CS184b: Computer Architecture (Abstractions and Optimizations)

Day 9: April 25, 2003
Virtual Memory and Caching

Today

- Virtual Memory
  - Problems
    - memory size
    - multitasking
  - Different from caching?
  - TLB
  - Co-existing with caching
- Caching
  - Spatial, multi-level …

Problem 1:

- Real memory is finite
- Problems we want to run are bigger than the real memory we may be able to afford…
  - larger set of instructions / potential operations
  - larger set of data
- Given a solution that runs on a big machine
  - would like to have it run on smaller machines, too
    - but maybe slower / less efficiently

Opportunity 1:

- Instructions touched < Total Instructions
- Data touched
  - not uniformly accessed
  - working set < total data
  - locality
    - temporal
    - spatial

Problem 2:

- Convenient to run more than one program at a time on a computer
- Convenient/Necessary to isolate programs from each other
  - shouldn't have to worry about another program writing over your data
  - shouldn't have to know about what other programs might be running
  - don't want other programs to be able to see your data

Problem 2:

- If share same address space
  - where program is loaded (puts its data) depends on other programs (running? Loaded?) on the system
- Want abstraction
  - every program sees same machine abstraction independent of other running programs

One Solution

- Support large address space
- Use cheaper/larger media to hold complete data
- Manage physical memory “like a cache”
- Translate large address space to smaller physical memory
- Once do translation
  - translate multiple address spaces onto real memory
  - use translation to define/limit what can touch

Conventionally

- Use magnetic disk for secondary storage
- Access time in ms
  - e.g. 9ms
  - 9 million cycles latency
- bandwidth ~100Mb/s
  - vs. read 64b data item at GHz clock rate
    - 64Gb/s

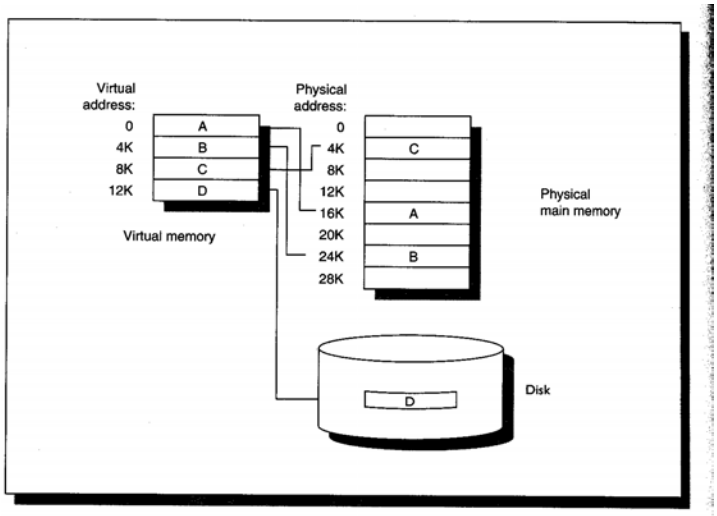
Like Caching?

- Cache tags on all of Main memory?
- Disk Access Time >> Main Memory time
- Disk/DRAM >> DRAM/L1 cache
  - bigger penalty for being wrong
    - conflict, compulsory
- …also historical
  - solution developed before widespread caching...

Mapping

- Basic idea
  - map data in large blocks (pages)
  - use memory table
  - to record physical memory location for each, mapped memory block

Address Mapping



[Hennessy and Patterson 5.36e2/5.31e3]

Mapping

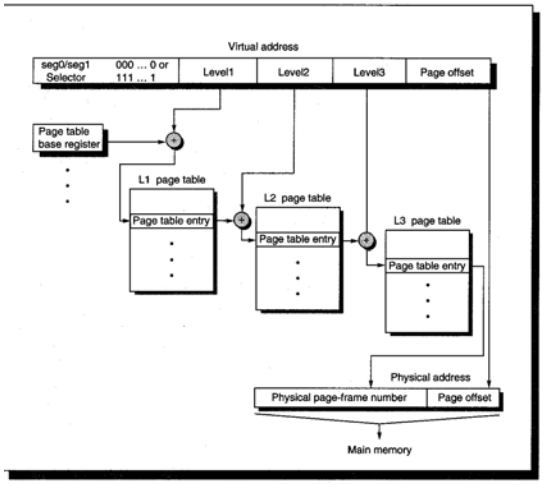
- 32b address space
- 4Kb pages
- $2^{32}/2^{12}=2^{20}$=1M address mappings

- Very large translation table

Translation Table

- Traditional solution
  - from when 1M words >= real memory
    - (but we're also growing beyond 32b addressing)
  - break down page table hierarchically
  - divide 1M entries into 4*1M/4K=1K pages
  - use another translation table to give location of those 1K pages
  - …multi-level page table

Page Mapping



[Hennessy and Patterson 5.43e2/5.39e3]

Page Mapping Semantics

- Program wants value contained at A
- pte1=top_pte[A[32:24]]
- if pte1.present
  - ploc=pte1[A[23:12]]
  - if ploc.present
    - Aphys=ploc<<12 + (A [11:0])
    - Give program value at Aphys
  - else … load page
- else … load pte

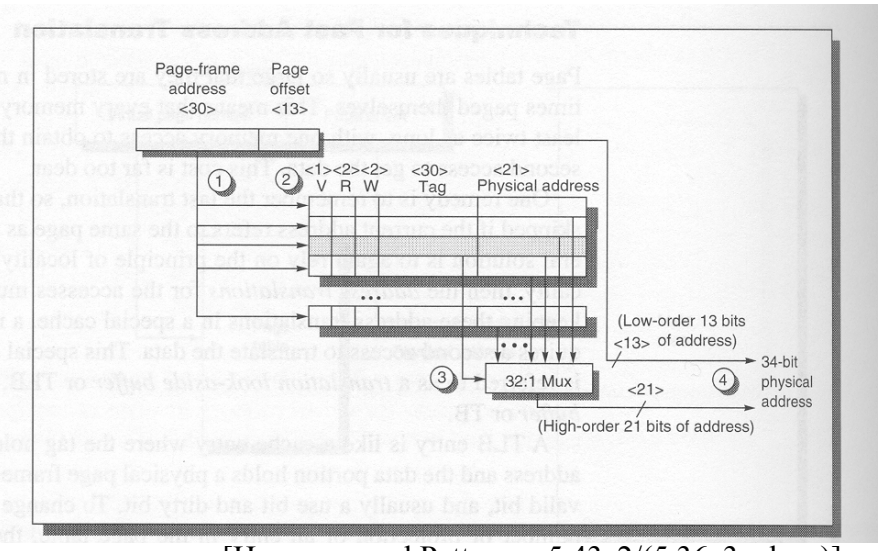
Early VM Machine

- Did something close to this...

Modern Machines

- Keep hierarchical page table
- Optimize with lightweight hardware assist
- Translation Lookaside Buffer (TLB)
  - Small associative memory
  - maps virtual address to physical
  - in series/parallel with every access
  - faults to software on miss
  - software uses page tables to service fault

TLB



[Hennessy and Patterson 5.43e2/(5.36e3, close)]

VM Page Replacement

- Like cache capacity problem
- Much more expensive to evict wrong thing
- Tend to use LRU replacement
  - touched bit on pages (cheap in TLB)
  - periodically (TLB miss? Timer interrupt) use to update touched epoch
- Writeback (not write through)
- Dirty bit on pages, so don't have to write back unchanged page (also in TLB)

VM (block) Page Size

- Larger than cache blocks
  - reduce compulsory misses
  - full mapping
    - not increase conflict misses
    - could increase capacity misses
  - reduce size of page tables, TLB required to maintain working set

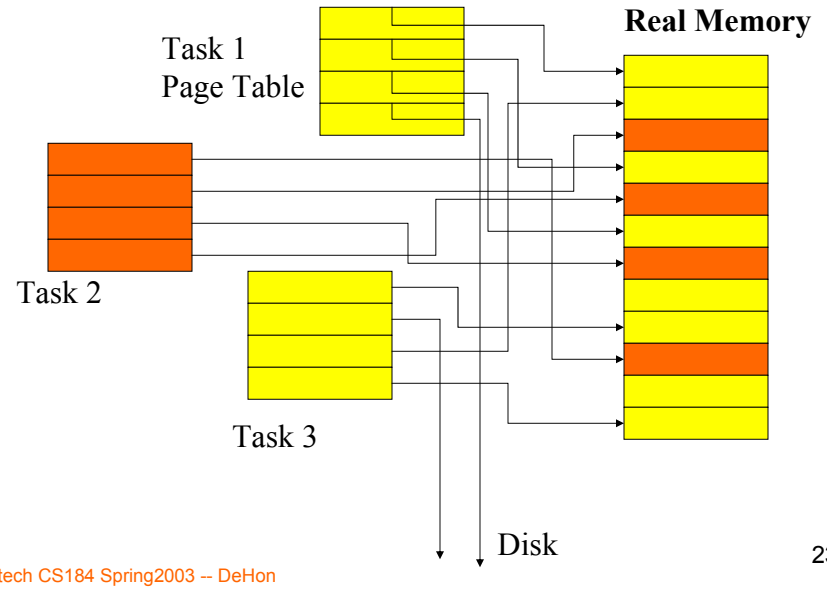
VM Page Size

- Modern idea: allow variety of page sizes
  - "super" pages
  - save space in TLBs where large pages viable
    - instruction pages
  - decrease compulsory misses where large amount of data located together
  - decrease fragmentation and capacity costs when not have locality

VM for Multitasking

- Once we're translating addresses
  - easy step to have more than one page table
  - separate page table (address space) for each process
  - code/data can live anywhere in real memory and have consistent virtual memory address
  - multiple live tasks may map data to same VM address and not conflict
    - independent mappings

Multitasking Page Tables



VM Protection/Isolation

- If a process cannot map an address
  - real memory
  - memory stored on disk
- and a process cannot change it page-table
  - and cannot bypass memory system to access physical memory...
- the process has no way of getting access to a memory location

Elements of Protection

- Processor runs in (at least) two modes of operation
  - user
  - privileged / kernel
- Bit in processor status indicates mode
- Certain operations only available in privileged mode
  - e.g. updating TLB, PTEs, accessing certain devices

System Services

- Provided by privileged software
  - e.g. page fault handler, TLB miss handler, memory allocation, io, program loading
- System calls/traps from user mode to privileged mode
  - …already seen trap handling requirements...
- Attempts to use privileged instructions (operations) in user mode generate faults

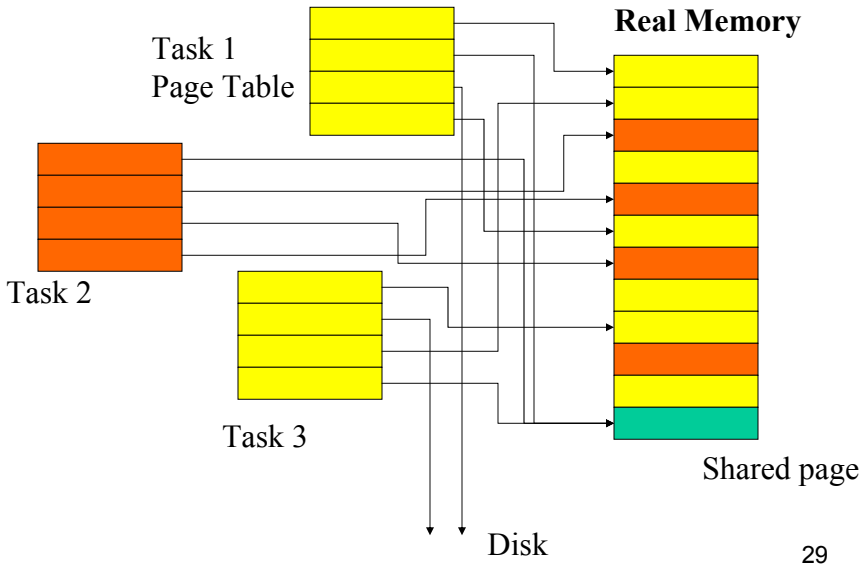
System Services

- Allows us to contain behavior of program
  - limit what it can do
  - isolate tasks from each other
- Provide more powerful operations in a carefully controlled way
  - including operations for bootstrapping, shared resource usage

Also allow controlled sharing

- When want to share between applications
  - read only shared code
    - e.g. executables, common libraries
  - shared memory regions
    - when programs want to communicate
    - (do know about each other)

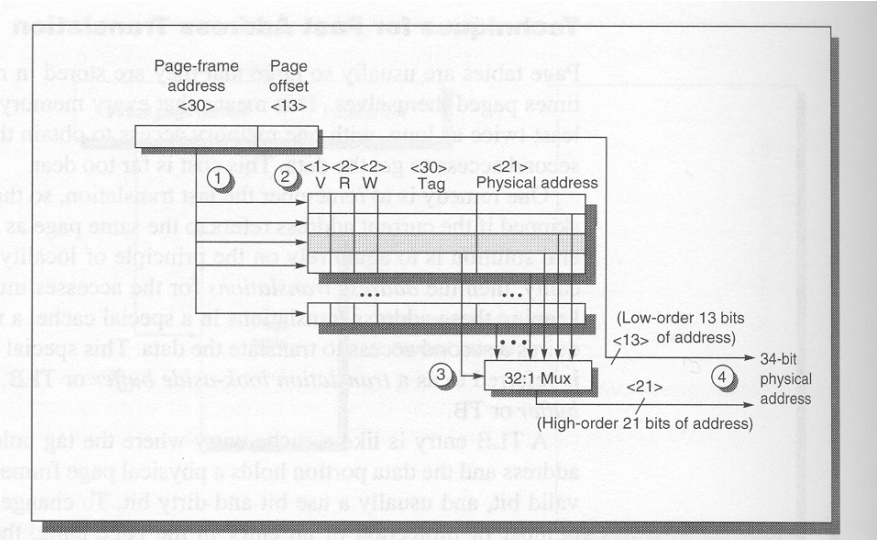
Multitasking Page Tables



Page Permissions

- Also track permission to a page in PTE and TLB
  - read
  - write
    - support read-only pages
    - pages read by some tasks, written by one

TLB



[Hennessy and Patterson 5.43e2]

Page Mapping Semantics

- Program wants value contained at A
- pte1=top_pte[A[32:24]]
- if pte1.present
  - ploc=pte1[A[23:12]]
  - if ploc.present and ploc.read
    - Aphys=ploc<<12 + (A [11:0])
    - Give program value at Aphys
  - else … load page
- else … load pte

VM and Caching?

- Should cache be virtually or physically tagged?
  - Tasks speaks virtual addresses
  - virtual addresses only meaningful to a single process

Virtually Mapped Cache

- L1 cache access directly uses address
  - don't add latency translating before check hit
- Must flush cache between processes?

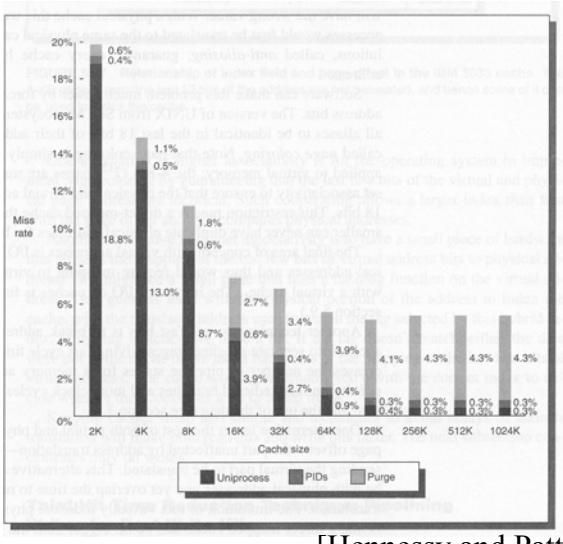
Physically Mapped Cache

- Must translate address before can check tags
  - TLB translation can occur in parallel with cache read
    - (if direct mapped part is within page offset)
  - contender for critical path?
- No need to flush between tasks
- Shared code/data not require flush/reload between tasks
- Caches big enough, keep state in cache between tasks

Virtually Mapped

- Mitigate against flushing
  - also tagging with process id
  - processor (system?) must keep track of process id requesting memory access
- Still not able to share data if mapped differently
  - may result in aliasing problems
    - (same physical address, different virtual addresses in different processes)

Virtually Addressed Caches



[Hennessy and Patterson 5.26]

Spatial Locality

Spatial Locality

- Higher likelihood of referencing nearby objects
  - instructions
    - sequential instructions
    - in same procedure (procedure close together)
    - in same loop (loop body contiguous)
  - data
    - other items in same aggregate
    - other fields of struct or object
    - other elements in array
    - same stack frame

Exploiting Spatial Locality

- Fetch nearby objects
- Exploit
  - high-bandwidth sequential access (DRAM)
  - wide data access (memory system)
- To bring in data around memory reference

Blocking

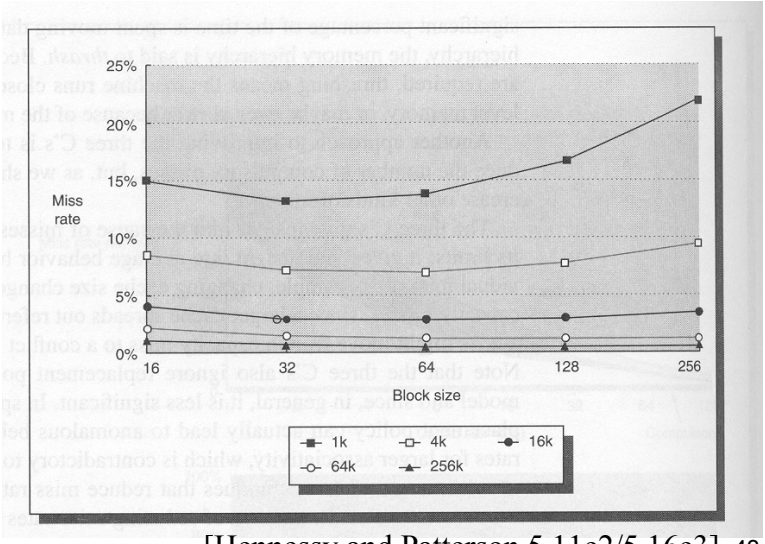
- Manifestation: Blocking / Cache lines
- Cache line bigger than single word
- Fill cache line on miss

- Size b-word cache line
  - sequential access, miss only 1 in b references

Blocking

- Benefit
  - less miss on sequential/local access
  - amortize cache tag overhead
    - (share tag across b words)
- Costs
  - more fetch bandwidth consumed (if not use)
  - more conflicts
    - (maybe between non-active words in cache line)
  - maybe added latency to target data in cache line

Block Size



[Hennessy and Patterson 5.11e2/5.16e3]

Optimizing Blocking

- Separate valid/dirty bit per word
  - don't have to load all at once
  - writeback only changed
- Critical word first
  - start fetch at missed/stalling word
  - then fill in rest of words in block
  - use valid bits deal with those not present

Multi-level Cache

From last time

500ps cycle

Cache Numbers

- No Cache
  - CPI=Base+0.3*100=Base+30
- Cache at CPU Cycle (10% miss)
  - CPI=Base+0.3*0.1*100=Base +3
- Cache at CPU Cycle (1% miss)
  - CPI=Base+0.3*0.01*100=Base +0.3

Implication (Cache Numbers)

- To get 1% miss rate?
  - 64KB-256KB cache
  - not likely to support multi GHz CPU rate
- More modest
  - 4KB-8KB
  - 7% miss rate
- 100x performance gap cannot really be covered by single level of cache

…do it again...

- If something works once,
  - try to do it again
- Put second (another) cache between CPU cache and main memory
  - larger than fast cache
  - hold more … less misses
  - smaller than main memory
  - faster than main memory

Multi-level Caching

- First cache: Level 1 (L1)
- Second cache: Level 2 (L2)
- CPI = Base CPI
  +Refs/Instr (L1 Miss Rate)(L2 Latency) +
  +Ref/Instr (L2 Miss Rate)(Memory Latency)

Multi-Level Numbers

- L1, 500ps, 4KB, 10% miss
- L2, 4ns, 128KB, 1% miss
- Main, 50ns
- L1 only CPI=Base+0.3*0.1*100=Base +3
- L2 only CPI=Base+0.3*(0.99*7+0.01*92) =Base+3.35
- L1/L2=Base+(0.3*0.1*7 + 0.01*92) =Base+1.49

Numbers

- Maybe could use L3?
  - Hypothesize: L3, 10ns, 1MB, 0.2%
- L1/L2/L3=Base+(0.3*(0.1*7 + 0.01*12+0.002*80)
  =Base+0.21+0.048+0.048 =Base+1.29

Rate Note

- Previous slides:
  - "L2 miss rate" = miss of L2
    - all access; not just ones which miss L1
  - If talk about miss rate wrt only L2 accesses
    - higher since filter out locality from L1
- H&P: **global miss rate**
- **Local miss rate**: misses from accesses seen in L2
- Global miss rate
  - L1 miss rate × L2 local miss rate

Segregation

I-Cache/D-Cache

- Processor needs one (or several) instruction words per cycle
- In addition to the data accesses
  - Instr/Ref*Instr Issue
- Increase bandwidth with separate memory blocks (caches)

I-Cache/D-Cache

- Also different behavior
  - more locality in I-cache
  - afford less associativity in I-cache?
  - Make I-cache wide for multi-instruction fetch
  - no writes to I-cache
- Moderately easy to have multiple memories
  - know which data where

By Levels?

- L1
  - need bandwidth
  - typically split (contemporary)
- L2
  - hopefully bandwidth reduced by L1
  - typically unified

Non-blocking

How disruptive is a Miss?

- With
  - multiple issue
  - a reference every 3-4 instructions
- memory references 1+ times per cycle
- Miss means multiple (8,20,100?) cycles to service
- Each miss could holds up 10's to 100's of instructions...

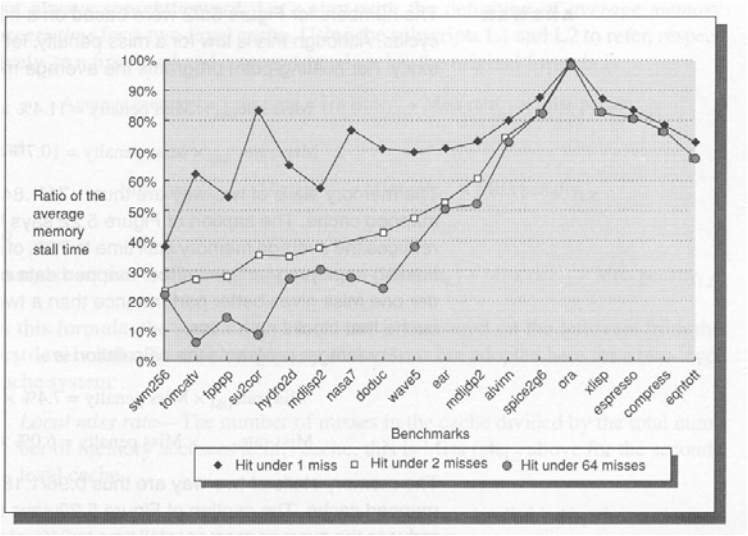
Minimizing Miss Disruption

- Opportunity:
  - out-of-order execution
    - maybe we can go on without it
    - scoreboarding/tomasulo do dataflow on arrival
    - go ahead and issue other memory operations
  - next ref might be in L1 cache
    - …while miss referencing L2, L3, etc.
  - next ref might be in a different bank
    - can access (start access) while waiting for bank latency

Non-Blocking Memory System

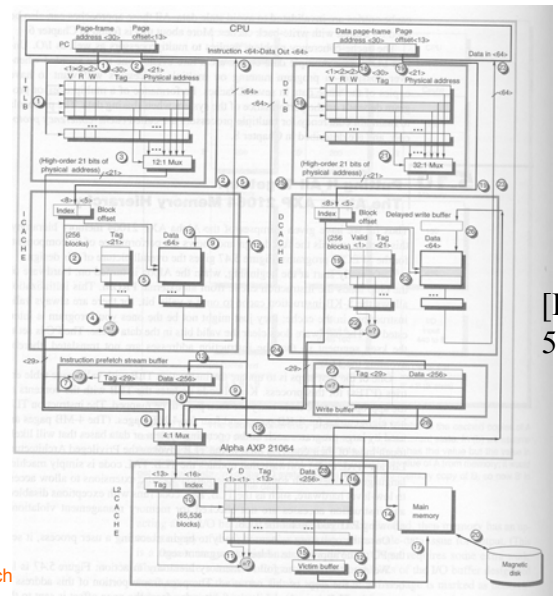
- Allow multiple, outstanding memory references
- Need split-phase memory operations
  - separate request data
  - from data reply (read -- complete for write)
- Reads:
  - easy, use scoreboarding, etc.
- Writes:
  - need write buffer, bypass...

Non-Blocking



[Hennessy and Patterson 5.22e2/5.23e3]

Processor Memory Systems



[Hennessy and Patterson 5.47e2, 5.43e3 similar]

Big Ideas

- Virtualization
  - share scarce resource among many consumers
  - provide “abstraction” that own resource
    - not sharing
  - make small resource look like bigger resource
    - as long as backed by (cheaper) memory to manage state and abstraction
- Common Case
- Add a level of Translation

Big Ideas

- Structure
  - spatial locality
- Engineering
  - worked once, try it again…until won’t work