https://hao-ai-lab.github.io/dsc204a-w24/

DSC 204A: Scalable Data Systems
Winter 2024

Machine Learning Systems

Big Data

Cloud

Foundations of Data Systems

OS: basically, a software between apps and hardware

- Goal 1: Provide convenience to users
- Goal 2: Efficiency -- Manage compute, memory, storage resources
  - Goal 2.1: Running N processes Not N times slower
    - As fast as possible
  - Goal 2.2: Running N apps
    - Even when their total memory >> physical memory cap
- Goal 3: Provide **protection**
  - One process won't mess up the entire computer
  - One process won't mess up with other processes

Process management

Memory management

System calls

Process management: Can we do better?

“System Call” APIs: Isolation and protection

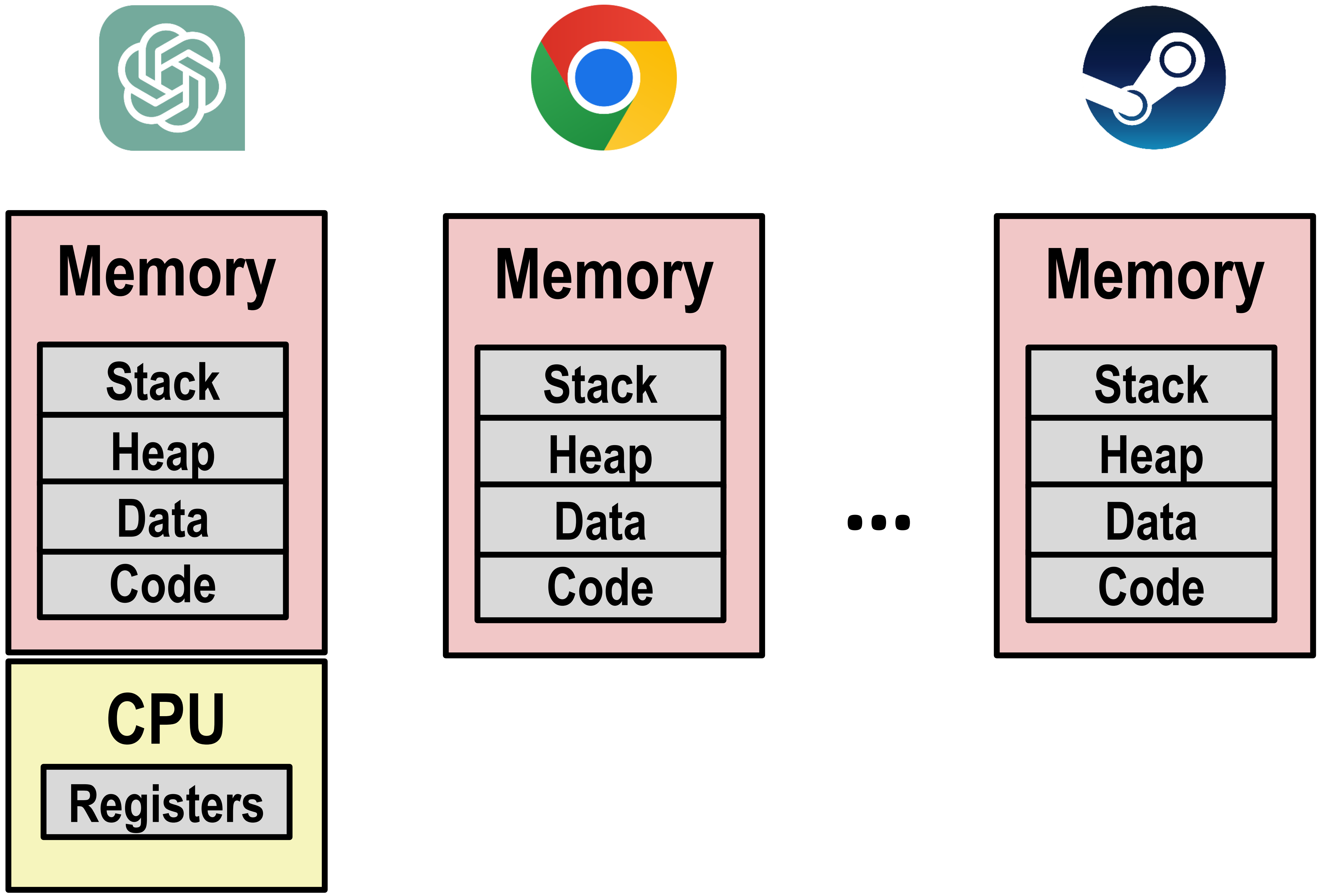
Kernel Components	Process Management	Main Memory Management	Filesystems	Networking	Device Drivers
Functionality	*Virtualize* processor; “**Process**” abstraction;	*Virtualize* Main Memory	*Virtualize* disks; “**File**” abstraction;	Commun. over network	Talk to other I/O devices

Hardware device-specific programs

Hardware

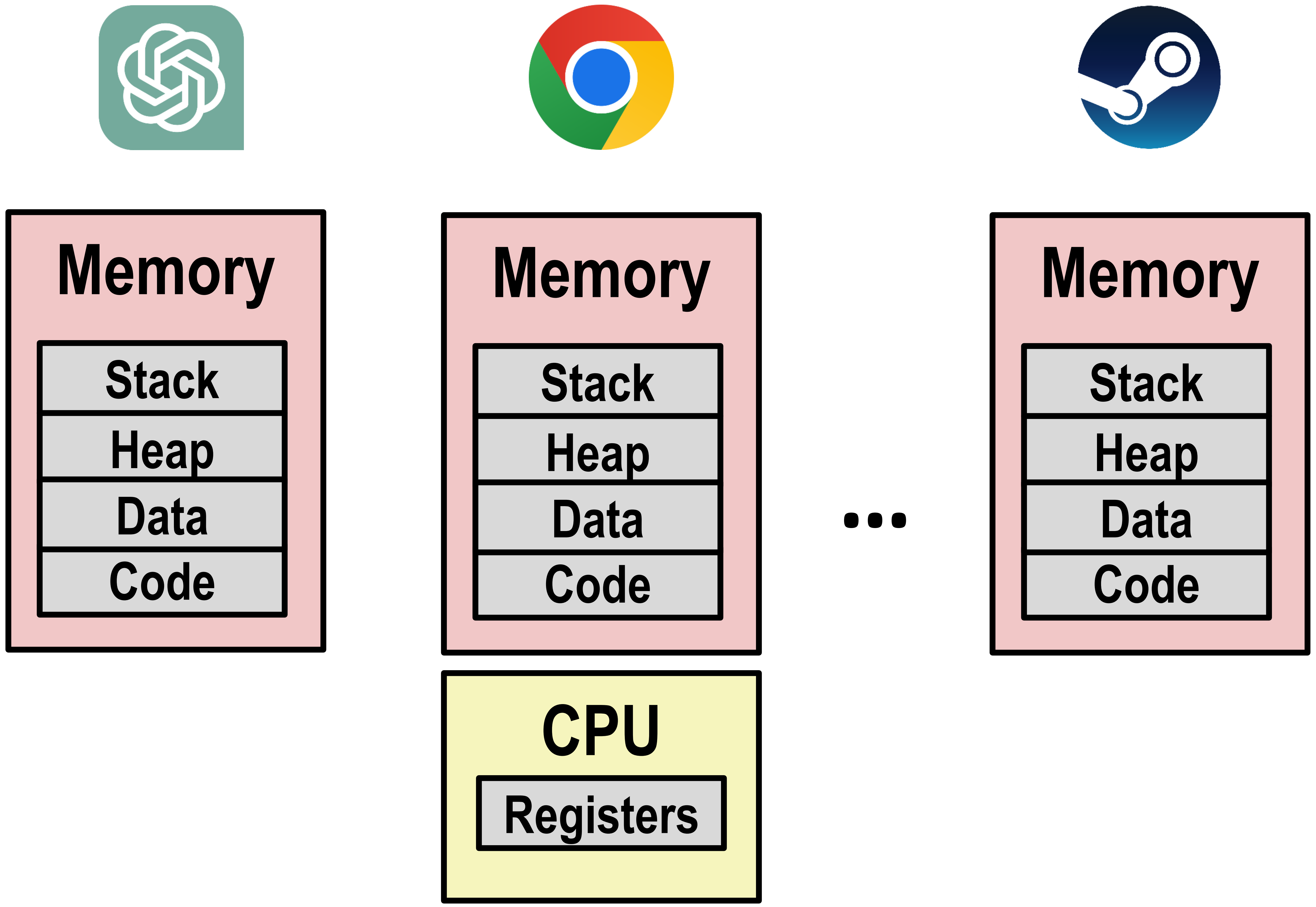
Multiprocessing: A strawman solution

- Assign individual memory (say 1/3) to each APP
- Assign CPU to work on an APP until completion -> then next



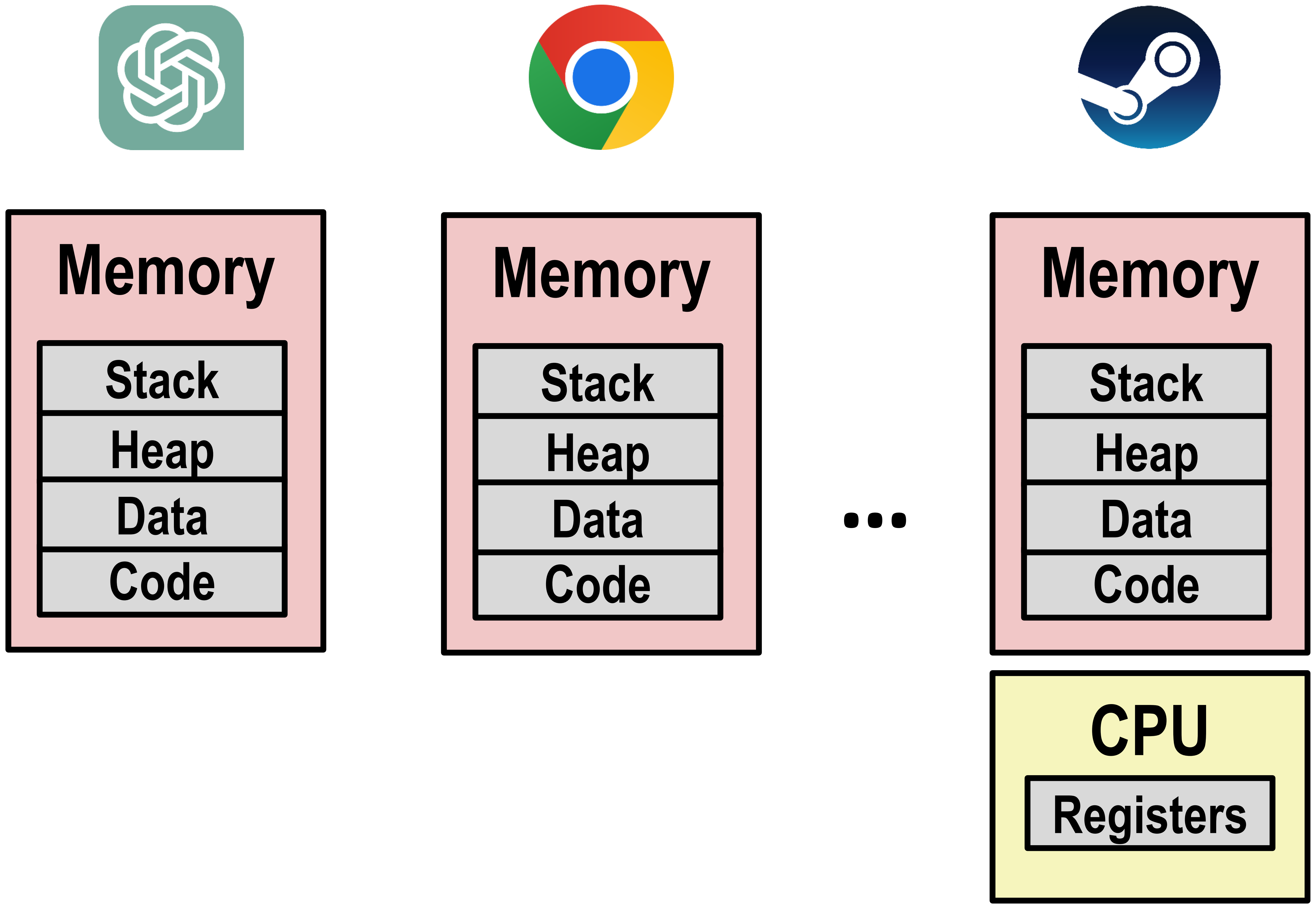
Multiprocessing: A strawman solution

- Assign individual memory (say 1/3) to each APP
- Assign CPU to work on an APP until completion -> then next



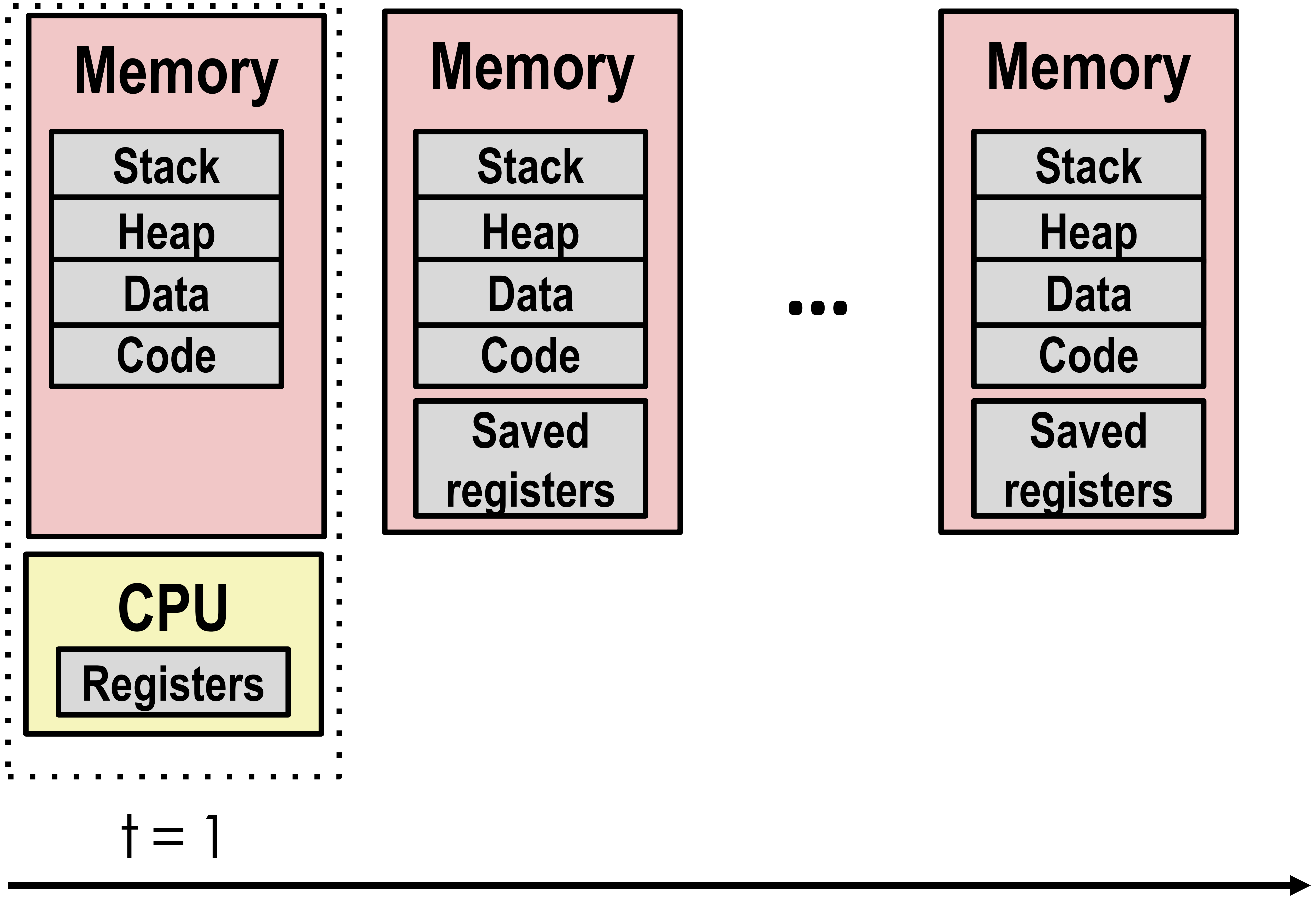
Multiprocessing: A strawman solution

- Assign individual memory (say 1/3) to each APP
- Assign CPU to work on an APP until completion -> then next



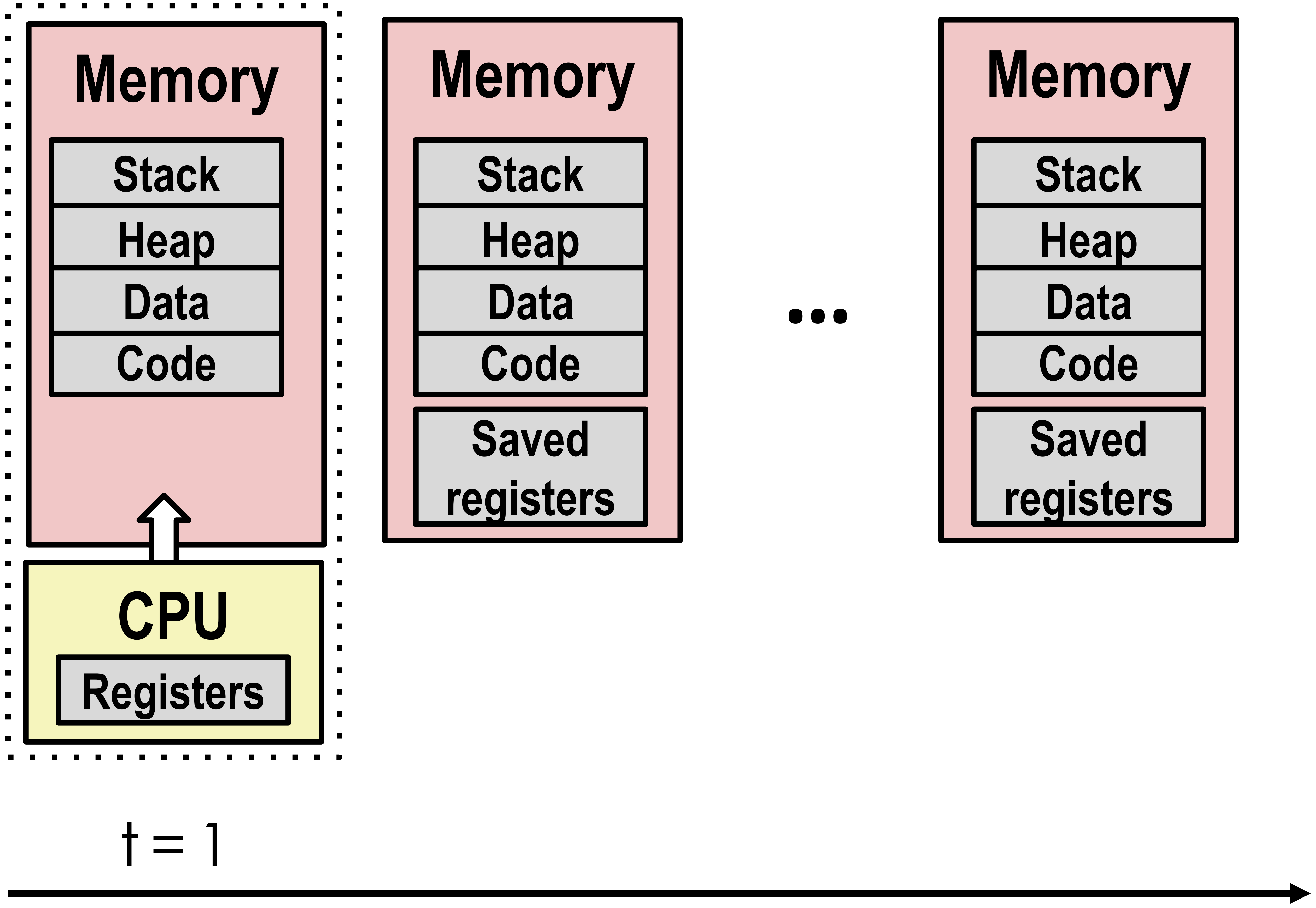
G1. Convenient?
G3: protection?
G2. Efficient?
- G2.1 can I run N processes but not N times slower?

Multiprocessing: Time sharing of processors



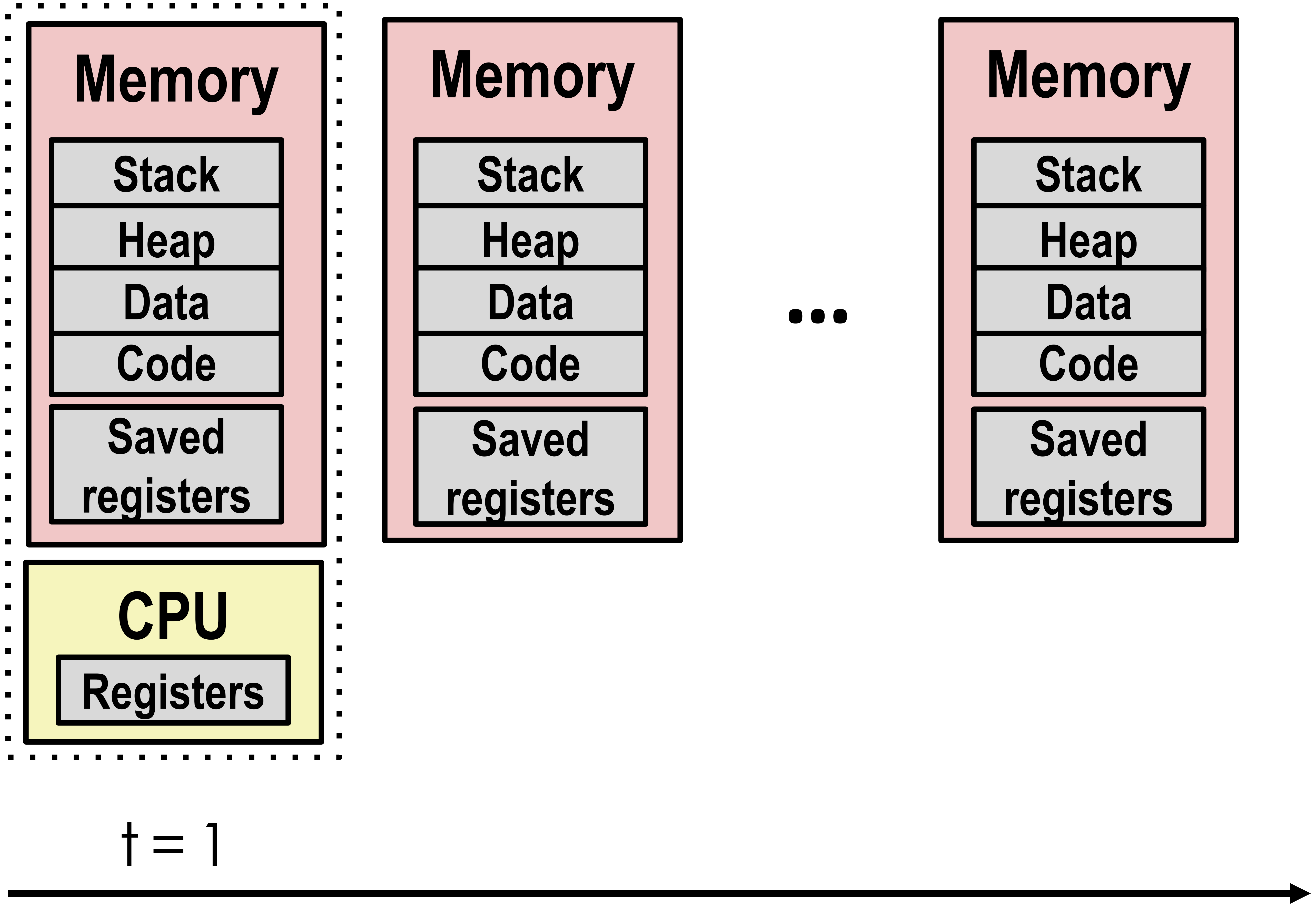
- Idea: Virtualize the CPU time as time slices
- Assign time slices to different processes

Multiprocessing: Time sharing of processors



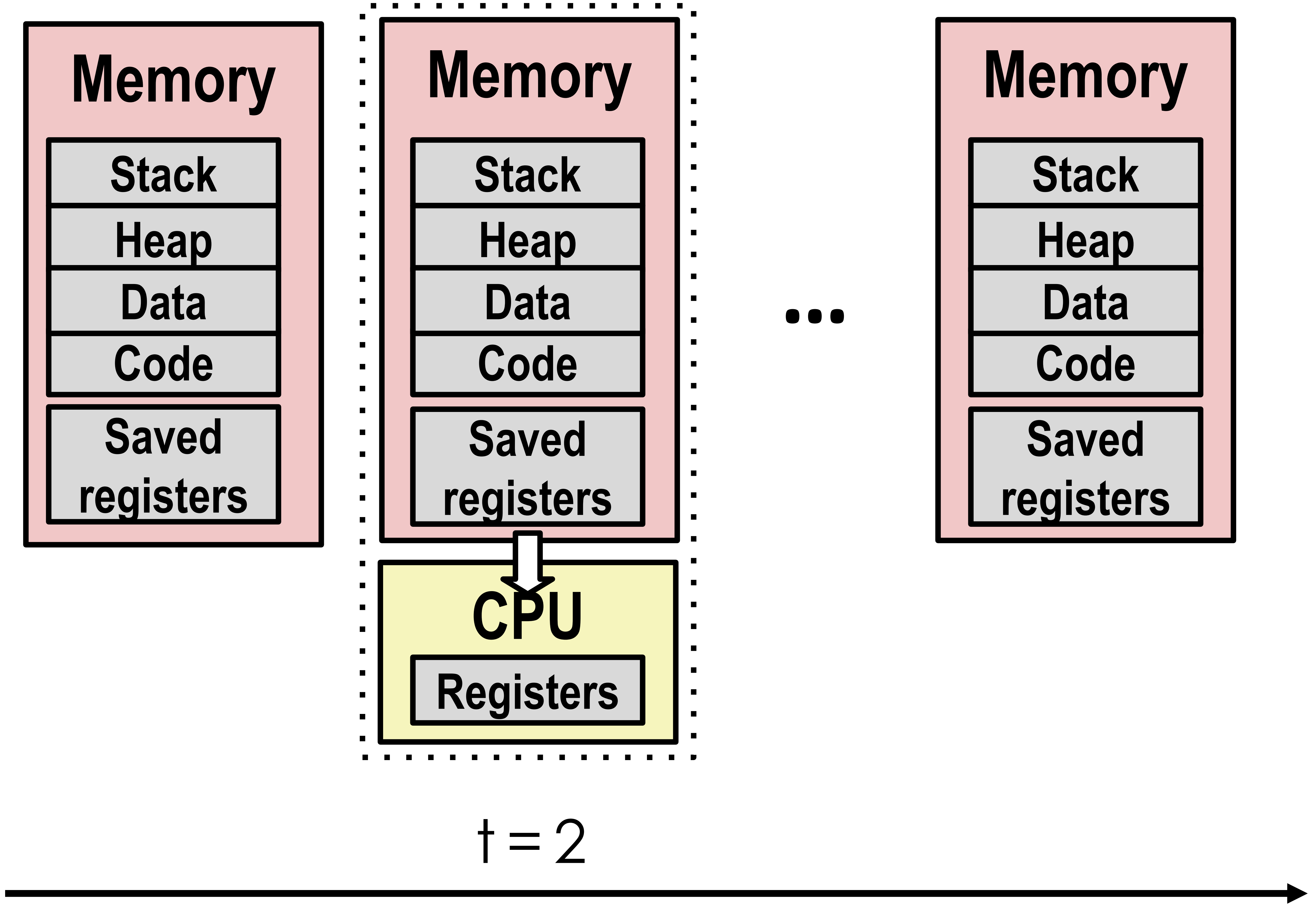
- Save current registers in memory

Multiprocessing: Time sharing of processors



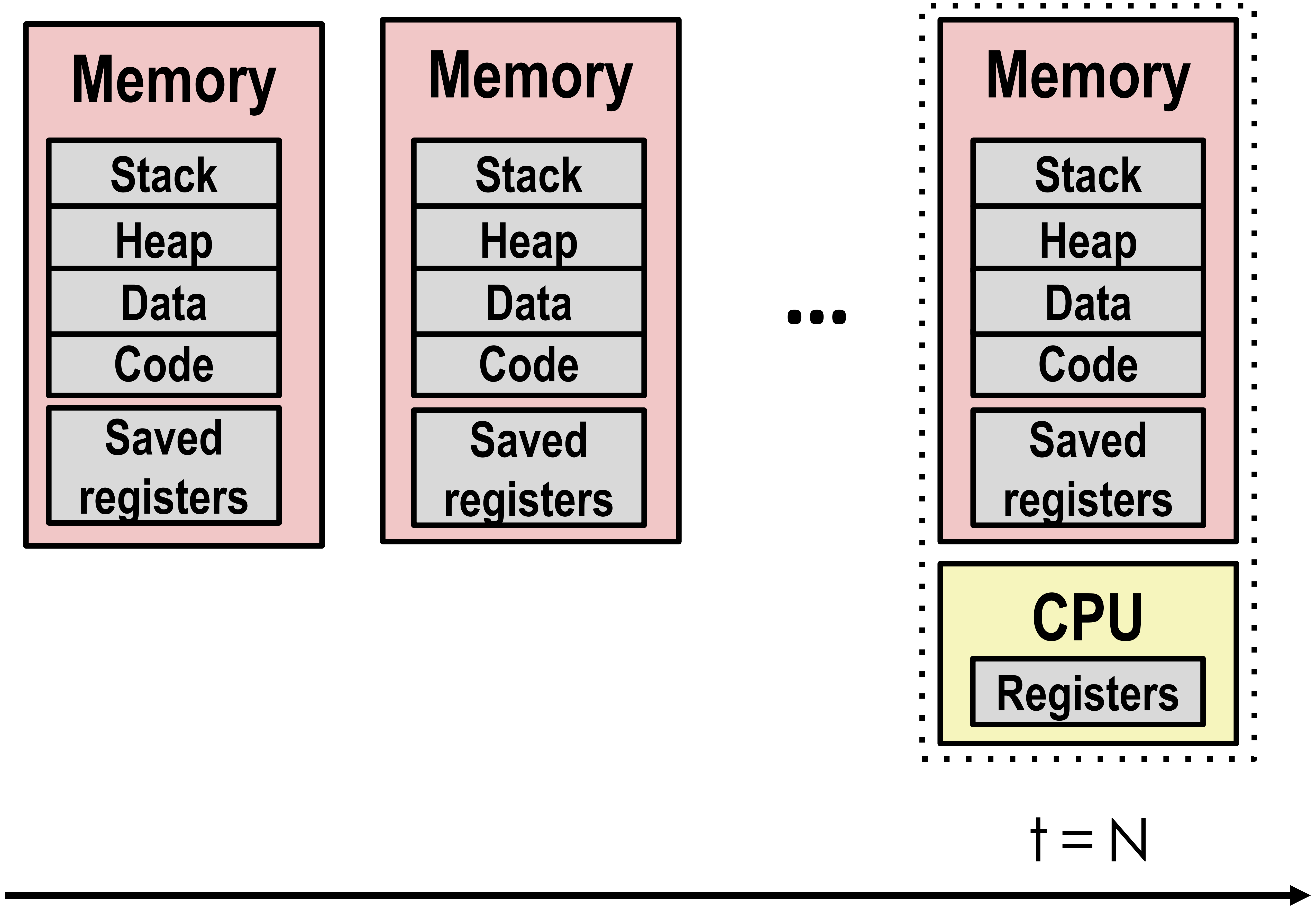
- Save current registers in memory

Multiprocessing: Time sharing of processors



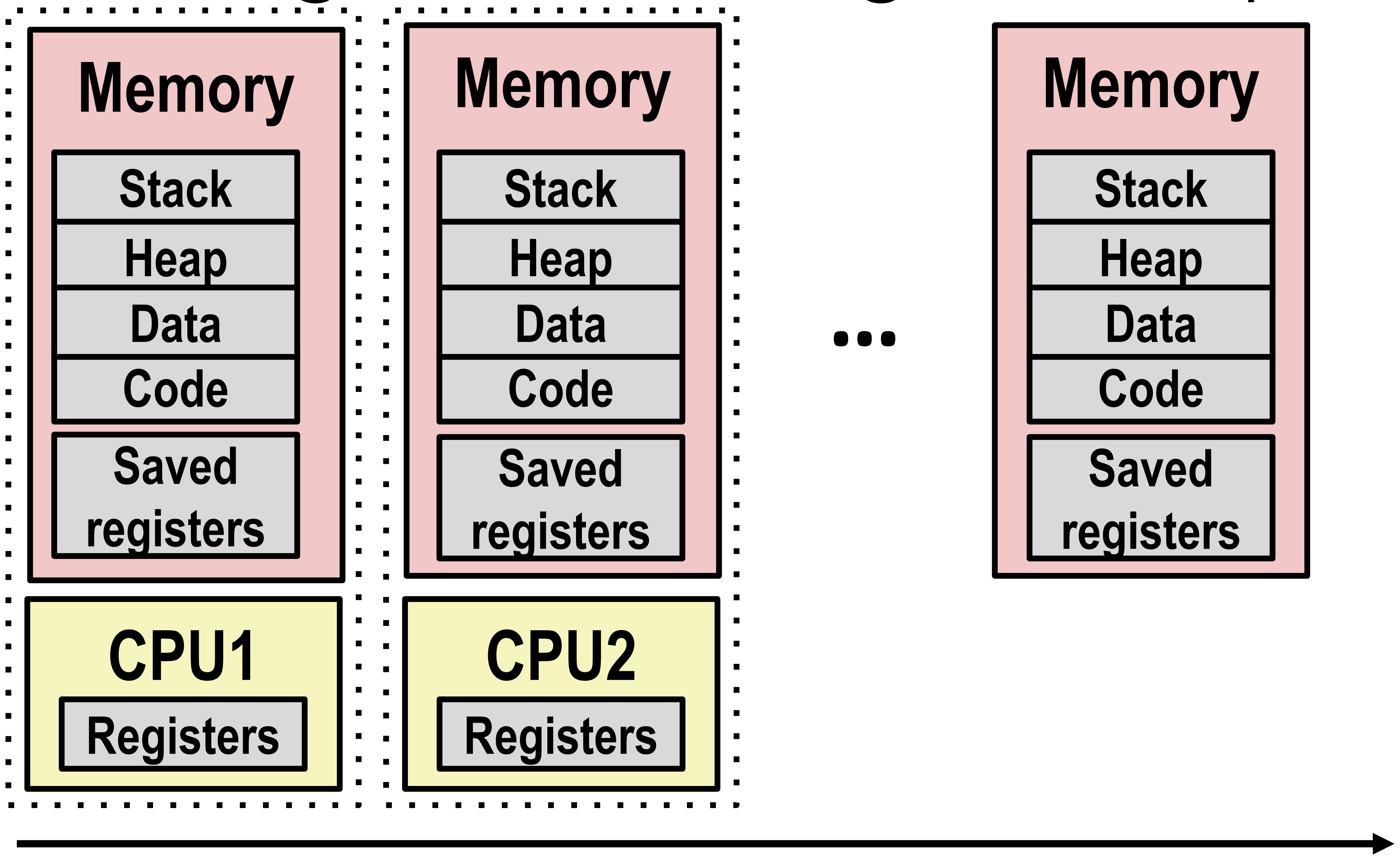
- Assign time slice t = 2 to the next process
- Resume progress: Move Saved registers from memory to CPU

Multiprocessing: Time sharing of processors



- Then we repeat.
- This is called **context switch**

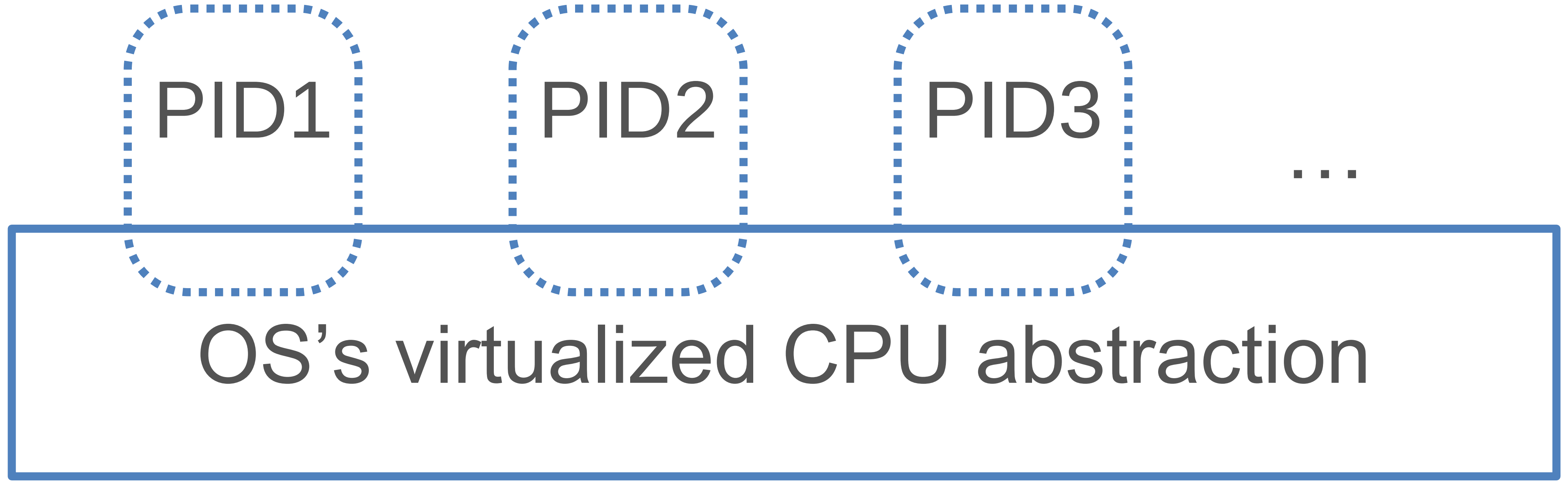
Multiprocessing: Time sharing of multiple processors



Multiple CPU cores?

1. All processors sweep from left (1st process) to right (last process)
2. Each process accounts for ½ of the processes

Let’s Implement It!

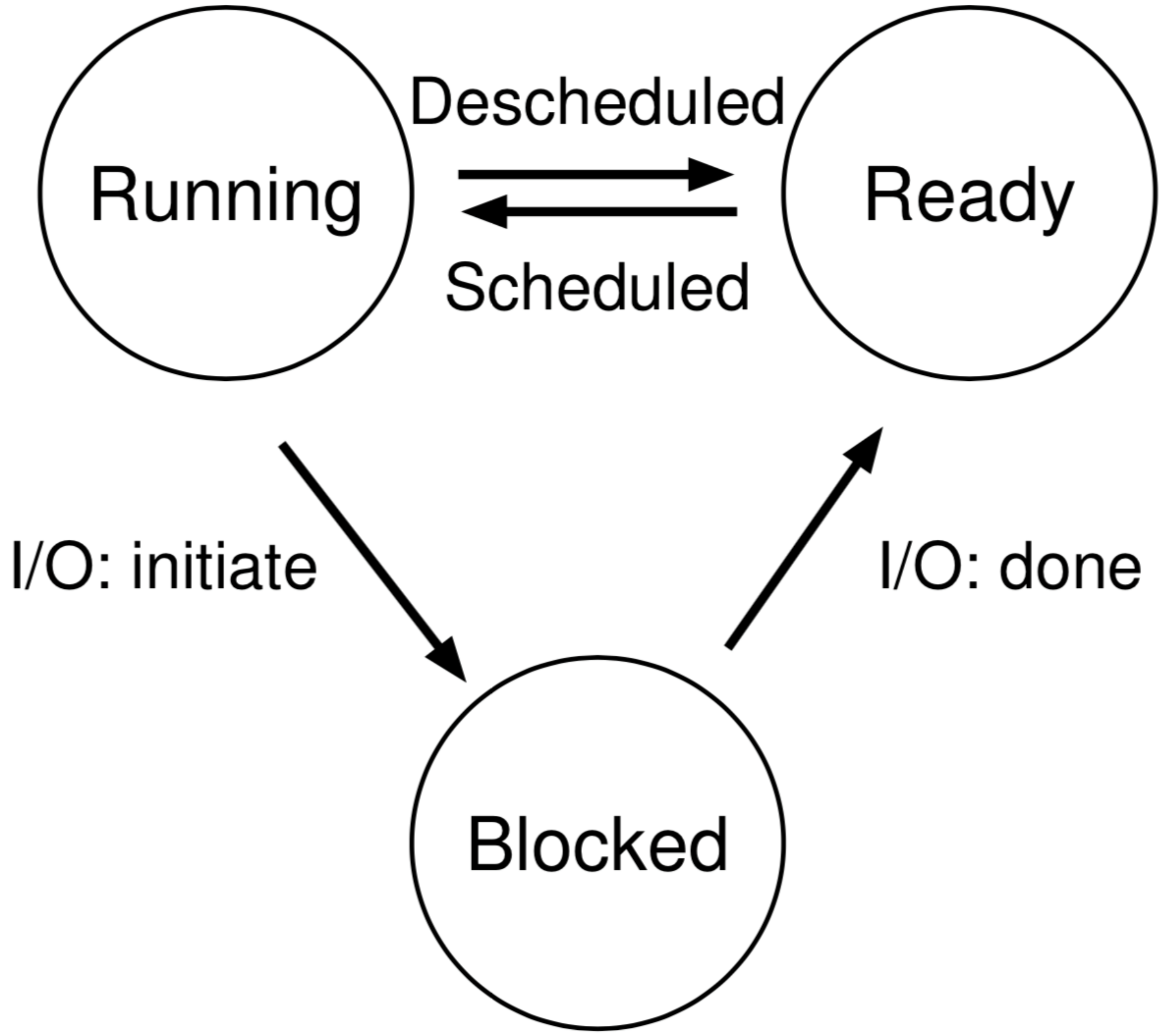


GAP1: How to virtualize CPU resources **temporally** and **spatially**?

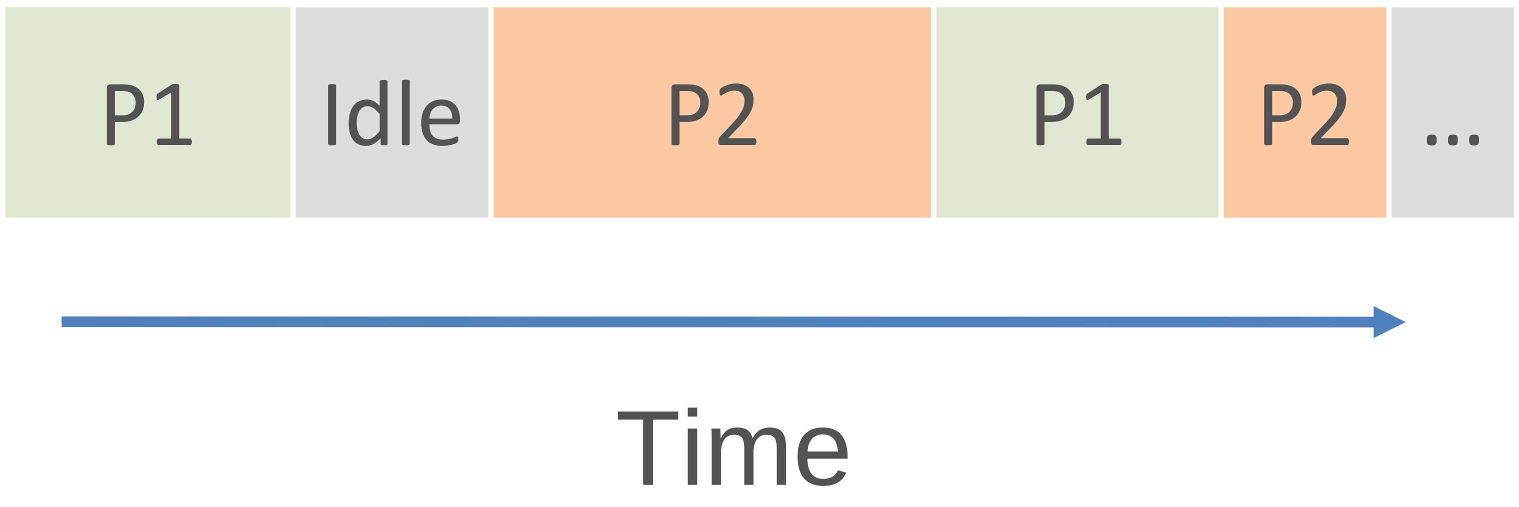
Physical Processor

Temporal Abstraction: Process State and CPU Time

- OS keeps moving processes between 3 states:



- Gantt Chart: A viz. to show what process runs when (on processor)



Scheduling question naturally emerges:
Q: how to schedule processes on time axis so **the objective** is optimal?

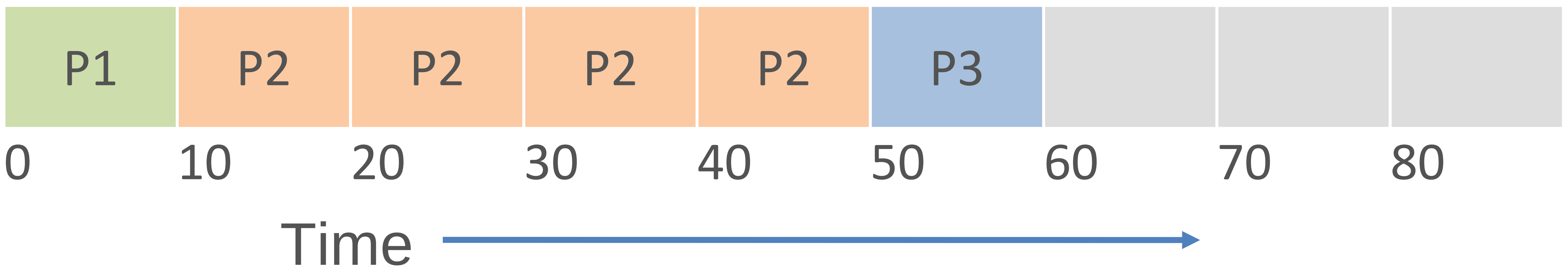
Scheduling Policies/Algorithms

- 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: Time 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

Scheduling Policy: FIFO

- First-In-First-Out aka First-Come-First-Serve (FCFS)
- Ranking criterion: Arrival Time; no preemption allowed

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order



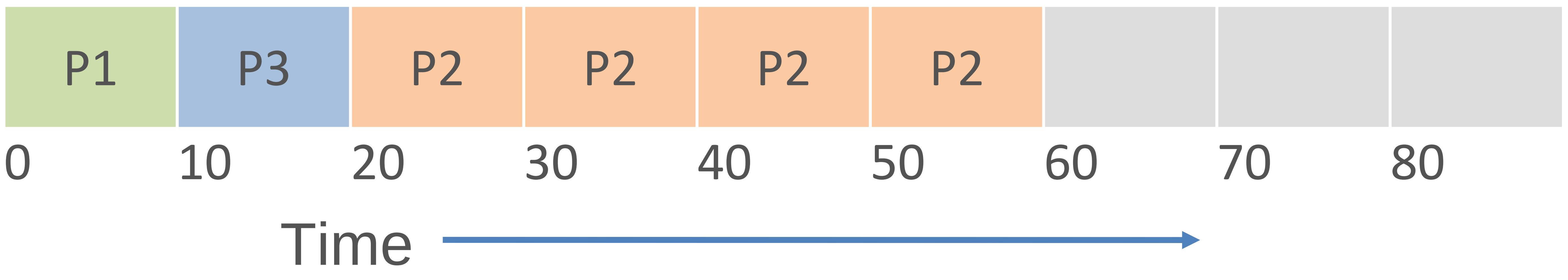
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	10	50	10	50
P3	0	50	60	50	60
			Avg:	20	40

- Main con: Short jobs may wait a lot, aka “Convoy Effect”

Scheduling Policy: SJF

- Shortest Job (next) First
- Ranking criterion: Job Length; no preemption allowed

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order



Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	20	60	20	60
P3	0	10	20	10	20
	(FIFO Avg: 20 and 40)		Avg:	10	30

- Main con: Not all Job Lengths might be known beforehand
- Long processes may be held off indefinitely

Example Exam Q1: Round Robin Schedule

- RR does not need to know job lengths
- Fixed time *quantum* given to each job; cycle through jobs

Example: P1, P2, P3 of lengths 10,40,10 units arrive closely in that order

P1	P2	P3	P1	P2	P3	P2	P2	P2	P2	P2	P2				
0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75

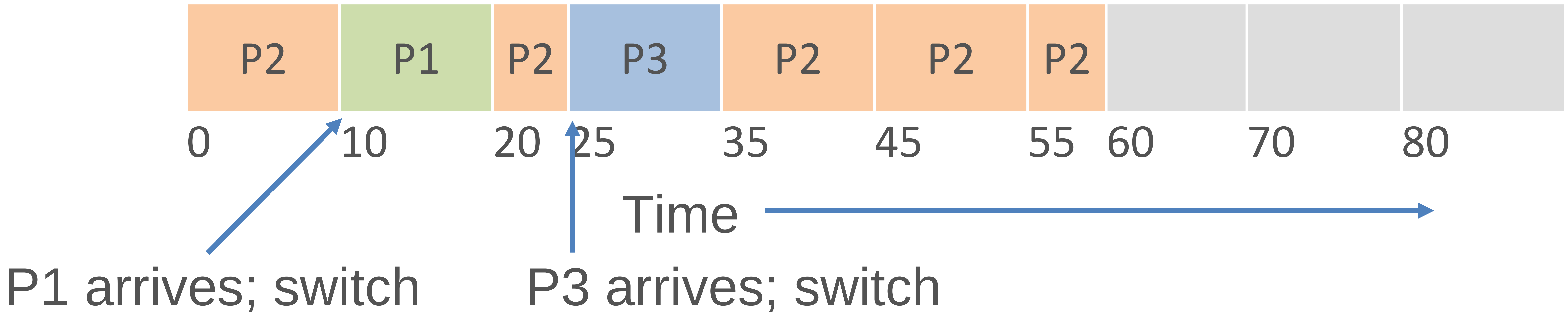
Quantum is 5 Time →

- RR is often very fair, but Avg Turnaround Time goes up!

Example Exam Q2: SCTF

- Shortest Completion Time First
- Jobs might not all arrive at same time; preemption possible

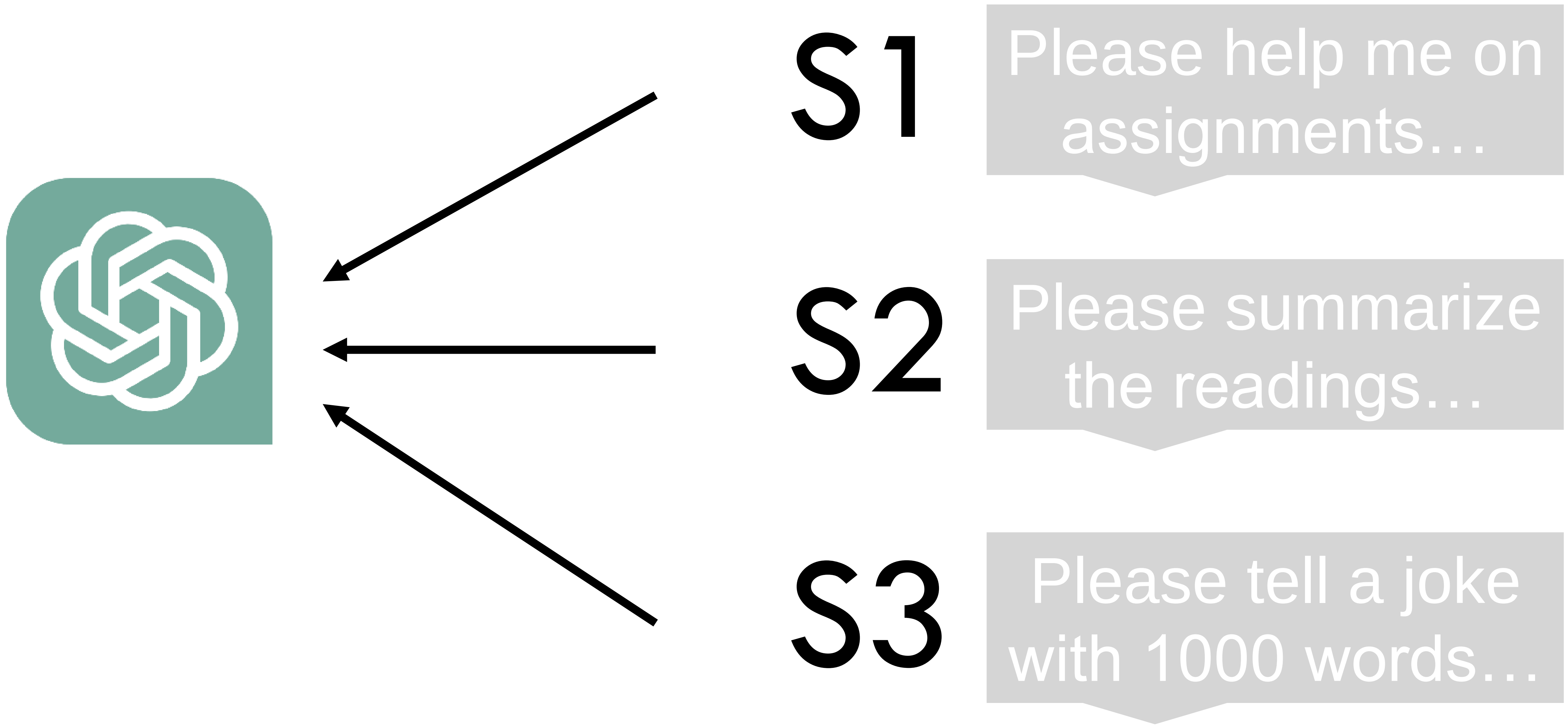
Example: P1, P2, P3 of lengths 10,40,10 units arrive at different times



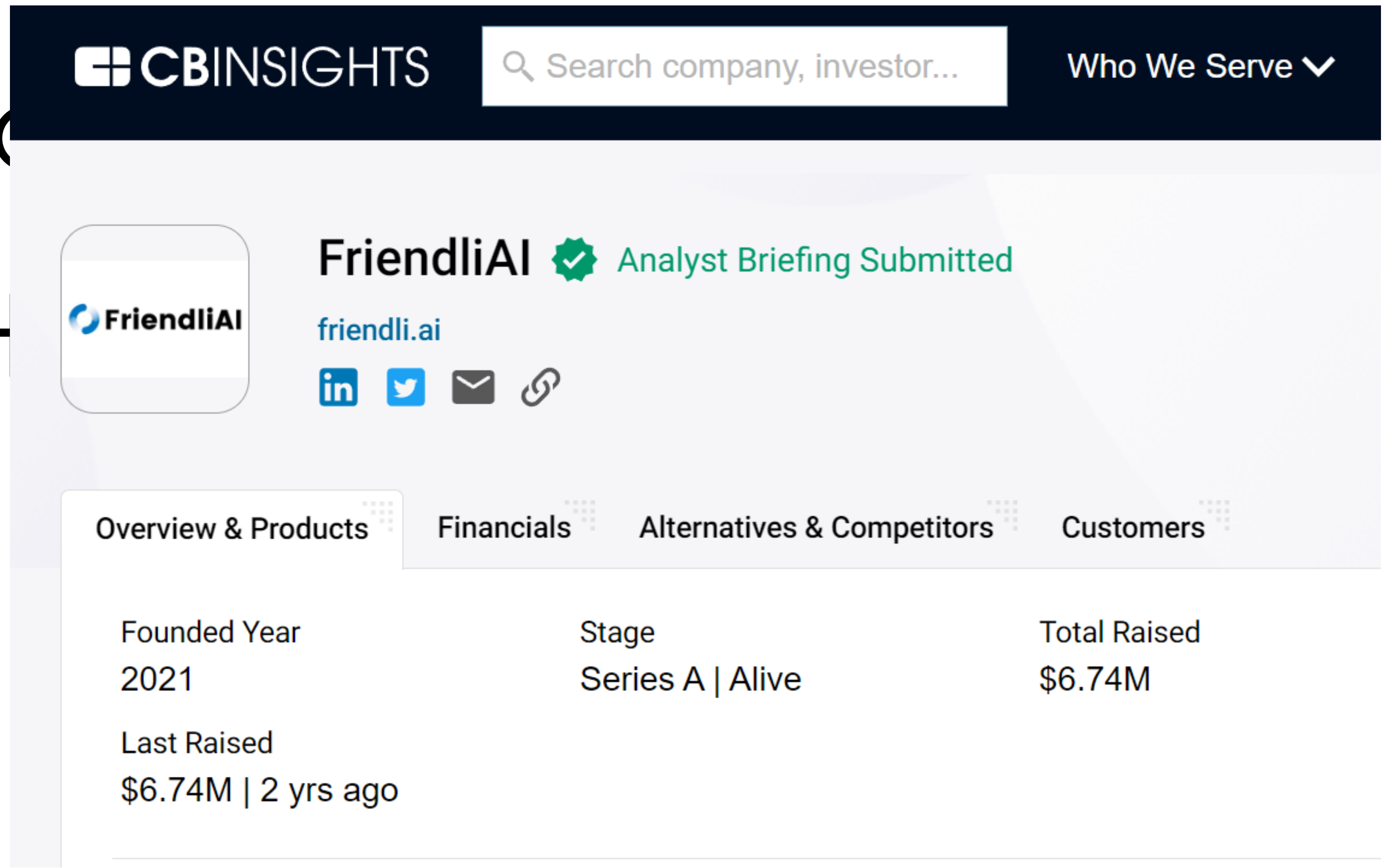
Scheduling Policies/Algorithms

- In general, not all Arrival Times and Job Lengths will be known 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*
  - Many fairness metrics exist, e.g., Jain's fairness index
- 100s of scheduling policies studied! Well-known ones: FIFO, SJF, STCF, Round Robin, Random, etc.
  - Different criteria for ranking; preemptive vs not
  - Complex "multi-level feedback queue" schedulers
  - ML-based schedulers are "hot" nowadays!

Scheduling in ChatGPT



- What is the response time
- What is the turnover time
- What is fairness?
- Do we know the job length?
- Can we run S1/S2/S3 t
- H



Orca: A Distributed Serving System for Transformer-Based Generative Models

Authors:
Gyeong-In Yu and Joo Seong Jeong, *Seoul National University;* Geon-Woo Kim, *FriendliAI and Seoul National University;* Soojeong Kim, *FriendliAI;* Byung-Gon Chun, *FriendliAI and Seoul National University*

Let’s Implement It!

PID1 PID2 PID3 …

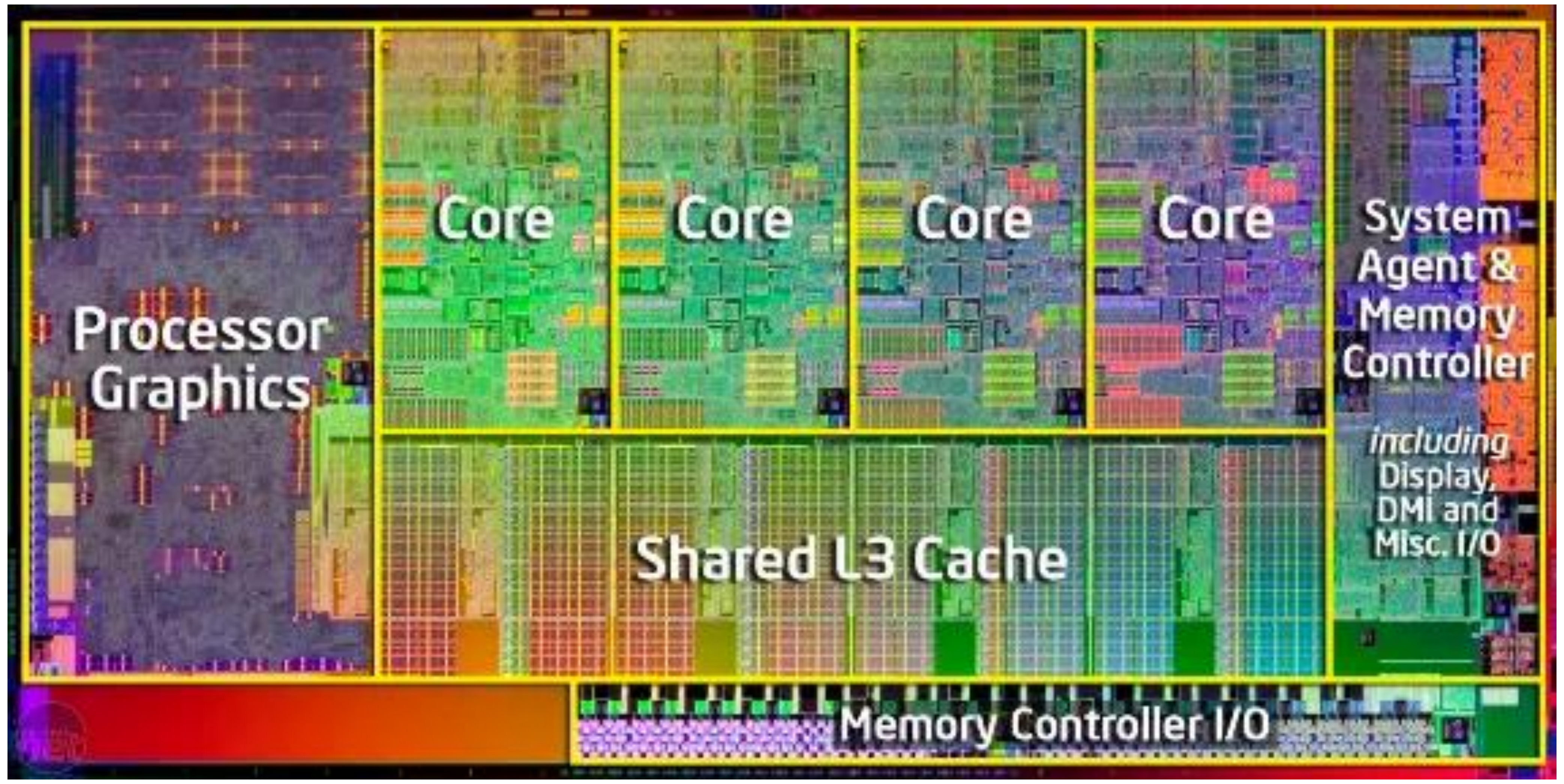
OS’s virtualized CPU abstraction

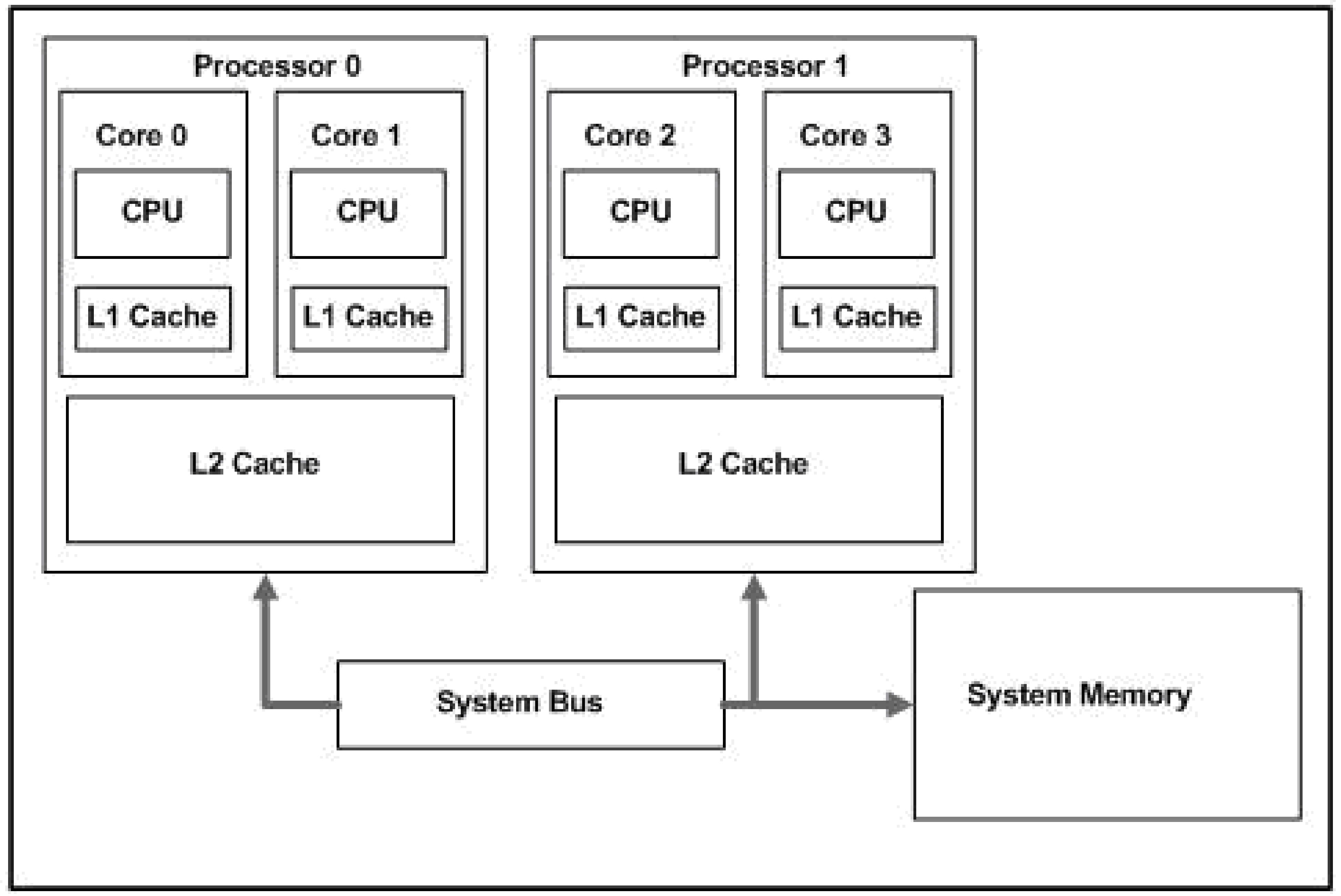
GAP2: How to virtualize CPU resources temporally and **spatially**?

Physical Processor

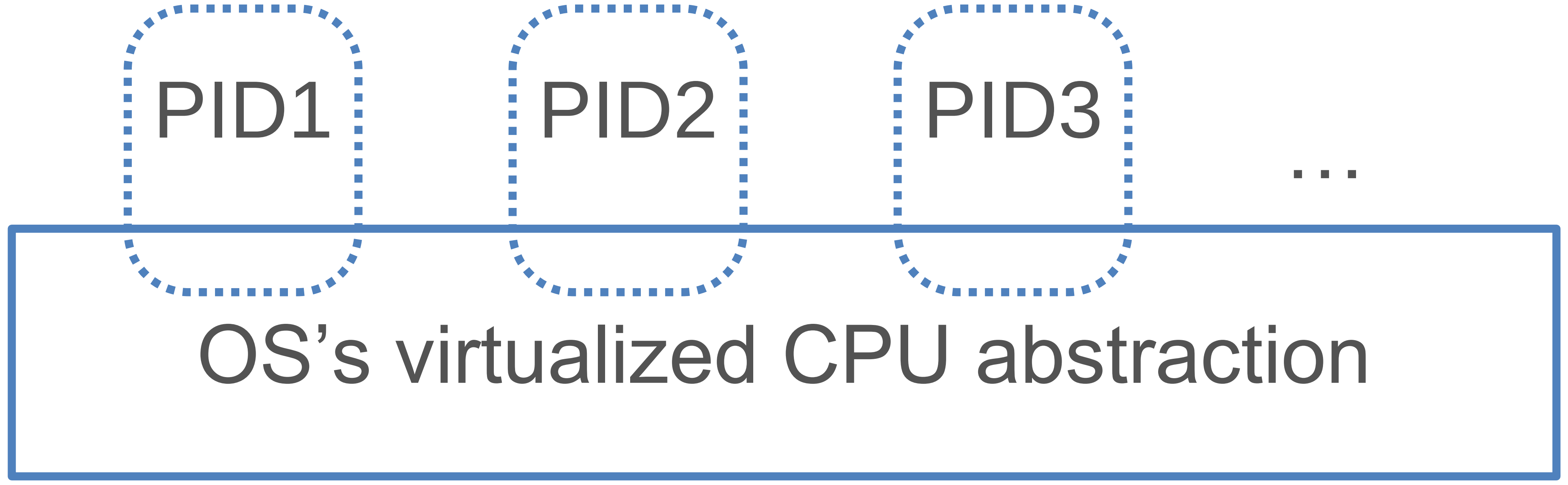
Concurrency

- Modern computers often have multiple processors and multiple *cores* per processor
- Concurrency: Multiple processors/cores run different/same set of instructions simultaneously on different/*shared* data





Let’s Implement It!



GAP2: How to virtualize CPU resources temporally and **spatially**?

Physical Processor

“Placement“ naturally emerges:

Q: how to place processes on each processor so **the objective** is optimal?

Concurrency

- Scheduling for multiprocessing/multicore is more complex
- **Load Balancing:** Ensuring different cores/proc. are kept roughly equally busy, i.e., reduce **idle times**
- Multi-queue multiprocessor scheduling (MQMS) is common
  - Each proc./core has its own job queue
  - OS moves jobs across queues based on load
  - Example Gantt chart for MQMS:

CPU 1:	P1	P1	P3	P3	P3	P3	P1	P1	P1
CPU 2:	P2	P2	P2	P1	P1	P2	P2	P3	P3
	0	10	20	30	40	50	60	70	80