Operating Systems:
Memory and File System

DS 5110: Big Data Systems (Spring 2023)

Lecture 2c

Yue Cheng

Some material taken/derived from:
- Wisconsin CS301 by Tyler Harter.

The memory hierarchy

Recap: How fast can a CPU run a program?

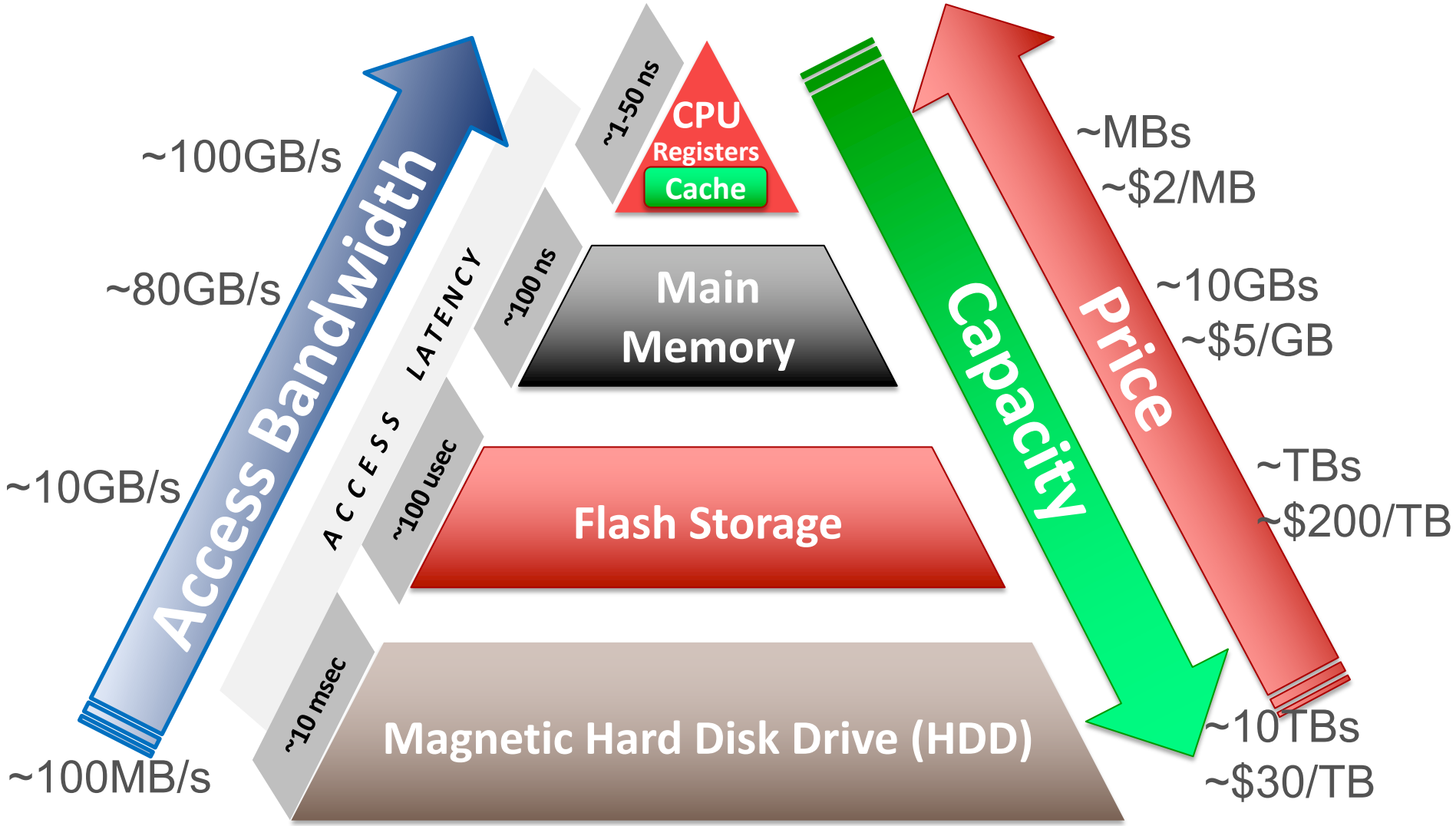
- Most programs do not keep CPU always busy
  - Memory access instructions stall the CPU: i.e., ALU & CU idle during DRAM-register transfer
  - Worse, data may not be in DRAM – wait for disk I/O!
  - Actual runtime of a program may be 10-100x higher than what clock rate calculation model suggests

Recap: How fast can a CPU run a program?

- Most programs do not keep CPU always busy
  - Memory access instructions stall the CPU: i.e., ALU & CU idle during DRAM-register transfer
  - Worse, data may not be in DRAM – wait for disk I/O!
  - Actual runtime of a program may be 10-100x higher than what clock rate calculation model suggests

Key principle: Optimizing use of CPU caches (and faster storage) is critical for processor performance!

Memory-storage hierarchy



*UCSD DSC 102: Systems for scalable analysis. Arun Kumar

Workload characteristics

Application A

```
sum = 0
for i in range(0,1024):
  sum += a[i]
```

Workload characteristics

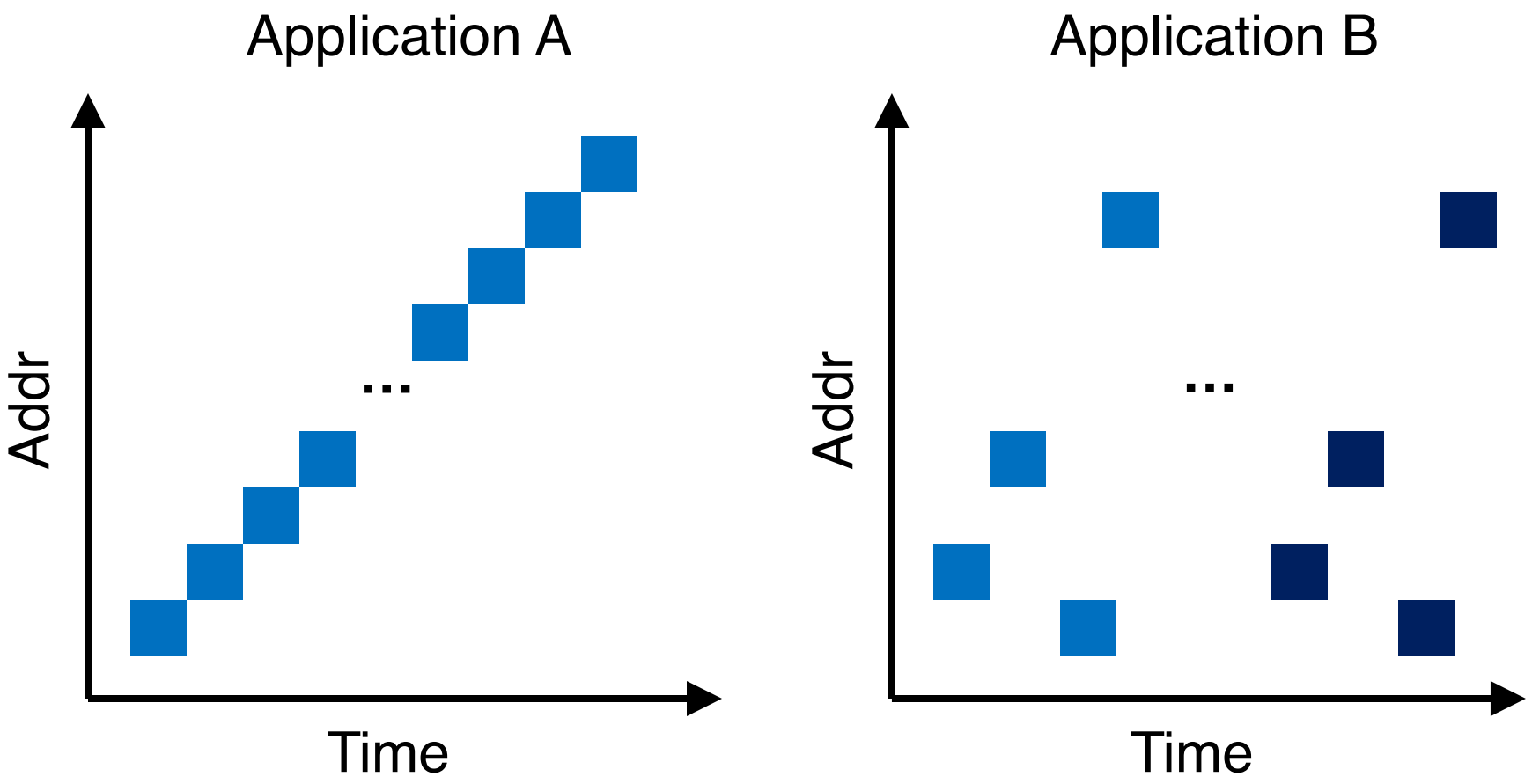
Application A

```
sum = 0
for i in range(0,1024):
  sum += a[i]
```

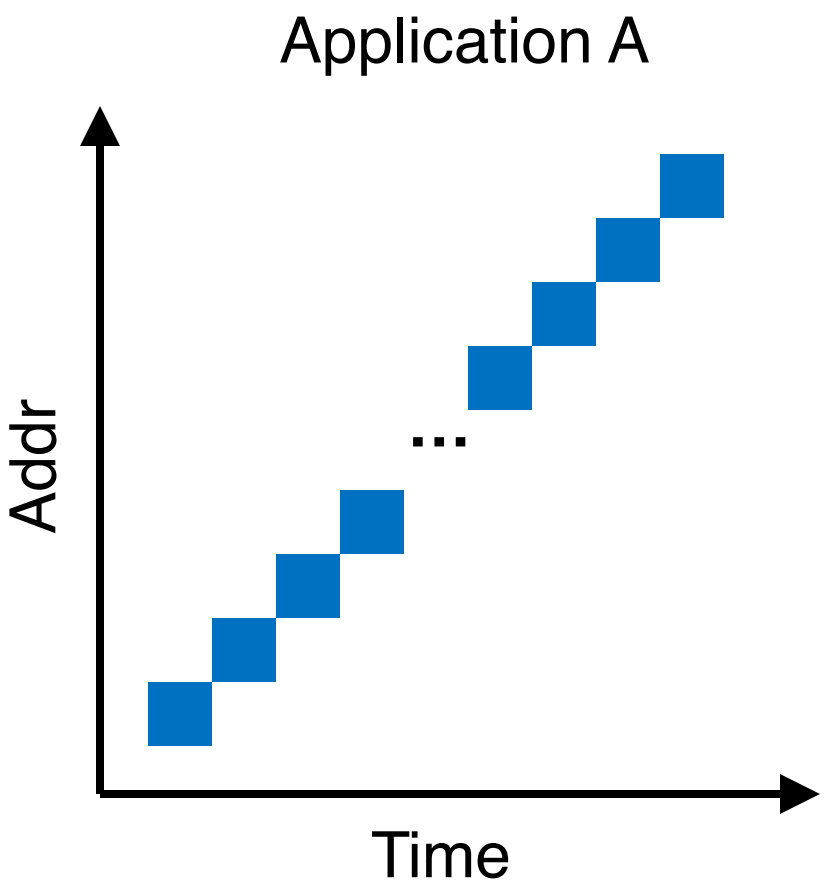
Application B

```
import random

sum = 0
random.seed(1234);
for i in range(0,512):
  sum += a[random.randint(0,1023)]

random.seed(1234) # same seed
for i in range(0,512):
  sum += a[random.randint(0,1023)]
```

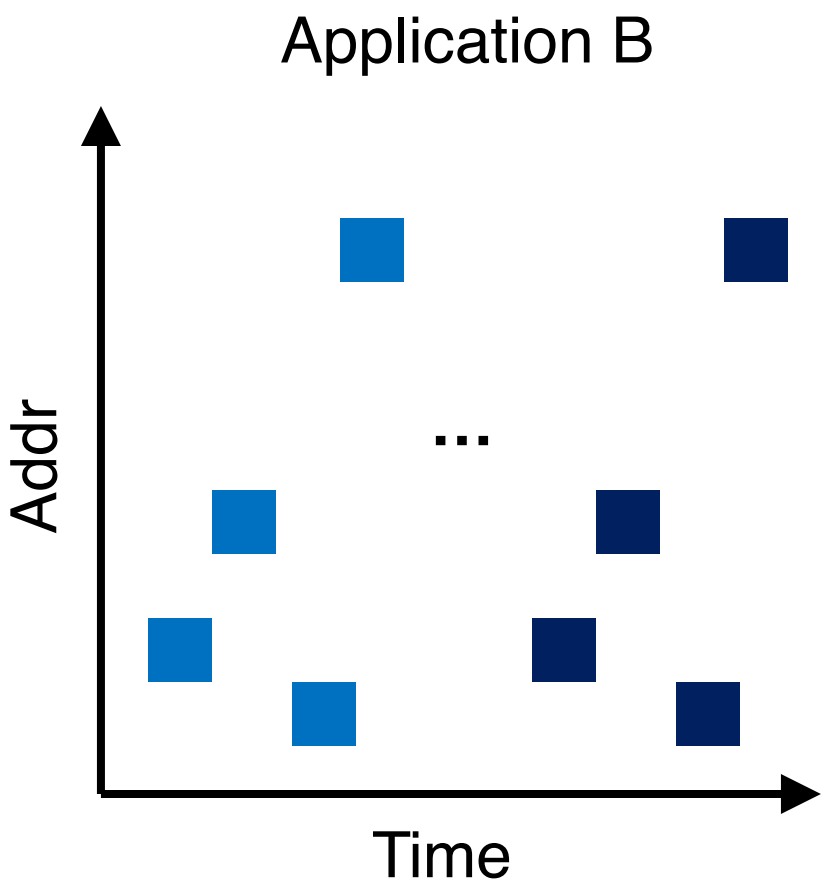
Access patterns



Access patterns



Spatial Locality



Temporal Locality

Locality of data accesses

- Spatial locality:
  - Future access will be to nearby addresses
- Temporal locality:
  - Future access will be repeated to the same data

Locality of data accesses

- Spatial locality:
  - Future access will be to nearby addresses
- Temporal locality:
  - Future access will be repeated to the same data
- Q: What is the **implication of data locality** to Data Science applications?

Locality optimization in Data Science

- Consider a tensor (matrix) named data with `128*128` elements
- Each row is of size 128 words and prefetching+caching means full row of accessed data item is brought to CPU cache

Locality optimization in Data Science

- Consider a tensor (matrix) named data with `128*128` elements
- Each row is of size 128 words and prefetching+caching means full row of accessed data item is brought to CPU cache
- Program 1

```
for j in range(0,128):
  for i in range(0,128):
    data[i][j] = 0
```

128 x 128 = 16,384 CPU cache misses

Not too hardware-efficient (not able to exploit prefetching+caching)

Locality optimization in Data Science

- Consider a tensor (matrix) named data with `128*128` elements
- Each row is of size 128 words and prefetching+caching means full row of accessed data item is brought to CPU cache
- Program 1

```
for j in range(0,128):
  for i in range(0,128):
    data[i][j] = 0
```

128 x 128 = 16,384 CPU cache misses

Not too hardware-efficient (not able to exploit prefetching+caching)

- Program 2

```
for i in range(0,128):
  for j in range(0,128):
    data[i][j] = 0
```

Only 128 CPU cache misses

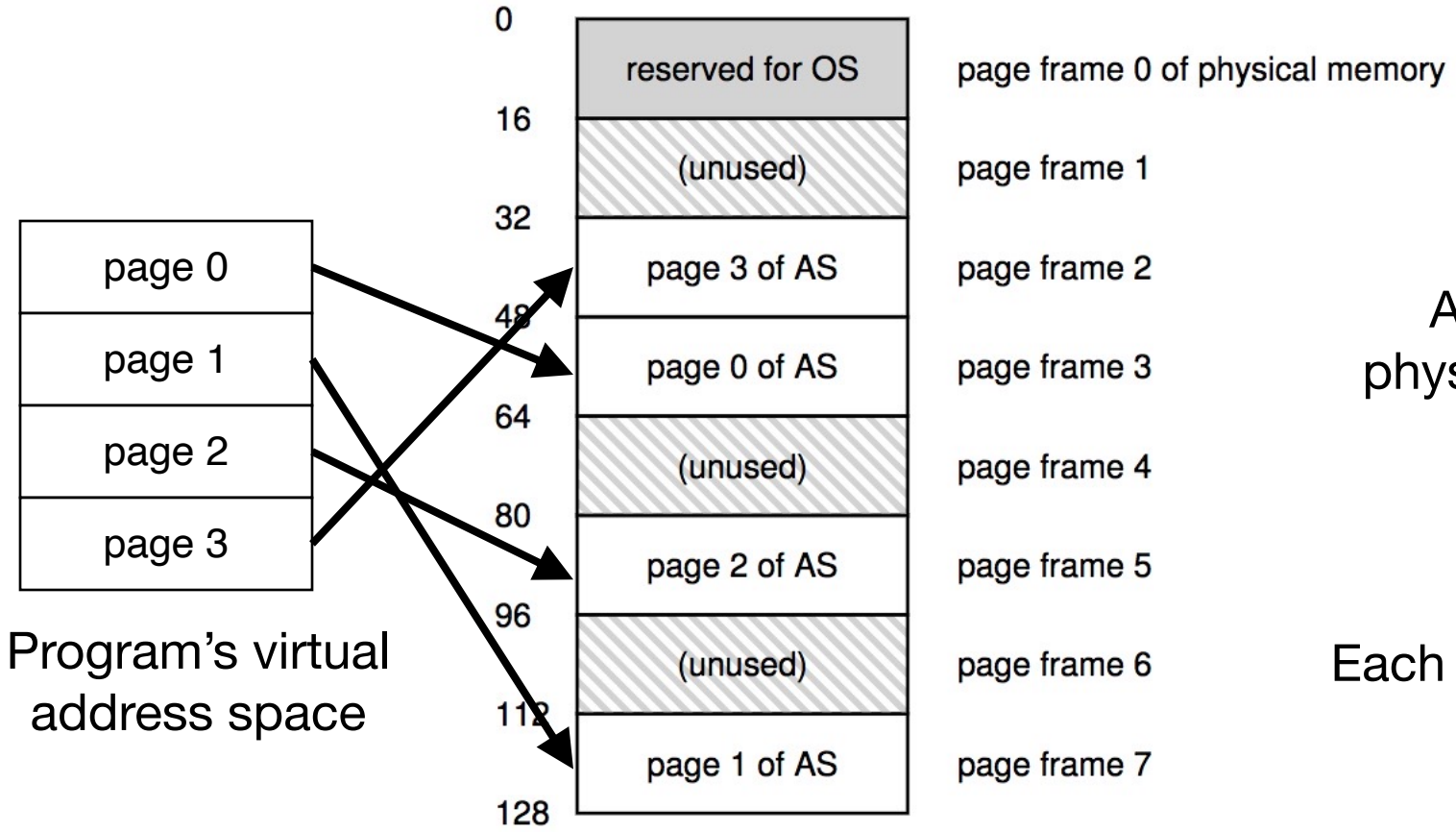
Row `data[i]` is prefetched to cache so subsequent accesses are hits!

Virtualizing (sharing) memory

OS memory management: Paging

- Paging is a memory management scheme that allows the physical address space of a process to be non-contiguous
- Divide physical memory into fixed-sized blocks called frames
- Divide a program's virtual memory into blocks of same size called pages
- Flexible mapping: Any page can go to any free frame
- Scalability: To run a program of size *n* pages, need to find *n* free frames and load program
  - Grow memory segments wherever we please!

A toy example



An 8-frame physical memory (128B)

Each page is of 16B

File system abstraction

What is a **File**?

- File: Array of bytes on a disk
  - Ranges of bytes can be read/written

- File system (FS) is an on-disk data structure that consists of many files

- Files need names so programs can choose the right one

File names

- Three types of names (abstractions)
  - inode (low-level names)
  - path (human readable)
  - file descriptor (runtime state)

Inodes

- Each file has exactly one inode number
- Inodes are unique (at a given time) within a FS
- Numbers may be recycled after deletes

Inodes

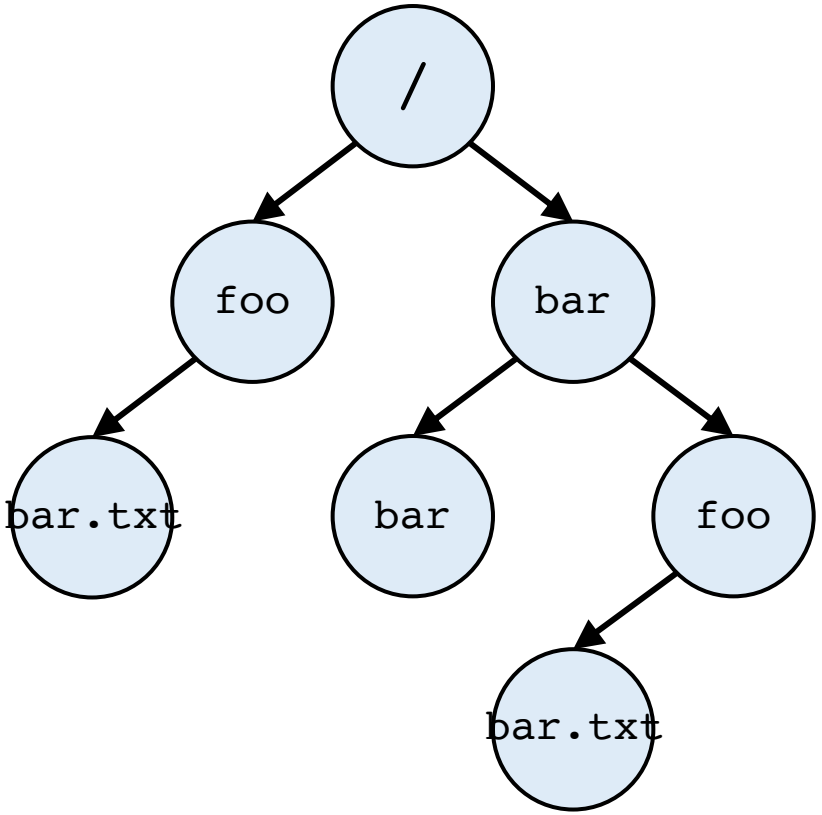
- Each file has exactly one inode number
- Inodes are unique (at a given time) within a FS
- Numbers may be recycled after deletes
- Show inodes via `stat`
  - `$ stat <file or dir>`

`stat` example

```
PROMPT>: stat test.dat
File: ‘test.dat’  Size: 5     Blocks: 8     IO Block: 4096   regular
file
Device: 803h/2051d       Inode: 119341128   Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1001/     yue)   Gid: ( 1001/     yue)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2015-12-17 04:12:47.935716294 -0500
Modify: 2014-12-12 19:25:32.669625220 -0500
Change: 2014-12-12 19:25:32.669625220 -0500
Birth: -
```

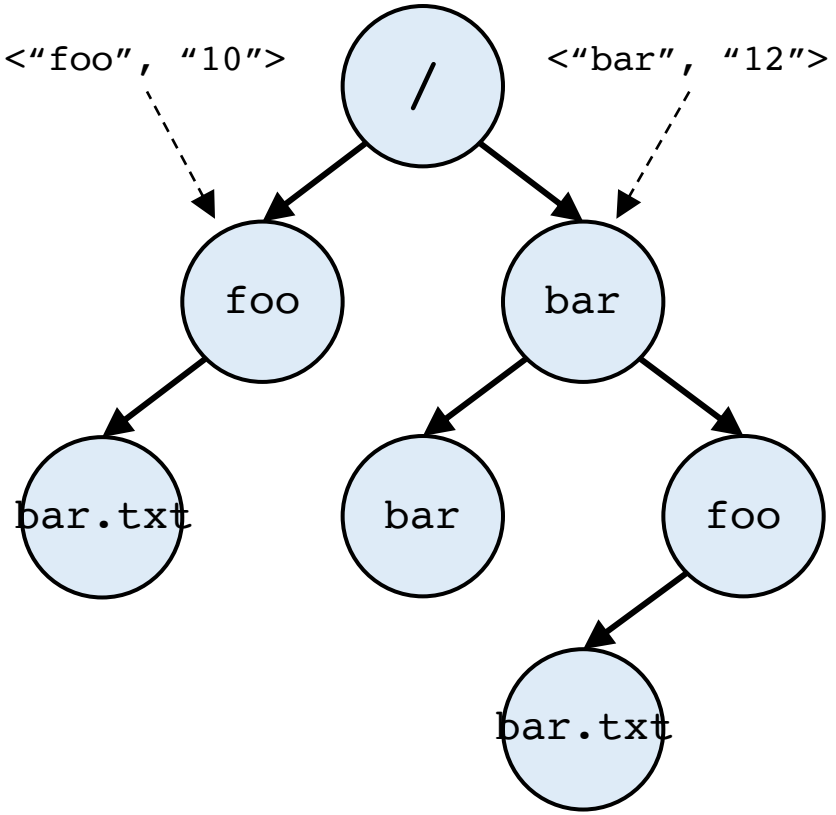
Path (multiple directories)

- A directory is a file
  - Associated with an inode
- Contains a list of <user-readable name, low-level name> pairs



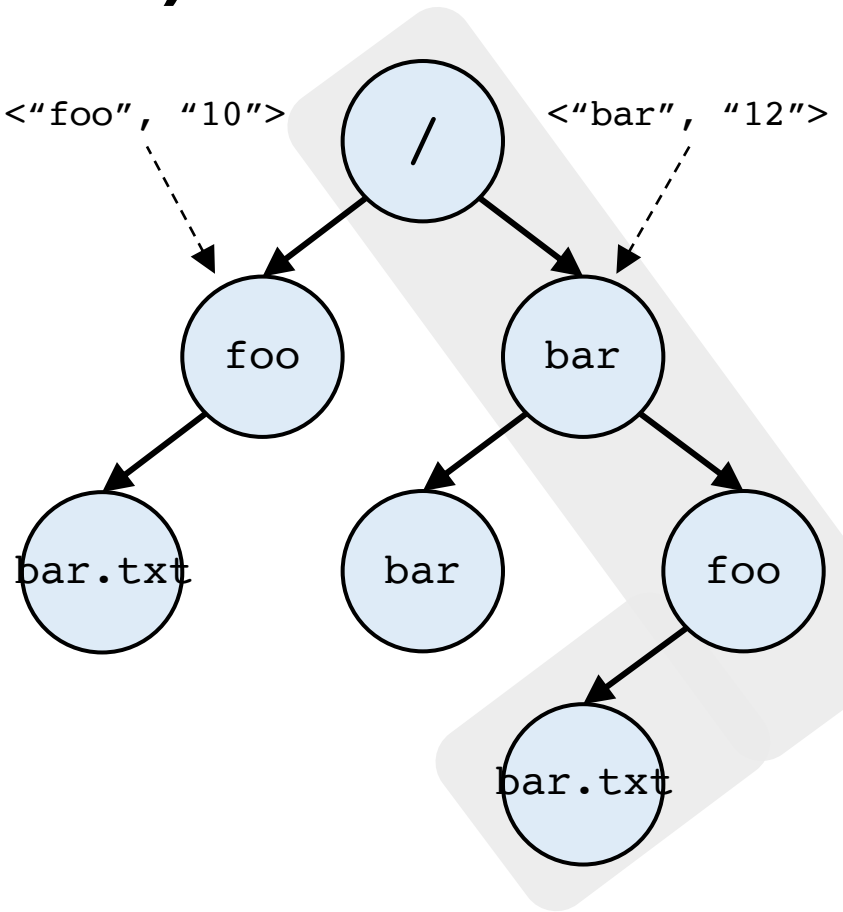
Path (multiple directories)

- A directory is a file
  - Associated with an inode
- Contains a list of <user-readable name, low-level name> pairs



Path (multiple directories)

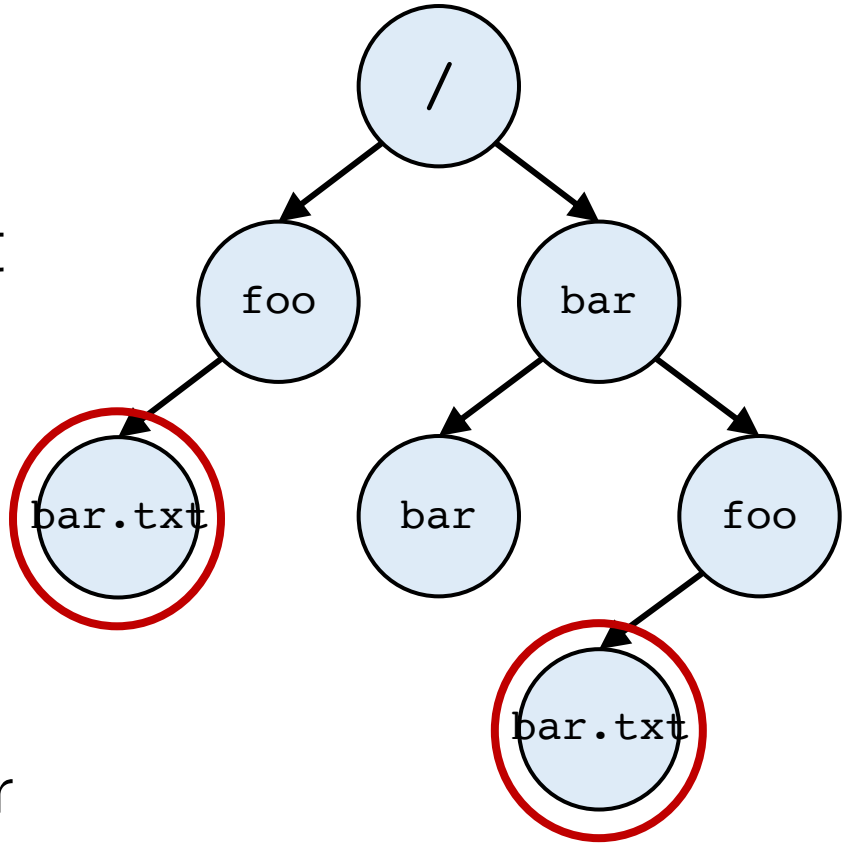
- A directory is a file
  - Associated with an inode
- Contains a list of <user-readable name, low-level name> pairs
- Directory tree: reads for getting final inode called **traversal**



[traverse `/bar/foo/bar.txt`]

File naming

- Directories and files can have the same name as long as they are in different locations of the file-system tree



- .txt, .c, etc.
  - Naming convention
  - In Linux, no enforcement for extension name

Special directory entries

```
prompt> ls -al
total 216
drwxr-xr-x  19 yue  staff   646 Nov 23 16:28 .
drwxr-xr-x+ 40 yue  staff  1360 Nov 15 01:41 ..
-rw-r--r--@  1 yue  staff  1064 Aug 29 21:48 common.h
-rwxr-xr-x   1 yue  staff  9356 Aug 30 14:03 cpu
-rw-r--r--@  1 yue  staff   258 Aug 29 21:48 cpu.c
-rwxr-xr-x   1 yue  staff  9348 Sep  6 12:12 cpu_bound
-rw-r--r--   1 yue  staff   245 Sep  5 13:10 cpu_bound.c
...
```

Basic file interactions

Basic file interactions

- Basic file system operations
  - opening/closing
  - reading/writing

- OS-related module
  - listdir, mkdir, exists, join

Basic file interactions

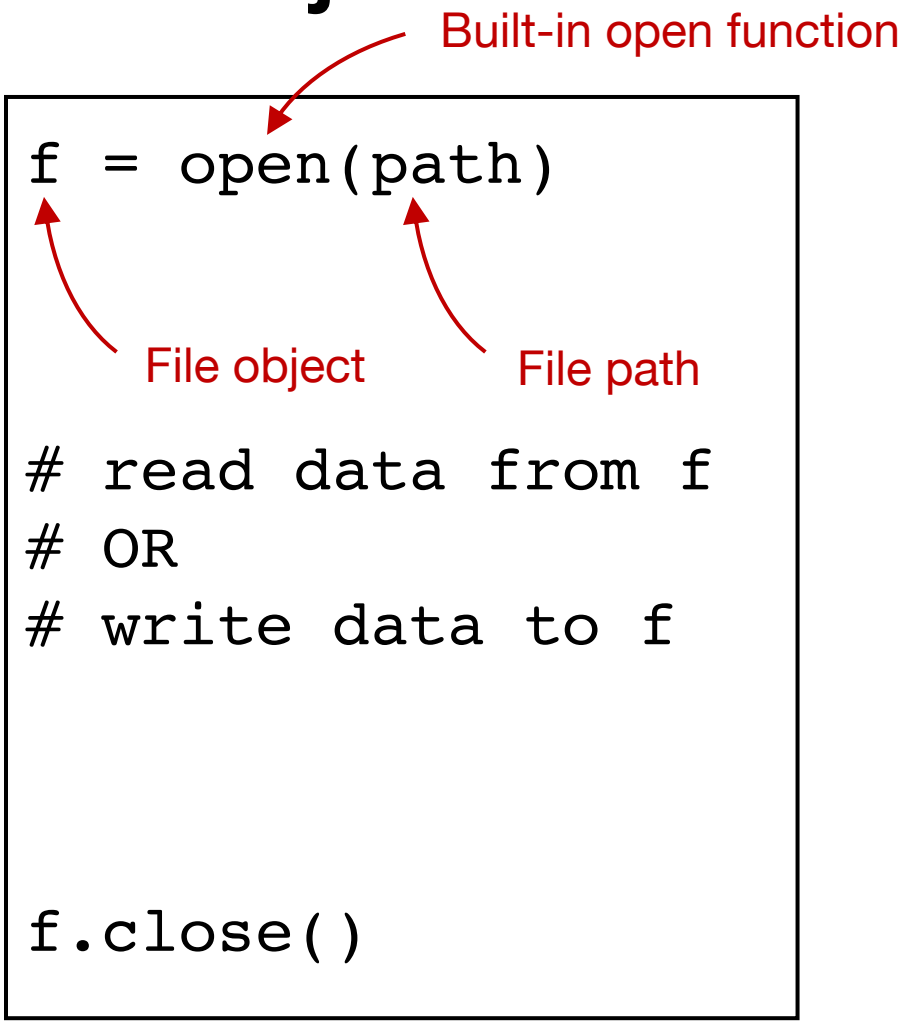
- Basic file system operations
  - opening/closing
  - reading/writing

- OS-related module
  - listdir, mkdir, exists, join

File objects

```
f = open(path)



# read data from f
# OR
# write data to f



f.close()
```

File objects



File objects

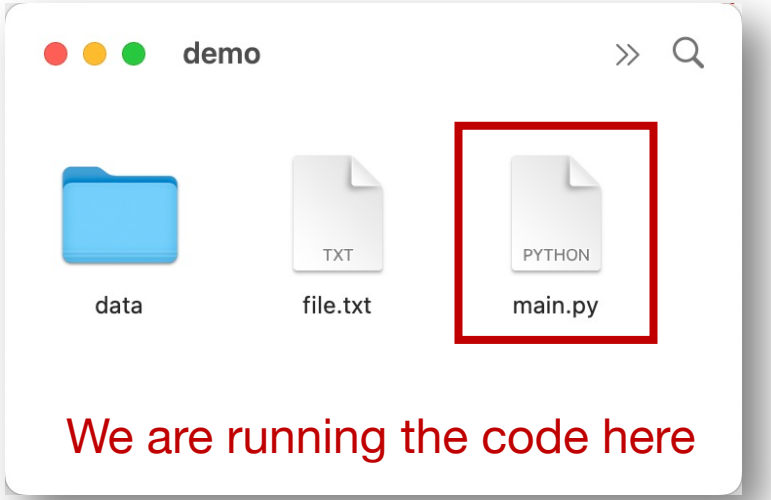
main.py

Built-in open function

```
f = open(path)

# read data from f
# OR
# write data to f



f.close()
```

File object

File path



File objects

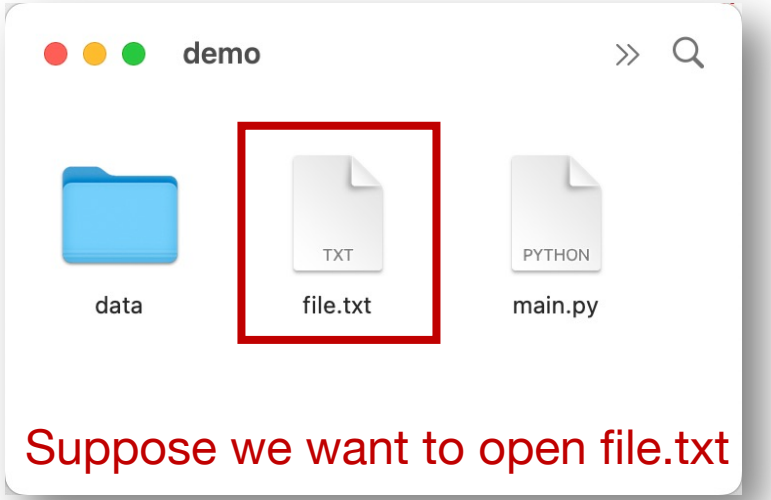
main.py

Built-in open function

```
f = open(path)

# read data from f
# OR
# write data to f



f.close()
```

File object

File path



File objects

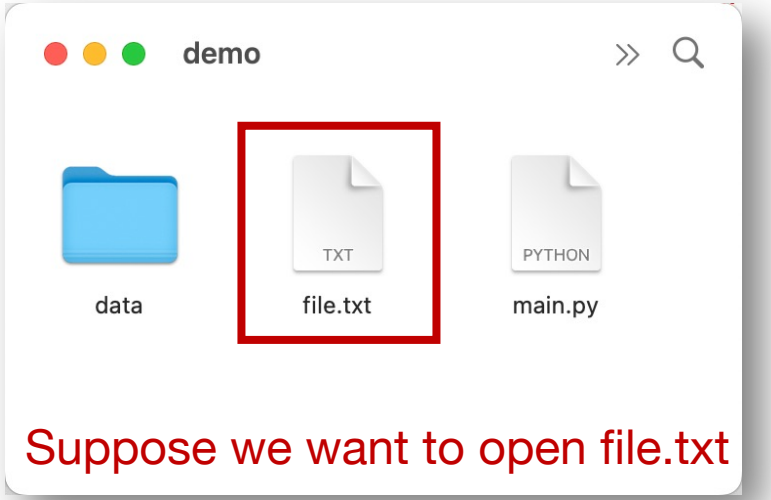
main.py

Built-in open function

```
f = open("file.txt")

# read data from f
# OR
# write data to f



f.close()
```

File object

File path



File objects

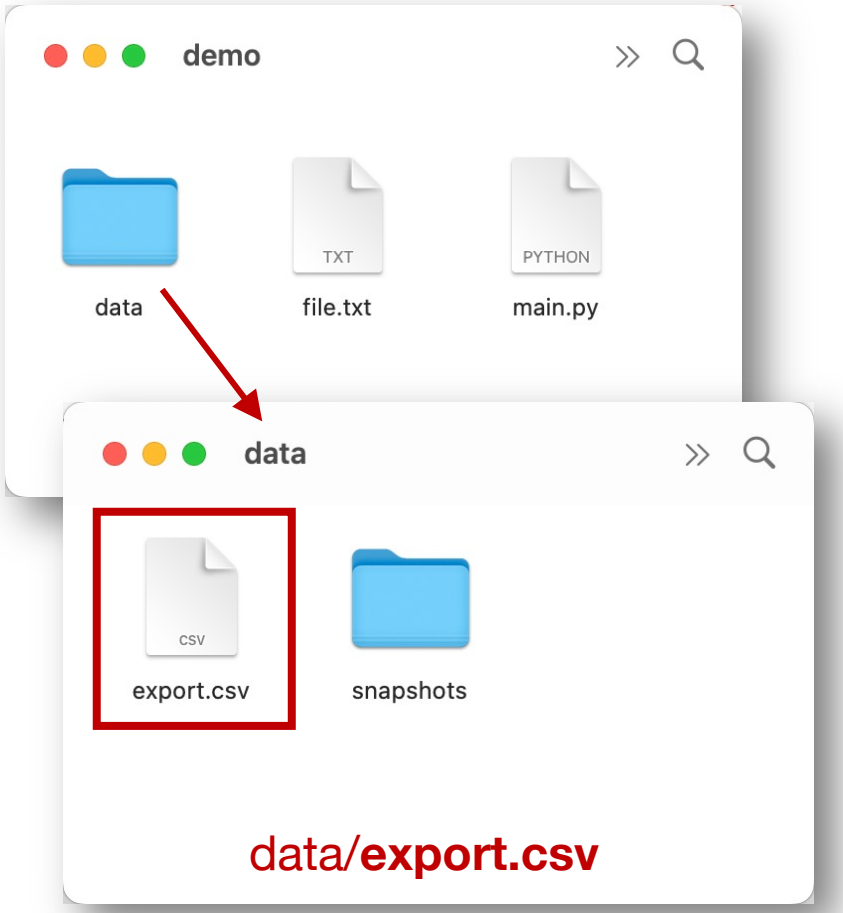
main.py

Built-in open function

```
f = open(
    "data/export.csv")

# read data from f
# OR
# write data to f



f.close()
```

File object

File path



File objects

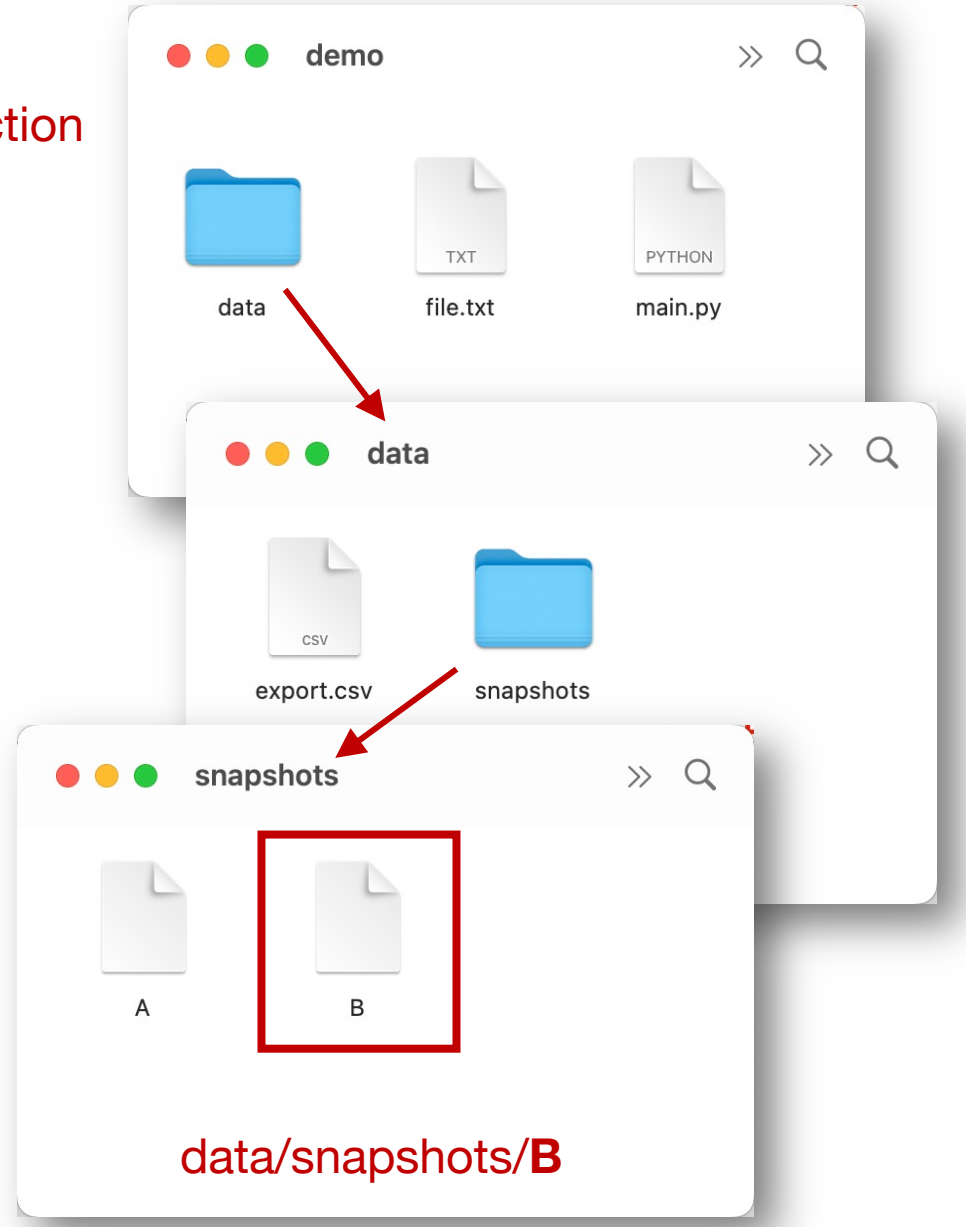
main.py

Built-in open function

```
f = open(
"data/snapshots/B")

# read data from f
# OR
# write data to f



f.close()
```

File object

File path



File objects

main.py

```
f = open("file.txt")



# read data from f
# OR
# write data to f



f.close()
```

File objects

main.py

```
f = open("file.txt")



# read data from f
# OR
# write data to f



f.close()
```

Using file

File objects

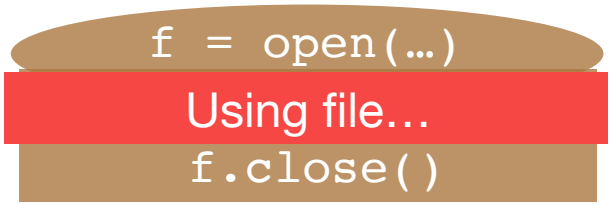
main.py

```
f = open("file.txt")



# read data from f
# OR
# write data to f



f.close()
```

Using file

Cleanup

Imagine a file object as a sandwich …



File objects

main.py

```
f = open("file.txt")



# read data from f
# OR
# write data to f



f.close()
```

Using file

Cleanup

Reasons for **closing**:

- Avoid data loss
- Limited number of open files

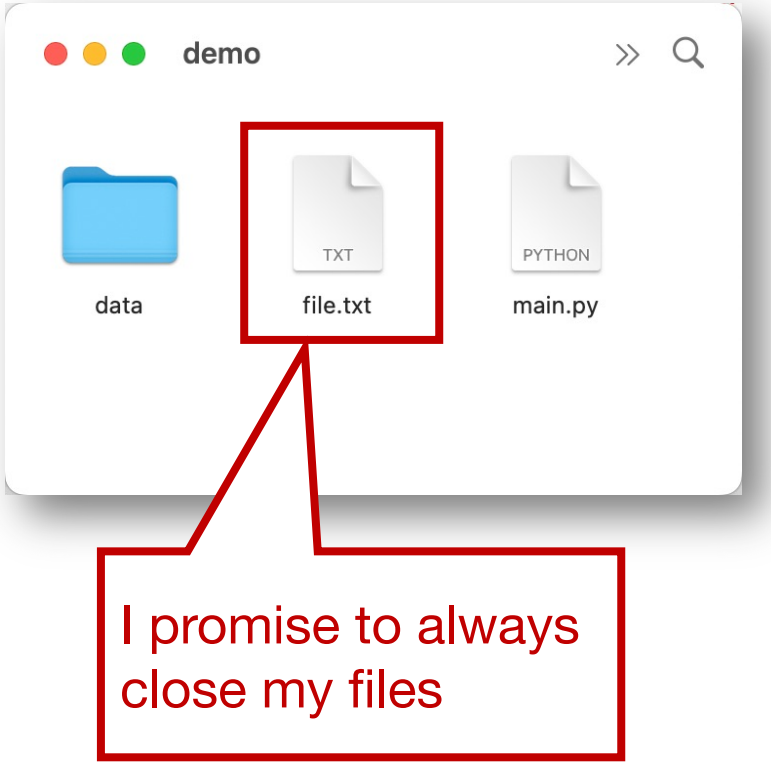
Basic file interactions

- Basic file system operations
  - opening/closing
  - reading/writing

- OS-related module
  - listdir, mkdir, exists, join

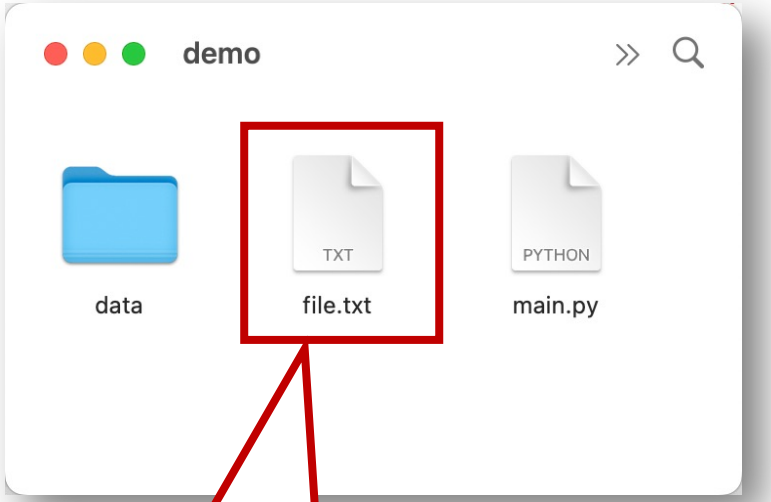
Reading a file

```
f = open("file.txt")



# read data from f
# OR
# write data to f



f.close()
```



Reading a file

```
f = open("file.txt")



data = f.read()

print(data)

f.close()
```



I promise to always close my files

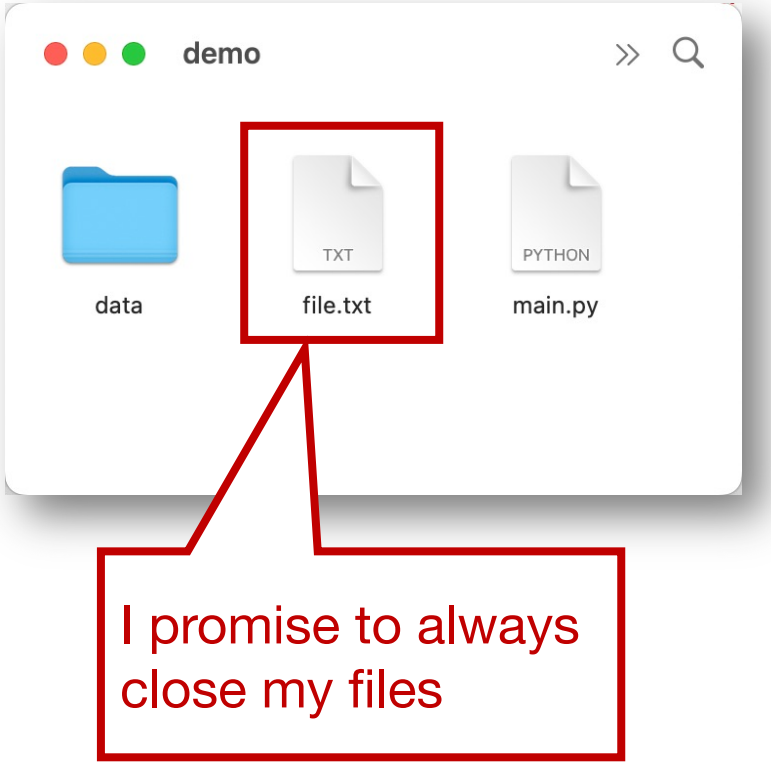
data is: "I promise to always\nclose my files"

`read()` method

- Fetch entire content
- Return as a string

Writing a file

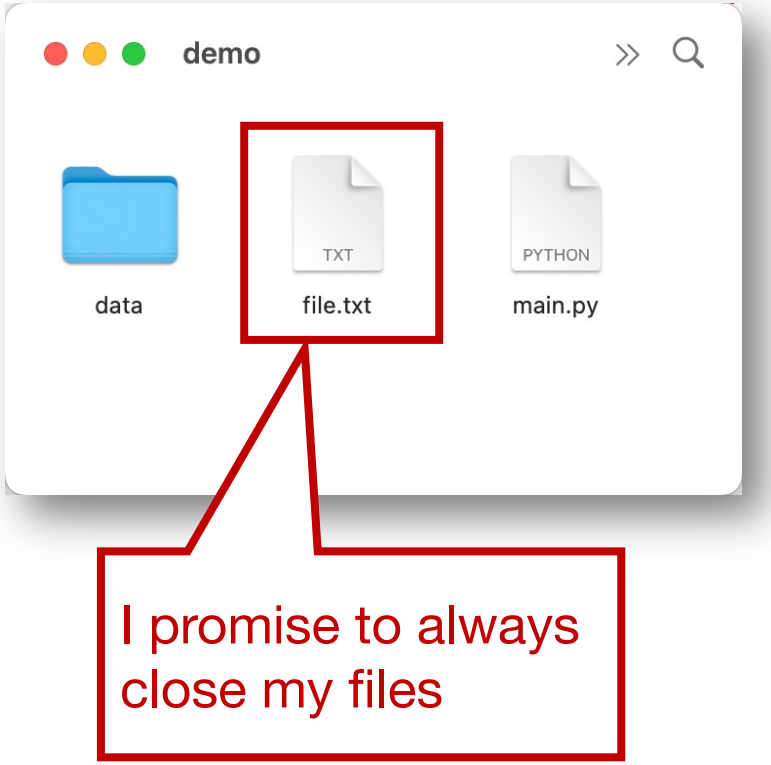
```
f = open("file.txt")



# read data from f
# OR
# write data to f



f.close()
```



Writing a file

```
f = open("file.txt",
    "w")

# read data from f
# OR
# write data to f



f.close()
```

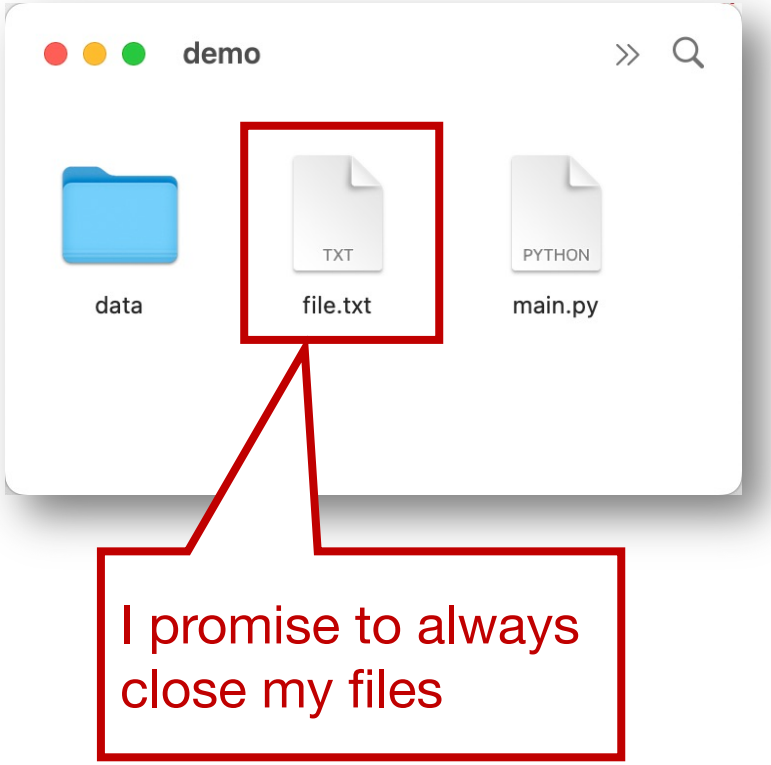
"w" mode indicates to write to this file



Writing a file

```
f = open("file.txt",
    "w")
f.write("hello")
f.write(" world\n")
f.write("!!!\n")



f.close()
```

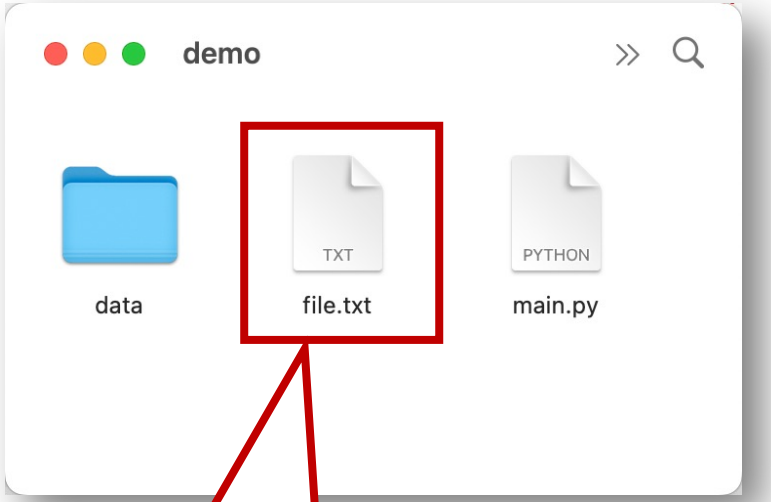
"w" mode indicates to write to this file



Writing a file

```
f = open("file.txt",
    "w")
f.write("hello")
f.write(" world\n")
f.write("!!!\n")



f.close()
```

"w" mode indicates to write to this file



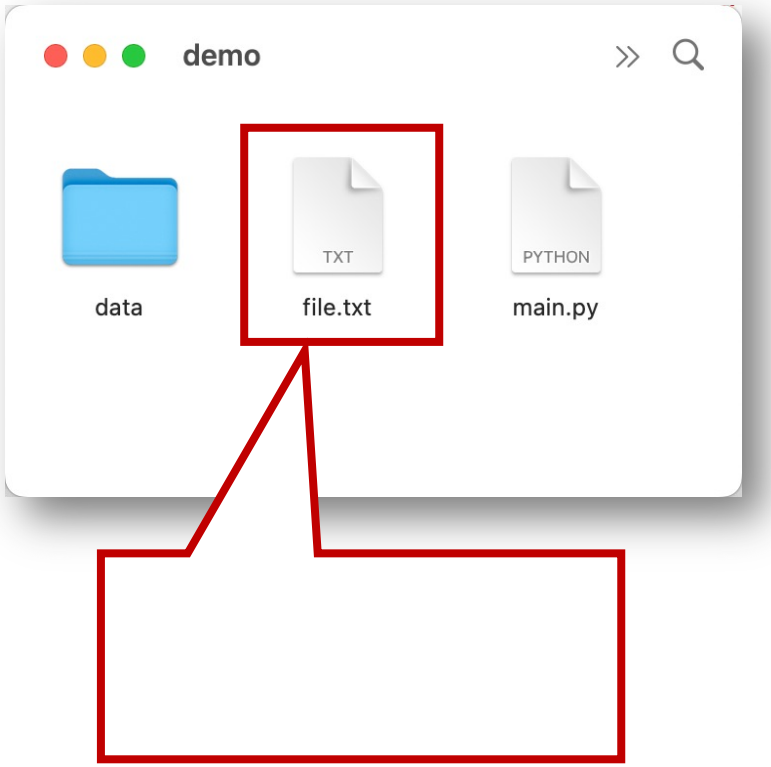
I promise to always close my files

Let's run it!

Writing a file

```
f = open("file.txt",
    "w")
f.write("hello")
f.write(" world\n")
f.write("!!!\n")


f.close()
```

"w" mode indicates to write to this file



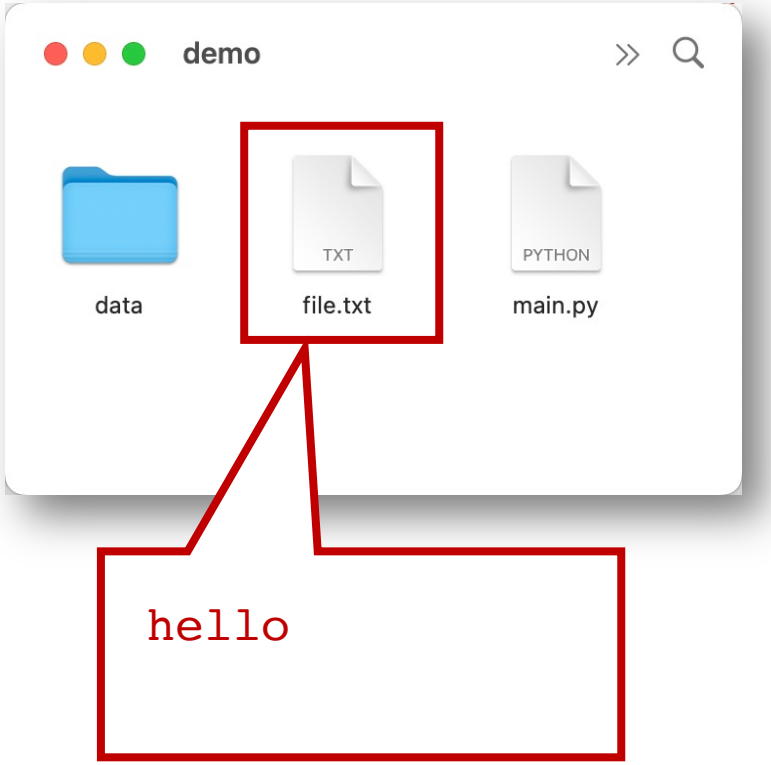
Open with "w" is dangerous. It immediately wipes out your file

(Or create a new one if there isn't already a file.txt)

Writing a file

```
f = open("file.txt",
    "w")
f.write("hello")
f.write(" world\n")
f.write("!!!\n")


f.close()
```

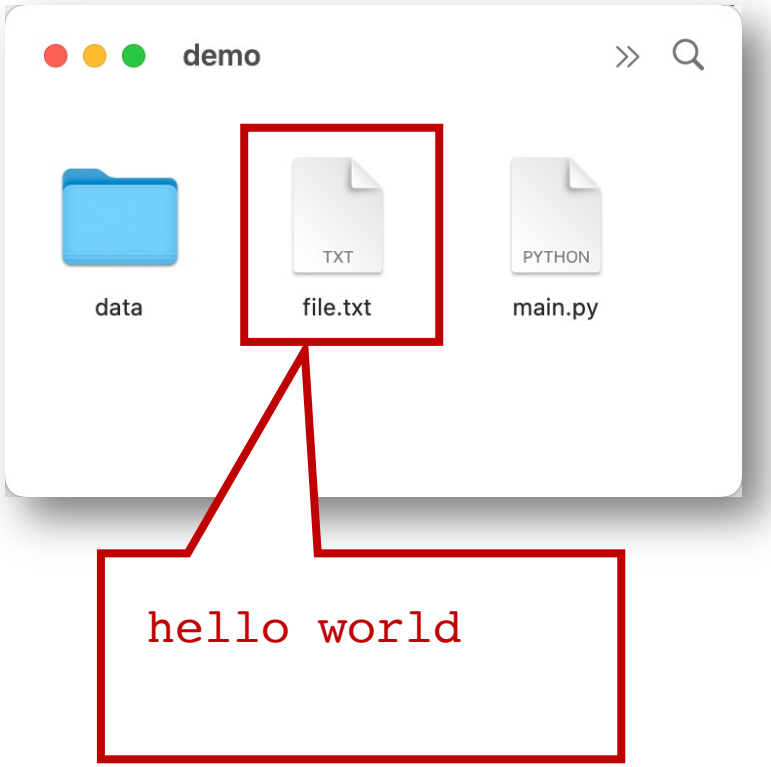
"w" mode indicates to write to this file



Writing a file

```
f = open("file.txt",
    "w")
f.write("hello")
f.write(" world\n")
f.write("!!!\n")


f.close()
```

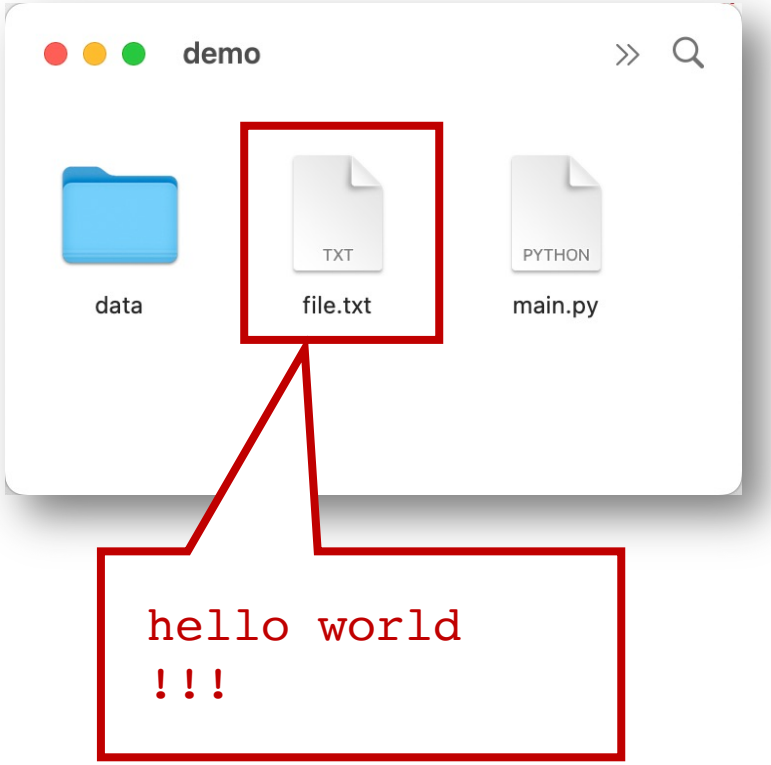
"w" mode indicates to write to this file



Writing a file

```
f = open("file.txt",
    "w")
f.write("hello")
f.write(" world\n")
f.write("!!!\n")


f.close()
```

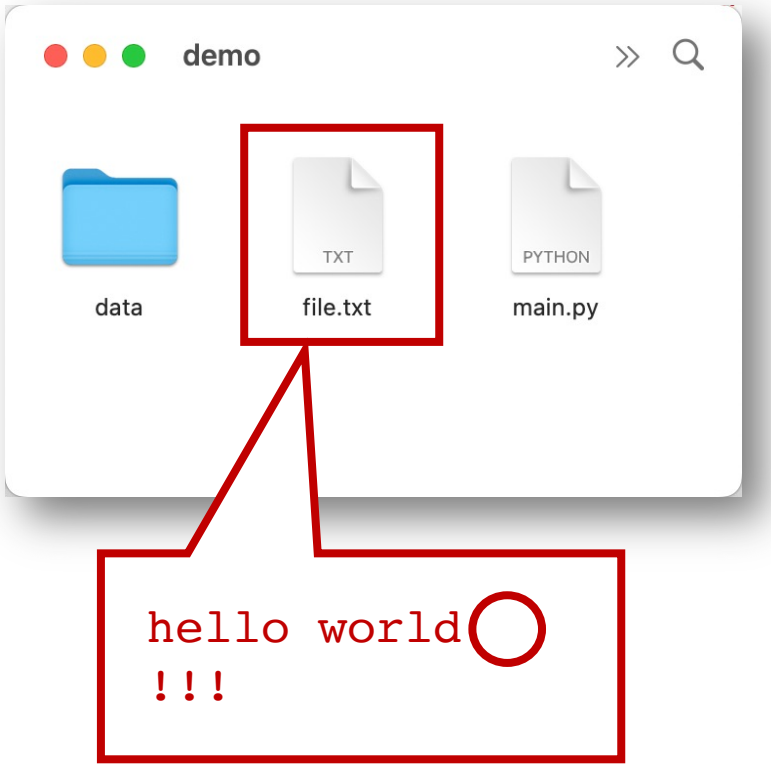
"w" mode indicates to write to this file



Writing a file

```
f = open("file.txt",
    "w")
f.write("hello")
f.write(" world\n")
f.write("!!!\n")



f.close()
```

"w" mode indicates to write to this file



Be careful with newlines
(write doesn't add them like print does)

Basic file interactions

- Basic file system operations
  - opening/closing
  - reading/writing

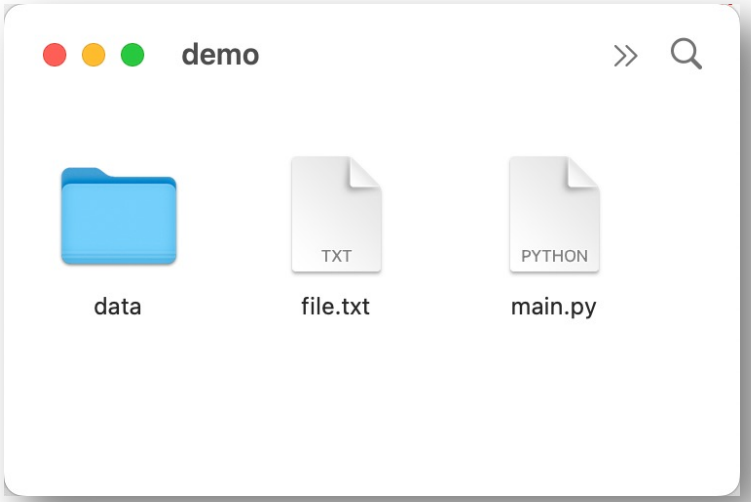
- OS-related module
  - listdir, mkdir, exists, join

OS module

- Many functions in os and os.path for working w/ files
  - os.listdir
  - os.mkdir
  - os.path.exists
  - os.path.join
  - …

OS module

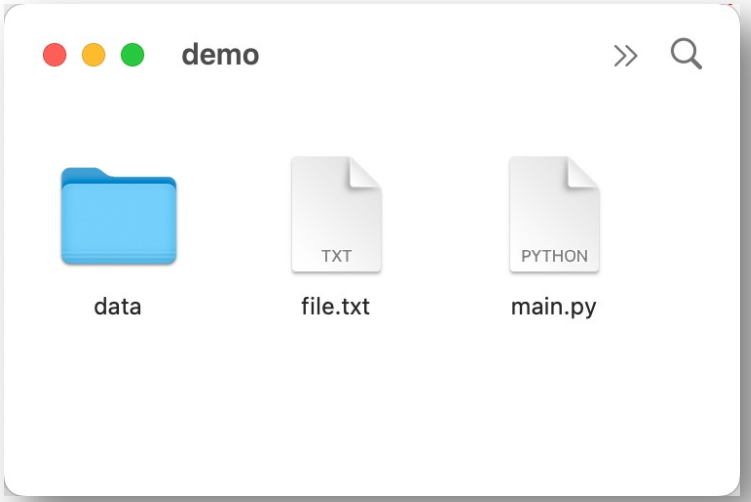
- Many functions in os and os.path for working w/ files
  - **os.listdir**
  - os.mkdir
  - os.path.exists
  - os.path.join
  - …




```
>>> import os
>>> os.listdir(“.”)
[“file.txt”, “main.py”, “data”]
```

OS module

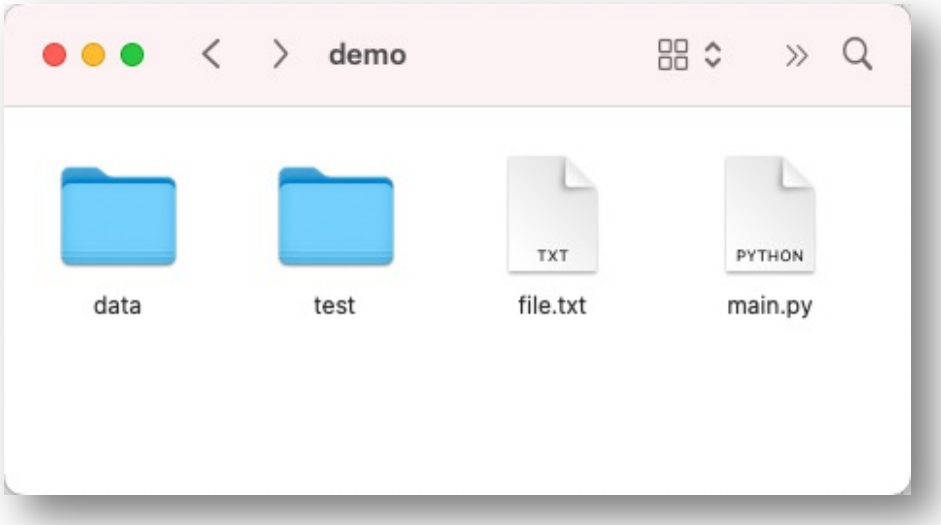
- Many functions in os and os.path for working w/ files
  - os.listdir
  - **os.mkdir**
  - os.path.exists
  - os.path.join
  - …




```
>>> import os
>>> os.mkdir("test")
```

OS module

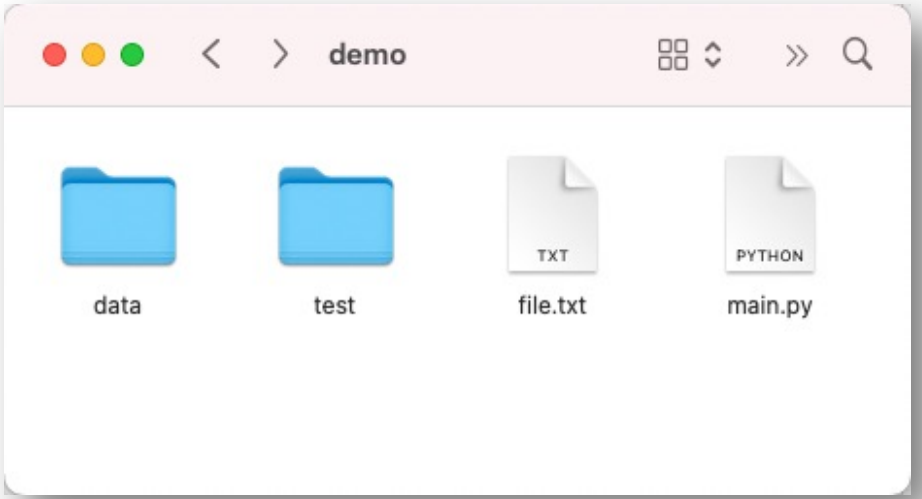
- Many functions in os and os.path for working w/ files
  - os.listdir
  - **os.mkdir**
  - os.path.exists
  - os.path.join
  - …




```
>>> import os
>>> os.mkdir("test")
```

OS module

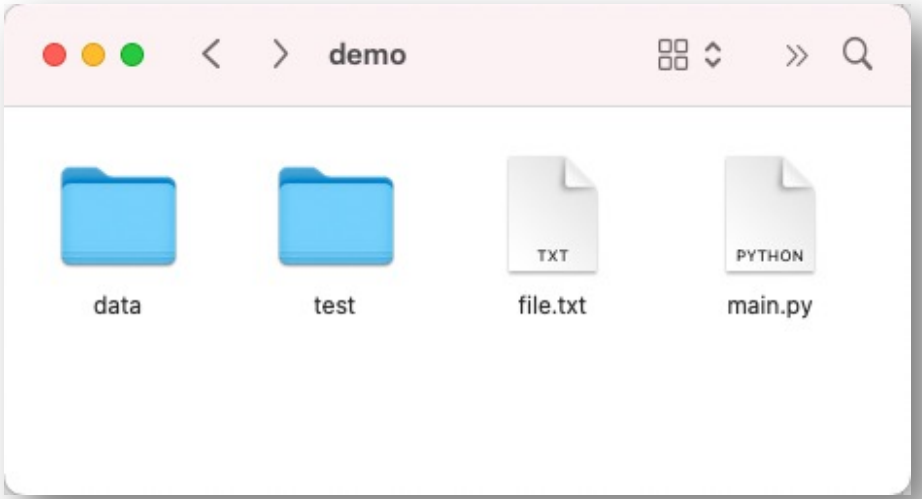
- Many functions in os and os.path for working w/ files
  - os.listdir
  - os.mkdir
  - **os.path.exists**
  - os.path.join
  - …




```
>>> import os
>>> os.path.exists("file.txt")
True
```

OS module

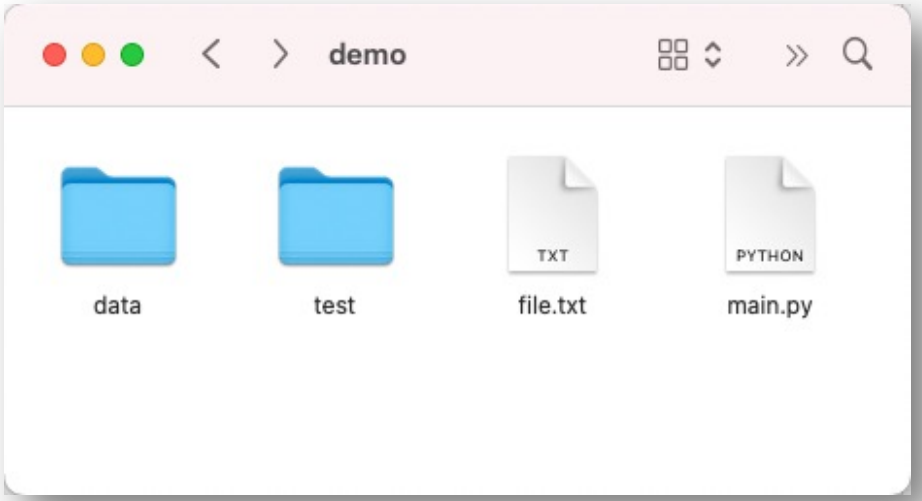
- Many functions in os and os.path for working w/ files
  - os.listdir
  - os.mkdir
  - **os.path.exists**
  - os.path.join
  - …




```
>>> import os
>>> os.path.exists("haha.txt")
False
```

OS module

- Many functions in os and os.path for working w/ files
  - os.listdir
  - os.mkdir
  - os.path.exists
  - **os.path.join**
  - …




```
>>> import os
>>> os.path.join("data", "export.csv")
data/export.csv
```

on Linux/MacOS

on Windows: **`data\export.csv`**

Tabular data: CSVs/JSONs vs. Databases

CSV

State	Capital	Population	Area
VA	Richmond	226,604	42774.2
…	…	…	…

Characteristics

- One table

SQL Database

Capitals

State	Capital
VA	Richmond
…	…

Population

State	Population
VA	226,604
…	…

Areas

State	Area
VA	42774.2
…	…

Characteristics

- Collections of tables, each named

CSV

State	Capital	Population	Area
VA	Richmond	226,604	42774.2
…	…	…	…

Characteristics
- One table
- Columns sometimes named

SQL Database

Capitals

State	Capital
VA	Richmond
…	…

Population

State	Population
VA	226,604
…	…

Areas

State	Area
VA	42774.2
…	…

Characteristics
- Collections of tables, each named
- Columns always named

CSV

State	Capital	Population	Area
string	string	string	string
string	string	string	string
string	string	string	string
string	string	string	string

Characteristics
- One table
- Columns sometimes named
- Everything is a **string**

SQL Database

Capitals

State	Capital
text	text
text	text
text	text
text	text

Population

State	Population
text	integer
text	integer
text	integer
text	integer

No text allowed

Areas

State	Area
text	real
text	real
text	real
text	real

Characteristics
- Collections of tables, each named
- Columns always named
- Types per column (enforced)

Why use a database?

1. More relations

Database

A	B	C
text	integer	real
text	integer	real
text	integer	real
text	integer	real

Same fields and same types in every column

CSV

```
A,B,C
string,string,string
string,string,string
string,string,string
string,string,string
```

Everything is a **string**

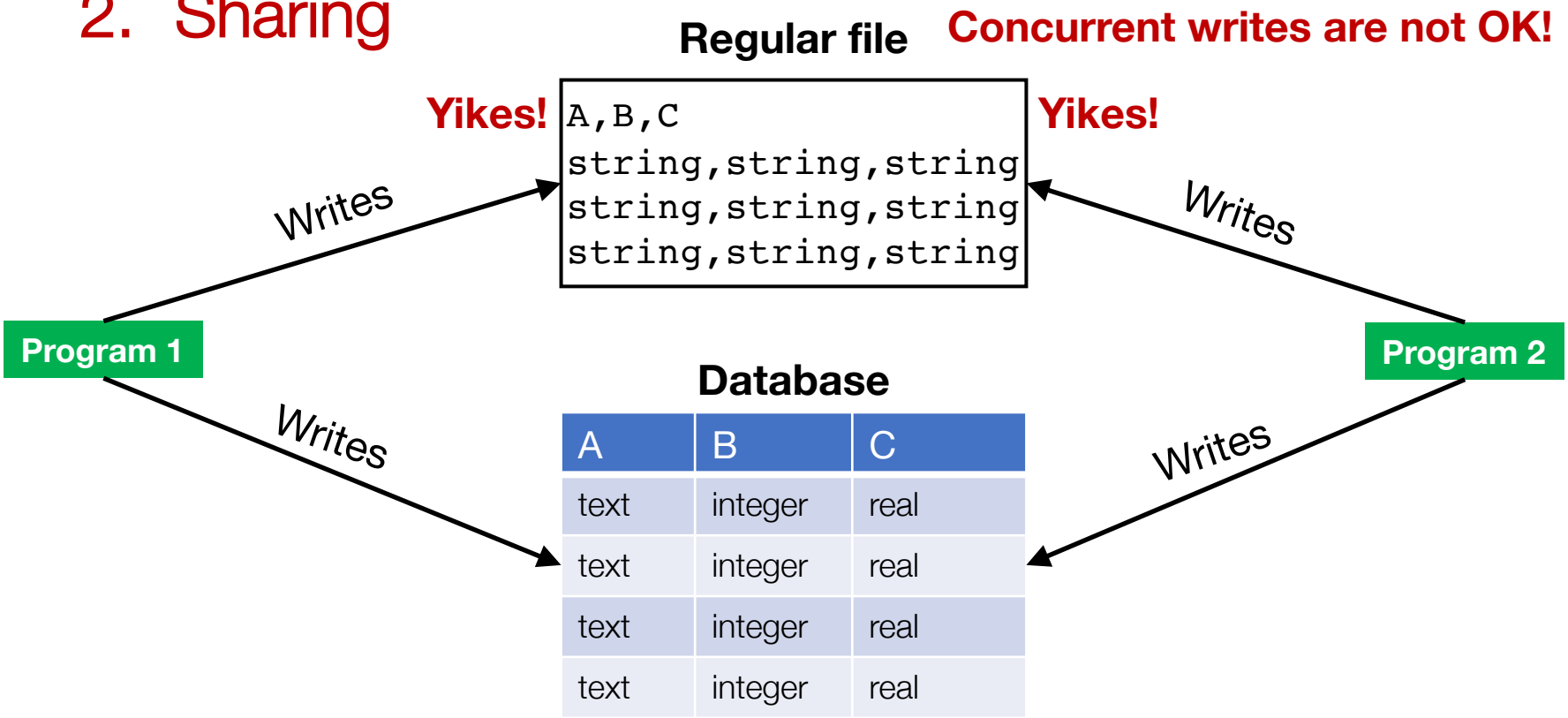
JSON

```
[{“A”:“val”,“B”:10,“C”:2.1},
{“A”:“val”},
{“A”:“v3”,“B”:2,“C”:True}]
```

Types, but…
Semi-structured data
Missing values
Types may differ across columns

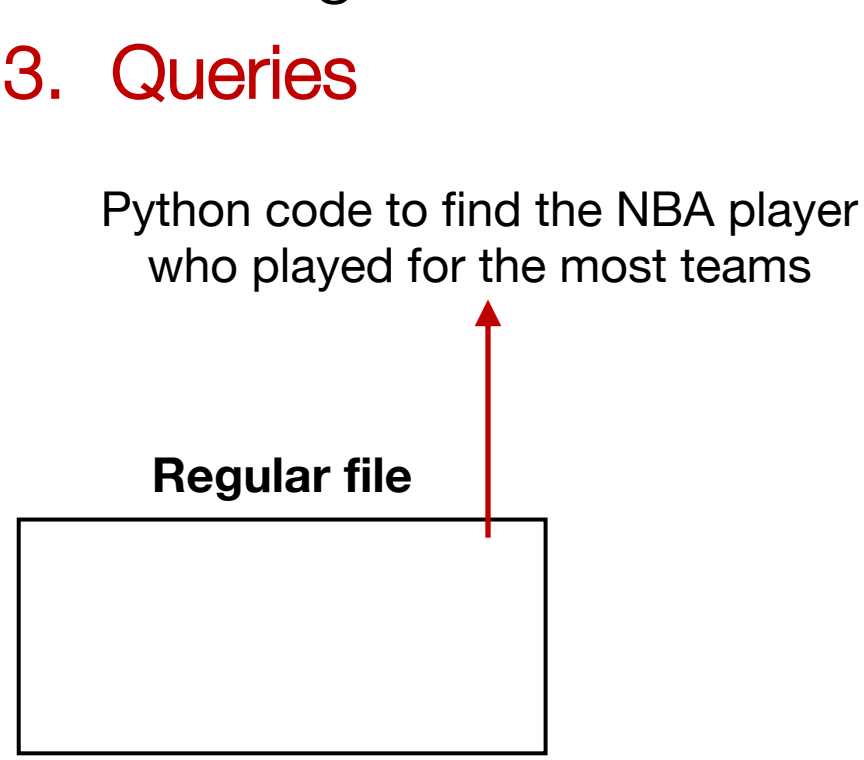
Why use a database?

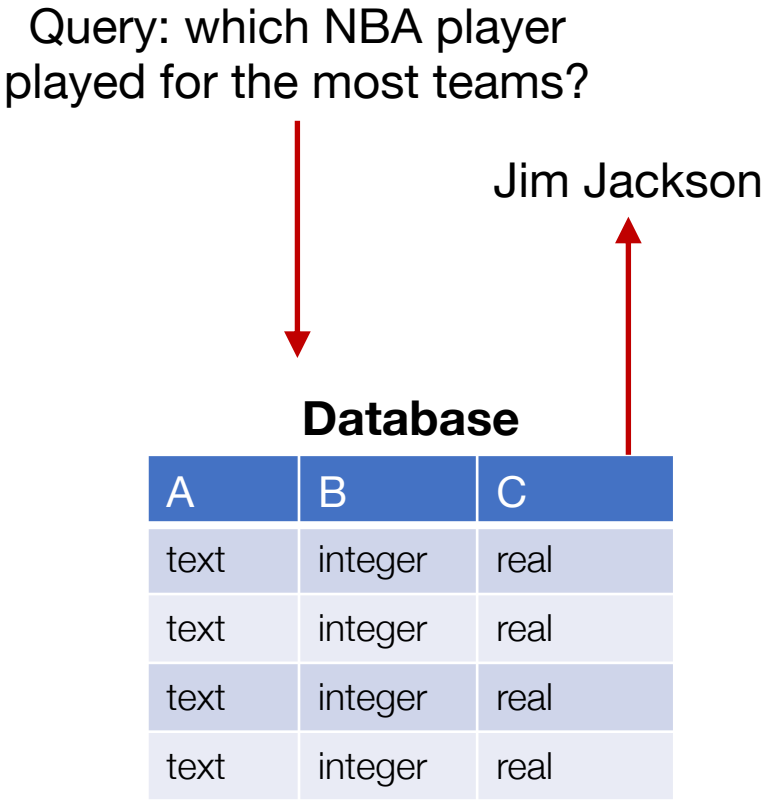
1. More relations
2. Sharing



Why use a database?

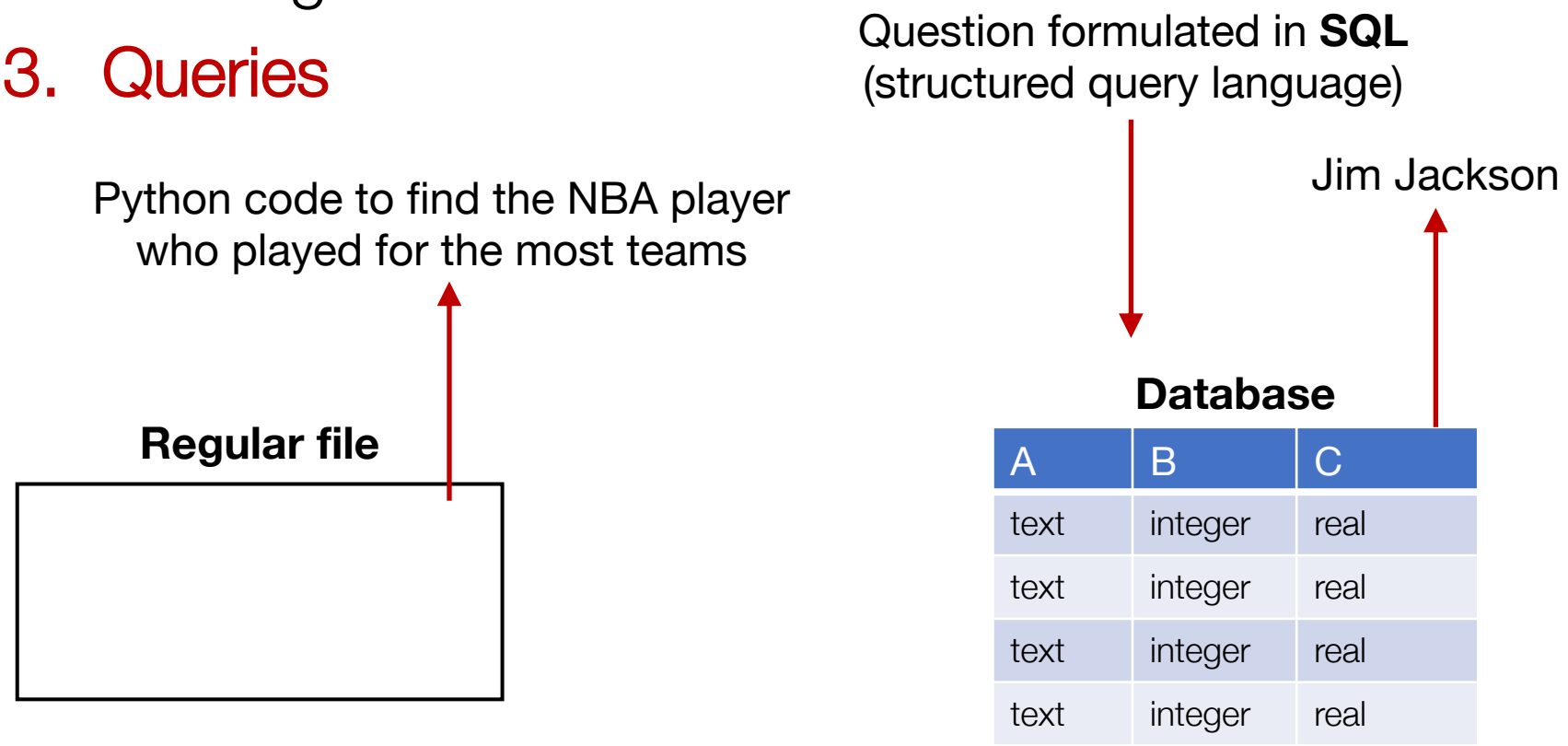
1. More relations
2. Sharing
3. Queries





Why use a database?

1. More relations
2. Sharing
3. Queries



Why use a database?

1. More relations
2. Sharing
3. Queries
4. Performance

Why use a database?

1. More relations
2. Sharing
3. Queries
4. Performance

Exercise:

- I'm going to show a table and ask you **two questions**
- You get out your **stop watch**
- Answer both questions, measuring how long each took you

Name	Age	Score
Parker	26	21
Heidy	22	22
Shirly	27	22
Arla	21	22
Bella	22	22
Bill	28	22
Hollis	26	23
Maurita	22	24
Milda	22	25
Pearline	29	25
Teresa	25	25
Ceola	30	26
Milford	25	26
Alisha	30	27
Antonio	28	28
Ryan	25	28
Karma	23	28
Breana	21	30
Sara	26	30

Question 1:

How many people are 23 or younger?

Question 2:

How many people scored 23 or less?

Name	Age	Score
Parker	26	21
Heidy	22	22
Shirly	27	22
Arla	21	22
Bella	22	22
Bill	28	22
Hollis	26	23
Maurita	22	24
Milda	22	25
Pearline	29	25
Teresa	25	25
Ceola	30	26
Milford	25	26
Alisha	30	27
Antonio	28	28
Ryan	25	28
Karma	23	28
Breana	21	30
Sara	26	30

Question 1:

How many people are 23 or younger?

Question 2:

How many people scored 23 or less?

Which question took longer? Why?

Name	Age	Score
Parker	26	21
Heidy	22	22
Shirly	27	22
Arla	21	22
Bella	22	22
Bill	28	22
Hollis	26	23
Maurita	22	24
Milda	22	25
Pearline	29	25
Teresa	25	25
Ceola	30	26
Milford	25	26
Alisha	30	27
Antonio	28	28
Ryan	25	28
Karma	23	28
Breana	21	30
Sara	26	30

DBs can keep multiple copies of the same data

- Which copy to use is **automatically determined** based on the question (query) being asked

Name	Age	Score
Arla	21	22
Breana	21	30
Heidy	22	22
Bella	22	22
Maurita	22	24
Milda	22	25
Karma	23	28
Teresa	25	25
Milford	25	26
Ryan	25	28
Parker	26	21
Hollis	26	23
Sara	26	30
Shirly	27	22
Bill	28	22
Antonio	28	28
Pearline	29	25
Ceola	30	26
Alisha	30	27

Copy 1

Name	Age	Score
Parker	26	21
Heidy	22	22
Shirly	27	22
Arla	21	22
Bella	22	22
Bill	28	22
Hollis	26	23
Maurita	22	24
Milda	22	25
Pearline	29	25
Teresa	25	25
Ceola	30	26
Milford	25	26
Alisha	30	27
Antonio	28	28
Ryan	25	28
Karma	23	28
Breana	21	30
Sara	26	30

Copy 2

Why use a database?

1. More relations
2. Sharing
3. Queries
4. Performance

Why use a database?

1. More relations
2. Sharing
3. Queries
4. Performance

Why not use a database?

1. It's often an overkill and too heavyweight
2. For many situations, writing ad-hoc Python code on a simple CSV/JSON file is easier to use