IC220 Set #17:
Caching Finale and Virtual Reality
(Chapter 5)

ADMIN

- Reading – finish Chapter 5
  - Sections 5.4 (skip 511-515), 5.5, 5.11, 5.12

Cache Performance

- Simplified model:

execution time = (execution cycles + stall cycles) × cycle time
= execTime + stallTime

$$\text{stall cycles} = \frac{MemoryAccesses}{Program} \bullet MissRate \bullet MissPenalty$$

$$\text{(or)} = \frac{Instructions}{Program} \bullet \frac{Misses}{Instruction} \bullet MissPenalty$$

- Two typical ways of improving performance:
  - decreasing the miss rate
  - decreasing the miss penalty

What happens if we increase block size?

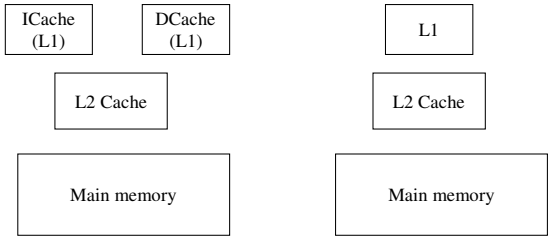
Add associativity?

Performance Example

- Suppose processor has a CPI of 1.5 given a perfect cache. If there are 1.2 memory accesses per instruction, a miss penalty of 20 cycles, and a miss rate of 10%, what is the effective CPI with the real cache?

Split Caches

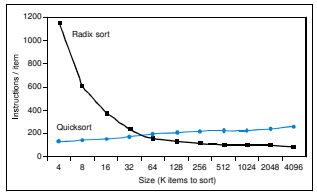
- **Instructions and data have different properties**
  - **May benefit from different cache organizations (block size, assoc…)**



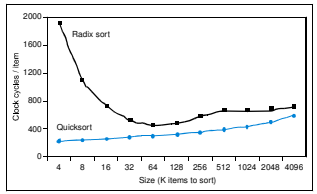
- **Why else might we want to do this?**

Cache Complexities

- **Not always easy to understand implications of caches:**



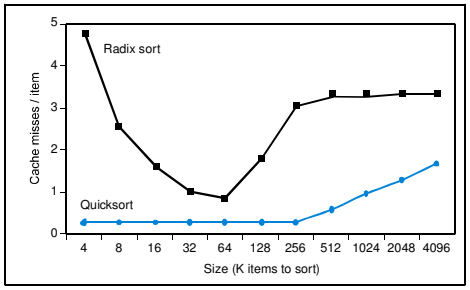
Theoretical behavior of Radix sort vs. Quicksort



Observed behavior of Radix sort vs. Quicksort

Cache Complexities

- **Here is why:**



- **Memory system performance is often critical factor**
  - **multilevel caches, pipelined processors, make it harder to predict outcomes**
  - **Compiler optimizations to increase locality sometimes hurt ILP**
- **Difficult to predict best algorithm: need experimental data**

Program Design for Caches – Example 1

- **Option #1**

```
for (j = 0; j < 20; j++)
  for (i = 0; i < 200; i++)
    x[i][j] = x[i][j] + 1;
```

- **Option #2**

```
for (i = 0; i < 200; i++)
  for (j = 0; j < 20; j++)
    x[i][j] = x[i][j] + 1;
```

Program Design for Caches – Example 2

- **Why might this code be problematic?**

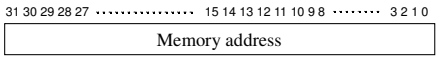
```
int  A[1024][1024];
int  B[1024][1024];
for (i = 0; i < 1024; i++)
  for (j = 0; j < 1024; j++)
   A[i][j] += B[i][j];
```

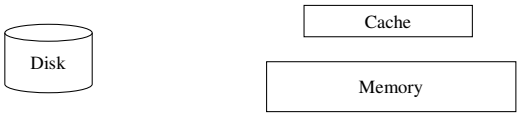
- **How to fix it?**

VIRTUAL MEMORY

Virtual memory summary (part 1)

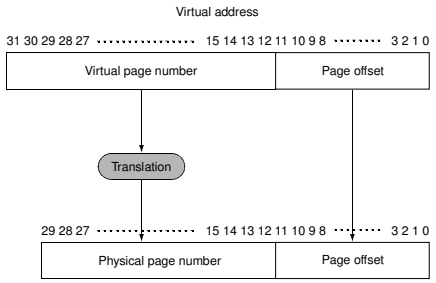
Data access <u>without</u> virtual memory:

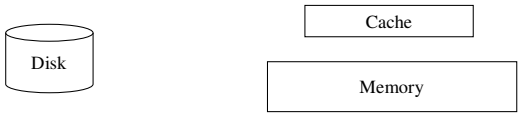




Virtual memory summary (part 2)

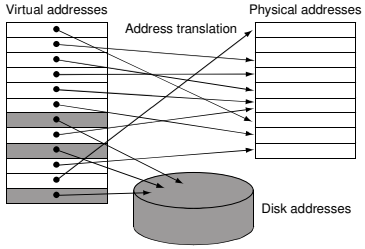
Data access <u>with</u> virtual memory:





Virtual Memory

- **Main memory can act as a cache for the secondary storage (disk)**

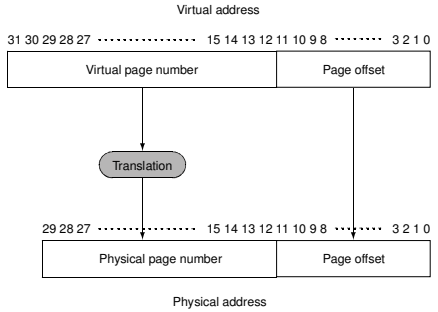


- **Advantages:**
  - **Illusion of having more physical memory**
  - **Program relocation**
  - **Protection**
- **Note that main point is caching of disk in main memory but will affect all our memory references!**

Address Translation

Terminology:

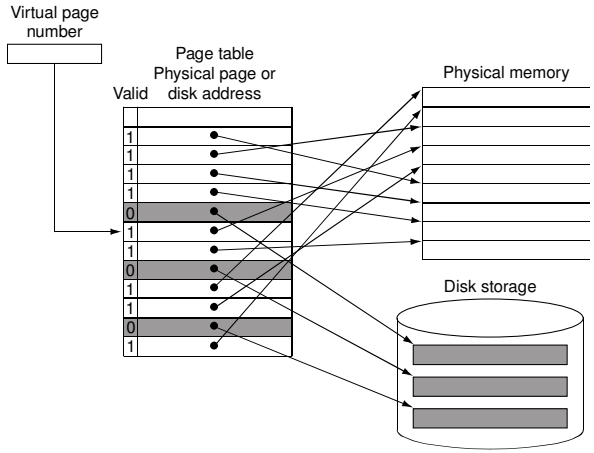
- **Cache block →**
- **Cache miss →**
- **Cache tag →**
- **Byte offset →**



Pages: virtual memory blocks

- **Page faults: the data is not in memory, retrieve it from disk**
  - **huge miss penalty (slow disk), thus**
    - pages should be fairly
    - Replacement strategy:
  - **can handle the faults in software instead of hardware**
- **Writeback or write-through?**

Page Tables



Example – Address Translation Part 1

- **Our virtual memory system has:**
  - **32 bit virtual addresses**
  - **28 bit physical addresses**
  - **4096 byte page sizes**
- **How to split a virtual address?**

Virtual page #	Page offset

- **What will the physical address look like?**

Physical page #	Page offset

- **How many entries in the page table?**

Example – Address Translation Part 2

EX 7-31…

Translate the following addresses:

1. **C0001560**
2. **C0006123**
3. **C0002450**

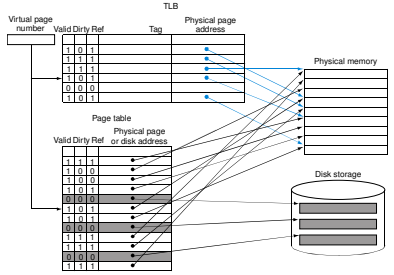
Page Table

	Valid?	Physical Page or Disk Block #
C0000	1	A204
C0001	1	A200
C0002	0	FB00
C0003	1	8003
C0004	1	7290
C0005	0	5600
C0006	1	F5C0

…

Making Address Translation Fast

- **A cache for address translations: translation lookaside buffer**



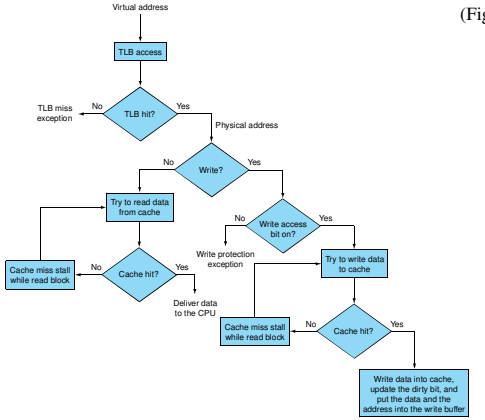
Typical values: 16-512 entries,
miss-rate: .01% - 1%
miss-penalty: 10 – 100 cycles

Protection and Address Spaces

- **Every program has its own "address space"**
  - **Program A's address 0xc000 0200 not same as program B's**
  - **OS maps every virtual address to distinct physical addresses**
- **How do we make this work?**
  - **Page tables –**
  - **TLB –**
- **Can program A access data from program B? Yes, if…**
  1. **OS can map different virtual page #'s to same physical page #'s**
     - So A's 0xc000 0200 = B's 0xb320 0200
  2. **Program A has read or write access to the page**
  3. **OS uses supervisor/kernel protection to prevent user programs from modifying page table/TLB**

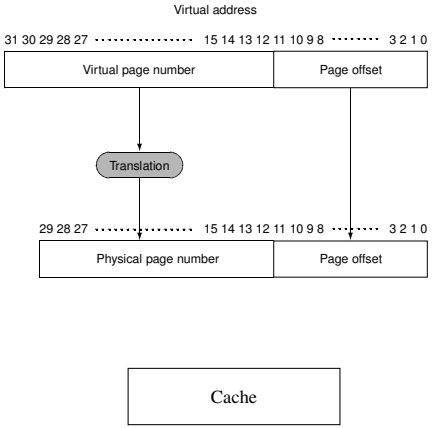
Integrating Virtual Memory, TLBs, and Caches

(Figure 5.25)



TLBs and Caches

What happens after translation?



Modern Systems

Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 cache associativity	4-way (I), 8-way (D) set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 cache associativity	16-way set associative	32-way set associative
L3 replacement	Not Available	Evict block shared by fewest cores
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

Concluding Remarks

- **Fast memories are small, large memories are slow**
  - **We really want fast, large memories**
  - **Caching gives this illusion**
- **Principle of locality**
  - **Programs use a small part of their memory space frequently**
- **Memory hierarchy**
  - **L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk**
- **Memory system design is critical for multiprocessors**