[544] File Systems and File Formats

Tyler Caraza-Harter

Learning Objectives

- describe different kinds of file systems
- interpret the output of tools like "mount" and "df" to understand the structure of a mount namespace
- describe different file formats in terms of orientation, encoding, compression, and schemas
- differentiate between transactions workloads and analytics workloads

Outline

File Systems

FS Demos

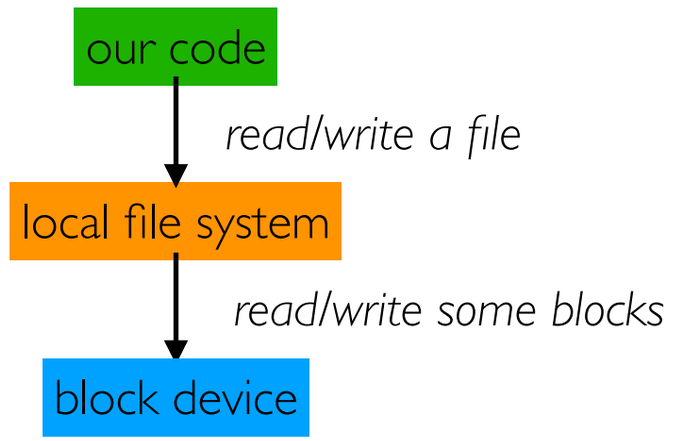
File Formats

Format Demos

File Systems

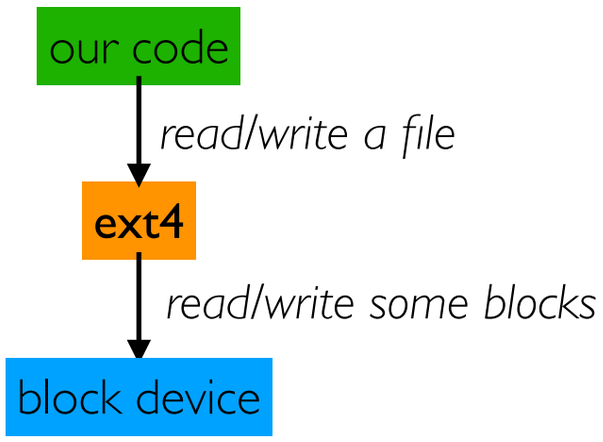
Difficult: writing code to store data in blocks
Easier: writing code to store data in files

Files systems *abstract* storage for us. We write to data **blocks** without thinking about it by writing data to **files** in a local file system.

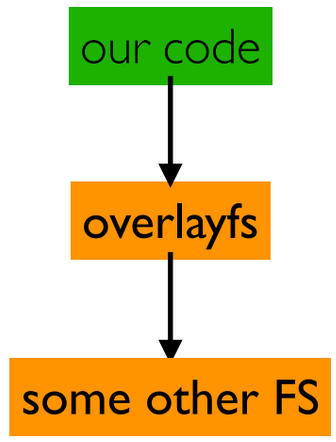


Types of File System (FS)

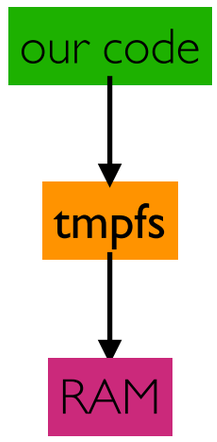
local FS



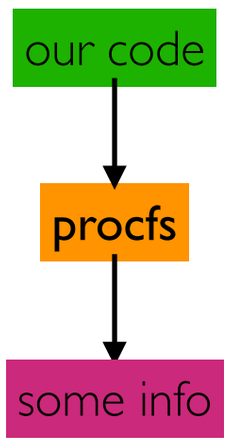
layered FS (for Docker)



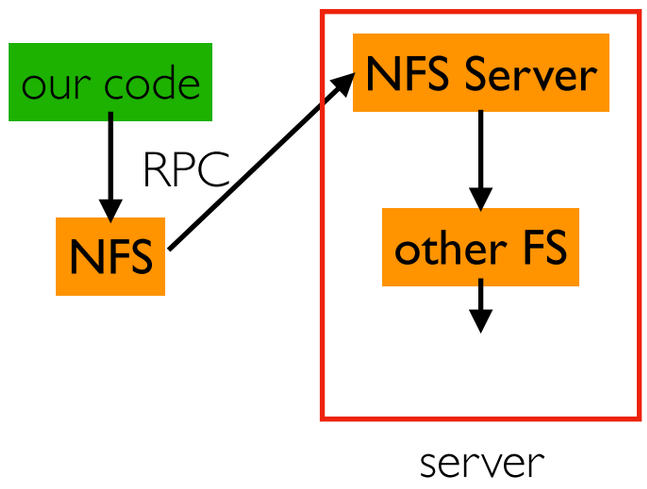
in-memory FS (Temp Files)



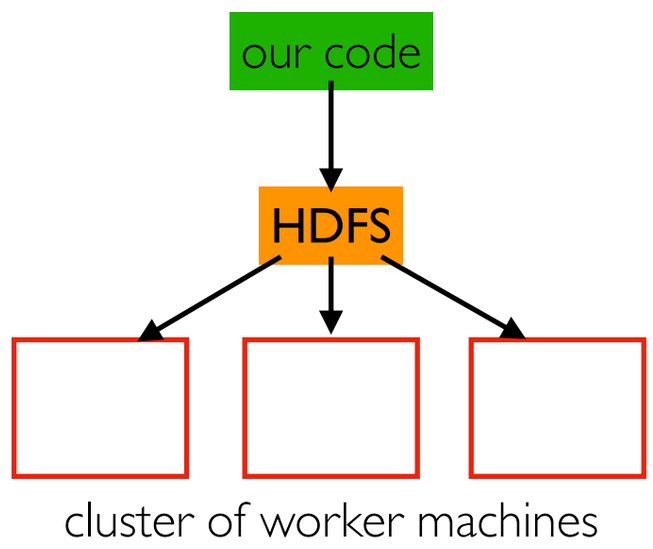
pseudo FS (Stats)



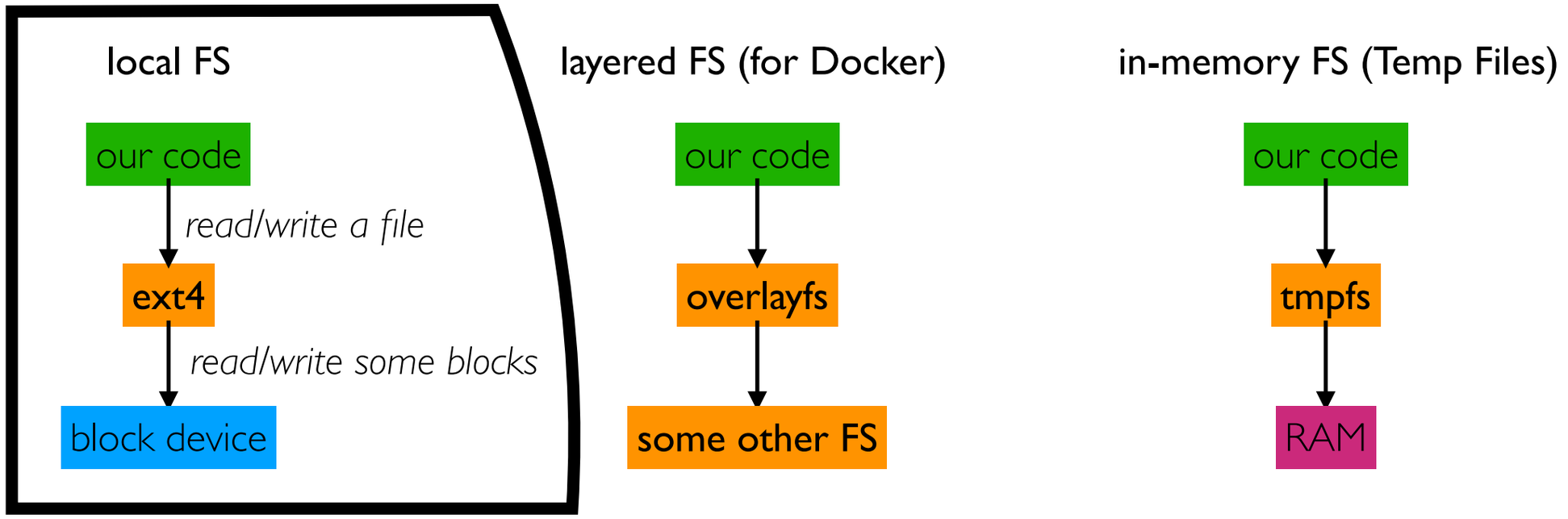
network FS

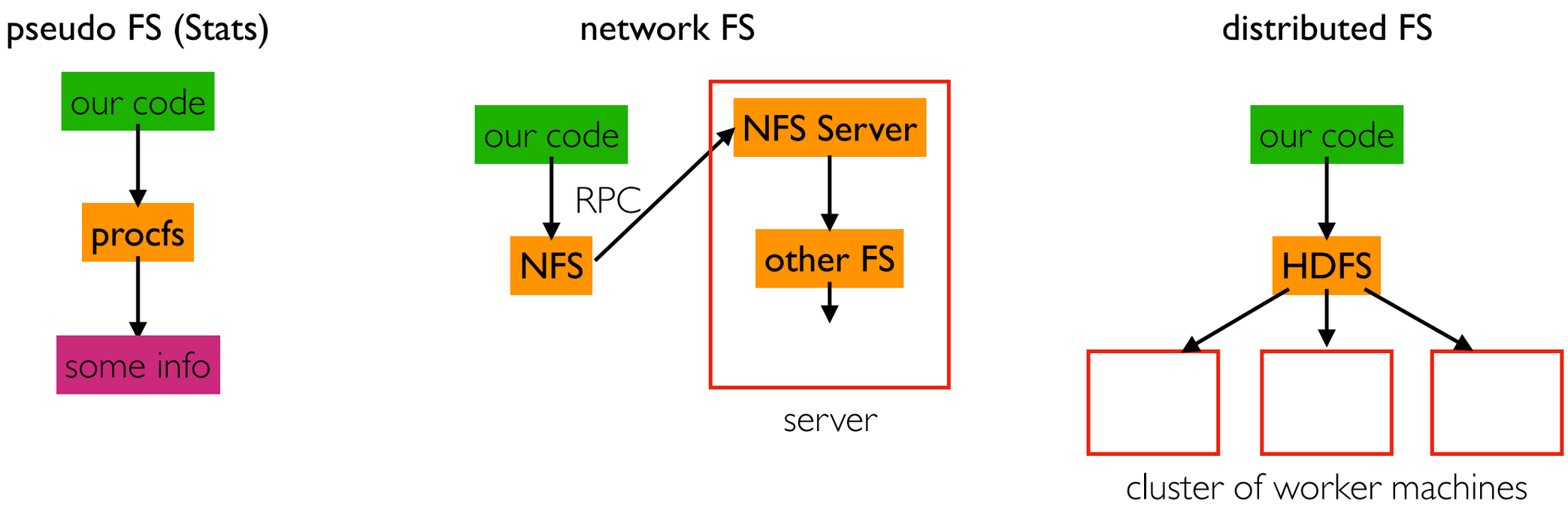


distributed FS

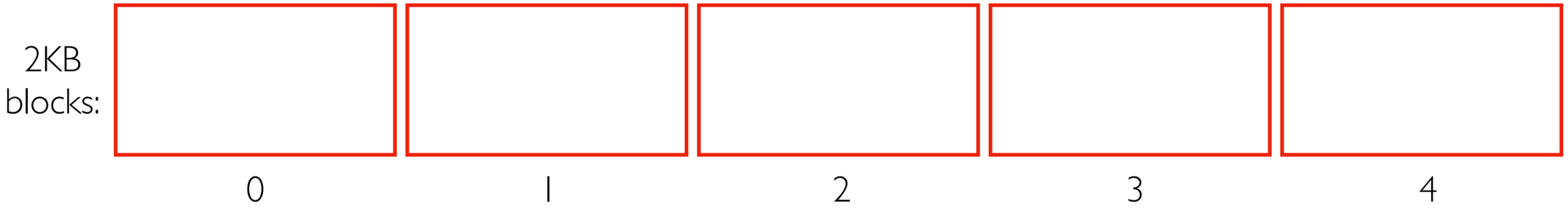


Types of File System (FS)



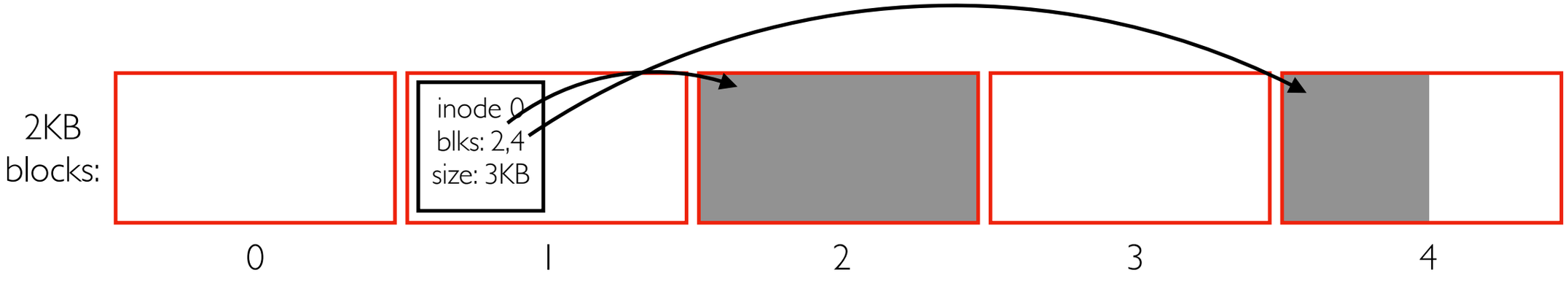


Local File Systems



How does a local FS use blocks?

Local File Systems

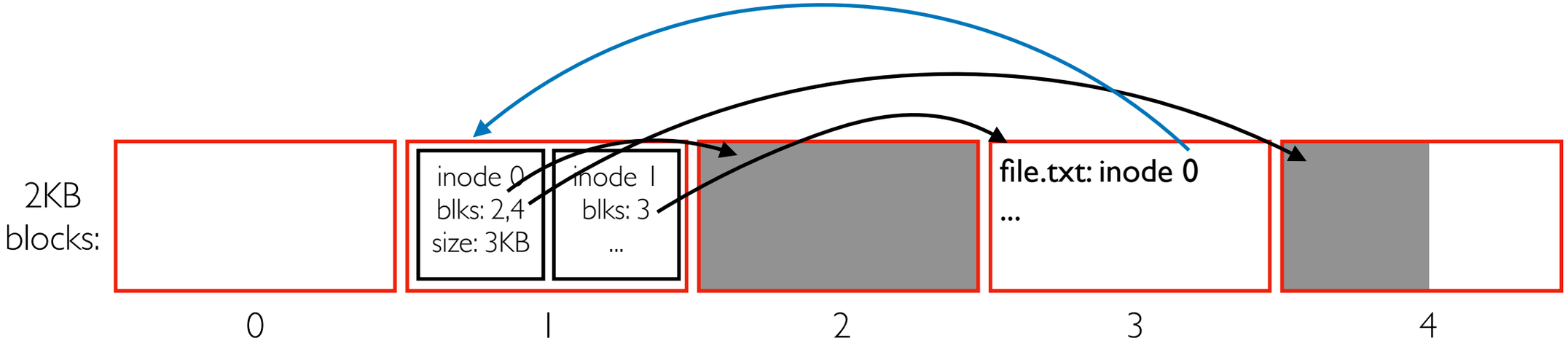


How does a local FS use blocks? Many possibilites. One example...

Files

- some metadata, like size, block locations
- each is represented by an "inode" structure (above file is fragmented)

Local File Systems



How does a local FS use blocks? Many possibilites. One example...

Files

- some metadata, like size, block locations
- each is represented by an "inode" structure (above file is fragmented)

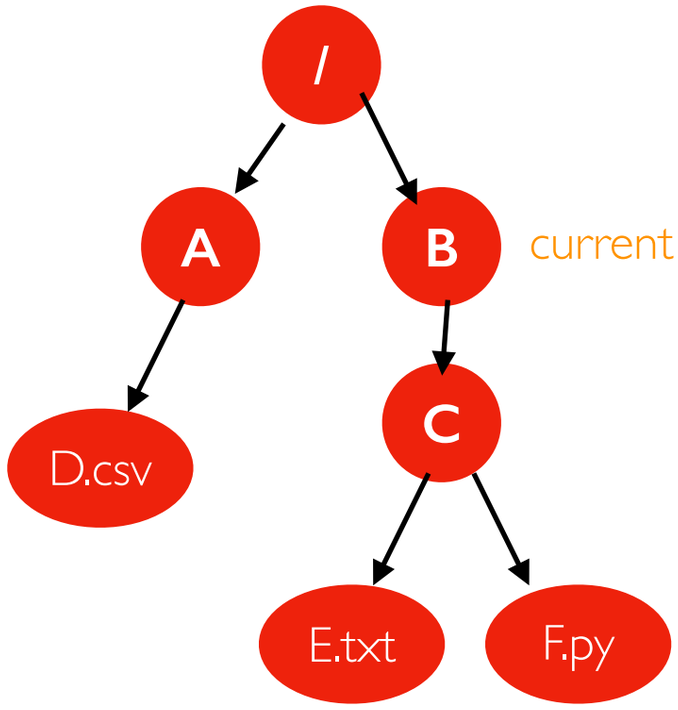
Directories

- special files containing name => inode mappings
- file extensions (like .txt) don't mean anything to the file system (just for documentation)
- the same inode could be in multiple directories
- each file system has a "root" directory from which you can reach everything else recursively
- formatting a disk creates initial structures (like the root directory)

File System Trees

Nesting of directories and files logically create "trees"

- technically DAGs (directed acyclic graphs) because the same inode number can have multiple names in different directories
- leaves: files and empty directories

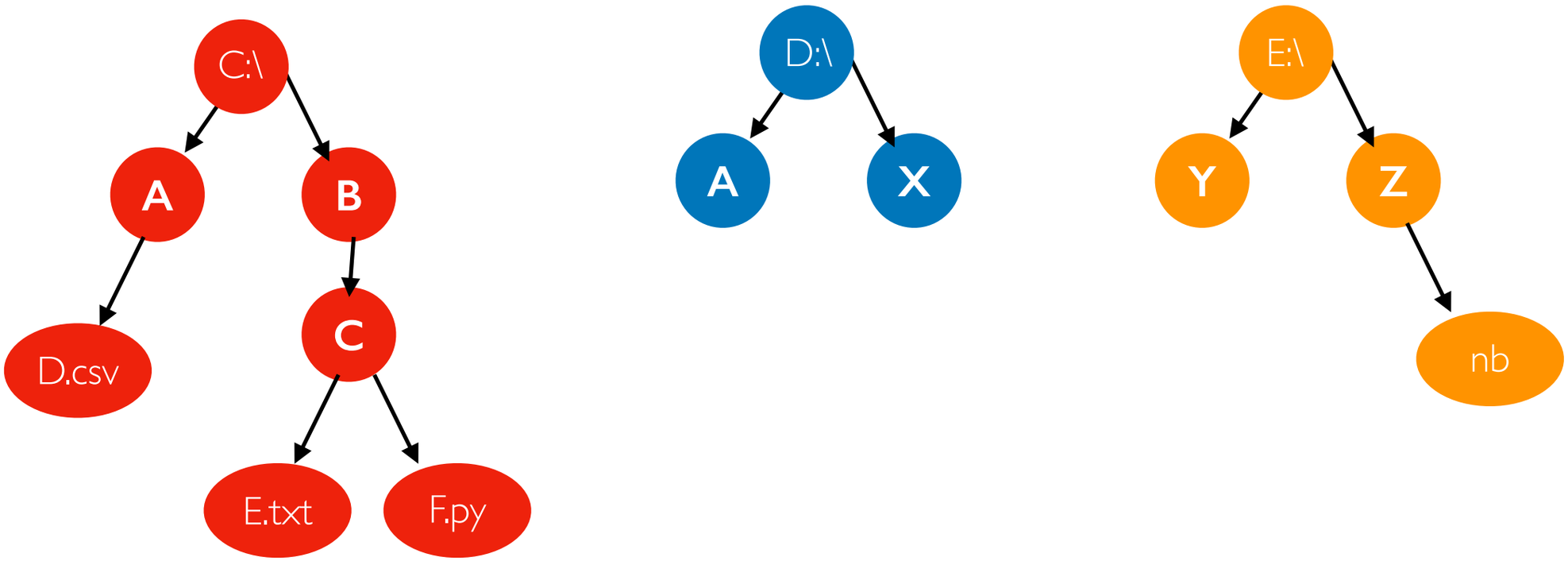


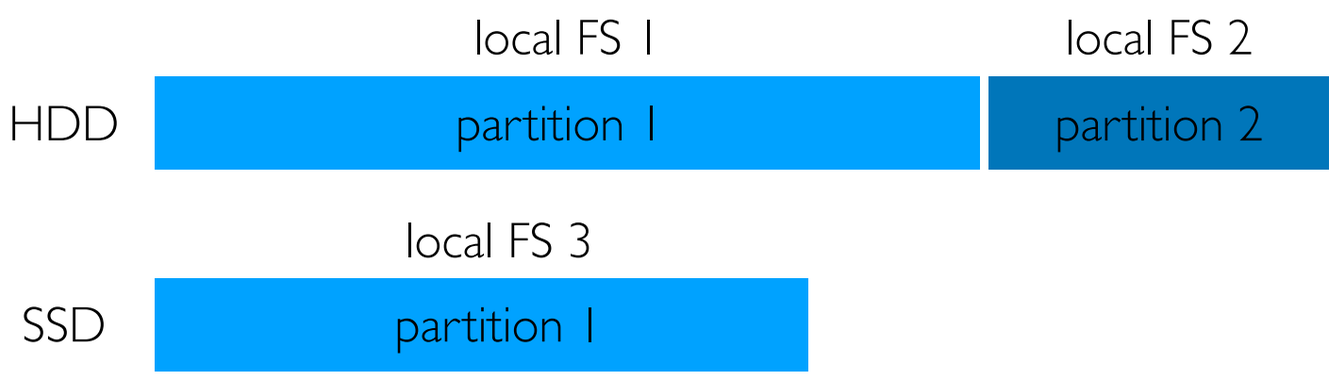
current working directory (CWD)

relative path to E.txt: `C/E.txt`
absolute path to E.txt: `/B/C/E.txt`
relative path to D.csv: `../A/D.csv`
absolute path to D.csv: `TopHat`

Multiple File Systems: Windows Approach

have multiple trees (each is a "drive")





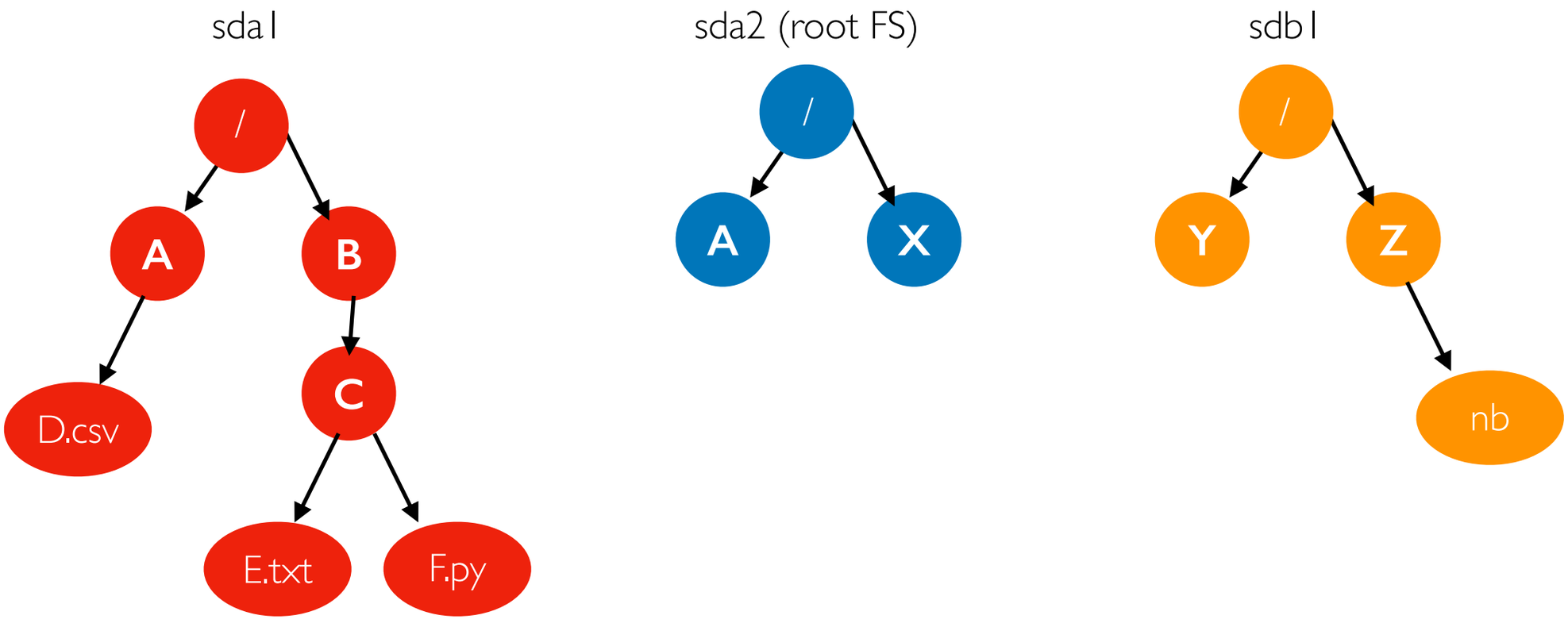
Multiple File Systems: Unix Approach

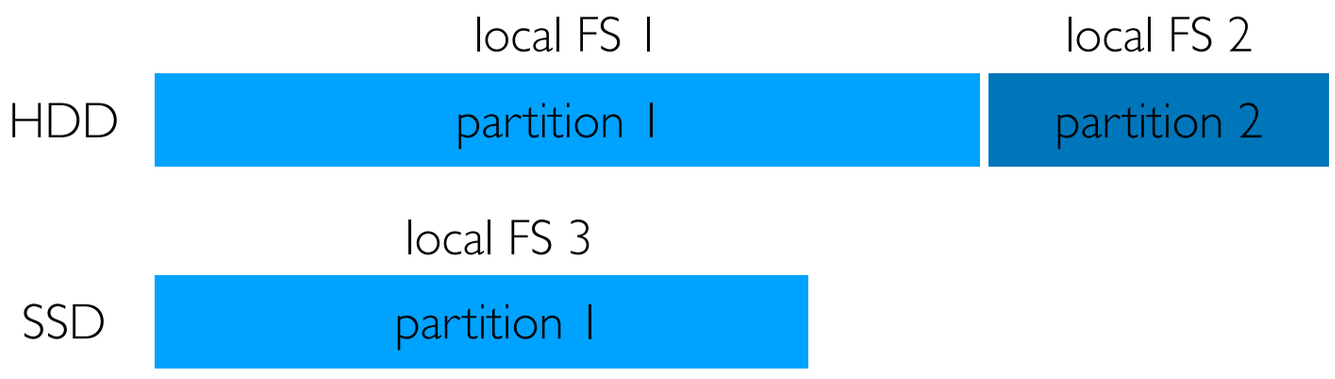
mount file systems over directories of other file systems to make one big tree

https://www.brit.co/fruit-salad-tree/

Multiple File Systems: Unix Approach

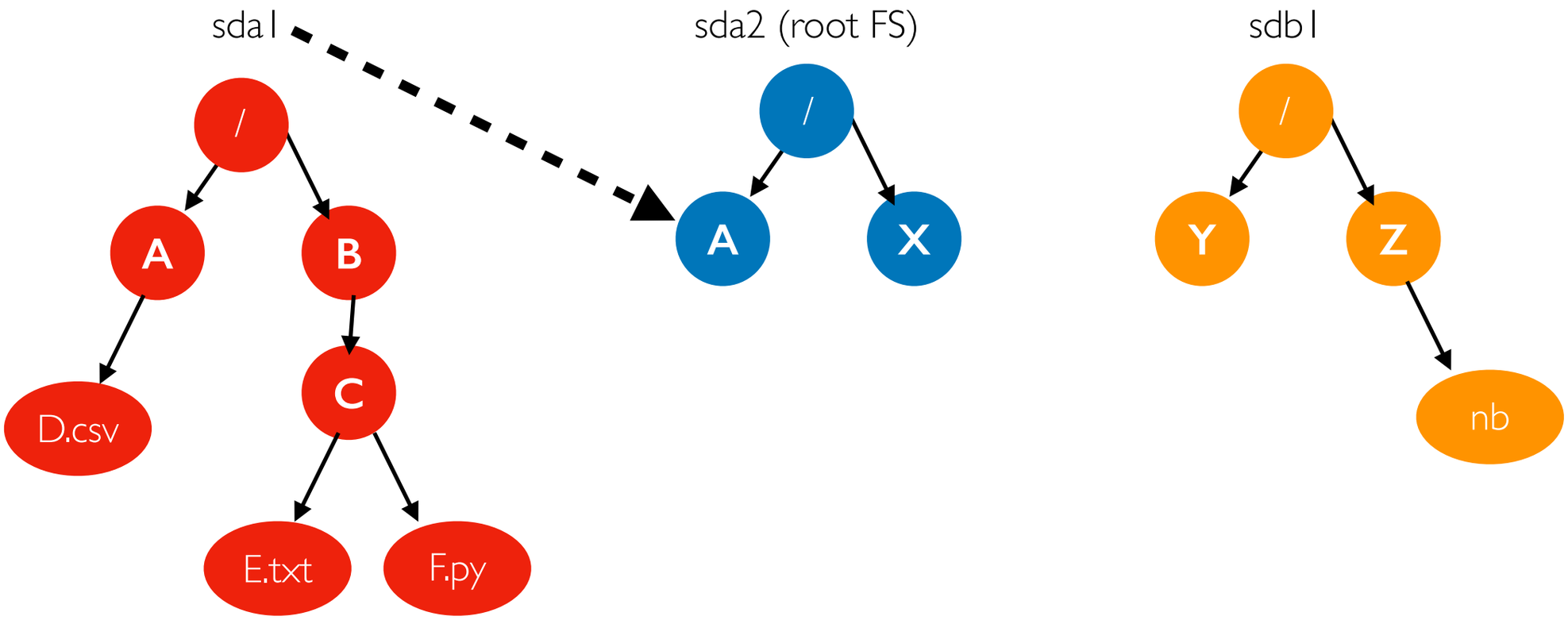
mount file systems over directories of other file systems to make one big tree





Multiple File Systems: Unix Approach

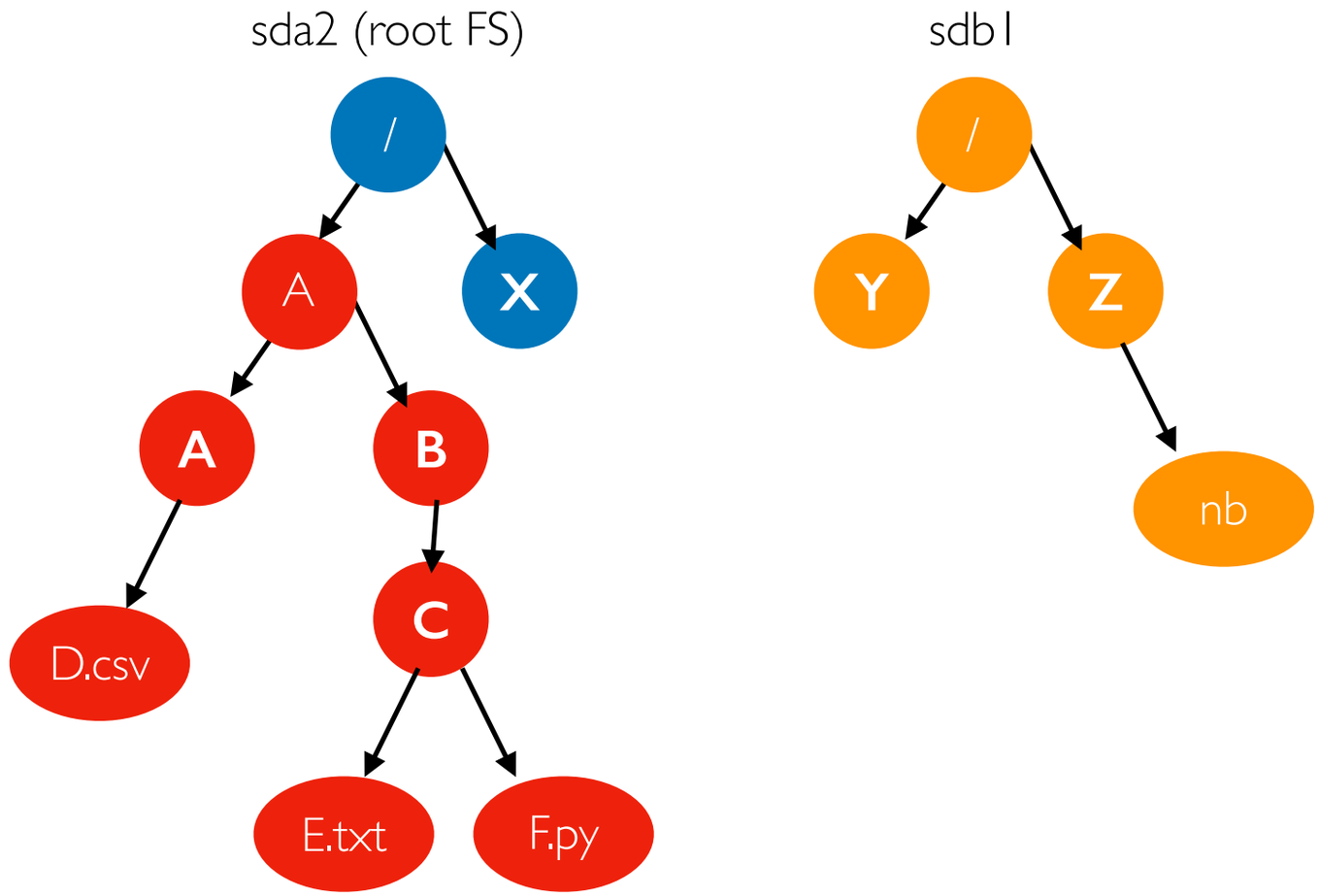
mount file systems over directories of other file systems to make one big tree




```
mount /dev/sda1 /A
```

Multiple File Systems: Unix Approach

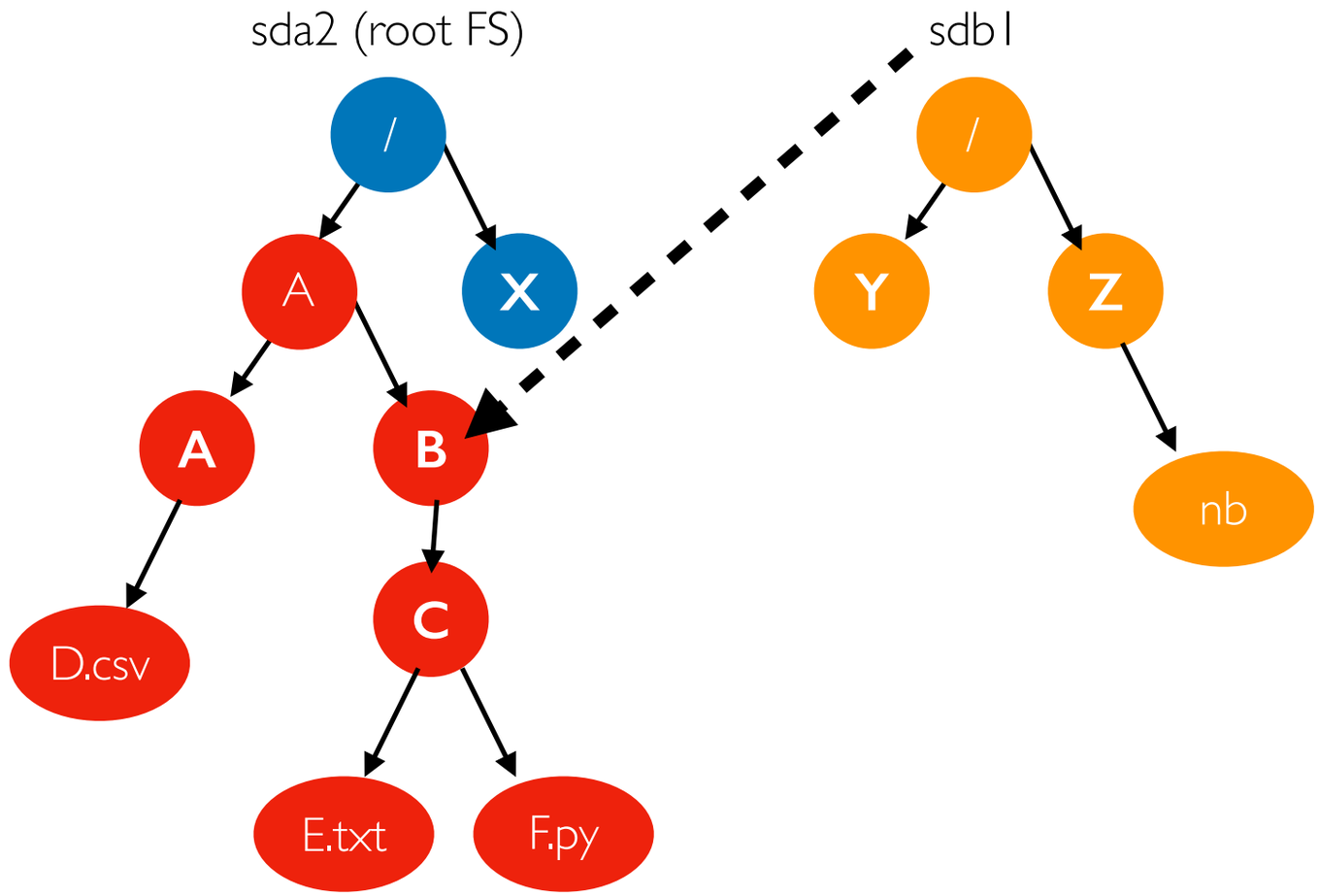
mount file systems over directories of other file systems to make one big tree




```
mount /dev/sda1 /A
```

Multiple File Systems: Unix Approach

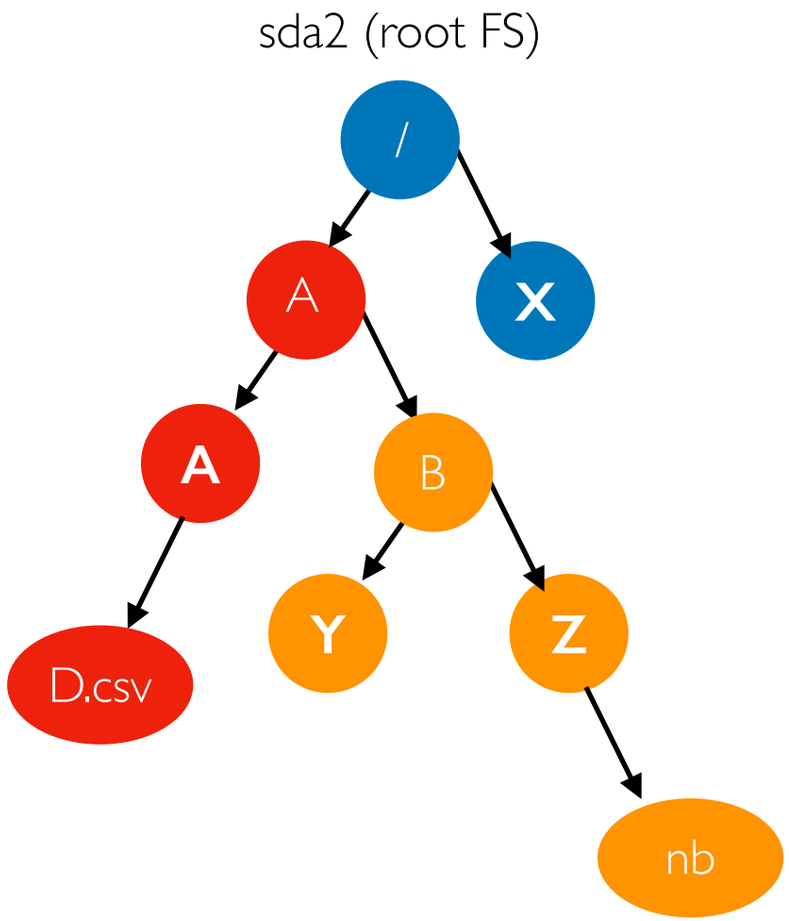
mount file systems over directories of other file systems to make one big tree




```
mount /dev/sda1 /A
mount /dev/sdb1 /A/B
```

Multiple File Systems: Unix Approach

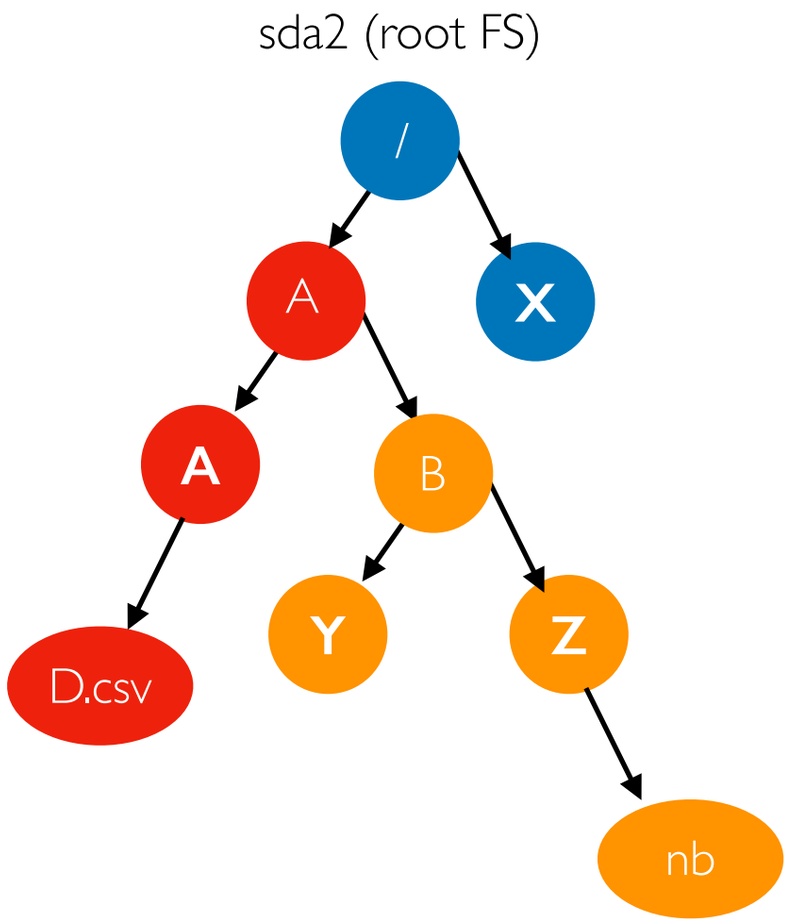
mount file systems over directories of other file systems to make one big tree




```
mount /dev/sda1 /A
mount /dev/sdb1 /A/B
```

Multiple File Systems: Unix Approach

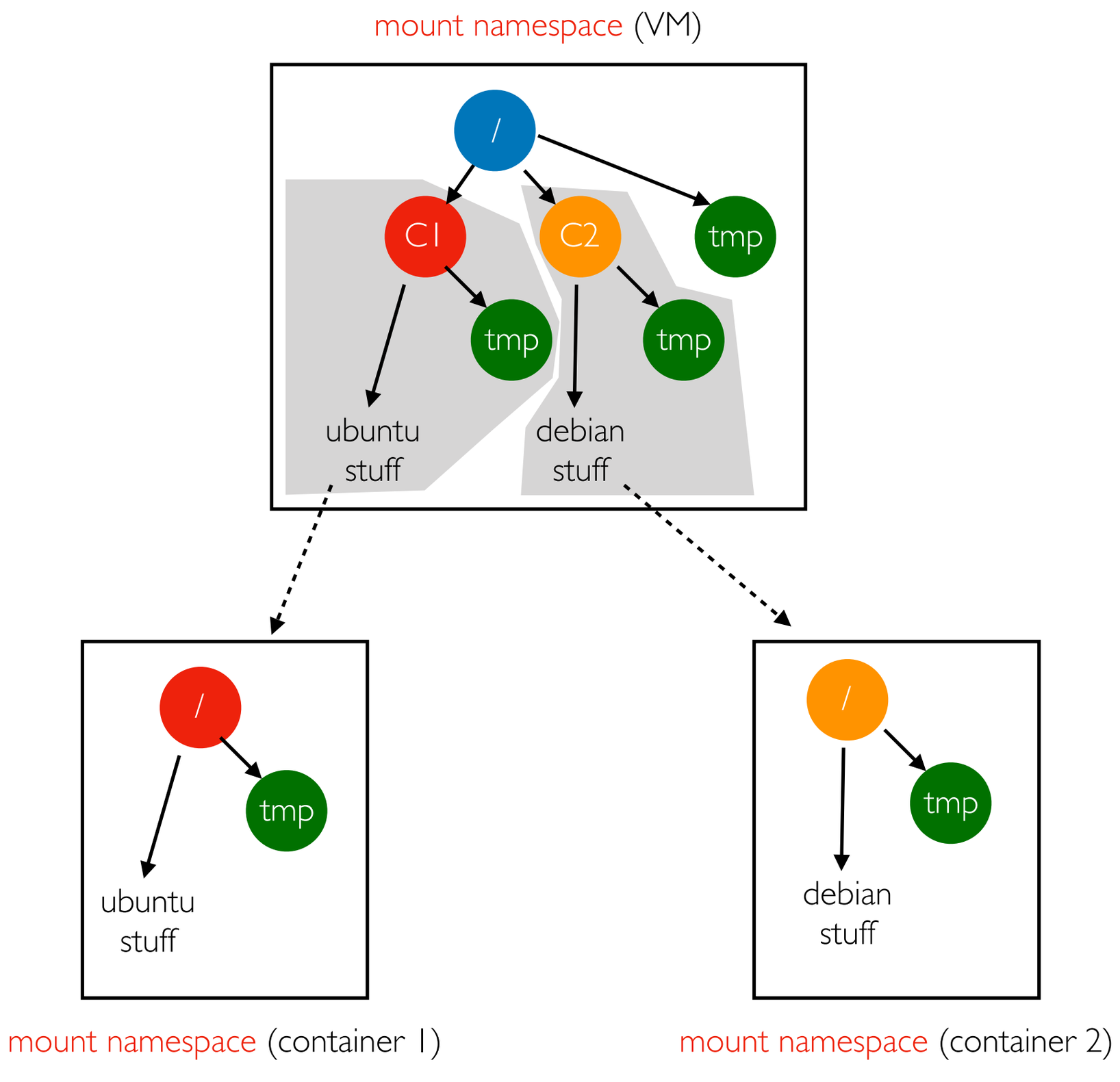
mount file systems over directories of other file systems to make one big tree



Note: each container has it's own root file system and mount namespace

```
mount /dev/sda1 /A
mount /dev/sdb1 /A/B
```

Container File Systems (Simplified)



Outline

File Systems

FS Demos

File Formats

Format Demos

Outline

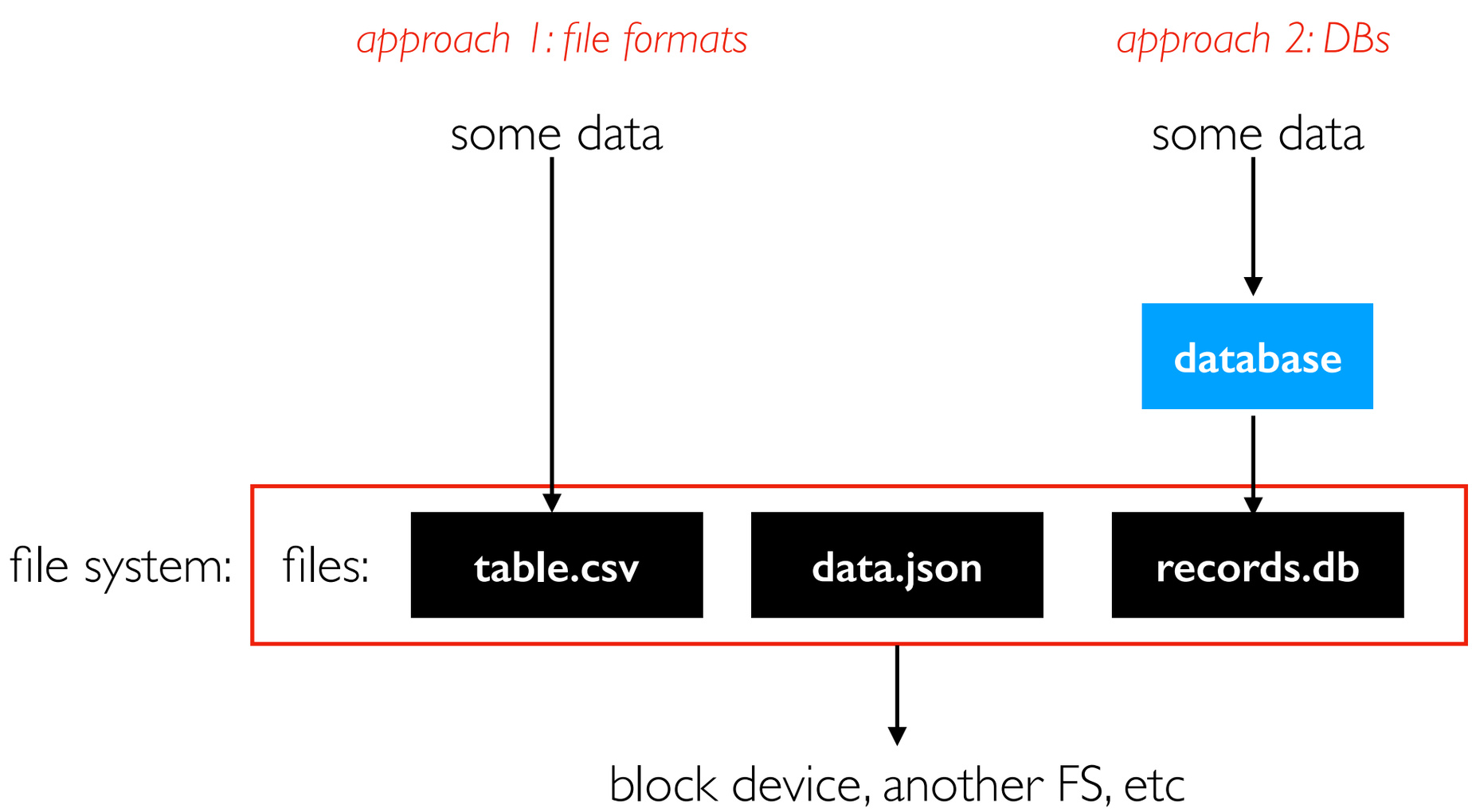
File Systems

FS Demos

File Formats

Format Demos

File systems let us give names to sequences of bytes (files) and hierarchically organize those files (via directories). We usually want some structure for those bytes.



Outline

File Systems

FS Demos

File Formats

	CSV	Parquet
orientation	row	column
encoding	text	binary
compression	none	snappy
schemas	inferred	explicit

Format Demos

Databases

File Layout

Goals

- efficient input/output from storage (large enough reads/writes, sequential accesses)
- minimize parsing/deserialization computation time

Assumptions

- many file systems will try to map consecutive bytes of a file to consecutive blocks on a storage device (but note that in some cases sequential file I/O becomes random disk I/O)
- need to clarify assumptions about how code will access the data (for example, one whole column? a row at a time?)

```
ACW00011604  17.1167  -61.7833   10.1    ST JOHNS COOLIDGE FLD
ACW00011647  17.1333  -61.7833   19.2    ST JOHNS
AE000041196  25.3330   55.5170   34.0    SHARJAH INTER. AIRP          GSN     41196
AEM00041194  25.2550   55.3640   10.4    DUBAI INTL                           41194
AEM00041217  24.4330   54.6510   26.8    ABU DHABI INTL                       41217
AEM00041218  24.2620   55.6090  264.9    AL AIN INTL                          41218
AF000040930  35.3170   69.0170 3366.0    NORTH-SALANG                 GSN     40930
AFM00040938  34.2100   62.2280  977.2    HERAT                                40938
AFM00040948  34.5660   69.2120 1791.3    KABUL INTL                           40948
AFM00040990  31.5000   65.8500 1010.0    KANDAHAR AIRPORT                     40990
AG000060390  36.7167    3.2500   24.0    ALGER-DAR EL BEIDA           GSN     60390
AG000060590  30.5667    2.8667  397.0    EL-GOLEA                     GSN     60590
AG000060611  28.0500    9.6331  561.0    IN-AMENAS                    GSN     60611
AG000060680  22.8000    5.4331 1362.0    TAMANRASSET                  GSN     60680
AGE00135039  35.7297    0.6500   50.0    ORAN-HOPITAL MILITAIRE
AGE00147704  36.9700    7.7900  161.0    ANNABA-CAP DE GARDE
AGE00147705  36.7800    3.0700   59.0    ALGIERS-VILLE/UNIVERSITE
AGE00147706  36.8000    3.0300  344.0    ALGIERS-BOUZAREAH
```

ghcnd-stations.txt

bad: need to read everything to access any one column

good: just read the one block containing the row

File Layout

Goals

- efficient input/output from storage (large enough reads/writes, sequential accesses)
- minimize parsing/deserialization computation time

Assumptions

- many file systems will try to map consecutive bytes of a file to consecutive blocks on a storage device (but note that in some cases sequential file I/O becomes random disk I/O)
- need to clarify assumptions about how code will access the data (for example, one whole column? a row at a time?)

Major access patterns

- transactions processing: *reading/changing a row (or few rows) as needed by an application* (note: "transaction" has other meanings for databases as well -- more later...)
- analytics processing: *computing over many rows for specific columns*

Row-Oriented vs. Column-Oriented Layout

col1	col2	col3
1	5	A
2	6	B
3	7	C
4	8	D

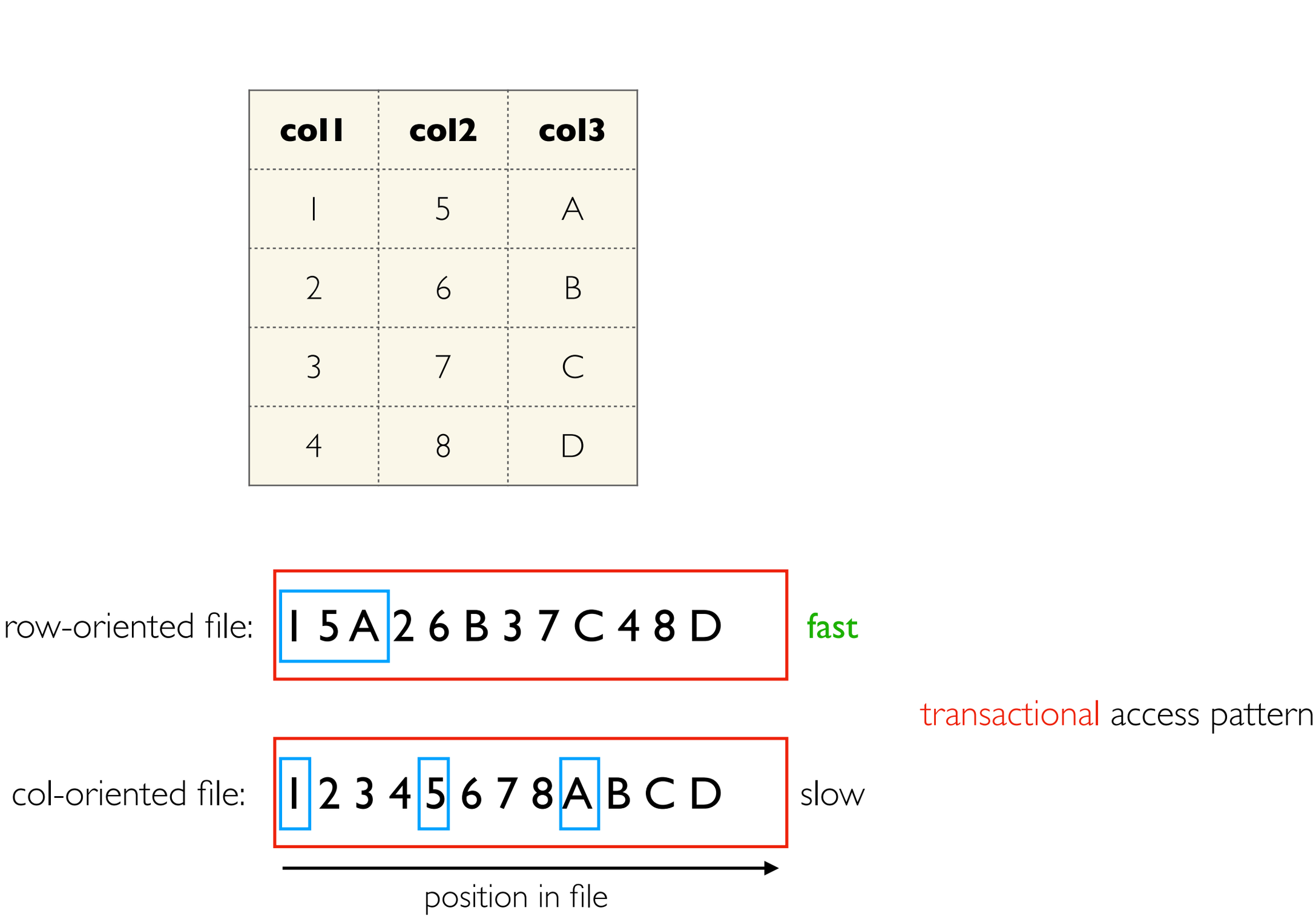
row-oriented file: 1 5 A 2 6 B 3 7 C 4 8 D

col-oriented file: 1 2 3 4 5 6 7 8 A B C D

position in file

Row-Oriented vs. Column-Oriented Layout

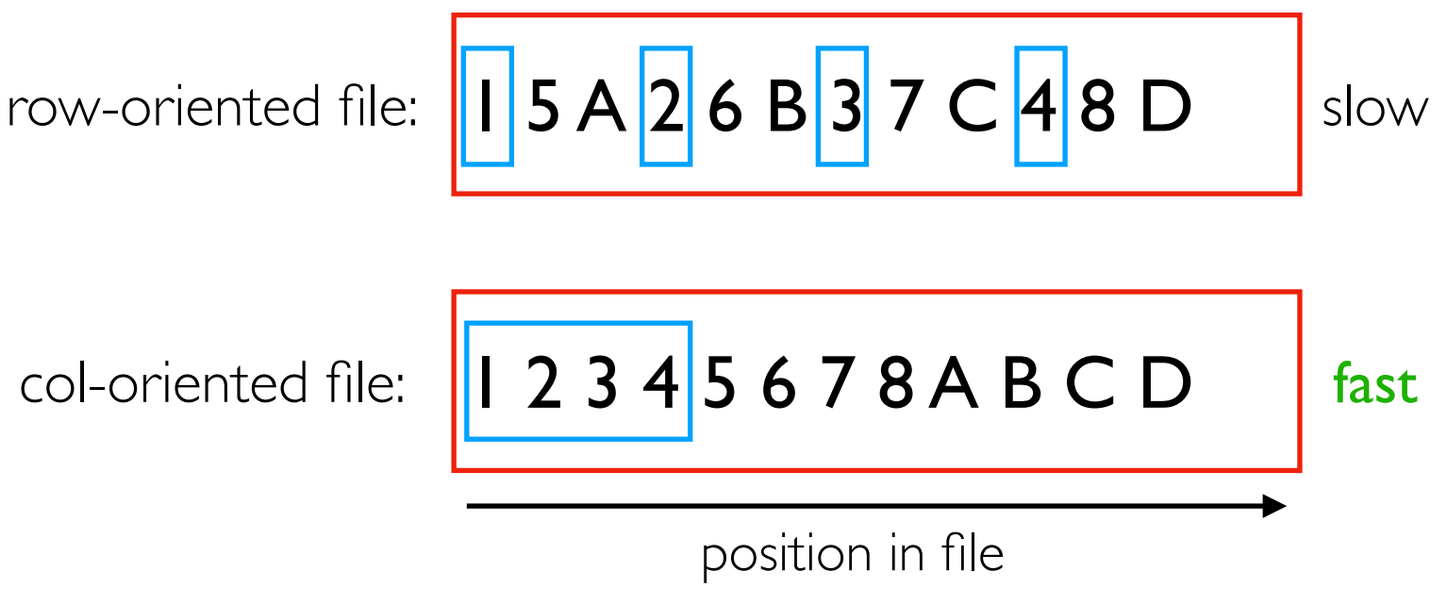
col1	col2	col3
1	5	A
2	6	B
3	7	C
4	8	D



transactional access pattern

Row-Oriented vs. Column-Oriented Layout

col1	col2	col3
1	5	A
2	6	B
3	7	C
4	8	D



analytics access pattern

Row-Oriented vs. Column-Oriented Layout

col1	col2	col3
1	5	A
2	6	B
3	7	C
4	8	D

row-oriented file: 1 5 A 2 6 B 3 7 C 4 8 D CSV

col-oriented file: 1 2 3 4 5 6 7 8 A B C D Parquet

position in file

Outline

File Systems

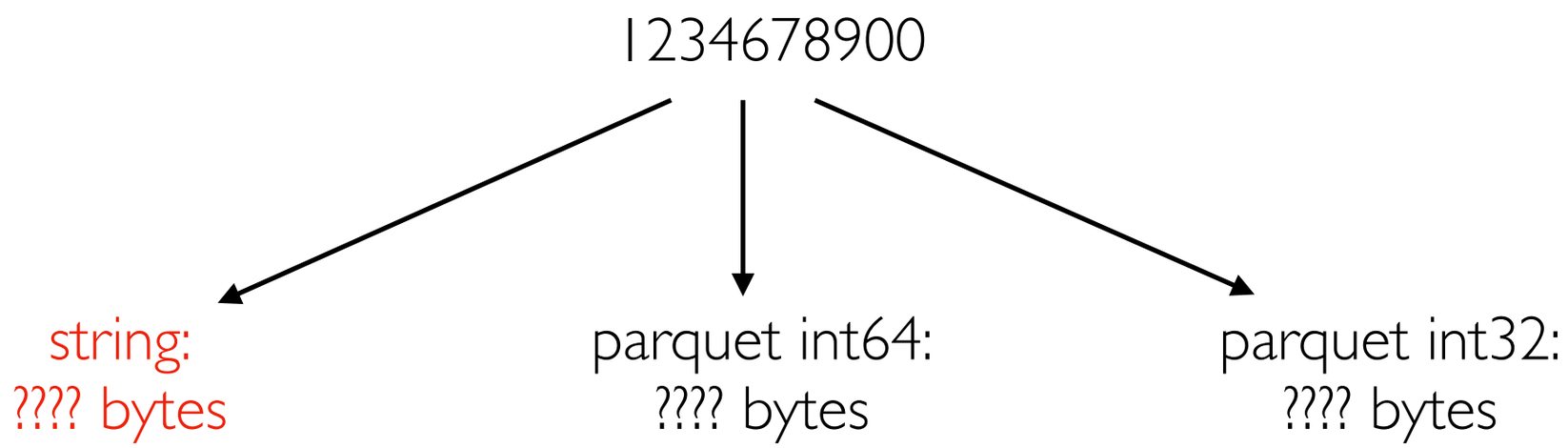
FS Demos

File Formats

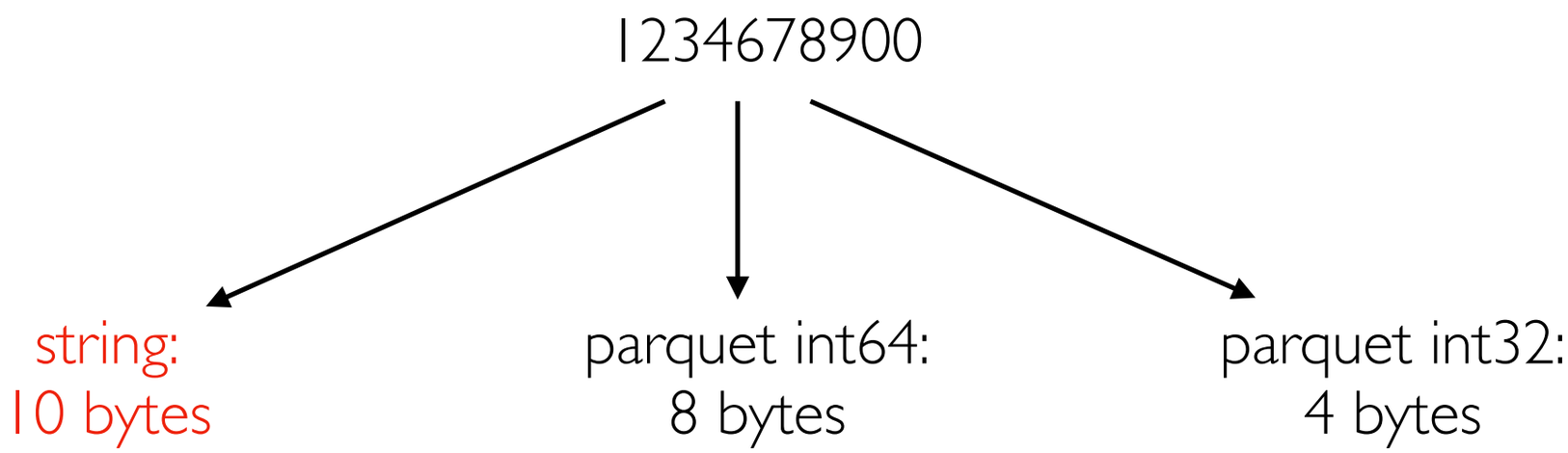
	CSV	Parquet
orientation	row	column
encoding	text	binary
compression	none	snappy
schemas	inferred	explicit

Format Demos

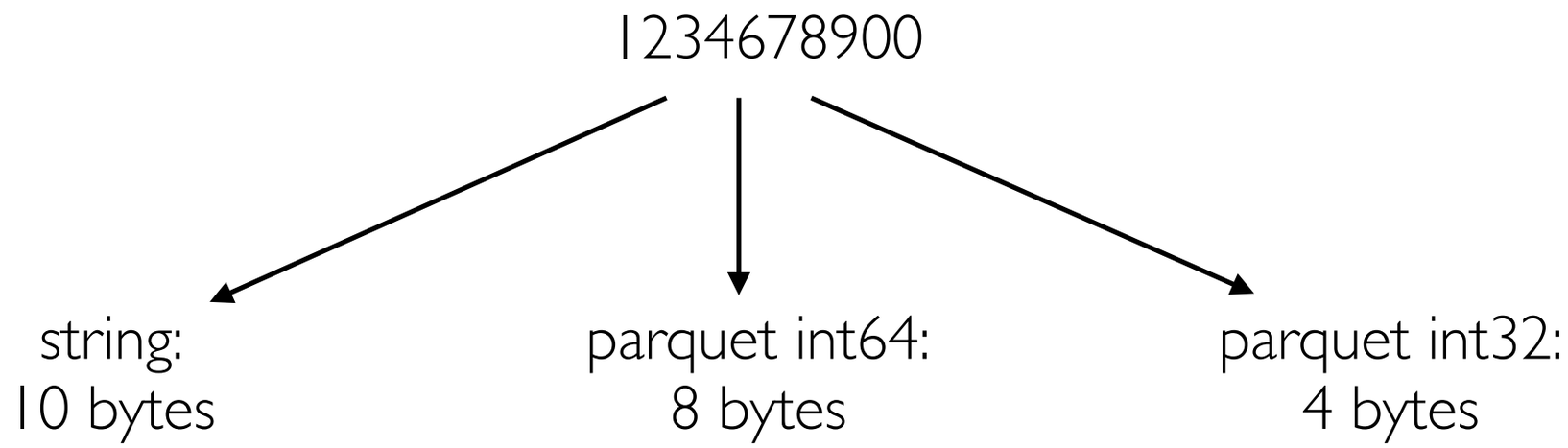
Text vs. Binary

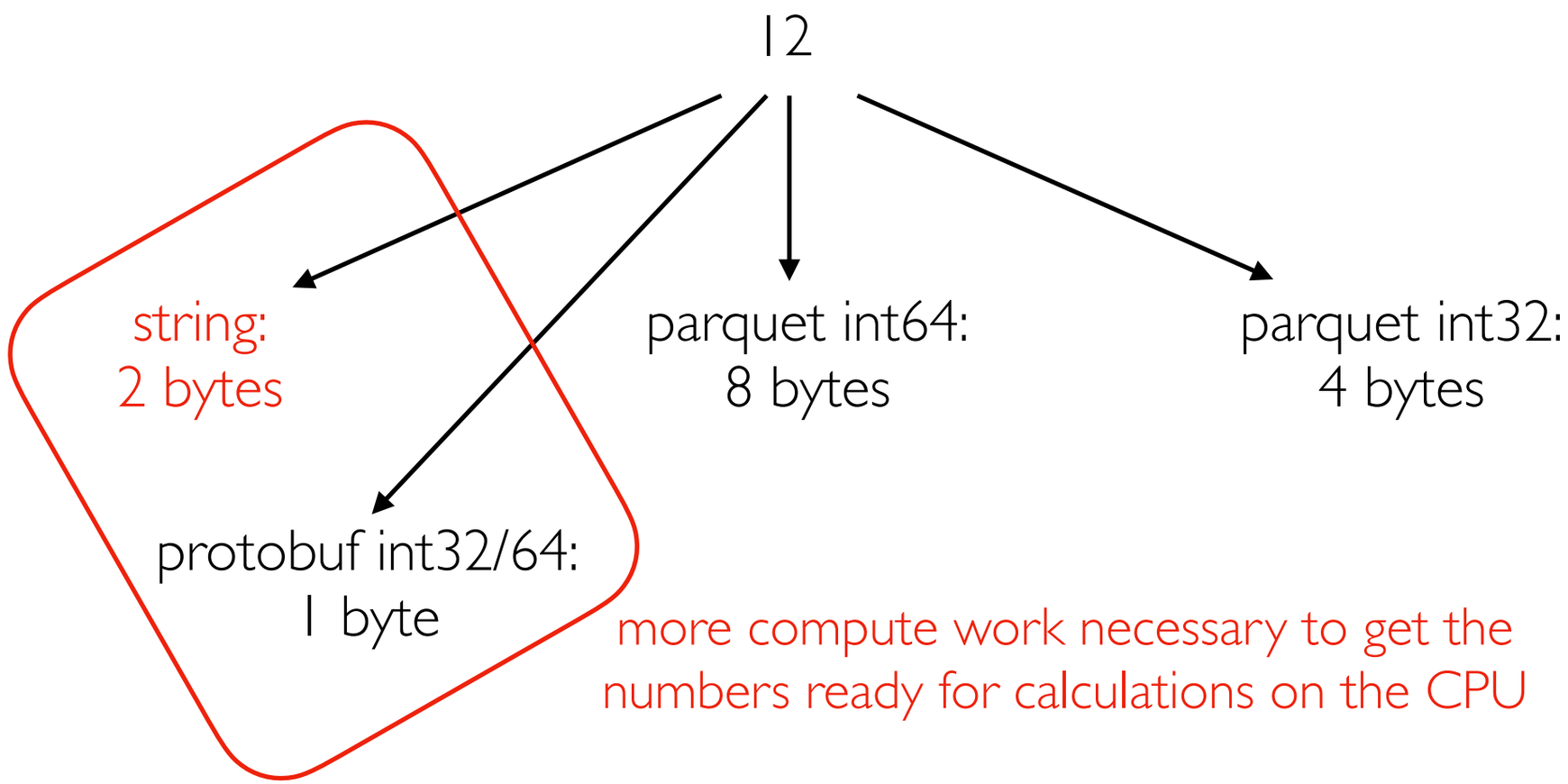


Text vs. Binary



Text vs. Binary





Outline

File Systems

FS Demos

File Formats

	CSV	Parquet
orientation	row	column
encoding	text	binary
compression	none	snappy
schemas	inferred	explicit

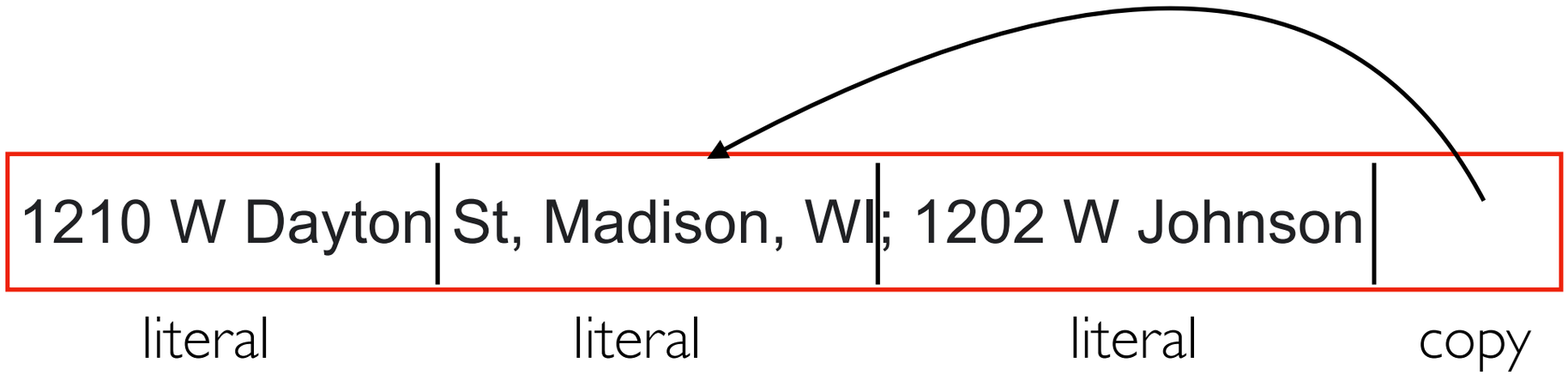
Format Demos

Compression

Idea: avoid repeating yourself

- repetitive datasets are more compressible
- more compute time finding repetition => better compression ratio (original/compressed size)

Example: snappy compression (parquet default):

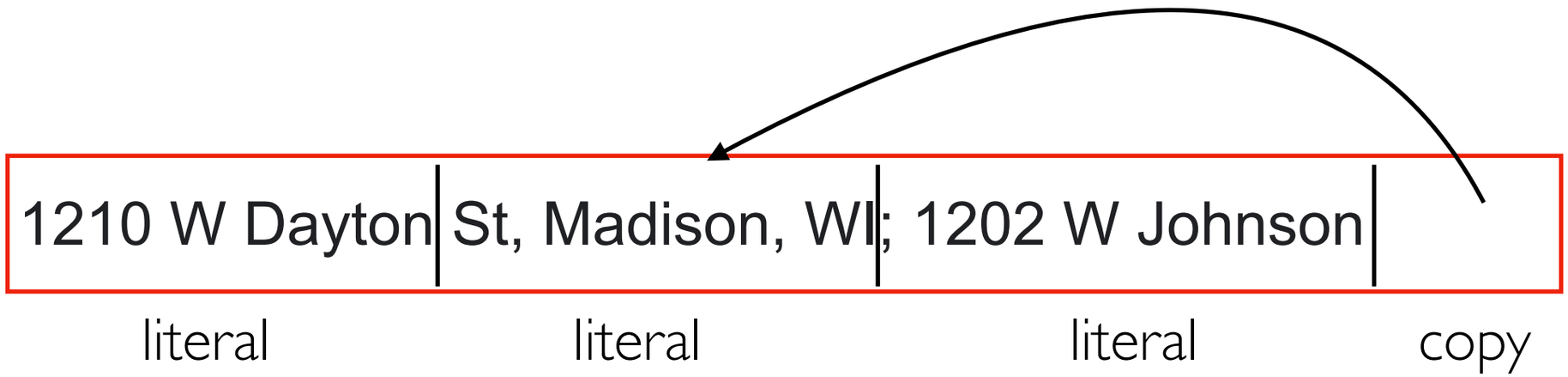


"[Snappy] does not aim for maximum compression, or compatibility with any other compression library; instead, it aims for very high speeds and reasonable compression."

Snappy documentation

- https://github.com/google/snappy
- https://github.com/google/snappy/blob/main/format_description.txt

Challenge: Small Updates



can't just update this first address in isolation
(need to rewrite other parts of the file)

Compression Window/Block

"the current Snappy compressor works in 32 kB blocks and does not do matching across blocks"

row-oriented file: 1 5 A 2 6 B 3 7 A 4 8 B

col-oriented file: 1 2 3 4 5 6 7 8 A B A B

position in file

will compression generally work better for row-oriented formats or column-oriented formats?

Outline

File Systems

FS Demos

File Formats

	CSV	Parquet
orientation	row	column
encoding	text	binary
compression	none	snappy
schemas	inferred	explicit

Format Demos

Databases

Schemas

Schema: "A description of the structure of some data, including its fields and datatypes." -- Kleppmann

CSVs:

schema specified as a dict

- in the file, everything is text
- pd.read_csv("file.csv", dtype={"col1": str, "col2": int, ...}) # specify schema (annoying)
- pd.read_csv("file.csv", dtype=None) # infer schema (slow, error prone!)

parquet files:

- type specification is part of the file
- fast: no need for very slow schema inference

Outline

File Systems

FS Demos

File Formats

Format Demos