Parallel Computing Architectures

Marcus Chow

Logistics

- Please sign up for a week to present
  - One presentation per group for now
- Emerson will present DVFS in Heterogeneous Embedded Systems next week
- As always, feel free to message or email me if you have any questions or concerns
- Let's start working!!!!

Today

- Provide you a helpful resource for some concepts, terminology, and history
- Accelerated workshop on various computer architectures
- Hope to peak your interests enough for you to start to ask questions

Parallel **Computing Architectures**

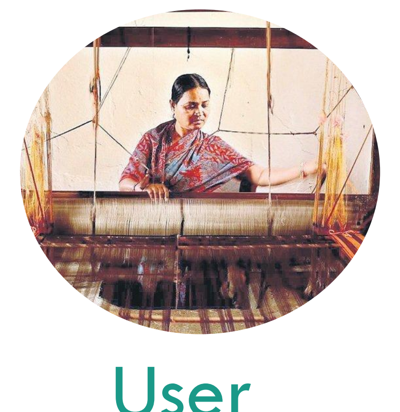
How is computing machine is design?

COMPUTING ARCHITECTURES

A Computing Architecture is the design of functionality, organization, and implementation of a computing system

Hardware

Software

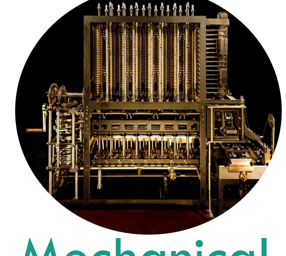


User

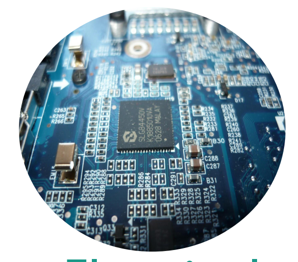
COMPUTING HARDWARE

Physical components of a computing system

Biological



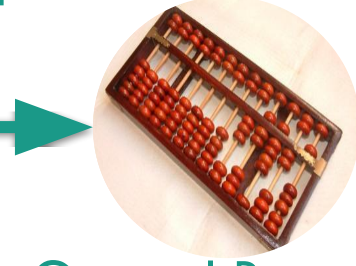
Mechanical



Electrical

Designed to solve some computational problem

Application Specific



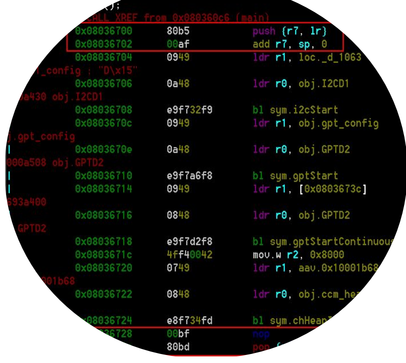
General Purpose

COMPUTING SOFTWARE

A set of instructions that directs a computer's hardware to perform a task

Biological

Mechanical

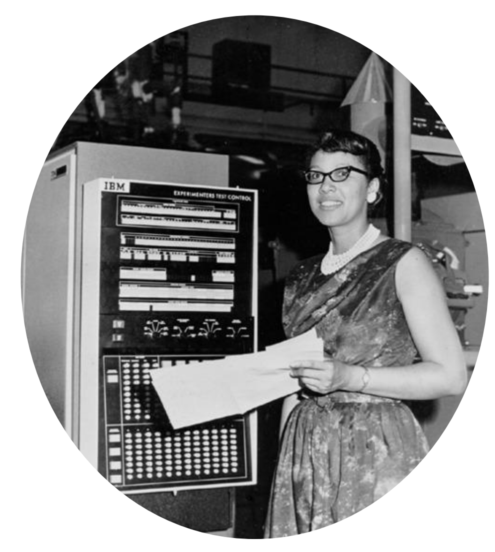


Electrical

THE USERS

The people who creates software and operates the hardware

Marget Hamiliton



Melba Roy Mouton

Dorothy Vaughan

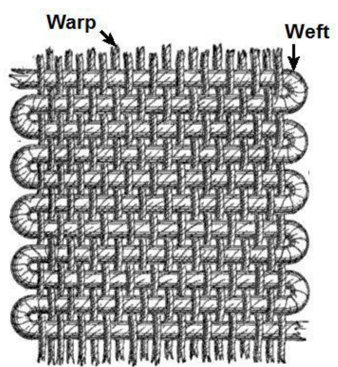
WEAVING AS CODE

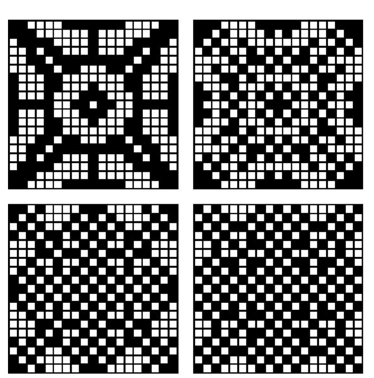
A warp thread is woven either above or belove the weft thread

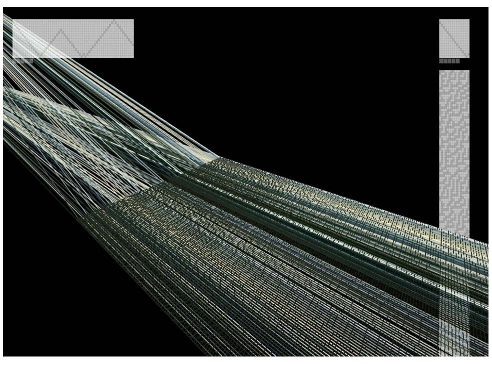
Can be visualized through binary representation (drawdown)

A drawdown pattern can be made to produce a non-repeating automata

It is possible to generate a Turing-complete program woven into the fabric itself







https://www2.cs.arizona.edu/patterns/weaving/webdocs/gre_dda1.pdf

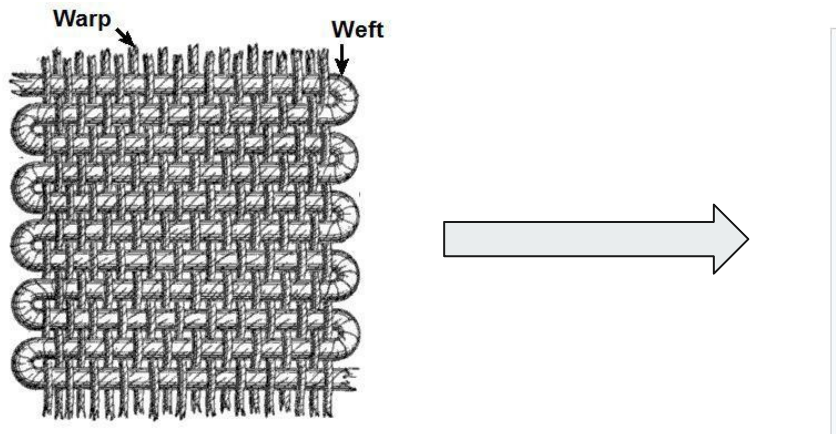
http://blog.blprnt.com/blog/blprnt/infinite-weft-exploring-the-old-aesthetic

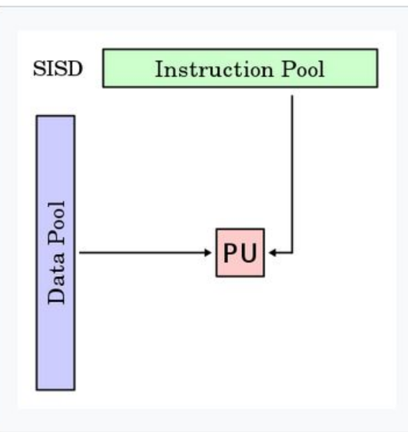
A History of Women's Role in Computing Architectures

https://www.youtube.com/watch?v=GuEyXXZA3Ms&t=18s

Threads as a streams of information

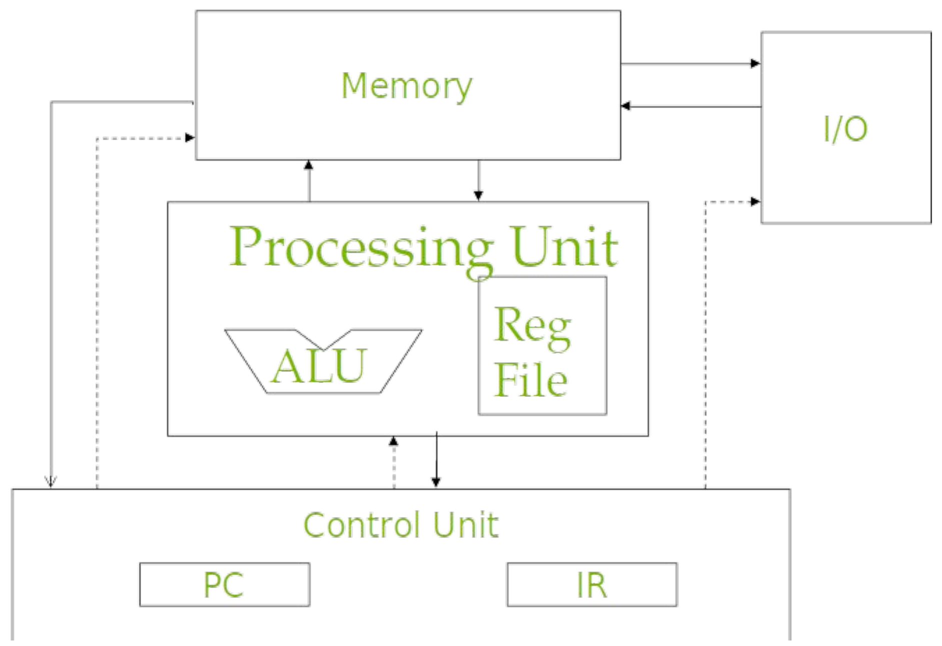
- Visualize two separate flows of information through a computer
- One for instructions (Thread) and the Data that is computed on
- Mixing the two streams produces a single output



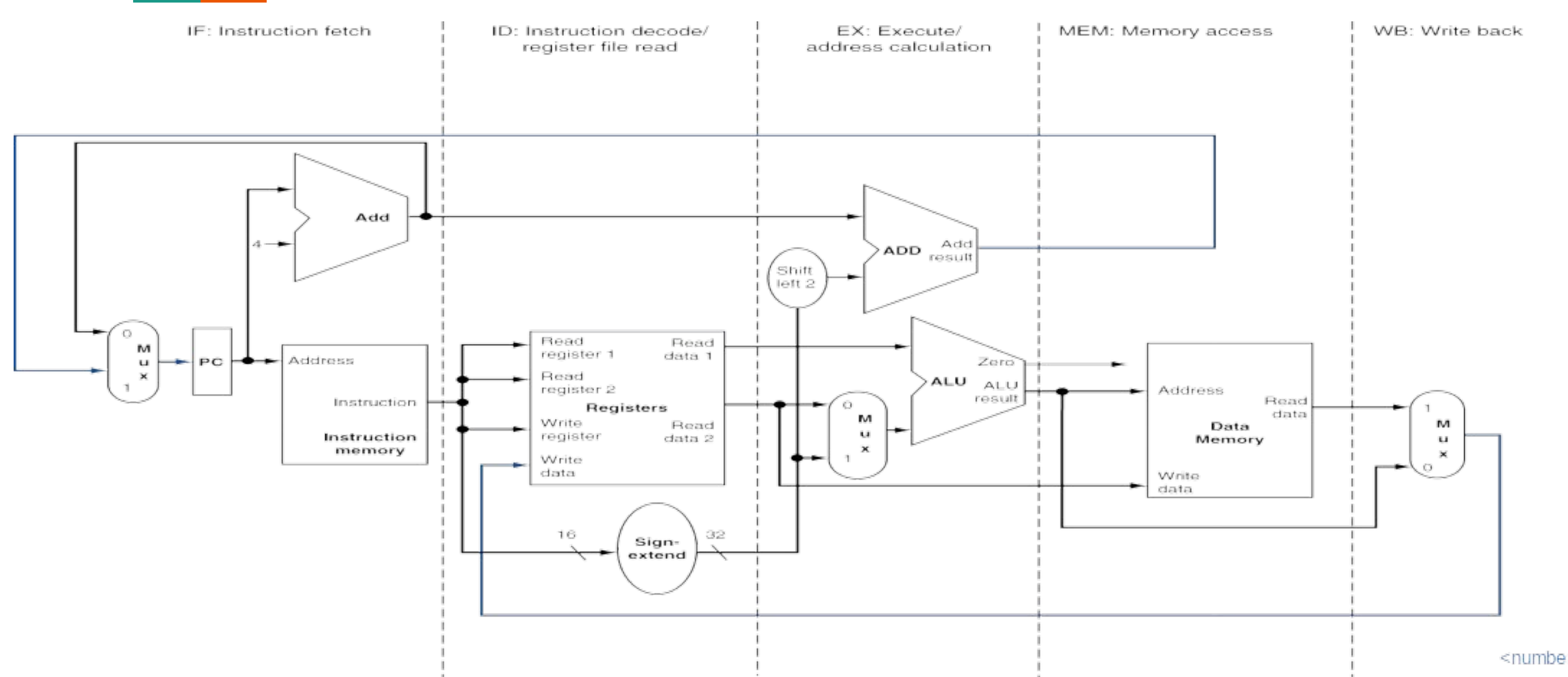


A thread as a Von Neumon processor

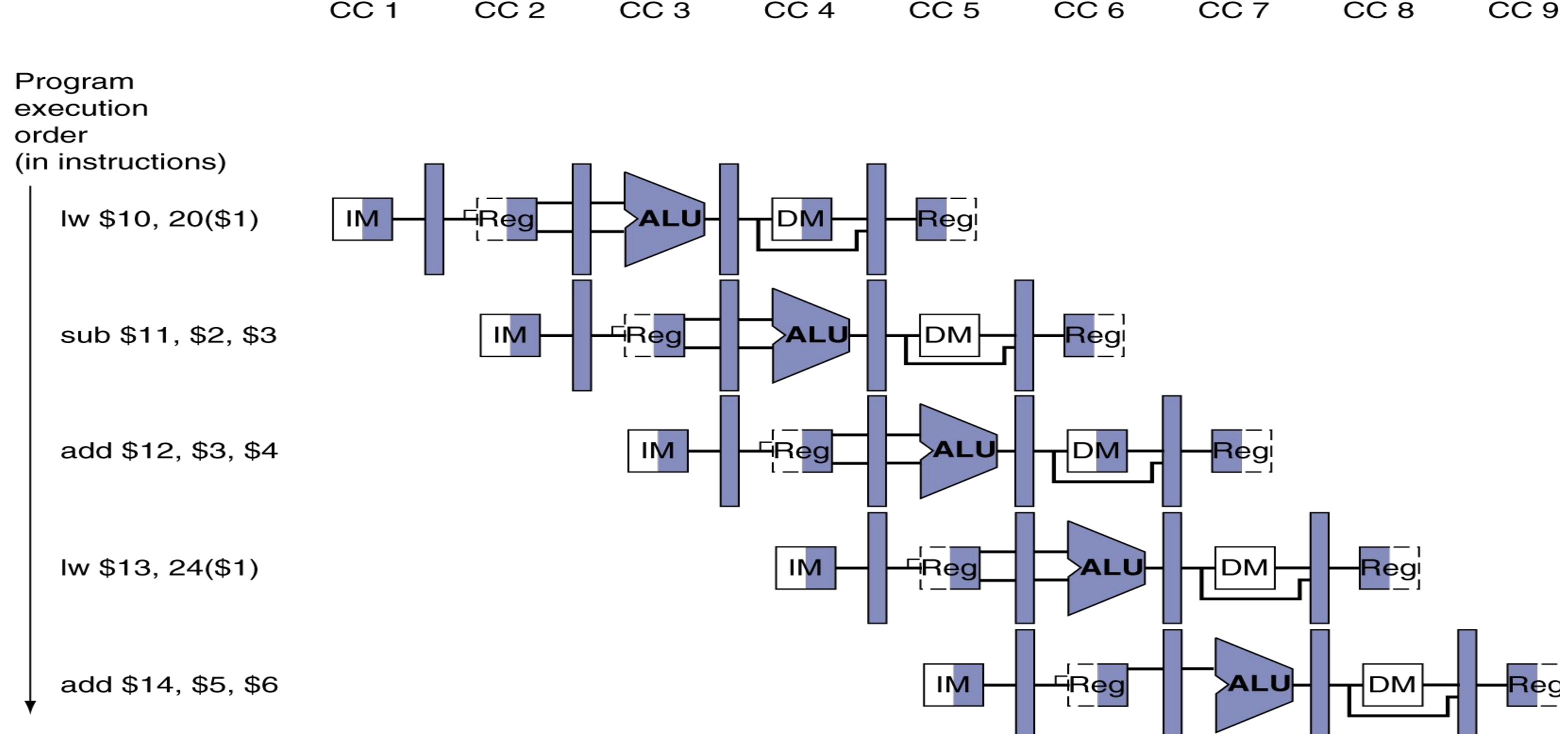
- Von-Neumann Processor is an abstract diagram of a computer
- Contains 4 units
  - Processing Unit
  - Control Unit
  - Memory
  - I/O



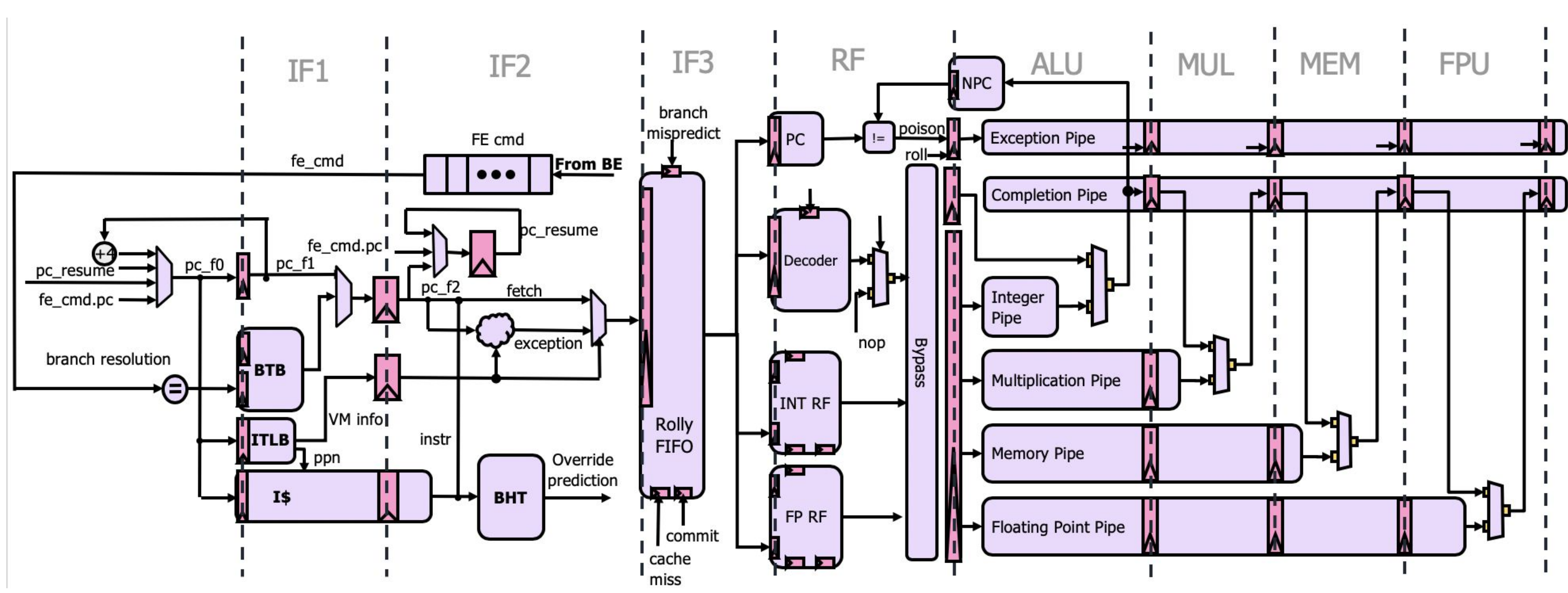
5-stage Compute pipeline



5-stage Compute pipeline

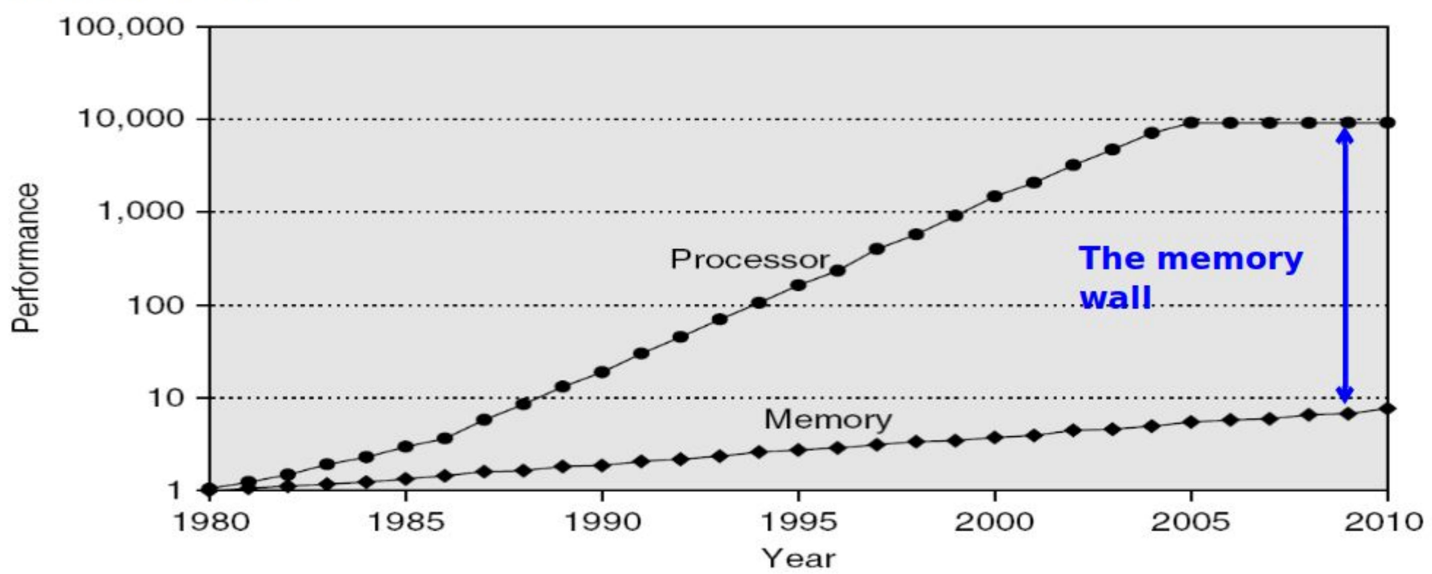


Increased Complexity

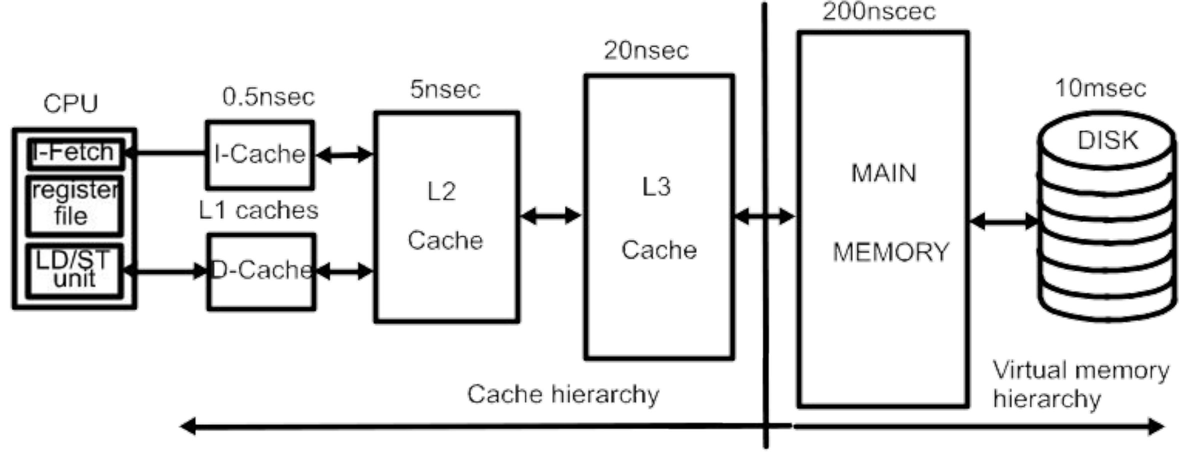


Memory Performance Gap

- Memory Performance Gap means the CPU is underutilized while it waits for data



Typical Memory Hierarchy



- Principle of locality:
- A program accesses a relatively small portion of the address space at a time
- Two different types of locality:
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon
  - Spatial locality: if an item is referenced, items whose addresses are close tend to be referenced soon

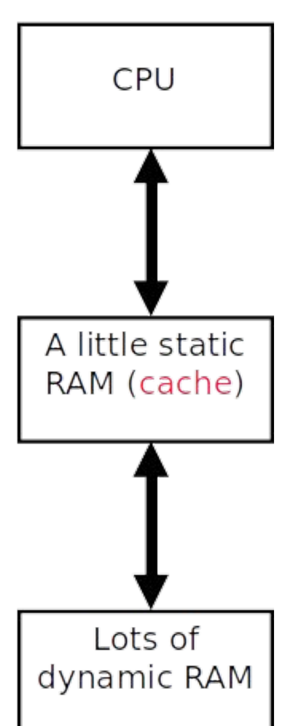
Principle of Locality

- Why does the hierarchy work?

- Because most programs exhibit *locality*, which the cache can take advantage of.
  - The principle of temporal locality says that if a program accesses one memory address, there is a good chance that it will access the same address again.
  - The principle of spatial locality says that if a program accesses one memory address, there is a good chance that it will also access other nearby addresses.

Principle of Locality

- First time the processor reads from an address in main memory, a copy of that data is also stored in the cache.
  - The next time that same address is read, we can use the copy of the data in the cache *instead* of accessing the slower dynamic memory.
  - So the first read is a little slower than before since it goes through both main memory and the cache, but subsequent reads are much faster.
- This takes advantage of temporal locality—commonly accessed data is stored in the faster cache memory.
- By storing a block (multiple words) we also take advantage of spatial locality



Temporal locality in instructions

- Loops are excellent examples of temporal locality in programs.
  - The loop body will be executed many times.
  - The computer will need to access those same few locations of the instruction memory repeatedly.
- For example:
  - Each instruction will be fetched over and over again, once on every loop iteration.

```
Loop:   lw      $t0, 0($s1)
        add     $t0, $t0, $s2
        sw      $t0, 0($s1)
        addi    $s1, $s1, -4
        bne     $s1, $0, Loop
```

Temporal locality in Data

- Programs often access the same variables over and over, especially within loops. Below, sum and i are repeatedly read and written.
- Commonly-accessed variables can sometimes be kept in registers, but this is not always possible.
  - There are a limited number of registers.
  - There are situations where the data must be kept in memory, as is the case with shared or dynamically-allocated memory.

```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + f(i);
```

Spatial locality in Instructions

- Nearly every program exhibits spatial locality, because instructions are usually executed in sequence — if we execute an instruction at memory location *i*, then we will probably also execute the next instruction, at memory location *i+1*.
- Code fragments such as loops exhibit *both* temporal and spatial locality.

```
sub  $sp, $sp, 16
sw   $ra, 0($sp)
sw   $s0, 4($sp)
sw   $a0, 8($sp)
sw   $a1, 12($sp)
```

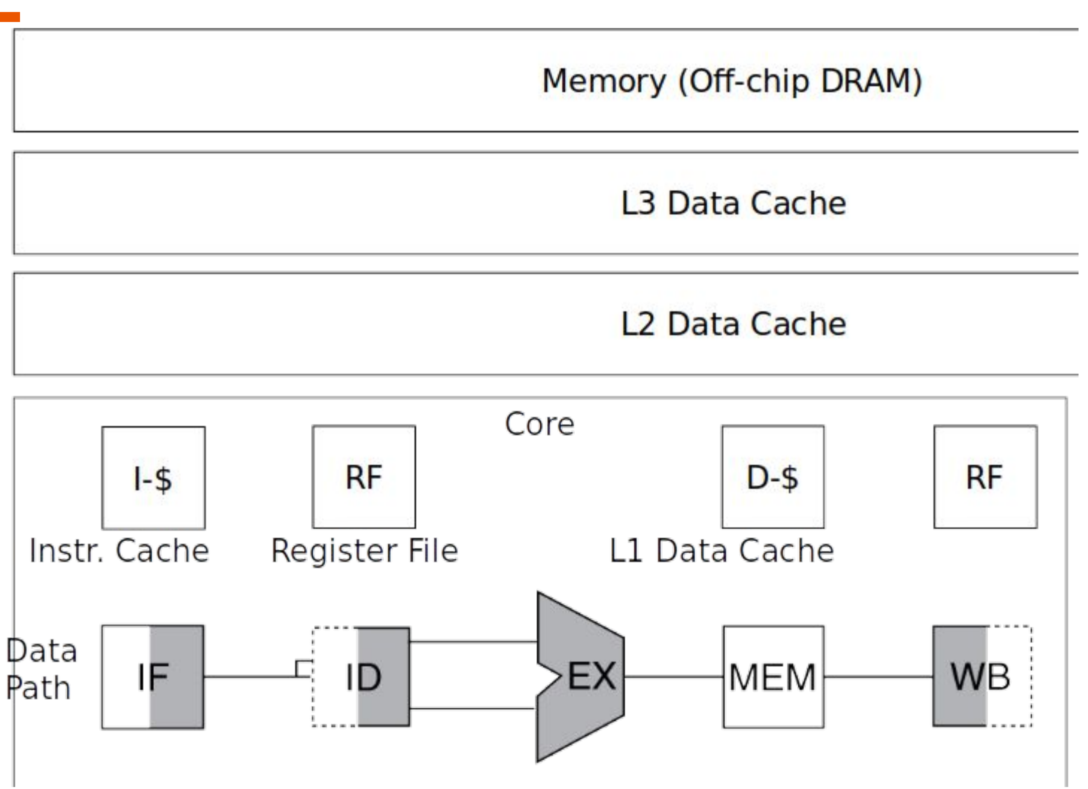
Spatial locality in Data

- Programs often access data that is stored contiguously.
  - Arrays, like a in the code on the top, are stored in memory contiguously.
  - The individual fields of a record or object like employee are also kept contiguously in memory.

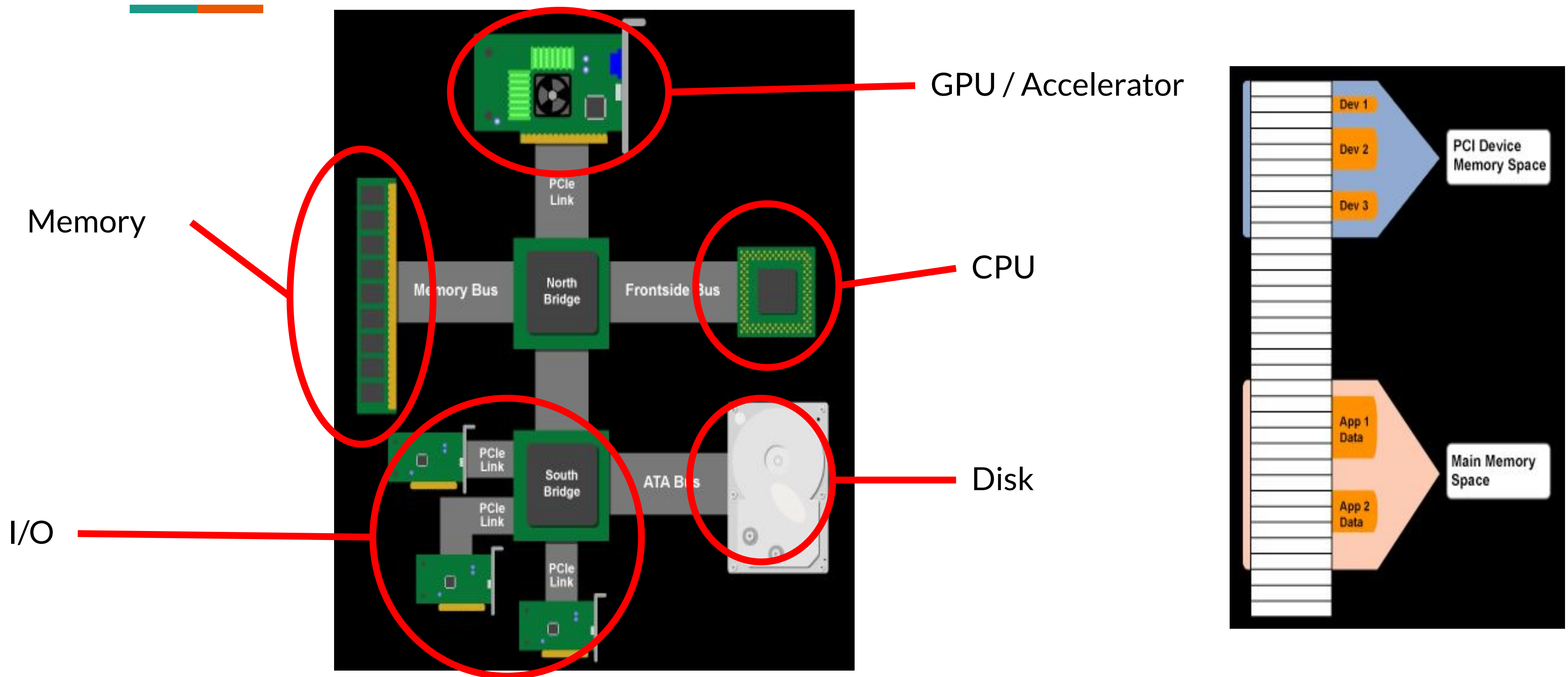
```
sum = 0;
for (i = 0; i < MAX; i++)
    sum = sum + a[i];
```

```
employee.name = "Homer Simpson";
employee.boss = "Mr. Burns";
employee.age = 45;
```

Abstracted CPU



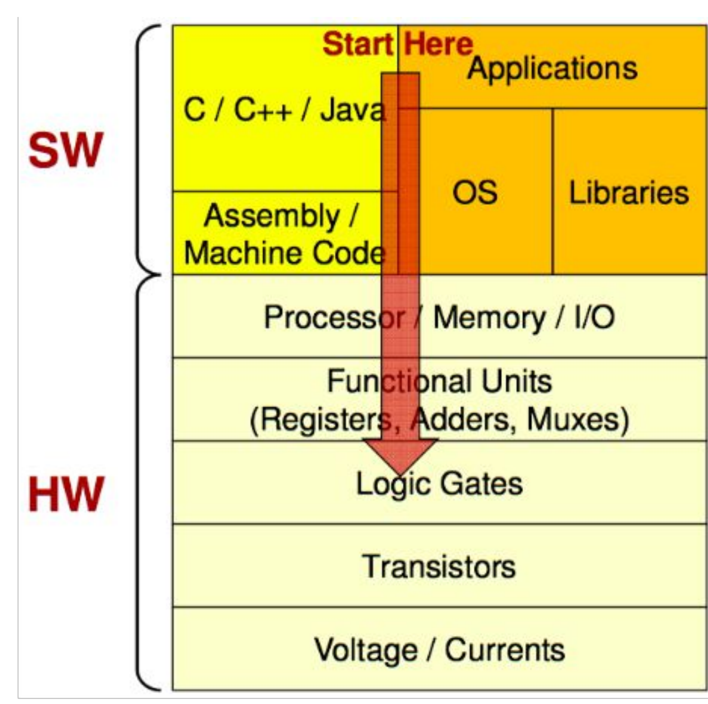
CPU in a Computer System



Software-Hardware Stack

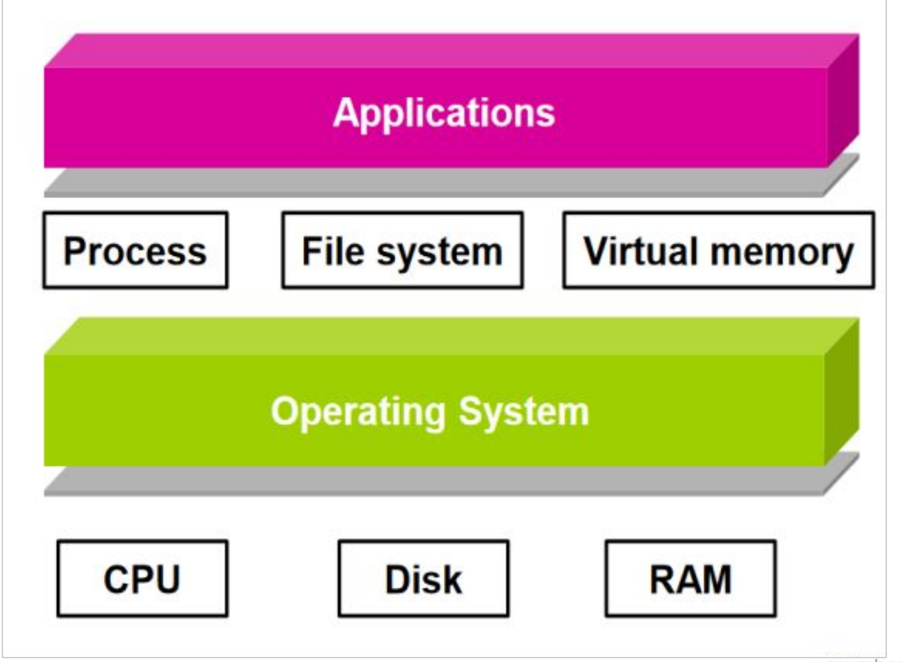
- Computer Architecture spans the connection between hardware and software
- Efficiency through understanding how each level interacts
- Hardware design informs software design and vice versa

Credit:: Mark Redekopp, USC



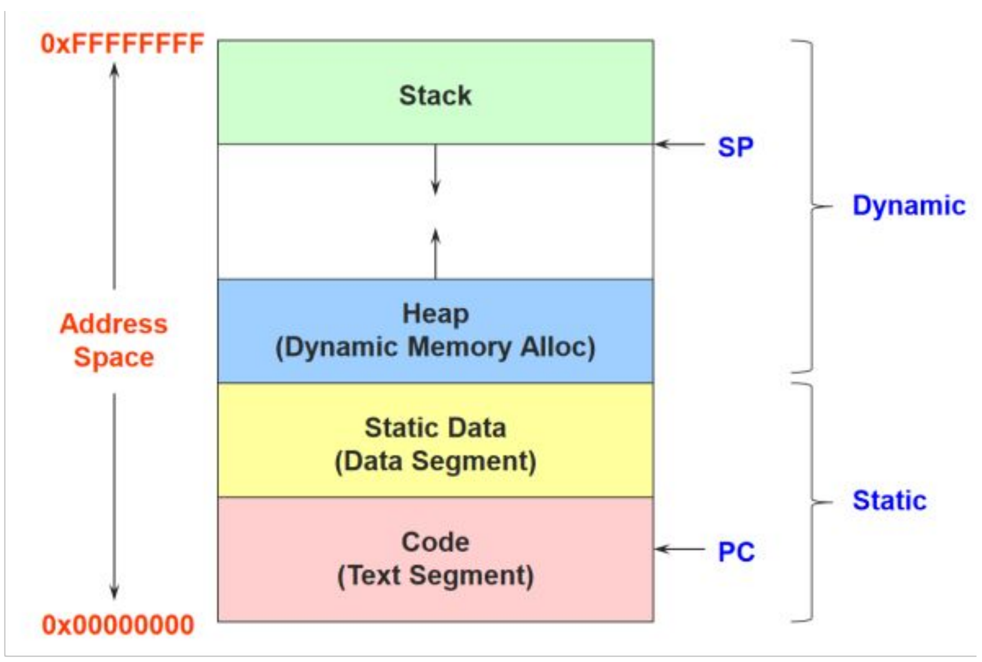
The Process in Operating Systems

- The Process is the OS abstraction for execution
  - Unit of execution
  - Unit of scheduling
- It is a program in execution
- This can introduce overheads when handling many processes



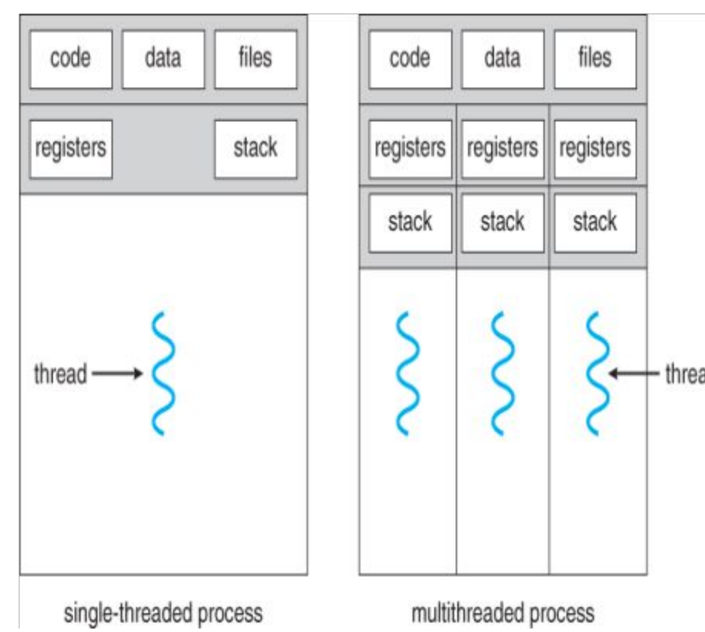
The Process in Operating Systems

- It contains all the state for a program in execution
- An Address space containing:
  - Static Memory
    - Code and input data for the program
  - Dynamic Memory
    - Allocated memory
    - Execution stack
- A set of control and general purpose registers



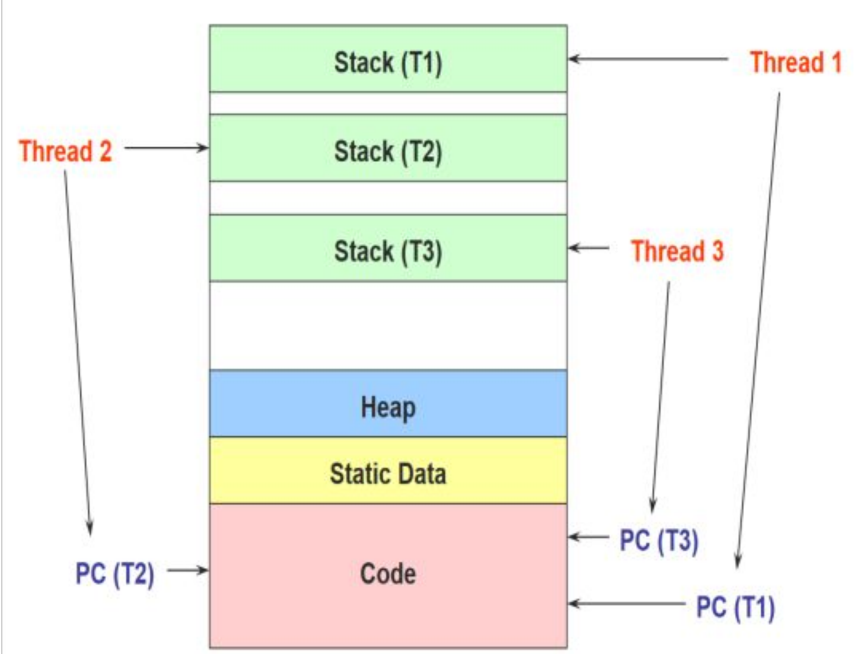
Process vs Thread

- Separate execution and resource container roles
  - The thread defines a sequential execution stream within a process (PC, SP, registers)
  - The process defines the address space, resources, and general process attributes (everything but threads)
- Threads become the unit of scheduling
  - Processes are now the containers in which threads execute
  - Processes become static, threads are the dynamic entities



Process vs Thread

- Separating threads and processes makes it easier to support multithreaded applications
- Concurrency (multithreading) can be very useful
  - Improving program structure
  - Handling concurrent events (e.g., Web requests)
  - Writing parallel programs

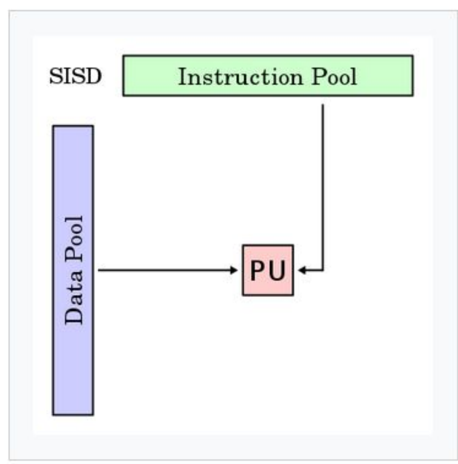


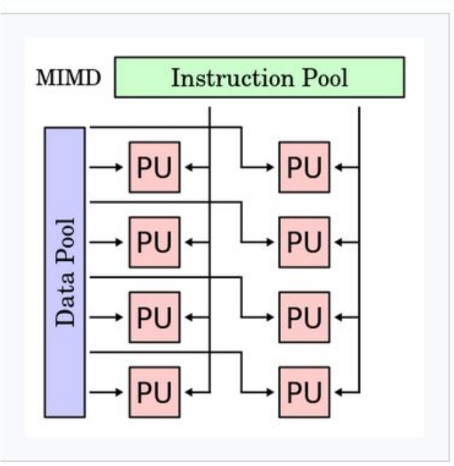
Parallel Computing Architectures

How do we process more data at once?

Multicore Designs

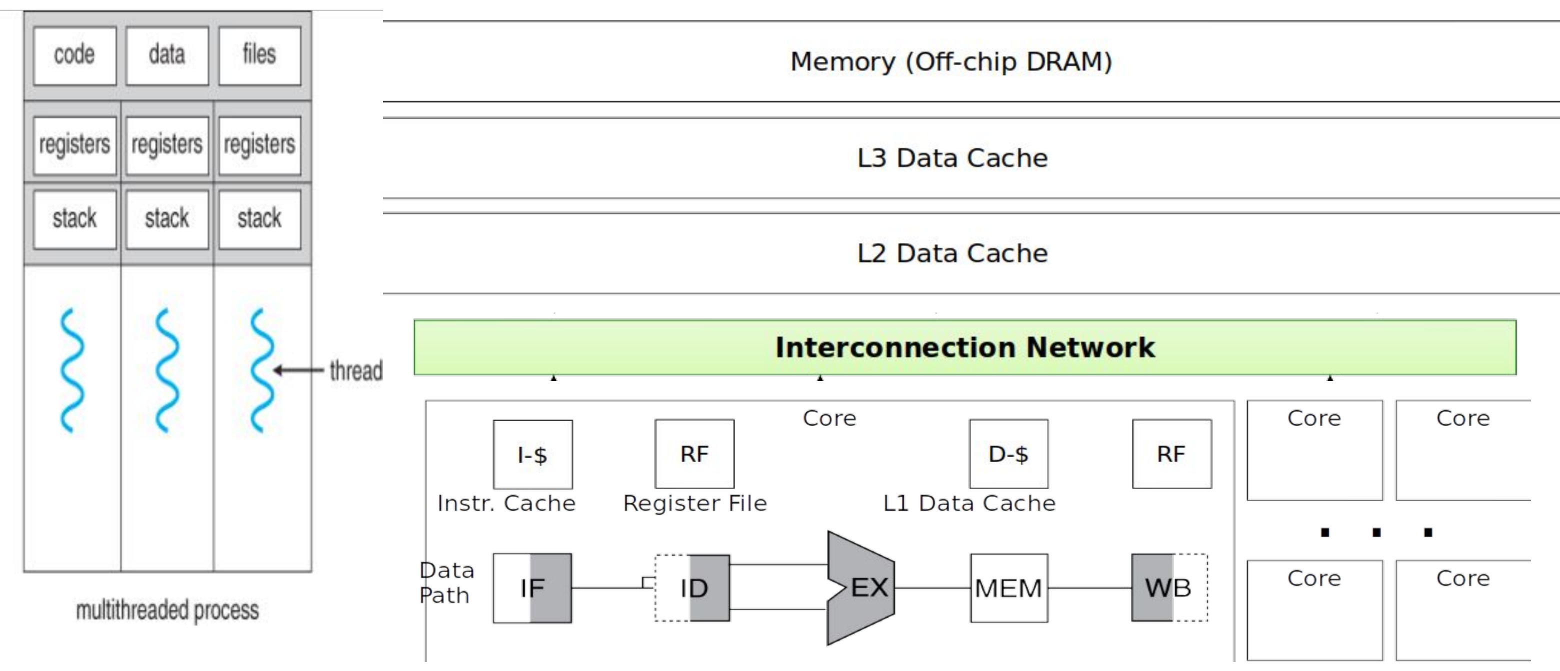
- Gain parallelism by adding addition cores
- Each core is independent of one another
- Multiple Instruction Multiple Data
- How do you connect each core together?





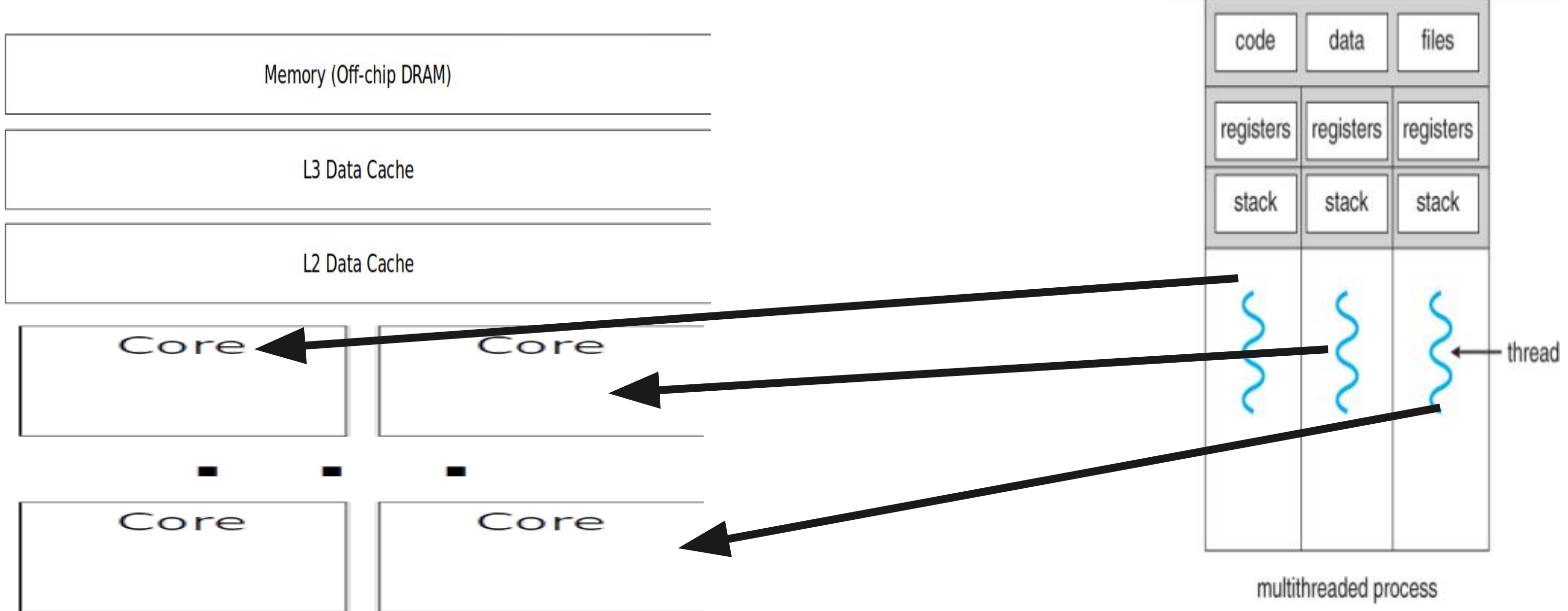
Interconnection Network

- Independent Cores communicate through higher level caches (L2 and L3)
- Are connected through bus system interconnect



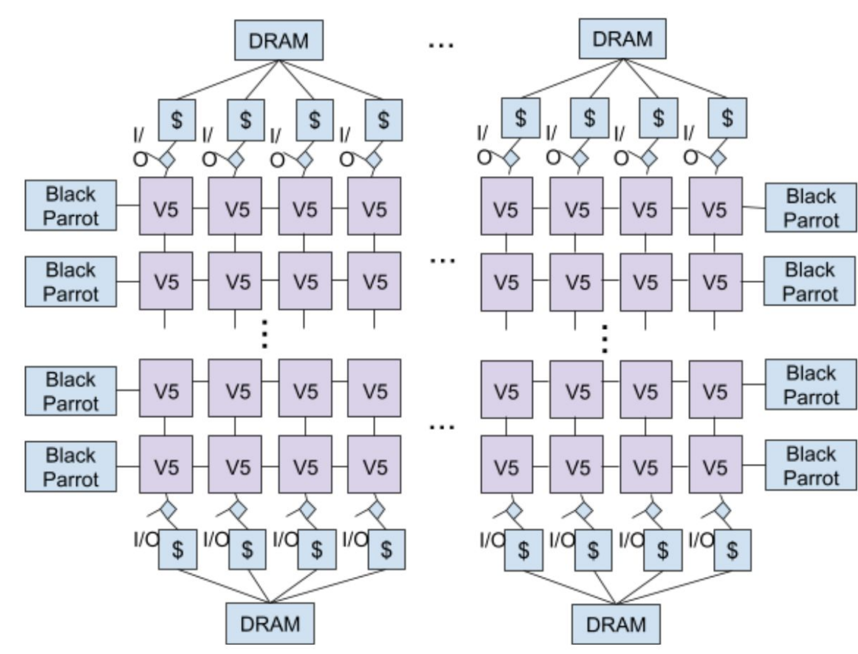
Utilizing multiple cores

- Multithreaded process can map a thread to a single core
- Can have multiple process run on different cores



Tiled Mesh Networks

- Can connect cores through a Mesh Network
- Each tile is very tiny
- Each tile routes messages to its neighbors
- Hammerblade Manycore project
- How to efficiently write parallel algorithms for it?
- How can we utilize the fact that each tile is independent of one another?

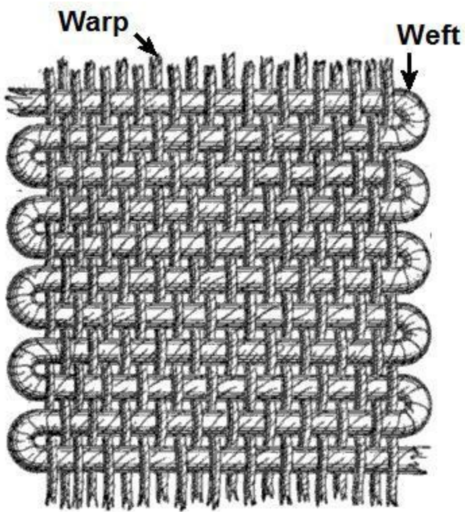


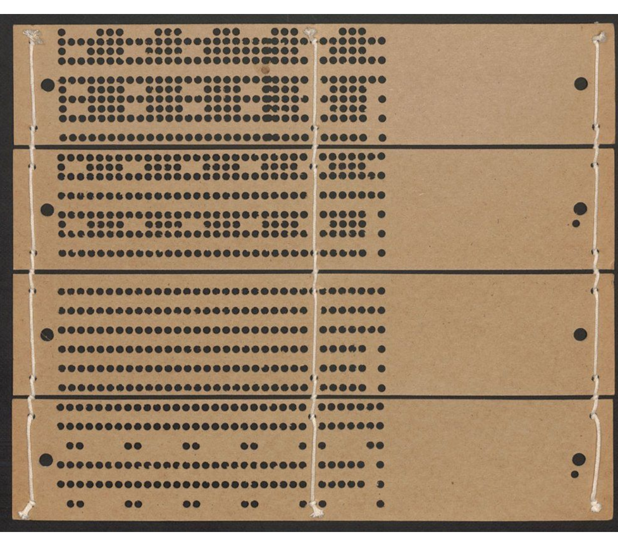
Parallel Other than MIMD

- What if we don't need to have independent cores?
- How can we still increase parallelization within the hardware?

Vectorization

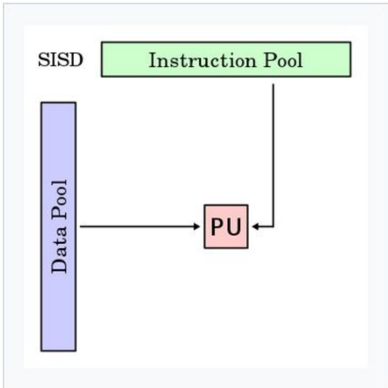
- We want to be able process more data
- Add more physical compute units
- this is how a loom works

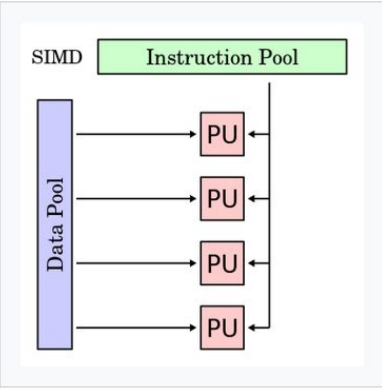




Back to Theory

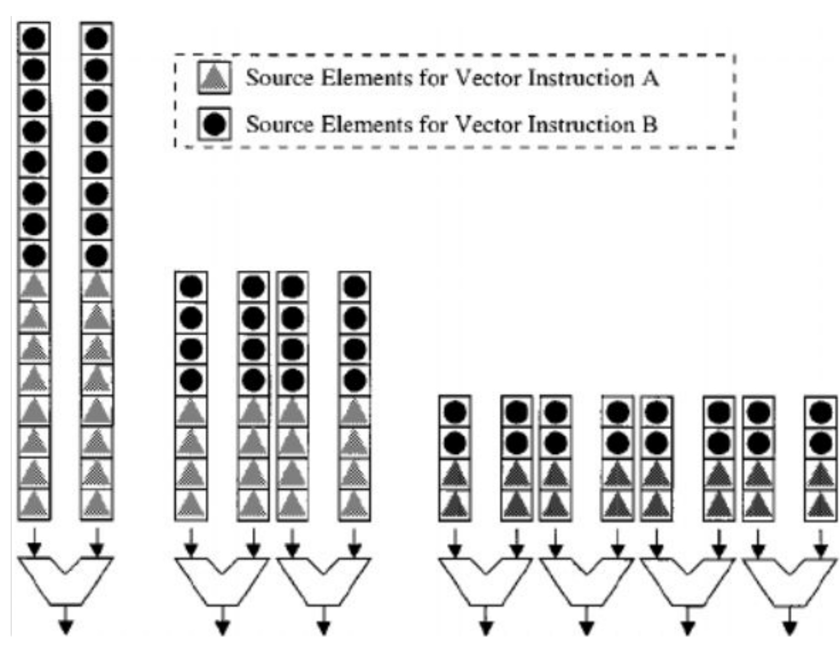
- Single Data -> Multiple Data





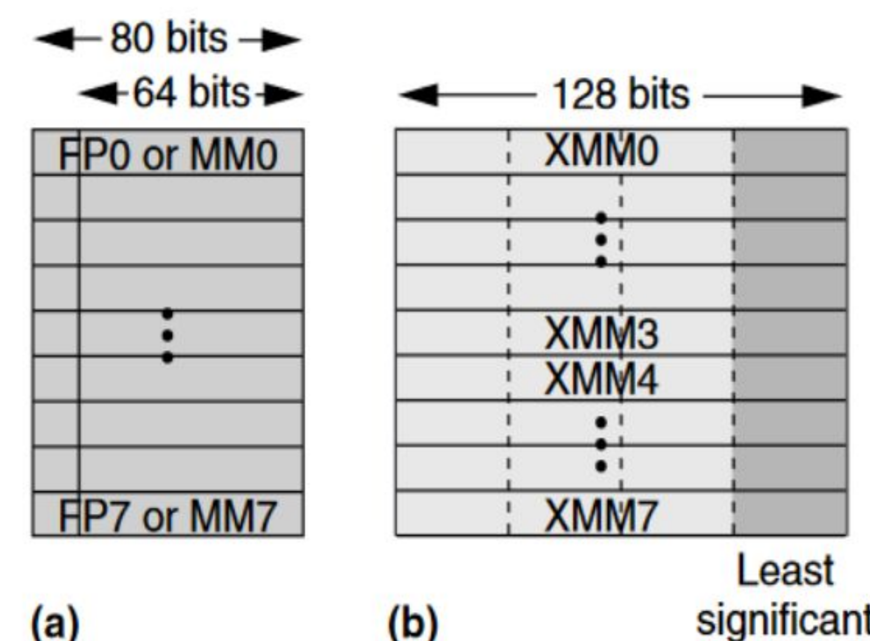
Vector Processors

- Increase the number of ALU hardware units
- Increasing the number of datapaths per functional unit reduces the execution time, as more element operations can be processed concurrently.



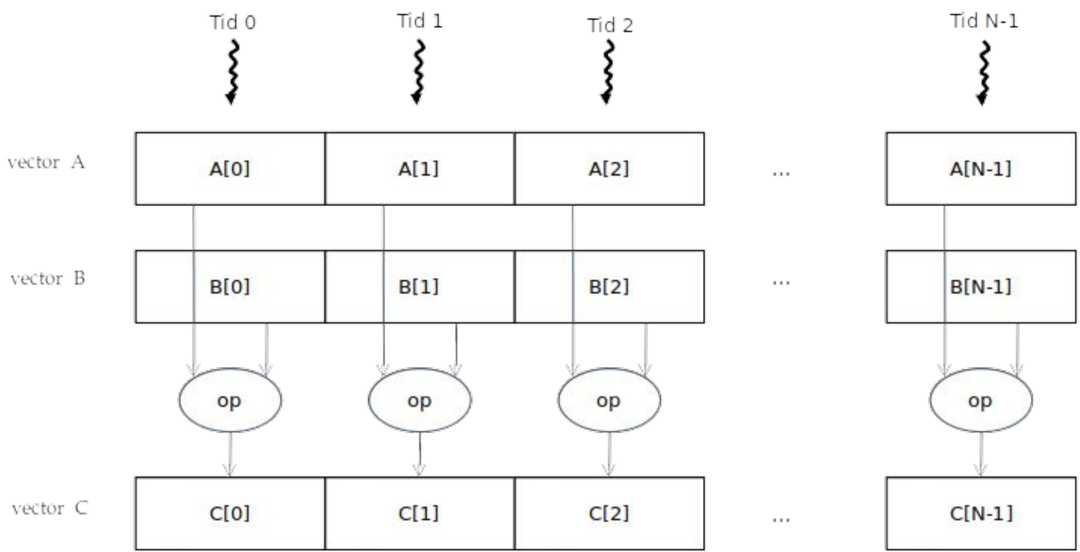
Vector Processors

- Increase size (width) of registers
- Increasing number of datapaths requires more data to be accessed within a single cycle



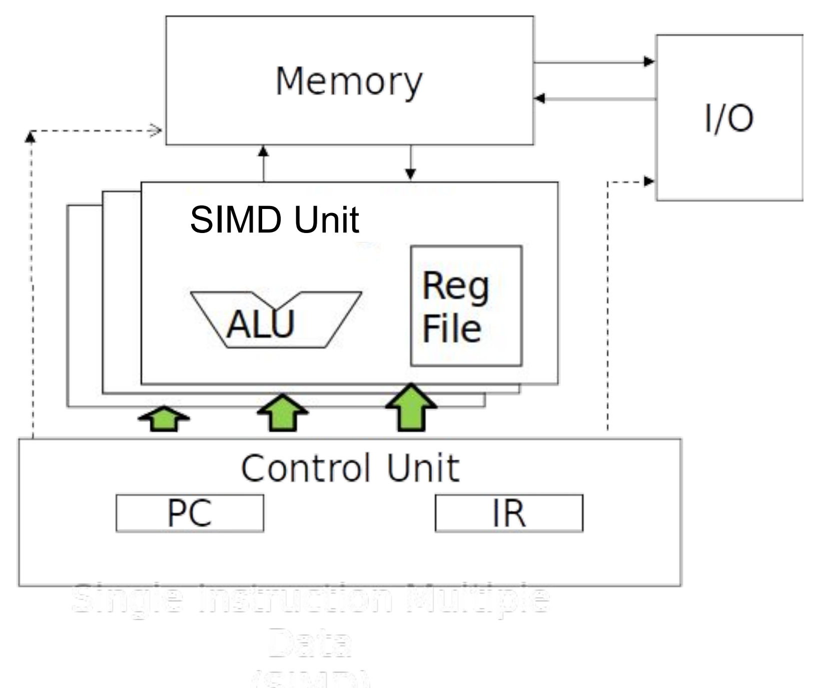
Vector Operations

- Assign a thread to each element of an array
- One thread operates on a single element

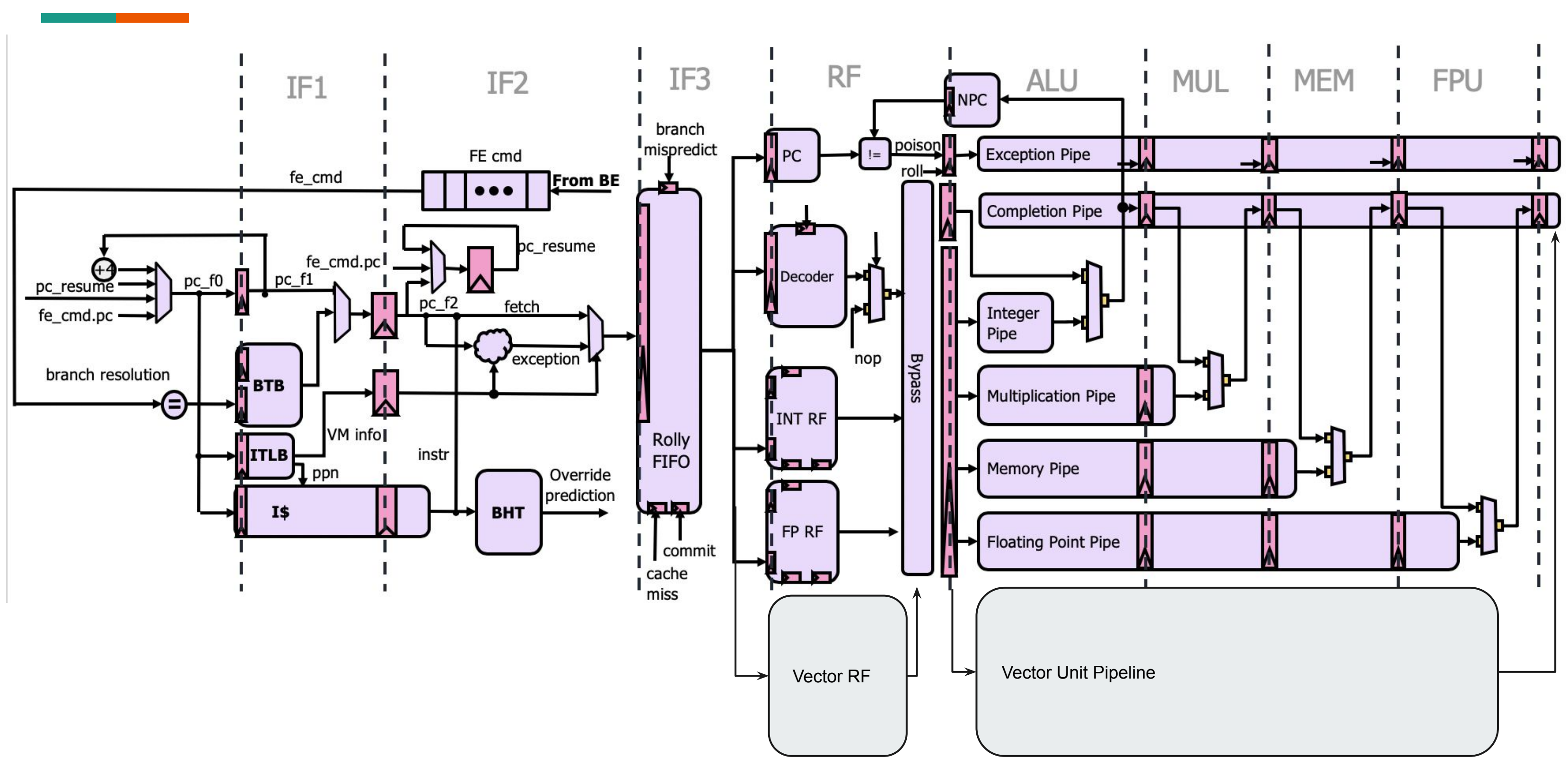


Von-Neumann Model with SIMD units

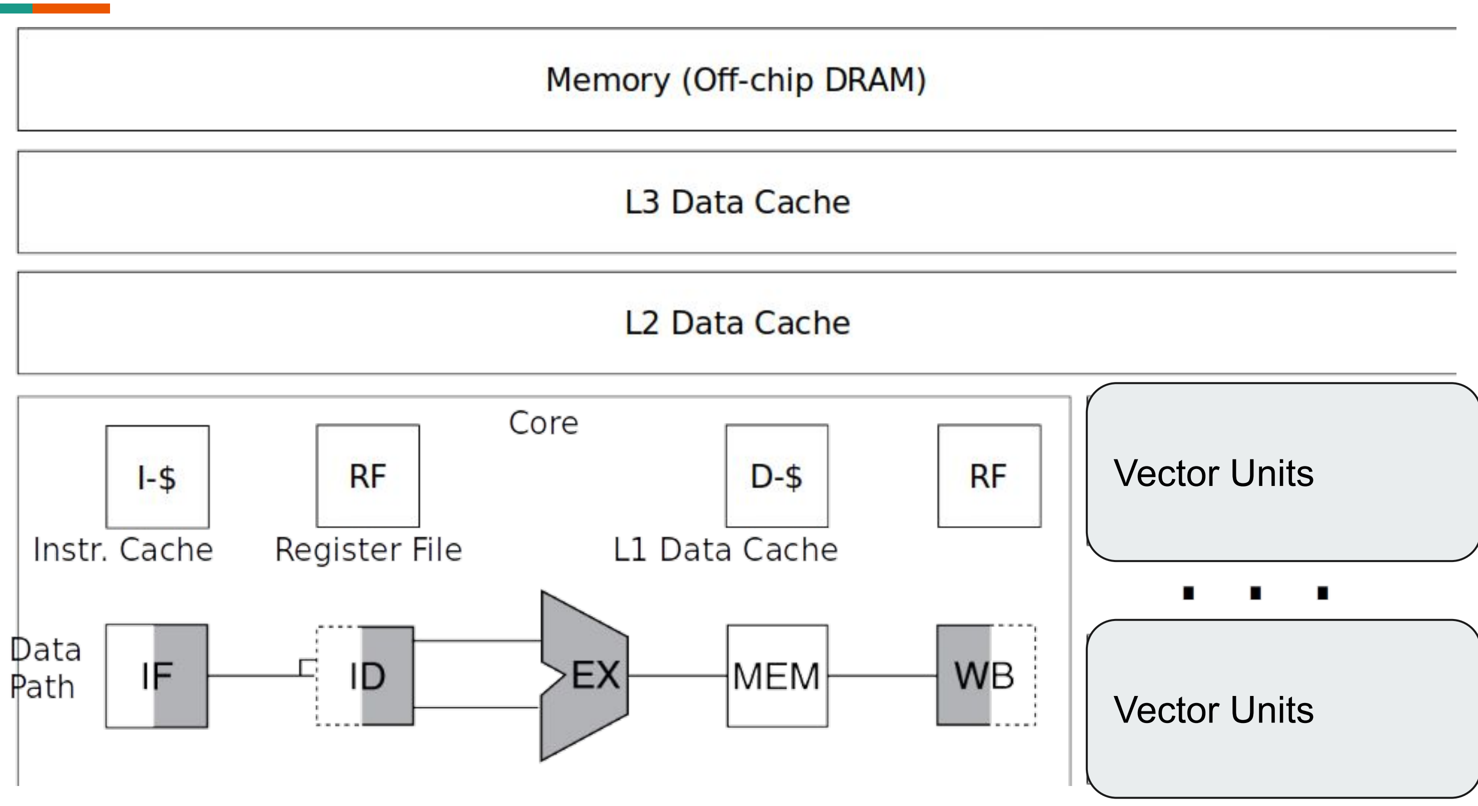
– Duplicate Processing Unit to support vector instructions
– Still share a single control unit



Pipeline with SIMD Units

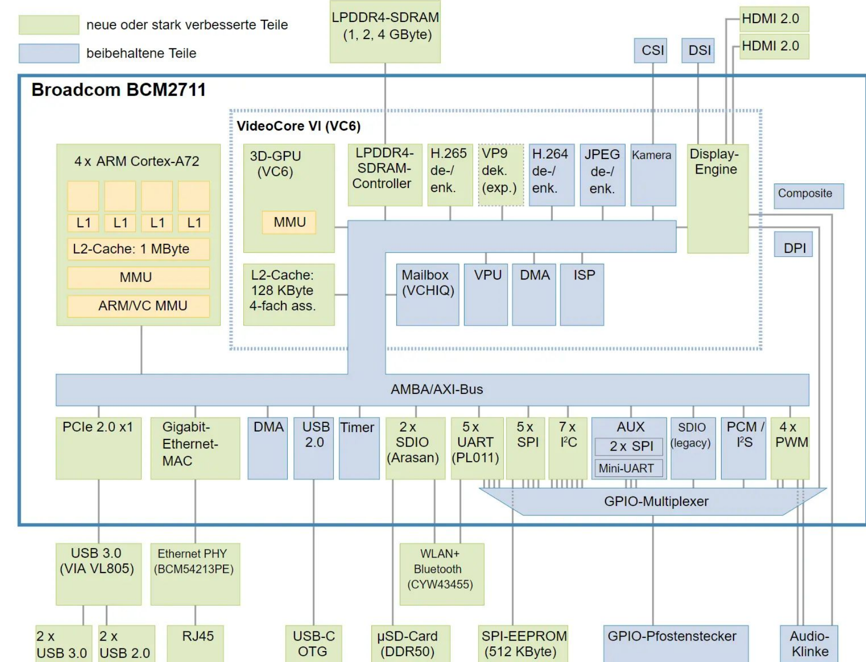


Abstracted CPU with Vector Units



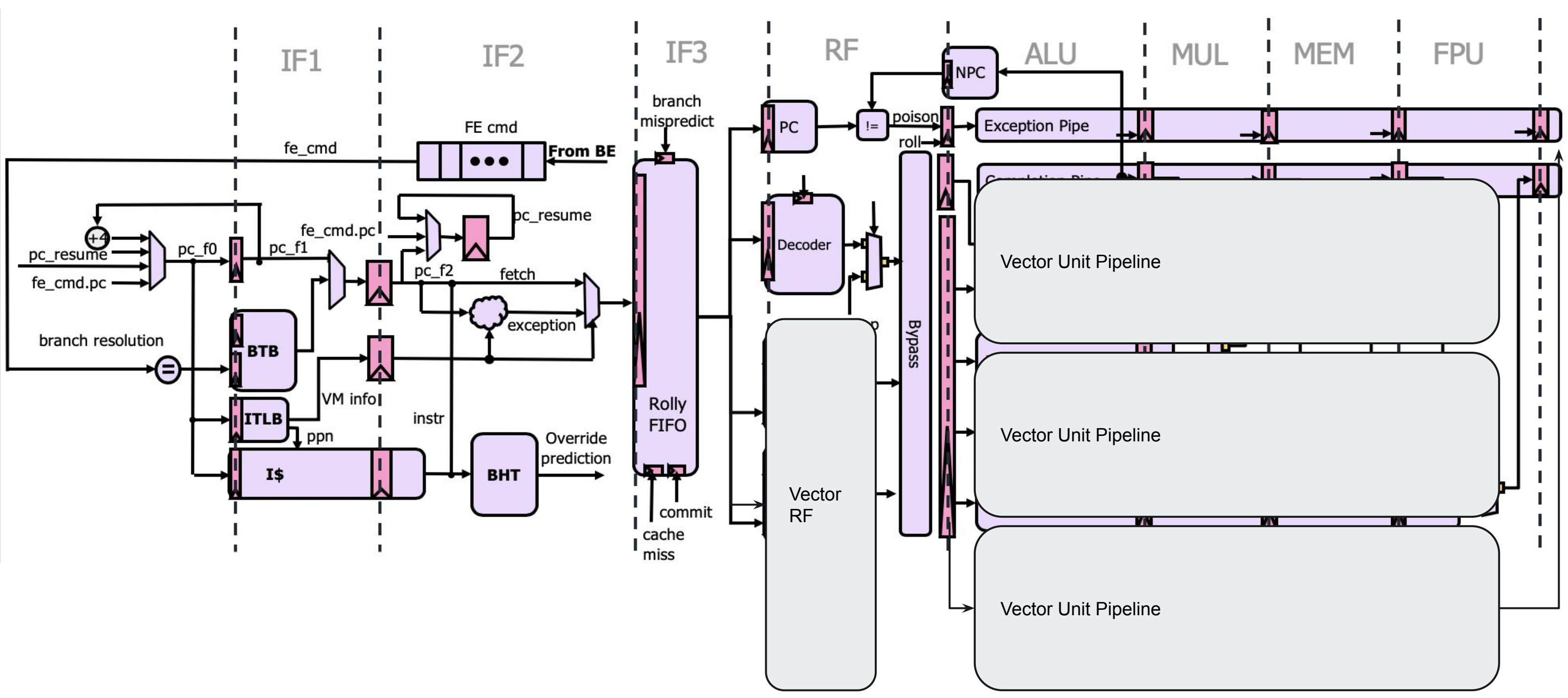
Raspberry Pi QPU

- Single instruction stream has mix of scalar and vector instructions
- How to program this style of simd unit?
- What are the overheads of needing to copy data to other parts of the chip?

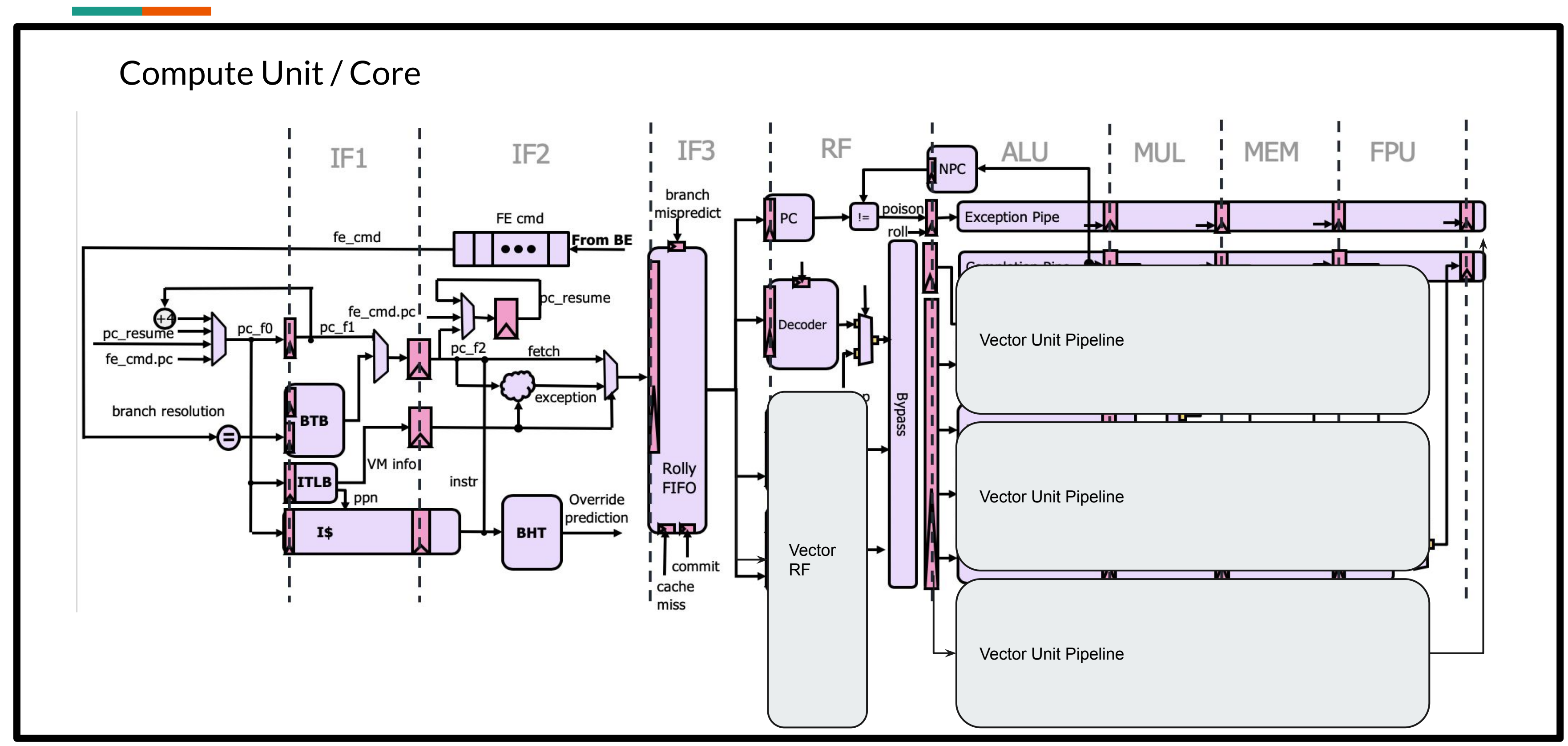


Massively Parallel Vector Processors

- Give more space to vector units, reduce the number of scalar units

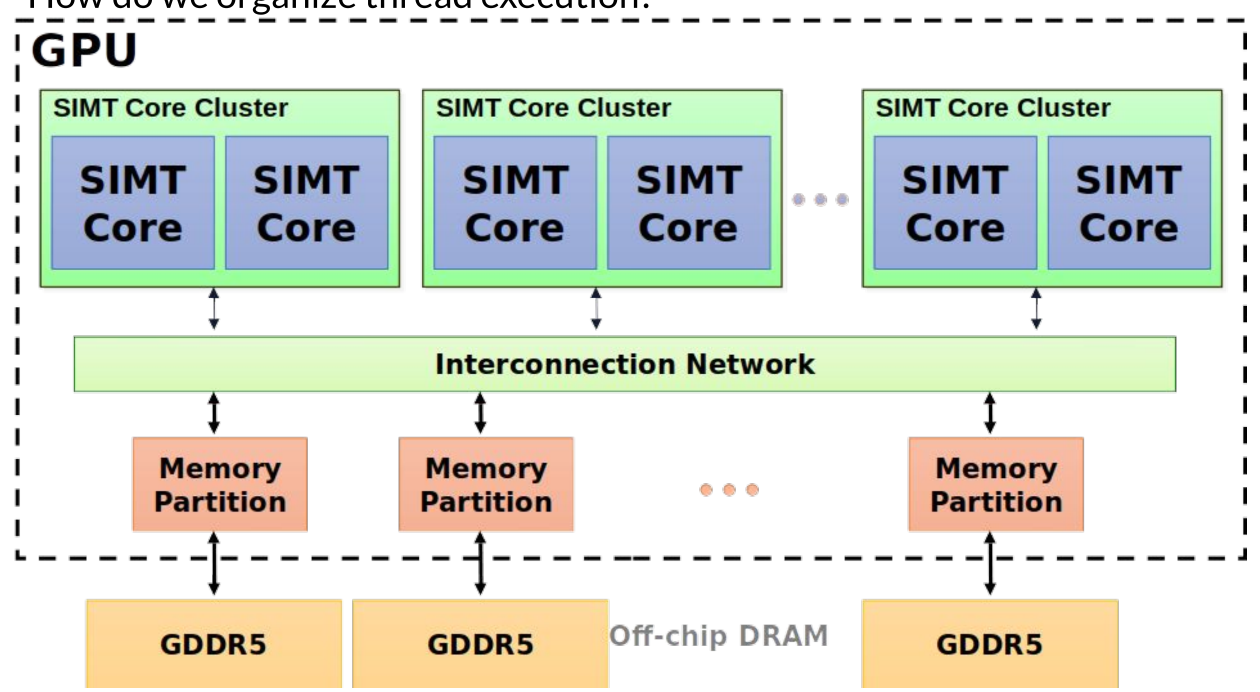


Massively Parallel Vector Processors



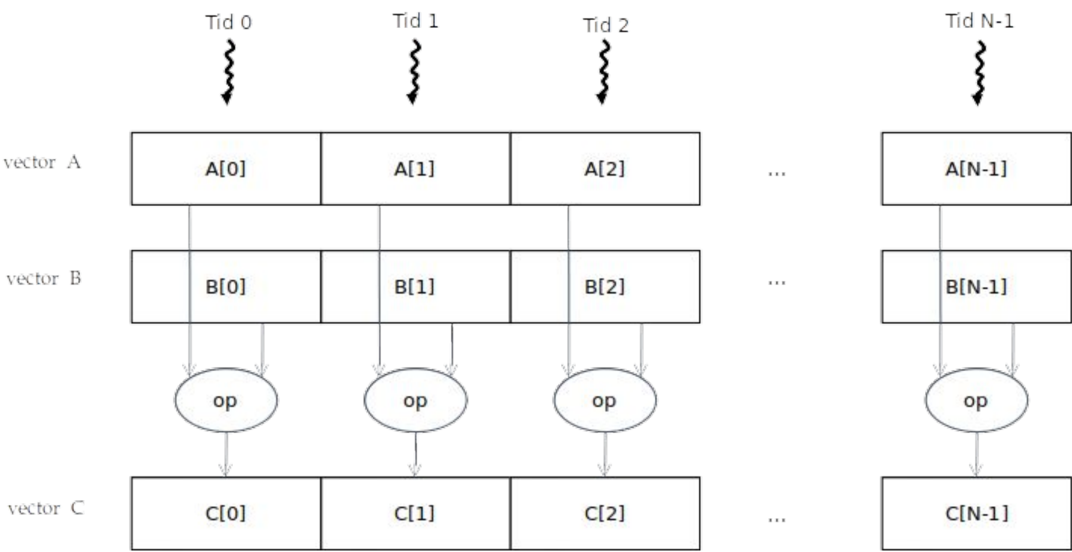
GPU Architecture

- Now we can handle thousands of threads!!
- How do we organize thread execution?



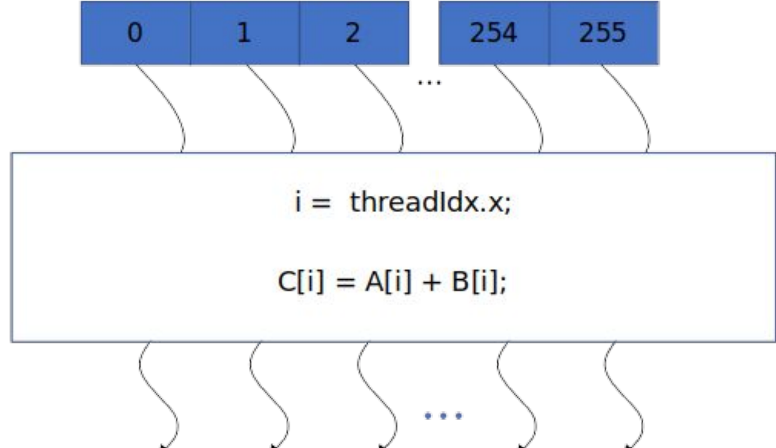
Vector Operations

- Assign a thread to each element of an array
- One thread operates on a single element



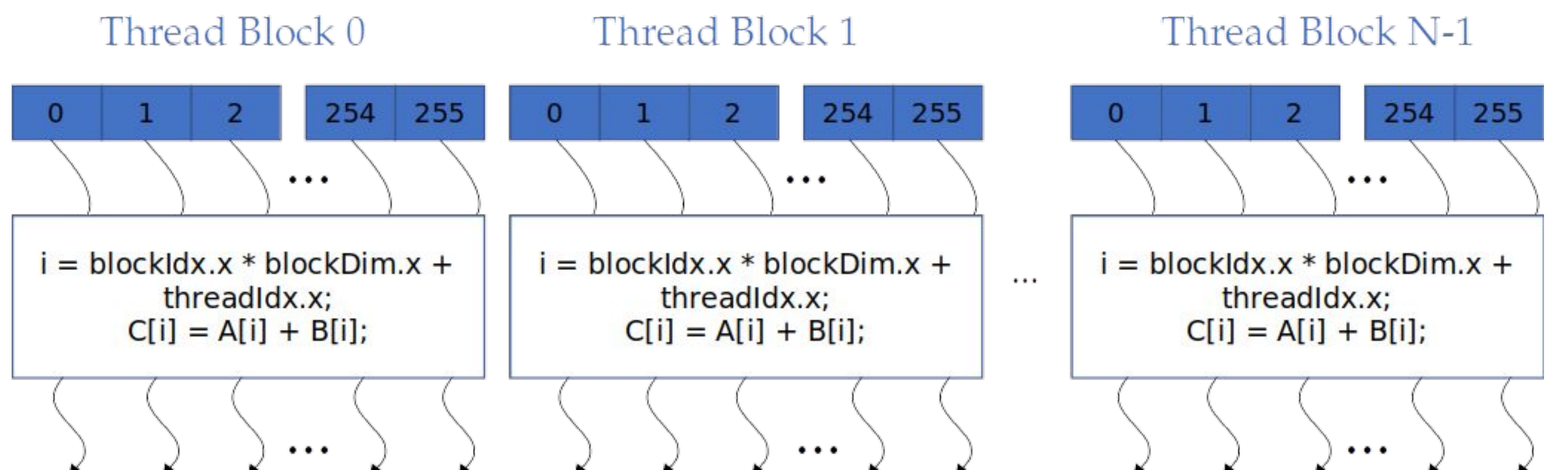
Arrays of Parallel Threads

- A CUDA kernel is executed by a grid (array) of threads
  - All threads in a grid run the same kernel code (Single Program Multiple Data)
  - Each thread has indexes that it uses to compute memory addresses and make control decision
- However this is not scalable with large arrays or matricies
- Nothing inherent with id to map to a specific SM to execute



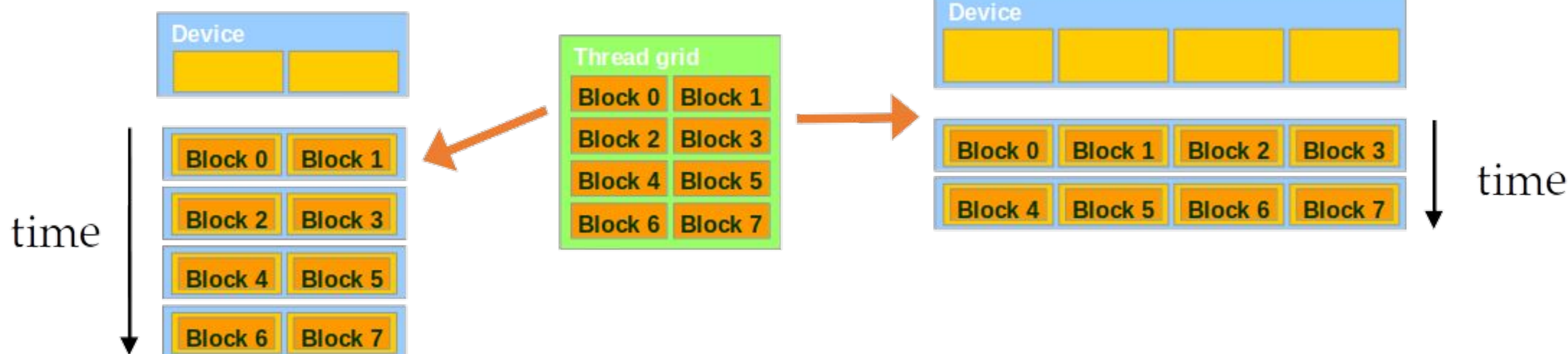
Thread Blocks: Scalable Cooperation

- Divide thread array into multiple blocks
- Thread Blocks become Unit of scheduling to an SM
- Can easily scale to different number of SMs



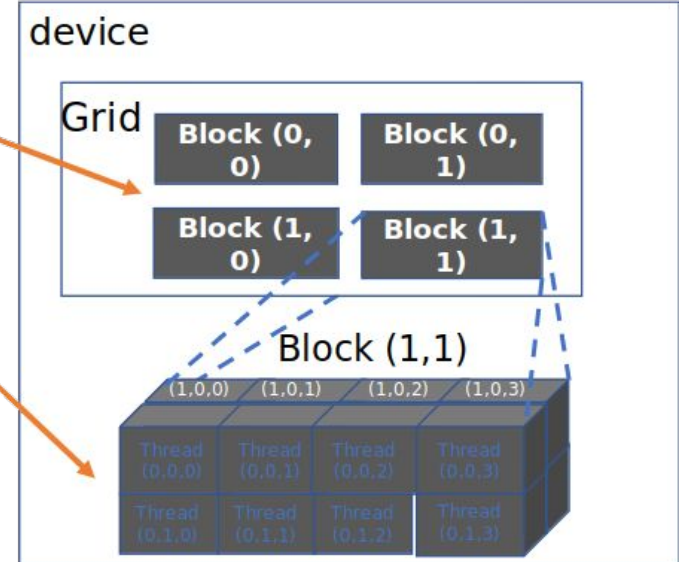
Transparent Scalability

- Each block can execute in any order relative to others.
- Hardware is free to assign blocks to any processor at any time
  - A kernel scales to any number of parallel processors



blockIdx and threadIdx

- Each thread uses indices to decide what data to work on
  - blockIdx: 1D, 2D, or 3D
  - threadIdx: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes
  - …

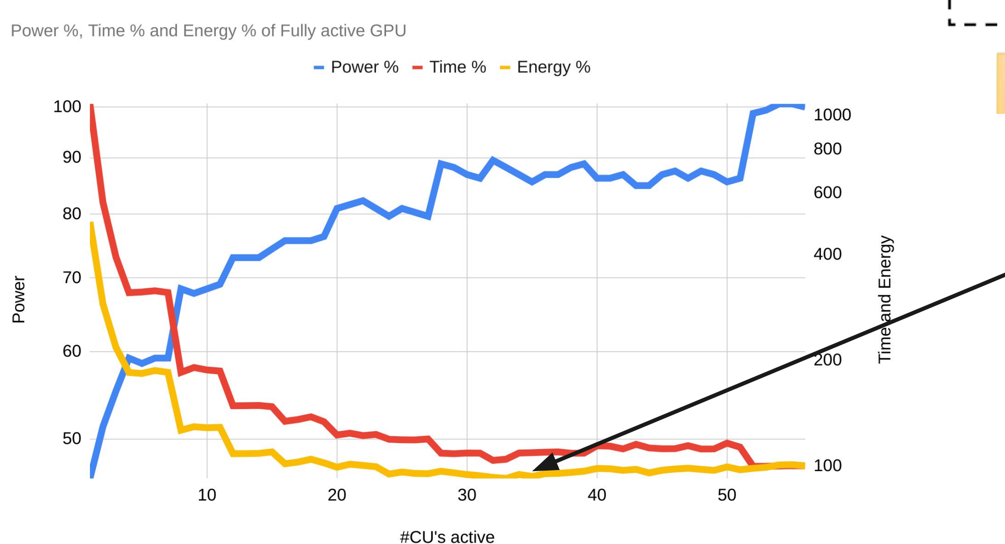


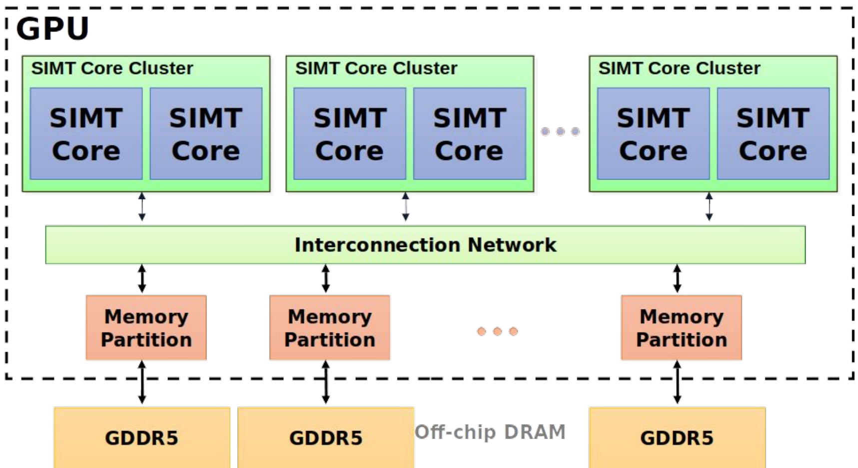
GPU Hierarchies

	Scalar	Vector	Core	Card
Hardware	ALU	ALU	SIMD	SM SM
	ALU Unit	SIMD Unit	SM	GPU
Threads				
	Thread	Warp	Thread Block	Block Grid
Memory	Register File		L1 Cache	L2 / Memory
Address Space	Local per thread		Shared Memory	Global

CU Scaling

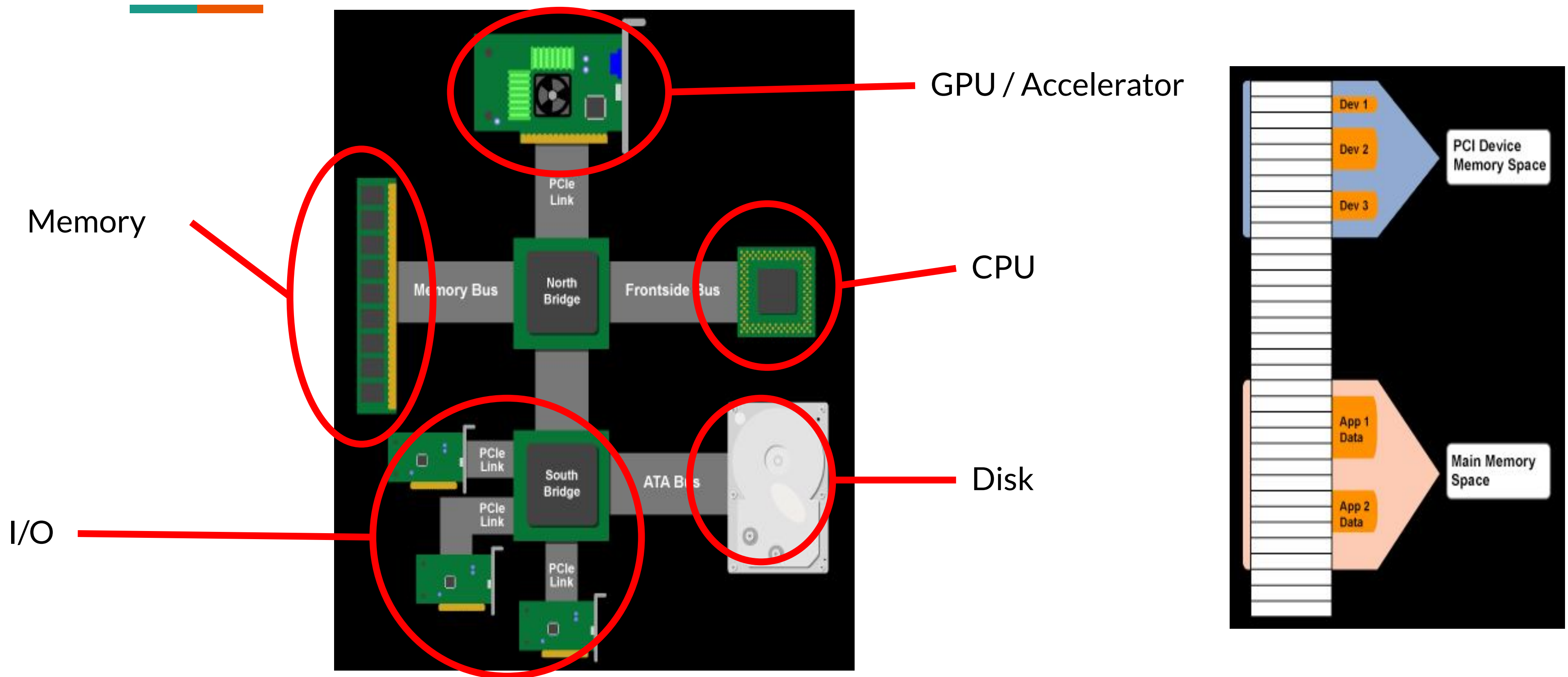
- Are using this many cores efficient?
- How should we manage all of the CUs/SEs?
- How do you keep track of utilization?





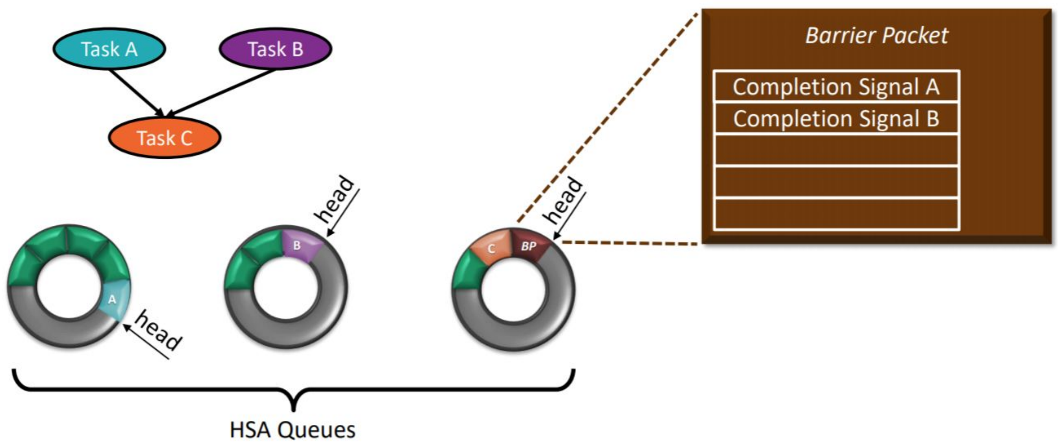
8% less energy by using only 32 CUs out of 56 total

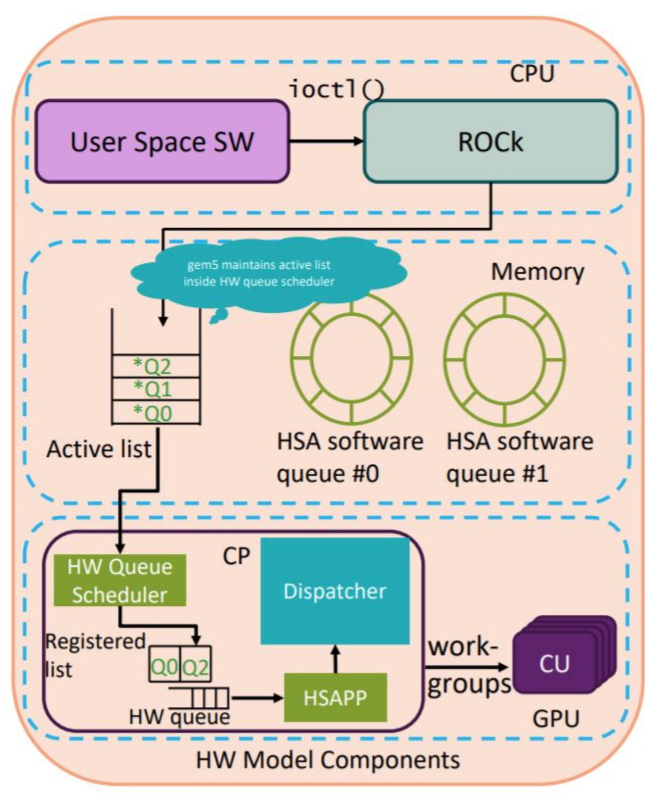
GPU in a Computer System



Kernel Launching and Tasking

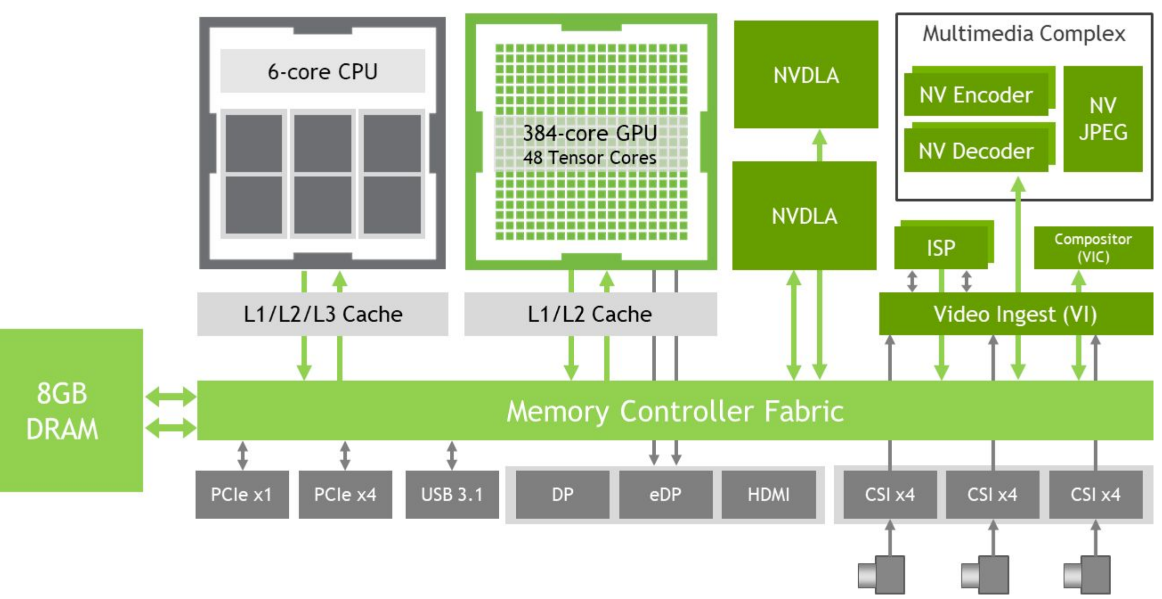
- How are kernels launched from the CPU to GPU?
- How to enforce dependencies between kernels?
- What kind of parallel algorithms does this mechanism allow us to make?





Integrated GPU Architectures

- What if CPU and GPU are on the same chip?
- How does that affect power consumption?
- How do we control both set of cores now?



Sustainable Computing Architectures

- Parallel architectures are energy efficient
- Everyone is trying to make using them more efficient and cost effective