# CSE261: Computer Architecture

## 15. Memory Hierarchy (1): Introduction

Seongil Wi

Department of Computer Science and Engineering

# HW2



Due date: **11/26, 11:59PM**

# Where Are We?

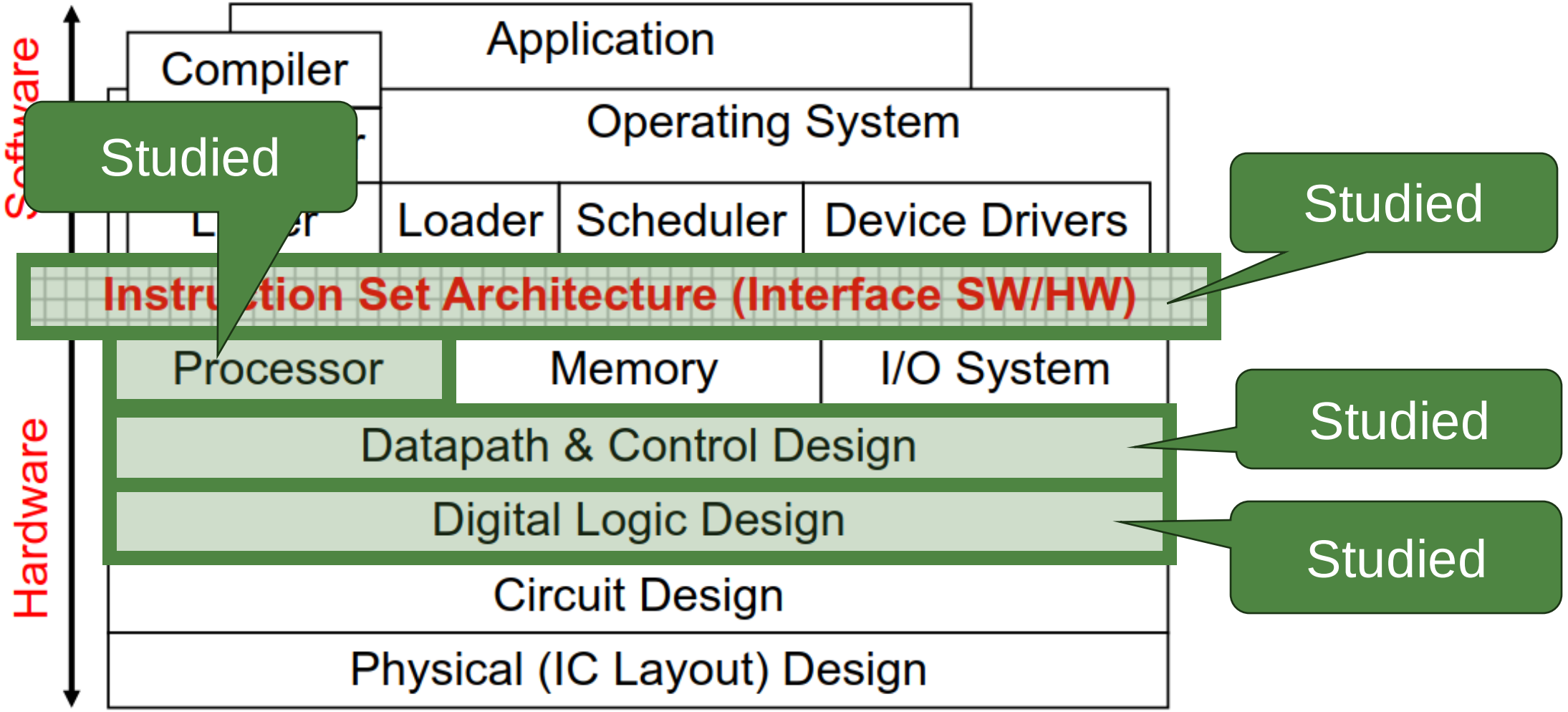



Image from: https://medium.com/@shb8086/tutorial-series-isa-fa9648c5e9e5

# Today's Topic – Memory

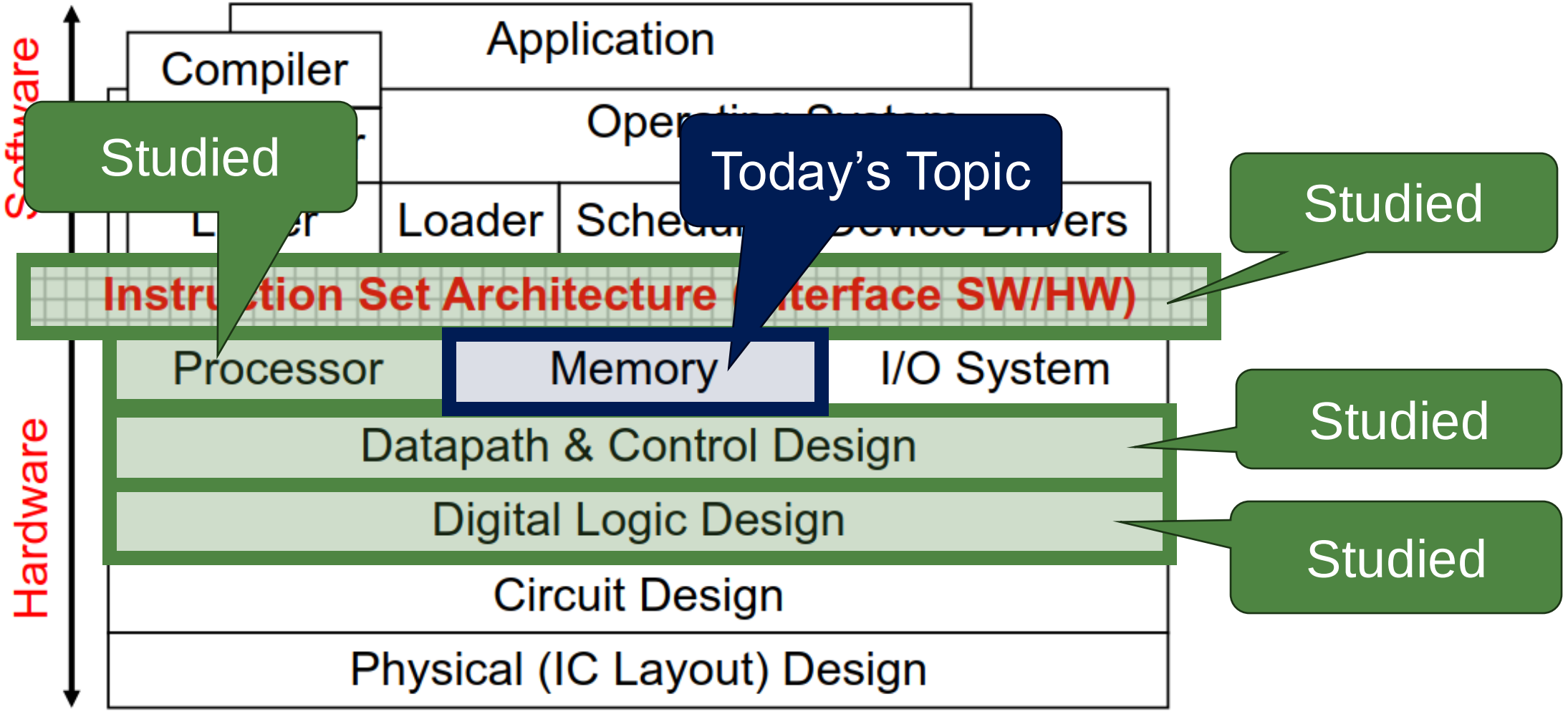



Image from: https://medium.com/@shb8086/tutorial-series-isa-fa9648c5e9e5

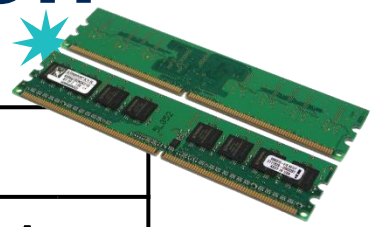
# Today's Topic



## Memory

**Accessing memory is one of the biggest performance bottlenecks ☹**

# Recap: Program Execution

Control (brain)

01010010

Datapath

Processor (CPU) (e.g., Intel I7)

*Load/Store the Data*

*3. Execute the instruction*

*2. Fetch the instruction (One-by-one)*

| |
|---|
| |
| Data A |
| |
| |
| |
| |
| |
| 01010010 |
| 00110110 |
| |
| |
| |

Main memory

*1. Load the Program A*

01010010
00110110
Program A

11001110
01000110
Program B

01101101
11010111
Program C

Hard disk

# Performance Bottleneck

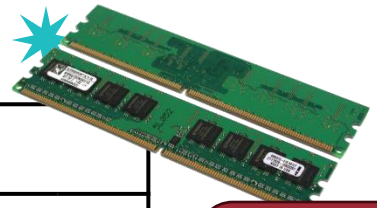

Performance bottleneck ☹

Control (brain)

01010010

Datapath

Processor (CPU) (e.g., Intel I7)



Data A

01010010

00110110

Main memory

Performance bottleneck ☹

01010010
00110110
Program A

11001110
01000110
Program B

01101101
11010111
Program C

Hard disk

# Recap: Storage Inside in CPU


Control (brain)

Register 1

Register 2

01010010

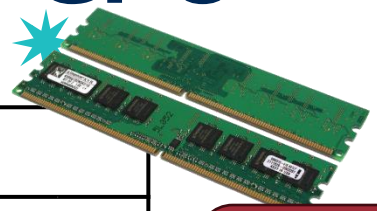
Datapath

(CPU)

(e.g., Intel I7)

Performance bottleneck ☹

Fast but small



Data A

01010010

00110110

Main memory

Performance bottleneck ☹

Large but slow

01010010
00110110

Program A

11001110
0100011

01101101
11010111

Program C

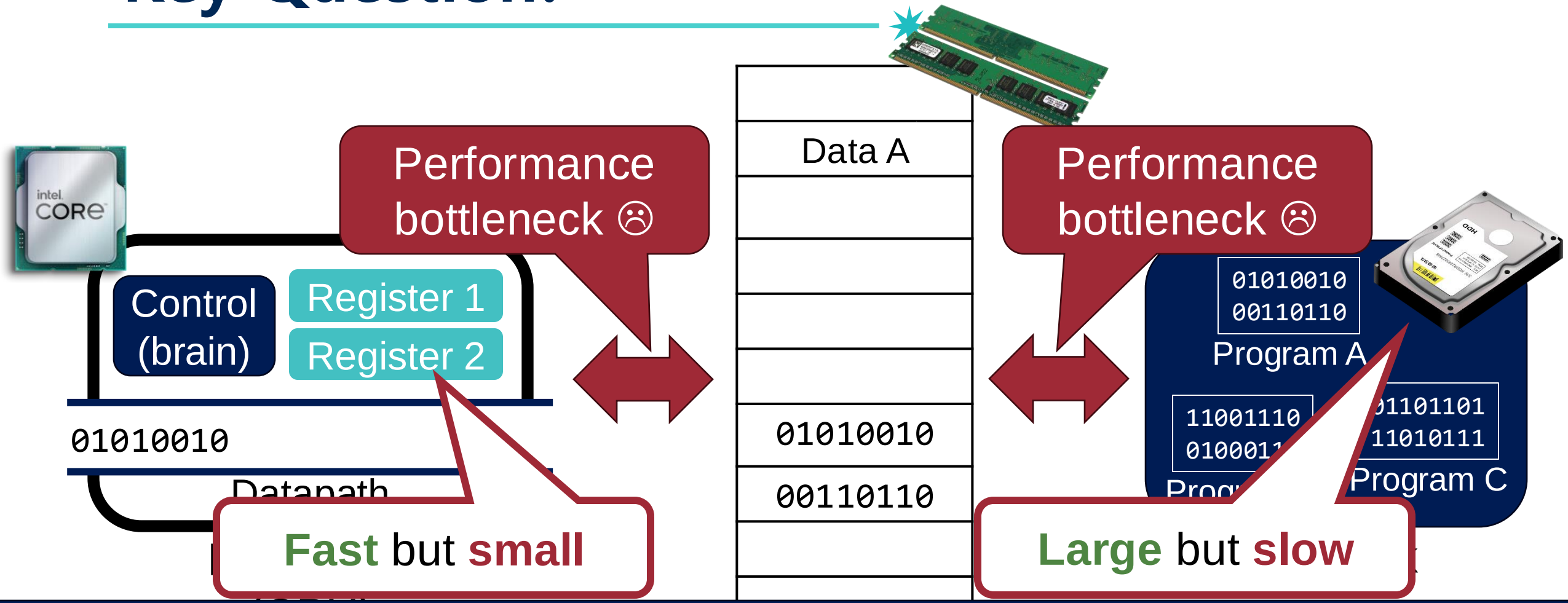
# Recall that 'Smaller is Faster!'



The faster memories are *more expensive per bit* than the slower memories and thus are smaller!

| Memory | Typical access time | $ per GB (in 2020) |
|---|---|---|
| Static RAM (SRAM) | 0.5ns – 2.5ns | $2,000 – $5,000 |
| Dynamic RAM (DRAM) | 50ns – 70ns | $20 – $75 |
| Magnetic Disk | 5ms – 20ms | $0.20 – $2 |

# Key Question!




There is a *conflict* having large and fast memory ☹

**How can we achieve *large and fast memory*?**

# Solution: Memory Hierarchy



A structure that uses multiple levels of memories

# Exploiting Hierarchy Example: Book Storage

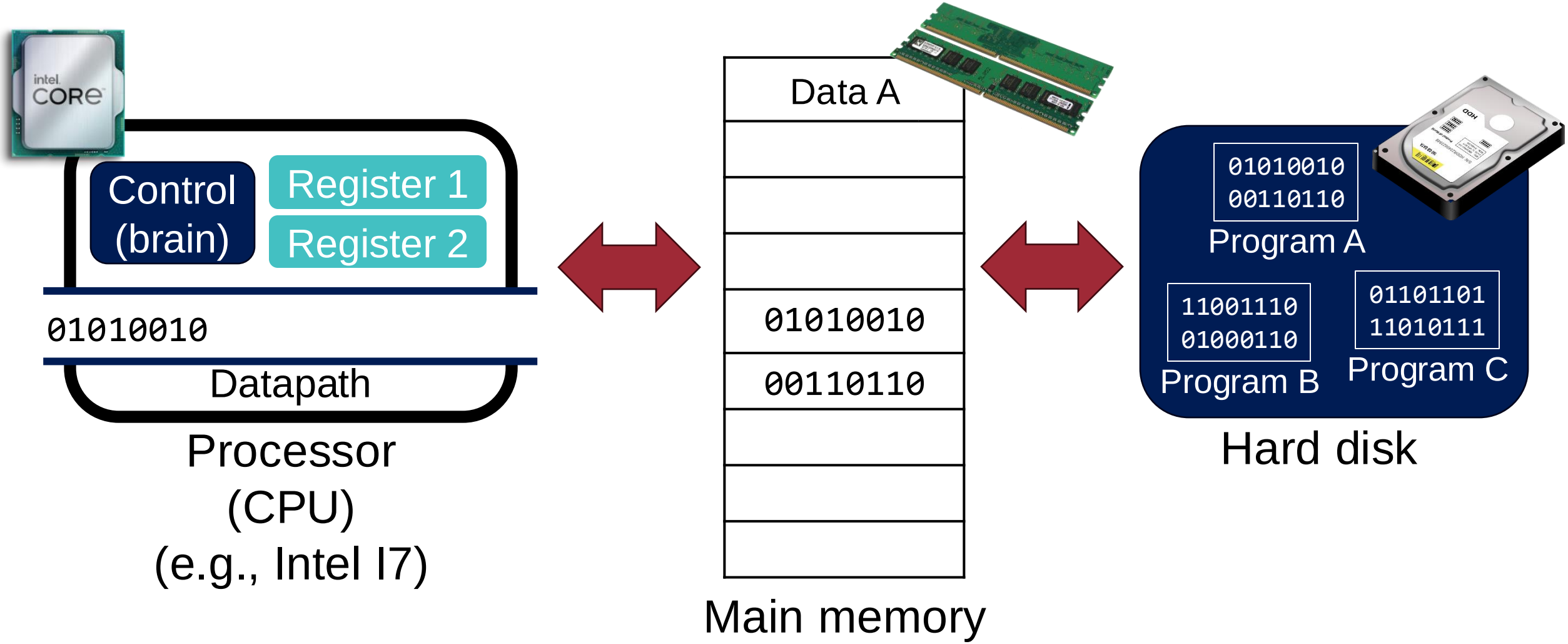


# Exploiting Hierarchy Example: Book Storage


How about Level 3?

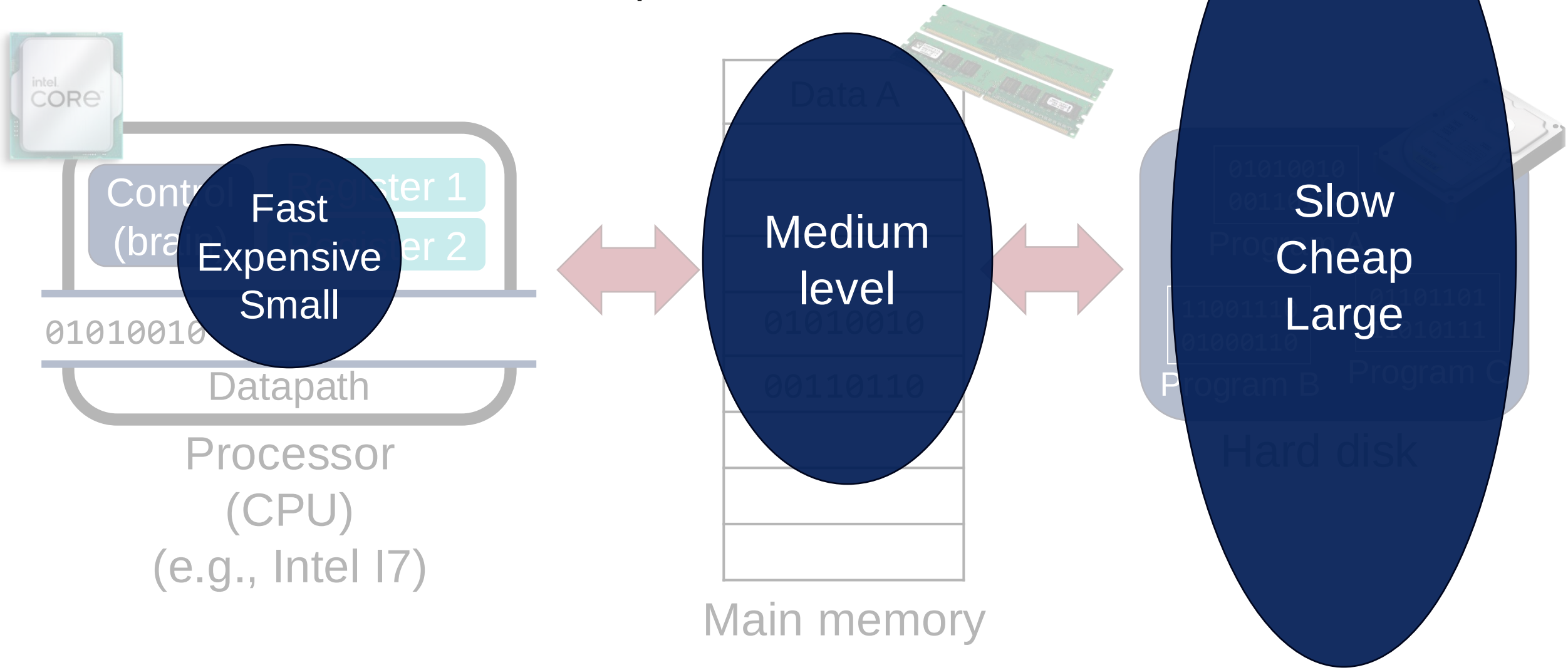
# Solution: Memory Hierarchy

A structure that uses multiple levels of memories



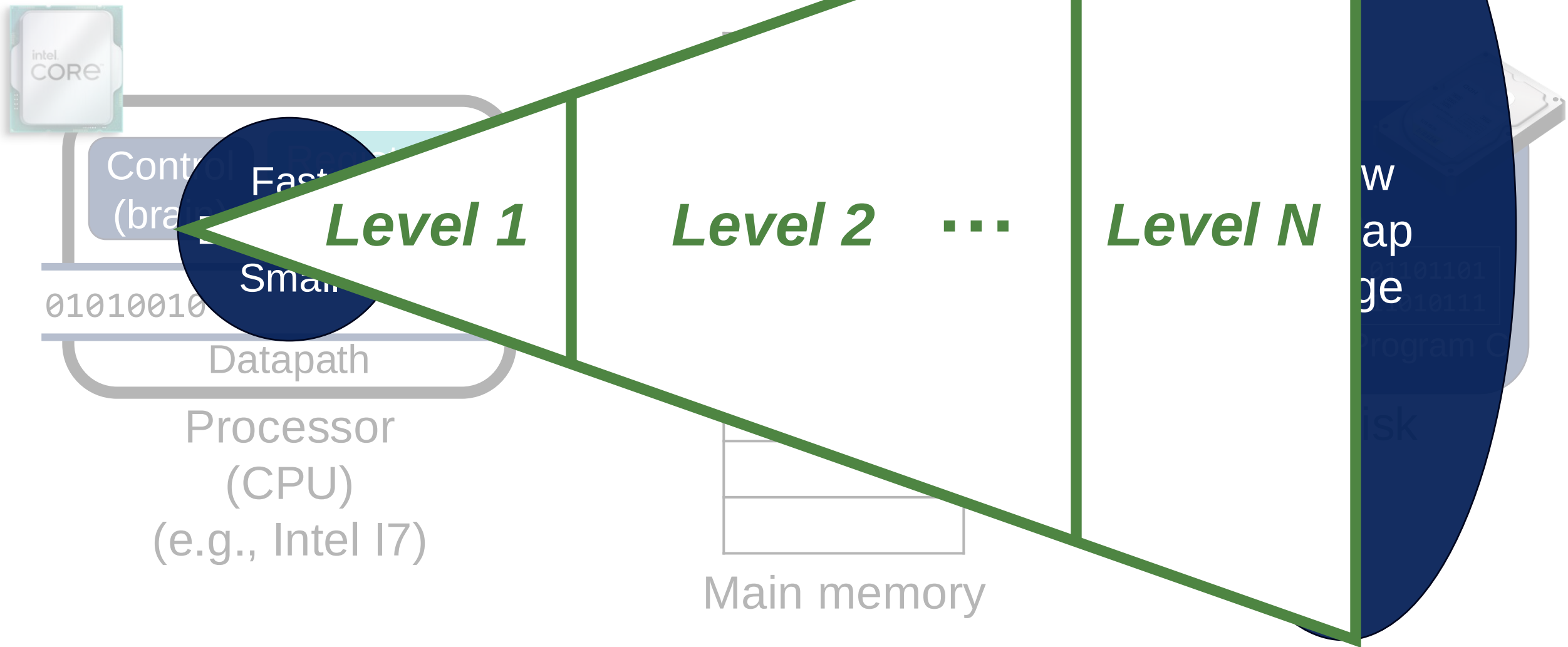
# Solution: Memory Hierarchy

A structure that uses multiple levels of memories



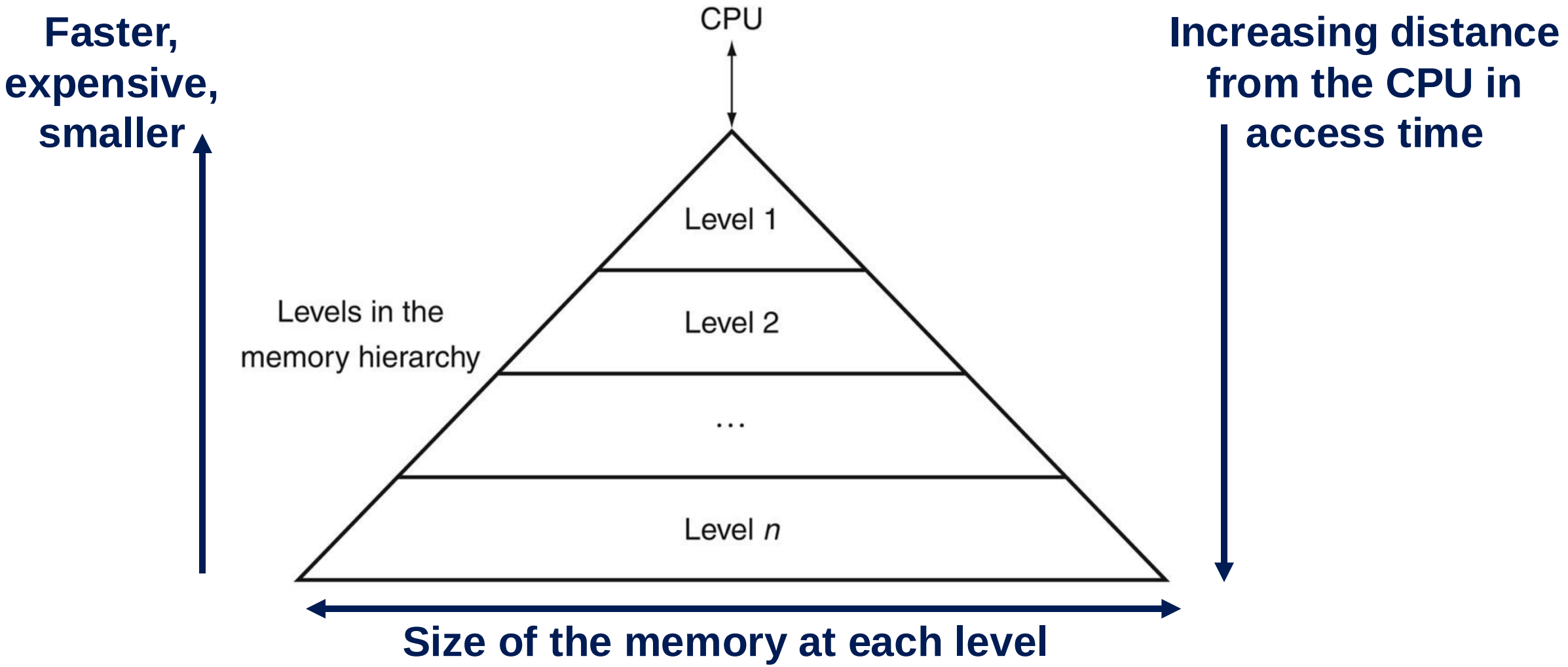
# Solution: Memory Hierarchy

A structure that uses multiple levels of mem



# Solution: Memory Hierarchy

A structure that uses multiple levels of memories



# Memory Hierarchy Details


**Register 1**

**Register 2**

**On-chip cache** (L1 cache)

Processor (CPU) (e.g., Intel I7)

**SRAM** (L2 cache)

**DRAM** (Main memory)

Main memory

**Secondary storage** (Hard disk)

# Why does the Memory Hierarchy Work?



Because of the principle of ***locality!***

# Important Principle: Locality (지역성)



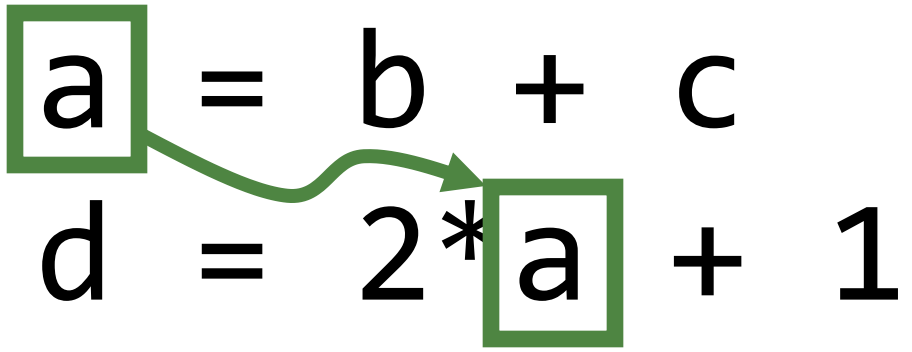
The tendency to access the same set of memory locations *repetitively* over a short period of time

1. **Temporal locality** (locality in time)

2. **Spatial locality** (locality in space)

# Temporal Locality Example






**Temporal locality**: items accessed recently are likely to be accessed again soon!

# Important Principle: Locality (지역성)



The tendency to access the same set of memory locations *repetitively* over a short period of time

1. **Temporal locality** (locality in time)
   - If an item is referenced, *the same item* will tend to be referenced again soon

2. **Spatial locality** (locality in space)

# Spatial Locality Example



```
for (i=0; i<10; i++)
    sum = sum + a[i]
```

**Spatial locality**: Items near those accessed recently are likely to be accessed soon

# Important Principle: Locality (지역성)



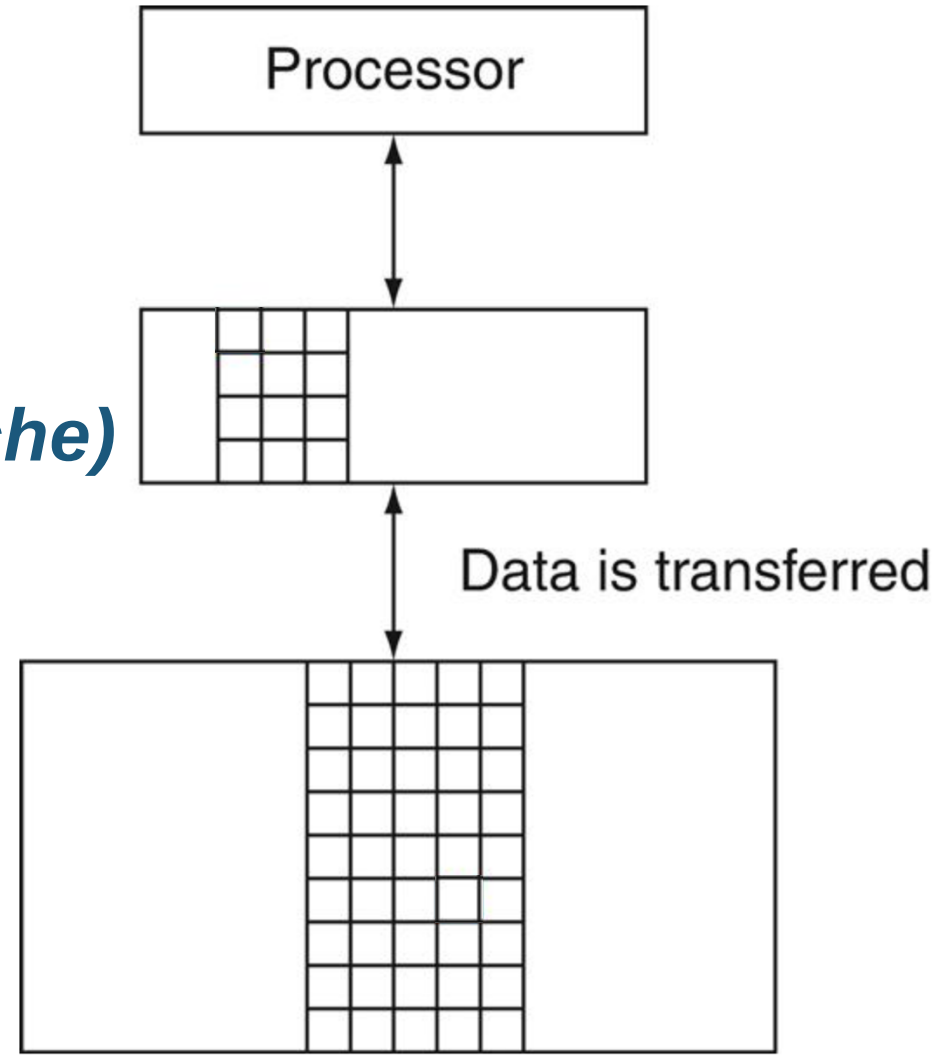
The tendency to access the same set of memory locations *repetitively* over a short period of time

1. **Temporal locality** (locality in time)
   - If an item is referenced, *the same item* will tend to be referenced again soon

2. **Spatial locality** (locality in space)
   - If an item is referenced, *nearby items* will tend to be referenced soon

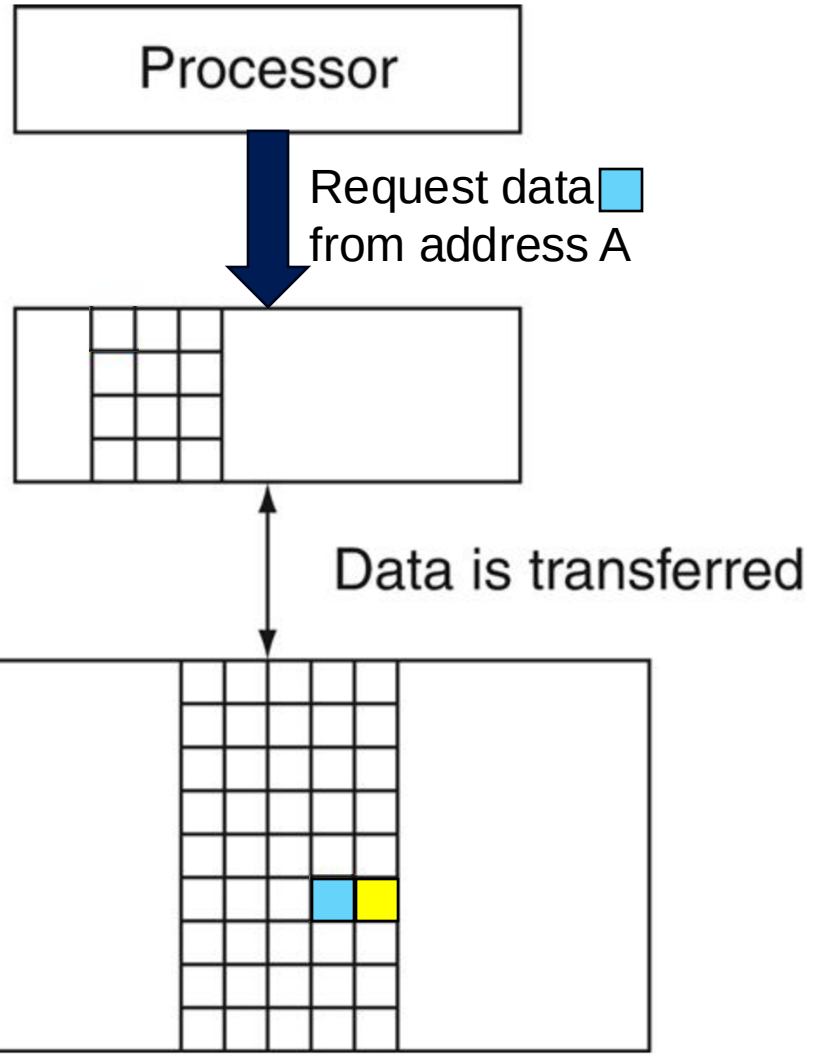
# Locality Example: 1st Access

- Assumption: two levels (*upper*, *lower*)
  - Each pair of levels in the memory hierarchy can be thought of as having an *upper* and *lower level*



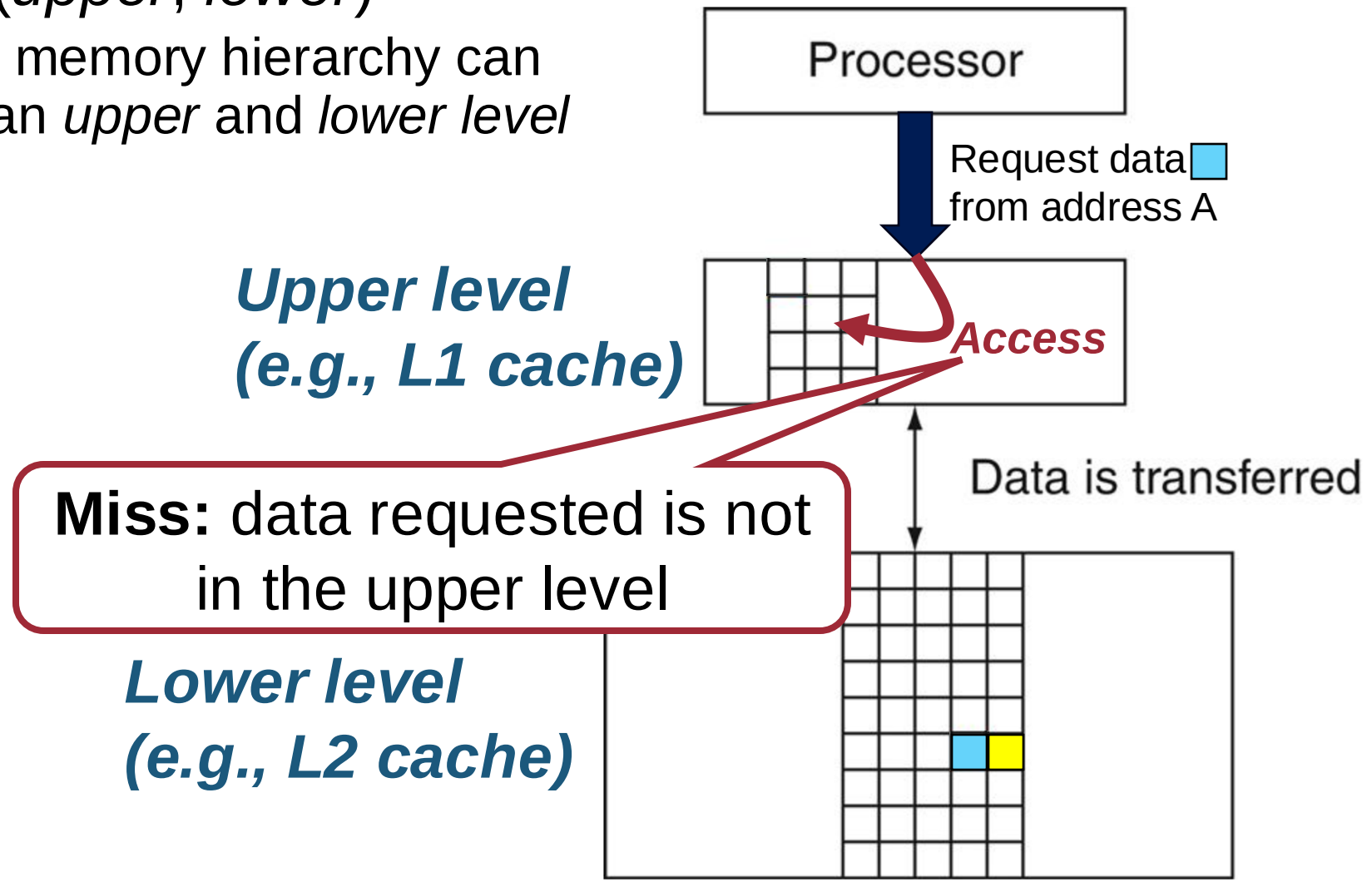
# Locality Example: 1st Access

- Assumption: two levels (*upper*, *lower*)
  - Each pair of levels in the memory hierarchy can be thought of as having an *upper* and *lower level*



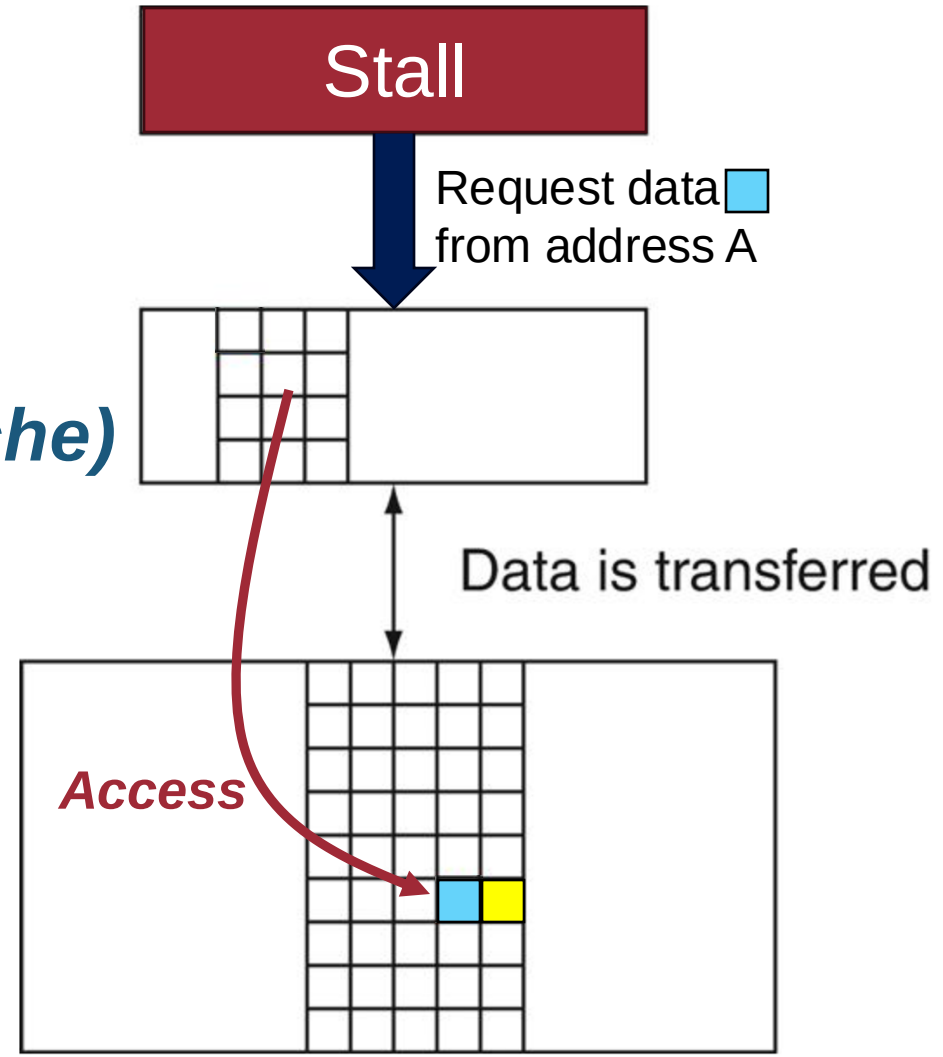
# Locality Example: 1st Access

- Assumption: two levels (*upper*, *lower*)
  - Each pair of levels in the memory hierarchy can be thought of as having an *upper* and *lower level*



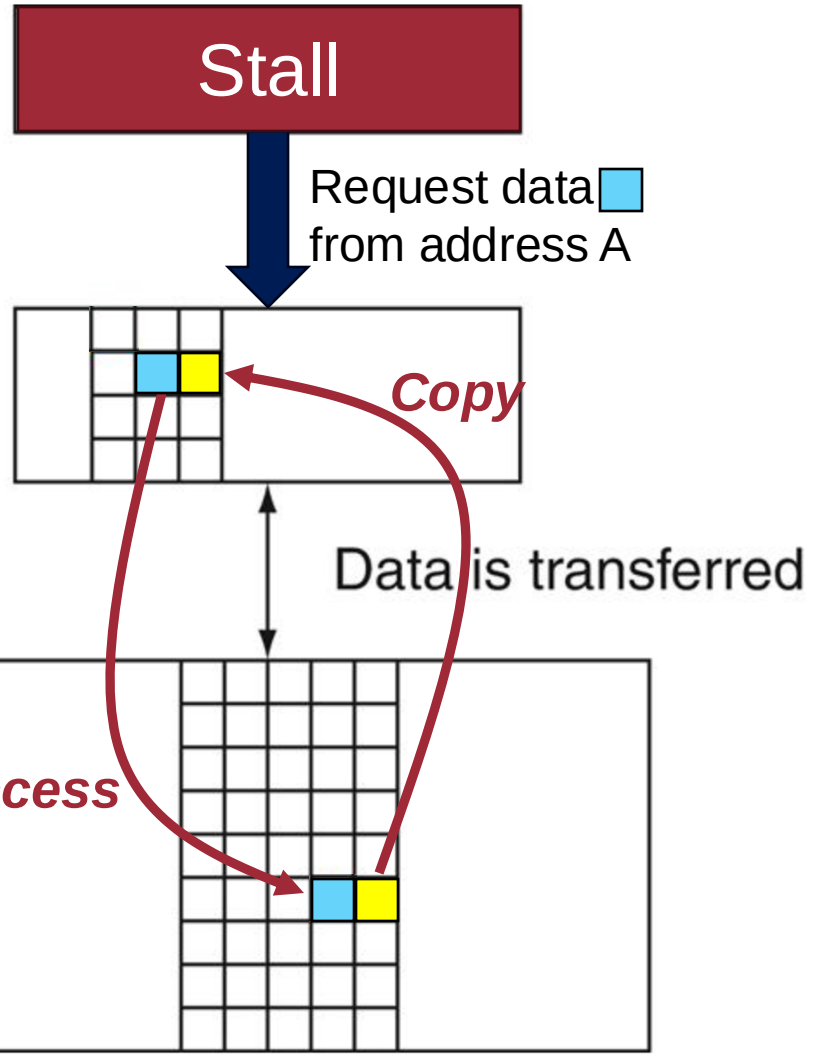
# Locality Example: 1st Access

- Assumption: two levels (*upper*, *lower*)
  - Each pair of levels in the memory hierarchy can be thought of as having an *upper* and *lower level*



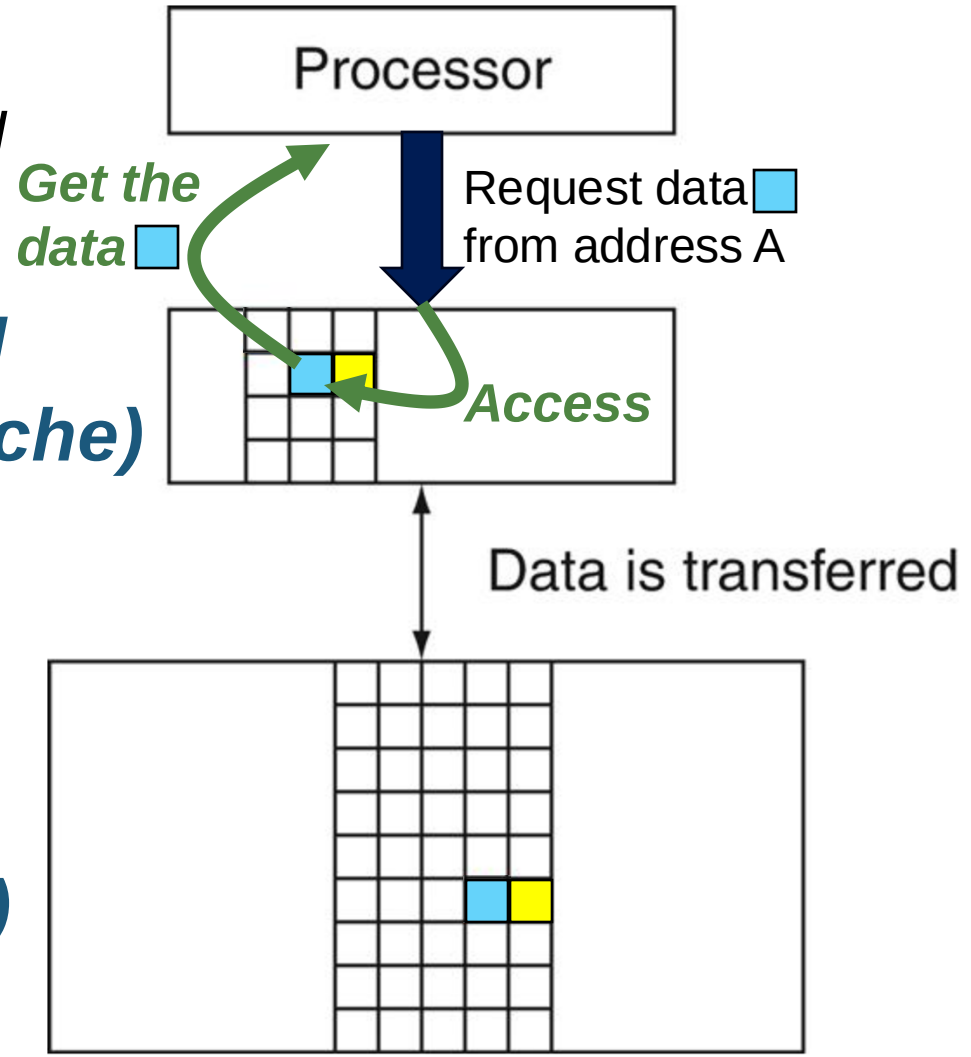
# Locality Example: 1st Access

- Assumption: two levels (*upper*, *lower*)
  - Each pair of levels in the memory hierarchy can be thought of as having an *upper* and *lower level*



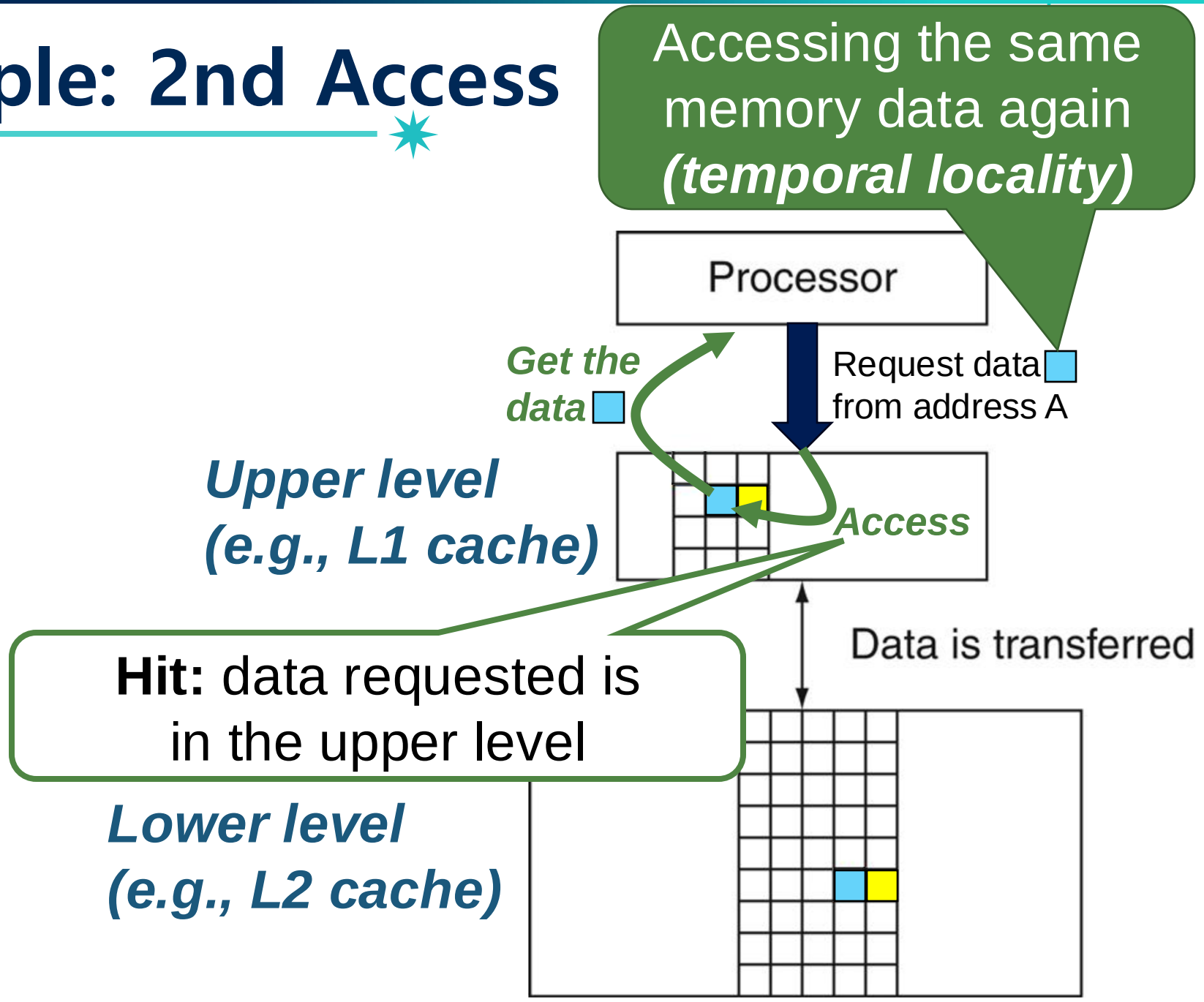
# Locality Example: 1st Access

- Assumption: two levels (*upper*, *lower*)
  - Each pair of levels in the memory hierarchy can be thought of as having an *upper* and *lower level*

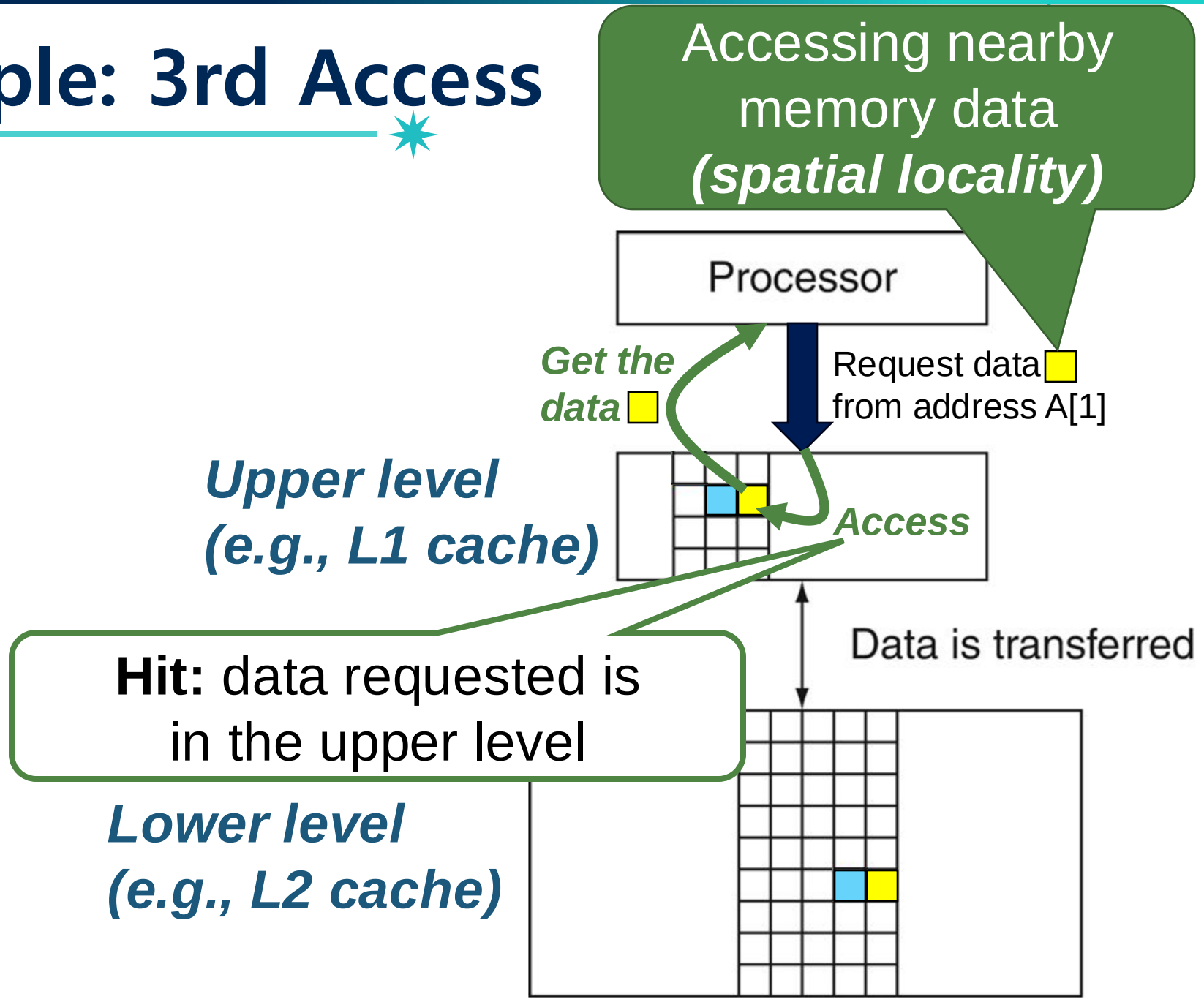


When a miss occurs, performance degrades significantly ☹

# Locality Example: 2nd Access



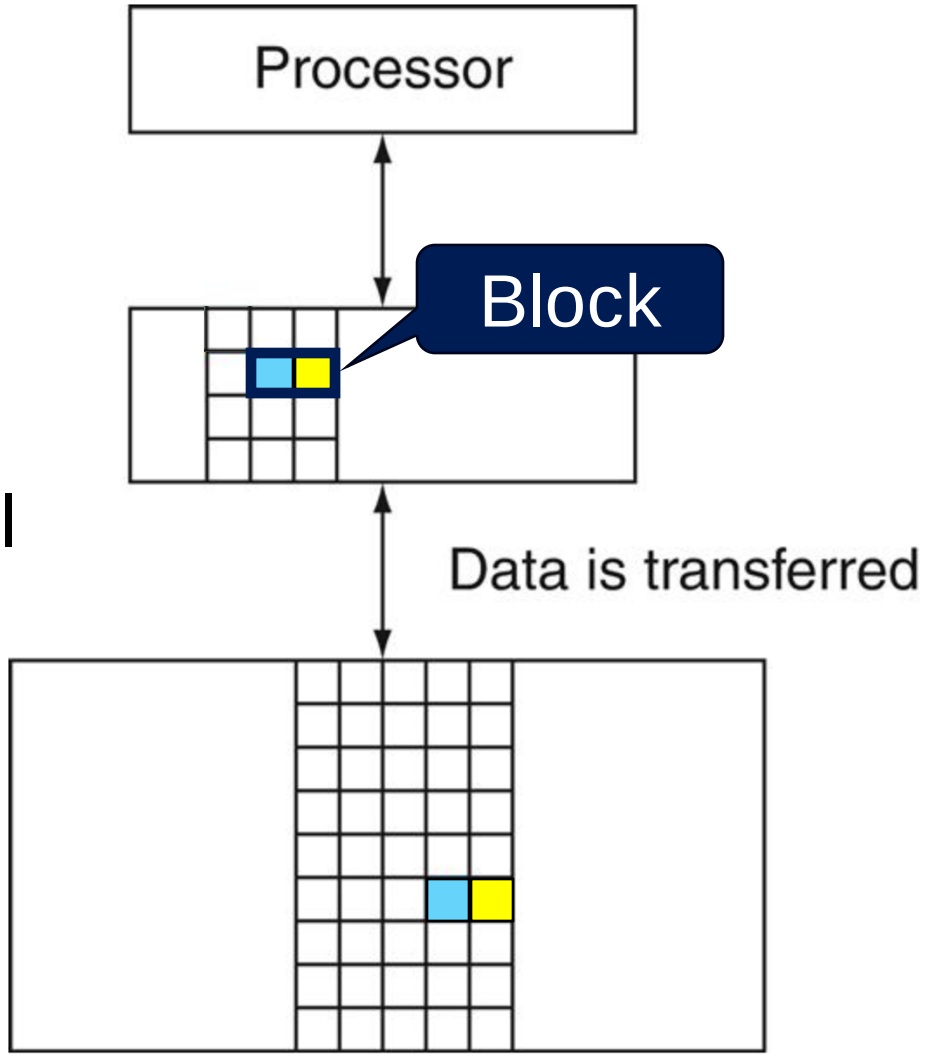
# Locality Example: 3rd Access



When a miss occurs, performance degrades significantly ☹
However, once a miss happens, many hits can be expected due to ***locality***, leading to performance improvement!

# Terms

- Block (a.k.a., line): unit of copying
  - Several words in cache memory
- **Hit**: data requested is in the upper level
  - Hit ratio: hits/accesses
- **Miss**: data requested is not in the upper level
  - Block copied from lower leve
  - Miss penalty: time taken to resolve miss
  - Miss ratio: misses/accesses
    = 1 – hit ratio

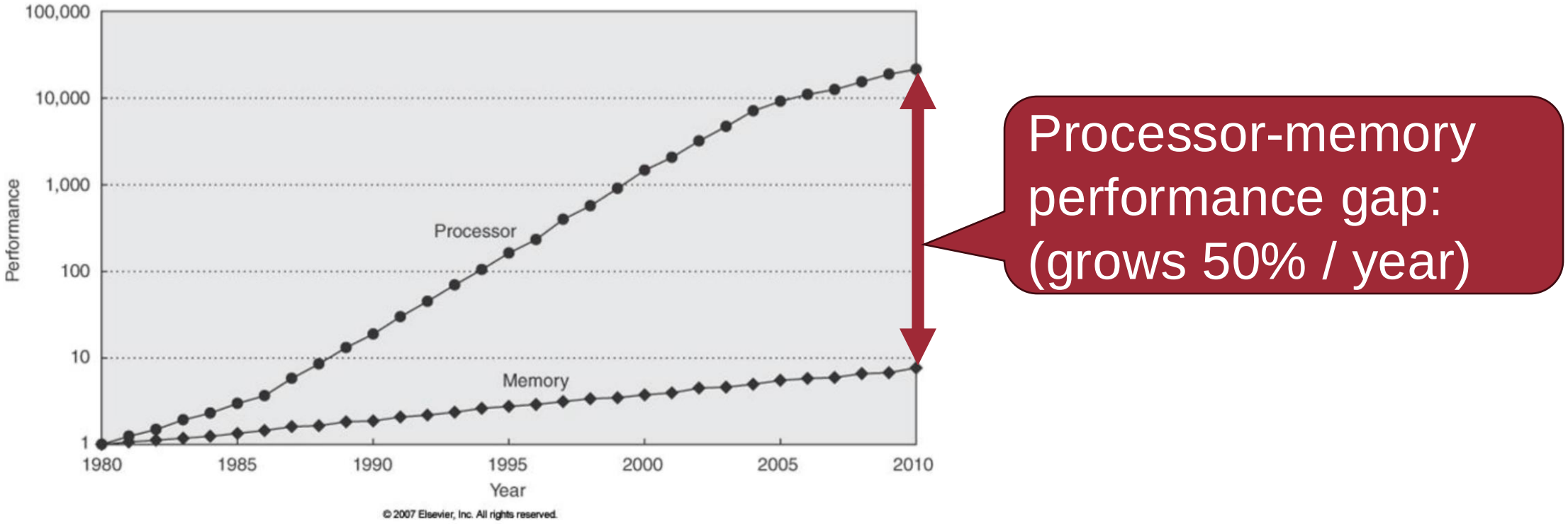


# Memory Hierarchy

- Taking advantage of ***locality***
  - Store everything on disk
  - Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory

# Why Memory Hierarchy Crucial?

- Processor speed improvement: 60%/ year (2x/1.5 year)
- Memory latency improvement: 9% / year (2x/10 years)




- Still it takes around 100+ CPU cycles for DRAM access
  - Hierarchy (계층구조) is the key!

# Question?