

Virtual Memory


I wish we were still doing NAND gates...

Finally! A lecture on something I understand – PAGE FAULTS!

```

int vread(int vpage, int p0) {
    if (!vpage) == 0)
        return 0;
    return (PMM[Vpage]) << p0 | p0;
}

/* Handle a missing page... */
void pageFault(int vpage) {
    int i;
    i = selectEmptyPage();
    if (i != -1)
        noLockPage(DLAddr(i), PMM(i));
    P[i] = 0;
    PMM[Vpage] = PMM(i);
    noLockPage(DLAddr(Vpage), PMM(i));
    P[Vpage] = 1;
    D[Vpage] = 0;
}
    
```



Study Chapters 7.4-7.6

Comp 120 – Spring 2005
04/14/04
L21 – Virtual Memory 1

Recap: Memory Hierarchy of a Modern Computer System

By taking advantage of the principle of locality:

- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.

Processor

Control

Datapath

Registers

On-Chip Cache

Caching

Second Level Cache (SRAM)

Virtual Memory

Main Memory (DRAM)

Secondary Storage (Disk)

Tertiary Storage (Disk)

Speed (ns):	1s	10s	100s	10,000,000s	10,000,000,000s
Size (bytes):	100s	Ks	Ms	Gs	Ts
				(10s ms)	(10s sec)

DAP Fa97, © U.C.B
L21 – Virtual Memory 2

Comp 120 – Spring 2005
04/14/04

You can never have too much memory!



Now that we've learned how to FAKE a FAST memory, we'll turn our attention to FAKING a large memory.

Top 10 Reasons for a BIG Address Space



10. Keeping Micron's memory division in business.
9. Unique addresses within every internet host.
8. Generates good Comp 120 Final problems.
7. Performing ADD via table lookup
6. Support for meaningless advertising hype
5. Emulation of a Turing Machine's tape.
4. Supporting lazy programmers.
3. Isolating ISA from IMPLEMENTATION
 - details of HW configuration shouldn't enter into SW design
2. Usage UNCERTAINTY
 - provides for run-time expansion of stack and heap
1. Programming CONVENIENCE
 - create regions of memory with different semantics: read-only, shared, etc.
 - avoid annoying bookkeeping

Lessons from History...

There is only one mistake that can be made in computer design that is difficult to recover from—not having enough address bits for memory addressing and memory management.

Gordon Bell and Bill Strecker
speaking about the PDP-11 in 1976

A partial list of successful machines that eventually starved to death for lack of address bits includes the PDP 8, PDP 10, PDP 11, Intel 8080, Intel 8086, Intel 80186, Intel 80286, Motorola 6800, AMI 6502, Zilog Z80, Cray-1, and Cray X-MP.

Hennessey & Patterson

Why? Address size determines minimum width of anything that can hold an address: PC, registers, memory words, HW for address arithmetic (branches/jumps, loads/stores). When you run out of address space it's time for a new ISA!

Squandering Address Space

Address Space



STACK: How much to reserve? (consider RECURSION!)

HEAP: N variable-size data records...
Bound N? Bound Size?

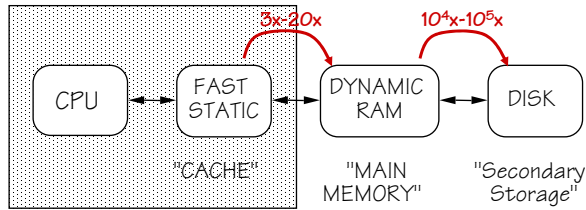
OBSERVATIONS:

- Can't BOUND each usage... without compromising use.
- Actual use is SPARSE
- Working set even MORE sparse

CODE, large monolithic programs (eg. Office, Netscape)...

- only small portions might be used
- add-ins and plug-ins
- shared libraries/DLLs
-

Extending the Memory Hierarchy



So far, we've used SMALL fast memory + BIG slow memory to fake a BIG FAST memory.

Can we combine RAM and DISK to fake DISK sized at near RAM speeds?

VIRTUAL MEMORY

- use of RAM as cache to much larger storage pool, on slower devices
- TRANSPARENCY - VM locations "look" the same to program whether on DISK or in RAM.
- ISOLATION of RAM size from software.

Virtual Memory trade-offs

Memory:

40 cycles latency, then 1 word every 0.5 cycles (80/1)

Harddisk: 10ms latency, 50MB/s BW

10M cycles latency, then 1 word every 80 cycles (125,000/1)

Impact?



Virtual Memory

ILLUSION: Huge memory
(2^{32} bytes? 2^{64} bytes?)

ACTIVE USAGE: small fraction (2^{24} bytes?)

Actual HARDWARE:

- 2^{26} (64M) bytes of RAM
- 2^{35} (32G) bytes of DISK...
- ... maybe more, maybe less!

ELEMENTS OF DECEIT:

- Partition memory into "Pages" (2K-4K-8K)
- MAP a few to RAM, others to DISK
- Keep "HOT" pages in RAM.

Memory Management Unit

CPU VA → MMU → PA → RAM

"VIRTUAL" MEMORY PAGES → "PHYSICAL" MEMORY PAGES

Comp 120 – Spring 2005 04/14/04 L21 – Virtual Memory 9

Simple Page Map Design

Page Index

Virtual Page #

Page Map

Physical Page #

FUNCTION: Given Virtual Address,

- Map to PHYSICAL address
- OR
- Cause **PAGE FAULT** allowing page replacement

Why use HIGH address bits to select page?
... LOCALITY.
Keeps related data on same page.

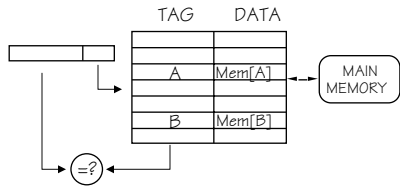
Why use LOW address bits to select cache line?
... LOCALITY.
Keeps related data from competing for same cache lines.

Virtual Memory → PAGEMAP → Physical Memory

D R P P N

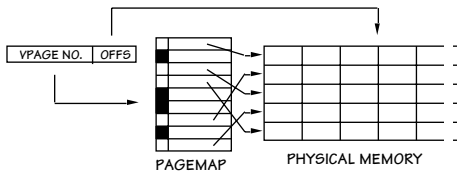
Comp 120 – Spring 2005 04/14/04 L21 – Virtual Memory 10

Virtual Memory vs. Cache



CACHE:

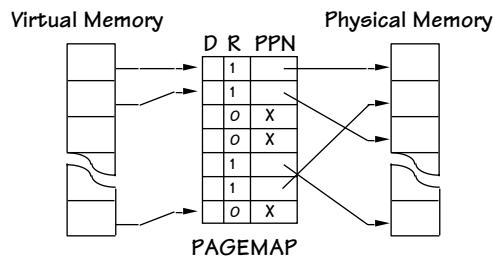
- Relatively short blocks
- Few lines: scarce resource
- miss time: 3x-20x hit time



VIRTUAL MEMORY:

- Disk: long latency, fast xfer
→ miss time: $\sim 10^5$ x hit time
→ write-back essential!
→ large pages in RAM
- Lots of lines: one for each page
- Tags in page map, data in physical memory

Virtual Memory: An EE's view

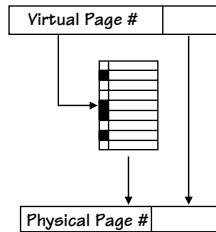


Pagemap Characteristics:

- One entry per **virtual** page!
- RESIDENT bit = 1 for pages stored in RAM, or 0 for non-resident (disk or unallocated). Page fault when R = 0.
- Contains PHYSICAL page number (PPN) of each resident page
- DIRTY bit says we've changed this page since loading it from disk (and therefore need to write it to disk when it's replaced)
- More bits possible to determine access, e.g. read-only

Virtual Memory: A CS's view

Problem: Translate
VIRTUAL ADDRESS
to PHYSICAL ADDRESS



```
int VtoP(int VPageNo, int PO) {
    if (R[VPageNo] == 0)
        PageFault(VPageNo);
    return (PPN[VPageNo] << p) | PO;
}

/* Handle a missing page... */
void PageFault(int VPageNo) {
    int i;

    i = SelectLRUPage();
    if (D[i] == 1)
        WritePage(DiskAdr[i], PPN[i]);
    R[i] = 0;

    PPN[VPageNo] = PPN[i];
    ReadPage(DiskAdr[VPageNo], PPN[i]);
    R[VPageNo] = 1;
    D[VPageNo] = 0;
}
```

The HW/SW Balance

IDEA:

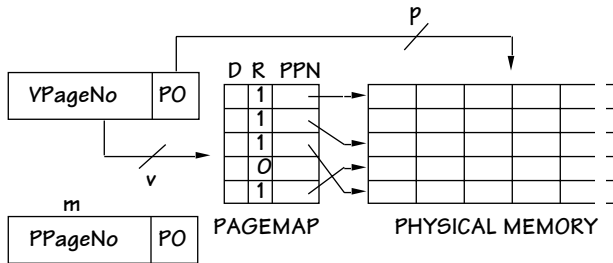
- devote **HARDWARE** to high-traffic, performance-critical path
- use (slow, cheap) **SOFTWARE** to handle exceptional cases

hardware	{	<pre>int VtoP(int VPageNo, int PO) { if (R[VPageNo] == 0) PageFault(VPageNo); return (PPN[VPageNo] << p) PO; }</pre>
software	{	<pre>/* Handle a missing page... */ void PageFault(int VPageNo) { int i = SelectLRUPage(); if (D[i] == 1) WritePage(DiskAdr[i], PPN[i]); R[i] = 0; PA[VPageNo] = PPN[i]; ReadPage(DiskAdr[VPageNo], PPN[i]); R[VPageNo] = 1; D[VPageNo] = 0; }</pre>

HARDWARE performs address translation, detects page faults:

- running program interrupted ("suspended");
- PageFault(...) is forced;
- On return from PageFault; running program continues

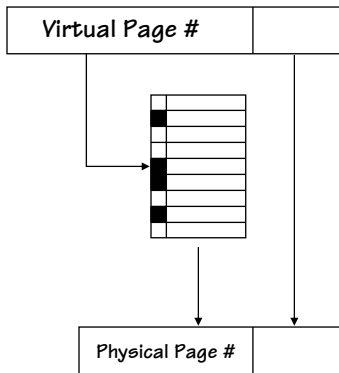
Page Map Arithmetic



- $(v + p)$ bits in virtual address
- $(m + p)$ bits in physical address
- 2^v number of VIRTUAL pages
- 2^m number of PHYSICAL pages
- 2^p bytes per physical page
- 2^{v+p} bytes in virtual memory
- 2^{m+p} bytes in physical memory
- $(m+2)v$ bits in the page map

Typical page size: 1K – 8K bytes
 Typical $(v+p)$: 32 (or more) bits
 Typical $(m+p)$: 26 – 30 bits
 (64 – 1024 MB)

Example: Page Map Arithmetic



SUPPOSE...

- 32-bit Virtual address
- 2^{13} page size (8 KB)
- 2^{26} RAM (64 MB)

THEN:

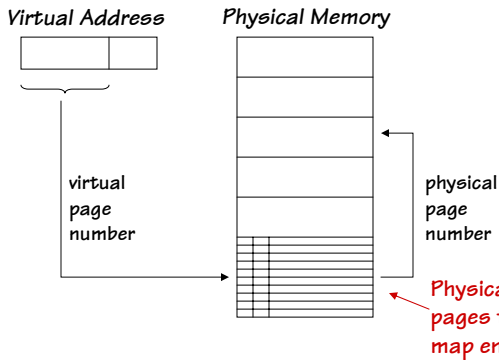
- # Physical Pages = $\frac{2^{26-13}}{2^{13}} = 2^{13} = 8192$
- # Virtual Pages = $\frac{2^{32-13}}{2^{13}} = 2^{19}$
- # Page Map Entries = 524,288

Use SRAM for page map??? **OUCH!**

RAM-Resident Page Maps

SMALL page maps can use dedicated RAM... but, gets expensive for big ones!

SOLUTION: Move page map into MAIN MEMORY:

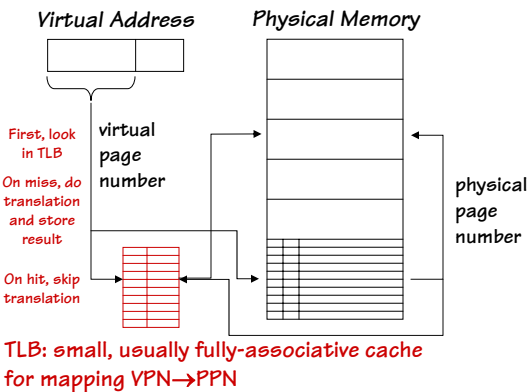


PROBLEM:
Each memory reference
now takes 2 accesses
to physical memory!

Translation Look-aside Buffer (TLB)

PROBLEM: 2x performance hit... each memory reference now takes 2 accesses!

SOLUTION: CACHE the page map entries

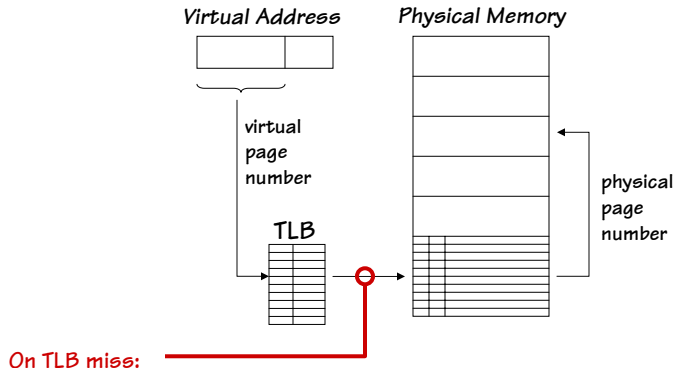


IDEA:
LOCALITY in memory
reference patterns →
SUPER locality in references
to page map

VARIATIONS:

- sparse page map storage
- paging the page map (next slides)

Optimizing Sparse Page Maps



On TLB miss:

- look up VPN in "sparse" data structure (e.g., a list of VPN-PPN pairs)
- use hash coding to speed search
- only have entries for ALLOCATED pages
- allocate new entries "on demand"
- time penalty? LOW if TLB hit rate is high...

Should we do this in HW or SW?

Multilevel Page Maps

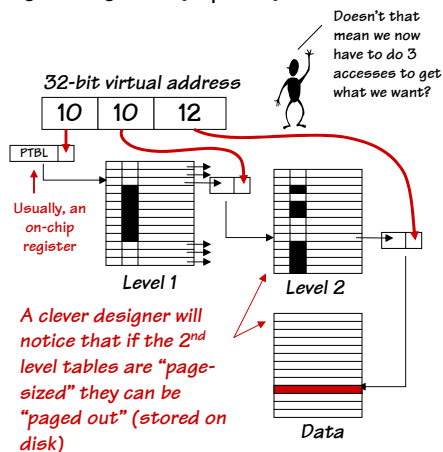
Given a HUGE virtual memory, the cost of storing all of the page map entries in RAM may STILL be too expensive...

SOLUTION: A hierarchical page map... take advantage of the observation that while the virtual memory address space is large, it is generally sparsely populated with clusters of pages.

Consider a machine with a 32-bit virtual address space and 64 MB (26-bit) of physical memory that uses 4 KB pages.

Assuming 4 byte page-table entries, a single-level page map requires 4MB (>6% of the available memory). Of these, more than 98% will reference non-resident pages (Why?).

A 2-level look-up increases the size of the worse-case page table slightly. However, if a first level entry is non-resident bit it saves large amounts of memory.



Example: Mapping VAs to PAs

Suppose

- virtual memory of 2^{32} (4G) bytes
- physical memory of 2^{30} (1G) bytes
- page size is 2^{14} (16 K) bytes

VPN	R	D	PPN
0	0	0	2
1	1	1	7
2	1	0	0
3	1	0	5
4	0	0	5
5	1	0	3
6	1	1	2
7	1	0	4
8	1	0	1
...			

1. How many pages can be stored in physical memory at once?
 $2^{30-14} = 2^{16} = 64K$
2. How many entries are there in the page table?
 $2^{32-14} = 2^{18} = 256K$
3. How many bits are necessary per entry in the page table? (Assume each entry has PPN, resident bit, dirty bit)
 $16 \text{ (PPN)} + 2 = 18$
4. How many pages does the page table require?
 $2^{18} / (2^{14-2}) = 2^6 = 64$
5. A portion of the page table is given to the left. What is the physical address for virtual address $0x0000c120$?

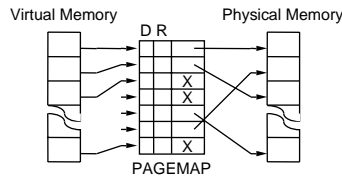
...01010000100110000
 $0x00014120$

Contexts

A context is a complete set of mappings from VIRTUAL to PHYSICAL locations, as dictated by the full contents of the page map:

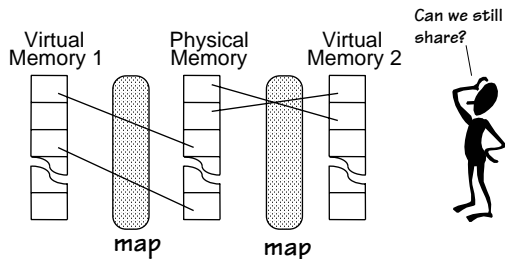


We might like to support multiple VIRTUAL to PHYSICAL Mappings and, thus, multiple Contexts.

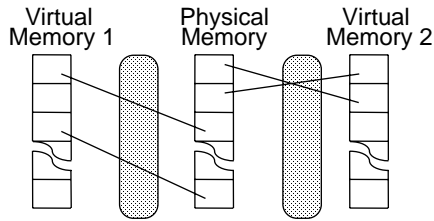


Several programs may be simultaneously loaded into main memory, each in its separate context:

“Context Switch”:
Reload the page map!



Contexts: A Sneak Preview



Every application can be written as if it has access to all of memory, without considering where other applications reside.

First Glimpse at a VIRTUAL MACHINE

1. TIMESHARING among several programs --

- Separate context for each program
- OS loads appropriate context into pagemap when switching among pgms

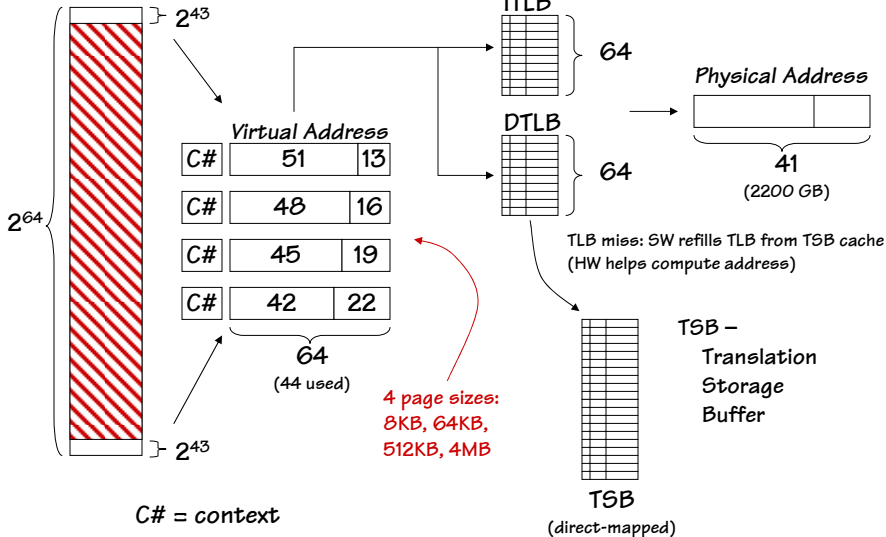
2. Separate context for OS "Kernel" (eg, interrupt handlers)...

- "Kernel" vs "User" contexts
- Switch to Kernel context on interrupt;
- Switch back on interrupt return.

HARDWARE SUPPORT: 2 HW pagemaps

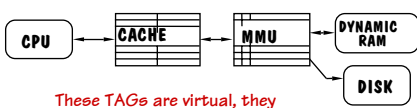
Example: UltraSPARC II MMU

Huge 64-bit address space (only 44-bits implemented)



Using Caches with Virtual Memory

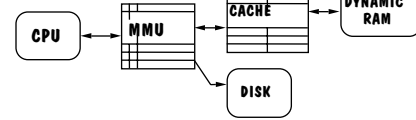
Virtual Cache
Tags match virtual addresses



These TAGs are virtual, they represent addresses before translation.

- Problem: cache invalid after context switch
- FAST: No MMU time on HIT

Physical Cache
Tags match physical addresses



These TAGs are physical, they hold addresses after translation.

- Avoids stale cache data after context switch
- SLOW: MMU time on HIT

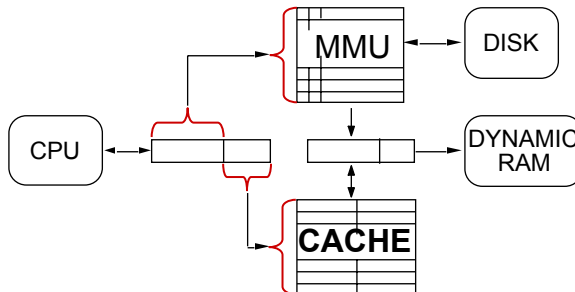
Counter intuitively perhaps, physically addressed Caches are the trend, because they better support parallel processing

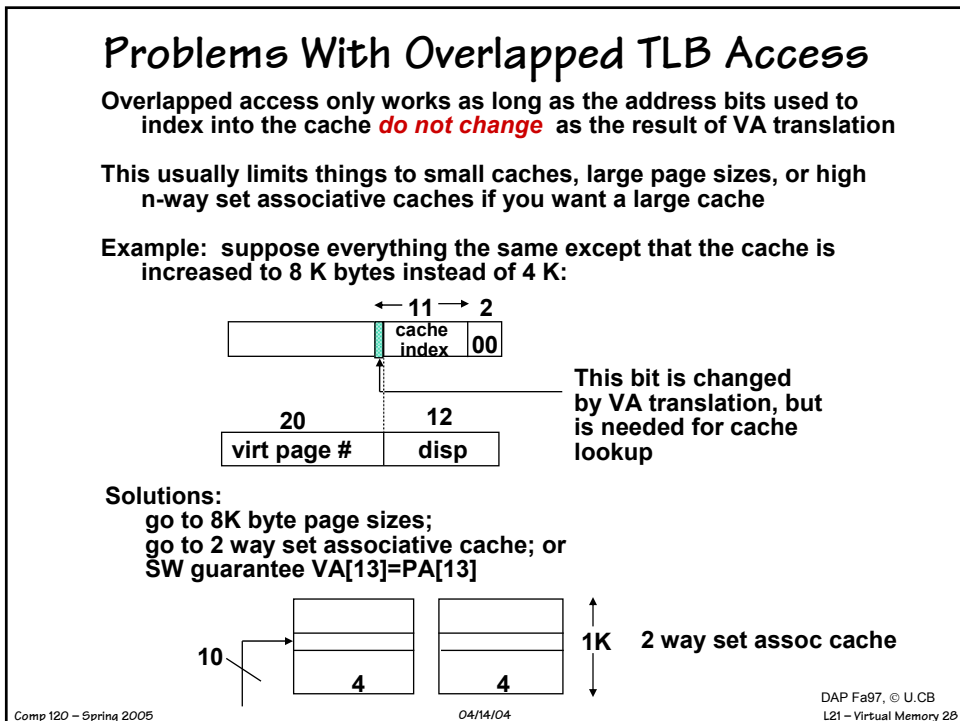
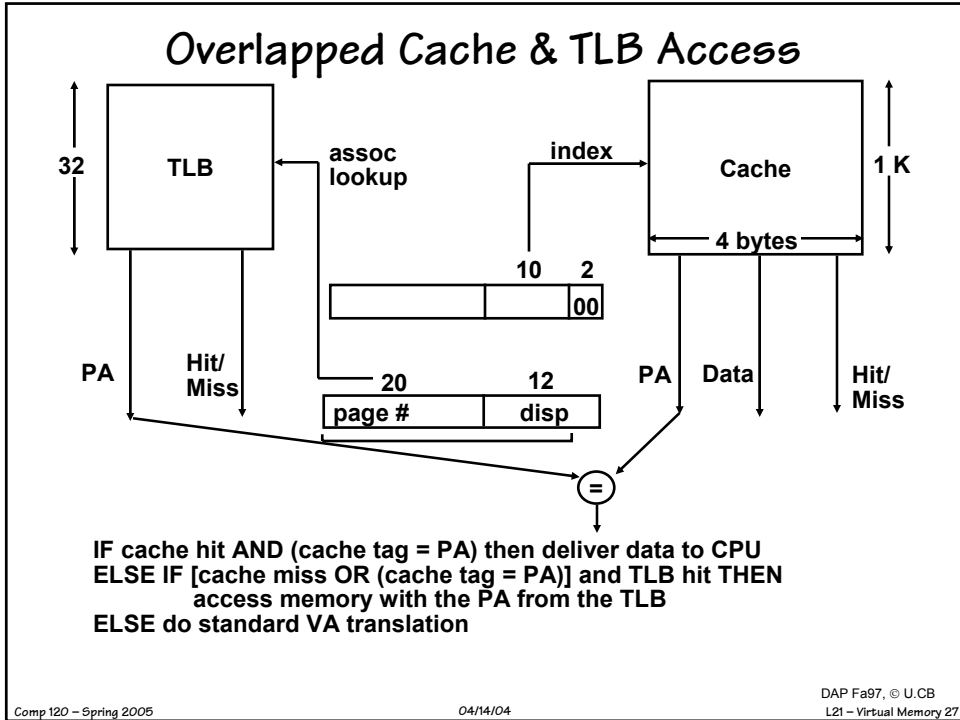
Reducing Translation Time

Machines with TLBs go one step further to reduce # cycles/cache access

They overlap the cache access with the TLB access

Works because high order bits of the VA are used to look in the TLB, while low order bits are used as index into cache





Memory hierarchy summary (1/4)

The Principle of Locality:

Program likely to access a relatively small portion of the address space at any instant of time.

Temporal Locality: Locality in Time

Spatial Locality: Locality in Space

Three Major Categories of Cache Misses:

Compulsory Misses: sad facts of life. Example: cold start.

Conflict Misses: increase cache size and/or associativity.

Nightmare Scenario: ping pong effect!

Capacity Misses: increase cache size

Cache Design Space

total size, block size, associativity

replacement policy

write-hit policy (write-through, write-back)

write-miss policy

Comp 120 – Spring 2005

04/14/04

DAP Fa97, © U.CB

L21 – Virtual Memory 29

Memory hierarchy summary (2/4)

Several interacting dimensions

cache size

block size

associativity

replacement policy

write-through vs write-back

write allocation

The optimal choice is a compromise

depends on access characteristics

workload

use (I-cache, D-cache, TLB)

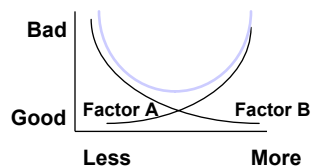
depends on technology / cost

Simplicity often wins

Cache Size

Associativity

Block Size



Comp 120 – Spring 2005

04/14/04

DAP Fa97, © U.CB

L21 – Virtual Memory 30

Memory hierarchy summary (3/4)

Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:

1. Where can block be placed?
2. How is block found?
3. What block is replaced on miss?
4. How are writes handled?

Page tables map virtual address to physical address

TLBs are important for fast translation

TLB misses are significant in processor performance: (funny times, as most systems can't access all of 2nd level cache without TLB misses!)

Memory hierarchy summary (4/4)

Virtual memory was controversial at the time:

can SW automatically manage 64KB across many programs?

1000X DRAM growth removed the controversy

Today VM allows many processes to share single memory without having to swap all processes to disk; VM protection is more important than memory hierarchy

Today CPU time is a function of (ops, cache misses) vs. just $f(\text{ops})$:

What does this mean to Compilers, Data structures, Algorithms?