CS 354 - Machine Organization & Programming
Thursday, October 12, 2017

Project p2 (6%): DUE at 10 pm on Sunday, October 15th

Homework hw3 (1.5%): DUE at 10 pm TOMORROW, Friday, October 13th

Project p3 (6%): Assigned Tomorrow

Last Time

Placement Policy
Free Block - Too Much/Too Little
Coalescing Free Blocks
Footers
Explicit Free List

Today

Explicit Free List (from last time)
Ordering Free Blocks (from last time)
Heap Caveats
Locality
Memory Hierarchy

Next Time

Designing Cache Memories
Read: B&O 6.4 intro - 6.4.2

Heap Caveats

Don't assume consecutive heap allocation result in continuous payloads!

→ Why? Payloads are interspersed with heap headers and padding.

Don't access uninitialized heap memory!

→ Isn't heap memory initialized to 0?
No! Unless use calloc.
Note the OS does initially clear heap memory for security but not when you reuse blocks.

Do free all heap memory that your program allocates!

→ Why are memory leaks bad?
Slowly kills your programs performance by cluttering the heap with wasted blocks.

→ Do memory leaks persist when a program ends?
No. Memory is returned to OS to reuse.

Don't free heap memory more than once!

→ What is the best way to avoid this mistake?

Null out free pointers.

Don't access data in freed heap blocks!

→ What kind of error will result?

Intermittent error.

Don't change heap memory outside of your payload!

→ Why? There is internal structure that you could mess up.

Do check if your memory intensive program has run out of heap memory!

→ How? Check allocator retur value.
If null, it means allocation failed.

Three Faces of Memory

Goal Enable multiple process's to run on a single machine.
By creating the illusion that each process is running on its own machine.

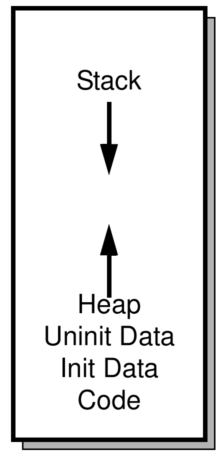
Process/Virtual View

How the memory looks to a process.

Problem: Make coding easier by having a simple & uniform view of memory.

Solution? View is of

A Virtual address space with uniform memory segments where a process generates virtual addresses to access the virtual memory.



Hardware/Physical View

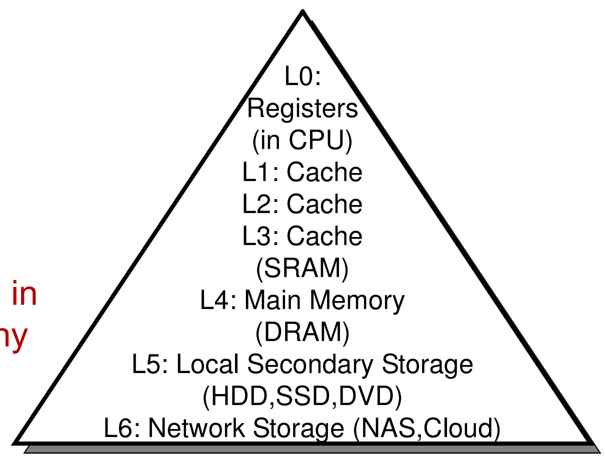
How memory looks in the actual machine.

Problem: Keep the CPU busy

Speed/Efficiency

Solution? View is of

View of a physical address space of installed memory where physical addresses locate data in physical memory which is a multi-level hierarchy that ensures frequently used data is closer to CPU.



System View (CS 537)

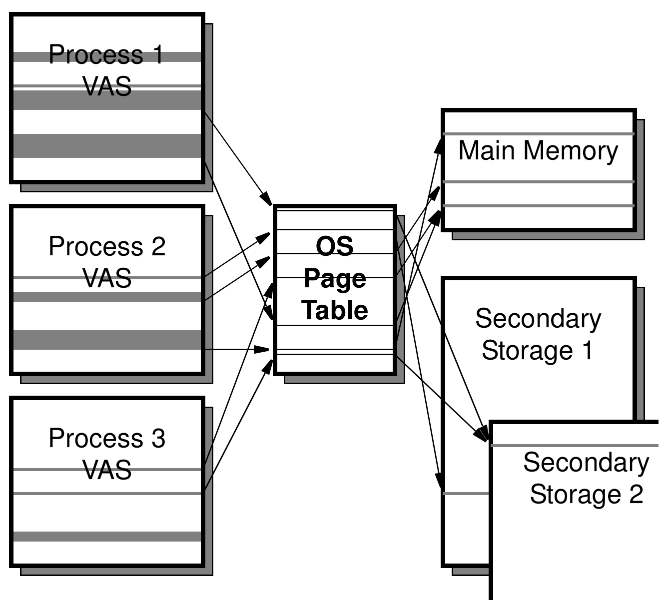
How memory looks to the OS.

Problem: Make memory sharable, efficient, secure.

Solution? View is of

View of memory divided into chunks called pages. And the OS keeps a page table to map virtual pages to physical pages.

The page table along with hardware called the MMU that maps virtual addresses to physical addresses.



Locality

✵ Programs with good locality run faster than those with poor locality.

Why?

Keeps data at the top of the memory hierarchy where it can be quickly accessed by the CPU.

What?

temporal locality

When a recently accessed memory location is accessed repeatedly in a near future.

spatial locality

When a recently accessed memory location is followed by access of nearby memory locations in the near future.

Example

```
int sumArray(int a[], int size, int stride) {
   int sum = 0;
   for (int i = 0; i < size; i += stride)
      sum += a[i];
   return sum;
}
```

→ List the variables that demonstrate temporal locality.

i, size, sum, stride

→ List the variables that demonstrate spatial locality.

a if stride is small

stride distance (step sizes) words in the address space between sequencial accesses

Locality = 1/stride

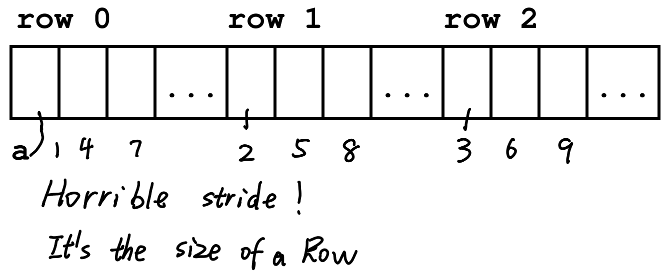
Bad Locality

Why is this code bad?

Assume 3

```
int a[ROWS][COLS];
for (int c = 0; c < COLS; c++)
   for (int r = 0; r < ROWS; r++)
      a[r][c] = r * c;
```

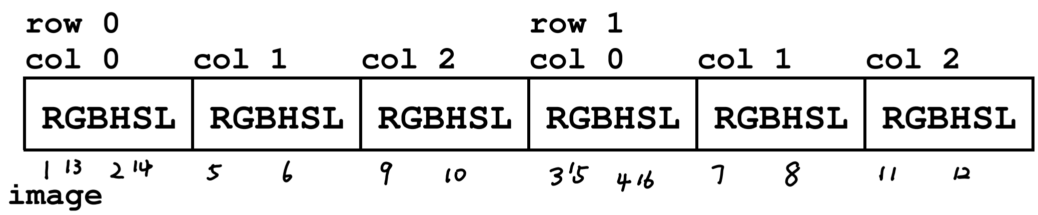
Flip to Fix



Key Questions:

1. What does the memory layout look like ?
2. What is the stride of the code ?

Why is this code bad?




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

Assume 2 rows 3 cols

```
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;
      }
```

Good or bad locality?

- Instruction Flow:

  Sequencing: Good Spatial Locality
  Selection: Bad spatial locality
  Repitition: Good temporal locality

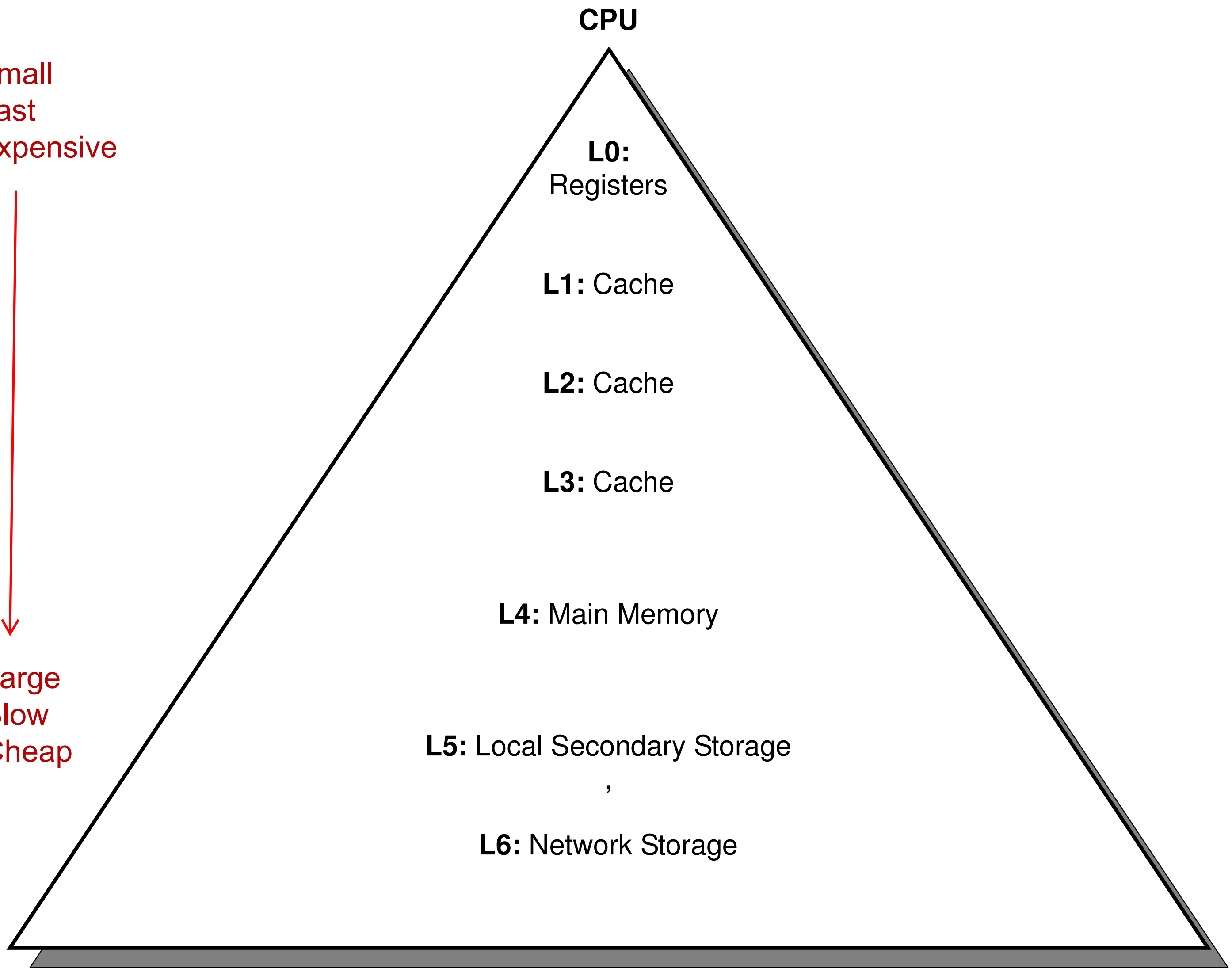
- Searching Algorithms:

  Linear Search: Good Spatial Locality
  Binrary Search: Bad Spatial Locality

Memory Hierarchy

Small
Fast
Expensive

Large
Slow
Cheap



cache block Unit of memory transfered/managed by main memory & caches
It implement spatial locality,

latency Delay due to transit in memory hierarchy