

Locality & Caching

Comp-sys Recap

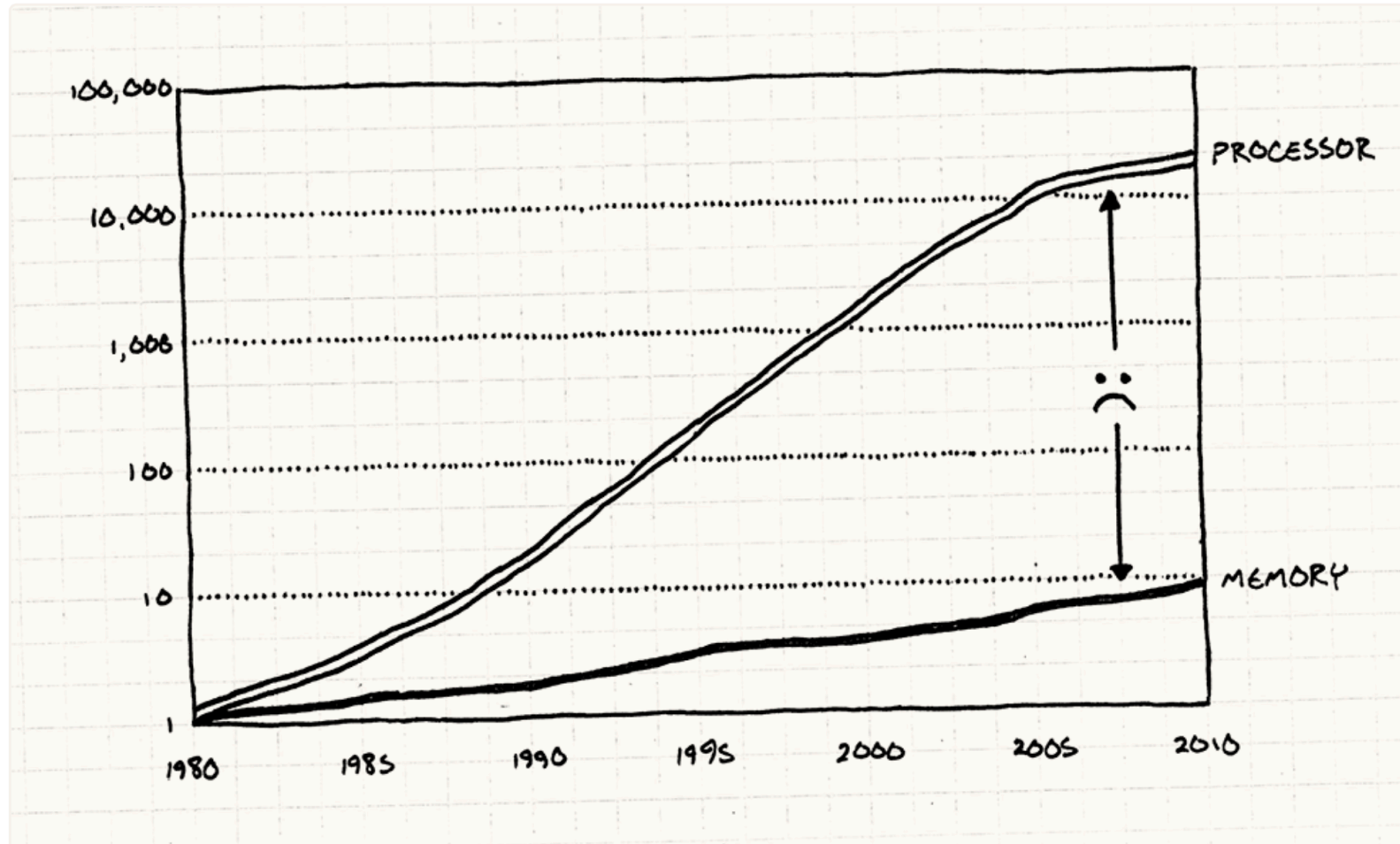
Walter Restelli-Nielsen



Outline

- Motivation
- Locality
- Caching
- Examples


Memory Gap





Memory Gap – size overview

- CPU cycle: Seconds
- RAM access: Minutes
- SSD access: Days
- Disk access: Years

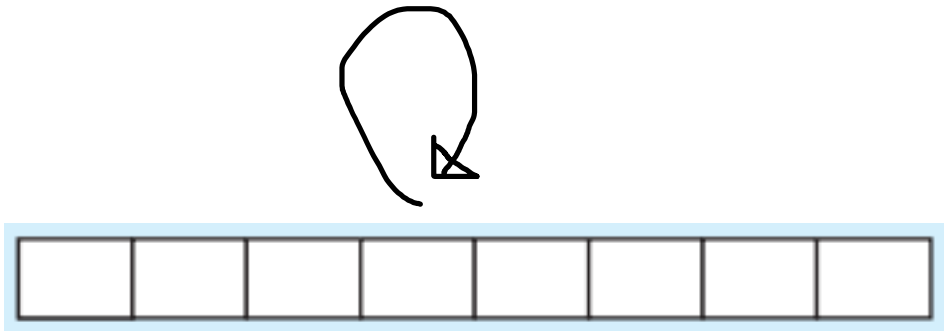


Memory Gap – Solutions?

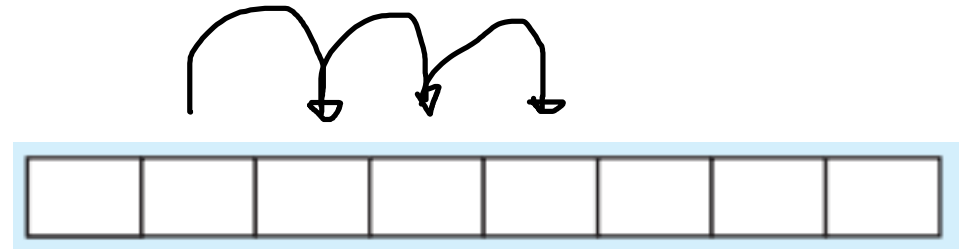
- Run multiple processes at once
- Caching
 - Reuse the same data
 - Access data close by

Locality

Temporal locality



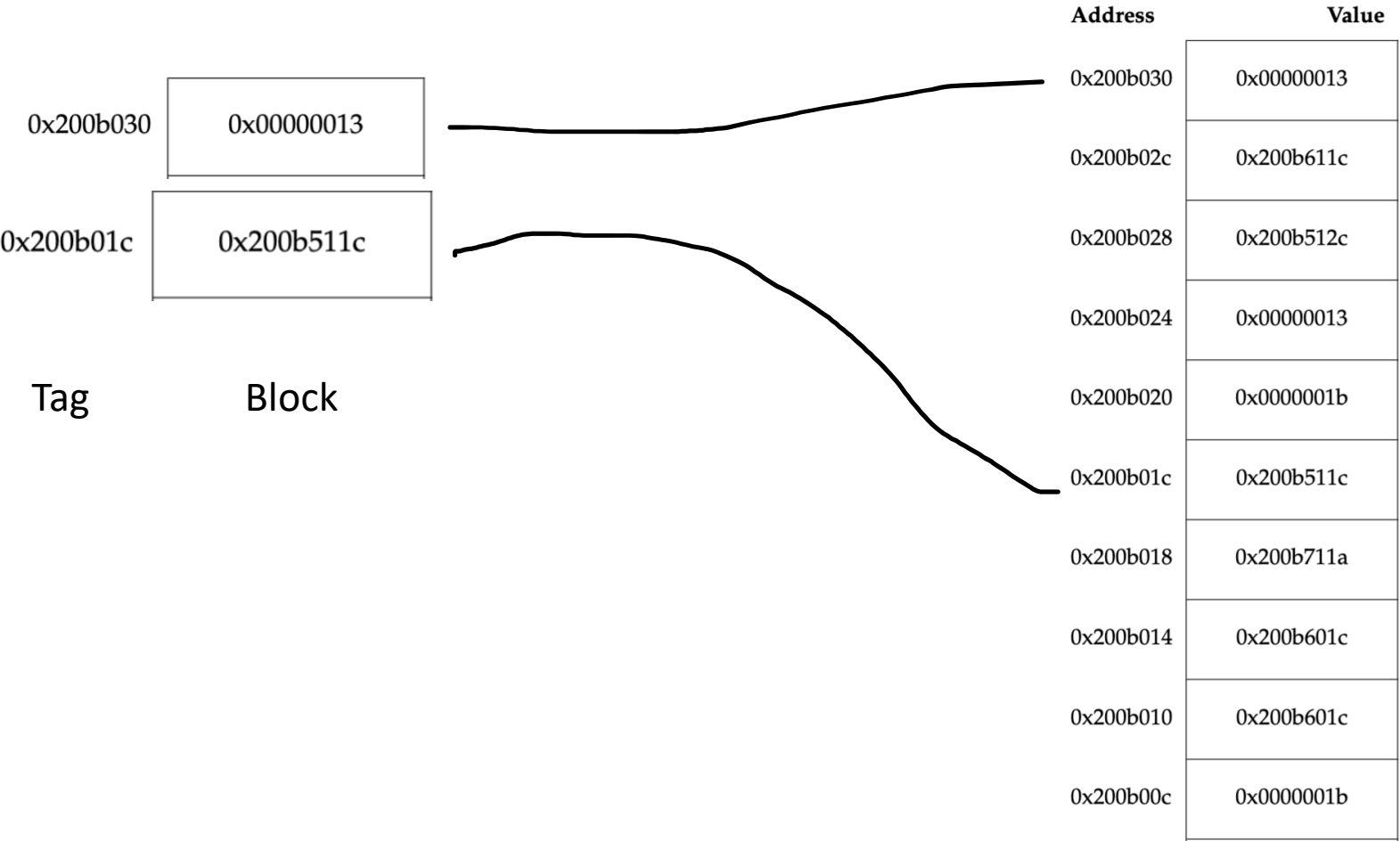
Spatial locality



CPU

Cache

Main memory (RAM)

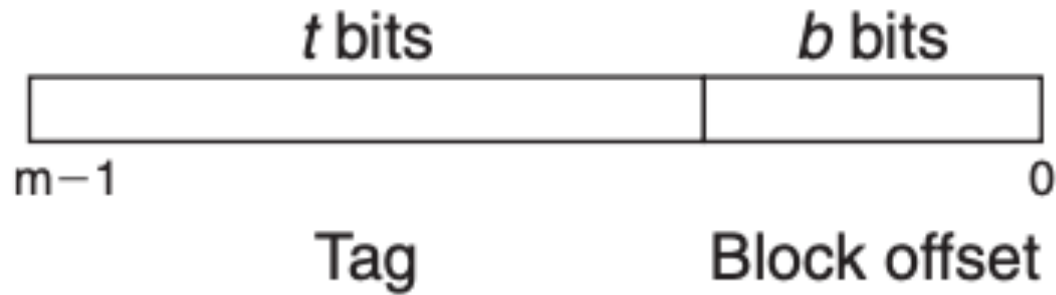


Basic cache idea – full associative

m : address size

B : size of each line

E : entries per set



$$b = \log_2(B)$$

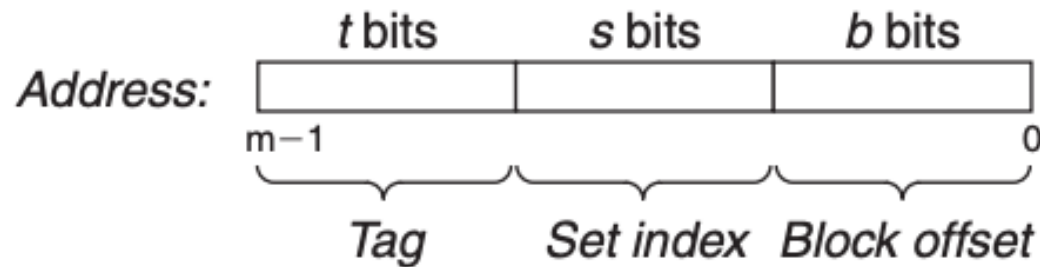
$$t = m - b$$

S sets, direct mapped cache

m : address size

B : Size of each line

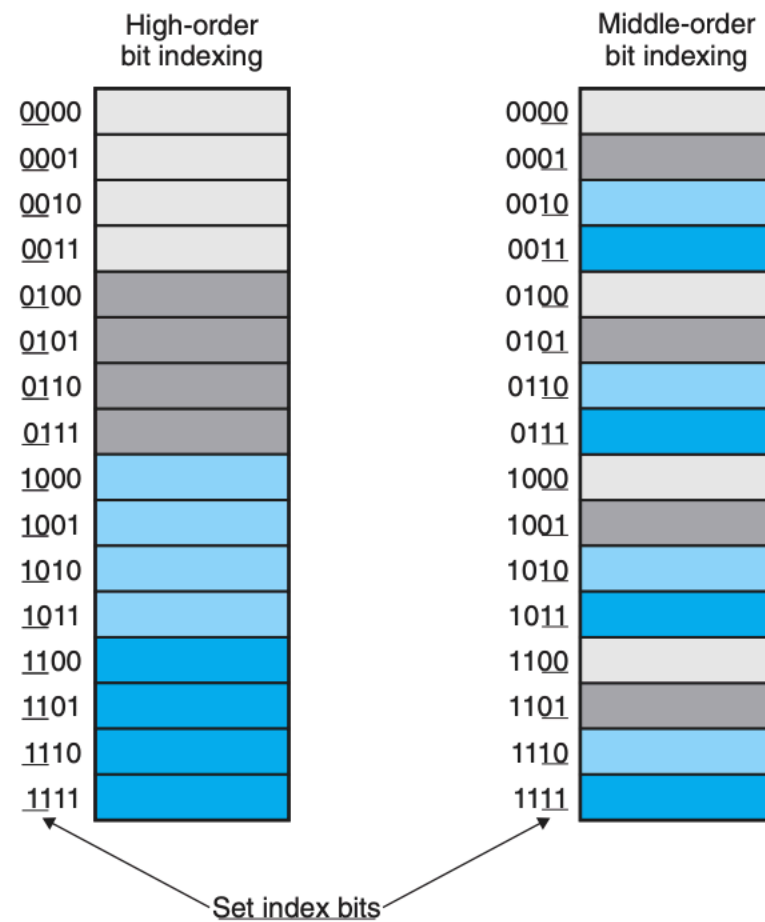
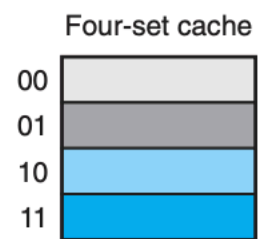
S : Number of sets



$$b = \log_2(B)$$

$$s = \log_2(S)$$

$$t = m - s - b$$



Final version: set associative cache

m : address size

B : Size of each line

S : Number of sets

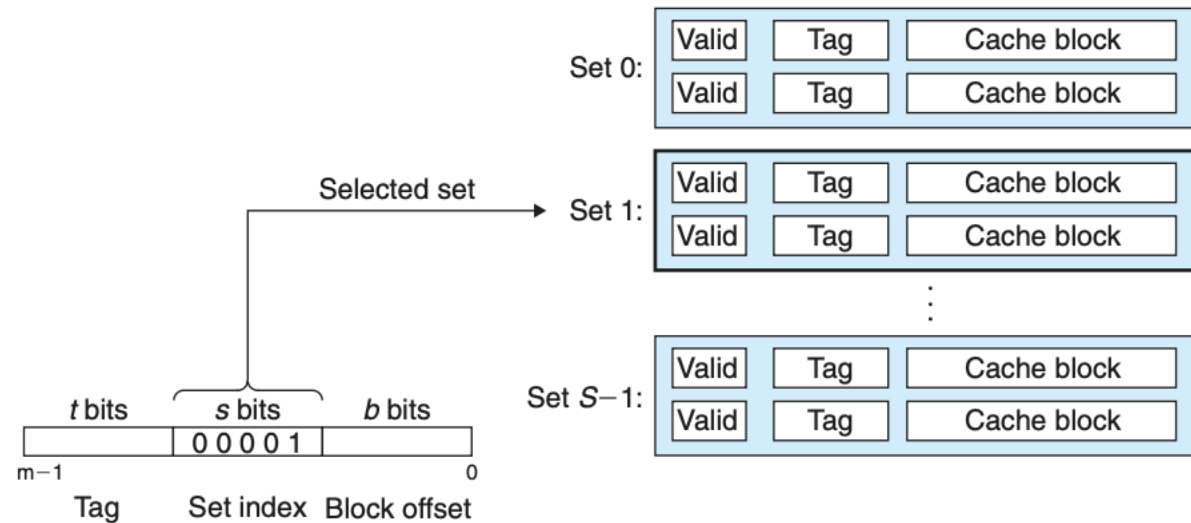
E : entries per set

C : Size = $S * E * B$

$$b = \log_2(B)$$

$$s = \log_2(S)$$

$$t = m - s - b$$



extra

- Cache hit/miss – (conflict miss, capacity miss, cold miss / thrashing)
- Write back, Dirty bits (compared to write through)
- Replacement policies (Least recently used)

Example 1/3

2-way set associative cache												
Set index	Line 0						Line 1					
	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10	00	0	—	—	—	—
1	45	1	60	4F	E0	23	38	1	00	BC	0B	37
2	EB	0	—	—	—	—	0B	0	—	—	—	—
3	06	0	—	—	—	—	32	1	12	08	7B	AD
4	C7	1	06	78	07	C5	05	1	40	67	C2	3B
5	71	1	0B	DE	18	4B	6E	0	—	—	—	—
6	91	1	A0	B7	26	2D	F0	0	—	—	—	—
7	46	0	—	—	—	—	DE	1	12	C0	88	37

address: 0xE34

Example

2-way set associative cache

Set index	Line 0						Line 1					
	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10	00	0	—	—	—	—
1	45	1	60	4F	E0	23	38	1	00	BC	0B	37
2	EB	0	—	—	—	—	0B	0	—	—	—	—
3	06	0	—	—	—	—	32	1	12	08	7B	AD
4	C7	1	06	78	07	C5	05	1	40	67	C2	3B
5	71	1	0B	DE	18	4B	6E	0	—	—	—	—
6	91	1	A0	B7	26	2D	F0	0	—	—	—	—
7	46	0	—	—	—	—	DE	1	12	C0	88	37

12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	1	0	1	0	0
CT	CT	CT	CT	CT	CT	CT	CT	CI	CI	CI	CO	CO

2-way set associative cache

Set index	Line 0						Line 1					
	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10	00	0	—	—	—	—
1	45	1	60	4F	E0	23	38	1	00	BC	0B	37
2	EB	0	—	—	—	—	0B	0	—	—	—	—
3	06	0	—	—	—	—	32	1	12	08	7B	AD
4	C7	1	06	78	07	C5	05	1	40	67	C2	3B
5	71	1	0B	DE	18	4B	6E	0	—	—	—	—
6	91	1	A0	B7	26	2D	F0	0	—	—	—	—
7	46	0	—	—	—	—	DE	1	12	C0	88	37

12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	1	0	1	0	0
CT	CT	CT	CT	CT	CT	CT	CT	CI	CI	CI	CO	CO

Parameter	Value
Cache block offset (CO)	0x0
Cache set index (CI)	0x5
Cache tag (CT)	0x71
Cache hit? (Y/N)	Y
Cache byte returned	0xB

Example 2

For this and the following questions, assume that for a given program of interest, the miss rate for the primary cache is 5% and the miss rate for the secondary cache is 20%. A run of the program comprises a total of 1,000,000,000 instructions. Of these instructions 40% are instructions referencing memory.

Caching and the memory hierarchy, 1.3.4: How many cache lookups are performed against the secondary cache? Explain your answer.

Example 3

```
1 void arr_update(long arr[], int n) {  
2     for (int i = 0; i < n; i++) {  
3         for (int j = i+1; j < n; j++) {  
4             arr[j] += 3 * arr[i];  
5         }  
6     }  
7 }
```

In the execution of the program the local variables (i and j) and the integral input parameter allocated in registers. The array (arr) is located in memory.

Note: Following Questions 1.5.1 and 1.5.2 are completely independent.

Data Cache and Program Locality, 1.5.1: Given a byte-addressed machine with 32-bit addresses equipped with a fully associative L1 data-cache. It has **4 cache lines** each with a **block size of 16**. It functions as write-allocate and updates with LRU-replacement. We also know that `sizeof(int)` and `sizeof(long)` == 8.

Assume that we are executing arr_update under the following conditions:

- the arr array starts at address 0x00008000,
- the size of array is 12 (n == 12), and
- the cache is assumed to be cold on entry.

For each of the first 8 iterations (i == 0 to i == 7) indicate whether each access results in a cache hit (h), a miss (m), or if it is unused (keep it empty). For example, in first iteration (i == 0) reading is a miss as the cache is cold. Explain the considerations behind your answer below the table.

	arr indices											
	0	1	2	3	4	5	6	7	8	9	10	11
Iteration 1 (i = 0)	m	h	m	h	m	h	m	h	m	h	m	h
Iteration 2 (i = 1)		h	m	h	m	h	m	h	m	h	m	h
Iteration 3 (i = 2)			m	h	m	h	m	h	m	h	m	h
Iteration 4 (i = 3)				h	m	h	m	h	m	h	m	h
Iteration 5 (i = 4)					m	h	m	h	m	h	m	h
Iteration 6 (i = 5)						h	h	h	h	h	h	h
Iteration 7 (i = 6)							h	h	h	h	h	h
Iteration 8 (i = 7)								h	h	h	h	h