CS 354 - Machine Organization & Programming
Tuesday, March 7 and Thursday March 9, 2023

Print paper copies of this outline for best use.

Week 07 Activity: Heap Practice Assignment

Project p3: DUE on or before Friday March 25

Homework 3: DUE on or before Friday March 11

Last Week

Placement Policies (finish) Free Block - Too Large/Too Small Coalescing Free Blocks Free Block Footers	Explicit Free list Explicit Free List Improvements --- Heap Caveats Memory Hierarchy

This Week: MEMORY MANAGEMENT via CACHING blocks of memory for fast access

Heap Caveats Memory Hierarchy Locality & Caching Bad Locality Caching: Basic Idea & Terms Designing a Cache: Blocks Rethinking Addressing	Designing a Cache: Sets and Tags Basic Cache Lines Basic Cache Operation Basic Cache Practice
Next Week after Spring Break: Vary cache set size and Cache Writes B&O 6.4.3 Set Associative Caches 6.4.4 Fully Associative Caches 6.4.5 Issues with Writes 6.4.6 Anatomy of a Real Cache Hierarchy 6.4.7 Performance Impact of Cache Parameters	

DO NOT TAKE THIS WEEK OFF if you plan to take Spring Break off
Note: p4A and p4B will be released on Monday after Spring Break

Get p3 done this week. It is possible and will make for an actual Spring Break for you!
p3 - implement and test alloc (partA) and free (partB) by Monday and submit progress
p3 - implement coalesce by Wednesday and submit progress
p3 - complete testing and debugging by Friday and complete final submission

Heap Caveats

Consecutive heap allocations don't result in contiguous payloads!

→ Why?
- Payloads are interspersed with heap structure and possibly padding.
- Placement policies & heap structure can scatter allocations throughout heap.

Don't assume heap memory is initialized to 0!

OS initially clears heap pages for security,
but your recycled heap memory will have your old data
unless you use calloc()

Do free all heap memory that your program allocates!

→ Why are memory leaks bad?
They slowly kill your program's performance by cluttering heap with garbage blocks.
Bad leaks could ultimately consume your entire heap!

→ Do memory leaks persist when a program ends?
No, heap pages are returned to the OS for other uses.

Don't free heap memory more than once!

→ What is the best way to avoid this mistake?NULL freed pointers.

Don't read/write data in freed heap blocks!

→ What kind of error will result? Intermittent error

Don't change heap memory outside of your payload!

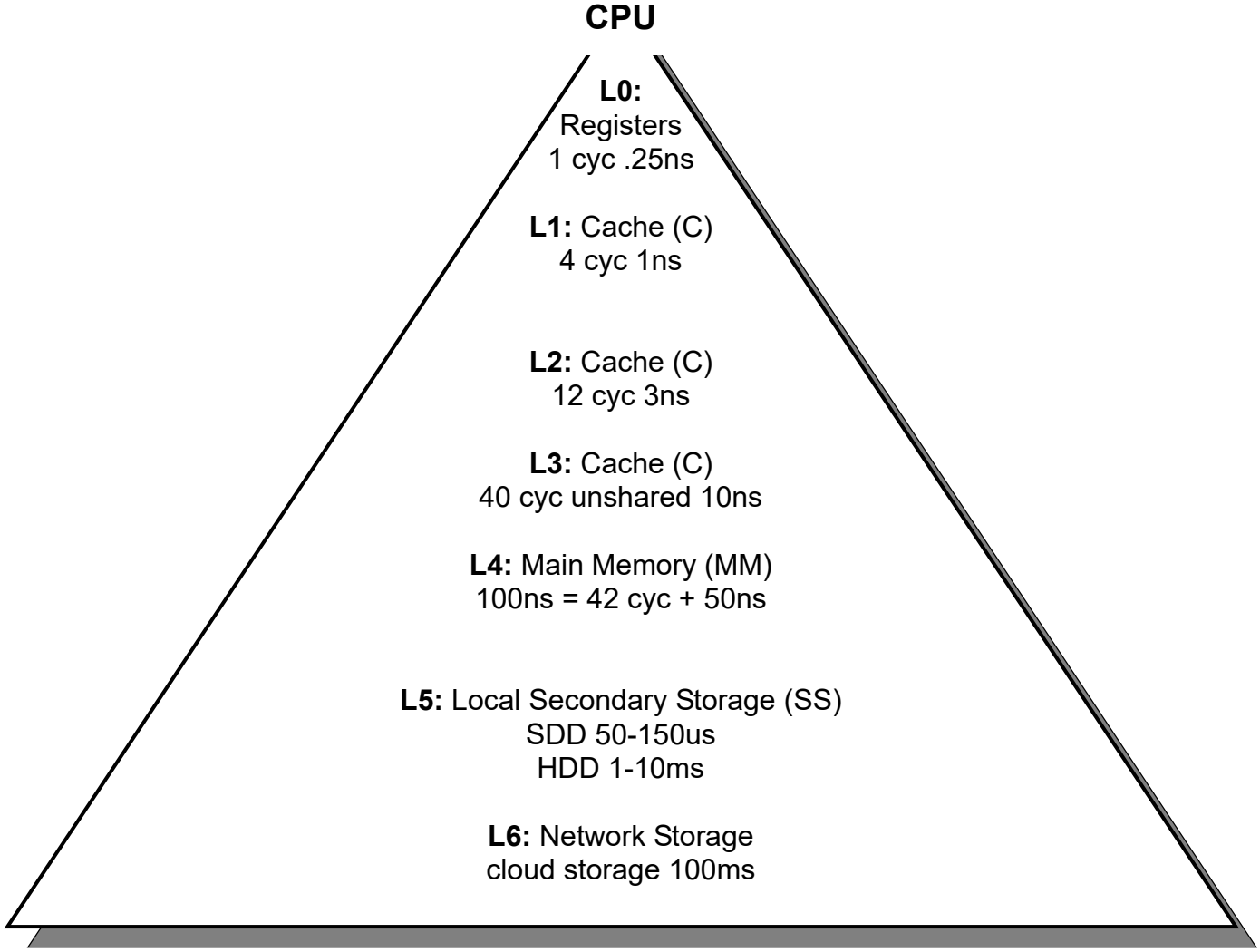
→ Why? You'll trash the heap's internal structure and/or another block's payload.

Do check if your memory intensive program has run out of heap memory!

→ How? Check that allocator's return value is not NULL

Memory Hierarchy

❋ *The memory hierarchygives the illusion of having lots of fast memory.*



Cache

is a smaller faster mem that acts as a staging area for data stored in a larger slower mem

Memory Units

word: size used by CPU transfer betweenL1 & CPU

block: size used by C transfer betweenC levels & MM

page: size used by MM transfer betweenMM & SS

Memory Transfer Time: https://simple.wikipedia.org/wiki/Orders_of_magnitude_(time)

cpu cycles:**used to measure time**

latency:**memory access time (delay)**

Locality & Caching

What?

temporal locality: when a recently accessed memory location
is repeatedly accessed in the near future

spatial locality: when a recently accessed memory location
is followed by nearby memory locations being accessed in the near future

locality is designed into

Example

```
int sumArray(int a[], int size, int step) {
   int sum = 0;
   for (int i = 0; i < size; i += step)
      sum += a[i];
   return sum;
}
```

→ List the variables that clearly demonstrate temporal locality.

→ List the variables that clearly demonstrate spatial locality.

stride:

❋ *The caching system uses localityto predict what the cpu will need in the near future.*

How? The caching system

temporal: anticipates data will be reused so it copies

spatial: anticipates nearby data will be used so it copies

cache block:

❋ *Programs with good localityrun faster since they work better with the caching system!*

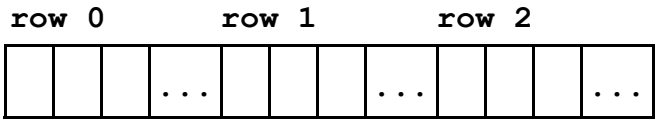
Why? Programs with good locality

Bad Locality

Why is this code bad?

```
int a[ROWS][COLS];

for (int c = 0; c < COLS; c++)
   for (int r = 0; r < ROWS; r++)
      a[r][c] = r * c;
```



→ How would you improve the code to reduce stride?

Key Questions for Determining Spatial Locality:

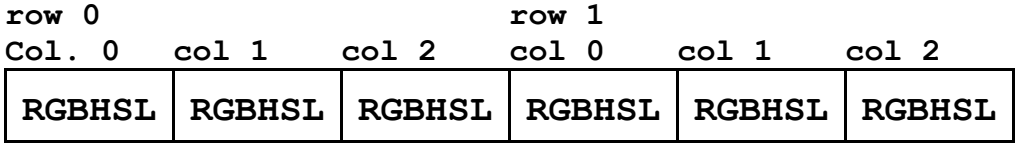
1. What does the memory layout look like for the data?

2. What is the stride of the code across the data?

Why is this code bad?

```
struct {
   float rgb[3];
   float hsl[3];
} image[HEIGHT][WIDTH];

for (int v = 0; v < 3; v++)
   for (int c = 0; c < WIDTH; c++)
      for (int r = 0; r < HEIGHT; r++) {
         image[r][c].rgb[v] = 0;
         image[r][c].hsl[v] = 0;
      }
```



➢ How would you improve the code to reduce stride?

Good or bad locality?

- Instruction Flow:
  - sequencing?
  - selection?
  - repetition?
- Searching Algorithms:
  - linear search
  - binary search

Caching: Basic Idea & Terms

Assume: Memory is divided into 32 byte blocks and all blocks are already in main memory. Cache L1 has 4 locations to store blocks and L2 has 16 locations to store blocks.

→ Update the memory hierarchy below given blocks are accessed in this sequence: 22,11,22,44,11,33,11,22,55,27,44

cache miss

cold miss

capacity miss

conflict miss

cache hit

placement policies

1.

2.

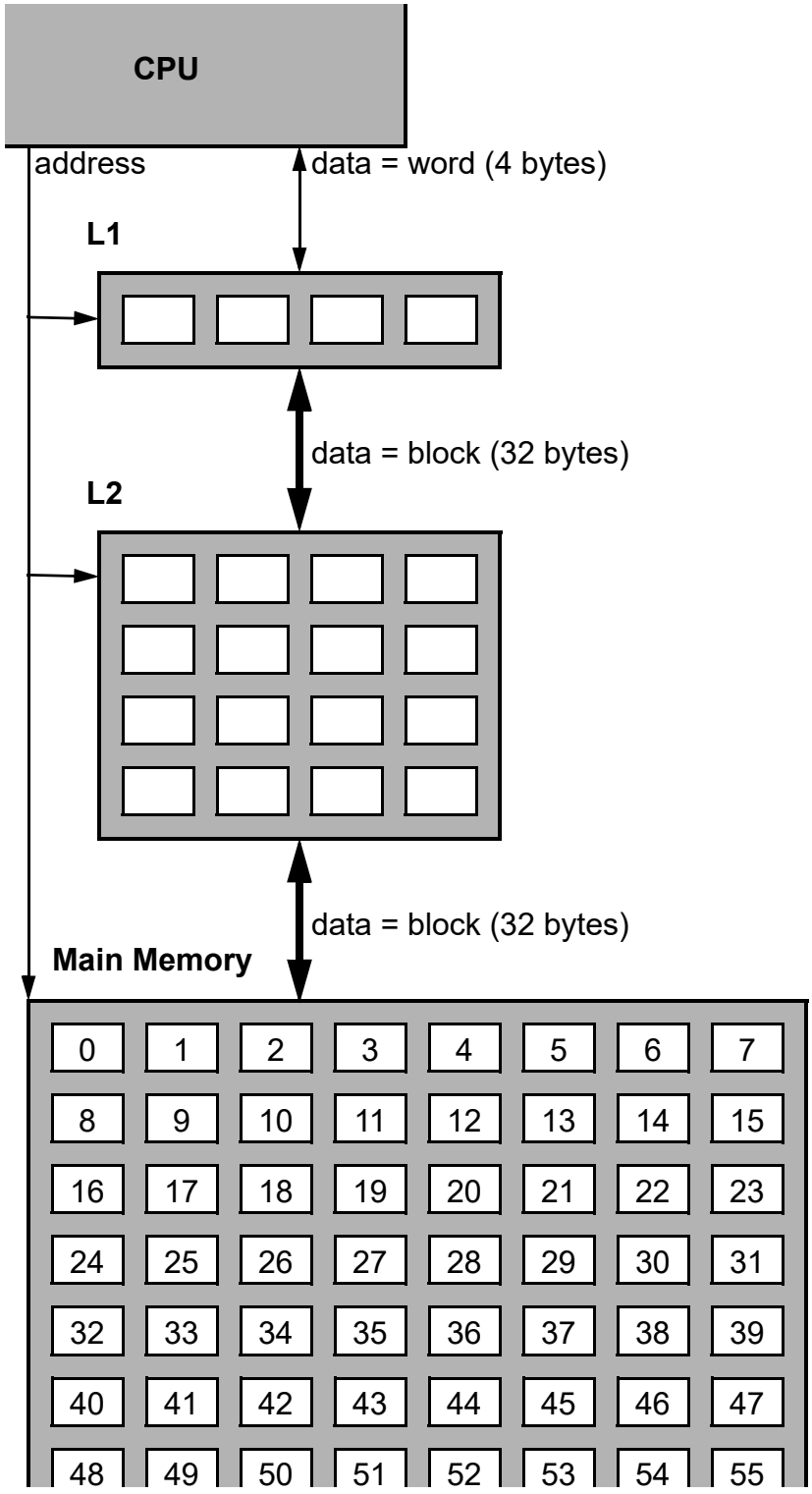
replacement policies

1.

2.

victim block

working set

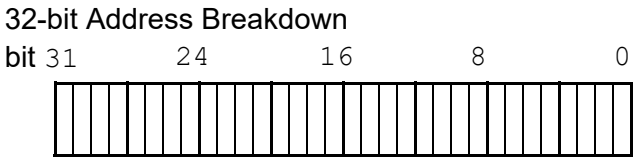


Designing a Cache: Blocks

❋ *The bits of an address*

How many bytes in an address space?

Let M be number of bytes in AS, IA-32 is 4GB



$M = 2^m$
$m = \log_2 M$

Thus m is number of bits in an address, IA-32 is 32

How big is a block?

❋ *Cache blocks must be big enough but small enough*

Let B be number of bytes per block, IA-32 is 32 bytes

$B = 2^b$
$b = \log_2 B$

b bits:

word offset

byte offset

➢ What is the problem with using the most significant bits (left side) for the b bits?

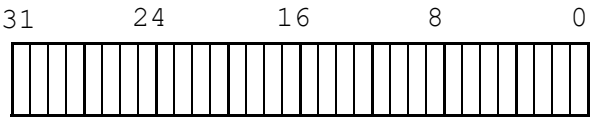
How many 32-byte blocks of memory in a 32-bit address space?

❋ *The remaining bits of an address encode the block number.*

Rethinking Adressing

- *An address identifies which byte in the VAS to access.*
- *An address is*

32-bit Address Breakdown

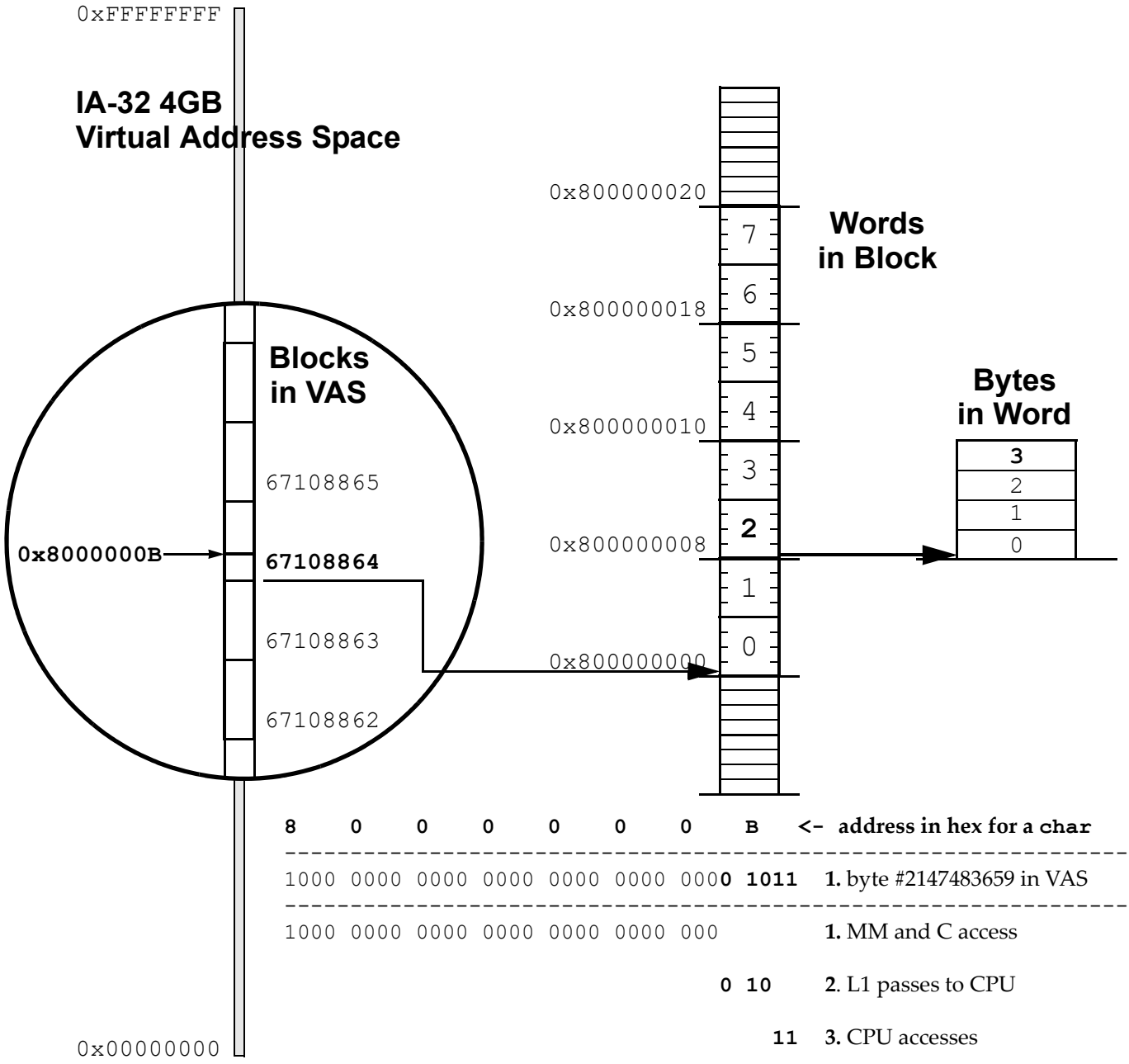


Memory Access in Caching System

step 1. Identify which

step 2. Identify which

step 3. Identify which



8	0	0	0	0	0	0	B	<- address in hex for a char	
1000	0000	0000	0000	0000	0000	0000	1011	**1.** byte #2147483659 in VAS	
1000	0000	0000	0000	0000	0000	000		**1.** MM and C access	
							0 10	**2.** L1 passes to CPU	
							11	**3.** CPU accesses	

Designing a Cache: Sets & Tags

❋ *A cache must be searched*

→ Problem?

Improvement?

set:

❋ *The block number bits of an address*

1.

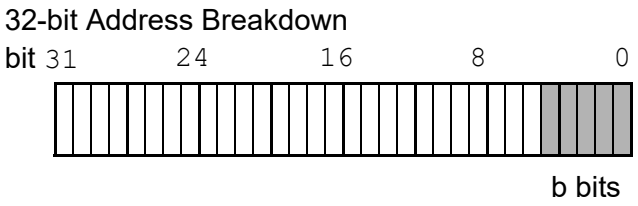
2.

How many sets in the cache?

Let S bethe number of sets in cache

$S = 2^s$

$s = \log_2 S$

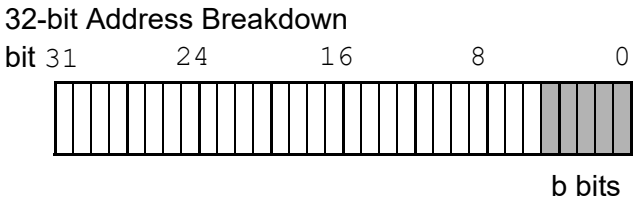


s bits:

➢ What is the problem with using the most significant bits (left side) for the s bits?

→ How many blocks map to each set for a 32-bit AS and a cache with 1024 sets? 8192 sets?

Since different blocks map to the same set how do we know which block is in a set?



t bits:

❋ *When a block is copied into a cache its t bits are also stored as its tag*

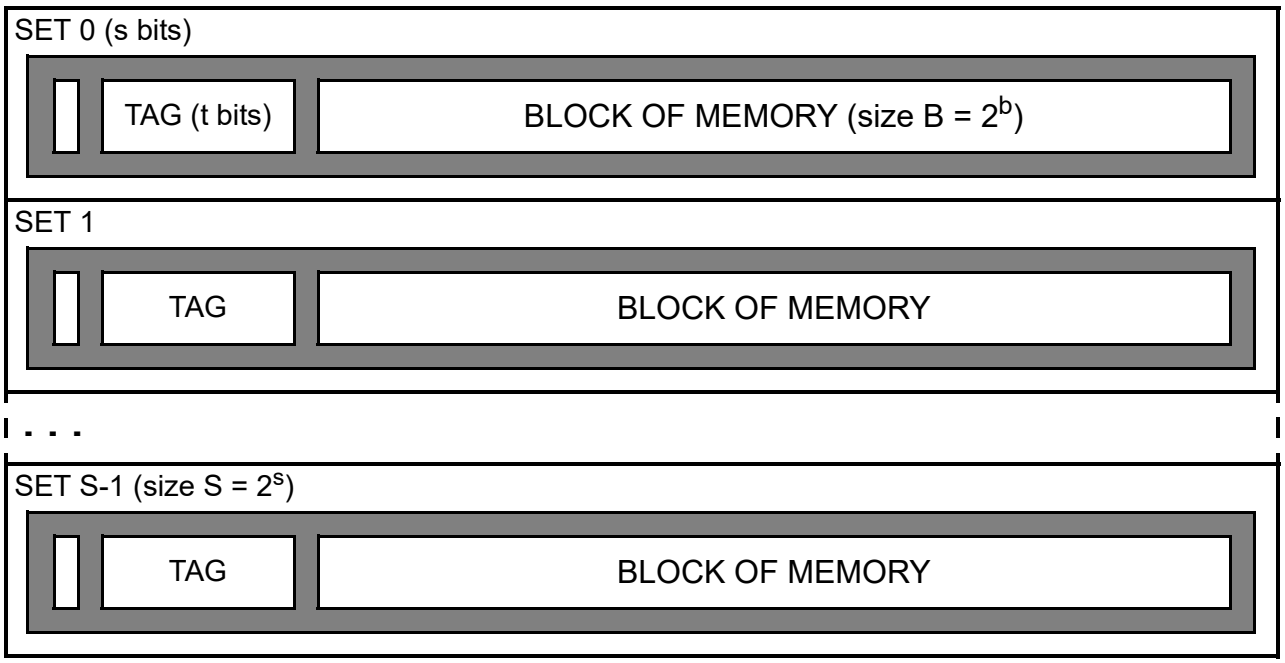
Basic Cache Lines

What? A *line* is

-
-

❋ *In our basic cache each cache set*

Basic Cache Diagram

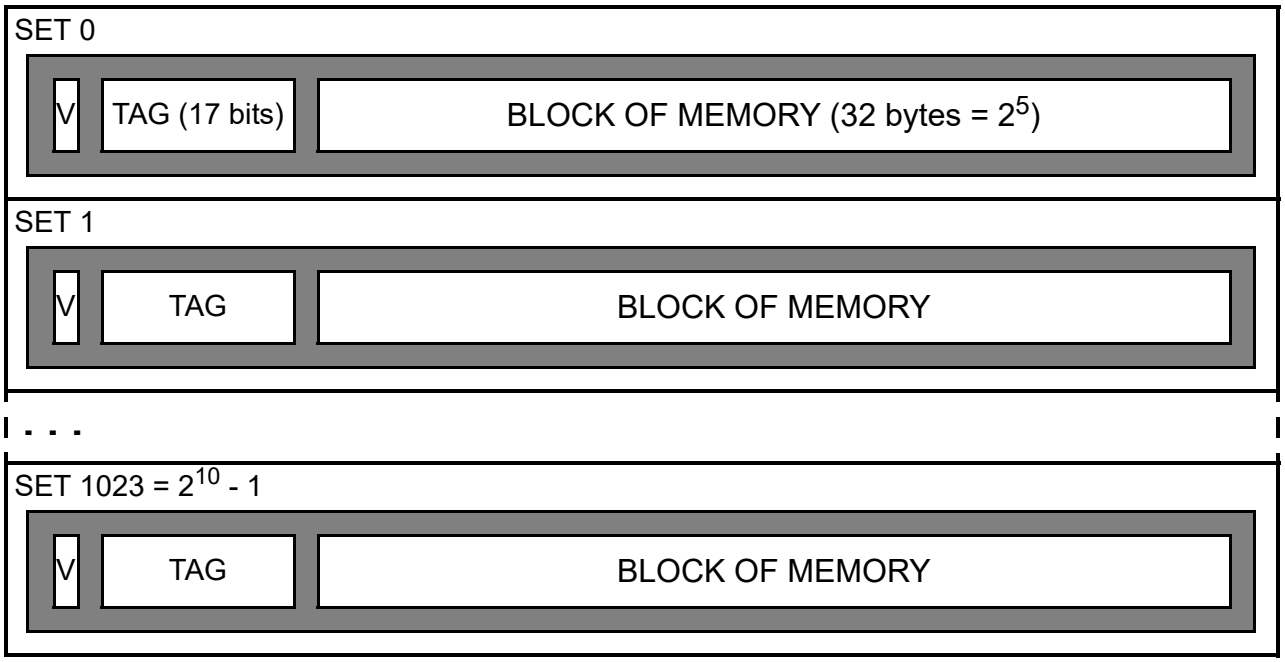


→ How do you know if a line in the cache is used or not?

→ How big is a basic cache given S sets with blocks having B bytes?

Basic Cache Operation

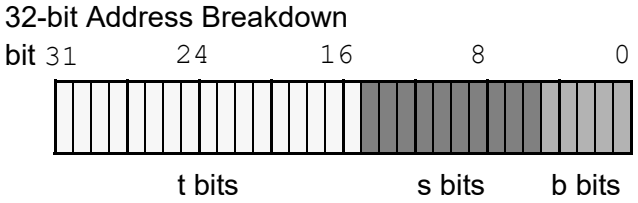
Basic Cache Diagram



→ How big is this basic cache?

How does a cache process a request for a word at a particular address?

1. *Set Selection*



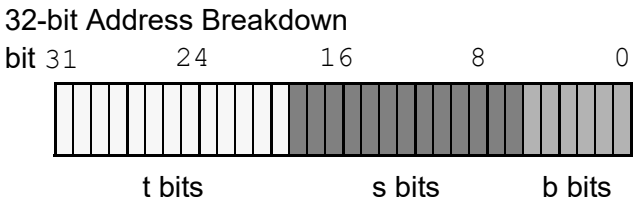
2. *Line Matching*

if no match or valid bit is 0

if match and valid bit is 1

Basic Cache Practice

You are given the following 32-bit address breakdown used by a cache:



→ How big are the blocks?

→ How many sets?

→ How big is this basic cache?

Assume the cache design above is given the following specific address: `0x07515E2B`

→ Which set should be checked given the address above?

→ Which word in the block does the L1 cache access for the address?

➢ Which byte in the word does the address specify?

Assume address above maps to a set with its line having the following V status and tag.

→ Does the address above produce a hit or miss?

	V	tag
1.)	1	0x0750
2.)	0	0x0750
3.)	1	0x00EA
4.)	0	0x00EA