CS 110
Computer Architecture
Performance and Floating Point Arithmetic

Instructor:
Sören Schwertfeger

http://shtech.org/courses/ca/

School of Information Science and Technology SIST

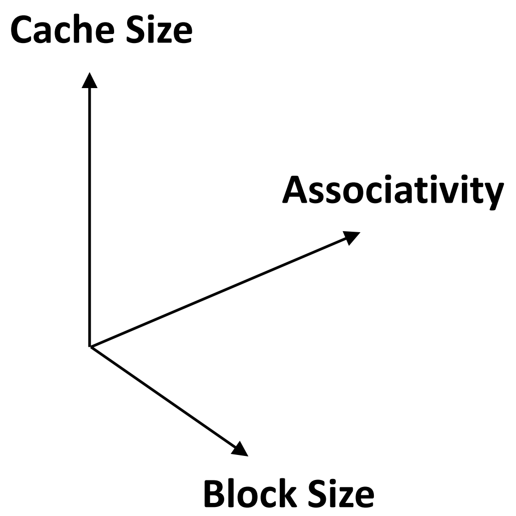
ShanghaiTech University

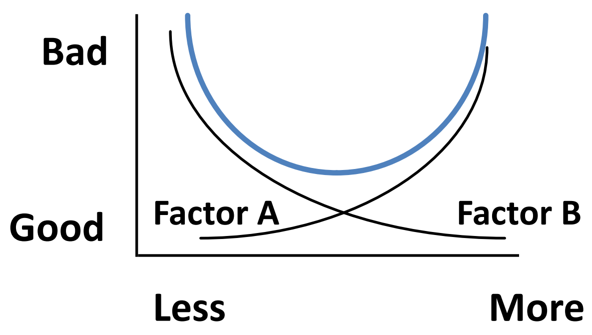
Slides based on UC Berkley's CS61C

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

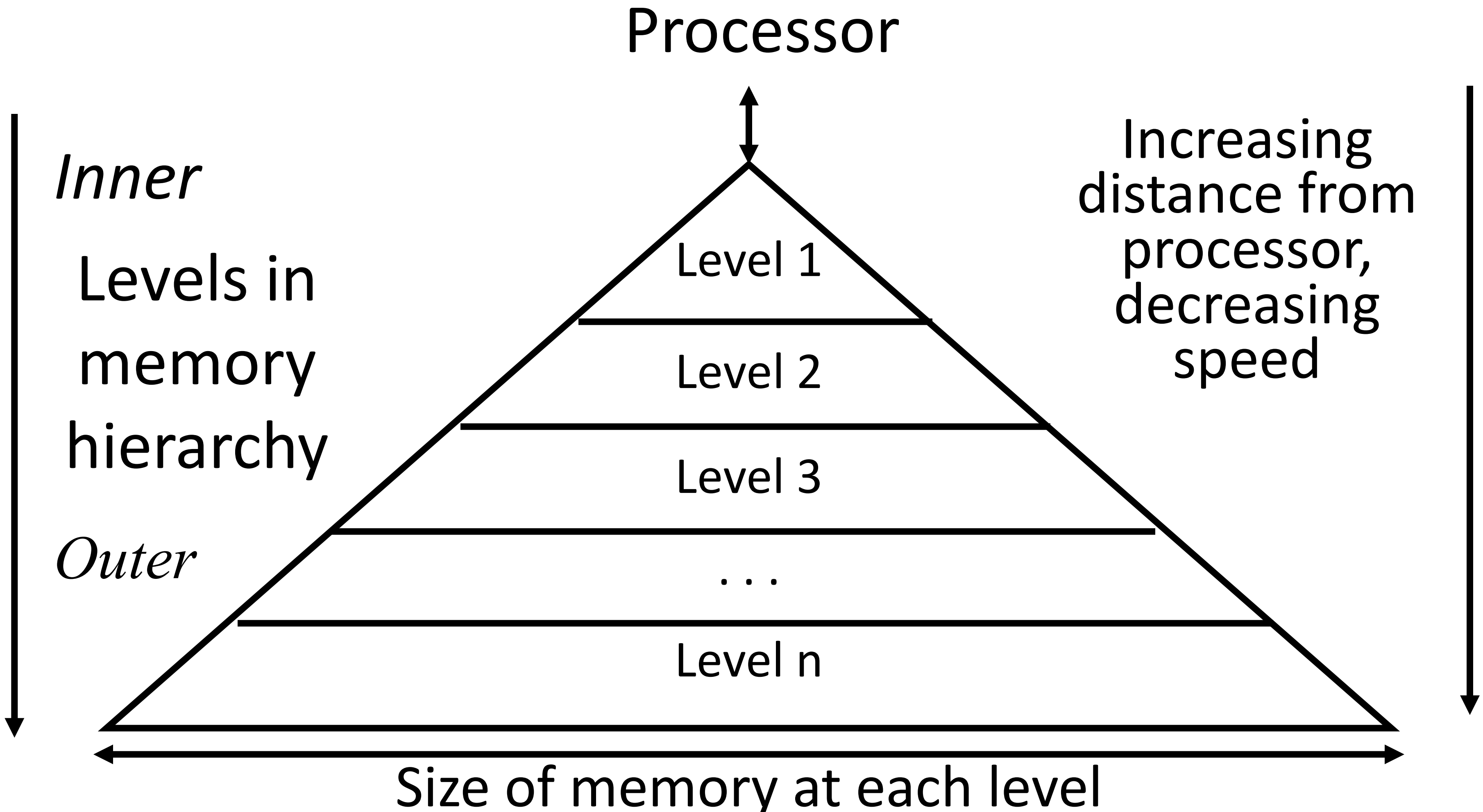




How to Reduce Miss Penalty?

- 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.

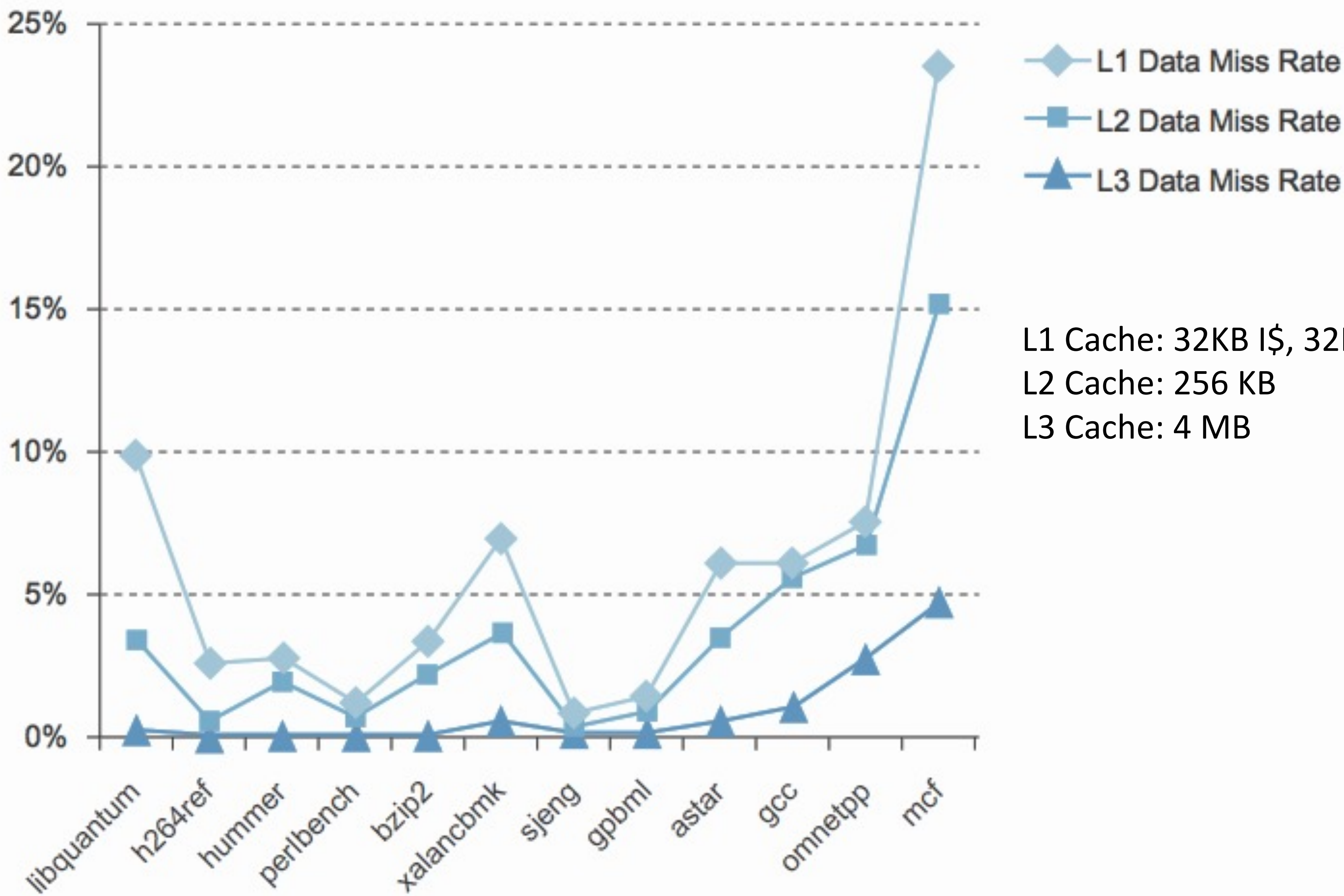
Review: Memory Hierarchy



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

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
  = L2$ Misses / total_L2_accesses
- *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



L1 Cache: 32KB I$, 32KB D$
L2 Cache: 256 KB
L3 Cache: 4 MB

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

- *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)

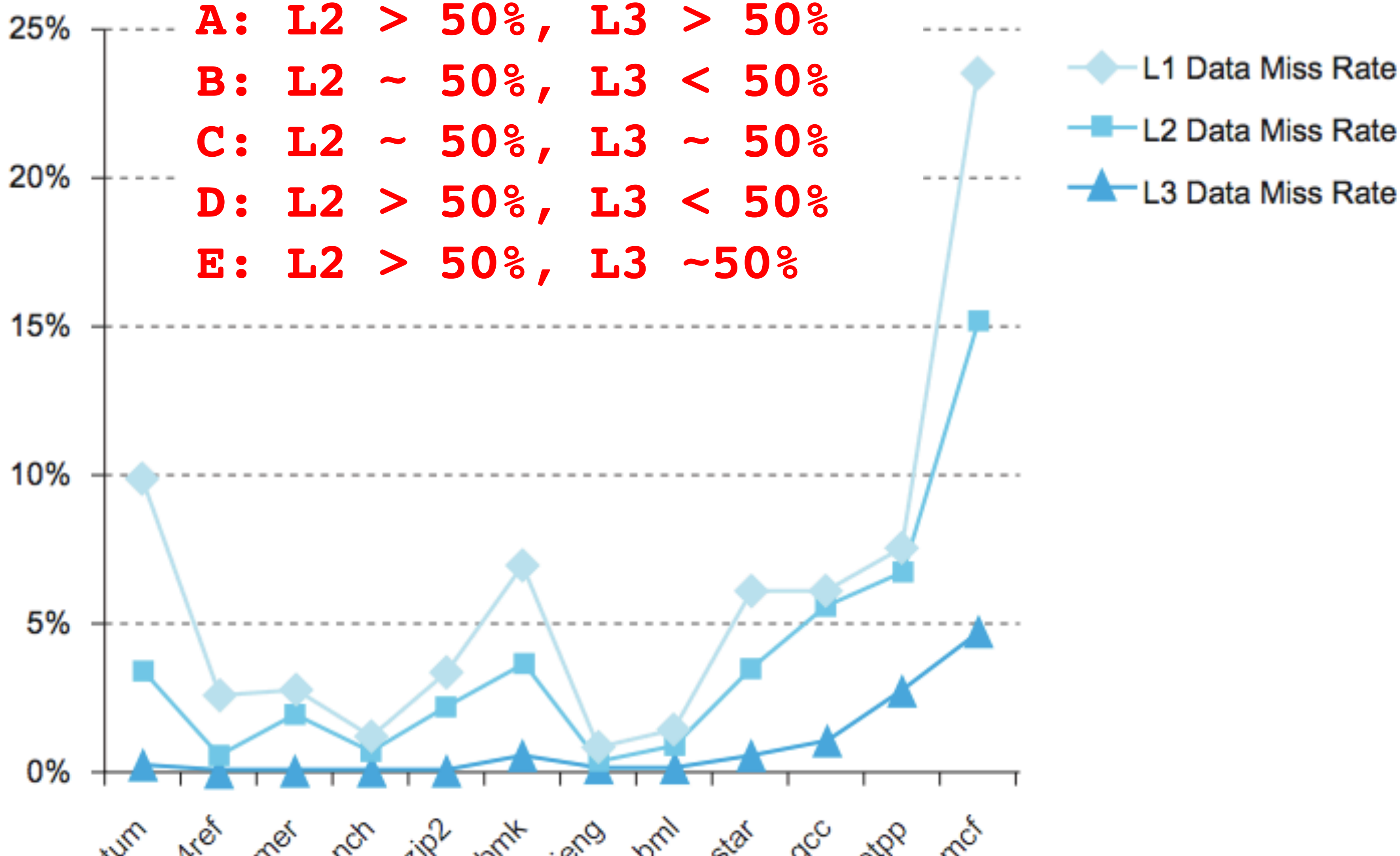
  = Local Miss rate L2$ × Local Miss rate L1$

- AMAT = Time for a hit + Miss rate × Miss penalty
- AMAT = Time for a L1$ hit + (local) Miss rate L1$ × (Time for a L2$ hit + (local) Miss rate L2$ × L2$ Miss penalty)

Question

- 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%



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

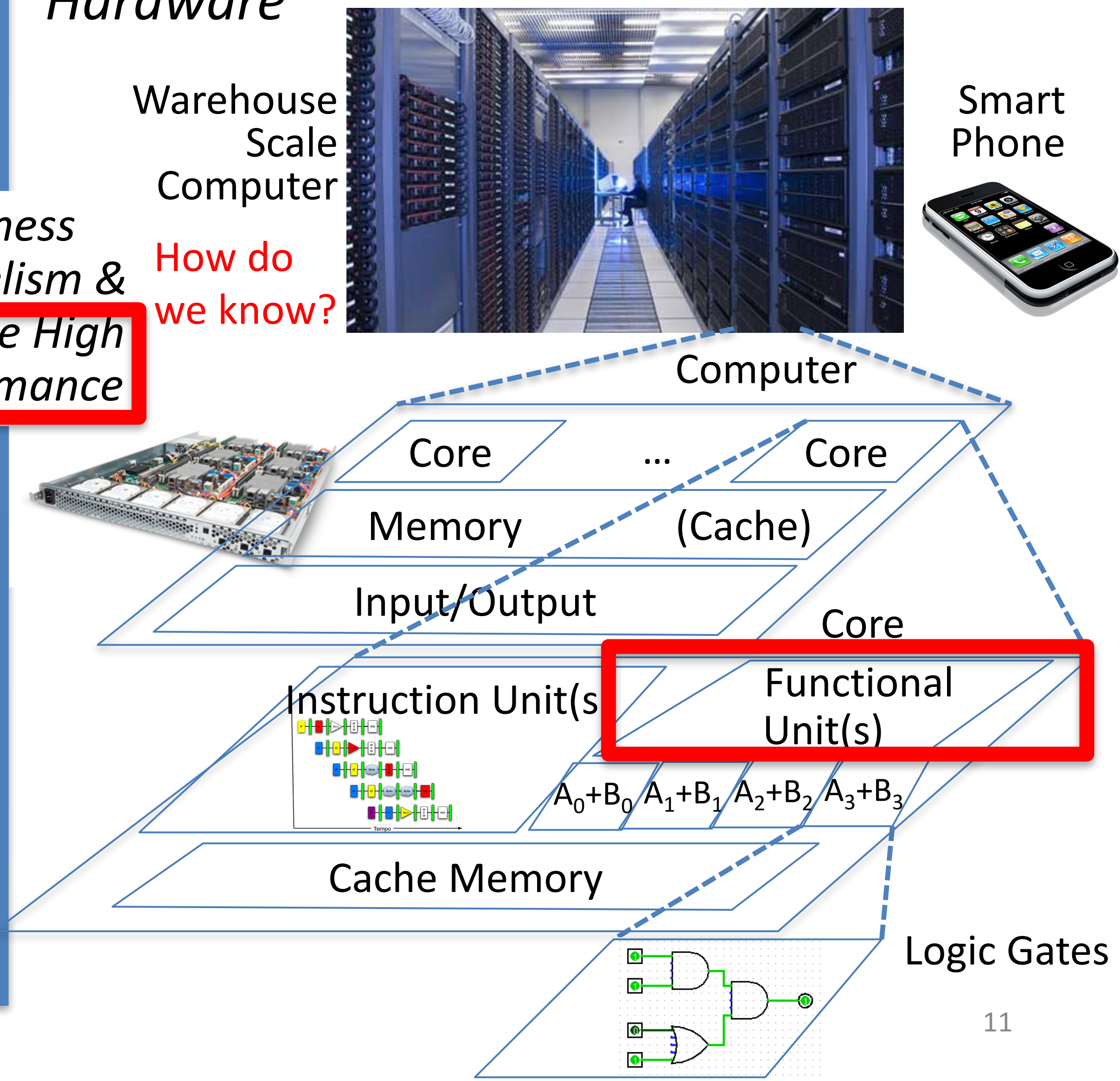
Name	CPI	L1 D cache misses/1000 instr (Data Only)	L2 D cache misses/1000 instr (Data Only)	DRAM accesses/1000 instr (Instructions and Data)
perl	0.75	3.5	1.1	1.3
bzip2	0.85	11.0	5.8	2.5
gcc	1.72	24.3	13.4	14.8
mcf	10.00	106.8	88.0	88.5
go	1.09	4.5	1.4	1.7
hmmer	0.80	4.4	2.5	0.6
sjeng	0.96	1.9	0.6	0.8
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

New-School Machine Structures (It's a bit more complicated!)

Software | *Hardware*

- 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

Harness Parallelism & Achieve High Performance



What is Performance?

- *Latency* (or *response time* or *execution time*)
  - Time to complete one task
- *Bandwidth* (or *throughput*)
  - Tasks completed per unit time

Cloud Performance: Why Application Latency Matters

Server Delay (ms)	Increased time to next click (ms)	Queries/ user	Any clicks/ user	User satisfac-tion	Revenue/ User
50	--	--	--	--	--
200	500	--	-0.3%	-0.4%	--
500	1200	--	-1.0%	-0.9%	-1.2%
1000	1900	-0.7%	-1.9%	-1.6%	-2.8%
2000	3100	-1.8%	-4.4%	-3.8%	-4.3%

Figure 6.10 Negative impact of delays at Bing search server on user behavior [Brutlag and Schurman 2009].

- Key figure of merit: application responsiveness
  - Longer the delay, the fewer the user clicks, the less the user happiness, and the lower the revenue per user

Defining CPU Performance

- What does it mean to say X is faster than Y?
- Ferrari vs. School Bus?

- 2013 Ferrari 599 GTB
  - 2 passengers, quarter mile in 10 secs
- 2013 Type D school bus
  - 50 passengers, quarter mile in 20 secs

- *Response Time* (*Latency)*: e.g., time to travel ¼ mile
- *Throughput* (*Bandwidth)*: e.g., passenger-mi in 1 hour

Defining Relative CPU Performance

- $Performance_X$ = 1/Program Execution $Time_X$
- $Performance_X > Performance_Y$ =>
  1/Execution $Time_X$ > 1/Execution $Time_y$ =>
  Execution $Time_Y$ > Execution $Time_X$
- Computer X is N times faster than Computer Y
  $Performance_X$ / $Performance_Y$ = N or
  Execution $Time_Y$ / Execution $Time_X$ = N

- Bus to Ferrari performance:
  - Program: Transfer 1000 passengers for 1 mile
  - Bus: 3,200 sec, Ferrari: 40,000 sec

Measuring CPU Performance

- Computers use a clock to determine when events takes place within hardware
- *Clock cycles:* discrete time intervals
  - aka clocks, cycles, clock periods, clock ticks
- *Clock rate* or *clock frequency:* clock cycles per second (inverse of clock cycle time)
- 3 GigaHertz clock rate
  => clock cycle time = $1/(3 \times 10^9)$ seconds
  clock cycle time = 333 picoseconds (ps)

CPU Performance Factors

- To distinguish between processor time and I/O, *CPU time* is time spent in processor
- `CPU Time/Program`

  `= Clock Cycles/Program`

  `x Clock Cycle Time`
- Or

  `CPU Time/Program`

  `= Clock Cycles/Program ÷ Clock Rate`

Iron Law of Performance

- A program executes instructions
- `CPU Time/Program`
```
  = Clock Cycles/Program x Clock Cycle Time
  = Instructions/Program
    x Average Clock Cycles/Instruction
    x Clock Cycle Time
```
- 1st term called *Instruction Count*
- 2nd term abbreviated *CPI* for average *__C__lock Cycles __P__er __I__nstruction*
- 3rd term is 1 / Clock rate

Restating Performance Equation

- Time = $\dfrac{\text{Seconds}}{\text{Program}}$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}$$

What Affects Each Component? A)Instruction Count, B)CPI, C)Clock Rate

	Affects What?
Algorithm	
Programming Language	
Compiler	
Instruction Set Architecture	

What Affects Each Component? Instruction Count, CPI, Clock Rate

	Affects What?
Algorithm	Instruction Count, CPI
Programming Language	Instruction Count, CPI
Compiler	Instruction Count, CPI
Instruction Set Architecture	Instruction Count, Clock Rate, CPI

Question

Computer	Clock frequency	Clock cycles per instruction	#instructions per program	
A	1GHz	2	1000	
B	2GHz	5	800	
C	500MHz	1.25	400	
D	5GHz	10	2000	

- Which computer has the highest performance for a given program?

Question

Computer	Clock frequency	Clock cycles per instruction	#instructions per program	Calculation
A	1GHz	2	1000	1ns * 2 * 1000 = 2μs
B	2GHz	5	800	0.5ns 5 *800 = 2μs
C	500MHz	1.25	400	2ns 1.25 * 400 = 1μs
D	5GHz	10	2000	0.2ns * 10 * 2000 = 4μs

- Which computer has the highest performance for a given program?

Workload and Benchmark

- *Workload:* Set of programs run on a computer
  - Actual collection of applications run or made from real programs to approximate such a mix
  - Specifies programs, inputs, and relative frequencies
- *Benchmark:* Program selected for use in comparing computer performance
  - Benchmarks form a workload
  - Usually standardized so that many use them

SPEC
(System Performance Evaluation Cooperative)

- Computer Vendor cooperative for benchmarks, started in 1989
- SPECCPU2006
  - 12 Integer Programs
  - 17 Floating-Point Programs
- Often turn into number where bigger is faster
- *SPECratio*: reference execution time on old reference computer divide by execution time on new computer to get an effective speed-up

SPEC CPU 2017

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

SPECINT2006 on AMD Barcelona

Description	Instruction Count (B)	CPI	Clock cycle time (ps)	Execution Time (s)	Reference Time (s)	SPEC-ratio
Interpreted string processing	2,118	0.75	400	637	9,770	15.3
Block-sorting compression	2,389	0.85	400	817	9,650	11.8
GNU C compiler	1,050	1.72	400	724	8,050	11.1
Combinatorial optimization	336	10.0	400	1,345	9,120	6.8
Go game	1,658	1.09	400	721	10,490	14.6
Search gene sequence	2,783	0.80	400	890	9,330	10.5
Chess game	2,176	0.96	400	837	12,100	14.5
Quantum computer simulation	1,623	1.61	400	1,047	20,720	19.8
Video compression	3,102	0.80	400	993	22,130	22.3
Discrete event simulation library	587	2.94	400	690	6,250	9.1
Games/path finding	1,082	1.79	400	773	7,020	9.1
XML parsing	1,058	2.70	400	1,143	6,900	6.0

Summarizing Performance …

System	Rate (Task 1)	Rate (Task 2)
A	10	20
B	20	10

Clickers: Which system is faster?

A: System A

B: System B

C: Same performance

D: Unanswerable question!

… Depends Who's Selling

System	Rate (Task 1)	Rate (Task 2)	Average
A	10	20	15
B	20	10	15

Average throughput

System	Rate (Task 1)	Rate (Task 2)	Average
A	0.50	2.00	1.25
B	1.00	1.00	1.00

Throughput relative to B

System	Rate (Task 1)	Rate (Task 2)	Average
A	1.00	1.00	1.00
B	2.00	0.50	1.25

Throughput relative to A

Summarizing SPEC Performance

- Varies from 6x to 22x faster than reference computer
- *Geometric mean* of ratios: N-th root of product of N ratios

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

  - Geometric Mean gives same relative answer no matter what computer is used as reference
- Geometric Mean for Barcelona is 11.7

Admin

- Regrade requests for Midterm-I will be closed on Friday!
- Project 2.2 will be published today!
- Next HW will be a little delayed.

Review of Numbers

- Computers are made to deal with numbers
- What can we represent in N bits?
  - 2^N things, and no more! They could be…
  - Unsigned integers:

    0 to $2^N - 1$

    (for N=32, $2^N-1 = 4{,}294{,}967{,}295$)
  - Signed Integers (Two's Complement)

    $-2^{(N-1)}$ to $2^{(N-1)} - 1$

    (for N=32, $2^{(N-1)} = 2{,}147{,}483{,}648$)

What about other numbers?

1. Very large numbers? (seconds/millennium)
   => $31{,}556{,}926{,}000_{10}$ ($3.1556926_{10} \times 10^{10}$)
2. Very small numbers? (Bohr radius)
   => 0.0000000000529177_{10}m ($5.29177_{10} \times 10^{-11}$)
3. Numbers with <u>both</u> integer & fractional parts?
   => 1.5

First consider #3.

...our solution will also help with #1 and #2.

Representation of Fractions

"Binary Point" like decimal point signifies boundary between integer and fractional parts:

Example 6-bit representation: xx.yyyy

2^{1} 2^{0} 2^{-1} 2^{-2} 2^{-3} 2^{-4}

$$\mathbf{10.1010_{two} = 1x2^{1} + 1x2^{-1} + 1x2^{-3} = 2.625_{ten}}$$

If we assume "fixed binary point", range of 6-bit representations with this format:

0 to 3.9375 (almost 4)

Fractional Powers of 2

i	2^{-i}	
0	1.0	1
1	0.5	1/2
2	0.25	1/4
3	0.125	1/8
4	0.0625	1/16
5	0.03125	1/32
6	0.015625	
7	0.0078125	
8	0.00390625	
9	0.001953125	
10	0.0009765625	
11	0.00048828125	
12	0.000244140625	
13	0.0001220703125	
14	0.00006103515625	
15	0.000030517578125	

Representation of Fractions with Fixed Pt.

What about addition and multiplication?

Addition is straightforward:

```
  01.100   1.5ten
+ 00.100   0.5ten
  10.000   2.0ten
```

Multiplication a bit more complex:

```
     01.100   1.5ten
     00.100   0.5ten
     00 000
    000 00
   0110 0
  00000
 00000
0000110000
```

Where's the answer, `0.11`? (need to remember where point is)

Representation of Fractions

So far, in our examples we used a "fixed" binary point. What we really want is to "float" the binary point. Why?

Floating binary point most effective use of our limited bits (and thus more accuracy in our number representation):

example: put 0.1640625_{ten} into binary. Represent with 5-bits choosing where to put the binary point.

… 000000.001010100000…

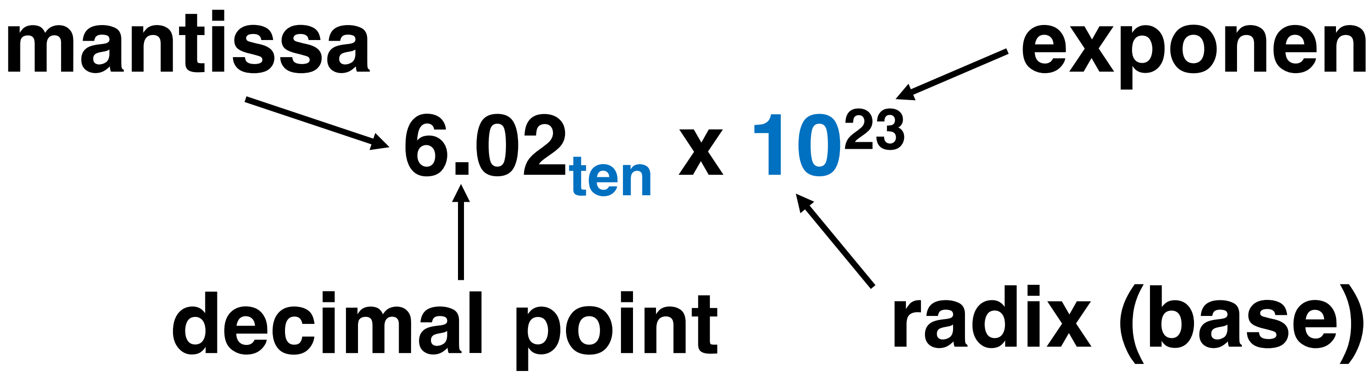
Store these bits and keep track of the binary point 2 places to the left of the MSB

Any other solution would lose accuracy!

With floating-point rep., each numeral carries an exponent field recording the whereabouts of its binary point.

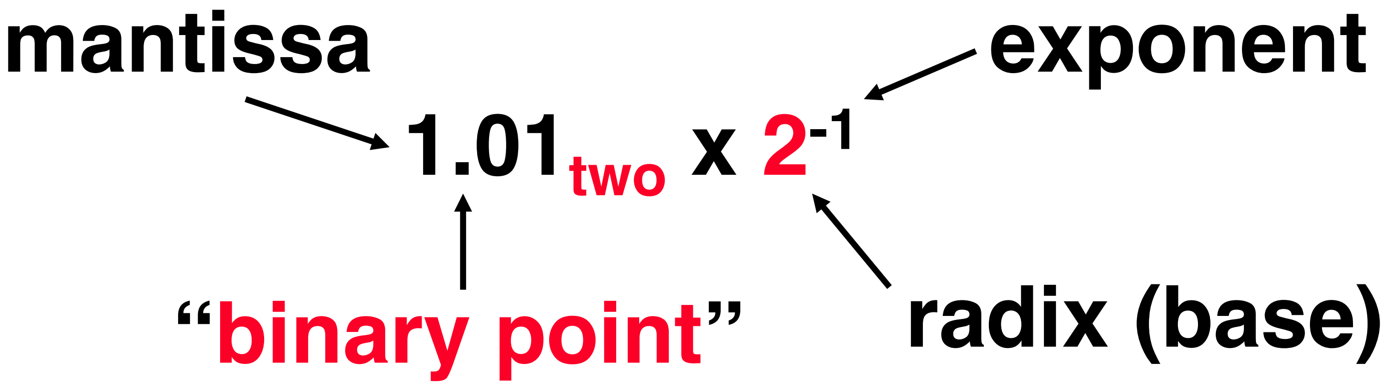
The binary point can be outside the stored bits, so very large and small numbers can be represented.

Scientific Notation (in Decimal)



- Normalized form: no leadings 0s
  (exactly one digit to left of decimal point)
- Alternatives to representing 1/1,000,000,000
  - Normalized: 1.0×10^{-9}
  - Not normalized: 0.1×10^{-8}, 10.0×10^{-10}

Scientific Notation (in Binary)



- Computer arithmetic that supports it called floating point, because it represents numbers where the binary point is not fixed, as it is for integers
  - Declare such variable in C as `float`
    - `double` for double precision.

Floating-Point Representation (1/2)

- **Normal format: $+1.xxx\ldots x_{two} * 2^{yyy\ldots y_{two}}$**
- **Multiple of Word Size (32 bits)**

31	30 … 23	22 … 0
S	Exponent	Significand
1 bit	8 bits	23 bits

- **S represents Sign**
  Exponent represents y's
  Significand represents x's

- **Represent numbers as small as $2.0_{ten} \times 10^{-38}$ to as large as $2.0_{ten} \times 10^{38}$**

Floating-Point Representation (2/2)

- What if result too large?

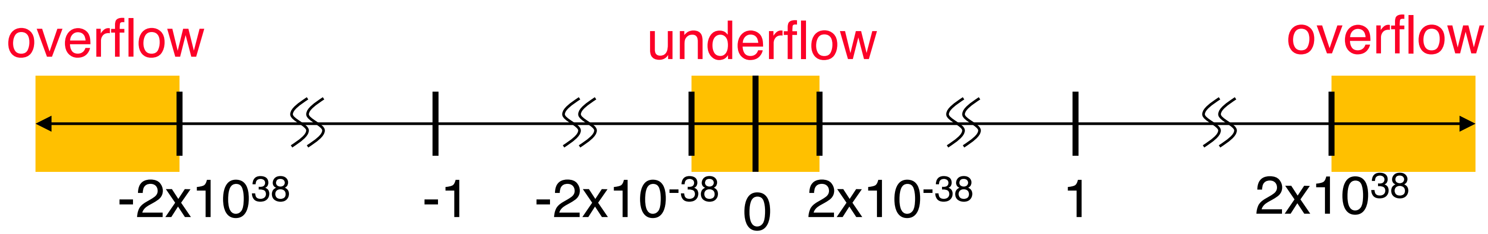
  (> 2.0×10^{38} , < -2.0×10^{38})

  - **Overflow!** => Exponent larger than represented in 8-bit Exponent field

- What if result too small?

  (>0 & < 2.0×10^{-38} , <0 & > -2.0×10^{-38})

  - **Underflow!** => Negative **exponent** larger than represented in 8-bit Exponent field



- What would help reduce chances of overflow and/or underflow?

IEEE 754 Floating-Point Standard (1/3)

Single Precision (Double Precision similar):

31	30 … 23	22 … 0
S	Exponent	Significand
1 bit	8 bits	23 bits

- Sign bit: 1 means negative 0 means positive
- Significand in *sign-magnitude* format (not 2's complement)
  - To pack more bits, leading 1 implicit for normalized numbers
  - 1 + 23 bits single, 1 + 52 bits double
  - always true: 0 < Significand < 1 (for normalized numbers)
- Note: 0 has no leading 1, so reserve exponent value 0 just for number 0