信息科学与技术学院
School of Information Science and Technology

CS 110
Computer Architecture

Cache I

Instructors:
Chundong Wang, Siting Liu & Yuan Xiao
Course website: https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2025/index.html
School of Information Science and Technology (SIST)
ShanghaiTech University

2025/4/15

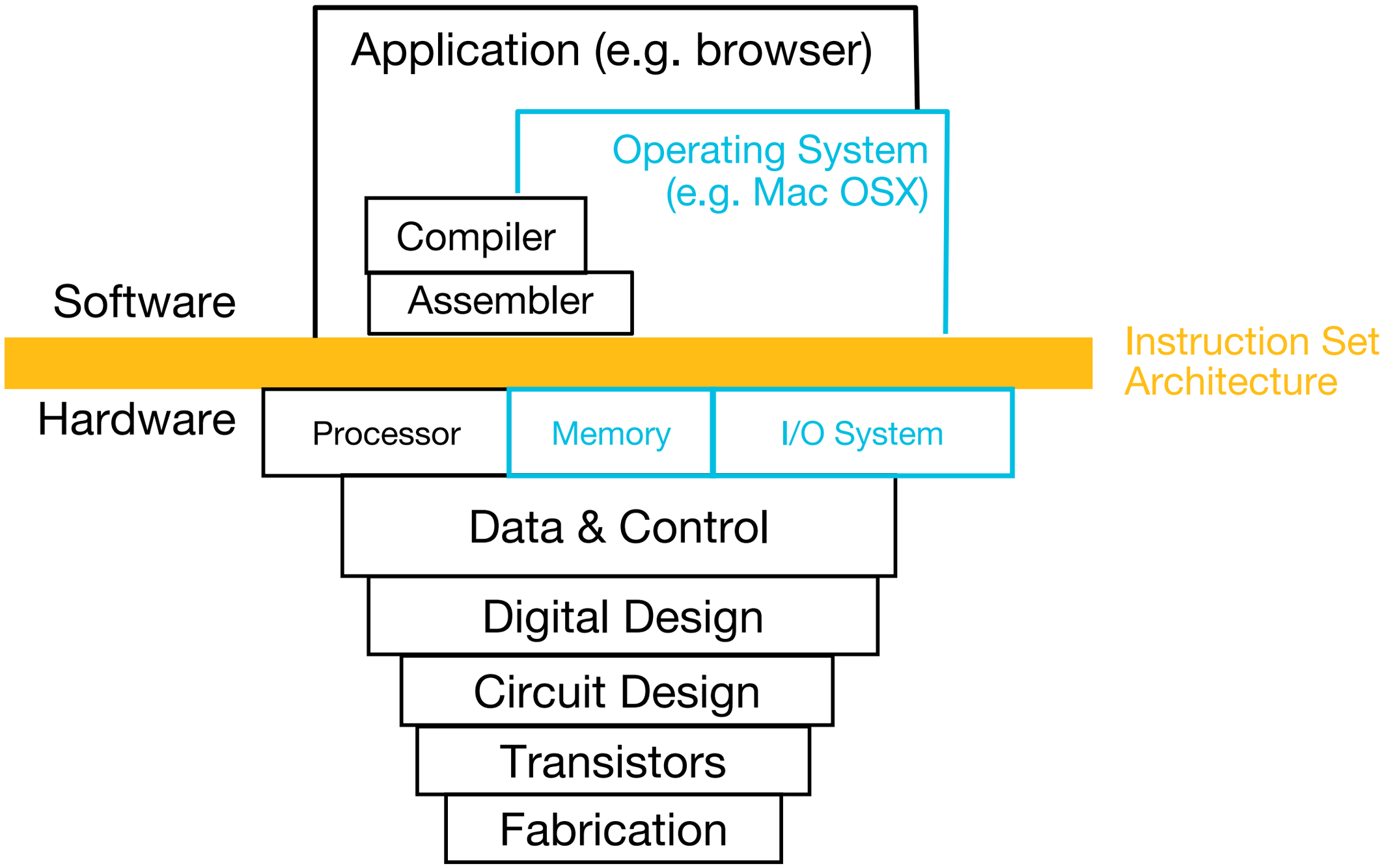
Administratives

- Mid-term I score published. Please check if there are any questions regarding your marks. Submit regrade request if you have any concerns before this FRIDAY (Apr. 18th 23:59:59)!
- Project 2.1 released, start early!!!
- HW4 ddl TODAY! Submit your answer to gradescope.
- Project 1.2 ddl Apr. 17th.
- Lab 8 checking this week. Lab 9 will be released and checked next week.

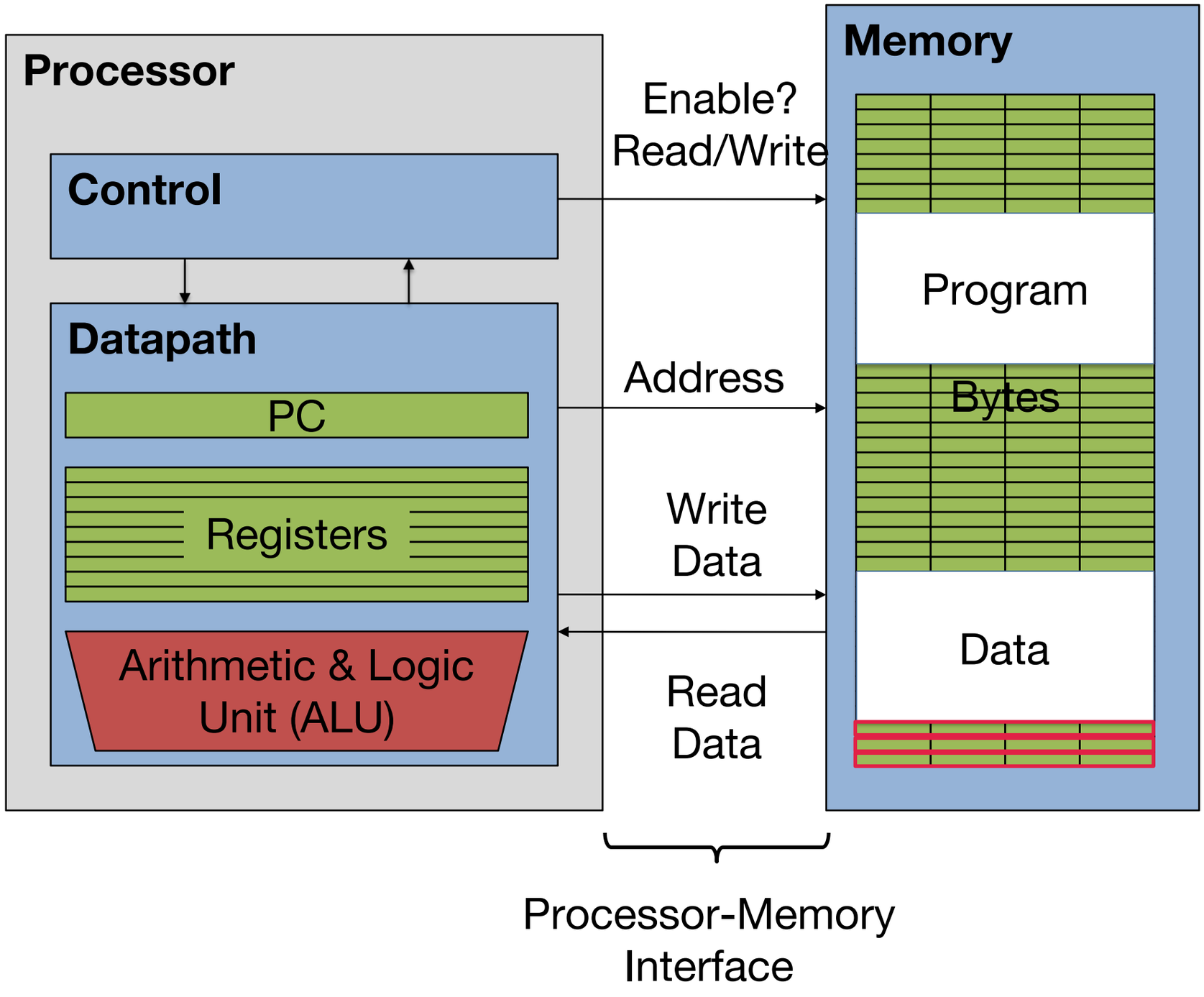
Outline

- Starting this lecture, we will improve the performance of our CPU
  - Memory hierarchy (cache)
    - Introduction to cache
    - Principle of Locality
    - Simple Cache
      - Direct Mapped & Set-Associative Caches
      - Stores to Caches
      - Cache Performance
      - Cache Misses
  - Multi-Level Caches
    - Cache Configurations
    - Cache Examples
    - Other topics (advanced cache, coherence, inclusiveness, etc.)

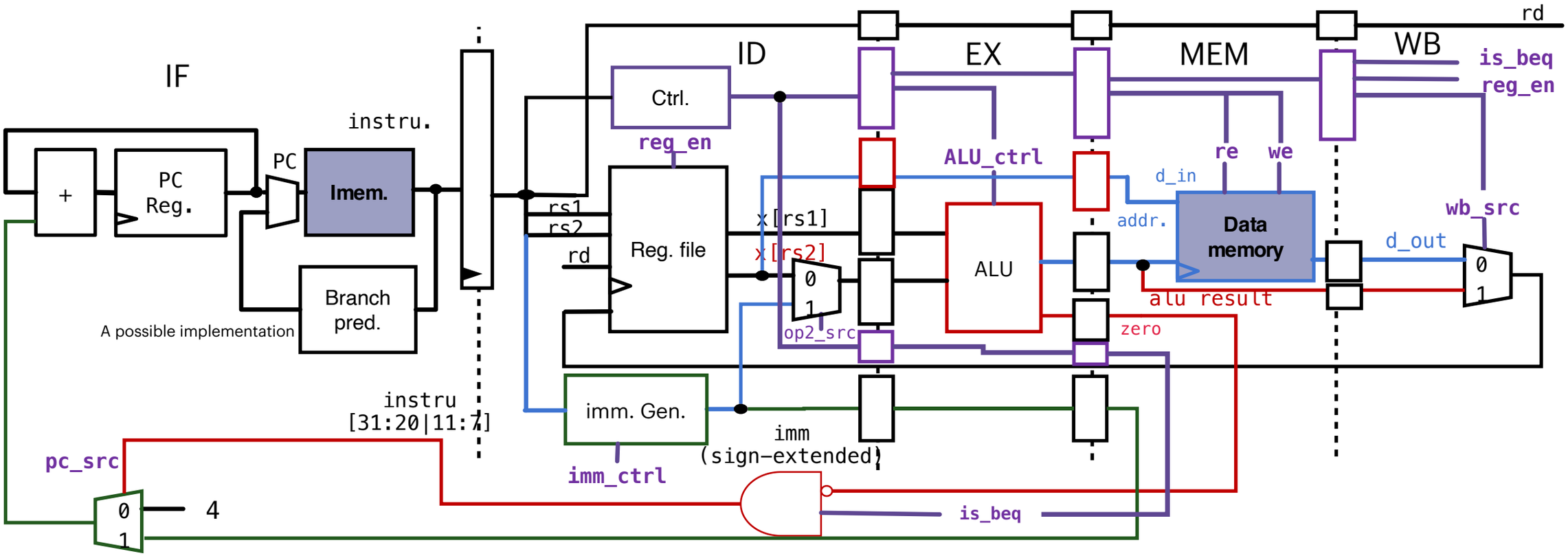
Where were we?



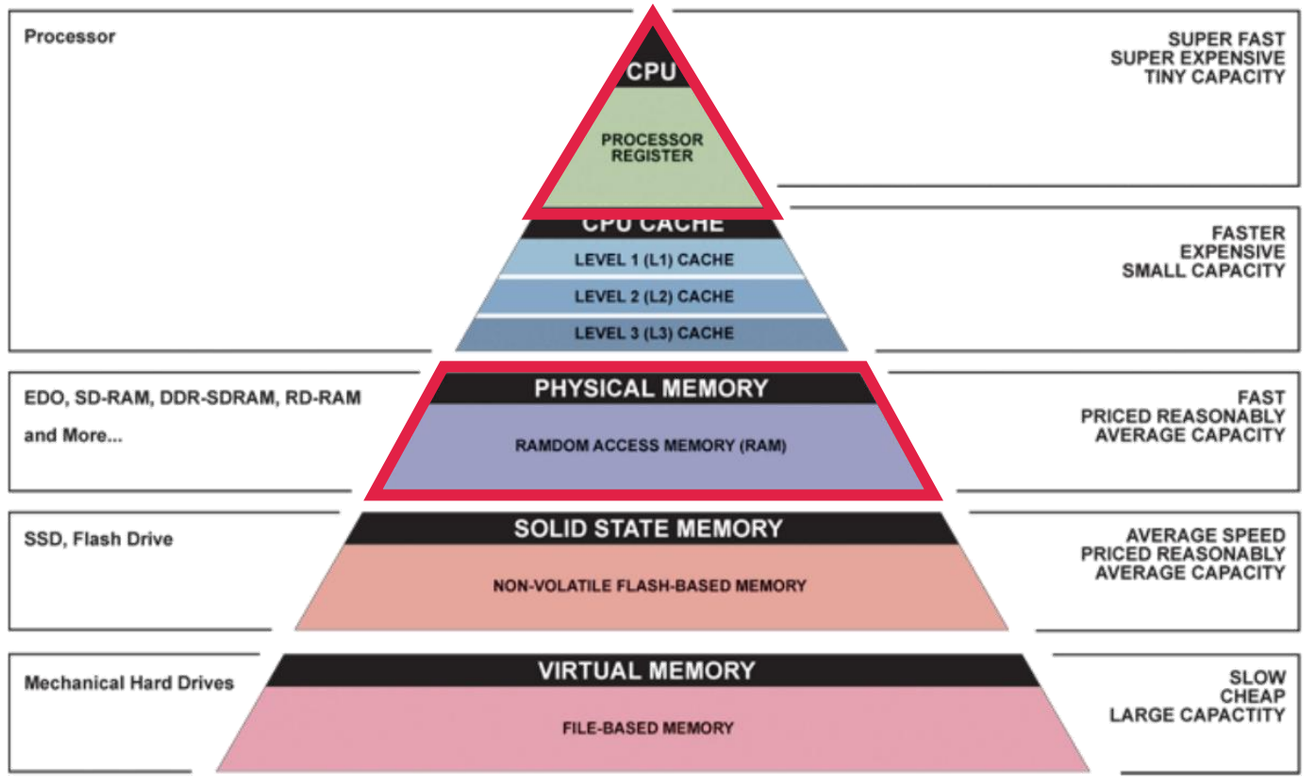
Where were we?



Where were we?



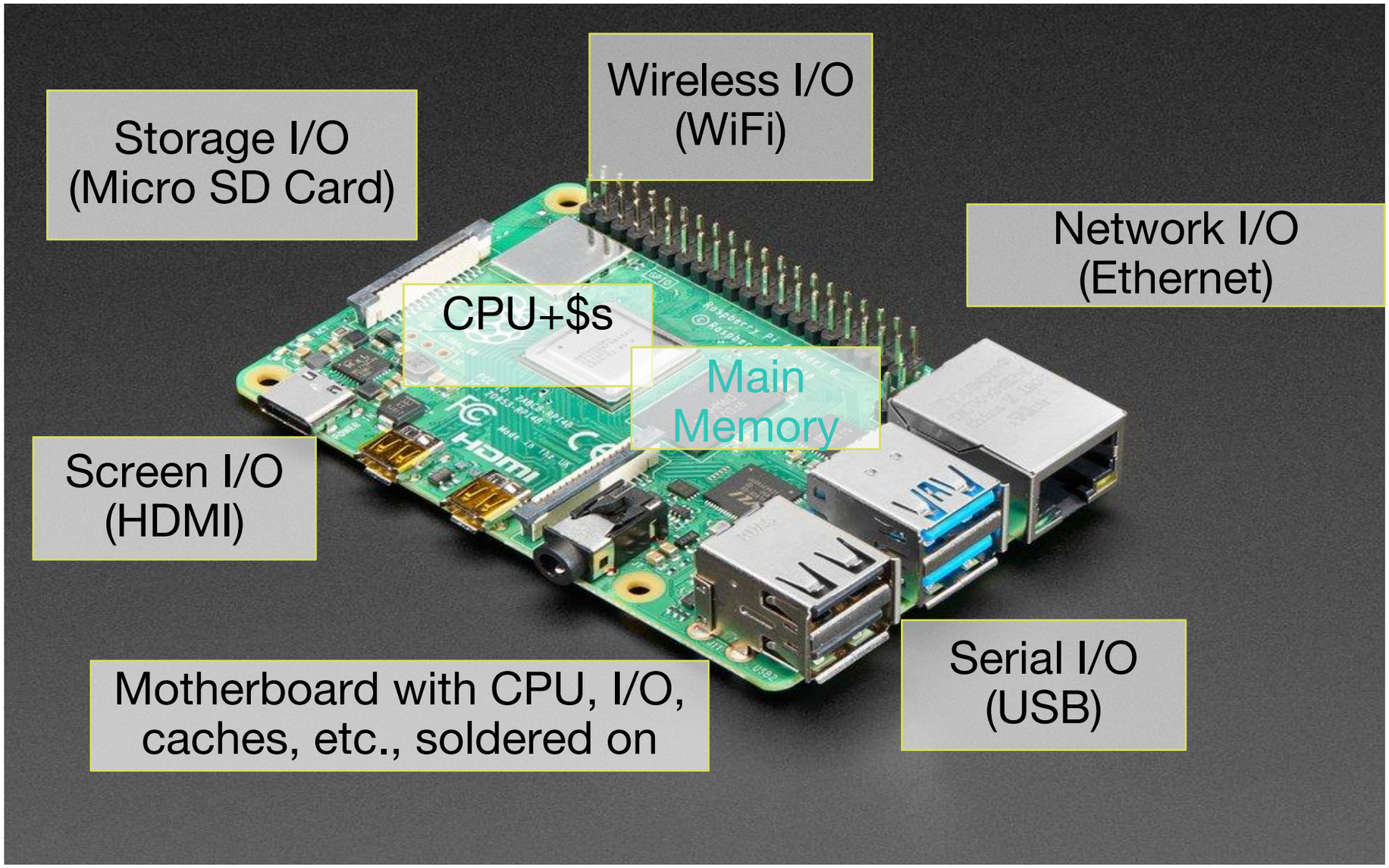
Rough access time (ns)	Components
1	Registers
10	Cache
100	Main Memory
10^6	Solid State Drive Hard Disk Drive



Introduction to memory hierarchy

- The memories we have got due to the advance of material science

The Raspberry Pi is a low-cost computer.



- Cache Memory on processor (“$” stands for cache)
- Main Memory
- SD card as secondary memory

DRAM & SRAM

- Dynamic Random Access Memory as main memory:
  - Latency to access first word: ~10ns (~30-40 processor cycles), each successive (0.5ns – 1ns)
  - Each access brings 64 (depending on the actual hardware) bits
  - $3/GiB
- Data is impermanent:
  - Dynamic: capacitors store bits, so needs periodic refresh to maintain charge
  - Volatile: when power is removed, loses data.
- Contrast with SRAM (for caches that will be covered in the following lectures, on-chip memory, also can be used for register file):
  - Static (no capacitors) but still volatile
  - Faster (0.5 ns)/more expensive/lower density

Storage/"Disk"/Secodary Memory

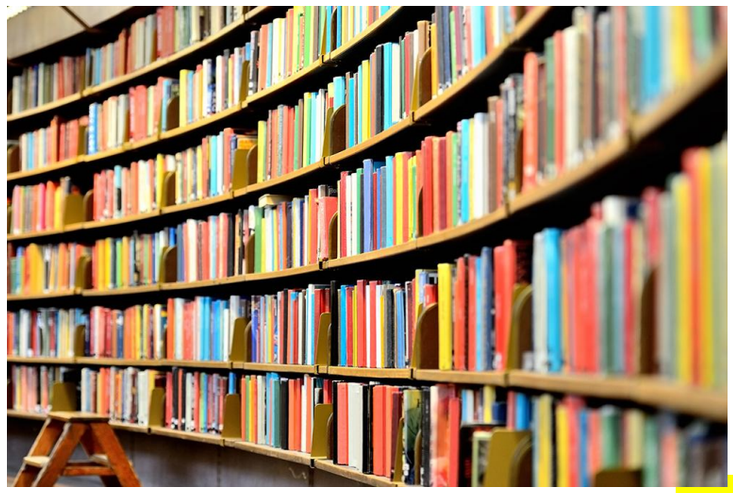
Usually attached as a peripheral I/O device and non-volatile.

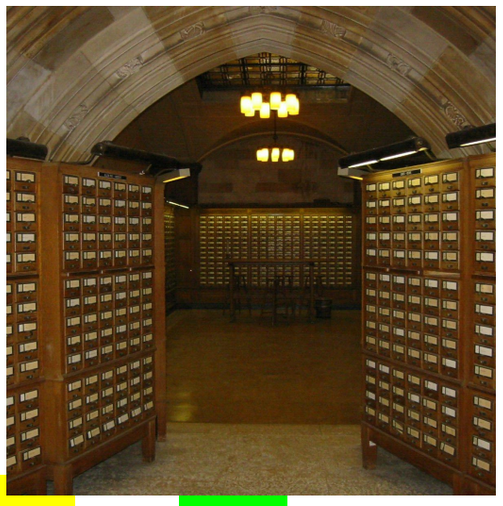
- Hard Disk Drive (HDD)
  - Access: <5-10ms (10-20M proc. cycles)
  - $0.01-0.1/GB
  - Mechanical

- Solid-State Drive (SSD)
  - Access: 40-100µs (~100k proc. cycles)
  - $0.05-0.5/GB
  - Usually flash memory

Introduction to memory hierarchy

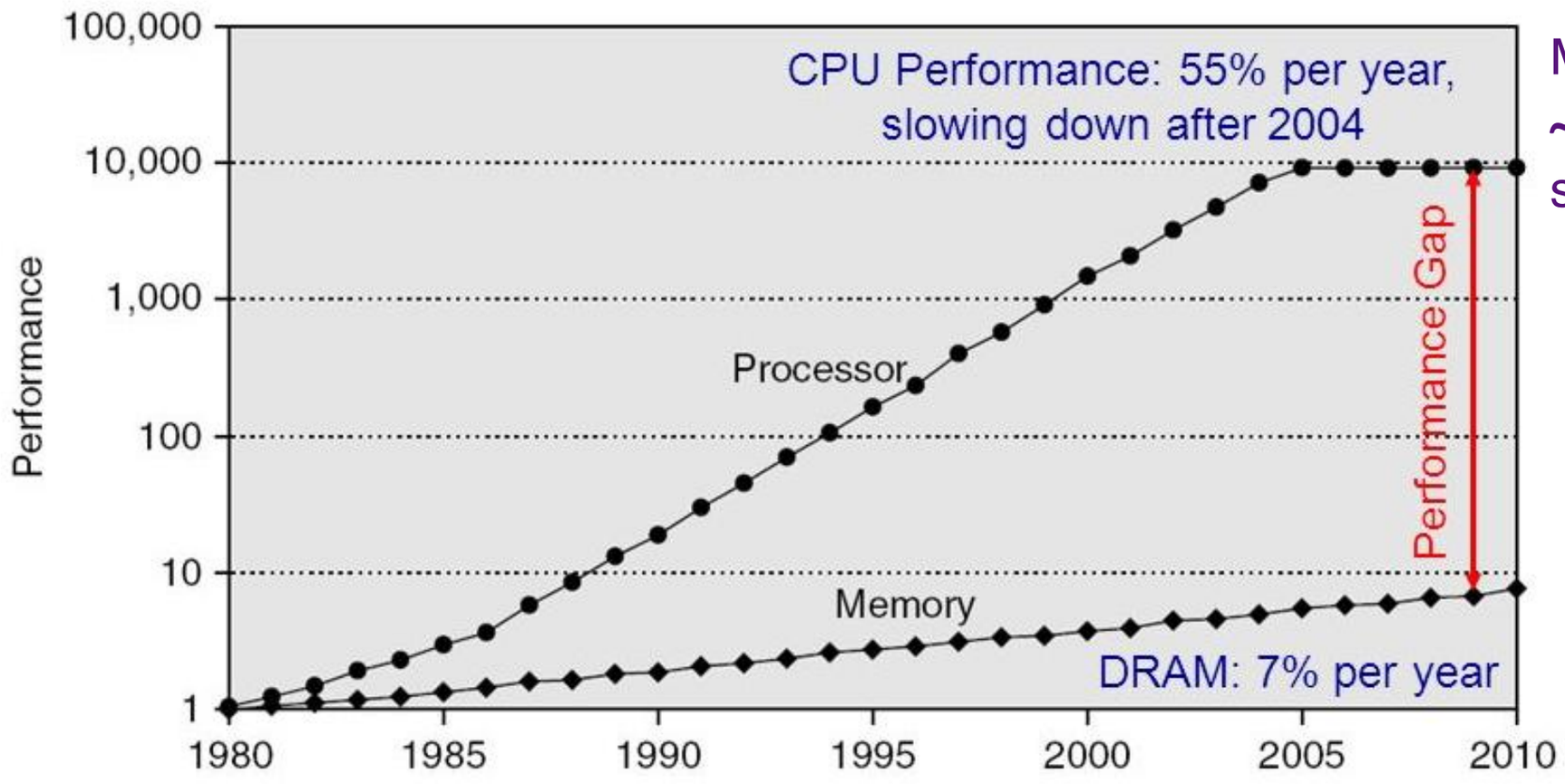
- Motivation: Large memories slow? Library Analogy
  - Finding a book in a large library takes time
    - Takes time to search a large card catalog – (mapping title/author to index number)
    - Round-trip time to walk to the stacks and retrieve the desired book.
  - Larger libraries makes both delays worse
  - Electronic memories have the same issue, *plus* the technologies that we use to store an individual bit get slower as we increase density (SRAM versus DRAM versus Magnetic Disk)





However what we want is a large yet fast memory!

Processor-DRAM Gap (Latency)



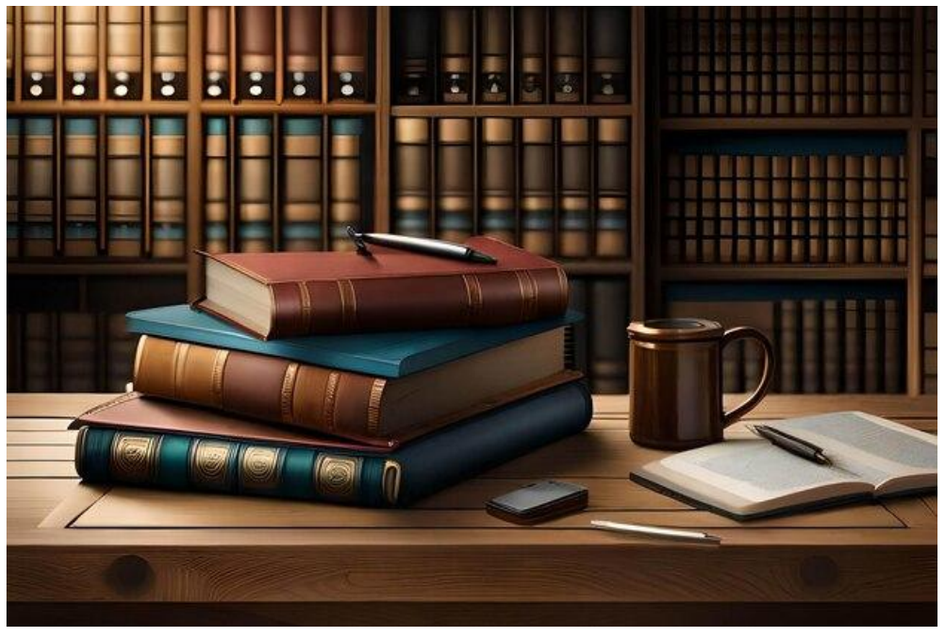
Microprocessor executes **~1000 instructions** in the same time as DRAM access

Microprocessor executes ~one instruction in the same time as DRAM access

Slow DRAM access has disastrous impact on CPU performance!

Library Analogy

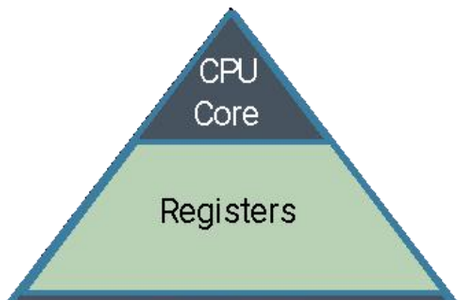
- Want to write a report for CA using library books
- Go to library, look up relevant CA books, fetch from stacks, and place on desk in library
- If need more, check them out and keep on desk
  - But don't return earlier books since might need them (locality)
- You hope this collection of ~10 books on desk enough to write the report, despite 10 being only a tiny fraction of books available



Great Ideas

- Memory access is a bottleneck
- Make common case fast (quick access to frequently used data)
- Memory hierarchy/Principle of locality

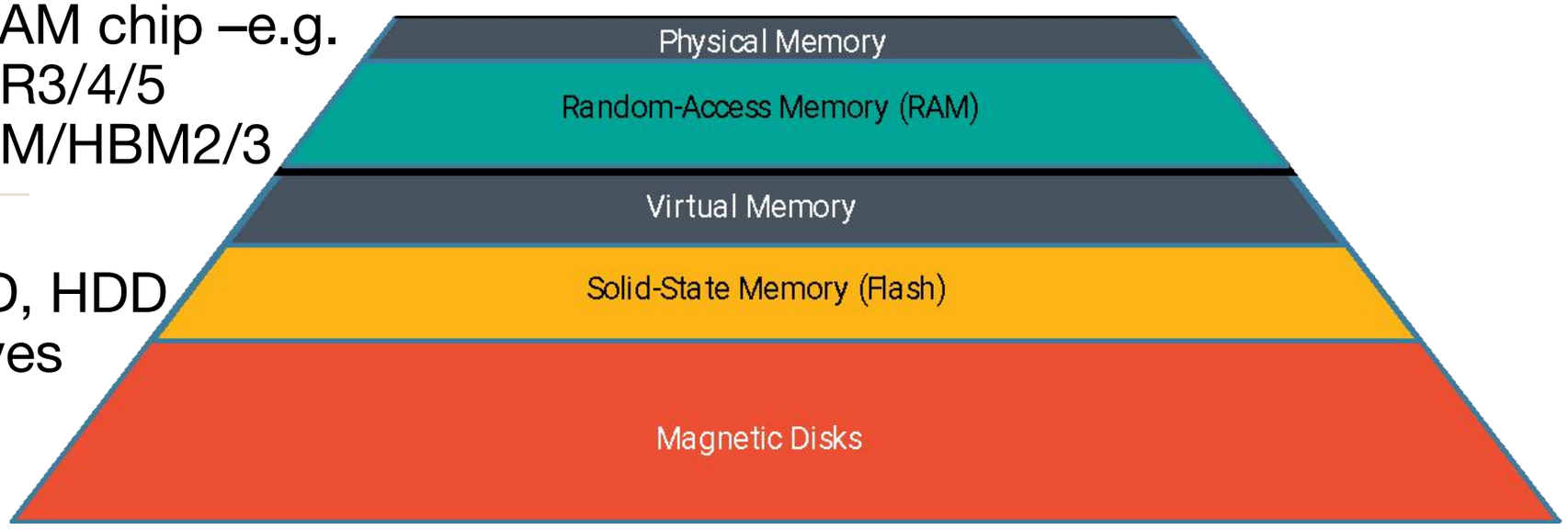
Processor chip



Super fast, super expensive, tiny capacity

DRAM chip –e.g. DDR3/4/5 HBM/HBM2/3

SSD, HDD Drives



Faster, expensive, small capacity

Fast, reasonable cost, average capacity

Great Ideas

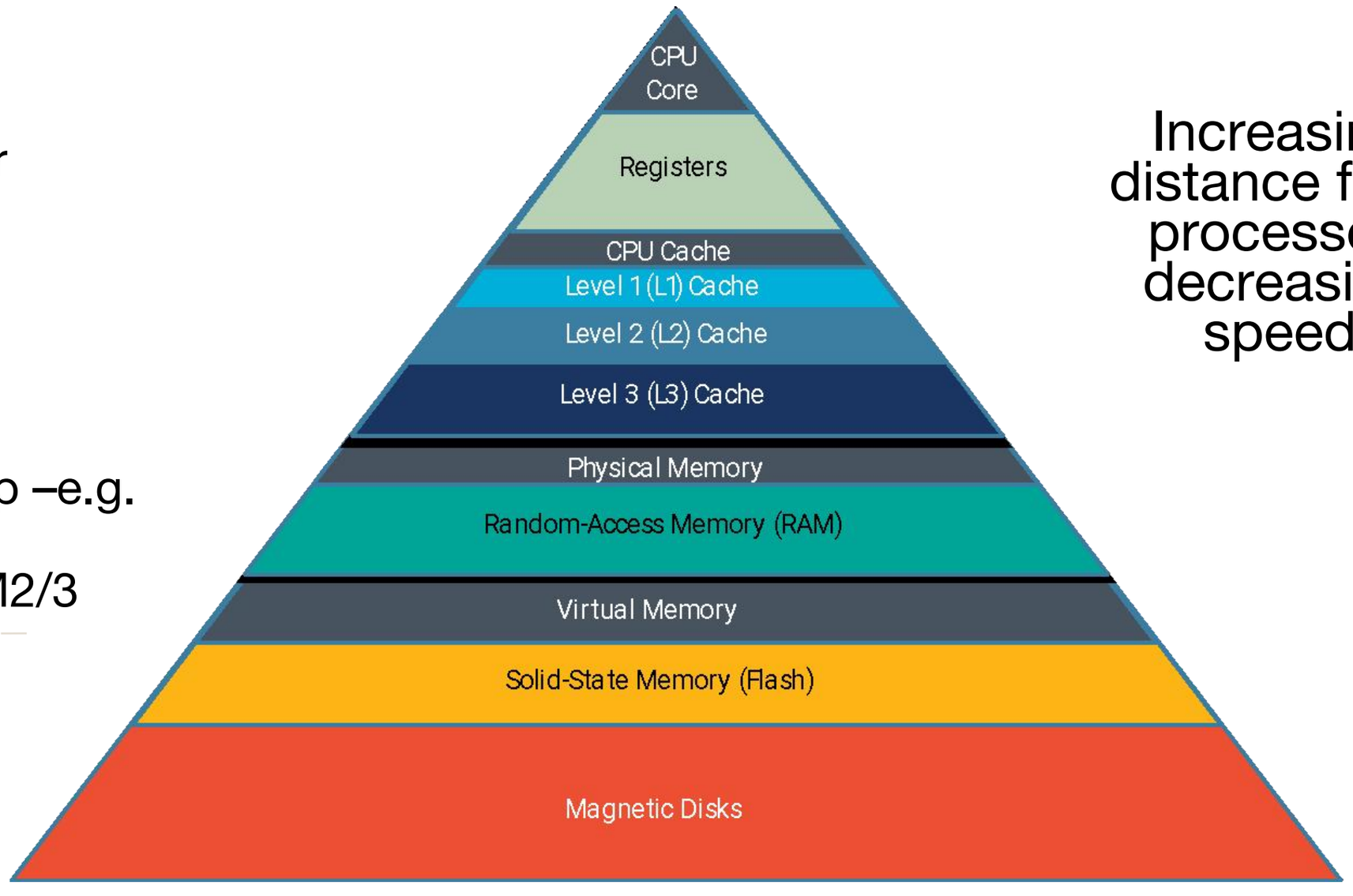
- Memory access is a bottleneck
- Make common case fast (quick access to frequently used data)
- Memory hierarchy/Principle of locality

Processor chip

On-chip SRAM

DRAM chip –e.g. DDR3/4/5 HBM/HBM2/3

SSD, HDD Drives



Great Ideas

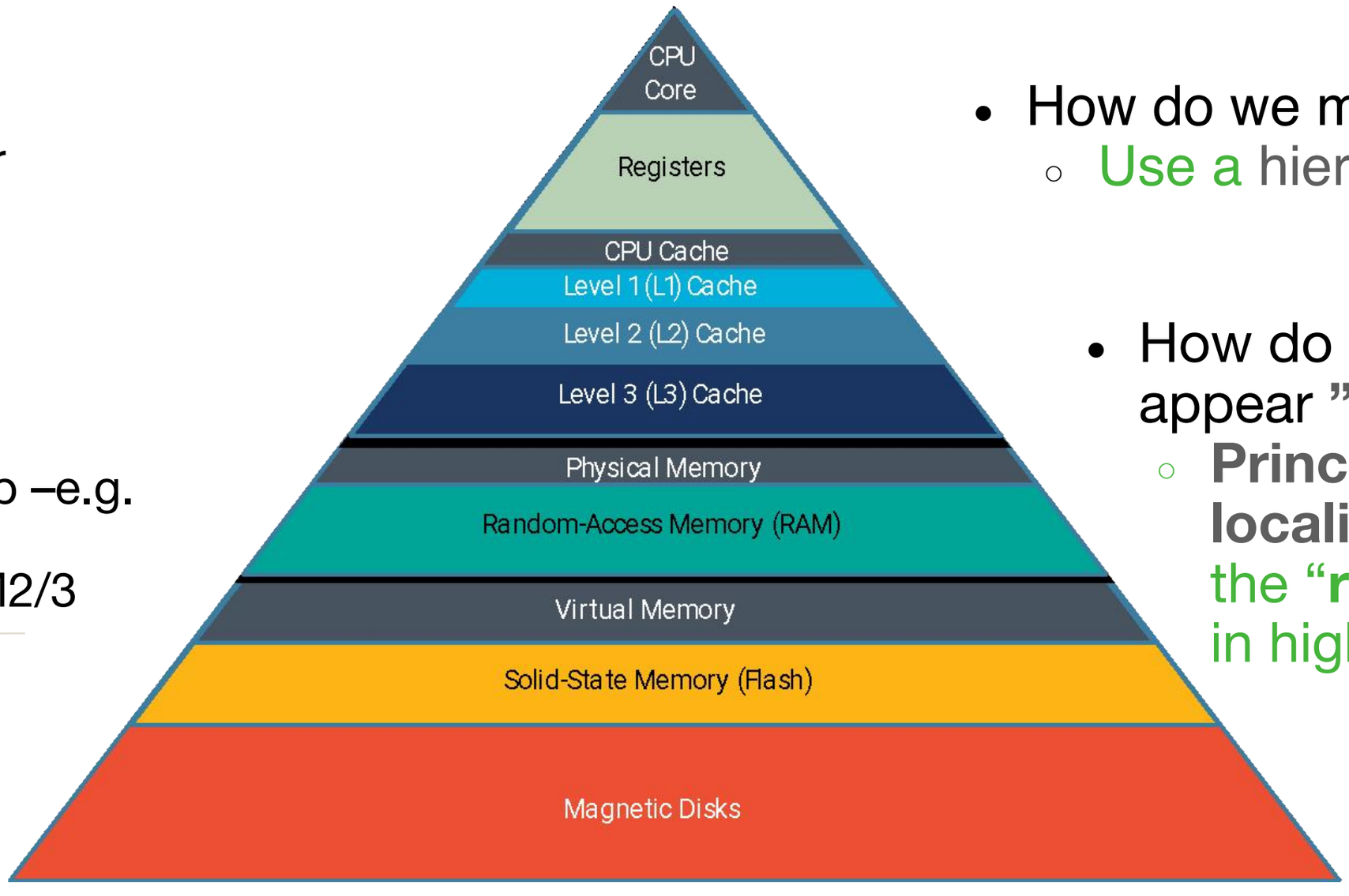
- Memory access is a bottleneck
- Make common case fast (quick access to frequently used data)
- Memory hierarchy/Principle of locality

Processor chip

On-chip SRAM

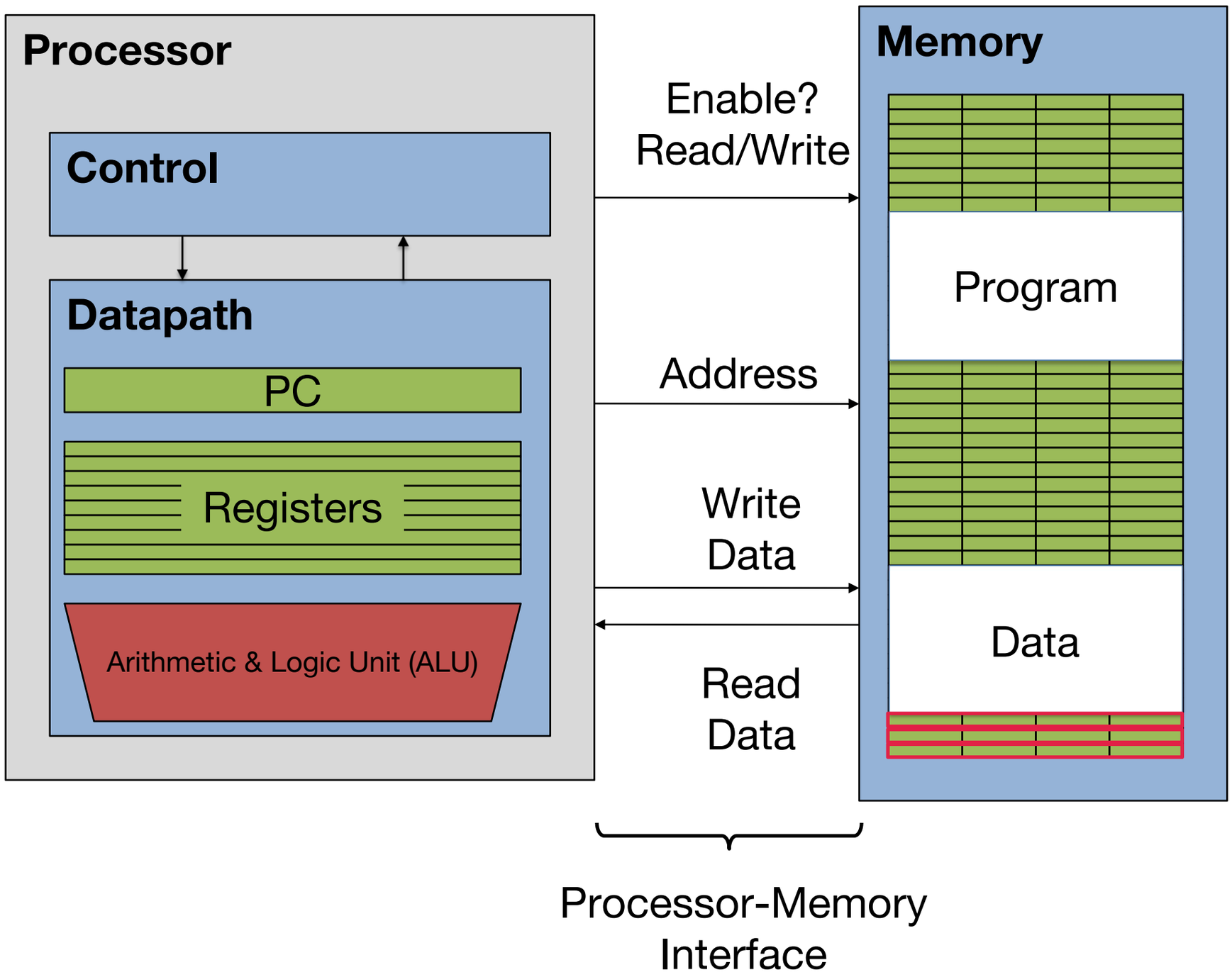
DRAM chip –e.g. DDR3/4/5 HBM/HBM2/3

SSD, HDD Drives

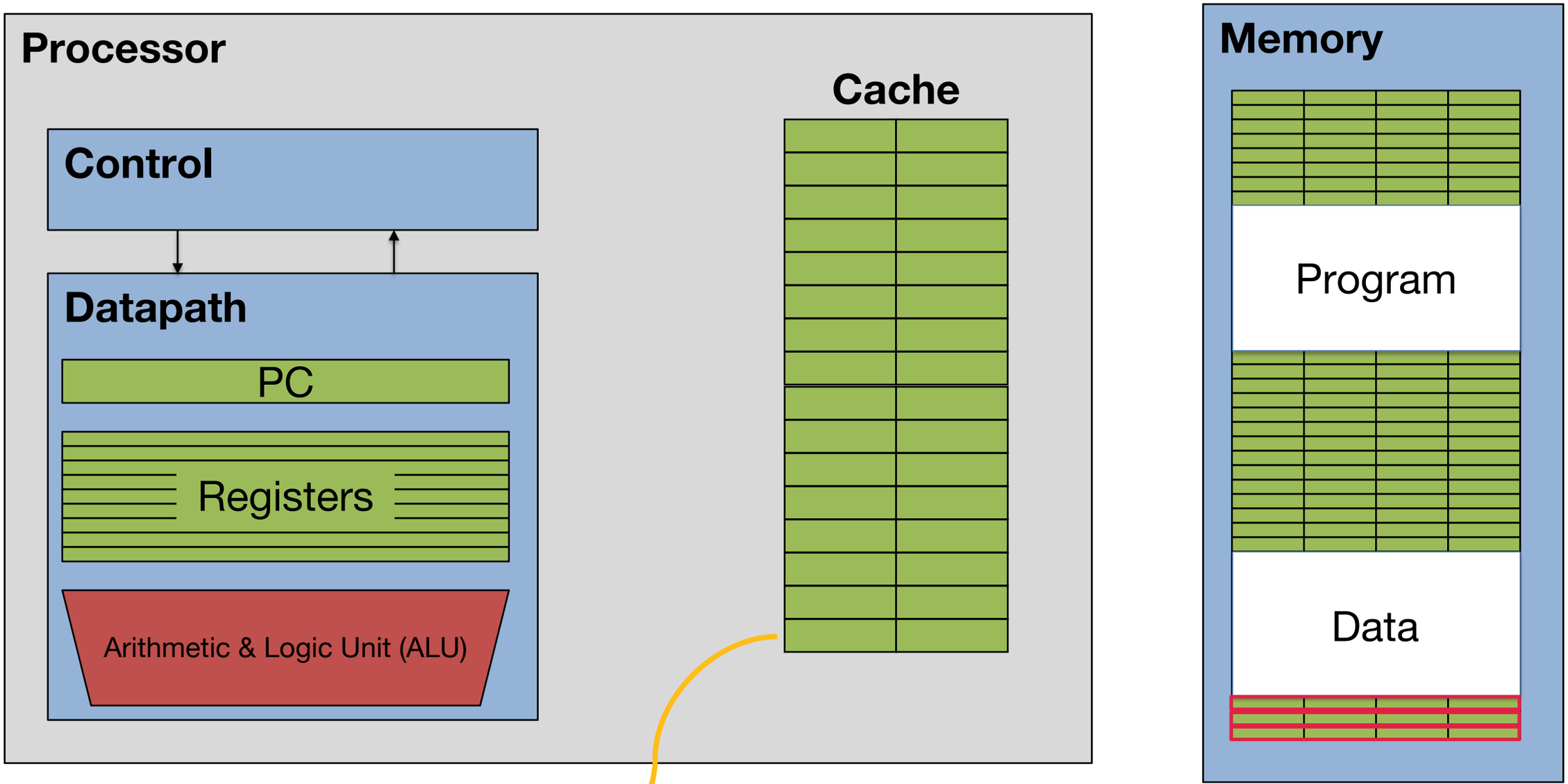


- How do we make it fast?
  - Use a hierarchy.
- How do we make it appear ”large”?
  - **Principle of locality**: **Cache** the “**right**” data in higher levels.

Hardware Implementation



Hardware Implementation



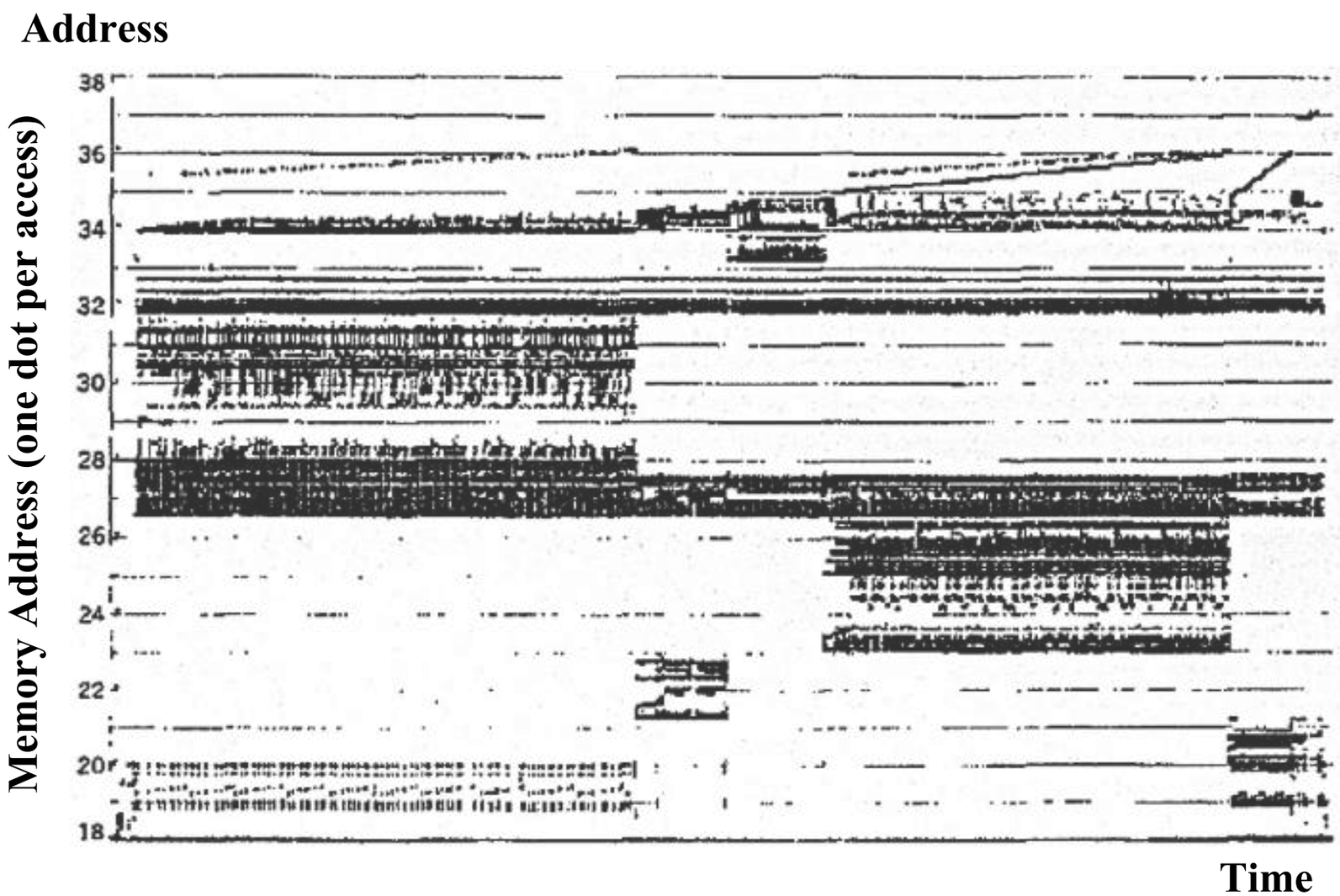
Lines (Blocks) of data copied from the main memory (Copy of subset of the main memory)

Caching: the Basis of the Memory Hierarchy

- A cache contains copies of data that are being used.
- A cache works on the principles of temporal and spatial locality.

	Temporal Locality	Spatial Locality
Idea	If we use it now, chances are that we'll want to use it again soon.	If we use a piece of memory, chances are we'll use the neighboring pieces soon.
Library Analogy	We keep a book on the desk while we check out another book.	If we check out a book's vol. 1 while we're at it, we'll also check out vol. 2.
Memory	If a memory location is referenced, then it will tend to be **referenced again soon**. Therefore, keep **most recently accessed data** items closer to the processor.	If a memory location is referenced, the locations with **nearby addresses** will tend to be referenced soon. Move **lines** consisting of **contiguous words** closer to the processor.

Real Memory Reference Patterns



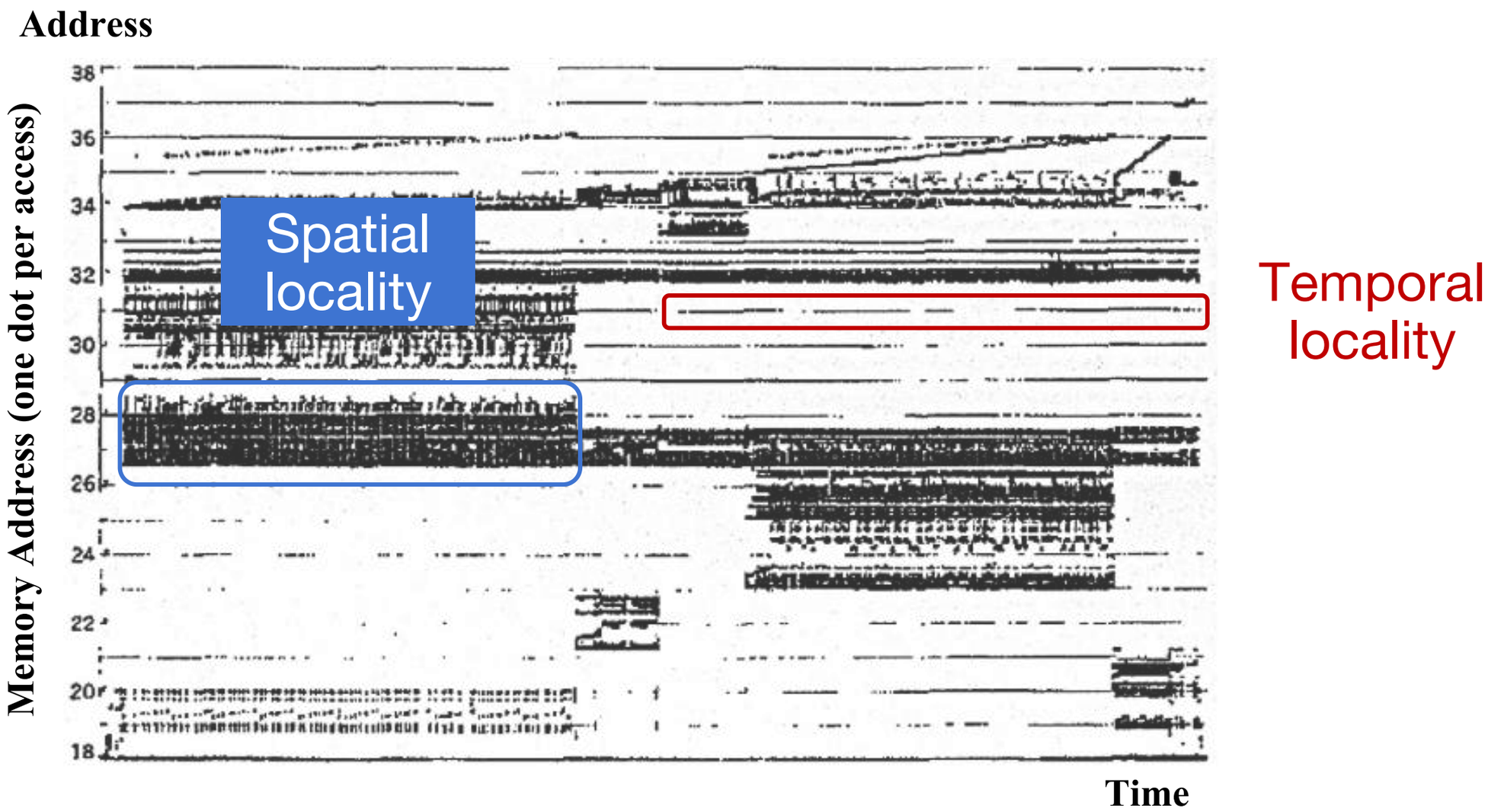
Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Locality!!!

- **Temporal locality** (locality in time)
  - If a memory location is referenced, then it will tend to be referenced again soon
- **Spatial locality** (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

```
// Sample code for CS110@Spring 2025 -- Chundong
for (i = 0, sum = 0; i < n; ++i)
{
            sum += a[i];
}
```

Real Memory Reference Patterns

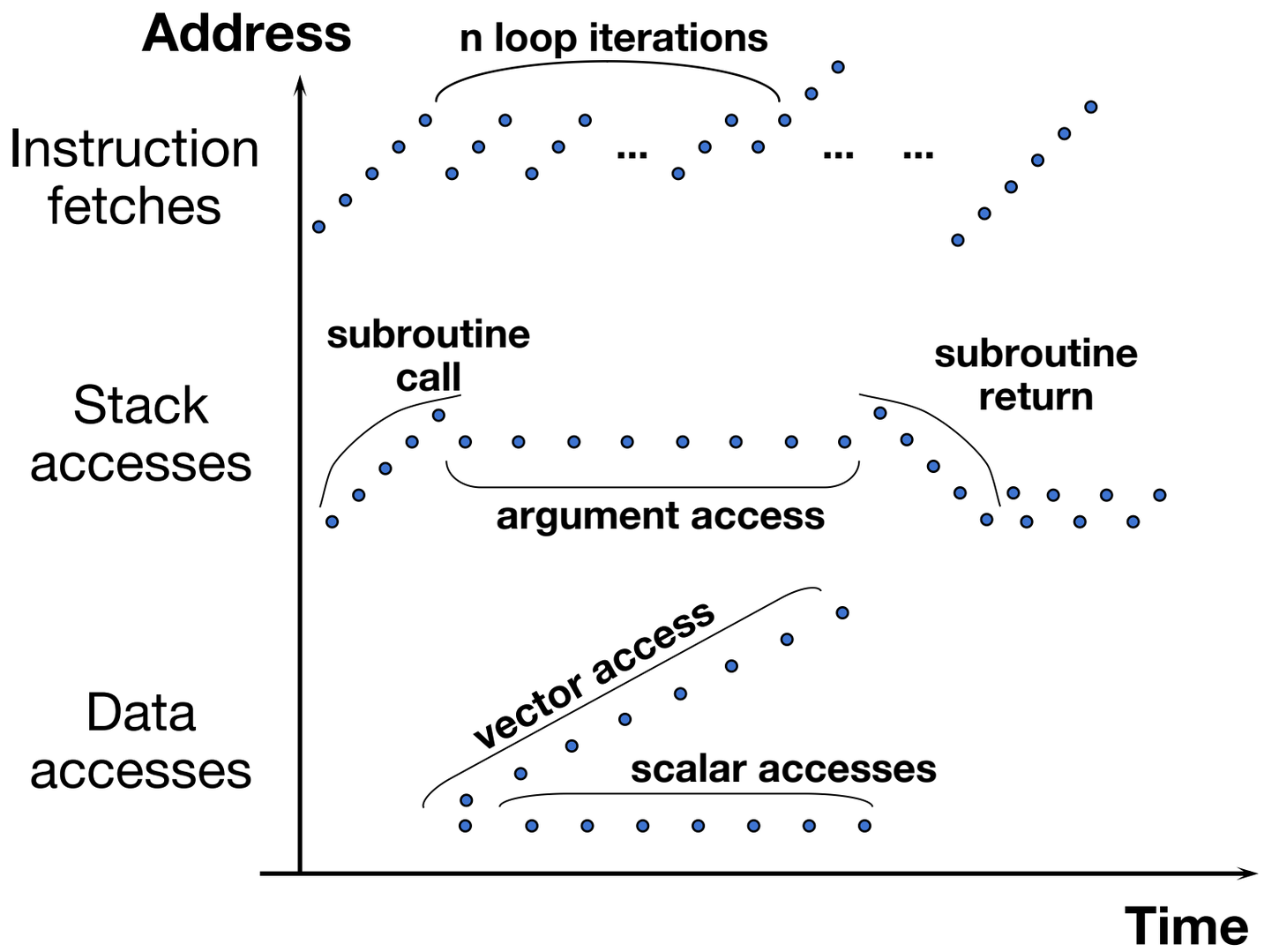


Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Principle of Locality

- *Principle of Locality*: Programs access small portion of address space at any instant of time (spatial locality) and repeatedly access that portion (temporal locality)
- What program structures lead to temporal and spatial locality in **instruction** accesses?
- In **data** accesses?

Memory Reference Pattern

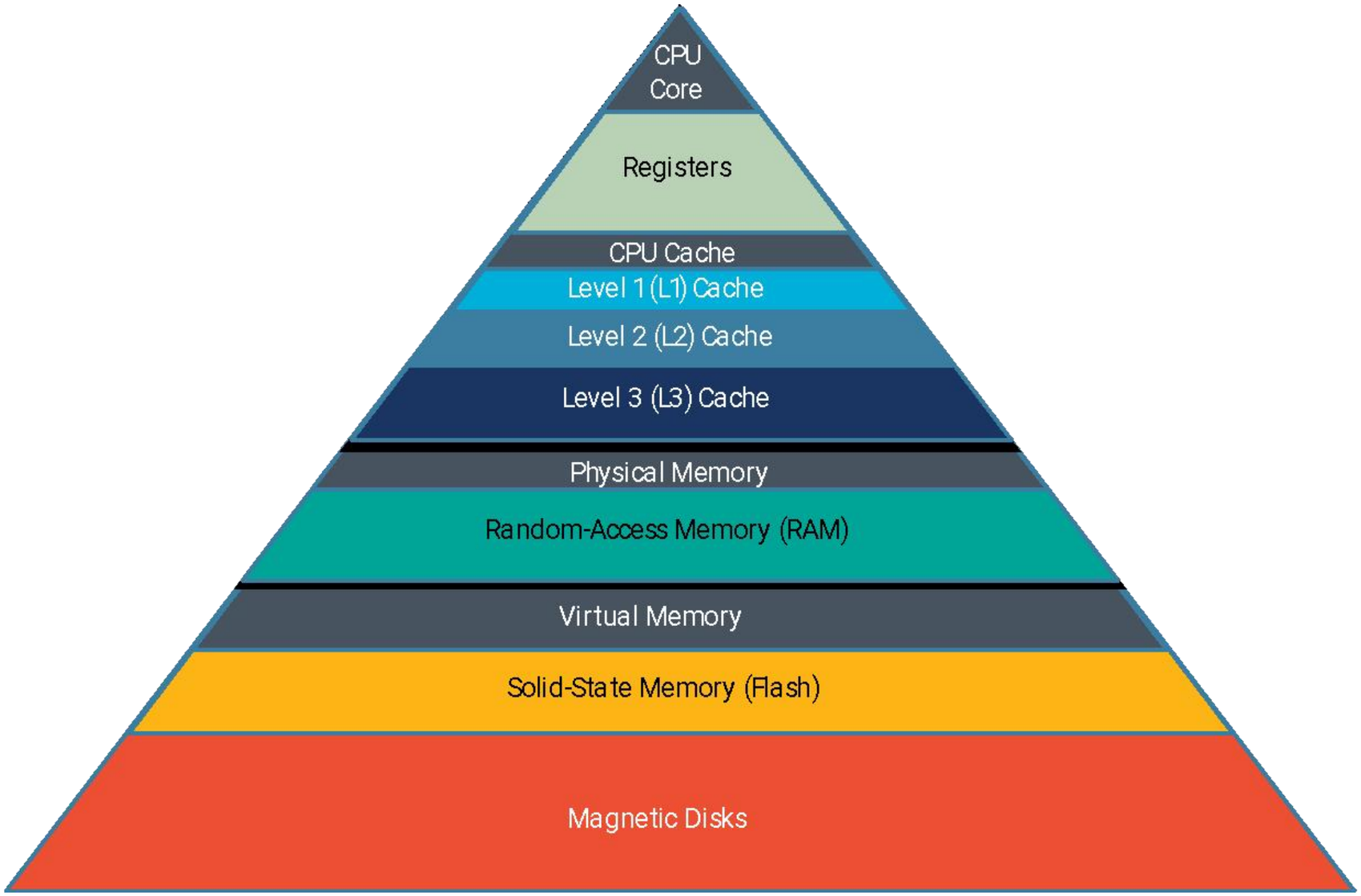


Bane of Locality: Pointer Chasing

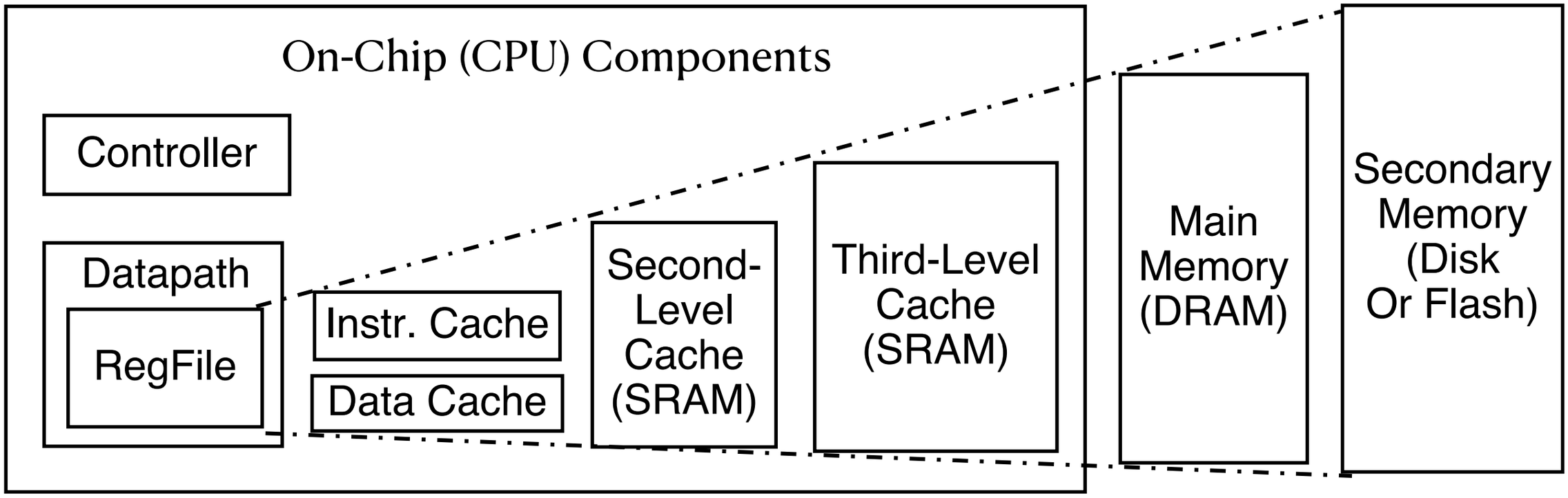
- Special data structures: linked list, tree, etc.
  - Easy to append onto and manipulate...
- But they have horrid locality preferences
  - Every time you follow a pointer it is to an unrelated location: No spacial reuse from previous pointers
  - And if you don't chase the pointers again you don't get temporal reuse either

Cache Philosophy

- The **memory hierarchy** presents the processor with the illusion of a **very large and fast memory** by taking advantages of **locality**.



Typical Memory Hierarchy

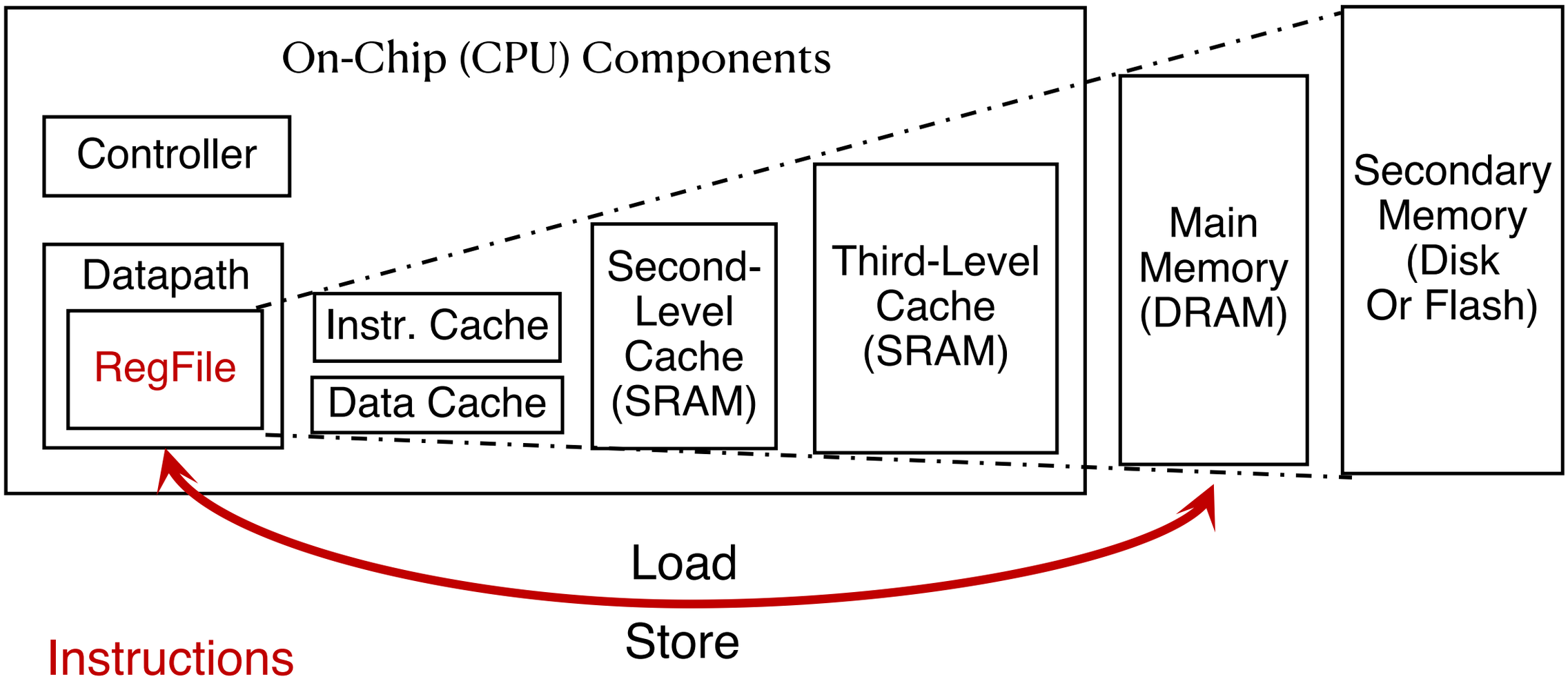


Speed (cycles):	½'s	1's	10's	100's	1,000,000's
Size (bytes):	100's	10K's	M's	G's	T's

Cost/bit: highest ⟷ lowest

- Principle of locality + memory hierarchy presents programmer with ≈ as much memory as is available in the *cheapest* technology at the ≈ speed offered by the *fastest* technology

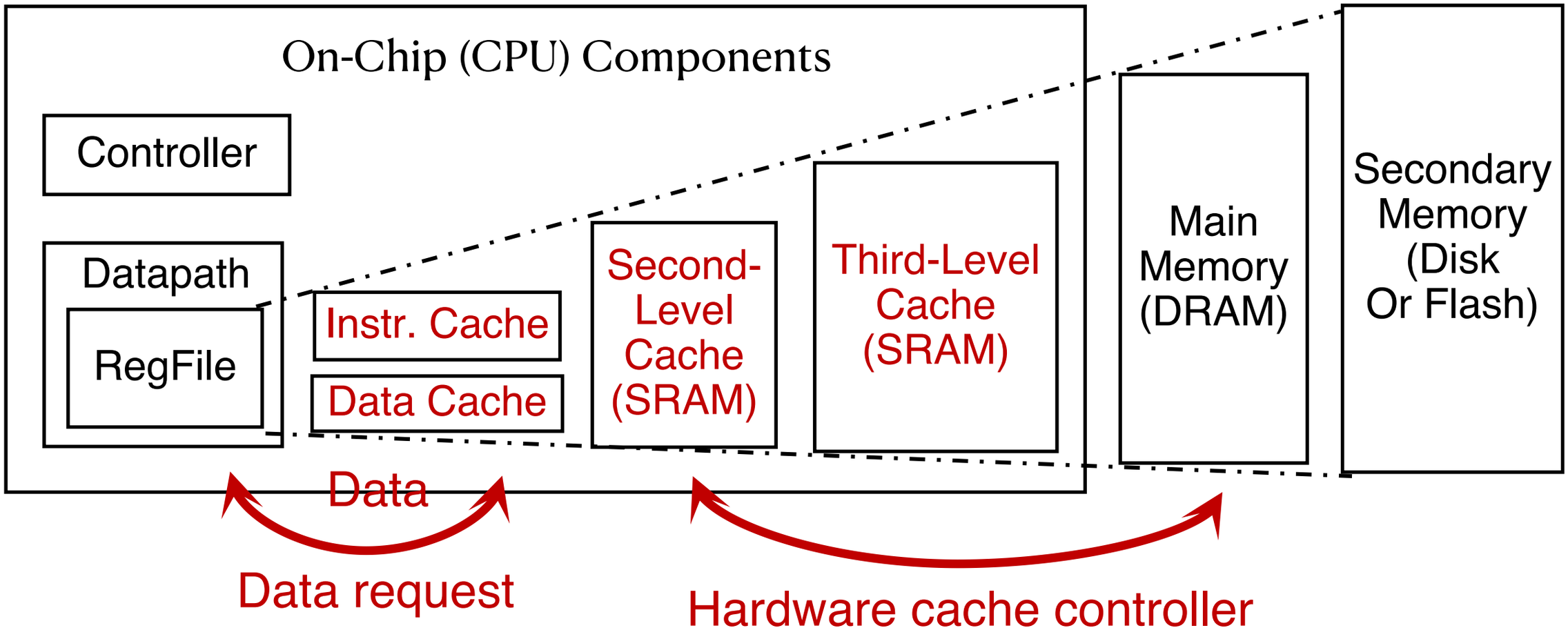
Memory Hierarchy Management



Instructions

- Either by assembly programers or generated by a compiler;
- Does not define how it is achieved.

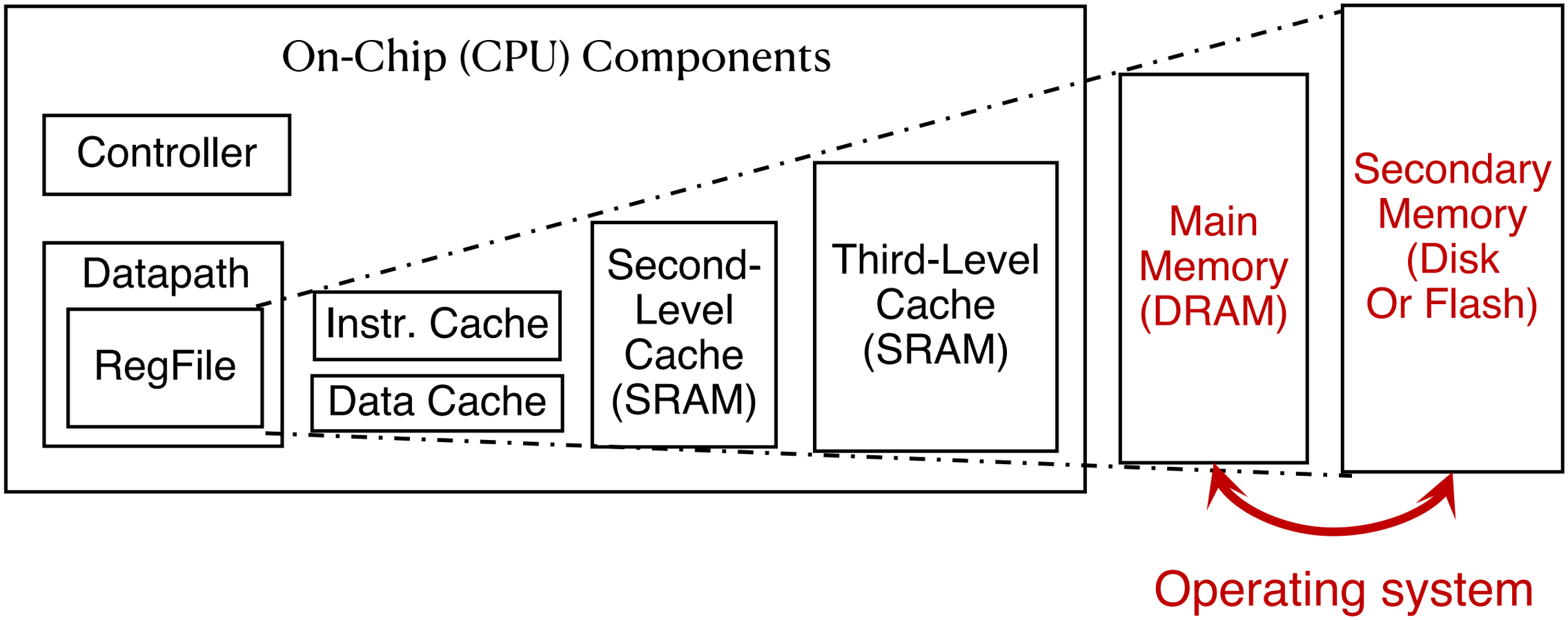
Memory Hierarchy Management



- With cache, the datapath/core does not directly access the main memory;
- Instead the core asks the caches for data with improved speed;
- A hardware cache controller is deviced to provide the desired data

(with various strategies that will be covered in future lectures).

Memory Hierarchy Management



- By the operating system (virtual memory)
- Virtual to physical address mapping assisted by the hardware (‘translation lookaside buffer’ or TLB, also a cache)
- By the programmer (files)

Memory with/without Cache Example

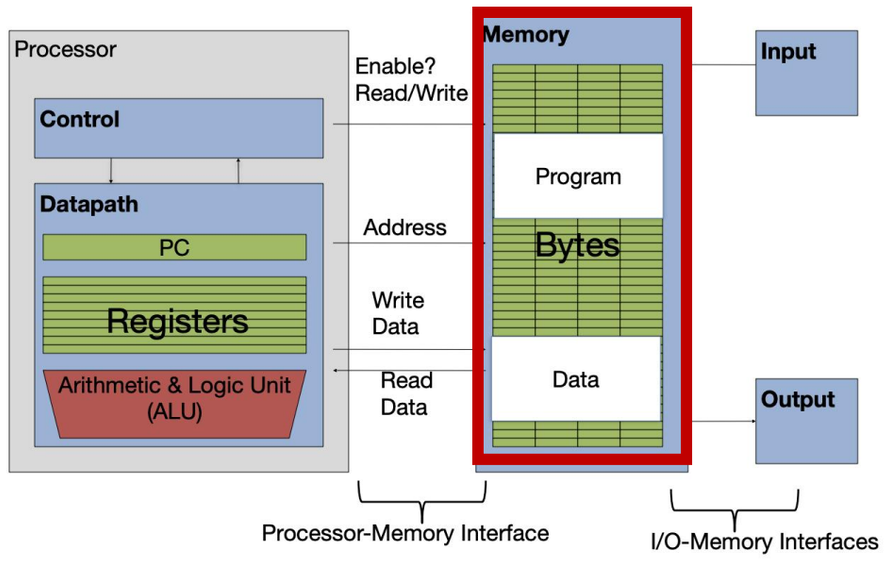
Load word instruction:
`lw t0 0(t1)`

`t0`	`1234`
`t1`	`0x12F0`

`Memory[0x12F0] = 1234`

Memory access without cache:

1. Processor issues address `0x12F0` to memory
2. Memory reads 1234 @ address `0x12F0`
3. Memory sends 1234 to Processor
4. Processor loads 1234 into register `t0`



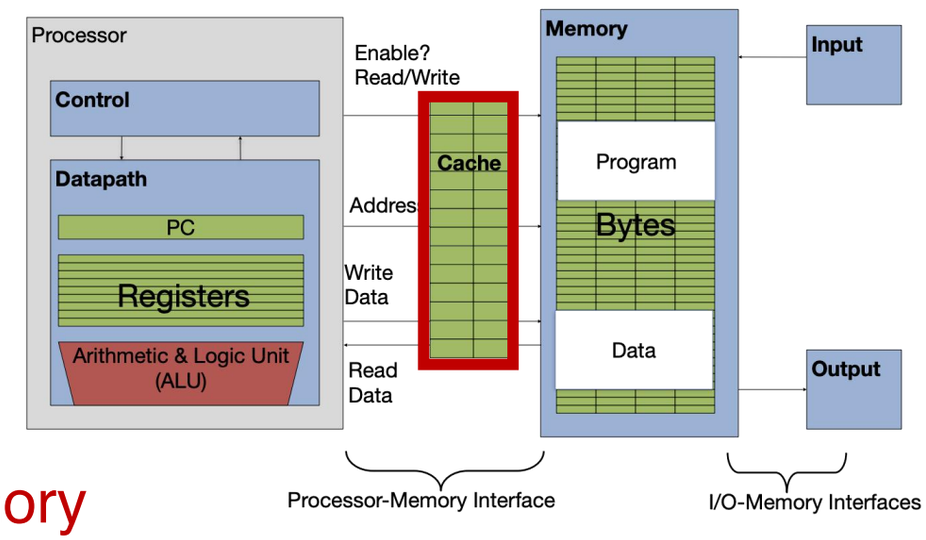
Memory with/without Cache Example

Load word instruction:
`lw t0 0(t1)`

`t0`	`1234`
`t1`	`0x12F0`

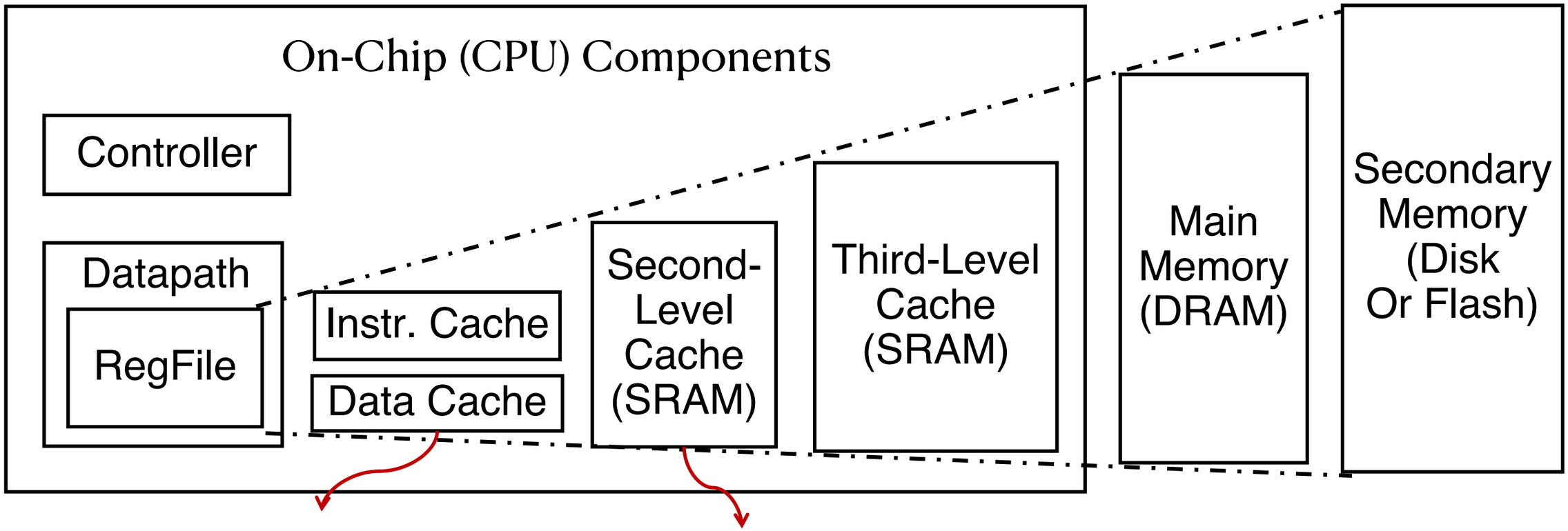
`Memory[0x12F0] = 1234`

Memory access with cache:



1. Processor issues address `0x12F0` to memory
2. Cache checks if data @address `0x12F0` is in it
   - if it is in the cache, cache hit and read 1234
   - if not matched, called cache miss and
     - Cache sends address to `0x12F0` the memory
     - Memory read address `0x12F0` and send 1234 to cache
     - Due to limited size, cache replace some data with 1234
3. Cache sends 1234 to Processor
4. Processor loads 1234 into register `t0`

Typical Values



- **L1 cache**
  - size: tens of KB
  - hit time: complete in one clock cycle
  - miss rates: 1-5%
- **L2 cache**
  - size: hundreds of KB
  - hit time: few clock cycles
  - miss rates: 10-20%

- **The L2 miss rate is the fraction of L1 misses that also miss in L2.**
  - Why so high? (more later)

Detailed Considerations

Load word instruction:
`lw t0 0(t1)`

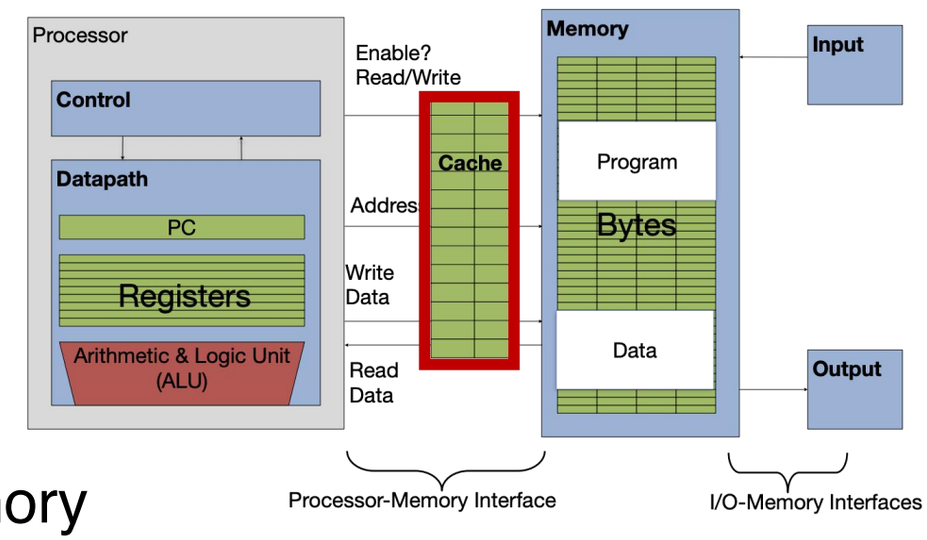
`t0`	`1234`
`t1`	`0x12F0`

`Memory[0x12F0] = 1234`

How?

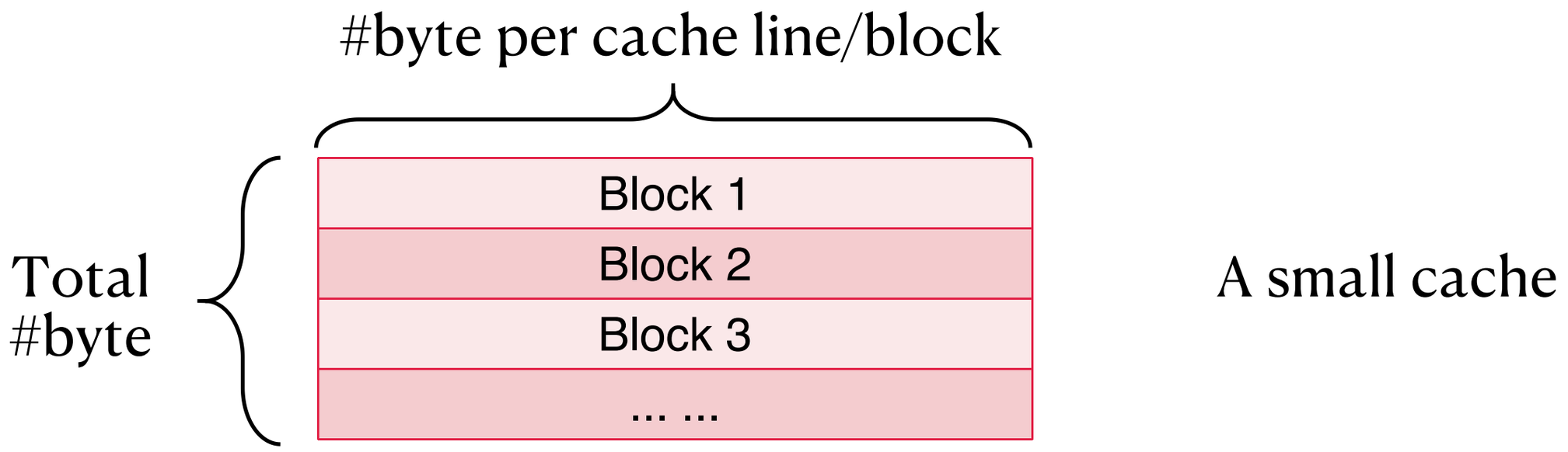
Memory access w

1. Processor issues address `0x12F0` to memory
2. Cache checks if data @address `0x12F0` is in it
   - if it is in the cache, cache hit and read `1234`
   - if not matched, called cache miss and
     - Cache sends address to `0x12F0` the memory
     - Memory read address `0x12F0` and send `1234` to cache
     - Due to limited size, cache replace some data with `1234`
3. Cache sends `1234` to Processor
4. Processor loads `1234` into register `t0`



Cache Terminology

- Cache line/block: a single entry in cache
- Cache line/block size: #byte per cache line/block
- Capacity: total #byte that can be stored in a cache



Cache “Tag”

- Need a wag to tell if the cache have copy of location in memory so that can decide on hit or miss;
- On cache miss, put memory address of block in “tag address” of cache block;
- Previous example: address used as tag.

Load word instruction:

```
lw t0 0(t1)
```

t0	1234
t1	0x12F0

`Memory[0x12F0] = 1234`

A small cache

Address	DATA
0x1100	0
0x12F0	1234
0x1000	110
0x4000	100

Example: Anatomy of a 16 Byte Cache

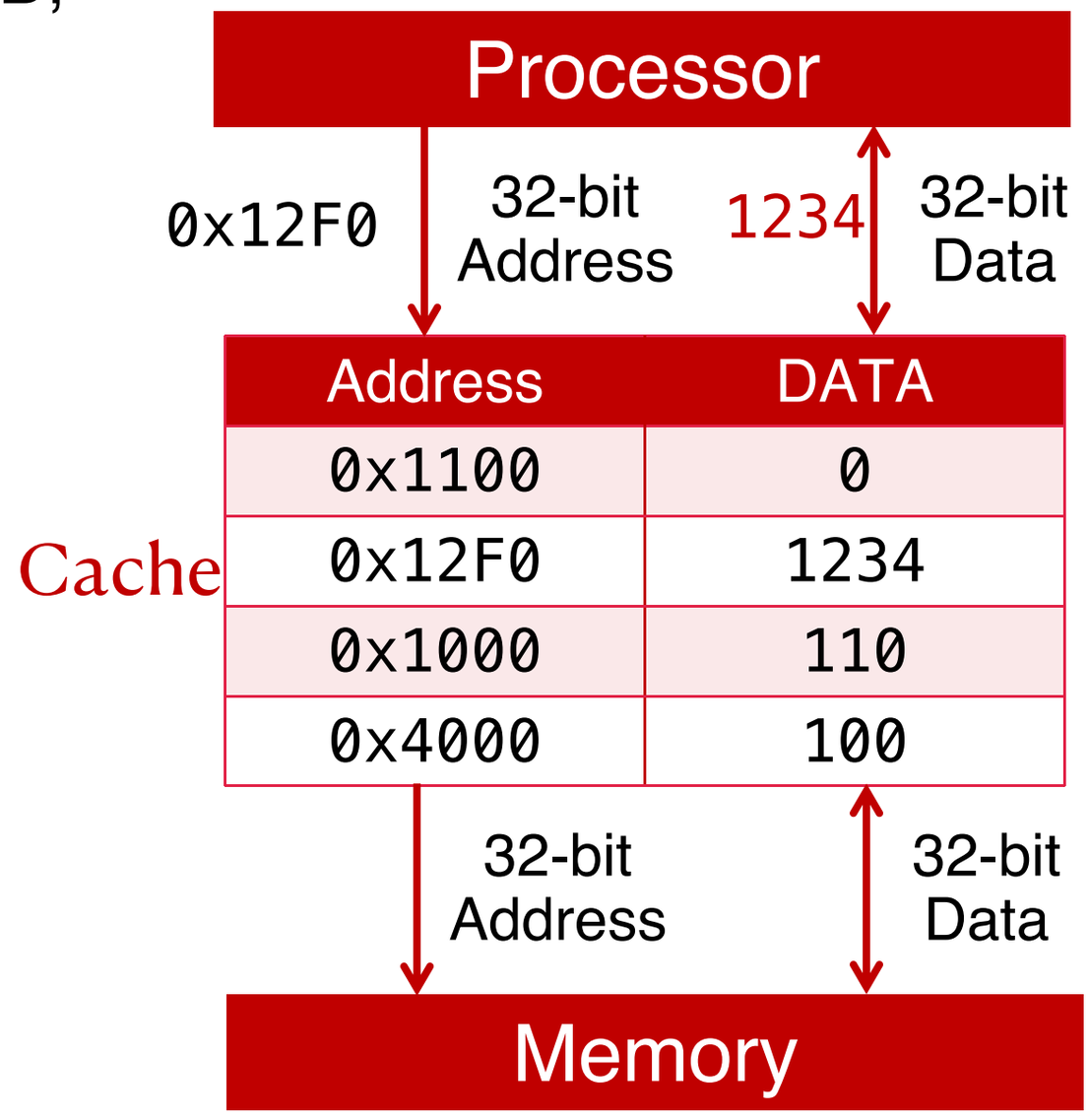
- Cache capacity: 16 B; Block size: 4 B;
- Thus 4 cache blocks;

Load word instruction:
`lw t0 0(t1)`

t0	1234
t1	0x12F0

`Memory[0x12F0] = 1234`

- Compare address: HIT!
- Fetch the data from cache



Example: Anatomy of a 16 Byte Cache

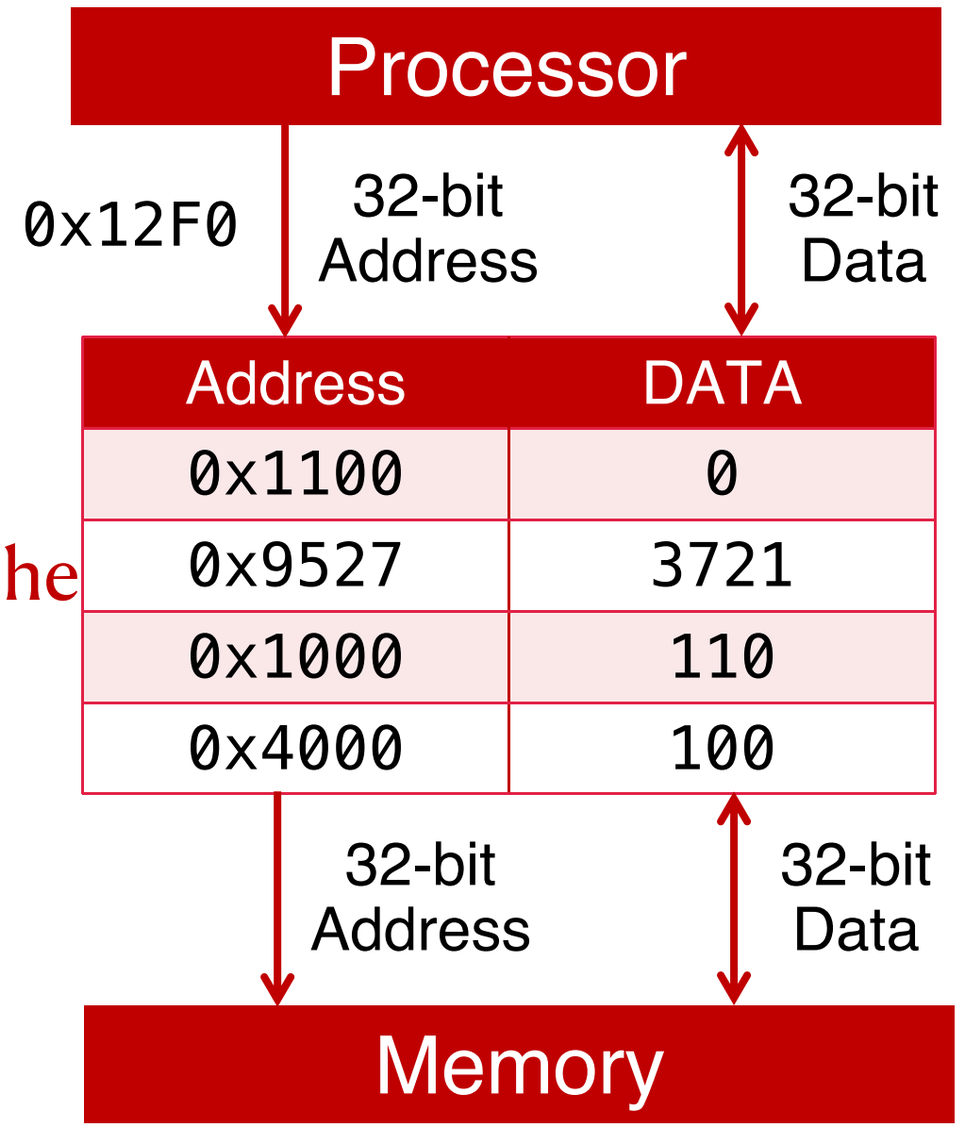
- Cache capacity: 16 B; Block size: 4 B;
- Thus 4 cache blocks;

Load word instruction:
`lw t0 0(t1)`

t0	1234
t1	0x12F0

`Memory[0x12F0] = 1234`

- Compare address: MISS!
- Replace the data in the cache
  - Must “evict” one resident block to make room (Policy covered in later lectures)



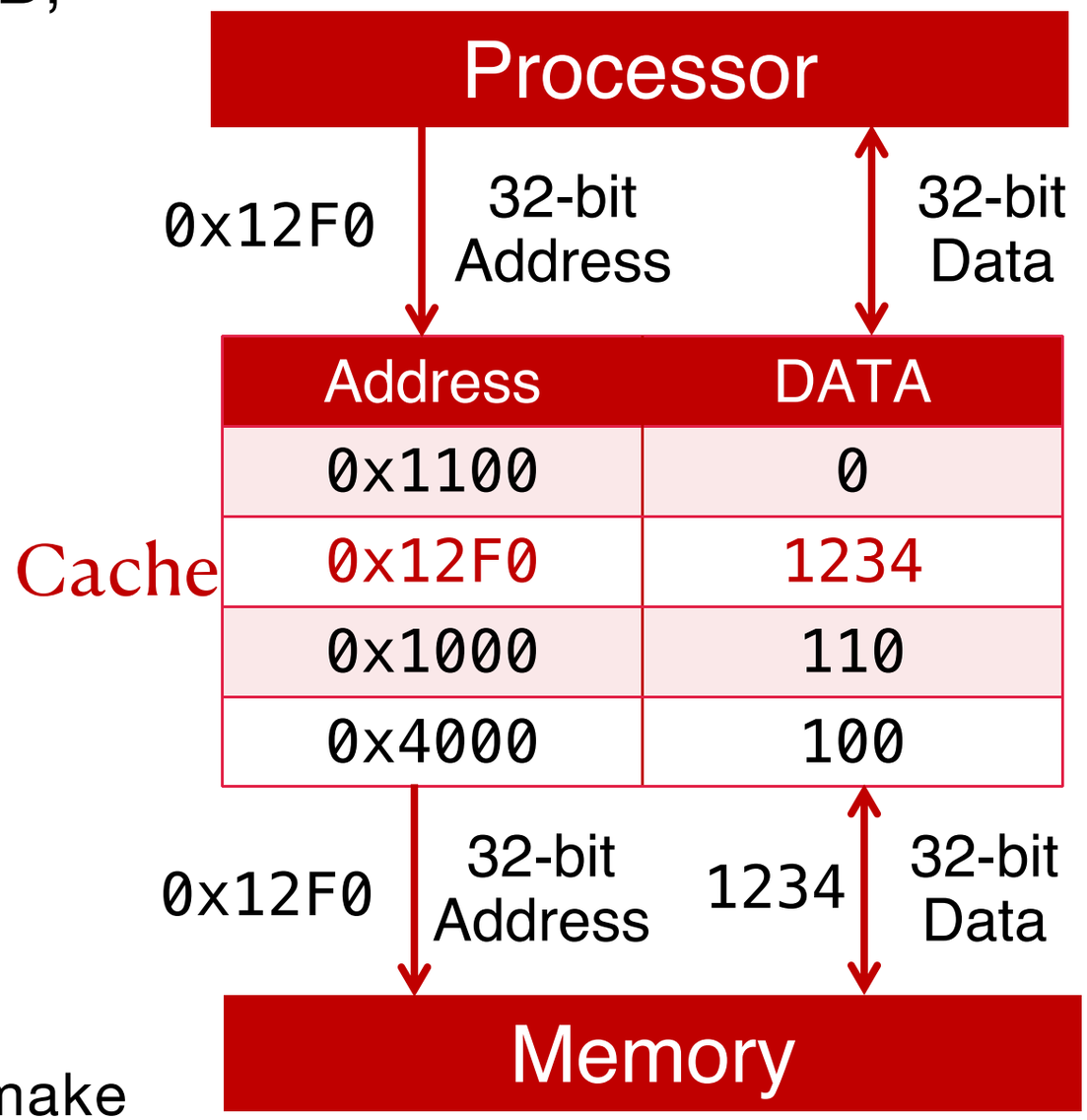
Cache Replacement

- Cache capacity: 16 B; Block size: 4 B;
- Thus 4 cache blocks;

Load word instruction:
`lw t0 0(t1)`

t0	1234
t1	0x12F0

`Memory[0x12F0] = 1234`



- Compare address: MISS!
- Replace the data in the cache
  - Must "evict" one resident block to make room (Policy covered in later lectures)
  - Replace "victim" with new memory block at address 0x12F0
- Fetch 1234 from cache

Summary

- Cache is added to improve the read/write efficiency;
- It is transparent to the programmer while the management is automatically done by a cache controller;
- It exploits locality to make the memory looks big and fast.