Lec 18: Virtual Memory

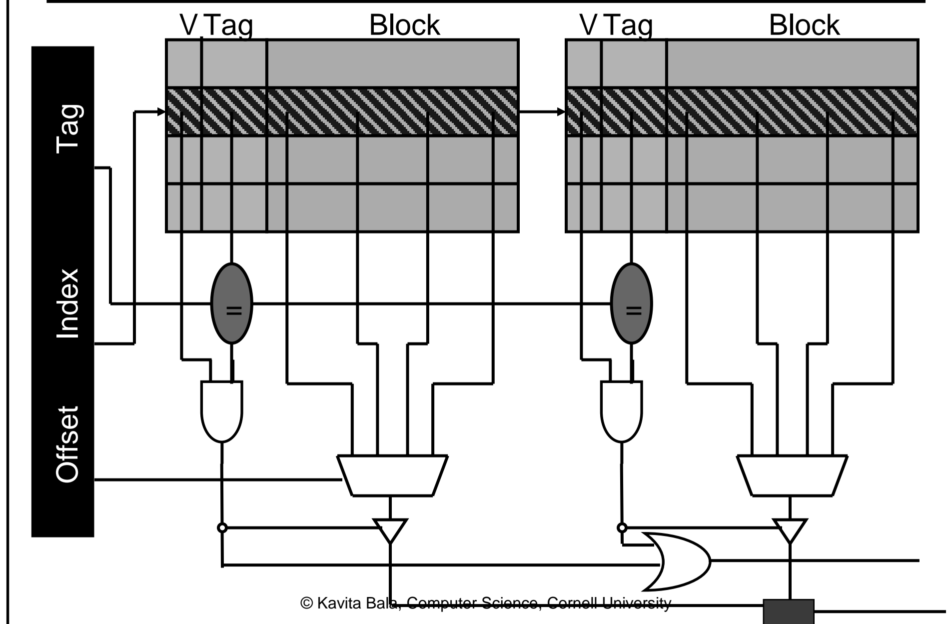
Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

Announcements

- HW 3 out: cache simulation
  - Recitations this week on C/Unix/etc.
  - Due Nov 6th
- HW 4 out on Nov 5th
  - Due Nov 14th
- PA 3 out Nov 14th
  - Due Nov 25th (feel free to turn it in early)
  - Demos and pizza party: Dec 1st or 2nd
- Prelim 2: Dec 4th
- Final project: Due exam week

© Kavita Bala, Computer Science, Cornell University

2-Way Set-Associative Cache



© Kavita Bala, Computer Science, Cornell University

Cache Design

- Need to determine parameters
  - Block size
  - Number of ways of set-associativity
  - Eviction policy
  - Write policy
  - Separate I-cache from D-cache

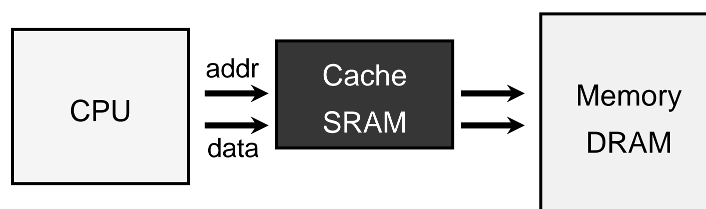
© Kavita Bala, Computer Science, Cornell University

Tradeoff

- Larger sizes reduce the overhead by
  - Reducing the number of tags
  - Reducing the size of each tag
- But
  - Have fewer blocks available
  - And the time to fetch the block on a miss is longer

© Kavita Bala, Computer Science, Cornell University

Cache Writes



- No-Write
  - writes invalidate the cache and go to memory
- Write-Through
  - writes go to main memory and cache
- Write-Back
  - write cache, write main memory only when block is evicted

© Kavita Bala, Computer Science, Cornell University

What about Stores?

- Where should you write the result of a store?
  - If that memory location is in the cache?
    - Send it to the cache
    - Should we also send it to memory right away? (write-through policy)
    - Wait until we kick the block out (write-back policy)
  - If it is not in the cache?
    - Allocate the line (put it in the cache)? (write allocate policy)
    - Write it directly to memory without allocation? (no write allocate policy)

© Kavita Bala, Computer Science, Cornell University

Write-through vs. Write-back

- Write-through is slower
  - But cleaner (memory always consistent)

- Write-back is faster
  - But complicated when multi cores sharing memory

© Kavita Bala, Computer Science, Cornell University

Dirty Bits and Write-Back Buffers

V	D	Tag	Data Byte 0, Byte 1 … Byte N
1	0		
1	1		
1	0		

Line

- Dirty bits indicate which lines have been written
- Dirty bits enable the cache to handle multiple writes to the same cache line without having to go to memory
- Dirty bit reset when line is allocated
- Set when block is written
- Write-back buffer
  - A queue where dirty lines are placed
  - Items added to the end as dirty lines are evicted from the cache
  - Items removed from the front as memory writes are completed

© Kavita Bala, Computer Science, Cornell University

Short Performance Discussion

- Complicated
  - Time from start-to-end (wall-clock time)
  - System time, user time
  - CPI (Cycles per instruction)
- Ideal CPI?

© Kavita Bala, Computer Science, Cornell University

Cache Performance

- Consider hit (H) and miss ratio (M)
- $H \times AT_{cache} + M \times AT_{memory}$
- Hit rate = 1 – Miss rate
- Access Time is given in cycles
- Ratio of Access times, 1:50

- 90% : .90 + .1 x 50 = 5.9
- 95% : .95 + .05 x 50 = .95+2.5=3.45
- 99% : .99 + .01 x 50 = 1.49
- 99.9%: .999 + .001 x 50 = 0.999 + 0.05 = 1.049

© Kavita Bala, Computer Science, Cornell University

Cache Hit/Miss Rate

- Consider processor that is 2x times faster
  - But memory is same speed

- Since AT is access time in terms of cycle time: it doubles 2x
- $H \times AT_{cache} + M \times AT_{memory}$
- Ratio of Access times, 1:100
- 99% : .99 + .01 x 100 = 1.99

© Kavita Bala, Computer Science, Cornell University

Cache Hit/Miss Rate

- Original is 1GHz, 1ns is cycle time
- CPI (cycles per instruction): 1.49
- Therefore, 1.49 ns for each instruction

- New is 2GHz, 0.5 ns is cycle time.
- CPI: 1.99, 0.5ns. 0.995 ns for each instruction.

- So it doesn't go to 0.745 ns for each instruction.
- Speedup is 1.5x (not 2x)

© Kavita Bala, Computer Science, Cornell University

Adding a L2 cache

- CPI: 1.0, Clock: 2GHz
- Access time is 100 ns
- Miss rate: 2%

- Say we add a L2 cache 5ns access time
  - New Miss rate: 0.5%

© Kavita Bala, Computer Science, Cornell University

Misses

- Three types of misses
  - Cold
    - The line is being referenced for the first time
  - Capacity
    - The line was evicted because the cache was not large enough
  - Conflict
    - The line was evicted because of another access whose index conflicted

© Kavita Bala, Computer Science, Cornell University

Cache Conscious Programming

```
int a[NCOL][NROW];
int sum = 0;

for(j = 0; j < NCOL; ++j)
  for(i = 0; i < NROW; ++i)
      sum += a[j][i];
```

- Speed up this program!

© Kavita Bala, Computer Science, Cornell University

Cache Conscious Programming

```
int a[NCOL][NROW];
int sum = 0;

for(j = 0; j < NCOL; ++j)
        for(i = 0; i < NROW; ++i)
                sum += a[j][i];
```

- Every access is a cache miss!

© Kavita Bala, Computer Science, Cornell University

Cache Conscious Programming

```
int a[NCOL][NROW];
int sum = 0;

for(i = 0; i < NROW; ++i)
        for(j = 0; j < NCOL; ++j)
                sum += a[j][i];
```

- Block size: 4, 75% hit rate
- Block size: 8, 87.5% hit rate
- Block size: 16, 93.75% hit rate
- Or you can warm the cache… when does that make sense?

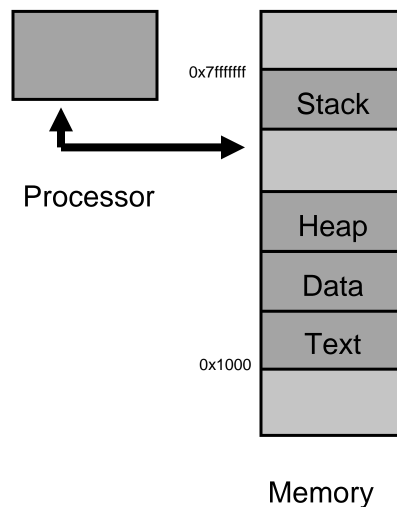
© Kavita Bala, Computer Science, Cornell University

Lec 18: Virtual Memory

Kavita Bala
CS 3410, Fall 2008
Computer Science
Cornell University

Processor & Memory

- Processor's address lines are routed via the system bus to the memory banks
  - Simple, fast
- What happens when the program issues a lw/sw to an invalid location?
  - e.g. 0x000000000
  - Or, uninitialized pointer



© Kavita Bala, Computer Science, Cornell University

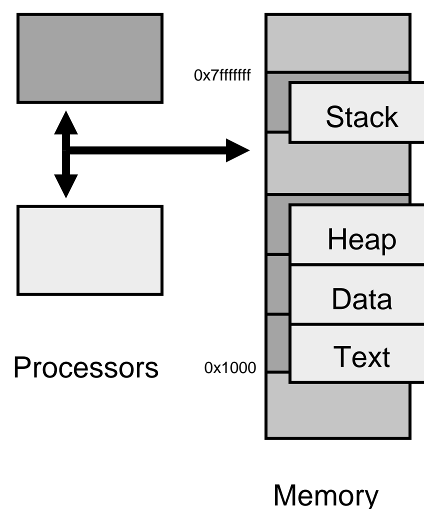
Multiple Processes

- Mail, web browser, skype, …
  - Co-exist
    - How? Take turns?
    - Do they stomp on each other?
  - Multiple processes must co-exist
- Many cores in a computer
  - Multiple processors must co-exist

© Kavita Bala, Computer Science, Cornell University

Multiple Processors

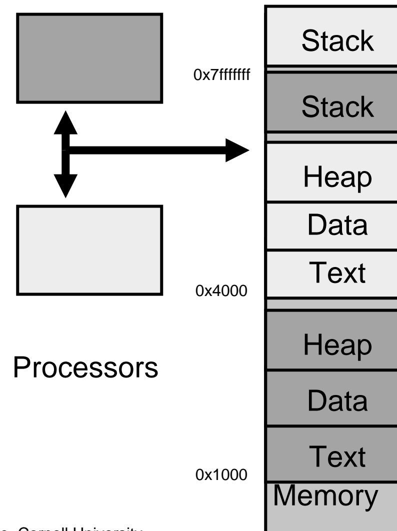
- What happens when another program is executed concurrently on another processor?
  - The addresses will conflict



© Kavita Bala, Computer Science, Cornell University

Solution? Multiple processes/processors

- We could try to relocate the second program to another location
  - Assuming there is one
  - Introduces more problems!
    - What if they don't fit
    - What if it is not contiguous
    - Etc.

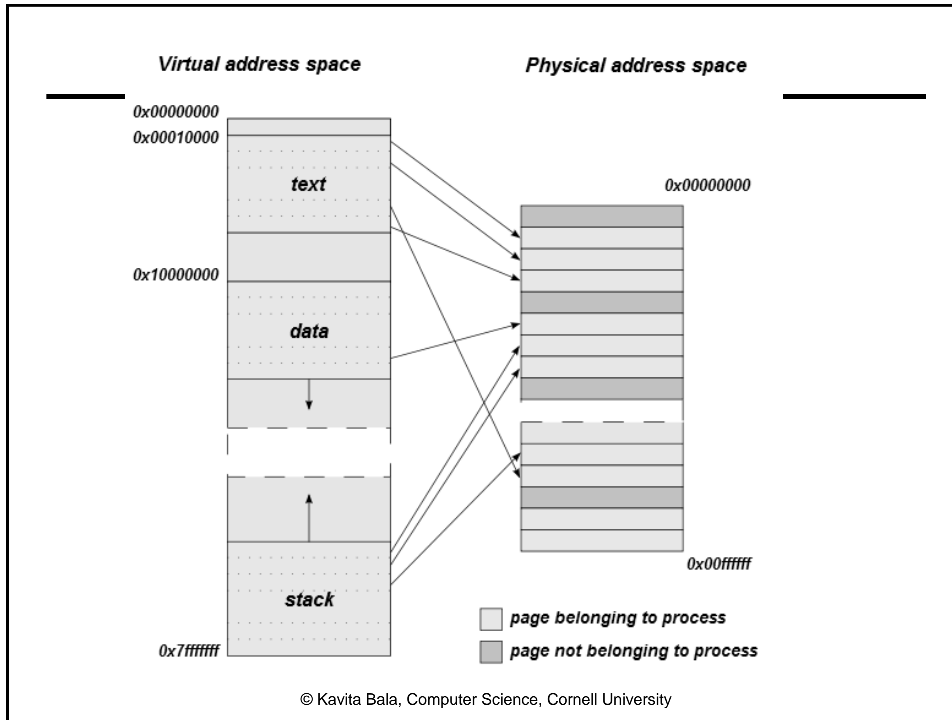


© Kavita Bala, Computer Science, Cornell University

Virtual Memory

- Solves all these problems!
- Each process has its own view of memory
  - Called virtual address space
  - So can conceptually put your code, data in the place you want it
- On-the-fly at runtime
  - Need translation from virtual address space to physical address space of machine
  - Relocate loads and stores to actual memory

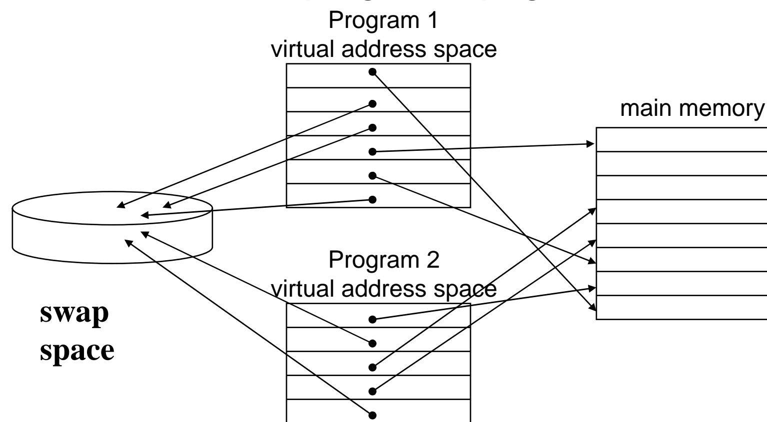
© Kavita Bala, Computer Science, Cornell University



© Kavita Bala, Computer Science, Cornell University

Two Programs Sharing Physical Memory

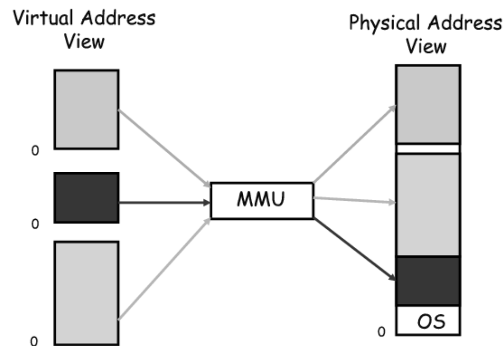
- The starting location of each page (either in main memory or in secondary memory) is contained in the program's page table



© Kavita Bala, Computer Science, Cornell University

Address Space

- Interface
  - Programs load/store to virtual addresses
  - Actual memory uses physical addresses
- Memory Management Unit (MMU)



Virtual Memory Advantages

- Easy relocation
  - Simplifies loading a program for execution by providing for code relocation (i.e., the code can be loaded anywhere in main memory)
  - Also, illusion of contiguous memory
- Easy sharing
  - Allows efficient and safe sharing of memory among multiple programs/multiple cores

© Kavita Bala, Computer Science, Cornell University

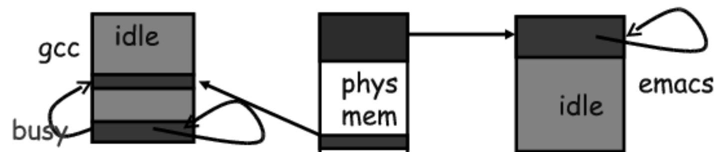
Virtual Memory Advantages

- Can run programs larger than physical memory
  - “Virtualization”
  - Use main memory as a “cache” for secondary memory: illusion of large memory
  - Based on Principle of Locality
    - The 90/10 rule
    - A program is likely to access a relatively small portion of its address space during any period of time

© Kavita Bala, Computer Science, Cornell University

Virtual Memory Advantages

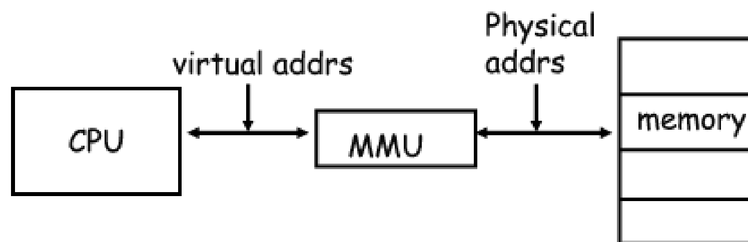
- Virtualization
  - CPU: if process is not doing anything, switch
  - Memory: when not using it, somebody else can use it



© Kavita Bala, Computer Science, Cornell University

How to make it work?

- Challenge: Virtual Memory can be slow!
- At run-time: virtual address must be translated to a physical address
- MMU (combination of hardware and software)



© Kavita Bala, Computer Science, Cornell University

Address Translation

- How to translate addresses?
  - Per word? Much too expensive
  - Per block? Sure, but what is block size?

- Costs dictate granularity of translation
  - Cost to disk is very large
  - Block size has to be large too
    - Amortization

© Kavita Bala, Computer Science, Cornell University

Paging

- Divide memory into small pages
- A program's address space is divided into pages (all one fixed size)
  - Typical: 4KB to 16KB
  - Example:
    - virtual address space: 2^32 = 4GB = 4 x 2^30
    - physical address space: 512 MB or 1 GB
    - page size: 4KB
    - Number of pages in memory: 2^18
    - Number of page table entries: 2^20
    - Page Table size = 4MB

© Kavita Bala, Computer Science, Cornell University

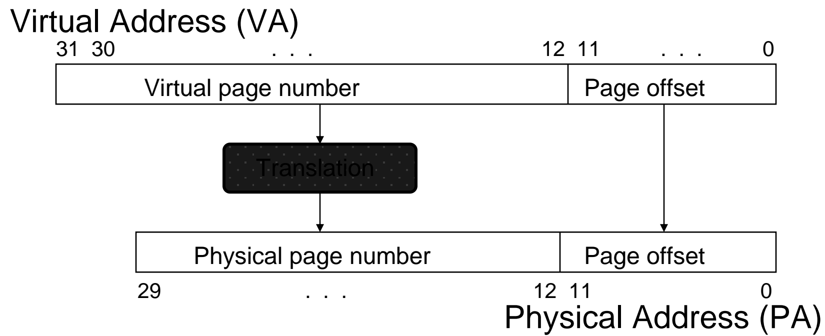
Page Table

- Each process has separate mapping of virtual to physical pages
- Page Table: stores this translation
  - Basically a huge array of translations

© Kavita Bala, Computer Science, Cornell University

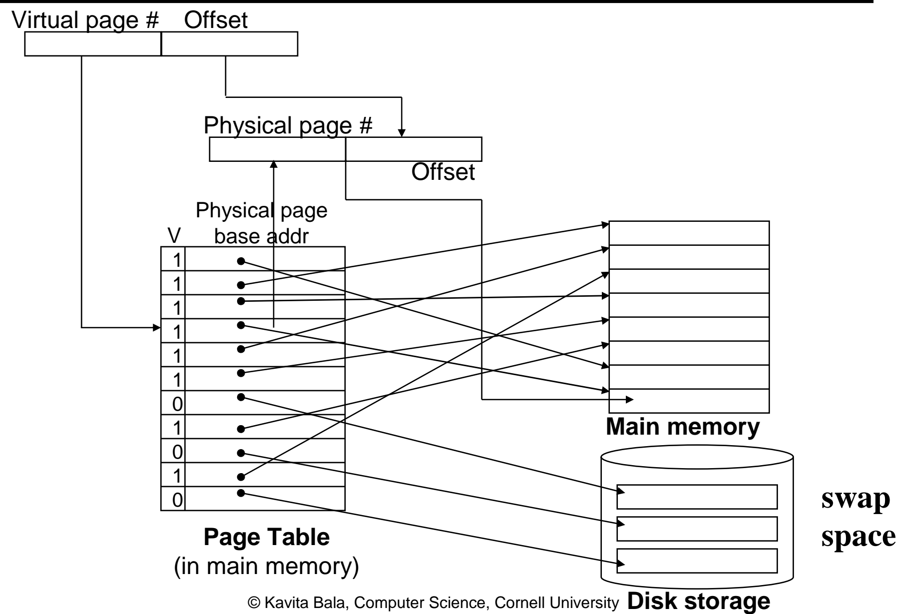
Address Translation

- So each memory request *first* requires an address translation from the virtual space to the physical space



© Kavita Bala, Computer Science, Cornell University

Page Table for Translation



© Kavita Bala, Computer Science, Cornell University