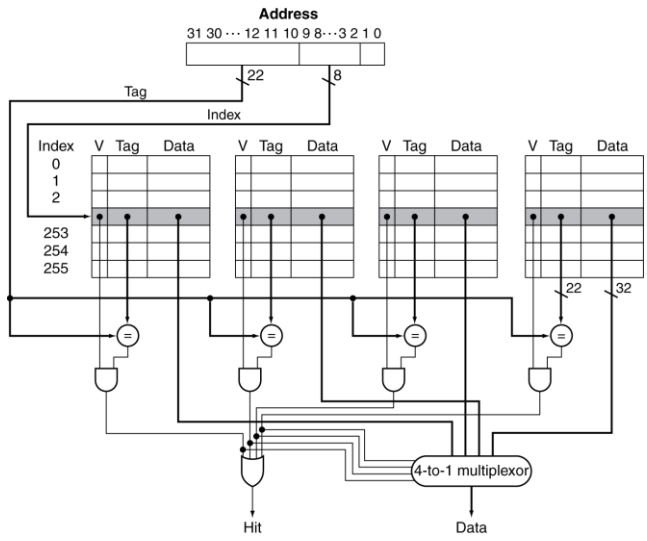
How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Set Associative Cache Organization



Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 500 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

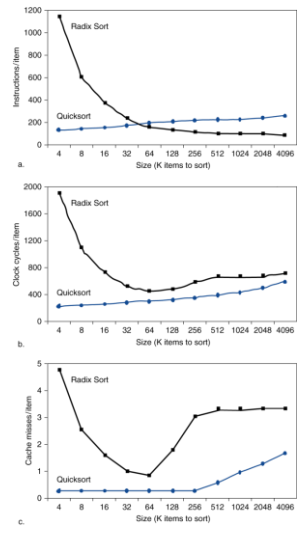
Pop Quiz

- Our cache performance depends on:

- A: CPU instructions
- B: Data access patterns
- C: Compilers
- D: All of the above

Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



Chapter 5 — Large and Fast: Exploiting Memory Hierarchy — 70

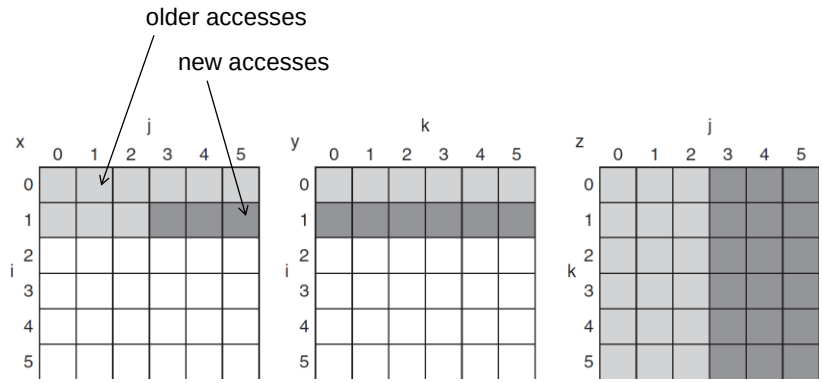
Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

```
for (int j = 0; j < n; ++j)
{
  double cij = C[i+j*n];
  for( int k = 0; k < n; k++ )
    cij += A[i+k*n] * B[k+j*n];
  C[i+j*n] = cij;
}
```


DGEMM Access Pattern

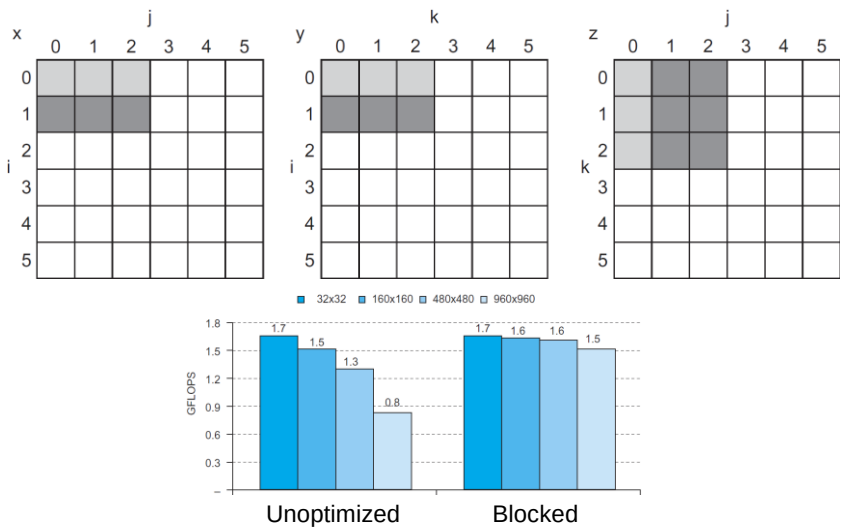
- C, A, and B arrays



Cache Blocked DGEMM

```
1 #define BLOCKSIZE 32
2 void do_block (int n, int si, int sj, int sk, double *A, double
3 *B, double *C)
4 {
5  for (int i = si; i < si+BLOCKSIZE; ++i)
6   for (int j = sj; j < sj+BLOCKSIZE; ++j)
7   {
8    double cij = C[i+j*n];/* cij = C[i][j] */
9    for( int k = sk; k < sk+BLOCKSIZE; k++ )
10    cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
11   C[i+j*n] = cij;/* C[i][j] = cij */
12  }
13 }
14 void dgemm (int n, double* A, double* B, double* C)
15 {
16  for ( int sj = 0; sj < n; sj += BLOCKSIZE )
17   for ( int si = 0; si < n; si += BLOCKSIZE )
18    for ( int sk = 0; sk < n; sk += BLOCKSIZE )
19     do_block(n, si, sj, sk, A, B, C);
20 }
```


Blocked DGEMM Access Pattern



Disk caching

- Use main memory as a *cache* for magnetic disk
- We can do this for a number of reasons:
  - speed up disk access
  - pretend we have more main memory than we really have
  - support multiple programs easily (each can pretend it has all the memory)

Our focus

- We will focus on using the disk as a storage area for chunks of main memory that are not being used
- The basic concepts are similar to providing a cache for main memory, although we now view part of the hard disk as *being the memory*

§5.7 Virtual Memory

Virtual Memory

- Use main memory as a "cache" for secondary (disk) storage
  - Managed jointly by CPU hardware and the operating system (OS)
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses
  - VM "block" is called a page
  - VM translation "miss" is called a page fault

Motivation

- Pretend we have 4GB, we really have only 512MB
- At any time, the processor needs only a small portion of the 4GB memory
  - only a few programs are active
  - an active program might not need all the memory that has been reserved by the program
- We just keep the stuff needed in the main memory, and store the rest on disk

A Program's view of memory

- We can write programs that address the *virtual memory*
- There is hardware that translates these virtual addresses to physical addresses
- The operating system is responsible for managing the movement of memory between disk and main memory, and for keeping the address translation table accurate

Advantages of Virtual Memory

- A program can be written (linked) to use whatever addresses it wants to! It doesn't matter where it is physically loaded!
- When a program is loaded, it doesn't need to be placed in continuous memory locations
  - any group of physical memory *pages* will do fine

Pop Quiz

- Each process on our computer has its own set of virtual memory addresses.

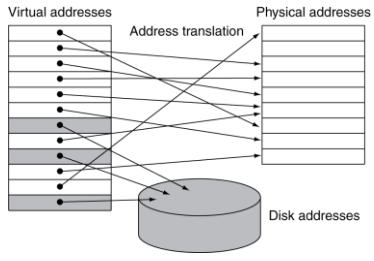
- A: True
- B: False

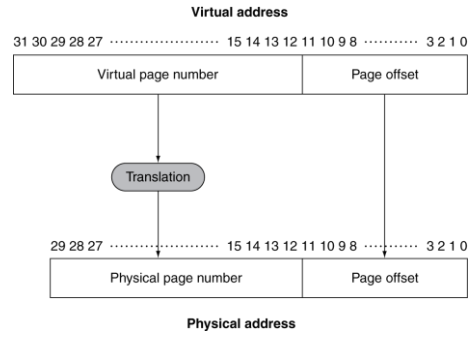
Terminology

- *page*: The unit of memory transferred between disk and the main memory
- *page fault*: when a program accesses a virtual memory location that is not currently in the main memory
- *address translation*: the process of finding the physical address that corresponds to a virtual address

Address Translation

- Fixed-size pages (e.g., 4K)





Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
- Try to minimize page fault rate
  - Fully associative placement
  - Smart replacement algorithms

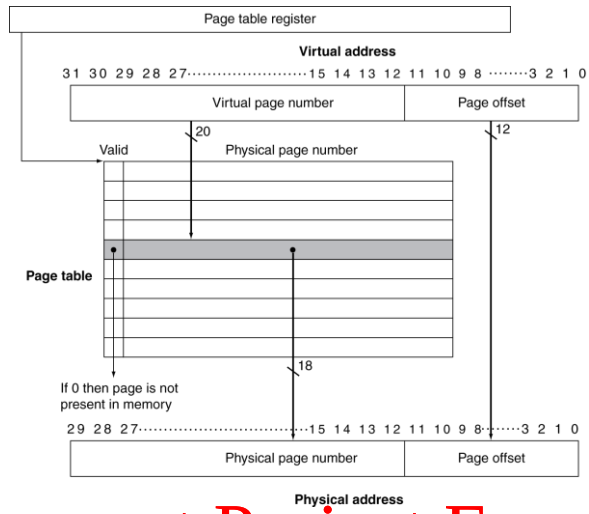
Page Fault Penalty (cont.)

- A Page Fault is a disaster!
  - disk is very, very, very slow compared to memory – millions of cycles!
- Minimization of faults is the primary design consideration for virtual memory systems

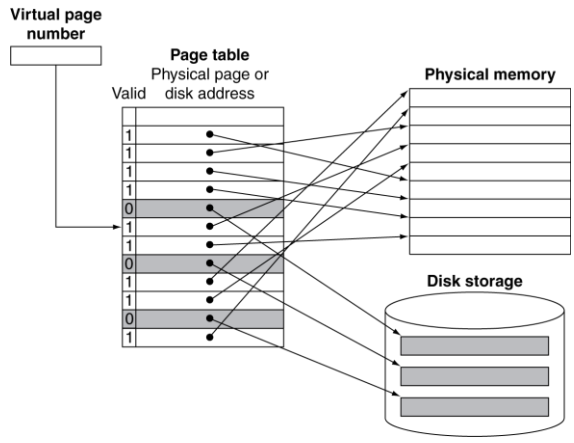
Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present
  - PTE can refer to location in swap space on disk

Translation Using a Page Table



Mapping Pages to Storage



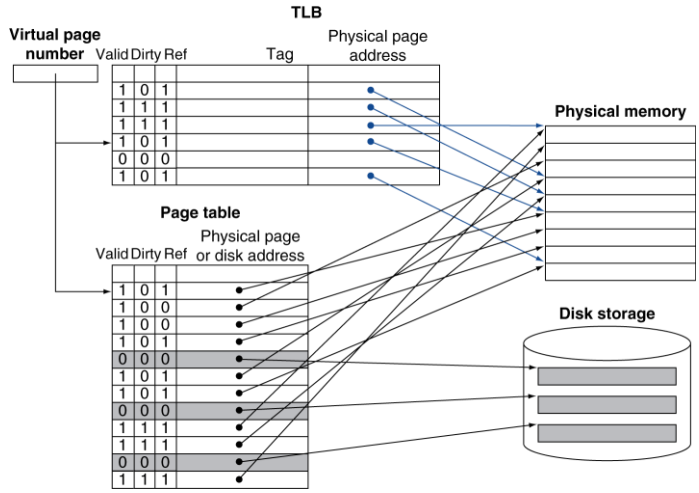
Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

Fast Translation Using a TLB

- Address translation would appear to require extra memory references
  - One to access the PTE
  - Then the actual memory access
- But access to page tables has good locality
  - So use a fast cache of PTEs within the CPU
  - Called a Translation Look-aside Buffer (TLB)
  - Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
  - Misses could be handled by hardware or software

Fast Translation Using a TLB



TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

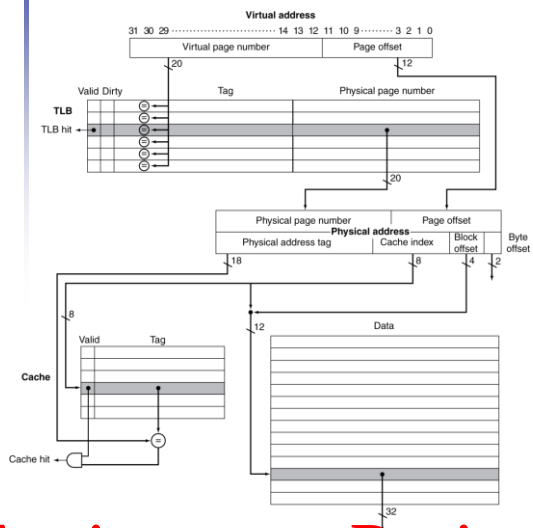
TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not preset
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

TLB and Cache Interaction



- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address

Pop Quiz

- If our cache is using only virtual addresses, after a context switch we must:

- A: Invalidate cached code
- B: Invalidate cached data
- C: Invalidate shared libraries
- D: Invalidate everything

Memory Protection

- Different tasks can share parts of their virtual address spaces
  - But need to protect against errant access
  - Requires OS assistance
- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)

Possible TLB Outcomes

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.