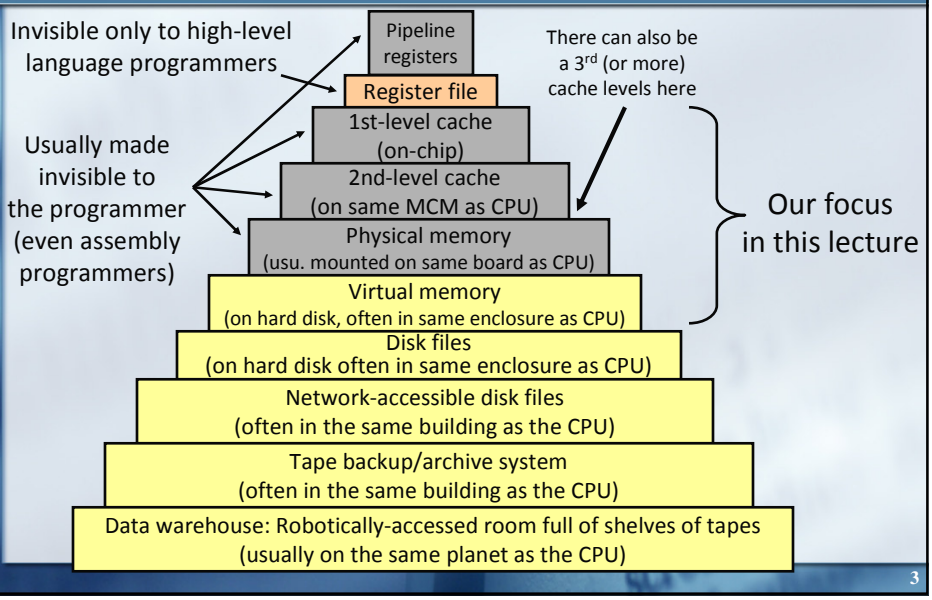
CDA 5106 Advanced Computer Architecture 1

Module 6 | Memory Hierarchy Review

Review of Memory Hierarchy

- Introduction
- Cache Basics
- Cache Performance
- Six Basic Cache Optimizations
- Virtual Memory
- Conclusion

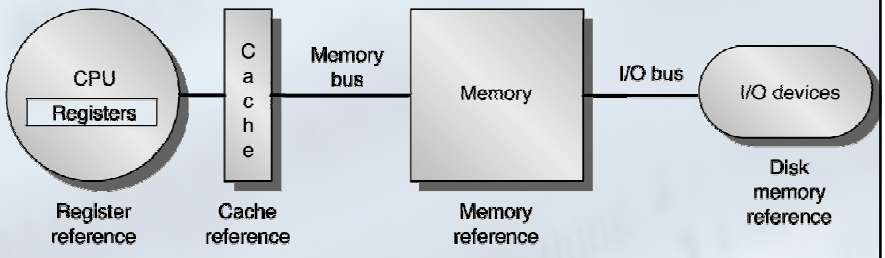
Many Levels in Memory Hierarchy



3

Simple Hierarchy Example

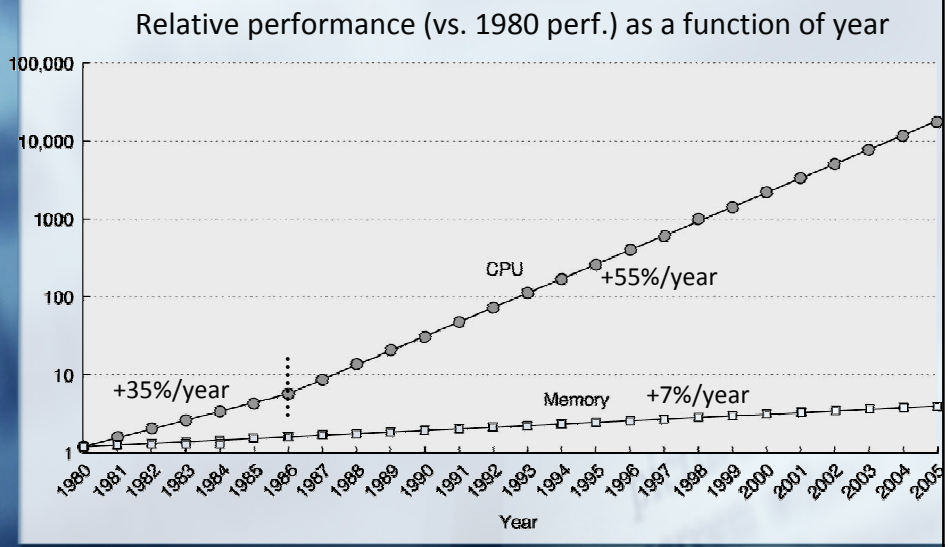
- Note many orders of magnitude change in characteristics between levels:



Size	1 KB	16 MB	512 GB	1 TB
Access Time (ns)	0.25-0.5	0.5-25	50-250	5,000,000
Bandwidth (MB/s)	50K-500K	5,000-20,000	2,500-10,000	50-500

4

CPU vs. Memory Performance Trends



5

Review of Memory Hierarchy

- Introduction
- Cache Basics
- Cache Performance
- Six Basic Cache Optimizations
- Virtual Memory
- Conclusion

6

Cache Basics

- A *cache* is a (hardware managed) storage, intermediate in size, speed, and cost-per-bit between the programmer-visible registers and main physical memory
- The cache itself may be SRAM or fast DRAM
- There may be more than one level of caches
- Basis for cache to work: Principle of Locality
- When a location is accessed, it *and "nearby"* locations are likely to be accessed again soon
  - "Temporal" locality - Same location likely again soon
  - "Spatial" locality - Nearby locations likely

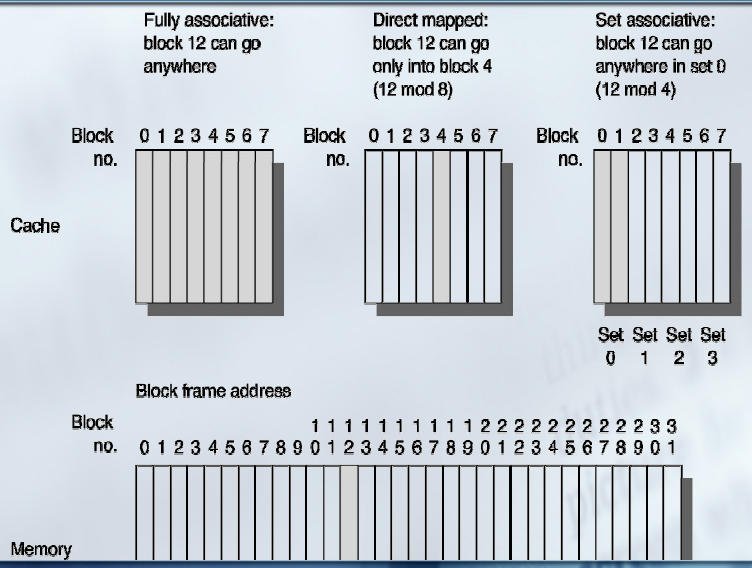
7

Four Basic Questions

- Consider levels in a memory hierarchy
  - Use block as unit of data transfer between cache levels and memory; satisfy Principle of Locality
- The level design is described by four behaviors
  - Block Placement:
    - Where could a new block be placed in the level?
  - Block Identification:
    - How is a block found if it is in the level?
  - Block Replacement:
    - Which existing block should be replaced if necessary?
  - Write Strategy:
    - How are writes to the block handled?

8

Block Placement Schemes



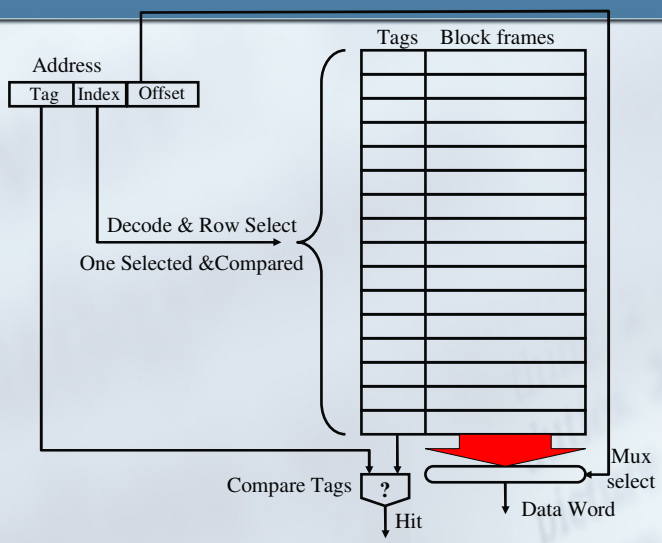
9

Direct-Mapped Placement

- A block can only go into one frame in the cache
  - Determined by block's address (in memory space)
    - Frame number usually given by some low-order bits of block address
- This can also be expressed as:
  - (Frame number) = (Block address) ***mod*** (Number of frames/sets in cache)
- In a direct-mapped cache
  - block placement & replacement are both completely determined by the address of the new block that is to be accessed

10

Direct-Mapped Identification



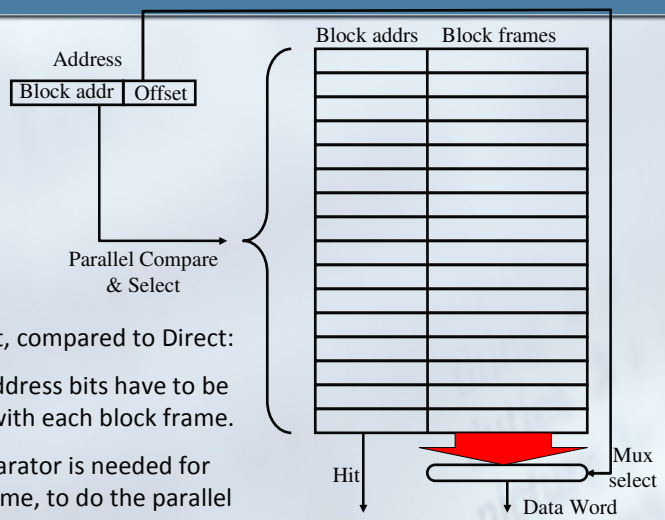
11

Fully-Associative Placement

- One alternative to direct-mapped is:
  - Allow block to fill ***any*** empty frame in the cache
- How do we then locate the block later?
  - Can *associate* each stored block with a *tag*
    - Identifies the block's location in cache
  - When the block is needed, treat the cache as an *associative memory*, using the tag to match all frames in parallel, to pull out the appropriate block
- Another alternative to direct-mapped is placement under full program control
  - A register file can be viewed as a small programmer-controlled cache (w. 1-word blocks)

12

Fully-Associative Identification



Note that, compared to Direct:

- More address bits have to be stored with each block frame.
- A comparator is needed for *each* frame, to do the parallel associative lookup.

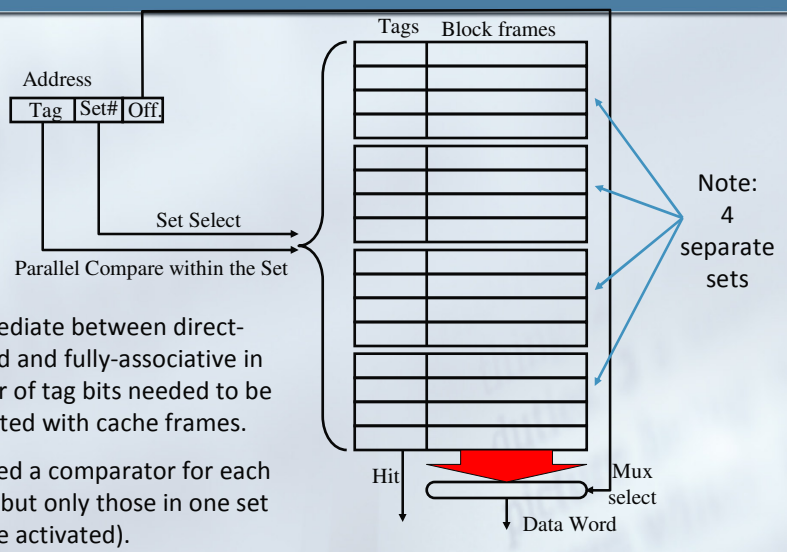
13

Set-Associative Placement

- The block address determines not a single frame, but a *frame set* (several frames, grouped together)
  - Frame set # = Block address **mod** # of frame sets
- The block can be placed associatively anywhere within that frame set
- If there are *n* frames in each frame set, the scheme is called "*n*-way set-associative"
- Direct mapped = 1-way set-associative
- Fully associative: There is only 1 frame set

14

Set-Associative Identification

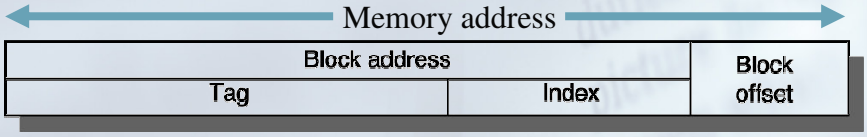


- Intermediate between direct-mapped and fully-associative in number of tag bits needed to be associated with cache frames.
- Still need a comparator for each frame (but only those in one set need be activated).

15

Cache Size Equation

- Simple equation for the size of a cache:
  - (Cache size) = (Block size) × (Number of sets) × (Set Associativity)
- Can relate to the size of various address fields:
  - (Block size) = $2^{(\#\text{ of offset bits})}$
  - (Number of sets) = $2^{(\#\text{ of index bits})}$
  - (# of tag bits) = (# of memory address bits) – (# of index bits) – (# of offset bits)



16

Replacement Strategies

- Which block do we replace when a new block comes in (on cache miss)?
- Direct-mapped: There's only one choice!
- Associative (fully- or set-):
  - If any frame in the set is empty, pick one of those.
  - Otherwise, there are many possible strategies:
    - Random: Simple, fast, and fairly effective
    - Least-Recently Used (LRU), and approximations thereof
      - Require bits to record replacement info., e.g. 4-way requires 4! = 24 permutations, need 5 bits to define the MRU to LRU positions
    - FIFO: Replace the oldest block.

17

Implementation of LRU Replacement

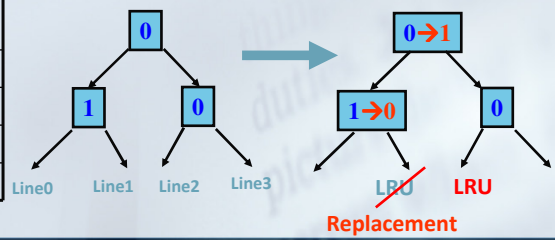
- Pure LRU, 4-way ➔ use 6 bits (minimum 5 bits)

1→2	1→3	1→4	2→3	2→4	3→4

- Partitioned LRU (Pseudo LRU):
  - Instead of recording full combination, use a binary tree to maintain only (n-1) bits for n-way set associativity
  - 4-way example: '1' represents left side is MRU and vice versa

State	Replace	Next State
00x	Line0	11_
01x	Line1	10_
1x0	Line2	0_1
1x1	Line3	0_0

x : don't care _ : remains same



18

Write Strategies

- Most accesses are reads, not writes
  - Especially if instruction reads are included
- Optimize for reads – performance matters
  - Direct mapped can return value before valid check
- Writes are more difficult
  - Can't write to cache until we *know* the right block
  - Object written may have various sizes (1-8 bytes)
- When to synchronize cache with memory?
  - *Write through* - Write to cache and to memory
    - Prone to stalls due to high bandwidth requirements
  - *Write back* - Write to memory upon replacement
    - Memory may be out of date

19

Another Write Strategy

- Maintain a FIFO queue (write buffer) of cache frames (*e.g.* can use a doubly-linked list)
- Meanwhile, take items from top of queue and write them to memory as fast as bus can handle
  - Reads might take priority, or have a separate bus
- Advantages: *Write stalls* are minimized, while keeping memory as up-to-date as possible

20

Write Miss Strategies

- What do we do on a write to a block that's not in the cache?
- Two main strategies: Both do not stop processor
  - Write-allocate (fetch on write) - cache the block
  - No write-allocate (write around) - write to memory
- Write-back caches tend to use write-allocate
- Write-through tends to use no-write-allocate
- Use *dirty bit* to indicate write-back is needed in write-back strategy

21

Write-Allocate vs. No-Write-Allocate Policy

- Example code:

```
WriteMem[100];
WriteMem[200];
ReadMem[200];
WriteMem[200];
WriteMem[100].
```

Assume cache starts empty. Calculate number of hits and misses for each write miss policy.

Op	Write-Allocate	No-Write-Allocate
WriteMem[100]	Miss	Miss
WriteMem[200]	Miss	Miss
ReadMem[200]	Hit	Miss
WriteMem[200]	Hit	Hit
WriteMem[100]	Hit	Miss

22

Instruction vs. Data Caches

- Instructions and data have different patterns of temporal and spatial locality
  - Also instructions are generally read-only
- Can have *separate* instruction & data caches
  - Advantages
    - Doubles bandwidth between CPU & memory hierarchy
    - Each cache can be optimized for its pattern of locality
  - Disadvantages
    - Slightly more complex design
    - Can't dynamically adjust cache space taken up by instructions vs. data

24

Inst./Data Split and Unified Caches

Size	I-Cache	D-Cache	Unified Cache
8KB	8.16	44.0	63.0
16KB	3.82	40.9	51.0
32KB	1.36	38.4	43.3
64KB	0.61	36.9	39.4
128KB	0.30	35.3	36.2
256KB	0.02	32.6	32.9

- Misses per 1000 accesses
- Much lower instruction miss rate than data miss rate

25

Review of Memory Hierarchy

- Introduction
- Cache Basics
- Cache Performance
- Six Basic Cache Optimizations
- Virtual Memory
- Conclusion

26

Basic Cache Performance Formulas

$$Miss\ Rate = \frac{Misses\ per\ instruction}{Memory\ accesses\ per\ instruction}$$

- Memory access - data transfer requests (on load/store) and instruction memory access (always 1 per instruction)

$$Average\ memory\ access\ time = Hit\ time + Miss\ rate \times Miss\ penalty$$

- Units of measurement:
  - Hit time, Miss penalty – *ns* or *clock cycles*
  - Miss rate – unitless
  - Average memory access time – *ns* or *clock cycles*

27

Cache Performance Equations

- Memory stalls per program (blocking cache):

$$Memory\ Stall\ Cycles = IC \times (\frac{Memory\ Accesses}{Instruction}) \times Miss\ Rate \times Miss\ Penalty$$

$$Memory\ Stall\ Cycles = IC \times (\frac{Misses}{Instruction}) \times Miss\ Penalty$$

- CPU time formula:

$$CPU\ Time = IC \times (CPI_{Execu} + \frac{Memory\ Stall\ Cycles}{Instruction}) \times Cycle\ Time$$

- More cache performance later

28

Cache Performance Example

- Ideal CPI=2.0, memory references / inst=1.5, cache size=64KB, miss penalty=75ns, hit time=1 clock cycle
- Compare performance of two caches:
  - Direct-mapped (1-way): cycle time=1ns, miss rate=1.4%
  - 2-way: cycle time=1.25ns, miss rate=1.0%
  - Quick test – calculate average memory access time for each?

$$Miss\ Penalty_{1-way} = \left\lceil \frac{75ns}{1ns} \right\rceil = 75\ Cycles$$

$$Miss\ Penalty_{2-way} = \left\lceil \frac{75ns}{1.25ns} \right\rceil = 60\ Cycles$$

$$CPU\ Time_{1-way} = IC \times (2.0 + 1.4\% \times 1.5 \times 75) \times 1ns = 3.575 \times IC$$

$$CPU\ Time_{2-way} = IC \times (2.0 + 1\% \times 1.5 \times 60) \times 1.25ns = 3.625 \times IC$$

29

Out-Of-Order Processor

- Define new "miss penalty" considering overlap

$$\frac{Mem.Stall\,Cycles}{Instruction} = \frac{Misses}{Instruction} \times (Total\,miss\,latency - Overlap\,miss\,latency)$$

  - Compute *memory latency* and *overlapped latency*
- Example (from previous slide)
  - Assume 30% of 75ns penalty can be overlapped, but with longer (1.25ns) cycle on 1-way design due to Out-of-Order

$$Miss\ Penalty_{1-way} = \left\lceil \frac{75ns \times 70\%}{1.25ns} \right\rceil = 42\ Cycles$$

$$CPU\ Time_{1-way,OOO} = IC \times (2.0 + 1.4\% \times 1.5 \times 42) \times 1.25ns = 3.60 \times IC$$

30

Cache Performance Improvement

Average memory access time = (Hit time) + (Miss rate)×(Miss penalty)

"Amortized miss penalty"

- Reduce miss penalty:
  - Multilevel cache; Critical word first and early restart; priority to read miss; Merging write buffer; Victim cache
- Reduce miss rate:
  - Larger block size; Increase cache size; Higher associativity; Way prediction and Pseudo-associative caches; Compiler optimizations
- Reduce miss penalty/rate via parallelism:
  - Non-blocking cache; Hardware and Compiler-controlled prefetching
- Reduce hit time:
  - Small simple cache; Avoid address translation in indexing cache; Pipelined cache access; Trace caches

31

Review of Memory Hierarchy

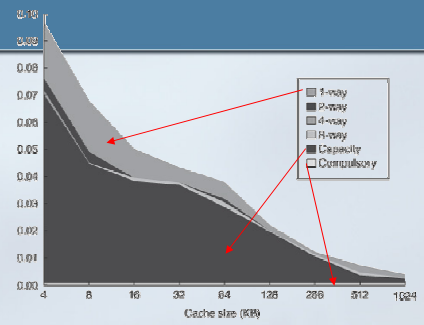
- Introduction
- Cache Basics
- Cache Performance
- Six Basic Cache Optimizations
- Virtual Memory
- Conclusion

Three Types of Misses

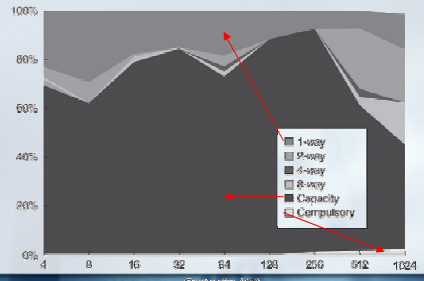
- Compulsory
  - During a program, the very first access to a block will not be in the cache (unless pre-fetched)
- Capacity
  - The *working set* of blocks accessed by the program is too large to fit in the cache
- Conflict
  - Unless cache is fully associative, sometimes blocks may be evicted too early because too many frequently-accessed blocks map to the same limited set of frames/sets.

Miss Rate Analysis

Total Miss Rate



Miss Rate Distribution



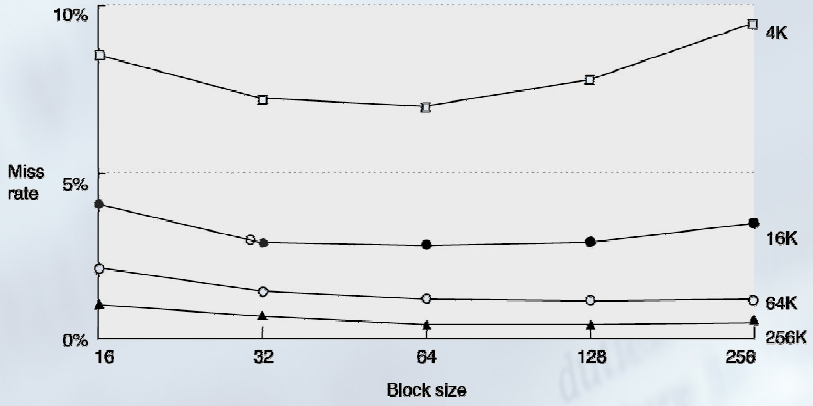
34

First Optimization: Larger Block Size

- Keep cache size & associativity constant
- Reduces compulsory misses
  - Due to spatial locality
  - More accesses are to a pre-fetched block
- Increases capacity misses
  - More unused locations pulled into cache
- May increase conflict misses (slightly)
  - Fewer sets may mean more blocks utilized per set
  - Depends on pattern of addresses accessed
- Increases miss penalty - longer block transfers

35

Block Size Effect



Miss rate actually goes up if the block is too large relative to the cache size

36

Block Size vs. Miss Rate

	Cache Size			
Block Size	**4K**	**16K**	**64K**	**256K**
16	8.57%	3.94%	2.04%	1.09%
32	**7.24%**	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	**9.51%**	3.29%	1.15%	**0.49%**

37

Second Optimization: Larger Caches

- Keep block size, set size, *etc.* constant
- No effect on compulsory misses
  - Block still won't be there on its 1st access
- Reduces capacity misses
  - More capacity
- Reduces conflict misses (in general)
  - Working blocks spread out over more frame sets
  - Fewer blocks map to a set on average
  - Less chance that the number of active blocks that map to a given set exceeds the set size
- But, increases hit time (and cost)

38

Block Size vs. Avg. Access Time

		Cache Size			
Block Size	**Miss Penalty**	**4K**	**16K**	**64K**	**256K**
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

39

Third Optimization : Higher Associativity

- Keep cache size & block size constant
  - Decreasing the number of sets
- No effect on compulsory misses
- No effect on capacity misses
  - By definition, these are misses that would happen anyway in fully-associative
- Decreases conflict misses
  - Blocks in active set may not be evicted early
- Can increase hit time (slightly)
  - Direct-mapped is fastest
  - *n*-way associative lookup a bit slower for larger *n*

40

Performance Comparison

- Assume:

$$Hit\,Time_{1-way} = Cycle\,Time_{1-way} \qquad Miss\,Penalty = 25 \times Cycle\,Time_{1-way}$$

$$Hit\,Time_{2-way} = 1.36 \times Cycle\,Time_{1-way}$$

$$Hit\,Time_{4-way} = 1.44 \times Cycle\,Time_{1-way}$$

$$Hit\,Time_{8-way} = 1.52 \times Cycle\,Time_{1-way}$$

- 4KB, 1-way miss-rate=9.8%; 4-way miss-rate=7.1%

$$Average\,Mem\,Access\,Time_{1-way} = 1.00 + Miss\,Rate \times 25$$
$$= 1.00 + 0.098 \times 25 = 3.45$$

$$Average\,Mem\,Access\,Time_{4-way} = 1.44 + Miss\,Rate \times 25$$
$$= 1.44 + 0.071 \times 25 = 3.215$$

41

Higher Set-Associativity

Cache Size	1-way	2-way	4-way	8-way
4KB	3.44	3.25	3.22	3.28
8KB	2.69	2.58	2.55	2.62
16KB	2.23	2.40	2.46	2.53
32KB	2.06	2.30	2.37	2.45
64KB	1.92	2.14	2.18	2.25
128KB	1.52	1.84	1.92	2.00
256KB	1.32	1.66	1.74	1.82
512KB	1.20	1.55	1.59	1.66

- Higher associativity increases the cycle time
- The table shows the average memory access time
  - 1-way is better in most of the cases

42

4th Optimization: Multi-Level Caches

- What is important?
  - faster caches?
  - or larger caches?
- Average memory access time
  - Hit time (L1) + Miss rate (L1) x Miss Penalty (L1)
- Miss penalty (L1)
  - Hit time (L2) + Miss rate (L2) x Miss Penalty (L2)
- Can plug 2nd equation into the first:
  - Average memory access time =

  Hit time(L1) + Miss rate(L1) x (Hit time(L2) + Miss rate(L2) x Miss penalty(L2))

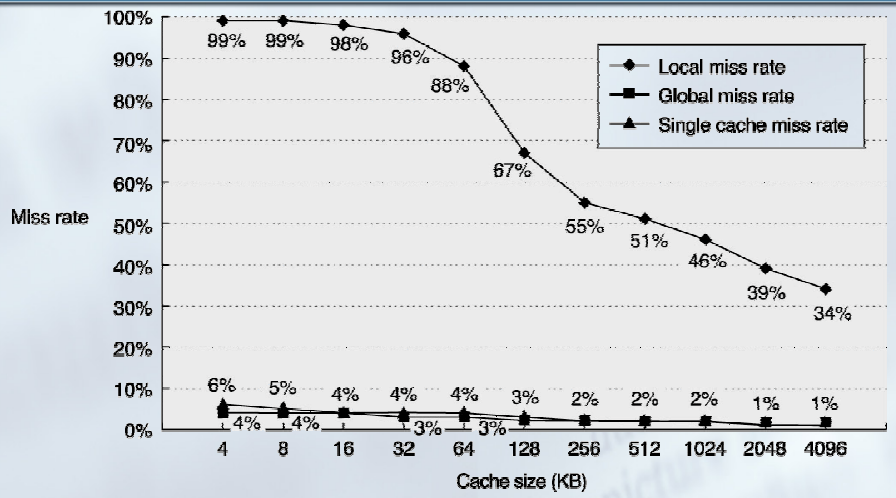
43

Multi-level Cache Terminology

- "Local miss rate"
  - The miss rate of one hierarchy level by itself
  - # of misses at that level / # accesses to that level
  - e.g. Miss rate(L1), Miss rate(L2)
- "Global miss rate"
  - The miss rate of a whole group of hierarchy levels
  - # of accesses coming out of that group (to lower levels) / # of accesses to that group
  - Generally this is the product of the miss rates at each level in the group
  - Global L2 Miss rate = Miss rate(L1) × Local Miss rate(L2)

44

L2 Cache Performance



1. Global cache miss rate is similar to the single cache miss rate
2. Local miss rate is not a good measure of secondary caches

45

Effect of 2-level Caching

- L2 size usually much bigger than L1
  - Provide reasonable hit rate
  - Decreases miss penalty of 1st-level cache
  - May increase L2 miss penalty
- Multiple-level cache inclusion property
  - Inclusive cache: L1 is subset of L2; simplify cache coherence mechanism, effective cache size = L2
  - Exclusive cache: L1, L2 are exclusive; increase effect cache sizes = L1 + L2
  - Enforce inclusion property: Backward invalidation on L2 replacement

46

5th Opt.: Read Misses Take Priority

- Processor must wait on a read, not on a write
  - Miss penalty is higher for reads to begin with and more benefit from reducing read miss penalty
- Write buffer can queue values to be written
  - Until memory bus is not busy with reads
  - Careful about the memory consistency issue
- What if we want to read a block in write buffer?
  - Wait for write, then read block from memory
  - Better: Read block out of write buffer
- Dirty block replacement when reading
  - Write old block, read new block - Delays the read
  - Old block to buffer, read new, write old (better)

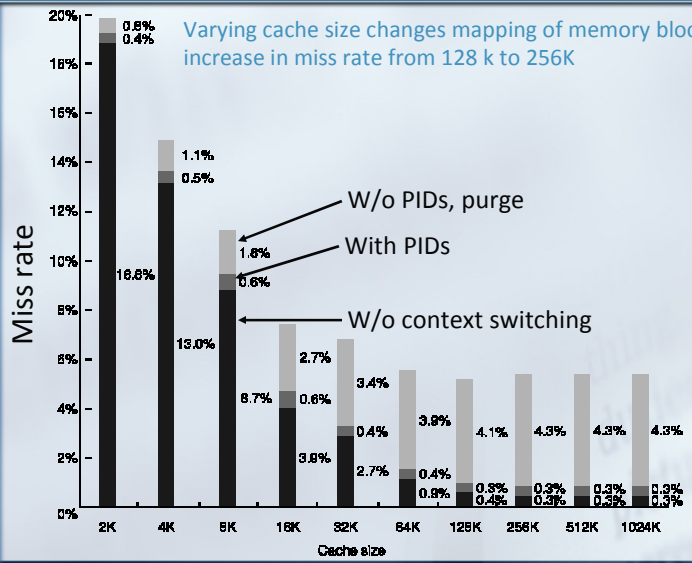
47

6th Opt.: Avoid Address Translation

- In systems with virtual address spaces, virtual addr. must be mapped to physical addresses
- If cache blocks are indexed/tagged w. physical addresses, we must do this translation before we can do the cache lookup. *Long hit time!*
- Solution: Access cache using the *virtual* address. Call this a "Virtual Cache"
  - Drawback: Cache flush on context switch
    - Can fix by tagging blocks with Process Ids (PIDs)
  - Another problem: "Aliasing", i.e. two virtual addresses mapped to same real address
    - Fix with anti-aliasing or page coloring

48

Benefit of PID Tags in Virtual Cache



49

Review of Memory Hierarchy

- Introduction
- Cache Basics
- Cache Performance
- Six Basic Cache Optimizations
- Virtual Memory
- Conclusion

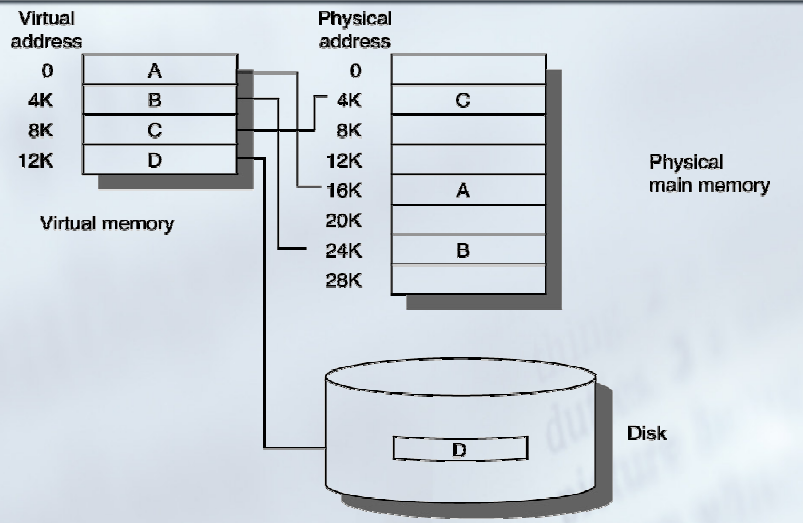
50

Why Virtual Memory?

- Sharing a smaller amount of physical memory
  - Each process uses a small part of address space
  - Needs a protection mechanism
- Virtual memory was invented to automatically manage two levels of memory hierarchy
  - Old days: main memory small and programs - big
  - Programmer: divide the programs into parts (overlays) and load into memory so that the part does not access outside physical main memory
- It also enables relocation

51

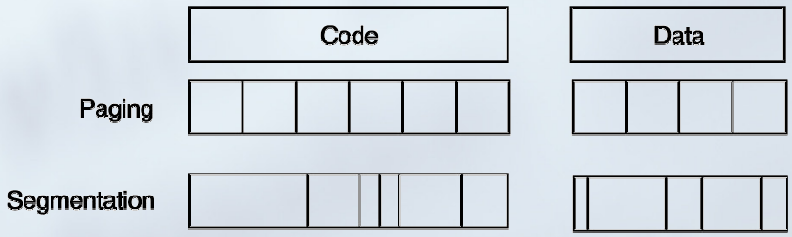
Virtual Memory



The addition of the virtual memory mechanism complicated the cache access

52

Paging vs. Segmentation



- *Paged Segment*: Each segment has integral number of pages for easy replacement and can still treat each segmentation as a unit

53

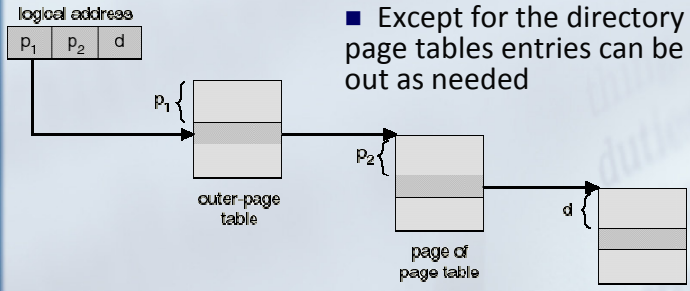
Page Tables and Virtual Memory

- Page tables can be very large
  - (32 - 64 bit logical addresses today)
  - If (only) 32 bits are used (4GB) with 12 bit offset (4KB pages), a page table may have 2^{20} (1M) entries. Every entry will be at least several bytes.
- The entire page table can take up a lot of main memory.
- We may have to use a 2-level (or more) structure for the page table itself.

54

Multilevel Page Tables

- With a 2 level page table (386, Pentium), the page number is split into two numbers p1 and p2
- p1 indexes the outer page table (directory) in main memory whose entries point to a page containing page table entries for some range of virtual memory
- The second level entry is indexed by p2

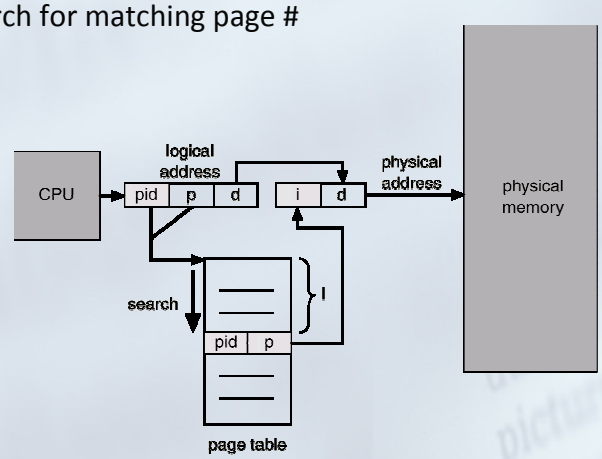


- Except for the directory (outer table), page tables entries can be swapped in and out as needed

55

Inverted Page Table

- One entry per FRAME rather than one per PAGE
- Search for matching page #



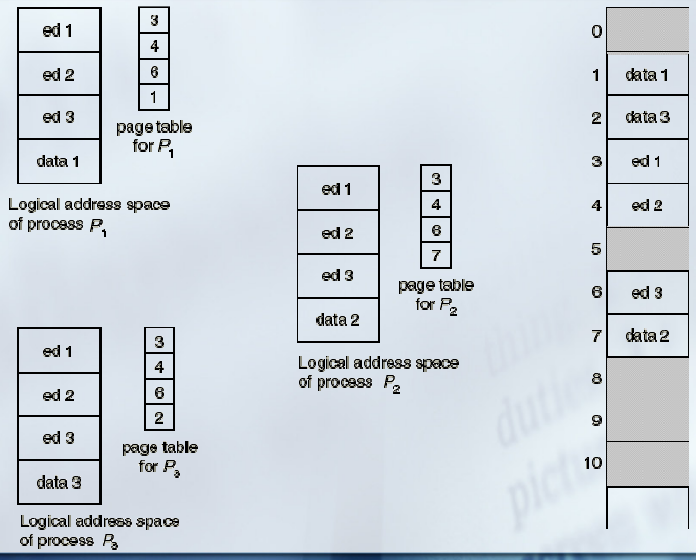
56

Sharing Pages

- Several users can share one copy of the same program
- Shared code must be reentrant (non self-modifying) so that more than one process can execute the same code
- With paging, each process will have a page table with entries pointing to the same code frames
  - Only one copy of each page is actually in a frame in main memory
- Each user also needs to have its own private data pages

57

Sharing Pages: Text Editor



Segmentation

- Processes actually consist of logical parts (segments), such as:
  - one or more executable segments
  - one or more data segments
  - stack segment
- Another idea: make allocation more flexible by loading segments independently

Segmentation

- Divide each program into unequal size blocks called segments
- When a process is loaded into main memory, its individual segments can be located anywhere
- The methods for allocating memory to segments are those we have seen so far: just replace *process* by *segment*
- Because segments are of unequal size, this is similar to *dynamic partitioning* (at the segment level)

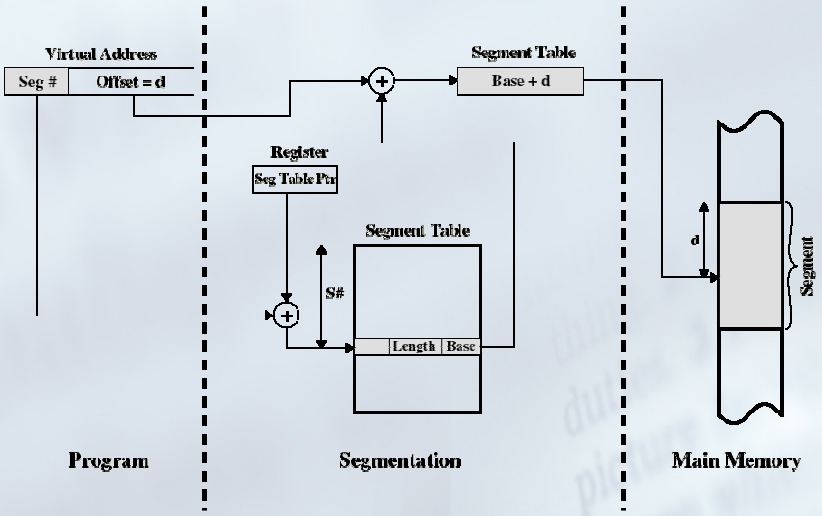
60

Logical address used in simple segmentation with dynamic partitioning

- We need a segment table for each process, containing:
  - the starting physical address of that segment
  - the length of that segment (for protection)
- A CPU register holds the starting address of the the segment table
- Given a logical address (segment, offset) = (s,d), we access the s^{th} entry in the segment table to get base physical address k and the length l of that segment
- The physical address is obtained by adding d to k
  - The hardware also compares the offset d with the length l to determine if the address is valid

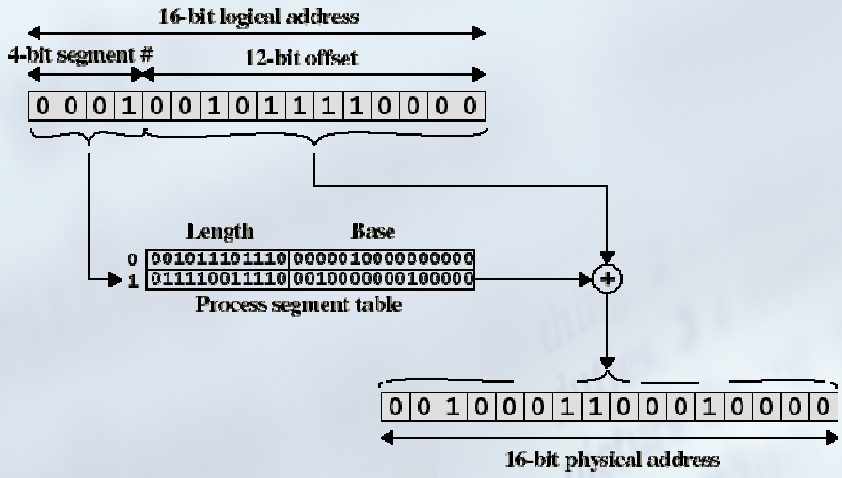
61

Address Translation in Segmentation



62

Logical-to-Physical Address Translation in Segmentation



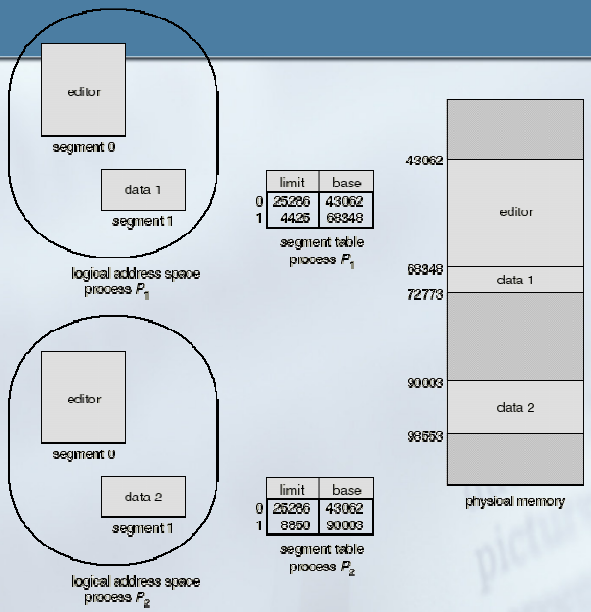
63

Sharing in Segmentation Systems

- The segment tables of 2 different processes can point to the same physical locations
- Example: one shared copy of the the code segment for the text editor
- Each user still needs its own private data segment
  - ➔ more logical than sharing pages

64

Segment Sharing: Text Editor Example



65

Evaluation of Simple Segmentation

- Advantage: memory allocation unit is a logically natural view of program
  - Segments can be loaded individually on demand (*dynamic linking*)
  - Natural unit for protection purposes
  - No internal fragmentation
- Disadvantage: same problems as dynamic partitioning
  - External fragmentation
  - Unlike paging, it is not transparent to programmer
  - No simple relationship between logical and physical address

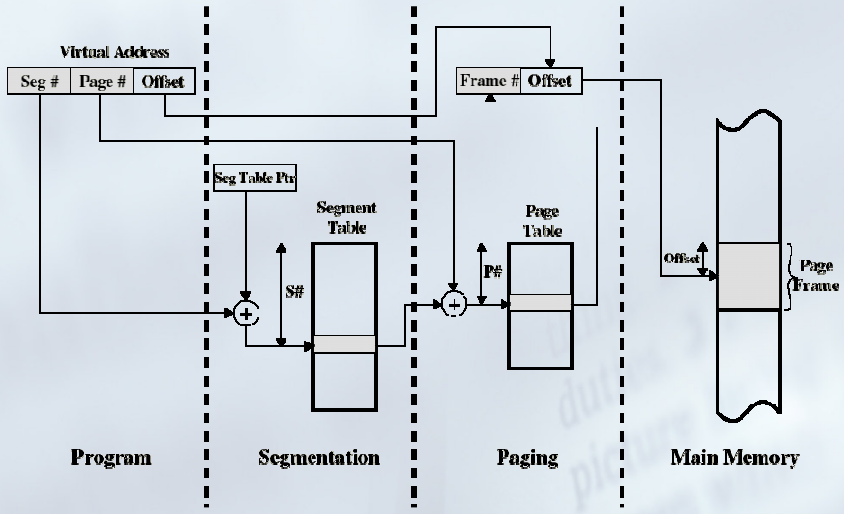
66

Combined Segmentation and Paging

- Combines advantages of both
- Several combinations exist. Example:
  - Each process has:
    - one segment table
    - one page table per segment
  - Virtual address consists of:
    - segment number: index the segment table to get starting address of the page table for that segment
    - page number: index that page table to obtain the physical frame number
    - offset: to locate the word within the frame
- Segment and page tables can themselves be paged

67

Address Translation in Combined Segmentation/Paging System



68

Advantages of Segmentation + Paging

- Solves problems of both loading and linking.
  - Linking a new segment amounts to adding a new entry to a segment table
- Segments can grow without having to be moved in physical memory (just map more pages!)
- Protection and sharing can be done at the 'logical' segment level

69

Paging versus Segmentation

	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Invisible to application programmer	May be visible to application programmer
Memory use inefficiently	Internal fragmentation (unused portion of page)	External fragmentation (unused pieces of main memory)
Efficient disk traffic	Yes - adjust page size to balance access/transfer time	Not always (small segments may transfer just a few bytes)

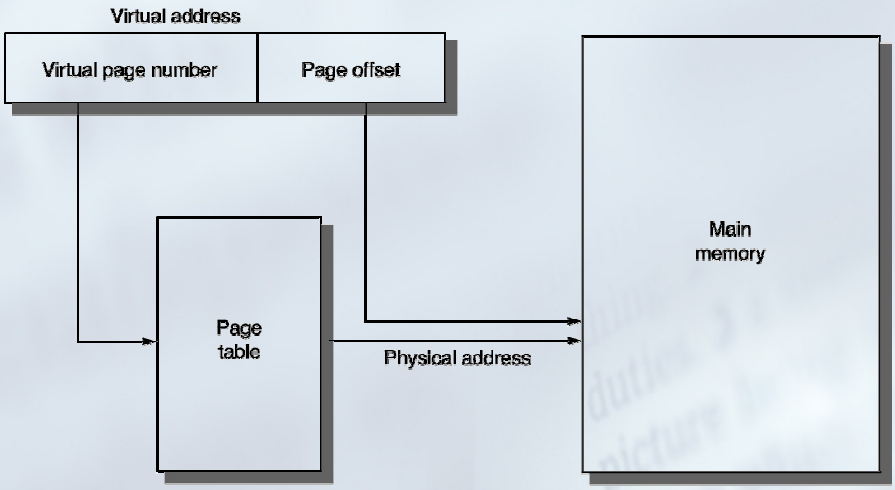
70

Four Important Questions

- Where to place a block in main memory?
  - Operating system takes care of it
  - Replacement takes very long – fully associative
- How to find a block in main memory?
  - Page table is used
    - Offset is concatenated when paging is used
    - Offset is added when segmentation is used
- Which block to replace when needed?
  - Obviously – LRU is used to minimize page faults
- What happens on a write?
  - Magnetic disks takes millions of cycles to access
  - Always write back (use of dirty bit)

71

Addressing Virtual Memories



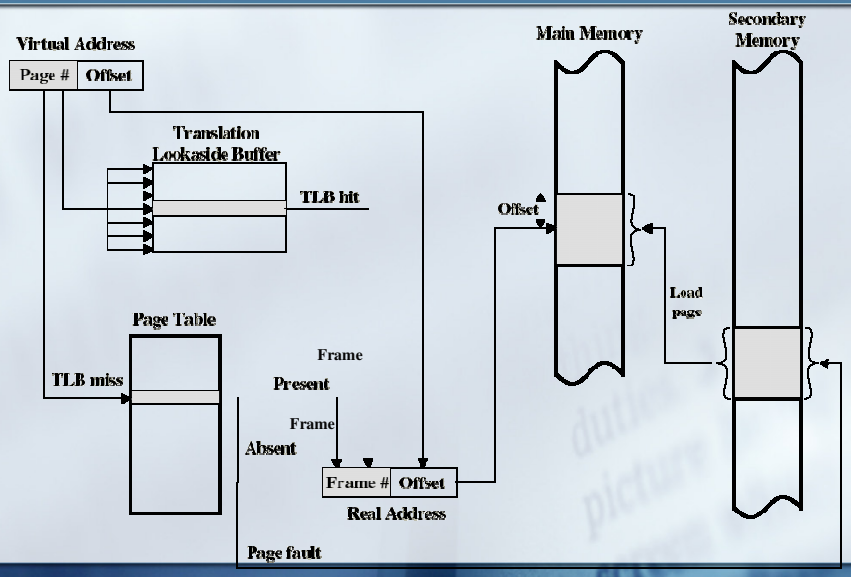
72

Fast Address Calculation

- Page tables are very large
  - Kept in main memory
    - Two memory accesses for one read or write
- Remember the last translation
  - Reuse if the address is on the same page
- Exploit the principle of locality
  - If access have locality, the address translations should also have locality
  - Keep the address translations in a cache
    - Translation look-aside buffer (TLB)
    - The tag part stores the virtual address and the data part stores the page number

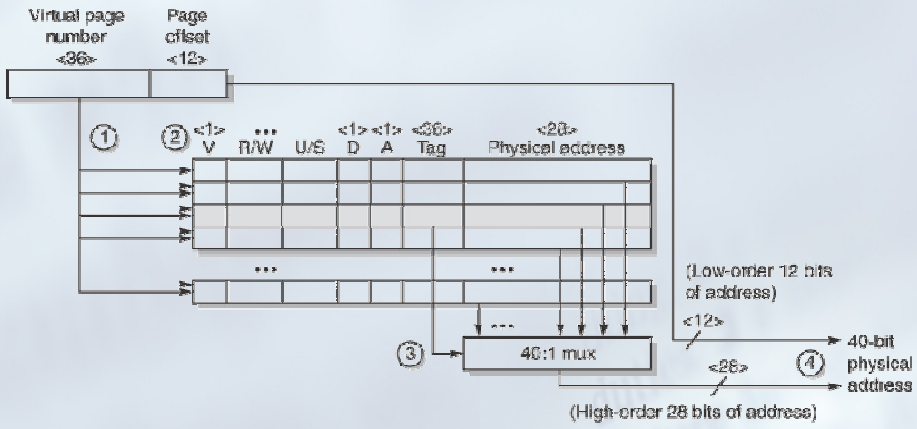
73

Translation Lookaside Buffer



74

TLB Example: AMD Opteron



75

Protection of Virtual Memory

- Maintain two registers
  - Base
  - Bound
- For each address check
  - base <= address <= bound
- Provide two modes
  - User
  - OS (kernel, supervisor, executive)

77

Infosessions

- Hardware virtualization
  - Virtual machine concept
  - Hardware abstraction

81