TDTS08:
Advanced Computer Architecture

Lesson

LINKÖPING UNIVERSITY

Outline

- Lab organization and goals
- SimpleScalar architecture and tools
- Lab 5: article review
- Exercises

Organization

- Assistant: Yungang Pan
- Web page
  - http://www.ida.liu.se/~TDTS08
  - Check the lab page!

Organization

- Sign up in Webreg latest Sept. 13 (**This Friday!**).
- Deadline for the assignments:

Lab 1, Lab 2	Oct. 2
Lab 3, Lab 4	Oct. 23
Lab 5	Nov. 8

- Rules: Read them!

Examination

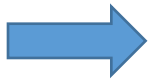
Written report for each lab:

- Demonstration
- Hand in the report, in PDF or DOC format
- Submit the report via **Email** Yungang Pan

Subject: TDTS08-LAB*x*-A/B*x*

Labs

- Five labs:
  1. Cache Memories (2 lab sessions)
  2. Instruction Pipelining (2 lab sessions)
  3. Superscalar Processors (2 lab sessions)
  4. VLIW processors (2 lab sessions)
  5. Article review on multiprocessor and multi-computer systems (no lab session)

Theoretical knowledge  Simulation  Analysis

Remote

- Thinlinc client: thinlinc.edu.liu.se

- SSH client: ssh.edu.liu.se

Note! Two-step verification is required to use remote login.

Environment

- Linux
- Simulations are started from a command line (i.e., terminal)
  - To open a new terminal you can press ctrl+alt+t
- Get yourself familiarized with the terminal
  - Ask Google first
  - Ask your assistant
- Make sure you learn the basic commands (i.e., *cd, ls, cp*, …)

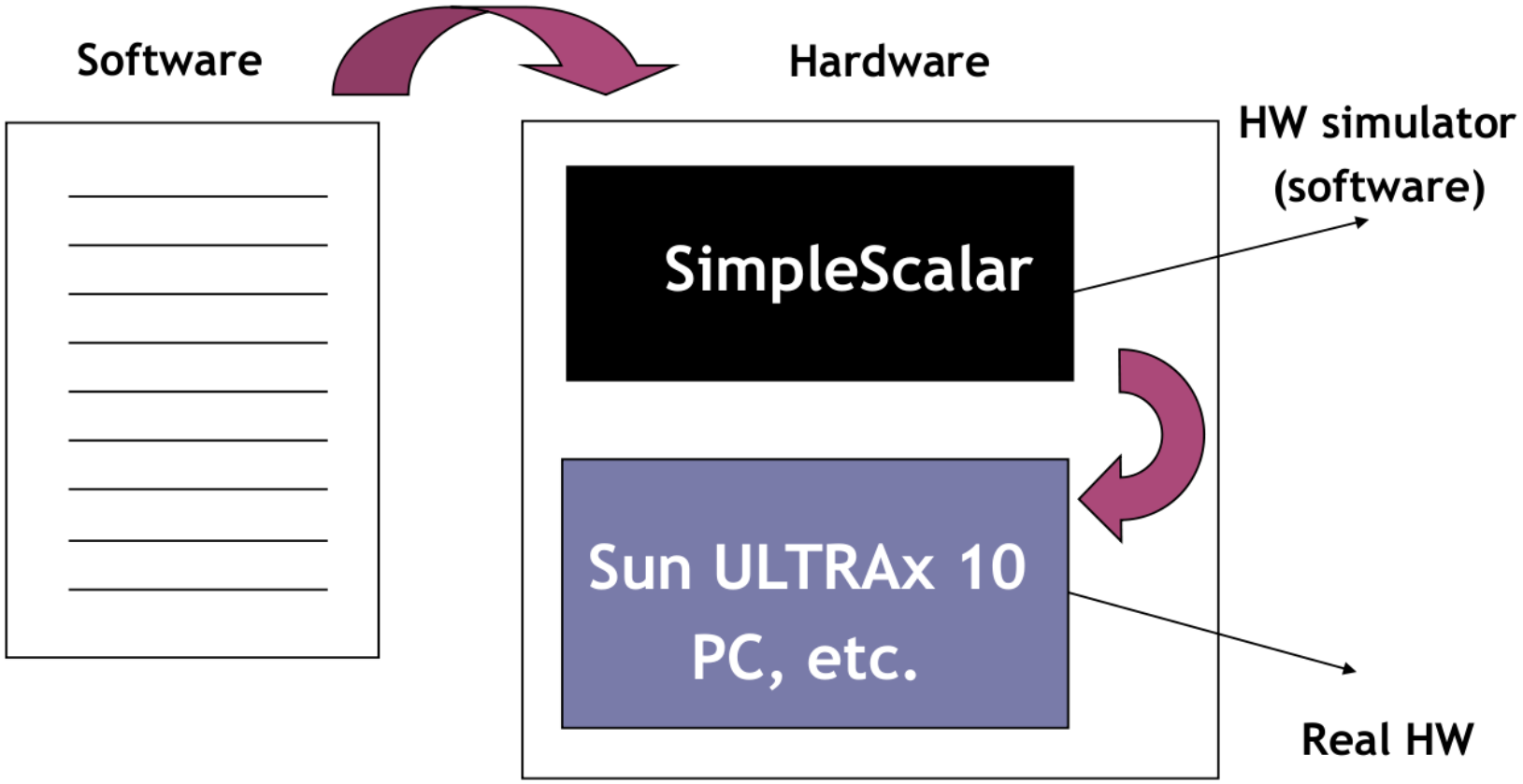
Tool Setup

- Don’t forget the instructions in lab0
- Instructions should be clear and easy to follow, but if you face difficulties
  - Don’t get frustrated :)
  - Read again carefully (without skipping over the lines)
  - Consult your assistant

Outline

- Lab organization and goals
- SimpleScalar architecture and tools
- Lab 5: article review
- Exercises

Architecture Simulation

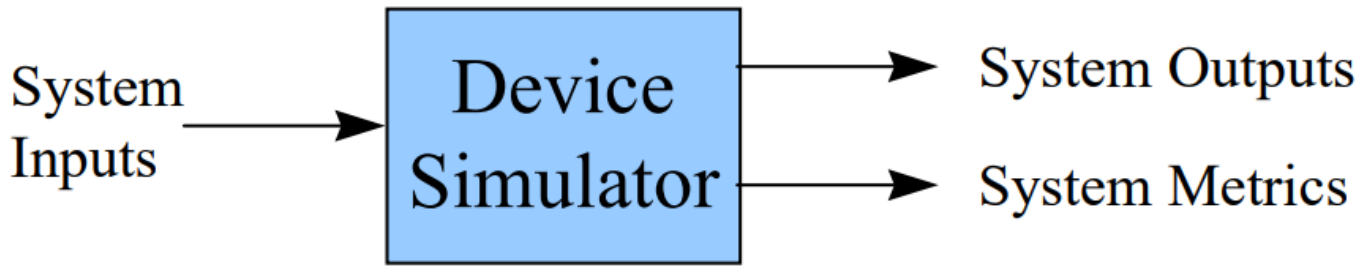


SimpleScalar: Literature

- "*The SimpleScalar Tool Set, Version 2.0*", by Doug Burger and Todd M. Austin
  - Very important preparation for the labs
  - This is your main reference for the tool!
- "User's and Hacker's guide", slides by Austin

A Computer Architecture Simulator Primer

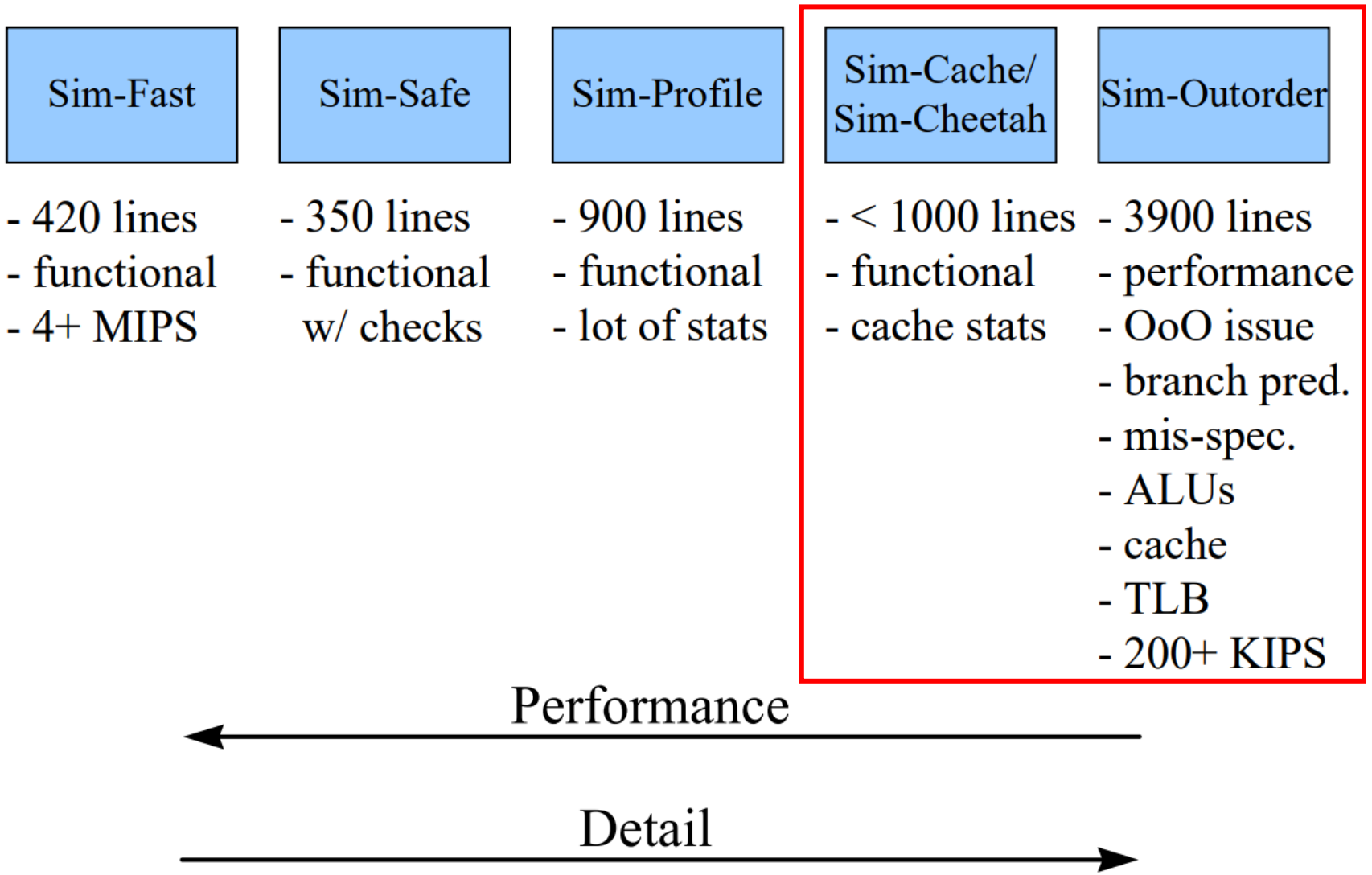
- What is an architectural simulator?
  - a tool that reproduces the behavior of a computing device



- Why use a simulator?
  - leverage faster, more flexible S/W development cycle
    - permits more design space exploration
    - facilitates validation before H/W becomes available
    - level of abstraction can be throttled to design task
    - possible to increase/improve system instrumentation

Source: https://www.cs.virginia.edu/~skadron/cs654/slides/hack_guide.pdf

Simulation Suite Overview



Source: https://www.cs.virginia.edu/~skadron/cs654/slides/hack_guide.pdf

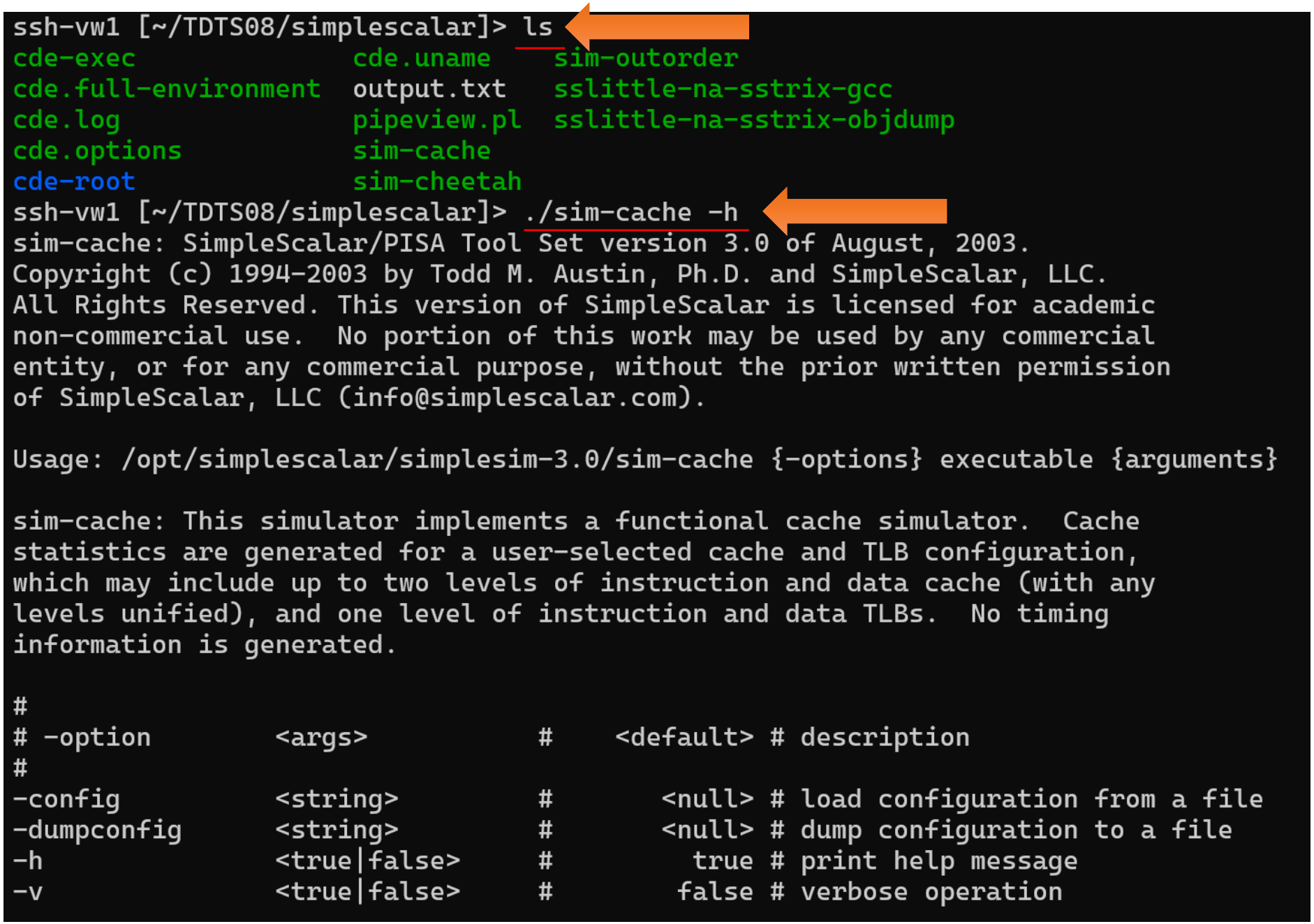
Global Simulator Options

- supported on all simulators:

`-h`	- print simulator help message
`-d`	- enable debug message
`-i`	- start up in DLite! debugger
`-q`	- terminate immediately (use with `-dumpconfig`)
`-config <file>`	- read configuration parameters from `<file>`
`-dumpconfig <file>`	- save configuration parameters into `<file>`

- configuration files:
  - to generate a configuration file:
    - specify non-default options on command line
    - and, include "`-dumpconfig <file>`" to generate configuration file
  - comments allowed in configuration files:
    - text after "#" ignored until end of line
  - reload configuration files using "`-config <file>`"
  - config files may reference other configuration files

Source: https://www.cs.virginia.edu/~skadron/cs654/slides/hack_guide.pdf


ssh-vw1 [~/TDTS08/simplescalar]> ls
cde-exec
cde.full-environment
cde.log
cde.options
cde-root
cde.uname
output.txt
pipeview.pl
sim-cache
sim-cheetah
sim-outorder
sslittle-na-sstrix-gcc
sslittle-na-sstrix-objdump
ssh-vw1 [~/TDTS08/simplescalar]> ./sim-cache -h
sim-cache: SimpleScalar/PISA Tool Set version 3.0 of August, 2003.
Copyright (c) 1994-2003 by Todd M. Austin, Ph.D. and SimpleScalar, LLC.
All Rights Reserved. This version of SimpleScalar is licensed for academic
non-commercial use. No portion of this work may be used by any commercial
entity, or for any commercial purpose, without the prior written permission
of SimpleScalar, LLC (info@simplescalar.com).
Usage: /opt/simplescalar/simplesim-3.0/sim-cache {-options} executable {arguments}
sim-cache: This simulator implements a functional cache simulator. Cache
statistics are generated for a user-selected cache and TLB configuration,
which may include up to two levels of instruction and data cache (with any
levels unified), and one level of instruction and data TLBs. No timing
information is generated.
-option <args> # <default> # description
-config <string> # <null> # load configuration from a file
-dumpconfig <string> # <null> # dump configuration to a file
-h <true|false> # true # print help message
-v <true|false> # false # verbose operation

Sim-Cache: Multi-level Cache Simulator

- generates one- and two-level cache hierarchy statistics and profiles
- extra options (also supported on sim-outorder):

`-cache:dl1 <config>` - level 1 data cache configuration
`-cache:dl2 <config>` - level 2 data cache configuration
`-cache:il1 <config>` - level 1 instruction cache configuration
`-cache:il2 <config>` - level 2 instruction cache configuration
`-tlb:dtlb <config>` - data TLB configuration
`-tlb:itlb <config>` - instruction TLB configuration
`-flush <config>` - flush caches on system calls
`-icompress` - remaps 64-bit inst addresses to 32-bit equiv.
`-pcstat <stat>` - record statistic `<stat>` by text address

Source: https://www.cs.virginia.edu/~skadron/cs654/slides/hack_guide.pdf

Specifying Cache Configurations

- all caches and TLB configurations specified with same format:

```
<name>:<nsets>:<bsize>:<assoc>:<repl>
```

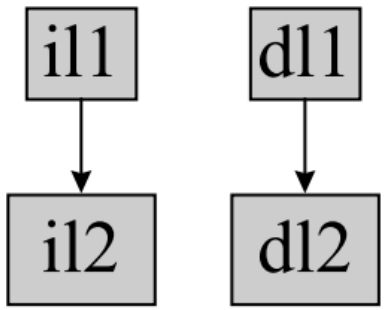
- where:
  - `<name>` - cache name (make this unique)
  - `<nsets>` - number of sets
  - `<assoc>` - associativity (number of "ways")
  - `<repl>` - set replacement policy
    - `l` - for LRU
    - `f` - for FIFO
    - `r` - for RANDOM
- examples:

`il1:1024:32:2:l`	2-way set-assoc 64k-byte cache, LRU

Source: https://www.cs.virginia.edu/~skadron/cs654/slides/hack_guide.pdf

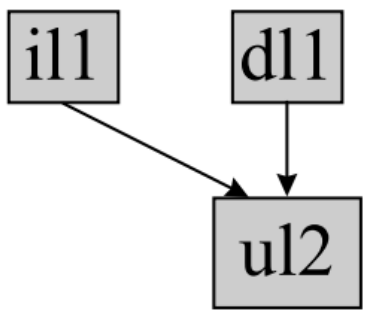
Specifying Cache Hierarchies

- specify all cache parameters in no unified levels exist, e.g.,




```
-cache:il1 il1:128:64:1:l -cache:il2 il2:128:64:4:l
-cache:dl1 dl1:256:32:1:l -cache:dl2 dl2:1024:64:2:l
```

- to unify any level of the hierarchy, “point” an I-cache level into the data cache hierarchy:




```
-cache:il1 il1:128:64:1:l -cache:il2 dl2
-cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
```

Source: https://www.cs.virginia.edu/~skadron/cs654/slides/hack_guide.pdf

Sim-Cheetah: Multi-Config Cache Simulator

- generates cache statistics and profiles for multiple cache configurations in a single program execution

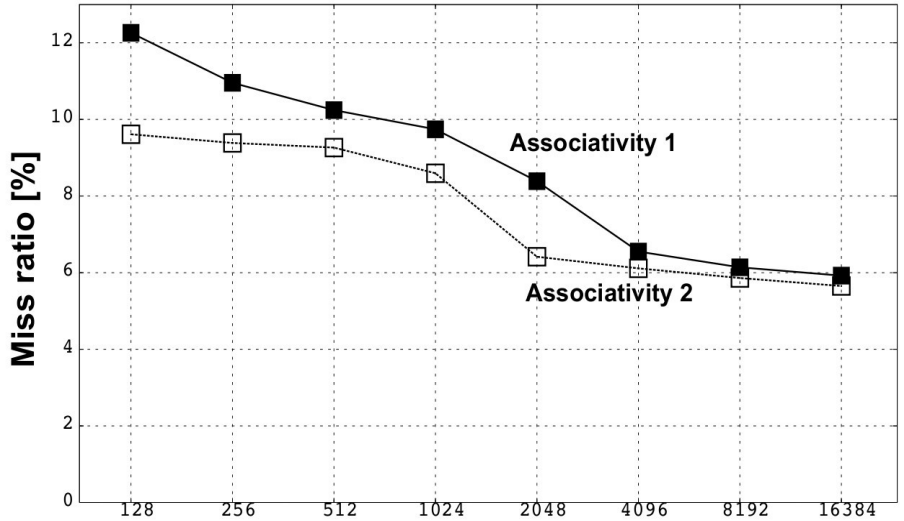
- extra options:

`-refs {inst,data,unified}`	- specify reference stream to analyze
`-C {fa,sa,dm}`	- cache config. i.e., fully or set-assoc or direct
`-R {lru, opt}`	- replacement policy
`-a <sets>`	- log base 2 number of set in minimum config
`-b <sets>`	- log base 2 number of set in maximum config
`-l <line>`	- cache line size in bytes
`-n <assoc>`	- maximum associativity to analyze (log base 2)
`-in <interval>`	- cache size interval for fully-assoc analyses
`-M <size>`	- maximum cache size of interest
`-c <size>`	- cache size for direct-mapped analyses

Source: https://www.cs.virginia.edu/~skadron/cs654/slides/hack_guide.pdf

An Example

- Lab1, assignment 3
  - Dump the default configuration of sim-cheetah
  - Modify the configuration and simulate
  - Plot the results (e.g. OpenOffice, Gnuplot, Matlab, Excel)



Outline

- Lab organization and goals
- SimpleScalar architecture and tools
- Lab 5: article review
- Exercises

Lab 5: Article Review

- Select an article on a multi-core, multiprocessor, multi-computer system, or a graphics processor
  - List of papers is available on the course page
  - You may select other articles if your lab assistant agrees
- Review the selected article
- Write a review report on the article
- Self-learning based; No lab session allocated
- Read and understand the paper
  - If the course literature does not help you, investigate the referenced papers

Lab 5: Article Review (cont'd)

- Analyze the paper
- Classify the architecture (e.g. MIMD, SIMD, NUMA)
- Possible questions to ask
  - Why has the actual method/approach been selected?
  - What are the advantages and disadvantages?
  - What is the application area?
  - What has been demonstrated?
  - …

Lab 5: Article Review (cont’d)

- Write a report
  - ~1000 words
  - Submit, in PDF format, to your lab assistant's **urkund** account (yungang.pan.liu@analys.urkund.se)

Outline

- Lab organization and goals
- SimpleScalar architecture and tools
- Lab 5: article review
- Exercises

Exercises

Problem 1. Review questions

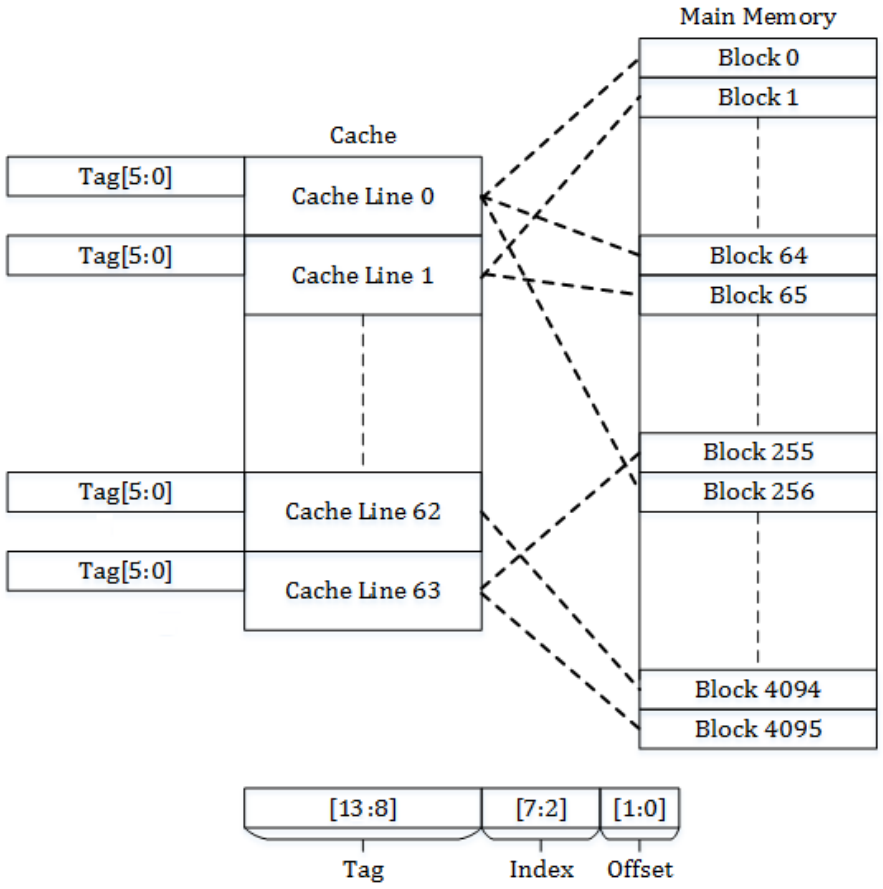
Problem 2 and 3. Mandatory for lab 1

Problem 4 and 5. Additional exercises (if you feel up to the challenge)

Problem 1 (review questions)

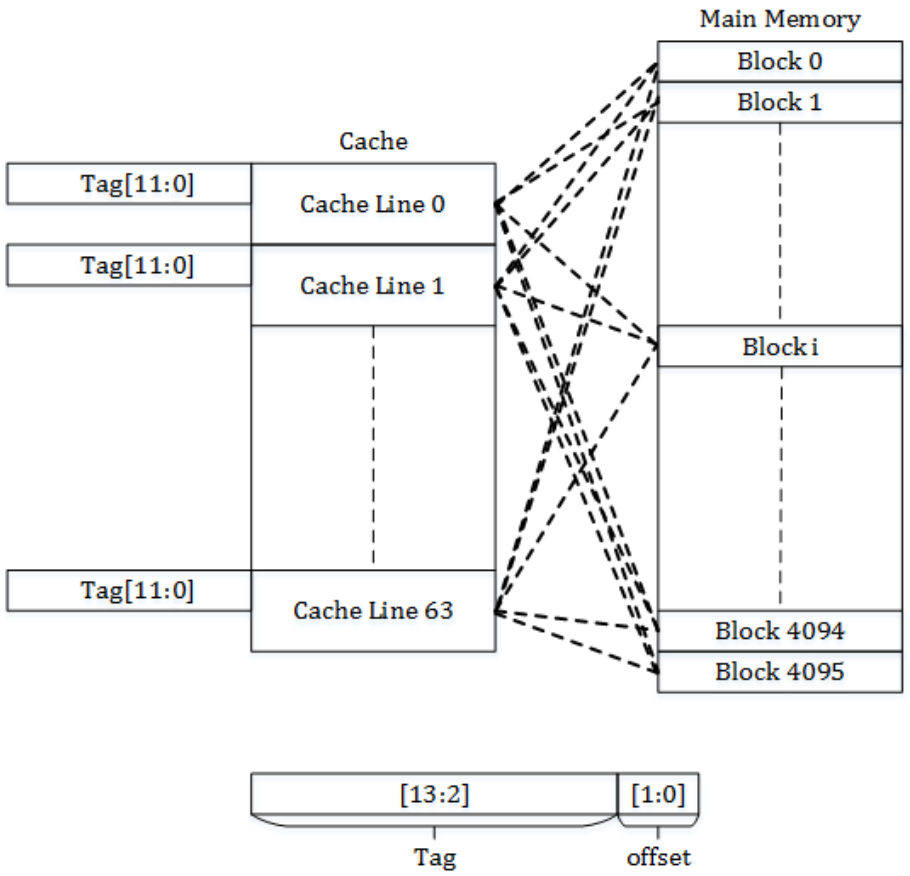
1) What are the differences among direct mapping, associative mapping, and set-associative mapping?
2) What is the distinction between spatial locality and temporal locality?

Cache (Direct-Mapped)



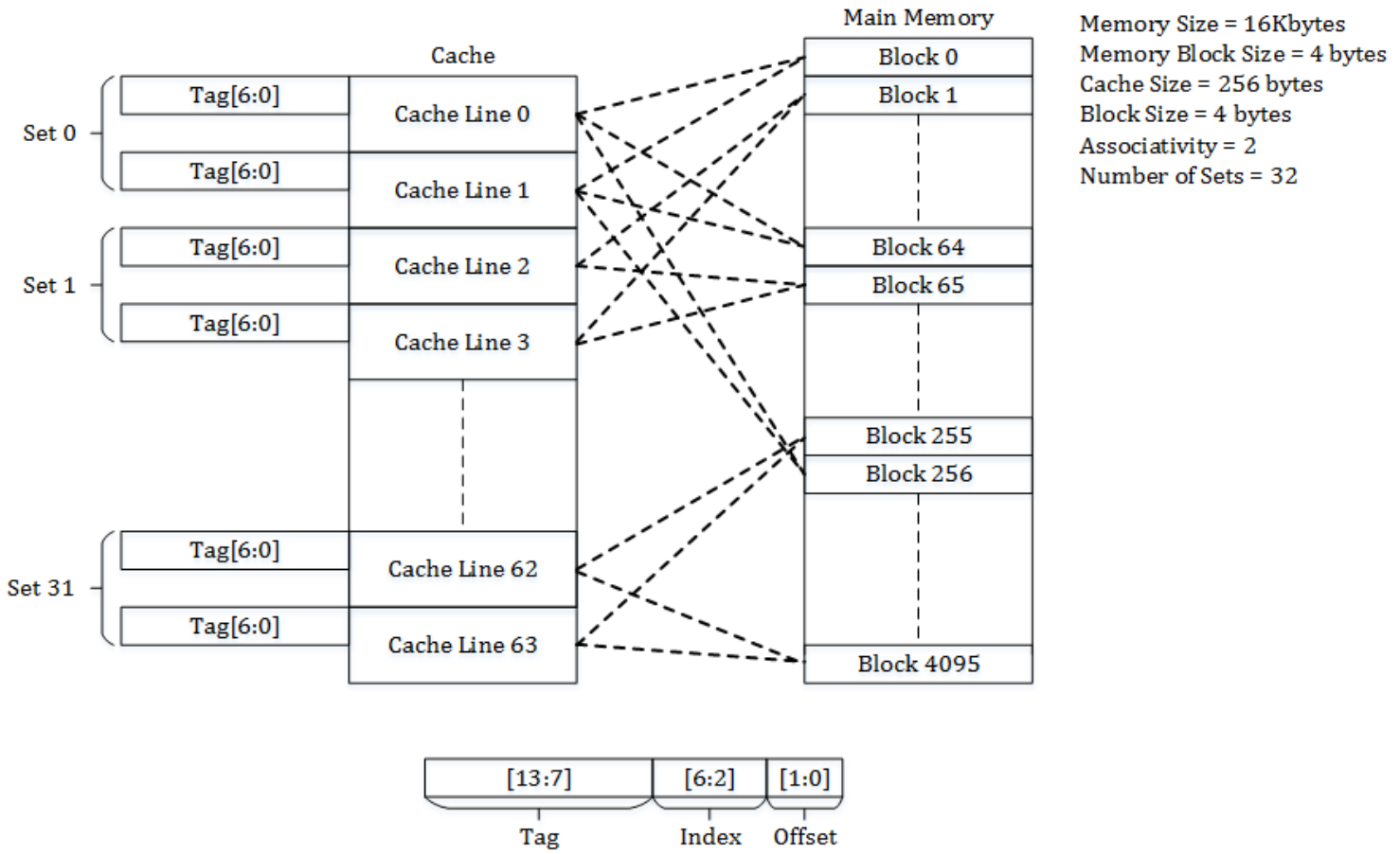
Memory Size = 16Kbytes
Memory Block Size = 4 bytes
Cache Size = 256 bytes
Block Size = 4 bytes
Associativity = 1
Number of Sets = 64

Cache (Fully-Associative)



Memory Size = 16Kbytes
Memory Block Size = 4 bytes
Cache Size = 256 bytes
Block Size = 4 bytes
Number of Cache Lines = 64

Cache (Set-Associative)



Memory Size = 16Kbytes
Memory Block Size = 4 bytes
Cache Size = 256 bytes
Block Size = 4 bytes
Associativity = 2
Number of Sets = 32

Direct Mapping

Cache Address Structure

Memory Cache Parameters

Memory Size 32MB

Cache Size 64KB

Block Size 2B

Cache Scheme

Direct Mapping

Set Associative

Set Size 4 blocks

Show

HELP

Return to Main Menu

Address Bit Partitioning

TAG									INDEX													OFFSET
24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Compare Bits									Set Select Bits															Byte Select Bits

The Compare Bits are compared with the corresponding Tag Bits in the Cache Directory.
The Set Select Bits are used to select a particular Set in the Cache.
The Byte Select Bits are used to select a particular byte in the accessed block.

Memory size = 32MB = 2^{25}

Block size = 2Bytes = 2^{1}

Number of blocks in cache = Cache size/Block size = 64KB/2B = $2^{16}/2^{1} = 2^{15}$

Number of bits in Tag = Total bits - Index bits - Offset bits = 25-15-1 = 9

Set Associative Mapping

Cache Address Structure

Memory Cache Parameters

Memory Size 32MB

Cache Size 64KB

Block Size 32B

Cache Scheme

Direct Mapping ○

Set Associative ◉

Set Size 4 blocks

Show

HELP

Return to Main Menu

Address Bit Partitioning

TAG											INDEX									OFFSET				
24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Compare Bits											Set Select Bits									Byte Select Bits				

The Compare Bits are compared with the corresponding Tag Bits in the Cache Directory.
The Set Select Bits are used to select a particular Set in the Cache.
The Byte Select Bits are used to select a particular byte in the accessed block.

Memory size = 32MB = 2^{25}

Block size = 32Bytes = 2^{5}

Number of sets in cache = Cache size/(Set size * Block size) = 64KB/(4 blocks * 32B) = $2^{16}/(2^{2} * 2^{5}) = 2^{9}$

Number of bits in Tag = Total bits - Index bits - Offset bits = 25-9-5 = 11

Question 2

Principle of Locality

Program instructions access a small proportion of their address space at any time

Question 2

Principle of Locality

Program instructions access a small proportion of their address space at any time

- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon

Question 2

Principle of Locality

Program instructions access a small proportion of their address space at any time

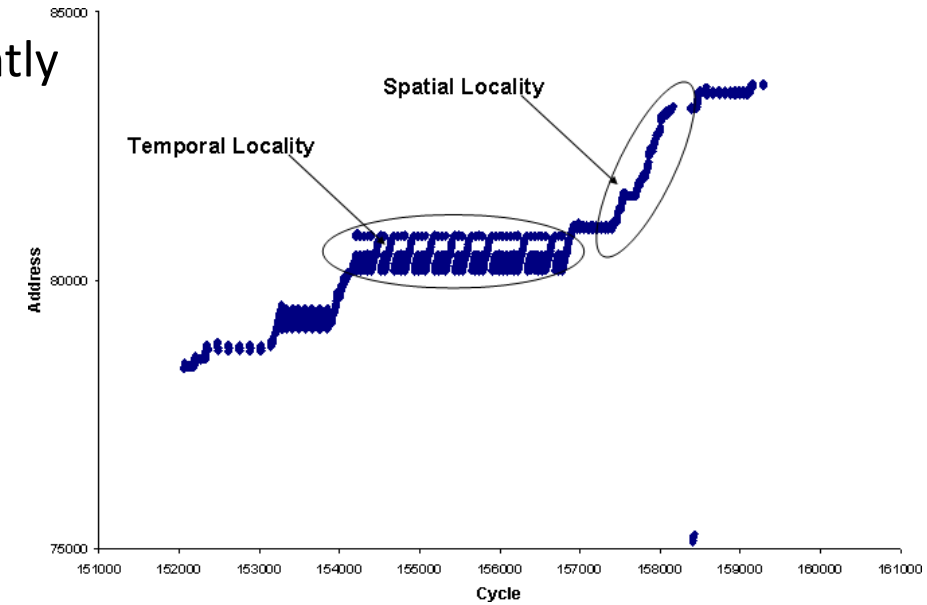
- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon
- **Spatial locality**
  - Items near those accessed recently
    are likely to be accessed soon

Question 2

Principle of Locality

Program instructions access a small proportion of their address space at any time

- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon
- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon



Problem 2 (mandatory for lab 1)

Consider a machine with a byte addressable main memory of 2^8 bytes and block size of 4 bytes. Assume that a direct mapped cache consisting of 8 lines is used with this machine.

1) How is an 8-bit memory address divided into tag, line number, and byte number?
2) Into what line would bytes with each of the following addresses be stored?

   0001 1011
   0011 0100
   1101 0000
   1010 1010

3) Suppose the byte with address 1010 0001 is stored in the cache. What are the addresses of the other bytes stored along with it?
4) How many total bytes of memory can be stored in the cache?
5) Why is the tag also stored in the cache?

Problem 2 (mandatory for lab 1)

Consider a machine with a byte addressable main memory of 2^8 bytes and block size of 4 bytes. Assume that a direct mapped cache consisting of 8 lines is used with this machine.

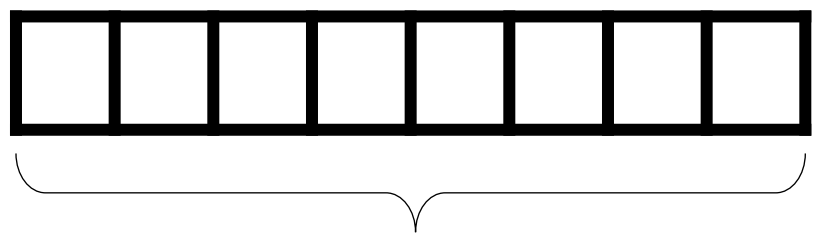
1) How is an 8-bit memory address divided into tag, line number, and byte number?
2) Into what line would bytes with each of the following addresses be stored?

0001 1011

0011 0100

1101 0000

1010 1010



Problem 2 (mandatory for lab 1)

Consider a machine with a byte addressable main memory of 2^8 bytes and block size of 4 bytes. Assume that a direct mapped cache consisting of 8 lines is used with this machine.

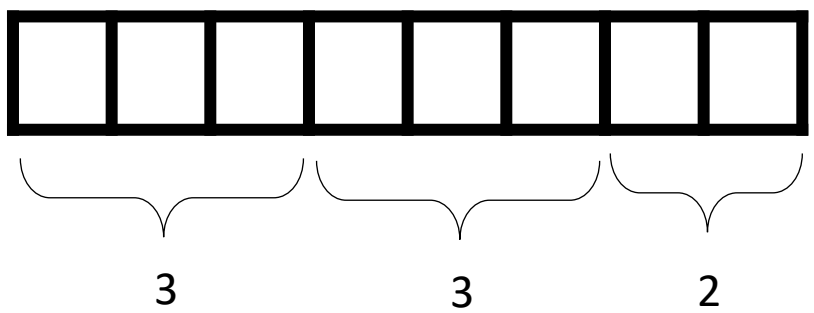
1) How is an 8-bit memory address divided into tag, line number, and byte number?
2) Into what line would bytes with each of the following addresses be stored?

0001 1011

0011 0100

1101 0000

1010 1010

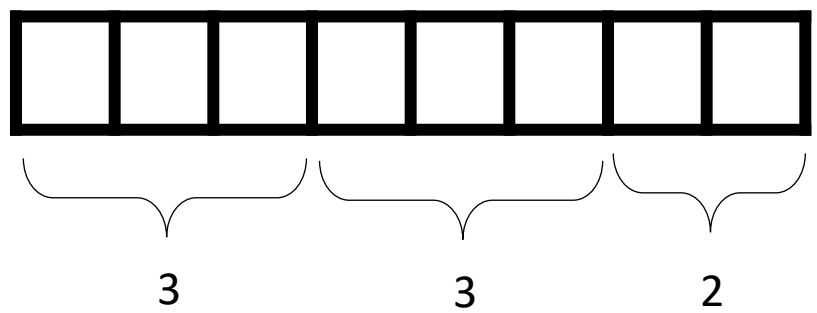


Problem 2 (mandatory for lab 1)

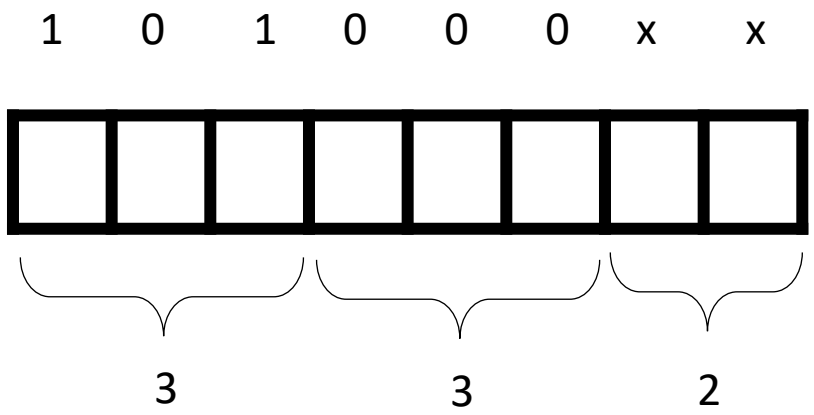
Consider a machine with a byte addressable main memory of 2^8 bytes and block size of 4 bytes. Assume that a direct mapped cache consisting of 8 lines is used with this machine.

1) How is an 8-bit memory address divided into tag, line number, and byte number?
2) Into what line would bytes with each of the following addresses be stored?

0001 1011
0011 0100
1101 0000
1010 1010

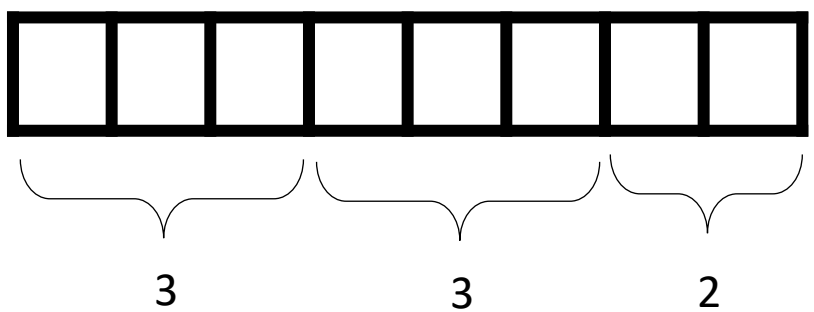


3) Suppose the byte with address 1010 0001 is stored in the cache. What are the addresses of the other bytes stored along with it?
4) How many total bytes of memory can be stored in the cache?
5) Why is the tag also stored in the cache?

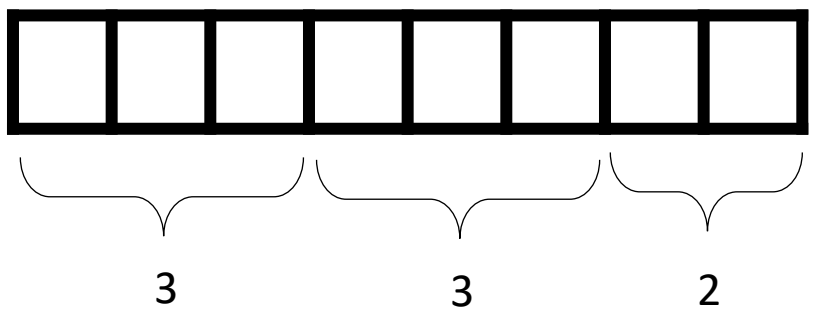


3) Suppose the byte with address 1010 0001 is stored in the cache. What are the addresses of the other bytes stored along with it?
4) How many total bytes of memory can be stored in the cache?
5) Why is the tag also stored in the cache?

8 lines x 4 bytes = 32 bytes

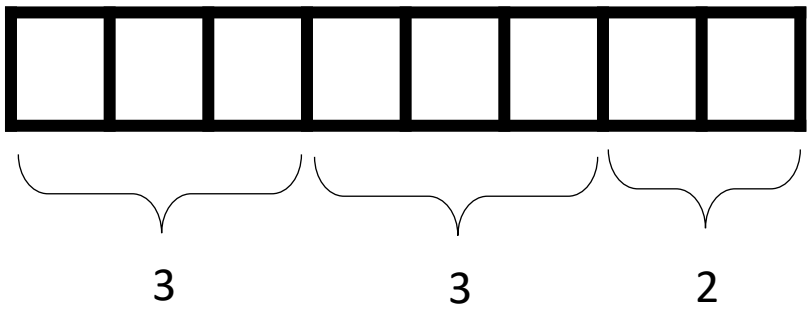


3) Suppose the byte with address 1010 0001 is stored in the cache. What are the addresses of the other bytes stored along with it?
4) How many total bytes of memory can be stored in the cache?
5) Why is the tag also stored in the cache?



3) Suppose the byte with address 1010 0001 is stored in the cache. What are the addresses of the other bytes stored along with it?
4) How many total bytes of memory can be stored in the cache?
5) Why is the tag also stored in the cache?

Because we have a large main memory but a limited and finite set of cache lines. More than one address go in a particular cache line. We need tag to identify which block is in the cache line.



Problem 3 (mandatory for lab 1)

Consider the following code:

```
cout << "Hello World";
cin >> a;
for(i = 0; i < 50; i++)
    cout<<i;
```

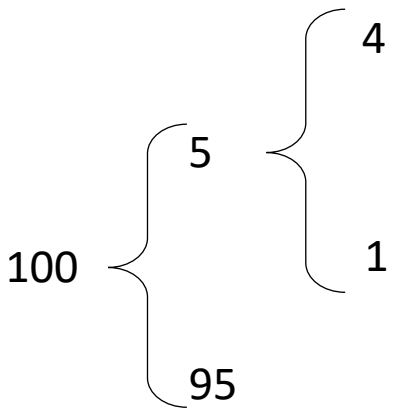
1) Give one example of the spatial locality in the code.
2) Give one example of the temporal locality in the code.

Problem 4 (additional)

Average memory-access time. A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 15 ns are required to access it. If it is in main memory but not in the cache, 70 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 10 ms are required to fetch the word from disk, followed by 50 ns to copy it to the cache, and the reference is started again. The cache hit ratio is 0.95 and the main memory hit ratio is 0.8. what is the average time in nanoseconds required to access a referenced word on this system?

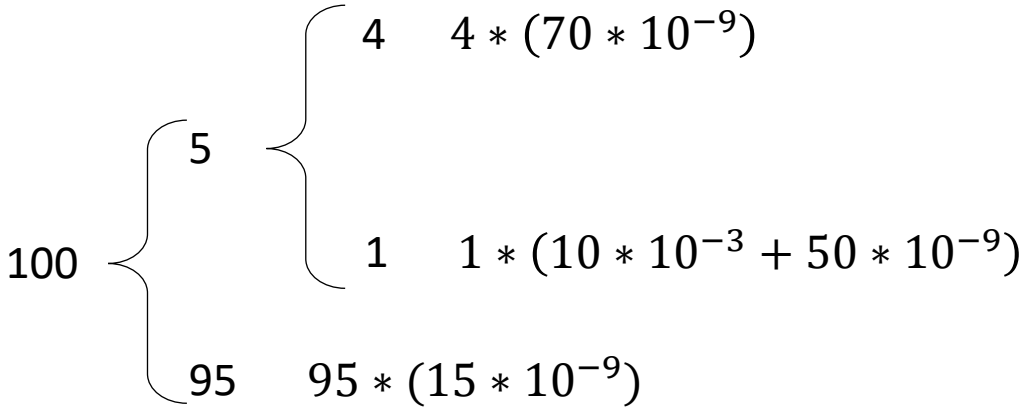
Problem 4 (additional)

Average memory-access time. A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 15 ns are required to access it. If it is in main memory but not in the cache, 70 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 10 ms are required to fetch the word from disk, followed by 50 ns to copy it to the cache, and the reference is started again. The cache hit ratio is 0.95 and the main memory hit ratio is 0.8. what is the average time in nanoseconds required to access a referenced word on this system?



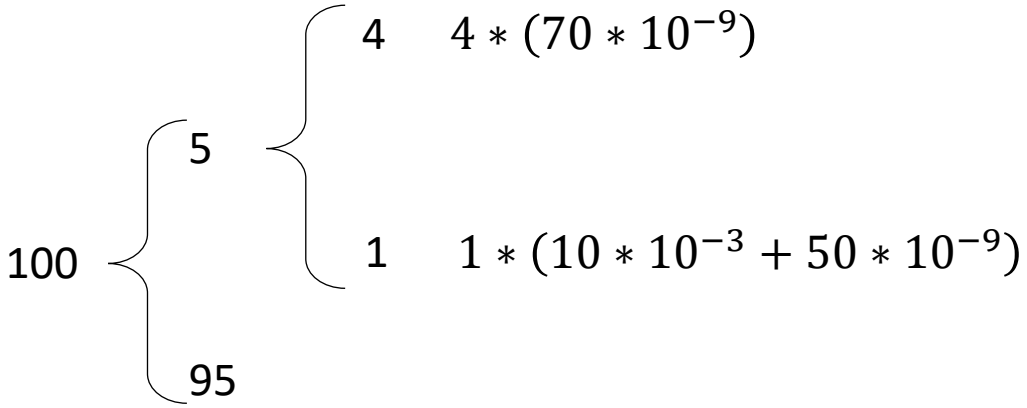
Problem 4 (additional)

Average memory-access time. A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 15 ns are required to access it. If it is in main memory but not in the cache, 70 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 10 ms are required to fetch the word from disk, followed by 50 ns to copy it to the cache, and the reference is started again. The cache hit ratio is 0.95 and the main memory hit ratio is 0.8. what is the average time in nanoseconds required to access a referenced word on this system?



Problem 4 (additional)

Average memory-access time. A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 15 ns are required to access it. If it is in main memory but not in the cache, 70 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 10 ms are required to fetch the word from disk, followed by 50 ns to copy it to the cache, and the reference is started again. The cache hit ratio is 0.95 and the main memory hit ratio is 0.8. what is the average time in nanoseconds required to access a referenced word on this system?



$$\frac{1 * (10 * 10^{-3} + 50 * 10^{-9}) + 4 * (70 * 10^{-9}) + 95 * (15 * 10^{-9})}{100}$$

Problem 5 (additional)

Performance enhancement using cache. A computer system contains a main memory of 32K 16-bit words. It also has a 4K-word cache divided into four-line sets with 64 words per line. Assume that the cache is initially empty. The processor fetches words from locations 0, 1, 2, …, 4351 in that order. If then repeats this fetch sequence nine more times. The cache is 10 times faster than main memory. Estimate the improvement resulting from the use of the cache. Assume an LRU policy for block replacement

Problem 5 (additional)

Performance enhancement using cache. A computer system contains a main memory of 32K 16-bit words. It also has a 4K-word cache divided into four-line sets with 64 words per line. Assume that the cache is initially empty. The processor fetches words from locations 0, 1, 2, …, 4351 in that order. If then repeats this fetch sequence nine more times. The cache is 10 times faster than main memory. Estimate the improvement resulting from the use of the cache. Assume an LRU policy for block replacement

4 x 1024 = 4 x 64 x #sets ➔ #sets = 16

set	tag	data	tag	data	tag	data	tag	data
0		64 word						
1								
2								
3								
	…	…	…	…	…	…	…	…
15								

Cache structure

set	tag	data	tag	data	tag	data	tag	data
0		[0-63]		[1024-1087]		[2048-2111]		[3072-4159]
1		[64-127]						
2								
3								
	...	...	...	...	...	...	...	...
15		[960-1023]		[1984-2047]		[3008-3071]		[4032-4095]

Cache structure

set	tag	data	tag	data	tag	data	tag	data
0		[4096-4159]		[1024-1087]		[2048-2111]		[3072-4159]
1		[4160-4223]						
2		[4224-4287]						
3		[4288-4351]						
	…	…	…	…	…	…	…	…
15		[960-1023]		[1984-2047]		[3008-3071]		[4032-4095]

Cache structure

1st round: 4 x 16 + 4 misses
2nd-9th round: 4 x 4 + 4 misses
Total misses: 4 x 16 + 4 + (4 x 4 + 4) x 9 = 248

With cache: 248 x 10s + (4351 x 10 – 248) x 1s = 45742
Without cache: 4351 x 10 x 10s = 435100

Speed-up:
435100/45742 ~= 9.5