CS 354 - Machine Organization & Programming
Tuesday Oct 17, and Thursday Oct 19, 2023

Print paper copies of this outline for best use.

Week 07 Activity: Heap Practice Assignment

Project p3A: DUE on or before Friday 10/20 & p3B on10/27

Homework 3: DUE on or before Monday 10/16

Learning Objectives

- Describe the relative difference in speed and size of various types of memory and storage.
- Identify and describe the units of transfer used by each storage type.
- Define, identify, and describe spatial locality and temporal locality
- Identify good locality in program code
- Explain why programs with good locality work better with caching.
- Compute stride (in words) of array memory accesses
- Determine if common algorithms produce good or bad locality for each type.
- Define and use basic cache terminology
- Convert hex to binary, use bits of an address to determine if the address is in a given cache
- Extract bits and compute the set index, tag bits, and byte, determine if byte is in the cache

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

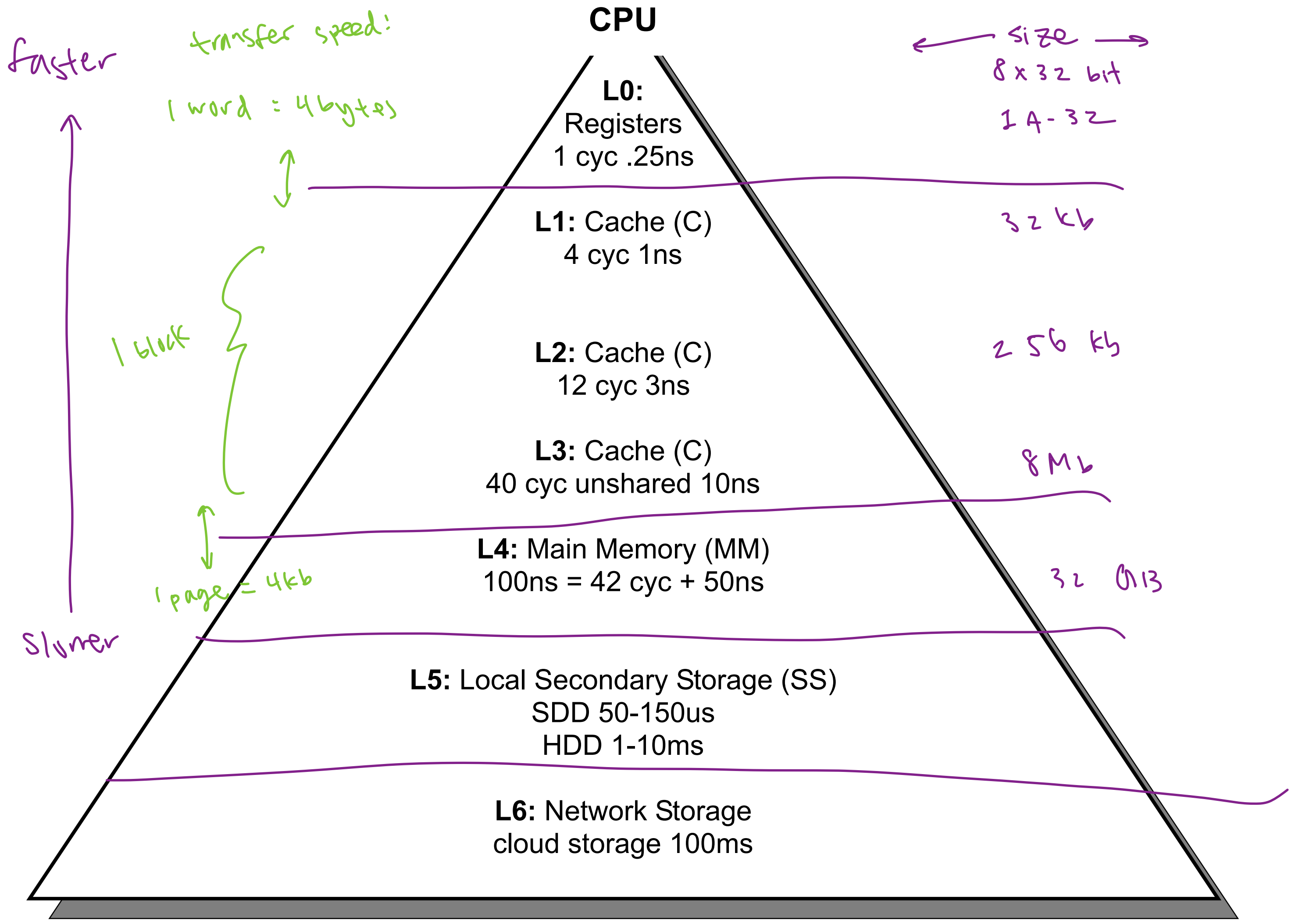
Memory Hierarchy Locality & Caching Bad Locality Caching: Basic Idea & Terms Designing a Cache: Blocks	Rethinking Addressing Designing a Cache: Sets and Tags Basic Cache Lines Basic Cache Operation Basic Cache Practice
Next Week: Vary cache set size and Cache Writes B&O 6.4.3 Set Associative Caches 6.4.4 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	

Note: p4A and p4B will be released next week

Get p3A and p3B done this week and avoid the rush!
p3 - implement and test alloc (partA) and free (partB) by Monday and submit progress
p3 - implement immediate coalescing by Wednesday and submit progress
p3 - complete testing and debugging by Friday and complete final submission

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	= 4bytes / word
block: size used by C	transfer betweenC levels & MM	= 32bytes / block
page: size used by MM	transfer betweenMM & SS	= 4 kb/ page

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   int i in for

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

locality is designed into hardware
o.s.
application

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. i, sum, step

→ List the variables that clearly demonstrate spatial locality. a, a[i]

stride: step size in words (4 bytes)
good spacial locality when stride ≈ 1 word

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

How? The caching system

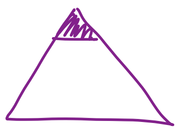
temporal: anticipates data will be reused so it copies value into cache

spatial: anticipates nearby data will be used so it copies a block

cache block: unit of memory transfered between main mem. and cache levels

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

Why? Programs with good locality maximize use of data @ top hierarchy

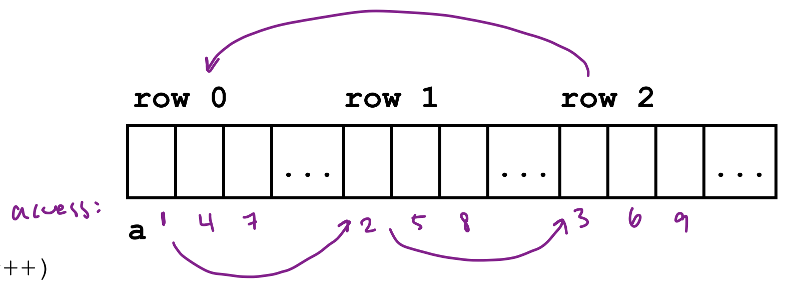


Bad Locality

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;
```



not good spacial

stride length = # cols

➔ How would you improve the code to reduce stride?

change for loop order

Key Questions for Determining Spatial Locality:

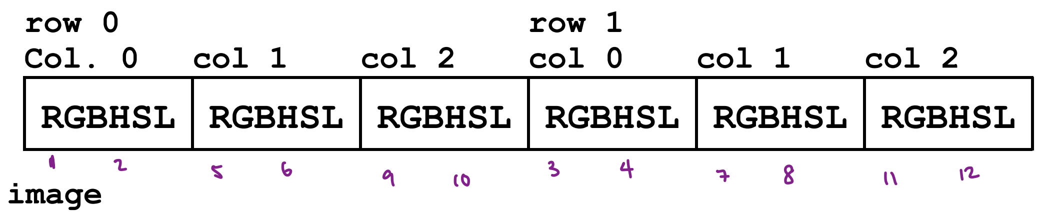
1. What does the memory layout look like for the data? 2D SAA are in row-major order
2. What is the stride of the code across the data?

Why is this code bad?

H = 2 , W = 3

```
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;
      }
```



for each row
  for each col
    for each v
      .rgb [v]
      .~~hsl [v]~~
    for each v
      .hsl [v]

is better

➢ How would you improve the code to reduce stride?

Good or bad locality?

- Instruction Flow:
  - sequencing? good spacial locality , poor temporal ↗ no repetition
  - selection? poor 〃 , poor 〃
  - repetition? dept. on loop , good temporal locality for data repeated access
- Searching Algorithms:
  - linear search array - good spacial , good temp for value being matched
    linked list - poor spacial , 〃 〃 and temp val
  - binary search array - poor spacial

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.

working set →

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

L1 Miss
L2 Miss
L3 Hit

cache miss

block not found

cold miss

when cache has room but block not there

capacity miss

when cache is too small for working set

conflict miss

when ≥ 2 blocks map to same location

cache hit = FASTER

when block is found in cache

placement policies

w1 1. unrestricted

w1 2. restricted

replacement policies

w1 1. choose any

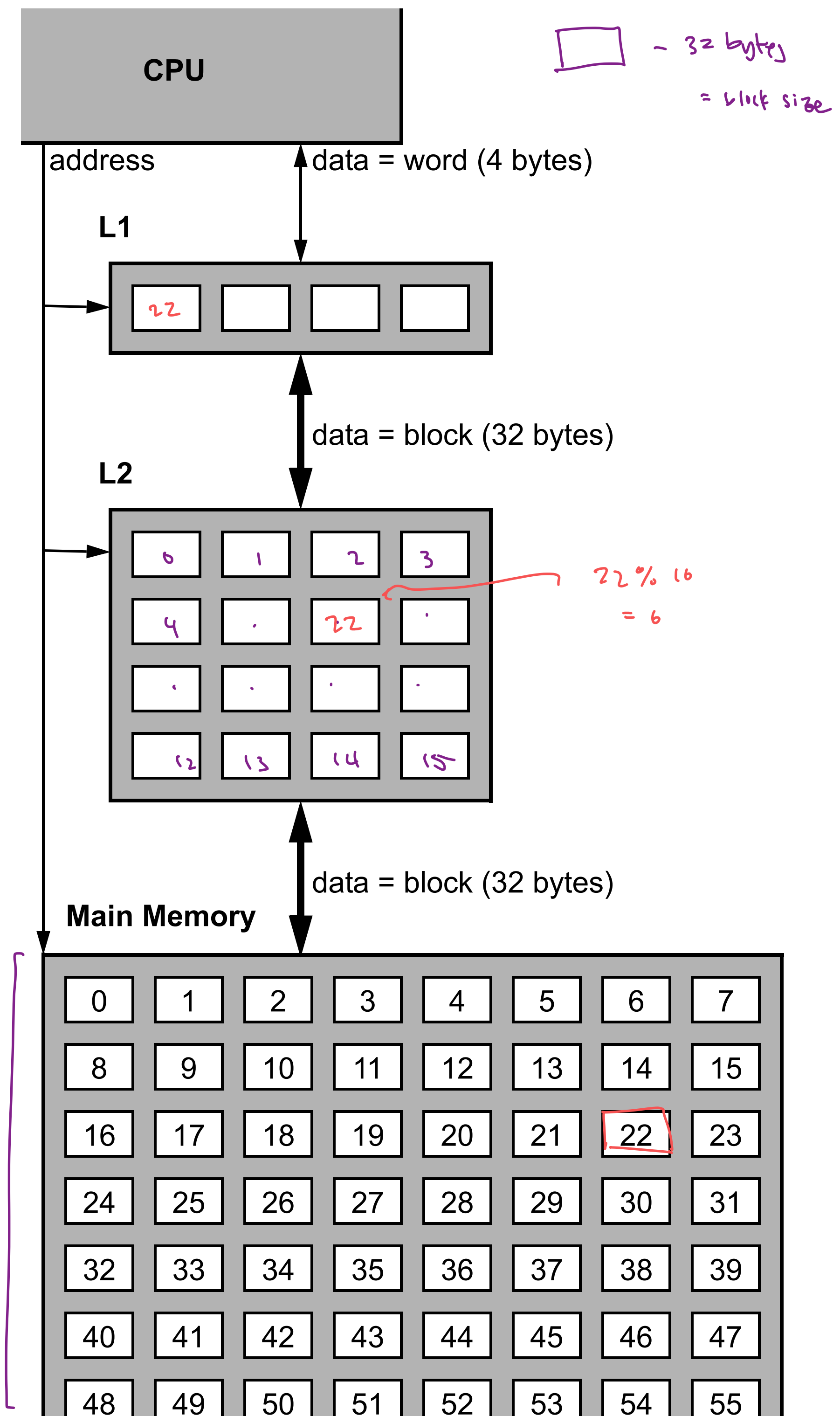
w1 2. no choice, replace existing block

victim block

cache block chosen to be replace

working set

set of blocks used during working time



□ - 32 bytes = block size

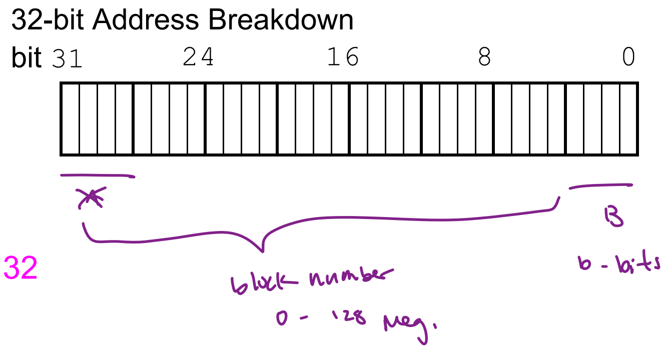
1 page
everything fits

Designing a Cache: Blocks

✵ *The bits of an address* are used to determine if blocks containing that address is in the cache

How many bytes in an address space?

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



$4GB = 2^{32}$

$M = 2^m$

$m = \log_2 M$ $\quad \therefore\ 32 = \log_2 (4GB)$

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

How big is a block? # of bytes - 8 bits / byte

✵ *Cache blocks must be big enough* to capture spacial locality
but small enough to minimize latency

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

$B = 2^b$ = 32 bytes

$b = \log_2 B$ = 5

<u>b bits</u>: # of addr. bits to determine which byte in a block

<u>word offset</u> identifies which word in a block (8 words /block)

<u>byte offset</u> identifies which bytes in a word (4 bytes / word)

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

we are picking bytes very far away from each other as a small change in the most sig. bit skips many bits $\longrightarrow$ lose spacial locality $\therefore$ use least sig. bits

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

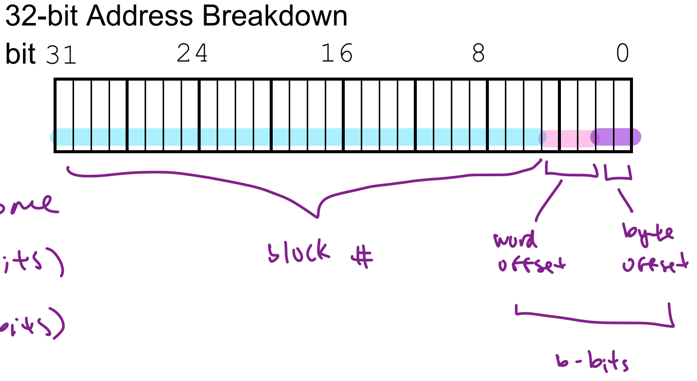
$$\frac{2^{32}}{2^5} = 2^{32-5} = 2^{27} = 2^7 \cdot 2^{20} = 128 \cdot 1MB = 128\ M \text{ of blocks}$$

✵ *The remaining bits of an address encode the block number.* = 124,217,728 blocks

Rethinking Adressing

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

✵ *An address is* divided into parts to access mem. in steps

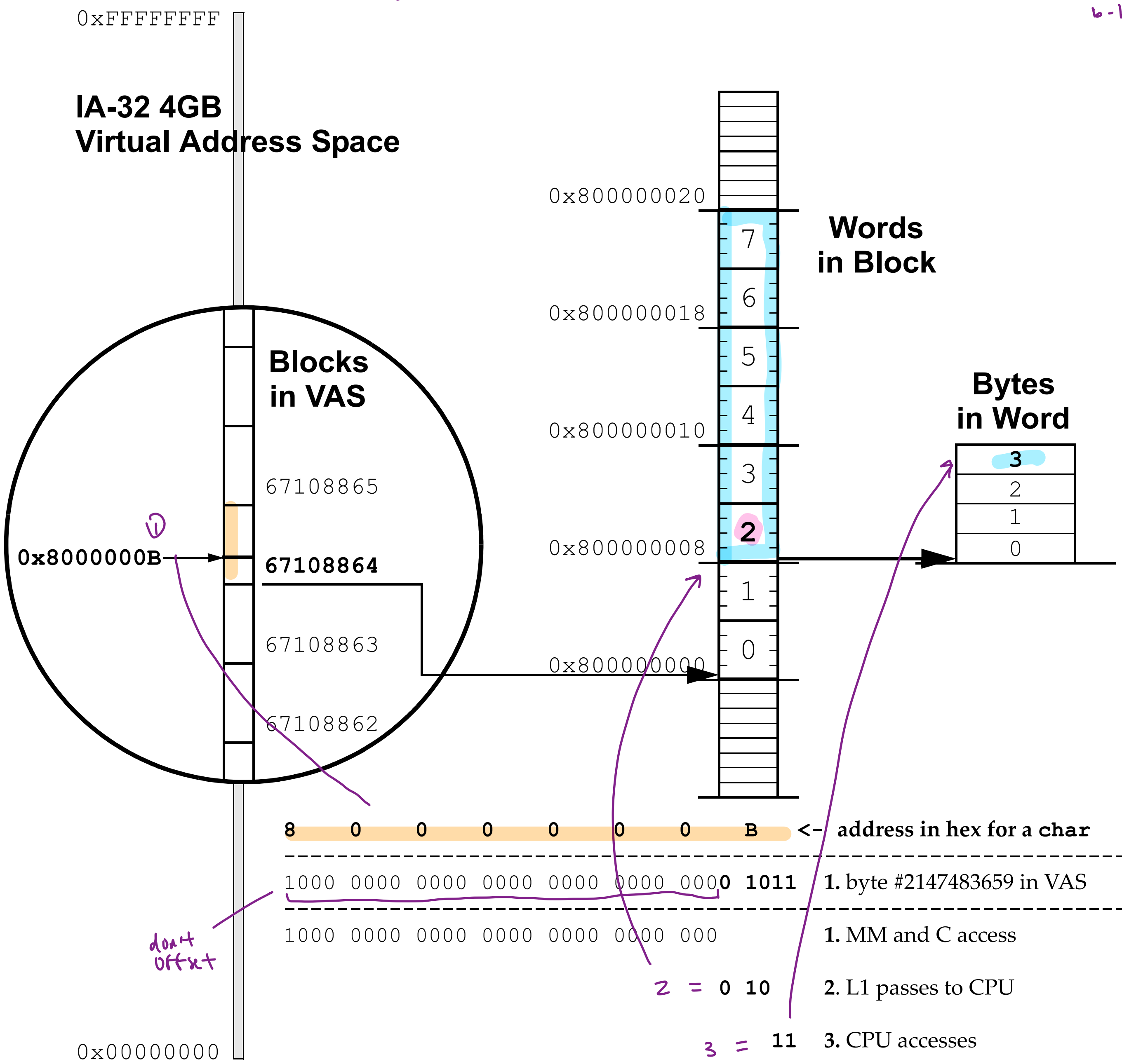


Memory Access in Caching System

step 1. Identify which block in virtual addr space

step 2. Identify which word in block (3 bits)

step 3. Identify which byte in word (2 bits)



```
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

                           2 = 0 10         2. L1 passes to CPU

                             3 = 11         3. CPU accesses
```

Designing a Cache: Sets & Tags

✵ *A cache must be searched* if unrestricted placement policy

→ Problem? slow - O(n) where n = # locations in cache (L 1)

Improvement? limit (restrict) where each block can be stored (L 2)

<u>set</u>: where block is uniquely mapped in a cache

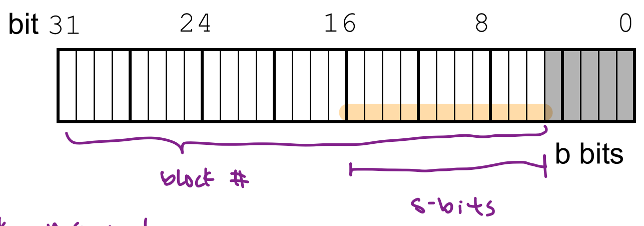
✵ *The block number bits of an address* are devided into 2 parts
- 27 most sig. bits

1. set - maps blocks to specific set in cache
2. tag - uniquely identify blocks in the set

How many sets in the cache?

Let S be the number of sets in cache

32-bit Address Breakdown



$S = 2^s$ let $S = 1024$ $S = 8192$

$s = \log_2 S$ $s = 10$ bits $s = 13$

<u>s bits</u>: bits that identify which set the block maps to

(in next least sig. bits after (before) b-bits)

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

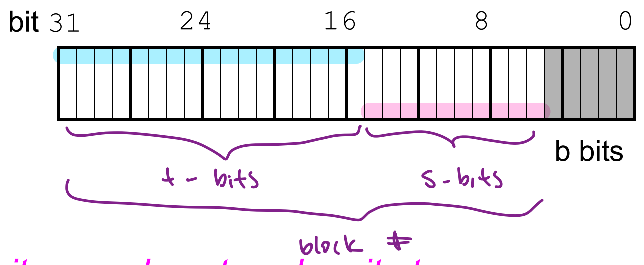
lose spatial locality

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

$2^{32-5} = \frac{2^{27} \text{ blocks}}{2^{10} \text{ sets}} = 2^{17}$ blocks/set $\frac{2^{27} \text{ blocks}}{2^{13} \text{ sets}} = 2^{14}$ blocks/set

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

32-bit Address Breakdown



use remaining bits as unique tab

<u>t bits</u>: bits of addr. that identify which blocks in the set

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

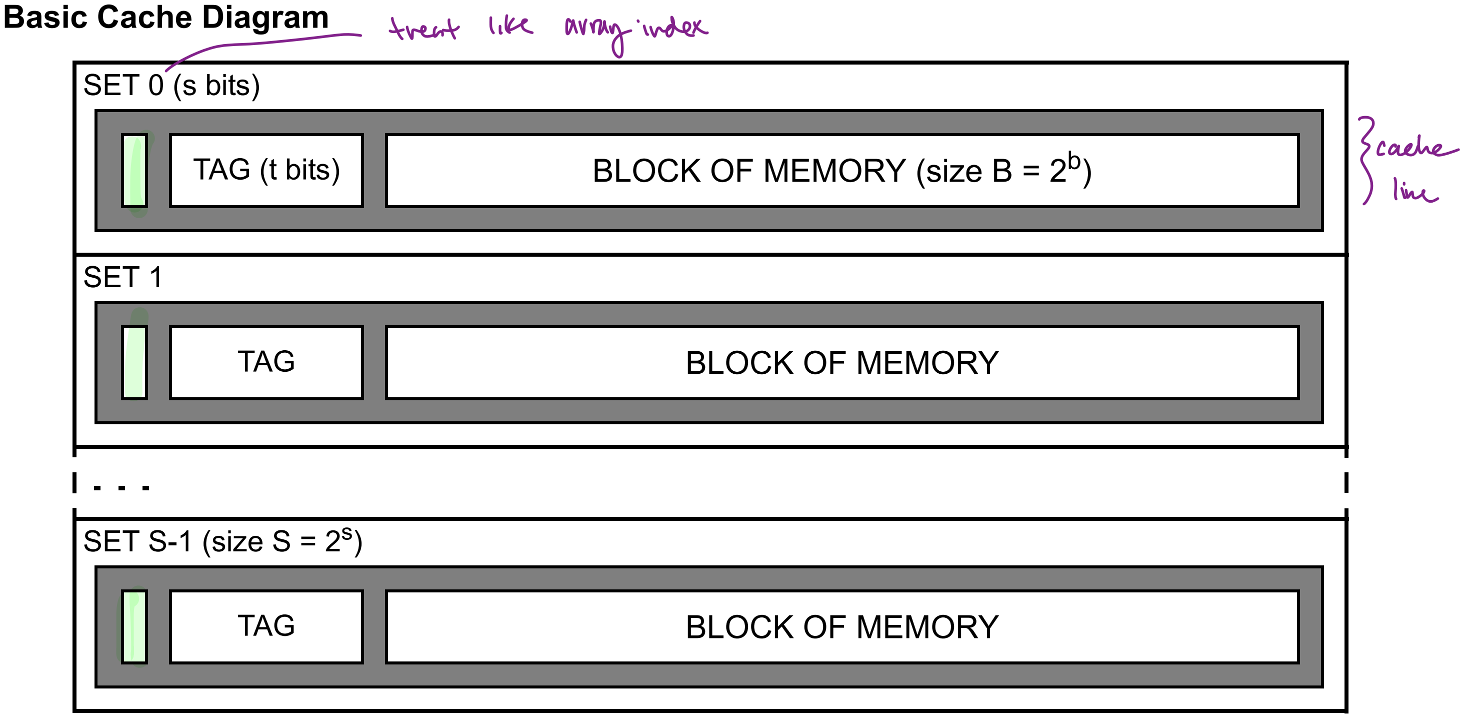
Basic Cache Lines

What? A *line* is

- location in the cache that can store one block of memory
- composed of storage for the block and info needed for cache operation

✵ *In our basic cache each cache set* has only one line

Basic Cache Diagram



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

use a status bit (v bit)

if v = 1, cache block is copied to cache line

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

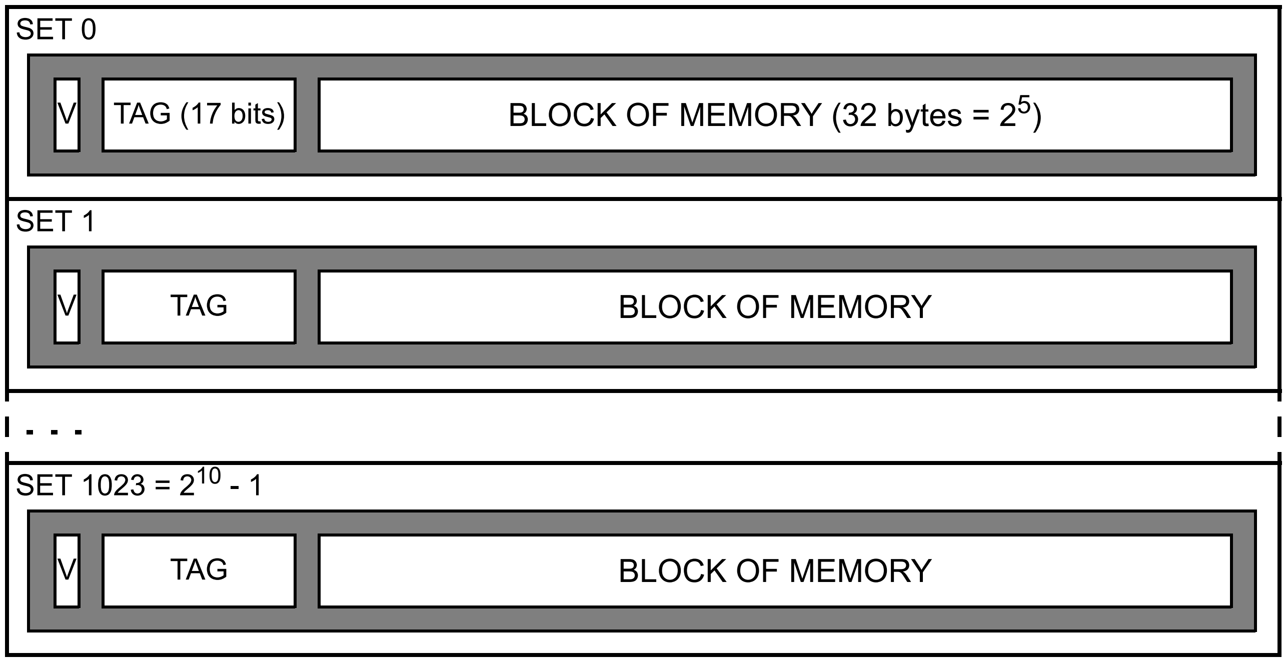
$C = S \times B$

# of bytes — # of sets — bytes/block — b/c 1 block/set

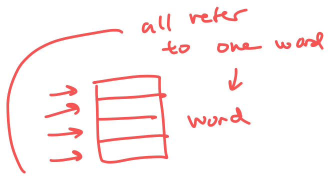
tag and v-bit are not included in cache size

Basic Cache Operation

Basic Cache Diagram

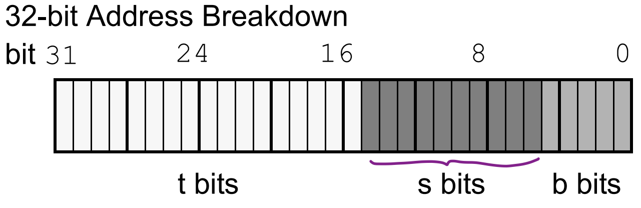


→ How big is this basic cache? $C = S \times B = \underset{2^{10}}{1024} \times \underset{32}{2^5} = 2^{15}$ bytes $= 32$ Kb



How does a cache process a request for a $\overset{\text{4-bytes}}{\underline{\textbf{word}}}$ at a particular address?

1. *Set Selection* identify the set
   extract s-bit
   use as index
2. *Line Matching* extract t-bits
   compare t-bits w/ stored tag in line of the set



if no match or valid bit is 0 , cache miss

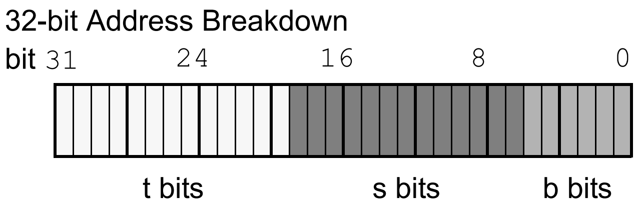
if match and valid bit is 1 , cache hit

on HWs & Exams, questions will be
"is this addr in the cache?"

For L1 cache only , must now extract word from block using word offset

Basic Cache Practice

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



→ How big are the blocks?

$B = 2^6 = 2^6 = 64$ bytes / block

→ How many sets?

$S = 2^s = 2^{13} = 8192$ sets / cache

→ How big is this basic cache?

1 block / cache line

$C = S \times B = 2^{13} \times 2^6 = 2^{19} = 2^9 \cdot 2^{10} = 512K$

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

memorize hex → bin

0 7 5 1 5 E 2 B

0000 0111 0101 0001 0101 1110 0010 1011

word offset

t-bits s-bits b-bits

→ Which set should be checked given the address above? s-bits

0010 1011 1100 0 = 8 + 16 + 32 + 64 + 256 + 1024

= 1400_{10} ∴ set index is 1400

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

1010

10 1011 → = word 10 of 64 byte block

which word byte

➢ Which byte in the word does the address specify?

11 = byte 3 of word 10 of our block

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?

1) blocks, bytes, data
2) tag - which block
3) V - valid or not

	V	tag		
1.)	1	0x0750	MISS	- wrong tag
2.)	0	0x0750	〃	〃
[✓ 3.)]	1	0x00EA	HIT	0 0000 1110 1010 / 0 0 E A
4.)	0	0x00EA	MISS - Not valid	