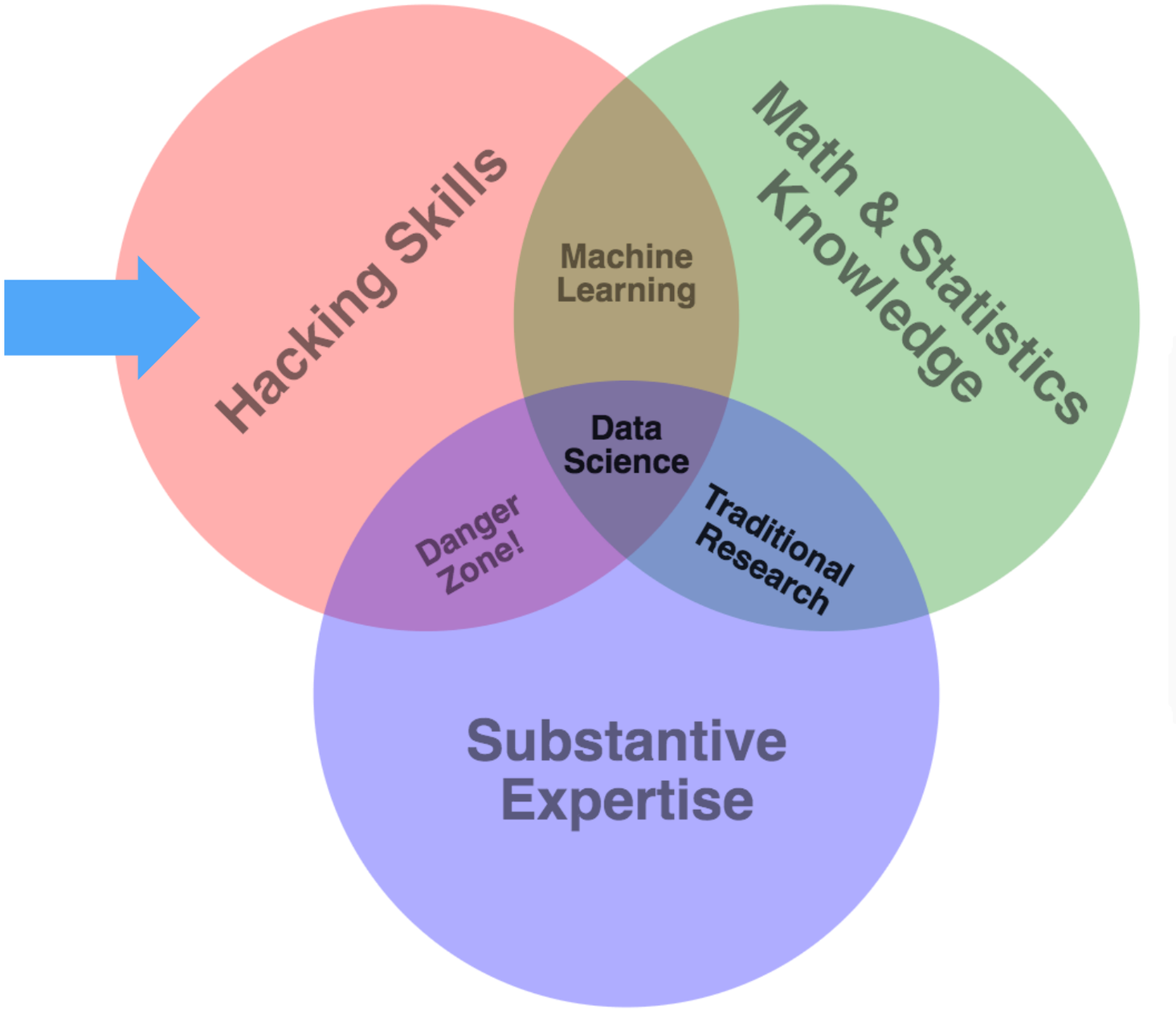
Lecture 4: Data Processing

COSC 480 Data Science, Spring 2017

Michael Hay

we are still here



Credit: https://s3.amazonaws.com/aws.drewconway.com/viz/venn_diagram/data_science.html

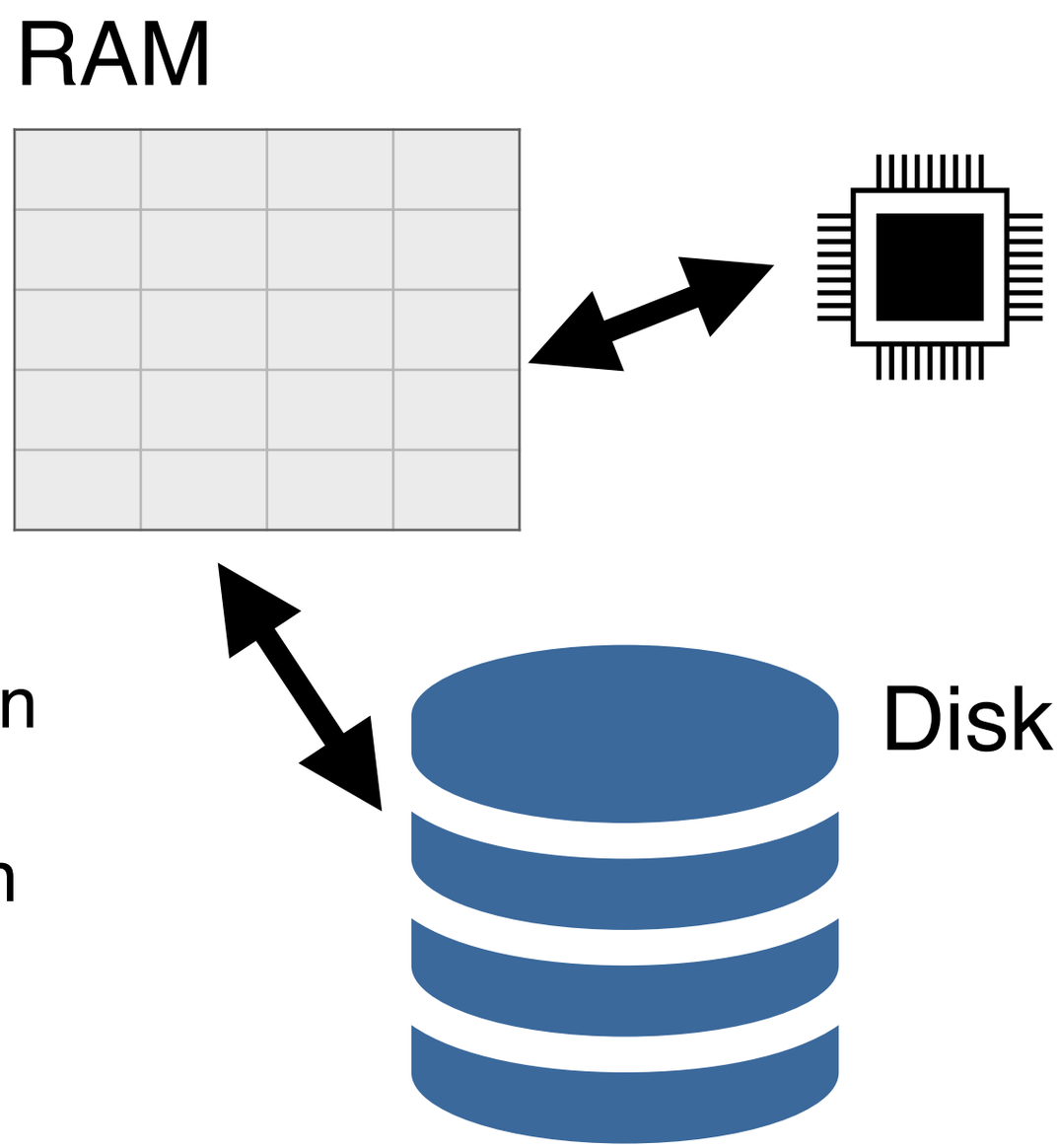
Goals for today

- Introduce some practical tools: unix utilities for working with large text files (e.g., a large CSV)
    - Useful for working with data directly, or for prepping data for storage in database
- Start to explore some of the practical/conceptual challenges that arise when working with "big" data
    - Specifically: "out of core" algorithms, rendezvous
- Motivate the desire for data manipulation tools (databases, python pandas)

Model of computation

(simplified Von Neuman Architecture)

1. Read something
2. Do something
3. Write something
4. Go to 1



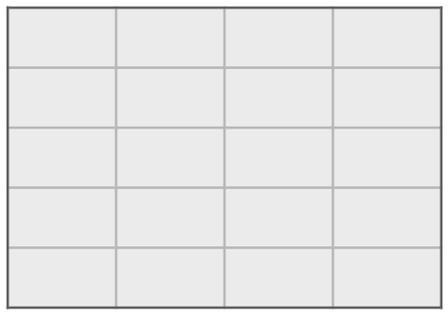
- Data moves between disk, RAM, CPU
- What happens when data is "big"?

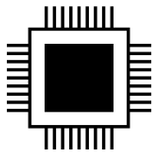
processor by Prasad from the Noun Project

Degrees of Big

	M	L	XL	even bigger
Meaning	fits in RAM	fits on disk	multiple disks	storage cost exceeds budget
Approach	usual	out-of-core algorithms	parallel data	data streams
Practical	use whatever you want	use DB! (often requires some data prep)	use distributed DB!	clever approximation algorithms
	Python pandas, SQL database	SQL: postgresql, … NoSQL: mongodb, …	Hadoop ecosystem (includes map-reduce)	(later in semester: sampling, bloom filters, etc.)

Out-of-core

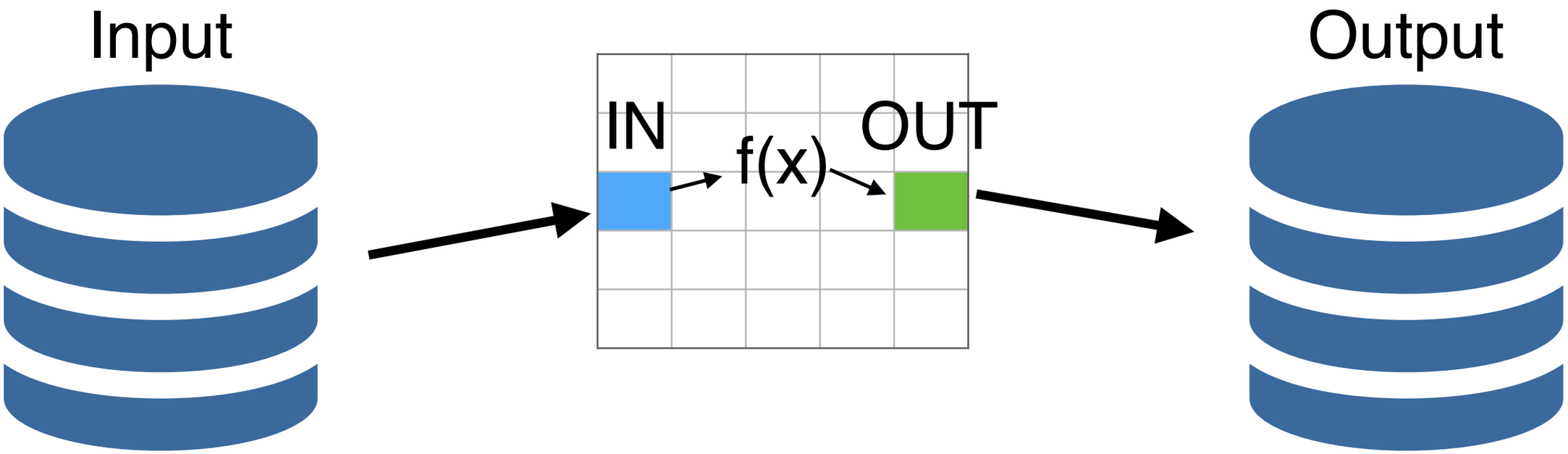




- RAM smaller than data.
- Data resides "out of core" memory
- Work on things in *batches*
  - Batch small enough to fit in RAM

Streaming through RAM

- Simple case: map
  - Goal: Compute f(x) for each record, write out the result
  - Approach: read in a chunk from INPUT into IN *buffer*, do f(x) using available memory, write f(x) to OUT *buffer*
  - When IN buffer is empty, read another chunk; when OUT buffer is full, write to OUTPUT
- Reads and writes are *not* coordinated
  - Example: if *f(x)* is compress, then read many more than write
  - Example: if *f(x)* is uncompress, then write many more than read



Example

- Given a "large" text file containing "The Adventures of Sherlock Holmes," let's find & print proper nouns.

- Working definition of a proper noun: *a word that starts with capital letter*.

- Example: map1.py

- Ways to improve our proper noun detector?

Stream processing with UNIX utilities and pipes

- Several UNIX utilities for text processing — very useful in data cleaning/ wrangling!

- Text is treated as a stream of lines

- Streams: contents of a file, STDIN, STDOUT

- Connect with pipes |

    - OS will do chunking for you

- Example: "given csv file of student grades, find names of students who got at least one 100 and no zeros on assignments."

```
$ cat students.csv | grep ',100' | grep -v ',0' | cut -f 2 -d ','
```

Handy UNIX streaming utilities

- cat, head, tail, tee, cut, grep, split, sed, awk
- Handy tip: `tail -n+2 students.csv` skips csv header
- You can also mix unix utilities with python
  - Example map2.py

```
$ head small.txt | python map2.py | grep '^[A-Z][a-z]\+$'
```

Rendezvous

- Streaming: one chunk at a time. Easy.
- But… some algorithm need certain items to be co-resident in memory
  - Not guaranteed to appear in same input chunk
  - There may be many groups of such items
  - Example: building index of word occurrences
    ("Watson" appears on lines 1, 4, 10, …
    "Sherlock" appears on lines 1, 2, 6, 10, …)
- ***Time-space rendezvous***: data items meet in same place (RAM) at same time

Achieving rendezvous

- Two common building blocks: sorting and hashing
- **Sorting**
  - Choose sort order so that items that need to be co-resident are near each other in sort order
  - Out-of-core algorithm for sorting? (*Up next!*)
- **Hashing**
  - Choose hash function so that items that need to be co-resident hash to same key
  - Out-of-core hashing? Recursive hashing (*Details omitted. Take cosc460!*)
  - **Aside**: if *set of keys is small* and only need to maintain *aggregation* of data items, then you may not need rendezvous but *accumulate* result in RAM.

Key insight: complicated tasks can often be broken down into steps: stream, rendezvous (sort/hash), stream, rendezvous (sort/hash), …

Out-of-core sorting

- External merge sort (shown on the board)

Out-of-core sorting

- External merge sort (shown on the board)
- How do we merge chunks when each file is as large (or larger) than the available RAM?
- See merge.py
- Note: merge.py is not entirely correct. What's wrong? (Play around with it?)

Exercise

Instructions: ~1 minute to think/ answer on your own; then discuss with neighbors; then I will call on one of you

- A *round* of merging corresponds to taking the current collection of chunk files and merging adjacent pairs.
- If we start with k chunk files, how many rounds of merging until we are done?
- If the input file has N units, and our RAM can hold B units, how many rounds of merging are required?
- Challenge: Can you think of any way to improve the algorithm and *lower* the number of rounds?

Handy UNIX r'vous utilities

- Non-streaming: **sort**, wc, tsort
- Stream rendezvous: uniq, join, paste, …
  - these require sorted input
- More information:
  - http://en.wikipedia.org/wiki/List_of_Unix_utilities
  - Sort on Category, search for "Text Processing"

Exercise

Instructions: ~2 minutes to think/ answer on your own; then discuss with neighbors; then I will call on one of you

- Problem: Given a large text file, produce a list of the top 100 most frequent words in descending order

- How might you solve this problem?
  - Give a high level description of your approach.
  - Use sorting to achieve rendezvous.

Exercise

Instructions: ~2 minutes to think/ answer on your own; then discuss with neighbors; then I will call on one of you

- Problem: Given the logs of an Internet Service Provider (ISP) of the form

```
ip,url,datetime,...
127.0.0.1, 'https://www.nytimes.com/2016/06/26/...', 2016-11-30 8:00,...
127.0.0.1, 'http://www.espn.com/nfl/team/_/name/ne/', 2016-11-30 8:12,...
127.0.0.3, 'http://www.espn.com/nfl/index?tab=2', 2016-11-30 8:14,...
127.0.0.3, 'http://www.cnn.com/election/results', 2016-11-30 8:14,...
127.0.0.3, 'http://www.cnn.com/main', 2016-11-30 8:15,...
127.0.0.1, 'https://www.nytimes.com/section/science', 2016-11-30 8:19,...
127.0.0.1, 'https://xkcd.com/1792/', 2016-11-30 8:29,...
...
```

- Find sites (e.g., nytimes.com) that have at least *k* pages each getting at least *m* visits. Use sorting to r'vous.

- Challenge: how would your answer change if it was *m* distinct *visitors* (determined by ip address)?