332
Advanced Computer Architecture

Chapter 3: Caches and Memory Systems
Part 1: miss rate reduction using hardware

(the first of five shorter lectures on caches, address translation and the memory system)

October 2023

Paul H J Kelly

These lecture notes are partly based on the course text, Hennessy and Patterson's *Computer Architecture, a quantitative approach (3rd, 4th, 5th and 6th eds)*, and on the lecture slides of David Patterson and John Kubiatowicz's Berkeley course

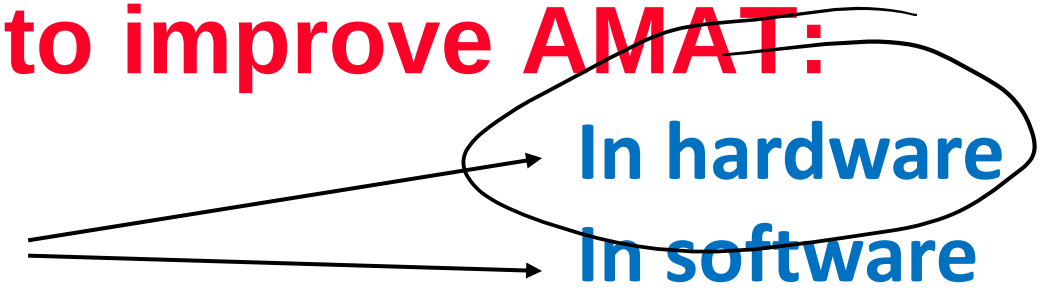
Average memory access time:

Hennessy and Patterson 6th ed Appendix B.3 pB22

$$AMAT = HitTime + MissRate \times MissPenalty$$

There are three ways to improve AMAT:

1. Reduce the miss rate
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache

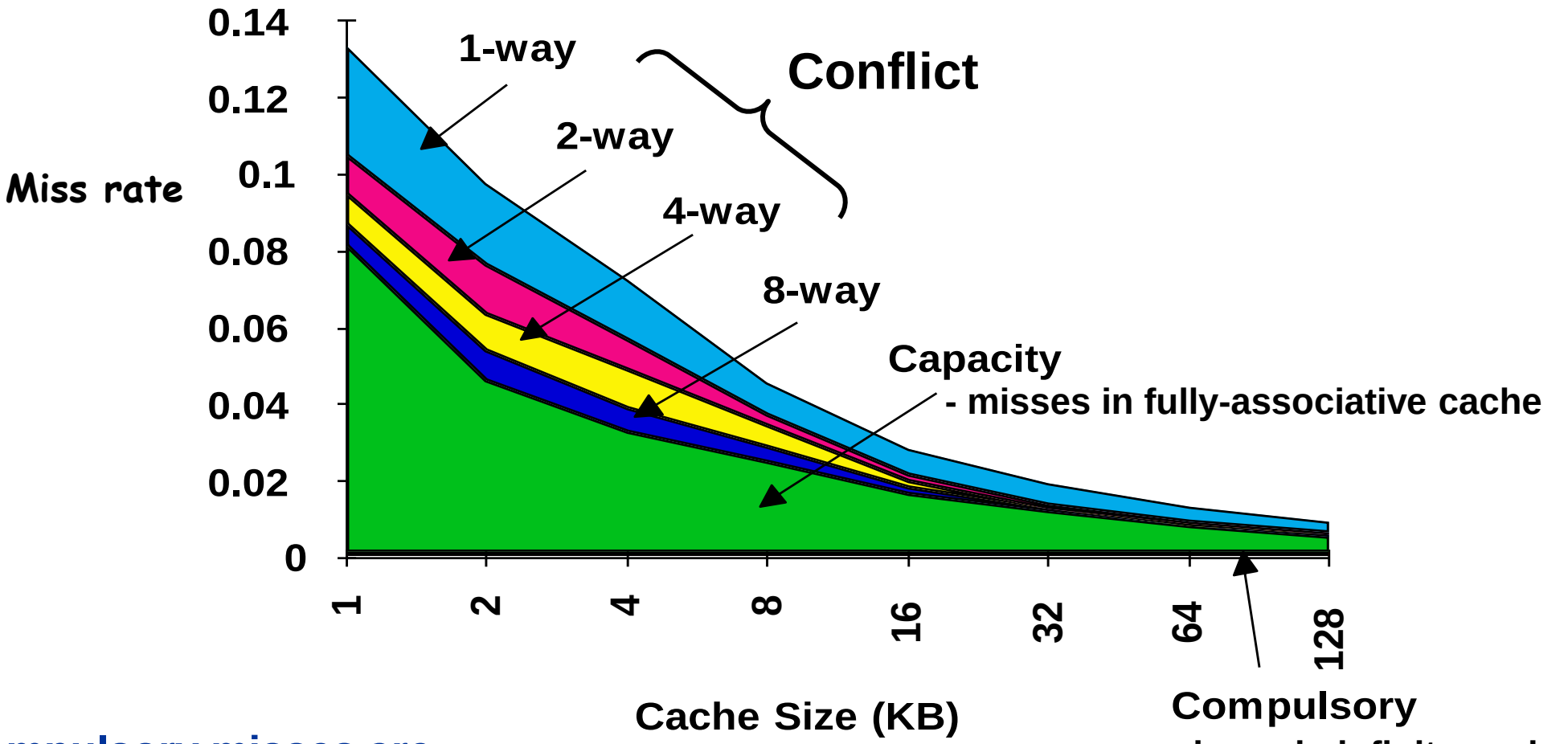


Over the next few lectures we look at each of these in turn…

Reducing Misses

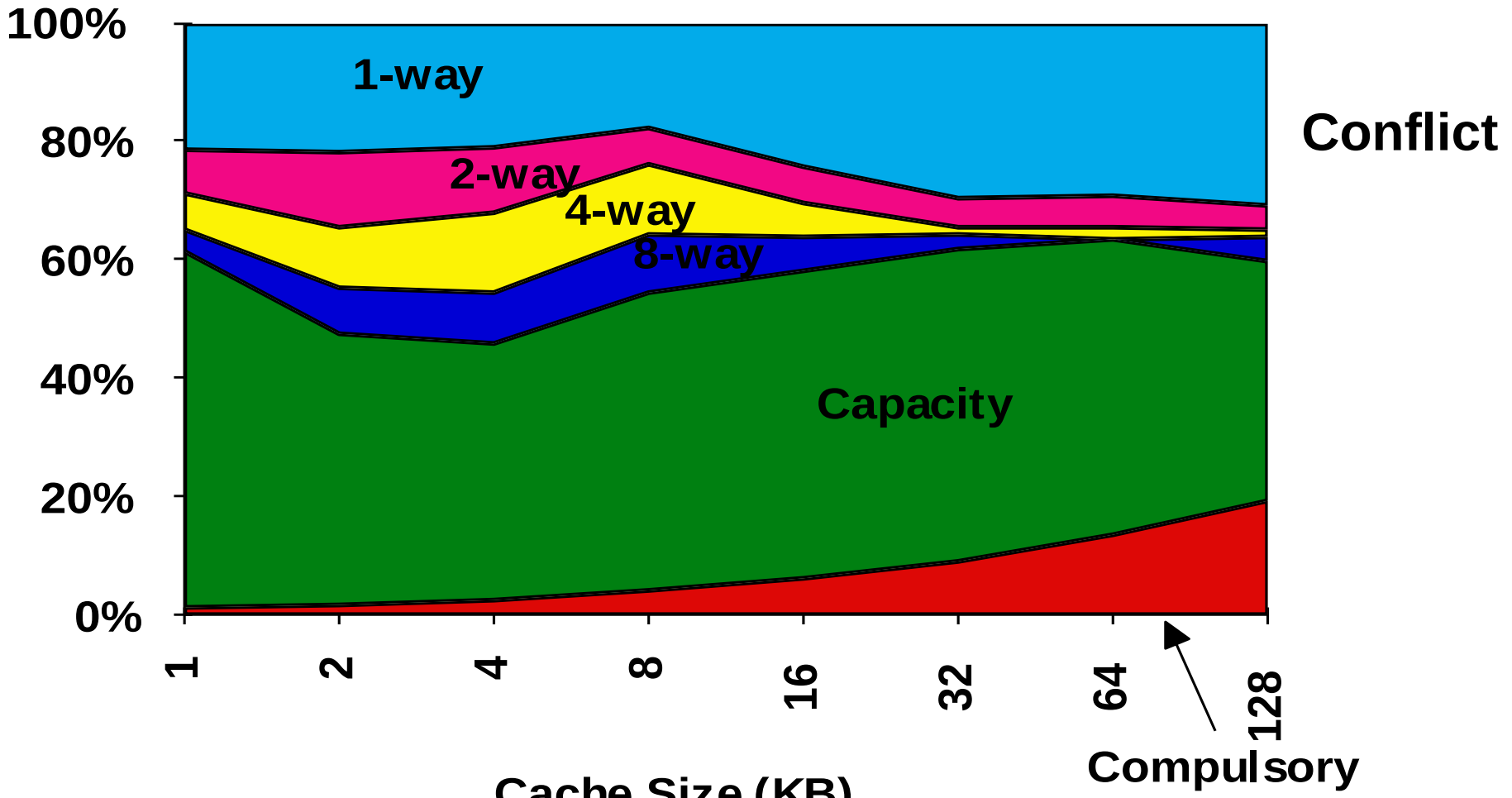
- **Classifying Misses: 3 Cs**
  - ***Compulsory*—The first access to a block is not in the cache, so the block must be brought into the cache. Also called *cold start misses* or *first reference misses*.**
    (Misses in even an Infinite Cache)
  - ***Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.**
    (Misses in Fully Associative Size X Cache)
  - ***Conflict*—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called *collision misses* or *interference misses*.**
    (Misses in w-way Associative, Size X Cache)
- **Maybe four: 4th "C":**
  - ***Coherence* - Misses caused by cache coherence: data may have been invalidated by another processor or I/O device**

3Cs Absolute Miss Rate (SPEC92)

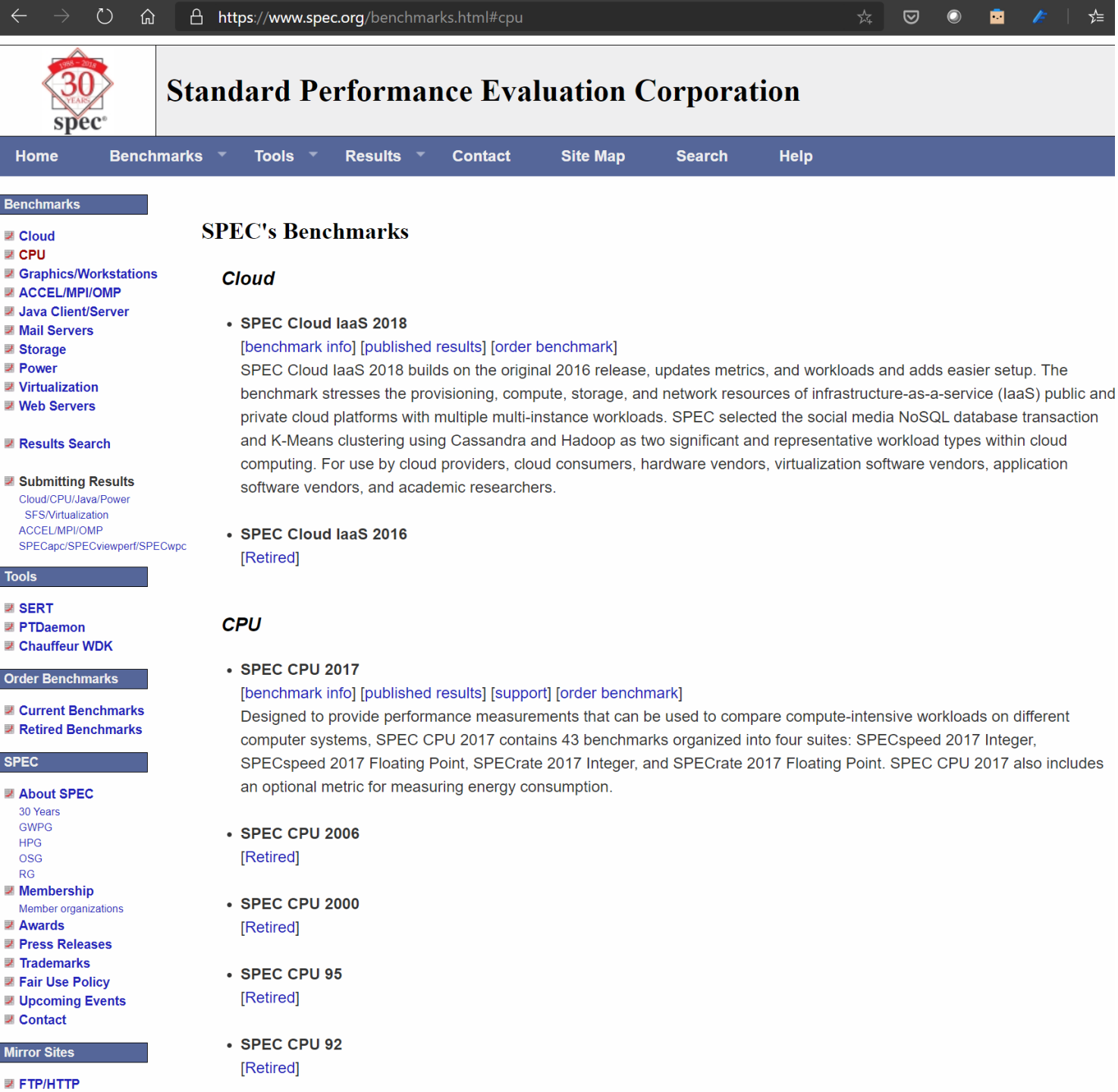


Compulsory misses are often vanishingly Few (unless??)

3Cs Relative Miss Rate



Same data, shown as proportion of total



- We will make heavy use of simulation studies based on benchmark suites
- Much of the published research relies on the SPEC CPU benchmarks
- The suite has been revised several times
- Extended, refined, broadened

Q13. What are the benchmarks?

SPEC CPU 2017 has 43 benchmarks, organized into 4 suites:

SPECrate®2017 Integer	SPECspeed®2017 Integer	Language [1]	KLOC [2]	Application Area
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1,304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

SPECrate®2017 Floating Point	SPECspeed®2017 Floating Point	Language [1]	KLOC [2]	Application Area
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity
508.namd_r		C++	8	Molecular dynamics
510.parest_r		C++	427	Biomedical imaging: optical tomography with finite elements
511.povray_r		C++, C	170	Ray tracing
519.lbm_r	619.lbm_s	C	1	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	991	Weather forecasting
526.blender_r		C++, C	1,577	3D rendering and animation
527.cam4_r	627.cam4_s	Fortran, C	407	Atmosphere modeling
	628.pop2_s	Fortran, C	338	Wide-scale ocean modeling (climate level)
538.imagick_r	638.imagick_s	C	259	Image manipulation
544.nab_r	644.nab_s	C	24	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	Computational Electromagnetics
554.roms_r	654.roms_s	Fortran	210	Regional ocean modeling

[1] For multi-language benchmarks, the first one listed determines library and link options (details)

[2] KLOC = line count (including comments/whitespace) for source files used in a build / 1000

SPEC CPU concerns **CPU-intensive** applications (no OS, no I/O)

- **Integer benchmarks** tend to make more intensive use of pointers and hard-to-predict branches
  - Hard to parallelise
- **Floating point benchmarks** may benefit more from automatic parallelisation
- **Speed**: execution time for one run of the program (possibly using multiple cores)
- **Rate**: maximum throughput of completed jobs/second

CPU2017 integer *speeds* (normalised to performance of 2006 SunFire V490 (2100MHz UltraSPARC IV+)

Test Sponsor	System Name	Parallel	Base Threads	Processor			Results		Energy	
				Enabled Cores	Enabled Chips	Threads/ Core	Base	Peak	Base	Peak
ASUSTeK Computer Inc.	ASUS RS500A-E10(KRPA-U16) Server System 2.25 GHz, AMD EPYC 7742 HTML | CSV | Text | PDF | PS | Config	Yes	64	64	1	2	8.98	9.27	--	--
ASUSTeK Computer Inc.	ASUS ESC8000 G4(Z11PG-D24) Server System (2.40 GHz, Intel Xeon Platinum 8260M) HTML | CSV | Text | PDF | PS | Config	Yes	48	48	2	1	10.8	11.0	--	--
ASUSTeK Computer Inc.	ASUS ESC8000 G4(Z11PG-D24) Server System (2.60 GHz, Intel Xeon Gold 6240M) HTML | CSV | Text | PDF | PS | Config	Yes	36	36	2	1	10.6	10.8	--	--
ASUSTeK Computer Inc.	ASUS ESC8000 G4(Z11PG-D24) Server System (2.10 GHz, Intel Xeon Gold 6252) HTML | CSV | Text | PDF | PS | Config	Yes	48	48	2	1	10.3	10.5	--	--
ASUSTeK Computer Inc.	ASUS ESC8000 G4(Z11PG-D24) Server System (3.80 GHz, Intel Xeon Platinum 8256) HTML | CSV | Text | PDF | PS | Config	Yes	16	8	2	2	10.1	10.3	--	--
Fujitsu	PRIMERGY TX1320 M4, Intel Xeon E-2288G, 3.70 GHz HTML | CSV | Text | PDF | PS | Config	Yes	16	8	1	2	12.1	Not Run	219	Not Run
Oracle Corporation	Sun Fire V490 HTML | CSV | Text | PDF | PS | Config	Yes	1	8	4	1	1.00	Not Run	1.00	--

CPU2017 floating point *rates* (normalised to performance of 2006 SunFire V490 (2100MHz UltraSPARC IV+)

Test Sponsor	System Name	Base Copies	Processor			Results		Energy	
			Enabled Cores	Enabled Chips	Threads/ Core	Base	Peak	Base	Peak
ASUSTeK Computer Inc.	ASUS RS700-E9(Z11PP-D24) Server System (2.70 GHz, Intel Xeon Gold 6150) HTML | CSV | Text | PDF | PS | Config	72	36	2	2	199	201	--	--
ASUSTeK Computer Inc.	ASUS RS700-E9(Z11PP-D24) Server System (2.10 GHz, Intel Xeon Platinum 8176) HTML | CSV | Text | PDF | PS | Config	112	56	2	2	233	237	--	--
Dell Inc.	PowerEdge R7425 (AMD EPYC 7601, 2.20 GHz) HTML | CSV | Text | PDF | PS | Config	128	64	2	2	257	259	--	--
Fujitsu	Fujitsu SPARC M12-2S HTML | CSV | Text | PDF | PS | Config	768	96	8	8	663	796	--	--
Fujitsu	Fujitsu SPARC M12-2S HTML | CSV | Text | PDF | PS | Config	1536	192	16	8	1250	1520	--	--
IBM Corporation	IBM Power S924 (3.4 - 3.9 GHz, 24 core, SLES) HTML | CSV | Text | PDF | PS | Config	144	24	2	8	213	277	--	--
IBM Corporation	IBM Power E950 (3.4 - 3.8 GHz, 40 core, SLES) HTML | CSV | Text | PDF | PS | Config	320	40	4	8	392	475	--	--

Arbitrarily selected - see https://www.spec.org/cpu2017/results/cpu2017.html for full results, including integer rates and floating-point speeds, and many more details.

SPEC CPU®2017 Integer Speed Result

ASUSTeK Computer Inc.

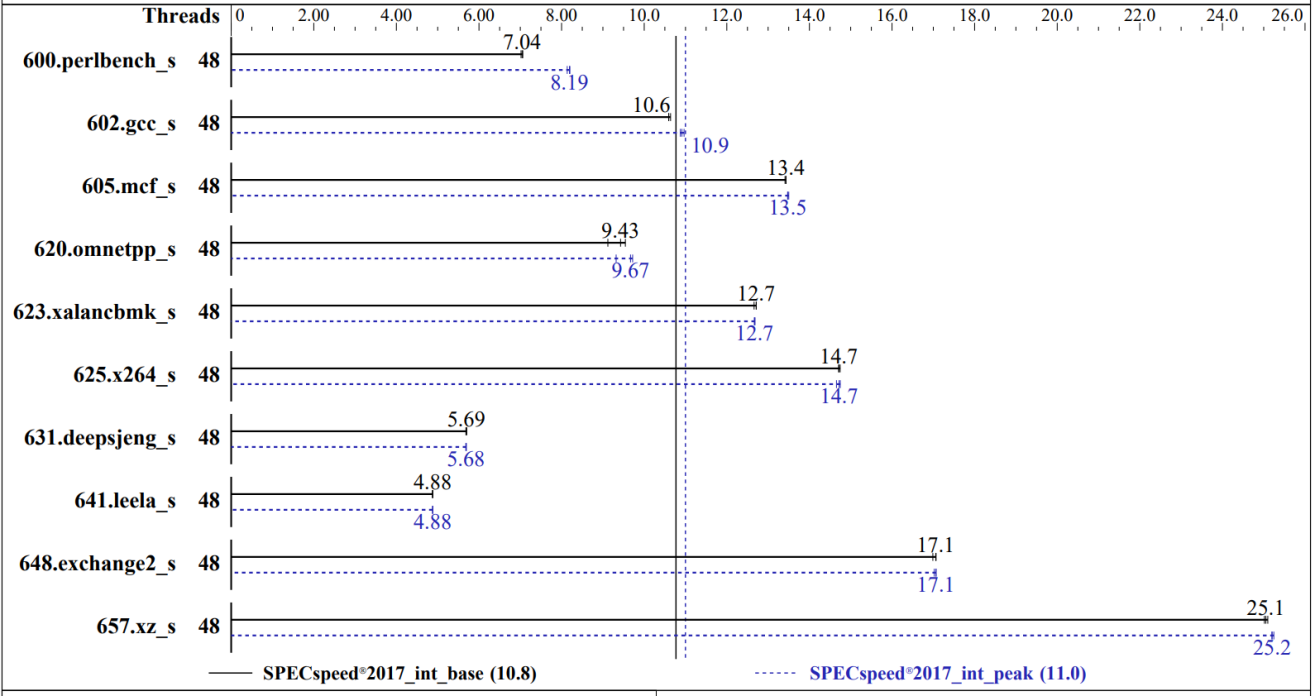
ASUS ESC8000 G4(Z11PG-D24) Server System
(2.40 GHz, Intel Xeon Platinum 8260M)

SPECspeed®2017_int_base = 10.8

SPECspeed®2017_int_peak = 11.0

CPU2017 License: 9016
Test Sponsor: ASUSTeK Computer Inc.
Tested by: ASUSTeK Computer Inc.

Test Date: Aug-2019
Hardware Availability: Apr-2019
Software Availability: May-2019



Hardware

CPU Name:	Intel Xeon Platinum 8260M
Max MHz:	3900
Nominal:	2400
Enabled:	48 cores, 2 chips
Orderable:	1,2 chips
Cache L1:	32 KB I + 32 KB D on chip per core
L2:	1 MB I+D on chip per core
L3:	35.75 MB I+D on chip per chip
Other:	None
Memory:	768 GB (24 x 32 GB 2Rx4 PC4-2933Y-R)
Storage:	1 x 1 TB SATA SSD
Other:	None

Software

OS:	SUSE Linux Enterprise Server 15 Kernel 4.12.14-23-default
Compiler:	C/C++: Version 19.0.4.227 of Intel C/C++ Compiler Build 20190416 for Linux; Fortran: Version 19.0.4.227 of Intel Fortran Compiler Build 20190416 for Linux
Parallel:	Yes
Firmware:	Version 5102 released Feb-2019
File System:	xfs
System State:	Run level 3 (multi-user)
Base Pointers:	64-bit
Peak Pointers:	64-bit
Other:	jemalloc: jemalloc memory allocator library V5.0.1
Power Management:	--

- Each reported benchmark result includes elaborate details of hardware and software configuration
- Including details of compiler optimisation flags
- For **base**, same compiler flags for all benchmark programs
- For **peak**, per-benchmark tuning of compiler flags
  - All compiler flags are recorded in the benchmark report

SPEC CPU®2017 Integer Speed Result

ASUSTeK Computer Inc.

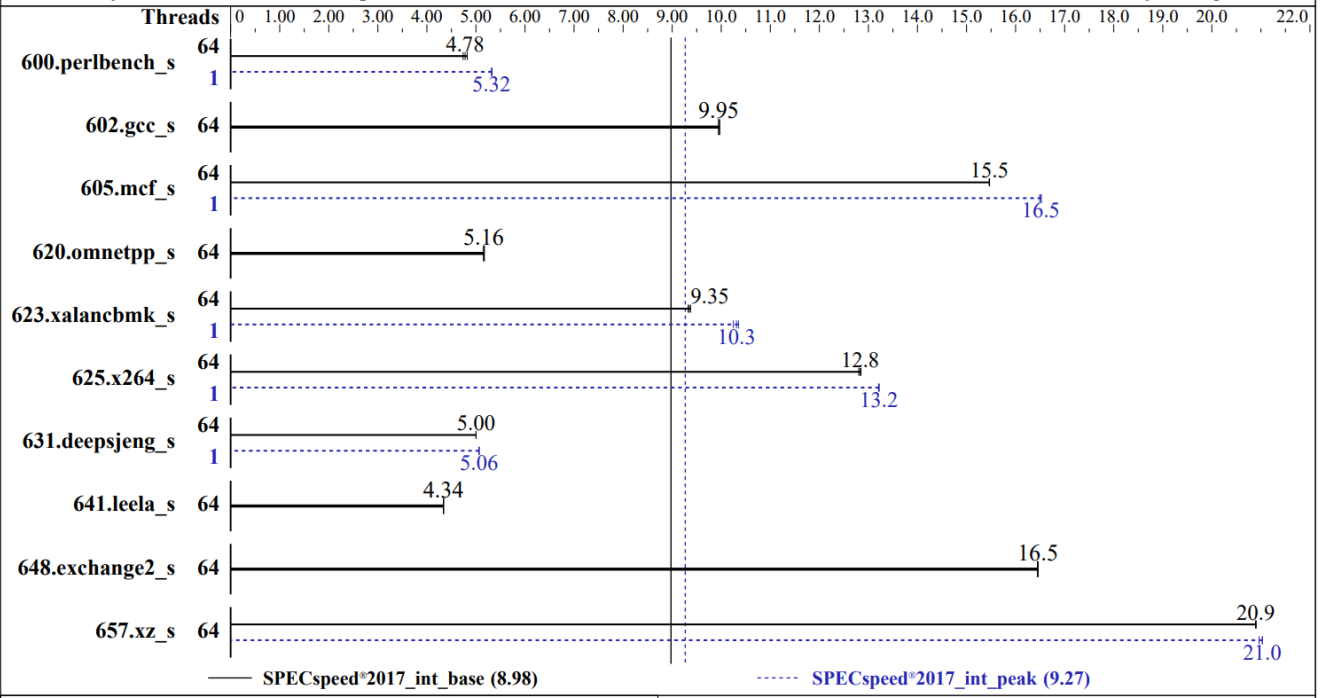
ASUS RS500A-E10(KRPA-U16) Server System
2.25 GHz, AMD EPYC 7742

SPECspeed®2017_int_base = 8.98

SPECspeed®2017_int_peak = 9.27

CPU2017 License: 9016
Test Sponsor: ASUSTeK Computer Inc.
Tested by: ASUSTeK Computer Inc.

Test Date: Aug-2019
Hardware Availability: Aug-2019
Software Availability: Aug-2019



Hardware

CPU Name:	AMD EPYC 7742
Max MHz:	3400
Nominal:	2250
Enabled:	64 cores, 1 chip, 2 threads/core
Orderable:	1 chip
Cache L1:	32 KB I + 32 KB D on chip per core
L2:	512 KB I+D on chip per core
L3:	256 MB I+D on chip per chip, 16 MB shared / 4 cores
Other:	None
Memory:	256 GB (8 x 32 GB 2Rx4 PC4-3200AA-R)
Storage:	1 x 1 TB SATA SSD
Other:	None

Software

OS:	SUSE Linux Enterprise Server 15 SP1 (x86_64) Kernel 4.12.14-195-default
Compiler:	C/C++/Fortran: Version 2.0.0 of AOCC
Parallel:	Yes
Firmware:	Version 0302 released Aug-2019
File System:	xfs
System State:	Run level 3 (multi-user)
Base Pointers:	64-bit
Peak Pointers:	32/64-bit
Other:	jemalloc: jemalloc memory allocator library v5.1.0
Power Management:	--

- Different systems achieve different relative performance on different programs in the benchmark suite
- Performance is averaged across the suite to produce the overall speed result
- The geometric mean is used (not the arithmetic mean)
  - See https://en.wikipedia.org/wiki/Geometric_mean
- Devising appropriate summary statistics is a subtle problem
- What are the criteria for good benchmark suite design?

How We Can Reduce Misses?

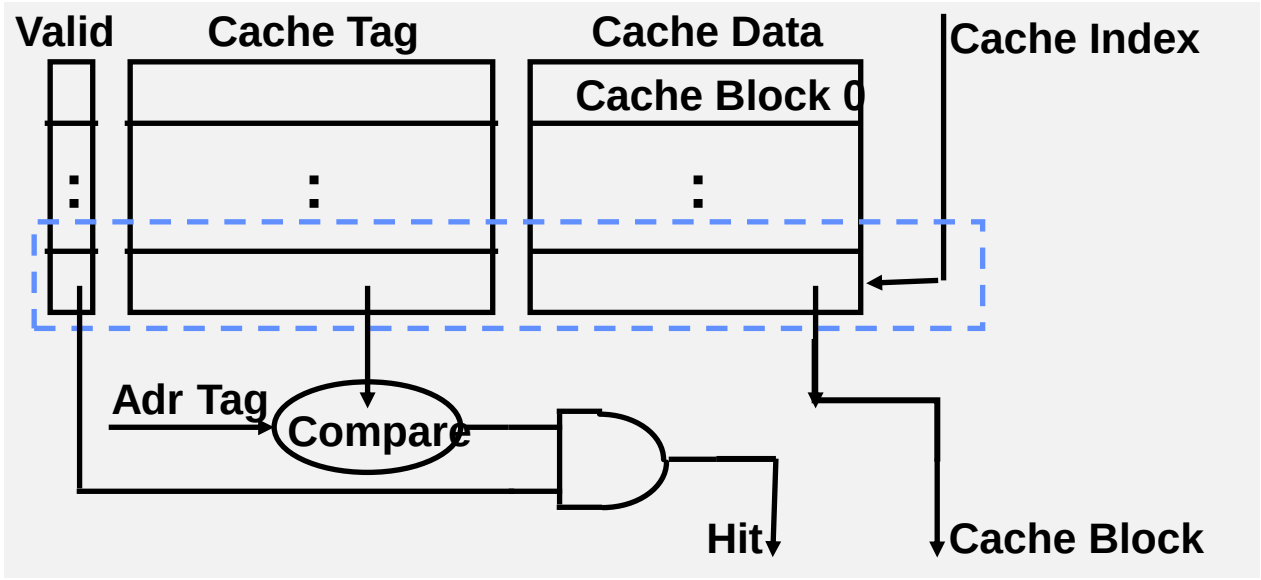
- **3 Cs: Compulsory, Capacity, Conflict**
- **In all cases, assume total cache size not changed:**
- **What happens if:**

1) Change Block Size:
Which of 3Cs is obviously affected?

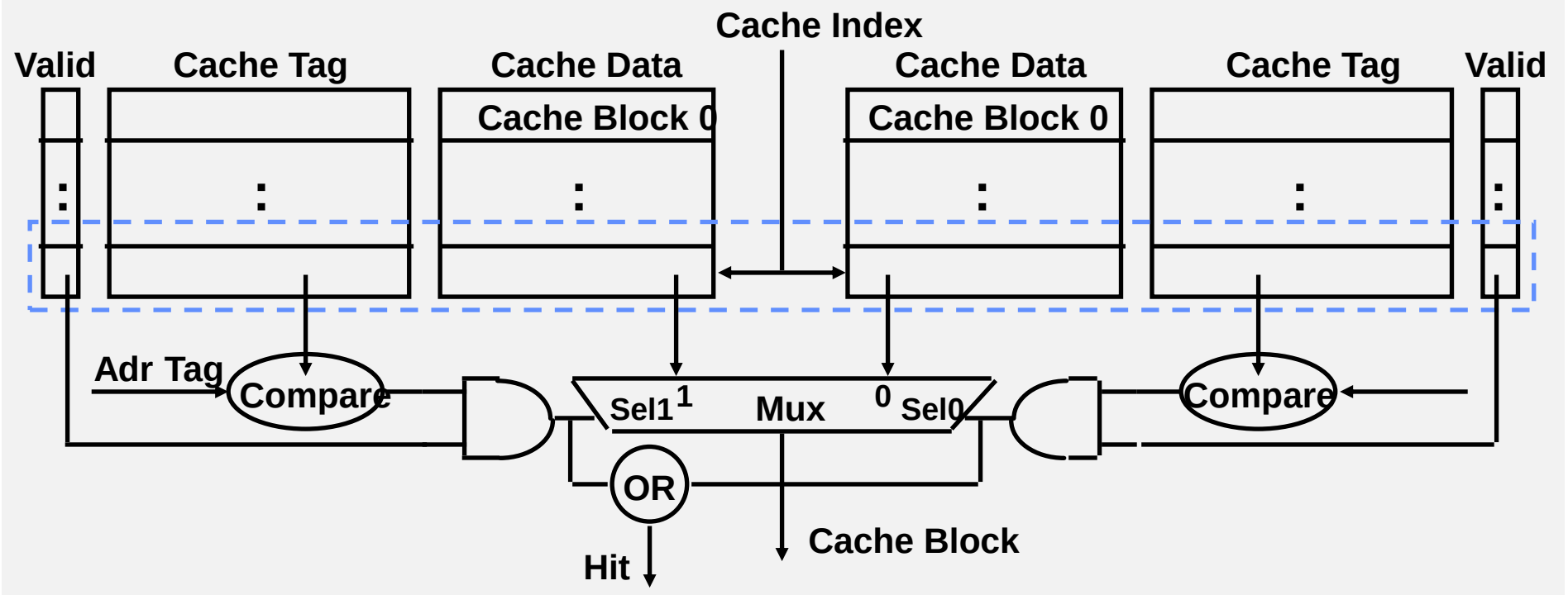
2) Change Associativity:
Which of 3Cs is obviously affected?

3) Change Compiler:
Which of 3Cs is obviously affected?

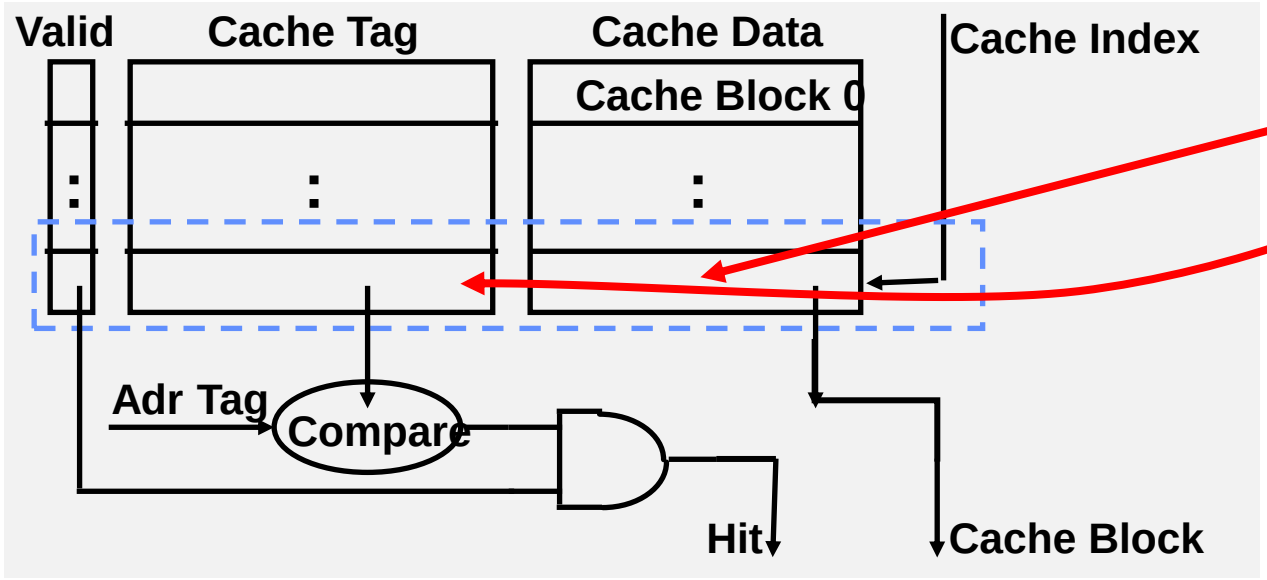
We will look at each of these in turn…



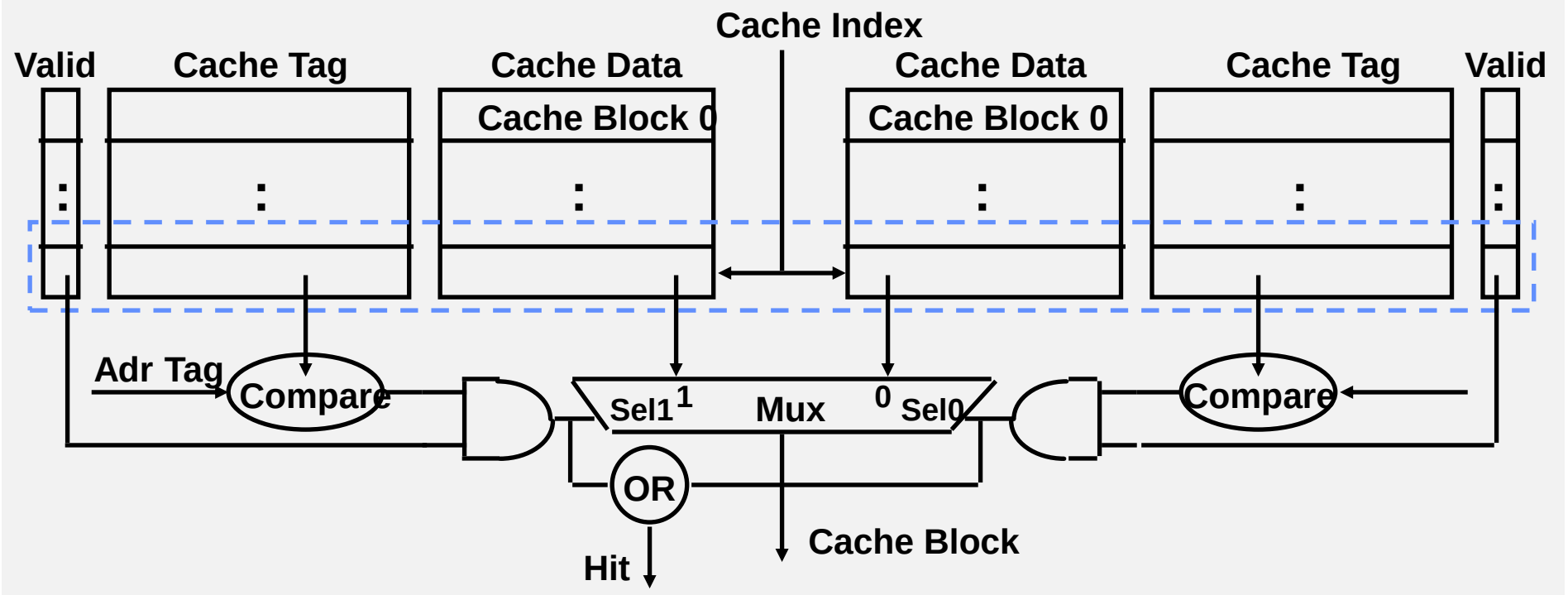
- **Recall: direct-mapped cache**



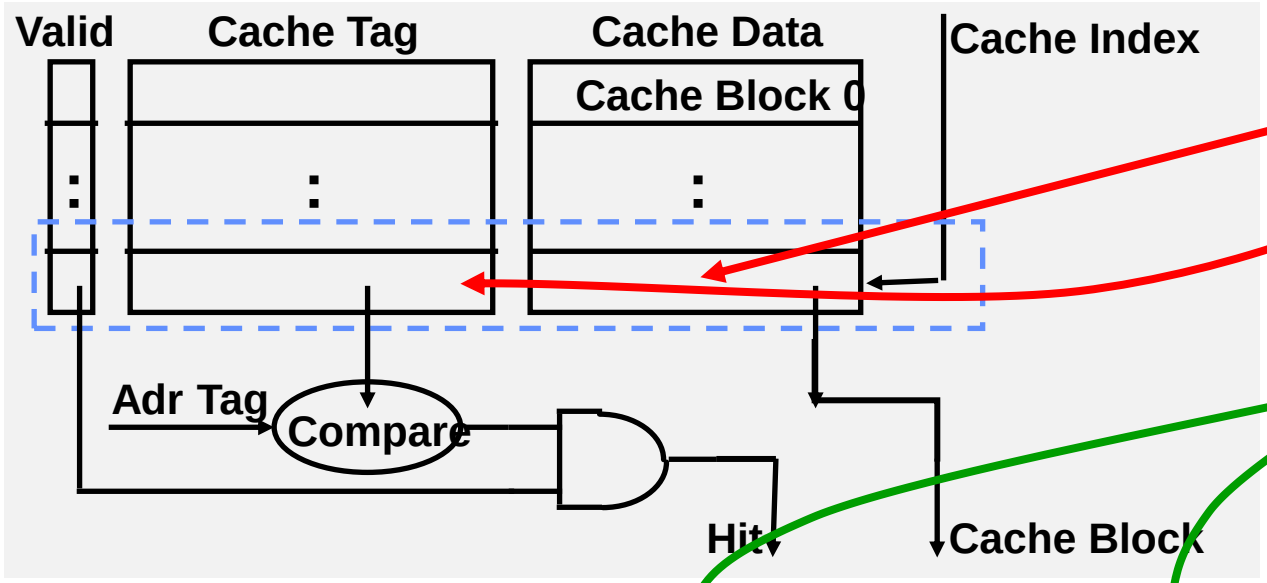
- **Recall: 2-way set-associative cache**



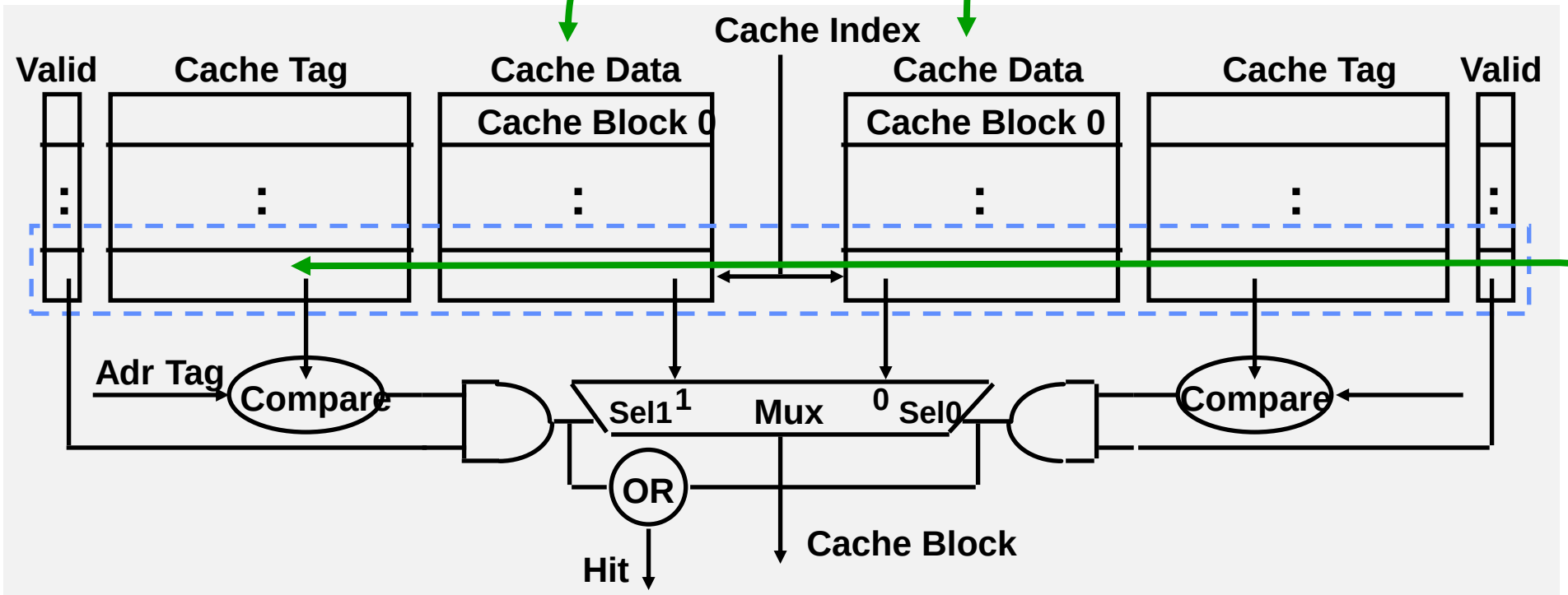
- Cache anatomy:
  - ***cache block/cache line***: unit of allocation
  - ***Tag***: memory address of cached block



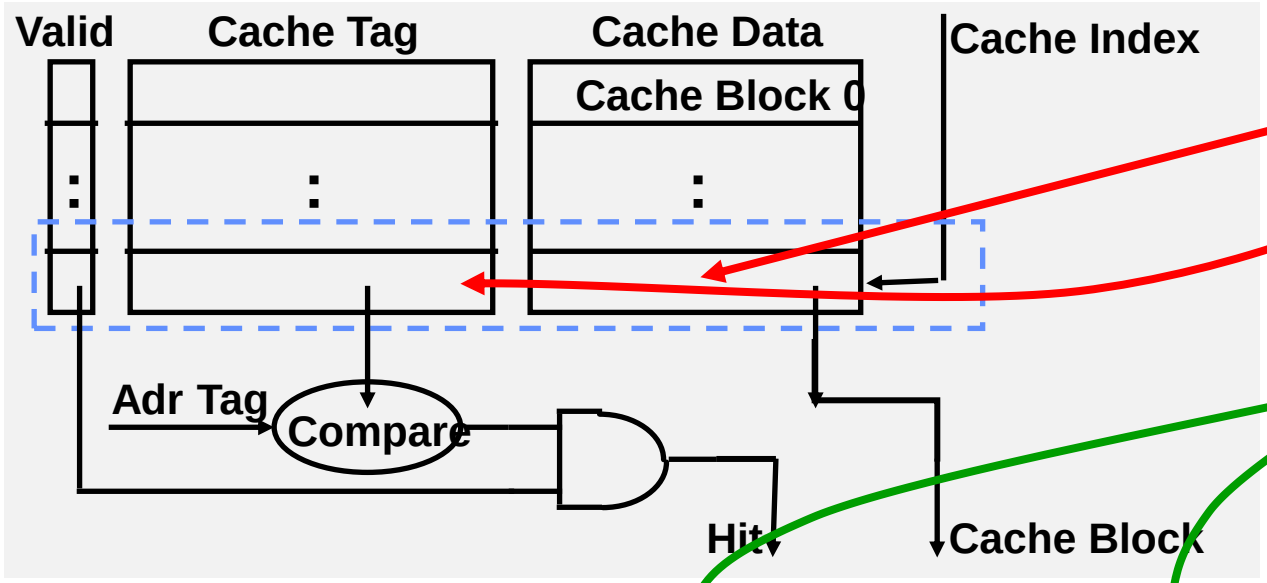
- **Recall: 2-way set-associative cache**



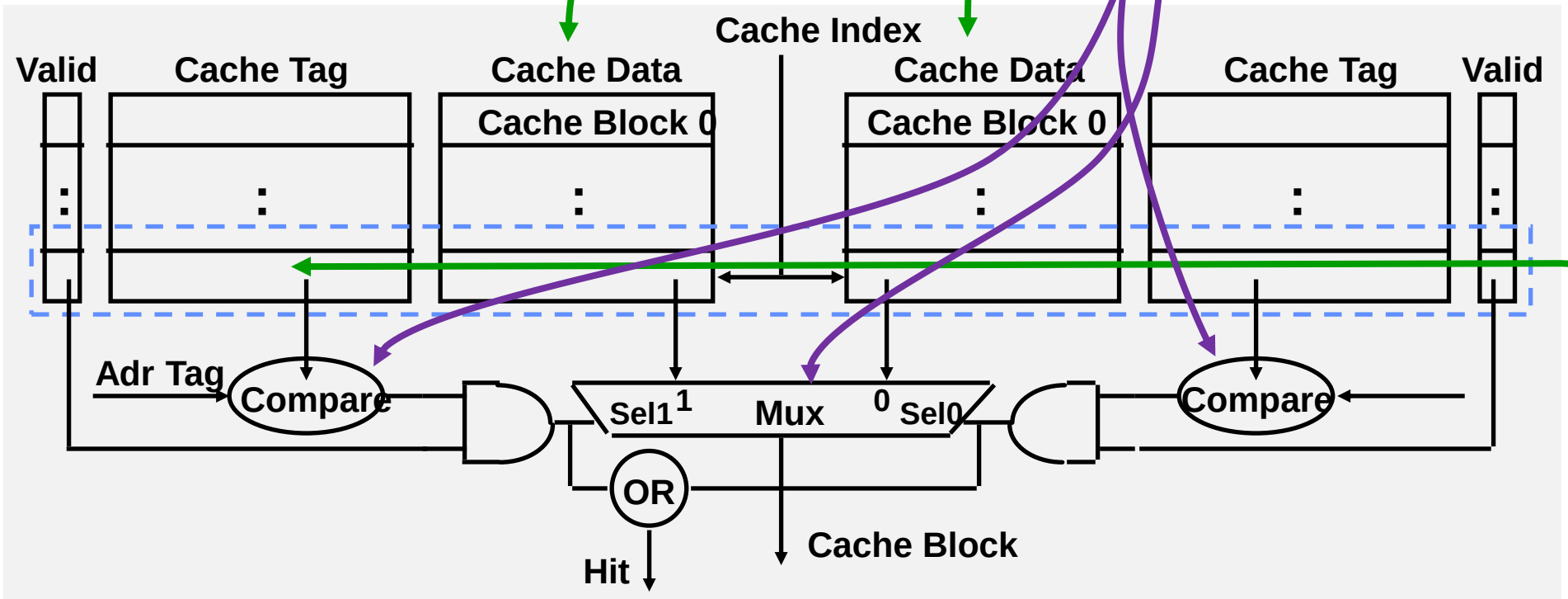
- Cache anatomy:
  - ***cache block/cache line***: unit of allocation
  - ***Tag***: memory address of cached block
  - ***Set***: set of cache blocks indexed by a given cache index
  - ***Way***: set of alternative locations for a stored block in a given set



Recall: 2-way set-associative cache

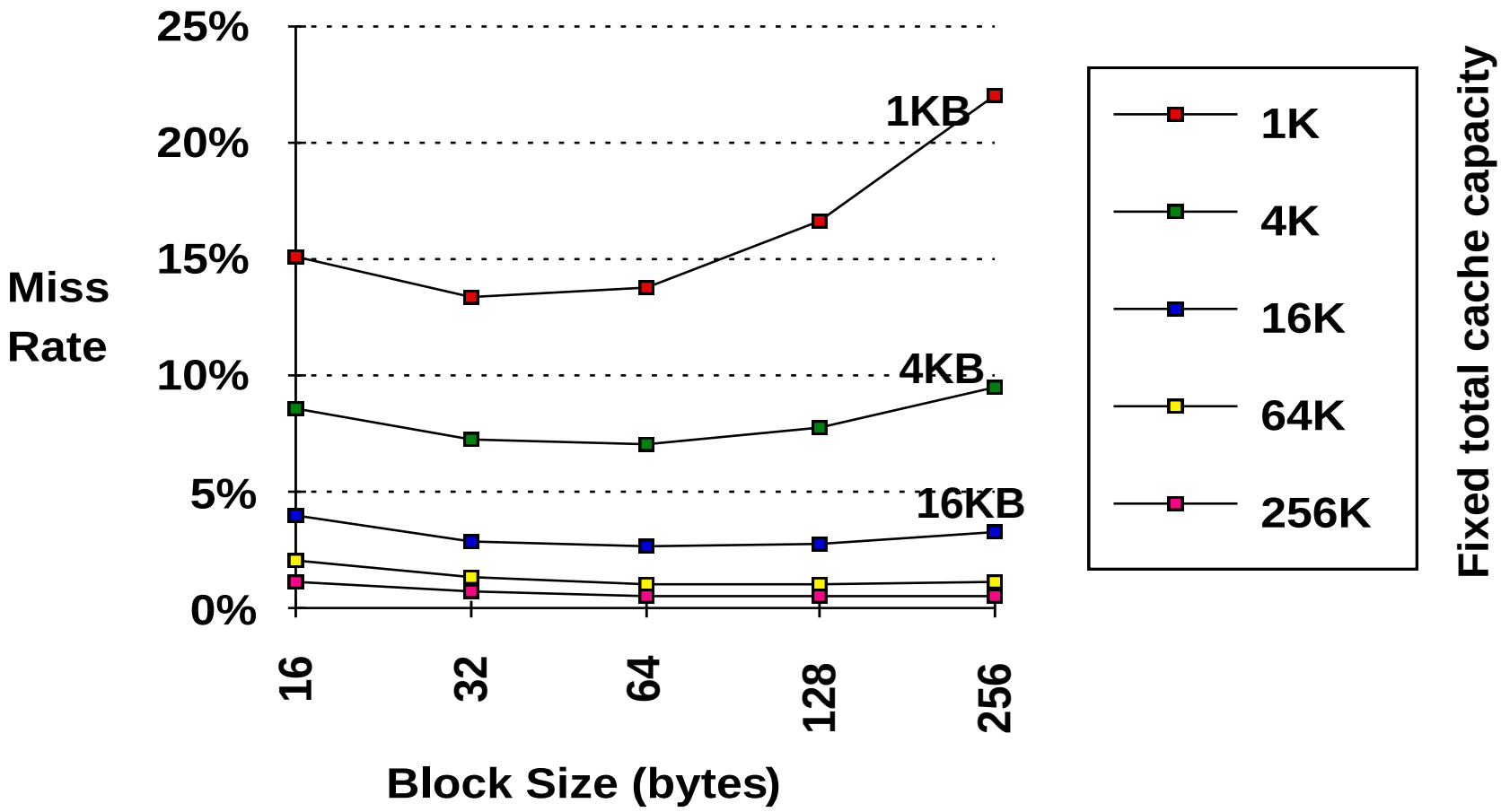


- Cache anatomy:
  - ***cache block/cache line***: unit of allocation
  - ***Tag***: memory address of cached block
  - ***Set***: set of cache blocks indexed by a given cache index
  - ***Way***: set of alternative locations for a stored block in a given set
  - ***Comparator***: check tag matches address
  - ***Selector***: picks the data from the way with the matching tag (if any)



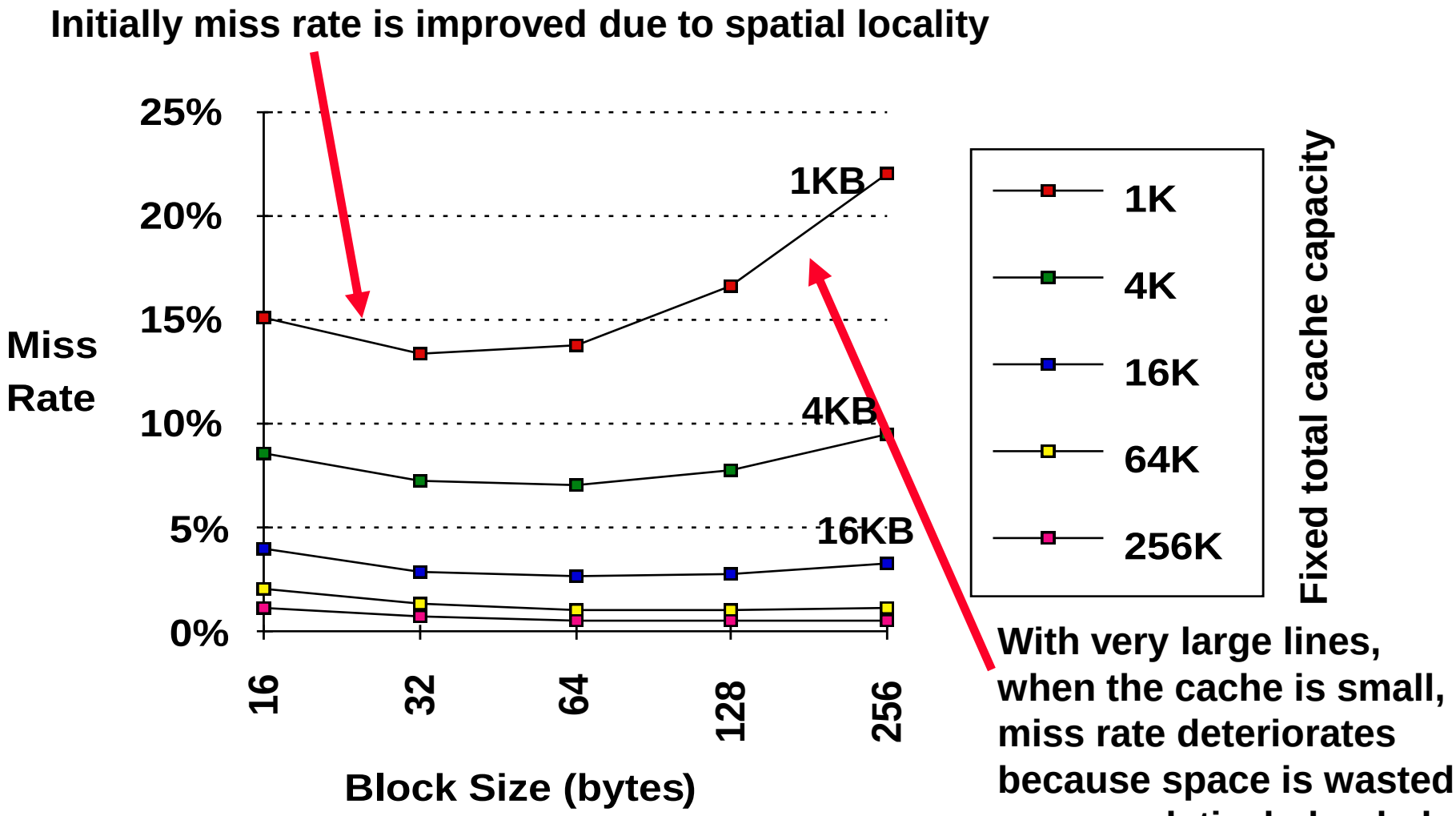
Recall: 2-way set-associative cache

Reduce misses via larger block size



Bigger blocks allow us to exploit more spatial locality – but...

Reduce misses via larger block size



Note that we are looking *only* at miss rate – large blocks will take longer to load (ie a higher *miss penalty*)

Later we will see

- Better ways to exploit spatial locality, such as prefetching
- Ways to reduce the miss penalty, eg critical word first and sectoring

Associativity: Average Memory Access Time vs. miss rate

- **Beware: Execution time is all that really matters**
  - Will Clock Cycle time increase?
  - For example because the cache's selector logic is deeper
- Example: suppose clock cycle time (CCT) =
  - 1.10 for 2-way,
  - 1.12 for 4-way,
  - 1.14 for 8-way
  - vs. CCT = 1.0 for direct mapped
- Although miss rate is improved by increasing associativity, the cache hit time is increased slightly

Cache Size (KB)	Associativity: 1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	**2.01**
2	1.98	1.86	1.76	**1.68**
4	1.72	1.67	1.61	**1.53**
8	1.46	*1.48*	*1.47*	**1.43**
16	***1.29***	*1.32*	*1.32*	*1.32*
32	***1.20***	*1.24*	*1.25*	*1.27*
64	***1.14***	*1.20*	*1.21*	*1.23*
128	***1.10***	*1.17*	*1.18*	*1.20*

Average memory access time (cycles)

(*Red* means A.M.A.T. not improved by more associativity)

- Illustrative benchmark study. Real clock cycle cost likely smaller

Associativity: Average Memory Access Time vs. miss rate

- **Beware: Execution time is all that really matters**
  - Will Clock Cycle time increase?
  - For example because the cache's selector logic is deeper
- Example: suppose clock cycle time (CCT) =
  - 1.10 for 2-way,
  - 1.12 for 4-way,
  - 1.14 for 8-way
  - vs. CCT = 1.0 for direct mapped
- Although miss rate is improved by increasing associativity, the cache hit time is increased slightly
- Illustrative benchmark study. Real clock cycle cost likely smaller

- **Solution?**
  - Way prediction
  - See H&P6ed p98

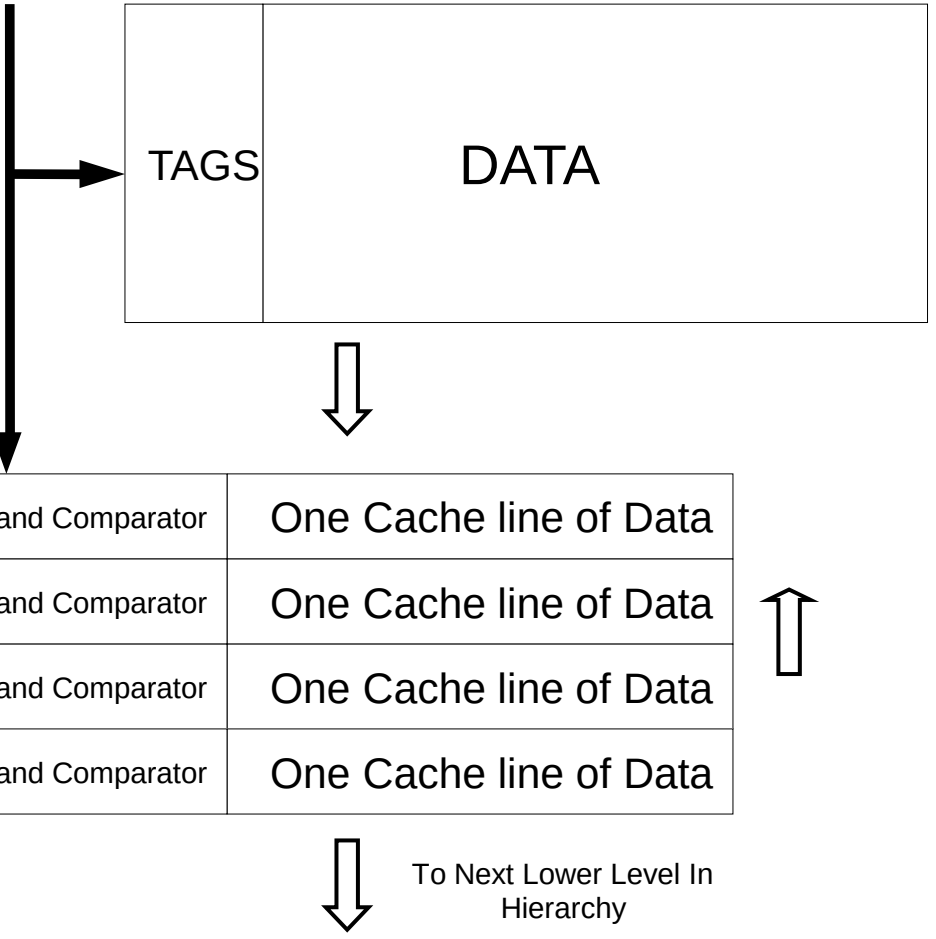
Cache Size (KB)	Associativity: 1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	**2.01**
2	1.98	1.86	1.76	**1.68**
4	1.72	1.67	1.61	**1.53**
8	1.46	*1.48*	*1.47*	**1.43**
16	***1.29***	*1.32*	*1.32*	*1.32*
32	***1.20***	*1.24*	*1.25*	*1.27*
64	***1.14***	*1.20*	*1.21*	*1.23*
128	***1.10***	*1.17*	*1.18*	*1.20*

Average memory access time (cycles)

(*Red* means A.M.A.T. not improved by more associativity)

Another way to reduce associativity conflict misses: "Victim Cache"

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- On miss, allocate into direct-mapped cache
- On replacement, allocate into victim cache
- On access, check both
- On victim cache, re-allocate into direct-mapped cache



Rarely used for L1 but commonly used for last-level caches

~~HP Fellow~~
~~Director, Exascale Computing Lab~~
~~Palo Alto~~
Distinguished Hardware Engineer at Google

Jouppi, N. P. 1998. Improving direct-mapped cache performance by the addition of a small fully-associative cache prefetch buffers. In *25 Years of the international Symposia on Computer Architecture (Selected Papers)* (Barcelona, Spain, June 27 - July 02, 1998). G. S. Sohi, Ed. ISCA '98. ACM, New York, NY, 388-397. DOI= http://doi.acm.org/10.1145/285930.285998

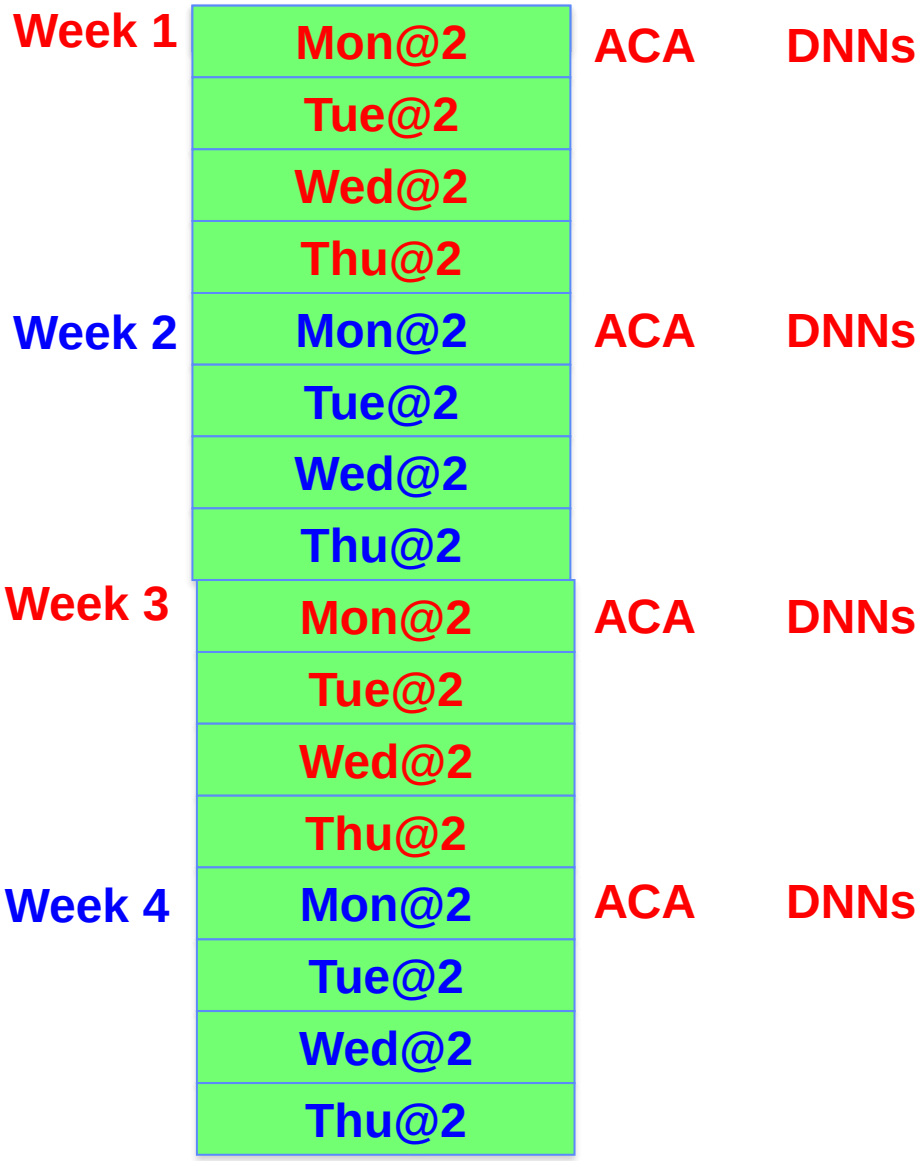
(A digression: competitive algorithms

- Given two strategies
  - Each strategy is good for some cases but disastrous for others *(eg direct mapped vs fully-associative)*
  - Can we combine the two to create a good composite strategy?
  - What price do we have to pay?
- Example: ski rental problem (https://en.wikipedia.org/wiki/Ski_rental_problem)
- Example: spinlocks vs context-switching
- Example: paging (should I stay or should I go)
- Related: the Secretary problem (actually best understood as dating)
- I hope you will demand a course in competitive algorithms and apply them to diverse computer systems problems
- See http://www14.in.tum.de/personen/albers/papers/brics.pdf)

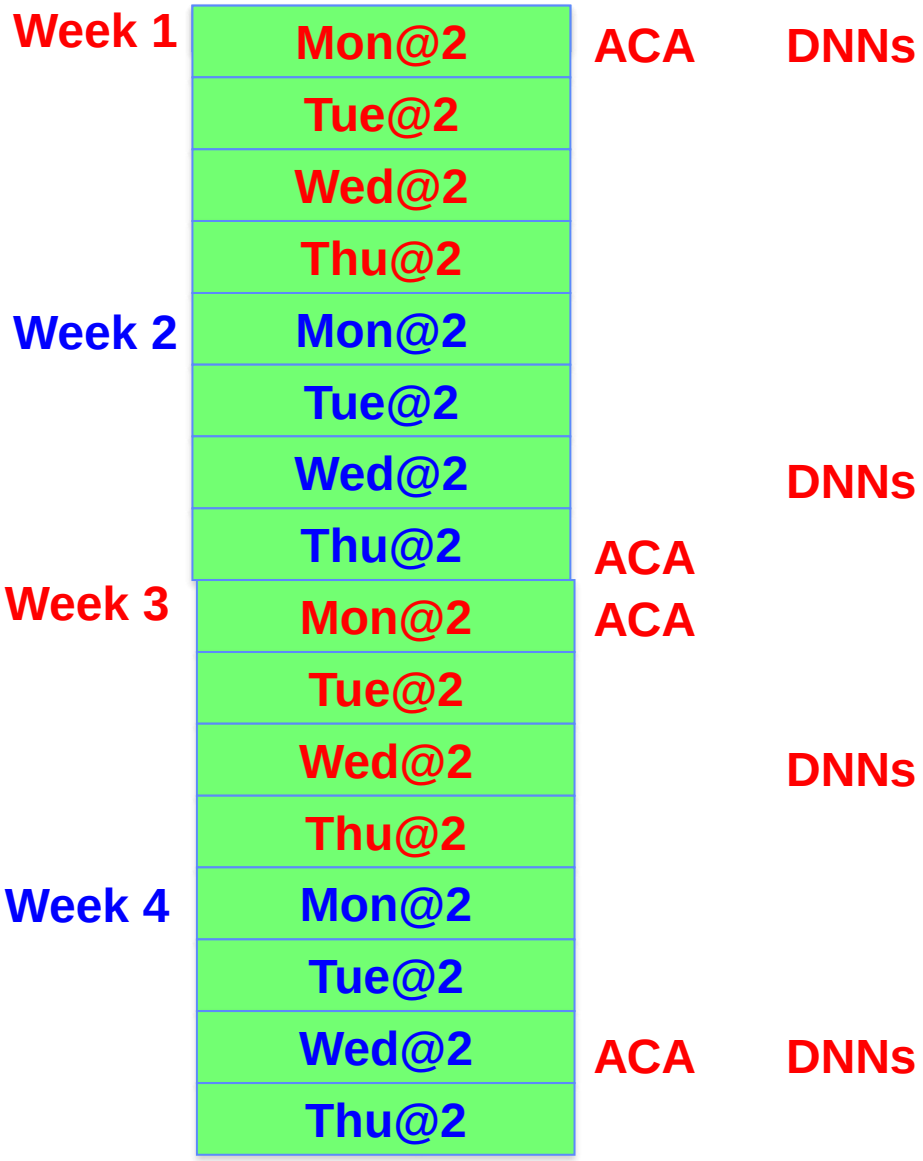
Note also the role of randomisation

- How to timetable all of DoC and EEE's 3rd-year, 4th-year and MSc courses
- With limited number of rooms and times in the week
- There must be some clashes

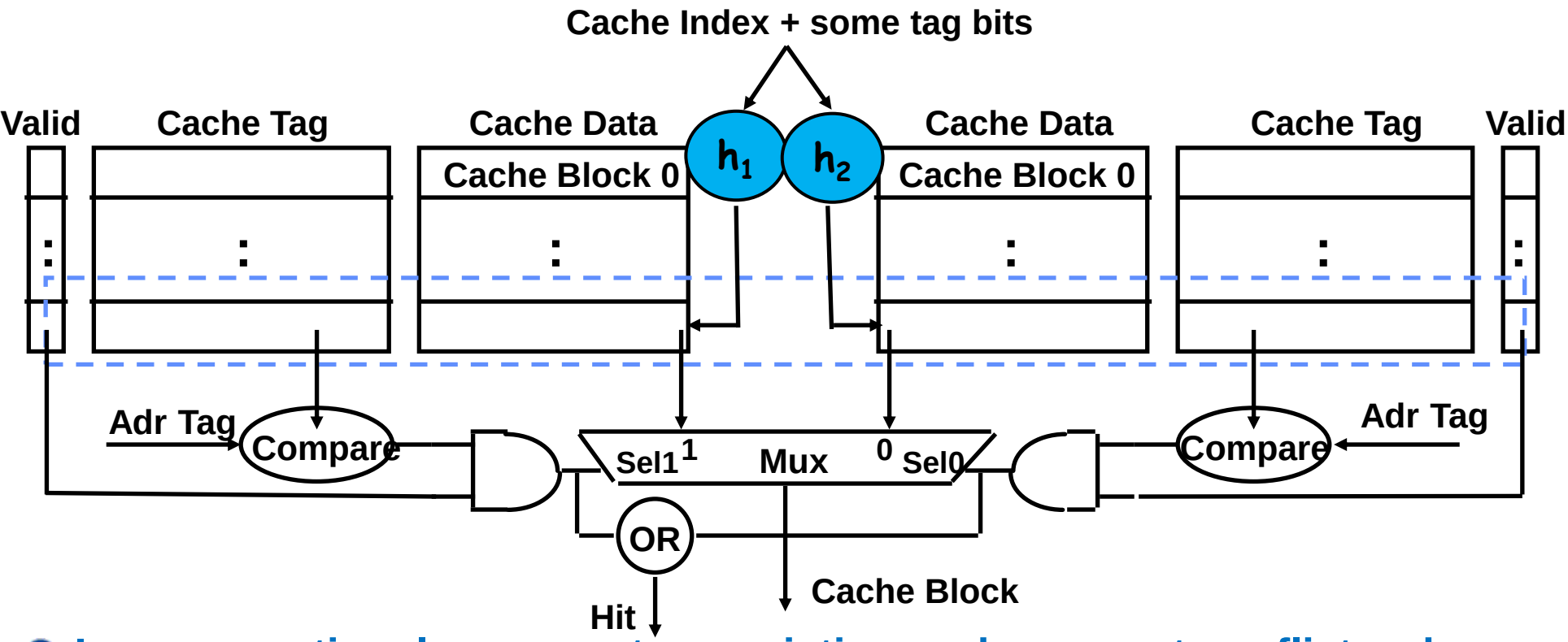
- Suppose you want to take two courses, "ACA" and "DNNs"
- If you're lucky they are scheduled on different slots
- If not, they clash every week!



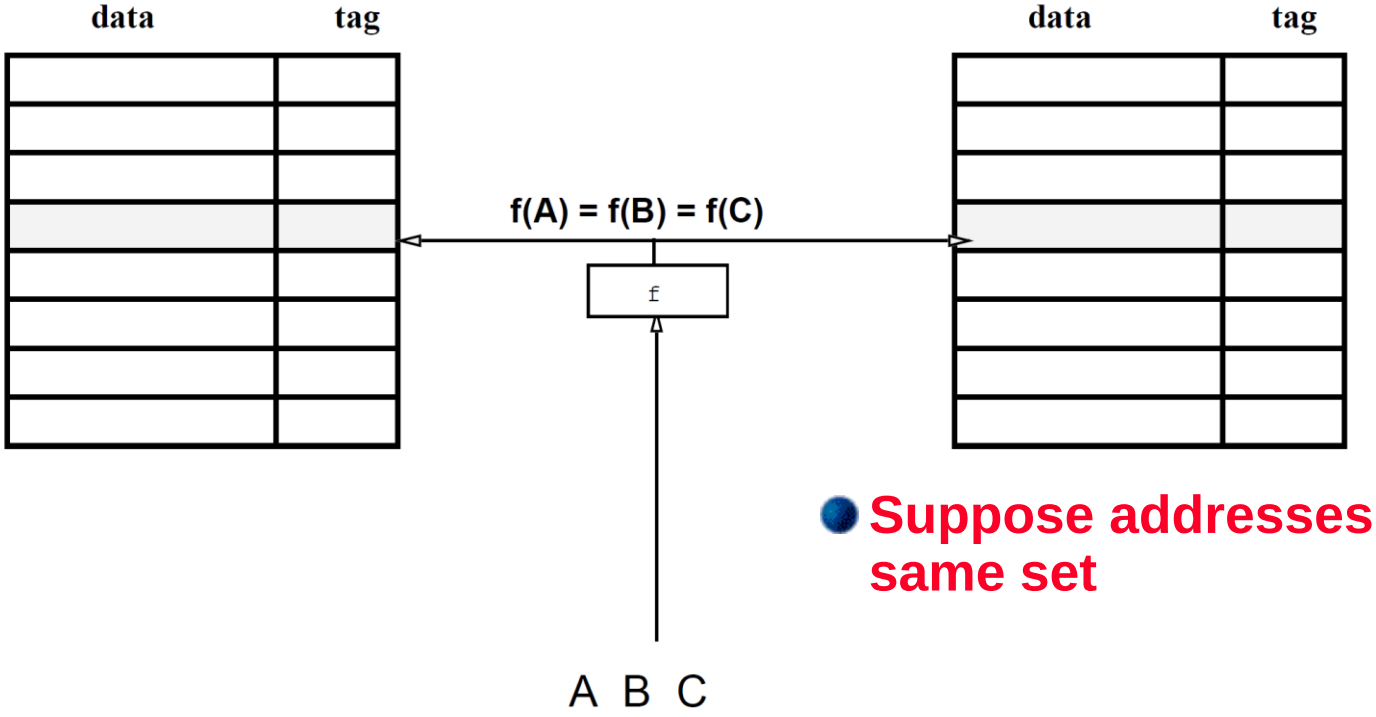
- How to timetable all of DoC and EEE's 3rd-year, 4th-year and MSc courses
- With limited number of rooms and times in the week
- There must be some clashes
- Suppose you want to take two courses, "ACA" and "DNNs"
- If you're lucky they are scheduled on different slots
- If not, they clash every week!
- Let's rehash every week….



Skewed-associative caches

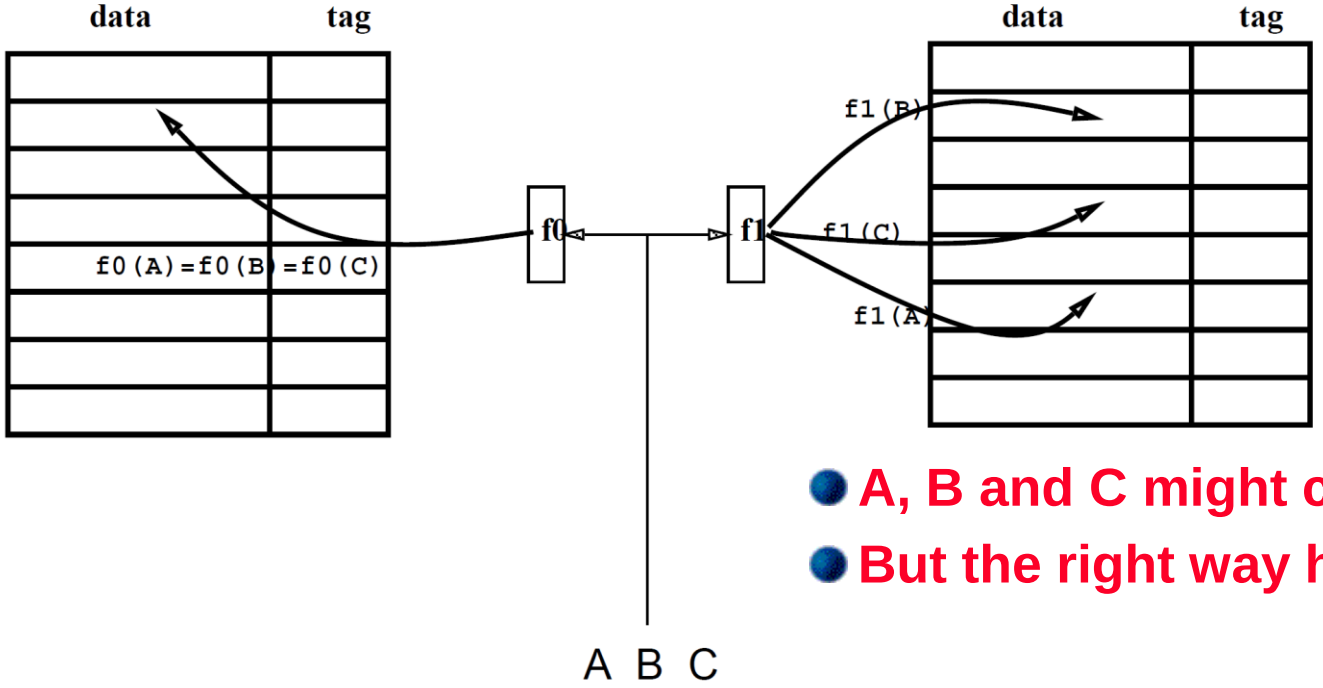


- In a conventional *w*-way set-associative cache, we get conflicts when *n*+1 blocks have the same address index bits
- Idea: reduce conflict misses by using *different* indices in each cache way
  - We introduce simple hash function,
    - Eg XOR some index bits with tag bits and reorder index bits



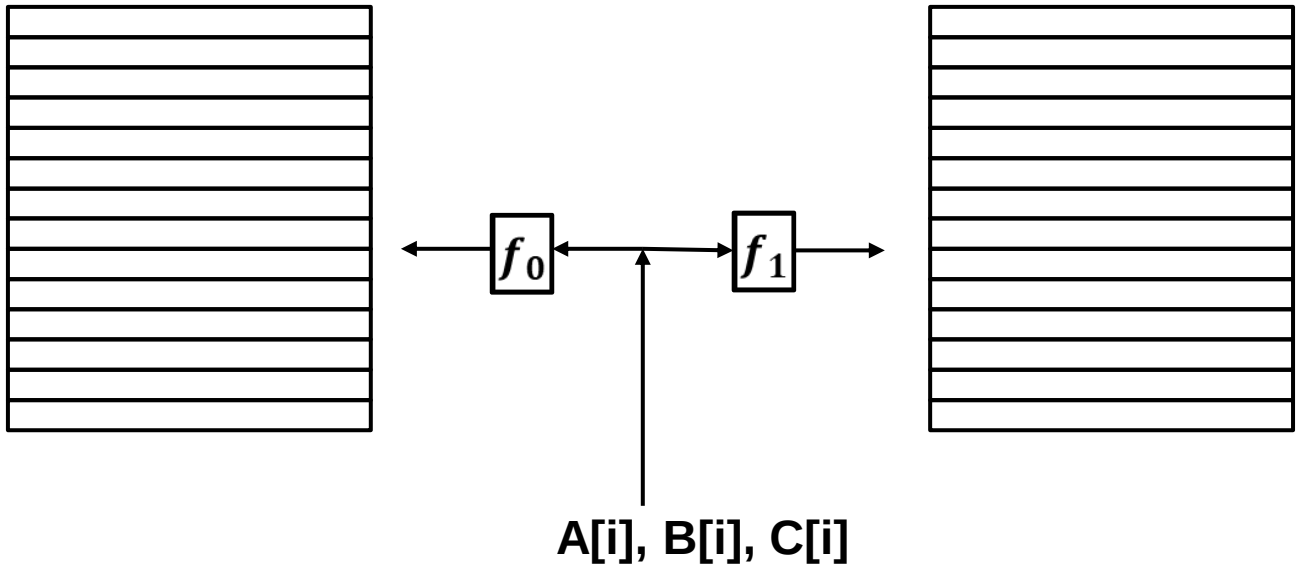
Conventional two-way set-associative

- Suppose addresses A, B and C map to the same set



Skewed two-way set-associative

- A, B and C might conflict in the left way
- But the right way has a different mapping



Skewed-associative caches: loops and arrays

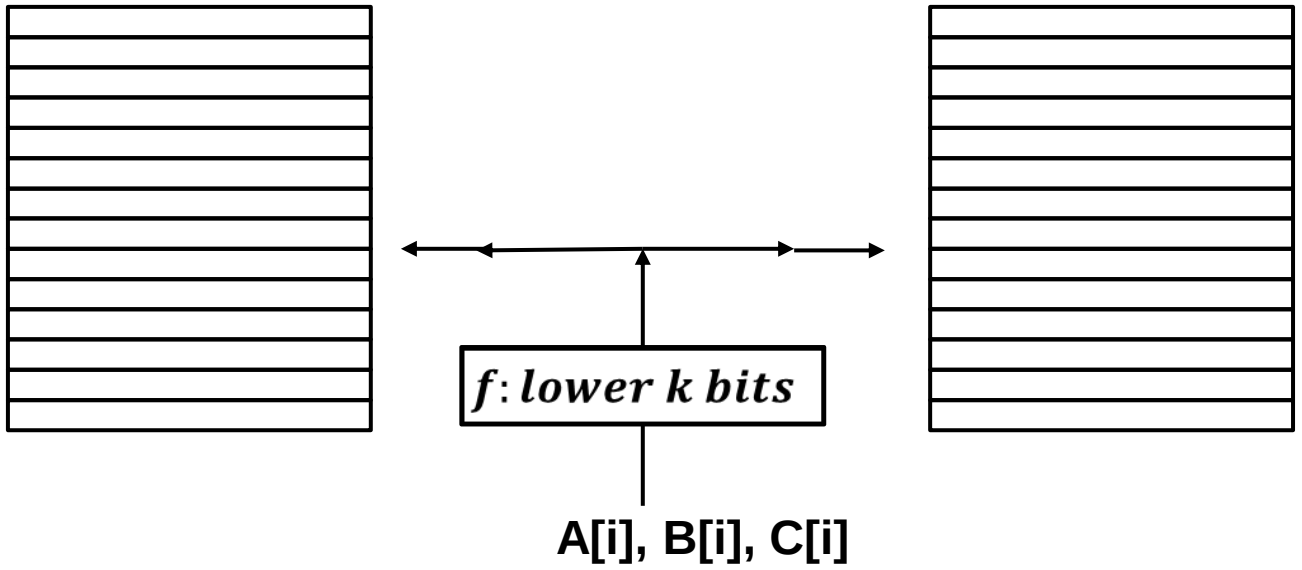
- Suppose we are traversing three arrays A, B and C:
  - Suppose we are unlucky:

    $f_0(A[i])=f_0(B[i])=f_0(C[i])$ *and* $f_1(A[i])=f_1(B[i])=f_1(C[i])$

    we get a conflict – only two of the three values can be in the cache at the same time
  - But since f_0 and f_1 are pseudo-random, it's unlikely that

    $f_0(A[i+1])=f_0(B[i+1])=f_0(C[i+1])$ *and* $f_1(A[i+1])=f_1(B[i+1])=f_1(C[i+1])$

Seznec, A. and Bodin, F. 1993. Skewed-associative Caches. In *Proceedings of the 5th international PARLE Conference on Parallel Architectures and Languages Europe* (June 14 - 17, 1993). A. Bode, M. Reeve, and G. Wolf, Eds. Lecture Notes In Computer Science, vol. 694. Springer-Verlag, London, 304-316.



In contrast 2-way set-associative cache

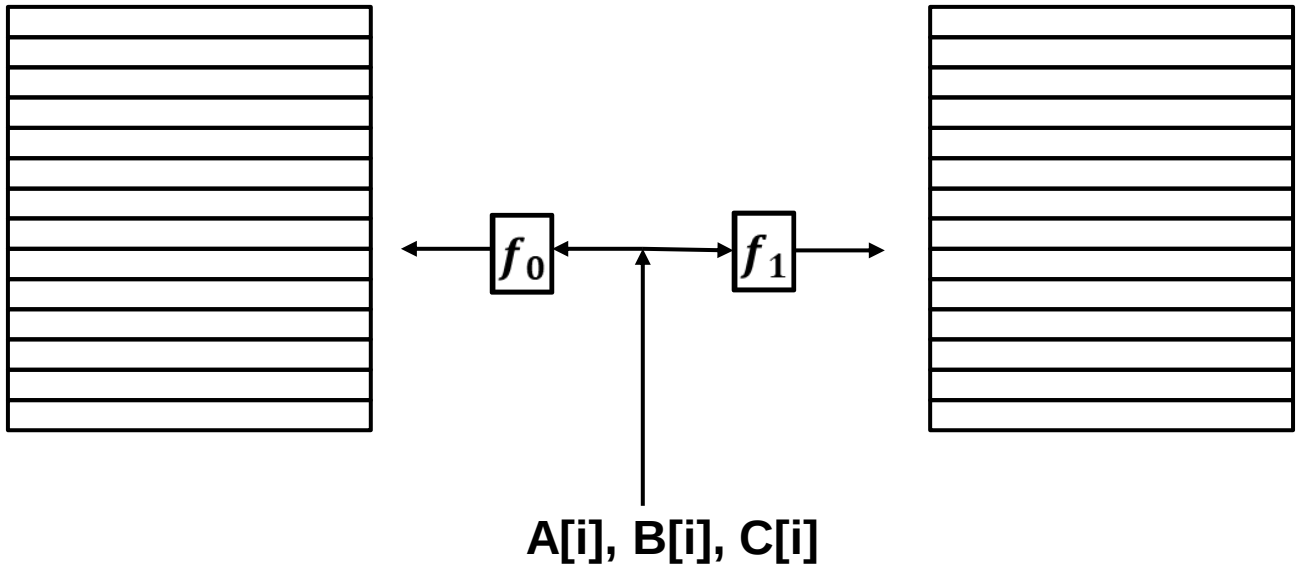
- Suppose we are traversing three arrays A, B and C:
  - We can easily be unlucky, eg due to power-of-2 alignment:

    $f(A[i])=f(B[i])=f(C[i])$

    So we get an associativity conflict – only two of the three values can be in the cache at the same time
  - And if that happens, we definitely get a conflict on next iteration:

    $f(A[i+1])=f(B[i+1])=f(C[i+1])$

Seznec, A. and Bodin, F. 1993. Skewed-associative Caches. In *Proceedings of the 5th international PARLE Conference on Parallel Architectures and Languages Europe* (June 14 - 17, 1993). A. Bode, M. Reeve, and G. Wolf, Eds. Lecture Notes In Computer Science, vol. 694. Springer-Verlag, London, 304-316.

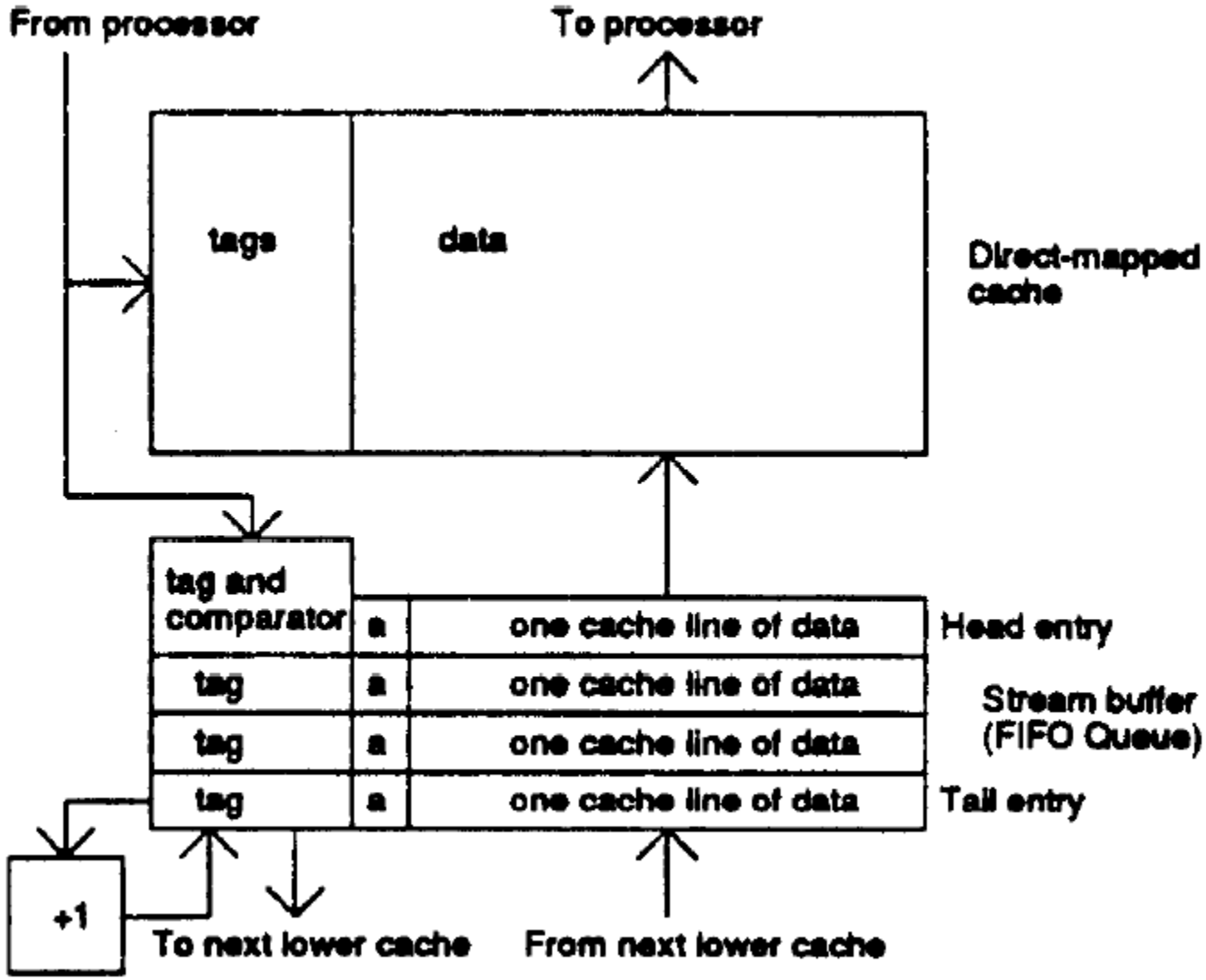


- We may be able to reduce associativity
- We have more predictable *average* performance
- It's hard to write a program that is free of associativity conflicts
- Costs?
  - One address decoder per way
  - Latency of hash function (?)
  - difficulty of implementing LRU
  - index hash uses *translated* bits [see later].

P. Michaud. 2003. A statistical model of skewed-associativity. In Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '03).

Reducing Misses by Hardware Prefetching of Instructions & Data

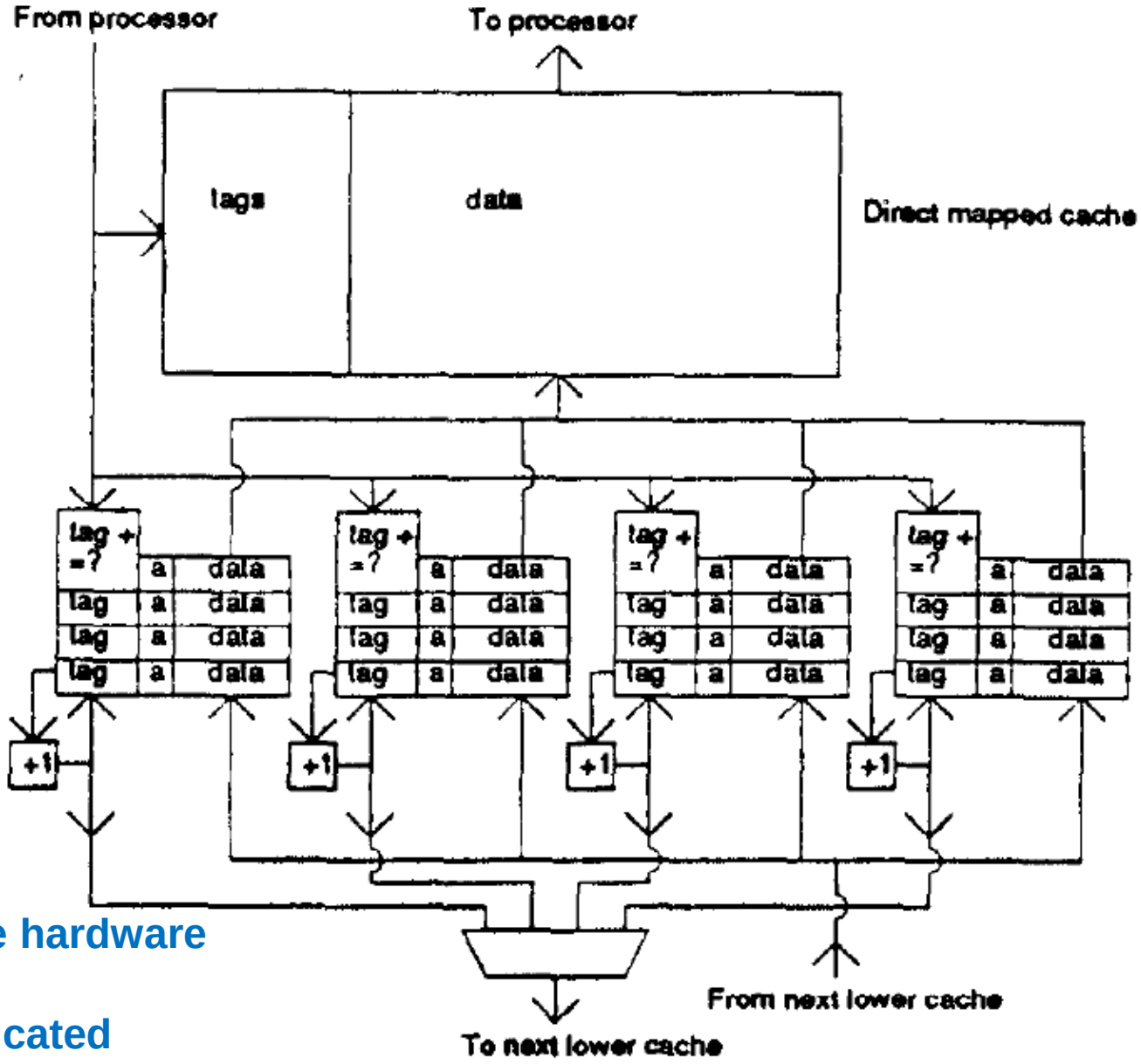
- Extra block placed in "stream buffer"
- After a cache miss, stream buffer initiates fetch for *next* block
- But it is not allocated into cache – to avoid "pollution"
- On access, check stream buffer in parallel with cache
- relies on having extra memory bandwidth



From Jouppi [1990]: http://doi.acm.org/10.1145/285930.285998

Multi-way stream-buffer

- We can extend this idea to track multiple access streams simultaneously:
  - One stream is good for instruction-cache misses
  - Multiple streams often important for data
    - Eg traversing multiple arrays
  - *[[Q: would it be better to prefetch n+k instead of n+1?]]*

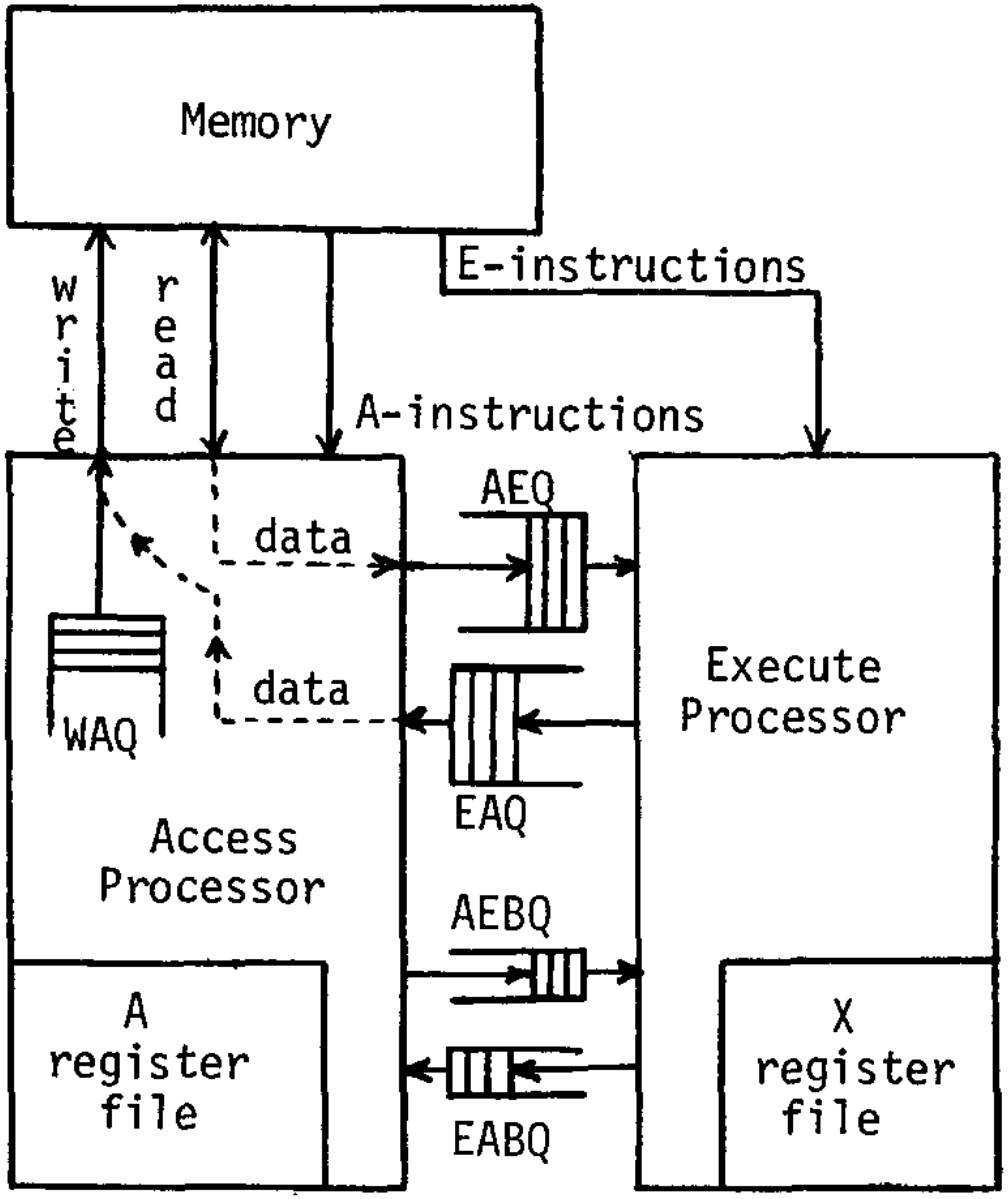


Many (many) modern CPUs have hardware prefetching

- Often more elaborate/sophisticated
- Initiated at L1, or perhaps initiated on L2 misses?

Beyond prefetch: decoupled access-execute

- Idea: separate the instructions that generate addresses from the instructions that use memory results
- Let the address-generation side of the machine run ahead



From James E. Smith. 1982. Decoupled access/execute computer architectures. In Proceedings of the 9th annual symposium on Computer Architecture (ISCA '82). IEEE Computer Society Press, Los Alamitos, CA, USA, 112-119

See also ACRI supercomputer project, http://www.paralogos.com/DeadSuper/ACRI/

And Scout threads in Sun's Rock: https://ieeexplore.ieee.org/document/4523067

Summary

We can reduce the miss rate through hardware…..

- **With a bigger cache (Capacity)**
  - **But a bigger cache will be slower, or will have to be pipelined**
- **With larger blocks (aka cachelines)**
  - **But if that increases the miss penalty, you lose**
- **With higher associativity (Conflicts)**
  - **But direct-mapped caches are (a bit) faster**
- **We can reduce the miss rate due to associativity conflicts by adding a victim cache**
- **We can reduce the miss rate due to associativity conflicts using a skewed-associative cache (reduce… on average?)**
- **We can reduce miss *delays* by prefetching using a prefetch predictor and a stream buffer**
- **We can reduce miss *delays* by issuing loads early enough, for example in a decoupled architecture**

Further reading

We have not discussed replacement policy

- Some theory eg
  - Pierre Michaud. Some mathematical facts about optimal cache replacement. ACM Transactions on Architecture and Code Optimization, Association for Computing Machinery, 2016, 13 (4), ff10.1145/3017992ff. ffhal-01411156v2f
- Fast cheap hardware for approximating LRU:
  - Pseudo-LRU https://en.wikipedia.org/wiki/Pseudo-LRU
  - What does the pessimal replacement policy look like?
  - See https://link.springer.com/chapter/10.1007/978-3-540-72914-3_13
  - From the wonderful Fun with Algorithms conference series
    - https://sites.google.com/view/fun2020/home
    - And entirely unrelated: http://www.toroidalsnark.net/mathknit.html

Piazza question: stride and depth in prefetching

"Stride" is the size of the pointer increment on each access, eg in

```
double A[], B[]; // 8-byte per word
for (int i=0; i<N; ++i)
  B[i] = A[3*i];
```

the load has stride 24bytes, while the store has stride 8bytes.

"Depth" concerns how many iterations ahead we prefetch. Eg

```
for (int i=0; i<N; ++i)
  prefetch(&A[i+D];
  B[i] = A[i]+s;
```

D is the prefetch depth. It's often a good idea for D to be bigger than one, in order to get multiple accesses in flight and to cover the memory access latency.