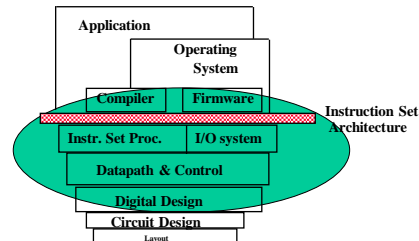
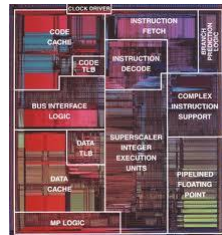


# CS/SE 3340

## Computer Architecture

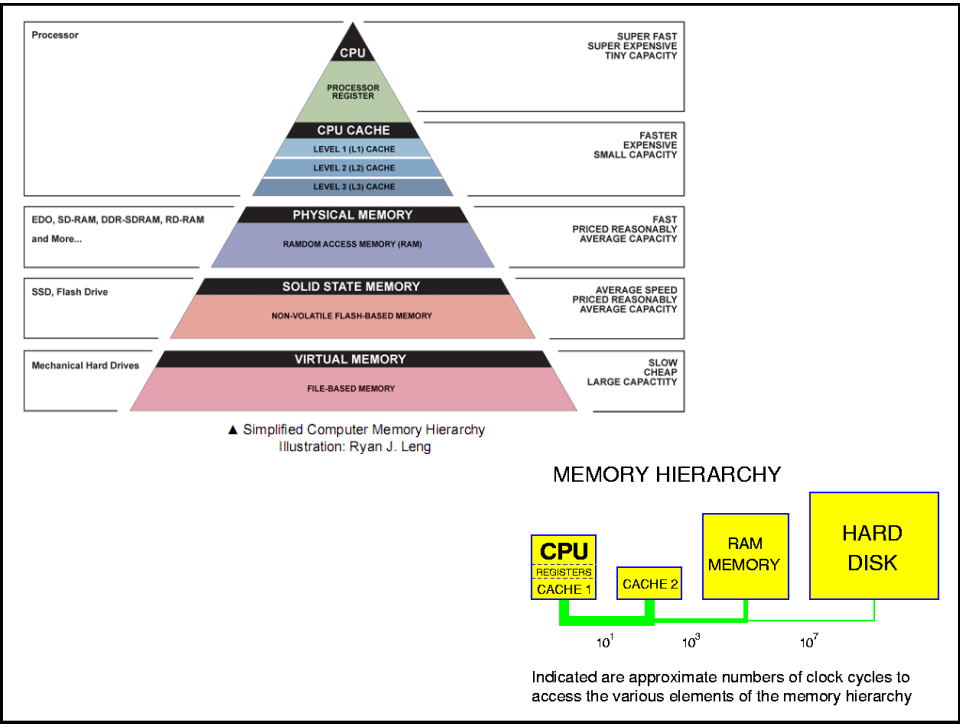


## Memory Hierarchy – An Overview

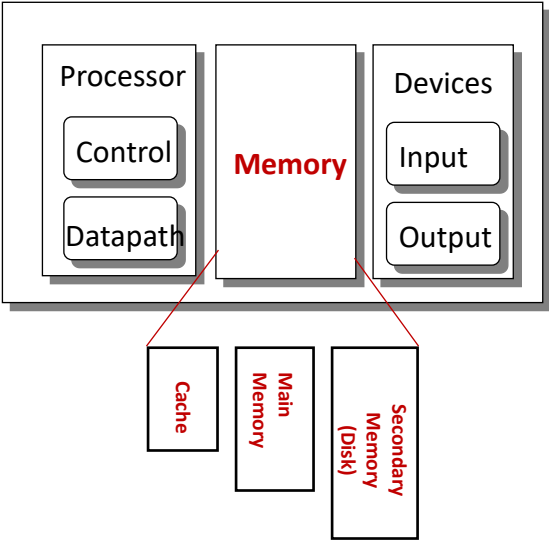
*Adapted from slides by Profs. M. Irwin, D. Patterson and J. Hennessey*

## Questions

1. What/why/how of memory hierarchy?
2. What/how of '*principle of locality*'
3. What/how of *caching*?
4. Where to find data in cache?
5. What/why of tags and valid bit
6. What is write-through/write-back policy?

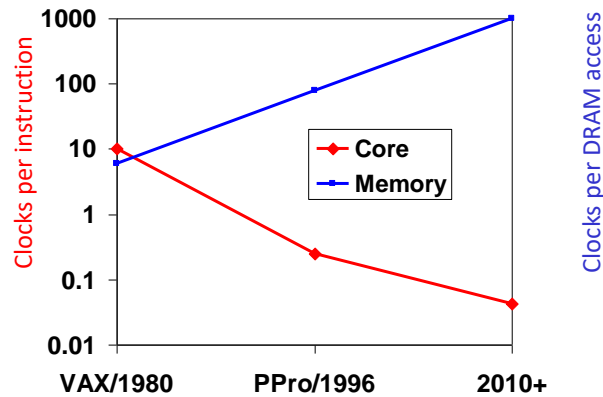


Major Components of a Computer



## The “Memory Wall”

- Processor v.s. main memory speed disparity continues to grow



Good **memory hierarchy** design is increasingly important to overall performance of a computer system!

5

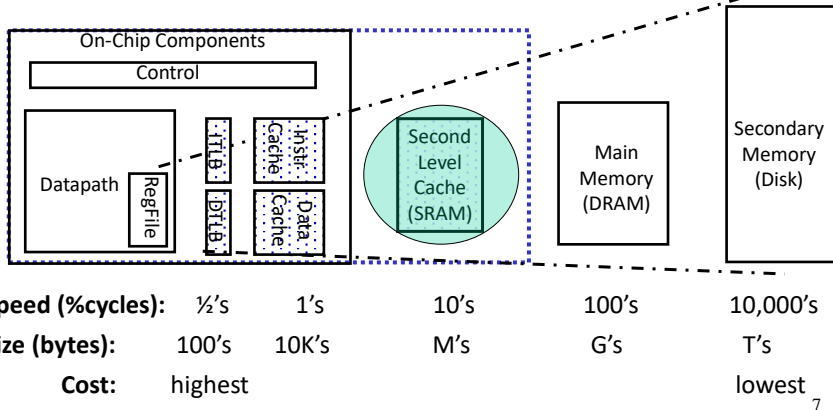
## The Memory Hierarchy Goal

- Due to cost constraints
  - **Fast** memories are *small* (static RAM)
  - **Large** memories are *slow* (dynamic RAM)
- How do we create a memory system that gives the illusion of being *large*, *cheap* and *fast* (most of the time)?
  - With **hierarchy**
  - With **locality**

6

## A Typical Memory Hierarchy

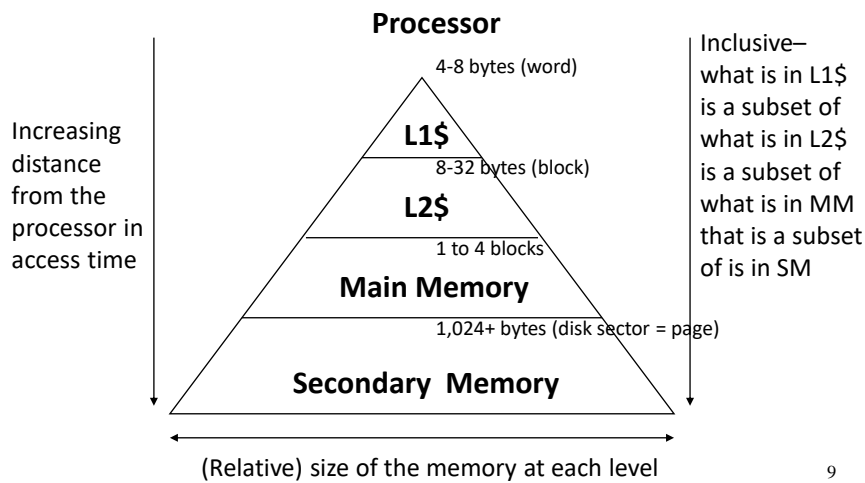
- Take advantage of the **principle of locality** to present the user with as much memory as is available in the **cheapest** technology at the speed offered by the **fastest** technology



## Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality**
  - Items accessed recently are likely to be accessed again soon, e.g. instructions in a loop, induction variables
  - ⇒ Keep **most recently accessed** data items closer to the processor
- Spatial locality**
  - Items near those accessed recently are likely to be accessed soon, e.g. array data
  - ⇒ Move blocks consisting of **contiguous words** closer to the processor

# Characteristics of the Memory Hierarchy



## How is the Hierarchy Managed?

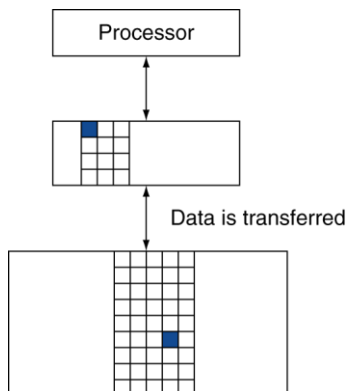
- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (**Translation Lookaside Buffer, TLB**) -> next session's topic
  - by the programmer (files)

# Cache Terminology

- **Capacity (C):**
  - number of data bytes in cache
- **Block (aka line) size (b):**
  - bytes of data brought into cache at once, unit of data transfer
- **Number of blocks ( $B = C/b$ ):**
  - number of cache blocks in cache:  $B = C/b$
- **Degree of associativity (N):**
  - number of cache blocks in a set
- **Number of sets ( $S = B/N$ ):**
  - each memory address maps to exactly one cache set

11

# Memory Hierarchy Levels



- **Block (aka line):** unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - **Hit:** access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - **Miss:** block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses  
= 1 – hit ratio
  - Then accessed data supplied from upper level

12

# Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses  $X_1, \dots, X_{n-1}, X_n$

$X_4$
$X_1$
$X_{n-2}$
$X_{n-1}$
$X_2$
$X_3$

a. Before the reference to  $X_n$

$X_4$
$X_1$
$X_{n-2}$
$X_{n-1}$
$X_2$
$X_n$
$X_3$

b. After the reference to  $X_n$

- How do we know if the data is present?
- Where do we look?

13

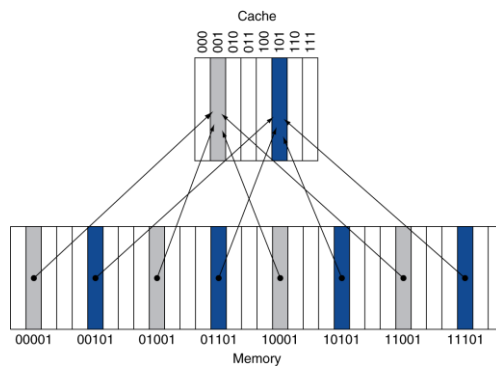
## Where To Find Data on Cache?

- Cache organized into  $S$  sets
  - Each memory address maps to exactly one set
- Cache is categorized by # of blocks in a set:
  - **Direct mapped**
    - 1 block per set
  - **N-way set associative**
    - $N$  blocks per set
  - **Fully associative**
    - all cache blocks in 1 set

14

## Where – Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2 – *Why?*
- Use low-order address bits

15

## How – Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the **tag**
- What if there is no data in a location?
  - **Valid bit (V)**: 1 = present, 0 = not present
  - Initially 0

16



# Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

17

# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

18

# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

19

# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

20

# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

21

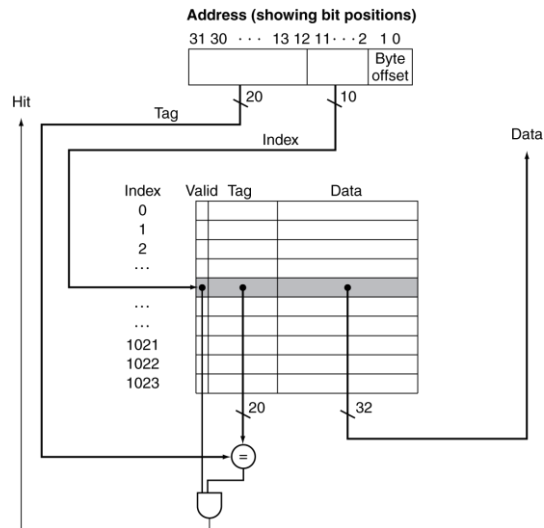
# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

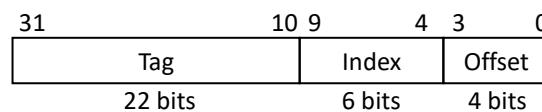
22

## Address Subdivision Example



23

## Example: Larger Block Size



- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
- Block address =  $\lfloor 1200/16 \rfloor = 75$
- Block number = 75 modulo 64 = 11

24

## Block Size Considerations

- Larger blocks should *reduce miss rate*
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks  $\Rightarrow$  fewer of them
    - More competition  $\Rightarrow$  *increased miss rate*
  - Larger blocks  $\Rightarrow$  pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

25

## Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

26

## Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- *Write through*: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI =  $1 + 0.1 \times 100 = 11$
- Solution: *write buffer*
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

27

## Write-Back

- On data-write hit, just update the block in cache
  - Keep track of whether each block is *dirty*
- When a *dirty* block is replaced
  - Write it *back* to memory
  - Can use a write buffer to allow replacing block to be read first

28

## Summary

- Principle of **locality** (data access) in computer systems
  - *Temporal*
  - *Spatial*
- Memory hierarchy exploits this principle to present the processor with as much memory as is available in the cheapest technology at the speed offered by the fastest technology
- *Cache*: bring often used data closer to the CPU
- **Write access** requires special consideration to maintain data consistency: *write-through & write-back*

29