CS 61C: Great Ideas in Computer Architecture
Caches Part 3

Instructors:

Nicholas Weaver & Vladimir Stojanovic

http://inst.eecs.berkeley.edu/~cs61c

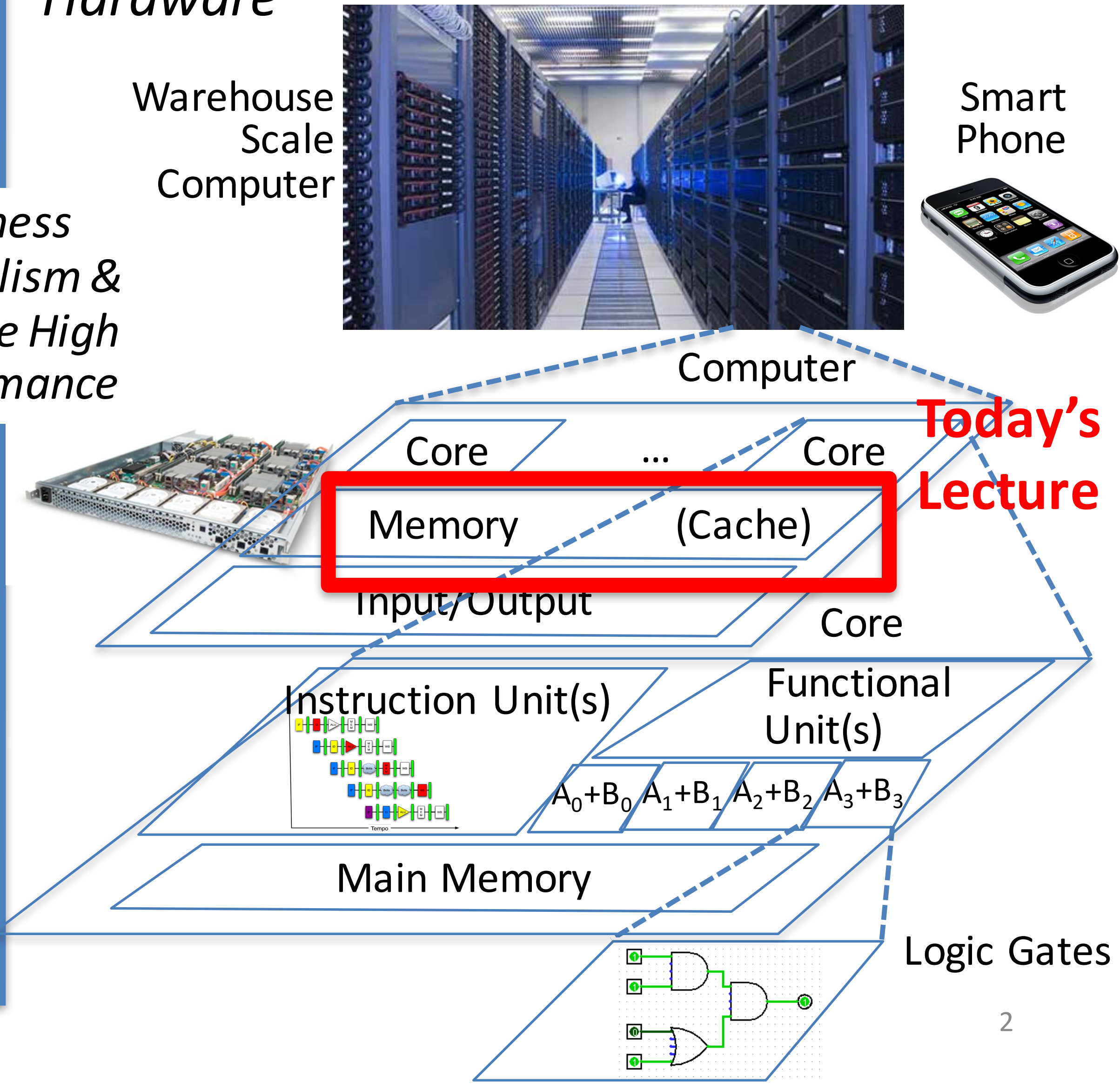
You Are Here!

Software

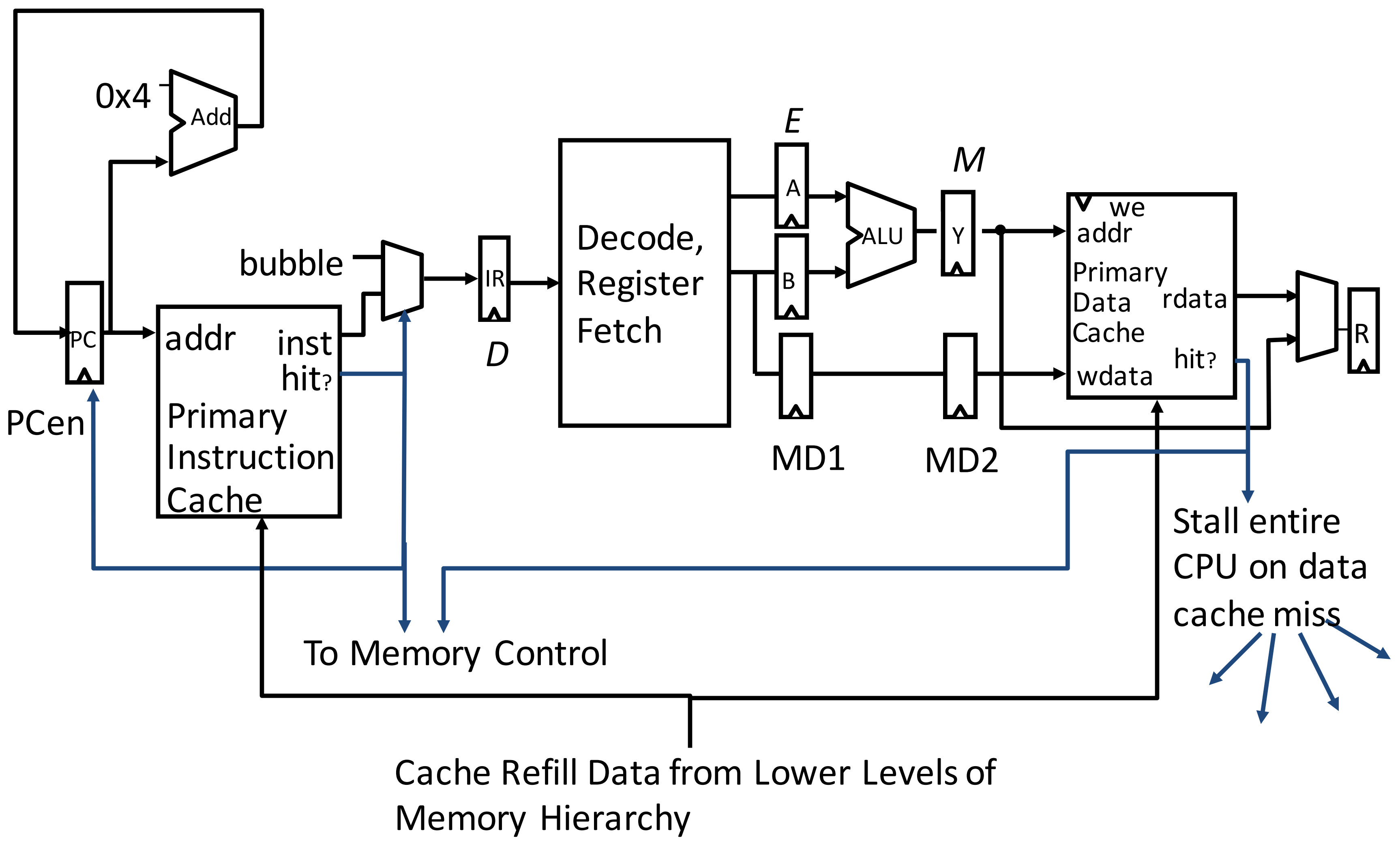
- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Hardware

Harness Parallelism & Achieve High Performance



CPU-Cache Interaction
(5-stage pipeline)



Improving Cache Performance

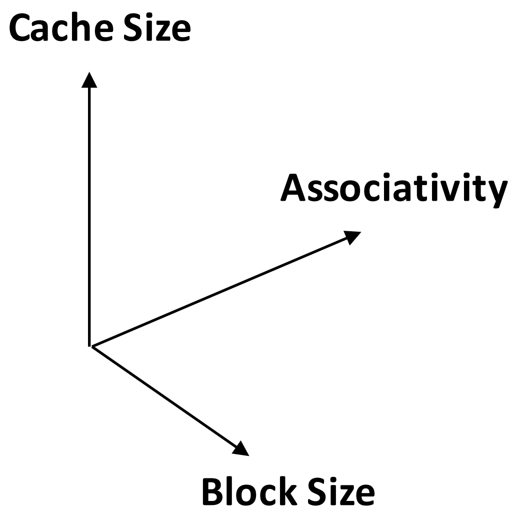
AMAT = Time for a hit + Miss rate x Miss penalty

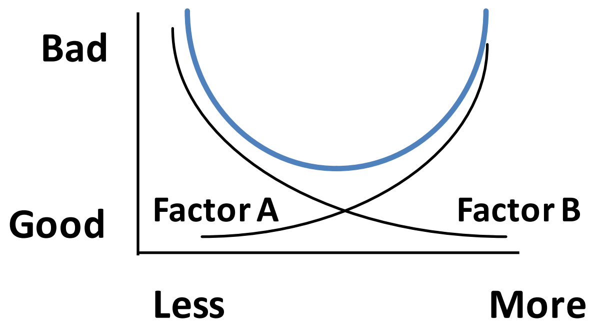
- Reduce the time to hit in the cache
  - E.g., Smaller cache
- Reduce the miss rate
  - E.g., Bigger cache
    Longer cache lines (somewhat)
- Reduce the miss penalty
  - E.g., Use multiple cache levels

Cache Design Space

Computer architects expend considerable effort optimizing organization of cache hierarchy – big impact on performance and power!

- Several interacting dimensions
  - **Cache size**
  - **Block size**
  - **Associativity**
  - Replacement policy
  - Write-through vs. write-back
  - Write allocation
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache)
  - Depends on technology / cost
- Simplicity often wins





Primary Cache Parameters

- Block size
  - how many bytes of data in each cache entry?
- Associativity
  - how many ways in each set?
  - Direct-mapped => Associativity = 1
  - Set-associative => 1 < Associativity < #Entries
  - Fully associative => Associativity = #Entries
- Capacity (bytes) = Total #Entries * Block size
- #Entries = #Sets * Associativity

Clickers/Peer Instruction:
For fixed capacity and fixed block size, how does increasing associativity effect AMAT?

A: Increases hit time, decreases miss rate

B: Decreases hit time, decreases miss rate

C: Increases hit time, increases miss rate

D: Decreases hit time, increases miss rate

Increasing Associativity?

- Hit time as associativity increases?
  - Increases, with large step from direct-mapped to >=2 ways, as now need to mux correct way to processor
  - Smaller increases in hit time for further increases in associativity
- Miss rate as associativity increases?
  - Goes down due to reduced conflict misses, but most gain is from 1->2->4-way with limited benefit from higher associativities
- Miss penalty as associativity increases?
  - Unchanged, replacement policy runs in parallel with fetching missing line from memory

Increasing #Entries?

- Hit time as #entries increases?
  - Increases, since reading tags and data from larger memory structures
- Miss rate as #entries increases?
  - Goes down due to reduced capacity and conflict misses
  - *Architects rule of thumb: miss rate drops ~2x for every ~4x increase in capacity (only a gross approximation)*
- Miss penalty as #entries increases?
  - Unchanged

At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance

Clickers: Impact of larger blocks on AMAT

- For fixed total cache capacity and associativity, what is effect of larger blocks on each component of AMAT:
  - A: Decrease, B: Unchanged, C: Increase
- Hit Time?
- Miss Rate?
- Miss Penalty?

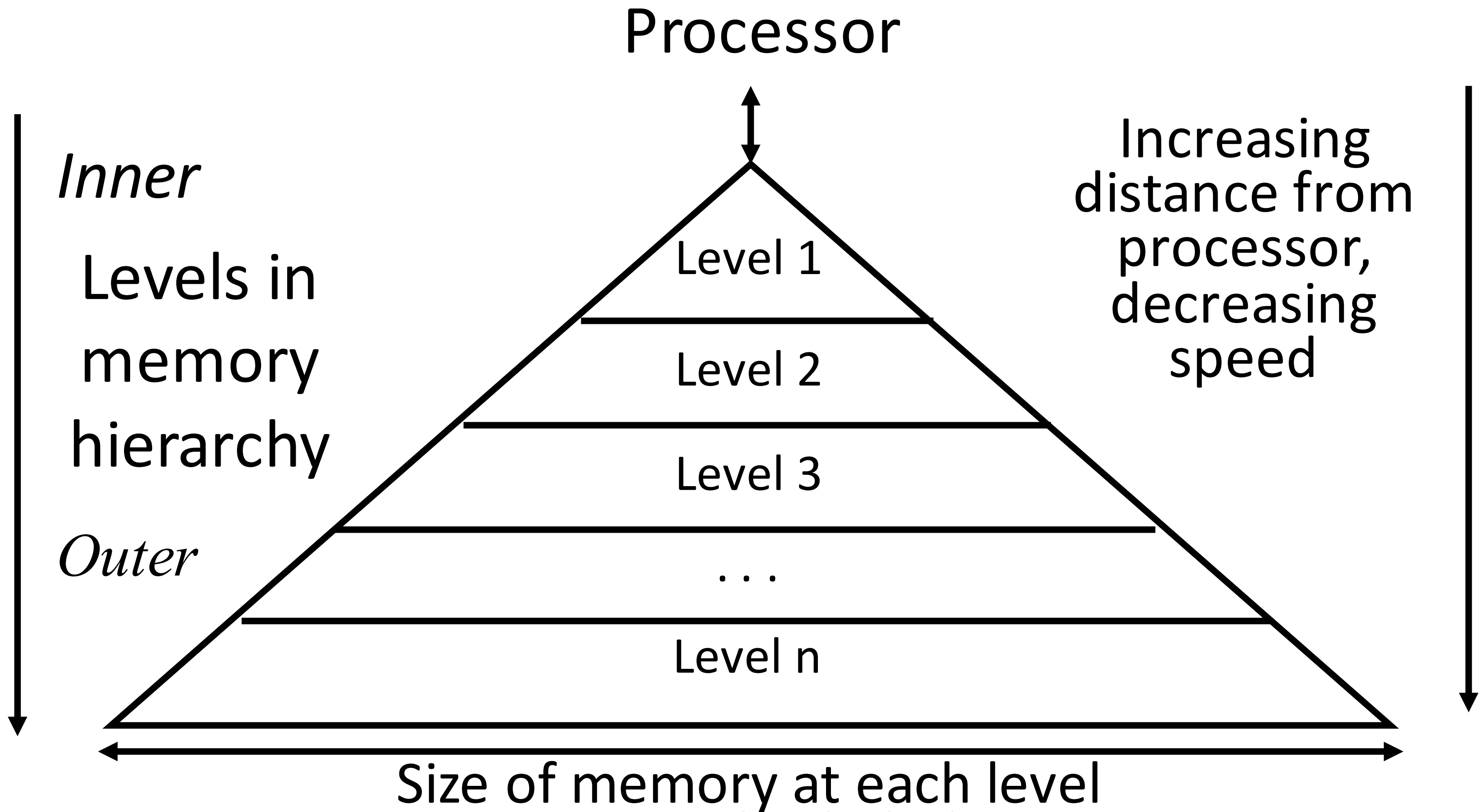
Increasing Block Size?

- Hit time as block size increases?
  - Hit time unchanged, but might be slight hit-time reduction as number of tags is reduced, so faster to access memory holding tags
- Miss rate as block size increases?
  - Goes down at first due to spatial locality, then increases due to increased conflict misses due to fewer blocks in cache
- Miss penalty as block size increases?
  - Rises with longer block size, but with fixed constant initial latency that is amortized over whole block

How to Reduce Miss Penalty?

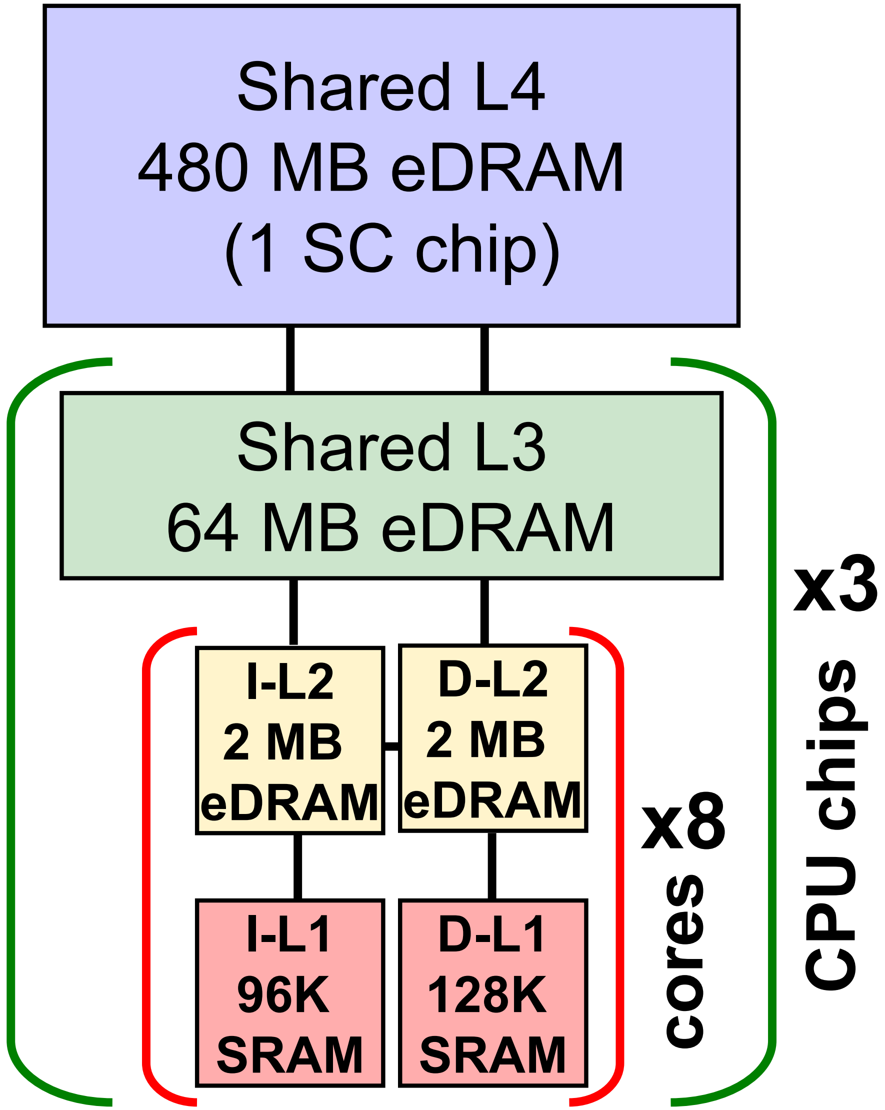
- Could there be locality on misses from a cache?
- Use multiple cache levels!
- With Moore's Law, more room on die for bigger L1 caches and for second-level (L2) cache
- And in some cases even an L3 cache!
- IBM mainframes have ~1GB L4 cache off-chip.

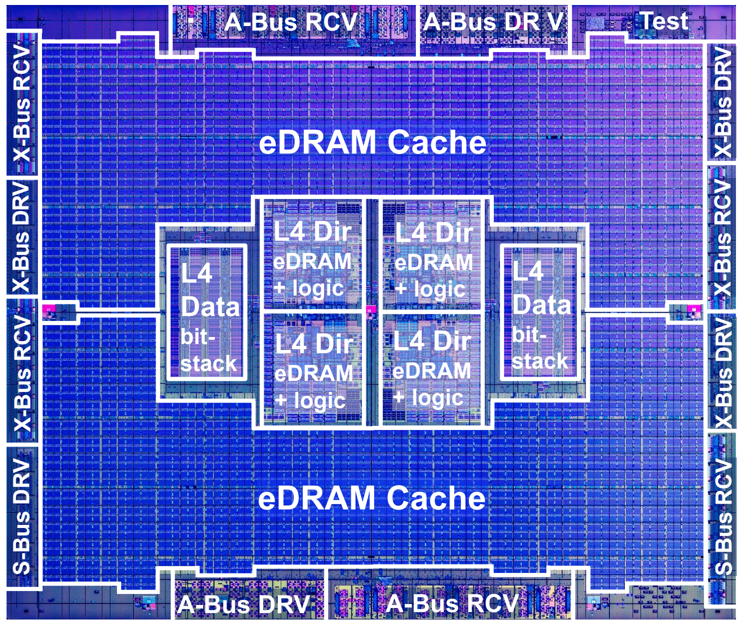
Review: Memory Hierarchy

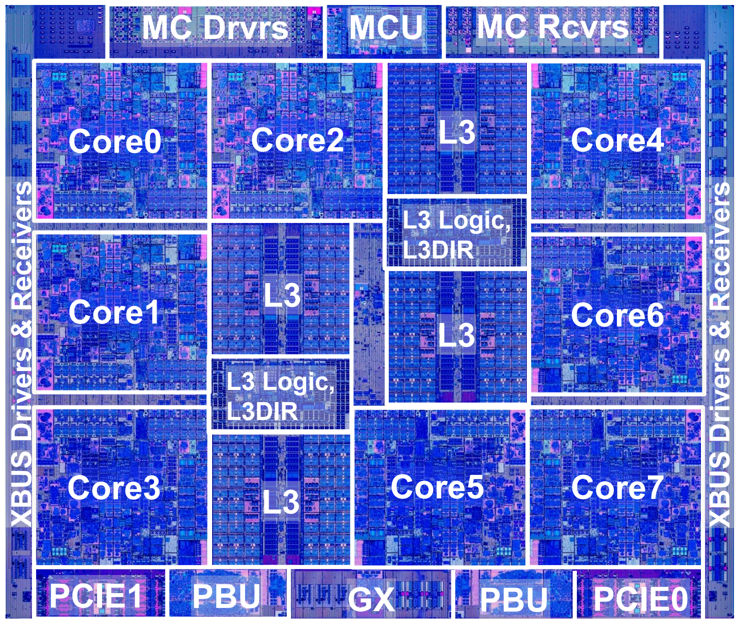

Processor
Inner
Levels in memory hierarchy
Outer
Level 1
Level 2
Level 3
. . .
Level n
Increasing distance from processor, decreasing speed
Size of memory at each level

As we move to outer levels the latency goes up and price per bit goes down.

IBM z13 Memory Hierarchy







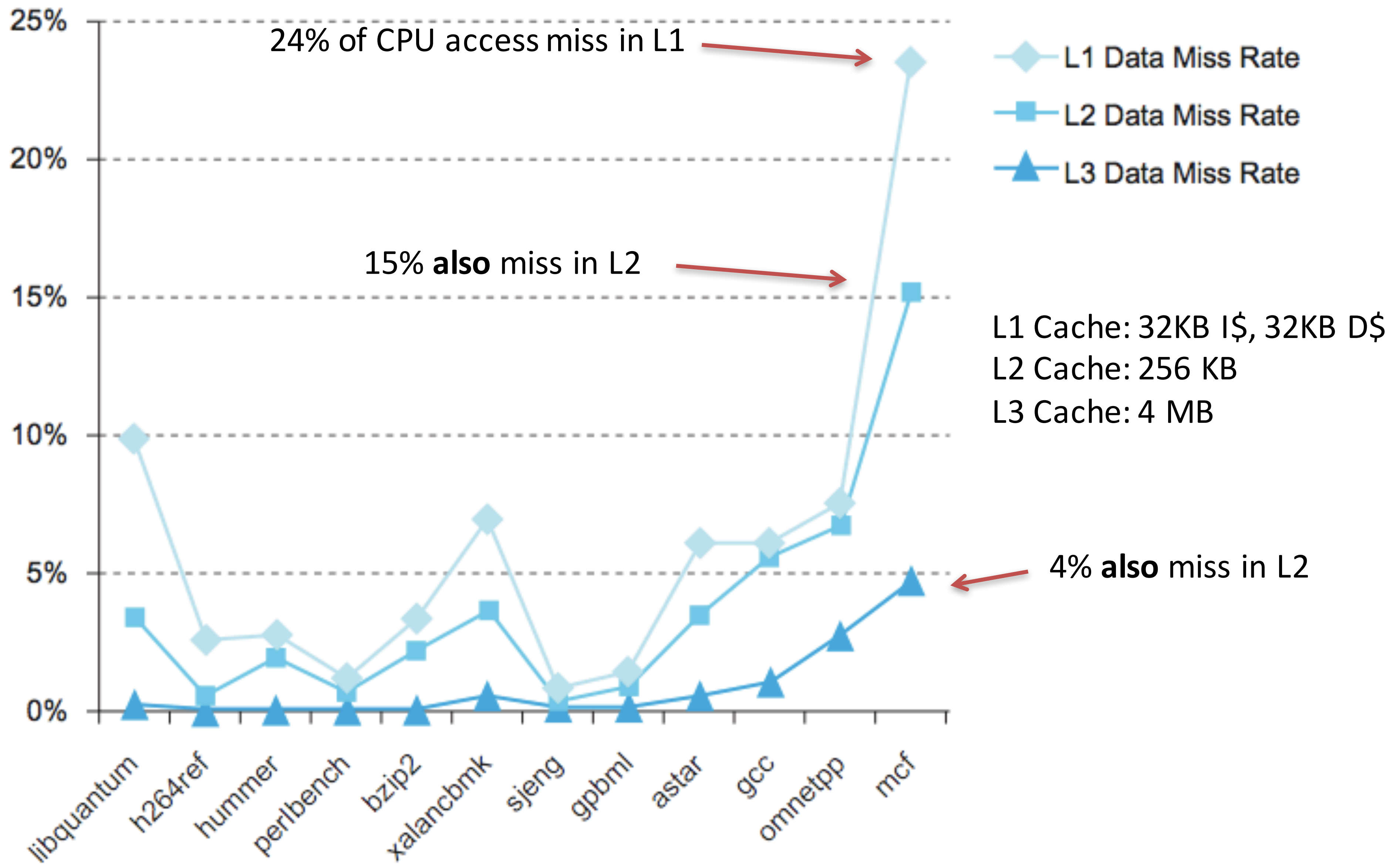


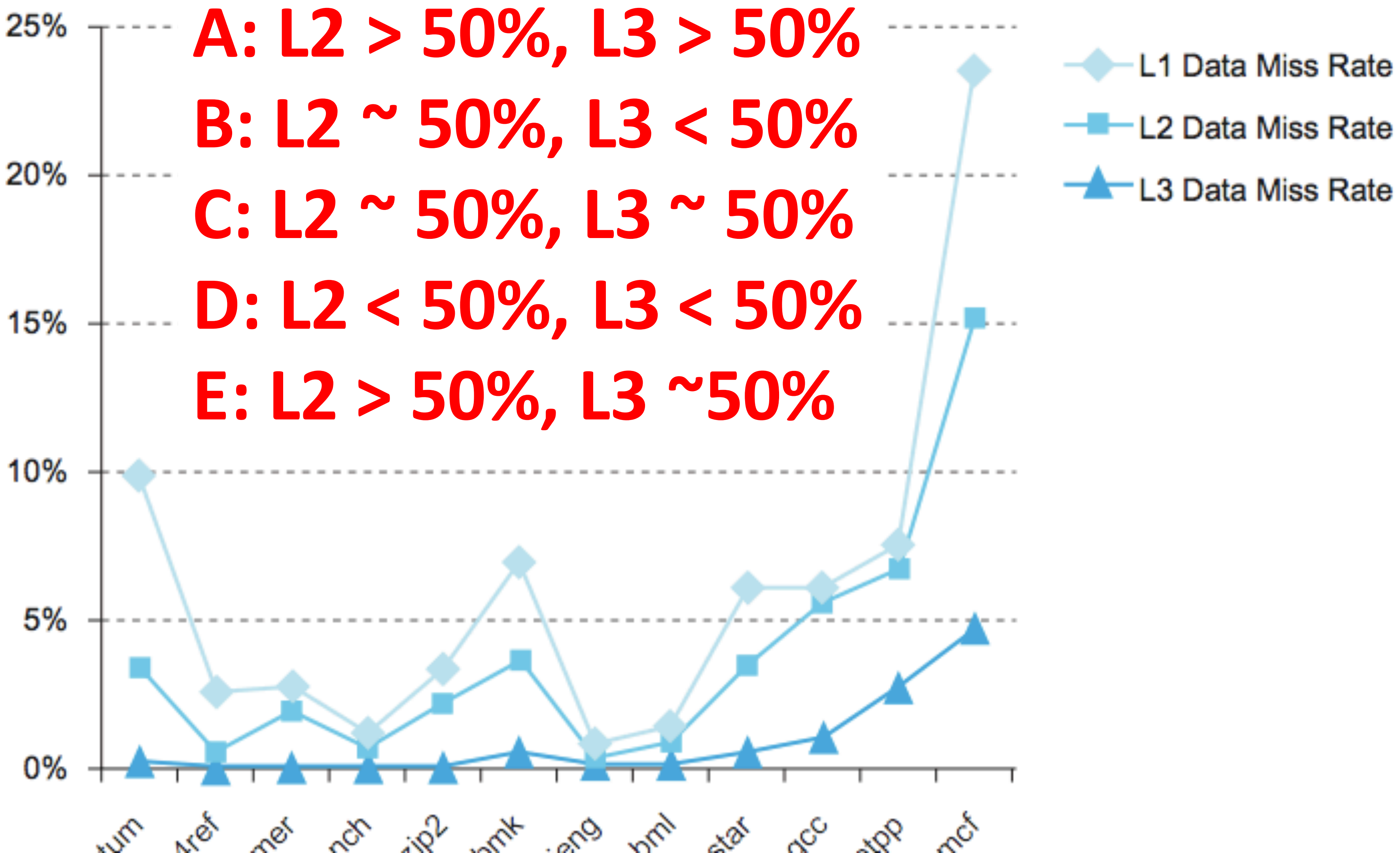
FIGURE 5.47 The L1, L2, and L3 data cache miss rates for the Intel Core i7 920 running the full integer SPECCPU2006 benchmarks.

Local vs. Global Miss Rates

- *Global miss rate* – the fraction of references that miss some level of a multilevel cache
  - *misses in this cache divided by the total number of memory accesses generated by the CPU*
- *Local miss rate* – the fraction of references to one level of a cache that miss
- Local Miss rate L2$ = L2$ Misses / L1$ Misses
  = L2$ Misses / total_L2_accesses
- L2$ local miss rate >> than the global miss rate

Clickers/Peer Instruction

- Overall, what are L2 and L3 local miss rates?



A: L2 > 50%, L3 > 50%

B: L2 ~ 50%, L3 < 50%

C: L2 ~ 50%, L3 ~ 50%

D: L2 < 50%, L3 < 50%

E: L2 > 50%, L3 ~50%

Local vs. Global Miss Rates

- *Local miss rate* – the fraction of references to one level of a cache that miss
- Local Miss rate L2$ = $L2 Misses / L1$ Misses
- *Global miss rate* – the fraction of references that miss in all levels of a multilevel cache
  - L2$ local miss rate >> than the global miss rate
- Global Miss rate = L2$ Misses / Total Accesses

  = (L2$ Misses / L1$ Misses) × (L1$ Misses / Total Accesses)

  = Local Miss rate L2$ × Local Miss rate L1$

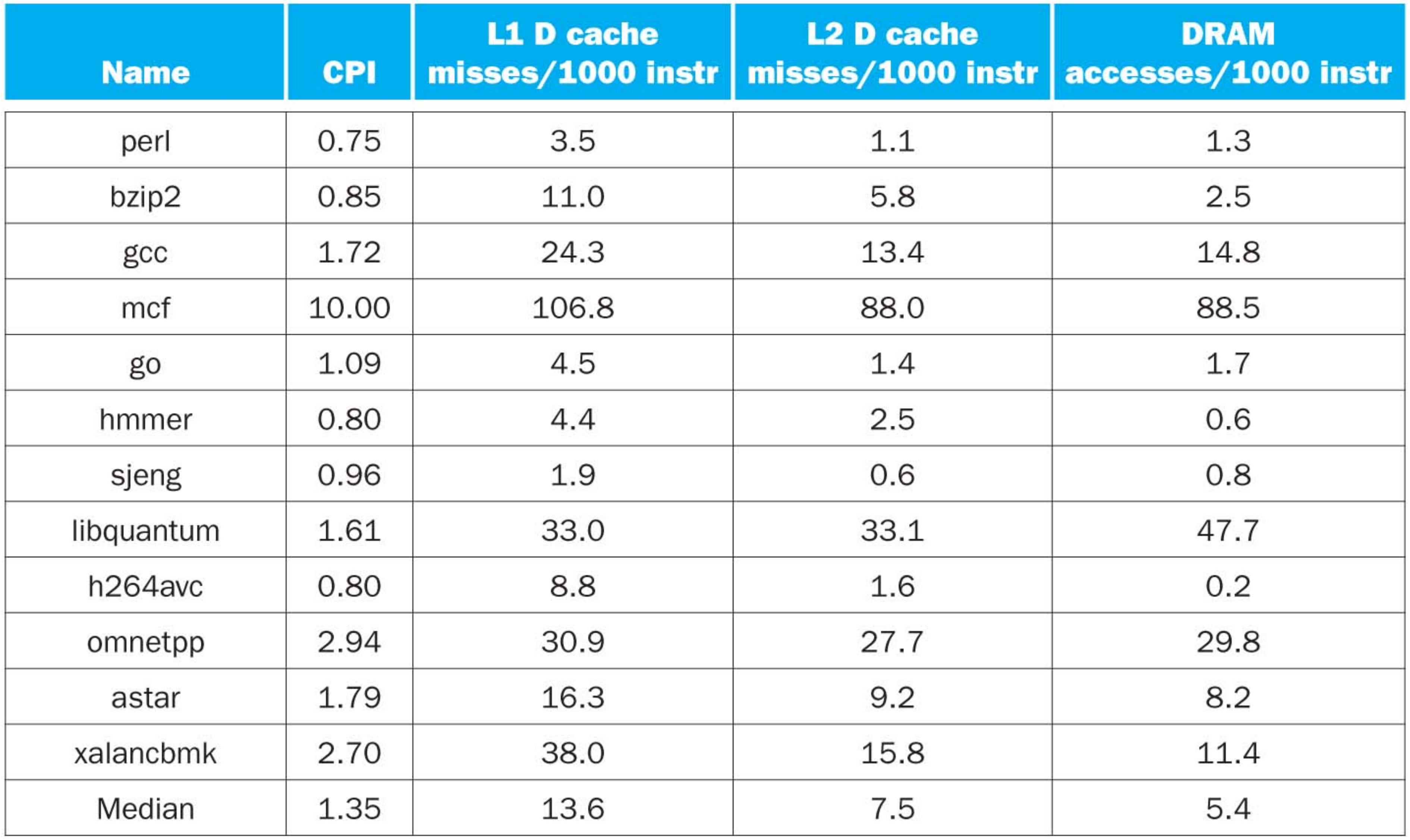
- AMAT = Time for a hit + Miss rate × Miss penalty
- For 2-level cache system:

  AMAT = Time for a L1$ hit + Miss rate L1$ ×

(Time for a L2$ hit + (local) Miss rate L2$ × L2$ Miss penalty)

Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

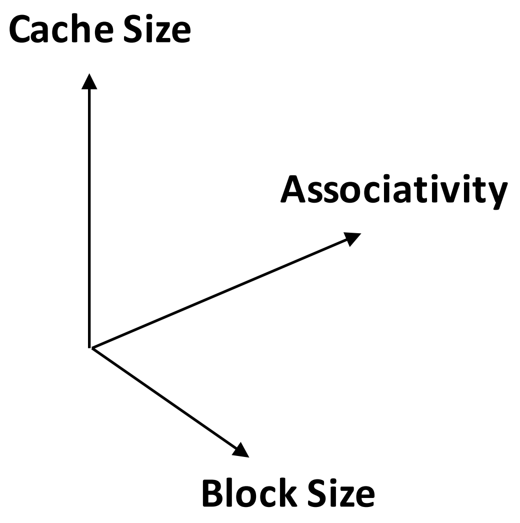
CPI/Miss Rates/DRAM Access SpecInt2006

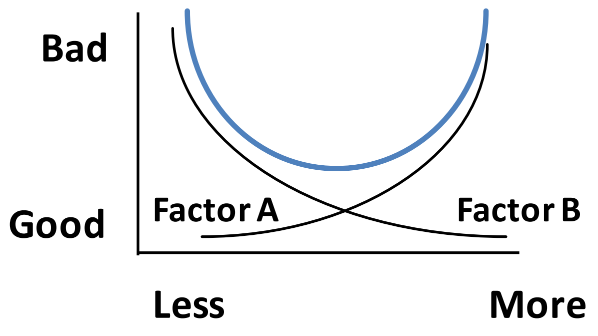


		Data Only	Data Only	Instructions and Data
Name	**CPI**	**L1 D cache misses/1000 instr**	**L2 D cache misses/1000 instr**	**DRAM accesses/1000 instr**
perl	0.75	3.5	1.1	1.3
bzip2	0.85	11.0	5.8	2.5
gcc	1.72	24.3	13.4	14.8
mcf	10.00	106.8	88.0	88.5
go	1.09	4.5	1.4	1.7
hmmer	0.80	4.4	2.5	0.6
sjeng	0.96	1.9	0.6	0.8
libquantum	1.61	33.0	33.1	47.7
h264avc	0.80	8.8	1.6	0.2
omnetpp	2.94	30.9	27.7	29.8
astar	1.79	16.3	9.2	8.2
xalancbmk	2.70	38.0	15.8	11.4
Median	1.35	13.6	7.5	5.4

In Conclusion, Cache Design Space

- Several interacting dimensions
  - Cache size
  - Block size
  - Associativity
  - Replacement policy
  - Write-through vs. write-back
  - Write-allocation
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache)
  - Depends on technology / cost
- Simplicity often wins





More Misses…

- We have ***Compulsory***, ***Capacity***, and ***Conflict***…
- We also have ***Coherence***
  - Two different processor may share memory...
    - They implement ***cache coherence*** so that both processors see the same ***shared memory***
    - When one processor writes to memory, it ***invalidates*** the other processor's cache entry for that memory
  - Thus if both processors are working on the same data…
    - This causes Coherence misses
- A related problem can occur if one shared cache is working on two ***unrelated*** problems
  - You get additional capacity misses: Can happen in "multithreaded" (aka 'Intel Hyperthreaded') processor cores

Fun Additional Stuff: Nick's Caches

- Note: These won't be on the exam, but they are interesting asides
  - Nick's research has used this material in multiple ways
- Predictability and caches
  - Why its bad
  - Unpredictable caches: Permutation caches and location-associative permutation caches

Predictability and Caches

- Caches improve performance but…
  - The performance improvement depends on the input
    - E.g. conflict misses depend on input patterns
- An attacker can take advantage of this
  - Timing of operations can tell something about the input
  - Attacker selected inputs can degrade performance

Why Timing Matters

- Timing enables "side-channel" attacks on cryptography
  - The ability to know some detail of an encryption system based on how long operations take
    - Part of a larger class of side-channel attacks
- It is a fundamentally hard problem to build cryptographic systems that don't have sidechanels
  - Modern processors make this even harder

Attacker Selected Input

- Alternatively, if the attacker can select the input…
  - The attacker can select ***hard*** input:
    E.G. Traffic that causes ping-ponging
- Nick's problem:
  - He had to cache IP addresses (32 bit values)
    - This is a network application for security
  - He only wants to store a small amount of information
    - On chip storage expensive (in this case, on an FPGA)

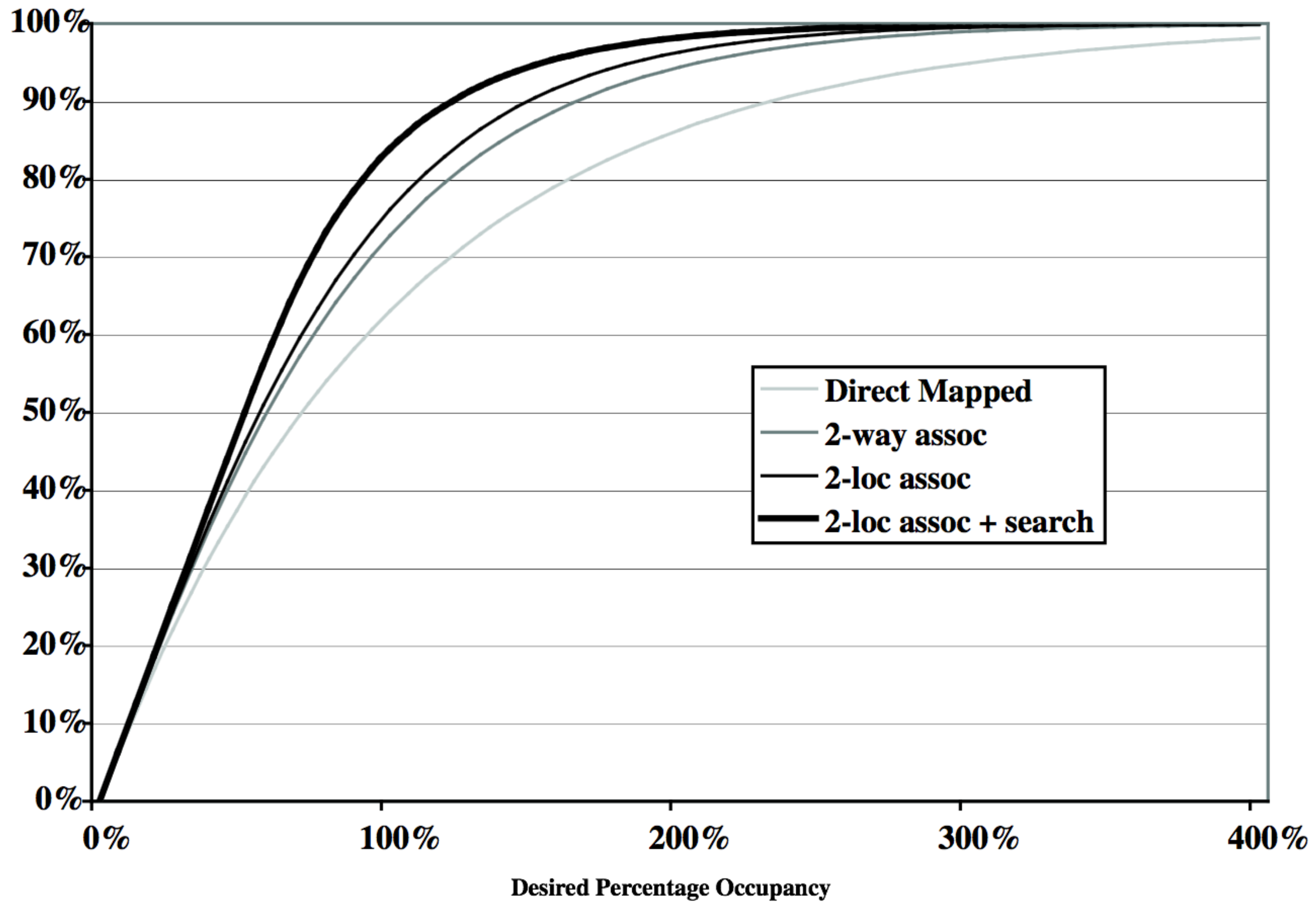
#1: Permutation Cache

- Traditionally, you would hash the address
  - With a "salt" to randomize things
  - But this requires storing the whole hash value or whole IP for your tag
- Instead of a hash, use a 32b keyed permutation
  - Aka a 32b block cypher
- Now you can use a conventional tag/index approach
  - Requires only storing the tag -> space mattered in this application

#2: Location Associativity

- The fabric Nick had used "dual-ported" memories
  - Like your register file on your processor design: two independent read ports
- Rather than using set associativity...
  - Instead do two different permutations (keys) and have one of two possible locations
- If X, Y, and Z map to the same location with one key...
  - They probably ***do not*** on the other key: fewer ***conflict*** misses
  - Even better, can probably ***move*** a value to further reduce ***conflict*** misses

Simulation…


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
0%
100%
200%
300%
400%
Desired Percentage Occupancy
Direct Mapped
2-way assoc
2-loc assoc
2-loc assoc + search