Fiscal Year 2018

Ver. 2018-12-08a

Course number: CSC.T433
School of Computing,
Graduate major in Computer Science

Advanced Computer Architecture

3. Memory Hierarchy Design

www.arch.cs.titech.ac.jp/lecture/ACA/
Room No.W936
Mon 13:20-14:50, Thr 13:20-14:50

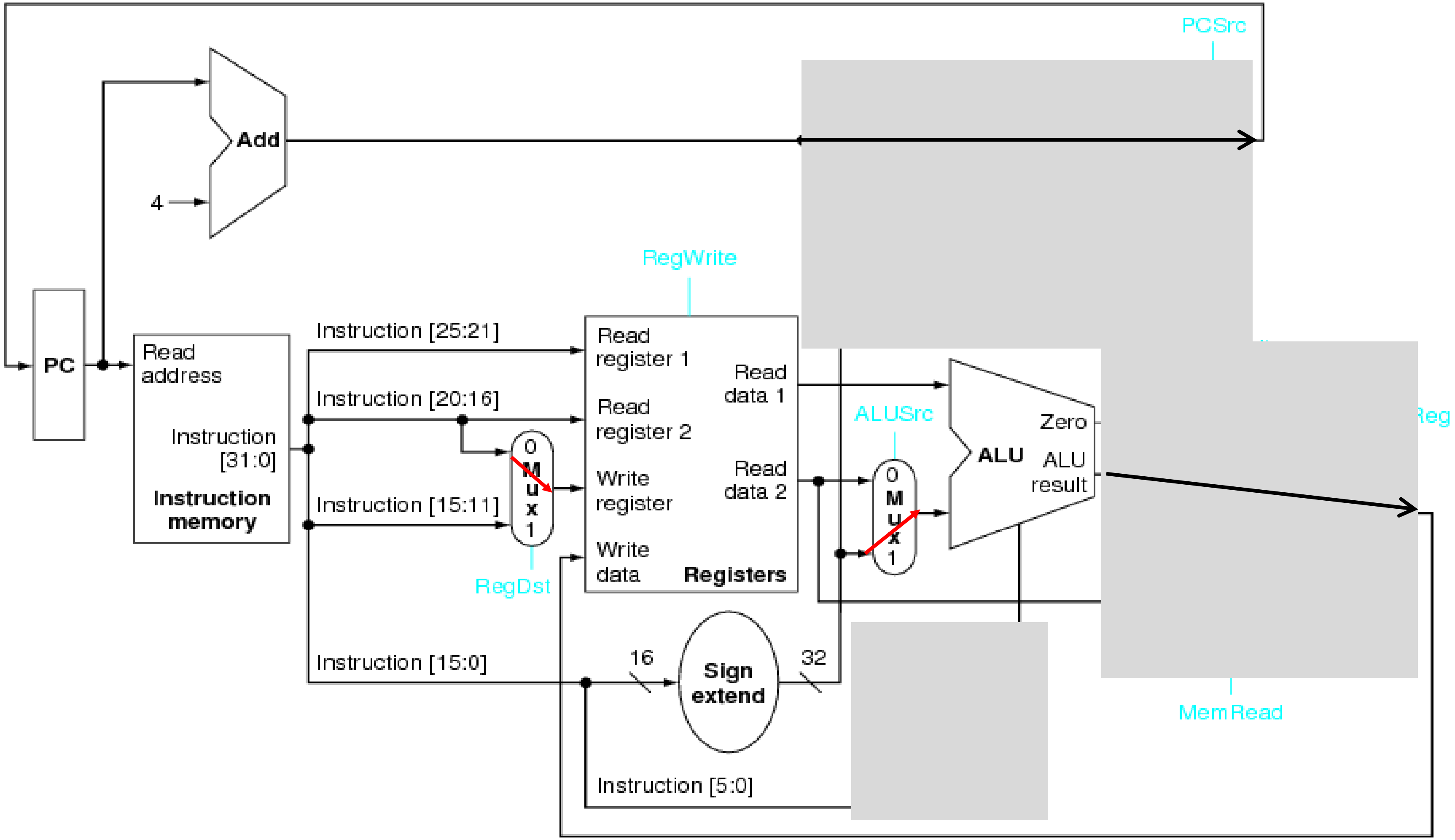
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Datapath of processor supporting ADD and ADDI

	IR[25:21]	IR[20:16]	
op	rs	rt	16 bit immediate

I format

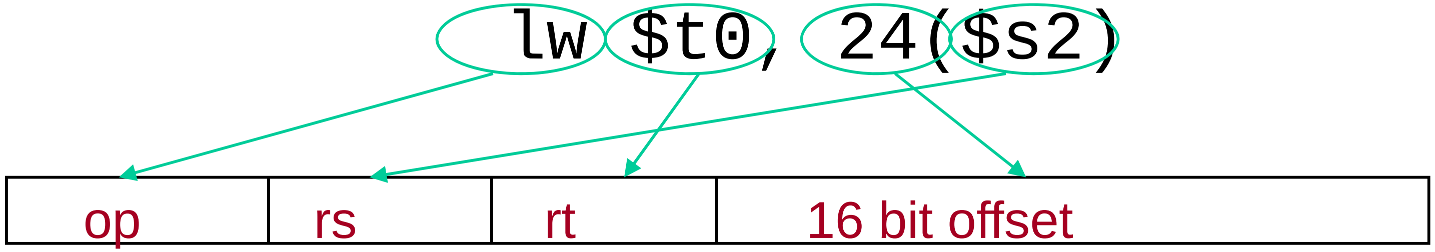
```
0x804   addi $t1, $t0, 3       [ addi $9, $8, 3 ]
```



$8 = 7

Machine Language - Load Instruction

- Load/Store Instruction Format (I format):



lw $t0, 24($s2)

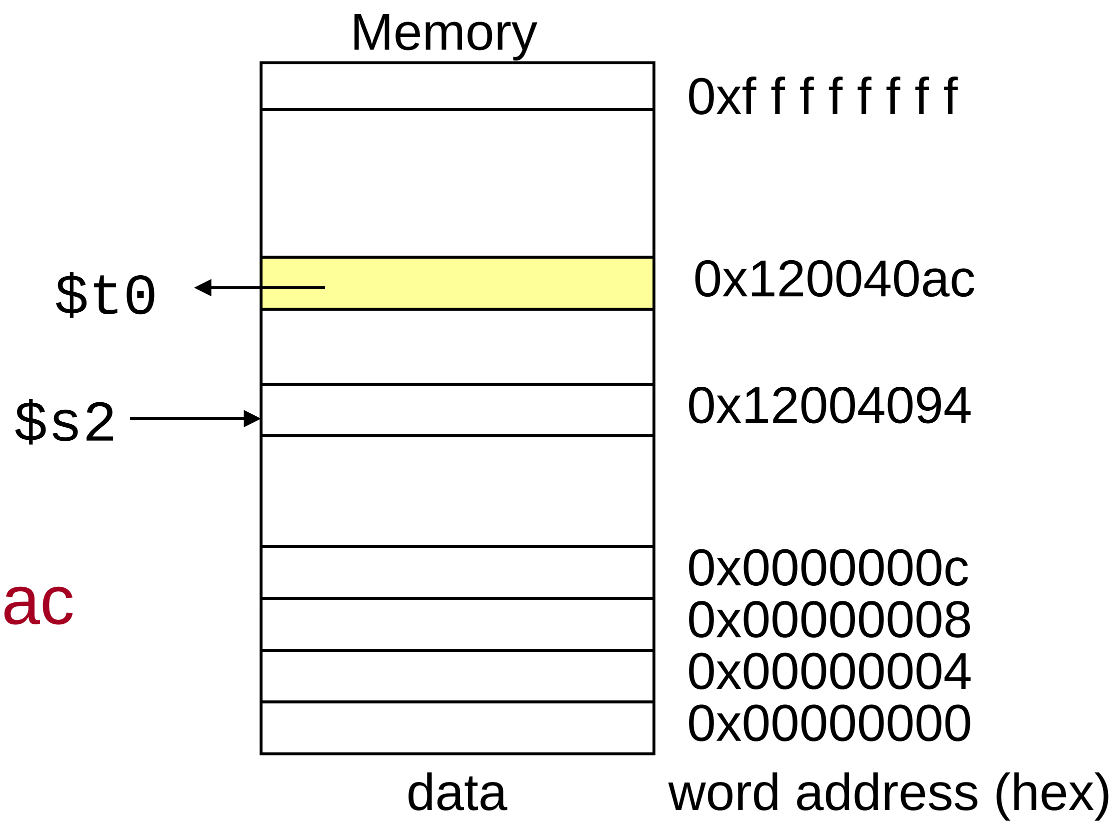
op	rs	rt	16 bit offset

24_{10} + $s2 =

. . . 0001 1000

+ . . . 1001 0100

. . . 1010 1100 = 0x120040ac



Memory

$t0

$s2

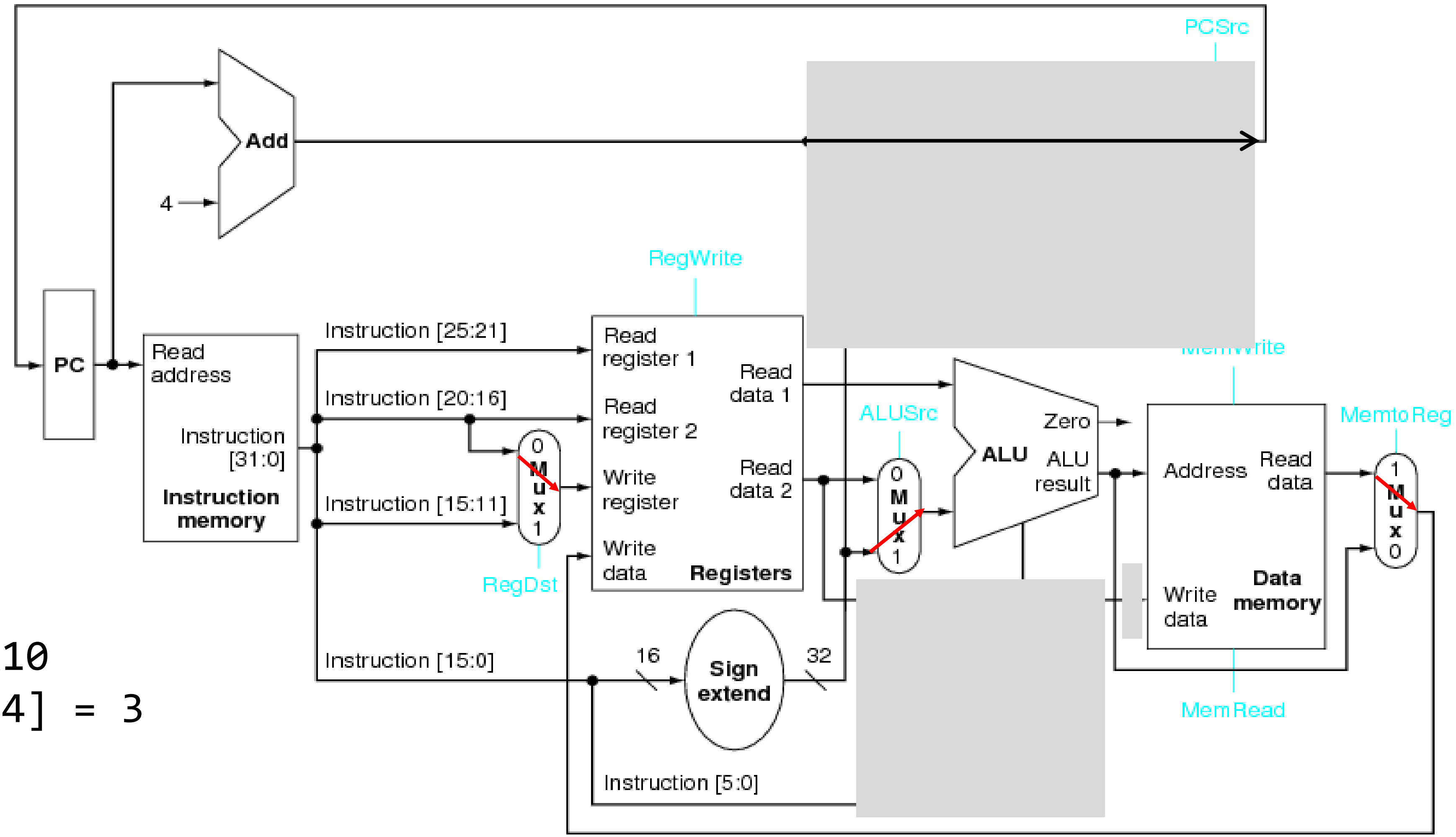
data	word address (hex)
	0xf f f f f f f f
	0x120040ac
	0x12004094
	0x0000000c
	0x00000008
	0x00000004
	0x00000000

Datapath of processor supporting ADD, ADDI, LW

	IR[25:21]	IR[20:16]	
op	rs	rt	16 bit immediate

I format

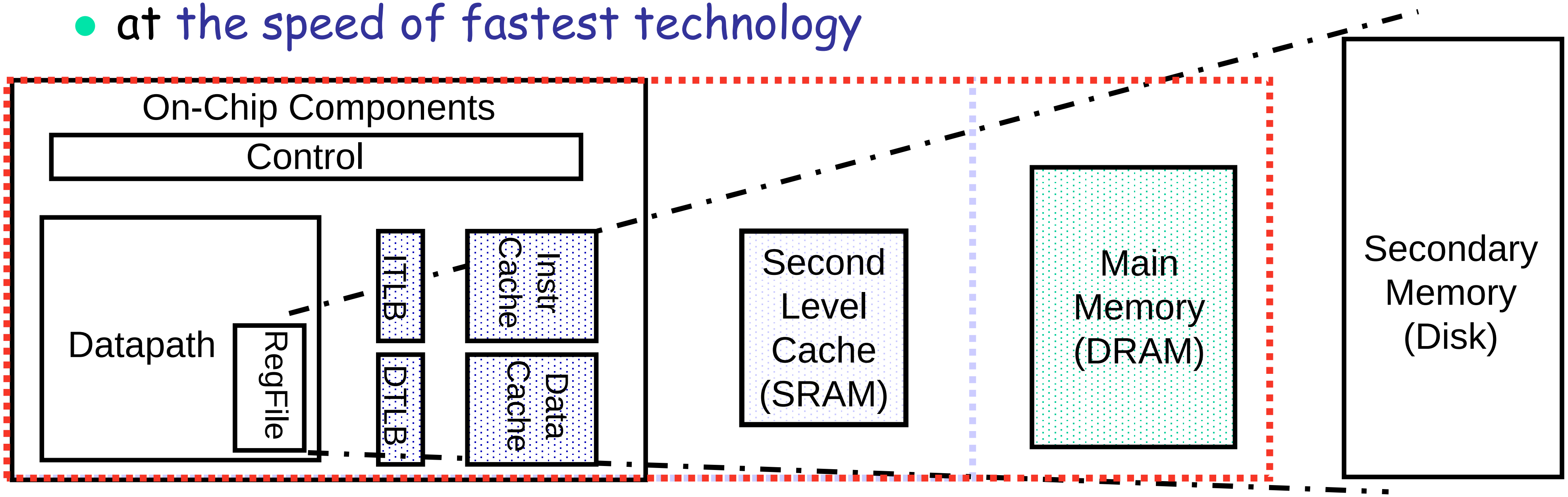
```
0x808   lw $t2, 4($t0)        [ lw $10, 4($8) ]
```



```
$8 = 0x10
mem[0x14] = 3
```

A Typical Memory Hierarchy

- By taking advantage of the principle of locality
  - Present much memory in the cheapest technology
  - at the speed of fastest technology

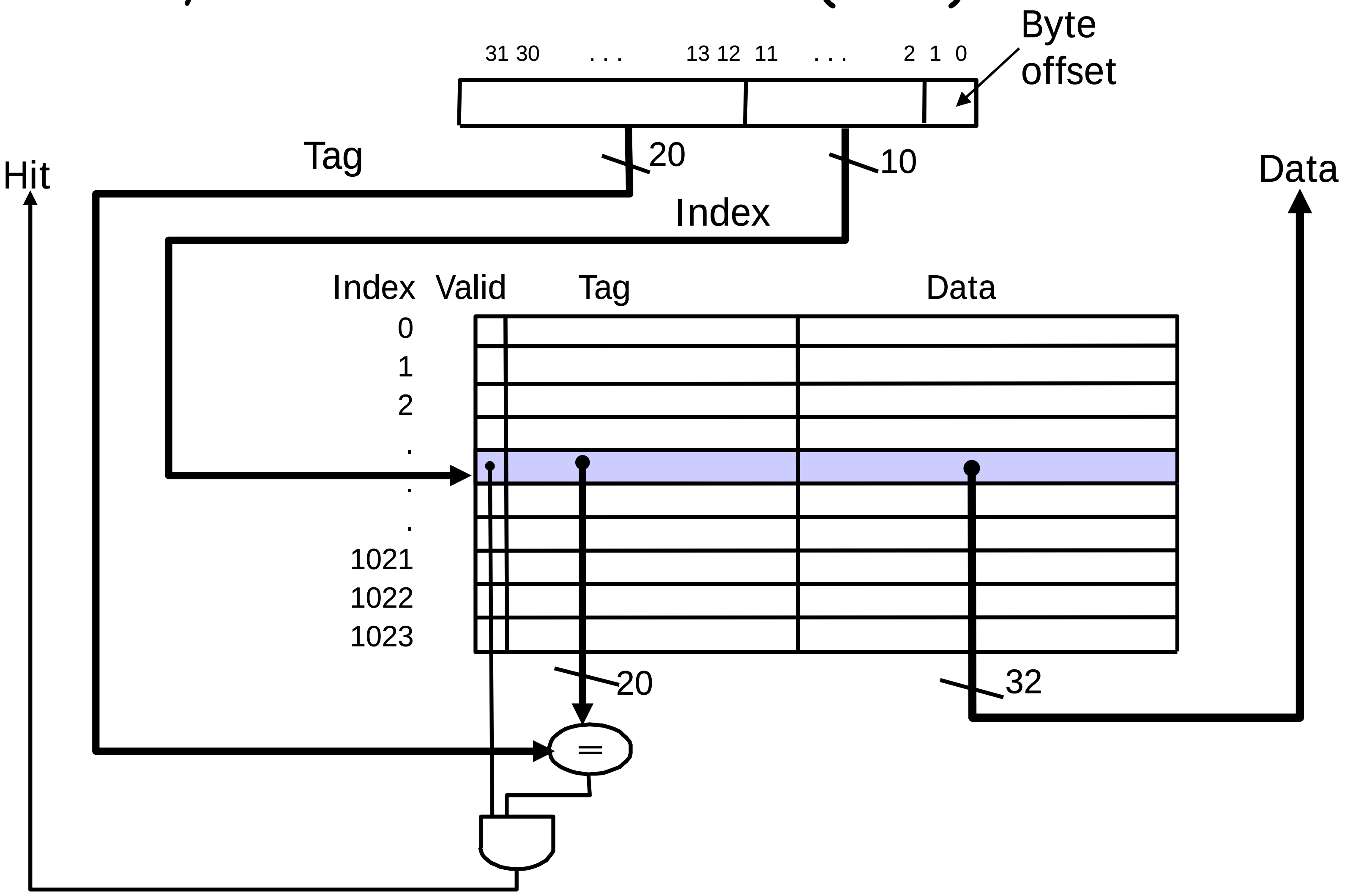


Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	10K's	M's	G's to T's
Cost:	highest				lowest

TLB: Translation Lookaside Buffer

MIPS Direct Mapped Cache Example

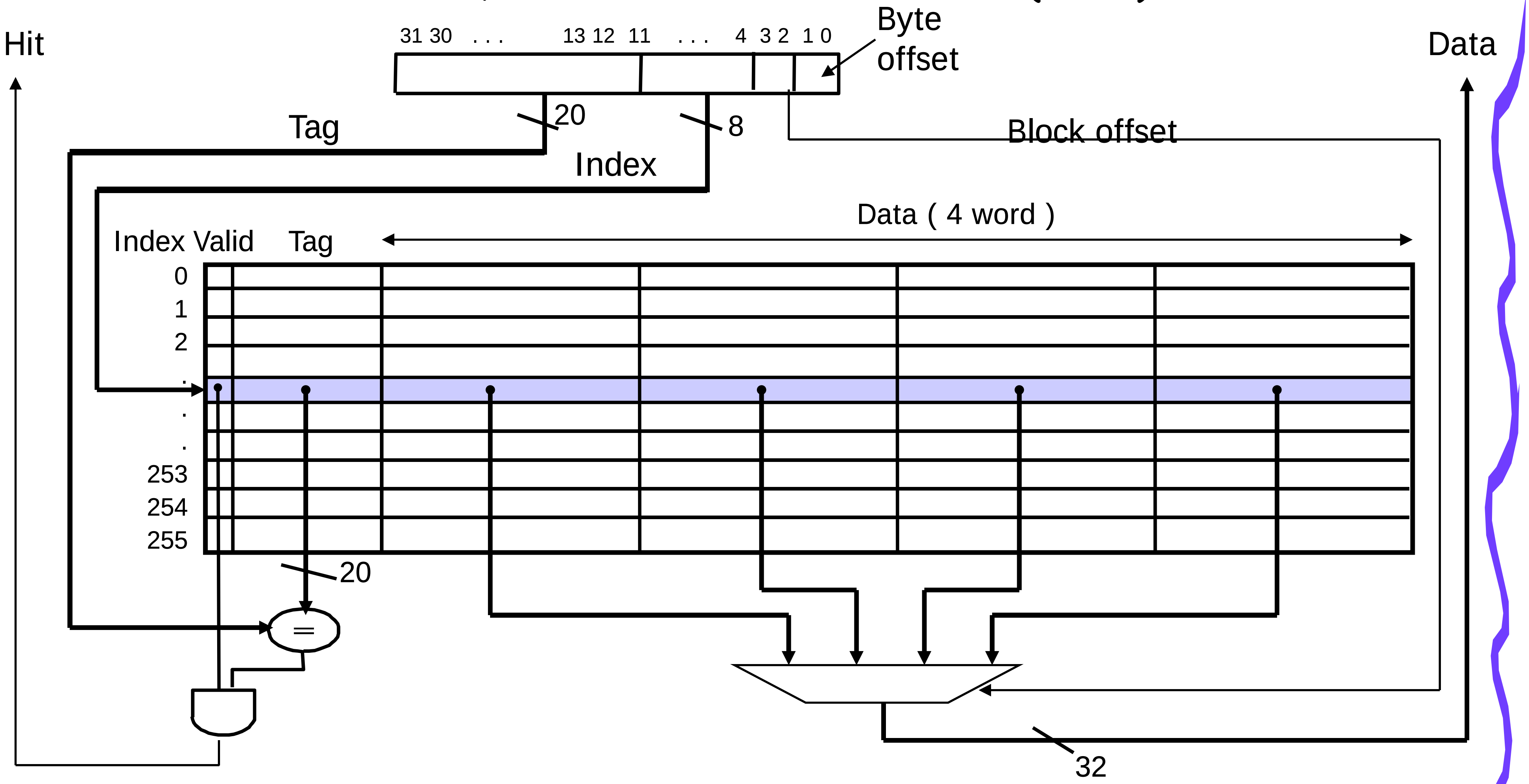
- One word/block, cache size = 1K words (4KB)



What kind of locality are we taking advantage of?

Multiword Block Direct Mapped Cache

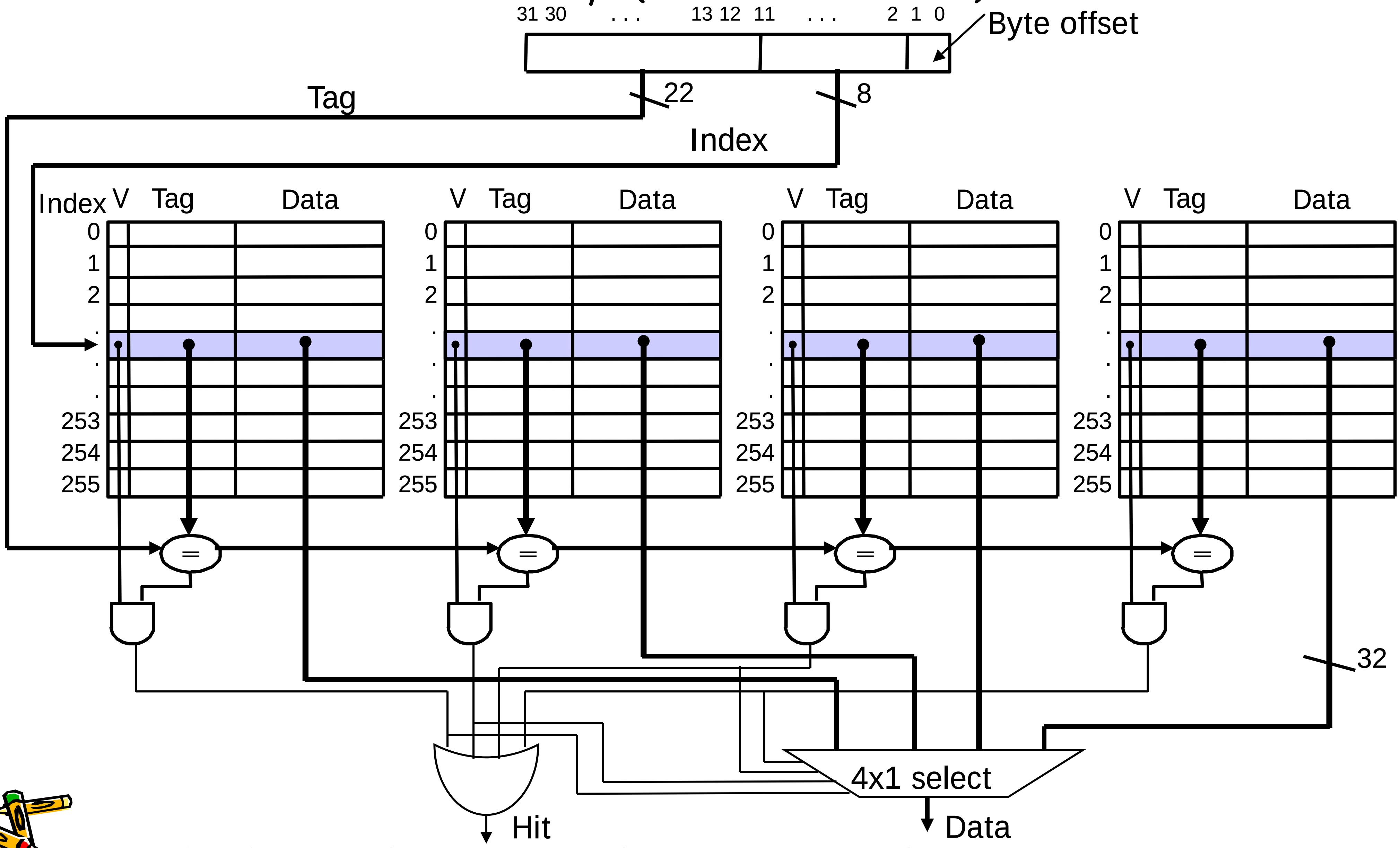
- Four words/block, cache size = 1K words (4KB)



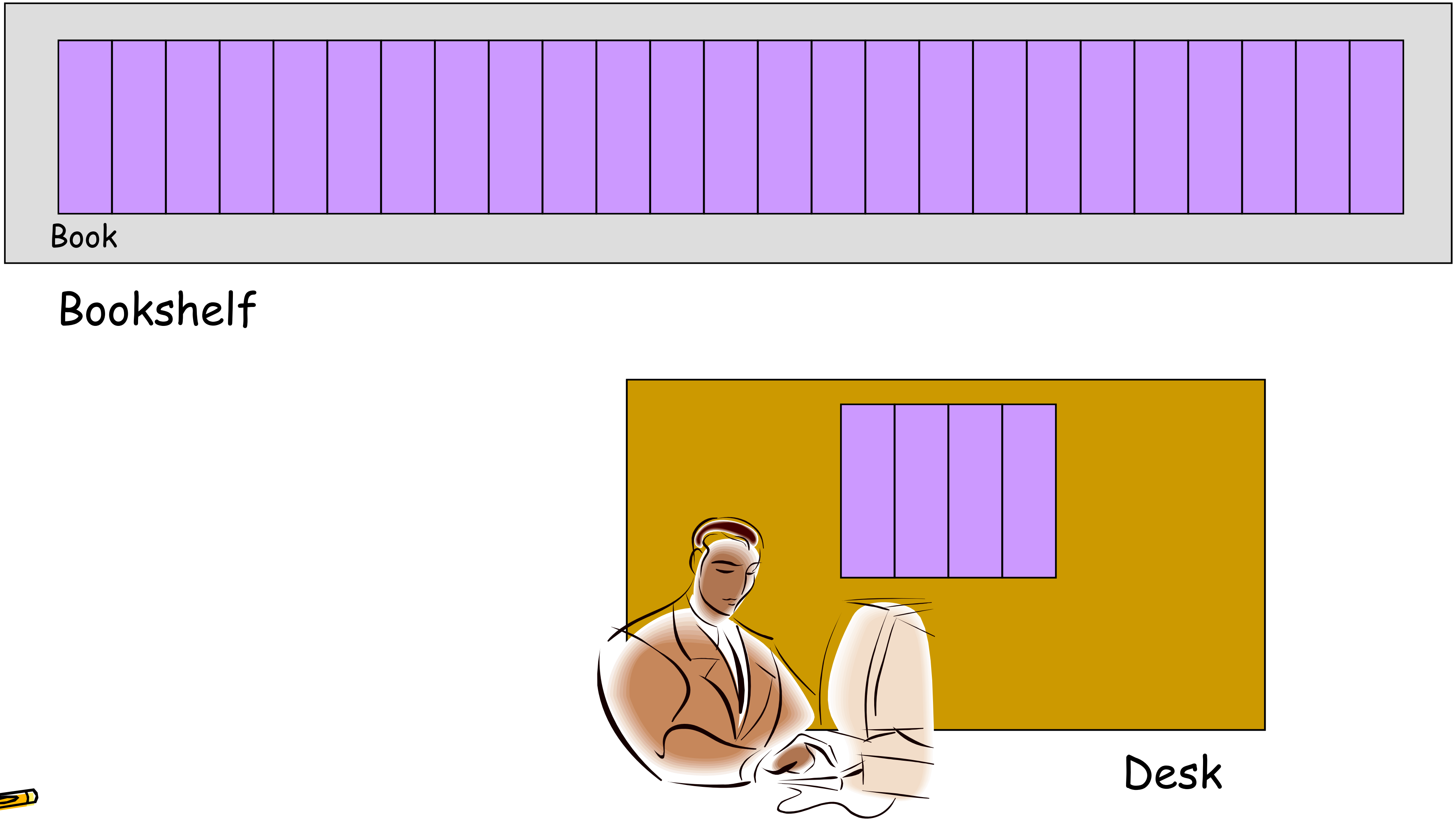
What kind of locality are we taking advantage of?

Four-Way Set Associative Cache

- 2^8 = **256 sets** each with four ways (each with one block)



Cache Associativity & Replacement Policy



Costs of Set Associative Caches

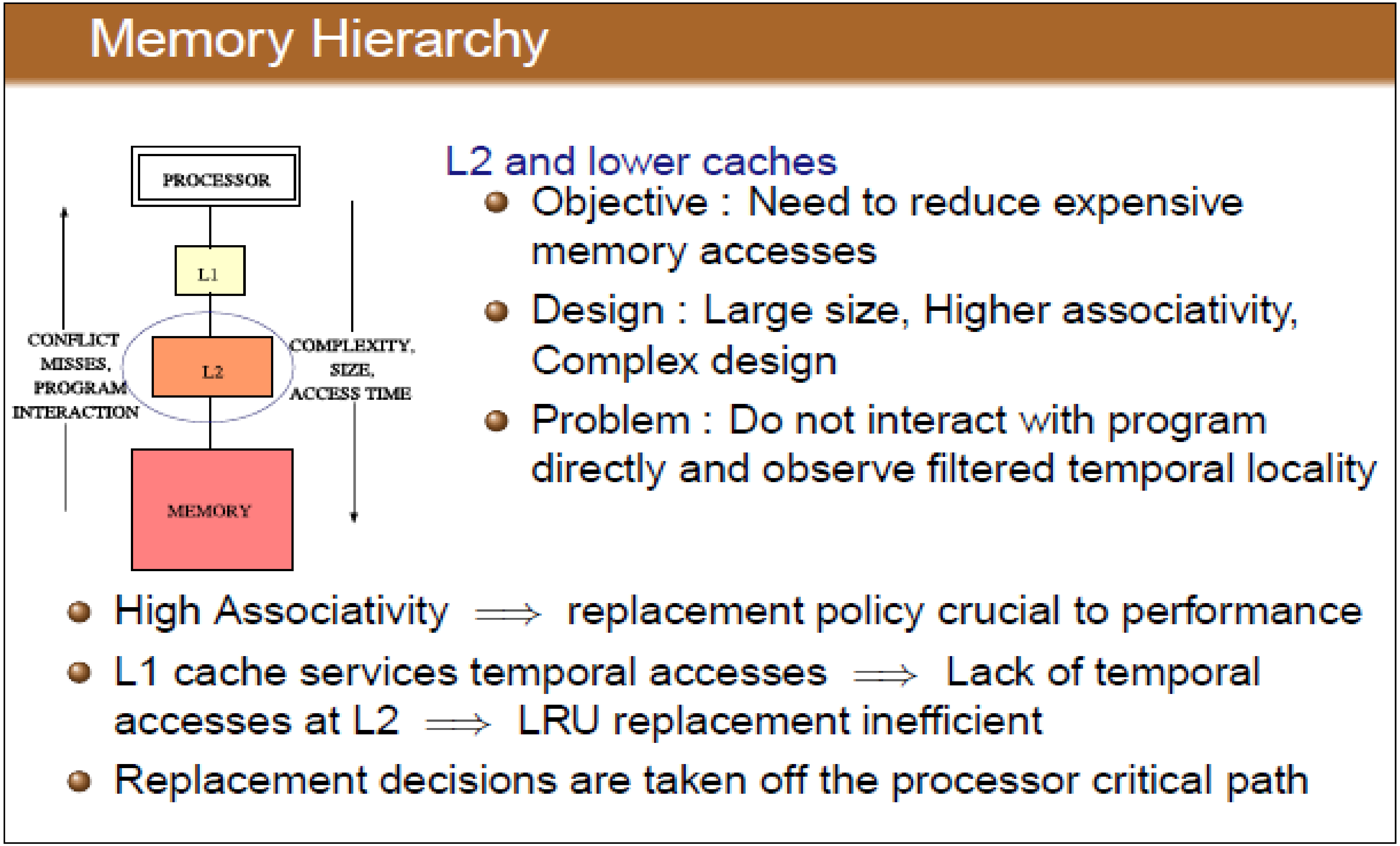
- N-way set associative cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available after set selection and Hit/Miss decision.
- When a miss occurs,
  which way's block do we pick for replacement ?
  - Least Recently Used (LRU):
    the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way's block was used
    - For 2-way set associative, takes one bit per set →
      set the bit when a block is referenced
      (and reset the other way's bit)
  - Random

Recommended Reading

- Emulating Optimal Replacement with a Shepherd Cache
  - Kaushik Rajan, Govindarajan Ramaswamy, Indian Institute of Science
  - MICRO-40, pp. 445-454, 2007
  - Session 8: Cache Replacement Policies

- A quote:
  "The inherent temporal locality in memory accesses is filtered out by the L1 cache. As a consequence, an L2 cache with LRU replacement incurs significantly higher misses than the optimal replacement policy (OPT). We propose to narrow this gap through a novel replacement strategy that mimics the replacement decisions of OPT."

Memory Hierarchy Design



Memory Hierarchy

PROCESSOR
L1
L2
MEMORY
CONFLICT MISSES, PROGRAM INTERACTION
COMPLEXITY, SIZE, ACCESS TIME

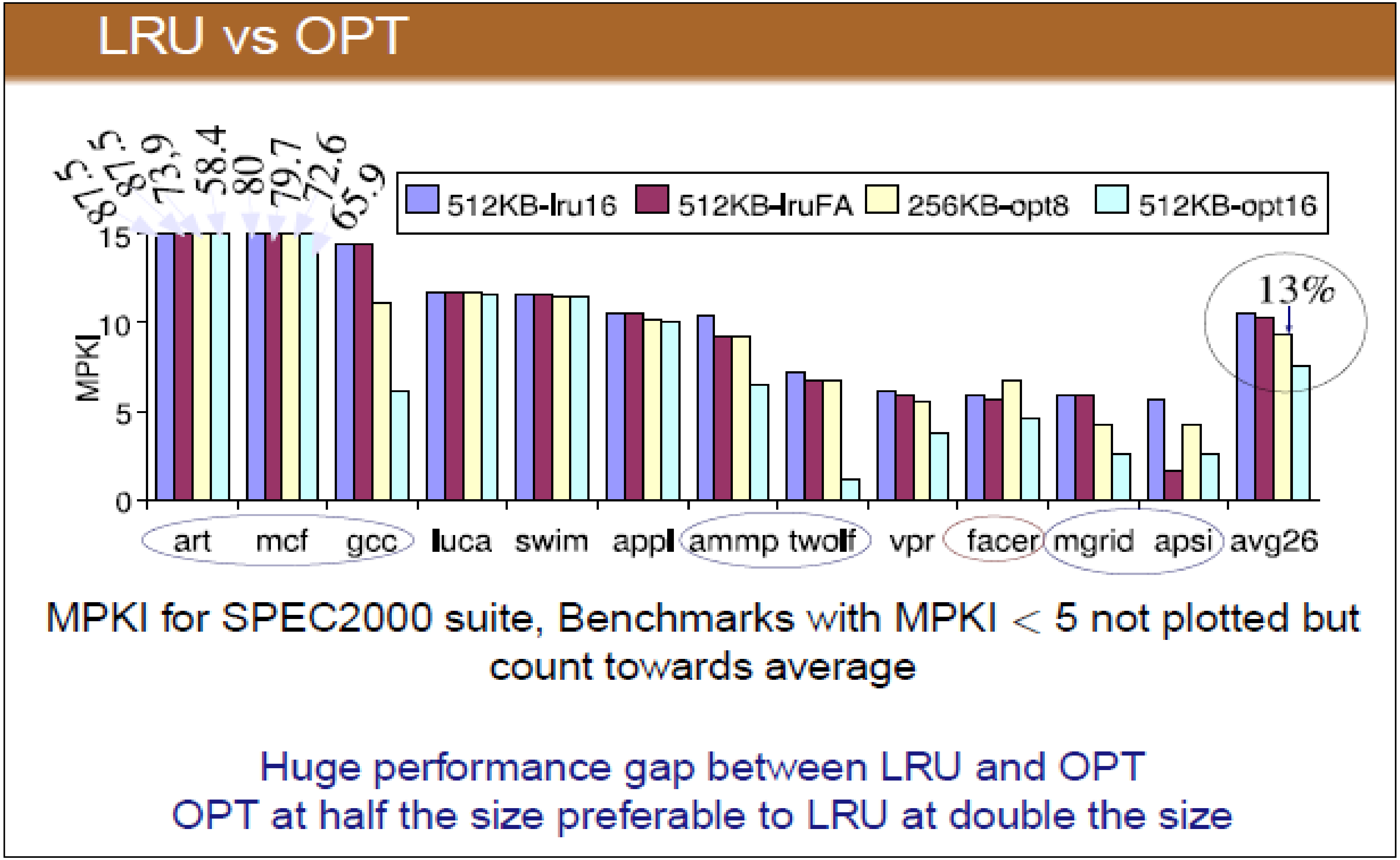
L2 and lower caches

- Objective : Need to reduce expensive memory accesses
- Design : Large size, Higher associativity, Complex design
- Problem : Do not interact with program directly and observe filtered temporal locality

- High Associativity ⟹ replacement policy crucial to performance
- L1 cache services temporal accesses ⟹ Lack of temporal accesses at L2 ⟹ LRU replacement inefficient
- Replacement decisions are taken off the processor critical path

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007

LRU has room for improvement



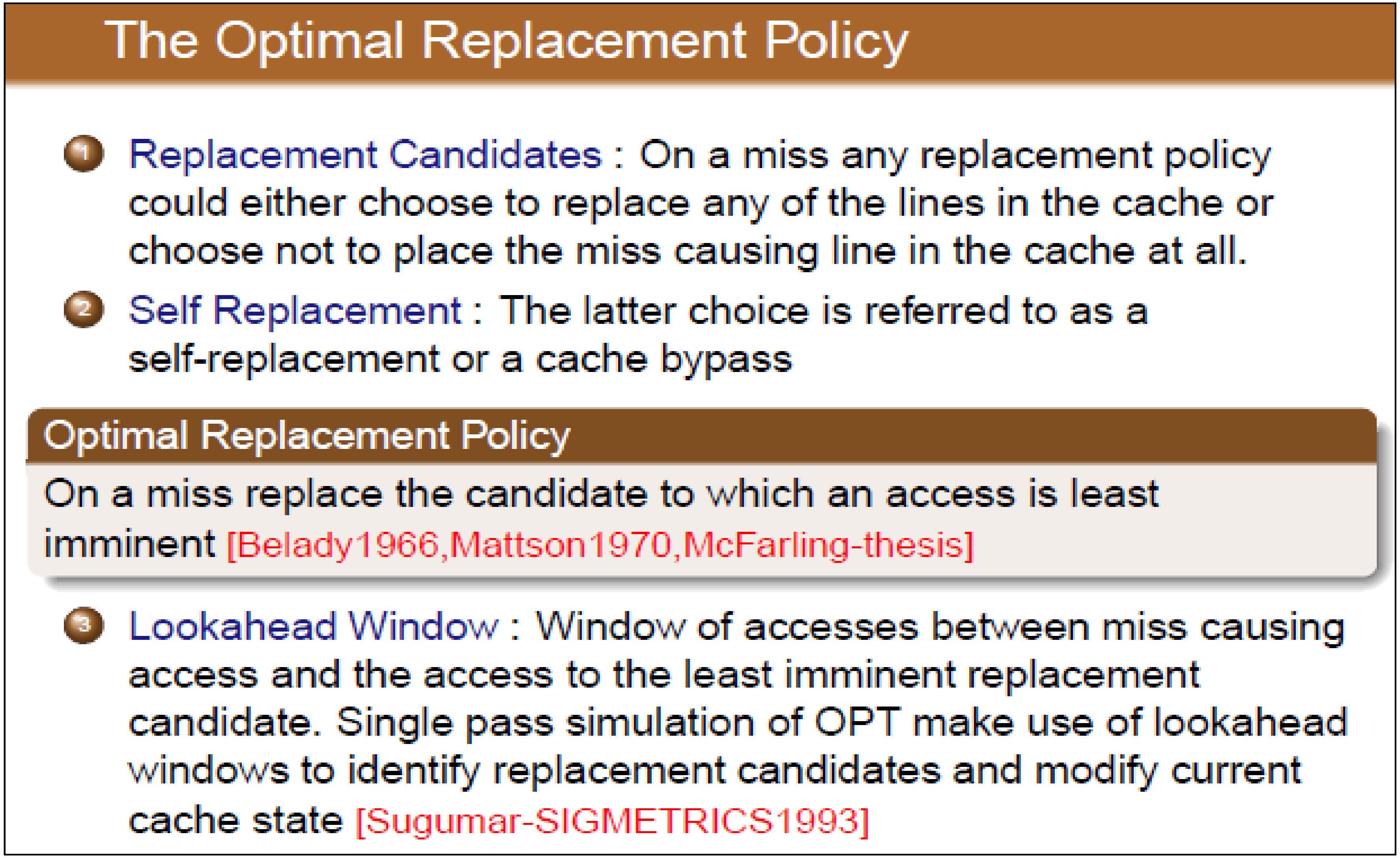
LRU vs OPT

MPKI for SPEC2000 suite, Benchmarks with MPKI < 5 not plotted but count towards average

Huge performance gap between LRU and OPT
OPT at half the size preferable to LRU at double the size

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007

OPT: Optimal Replacement Policy



The Optimal Replacement Policy

1. Replacement Candidates : On a miss any replacement policy could either choose to replace any of the lines in the cache or choose not to place the miss causing line in the cache at all.
2. Self Replacement : The latter choice is referred to as a self-replacement or a cache bypass

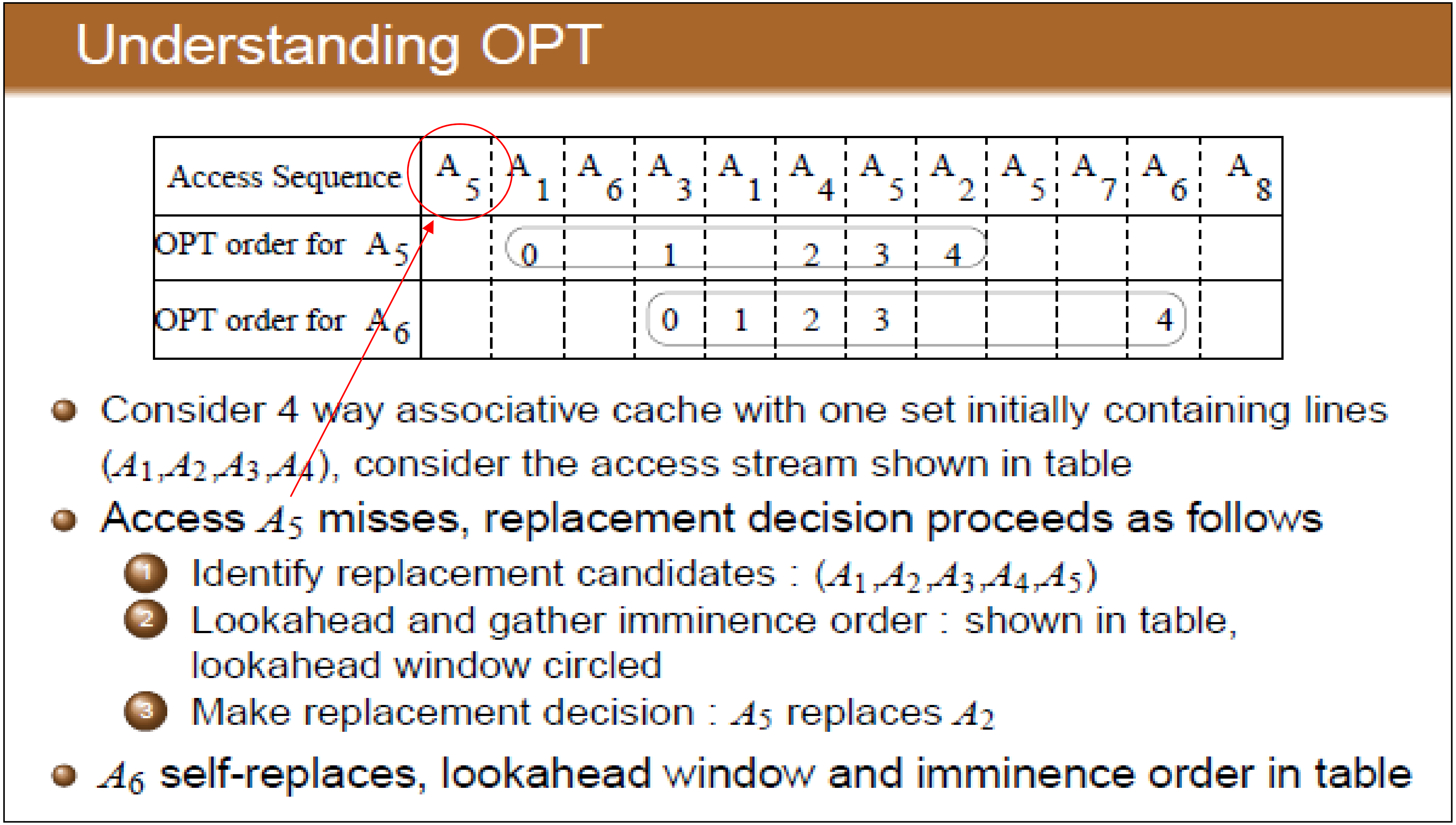
Optimal Replacement Policy

On a miss replace the candidate to which an access is least imminent [Belady1966,Mattson1970,McFarling-thesis]

3. Lookahead Window : Window of accesses between miss causing access and the access to the least imminent replacement candidate. Single pass simulation of OPT make use of lookahead windows to identify replacement candidates and modify current cache state [Sugumar-SIGMETRICS1993]

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007

Example of Optimal Replacement Policy



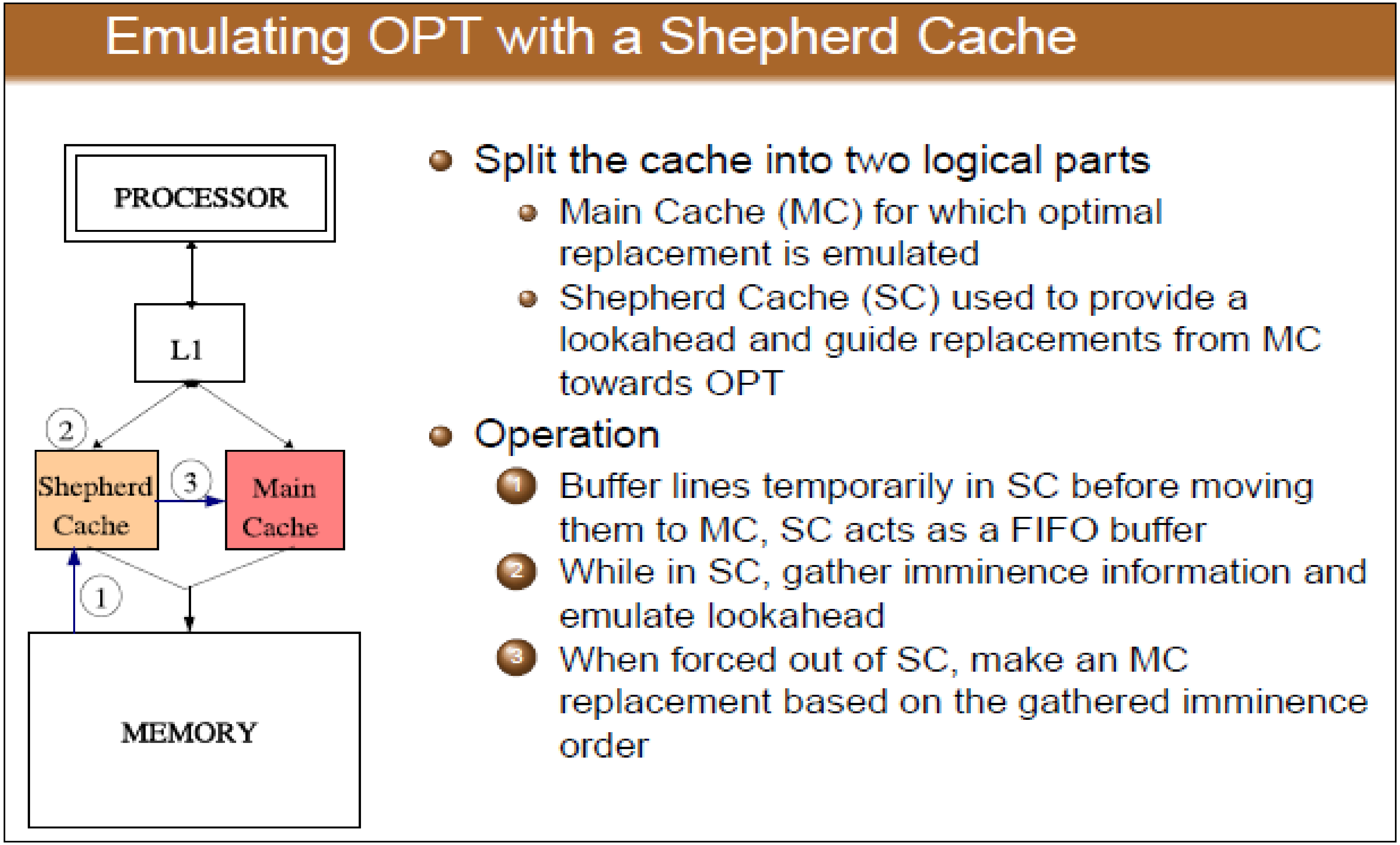
Understanding OPT

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1, A_2, A_3, A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows
  1. Identify replacement candidates : $(A_1, A_2, A_3, A_4, A_5)$
  2. Lookahead and gather imminence order : shown in table, lookahead window circled
  3. Make replacement decision : A_5 replaces A_2
- A_6 self-replaces, lookahead window and imminence order in table

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007

Shepherd Cache emulation OPT

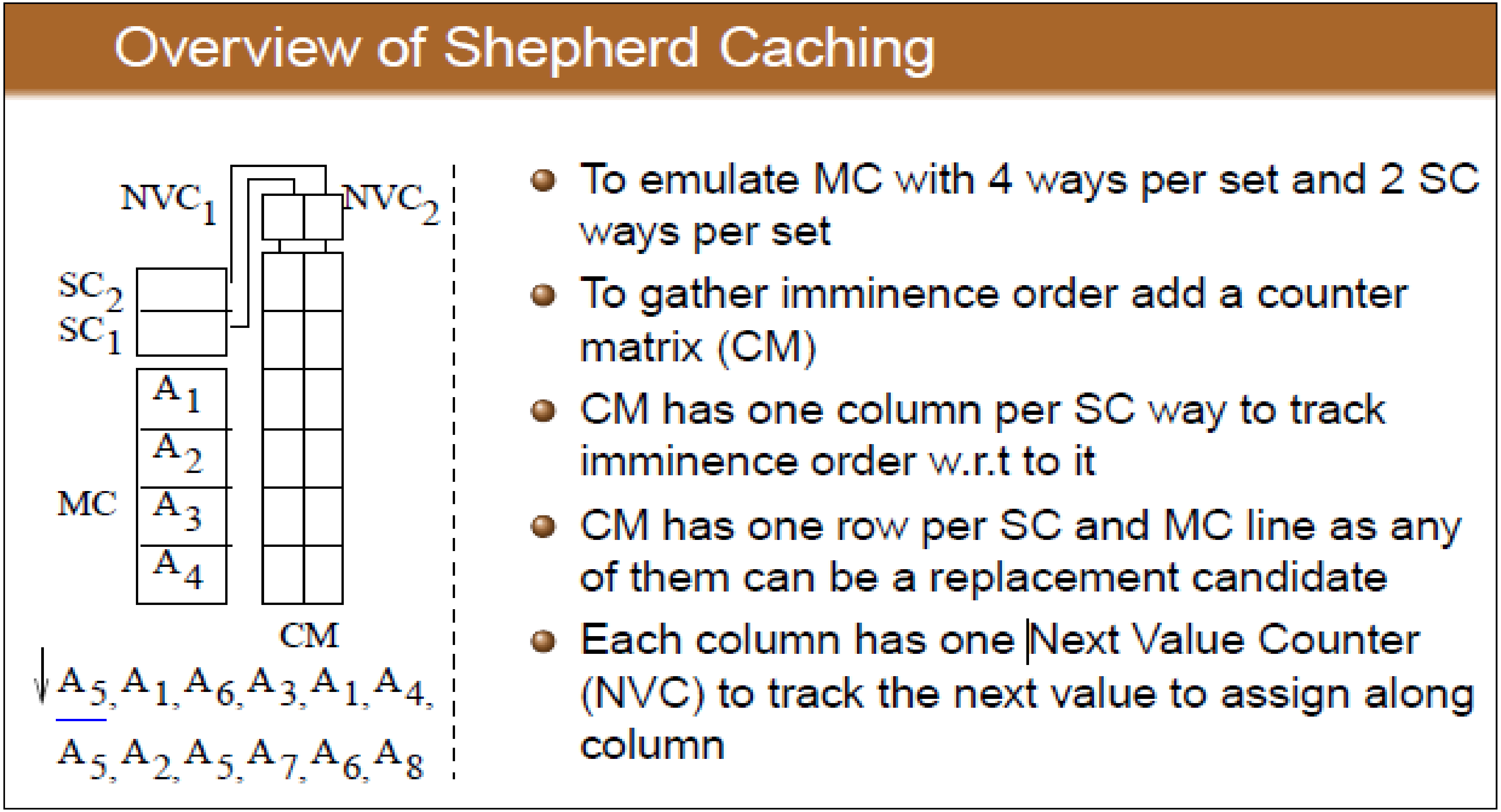


Emulating OPT with a Shepherd Cache

- Split the cache into two logical parts
  - Main Cache (MC) for which optimal replacement is emulated
  - Shepherd Cache (SC) used to provide a lookahead and guide replacements from MC towards OPT
- Operation
  1. Buffer lines temporarily in SC before moving them to MC, SC acts as a FIFO buffer
  2. While in SC, gather imminence information and emulate lookahead
  3. When forced out of SC, make an MC replacement based on the gathered imminence order

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007

Shepherd Cache Overview

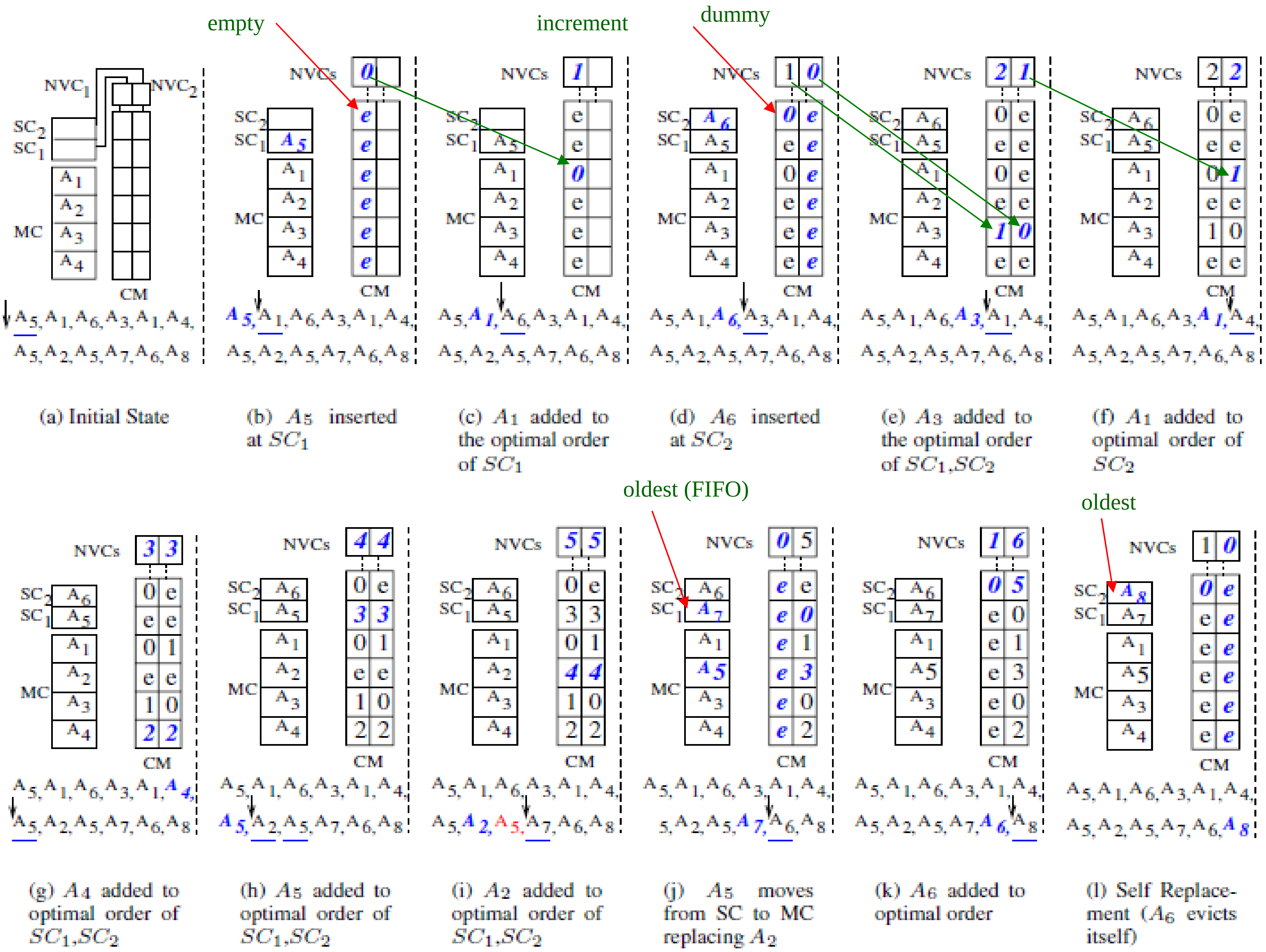


Overview of Shepherd Caching

NVC$_1$
NVC$_2$
SC$_2$
SC$_1$
A$_1$
A$_2$
MC
A$_3$
A$_4$
CM
↓A$_5$, A$_1$, A$_6$, A$_3$, A$_1$, A$_4$,
A$_5$, A$_2$, A$_5$, A$_7$, A$_6$, A$_8$

- To emulate MC with 4 ways per set and 2 SC ways per set
- To gather imminence order add a counter matrix (CM)
- CM has one column per SC way to track imminence order w.r.t to it
- CM has one row per SC and MC line as any of them can be a replacement candidate
- Each column has one Next Value Counter (NVC) to track the next value to assign along column

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007



(a) Initial State

(b) A_5 inserted at SC_1

(c) A_1 added to the optimal order of SC_1

(d) A_6 inserted at SC_2

(e) A_3 added to the optimal order of SC_1,SC_2

(f) A_1 added to optimal order of SC_2

(g) A_4 added to optimal order of SC_1,SC_2

(h) A_5 added to optimal order of SC_1,SC_2

(i) A_2 added to optimal order of SC_1,SC_2

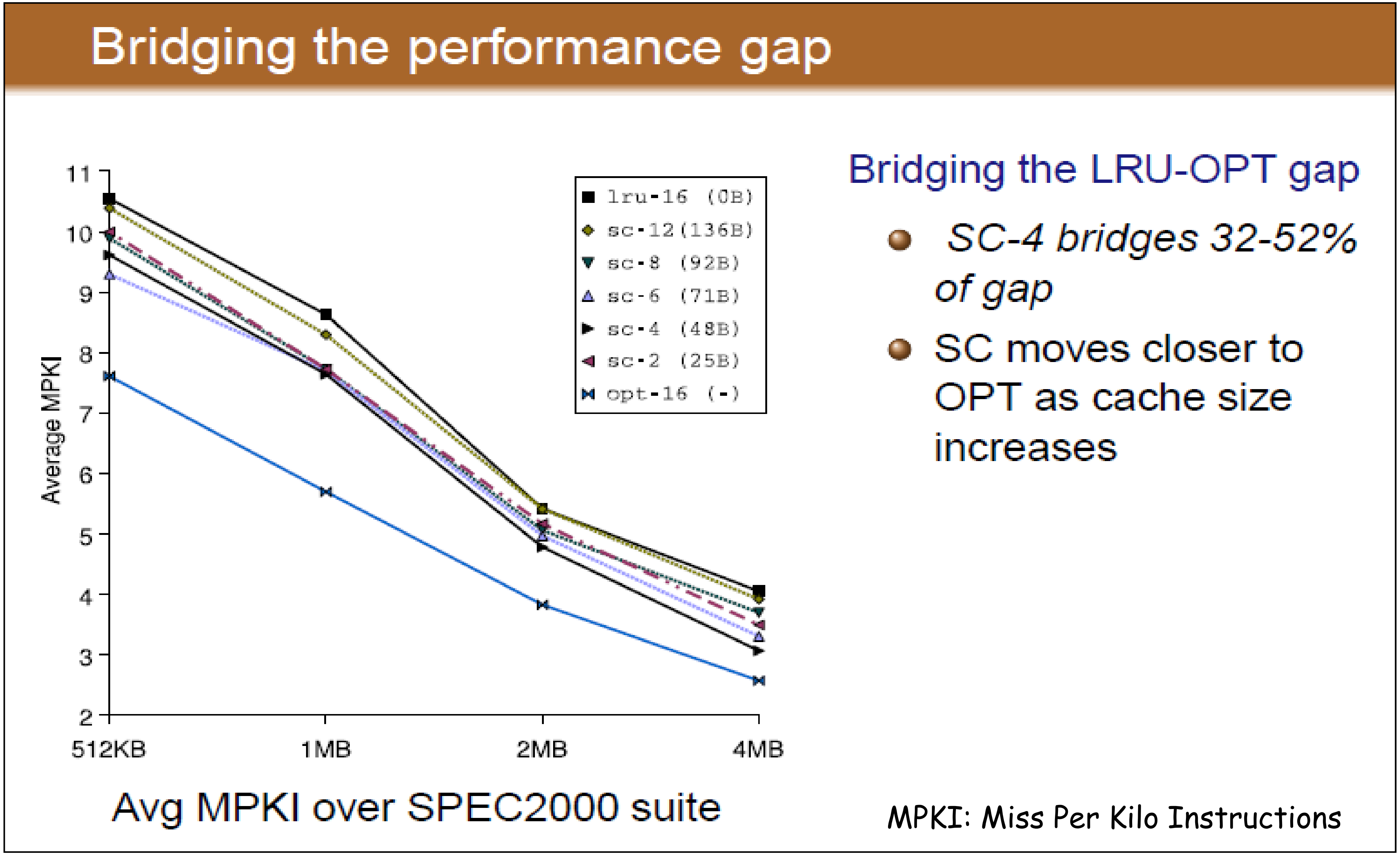
(j) A_5 moves from SC to MC replacing A_2

(k) A_6 added to optimal order

(l) Self Replacement (A_6 evicts itself)

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007

Shepherd cache bridges 32 - 52% of the gap



Bridging the performance gap

Avg MPKI over SPEC2000 suite

Bridging the LRU-OPT gap

- *SC-4 bridges 32-52% of gap*
- SC moves closer to OPT as cache size increases

MPKI: Miss Per Kilo Instructions

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007

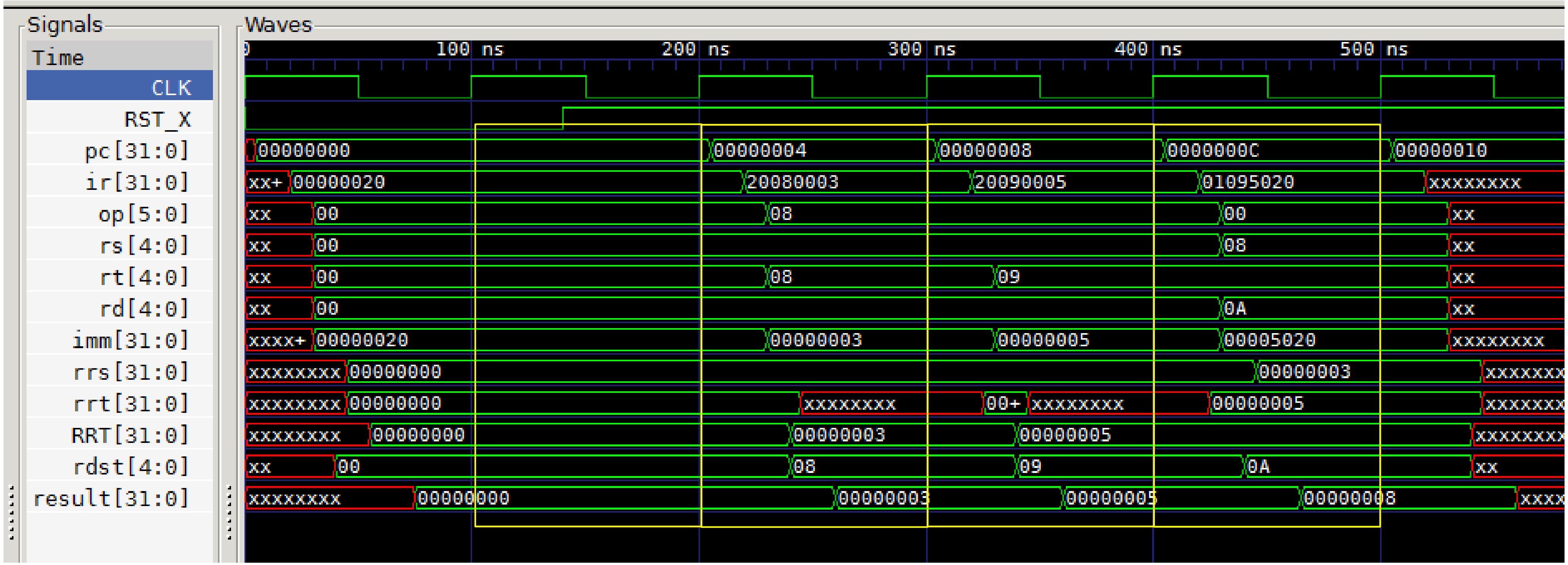
Homework 3

1. Design a single-cycle processor supporting MIPS add, addi, lw and sw instructions in Verilog HDL. Please download proc03.v from the support page and refer it.
2. Verify the behavior of designed processor using following assembly code

```
•  add  $0,  $0,  $0   # NOP {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20}
•  addi $t0, $zero, 8  # {6'h8, 5'd0, 5'd8, 16'd8}
•  sw   $t0, 4($t0)    #
•  lw   $t1, 4($t0)    #
•  addi $t2, $t1, 6    #
```

3. Submit a report printed on A4 paper at the beginning of the next lecture.
   - The report should include a block diagram, a source code in Verilog HDL, and obtained waveforms of your design.

Waveform of proc02



Signals
Waves
Time
0
100 ns
200 ns
300 ns
400 ns
500 ns
CLK
RST_X
pc[31:0] 00000000 00000004 00000008 0000000C 00000010
ir[31:0] xx+ 00000020 20080003 20090005 01095020 xxxxxxxx
op[5:0] xx 00 08 00 xx
rs[4:0] xx 00 08 xx
rt[4:0] xx 00 08 09 xx
rd[4:0] xx 00 0A xx
imm[31:0] xxxx+ 00000020 00000003 00000005 00005020 xxxxxxxx
rrs[31:0] xxxxxxxx 00000000 00000003 xxxxxxx
rrt[31:0] xxxxxxxx 00000000 xxxxxxxx 00+ xxxxxxxx 00000005 xxxxxxx
RRT[31:0] xxxxxxxx 00000000 00000003 00000005 xxxxxxxx
rdst[4:0] xx 00 08 09 0A xx
result[31:0] xxxxxxxx 00000000 00000003 00000005 00000008 xxxx

```
add  $0,   $0,   $0   # NOP {6'h0, 5'd0, 5'd0, 5'd0, 5'd0, 6'h20}
addi $t0, $zero, 3   # {6'h8, 5'd0, 5'd8, 16'd3}
addi $t1, $zero, 5   # {6'h8, 5'd0, 5'd9, 16'd5}
add  $t2, $t0, $t1   # {6'h0, 5'd8, 5'd9, 5'd10, 5'd0, 6'h20}
```