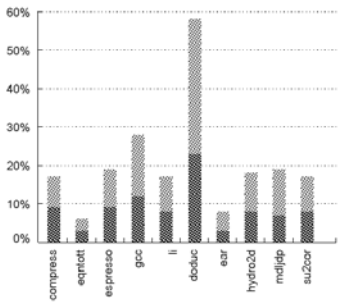
Memory and I/O Organization

Principle of Locality

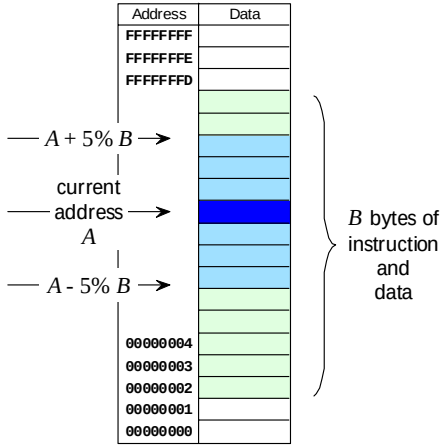
Locality — small proportion of memory accounts for most run time

Rule of thumb — For 90% of run time next instruction/data will come from 10% of program/data closest to current instruction

Amdahl's Law — make access to most local memory as fast as possible

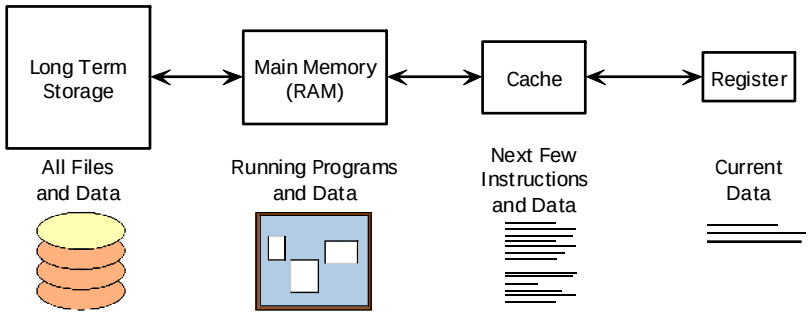


Percentage of memory accounting for 90% of run time for SPEC 92



Memory Hierarchy

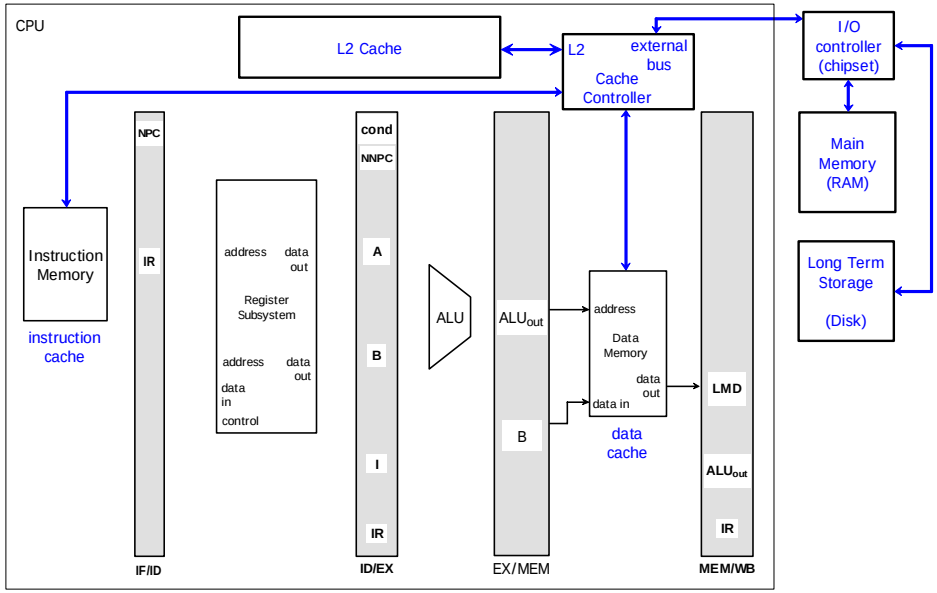
Memory locations outside CPU and RAM	Memory location outside CPU	Memory location in or near CPU	Memory location inside CPU
Stores data and instructions of "all" programs	Stores "all" data and instructions of running programs	Fast access to important data and instructions from RAM	Fast access to small amount of information
Organized by OS	Organized by addresses	Copy of RAM section	Organized by CPU



Memory Hierarchy in RISC Workstation/Server

Level	Location	Quantity	Access Time in System	Technology
General Registers	CPU Internal	Max < 2KB Typical 512 B	0.1 to 0.5 ns	CMOS latches
L1 cache (Level 1)	CPU Internal	Max < 64 KB Typical 32 KB	0.5 to 1.0 ns	CMOS SRAM
L2 cache (Level 2)	CPU Internal	Max < 8 MB Typical 2 MB	3 to 10 ns	CMOS SRAM
L3 cache (Level 3)	CPU Internal	Max < 8 MB Typical 0	10 to 20 ns	CMOS DRAM
Main Memory	External	~ 4 – 64 GB	10 to 50 ns	CMOS DRAM
Disk	External	Max ~ 8 TB Typical 1 TB	4 to 20 ms	Magnetic
Solid State Drive (SSD)	External	Max ~ 60 TB Typical 500 GB	0.035 to 0.1 ms	Flash

CPU and Memory Hierarchy



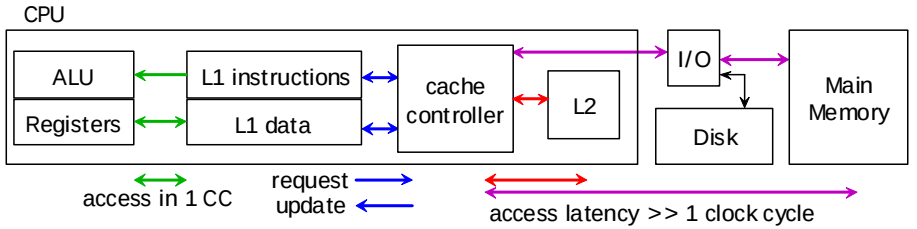
CPU and Memory Hierarchy — 1

L1 (level 1 cache) holds copy of small section of main memory

- Most recently accessed addresses (memory locations)
- L1 split into physically separate Instruction Cache and Data Cache

CPU accesses L1 cache directly

- If (address in L1 cache) {access performed in 1 clock cycle}
- Else {
  - L1 cache accesses cache controller
  - If (address in L2 cache) {controller copies contents to L1 from L2}
  - Else {controller copies location to L1 from main memory}
  - }



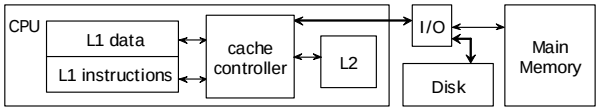
CPU and Memory Hierarchy — 2

CPU accesses disk and I/O devices by memory addressing

- Part of total address space reserved for I/O and storage devices
- Disk write of k bytes — CPU performs k stores to same I/O address
- Disk read of k bytes — CPU performs k loads from same I/O address
- I/O addresses are not copied to cache (marked **NON-CACHEABLE**)

Virtual Memory

- Total memory space larger than physical memory
- Divided into pages of specific size
- Infrequently used pages moved to "page file" on disk
- Page properties and location specified in page tables
- Virtual address divided into
  - Page address — points to page table entry
  - Offset — points to byte address in page



Memory Organization — 1

n-bit address space

Physical Address = $A_{n-1}\, A_{n-2} \ldots A_1\, A_0$

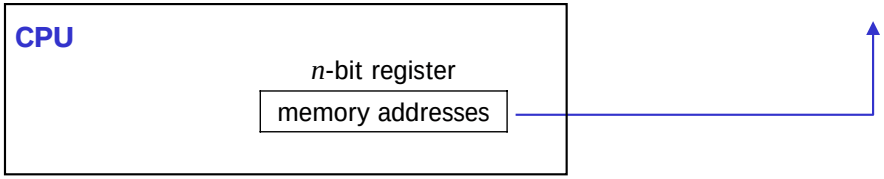
Can form 2^n addresses, from 0 to $(2^n - 1)$

Every byte in RAM has an n-bit address

Processor refers to memory locations by physical RAM addresses

Processor stores memory addresses in n-bit address registers

Memory Location	Address
Data Byte	11111…111
Data Byte	11111…110
Data Byte	11111…101
Data Byte	11111…100
…	…
Data Byte	00000…111
Data Byte	00000…110
Data Byte	00000…101
Data Byte	00000…100
Data Byte	00000…011
Data Byte	00000…010
Data Byte	00000…001
Data Byte	00000…000



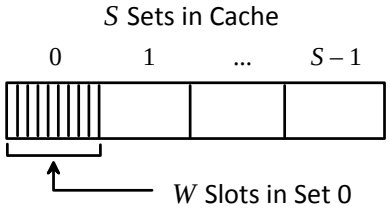
Memory Organization — 2

Copy memory data to cache

How much data to copy?

Which data is in cache?

How to find addresses in cache?



Copy DATA BLOCK to cache

Block = B bytes

Page size (virtual memory) = integer × block size

Sets and Slots

Cache = S sets

Set = W slots

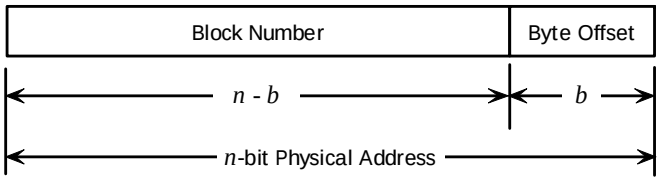
Slot = 1 data block

Copy memory block to:
- Deterministic set
- Any slot in set

Memory Organization — 3

Address space has n-bit physical address

$N = 2^n$ byte address space



Address space divided (logically) into address BLOCKS (lines)

Block size $B = 2^b$ bytes / block

Blocks in address space $= N / B = 2^{n-b}$

$(n\text{-}b)$-bit Block Number = Int (Address / B)

b-bit byte offset = Address % B = 0, 1, … , $B - 1 = 2^b - 1$

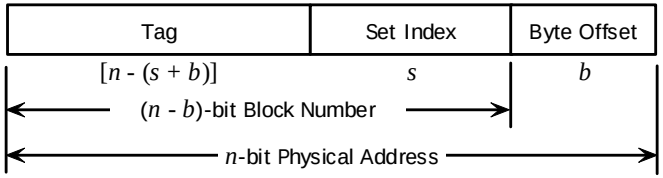
Memory Organization — 4

Cache organized into $S = 2^s$ Sets

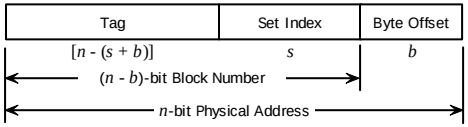
Set Index = 0, 1, … , $S - 1 = 2^s - 1$

Address Block in cache must be copied into specific Set

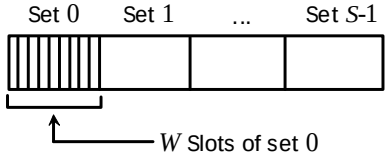
Set Index (location of block in cache) = Block Number % S



Memory Organization — 5



W-way associative organization



Each set contains $W = 2^w$ slots

One block copied to one slot

Blocks copied to slots in any convenient order

TAG written near block content to identify which block is in cache

$[n - (s + b)]$-bit tag = Int (Block Number / S)

Total Cache Size

$$\text{total cache size} = S \times W \times B = \frac{\text{sets}}{\text{cache}} \times \frac{\text{blocks}}{\text{set}} \times \frac{\text{bytes}}{\text{block}} = \frac{\text{bytes}}{\text{cache}}$$

Example of Memory Organization — 1

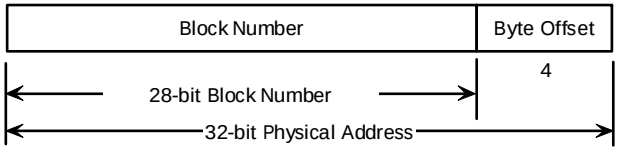
n = 32-bit address

N = 2^{32} bytes = 4,294,967,296 bytes = 4 GB

B = 16 = 24 bytes per block

b = 4-bit block offset N / B = 2^{32-4} blocks = 268,435,456 blocks

= 256 Mblocks



Block	Byte Address															
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	...											
...	...															
268,435,455	4,294,967,280				...								4,294,967,295			

Example of Memory Organization — 2

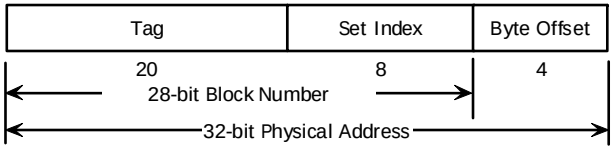
n = 32-bit address N = 2^{32} bytes = 4,294,967,296 bytes = 4 GB

B = 24 bytes per block

N / B = 268,435,456 blocks = 256 Mblocks

S = 256 = 2^8 sets in cache

s = 8-bit set index = Block Number % 256

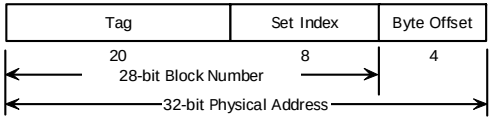


Set	Blocks that can be Assigned to Set							
0	0	256	512	768	1024	1280	...	268,435,200
1	1	257	513	769	1025	1281	...	
2	2	258	514	770	1026	1282	...	
	...							
255	255	511	767	1023	1279	1535	...	268,435,455

Example of Memory Organization — 3

n = 32-bit address N = 2^{32} bytes = 4,294,967,296 bytes = 4 GB

B = 24 bytes per block



S = 256 = 2^8 sets in cache

s = 8-bit **set index** = **Block Number** % 256

W = 4 = 2^2 way associative

Tag = 32 – (4 + 8) = 20 bits

Set	Possible Cache Content (Block Numbers)				Tags			
0	256	0	8192	1024	1	0	32	4
1	1025	513	1281	257	4	2	5	1
2	514	258	...		2	1	...	

Address 20509_{10} = 0x0000501D		
00000000000000000101	00000001	1101
00000000000000000101000000001 = 1281_{10}		13_{10}

Total Cache Size = 256 sets/cache × 4 blocks/set × 16 bytes/block = 16 KB

Cache Definitions and Policies

Cache hit

CPU finds block in cache

Cache miss

CPU needs block not in cache

Cache loads block on cache read miss

Write allocate

Cache loads block on cache write miss

No write allocate

Write to RAM without loading block on cache write miss

Swapping out cache block

Need a new block in a full set

Remove block that is **LEAST RECENTLY USED** (LRU)

WRITE BACK — update RAM when block is swapped out

WRITE THROUGH — update RAM on every write to cache

Cache Performance Issues

L1 Cache hit

CPU reads/writes memory in 1 clock cycle

L1 Cache Miss

CPU stalls while L1 loads missing block

Miss Rate

Cache misses per cache accesses (instruction fetch, load, store)

Depends on cache size and organization

Miss Penalty

Number of stall cycles while cache loads missing block

Depends on cache hardware technology and organization

For 1 level unified (not split) cache

$$CPI^{stall} = \frac{\text{stall cycles}}{\text{stalls}} \times \frac{\text{stalls}}{\text{instruction type}} \times \frac{\text{instruction types}}{\text{instruction}}$$

$$= \text{miss penalty} \times \text{miss rate} \times \frac{IC + IC^{load} + IC^{store}}{IC}$$

Performance of 1-Level Split Cache

$$CPI^{stall} = \frac{\text{instruction stall cycles}}{\text{stalls}} \times \frac{\text{instruction stalls}}{\text{instruction access}} \times \frac{\text{instruction accesses}}{\text{instruction}}$$

$$+ \frac{\text{data stall cycles}}{\text{stalls}} \times \frac{\text{data stalls}}{\text{data access}} \times \frac{\text{data accesses}}{\text{instruction}}$$

$$= \text{instruction miss penalty} \times \text{instruction miss rate} \times \frac{1 \text{ instruction access}}{\text{instruction}}$$

$$+ \text{data miss penalty} \times \text{data miss rate} \times \frac{IC^{load} + IC^{store}}{IC}$$

Typical values

Instruction Miss Penalty = Data Miss Penalty = 50 cycles per stall

Instruction Miss Rate = 0.5%

Data Miss Rate = 5.0%

IC^{load} = 25%

IC^{store} = 15%

$$CPI^{stall} = 50 \times 0.005 \times 1 + 50 \times 0.05 \times 0.40 = 0.25$$

$CPI = 1.25 \Rightarrow$ 20% degradation

Two-Level Unified Cache — Definitions

Miss Rates

M_{L1} = Miss Rate at L1

$= \frac{\text{L1 miss}}{\text{L1 access}}$

$1 - M_{L1}$ = Hit Rate at L1

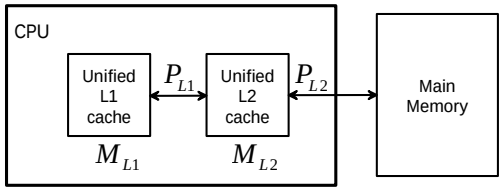
$= \frac{\text{L1 hit}}{\text{L1 access}}$

M_{L2} = Miss Rate at L2

$= \frac{\text{L2 miss}}{\text{L2 access}}$

$1 - M_{L2}$ = Hit Rate at L2

$= \frac{\text{L2 hit}}{\text{L2 access}}$



Miss Penalties

P_{L1} = Miss Penalty (cycles) from L1 to L2

$= \frac{\text{stall cycles}}{\text{(L1 miss, L2 hit)}}$

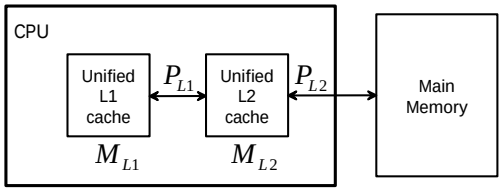
P_{L2} = Miss Penalty (cycles) from L2 to Main Memory

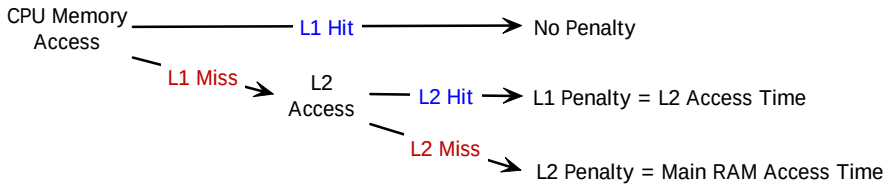
$P_{L1} + P_{L2} = \frac{\text{stall cycles}}{\text{(L1 miss, L2 miss)}}$

IC^A = Data Access Instructions (Load/Store)

IC = Total Instructions

Performance of Two-Level Unified Cache

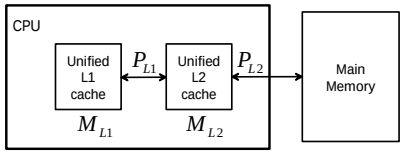




$$\text{CPI}^{\text{stall}} = \frac{\text{stall cycles}}{\text{IC}} = \sum_{n\ =\ \text{stall types}} \frac{\text{stall cycles}}{\text{stall type } n} \times \frac{\text{stalls of type } n}{\text{IC}}$$

$$\text{stall types} = \sum_{i\ =\ (\text{instruction, data})} \left\{(\text{L1 miss, L2 hit})_i, (\text{L1 miss, L2 miss})_i\right\}$$

Stalls in Two-Level Unified Cache



$$\text{CPI}^{\text{stall}} = \sum_{i\ =\ \{\text{data, instr}\}} \left[\frac{\text{stall cycles}}{(\text{L1 miss, L2 hit})_i} \times \frac{(\text{L1 miss, L2 hit})_i}{\text{IC}} + \frac{\text{stall cycles}}{(\text{L1 miss, L2 miss})_i} \times \frac{(\text{L1 miss, L2 miss})_i}{\text{IC}} \right]$$

$$= \sum_{i\ =\ \{\text{data, instr}\}} \left[\frac{\text{stall cycles}}{(\text{L1 miss, L2 hit})_i} \times \frac{(\text{L1 miss, L2 hit})_i}{\text{L1 access}_i} \times \frac{\text{L1 access}_i}{\text{IC}} + \frac{\text{stall cycles}}{(\text{L1 miss, L2 miss})_i} \times \frac{(\text{L1 miss, L2 miss})_i}{\text{L1 access}_i} \times \frac{\text{L1 access}_i}{\text{IC}} \right]$$

Assume L1 and L2 Are Statistically Independent

$$\frac{(\text{L1 miss, L2 hit})_i}{\text{L1 access}_i} = \frac{\text{L1 miss}_i \text{ followed by L2 hit}}{\text{L1 access}_i} = \frac{\text{L1 miss}_i}{\text{L1 access}_i} \times \underbrace{\frac{\text{L2 access}}{\text{L1 miss}_i}}_{1} \times \frac{\text{L2 hit}}{\text{L2 access}}$$

$$= \frac{\text{L1 miss}_i}{\text{L1 access}_i} \times \frac{\text{L2 hit}}{\text{L2 access}}$$

$$\frac{(\text{L1 miss, L2 miss})_i}{\text{L1 access}_i} = \frac{\text{L1 miss}_i \text{ followed by L2 miss}}{\text{L1 access}_i} = \frac{\text{L1 miss}_i}{\text{L1 access}_i} \times \underbrace{\frac{\text{L2 access}}{\text{L1 miss}_i}}_{1} \times \frac{\text{L2 miss}}{\text{L2 access}}$$

$$= \frac{\text{L1 miss}_i}{\text{L1 access}_i} \times \frac{\text{L2 miss}}{\text{L2 access}}$$

$$\text{CPI}^{\text{stall}} = \sum_{i\ =\ \{\text{data, instr}\}} \left[\frac{\text{stall cycles}}{(\text{L1 miss, L2 hit})_i} \times \frac{\text{L1 miss}_i}{\text{L1 access}_i} \times \frac{\text{L2 hit}}{\text{L2 access}} \times \frac{\text{L1 access}_i}{\text{IC}} + \frac{\text{stall cycles}}{(\text{L1 miss, L2 miss})_i} \times \frac{\text{L1 miss}_i}{\text{L1 access}_i} \times \frac{\text{L2 miss}}{\text{L2 access}} \times \frac{\text{L1 access}_i}{\text{IC}} \right]$$

Simplifying

$$\text{CPI}^{\text{stall}} = \sum_{i\ =\ \{\text{data, instr}\}} \left[\frac{\text{stall cycles}}{(\text{L1 miss, L2 hit})_i} \times \frac{\text{L1 miss}_i}{\text{L1 access}_i} \times \frac{\text{L2 hit}}{\text{L2 access}} \times \frac{\text{L1 access}_i}{\text{IC}} + \frac{\text{stall cycles}}{(\text{L1 miss, L2 miss})_i} \times \frac{\text{L1 miss}_i}{\text{L1 access}_i} \times \frac{\text{L2 miss}}{\text{L2 access}} \times \frac{\text{L1 access}_i}{\text{IC}} \right]$$

$$= P_{L1} \times M_{L1} \times (1 - M_{L2}) \times \frac{IC + IC^A}{IC} + (P_{L1} + P_{L2}) \times M_{L1} \times M_{L2} \times \frac{IC + IC^A}{IC}$$

$$= M_{L1} \times \left[1 + \frac{IC^A}{IC} \right] \times \left[P_{L1} \times (1 - M_{L2}) + (P_{L1} + P_{L2}) \times M_{L2} \right]$$

$$= M_{L1} \times \left[1 + \frac{IC^A}{IC} \right] \times \left[P_{L1} - P_{L1} \times M_{L2} + P_{L1} \times M_{L2} + P_{L2} \times M_{L2} \right]$$

$$\text{CPI}^{\text{stall}} = M_{L1} \times \left[1 + \frac{IC^A}{IC} \right] \times \left[P_{L1} + P_{L2} \times M_{L2} \right]$$

Split L1 Cache with Unified L2 Cache

$$\frac{\text{misses at L1}}{\text{IC}} = \frac{\text{misses at L1}}{\text{accesses at L1}} \times \frac{\text{accesses at L1}}{\text{IC}}$$

$$= M_{L1} \times \left[1 + \frac{IC^A}{IC} \right] = M_{L1} + M_{L1} \times \frac{IC^A}{IC}$$

$$\downarrow$$

$$\frac{\text{data misses at L1}}{\text{data accesses at L1}} + \frac{\text{instruction misses at L1}}{\text{instruction accesses at L1}} = \sum_{i=(\text{data,instructions})} \frac{\text{misses at L1}_i}{\text{accesses at L1}_i} \times \frac{\text{accesses at L1}_i}{\text{IC}}$$

$$= M_{L1}^{\text{instructions}} \times 1 + M_{L1}^{\text{data}} \times \frac{IC^A}{IC}$$

$$= M_{L1}^{\text{I}} + M_{L1}^{\text{D}} \times \frac{IC^A}{IC}$$

$$\text{CPI}^{\text{stall}}_{\text{unified}} = M_{L1} \times \left[1 + \frac{IC^A}{IC} \right] \times \left[P_{L1} + P_{L2} \times M_{L2} \right]$$

$$\downarrow$$

$$\text{CPI}^{\text{stall}}_{\text{split}} = \left[M_{L1}^{\text{I}} + M_{L1}^{\text{D}} \times \frac{IC^A}{IC} \right] \times \left[P_{L1} + P_{L2} \times M_{L2} \right]$$

Second Layer Cache (L2) with Split L1

Definitions

P_{L1} = Miss Penalty (cycles) at L1

P_{L2} = Miss Penalty (cycles) at L2

IC^A = Data Access Instructions (Load/Store)

IC = Total Instructions

M_{L1}^I = Instruction Miss Rate at L1

M_{L1}^D = Data Miss Rate at L1

M_{L2} = Miss Rate at L2

One layer (L1) of split cache

$$CPI_{1-level}^{stall} = \left[M_{L1}^I + M_{L1}^D \times \frac{IC^A}{IC} \right] \times P_{L1}$$

Miss Penalty at L1 (to main memory) P_{L1} ~ 50 cycles

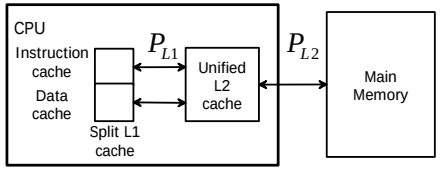
Split L1 cache and unified L2 cache

Miss Penalty at L1 (to L2) P_{L1} ~ 5 cycles

Miss Penalty at L2 (to main memory) P_{L2} ~ 45 cycles

Miss Rate at L2 ~ 1%

$$CPI_{2-level}^{stall} = \left[M_{L1}^I + M_{L1}^D \times \frac{IC^A}{IC} \right] \times \left(P_{L1} + P_{L2} \times M_{L2} \right)$$



$$P_{L1} + P_{L2} \times M_{L2} = 5 + 45 \times 0.01 = 5.45$$

Issues Affecting Miss Rate

Compulsory miss

Block not copied to cache until first access to byte in block

First access to block always misses in cache

Compulsory miss not affected by cache properties

Capacity miss

Cache is smaller than main memory

Some blocks removed from cache to make room for required blocks

Capacity miss rate lower for larger cache size

Conflict miss

Block must be copied to specific set

Some blocks removed from set to make room for required blocks

- Not caused by overall capacity miss
- For example, misses caused by address aliasing

Conflict miss rate lower when block placement more flexible

Cache Miss Statistics — 1

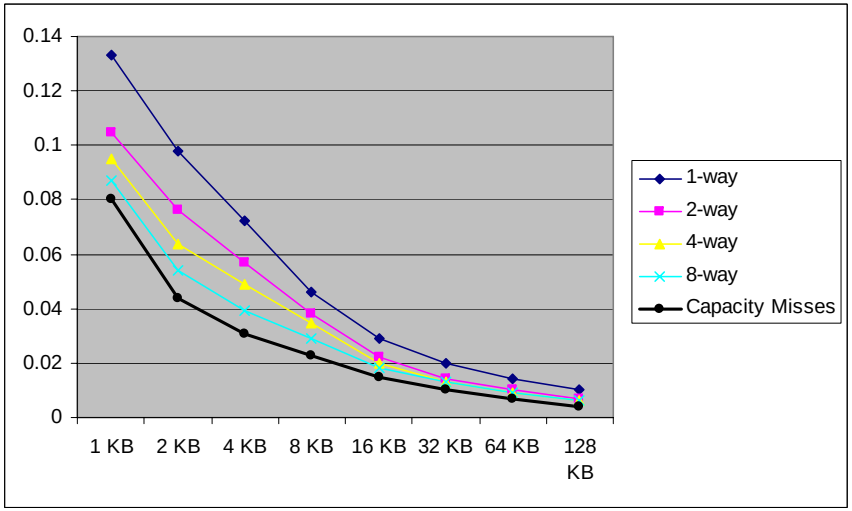
Cache size	Associativity	Total miss rate	Miss rate components (relative percent) (Sum = 100% of total miss rate)					
			Compulsory		Capacity		Conflict	
1 KB	1-way	0.133	0.002	1%	0.080	60%	0.052	39%
1 KB	2-way	0.105	0.002	2%	0.080	76%	0.023	22%
1 KB	4-way	0.095	0.002	2%	0.080	84%	0.013	14%
1 KB	8-way	0.087	0.002	2%	0.080	92%	0.005	6%
2 KB	1-way	0.098	0.002	2%	0.044	45%	0.052	53%
2 KB	2-way	0.076	0.002	2%	0.044	58%	0.030	39%
2 KB	4-way	0.064	0.002	3%	0.044	69%	0.018	28%
2 KB	8-way	0.054	0.002	4%	0.044	82%	0.008	14%
4 KB	1-way	0.072	0.002	3%	0.031	43%	0.039	54%
4 KB	2-way	0.057	0.002	3%	0.031	55%	0.024	42%
4 KB	4-way	0.049	0.002	4%	0.031	64%	0.016	32%
4 KB	8-way	0.039	0.002	5%	0.031	80%	0.006	15%
8 KB	1-way	0.046	0.002	4%	0.023	51%	0.021	45%
8 KB	2-way	0.038	0.002	5%	0.023	61%	0.013	34%
8 KB	4-way	0.035	0.002	5%	0.023	66%	0.010	28%
8 KB	8-way	0.029	0.002	6%	0.023	79%	0.004	15%

Hennessy and Patterson, figure 5.9

Cache Miss Statistics — 2

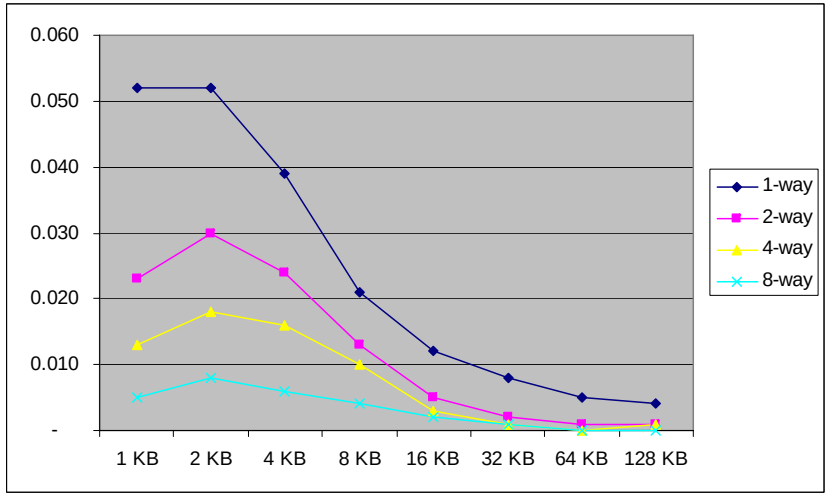
Cache size	Associativity	Total miss rate	Miss rate components (relative percent) (Sum = 100% of total miss rate)					
			Compulsory		Capacity		Conflict	
16 KB	1-way	0.029	0.002	7%	0.015	52%	0.012	42%
16 KB	2-way	0.022	0.002	9%	0.015	68%	0.005	23%
16 KB	4-way	0.020	0.002	10%	0.015	74%	0.003	17%
16 KB	8-way	0.018	0.002	10%	0.015	80%	0.002	9%
32 KB	1-way	0.020	0.002	10%	0.010	52%	0.008	38%
32 KB	2-way	0.014	0.002	14%	0.010	74%	0.002	12%
32 KB	4-way	0.013	0.002	15%	0.010	79%	0.001	6%
32 KB	8-way	0.013	0.002	15%	0.010	81%	0.001	4%
64 KB	1-way	0.014	0.002	14%	0.007	50%	0.005	36%
64 KB	2-way	0.010	0.002	20%	0.007	70%	0.001	10%
64 KB	4-way	0.009	0.002	21%	0.007	75%	0.000	3%
64 KB	8-way	0.009	0.002	22%	0.007	78%	0.000	0%
128 KB	1-way	0.010	0.002	20%	0.004	40%	0.004	40%
128 KB	2-way	0.007	0.002	29%	0.004	58%	0.001	14%
128 KB	4-way	0.006	0.002	31%	0.004	61%	0.001	8%
128 KB	8-way	0.006	0.002	31%	0.004	62%	0.000	7%

Total Miss Rate



Total miss rate drops as capacity or associativity increases

Conflict Miss Rate



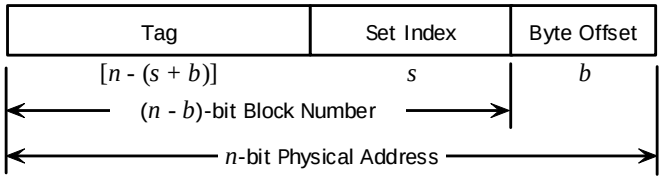
Conflict miss rate drops as associativity increases

Associativity Trade-Off

Block can be anywhere in set

Finding (or missing) block in set requires searching every tag in set

Larger associativity ⇒ more blocks per set ⇒ longer search time



For fixed cache capacity = $S \times W \times B$

Larger associativity W

⇒ fewer sets S

⇒ smaller set index s

⇒ larger tag size $n-(s+b)$

⇒ longer tag search

Small advantage beyond 4-way associativity

Miss Example — Extreme Locality

Program

```
int i,a;
for (i = 0 ; i < 4096 ; i++){
    a = i;
}
```

Compiler assignments

Register ← `i`

Memory ← `a`

Memory accesses

4096 write accesses to integer **a**

1 compulsory cache miss (write allocate) on `i = 0`

4095 cache hits on `i > 0`

0 read accesses to **a**

Miss rate

$$\text{miss rate} = \frac{1 \text{ miss}}{4096 \text{ accesses}} = 2.44 \times 10^{-4}$$

Miss Example — Extreme Non-Locality

Block size

16 bytes/block = 4 words/block = 4 array elements

Program

```
int i,a[4096];
for (i = 0 ; i < 4096 ; i++){
    a[i] = i;
}
```

Compiler assignments
Register ← `i`
Memory ← `a[]`

Memory accesses

4096 write accesses to integer array **`a[]`**

Compulsory cache miss every 4 array elements

4096/4 = 1024 cache misses (write allocate)

0 read accesses to **`a[]`**

Miss rate

$$\text{miss rate} = \frac{1024 \text{ misses}}{4096 \text{ accesses}} = 0.25$$

Miss Example — Good Locality

Cache parameters	B	S	W	Capacity = B × S × W
	16	256	4	16 KB = 4 Kwords

Program

```
int a[4096],i,j;
for (i = 0 ; i < 10 ; i++){
    for (j = 0 ; j < 4096 ; j++){
    a[j] = i + j;
    }
}
```

Compiler assignments
Register ← `i,j`
Memory ← `a[]`

Memory accesses

40960 write accesses to integer array **`a[]`**

`i = 0:` Compulsory cache miss every 4 elements

`i > 0:` Entire array in cache ⇒ cache hits

sets				
	255	511	767	1023
	...	...	...	...
	2	259	514	770
	1	257	513	769
	0	256	512	768
	0	1	2	3

4 slots

4K integers = 16 Kbytes
= 1024 blocks

Miss rate

$$\text{miss rate} = \frac{1024 \text{ misses}}{40960 \text{ accesses}} = 0.025$$

Miss Example — Extreme Non-Locality

Cache parameters	B	S	W	Capacity = B × S × W
	16	256	4	16 KB = 4 Kwords

Program

```
int a[5120],i,j;
for (i = 0 ; i < 10 ; i++){
    for (j = 0 ; j < 5120 ; j++){
    a[j] = i + j;
    }
}
```

Compiler assignments
Register ← `i,j`
Memory ← `a[]`

Memory accesses

51200 write accesses to integer array **`a[]`**

Compulsory cache miss every 4 elements

LRU: Next element in never cache

Miss rate

$$\text{miss rate} = \frac{12800 \text{ misses}}{51200 \text{ accesses}} = 0.25$$

	...			
	768 1024	0	256	512
	1024	0	256	512 768
	1024	0	256 512	768
i = 1	1024	0 256	512	768
i = 0	1024 0	256	512	768

Set 0 (Bold = Miss)

Miss Example — Better Locality Using MRU

Cache parameters	B	S	W	Capacity = B × S × W
	16	256	4	16 KB = 4 Kwords

Program

```
int a[5120],i,j;
for (i = 0 ; i < 10 ; i++){
    for (j = 0 ; j < 5120 ; j++){
    a[j] = i + j;
    }
}
```

Compiler assignments
Register ← `i,j`
Memory ← `a[]`

Memory accesses with MRU

51200 write accesses to integer array **`a[]`**

`i = 0:` Compulsory cache miss every 4 elements

`i > 0:` Conflict misses to slot 3 (2 out of 5 accesses)

j=0...256, 1024...1279

255	511	767	1023	1279
...	...	...	...	...
2	259	514	770	1026
1	257	513	769	1025
0	256	512	768	1024
0	1	2	3	

4 slots

5K integers = 1280 blocks

Miss rate

$$\text{miss rate} = \frac{1280 + 512 \times 9 \text{ misses}}{51200 \text{ accesses}} = 0.025 \times \left(1 + \frac{2}{5} \times 9\right) = 0.115$$

Miss Example — Improved Locality

Cache parameters	B	S	W	Capacity = B × S × W
	16	256	4	16 KB = 4 Kwords

Program

```
int a[5120],i,j;
for (i = 0 ; i < 10 ; i++){
     for (j = 0 ; j < 4096 ; j++){
     a[j] = i + j;
     }
for (i = 0 ; i < 10 ; i++){
     for (j = 4096 ; j < 5120 ; j++){
     a[j] = i + j;
     }
}
```

Compiler assignments

Register ← `i,j`

Memory ← `a[]`

255	511	767	1023
...	...	...	...
2	259	514	770
1	257	513	769
0	256	512	768

1279	511	767	1023
...	...	...	...
1026	259	514	770
1025	257	513	769
1024	256	512	768

5K integers = 1280 blocks

Memory accesses

51200 write accesses to integer array `a[]`

`i = 0:` Compulsory cache miss every 4 elements

`i > 0:` Entire array in cache ⇒ cache hits

Miss rate

$$\text{miss rate} = \frac{1024 + 256 \text{ misses}}{40960 + 10240 \text{ accesses}} = 0.025$$

Miss Example — Address Aliasing

Cache parameters	B	S	W	Capacity = B × S × W
	16	256	2	8 KB = 2 Kwords

Program

```
int a[512],b[512],c[512],i,j;
 for (i = 0 ; i < 20; i++){
   for (j = 0 ; j < 512 ; j++){
     c[j] = a[j] + b[j] + c[j];
   }
 }
```

Compiler assignments

Register ← `i,j`

Memory (address = `0200`AS_1S_2B)

`a: 02000000 – 020007FF`

`b: 02001000 – 020017FF`

`c: 02002000 – 020027FF`

3 × 512 integers = 6 KB = 384 blocks

Memory accesses

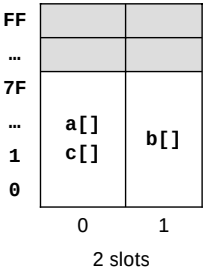
20 × 3 × 512 = 30720 read accesses to `a[], b[], c[]`

20 × 512 = 10240 write accesses to array `c[]`

Set assignment = `int(0200`AS_1S_2B`/10)%100 =` S_1S_2

`i = 0:` 3 × 512 / 4 = 384 compulsory misses

`i > 0:` 2 × 512 / 4 = 256 conflict misses for sets `a[], c[]`



Miss rate

$$\text{miss rate} = \frac{384 + 256 \times 19 \text{ misses}}{30720 + 10240 \text{ accesses}} = \frac{2688}{40960} = 0.128$$

Miss Example — Address Aliasing with Larger W

Cache parameters	B	S	W	Capacity = B × S × W
	16	128	4	8 KB = 2 Kwords

Program

```
int a[512],b[512],c[512],i,j;
for (i = 0 ; i < 20; i++){
  for (j = 0 ; j < 512 ; j++){
    c[j] = a[j] + b[j] + c[j];
  }
}
```

Compiler assignments

Register ← `i,j`

Memory (address = `0200`AS_1S_2B)

`a: 02000000 – 020007FF`

`b: 02001000 – 020017FF`

`c: 02002000 – 020027FF`

3 × 512 integers = 6 KB = 384 blocks

Memory accesses

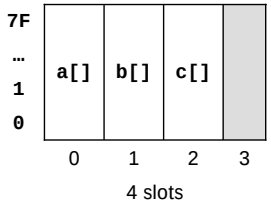
20 × 3 × 512 = 30720 read accesses to `a[], b[], c[]`

20 × 512 = 10240 write accesses to array `c[]`

Set assignment = `int(0200`AS_1S_2B`/10)%80 =` S_1S_2

`i = 0:` 3 × 512 / 4 = 384 compulsory misses

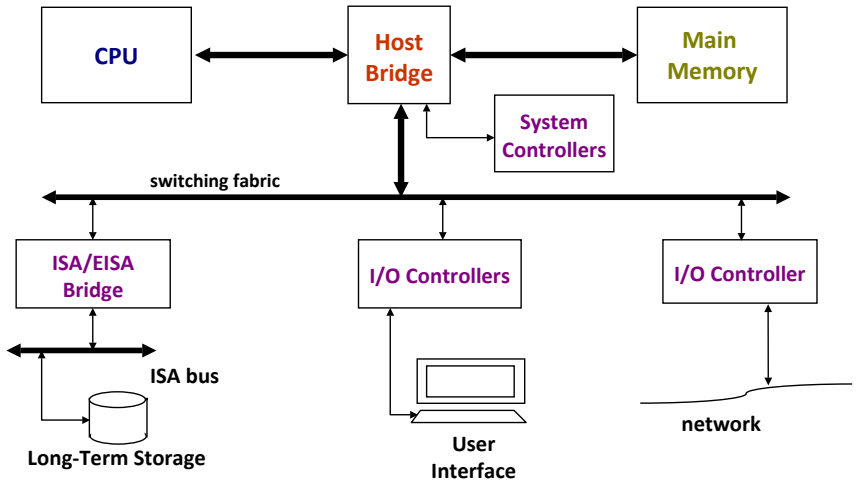
`i > 0:` All arrays in cache ⇒ cache hits



Miss rate

$$\text{miss rate} = \frac{384 \text{ misses}}{30720 + 10240 \text{ accesses}} = \frac{384}{40960} = 0.009375$$

Workstation Layout with PCI



ATA disk controllers

PATA — parallel ATA

SATA — serial ATA

PCI Services

Boot services

BIOS (basic input/output system)

ROM-based software for initiating system boot

Timers

System timers, counters and real time clocks

Interrupt controllers

Programmable interrupt control

IRQ — interrupt requests

Interrupt messages

Direct Memory Access (DMA)

Permit devices to access memory without CPU intervention

BIOS

Hardware system is started or reset

CPU performs self-check

CPU fetches instruction from address **FFFF0h**

Address **FFFF0h** contains branch instruction

Target of branch is firmware code located in PCI BIOS ROM

- ROM = Read Only Memory
- Usually E^2PROM = Electrically Erasable Programmable ROM

BIOS = BASIC INPUT/OUTPUT SYSTEM

BIOS

Locates keyboard, display, boot device

UEFI (Unified Extensible Firmware Interface) system

- BIOS load UEFI
- Hardware-oriented operating system that runs above firmware
- Performs system management including boot of main OS

Non-UEFI system

- BIOS begins loading OS from boot device

Interrupt Handling

APIC

Advanced Programmable Interrupt Control

Local APIC

Interrupt controller in CPU

Local interrupt

- INTR + int_number from device

Interrupt messages

- Structured message

I/O APIC

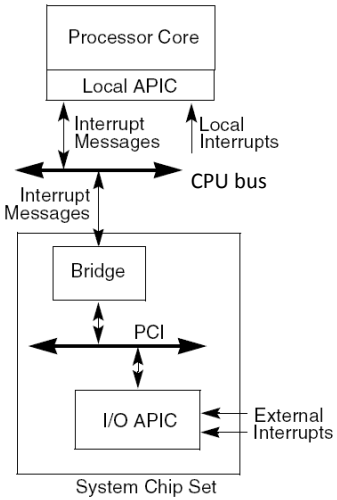
Interrupt controller in PCI chipset

Sends/receives interrupt messages

Replaces old IRQ system (each device assigned private IRQ)

All device interrupts defined as IRQ9

Interrupt message describes external event



Interprocessor Interrupts

CPU can send /receive Interprocessor Interrupt (IPI)

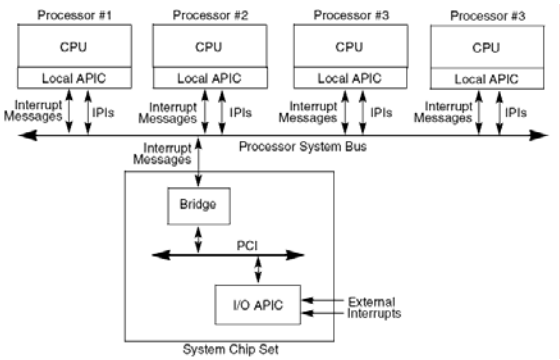
Used in multiprocessor (MP) systems

Standard APIC interrupt message syntax

Generating IPI message

CPU writes to interrupt command register (ICR) in local APIC

Local APIC issues IPI message on system bus



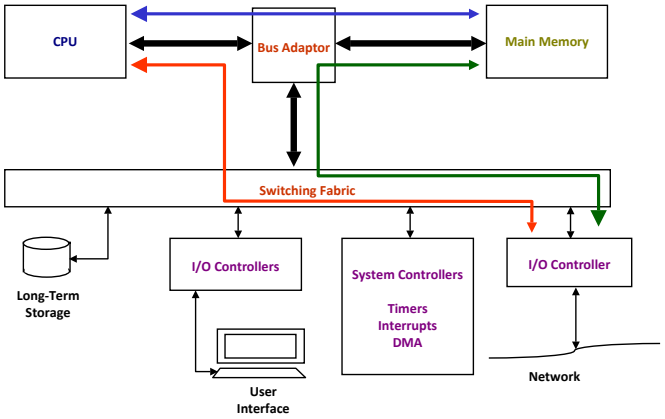
DMA — Direct Memory Access

Peripheral device accesses memory directly

- No need for CPU to execute transfer instructions
- Used for large data transfers

CPU works concurrently

- Can preempt DMA for cache update



DMA Operation

Interrupt mode

- CPU
  - Instructions to DMA controller set up transfer
  - Start address
  - Number of bytes to transfer
- DMA controller
  - Acts as master of data path
  - Transfers data between RAM and peripheral device
  - IRQ at end of transfer
- CPU takes back bus control

PCI arbitration

- PCI device
  - Requests control of bus
  - Requests read / write memory access
- PCI bridge
  - Grants bus control

Cache Coherency and Consistency

Multiple processors share data with main memory

- Each processor copies data blocks to cache
- Differences can develop between caches and/or main memory

Example

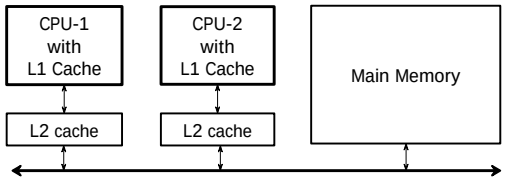
- CPU-1 and CPU-2 read X to cache from main memory
- CPU-1 writes to X in cache
- Invalid copies of X in CPU-2 cache and main memory

Consistency

- Copies of data locations are always identical

Coherence

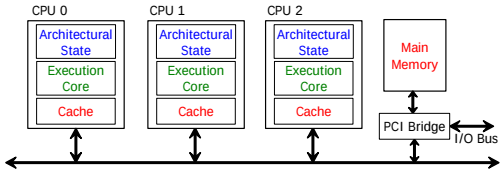
- Reads and writes occur in the correct order
- Easier than consistency



Bus Snooping

Write-back cache policy

- CPU updates private cache
  - No update to main memory
- CPU evicts cache block
  - Updates main memory



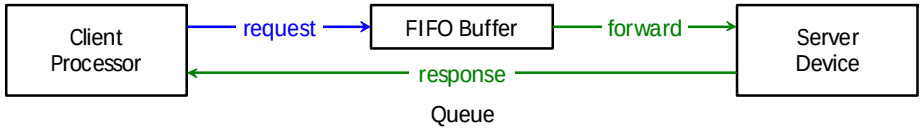
Bus snooping

- On CPU write
  - CPU always writes destination addresses on bus (short bus cycle)
  - CPU writes data to private cache (not on bus)
- Bus devices monitor all addresses written on bus
  - See which CPUs are loading a cache block
  - See which CPUs are writing to a cache block

Write synchronization

- Bus arbitration prevents writes to multiple copies of cache block
- Only one CPU places target address on memory bus per bus cycle
- Only one CPU can write to same cache block in one bus cycle

Client/Server Model of I/O



Fast CPU is client for slower I/O services

Buffer stores client requests and forwards at slower server response rate

Latency

Time between client request and buffer response

Throughput

Number of services provided per unit time

Bandwidth

Maximum data transfer rate of I/O channel (including buffer)

Capacity

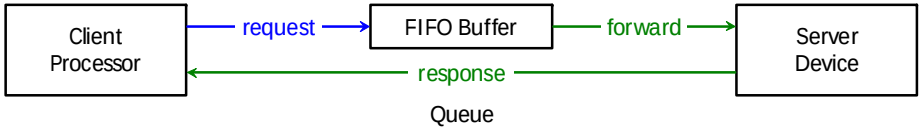
Maximum throughput of server through buffer

Depends on bandwidth and service rate (server speed)

Utilization

Request rate as proportion of capacity

Buffer Operation



Client requests

Intermittent high speed requests (**bursty**)

Peak client request rate >> **Average** client request rate

Service responses

Peak client rate > Server rate > **Average** client rate

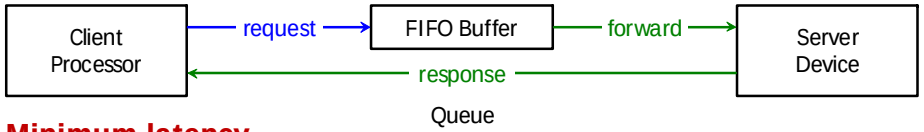
FIFO buffers requests in order of arrival

Stores requests arriving at higher client request rate

Forwards requests to server at lower server response rate

Request forwarding rate = server response rate

Buffer Latency and Overflow



Minimum latency

Determined by maximum service response rate

Buffer overflow

Buffer fill rate = Client Request Rate – Service Response Rate

Buffer fills continuously for too long ⇒ buffer overflow

Example

Peak CPU disk read rate = 1 read/cycle = 10^9 read requests/second

Disk can provide = 10^7 responses per second = 100 CPU cycles/read

CPU sees minimum latency of about 100 CPU cycles

Buffer can hold 1000 requests

Continuous requests ⇒ overflow in $1000/(10^9 - 10^7) \sim 10^{-6}$ seconds

Utilization — Latency Trade-Off

Utilization

Higher client request rate

More services per second ⇒ Higher utilization

Server cannot work faster (service rate is fixed)

More requests are buffered ⇒ longer queue length (higher buffer level)

Total latency for one request = server latency + queuing time

More requests in buffer queue ⇒ longer queuing time

Higher utilization ⇒ Higher total latency

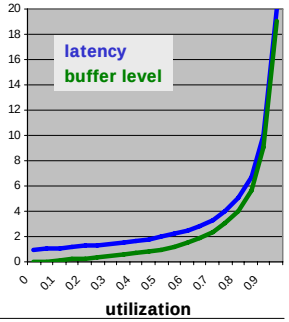
Buffer overflow

Average request rate > average response rate

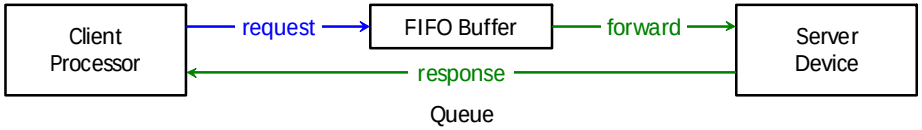
More requests enter buffer than leave

Buffer level rises

After long time, buffer overflows



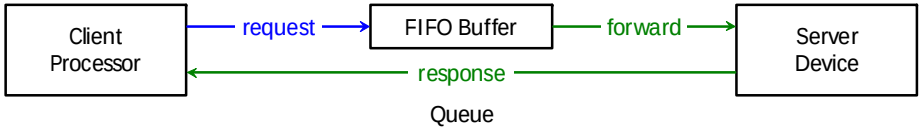
Queuing Theory — 1



Assumptions

- Client Requests
  - Arrive independently (Poisson statistics)
  - Have random length (bytes to transfer)
  - **Average Request Rate** in steady state
- Buffer
  - Stores requests and forwards in order of arrival (FIFO) at service rate
  - **Average Buffer Level** (stored requests) in steady state
- Server
  - Provides services to each request independently (Poisson statistics)
  - **Average Service Rate** in steady state

Queuing Theory — 2

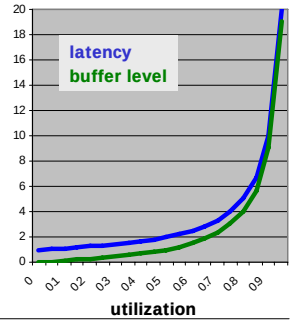


Results

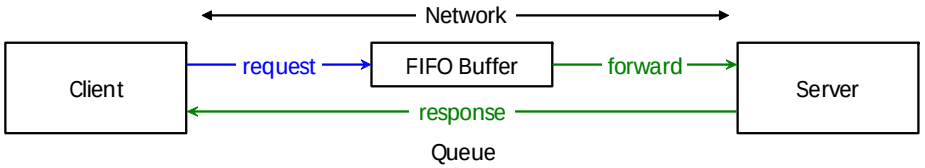
$$\text{Utilization} = \frac{\text{Request Rate}}{\text{Service Rate}}$$

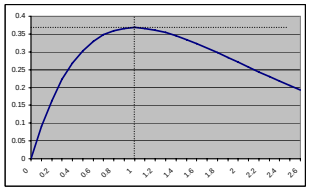
$$\text{Latency} = \frac{1}{\text{Service Rate} - \text{Request Rate}} = \frac{1}{\text{Service Rate}}\left(\frac{1}{1 - \text{Utilization}}\right)$$

$$\text{Buffer Level} = \text{Latency} \times \text{Request Rate} = \text{Utilization} \times \left(\frac{1}{1 - \text{Utilization}}\right)$$



Traffic Shaping





High utilization causes congestion

- Higher packet error rate (noise + collisions)
- Buffer overflow
- Re-transmit lost packets ⇒ more requests ⇒ more collisions

Traffic shaping

- Buffer at client imposes **request quotas** on client
- Client request rate = maximum transmission rate on network
- Forward rate = actual transmission rate = optimum network rate