CS 354 - Machine Organization & Programming
Tuesday March 5, and Thursday Mar 7, 2024

Print paper copies of this outline for best use.

Week 07 Activity: Heap Practice Assignment

Project p3A: DUE on or before Friday 3/8 & p3B on 3/15

Homework 3: DUE on or before Monday 3/4**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.
- Identify and describe case with 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

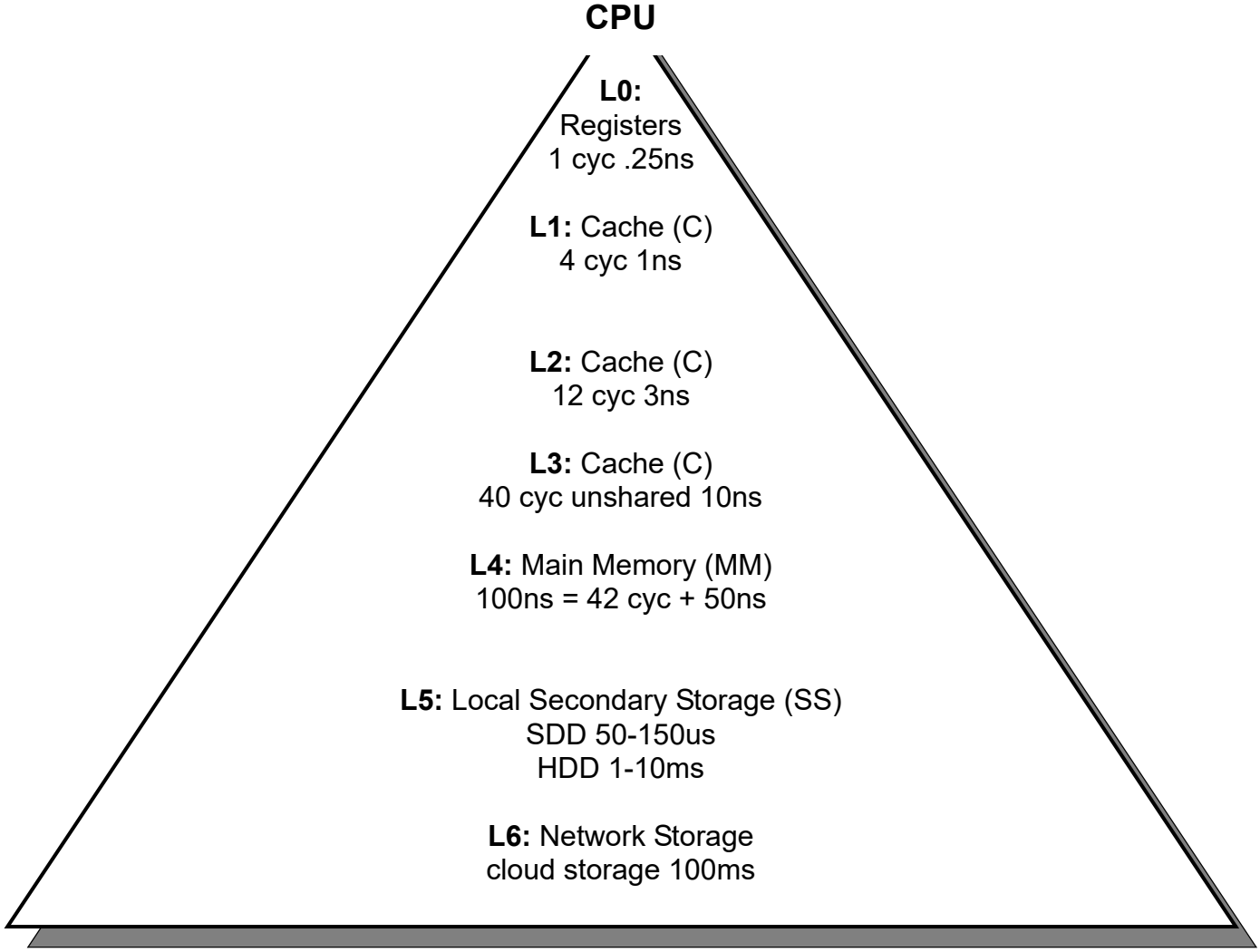
Memory Hierarchy Locality & Caching Bad Locality Caching: Basic Idea & Terms Designing a Cache: Blocks	Rethinking Addressing Designing a Cache: Sets and Tags Basic Cache Lines Basic Cache Operation Basic Cache Practice
Next Week: Vary cache set size and Cache Writes B&O 6.4.3 Set Associative Caches 6.4.4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

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 recently used location is repeatedly used in the future

on the same pallet

spatial locality: when recently used location is followed by access of nearby location

locality is designed into Hardware
OS
Applications - programs

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.

sum, i, step, size

→ List the variables that clearly demonstrate spatial locality.

a[i]

stride: step size in words between sequential access in the block

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

How?

temporal: anticipates data location will be reused
spatial: anticipates nearby data will be used

cache block: unit of memory transfer between main memory and cache levels
typical cache block size is 32 bytes per block

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

Why?

programs with good locality maximise the use of data at the TOP of the memory hierarchy

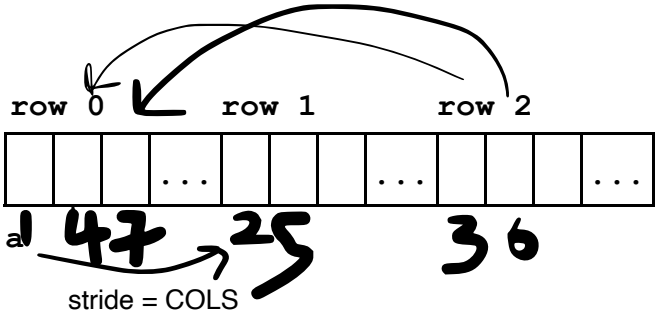
Bad Locality

Why is this code bad?

ROWS = 3
COLS = ?

```
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?

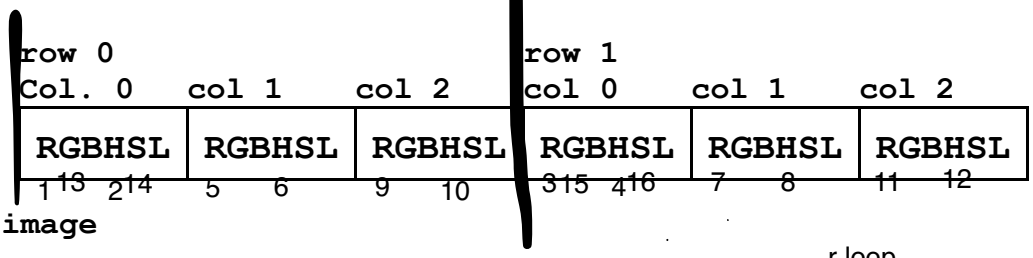
Java, C row major oriented language — row data grouped tgt swap row to outer loop and col to inner loop

Key Questions for Determining Spatial Locality:

1. What does the memory layout look like for the data? recall 2D SAA, **[], **x

2. What is the stride of the code across the data? # of columns, smaller is better

Why is this code bad?




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

          row       col
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;
      }
```

stride = 3, 15, 3, -15, … ==

r loop
  c loop
    v loop
      rgb
      hsl

3 word stride

r loop
  c loop
    v loop
      rgb
    v loop
      hsl

to get one word stride

➢ How would you improve the code to reduce stride?

v loop innermost, then c middle, r outermost, second v loop

Good or bad locality?

- Instruction Flow:
  - sequencing? good spatial, not good temporal since going to next instruction each time
  - selection? if else, etc… bad spatial and bad temporal
  - repetition? good temporal, good spatial if stride is small

- Searching Algorithms:
  - linear search — good spatial if linear array, bad spatial if linked list, tree
  - binary search — array based — not good spatial

} some temporal locality for tracking variables, and matching variables

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.

in this context, 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

cache miss
when block is not in cache

C cold miss when cache location is empty

X capacity miss
when cache is too small for working set

F conflict miss
when 2 or more blocks map to same location

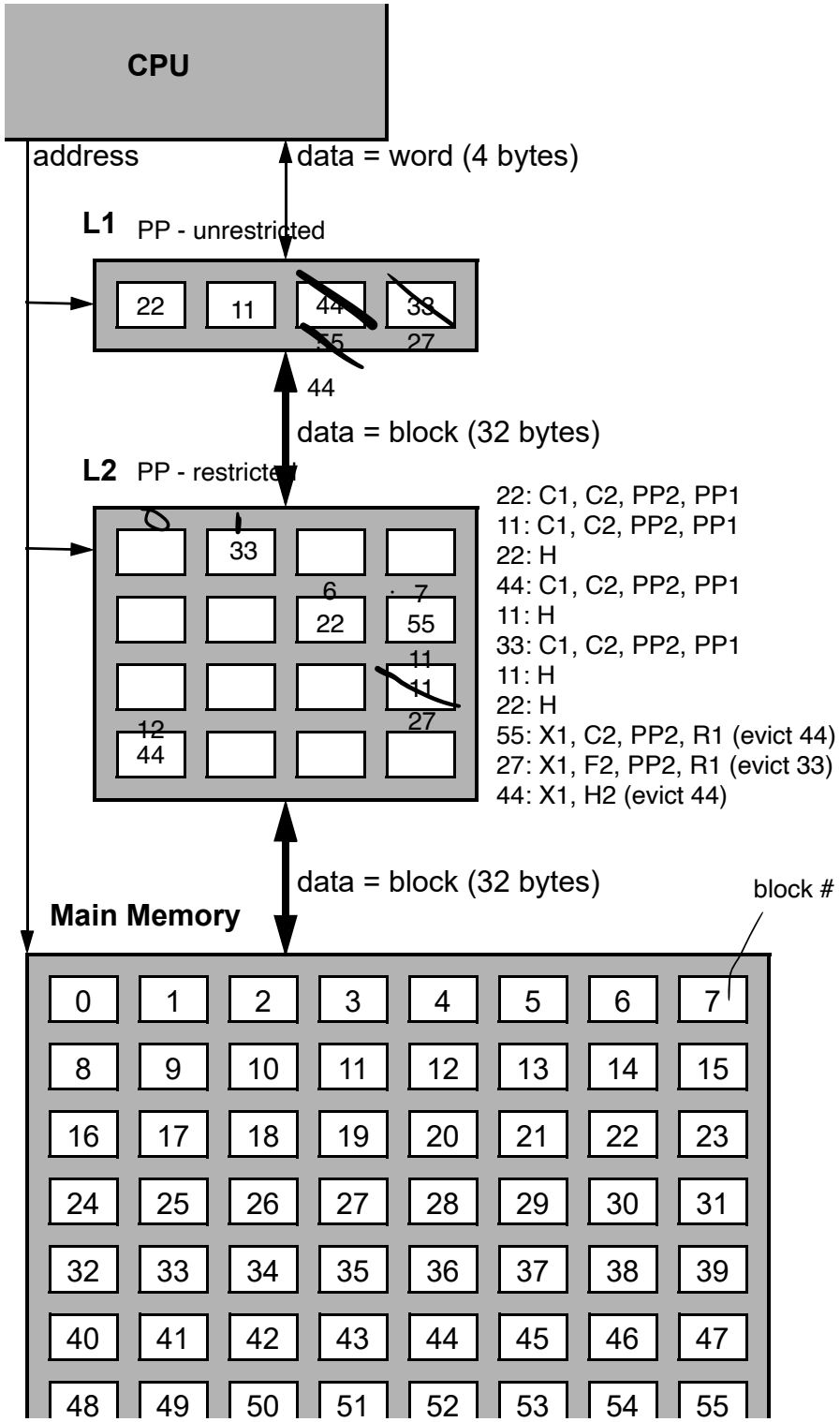
cache hit
when block is in cache

placement policies
1. unrestricted — can go anywhere (L1)
2. restricted — mapped to set of lcoations (L2)
block # % 16

replacement policies
1. choose any block location (L1)
2. restricted (L2)
block # % 16

victim block
cache block chosen to be replaced

working set
set of blocks accessed given some interval of time



Designing a Cache: Blocks

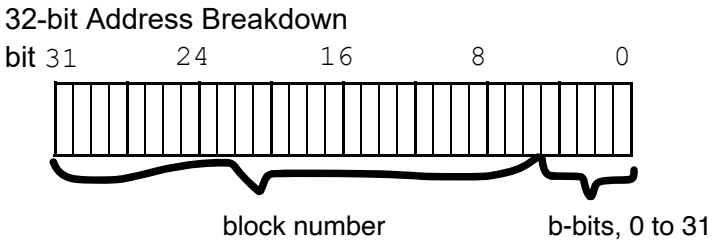
❋ *The bits of an address* are used to "lookup" if block containing that address is in cache or not

How many bytes in an address space?

Let M is number of bytes — IA-32 4GB

$M = 2^m$

$m = \log_2 M$ = number of bits in address



Thus m is the number of bits in address — IA-32 is 32 bits address

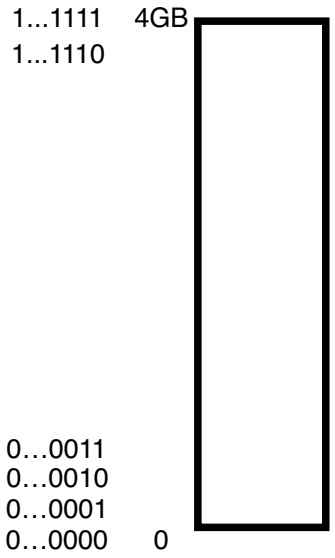
How big is a block?

❋ *Cache blocks must be big enough* to capture spatial locality
but small enough to minimize latency

Let B be the number of bytes per block — IA-32 32 bytes per block

$B = 2^b$ = 32 bytes per block

$b = \log_2 B$ = 5



b bits: the number of bits of address to determine which byte of block

word offset identifies which word of the block

byte offset identifies which byte of the word

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

LOSE SPATIAL LOCALITY, all byte addr near each other wont be in the same block,
imagine 11111…..11111111 with first 5 bits as b-bits, next addr below is 11111…11111110
and since first 5 is b bits, rest is block number, means these two are diff block number

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

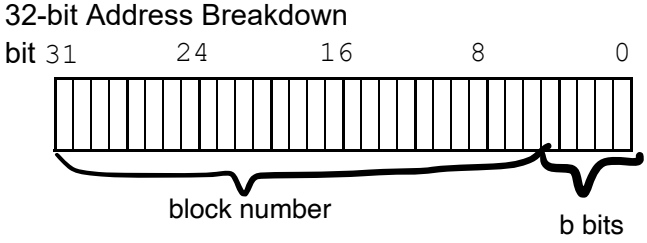
VAS / B = 2^32 / 2^5 = 2^27 = 2^7 * 2^20 = 128 Mb (Megabytes)

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

Rethinking Adressing

- *An address identifies* which byte in the Virtual Address Space to access
- *An address is* divided into parts to access memory in steps

32-bit Address Breakdown

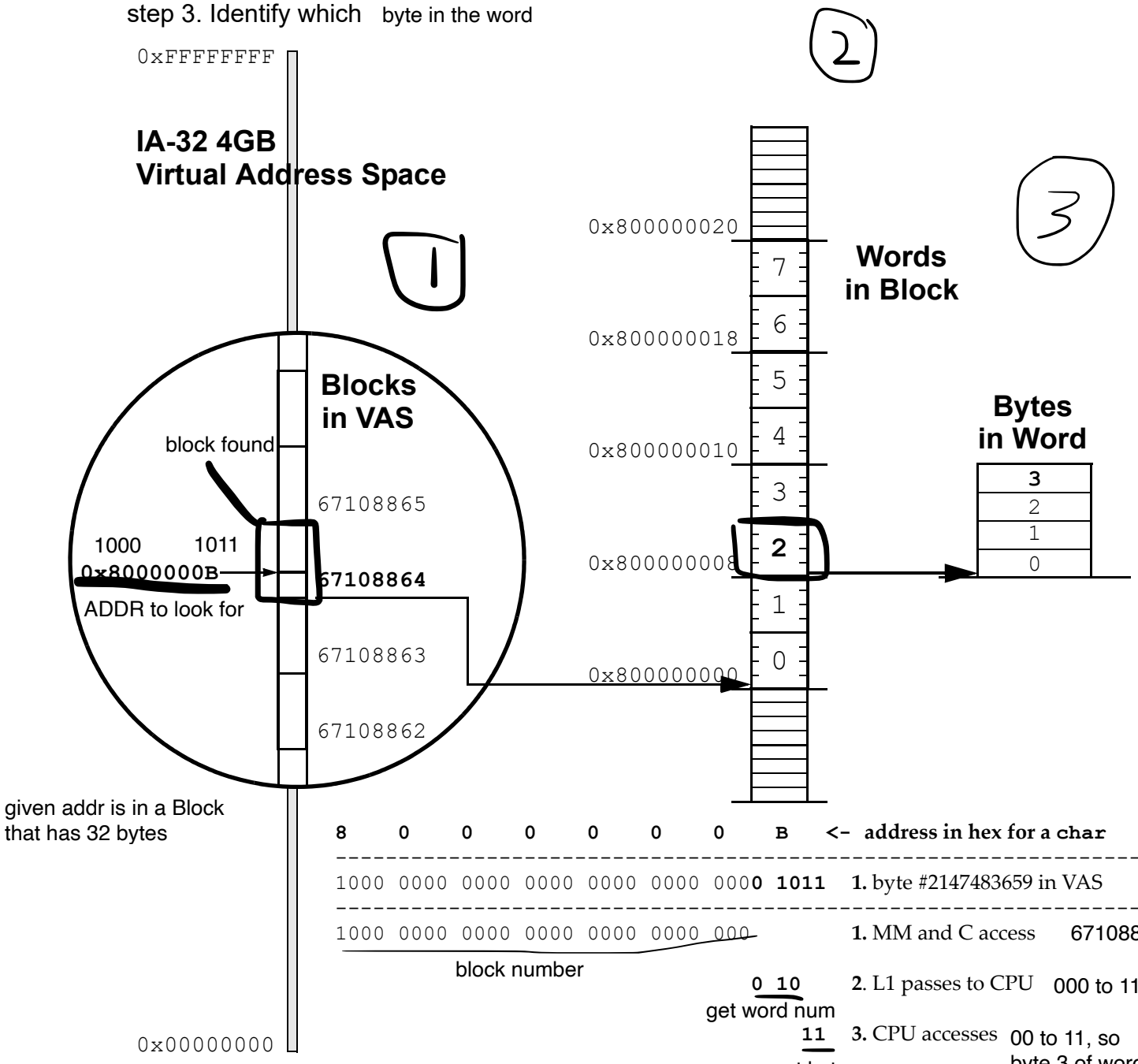


Memory Access in Caching System

step 1. Identify which block in Virtual Address Space

step 2. Identify which word in the Block

step 3. Identify which byte in the word



given addr is in a Block that has 32 bytes

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 67108864

block number

0 10 **2.** L1 passes to CPU 000 to 111, so word 2

get word num

11 **3.** CPU accesses 00 to 11, so byte 3 of word 2 of block 67108864

get byte num

Designing a Cache: Sets & Tags

❋ *A cache must be searched* if unrestricted policy (L1 example on earlier page)

→ Problem? slow O(n) linear search, where n is the number of locations

Improvement? limit the locations where block can be found

set: where a block of memory is uniquely mapped

❋ *The block number bits of an address* are divided into two parts

1. "set" — maps block to specific set in cache
2. "tag" — uniquely identifies a particular block within a set

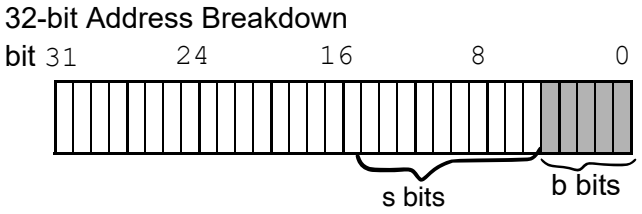
How many sets in the cache?

Let S be the number of sets in cache

$S = 2^s$

$s = \log_2 S$ the number of bits we need for the sets

if S = 1024, s = 10, if S = 8192, s = 13



s bits: bits that identifies which set block maps to

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

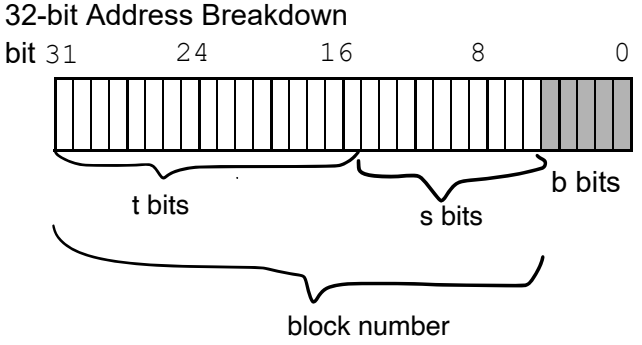
we dont want consectuive blocks to be in the same set, imagine array, we want consecutive blocks in sequential sets, not mapped to same set and displaced each other => LOST SPATIAL LOCATLITY

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

(VAS / B) / S = 2^27 / 2^10 = 2^(27-10) = 2^17 = 2^7 * 2^10 = 128K = 131072 block maps to each set

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

use remaining bits as unique tag



t bits: bits of address that identify which block in set

❋ *When a block is copied into a cache*

its t bits are also stored

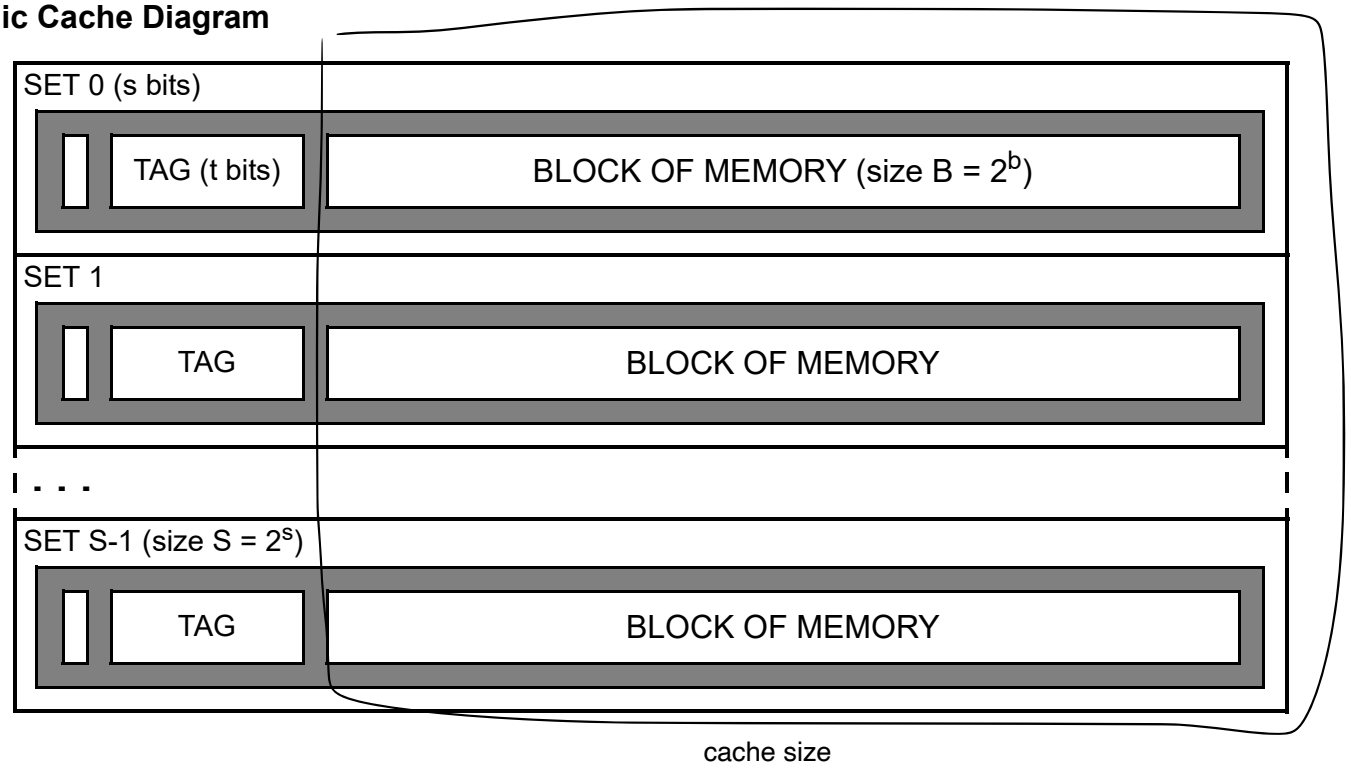
Basic Cache Lines

What? A *line* is

- a location in cache that can store one block of memory (and tag)
- composed of storage for block data and info needed for cache operation

❋ *In our basic cache each cache set* has only one cache 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-bit = 1, valid cache line

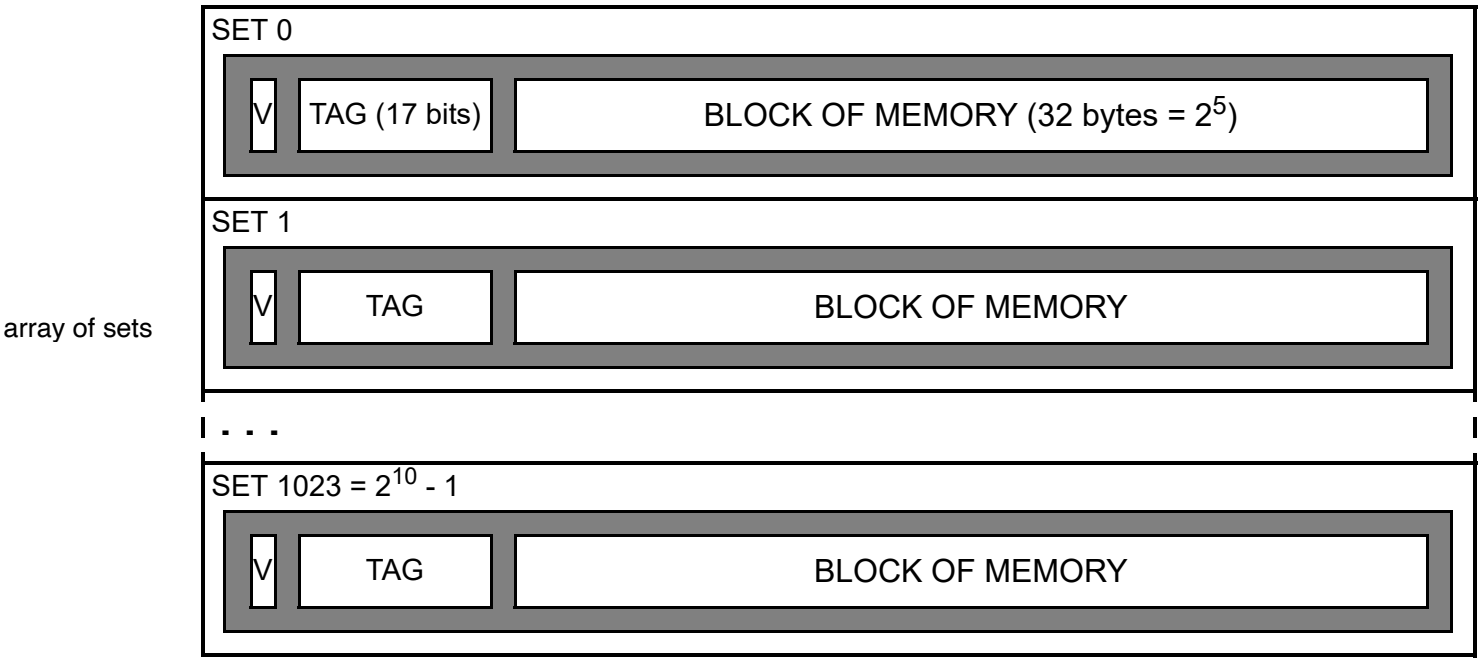
→ How big is a basic cache given S sets with blocks having B bytes?

Cache C = S * B
size in bytes = number sets * block size
does not include bits of overhead

Basic Cache Operation

Basic Cache Diagram

array of sets represents cache



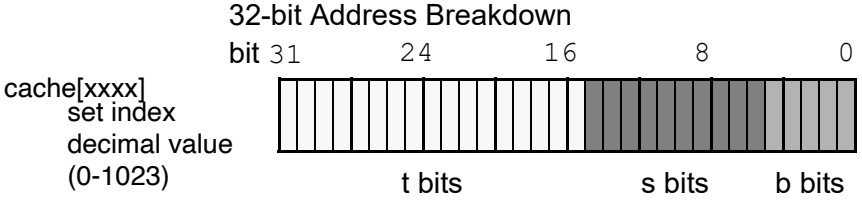
→ How big is this basic cache?

C = S * B = 2^10 * 2^5 = 2^15

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

address tells u where to look in the cache

1. *Set Selection*
   extract s bits,
   use as index



2. *Line Matching*

   extract t bits and compare t bits with tag for the line

   have to match both the tag and have valid v bit

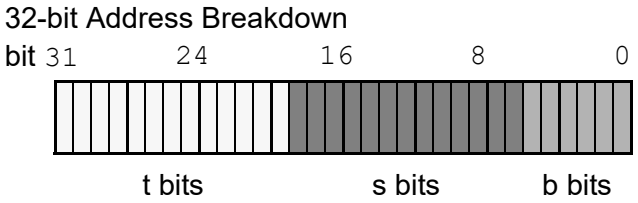
   if no match or valid bit is 0 cache miss

   if match and valid bit is 1 cache hit

For L1 cache only (highest level cache), must extract correct WORD from the BLOCK

Basic Cache Practice

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



→ How big are the blocks?

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

→ How many sets?

2^13 = 2^3 * 2^10 = 8Kb

→ How big is this basic cache?

C = S * B = 2^13 * 2^6 = 2^19 = 2^9 * 2 ^10 = 512 Kb

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

0000 0111 0101 0001 0101 1110 0010 1011

0000 0111 0101 0 001 0101 1110 00 10 1011

→ Which set should be checked given the address above?

set: 0 0101 0111 1000 = 8 + 16 + 32 + 64 + 256 + 1024 = 1400

set index = 1400

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

b bits: 1010 11 word index = 10

4 bytes in word, 16 words in block

➢ Which byte in the word does the address specify?

byte index = 3

byte 3 of word 10

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	00000 1110 1010 -> EA
2.)	0	0x0750	
3.)	1	0x00EA	3 since v bit must be valid
4.)	0	0x00EA	