EEL 5764: Graduate Computer Architecture

Appendix C – Memory Hierarchy Review
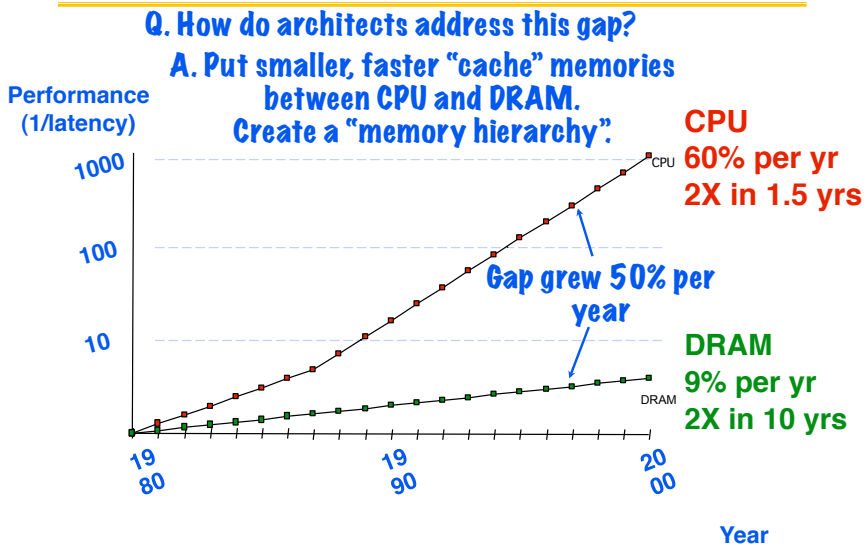
Ann Gordon-Ross
Electrical and Computer Engineering
University of Florida

http://www.ann.ece.ufl.edu/

These slides are provided by:
David Patterson
Electrical Engineering and Computer Sciences, University of California, Berkeley
Modifications/additions have been made from the originals

Since 1980, CPU has outpaced DRAM ...

Q. How do architects address this gap?

A. Put smaller, faster "cache" memories between CPU and DRAM. Create a "memory hierarchy".

Performance (1/latency)

CPU
60% per yr
2X in 1.5 yrs

Gap grew 50% per year

DRAM
9% per yr
2X in 10 yrs

1000

100

10

1980   1990   2000

Year

1977: DRAM faster than microprocessors

Apple][(1977)

CPU: 1000 ns
DRAM: 400 ns

Steve Jobs   Steve Wozniak

RAM Complement	Apple II System
4K	$ 1,298.00
48K	2,638.00

TIMING: 6502 PROCESSOR'S Φ₁ CLOCK SHOWING WHEN AND BY WHOM MEMORY IS ACCESSED.

Levels of the Memory Hierarchy

Capacity
Access Time
Cost

Upper Level

Staging
Xfer Unit

faster

CPU Registers
100s Bytes
<10s ns

Registers

Instr. Operands

prog./compiler
1-8 bytes

Cache
K Bytes
10-100 ns
1-0.1 cents/bit

Cache

Blocks

cache cntl
8-128 bytes

Main Memory
M Bytes
200ns- 500ns
$.0001-.00001 cents /bit

Memory

Pages

OS
512-4K bytes

Disk
G Bytes, 10 ms
(10,000,000 ns)

$10^{-5} - 10^{-6}$ cents/bit

Disk

Files

user/operator
Mbytes

Tape
infinite
sec-min
10^{-8}

Tape

Larger

Lower Level

Memory Hierarchy: Apple iMac G5

Managed by compiler		Managed by hardware			Managed by OS, hardware, application	
07	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

iMac G5 1.6 GHz

Goal: Illusion of large, fast, cheap memory

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access

# iMac's PowerPC 970: All caches on-chip

L1 (64K Instruction)

Registers

(1K)

512K L2

L1 (32K Data)
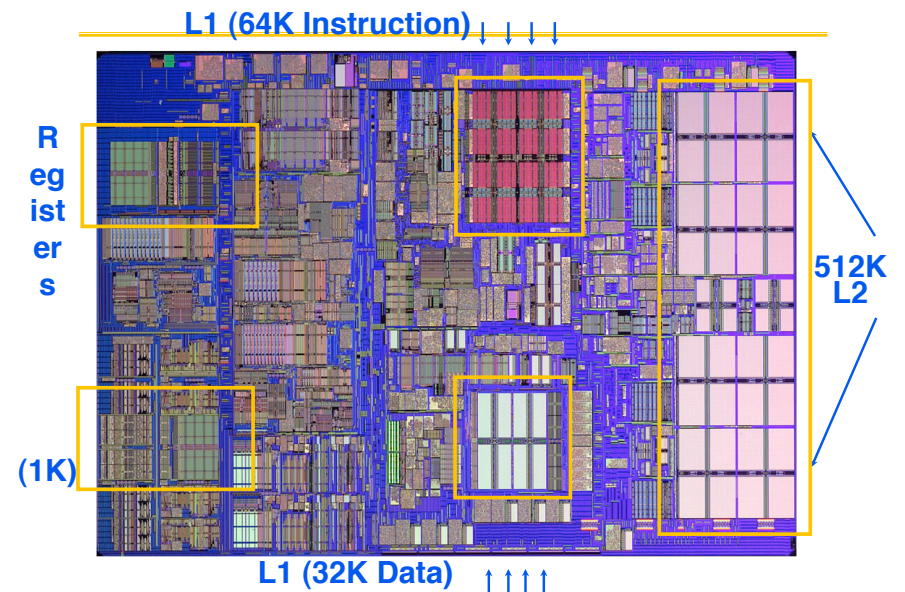
Why are Caches Here to Stay?? The Principle of Locality

- **The Principle of Locality:**
  - Program access a relatively small portion of the address space at any instant of time.
- **Two Different Types of Locality:**
  - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- **Last 15 years, HW relied on locality for speed**

It is a property of programs which is exploited in machine design.

Programs with locality cache well ...

Bad locality behavior

Temporal Locality

Spatial Locality

Memory Address (one dot per access)

Time
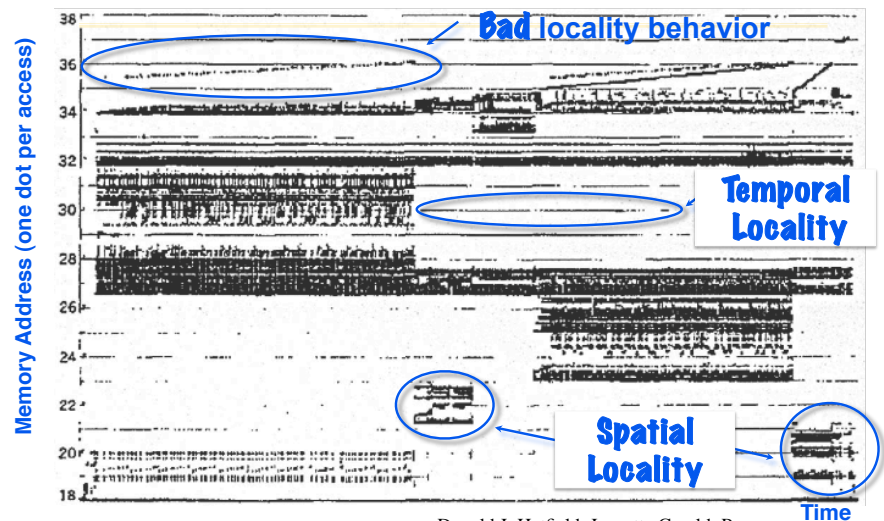
Paper made the case for virtual memory

Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Memory Hierarchy: Terminology

- **Hit**: data appears in some block in the upper level (example: Block X)
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- **Miss**: data needs to be retrieve from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level + Time to deliver the block the processor
- **Hit Time << Miss Penalty (500 instructions)**
  - May be better to recalculate results instead of refetching!
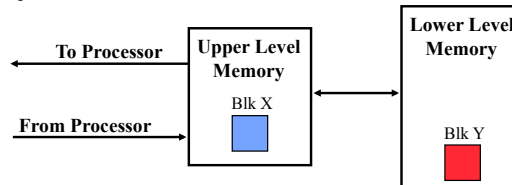
Cache Measures

- *Hit rate*: fraction found in that level
  - So high that usually talk about *Miss rate*
  - Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- Average memory-access time (better metric)
     = Hit time + Miss rate x Miss penalty
              (ns or clocks)
- *Miss penalty*: time to replace a block from lower level, including time to replace in CPU
  - *access time*: time to lower level
    = f(latency to lower level)
  - *transfer time*: time to transfer block
    =f(BW between upper & lower levels)

## 4 Questions for Memory Hierarchy

- **Q1: Where can a block be placed in the upper level?**
     (Block placement)

- **Q2: How is a block found if it is in the upper level?**
     (Block identification)

- **Q3: Which block should be replaced on a miss?**
     (Block replacement)

- **Q4: What happens on a write?**
     (Write strategy)

Q1: Where can a block be placed in the upper level?

- **Block 12 placed in 8 block cache:**
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets
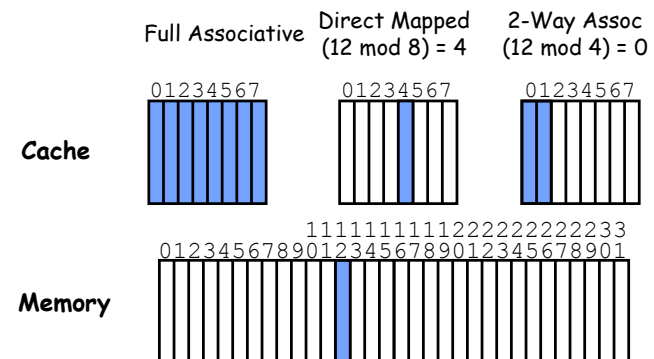
Q2: How is a block found if it is in the upper level?

- **Tag on each block**
  - No need to check index or block offset
- **Increasing associativity shrinks index, expands tag**

Block Address		Block Offset
Tag	Index	

Q3: Which block should be replaced on a miss?

- **Easy for Direct Mapped**
- **Set Associative or Fully Associative:**
  - Random
  - LRU (Least Recently Used)
    - » But more complex as associativity goes up

Assoc	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 kB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 kB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 kB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option – What happens on a write miss? Write allocate or write no-allocate

Write Buffers for Write-Through Caches

```
 ┌───────────┐      ┌────────┐      ┌────────┐
 │ Processor │◄────►│ Cache  │◄────►│ Lower  │
 │           │      │        │      │ Level  │
 │           │      │        │      │ Memory │
 └───────────┘      └────────┘      └────────┘
                    Write Buffer
```
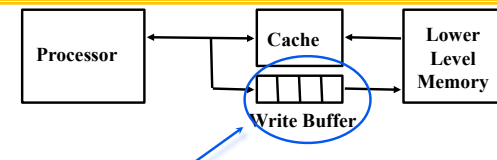
Holds data awaiting write-through to lower level memory

Q. Why a write buffer ? — **A. So CPU doesn't stall**

Q. Why a buffer, why not just one register ? — **A. Bursts of writes are common.**

Q. Are Read After Write (RAW) hazards an issue for write buffer? — **A. Yes! Drain buffer before next read, or send read 1st after check write buffers.**

5 Basic Cache Optimizations

- **Reducing Miss Rate**
1. **Larger Block size (compulsory misses)**
2. **Larger Cache size (capacity misses) – can affect hit time**
3. **Higher Associativity (conflict misses) – can affect hit time**

- **Reducing Miss Penalty**
4. **Multilevel Caches**

- **Reducing hit time**
5. **Giving Reads Priority over Writes**
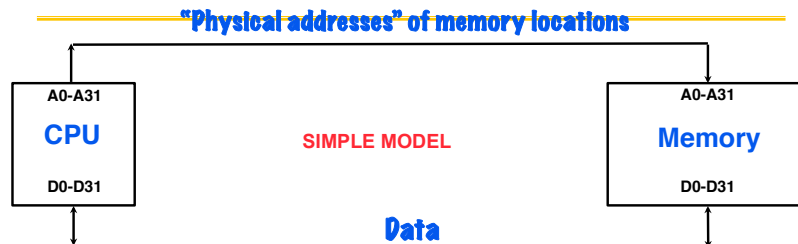   - **E.g., Read complete before earlier writes in write buffer**

Outline

- Memory hierarchy
- Locality
- Cache design
- **Virtual address spaces**
- **Page table layout**
- **TLB design options**

The Limits of Physical Addressing

"Physical addresses" of memory locations

A0-A31

CPU SIMPLE MODEL **Memory**
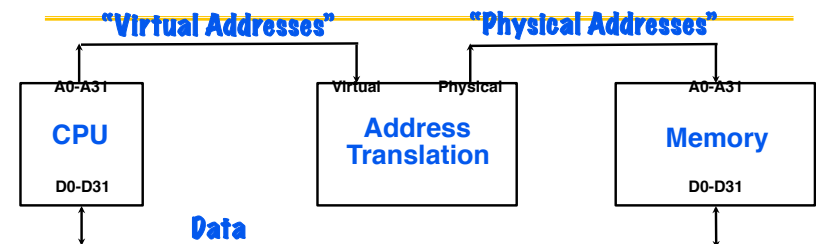
A0-A31

D0-D31 **D0-D31**

Data

All programs share one address space:
The physical address space

Machine language programs must be aware of the machine organization

No way to prevent a program from accessing any machine resource

Solution: Add a Layer of Indirection

"Virtual Addresses" *"Physical Addresses"*

A0-A31 Virtual Physical **A0-A31**

CPU **Address Translation** **Memory**

D0-D31 **D0-D31**

Data

User programs run in an 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

Three Advantages of Virtual Memory

- **Translation:**
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program ("Working Set") must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- **Protection:**
  - Different processes protected from each other.
  - Different pages can be given special behavior
    - » (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs
- **Sharing:**
  - Can map same physical page to multiple users ("Shared memory")
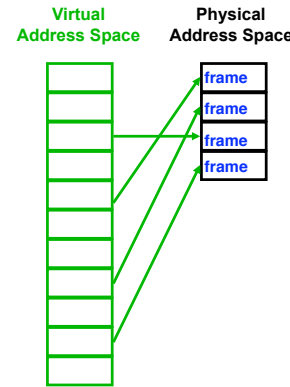
Page tables encode virtual address spaces

Virtual Address Space **Physical Address Space**
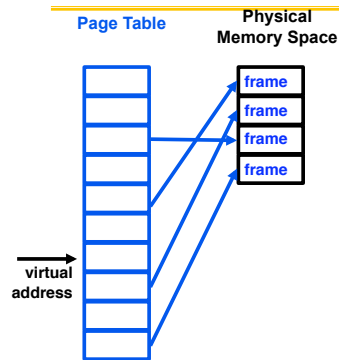
frame
frame
frame
frame

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

Page Table **Physical Memory Space**

frame
frame
frame
frame

virtual address

OS manages the page table for each ASID

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

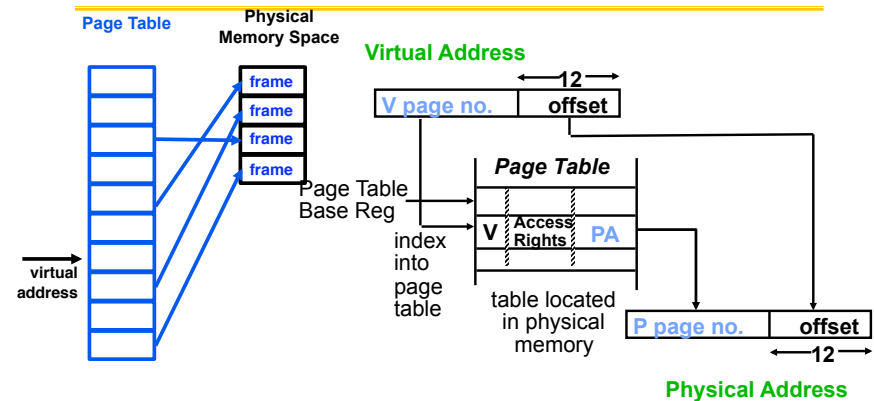A machine usually supports pages of a few sizes (MIPS R4000):

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

A 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 **Physical Memory Space**

frame
frame
frame
frame

virtual address

Virtual Address

| V page no. | offset |

←12→

Page Table Base Reg

index into page table

Page Table

V | Access Rights | PA

table located in physical memory

| P page no. | offset |

←12→

Physical Address

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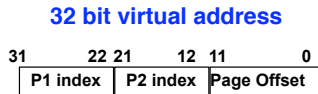- Virtual memory => treat memory ≈ cache for disk

Page tables may not fit in memory!

A table for 4KB pages for a 32-bit address space has 1M entries

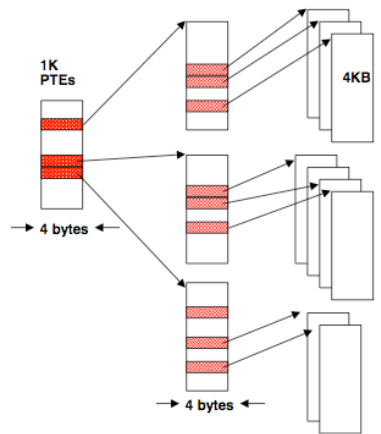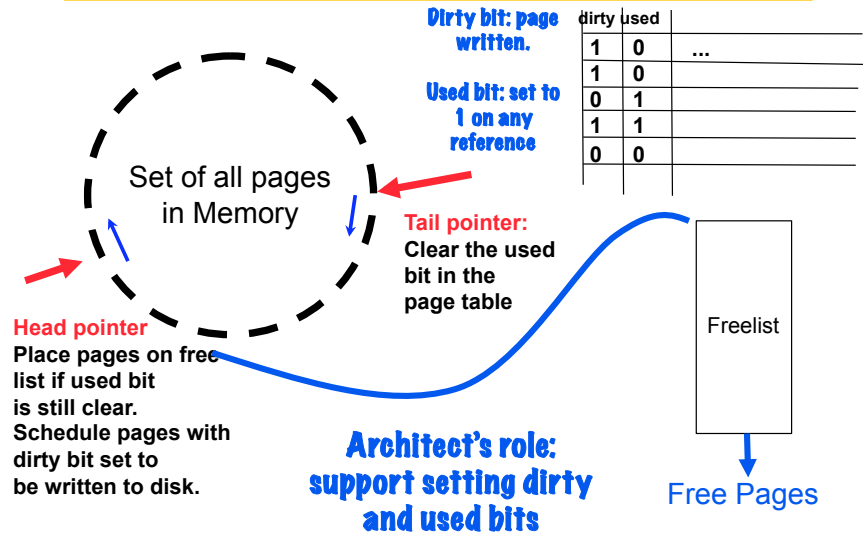**Each process needs its own address space!**

Two-level Page Tables

32 bit virtual address

31	22 21	12 11	0
P1 index	P2 index	Page Offset	

Top-level table wired in main memory

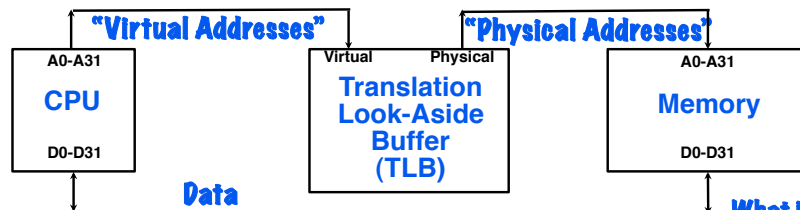**Subset of 1024 second-level tables in main memory; rest are on disk or unallocated**

1K PTEs

4KB

← 4 bytes →

← 4 bytes →

VM and Disk: Page replacement policy

Page Table

Dirty bit: page written.

Used bit: set to 1 on any reference

dirty	used	
1	0	...
1	0	
0	1	
1	1	
0	0	

Set of all pages in Memory

Tail pointer: Clear the used bit in the page table

Head pointer Place pages on free list if used bit is still clear. Schedule pages with dirty bit set to be written to disk.

Freelist

Free Pages

Architect's role: support setting dirty and used bits

MIPS Address Translation: How does it work?

Each memory access may take twice as long – page fault and data fetch!

"Virtual Addresses"

"Physical Addresses"

A0-A31	Virtual Physical	A0-A31
CPU	**Translation Look-Aside Buffer (TLB)**	**Memory**
D0-D31		D0-D31

Data

What is the table of mappings that it caches?

Translation Look-Aside Buffer (TLB)
A small fully-associative cache of mappings from virtual to physical addresses

TLB also contains protection bits for virtual address

Fast common case: Virtual address is in TLB, process has permission to read/write it.