ANITA'S SUPER AWESOME RECITATION SLIDES

15/18-213: Introduction to Computer Systems

Memory and Caches, 30 Sept 2013

Anita Zhang

Up to Speed Yet?

- Buflab
  - Due tomorrow, 11:59 PM
  - Your late days are wasted here
- Cachelab
  - Out tomorrow, 11:59 PM
  - Due Thursday, October 10, 2013, 11:59 PM
    - Labs will be going back to regular Thursday due date

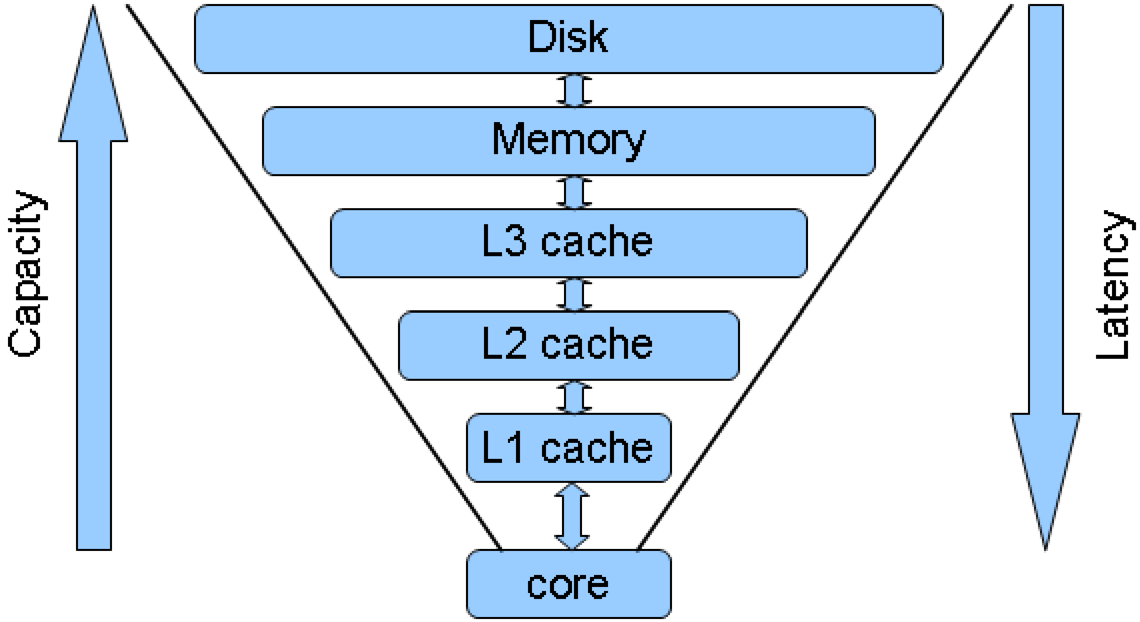
This and That and What's Today

- Exam Talk
- Alignment
- Memory Organization
- Cachelab Part A
  - Data Lab Style Masking
  - Helpful Functions

Impending Doom

- **Midterm: Wed, 16 Oct – Sat, 19 Oct 2013**
  - In 2 weeks!!
- The #1 best way to prepare: do previous exams.
  - Do enough midterms until you feel comfortable with the material (at least 5 recent ones).
    - Depending on the semester, caches can be found in Exam 2

Motivation: Why Bother with the ECEs?



Structs, what are they?

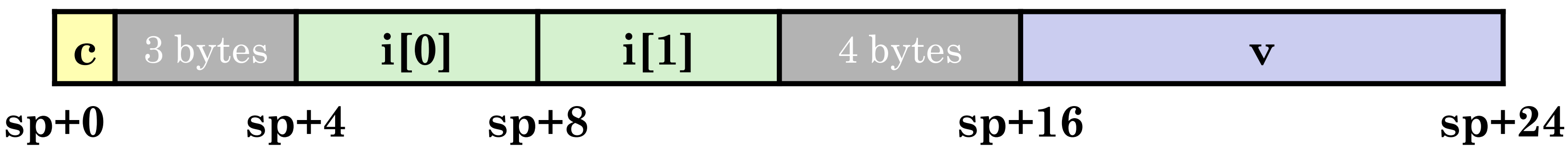
- An object with sets of (related) values that can be passed around together
- Values not necessarily contiguous in memory
  - But they are considered “next to each other”
    - Alignment padding throws off contiguousness
  - Each object may have a different alignment rule
  - Constant offset from the beginning of the struct

Alignment of structs

- Entire struct aligns according to the largest alignment constraint of its member.
  - Enforced by the compiler.
    - Different compilers have different alignment rules!
    - Luckily we only use GCC in this class.
- Overall structure length a multiple of K.
  - K → largest alignment requirement of an element.
  - Optimize length by declaring largest elements first.

Example of a struct (from Lecture)

```
struct S1 {
  char c;
  int i[2];
  double v;
} *sp;
```

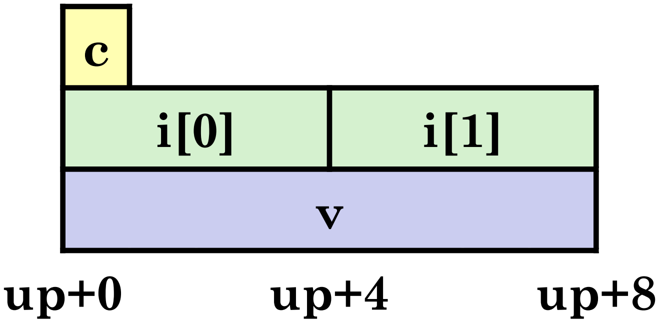


What are Unions?

- A place in memory used to store data types
- Unlike structs, union elements are not placed "next to each other in memory"
  - Rather they are placed "on top"
- Size is decided by the largest element
- Only one field used at a time
  - Each write overwrites some part of another
- This class does not deal with unions very much
  - If think you want unions… You probably want structs

Union Example (from Lecture)

```
union U1 {
  char c;
  int i[2];
  double v;
} *up;
```



Structs on Exams

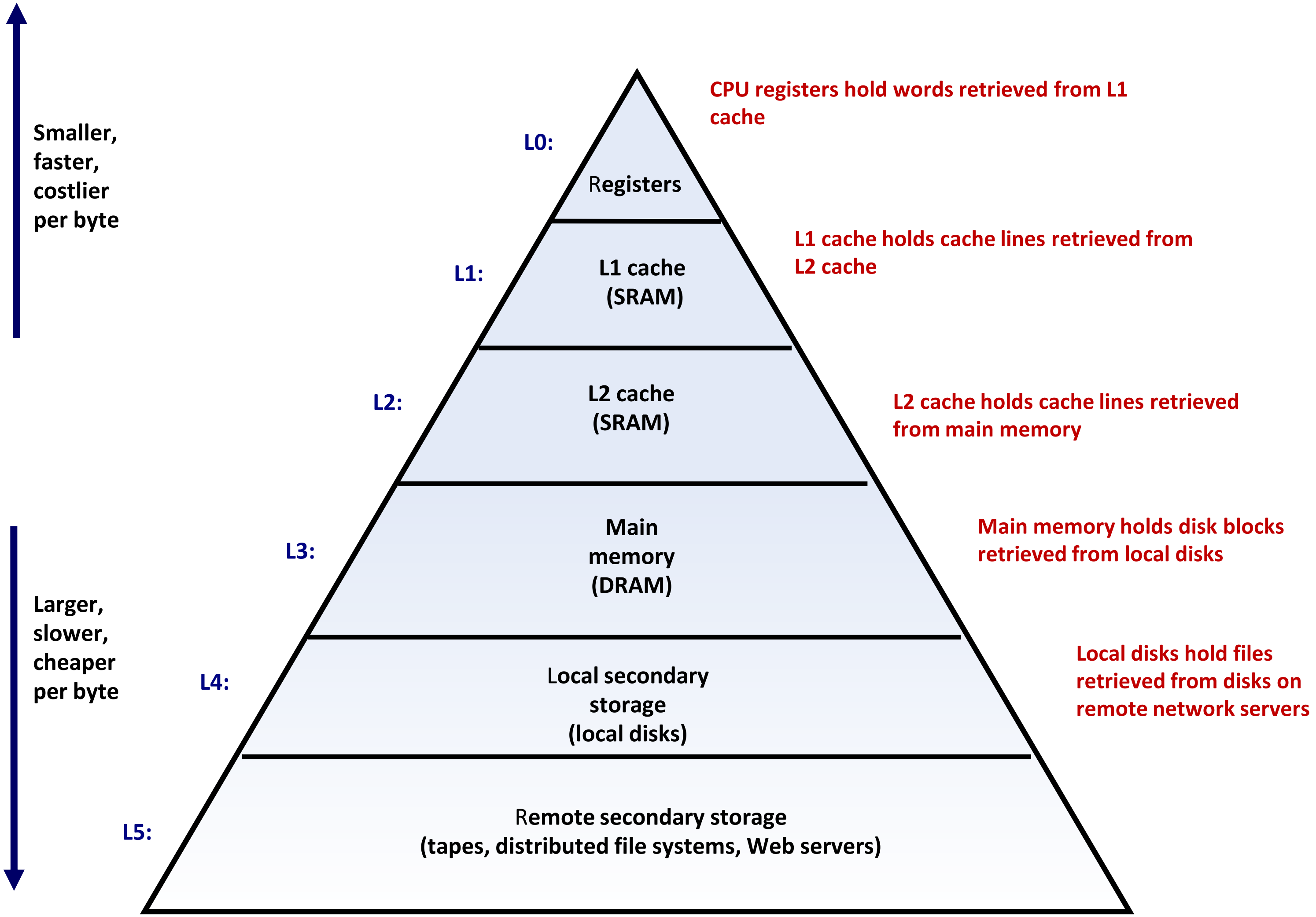
```
struct stats {
  int num_views;
  short sum;
};
```

Goal: Align `struct system_f` according to a 64-bit Linux system

```
struct system_f {
  char a;
  int* b;
  int c[3];
  long d;
  struct stats e;
  short f;
};
```

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
a	X	X	X	X	X	X	X	b	b	b	b	b	b	b	b
c	c	c	c	c	c	c	c	c	c	c	c	X	X	X	X
d	d	d	d	d	d	d	d	e	e	e	e	e	e	X	X
f	f	X	X	X	X	X	X								

Memory Hierarchy (From Lecture)

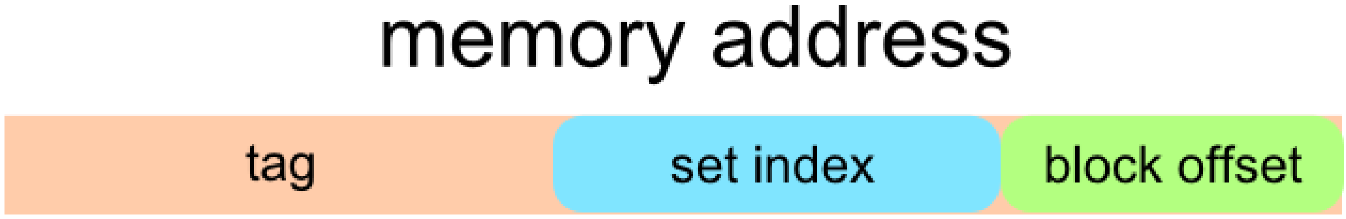


SRAM vs DRAM

- SRAM
  - Faster (L1 Cache: 1 CPU cycle)
  - Smaller (L1 in kilobytes; L2 in megabytes)
  - More expensive and “energy-hungry”
- DRAM (Main memory)
  - Relatively slower (hundreds of CPU cycles)
  - Larger (Gigabytes)
  - Cheaper

Address Division in Caches

- On the Shark machines, addresses are 64-bits
- Dividing a memory address
  - Block offset: b bits
  - Set index: s bits
  - Tag bits: address size – b – s



Cache Parameters

- A cache is a set of $S = 2^s$ cache sets
- A cache set is a set of E cache lines
  - E is called associativity
  - If E = 1, the cache is "direct-mapped"
- Each cache line stores a block
  - Each block has $B = 2^b$ bytes
- Total capacity $C = S * B * E$

Cache Lookup Steps

- Divide address into parts
  - Block offset: Low b bits
  - Set number: Next s bits
  - Tag: Remaining ((address size) – b – s) bits
- Check each line in a set, compare tags
  - If one matches and it's valid, it's a hit!
  - If none match, it's a miss. Add block to cache
    - If there's no room, evict a line from the set

Cache Lab Part A

- Cache Simulator
  - Implement for variable s, b, and E values
    - Values read in from a trace file (at runtime)
  - Least Recently Used (LRU) Policy
- Cache Simulator != Cache
  - This simulator does NOT store memory contents
    - Only performs lookups/ evictions for various addresses
  - We do NOT care about block offsets here
  - Goal: count the number of hits, misses, and evictions
    - Read addresses from files and return these numbers

Pulling out Cache Parameters

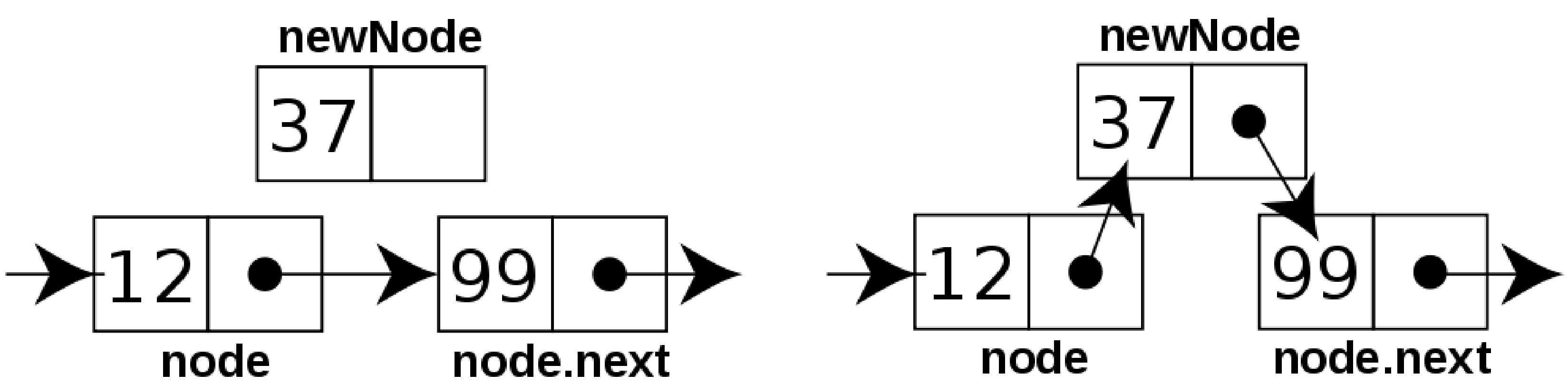
- Remember how to use masks from Data Lab?
- Example address: AABBCCDD
  - 8 block offset bits (b):
    - 0xAABBCCDD & 0xFF = DD
  - 8 set bits (s):
    - (0xAABBCCDD >> b) & 0xFF = CC
  - Remaining tag bits:
    - (0xAABBCCDD >> (b + s)) & 0xFFFF = AABB

General Simulator Design 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"

ANITA'S FAVORITE DATA STRUCTURE

- Linked lists
  - "The only data structure you will ever need"
  - (Heavily) used in cache and malloc lab
  - A lesson on linked list in the credits page



Food for Thought/ Other Designs

- How necessary is the LRU counter?
  - We have the power to insert nodes wherever we want
    - So why use a counter?
- As a C programmer, implementing a linked list should be second nature
  - The same deal every time
    - Pointers to each node
    - Traversal helper functions
    - Checking invariants

Function 1: getopt

- `getopt` automates parsing elements on the unix command line
  - Typically called in a loop to retrieve arguments
  - Use a switch statement to handle options
  - Returns -1 when there are no more arguments
- Must include the 2 header files:
  - `unistd.h`
  - `getopt.h`

FUNCTION 1: GETOPT USAGE

- Switch statement used on the (local) variable holding the return value from getopt
  - getopt does not check number of arguments
  - Each command line input can be handled separately
    - Colon in specifier means required argument
  - optarg – Points to the value of the option argument
    - This is set by the getopt function
- Food for thought
  - How do we handle invalid inputs?

Function 1: getopt Example

- Suppose we had an executable called “foo”
  - Example call from shell: `unix> ./foo -x 1`

- Next slide: Parsing the argument to the x option
  - Notice: We passed in an int which is read as a char *
  - We use `atoi` to convert the string to an int

Function 1: getopt Example Cont.

```
int main(int argc, char** argv){
    int opt, x;

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

Function 2: fscanf

- The `fscanf` function is just like `scanf/sscanf`
  - But it can specify a stream to read from
  - `scanf` always reads from stdin
  - `sscanf` reads from a string
- Parameters:
  - File pointer
  - Format string with information on how to read file
  - Variable number of pointers to with locations for storing data from file
- Typically use in a loop until it hits the end of file
- `fscanf` is useful in reading from the trace files

Function 2: fscanf Example

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

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

int x, y;
char c;

/* read two ints and a char from file */
while(fscanf(pFile, “%d %d %c”, &x, &y, &c) > 0){
    // Do stuff
}

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

Variation Using fgets/sscanf

```
FILE *pFile = fopen ("myfile.txt", “r");

if(!pFile){   /* NULL check */
    printf("%s: %s\n", pFile, strerror(errno));
    exit(1);
}

int x, y;
char buf[1000];

while(fgets(buf, 1000, pFile) != NULL) {
    sscanf(buf, "%x %x", &x, &y);
    // Do stuff
}

fclose(pFile);
```

Style and Tips for Life

- Check for failures and errors ALWAYS
  - Functions don’t always succeed
  - What happens when a system call fails?
- Common cases of failure:
  - Not checking the return of `malloc`
  - Not handling invalid inputs
  - Generally, not checking returns of functions

I Stole From These Places

- Upside down CPU Cache Pyramid
- Wikipedia: Linked Lists
- C Linked List Example
- getopt from GNU
- fscanf from CPlusPlus.com