



LUND
UNIVERSITY

EITF20: Computer Architecture

Part4.1.1: Cache - 1

Liang Liu
liang.liu@eit.lth.se



Outline

- Reiteration
- Memory hierarchy
- Cache memory
- Cache performance
- Main memory
- Summary

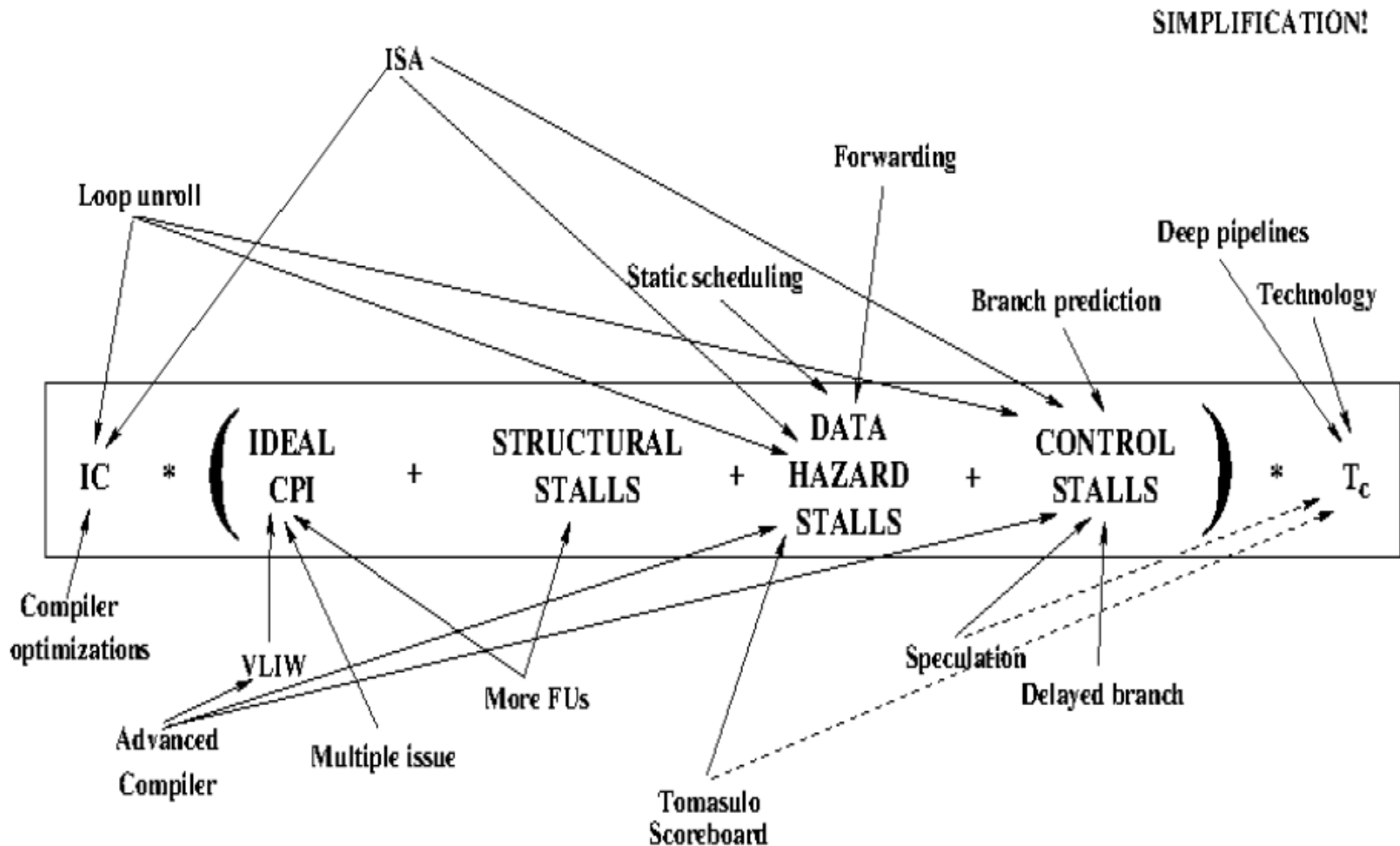


Dynamic scheduling, speculation summary

- ❑ **Tolerates unpredictable delays**
- ❑ **Compile for one pipeline - run effectively on another**
- ❑ **Allows speculation**
 - multiple branches
 - in-order commit
 - precise exceptions
 - time, energy; recovery
- ❑ **Significant increase in HW complexity**
- ❑ **Out-of-order execution, completion**
- ❑ **Register renaming**
- ❑ **Tomasulo, CDB, ROB**



CPU performance equation



Summary pipeline - implementation

Problem	Simple	Scoreboard	Tomasulo	Tomasulo + Speculation
	Static Sch	Dynamic Scheduling		
RAW	forwarding stall	wait (Read)	CDB stall	CDB stall
WAR	-	wait (Write)	Reg. rename	Reg. rename
WAW	-	wait (Issue)	Reg. rename	Reg. rename
Exceptions	precise	?	?	precise, ROB
Issue	in-order	in-order	in-order	in-order
Execution	in-order	out-of-order	out-of-order	out-of-order
Completion	in-order	out-of-order	out-of-order	in-order
Structural hazard	-	many FU stall	many FU, CDB, stall	many FU, CDB, stall
Control hazard	Delayed br., stall	Branch prediction	Branch prediction	Br. pred, speculation

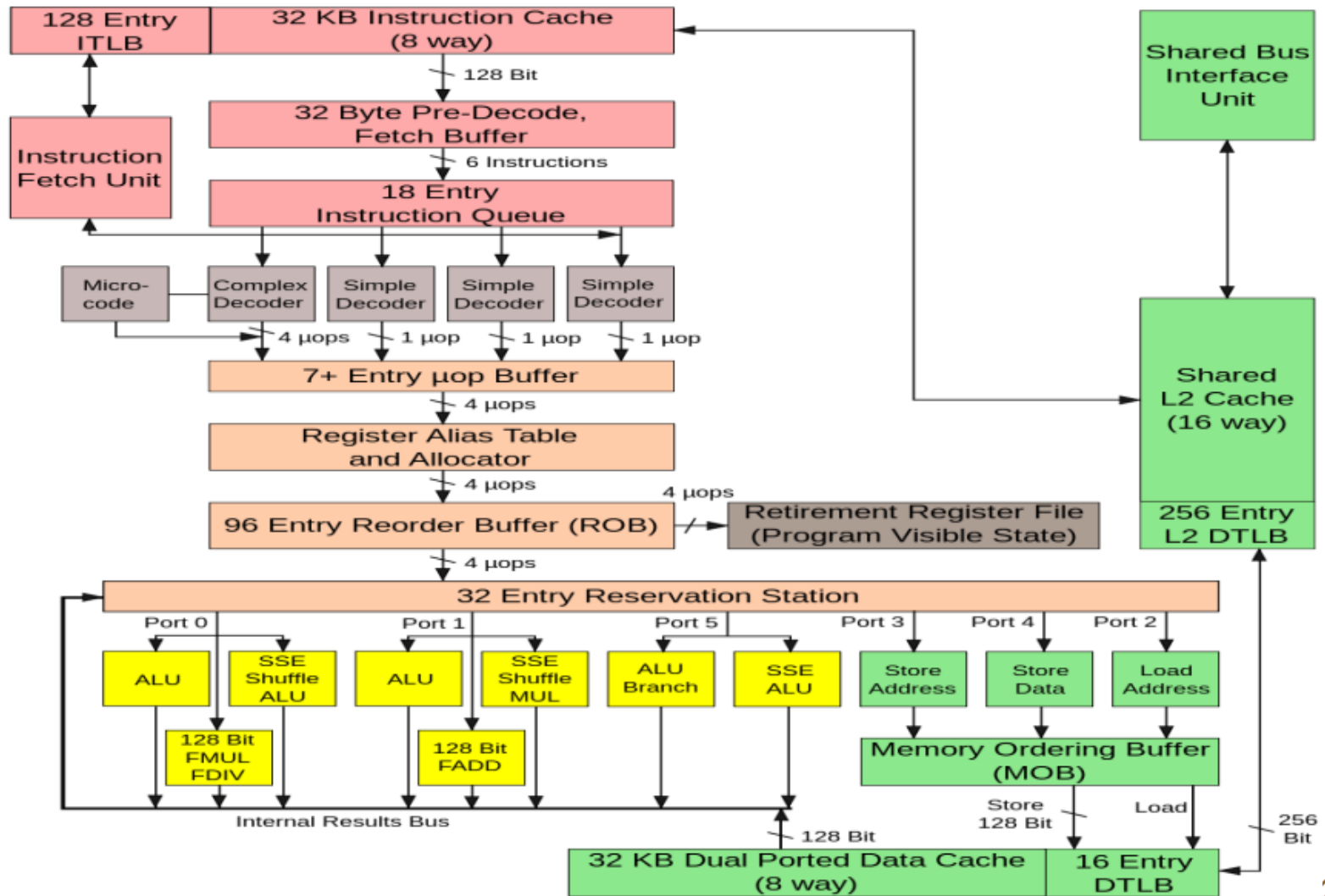


Approches for multiple issuing

	Issue	Hazard detection	Scheduling	Characteristics /examples
Superscalar	dynamic	HW	static	in-order execution ARM
Superscalar	dynamic	HW	dynamic	out-of-order execution
Superscalar	dynamic	HW	dynamic	speculation Pentium 4 IBM power5
VLIW	static	compiler	static	TI C6x
EPIC	static	compiler	mostly static	Itanium



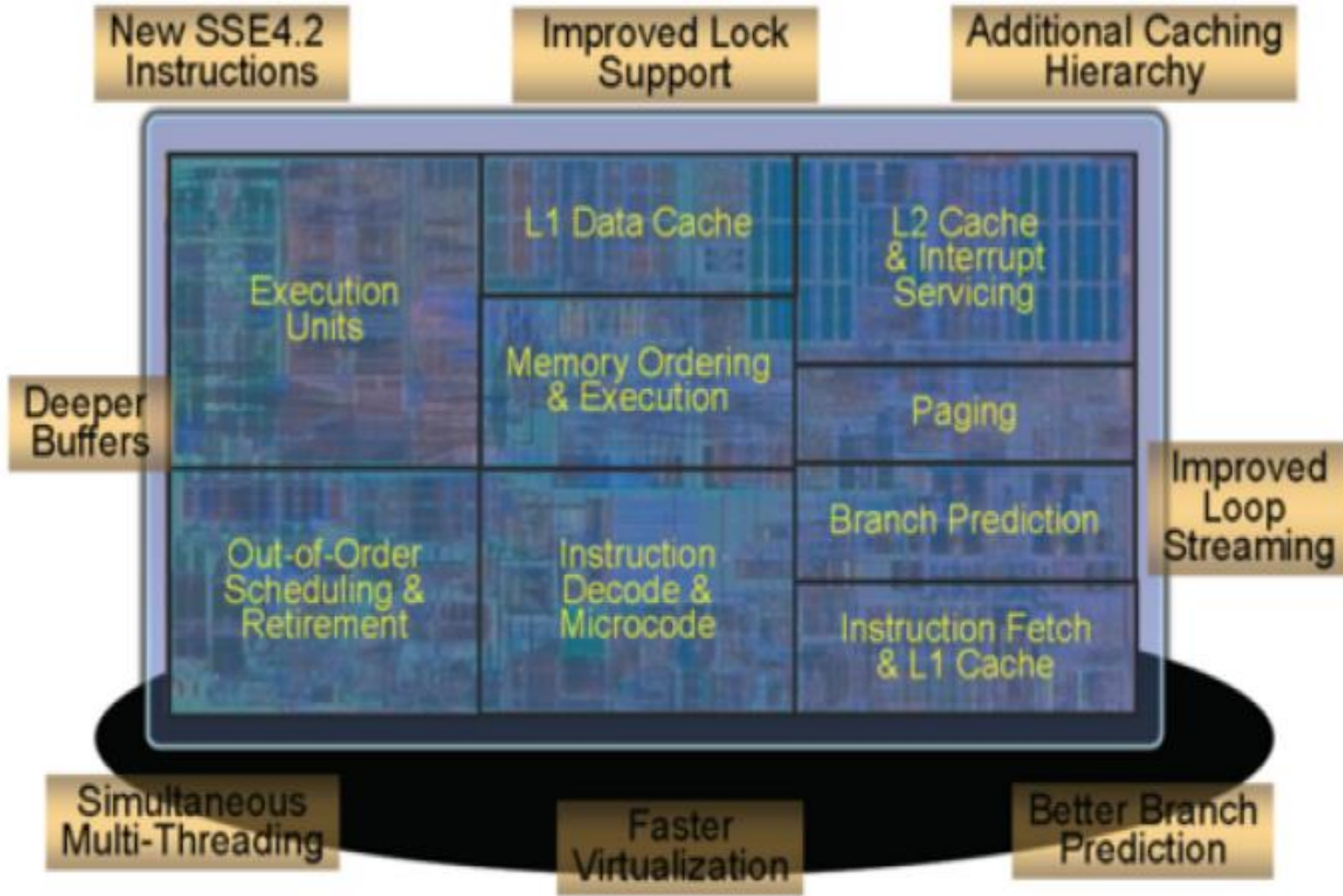
Intel core-2 chip



Intel Core 2 Architecture



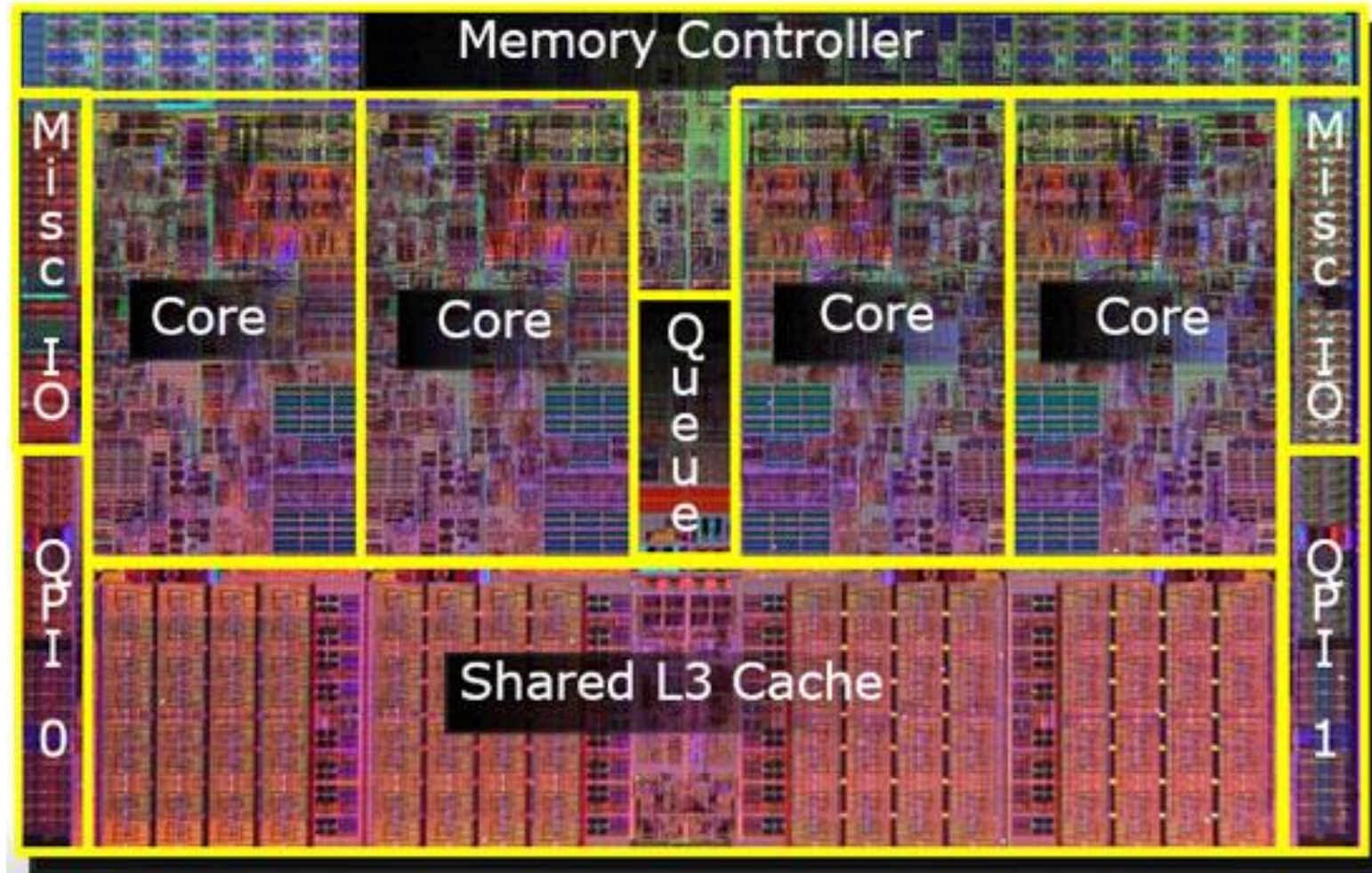
Intel core-2 chip



Questions?
Comments?



Intel core i7 chip



Outline

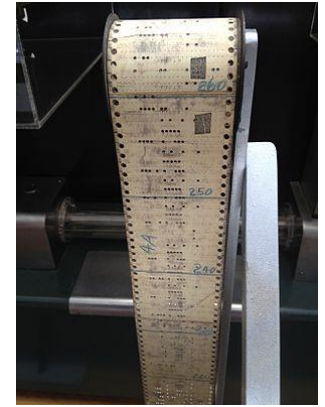
- Reiteration
- **Memory hierarchy**
- Cache memory
- Cache performance
- Main memory
- Summary



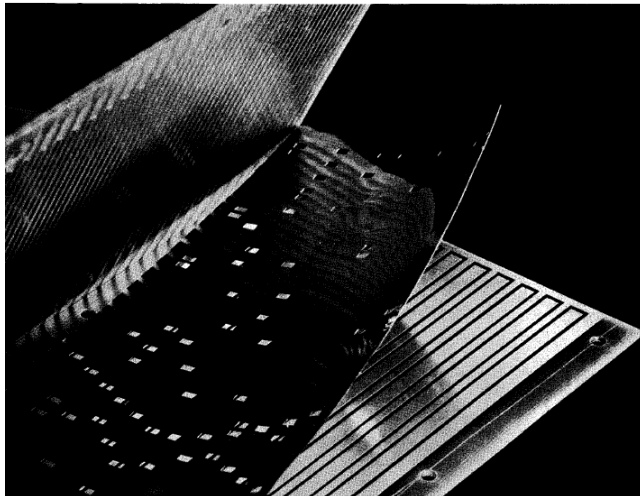
Memory in early days



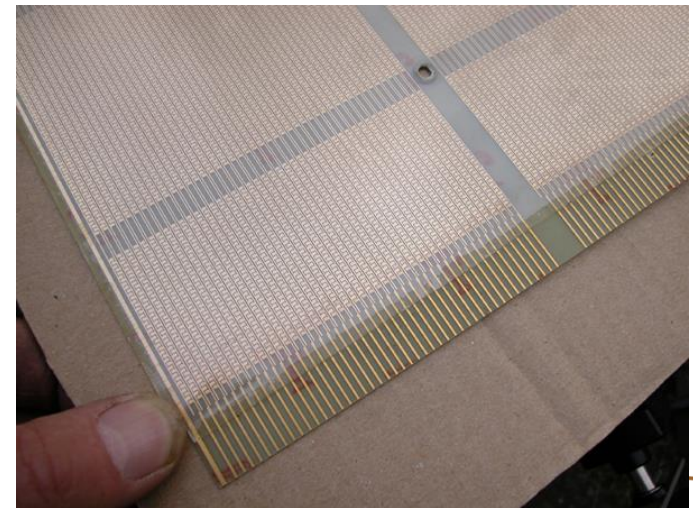
Punched cards, From early 1700s through Jacquard Loom, Babbage, and then IBM



Punched paper tape, instruction stream in Harvard Mk 1 (1950s)



IBM Card Capacitor ROS (360)



IBM Balanced Capacitor ROS (1968)



Memory in early days





Browse Journals & Magazines > IBM Journal of Research and D ...> Volume:10 Issue:2 ?



Design of a Printed Card Capacitor Read-Only Store

 Full Text
Sign-In or Purchase

1 Author(s) [Haskell, J.W.](#)

Abstract Authors References Cited By Keywords Metrics

 Download Citations
 Email
 Print
 Export

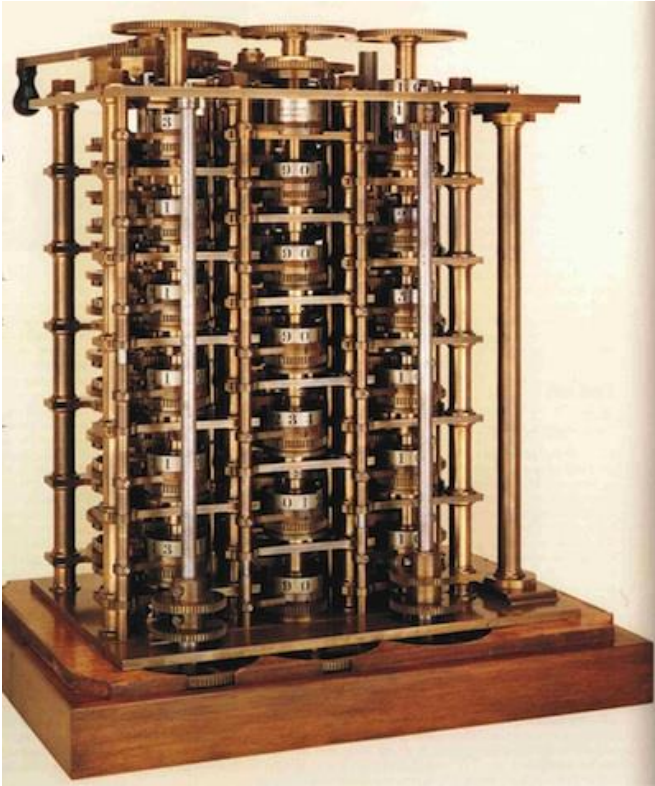



The Printed Card Capacitor Read-Only Store system is introduced as one of three Read-Only Store (ROS) technologies presently employed in the IBM System/360 computers. A detailed description is given of the elementary storage structure—a capacitor matrix with a novel arrangement for rapidly and economically changing the matrix configuration. The changeable element is an etched or printed Mylar card, prepared with standard IBM card-punch equipment. The assembly of storage structures into a compact 4032-word, 60-bit module is described. Considerations involved in the selection of matrix parameters are reviewed and a theoretical analysis of matrix performance is presented. It is shown that some of the limitations associated with a linear matrix are minimized by an incomplete integration of the matrix output. The impact on over-all ROS performance of factors external to the capacitor matrix is considered. These factors include the effects of a noisy drive system and the asymmetrical character of the matrix relative to the access circuitry. Computed and observed results are compared for typical System/360, Model 30 ROS operation. The actual matrix response amplitude is observed to be greater than the computed value due to the level of system background noise; however, computed and observed waveforms are otherwise in good agreement.

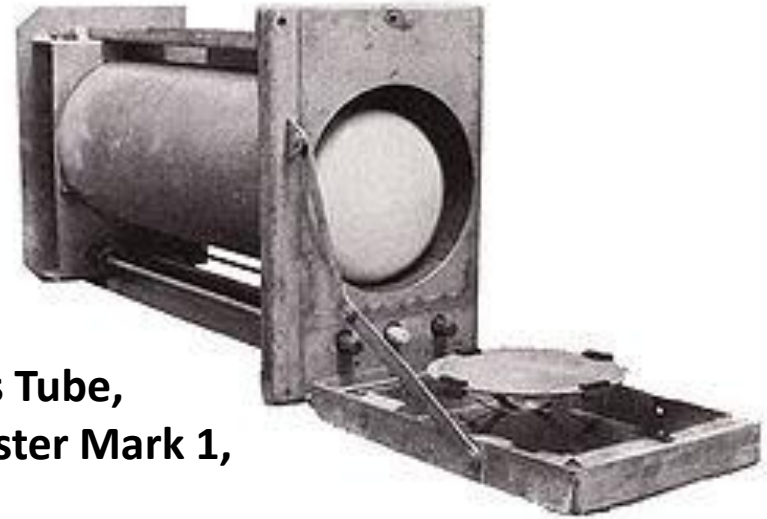


Memory in early days

Babbage, 1800s: Digits stored on mechanical



Also, regenerative capacitor memory on Atanasoff-Berry computer, and rotating magnetic drum memory on IBM 650

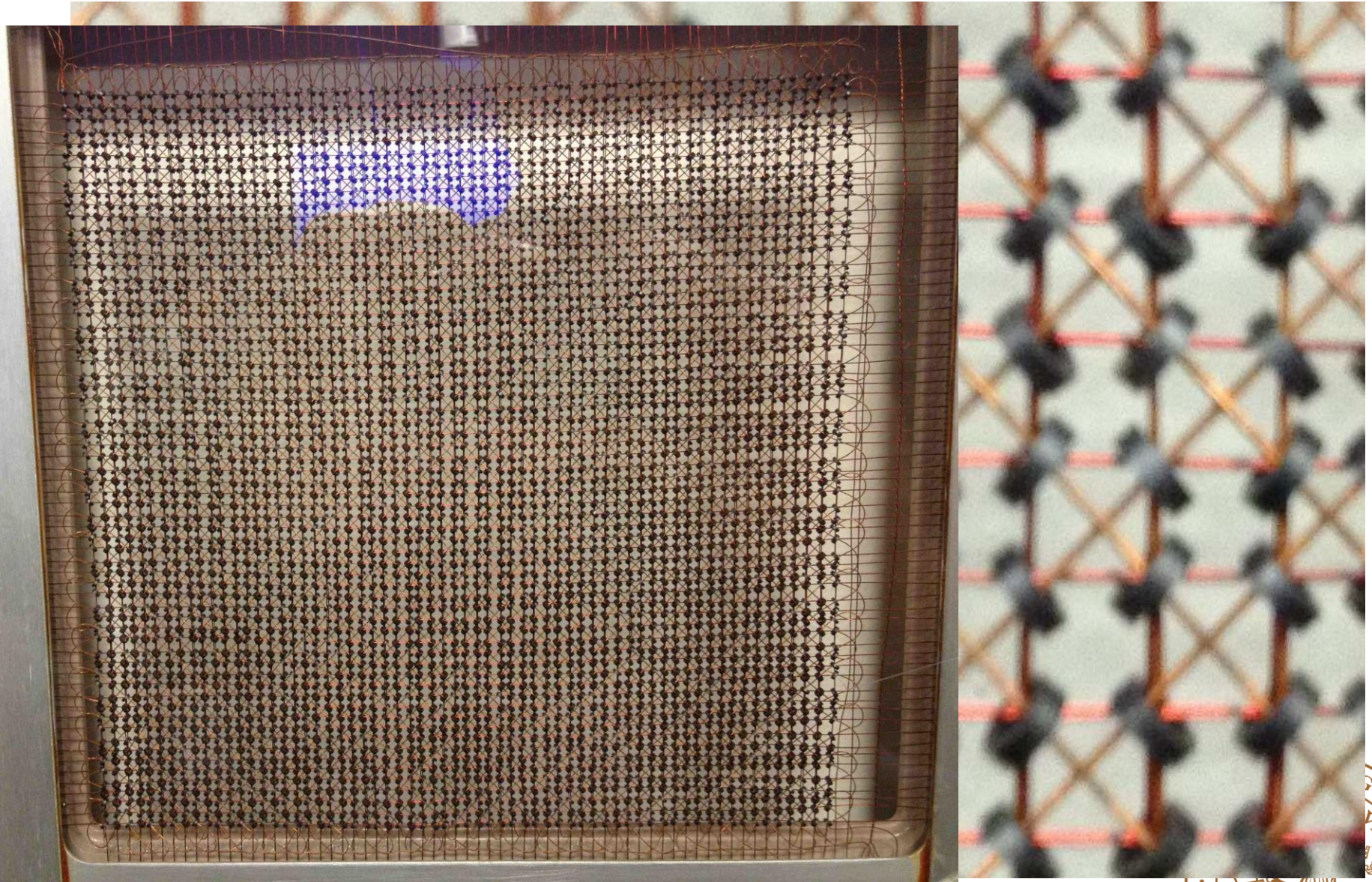


**Williams Tube,
Manchester Mark 1,
1947**

Mercury Delay Line, Univac 1, 1951

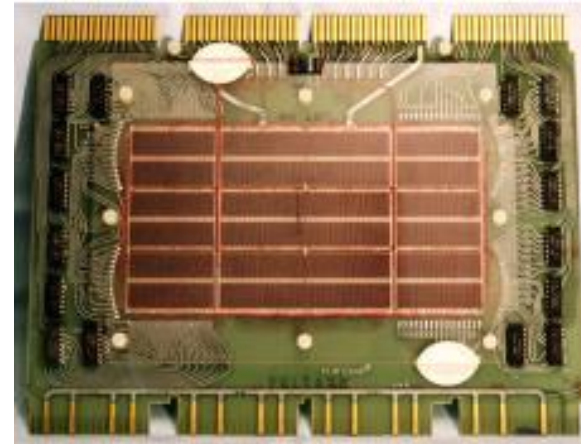


MIT Whirlwind Core Memory

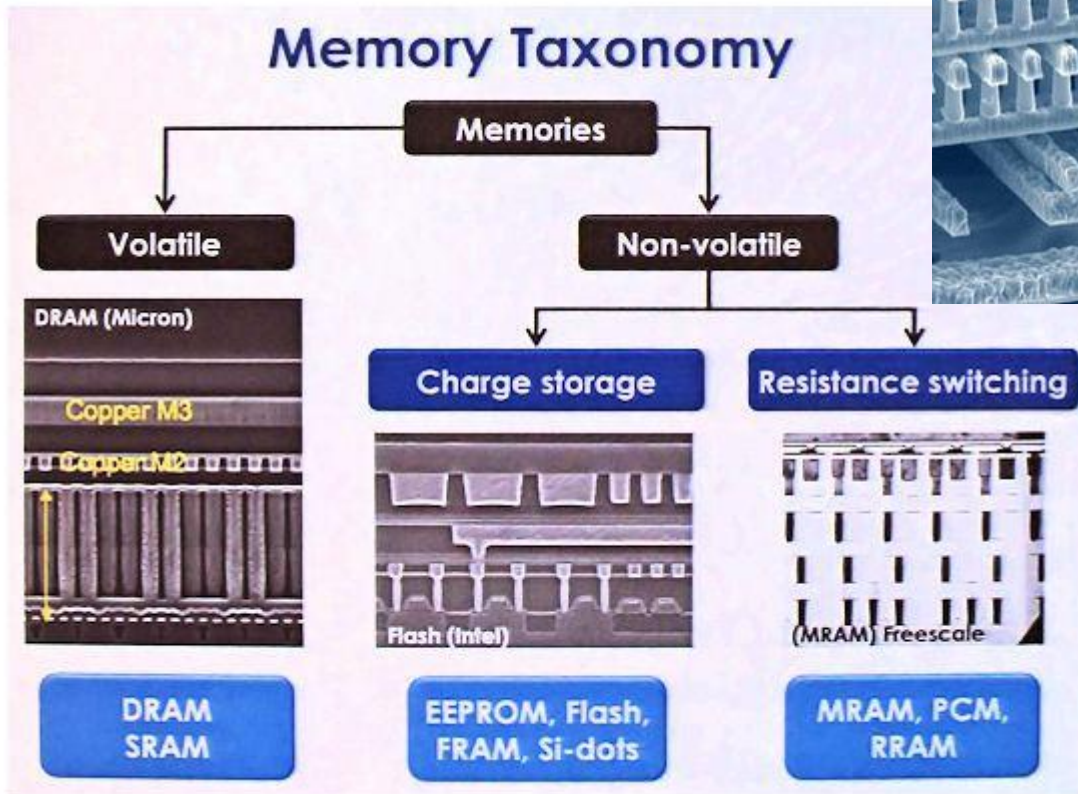
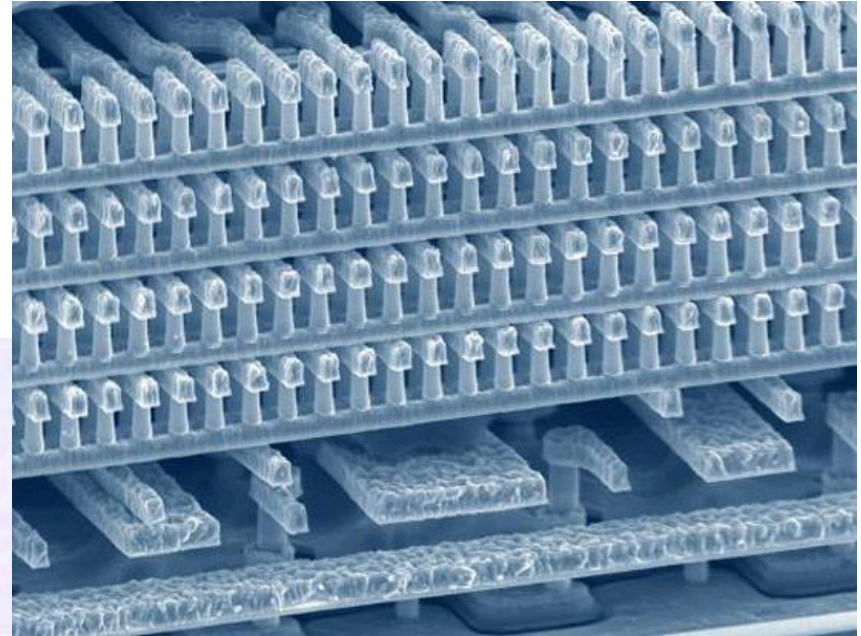


Core Memory

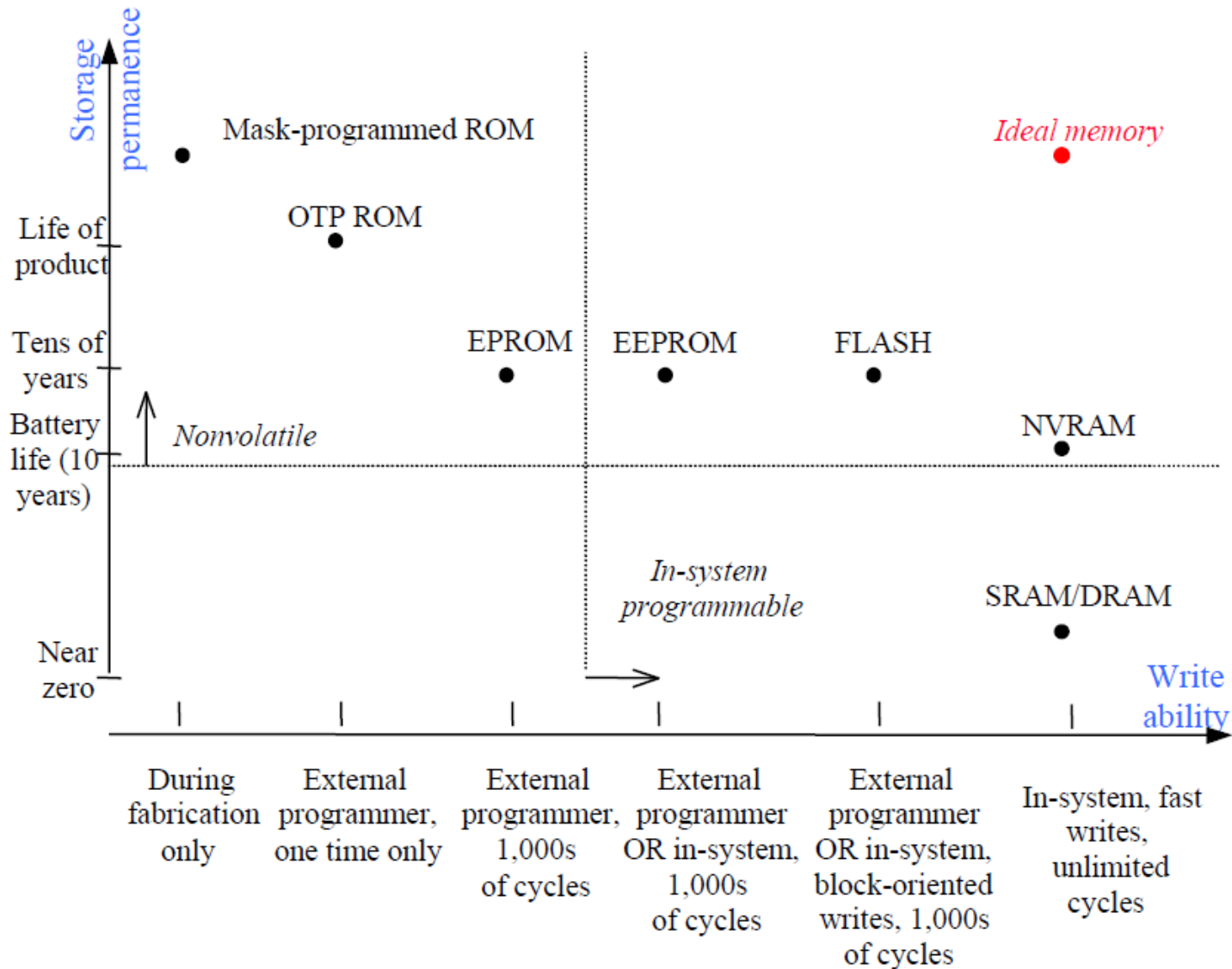
- ❑ Core memory was first large scale reliable main memory
- ❑ invented by Forrester in late 40s/early 50s at MIT for Whirlwind project
- ❑ Bits stored as magnetization polarity on small ferrite cores threaded onto two-dimensional grid of wires
- ❑ Robust, non-volatile storage
- ❑ Used on space shuttle computers
- ❑ Core access time ~ 1ms



Semiconductor memory



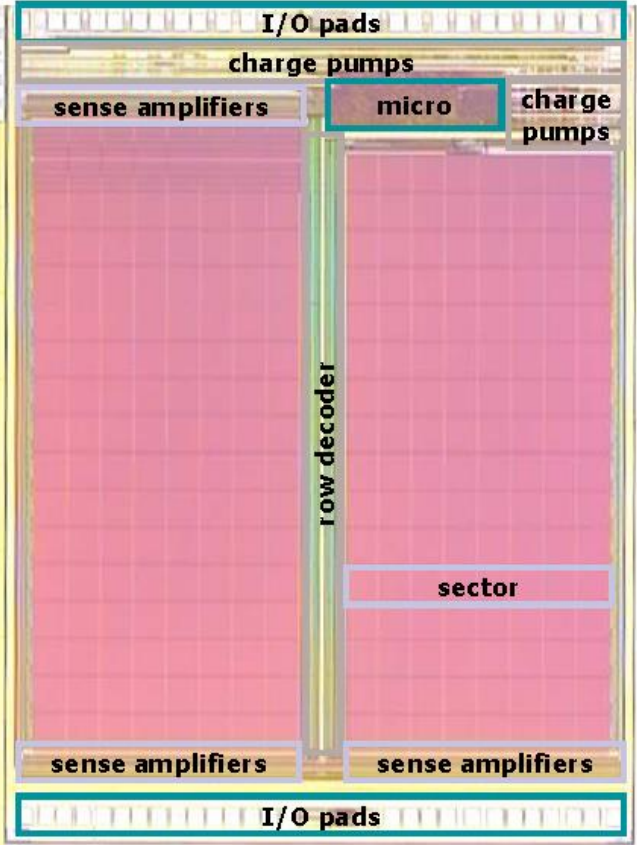
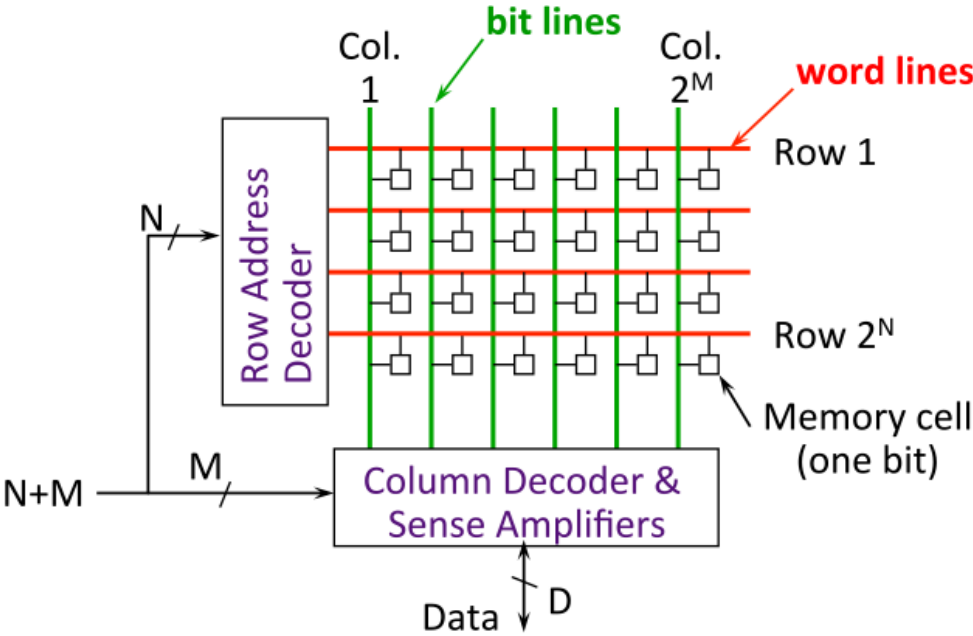
Memory Classification



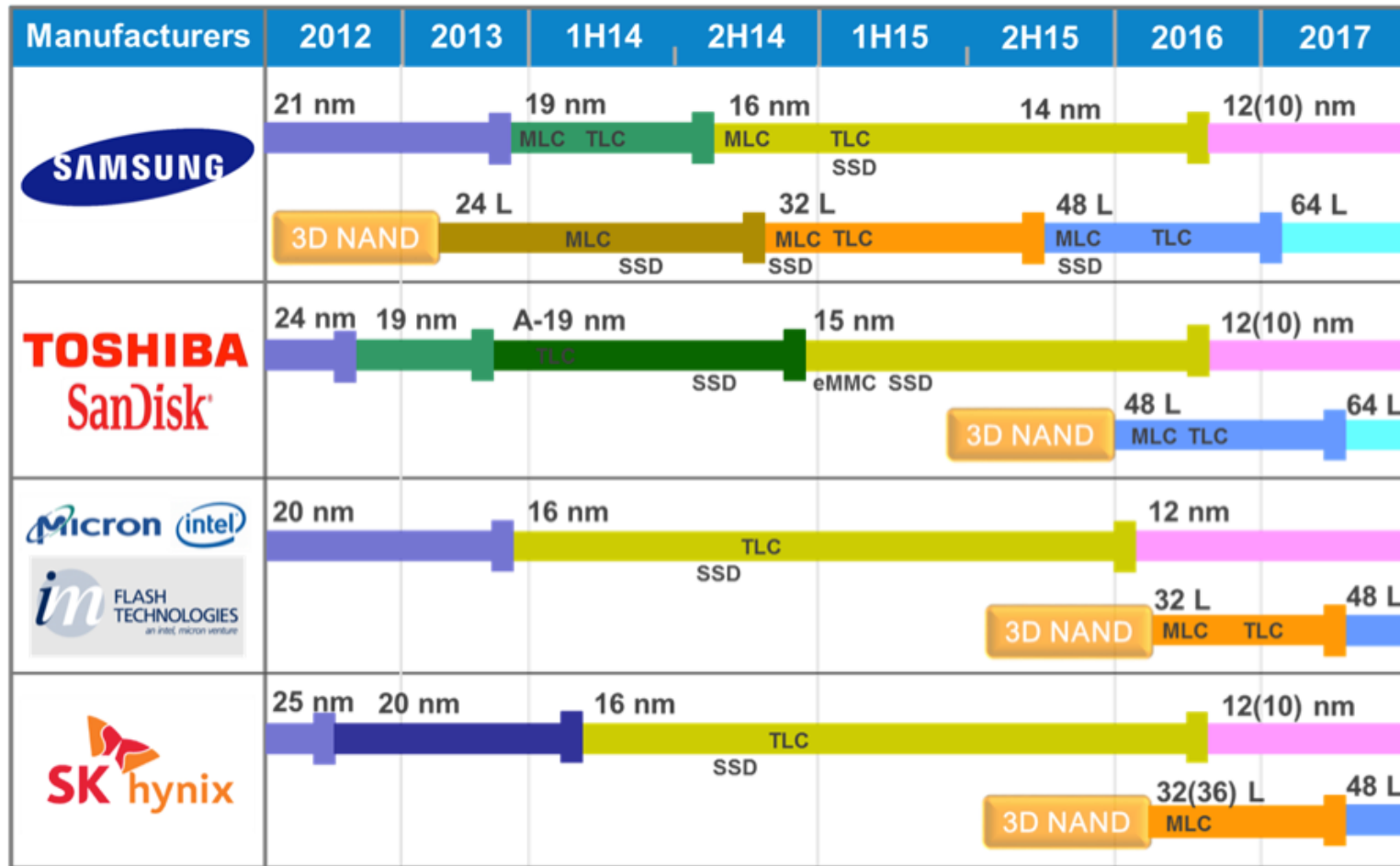
Picture from *Embedded Systems Design: A Unified Hardware/Software Introduction*



Memory Architecture



Semiconductor memory

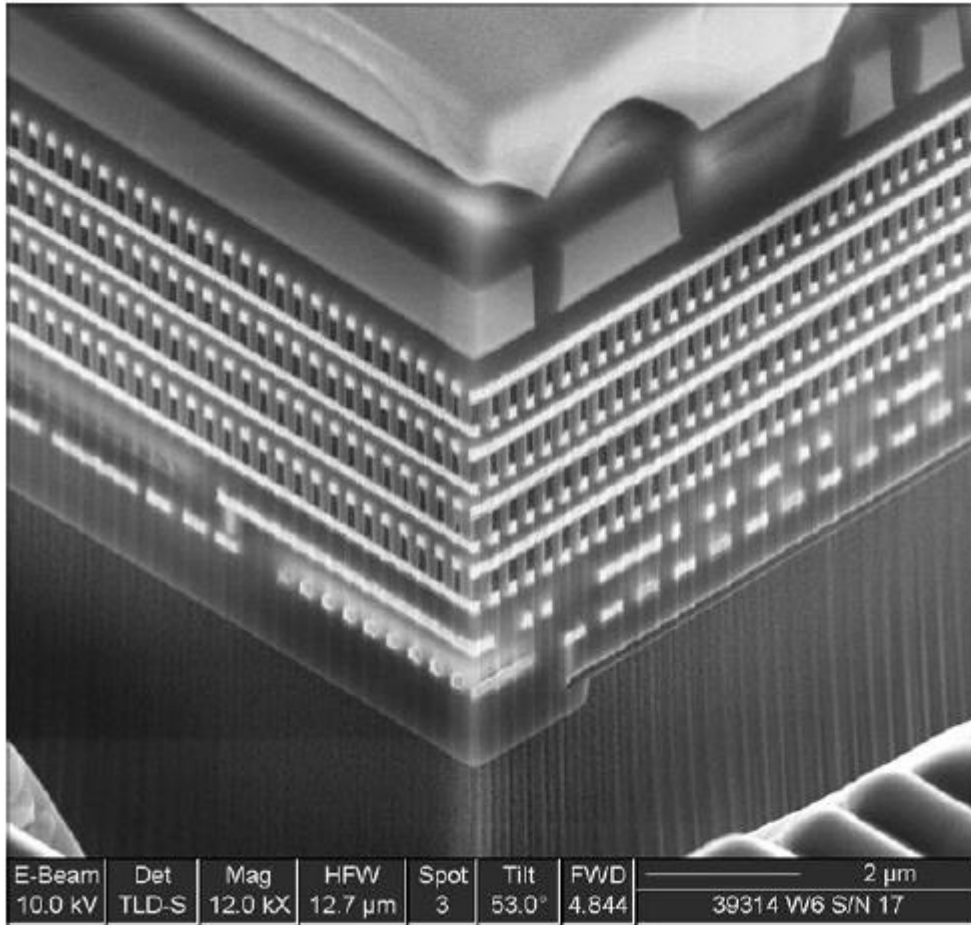


First 32nm NAND Flash memory, **2009**, Toshiba

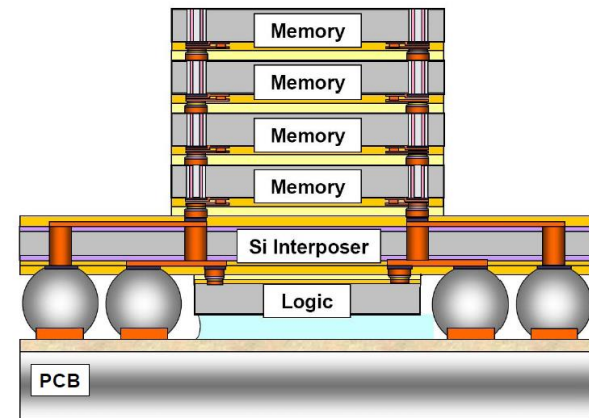
First 32nm CPU released, **2010**, Intel Core i3



Semiconductor memory



- Al top metal
- 4 layers of memory cells + tungsten interconnect
- 2 levels of tungsten routing
- LV + HV CMOS logic



Example of 3-D integrated construction (Image courtesy of DuPont Electronics)

First 22-nm SRAMs using Tri-Gate transistors, in Sept. **2009**
First 22-nm Tri-Gate microprocessor (Ivy Bridge), released in **2013**



Memory design

You want a memory to be:

FAST

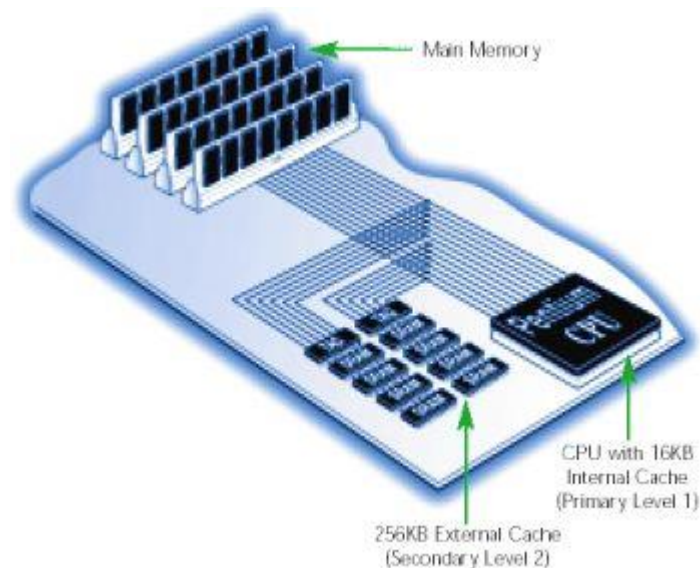
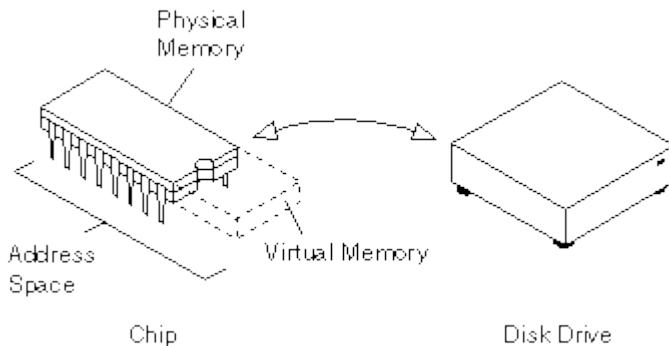
BIG



Memory, big fast, cheap

Use two “magic” tricks (from architecture)

- ❑ Make a small memory seem bigger (Without making it much slower) => **virtual memory**
- ❑ Make a slow memory seem faster (Without making it smaller) => **cache memory**



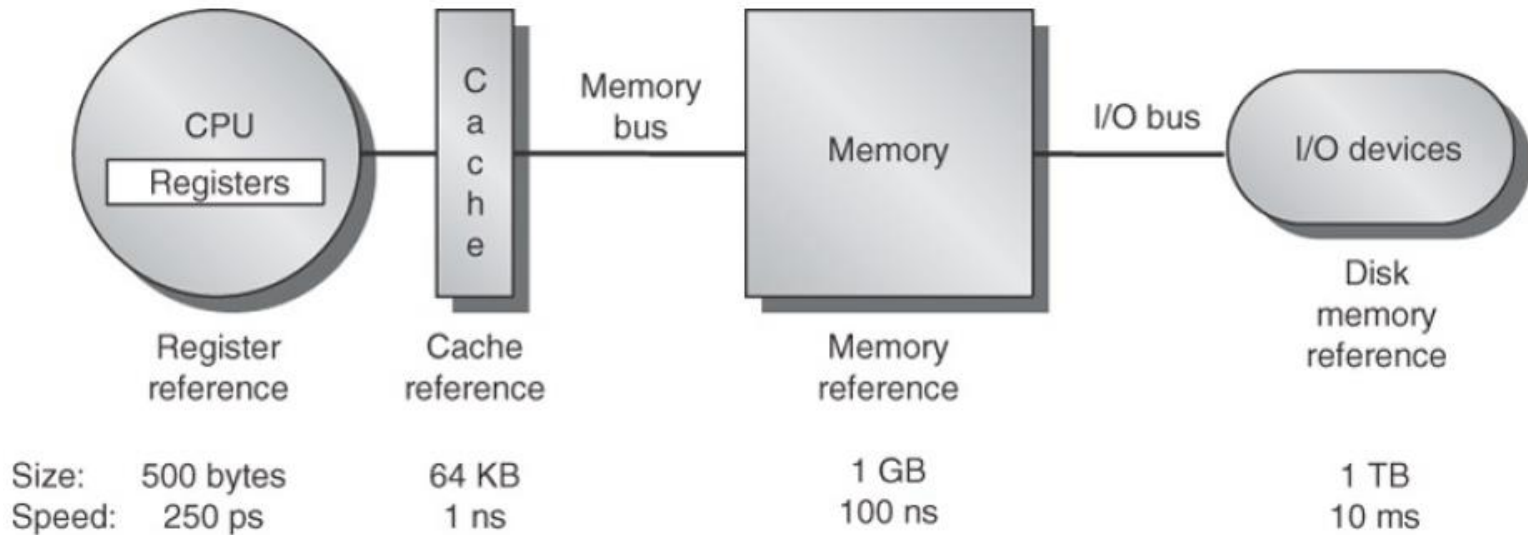
Memory tricks (techniques)

Use a hierarchy

CPU	superfast	Registers	instructions	
Memory	FAST	Cache	cache memory	(HW)
	CHEAP	Main memory	virtual memory	(SW)
	BIG	Disk		



Initial model of memory hierarchy



© 2007 Elsevier, Inc. All rights reserved.



Levels of memory hierarchy

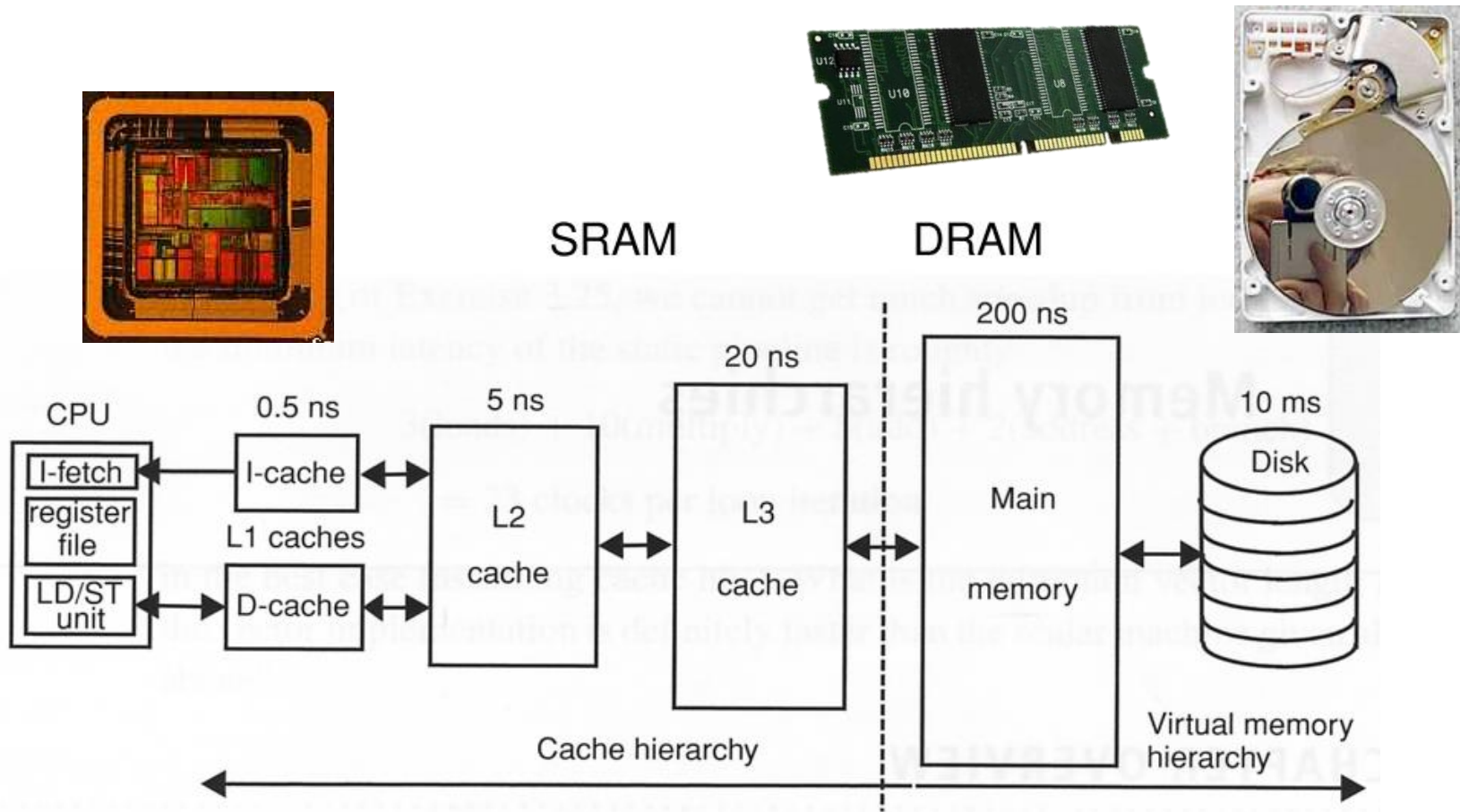
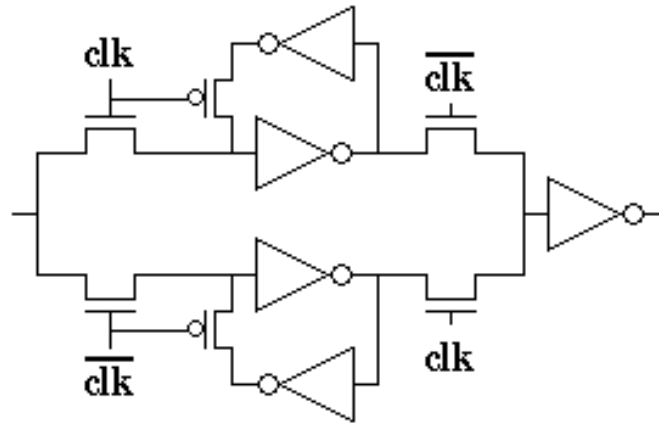


Figure 4.1. Typical memory hierarchy with three levels of caches.

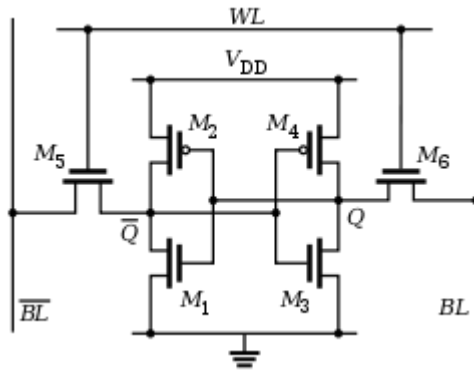
(From Dubois, Annavaram, Stenström: "Parallel Computer Organization and Design", ISBN 978-0-521-88675-8)



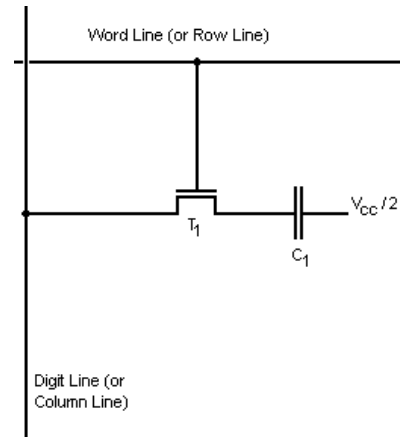
Register, SRAM, DRAM



DFF Cell (16T)



SRAM Cell (6T)



DRAM Cell (1T)



Levels of memory hierarchy

Capacity
Access Time
Cost/bit

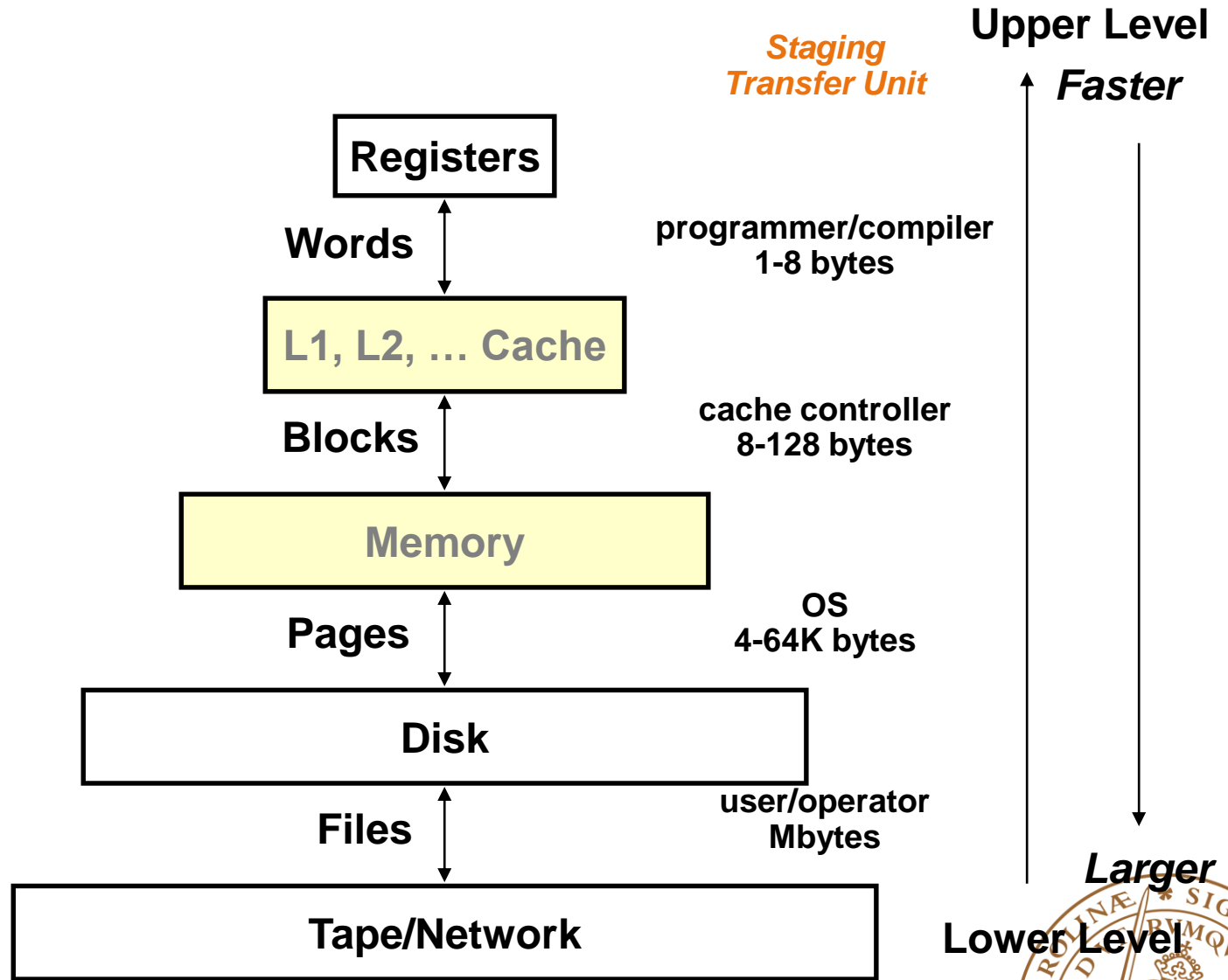
CPU Registers
 500 Bytes
 0.25 ns
 ~\$.01

Cache
 16K-1M Bytes
 1 ns
 ~\$.0001

Main Memory
 64M-2G Bytes
 100ns
 ~\$.0000001

Disk
 100 G Bytes
 5 ms
 10⁻⁵- 10⁻⁷ cents

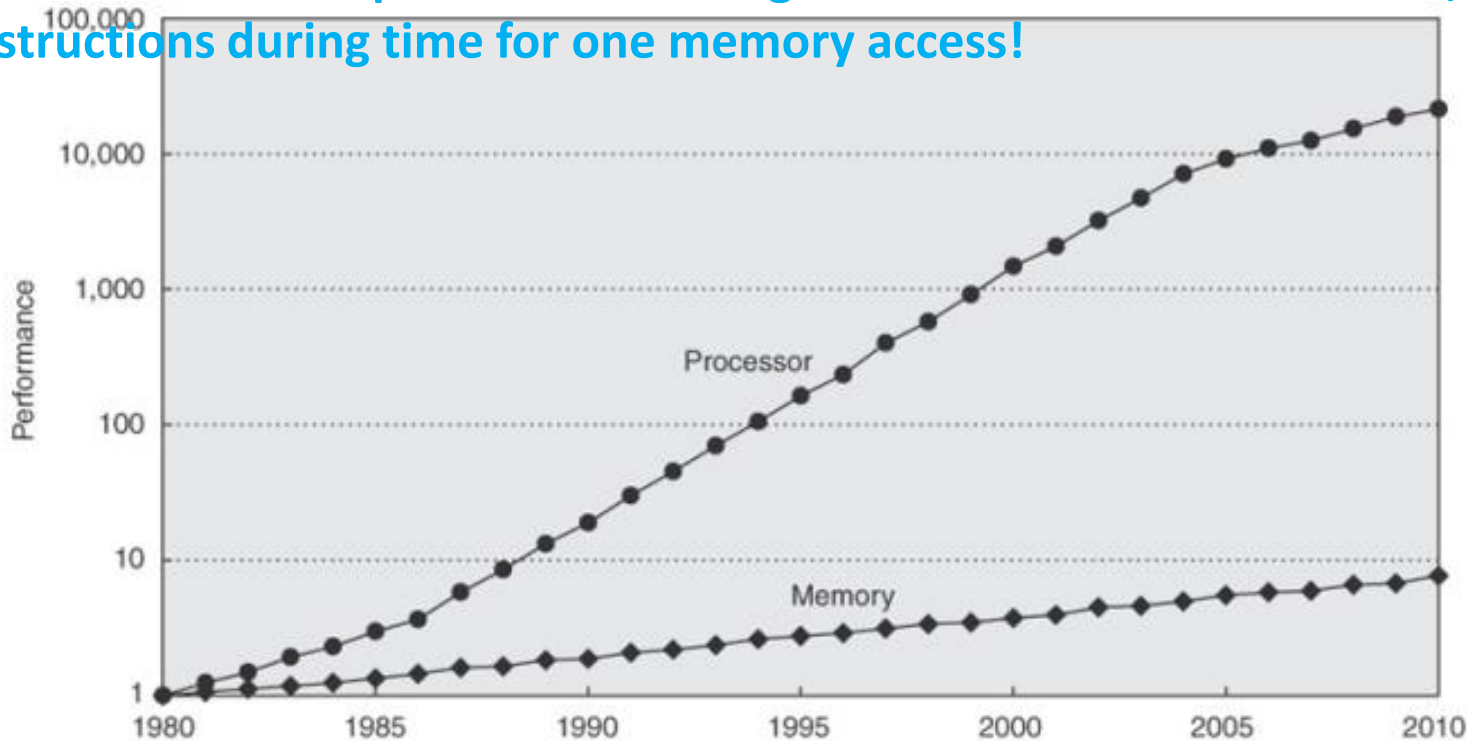
Tape/Network
 "infinite"
 secs.
 10⁻⁸ cents



The motivation

- ❑ 1980: no cache in microprocessors
- ❑ 1995: 2-level caches in a processor package
- ❑ 2000: 2-level caches on a processor die
- ❑ 2003: 3-level caches on a processor die

Four-issue 3GHz superscalar accessing 100ns DRAM could execute 1,200 instructions during time for one memory access!



Bandwidth example

Assume an “ideal” CPU with no stalls, running at 3 GHz and capable of issuing 3 instructions (32 bit) per cycle.

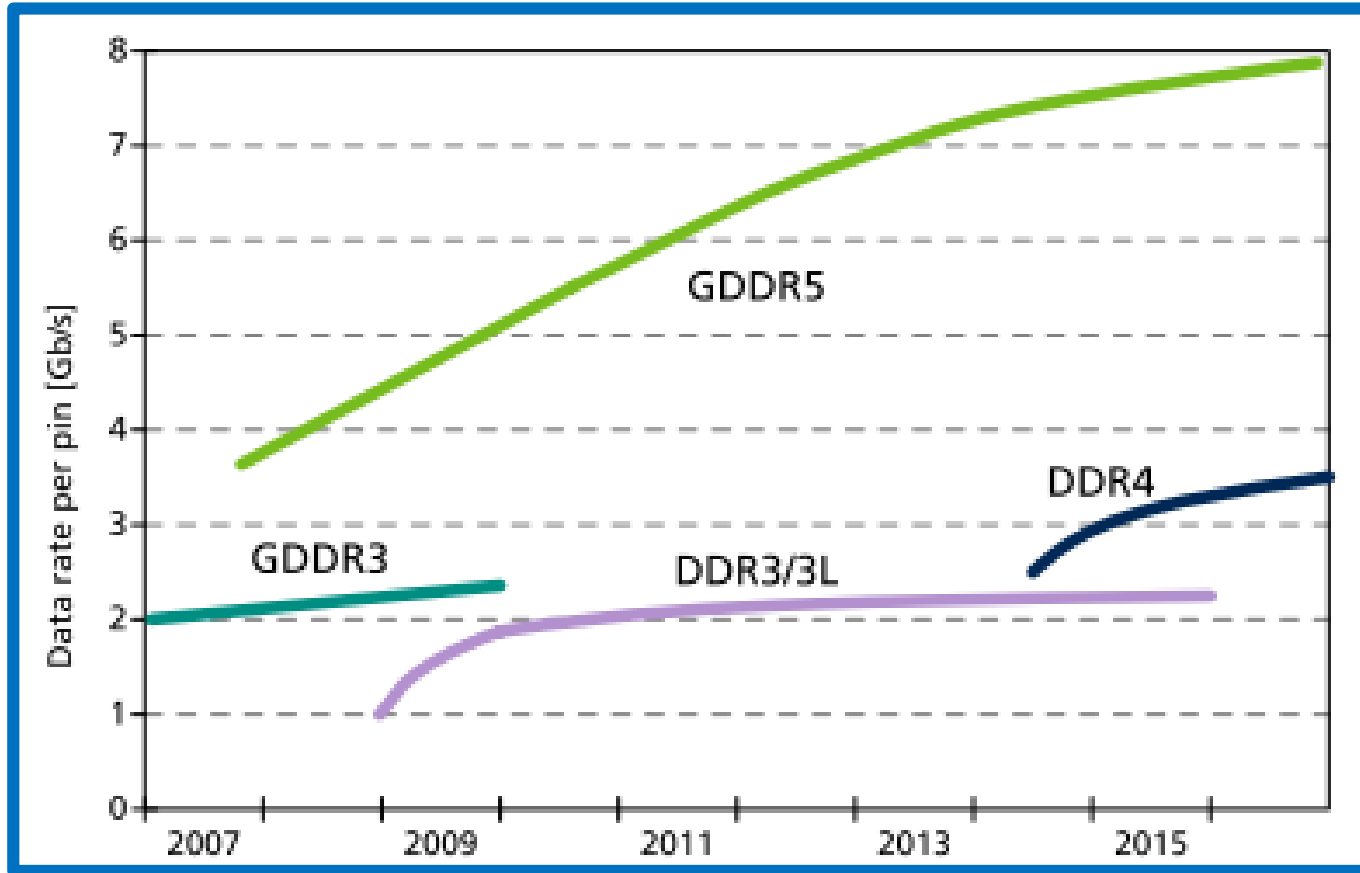
Instruction fetch	4	bytes/instr
Load & store	1	byte/instr
Instruction frequency	$3 * 3 = 9$	GHz
Bandwidth	$9 * (4 + 1) = 45$	GB/sec

DDR4-2800	2800 MT/s	PC-22400	22400 MB/s
DDR4-3000	3000 MT/s	PC-24000	17066 MB/s
DDR4-3200	3200 MT/s	PC-25600	25600 MB/s



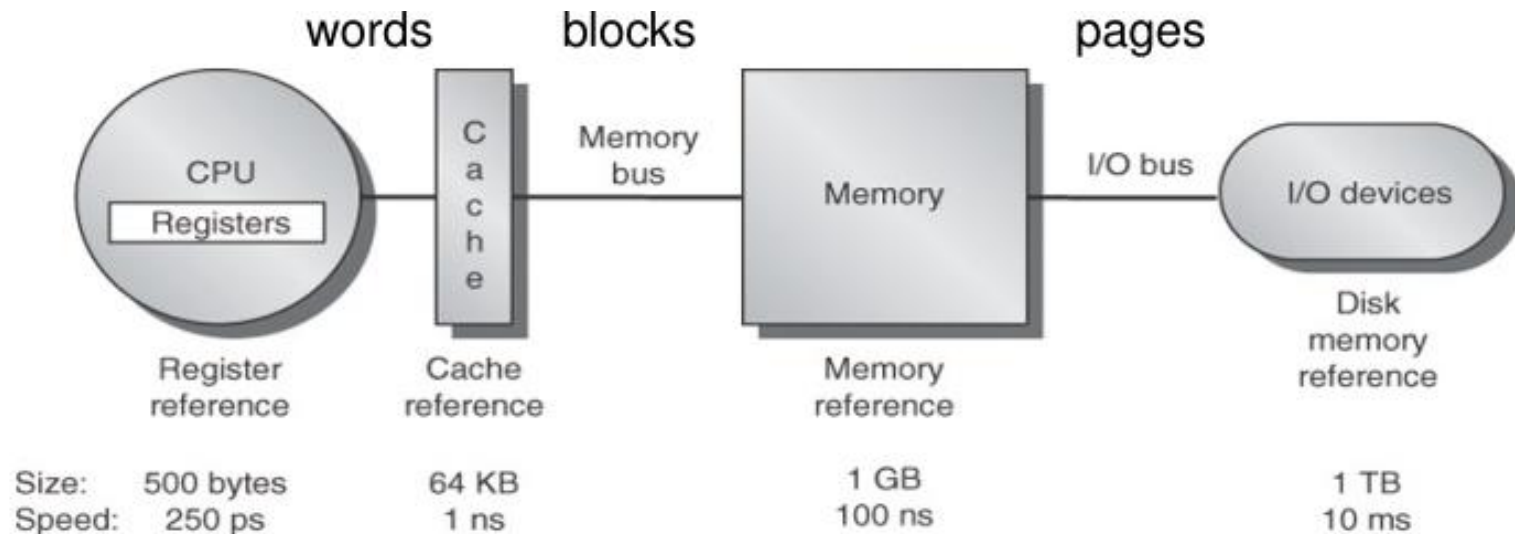
Main memory - trend

□ Interface



Memory hierarchy functionality

- CPU tries to access memory at address A. If A is in the cache, deliver it directly to the CPU
- If not – transfer a **block of memory words**, containing A, from the memory to the cache. Access A in the cache
- If A not present in the memory – transfer a **page of memory blocks**, containing A, from disk to the memory, then transfer the block containing A from memory to cache. Access A in the cache



© 2007 Elsevier, Inc. All rights reserved.



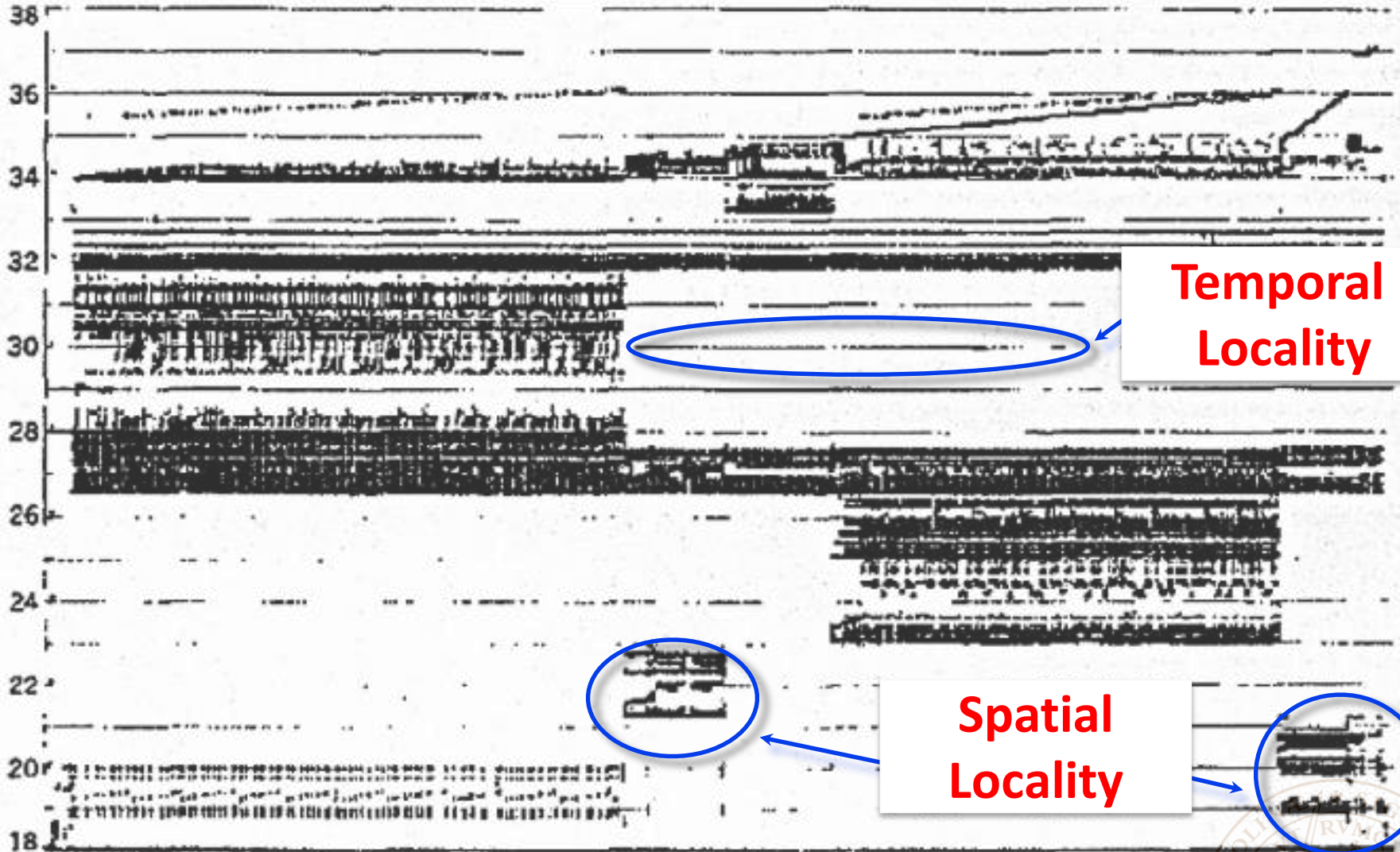
The principle of locality

- ❑ A program access a relatively **small portion** of the address space at any instant of time
- ❑ **Two different types of locality:**
 - **Temporal locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
 - **Spatial locality** (Locality in space): If an item is referenced, items whose addresses are close, tend to be referenced soon

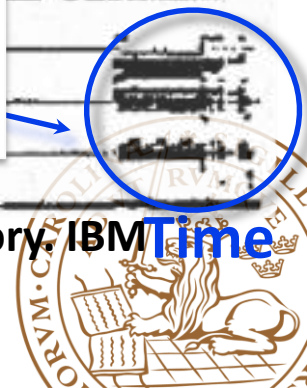


The principle of locality

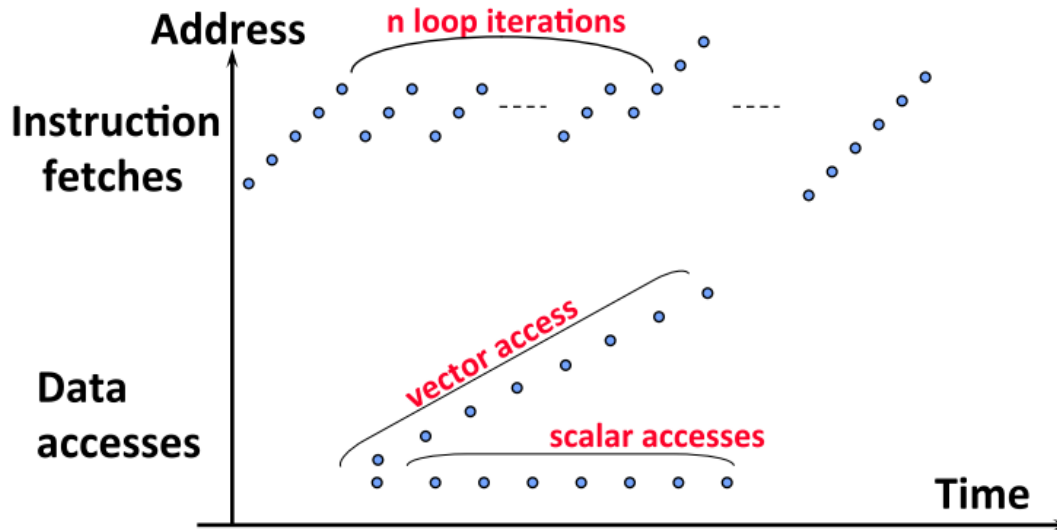
Memory Address (one dot per access)



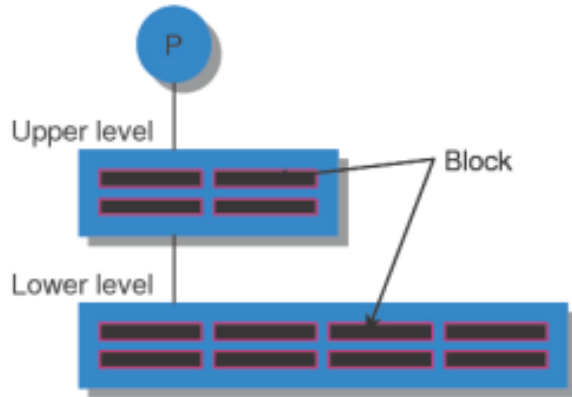
Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Time Systems Journal 10(3): 168-192 (1971)



The principle of locality



Memory hierarchy terminology



- $\text{Size}_{\text{upper}} < \text{Size}_{\text{lower}}$
- $\text{Access time}_{\text{upper}} < \text{Access time}_{\text{lower}}$
- $\text{Cost}_{\text{upper}} > \text{Cost}_{\text{lower}}$

Upper level = cache; Lower level = Main memory

- **Block**: The smallest amount of data moved between levels
- **Hit**: A memory reference that is satisfied in the cache
- **Miss**: A memory reference that *cannot* be satisfied in the cache



Outline

- Reiteration
- Memory hierarchy
- **Cache memory**
- Cache performance
- Main memory
- Summary



Cache measures

- ❑ **hit rate** = (# of accesses that hit)/(# of accesses)
 - close to 1, more convenient with
- ❑ **miss rate** = $1.0 - \text{hit rate}$
- ❑ **hit time**: cache access time plus time to determine hit/miss
- ❑ **miss penalty**: time to replace a block
 - measured in ns or # of clock cycles and depends on:
 - latency: time to get first word
 - bandwidth: time to transfer block

out-of-order execution can hide some of the miss penalty

- ❑ **Average memory access time** = hit time + miss rate * miss penalty



Four memory hierarchy questions

❑ Q1: Where can a block be placed in the upper level?

(Block placement)

❑ Q2: How is a block found if it is in the upper level?

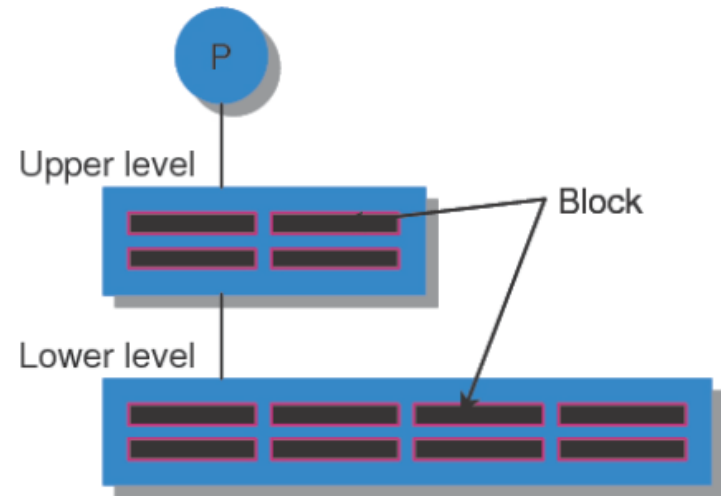
(Block identification)

❑ Q3: Which block should be replaced on a miss?

(Block replacement)

❑ Q4: What happens on a write?

(Write strategy)



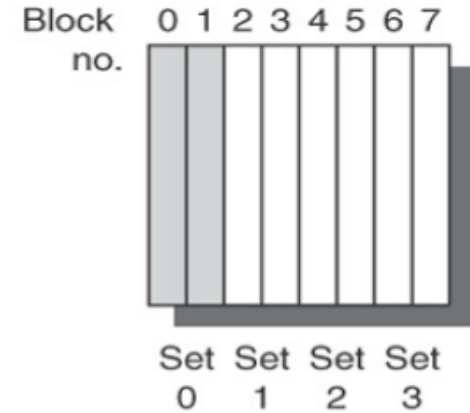
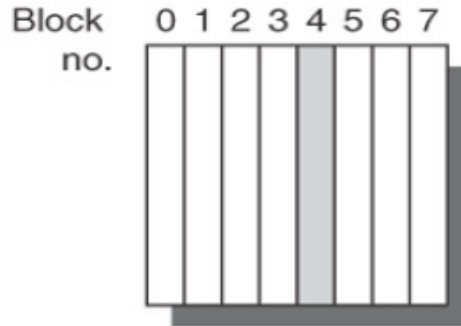
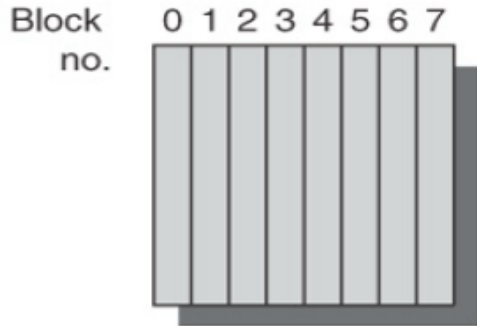
Block replacement

Fully associative:
block 12 can go
anywhere

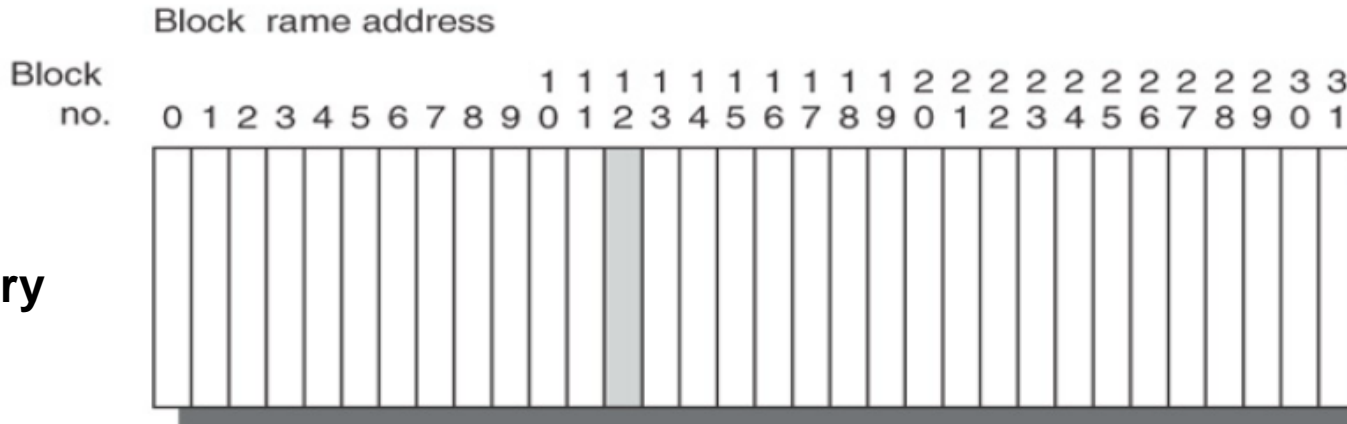
Direct mapped:
block 12 can go
only into block 4
($12 \bmod 8$)

Set associative:
block 12 can go
anywhere in set 0
($12 \bmod 4$)

cache



memory



Block replacement

□ Direct Mapped Cache

- Each memory location can only mapped to 1 cache location
- No need to make any decision => Current item replaces previous item in that cache location

□ N-way Set Associative Cache

- Each memory location have a choice of N cache locations

□ Fully Associative Cache

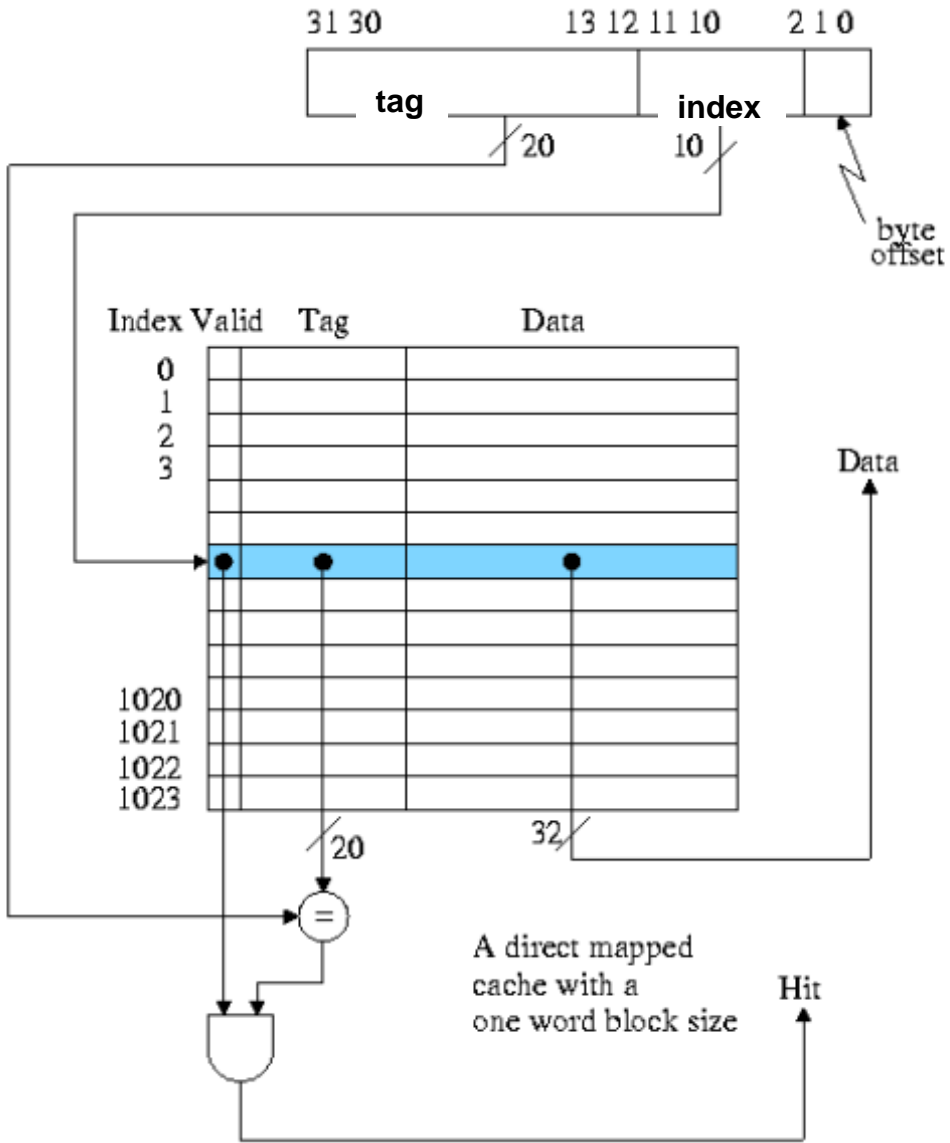
- Each memory location can be placed in ANY cache location

□ Cache miss in a N-way Set Associative or Fully Associative Cache

- Bring in new block from memory
- Throw out a cache block to make room for the new block
- **Need to decide which block to throw out!**



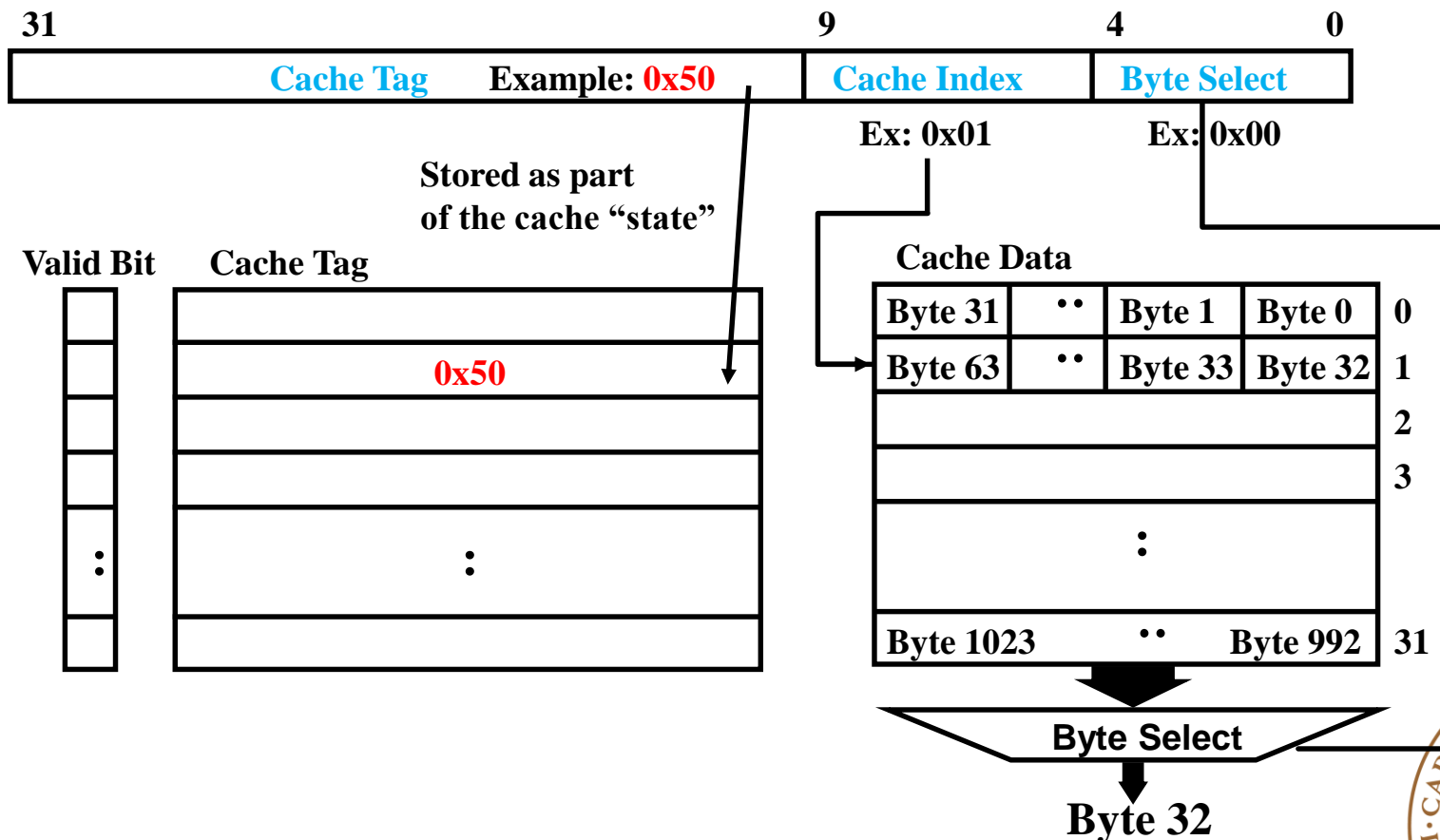
Block identification



Example: 1KB, Direct-Mapped, 32B Blocks

□ For a 1024 (2^{10}) byte cache with 32-byte blocks

- The uppermost **22** = (32 - 10) address bits are the tag
- The lowest **5** address bits are the Byte Select (Block Size = 2^5)
- The next **5** address bits (bit5 - bit9) are the Cache Index



Which block should be replaced on a Cache miss?

❑ Direct mapped caches don't need a block replacement policy (why?)

❑ Primary strategies:

- Random (easiest to implement)
- LRU – Least Recently Used (best, hard to implement)
- FIFO – Oldest (used to approximate LRU)

Associativity

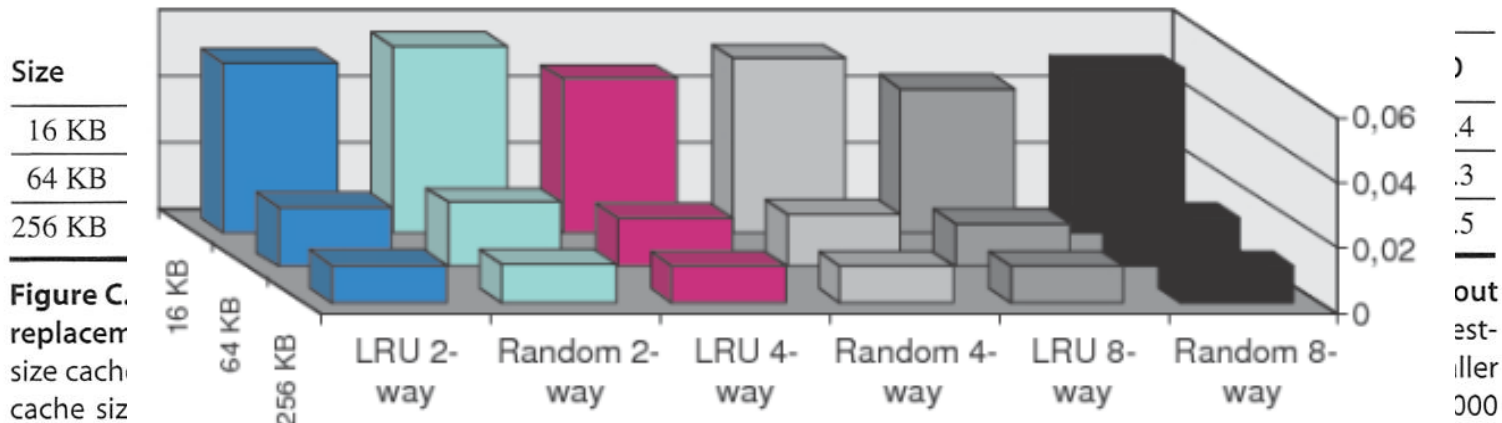


Figure C.1. Cache replacement policies for different cache sizes and associativities.

benchmarks. Five are from SPECint2000 (gap, gcc, gzip, mct, and perl) and five are from SPECfp2000 (appi, art, equake, lucas, and swim). We will use this computer and these benchmarks in most figures in this appendix.



Cache read

- CPU tries to read memory address A
- Search the cache to see if A is there

- | hit | miss |
|-----|--|
| ↓ | Copy the block that contains A from lower-level memory to the cache. (Possibly replacing an existing block.) |
- Send data to the CPU

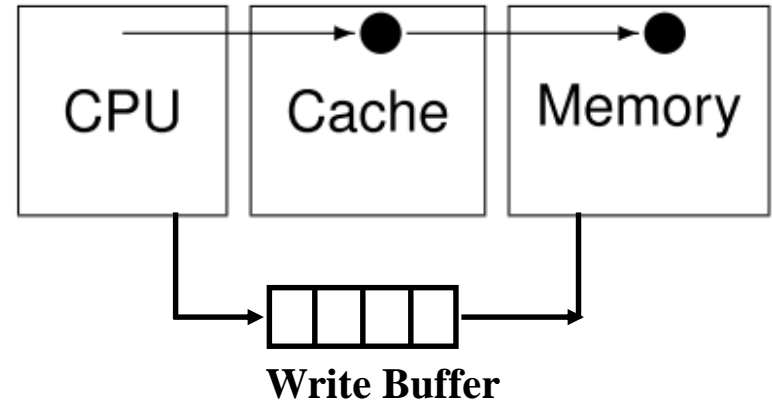
Reads dominate processor cache accesses and are more critical to processor performance



Cache write

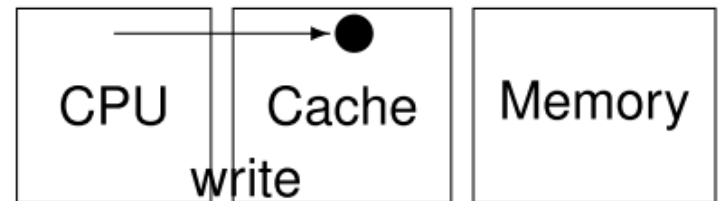
Write through:

- The information is written to both the block in the cache and to the block in the lower-level memory
- Is always combined with write buffers so that the CPU doesn't have to wait for the lower level memory

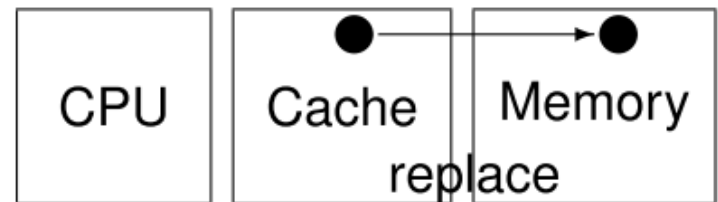


Write back:

- The information is written only to the block in the cache
- Copy a modified cache block to main memory only when replaced
- Is the block clean or modified? (dirty bit, several write to the same block)



...



On a write miss

□ Do we allocate a cache block on a write miss?

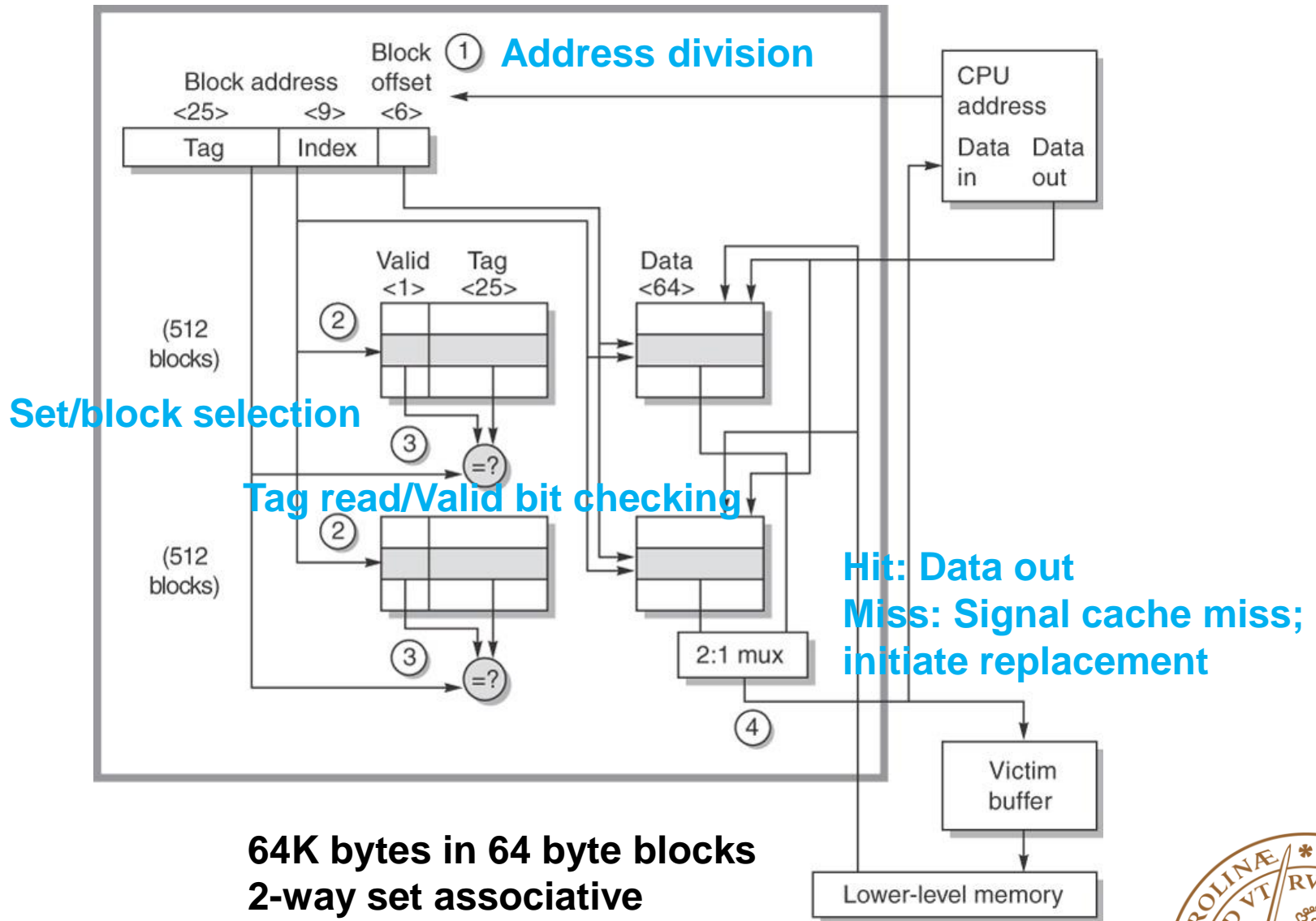
- Write allocate (allocate a block in the cache)
- No-write allocate (no cache block is allocated. Write is only to main memory, or next level of hierarchy)

□ General combination

- A **write-back** cache uses **write allocate**, hoping for subsequent writes (or even reads) to the same location, which is now cached.
- A **write-through** cache uses **no-write allocate**. Here, subsequent writes have no advantage, since they still need to be written directly to the backing store.



Cache micro-ops sequencing (AMD Opteron)



© 2007 Elsevier, Inc. All rights reserved.



Outline

- Reiteration
- Memory hierarchy
- Cache memory
- **Cache performance**
- Main memory
- Summary



Cache performance

*memory stall cycles*_{cache}

$$= IC * (CPI_{execution} + \frac{mem\ accesses}{instruction} * miss\ rate * miss\ penalty) * T_C$$



Interlude – CPU performance equation

CPU execution Time =

$$IC * (CPI_{execution} + CPI_{cache}) * T_C =$$

$$IC * (CPI_{ideal} + CPI_{pipeline} + CPI_{cache}) * T_C =$$

$$IC * (CPI_{ideal} +$$

Structural Stalls + Data Hazard Stalls + Control Stalls +

$$\frac{mem\ accesses}{instruction} * miss\ rate * miss\ penalty$$

$$) * T_C$$



Cache performance

$$\text{CPU execution Time} = IC * (CPI_{\text{execution}} + \frac{\text{mem accesses}}{\text{instruction}} * \text{miss rate} * \text{miss penalty}) * T_C$$

□ Three ways to increase performance:

- Reduce miss rate
- Reduce miss penalty
- Reduce hit time (improves T_C)

$$\text{Average memory access time} = \text{hit time} + \text{miss rate} * \text{miss penalty}$$



Cache performance, example

$$\text{CPU execution Time} = IC * (CPI_{\text{execution}} + \frac{\text{mem accesses}}{\text{instruction}} * \text{miss rate} * \text{miss penalty}) * T_C$$

Example:

miss rate (%)	1
miss penalty (cycles)	50
$\frac{\text{mem accesses}}{\text{instruction}}$	k
CPI increase	$k * 0.01 * 50$



Sources of Cache miss

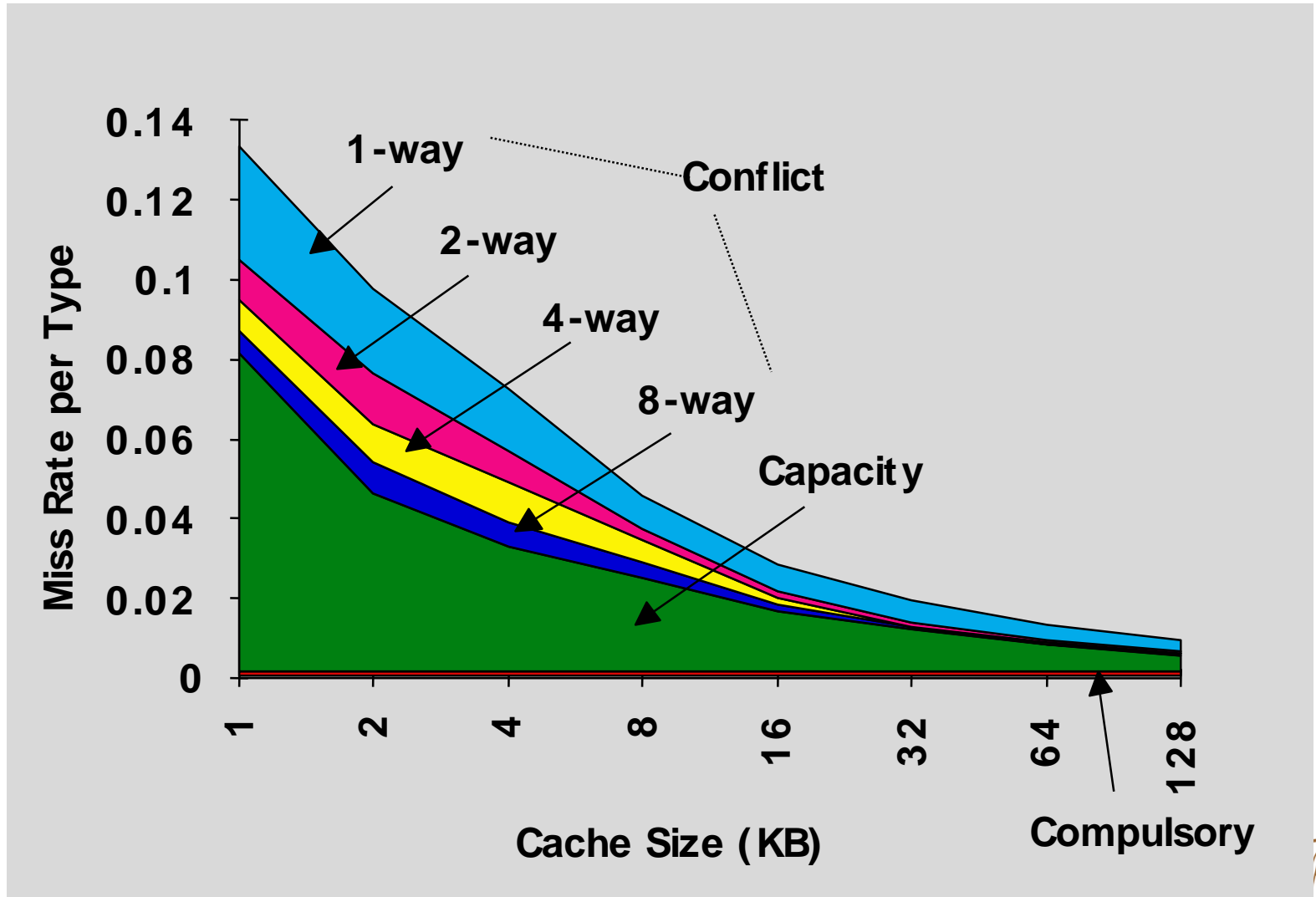
□ A cache miss can be classified as a:

- **Compulsory miss**: The first reference is always a miss
- **Capacity miss**: If the cache memory is too small it will fill up and subsequent references will miss
- **Conflict miss**: Two memory blocks may be mapped to the same cache block with a direct or set-associative address mapping

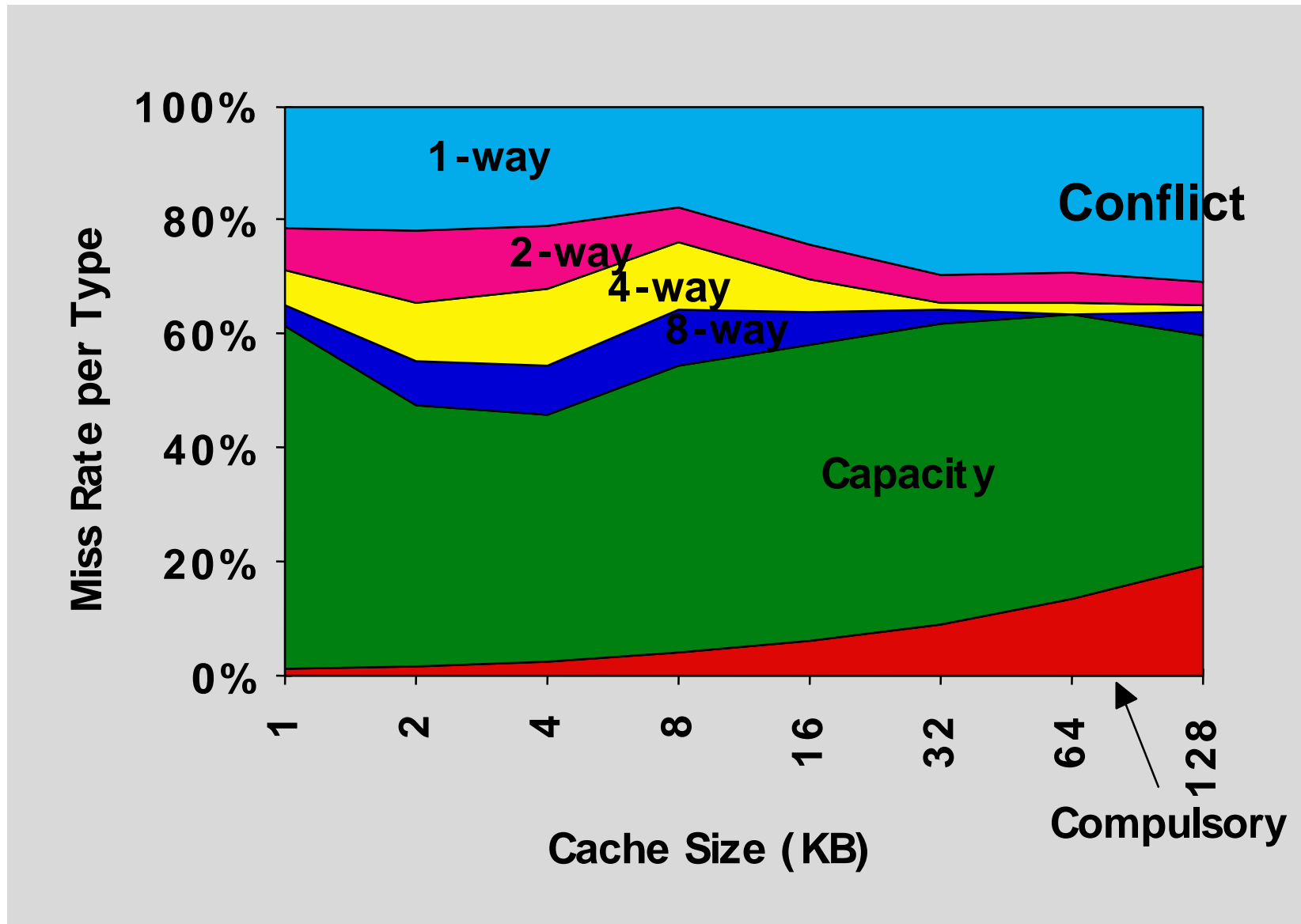
3 C's



Miss rate components – 3 C's



Miss rate (relative) components – 3 C's



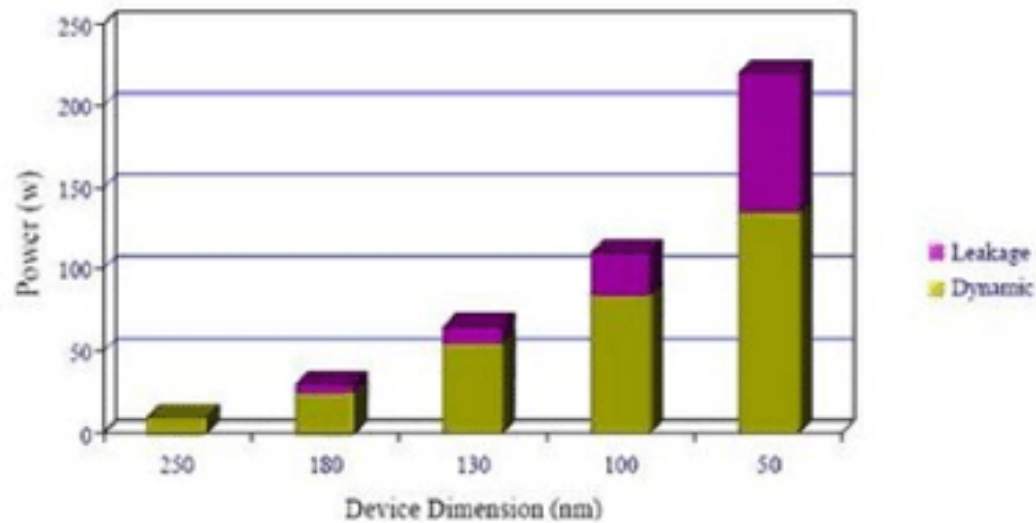
Miss rate components

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low(er)	Medium	High



Cache size: power

Size	Leakage	Dynamic
8M Byte	76mW	30mW



Miss rate components – 3 C's

- ❑ Small percentage of compulsory misses
- ❑ Capacity misses are reduced by larger caches
- ❑ Full associativity avoids all conflict misses
- ❑ Conflict misses are relatively more important for small set-associative caches

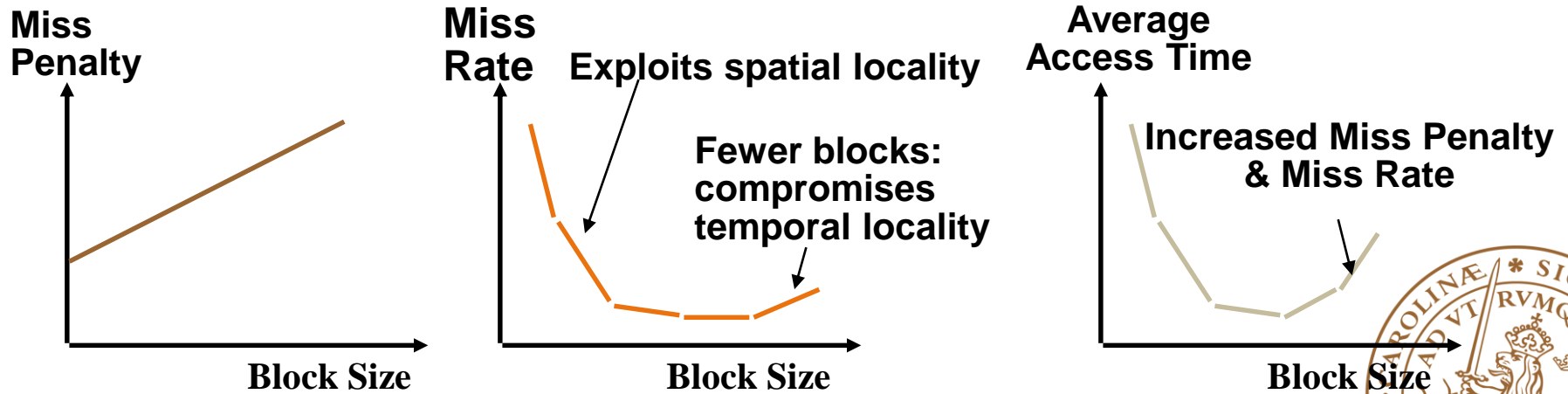


Block size tradeoff

□ In general, larger block size

- Take advantage of spatial locality, BUT
- Larger block size means larger miss penalty => Takes longer time to fill up the block
- If block size is too big relative to cache size, miss rate will go up => Too few cache blocks

Average memory access time =
 $hit\ time + miss\ rate * miss\ penalty$



Outline

- Reiteration
- Memory hierarchy
- Cache memory
- Cache performance
- **Main memory**
- Summary



Main memory - background

□ Performance of main memory:

- Latency affects: Cache miss penalty
 - Access time: time between request and word arrives
 - Cycle time: time between requests
- Bandwidth affects: I/O, multiprocessors (& cache miss penalty)

□ Main memory is DRAM: Dynamic RAM

- Dynamic - memory cells need to be refreshed
- 1 transistor and a capacitor per bit
- Complicated addressing

□ Cache memory is SRAM: Static RAM

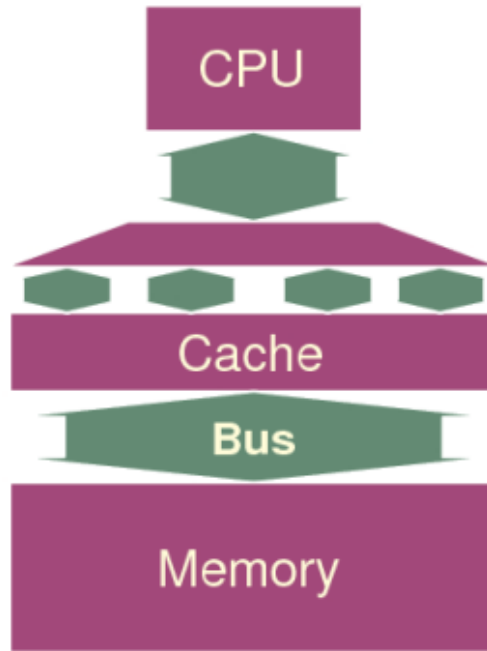
- No refresh
- 6 transistors per bit
- Simple addressing



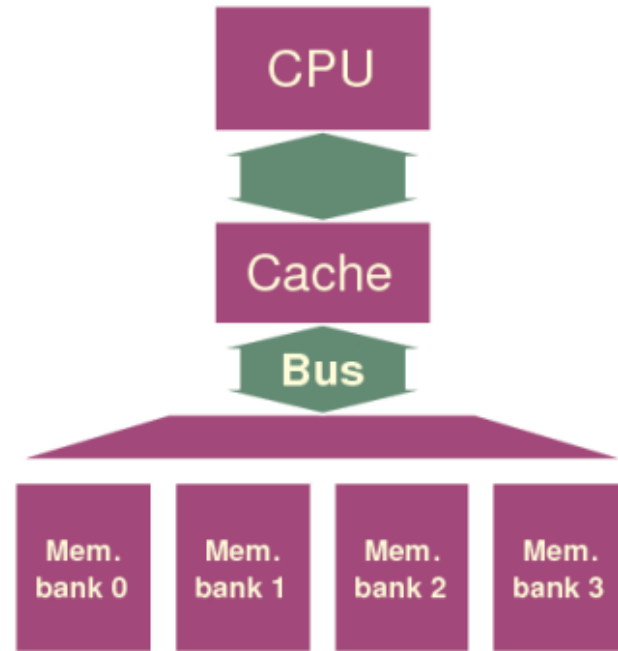
Improving main memory performance



normal
memory



wide memory



interleaving

Improves bandwidth.



Interleaving

Normal

addr	access + cycle	data
------	----------------	------

addr	access + cycle	data
------	----------------	------

addr	access + cycle	data
------	----------------	------

addr	access + cycle	data
------	----------------	------

Interleaving 4-way

addr	access + cycle	data
------	----------------	------

addr	access + cycle	data
------	----------------	------

addr	access + cycle	data
------	----------------	------

addr	access + cycle	data
------	----------------	------



Summary

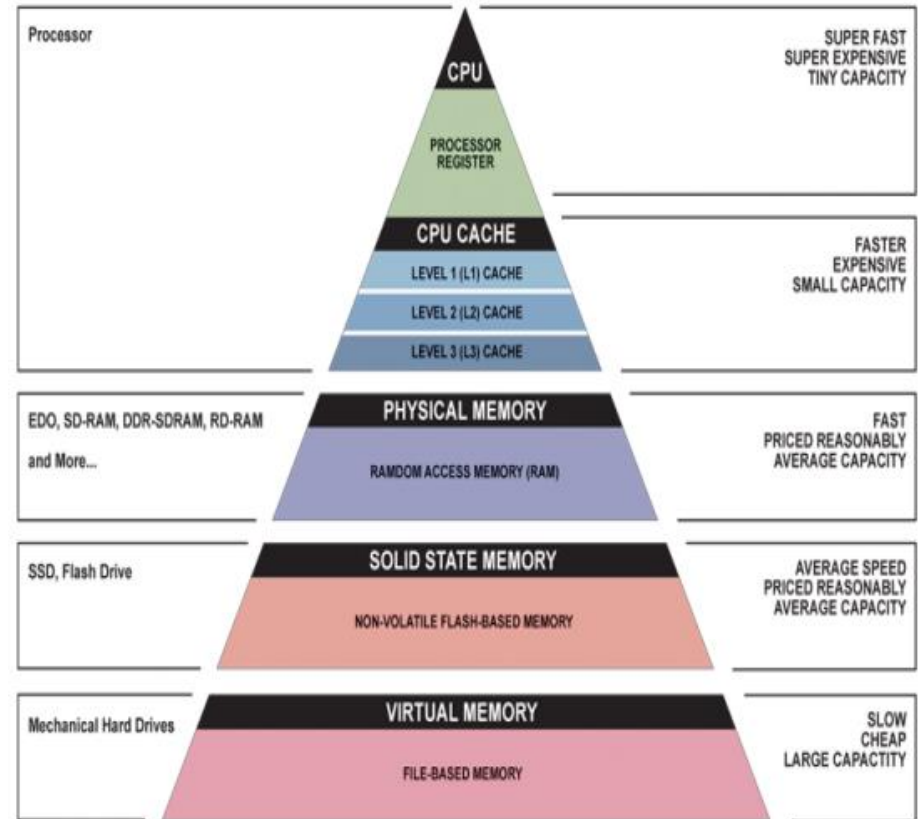
□ The performance gap between CPU and memory is widening

□ Memory Hierarchy

- Cache level 1
- Cache level ...
- Main memory
- Virtual memory

□ Four design issues:

- Block placement
- Block identification
- Block replacement
- Write strategy



▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng



Summary

- ❑ Cache misses increases the CPI for instructions that access memory

$$\text{Average memory access time} = \text{hit time} + \text{miss rate} * \text{miss penalty}$$

- ❑ Three types of misses:

- Compulsory
- Capacity
- Conflict

- ❑ Main memory performance:

- Latency
- Bandwidth

