Handout 4 – Memory Hierarchy

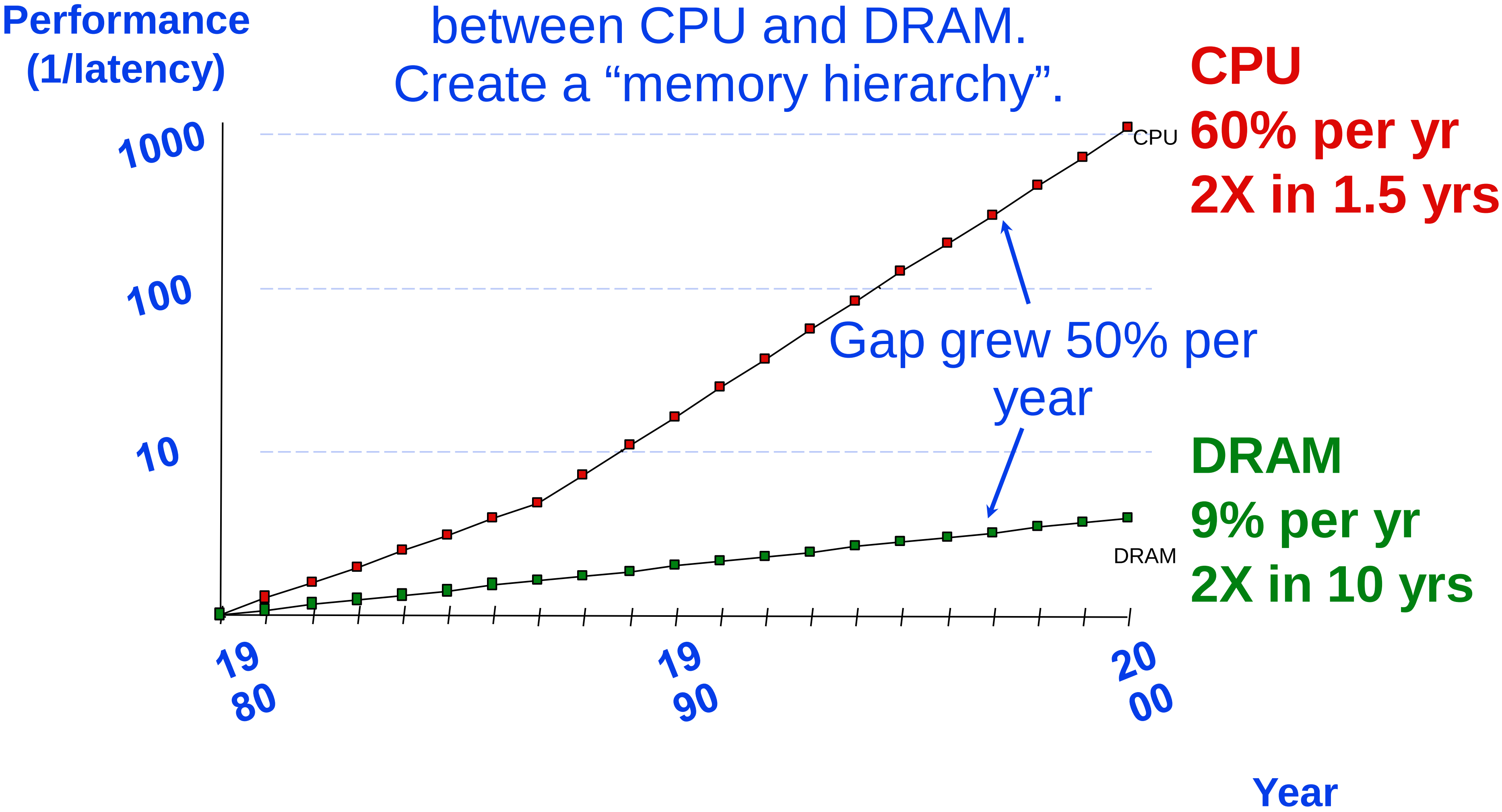
Outline

- Memory hierarchy
- Locality
- Cache design
- Virtual address spaces
- Page table layout
- TLB design options (MMU Sub-system)
- Conclusion

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".

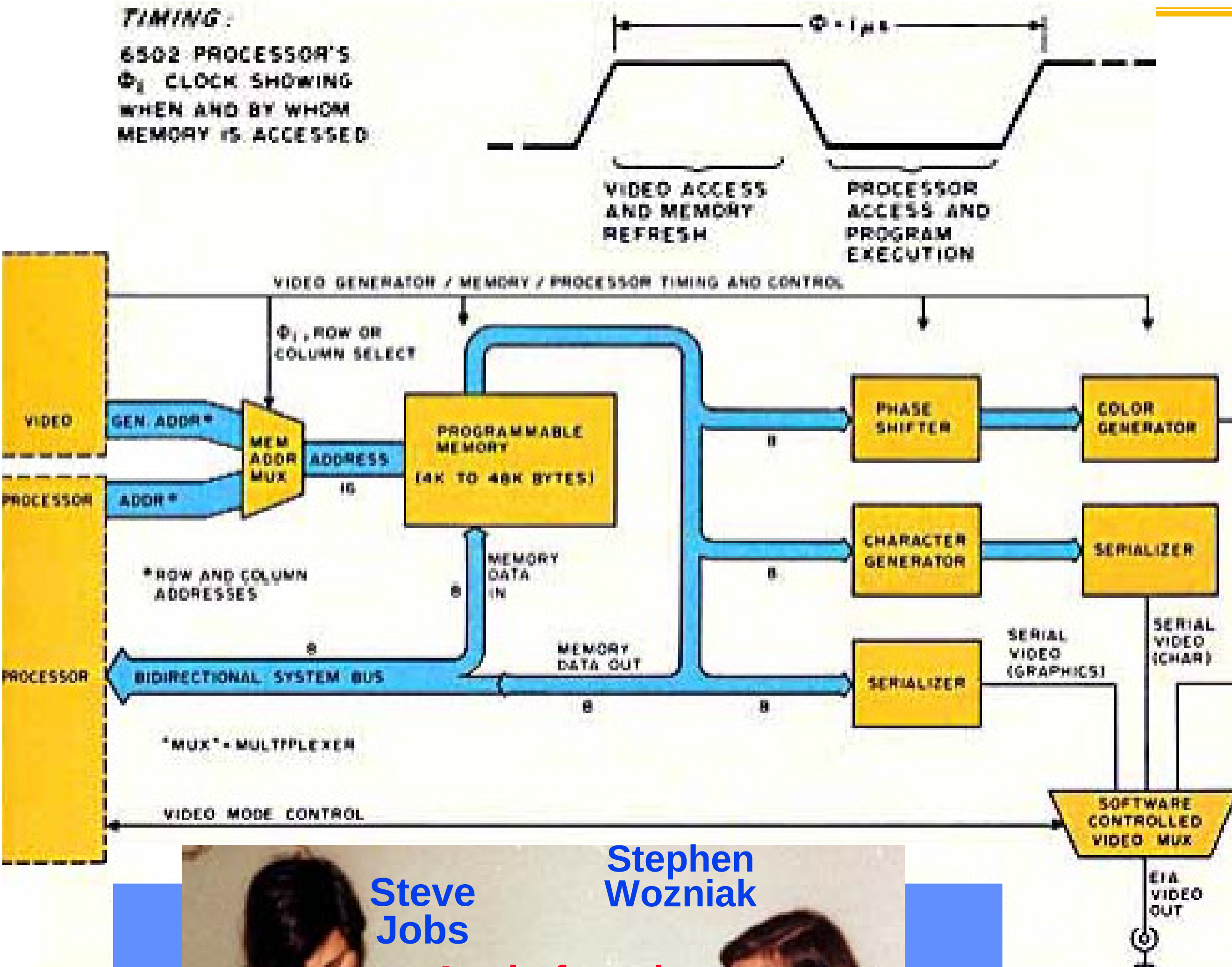


But in 1977: DRAM faster than microprocessors

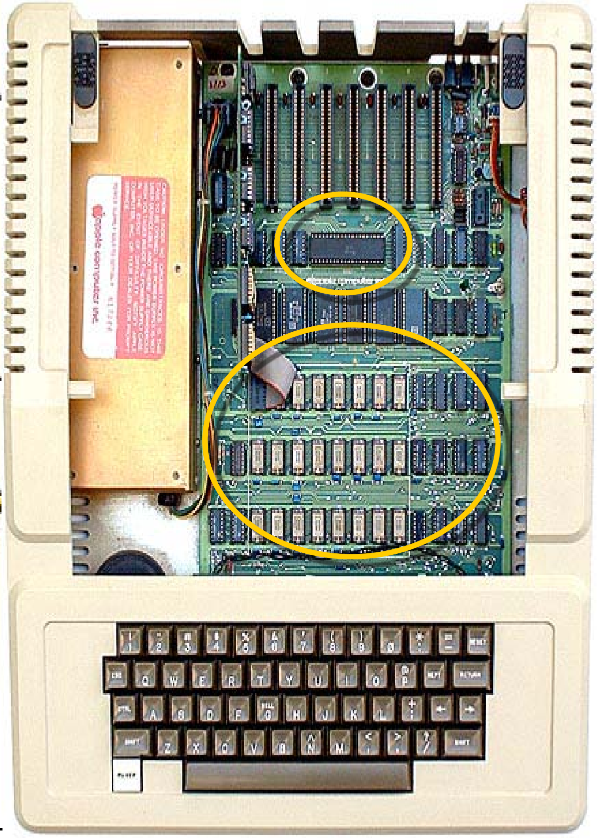
Apple][(1977)

CPU: 1000 ns

DRAM: 400 ns

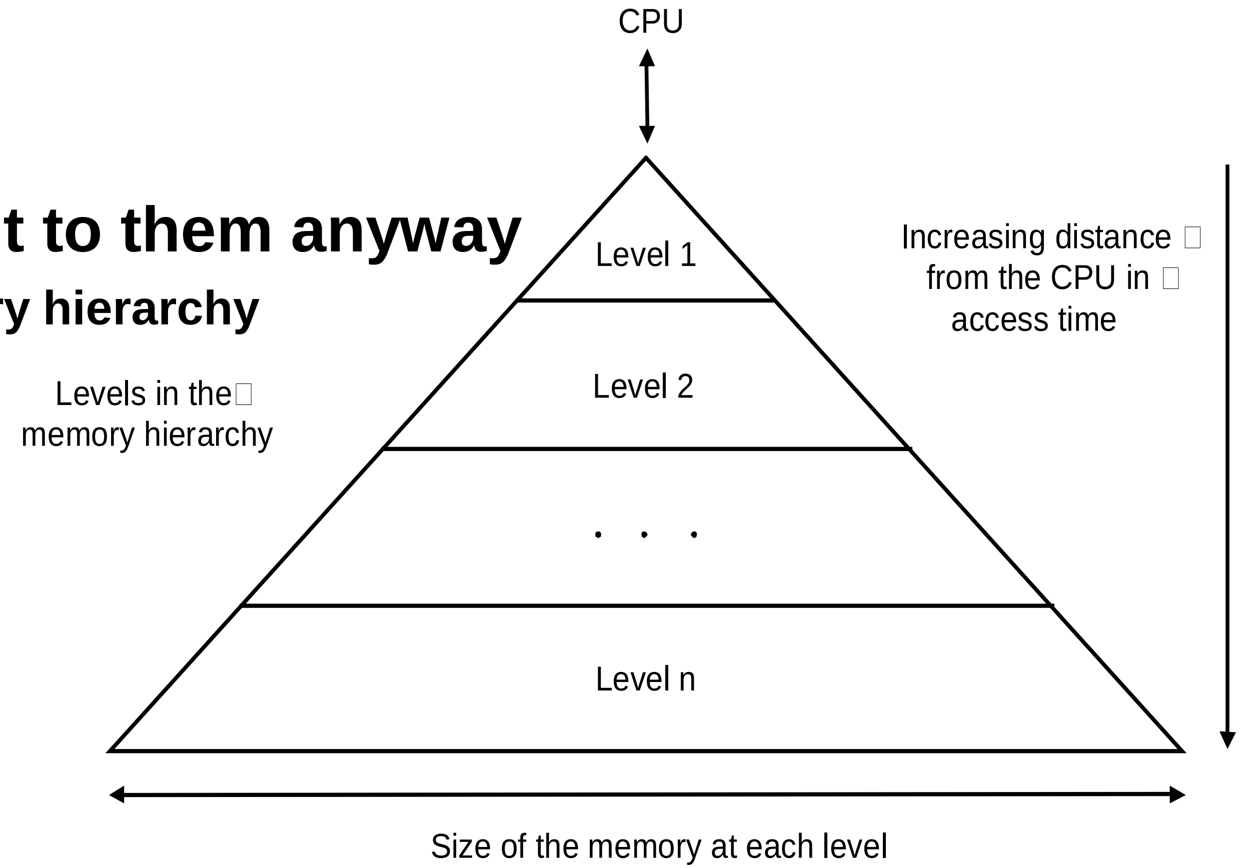


RAM Complement	Apple II System
4K	$ 1,298.00
48K	2,638.00

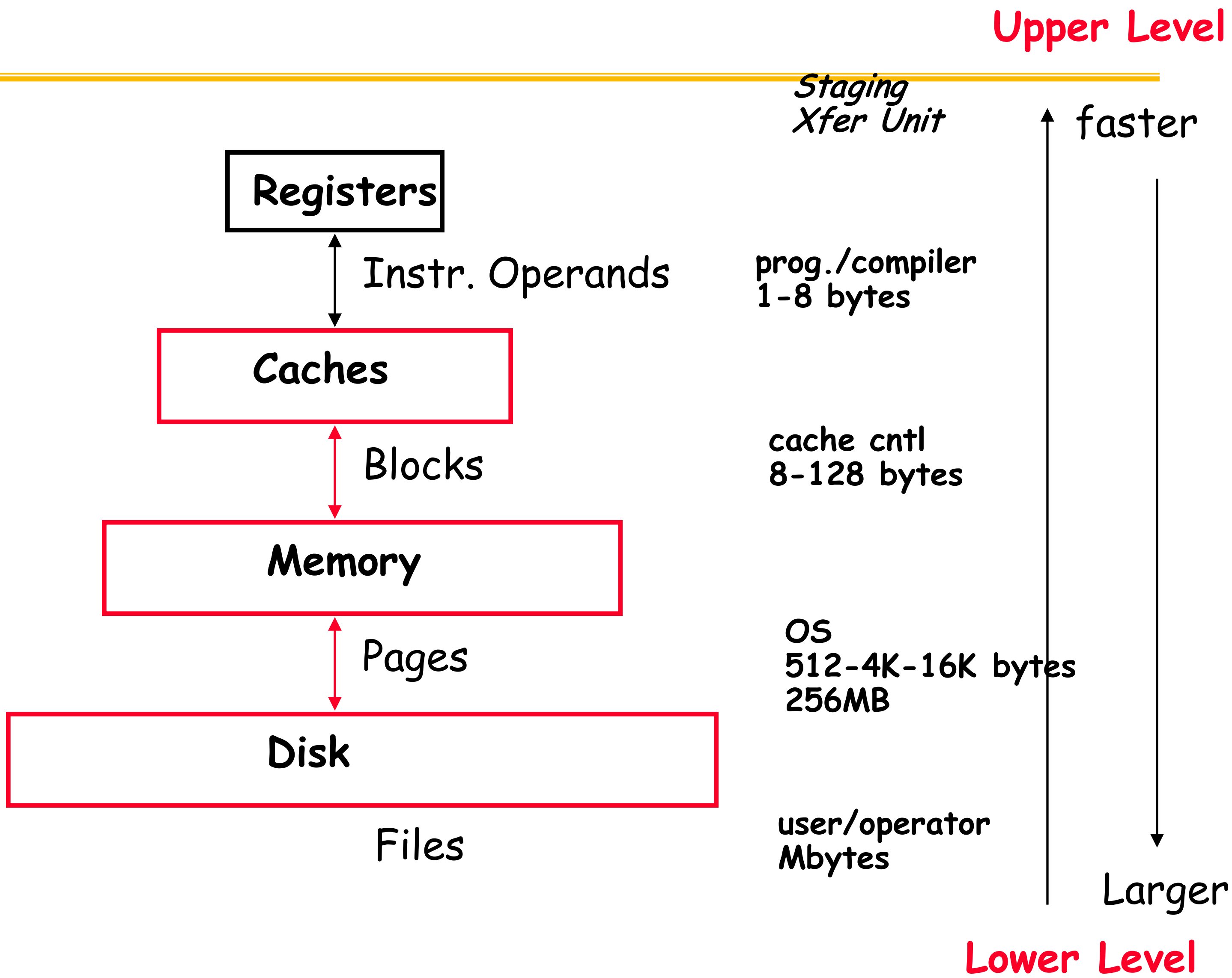


Exploiting Memory Hierarchy

- **Users want large and fast memories!**
  Flip-flops
  SRAM
- **DRAM**
  Disk
- **Try and give it to them anyway**
  - **build a memory hierarchy**



Levels of the Memory Hierarchy



Memory Hierarchy: Apple iMac G5

Managed by compiler → Reg

Managed by hardware → L1 Inst, L1 Data, L2, DRAM

Managed by OS, hardware, application → Disk

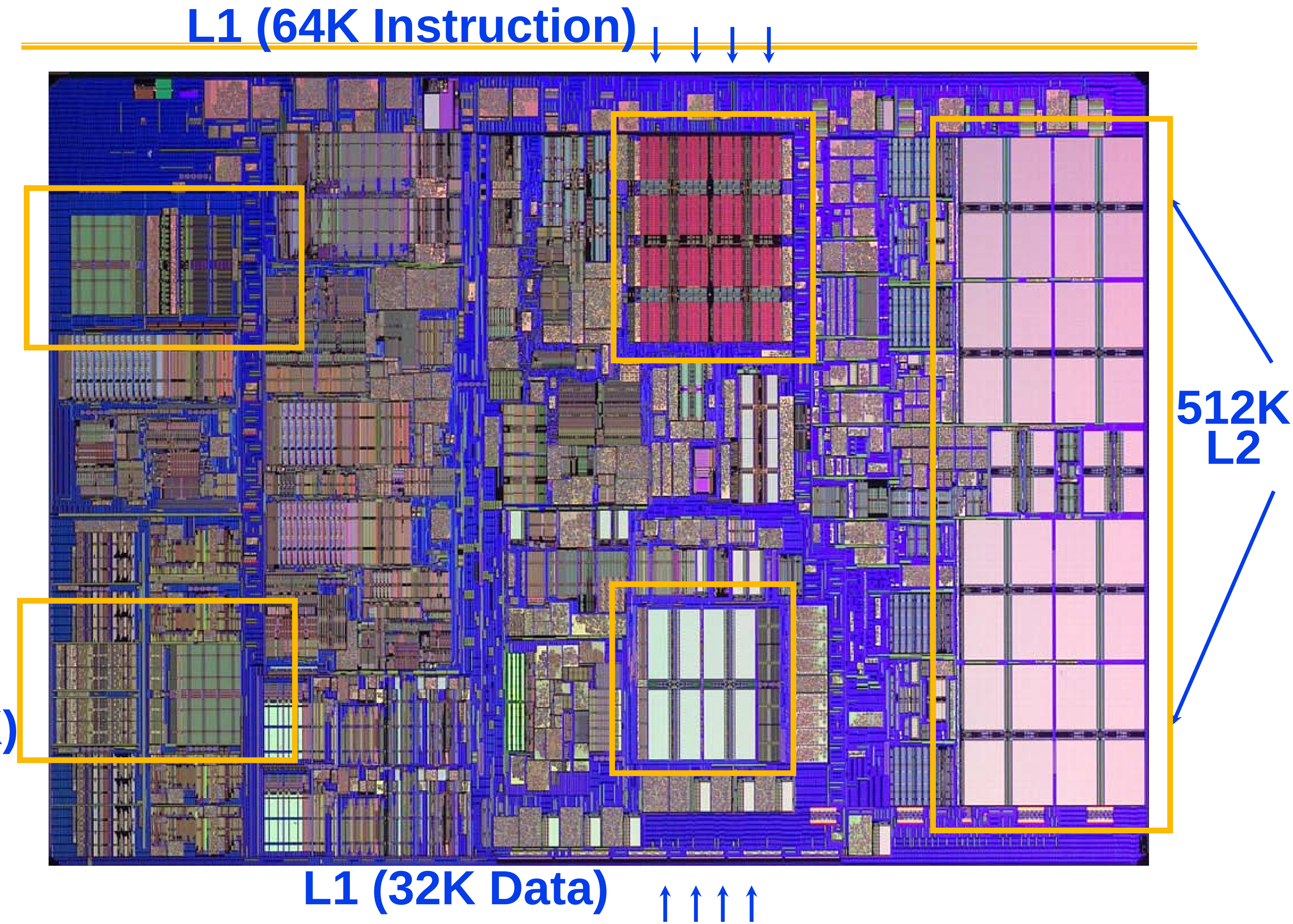
07	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

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



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.

Hits vs. Misses

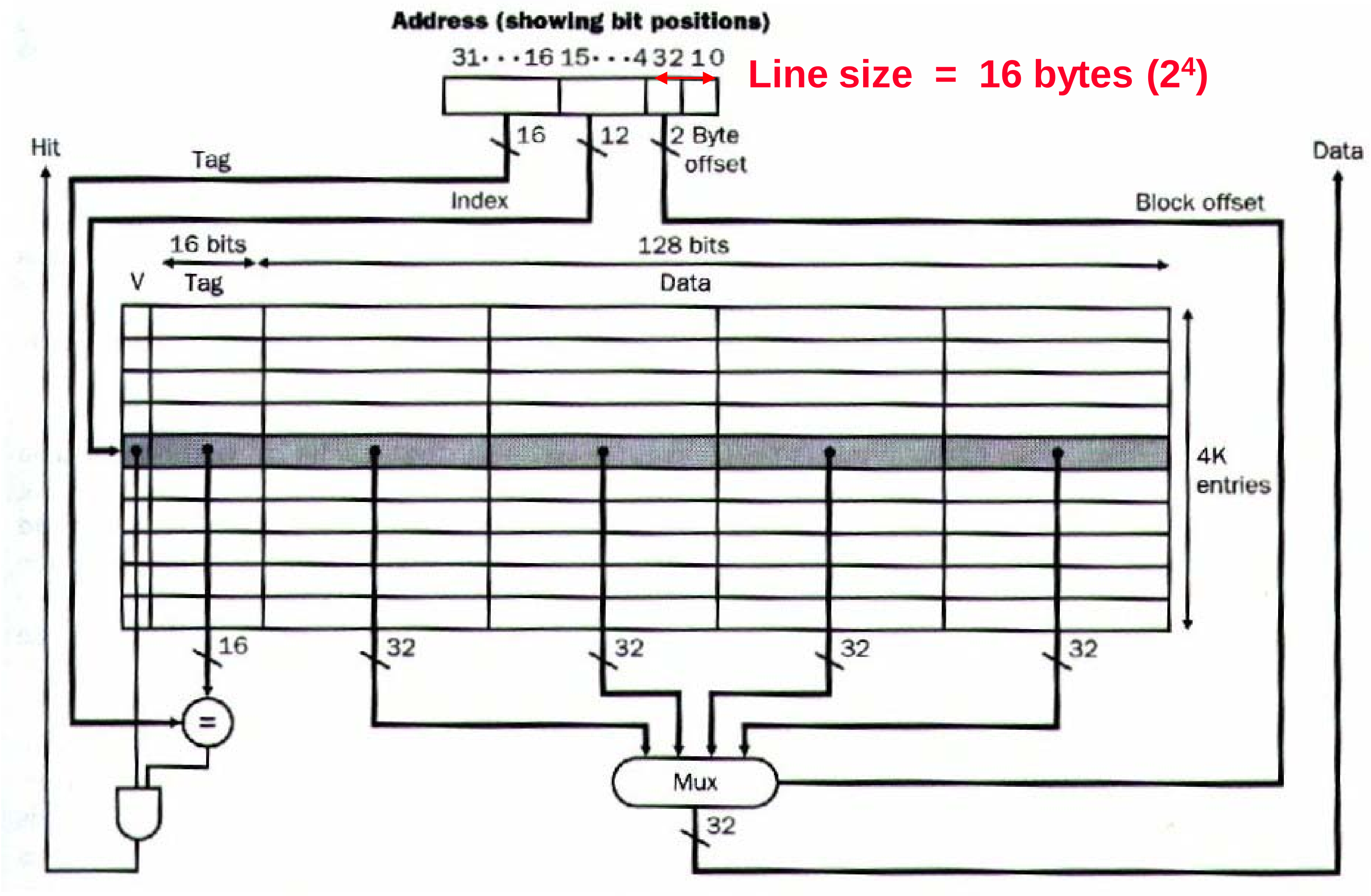
- **Misses: compulsory misses (cold miss), capacity misses, conflict misses.**
- **Read hits**
  - **this is what we want!**
- **Read misses**
  - **stall the CPU, fetch block from memory, deliver to cache, restart**
- **Write hits:**
  - **can replace data in cache and memory (write-through)**
  - **write the data only into the cache (write-back the cache later)**
- **Write misses:**
  - **read the entire block into the cache, then write the word (write allocate)**
  - **or just write around the cache**

Write policy

- Write hit
  - Write-through (WT)
  - Write-back (WB)
- Write miss
  - Write allocate (or write allocation)
    - » **Read the missing block into cache first**
    - » **then WT or WB**
  - Write around
    - » **Write the data into the next level memory**

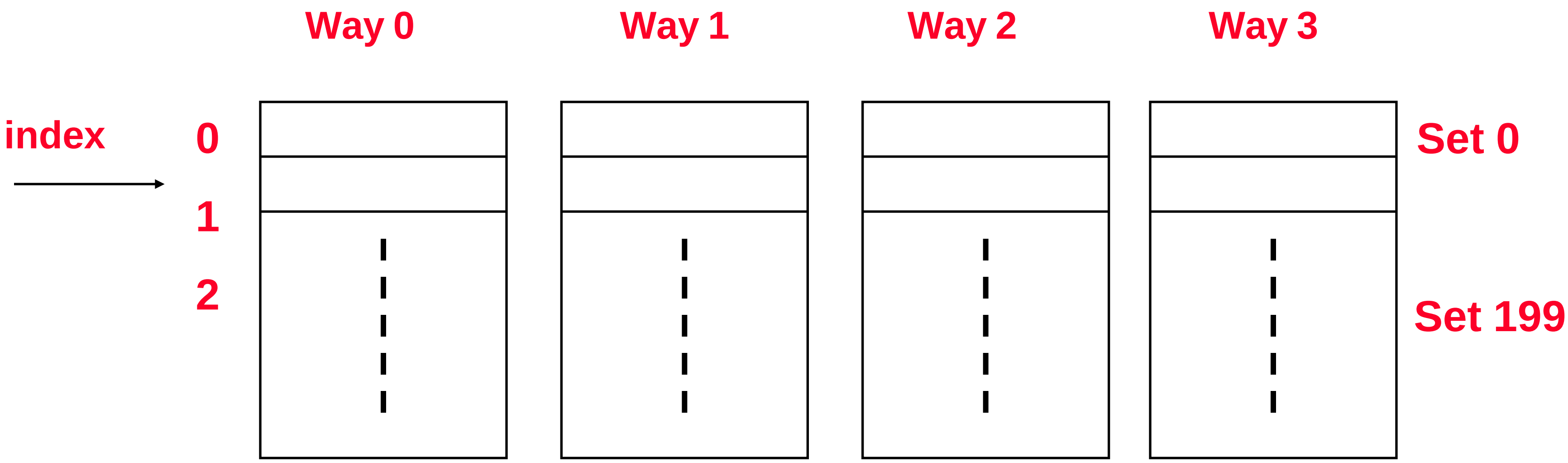
Example: A Direct Mapped Cache

- **Taking advantage of spatial locality: longer line size**



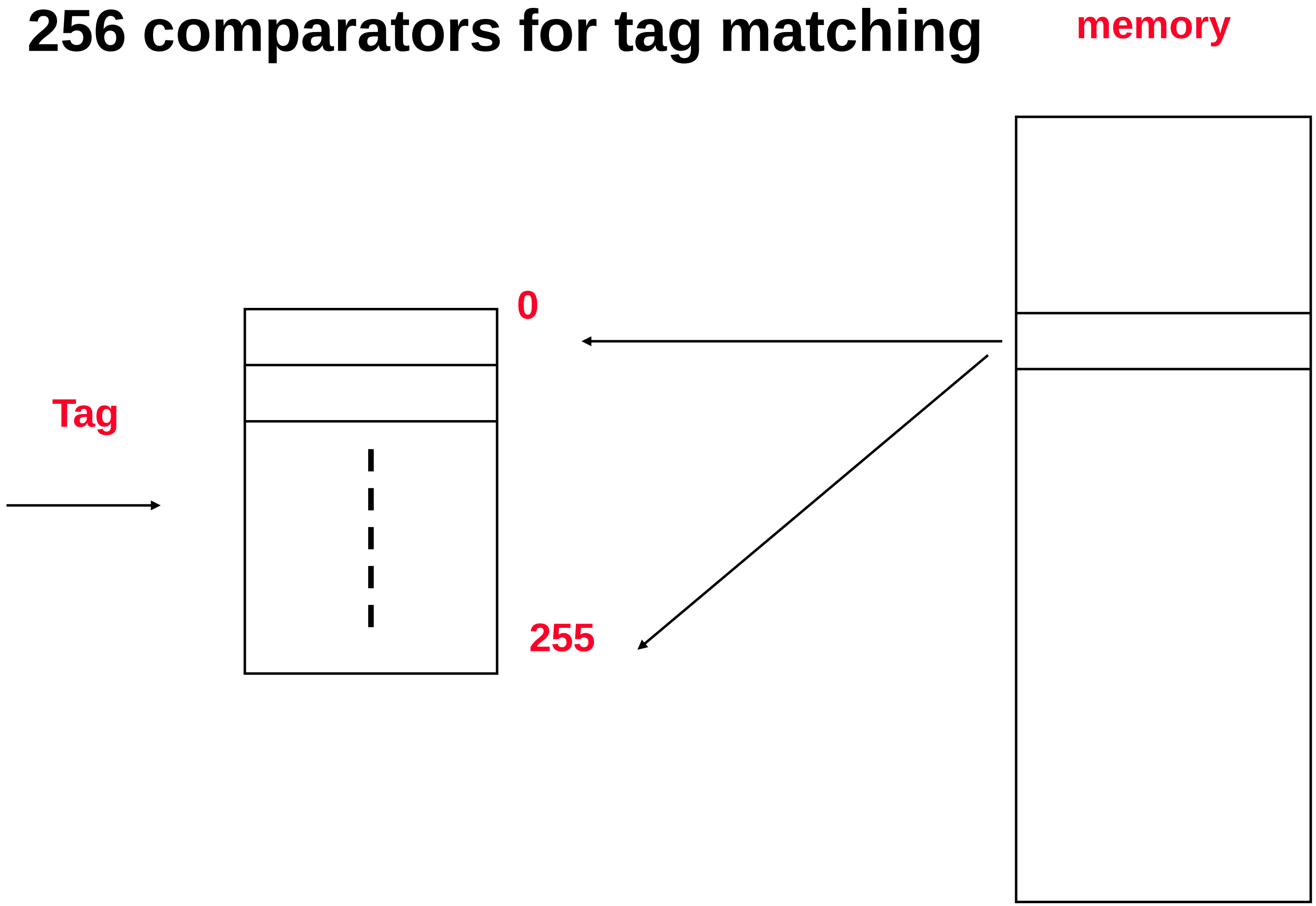
N-way set associative

- Tag::index::line size
- N direct mapped caches in parallel
- An index gets N blocks



Fully set associative

- **Tag::line size**
- **256 comparators for tag matching**



Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

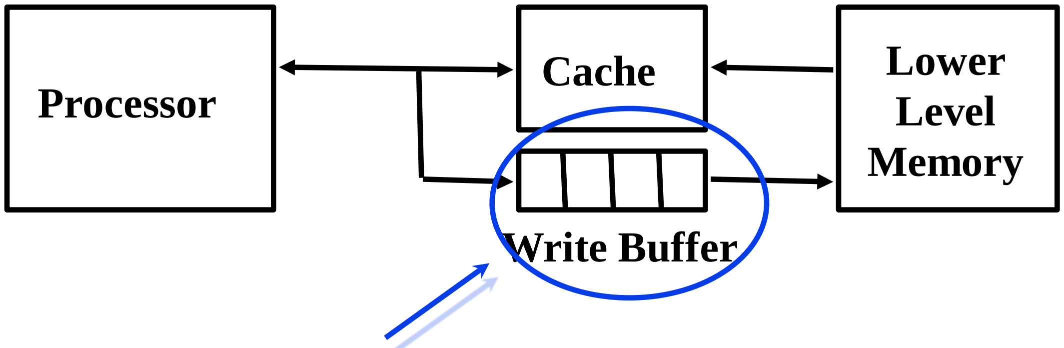
Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

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 Copy-back when replacing a dirty copy
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option -- let writes to an un-cached address allocate a new cache line ("write-allocate").

Write Buffers for Write-Through Caches



Holds data awaiting write-through to lower level memory

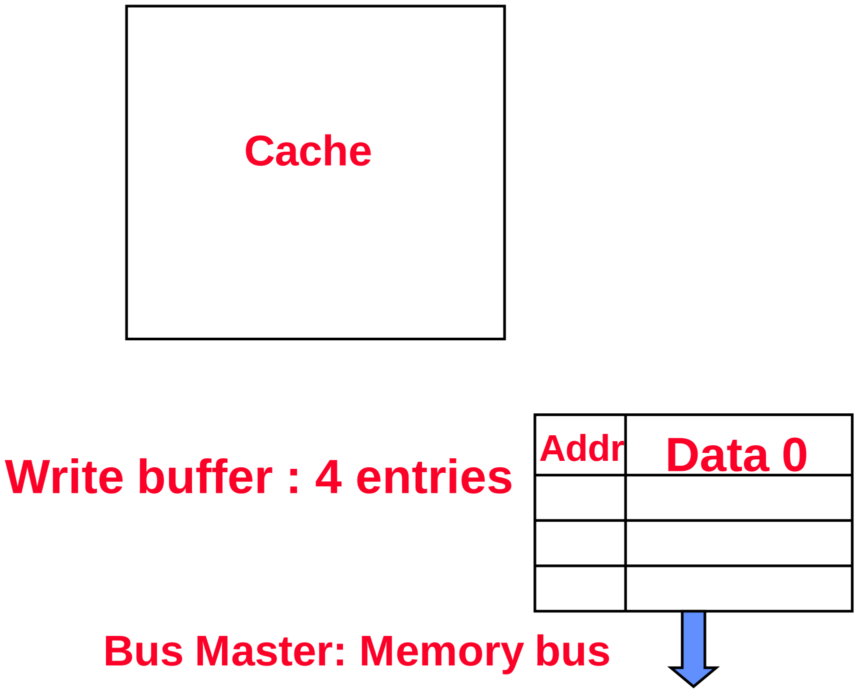
Q. Why a write buffer ? A. So CPU doesn't stall

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.

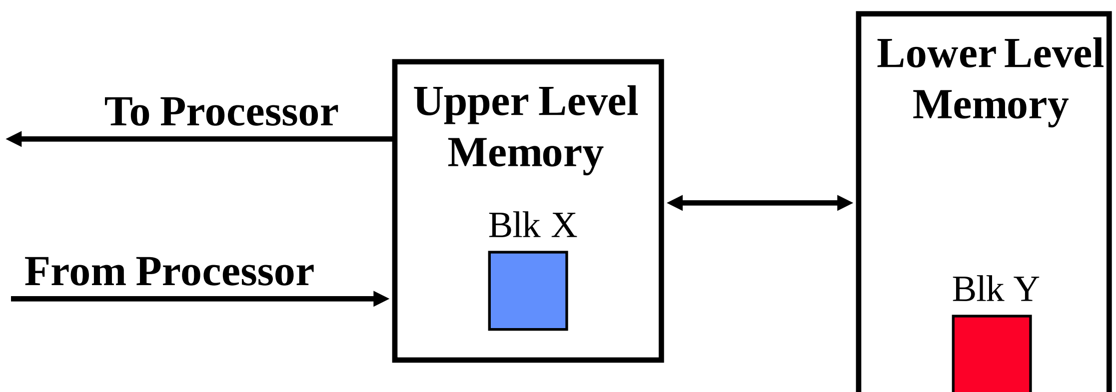
Advanced issue in write buffer

- **A buffer for**
  - Write through data into the next level memory or
  - for the replaced block due to write back
  - Read-bypassing write buffer
  - Exception handling: if write buffer still has data and an exception occurs.
  - Depth of write buffer



Hit time and Miss penalty

- **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 retrieved 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 on 21264!)**



Memory System Performance

- ***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**
  = Hit time + Miss rate x Miss penalty

- ***Miss penalty***: **cache line filling latency**
  - ***access time***: **time to lower level**
    = f(latency to lower level)
  - ***transfer time***: **time to transfer block**
    =f(BW between upper & lower levels)

5 Basic Cache Optimizations

- **Reducing Miss Rate**

1. **Larger Block size (compulsory misses)**
2. **Larger Cache size (capacity misses)**
3. **Higher Associativity (conflict misses)**

- **Reducing Miss Penalty**

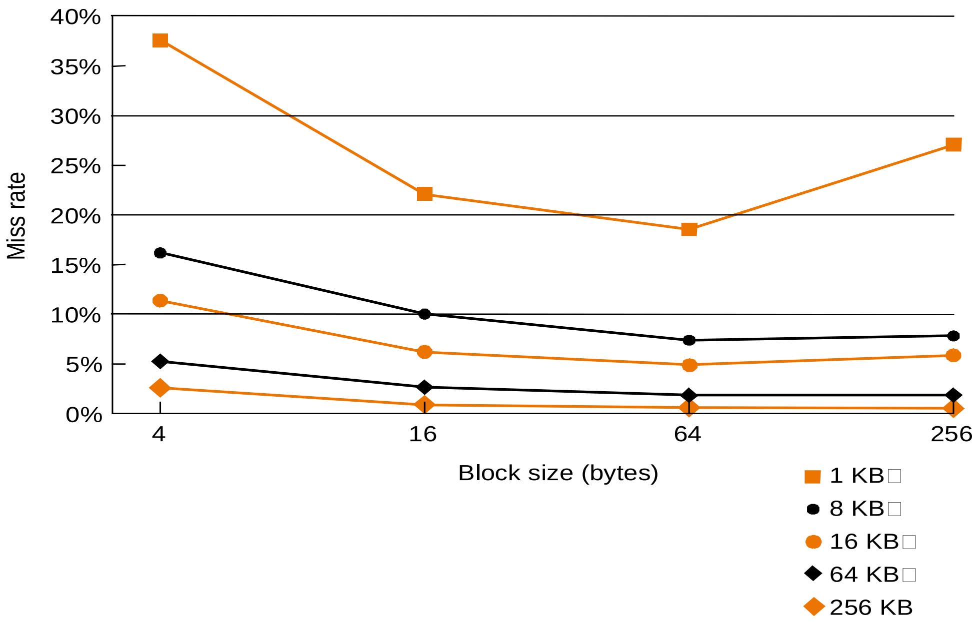
4. **Multilevel Caches**

- **Reducing hit time**

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

Line size and locality

- **Increasing the block size tends to decrease miss rate:**

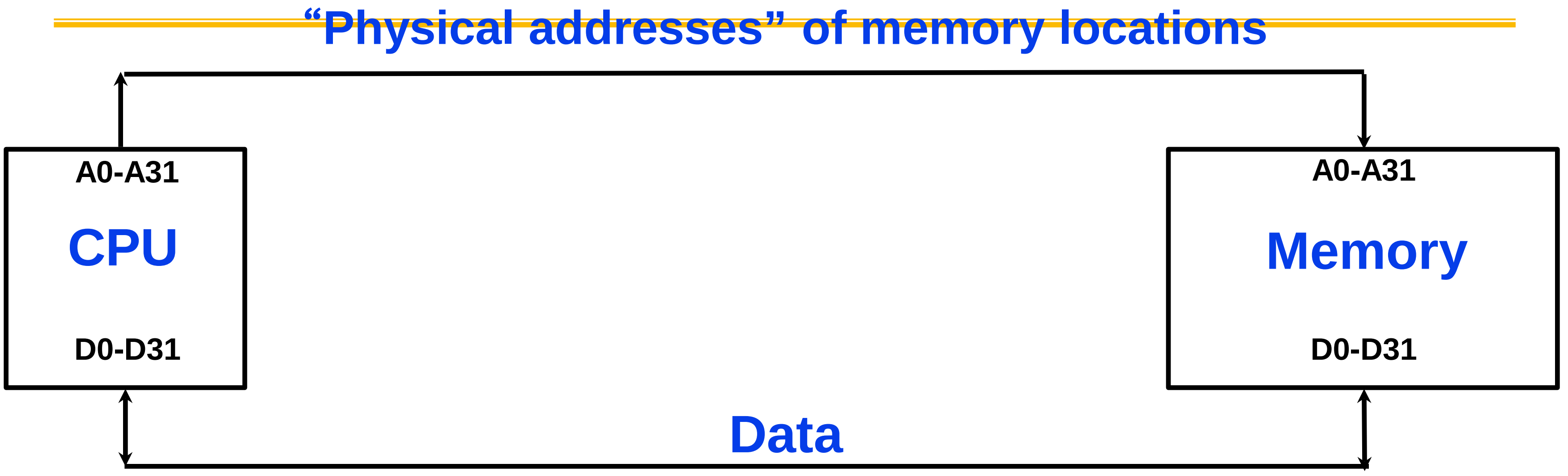


- **Use split caches because there is more spatial locality in code:**

Outline

- Review
- Redo Geomtric Mean, Standard Deviation
- Memory hierarchy
- Locality
- Cache design
- **Virtual address spaces**
- **Page table layout**
- **TLB design options**
- **Conclusion**

The Limits of Physical Addressing

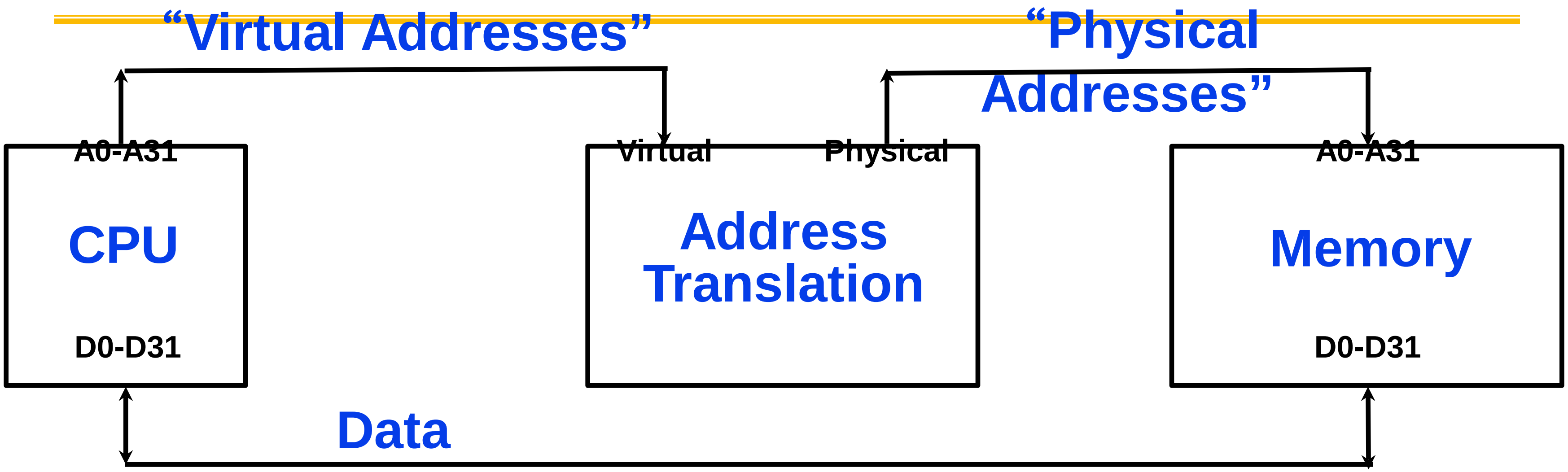


All programs share one address space: The physical address space

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



User programs run in an standardized virtual address space

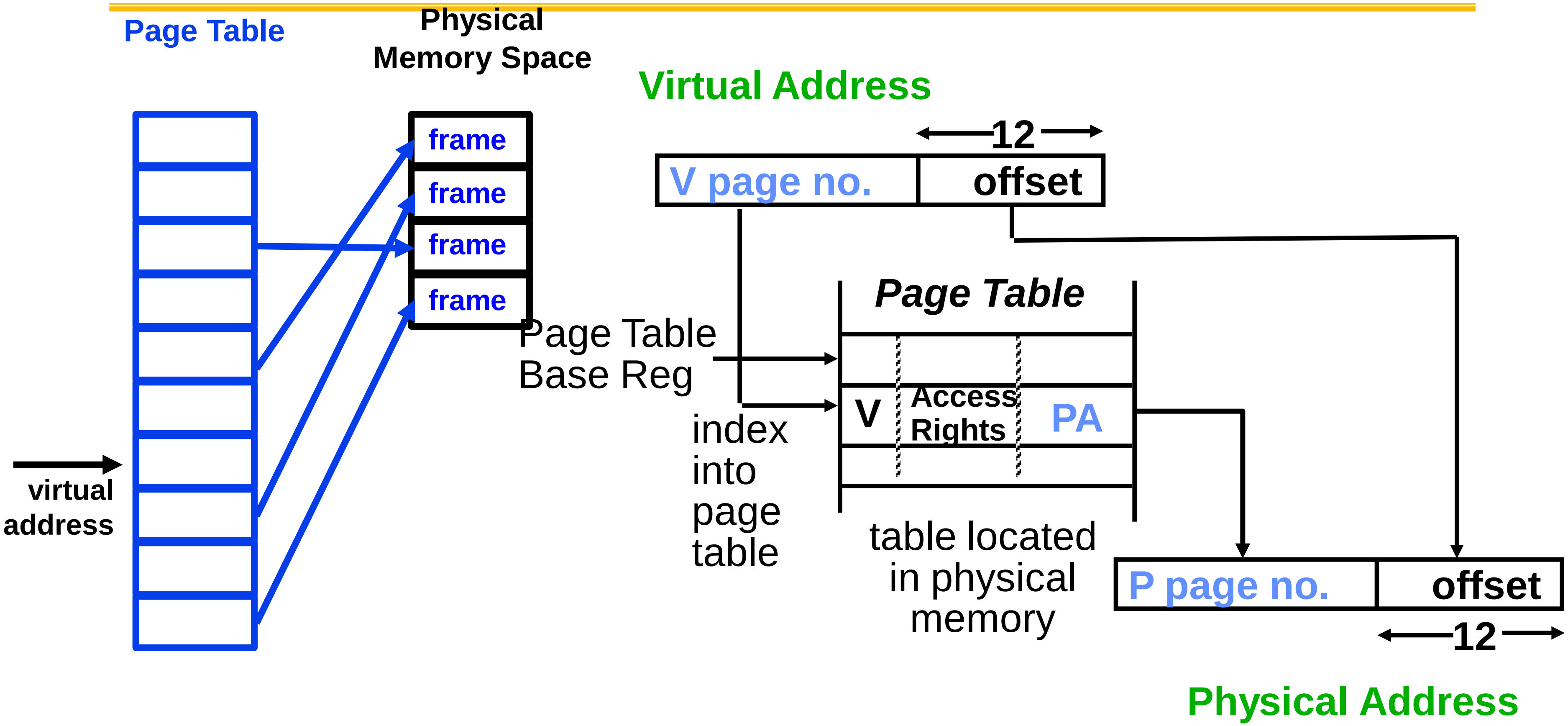
Address Translation hardware managed by the operating system (OS) maps virtual address to physical memory

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 threads (or 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”)

Details of Page Table



- **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!

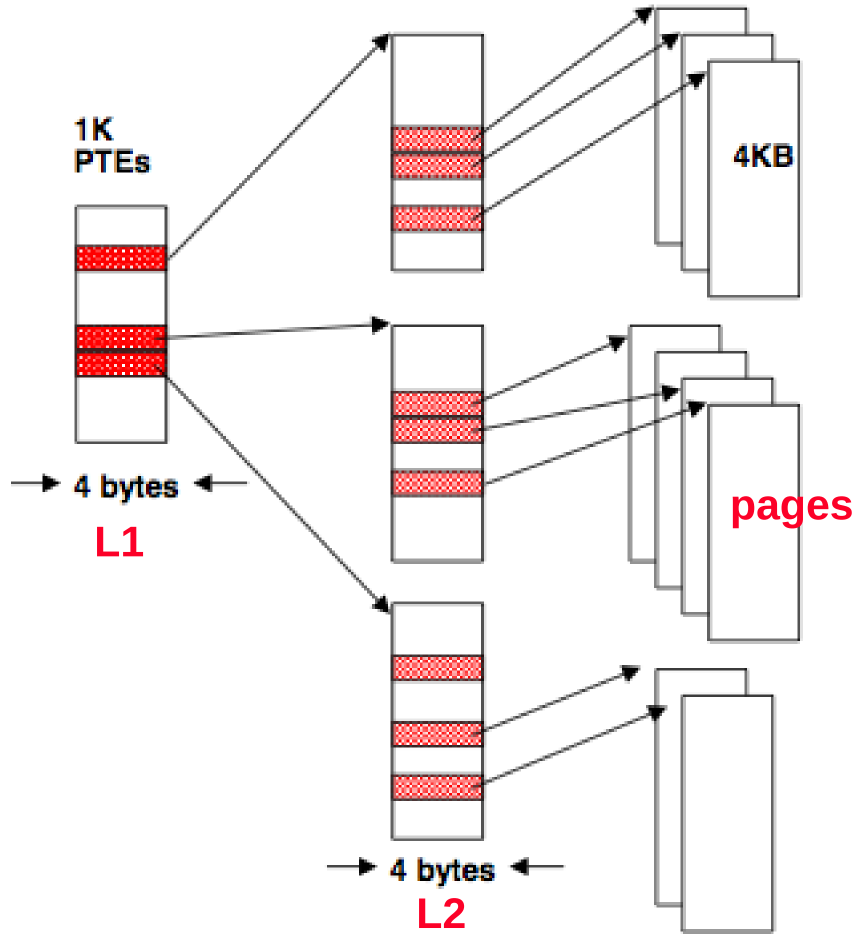
A table for 4KB pages for a 32-bit address space has 1M entries

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

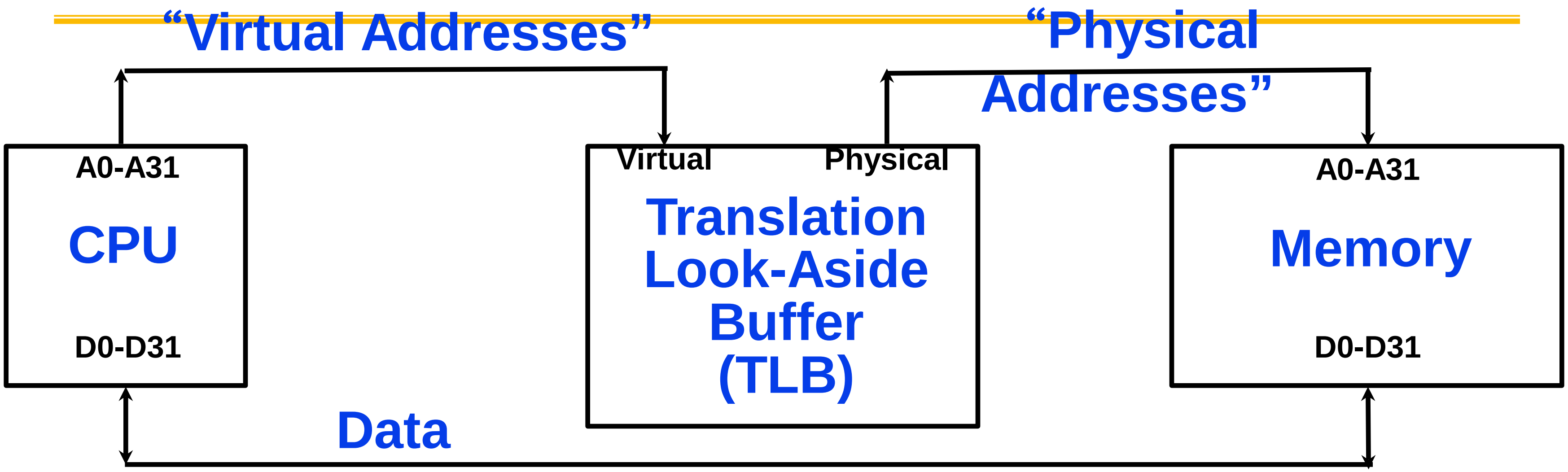
Subset of 1024 second-level tables in main memory; rest are on disk or unallocated

Virtual Memory System

- See the main memory as the cache of the disk storage system
- Features of this cache
  - Write back cache
  - Fully set associative

TLB Design Concepts

MIPS Address Translation: How does it work?



Translation Look-Aside Buffer (TLB)

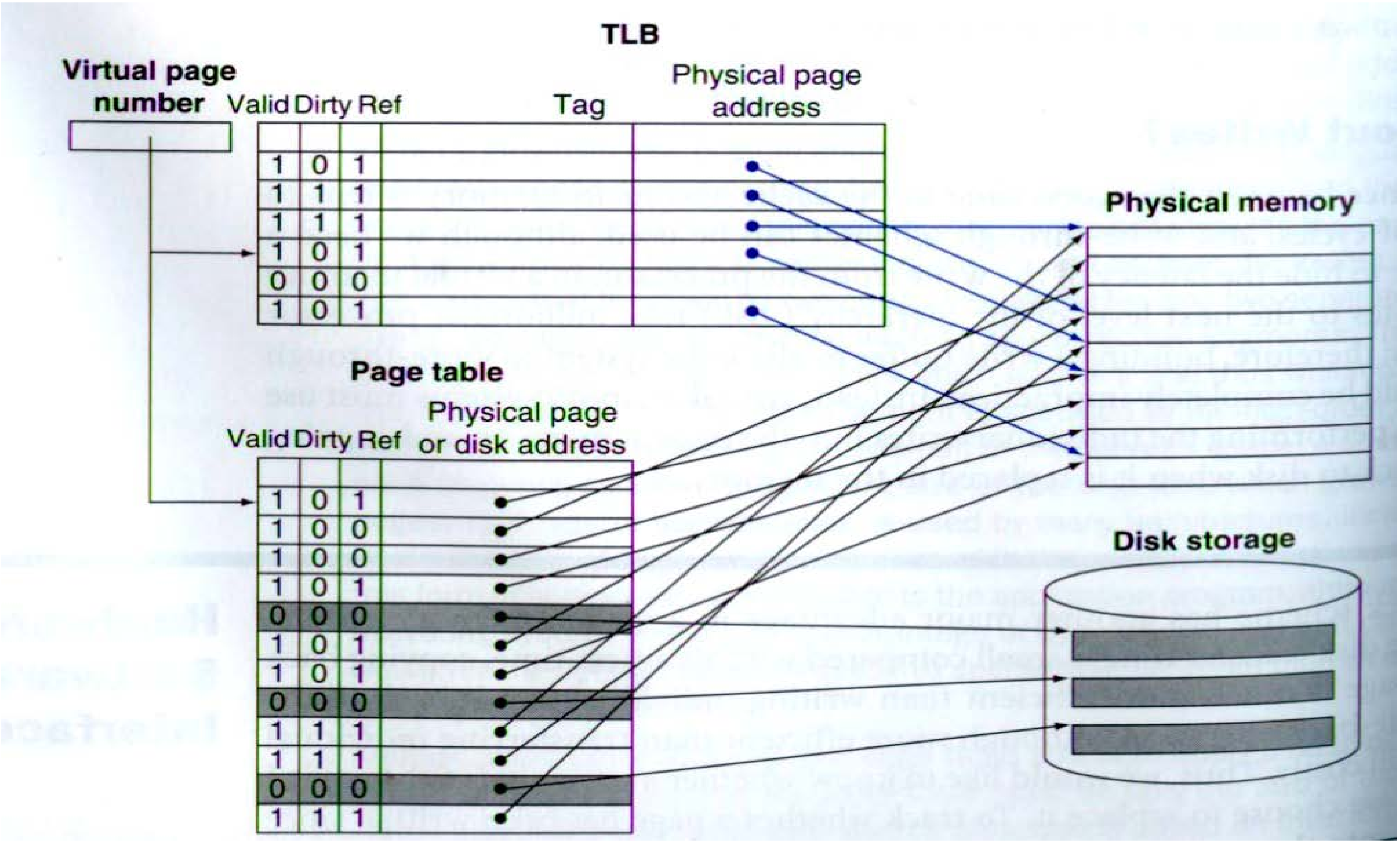
A small fully-associative cache of mappings from virtual to physical addresses

TLB also contains protection bits for virtual address

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

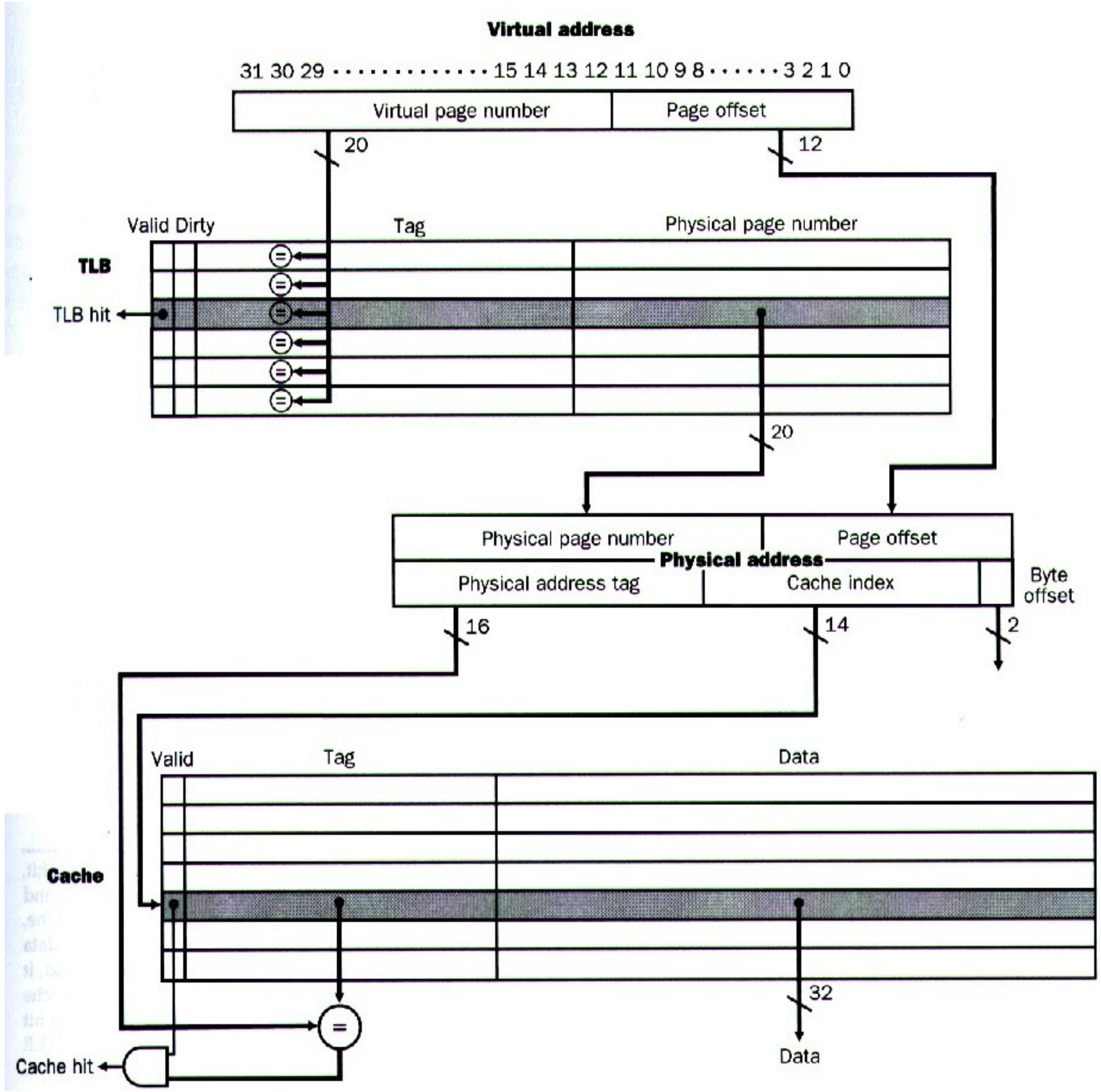
Making Address Translation Fast

- **A cache for address translations: translation look-aside buffer (TLB)**

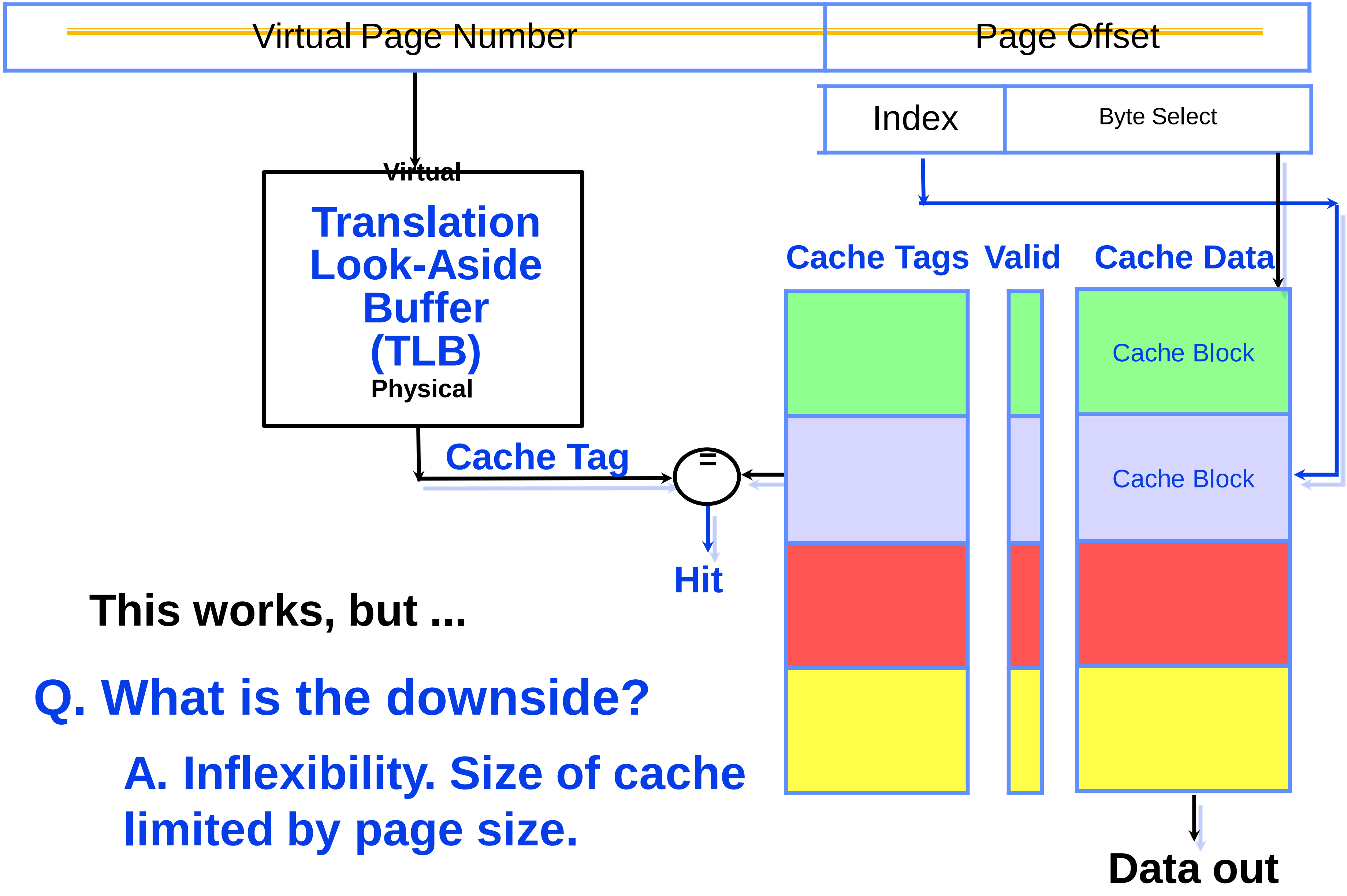


TLB and Translation: Physically addressed L1 cache, physically indexed

- TLB: cache for page table
- TLB miss
  - Hardware
  - software



Physically addressed cache: virtually indexed

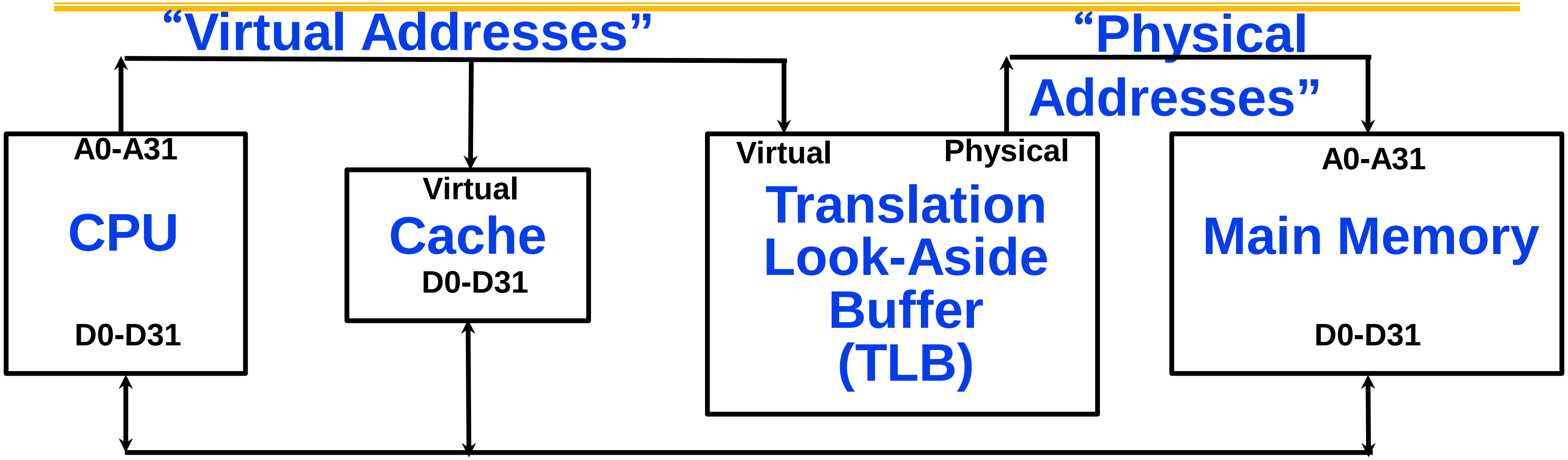


This works, but ...

Q. What is the downside?

A. Inflexibility. Size of cache limited by page size.

Virtually addressed cache



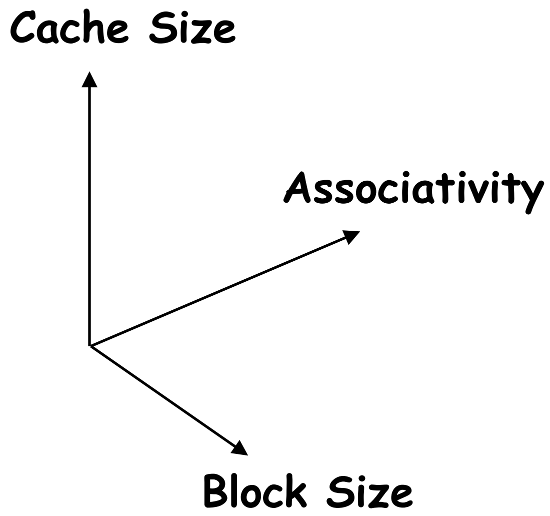
Only use TLB on a cache miss !

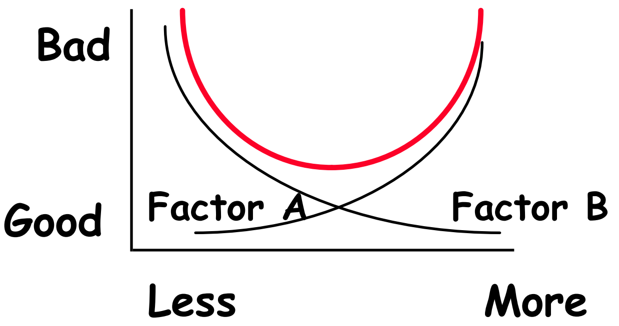
Downside: a subtle, fatal problem. What is it?

A. Synonym problem. If two address spaces share a physical frame, data may be in cache twice. Maintaining consistency is a nightmare.

Summary #1/3: The Cache Design Space

- **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**





Summary #2/3: Caches

- **The Principle of Locality:**
  - **Program 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 misses.**
  - **Capacity Misses: increase cache size**
  - **Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!**
- **Write Policy: Write Through vs. Write Back**
- **Today CPU time is a function of (ops, cache misses) This affects Compilers, Data structures, and Algorithms**

Summary #3/3: TLB, Virtual Memory

- **Page tables map virtual address to physical address**
- **TLBs are important for fast translation**
- **TLB misses are significant in processor performance**
- **Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:**
  1) Where can a block be placed?
  2) How is block found?
  3) What block is replaced on miss?
  4) How are writes handled?
- **Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is more important than memory hierarchy benefits**