System Programming
Lab #5

2018-04-25

sp-tas

Lab Assignment #4 : Cache Lab

- Copy skeleton code from following directory
  # /home/public/cachelab/cachelab-handout.tar

- Hand In
  - make directory for submit (~/submit/cachelab)
  - copy your tarball (userid-handin.tar) into the submit directory
    - Makefile creates a tarball, so put it in the right place
  - Mistakes will take your points

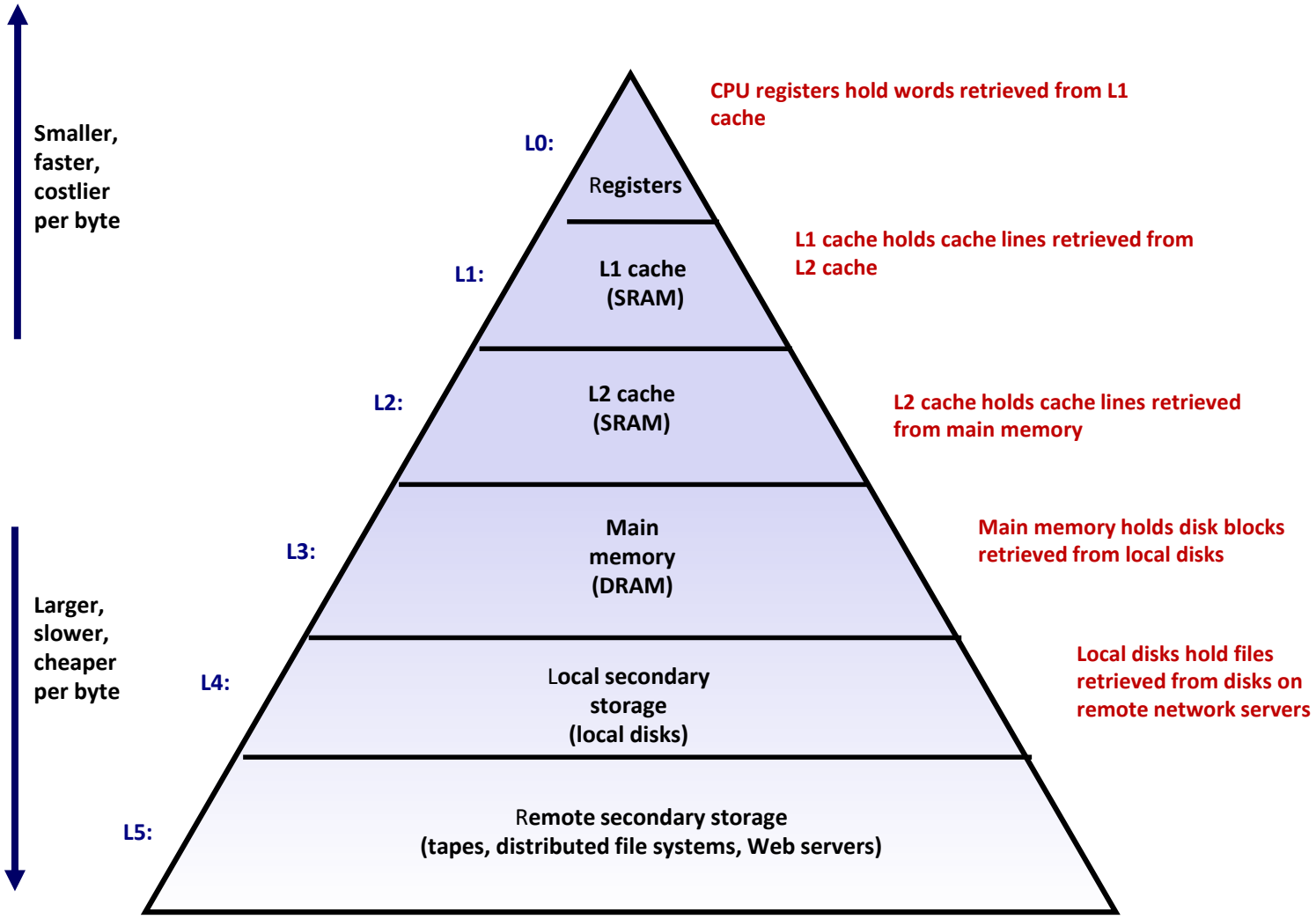
- PLEASE, READ the Hand-out!!! Start early!!!

- Assigned: Apr. 25, 09:00:00 PM
- Deadline: May 9, 04:59:59 PM

Outline

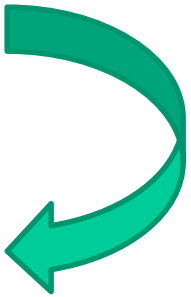
- **Memory organization**
- **Caching**
  - Different types of locality
  - Cache organization
- **Cache lab**
  - Part (a) Building Cache Simulator
  - ~~*Part (b) Efficient Matrix Transpose*~~
  - ~~*Blocking*~~

Memory Hierarchy



Memory Hierarchy

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



We will discuss this interaction

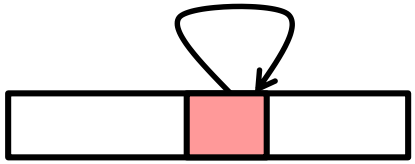
SRAM vs DRAM tradeoff

- **SRAM (cache)**
  - Faster (L1 cache: 1 CPU cycle)
  - Smaller (Kilobytes (L1) or Megabytes (L2))
  - More expensive and “energy-hungry”

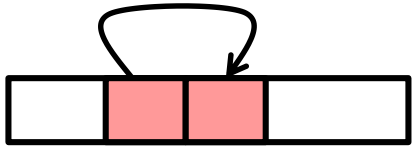
- **DRAM (main memory)**
  - Relatively slower (hundreds of CPU cycles)
  - Larger (Gigabytes)
  - Cheaper

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

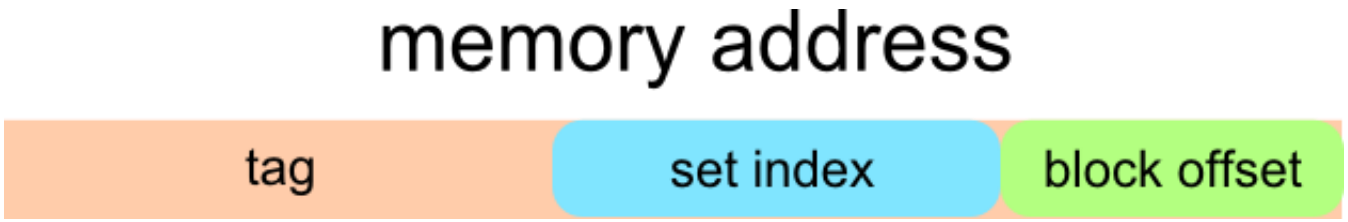


- **Spatial locality**
  - Items with nearby addresses tend to be referenced close together in time
  - After accessing address X, save the block of memory around X in cache for future access



Memory Address

- 64-bit on sysprog machines

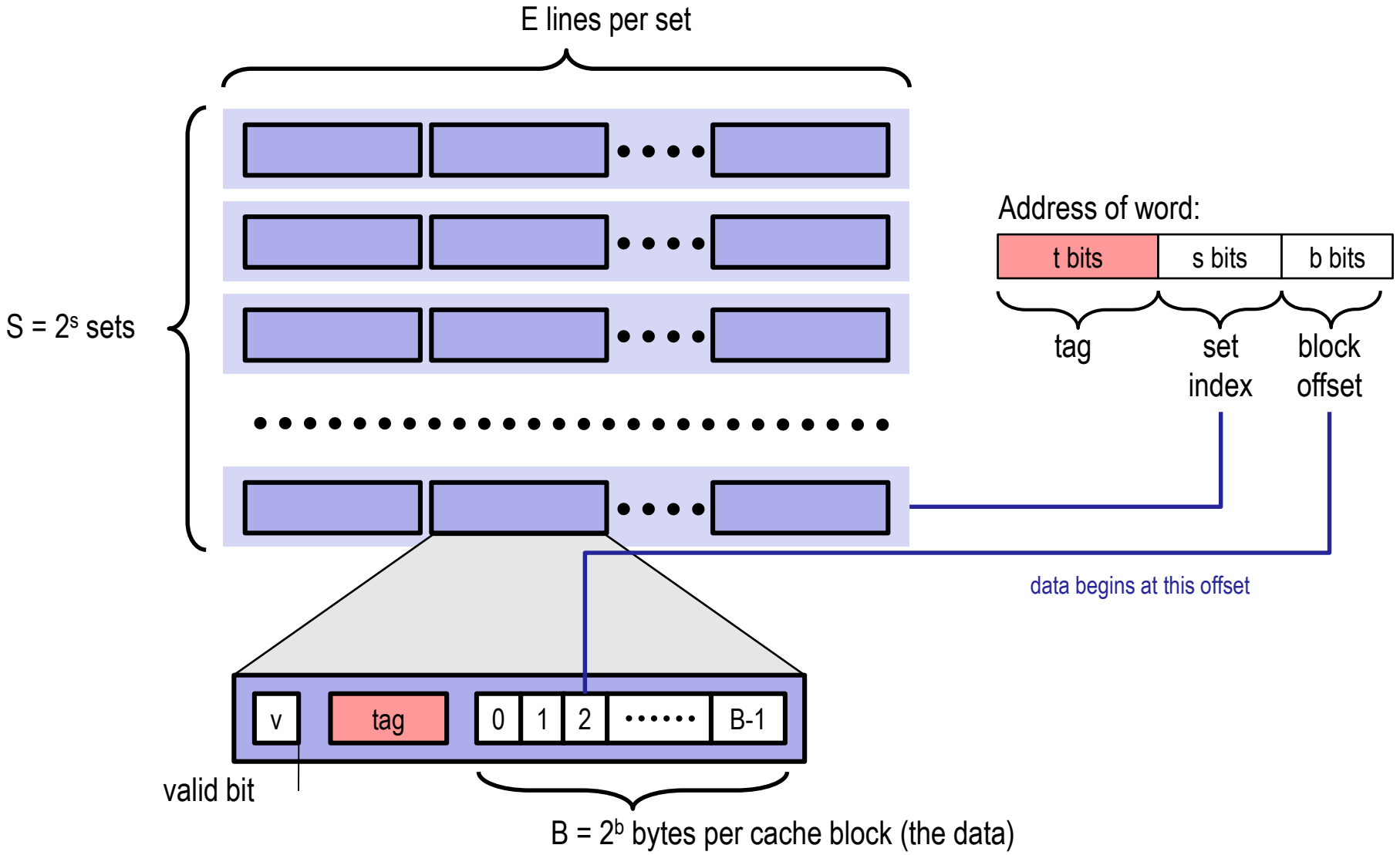


- Block offset: b bits
- Set index: s bits
- Tag Bits: (Address Size – b – s)

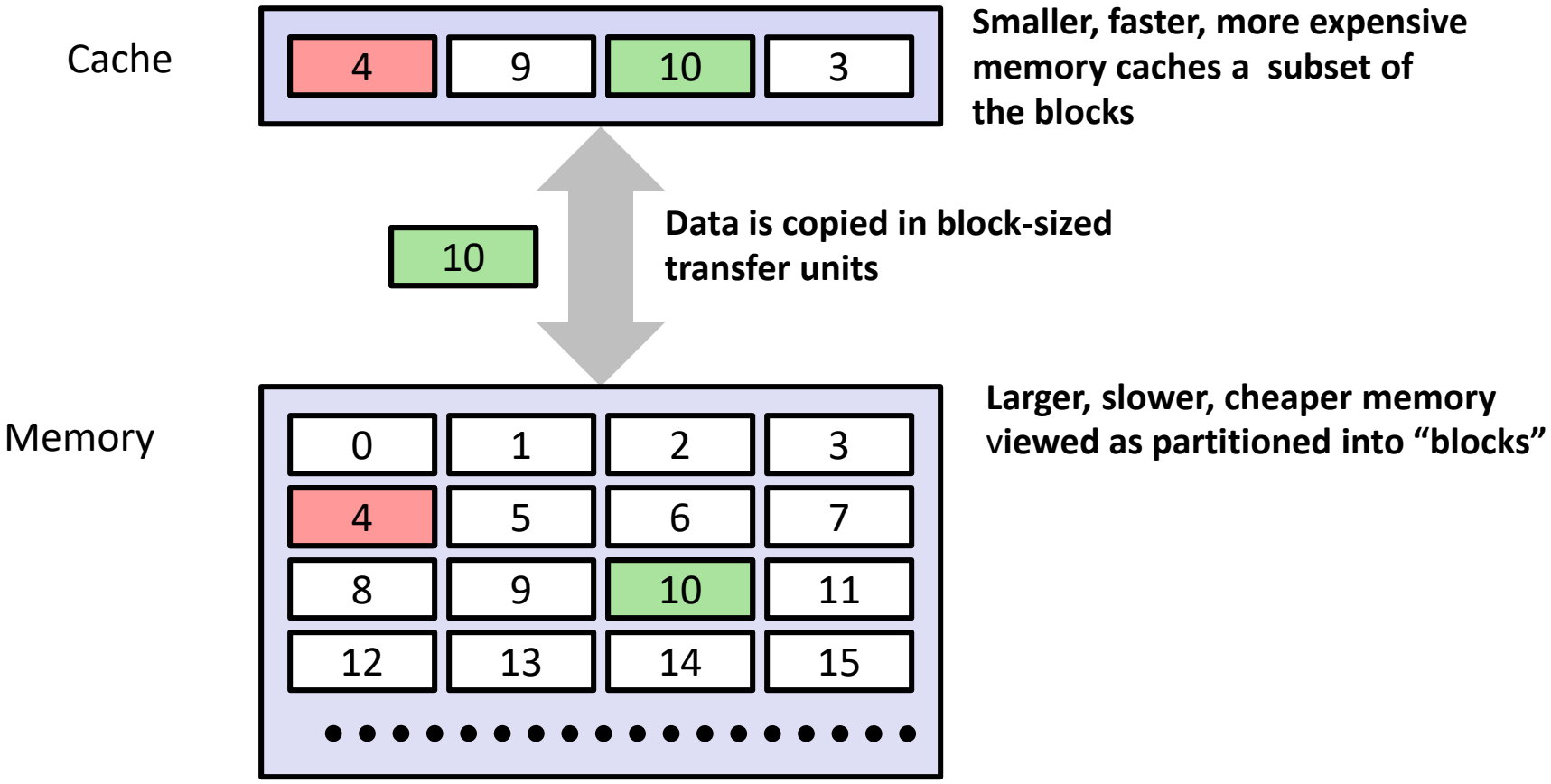
Cache

- **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**

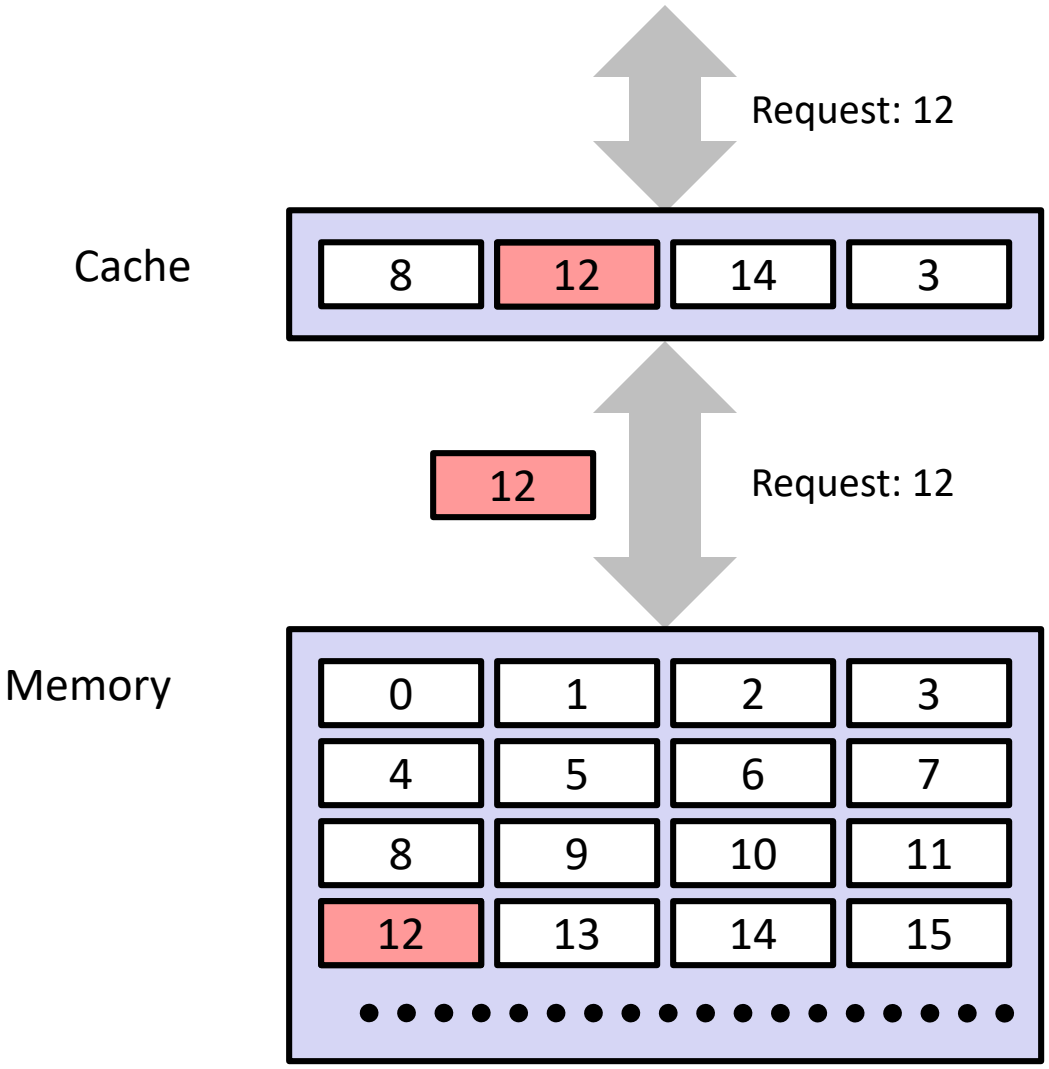
Visual Cache Terminology



General Cache Concepts



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

- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache

Cache Lab

- **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 needs 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

- **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

- **Each cache_line has:**
  - Valid bit
  - Tag
  - LRU counter (only if you are not using a queue)

Part (a) : getopt

- **getopt() automates parsing elements on the unix command line If function declaration is missing**
  - Typically called in a loop to retrieve arguments
  - Its return value is stored in a local variable
  - When getopt() returns -1, there are no more options

- **To use getopt, your program must include the header file**
  #include <unistd.h>

- **If not running on the sysprog machines then you will need #include <getopt.h>.**
  - Better Advice: Run on sysprog Machines !

Part (a) : getopt

- **A switch statement is used on the local variable holding the return value from getopt()**
  - Each command line input case can be taken care of separately
  - “optarg” is an important variable – it will point to the value of the option argument

- **Think about how to handle invalid inputs**

- **For more information,**
  - look at man 3 getopt
  - http://www.gnu.org/software/libc/manual/html_node/Getopt.html

Part (a) : getopt Example

```
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;
        }
    }
}
```

- Suppose the program executable was called "foo". Then we would call "./foo -x 1 –y 3" to pass the value 1 to variable x and 3 to y.

Part (a) : fscanf

- **The fscanf() function is just like scanf() except it can specify a stream to read from (scanf always reads from stdin)**
  - parameters:
    - A stream pointer
    - format string with information on how to parse the file
    - the rest are pointers to variables to store the parsed data
  - You typically want to use this function in a loop. It returns -1 when it hits EOF or if the data doesn't match the format string
- **For more information,**
  - man fscanf
  - http://crasseux.com/books/ctutorial/fscanf.html
- **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**

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**
  - Already added in Makefile of the handout, so don't modify

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

Questions?