

Memory Hierarchy

Dave Eckhardt
Bruce Maggs

Am I in the wrong class?

- “Memory hierarchy”: OS or Architecture?
 - Yes
- Why cover it here?
 - OS manages several layers
 - RAM cache(s)
 - Virtual memory
 - File system buffer cache
 - Learn core concept, apply as needed

Memory Desiderata (i.e., desirable properties)

- Capacious
- Fast
- Cheap
- Compact
- Cold
 - Pentium-4 2 Ghz: *75 watts!?*
- Non-volatile (can remember w/o electricity)

You can't have it all

- Pick one
 - ok, maybe two
- Bigger \Rightarrow slower (speed of light)
- Bigger \Rightarrow more defects
 - If constant per unit area
- Faster, denser \Rightarrow hotter
 - At least for FETs

Users want it all

- The ideal
 - Infinitely large, fast, cheap memory
 - Users want it (those pesky users!)
- They can't have it
 - Ok, so cheat!

Locality of reference

- Users don't really access 4 gigabytes uniformly
- 80/20 “rule”
 - 80% of the time is spent in 20% of the code
 - Great, only 20% of the memory needs to be fast!
- Deception strategy
 - Harness 2 (or more) kinds of memory together
 - Secretly move information among memory types

Cache

- Small, fast memory...
- Backed by a large, slow memory
- Indexed via the *large memory's* address space
- Containing the most popular parts
 - (at least at the moment)

Covered in Detail in 15-213 Lecture

- Memory technology (SRAM, DRAM)
- Disk technology (platters, tracks, sectors)
- Technology trends and statistics
- Concepts of spatial and temporal locality
- Basic description of a cache
- Types of cache misses
- Write policies
- Examples of caches

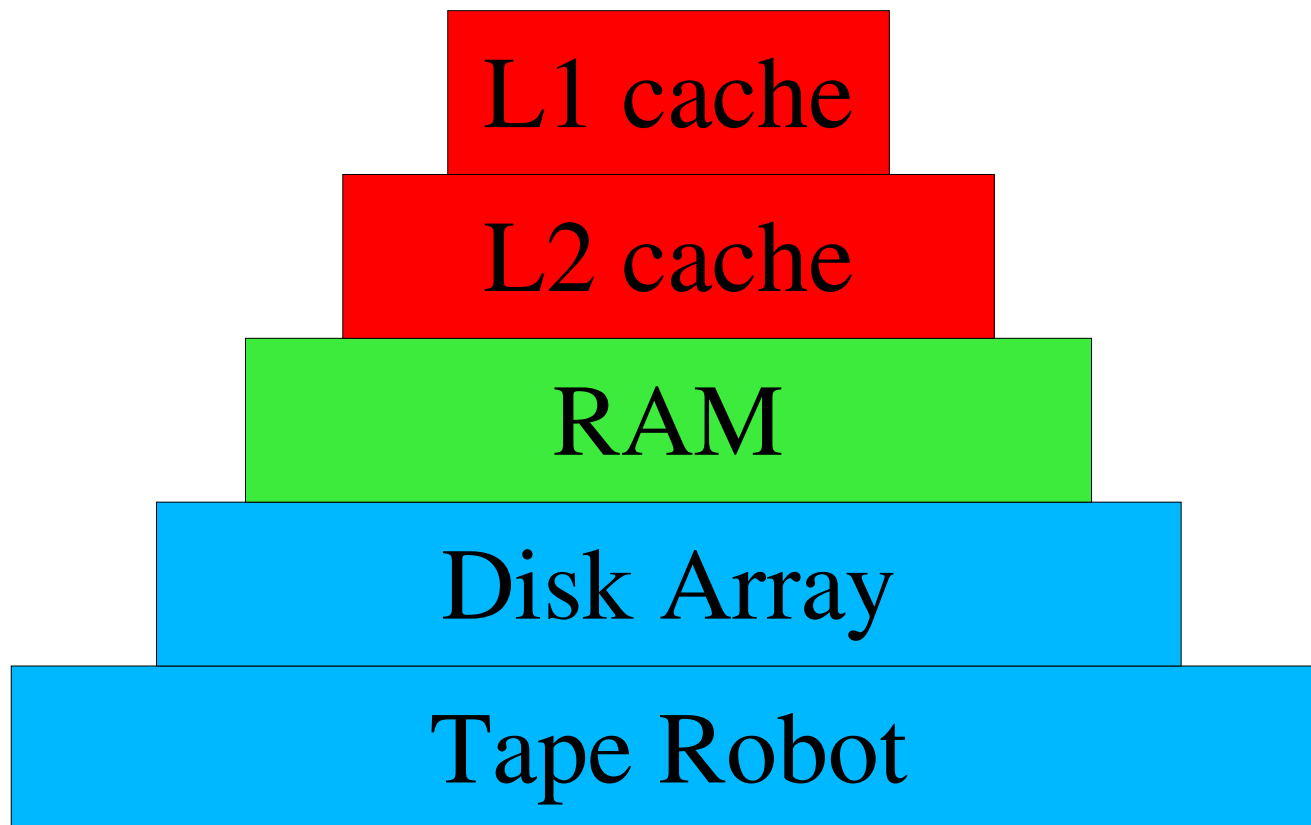
Cache Example – Satellite Images

- SRAM cache holds popular pixels
- DRAM holds popular image areas
- Disk holds popular satellite images
- Tape holds one orbit's worth of images

Great Idea...

- Clean general-purpose implementation?
 - `#include <cache.h>`
- No: tradeoffs different at each level
 - Size ratio: data address / data size
 - Speed ratio
 - Access time = $f(\text{address})$
- But *the idea* is general-purpose

Pyramid of Deception



Key Questions

- Line size
- Placement/search
- Miss policy
- Eviction
- Write policy

Content-Addressable Memory

- RAM
 - store(address, value)
 - fetch(address) \Rightarrow value
- CAM
 - store(address, value)
 - fetch(value) \Rightarrow address
- “It's always the last place you look”
 - Not with a CAM!

Main Memory Contents

Address	Contents
00010020	8B047E8D
00010024	565704EC
00010028	04758D53
0001002C	83E58955

CAM + SRAM = Cache

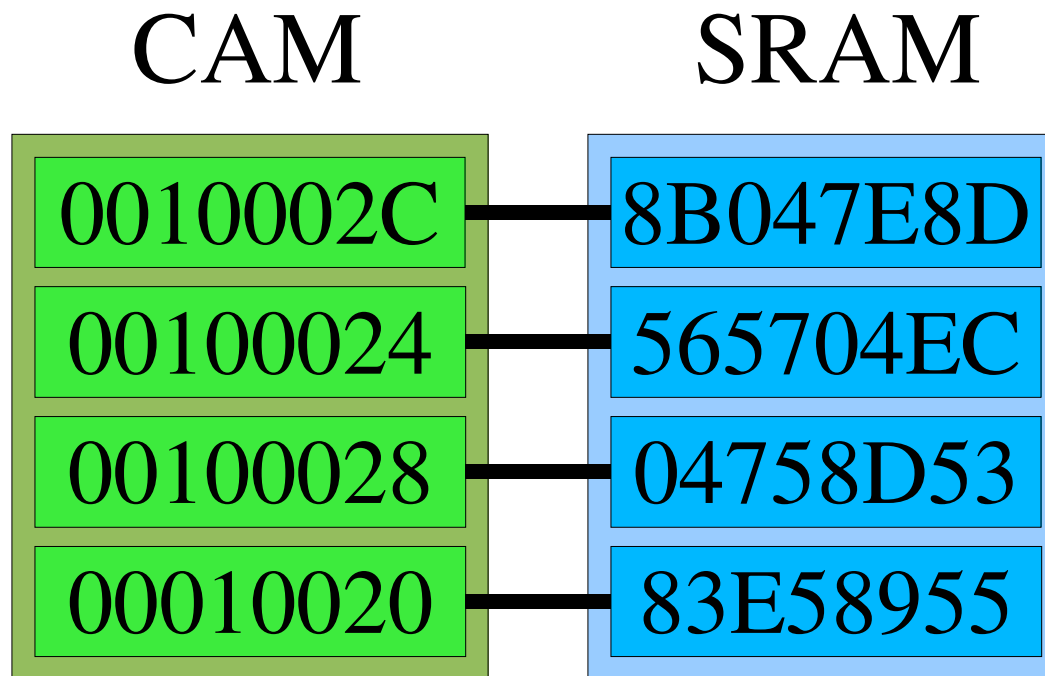
CAM

00100020	0
00100028	4
0010002C	8
00010024	12

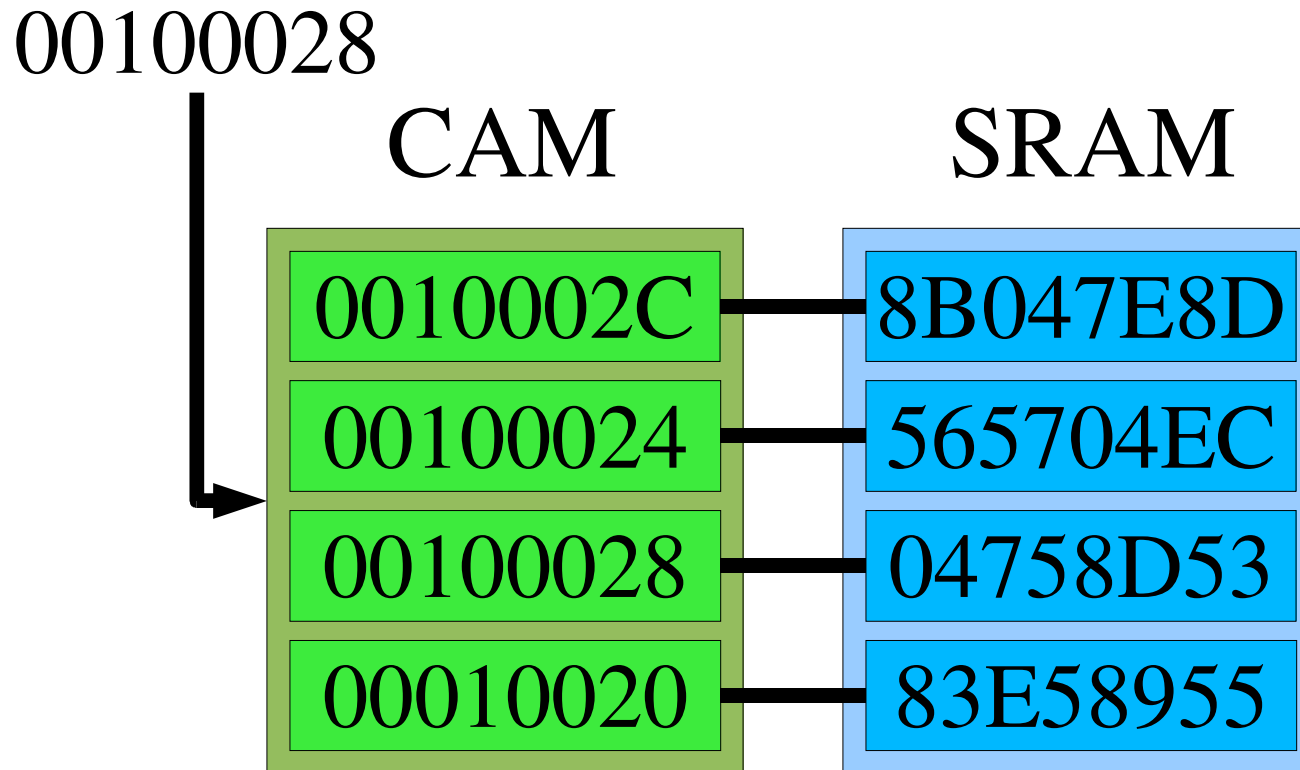
SRAM

0	8B047E8D
4	04758D53
8	83E58955
12	565704EC

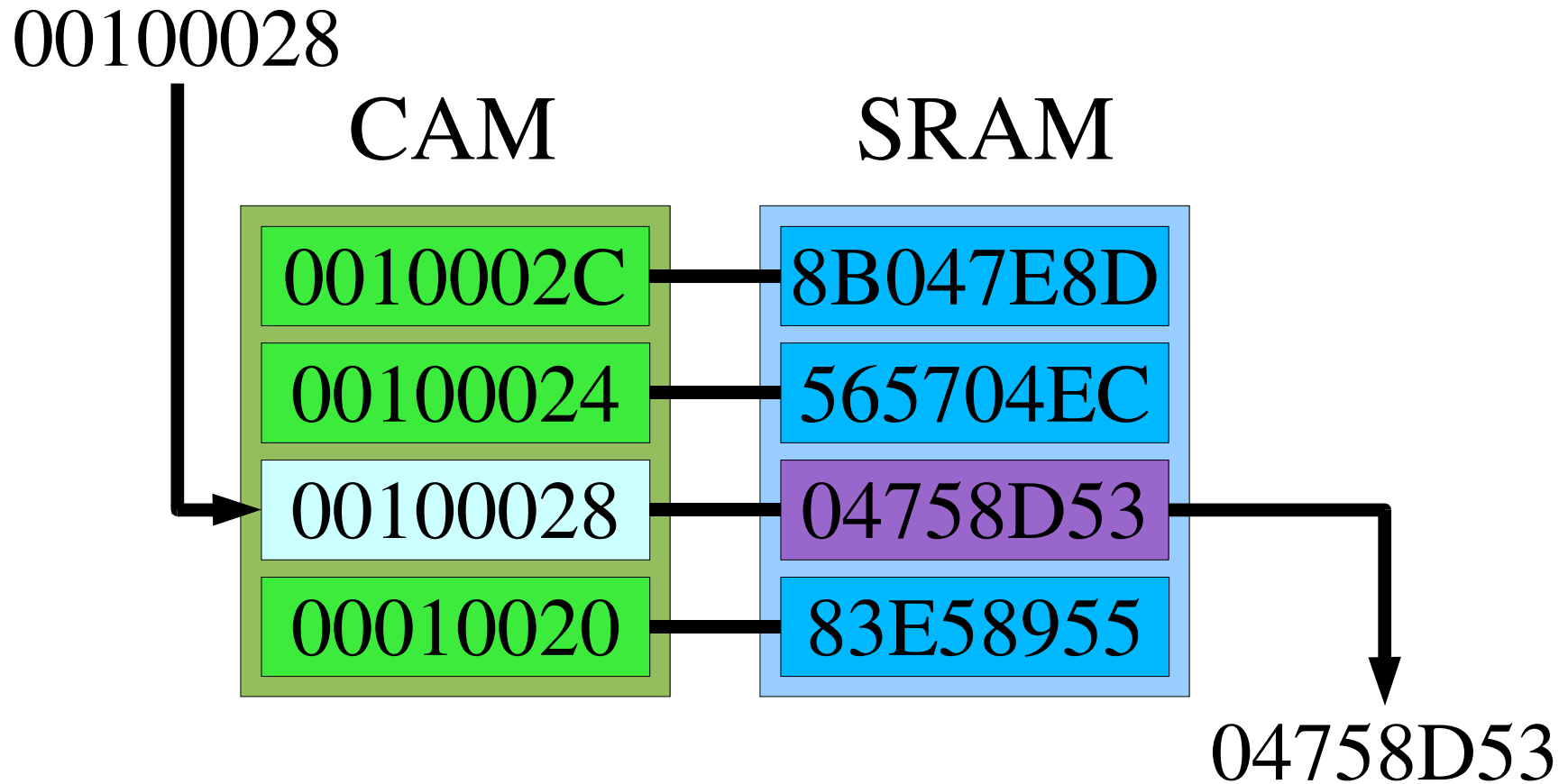
CAM + SRAM = Cache



Cache Lookup via CAM



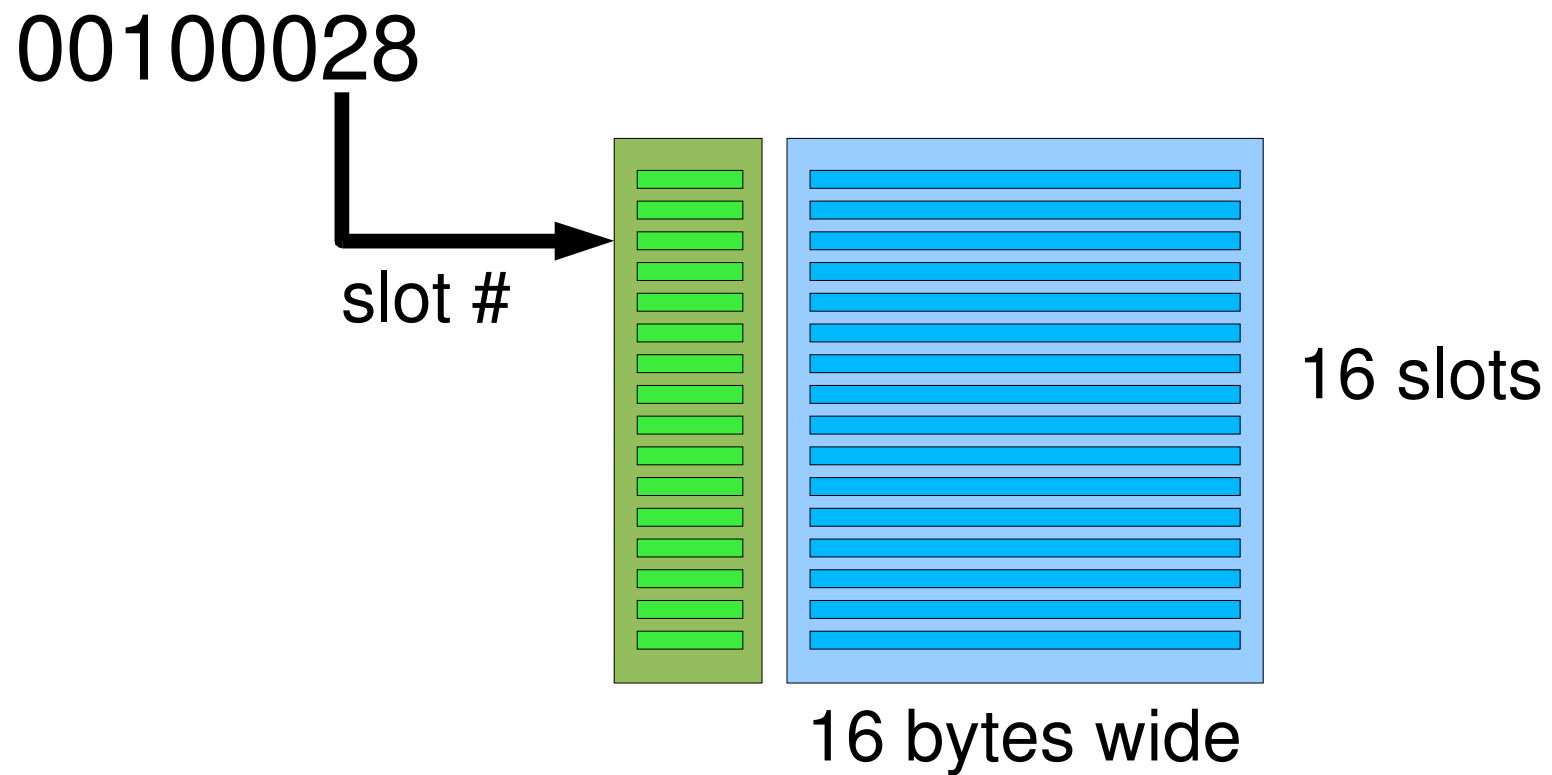
CAM Match Indicates Cache Slot



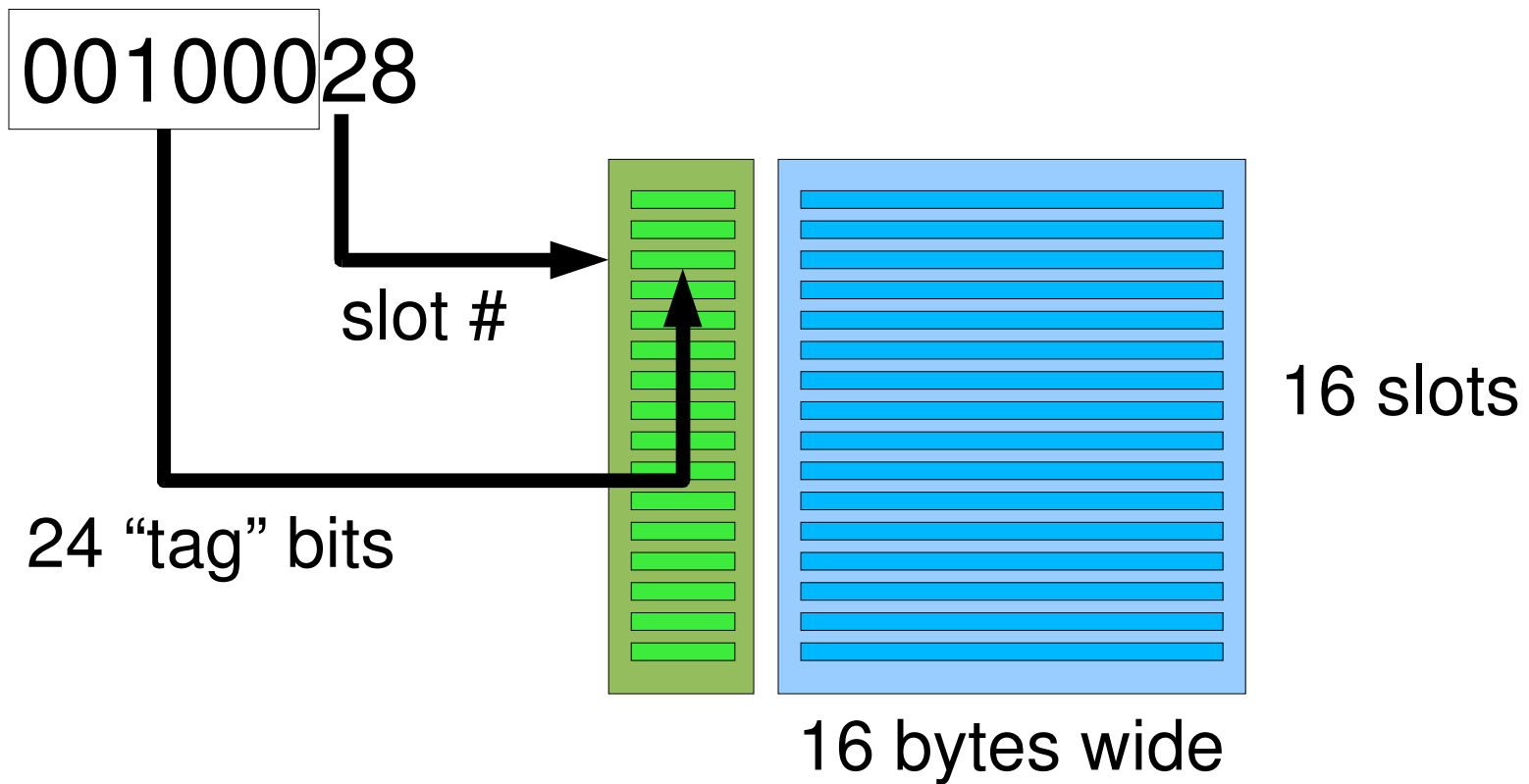
Content-Addressable Memory

- CAMS are cool!
- But fast CAMs are small (speed of light, etc.)
- If this were an architecture class...
 - We would have 5 slides on *associativity*
 - Not today: only 2

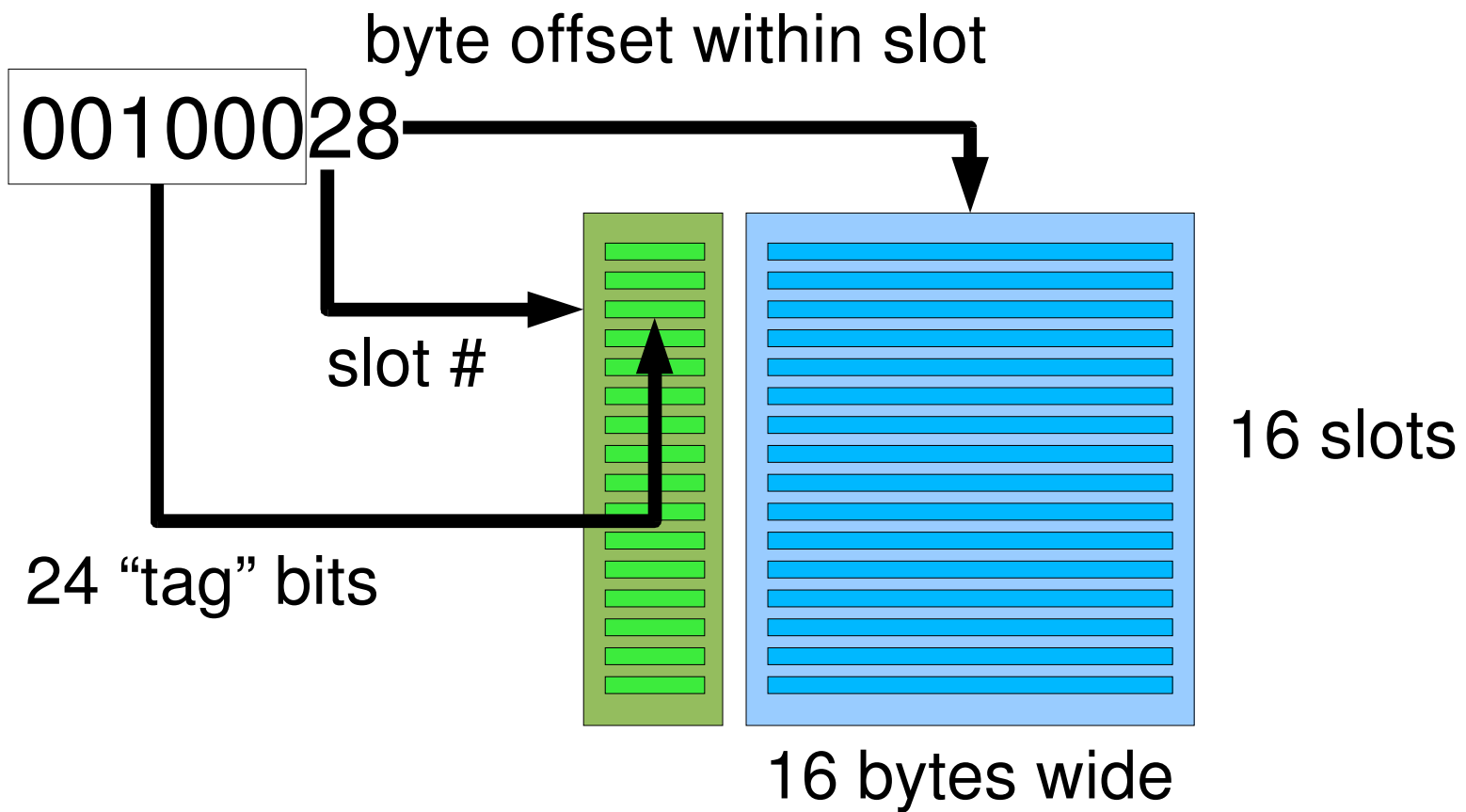
Direct-Mapped Cache



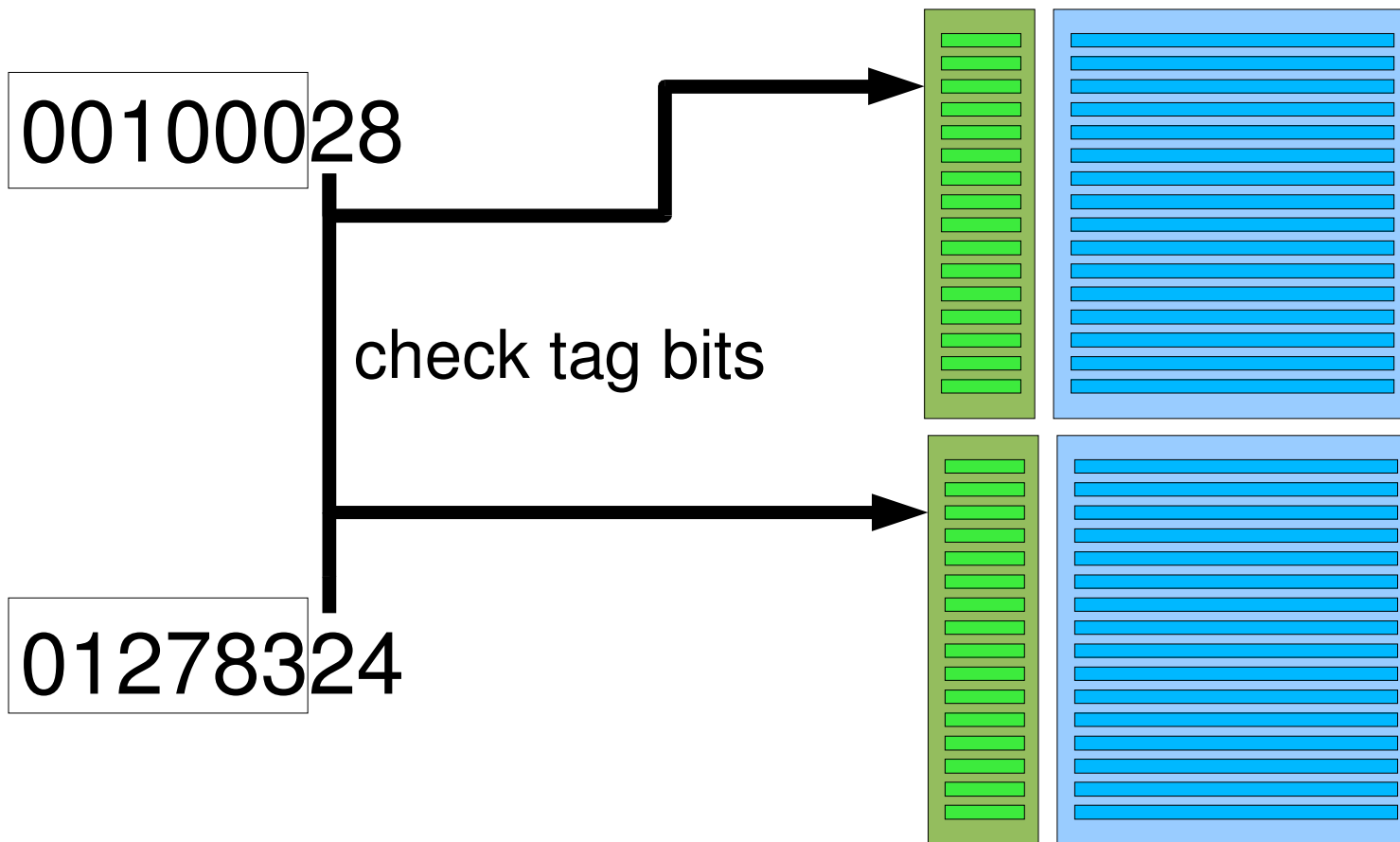
Direct-Mapped Cache



Direct-Mapped Cache



Two-Way Associative Cache



Placement/search

- Placement = "Where can we put ____?"
 - “Direct mapped” - each item has *one place*
 - Think: hash function
 - "Fully associative" - each item can be *any* place
 - Think: CAM

Placement/search

- Direct Mapped
 - Placement & search are trivial
 - False collisions are common
 - String move: `*q++ = *p++;`
 - Each iteration could be *two* cache misses!

Placement/search

- Fully Associative
 - No false collisions
 - Cache size/speed limited by CAM size
- Choosing associativity
 - Trace-driven simulation
 - Hardware constraints

Thinking the CAM way

- Are we having P2P yet?
 - I want the latest *freely available* Janis Ian song...
 - www.janisian.com/article-internet_debacle.html
 - ...who on the Internet has a copy for me to download?
- I know what I want, but not where it is...
 - ...Internet as a CAM

Sample choices

- L1 cache
 - Often direct mapped
 - Sometimes 2-way associative
 - Depends on phase of transistor
- Disk block cache
 - Fully associative
 - Open hash table = large variable-time CAM
 - Fine since "CAM" lookup time \ll disk seek time

Cache Systems Managed by OS

- Virtual memory (with hardware assistance)
- Translation caches
- Disk cache
- File system cache (AFS/NFS)
- Web cache
- ARP cache
- DNS cache

Example

- Disk block cache
 - Holds disk sectors in RAM
 - Entirely defined by software
 - ~ 0.1% to maybe 1% of disk (varies widely)
 - Indexed via (device, block number)

Eviction

- “The steady state of disks is 'full'”.
- Each placement requires an eviction
 - Easy for direct-mapped caches
 - Otherwise, policy is necessary
- Common policies
 - Optimal, LRU
 - LRU may be great, can be awful
 - 4-slot associative cache: 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, ...

Eviction

- Random
 - Pick a random item to evict
 - Randomness protects against pathological cases
 - When could it be *good*?
- L1 cache
 - LRU is easy for 2-way associative!
- Disk block cache
 - Frequently LRU, frequently modified
 - “Prefer metadata”, other hacks

Translation Caches

- Address mapping
 - CPU presents virtual address (%CS:%EIP)
 - Fetch segment descriptor from L1 cache (or not)
 - Fetch page directory from L1 cache (or not)
 - Fetch page table entry from L1 cache (or not)
 - Fetch the actual word from L1 cache (or not)

“Translation lookaside buffer” (TLB)

- Observe result of first 3 fetches
 - Segmentation, virtual \Rightarrow physical mapping
- Cache the *mapping*
 - Key = virtual address
 - Value = physical address
- Q: Write policy?

Challenges - Coherence

- Multiprocessor: 4 L1 caches share L2 cache
 - What if L1 does write-back?
- TLB: $v \Rightarrow p$ all wrong after context switch
- What about non-participants?
 - I/O device does DMA
- Solutions
 - Snooping
 - Invalidation messages (e.g., `set_cr3()`)