EE 312
Day 4
The Program Stack

Vallath Nandakumar

Dept of ECE, UT Austin

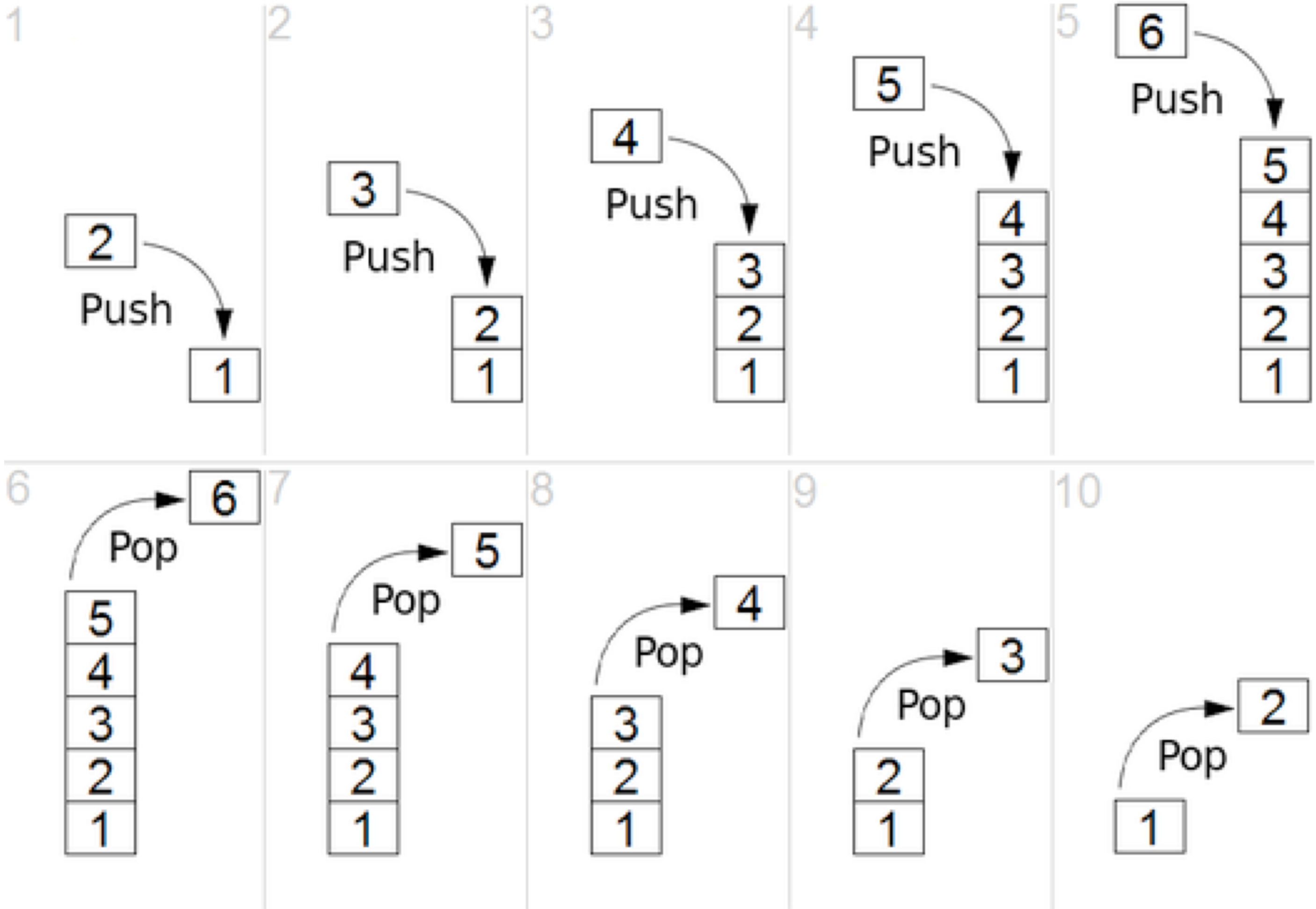
Agenda

- Project 1 questions?
- Review of last lecture
  - Pointers to variables
  - Char arrays
  - Arrays of Strings

Stack vs. Heap

- Stack and heap are different regions of memory
- When a process starts up, the operating system allocates a portion of memory – the stack
  - Usually a fixed (and not very big, a few MB) size
- The heap can occupy a lot of the remaining memory

Data Structure Stack operations



Stack overview

- Special region of your computer's memory that stores temporary variables created by each function (including the main() function).
- Every time a function declares a new variable, it is "pushed" onto the stack. Then every time a function exits, **all** of these variables are freed (deleted).
- Once a stack variable is freed, that region of memory becomes available for other stack variables.

Stack management

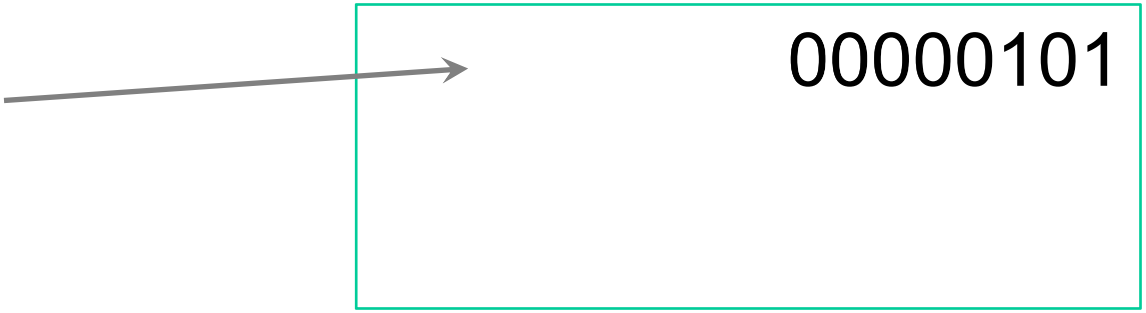
- Memory in stack is managed for user
- Very fast access to stack contents

Stack variables

- Local in nature
- When a function exits, all of its variables are popped off of the stack (and hence lost forever)

Stack growth -- main

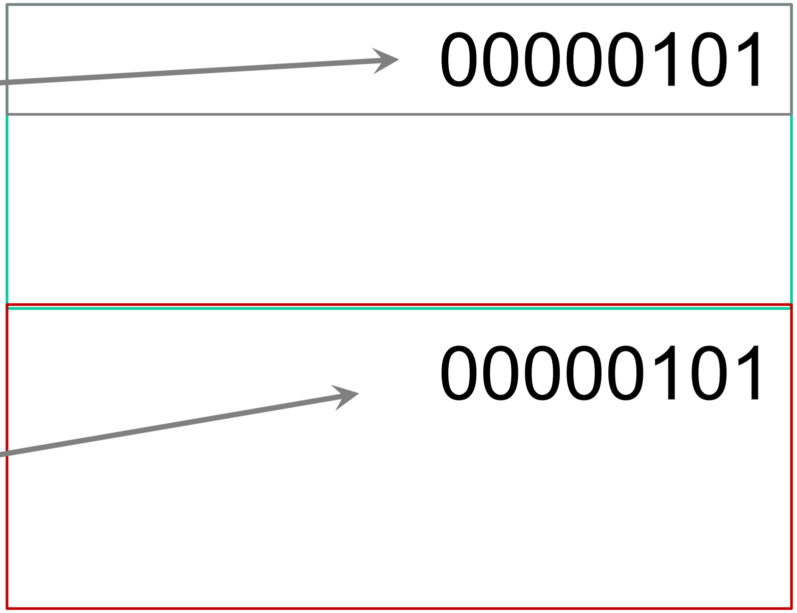
```
int main () {
    int a = 5;
}
```



00000101

Stack growth – foo called

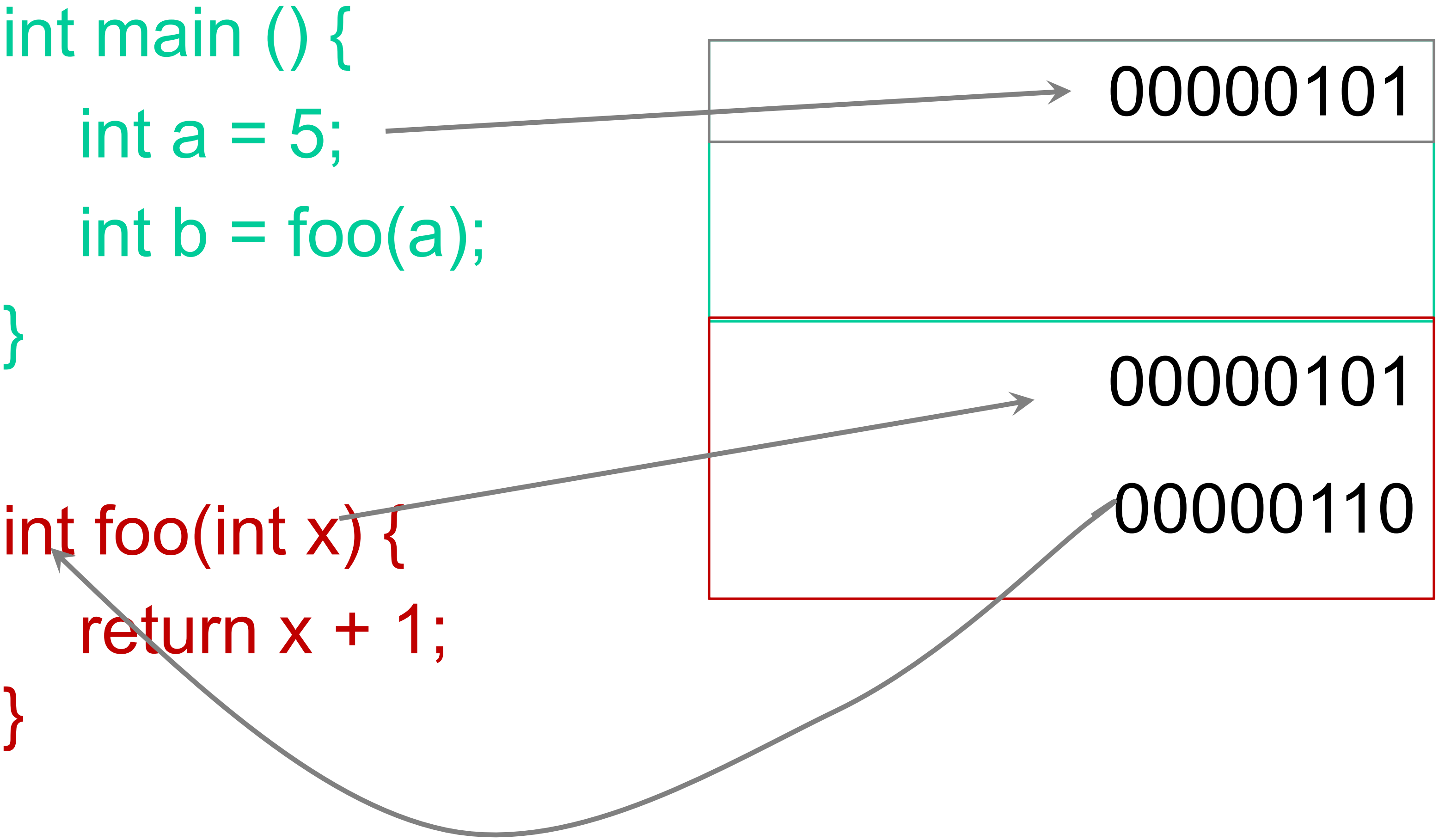
```
int main () {
    int a = 5;
    int b = foo(a);
}

int foo(int x) {
    return x + 1;
}
```



00000101

00000101

Stack growth – foo done



```
int main () {
    int a = 5;
    int b = foo(a);
}

int foo(int x) {
    return x + 1;
}
```

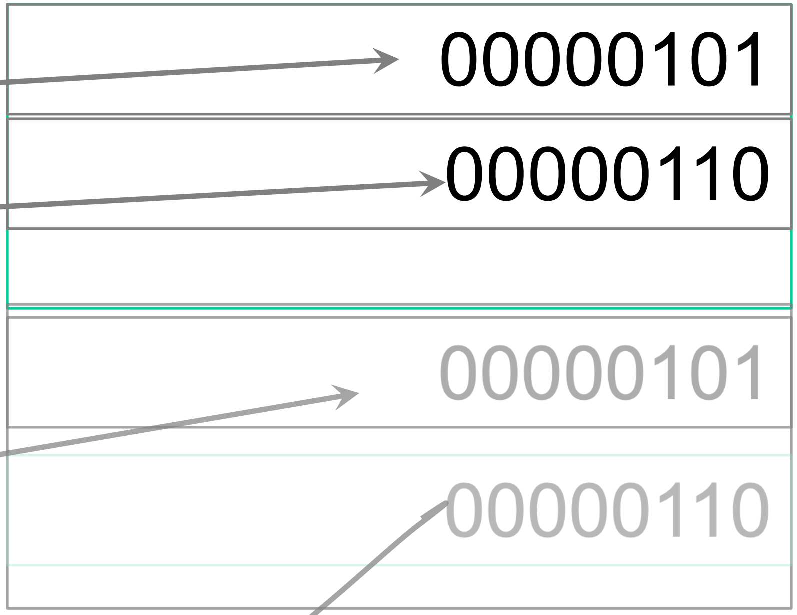
00000101

00000101

00000110

Stack growth – foo returns

```
int main () {
    int a = 5;
    int b = foo(a);
}

int foo(int x) {
    return x + 1;
}
```

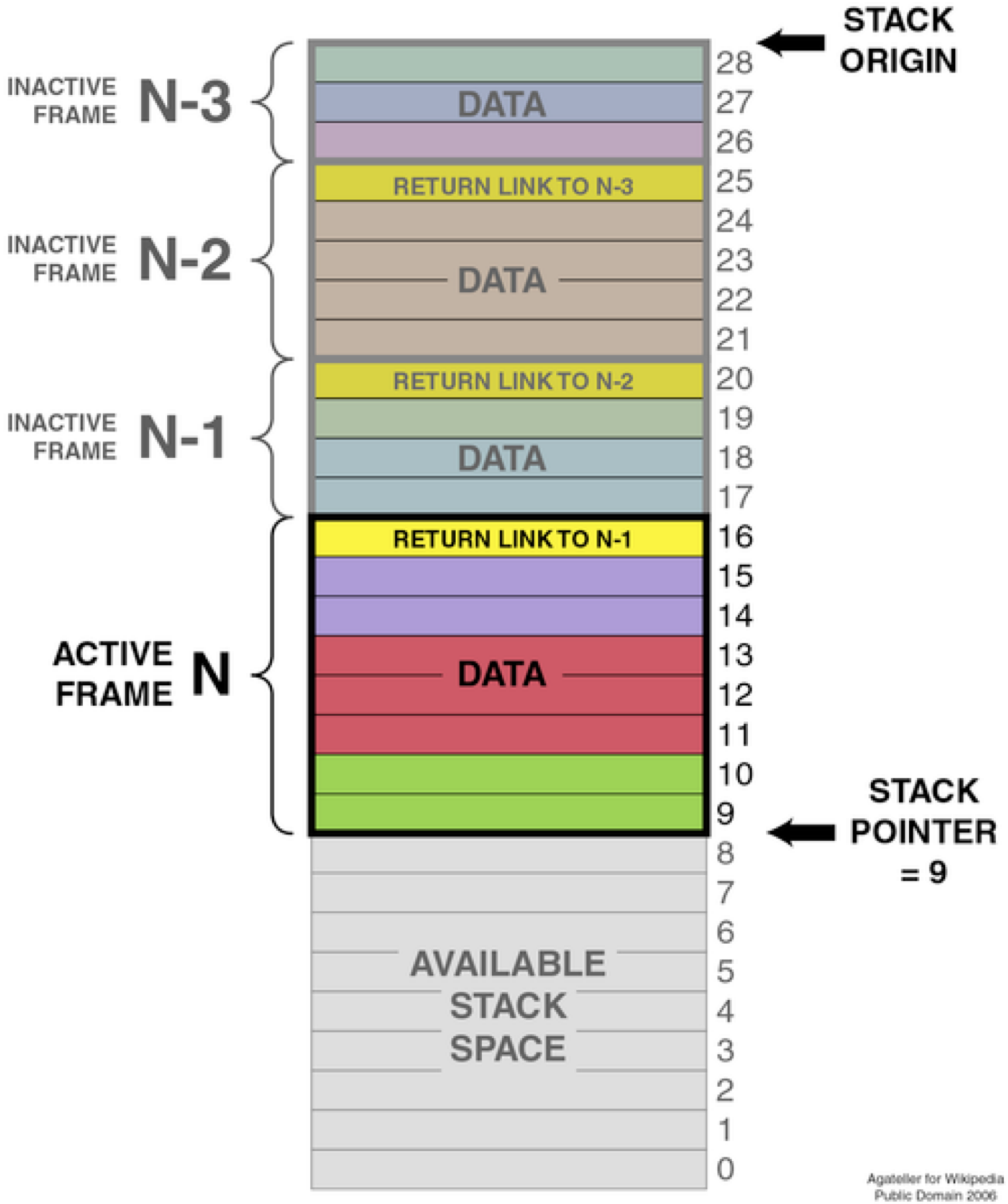


00000101

00000110

00000101

00000110



Agateller for Wikipedia
Public Domain 2006

Stack summary

- The stack grows and shrinks as functions push and pop local variables
- There is no need to manage the memory yourself; variables are allocated and freed automatically
- The stack has size limits – typically, a few MB are allocated to each program.
- Stack variables only exist while the function that created them is running

Heap

- A ‘pile’ of memory
  - Not as tightly managed by CPU
  - User has to allocate and de-allocate memory
- Allocating some memory, and then not deallocating it even after it is not in use any more results in memory leak
- Variables created on the heap are accessible by any function, anywhere in the program. Heap variables are essentially global in scope.
- Variable in size

A process's view of memory

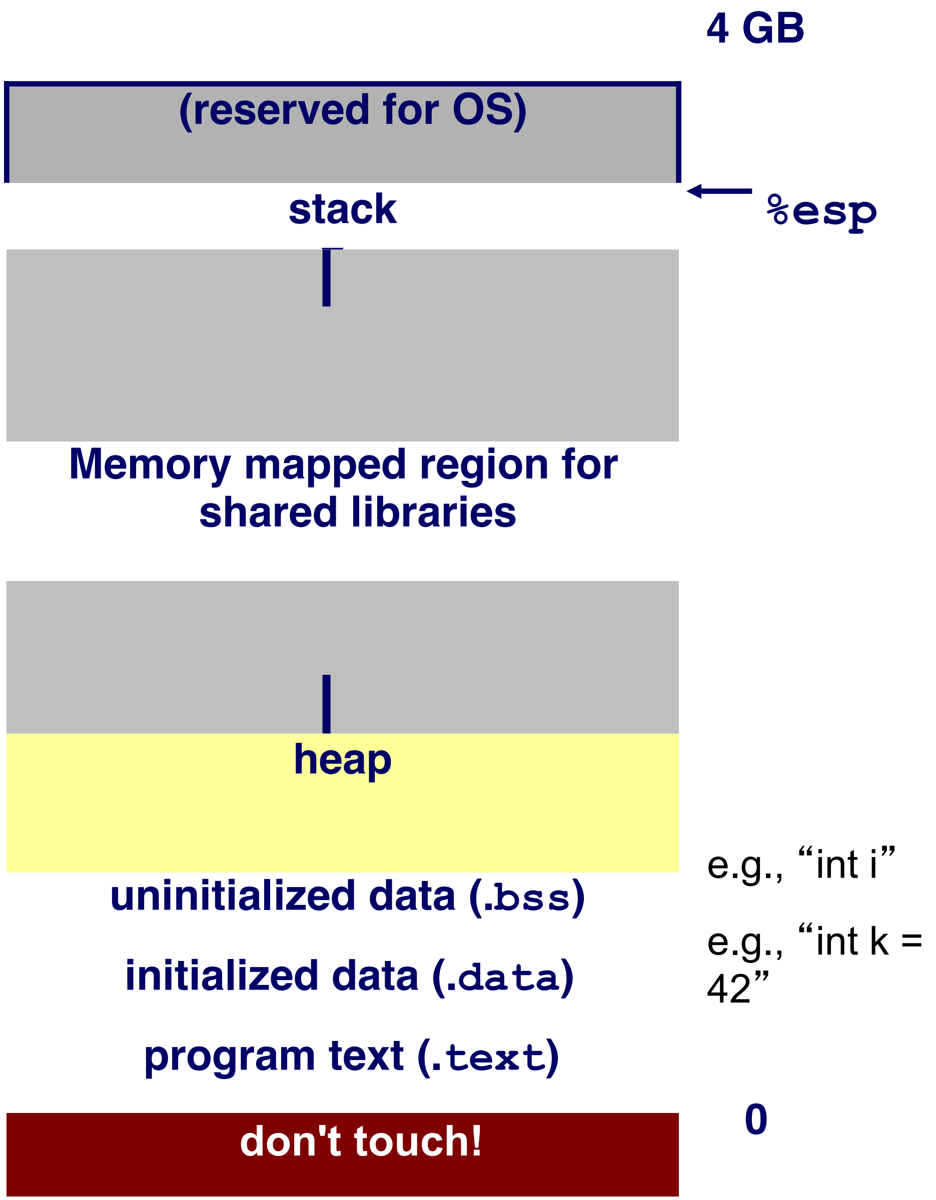
Process not allowed to
read or write this region

Program loader maps
in standard libs here

Dynamically-allocated
memory (via `malloc`)

Global vars Program

code



4 GB

(reserved for OS)

stack

%esp

Memory mapped region for
shared libraries

heap

uninitialized data (.bss)

initialized data (.data)

program text (.text)

e.g., “int i”

e.g., “int k = 42”

0

don't touch!