Page No.__

Outline

- Lab Learnings
  - Box of bits (scalars) → Bunch of boxes
    - indexed : Arrays
    - group : struct
    - indired : pointers

    } each combines with the others
- Where data lives
- Stacks, heaps, buffers & Attacks
- More about bits

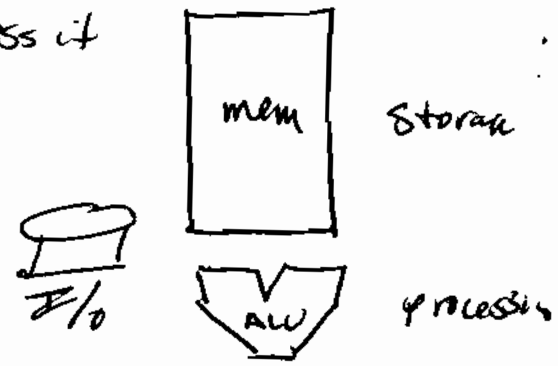
Perspective

VHLL
Java
Ruby

MLL
C

M/C
- Finite
  - word width
    ⇒ Range
    precision
- Physical

- Scalars
  - Size ≤ word width
  - meaning ⇐ what process it
    - Arithmetic
    - Comparison / Test
    - * addresses
    - characters
      display / keys



others?
- pixels
- audio ?
- geometric shapes

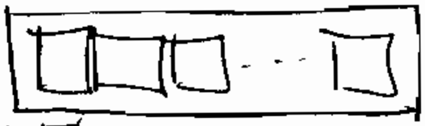
Witnessed & Understood by me, | Date | Invented by | Date
Recorded by

From Page No

Lab learnings

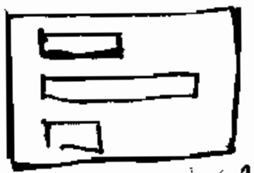
Bunch of Boxes

contiguous homogeneous
Array : sequence of like elements, indexed



- ~~count~~ START, next/prev, count, i^{th}
  contents

Struct : collection of heterogeneous elements



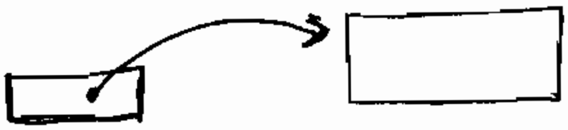
ops on the struct
- move it, size
- select fields

no compare

⇒ Array of structs
⇒ struct containing array

- named fields, size,

Pointer



- reference to a noncontiguous element
- pointer, object it points to
  reference, dereference
- NULL

⇒ ptr to array
array of ptrs
ptr to struct
~~ptr~~ ~~to~~ ptr

Roles
- Uniform handle on non-uniform object
- Non contiguous objects
- sharing
- mutation
- delayed allocation

To Page No. 2

m Page No.

Where data lives (and for how long)

langage level

file.c

```
int a; (1)
extern e;
static int d;
Function foo (4) (int c (3), ) {
  int b (2)
   .. a+b
}
main (    ) {
   foo (17, )
}
```

"external" variable
- defined here, visible in rest of file and outside.
- declared here but defined elsewhere

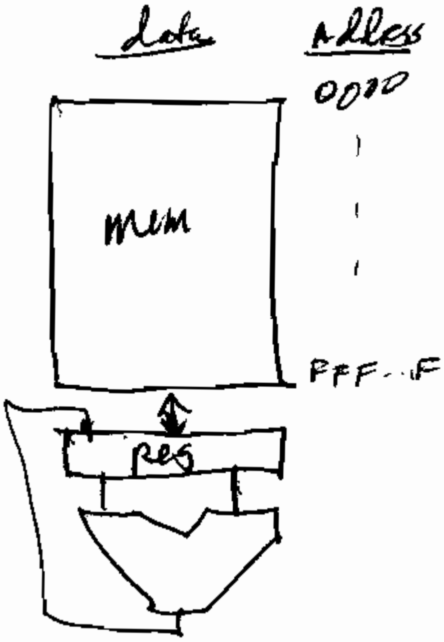
"automatic" variable
- local to function

local copy of call parameter value

external too

Machine level

- logically all data resides in memory
- loaded into proc reg's operate store back to mem

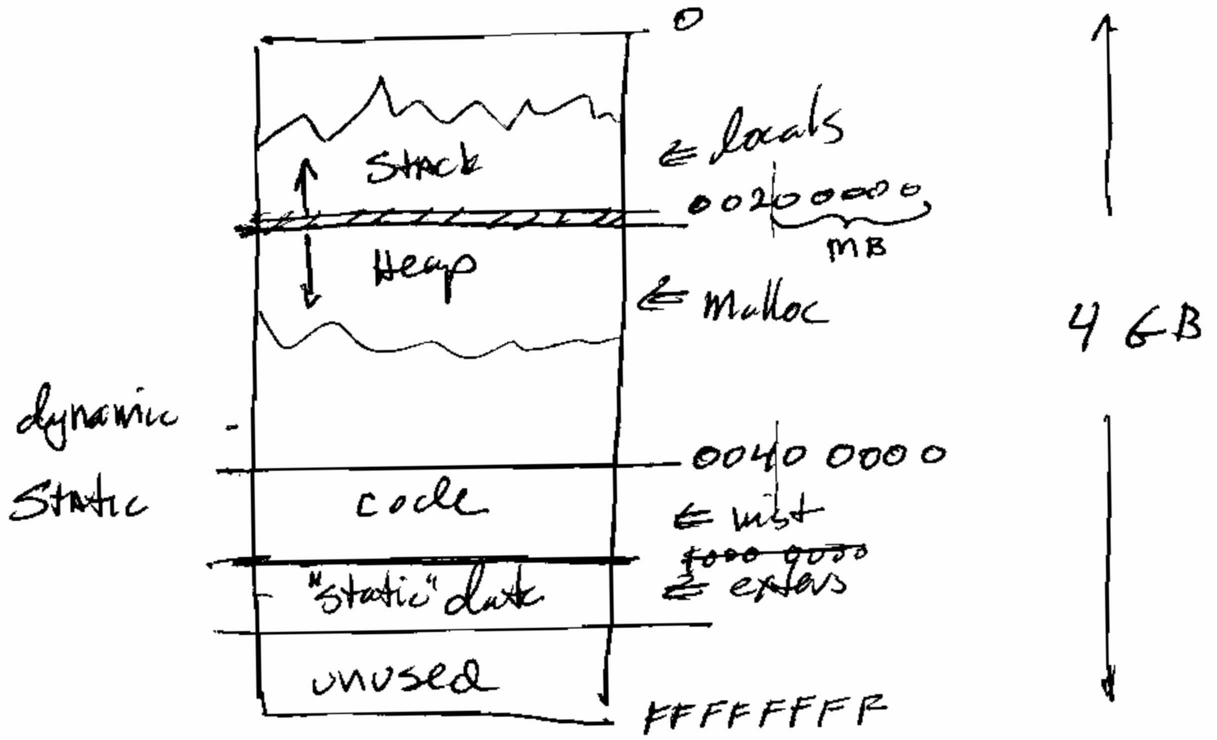


How do langage concepts map to machine?

Witnessed & Understood by me, | Date | Invented by | Date
Recorded by

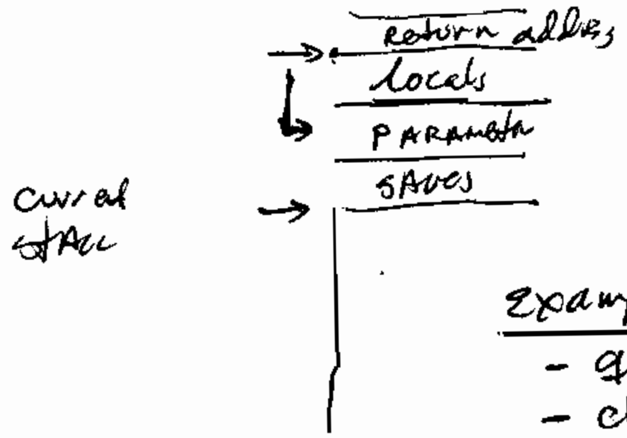
From Page No.__

$2^{10} \sim 10^3$
10 bits k
20 bits M
30 bits G



machine doesn't know
Language run time sets it up

What happens when a function is called?
- saves
- parameters
- locals



Where
- scalars
- arrays
- structs
- ptrs

Examples from lab
- queue
- char buffer

To Page No.__

From Page No.

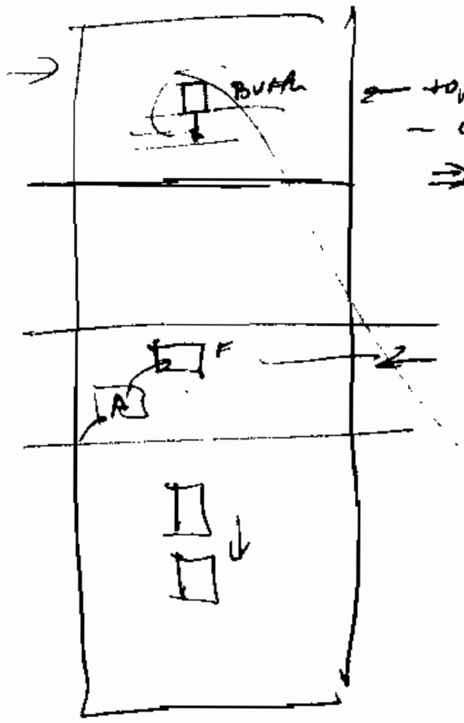
Attacks

or service
or application

Cause to OS (a program) to do things it shouldn't

- ~~write~~
- clobber data
  - pass strings with no termination
  - pass bad length parameters
  - pass bad ptrs

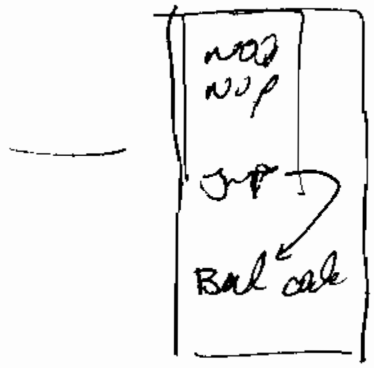
- corrupting system data may cause it to do ~~other~~ stupid things



top of st
- clobber ~~other~~ variable, clobber return addr

⇒ cause OS to start executing out of the stack
  - insts are bits, bits can be inst

execute



- safe string libraries
- segments (protection on portions of address space)
- stack checking

To Page No. 5

Bit more on bits

- what happens when you assign a small scalar to a larger one

Signed
long int i = short int j;

unsigned
int i = char a;

zero extend (unsigned) sign extend (signed)

Prove this is valid

intuitive 2008 ≡ 02008 ≡ 00002008

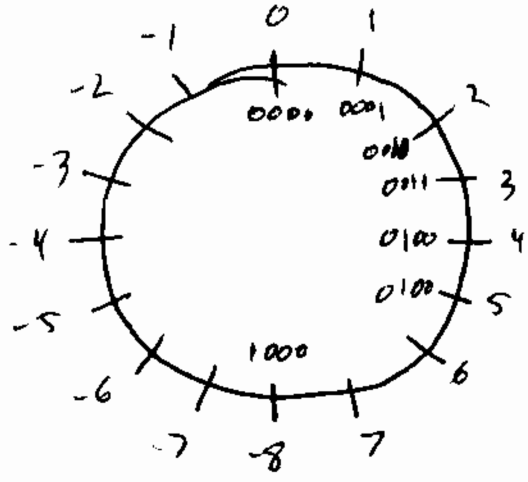
Formally

$$b_{n-1} \cdot 2^{n-1} + \cdots + b_1 2 + b_0 = 0 \cdot 2^{m-1} + \cdots + 0 \cdot 2^{n} + b_{n-1} 2^{n-1} + \cdots + b^0$$

Can you prove it works for signed numbers?

From Page No.

Recall



$$-b_{n-1}\cdot 2^{n-1} + b_{n-2}\cdot 2^{n-2} + b_{n-3}\cdot 2^{n-3} + \cdots + b_0$$

Case $b_{n-1} = 0$ (non negative)

$$-b_{m-1}\cdot 2^{m-1} + \cdots + b_n 2^{n} + b_{n-1} 2^{n-1} + b_{n-2} 2^{n-2} + \cdots + b_1 2^1 + b_0$$

se $b_{n-1}=0$ 0

se $b_{n-1}=1$

$$-2^{m-1} + 2^{m-2} + \cdots + 2^{n-1} = \left(-2^{m-1-n+1} + \quad + 2^0\right)\cdot 2^{n-1}$$

$$\underbrace{= -2^{m-n} + \underbrace{2^{m-n-1} + \cdots + 2^0}_{2^{m-n}-1}}_{-1}$$

To Page No. 7