

Lecture 11: SMT and Caching Basics

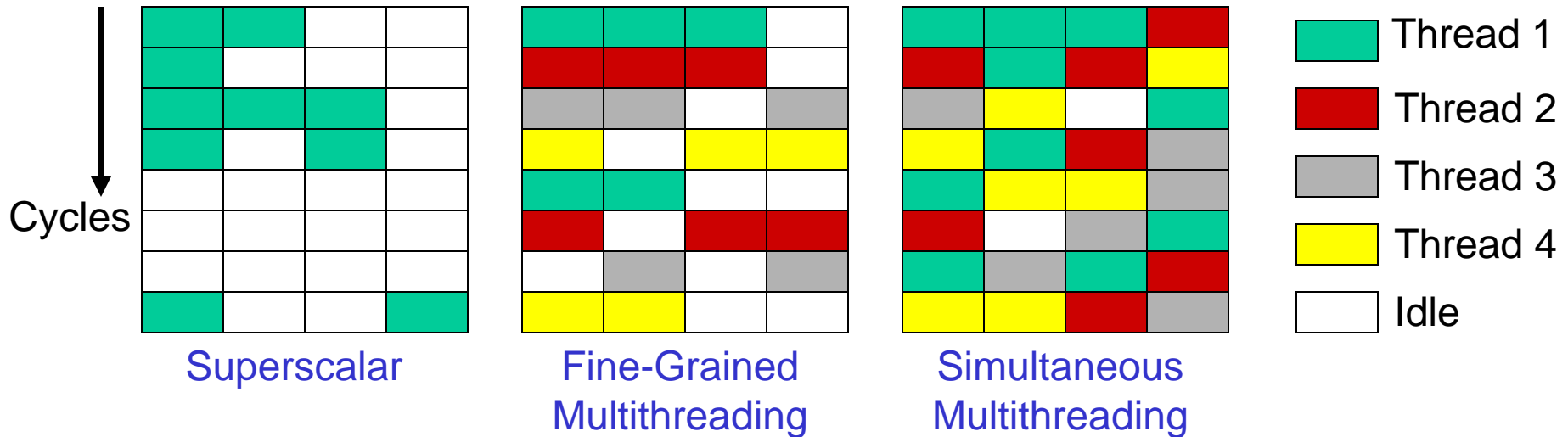
- Today: SMT, cache access basics
(Sections 3.5, 5.1)

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)

How are Resources Shared?

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

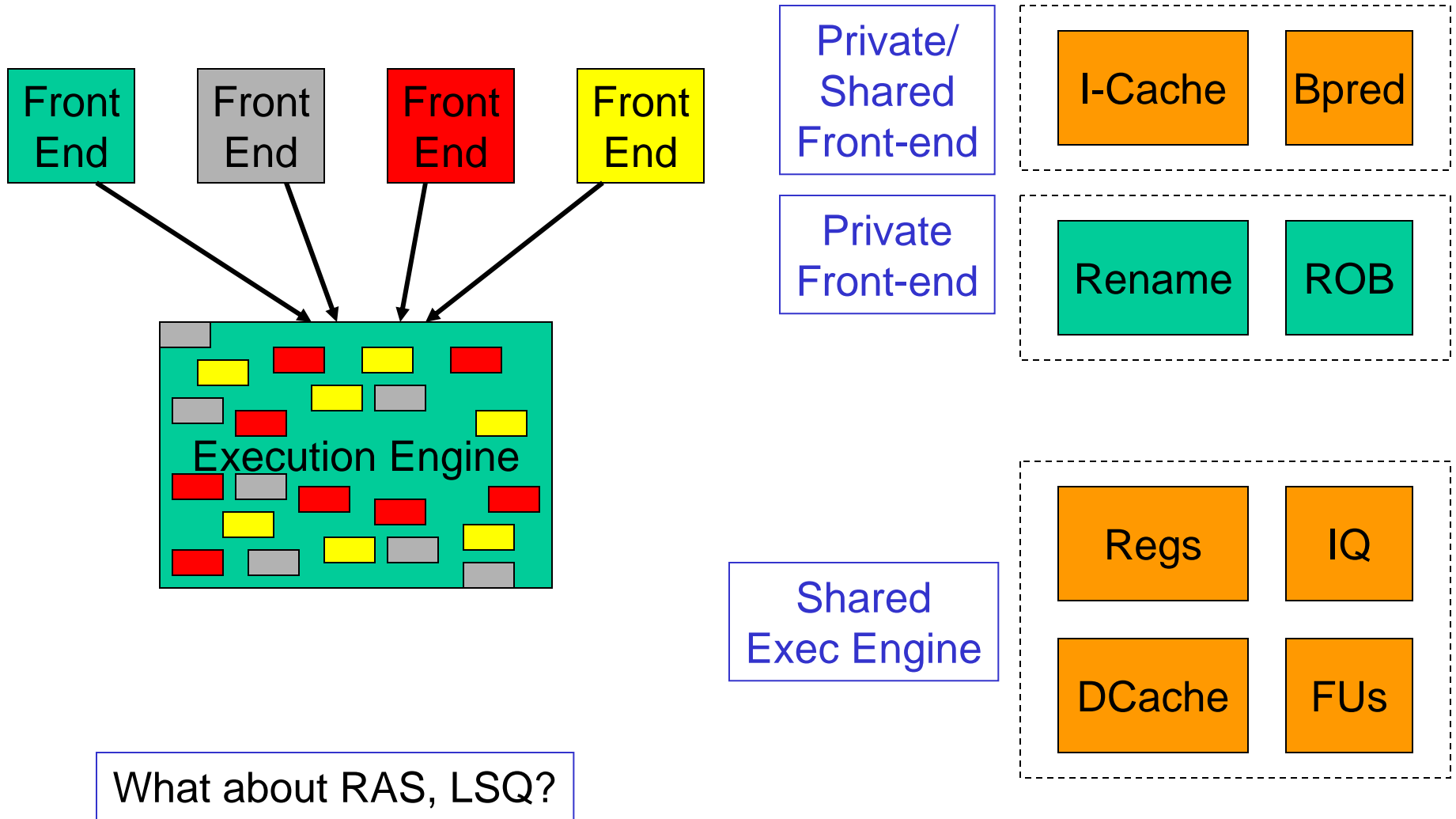


- Superscalar processor has high under-utilization – not enough work every cycle, especially when there is a cache miss
- Fine-grained multithreading can only issue instructions from a single thread in a cycle – can not find max work every cycle, but cache misses can be tolerated
- Simultaneous multithreading 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, its own logical regs (and its own mapping from logical to phys regs)
- For performance, each thread could have its own ROB (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, rename table, and ROB – cheap!

Pipeline Structure



Resource Sharing

Thread-1

$R1 \leftarrow R1 + R2$
 $R3 \leftarrow R1 + R4$
 $R5 \leftarrow R1 + R3$

Instr Fetch

$P73 \leftarrow P1 + P2$
 $P74 \leftarrow P73 + P4$
 $P75 \leftarrow P73 + P74$

Instr Rename

Instr Fetch

$R2 \leftarrow R1 + R2$
 $R5 \leftarrow R1 + R2$
 $R3 \leftarrow R5 + R3$

Thread-2

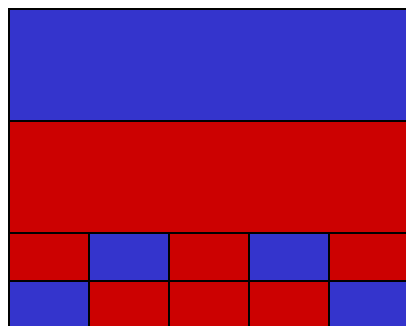
Instr Rename

$P76 \leftarrow P33 + P34$
 $P77 \leftarrow P33 + P76$
 $P78 \leftarrow P77 + P35$

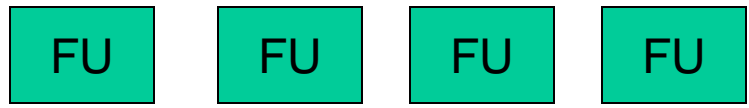
Issue Queue

$P73 \leftarrow P1 + P2$
 $P74 \leftarrow P73 + P4$
 $P75 \leftarrow P73 + P74$
 $P76 \leftarrow P33 + P34$
 $P77 \leftarrow P33 + P76$
 $P78 \leftarrow P77 + P35$

Register File



↔



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
- Alpha 21464 and Intel Pentium 4 are examples of SMT

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

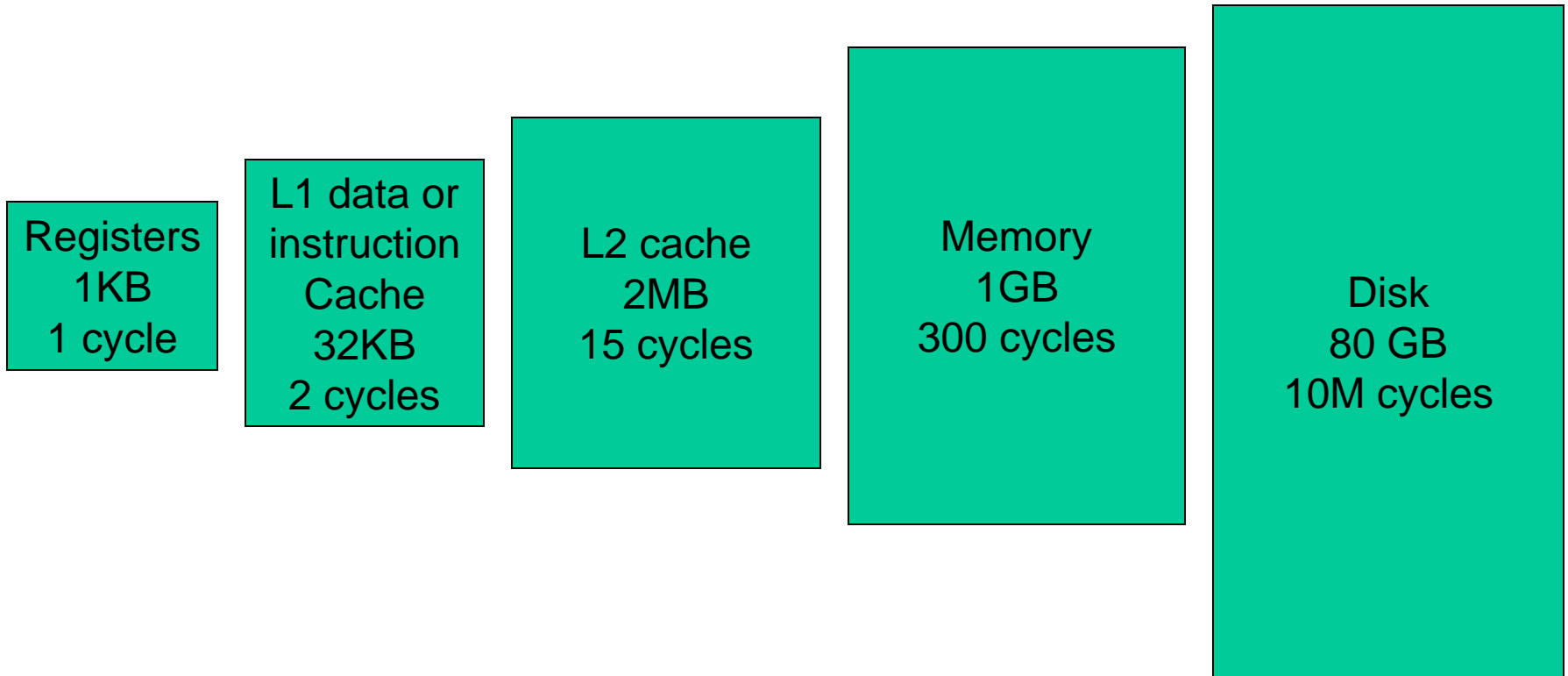
Multi-Programmed Speedup

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
perlbnk	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
ampp	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

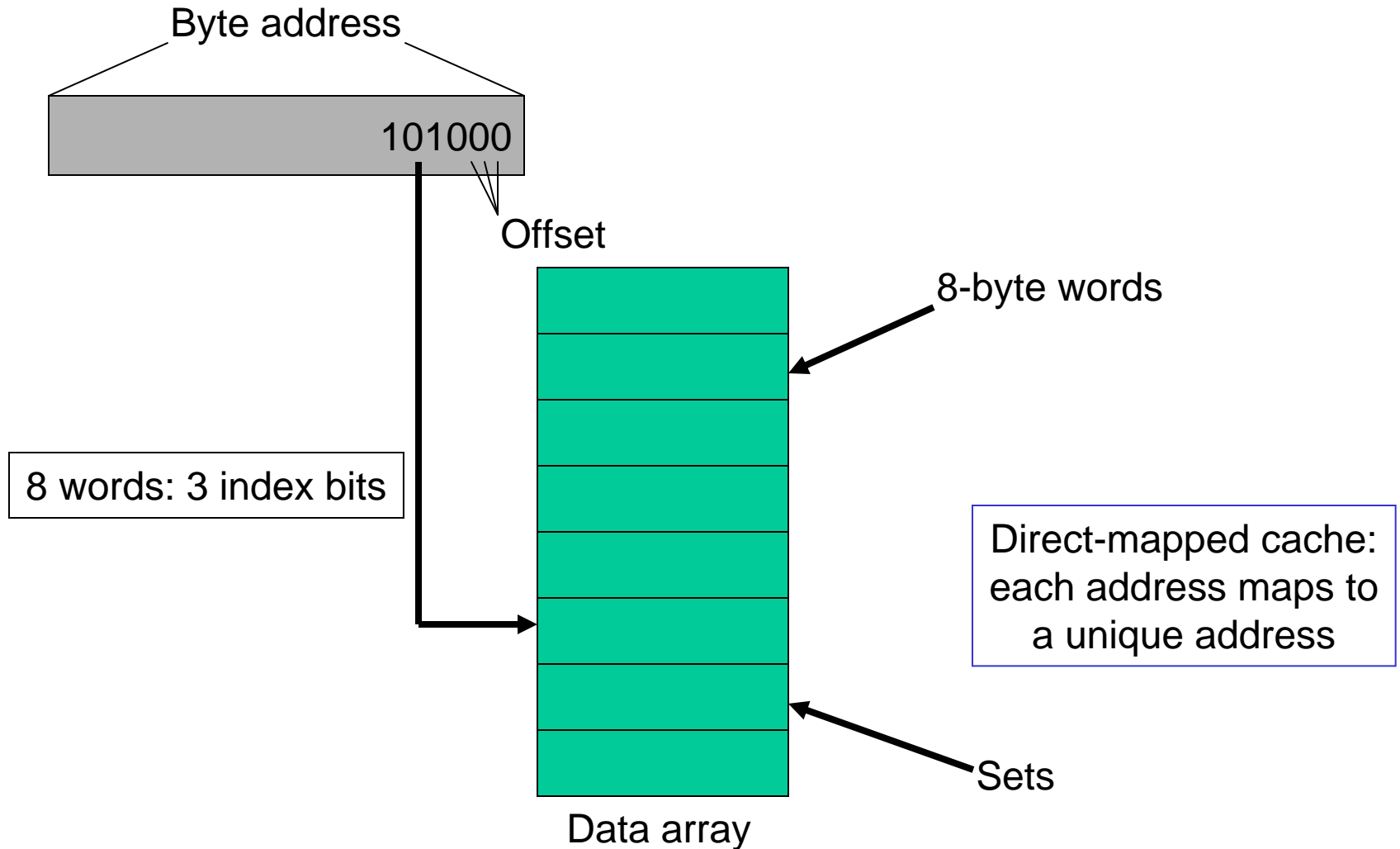
- sixtrack and eon do not degrade their partners (small working sets?)
- swim and art degrade their partners (cache contention?)
- Best combination: swim & sixtrack
worst combination: swim & art
- Static partitioning ensures low interference – worst slowdown is 0.9

Memory Hierarchy

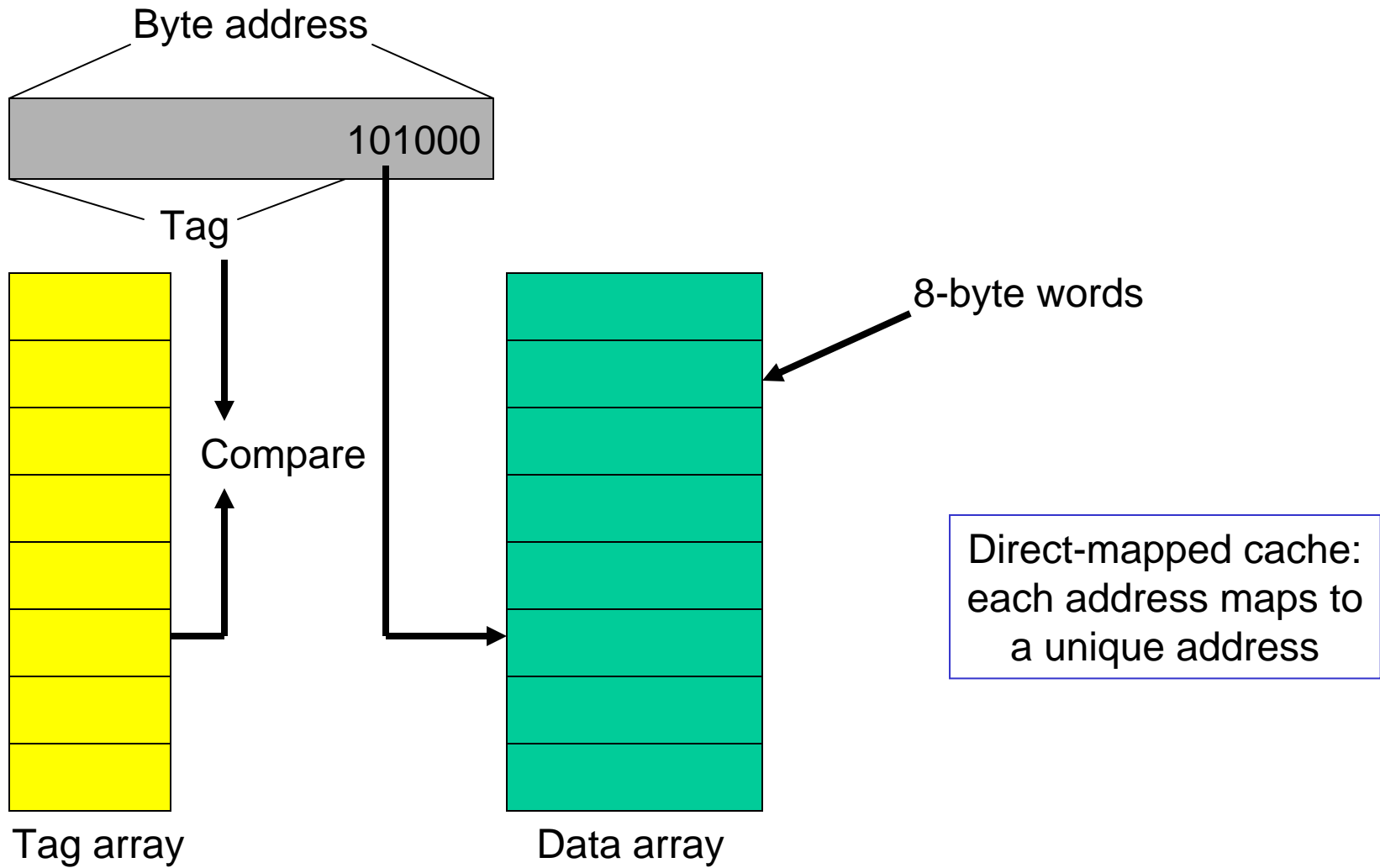
- As you go further, capacity and latency increase



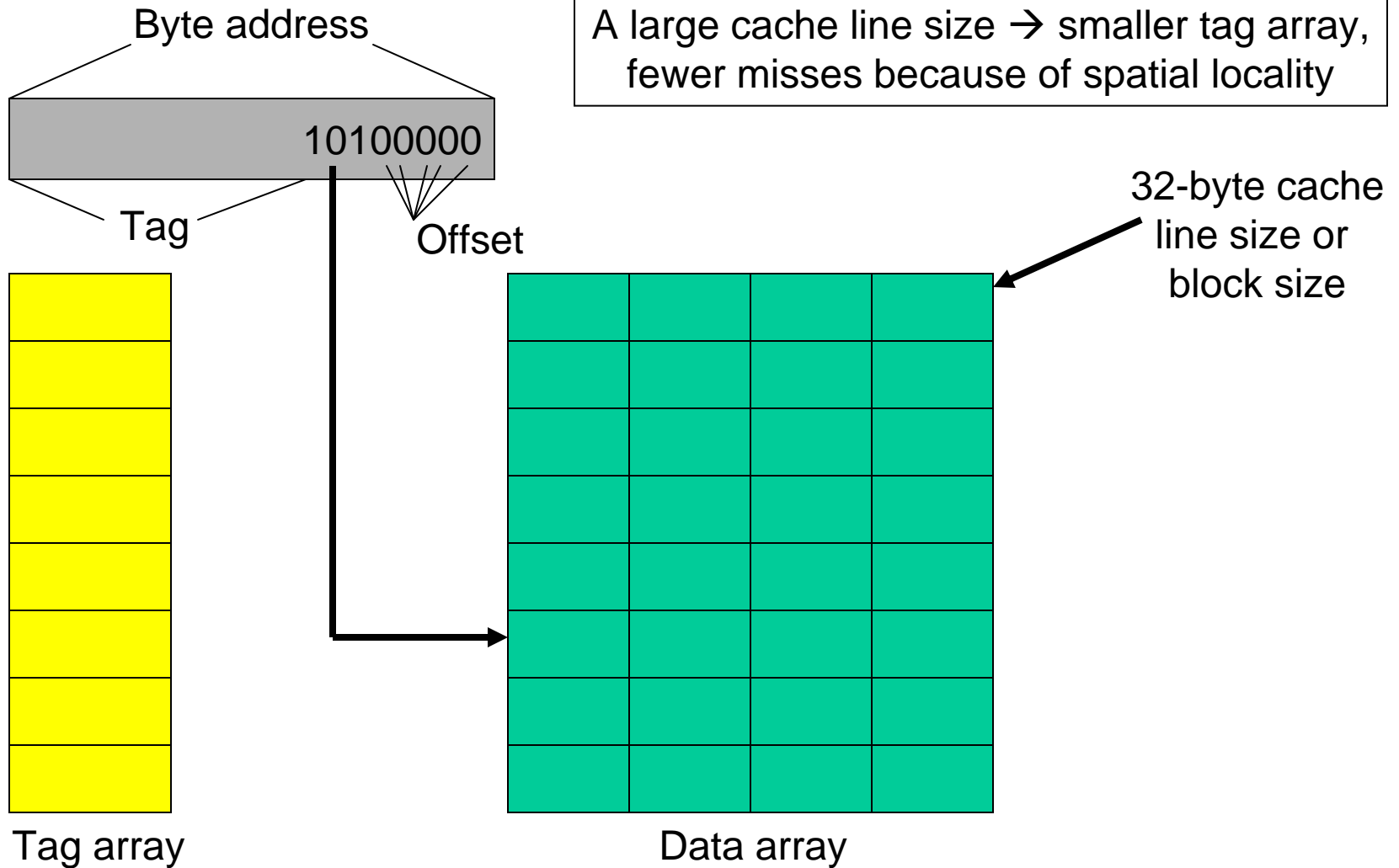
Accessing the Cache



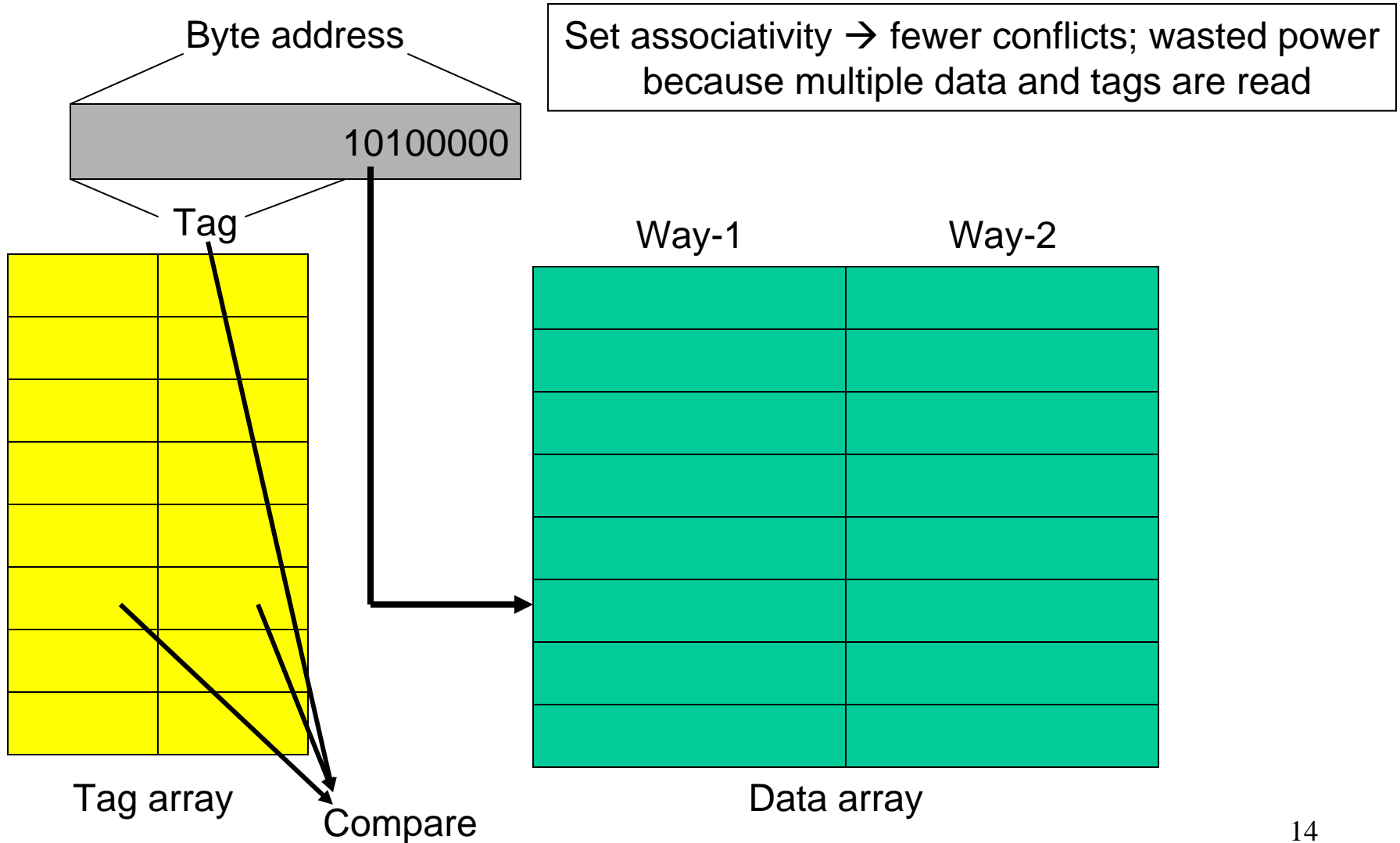
The Tag Array



Increasing Line Size



Associativity



Example

- 32 KB 4-way set-associative data cache array with 32 byte line sizes
- How many sets?
- How many index bits, offset bits, tag bits?
- How large is the tag array?

Cache Misses

- On a write miss, you may either choose to bring the block into the cache (write-allocate) or not (write-no-allocate)
- On a read miss, you always bring the block in (spatial and temporal locality) – but which block do you replace?
 - no choice for a direct-mapped cache
 - randomly pick one of the ways to replace
 - replace the way that was least-recently used (LRU)
 - FIFO replacement (round-robin)

Writes

- When you write into a block, do you also update the copy in L2?
 - write-through: every write to L1 → write to L2
 - write-back: mark the block as dirty, when the block gets replaced from L1, write it to L2
- Writeback coalesces multiple writes to an L1 block into one L2 write
- Writethrough simplifies coherency protocols in a multiprocessor system as the L2 always has a current copy of data

Title

- Bullet