Anti-Patterns:
- No worst case stack size analysis
- Use of recursion
- No memory protection for stack

The stack stores data for subroutines
- Automatic (non-static) variables
  - Also, subroutine & interrupt register saves
- Calls put data on stack
  - Interrupts & RTOS calls put data on stack too
- But what if the stack overflows?
  - Need to handle worst-case stack size
Stack Overflow Corrupts Memory

- If stack gets too big, it stomps on other memory: **Stack Overflow**
  - Can corrupt static variables and globals
  - Can corrupt RTOS data structures
    - System-wide task information corruption

- Can cause system crashes
  - Worse, can cause subtle system corruption
    - Task death, task period alteration
    - Security exploits via access to OS data
Prevent & Detect Stack Overflow

Preferred approaches:
- Static analysis of stack depth
  - Tool can figure out maximum depth
  - MMU hardware memory protection

At Run-Time: Stack Sentinels
- At system start, fill stack with a sentinel value (e.g., \texttt{0xAA44CC33})
- Program execution writes to stack
  - Sentinels permanently overwritten
- Periodically check to see how many sentinels are left (stack size margin)
Determine worst case stack depth

- Sentinels are a good start
  - But you might not see true worst-case depth in testing
  - Worst-case stack depth for deeply nested calls + safety margin
- Use a tool if you have one, or use a disassembler
  - PLUS: Biggest interrupt service routine stack use
  - PLUS: RTOS call use of stack (can be significant)

Protect stack at run time

- Use MMU hardware protection if you have it
- Use sentinels & periodic check to detect stack overflow
  - Also helps with experimental confirmation of depth analysis

Avoid recursion – makes worst case problematic

- Be mindful that big data structures can make stack big
The #1 programmer excuse for legitimately slacking off:

"My code's compiling."

Hey! Get back to work!

Compiling!

Oh. Carry on.

https://xkcd.com/303/