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

DSC 204A: Scalable Data Systems
Winter 2024

Machine Learning Systems

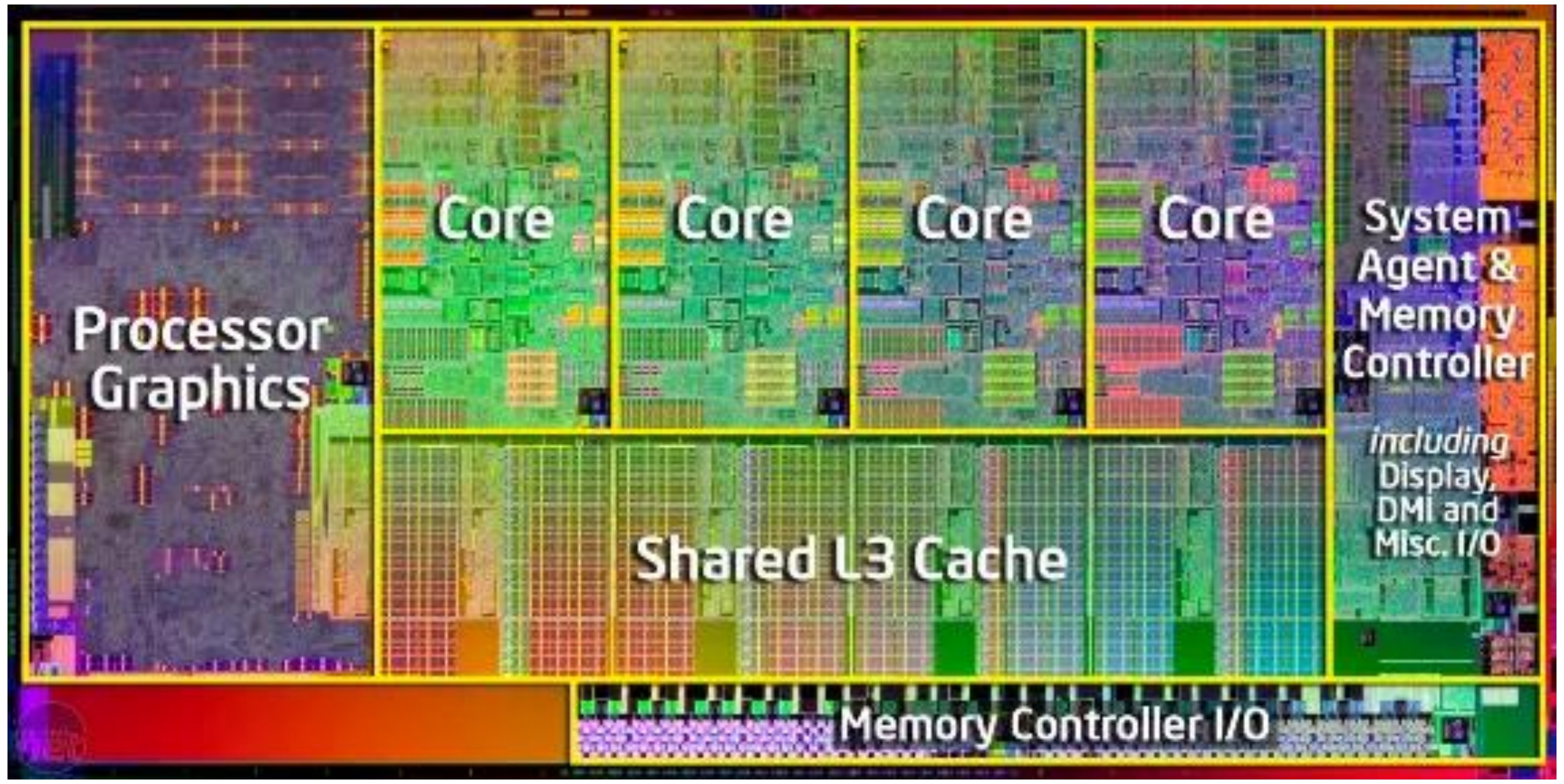
Big Data

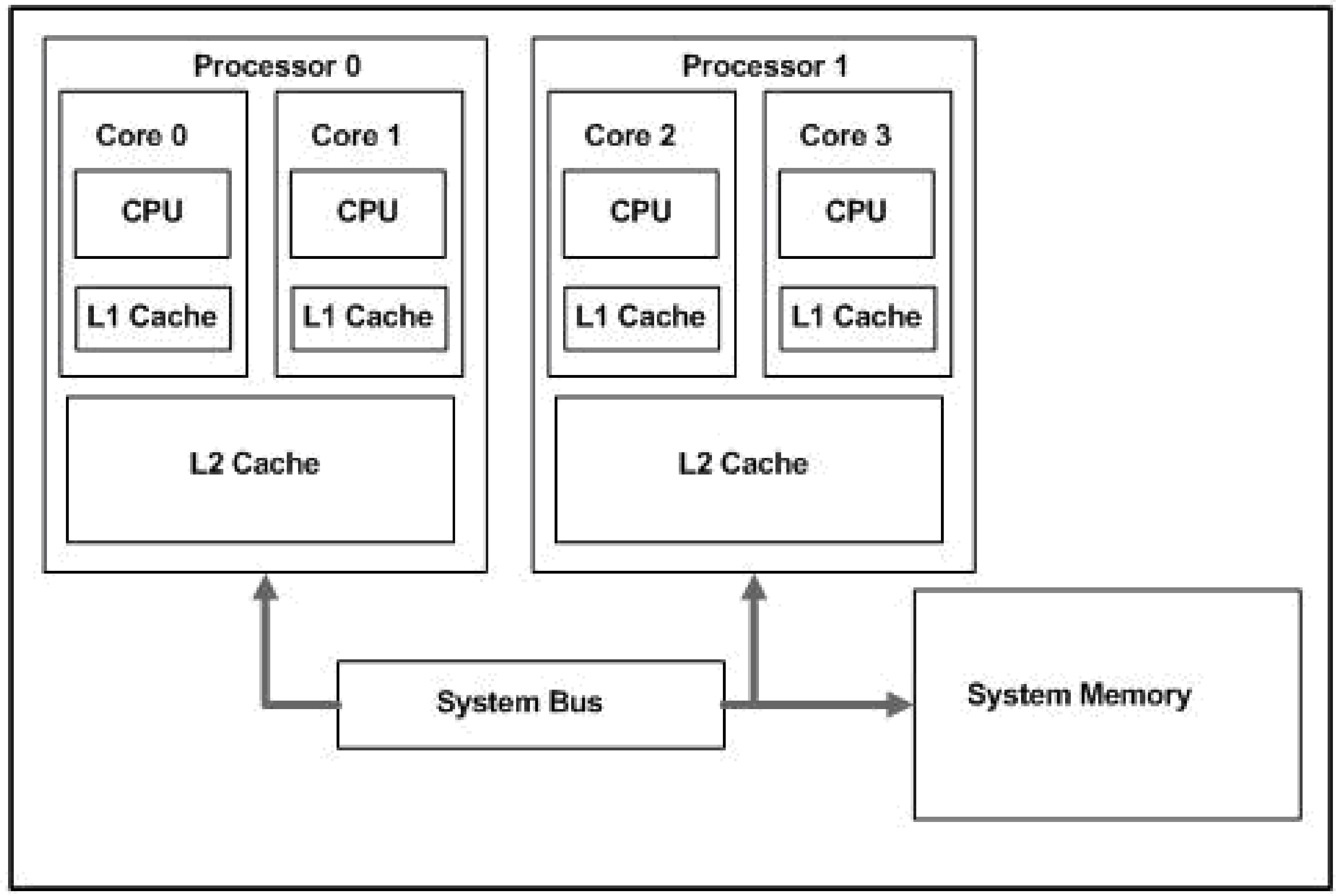
Cloud

Foundations of Data Systems

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:

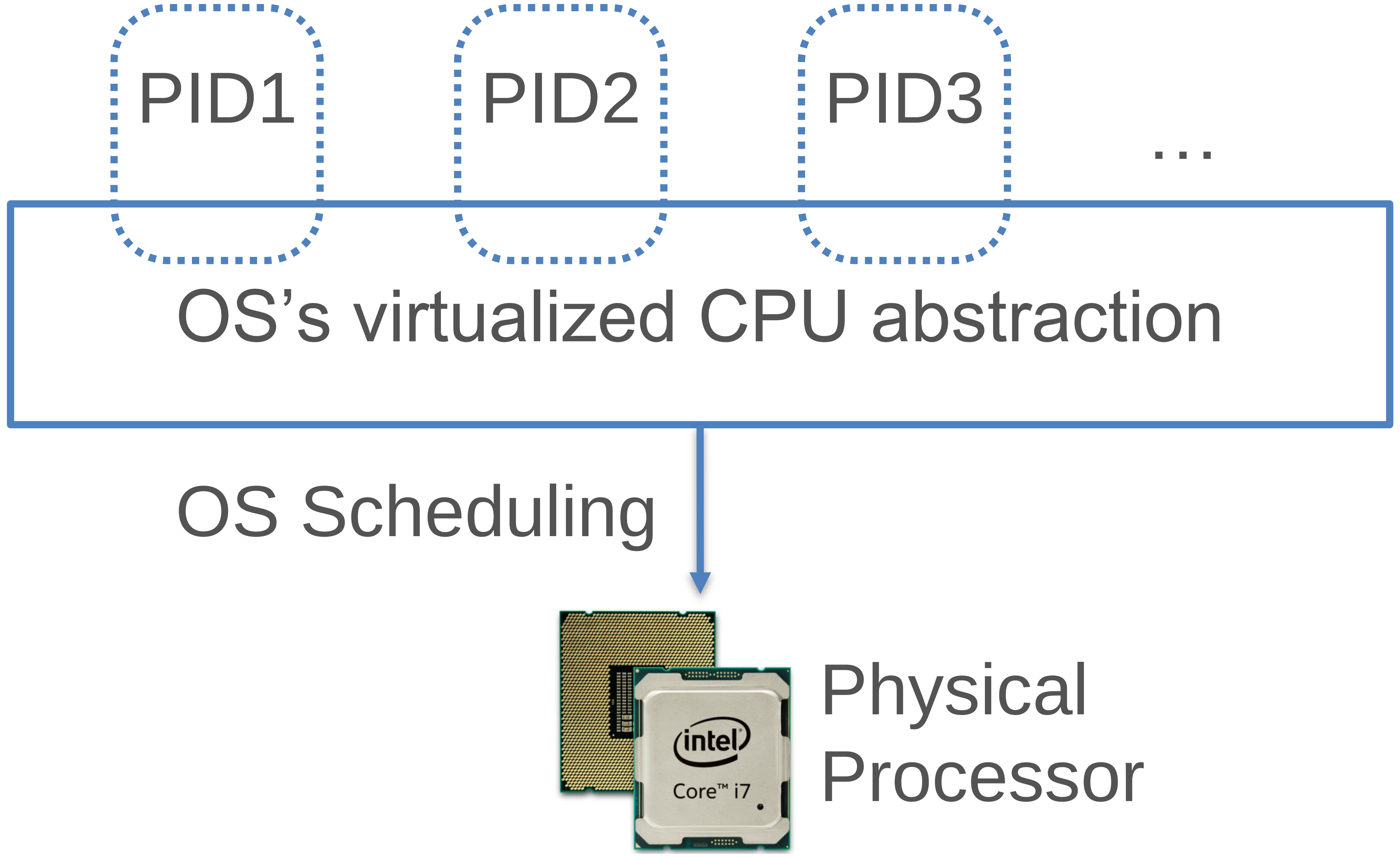
Q: how to place processes on each processor so **the objective** is optimal?

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

Inter-process communication (IPC)



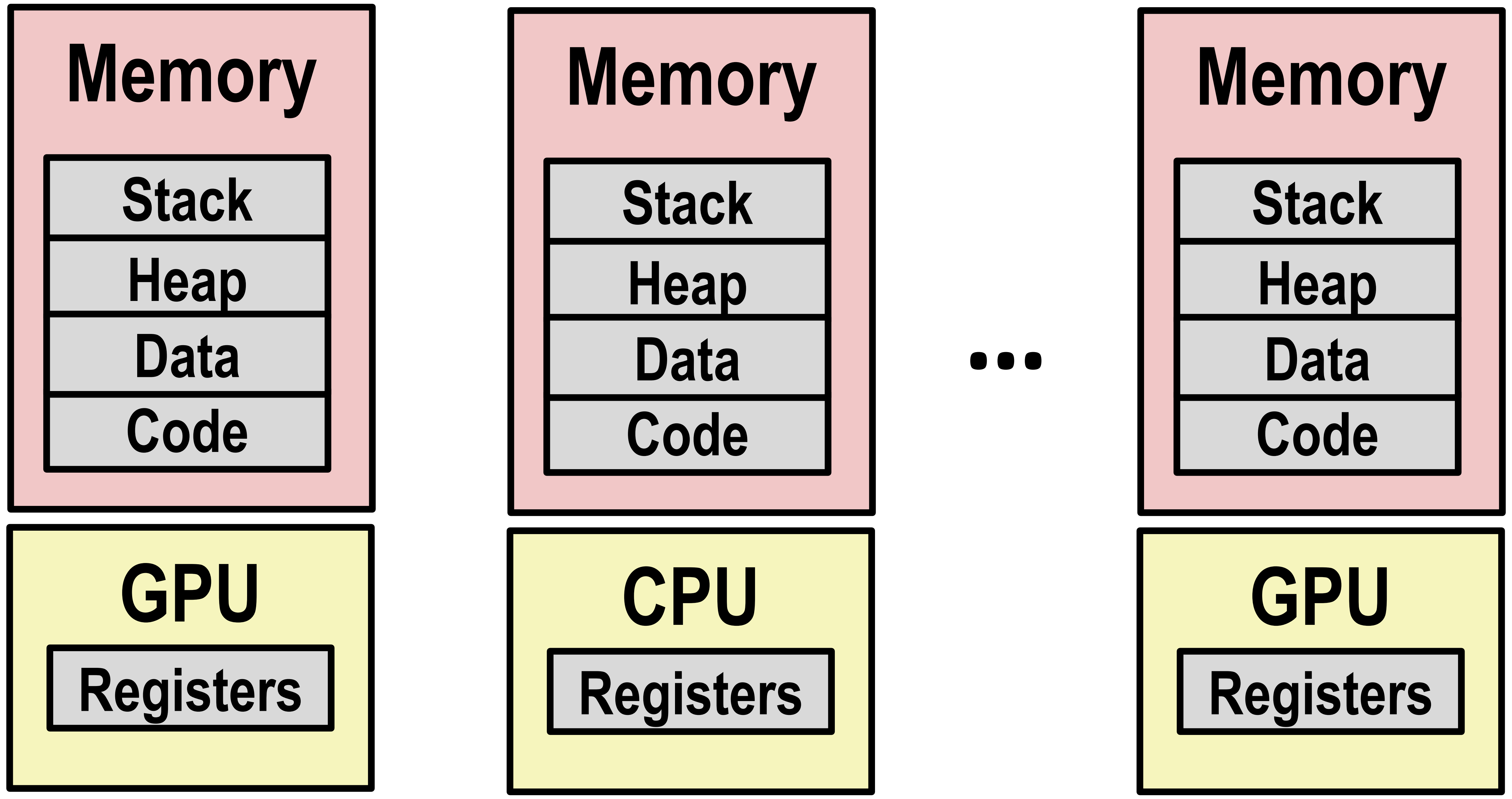
- Inter-process communication is provided in System Calls API

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

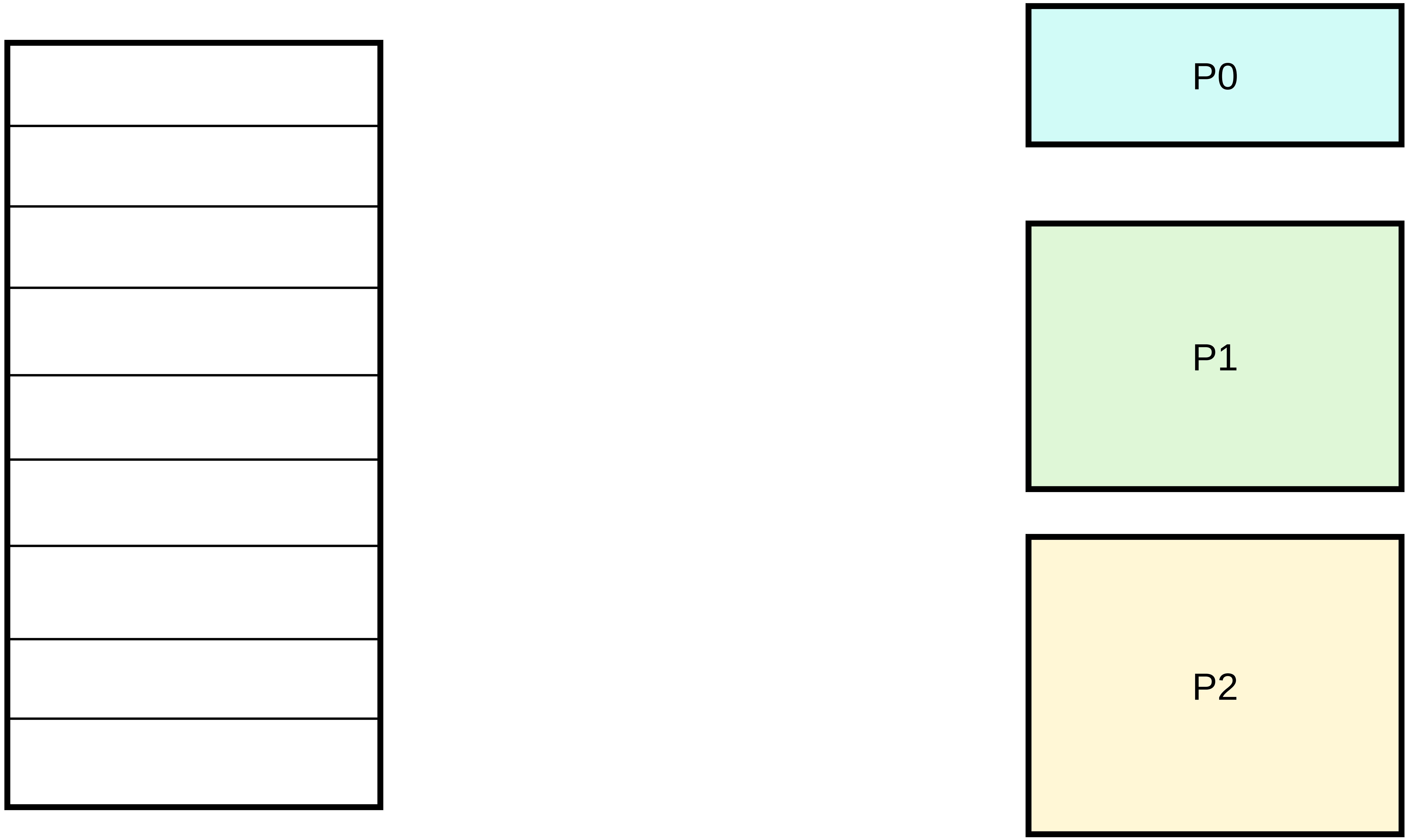
Mutliprocessing: memory management

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

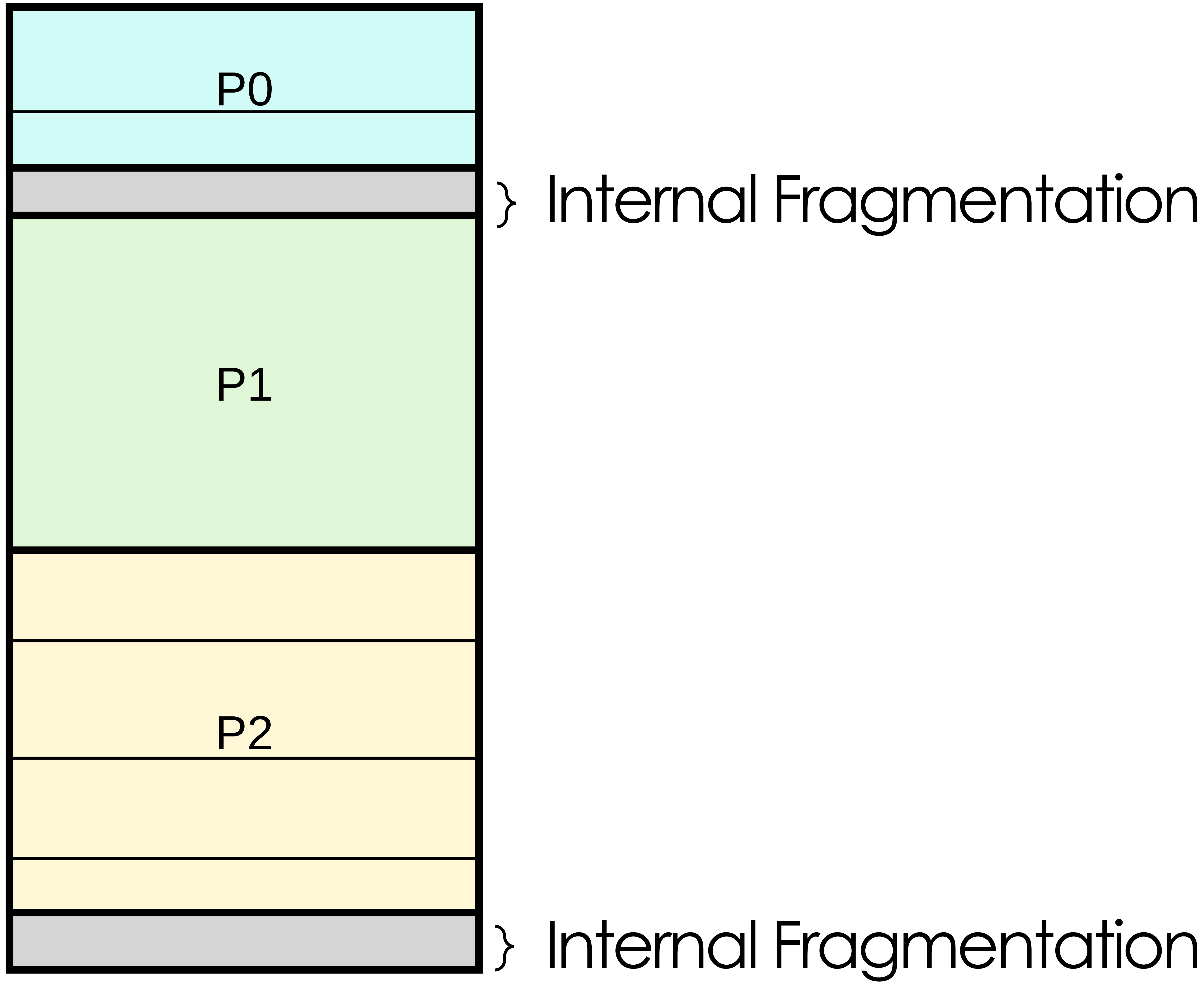


G1. Convenient?
G3: protection?
G2. Efficient?
- G2.1 can I run N processes but not N times slower?
- **G2.2 can I run N apps with total mem > physical memory cap**

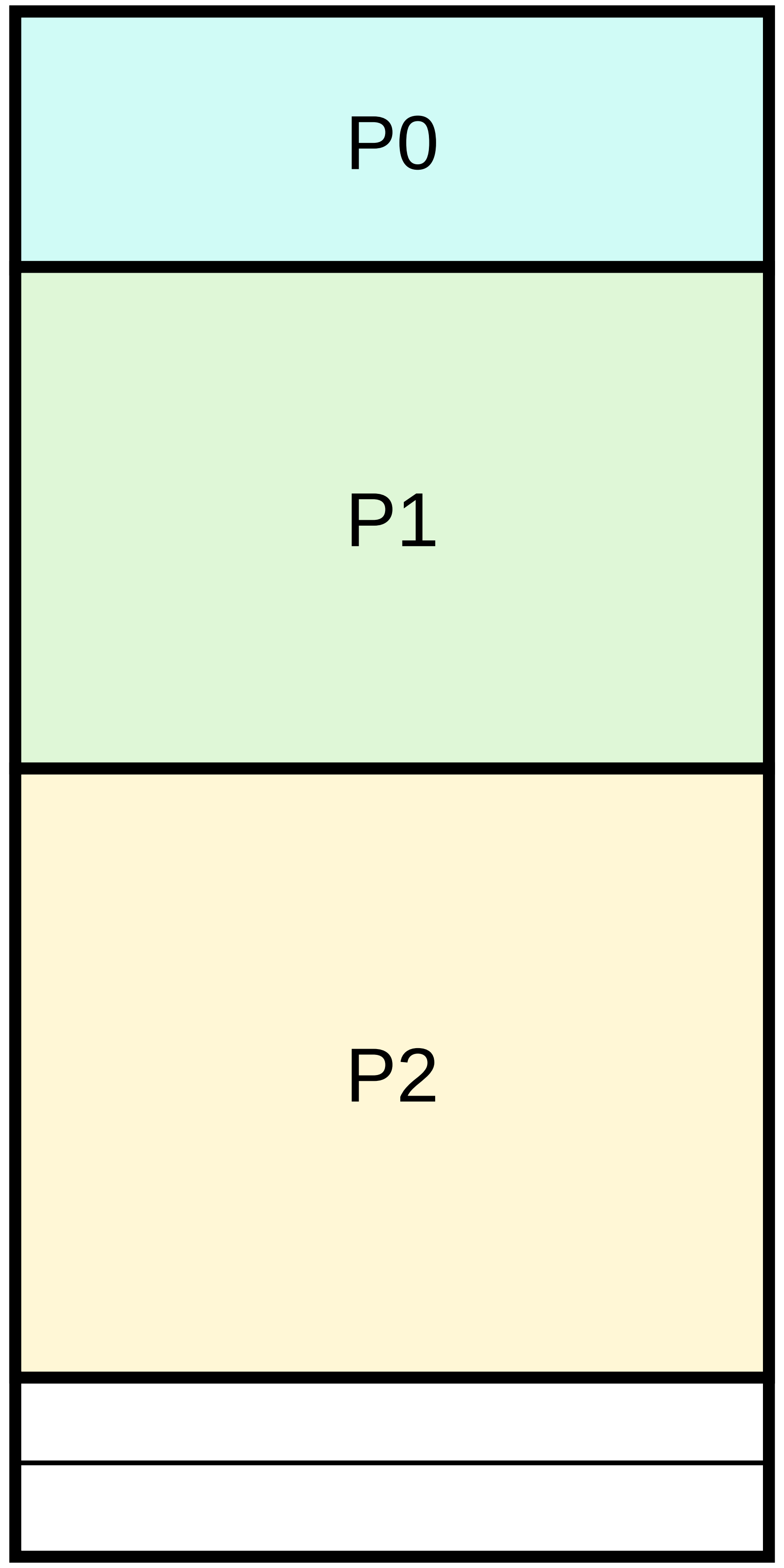
Memory management v0



Memory management v0: Internal fragmentations

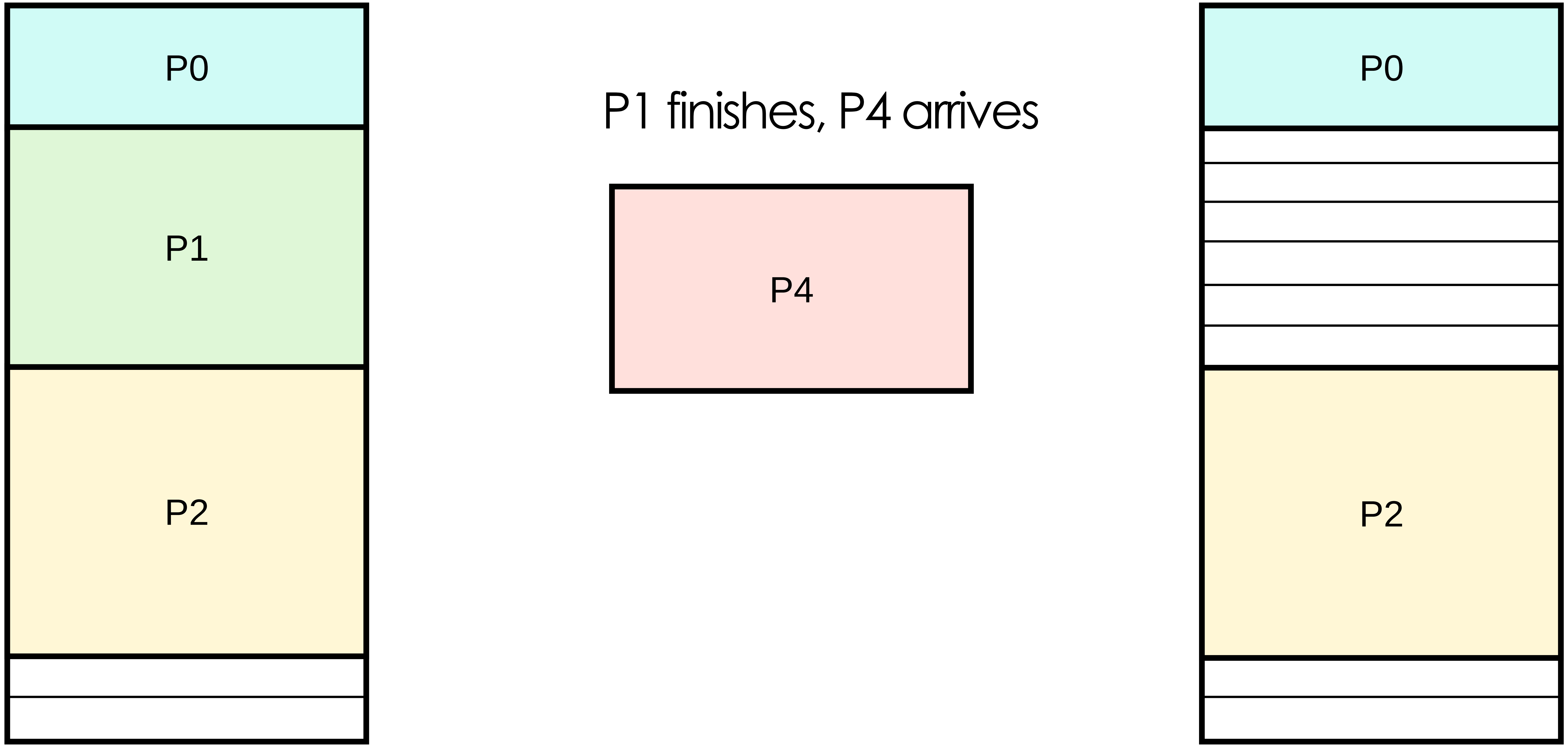


Memory management v1: use a smaller chunk

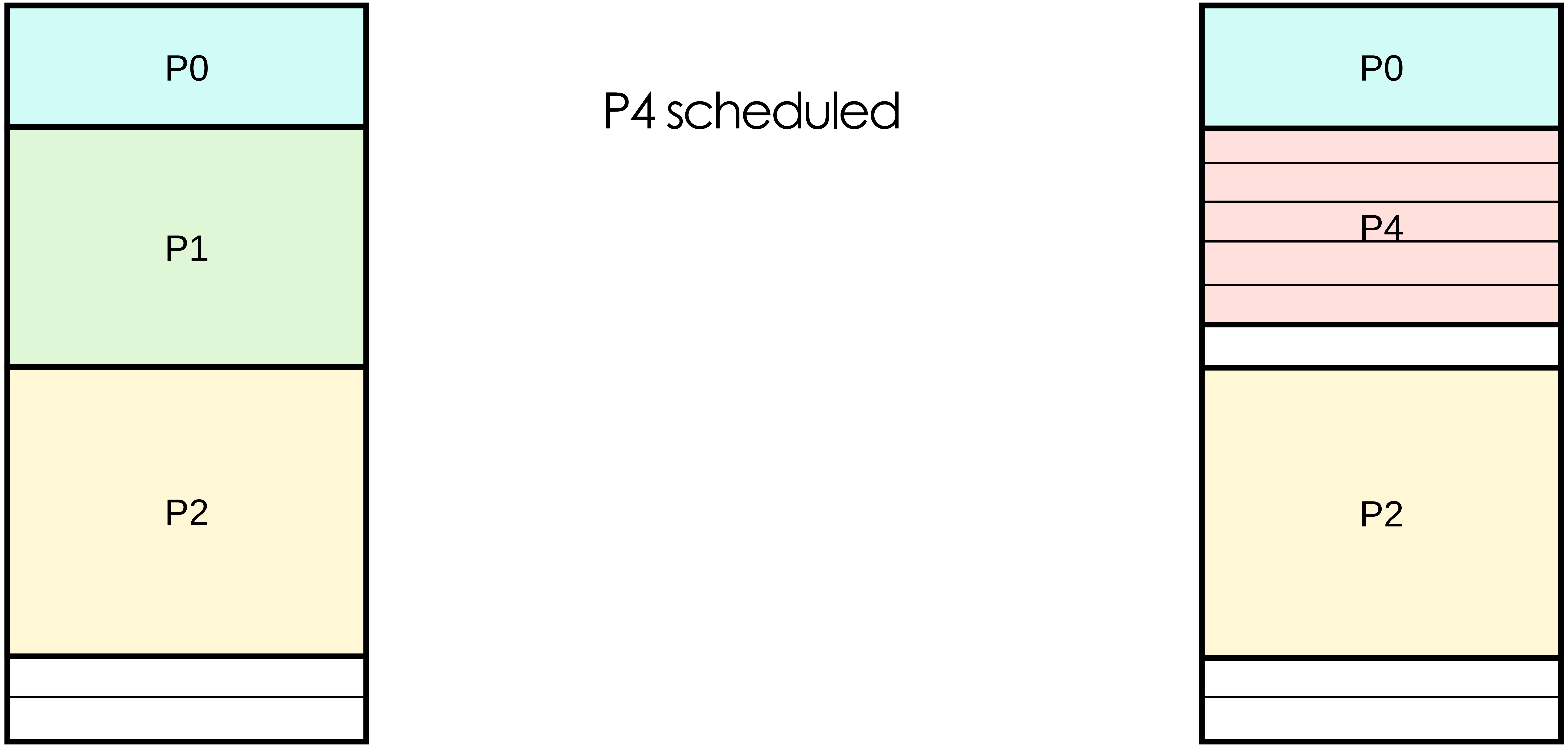


Q: What is the maximum possible amount of internal fragmentation per process?

Memory management v1

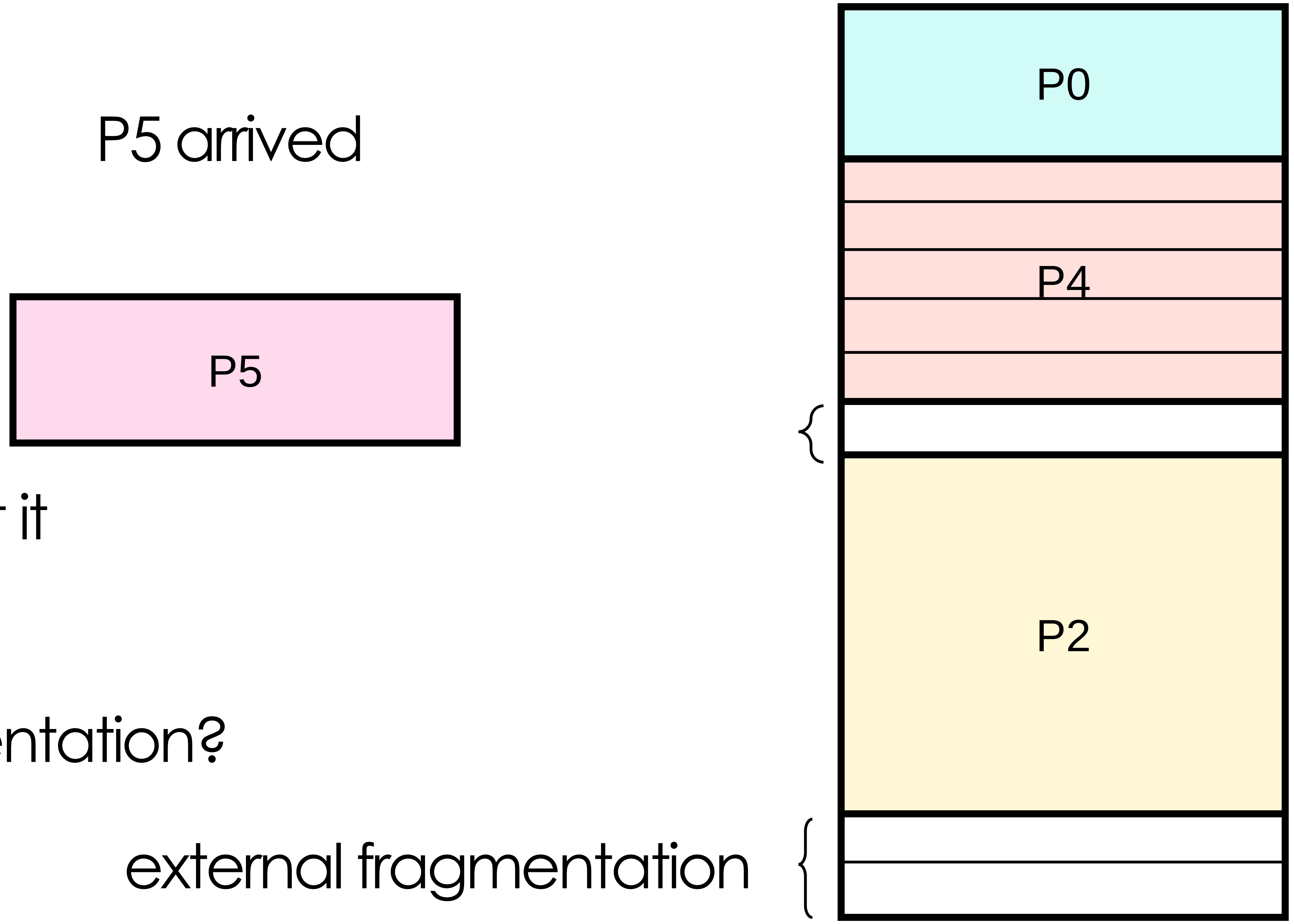


Memory: v2



Memory: v2

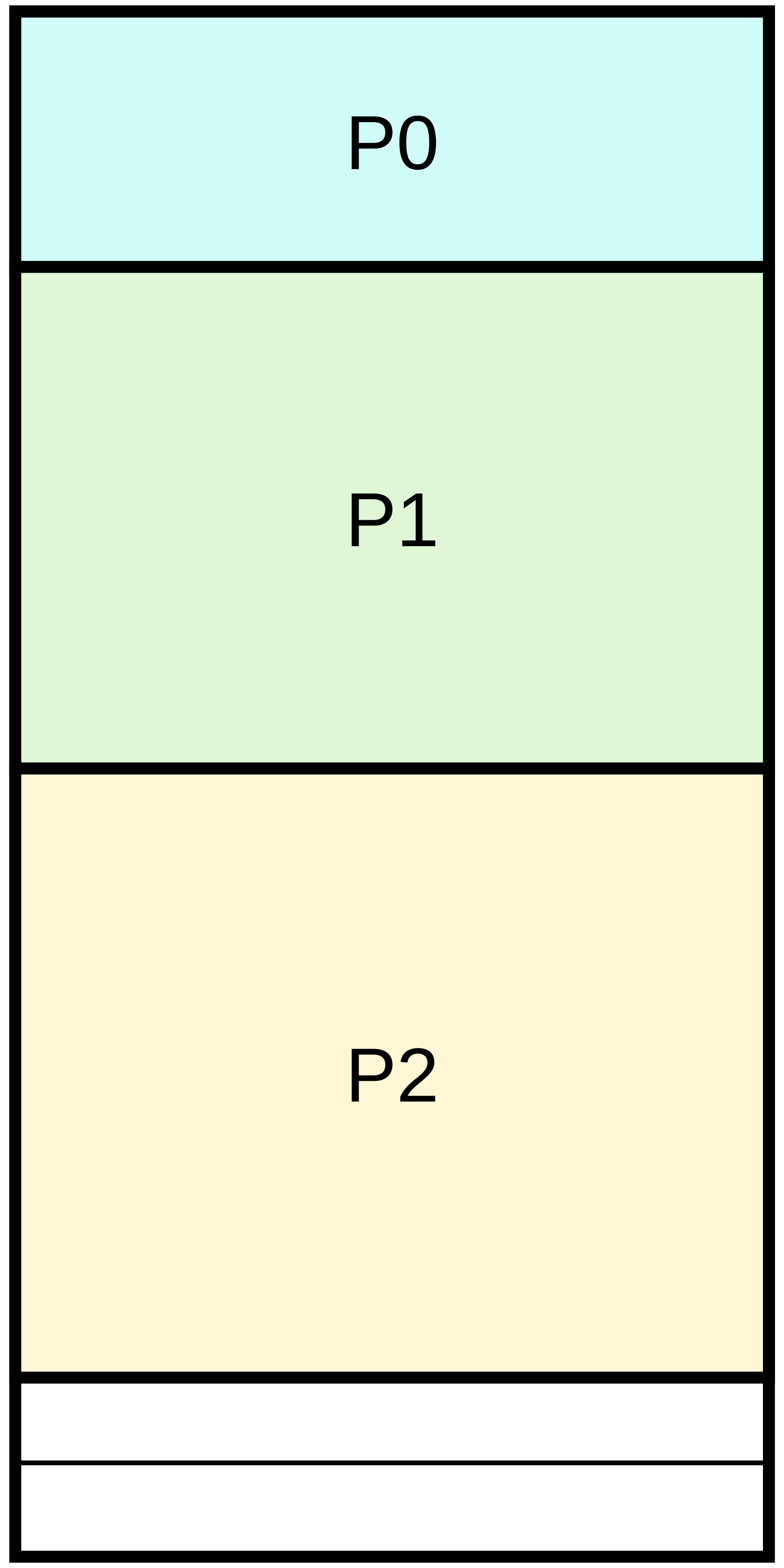
P5 arrived



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

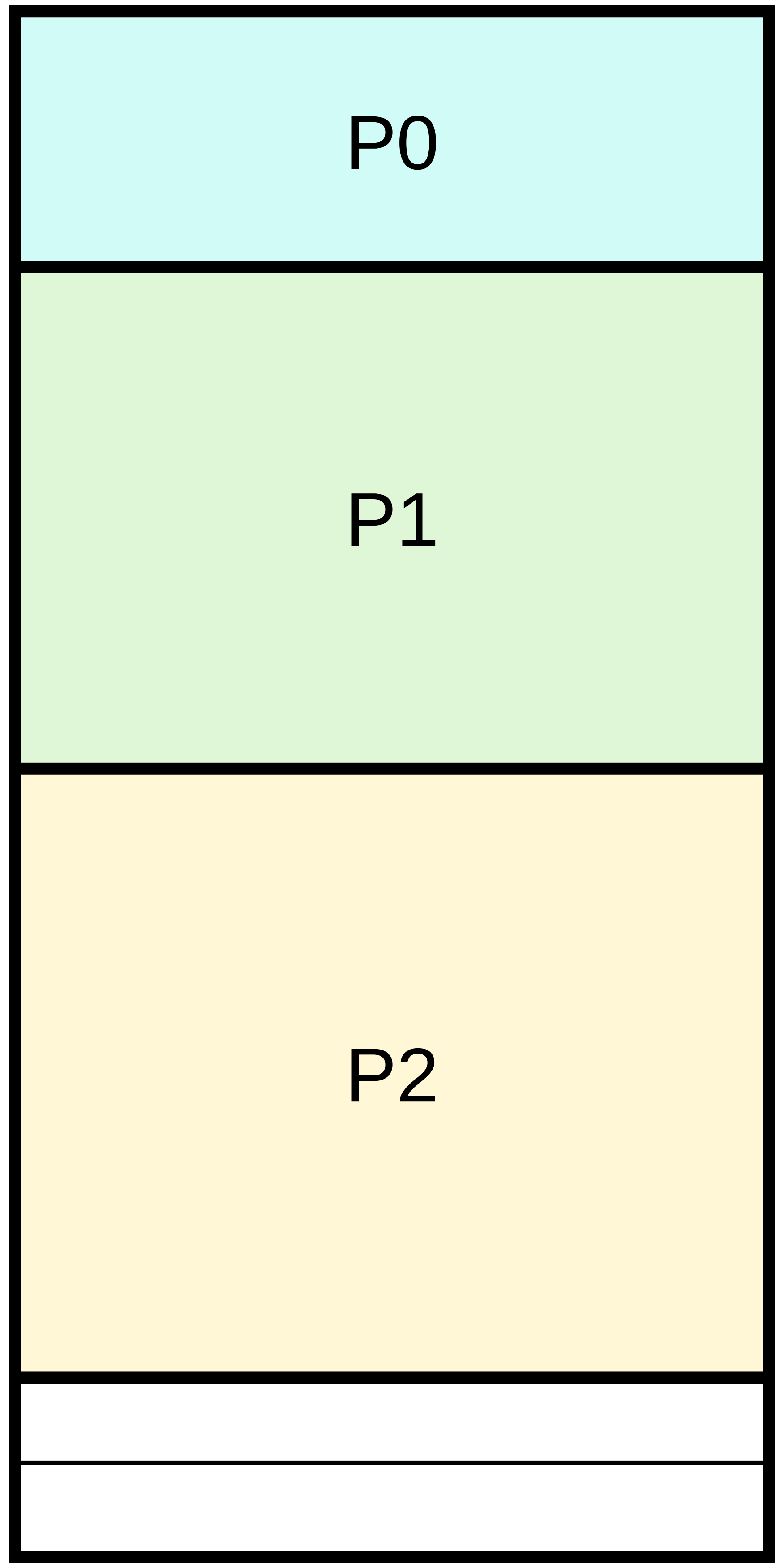
Q: How to address external fragmentation?

Other Problems?



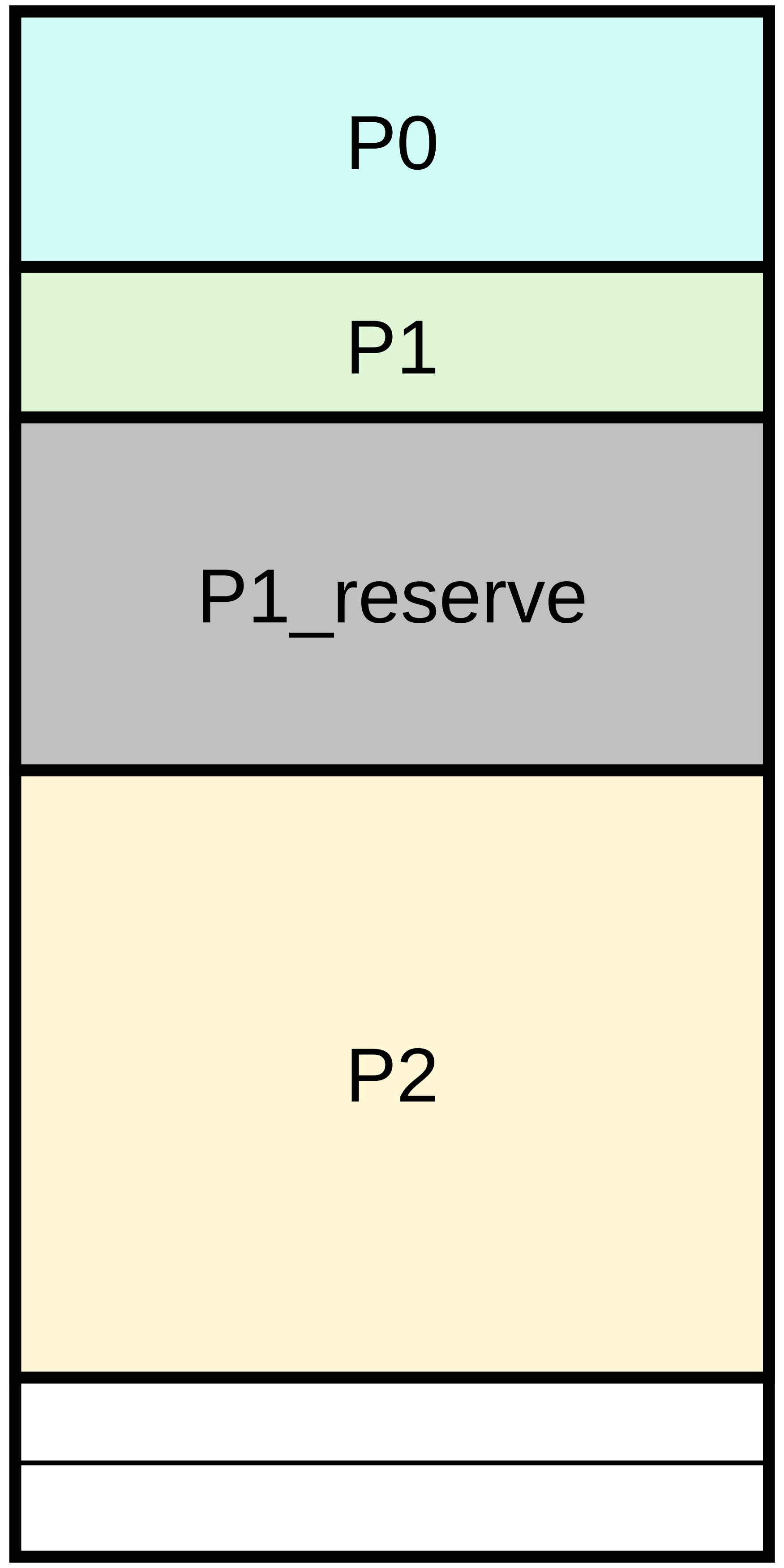
Problem: We can never schedule processes with their memory consumption greater than memory cap

Other Problems?



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

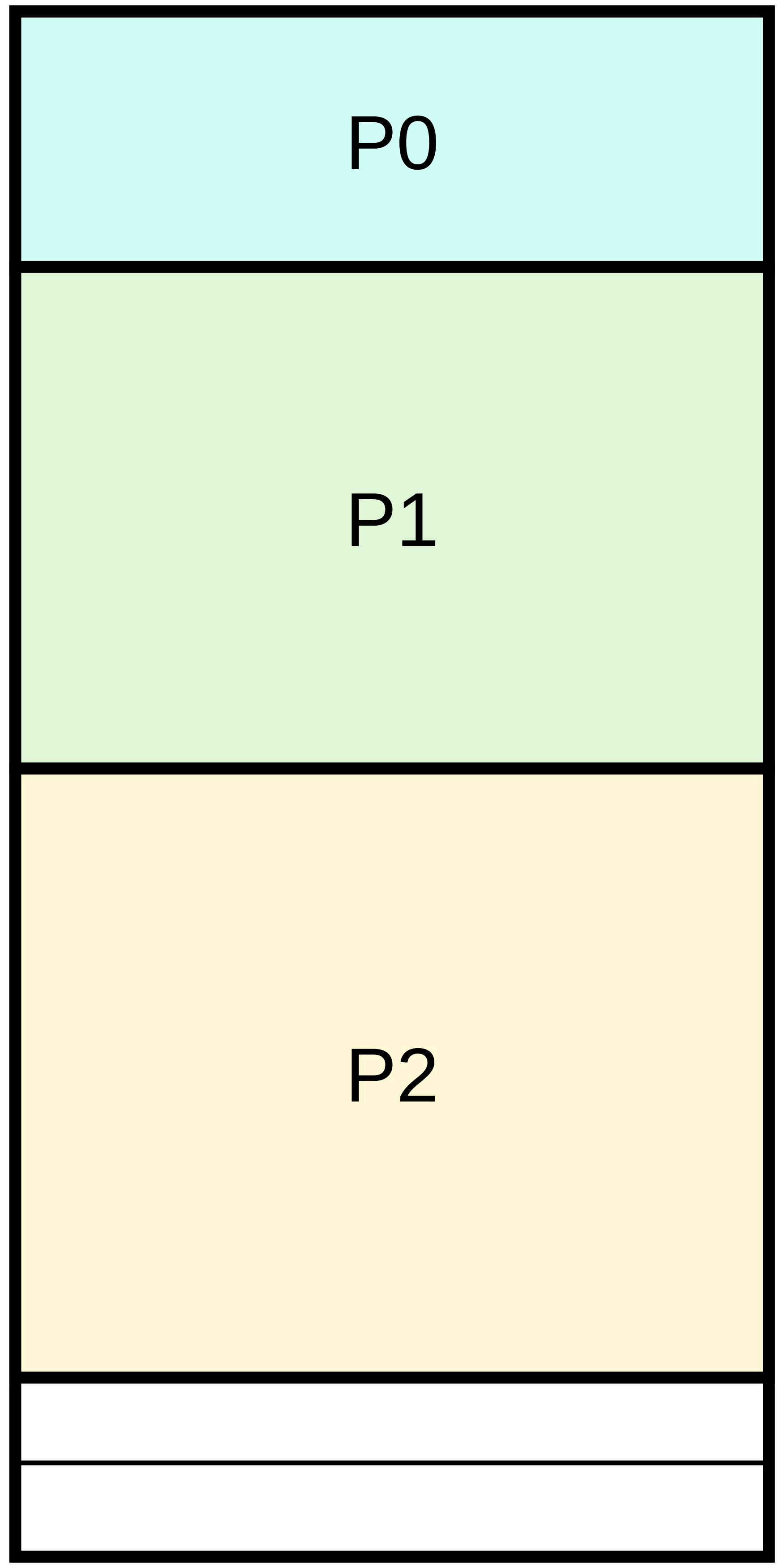
Other Problems?



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

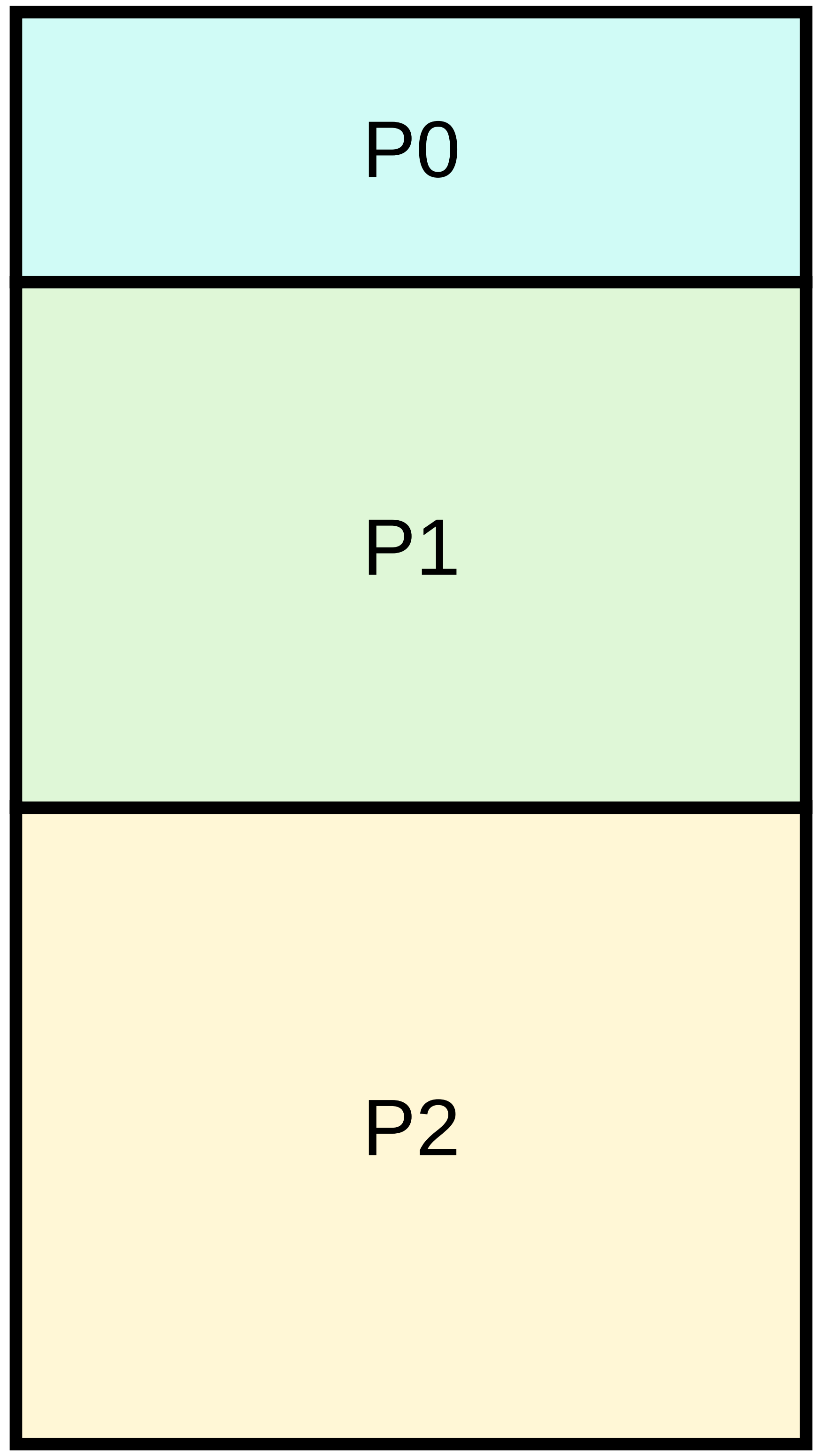
P1_reserve is the reservation overhead

Other Problems?

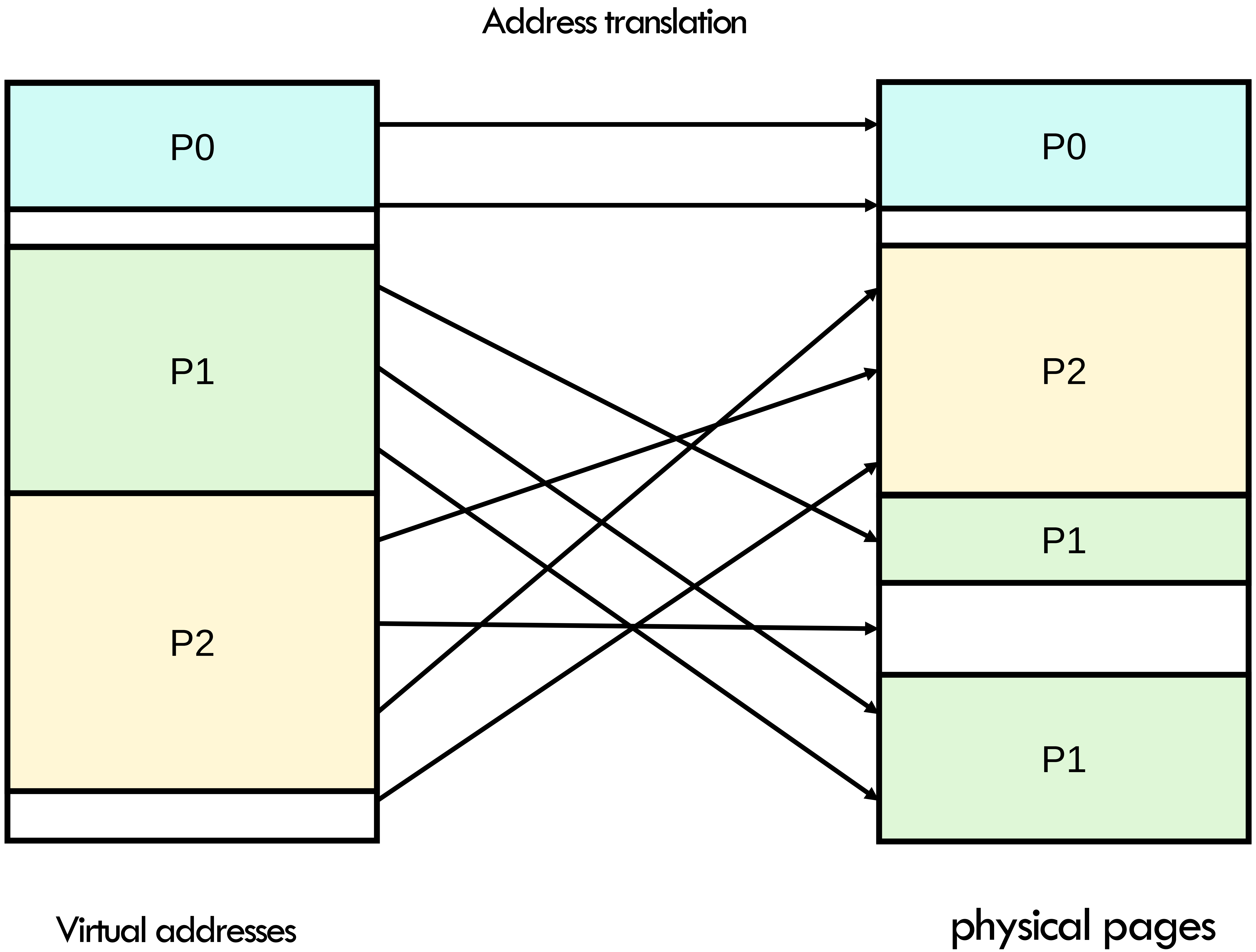


What if we **know exactly** how much memory P0/P1/P2 will **eventually** use, any problem?

Virtual Address Table



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



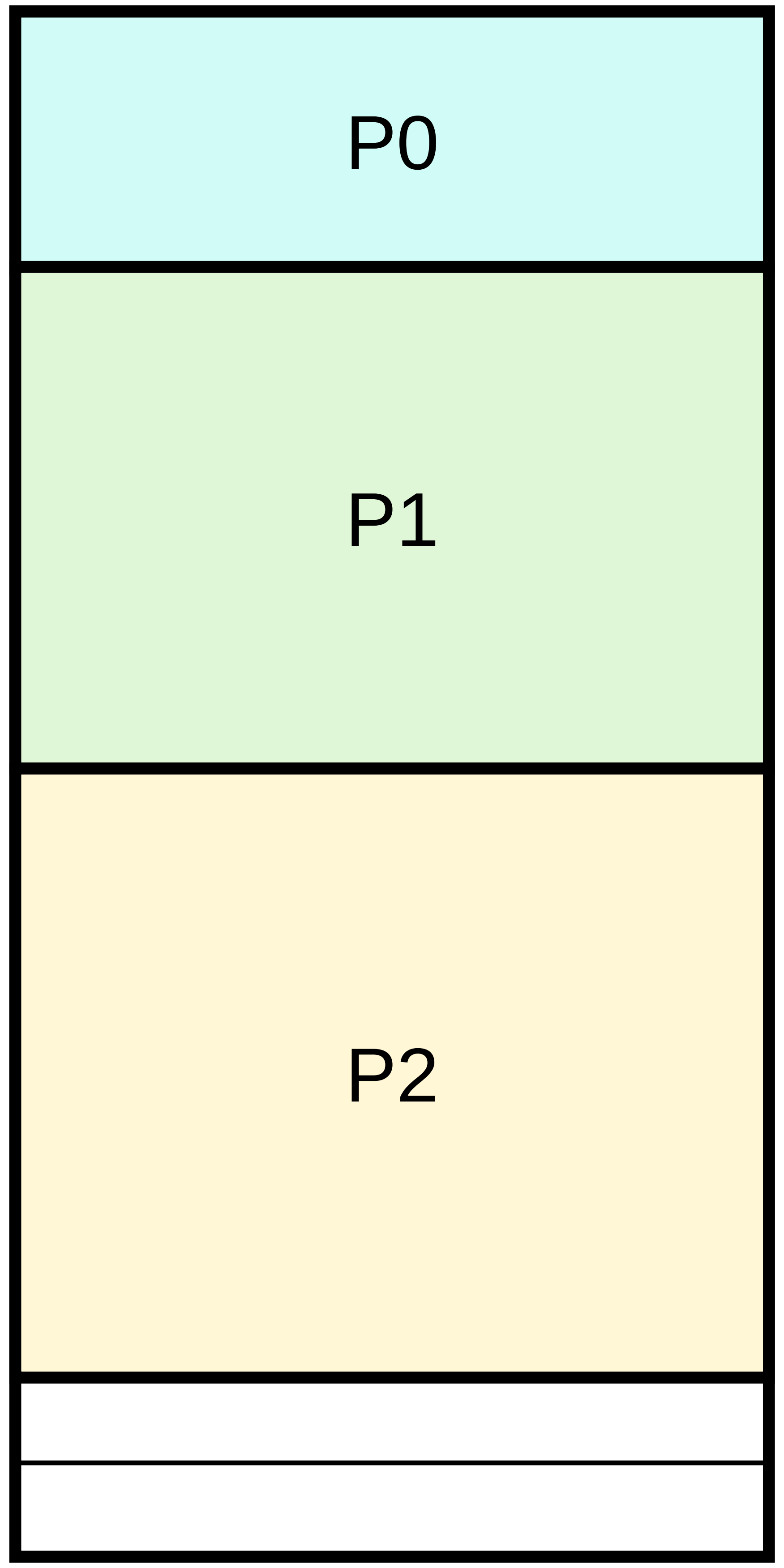
Pages and virtual memory

- **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.

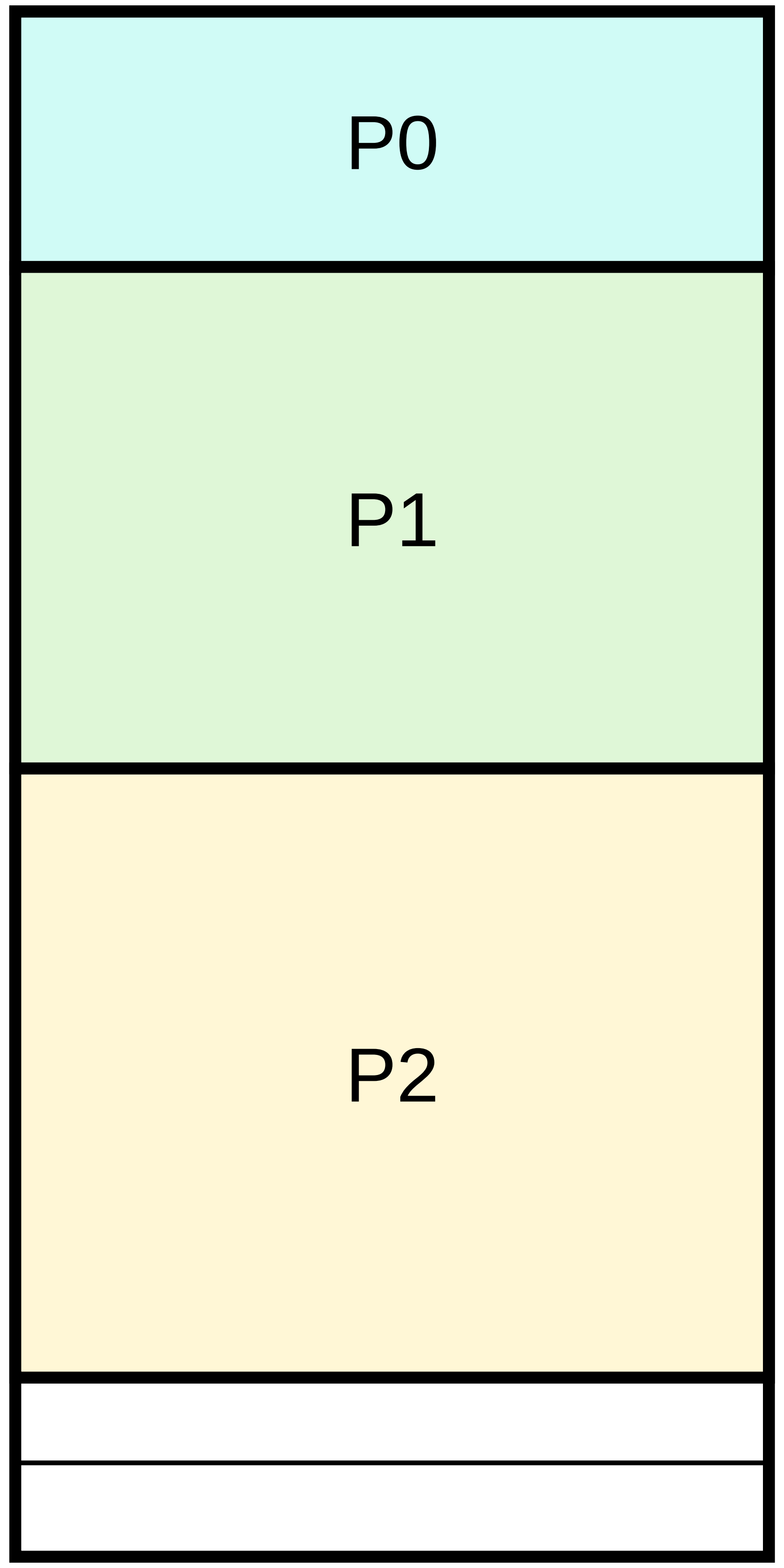
Problem addressed?



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.

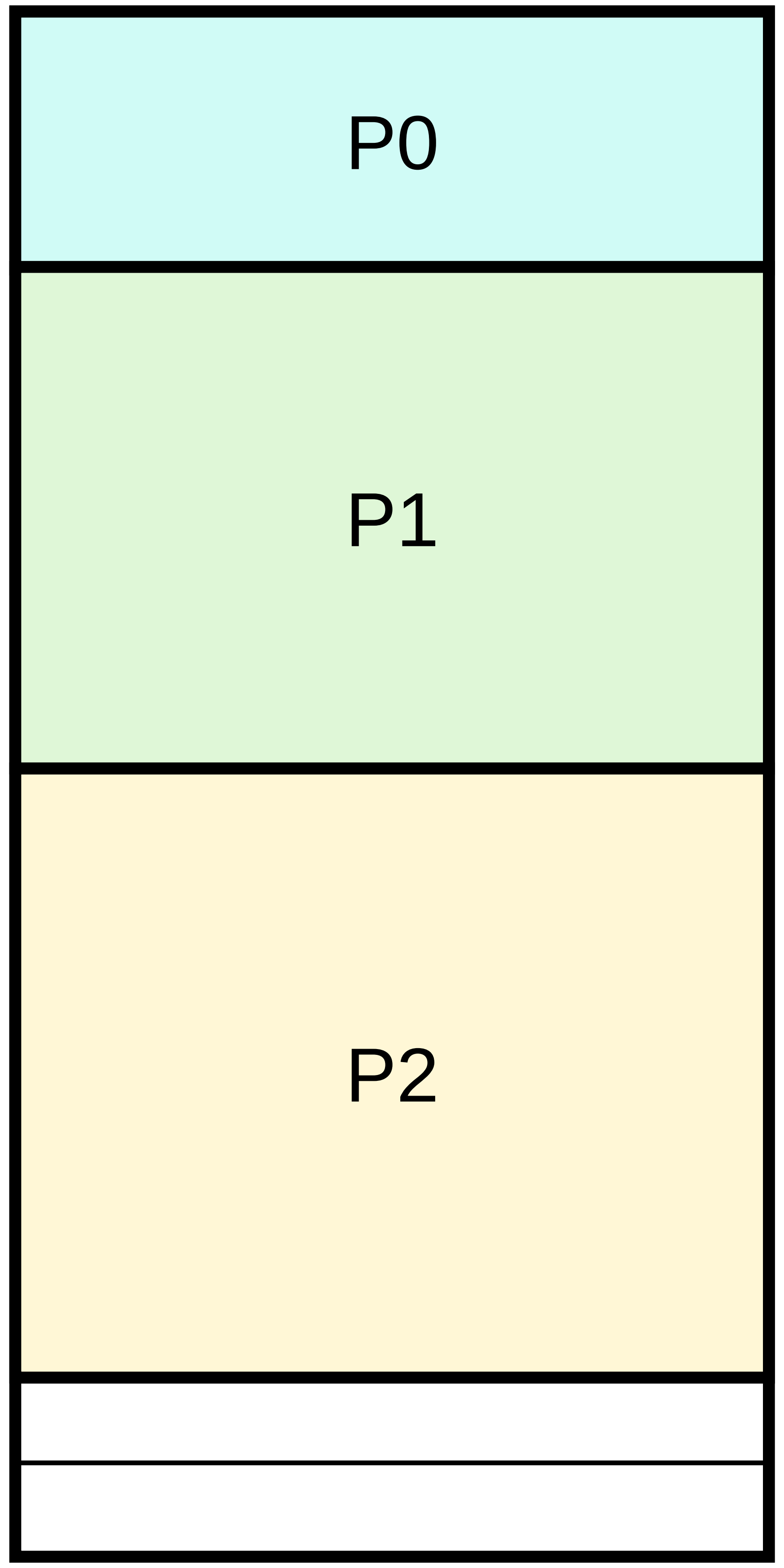
Problem addressed?



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

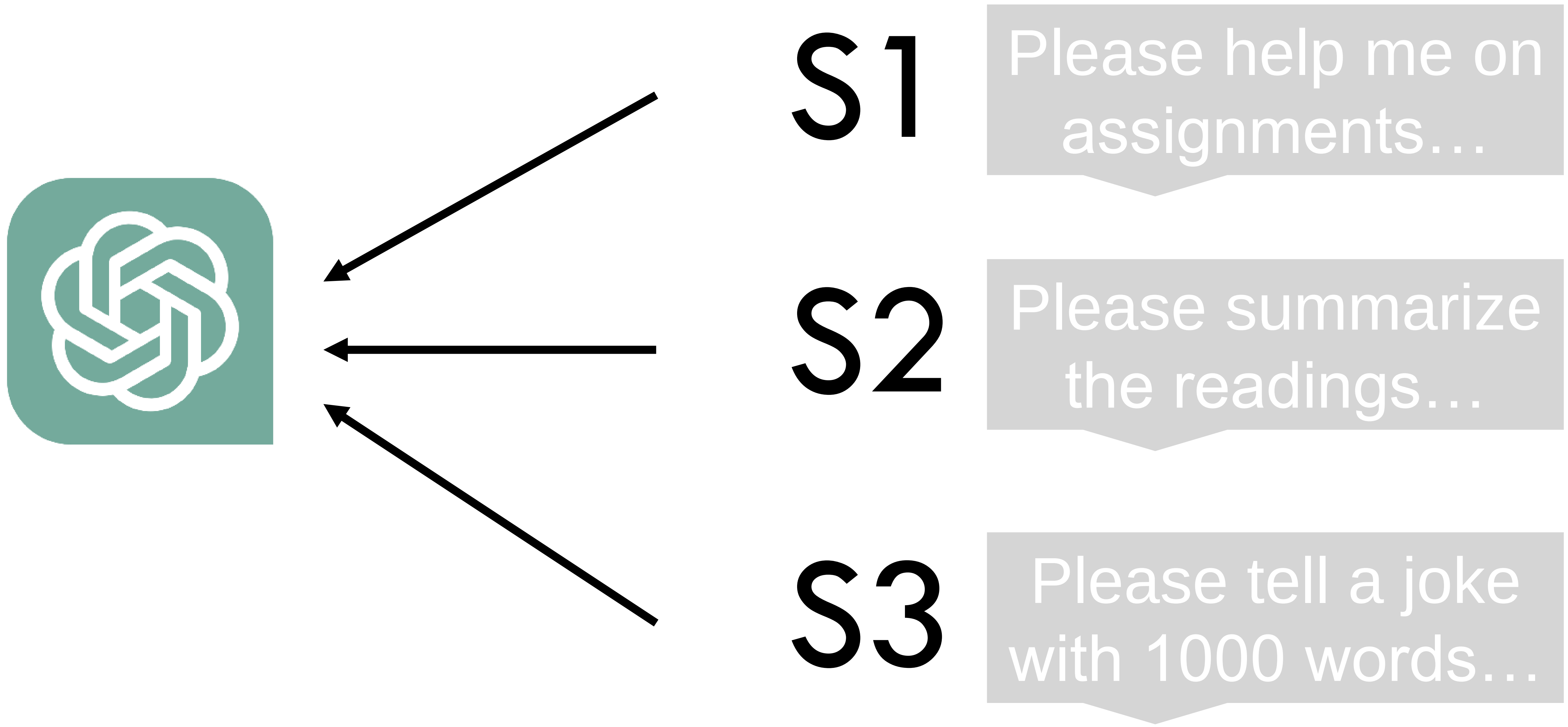
Problem addressed?



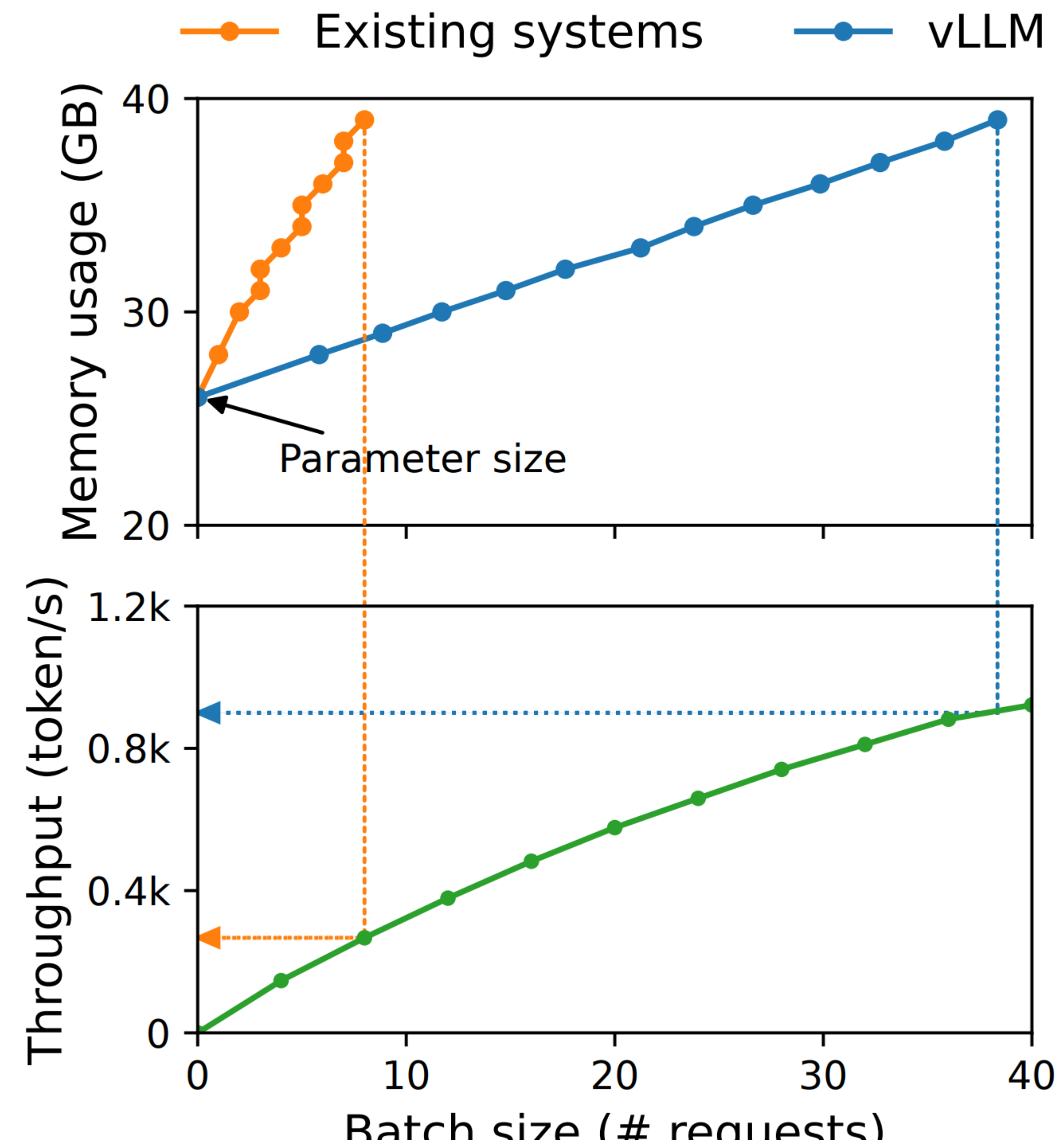
What if we **know exactly** how much memory P0/P1/P2 will **eventually** use, any problem?

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

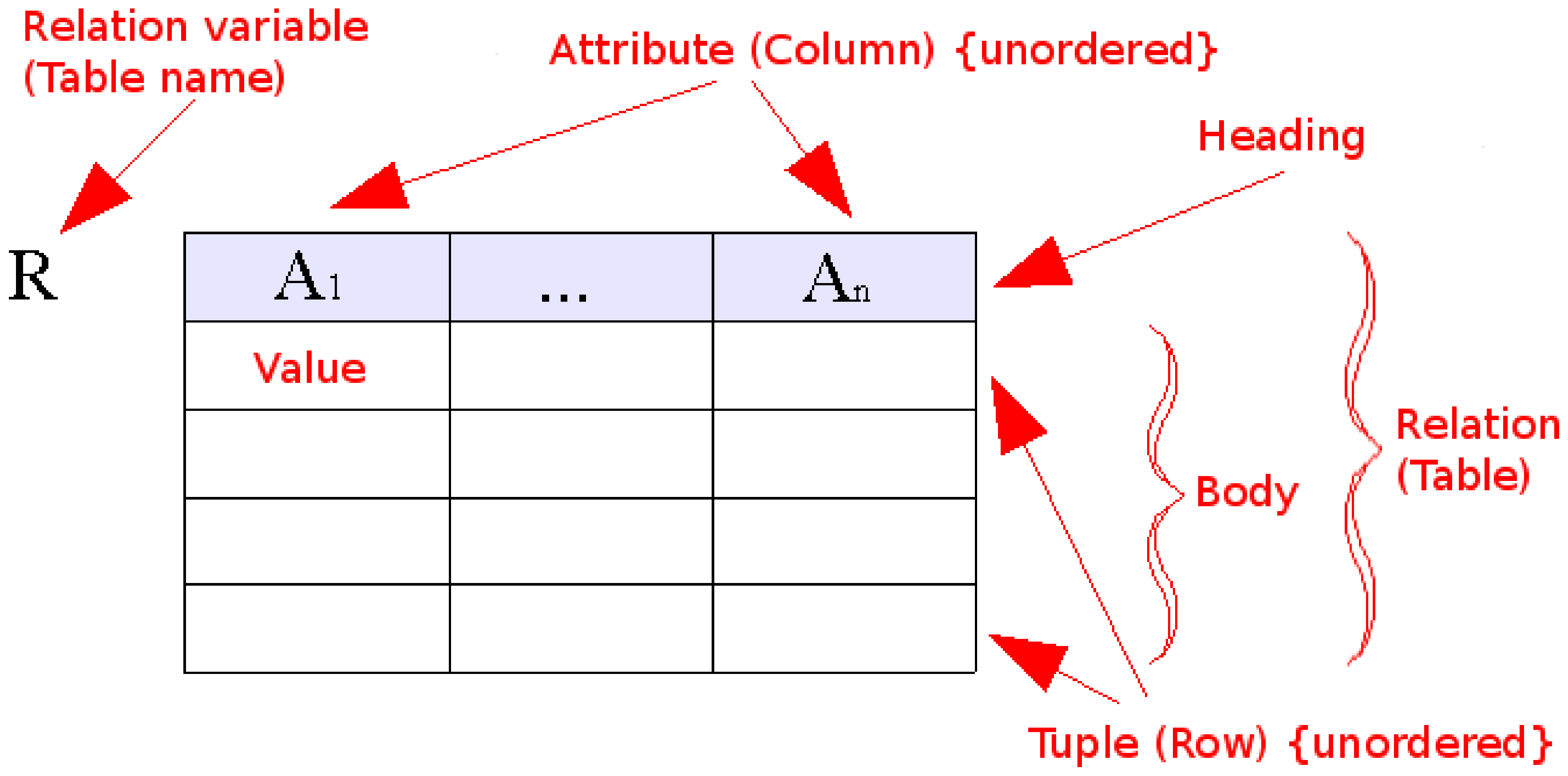
Files Vs Databases: Data Model

- 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

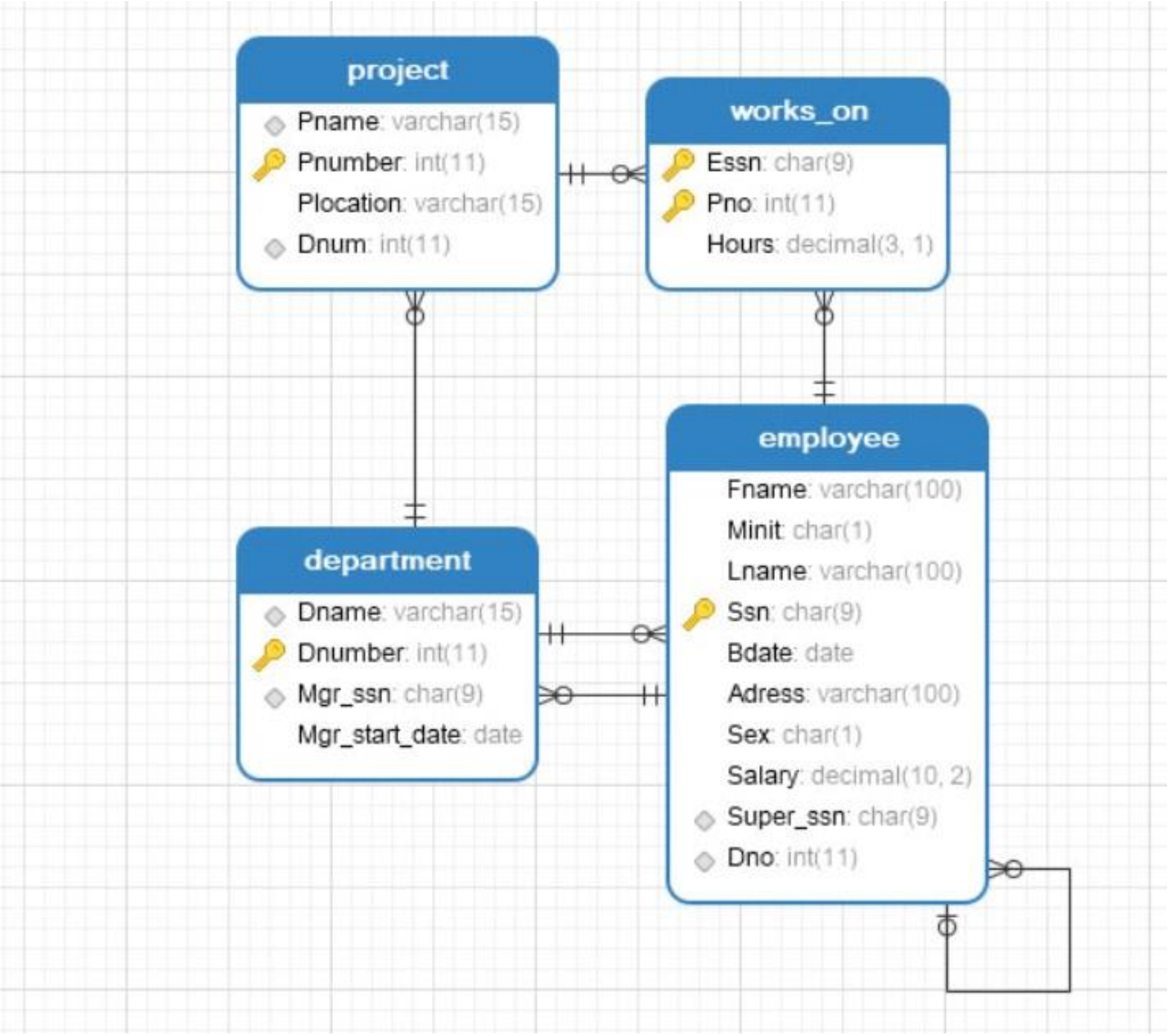
Data as File: Structured

- **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

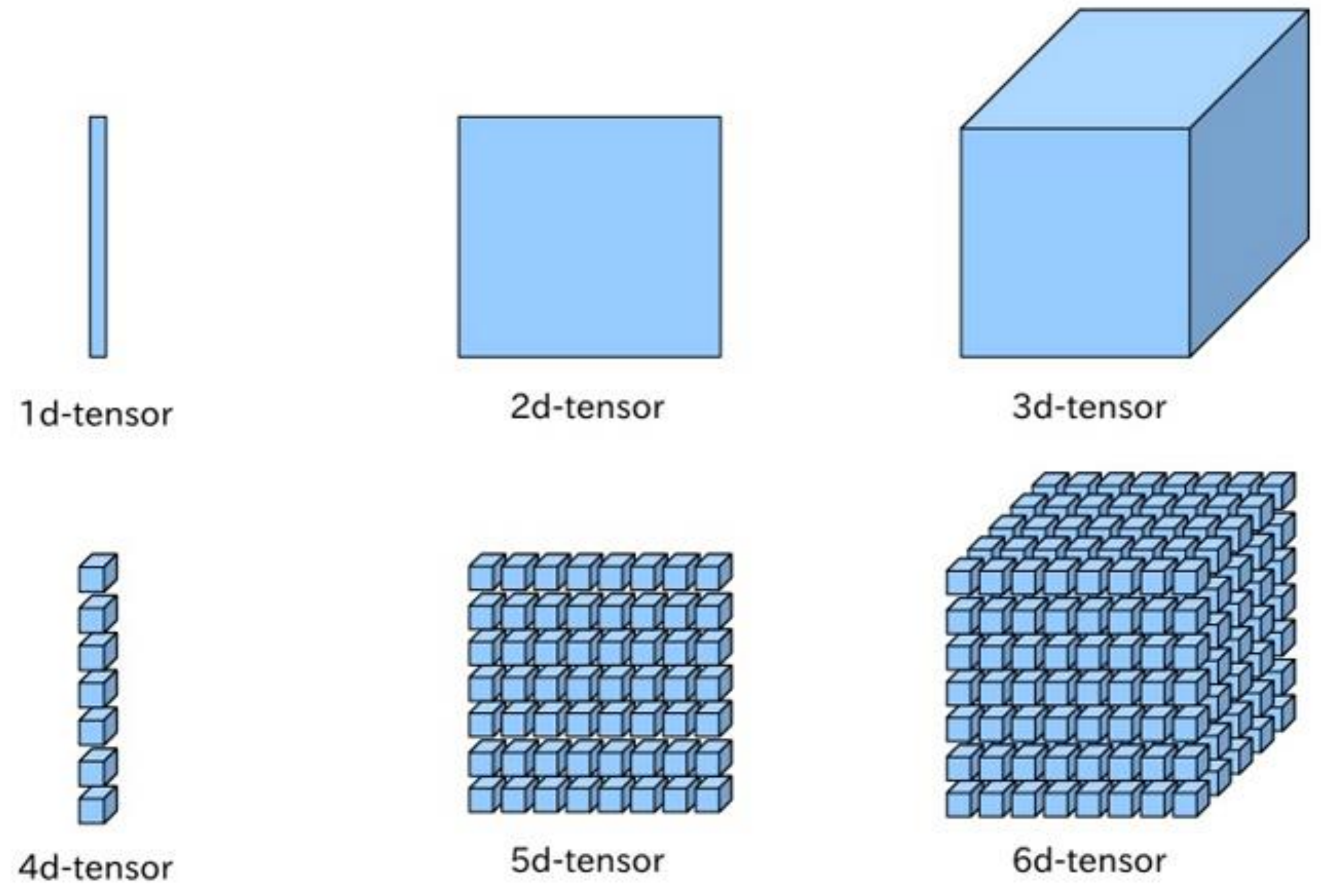
Data as File: Structured

- Structured Data: A form of data with regular substructure

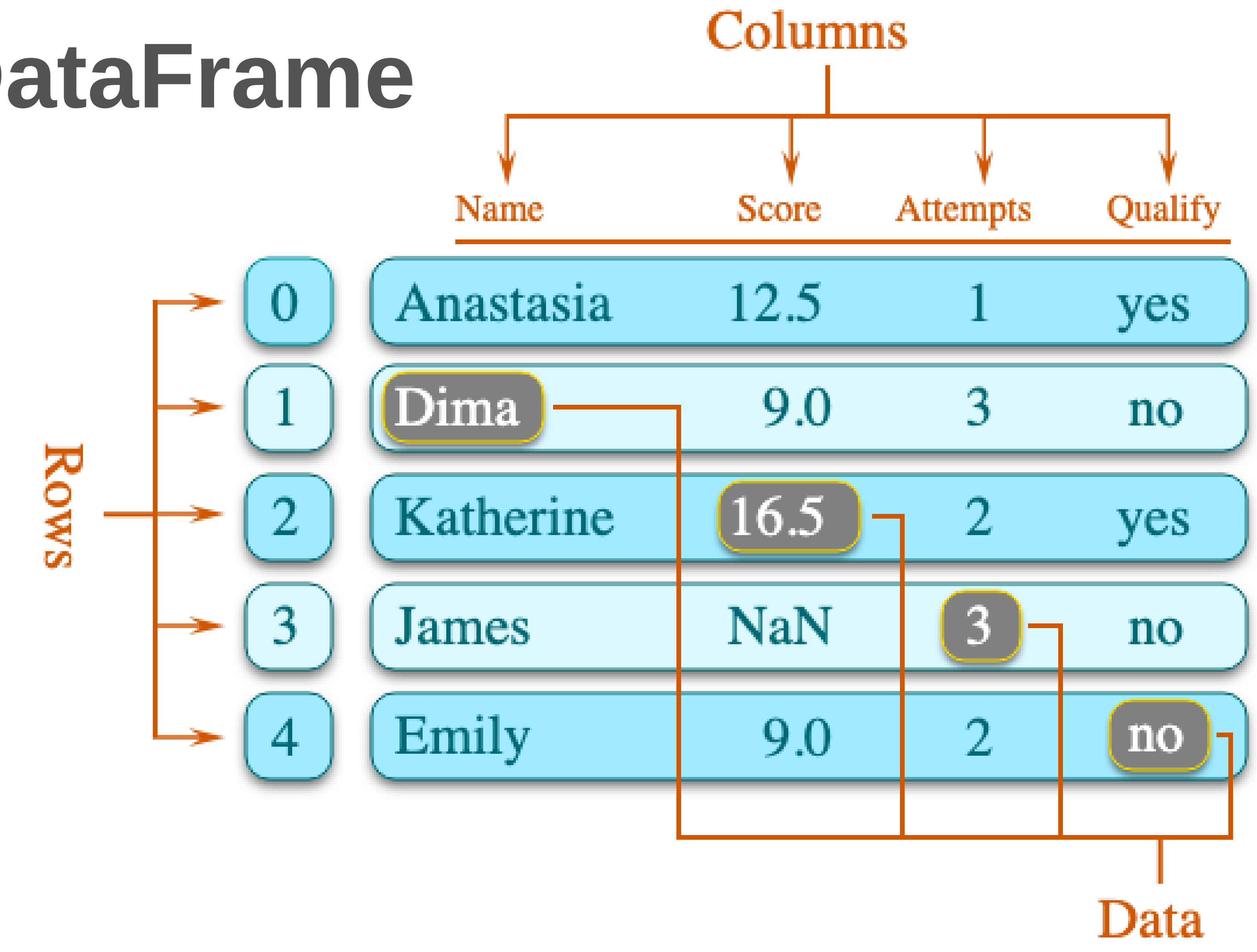
Matrix

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

Tensor



DataFrame

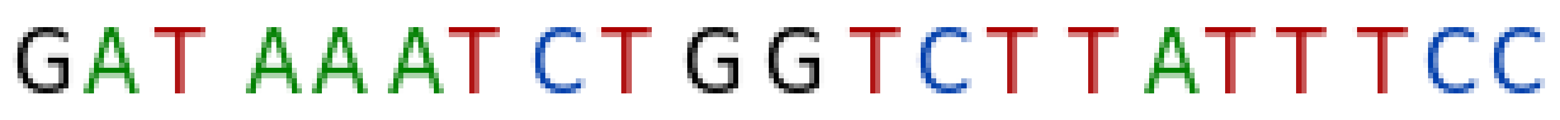


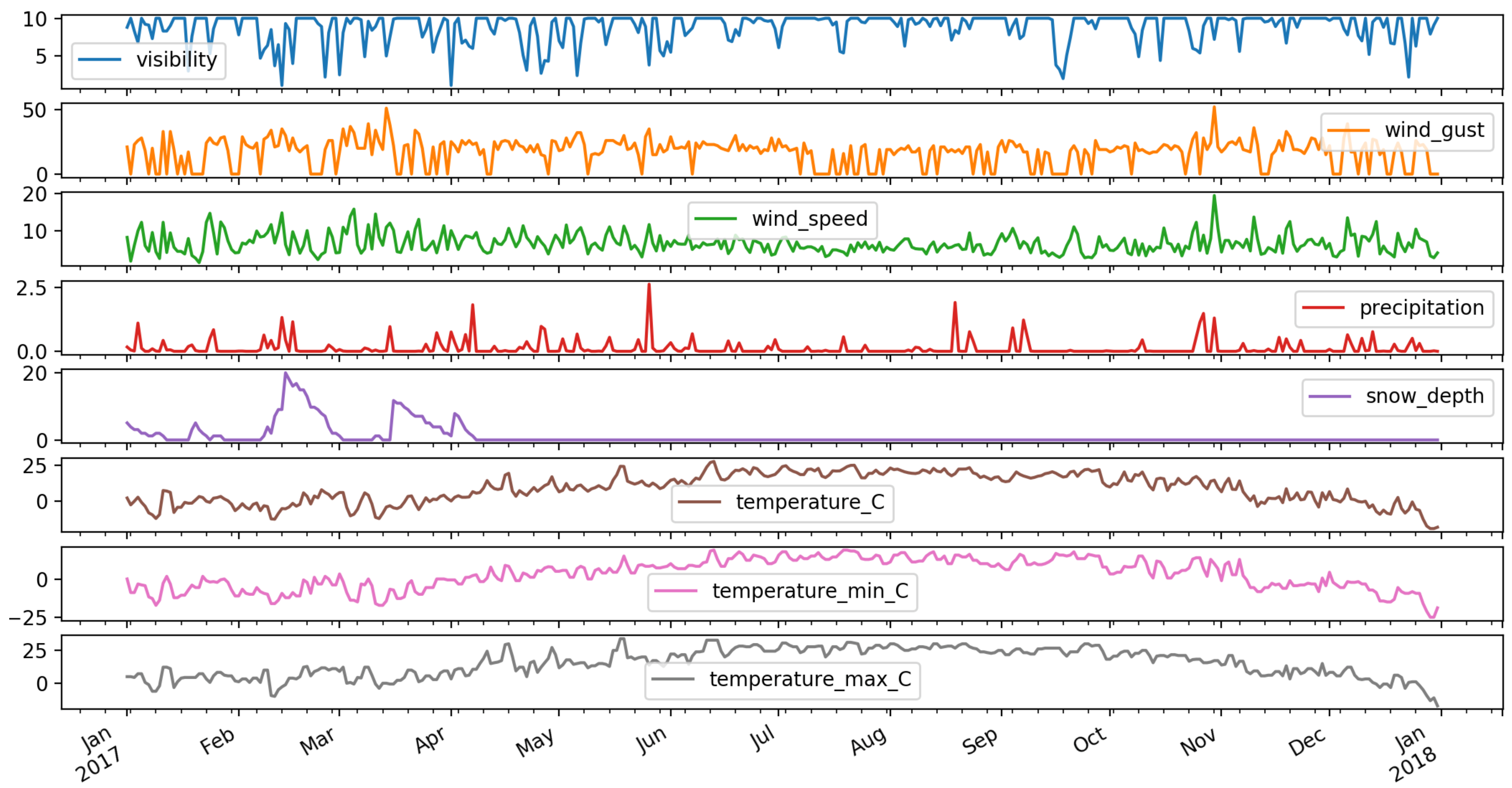
- Typically serialized as restricted ASCII text file (TSV, CSV, etc.)
- Matrix/tensor as binary too
- Can layer on Relations too!

Data as File: Structured

- Structured Data: A form of data with regular substructure

Sequence (Includes Time-series)

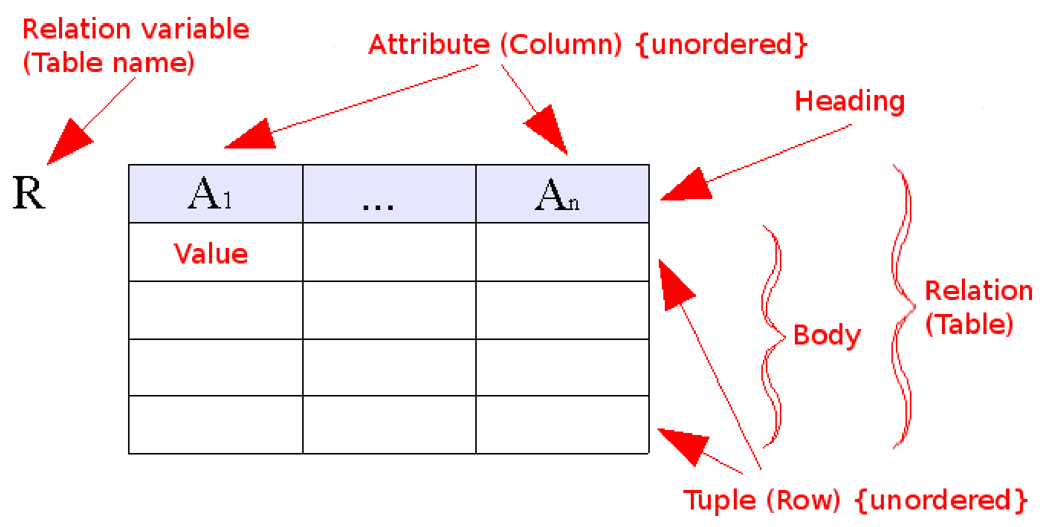




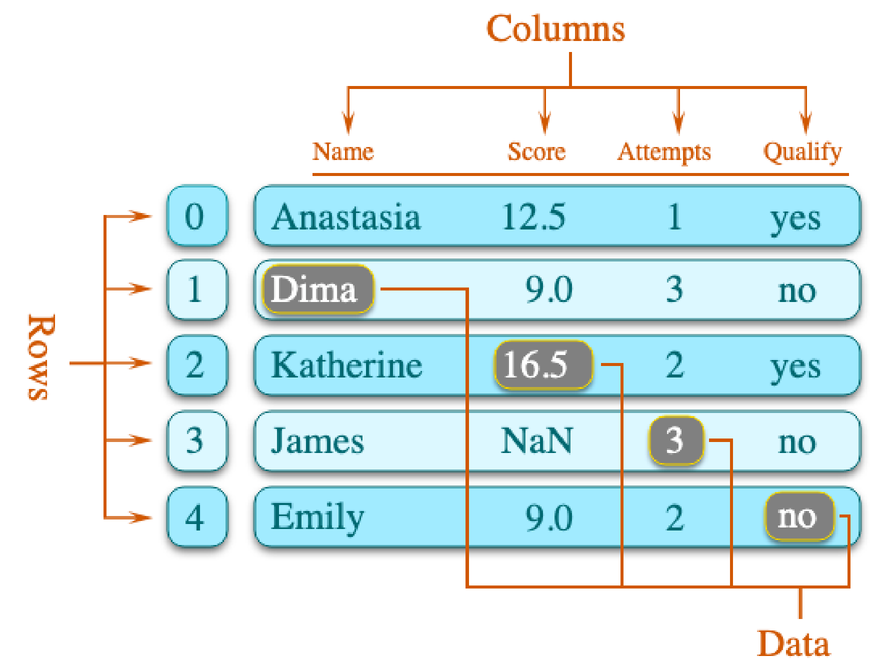
- Can layer on Relations, Matrices, or DataFrames, or be treated as first-class data model
- Inherits flexibility in file formats (text, binary, etc.)

Comparing Struct. Data Models

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



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

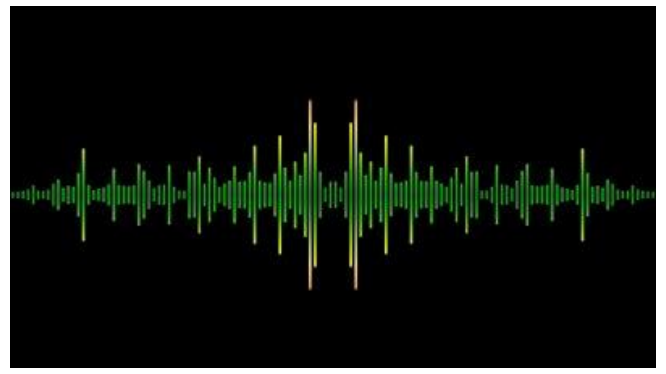


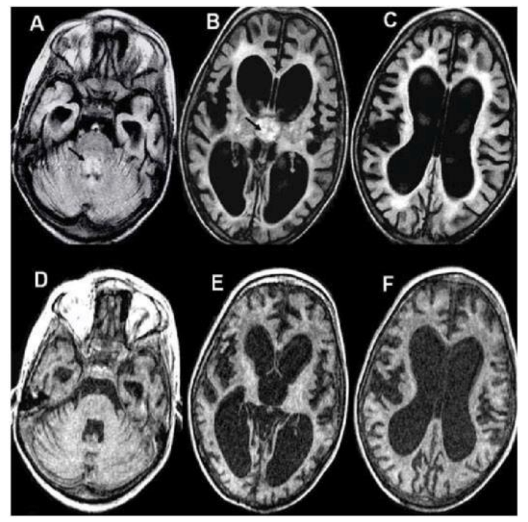
- Ordering: Matrix and DataFrame have row/col numbers; Relation is orderless on both axes!
- Schema Flexibility: Matrix cells are numbers. Relation tuples conform to pre-defined schema. DataFrame has no pre-defined schema but all rows/cols can have names; col cells can be mixed types!
- Transpose: Supported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

If interested in reading more: https://towardsdatascience.com/preventing-the-death-of-the-dataframe-8bca1c

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