CSE 350

Advanced Data Structures

Lecture 1: Intro

Main Website/Syllabus:
https://odin.cse.buffalo.edu/teaching/cse-350/

Piazza:
https://piazza.com/buffalo/fall2025/cse350

Autolab:
https://autolab.cse.buffalo.edu

Me:

Oliver Kennedy
okennedy@buffalo.edu
Capen 212

Research
- Managing messy data (spreadsheets)
- Data go vroom (compilers, data structures)

Other
- Western Martial Arts
- Bikes

What is Advanced Data Structures?

```
data = malloc(N * sizeof(int))

// do stuff with data //

int count = 0
for i in 0 .. N:
  count += data[i]
```

O(1) + O(N) = O(N)

O(1)

O(1)] O(1)] N Σ O(1) = O(N)

```
int count = 0
for i in 0 .. N:
  count += data[rand() % N]
```

O(1)

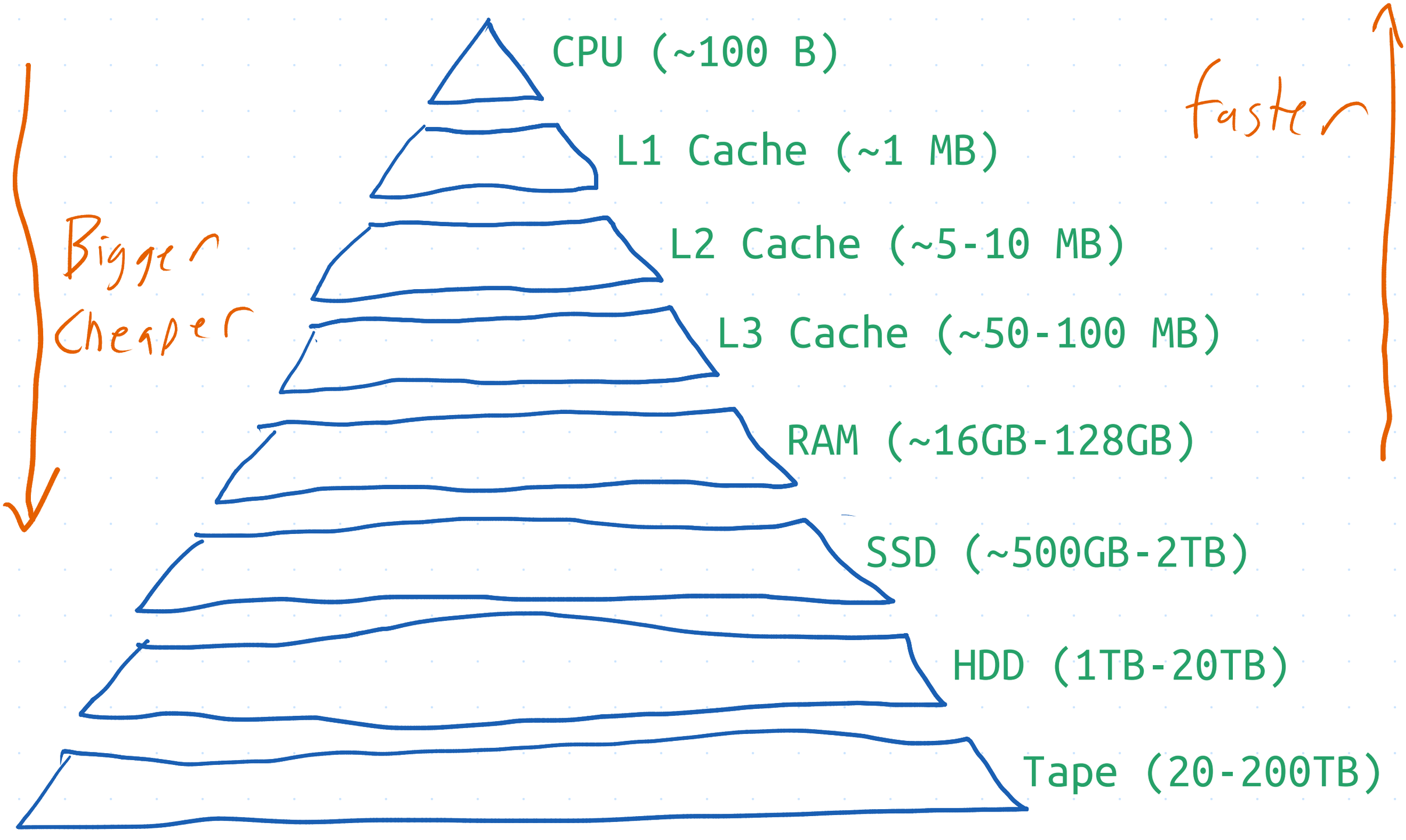
O(1)] O(1)] N Σ O(1) = O(N)

O(1) + O(N) = O(N)

Sequential access

Random access

Still O(1), But much much worse


CPU (~100 B)
L1 Cache (~1 MB)
L2 Cache (~5-10 MB)
L3 Cache (~50-100 MB)
RAM (~16GB-128GB)
SSD (~500GB-2TB)
HDD (1TB-20TB)
Tape (20-200TB)
Bigger
Cheaper
faster

RAM Model

- Only one "memory"
- O(1) to access it
- No explicit data movement

EM Model

- Two or more levels of memory
- O(1) to access the upper level
- Need to explicitly move data

How good is an algorithm?

- Runtime Cost
- *Memory Bounds*
- *IO Cost*

CSE 250

New for 350

Databases

- The relational model (tidy data)
- SQL
- Indexing

Databases abstract the messiness of the EM model, and let you focus on what you want.

Code

- Projects provided in C
- Must compile with gcc -std=gnu17
- Any libc header on autolab is fair game

Projects (~bi-weekly)

- AI Quiz (Sep 1)
- Warmup: Page-based File IO (Sep 8)
- Buffer Manager
- Buffered Linked List
- ISAM Index
- B+Tree
- On-Disk Joins

Midterm

Where: In Class

When: Oct 21

Final Exam

Where: See Hub

When: See Hub

Do...

- work together to brainstorm ideas
- explain concepts to each other
- include a list of collaborators

Do not...

- collaborate to write code
- describe the details of solutions
- leave your code accessible to others
- use LLMs for code
- assume that LLMs are truthful

If in doubt, ask me

Why?

- We learn by failing.
- You wouldn't use a forklift at the gym.
- AI violations earn an F.

You are liable if someone else submits your work as their own.

If you contact me before I discover an AI violation, you may withdraw your submission

250 Recap

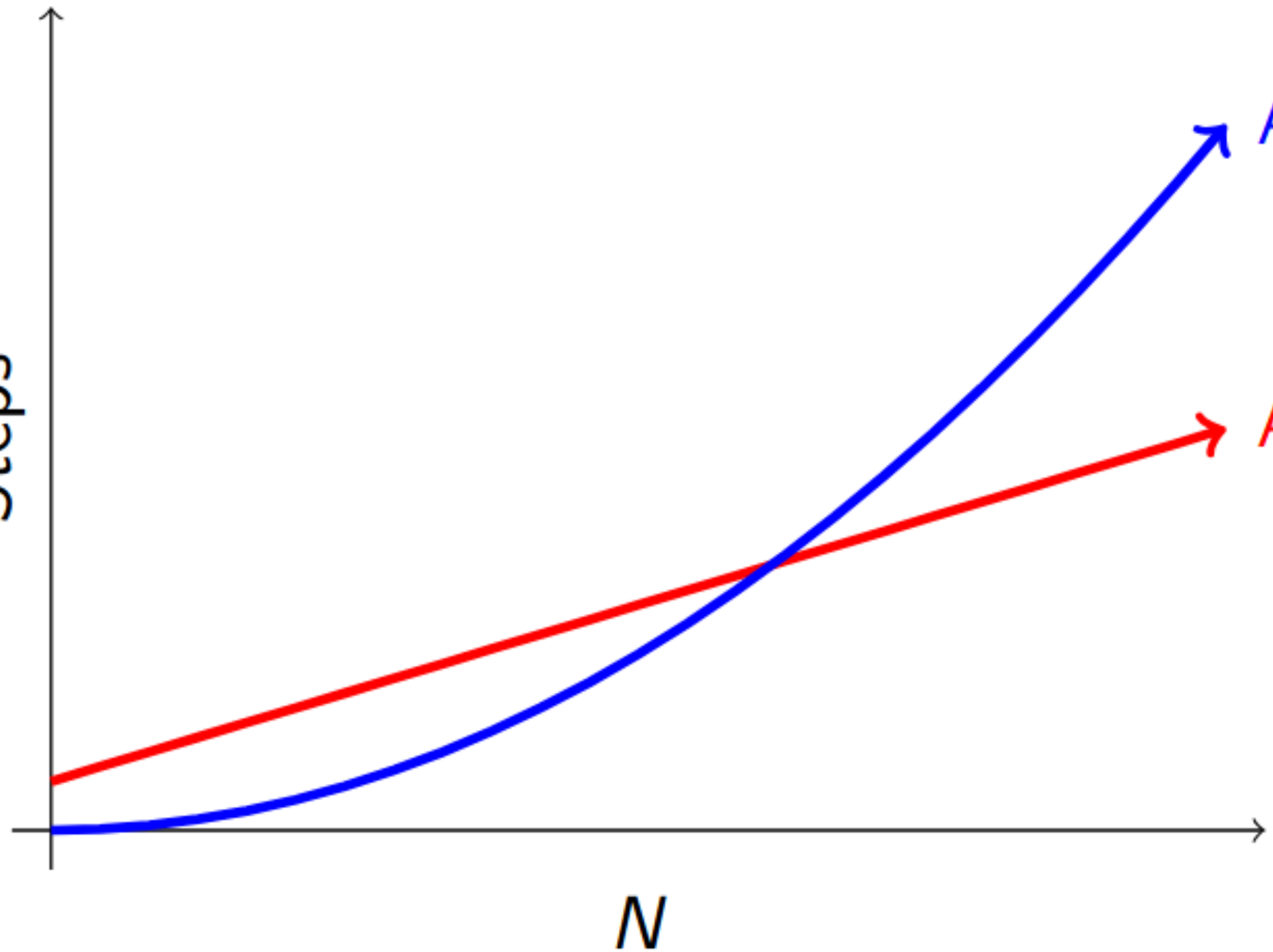
Big-O

Algorithm 1 takes 100N steps] Better for N>20

Algorithm 2 takes $5N^2$ steps] Better for N<20

(N is the size of the input)

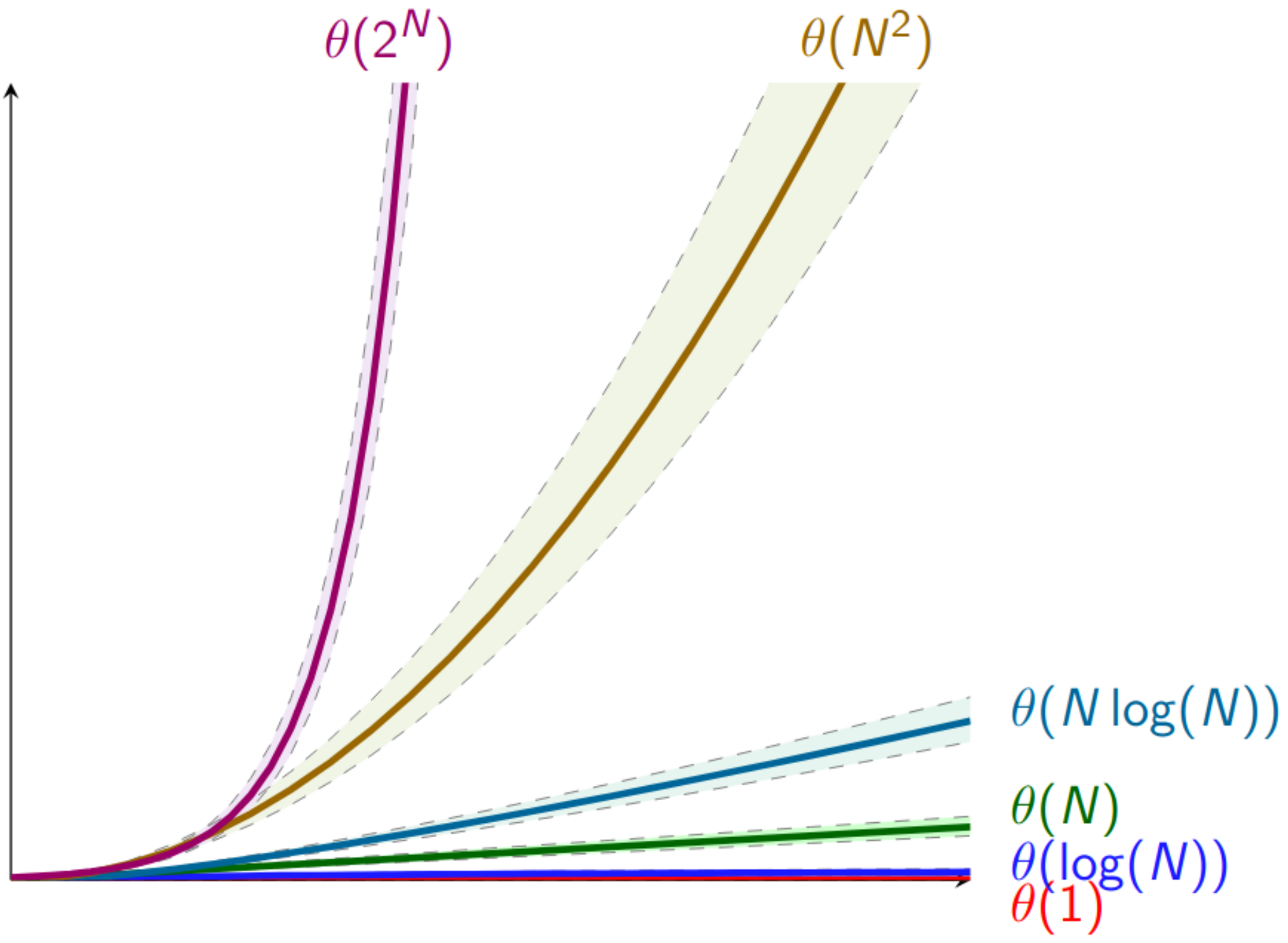
Which is better?


Steps
Alg. 2
Alg. 1
N

O(1) - Constant
O(log(N)) - Logarithmic
O(N) - Linear
O(N log(N)) - Loglinear
O(N^2) - Quadratic

...

O(2^N) - Exponential


$\theta(2^N)$
$\theta(N^2)$
$\theta(N \log(N))$
$\theta(N)$
$\theta(\log(N))$
$\theta(1)$

- $f(N)$ is a Worst-Case Bound $(T(N) \in O(f(N)))$

  The algorithm **always** runs in at most $c \cdot f(N)$ steps.

- $f(N)$ is an Amortized Worst-Case Bound

  N **invocations** of the algorithm **always** run in at most $N \cdot c \cdot f(N)$ steps.

- $f(N)$ is an Expected Worst-Case Bound $(E[T(N)] \in O(f(N)))$

  The algorithm is **statistically likely** to run in at most $c \cdot f(N)$ steps.

Examples

```
count = 0
for i in 0 .. N:
  for j in 0 .. N:
    count += data1[i] * data2[j]
```

$\sum_{N} O(1) = O(N)$

$\sum_{N} O(N) = O(N^2)$

```
data = array_list_init()

for i in 0 .. N:
    array_list_append(data, i)
```

O(1)

Amortized O(1)
Unqualified O(N)

$\sum_{N} O(N) = O(N^2)$

But since inner loop can use Amortized

$\sum_{N} O(1) = O(N)$

$O(N) + O(1) = \boxed{O(N)}$

O(1)

```
data = array_list_init()

for i in 0 .. N:
    array_list_append(data, i)
    count = 0
    for j in 0 .. i:
        count += array_list_get(data, j)
```

Am O(1)
Unq O(N)

O(1)

O(1)

$\sum_{i} O(1) = O(i)$

Am: $O(1) + O(1) + O(i) = O(i)$; $i < N$

Unq: $O(N) + O(1) + O(i) = O(N + i) = O(N)$

Can use Amortized due to loop: $\sum_{i}^{N} O(i) = O(N^2)$

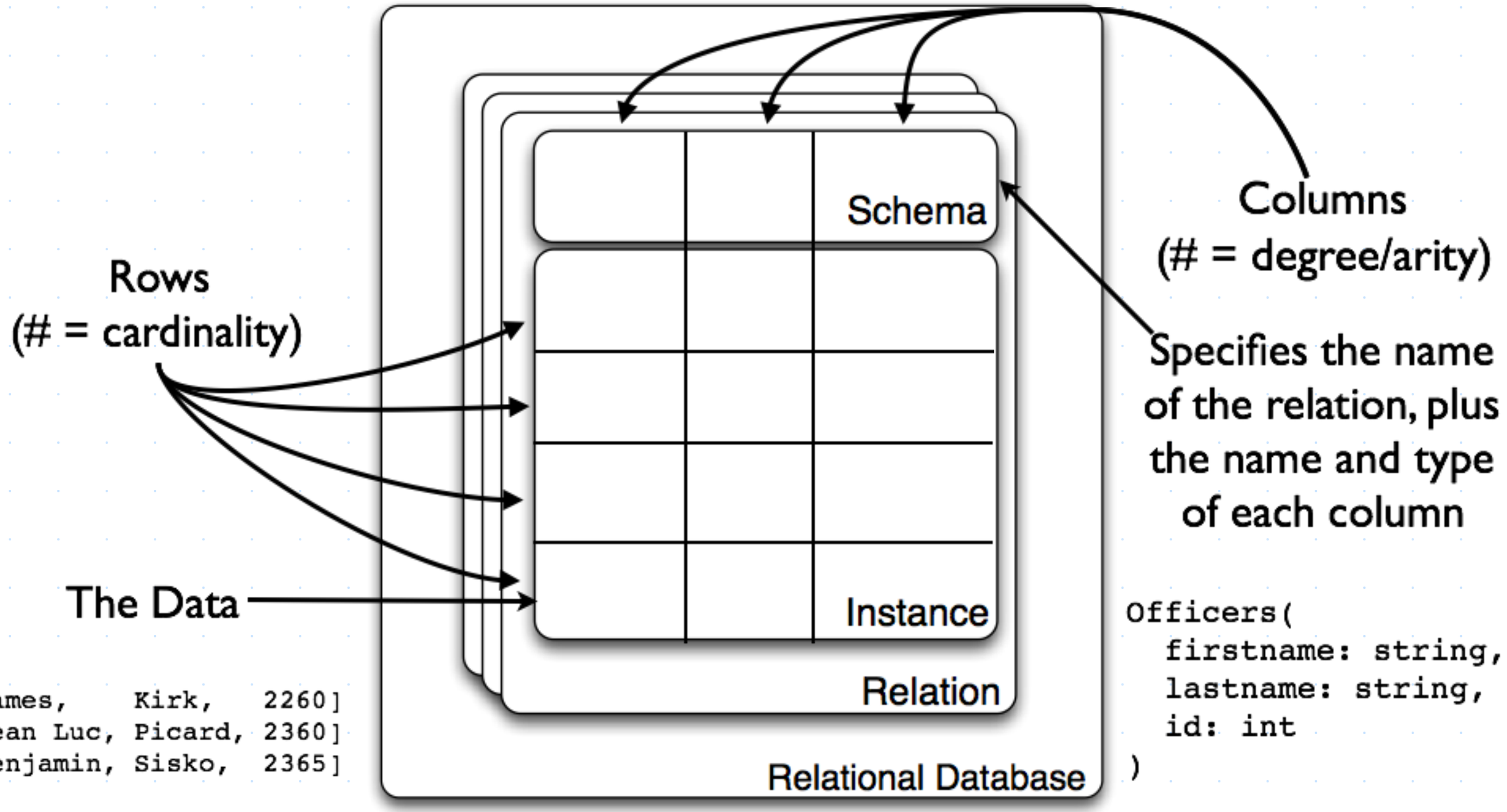
Collections

Collection ADTs

Collection	Ordered?	Unique?	Indexed
Set	No	Yes	by Value
List	Yes	No	by Position
Map	No	Yes	by Key
Table	†	†	†

```
Set

{ A, B, C }

List

[ A, A, B, C ]
```

```
Map

{ A → 1, B → 3 }
```


Schema
Columns
(# = degree/arity)
Rows
(# = cardinality)
Specifies the name
of the relation, plus
the name and type
of each column
The Data
Instance
Relation
Relational Database
[James, Kirk, 2260]
[Jean Luc, Picard, 2360]
[Benjamin, Sisko, 2365]
Officers(
firstname: string,
lastname: string,
id: int
)

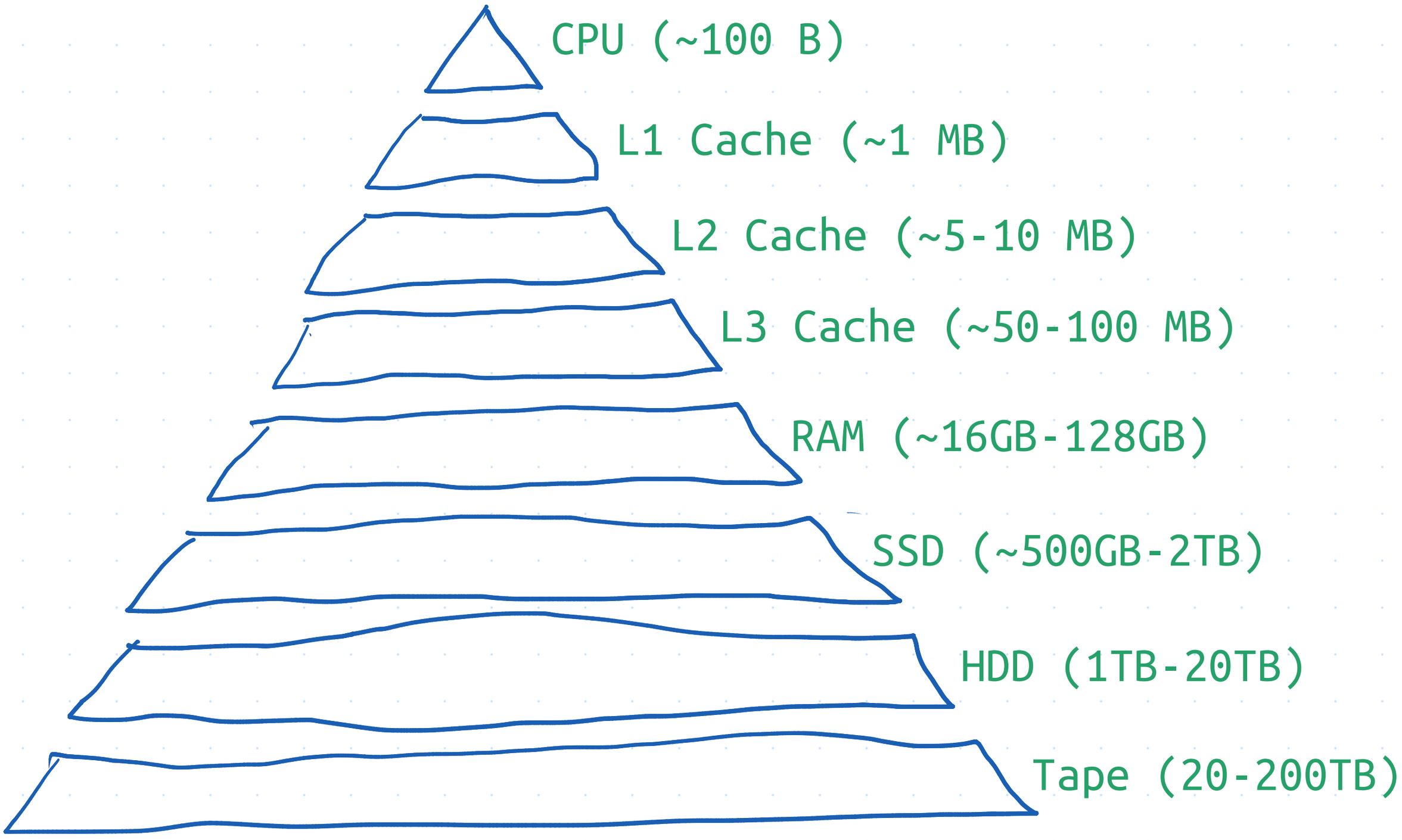
Database Layer	API	Abstracts...
Query/Optimizer (CSE 462)	SQL	Algorithm/Datastructure choices
Datastructures/Index	Cursors (Iterators)	Algorithm/Datastructure implementation
Transaction	Pages/Frames	Concurrent Access
Storage	pages/Frames	Caching / page load
Hardware	Pages	physics

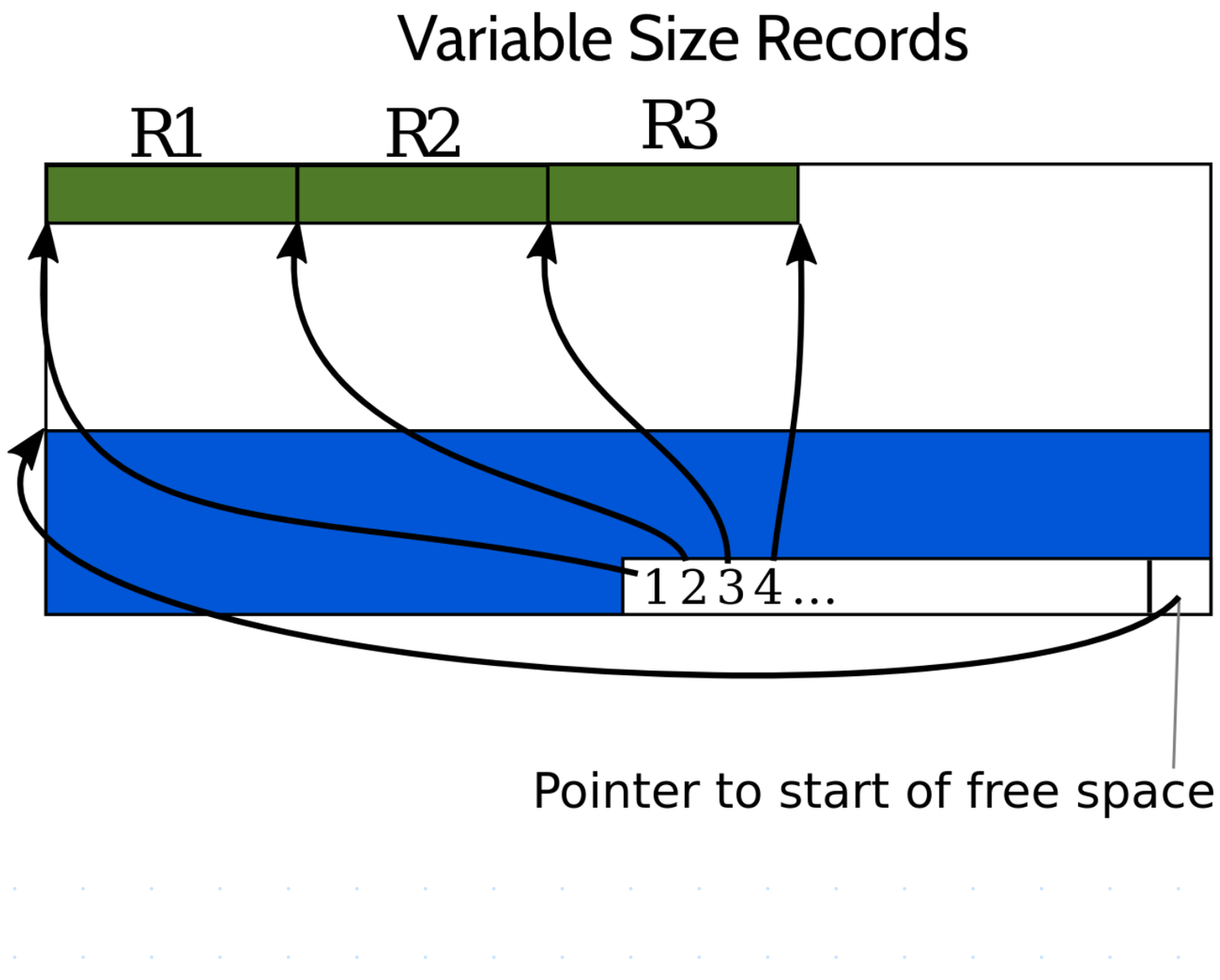
projects 3-6

projects 1,2

project 0

Also covered in lecture


CPU (~100 B)
L1 Cache (~1 MB)
L2 Cache (~5-10 MB)
L3 Cache (~50-100 MB)
RAM (~16GB-128GB)
SSD (~500GB-2TB)
HDD (1TB-20TB)
Tape (20-200TB)


Variable Size Records
R1
R2
R3
1 2 3 4 ...
Pointer to start of free space

Project 0: File API

Implement

```
Result page_read(FILE *file, PageID page_id, Frame frame);
Result page_write(FILE *file, PageID page_id, Frame frame);
```

Notes:

- Frame is a PAGE_SIZE block of data
- Result is SUCCESS or ERROR

Makefile

- make: compile
- make test: compile and run tests in tests/
- make run: compile and run

Test REPL

- 'h': Print Hello World
- 'W[page_id][data]': Write `data` to `page_id`
- 'R[page_id]': Read out `data`
- 'f': Flush data to disk