Lecture 03 – Control Flow

Stephen Checkoway

CS 343 – Fall 2020

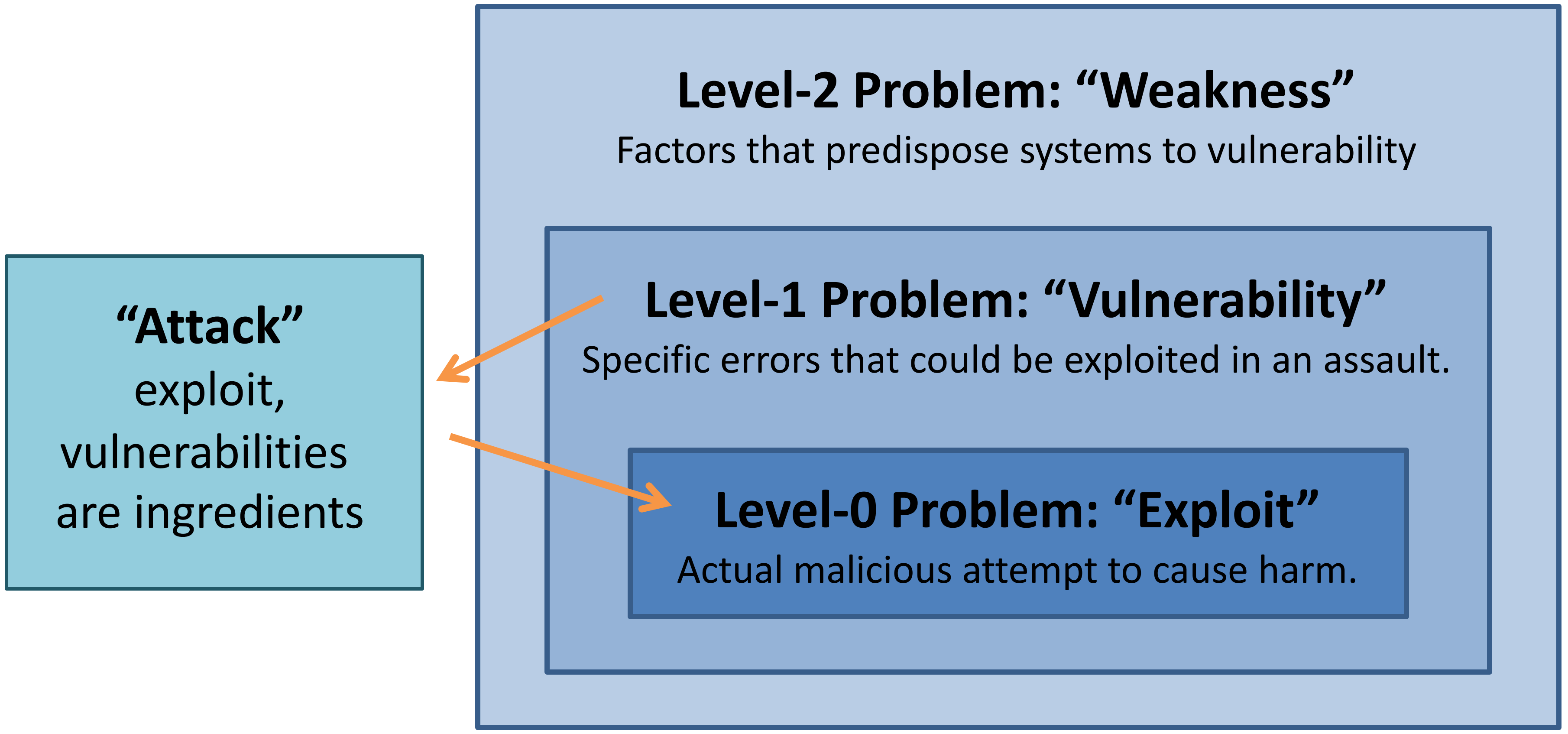
Adapted from Michael Bailey’s ECE 422

Outline

- Computer
  - CPU
  - Instructions
- The Stack (x86)
  - What is a stack
  - How it is used by programs
  - Technical details
- Attacks
- Buffer overflows

- Adapted from Aleph One’s “Smashing the Stack for Fun and Profit”

“Insecurity”?


“Attack”
exploit,
vulnerabilities
are ingredients
Level-2 Problem: “Weakness”
Factors that predispose systems to vulnerability
Level-1 Problem: “Vulnerability”
Specific errors that could be exploited in an assault.
Level-0 Problem: “Exploit”
Actual malicious attempt to cause harm.

Why Study Attacks?

- Identify vulnerabilities so they can be fixed.
- Create incentives for vendors to be careful.
- Learn about new classes of threats.
  - Determine what we need to defend against.
  - Help designers build stronger systems.
  - Help users more accurately evaluate risk.

```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
        OSStatus        err;
        ...

        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        ...

fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

Virtual memory

- Each running process has its own virtual memory space
  - Your computer has a bunch of RAM
  - RAM is an array of bytes indexed from 0
  - It would be bad if any process could read/write any byte of memory
  - The OS and hardware carve up memory and hand it out to processes

Image from Wikipedia

Virtual address space

- OS presents each process with the fiction that it has access to the entire valid range of memory from index 0 to the maximum index (2^32 - 1 or 2^64 - 1)
- It does this by mapping virtual addresses used by processes to physical addresses used by the hardware

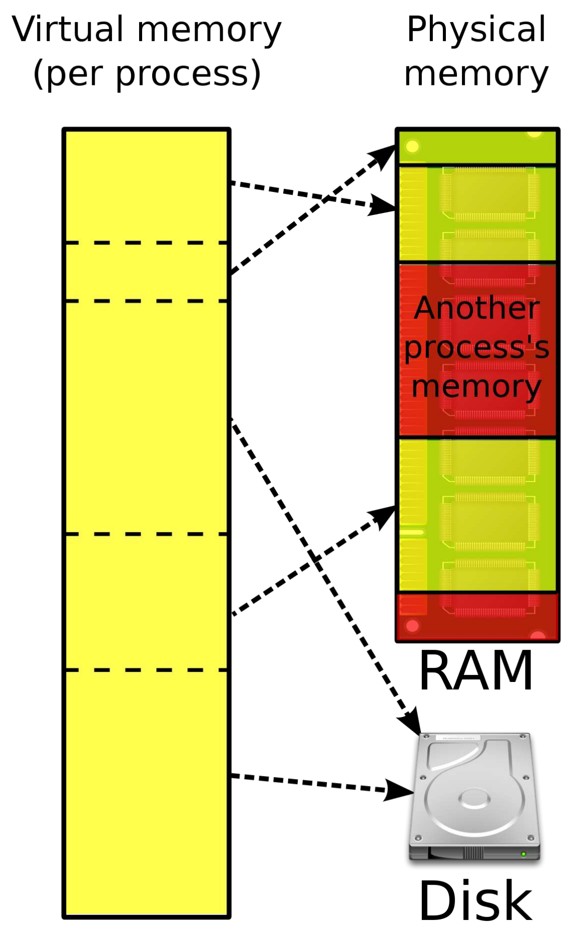


Image from Wikipedia

Virtual address space layout

- Each function called in a program is allocated a *stack frame* on the call stack; it stores
  - The return address
  - Local variables
  - Arguments to functions it calls
- The software maintains two pointers
  - Stack pointer: points to the top (lowest address) of the stack
  - Frame pointer: points to the call frame (optional)

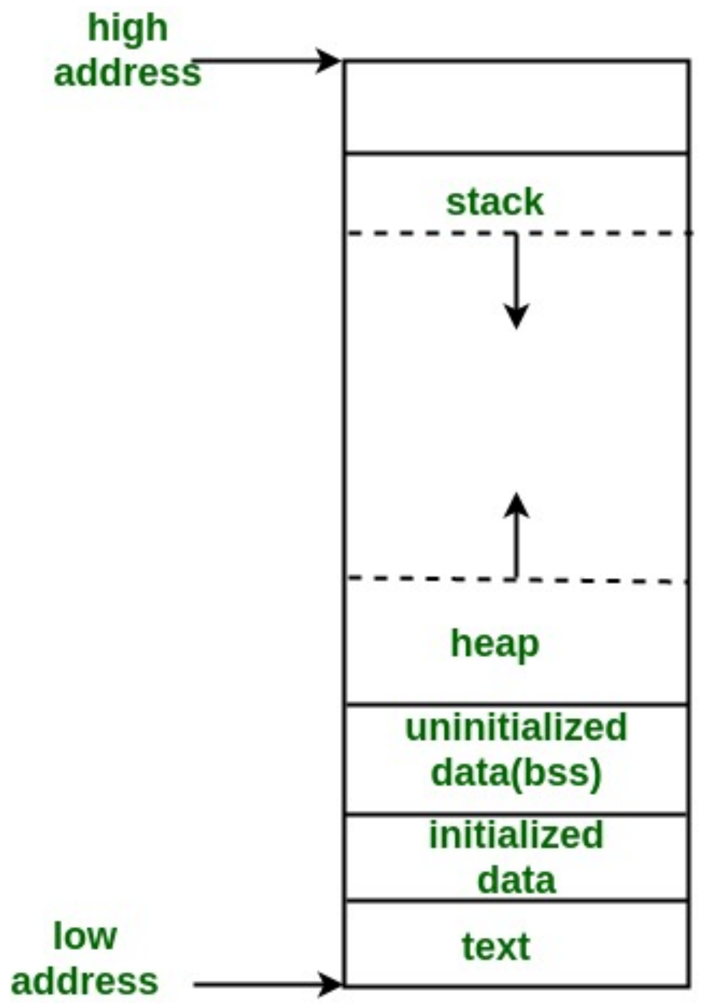


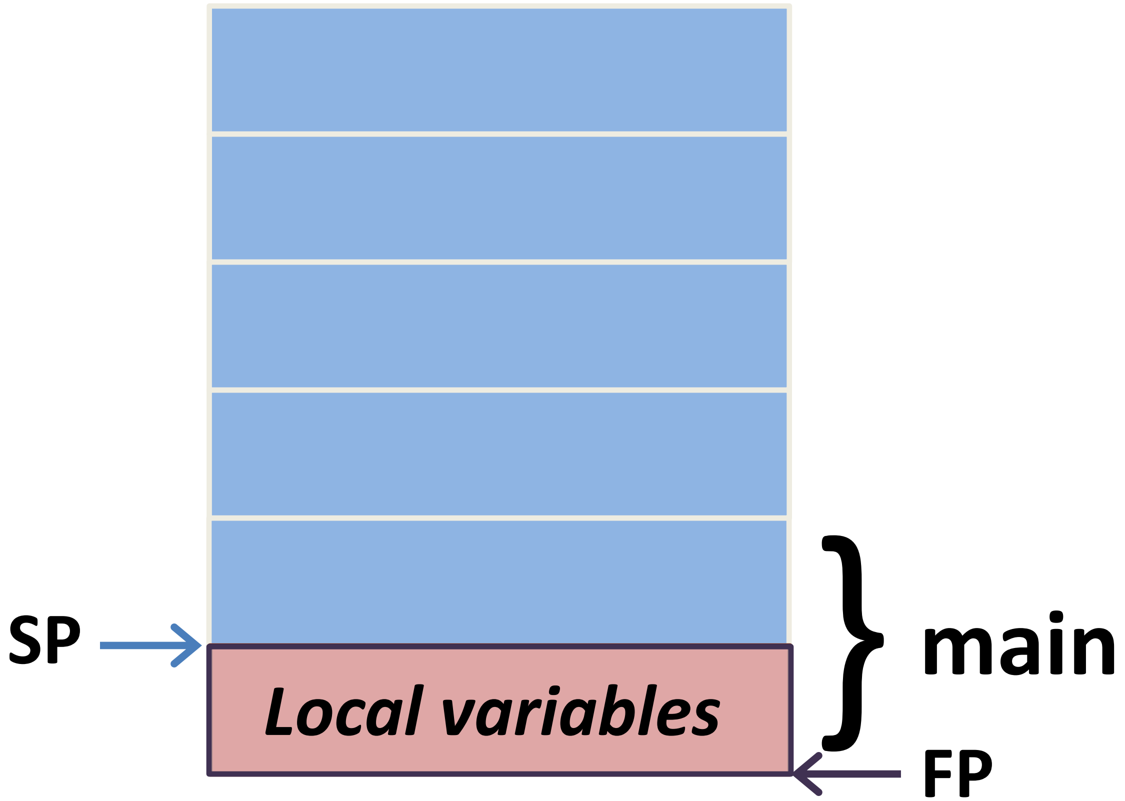
Image from https://blog.adafruit.com/2019/07/19/the-pitfalls-of-uninitialized-memory-programming-c-rust/

example.c

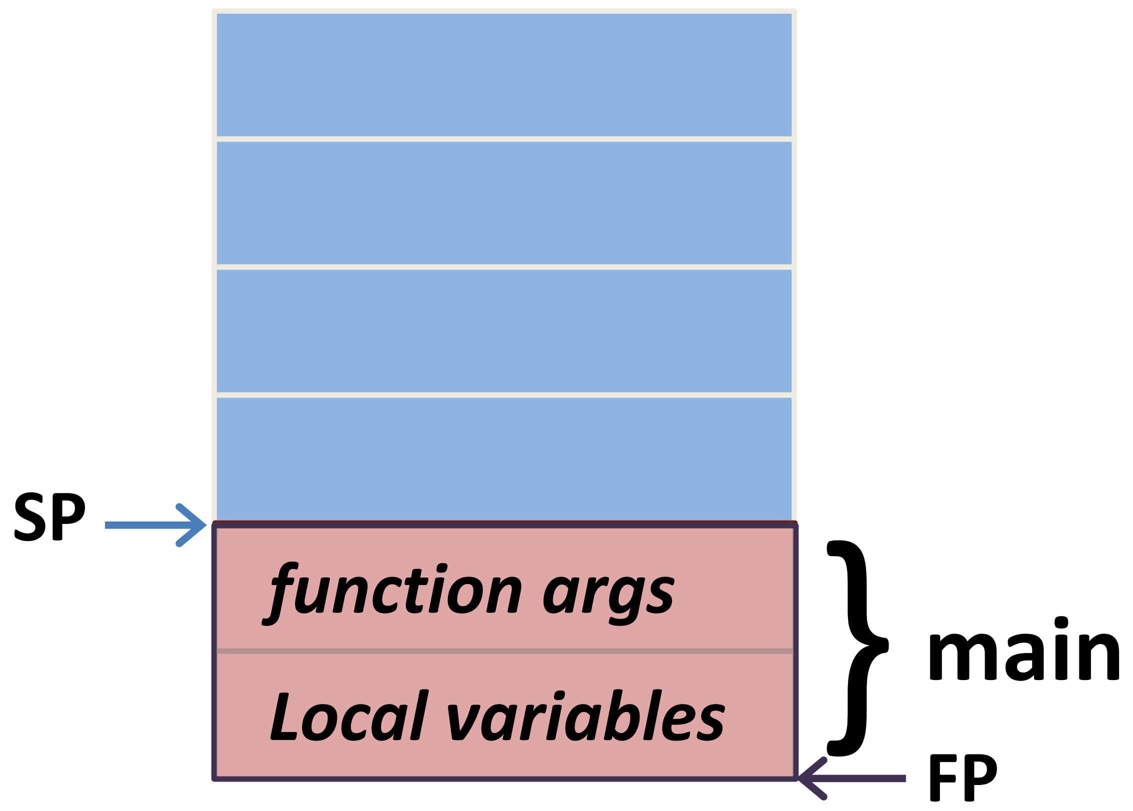
```
void foo(int a, int b) {
    char buf1[10];
}

void main() {
    foo(3,6);
}
```

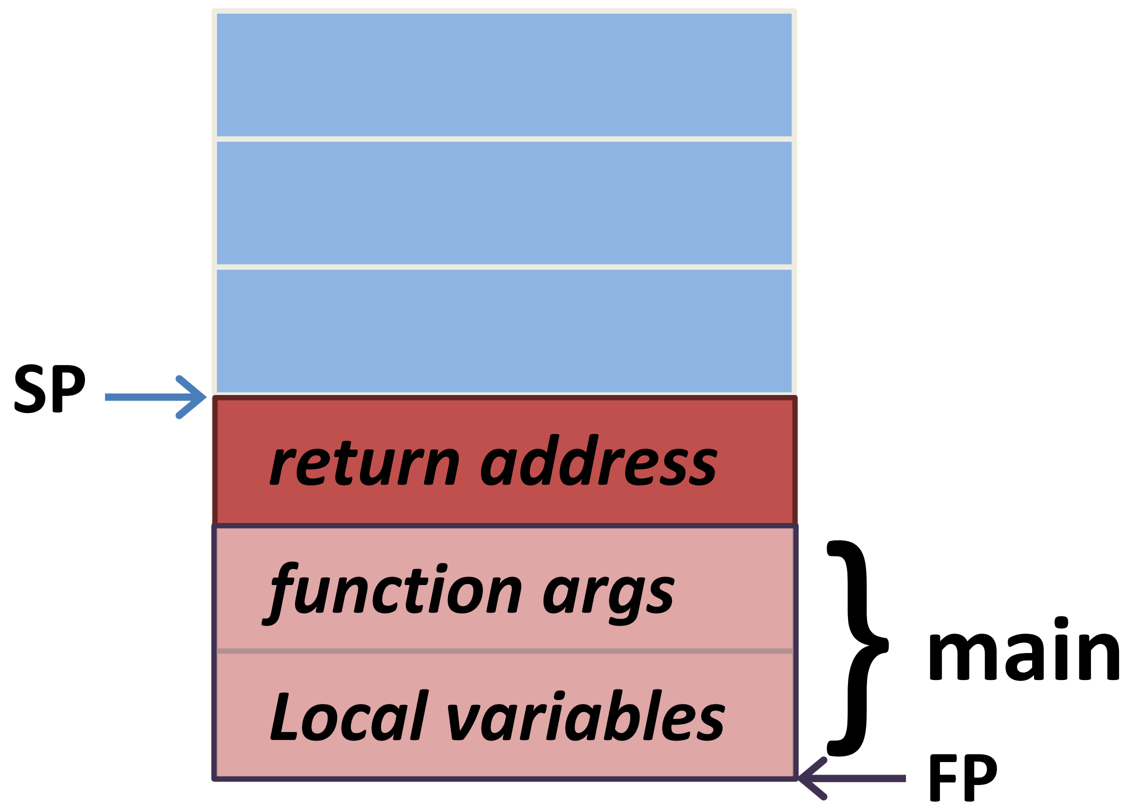
C stack frames



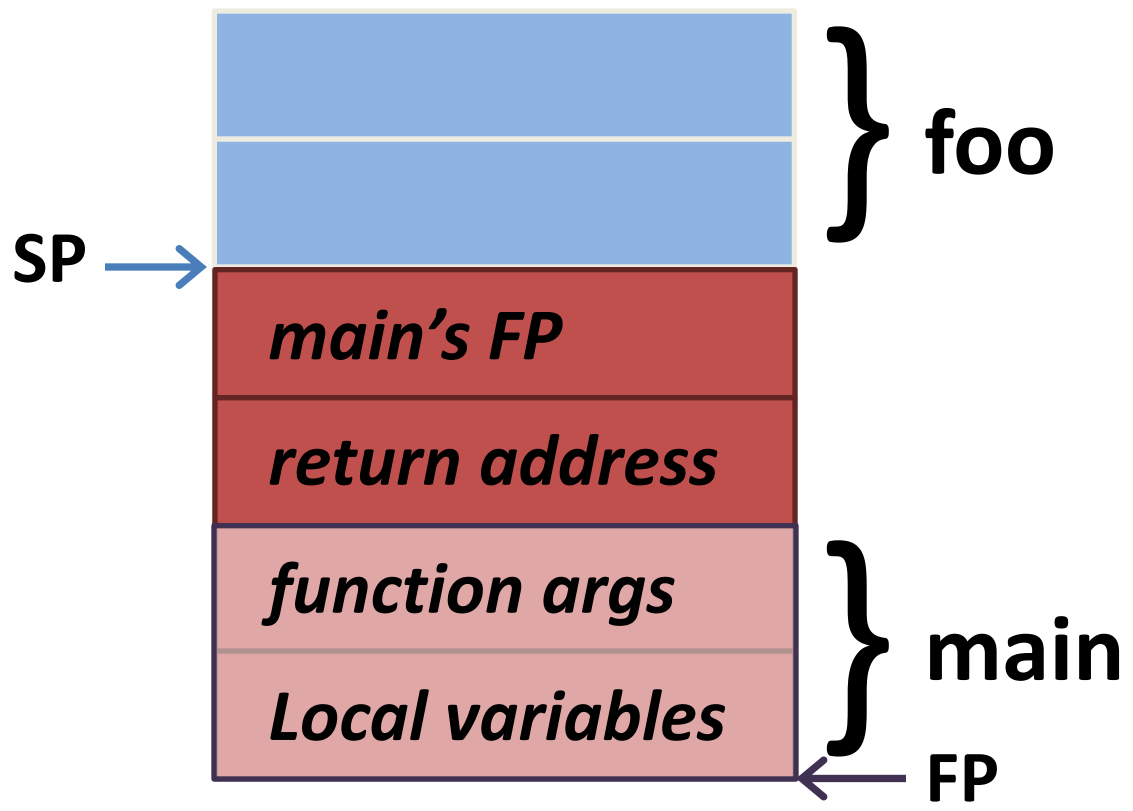
C stack frames



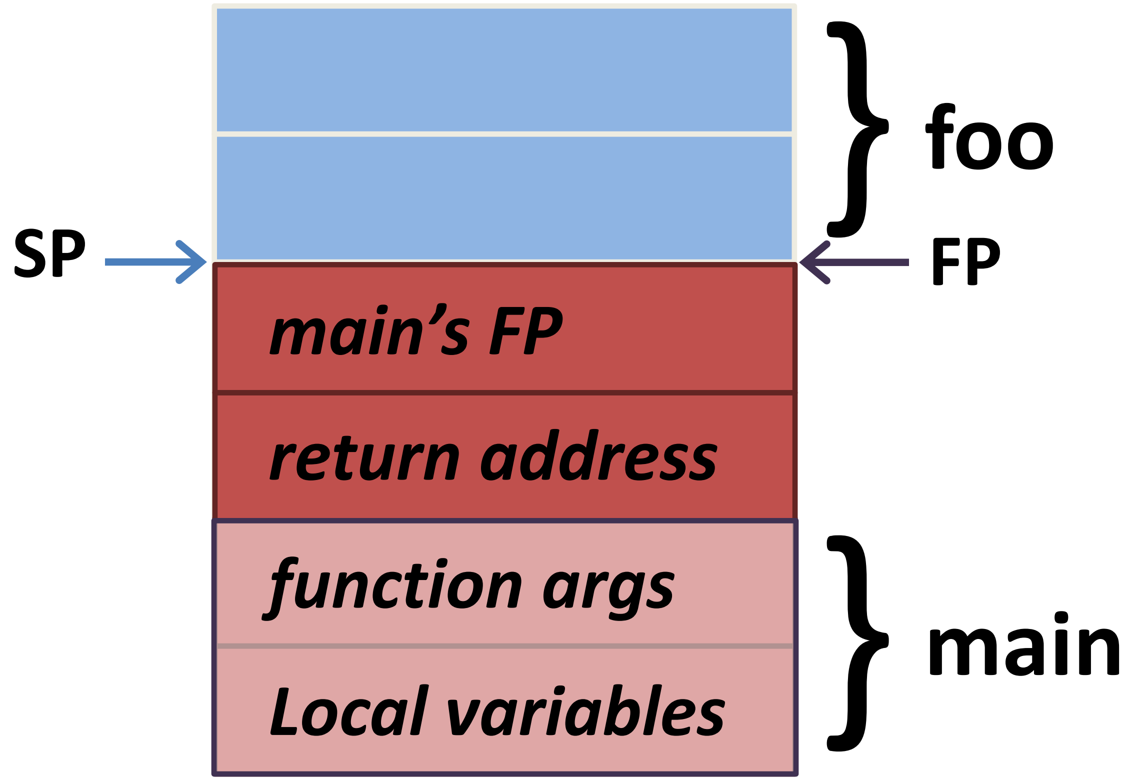
C stack frames



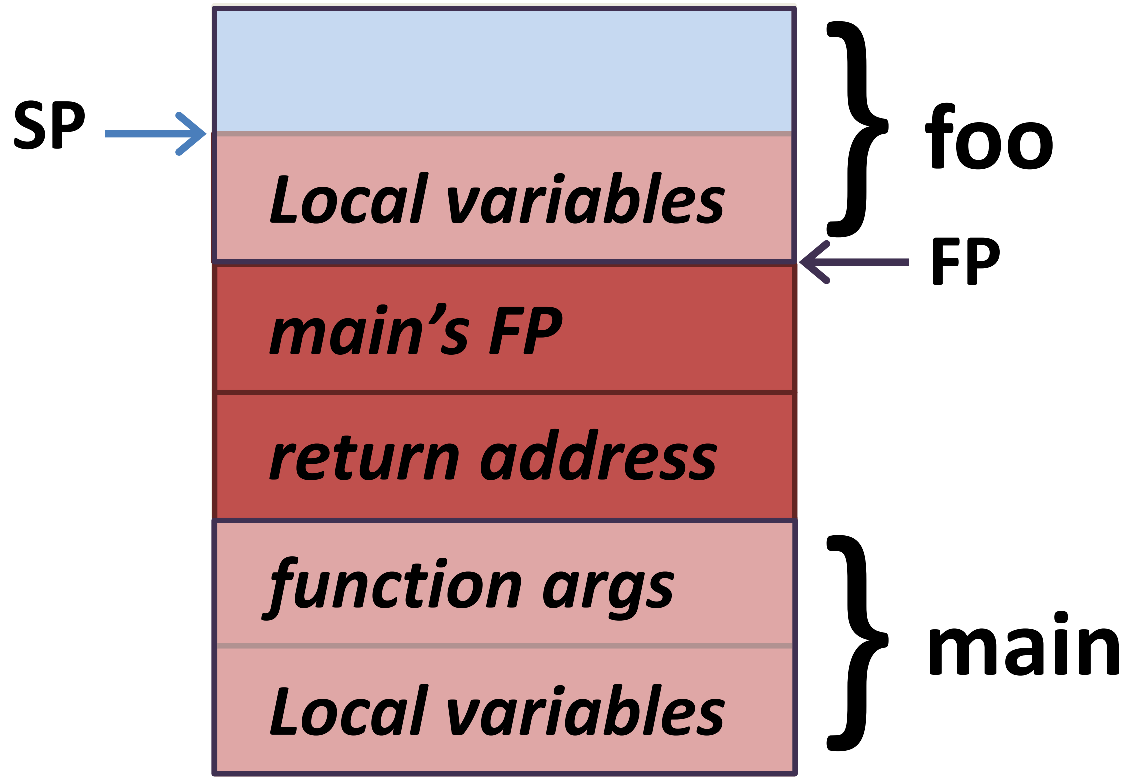
C stack frames



C stack frames



C stack frames


SP
Local variables
main's FP
return address
function args
Local variables
foo
FP
main