Concurrency: Honors Welcome to cs378h

Chris Rossbach

Outline for Today

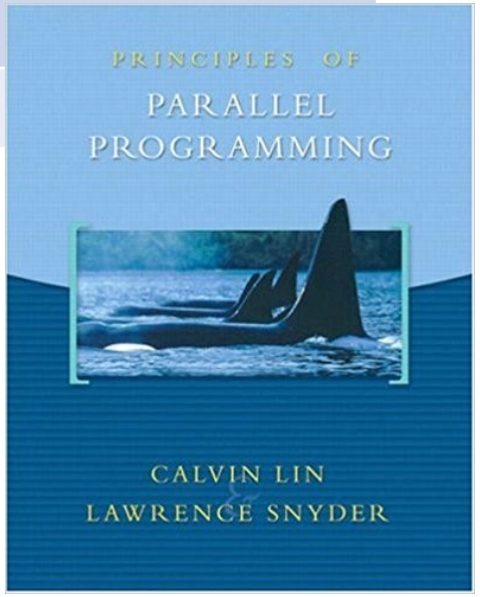
- Questions?
- Administrivia
- Course Overview
- Course Details and Logistics
- Concurrency & Parallelism Basics

Acknowledgments: some materials in this lecture borrowed from:

- *Emmett Witchel, who borrowed them from: Kathryn McKinley, Ron Rockhold, Tom Anderson, John Carter, Mike Dahlin, Jim Kurose, Hank Levy, Harrick Vin, Thomas Narten, and Emery Berger*
- *Mark Silberstein, who borrowed them from: Blaise Barney, Kunle Olukoton, Gupta*

Course Details

Course Name:	CS378H – Concurrency: Honors
Unique Number:	52670
Lectures:	T-Th 9:30-11:00AM Zoom
Class Web Page:	http://www.cs.utexas.edu/users/rossbach/cs378h
Instructor:	Chris Rossbach
TA:	Karthik Velayutham
Text:	Principles of Parallel Programming (ISBN-10: 0321487907)



Please read the syllabus!

…More on this shortly…

Why you should take this course

Why you should take this course

- Concurrency is super-cool, and super-important

Why you should take this course

- Concurrency is super-cool, and super-important
- You'll learn important concepts and background

Why you should take this course

- Concurrency is super-cool, and super-important
- You'll learn important concepts and background
- Have *fun* programming cool systems
  - GPUs! (optionally) FGPAs!
  - Modern Programming languages: Go! Rust!
  - Interesting synchronization primitives (not just boring old locks)
  - Programming tools people use to program *super-computers (ooh…)*

Why you should take this course

- Concurrency is super-cool, and super-important
- You'll learn important concepts and background
- Have *fun* programming cool systems
  - GPUs! (optionally) FGPAs!
  - Modern Programming languages: Go! Rust!
  - Interesting synchronization primitives (not just boring old locks)
  - Programming tools people use to program *super-computers (ooh…)*

Two perspectives:
- The "just eat your kale and quinoa" argument
- The "it's going to be fun" argument

My first computer

My first computer

My first computer

CPU

My first computer

CPU

Storage

My first computer

CPU

Storage

Tape drive!

(also good for playing heavy metal music)

My first computer

CPU

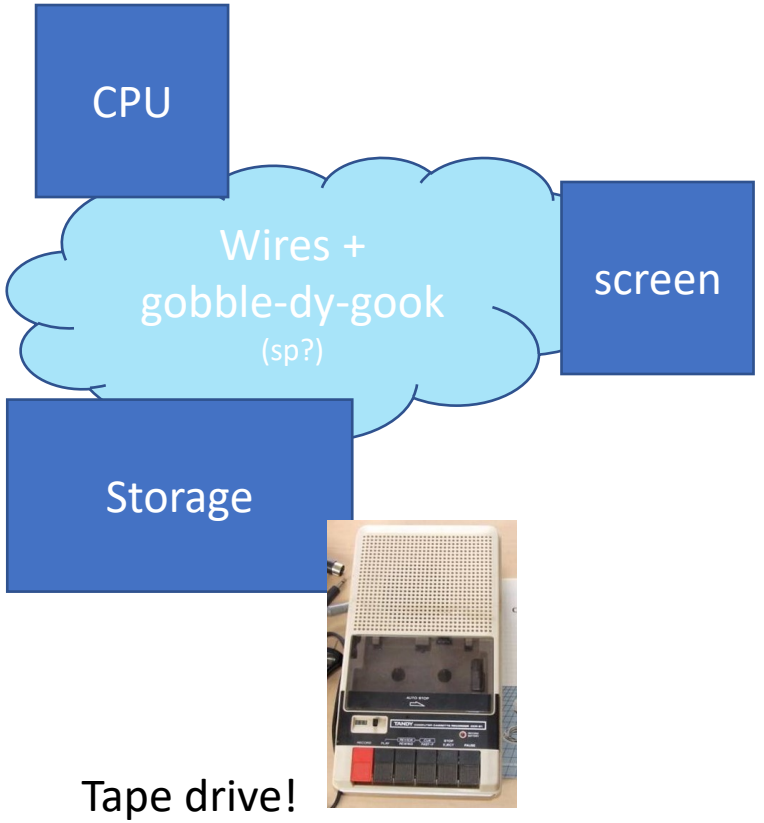
screen

Storage

Tape drive!

(also good for playing heavy metal music)

My first computer



Tape drive!

(also good for playing heavy metal music)

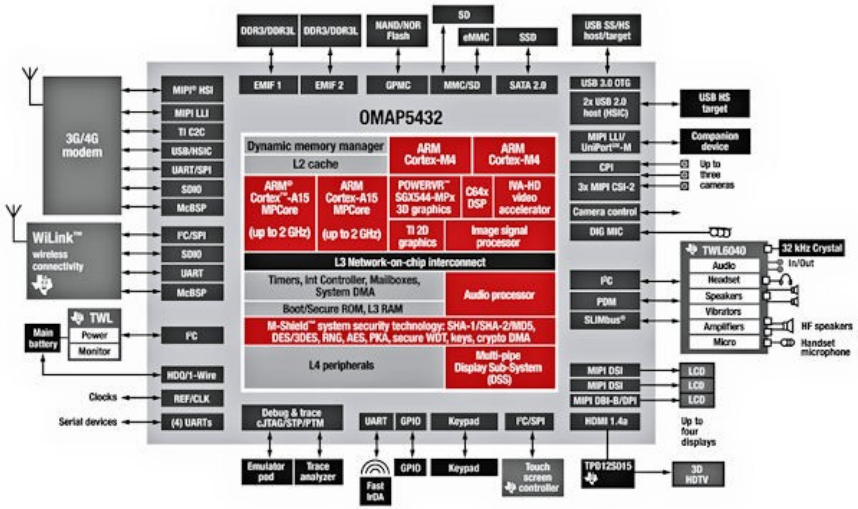
My current computer

My current computer

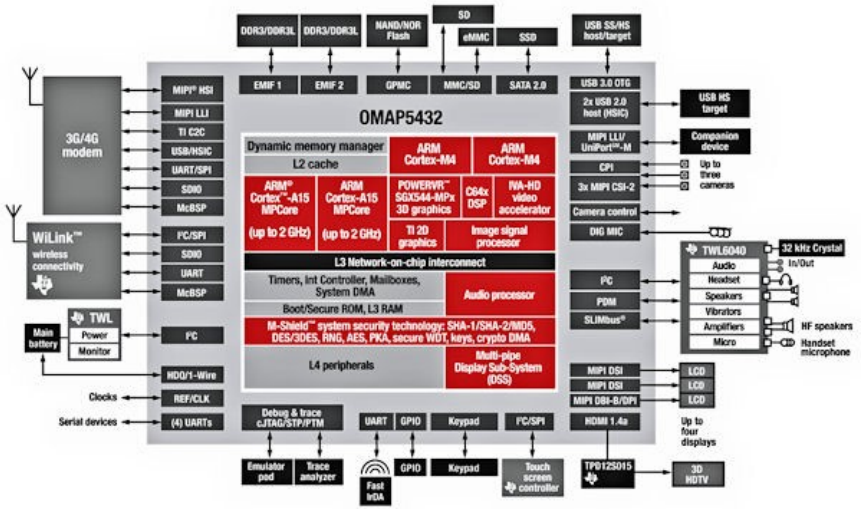
Too boring…

Another of my current computers

Another of my current computers

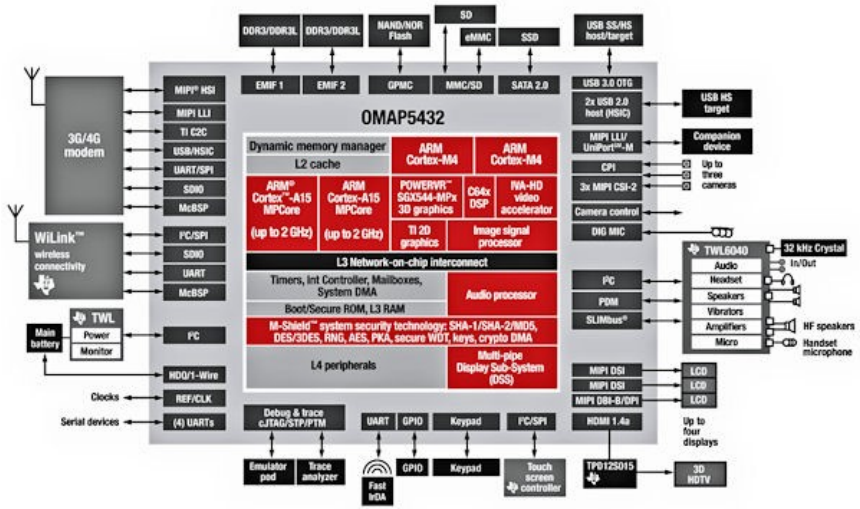


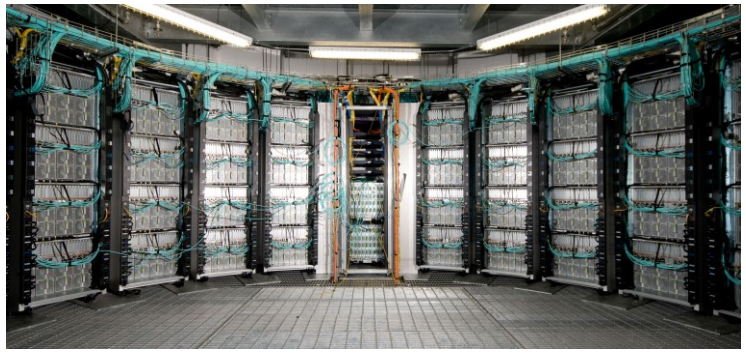
Another of my current computers



- CPU
- CPU
- GPU
- Image DSP
- Crypto
- …

Another of my current computers

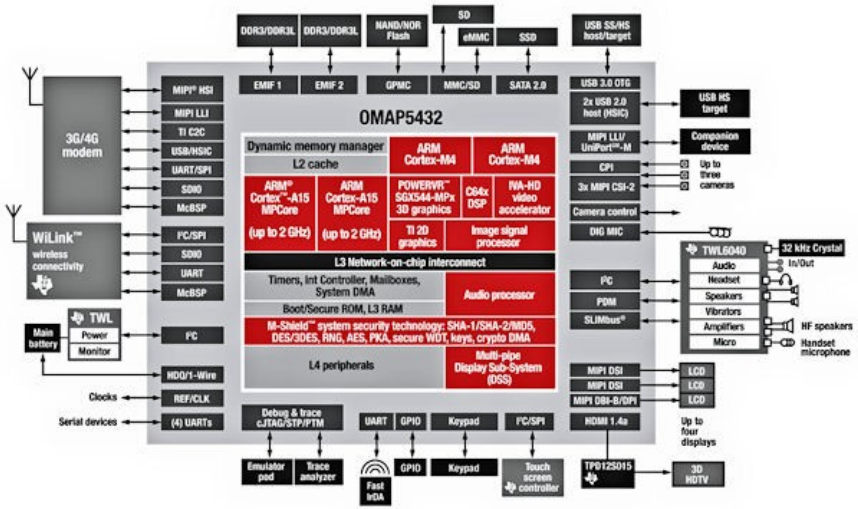


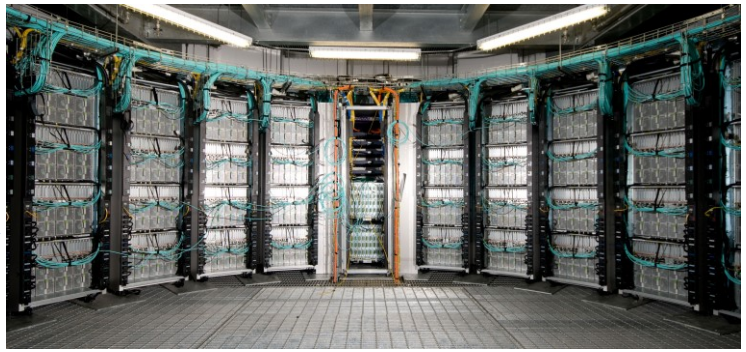


- CPU
- CPU
- GPU
- Image DSP
- Crypto
- …

Another of my current computers

A lot has changed but…
the common theme is…??





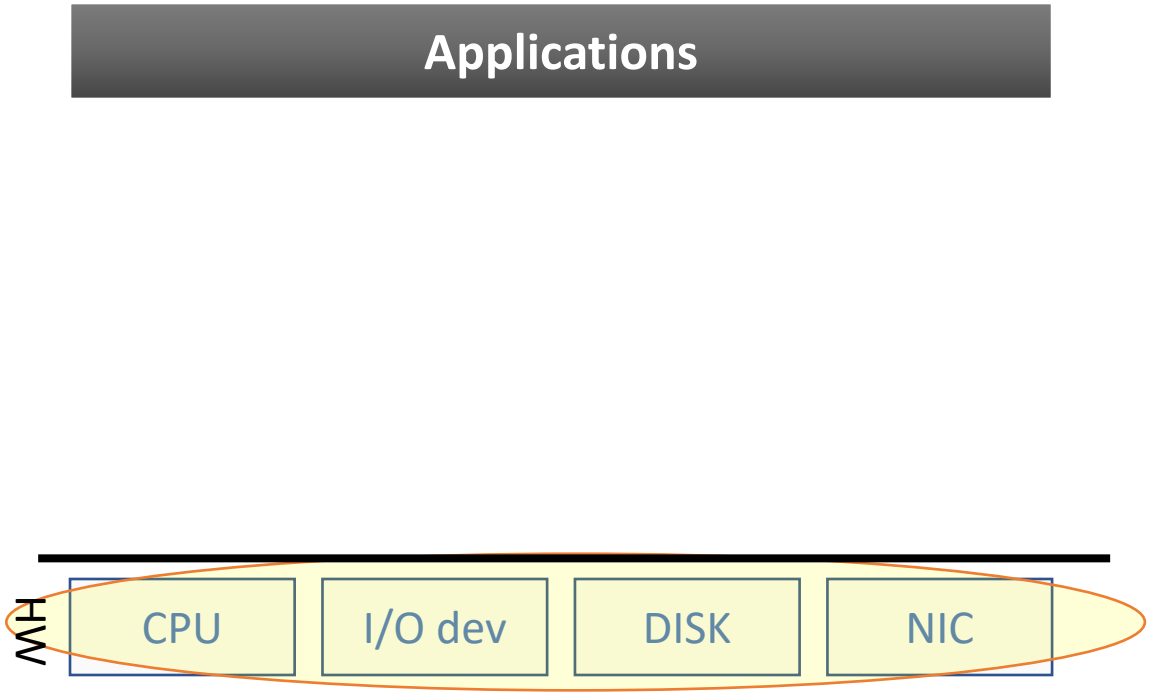
- CPU
- CPU
- GPU
- Image DSP
- Crypto
- …

Modern Technology Stack

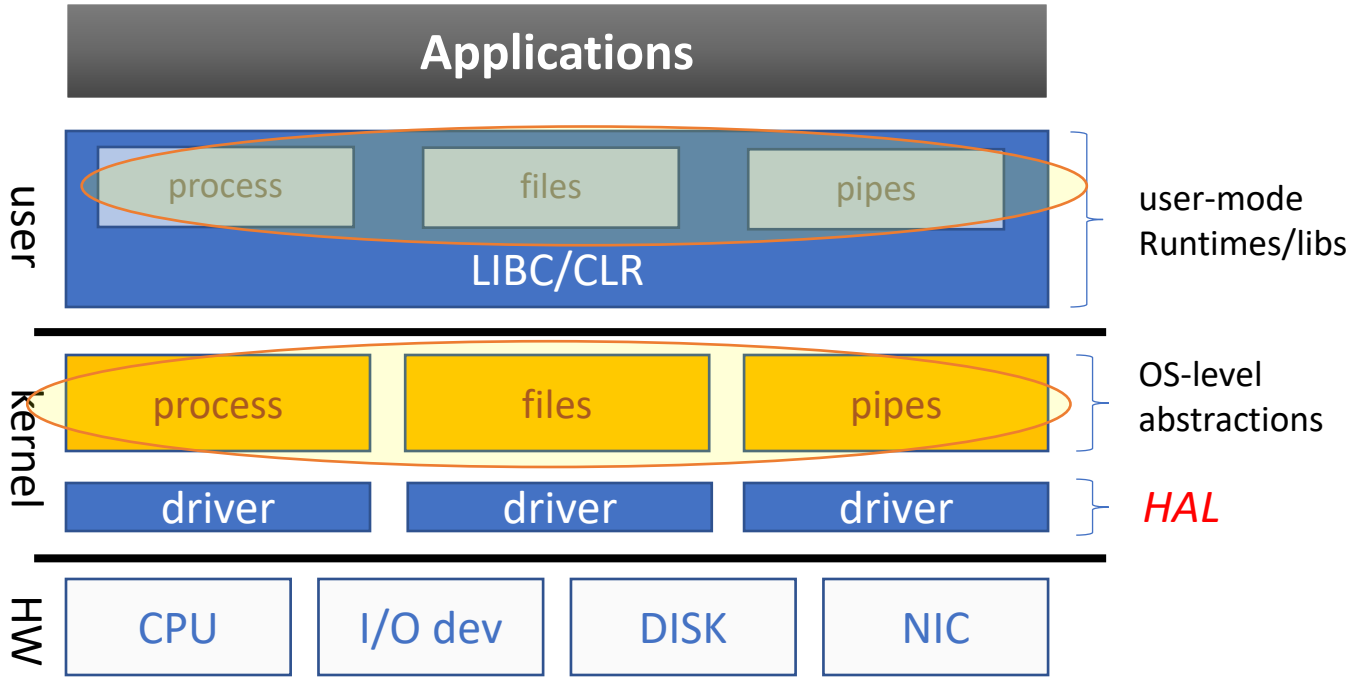
Modern Technology Stack

Modern Technology Stack

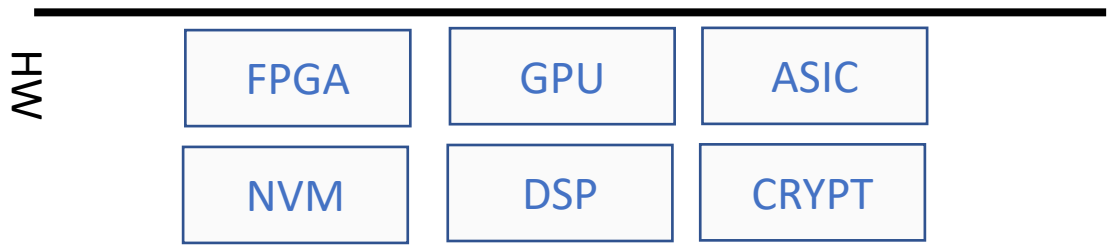
Modern Technology Stack



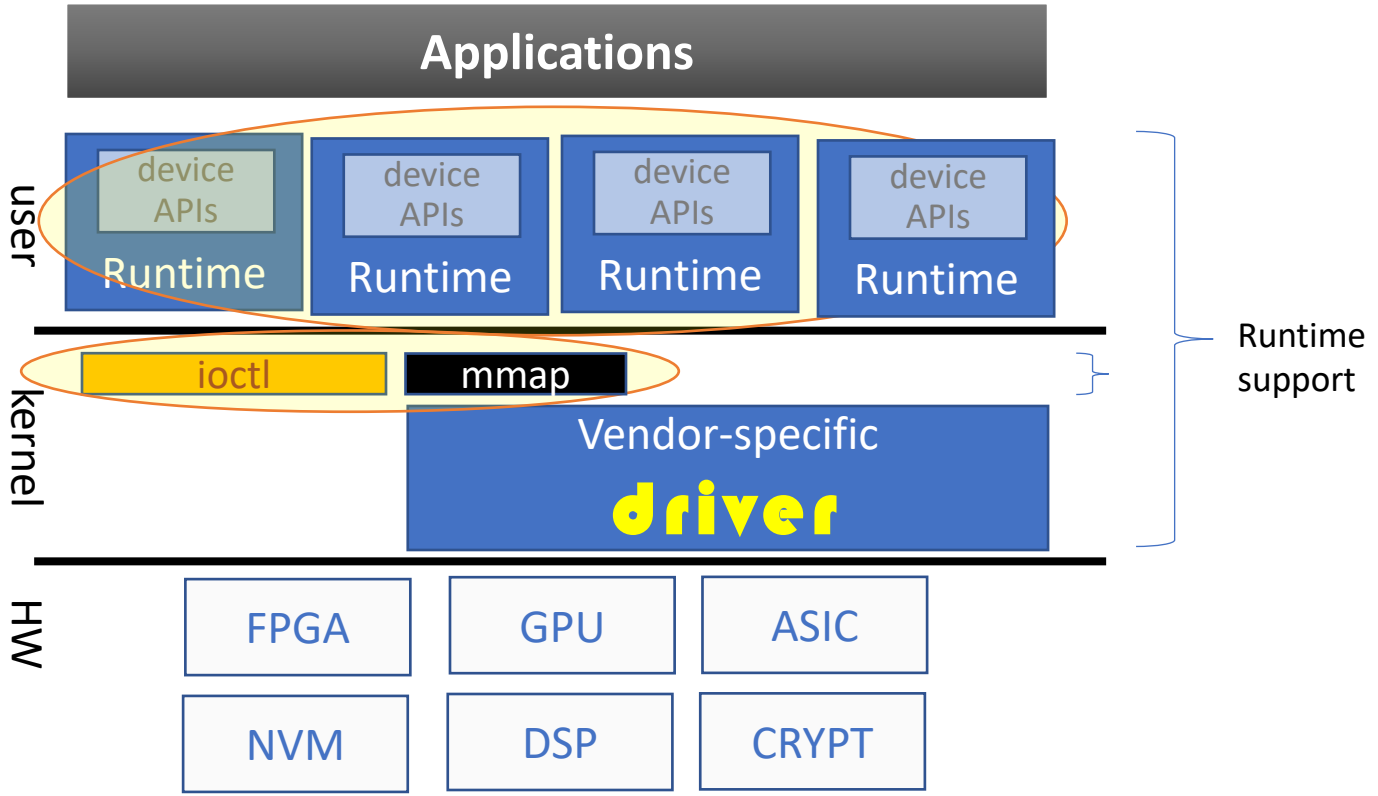
Modern Technology Stack



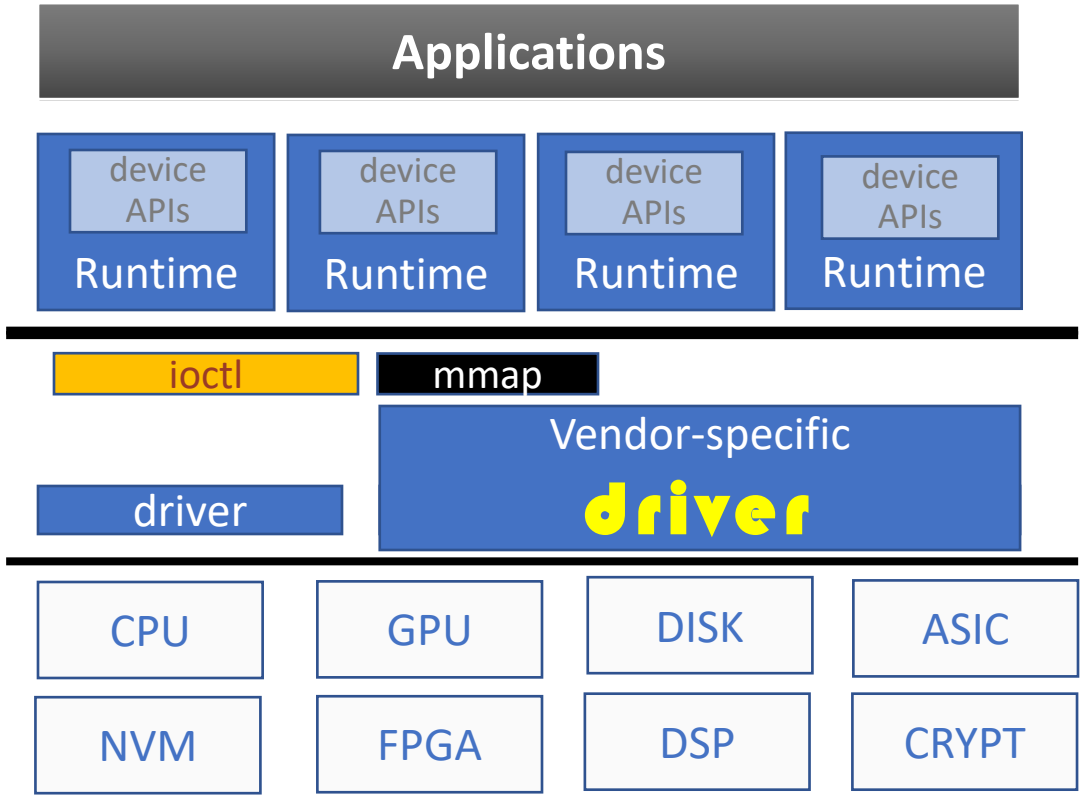
Modern Technology Stack



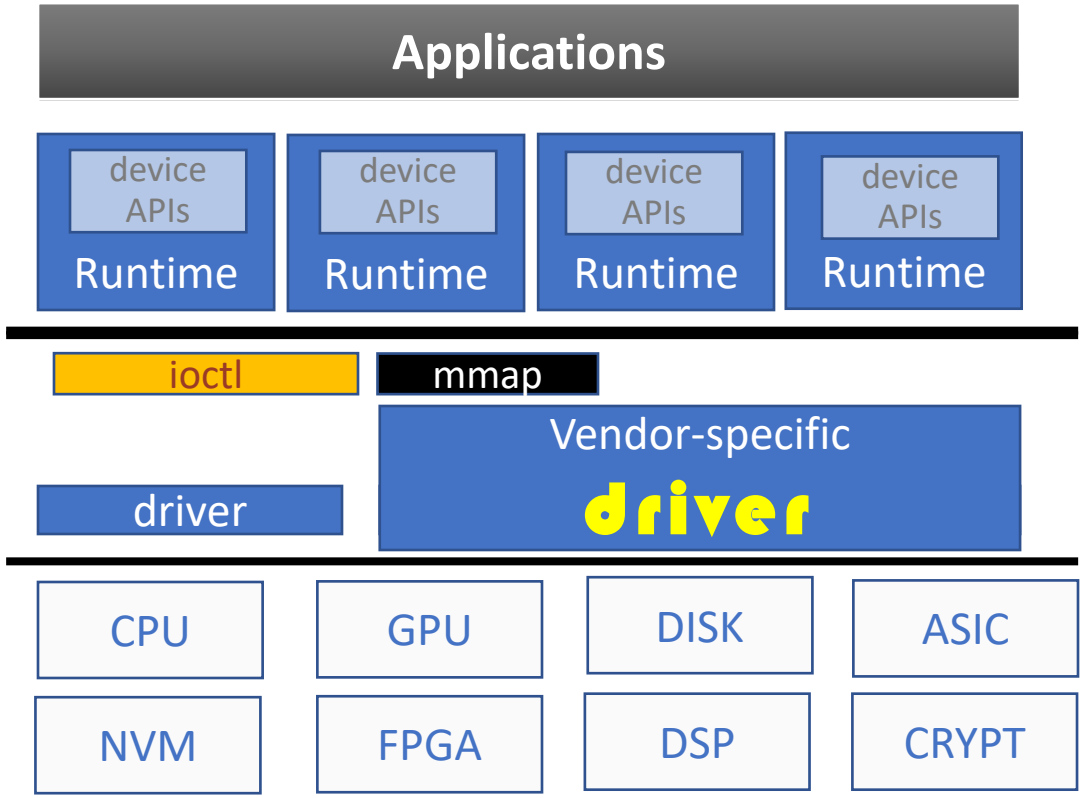
Modern Technology Stack



Concurrency and Parallelism are Everywhere



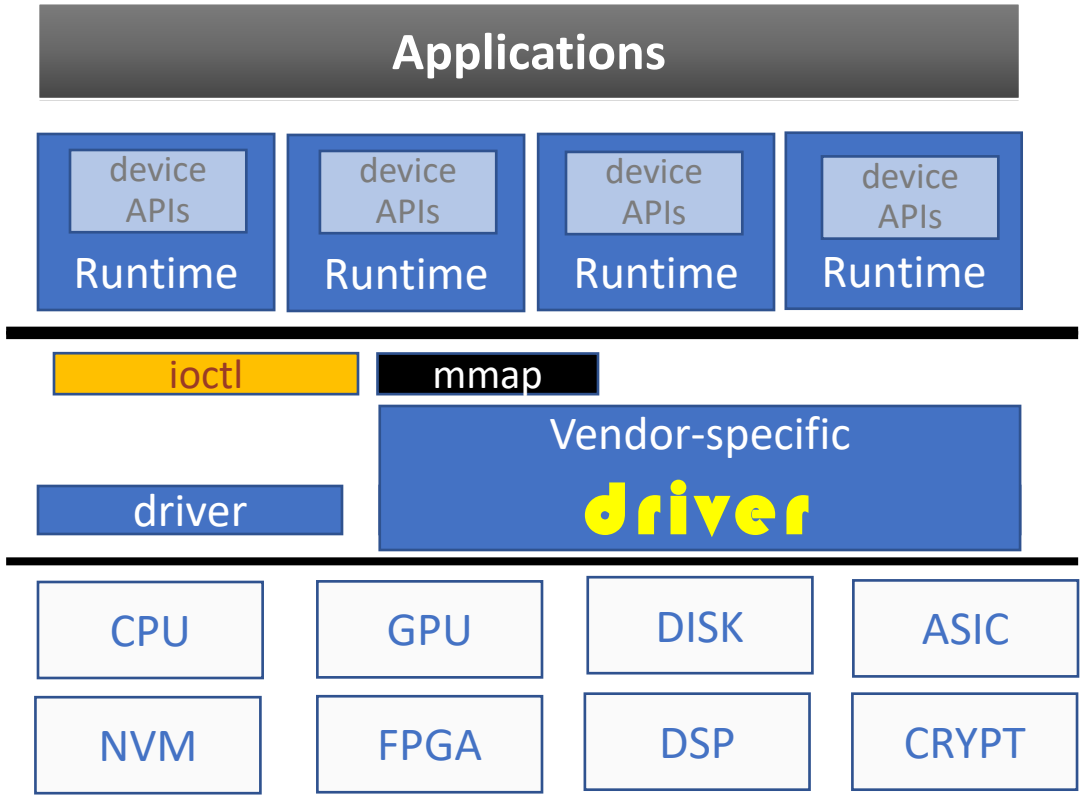
Concurrency and Parallelism are Everywhere



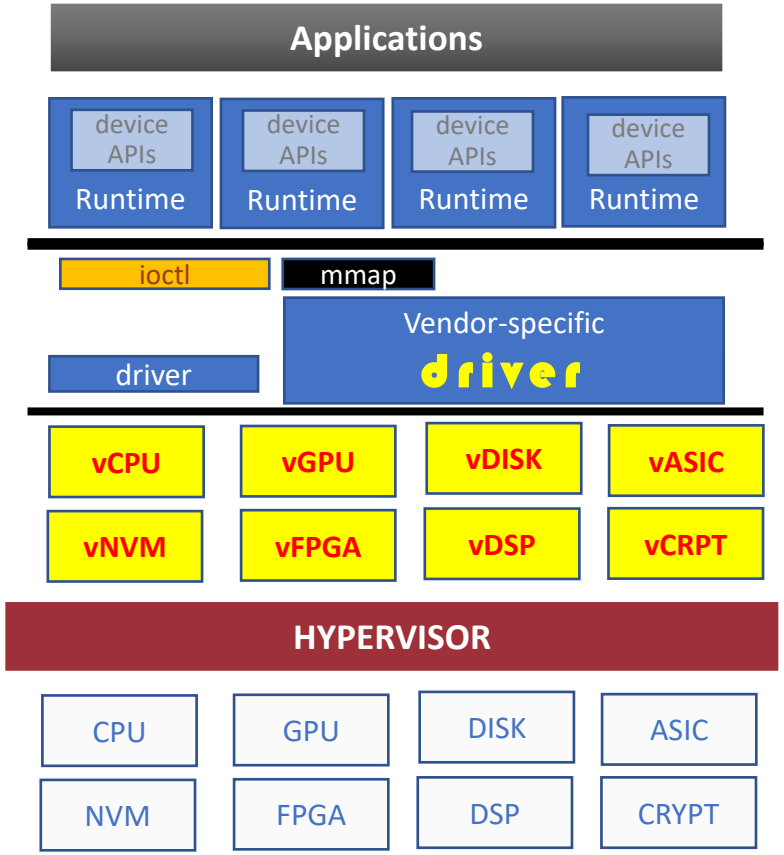
Wait!

- What's concurrency?
- What's parallelism?

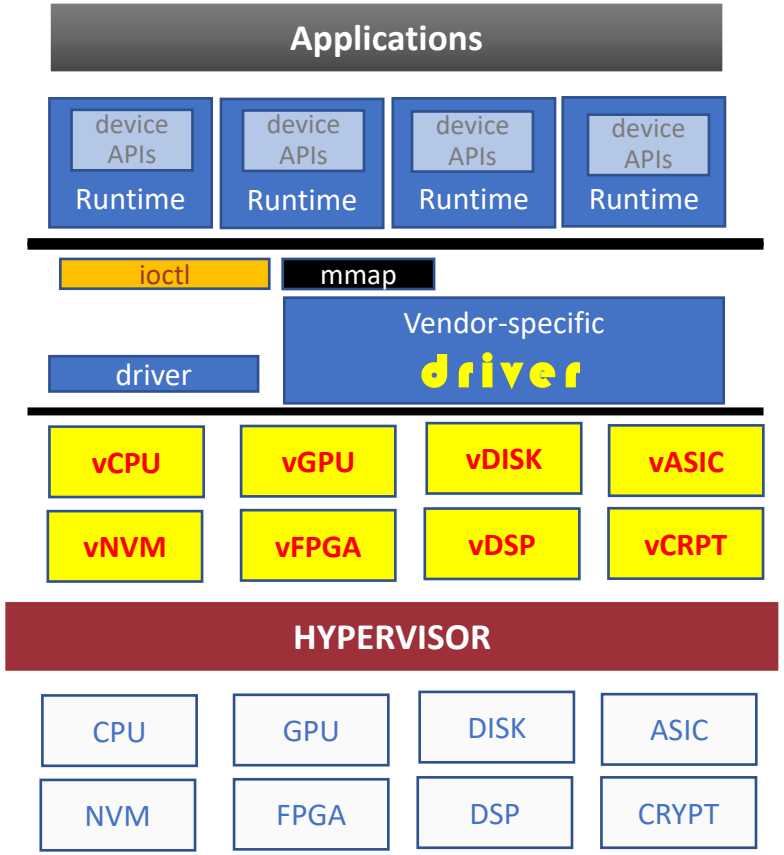
Concurrency and Parallelism are Everywhere



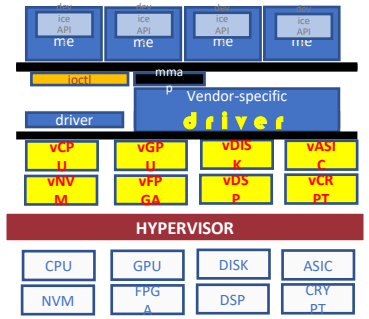
Concurrency and Parallelism are Everywhere



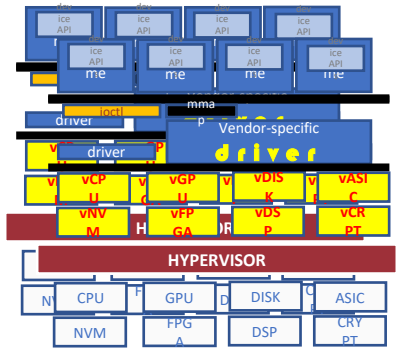
Concurrency and Parallelism are Everywhere



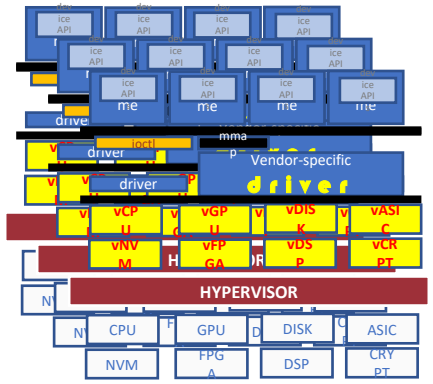
Concurrency and Parallelism are Everywhere



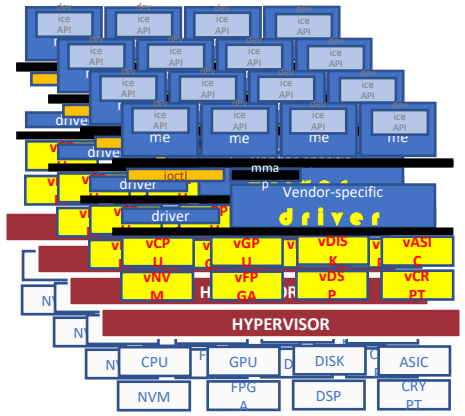
Concurrency and Parallelism are Everywhere



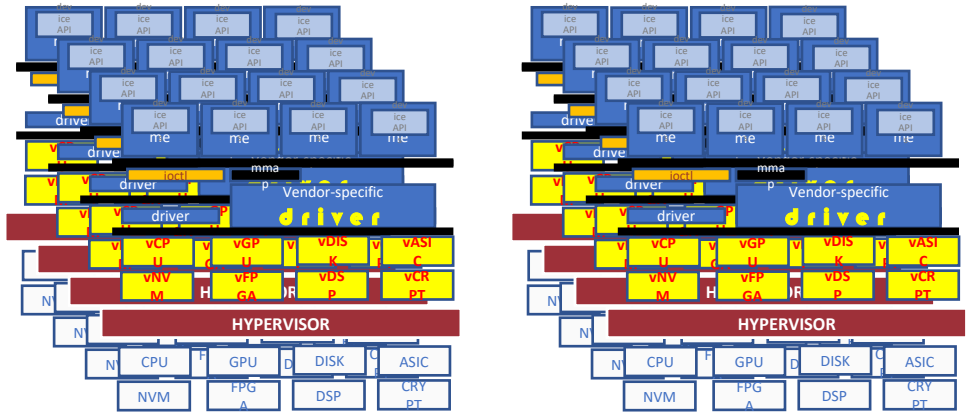
Concurrency and Parallelism are Everywhere



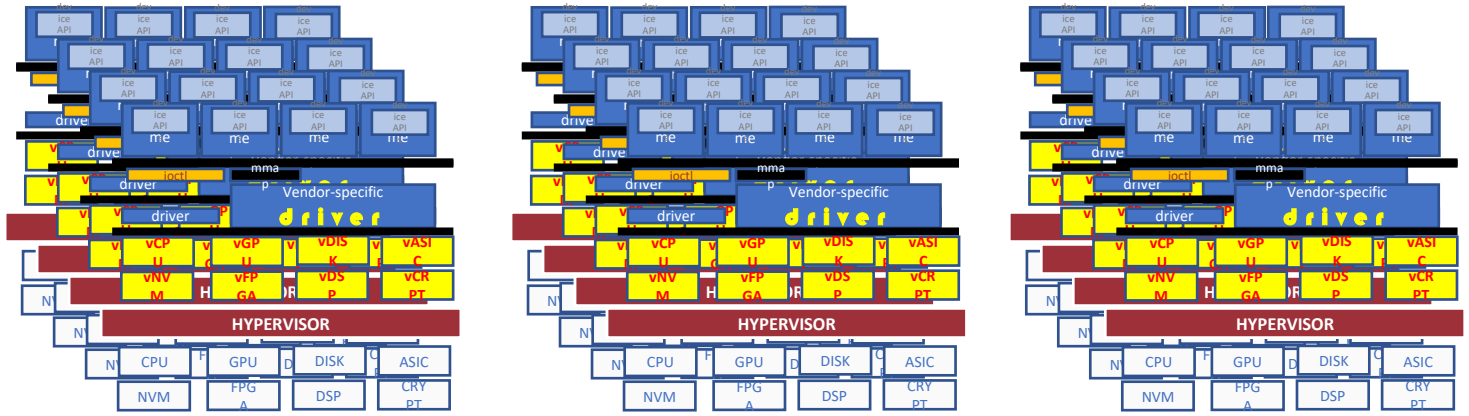
Concurrency and Parallelism are Everywhere



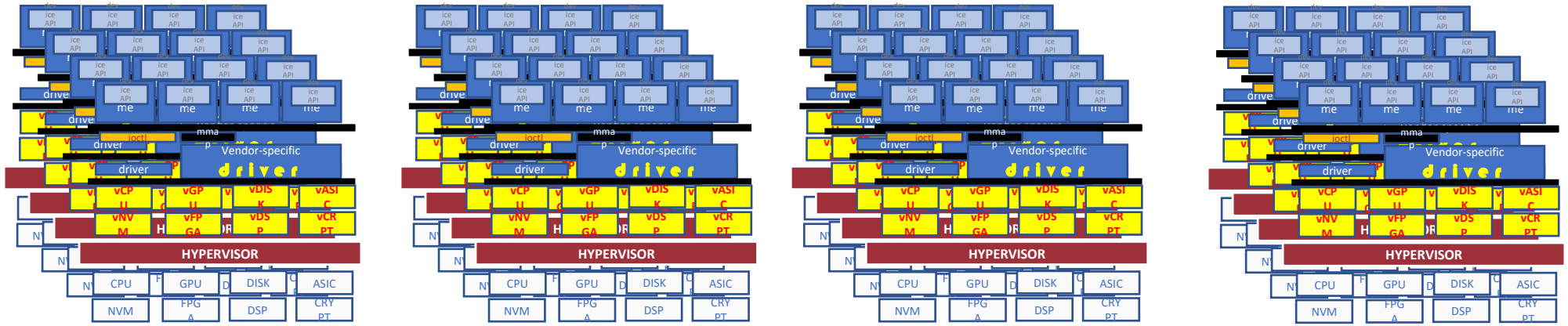
Concurrency and Parallelism are Everywhere



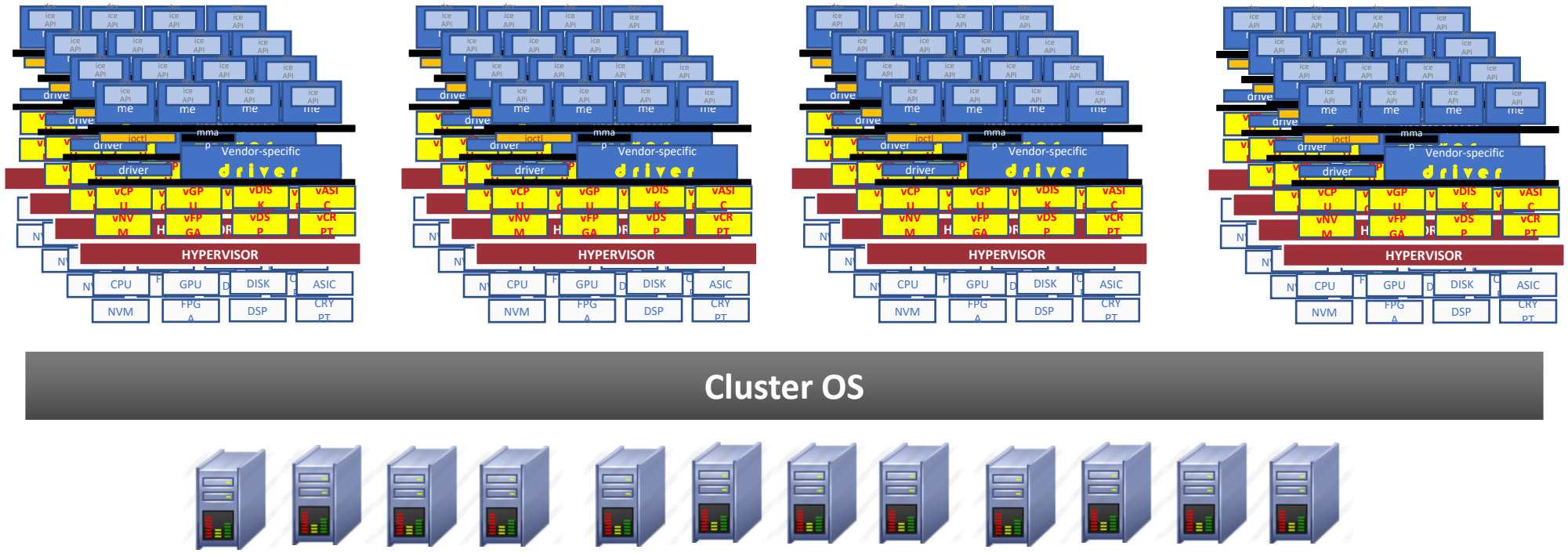
Concurrency and Parallelism are Everywhere



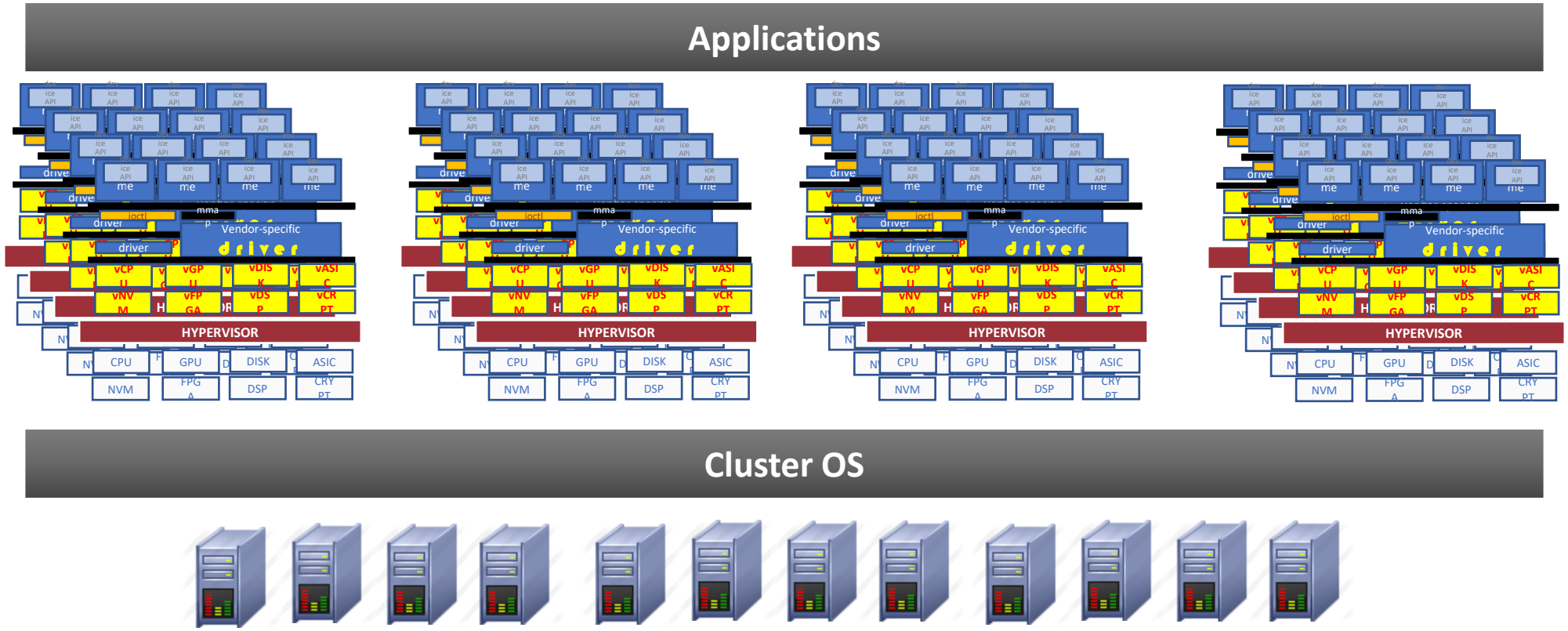
Concurrency and Parallelism are Everywhere



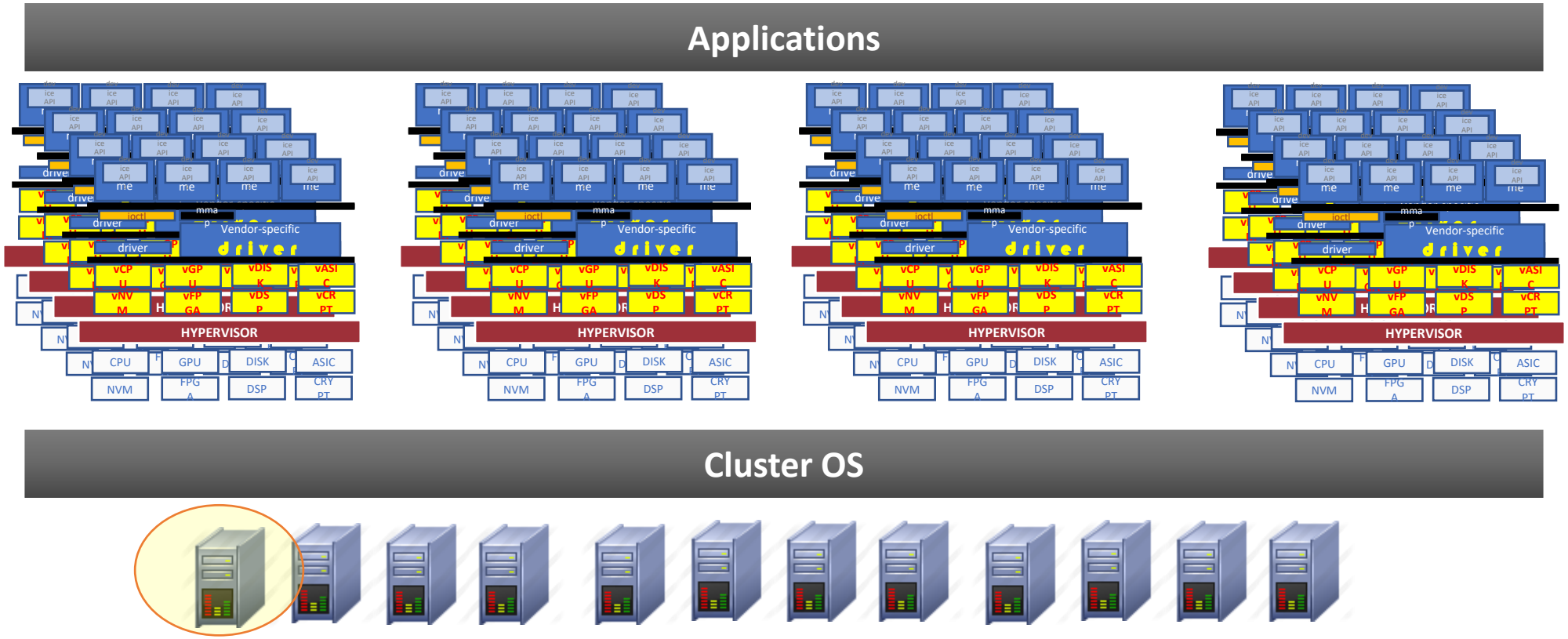
Concurrency and Parallelism are Everywhere



Concurrency and Parallelism are Everywhere

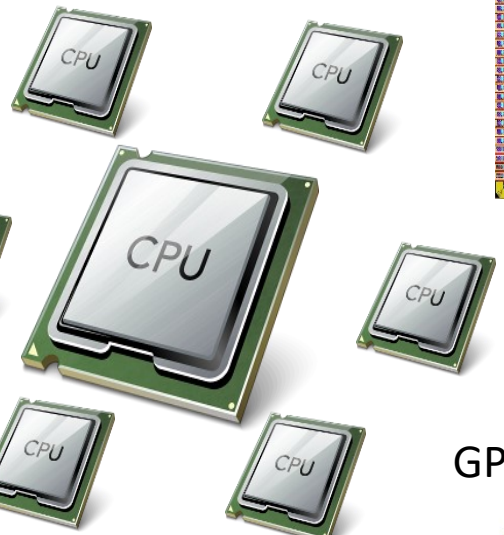


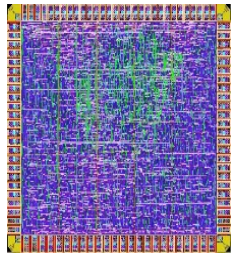
Concurrency and Parallelism are Everywhere



Concurrency and Parallelism are everywhere

DSP

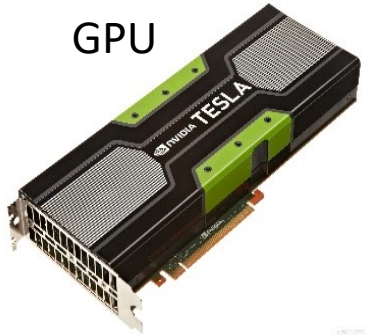




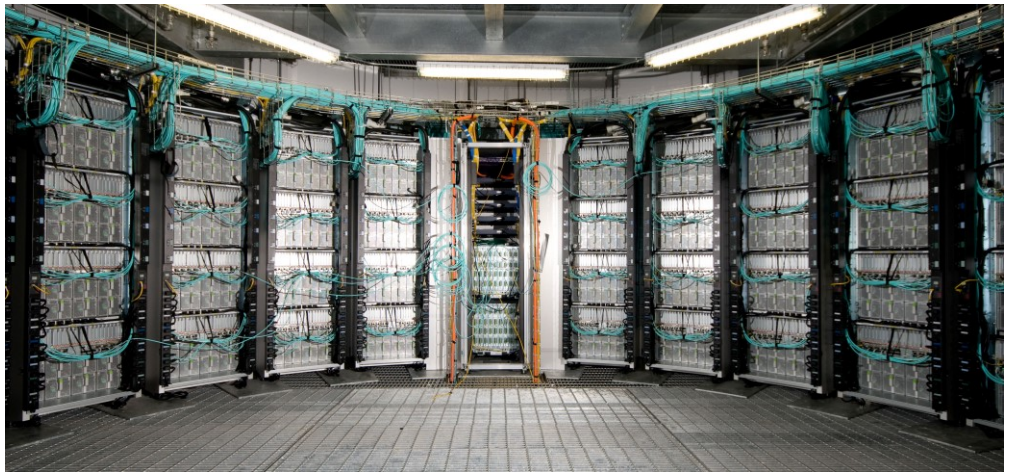
FPGA

Crypto

GPU



- CPU(s)
- GPU
- Image DSP
- Crypto
- …

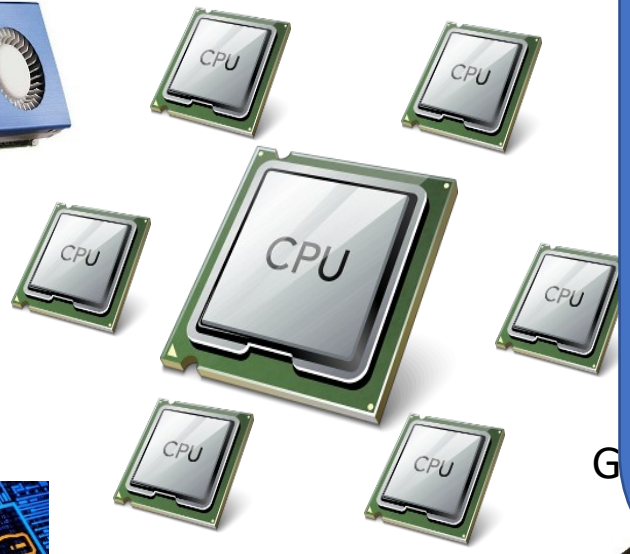


Concurrency and Parallelism are everywhere

How much parallel and concurrent programming have you learned so far?

- CPU(s)
- GPU
- Image DSP
- Crypto
- …

DSP

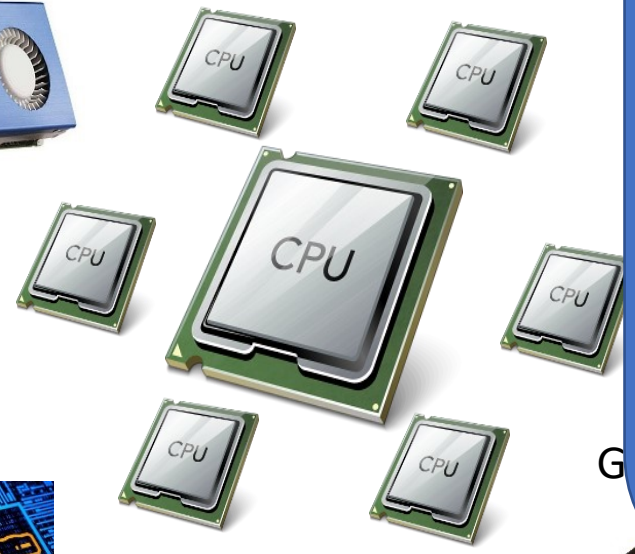


Crypto

G

Concurrency and Parallelism are everywhere

DSP



Crypto

G

How much parallel and concurrent programming have you learned so far?

- Concurrency/parallelism can't be avoided anymore (want a job?)
- A program or two playing with locks and threads isn't enough
- I've worked in industry a lot—I know

Course goal is to expose you to lots of ways of programming systems like these

…So "you should take this course because it's good for you" (eat your #$(*& kale!)

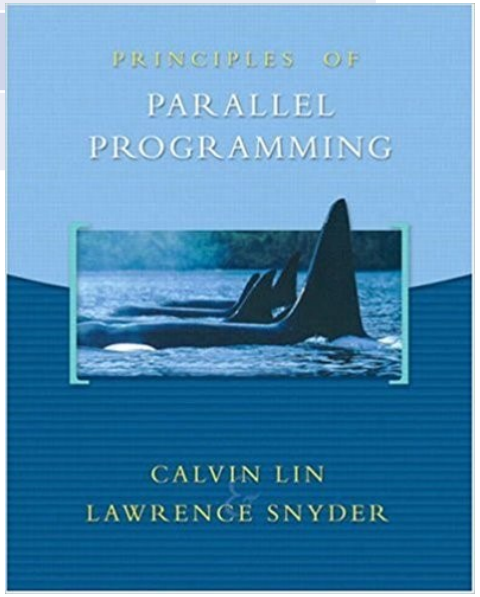
- CPU(s)
- GPU
- Image DSP
- Crypto
- …

Goal: Make Concurrency Your Close Friend
Method: Use Many Different Approaches to Concurrency

Abstract	Concrete
Locks and Shared Memory Synchronization	Prefix Sum with pthreads
Language Support	Go lab: condition variables, channels, go routines Rust lab: 2PC
Parallel Architectures	GPU Programming Lab ***(Optional) FPGA Programming Lab***
HPC	Optional MPI lab
Distributed Computing / Big Data	Rust 2PC / MPI labs
Modern/Advanced Topics	• Specialized Runtimes / Programming Models • Auto-parallelization • Race Detection
Whatever Interests YOU	Project

Logistics Reprise

Course Name:	CS378H – Concurrency: Honors
Unique Number:	52670
Lectures:	TTh 9:30-11:00AM WAG 420
Class Web Page:	http://www.cs.utexas.edu/users/rossbach/cs378h
Instructor:	Chris Rossbach
TA:	Karthik Velayutham
Text:	Principles of Parallel Programming (ISBN-10: 0321487907)



Seriously, read the syllabus!
Also, start Lab 1!

Two Super-Serious Notes

Two Super-Serious Notes

- Inclusivity and respect are *absolute* musts

Two Super-Serious Notes

- Inclusivity and respect are *absolute* musts

- Don’t make your repos public or look at other people’s public repos

Two Super-Serious Notes

- Inclusivity and respect are *absolute* musts

- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos

Two Super-Serious Notes

- Inclusivity and respect are *absolute* musts

- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos

Two Super-Serious Notes

- Inclusivity and respect are *absolute* musts

- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos

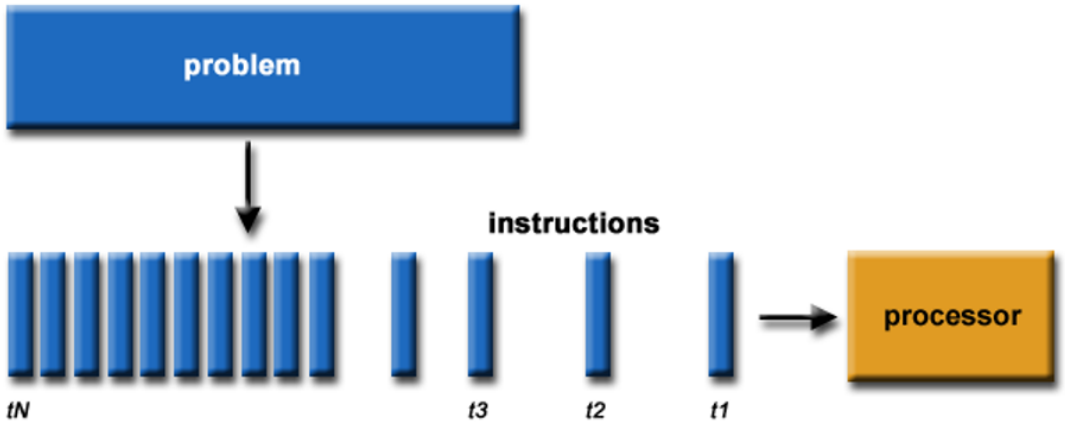
Two Super-Serious Notes

- Inclusivity and respect are *absolute* musts

- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos
- Don’t make your repos public or look at other people’s public repos

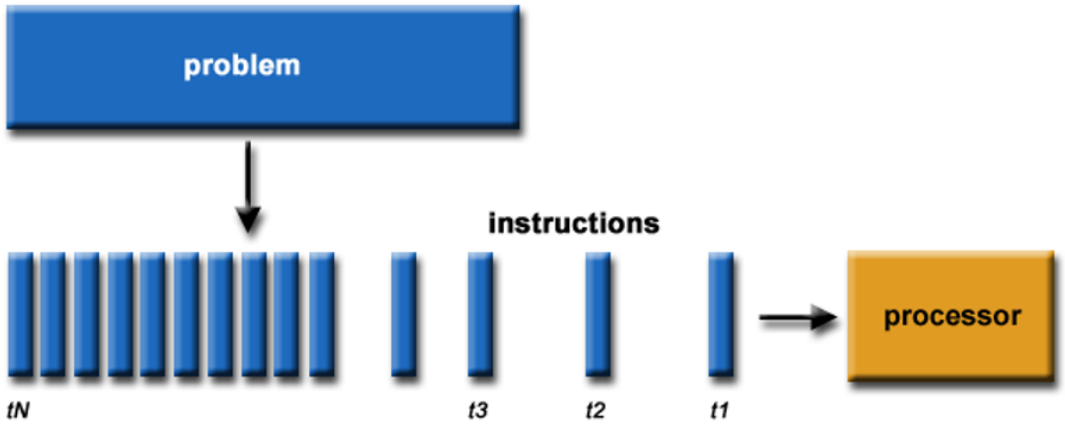
Serial vs. Parallel Program

Serial vs. Parallel Program

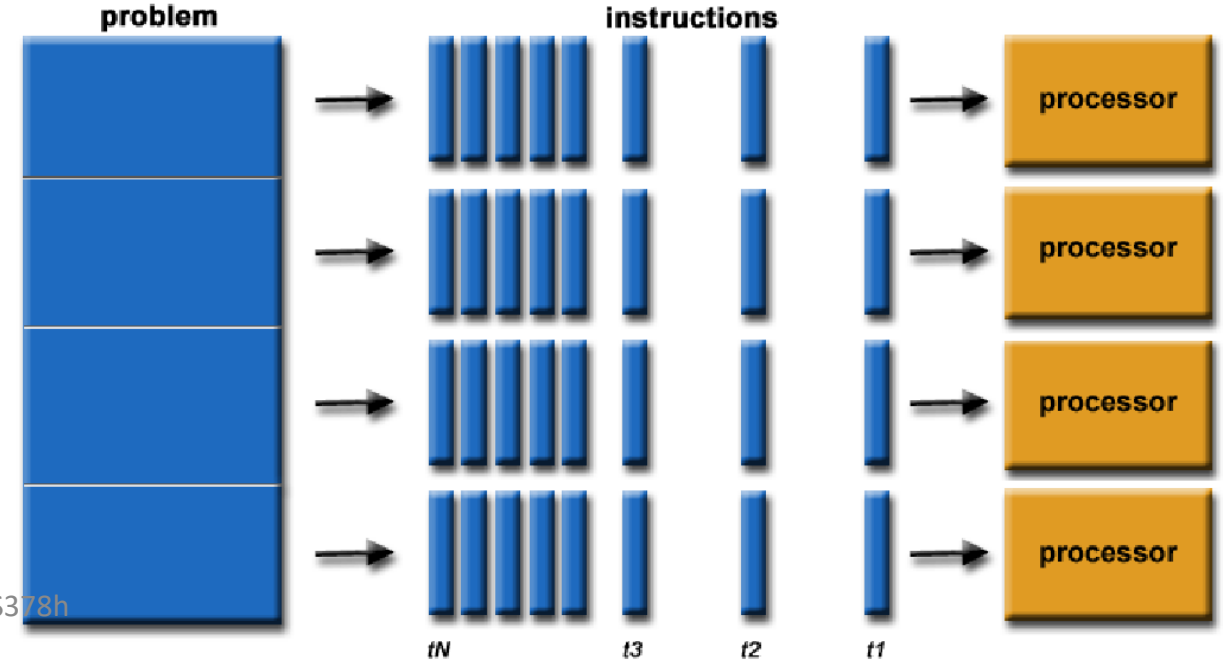


One instruction at a time (apparently)

Serial vs. Parallel Program

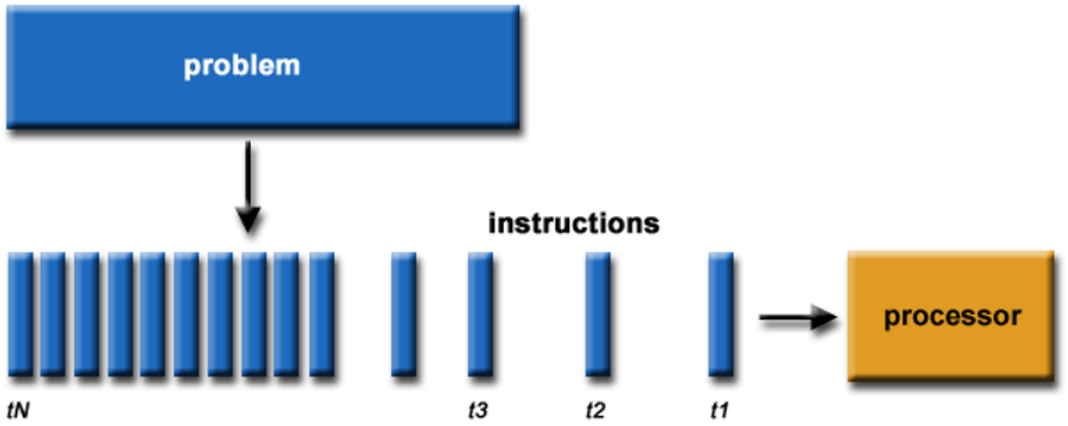


One instruction at a time (apparently)

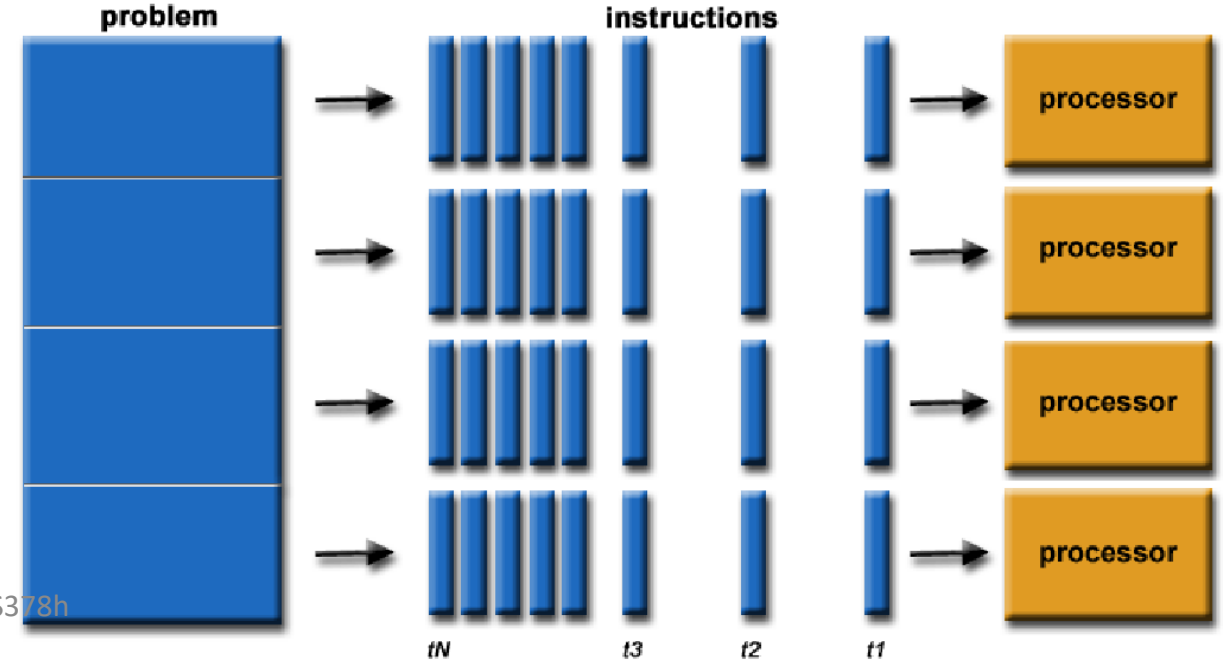


Multiple instructions in parallel

Serial vs. Parallel Program

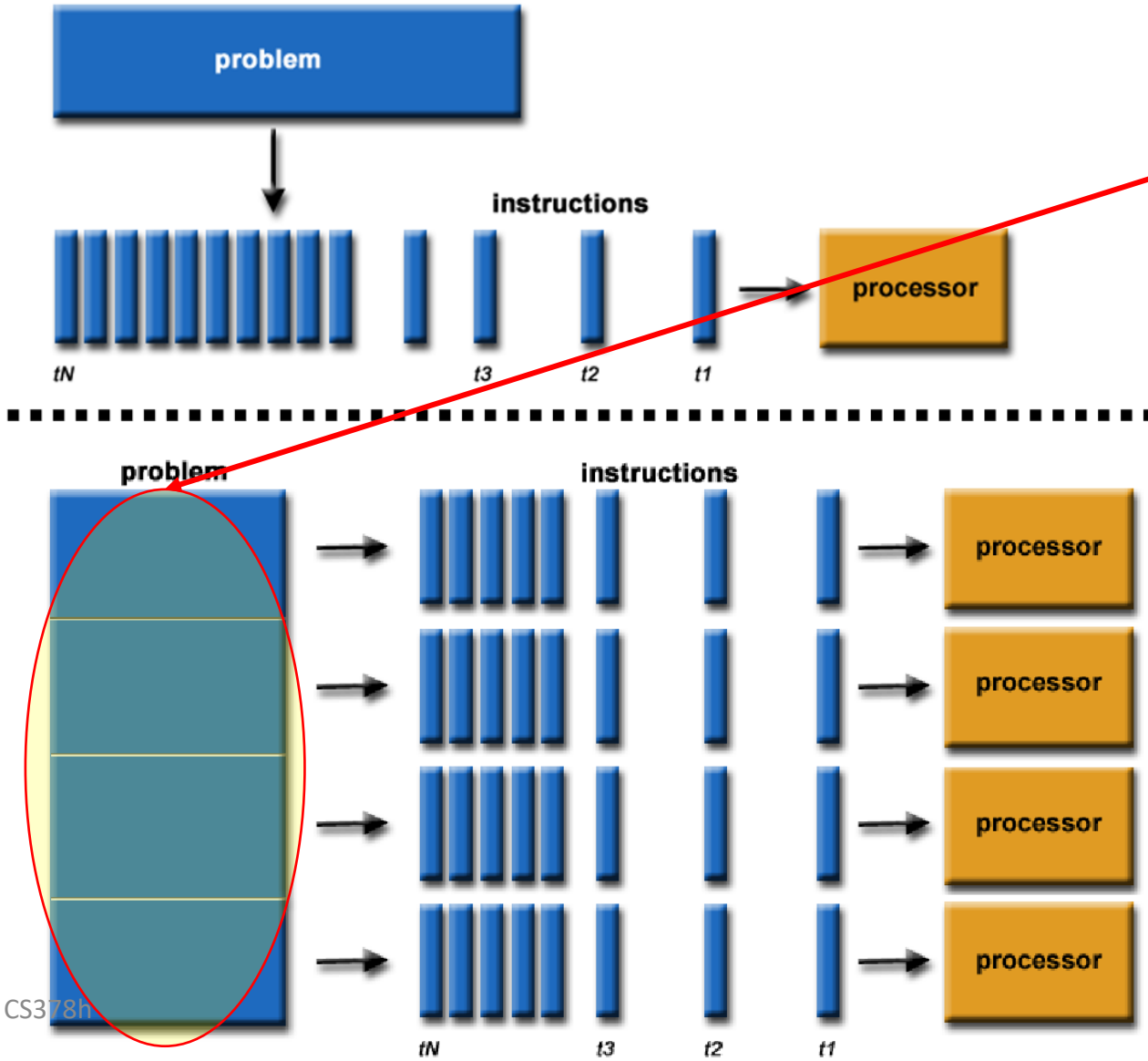


Key concerns:



Multiple instructions in parallel

Serial vs. Parallel Program

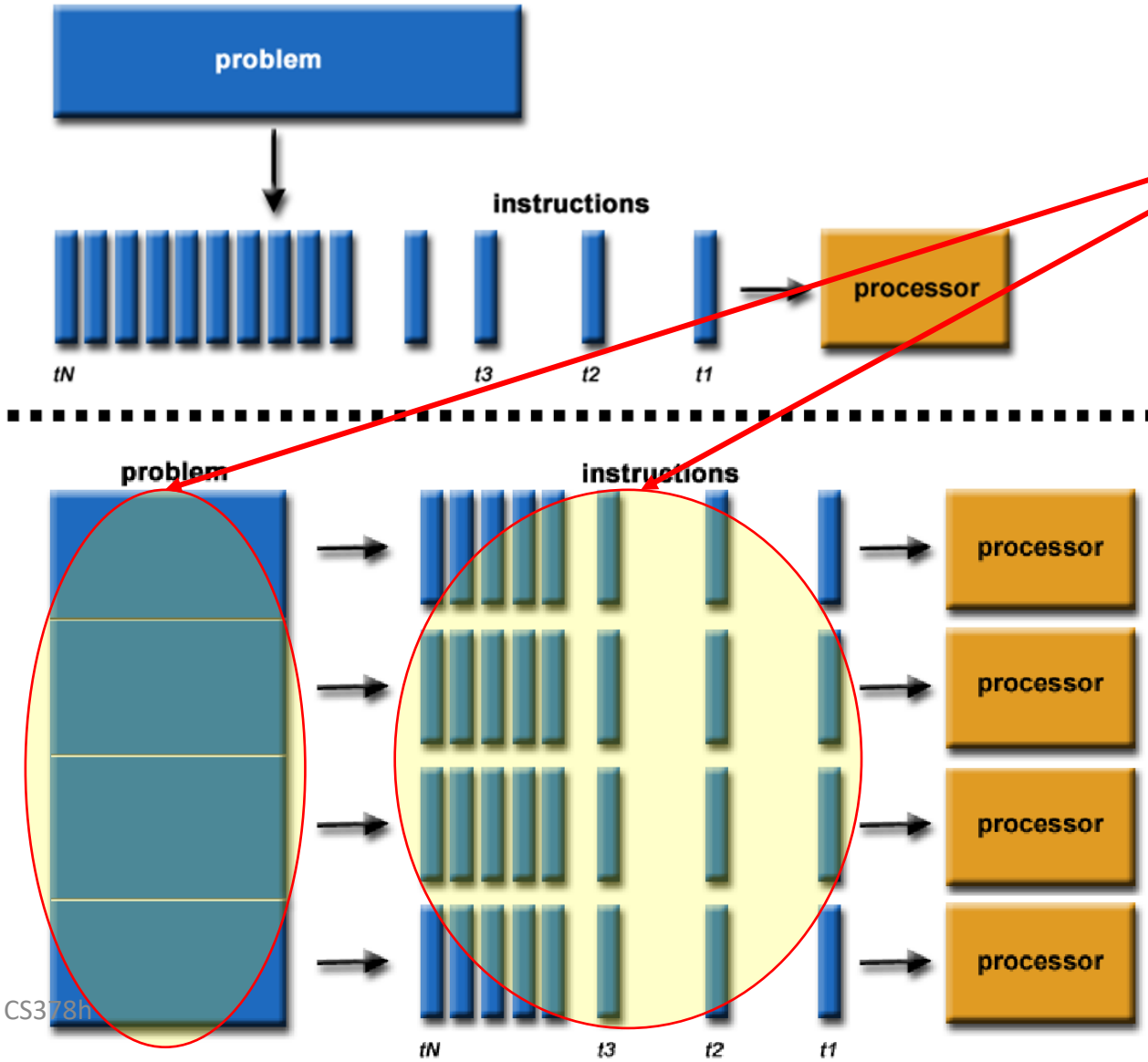


Key concerns:

- Programming model

Multiple instructions in parallel

Serial vs. Parallel Program

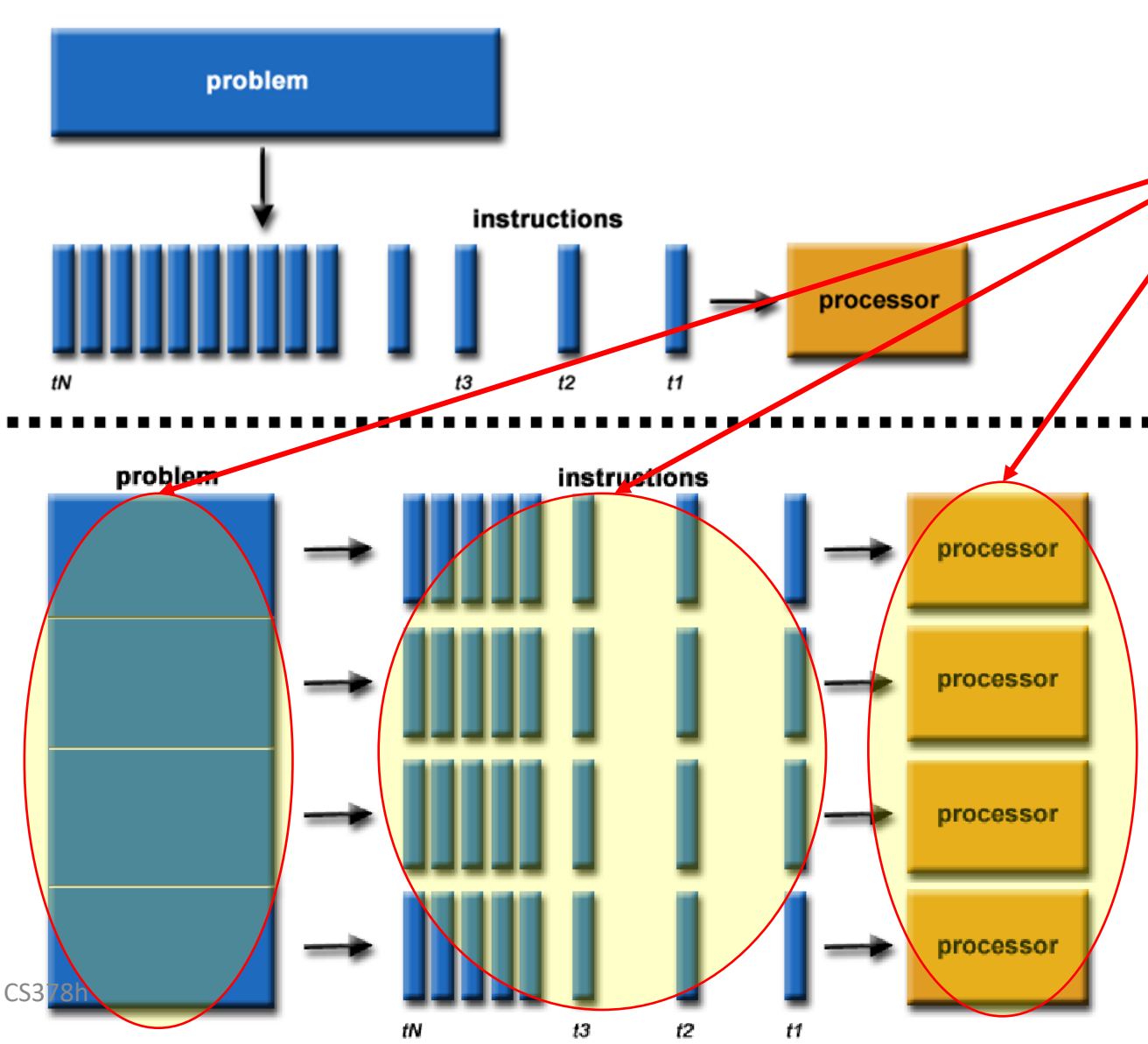


Key concerns:

- Programming model
- Execution Model

Multiple instructions in parallel

Serial vs. Parallel Program

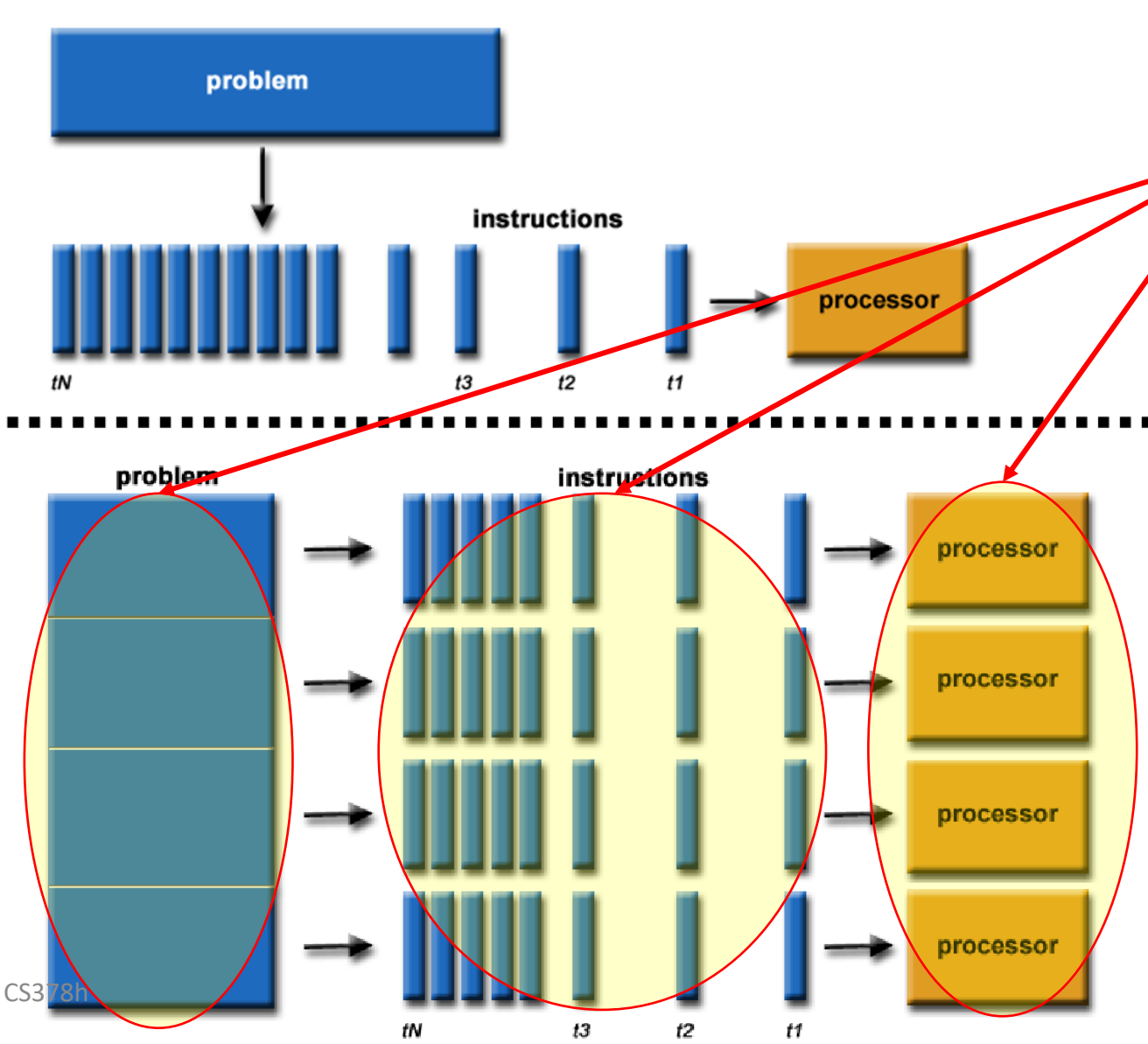


Key concerns:

- Programming model
- Execution Model
- Performance/Efficiency

Multiple instructions in parallel

Serial vs. Parallel Program

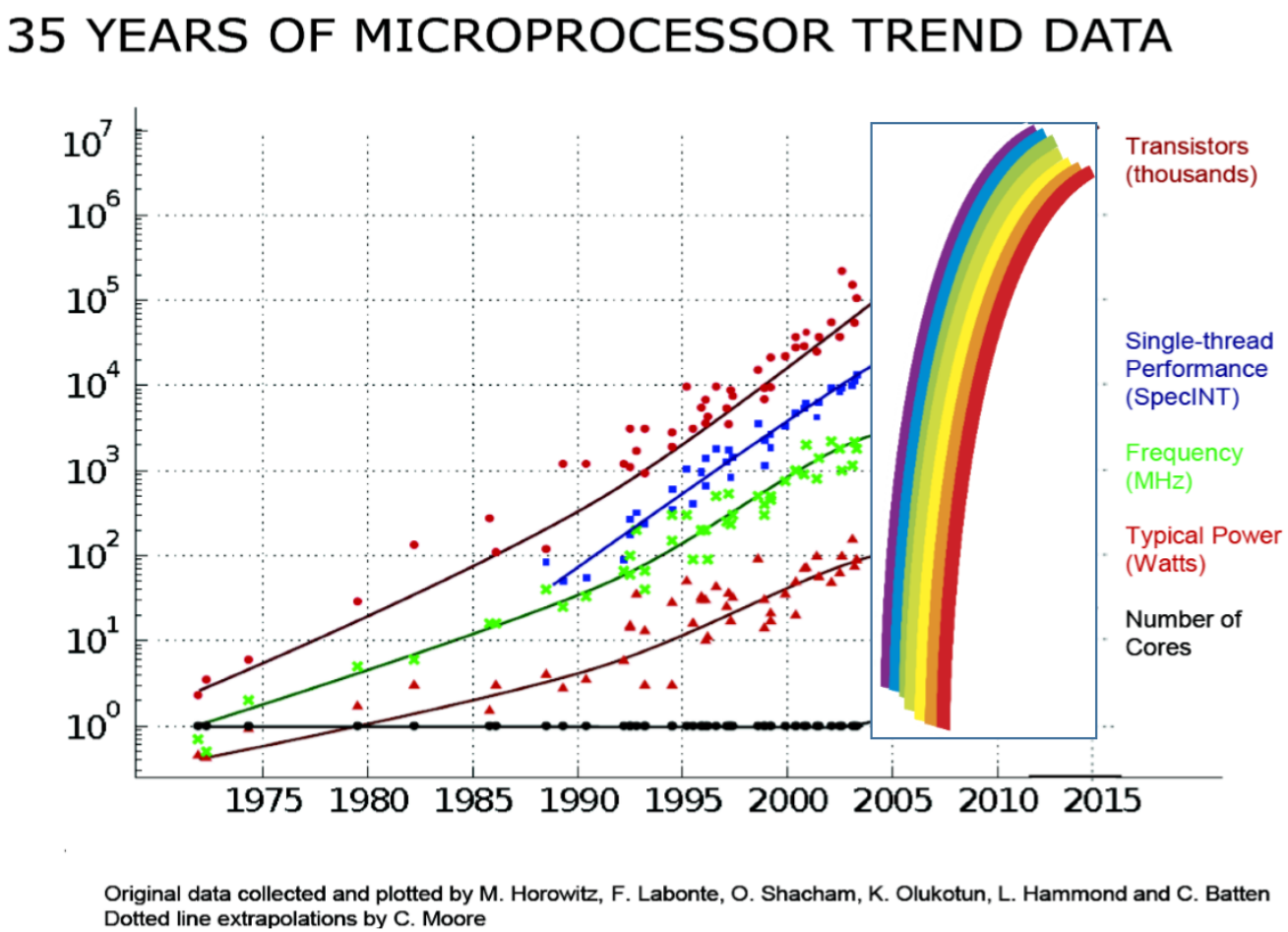


Key concerns:

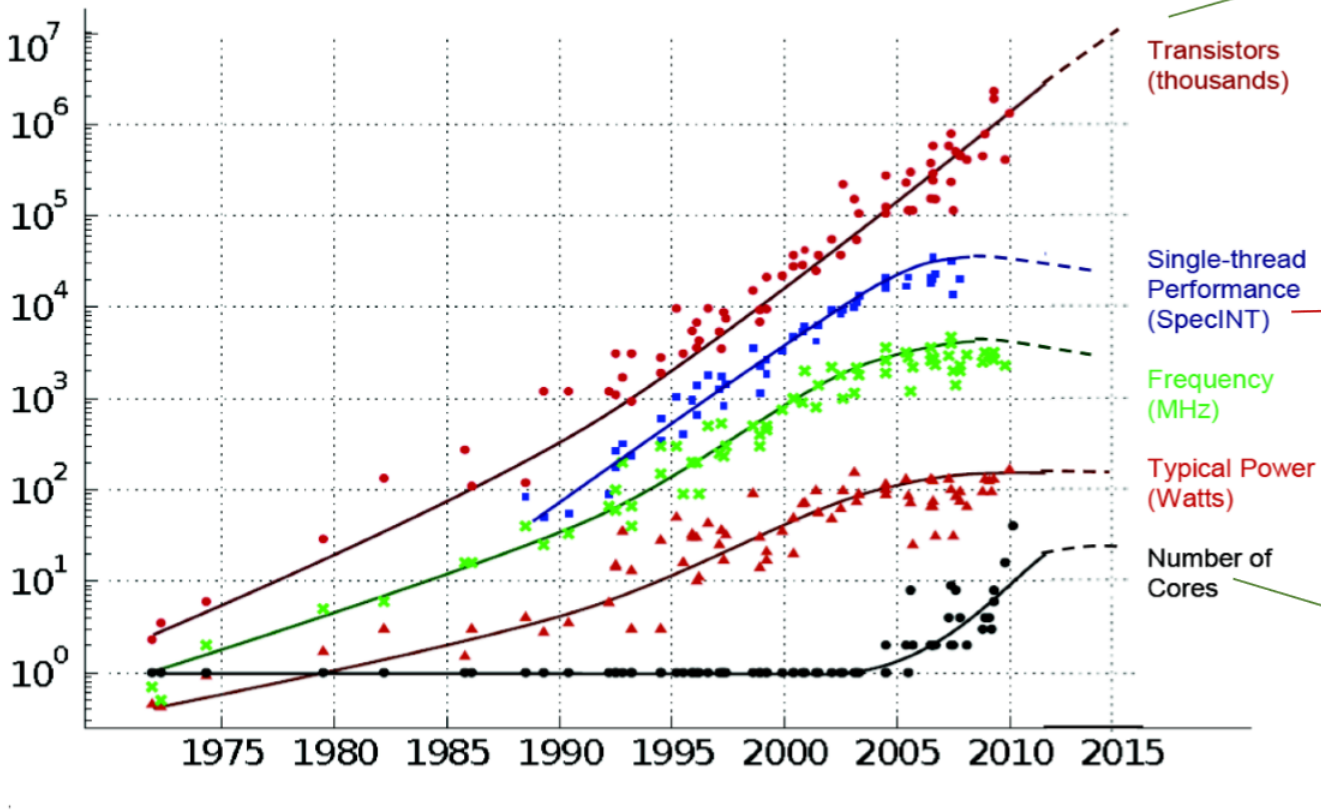
- Programming model
- Execution Model
- Performance/Efficiency
- Exposing parallelism

Multiple instructions in parallel

Free lunch…


35 YEARS OF MICROPROCESSOR TREND DATA
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0
1975 1980 1985 1990 1995 2000 2005 2010 2015
Transistors (thousands)
Single-thread Performance (SpecINT)
Frequency (MHz)
Typical Power (Watts)
Number of Cores
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Free lunch – is over ☹



Transistor number grows (Moore's law)

Sequential performance no longer improves

Cores number grows

Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

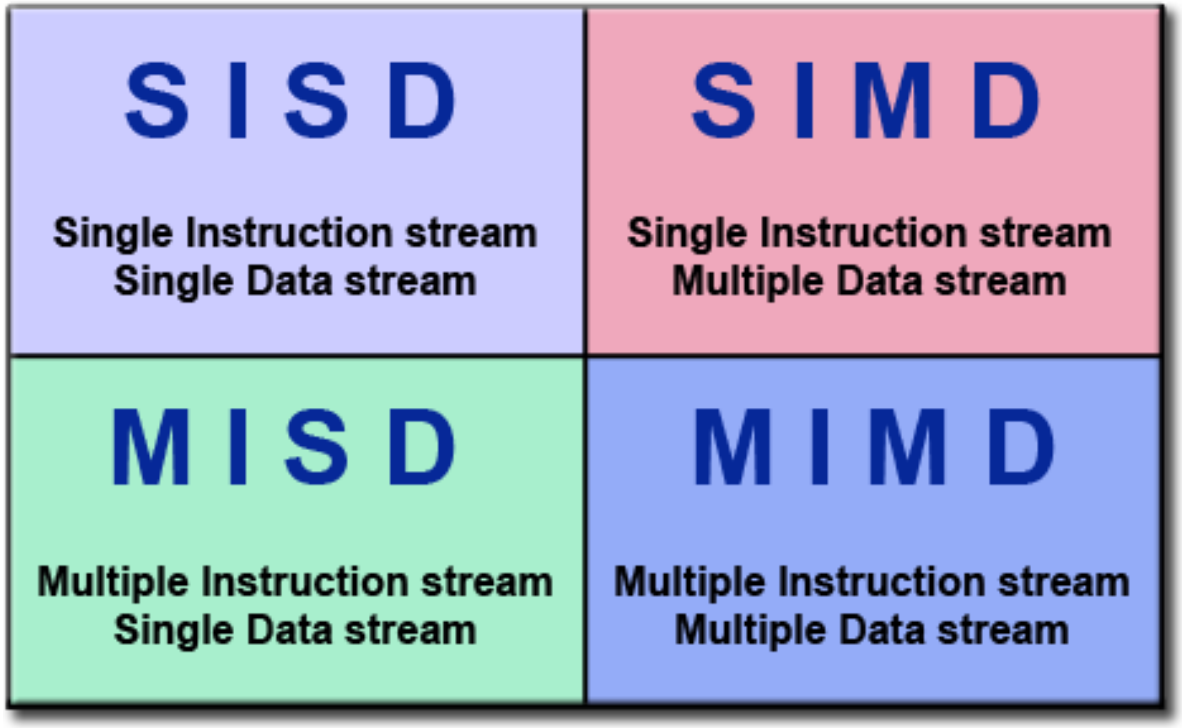
Flynn’s Taxonomy

Flynn’s Taxonomy

SISD	SIMD
MISD	MIMD

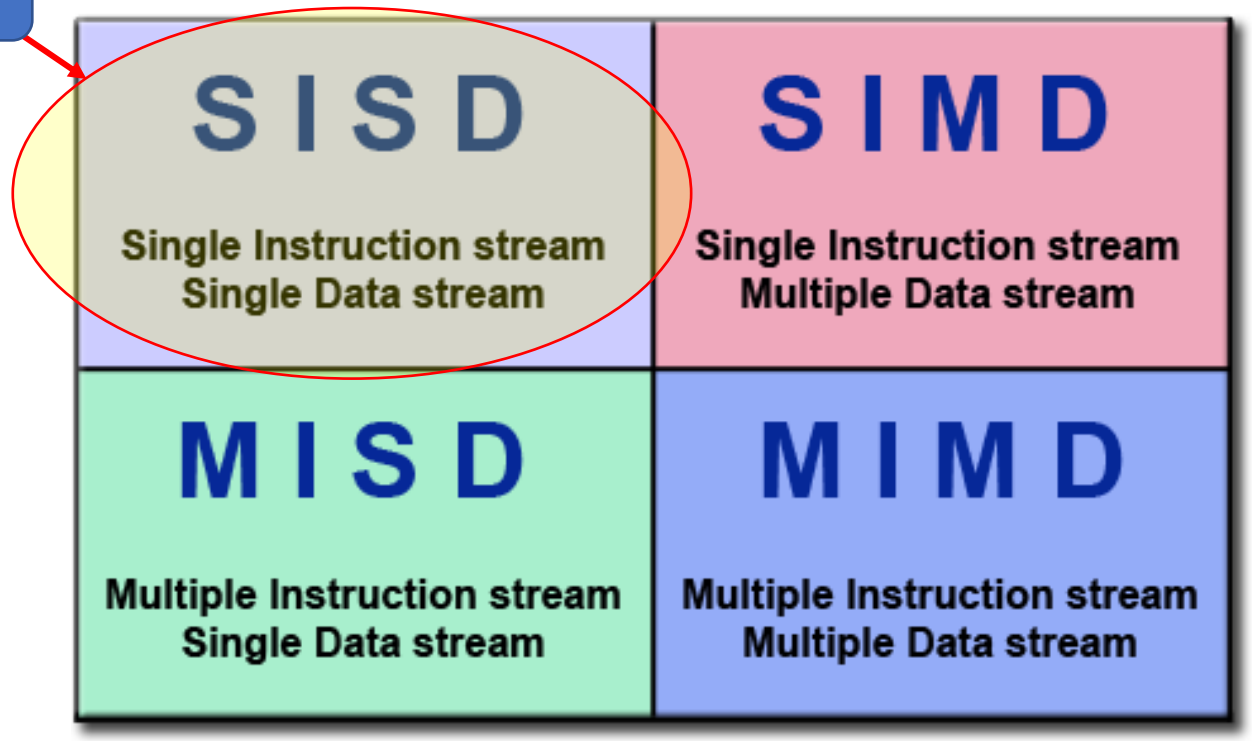
Execution Models: Flynn’s Taxonomy

Execution Models: Flynn's Taxonomy

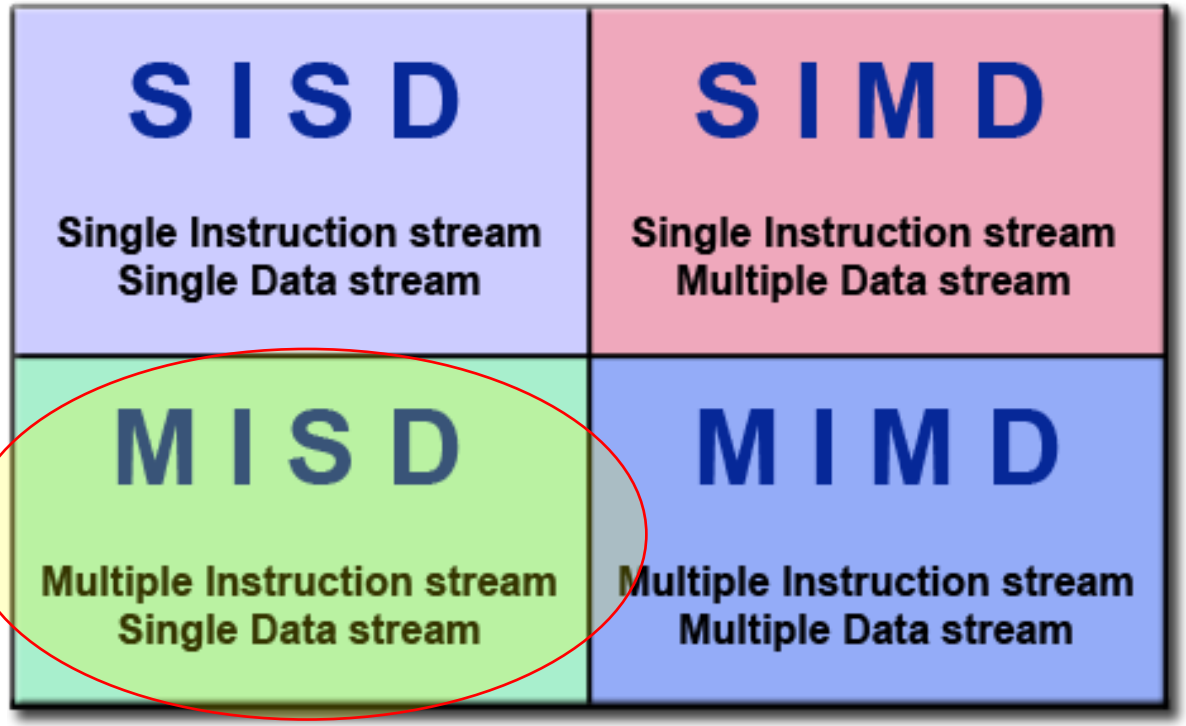


Execution Models: Flynn's Taxonomy

Normal Serial program

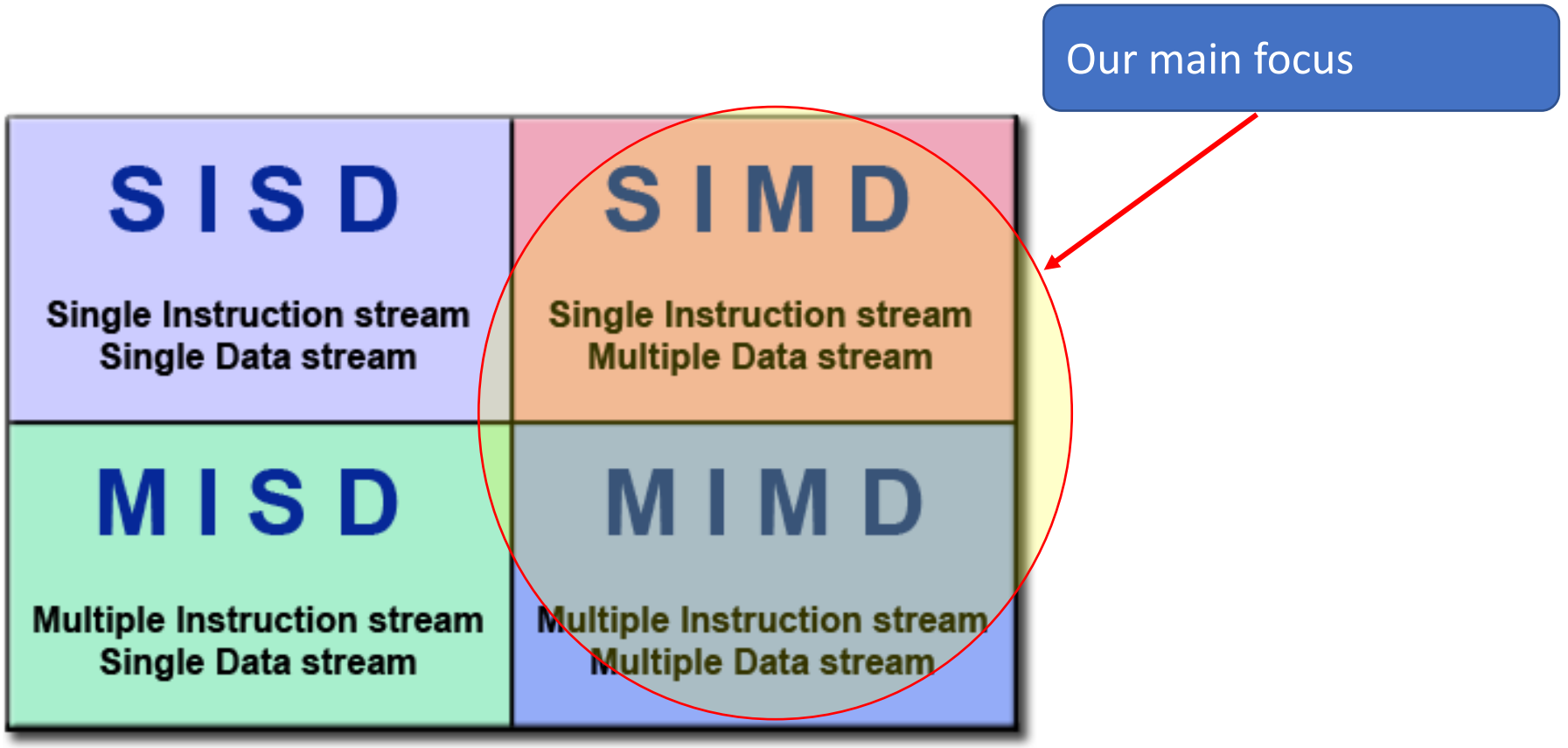


Execution Models: Flynn's Taxonomy



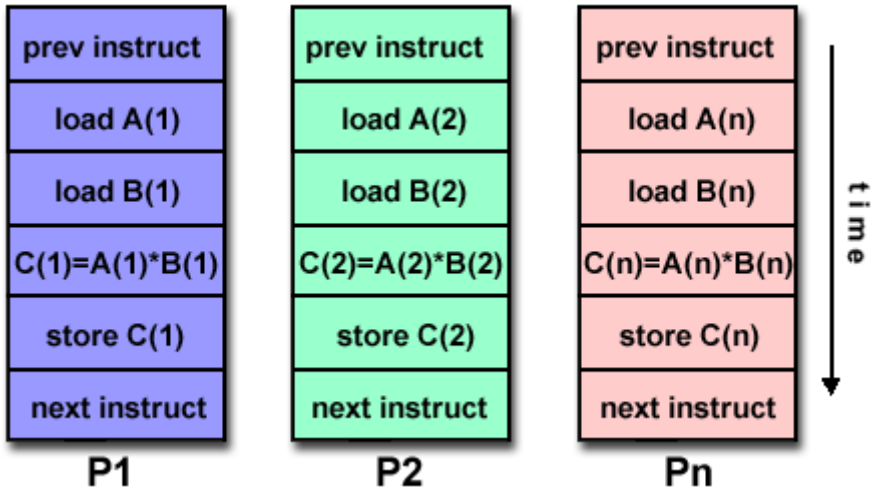
Uncommon architecture:
Fault – tolerance
Pipeline parallelism

Execution Models: Flynn's Taxonomy

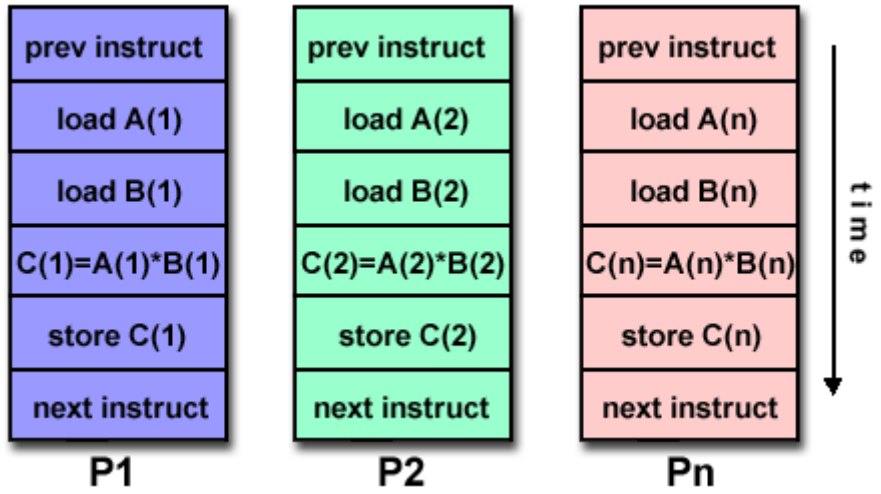


SIMD

SIMD

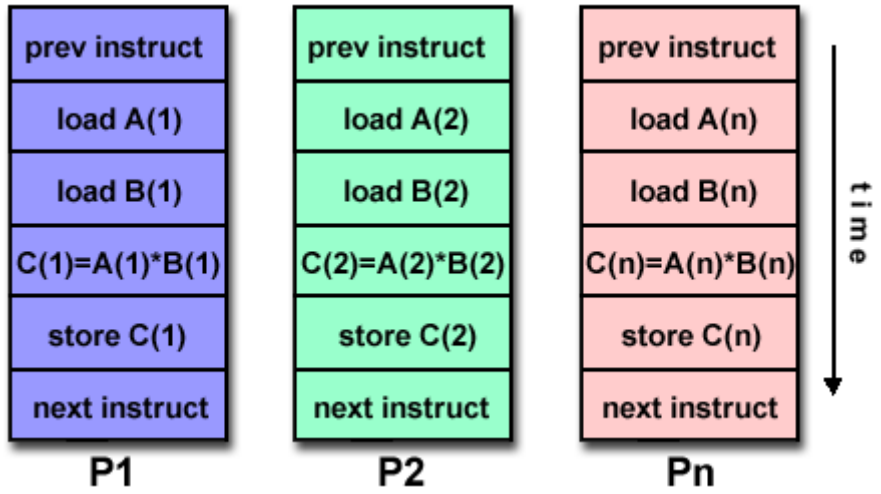

prev instruct
load A(1)
load B(1)
C(1)=A(1)*B(1)
store C(1)
next instruct
P1
prev instruct
load A(2)
load B(2)
C(2)=A(2)*B(2)
store C(2)
next instruct
P2
prev instruct
load A(n)
load B(n)
C(n)=A(n)*B(n)
store C(n)
next instruct
Pn
time

SIMD

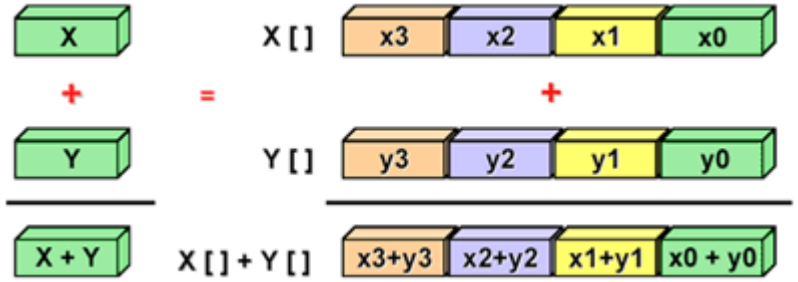


- Example: vector operations (e.g., Intel SSE/AVX, GPU)

SIMD



- Example: vector operations (e.g., Intel SSE/AVX, GPU)



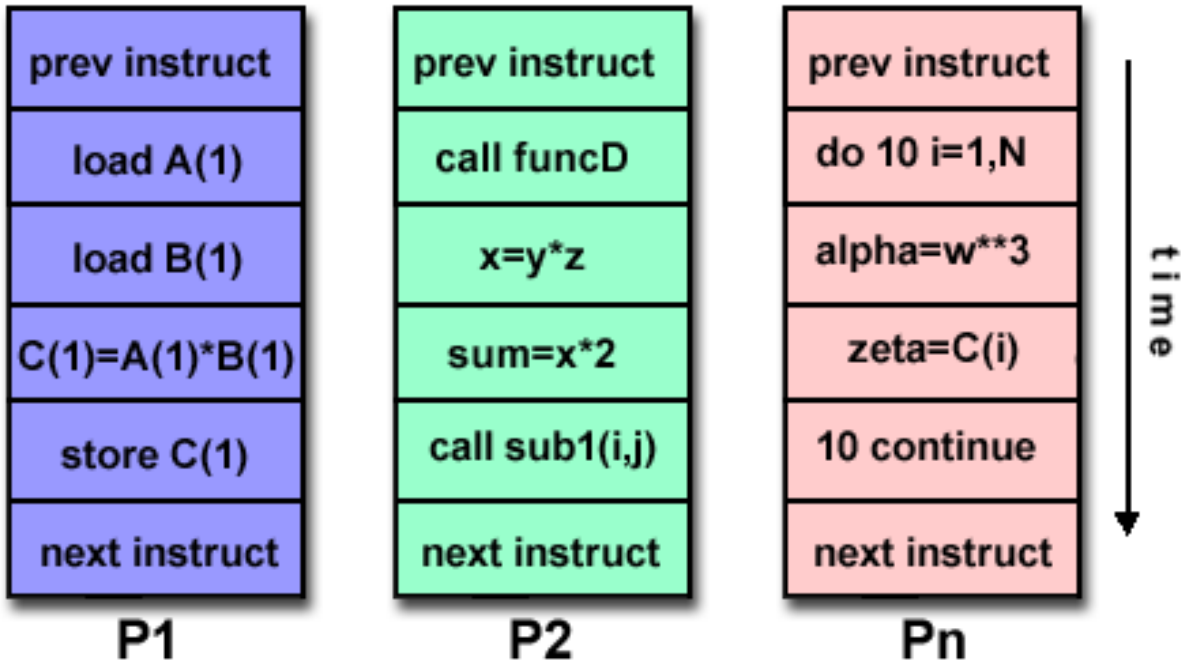
MIMD

MIMD

- Example: multi-core CPU

MIMD

- Example: multi-core CPU



Problem Partitioning

Problem Partitioning

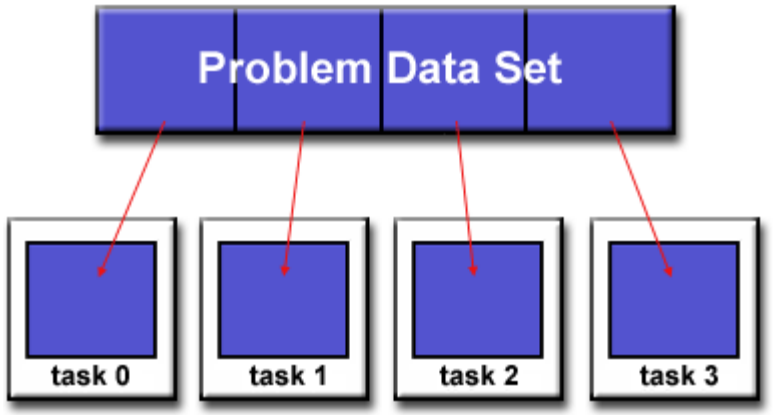
- Decomposition: Domain v. Functional

Problem Partitioning

- Decomposition: Domain v. Functional
- Domain Decomposition
  - SPMD
  - Input domain
  - Output Domain
  - Both

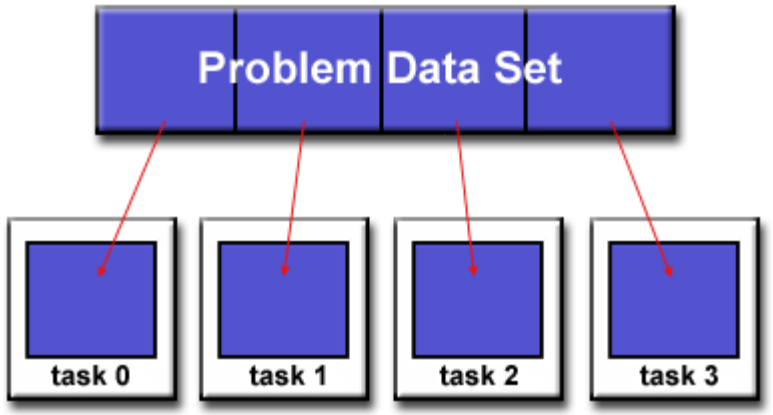
Problem Partitioning

- Decomposition: Domain v. Functional
- Domain Decomposition
  - SPMD
  - Input domain
  - Output Domain
  - Both



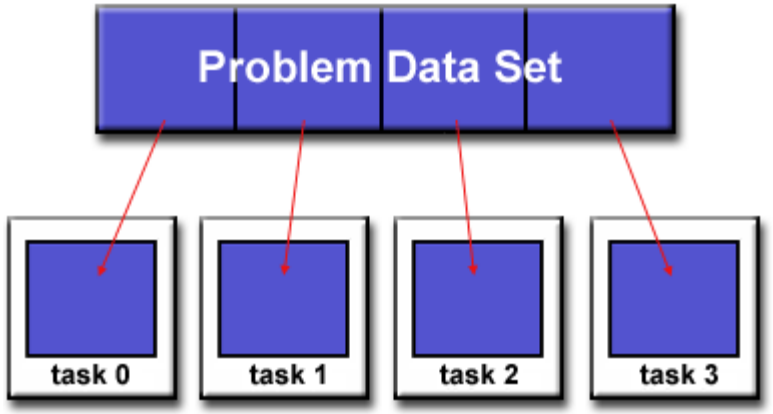
Problem Partitioning

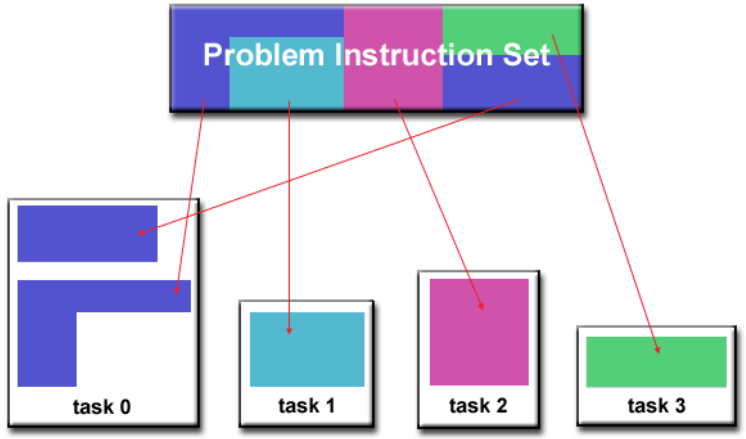
- Decomposition: Domain v. Functional
- Domain Decomposition
  - SPMD
  - Input domain
  - Output Domain
  - Both
- Functional Decomposition
  - MPMD
  - Independent Tasks
  - Pipelining



Problem Partitioning

- Decomposition: Domain v. Functional
- Domain Decomposition
  - SPMD
  - Input domain
  - Output Domain
  - Both
- Functional Decomposition
  - MPMD
  - Independent Tasks
  - Pipelining

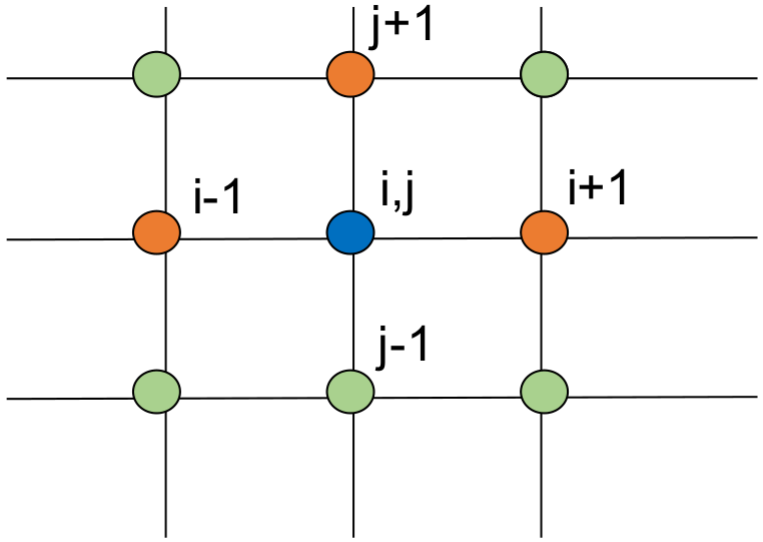




Game of Life

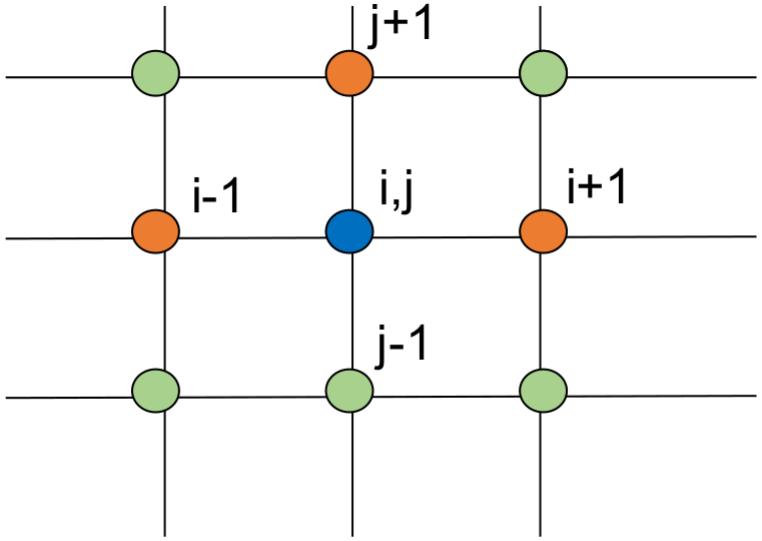
Game of Life

- Given a 2D Grid:
- $v_t(i,j) = F\left(v_{t-1}(of\ all\ its\ neighbors)\right)$

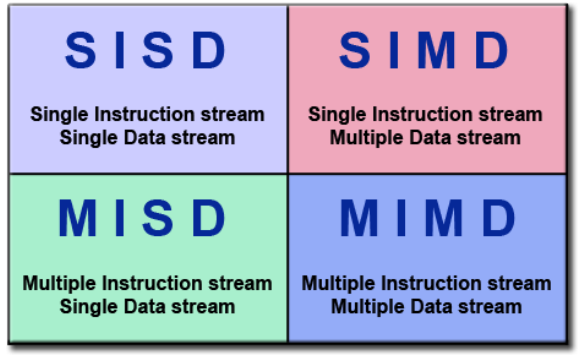


Game of Life

- Given a 2D Grid:
- $v_t(i,j) = F\left(v_{t-1}(of\ all\ its\ neighbors)\right)$



What model fits “best”?



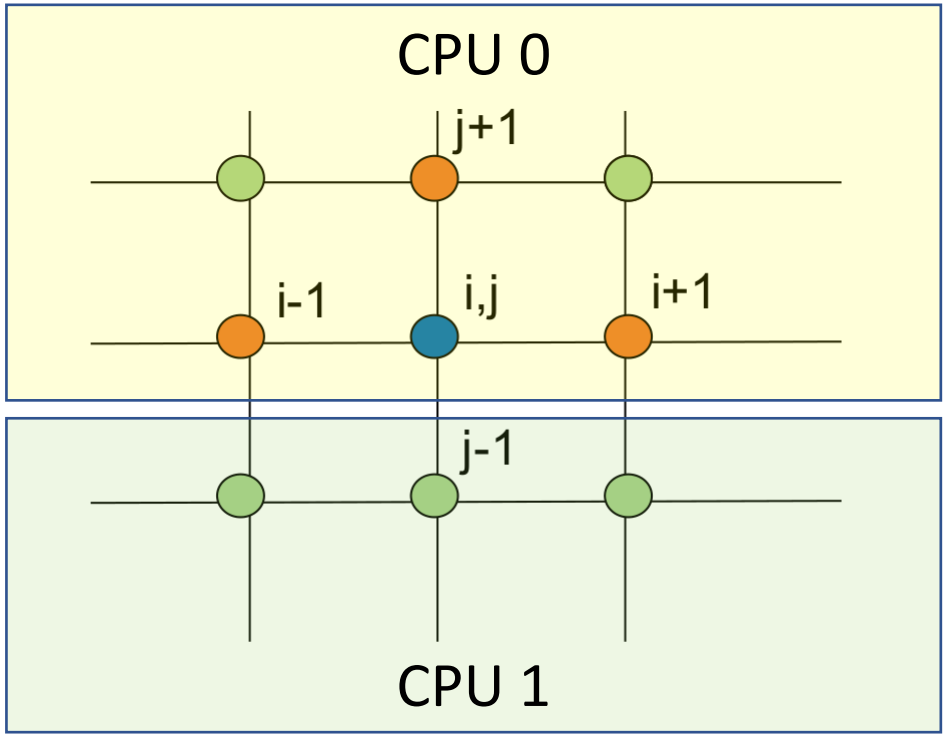
Domain decomposition

Domain decomposition

- Each CPU gets part of the input

Domain decomposition

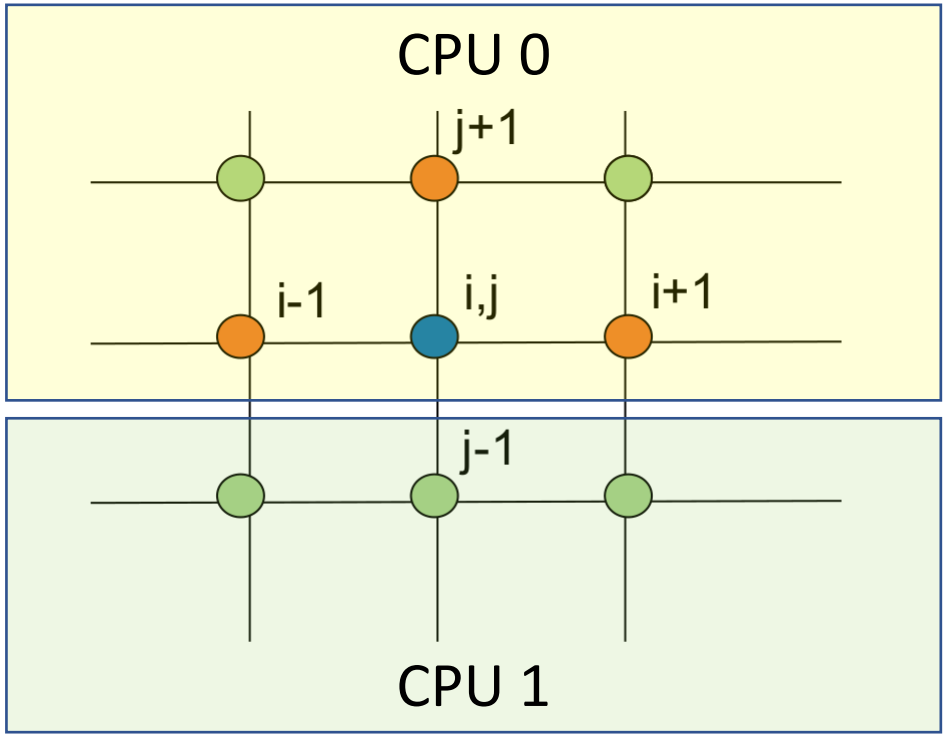
- Each CPU gets part of the input



Domain decomposition

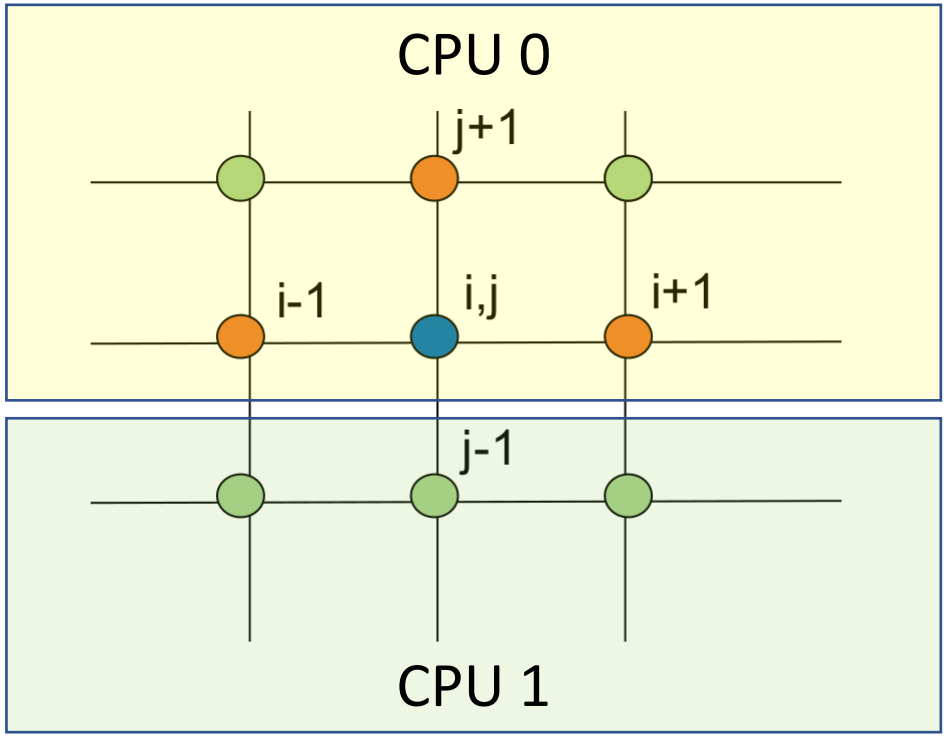
- Each CPU gets part of the input

Issues?



Domain decomposition

- Each CPU gets part of the input

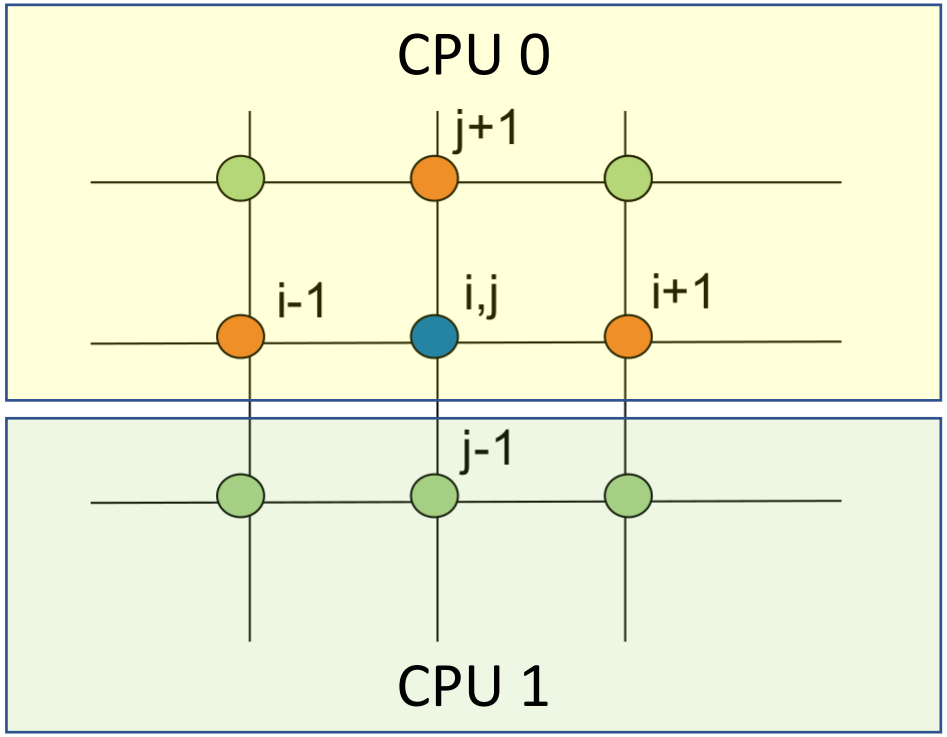


Issues?

- Accessing Data

Domain decomposition

- Each CPU gets part of the input

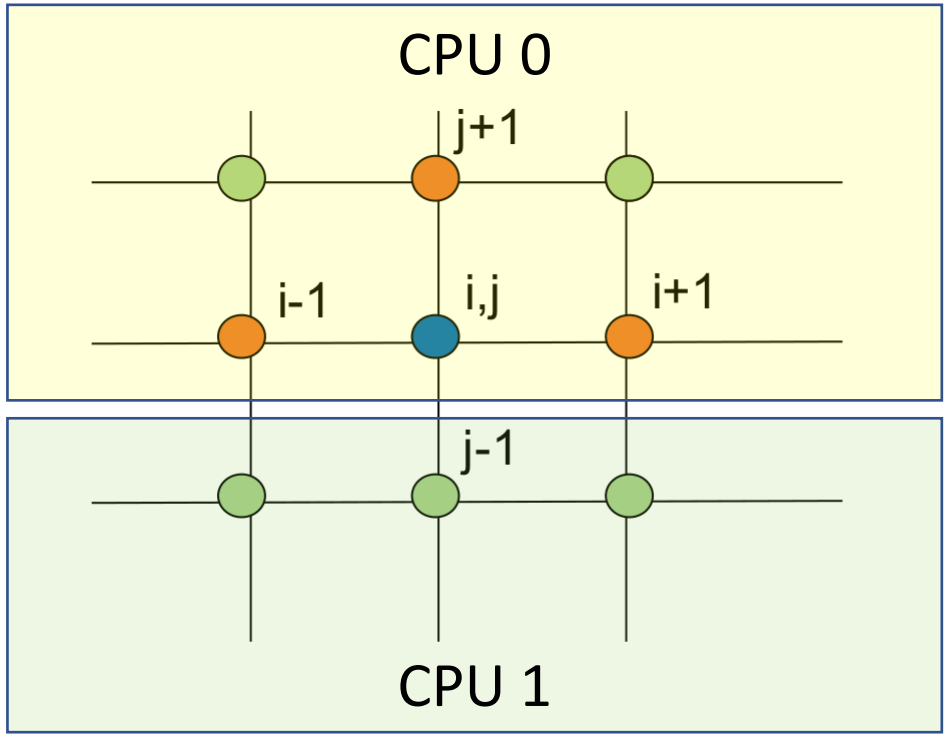


Issues?

- Accessing Data
  - Can we access v(i+1, j) from CPU 0

Domain decomposition

- Each CPU gets part of the input

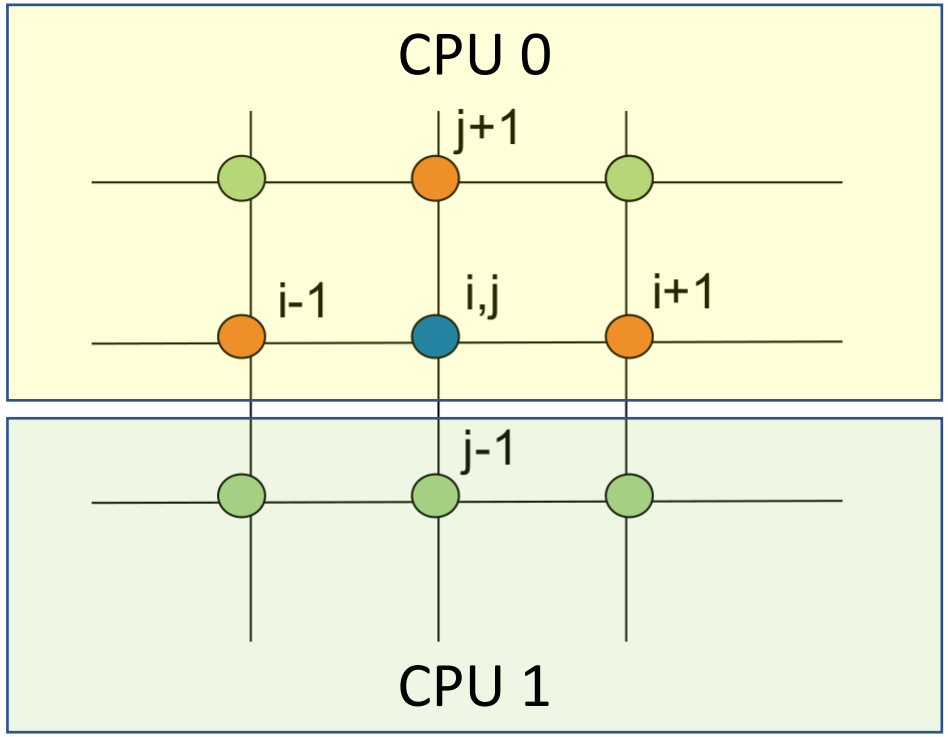


Issues?

- Accessing Data
  - Can we access v(i+1, j) from CPU 0
    - …as in a “normal” serial program?
    - Shared memory? Distributed?
  - Time to access v(i+1,j) == Time to access v(i-1,j) ?
  - *Scalability vs Latency*

Domain decomposition

- Each CPU gets part of the input

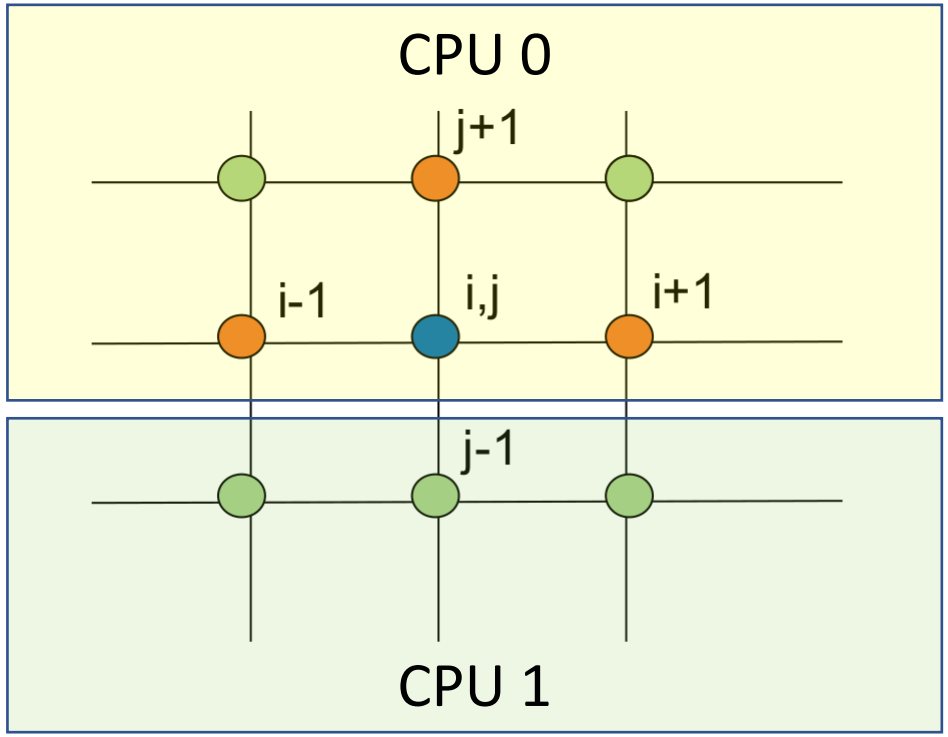


Issues?

- Accessing Data
  - Can we access v(i+1, j) from CPU 0
    - …as in a “normal” serial program?
    - Shared memory? Distributed?
  - Time to access v(i+1,j) == Time to access v(i-1,j) ?
  - *Scalability vs Latency*
- Control
  - Can we assign one vertex per CPU?
  - Can we assign one vertex per process/logical task?
  - *Task Management Overhead*

Domain decomposition

- Each CPU gets part of the input

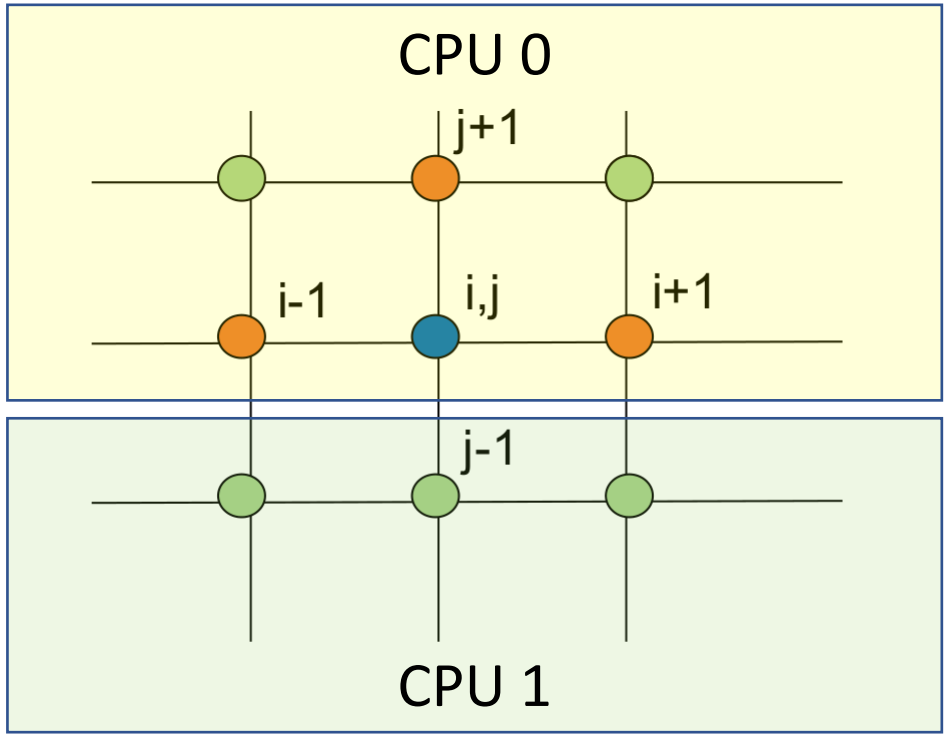


Issues?

- Accessing Data
  - Can we access v(i+1, j) from CPU 0
    - …as in a “normal” serial program?
    - Shared memory? Distributed?
  - Time to access v(i+1,j) == Time to access v(i-1,j) ?
  - *Scalability vs Latency*
- Control
  - Can we assign one vertex per CPU?
  - Can we assign one vertex per process/logical task?
  - *Task Management Overhead*
- *Load Balance*

Domain decomposition

- Each CPU gets part of the input

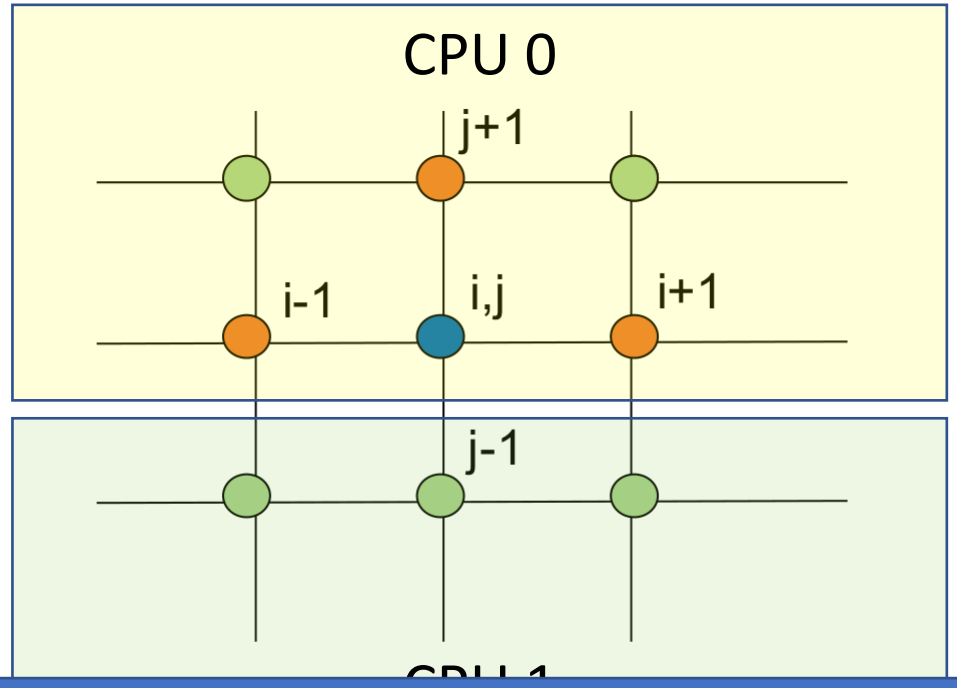


Issues?

- Accessing Data
  - Can we access v(i+1, j) from CPU 0
    - …as in a “normal” serial program?
    - Shared memory? Distributed?
  - Time to access v(i+1,j) == Time to access v(i-1,j) ?
  - *Scalability vs Latency*
- Control
  - Can we assign one vertex per CPU?
  - Can we assign one vertex per process/logical task?
  - *Task Management Overhead*
- *Load Balance*
- Correctness
  - order of reads and writes is non-deterministic
  - synchronization is required to enforce the order
  - *locks, semaphores, barriers, conditionals….*

Domain decomposition

- Each CPU gets part of the input



How could we do a functional decomposition?

Issues?

- Accessing Data
  - Can we access v(i+1, j) from CPU 0
    - …as in a “normal” serial program?
    - Shared memory? Distributed?
  - Time to access v(i+1,j) == Time to access v(i-1,j) ?
  - *Scalability vs Latency*
- Control
  - Can we assign one vertex per CPU?
  - Can we assign one vertex per process/logical task?
  - *Task Management Overhead*
- *Load Balance*
- Correctness
  - order of reads and writes is non-deterministic
  - synchronization is required to enforce the order
  - *locks, semaphores, barriers, conditionals….*

Lab #1

- Basic synchronization
- http://www.cs.utexas.edu/~rossbach/cs378/lab/lab0.html

- ***Start early!!!***

Questions?