# SECURITY IN COMPUTING, FIFTH EDITION

## Chapter 3: Programs and Programming

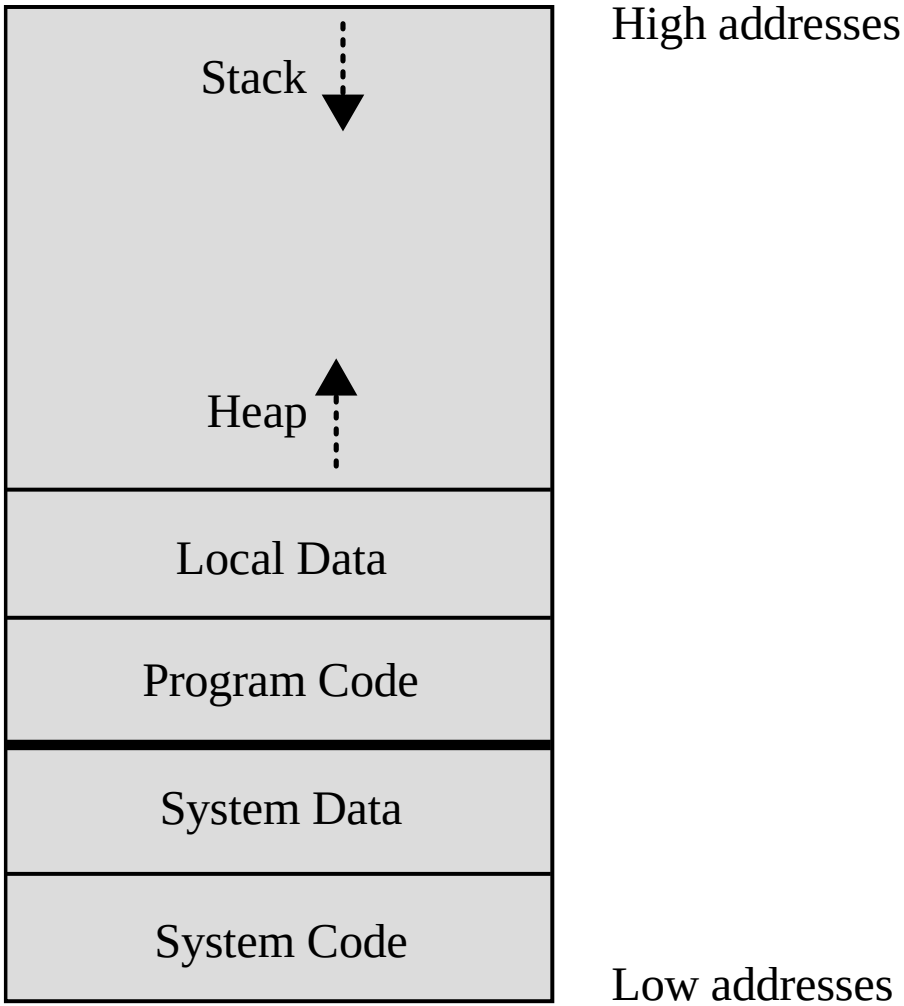
# Buffer Overflows

- Occur when data is written beyond the space allocated for it, such as a 10th byte in a 9-byte array
- In a typical exploitable buffer overflow, an attacker's inputs are expected to go into regions of memory allocated for data, but those inputs are instead allowed to overwrite memory holding executable code
- The trick for an attacker is finding buffer overflow opportunities that lead to overwritten memory being executed, and finding the right code to input

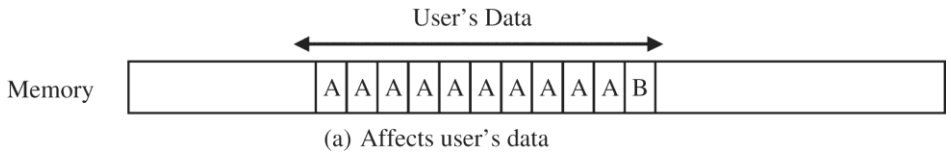
# How Buffer Overflows Happen

```
char sample[10];

int i;

for (i=0; i<=9; i++)
  sample[i] = 'A';

sample[10] = 'B';
```

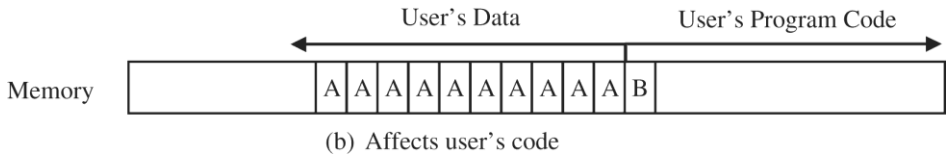
# Memory Organization



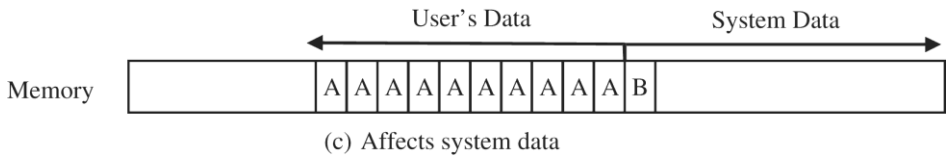
# Where a Buffer Can Overflow



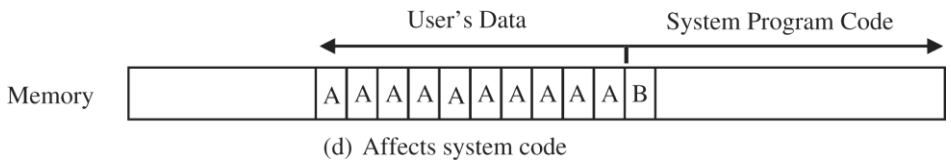

(a) Affects user's data




(b) Affects user's code

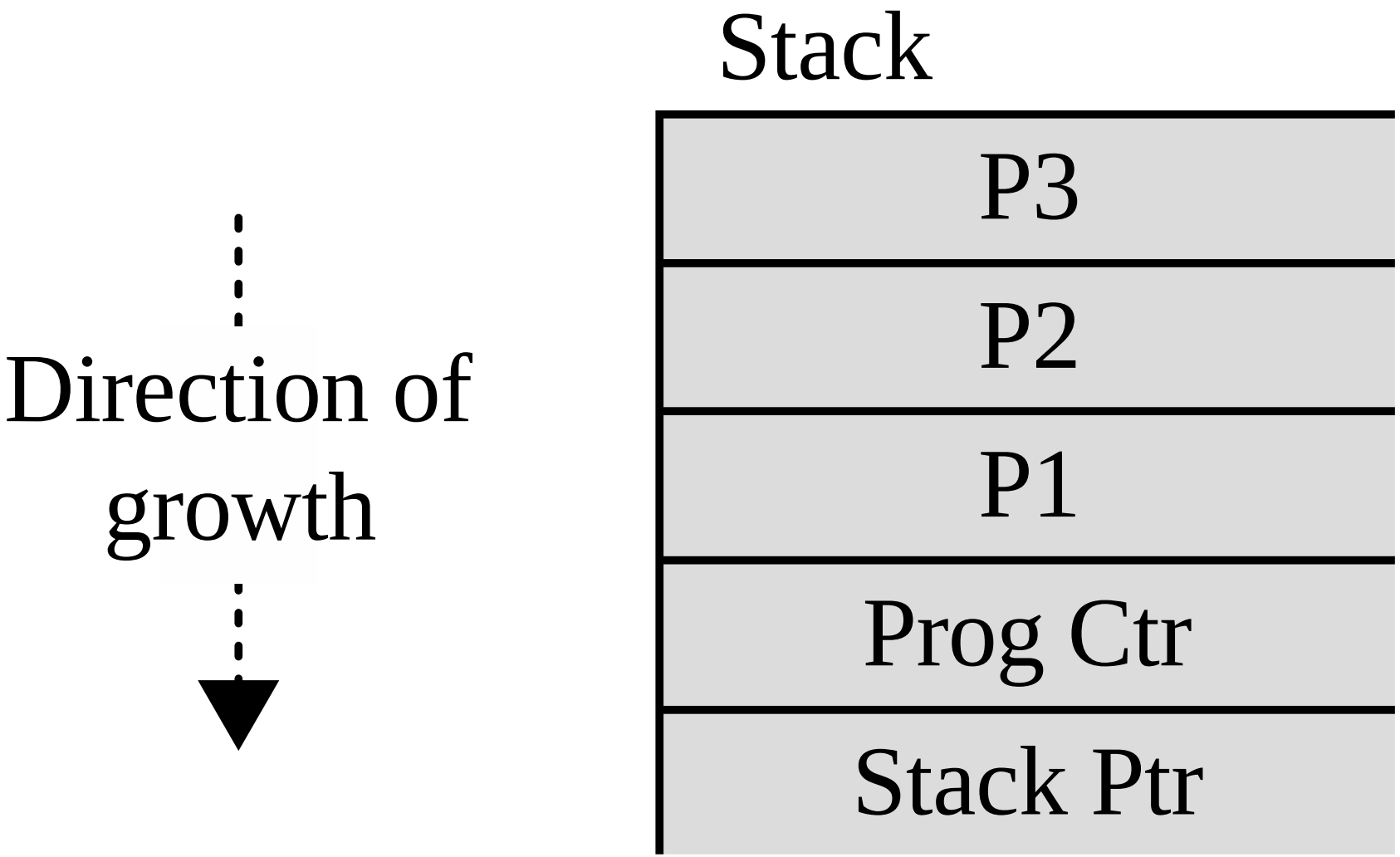



(c) Affects system data

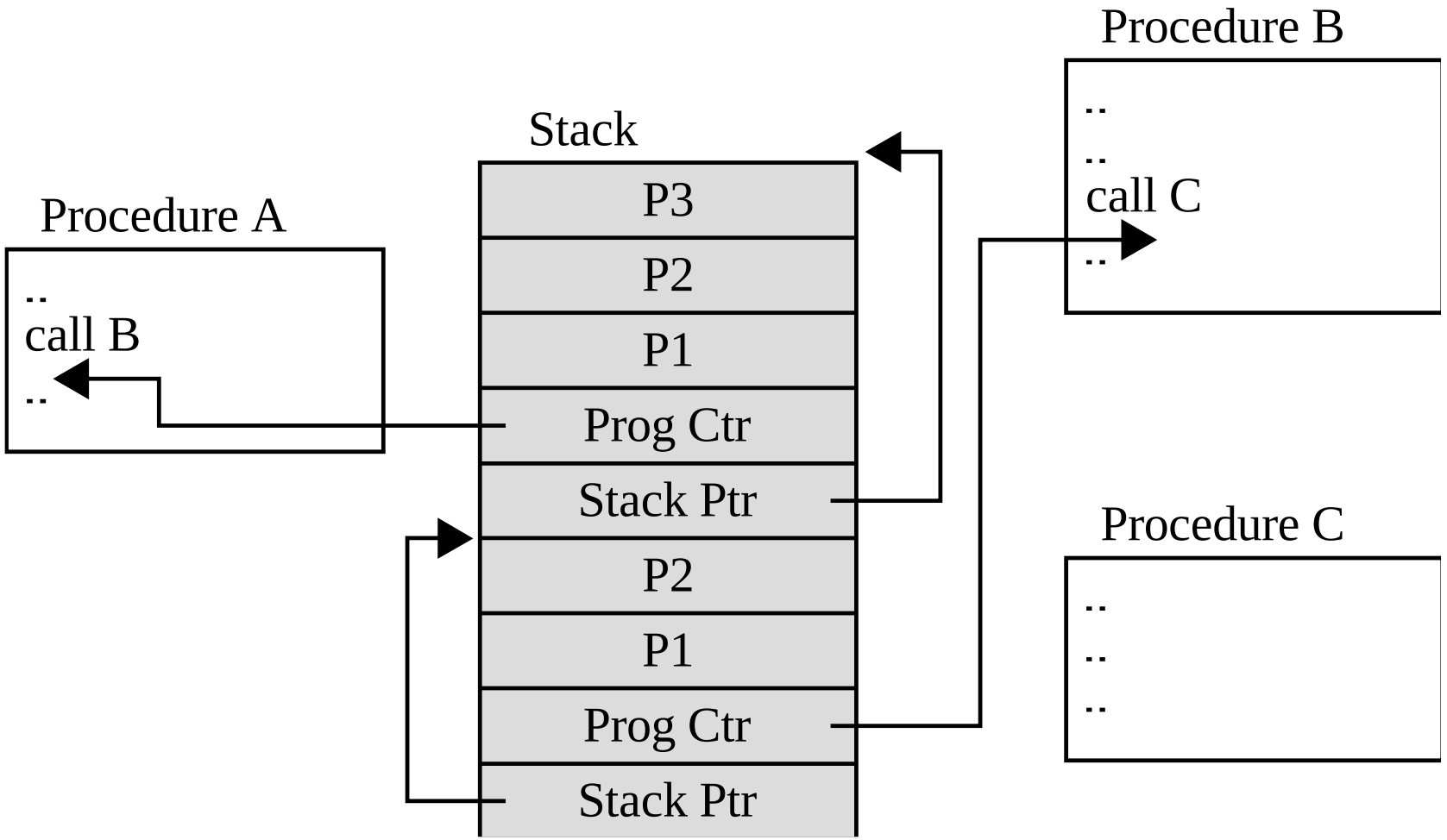



(d) Affects system code

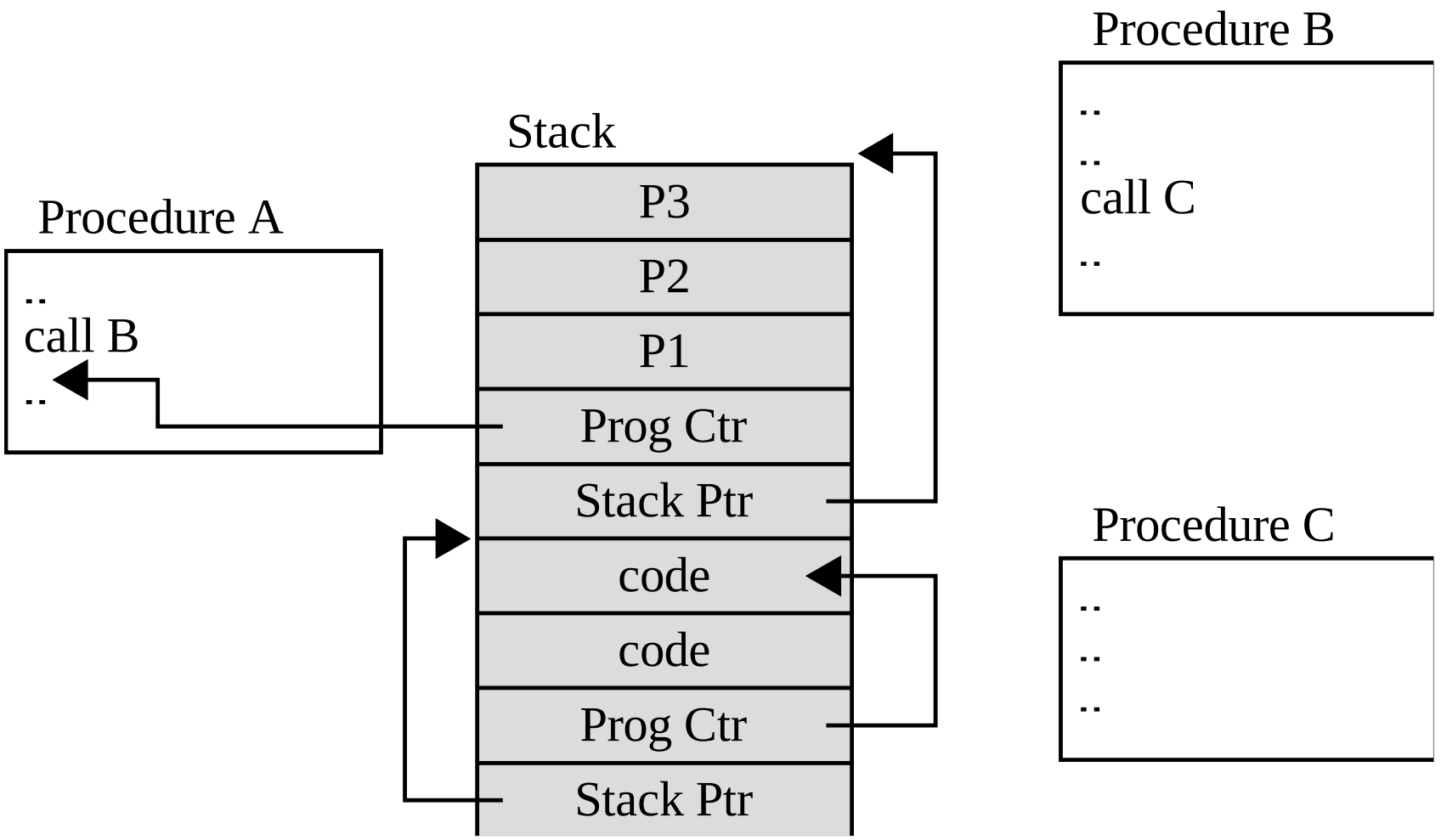
# The Stack



# The Stack after Procedure Calls



# Compromised Stack



# Overwriting Memory for Execution

- Overwrite the program counter stored in the stack
- Overwrite part of the code in low memory, substituting new instructions
- Overwrite the program counter and data in the stack so that the program counter points to the stack

# Harm from Buffer Overflows

- Overwrite:
  - Another piece of your program's data
  - An instruction in your program
  - Data or code belonging to another program
  - Data or code belonging to the operating system
- Overwriting a program's instructions gives attackers that program's execution privileges
- Overwriting operating system instructions gives attackers the operating system's execution privileges

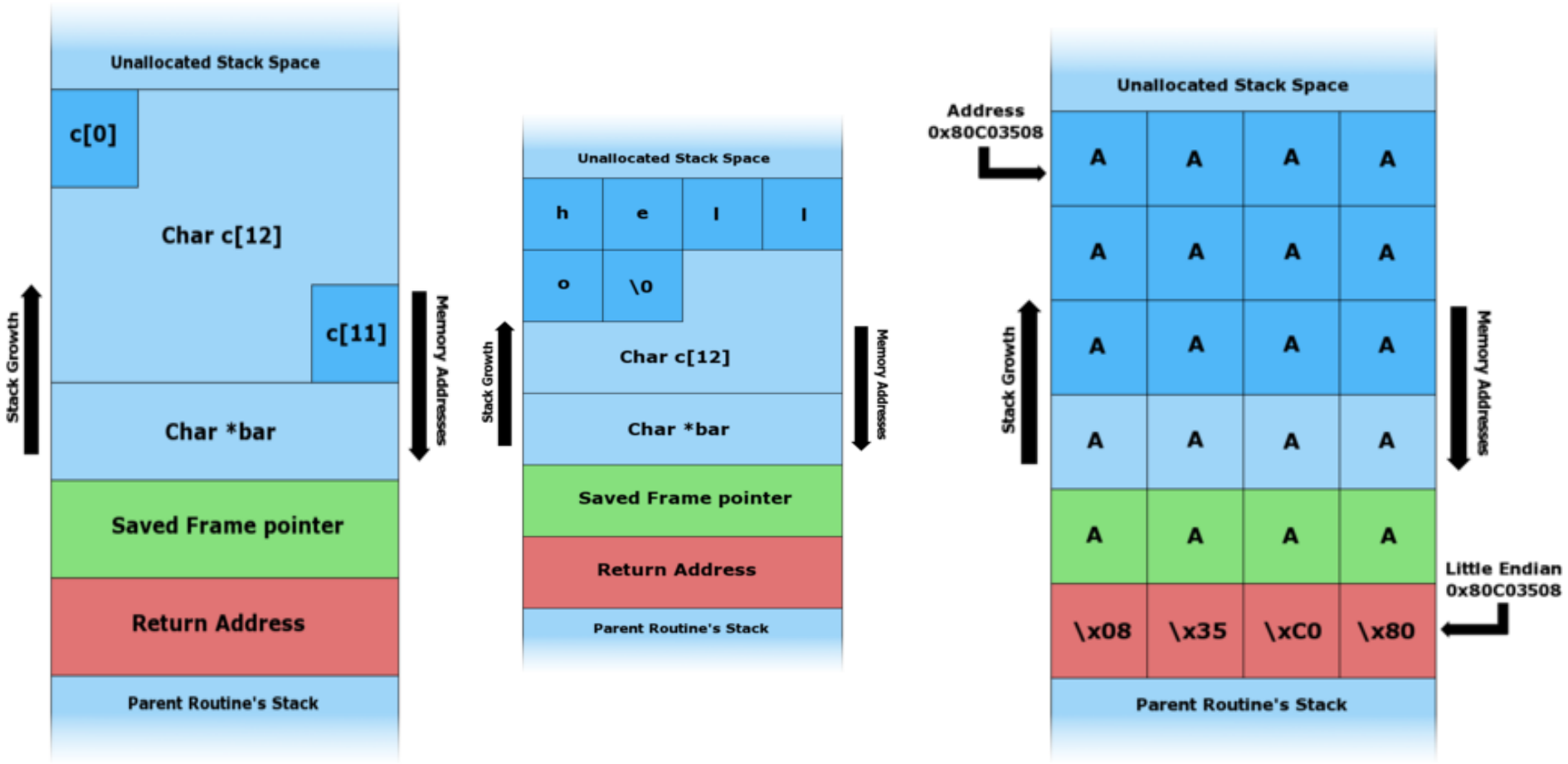
# Buffer Overflow Example 2

An example with memcpy

```
#include <string.h>
void foo (char *bar)
{ char c[12];
memcpy(c, bar, strlen(bar)); // no bounds checking...
 }
 int main (int argc, char **argv)
 {
foo(argv[1]); }
```

Link: http://en.wikipedia.org/wiki/Stack_buffer_overflow

# Buffer Overflow




- The program stack in foo() with various inputs
- A. - Before data is copied.
- B. - "hello" is the first command line argument.
- C. - "AAAAAAAAAAAAAAAAAAAA\x08\x35\xC0\x80" is the first command line argument.

# Harm from Buffer Overflows

- Overwrite:
  - Another piece of your program's data
  - An instruction in your program
  - Data or code belonging to another program
  - Data or code belonging to the operating system
- Overwriting a program's instructions gives attackers that program's execution privileges
- Overwriting operating system instructions gives attackers the operating system's execution privileges

# Overflow Countermeasures

- Staying within bounds
  - Check lengths before writing
  - Confirm that array subscripts are within limits
  - Double-check boundary condition code for off-by-one errors
  - Limit input to the number of acceptable characters
  - Limit programs' privileges to reduce potential harm
- Many languages have overflow protections
- Code analyzers can identify many overflow vulnerabilities
- Canary values in stack to signal modification

# Design Principles for Security

- Least privilege
- Economy of mechanism
- Open design
- Complete mediation
- Permission based
- Separation of privilege
- Least common mechanism
- Ease of use

# Other Countermeasures

- Good
  - Proofs of program correctness—where possible
  - Defensive programming
  - Design by contract
- Bad
  - Penetrate-and-patch
  - Security by obscurity