# Introduction

DSCI 551

Wensheng Wu

# Logistics

- Instructor email: wenshenw@usc.edu
  - please indicate section: 551 in the subject line.

- Class meeting times:
  - see syllabus

- Office hours:
  - After class
  - 10:30-11:30am Thursday, by appointment, online

# Logistics

- TAs & office hours
  - Check the announcements
- Class materials
  - Posted on course web site
- AWS EC2 (elastic compute cloud)

# Piazza

- Discussion forums
  - You may post general and homework questions
  - Do not post solutions
  - Please actively participate in helping others!
  - Do not abuse forum (an academic misconduct!)
- Check frequently for updates
- You should have received invitation to sign up

# Prerequisites

- Programming skills:
  - Python (homework, Spark), Java (e.g., for Hadoop only)
    - Scala
- Unix-like environment & shell commands (ls?)
  - E.g., Amazon EC2
- Basic knowledge of algorithms and data structures
  - Sorting, hashing, etc. (CS 570)//merge sort?
  - h(k) => if k is even, send (k,v) to R0; otherwise, send to R1
  - 3%2 = 1
  - 2%2 = 0
  - I/O
    - 1TB (data on SSD) 1GB main memory) => runs
    - I/O
- Basic probability and statistics

# Textbooks

- Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*, 2015 (selected chapters only). Available free at: http://pages.cs.wisc.edu/~remzi/OSTEP/

- Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. Database Systems: The Complete Book (Second Edition), Prentice Hall, 2009. (selected chapters only)
  - http://infolab.stanford.edu/~ullman/dscb.html

- See four more books in syllabus

# Additional readings

- Links can be found in Syllabus
  - Check out the schedule

# Grading structure

- Homework 20%
- Midterm 1 15%
- Midterm 2 15%
- Final/comprehensive 25%
- Lab tasks 5%
- Group project 20%

# Grading scale

- [94, 100] = A
- [90, 94) = A-
- [87, 90) = B+
- [83, 87) = B
- [80, 83) = B-
- [77, 80) = C+
- [73, 77) = C
- … (see Syllabus for complete breakdown)

# Lab tasks

- Four tasks:
  - EC2, HDFS, MongoDB, DynamoDB

# Exams

- Closed-notes & book
- Midterms:
  - See syllabus for time
- Final:
  - See syllabus for time

# Calculator

- Bring one to the tests
- If calculator is needed, we will either announce or state it on the tests
- Otherwise, no electronic devices are allowed

# Course project

- Details to be posted

- Done in phases
  - Proposal
  - Midterm report
  - Final report

# Late Policy

- No LATE submissions will be accepted

- Make up for tests are permitted only when
  - You have an emergency, typically medical
  - Let me know at least two weeks in advance
  - You may be required to contact Campus Support and Intervention office for verification of emergency

# Grading Corrections

- All coursework's grades are final one week after grades are posted
- Final exam grades (& all grades) are final after final exam regrading deadline (to be announced right before/after final)
- Please submit reasonable regrading requests
  - Irrational requests (e.g., simply asking for more points or special treatments) may result in reduction of your grades

# Academic Integrity

- **Cheating will NOT be tolerated**

- **All parties involved will receive a grade of F for the course and be reported to SJACS** WITHOUT EXCEPTION
  - USC Student Judicial Affairs and Community Standards

# Now, movie time ☺

- Explain big data:
  - https://www.youtube.com/watch?v=7D1CQ_LOizA

- Questions:

volume, velocity, variety

  - Where does big data come from?
  - What characteristics doe it have? 3Vs?
  - What big data technologies were mentioned?
    - Hadoop: HDFS and MapReduce

# Variety


TRADITIONAL DATA
* Documents
* Finances
* Stock records
* Personnel files
BIG DATA
* Photographs
* Audio & video
* 3D models
* Simulations
* Location data

# Internet Traffic in 2012

- 4.8 zettabyte = 4.8 billion terabytes

- Zettabyte (1000 exabytes)
- Exabyte
- Petabyte
- Terabyte = 2^40 (storage)
  - 1TB = 1024 (2^10) GB
  - 2^2 = 4, 2^3 = 8, 2^7 = 128
- Gigabyte = 2^30 (memory)
- Megabyte (128MB, HDFS)
  - 1MB = 2^20 = 2^10 * 2^10
- Kilobyte = 2^10 (1KB) = 1024B // 2^5 = 32

Main memory:
12GB

SSD:
1TB

123 (decimal) = 1 * 10^2 + 2
111 (binary) = 1*2^2 + 1*2^1

111 + 1 (binary) = 1000 = 8 (d
001
==
1000

# Notes

- main memory  volatile
- export data from MySQL into CSV/JSON format
- apache Hive – HQL
- very structured – relations (data in MySQL)
- semi-structured (JSON/XML)
- unstructured (texts) NLP

# Major topics

- Storage systems 
- File systems & file formats
- Database management systems (RDBMS)
  - R = relational
- Big data solution stack

# Storage Systems

- Hard disk
- SSD (Solid state drive)

  4KB = block size for HDD

  128MB = block size in HDFS

  128MB/4KB = 32K

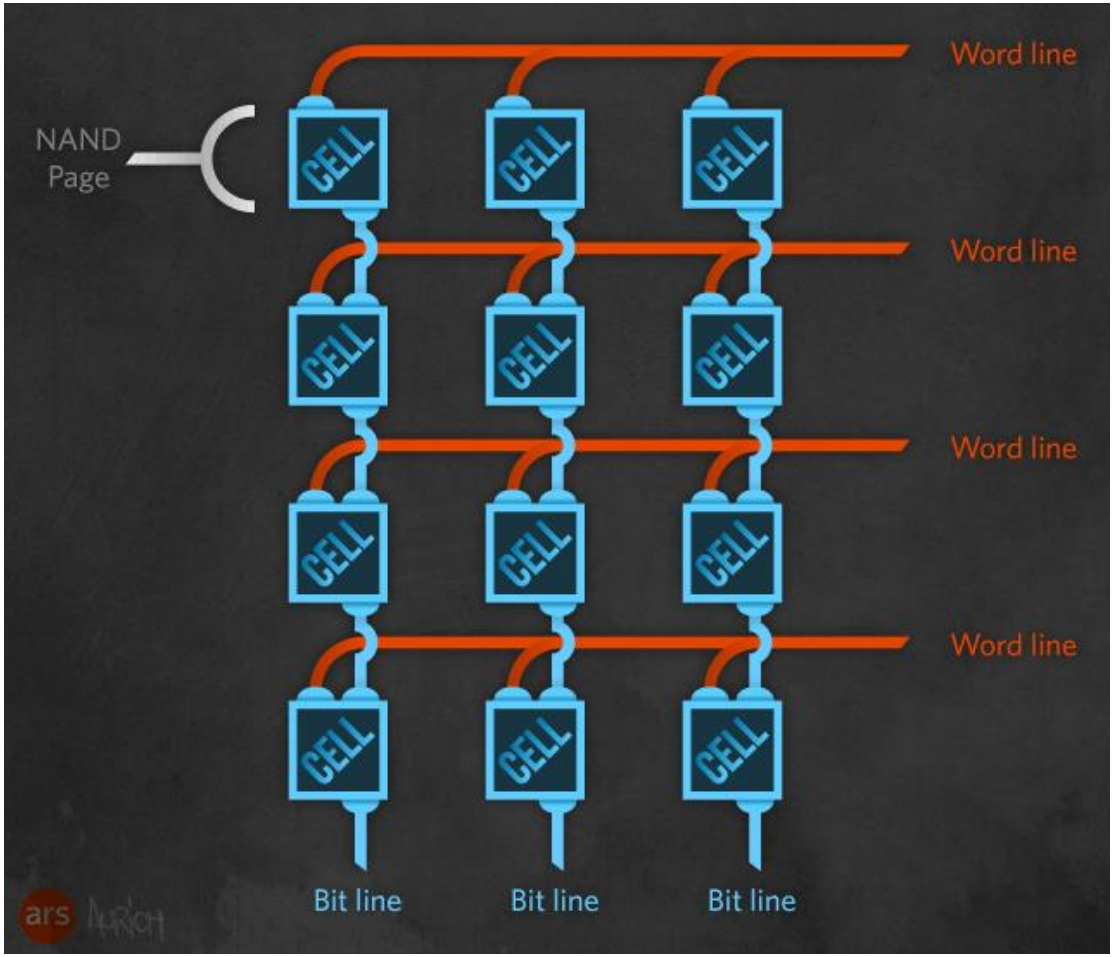

SSD (Solid State Drive)

HDD (Hard Disk Drive)

# Internal of hard disk

# NAND flash

NAND hard drives are a type of solid-state storage that use NAND flash memory, offering faster speeds and greater reliability than traditional spinning disk drives.



# Latencies: read, write, and erase

| | SLC | MLC | TLC | HDD | RAM |
|---|---|---|---|---|---|
| P/E cycles | 100k | 10k | 5k | * | * |
| Bits per cell | 1 | 2 | 3 | * | * |
| Seek latency (µs) | * | * | * | 9000 | * |
| Read latency (µs) | 25 | 50 | 100 | 2000-7000 | 0.04-0.1 |
| Write latency (µs) | 250 | 900 | 1500 | 2000-7000 | 0.04-0.1 |
| Erase latency (µs) | 1500 | 3000 | 5000 | * | * |
| *Notes* | * metric is not applicable for that type of memory | | | | |
| *Sources* | P/E cycles [20]<br>SLC/MLC latencies [1]<br>TLC latencies [23]<br>Hard disk drive latencies [18, 19, 25]<br>RAM latencies [30, 52]<br>L1 and L2 cache latencies [52] | | | | |

# Major topics

- Storage systems
- File systems & file formats 
- Database management systems
- Big data solution stack

# File Systems

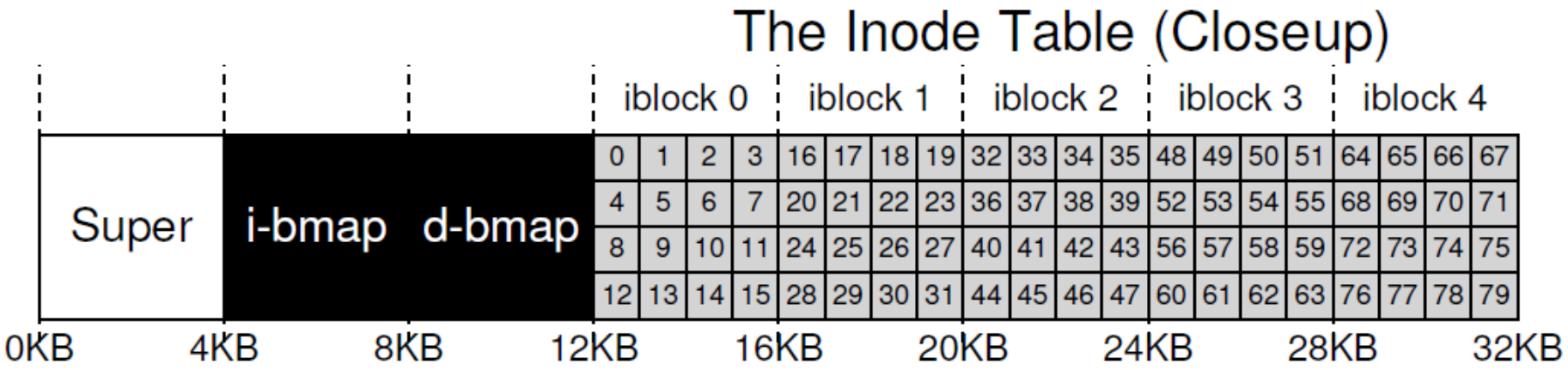
- Standalone
  - Single machine
- Distributed (e.g., Hadoop)
  - A number of data servers

# Standalone file systems

- Data structures
  - Data blocks
  - Metadata blocks (Inodes)
  - Bitmap blocks (for space allocation)
- Access paths
  - Read a file
  - Write a file

# Inode (index node)

- Each is identified by a number
  - Low-level number of file name: inumber
- Can figure out location of inode from inumber



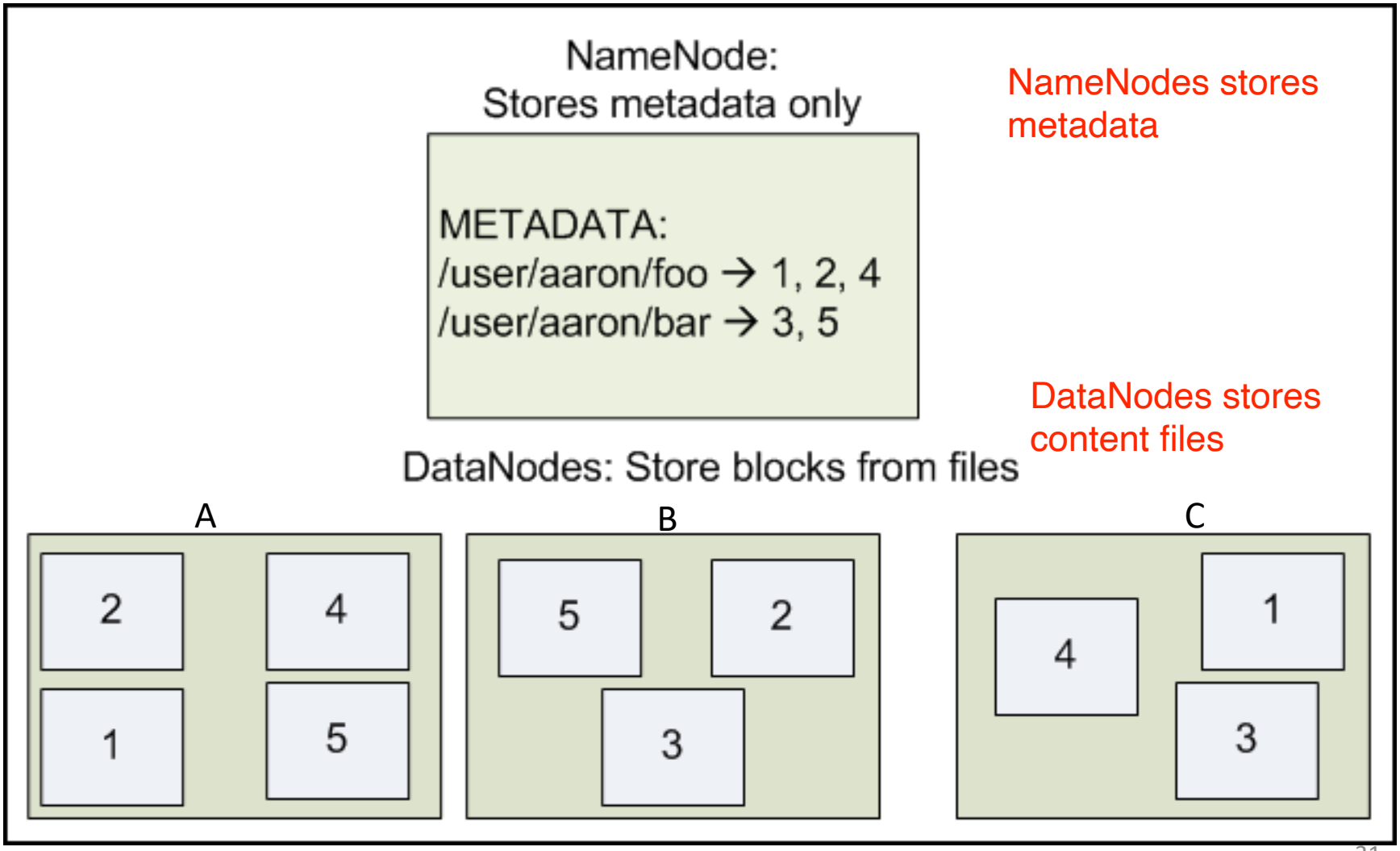
# Distributed file systems

Hadoop Distributed File System

- Hadoop HDFS (after GFS)
  - Data are distributed among data nodes
- Replication
  - Automatic creation of replica (typically 2 or 3 copies/replica of data)
- Fault-tolerant
  - Automatic recovery from node failure

# HDFS architecture

It's important for file systems to know the location of block that stores the content of the file



# File system image in namenode

```
</ErasureCodingSection>
▼<INodeSection>
  <lastInodeId>16422</lastInodeId>
  <numInodes>38</numInodes>
  ▼<inode>
    <id>16385</id>
    <type>DIRECTORY</type>
    <name/>
    <mtime>1581231015982</mtime>
    <permission>ec2-user:supergroup:0755</permission>
    <nsquota>9223372036854775807</nsquota>
    <dsquota>-1</dsquota>
  </inode>
  ▼<inode>
    <id>16386</id>
    <type>DIRECTORY</type>
    <name>user</name>
    <mtime>1581231034866</mtime>
    <permission>ec2-user:supergroup:0755</permission>
    <nsquota>-1</nsquota>
    <dsquota>-1</dsquota>
  </inode>
```

# Directory section

```
</SnapshotSection>
▼<INodeDirectorySection>
  ▼<directory>
      <parent>16385</parent>
      <child>16386</child>
    </directory>
  ▼<directory>
      <parent>16386</parent>
      <child>16387</child>
    </directory>
  ▼<directory>
      <parent>16387</parent>
      <child>16390</child>
      <child>16412</child>
      <child>16401</child>
      <child>16391</child>
      <child>16388</child>
    </directory>
  ▼<directory>
      <parent>16388</parent>
      <child>16389</child>
    </directory>
```

# Major topics

- Storage systems
- File systems & file formats 
- Database management systems
- Big data solution stack

# File Formats

- JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

# HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
      Abiteboul, Hull, Vianu
      <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
      Abiteoul, Buneman, Suciu
      <br> Morgan Kaufmann, 1999
```

# XML

```
<bibliography>
      <book>   <title> Foundations… </title>
                  <author> Abiteboul </author>
                  <author> Hull </author>
                  <author> Vianu </author>
                  <publisher> Addison Wesley </publisher>
                  <year> 1995 </year>
      </book>
      …
</bibliography>
```

XML describes the content

# XML usages

- Software configurations files
  - E.g., HDFS
- Android app development
  - Layout resource files, e.g., activity_main.xml
- Java archive (.jar file)
  - Manifest.xml

# Android app resource file

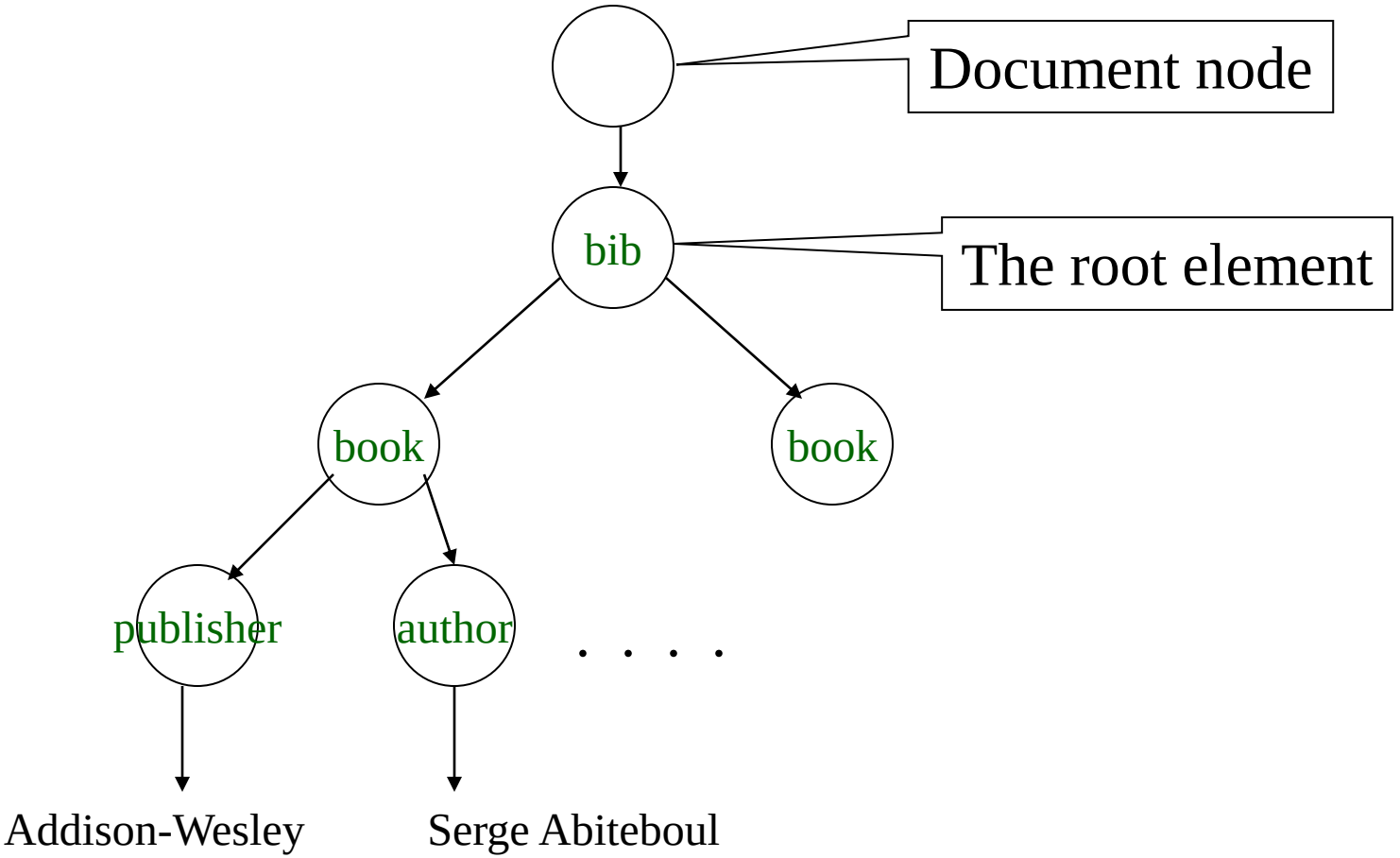
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <android.support.design.widget.TabLayout
        android:id="@+id/tabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <android.support.v4.view.ViewPager
        android:id="@+id/container"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@id/tabs" />
```

# Manifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.google.firebase.quickstart.database">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Firebase Database"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="Firebase Database"
            android:theme="@style/AppTheme" />
        <activity android:name=".NewPostActivity" />
        <activity android:name=".SignInActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
```

```
<bib>
…
<book price="35">
     <publisher>Addison-Wesley</publisher>
      <author>Serge Abiteboul</author>
      <author><first-name>Rick</first-name><last-name>Hull</last-name></author>
      <author age="20">Victor Vianu</author>
      <title>Foundations of Databases</title>
      <year>1995</year>
     <price>38.8</price>
</book>
<book price="55">
      <publisher>Freeman</publisher>
      <author>Jeffrey D. Ullman</author>
      <title>Principles of Database and Knowledge Base Systems</title>
      <year>1998</year>
</book>
…
</bib>
```

# Data Model for XPath



# XPath: Simple Expressions

/bib/book/year

Result: <year> 1995 </year>

<year> 1998 </year>

/bib/paper/year

Result: empty (there were no papers)

# Major topics

- Storage systems
- File systems & file formats
- Database management systems 
- Big data solution stack

# Relational DBMS

- Data models
  - E (entity set) R
  - Relational (redundancy => update anomaly)
- Schema
  - describes the structure of data
  - including constraints

# RDBMS

- Query languages
  - Relational algebra
  - SQL, constraints, views

- Data organization
  - Records and blocks
  - Index structure: B+-tree (external data structure)

B plus tree
B means block

# RDBMS

- Query execution algorithms
  - External sorting
  - One-pass algorithms
  - Nested-loop join, sorting, hashing-based
  - Multiple-pass algorithms

# RDBMS

- Rigid schema
- Strong consistency is the key design goal
  - Never read old data
  - Suitable for mission-critical applications, e.g., banking
- But may suffer from low availability
  - ACID vs CAP

# RDBMS

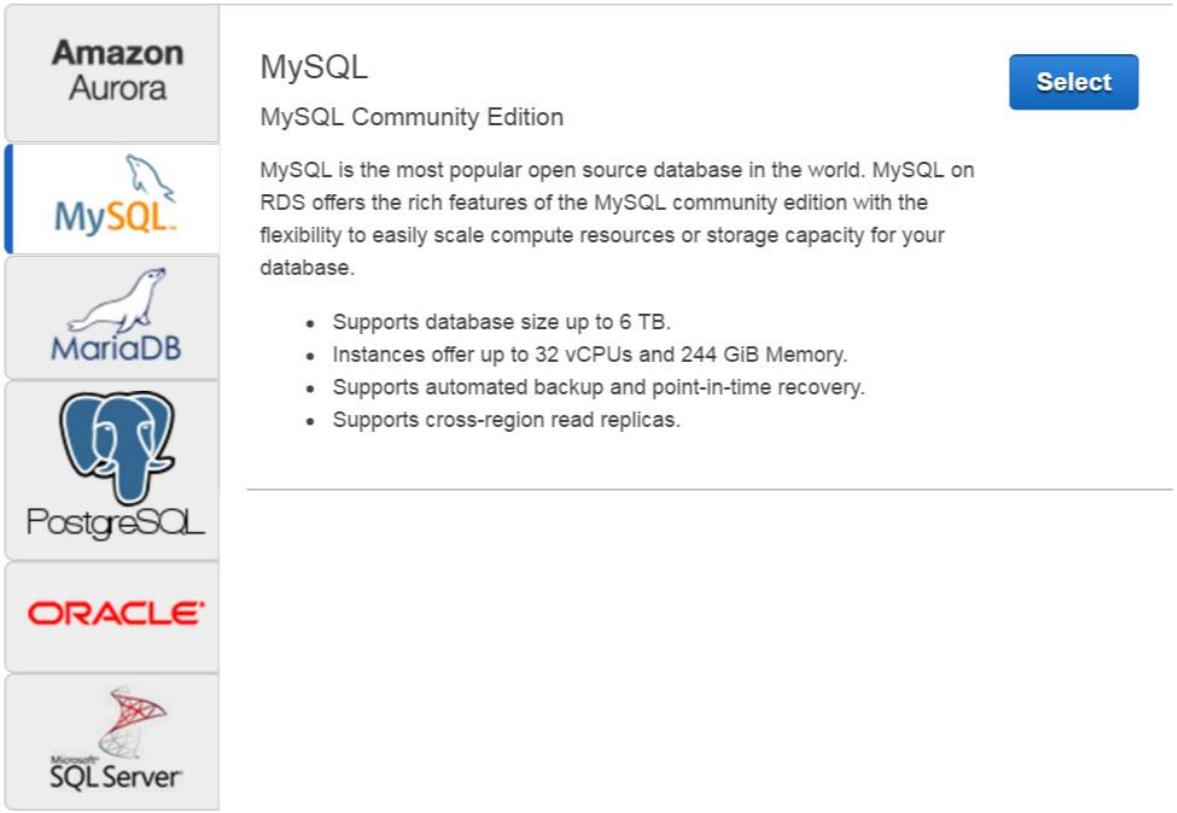
- Hard to scale out
  - Horizontal partitioning/sharding possible
  - But would need distributed storage & computing support like Hadoop & MapReduce

# RDBMS Examples

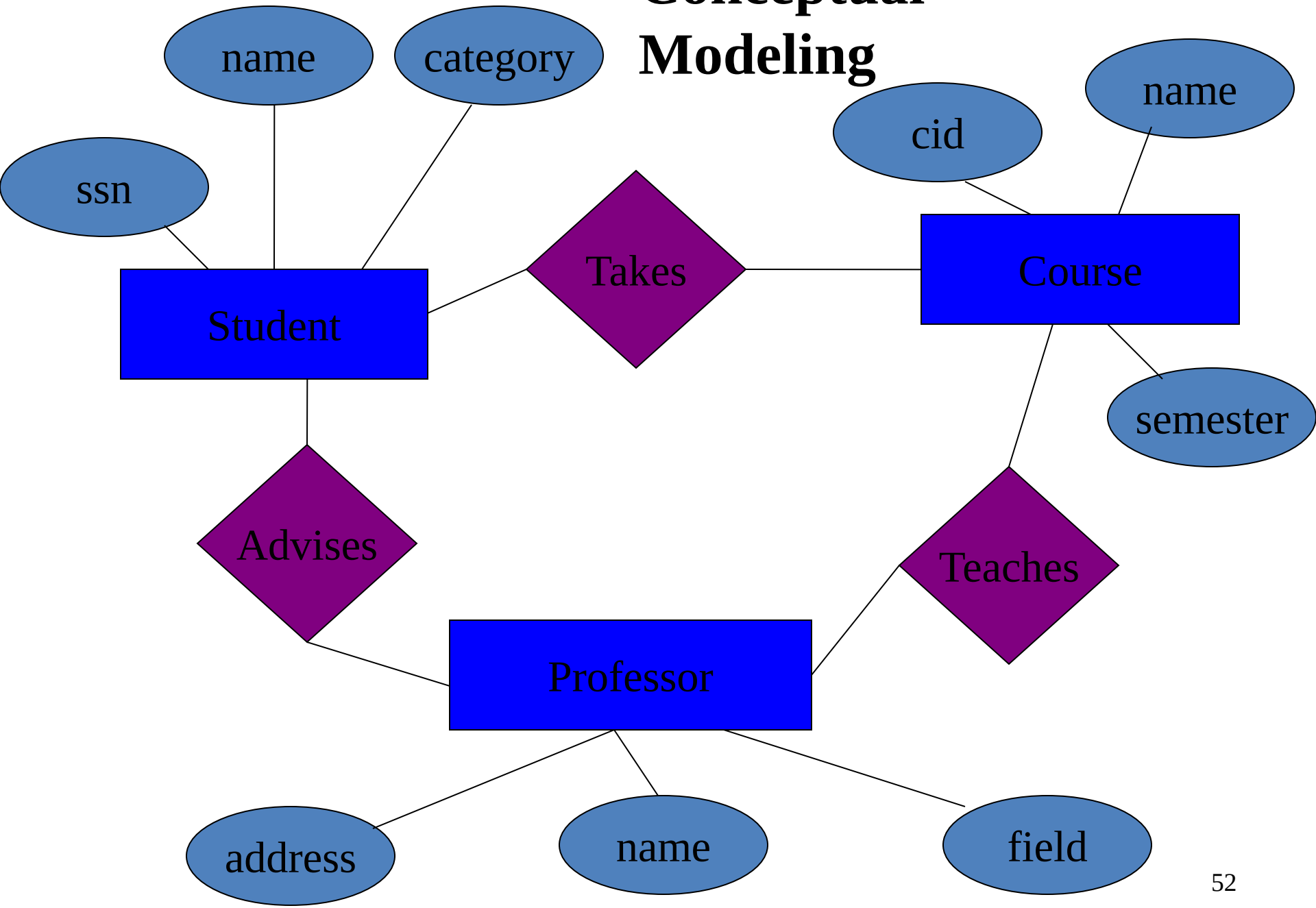
- MySQL (can be installed in Amazon AWS EC2)
- Amazon RDS (Relational database as a service)
  - DBMS in the cloud
  - Database as a service
- Data warehouse on RDBMS
  - OLAP

# Amazon RDS: Database-as-a-service

- MySQL, PostgreSQL, Oracle, SQL Server, etc.



# Conceptual Modeling



# Schema Design and Implementation

- Tables (relations):

Students:

| SSN | Name | Category |
|---|---|---|
| 123-45-6789 | Charles | undergrad |
| 234-56-7890 | Dan | grad |
| | … | … |

Takes:

| SSN | CID |
|---|---|
| 123-45-6789 | CSE444 |
| 123-45-6789 | CSE541 |
| 234-56-7890 | CSE142 |
| | … |

Courses:

| CID | Name | Semster |
|---|---|---|
| CSE444 | Databases | fall |
| CSE541 | Operating systems | spring |

- Separates the logical view from the physical view of the data.

# Querying a Database

Select A's ,agg
From R's
Where C's
Group by A's
Having
Order by
Limit ?
Offset ?
(pagination)

- Find all courses that "Mary" takes
- S(tructured) Q(uery) L(anguage)
  - clause

```
select  C.name
from    Students S, Takes T, Courses C
where   S.name = "Mary" and
        S.ssn = T.ssn and T.cid = C.cid
```

===
Insert
Update
Delete

Declarative (what)

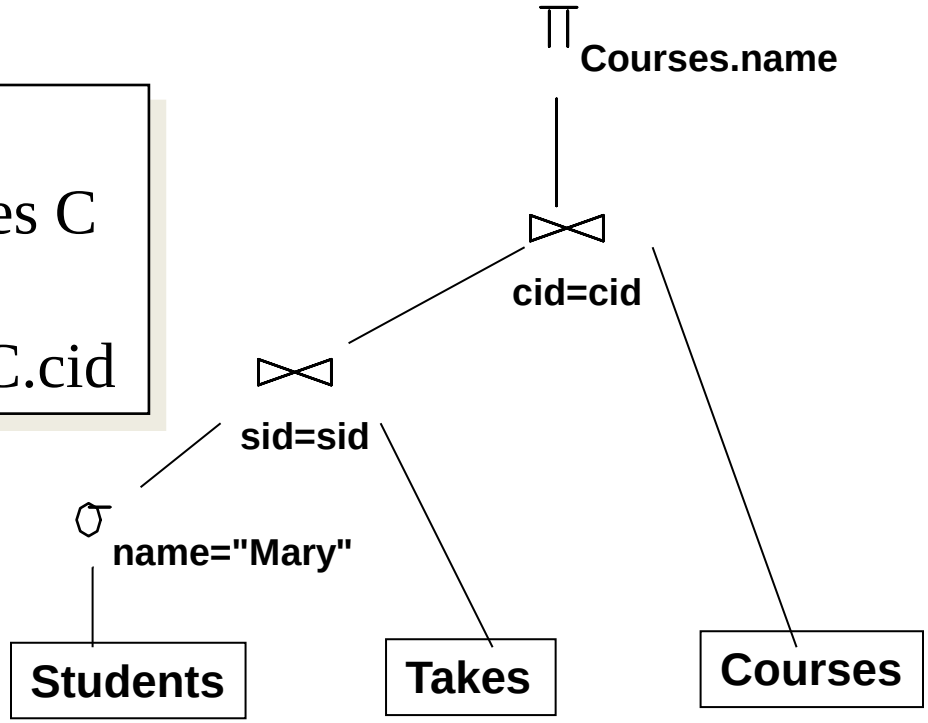
- Query processor figures out how to answer the query efficiently.

# Query Optimization

**Goal:**

*Declarative SQL query* → *Imperative query execution plan:*

```
select  C.name
from Students S, Takes T, Courses C
where S.name="Mary" and
        S.ssn = T.ssn and T.cid = C.cid
```




filtering
projection

*Plan:* tree of Relational Algebra operators,
choice of algorithms at each operator

# Major topics

- Storage systems
- File systems & file formats
- Database management systems
- Big data solution stack 

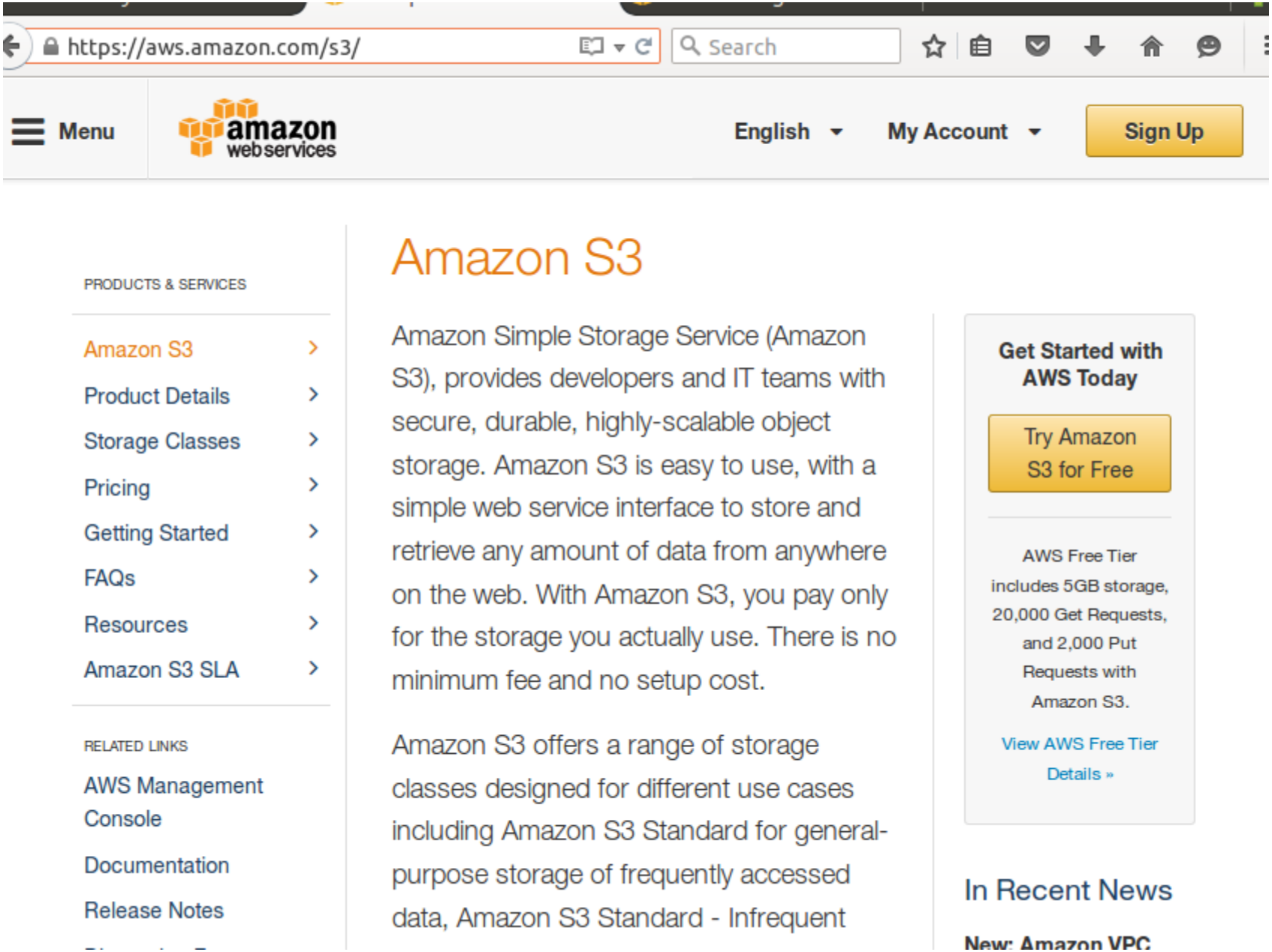
# Topics

- Big data management & analytics
  - Cloud data storage (Amazon S3)
  - NoSQL
    - Google Firebase (real-time database, …)
    - MongoDB (shell, mongo)
    - Amazon DynamoDB (row store, key-value)
    - Cassandra (not required)
  - Apache Hadoop & MapReduce
  - Apache Spark

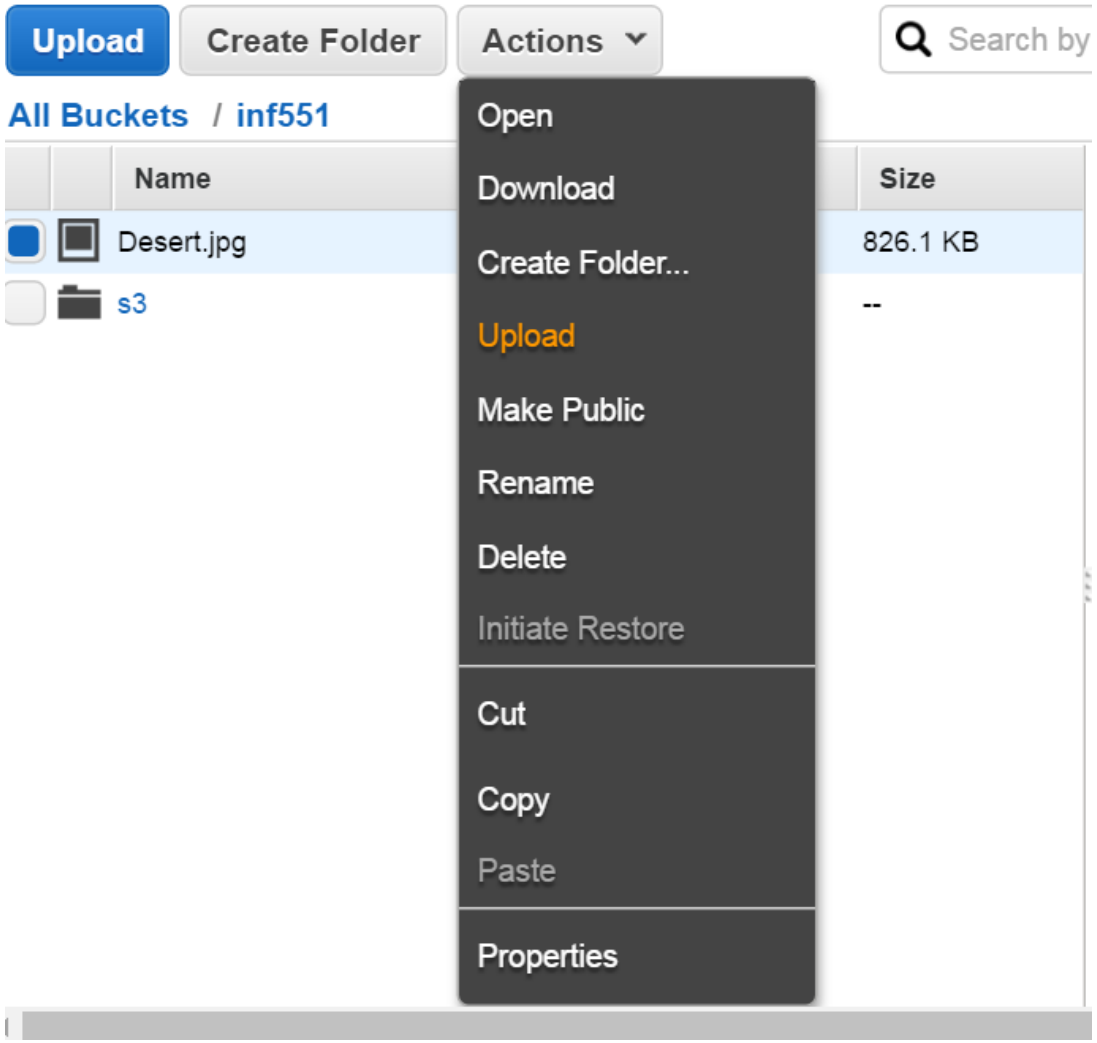
# Cloud data storage

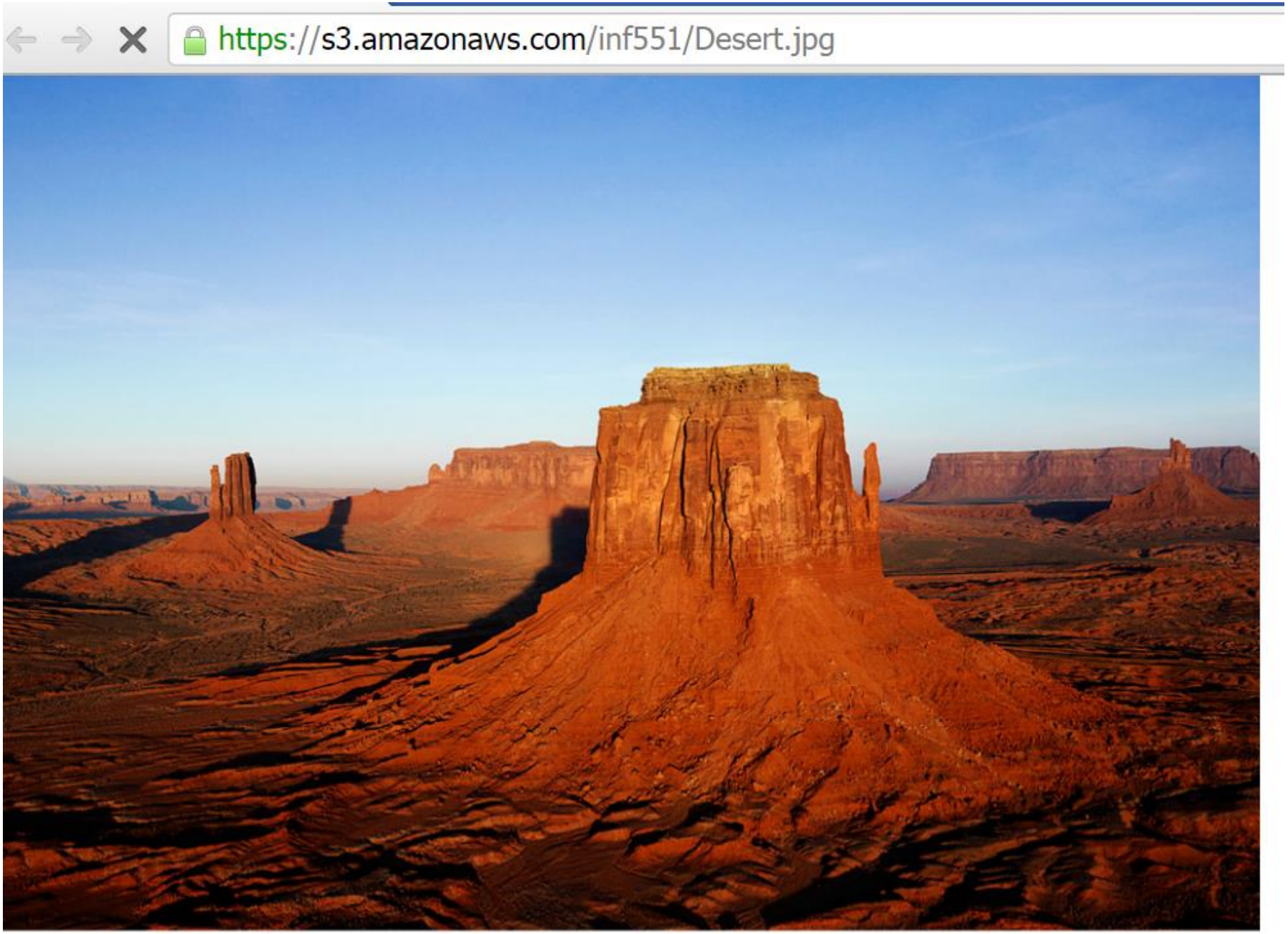
- Amazon S3 (simple storage service)
  - Ideal for storing large binary files
  - E.g., audio, video, image
  - Simple RESTful web service

- Eventual consistency for high availability


https://aws.amazon.com/s3/
Search
Menu
amazon web services
English
My Account
Sign Up
PRODUCTS & SERVICES
Amazon S3
Product Details
Storage Classes
Pricing
Getting Started
FAQs
Resources
Amazon S3 SLA
RELATED LINKS
AWS Management Console
Documentation
Release Notes
Amazon S3
Amazon Simple Storage Service (Amazon S3), provides developers and IT teams with secure, durable, highly-scalable object storage. Amazon S3 is easy to use, with a simple web service interface to store and retrieve any amount of data from anywhere on the web. With Amazon S3, you pay only for the storage you actually use. There is no minimum fee and no setup cost.
Amazon S3 offers a range of storage classes designed for different use cases including Amazon S3 Standard for general-purpose storage of frequently accessed data, Amazon S3 Standard - Infrequent
Get Started with AWS Today
Try Amazon S3 for Free
AWS Free Tier includes 5GB storage, 20,000 Get Requests, and 2,000 Put Requests with Amazon S3.
View AWS Free Tier Details »
In Recent News
New: Amazon VPC

# Upload a file




https://s3.amazonaws.com/inf551/Desert.jpg

# NoSQL

- Not only SQL
- Flexible schemas
  - e.g., JSON documents or key-value pairs
  - Ideal for managing a mix of structured, semi-structured, and unstructured data
- High availability (CAP)
- Weaker (e.g., eventual) consistency model

# Example NoSQL databases

- MongoDB, Firebase, etc.
  - Manage JSON documents
- Amazon DynamoDB
  - Row store
  - row = item = a collection of key-value pairs
- Apache Cassandra (not required)
  - Wide column store
  - Google's Bigtable clone
- Neo4J…

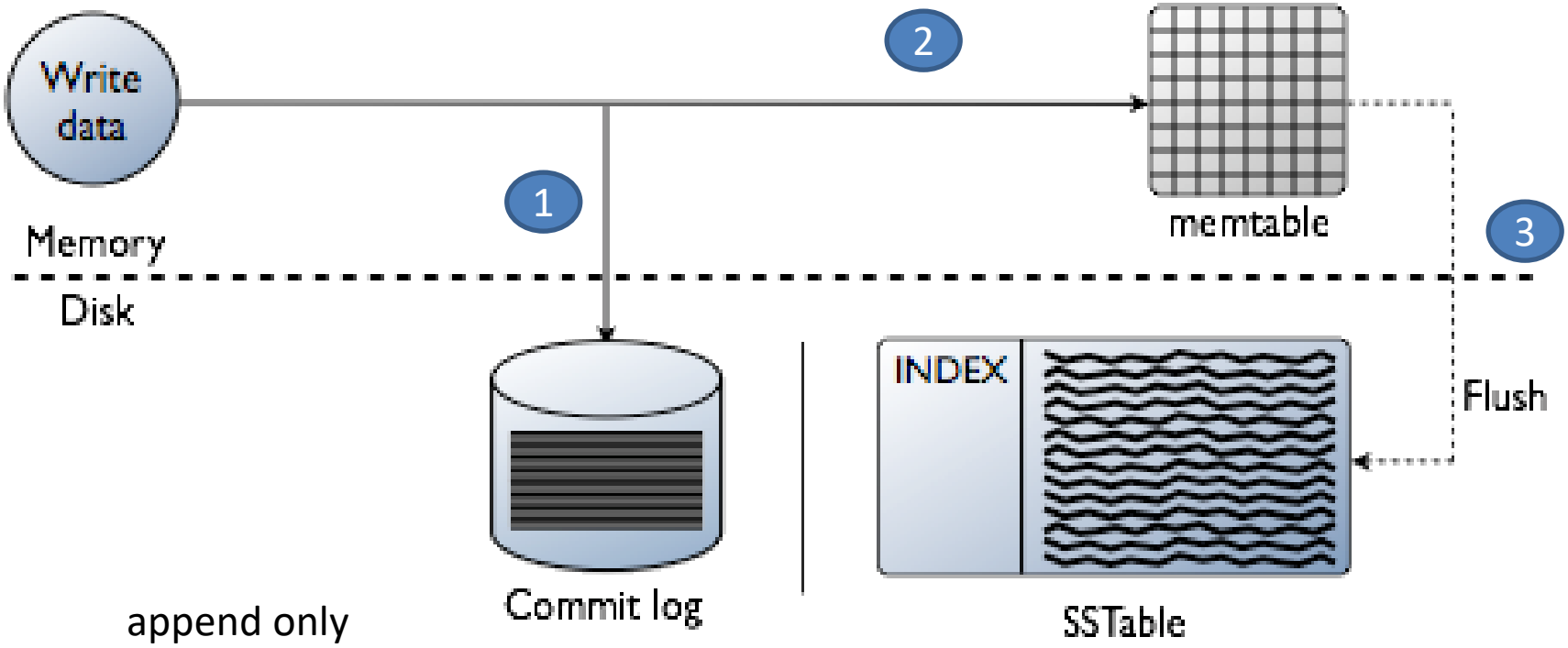
# Key techniques

- Consistent hashing (Cassandra, Dynamo)
  - Avoid moving too much data when adding new machines (scaling out)

- Efficient writes (for update-heavy apps)
  - Append-only
  - No overwrites
  - Avoid random seek
  - But compaction needed later

# Write path in Cassandra

Cassandra does not write into database
It has good write performance



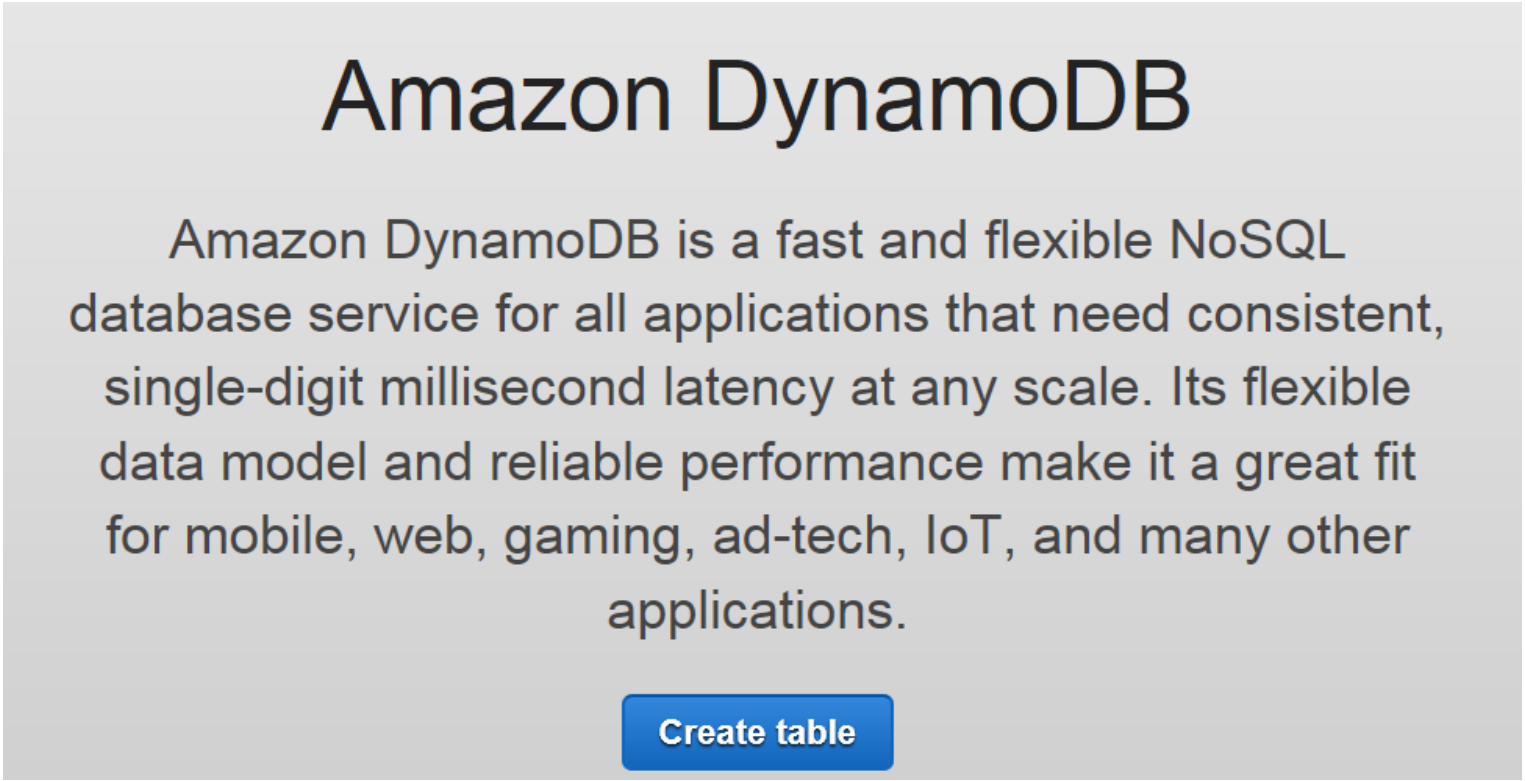

append only
log is append only
you never overwrite log data
so SSD is good with that

# Key techniques

- Compaction
  - Introduced in Google "Bigtable" paper
  - Merge multiple versions of data
  - Remove expired or deleted data

# DynamoDB

- https://console.aws.amazon.com/dynamodb/home?region=us-east-1#gettingStarted:



# Create DynamoDB table

Tutorial

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

**Table name*** Books

**Primary key*** Partition key

Author String

Add sort key

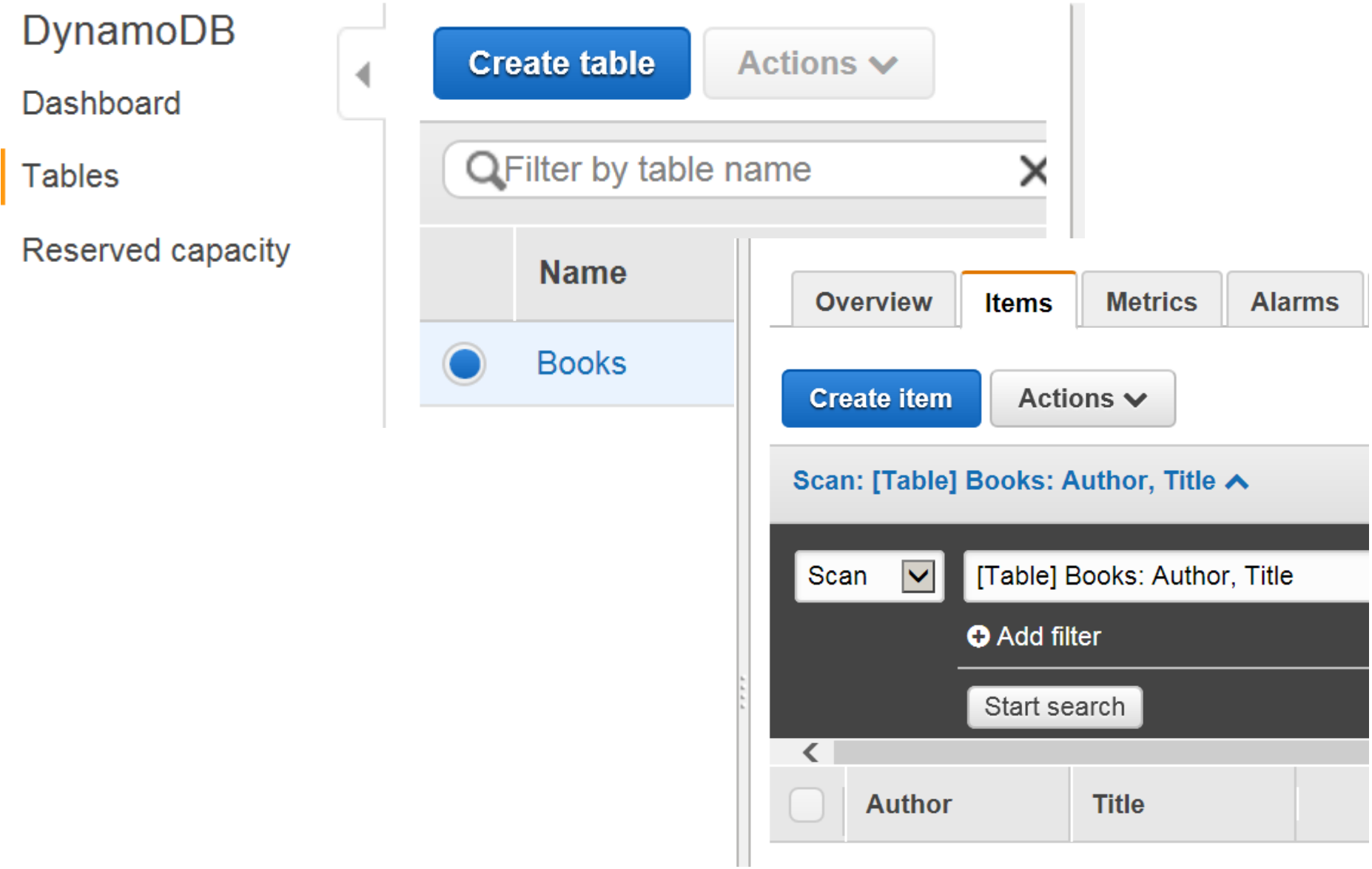
Title String

## Table settings

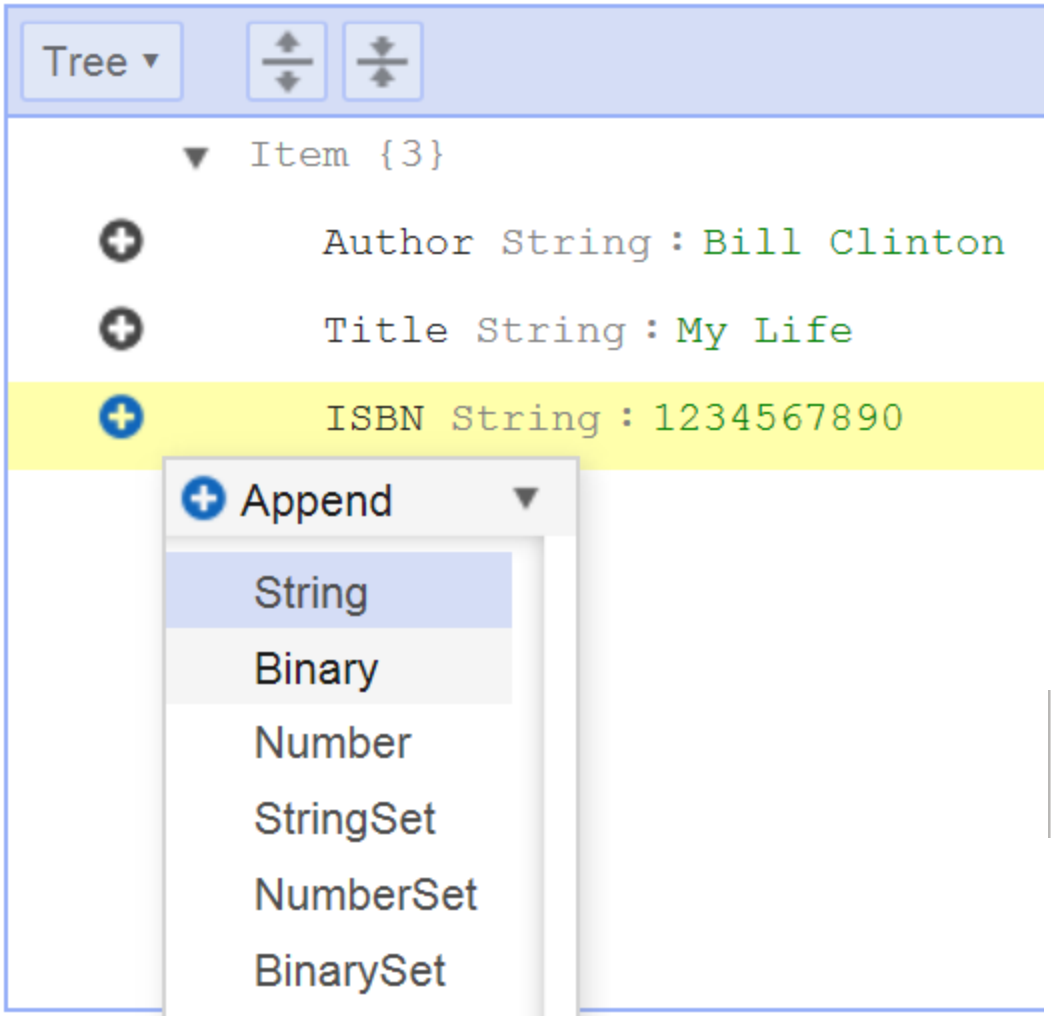
Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

Use default settings

# Insert items

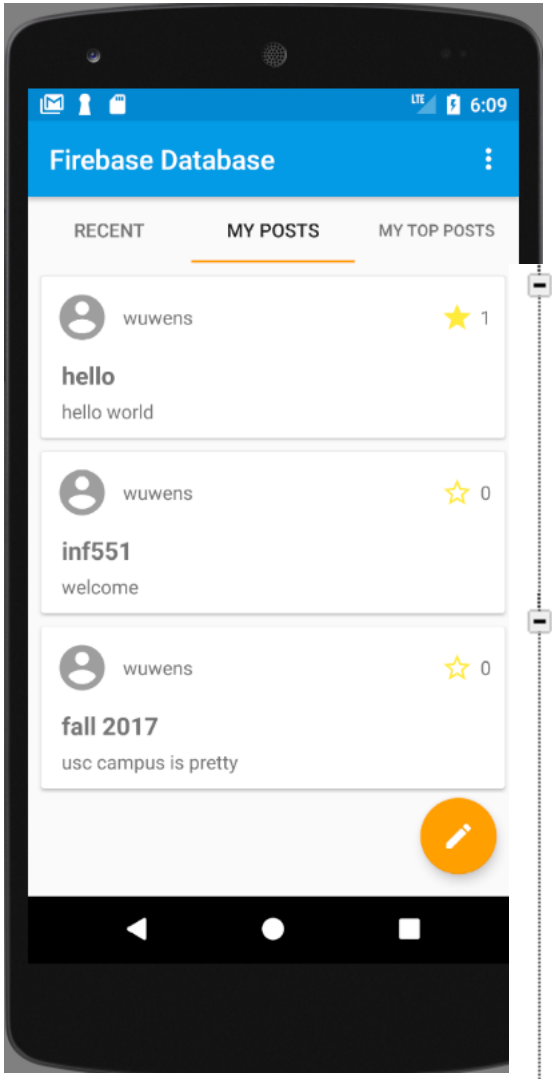


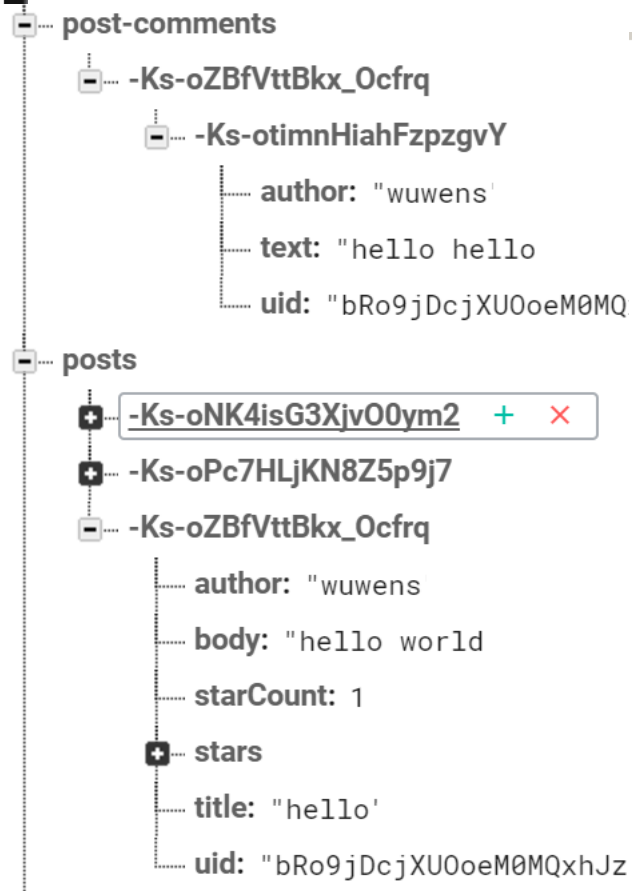
# May add new attributes




| Author | Title | ISBN |
|---|---|---|
| Bill Clinton | My Life | 1234567890 |

# Firebase: a cloud database

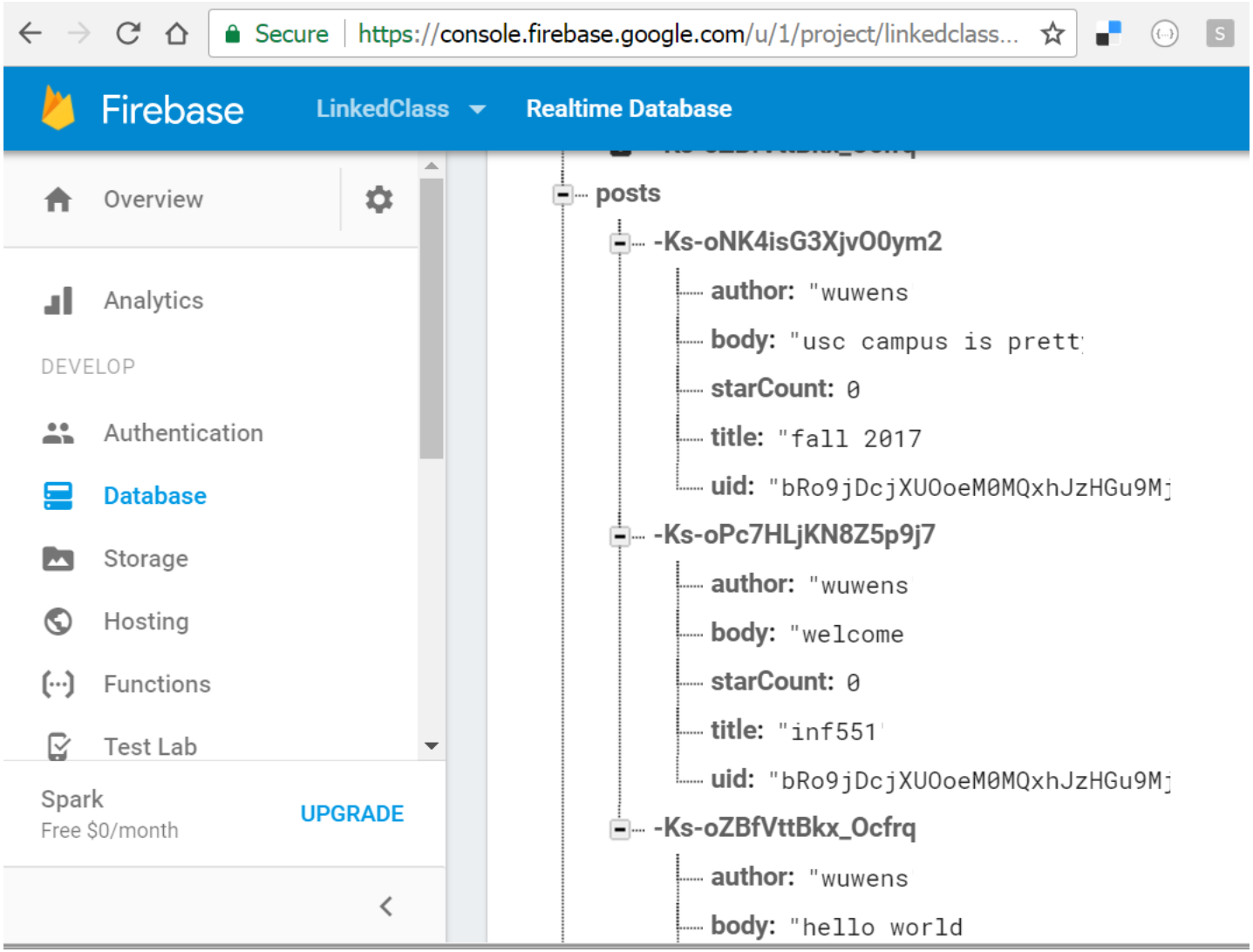




```
"post-comments" : {
  "-Ks-oZBfVttBkx_Ocfrq" : {
    "-Ks-otimnHiahFzpzgvY" : {
      "author" : "wuwens",
      "text" : "hello hello",
      "uid" : "bRo9jDcjXUOoeM0MQxhJzHGu9Mj2"
    }
  }
},
"posts" : {
  "-Ks-oNK4isG3XjvO0ym2" : {
    "author" : "wuwens",
    "body" : "usc campus is pretty",
    "starCount" : 0,
    "title" : "fall 2017",
    "uid" : "bRo9jDcjXUOoeM0MQxhJzHGu9Mj2"
  },
  "-Ks-oPc7HLjKN8Z5p9j7" : {
    "author" : "wuwens",
    "body" : "welcome",
    "starCount" : 0,
    "title" : "inf551",
    "uid" : "bRo9jDcjXUOoeM0MQxhJzHGu9Mj2"
  },
  "-Ks-oZBfVttBkx_Ocfrq" : {
    "author" : "wuwens",
    "body" : "hello world",
    "starCount" : 1,
    "stars" : {
      "bRo9jDcjXUOoeM0MQxhJzHGu9Mj2" : true
    },
    "title" : "hello",
    "uid" : "bRo9jDcjXUOoeM0MQxhJzHGu9Mj2"
  }
},
```

# Firebase


Secure
https://console.firebase.google.com/u/1/project/linkedclass...
Firebase
LinkedClass
Realtime Database
Overview
Analytics
DEVELOP
Authentication
Database
Storage
Hosting
Functions
Test Lab
Spark
Free $0/month
UPGRADE
posts
-Ks-oNK4isG3XjvO0ym2
author: "wuwens
body: "usc campus is prett
starCount: 0
title: "fall 2017
uid: "bRo9jDcjXUOoeM0MQxhJzHGu9Mj
-Ks-oPc7HLjKN8Z5p9j7
author: "wuwens
body: "welcome
starCount: 0
title: "inf551
uid: "bRo9jDcjXUOoeM0MQxhJzHGu9Mj
-Ks-oZBfVttBkx_Ocfrq
author: "wuwens
body: "hello world

# Topics

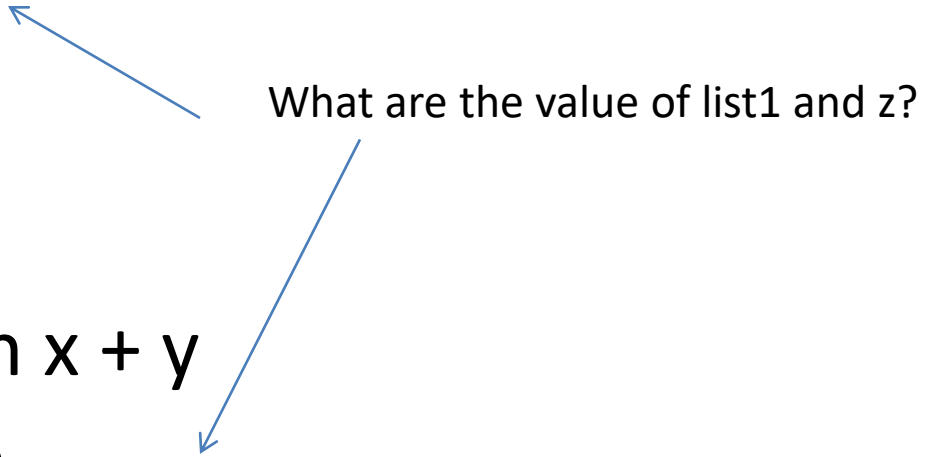
- Big data management & analytics
  - Cloud data storage (Amazon S3)
  - NoSQL (Amazon DynamoDB, Cassandra, MongoDB)
  - MapReduce 
  - Apache Hadoop
  - Apache Spark

# Roots in functional programming

- Functional programming languages:
  - Python, Lisp (list processor), Scheme, Erlang, Haskell
- Two functions:
  - Map: mapping a list => list
  - Reduce: reducing a list => value
- map() and reduce() in Python
  - https://docs.python.org/2/library/functions.html#map

# map() and reduce() in Python

- list = [1, 2, 3]
- def sqr(x): return x ** 2
- list1 = map(sqr, list)




- def add(x, y): return x + y
- z = reduce(add, list)

reduce() is in functools module of Python 3

# Lambda function

- Anonymous function (not bound to a name)

- list = [1, 2, 3]

- list1 = map(lambda x: x ** 2, list)
- z = reduce(lambda x, y: x + y, list)

# How is reduce() in Python evaluated?

- z = reduce(f, list) where f is add function

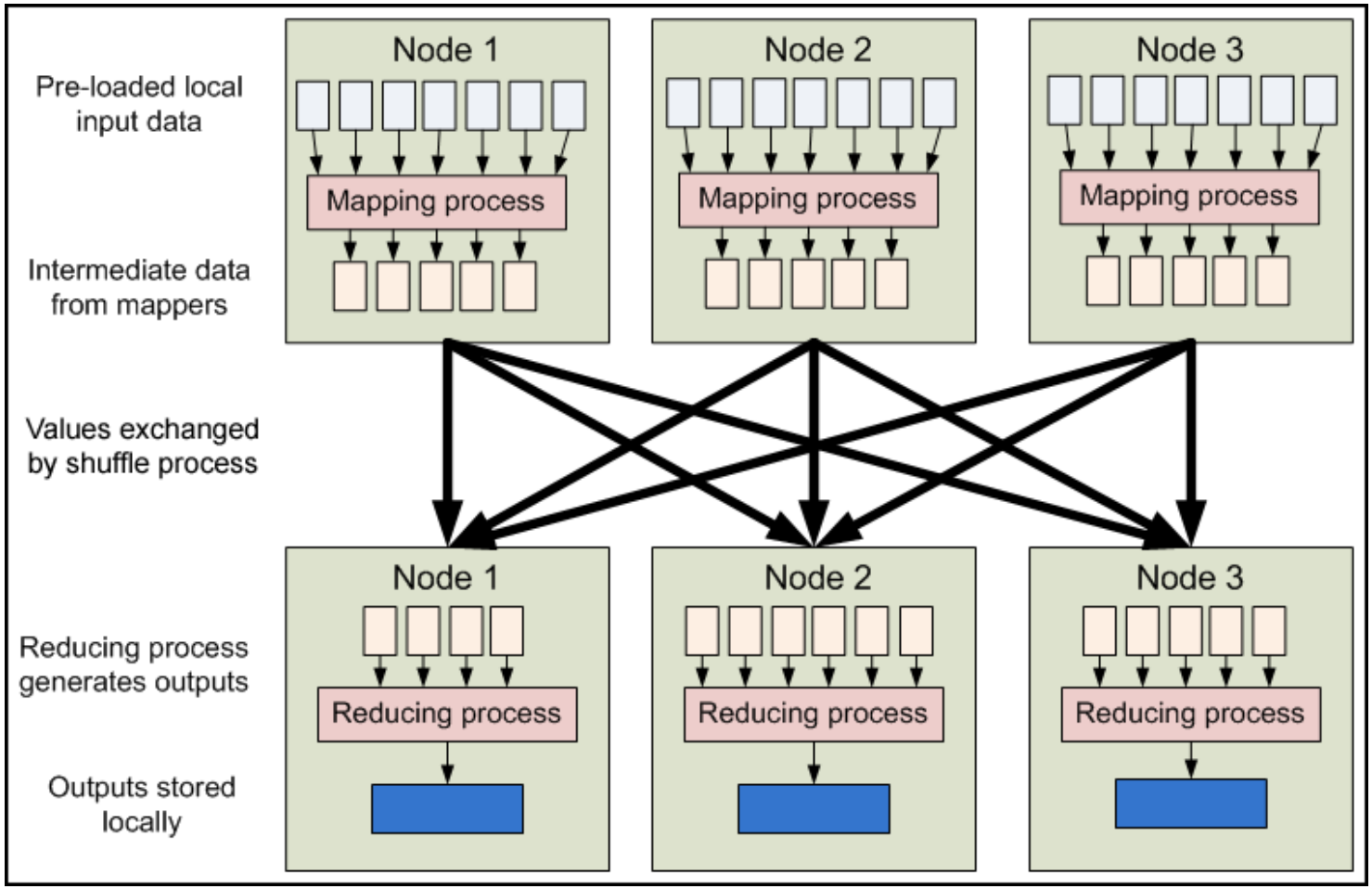
- Initially, z (an accumulator) is set to list[0]
- Next, repeat z = add(z, list[i]) for each i > 0
- Return final z

- Example: z = reduce(add, [1, 2, 3])
  - i = 0, z = 1; i = 1, z = 3; i = 2, z = 6

# Hadoop MapReduce

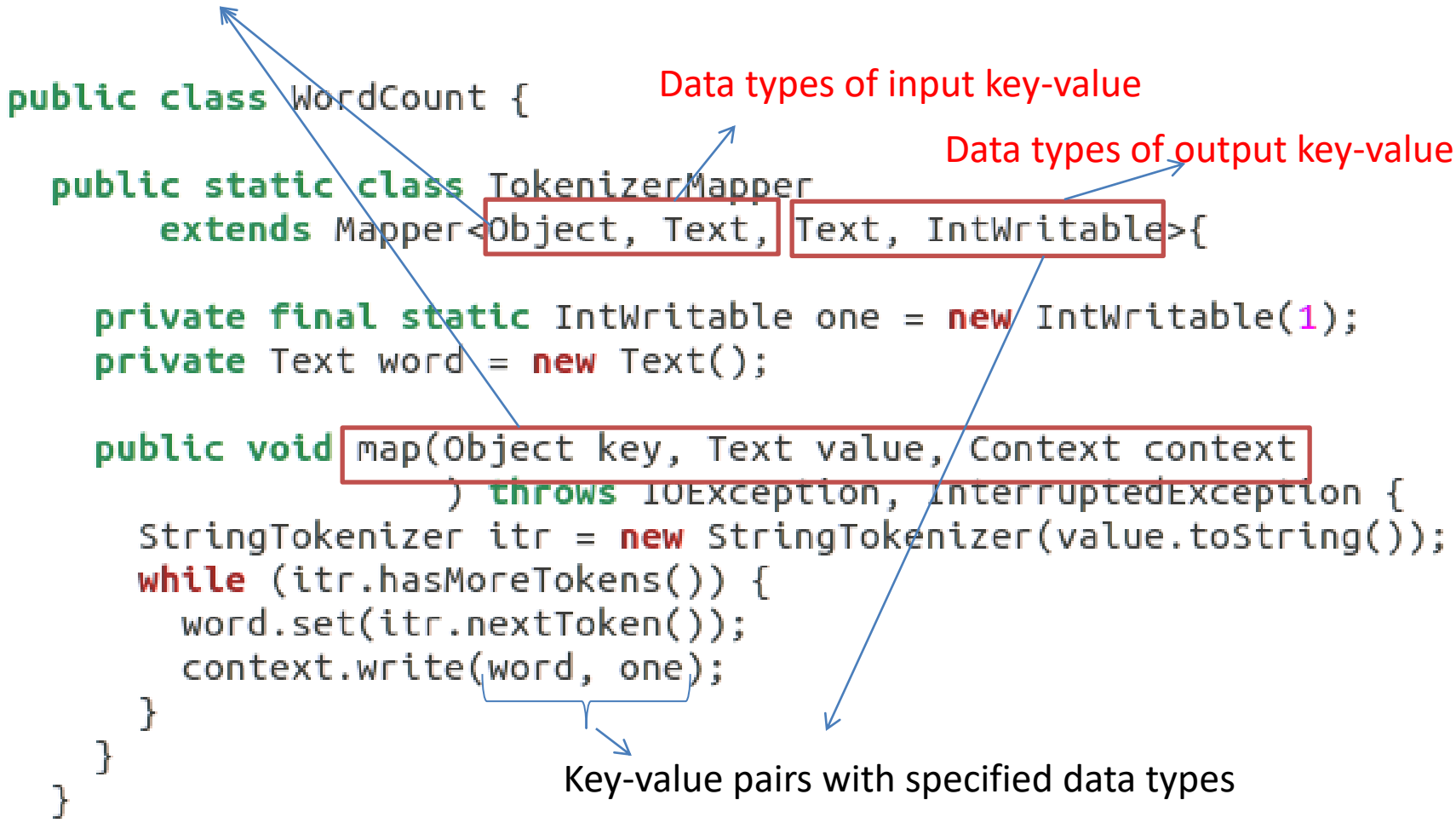
- Map
  - <k, v> => list of <k', v'>
- Reduce:
  - <k', list of v'> => list of <k'', v''>
- Write MapReduce programs on Hadoop
  - Using Java

# MapReduce


Pre-loaded local input data
Intermediate data from mappers
Values exchanged by shuffle process
Reducing process generates outputs
Outputs stored locally
Node 1
Node 2
Node 3
Mapping process
Mapping process
Mapping process
Node 1
Node 2
Node 3
Reducing process
Reducing process
Reducing process

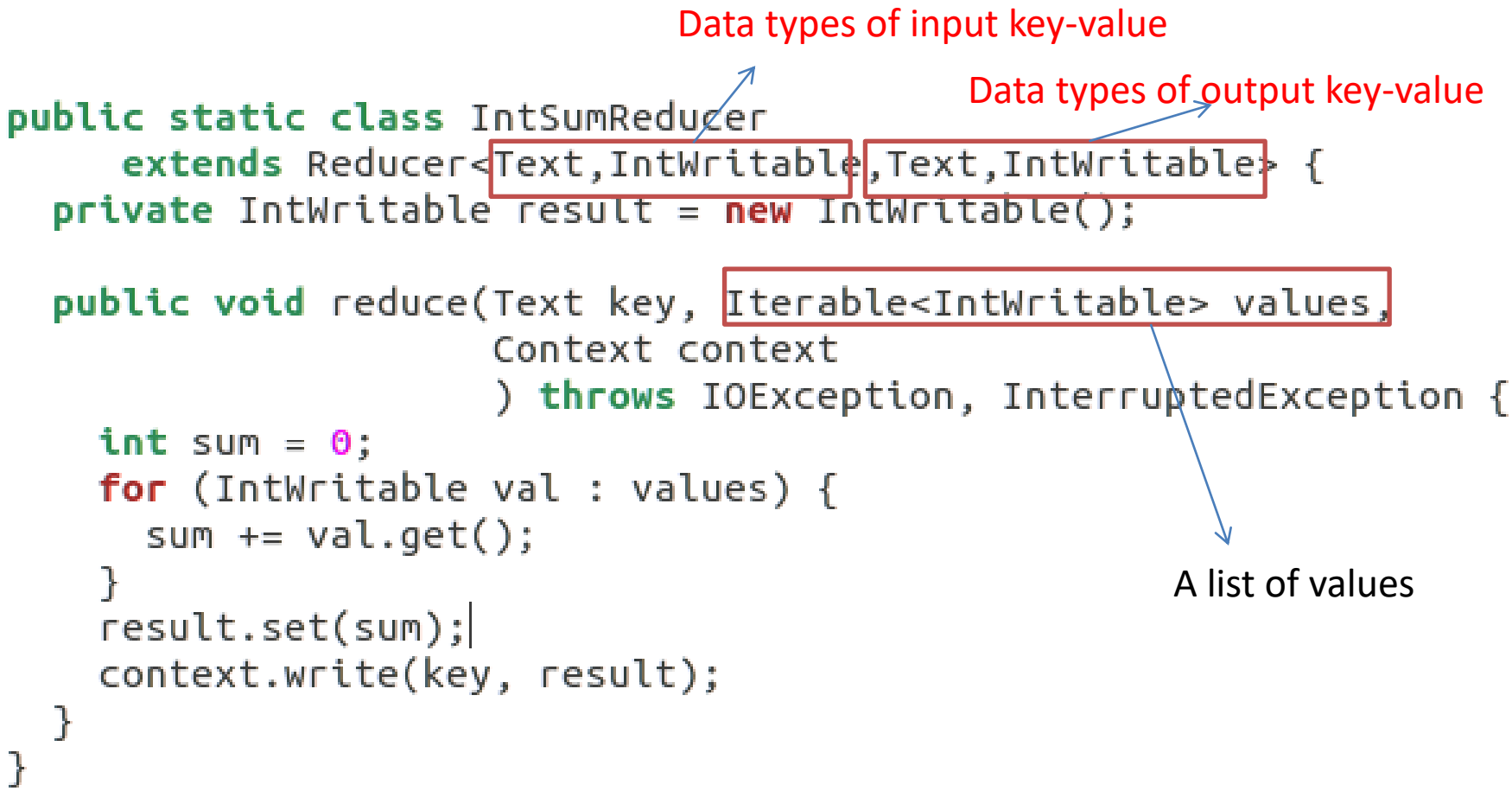
# WordCount: mapper

Object can be replaced with LongWritable




```
public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

# WordCount: reducer




```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

# Characteristics of Hadoop

- Acyclic data flow model
  - Data loaded from stable storage (e.g., HDFS)
  - Processed through a sequence of steps
  - Results written to disk

- Batch processing
  - No interactions permitted during processing

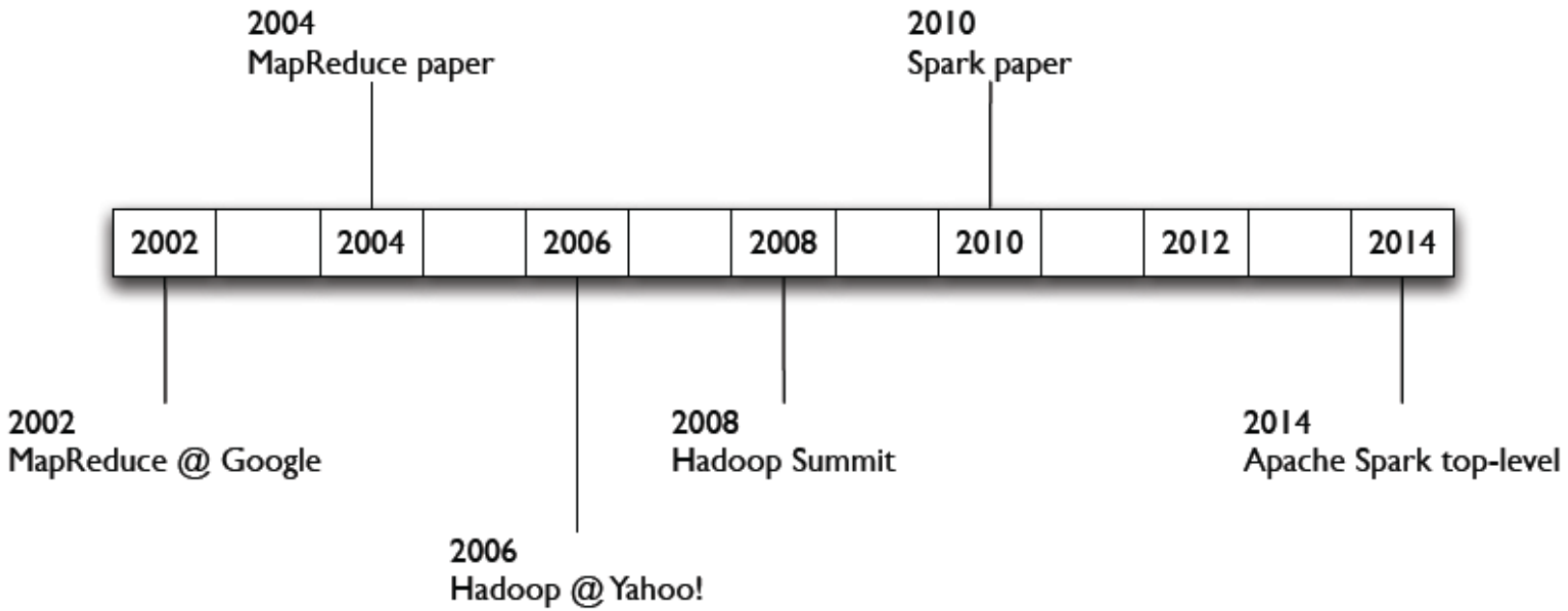
# Problems

- Ill-suited for iterative algorithms that requires repeated reuse of data
  - E.g., machine learning and data mining algorithms such as k-means, PageRank, logistic regression
- Ill-suited for interactive exploration of data
  - E.g., OLAP on big data

# In-memory MapReduce (Spark)

- Key concepts
  - RDD (resilient distributed dataset)
  - Transformations
  - Actions

# Apache Spark: history

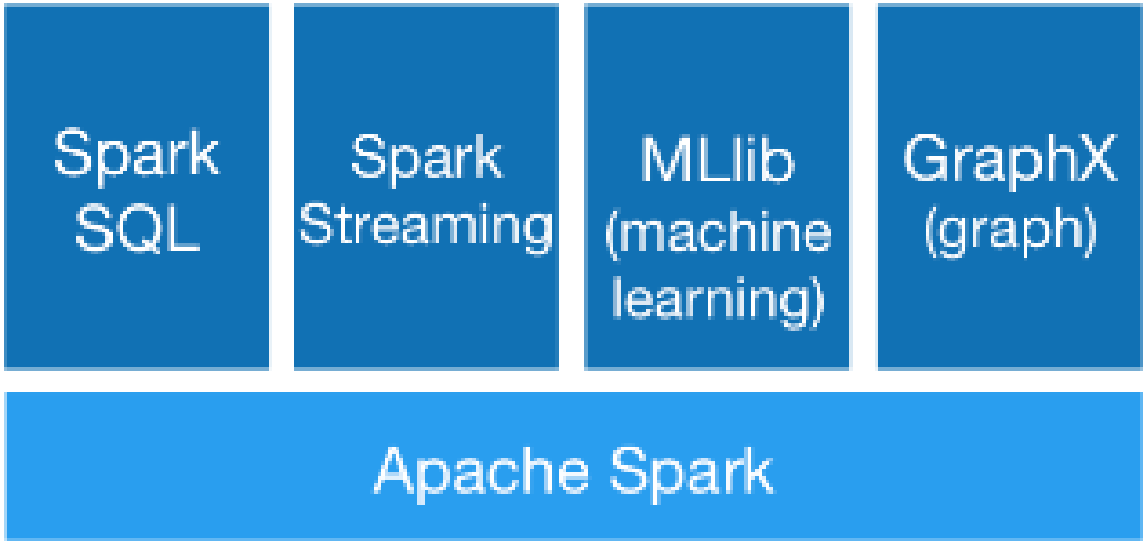


# Spark

- Support working sets through RDD
  - Enabling reuse & fault-tolerance
- 10x faster than Hadoop in iterative jobs
- Interactively explore 39GB with sub-second response time

# Spark

- Combine SQL, streaming, and complex analytics
- We will see DataFrame in Spark too



# Spark

- Run on Hadoop, Cassandra, HBase, etc.





# wc.py

```
from pyspark import SparkContext
from operator import add

sc = SparkContext(appName="dsci551")

lines = sc.textFile('hello.txt')

counts = lines.flatMap(lambda x: x.split(' ')) \
          .map(lambda x: (x, 1)) \
          .reduceByKey(add)

output = counts.collect()

for v in output:
   print(v[0], v[1])
```

# Coming up…

- Task: Setting up an EC2 instance

- Details:
  - see posted instructions and come to class!

# Resources

- Merge sort:
  - https://www.interviewbit.com/tutorial/merge-sort-algorithm/
  - https://www.youtube.com/watch?v=Nso25TkBsYI

- Hashing
  - https://www.tutorialspoint.com/python_data_structure/python_hash_table.htm
  - https://www.programiz.com/python-programming/methods/built-in/hash