Chapter 2: Memory Hierarchy Design

Introduction (Section 2.1, Appendix B)

Caches

- Review of basics (Section 2.1, Appendix B)
- Advanced methods

Main Memory

Virtual Memory

Memory Hierarchies: Key Principles

Make the common case fast

Common → Principle of locality

Fast → Smaller is faster

Principle of Locality

Temporal locality

Spatial locality

Examples:

*Principle of Locality***

Temporal locality

Locality in time

If a datum has been recently referenced, it is likely to be referenced again

Spatial locality

Examples:

Principle of Locality**

Temporal locality

- Locality in time
- If a datum has been recently referenced, it is likely to be referenced again

Spatial locality

- Locality in space
- When a datum is referenced, neighboring data are likely to be referenced soon

Examples:

*Principle of Locality***

Temporal locality

- Locality in time
- If a datum has been recently referenced, it is likely to be referenced again

Spatial locality

- Locality in space
- When a datum is referenced, neighboring data are likely to be referenced soon

Examples:

- Temporal locality: Top of stack, Code in a loop
- Spatial locality: Top of stack, Sequential instructions, Structure references

Smaller is Faster

Registers are fastest memory

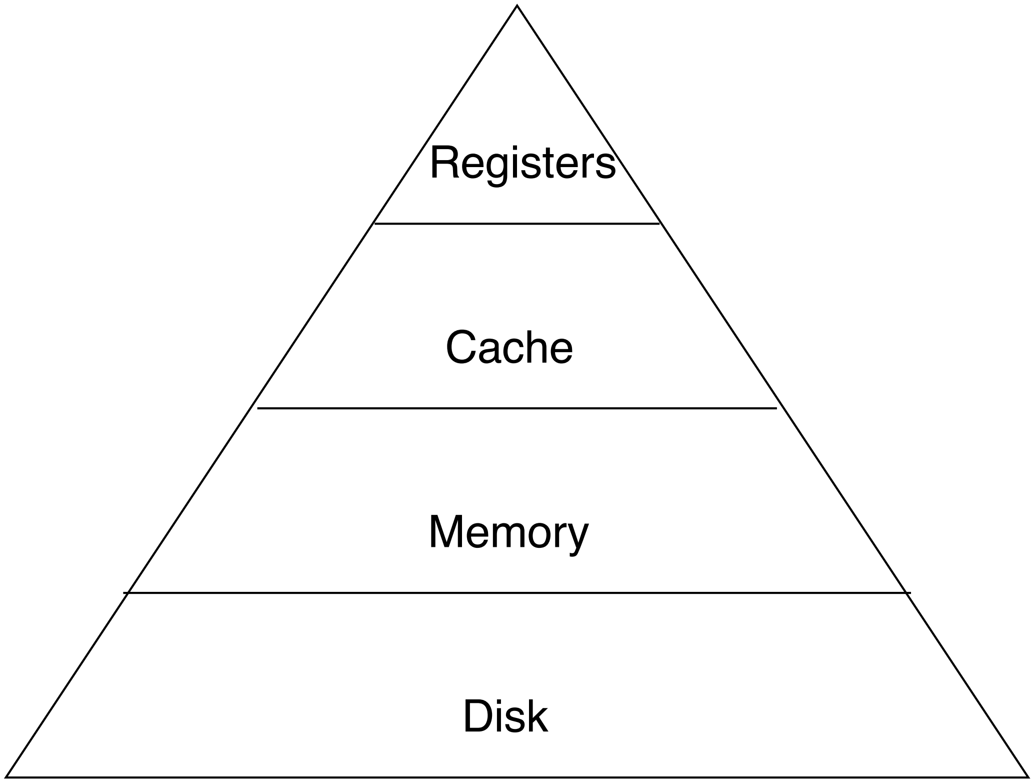
- Smallest and most expensive

Static RAMs are faster than DRAMs

- 10X faster
- 10X less dense

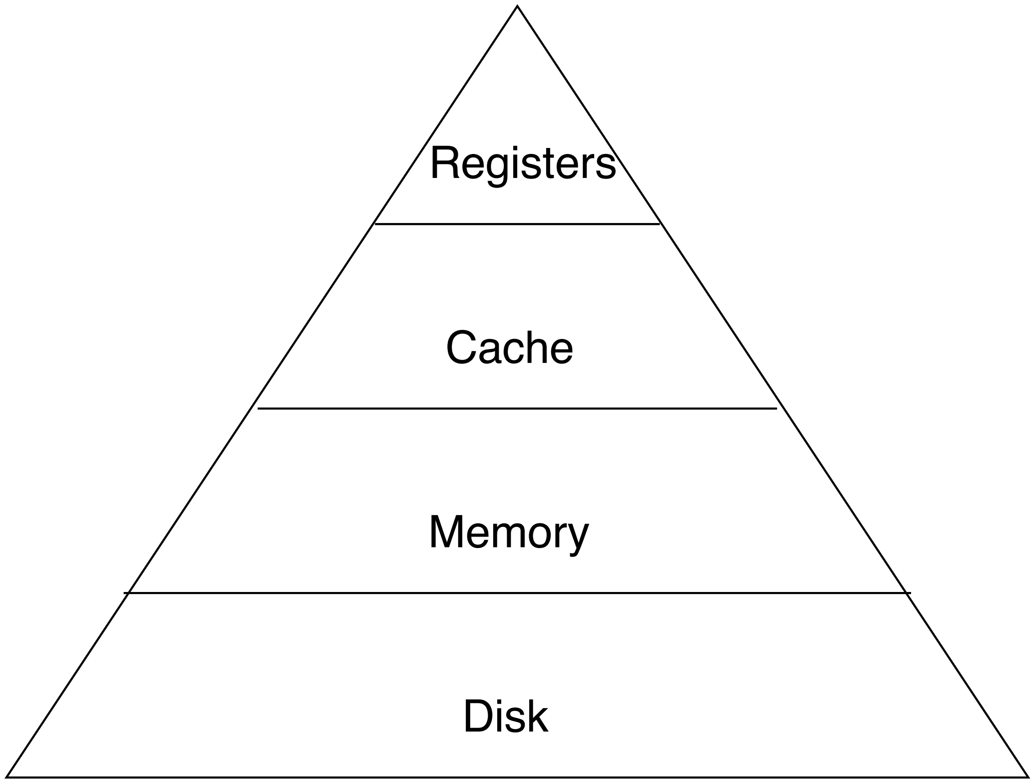
DRAMs are faster than disk, flash

Memory Hierarchy



Type	Size	Speed (x proc. clk)
Registers		
Cache		
Memory		
Disk, Flash		

*Memory Hierarchy***



Type	Size	Speed (x proc. clk)
Registers	32 to 128 I and F	1X
Cache	10s of KB to 10s of MB	~1 to 10X on-chip, ~10X off-chip
Memory	GB	~100X
Disk, Flash	GB to TB to …	~1000000X

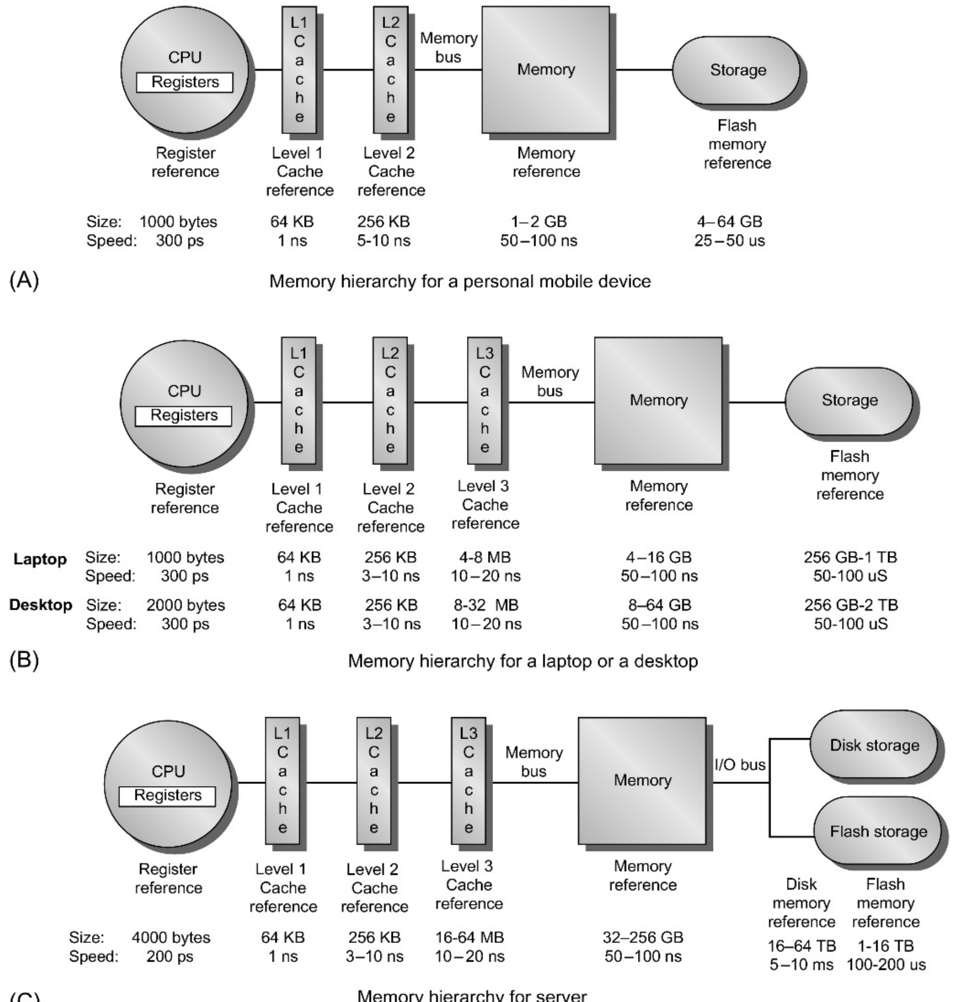


Figure 2.1 The levels in a typical memory hierarchy in a personal mobile device (PMD), such as a cell phone or tablet (A), in a laptop or desktop computer (B), and in a server (C). As we move farther away from the processor, the memory in the level below becomes slower and larger. Note that the time units change by a factor of 10^9 from picoseconds to milliseconds in the case of magnetic disks and that the size units change by a factor of 10^{10} from thousands of bytes to tens of terabytes. If we were to add warehouse-sized computers, as opposed to just servers, the capacity scale would increase by three to six orders of magnitude. Solid-state drives (SSDs) composed of Flash are used exclusively in PMDs, and heavily in both laptops and desktops. In many desktops, the primary storage system is SSD, and expansion disks are primarily hard disk drives (HDDs). Likewise, many servers mix SSDs and HDDs.

Memory Hierarchy Terminology

Block

- Minimum unit that may be present
- Usually fixed length

Hit – Block is found in upper level

Miss – Not found in upper level

Miss ratio – Fraction of references that miss

Hit Time – Time to access the upper level

Miss Penalty

- Time to replace block in upper level, plus the time to deliver the block to the CPU
- Access time – Time to get first word
- Transfer time – Time for remaining words

Memory Hierarchy Terminology

Memory Address

Block-frame address	Offset
0101010101010101011	01010101

Block Names

Cache: Line

VM: Page

Memory Hierarchy Performance

Indirect measures of time can be misleading

MIPS can be misleading

So can Miss ratio

Average (effective) access time is better

t_{avg} =

Example:

t_{hit} = 1

t_{miss} = 20

miss ratio = .05

t_{avg} =

Effective access time is still an indirect measure

*Memory Hierarchy Performance***

Time is always the ultimate measure

Indirect measures can be misleading

MIPS can be misleading

So can Miss ratio

Average (effective) access time is better

$$t_{avg} = t_{hit} + miss\ ratio \times t_{miss}$$

$$= t_{cache} + miss\ ratio \times t_{memory}$$

Example:

$t_{hit} = 1$

$t_{miss} = 20$

miss ratio = .05

$t_{avg} =$

Effective access time is still an indirect measure

Memory Hierarchy Performance**

Time is always the ultimate measure

Indirect measures can be misleading

MIPS can be misleading

So can Miss ratio

Average (effective) access time is better

$$t_{avg} = t_{hit} + miss\ ratio \times t_{miss}$$
$$= t_{cache} + miss\ ratio \times t_{memory}$$

Example:

$t_{hit} = 1$

$t_{miss} = 20$

miss ratio = .05

$t_{avg} = 1 + .05 \times 20 = 2$

Effective access time is still an indirect measure

Example

Poor question:

Q: What is a reasonable miss ratio?

A: 1%, 2%, 5%, 10%, 20% ???

A better question

Q: What is a reasonable t_{avg} ?

(assume t_{cache} = 1 cycle, t_{memory} = 20 cycles)

A: 1.2, 1.5, 2.0 cycles

What's a reasonable t_{avg} ?

*Example***

Poor question:

Q: What is a reasonable miss ratio?

A: 1%, 2%, 5%, 10%, 20% ???

A better question

Q: What is a reasonable t_{avg} ?

(assume t_{cache} = 1 cycle, t_{memory} = 20 cycles)

A: 1.2, 1.5, 2.0 cycles

What's a reasonable t_{avg} ?

Depends upon base CPI

t_{avg} = 2.0 might be OK for base $CPI = 10$,

but terrible for base $CPI = 1.2$

Example, cont.

Rearranging terms in

$$t_{avg} = t_{cache} + miss\ ratio \times t_{memory}$$

to solve for miss ratios yields

$$miss = \frac{(t_{avg} - t_{cache})}{t_{memory}}$$

Reasonable miss ratios (percent) - assume $t_{cache} = 1$

t_{memory} (cycles)	t_{avg} (cycles)		
	1.2	1.5	2.0
2	10.0	25.0	50.0
20	1.0	2.5	5.0
200	0.1	0.25	0.5

Proportional to acceptable t_{avg} degradation

Inversely proportional to t_{memory}

Basic Cache Questions

Block placement

Where can a block be placed in the cache?

Block Identification

How is a block found in the cache?

Block replacement

Which block should be replaced on a miss?

Write strategy

What happens on a write?

Cache Type

What type of information is stored in the cache?

Block Placement

FullyAssociative

Block goes in any block frame

Directmapped

Block goes in exactly one block frame

(Block frame #) mod (# of blocks)

SetAssociative

Block goes in exactly one set

(Block frame #) mod (# of sets)

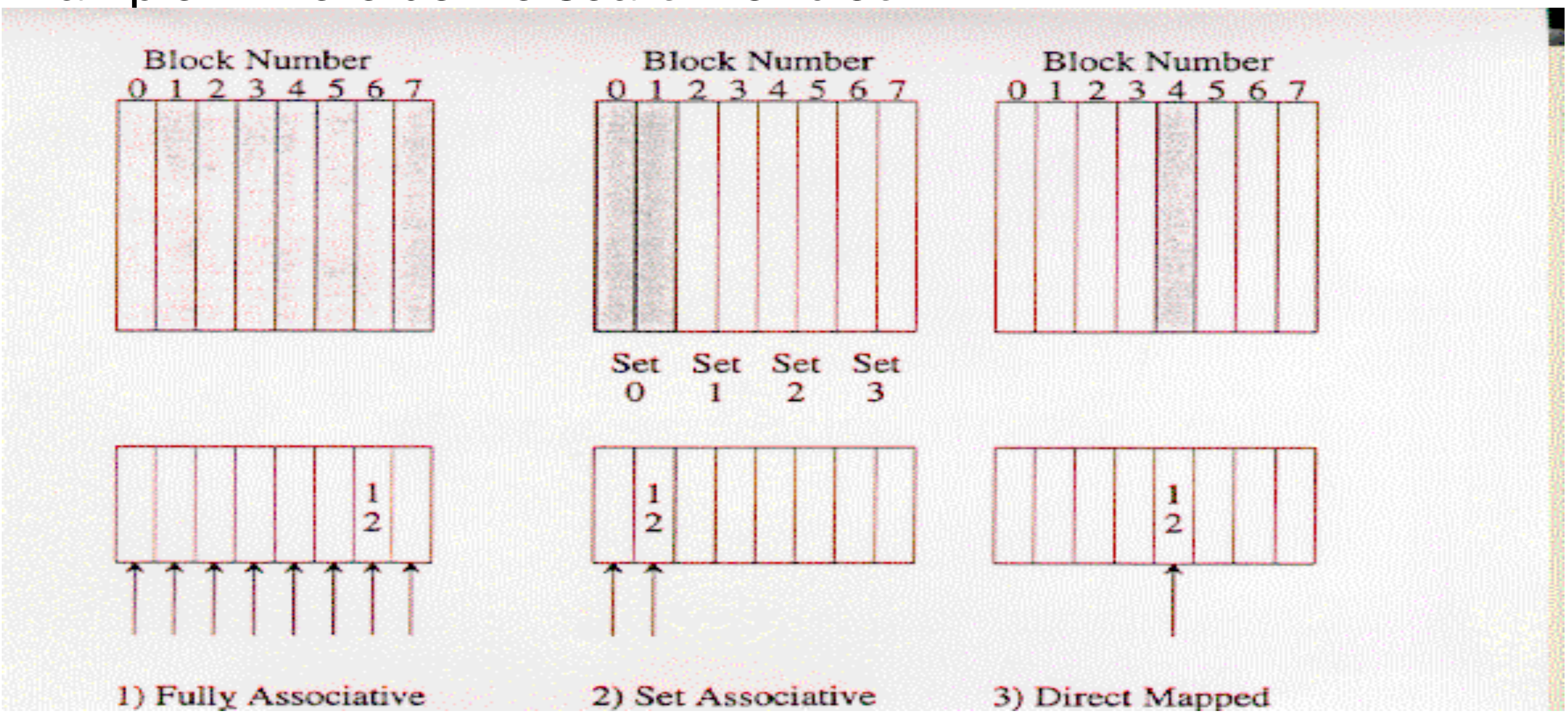
Example: Consider cache with 8 blocks, where does block 12 go?

Block Identification

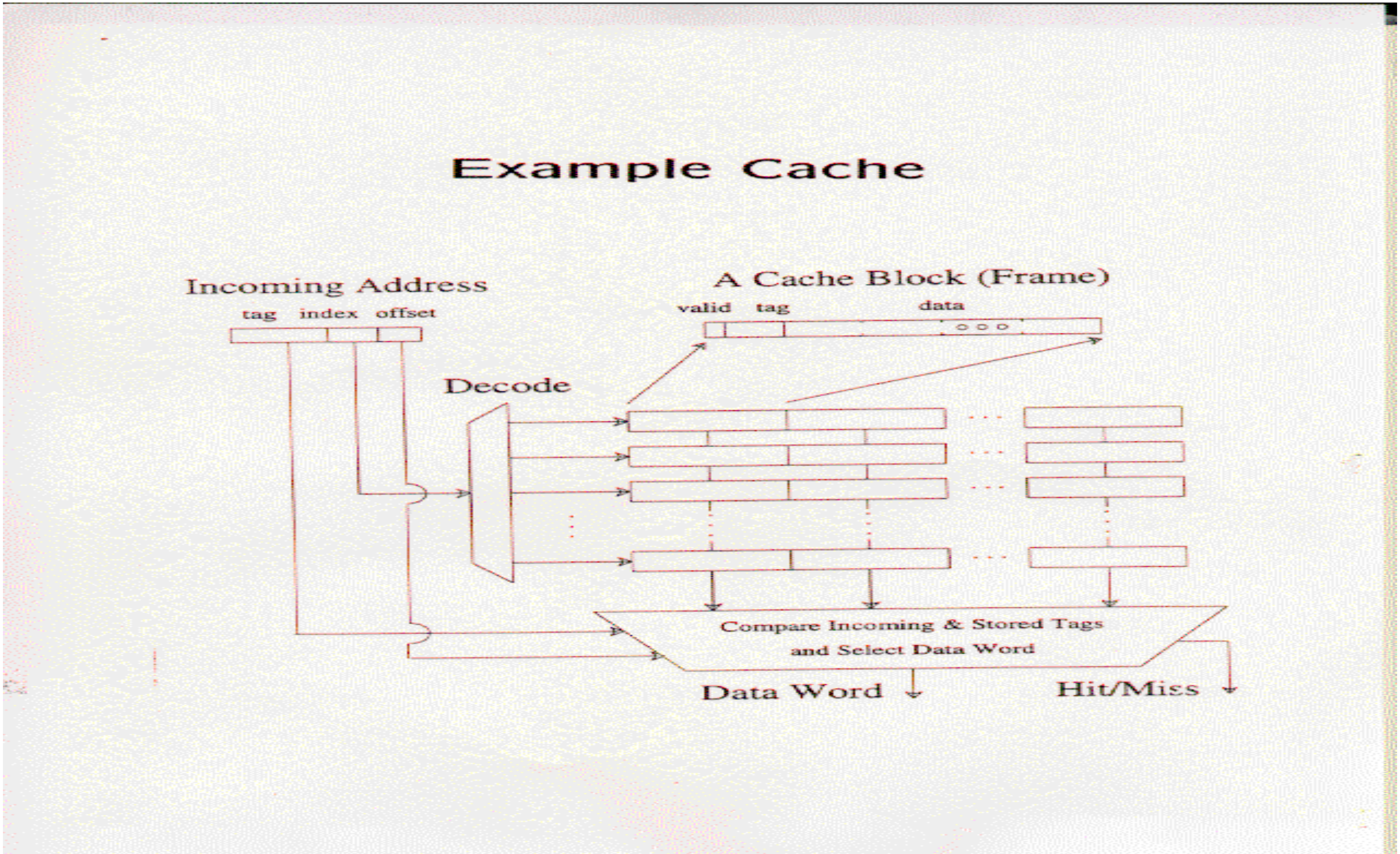
How to find the block?

- Tag comparisons
- Parallel search to speed lookup
- Check valid bit

Example: Where do we search for block 12?



Example Cache



Block Replacement

Which block to replace on a miss?

Least recently used (LRU)

- Optimize based on temporal locality
- Replace block unused for longest time
- State updates on nonMRU misses

Random

- Select victim at random
- Nearly as good as LRU, and easier

First-in First-out (FIFO)

- Replace block loaded first

Optimal

- ?

Block Replacement **

Which block to replace on a miss?

Least recently used (LRU)

- Optimize based on temporal locality
- Replace block unused for longest time
- State updates on non-MRU misses

Random

- Select victim at random
- Nearly as good as LRU, and easier

First-in First-out (FIFO)

- Replace block loaded first

Optimal

- Replace block used furthest in time

Write Policies

Writes are harder

- Reads done in parallel with tag compare; writes are not
- Thus, writes are often slower
- (but processor need not wait)

On hits, update memory?

- Yes writethrough (storethrough)
- No writeback (storein, copyback)

On misses, allocate cache block?

- Yes write-allocate (usually used w/ writeback)
- No no-write-allocate (usually used w/ writethrough)

Write Policies, cont.

WriteBack

- Update memory only on block replacement
- Dirty bits used so clean blocks can be replaced without updating memory
- Traffic/Reference =
- Traffic/Reference =
- Less traffic for larger caches

WriteThrough

- Update memory on each write
- Write buffers can hide write latency (later)
- Keeps memory uptodate (almost)
- Traffic/Reference =

Write Policies, cont.**

WriteBack

- Update memory only on block replacement
- Dirty bits used so clean blocks can be replaced without updating memory
- Traffic/Reference = $fractDirty \times miss \times B$
- Traffic/Reference = 1/2 × 0.05 × 4 = 0.10
- Less traffic for larger caches

WriteThrough

- Update memory on each write
- Write buffers can hide write latency (later)
- Keeps memory uptodate (almost)
- Traffic/Reference =

Write Policies, cont.**

WriteBack

- Update memory only on block replacement
- Dirty bits used so clean blocks can be replaced without updating memory
- Traffic/Reference = $fractDirty \times miss \times B$
- Traffic/Reference = 1/2 × 0.05 × 4 = 0.10
- Less traffic for larger caches

WriteThrough

- Update memory on each write
- Write buffers can hide write latency (later)
- Keeps memory uptodate (almost)
- Traffic/Reference = $fractionWrites$ = 0.20
- Traffic independent of cache parameters

Cache Type

Unified (mixed)

- Less costly
- Dynamic response
- Handles writes into Istream

Separate Instruction & Data (split, Harvard)

- 2x bandwidth
- Place closer to I and D ports
- Can customize
- Poorman's associativity
- No interlocks on simultaneous requests

Caches should be split if simultaneous instruction and data accesses are frequent (e.g., RISCs)

Cache Type Example

Consider building (a)16K byte I & D caches, or (b) a 32K byte unified cache.

Let t_{cache} is one cycle, t_{memory} is 10 cycles.

(a) $Imiss$ is 5 %, $Dmiss$ is 6 %, 75 % of references are instruction fetches.

t_{avg} =

(b) miss ratio is 4 %

t_{avg} =

Cache Type Example**

Consider building (a)16K byte I & D caches, or (b) a 32K byte unified cache.

Let t_{cache} is one cycle, t_{memory} is 10 cycles.

(a) $Imiss$ is 5 %, $Dmiss$ is 6 %, 75 % of references are instruction fetches.

$$t_{avg} = (1 + 0.05 \times 10) \times 0.75 + (1 + 0.06 \times 10) \times 0.25 = 1.5$$

(b) miss ratio is 4 %

$$t_{avg} =$$

*Cache Type Example***

Consider building (a)16K byte I & D caches, or (b) a 32K byte unified cache.

Let t_{cache} is one cycle, t_{memory} is 10 cycles.

(a) $Imiss$ is 5 %, $Dmiss$ is 6 %, 75 % of references are instruction fetches.

$$t_{avg} = (1 + 0.05 \times 10) \times 0.75 + (1 + 0.06 \times 10) \times 0.25 = 1.5$$

(b) miss ratio is 4 %

$$t_{avg} = 1 + 0.04 \times 10 = 1.4$$

*Cache Type Example***

Consider building (a)16K byte I & D caches, or (b) a 32K byte unified cache.

Let t_{cache} is one cycle, t_{memory} is 10 cycles.

(a) $Imiss$ is 5 %, $Dmiss$ is 6 %, 75 % of references are instruction fetches.

$$t_{avg} = (1 + 0.05 \times 10) \times 0.75$$
$$+ (1 + 0.06 \times 10) \times 0.25 = 1.5$$

(b) miss ratio is 4 %

$$t_{avg} = 1 + 0.04 \times 10 = 1.4 \text{ WRONG!}$$

$$t_{avg} = 1.4 + \text{cycles-lost-to-interference}$$

Will cycles-lost-to-interference < 0.1?

Not for "RISC" machines!

A Miss Classification (3Cs or 4Cs)

Cache misses can be classified as:

Compulsory (a.k.a. cold start)

The first access to a block

Capacity

Misses that occur when a replaced block is re-referenced

Conflict (a.k.a. collision)

Misses that occur because blocks are discarded because of the set-mapping strategy

Coherence (shared-memory multiprocessors)

Misses that occur because blocks are invalidated due to references by other processors