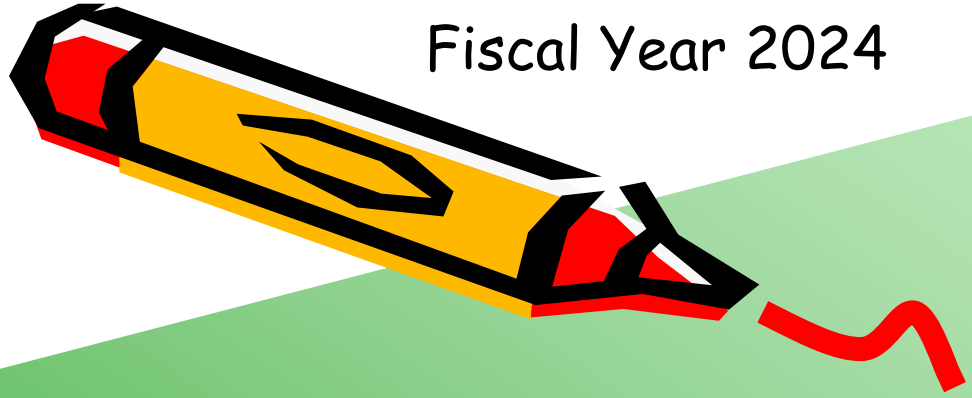


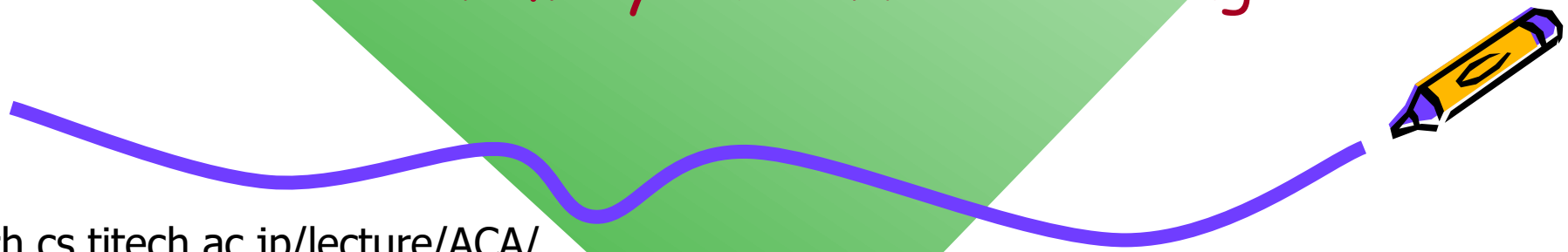
Fiscal Year 2024

Ver. 2025-01-20a

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

Advanced Computer Architecture

9. Instruction Level Parallelism: Cache Memory and Multithreading



www.arch.cs.titech.ac.jp/lecture/ACA/
Room No. W8E-308, Lecture (Face-to-face)
Mon 13:30-15:10, Thr 13:30-15:10

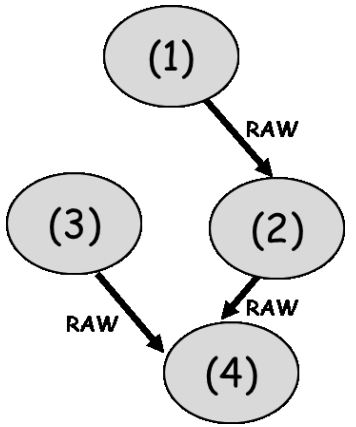
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp

Exploiting Instruction Level Parallelism (ILP)

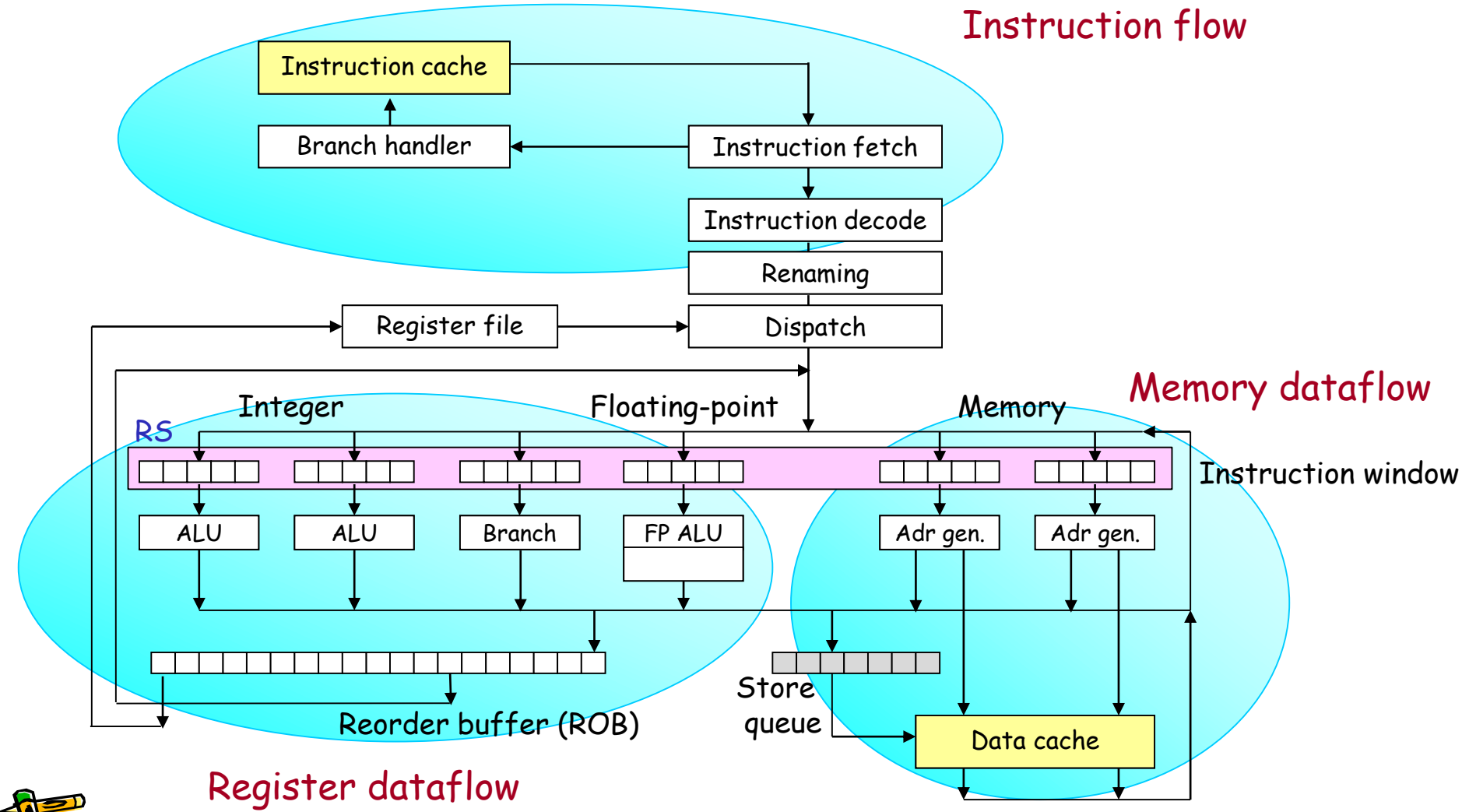
- A superscalar has to handle some flows efficiently to exploit ILP
  - Control flow (control dependence)
    - To execute ***n*** instructions per clock cycle, the processor has to fetch at least ***n*** instructions per cycle.
    - The main obstacles are branch instruction (BNE)
    - Prediction
    - Another obstacle is instruction cache
  - Register data flow (data dependence)
    - Out-of-order execution
      - Register renaming
      - Dynamic scheduling
  - Memory data flow
    - Out-of-order execution
    - Another obstacle is data cache

```
(1) add x5,x1,x2
(2) add x9,x5,x3
(3) lw  x4, 4(x7)
(4) add x8,x9,x4
```

```
(3) lw  x4, 4(x7)
(1) add x5,x1,x2
(2) add x9,x5,x3
(4) add x8,x9,x4
```

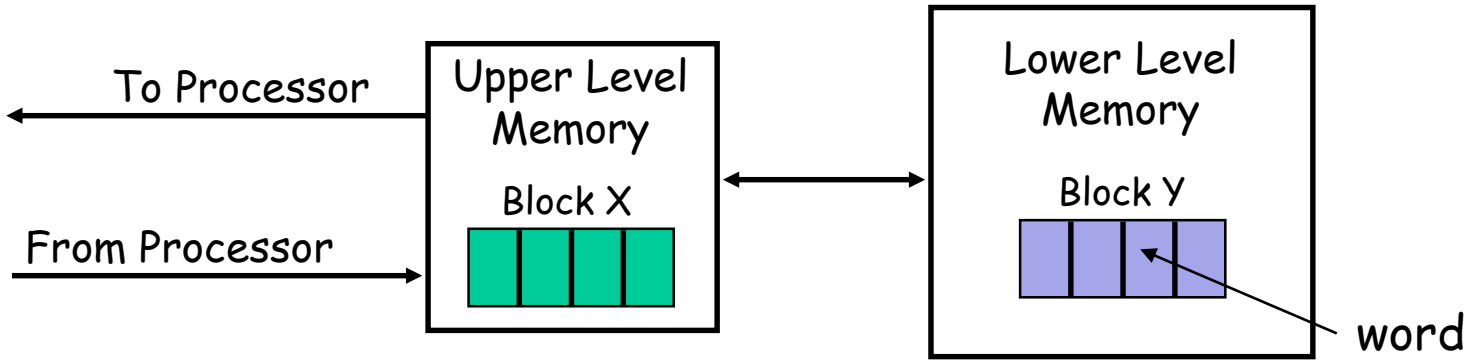


Datapath of OoO execution processor



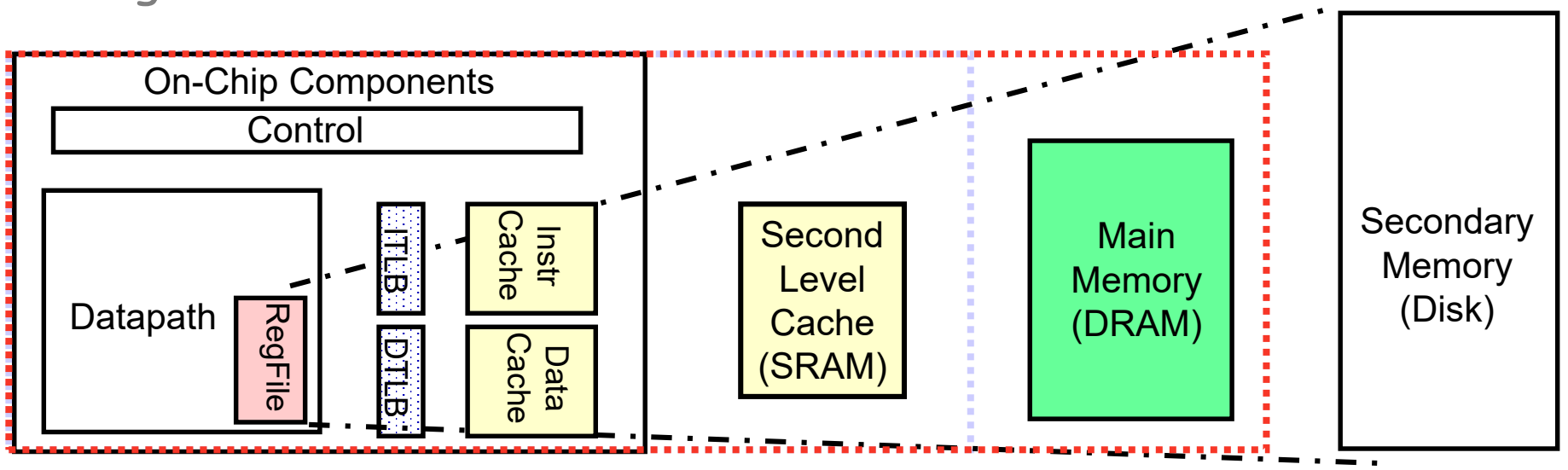
The Memory System's Fact and Goal

- Fact:
  Large memories are slow, and fast memories are small
- How do we create a memory that gives the illusion of being large, fast, and cheap?
- Cache memories
  - Temporal locality (Locality in Time):
    - Keep most recently accessed data items closer to the processor
  - Spatial locality (Locality in Space)
    - Move blocks consisting of contiguous words to the upper levels



Cache memory

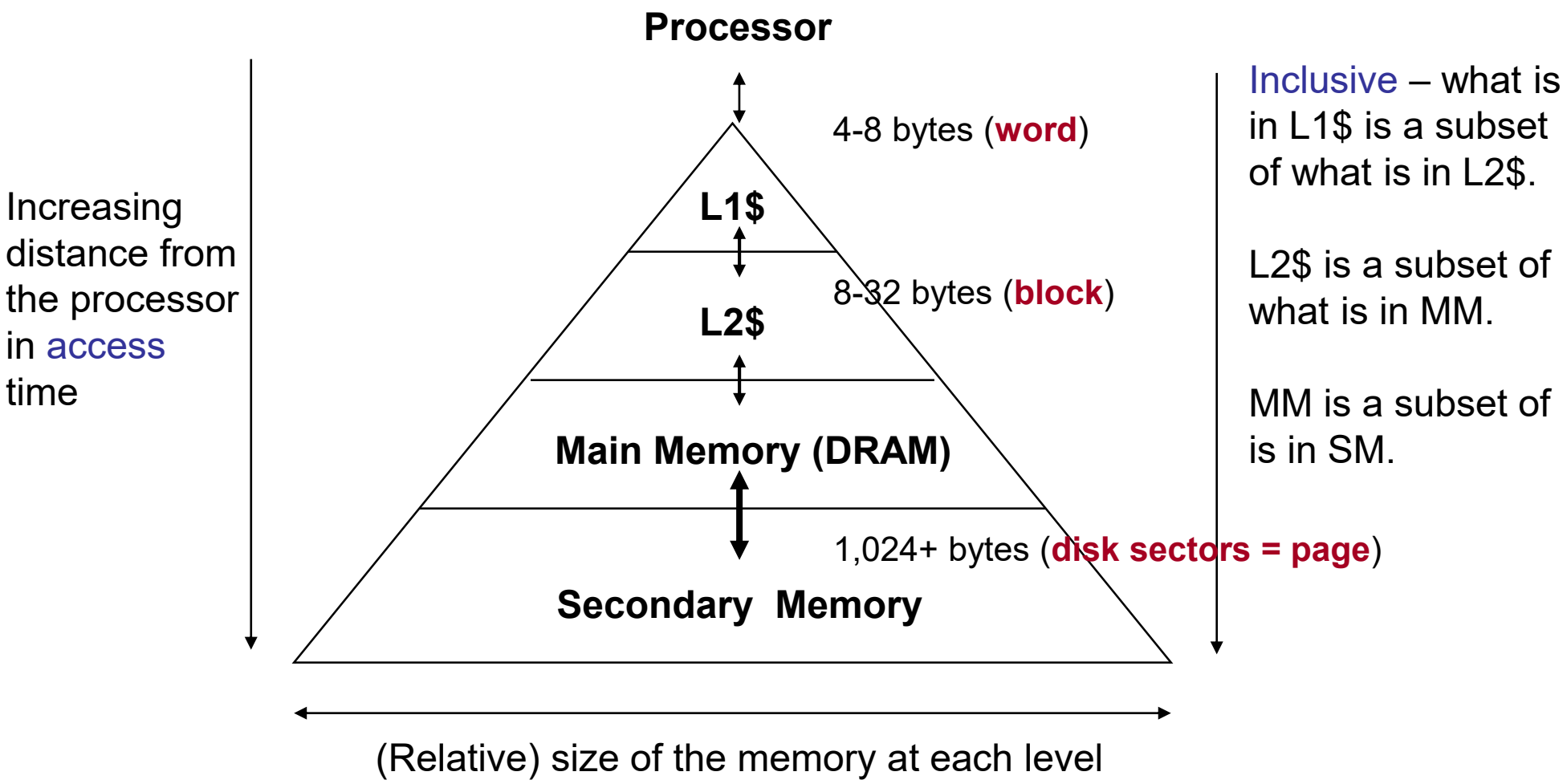
- *Cache memory* consists of a small, fast memory that acts as a buffer for the large memory.
- The nontechnical definition of *cache* is a safe place for hiding things.



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

Characteristics of the Memory Hierarchy



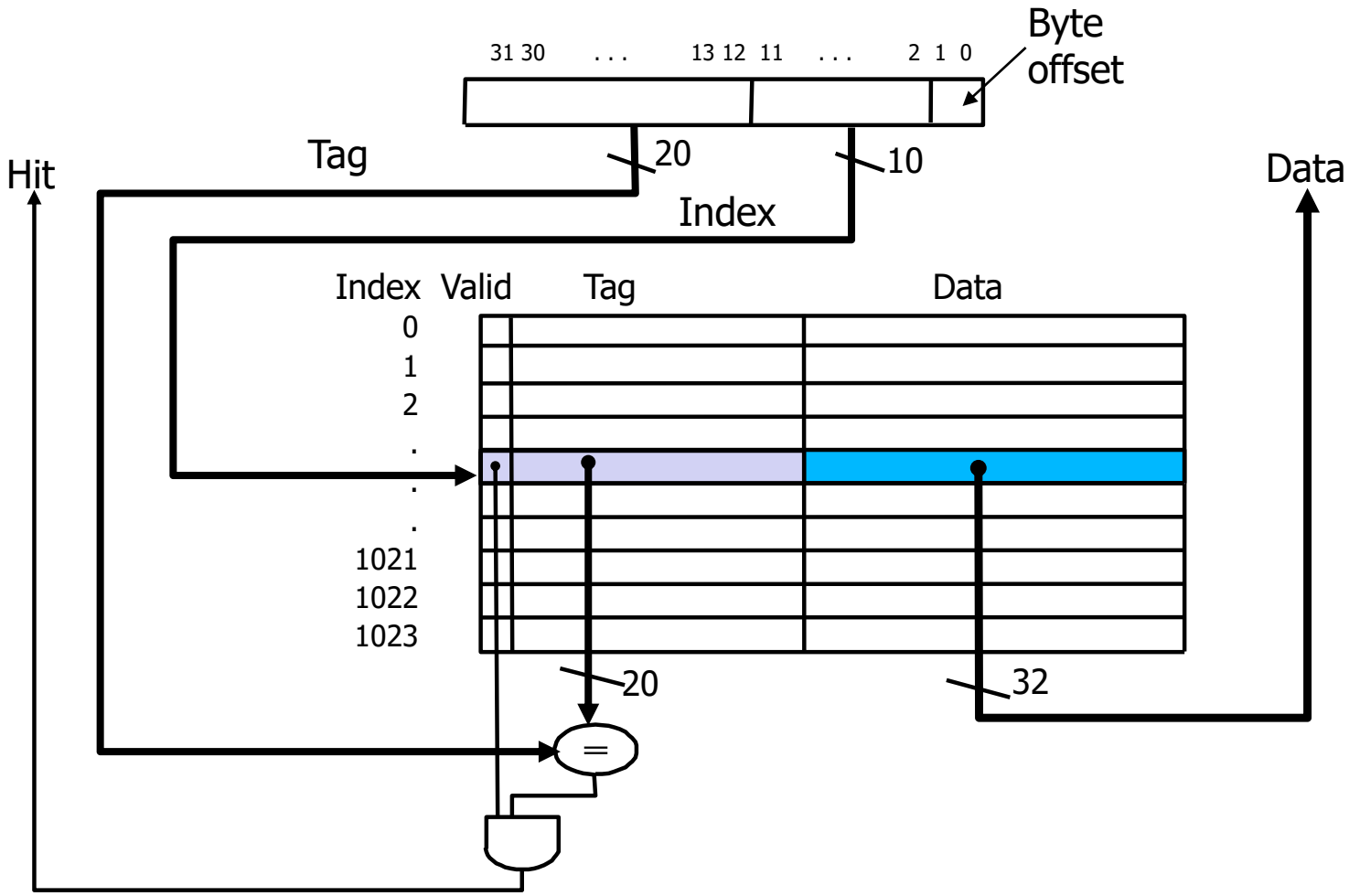
Inclusive – what is in L1$ is a subset of what is in L2$.

L2$ is a subset of what is in MM.

MM is a subset of is in SM.

Direct Mapped Cache Example

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



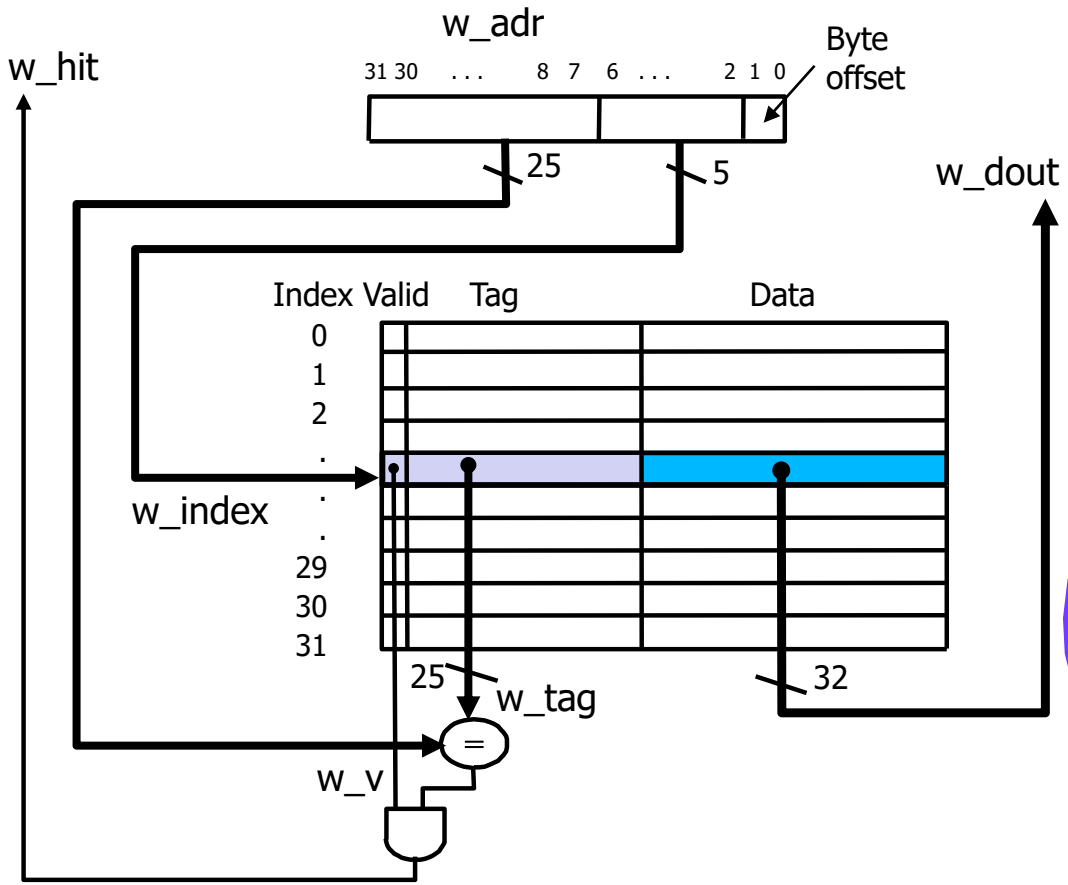
Direct Mapped Cache Example in Verilog HDL

```
module m_cache_direct_mapped_32 (
  input  wire        w_clk,
  input  wire        w_we,
  input  wire [31:0] w_adr,
  input  wire [4:0]  w_wadr,
  input  wire [57:0] w_wd,
  output wire        w_hit,
  output wire [31:0] w_dout
);

  reg [57:0] mem [0:31];
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;

  wire [4:0]  w_index = w_adr[6:2];
  wire        w_v;
  wire [24:0] w_tag;
  assign {w_v, w_tag, w_dout} = mem[w_index];
  assign w_hit = w_v & (w_adr[31:7]==w_tag);

  always @(posedge w_clk) if (w_we) mem[w_wadr] <= w_wd;
endmodule
```



Example Behavior of 4-entry Direct Mapped Cache

- Consider the main memory word reference string (word addresses) 0 1 2 3 4 3 4 15

Start with an empty cache - all blocks initially marked as not valid

Tag

0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

01 ~~00~~ Mem(0) 4

00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

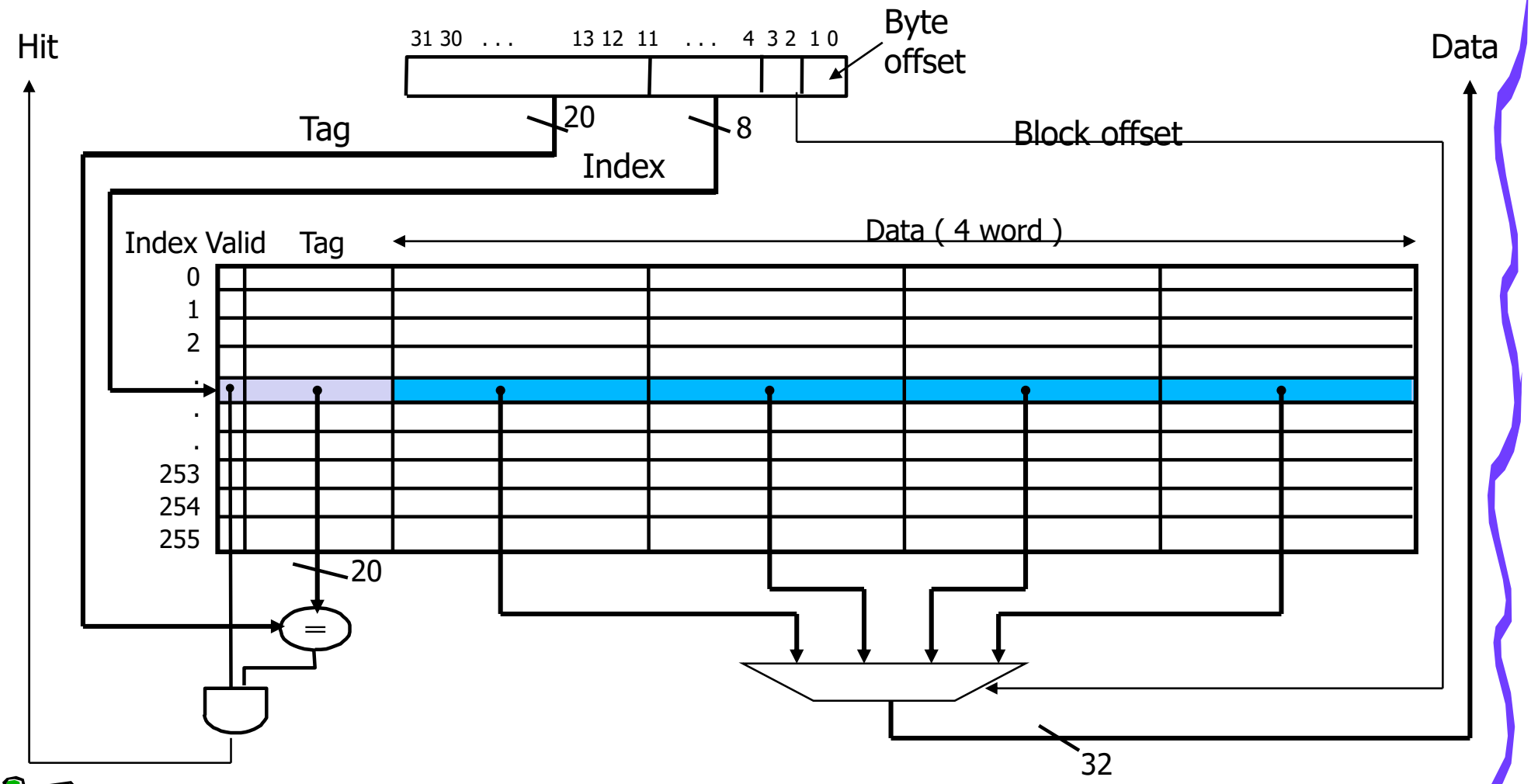
01	Mem(4)
00	Mem(1)
00	Mem(2)
~~00~~	Mem(~~3~~)

11 15

- 8 requests, 6 misses

Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words



Taking Advantage of Spatial Locality

- Let cache block hold more than one word (two words/block)
  - 0 1 2 3 4 3 4 15

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

01 ~~00~~	Mem(1) 5 ~~Mem(1)~~	Mem(0) 4 ~~Mem(0)~~
00	Mem(3)	Mem(2)

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

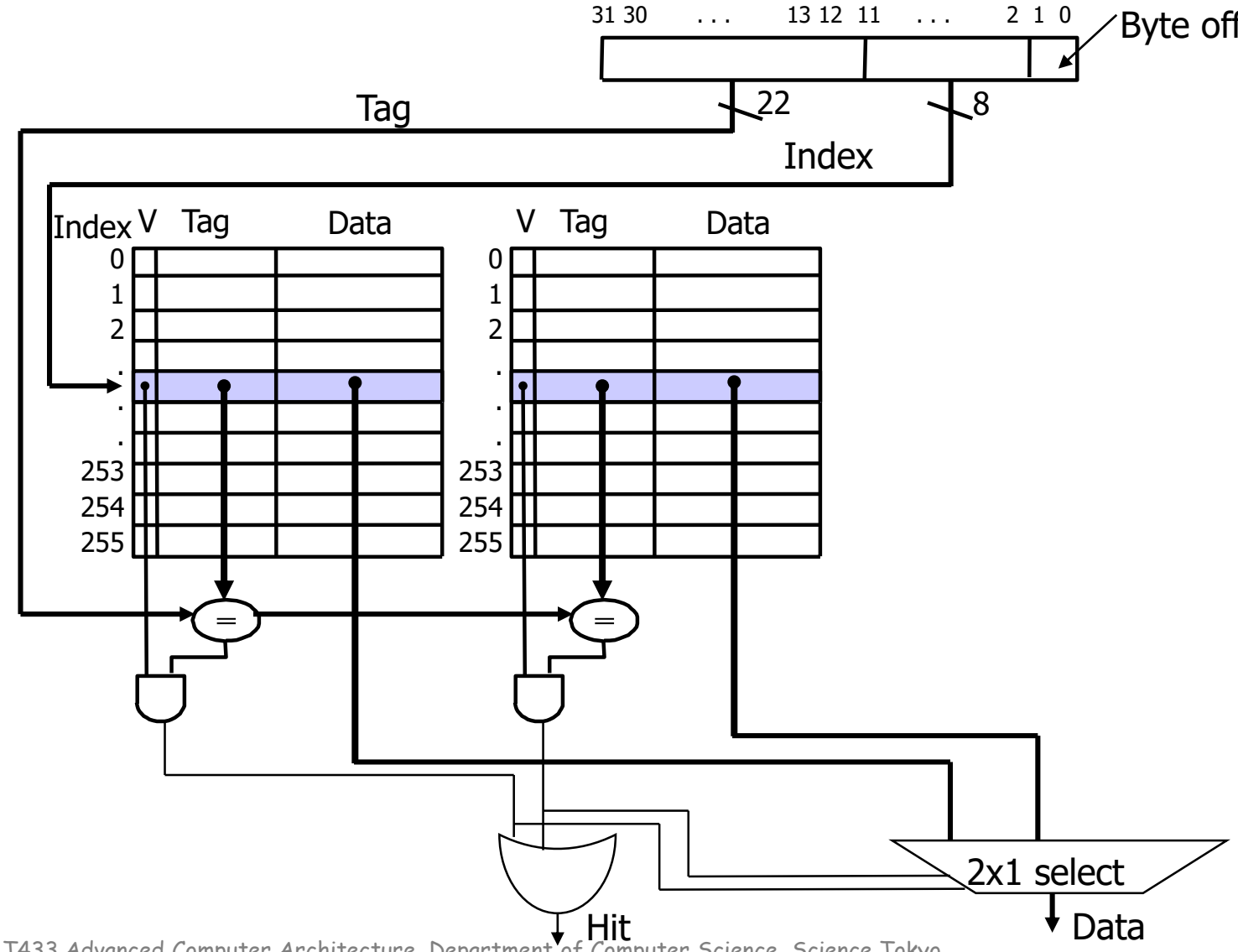
15 miss

11 01 ~~00~~	Mem(5) 15 ~~Mem(3)~~	Mem(4) 14 ~~Mem(2)~~

- 8 requests, 4 misses

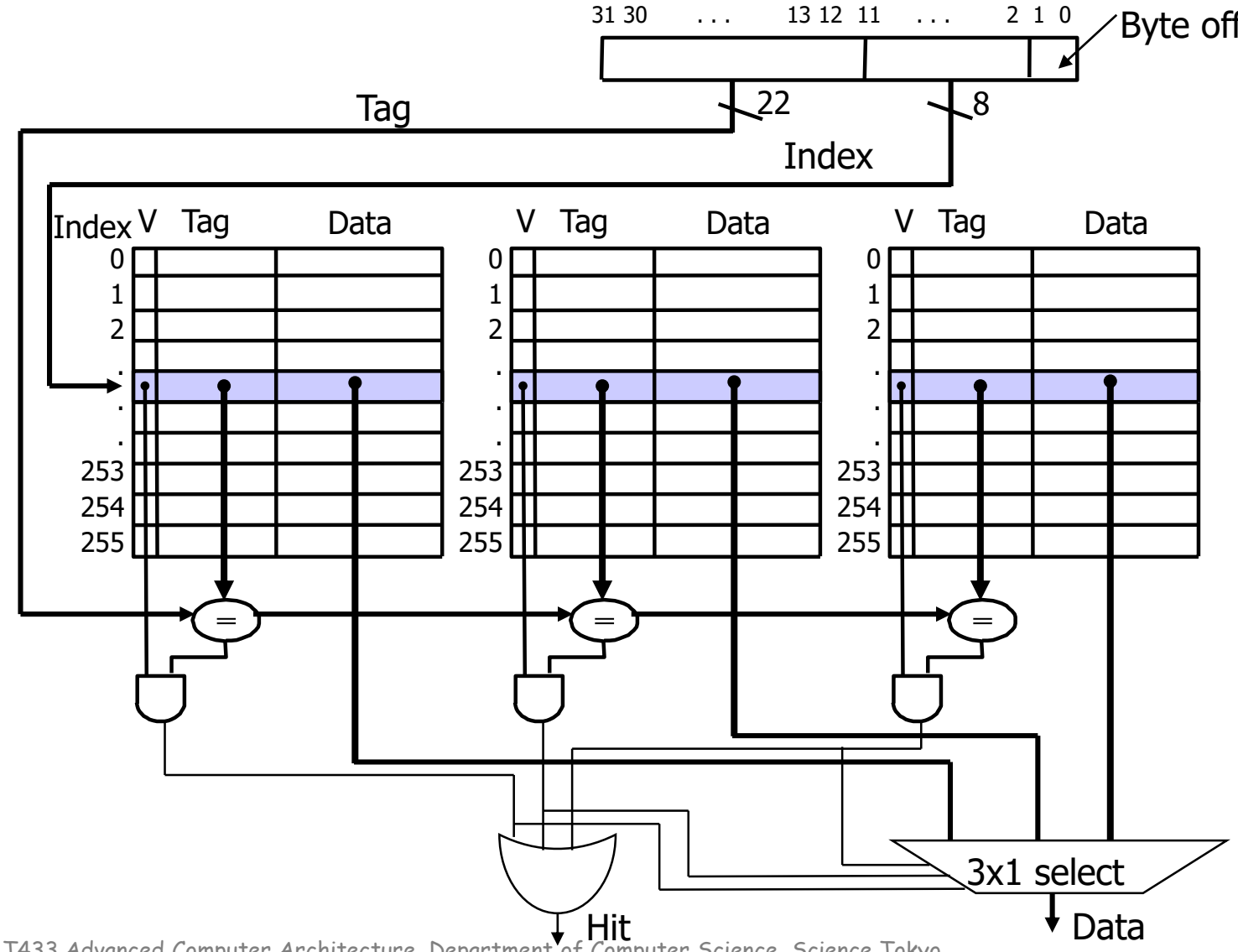
Two-Way Set Associative Cache

- One word/block, 2^8 = **256 sets** where each with three ways (each with one block)



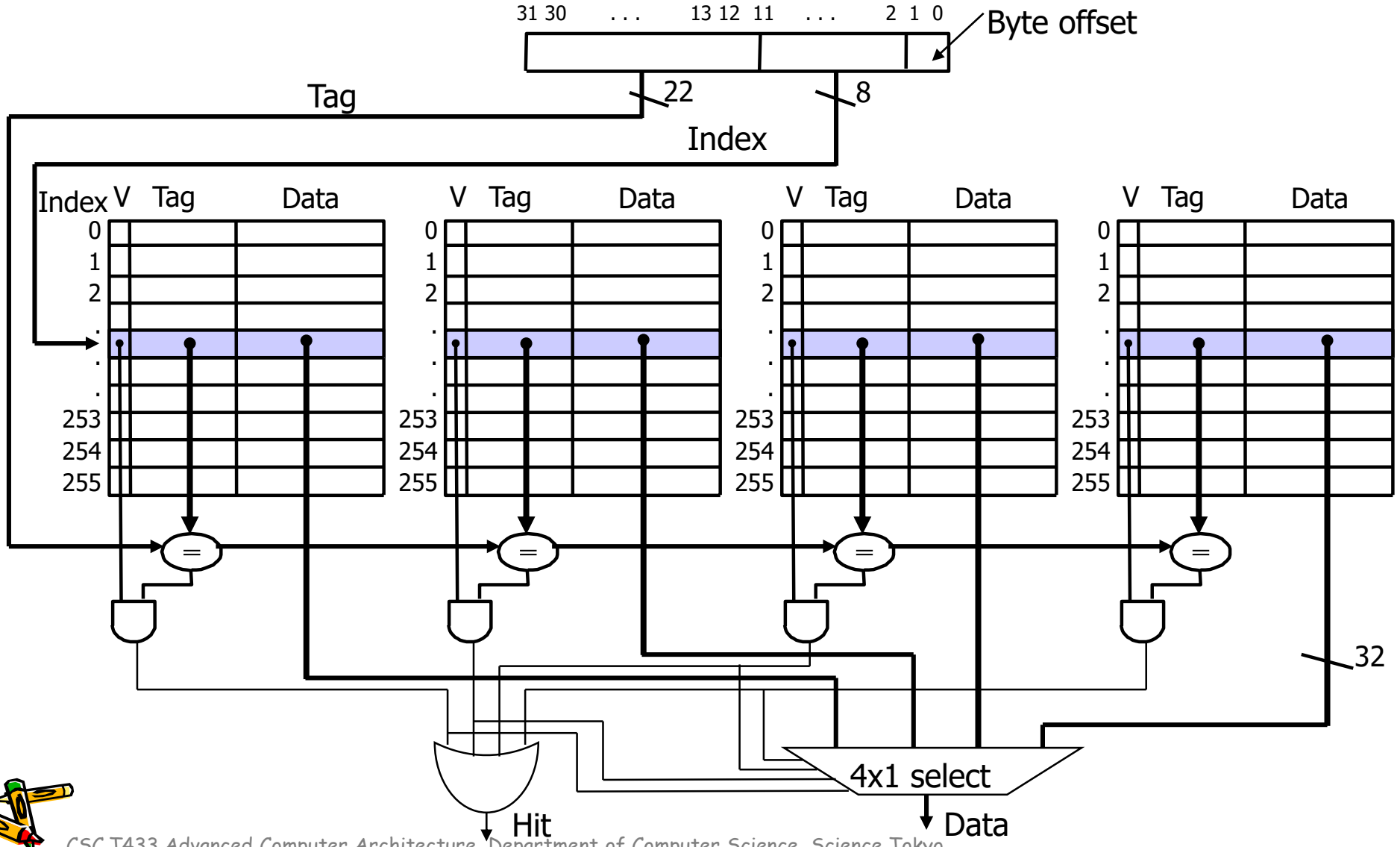
Three-Way Set Associative Cache

- One word/block, 2^8 = **256 sets** where each with three ways (each with one block)



Four-Way Set Associative Cache

- One word/block, 2^8 = **256 sets** where each with four ways (each with one block)



Set Associative Caches

- 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
- 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.

Recommended Reading (again)

- Focused Value Prediction
  - Sumeet Bandishte, Jayesh Gaur, Zeev Sperber, Lihu Rappoport, Adi Yoaz, and Sreenivas Subramoney, Intel
  - ACM/IEEE 47th International Symposium on Computer Architecture (ISCA), pp. 79-91, 2020
- A quote:
  "Value Prediction was proposed to speculatively break true data dependencies, thereby allowing Out of Order (OOO) processors to achieve higher instruction level parallelism (ILP) and gain performance. State-of-the-art value predictors try to maximize the number of instructions that can be value predicted, with the belief that a higher coverage will unlock more ILP and increase performance. Unfortunately, this comes at increased complexity with implementations that require multiple different types of value predictors working in tandem, incurring substantial area and power cost. In this paper we motivate towards lower coverage, but focused, value prediction. Instead of aggressively increasing the coverage of value prediction, at the cost of higher area and power, we motivate refocusing value prediction as a mechanism to achieve an early execution of instructions that frequently create performance bottlenecks in the OOO processor. Since we do not aim for high coverage, our implementation is light-weight, needing just 1.2 KB of storage. Simulation results on 60 diverse workloads show that we deliver 3.3% performance gain over a baseline similar to the Intel Skylake processor. This performance gain increases substantially to 8.6% when we simulate a futuristic up-scaled version of Skylake. In contrast, for the same storage, state-of-the-art value predictors deliver a much lower speedup of 1.7% and 4.7% respectively. Notably, our proposal is similar to these predictors in performance, even when they are given nearly eight times the storage and have 60% more prediction coverage than our solution.

Recommended Reading

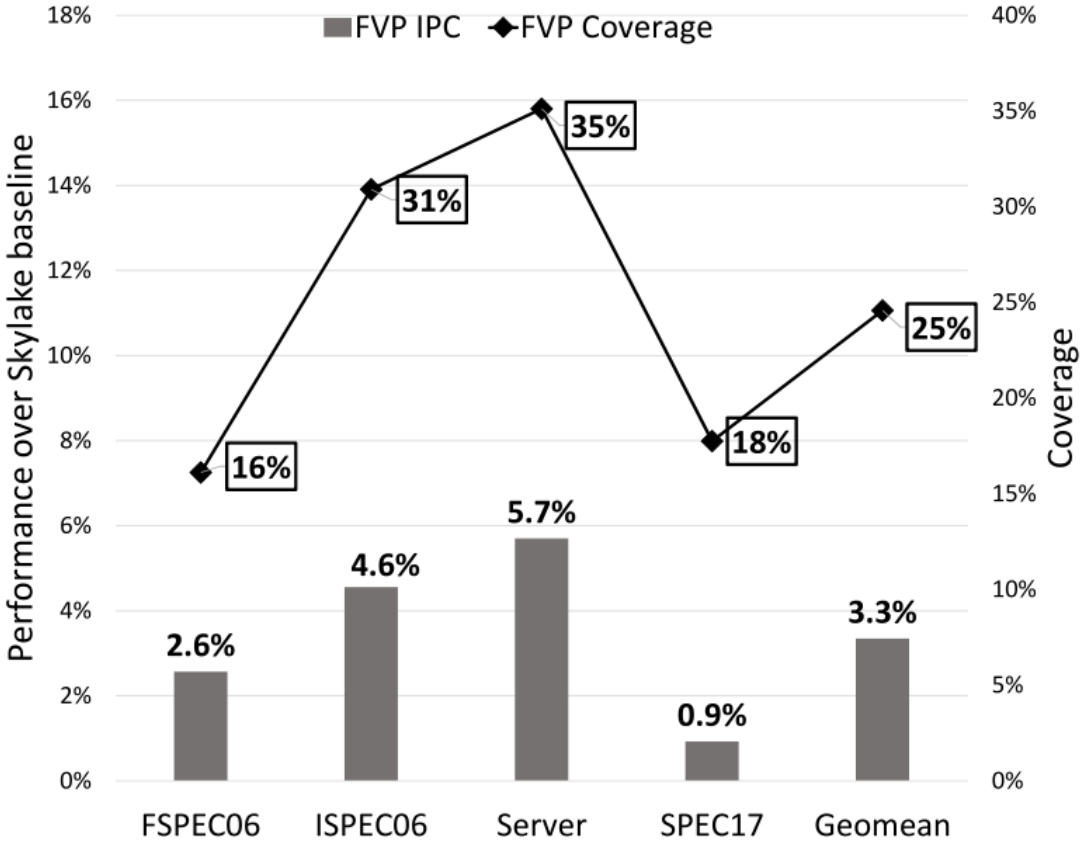


Fig. 6. Performance and Coverage of FVP on Skylake

FVP (Focused Value Prediction, proposal)

Front End	4 wide fetch and decode , TAGE/ITTAGE branch predictors [24], 20 cycles mis-prediction penalty, 64KB, 8-way L1 instruction cache, 4 wide rename into OOO with macro and micro fusion
Execution	224 ROB entries, 64 Load Queue entries, 60 Store Queue entries and 97 Issue Queue entries. 8 Execution units (ports) including 2 load ports, 3 store address ports (2 shared with load ports), 1 store-data port, 4 ALU ports, 3 FP/AVX ports, 2 branch ports. 8 wide retire and full support for bypass. Aggressive memory disambiguation predictor. Out of order load scheduling to L1
Caches	32 KB, 8-way L1 data caches with latency of 5 cycles, 256 KB 16-way L2 cache (private) with a round-trip latency of 15 cycles. 8 MB, 16 way shared LLC with data round-trip latency of 40 cycles. Aggressive multi-stream prefetching into the L2 and LLC. PC based stride prefetcher at L1
Memory	Two DDR4-2133 channels, two ranks per channel, eight banks per rank, and a data bus width per channel of 64 bits. 2 KB row buffer per bank with 15-15-15-39 (tCAS-tRCD-tRP-tRAS) timing parameters

TABLE II
CORE PARAMETERS FOR SIMULATION

Benchmarks	Category
perlbench, bzip2, gcc, mcf, h264ref, gobmk,hmmer, sjeng, libquantum, omnetpp, astar, xalancbmk	SPEC INT 2006 (ISPEC06)
bwaves, gamess, milc, zeusmp, soplex, povray, calculix, gemsfdtd, tonto, wrf, sphinx3 gromacs, cactusADM, leslie3D, namd, deall	SPEC FP 2006 (FSPEC06)
nab, cam4, pop2, roms, leela, cactubssn, xz, gcc, mcf, xalanc, exchange2, omnetpp, perlbench, bwaves, lbm, fotonik3d	SPEC17
lammps [4], hplinpack [3], tpce, spark, cassandra [1], specjbb [5], specjenterprise, hadoop [2], specpower [6]	Server

TABLE III
APPLICATIONS USED IN THIS STUDY

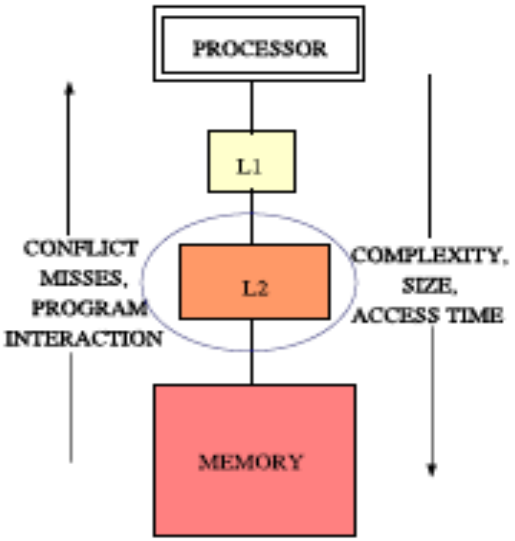
Recommended Reading

- 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



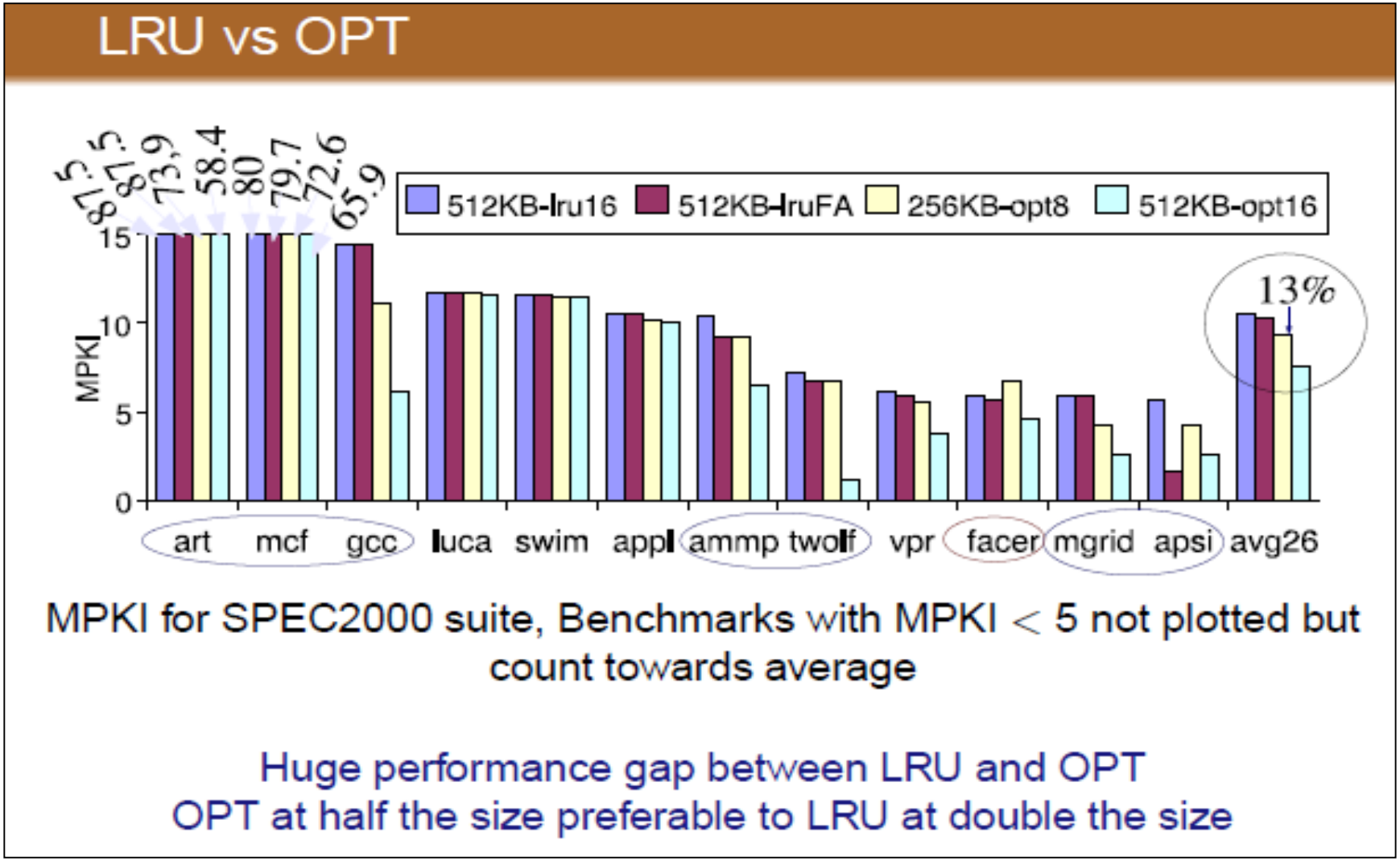
L2 and lower caches

- 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 $\Longrightarrow$ replacement policy crucial to performance
- L1 cache services temporal accesses $\Longrightarrow$ Lack of temporal accesses at L2 $\Longrightarrow$ 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



MPKI: Miss Per Kilo Instructions

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

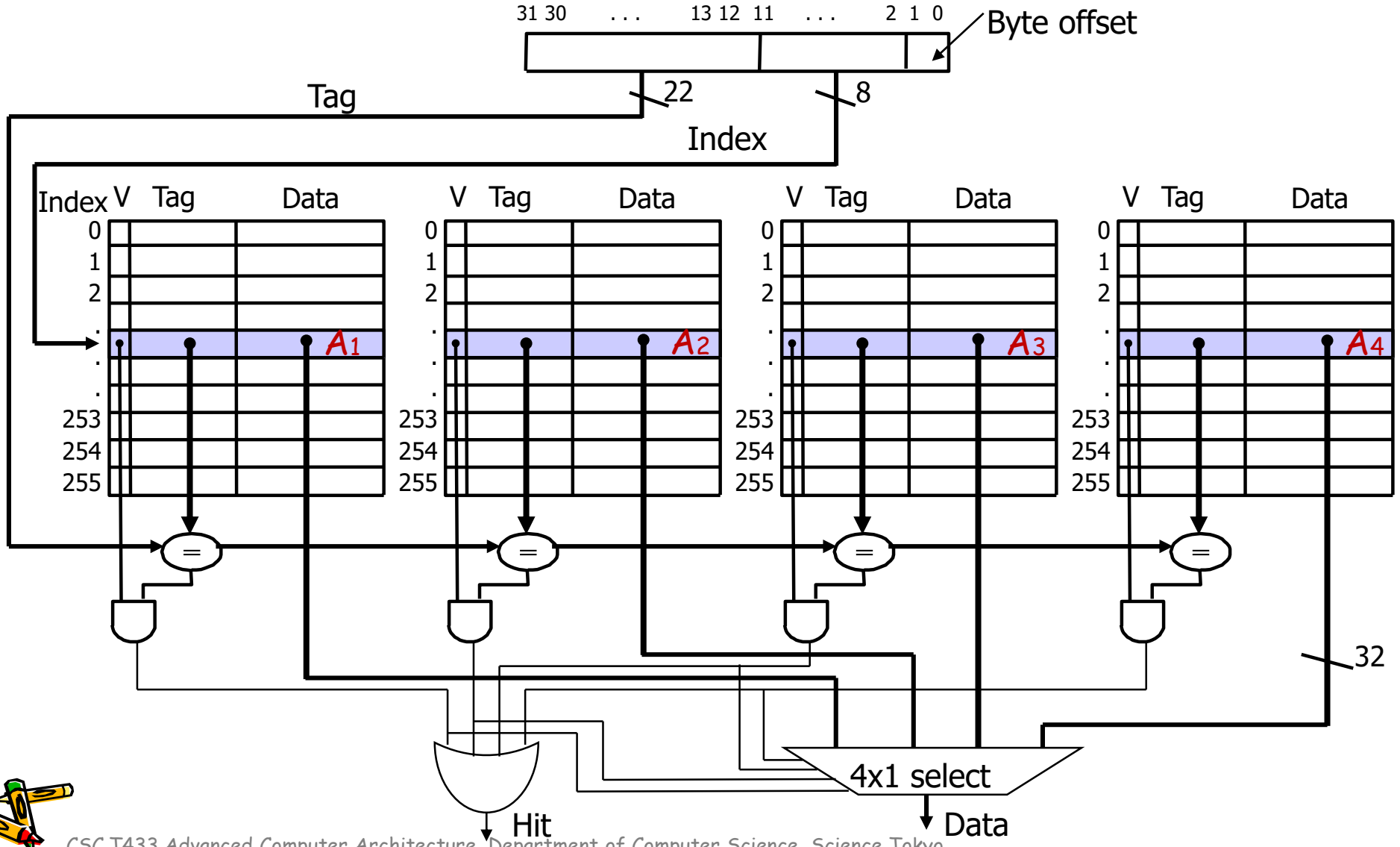
Understanding OPT

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

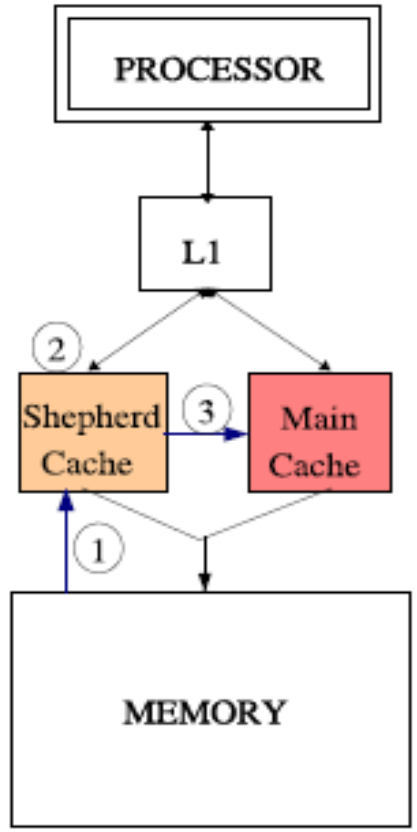
Four-Way Set Associative Cache

- One word/block, 2^8 = **256 sets** where each with four ways (each with one block)



Shepherd Cache emulation OPT

Emulating OPT with a Shepherd Cache

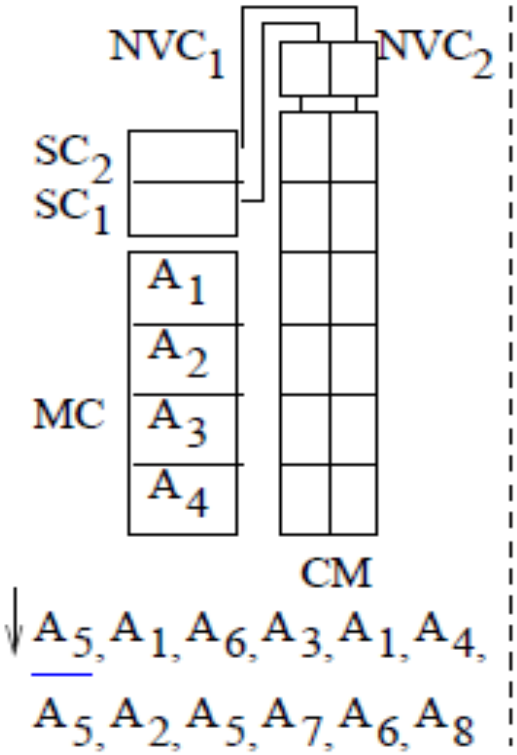


- Split the cache into two logical parts
  - Main Cache (MC) for which optimal replacement is emulated
  - Shepherd Cache (SC) used to provide a lookahead and guide replacements from MC towards OPT
- Operation
  1. Buffer lines temporarily in SC before moving them to MC, SC acts as a FIFO buffer
  2. 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

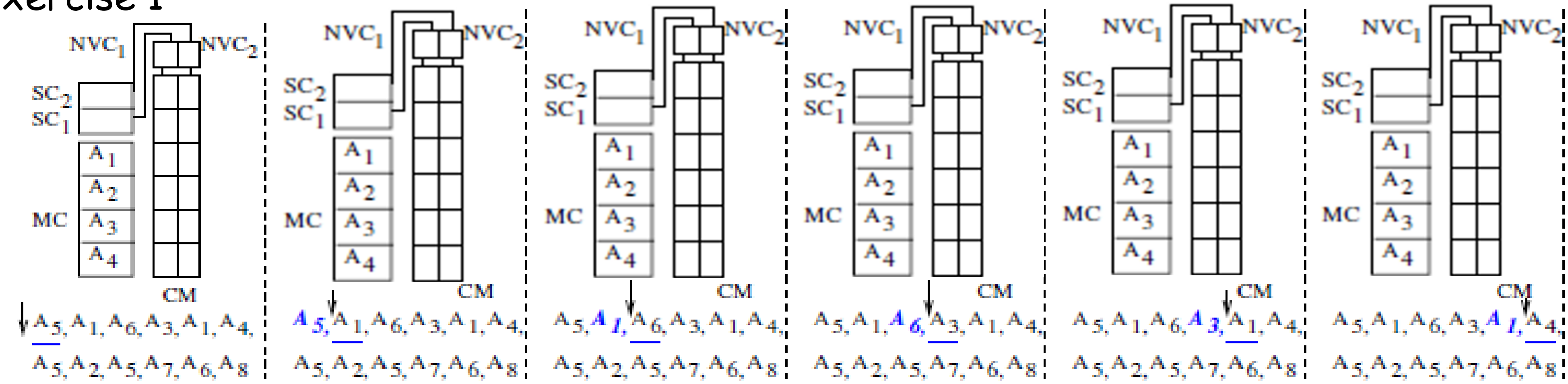
Overview of Shepherd Caching



- 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

Exercise 1



(a) Initial State

(b) A_5 inserted at SC_1

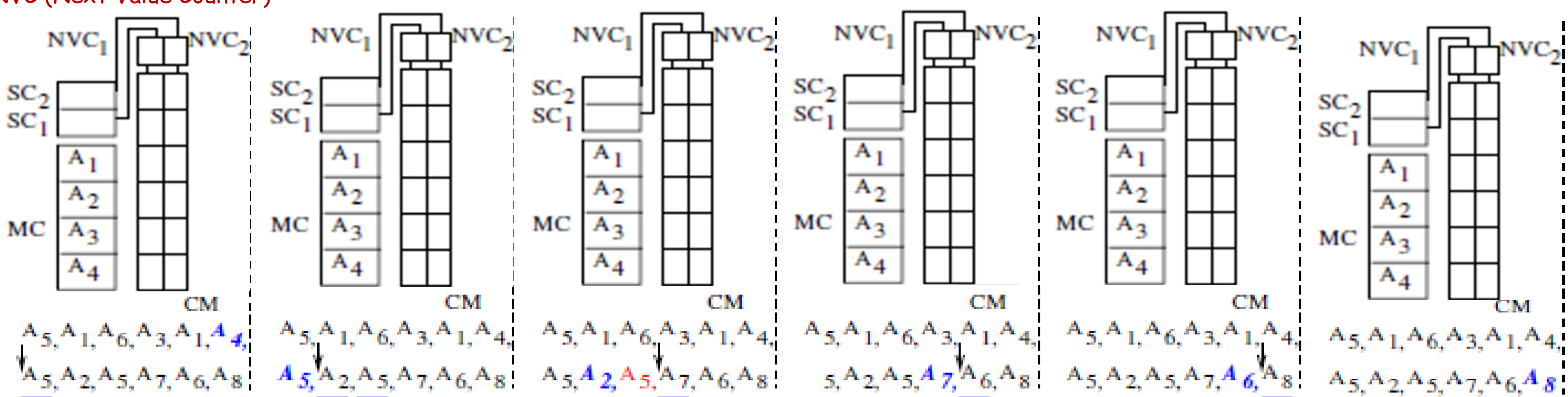
(c) A_1 added to the optimal order of SC_1

(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

MC (Main Cache)
SC (Schepherd Cache)
CM (Counter Matrix)
NVC (Next Value Counter)



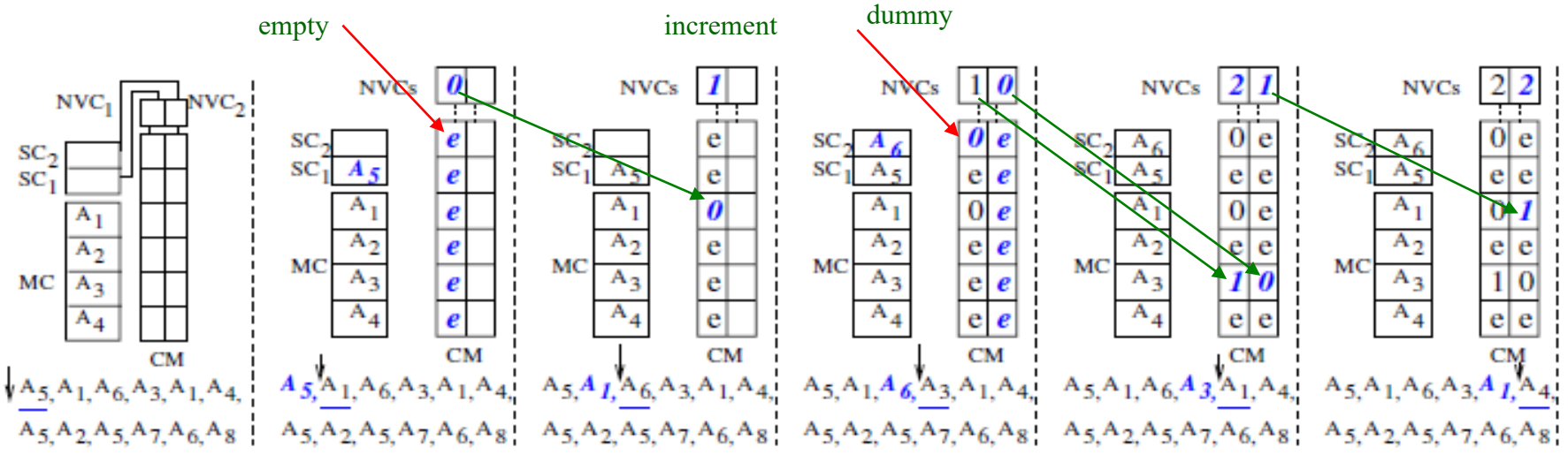
Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

A_2 added to …imal order of …$_1$,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)



(a) Initial State

(b) A_5 inserted at SC_1

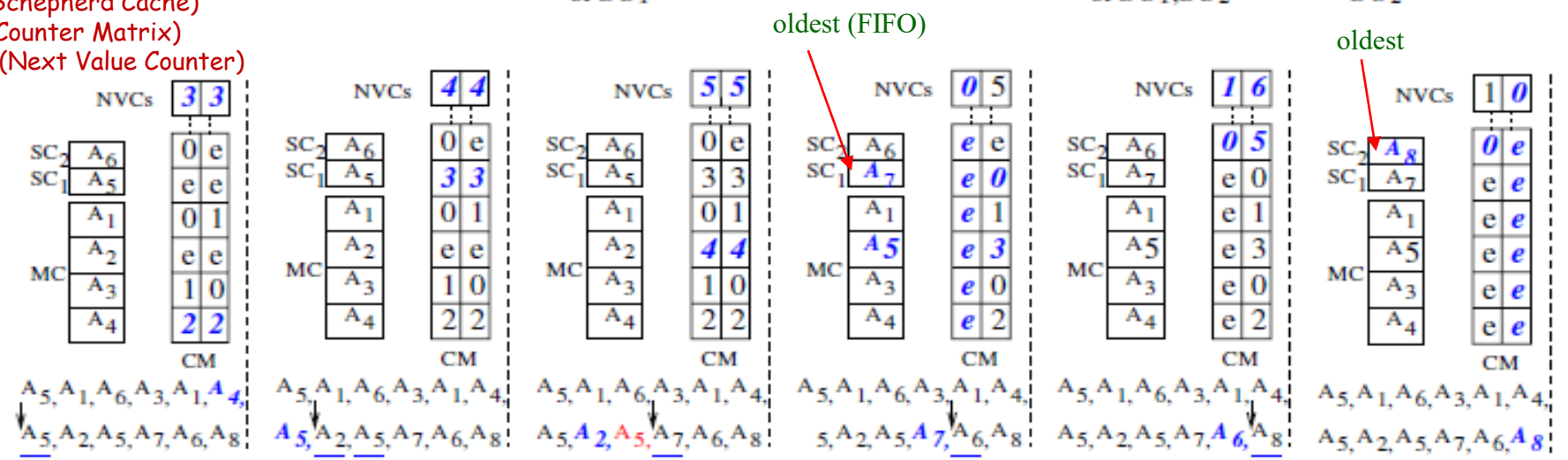
(c) A_1 added to the optimal order of SC_1

(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

MC (Main Cache)
SC (Schepherd Cache)
CM (Counter Matrix)
NVC (Next Value Counter)



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

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

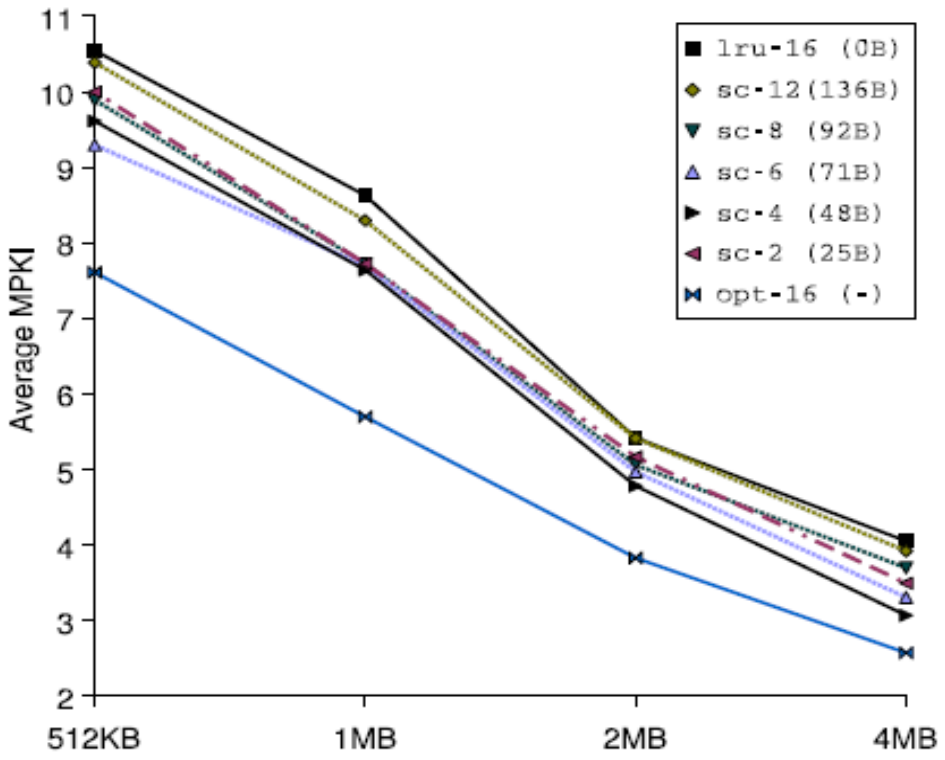
(i) A_2 added to optimal order of SC_1,SC_2

(j) from repla

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	

Shepherd cache bridges 32 – 52% of the gap

Bridging the performance gap



Avg MPKI over SPEC2000 suite

Bridging the LRU-OPT gap

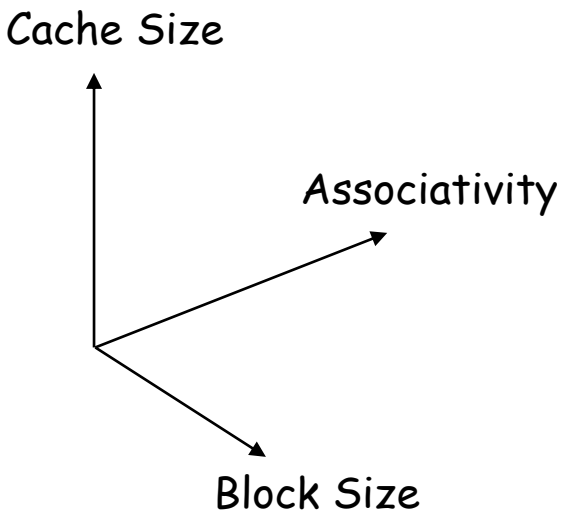
- *SC-4 bridges 32-52% of gap*
- SC moves closer to OPT as cache size increases

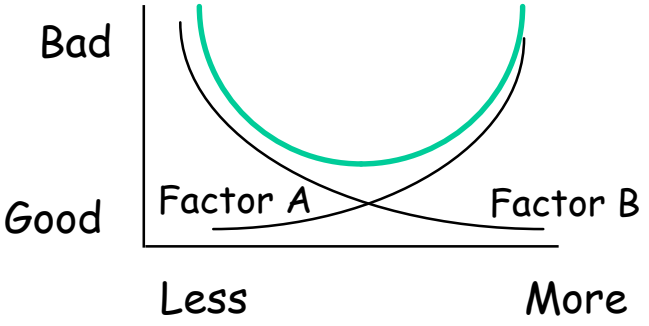
MPKI: Miss Per Kilo Instructions

Emulating Optimal Replacement with a Shepherd Cache, MICRO-2007

The Cache Design Space

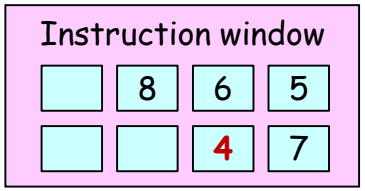
- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - I-cache, D-cache
  - depends on technology / cost
- Simplicity often wins



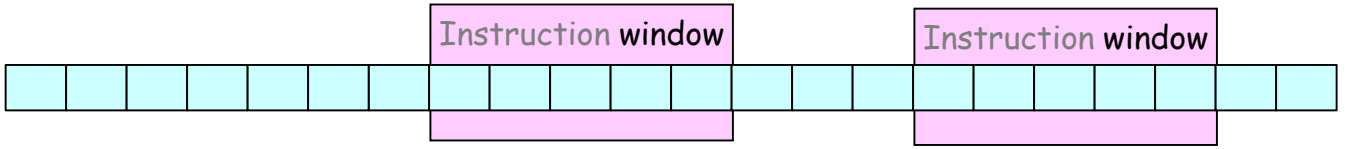


Multiprogramming

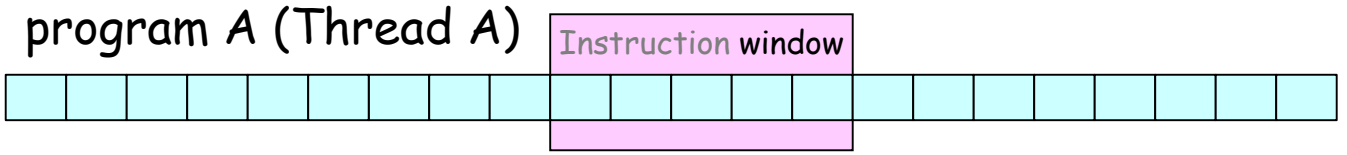
- Several independent programs run at the same time.



(c)



(d)



Multithreaded Execution Models

- During a branch miss recovery and access to the main memory by a cache miss, ALUs have no jobs to do and have to be idle.
  - interrupt, exception, or OS call
- Executing multiple independent threads (programs) will mitigate the overhead.
- They are called coarse-grained and fine-grained multithreaded processors having multiple architecture states.
- Simultaneous Multithreading (SMT) can improve hardware resource usage.

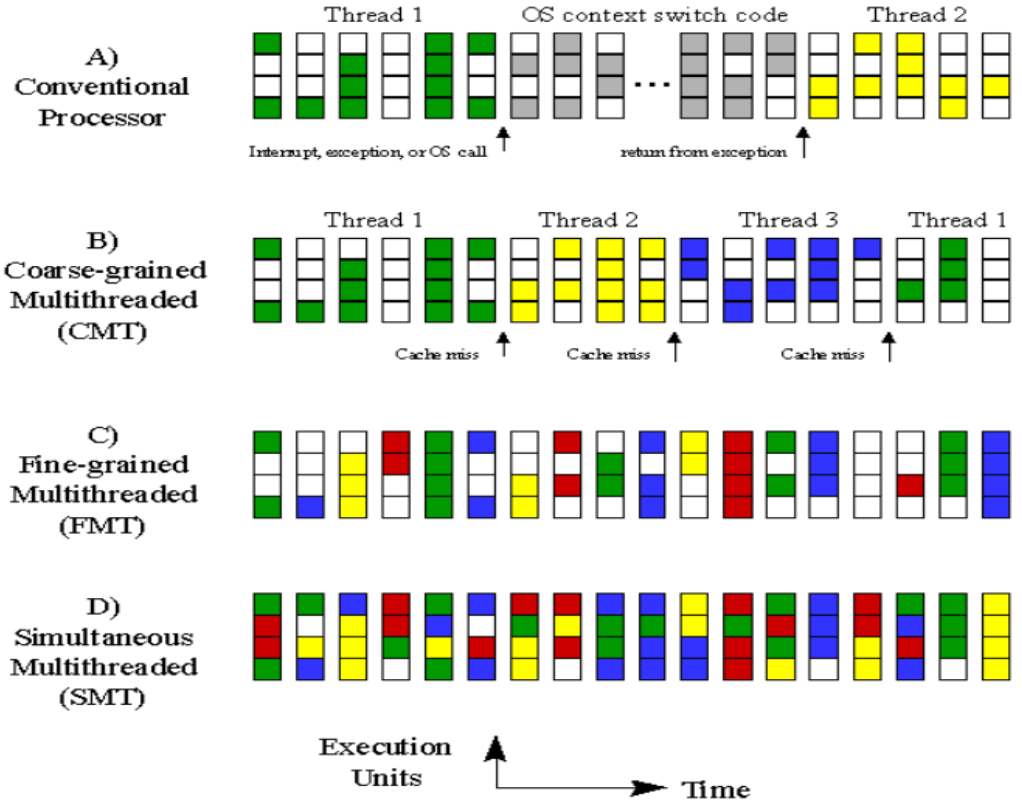
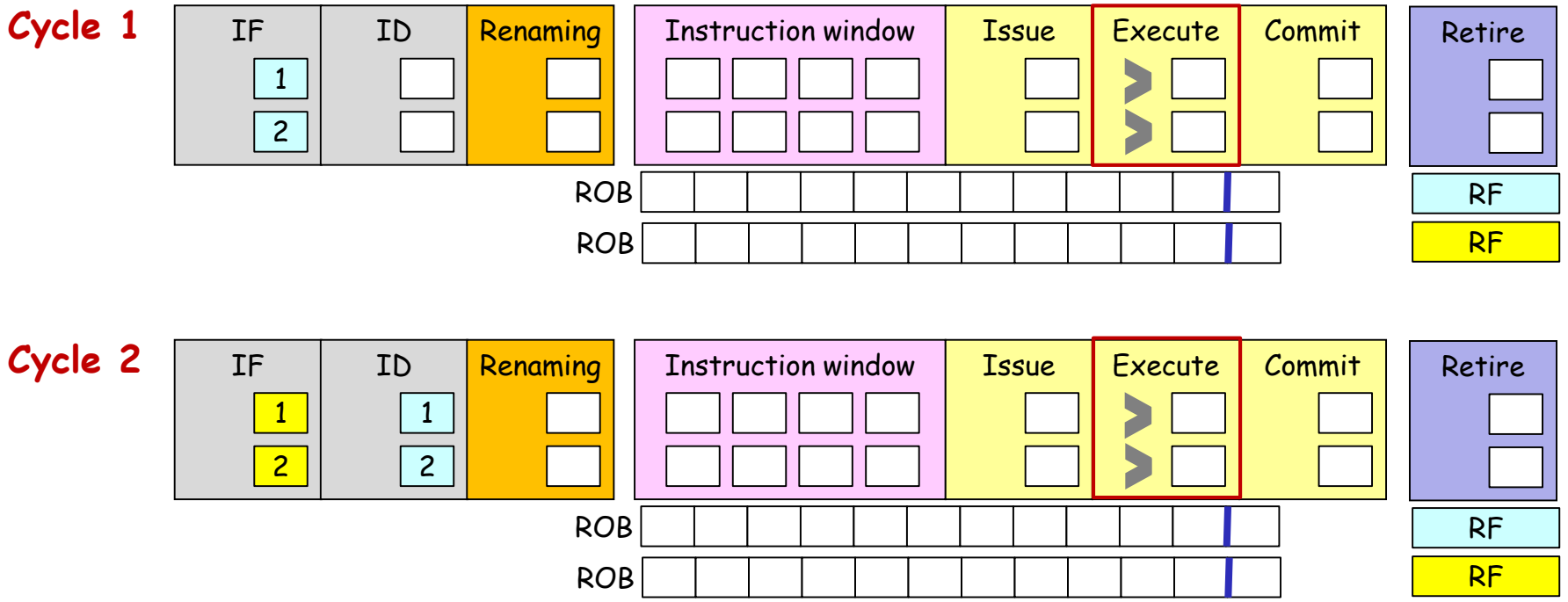
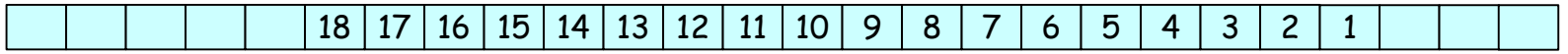


Figure 1. Multithreaded Execution with Increasing Levels of TLP Hardware Support

Simultaneous multithreading (SMT)

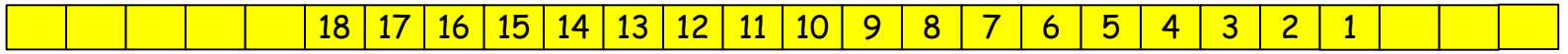


Instructions to be executed of program A



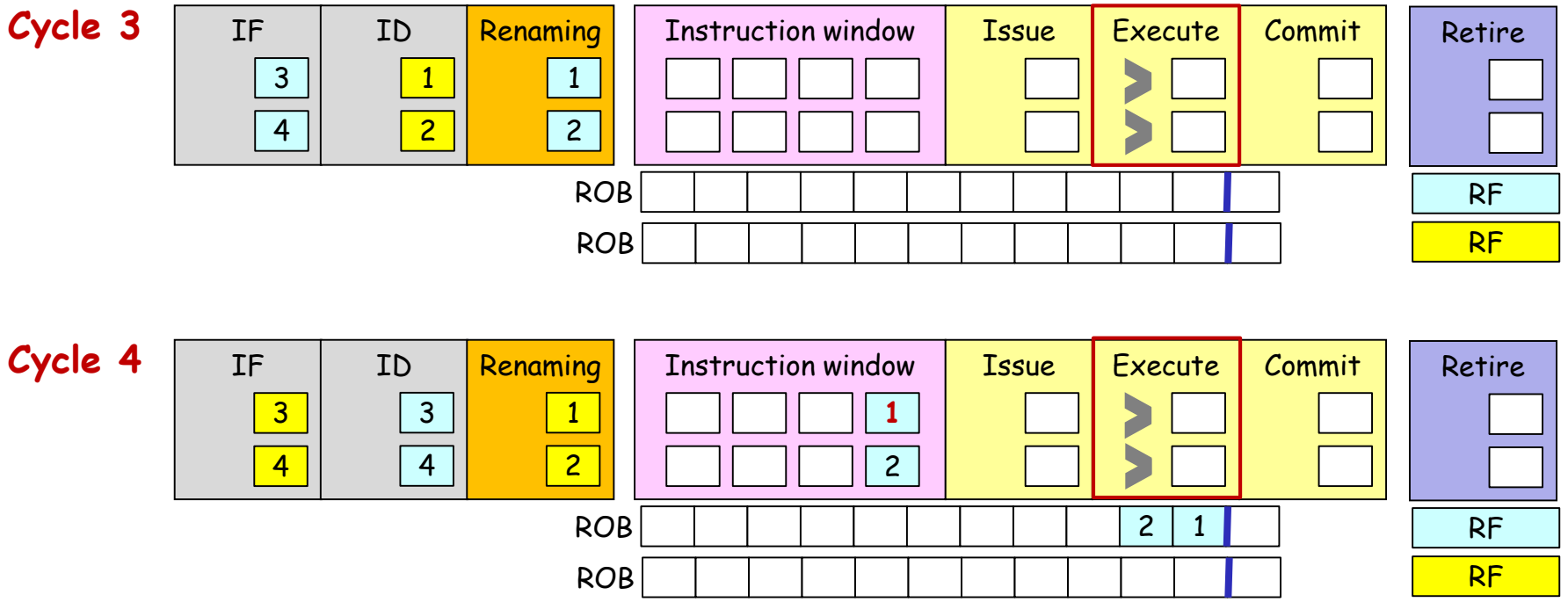
Newer instructions

Instructions to be executed of program B

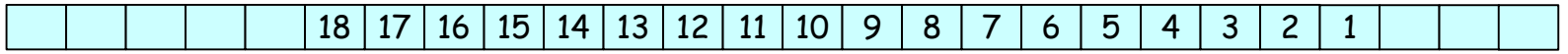


Newer instructions

Simultaneous multithreading (SMT)

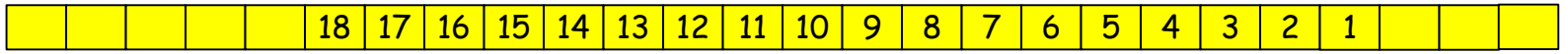


Instructions to be executed of program A



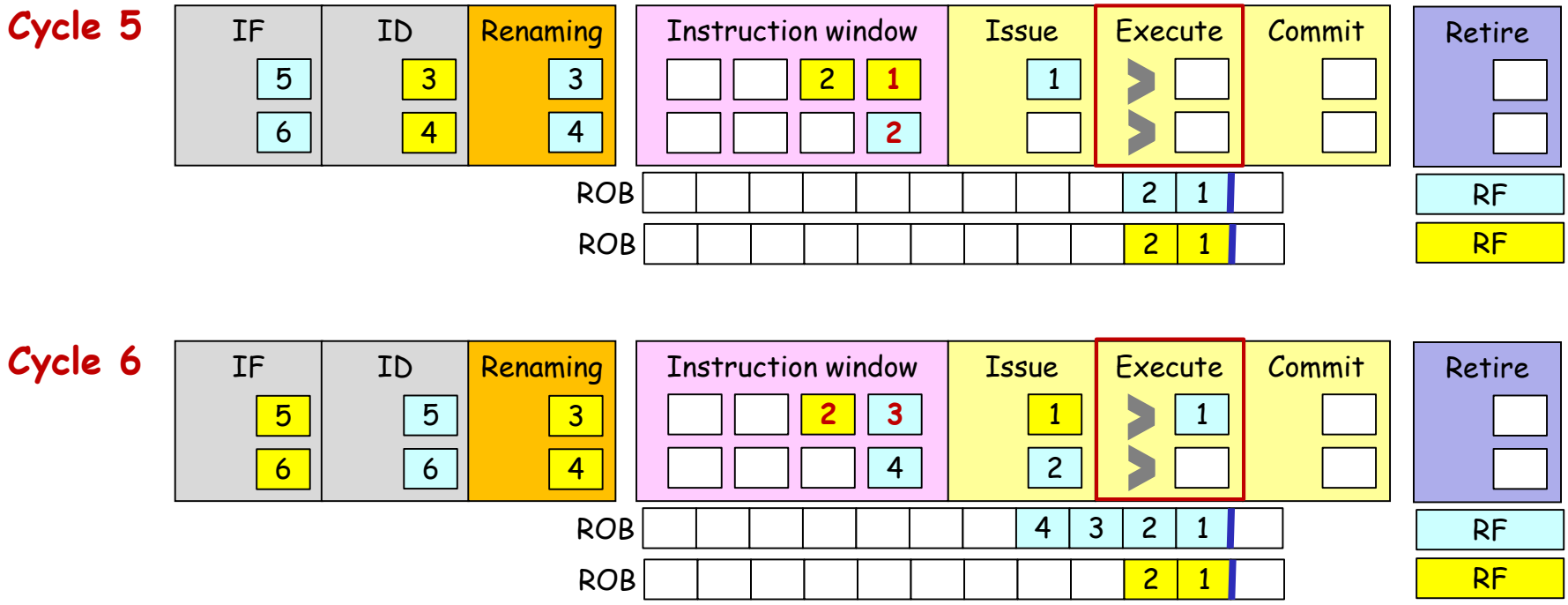
Newer instructions

Instructions to be executed of program B

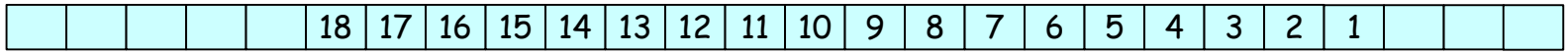


Newer instructions

Simultaneous multithreading (SMT)

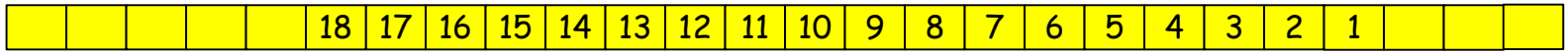


Instructions to be executed of program A



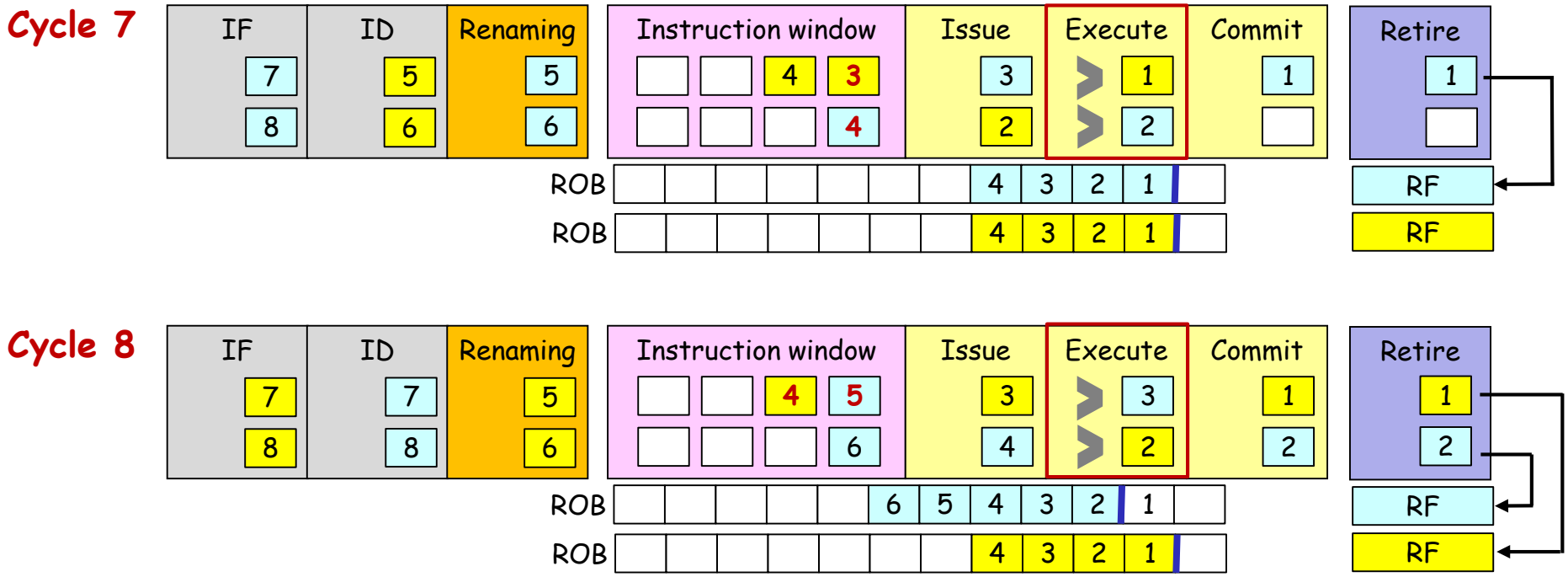
Newer instructions

Instructions to be executed of program B

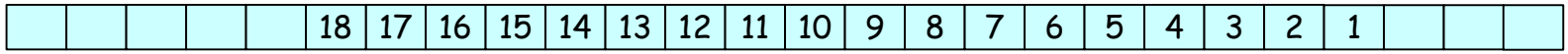


Newer instructions

Simultaneous multithreading (SMT)

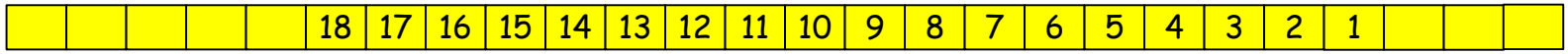


Instructions to be executed of program A



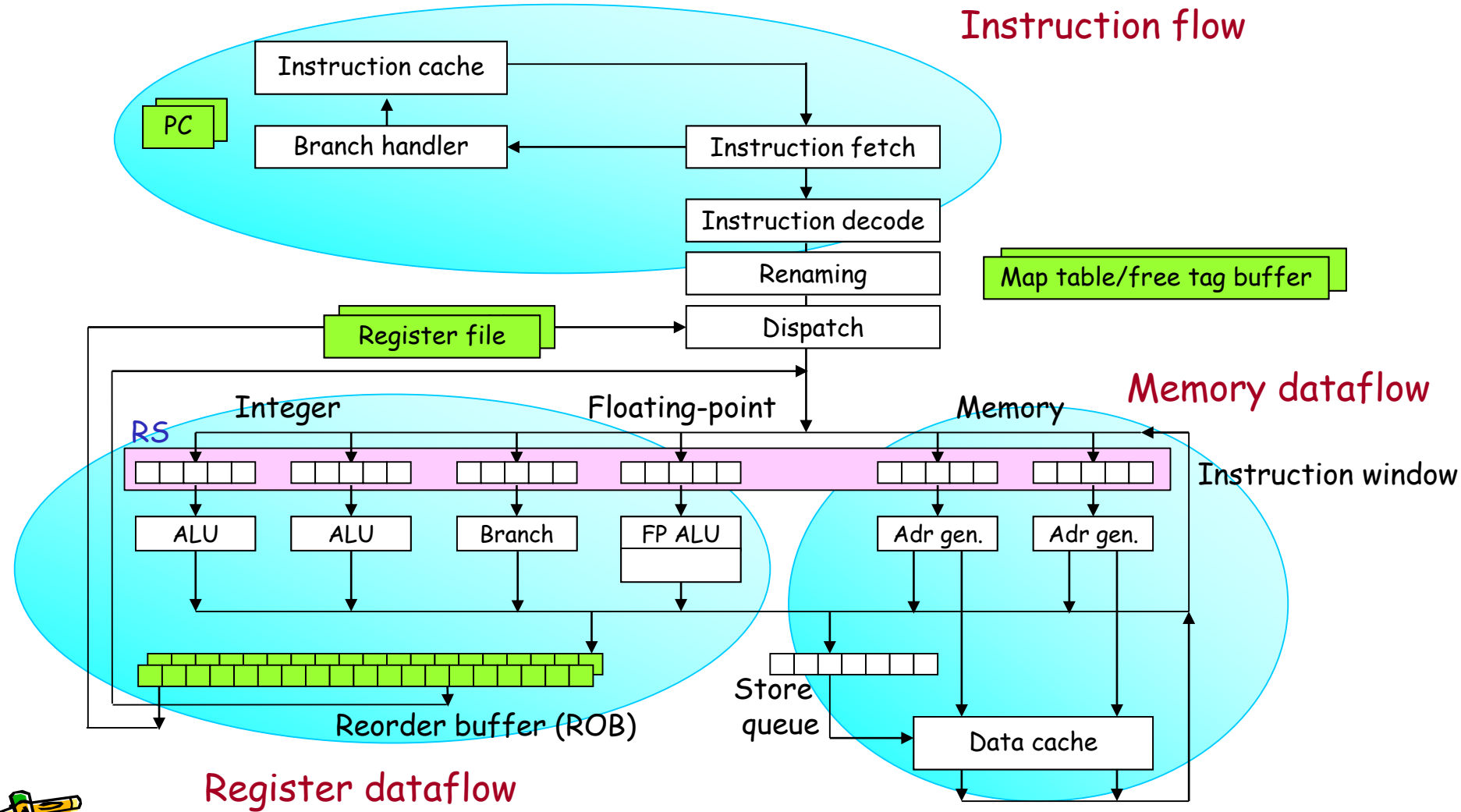
Newer instructions

Instructions to be executed of program B



Newer instructions

Datapath of SMT OoO execution processor



From multi-core era to many-core era

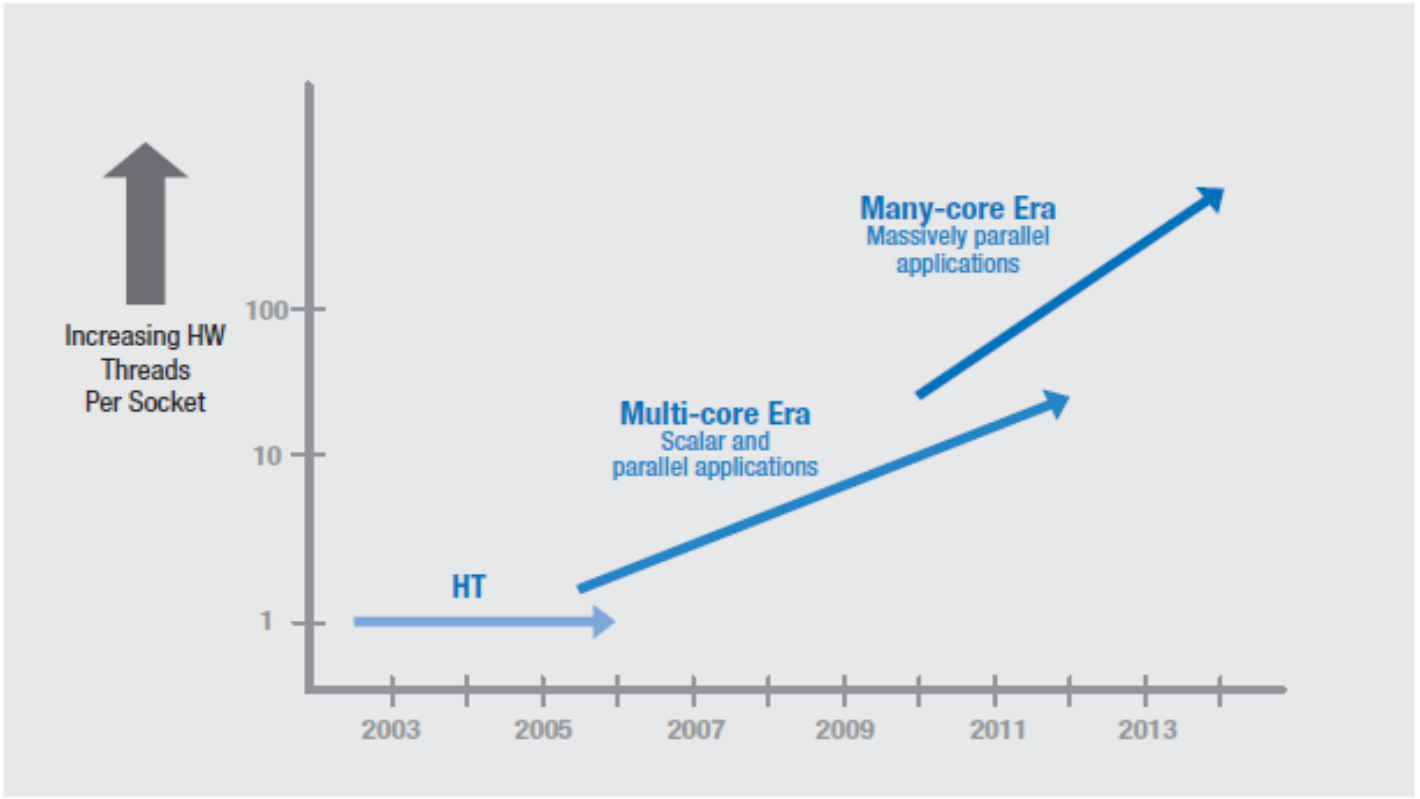


Figure 1: Current and expected eras of Intel® processor architectures

Platform 2015: Intel® Processor and Platform Evolution for the Next Decade, 2005