Lecture: SMT, Cache Hierarchies

- Topics: memory dependence wrap-up, SMT processors, cache access basics and innovations (Sections B.1-B.3, 2.1)

Problem 0

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume memory dependence prediction, with a default prediction that there is no dependence.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd		
LD R3 ← [R4]	6		adde		
ST R5 → [R6]	4	7	abba		
LD R7 ← [R8]	2		abce		
ST R9 → [R10]	8	3	abba		
LD R11 ← [R12]	1		abba		

Problem 0

- Consider the following LSQ and when operands are available. Estimate when the address calculation and memory accesses happen for each ld/st. Assume memory dependence prediction, with a default prediction that there is no dependence.

	Ad. Op	St. Op	Ad.Val	Ad.Cal	Mem.Acc
LD R1 ← [R2]	3		abcd	4	5
LD R3 ← [R4]	6		adde	7	8
ST R5 → [R6]	4	7	abba	5	commit
LD R7 ← [R8]	2		abce	3	4
ST R9 → [R10]	8	3	abba	9	commit
LD R11 ← [R12]	1		abba	2	3/10

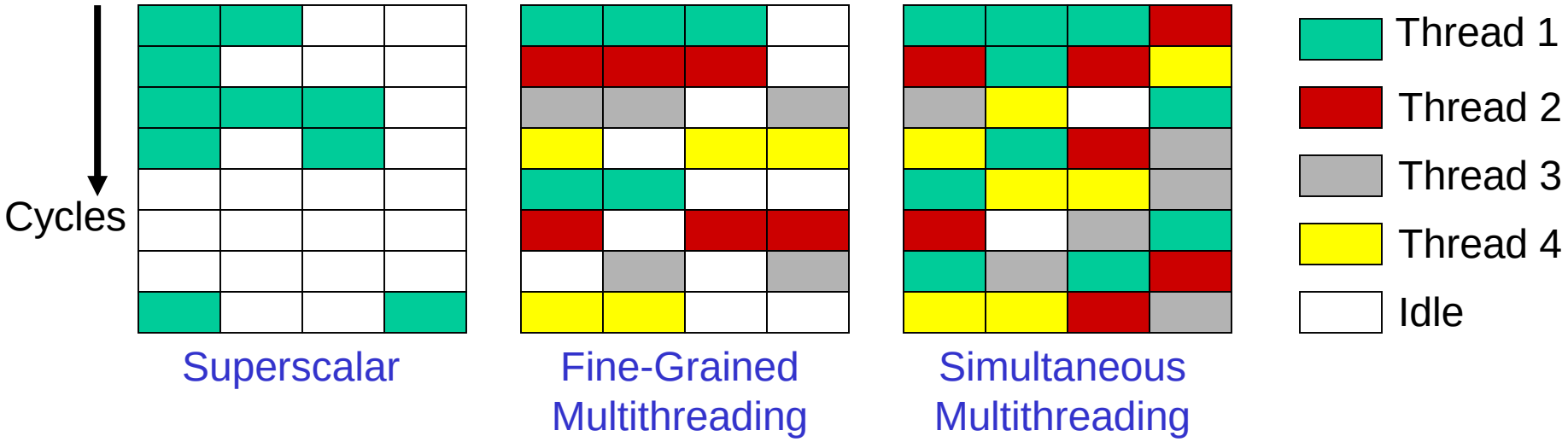
Thread-Level Parallelism

- Motivation:
  - a single thread leaves a processor under-utilized for most of the time
  - by doubling processor area, single thread performance barely improves

- Strategies for thread-level parallelism:
  - multiple threads share the same large processor → reduces under-utilization, efficient resource allocation
    Simultaneous Multi-Threading (SMT)
  - each thread executes on its own mini processor → simple design, low interference between threads
    Chip Multi-Processing (CMP) or multi-core

How are Resources Shared?

Each box represents an issue slot for a functional unit. Peak thruput is 4 IPC.

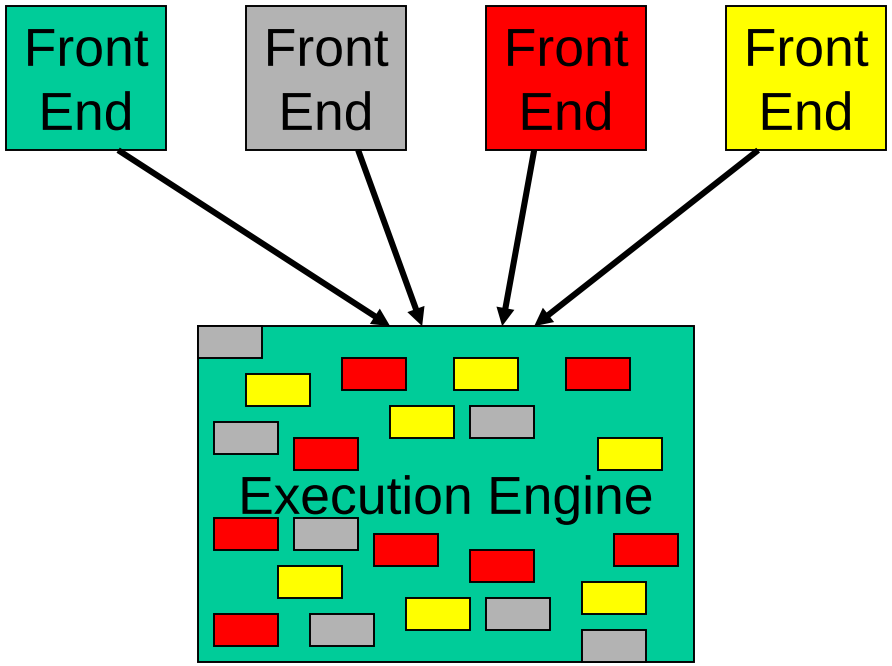


- Superscalar processor has high under-utilization – not enough work every cycle, especially when there is a cache miss
- Fine-grained multithreading can only issue instructions from a single thread in a cycle – can not find max work every cycle, but cache misses can be tolerated
- Simultaneous multithreading can issue instructions from any thread every cycle – has the highest probability of finding work for every issue slot

What Resources are Shared?

- Multiple threads are simultaneously active (in other words, a new thread can start without a context switch)
- For correctness, each thread needs its own PC, IFQ, logical regs (and its own mappings from logical to phys regs)
- For performance, each thread could have its own ROB/LSQ (so that a stall in one thread does not stall commit in other threads), I-cache, branch predictor, D-cache, etc. (for low interference), although note that more sharing → better utilization of resources
- Each additional thread costs a PC, IFQ, rename tables, and ROB – cheap!

Pipeline Structure



Private/ Shared Front-end

I-Cache

Bpred

Private Front-end

Rename

ROB

Shared Exec Engine

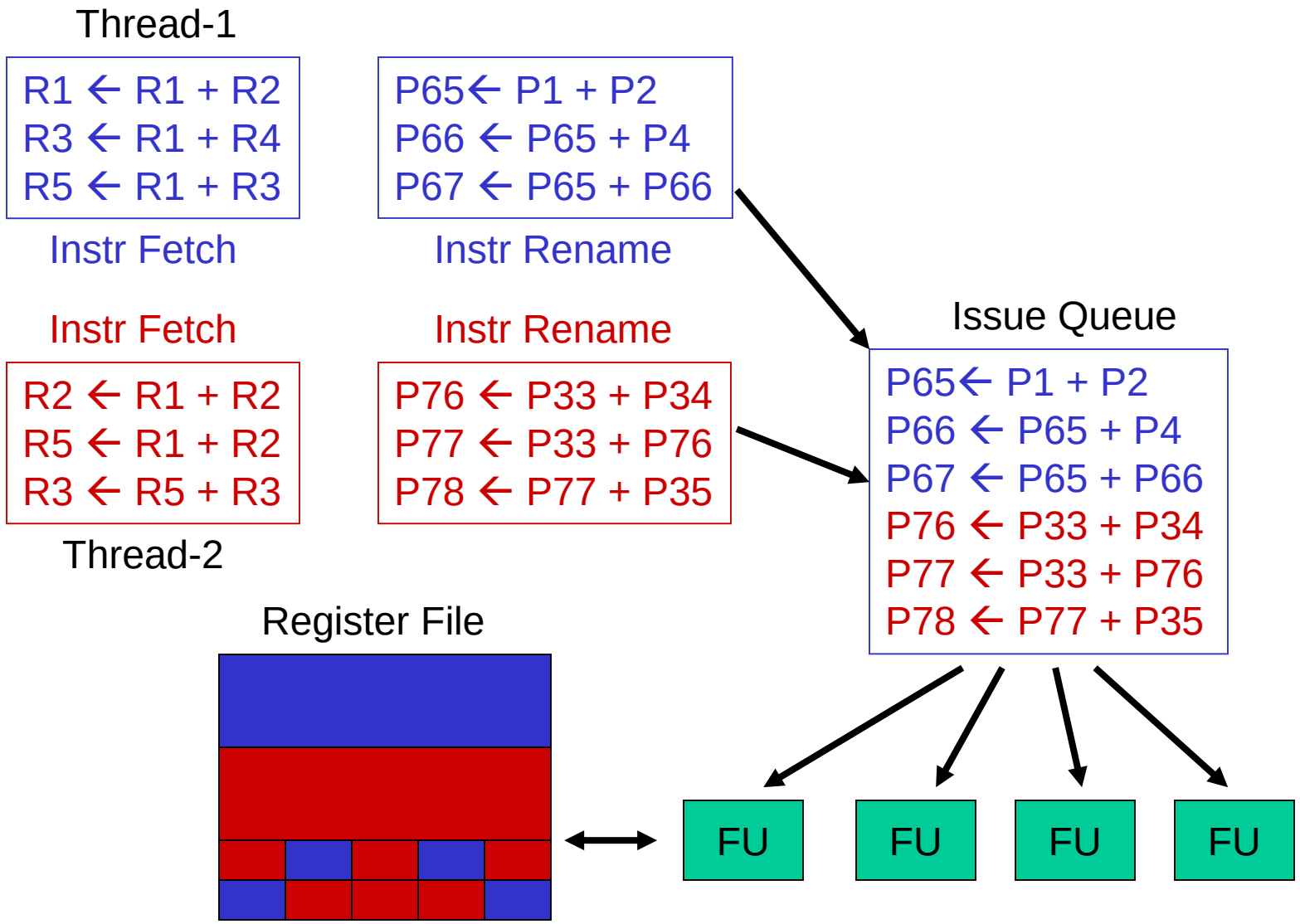
Regs

IQ

DCache

FUs

Resource Sharing



Performance Implications of SMT

- Single thread performance is likely to go down (caches, branch predictors, registers, etc. are shared) – this effect can be mitigated by trying to prioritize one thread

- While fetching instructions, thread priority can dramatically influence total throughput – a widely accepted heuristic (ICOUNT): fetch such that each thread has an equal share of processor resources

- With eight threads in a processor with many resources, SMT yields throughput improvements of roughly 2-4

Pentium4 Hyper-Threading

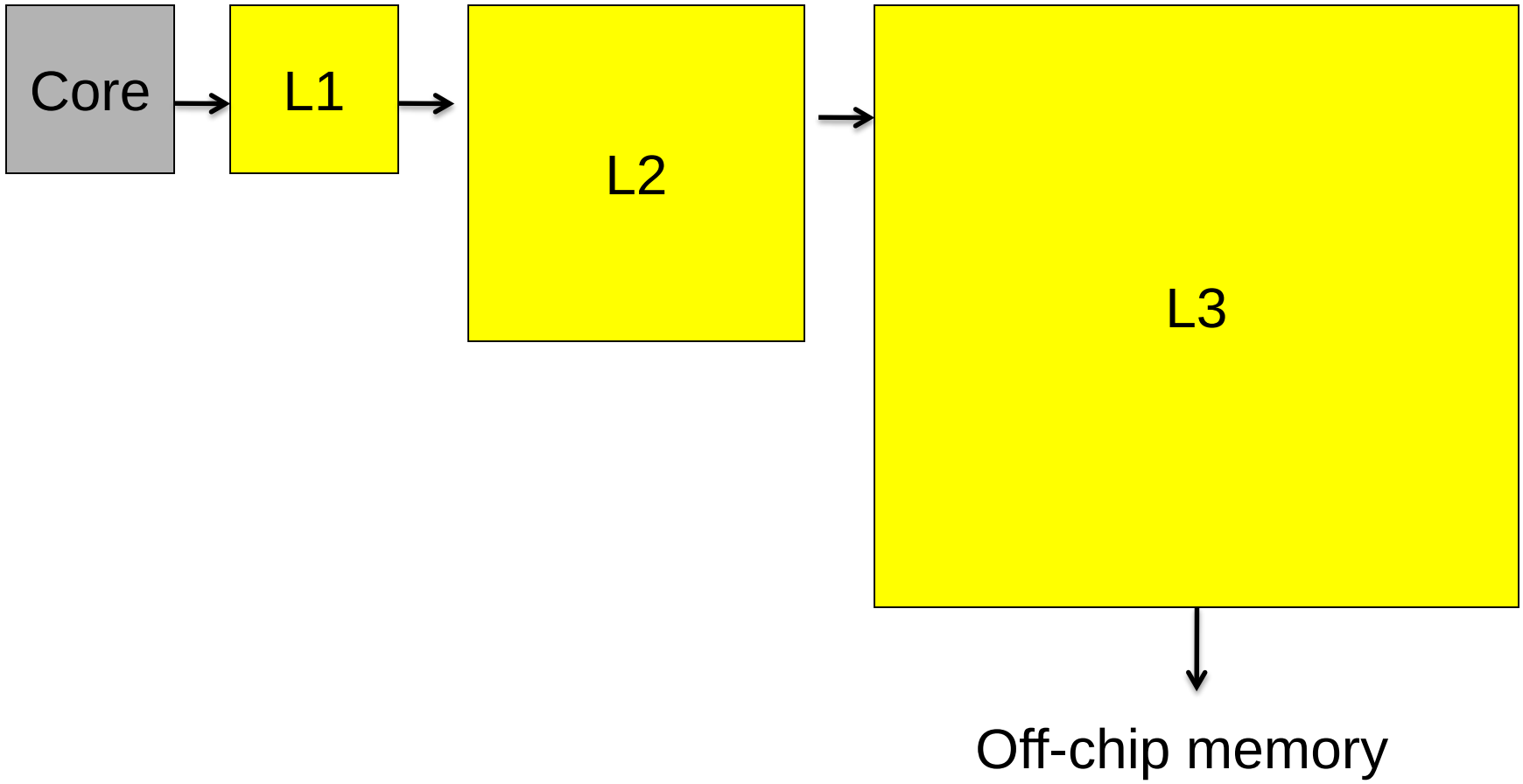
- Two threads – the Linux operating system operates as if it is executing on a two-processor system
- When there is only one available thread, it behaves like a regular single-threaded superscalar processor
- Statically divided resources: ROB, LSQ, issueq -- a slow thread will not cripple thruput (might not scale)
- Dynamically shared: trace cache and decode (fine-grained multi-threaded, round-robin), FUs, data cache, bpred

Multi-Programmed Speedup

Benchmark	Best Speedup	Worst Speedup	Avg Speedup
gzip	1.48	1.14	1.24
vpr	1.43	1.04	1.17
gcc	1.44	1.00	1.11
mcf	1.57	1.01	1.21
crafty	1.40	0.99	1.17
parser	1.44	1.09	1.18
eon	1.42	1.07	1.25
perlbmk	1.40	1.07	1.20
gap	1.43	1.17	1.25
vortex	1.41	1.01	1.13
bzip2	1.47	1.15	1.24
twolf	1.48	1.02	1.16
wupwise	1.33	1.12	1.24
swim	1.58	0.90	1.13
mgrid	1.28	0.94	1.10
applu	1.37	1.02	1.16
mesa	1.39	1.11	1.22
galgel	1.47	1.05	1.25
art	1.55	0.90	1.13
equake	1.48	1.02	1.21
facerec	1.39	1.16	1.25
ammp	1.40	1.09	1.21
lucas	1.36	0.97	1.13
fma3d	1.34	1.13	1.20
sixtrack	1.58	1.28	1.42
apsi	1.40	1.14	1.23
Overall	1.58	0.90	1.20

- sixtrack and eon do not degrade their partners (small working sets?)
- swim and art degrade their partners (cache contention?)
- Best combination: swim & sixtrack
  worst combination: swim & art
- Static partitioning ensures low interference – worst slowdown is 0.9

The Cache Hierarchy



Problem 1

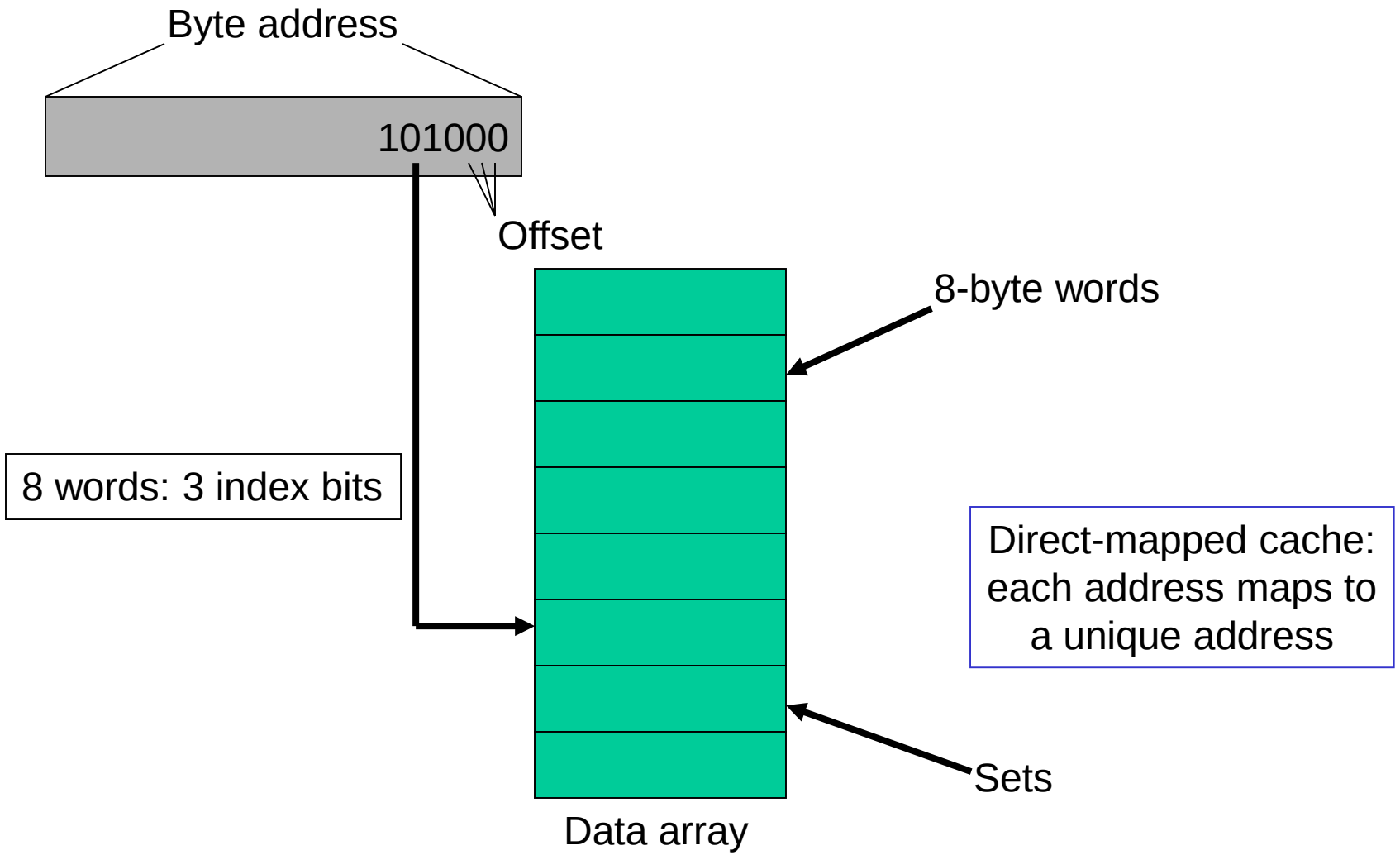
- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?

Problem 1

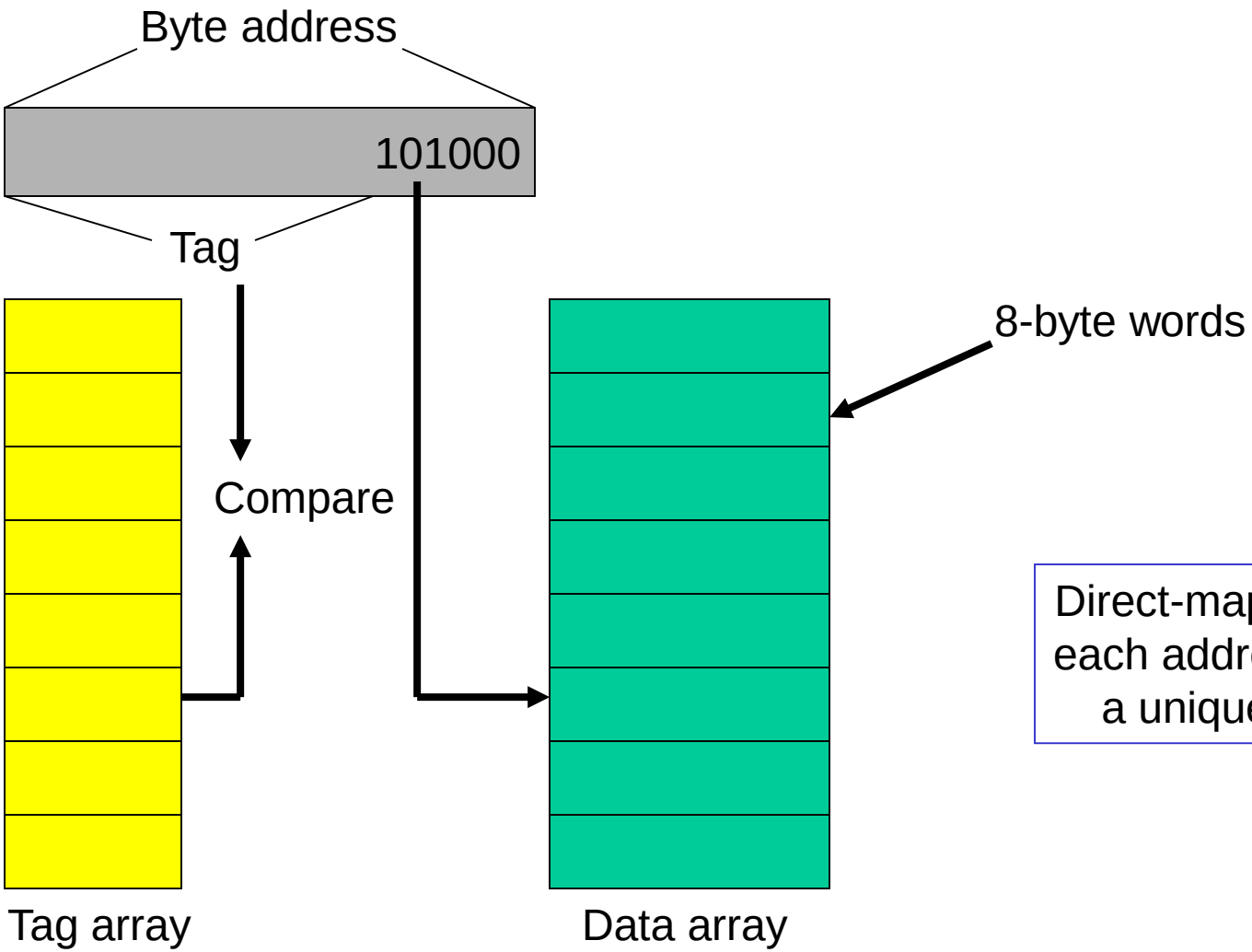
- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?

With L3: 1000 + 10x20 + 30x10 + 300x5 = 3000
Without L3: 1000 + 10x20 + 10x300 = 4200

Accessing the Cache

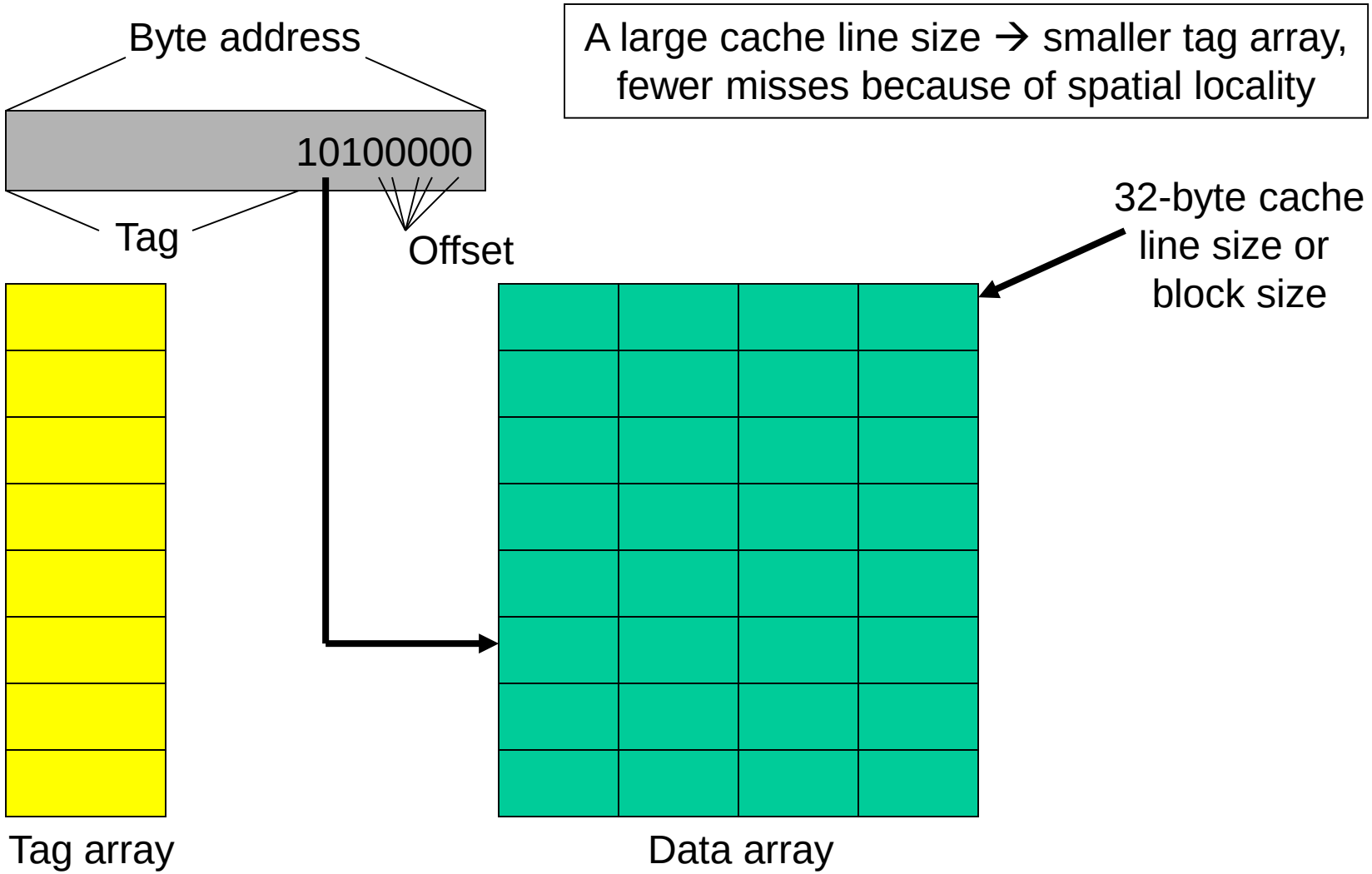


The Tag Array

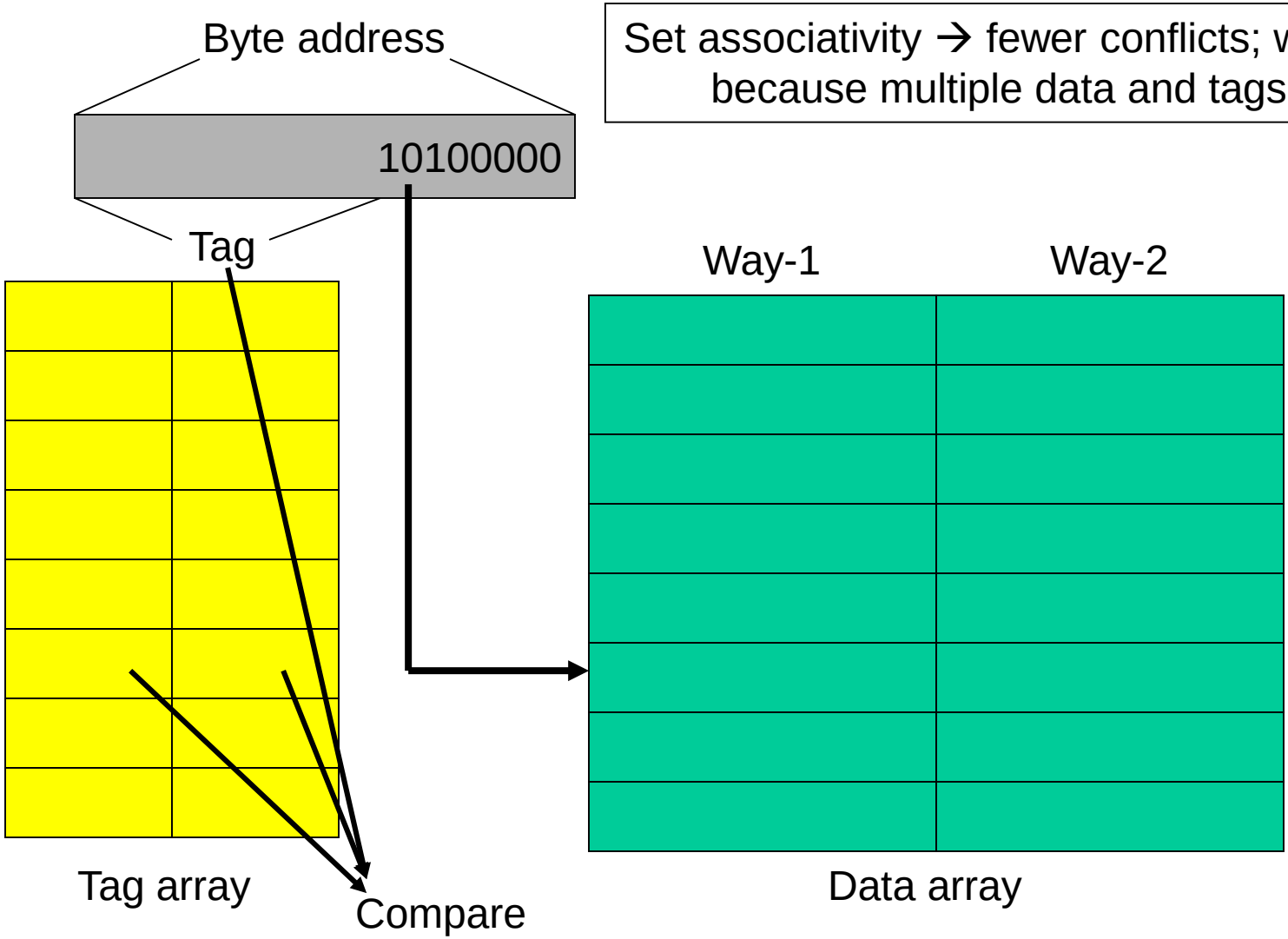


Direct-mapped cache: each address maps to a unique address

Increasing Line Size



Associativity



Set associativity → fewer conflicts; wasted power because multiple data and tags are read

Problem 2

- Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 0, and so on. For the following access pattern, estimate the hits and misses:

  A B B E C C A D B F A E G C G A

Problem 2

- Assume a direct-mapped cache with just 4 sets. Assume that block A maps to set 0, B to 1, C to 2, D to 3, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH MM H MM HM HMM M M M

Problem 3

- Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

  A B B E C C A D B F A E G C G A

Problem 3

- Assume a 2-way set-associative cache with just 2 sets. Assume that block A maps to set 0, B to 1, C to 0, D to 1, E to 0, and so on. For the following access pattern, estimate the hits and misses:

A B B E C C A D B F A E G C G A
M MH M MH MM HM HMM M H M

Problem 4

- 64 KB 16-way set-associative data cache array with 64 byte line sizes, assume a 40-bit address
- How many sets?
- How many index bits, offset bits, tag bits?
- How large is the tag array?

Problem 4

- 64 KB 16-way set-associative data cache array with 64 byte line sizes, assume a 40-bit address
- How many sets? 64
- How many index bits (6), offset bits (6), tag bits (28)?
- How large is the tag array (28 Kb)?

Title

- Bullet