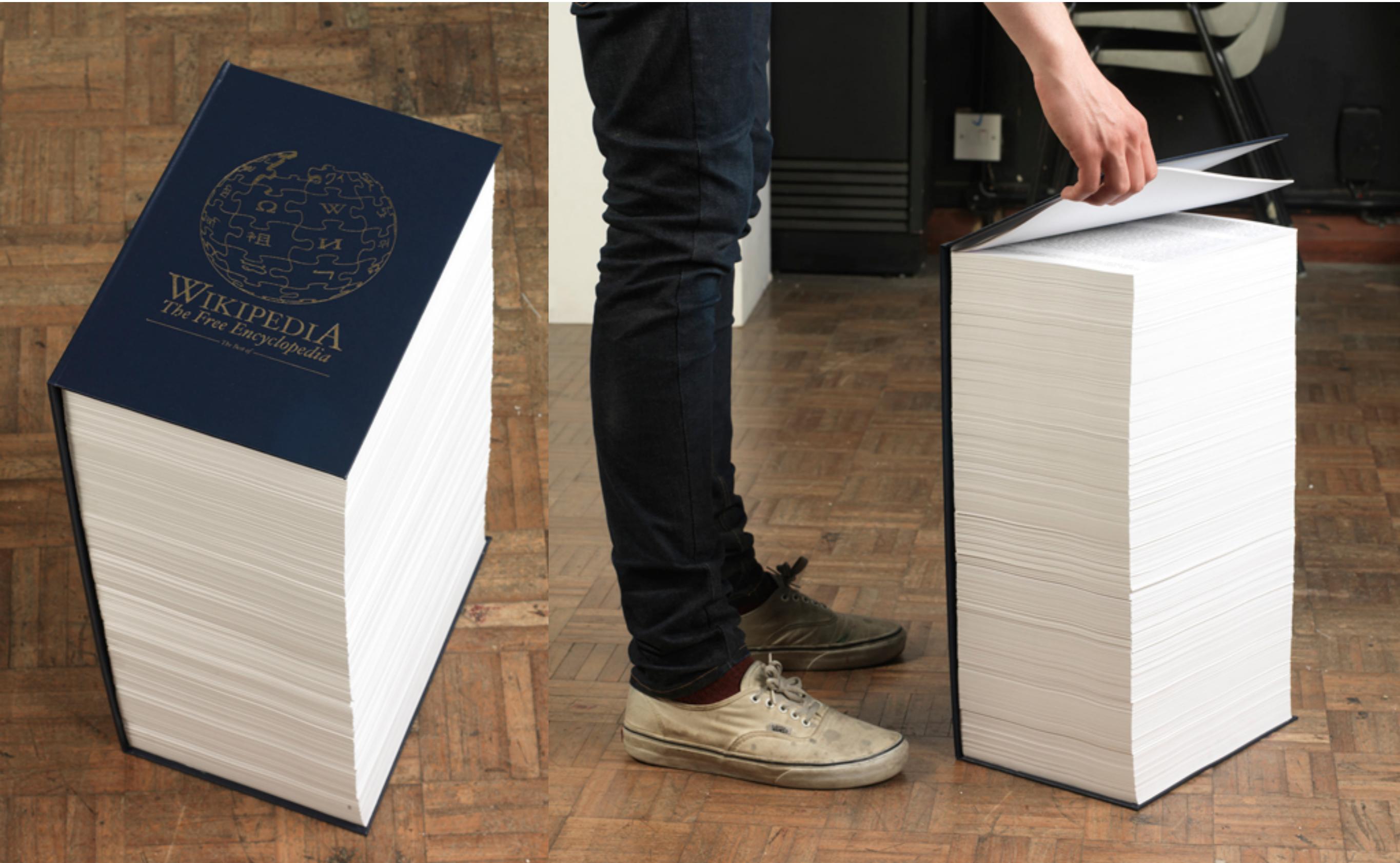
- ~~1,763 volumes~~
- (no, this does not exist)

Now grown to
2,696 volumes!



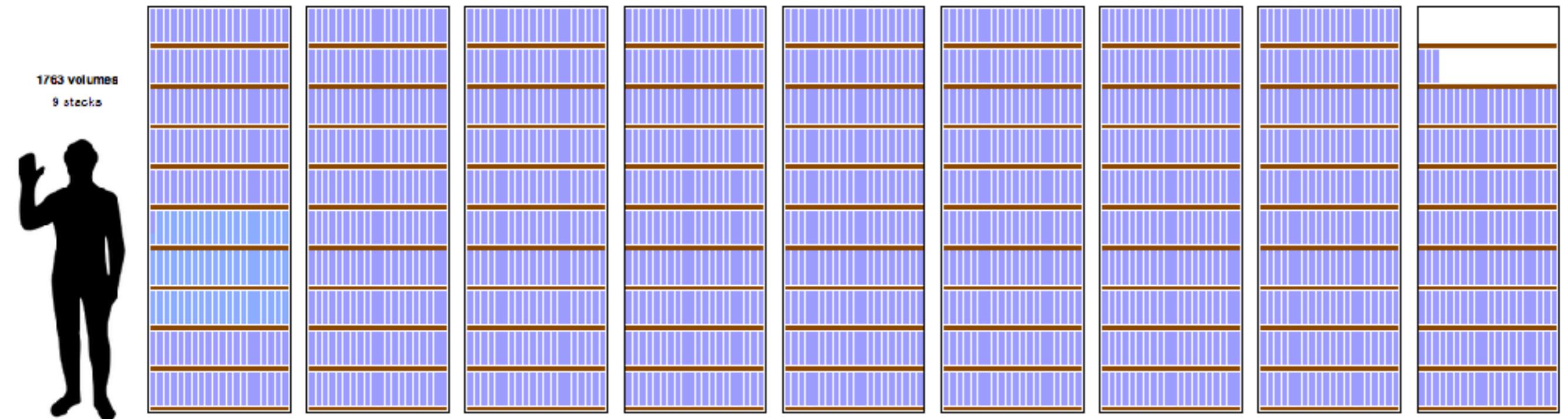
http://en.wikipedia.org/wiki/Wikipedia:Size_in_volumes

0.01% of Wikipedia



Big Data in Perspective

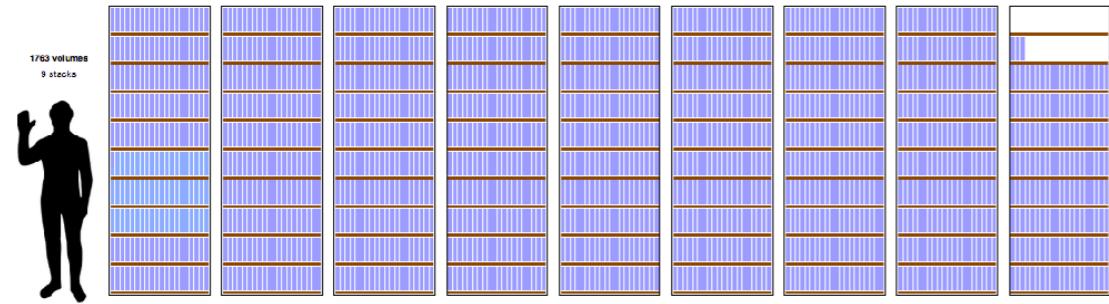
Wikipedia - 5TB of text



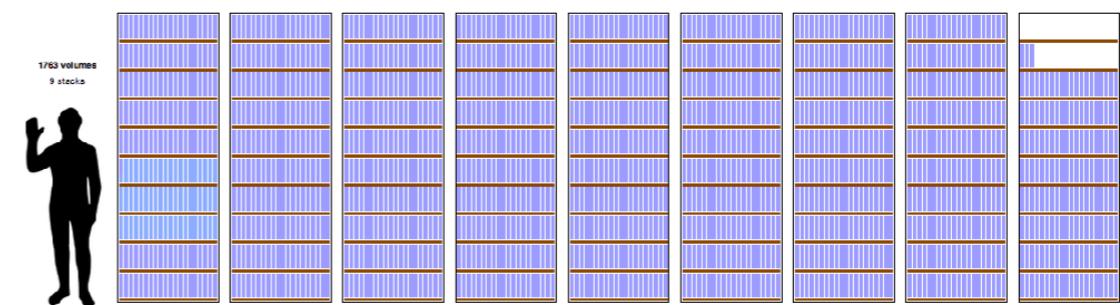
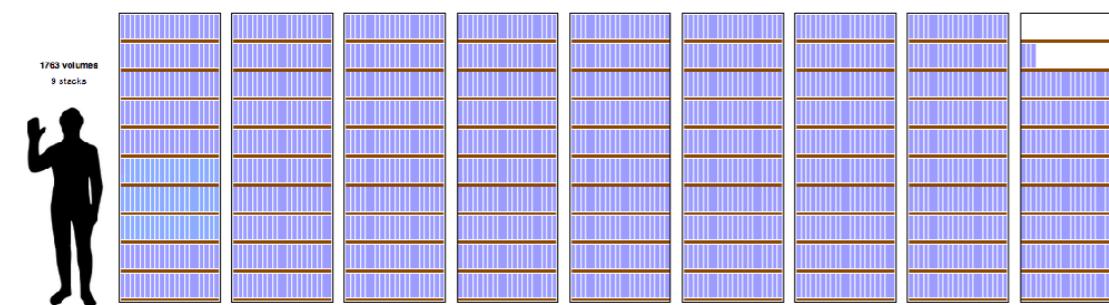
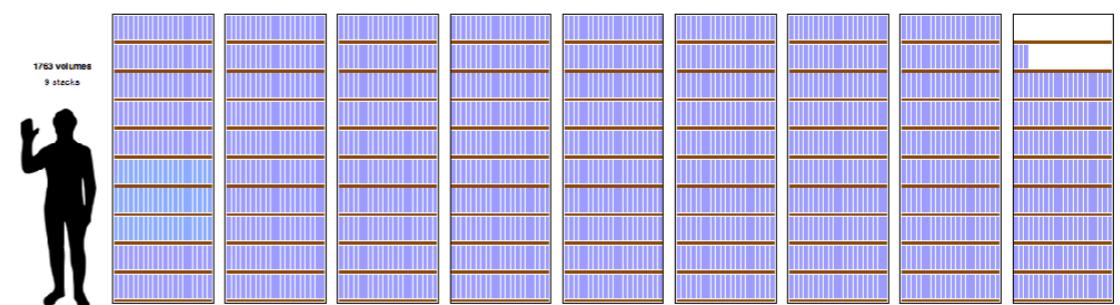
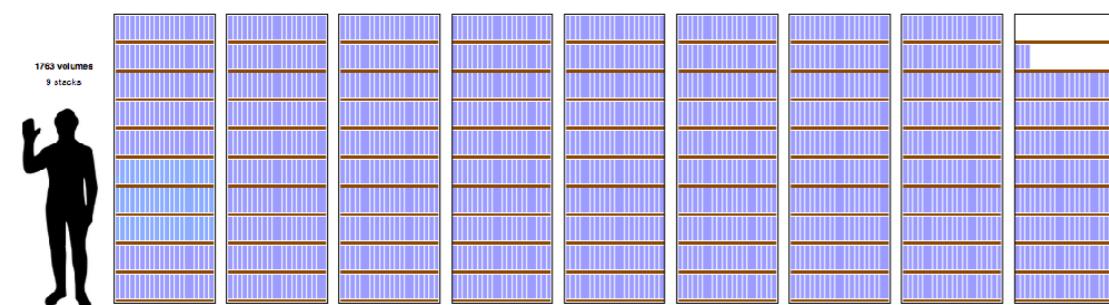
Facebook - ???

Big Data in Perspective

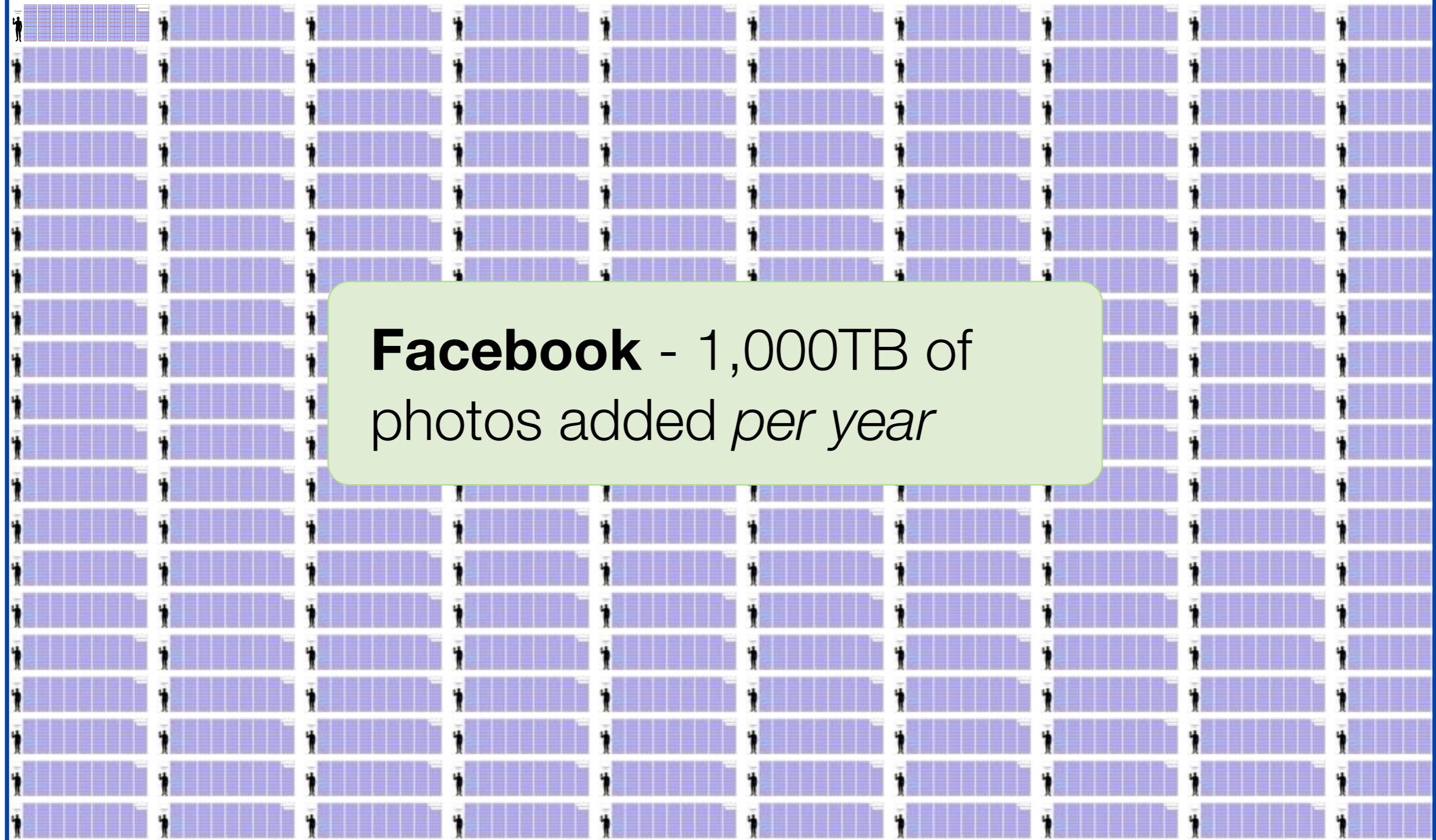
Wikipedia - 5TB of text



Facebook - 20TB of photos added each week

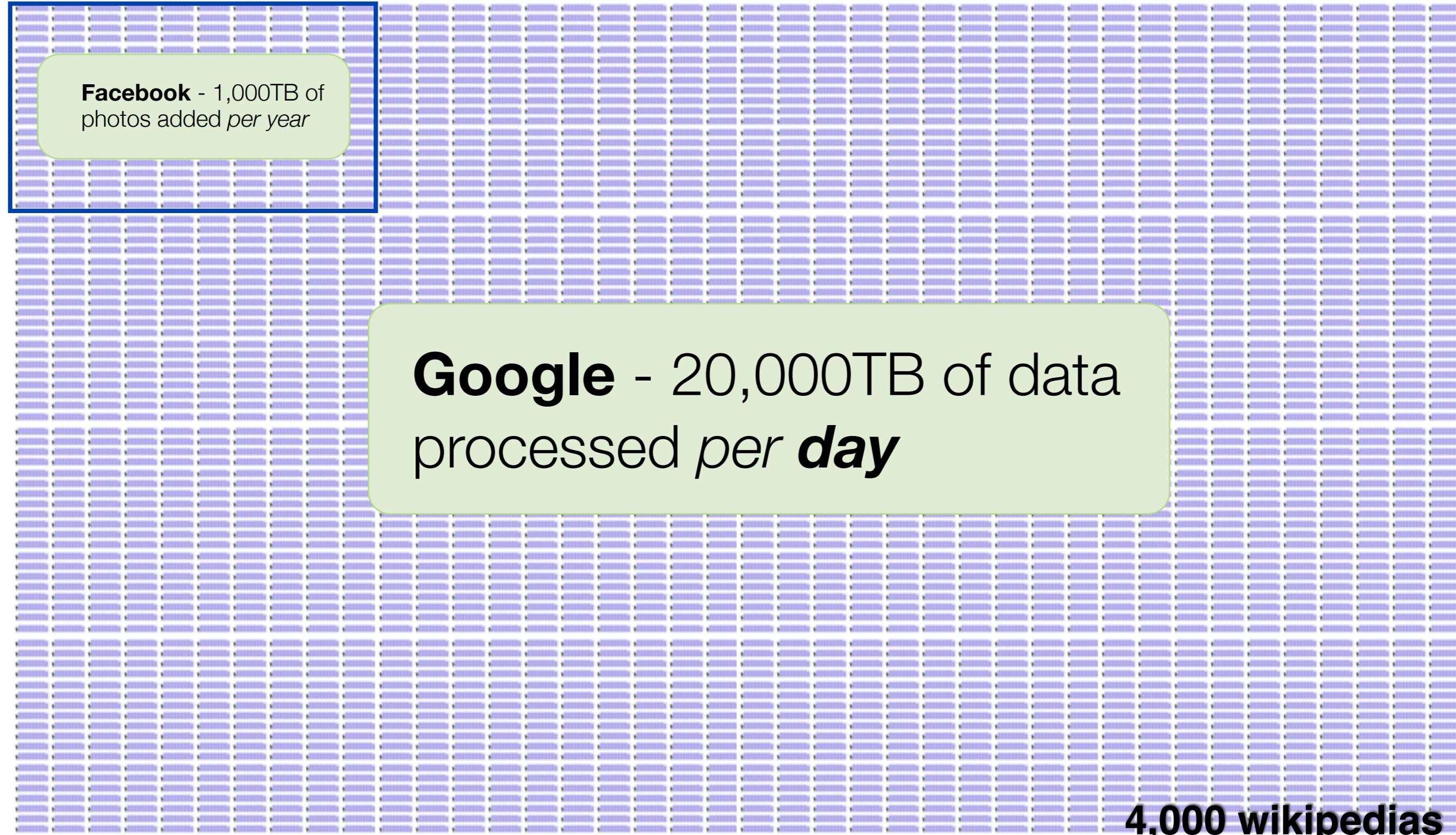


Big Data in Perspective



Facebook - 1,000TB of
photos added *per year*

Big Data in Perspective

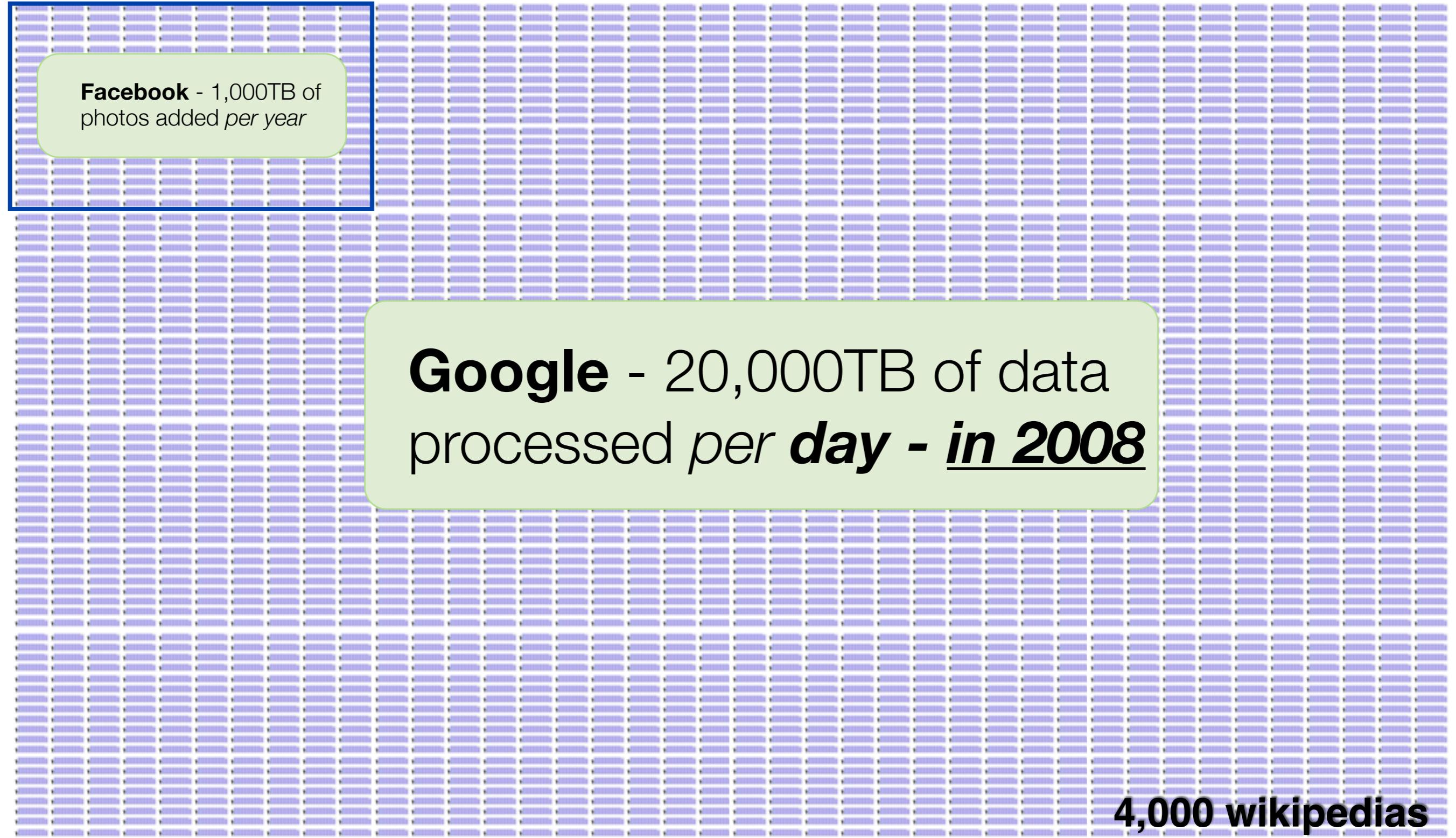


Facebook - 1,000TB of photos added *per year*

Google - 20,000TB of data processed *per day*

4,000 wikipedias

Big Data in Perspective



Facebook - 1,000TB of photos added *per year*

Google - 20,000TB of data processed *per day - in 2008*

4,000 wikipedias

How can google
process *so much*
information *so*
quickly?

Processing Data Quickly

$$1. \quad 3 + 6 = ?$$

Buy a **faster** computer

Buy **another** computer

Processing Data in PARALLEL

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.
- 10.
- 11.
- 12.

Let's try it at scale

I have lots of questions I need answered... help me out!

18 questions per page, 40 pages... how long should it take?

How could we do better?

What problems did we hit?

How could we optimize the process?

What things can't we prevent?

How do we solve problems at large scale?



Distributed Systems

Definition?

Examples?

???

Distributed Systems

“A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages.”

Does this cover all of our examples?

Is it too broad? Too narrow?

How to count words?

How would you count the frequency of each word in a large collection of text documents?

- 100M documents are stored in a distributed storage system
- You have 50 worker nodes that store and process data

Input: lots of text documents

Desired output:

is	54621
and	34213
are	30931
cat	20021

How should we do this at scale?

A Real Distributed System

Let's consider Map Reduce / Hadoop

Data analytic platform originally proposed by Google

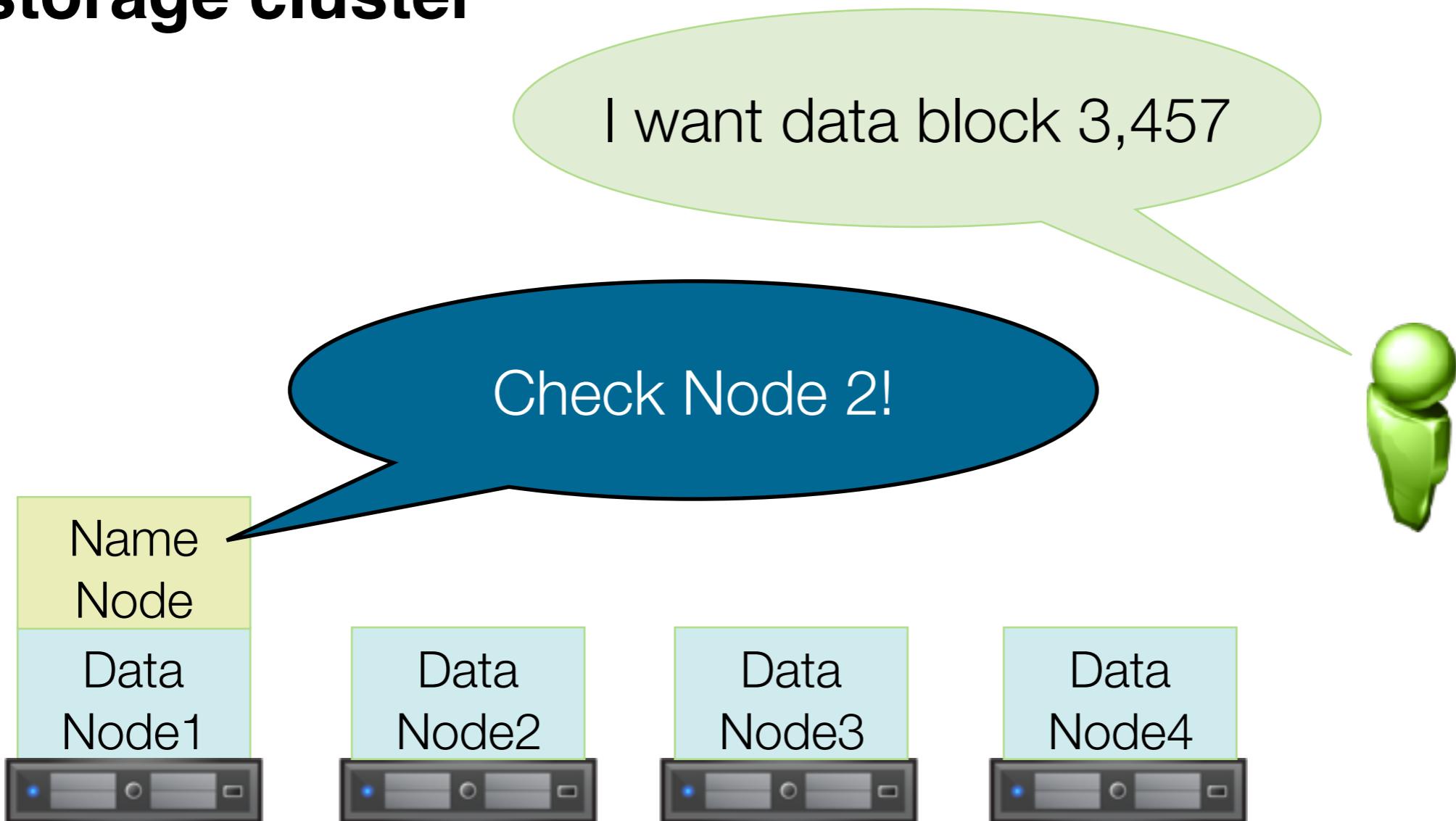
Hadoop is an open source version developed by Yahoo and others

Goal: Make it easy to use a cluster for large scale data processing tasks

- Parsing through web documents to determine content
- Analyzing genetic data to identify diseases

It starts with data

- 1) HDFS provides an abstract interface to a storage cluster



Data is replicated 3 times, and accesses are sent to nearby replicas

Hadoop programs

2) A user writes a program that takes some input data and processes it in stages:

Map the input data to a set of keys and values

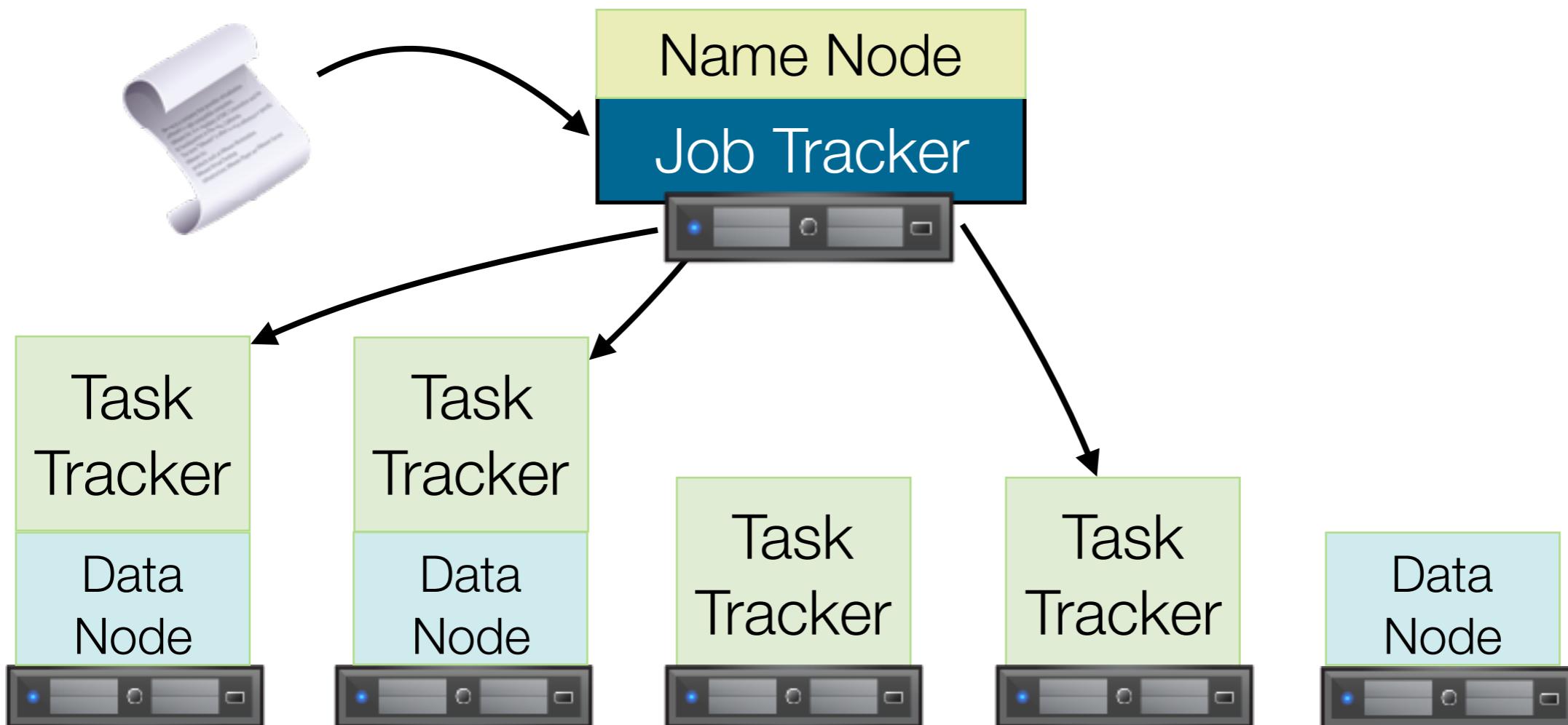
Shuffle the data so all identical output keys are processed by the same server

Reduce all output keys to a final result

(more details later)

Job scheduling

3) User submits their program as a job to the JobTracker

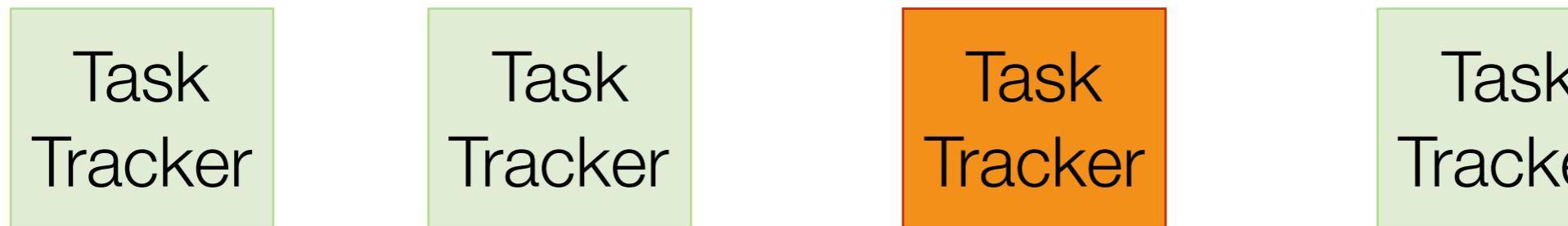


4) The Job Tracker splits job into small tasks and distributes them across Task Trackers

Prefers to place a task next to the data node with its data

Failure handling

- 5) If a task fails or is too slow, the Job Tracker restarts it on another task tracker.



Failures and stragglers are masked

Hadoop Components

1 JobTracker

- Centralized controller
- Assigns tasks to workers, monitors for failures

Lots of TaskTrackers

- Worker nodes
- Execute code and return results to Job Tracker

1 NameNode

- Centralized storage controller
- Determines where data is placed

Lots of DataNodes

- Storage nodes
- Hold data

One server may have multiple roles

How to count words?

How would you count the frequency of each word in a large collection of text documents?

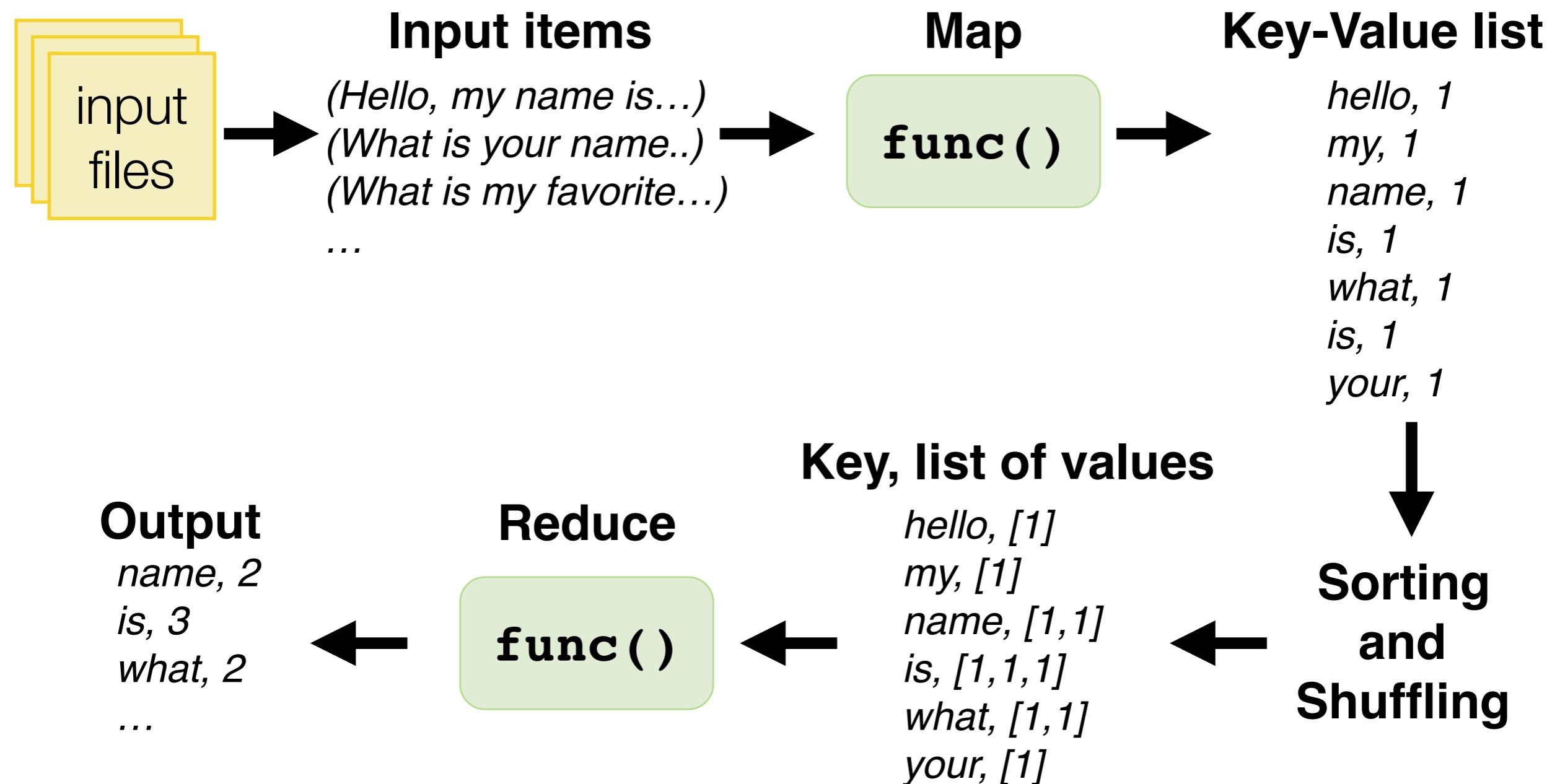
- 100M documents are stored in a distributed storage system
- You have 50 worker nodes that store and process data

Input: lots of text documents

Desired output:

is	54621
and	34213
are	30931
cat	20021

Map Reduce Flow



Map & Reduce

Map Phase

- input: data element
- Convert input data into an intermediate result
- All map functions can be called independently in parallel
- output: {list of keys and values}

Shuffle and partition

- Sort all outputs by key and combine values into a list
- Partition keys to create new Reduce tasks
- output: Key, {list of values}

Reduce Phase

- input: Key, {list of values}
- Combine the list of values for each key to produce an output

Map/Reduce Hadoop

What distributed systems concepts did we see?

Map/Reduce Hadoop

What distributed systems concepts did we see?

Replication for fault tolerance and scalability

Data partitioning

Scheduling

Automated resource management

Abstraction layers

Common Problems

What problems arise in the Hadoop design?

- Either problems that it solves or fails to solve

???

Common Problems

Communication protocols

Resource management

Security

Performance

Fault tolerance

Scalability

Which are **Systems** challenges and which are **Algorithmic** challenges?

Systems and Algorithms

This course is about **BOTH**

Systems:

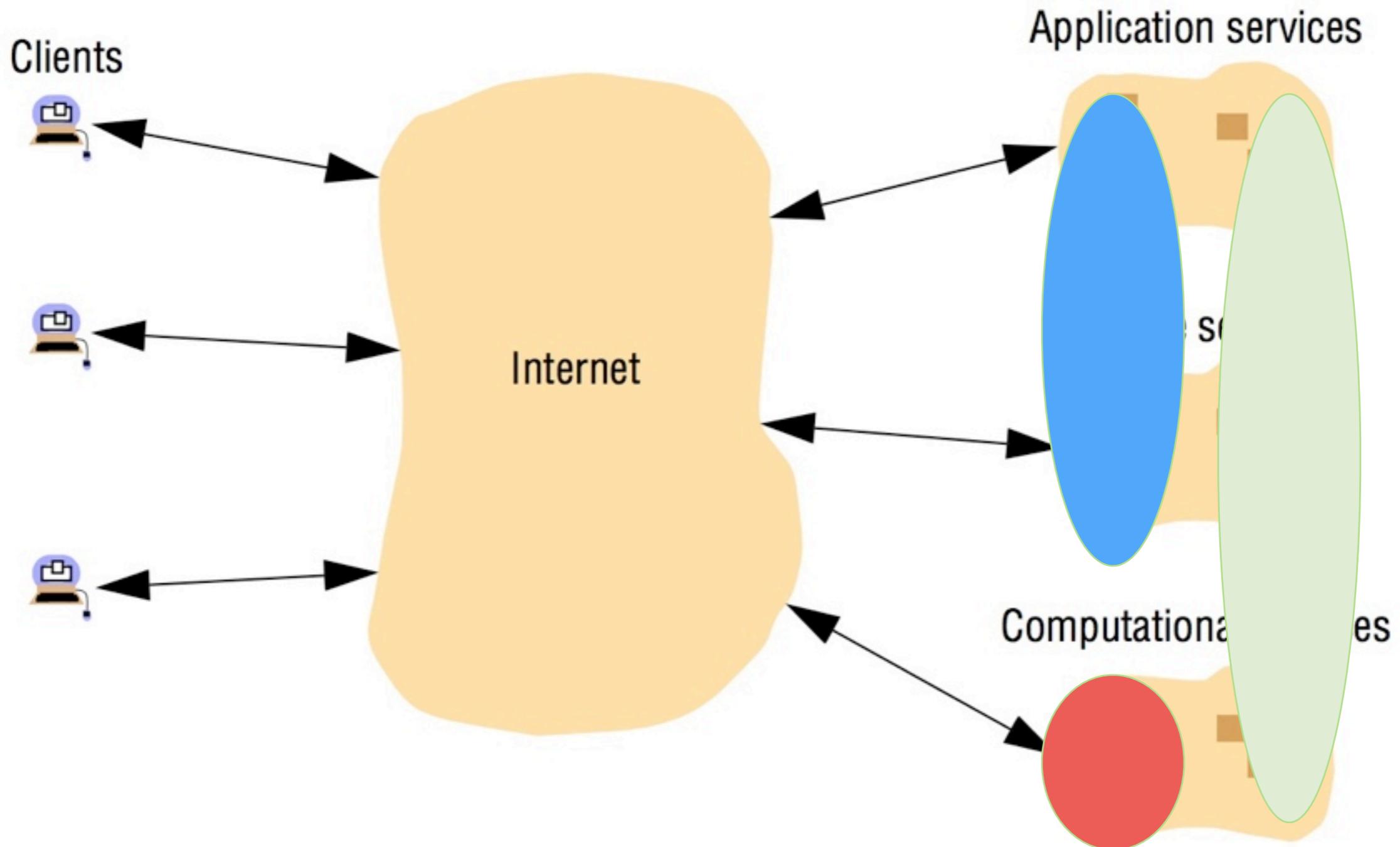
- How to **implement efficient** distributed applications and the **mechanisms** needed for them to be scalable, secure, and reliable

Algorithms:

- How to **design intelligent** distributed applications and the **policies** needed for them to be scalable, secure, and reliable

Distributed System Architectures and Principles

The Cloud



Bit Torrent

High Performance Computing

Challenges

Heterogeneity

Openness

Security

Failure Handling

Concurrency

Quality of Service

Scalability

Transparency

Scalability

What are some **principles** that help with scalability?

What are some **mechanisms** that help with scalability?

Scalability

Principles:

- No single machine has all state
- Make decisions based on local or partial information
- No single point of failure
- No global clock

Techniques:

- Asynchronous communication
- Distribution / Partitioning
- Caching
- Replication

Are these just for **performance**?

Transparency

What is transparency? Why is it useful?

How do some distributed systems provide transparency?

Transparency Examples

Transparency is provided by abstractions that hide the complexity of distributed systems

Dropbox: transparently replicates your files across several computers

MapReduce / Hadoop: abstracts away the complexity of assigning processing tasks to heterogeneous servers

Autonomous middleware: automatically allocates resources to applications / VMs in need