EITF20: Computer Architecture

Part4.1.1: Cache - 1

Liang Liu
liang.liu@eit.lth.se

Lectures Next Week

- 2018-11-27: Exercise I
- 2018-11-29: No Lecture

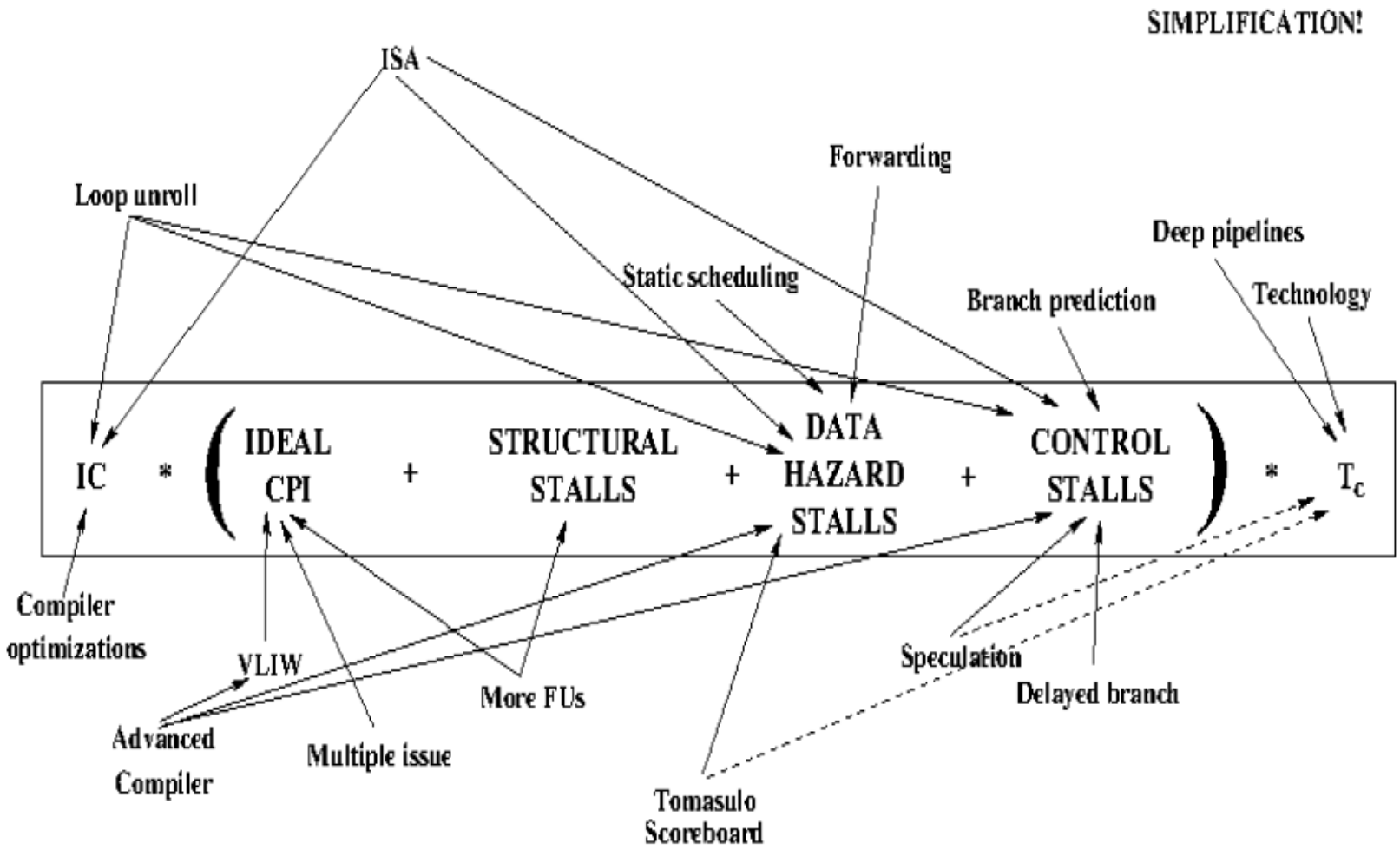
Outline

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

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

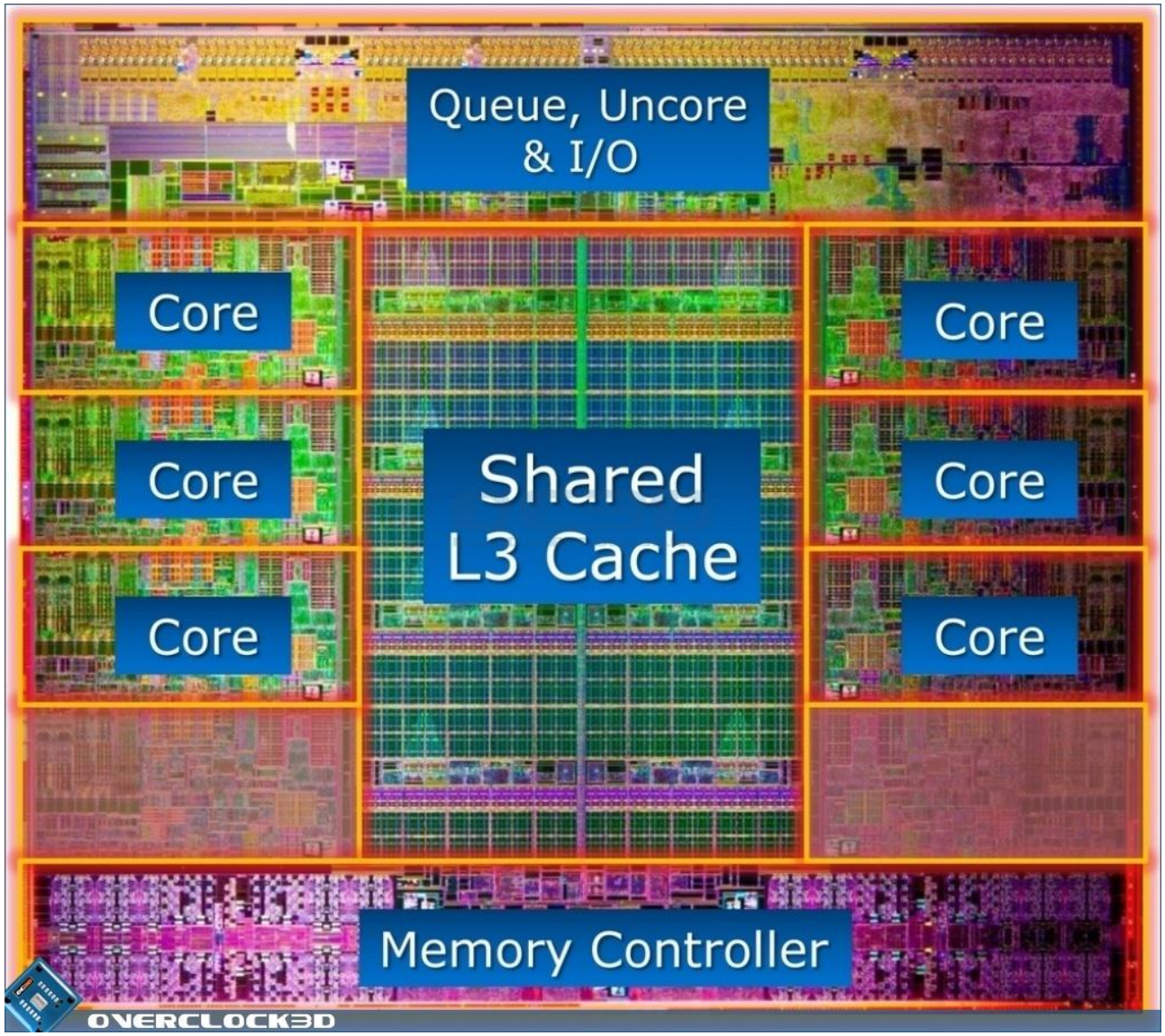
CPU performance equation



Outline

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

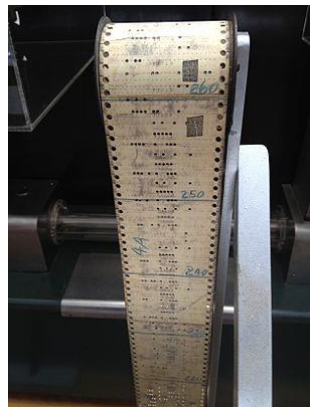
Intel core i7 chip



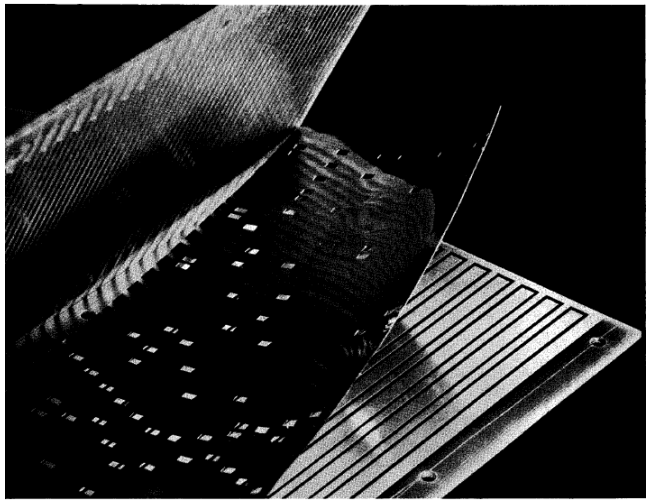
Memory in early days

it can print lists or statements at 1,100 lines a minute, it can read paper tape at 1,000 characters a second

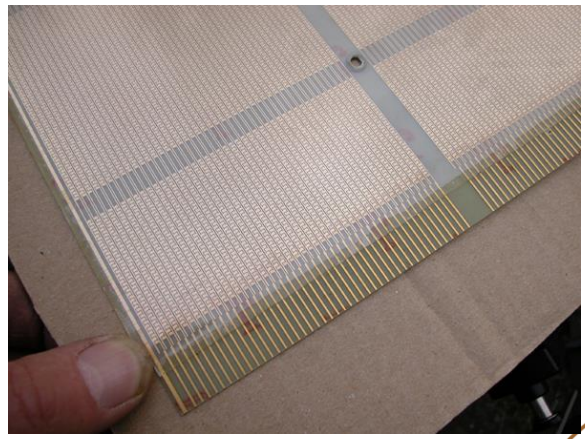
Punched cards, From early 1700s through Jaquard Loom, Babbage, and then IBM



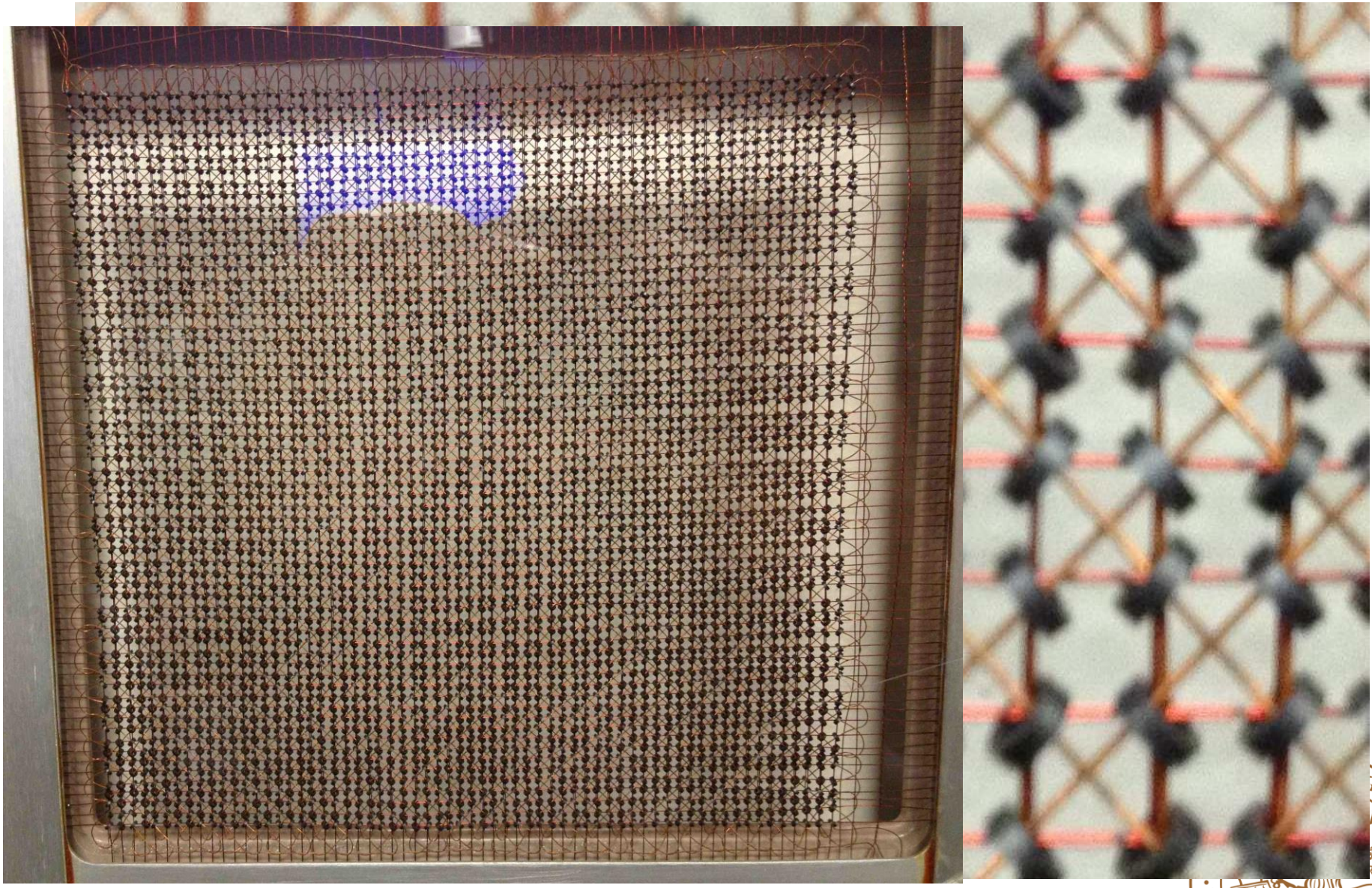
Punched paper tape, instruction stream in Harvard Mk 1 (1950s)



IBM Card Capacitor ROS (360)

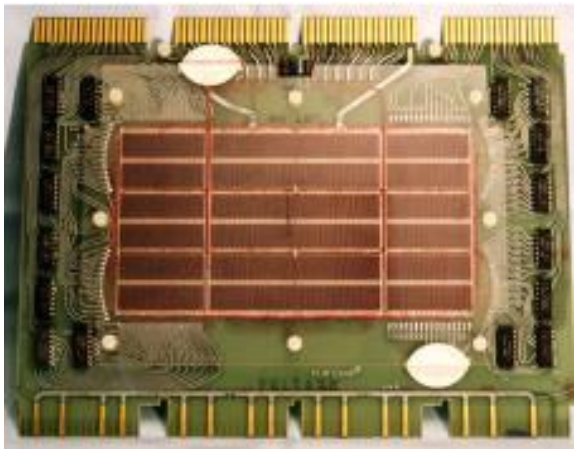


IBM Balanced Capacitor ROS (1968)



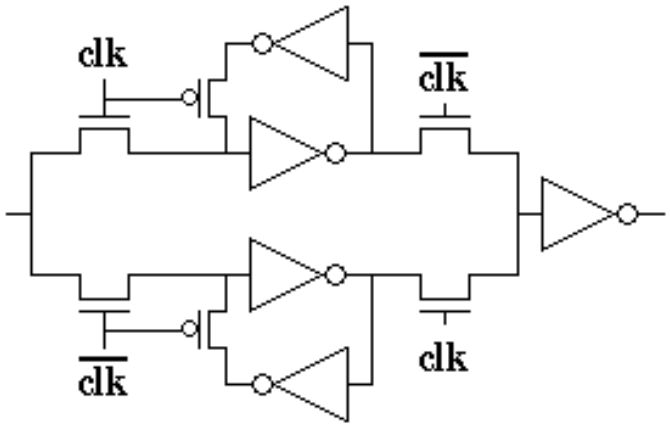
Core Memory

- 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

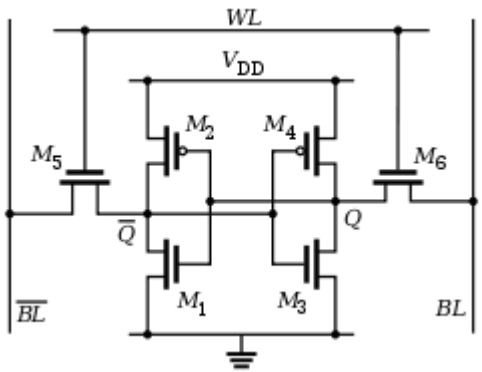


http://royal.pingdom.com/2008/04/08/the-history-of-computer-data-storage-in-pictures/

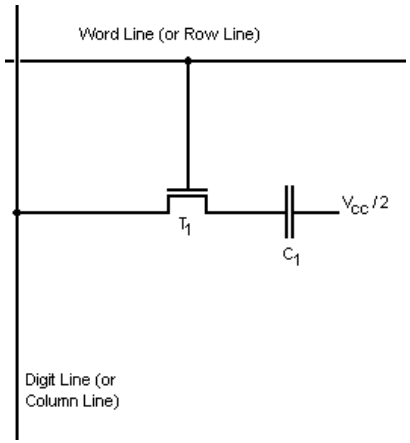
Semiconductor Memory: Register, SRAM, DRAM



Register (DFF) Cell (16T)

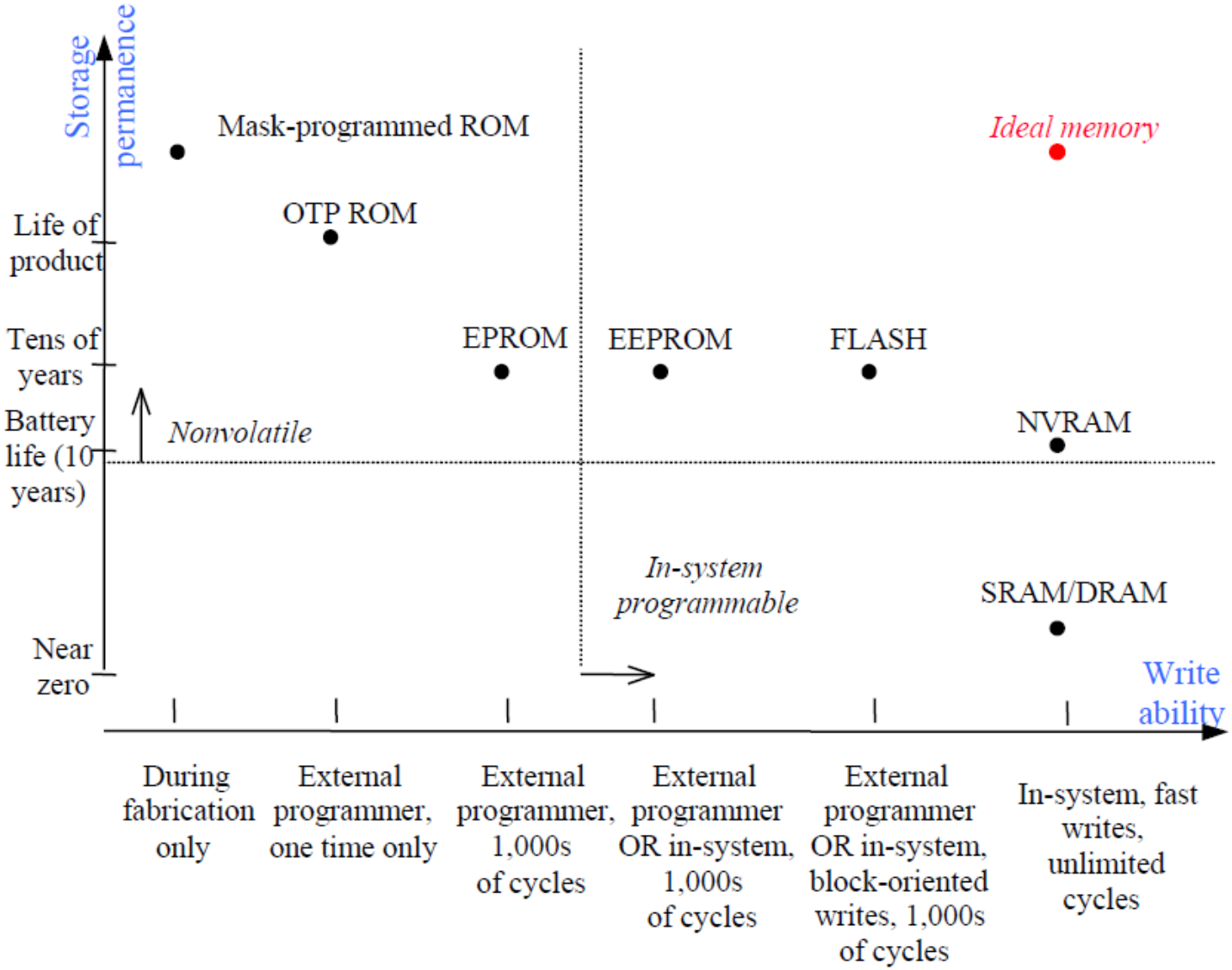


SRAM Cell (6T)



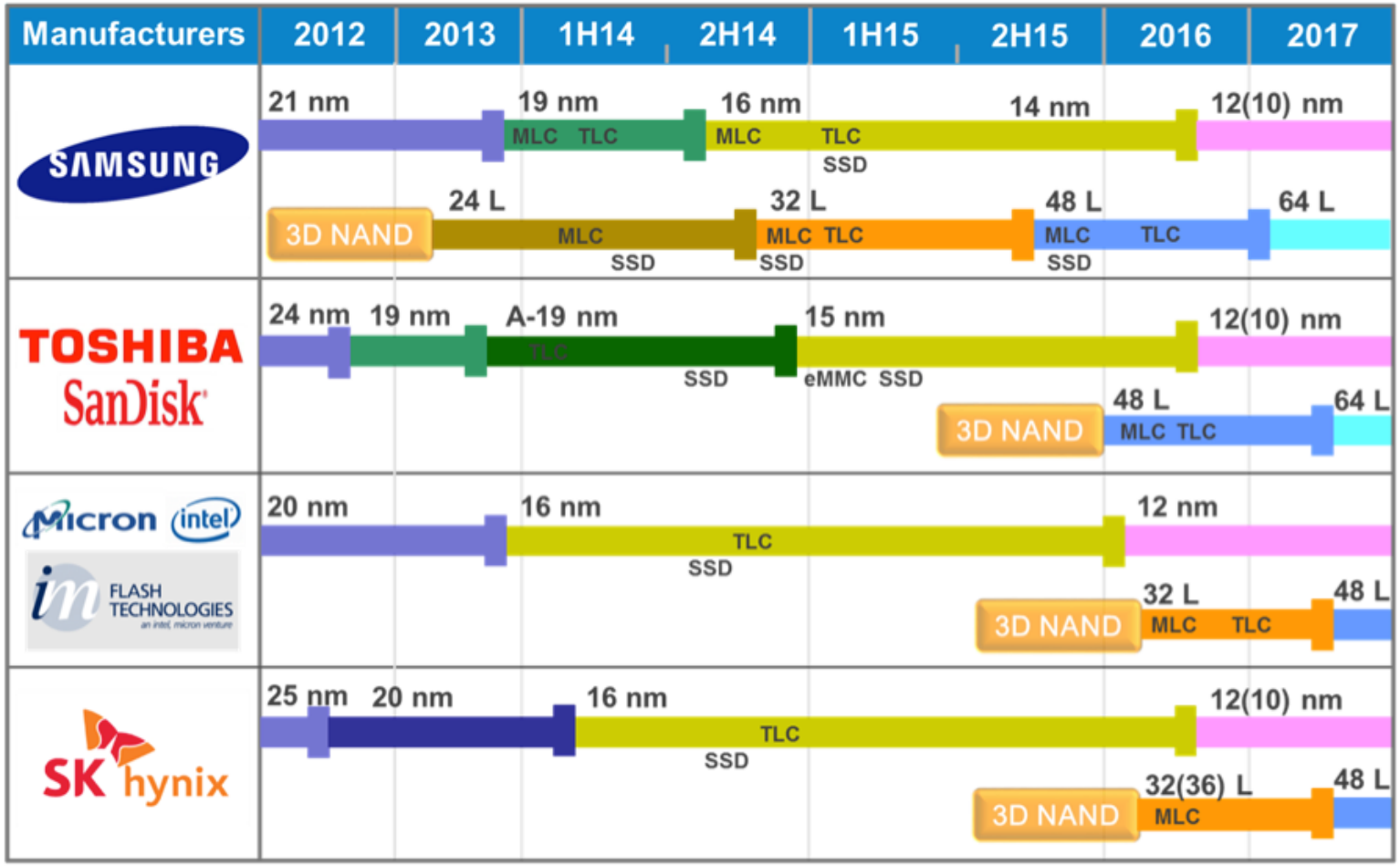
DRAM Cell (1T)

Memory Classification



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

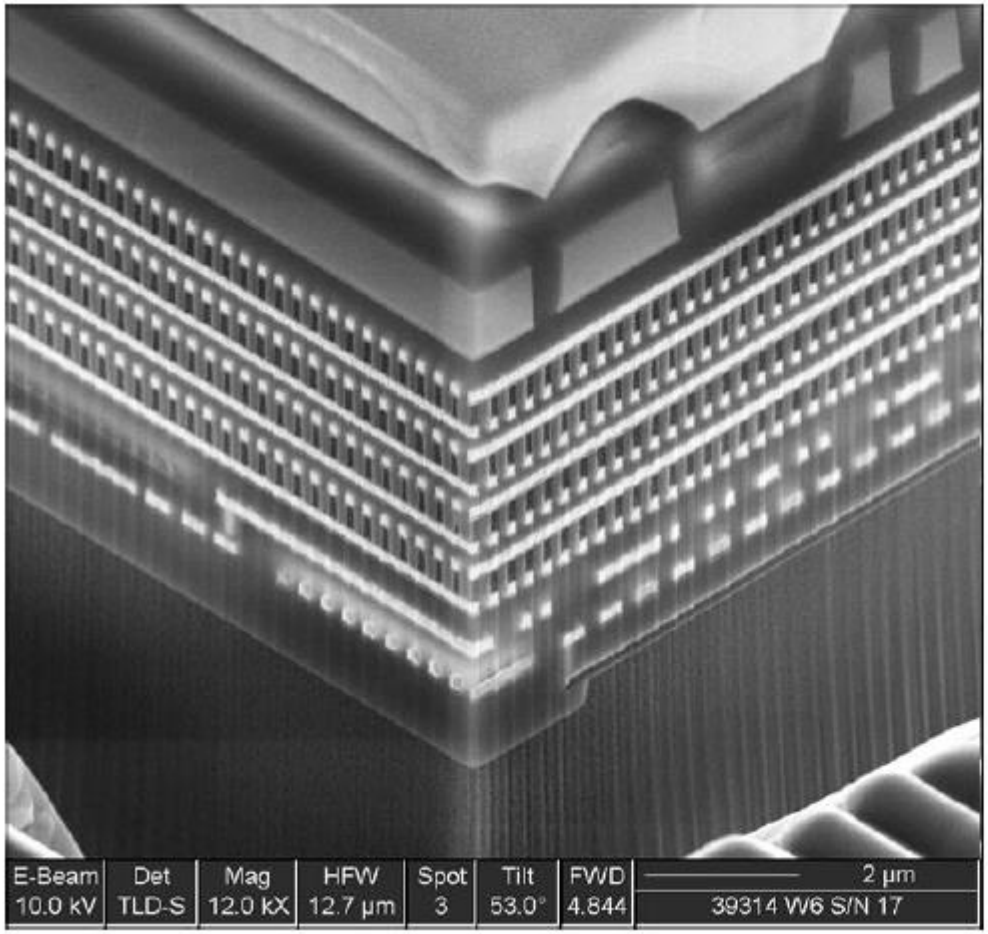
Semiconductor memory



First 32nm NAND Flash memory, 2009, Toshiba

First 32nm CPU released, 2010, Intel Core i3

Semiconductor memory

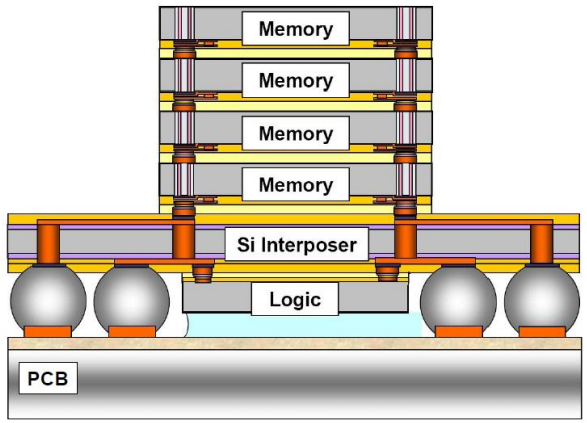


Al top metal

4 layers of memory cells + tungsten interconnect

2 levels of tungsten routing

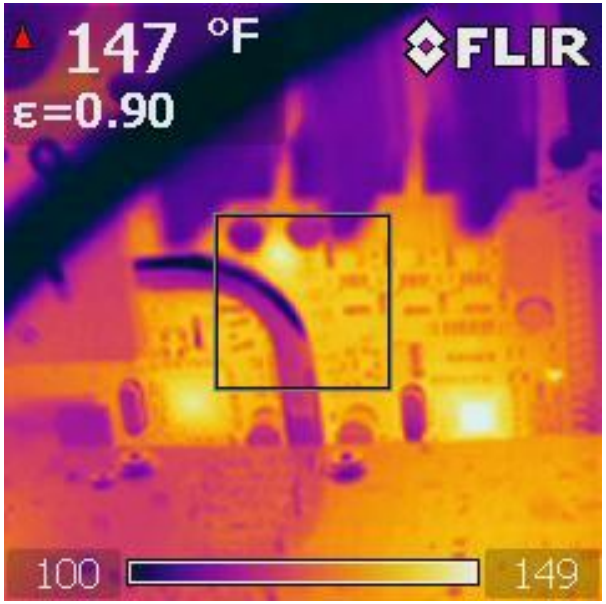
LV + HV CMOS logic



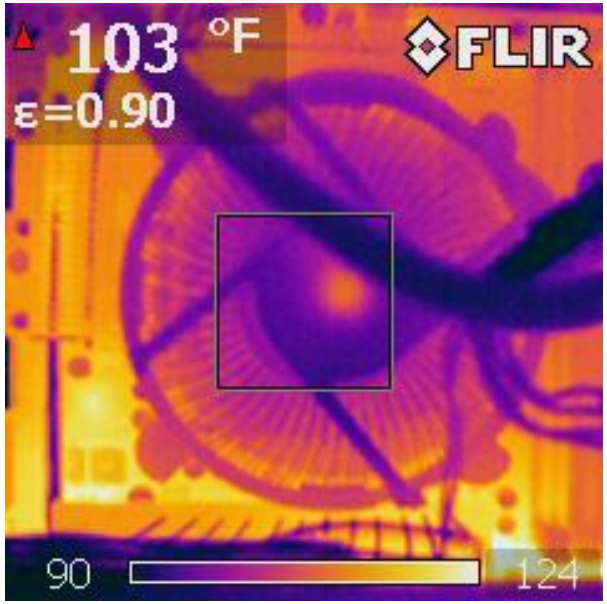
Example of 3-D integrated construction *(Image courtesy of DuPont Electronics)*

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

Themal imaging



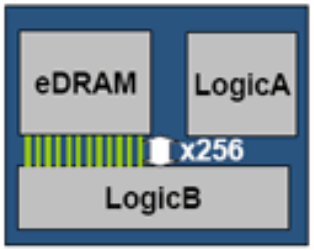
ASUS motherboard with an i7 quad core processor and triple channel memory.



The stock Intel cooler for quad core i7 processor

Embedded DRAM

Embedded DRAM



ON Chip DRAM
- Connect directly with logic
- Large band width

External DRAM

OFF Chip DRAM
- Connect with logic chip by base bonding
- Small band width

Intel's Embedded DRAM: New Era of Cache Memory

Overcoming SRAM's scaling

Embedded DRAM

Use of eDRAM in various products

Product name	Amount of eDRAM
Intel Haswell, Iris Pro Graphics 5200 (GT3e)	128 MB
Intel Broadwell, Iris Pro Graphics 6200 (GT3e)	128 MB
Intel Skylake, Iris Graphics 540 and 550 (GT3e)	64 MB
Intel Skylake, Iris Pro Graphics 580 (GT4e)	64 or 128 MB
PlayStation 2	4 MB
Xbox 360	10 MB
Wii U	32 MB

Meomory design

You want a memory to be:

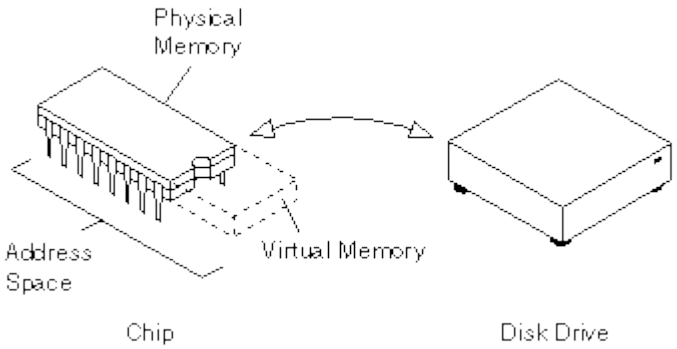
FAST

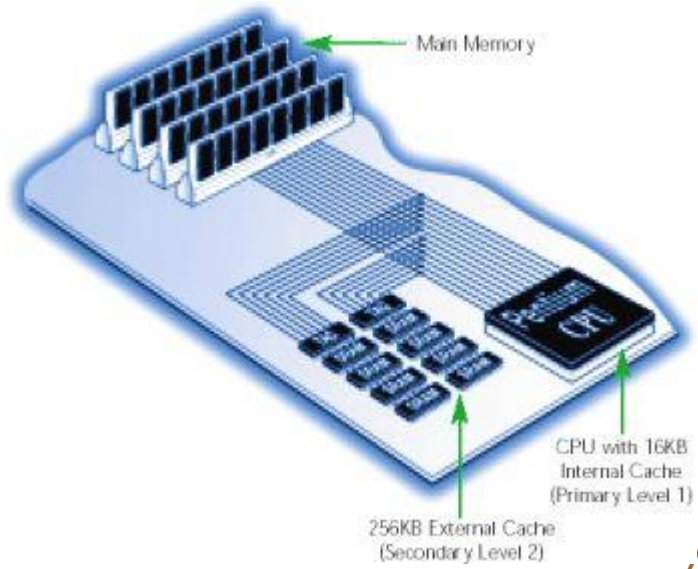
BIG

Memory, big fast, cheap

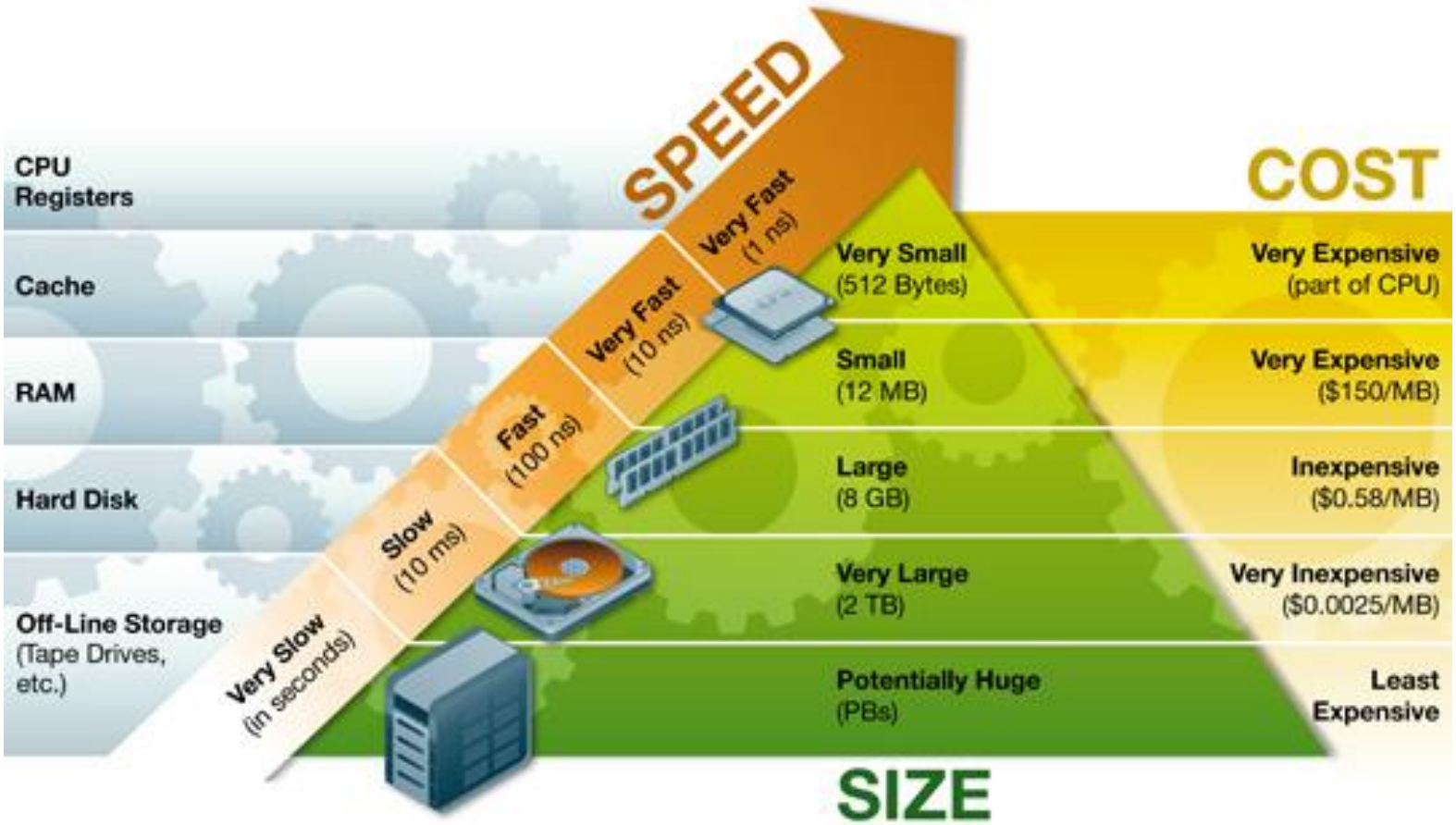
Use two “magic” tricks (from architecture)

- 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

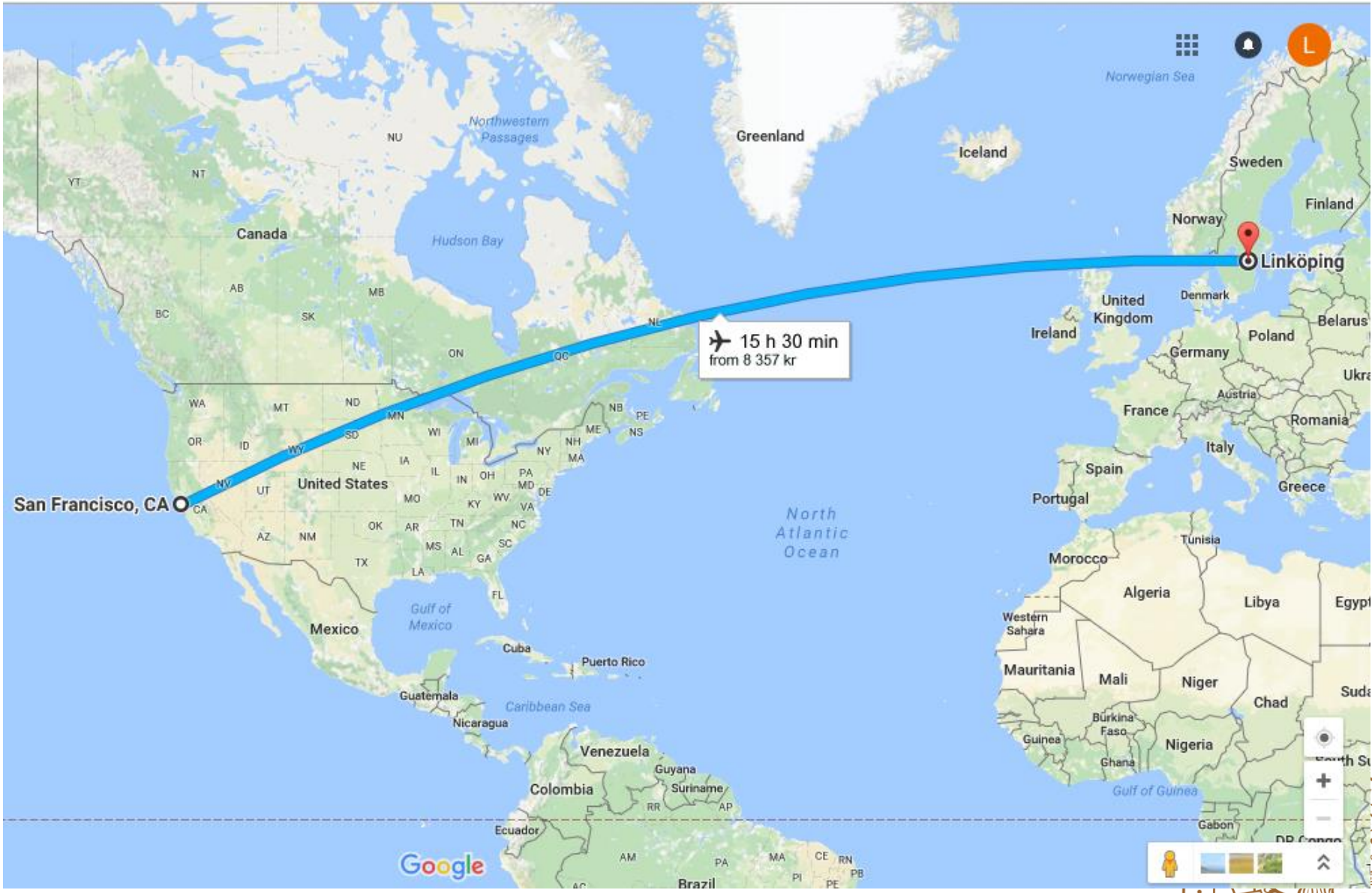




Levels of memory hierarchy



Similar Concept by Airlines



Hierarchy, Heterogeneous

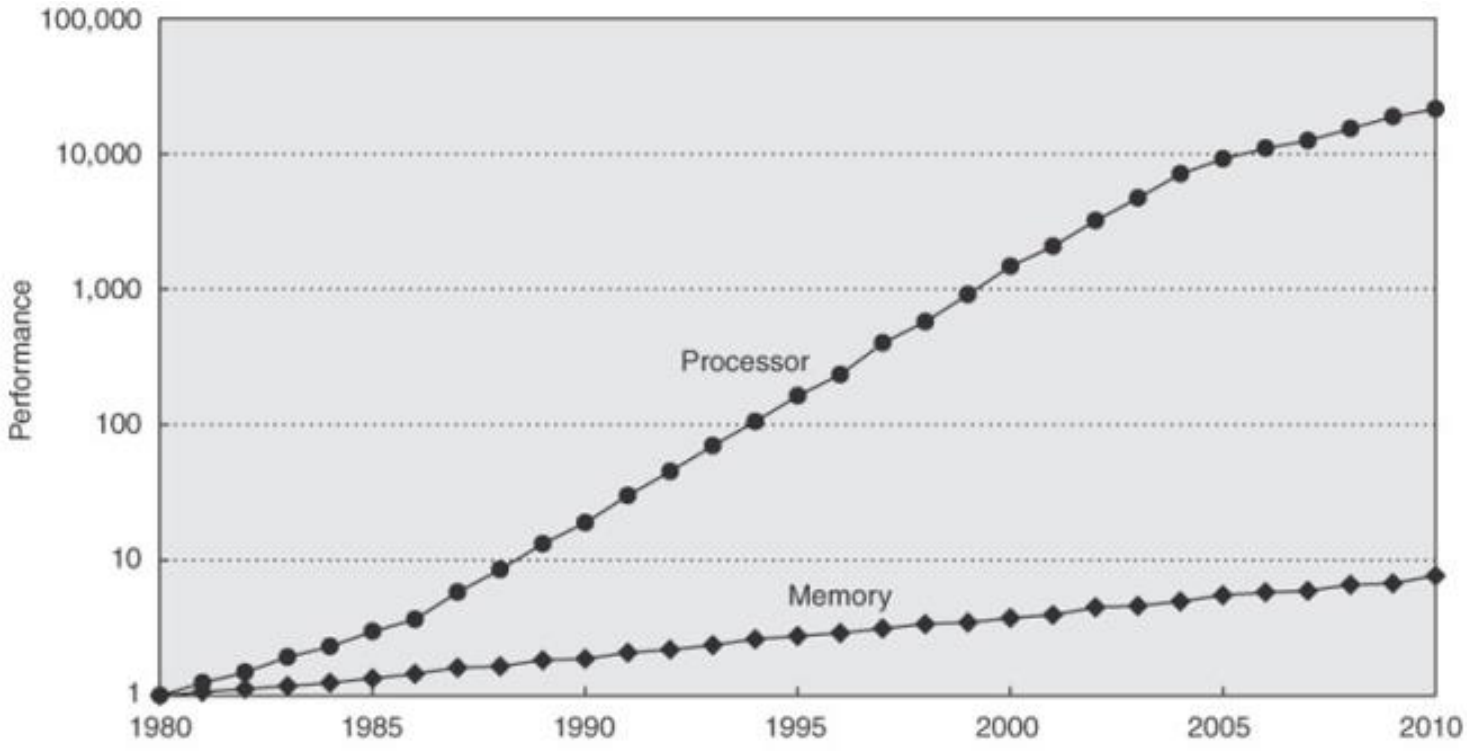
Linköping

Copenhagen

Hierarchy, Heterogeneous

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

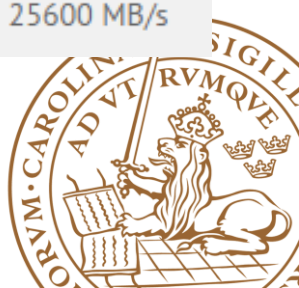


Bandwidth example

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

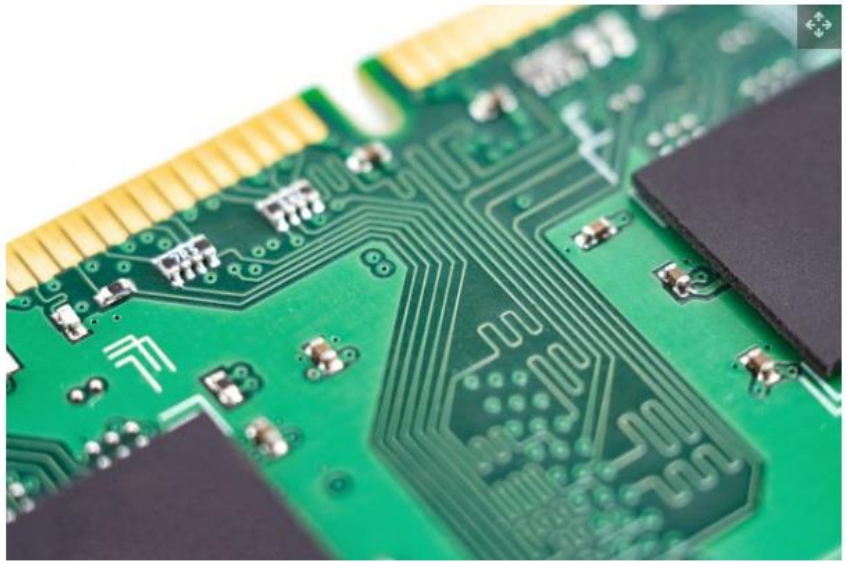
DDR4-2800	2800 MT/s	PC-22400	22400 MB/s
DDR4-3000	3000 MT/s	PC-24000	17066 MB/s
DDR4-3200	3200 MT/s	PC-25600	25600 MB/s



Bandwidth example

SK Hynix, Samsung Detail the DDR5 Products Arriving This Year

By Nathaniel Mott February 23, 2019 Components



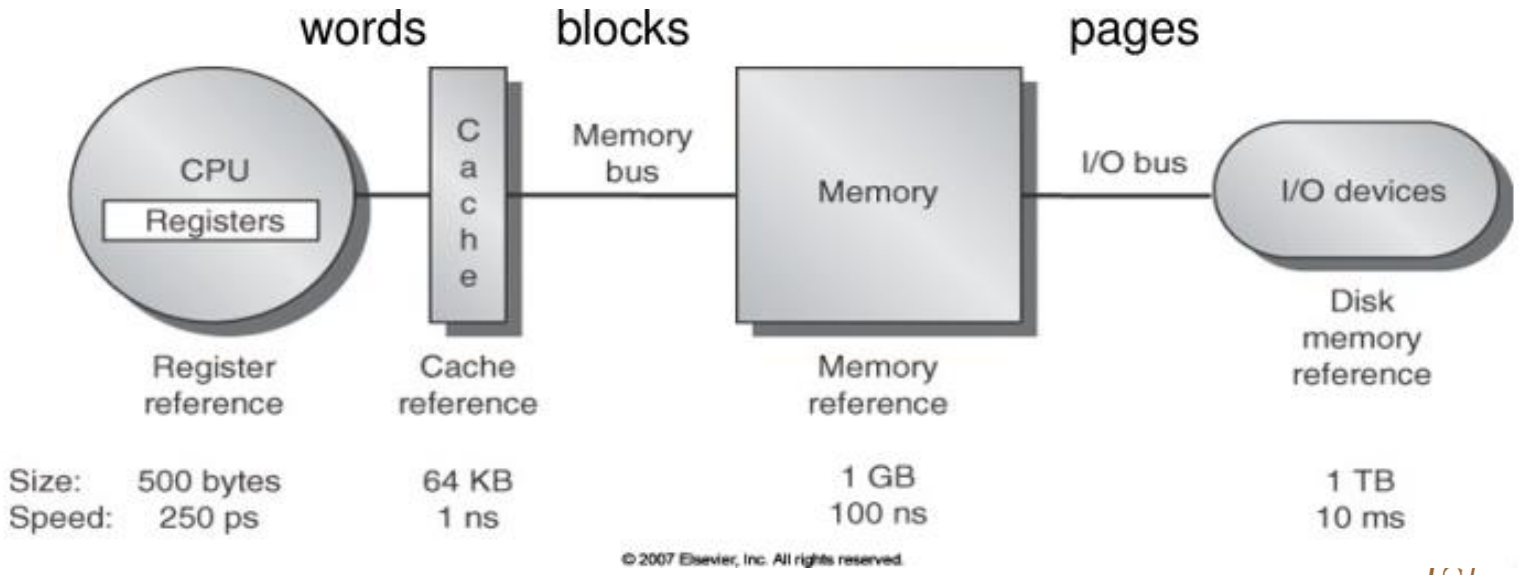
(Image credit: daniiD / Shutterstock)

SK Hynix and Samsung presented their goals for DDR5 memory at the International Solid-State Circuits Conference. Both companies plan to release DDR5 products by the end of 2019, with SK Hynix focusing on desktops and Samsung on mobile devices.

SK Hynix's presentation centered on a 16Gb DDR5 SDRAM module that, according to the company, operates at 1.1V while offering up to 6.4Gb/s of throughput for each pin. The module is said to be made with a 1y-nm process and measures just 76.22 square millimeters. (Some of which the company announced back in November 2018.)

Memory hierarchy functionality

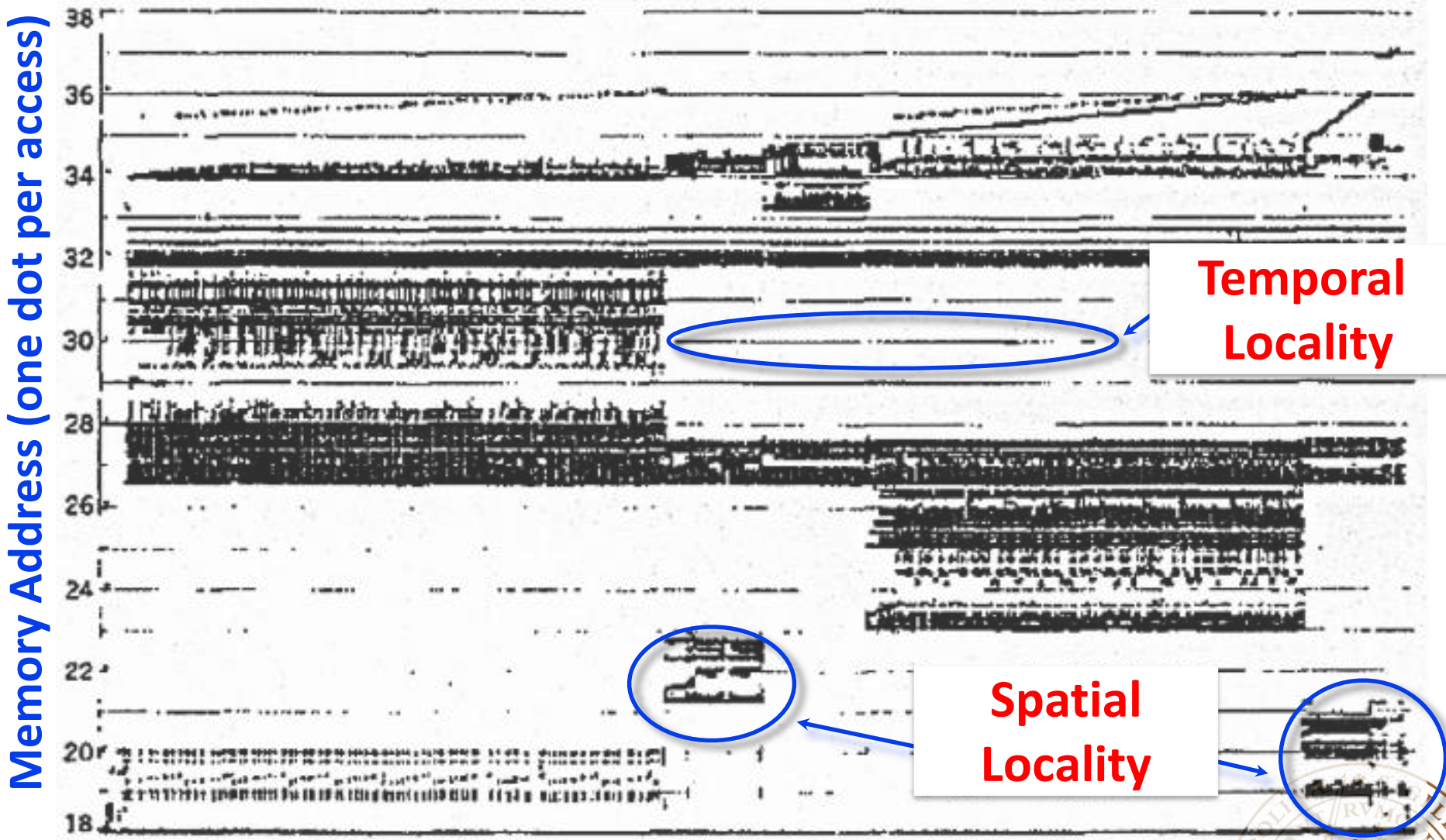
- CPU tries to access memory at address A. If A is in the cache, deliver it directly to the CPU
- If not – transfer a **block of memory words**, containing A, from the memory to the cache. Access A in the cache
- If A not present in the memory – transfer a **page of memory blocks**, containing A, from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



The principle of locality

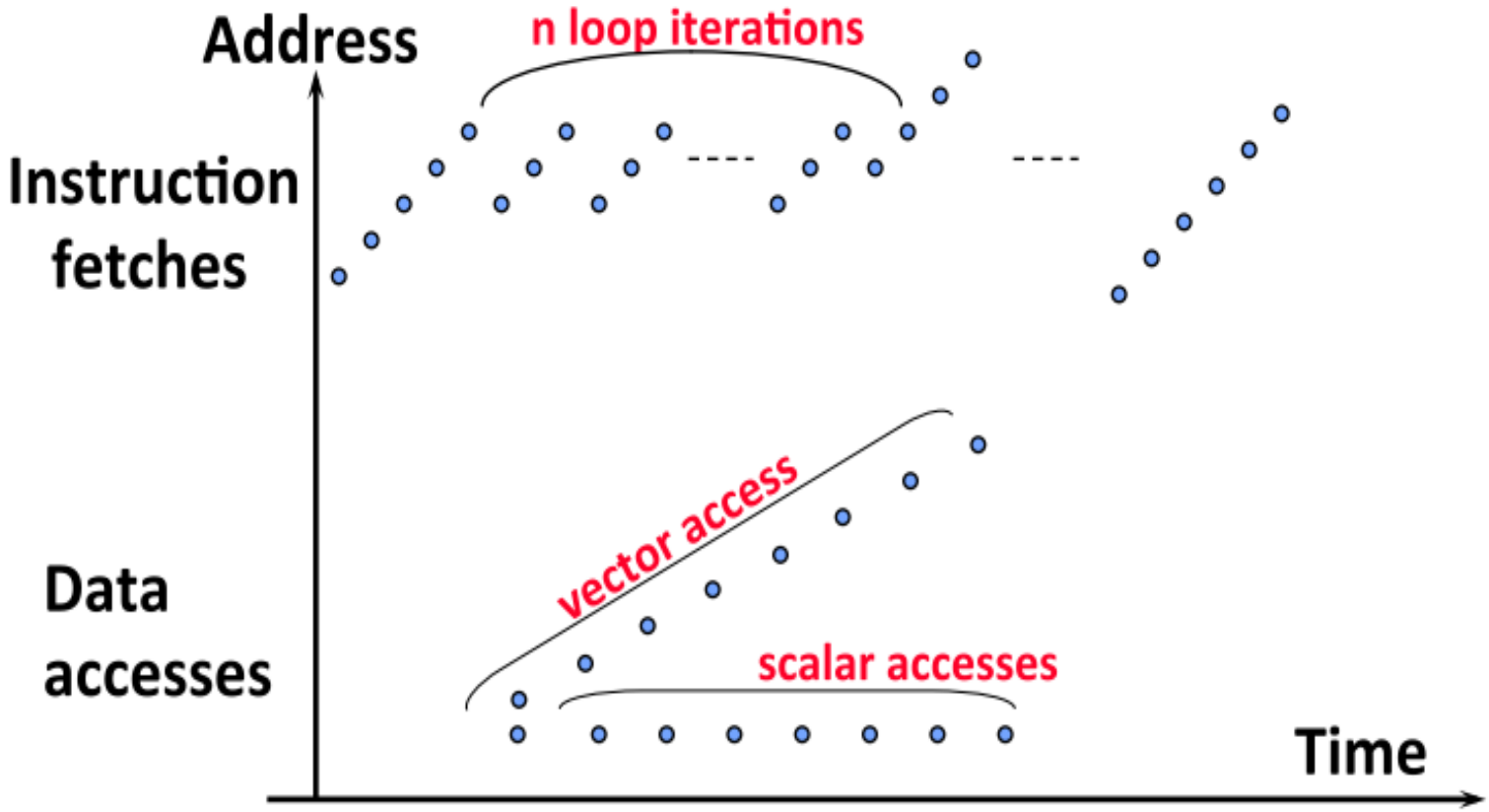
- **A program access a relatively small portion of the address space at any instant of time**
- **Two different types of locality:**
  - **Temporal locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - **Spatial locality** (Locality in space): If an item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

The principle of locality



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

The principle of locality

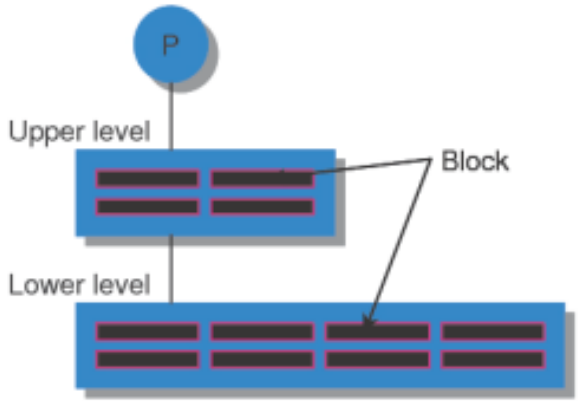


Example

```
sum = 0;
for (i = 0; i<n; i++)
    sum += a[i];
return sum;
```

- Data references
  - Reference array elements in succession (stride-1 reference pattern). — Spatial locality
  - Reference variable `sum` each iteration. — Temporal locality
- Instruction references
  - Reference instructions in sequence. — Spatial locality
  - Cycle through loop repeatedly. — Temporal locality

Cache 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
- Summary

Cache measures

- **hit rate = (# of accesses that hit)/(# of accesses)**
  - Ideal: close to 1
- **miss rate = 1.0 − 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?

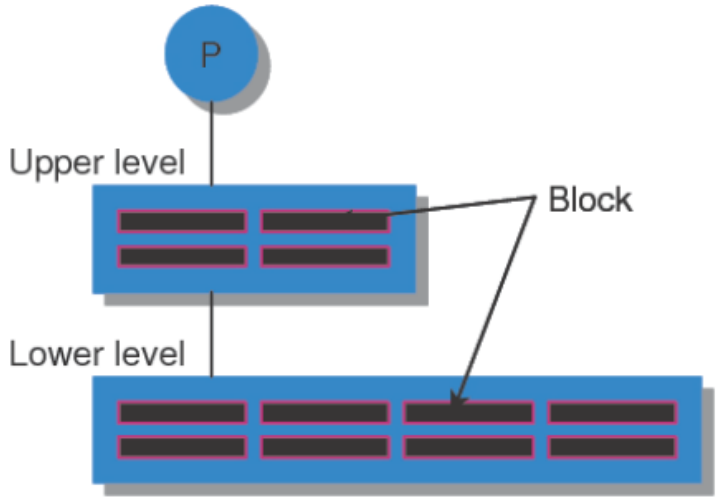
(Block identification)

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

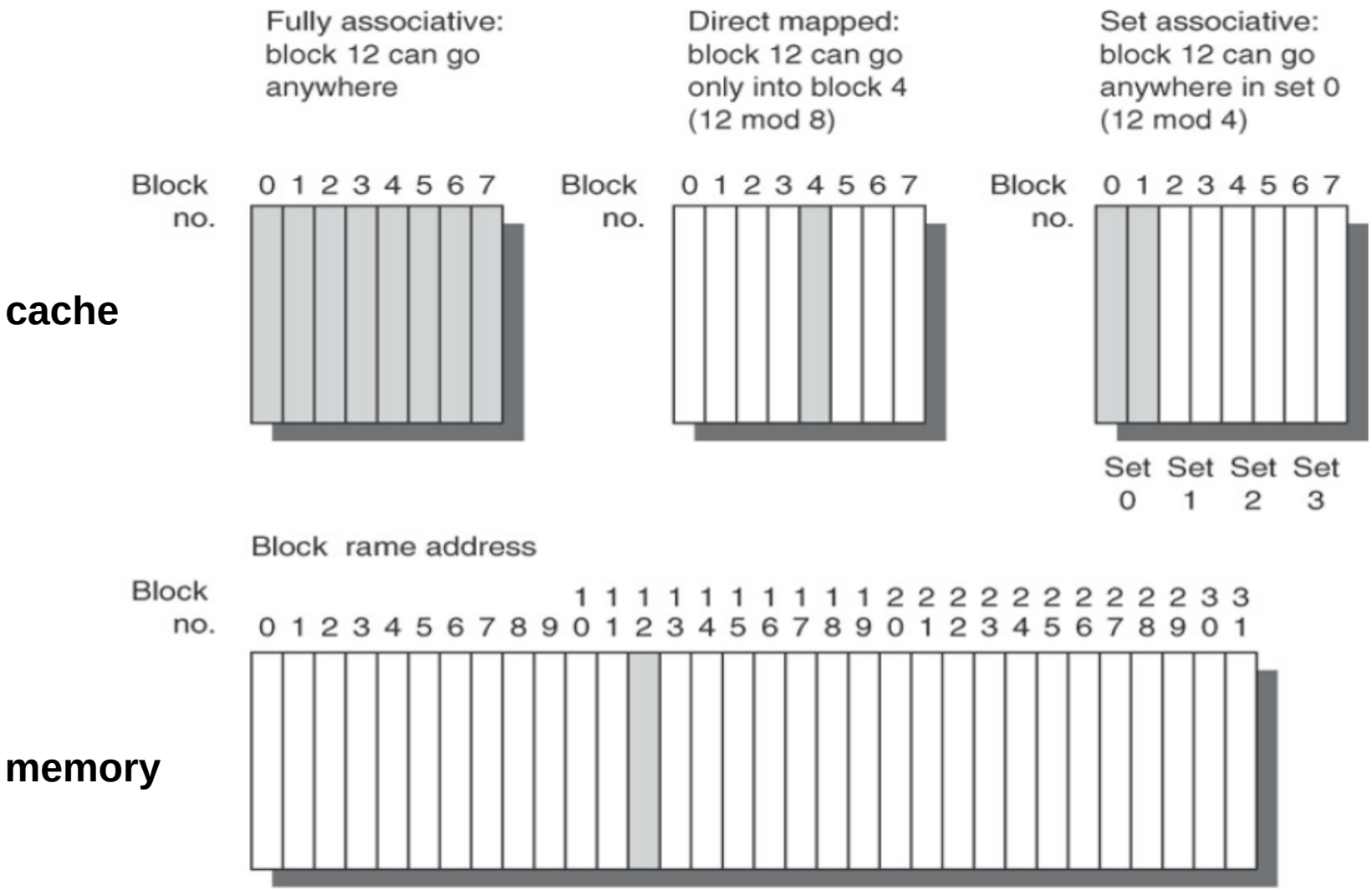
(Block replacement)

- Q4: What happens on a write?

(Write strategy)



Block placement



Block placement

- **Direct Mapped Cache**
  - Each memory location can only be 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!**

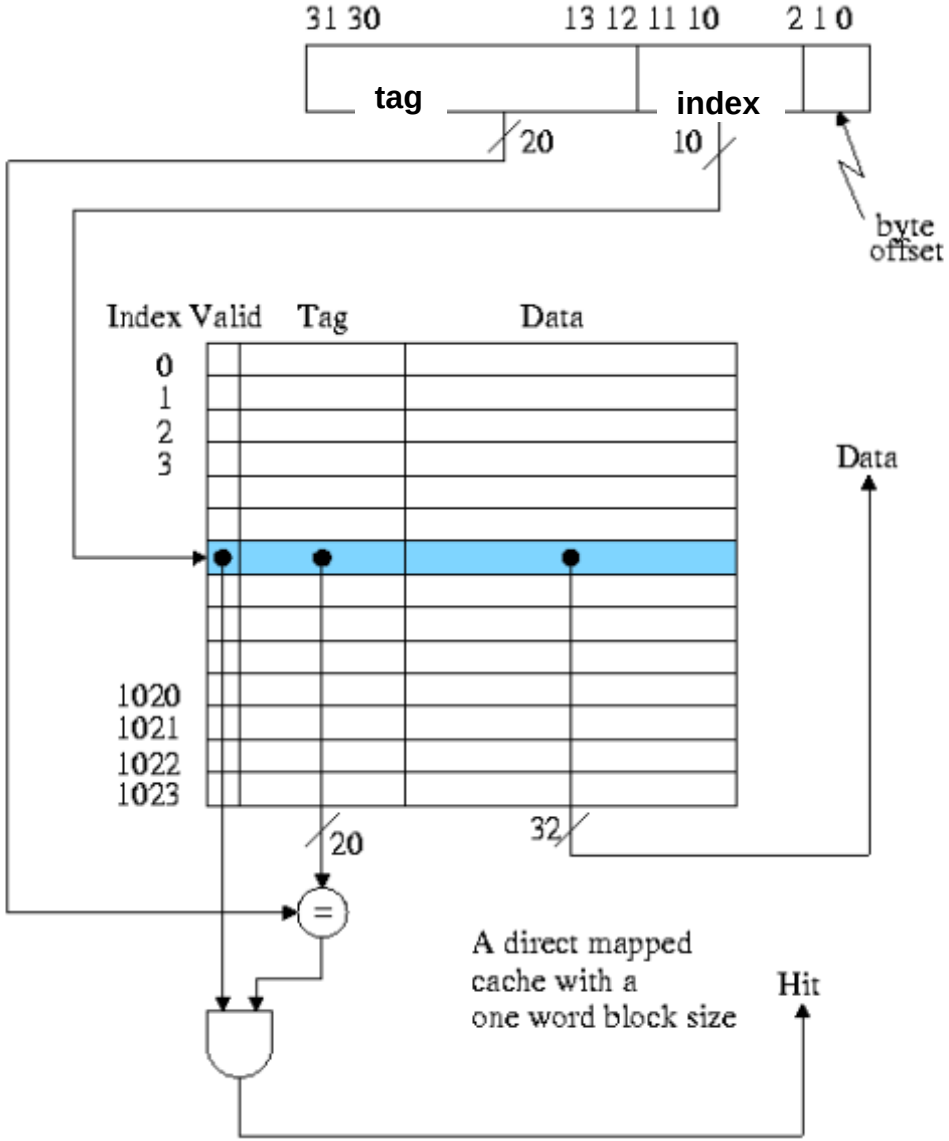
Which block should be replaced on a Cache miss?

- **Direct mapped caches don't need a block replacement policy**
- **Primary strategies:**
  - Random (easiest to implement)
  - LRU – Least Recently Used (best, hard to implement)
  - FIFO – Oldest (used to approximate LRU)

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

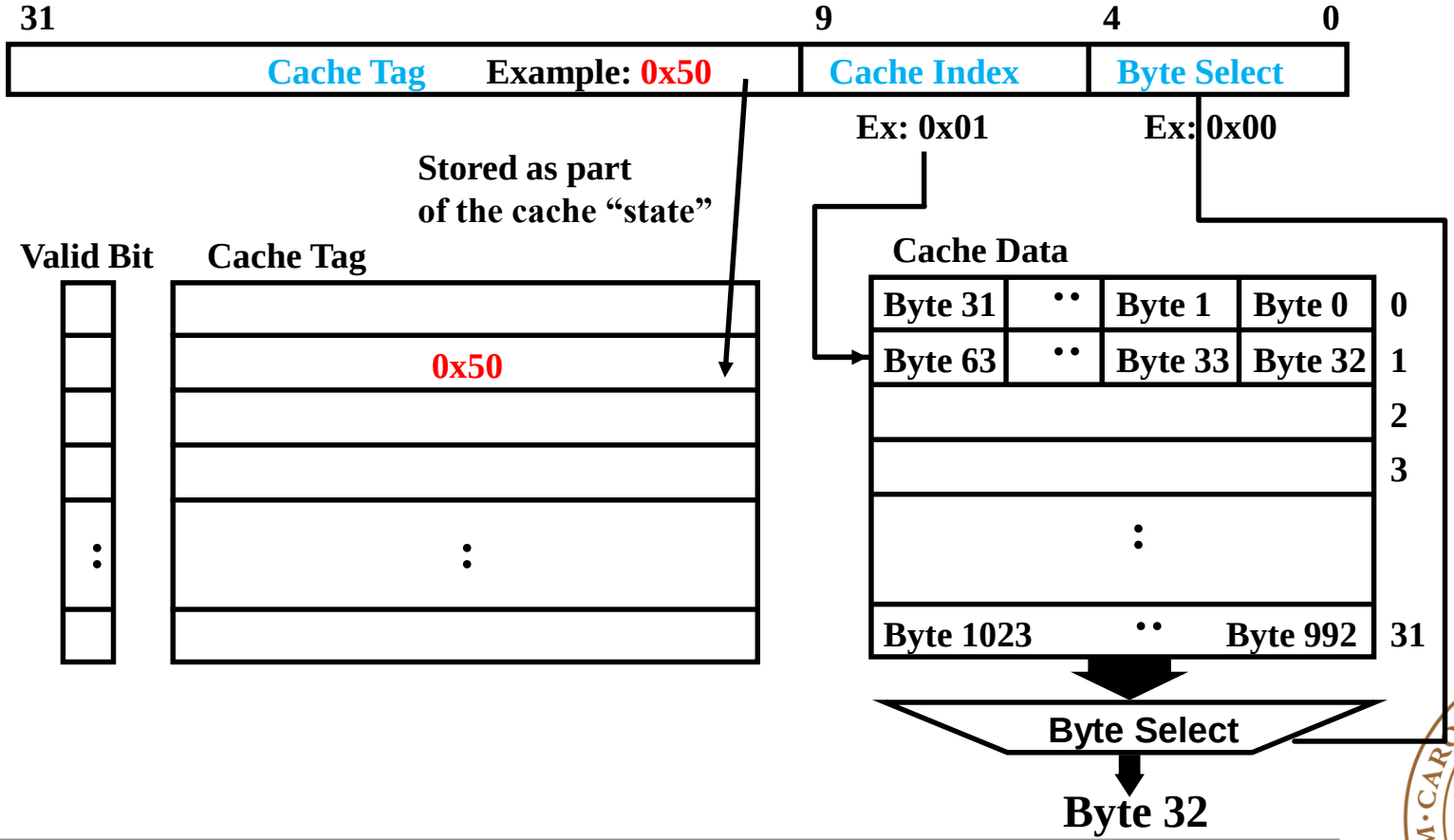
Figure C.4 Data cache misses per 1000 instructions comparing least-recently used, random, and first in, first out replacement for several sizes and associativities. There is little difference between LRU and random for the largest-size cache, with LRU outperforming the others for smaller caches. FIFO generally outperforms random in the smaller cache sizes. These data were collected for a block size of 64 bytes for the Alpha architecture using 10 SPEC2000 benchmarks. Five are from SPECint2000 (gap, gcc, gzip, mcf, and perl) and five are from SPECfp2000 (applu, art, equake, lucas, and swim). We will use this computer and these benchmarks in most figures in this appendix.

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



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.)

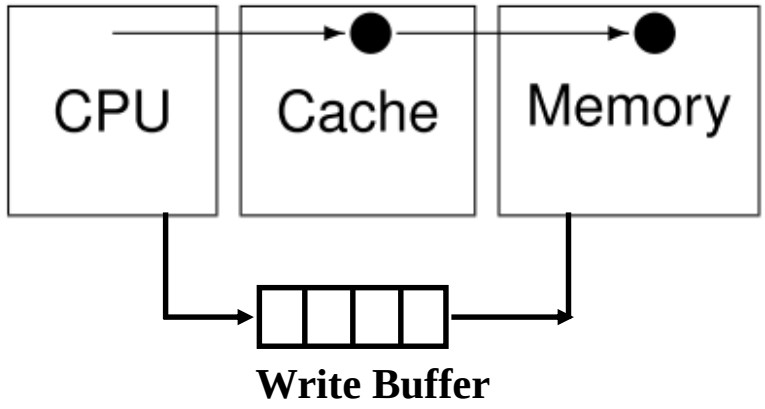
- Send data to the CPU

Reads dominate processor cache accesses and are more critical to processor performance but write is more complicated

Cache write (hit)

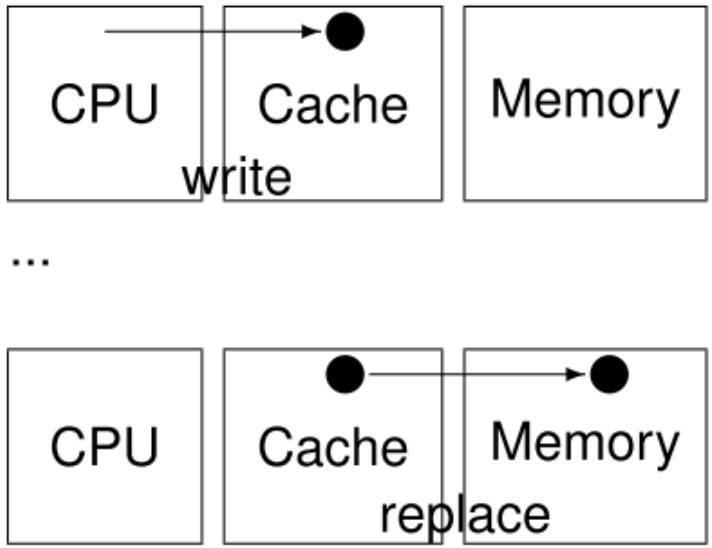
Write through:

- 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:

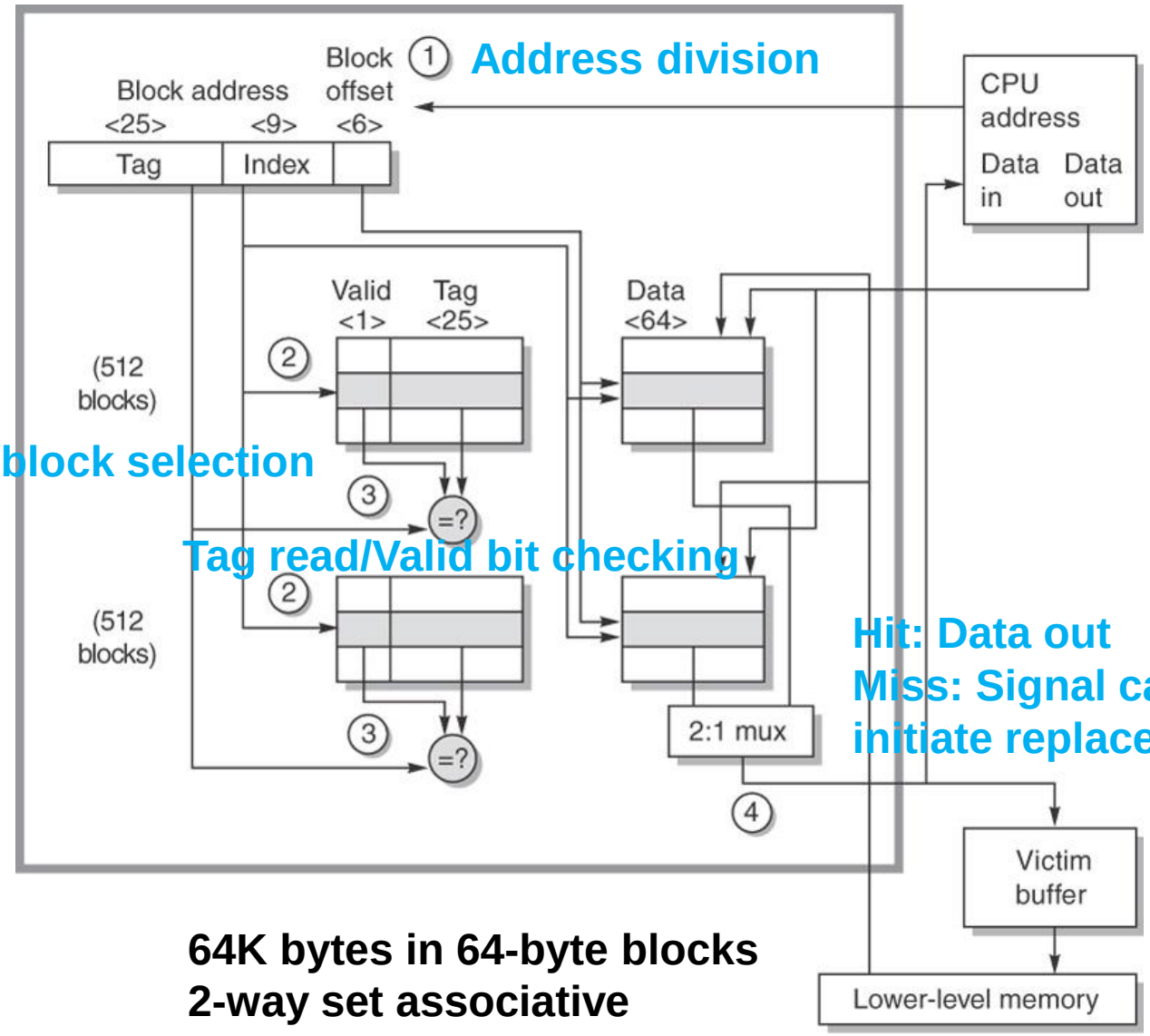
- The information is written only to the block in the cache
- Copy a modified cache block to main memory only when replaced
- Is the block clean or modified? (dirty bit, several write to the same block)



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 (not neccesarily) 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)



Outline

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

Cache performance

$$memory\ stall\ cycles_{cache}$$

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

Interlude – CPU performance equation

$$\text{CPU execution Time} =$$

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

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

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

$$\text{Structural Stalls} + \text{Data Hazard Stalls} + \text{Controll Stalls} +$$

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

$$) * T_C$$

Cache performance

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

- **Three ways to increase performance:**
  - Reduce miss rate
  - Reduce miss penalty
  - Reduce hit time (improves T_C)

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

Cache performance, example

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

Example:

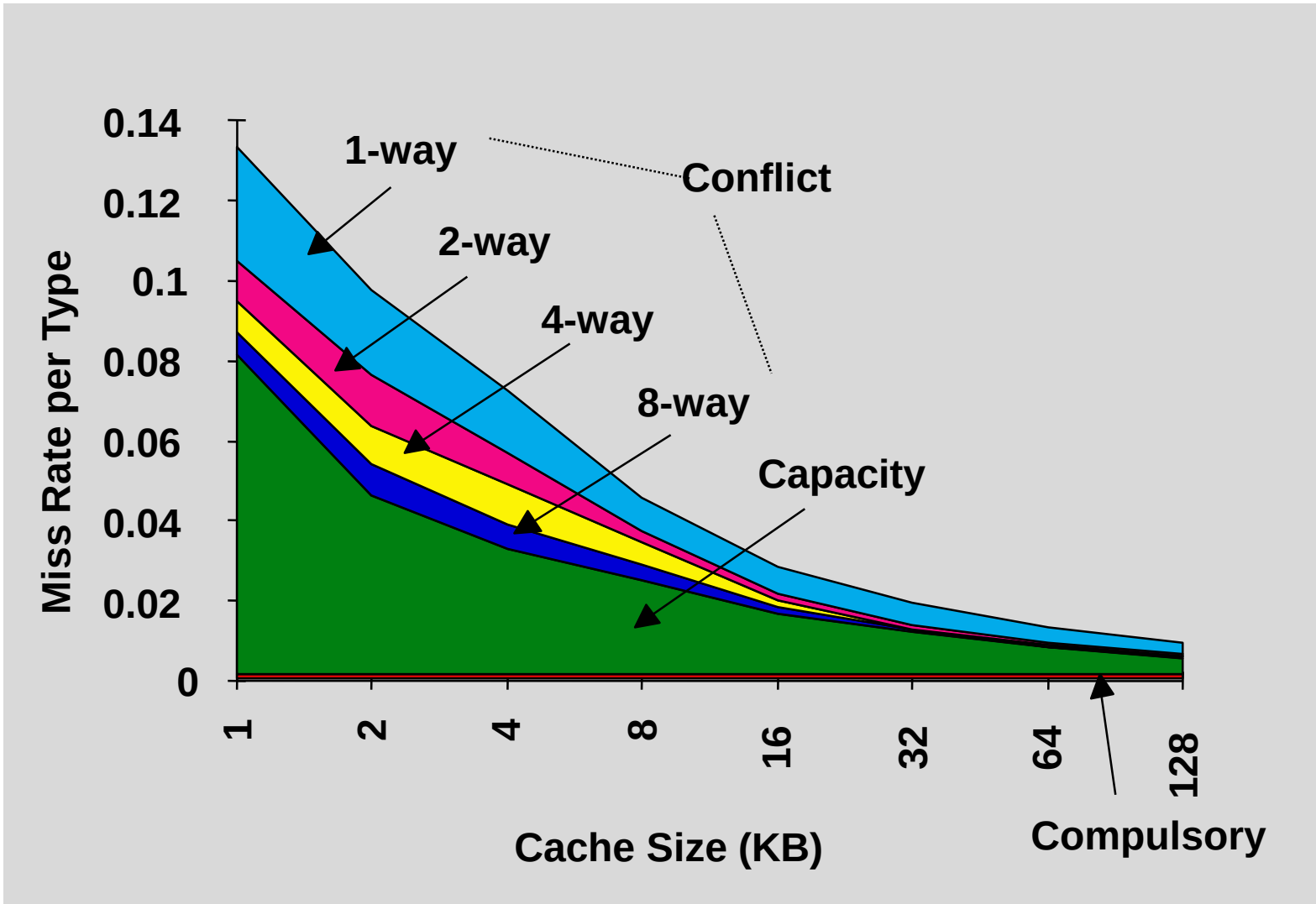
miss rate (%)	1
miss penalty (cycles)	50
$\frac{mem\ accesses}{instruction}$	k
CPI increase	$k * 0.01 * 50$

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 to 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 components – 3 C's

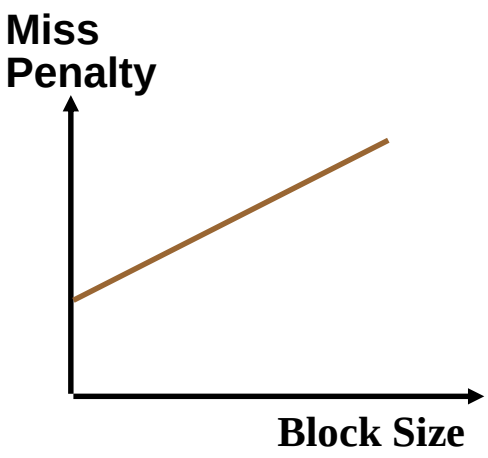
- Small percentage of compulsory misses
- Capacity misses are reduced by larger caches
- Full associativity avoids all conflict misses
- Conflict misses are relatively more important for small set-associative caches

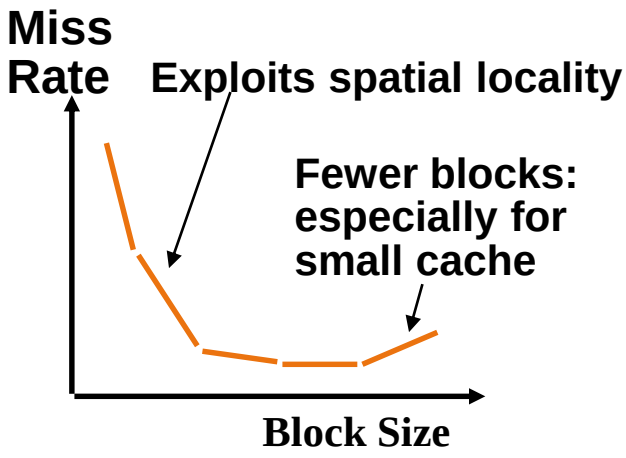
Miss may move from one to another!

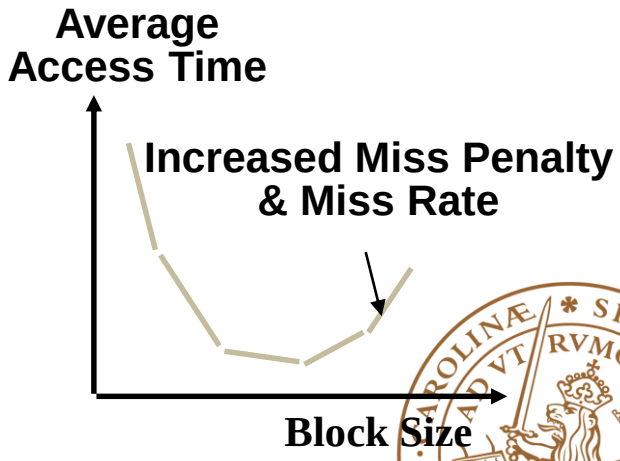
Block size tradeoff

- **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

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





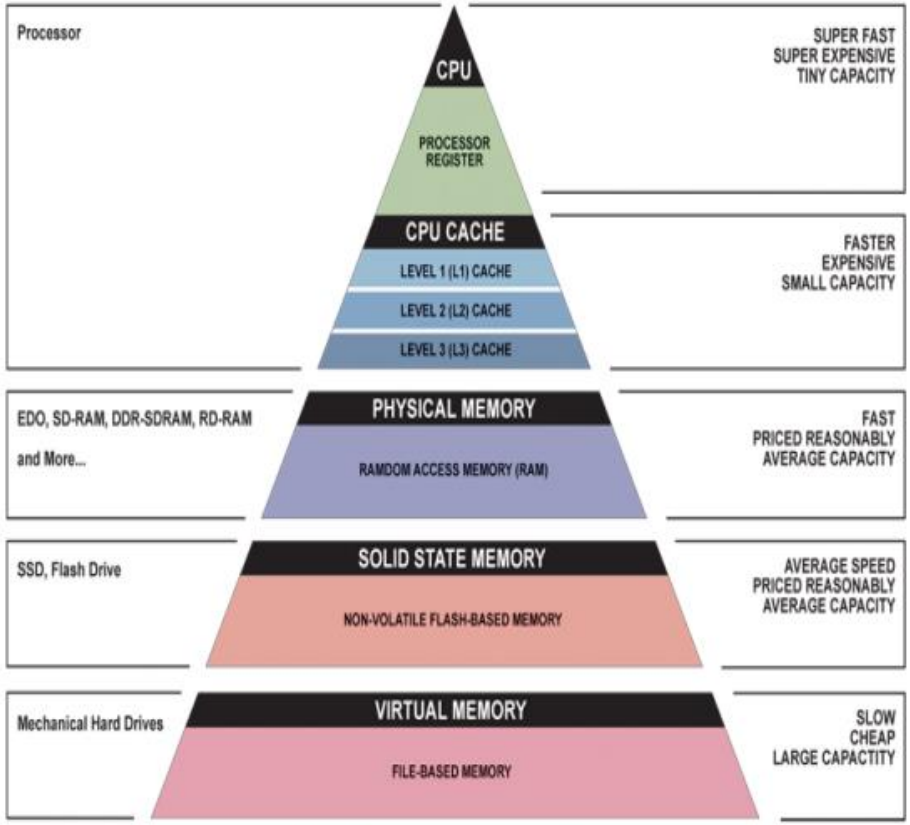


Outline

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

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**

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

- **Three types of misses:**
  - Compulsory
  - Capacity
  - Conflict
- **Main memory performance:**
  - Latency
  - Bandwidth