Computer Hardware Engineering (IS1200)
Computer Organization and Components (IS1500)

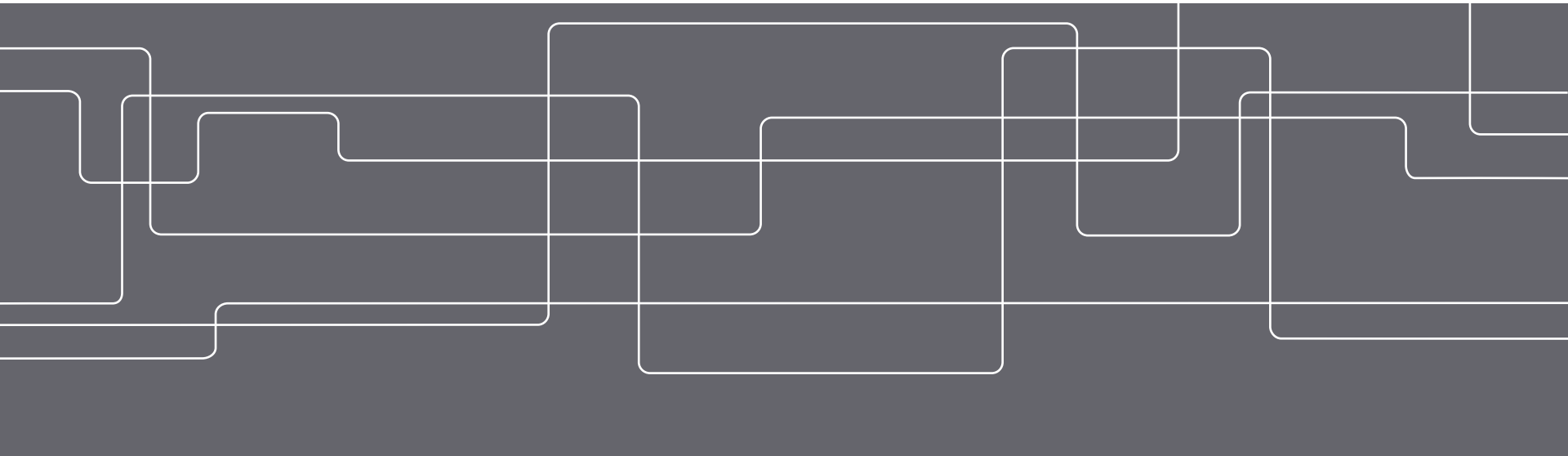
Spring 2021

Lecture 11: Memory Hierarchy

David Broman

Associate Professor, KTH Royal Institute of Technology

Slides by David Broman, KTH (Extensions by Artur Podobas)



Course Structure

Module 1: C and Assembly Programming

LE1 → LE2 → LE3 → LE4 →

EX1 LAB1 S1 LAB2

Module 2: I/O Systems

LE5 → LE6 →

EX2 LAB3

Module 3: Logic Design **(IS1500 only)**

PROJ START

LE7 → LE8 →

EX3 LD-LAB

Module 4: Processor Design

LE9 → LE10 →

EX4 S2 LAB4

Module 5: Memory Hierarchy

LE11 →

EX5 S3

Module 6: Parallel Processors and Programs

LE12 → LE13 →

EX6 S4

Proj. Expo LE14

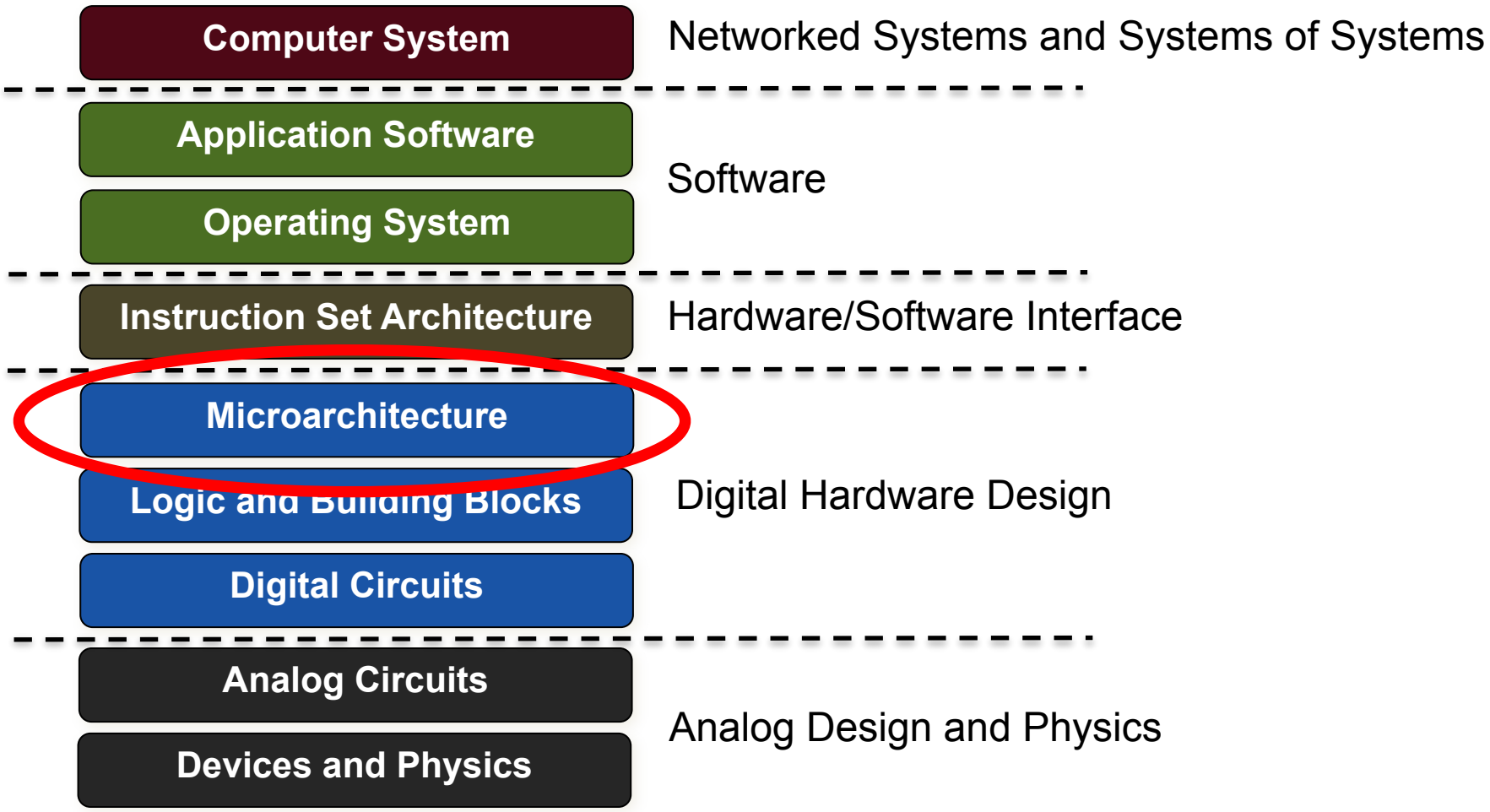
Artur Podobas
podobas@kth.se

Part I
Overview of Memory Hierarchies

Part II
Instruction and Data Caches

Part III
Virtual Memory

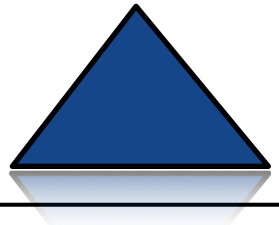
Abstractions in Computer Systems



Agenda

Part I

Overview of Memory Hierarchies



Part II

Instruction and Data Caches

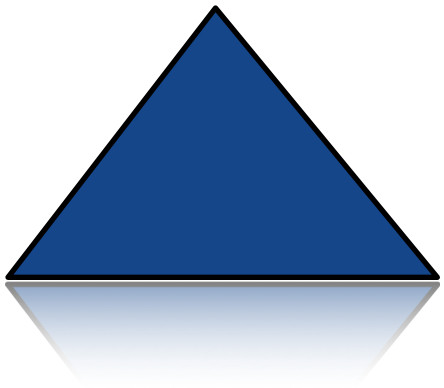
Part III

Virtual Memory

This lecture part is available online as a video. See the page "Literature and Resources" on the course web.

Part I

Overview of Memory Hierarchies

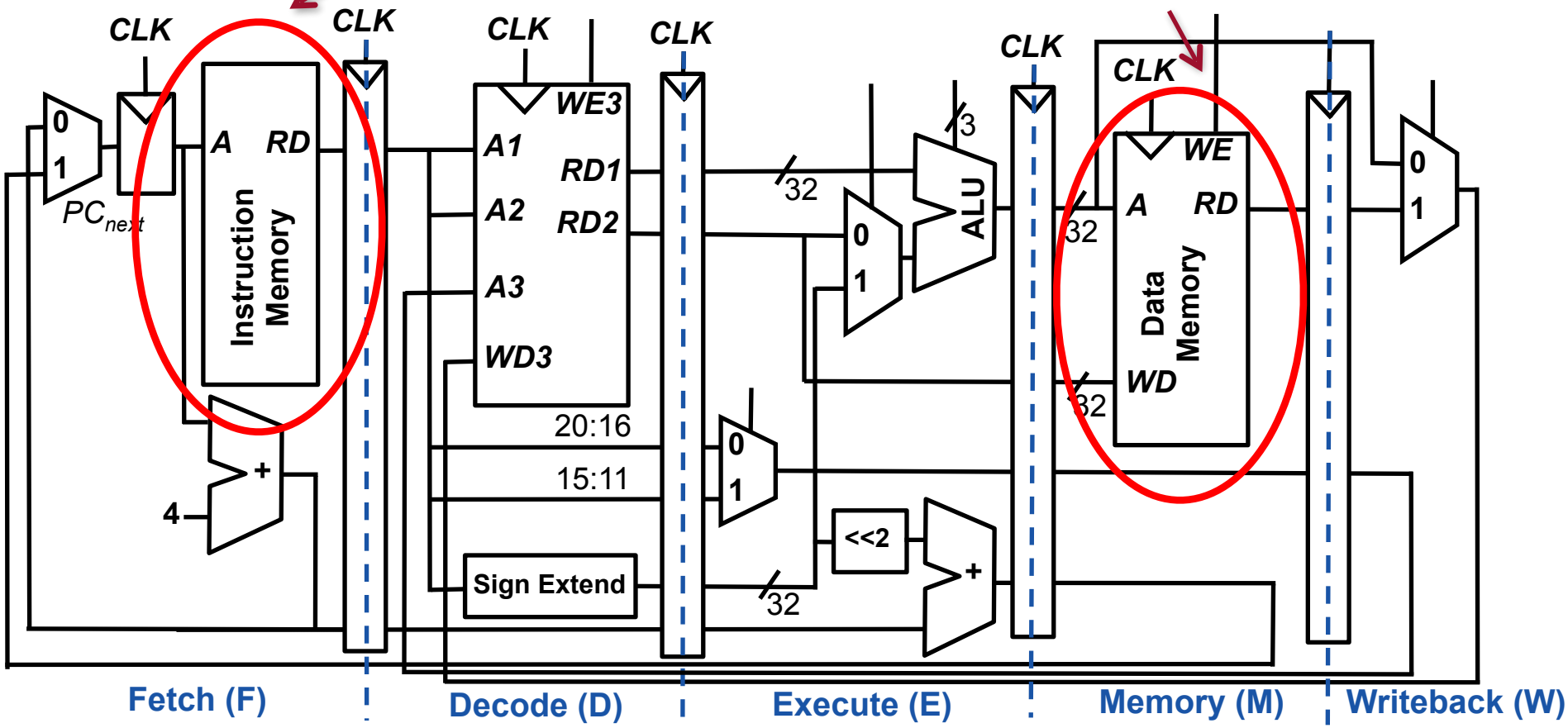


Acknowledgement: The structure and several of the good examples are derived from the book "Digital Design and Computer Architecture" (2013) by D. M. Harris and S. L. Harris.

David Broman
dbro@kth.se

Data Path (Revisited)
Instruction and Data Memory

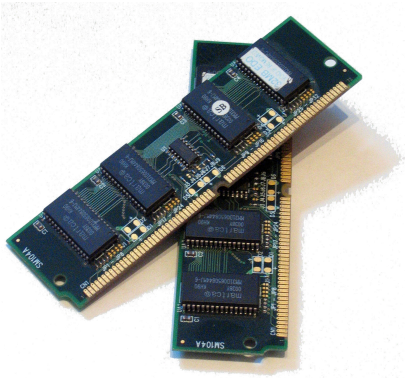
Problem: We assumed that each memory access takes 1 clock cycle.
This is *true* only for *very* small memories or *very* slow processors.



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



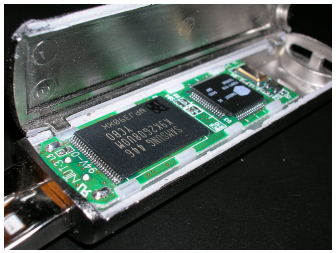
Douglas Whitaker, Wikipedia, CC BY-SA 2.5

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

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

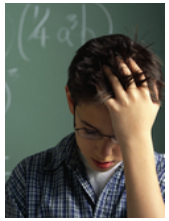
Temporal and Spatial Locality - The Library Example

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



Your Student Home

University Library

Other Libraries (loan agreement)

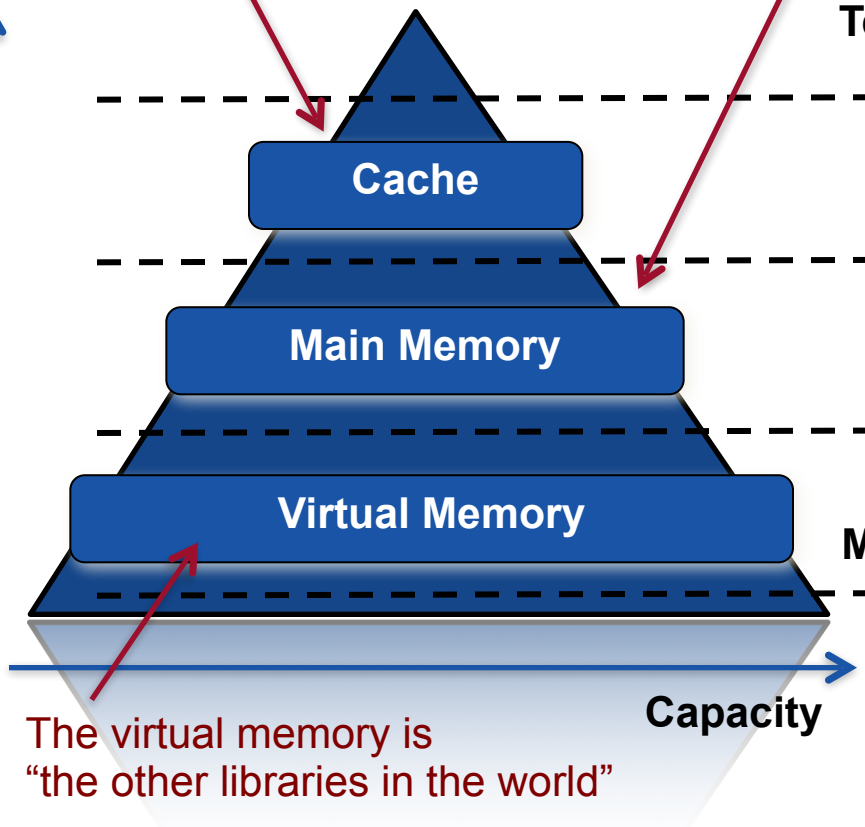
Solution 3. The library can borrow from other libraries, when requested by you.

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.

Memory Hierarchy



The cache is the "books in your student room"

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

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

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

Slow, large, inexpensive memories at the bottom

Source: Patterson and Hennessy, 2012

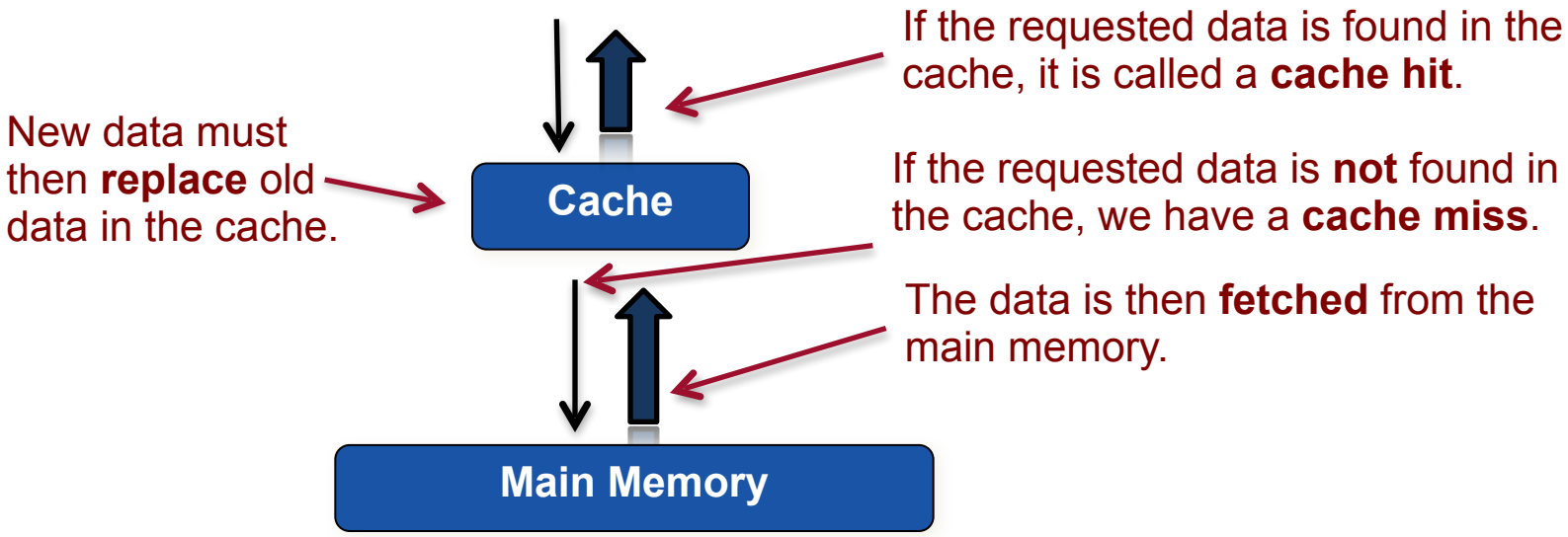
Part II

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.

David Broman
dbro@kth.se

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

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

We should keep recently used data in the cache.

Predict

We need to make **predictions**!

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

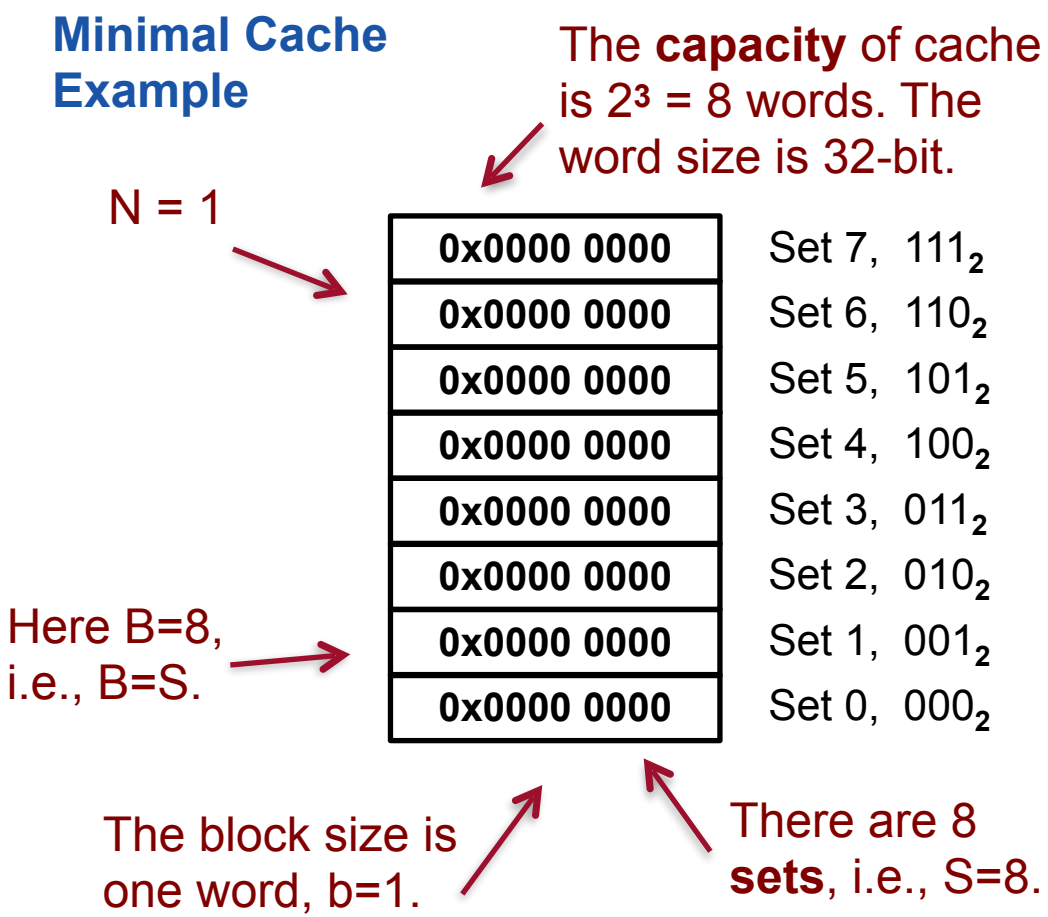
Capacity (C)
Number of words or bytes in the cache.

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

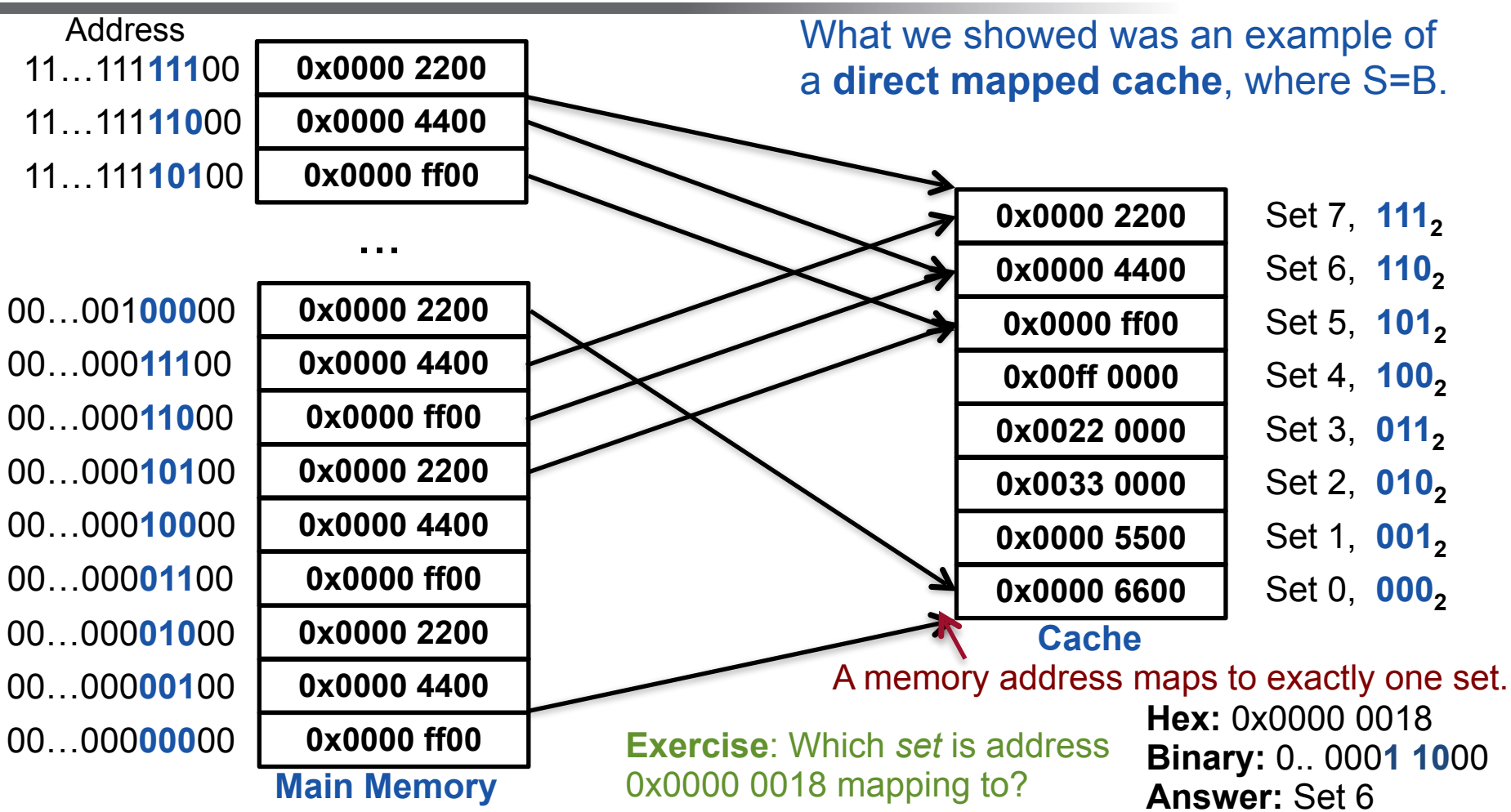
Block size (b)
Number of words or bytes in a block.

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

Degree of associativity (N)
N = B/S

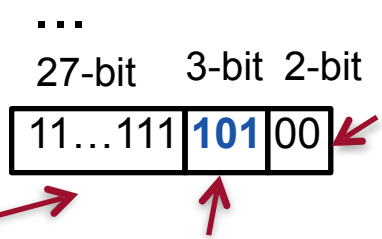


Direct Mapped Cache (1/3)
The mapping



Direct Mapped Cache (2/3)
Cache Fields

An address has certain components



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

A **tag** field specifies which address

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

It is possible to further divide the offset into block offsets (see H&H p. 488). However, for simplicity, in the exercises and exams in this course, we just talk about one offset, simply referred to as the **byte offset**.

How do we know which memory blocks that are stored in the cache**?**

Answer: We store the tag field in the 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?

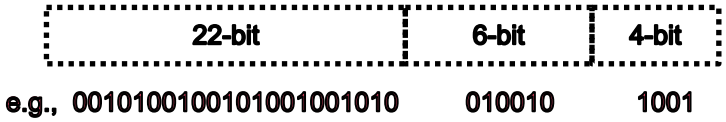
Answer:
byte offset field = 4 bits
set field = 6 bits
tag field = 22 bits

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 (2/3) Example

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?



Solution:
The total cache capacity is 256 Words.

Calculate byte offset width:

Each block contains 4 words
Each word = 4 byte.
Each block contains 4*4 B =16.
How many bits do we require to represent 16 numbers? 2^4 = 16. Our byte offset is **4 bits wide.**

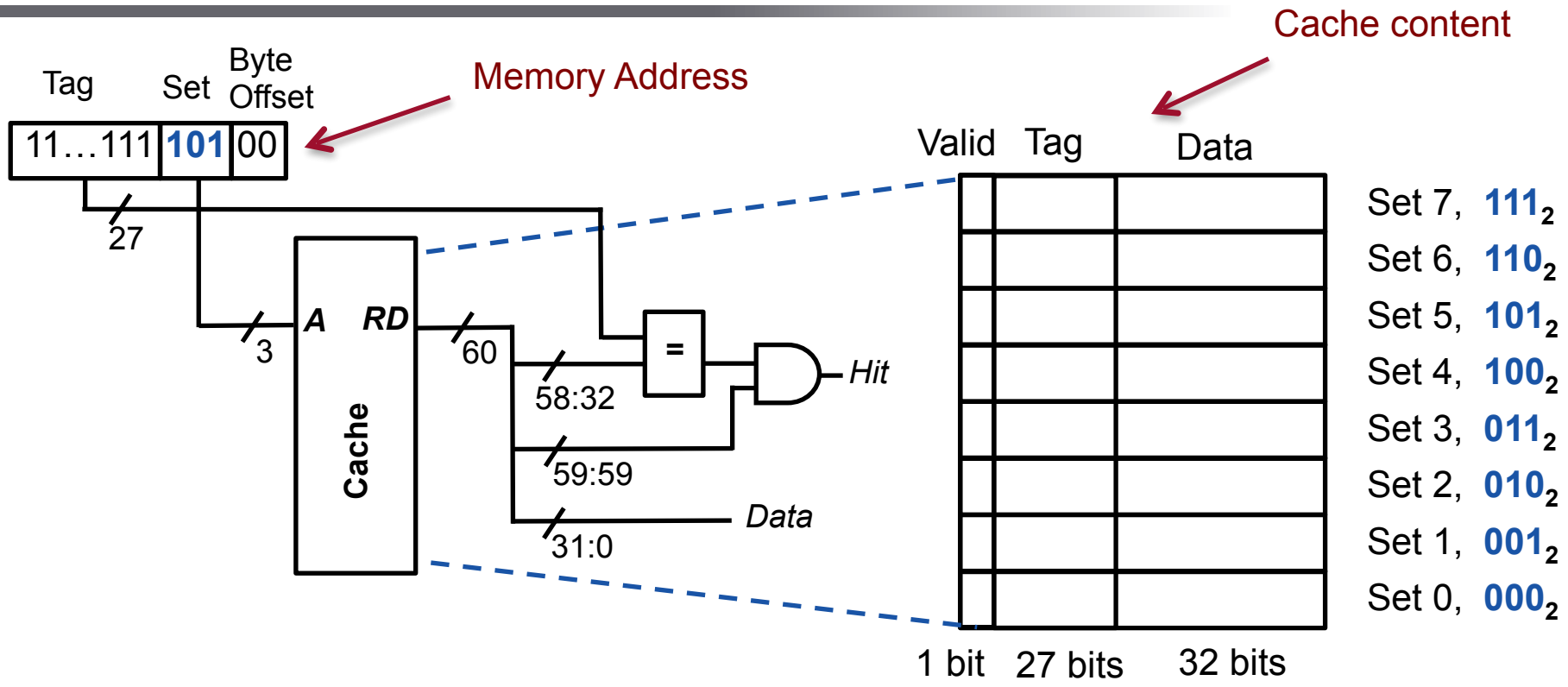
Calculate Set-field:

Our capacity is 256 Words, each block is 4 words.
Total sets: 256 / 4 = 64 sets. (N=1)
How many bits do we need to represent 64? 2^6=64.
Set field is 6 bit wide.

Calculate Tag:

Address is 32-bit – 6 bit (set field) – 4 bit (byte offs) = **22 bit.**

Direct Mapped Cache (3/3)
Hardware Implementation



Loop Example 1
Data Cache – Temporal 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:
```

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} = **001**00_2 $C_{16} = 12_{10}$ = **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, C = 4096 bytes and b = 16 bytes. 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 4**00**0 // Address to first addi
0x0000 4**01**0 // Address to second addi
The mapping does not conflict.

David Broman
dbro@kth.se

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:
```

0x0000 4000 M

0x0000 4004 H H

0x0000 4008 H H

0x0000 400c H H

0x0000 4010 M H

0x0000 4014 H H

	Valid	Tag	Data			
Set 0	1	0x00004	addi	beq	lw	lw
Set 1	1	0x00004	addi	j lop		
Set 2						
Set 3						
Set 4						
Set 5						
Set 6						
Set 7						
Set ...	Set 8..255 follow					

Exercise: Assume that the first `addi` instruction starts at address 0x0000 4000. The instruction cache has S = B, C = 4096 bytes and b = 16 bytes. How many instruction cache misses occurs when executing the program, presupposed that the cache was empty from the beginning.

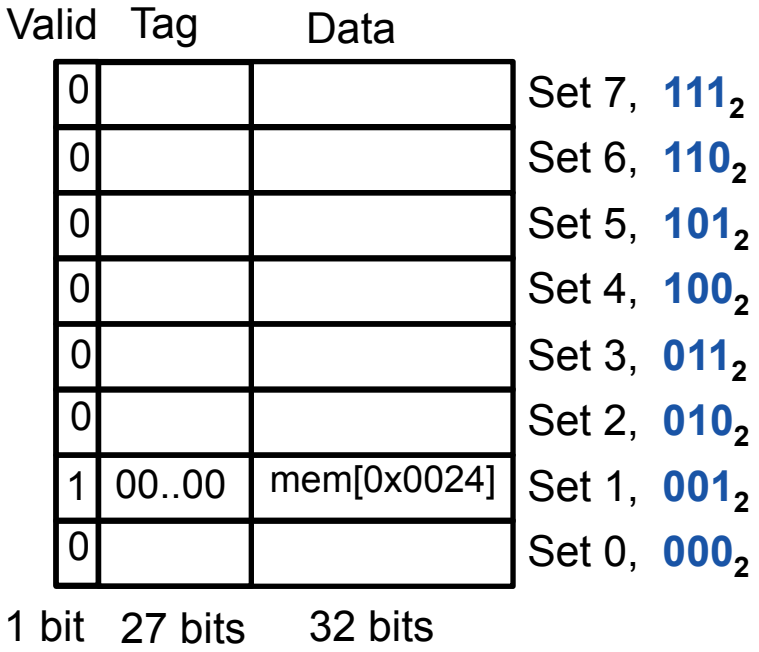
Sets = 4096 / 16 => 256 => **8**-bit set field

byteoffs = 16 => **4**-bit byte offset field

Total: 2 misses.

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:
```



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:

$4_{16} = 000\mathbf{0}\ \mathbf{01}00_2$ $\quad 24_{16} = 001\mathbf{0}\ \mathbf{01}00_2$

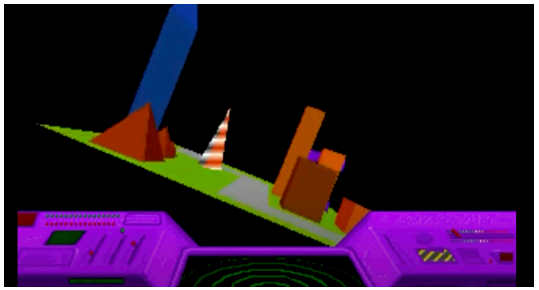
The miss rate is 10/10 = 100%.

Why?

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

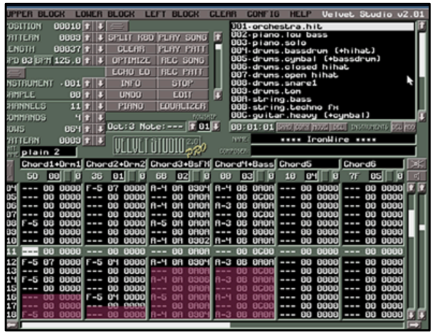
Why learning assembler?

Why understanding hardware architectures?



Competing with demos (1993)

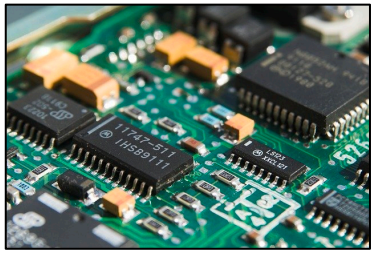
https://www.youtube.com/watch?v=Ow9bFnLwj3w



Music Tracker (1997)

https://people.kth.se/~dbro/software.html

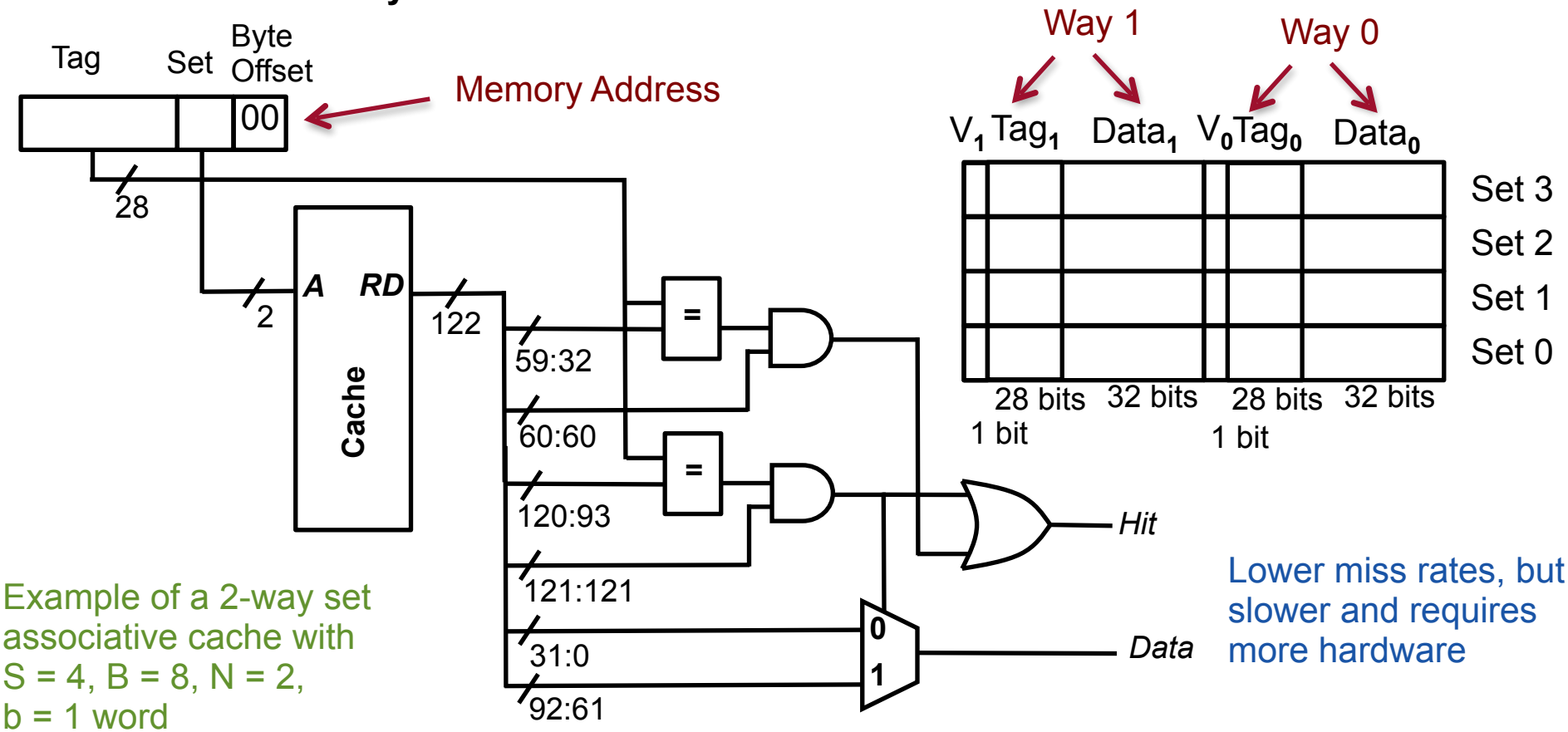
Don't view a computer as a black box



Understand the fundamentals:
- Compiler or OS development
- Performance optimization at higher levels
- This is what the university is about: the fundamentals

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

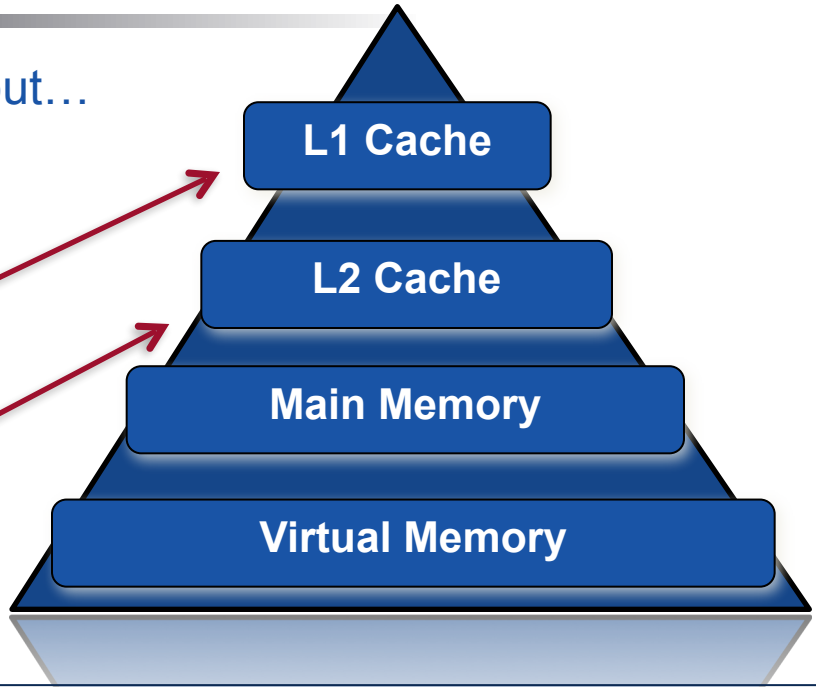
Large caches tend give lower miss rates, but…

Large caches tend to be slower.

Solution: Multi-Level Caches

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

The L2 cache 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

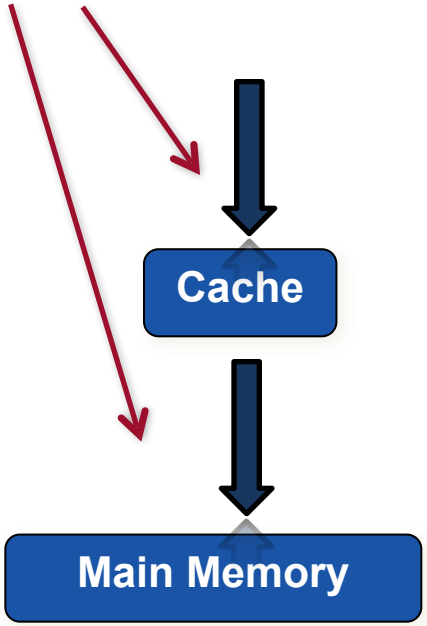
Intel Core-I7 920

- 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.

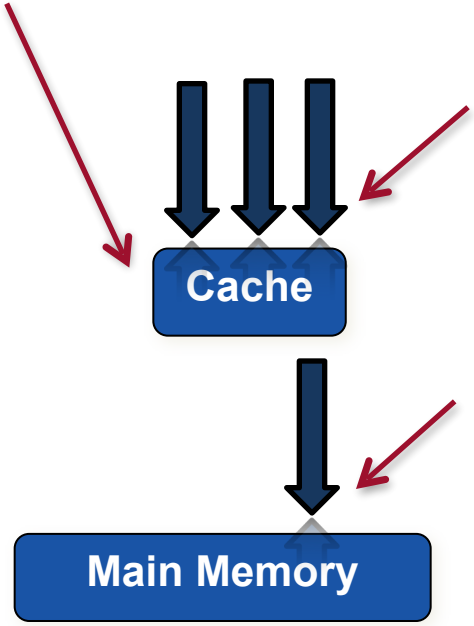


Write-Back Policy

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.

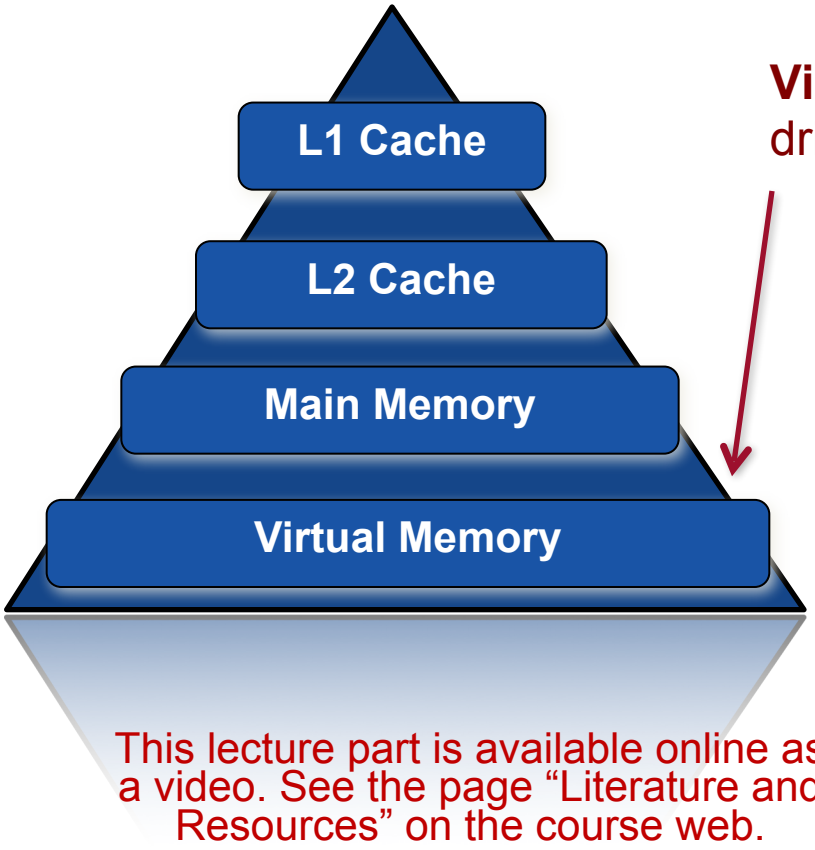
Data is written back to main memory when evicted and D = 1



Part III

Virtual Memory

Rationales for Virtual Memory



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

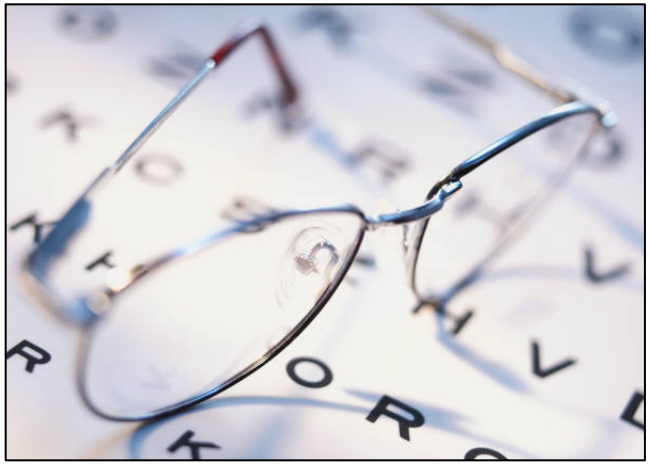
This lecture part is available online as a video. See the page “Literature and Resources” on the course web.

Why?

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

It’s soon time for coffee…

Reading Guidelines



Module 5: Memory Hierarchy

- H&H Chapters 5.5.1-5.5.6, 8.1-8.3
- H&H Chapter 8.4 (except 8.4.6)

Reading Guidelines
See the course webpage for more information.

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!