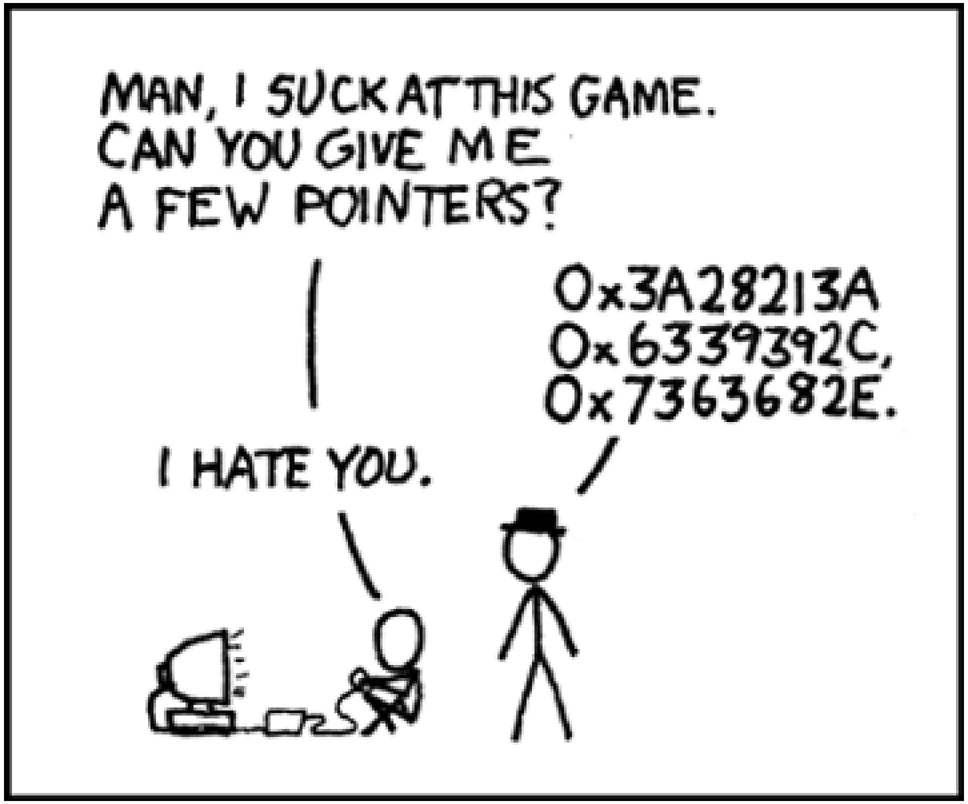
Cache Lab Implementation and Blocking

Lou Clark

February 24th, 2014

Welcome to the World of Pointers !



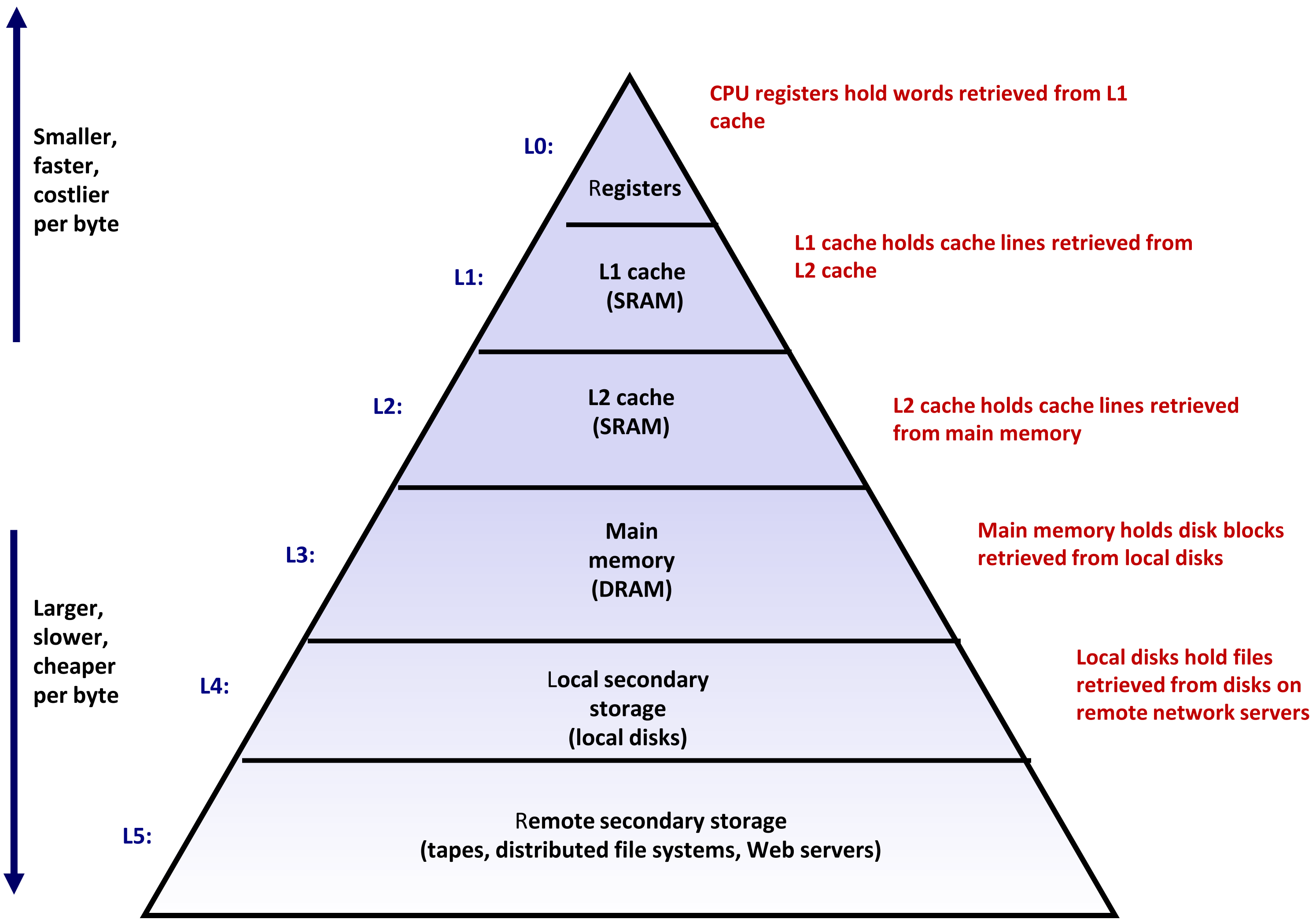
Class Schedule

- **Cache Lab**
  - Due Thursday.
  - Start soon if you haven't yet!
- **Exam Soon !**
  - Start doing practice problems.
  - Mon March 3rd – Wed March 5th

Outline

- **Schedule**
- **Memory organization**
- **Caching**
  - Different types of locality
  - Cache organization
- **Cachelab**
  - Part (a) Building Cache Simulator
  - Part (b) Efficient Matrix Transpose

Memory Hierarchy



Memory Hierarchy

- Registers
- SRAM
- DRAM
- Local Secondary storage
- Remote Secondary storage

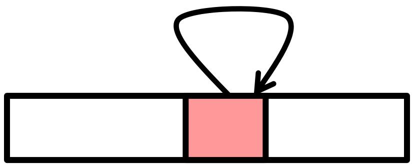
We will discuss this interaction

SRAM vs DRAM tradeoff

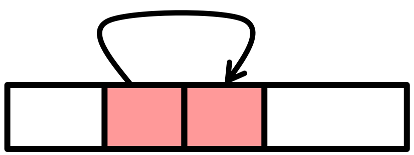
- **SRAM (cache)**
  - Faster (L1 cache: 1 CPU cycle)
  - Smaller (Kilobytes (L1) or Megabytes (L2))
  - Uses 4-6 transistors per bit
  - More expensive
- **DRAM (main memory)**
  - Relatively slower (hundreds of CPU cycles)
  - Larger (Gigabytes)
  - Uses 1 transistor, 1 capacitor per bit
  - Cheaper

Locality

- **Temporal locality**
  - Recently referenced items are likely to be referenced again in the near future
  - After accessing address X in memory, save the bytes in cache for future access

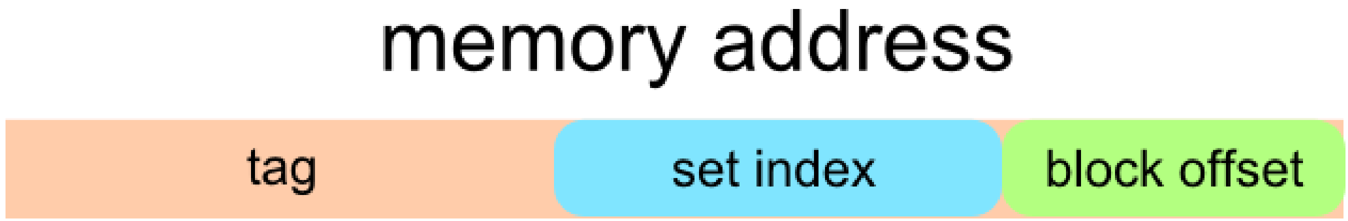


- **Spatial locality**
  - Items with nearby addresses tend to be referenced close together in time
  - After accessing address X, save the block of memory around X in cache for future access



Memory Address

- **64-bit on shark machines**



- **Block offset: b bits**
- **Set index: s bits**
- **Tag Bits: Address Size – b – s**

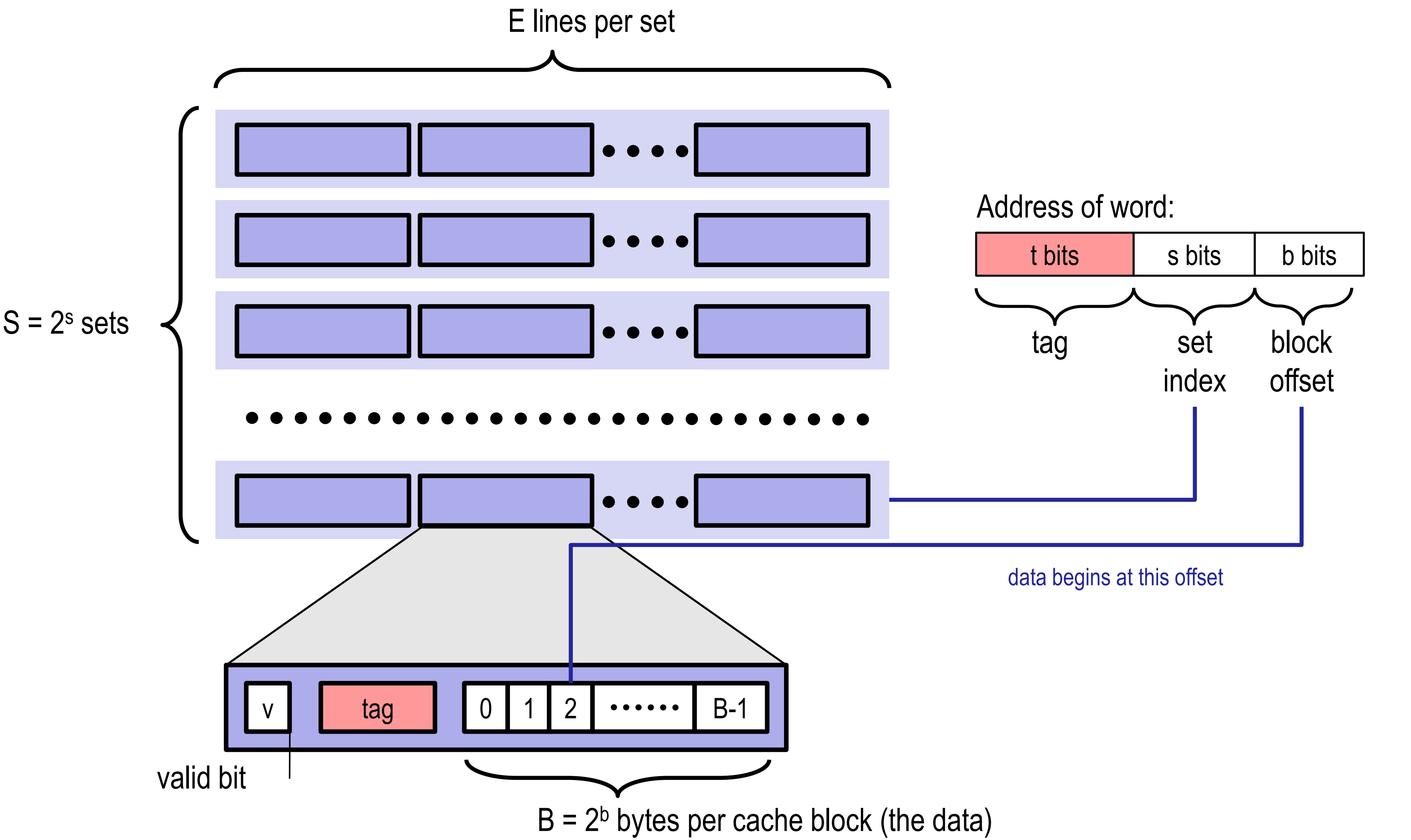
Cache

- **A cache is a set of 2^s *cache sets***

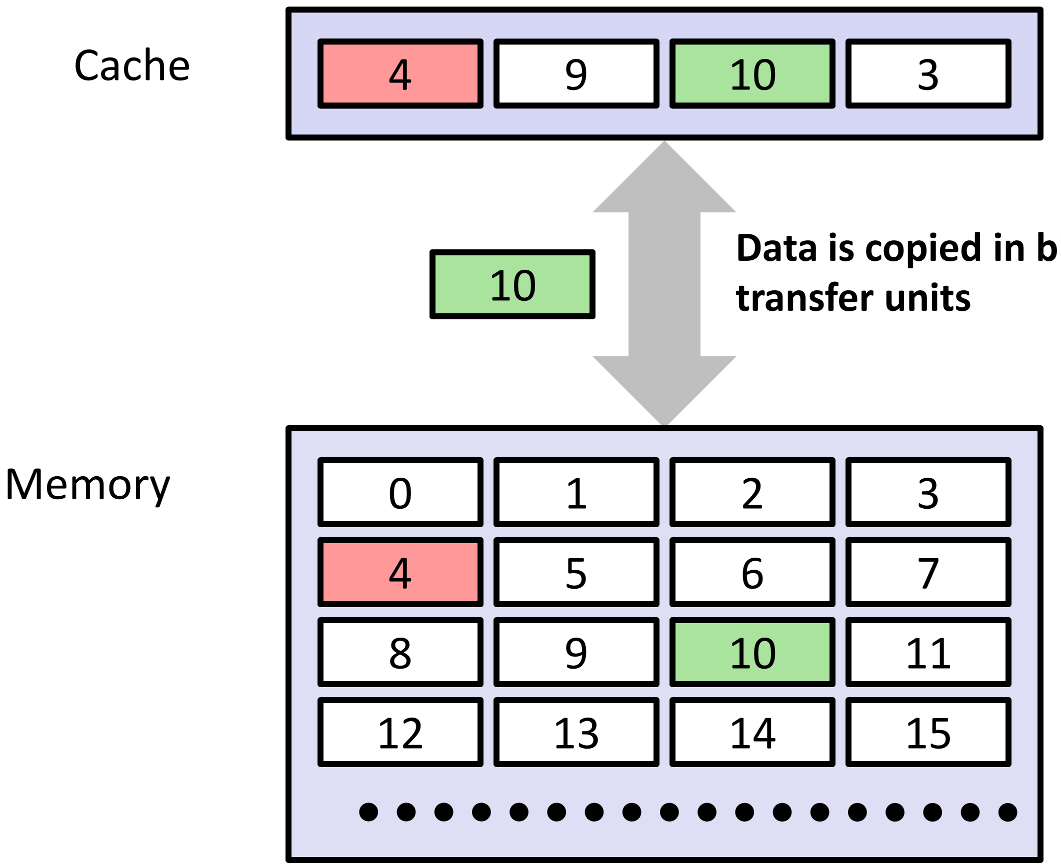
- **A *cache set* is a set of E *cache lines***
  - E is called associativity
  - If E=1, it is called "direct-mapped"

- **Each *cache line* stores a block**
  - Each block has B = 2^b bytes
- **Total Capacity = S*B*E**

Visual Cache Terminology



General Cache Concepts

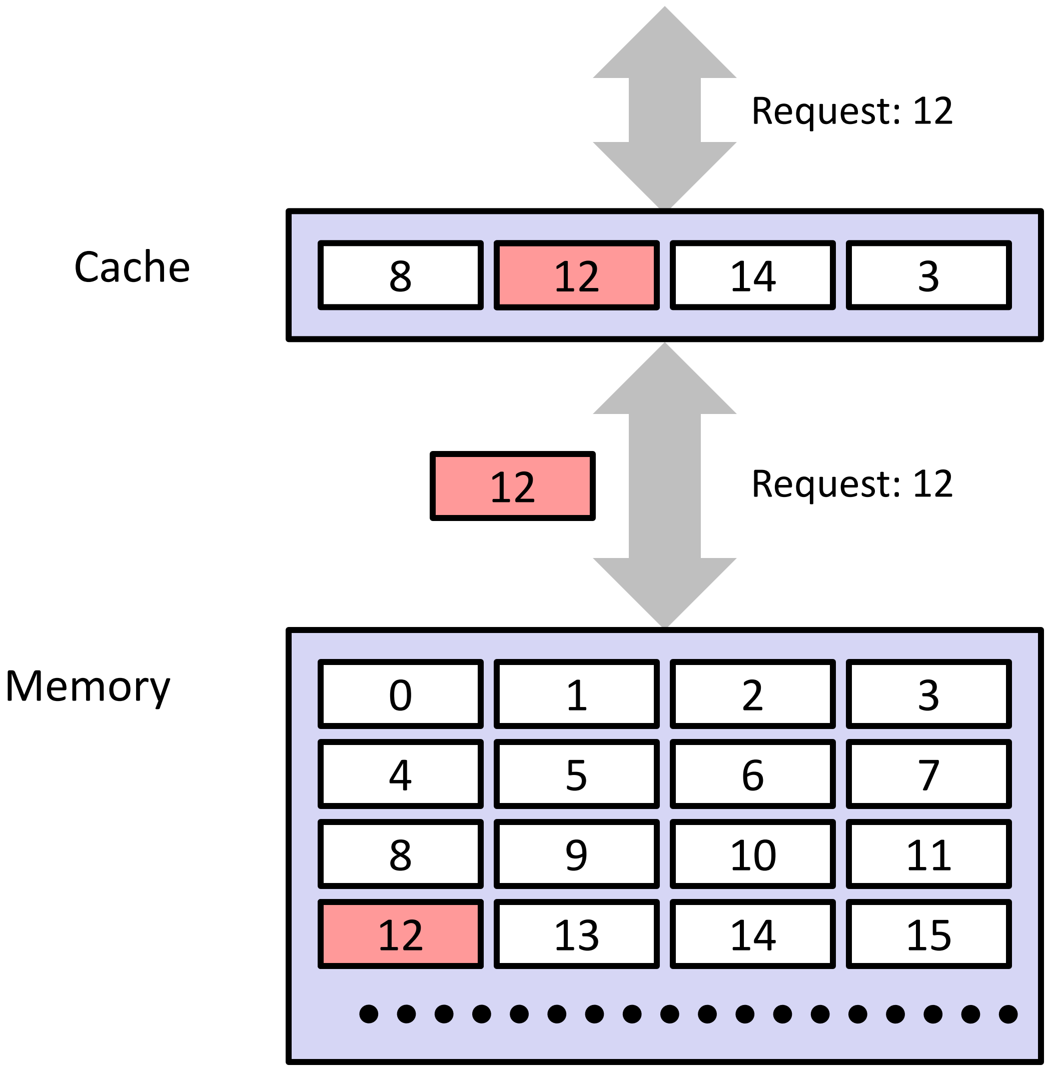


Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into "blocks"

General Cache Concepts: Miss



Data in block b is needed

Block b is not in cache:
Miss!

Block b is fetched from *memory*

Block b is stored in cache
- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

General Caching Concepts: Types of Cache Misses

- **Cold (compulsory) miss**
  - The first access to a block has to be a miss
- **Conflict miss**
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block
    - E.g., Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time with direct mapping
- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache

Cachelab

- Part (a) Building a cache simulator

- Part (b) Optimizing matrix transpose

Part (a): Cache simulator

- **A cache simulator is NOT a cache!**
  - Memory contents NOT stored
  - Block offsets are NOT used – the b bits in your address don't matter.
  - Simply **count** hits, misses, and evictions

- **Your cache simulator need to work for different s, b, E, given at run time.**

- **Use LRU – Least Recently Used replacement policy**
  - Evict the least recently used block from the cache to make room for the next block.
  - Queues ? Time Stamps ?

Part (a): Hints

- **Structs are a great way to represent a cache line. Each cache line has a:**
  - Valid bit
  - Tag
  - LRU counter (if you are not using a queue)

- **A cache is just 2D array of *cache lines*:**
  - struct cache_line cache[S][E];
  - S = 2^s, is the number of sets
  - E is associativity

Part (a): getopt

```
./foo -x 1 -y 3
```

- **getopt() automates parsing elements on the unix command line**
  - Typically called in a loop to retrieve each flag in turn
  - Returns -1 when there are no more inputs
  - Its return value is the flag it's currently parsing ("x", "y",)
    - Use a switch statement on the return value to take care of each flag separately
    - If a flag has an associated argument ("1", "3"), this **string** is saved in the external variable optarg
  - When getopt() returns -1, there are no more options
  - For more information, look at man 3 getopt
- **To use getopt, your program must include the header file unistd.h**

Part (a): getopt Example

./foo -x 1 -y 3

```
int main(int argc, char** argv){
    int opt, x,y;
    /* looping over arguments */
    while(-1 != (opt = getopt(argc, argv, "x:y:"))){
        /* determine which argument it's processing */
        switch(opt) {
            case 'x':
                x = atoi(optarg);
                break;
            case 'y':
                y = atoi(optarg);
                break;
            default:
                printf("wrong argument\n");
                break;
        }
    }
}
```

Part (a): fscanf

- **The fscanf() function is just like scanf() except it can specify a stream to read from (ie, an open file)**
- **It takes the following parameters:**
  1. a stream pointer (e.g. a file descriptor)
  2. a format string with information on how to parse the file

  3-n. the proper number of **pointers** to the variables you want to store the parsed data in
- **You typically want to use fscanf in a loop. It returns -1 if it hits EOF, or if the data doesn't match the format string**
- **fscanf will be useful in reading lines from the trace files.**
  - L 10,1
  - M 20,1

Part (a): fscanf example

```
FILE * pFile; //pointer to FILE object

pFile = fopen ("tracefile.txt",“r"); //open file for reading

char identifier;
unsigned address;
int size;
// Reading lines like " M 20,1" or "L 19,3"

while(fscanf(pFile,“ %c %x,%d”, &identifier, &address,
&size)>0){
   // Do stuff
}

fclose(pFile); //remember to close file when done
```

Part (a): malloc/free

- **Use malloc to allocate memory on the heap**
- **Always free what you malloc, otherwise may get memory leak**
  - Some_pointer_you_malloced = malloc(sizeof(int));
  - free(some_pointer_you_malloced);
- **Don't free memory you didn't allocate**

Part (a): Tutorials

- **getopt:**
  - Man 3 getopt
  - http://www.gnu.org/software/libc/manual/html_node/Getopt.html
- **fscanf :**
  - man fscanf
  - http://crasseux.com/books/ctutorial/fscanf.html
- **Google is your friend**

Part (b): Efficient Matrix Transpose

- **Matrix Transpose (A -> B)**

Matrix A

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

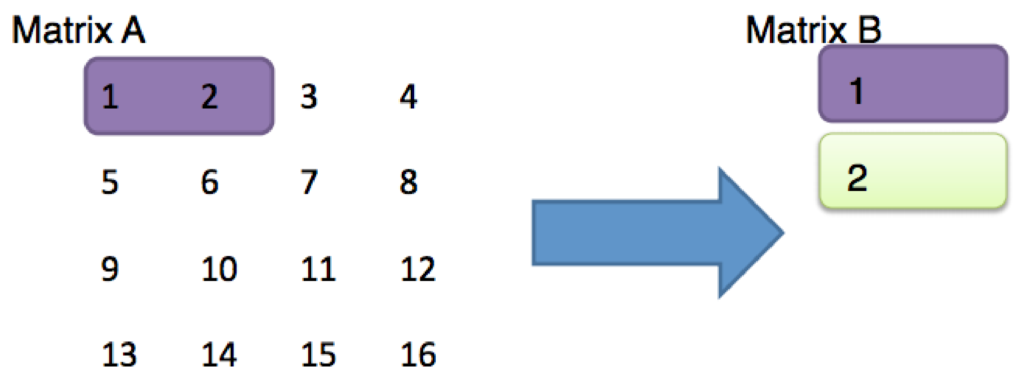
Matrix B

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

- **How do we optimize this operation using the cache?**

Part (b): Efficient Matrix Transpose

- Suppose Block size is 8 bytes ?



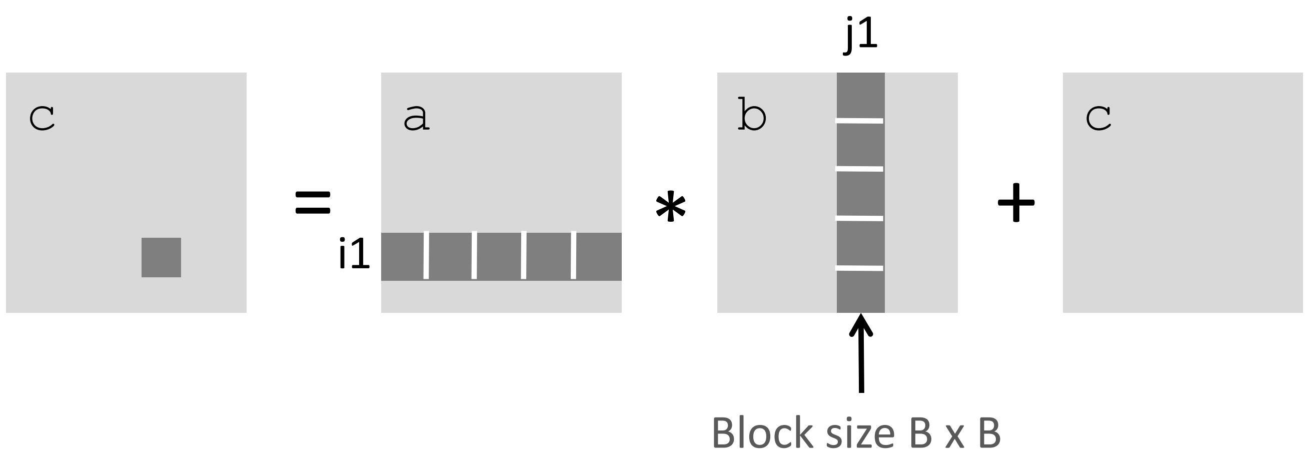
- Access A[0][0] cache miss
- Access B[0][0] cache miss
- Access A[0][1] cache hit
- Access B[1][0] cache miss

Should we handle 3 & 4 next or 5 & 6 ?

Part (b): Blocked Matrix Multiplication

```
c = (double *) calloc(sizeof(double), n*n);

/* Multiply n x n matrices a and b  */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i+=B)
     for (j = 0; j < n; j+=B)
             for (k = 0; k < n; k+=B)
          /* B x B mini matrix multiplications */
                 for (i1 = i; i1 < i+B; i++)
                     for (j1 = j; j1 < j+B; j++)
                         for (k1 = k; k1 < k+B; k++)
                         c[i1*n+j1] += a[i1*n + k1]*b[k1*n + j1];
}
```



Part (b): Blocking

- Blocking: dividing your matrix into sub-matrices
- The ideal size of the sub-matrices depends on your cache block size, cache size, and input matrix size
- Try different sub-matrix sizes

Part (b): Specs

- **Cache:**
  - You get 1 KB of cache
  - Directly mapped (E=1)
  - Block size is 32 bytes (b=5)
  - There are 32 sets (s=5)
- **Test Matrices:**
  - 32 by 32,
  - 64 by 64
  - 61 by 67
  - Your solution need not work on other size matrices

Hint: Warnings are Errors

- **Strict compilation flags**

- **Reasons:**
  - Avoid potential errors that are hard to debug
  - Learn good habits from the beginning

- **Add “-Werror” to your compilation flags**

Hint: Missing Header Files

- **Remember to include files that we will be using functions from**
- **If function declaration is missing**
  - Find corresponding header files
  - Use: man <function-name>
- **Live example**
  - man 3 getopt

Style

- **Read the style guideline**
  - But I already read it!
  - Good, read it again.
- **Pay special attention to failure and error checking**
  - Functions don’t always work
  - What happens when a syscall fails??
- **Start forming good habits now!**

Questions?