16.482 / 16.561
Computer Architecture and Design

Instructor: Dr. Michael Geiger

Spring 2014

Lecture 9:

Virtual memory

Cache optimizations

Lecture outline

- Announcements/reminders
  - HW 6 due today
  - HW 7 to be posted; due 4/24
- Review
  - Memory hierarchy design
- Today's lecture
  - Virtual memory
  - Cache optimizations

Review: memory hierarchies

- We want a large, fast, low-cost memory
  - Can’t get that with a single memory
- Solution: use a little bit of everything!
  - Small SRAM array → *cache*
    - Small means fast and cheap
    - More available die area → multiple cache levels on chip
  - Larger DRAM array → *main memory*
    - Hope you rarely have to use it
  - Extremely large *hard disk*
    - Costs are decreasing at a faster rate than we fill them

Review: Cache operation & terminology

- Accessing data (and instructions!)
  - Check the top level of the hierarchy
    - If data is present, **hit**, if not, **miss**
    - On a miss, check the next lowest level
      - With 1 cache level, you check main memory, then disk
      - With multiple levels, check L2, then L3
- Average memory access time gives overall view of memory performance

  AMAT = (hit time) + (miss rate) x (miss penalty)

  - Miss penalty = AMAT for next level
- Caches work because of locality
  - Spatial vs. temporal

Review: 4 Questions for Hierarchy

- Q1: Where can a block be placed in the upper level?
  (Block placement)
  - *Fully associative, set associative, direct-mapped*
- Q2: How is a block found if it is in the upper level?
  (Block identification)
  - *Check the tag—size determined by other address fields*
- Q3: Which block should be replaced on a miss?
  (Block replacement)
  - *Typically use least-recently used (LRU) replacement*
- Q4: What happens on a write?
  (Write strategy)
  - *Write-through vs. write-back*

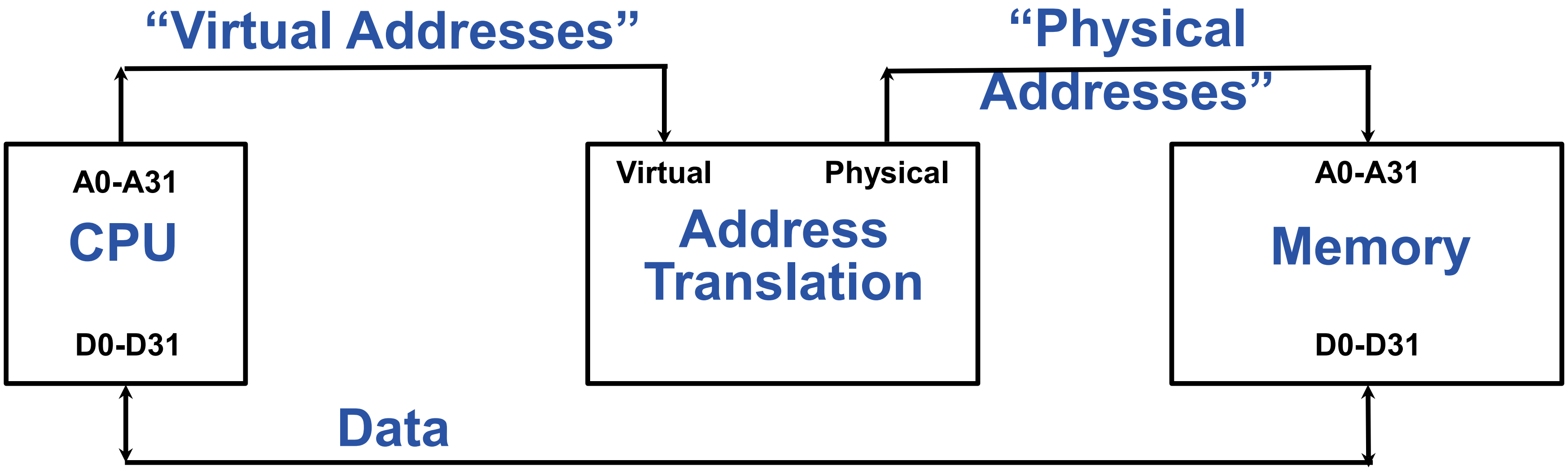
Problems with memory

- DRAM is too expensive to buy many gigabytes
  - We need our programs to work even if they require more memory than we have
  - A program that works on a machine with 512 MB should still work on a machine with 256 MB
- Most systems run multiple programs

Solutions

- Leave the problem up to the programmer
  - Assume programmer knows exact configuration
- Overlays
  - Compiler identifies mutually exclusive regions
- **Virtual memory**
  - Use hardware and software to automatically translate references from **virtual address** (what the programmer sees) to **physical address** (index to DRAM or disk)

Benefits of virtual memory



User programs run in a standardized virtual address space

Address Translation hardware managed by the operating system (OS) maps virtual address to physical memory

Hardware supports "modern" OS features: Protection, Translation, Sharing

4 Questions for Virtual Memory

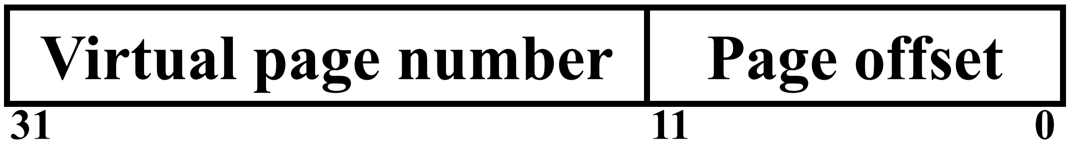
- Reconsider these questions for virtual memory
  - Q1: Where can a page be placed in main memory?
  - Q2: How is a page found if it is in main memory?
  - Q3: Which page should be replaced on a page fault?
  - Q4: What happens on a write?

4 Questions for Virtual Memory (cont.)

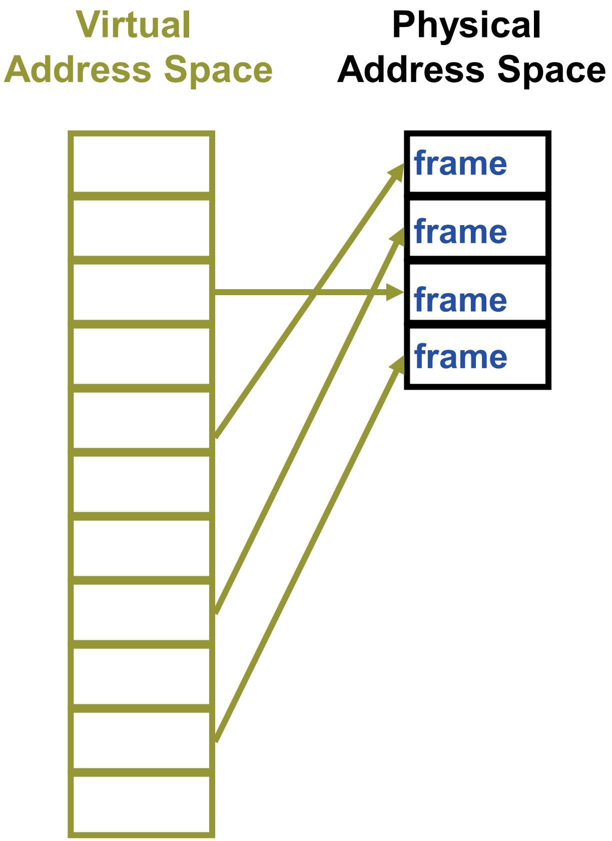
- Q1: Where can a page be placed in main memory?
  - Disk very slow → lowest MR→ fully associative
  - OS maintains list of free frames
- Q2: How is a page found in main memory?
  - Page table contains mapping from virtual address (VA) to physical address (PA)
    - Page table stored in memory
    - Indexed by page number (upper bits of virtual address)
    - Note: PA usually smaller than VA
      - Less physical memory available than virtual memory

Managing virtual memory

- Effectively treat main memory as a cache
  - Blocks are called **pages**
  - Misses are called **page faults**
- Virtual address consists of **virtual page number** and **page offset**

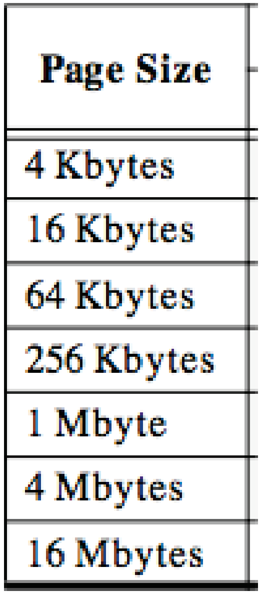


Page tables encode virtual address spaces



A virtual address space
is divided into blocks
of memory called pages

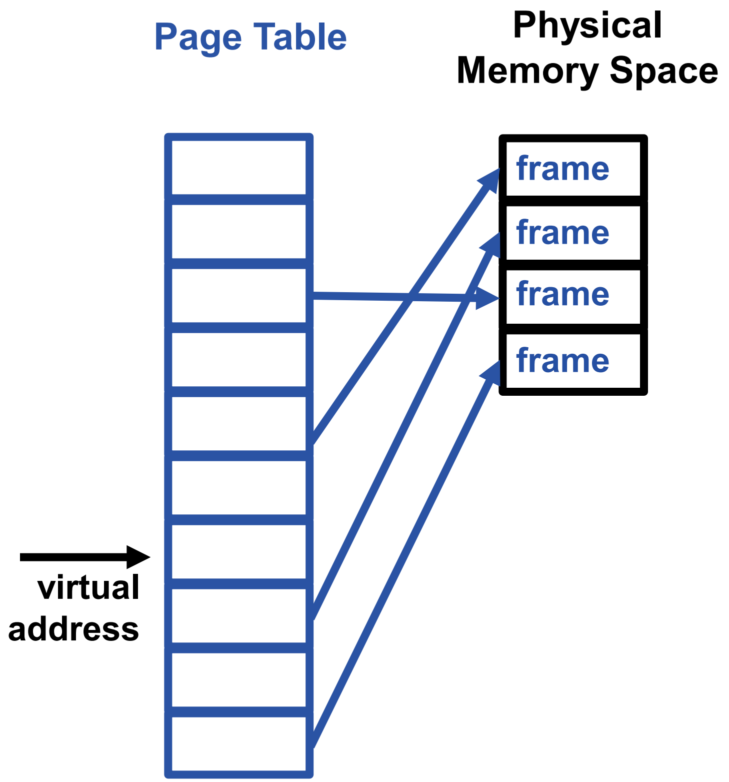
A machine usually supports pages of a few sizes (MIPS R4000):



Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

A valid page table entry codes physical memory "frame" address for the page

Page tables encode virtual address spaces



A virtual address space is divided into blocks of memory called pages

A machine usually supports pages of a few sizes (MIPS R4000):

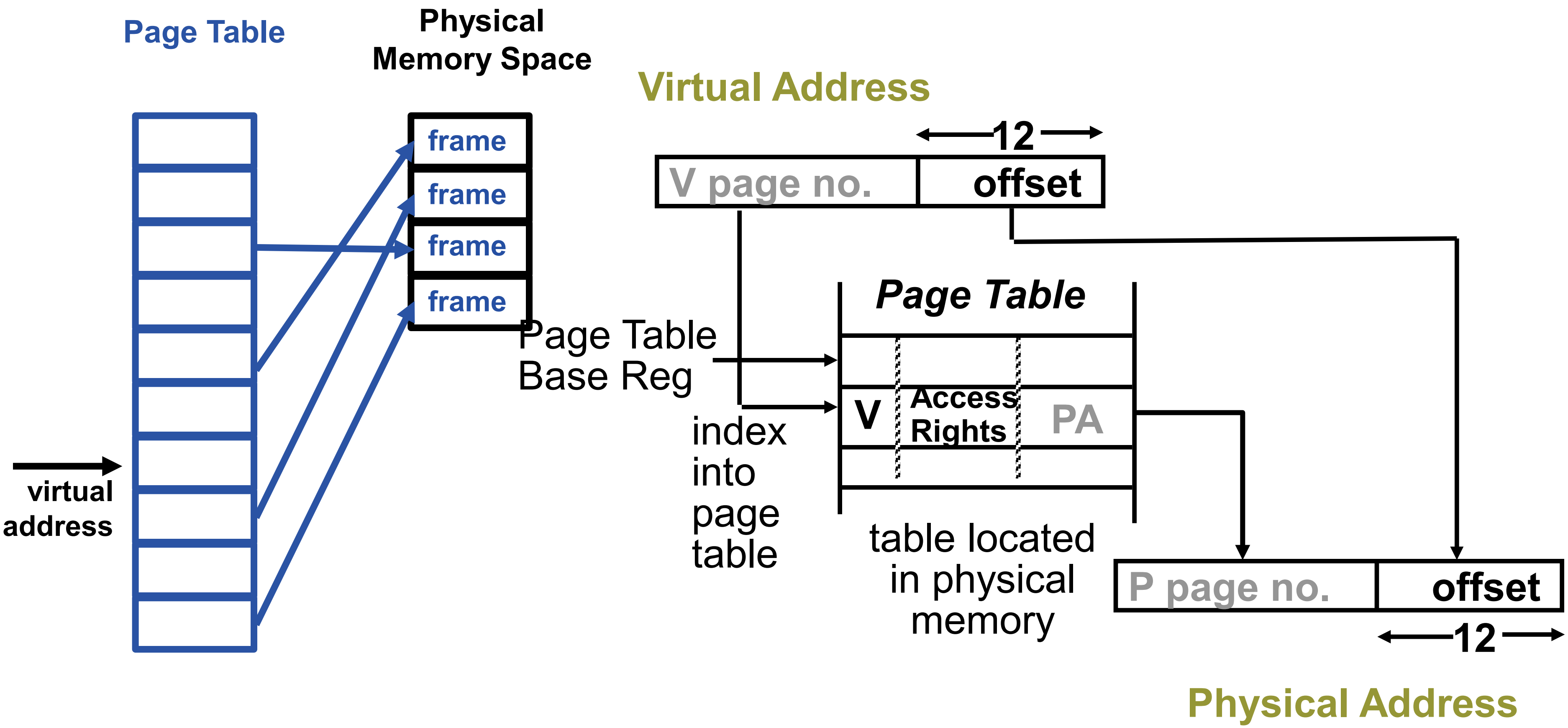
Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

OS manages the page table for each ASID

A page table is indexed by a virtual address

A valid page table entry codes physical memory "frame" address for the page

Details of Page Table



- Page table maps virtual page numbers to physical frames ("PTE" = Page Table Entry)
- Virtual memory => treat memory ≈ cache for disk

Virtual memory example

- Assume the current process uses the page table below:

Virtual page #	Valid bit	Reference bit	Dirty bit	Frame #
0	1	1	0	4
1	1	1	1	7
2	0	0	0	--
3	1	0	0	2
4	0	0	0	--
5	1	0	1	0

- Which virtual pages are present in physical memory?
- Assuming 1 KB pages and 16-bit addresses, what physical addresses would the virtual addresses below map to?
  - 0x041C
  - 0x08AD
  - 0x157B

Virtual memory example soln.

- Which virtual pages are present in physical memory?
  - All those with valid PTEs: 0, 1, 3, 5
- Assuming 1 KB pages and 16-bit addresses (both VA & PA), what PA, if any, would the VA below map to?
  - 1 KB pages → 10-bit page offset (unchanged in PA)
  - Remaining bits: virtual page # → upper 6 bits
    - Virtual page # chooses PTE; frame # used in PA
  - 0x041C = $0000\ 0100\ 0001\ 1100_2$
    - Upper 6 bits = 0000 01 = 1
    - PTE 1 → frame # 7 = 000111
    - PA = $0001\ 1100\ 0001\ 1100_2$ = 0x1C1C
  - 0x08AD = $0000\ 1000\ 1010\ 1101_2$
    - Upper 6 bits = 0000 10 = 2
    - PTE 2 is not valid → page fault
  - 0x157B = $0001\ 0101\ 0111\ 1011_2$
    - Upper 6 bits = 0001 01 = 5
    - PTE 5 → frame # 0 = 000000
    - PA = $0000\ 0001\ 0111\ 1011_2$ = 0x017B

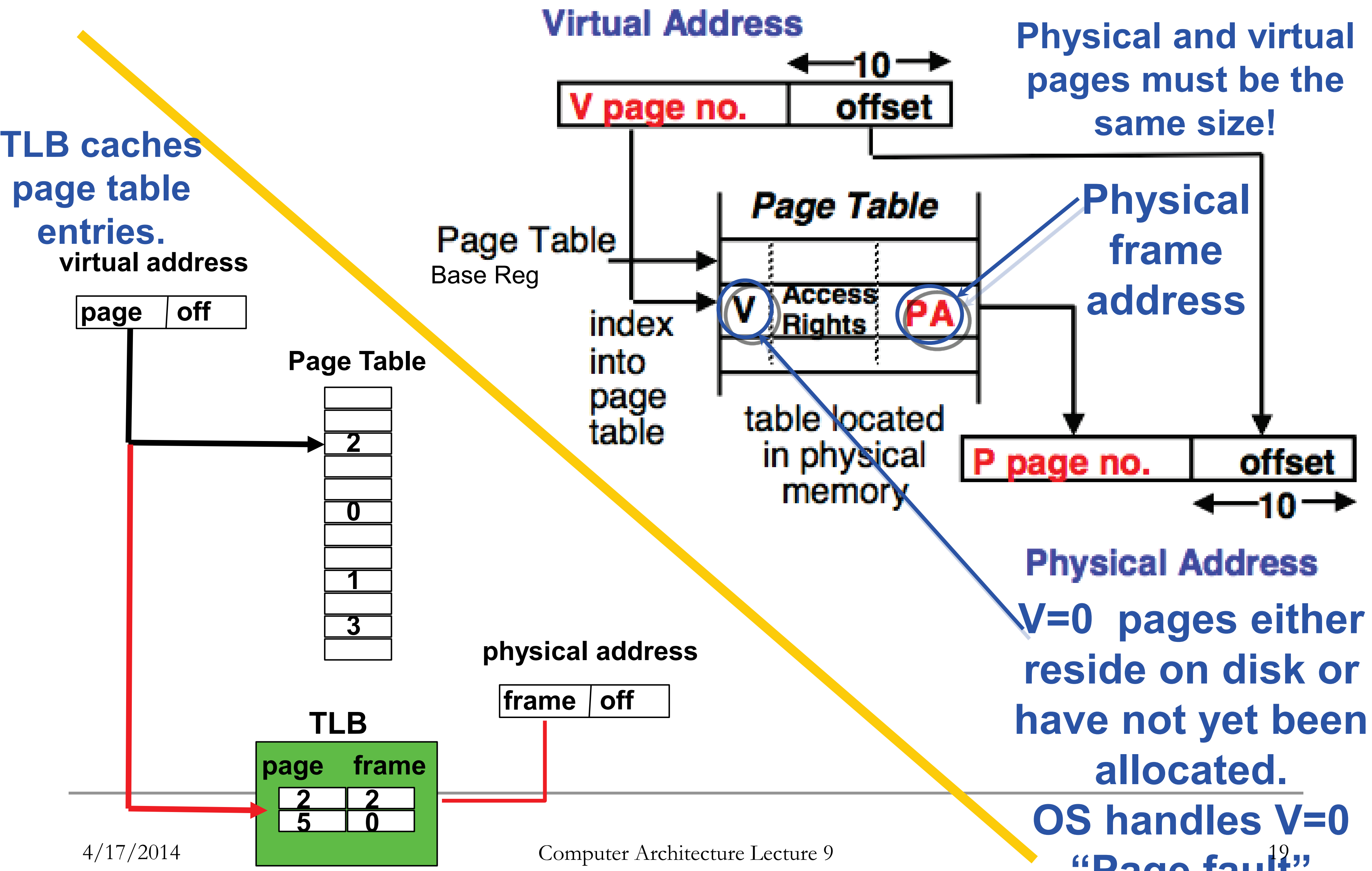
4 Questions for Virtual Memory (cont.)

- Q3: Which page should be replaced on a page fault?
  - Once again, LRU ideal but hard to track
  - Virtual memory solution: reference bits
    - Set bit every time page is referenced
    - Clear all reference bits on regular interval
    - Evict non-referenced page when necessary
- Q4: What happens on a write?
  - Slow disk → write-through makes no sense
  - PTE contains dirty bit

Virtual memory performance

- Address translation accesses memory to get PTE → every memory access twice as long
- Solution: store recently used translations
  - Translation lookaside buffer (TLB): a cache for page table entries
    - "Tag" is the virtual page #
    - TLB small → often fully associative
    - TLB entry also contains valid bit (for that translation); reference & dirty bits (for the page itself!)

The TLB caches page table entries



TLB caches page table entries.

Physical and virtual pages must be the same size!

Physical frame address

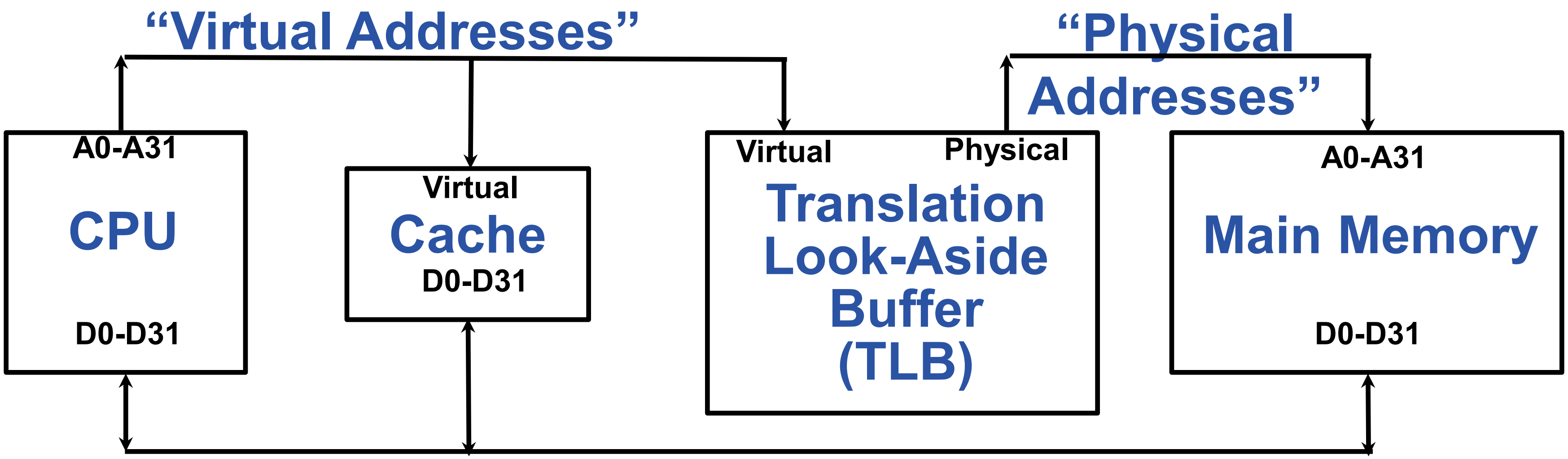
V=0 pages either reside on disk or have not yet been allocated.

OS handles V=0 “Page fault”

Caches and virtual memory

- Using two different addresses: virtual and physical
  - Which should we use to access cache?
  - Physical address
    - Pros: simpler to manage
    - Cons: slower access
  - Virtual address
    - Pros: faster access
    - Cons: aliasing, difficult management
  - Use both: virtually indexed, physically tagged

Use virtual addresses for cache?



Only use TLB on a cache miss !

Downside: a subtle, fatal problem. What is it?

A. Synonym problem. If two address spaces share a physical frame, data may be in cache twice. Maintaining consistency is a nightmare.

Back to caches …

- Reduce misses → improve performance
- Reasons for misses: “the three C’s”
    - First reference to an address: **Compulsory** miss
        - Increasing the block size
    - Cache is too small to hold data: **Capacity** miss
        - Increase the cache size
    - Replaced from a busy line or set: **Conflict** miss
        - Increase associativity
        - Would have had hit in a fully associative cache

Advanced Cache Optimizations

Reducing hit time

1. Way prediction
2. Trace caches

Increasing cache bandwidth

3. Pipelined caches
4. Multibanked caches
5. Nonblocking caches

Reducing miss penalty

6. Critical word first
7. Merging write buffers

Reducing miss rate

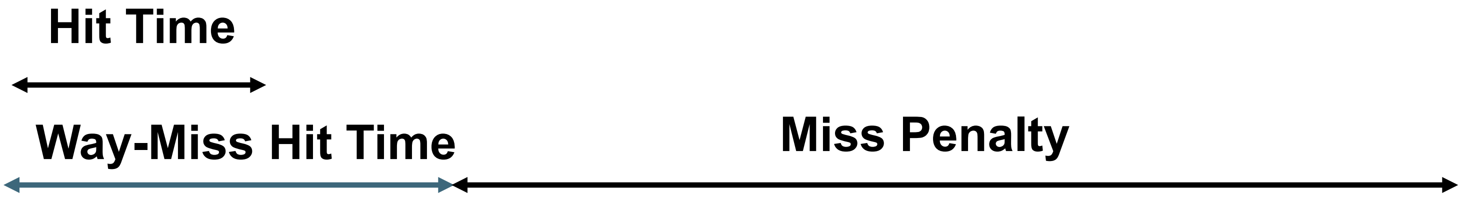
8. Compiler optimizations

Reducing miss penalty or miss rate via parallelism

9. Hardware prefetching
10. Compiler prefetching

Fast Hit times via Way Prediction

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Way prediction: keep extra bits in cache to predict the "way," or block within the set, of next cache access.
  - Multiplexor is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data
  - Miss ⇒ 1st check other blocks for matches in next clock cycle



Hit Time

Way-Miss Hit Time

Miss Penalty

- Accuracy ≈ 85%
- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Used for instruction caches vs. data caches

Fast Hit times via Trace Cache

- Find more instruction level parallelism?
  How avoid translation from x86 to microops?
- Trace cache in Pentium 4 *(only, possibly last, processor to use)*

1. Dynamic traces of the executed instructions vs. static sequences of instructions as determined by layout in memory
   - Built-in branch predictor
2. Cache the micro-ops vs. x86 instructions
   - Decode/translate from x86 to micro-ops on trace cache miss

+ ⇒ better utilize long blocks (don't exit in middle of block, don't enter at label in middle of block)

- ⇒ complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size

- ⇒ instructions may appear multiple times in multiple dynamic traces due to different branch outcomes

Increasing Cache Bandwidth by Pipelining

- Pipeline cache access to maintain bandwidth, but higher latency
- Instruction cache access pipeline stages:

  1: Pentium

  2: Pentium Pro through Pentium III

  4: Pentium 4

- ⇒ greater penalty on mispredicted branches
- ⇒ more clock cycles between the issue of the load and the use of the data

Increasing Cache Bandwidth: Non-Blocking Caches

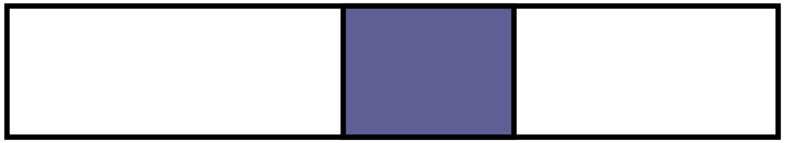
- *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
  - requires F/E bits on registers or out-of-order execution
  - requires multi-bank memories
- "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires multiple memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses

Increasing Bandwidth w/Multiple Banks

- Rather than treat the cache as a single monolithic block, divide into independent banks that can support simultaneous accesses
  - E.g.,T1 (“Niagara”) L2 has 4 banks
- Banking works best when accesses naturally spread themselves across banks ⇒ mapping of addresses to banks affects behavior of memory system
- Simple mapping that works well is “sequential interleaving”
  - Spread block addresses sequentially across banks
  - E,g, if there 4 banks, Bank 0 has all blocks whose address modulo 4 is 0; bank 1 has all blocks whose address modulo 4 is 1; …

Reduce Miss Penalty: Early Restart and Critical Word First

- Don’t wait for full block before restarting CPU
- *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - **Spatial locality ⇒ tend to want next sequential word, so not clear size of benefit of just early restart**
- *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block
  - Long blocks more popular today ⇒ Critical Word 1st Widely used



block

Merging Write Buffer to Reduce Miss Penalty

- Write buffer to allow processor to continue while waiting to write to memory
- If buffer contains modified blocks, the addresses can be checked to see if address of new data matches the address of a valid write buffer entry
- If so, new data are combined with that entry
- Increases block size of write for write-through cache of writes to sequential words, bytes since multiword writes more efficient to memory
- The Sun T1 (Niagara) processor, among many others, uses write merging

Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced caches misses by 75%
  on 8KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts(using tools they developed)
- Data
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows

Merging Arrays Example

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key; improve spatial locality

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
          x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
          x[i][j] = 2 * x[i][j];
```

Sequential accesses instead of striding through memory every 100 words; improved spatial locality

Loop Fusion Example

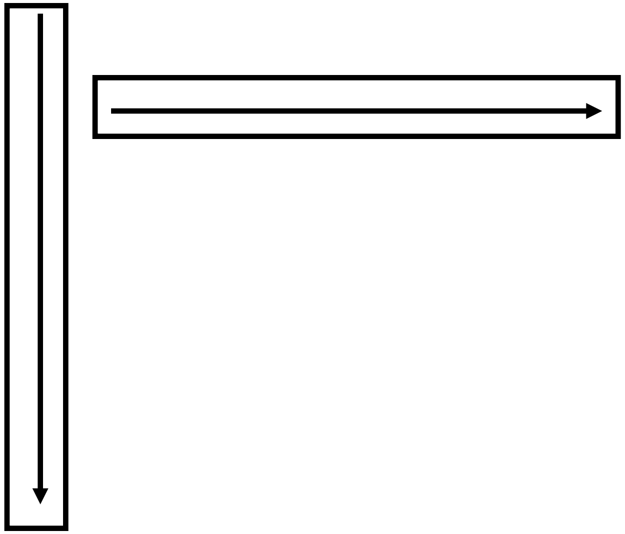
```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
         a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
         d[i][j] = a[i][j] + c[i][j];

/* After */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {     a[i][j] = 1/b[i][j] * c[i][j];
         d[i][j] = a[i][j] + c[i][j];}
```

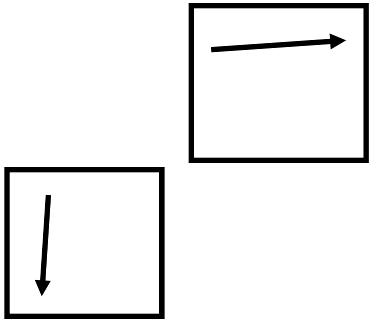
- 2 misses per access to `a` & `c` vs. one miss per access; improve spatial locality

Blocking Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     {r = 0;
      for (k = 0; k < N; k = k+1){
        r = r + y[i][k]*z[k][j];};
        x[i][j] = r;
     };
```



- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity Misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits

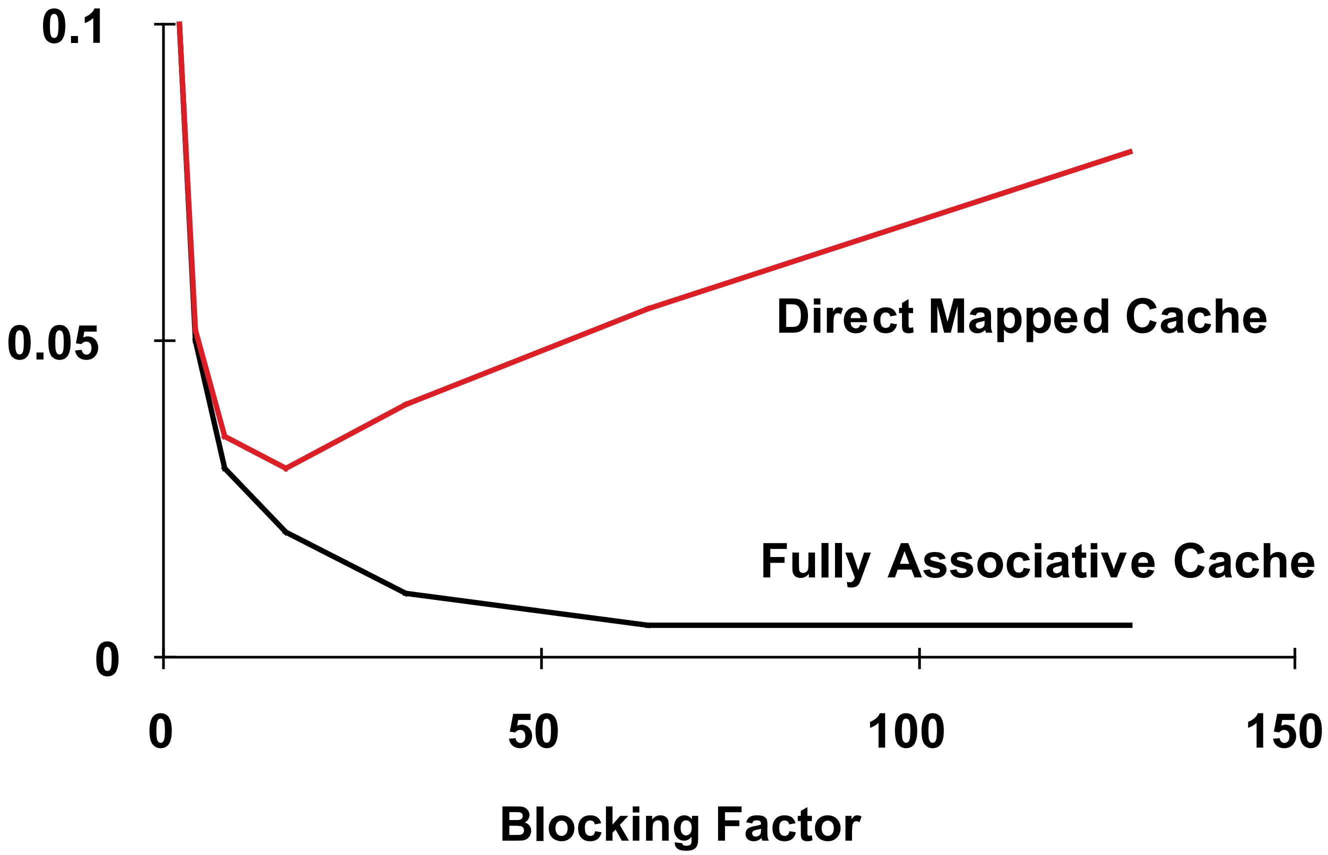


Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
    {r = 0;
     for (k = kk; k < min(kk+B-1,N); k = k+1) {
       r = r + y[i][k]*z[k][j];};
     x[i][j] = x[i][j] + r;
    };
```

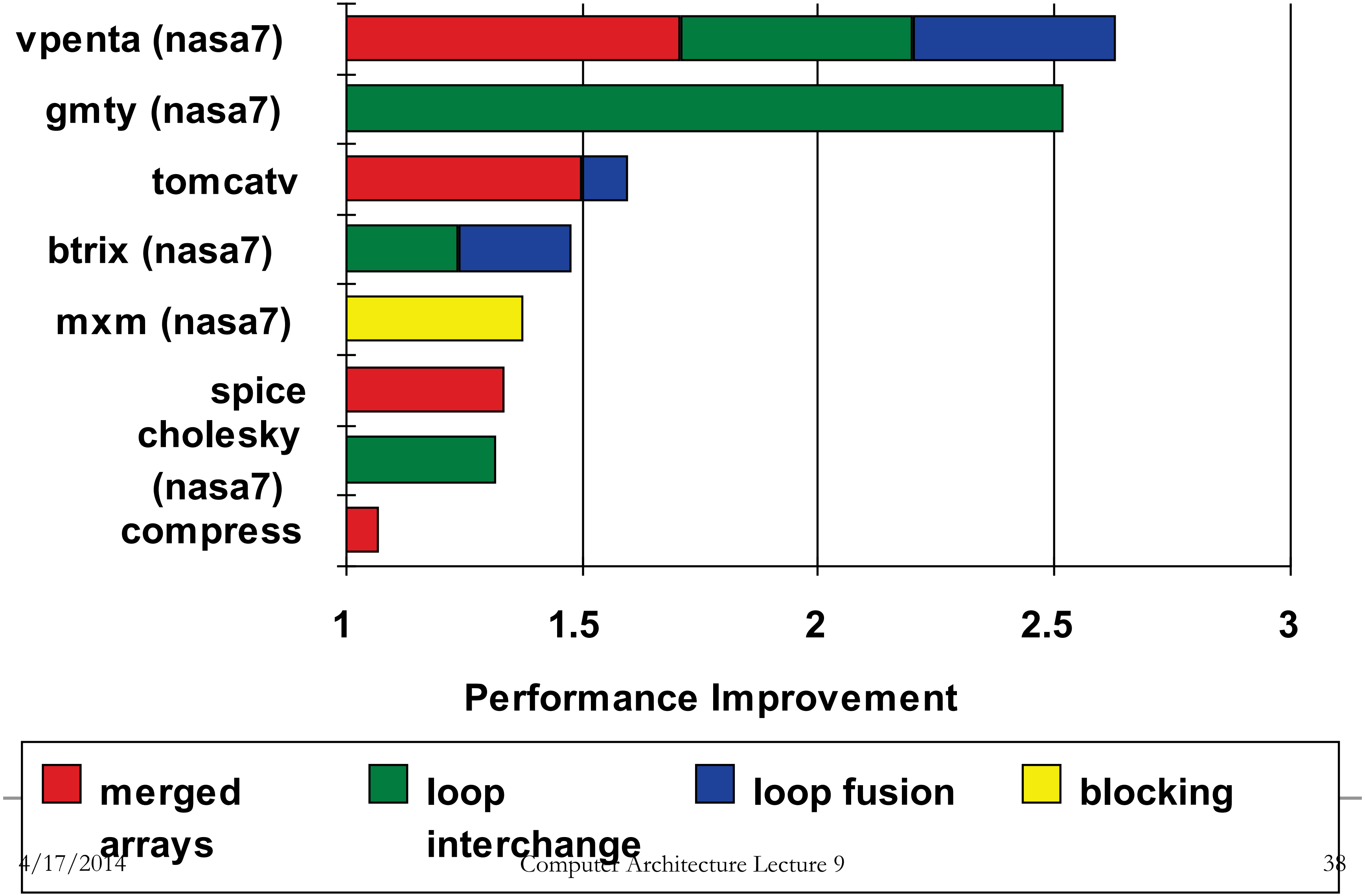
- B called *Blocking Factor*
- Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
- Conflict Misses Too?

Reducing Conflict Misses by Blocking



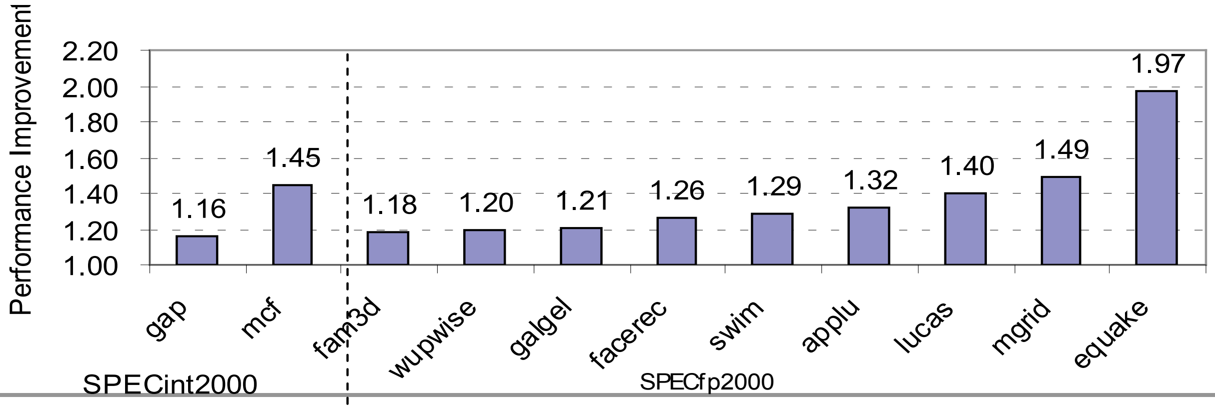
- Conflict misses in caches not FA vs. Blocking size
  - Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache

Summary of Compiler Optimizations to Reduce Cache Misses (by hand)



Reducing Misses by Hardware Prefetching of Instructions & Data

- Prefetching relies on having extra memory bandwidth that can be used without penalty
- Instruction Prefetching
  - Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.
  - Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer
- Data Prefetching
  - Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages
  - Prefetching invoked if 2 successive L2 cache misses to a page,
    if distance between those cache blocks is < 256 bytes



Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Load data into register (HP PA-RISC loads)
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  - Special prefetching instructions cannot cause faults; a form of speculative execution
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

Final notes

- Next time
  - Storage
  - Multiprocessors (primarily memory)
  - Final exam preview (exam is in class 5/1)
- Reminders
  - HW 6 due today
  - HW 7 to be posted; due 4/24