https://hao-ai-lab.github.io/dsc204a-f25/

DSC 204A: Scalable Data Systems
Fall 2025

Staff

Instructor: Hao Zhang

TAs: Mingjia Huo, Yuxuan Zhang

@haozhangml @haoailab

haozhang@ucsd.edu

Where We Are

Machine Learning Systems

Big Data

Cloud

Foundations of Data Systems

1980 - 2000

Logistics

- Beginning of Quarter Survey: 77% completion
- Finish the 3% and you all get 1 point!

Qualitative Estimates of Locality

Assuming row-major array

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Answer: yes

a[0][0]	• • •	a[0][N-1]	a[1][0]	• • •	a[1][N-1]	• • •	a[M-1][0]	• • •	a[M-1][N-1]

Question: Does this function have good locality with respect to array `a`?

Locality Example

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Answer: no, unless…

M is very small

- Question: Does this function have good locality with respect to array `a`?

a [0] [0]	. . .	a [0] [N-1]	a [1] [0]	. . .	a [1] [N-1]	. . .	a [M-1] [0]	. . .	a [M-1] [N-1]

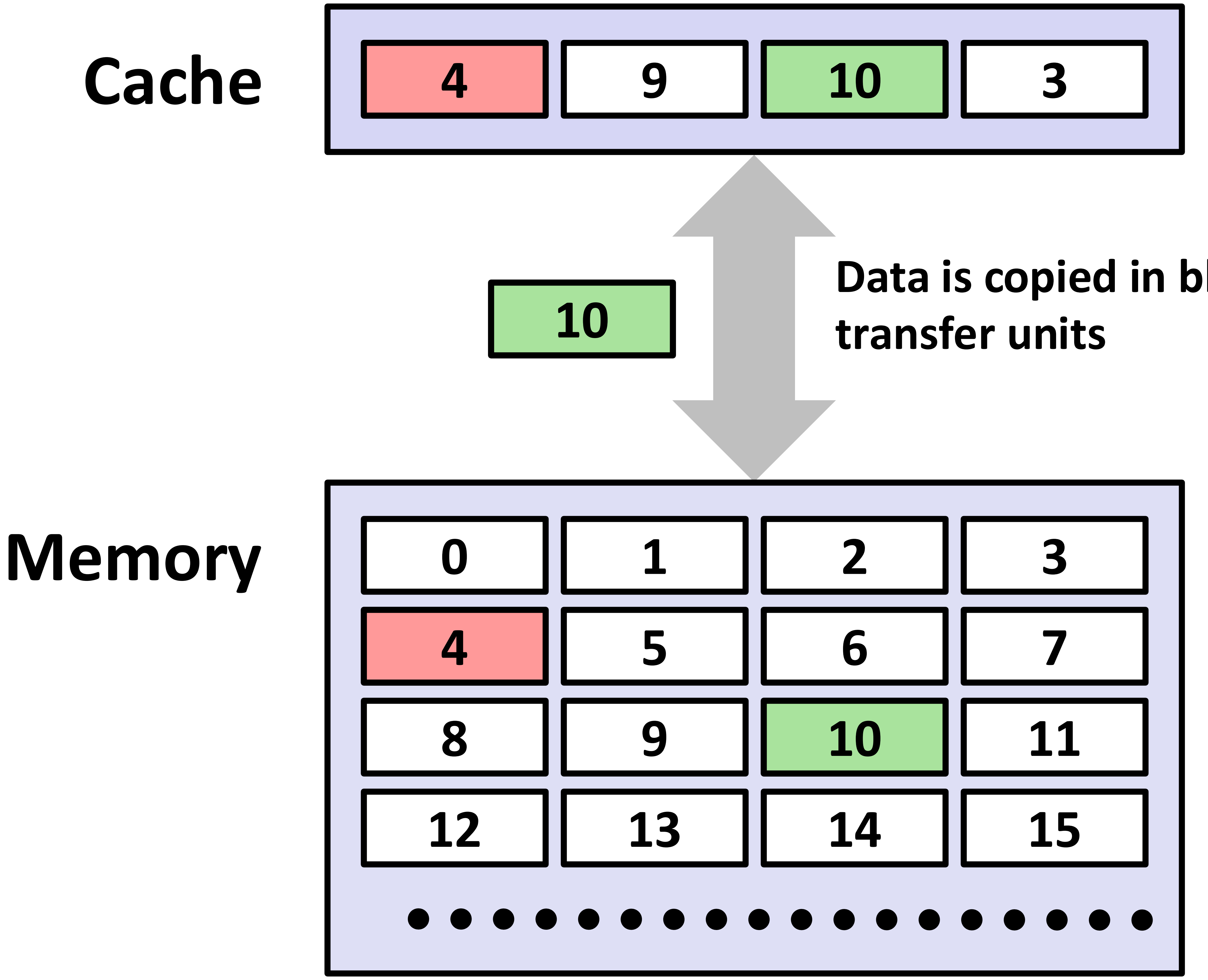
Example Exam Question

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < M; k++)
                sum += a[k][i][j];
    return sum;
}
```

- Question: Can you permute the loops so that the function scans the 3-d array `a` with a stride-1 reference pattern (and thus has good spatial locality)?

Cache in action

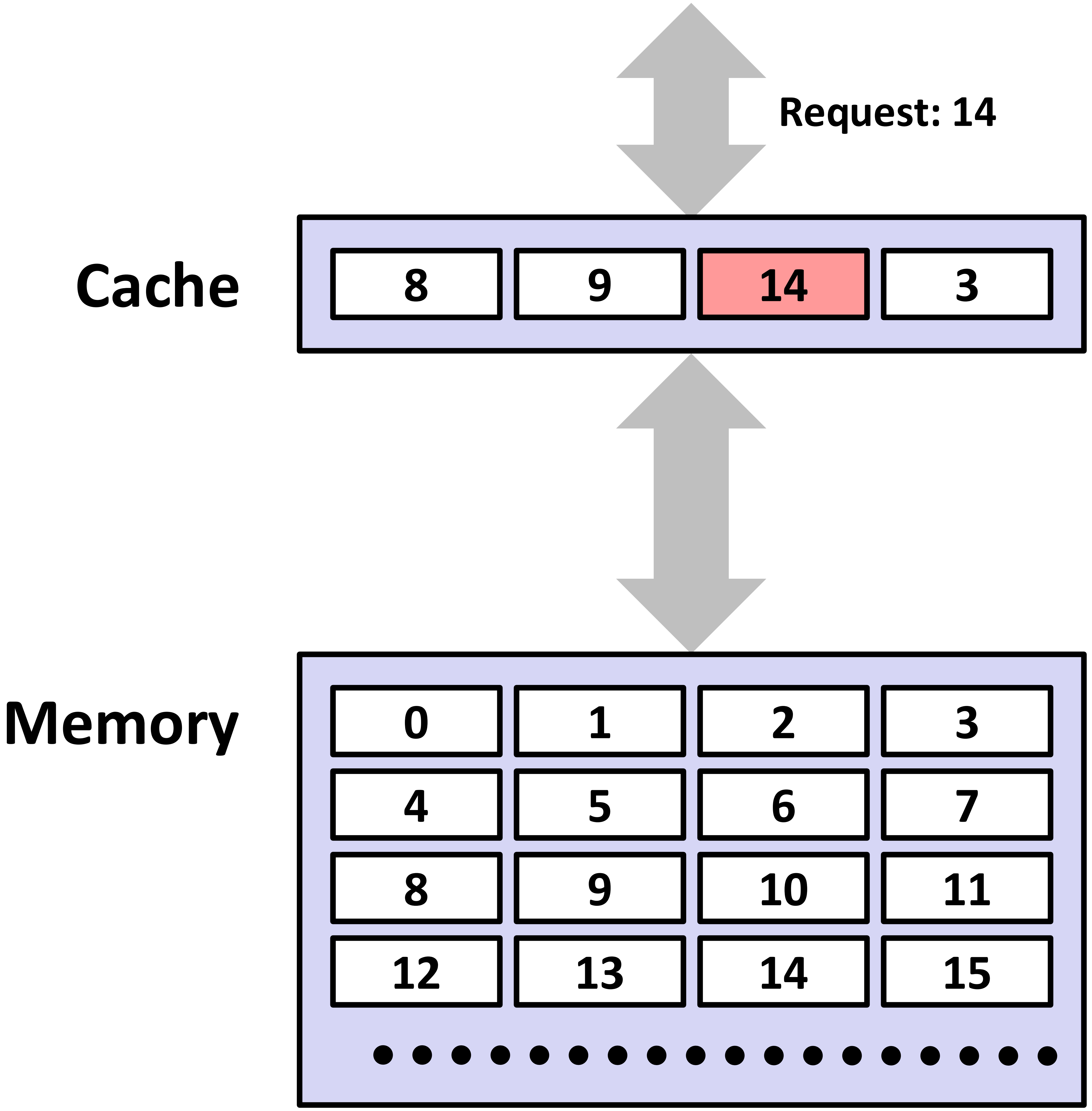


Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into “blocks”

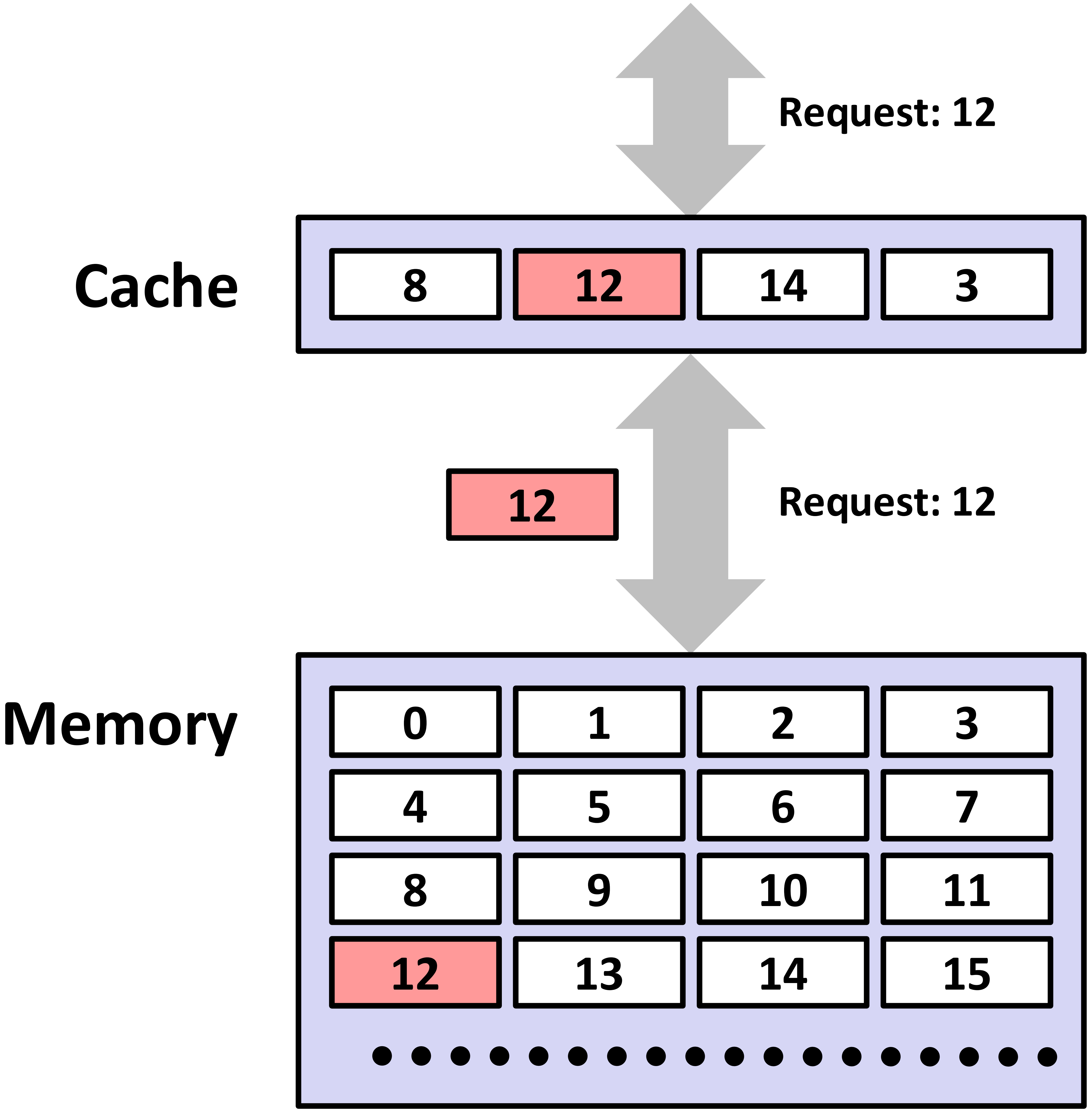
General Cache Concepts: Hit



Data in block 14 is needed

Block 14 is in cache:
Hit!

General Cache Concepts: Miss



Data in block 12 is needed

Block 12 is not in cache: Miss!

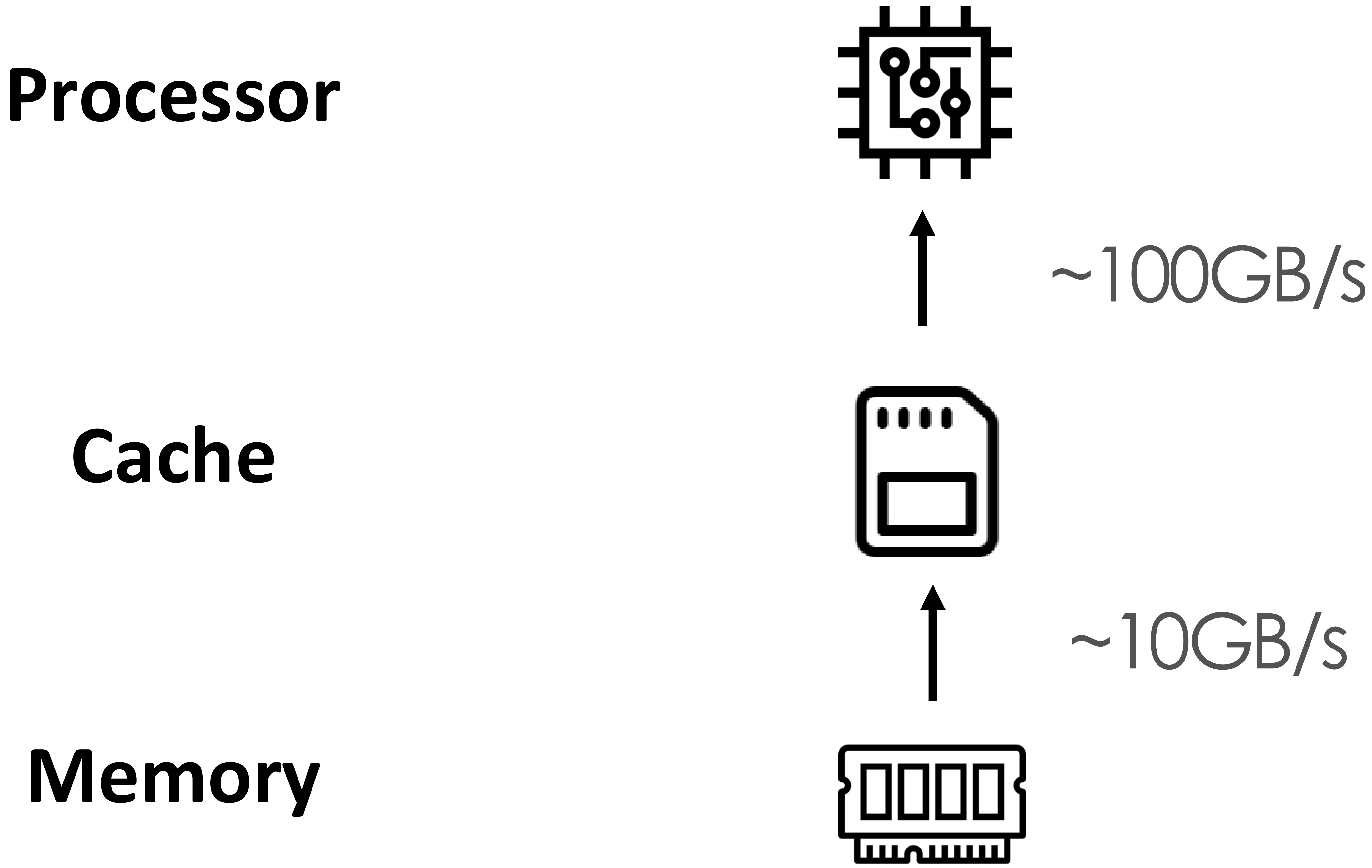
Block 12 is fetched from memory

Block 12 is stored in cache

- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

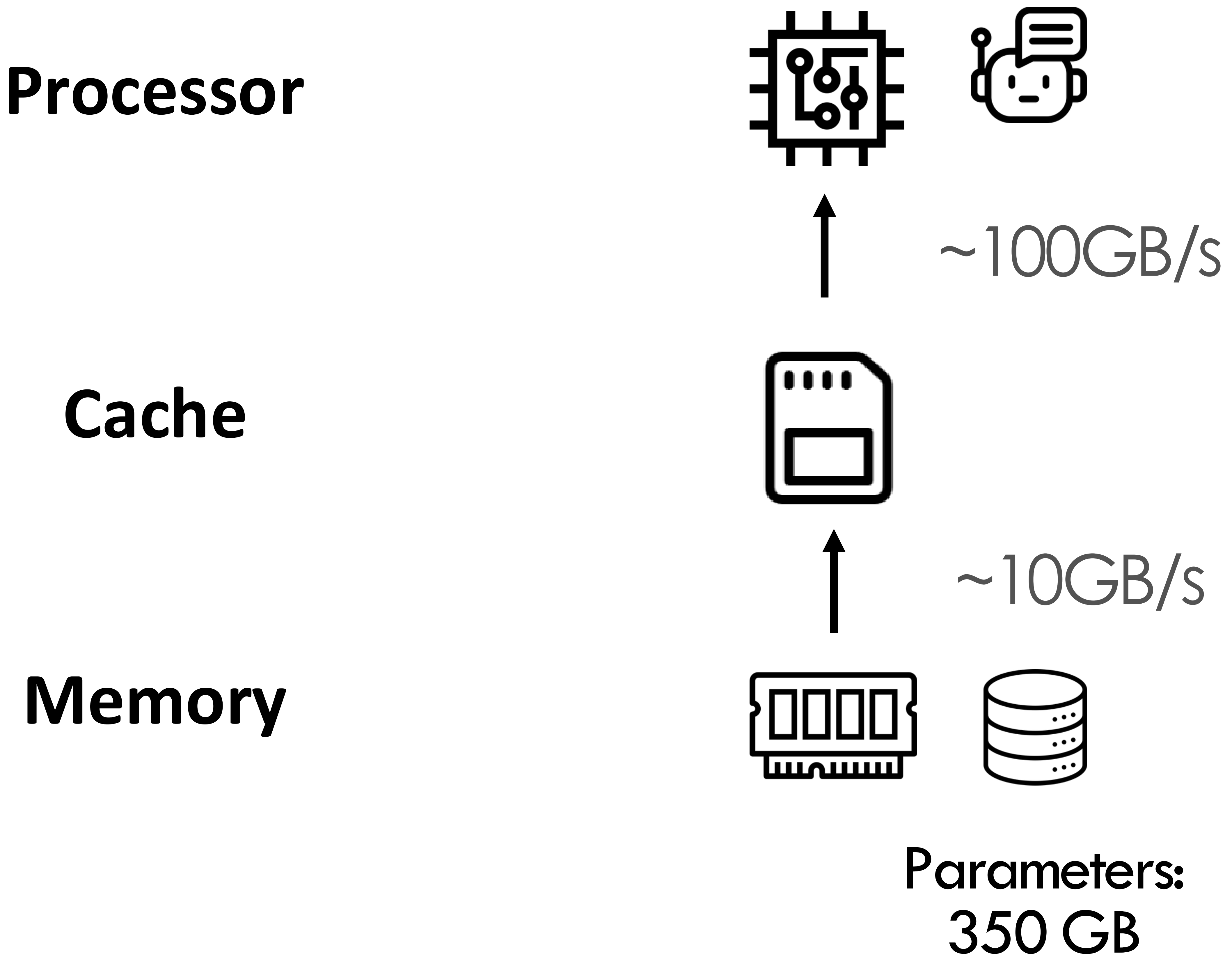
Cache in action

- If always cache hit, bandwidth?
- If always cache miss, bandwidth?



Open Question in Cache: ChatGPT

- ChatGPT: every time ChatGPT outputs token, it needs to see 350 GB parameters
- How to optimize this?



Foundation of Data Systems

- Computer Organization
  - Representation of data
  - processors, memory, storage
- **OS basics**
  - Process, scheduling
  - Memory

What is Operation System?

- Layers between applications and hardware

- OS makes computer hardware useful to programmers
  - Otherwise, users need to speak machine code to computer
- **[Usually]** Provides abstractions for applications
  - Manages and hides details of hardware
  - Accesses hardware through low/level interfaces unavailable to applications
- **[Often]** Provides protection
  - Prevents one app/user from clobbering another

A Primitive OS v1

- OS v1: just a library of standard services [no protection]

OS: interfaces above hw drivers

Hardware

- Simplifying assumptions:
  - System runs one program at a time
  - No bad users or programs (?)
- Problem: poor utilization
  - poor utilization of hardware (e.g., CPU idle while waiting for disk)
  - poor utilization of human user (must wait for each program to finish)

OS v2: Multi-tasking

- Say: we extend the OS a bit to support many APPs
  - When one process blocks (waiting for disk, network, user input, etc.) run another process

- Problem: What can ill-behaved process do?
  - Go into infinite loop and never relinquish CPU
  - Scribble over other processes' memory to make them fail
- OS provides mechanisms **protection** to address these problems:
  - Preemption – take CPU away from looping process
  - Memory protection – protect one process' memory from one another

What is A Real OS?

- OS: manage and assign hardware resources to apps
- Goal: with N users/apps, system not N times slower
  - **Idea:** Giving resources to users who actually need them
- What can go wrong?
  - One app can interfere with other app (need **isolation**)
  - Users are gluttons, use too much CPU, etc. (need **scheduling**)
  - Total memory usage of all apps/users greater than machine's RAM (need **memory management**)
  - Disks are shared across apps / users and must be arranged properly (need **file systems**)

Summary of OS: 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

Summary of OS: 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

OS provides Isolation using System Calls

- **System call:** The layer for isolation -- it abstracts the hardware and APIs for programs to use

	"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				

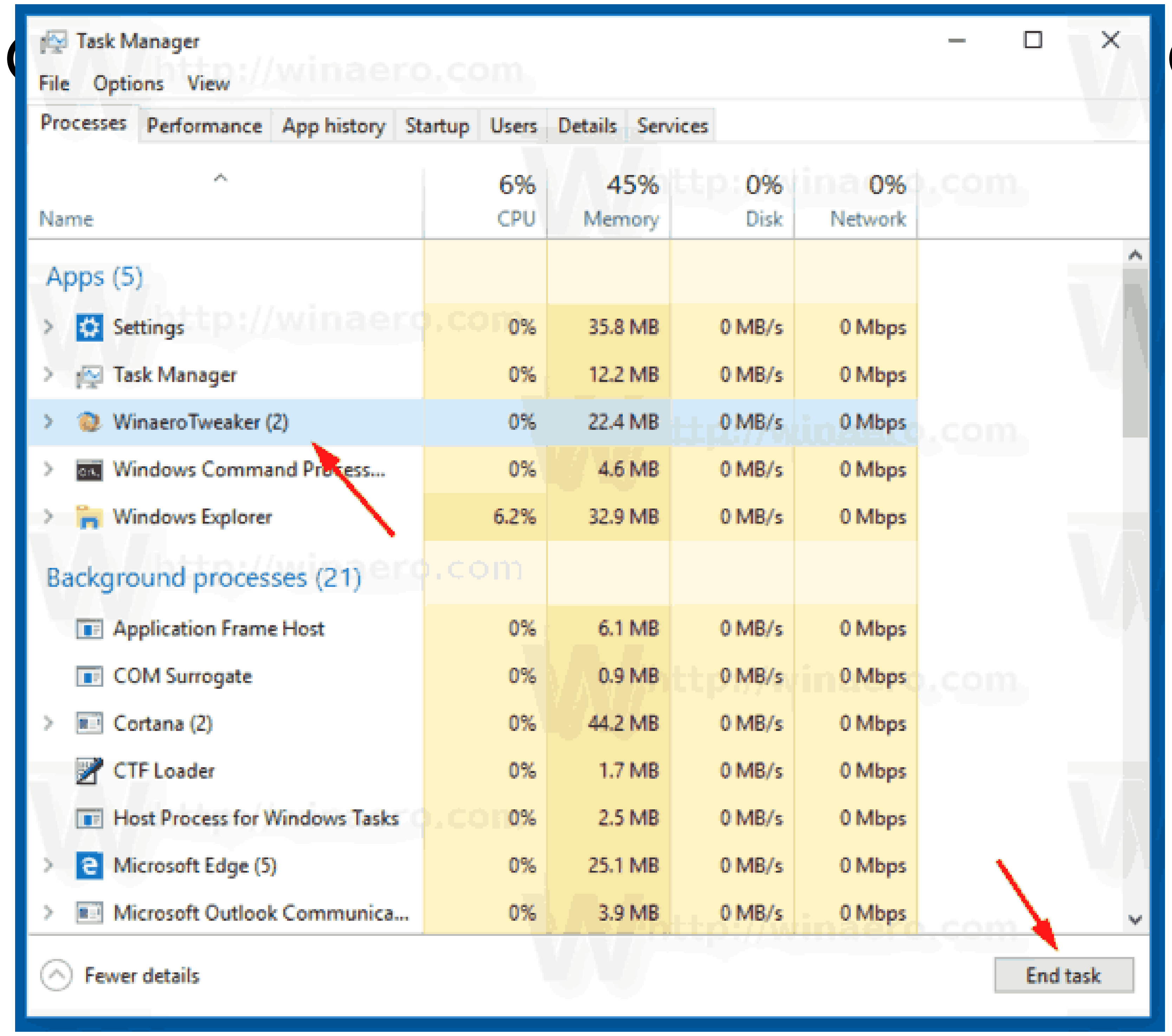
Hardware

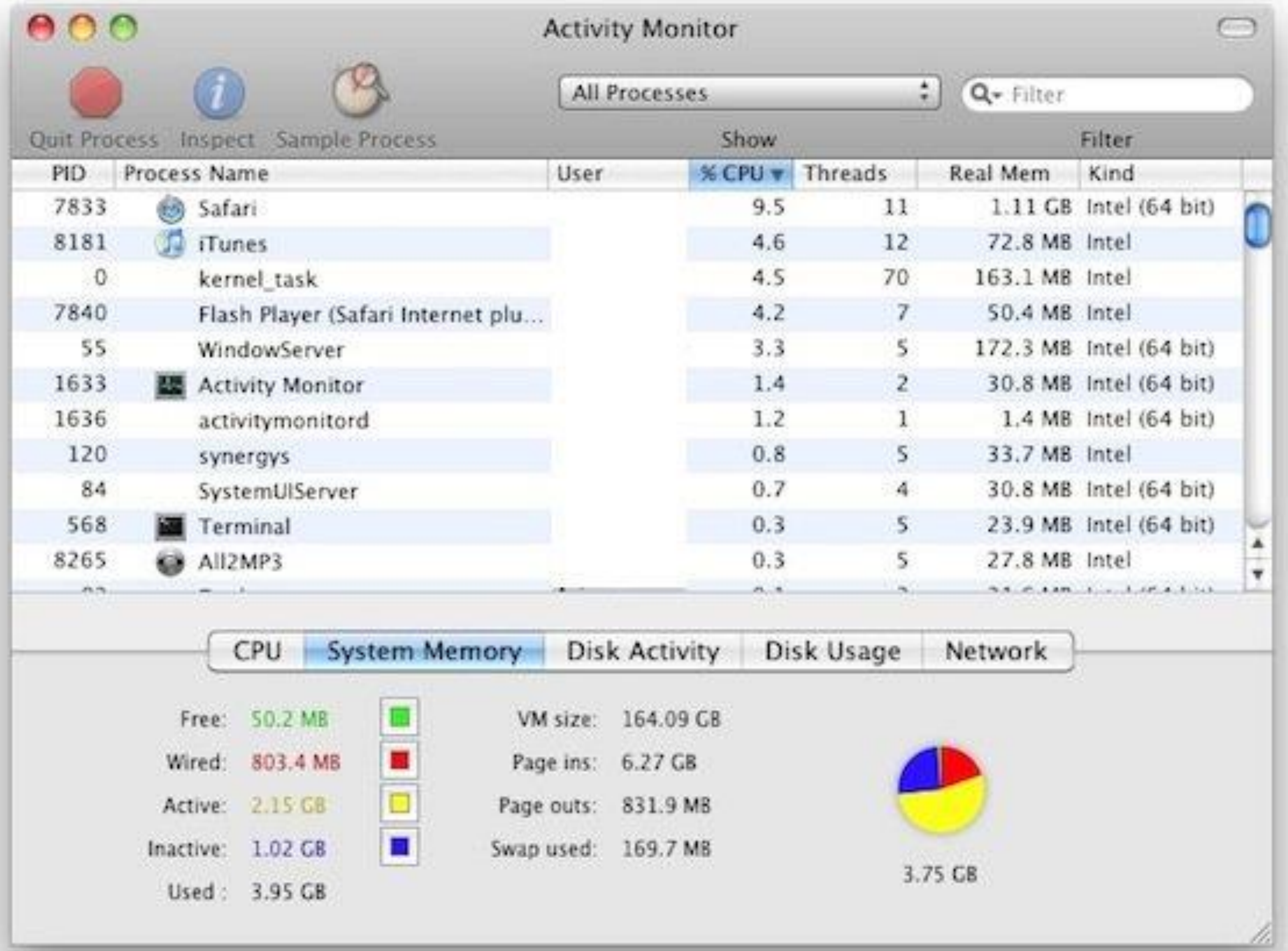
Foundation of Data Systems

- Computer Organization
  - Representation of data
  - processors, memory, storage
- OS basics
  - **Process, scheduling**
  - Memory

Processes - the central abstraction in OS

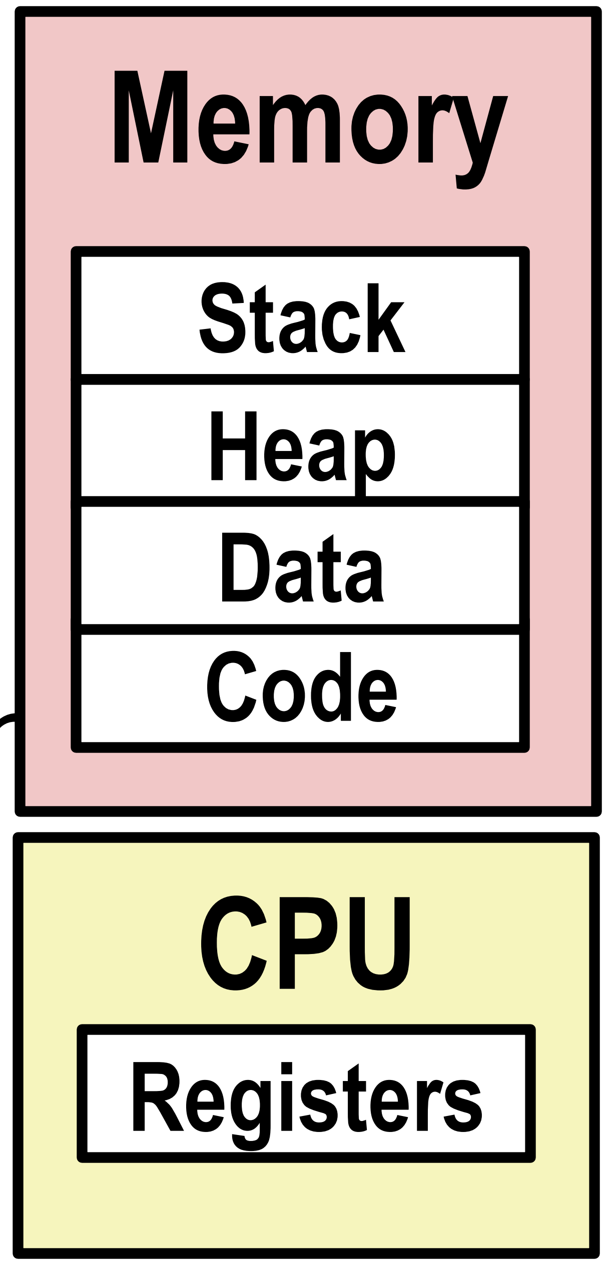
- Definition: A *process* is an instance of a running program.
  - One of the most profound ideas in computer science
  - N … or



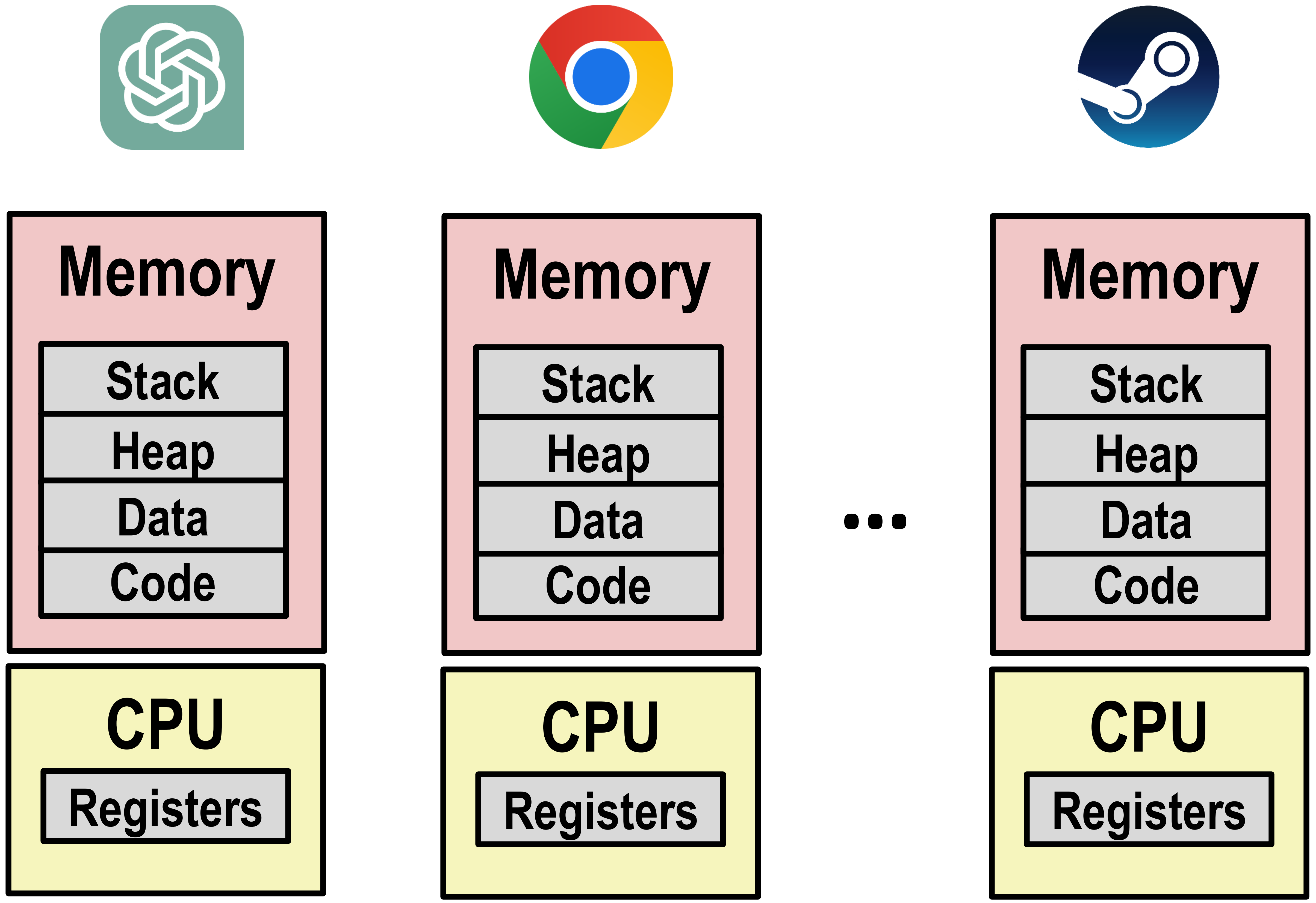


Processes - the central abstraction in OS

- Process provides each program with two key abstractions (for resources):
  - **Compute Resource**
    - Each program seems to have exclusive use of the CPU
    - Provided by kernel mechanism called *context switching*
  - **Memory Resource**
    - Each program seems to have exclusive use of main mem
    - Provided by kernel mechanism called virtual memory



Multiprocessing in OS: The Illusion



- Computer runs many processes simultaneously

Multiprocessing Example

top command in terminal: many processes, Identified by Process ID (**PID**)

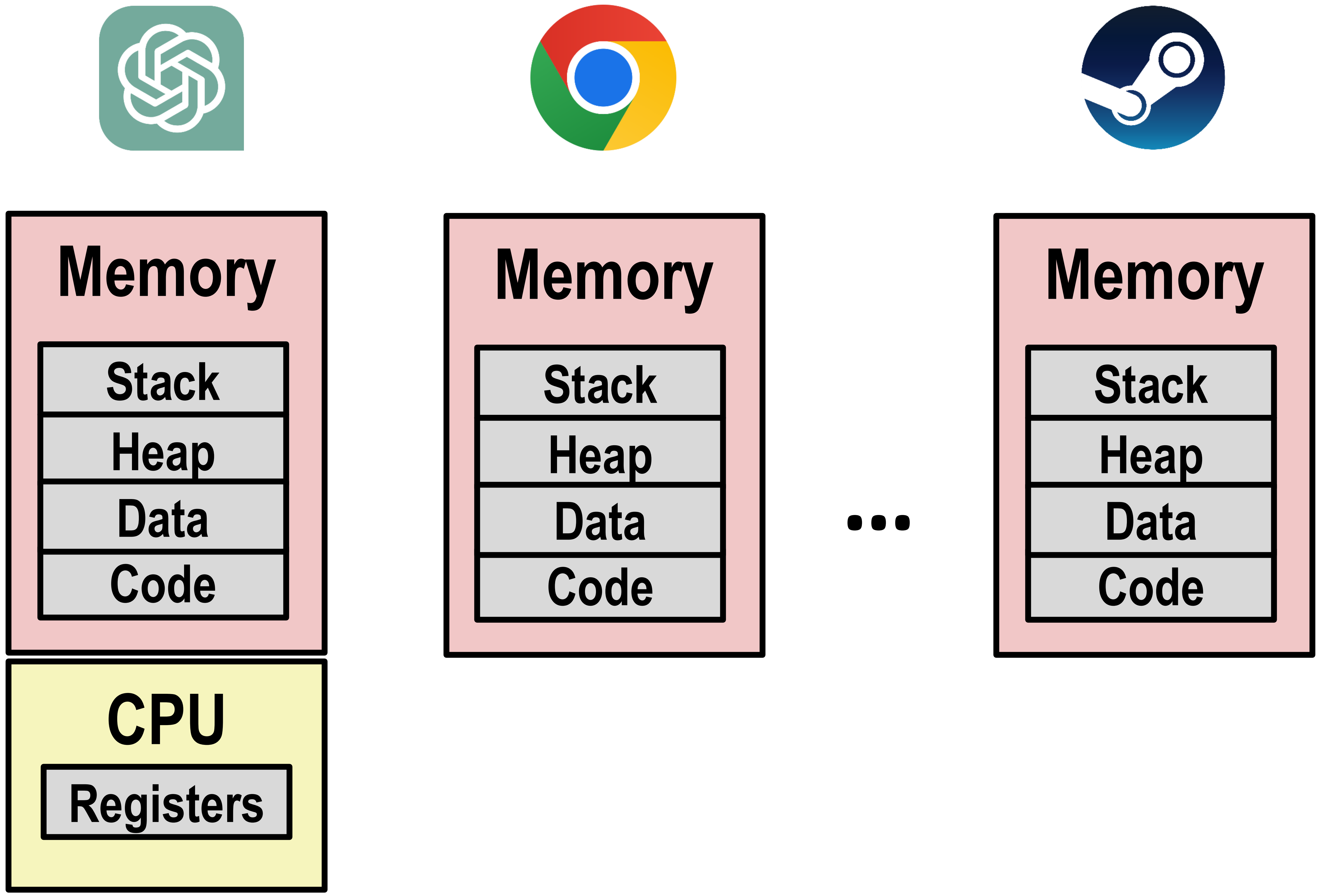
xterm

```
Processes: 123 total, 5 running, 9 stuck, 109 sleeping, 611 threads          11:47:07
Load Avg: 1.03, 1.13, 1.14  CPU usage: 3.27% user, 5.15% sys, 91.56% idle
SharedLibs: 576K resident, 0B data, 0B linkedit.
MemRegions: 27958 total, 1127M resident, 35M private, 494M shared.
PhysMem: 1039M wired, 1974M active, 1062M inactive, 4076M used, 18M free.
VM: 280G vsize, 1091M framework vsize, 23075213(1) pageins, 5843367(0) pageouts.
Networks: packets: 41046228/11G in, 66083096/77G out.
Disks: 17874391/349G read, 12847373/594G written.

PID    COMMAND      %CPU TIME     #TH   #WQ  #PORT #MREG RPRVT  RSHRD  RSIZE  VPRVT  VSIZE
99217- Microsoft Of 0.0  02:28.34 4     1    202   418   21M    24M    21M    66M    763M
99051  usbmuxd      0.0  00:04.10 3     1    47    66    436K   216K   480K   60M    2422M
99006  iTunesHelper 0.0  00:01.23 2     1    55    78    728K   3124K  1124K  43M    2429M
84286  bash         0.0  00:00.11 1     0    20    24    224K   732K   484K   17M    2378M
84285  xterm        0.0  00:00.83 1     0    32    73    656K   872K   692K   9728K  2382M
55939- Microsoft Ex 0.3  21:58.97 10    3    360   954   16M    65M    46M    114M   1057M
54751  sleep        0.0  00:00.00 1     0    17    20    92K    212K   360K   9632K  2370M
54739  launchdadd   0.0  00:00.00 2     1    33    50    488K   220K   1736K  48M    2409M
54737  top          6.5  00:02.53 1/1   0    30    29    1416K  216K   2124K  17M    2378M
54719  automountd   0.0  00:00.02 7     1    53    64    860K   216K   2184K  53M    2413M
54701  ocspd        0.0  00:00.05 4     1    61    54    1268K  2644K  3132K  50M    2426M
54661  Grab         0.6  00:02.75 6     3    222+  389+  15M+   26M+   40M+   75M+   2556M+
54659  cookied      0.0  00:00.15 2     1    40    61    3316K  224K   4088K  42M    2411M
53818  mdworker     0.0  00:01.67 4     1    52    91    7628K  7412K  16M    48M    2438M
50878  mdworker     0.0  00:11.17 3     1    53    91    2464K  6148K  9976K  44M    2434M
50410  xterm        0.0  00:00.13 1     0    32    73    280K   872K   532K   9700K  2382M
50078  emacs        0.0  00:06.70 1     0    20    35    52K    216K   88K    18M    2392M
```

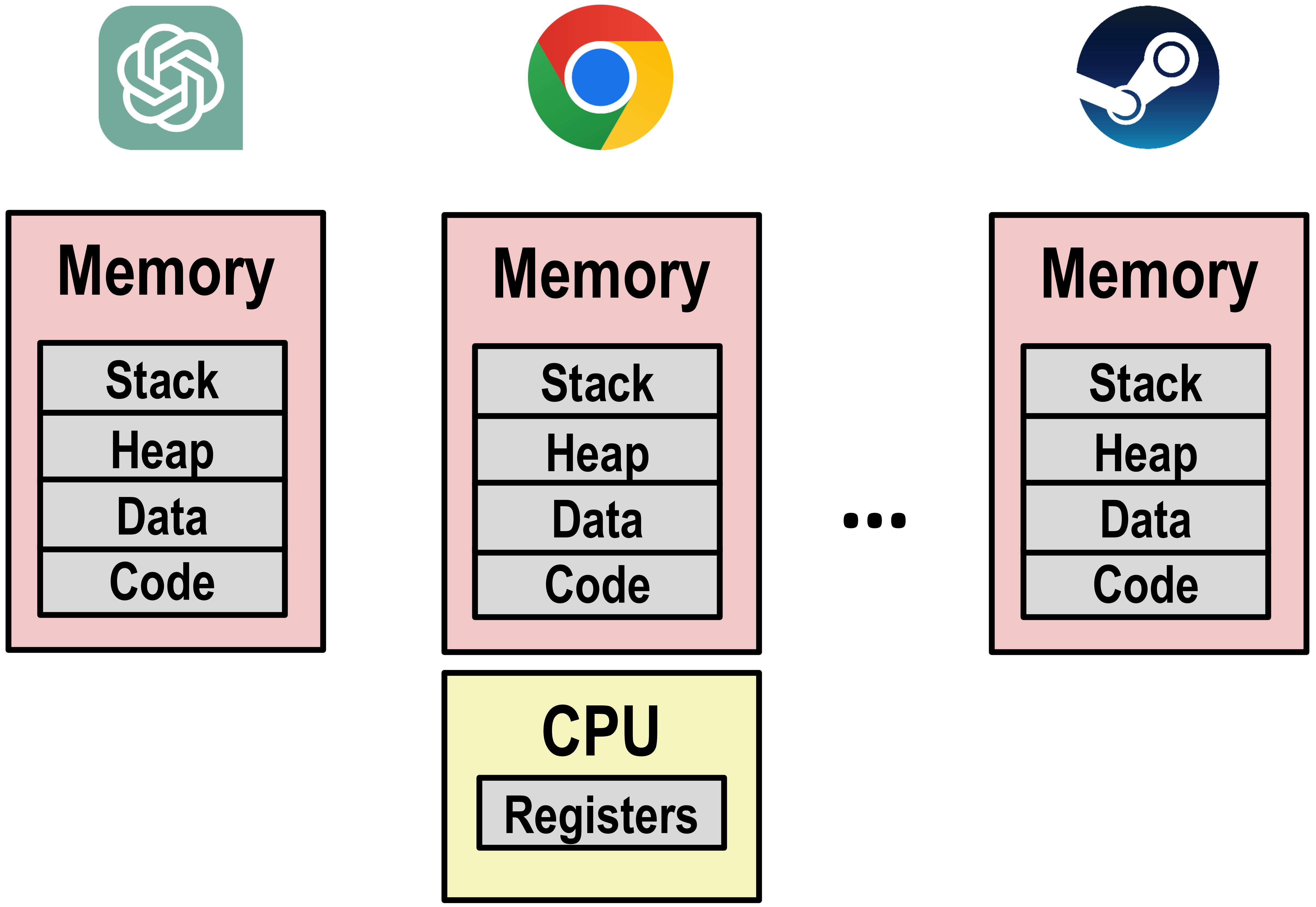
Multiprocessing: A strawman solution

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



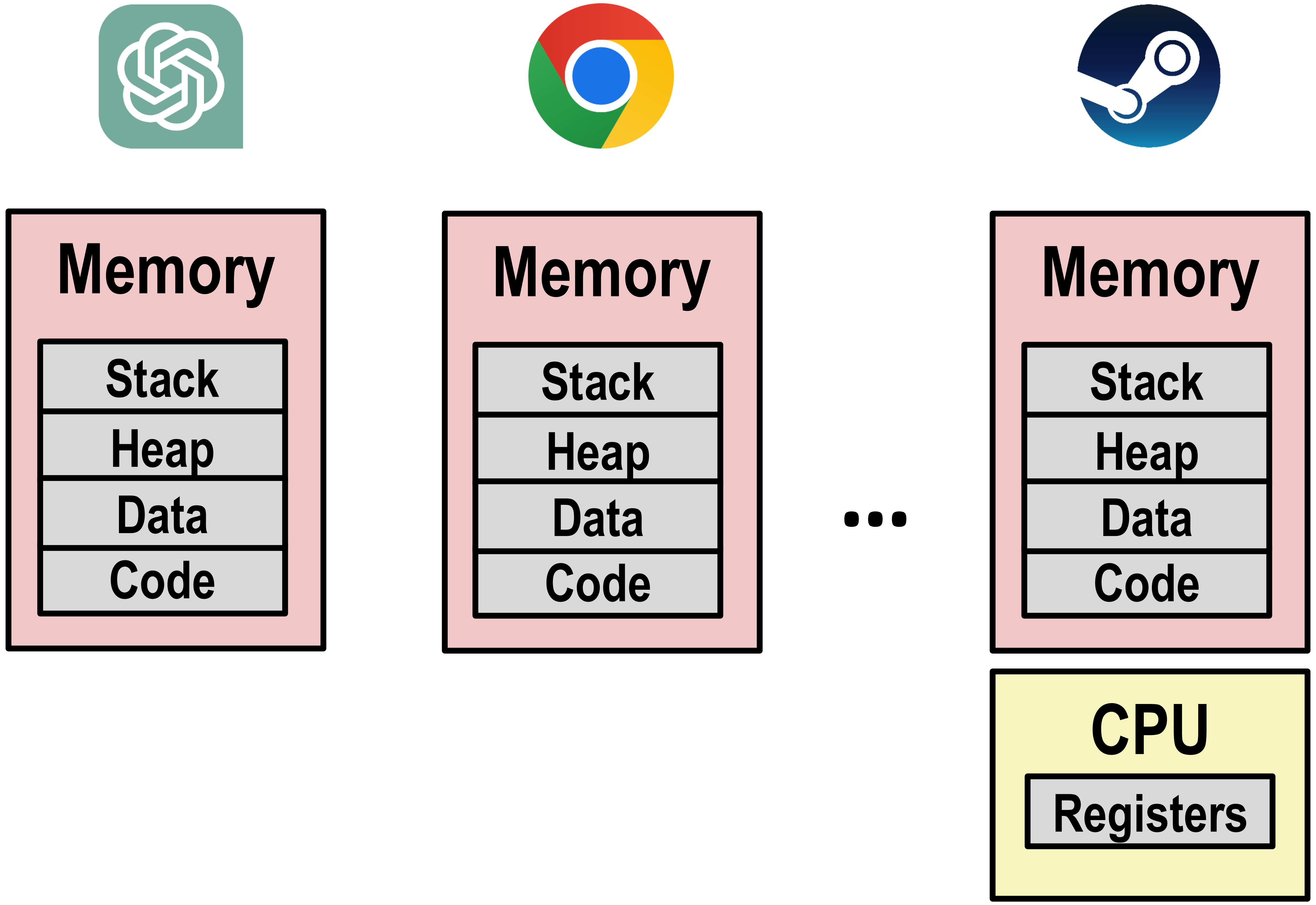
Multiprocessing: A strawman solution

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



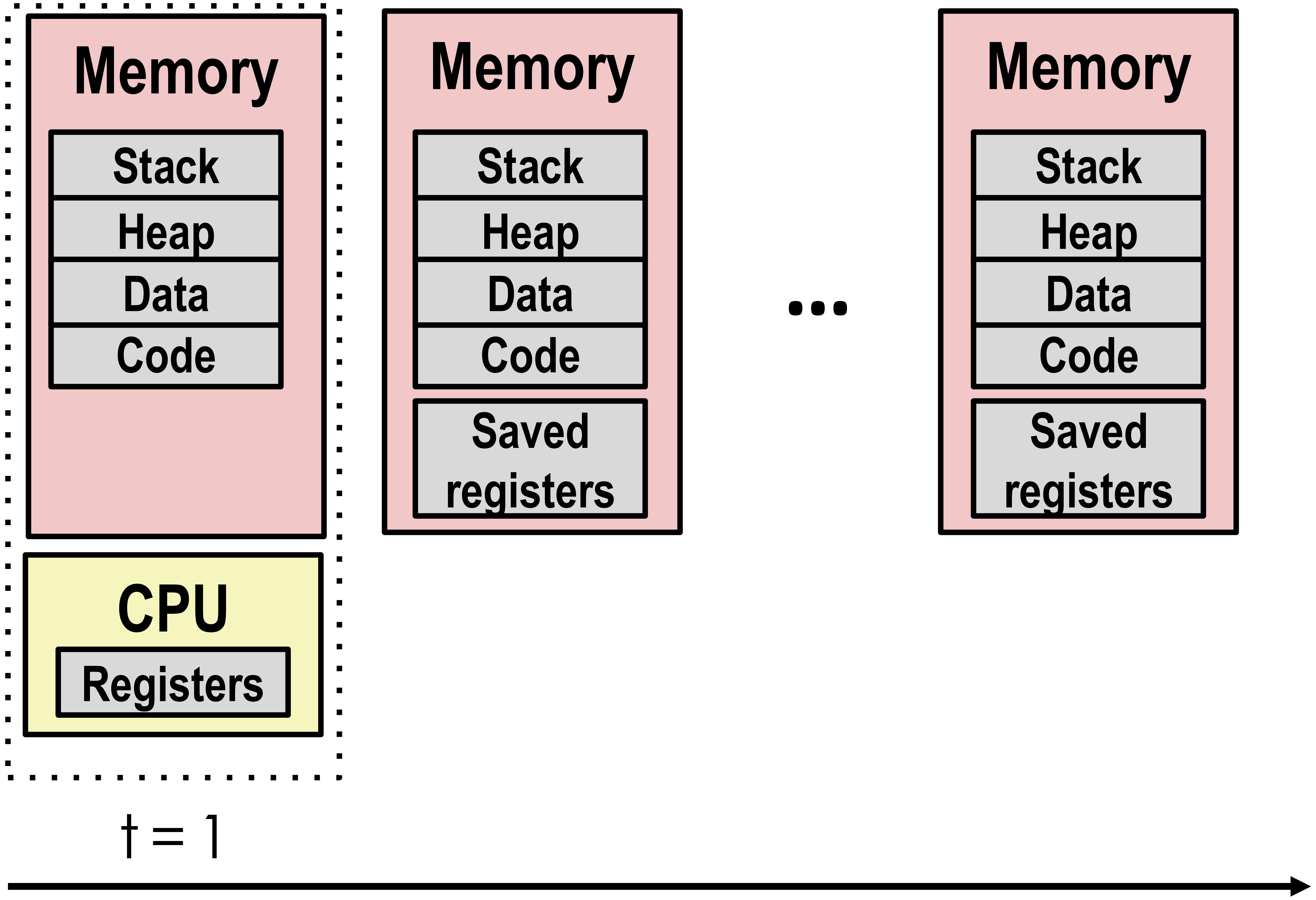
Multiprocessing: A strawman solution

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



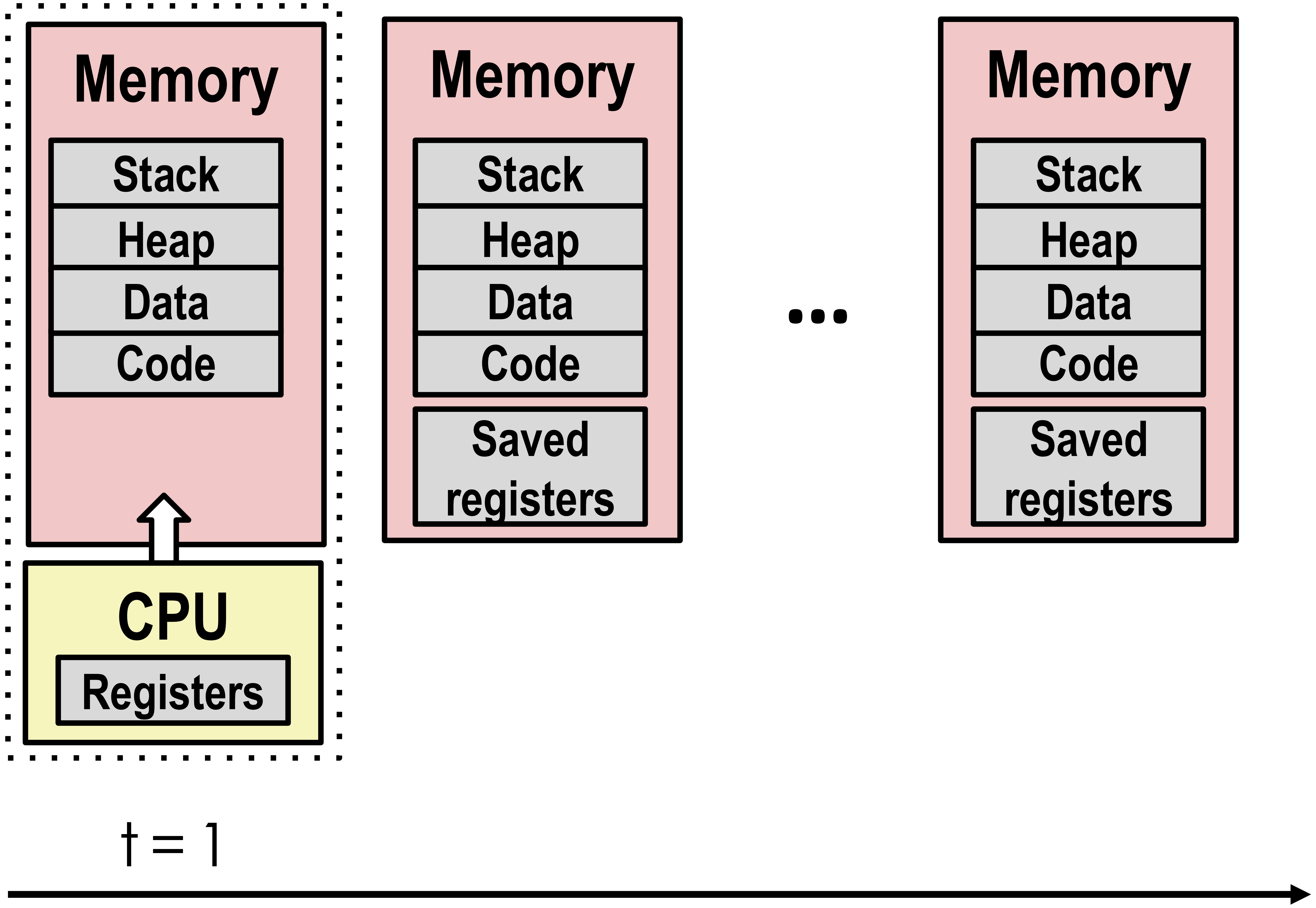
G1. Convenient?
G3: protection?
G2. Efficient?
!!!we are N times slower when running N processes

Multiprocessing: Time sharing of processors



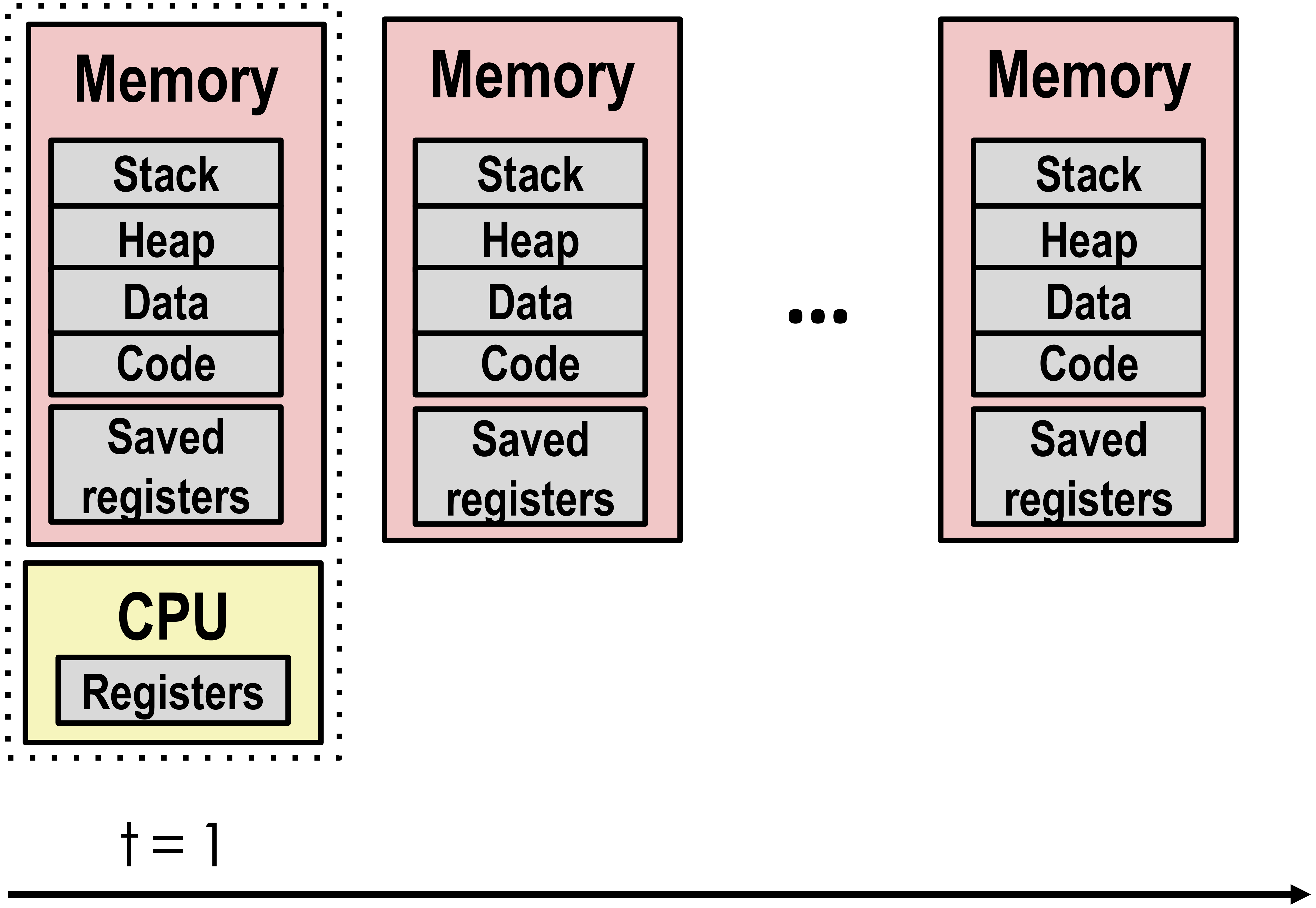
- Idea: Virtualize the CPU time as time slices
- Assign time slices to different processes

Multiprocessing: Time sharing of processors



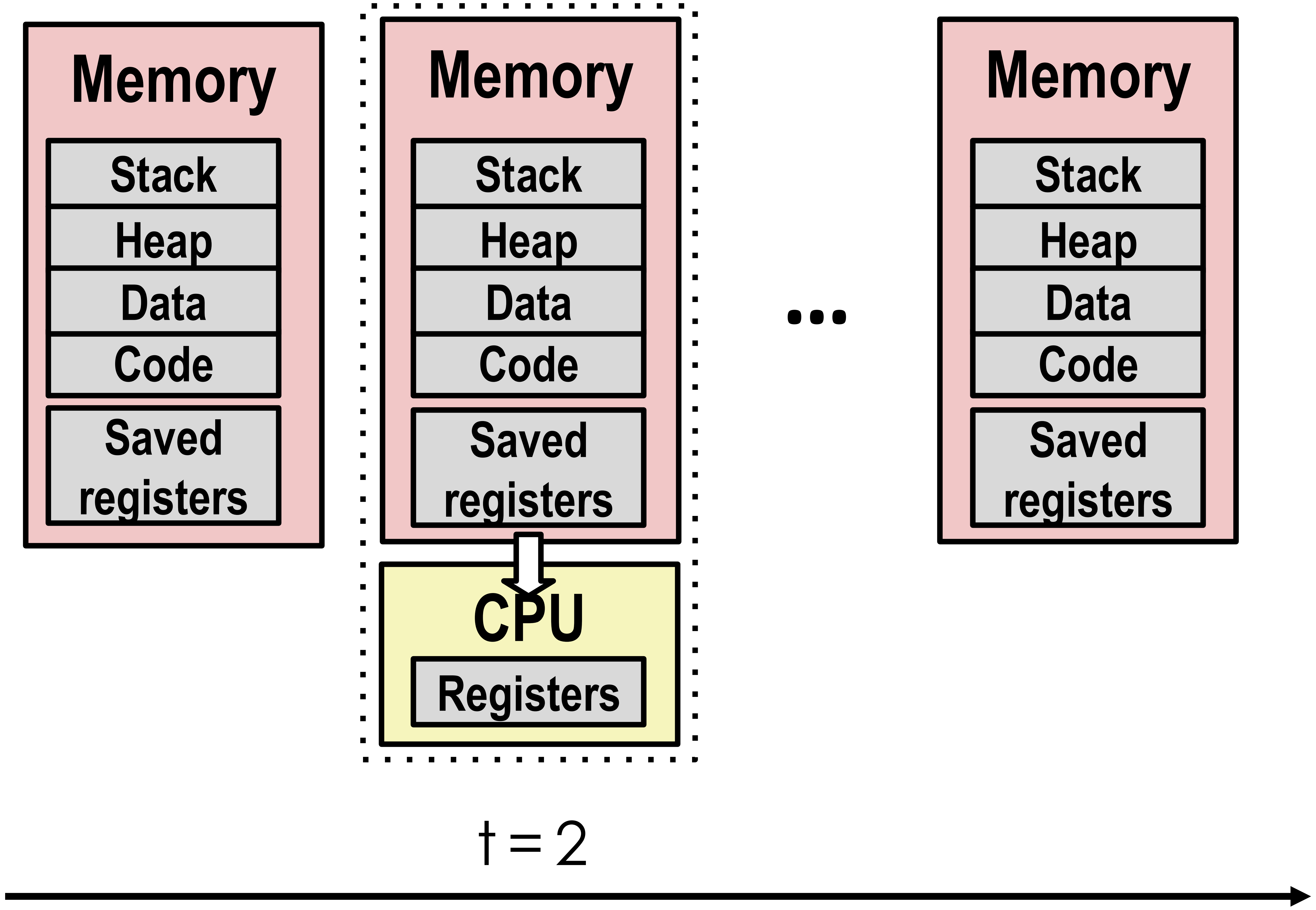
- Save current registers in memory

Multiprocessing: Time sharing of processors



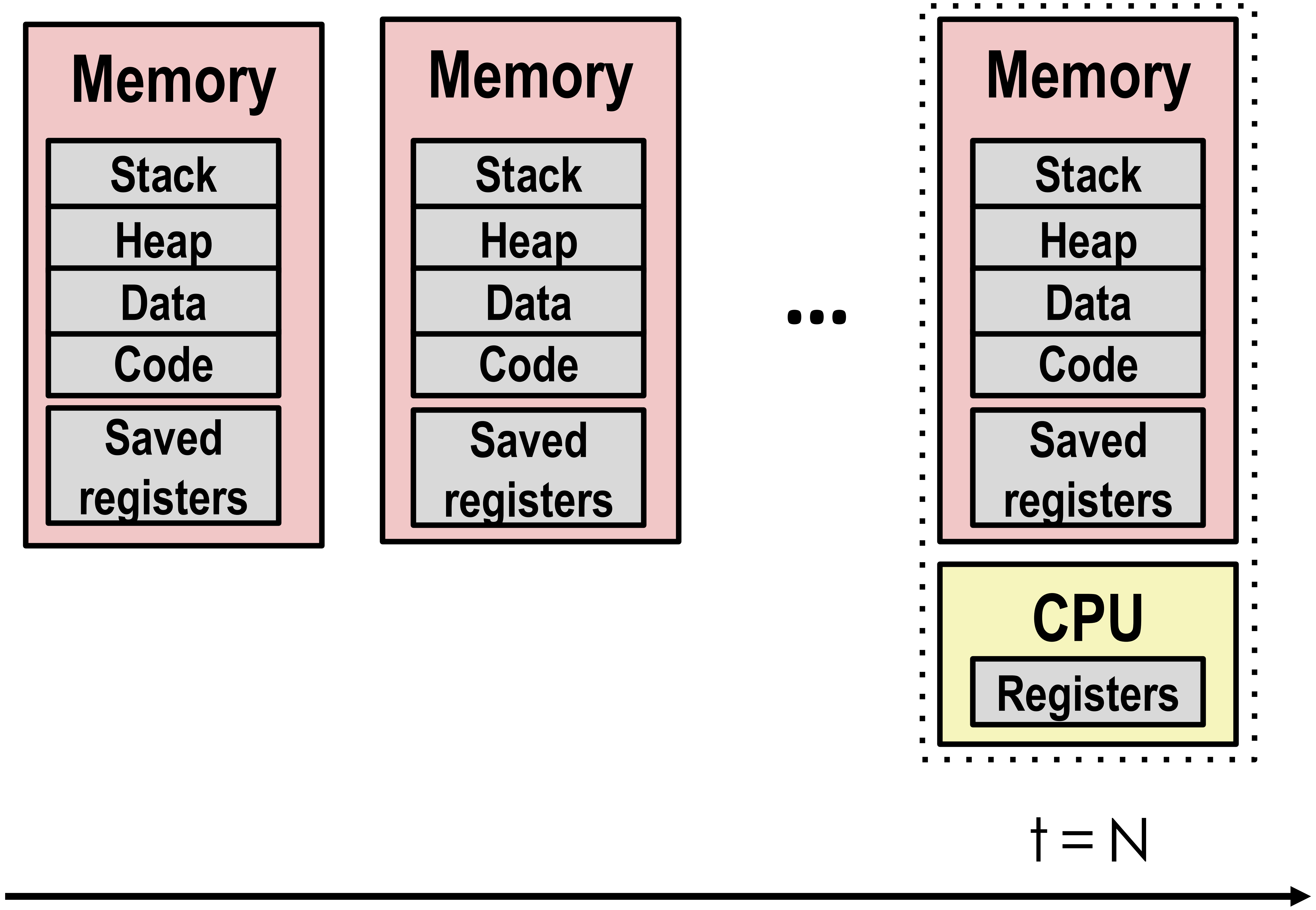
- Save current registers in memory

Multiprocessing: Time sharing of processors



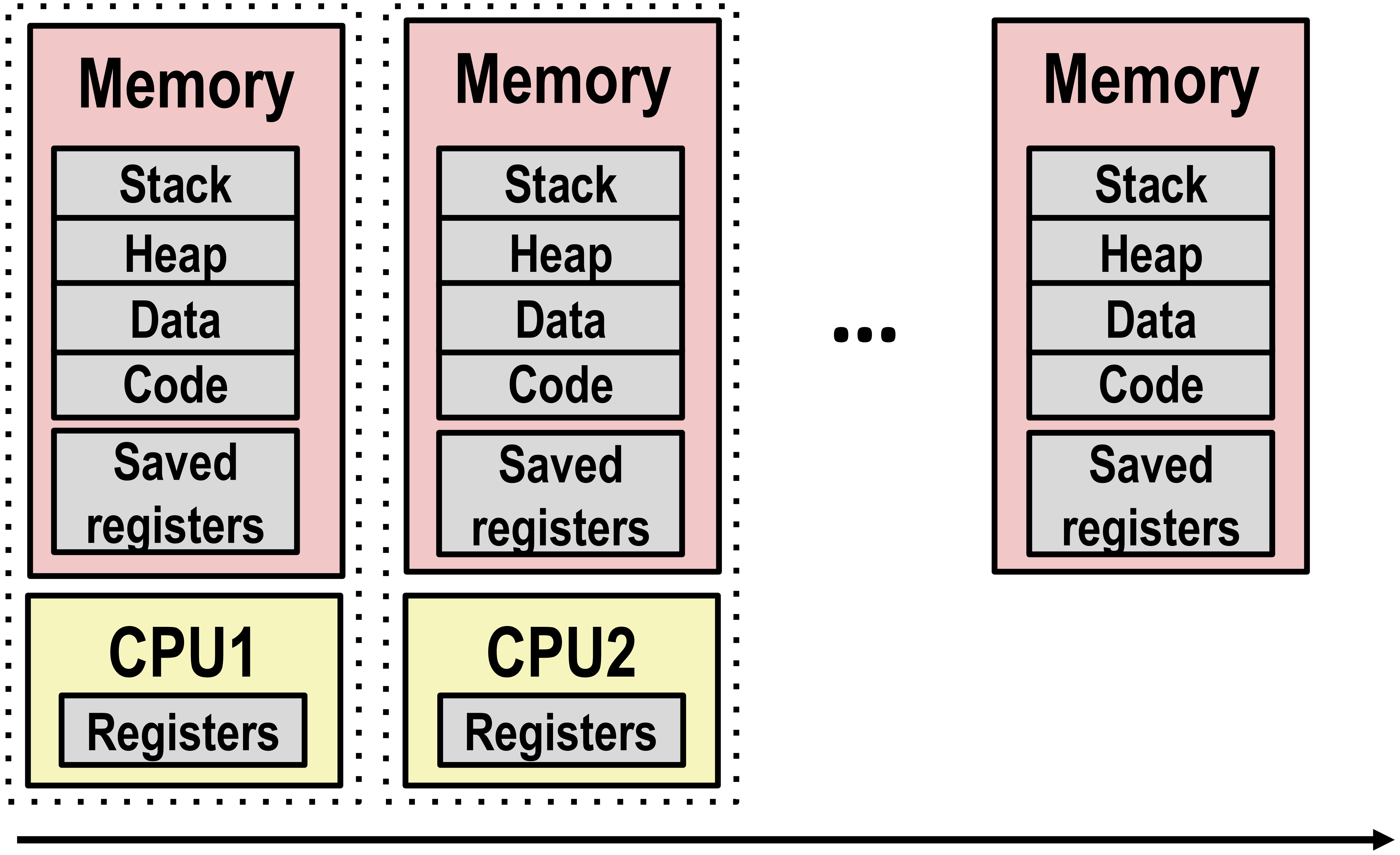
- Assign time slice t = 2 to the next process
- Resume progress: Move Saved registers from memory to CPU

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!

PID1 PID2 PID3 …

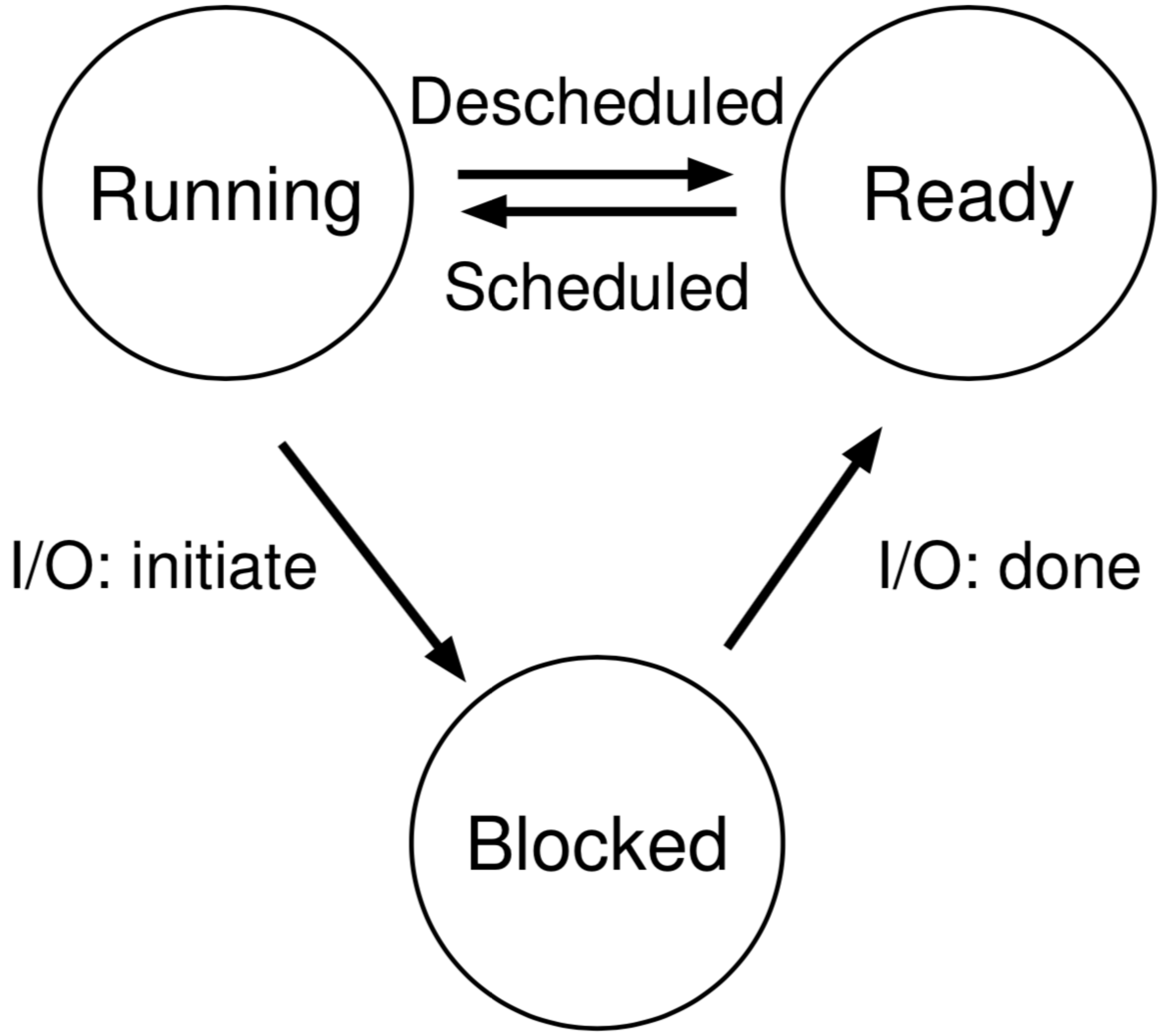
OS’s virtualized CPU abstraction

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

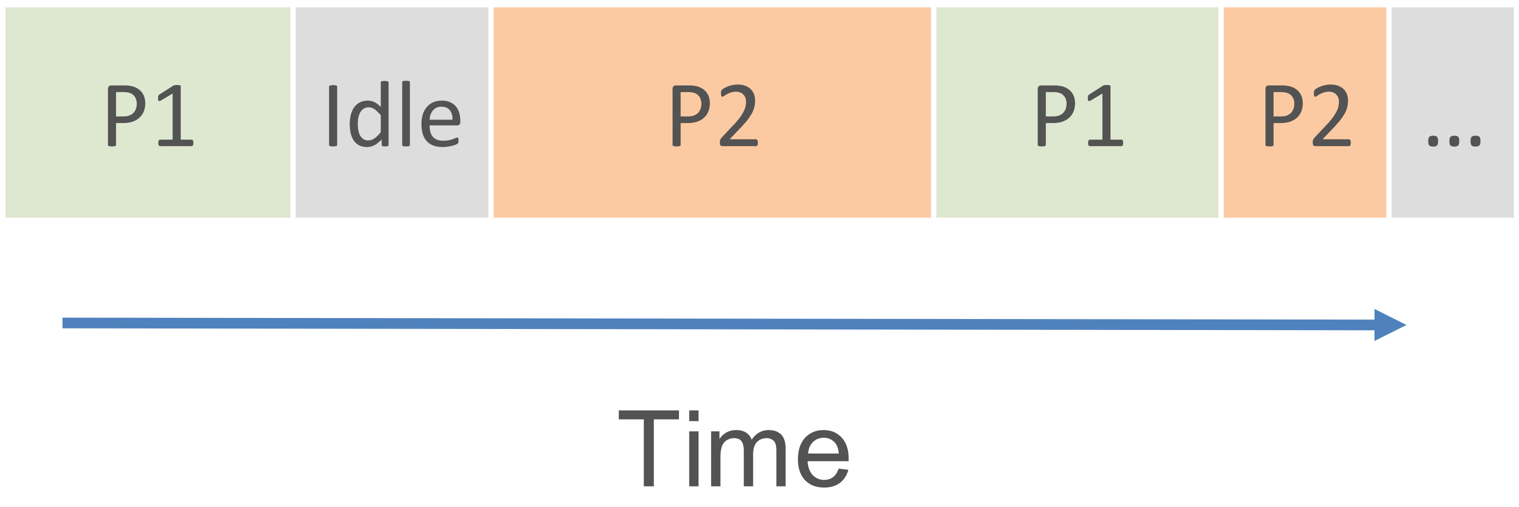
Physical Processor

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?

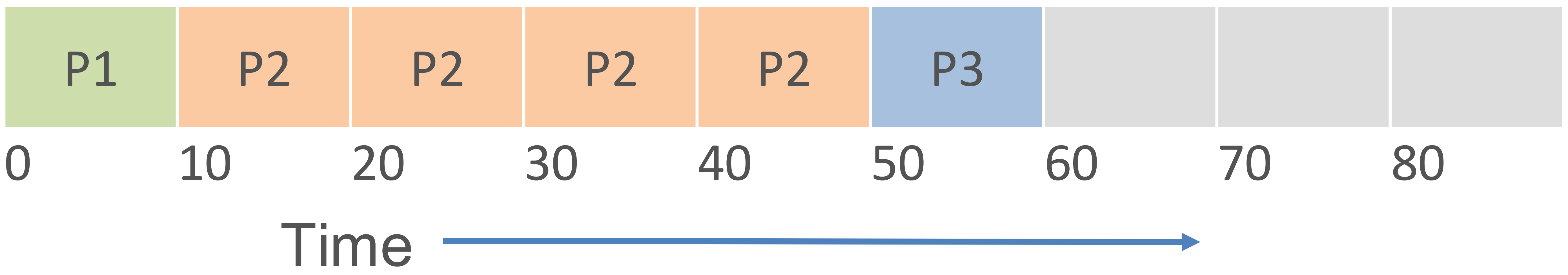
Scheduling Policies/Algorithms

- 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



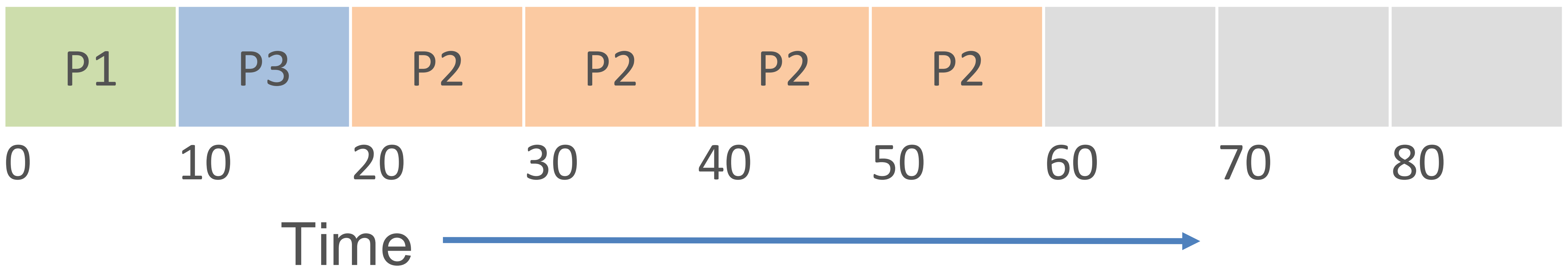
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	10	50	10	50
P3	0	50	60	50	60
			Avg:	20	40

- 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



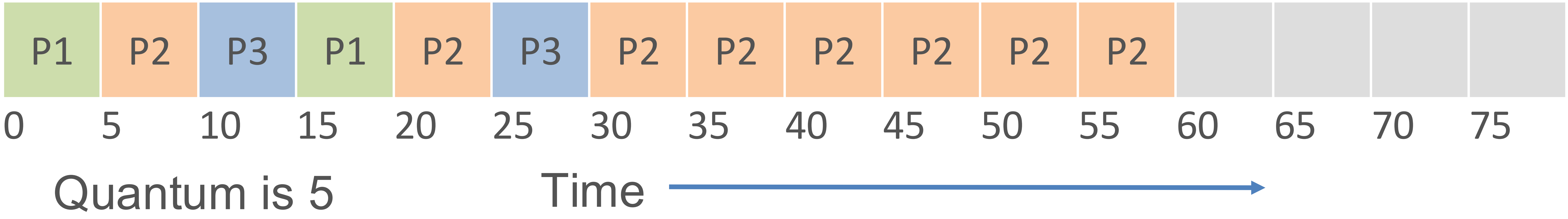
Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	20	60	20	60
P3	0	10	20	10	20
	(FIFO Avg: 20 and 40)		Avg:	10	30

- 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

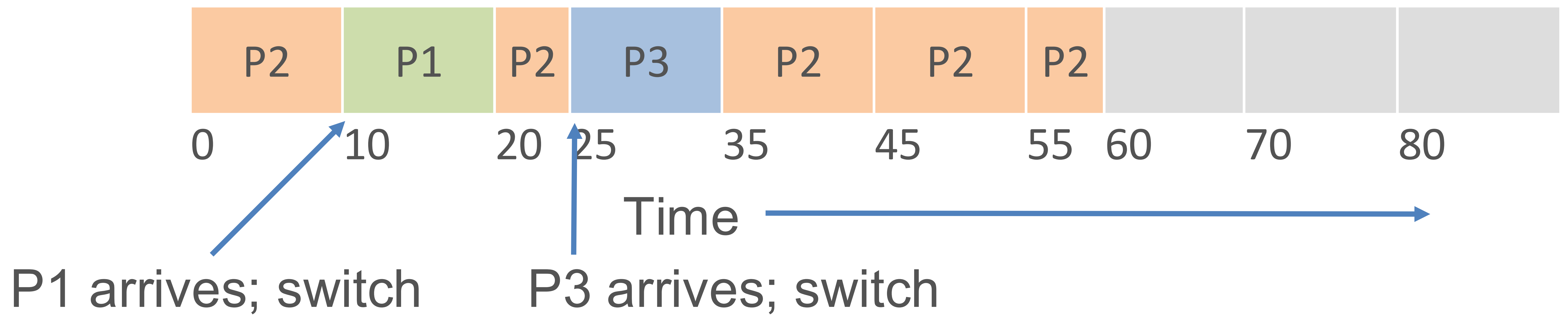


- RR is often very fair, but Avg Turnaround Time goes up!

Example Exam Q2: SCTF

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

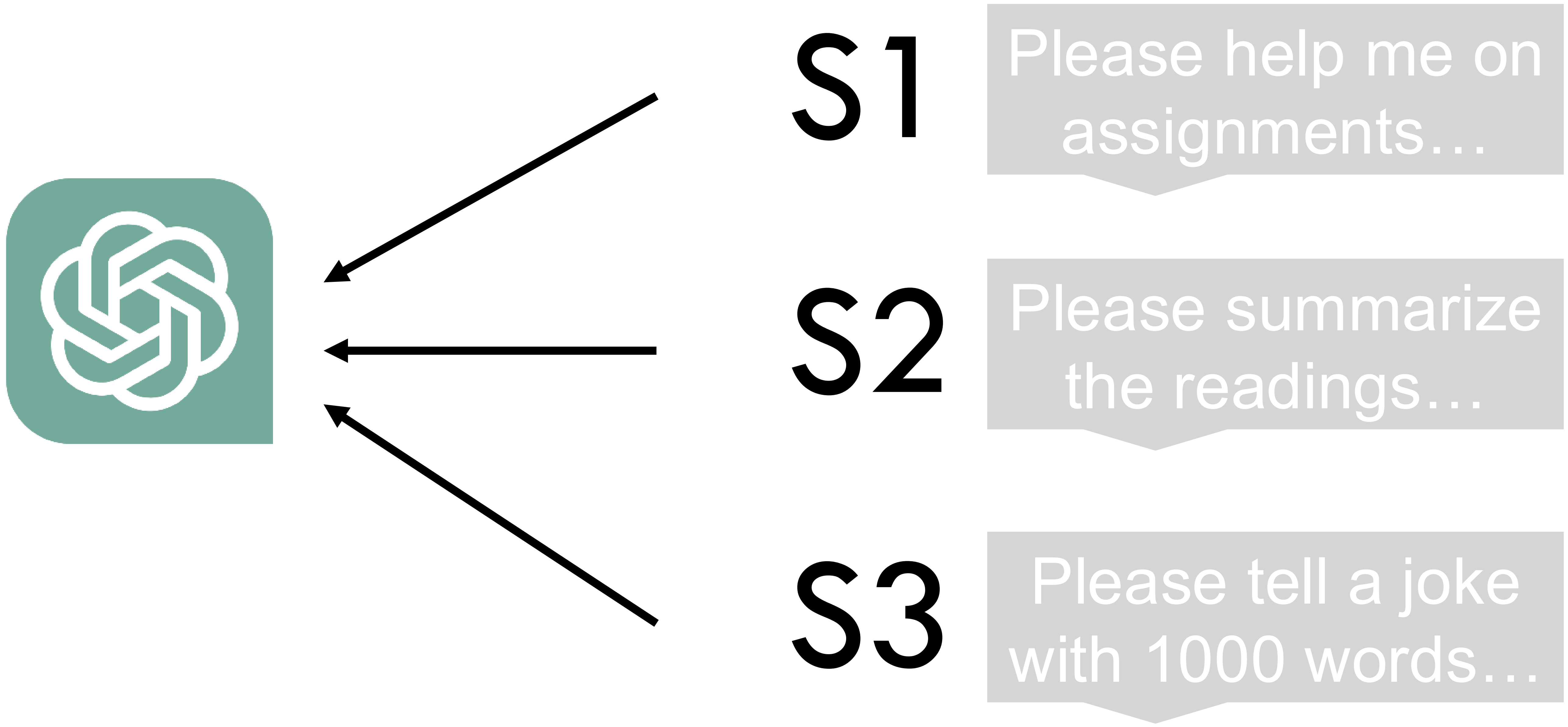
Example: P1, P2, P3 of lengths 10,40,10 units arrive at different times



Scheduling Policies/Algorithms

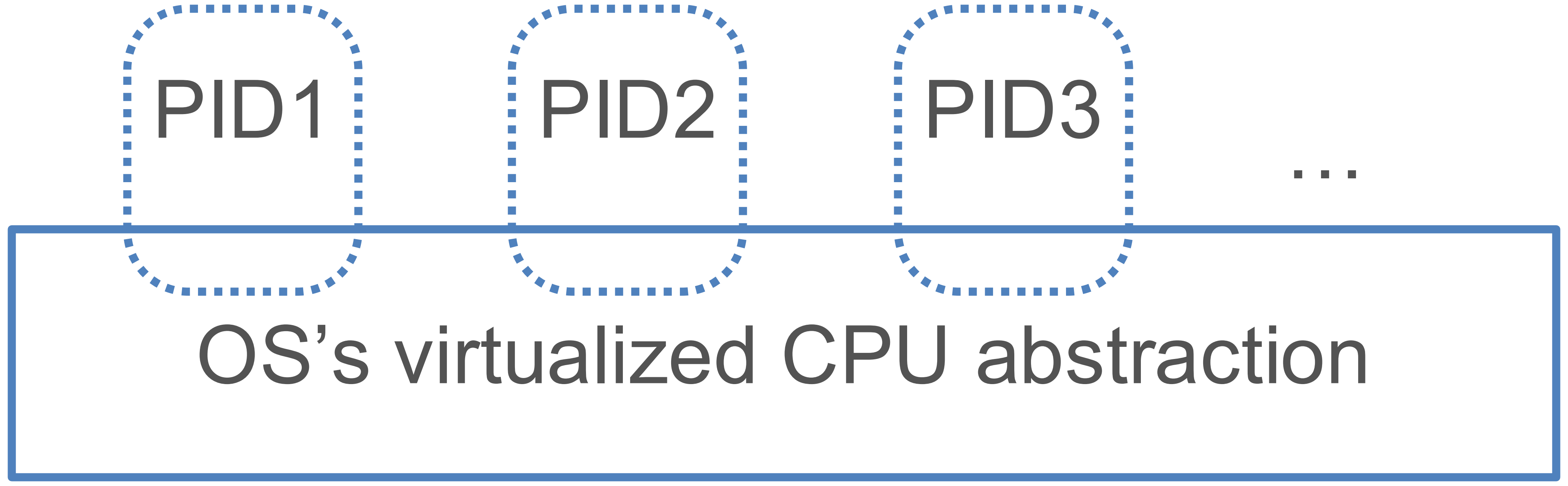
- In general, not all Arrival Times and Job Lengths will be known beforehand. But preemption is possible.
- Key Principle: Inherent tension in scheduling between overall workload *performance* and allocation *fairness*
  - Performance metric is usually *Average Turnaround Time*
  - Many fairness metrics 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 together?
- How to schedule?

Let’s Implement It!

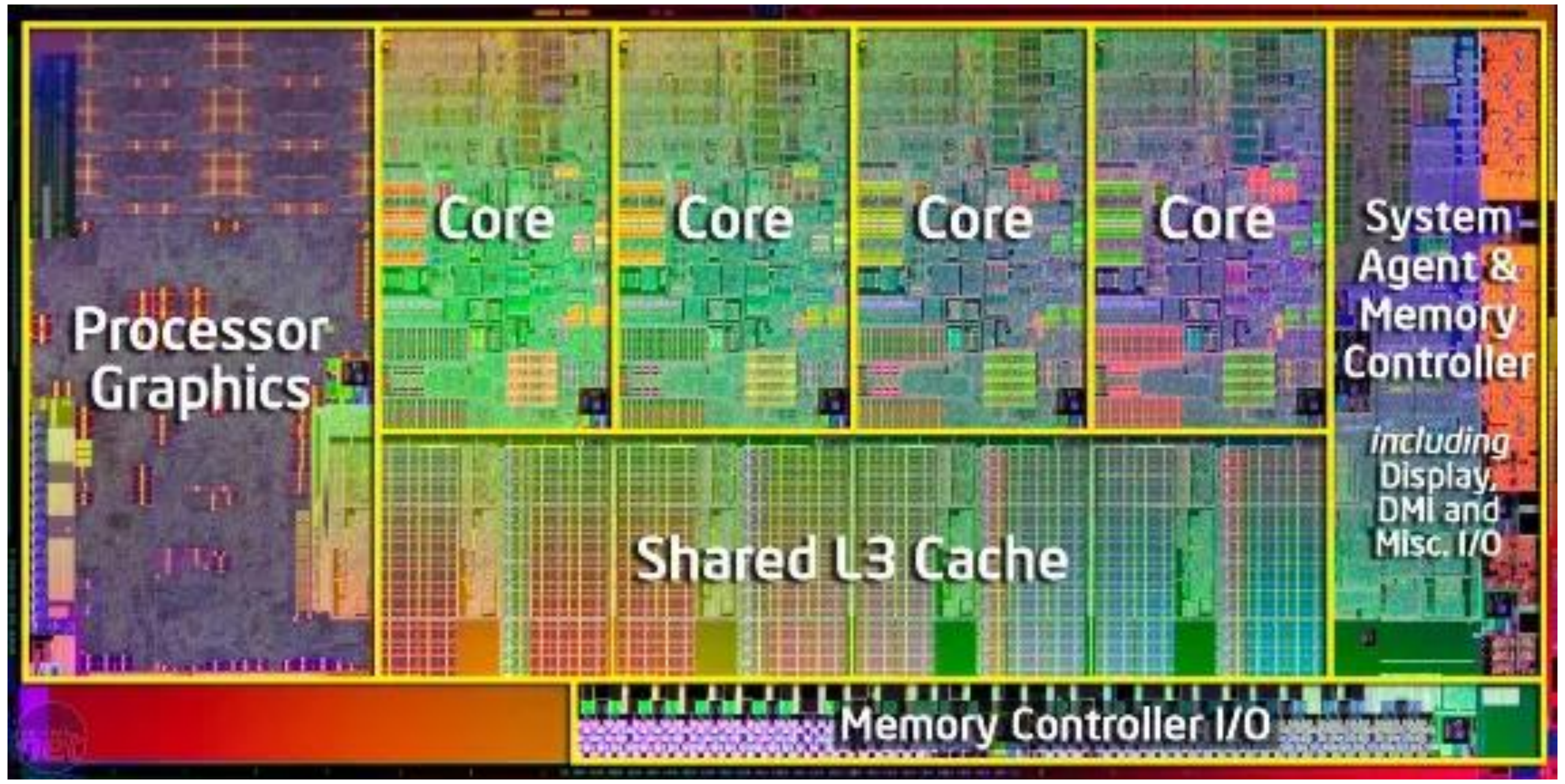


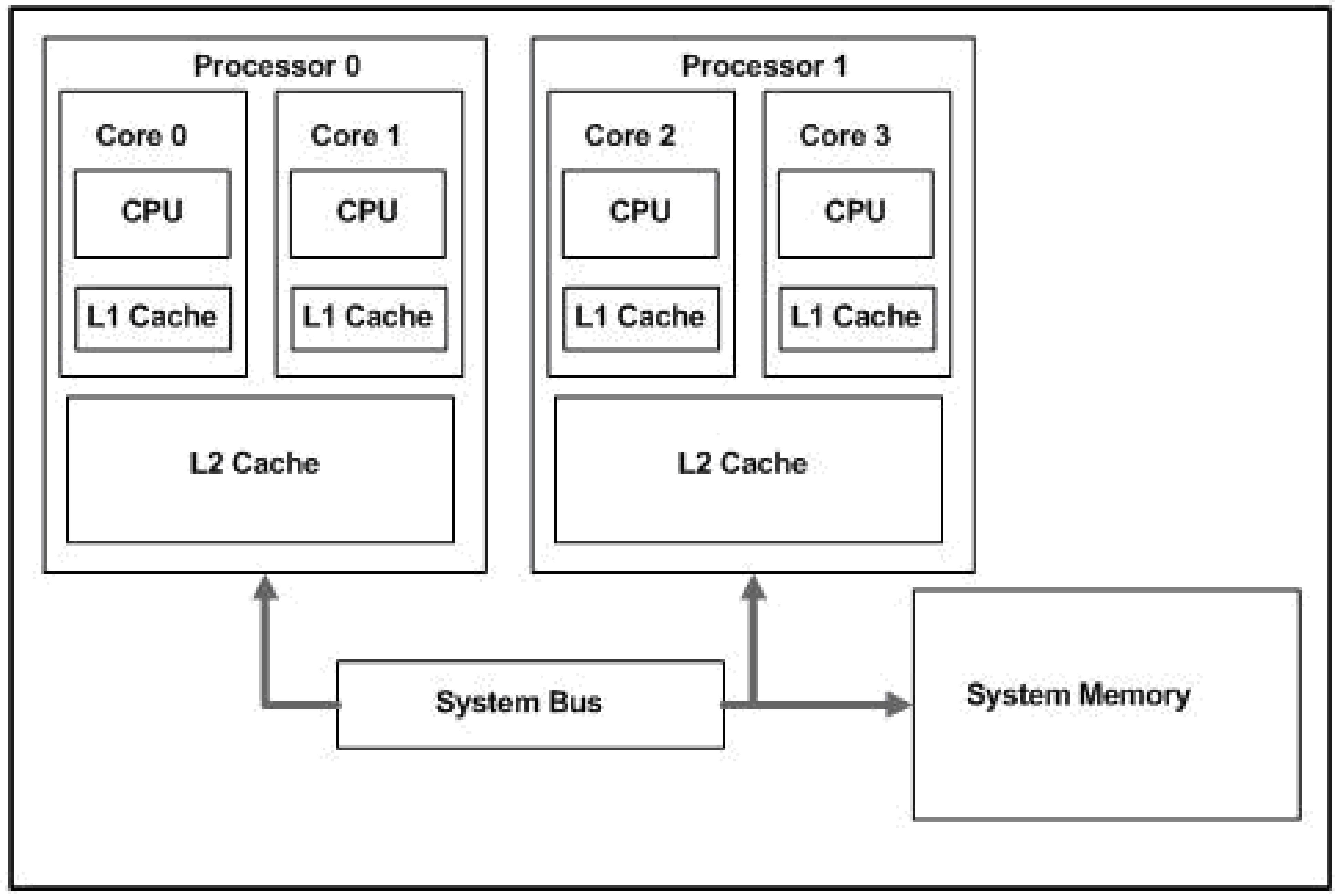
GAP2: How to virtualize CPU resources temporally and **spatially**?

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!

PID1 PID2 PID3 …

OS’s virtualized CPU abstraction

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?

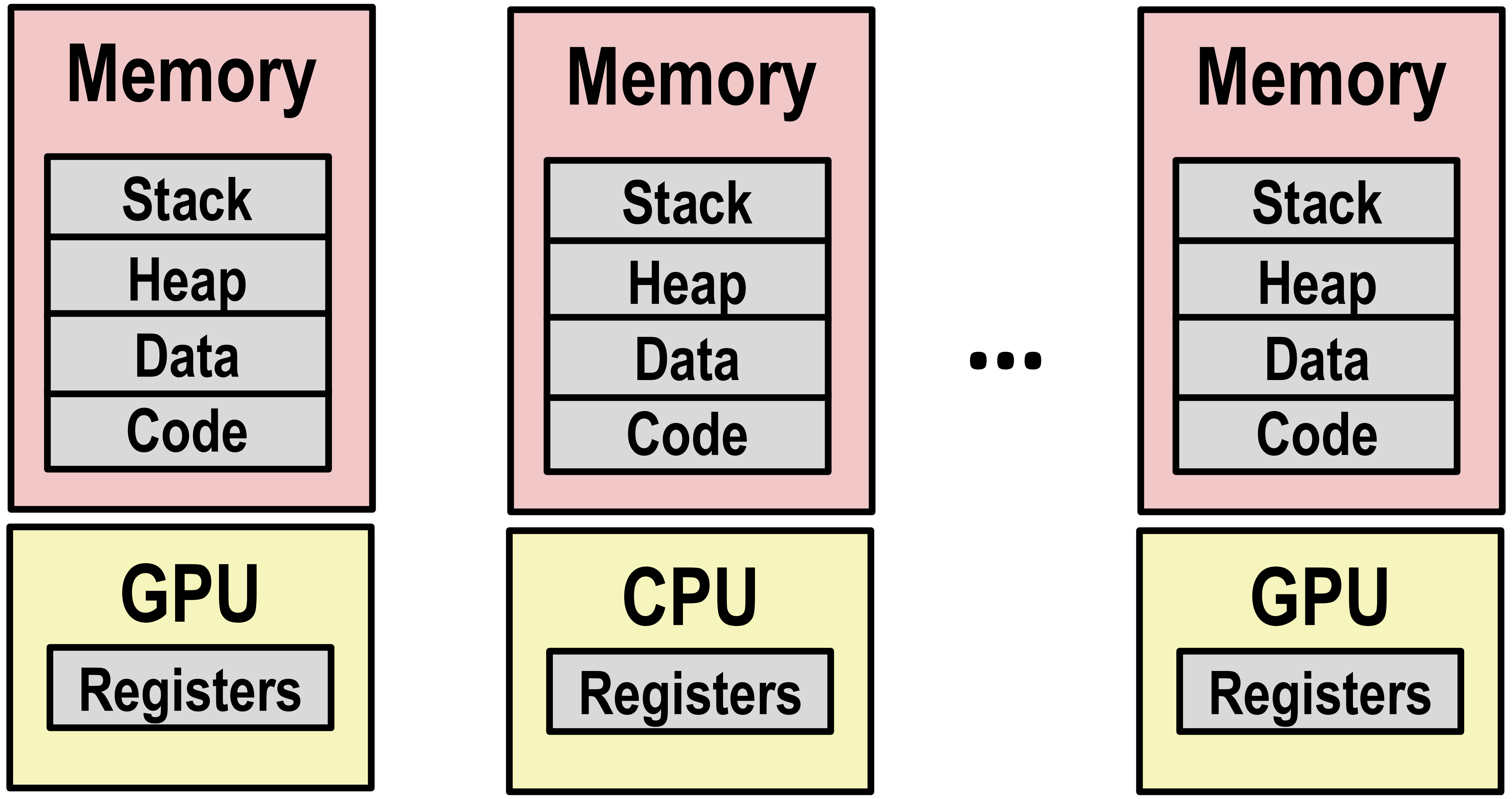
Concurrency

- 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

Mutliprocessing: memory management

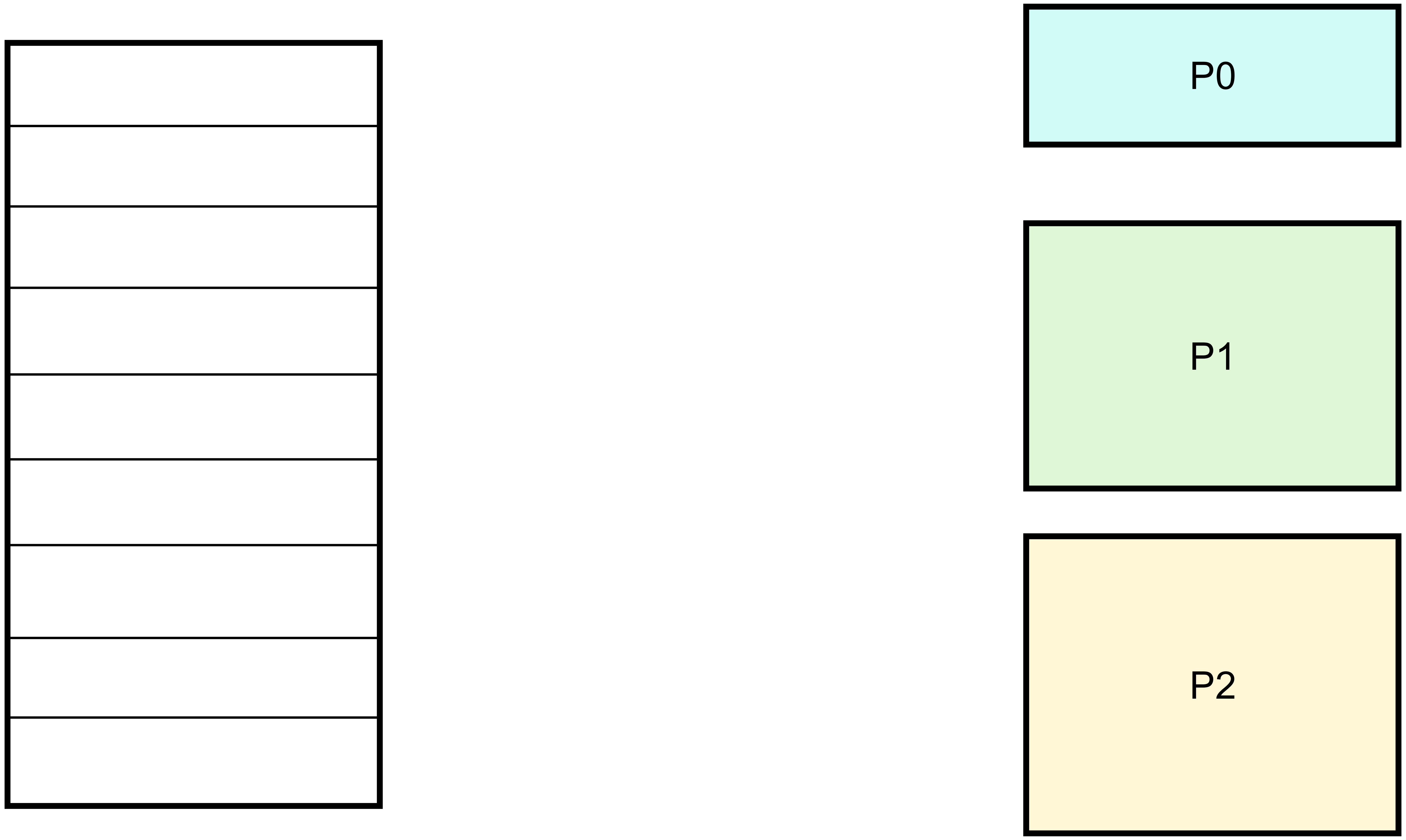
- ~~Strawman solution~~ **-> spatial-temporal sharing of CPUs with scheduling**
- Assign 1/3 of the memory to each APP



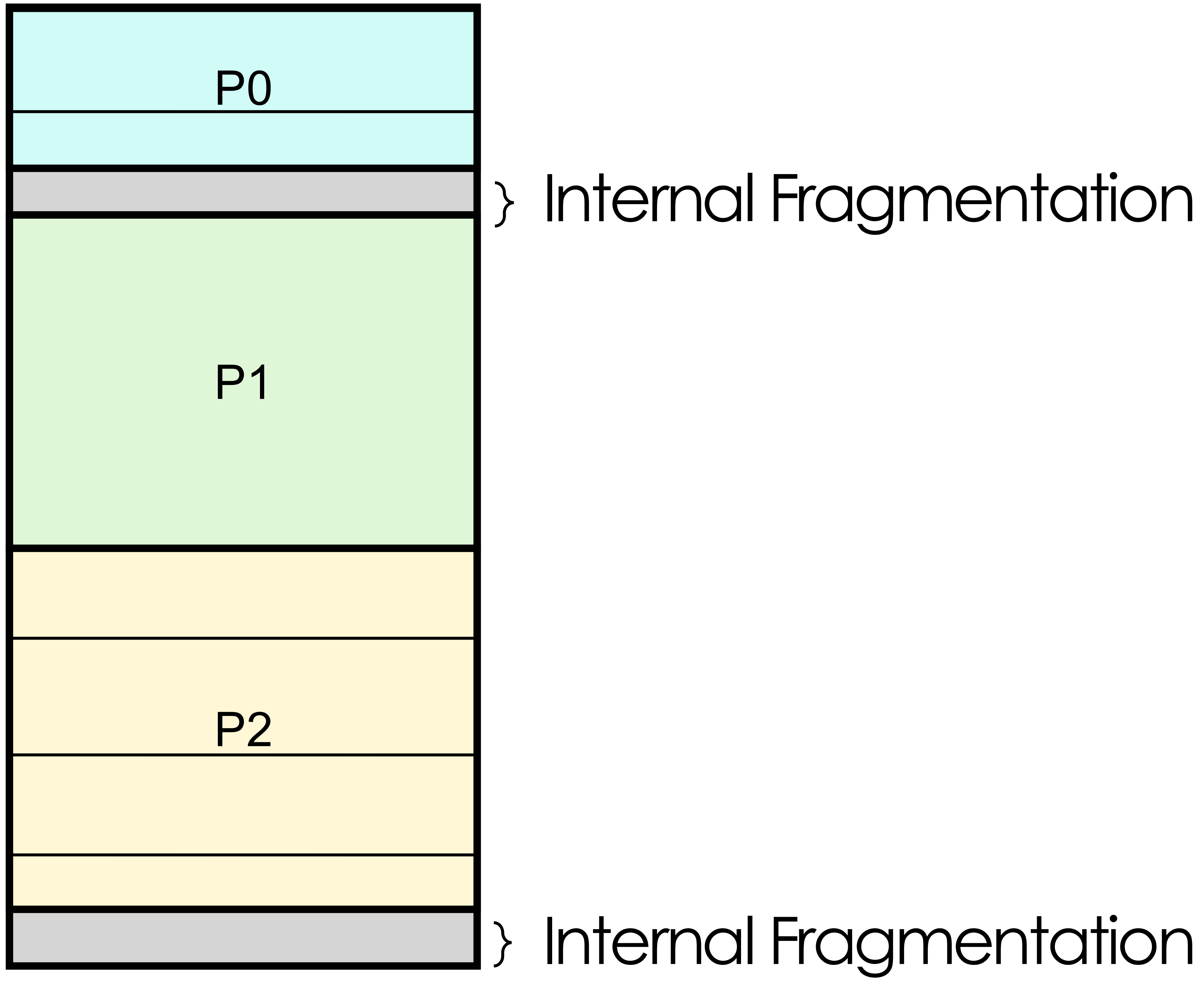
G1. Convenient?
G3: protection?
G2. Efficient?

- G2.1 can I run N processes but not N times slower?
- **G2.2 can I run N apps with total mem > physical memory cap**

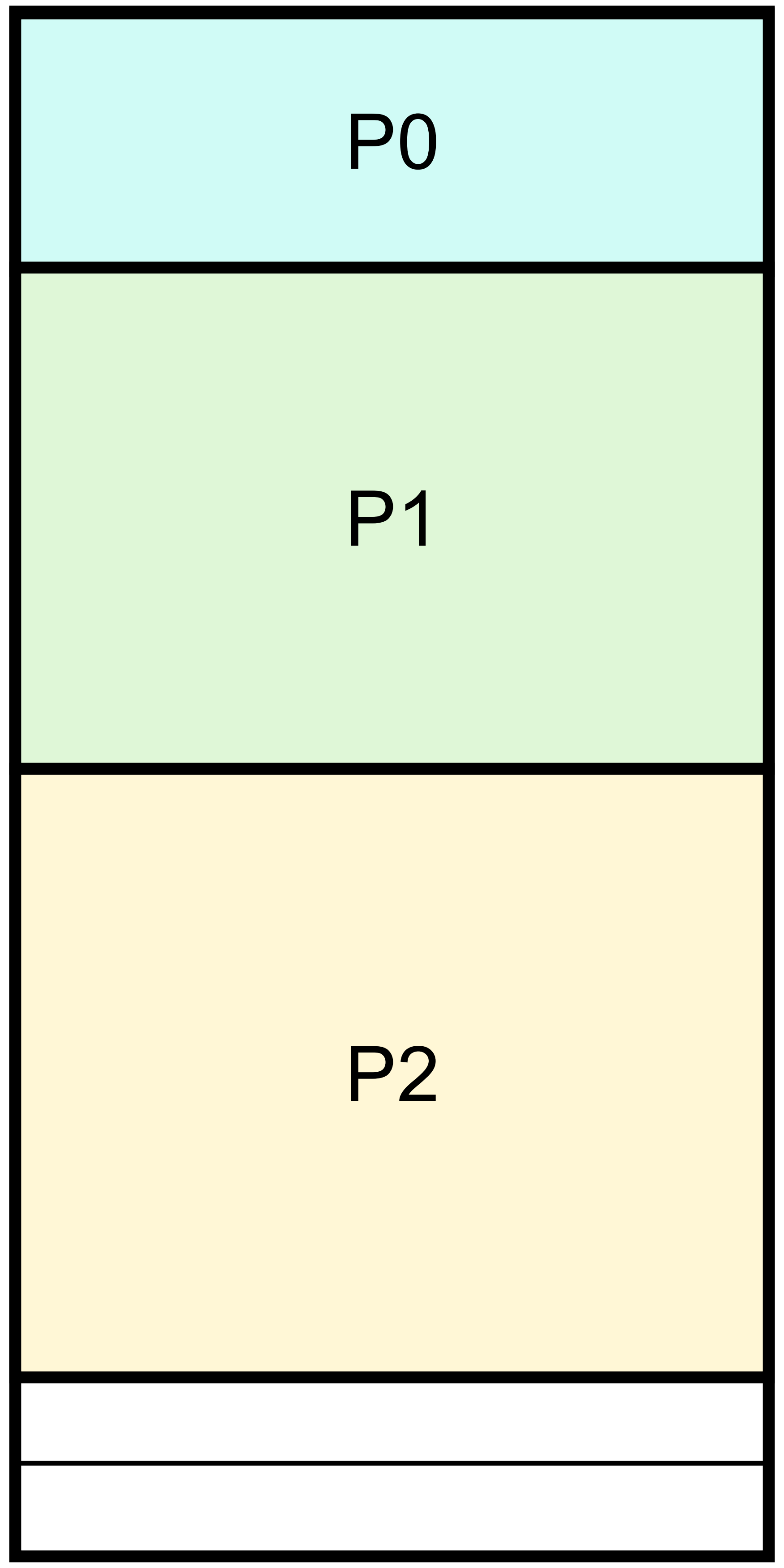
Memory management v0



Memory management v0: Internal fragmentations

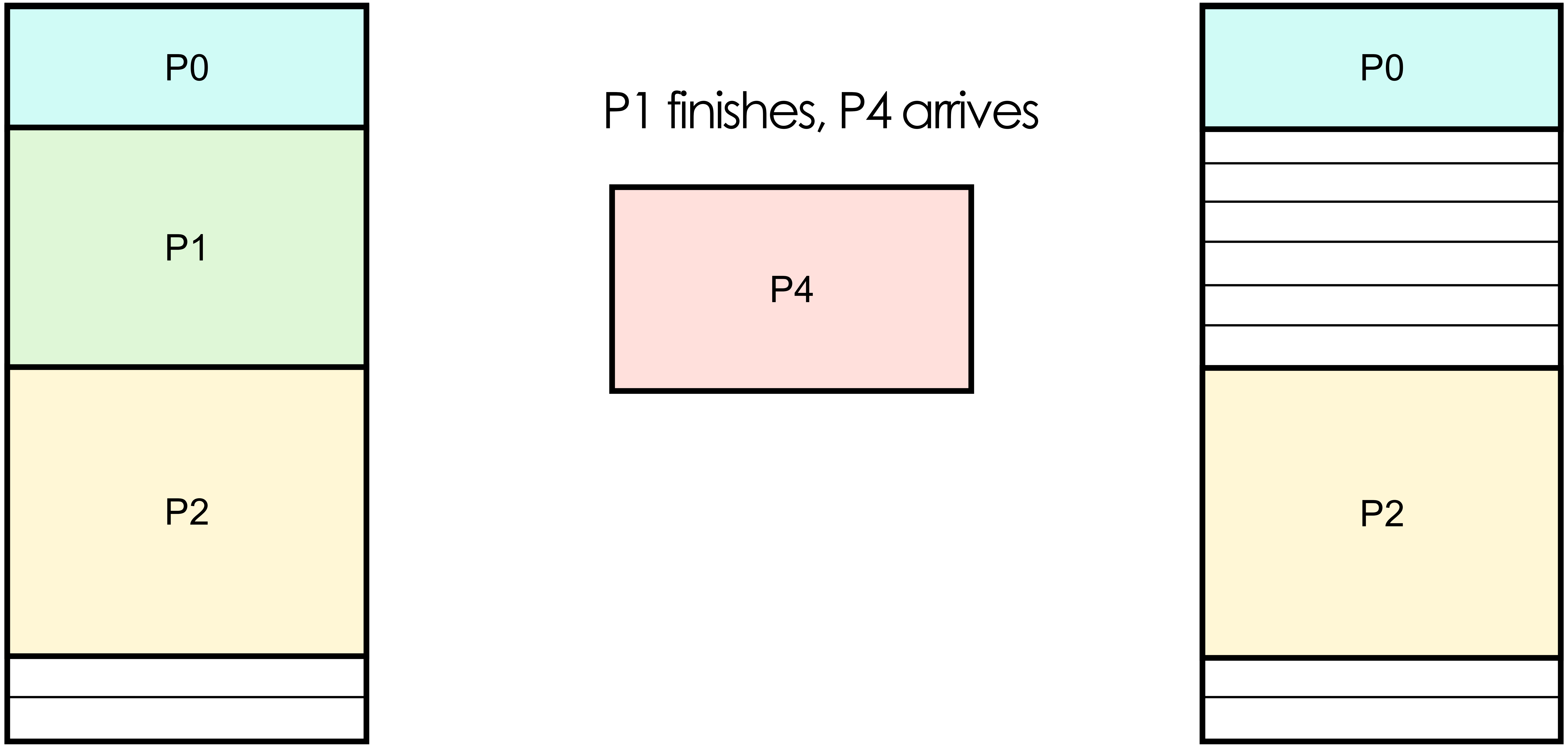


Memory management v1: use a smaller chunk

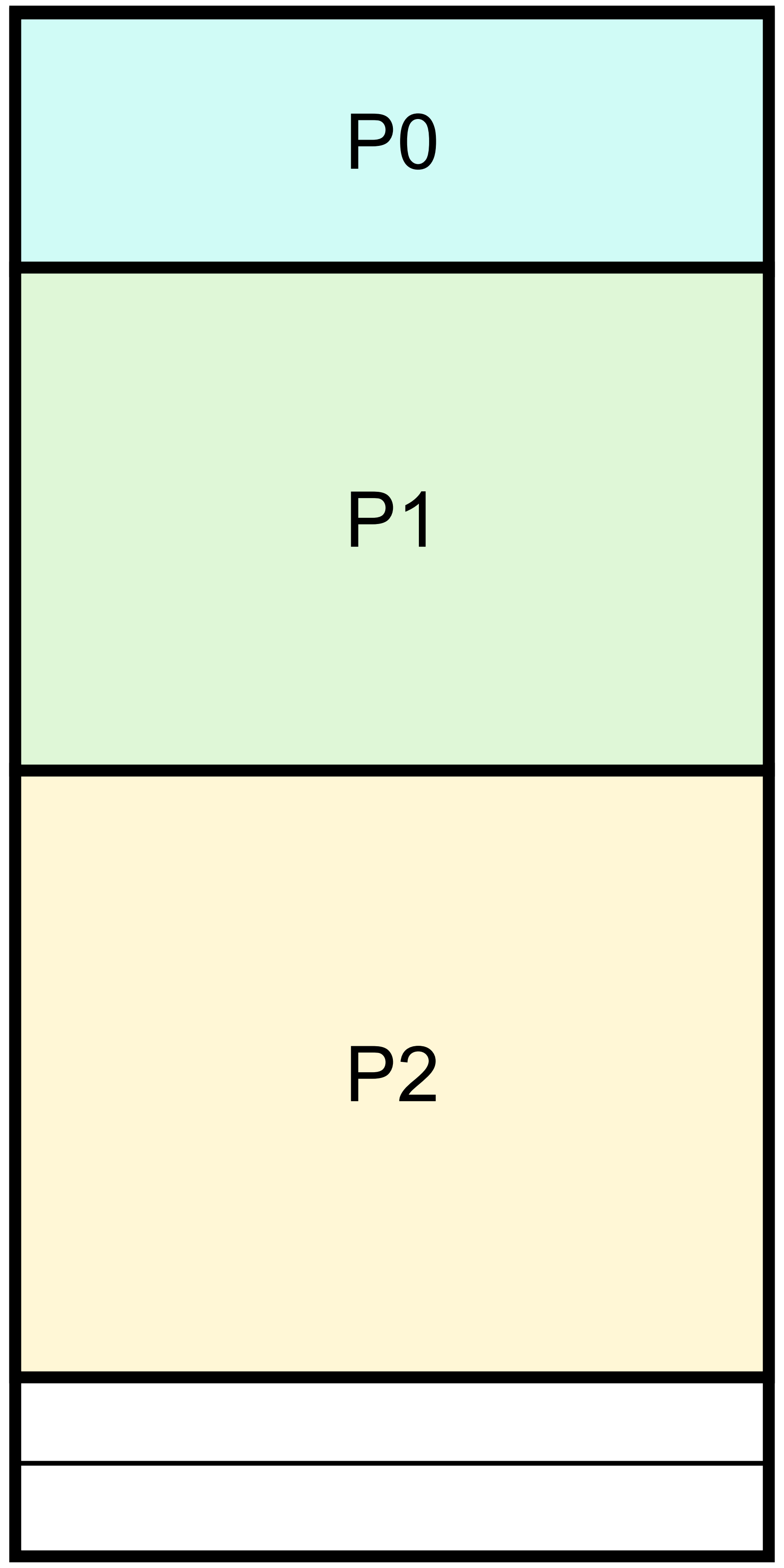


Q: What is the maximum possible amount of internal fragmentation per process?

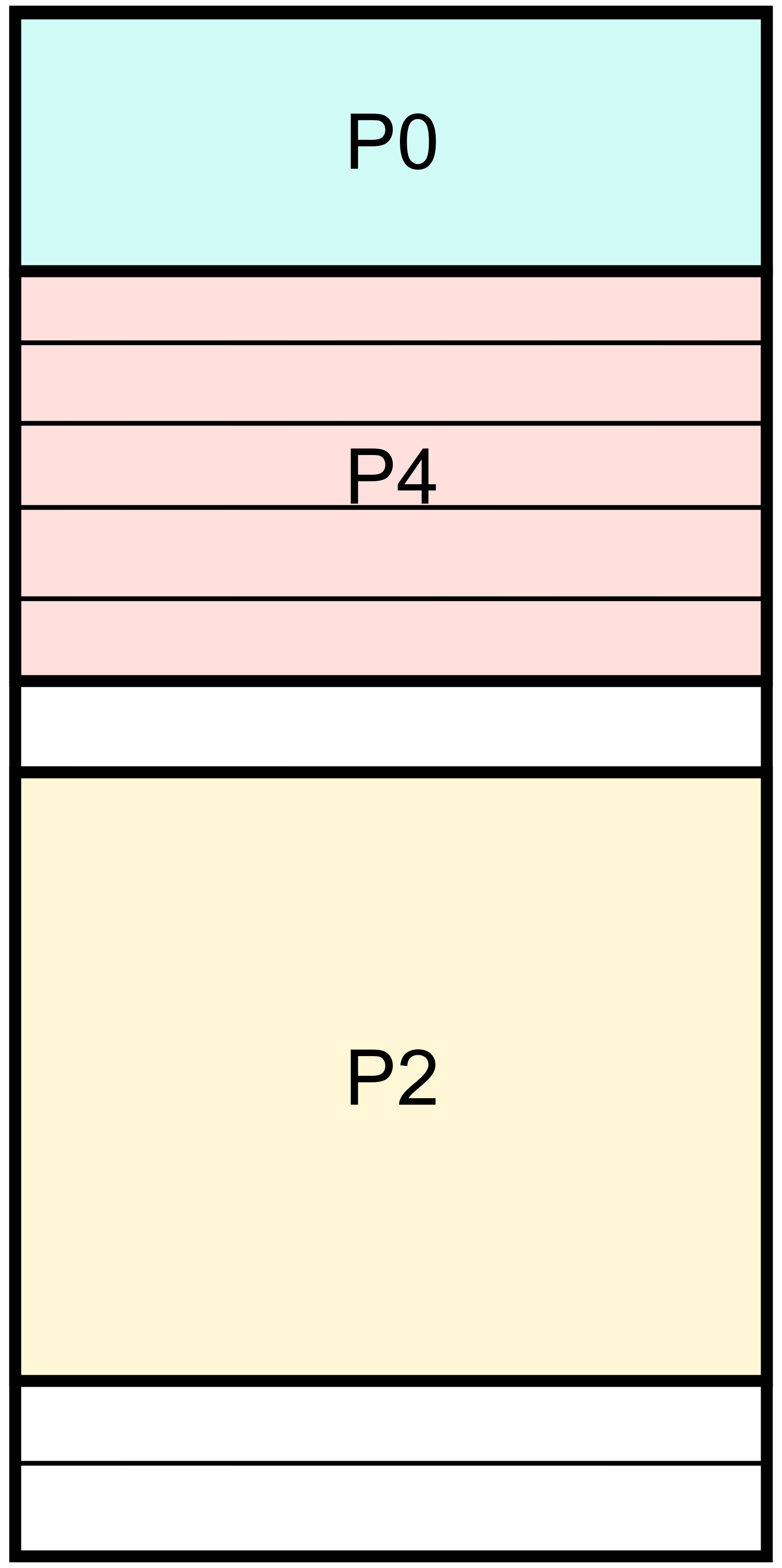
Memory management v1



Memory: v2

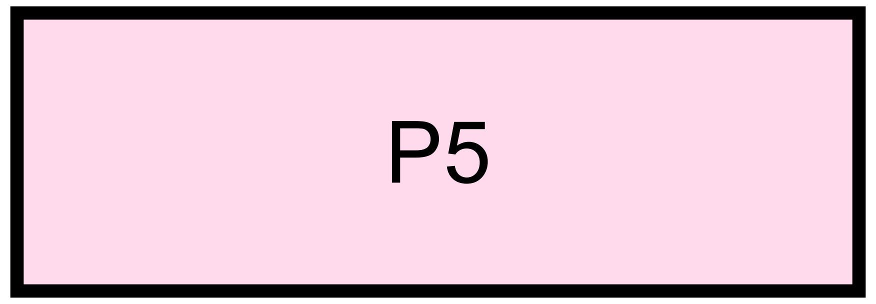


P4 scheduled



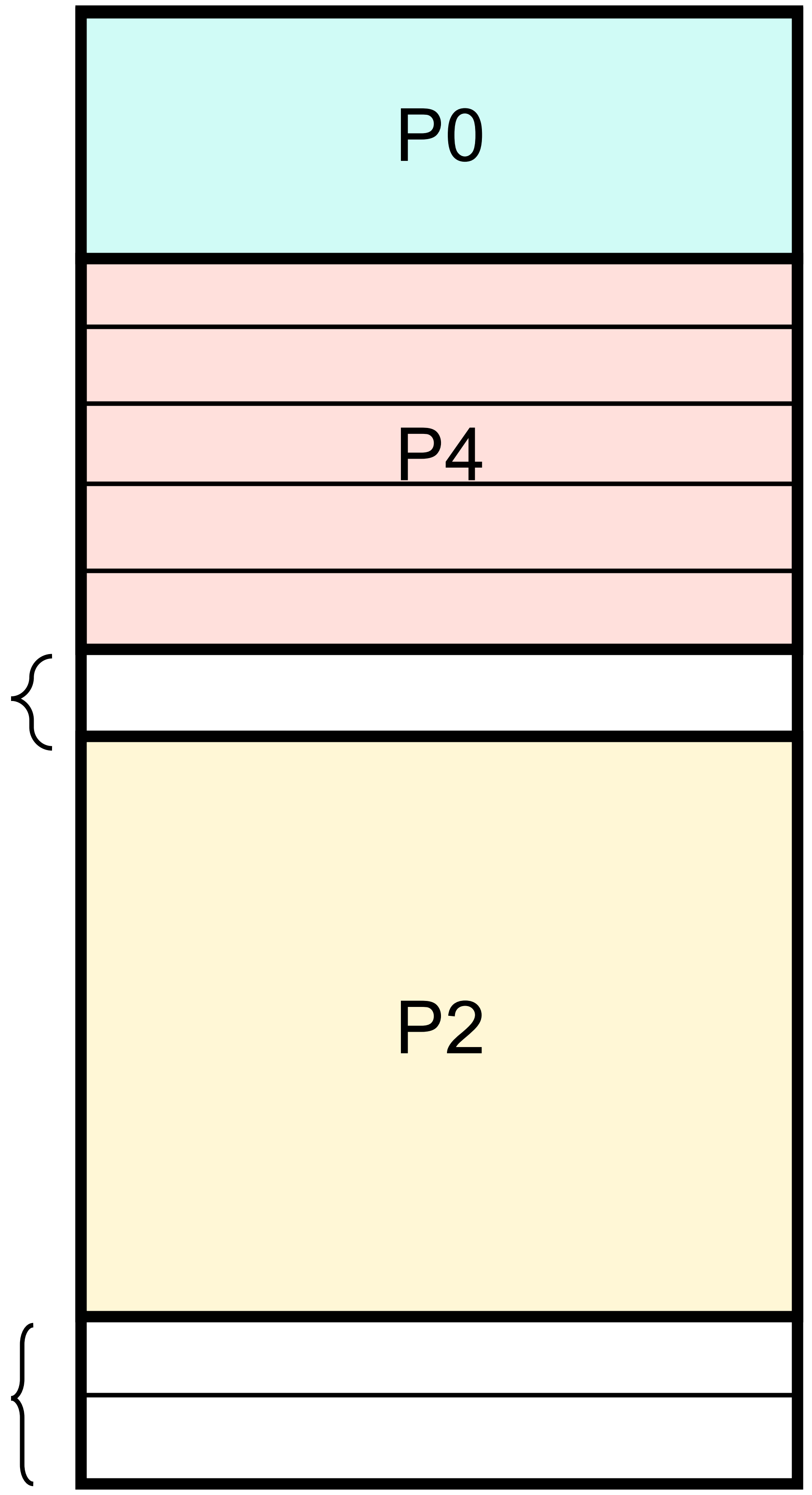
Memory: v2

P5 arrived

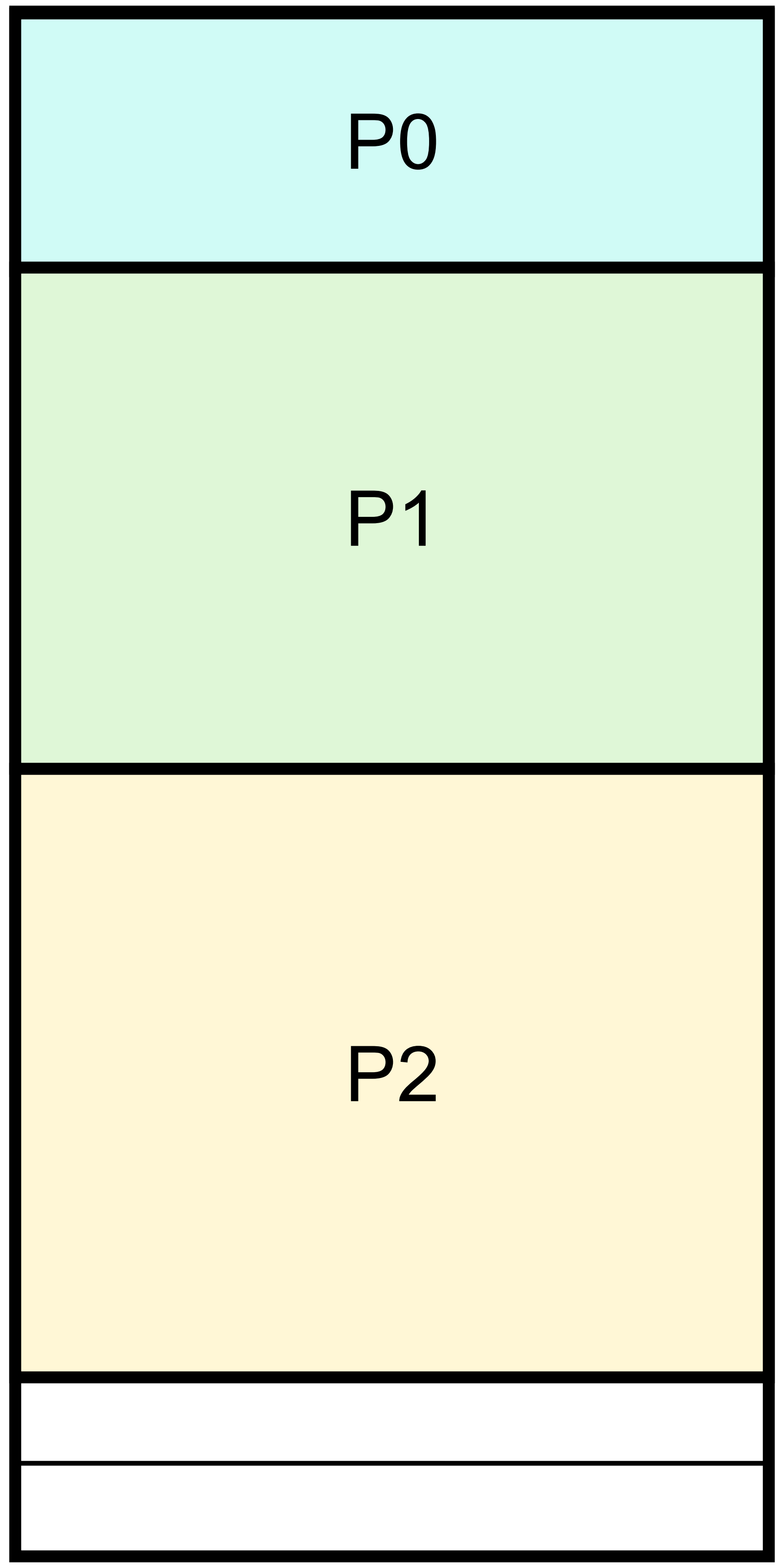


Problem:
There is enough memory for P5, but it cannot be scheduled.

Q: How to address external fragmentation?

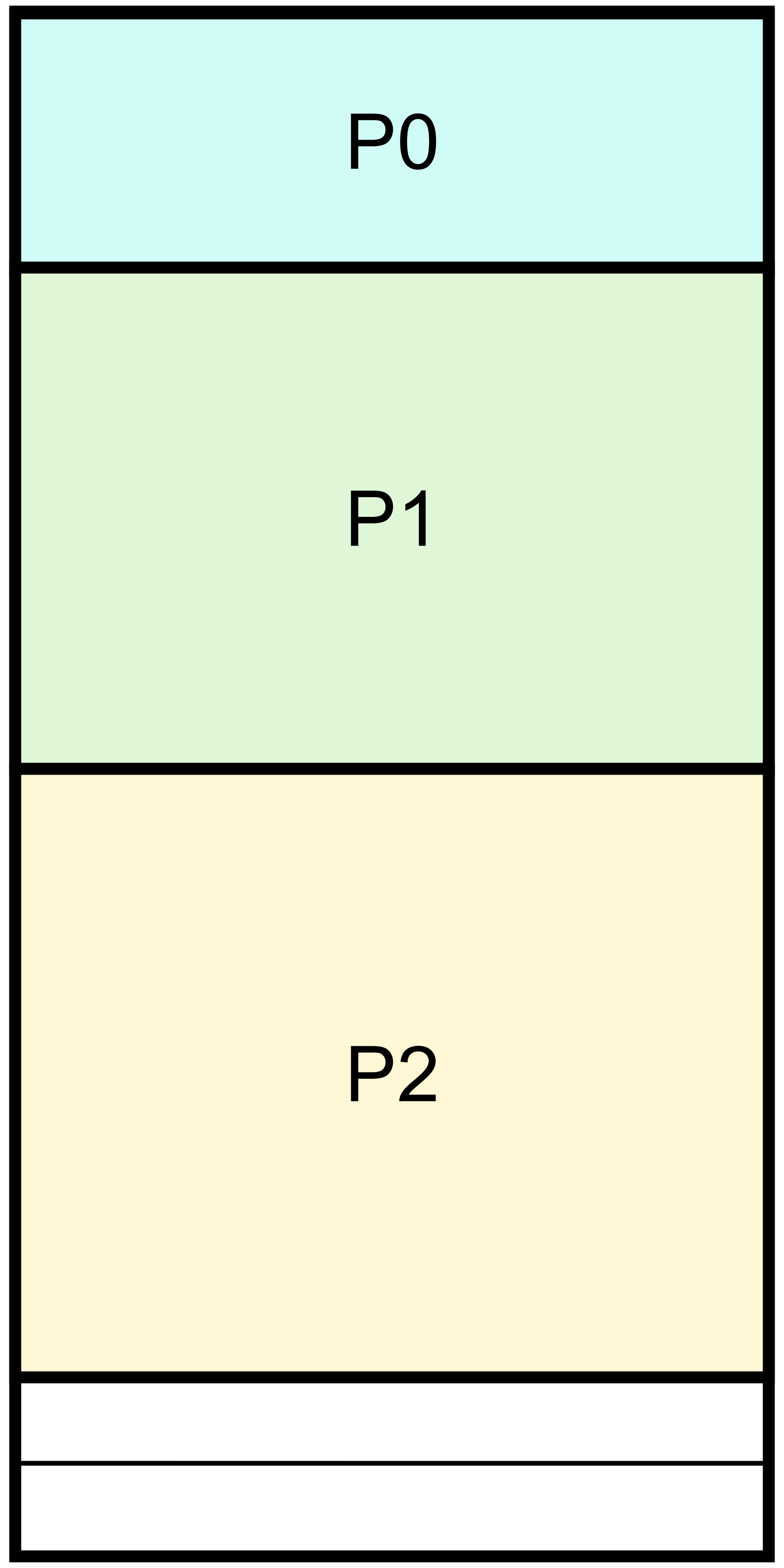


Other Problems?



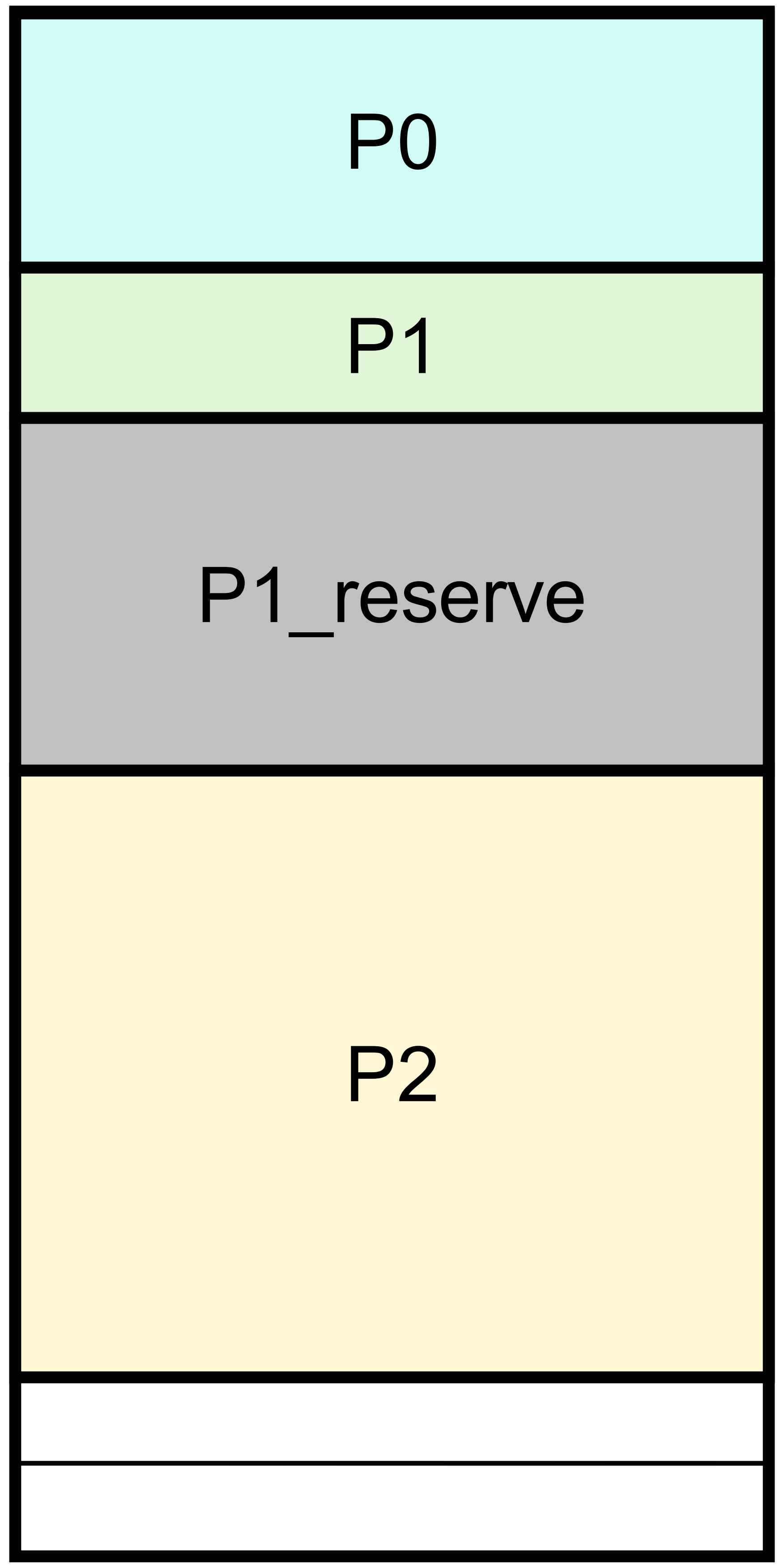
Problem: We can never schedule processes with their memory consumption greater than memory cap

Other Problems?



Problem:
What if we are unsure about how much memory P0/P1/P2 will eventually use?

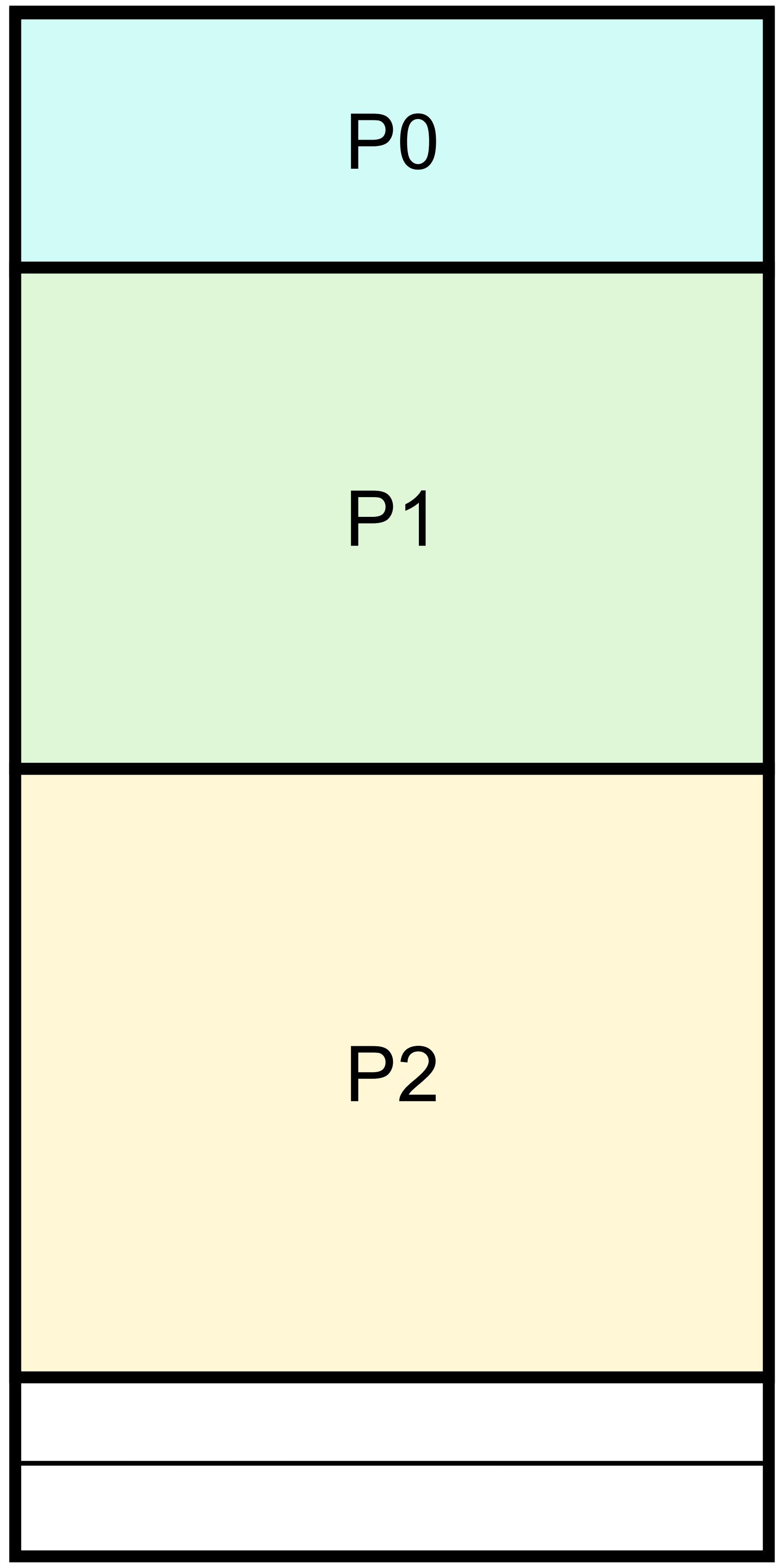
Other Problems?



Problem:
What if we are unsure about how much memory P0/P1/P2 will eventually use?

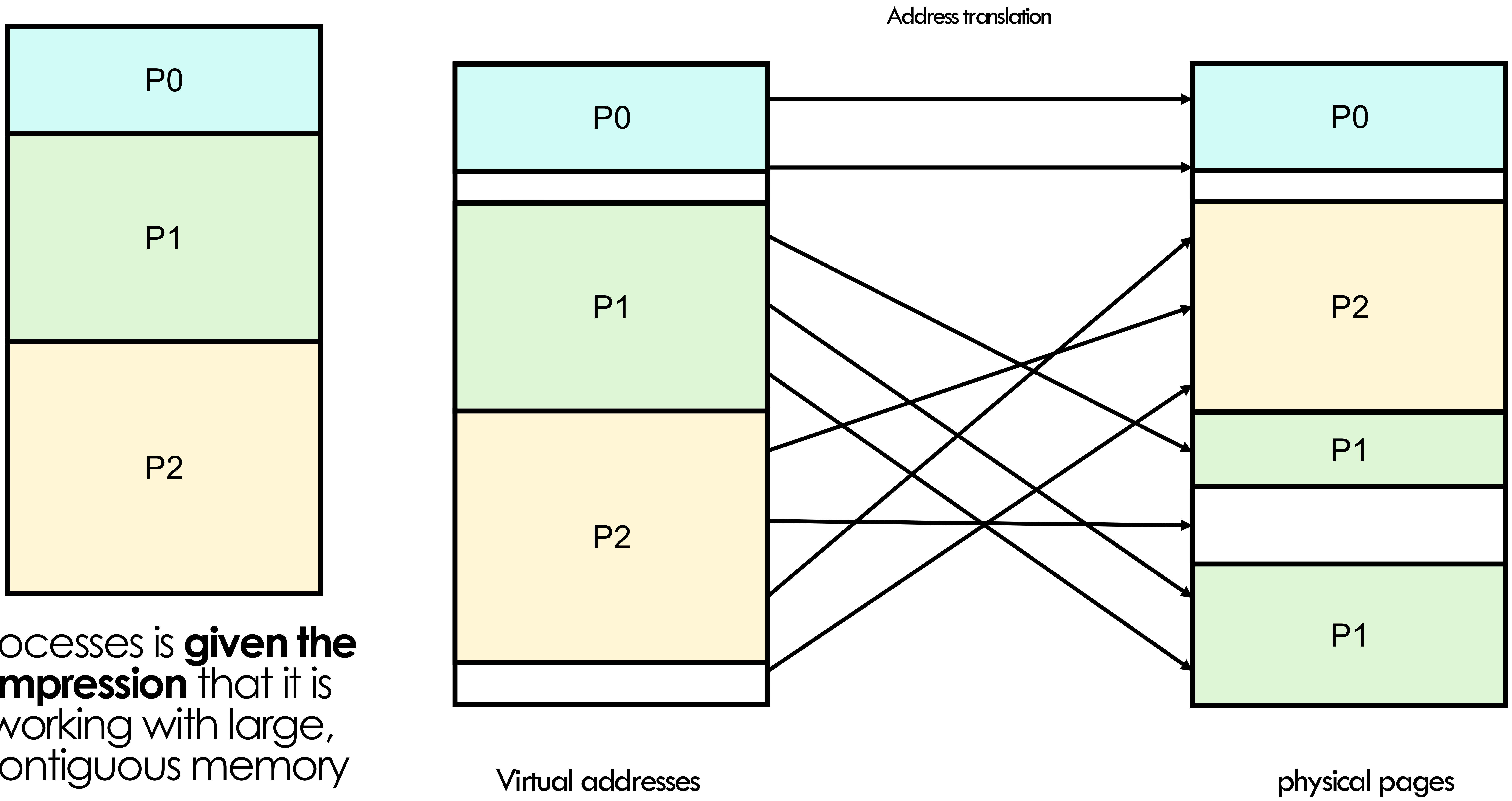
P1_reserve is the reservation overhead

Other Problems?



What if we **know exactly** how much memory P0/P1/P2 will **eventually** use, any problem?

Virtual Address Table



Processes is **given the impression** that it is working with large, contiguous memory

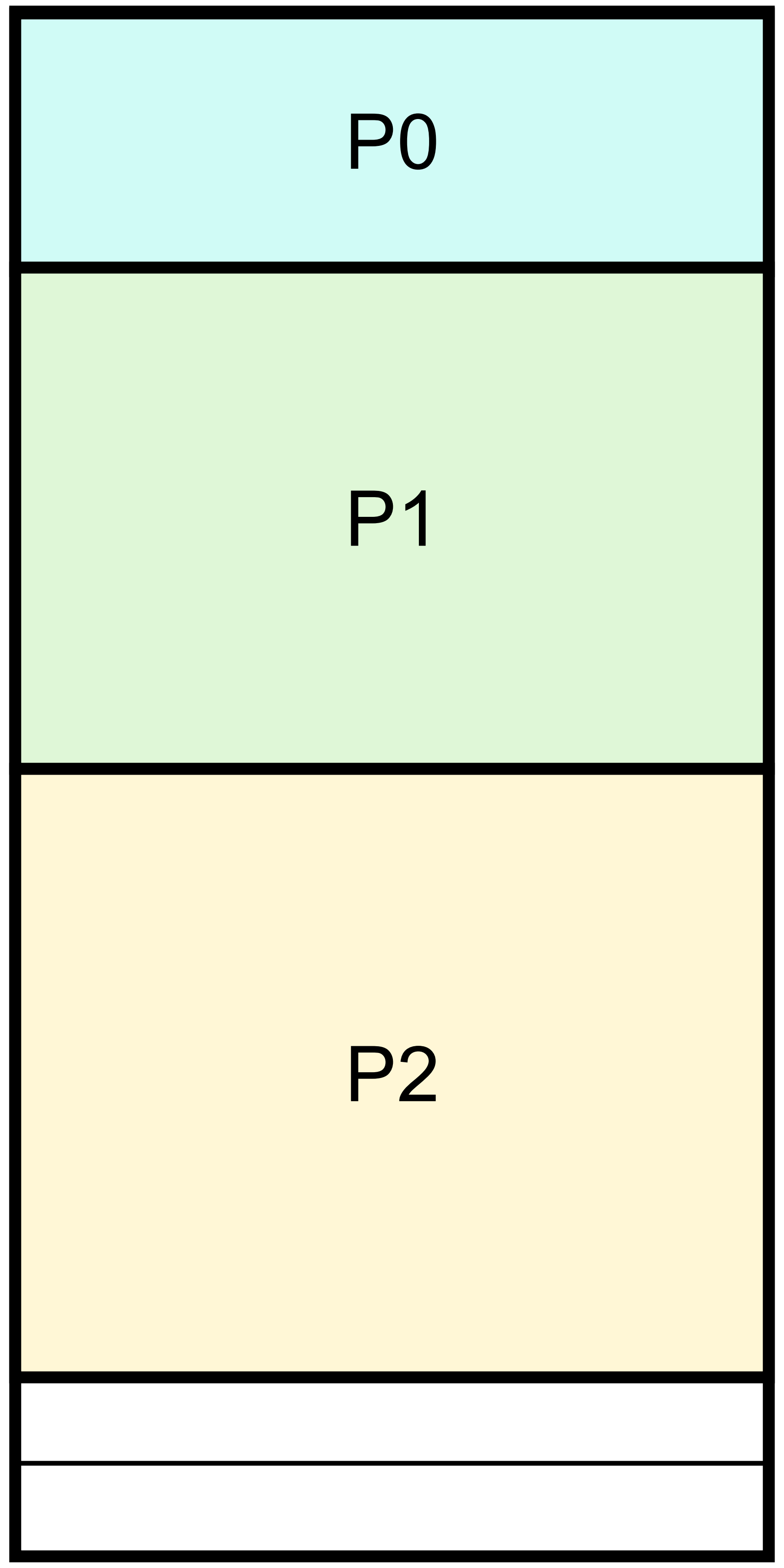
Pages and virtual memory

- **Page**: An abstraction of *fixed* size chunks of memory/storage
- **Page Frame**: Virtual slot in DRAM to hold a page's content
- Page size is usually an OS config
  - e.g., 4KB to 16KB
- OS **Memory Management** can
  - Identify pages uniquely
  - Read/write page from/to disk when requested by a process

Virtual Memory

- **Virtual** Address vs **Physical** Address:
  - Physical is tricky and not flexible for programs
  - Virtual gives “isolation” illusion when using DRAM
  - OS and hardware work together to quickly perform **address translation**
  - OS maintains **free space list** to tell which chunks of DRAM are available for new processes, avoid conflicts, etc.

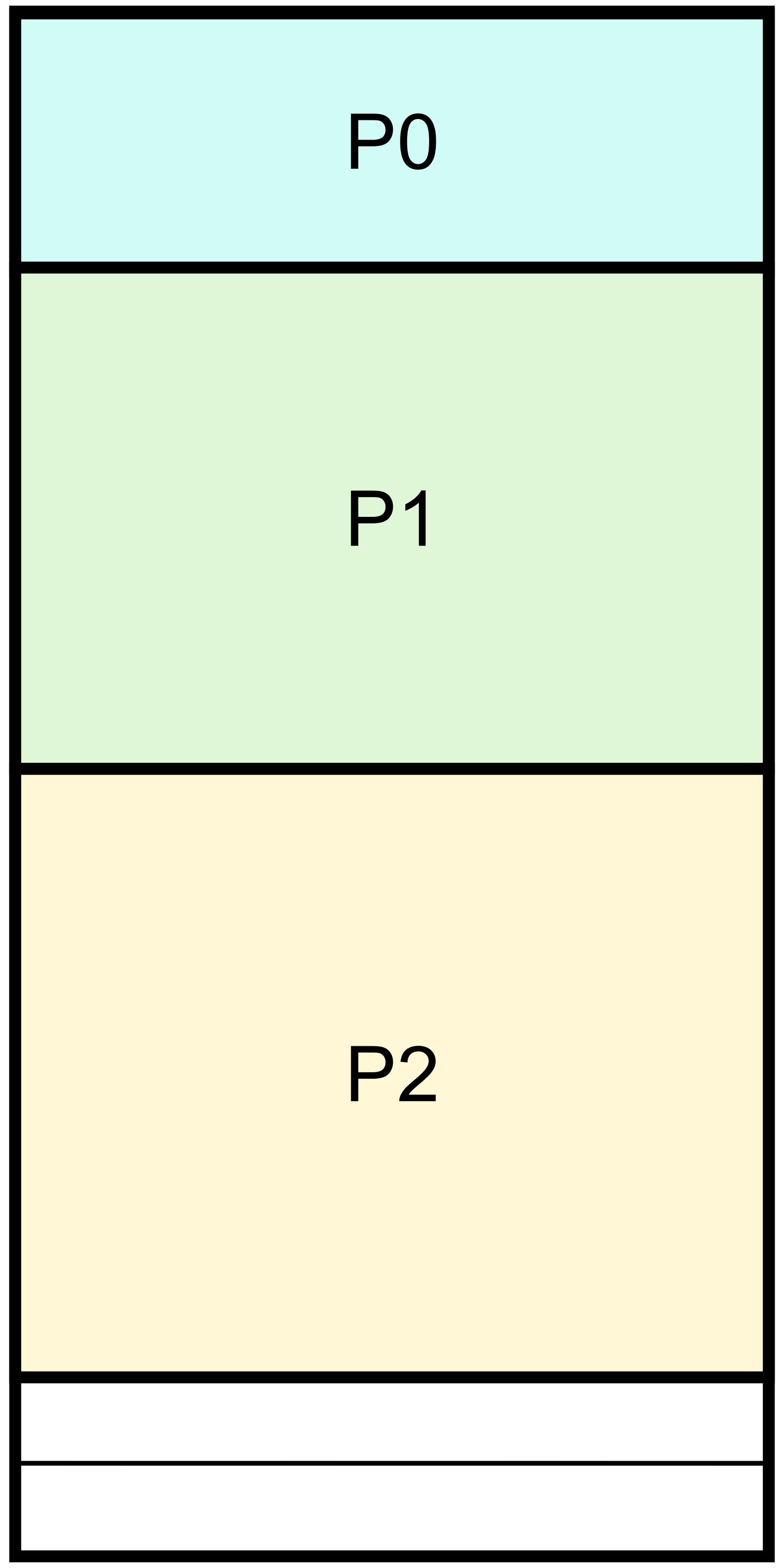
Problem addressed?



Problem: We can never schedule processes with their memory consumption greater than memory cap

Solution: create more virtual addresses than physical memory cap. Map additional ones to disk.

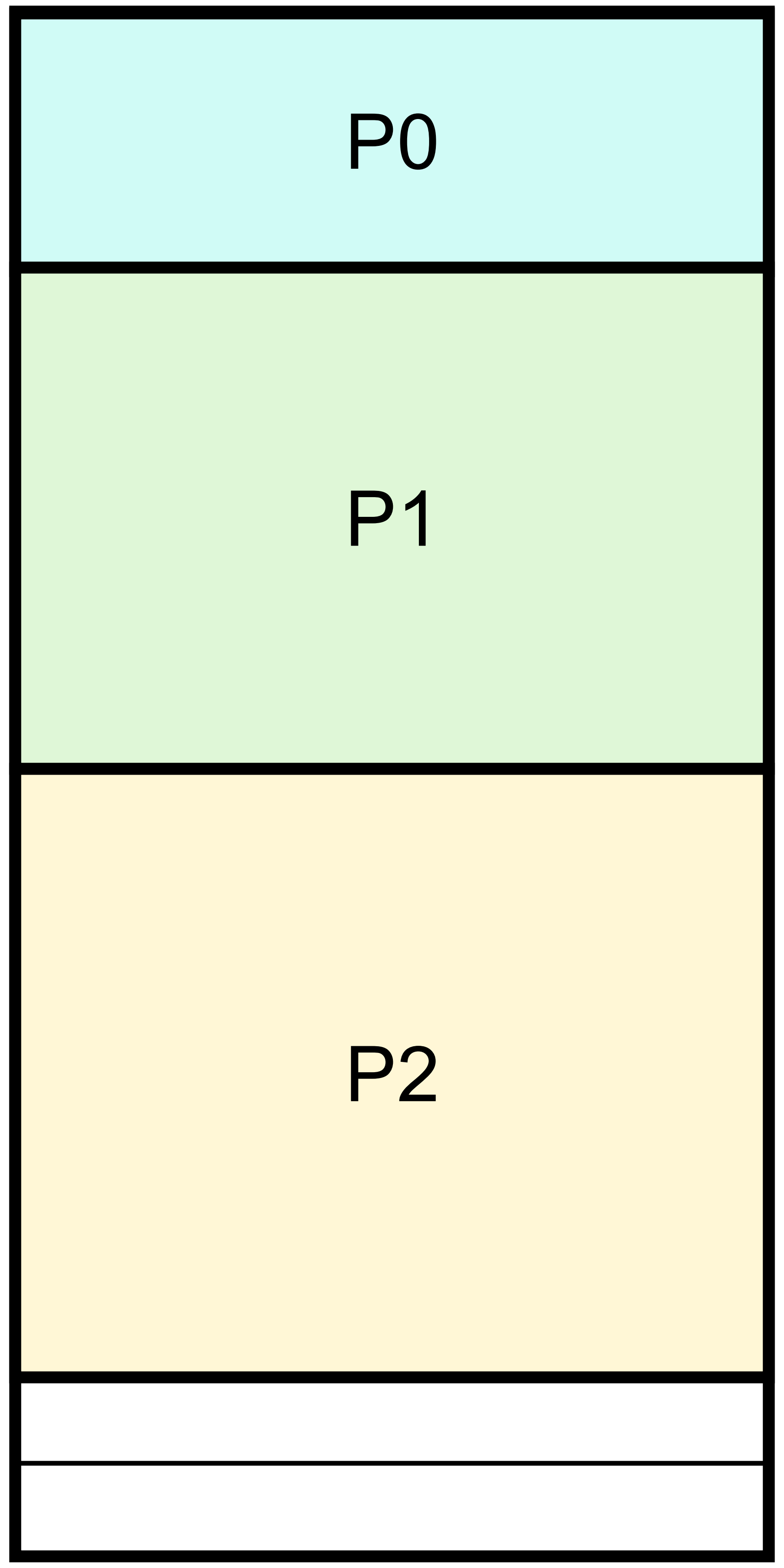
Problem addressed?



Problem:
What if we are unsure about how much memory P0/P1/P2 will eventually use?

Reserve on virtual address, resolve the mapping between virtual and physical pages on-the-fly

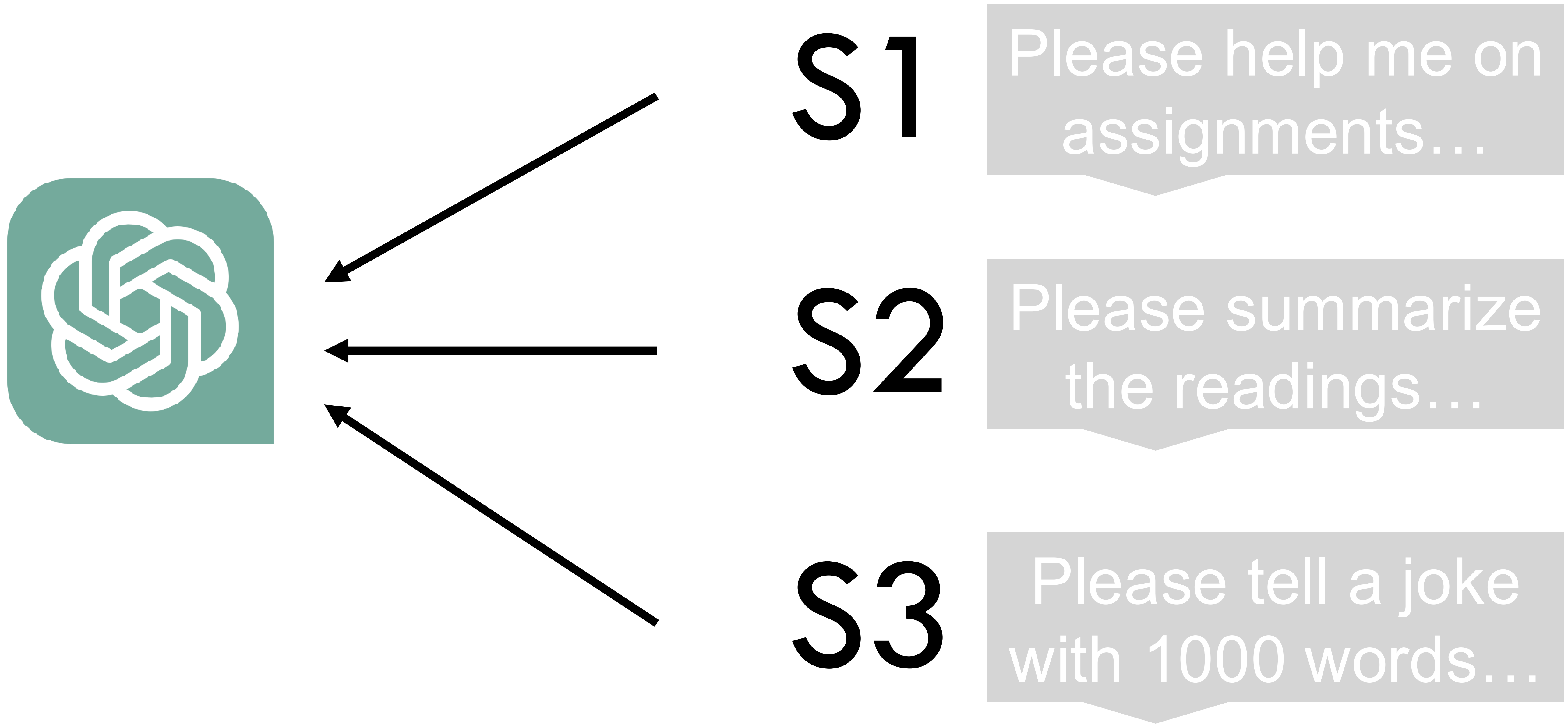
Problem addressed?



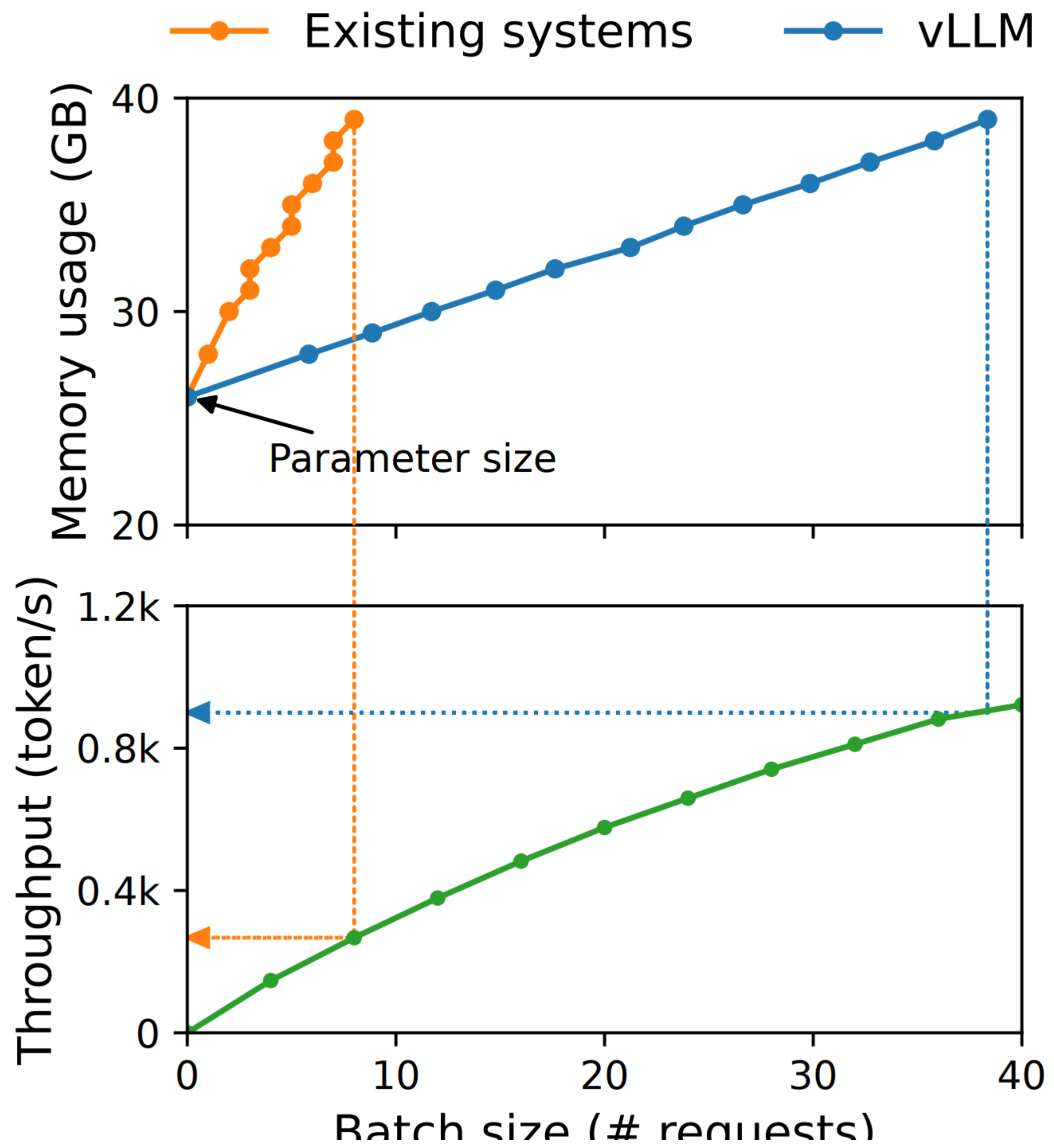
What if we **know exactly** how much memory P0/P1/P2 will **eventually** use, any problem?

Because we do everything on the fly – we minimize opportunity cost

Scheduling in ChatGPT



- How to allocate memory for LLM query?
- Why this could make per LLM request cheaper?



Efficient memory management for large language model serving with pagedattention
W Kwon, Z Li, S Zhuang, Y Sheng, L Zheng, CH Yu, J Gonzalez, H Zhang, ...
Proceedings of the 29th Symposium on Operating Systems Principles, 611-626

Foundation of Data Systems: where we are

- Computer Organization
  - Representation of Data
  - Processors, memory, storages
- Operating System Basics
  - Process, scheduling, concurrency
  - Memory management
  - **File systems**

Modules

- **System call:** The core of an OS with modules to abstract the hardware and APIs for programs to use

“System Call” APIs

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

Hardware device-specific programs

Hardware

Q: *What is a file?*


MORTGAGE
INVESTMENTS
SAVINGS

Abstractions: File and Directory

- File: A persistent sequence of bytes that stores a logically coherent digital object for an application
  - File Format: An application-specific standard that dictates how to interpret and process a file's bytes
  - 100s of file formats exist (e.g., TXT, DOC, GIF, MPEG); varying data models/types, domain-specific, etc.
  - Metadata: Summary or organizing info. about file content (aka *payload*) stored with file itself; format-dependent
- Directory: A cataloging structure with a list of references to files and/or (recursively) other directories
  - Typically treated as a special kind of file
  - Sub dir., Parent dir., Root dir.

Filesystem

- Filesystem: The part of OS that helps programs create, manage, and delete files on disk (sec. storage)
- Roughly split into *logical level* and *physical level*
  - Logical level exposes file and dir. abstractions and offers System Call APIs for file handling
  - Physical level works with disk firmware and moves bytes to/from disk to DRAM

Filesystem

- Dozens of filesystems exist, e.g., ext2, ext3, NTFS, etc.
  - Differ on how they layer file and dir. abstractions as bytes, what metadata is stored, etc.
  - Differ on how data integrity/reliability is assured, support for editing/resizing, compression/encryption, etc.
  - Some can work with (“mounted” by) multiple OSs

Virtualization of File on Disk

- OS abstracts a file on disk as a virtual object for processes
- File Descriptor: An OS-assigned +ve integer identifier/reference for a file's virtual object that a process can use
  - 0/1/2 reserved for STDIN/STDOUT/STDERR
  - File Handle: A PL's abstraction on top of a file descr. (fd)

Q: *What is a database? How is it different from just a bunch of files?*

Collection of files?

Virtualization of Files

Binary Representation on Disk storage

- Maintenance
- Performance
- Usability
- Security & privacy
- …

Files Vs Databases: Data Model

- Database: An *organized* collection of interrelated data
  - Data Model: An abstract model to define organization of data in a formal (mathematically precise) way
  - E.g., Relations, XML, Matrices, DataFrames

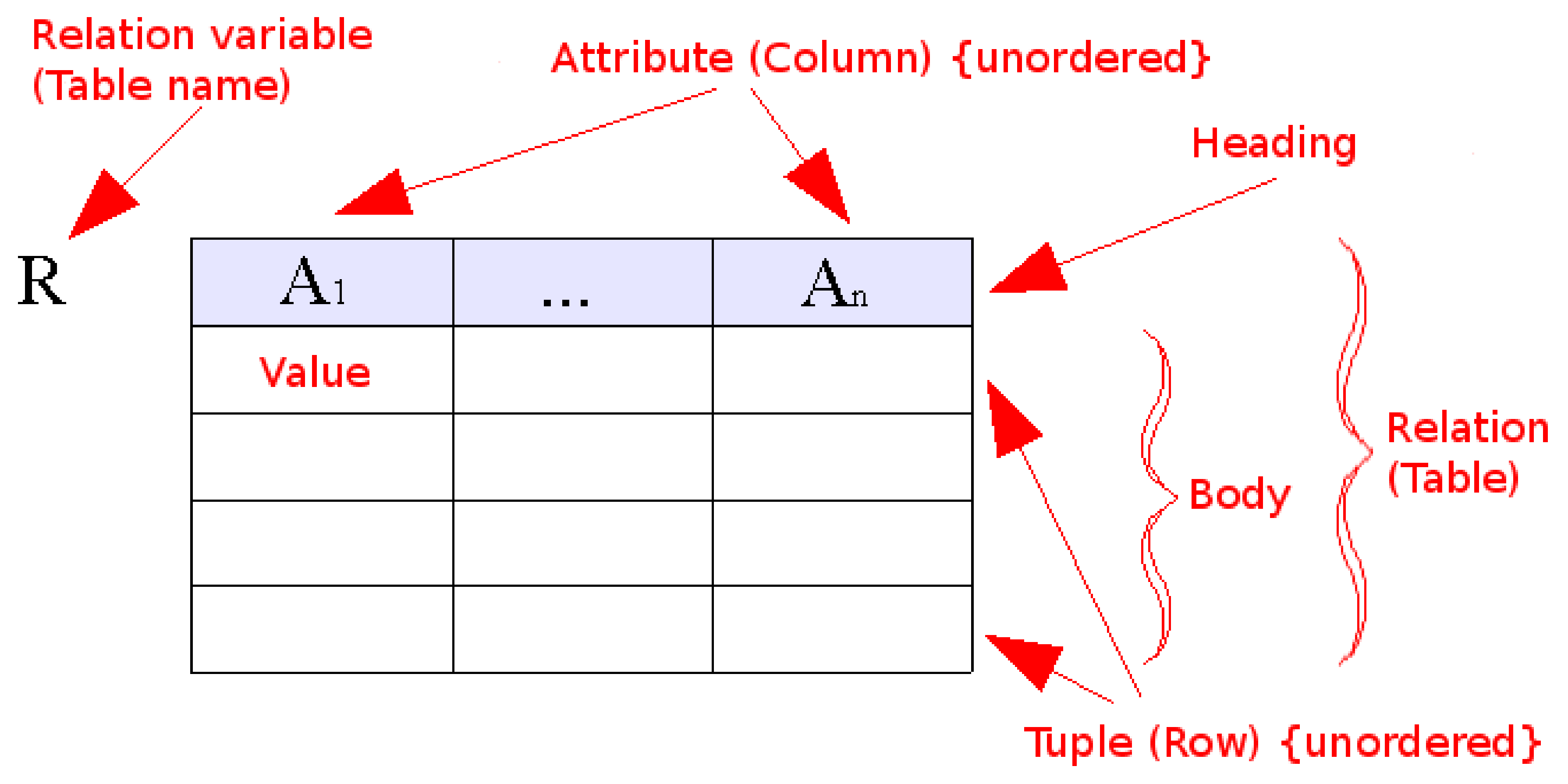
Files Vs Databases: Data Model

- Every database is just an *abstraction* on top of data files!
  - Logical level: Data model for higher-level reasoning
    - More in the later lectures.
  - Physical level: How bytes are layered on top of files
    - More in the later lectures.
  - All data systems (RDBMSs, Dask, Spark, TensorFlow, etc.) are application/platform software that use OS System Call API for handling data files

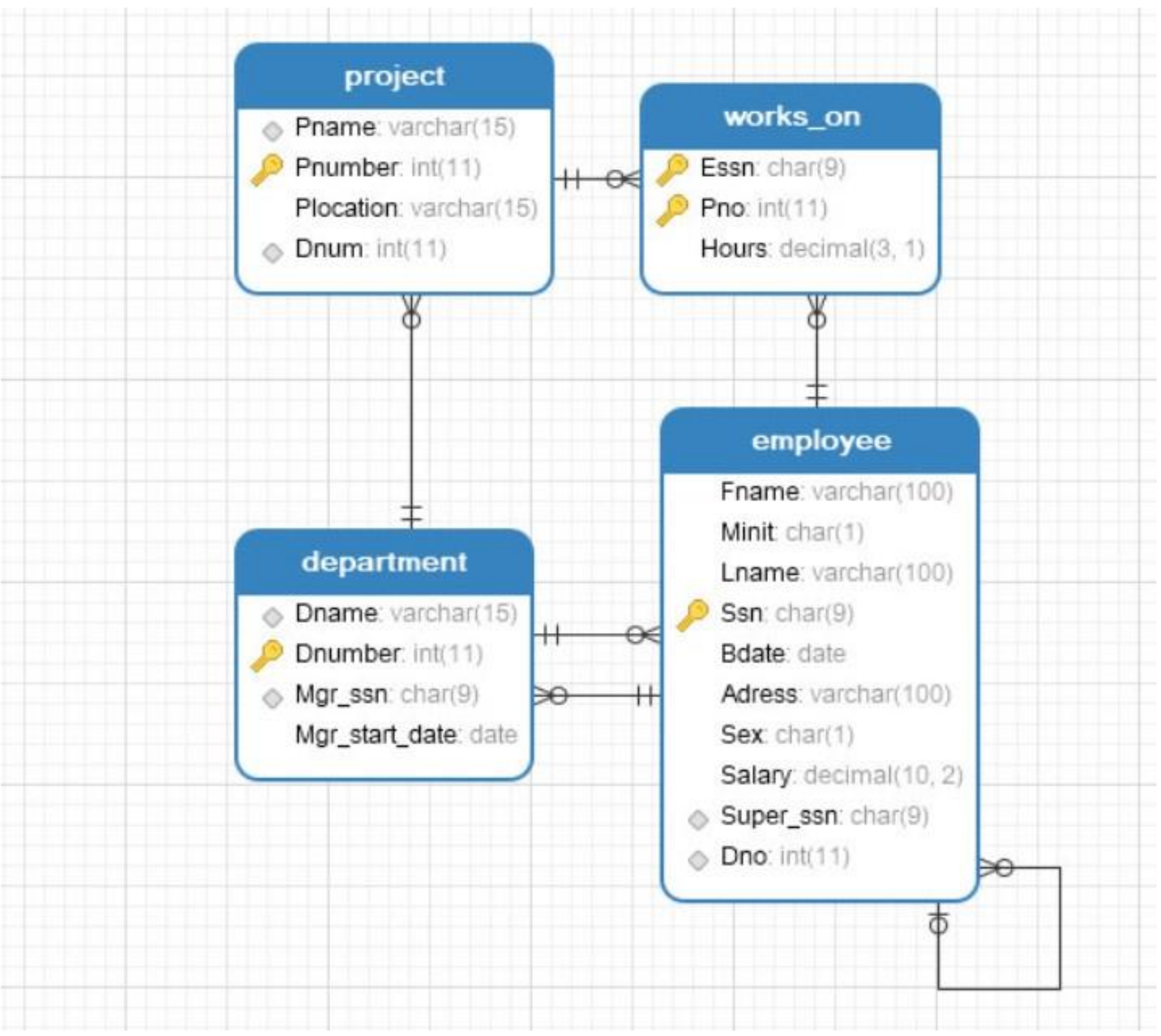
Data as File: Structured

- **Structured Data:** A form of data with regular substructure

Relation



Relational Database



- Most RDBMSs and Spark serialize a relation **as *binary* file(s)**, often compressed

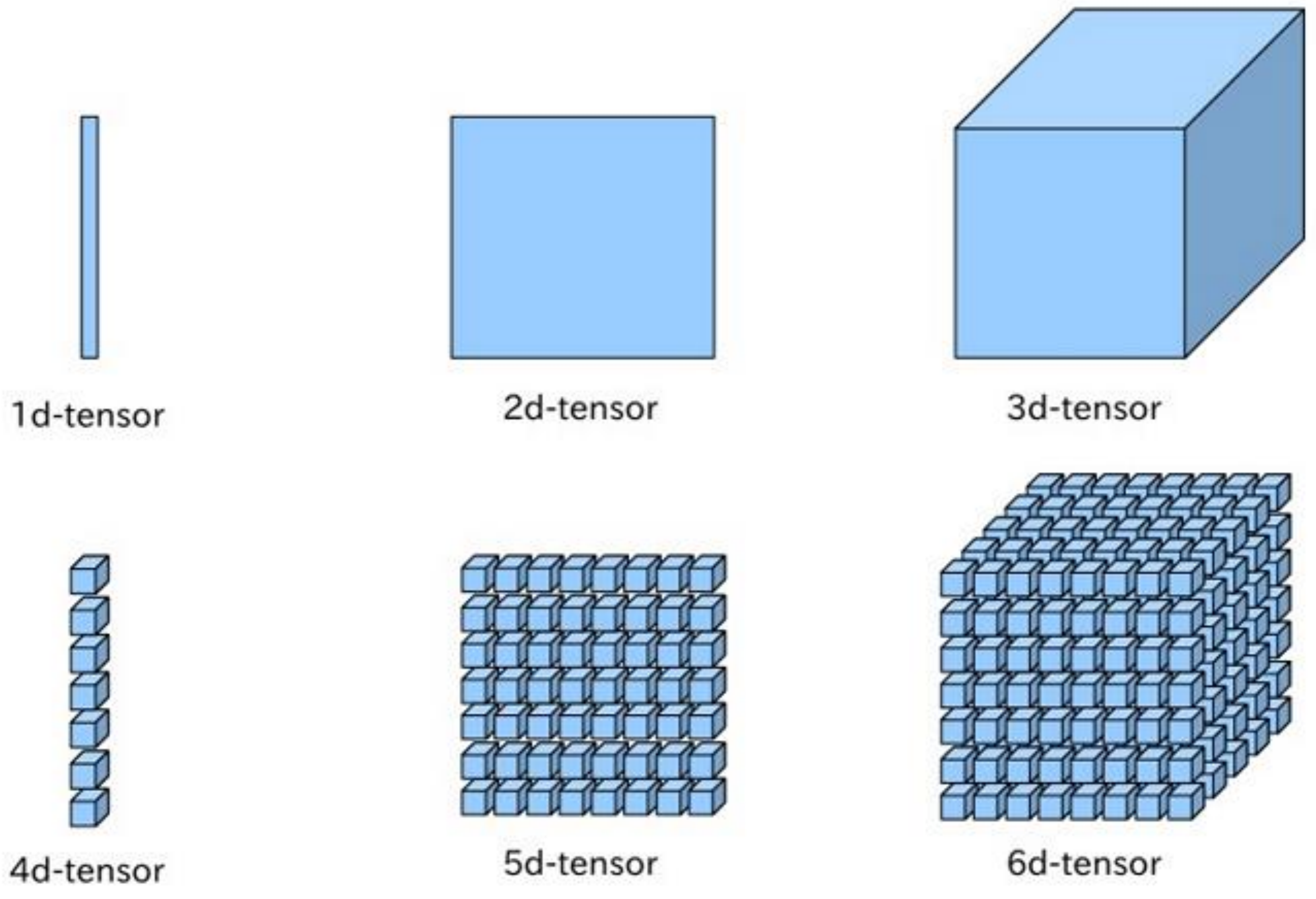
Data as File: Structured

- Structured Data: A form of data with regular substructure

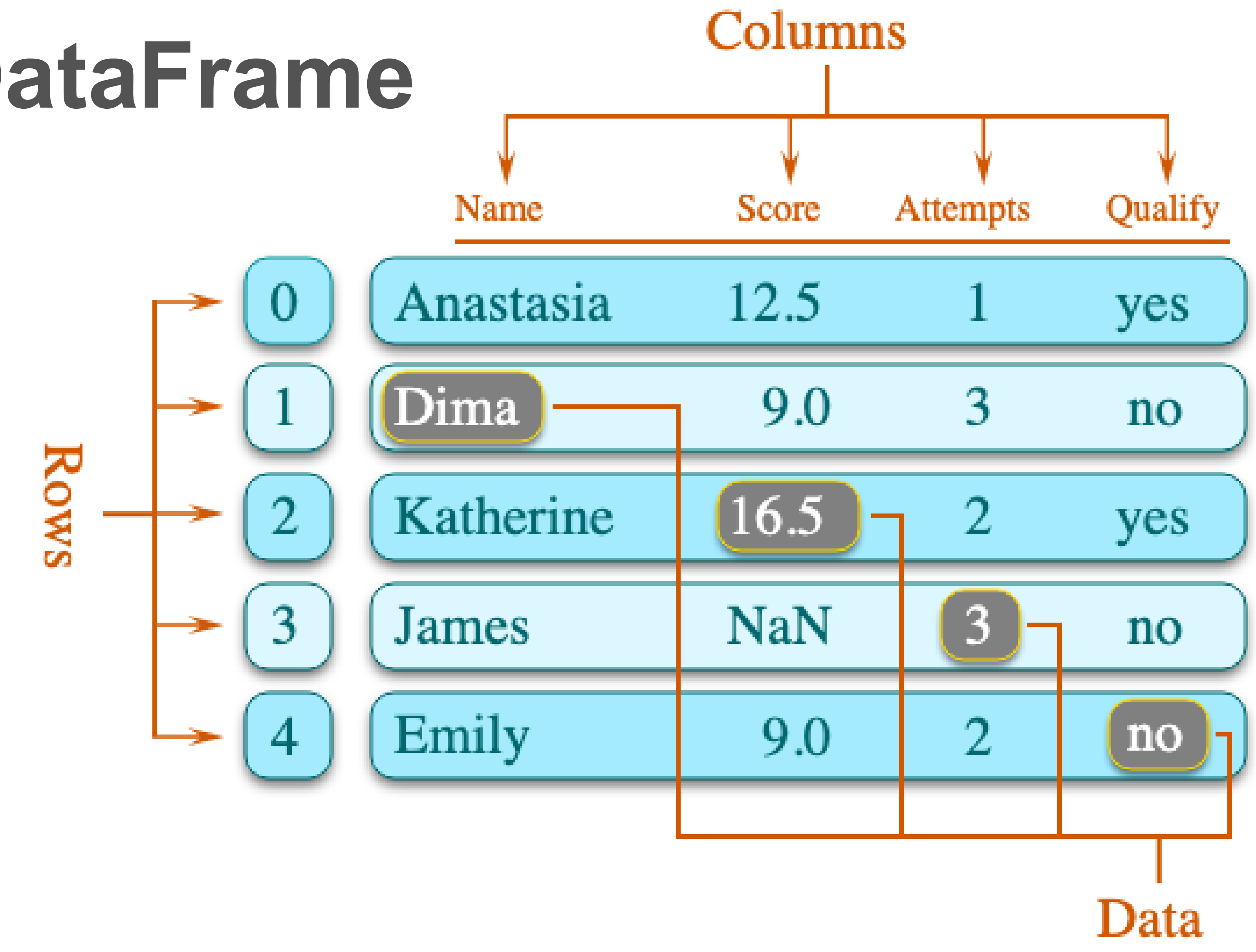
Matrix

$$
\begin{array}{c|cccc|}
 & 1 & 2 & \dots & n \\
\hline
1 & a_{11} & a_{12} & \dots & a_{1n} \\
2 & a_{21} & a_{22} & \dots & a_{2n} \\
3 & a_{31} & a_{32} & \dots & a_{3n} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
m & a_{m1} & a_{m2} & \dots & a_{mn}
\end{array}
$$

Tensor



DataFrame

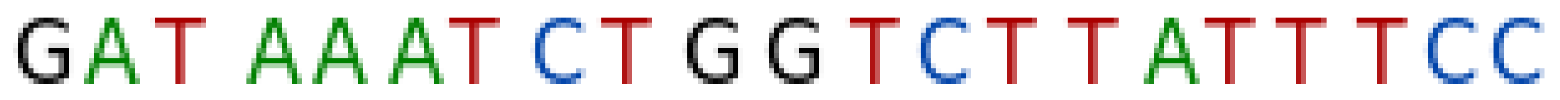


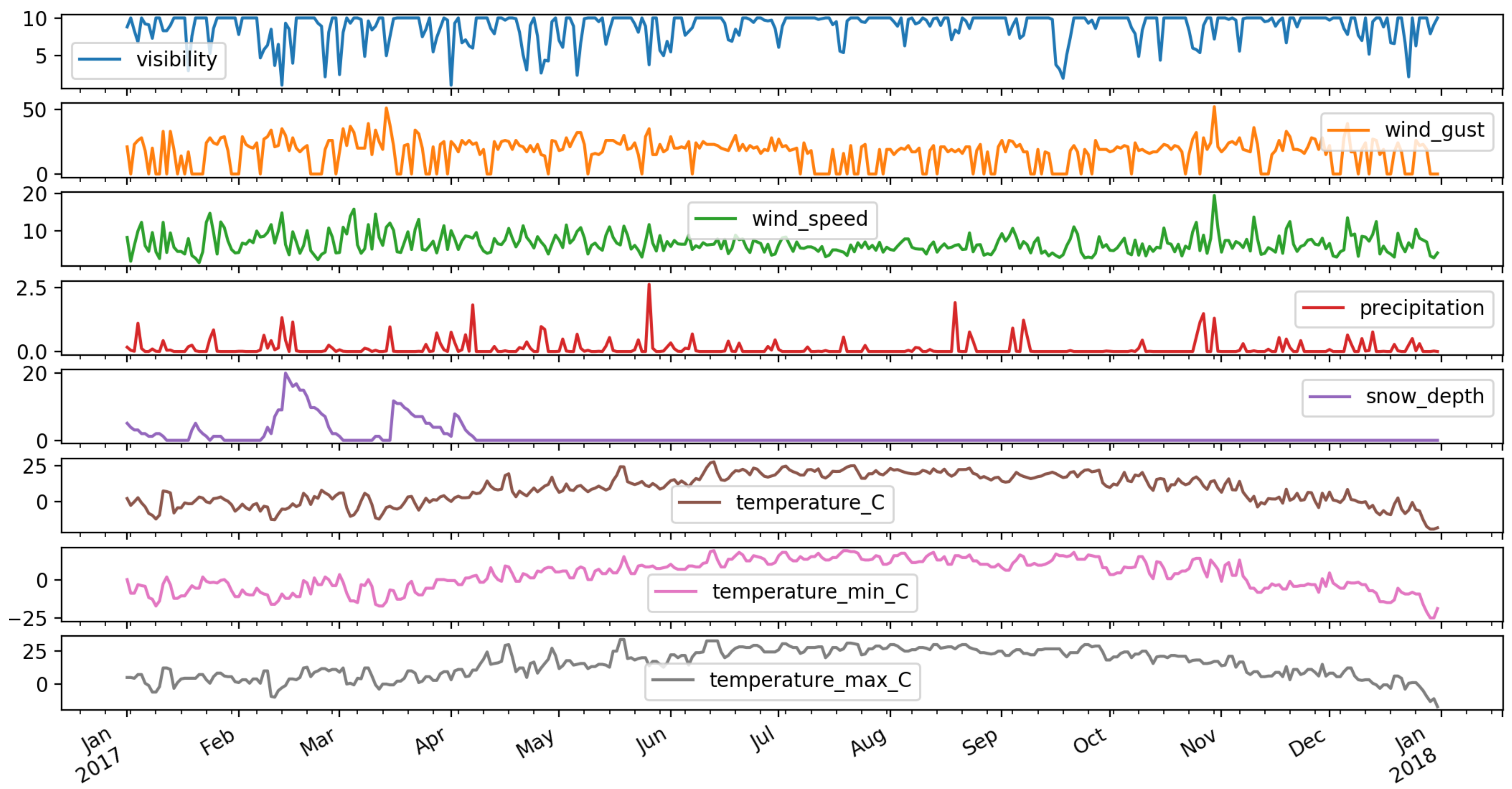
- Typically serialized as restricted ASCII text file (TSV, CSV, etc.)
- Matrix/tensor as binary too
- Can layer on Relations too!

Data as File: Structured

- Structured Data: A form of data with regular substructure

Sequence (Includes Time-series)

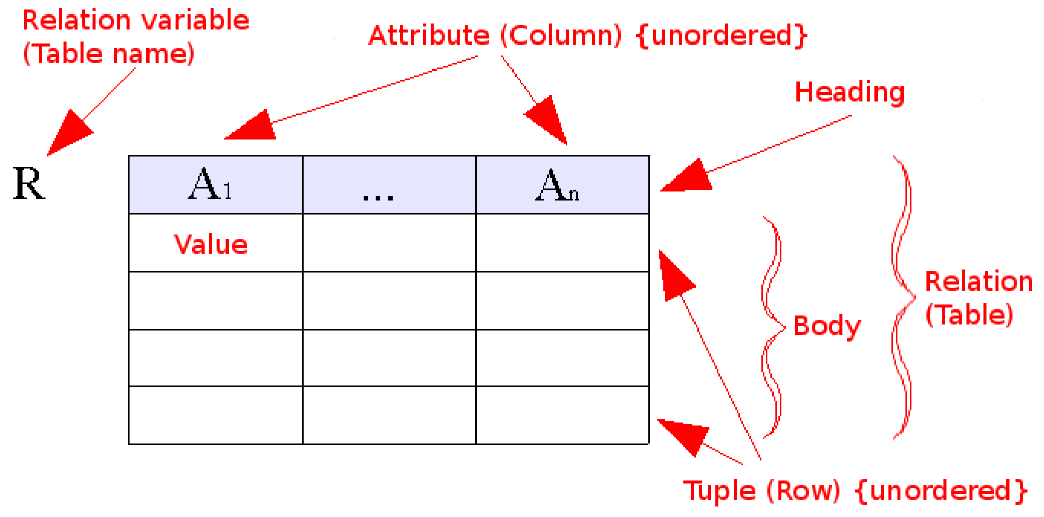




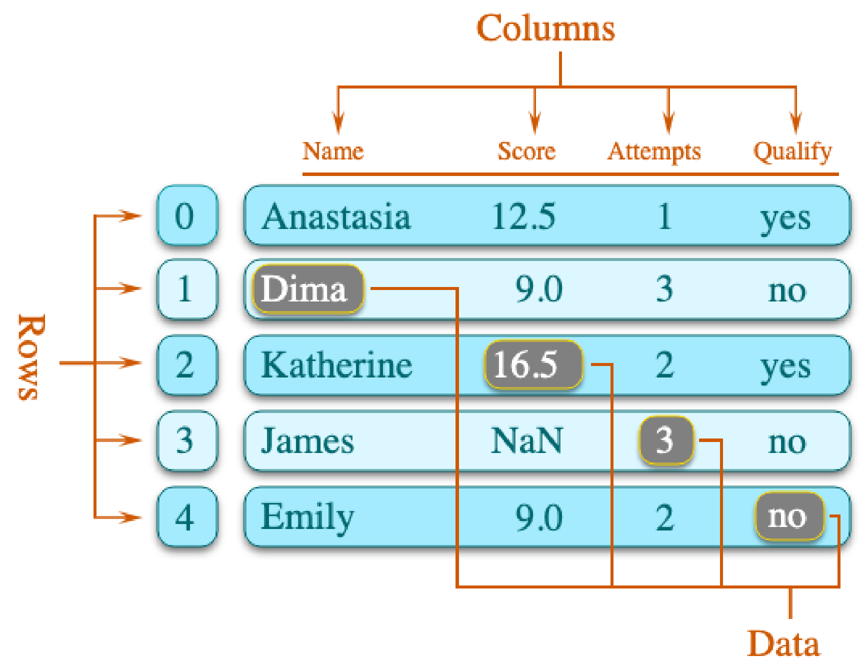
- Can layer on Relations, Matrices, or DataFrames, or be treated as first-class data model
- Inherits flexibility in file formats (text, binary, etc.)

Comparing Struct. Data Models

***Q:** What is the difference between Relation, Matrix, and DataFrame?*



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

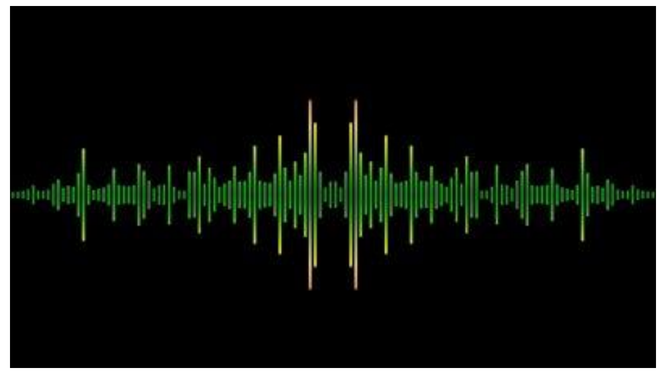
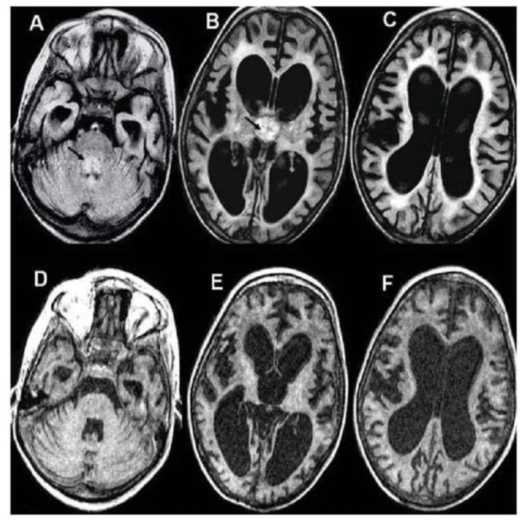


- Ordering: Matrix and DataFrame have row/col numbers; Relation is orderless on both axes!
- Schema Flexibility: Matrix cells are numbers. Relation tuples conform to pre-defined schema. DataFrame has no pre-defined schema but all rows/cols can have names; col cells can be mixed types!
- Transpose: Supported by Matrix & DataFrame, not Relation

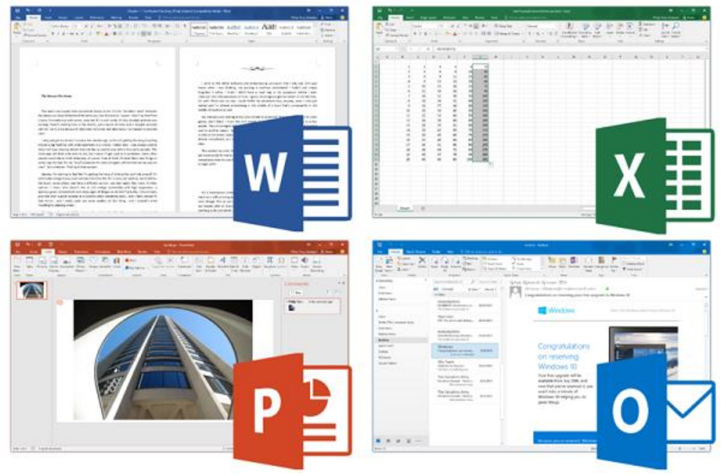
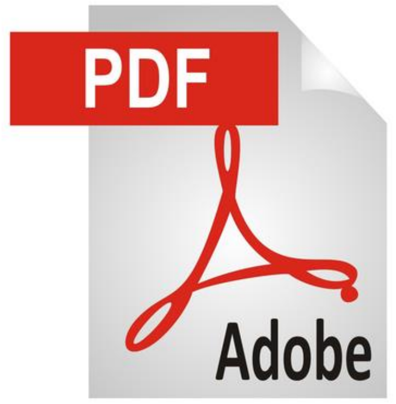
If interested in reading more: https://towardsdatascience.com/preventing-the-death-of-the-dataframe-8bca1

Data as File: Other Common Formats

- Machine Perception data layer on tensors and/or time-series
- Myriad binary formats, typically with (lossy) compression, e.g., WAV for audio, MP4 for video, etc.

- Text File (aka plaintext): Human-readable ASCII characters
- Docs/Multimodal File: Myriad app-specific rich binary formats

jupyter