

PRINCIPLES OF DATA SCIENCE

JOHN P DICKERSON

Lecture #3 – 9/12/2018

CMSC641
Wednesdays
7pm – 9:30pm



COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

ANNOUNCEMENTS

The website is up! Hopefully ...

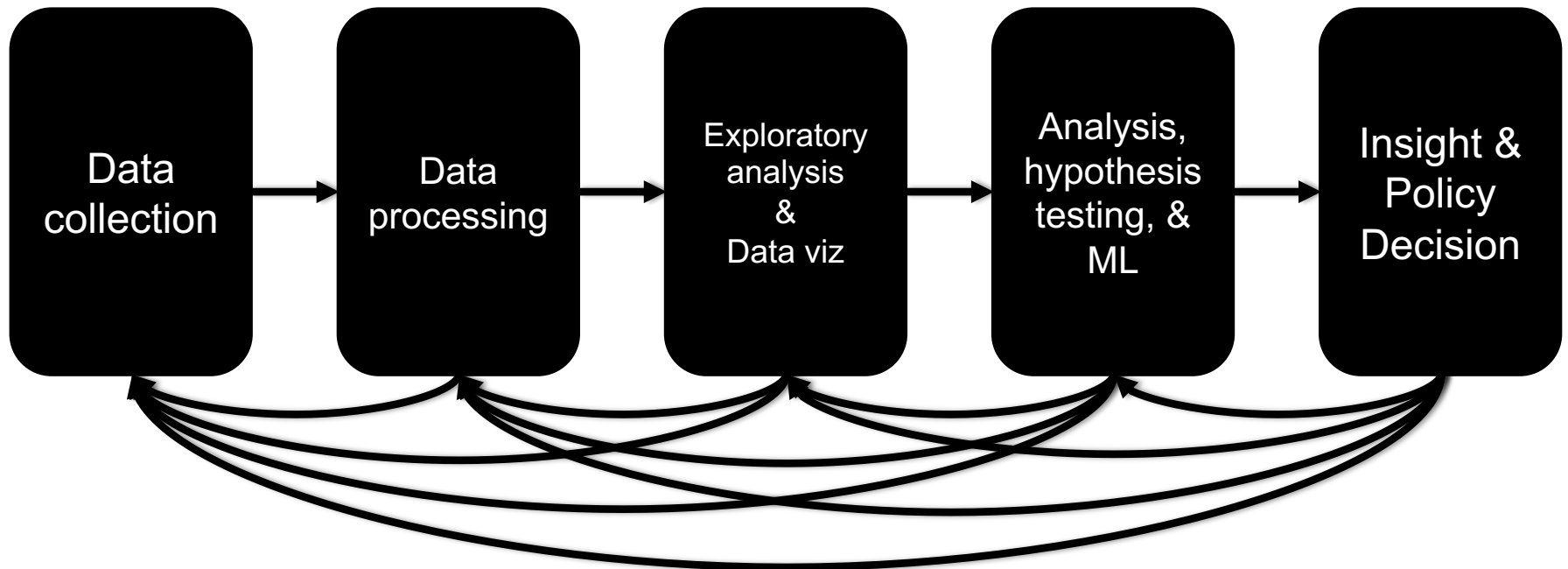
- <http://www.cs.umd.edu/class/fall2018/cmsc641/>

Office Hours: 6pm-7pm on Wednesdays, and also by appointment, or via email/text, or ...

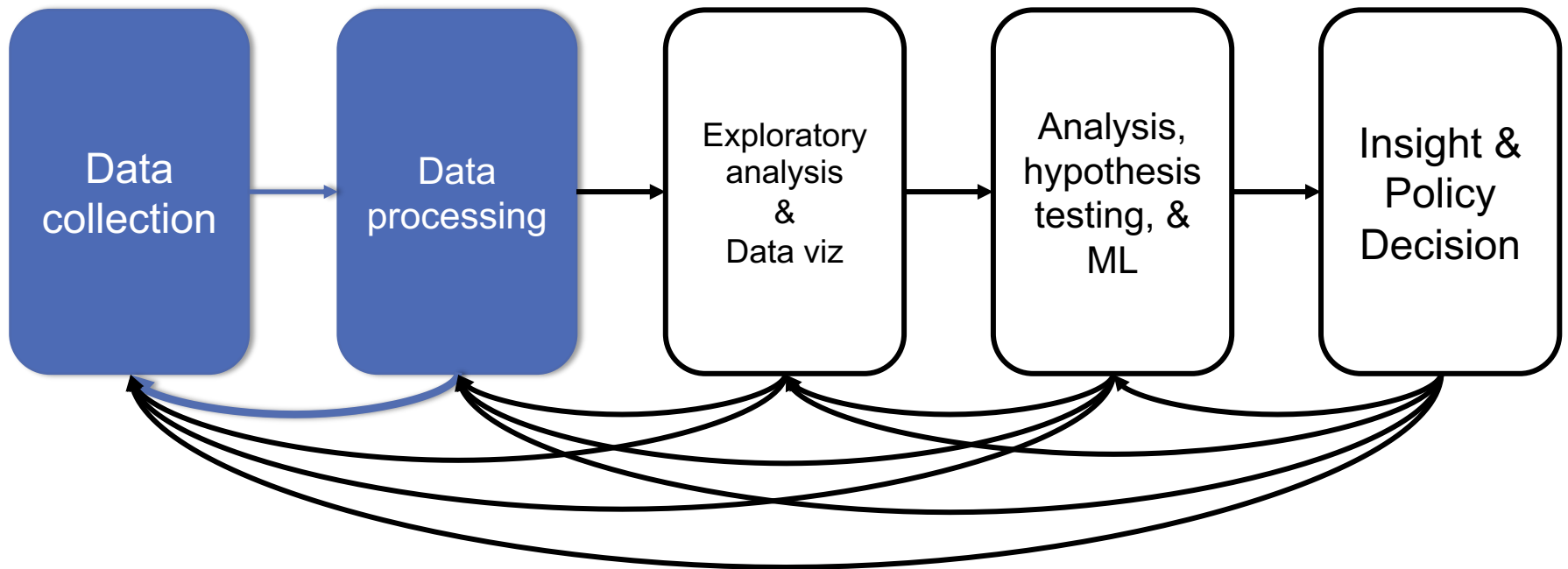
Reminder: Weekly quizzes, due on Mondays at noon

Project 1 will be released this week

THE DATA LIFECYCLE



THE DATA LIFECYCLE



DATA MANIPULATION AND COMPUTATION

Data Science == manipulating and computing on data

Large to very large, but somewhat “structured” data

We will see several tools for doing that this semester

Thousands more out there that we won't cover

Need to learn to shift thinking from:

Imperative code to manipulate data structures

to:

Sequences/pipelines of operations on data

Should still know how to implement the operations themselves, especially for debugging performance (covered in classes like 642?, 643?), but we won't cover that much

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

One-dimensional Arrays, Vectors

0.1	2	3.2	6.5	3.4	4.1
-----	---	-----	-----	-----	-----

"data"	"representation"	"i.e."
--------	------------------	--------

Indexing

Slicing/subsetting

Filter

'map' → apply a function to every element

'reduce/aggregate' → combine values to get a single scalar (e.g., sum, median)

Given two vectors: **Dot and cross products**

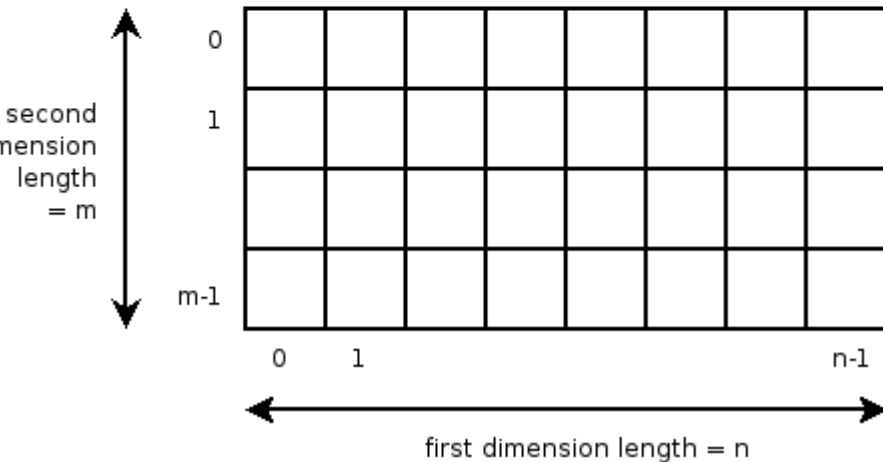
2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

n-dimensional arrays

Two-dimensional array



Indexing

Slicing/subsetting

Filter

'map' → apply a function to every element

'reduce/aggregate' → combine values across a row or a column (e.g., sum, average, median etc..)

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

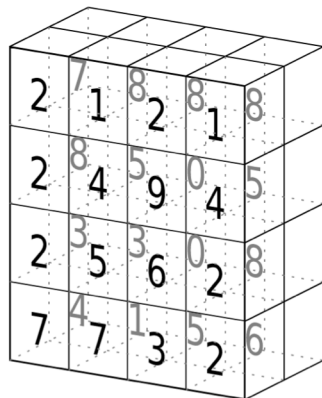
DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Matrices, Tensors

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

tensor of dimensions [6,4]
(matrix 6 by 4)



tensor of dimensions [4,4,2]

n-dimensional array operations
+

Linear Algebra

Matrix/tensor multiplication

Transpose

Matrix-vector multiplication

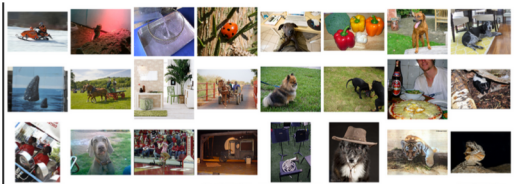
Matrix factorization

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Sets: of Objects



Filter
Map
Union

Reduce/Aggregate

Sets: of (Key, Value Pairs)

(amol@cs.umd.edu, (email1, email2, ...))

(john@cs.umd.edu, (email3, email4, ...))

Given two sets, **Combine/Join** using “keys”

Group and then aggregate

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Tables/Relations == Sets of Tuples

company	division	sector	tryint
00nil_Combined_Company	00nil_Combined_Division	00nil_Combined_Sector	14625
apple	00nil_Combined_Division	00nil_Combined_Sector	10125
apple	hardware	00nil_Combined_Sector	4500
apple	hardware	business	1350
apple	hardware	consumer	3150
apple	software	00nil_Combined_Sector	5625
apple	software	business	4950
apple	software	consumer	675
microsoft	00nil_Combined_Division	00nil_Combined_Sector	4500
microsoft	hardware	00nil_Combined_Sector	1890
microsoft	hardware	business	855
microsoft	hardware	consumer	1035
microsoft	software	00nil_Combined_Sector	2610
microsoft	software	business	1215
microsoft	software	consumer	1395

Filter rows or columns

”Join” two or more relations

”Group” and “aggregate” them

Relational Algebra formalizes some of them

Structured Query Language (SQL)

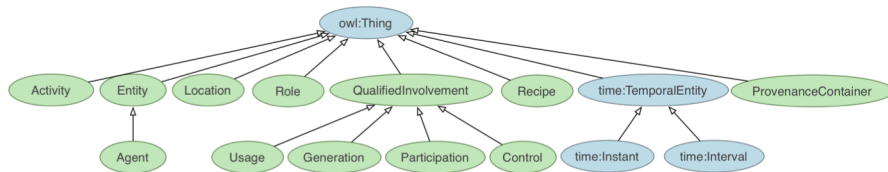
Many other languages and constructs, that look very similar

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Hierarchies/Trees/Graphs



"Path" queries

Graph Algorithms and Transformations

Network Science

Somewhat more ad hoc and special-purpose

Changing in recent years

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data
 2. **Data Processing Operations**, which take one or more datasets as input and produce
- **Why?**
 - Allows one to think at a higher level of abstraction, leading to simpler and easier-to-understand scripts
 - Provides "independence" between the abstract operations and concrete implementation
 - Can switch from one implementation to another easily
 - **For performance debugging, useful to know how they are implemented and rough characteristics**

NEXT COUPLE OF CLASSES

- 1. NumPy: Python Library for Manipulating nD Arrays**
Multidimensional Arrays, and a variety of operations including Linear Algebra
- 2. Pandas: Python Library for Manipulating Tabular Data**
Series, Tables (also called **DataFrames**)
Many operations to manipulate and combine tables/series
- 3. Relational Databases**
Tables/Relations, and SQL (similar to Pandas operations)
- 4. Apache Spark**
Sets of objects or key-value pairs
MapReduce and SQL-like operations

NEXT COUPLE OF CLASSES

1. NumPy: Python Library for Manipulating nD Arrays

Multidimensional Arrays, and a variety of operations including Linear Algebra

2. Pandas: Python Library for Manipulating Tabular Data

Series, Tables (also called **DataFrames**)

Many operations to manipulate and combine tables/series

3. Relational Databases

Tables/Relations, and SQL (similar to Pandas operations)

4. Apache Spark

Sets of objects or key-value pairs

MapReduce and SQL-like operations

NUMERIC & SCIENTIFIC APPLICATIONS

Number of third-party packages available for numerical and scientific computing

These include:

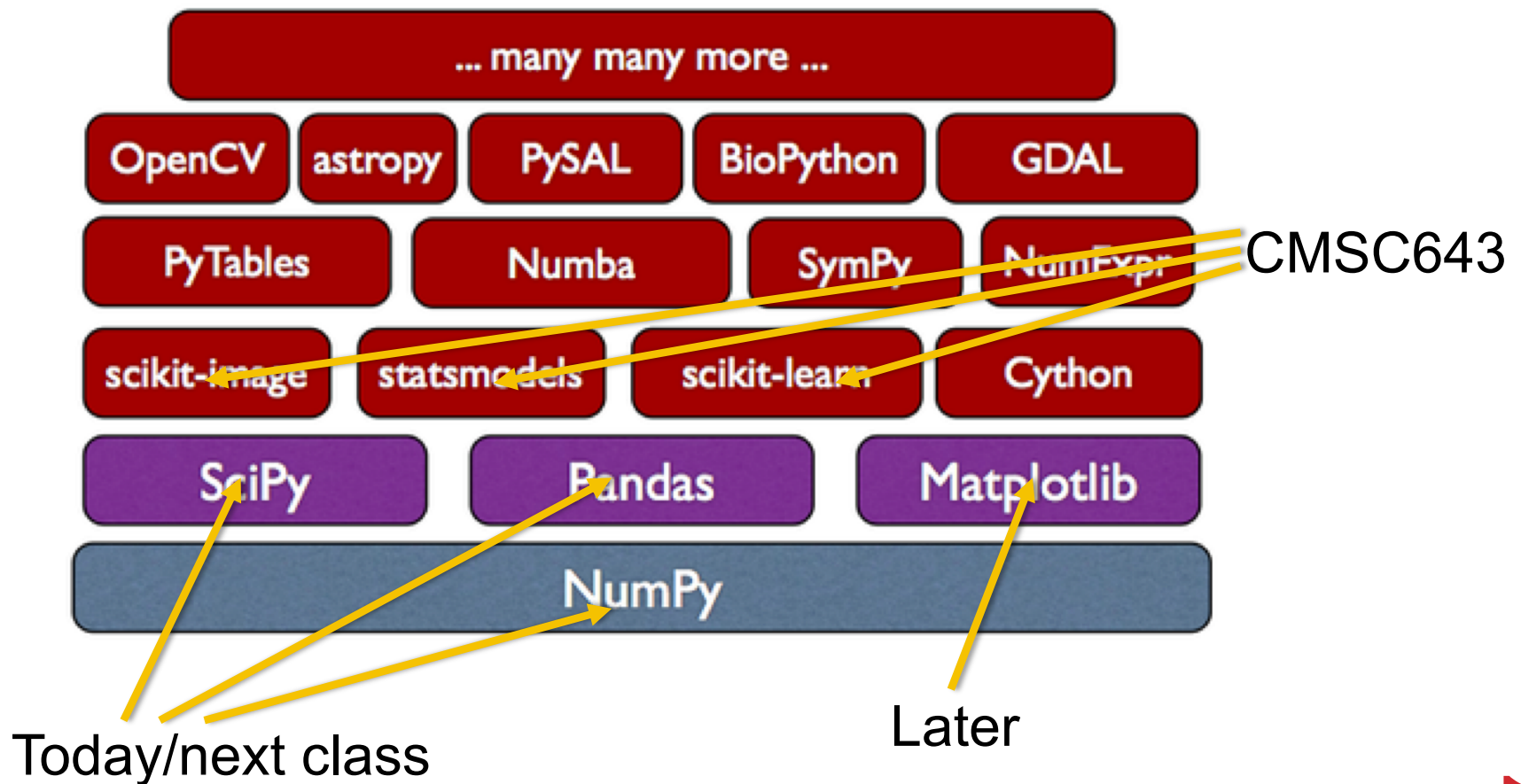
- NumPy/SciPy – numerical and scientific function libraries.
- numba – Python compiler that support JIT compilation.
- ALGLIB – numerical analysis library.
- pandas – high-performance data structures and data analysis tools.
- pyGSL – Python interface for GNU Scientific Library.
- ScientificPython – collection of scientific computing modules.

NUMPY AND FRIENDS

By far, the most commonly used packages are those in the NumPy stack. These packages include:

- NumPy: similar functionality as Matlab
- SciPy: integrates many other packages like NumPy
- Matplotlib & Seaborn – plotting libraries
- iPython via Jupyter – interactive computing
- Pandas – data analysis library
- SymPy – symbolic computation library

THE NUMPY STACK



NUMPY

Among other things, NumPy contains:

- A powerful n -dimensional array object.
- Sophisticated (broadcasting/universal) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities, etc.

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.



NUMPY

ndarray object: an n -dimensional array of homogeneous data types, with many operations being performed in compiled code for performance

Several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size. Modifying the size means creating a new array.
- NumPy arrays must be of the same data type, but this can include Python objects – may not get performance benefits
- More efficient mathematical operations than built-in sequence types.

NUMPY DATATYPES

Wider variety of data types than are built-in to the Python language by default.

Defined by the `numpy.dtype` class and include:

- `intc` (same as a C integer) and `intp` (used for indexing)
- `int8`, `int16`, `int32`, `int64`
- `uint8`, `uint16`, `uint32`, `uint64`
- `float16`, `float32`, `float64`
- `complex64`, `complex128`
- `bool_`, `int_`, `float_`, `complex_` are shorthand for defaults.

These can be used as functions to cast literals or sequence types, as well as arguments to NumPy functions that accept the `dtype` keyword argument.

NUMPY DATATYPES

```
>>> import numpy as np
>>> x = np.float32(1.0)
>>> x
1.0
>>> y = np.int_([1,2,4])
>>> y
array([1, 2, 4])
>>> z = np.arange(3, dtype=np.uint8)
>>> z
array([0, 1, 2], dtype=uint8)
>>> z.dtype
dtype('uint8')
```

NUMPY ARRAYS

There are a couple of mechanisms for creating arrays in NumPy:

- Conversion from other Python structures (e.g., lists, tuples)
 - Any sequence-like data can be mapped to a ndarray
- Built-in NumPy array creation (e.g., `arange`, `ones`, `zeros`, etc.)
 - Create arrays with all zeros, all ones, increasing numbers from 0 to 1 etc.
- Reading arrays from disk, either from standard or custom formats (e.g., reading in from a CSV file)

NUMPY ARRAYS

In general, any numerical data that is stored in an array-like container can be converted to an `ndarray` through use of the `array()` function. The most obvious examples are sequence types like lists and tuples.

```
>>> x = np.array([2,3,1,0])
```

```
>>> x = np.array([2, 3, 1, 0])
```

```
>>> x = np.array([[1,2.0],[0,0]],[1+1j,3.])])
```

```
>>> x = np.array([[ 1.+0.j, 2.+0.j], [ 0.+0.j, 0.+0.j],  
[ 1.+1.j, 3.+0.j]])
```

NUMPY ARRAYS

Creating arrays from scratch in NumPy:

- `zeros(shape)` – creates an array filled with 0 values with the specified shape. The default `dtype` is `float64`.

```
>>> np.zeros((2, 3))  
array([[ 0.,  0.,  0.], [ 0.,  0.,  0.]])
```

- `ones(shape)` – creates an array filled with 1 values.
- `arange()` – like Python's built-in `range`

```
>>> np.arange(10)  
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])  
>>> np.arange(2, 10, dtype=np.float)  
array([ 2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])  
>>> np.arange(2, 3, 0.2)  
array([ 2. ,  2.2,  2.4,  2.6,  2.8])
```

NUMPY ARRAYS

`linspace()` – creates arrays with a specified number of elements, and spaced equally between the specified beginning and end values.

```
>>> np.linspace(1., 4., 6)
array([ 1. , 1.6, 2.2, 2.8, 3.4, 4. ])
```

`random.random(shape)` – creates arrays with random floats over the interval $[0,1)$.

```
>>> np.random.random((2,3))
array([[ 0.75688597,  0.41759916,  0.35007419],
       [ 0.77164187,  0.05869089,  0.98792864]])
```

NUMPY ARRAYS

Printing an array can
be done with the print

- statement (Python 2)
- function (Python 3)

```
>>> import numpy as np
>>> a = np.arange(3)
>>> print(a)
[0 1 2]
>>> a
array([0, 1, 2])
>>> b = np.arange(9).reshape(3,3)
>>> print(b)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> c =
np.arange(8).reshape(2,2,2)
>>> print(c)
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

INDEXING

Single-dimension indexing is accomplished as usual.

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
```

Multi-dimensional arrays support multi-dimensional indexing.

```
>>> x.shape = (2,5) # now x is 2-dimensional
>>> x[1,3]
8
>>> x[1,-1]
9
```

INDEXING

Using fewer dimensions to index will result in a subarray:

```
>>> x = np.arange(10)
>>> x.shape = (2,5)
>>> x[0]
array([0, 1, 2, 3, 4])
```

This means that $x[i, j] == x[i][j]$ but the second method is less efficient.

INDEXING

Slicing is possible just as it is for typical Python sequences:

```
>>> x = np.arange(10)
>>> x[2:5]
array([2, 3, 4])
>>> x[: -7]
array([0, 1, 2])
>>> x[1:7:2]
array([1, 3, 5])
>>> y = np.arange(35).reshape(5,7)
>>> y[1:5:2, : :3]
array([[ 7, 10, 13], [21, 24, 27]])
```

ARRAY OPERATIONS

Basic operations apply element-wise. The result is a new array with the resultant elements.

```
>>> a = np.arange(5)
>>> b = np.arange(5)
>>> a+b
array([0, 2, 4, 6, 8])
>>> a-b
array([0, 0, 0, 0, 0])
>>> a**2
array([ 0,  1,  4,  9, 16])
>>> a>3
array([False, False, False, False,  True], dtype=bool)
>>> 10*np.sin(a)
array([ 0.,  8.41470985,  9.09297427,  1.41120008, -
 7.56802495])
>>> a*b
array([ 0,  1,  4,  9, 16])
```

ARRAY OPERATIONS

Since multiplication is done element-wise, you need to specifically perform a dot product to perform matrix multiplication.

```
>>> a = np.zeros(4).reshape(2,2)
>>> a
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> a[0,0] = 1
>>> a[1,1] = 1
>>> b = np.arange(4).reshape(2,2)
>>> b
array([[0, 1],
       [2, 3]])
>>> a*b
array([[ 0.,  0.],
       [ 0.,  3.]])
>>> np.dot(a,b)
array([[ 0.,  1.],
       [ 2.,  3.]])
```

ARRAY OPERATIONS

There are also some built-in methods of ndarray objects.

Universal functions which may also be applied include `exp`, `sqrt`, `add`, `sin`, `cos`, etc.

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.68166391,  0.98943098,
         0.69361582],
       [ 0.78888081,  0.62197125,
         0.40517936]])
>>> a.sum()
4.1807421388722164
>>> a.min()
0.4051793610379143
>>> a.max(axis=0)
array([ 0.78888081,  0.98943098,
        0.69361582])
>>> a.min(axis=1)
array([ 0.68166391,  0.40517936])
```

ARRAY OPERATIONS

An array shape can be manipulated by a number of methods.

`resize(size)` will modify an array in place.

`reshape(size)` will return a copy of the array with a new shape.

```
>>> a =
np.floor(10*np.random.random((3,4)))
>>> print(a)
[[ 9.  8.  7.  9.]
 [ 7.  5.  9.  7.]
 [ 8.  2.  7.  5.]]
>>> a.shape
(3, 4)
>>> a.ravel()
array([ 9.,  8.,  7.,  9.,  7.,  5.,  9.,
        7.,  8.,  2.,  7.,  5.])
>>> a.shape = (6,2)
>>> print(a)
[[ 9.  8.]
 [ 7.  9.]
 [ 7.  5.]
 [ 9.  7.]
 [ 8.  2.]
 [ 7.  5.]]
>>> a.transpose()
array([[ 9.,  7.,  7.,  9.,  8.,  7.],
       [ 8.,  9.,  5.,  7.,  2.,  5.]])
```

LINEAR ALGEBRA

One of the most common reasons for using the NumPy package is its linear algebra module.

It's like Matlab, but free!

```
>>> from numpy import *
>>> from numpy.linalg import *
>>> a = array([[1.0, 2.0],
              [3.0, 4.0]])

>>> print(a)
[[ 1.  2.]
 [ 3.  4.]]

>>> a.transpose()
array([[ 1.,  3.],
       [ 2.,  4.]])

>>> inv(a) # inverse
array([[ -2. ,  1. ],
       [ 1.5, -0.5]])
```

LINEAR ALGEBRA

```
>>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> j = array([[0.0, -1.0], [1.0, 0.0]])
>>> dot(j, j) # matrix product
array([[ -1.,  0.],
       [  0., -1.]])
>>> trace(u) # trace (sum of elements on diagonal)
2.0
>>> y = array([[5.], [7.]])
>>> solve(a, y) # solve linear matrix equation
array([[ -3.],
       [  4.]])
>>> eig(j) # get eigenvalues/eigenvectors of matrix
(array([ 0.+1.j, 0.-1.j]),
 array([[ 0.70710678+0.j, 0.70710678+0.j],
       [ 0.00000000-0.70710678j,
        0.00000000+0.70710678j]]))
```

SCIPY?



In its own words:

SciPy is a collection of mathematical algorithms and convenience functions **built on the NumPy extension** of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data.

Basically, SciPy contains various tools and functions for solving common problems in **scientific computing.**

SCIPY

SciPy gives you access to a ton of specialized mathematical functionality.

- **Just know it exists.** We won't use it much in this class.

Some functionality:

- Special mathematical functions (`scipy.special`) -- elliptic, bessel, etc.
- Integration (`scipy.integrate`)
- Optimization (`scipy.optimize`)
- Interpolation (`scipy.interpolate`)
- Fourier Transforms (`scipy.fftpack`)
- Signal Processing (`scipy.signal`)
- Linear Algebra (`scipy.linalg`)
- Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Spatial data structures and algorithms (`scipy.spatial`)
- Statistics (`scipy.stats`)
- Multidimensional image processing (`scipy.ndimage`)
- Data IO (`scipy.io`) – overlaps with pandas, covers some other formats

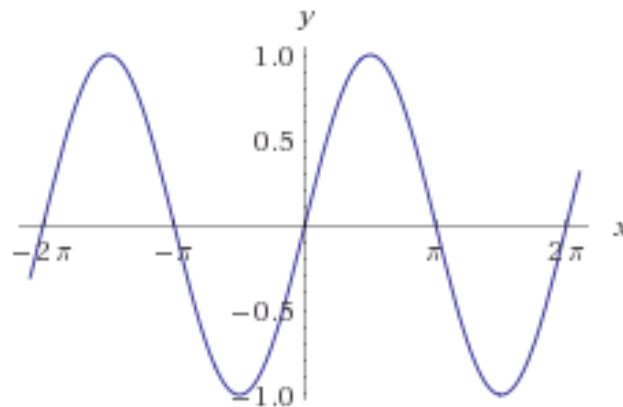
ONE SCIPY EXAMPLE

We can't possibly tour all of the SciPy library and, even if we did, it might be a little boring.

- Often, you'll be able to find higher-level modules that will work around your need to directly call low-level SciPy functions

Say you want to compute an integral:

$$\int_a^b \sin x \, dx$$



SCIPY.INTEGRATE

We have a function object – `np.sin` defines the sin function for us.

We can compute the definite integral from $x = 0$ to $x = \pi$ using the quad function.

```
>>> res = scipy.integrate.quad(np.sin, 0, np.pi)
>>> print(res)
(2.0, 2.220446049250313e-14) # 2 with a very small error
margin!
>>> res = scipy.integrate.quad(np.sin, -np.inf, +np.inf)
>>> print(res)
(0.0, 0.0) # Integral does not converge
```

SCIPY.INTEGRATE

Let's say that we don't have a function object, we only have some (x,y) samples that "define" our function.

We can estimate the integral using the trapezoidal rule.

```
>>> sample_x = np.linspace(0, np.pi, 1000)
>>> sample_y = np.sin(sample_x) # Creating 1,000 samples
>>> result = scipy.integrate.trapz(sample_y, sample_x)
>>> print(result)
1.99999835177

>>> sample_x = np.linspace(0, np.pi, 1000000)
>>> sample_y = np.sin(sample_x) # Creating 1,000,000
samples
>>> result = scipy.integrate.trapz(sample_y, sample_x)
>>> print(result)
2.0
```

WRAP UP

Shift thinking from imperative coding to operations on datasets

Numpy: A low-level abstraction that gives us really fast multi-dimensional arrays

In a bit:

Pandas: Higher-level tabular abstraction and operations to manipulate and combine tables

Reading Homework focuses on Pandas and SQL: Aim to release by tonight, probably by tomorrow

REST OF TODAY'S LECTURE

By popular request ...

- **Version control** primer!
- Specifically, git via GitHub and GitLab
- Thanks: Mark Groves (Microsoft), Ilan Biala & Aaron Perley (CMU), Sharif U., & the HJCB Senior Design Team!

And then a bit on keeping your data ... **tidy data**.



WHAT IS VERSION CONTROL?

```
Aaron@HELIOS ~/112_term_project
$ ls
termproject_actually_final  termproject_v10  termproject_v3
termproject_final          termproject_v11  termproject_v4
termproject_handin         termproject_v12  termproject_v5
termproject_old_idea       termproject_v13  termproject_v6
termproject_superfrogger   termproject_v14  termproject_v7
termproject_temp           termproject_v15  termproject_v8
termproject_this_one_works termproject_v16  termproject_v9
termproject_v1             termproject_v2
```

DEVELOPMENT TOOL

When working with a team, the need for a central repository is essential

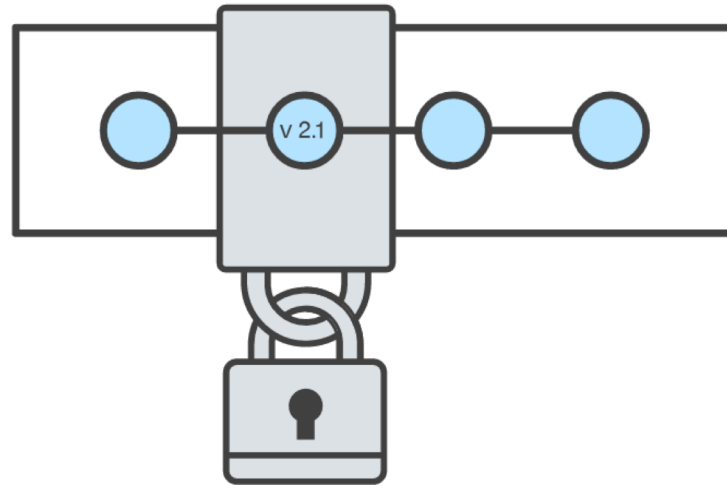
- Need a system to allow versioning, and a way to acquire the latest edition of the code
- A system to track and manage bugs was also needed

GOALS OF VERSION CONTROL

Be able to search through revision history and retrieve previous versions of any file in a project

Be able to share changes with collaborators on a project

Be able to confidently make large changes to existing files



atlassian.com/git/tutorials/what-is-version-control

NAMED FOLDERS APPROACH

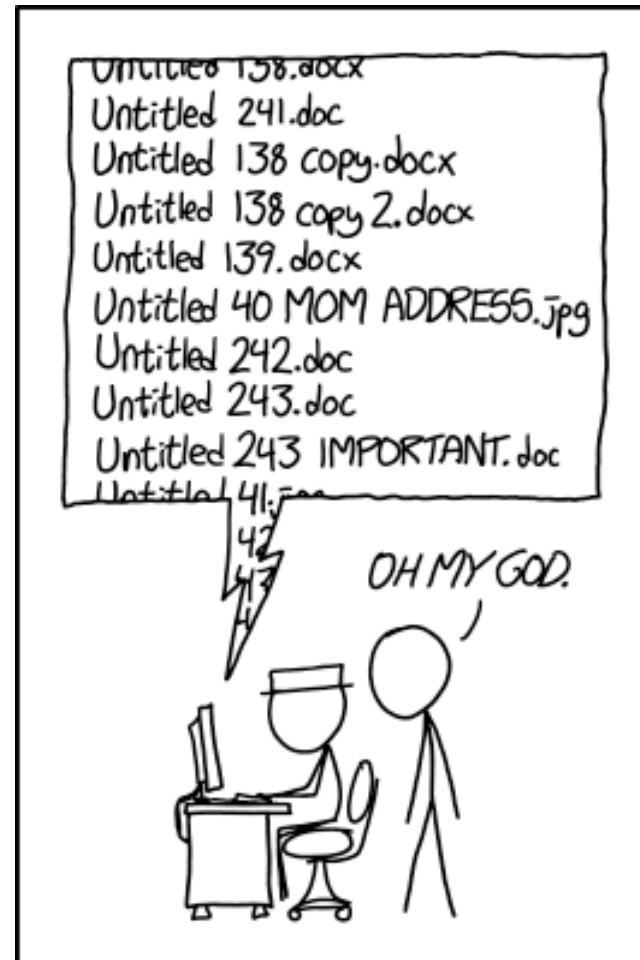
Can be hard to track

Memory-intensive

Can be slow

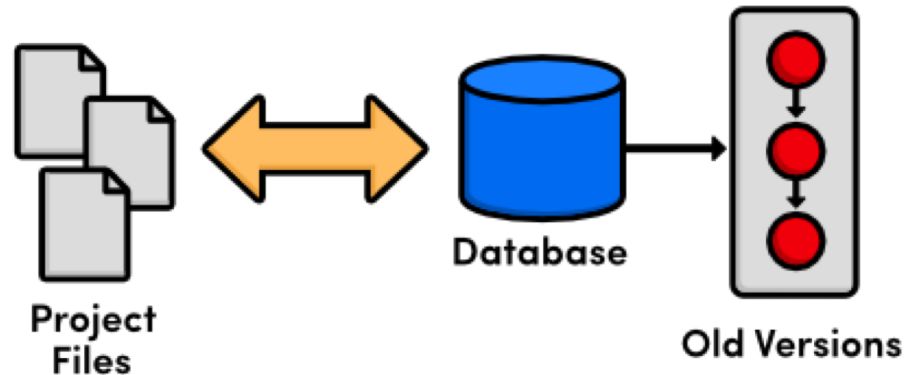
Hard to share

No record of authorship



PRO TIP: NEVER LOOK IN SOMEONE
ELSE'S DOCUMENTS FOLDER.

LOCAL DATABASE OF VERSIONS APPROACH



Provides an abstraction over finding the right versions of files and replacing them in the project

Records who changes what, but hard to parse that

Can't share with collaborators

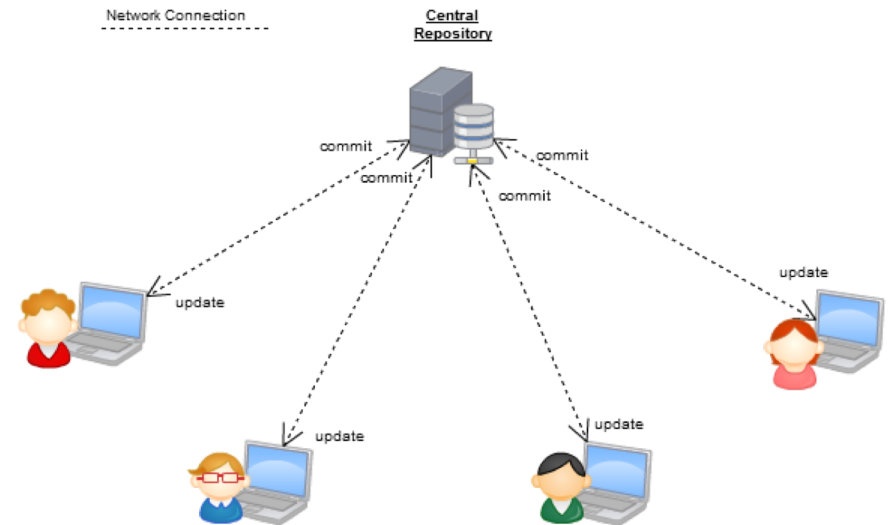
CENTRALIZED VERSION CONTROL SYSTEMS

A central, trusted repository determines the order of commits (“versions” of the project)

Collaborators “push” changes (commits) to this repository.

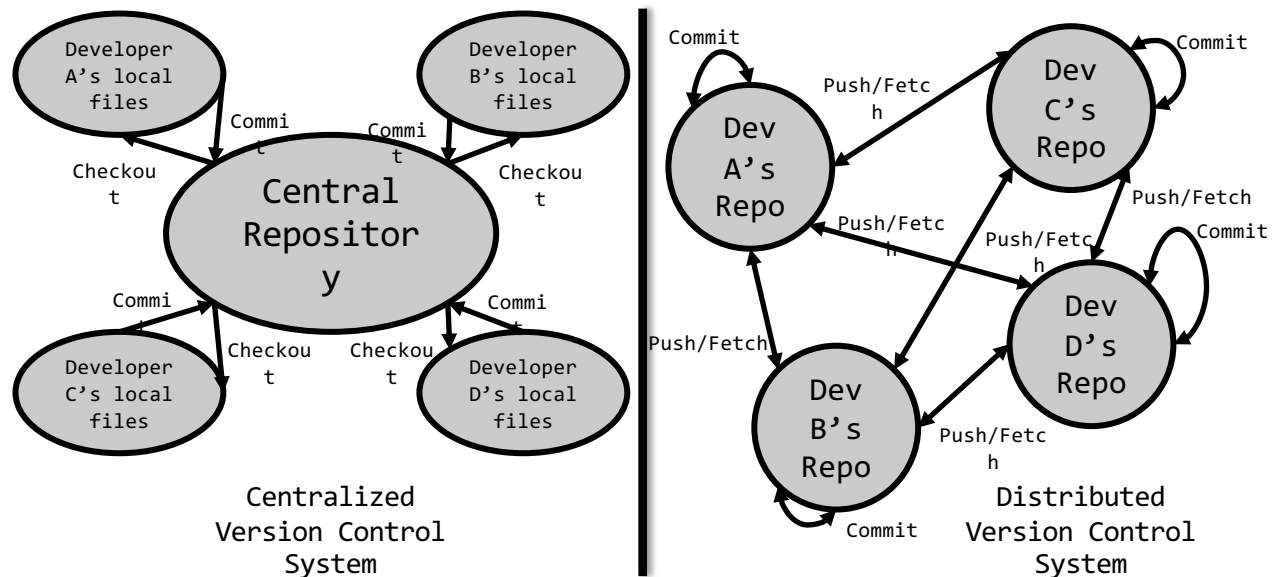
Any new commits must be compatible with the most recent commit. If it isn’t, somebody must “merge” it in.

Examples: SVN, CVS, Perforce



DISTRIBUTED VERSION CONTROL SYSTEMS (DVCS)

- No central repository
- Every repository has every commit
- Examples: **Git**, Mercurial



WHAT IS GIT

Git is a version control system

Developed as a repository system for both local and remote changes

Allows teammates to work simultaneously on a project

Tracks each commit, allowing for a detailed documentation of the project along every step

Allows for advanced merging and branching operations



A SHORT HISTORY OF GIT

Linux kernel development

1991-2002

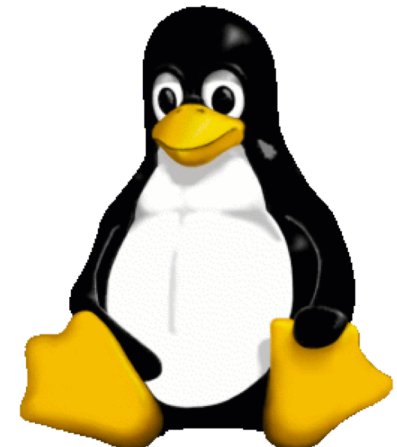
- Changes passed around as archived file

2002-2005

- Using a DVCS called BitKeeper

2005

- Relationship broke down between two communities (BitKeeper licensing issues)



A SHORT HISTORY OF GIT

Goals:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- **Fully distributed** – not a requirement, can be centralized
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

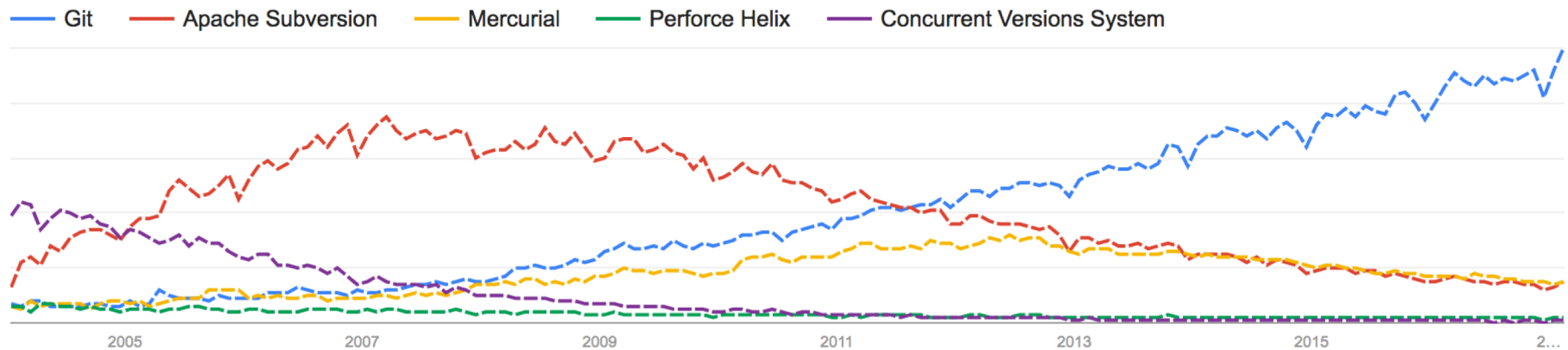
A SHORT HISTORY OF GIT

Popularity:

- Git is now the most widely used source code management tool
- 33.3% of professional software developers use Git (often through GitHub) as their primary source control system

[citation needed]

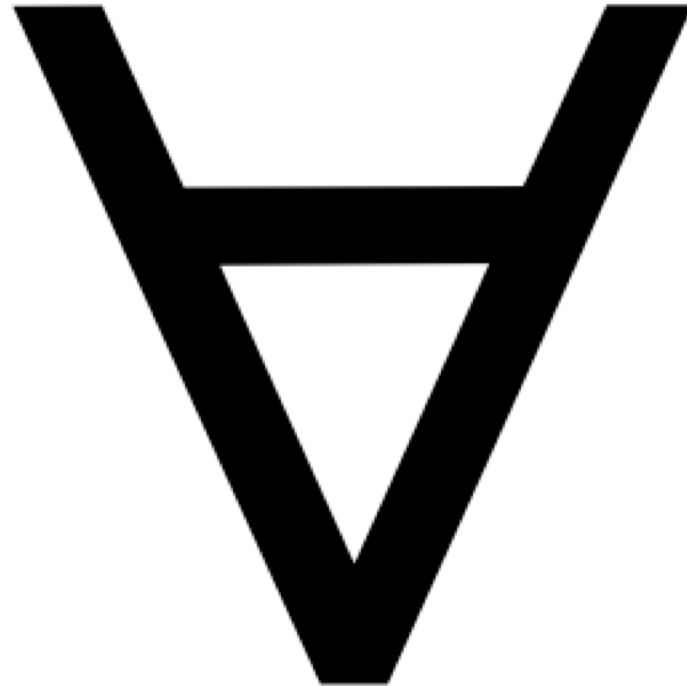
Interest over time. Web Search. Worldwide, 2004 - present.



GIT IN INDUSTRY

Companies and projects currently using Git

- Google
- Android
- Facebook
- Microsoft
- Netflix
- Linux
- Ruby on Rails
- Gnome
- KDE
- Eclipse
- X.org



GIT BASICS

Snapshots, not changes

- A picture of what all your files look like at that moment
- If a file has not changed, store a reference

Nearly every operation is local

- Browsing the history of project
- See changes between two versions

WHY GIT IS BETTER

Git tracks the content rather than the files

Branches are lightweight, and merging is a simple process

Allows for a more streamlined offline development process

Repositories are smaller in size and are stored in a single .git directory

Allows for advanced staging operations, and the use of stashing when working through troublesome sections

GIT VS {CVS, SVN, ...}

Why you should care:

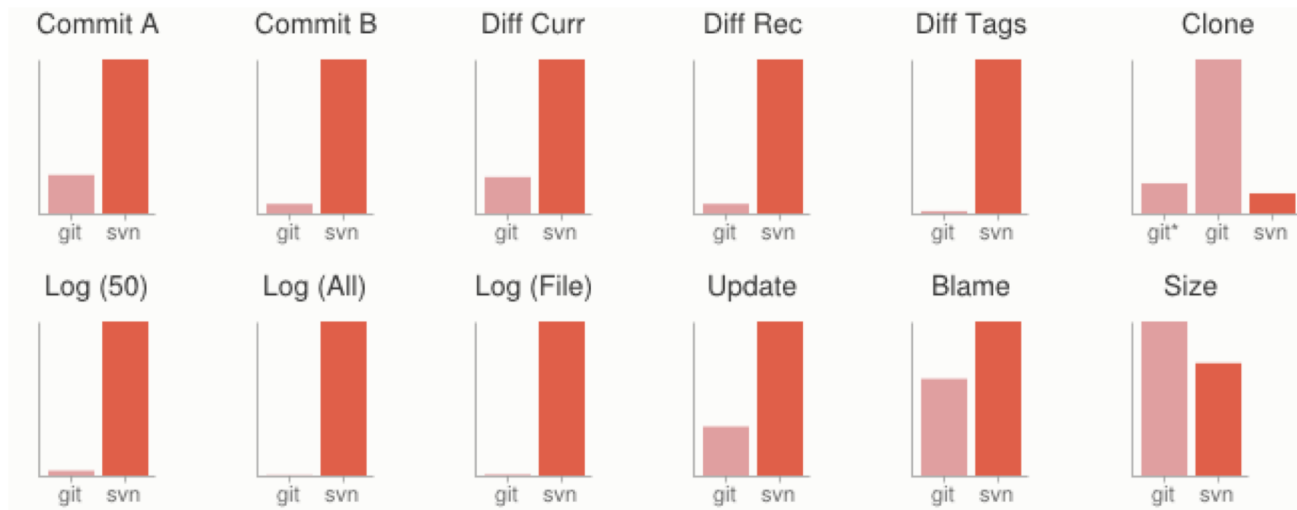
- Many places use legacy systems that will cause problems in the future – be the change you believe in!

Git is **much** faster than SVN:

- Coded in C, which allows for a great amount of optimization
- Accomplishes much of the logic client side, thereby reducing time needed for communication
- Developed to work on the Linux kernel, so that large project manipulation is at the forefront of the benchmarks

GIT VS {CVS, SVN, ...}

Speed benchmarks:



Benchmarks performed by <http://git-scm.com/about/small-and-fast>

GIT VS {CVS, SVN, ...}

Git is significantly smaller than SVN

- All files are contained in a small decentralized .git file
- In the case of Mozilla's projects, a Git repository was 30 times smaller than an identical SVN repository
- Entire Linux kernel with 5 years of versioning contained in a single 1 GB .git file
- SVN carries two complete copies of each file, while Git maintains a simple and separate 100 bytes of data per file, noting changes and supporting operations

Nice because you can (and do!) store the whole thing locally



GIT VS {CVS, SVN, ...}

Git is more **secure** than SVN

- All commits are uniquely hashed for both security and indexing purposes
- Commits can be authenticated through numerous means
 - In the case of SSH commits, a key may be provided by both the client and server to guarantee authenticity and prevent against unauthorized access

GIT VS {CVS, SVN, ...}

Git is decentralized:

- Each user contains an individual repository and can check commits against itself, allowing for detailed local revisioning
- Being decentralized allows for easy replication and deployment
- In this case, SVN relies on a single centralized repository and is unusable without

GIT VS {CVS, SVN, ...}

Git is **flexible**:

- Due to its decentralized nature, git commits can be stored locally, or committed through HTTP, SSH, FTP, or even by Email
- No need for a centralized repository
- Developed as a command line utility, which allows a large amount of features to be built and customized on top of it

GIT VS {CVS, SVN, ...}

Data assurance: a checksum is performed on both upload and download to ensure sure that the file hasn't been corrupted.

Commit IDs are generated upon each commit:

- Linked list style of commits
- Each commit is linked to the next, so that if something in the history was changed, each following commit will be rebranded to indicate the modification

GIT VS {CVS, SVN, ...}

Branching:

- Git allows the usage of advanced **branching** mechanisms and procedures
- Individual divisions of the code can be separated and developed separately within separate branches of the code
- Branches can allow for the separation of work between developers, or even for disposable experimentation
- Branching is a precursor and a component of the merging process

Will give an example shortly.

GIT VS {CVS, SVN, ...}

Merging

- The process of merging is directly related to the process of branching
- Individual branches may be merged together, solving code conflicts, back into the default or master branch of the project
- Merges are usually done automatically, unless a conflict is presented, in which case the user is presented with several options with which to handle the conflict

Will give an example shortly.

GIT VS {CVS, SVN, ...}

Merging: content of the files is tracked rather than the file itself:

- This allows for a greater element of tracking and a smarter and more automated process of merging
- SVN is unable to accomplish this, and will throw a conflict if, e.g., a file name is changed and differs from the name in the central repository
- Git is able to solve this problem with its use of managing a local repository and tracking individual changes to the code

INITIALIZATION OF A GIT REPOSITORY

```
C:\> mkdir CoolProject
C:\> cd CoolProject
C:\CoolProject > git init
Initialized empty Git repository in
C:/CoolProject/.git
C:\CoolProject > notepad README.txt
C:\CoolProject > git add .
C:\CoolProject > git commit -m 'my first
commit'
[master (root-commit) 7106a52] my first commit
1 file changed, 1 insertion(+)
create mode 100644 README.txt
```



GIT BASICS I

The three (or four) states of a **file**:

- **Modified:**
 - File has changed but not committed
- **Staged:**
 - Marked to go to next commit snapshot
- **Committed:**
 - Safely stored in local database
- **Untracked!**
 - Newly added or removed files

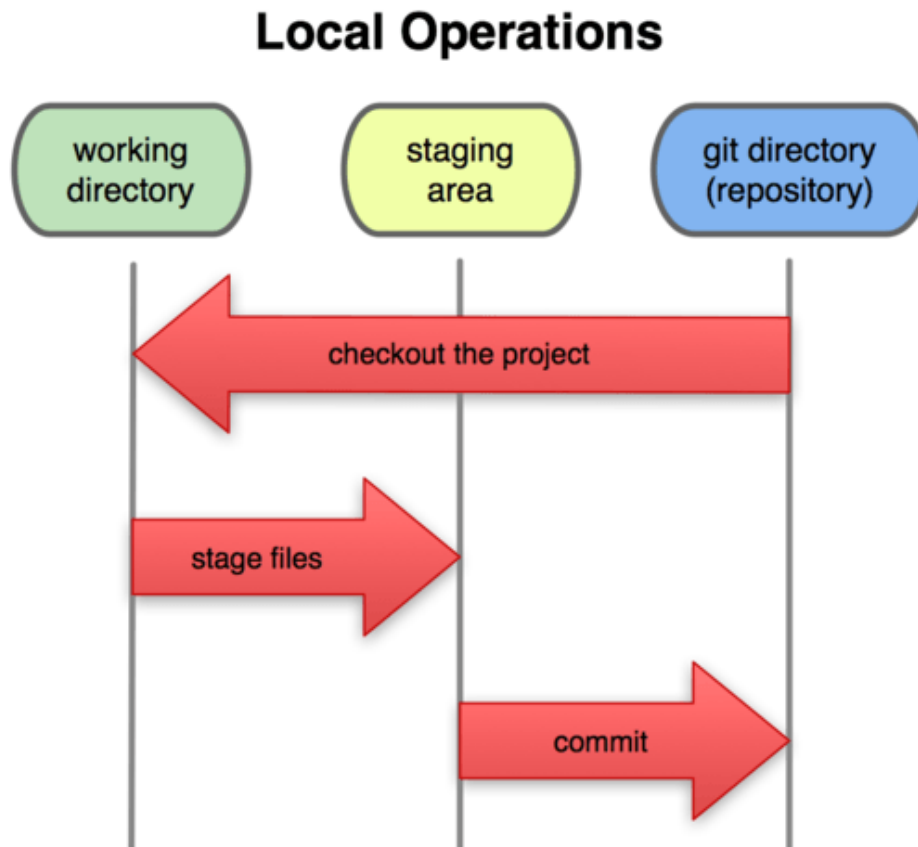
GIT BASICS II

Three main areas of a git **project**:

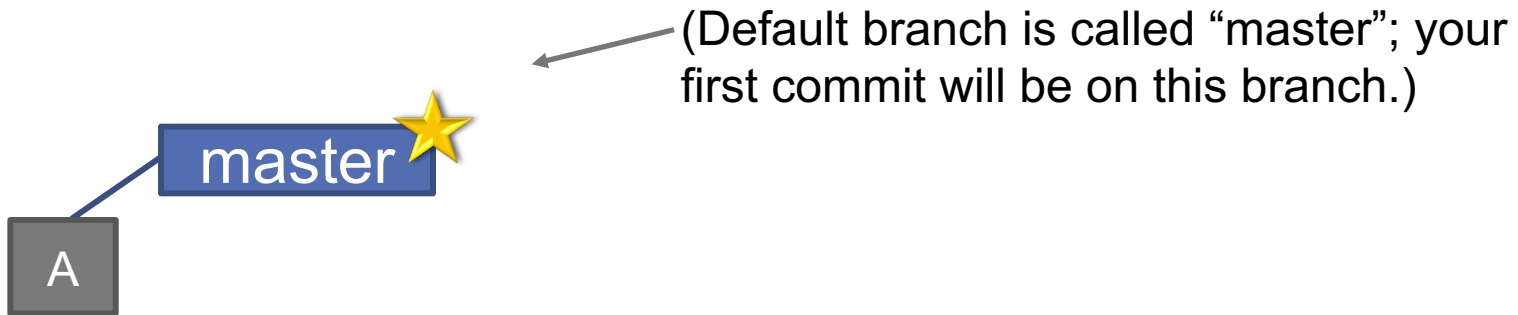
- **Working directory**
 - Single checkout of one version of the project.
- **Staging area**
 - Simple file storing information about what will go into your next commit
- **Git directory**
 - What is copied when cloning a repository

GIT BASICS III

Three main areas of a git **project**:

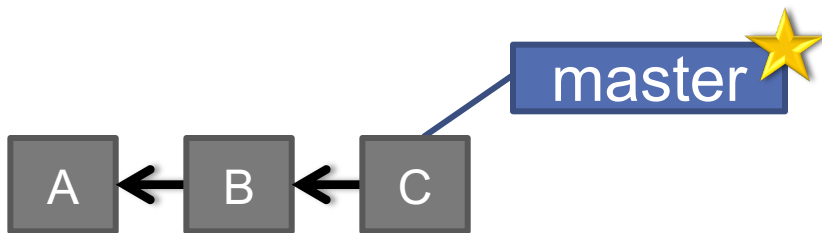


BRANCHES ILLUSTRATED



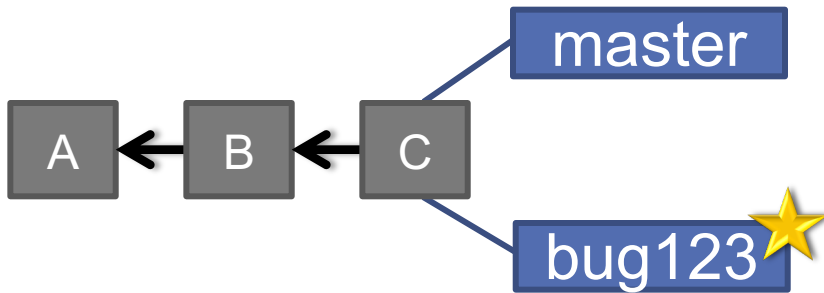
```
> git commit -m 'my first commit'
```

BRANCHES ILLUSTRATED



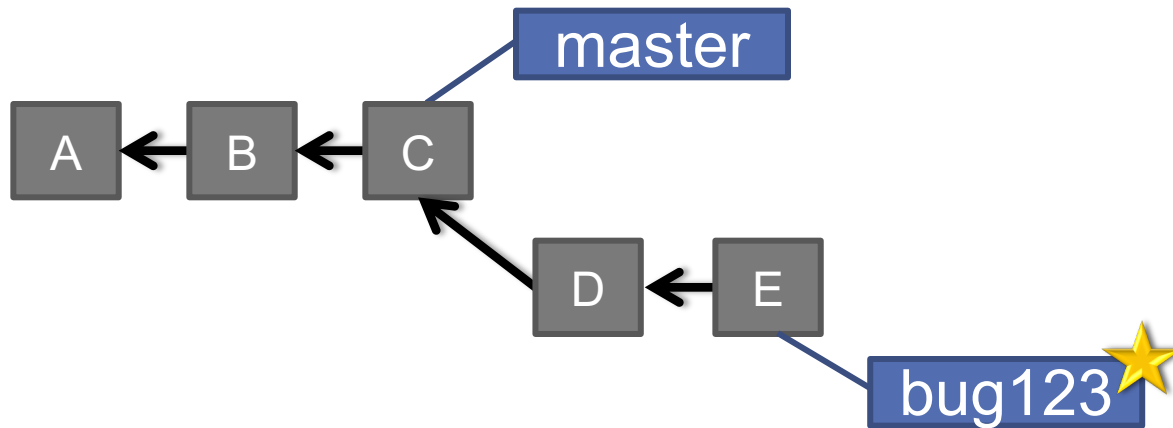
```
> git commit (x2)
```

BRANCHES ILLUSTRATED



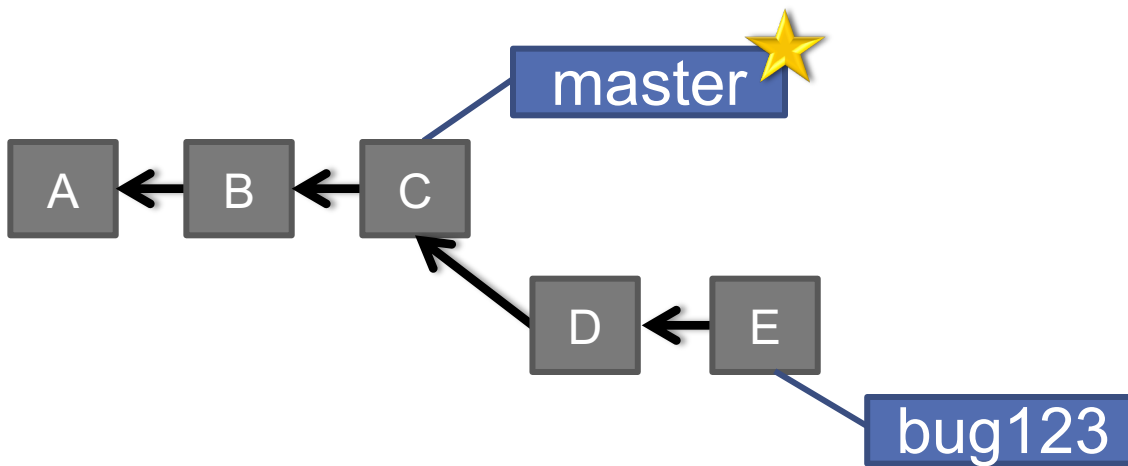
```
> git checkout -b bug123
```

BRANCHES ILLUSTRATED



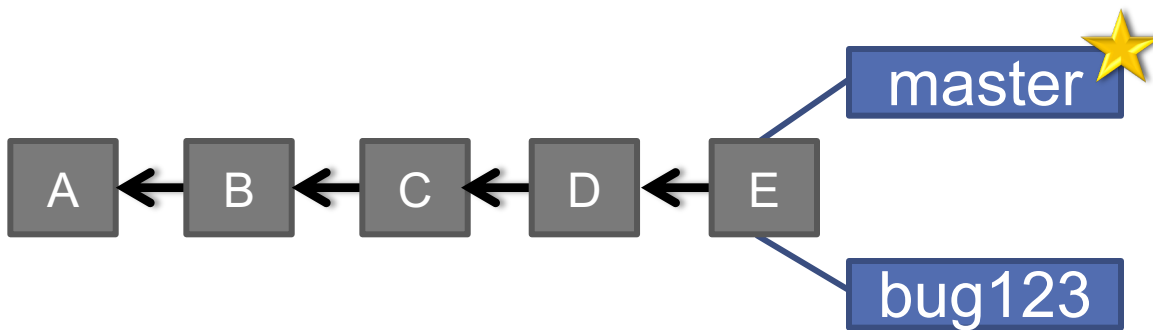
```
> git commit (x2)
```

BRANCHES ILLUSTRATED



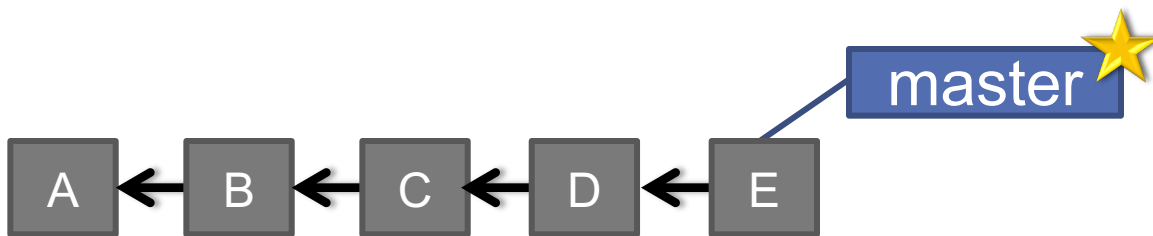
```
> git checkout master
```

BRANCHES ILLUSTRATED



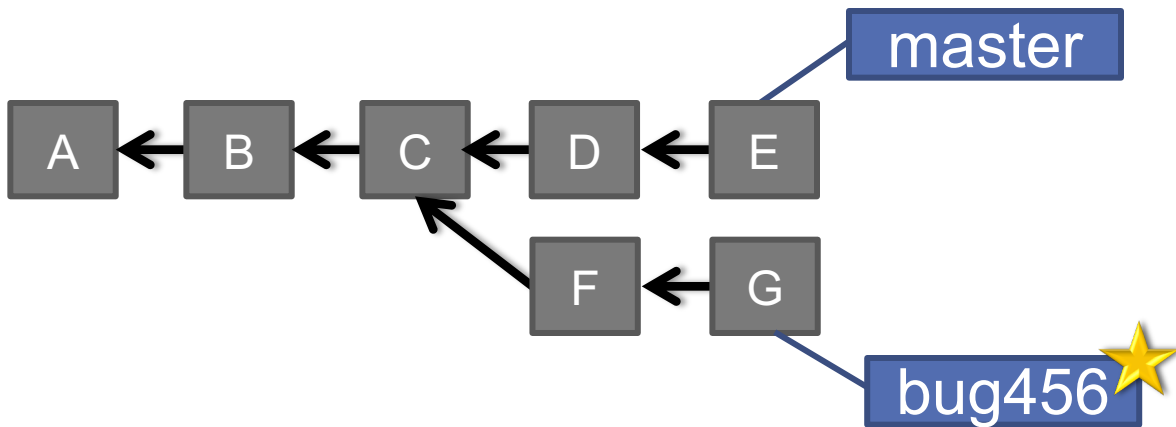
```
> git merge bug123
```


BRANCHES ILLUSTRATED

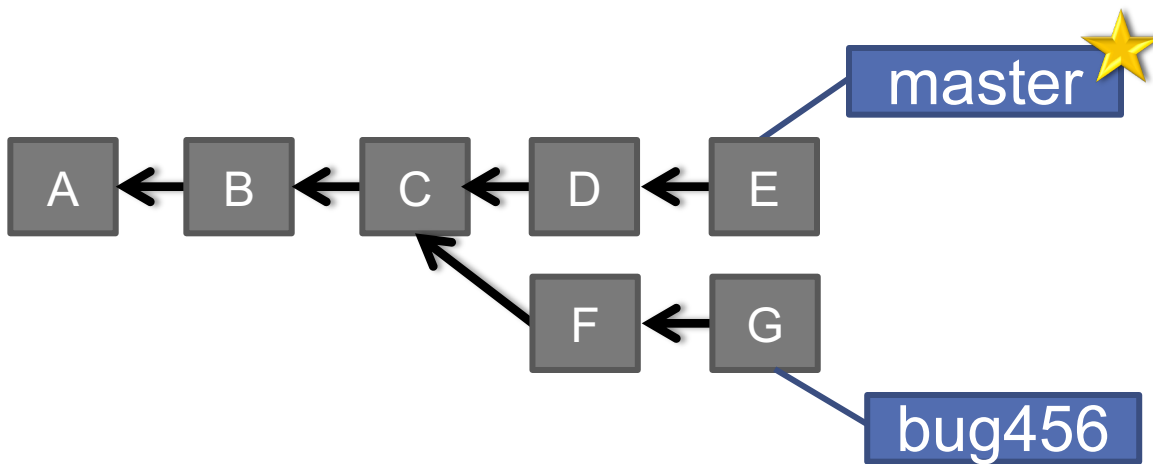


```
> git branch -d bug123
```

BRANCHES ILLUSTRATED

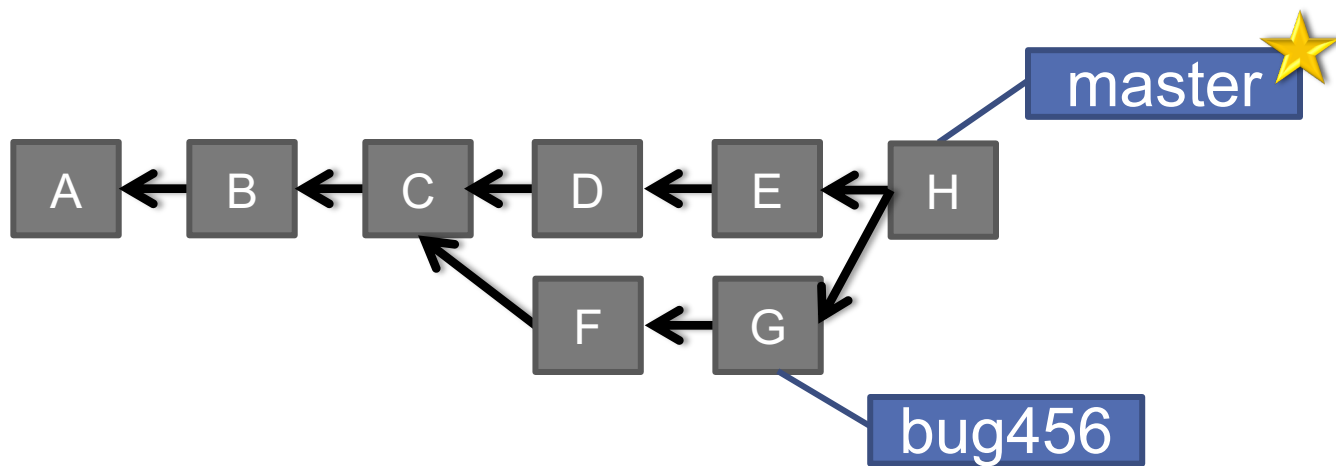


BRANCHES ILLUSTRATED



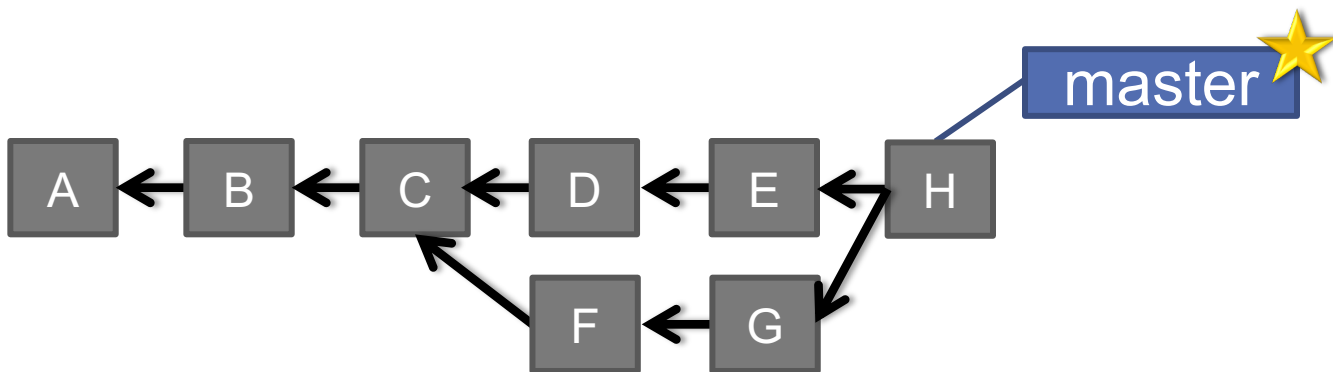
> git checkout master

BRANCHES ILLUSTRATED



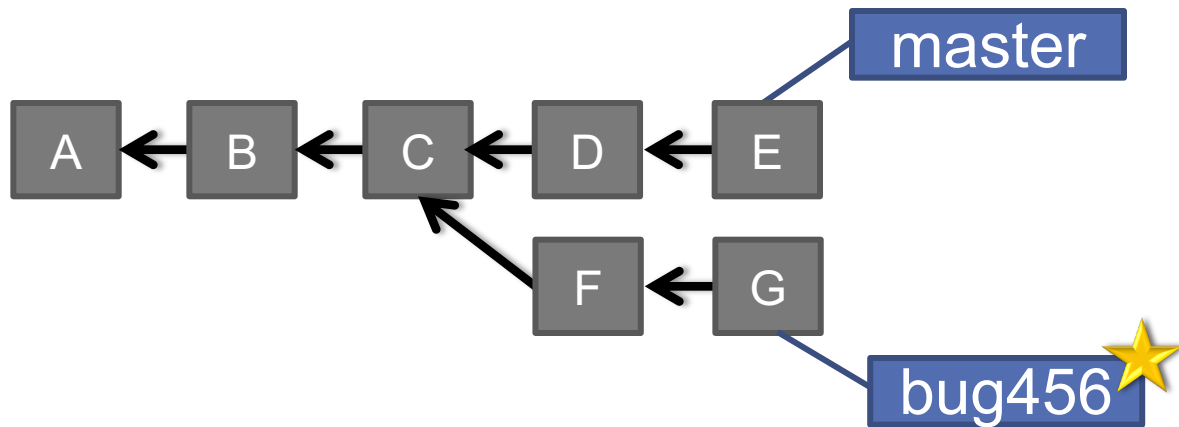
```
> git merge bug456
```

BRANCHES ILLUSTRATED

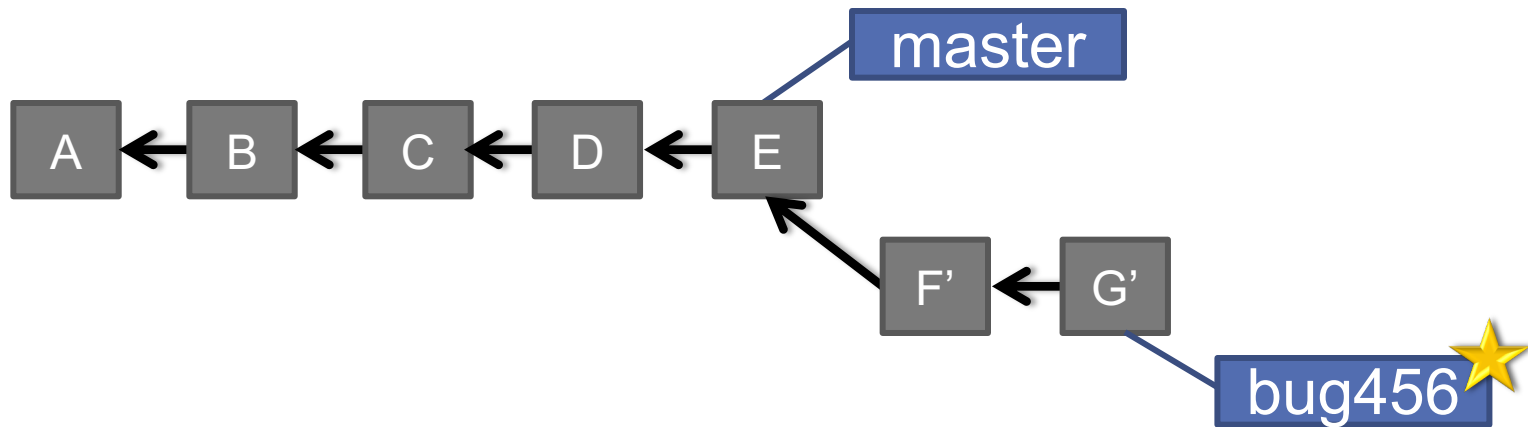


```
> git branch -d bug456
```

BRANCHES ILLUSTRATED

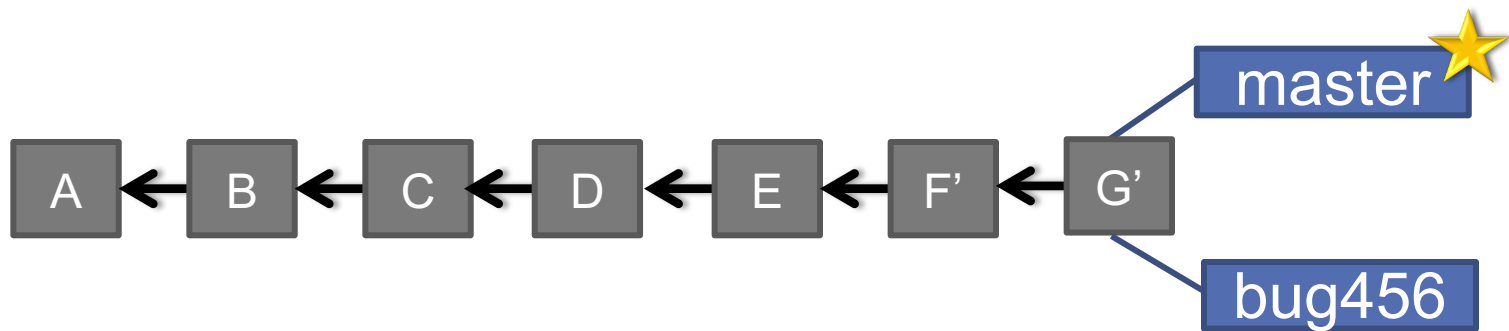


BRANCHES ILLUSTRATED



```
> git rebase master
```

BRANCHES ILLUSTRATED



- > `git checkout master`
- > `git merge bug456`

WHEN TO BRANCH?

General rule of thumb:

- **Anything in the master branch is always deployable.**

Local branching is very lightweight!

- New feature? Branch!
- Experiment that you won't ever deploy? Branch!

Good habits:

- Name your branch something descriptive (add-like-button, refactor-jobs, create-ai-singularity)
- Make your commit messages descriptive, too!



SO YOU WANT SOMEBODY ELSE TO HOST THIS FOR YOU ...

Git: general distributed version control system

GitHub / BitBucket / GitLab / ...: **hosting** services for git repositories

In general, GitHub is the most popular:

- Lots of big projects (e.g., Python, Bootstrap, Angular, D3, node, Django, Visual Studio)
- Lots of ridiculously awesome projects (e.g., <https://github.com/maxbbraun/trump2cash>)

There are reasons to use the competitors (e.g., private repositories, access control)



Bitbucket






“SOCIAL CODING”



John P. Dickerson
JohnDickerson

Assistant Professor of Computer Science, University of Maryland; Ph.D. in Computer Science, Carnegie Mellon University

 University of Maryland
 Washington, DC
 <http://jpdickerson.com>

Organizations



Overview Repositories 14 **Stars 71** Followers 34 Following 47

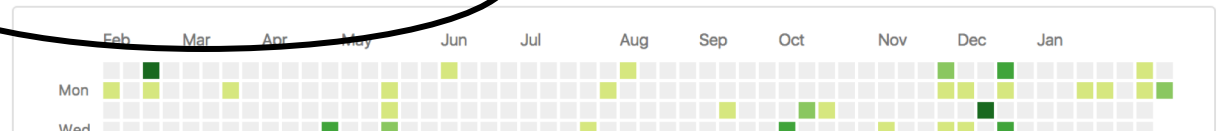
Popular repositories

Customize your pinned repositories

<p>KidneyExchange Kidney paired donation optimization code</p> <p>● Java ★ 8 🍴 7</p>	<p>TrackIt Modular suite that measures a child's sustained selective attention.</p> <p>● Java ★ 3 🍴 2</p>
<p>EnvyFree Computes envy-free allocations of items to agents.</p> <p>● Python ★ 2</p>	<p>VotingRules Compute winners in elections based on various voting rules.</p> <p>● Java ★ 1</p>
<p>muffins Fairly feeding hungry students</p> <p>● Python ★ 1</p>	<p>website My academic website.</p> <p>● TeX</p>

75 contributions in the last year

Contribution settings ▾



REVIEW: HOW TO USE

Git commands for everyday usage are relatively simple

- **git pull**
 - Get the latest changes to the code
- **git add .**
 - Add any newly created files to the repository for tracking
- **git add -u**
 - Remove any deleted files from tracking and the repository
- **git commit -m 'Changes'**
 - Make a version of changes you have made
- **git push**
 - Deploy the latest changes to the central repository

Make a repo on GitHub and **clone** it to your machine:

- <https://guides.github.com/activities/hello-world/>

STUFF TO CLICK ON

Git

- <http://git-scm.com/>

GitHub

- <https://github.com/>
- <https://guides.github.com/activities/hello-world/>
- **^-- Just do this one. You'll need it for your tutorial 😊.**

GitLab

- <http://gitlab.org/>

Git and SVN Comparison

- <https://git.wiki.kernel.org/index.php/GitSvnComparison>



TODAY/NEXT CLASS

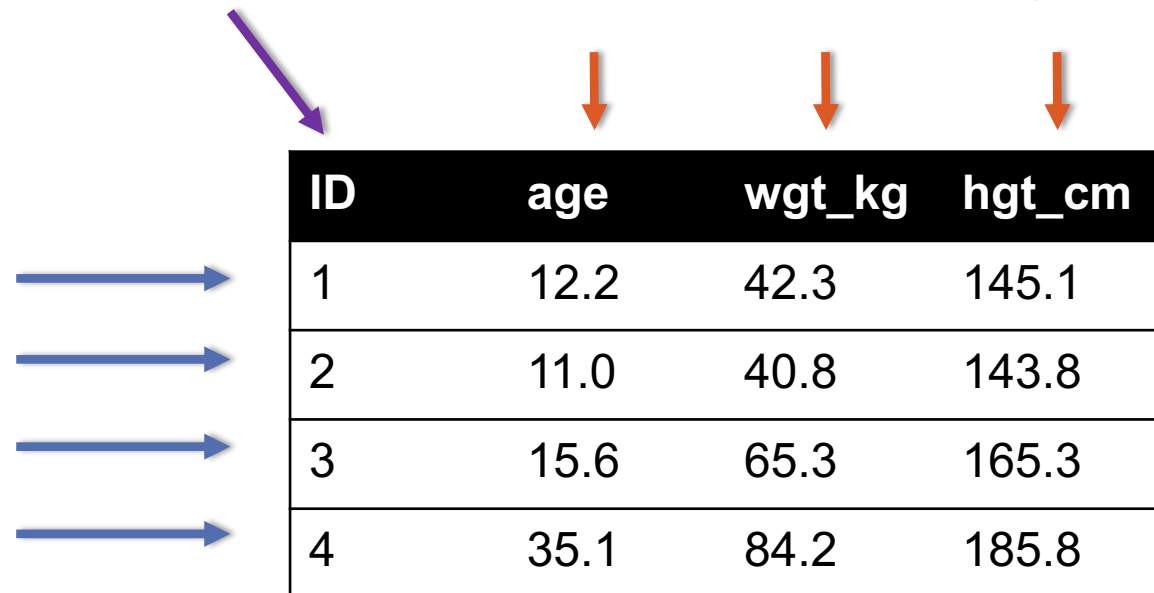
- **Tables**
 - Abstraction
 - Operations
- **Pandas**
- **Tidy Data**
- **SQL**

TABLES

Special Column, called “Index”, or
“ID”, or “Key”
Usually, no duplicates Allowed

Variables
(also called Attributes, or
Columns, or Labels)

Observations,
Rows, or
Tuples



ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

TABLES

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

ID	Address
1	College Park, MD, 20742
2	Washington, DC, 20001
3	Silver Spring, MD 20901

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245

unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985

199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085

1. SELECT/SLICING

Select only some of the rows, or some of the columns, or a combination

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Only rows
with wgt > 41

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Only columns
ID and Age

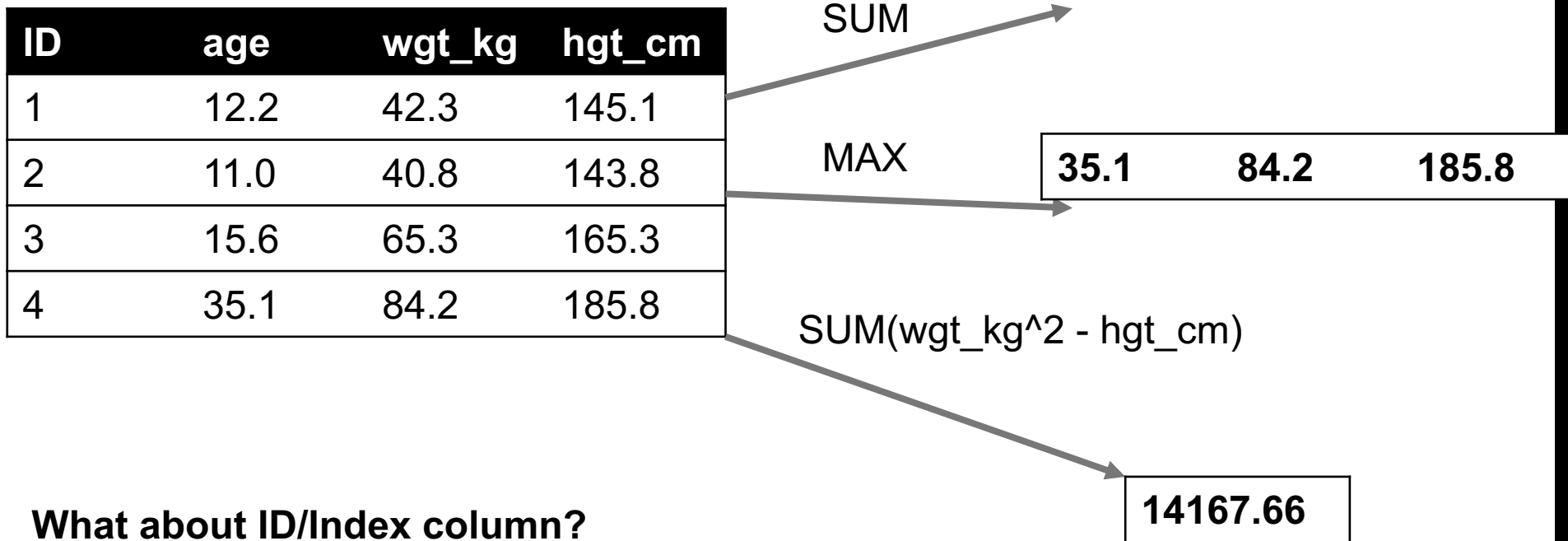
ID	age
1	12.2
2	11.0
3	15.6
4	35.1

Both

ID	age
1	12.2
3	15.6
4	35.1

2. AGGREGATE/REDUCE

Combine values across a column into a single value



What about ID/Index column?

Usually not meaningful to aggregate across it
May need to explicitly add an ID column

3. MAP

Apply a function to every row, possibly creating more or fewer columns

ID	Address
1	College Park, MD, 20742
2	Washington, DC, 20001
3	Silver Spring, MD 20901



ID	City	State	Zipcode
1	College Park	MD	20742
2	Washington	DC	20001
3	Silver Spring	MD	20901

Variations that allow one row to generate multiple rows in the output (sometimes called “flatmap”)

4. GROUP BY

Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'A' →

A = foo

ID	B	C
1	3	6.6
3	4	3.1
4	3	8.0
7	4	2.3
8	3	8.0

A = bar

ID	B	C
2	2	4.7
5	1	1.2
6	2	2.5

4. GROUP BY

Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'B' →

B = 1

ID	A	C
5	bar	1.2

B = 2

ID	A	C
2	bar	4.7
6	bar	2.5

B = 3

ID	A	C
1	foo	6.6
4	foo	8.0
8	foo	8.0

B = 4

ID	A	C
3	foo	3.1
7	foo	2.3

4. GROUP BY

Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'A', 'B'



A = bar, B = 1

ID	C
5	1.2

A = bar, B = 2

ID	C
2	4.7
6	2.5

A = foo, B = 3

ID	C
1	6.6
4	8.0
8	8.0

A = foo, B = 4

ID	C
3	3.1
7	2.3

5. GROUP BY AGGREGATE

Compute one aggregate

Per group

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group by 'B'
Sum on C

B = 1

ID	A	C
5	bar	1.2

B = 2

ID	A	C
2	bar	4.7
6	bar	2.5

B = 3

ID	A	C
1	foo	6.6
4	foo	8.0
8	foo	8.0

B = 4

ID	A	C
3	foo	3.1
7	foo	2.3

B = 1

Sum (C)
1.2

B = 2

Sum (C)
7.2

B = 3

Sum (C)
22.6

B = 4

Sum (C)
5.4

5. GROUP BY AGGREGATE

Final result usually seen

As a table

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group by 'B'
Sum on C

B = 1

Sum (C)
1.2

B = 2

Sum (C)
7.2

B = 3

Sum (C)
22.6

B = 4

Sum (C)
5.4



B	SUM(C)
1	1.2
2	7.2
3	22.6
4	5.4

6. UNION/INTERSECTION/DIFFERENCE

Set operations – only if the two tables have identical attributes/columns

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0

U

ID	A	B	C
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0



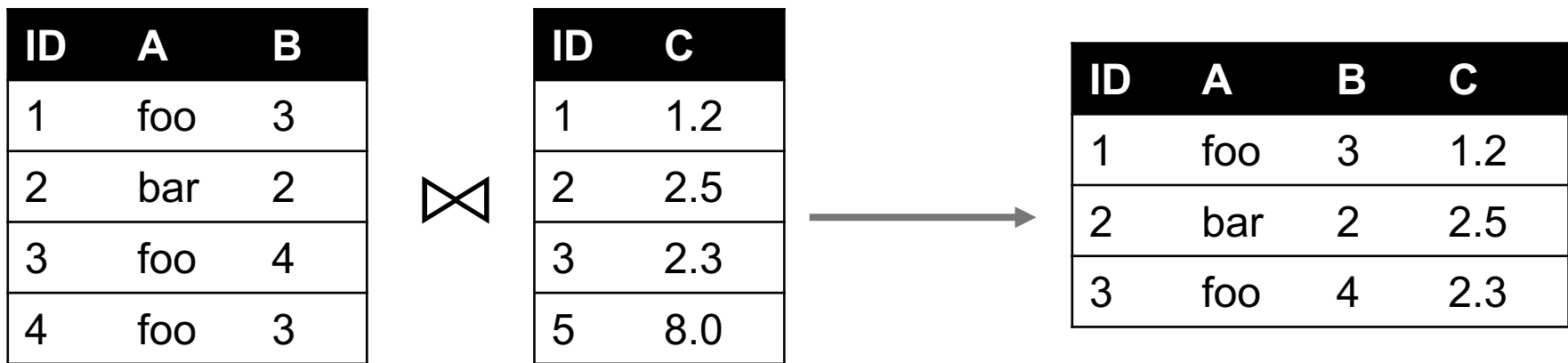
ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Similarly Intersection and Set Difference manipulate tables as Sets

IDs may be treated in different ways, resulting in somewhat different behaviors

7. MERGE OR JOIN

Combine rows/tuples across two tables if they have the same key



What about IDs not present in both tables?

Often need to keep them around

Can “pad” with NaN

7. MERGE OR JOIN

Combine rows/tuples across two tables if they have the same key

Outer joins can be used to "pad" IDs that don't appear in both tables

Three variants: LEFT, RIGHT, FULL

SQL Terminology -- Pandas has these operations as well

ID	A	B
1	foo	3
2	bar	2
3	foo	4
4	foo	3



ID	C
1	1.2
2	2.5
3	2.3
5	8.0



ID	A	B	C
1	foo	3	1.2
2	bar	2	2.5
3	foo	4	2.3
4	foo	3	NaN
5	NaN	NaN	8.0

SUMMARY

- **Tables: A simple, common abstraction**
 - Subsumes a set of “strings” – a common input
- **Operations**
 - Select, Map, Aggregate, Reduce, Join/Merge, Union/Concat, Group By
- **In a given system/language, the operations may be named differently**
 - E.g., SQL uses “join”, whereas Pandas uses “merge”
- **Subtle variations in the definitions, especially for more complex operations**

ID	A	B	C
1	foo	3	6.6
2	baz	2	4.7
3	foo	4	3.1
4	baz	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group By 'A'



**HOW MANY
TUPLES IN THE
ANSWER?**

- A. 1
- B. 3
- C. 5
- D. 8

ID	A	B	C
1	foo	3	6.6
2	baz	2	4.7
3	foo	4	3.1
4	baz	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group By 'A',
'B' →

**HOW MANY
GROUPS IN THE
ANSWER?**

- A. 1
- B. 3
- C. 4
- D. 6

ID	A	B
1	foo	3
2	bar	2
4	foo	4
5	foo	3



ID	C
2	1.2
4	2.5
6	2.3
7	8.0

**HOW MANY
TUPLES IN THE
ANSWER?**

- A. 1
- B. 2
- C. 4
- D. 6

ID	A	B
1	foo	3
2	bar	2
4	foo	4
5	foo	3



ID	C
2	1.2
4	2.5
6	2.3
7	8.0

FULL OUTER JOIN

All IDs will be present in the answer
With NaNs

**HOW MANY
TUPLES IN THE
ANSWER?**

- A. 1
- B. 4
- C. 6
- D. 8

TODAY/NEXT CLASS

- **Tables**
 - Abstraction
 - Operations
- **Pandas**
- **Tidy Data**
- **SQL and Relational Databases**

PANDAS: HISTORY

- **Written by: Wes McKinney**
 - Started in 2008 to get a high-performance, flexible tool to perform quantitative analysis on financial data
- **Highly optimized for performance, with critical code paths written in Cython or C**
- **Key constructs:**
 - Series (like a NumPy Array)
 - DataFrame (like a Table or Relation, or R data.frame)
- **Foundation for Data Wrangling and Analysis in Python**

PANDAS: SERIES

index **values**

A	→	5
B	→	6
C	→	12
D	→	-5
E	→	6.7

- Subclass of `numpy.ndarray`
- Data: any type
- Index labels need not be ordered
- Duplicates possible but result in reduced functionality

PANDAS: DATAFRAME

	columns	foo	bar	baz	qux
index					
A	→	0	x	2.7	True
B	→	4	y	6	True
C	→	8	z	10	False
D	→	-12	w	NA	False
E	→	16	a	18	False

- Each column can have a different type
- Row and Column index
- Mutable size: insert and delete columns

- **Note the use of word “index” for what we called “key”**
 - Relational databases use “index” to mean something else

- **Non-unique index values allowed**
 - May raise an exception for some operations

HIERARCHICAL INDEXES

Sometimes more intuitive organization of the data

Makes it easier to understand and analyze higher-dimensional data

e.g., instead of 3-D array, may only need a 2-D array

day		Fri	Sat	Sun	Thur
sex	smoker				
Female	No	3.125	2.725	3.329	2.460
	Yes	2.683	2.869	3.500	2.990
Male	No	2.500	3.257	3.115	2.942
	Yes	2.741	2.879	3.521	3.058

```
first second
bar one 0.469112
two -0.282863
baz one -1.509059
two -1.135632
foo one 1.212112
two -0.173215
gux one 0.119209
two -1.044236
dtype: float64
```

TODAY/NEXT CLASS

- **Tables**
 - Abstraction
 - Operations
- **Pandas**
- **Tidy Data**
- **SQL and Relational Databases**

TIDY DATA

Variables

	age	wgt_kg	hgt_cm
	12.2	42.3	145.1
	11.0	40.8	143.8
	15.6	65.3	165.3
	35.1	84.2	185.8

But also:

- Names of files/DataFrames = description of **one** dataset
- Enforce one data type per dataset (ish)

EXAMPLE

Variable: measure or attribute:

- age, weight, height, sex

Value: measurement of attribute:

- 12.2, 42.3kg, 145.1cm, M/F

Observation: all measurements for an object

- A specific person is [12.2, 42.3, 145.1, F]

TIDYING DATA I

Name	Treatment A	Treatment B
John Smith	-	2
Jane Doe	16	11
Mary Johnson	3	1

??????????????

Name	Treatment A	Treatment B	Treatment C	Treatment D
John Smith	-	2	-	-
Jane Doe	16	11	4	1
Mary Johnson	3	1	-	2

??????????????

TIDYING DATA II

2/21

Name	Treatment	Result
John Smith	A	-
John Smith	B	2
John Smith	C	-
John Smith	D	-
Jane Doe	A	16
Jane Doe	B	11
Jane Doe	C	4
Jane Doe	D	1
Mary Johnson	A	3
Mary Johnson	B	1
Mary Johnson	C	-
Mary Johnson	D	2

MELTING DATA I

religion	<\$10k	\$10-20k	\$20-30k	\$30-40k	\$40-50k	\$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Dont know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovahs Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

??????????????

MELTING DATA II

```
f_df = pd.melt(df,  
              ["religion"],  
              var_name="income",  
              value_name="freq")  
f_df = f_df.sort_values(by=["religion"])  
f_df.head(10)
```

religion	income	freq
Agnostic	<\$10k	27
Agnostic	\$30-40k	81
Agnostic	\$40-50k	76
Agnostic	\$50-75k	137
Agnostic	\$10-20k	34
Agnostic	\$20-30k	60
Atheist	\$40-50k	35
Atheist	\$20-30k	37
Atheist	\$10-20k	27
Atheist	\$30-40k	52

MORE COMPLICATED EXAMPLE

Billboard Top 100 data for songs, covering their position on the Top 100 for 75 weeks, with two “messy” bits:



- Column headers for each of the 75 weeks
- If a song didn't last 75 weeks, those columns have are null

year	artist.in verted	track	time	genre	date.ente red	date.pea ked	x1st.wee k	x2nd.we ek	...
2000	Destiny's Child	Independent Women Part I	3:38	Rock	2000-09- 23	2000-11- 18	78	63.0	...
2000	Santana	Maria, Maria	4:18	Rock	2000-02- 12	2000-04- 08	15	8.0	...
2000	Savage Garden	I Knew I Loved You	4:07	Rock	1999-10- 23	2000-01- 29	71	48.0	...
2000	Madonn a	Music	3:45	Rock	2000-08- 12	2000-09- 16	41	23.0	...
2000	Aguilera, Christina	Come On Over Baby	3:38	Rock	2000-08- 05	2000-10- 14	57	47.0	...
2000	Janet	Doesn't Really Matter	4:17	Rock	2000-06- 17	2000-08- 26	59	52.0	...

Messy columns!

MORE COMPLICATED EXAMPLE

```
# Keep identifier variables
id_vars = ["year",
           "artist.inverted",
           "track",
           "time",
           "genre",
           "date.entered",
           "date.peaked"]

# Melt the rest into week and rank columns
df = pd.melt(frame=df,
             id_vars=id_vars,
             var_name="week",
             value_name="rank")
```

Creates one row per week, per record, with its rank

MORE COMPLICATED EXAMPLE

```
# Formatting
df["week"] = df['week'].str.extract('(\d+)',
                                     expand=False).astype(int)
df["rank"] = df["rank"].astype(int)
```

```
[..., "x2nd.week", 63.0] → [..., 2, 63]
```

```
# Cleaning out unnecessary rows
df = df.dropna()

# Create "date" columns
df['date'] = pd.to_datetime(
    df['date.entered'] +
    pd.to_timedelta(df['week'], unit='w') -
    pd.DateOffset(weeks=1)
```


MORE COMPLICATED EXAMPLE

```
# Ignore now-redundant, messy columns
df = df[["year",
        "artist.inverted",
        "track",
        "time",
        "genre",
        "week",
        "rank",
        "date"]]

df = df.sort_values(ascending=True,
                   by=["year", "artist.inverted", "track", "week", "rank"])

# Keep tidy dataset for future usage
billboard = df

df.head(10)
```

MORE COMPLICATED EXAMPLE

year	artist.in verted	track	time	genre	week	rank	date
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	1	87	2000-02-26
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	2	82	2000-03-04
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	3	72	2000-03-11
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	4	77	2000-03-18
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	5	87	2000-03-25
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	6	94	2000-04-01
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	7	99	2000-04-08
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	1	91	2000-09-02
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	2	87	2000-09-09
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	3	92	2000-09-16

??????????????

MORE TO DO?

Column headers are values, not variable names?

- Good to go!

Multiple variables are stored in one column?

- Maybe (depends on if genre text in raw data was multiple)

Variables are stored in both rows and columns?

- Good to go!

Multiple types of observational units in the same table?

- Good to go! One row per song's week on the Top 100.

A single observational unit is stored in multiple tables?

- Don't do this!

Repetition of data?

- Lots! Artist and song title's text names. Which leads us to ...

TODAY/NEXT CLASS

- **Tables**
 - Abstraction
 - Operations
- **Pandas**
- **Tidy Data**
- **SQL and Relational Databases**

TODAY'S LECTURE

Relational data:

- What is a relation, and how do they interact?

Querying databases:

- SQL
- SQLite
- How does this relate to pandas?

Joins



RELATION

Simplest relation: a table aka tabular data full of **unique** tuples

Variables
(called attributes)

Labels		ID	age	wgt_kg	hgt_cm
		1	12.2	42.3	145.1
		2	11.0	40.8	143.8
Observations (called tuples)		3	15.6	65.3	165.3
		4	35.1	84.2	185.8

PRIMARY KEYS

ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

ID	Nationality
1	USA
2	Canada
3	Mexico

The primary key is a unique identifier for every tuple in a relation

- Each tuple has exactly one primary key

FOREIGN KEYS

ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

ID	Nationality
1	USA
2	Canada
3	Mexico

Foreign keys are attributes (columns) that point to a different table's primary key


- **A table can have multiple foreign keys**

SEARCHING FOR ELEMENTS

Find all people with nationality Canada (nat_id = 2):

????????????????

ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

$O(n)$ 

INDEXES

Like a hidden sorted map of references to a specific attribute (column) in a table; allows $O(\log n)$ lookup instead of $O(n)$

loc	ID	age	wgt_kg	hgt_cm	nat_id
0	1	12.2	42.3	145.1	1
128	2	11.0	40.8	143.8	2
256	3	15.6	65.3	165.3	2
384	4	35.1	84.2	185.8	1
512	5	18.1	62.2	176.2	3
640	6	19.6	82.1	180.1	1

nat_id	locs
1	0, 384, 640
2	128, 256
3	512

INDEXES

Actually implemented with data structures like B-trees

- (Take courses like CMSC424 or CMSC420)

But: indexes are not free

- Takes memory to store
- Takes time to build
- Takes time to update (add/delete a row, update the column)

But, but: one index is (mostly) free

- Index will be built automatically on the **primary key**

Think before you build/maintain an index on other attributes!



RELATIONSHIPS

Primary keys and foreign keys define interactions between different tables aka entities. Four types:

- One-to-one
- One-to-one-or-none
- One-to-many and many-to-one
- Many-to-many



Connects (one, many) of the rows in one table to (one, many) of the rows in another table

ONE-TO-MANY & MANY-TO-ONE

One person can have **one** nationality in this example, but one nationality can include **many** people.

Person

Nationality

ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

ID	Nationality
1	USA
2	Canada
3	Mexico



ONE-TO-ONE

Two tables have a one-to-one relationship if every tuple in the first table corresponds to **exactly one** entry in the other



In general, you won't be using these (why not just merge the rows into one table?) unless:

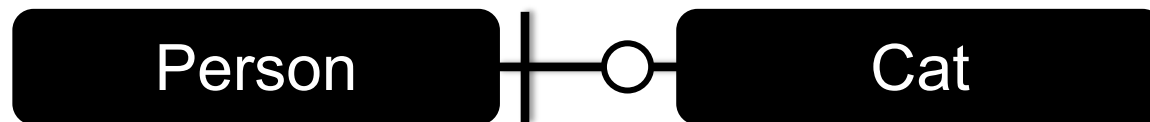
- Split a big row between SSD and HDD or distributed
- Restrict access to part of a row (some DBMSs allow column-level access control, but not all)
- Caching, partitioning, & serious stuff: take CMSC424

ONE-TO-ONE-OR-NONE

Say we want to keep track of people's cats:

Person ID	Cat1	Cat2
1	Chairman Meow	Fuzz Aldrin
4	Anderson Pooper	Meowly Cyrus
5	Gigabyte	Megabyte

People with IDs 2 and 3 do not own cats*, and are not in the table. **Each person has at most one entry in the table.**



Is this data **tidy**?

*nor do they have hearts, apparently.

MANY-TO-MANY

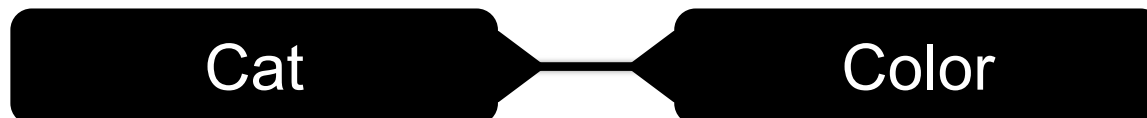
Say we want to keep track of people's cats' colorings:

ID	Name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin
4	Chairman Meow
5	Anderson Pooper
6	Gigabyte

Cat ID	Color ID	Amount
1	1	50
1	2	50
2	2	20
2	4	40
2	5	40
3	1	100

One column per color, too many columns, too many nulls

Each cat can have many colors, and each color many cats



ASSOCIATIVE TABLES

Cats

ID	Name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin
4	Chairman Meow
5	Anderson Pooper
6	Gigabyte

Cat ID	Color ID	Amount
1	1	50
1	2	50
2	2	20
2	4	40
2	5	40
3	1	100

Colors

ID	Name
1	Black
2	Brown
3	White
4	Orange
5	Neon Green
6	Invisible

Primary key ??????????????

- [Cat ID, Color ID] (+ [Color ID, Cat ID], case-dependent)

Foreign key(s) ??????????????

- Cat ID and Color ID

ASIDE: PANDAS

So, this kinda feels like pandas ...

- And pandas kinda feels like a relational data system ...

Pandas is **not strictly a relational data system:**

- No notion of primary / foreign keys

It does have indexes (and multi-column indexes):

- pandas.Index: ordered, sliceable set storing axis labels
- pandas.MultiIndex: hierarchical index

Rule of thumb: do heavy, rough lifting at the relational DB level, then fine-grained slicing and dicing and viz with pandas

SQLITE

On-disk relational database management system (RDMS)

- Applications connect directly to a **file**

Most RDMSs have applications connect to a server:

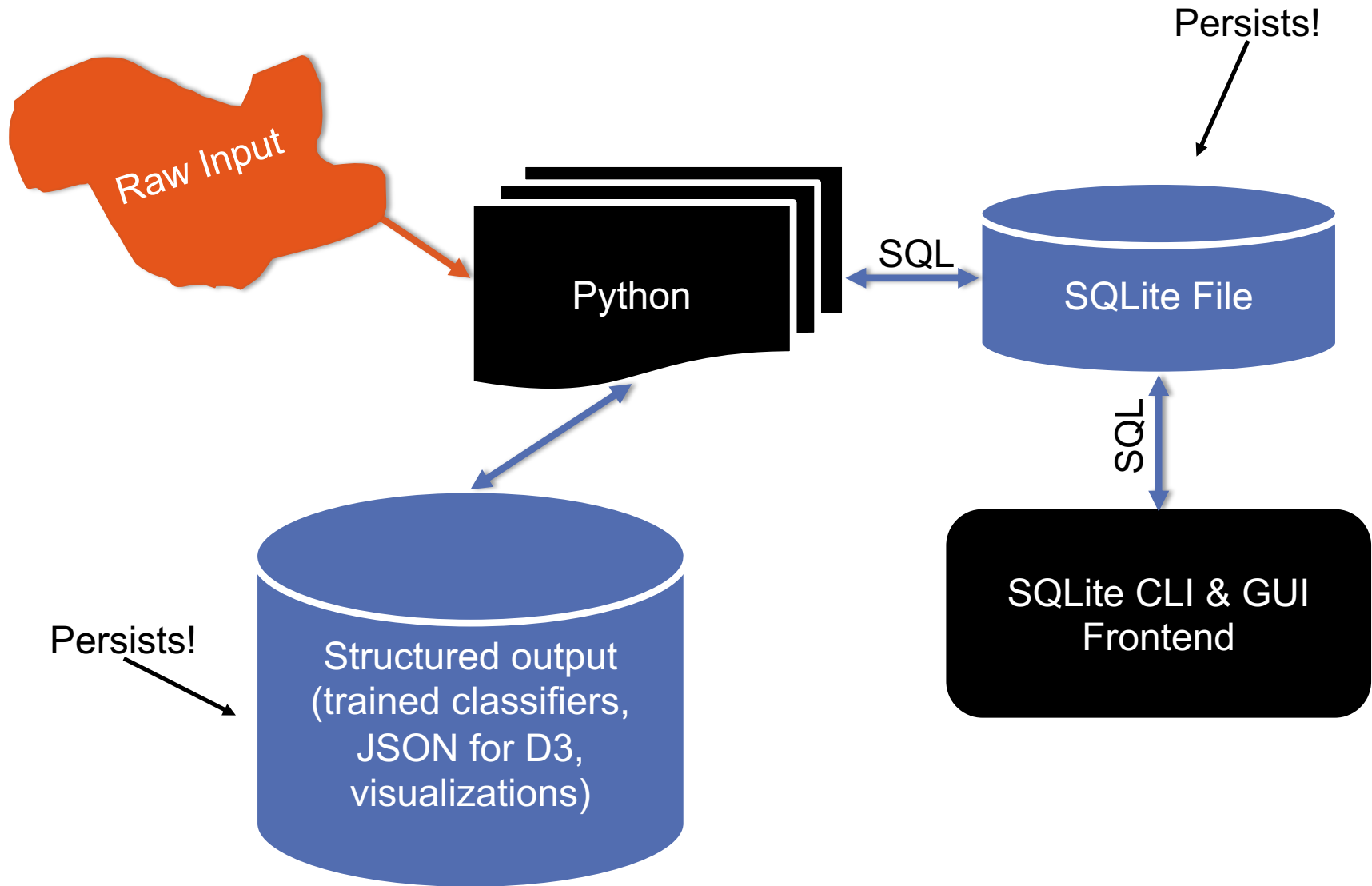
- Advantages include greater concurrency, less restrictive locking
- Disadvantages include, for this class, setup time 😊

Installation:

- `conda install -c anaconda sqlite`
- (Should come preinstalled, I think?)

All interactions use Structured Query Language (SQL)

HOW A RELATIONAL DB FITS INTO YOUR WORKFLOW



CRASH COURSE IN SQL (IN PYTHON)

```
import sqlite3

# Create a database and connect to it
conn = sqlite3.connect("cmisc320.db")
cursor = conn.cursor()

# do cool stuff
conn.close()
```

Cursor: temporary work area in system memory for manipulating SQL statements and return values

If you do not close the connection (`conn.close()`), any outstanding transaction is rolled back

- (More on this in a bit.)

CRASH COURSE IN SQL (IN PYTHON)

```
# Make a table
cursor.execute("""
CREATE TABLE cats (
    id INTEGER PRIMARY KEY,
    name TEXT
)""")
```

??????????

id	name
	cats

Capitalization doesn't matter for SQL reserved words

- SELECT = select = SeLeCt

Rule of thumb: capitalize keywords for readability

CRASH COURSE IN SQL (IN PYTHON)

Insert into the table

```
cursor.execute("INSERT INTO cats VALUE (1, 'Megabyte')")
cursor.execute("INSERT INTO cats VALUE (2, 'Meowly Cyrus')")
cursor.execute("INSERT INTO cats VALUE (3, 'Fuzz Aldrin')")
conn.commit()
```

id	name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin

Delete row(s) from the table

```
cursor.execute("DELETE FROM cats WHERE id == 2");
conn.commit()
```

id	name
1	Megabyte
3	Fuzz Aldrin



CRASH COURSE IN SQL (IN PYTHON)

```
# Read all rows from a table
for row in cursor.execute("SELECT * FROM cats"):
    print(row)
```

```
# Read all rows into pandas DataFrame
pd.read_sql_query("SELECT * FROM cats", conn, index_col="id")
```

id	name
1	Megabyte
3	Fuzz Aldrin

index_col="id": treat column with label "id" as an index

index_col=1: treat column #1 (i.e., "name") as an index

(Can also do multi-indexing.)

JOINING DATA

A **join** operation merges two or more tables into a single relation. Different ways of doing this:

- Inner
- Left
- Right
- Full Outer

Join operations are done **on** columns that explicitly link the tables together

INNER JOINS

id	name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin
4	Chairman Meow
5	Anderson Pooper
6	Gigabyte

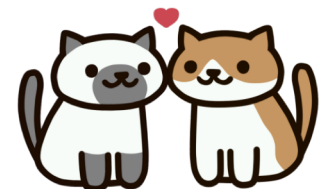
cats

cat_id	last_visit
1	02-16-2017
2	02-14-2017
5	02-03-2017

visits

Inner join returns merged rows that share the **same** value in the column they are being joined on (`id` and `cat_id`).

id	name	last_visit
1	Megabyte	02-16-2017
2	Meowly Cyrus	02-14-2017
5	Anderson Pooper	02-03-2017



INNER JOINS

```
# Inner join in pandas
df_cats = pd.read_sql_query("SELECT * from cats", conn)
df_visits = pd.read_sql_query("SELECT * from visits", conn)
df_cats.merge(df_visits, how = "inner",
              left_on = "id", right_on = "cat_id")
```

```
# Inner join in SQL / SQLite via Python
cursor.execute("""
    SELECT
        *
    FROM
        cats, visits
    WHERE
        cats.id == visits.cat_id
""")
```

LEFT JOINS

Inner joins are the most common type of joins (get results that appear in **both** tables)

Left joins: all the results from the left table, only **some** matching results from the right table

Left join (cats, visits) on (id, cat_id) ??????????????

id	name	last_visit
1	Megabyte	02-16-2017
2	Meowly Cyrus	02-14-2017
3	Fuzz Aldrin	NULL
4	Chairman Meow	NULL
5	Anderson Pooper	02-03-2017
6	Gigabyte	NULL

RIGHT JOINS

Take a guess!

Right join
(cats, visits)
on
(id, cat_id)
??????????????

id	name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin
4	Chairman Meow
5	Anderson Pooper
6	Gigabyte

cats

cat_id	last_visit
1	02-16-2017
2	02-14-2017
5	02-03-2017
7	02-19-2017
12	02-21-2017

visits

id	name	last_visit
1	Megabyte	02-16-2017
2	Meowly Cyrus	02-14-2017
5	Anderson Pooper	02-03-2017
7	NULL	02-19-2017
12	NULL	02-21-2017

LEFT/RIGHT JOINS

```
# Left join in pandas
df_cats.merge(df_visits, how = "left",
              left_on = "id", right_on = "cat_id")
```

```
# Left join in SQL / SQLite via Python
cursor.execute("SELECT * FROM cats LEFT JOIN visits ON
               cats.id == visits.cat_id")
```

```
# Right join in pandas
df_cats.merge(df_visits, how = "right",
              left_on = "id", right_on = "cat_id")
```

```
# Right join in SQL / SQLite via Python
```



FULL OUTER JOIN

Combines the left and the right join

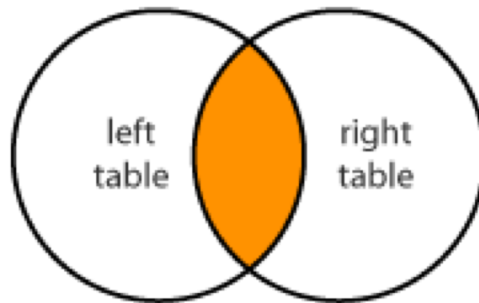
??????????????

id	name	last_visit
1	Megabyte	02-16-2017
2	Meowly Cyrus	02-14-2017
3	Fuzz Aldrin	NULL
4	Chairman Meow	NULL
5	Anderson Pooper	02-03-2017
6	Gigabyte	NULL
7	NULL	02-19-2017
12	NULL	02-21-2017

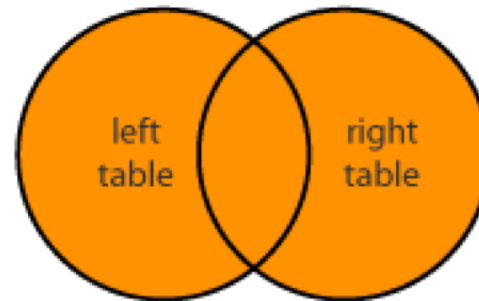
```
# Outer join in pandas
df_cats.merge(df_visits, how = "outer",
              left_on = "id", right_on = "cat_id")
```

GOOGLE IMAGE SEARCH ONE SLIDE SQL JOIN VISUAL

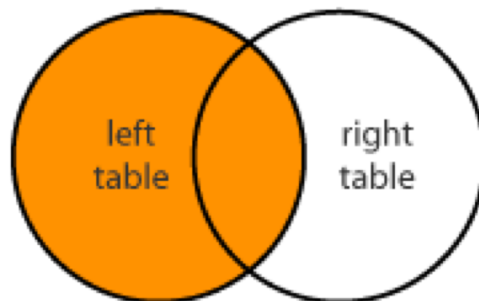
INNER JOIN



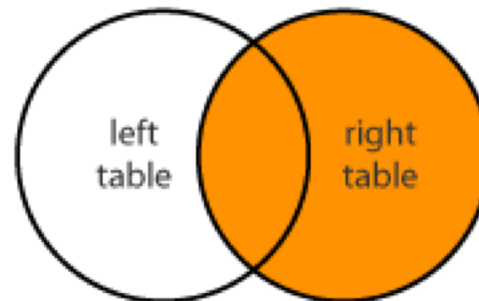
FULL JOIN



LEFT JOIN



RIGHT JOIN



RAW SQL IN PANDAS



If you “think in SQL” already, you’ll be fine with pandas:

- `conda install -c anaconda pandasql`
- Info: http://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html

```
# Write the query text
q = """
    SELECT
        *
    FROM
        cats
    LIMIT 10;"""

# Store in a DataFrame
df = sqldf(q, locals())
```

NEXT CLASS:
EXPLORATORY ANALYSIS

