Computer Hardware Engineering

IS1200, spring 2015

Lecture 8: Memory Hierarchy

David Broman

Associate Professor, KTH Royal Institute of Technology

Assistant Research Engineer, University of California, Berkeley

Slides version 1.0

Course Structure

Module 1: Logic Design

L, L → DCÖ1, DCÖ2 → Lab:dicom

Module 4: I/O Systems

L7 → E6 → E7 → Lab: nios2io

E9 → Lab: nios2int

Module 2: C and Assembly Programming

L1 → L2 → E1

L3 → E2 → E3 → Lab: nios2time

L4 → E4 → Home lab: C

Module 5: Memory Hierarchy

L8 → E8 → Home Lab: cache

Module 3: Processor Design

L5 → L6 → E5

Module 6: Parallel Processors and Programs

L9 → L10 → E10

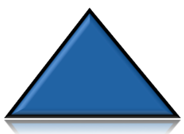
Abstractions in Computer Systems

Computer System	Networked Systems and Systems of Systems
Application Software Operating System	Software
Instruction Set Architecture	Hardware/Software Interface
Microarchitecture Logic and Building Blocks Digital Circuits	Digital Hardware Design
Analog Circuits Devices and Physics	Analog Design and Physics

Agenda

Part I

Overview of Memory Hierarchies



Part II

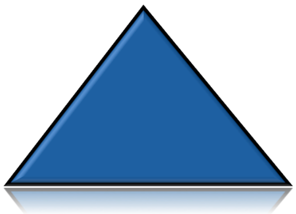
Instruction and Data Caches

Part III

Virtual Memory

Part I

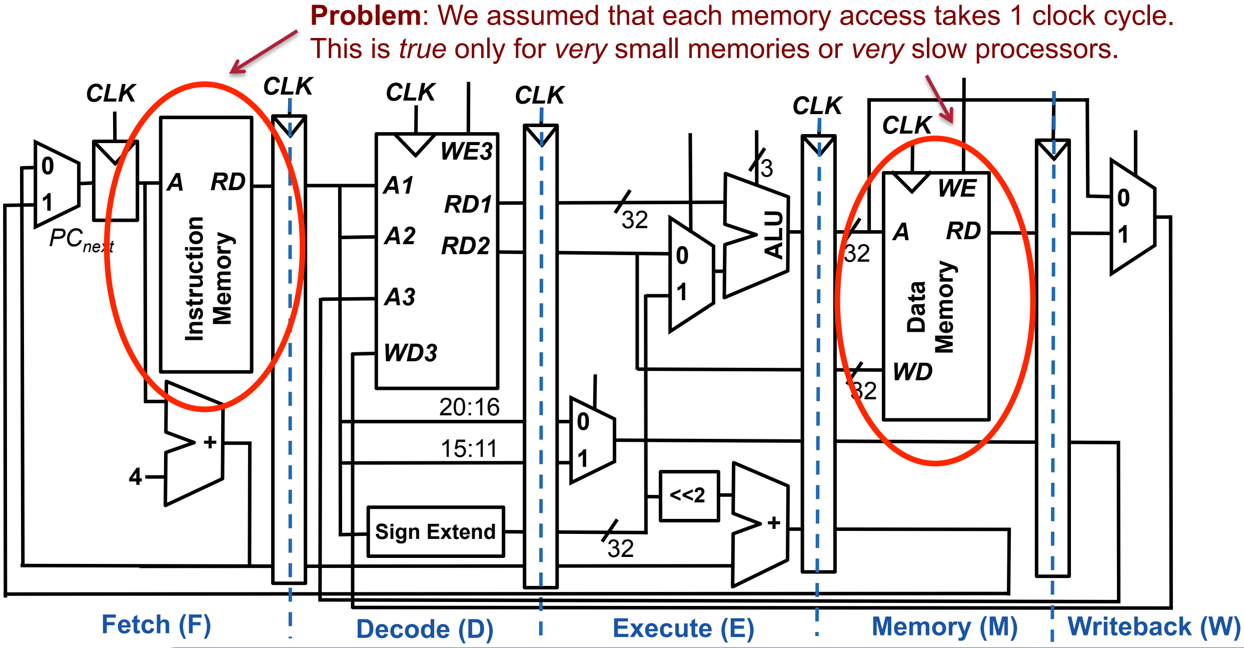
Overview of Memory Hierarchies



Acknowledgement: The structure and several of the good examples are derived from the book "Digital Design and Computer Architecture" (2013) by D. M. Harris and S. L. Harris.

Data Path (Revisited) Instruction and Data Memory

Problem: We assumed that each memory access takes 1 clock cycle. This is *true* only for *very* small memories or *very* slow processors.

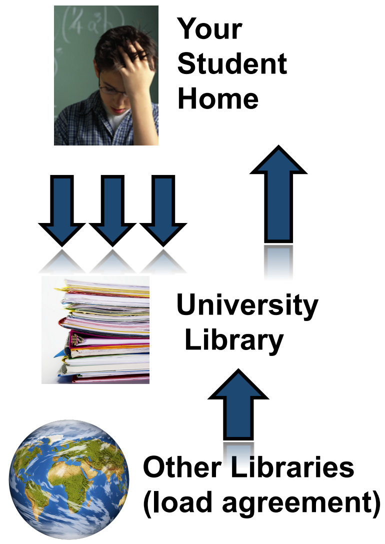


Temporal and Spatial Locality - The Library Example

Problem 1. Each time you have a problem, you go to the library and read. You have to walk a lot…

Solution 1. You borrow the book you read often.

Temporal Locality
If you recently used a book, it is likely that you will read it again.



Problem 2. Each time you find a closely related problem you, borrow a new book. You have to go and borrow books many times.

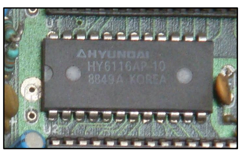
Solution 2. When you borrow a book, you borrow 10 books on the same shelf, with closely related topics.

Spatial Locality
When you borrow a particular book, you are probably interested in other similar books.

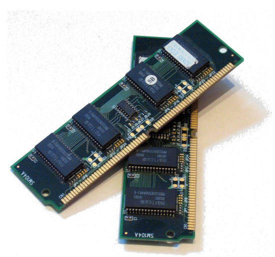
Problem 3. The library cannot store all possible books in the world on their shelves.

Solution 3. The library can borrow from other libraries, when requested by you.

Memory Technologies (1/2) SRAM and DRAM



SRAM (Static Random Access Memory)
- Simple integrated circuit, usually with one access port.
- Today, on-chip memory (same as processor)
- Access time 0.5-2.5ns Cost per GiB in 2012: $500-$1000



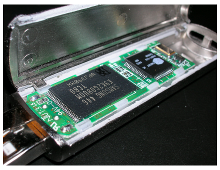
Douglas Whitaker, Wikipedia, CC BY-SA 2.5

DRAM (Dynamic Random Access Memory)
- Memory stored in capacitors – need to be refreshed
- One transistor per bit – much cheaper than SRAM
- SDRAM (synchronous DRAM). Uses clocks. Transfer data in bursts.
- DDR (Double Data Rate) SDRAM. Transfer data both on rising and falling clock edge.
- Access time: 50-70ns, Cost per GiB in 2012: $10-$20

Source: Patterson and Hennessy, 2012

David Broman
dbro@kth.se
Part I Overview of Memory Hierarchies
Part II Instruction and Data Caches
Part III Virtual Memory

Memory Technologies (2/2)
Flash and Magnetic Disk



Wikipedia, CC BY-SA 3.0

Flash Memory

- Electrically erasable programmable read-only memory (EEPROM)
- Can wear out
- Used in solid state drivers (SSD)
- Access time: 5,000-50,000 ns, Cost per GiB in 2012 $0.75-$1

Wikipedia Evan Amos, CC BY-SA 3.0

Magnetic Disk

- Collection of platters that spin 5,400 to 15,000 revolutions per minutes (rpm).
- Access time: 5,000,000-50,000,000 ns
  Cost per GiB in 2012 $0.05-$0.10

Clearly, there is a tradeoff between cost, access time, and size

How can we utilize these differences?

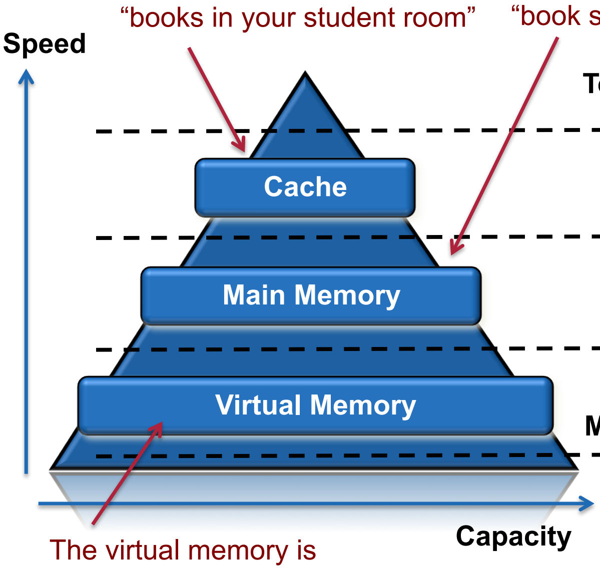
Source: Patterson and Hennessy, 2012

Memory Hierarchy

The cache is the "books in your student room"

The main memory is the "book shelves in the library"

Fast, small, but expensive memory on the top



Technology	Access Time (ns)	Price $ per GiB (2012)
SRAM	0.5 ns – 2.5 ns	$500 - $1000
DRAM	50 ns – 70 ns	$10 - $20
Flash	5 us – 50 us	$0.75 - $1.00
Magnetic Disk	5 ms – 20 ms	$0.05 - $0.10

The virtual memory is "the other libraries in the world"

Slow, large, inexpensive memories at the bottom

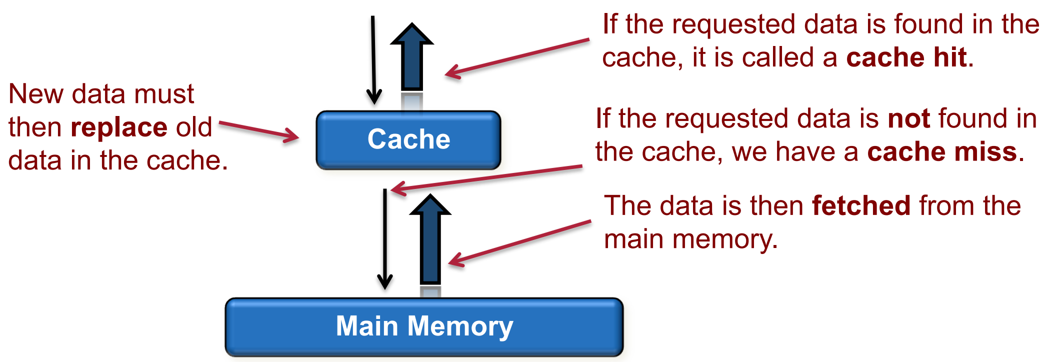
Source: Patterson and Hennessy, 2012

Part II

Instruction and Data Caches

Acknowledgement: The structure and several of the good examples are derived from the book "Digital Design and Computer Architecture" (2013) by D. M. Harris and S. L. Harris.

Reading From Memory



$$\text{Miss Rate} = \frac{\text{Number of misses}}{\text{Total number of memory accesses}}$$

$$\text{Hit Rate} = \frac{\text{Number of hits}}{\text{Total number of memory accesses}}$$

What data should be in the cache so that we maximize the hit rate?

How to achieve low miss rates?

"It's difficult to make predictions, especially about the future" — **Attributed to various persons in the history**

Predict

We need to make **predictions**!

Temporal locality:
The processor is likely to access recently accessed addresses again.

We should keep recently used data in the cache.

We can fetch more than one word of data at the same time.

Spatial locality:
The processor is likely to access addresses close to each other.

Cache Terminology

Capacity (C)
Number of words or bytes in the cache.

Number of Sets (S)
Each set holds one or more blocks of data. (Sometimes the term **row** is used instead of set)

Block size (b)
Number of words or bytes in a block.

Number of Blocks (B)
The total number of blocks.
Always: B >= S

Degree of associativity (N)
N = B/S

Minimal Cache Example

The **capacity** of cache is $2^3 = 8$ words. The word size is 32-bit.

N = 1

0x0000 0000	Set 7, 111_2
0x0000 0000	Set 6, 110_2
0x0000 0000	Set 5, 101_2
0x0000 0000	Set 4, 100_2
0x0000 0000	Set 3, 011_2
0x0000 0000	Set 2, 010_2
0x0000 0000	Set 1, 001_2
0x0000 0000	Set 0, 000_2

Here B=8, i.e., B=S.

The block size is one word, b=1.

There are 8 **sets**, i.e., S=8.

Direct Mapped Cache (1/3) The mapping

What we showed was an example of a **direct mapped cache**, where S=B.

Address	Main Memory
11…11111100	0x0000 2200
11…11111000	0x0000 4400
11…11110100	0x0000 ff00
…	
00…00100000	0x0000 2200
00…00011100	0x0000 4400
00…00011000	0x0000 ff00
00…00010100	0x0000 2200
00…00010000	0x0000 4400
00…00001100	0x0000 ff00
00…00001000	0x0000 2200
00…00000100	0x0000 4400
00…00000000	0x0000 ff00

Main Memory

Cache	
0x0000 2200	Set 7, 111_2
0x0000 4400	Set 6, 110_2
0x0000 ff00	Set 5, 101_2
0x00ff 0000	Set 4, 100_2
0x0022 0000	Set 3, 011_2
0x0033 0000	Set 2, 010_2
0x0000 5500	Set 1, 001_2
0x0000 6600	Set 0, 000_2

Cache

A memory address maps to exactly one set.

Exercise: Which *set* is address 0x0000 00018 mapping to?

Answer: Set 6

Direct Mapped Cache (2/3) Cache Fields

How do we know which memory blocks that are stored in the cache?
Answer: We store the tag field in the cache.

Address	Main Memory
11…11111100	0x0000 2200
11…11111000	0x0000 4400
11…11110100	0x0000 ff00
…	

An address has certain components

27-bit	3-bit	2-bit
11…111	101	00

A **tag** field specifies which address

A **set** field (also called the **index**) specifies which set in the cache

A **byte offset** field specifies which byte in the block

Cache	
0x0000 2200	Set 7, 111_2
0x0000 4400	Set 6, 110_2
0x0000 ff00	Set 5, 101_2
0x00ff 0000	Set 4, 100_2
0x0022 0000	Set 3, 011_2
0x0033 0000	Set 2, 010_2
0x0000 5500	Set 1, 001_2
0x0000 6600	Set 0, 000_2

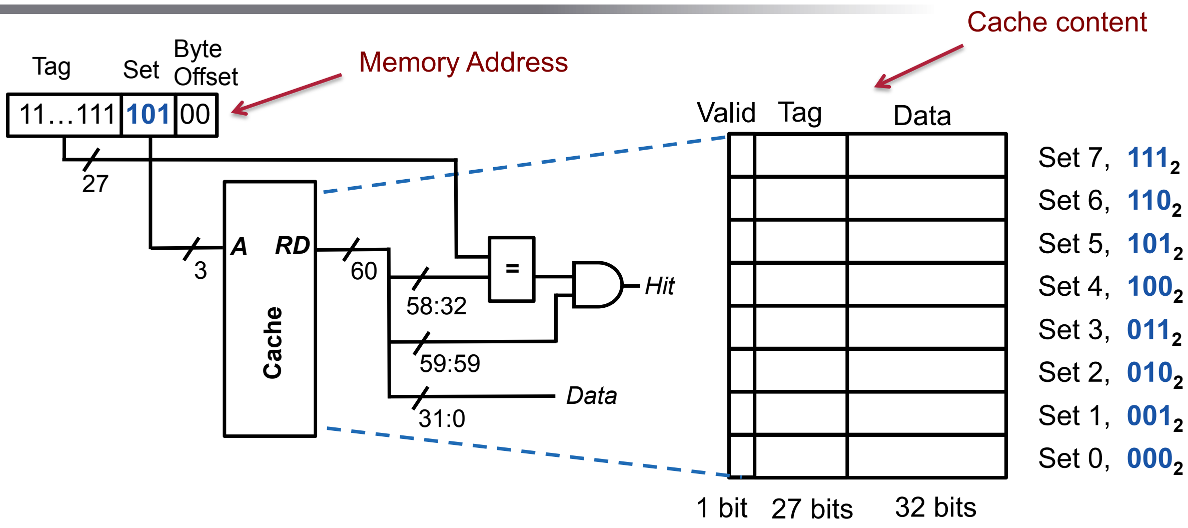
Cache

Exercise: If C = 256 words, N=1, b = 4 words, and address and word size are 32-bits. How many bits are then needed for the *tag* and the *set* fields?

Answer:
byte offset field = 4 bits
set field = 6 bits
tag field = 22 bits

How do we know if the data stored in the cache is valid?
Answer: We add a valid bit in the cache.

Direct Mapped Cache (3/3) Hardware Implementation



Loop Example 1 Data Cache – Temporal Locality

E

```
     addi $t0, $0, 5
loop:
     beq  $t0, $0, done
     lw   $t1, 0x4($0)
     lw   $t2, 0xC($0)
     addi $t0, $t0, -1
     j    loop
done:
```

Valid	Tag	Data	
0			Set 7, 111_2
0			Set 6, 110_2
0			Set 5, 101_2
0			Set 4, 100_2
1	00..00	mem[0x00..0C]	Set 3, 011_2
0			Set 2, 010_2
1	00..00	mem[0x0004]	Set 1, 001_2
0			Set 0, 000_2
1 bit	27 bits	32 bits	

Exercise: Assume that the cache is empty when entering the program. What is the *data* cache miss rate and the cache contents when reaching program point **done**.

Answer: The missrate is 2/10 = 20%.

$4_{16} = \mathbf{001}00_2$ $\quad C_{16} = 12_{10} = \mathbf{011}00_2$

Loop Example
Instruction Cache – Spatial Locality

```
    addi $t0, $0, 5
loop:
    beq  $t0, $0, done
    lw   $t1, 0x4($0)
    lw   $t2, 0xC($0)
    addi $t0, $t0, -1
    j    loop
done:
```

Note the **spatial locality**. Since we load 4 instruction each time, we do not get cache misses for each instruction.

Exercise: Assume that the first `addi` instruction starts at address 0x0000 4000. The instruction cache has S = B = 256 words and b = 4 words. How many instruction cache misses occurs when executing the program, presupposed that the cache was empty from the beginning.

Answer: Two cache misses

First, when loading the first 4 instructions, then when loading the two last instructions.

4 bits for representing 16 bytes block
0x0000 4000 // Address to first addi
0x0000 4010 // Address to second addi
The mapping does not conflict.

Loop Example 2
Data Cache – Conflicts

```
    addi $t0, $0, 5
loop:
    beq  $t0, $0, done
    lw   $t1, 0x4($0)
    lw   $t2, 0x24($0)
    addi $t0, $t0, -1
    j    loop
done:
```

Valid	Tag	Data	
0			Set 7, 111_2
0			Set 6, 110_2
0			Set 5, 101_2
0			Set 4, 100_2
0			Set 3, 011_2
0			Set 2, 010_2
1	00..00	mem[0x0024]	Set 1, 001_2
0			Set 0, 000_2
1 bit	27 bits	32 bits	

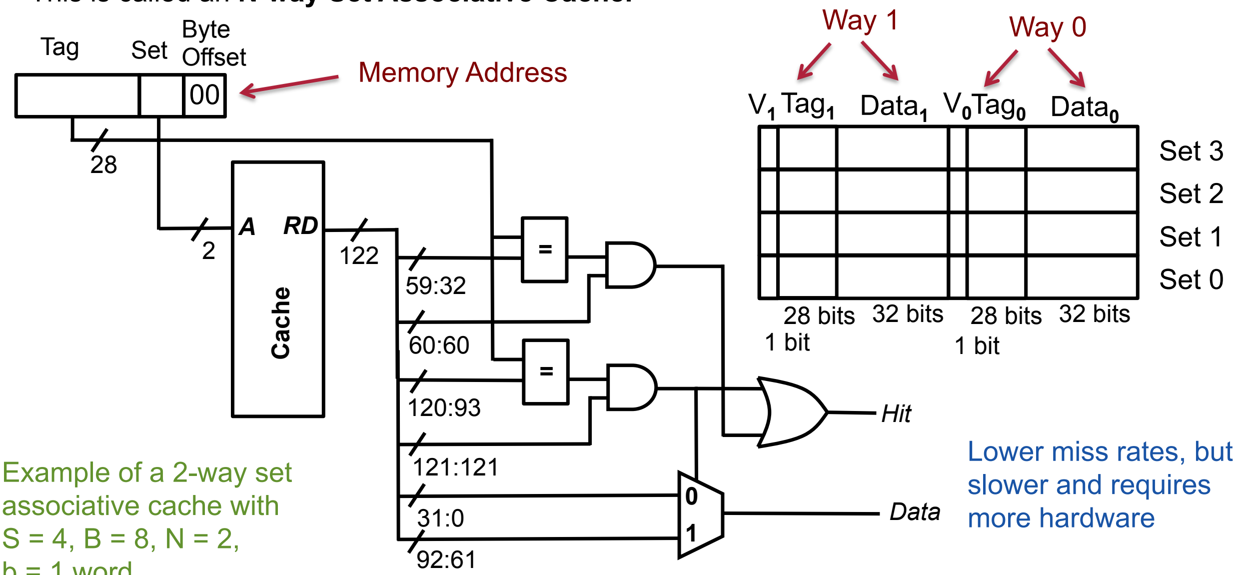
Exercise: Assume that the cache is empty when entering the program. What is the *data* cache miss rate and the cache contents when reaching program point **`done`**.

Answer:
$4_{16} = 0000\ 0100_2$ $24_{16} = 0010\ 0100_2$

The miss rate is 10/10 = 100%.

N-way Set Associative Cache

We can reduce conflicts by having N blocks in each *set*.
This is called an **N-way Set Associative Cache.**



Example of a 2-way set associative cache with S = 4, B = 8, N = 2, b = 1 word

Lower miss rates, but slower and requires more hardware

Fully Associative Cache and Direct Mapped Cache revisited

All caches are N-way set associative caches.

A **fully associative cache** has B = N and S = 1.

Another name for a fully associative cache is a **B-way set associative cache**.

Gives the lowest miss rate, but requires most hardware.

A **direct mapped cache** has N =1 and B = S.

Another name for a direct mapped cache is a **one-way set associative cache**.

Replacement Policy

Direct Mapped Cache

Each address maps to a unique block and set.
Hence, when a set is full, it must be replaced with the new data.

N-way Set Associative Cache where N > 1

- **Least Recently Used (LRU) Policy**. Simple with a 2-way set associative cache by using a *use bit U*. Commonly used.
- **Pseudo-LRU Policy**. For N-ways were N > 2. Indicate the least recently used *group* and upon replacement, randomly selects a way in the group.
- **First-in First-out (FIFO)** replacement policy.
- **Random** replacement policy.

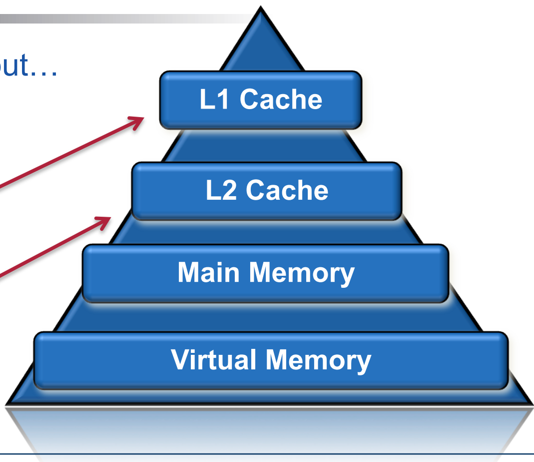
Multi-Level Caches

Large caches tend give lower miss rates, but…
Large caches tend to be slower.

Solution: Multi-Level Caches

L1 cache, small enough to get 1-2 cycle times.

L2 cache, is also build in SRAM, but is larger, and therefore slower.



Examples from Reality

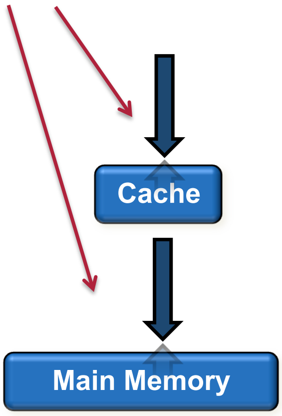
ARM Cortex-A8
- L1, 4-way, 32KiB, split instruction/data, random replacement
- L2, 8-way, 128KiB, unified inst./data, random replacement
- No L3 cache

Intel Core-I7 920
- L1, 4-way (i),8-way (d), 32KiB, split instruction/data, Approximate LRU
- L2, 8-way, 256KiB, unified inst./data
- L3, 16-way, 8MiB, Approximate LRU

Write Policy

Write-Through Policy

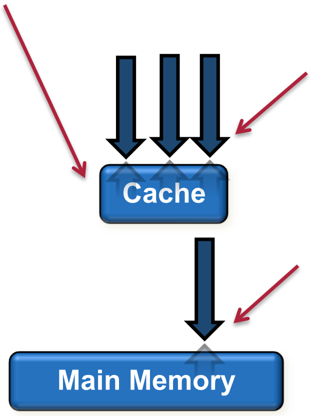
When writing to the cache, the data is simultaneously written back to main memory.



Write-Back Policy

A dirty bit (D) is associated with each cache block.

On a write, D is set to 1. If D is 0, the cache block has not been written to.

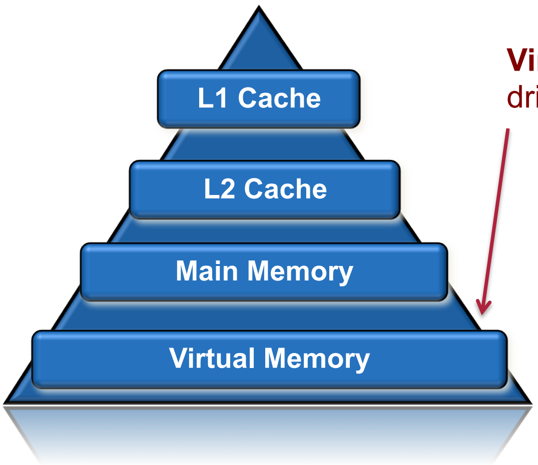


Data is written back to main memory when evicted and D = 1

Part III

Virtual Memory

Rationales for Virtual Memory



Virtual memory uses the hard drive. Slow, but large memory.

Why?

- Gives the illusion of a **very big memory.**
- Gives **memory protection** between concurrent running programs (each process has its on virtual memory space).

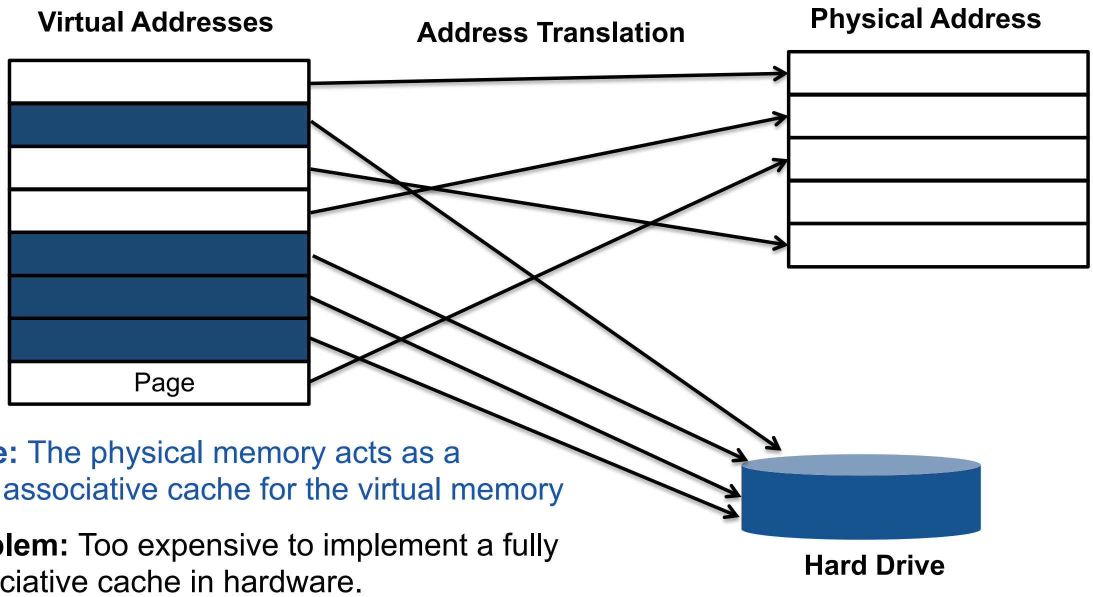
Virtual Memory vs. Caches

Virtual memory has similarities to caches, but uses another terminology.

Cache	**Virtual Memory**
Block	Page
Block size	Page Size
Block offset	Page offset
Miss	Page fault
Tag	Virtual Page number

Typically, 4KB or more

Virtual Memory Overview



Note: The physical memory acts as a fully associative cache for the virtual memory

Problem: Too expensive to implement a fully associative cache in hardware.

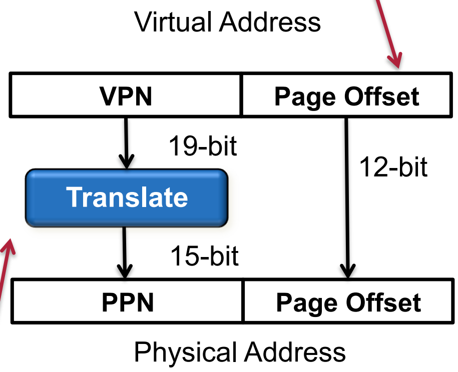
Page Table and the Memory Management Unit (MMU)

Solution: Page table, stored in Physical Memory

Valid	Physical Page Number	Virtual Page Number
0		0x5FFFF
0		0x5FFFE
0		0x5FFFD
0		0x5FFFC
0		0x5FFFB
0		….
1	0x5FFF	0x00001
0		0x00000

The **operating system (OS)** is responsible for updating the page table and moving data between memory and disk

No translation needed for the offset.



Translate the virtual page number (VPN) to the physical page number (PPN). Done in hardware in the **Memory Management Unit (MMU)**.

The Translation Lookaside Buffer (TLB)

Problem: Accessing the page table each memory access would result in extreme performance degradation.

A Translation Lookaside Buffer (TLB) caches the latest pages.

A TLB is typically organized as a **fully associative cache**. Holds typically 16 to 512 entries.

Due to temporal and spatial locality, a TLB typically has a hit rate of more than 99%

Summary

Some key take away points:

- **Memory hierarchies** are used because memories have different cost, size, and speed.
- There are two kinds of caches: **instruction caches** and **data caches**.
- Two important properties that make caches useful: **temporal locality** and **spatial locality.**
- Caches can be **direct mapped**, **N-way**, or **fully associative**.
- **Virtual memories** enable large virtual address spaces and enable memory protection between different concurrent programs.

Thanks for listening!