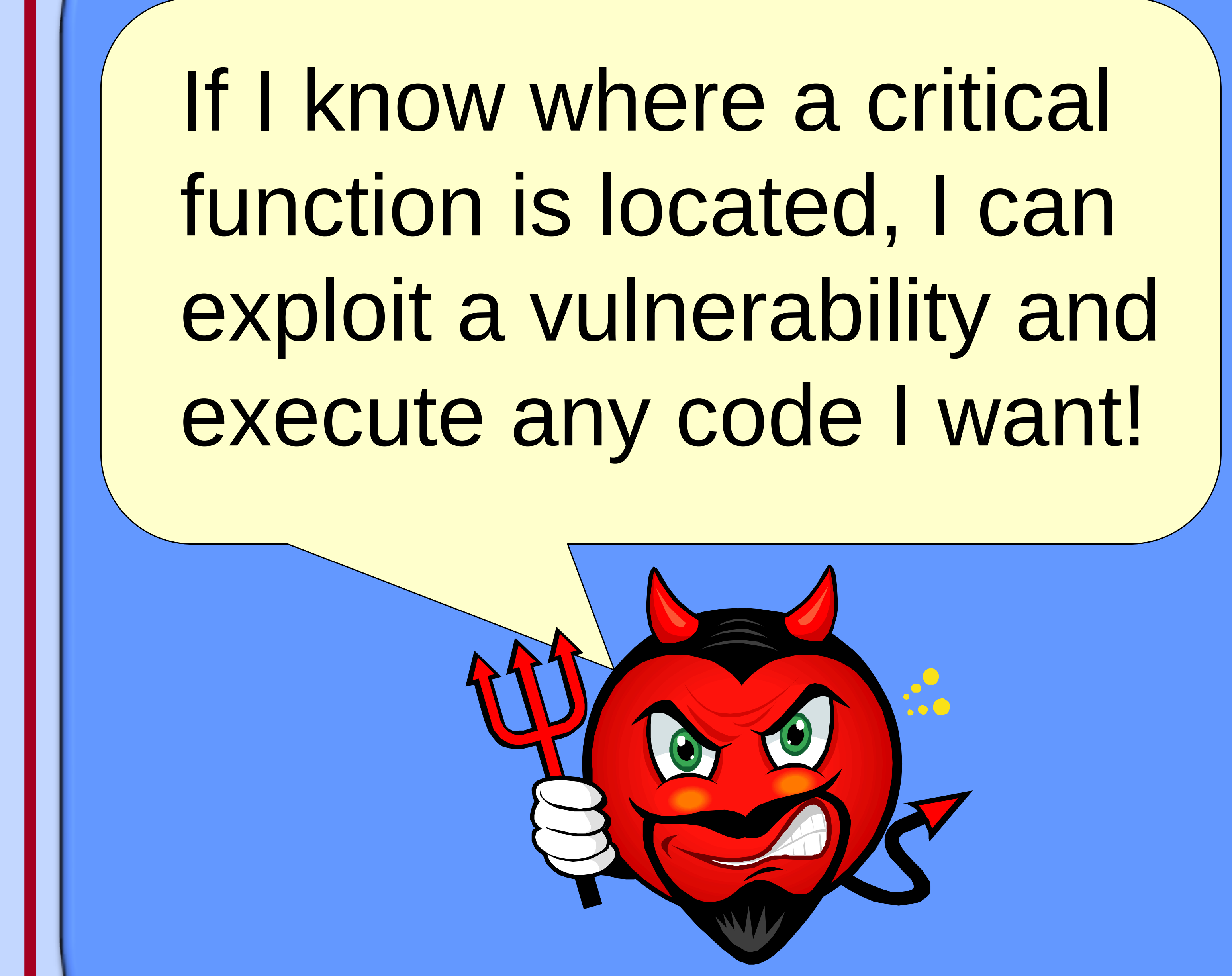
Dyn
inst

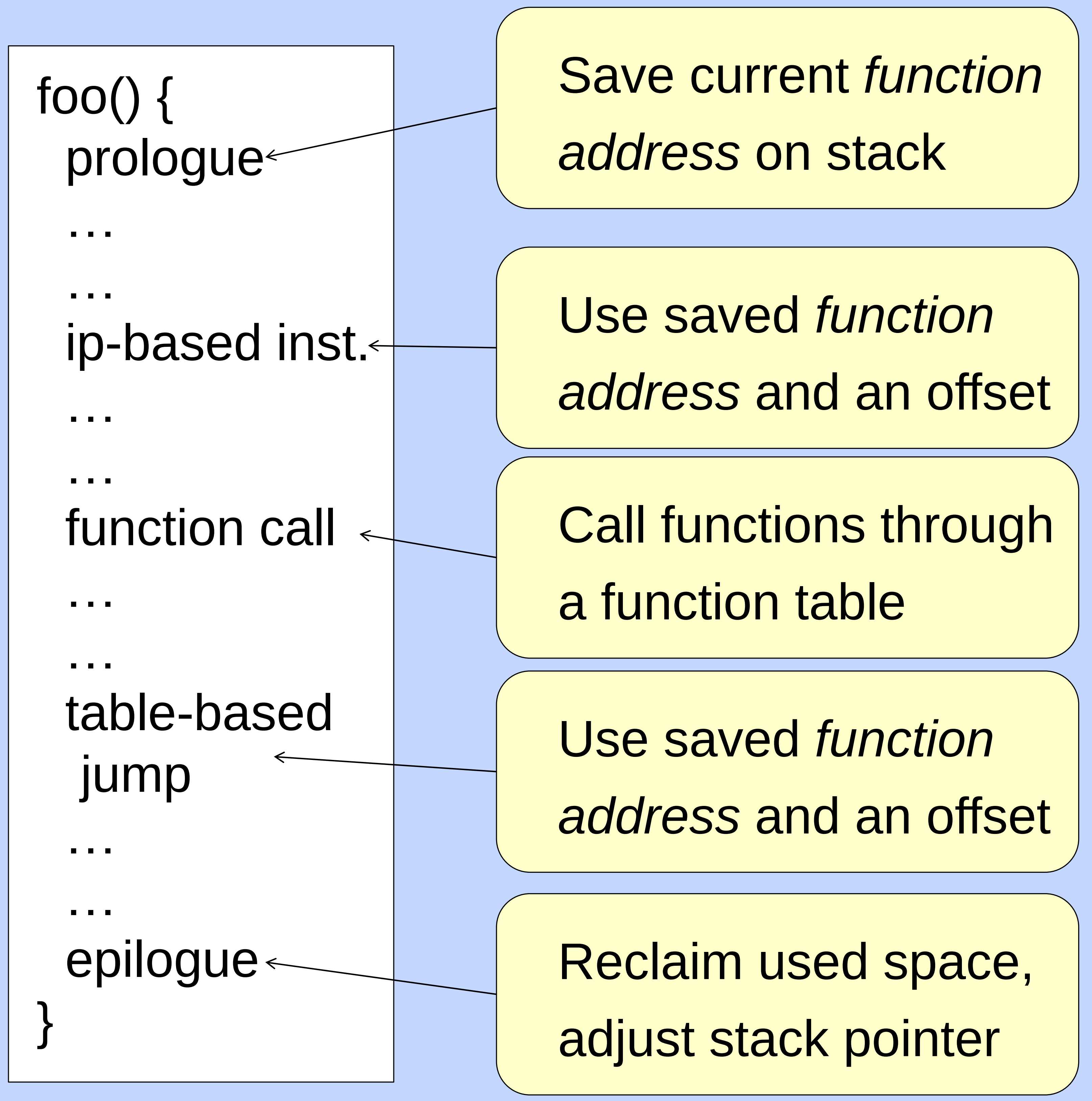
Security through Runtime Relocation

Function Relocation in Edited Binaries


If I know where a critical function is located, I can exploit a vulnerability and execute any code I want!

Return-to-libc Attack

- Purpose
  - Jump ('return') to a critical function to execute malicious code
- Requirements
  - A vulnerability such as a possible buffer overflow
  - Knowledge of the location of a critical function
- Method
  - Overflow the buffer to overwrite the return address stored on stack
  - Execution will 'return' to address written to stack


foo() {
prologue
…
…
ip-based inst.
…
…
function call
…
…
table-based jump
…
…
epilogue
}
Save current *function address* on stack
Use saved *function address* and an offset
Call functions through a function table
Use saved *function address* and an offset
Reclaim used space, adjust stack pointer

I will relocate functions during execution, so you will not be able to find critical functions!

Fully Relocatable Binaries

- Executables work as a whole
  - Changing relative distances break execution
- Solution: **Rewrite executables to be more relocatable**
  - Store the start address of function on stack
  - Rewrite instructions to use that address and a constant offset
- Create a table to store function locations
  - Calls use stored address in the table

Secure Executables

- **Attacks will fail even if return address can be written**
  - Exact location of function is unknown
- Attacks may even fail when the address is somehow obtained
  - Target function might be relocated before the attack