CS 110
Computer Architecture

Advanced Caches

Instructor:
Sören Schwertfeger and Chundong Wang

https://robotics.shanghaitech.edu.cn/courses/ca/20s

School of Information Science and Technology SIST

ShanghaiTech University

Slides based on UC Berkeley's CS61C and Carnegie Mellon Univ. ECE447 (2015)

Midterm

- Date: Tuesday, May 26th, 2020
- Time: 10:15- 12:15 (normal lecture slot++)
  - Be there latest 10:00 – **we start 10:15 sharp**!
- Venue: 4 rooms – <u>check on egate which room you are</u>!
  - SPST1-503
    SPST1-201
    SPST1-501
    SIST1A-106
- Closed book:
  - You can bring **<u>two</u>** A4 pages with notes (both sides; in **<u>English</u>**): Write your Chinese and **Pinyin** name on the top! **<u>Handwritten</u>** by you!
    - **Final**: you can bring **three** A4 pages
  - You will be provided with the RISC-V ”green sheet”
  - No other material allowed!

Admin

Midterm I

- Wear your Corona mask! =>
- Switch cell phones **off**!
  (not silent mode – off!)
  - Put them in your bags.
- Bags under the table. Nothing except paper, pen, 1 drink, 1 snack, your student ID card on the table!
- No other electronic devices are allowed!
  - No ear plugs, music, smartwatch…
- Anybody touching any electronic device will **FAIL** the course!
- Anybody found cheating (copy your neighbors answers, additional material, ...) will **FAIL** the course!

Admin


CASIO
JF-120TM
TWO WAY POWER
F CUT 5/4
4 3 2 1 0 A
+/−
MRC
C
AC
ON
TAX−
RATE SET
%
TAX RATE
TAX+
GT
0
00
1
2
3
4
5
6
7
8
9
=
+
−
×

Admin

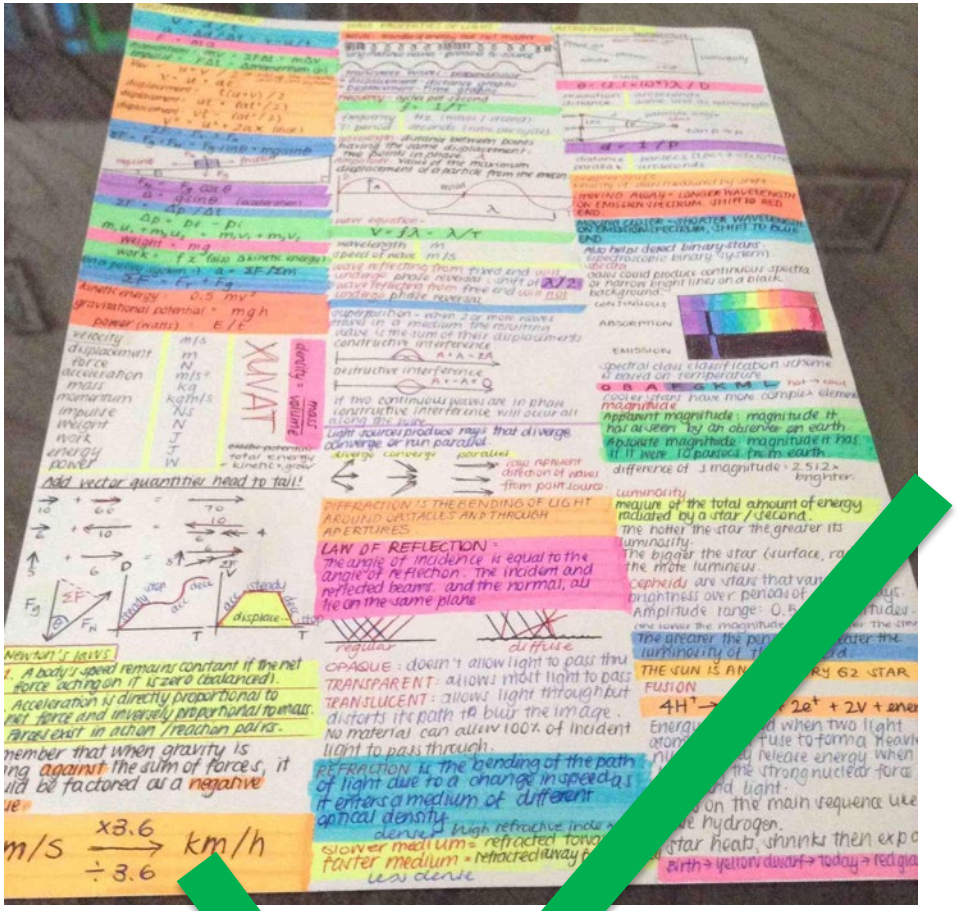
Admin

Admin

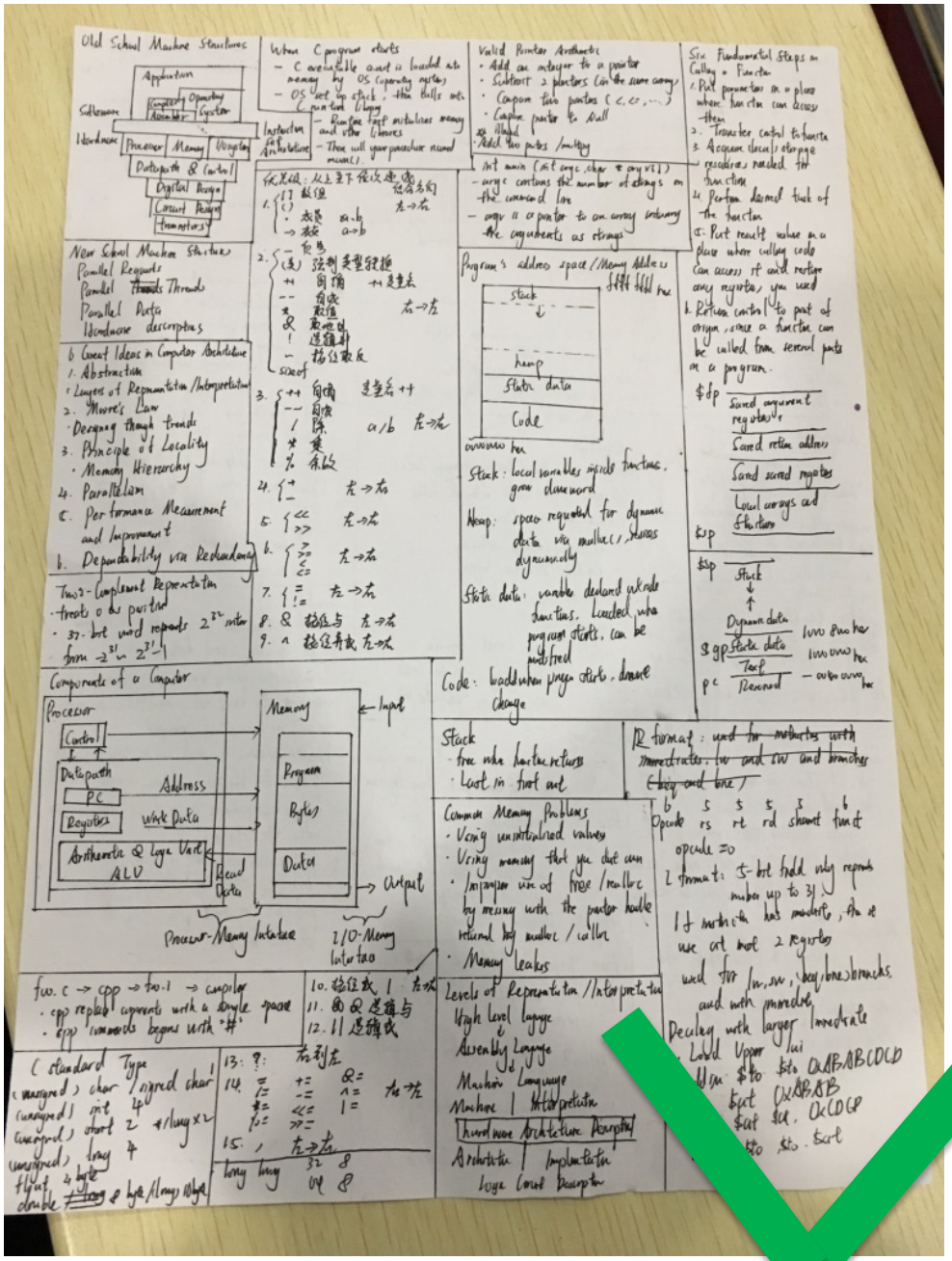

COMPUTER
ORGANIZATION
AND DESIGN
THE HARDWARE/SOFTWARE INTERFACE
RISC-V EDITION
MK
MORGAN KAUFMANN
DAVID A. PATTERSON
JOHN L. HENNESSY

Admin


FUNCTIONS OF SEVERAL VARIABLES
LEVEL CURVES
FUNCTION OF n VARIABLES
PARTIAL DERIVATIVES
SECOND PARTIAL DERIVATIVES
CLAIRAUT'S THEOREM
THE CHAIN RULE
IMPLICIT DIFFERENTIATION
TANGENT PLANE TO A LEVEL SURFACE
NORMAL LINE TO A LEVEL SURFACE
MAXIMUM AND MINIMUM VALUES
FINDING ABSOLUTE MAX. AND MINS.
MAXIMIZING AND MINIMIZING
DOMAINS: Allowed (x,y), (x,y,z) RANGES: 2D,3D's
ε-δ DEFINITION OF CONTINUITY
EQUATIONS OF TANGENT PLANES TO SURFACES
TOTAL DIFFERENTIAL
INCREMENTS Δx, Δy, Δz DIFFERENTIALS, dx, dy, dz
THEOREM
DEPENDENCY DIAGRAMS
THE GRADIENT VECTOR
DIRECTIONAL DERIVATIVES


Add vector quantities head to tail!
Newton's laws
LAW OF REFLECTION:
REFRACTION IS THE BENDING OF LIGHT AROUND OBSTACLES AND THROUGH APERTURES
OPAQUE: doesn't allow light to pass thru
TRANSPARENT: allows most light to pass
TRANSLUCENT: allows light through but distorts its path to blur the image.
No material can allow 100% of incident light to pass through.
REFRACTION is the bending of the path of light due to a change in speed as it enters a medium of different optical density
m/s
x3.6
÷3.6
km/h

Admin


Old School Machine Structures
Software
Hardware
Application
Operating System
Compiler
Assembler
Processor
Memory
Datapath & Control
Digital Design
Circuit Design
Transistors
New School Machine Structures
Parallel Requests
Parallel Threads
Parallel Data
Hardware descriptors
6 Great Ideas in Computer Architecture
1. Abstraction
2. Moore's Law
3. Principle of Locality
4. Parallelism
5. Performance Measurement and Improvement
6. Dependability via Redundancy
Two's-Complement Representation
Components of a Computer
Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit (ALU)
Memory
Input
Output
Processor-Memory Interface
I/O-Memory Interface
Stack
Heap
Static data
Code
Stack: local variables, spills functions, grow downward
Heap: space requested for dynamic data via malloc(); shrinks dynamically
Static data: variable declared outside functions. Loaded when program starts, can be modified
Code: loaded when program starts, doesn't change
Common Memory Problems
Using uninitialized values
Using memory that you don't own
Forgetting use of free/malloc by mixing with the pointer handle returned by malloc/calloc
Memory leaks
Levels of Representation/Interpretation
High Level Language
Assembly Language
Machine Language
Machine Interpretation
Hardware Architecture Description
Architecture Implementation
Logic Gate Description
Valid Pointer Arithmetic
Add an integer to a pointer
Subtract 2 pointers (in the same array)
Compare two pointers (<, >, ...)
Compare pointer to NULL
Add two pointers
Six Fundamental Steps in Calling a Function
Return control to parent
Acquire/release storage resources needed for function
Perform desired task of the function
Place result value in a place where calling code can access it and restore any registers you used
Return control to point of origin, since a function can be called from several points in a program.
Saved argument registers
Saved return address
Saved saved registers
Local arrays and structures
Stack
Dynamic data
Static data
Text
Reserved
Common Problems
Decrease with larger immediate
Lower Upper
Stack
Heap
Static
Text
Reserved

SIFT REFERENCE GUIDE (V.1.1) – CREATING TIMELINES WITH THE SIFT WORKSTATION

Admin

SANS COMPUTER FORENSICS and INCIDENT RESPONSE

THE PURPOSE OF THIS REFERENCE GUIDE IS TO WALK THROUGH THE PROCESS OF BOOTING THE SIFT WORKSTATION, CREATING A TIMELINE ("SUPER" OR "MICRO") AND REVIEWING IT.

puter-forensics11.sans.org/community/downloads

oad: SIFT Workstation VM Appliance

Do FT Workstation Installation

2

2. BOOT SIFT VM

Login: *sansforensics*
Password: *forensics*

3

$ sudo su

3. ELEVATE PRIVS

4

4. CONNECT IMAGE TO SIFT

Plug hard drive to physical host and attach to SIFT VM

5

5. HARD DRIVE MOUNTING *(if you are using log2timeline-sift and Single DD you can skip to 7-A)*

EWF/

SINGLE OR SPLIT IMAGE (2 options):

nt_ewf.py image.E01 /mnt/ewf

image.E01 /mnt/ewf/

Not Needed For 7-A

# mount -t ntfs -o ro,loop,show_sys_files,streams_interface=wind
offset=#### /mnt/ewf/<image> /mnt/windows_mount/

MOUNT TO MOUNT POINT

DD

SINGLE IM

mount -t ntfs -o w_sys_files,streams_interface=windows,offset=#### image.dd /m ws_mount/

SPLIT IMAGE (2 step pro

affuse image.001 /mnt/aff

mount –t ntfs-3g –o loop,ro,show_ mnt/aff/<image> /mnt/windows_

6

HOW TO CALCULATE THE OFFSET FOR MOUNTING

1. Run mmls to query partition layout
 # mmls image.E01
2. Identify partition and byte offset
3. (Partition byte offset) x (bytes per sector) = **offset #### to use!**
Example: 63 X 512 = 32256

Note: If needed, repeat for each partition. Make new mount point:
 # mkdir /mnt/windows_mount2/

6. log2timeline default timezone is set to examiner local host. To change use -z [TIMEZONE] option. To list all available timezones:
 # log2timeline -z list

7-A & 7-B

8

7-A: AUTOMATED SUPER TIMELINE CREATION

log2timeline-sift -o –z [TIMEZONE] -p [PARTITION #] -i [IMAGE FILE]

DISK IMAGE (prompt for partition, mount, and run):

XP # log2timeline-sift –z EST5EDT -i image

WIN7 # log2timeline-sift -win7 -z EST5EDT -i image

FOR PARTITION (mount and run using all applicable pl

XP # log2timeline-sift –z EST5EDT -p 0 -i parti

WIN7 # log2timeline-sift -win7 -z EST5EDT tion

OTHER USAGE EXAMPLES:

Display list of available plu
 # log2timeline -f list
Run log2timeline use use only specific plugins:
 # log2timeline-sift refetch –z EST5EDT -i image.dd
Help (man pag
 # log2timeli

8. CSV F PUT *(/cases/timeline-output-folder)*

-dat e event, in the format of MM/DD/YYYY
-t of day, expressed in a 24h format, HH:MM:SS
: the timezone that was used to call the tool with.
: MACB meaning of the fields, comp w/ mactime format.
rce: Source short name (i.e. registry entries are REG)
-sourcetype: Desc of the source ("Internet Explorer" instead of WEBHIST)
-type: Timestamp type (i.e. "Last Accessed", "Last Written")
-user: Username associated with the entry, if one is available.
-host: Hostname associated with the entry, if one is available.
-short: Contains less text than the full description field.
-desc: where majority info is stored, the actual parsed desc of the entry.
-version: Version number of the timestamp object.
-filename: Filename with the full path that contained the entry
-inode: inode number of the file being parsed.
-notes: Some input modules insert additional information in the form of a note, which comes here. Or it can be used during the review.
-format: Input module name used to parse the file.
-extra: Additional information parsed is joined together and put here.

ANUAL "MICRO" TIMELINE CREATION

TIONS] [-f FORMAT] [-z TIMEZONE] [-o OUTPUT MODULE] [-w
G_FILE/LOG_DIR [–] [FORMAT FILE OPTIONS]

EM METADATA (using log2timeline or fls)

stem data w/log2timeline from mounted file system:
 # log f mft -o mactime –r -z EST5EDT -w
mft.bo lume/
OR Extract M age using Sleuthkit:
 # fls -m "" -o offs dd > fls.body
Convert body file for format w/ mactime:
 # mactime –b fls.body –d

ARTIFACTS (run l2l on mounted file s h plugins recursively)

Extract artifacts w/ log2timeline and ru d file system:
 # log2timeline -f firefox3,chrome -o mactim EDT -w
web.body /mnt/volume/
Convert body file format to CSV format w/ mactime:
 # mactime –b log2timeline.body –d > log2timeline.csv

10

HELP? OPTIONS? USAGE?

log2timeline -help
Log2timeline-sift -help
L2t_process -help

OTHER log2timeline OUTPUT FORMATS

Note: CSV is Default Output
-BeeDocs - Mac OS X visualization tool
-CEF - Common Event Format - ArcSight
-CFTL - XML file- CyberForensics TimeLab visualization tool
-CSV - comma separated value file
-Mactime - Both older and newer version of the format supported for use by TSK's mactime
-SIMILE - XML file - SIMILE timeline visualization widget
-SQLite - SQLite database
-TLN - Tab Delimited File
-TLN - Format used by some of H Carvey tools, expressed as a ASCII output
-TLNX - Format used by some of H Carvey tools, expressed as a XML document

9

9. FILTER TIMELINE

Filter timeline with date range to include only:
l2t_process -b timeline.csv MM-DD-YYYY..MM-DD-YYYY > filtered.csv
Filter timeline with keyword list (one term per line in keywords.txt):
l2t_process -b timeline.csv -k keywords.txt > filtered.csv
What sources are in your timeline?
awk–F , '{print $6;}' timeline.csv| grep–v sourcetype|sort | uniq
Find all LNK files that reference E Drive
grep"Shortcut LNK" timeline.csv| grep"E:"
FiindMountPoints2 entries that reference E Drive
grep"MountPoints2 key" timeline.csv} | grep"E drive"
grepUSB timeline.csv| grep"SetupAPILog"

File System	M	A	C	B
Ext2/3	Modified	Accessed	Changed	N/A
FAT	Written	Accessed	N/A	Created
NTFS	File Modified	Accessed	MFT Modified	Created
UFS	Modified	Accessed	Changed	N/A

10. CONNECT TO SIFT

> SETTINGS -> OPTIONS -> Shared Folders -> Always Enabled (Check)

✓ 2. SIFT Desktop > VMware-Shared-Drive

✓ Access from a Win Machine \\SIFTWORKSTATION

11

11. REVIEW TIMELINE

Review timelines using:
- Open, Soft, Filter with Excel
- Import into SPLUNK
- SIMILE
- Tapestry

log2timeline PARSING PLUGINS

apache2_error - Apache2 error log file
chrome - Chrome history file
encase_dirlisting - CSV file that is exported from encase
evt - Windows 2k/XP/2k3 Event Log
evtx - Windows Event Log File (EVTX)
exif - Metadata information from files using ExifTool
ff_bookmark - Firefox bookmark file
firefox2 - Firefox 2 browser history
firefox3 - Firefox 3 history file
ftk_dirlisting - CSV file that is exported from FTK Imager (dirlisting)
generic_linux - Generic Linux logs that start with MMM DD HH:MM:SS
iehistory - index.dat file containg IE history
iis - IIS W3C log file
isatxt - ISA text export log file
jp_ntfs_change - CSV output file from JP (NTFS Change log)
mactime - Body file in the mactime format
mcafee - Log file
mft - NTFS MFT file
mssql_errlog - ERRORLOG file produced by MS SQL server
ntuser - NTUSER.DAT registry file
opera - Opera's global history file
oxml - OpenXML document pcap
pcap - PCAP file
pdf - Available PDF document metadata
prefetch - Prefetch directory
recycler - Recycle bin directory
restore 0.9 - Restore point directory
safari - Safari History.plist file
sam - SAM registry file
security - SECURITY registry file
setupapi - SetupAPI log file in Windows XP
skype_sql - Skype database
software - SOFTWARE registry file
sol - .sol (LSO) or a Flash cookie file
squid - Squid access log (http_emulate off)
syslog - Linux Syslog log file
system - SYSTEM registry file
tln - Body file in the TLN format
volatility - Volatility output files (psscan2, sockscan2, ...)
win_link - Windows shortcut file (or a link file)
wmiprov - wmiprov log file
xpfirewall - XP Firewall log

List plugins # log2timeline -f list
...HELP EXPAND THIS LIST. BUILD PLUGINS!!!

BY DAVID NIDES (12/16/2011)
TWITTER: @DAVNADS
BLOG: DAVNADS.BLOGSPOT.COM
EMAIL: DNIDES@KPMG.COM

CREDITS TO: ED GOINGS, ROB LE
KRISTINN GUDJONSSON, KPMG &
QUESTIONS/FEEDBACK–CONTACT US!

KEY
Red text – image/source
Blue text – mount point
Purple text – output file
Green text – log2timeline plugins
Brown text - TimeZone

Admin

Content

- Main topics: Everything till (including) Lecture 16
  - Number representation (int & float)
  - C
  - CALL
  - RISC-V
  - SDS; Datapath & Control
  - Pipelining & Superscalar
  - Caches
- Plus general ”Computer Architecture” knowledge

Quiz

Quiz on TLB

Piazza: "Video Lecture 22 VM"

- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB
- If we increase the page size, say, from 4KB to 16KB, can we increase the TLB reach?

A. Yes, of course
B. No, TLB reach is fixed given a processor and memory space

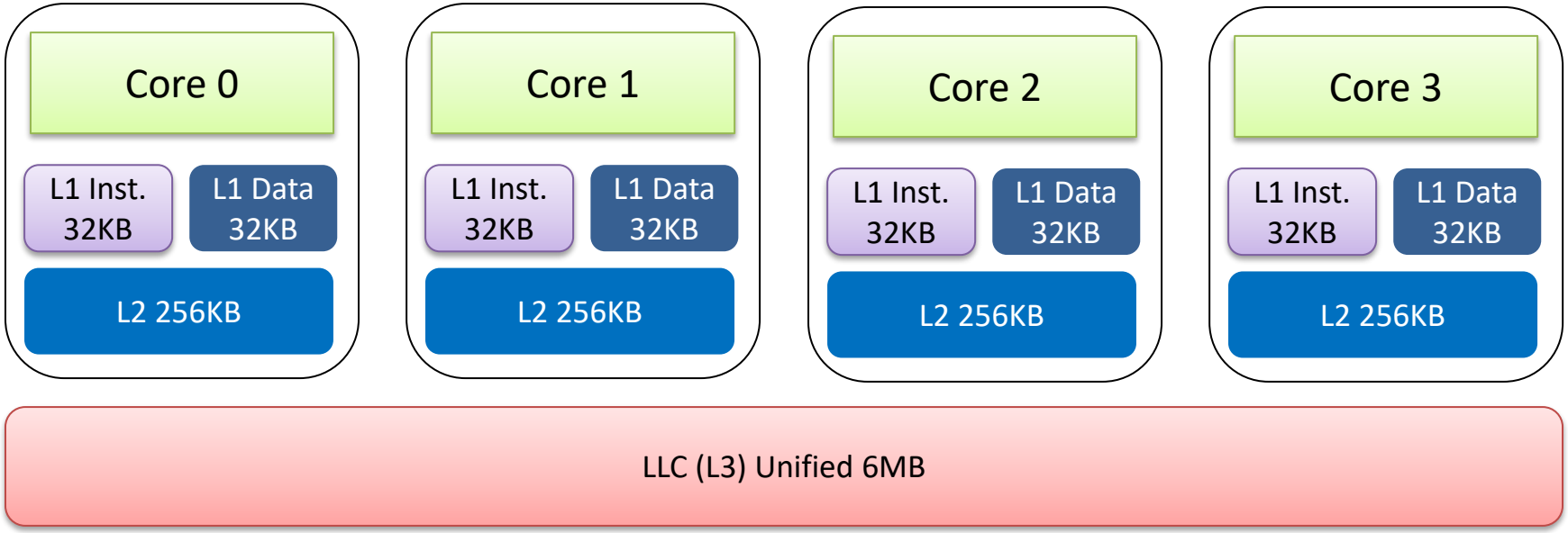
Outline

- MRU is LRU
- LLC is not monolithic
- Use a crystal ball to help cache
- No, do not cache us
- Yes, you can control the cache

MRU is LRU

Cache Inclusion

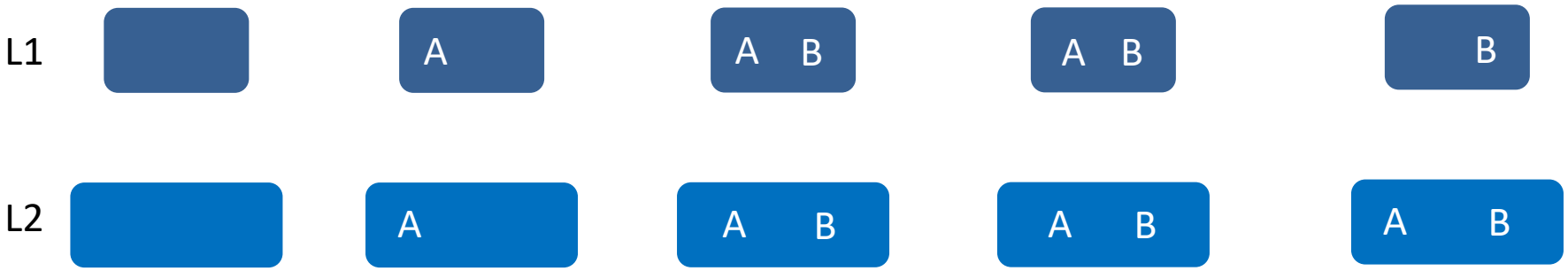
- Multilevel caches



Intel Ivy Bridge Cache Architecture (Core i5-3470)

If all blocks in the higher level cache are also present in the lower level cache, then the lower level cache is said to be **inclusive** of the higher level cache.

Inclusive



Initial state

Read A miss; load A into L1 and L2

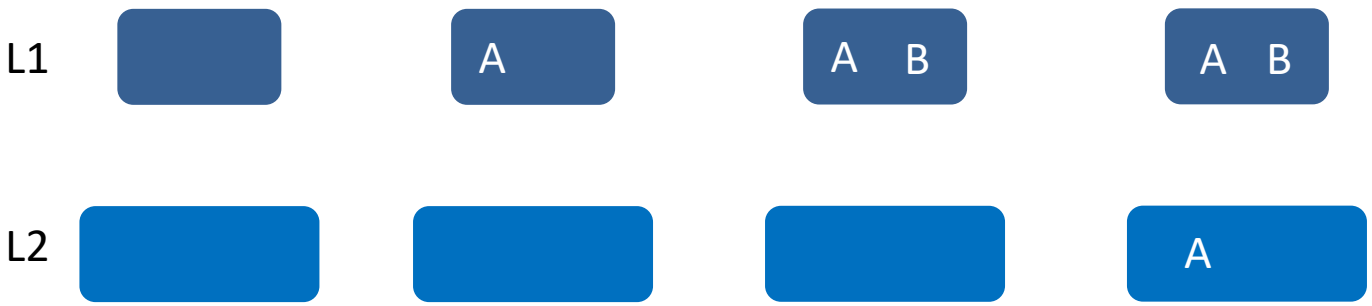
Read B miss; load B into L1 and L2

Evict A from L1 due to cache replacement

Evict B from L2 due to cache replacement

Back invalidation

Exclusive



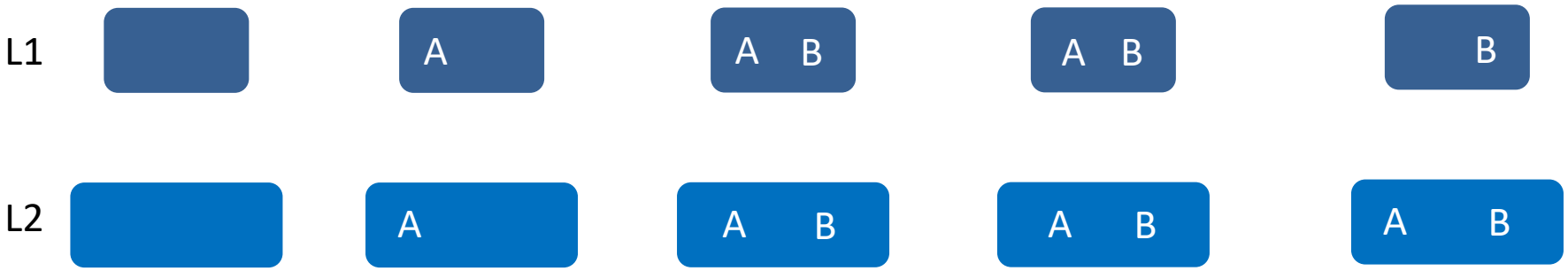
Initial state

Read A miss; load A into L1

Read B miss; load B into L1

Evict A from L1 due to cache replacement and place in L2

Non-inclusive



Initial state

Read A miss; load A into L1 and L2

Read B miss; load B into L1 and L2

Evict A from L1 due to cache replacement

Evict B from L2 due to cache replacement

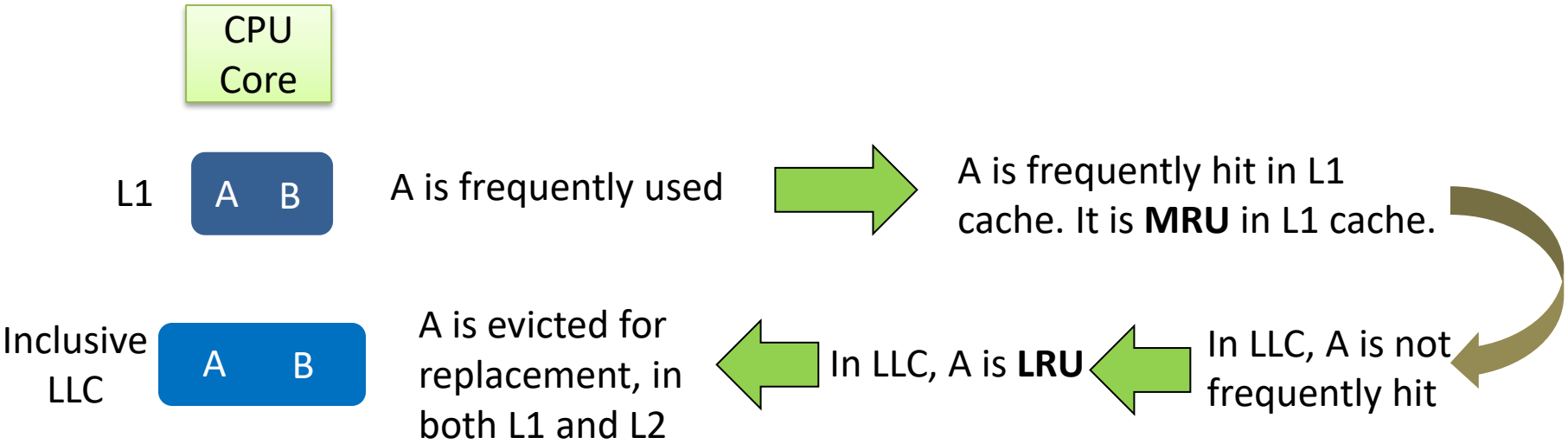
Real-world CPUs

- Intel Processors
  - Sandy bridge, inclusive
  - Haswell, inclusive
  - Skylake-S, inclusive
  - Skylake-X, *non-inclusive*
- ARM Processors
  - ARMv7, non-inclusive
  - ARMv8, non-inclusive

Inclusive, or not?

- Inclusive cache eases coherence
  - Updating a cache block in L1 entails an update in inclusive LLC.
  - A non-inclusive LLC, say L2 cache, which needs to evict a block, **must** ask L1 cache if it has the block, because such information is not present in LLC.
- Non-inclusive cache yields higher performance though, why?
  - No back invalidation
  - More data can be cached

‘*Sneaky*’ LRU for Inclusive Cache



As a result, MRU block that should be retained might be evicted, which causes performance penalty.

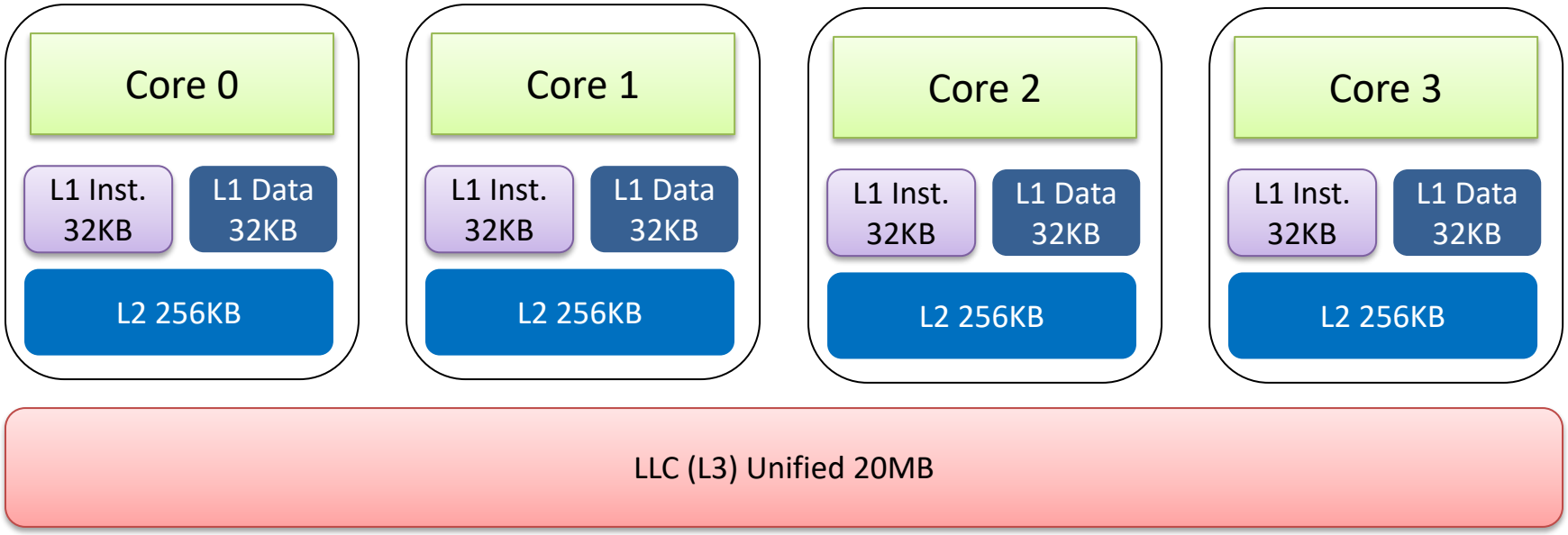
What if LLC is non-inclusive?

Should you be interested, you can click *https://doi.org/10.1109/MICRO.2010.52* to read the related research paper for details.

LLC is not monolithic

LLC is not monolithic

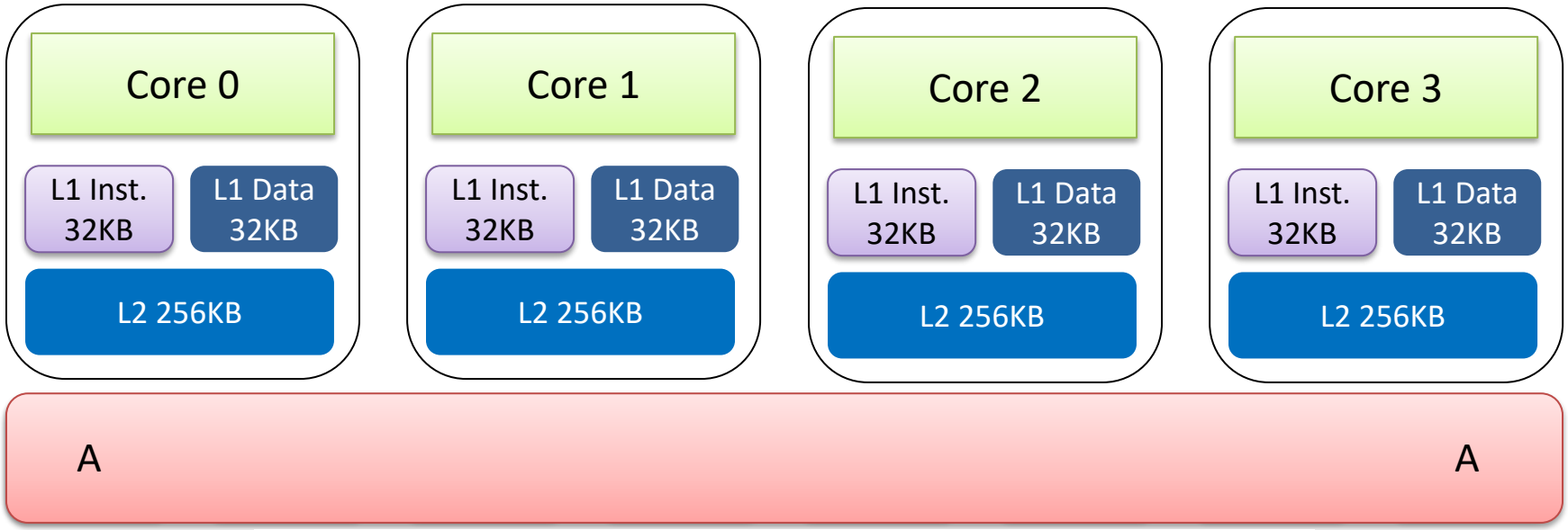
Intel® Xeon® Processor E5-2667 v3

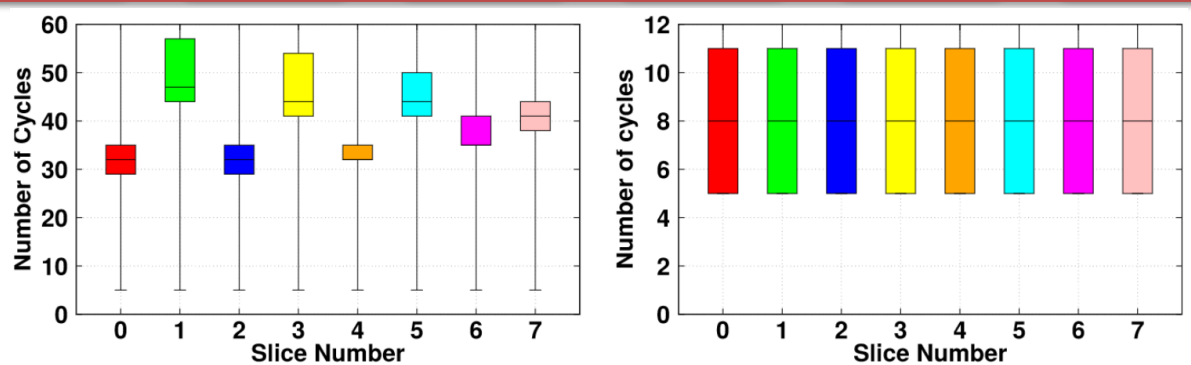


Previously, it's considered that, to CPU cores, LLC is monolithic. No matter where a cache block in the LLC, a core would load it into private L2 and L1 cache with **the same** time cost.

LLC is fine-grained

Intel® Xeon® Processor E5-2667 v3





(a) Read. **(b)** Write.

From the paper https://doi.org/10.1145/3302424.3303977

Slice-aware memory management

- The idea seems simple
  - Put your data closer to your program (core)
- But it not *EASY* to do so
  - Cache management is undocumented, not to mention even fine-grained slices
  - Researchers did a lot of efforts
    - Click https://doi.org/10.1145/3302424.3303977 for details
    - They managed to improve the average performance by 12.2% for GET operations of a key-value store.
    - 12.2% is a lot, if you consider the huge transactions every day for Taobao and JD

Use a crystal ball to help cache

Prefetch

Outline of Prefetching

- Why prefetch? Why could/does it work?
- The four questions
  - What (to prefetch), when, where, how
- Software prefetching
- Hardware prefetching

Prefetching

- Idea: Fetch the data before it is needed (i.e., ***pre***-fetch) by the program

- Why?
  - Memory latency is high. If we can prefetch accurately and early enough, we can reduce/eliminate that latency.
  - Can eliminate compulsory cache misses
  - Can it eliminate all cache misses? Capacity, conflict?

- Involves predicting which address will be needed in the future
  - Works if programs have predictable miss address patterns

Prefetching and Correctness

- Does a misprediction in prefetching affect correctness?

- No, prefetched data at a “mispredicted” address is simply not used
- There is no need for state recovery
  - In contrast to branch misprediction or value misprediction

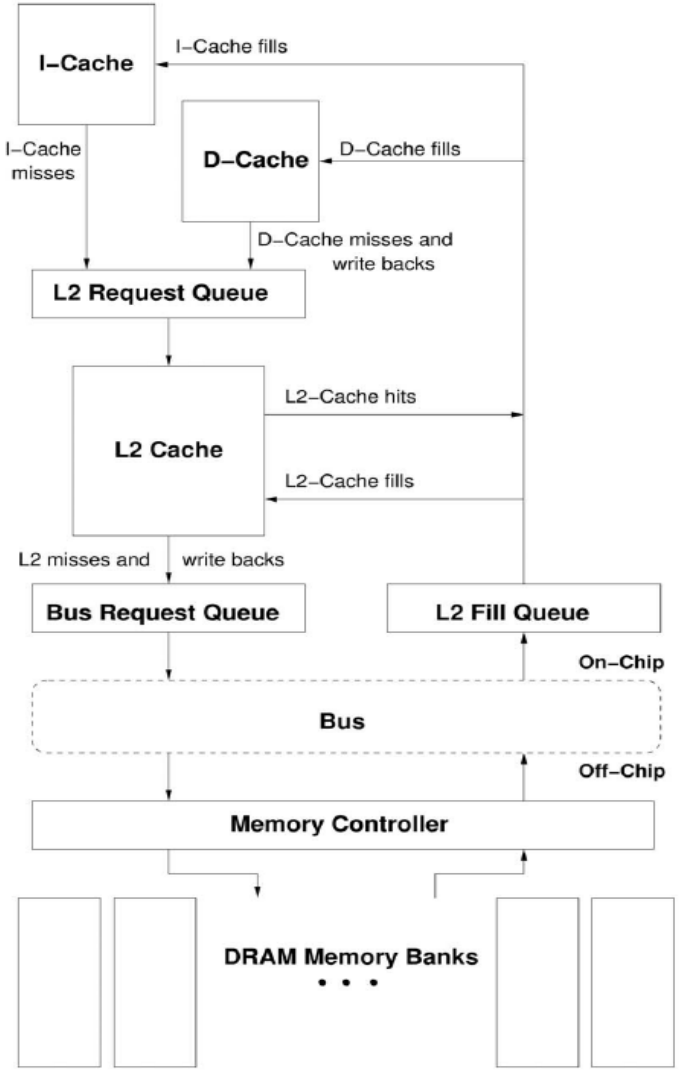
Basics

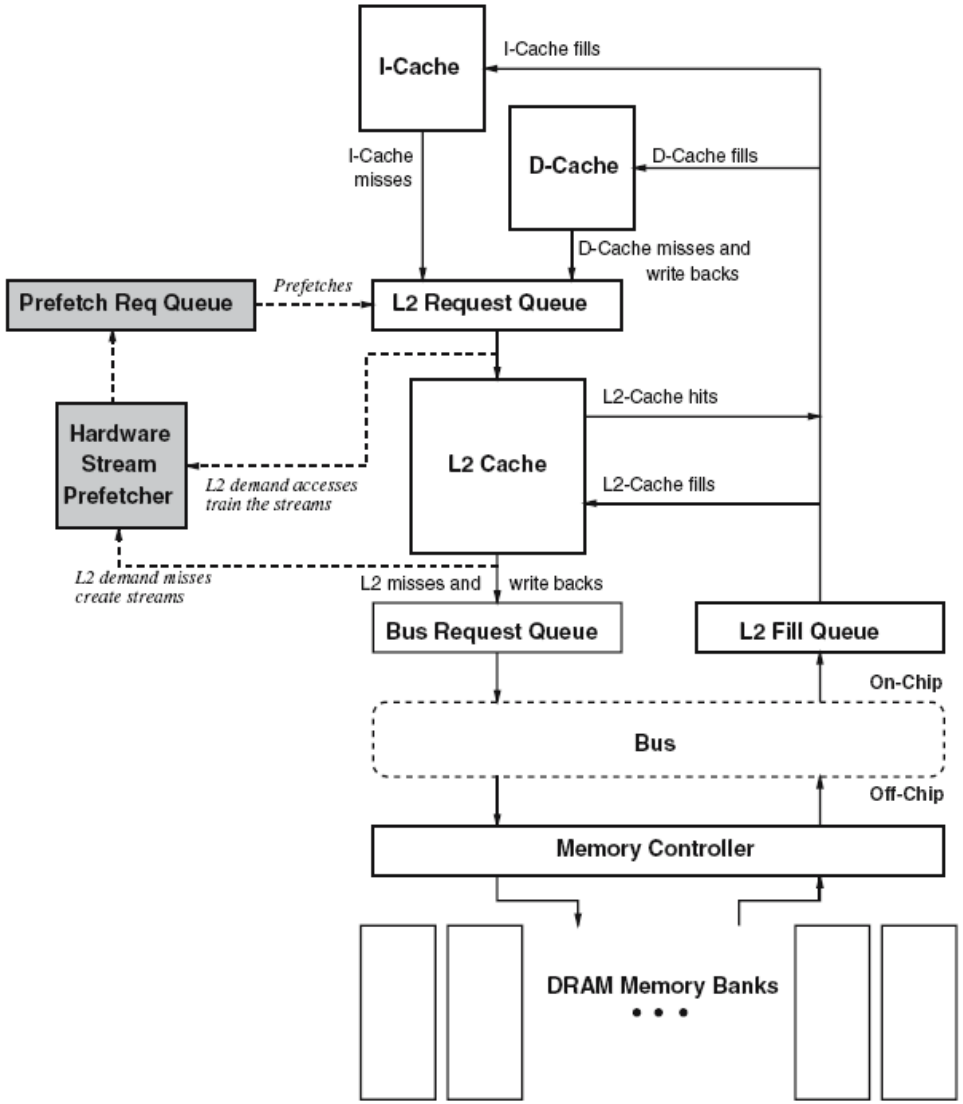
- In modern systems, prefetching is usually done in cache block granularity

- Prefetching is a technique that can reduce both
  - Miss rate
  - Miss latency

- Prefetching can be done by
  - hardware
  - compiler
  - programmer

How a HW Prefetcher Fits in the Memory System





Prefetching: The Four Questions

- What
  - What addresses to prefetch
- When
  - When to initiate a prefetch request
- Where
  - Where to place the prefetched data
- How
  - Software, hardware, execution-based, cooperative

Challenges in Prefetching: What

- What addresses to prefetch
  - Prefetching useless data wastes resources
    - Memory bandwidth
    - Cache or prefetch buffer space
    - Energy consumption
    - These could all be utilized by demand requests or more accurate prefetch requests
  - Accurate prediction of addresses to prefetch is important
    - Prefetch accuracy = used prefetches / sent prefetches
- How do we know what to prefetch
  - Predict based on past access patterns
  - Use the compiler's knowledge of data structures

- Prefetching algorithm determines what to prefetch

Challenges in Prefetching: When

- When to initiate a prefetch request
  - Prefetching too early
    - Prefetched data might not be used before it is evicted from storage
  - Prefetching too late
    - Might not hide the whole memory latency

- When a data item is prefetched affects the timeliness of the prefetcher
- Prefetcher can be made more timely by
  - Making it more aggressive: try to stay far ahead of the processor's access stream (hardware)
  - Moving the prefetch instructions earlier in the code (software)

Challenges in Prefetching: Where (I)

- Where to place the prefetched data
  - In cache
    - + Simple design, no need for separate buffers
    - -- Can evict useful demand data → cache pollution
  - In a separate prefetch buffer
    - + Demand data protected from prefetches → no cache pollution
    - -- More complex memory system design
      - - Where to place the prefetch buffer
      - - When to access the prefetch buffer (parallel vs. serial with cache)
      - - When to move the data from the prefetch buffer to cache
      - - How to size the prefetch buffer
      - - Keeping the prefetch buffer coherent
- Many modern systems place prefetched data into the cache
  - Intel Pentium 4, Core2's, AMD systems, IBM POWER4,5,6, …

Challenges in Prefetching: Where (II)

- Which level of cache to prefetch into?
  - Memory to L2, memory to L1. Advantages/disadvantages?
  - L2 to L1? (a separate prefetcher between levels)

- Where to place the prefetched data in the cache?
  - Do we treat prefetched blocks the same as demand-fetched blocks?
  - Prefetched blocks are not known to be needed
    - With LRU, a demand block is placed into the MRU position

- Do we skew the replacement policy such that it favors the demand-fetched blocks?
  - E.g., place all prefetches into the LRU position in a way?

Challenges in Prefetching: Where (III)

- Where to place the hardware prefetcher in the memory hierarchy?
  - In other words, what access patterns does the prefetcher see?
  - L1 hits and misses
  - L1 misses only
  - L2 misses only

- Seeing a more complete access pattern:

  + Potentially better accuracy and coverage in prefetching

  -- Prefetcher needs to examine more requests (bandwidth intensive, more ports into the prefetcher?)

Challenges in Prefetching: How

- Software prefetching
  - ISA provides prefetch instructions
  - Programmer or compiler inserts prefetch instructions (effort)
  - Usually works well only for “regular access patterns”

- Hardware prefetching
  - Hardware monitors processor accesses
  - Memorizes or finds patterns/strides
  - Generates prefetch addresses automatically

- Execution-based prefetchers
  - A “thread” is executed to prefetch data for the main program
  - Can be generated by either software/programmer or hardware

Software Prefetching (I)

- Idea: Compiler/programmer places prefetch instructions into appropriate places in code

- Mowry et al., "Design and Evaluation of a Compiler Algorithm for Prefetching," ASPLOS 1992.

- Prefetch instructions prefetch data into caches
- Compiler or programmer can insert such instructions into the program

x86 PREFETCH Instruction

PREFETCH*h*—Prefetch Data Into Caches

Opcode	Instruction	64-Bit Mode	Compat/ Leg Mode	Description
0F 18 /1	PREFETCHT0 *m8*	Valid	Valid	Move data from *m8* closer to the processor using T0 hint.
0F 18 /2	PREFETCHT1 *m8*	Valid	Valid	Move data from *m8* closer to the processor using T1 hint.
0F 18 /3	PREFETCHT2 *m8*	Valid	Valid	Move data from *m8* closer to the processor using T2 hint.
0F 18 /0	PREFETCHNTA *m8*	Valid	Valid	Move data from *m8* closer to the processor using NTA hint.

Description

Fetches the line of data from memory that contains the byte specified with the source operand to a location in the cache hierarchy specified by a locality hint:

- T0 (temporal data)—prefetch data into all levels of the cache hierarchy.
  - Pentium III processor—1st- or 2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T1 (temporal data with respect to first level cache)—prefetch data into level 2 cache and higher.
  - Pentium III processor—2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- T2 (temporal data with respect to second level cache)—prefetch data into level 2 cache and higher.
  - Pentium III processor—2nd-level cache.
  - Pentium 4 and Intel Xeon processors—2nd-level cache.
- NTA (non-temporal data with respect to all cache levels)—prefetch data into non-temporal cache structure and into a location close to the processor, minimizing cache pollution.
  - Pentium III processor—1st-level cache
  - Pentium 4 and Intel Xeon processors—2nd-level cache

microarchitecture dependent specification

different instructions for different cache levels

Software Prefetching (II)

```
for (i=0; i<N; i++) {
    __prefetch(a[i+8]);
    __prefetch(b[i+8]);
    sum += a[i]*b[i];
}
```

```
while (p) {
    __prefetch(p→next);
    work(p→data);
    p = p→next;
}
```

```
while (p) {
    __prefetch(p→next→next→next);
    work(p→data);
    p = p→next;
}
```

Which one is better?

- Can work for very regular array-based access patterns. Issues:
  - -- Prefetch instructions take up processing/execution bandwidth
  - – How early to prefetch? Determining this is difficult
    - -- Prefetch distance depends on hardware implementation (memory latency, cache size, time between loop iterations) → portability?
    - -- Going too far back in code reduces accuracy (branches in between)
  - – Need "special" prefetch instructions in ISA?
    - Alpha load into register 31 treated as prefetch (r31==0)
    - PowerPC *dcbt* (data cache block touch) instruction
  - -- Not easy to do for pointer-based data structures

Software Prefetching (III)

- Where should a compiler insert prefetches?
  - Prefetch for every load access?
    - Too bandwidth intensive (both memory and execution bandwidth)
  - Profile the code and determine loads that are likely to miss
    - What if profile input set is not representative?
  - How far ahead before the miss should the prefetch be inserted?
    - Profile and determine probability of use for various prefetch distances from the miss
      - What if profile input set is not representative?
      - Usually need to insert a prefetch far in advance to cover 100s of cycles of main memory latency → reduced accuracy

Hardware Prefetching (I)

- Idea: Specialized hardware observes load/store access patterns and prefetches data based on past access behavior

- Tradeoffs:
  + Can be tuned to system implementation
  + Does not waste instruction execution bandwidth
  -- More hardware complexity to detect patterns
    - Software can be more efficient in some cases

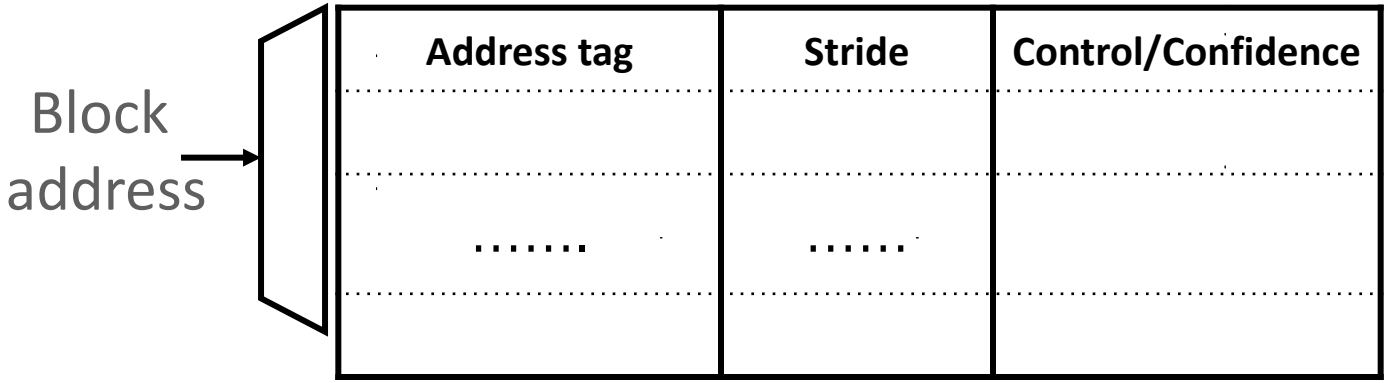
Next-Line Prefetchers

- Simplest form of hardware prefetching: always prefetch next N cache lines after a demand access (or a demand miss)
  - Next-line prefetcher (or next sequential prefetcher)
  - Tradeoffs:
    - + Simple to implement. No need for sophisticated pattern detection
    - + Works well for sequential/streaming access patterns (instructions?)
    - -- Can waste bandwidth with irregular patterns
    - -- And, even regular patterns:
      - - What if the program is traversing memory from higher to lower addresses?
      - - Also prefetch “previous” N cache lines?

Stride Prefetchers

- Two kinds
  - Instruction program counter (PC) based
  - Cache block address based
- Instruction based:
  - Baer and Chen, "An effective on-chip preloading scheme to reduce data access penalty," SC 1991.
  - Idea:
    - Record the distance between the memory addresses referenced by a load instruction (i.e. stride of the load) as well as the last address referenced by the load
    - Next time the same load instruction is fetched, prefetch last address + stride

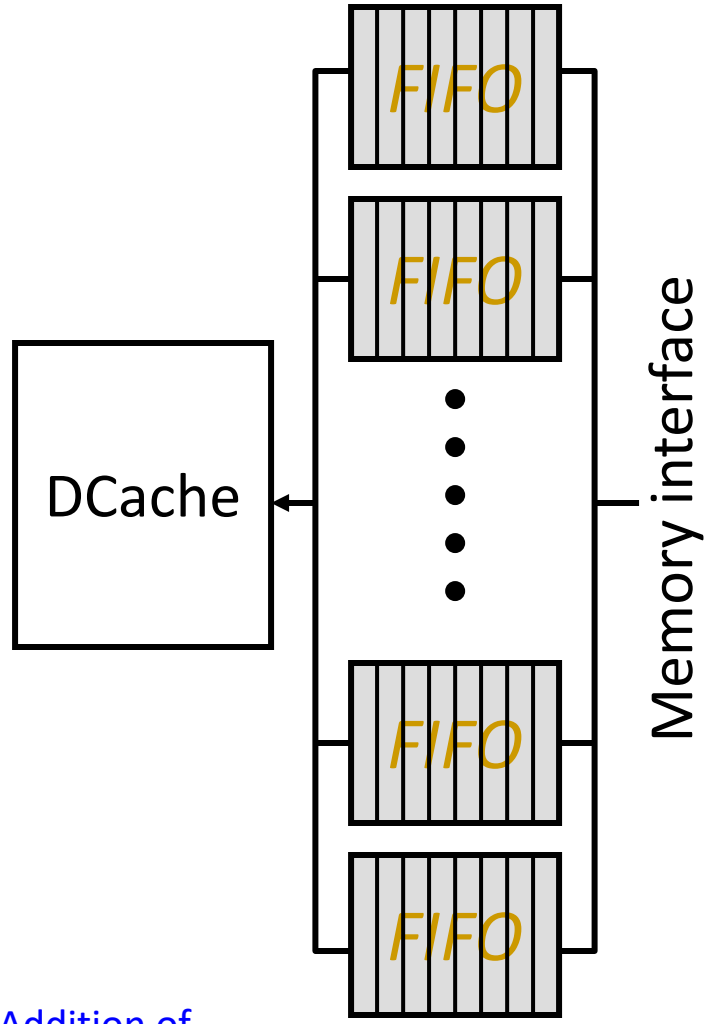
Cache-Block Address Based Stride Prefetching



- Can detect
  - A, A+N, A+2N, A+3N, …
  - Stream buffers are a special case of cache block address based stride prefetching where N = 1

Stream Buffers (Jouppi, ISCA 1990)

- Each stream buffer holds one stream of sequentially prefetched cache lines
- On a load miss check the head of all stream buffers for an address match
  - if hit, pop the entry from FIFO, update the cache with data
  - if not, allocate a new stream buffer to the new miss address (may have to recycle a stream buffer following LRU policy)
- Stream buffer FIFOs are continuously topped-off with subsequent cache lines whenever there is room and the bus is not busy



Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," ISCA 1990.

No, do not cache us

Why do we use cache?

- The data to be used will be reused soon
- But there is some data for which caching is not that useful
  - So-called *streaming* data
  - e.g., larger matrices filled but used much later
- Caching such data pollutes the cache

Non-temporal load & store

- Intel
  - MOVNTDQA: Load Double Quadword Non-Temporal Aligned Hint
  - MOVNTDQ: Store Double Quadword Using Non-Temporal Hint
- ARMv8
  - LDNP
  - STNP

Note that, such instructions only give a **hint** to the memory system that caching is not useful for this data

When you can use them?

- The data is unlikely to be used soon
- The data is large
- Examples
  - Logs (journals)
  - In file systems and databases, logs are used for recovery/retrieval

Yes, you can control the cache

Scratchpad Memory

Scratchpad Memory

- Strictly speaking, scratchpad memory (SPM) is **not** cache
  - Widely used in embedded systems
  - On-chip SRAM, like cache, close to ALU
  - *Software controlled*: software decides what data sections to be placed in SPM
    - By the programmer or the compiler before running
  - Memory-mapped to a predefined address range

Why SPM?

- To control the execution time
  - More predictable than hardware-controlled cache
  - Especially for WCET (worst-case execution time)
- With reduced area and energy consumptions
  - More space- and energy-efficient

Conclusion

- There are many interesting facts of CPU cache
- To make the best of cache can boost your program's performance!

Quiz

Quiz for prefetch accuracy

Piazza: "Video Lecture 23 Advanced Cache"

- What is the hardware prefetch accuracy if access stride = 1 and N = 2?

A. 0%
B. 25%
C. 50%
D. 75%
E. 100%