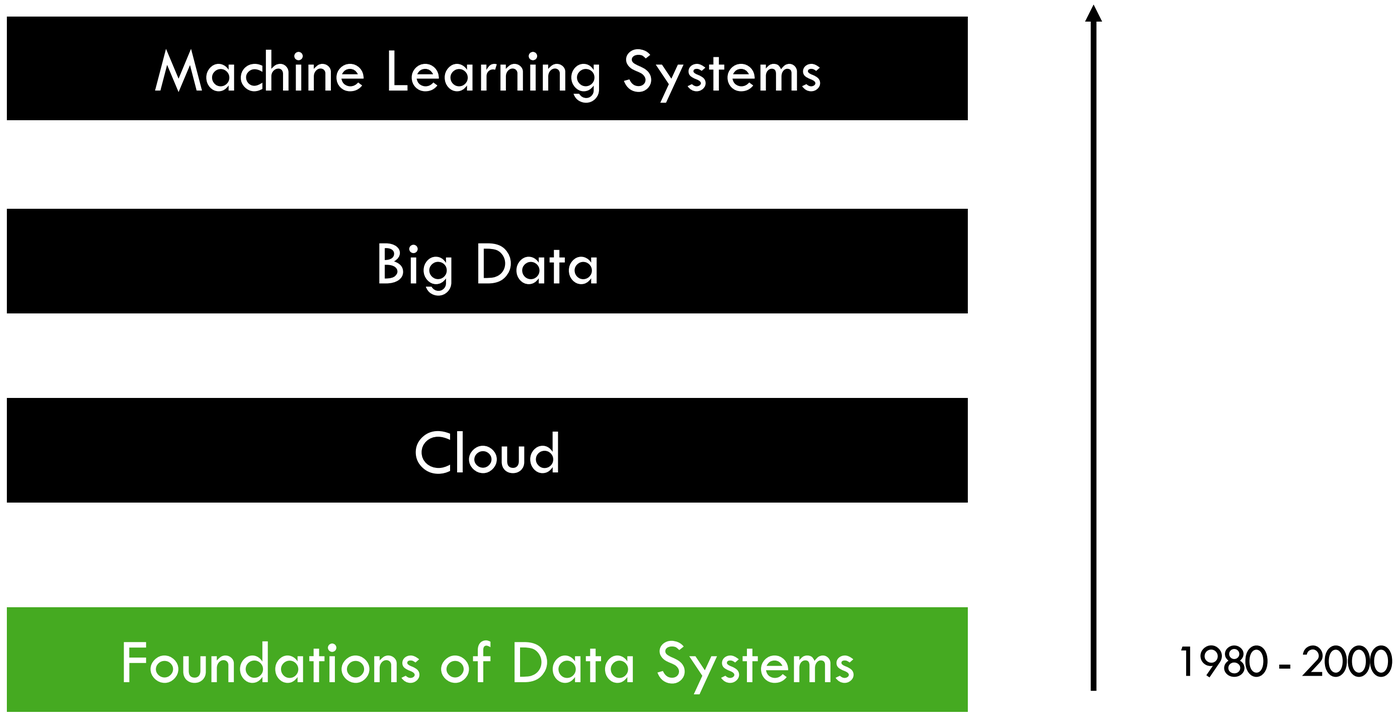
https://hao-ai-lab.github.io/dsc204a-f25/

DSC 204A: Scalable Data Systems
Fall 2025

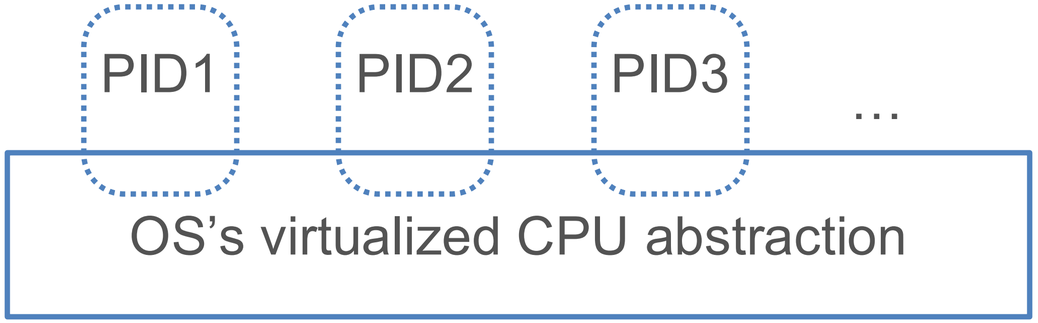
Staff
Instructor: Hao Zhang
TAs: Mingjia Huo, Yuxuan Zhang

@haozhangml @haoailab
haozhang@ucsd.edu

Where We Are


Machine Learning Systems
Big Data
Cloud
Foundations of Data Systems
1980 - 2000

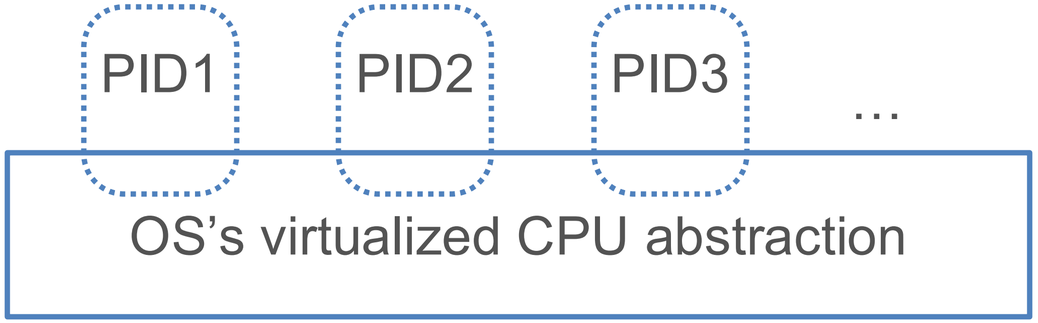
Let's Implement It!



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

Physical Processor

Let's Implement It!

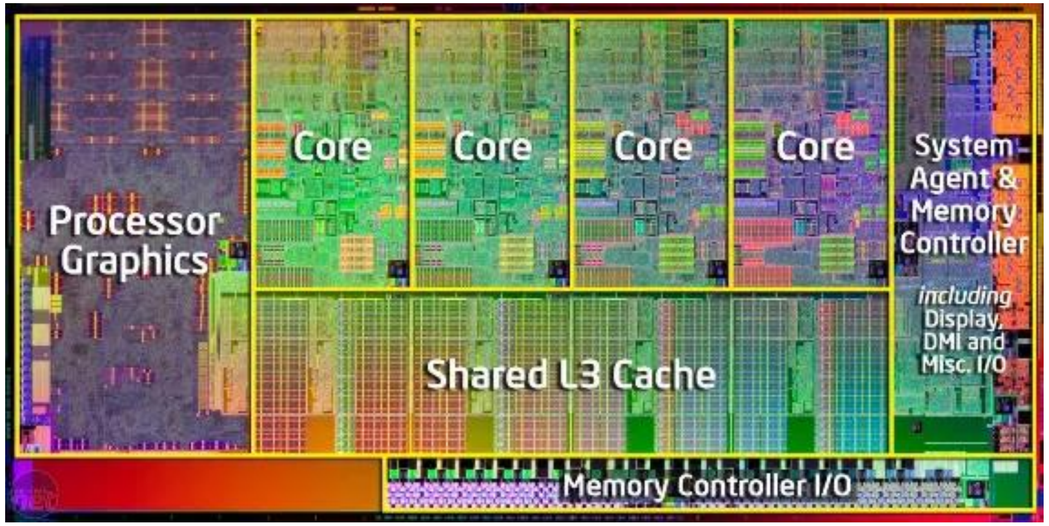


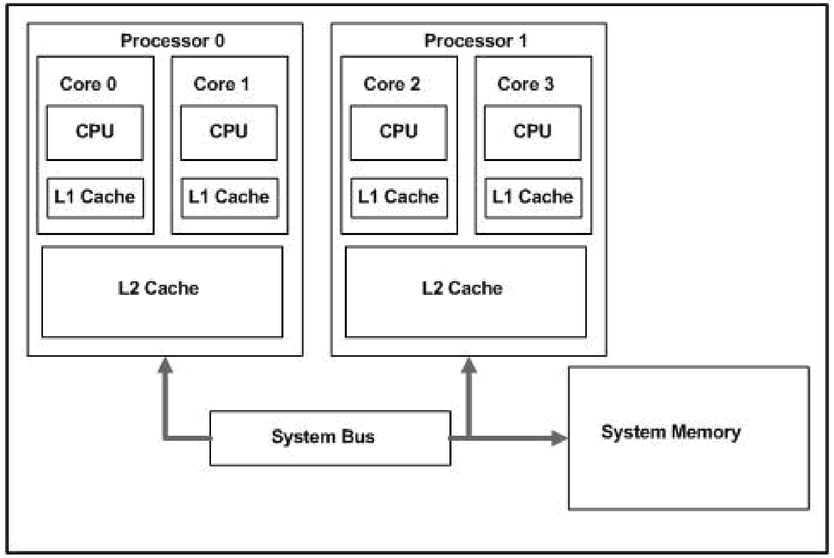
GAP2: How to virtualize CPU resources temporally and **spatially**?

Physical Processor

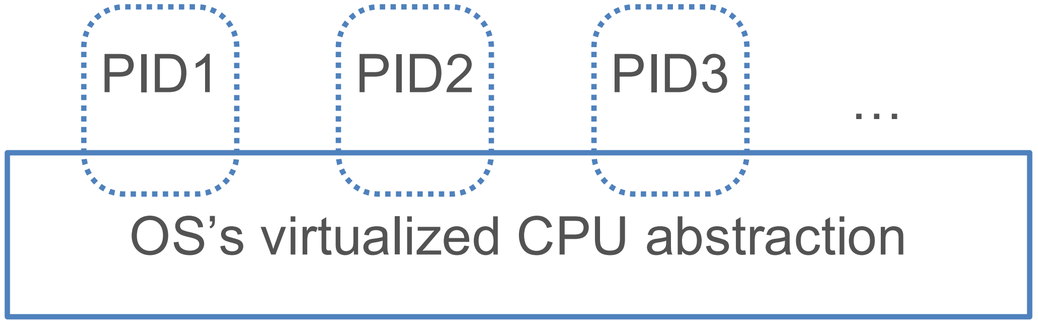
Concurrency

- 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





“Placement” (vs. Scheduling)



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

Physical Processor

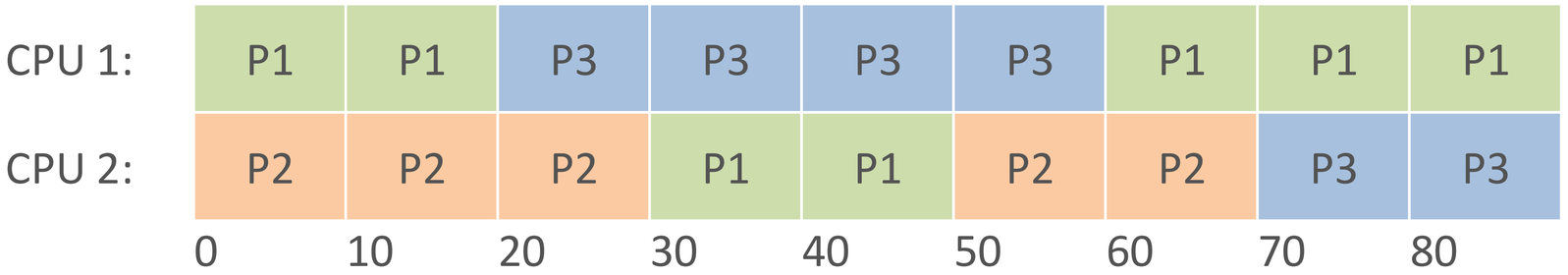
“Placement“ naturally emerges:

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

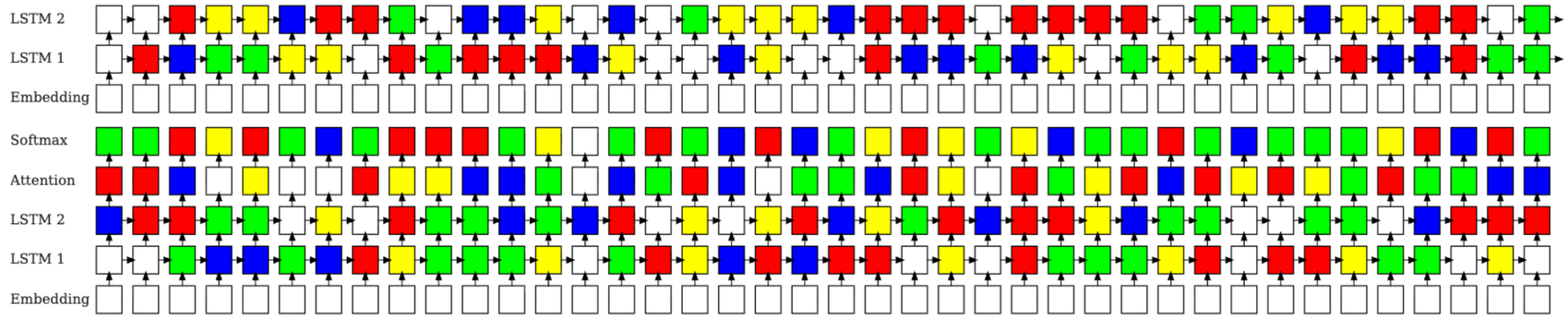
Placment Goal: Loading Balancing

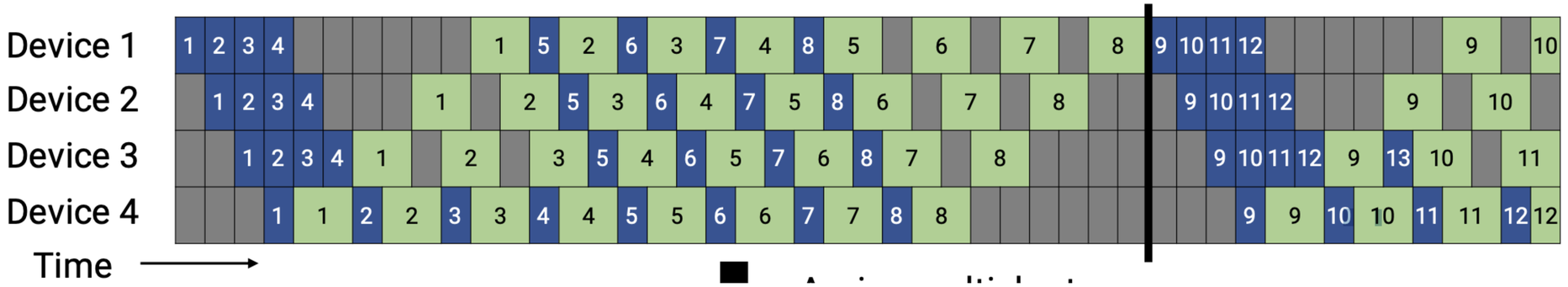
Load Balancing: Ensuring different cores/proc. are kept roughly equally busy, i.e., reduce **idle times**

- 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:



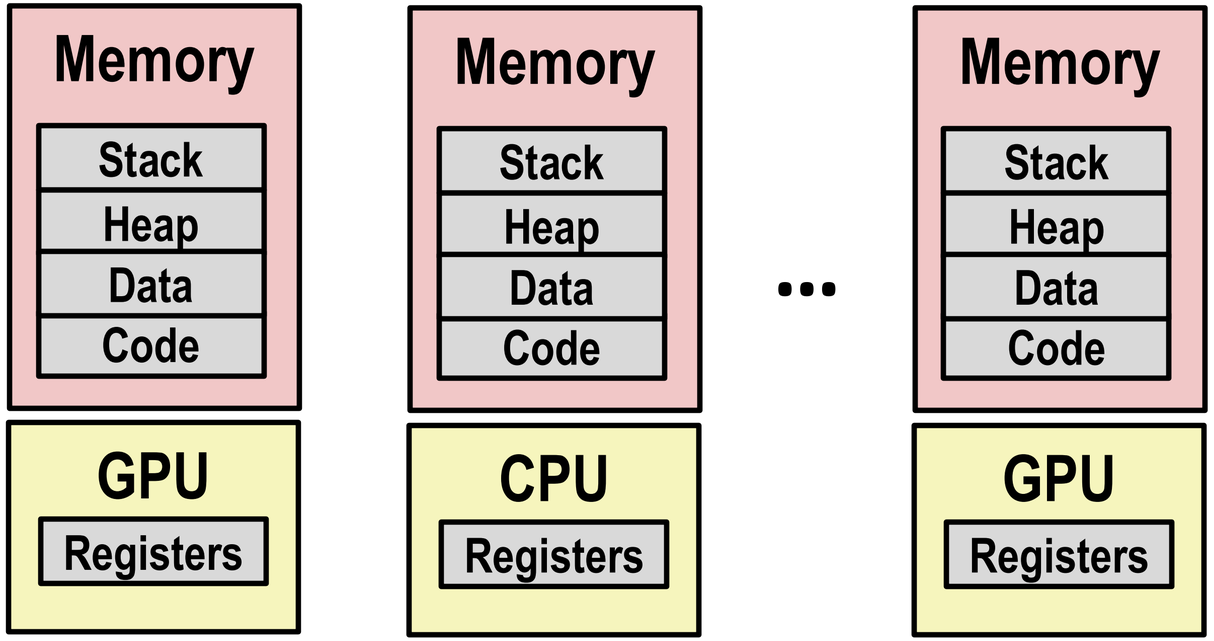
In ML, How Placement Optimizations Look Like





Mutliprocessing Part 2: memory management

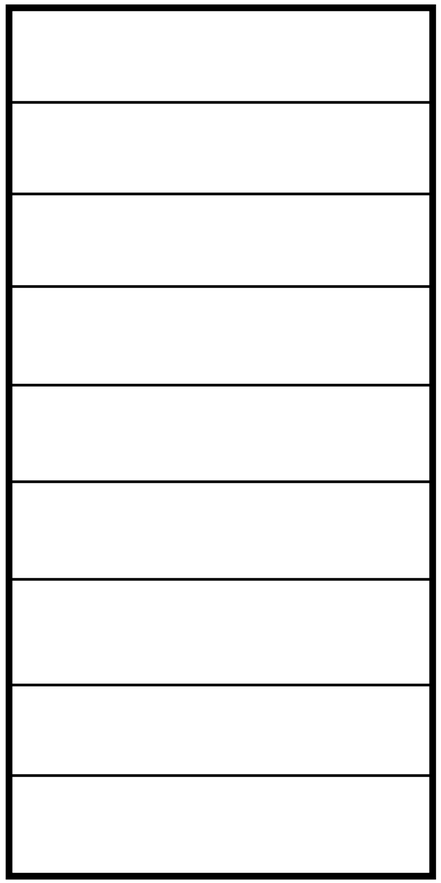
- ~~Strawman solution~~ **-> spatial-temporal sharing of CPUs with scheduling**
- Assign 1/3 of the memory to each APP

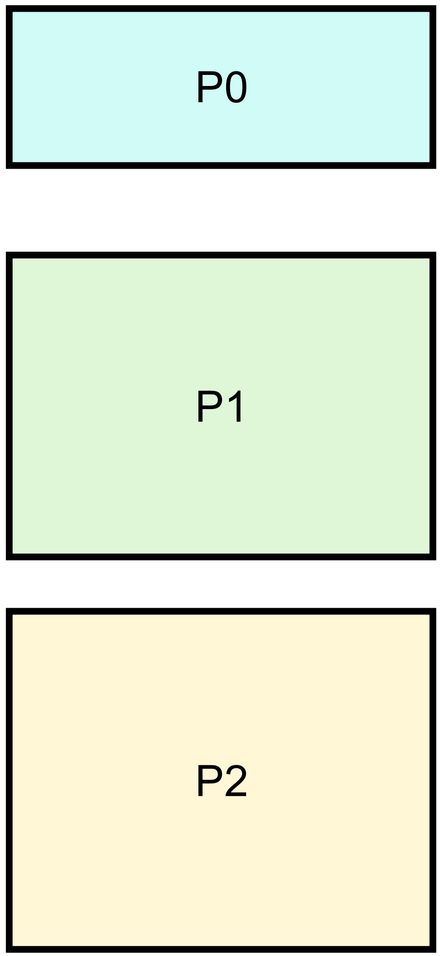


G1. Convenient?
G3: protection?
G2. Efficient?

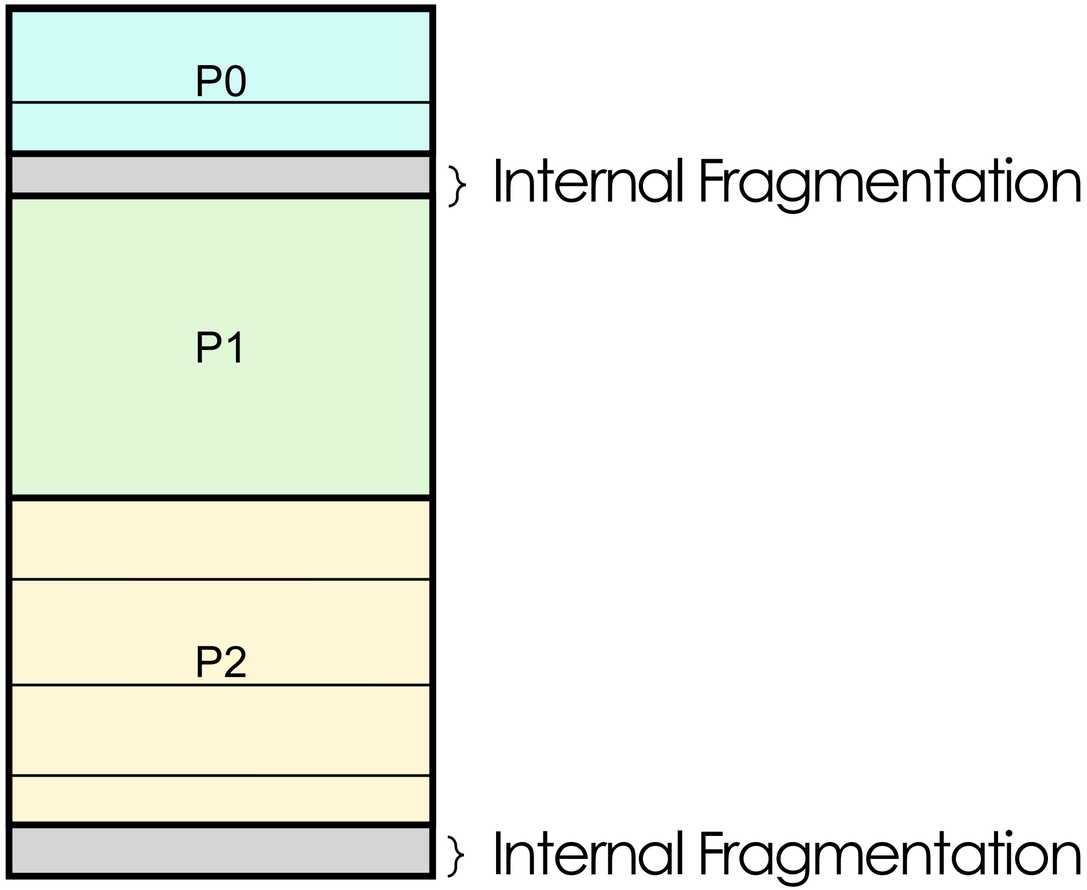
- G2.1 can I run N processes but not N times slower?
- **G2.2 can I run N apps with total mem > physical memory cap**

Memory management v0

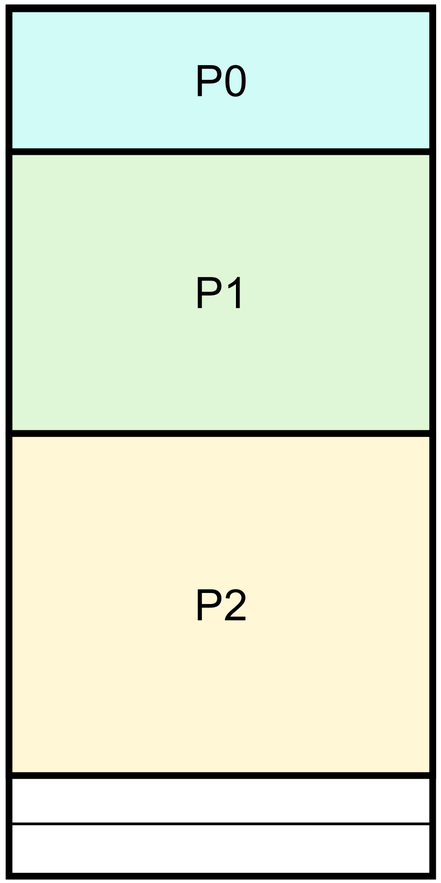




Memory management v0: Internal fragmentations

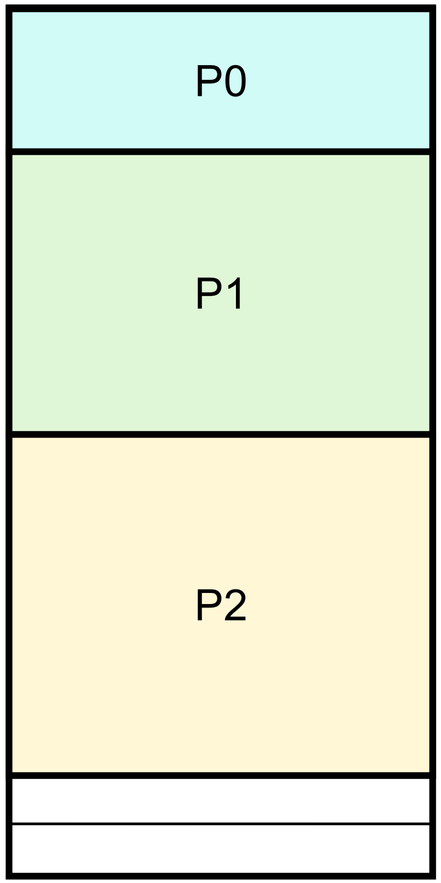


Memory management v1: use a smaller chunk

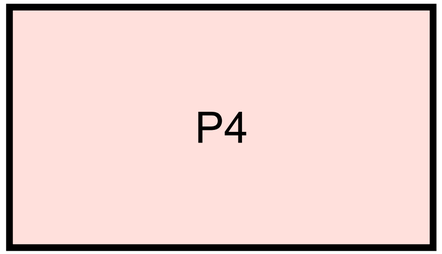


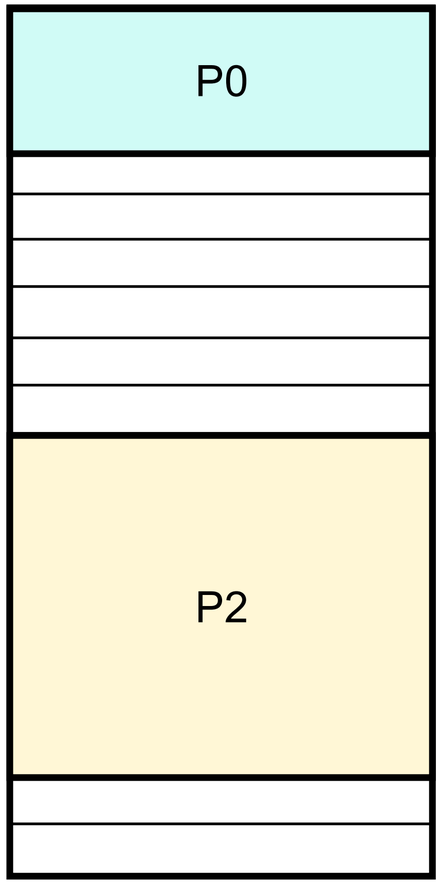
Q: What is the maximum possible amount of internal fragmentation per process?

Memory management v1

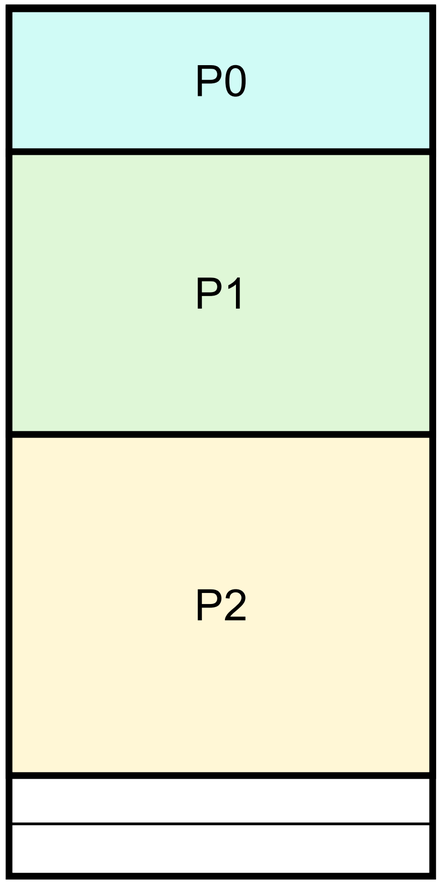


P1 finishes, P4 arrives

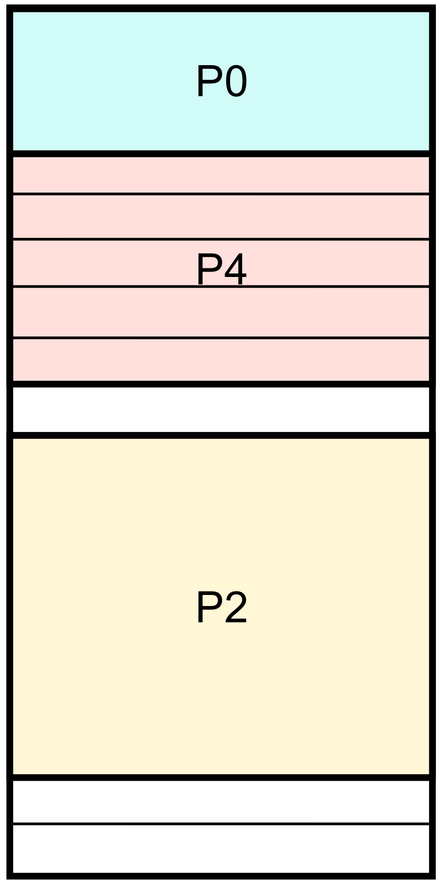




Memory: v2

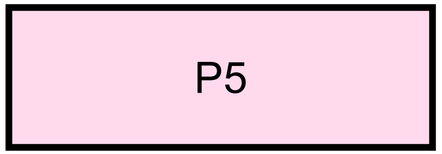


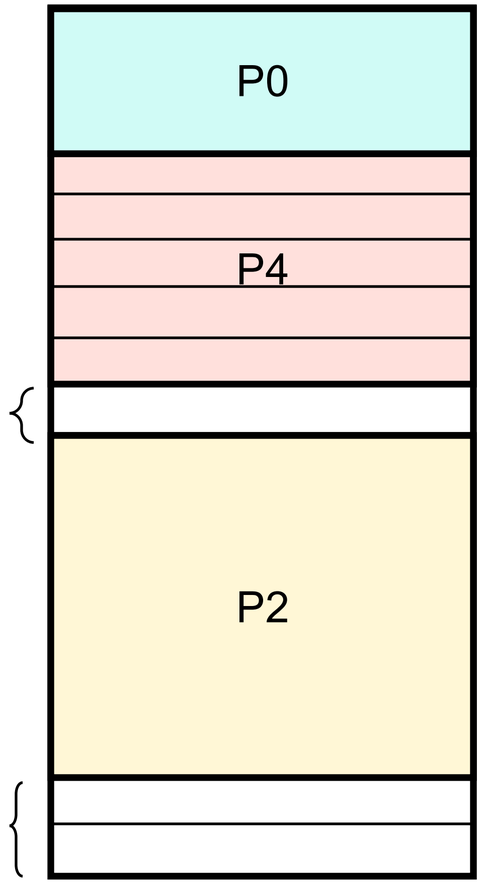
P4 scheduled



Memory: v2

P5 arrived



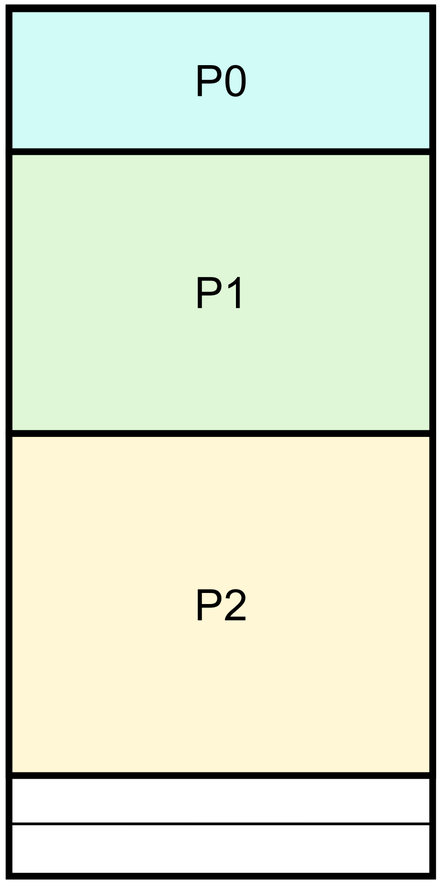


Problem:
There is enough memory for P5, but it cannot be scheduled.

Q: How to address external fragmentation?

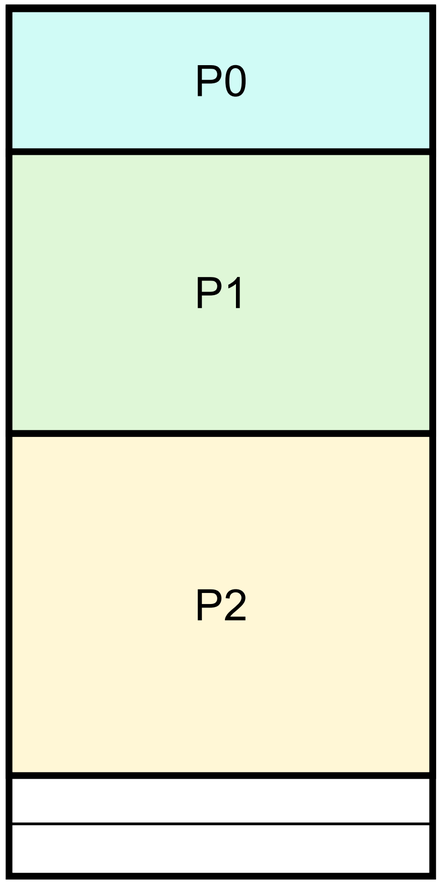
external fragmentation

Other Problems?



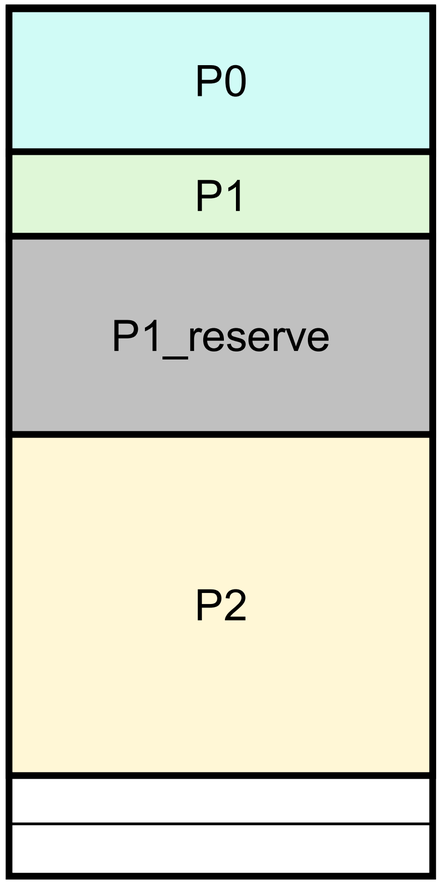
Problem: We can never schedule processes with their memory consumption greater than memory cap

Other Problems?



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

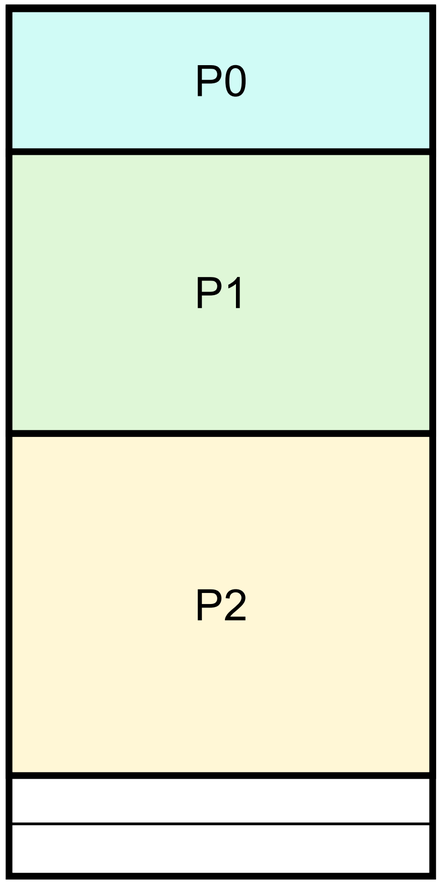
Other Problems?



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

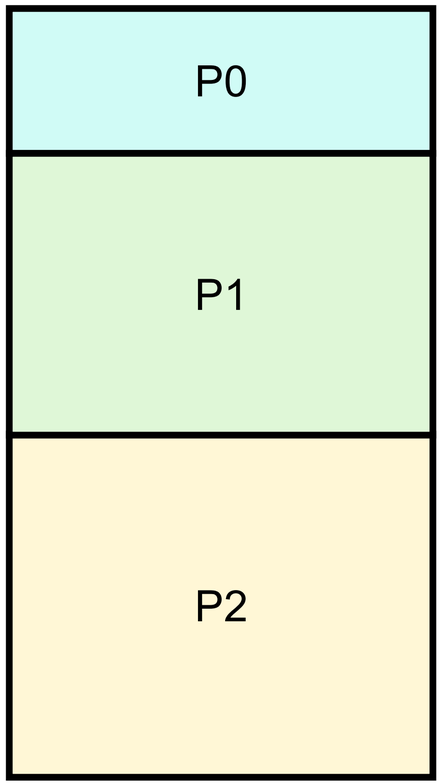
P1_reserve is the reservation overhead

Other Problems?

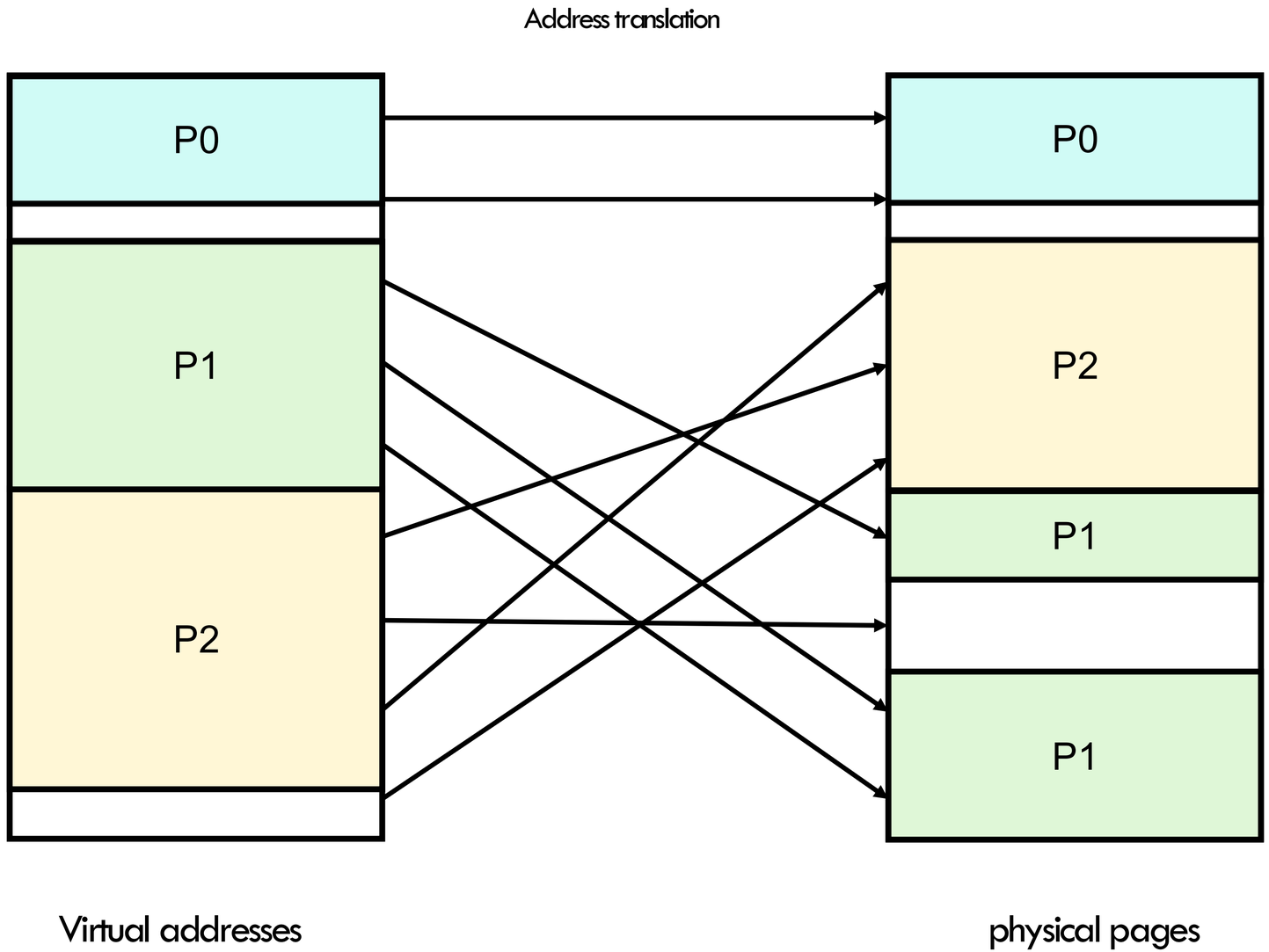


What if we **know exactly** how much memory P0/P1/P2 will **eventually** use, any problem?

Virtual Address Table



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



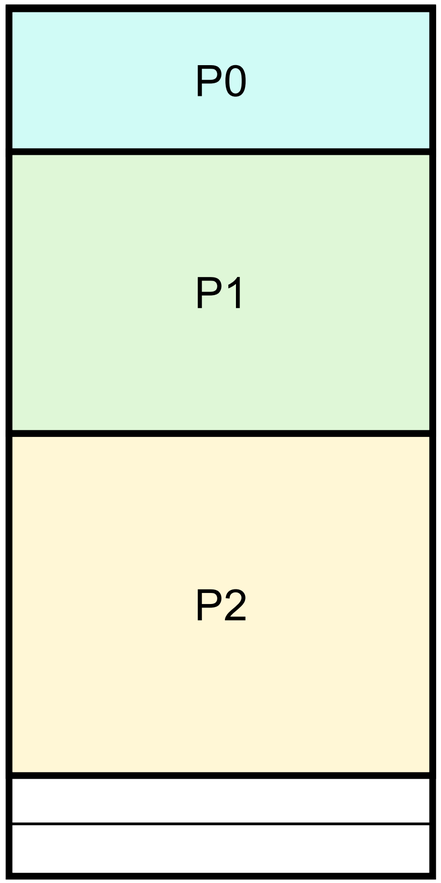
Pages and virtual memory

- **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.

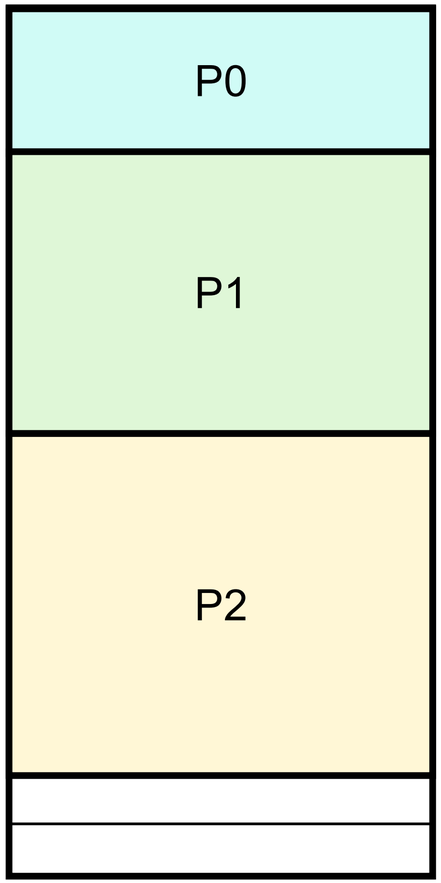
Problem addressed?



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.

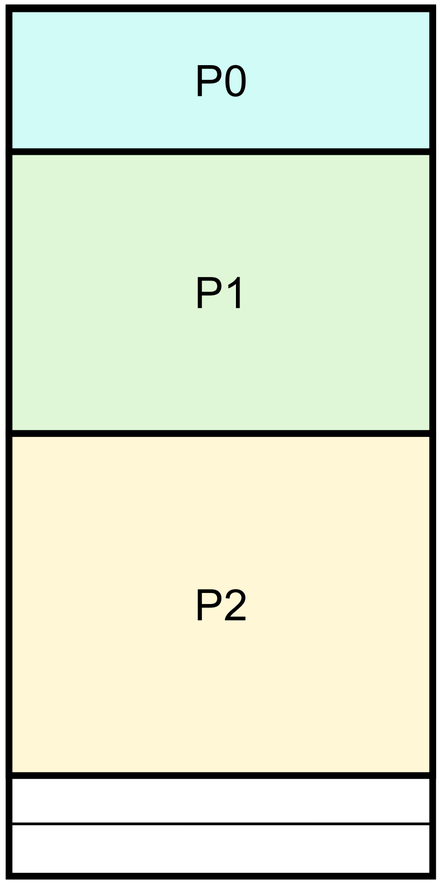
Problem addressed?



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

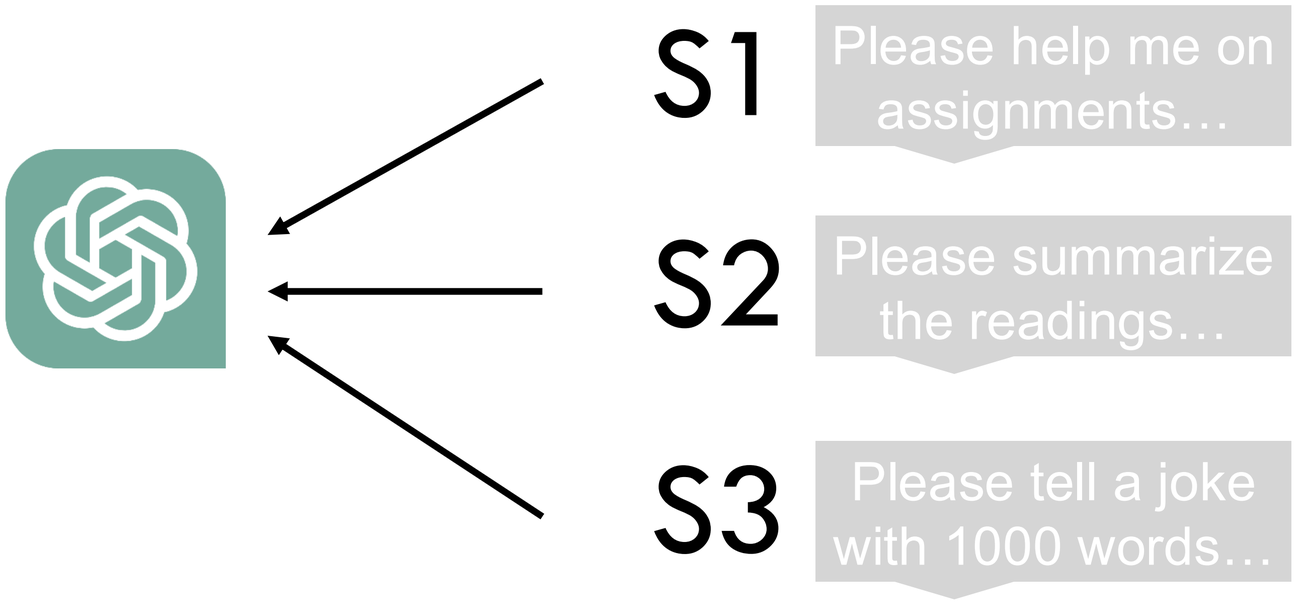
Problem addressed?



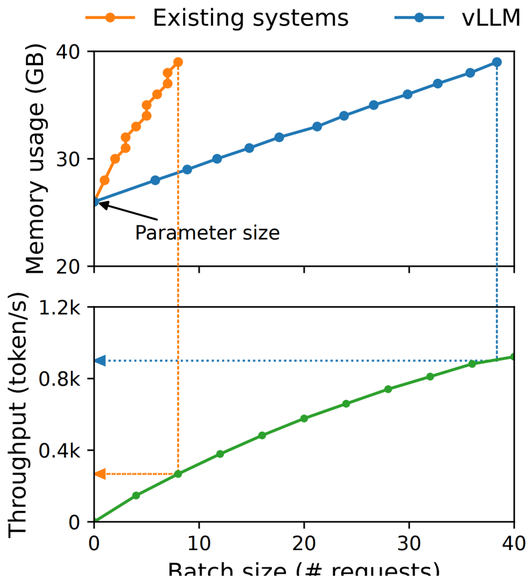
What if we **know exactly** how much memory P0/P1/P2 will **eventually** use, any problem?

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, ...

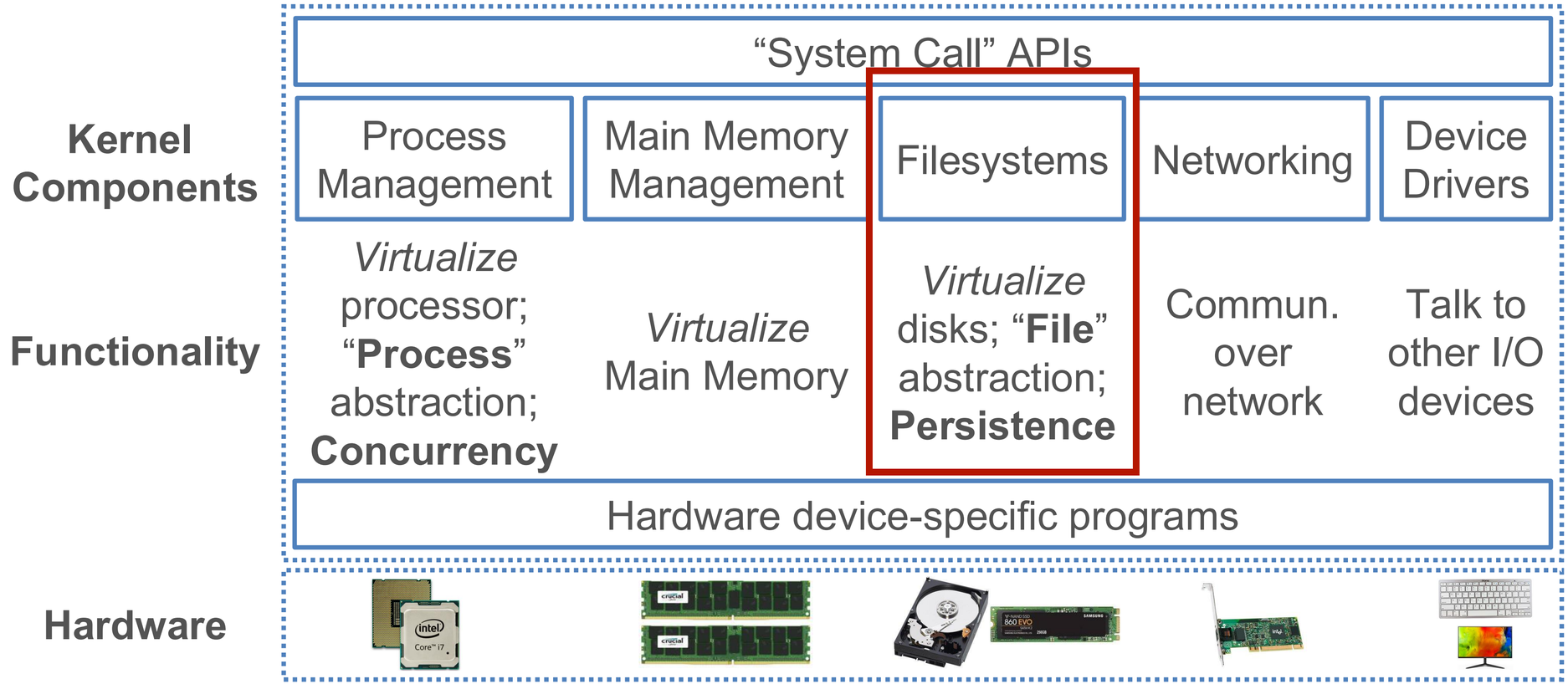
Proceedings of the 29th Symposium on Operating Systems Principles, 611-626

Foundation of Data Systems

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

Modules

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



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

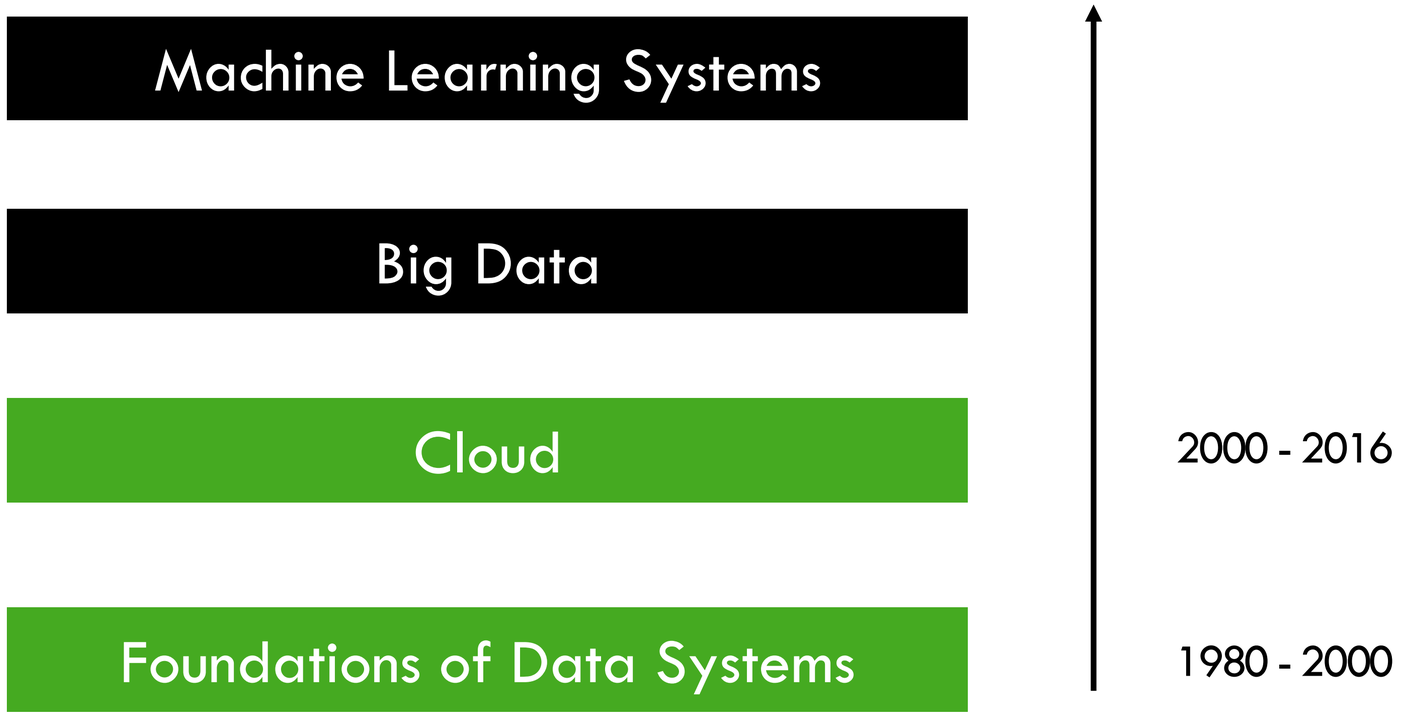
- Dozens of filesystems exist, e.g., ext2, ext3, NTFS, etc.
  - Differ on how they layer file and dir. abstractions as bytes, what metadata is stored, etc.
  - Differ on how data integrity/reliability is assured, support for editing/resizing, compression/encryption, etc.
  - Some can work with ("mounted"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

Where We Are


Machine Learning Systems
Big Data
Cloud
Foundations of Data Systems
2000 - 2016
1980 - 2000

Part 2: Cloud Computing and Distributed Systems

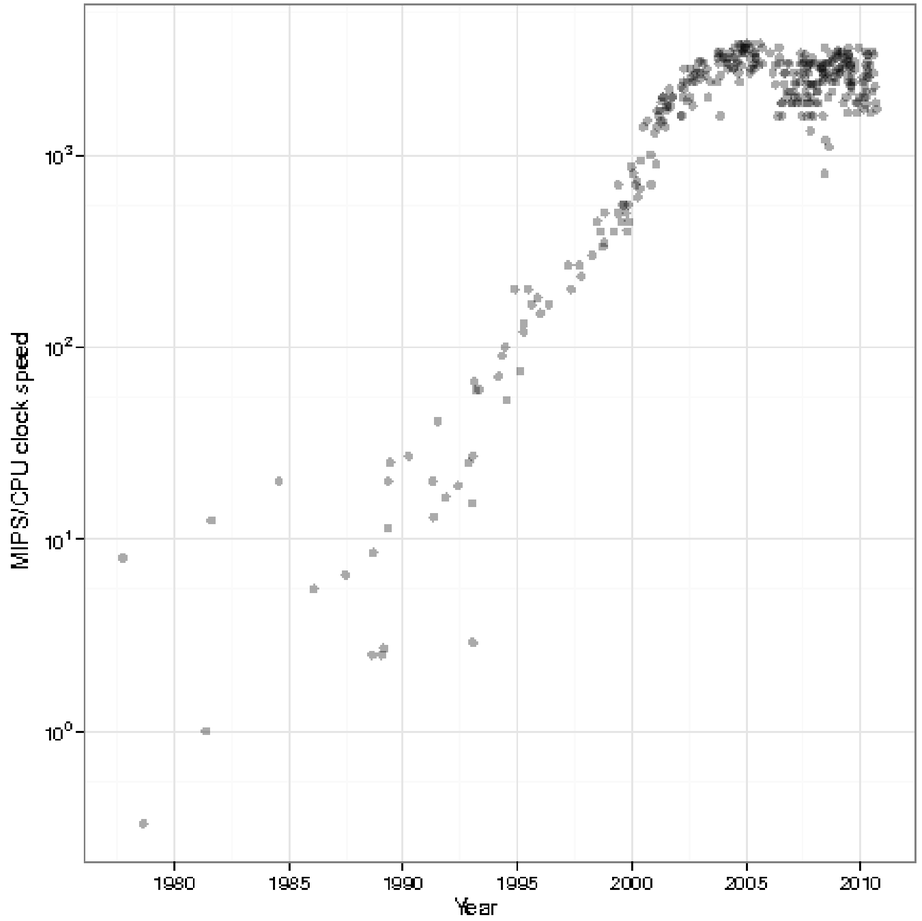
- Intro to Cloud Compute
- Networking
- Distributed Storage and file systems
- Distributed Computing
- Parallelism and consistency
- Advanced Topics

Today's topic

- Why cloud computing?
  - Need-based argument
  - Utility-based argument
- High-level Introduction of Cloud Computing:
  - Cloud computing evolution - sharing granularity
  - Cloud computing layers
  - Advantages of Cloud computing

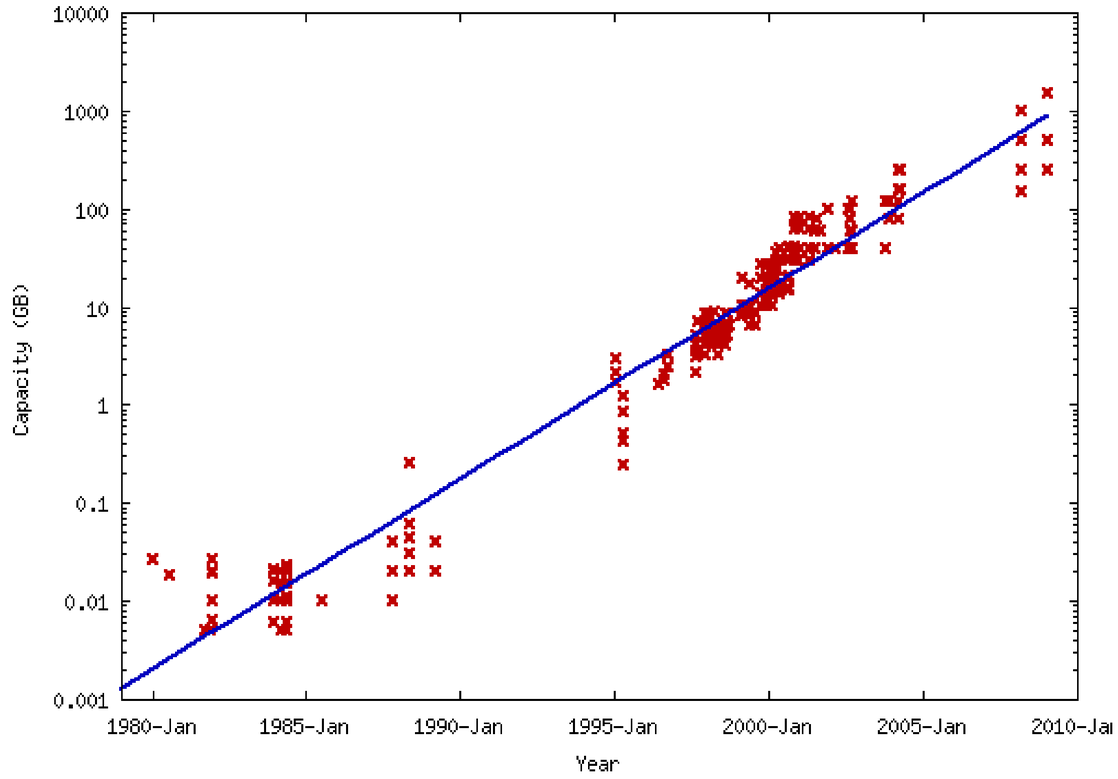
Background of Cloud Computing

- 1990: Heyday of parallel computing, multi-processors
  - 52% growth in performance per year!
- 2002: The thermal wall
  - Speed (frequency) peaks, but transistors keep shrinking
- The Multicore revolution
  - 15-20 years later than predicted, we have hit the performance wall



Data Explosion

- Billions of users connected through the net
  - WWW, FB, twitter, cell phones, …
- Storage getting cheaper
  - Store more data!
- Processing these data
  - Need more FLOPs!



Solving the Impedance Mismatch

- Computers not getting faster, and we are drowning in data
  - How to resolve the dilemma?

- Solution adopted by web-scale companies
  - Go massively *distributed* and *parallel*

Enter the World of Distributed Systems

- Distributed Systems/Computing
  - *Loosely coupled* set of computers, communicating through message passing, solving a common goal

- Distributed computing is *challenging*
  - Dealing with *partial failures* (examples?)
  - Dealing with *asynchrony* (examples?)

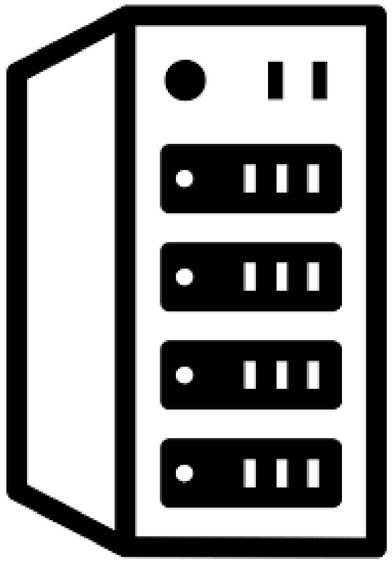
- Distributed Computing versus Parallel Computing?
  - distributed computing=parallel computing + partial failures

Dealing with Distribution: Programming (Part 3)

- We have seen several of the tools that help with distributed programming
  - Message Passing Interface (MPI)
  - Distributed Shared Memory (DSM)
  - Remote Procedure Calls (RPC)

- But, distributed programming is still very hard
  - Programming for scale, fault-tolerance, consistency, …

Recap: Basics of Computer Organization



To store and retrieve data, we need:

- Storages and Disks
- Memory

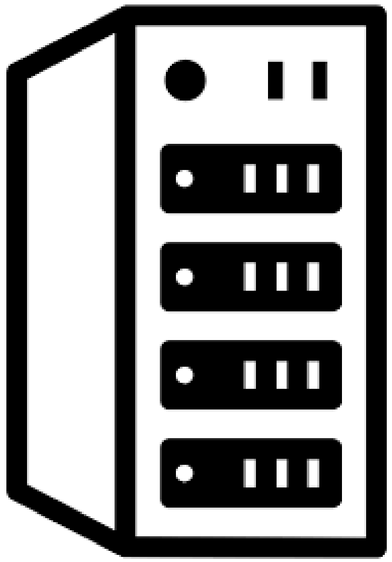
To process data:

- Processors: CPU and GPU

To retrieve data from remote

- Networks

Everything Goes Distributed



To store and retrieve data, we need:

- Distributed storage and disks
- Distributed and shared Memory

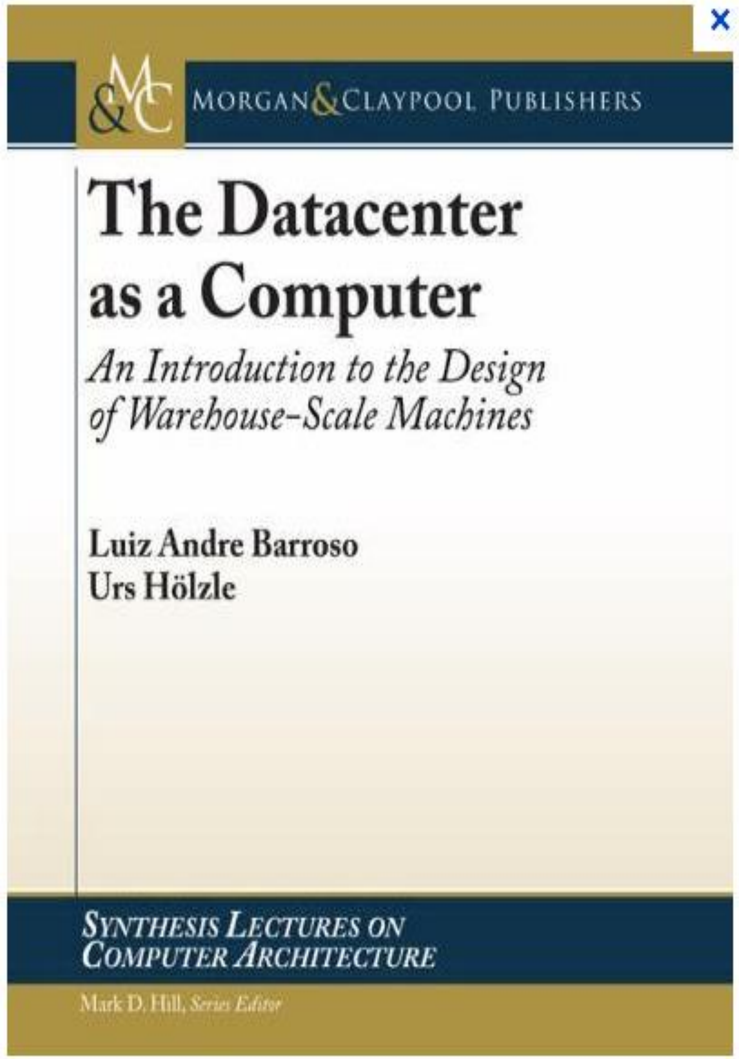
To process data:

- Distributed CPU and GPU

To retrieve data from remote

- Networks

The Datacenter is the new Computer



- *"Program"* == Web search, email, map/GIS, …
- *"Computer"* == 10,000's computers, storage, network
- Warehouse-sized facilities and workloads
- *Built from less reliable components than traditional datacenters*

Datacenter/Cloud Computing OS

- If the datacenter/cloud is the new computer
  - What is its **Operating System**?

Classical Operating Systems

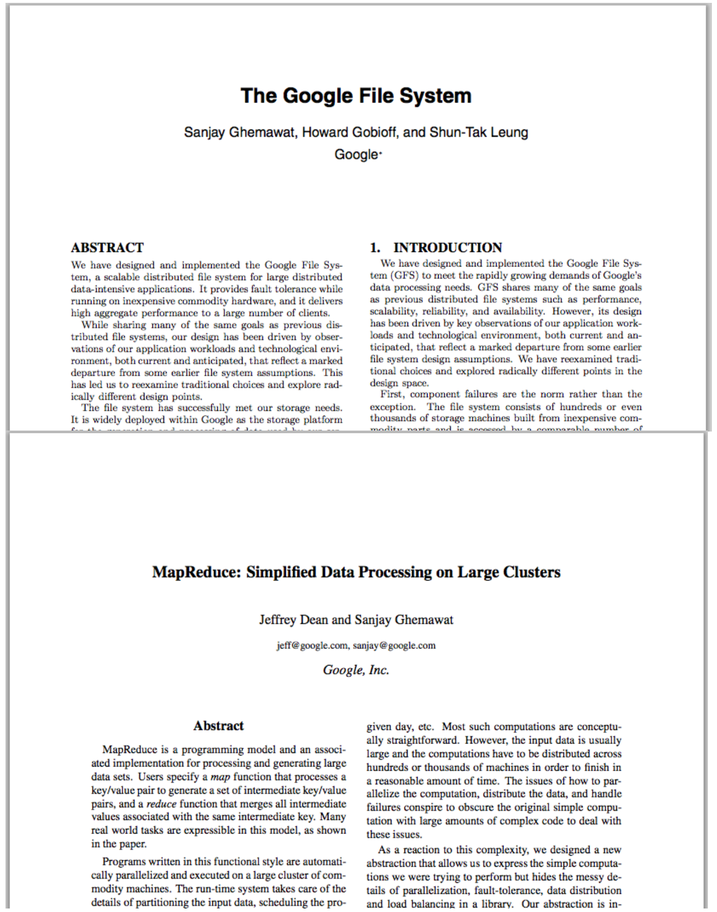
- Data storage and sharing
  - files, Inter-Process Communication, …
- Programming Abstractions
  - system calls, APIs, libraries, …
- Multiplexing of resources
  - Scheduling, virtual memory, file systems, …

Datacenter/Cloud Operating System

- Data sharing
  - key/value stores, distributed storage, data warehouse
- Programming Abstractions
  - MapReduce, PIG, Hive, Spark, Ray
- Multiplexing of resources
  - YARN (MRv2), ZooKeeper, BookKeeper, K8S, …

Pioneer: Google Cloud Infrastructure

- Google File System (GFS), 2003
  - Distributed File System for entire cluster
- Google MapReduce (MR), 2004
  - Runs queries/jobs on data
  - Manages work distribution & fault-tolerance
  - Colocated with file system
- Apache open source versions Hadoop DFS and Hadoop MR



The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive com-

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the pro-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is in-

Open Question after class

Google has pioneered and created many distributed systems and technologies that shape today's cloud computing, but why Amazon (and even Microsoft) wins over Google Cloud (GCP) on Cloud computing market shares?

Summary: need-based argument

Need more compute and storage

Single computer hits physical limits

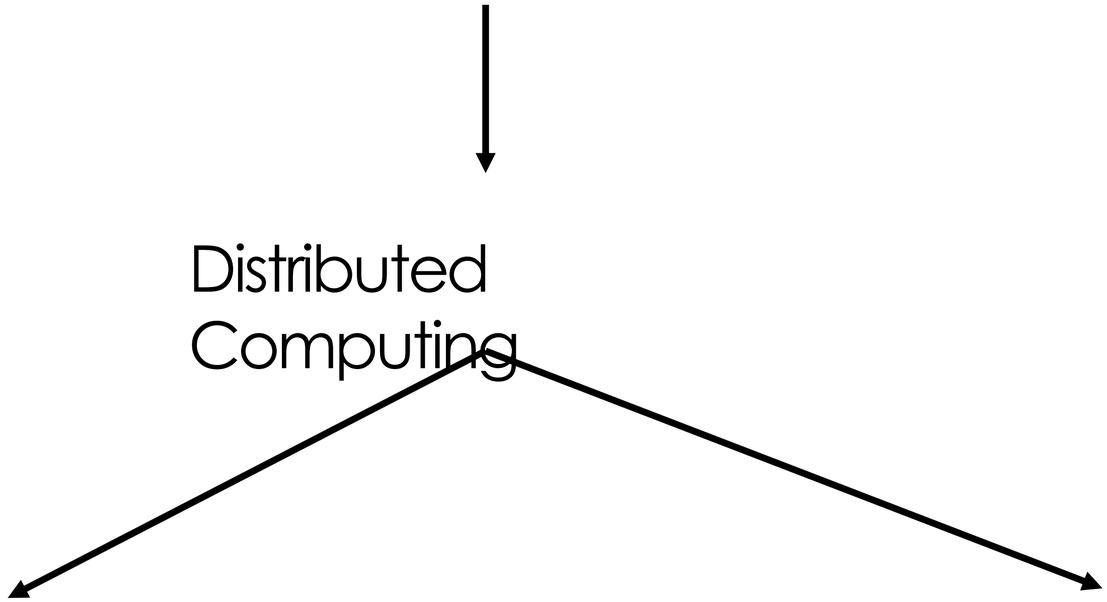
Distributed Computing

Cloud has a lot of compute and storage

Summary: need-based argument

Need more compute and storage

Single computer hits physical limits



Distributed Computing

Cloud has a lot of compute and storage

On-premise or supercomputers also have a lot of compute and storage

Today's topic

- Why cloud computing?
  - Need-based argument
  - **Utility-based argument**
- High-level Introduction of Cloud Computing:
  - Cloud computing evolution - sharing granularity
  - Cloud computing layers
  - Advantages of Cloud computing

Consider a Use Case

- A company needs more compute and storage

Traditional Model

- We manage and store computes **on premise**
- Responsible for security
- Responsible for power
- Responsible for network
- Responsible for …

Consider a Use Case

- A company needs more compute and storage

Traditional Model
If we need more computers (a.k.a. we want to scale)

- We order computers
- They are delivered to our site
- We install them and connect them to the cluster via network.

Consider a Use Case

- A company needs more compute and s

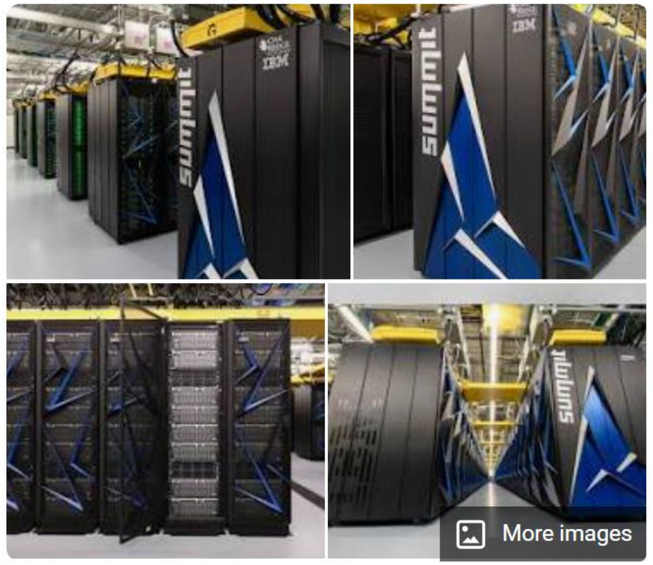
Traditional Model

If updates or security patches are issued:

- We make sure this is taken care of for each computer in the cluster.

Summit

Supercomputer



Summit or OLCF-4 is a supercomputer developed by IBM for use at Oak Ridge Leadership Computing Facility, a facility at the Oak Ridge National Laboratory, capable of 200 petaFLOPS thus making it the 5th fastest supercomputer in the world after Frontier, Fugaku, LUMI, and Leonardo, with Frontier being the fastest. Wikipedia

Speed: 200 petaFLOPS (peak)

Architecture: 9,216 POWER9 22-core CPUs; 27,648 Nvidia Tesla V100 GPUs

Operating system: Red Hat Enterprise Linux (RHEL)

Power: 13 MW

Purpose: Scientific research

Ranking: TOP500: 5

Storage: 250 PB

Cloud Computing Early Concept: Utility computing

- Utility computing
  - From concept of a public utility such as water or electricity
- Consider: everyday electricity usage
  - It is summer, we turn on A/C
    - We do not notify electric company when we need more electricity. It is just there.
    - We do not go to hardware store buy/install more generators
  - It is Spring, we turn off A/C
    - We do not notify electric company when we need less
  - It is Winter, we turn on heater
    - My usage goes up and down, but I just use

Early Concept: Utility computing

- Utility computing
  - Compute power is available on demand
  - I can scale up or down as needed
  - I don’t need to determine needs in advance
    - Not the case any more for GPU market

Consider a Use Case

- A company needs distributed compute and storage

Traditional Model

- Determine needs in advance
- Overestimate -> unused compute
- Underestimate -> shortage and wanting

Utility computing

- Don’t worry about accurately estimating needs
- Pay what it is used
- Scale up and down

Consider a Use Case

- A company needs distributed compute and storage

Traditional Model

- The company provides on-site security
- We provide backup power for emergencies

Utility computing

- Cloud infra company provides security
- Cloud infra company provide emergency or fault tolerance

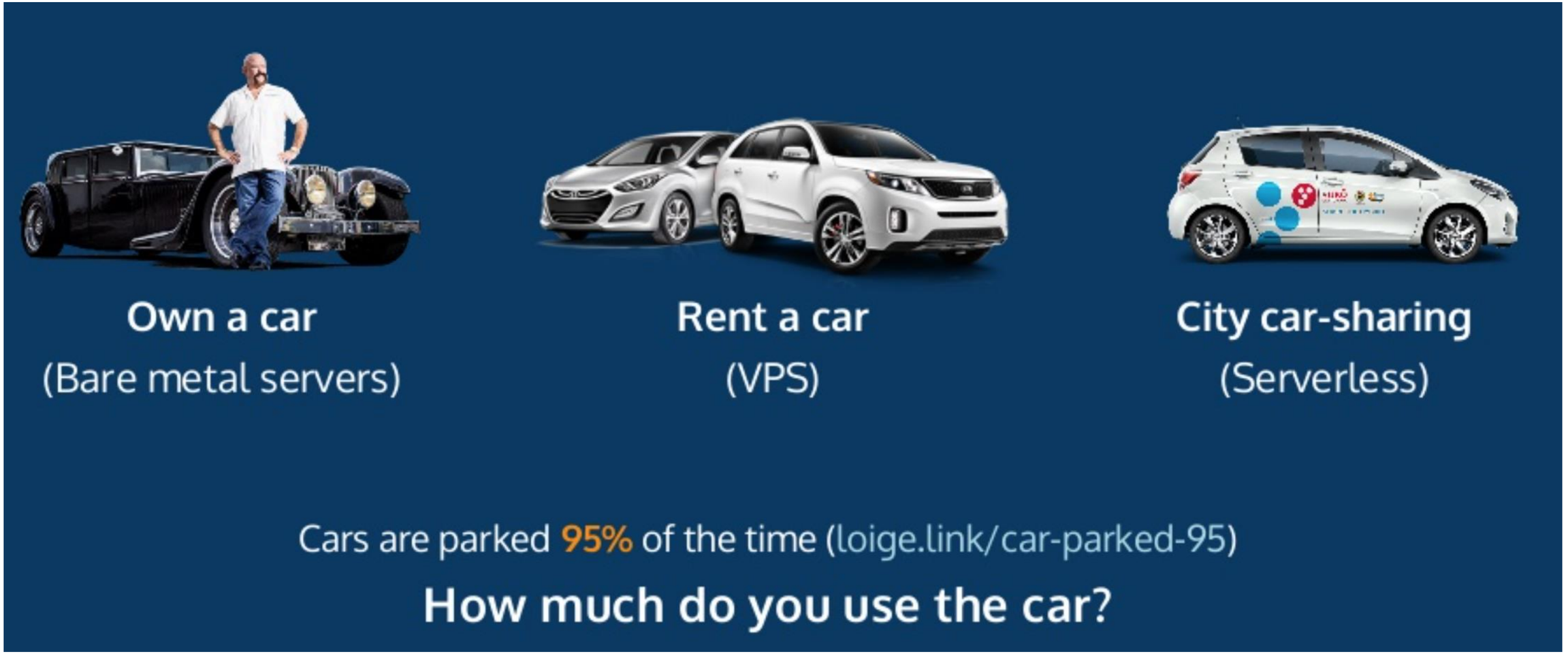
Cloud Computing

- Compute, storage, memory, networking, etc. are virtualized and exist on remote *servers*; rented by application users
- The opposite:
  - On-premises refers to IT infrastructure hardware and software applications that are hosted on-site.

Evolution of Cloud Infrastructure

- Data Center: Physical space from which a cloud is operated
- 3 generations of data centers/clouds:
  - Cloud 1.0 (Past)
  - Cloud 2.0 (Current)
  - Cloud 3.0 (Ongoing Research)

Car Analogy



https://www.slideshare.net/loige/building-a-serverless-company-with-nodejs-react-and-the-serverless-framework-jsday-2017-verona