UC San Diego

# DSC 102
# Systems for Scalable Analytics

Rod Albuyeh

Midterm Review

# Admin

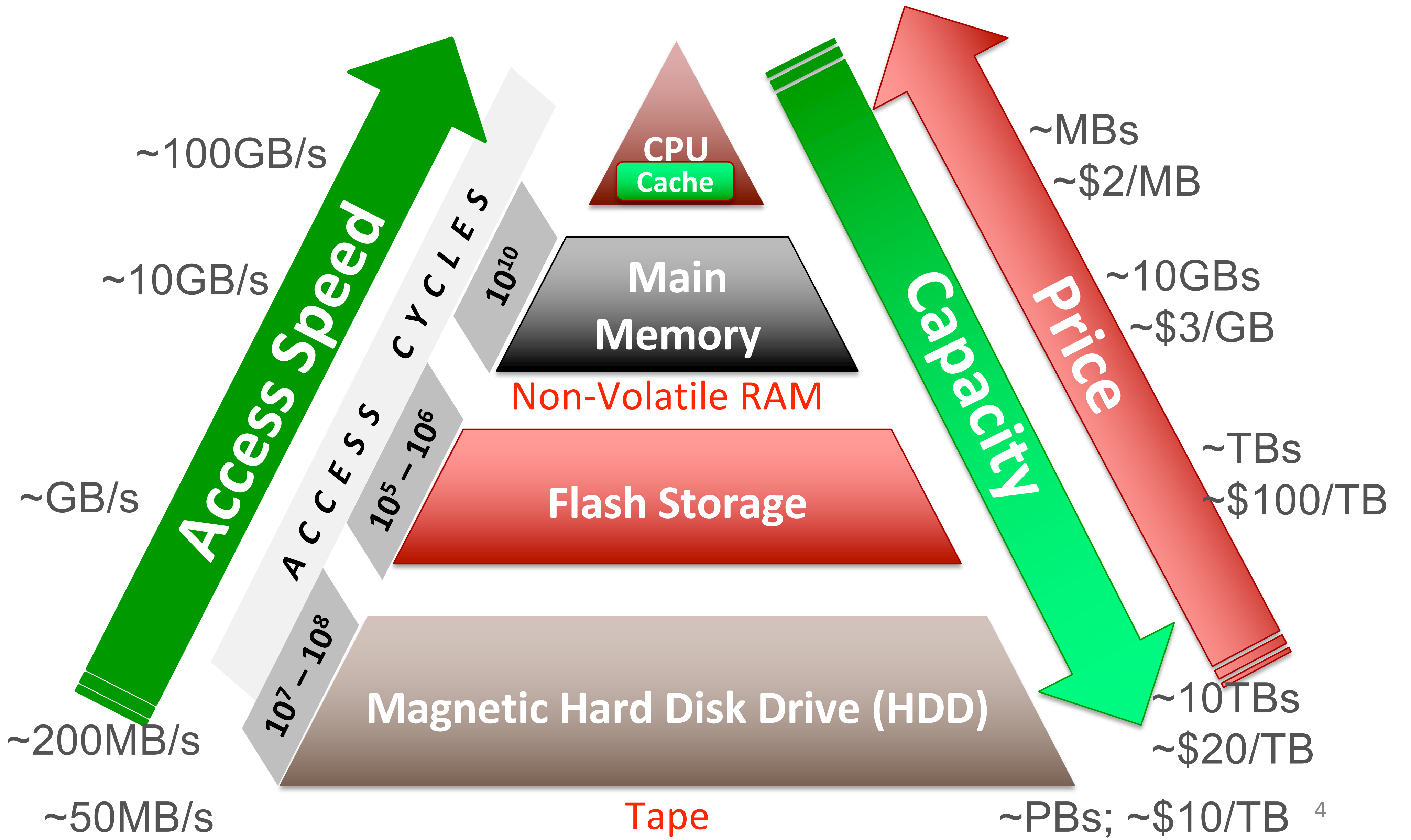
PA0 Debrief

- Average score: 7.18/8
- Five teams claimed runtime extra credit
- New runtime record: 241.73 seconds

# Midterm Overview

Similar format to sample midterms, lighter on the binary + hexadecimal questions and heavier on cloud.

# Recap: Memory Hierarchy



# Recap: Digital Representation of Data

***Q:** How many unique data items can be represented by 3 bytes?*

- Given k bits, we can represent $2^k$ unique data items
- 3 bytes = 24 bits => $2^{24}$ items, i.e., 16,777,216 items
- Common approximation: $2^{10}$ (i.e., 1024) ~ $10^3$ (i.e., 1000); kibibyte (KiB) = 1024 bytes, vs kilobyte (KB) = 1000 bytes

***Q:** How many bits are needed to distinguish 97 unique items?*

- For k unique items, invert the exponent to get $\log_2(k)$
- But #bits is an integer! So, we only need $\lceil \log_2(k) \rceil$
- So, we only need the next higher power of 2
- So... 7 bits

# Recap: Decimal <-> Binary

***Q:** How to convert from decimal to binary representation?*

1. Given decimal n

   if n is power of 2 (say, $2^k$), put 1 at bit position k; if k=0, stop; else pad with trailing 0s till position 0

   if n is not power of 2, identify the power of 2 just below n (say, $2^k$); #bits is then k; put 1 at position k
2. Reset n as n - $2^k$; return to Steps 1-2
3. Fill remaining positions in between with 0s

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Position/Exponent of 2 |
|---|---|---|---|---|---|---|---|---|---|
| Decimal | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | Power of 2 |
| $5_{10}$ | | | | | | 1 | 0 | 1 | |
| $47_{10}$ | | | 1 | 0 | 1 | 1 | 1 | 1 | |
| $163_{10}$ | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | |
| $16_{10}$ | | | | 1 | 0 | 0 | 0 | 0 | |

***Q:** Binary to decimal?*

# Recap: Hexadecimal representation

- *Hexadecimal* representation is a common stand-in for binary representation; more succinct and readable
  - Base 16 instead of base 2 cuts display length by ~4x
  - Digits are 0, 1, ... 9, A ($10_{10}$), B, … F ($15_{10}$)
  - Each hexadecimal digit represents 4 bits.

| Decimal | Binary | Hexadecimal |
|---|---|---|
| $5_{10}$ | $101_2$ | $5_{16}$ |
| $47_{10}$ | $10\ 1111_2$ | $2F_{16}$ |
| $163_{10}$ | $1010\ 0011_2$ | $A3_{16}$ |
| $16_{10}$ | $1\ 0000_2$ | $10_{16}$ |

Alternative notations

0xA3 or $A3_H$

# Hexadecimal representation continued

Let's unpack:

Base 10...

0 1 2 3 4 5 6 7 8 9

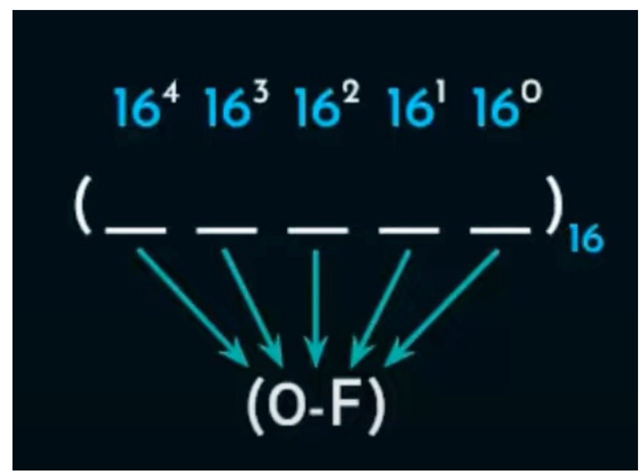
Base 2...

0 1

Base-16 Hexadecimal...

0 1 2 3 4 5 6 7 8 9 A B C D E F

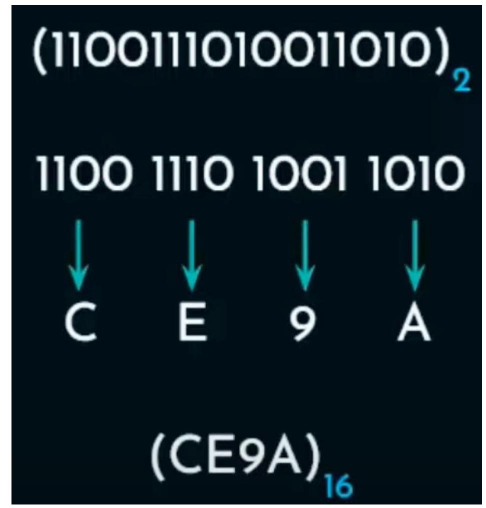
10 11 12 13 14 15



# An aside: Hexadecimal to binary relationship

**Hexadecimal Number System**

| 8 | 4 | 2 | 1 | Hex. Rep. |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | A |
| 1 | 0 | 1 | 1 | B |
| 1 | 1 | 0 | 0 | C |
| 1 | 1 | 0 | 1 | D |
| 1 | 1 | 1 | 0 | E |
| 1 | 1 | 1 | 1 | F |




or...

0xCE9A

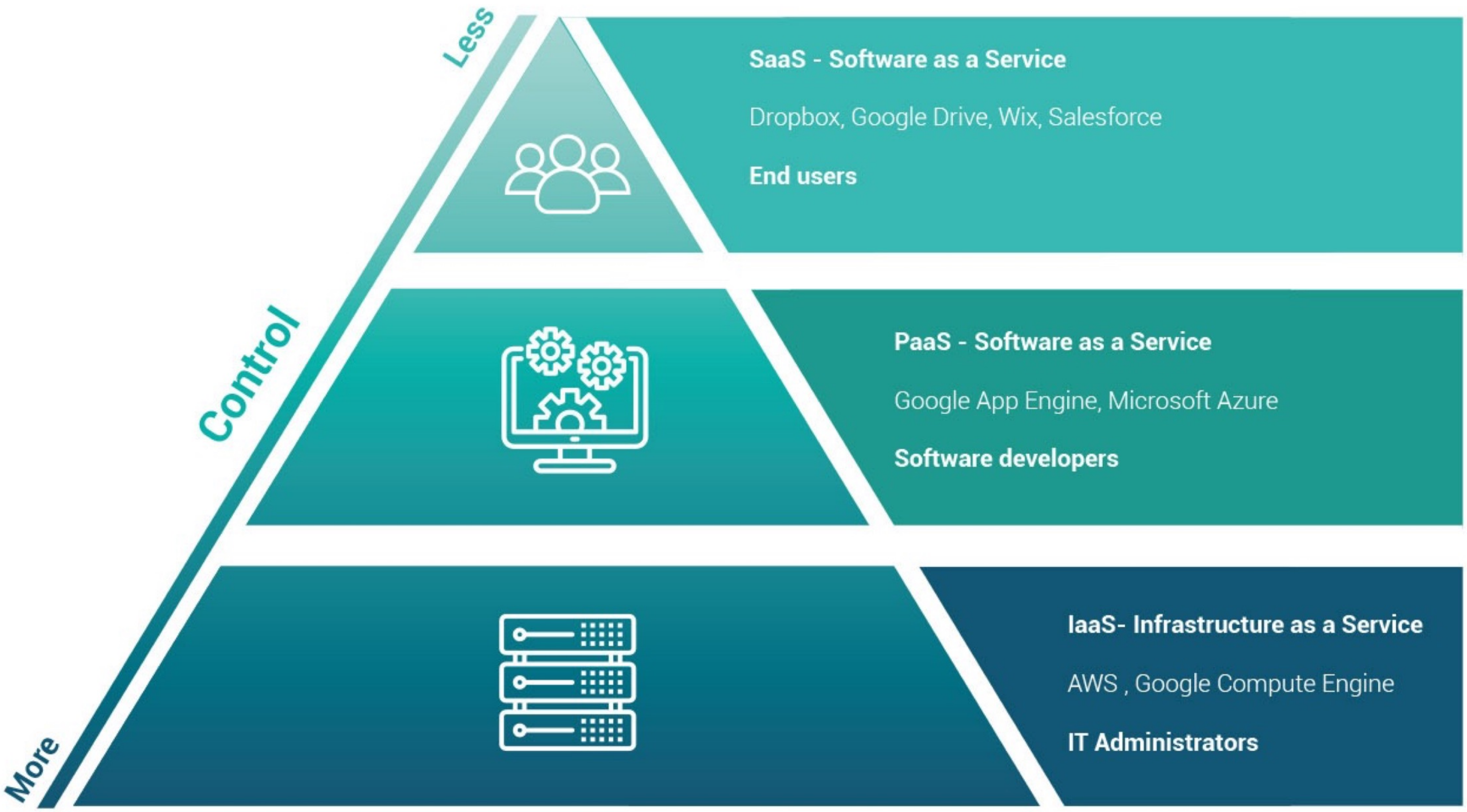
# Recap: Memory Management

- **Caching:** Buffering a copy of bytes (instructions and/or data) from a lower level at a higher level to exploit locality
- **Prefetching:** Preemptively retrieving bytes (typically data) from addresses not explicitly asked yet by program
- **Spill/Miss/Fault:** Data needed for program is not yet available at a higher level; need to get it from lower level
  - **Register Spill** (register to cache); **Cache Miss** (cache to main memory); **“Page” Fault** (main memory to disk)
- **Hit:** Data needed is already available at higher level
- **Cache Replacement Policy:** When new data needs to be loaded to higher level, which old data to evict to make room? Many policies exist with different properties

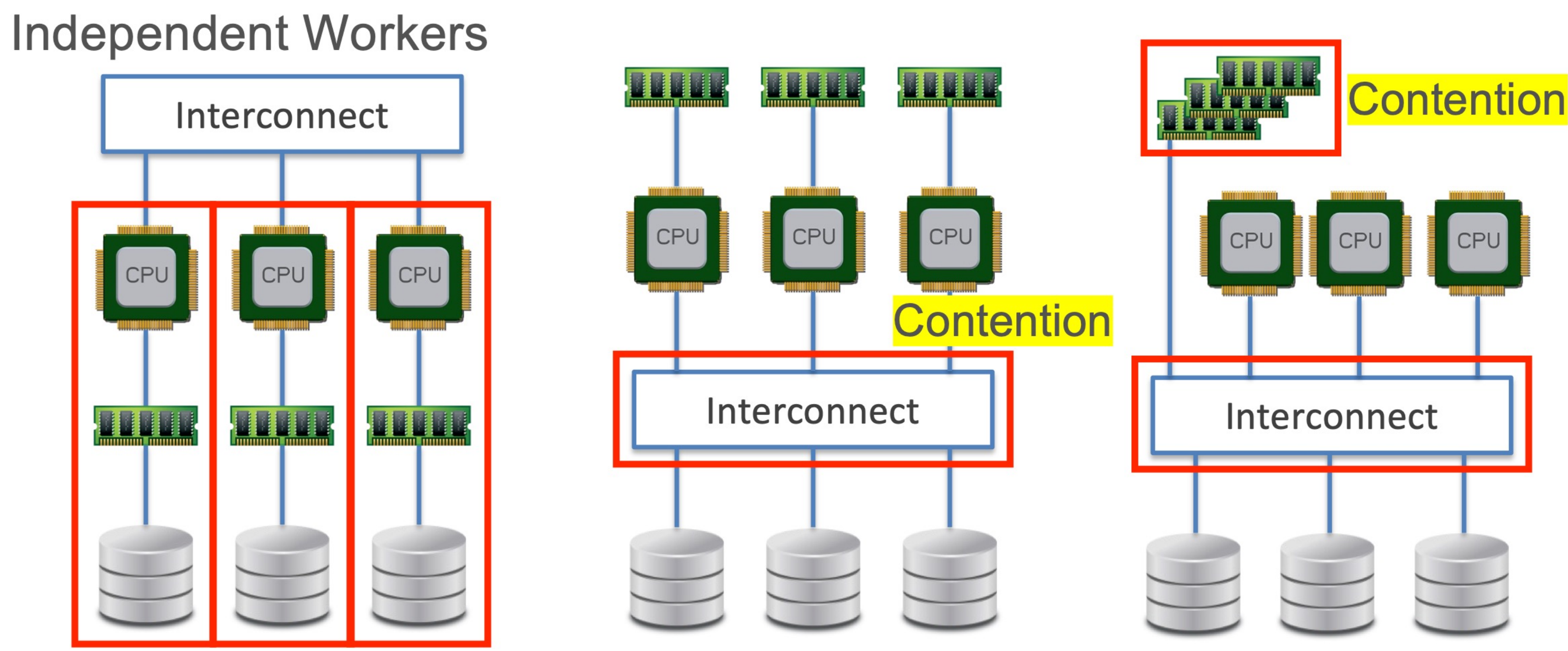
# Recap: Scheduling Policies/Algorithms

- **Schedule:** Record of what process runs on each CPU & when
- Policy controls how OS time-shares CPUs among processes
- Key terms for a process (aka **job**):
  - **Arrival Time**: Time when process gets created
  - **Job Length**: Duration of time needed for process
  - **Start Time**: Times when process first starts on processor
  - **Completion Time**: Time when process finishes/killed
  - **Response** Time = [Start Time] – [Arrival Time]
  - **Turnaround** Time = [Completion Time] – [Arrival Time]
- **Workload:** Set of processes, arrival times, and job lengths that OS Scheduler has to handle
- In general, OS may not know all Arrival Times and Job Lengths beforehand! But **preemption** is possible
- **Key Principle:** Inherent tension in scheduling between overall workload *performance* and allocation *fairness*
  - Performance metric is usually *Average Turnaround Time*

# Recap: Cloud Layers



# Recap: Parallelism Paradigms




Shared-Nothing Parallelism

Shared-Disk Parallelism

Shared-Memory Parallelism

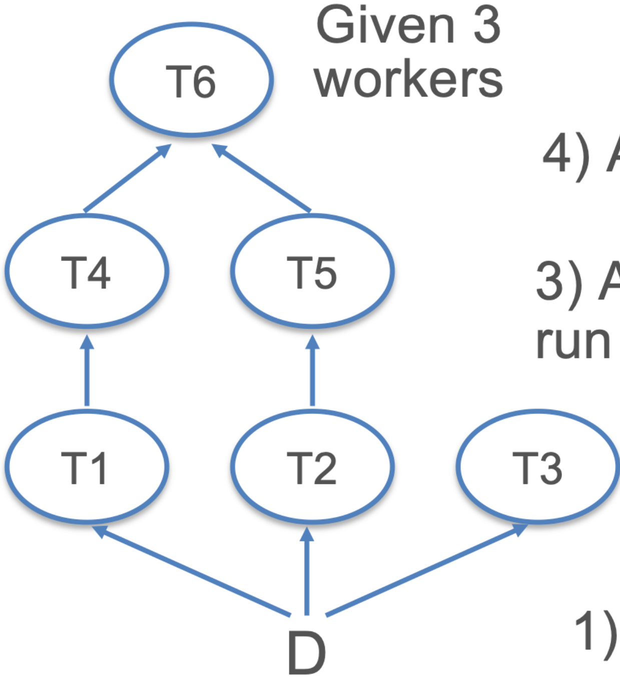
# Recap: Task Parallelism

**Basic Idea**: Split up *tasks* across workers; if there is a common dataset that they read, just make copies of it (aka *replication*)

**Example:**

*This is your PA1 setup! Except, Dask Scheduler puts tasks on workers for you.*

Given 3 workers




4) After T4 & T5 end, run T6 on W1; W2 is *idle*

3) After T1 ends, run T4 on W1; after T2 ends, run T5 on W2; after T3 ends, W3 is *idle*

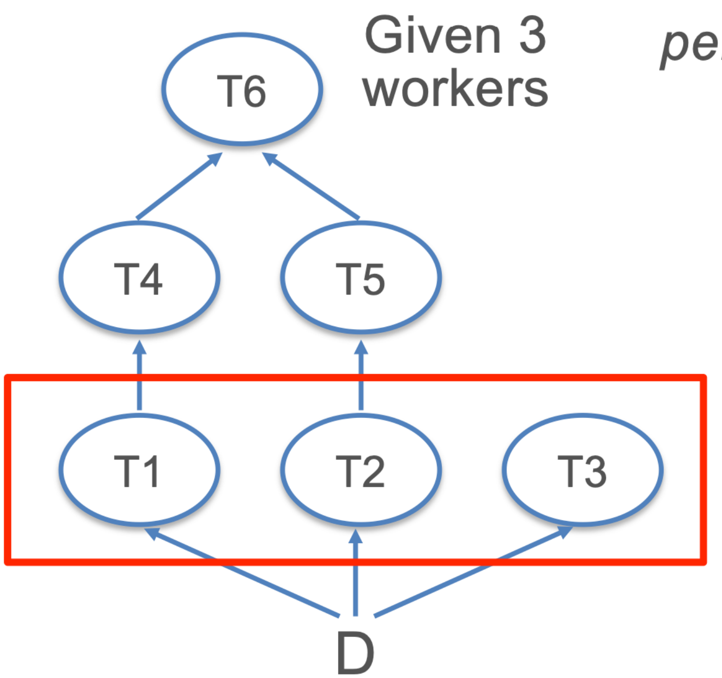
2) Put T1 on worker 1 (W1), T2 on W2, T3 on W3; run all 3 in parallel

1) Copy whole D to all workers

# Recap: Task Parallelism (continued)

- The largest amount of *concurrency* possible in the task graph, i.e., how many task can be run simultaneously

**Example:**




***Q:** How do we quantify the runtime performance benefits of task parallelism?*

But over time, degree of parallelism keeps dropping in this example

Degree of parallelism is only 3

So, more than 3 workers is not useful for this workload!

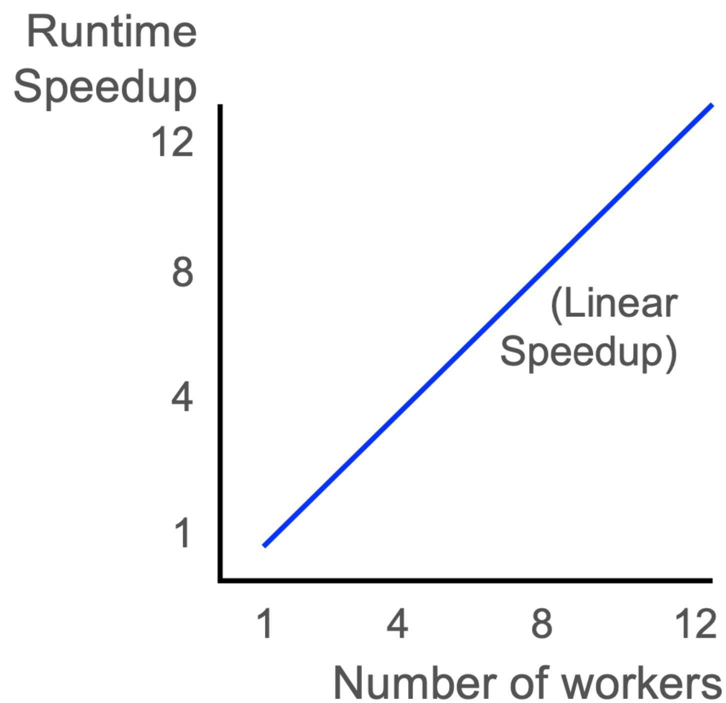
# Recap: Quantifying Parallelism Benefit

$$\textbf{Speedup} = \frac{\text{Completion time given only 1 worker}}{\text{Completion time given n (>1) workers}}$$

***Q:*** *But given n workers, can we get a speedup of n?*

It depends!

(On degree of parallelism, task dependency graph structure, intermediate data sizes, etc.)

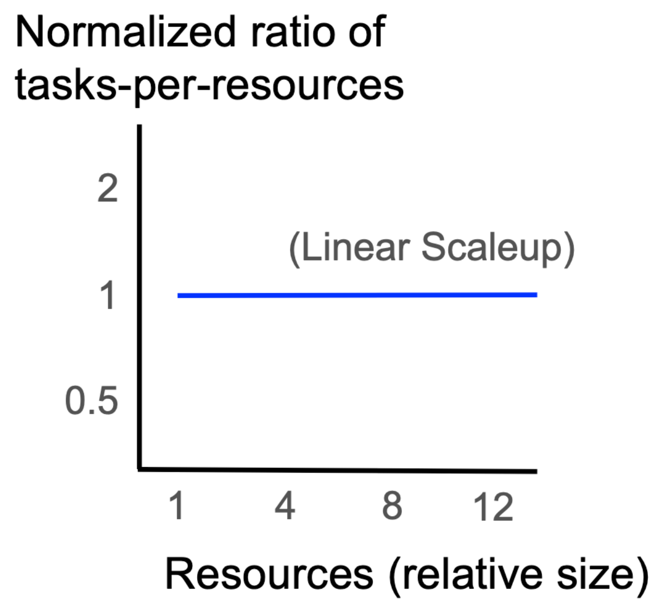


# Recap: Quantifying Parallelism Benefit (continued)

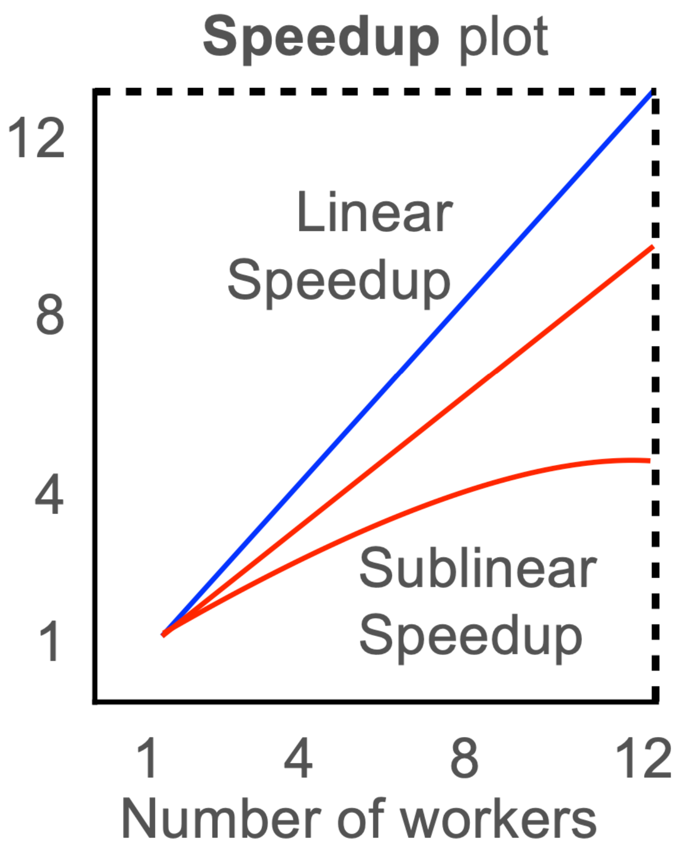
**Scaleup** refers to the ability of a system to retain the same performance ratio of tasks-per-resources when both the tasks and the resources increase at same rate.

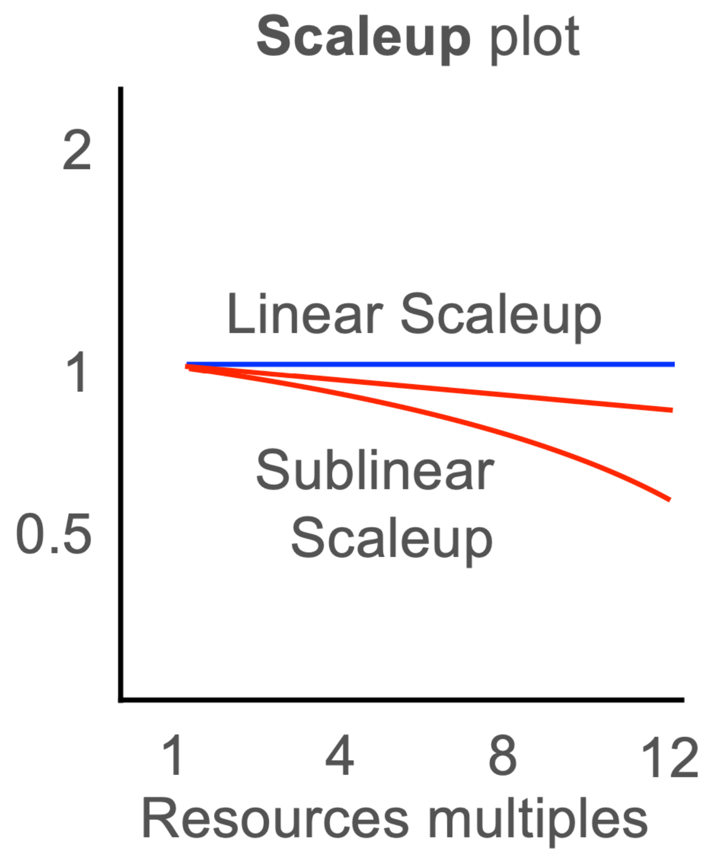
In the above:

- "Task" can refer to a single or series of computations, queries, etc.
- "Resources" can refer to # of workers, DRAM, storage size, etc.
- "Increase" refers to using multiple instances of an initial task and initial set of resources.



# Recap: Speedup vs Scaleup



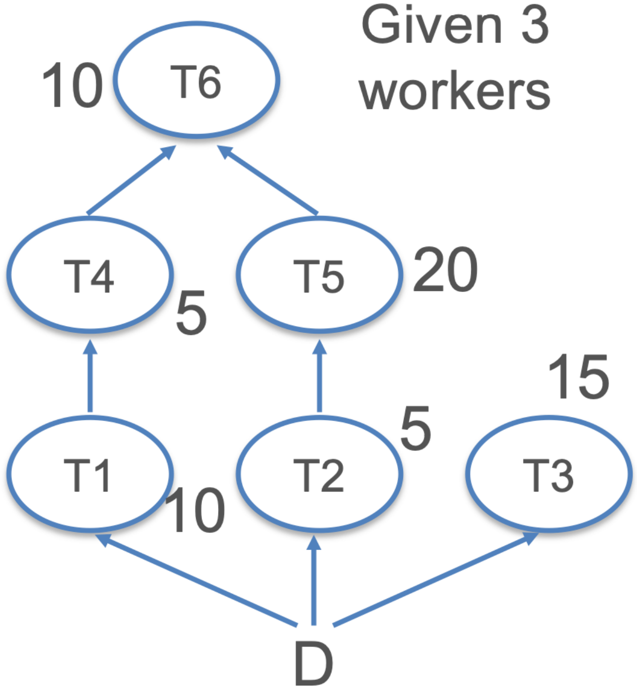




Most commonly, scaling does not demonstrate ideal linear behavior.

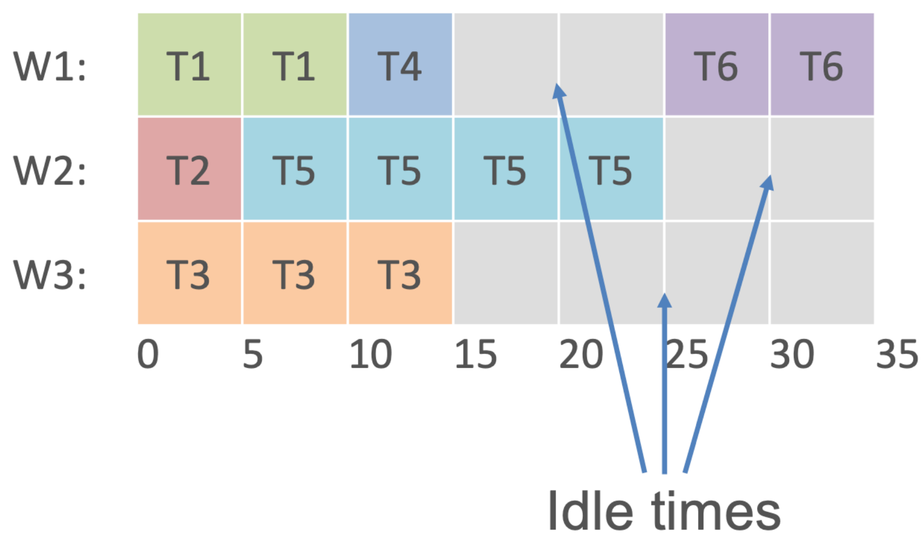
# Recap: Task Graphs and Gantt Chart

**Example:**

Given 3 workers




Gantt Chart visualization of schedule:




| | |
|---|---|
| Completion time with 1 worker | 10+5+15+5+ 20+10 = 65 |
| Parallel completion time | 35 |

Speedup = 65/35 = 1.9x

Ideal/linear speedup is 3x