9/20/10

Midterm #1
for Mon/Wed class
This Wed - 10-11:20
OHE 122
(regular classroom)

Project 2 - 3 Parts

Parts 1&2 - System Calls & Multiprogramming

Part 3 - Convert Carl's Jr to a Nachos user program

Memory Management

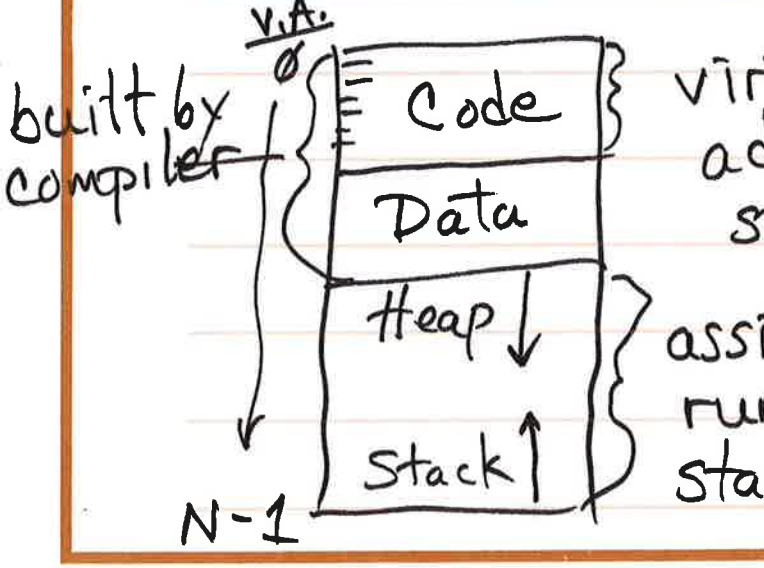
Nachos is uniprogrammed

- It can only run one single-threaded user program at a time

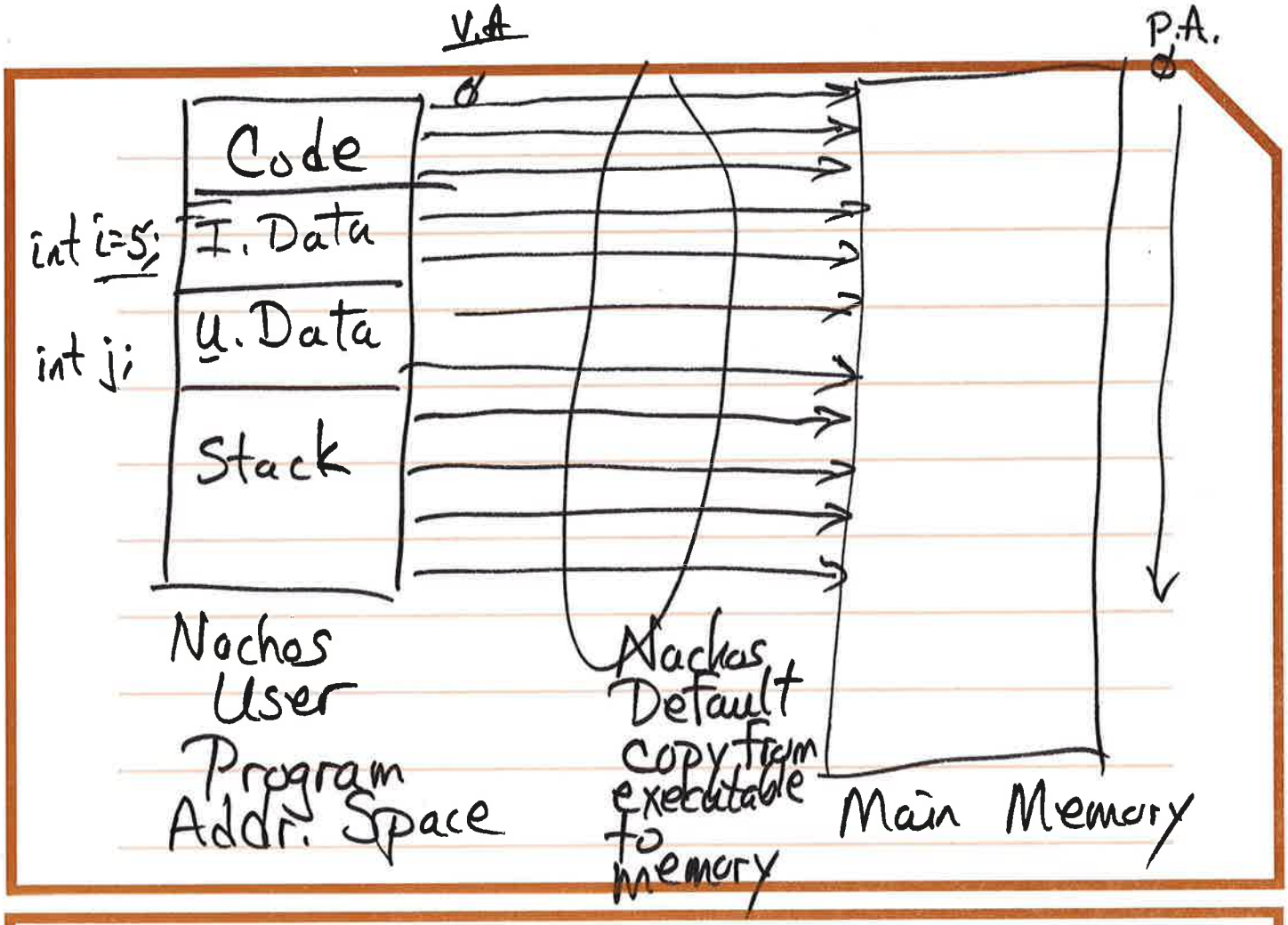
When Nachos loads a user program- it loads the program so that the "virtual addresses" are the same as the "physical addresses" (what & where resides in physical memory)

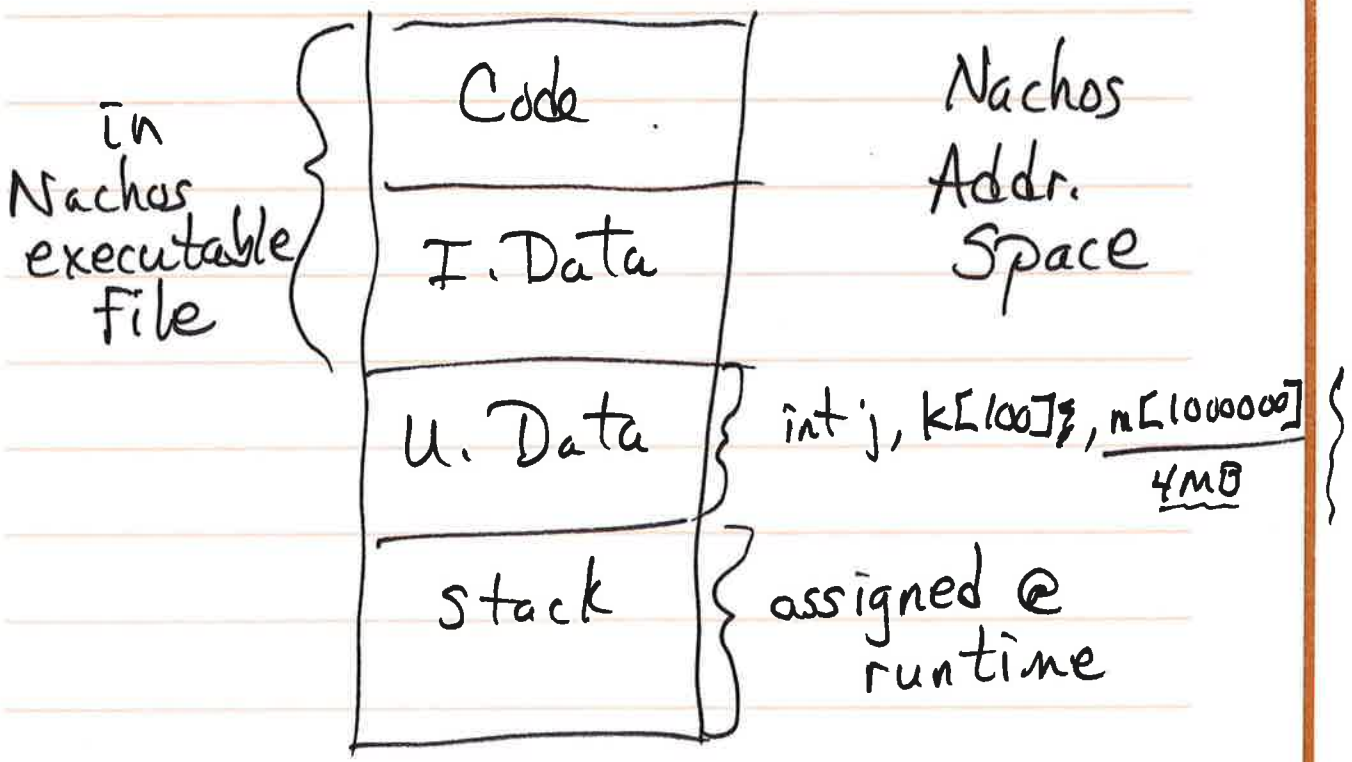
```
int i=0;
* i=5;
for(  ){
```

first executable instruction is at virtual address 0

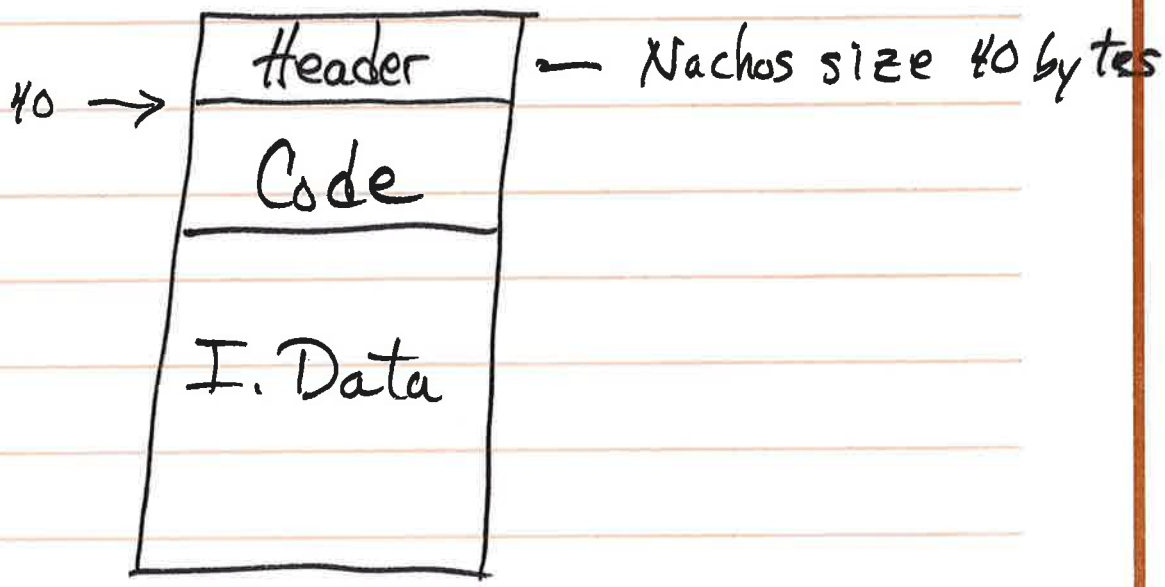


⇒ Part of our security model


V.A.
P.A.
0
Code
int i=5;
I. Data
U. Data
int j;
Stack
Nachos User Program Addr. Space
Nachos Default copy from executable to memory
Main Memory


in Nachos executable File
Code
I. Data
U. Data
stack
Nachos Addr. Space
int j, k[100], m[1000000]
4MB
assigned @ runtime

Nachos Executable File



Header contains 3 values for Code, I. Data, U. Data

- starting virtual address
- size of segment
- byte offset of segment start in executable (except for U. Data)

Nachos names

How to load the executable contents into memory?

- Memory is "divided" into equal size pieces - pages
- The OS allocates memory 1 page at a time ⇓ to a user program
- We copy the code & initialized data from the executable into memory 1 page at a time ⇓ an unused page

The O.S. must now track the used / unused pages of memory

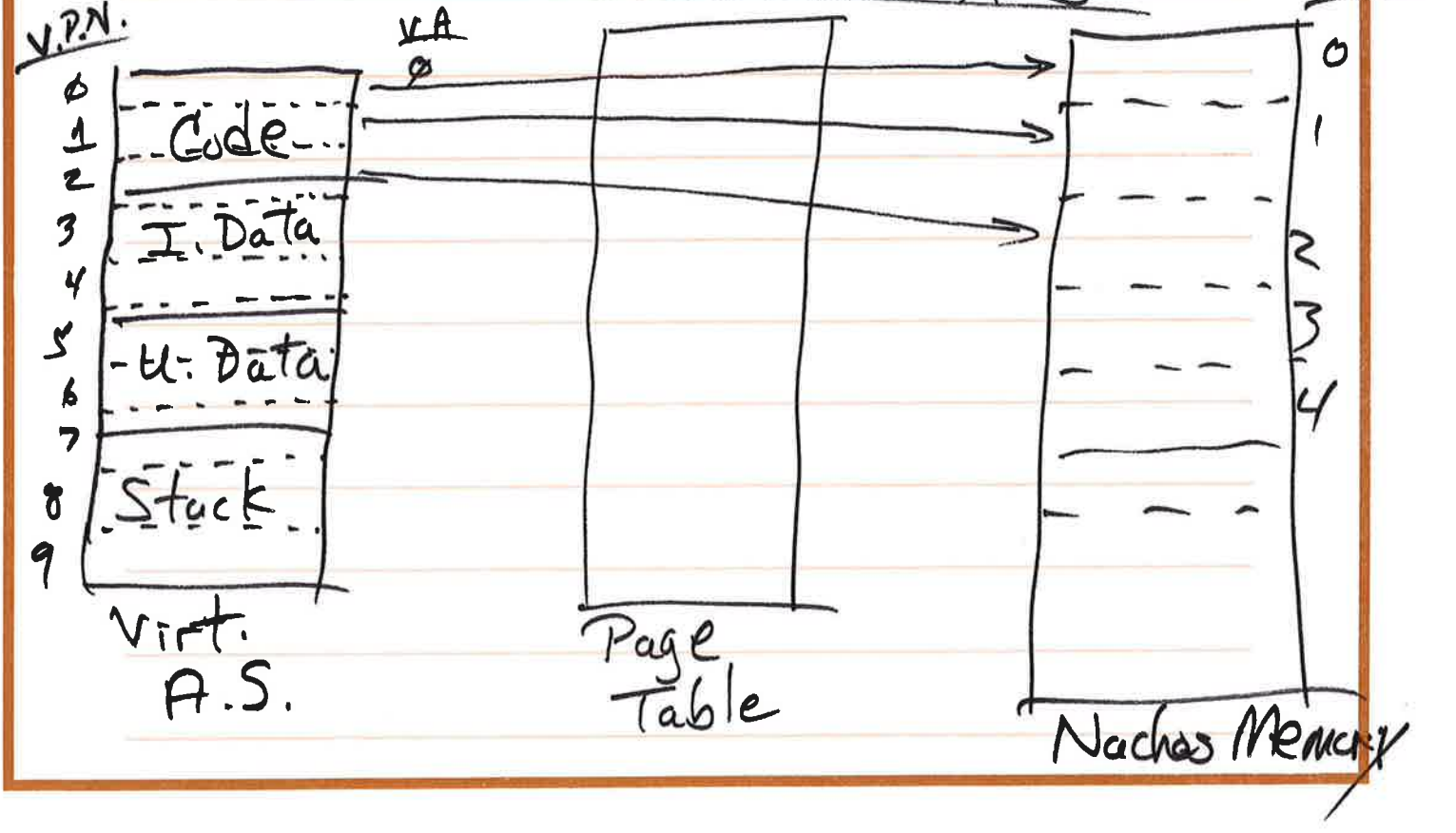
Page Table: For each process, we need to track where each virt. address space page is loaded into memory

Page Table

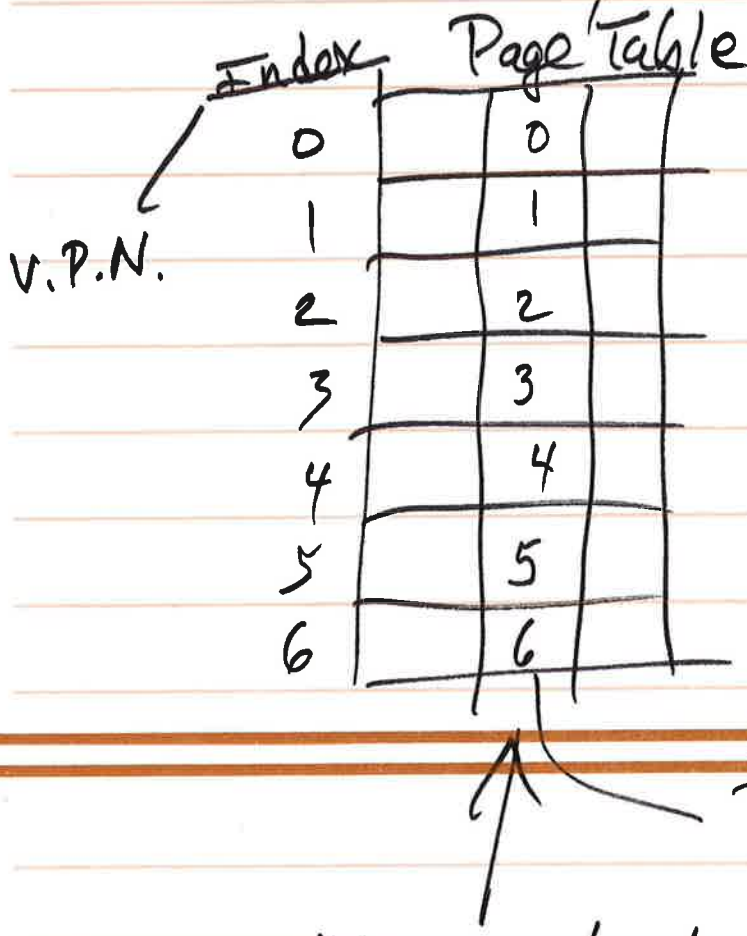
A mapping of virtual page locations (starting virtual address of that page) to the physical page locations (where were they copied into memory).

Nachos has a page table - as part of a process

Default Nachos Mapping



Page Table is an array
- indexed by V.P.N.



Nachos Default Loading

You must change this for multiprogramming

You must find an unused page of memory when loading a new virtual page

The Nachos page table is part of the AddrSpace class

- in userprog directory
- addrspace.h/.cc

Address space loading must be completed BEFORE the user program can execute

- in ~~code~~ AddrSpace constructor

Your task: When a new page of memory is needed by a process, select an unused page of memory

Use BitMap class - already part of Nachos

Has 2 methods

Find(); returns an int

Clear(_)

↳ takes an int

You create ONE BitMap object for the entire OS ⇒ system.h/.cc

~~The~~ Your main memory is to have the same # of entries as there physical pages of memory

- in machine.h

#define NumPhysPages

is the # of physical memory pages

use NumPhysPages in your BitMap declaration

To "find" an unused memory page call the bitmap Find() method.

- but... it needs a lock

Use Clear to release a page of memory back to the O.S.

Watch Out in AddrSpace Constructor

Nachos does NOT copy from the executable into memory by page. It does the entire segment

- YOU MUST copy only 1 page at a time

executable->ReadAt(location in memory, PageSize,

&mainMemory[PPN * PageSize],

128

start location in File to read

noffH.code.inFileAddr + (VPN * PageSize)

header object

Also, Nachos only allows for 1 thread per process

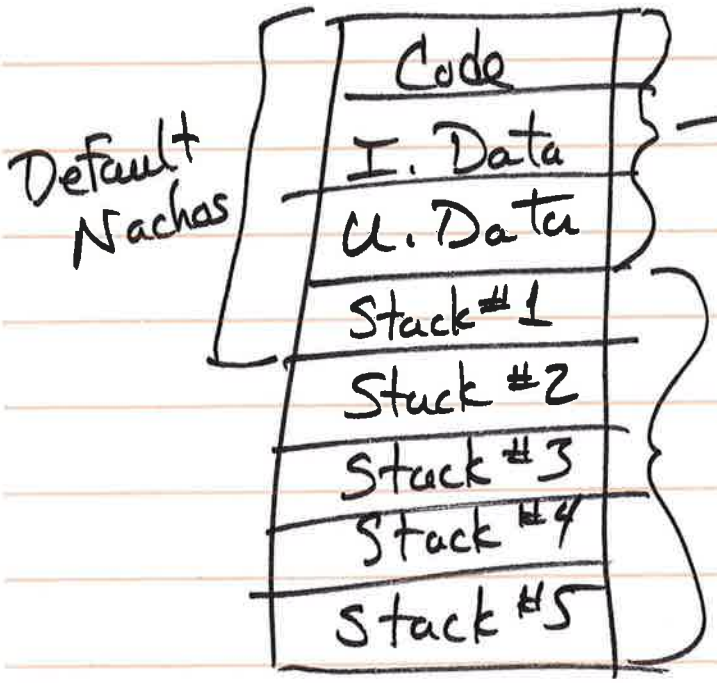
- Each thread MUST have its own stack.

You have to fix this, also.

You MUST allow a process to have stack space for each thread.

- A thread's stack is used & managed by Nachos.
- Your job is to ensure that each thread has its own 8 pages of stack space
- StackReg is the O.S. register for the top of a thread's user stack space

New Nachos Address Space



2 Fundamental Ways to do this

(1) Make the page table big enough for some fixed number of threads

Ex: 50 threads max

$50 * 8 = 400$ pages needed for stacks

Then, on a request to Fork a new thread, you "use" 8 unused pages of stack

(2) Make a new page table each time, 8 pages bigger

- Copy old pagetable data to new page table

Changes to Addrspace Constructor

1. Copy From executable to memory
   1 page @ a time

2. Use your main memory BitMap
   to find an unused memory page

3. Rework the page table so it
   can handle multiple stacks

System Calls

Nachos user programs exist in test directory
- Carl's Jr conversion go here

Part 1, some of Part 2, deal with system calls

↓

mechanism by which the OS allows user programs to perform Kernel tasks

Nachos has a system call mechanism that is setup

Several system calls are already implemented

System calls are implemented in exception.cc in userprog.

You will REALLY want to use the student documentation

Don't try to understand ALL system calls before implementing one.

You are going to write a set of Syscalls

- Allow for multiprogramming
  - Fork - creates a new thread
  - Exec - creates a new single-threaded process
  - Exit - when a thread is done executing

- Synchronization
  - Locks: Acquire, Release, CreateLock, DestroyLock
  - CVs: Wait, Signal, Broadcast, CreateCV, DestroyCV

Miscellaneous: Yield (do this first)
- 1 line of code