The Heap Side of Embedded

is

The Dark side of Embedded

Mohammad Mazarei

Introduction

I am an Electronic and Embedded System Engineer with .over 10 years of professional experience

Have been in contact with exciting topics such as Electronics, Programming (mainly for Hardware) Embedded Linux, Real-time operating systems, Embedded systems, Sensors and telecommunication networks, IoT devices, etc

and co-founder of **sisoog.com**

C Memory Management

memory management in C

Static memory allocation

Stack-based memory management

Heap-based memory management

Static memory preallocation

Memory pools

Static memory allocation

Every byte of RAM is accounted for at compile time.

- No memory leaks
- dangling pointers
- allocation failures
- No recursion or reentrant code possible.
- Interrupt routines can't call functions

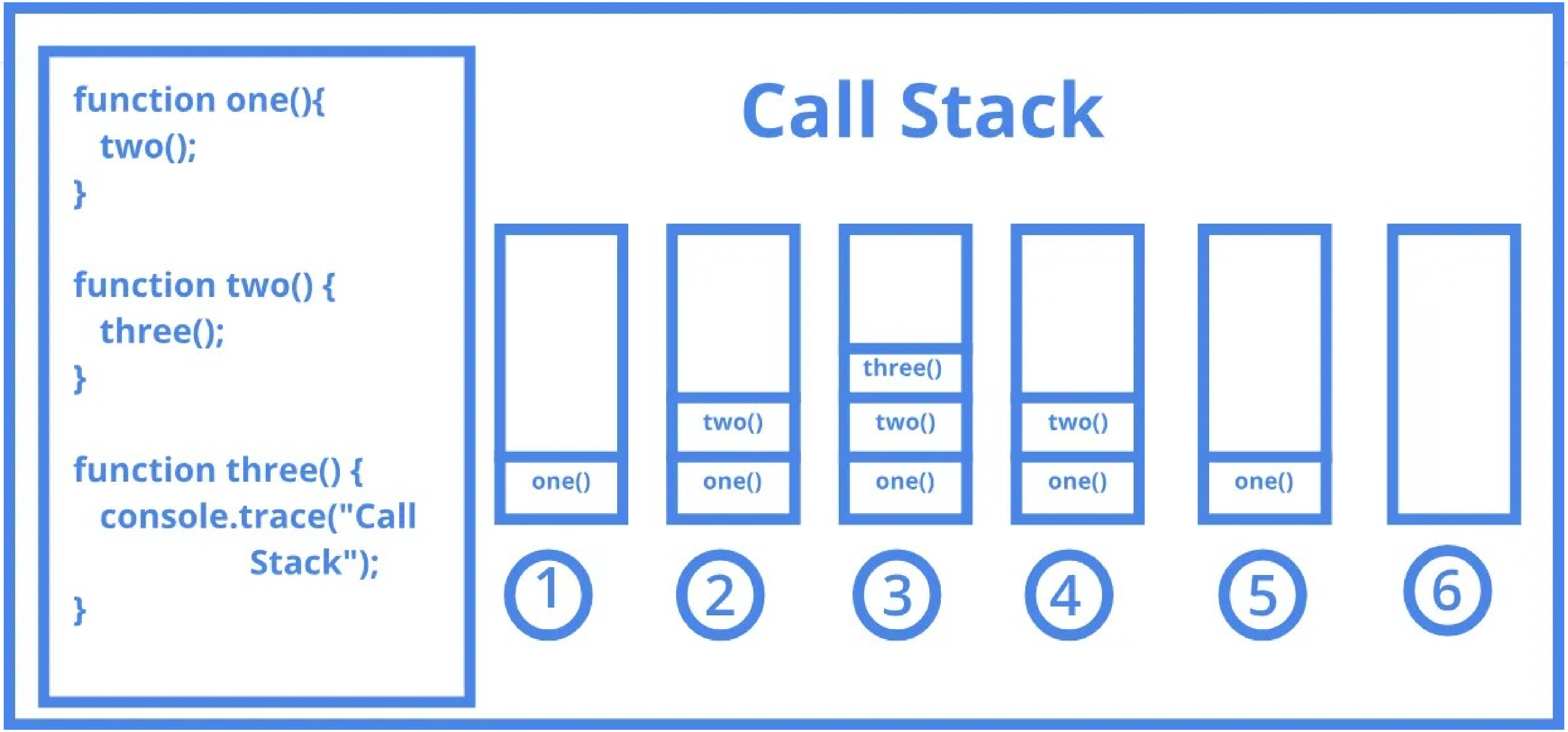
Static allocation doesn't scale well to large systems

Stack-based memory management

New Hardware Can Support Stack memory management

- A separate memory block (stack frame) is needed for every function call
- The stack grows and shrinks dynamically as the program runs.
- Manage by Compiler :)
- In multitasking systems, each task has its own stack

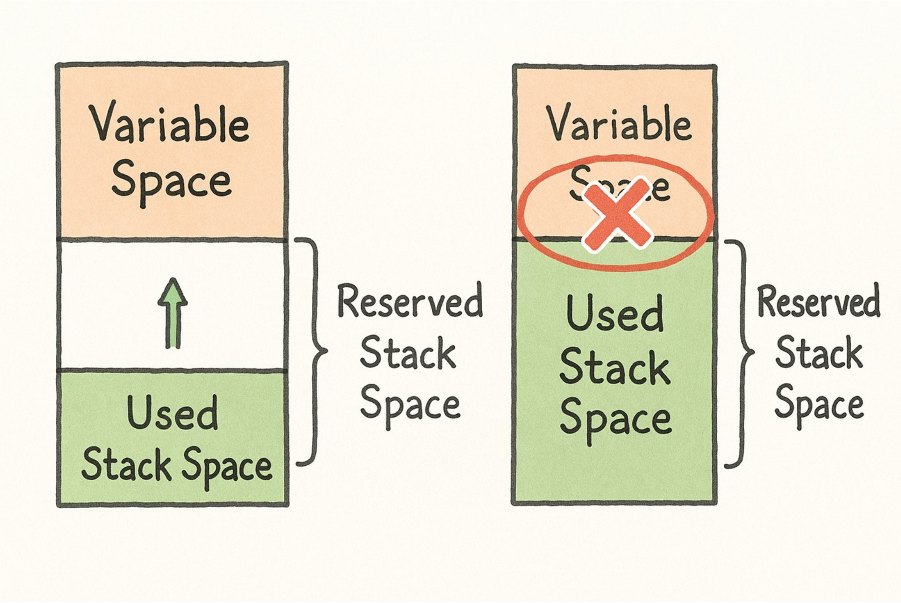
How to Stack Work



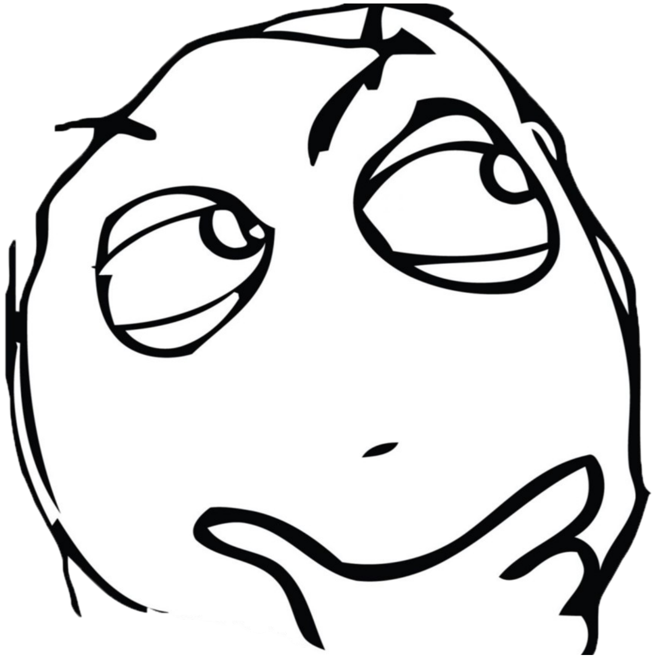
Challenges with Stack Usage

Worst-case stack size is hard to predict at compile time.

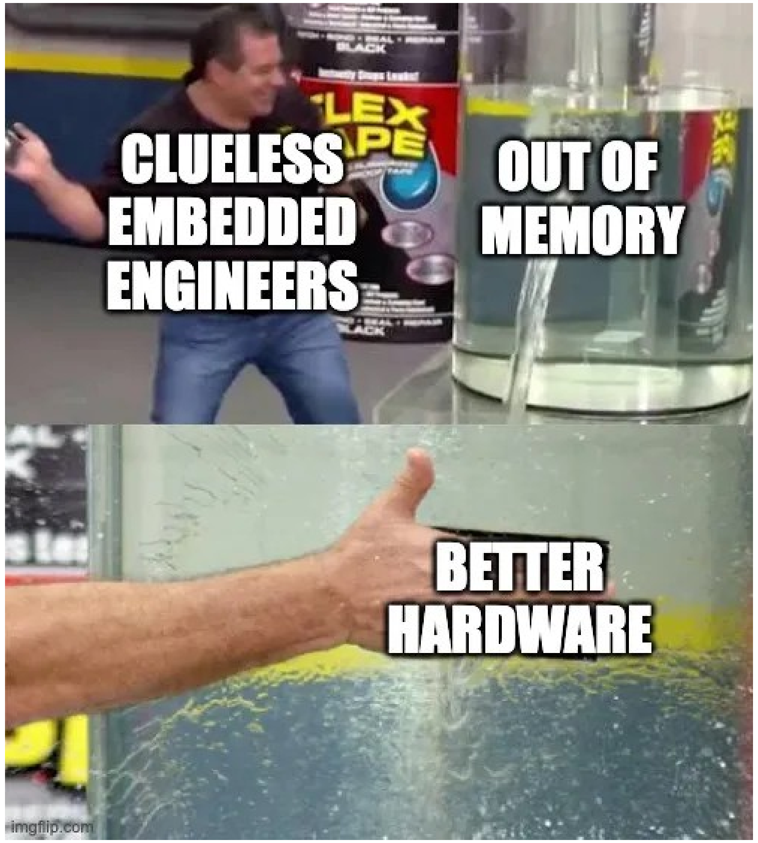
Stack overflow risk: Some stacks may overflow while others remain underutilized.



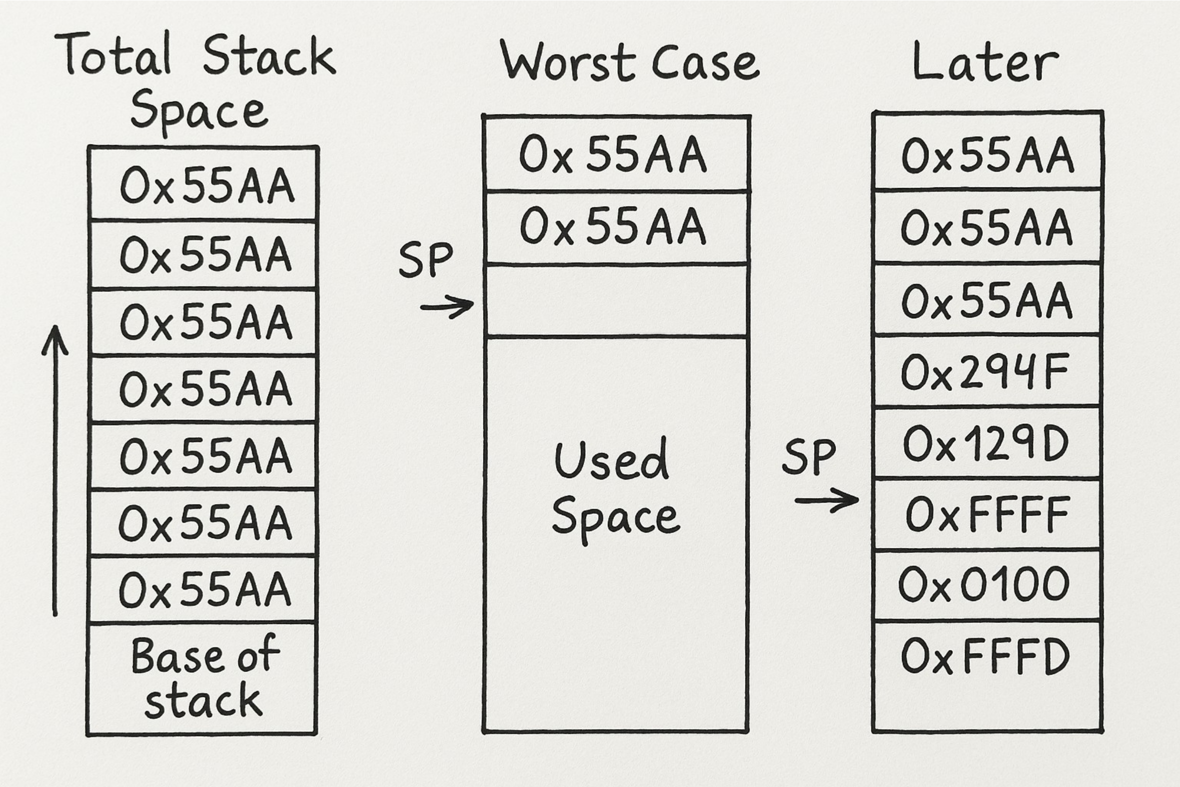
How to guarantee no stack overflow？



Add More Memory!



Stack size tracing



Dynamic allocation and heap

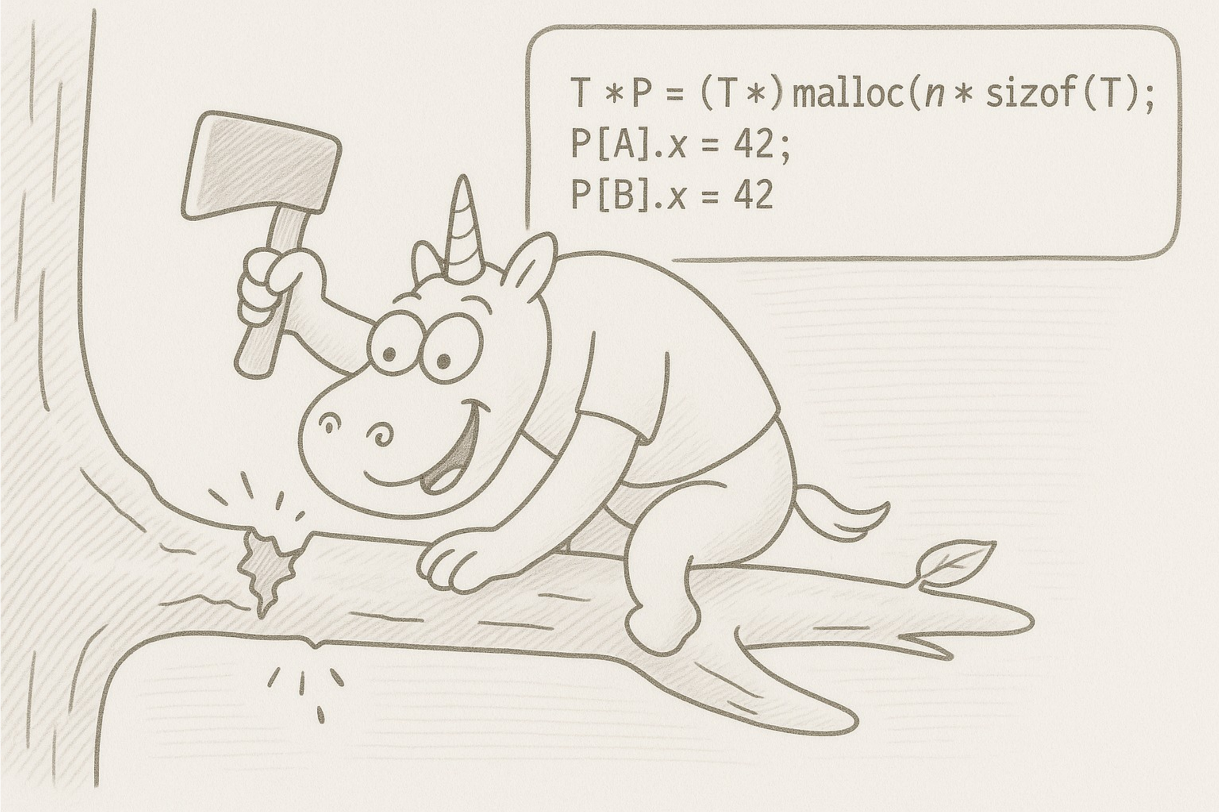
MISRA C:2004 – Rule 20.4

"Dynamic heap memory allocation shall not be used."

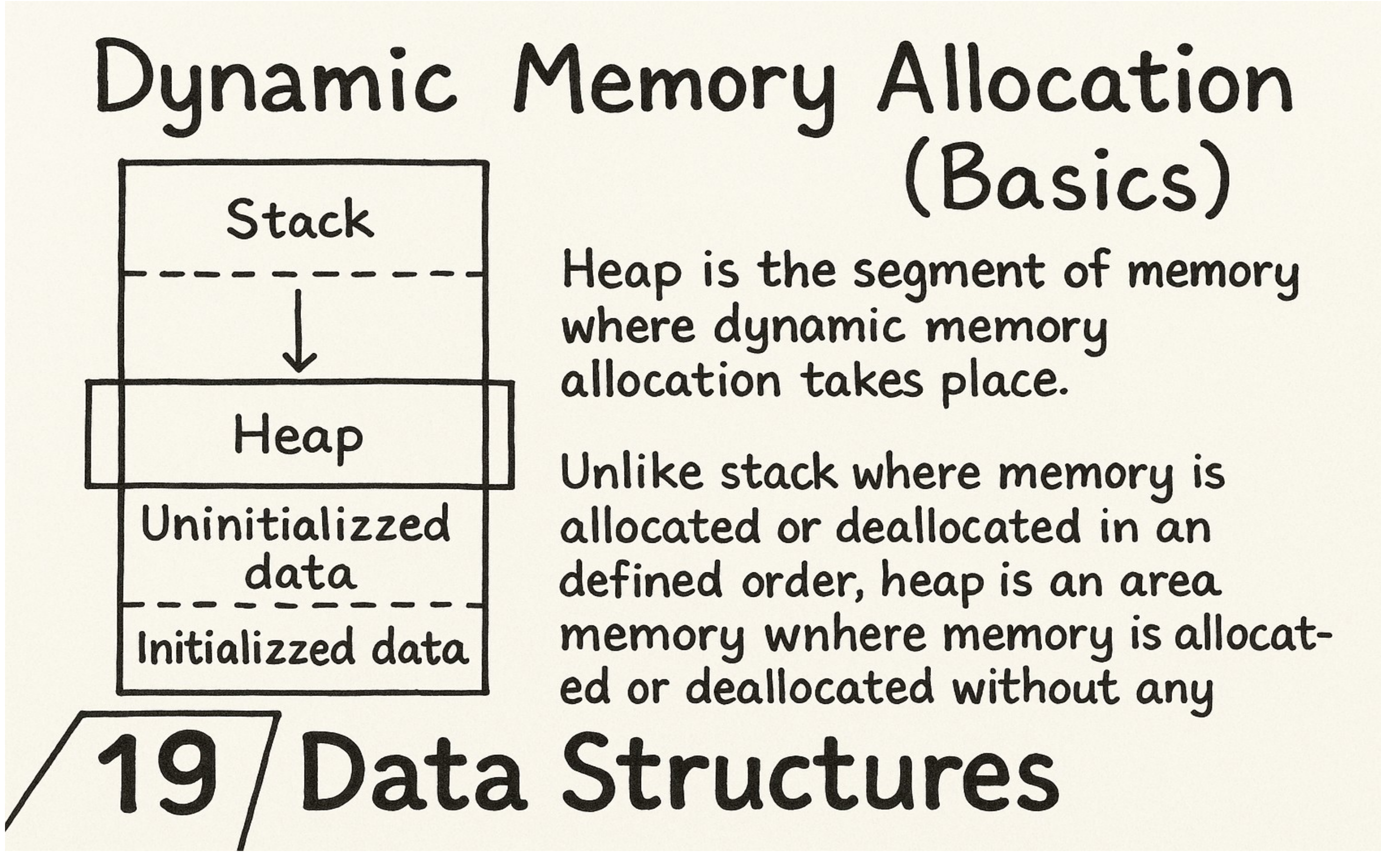
MISRA C:2012 – Rule 21.3

"The memory allocation and deallocation functions of <stdlib.h> shall not be used."

https://misra.org.uk/



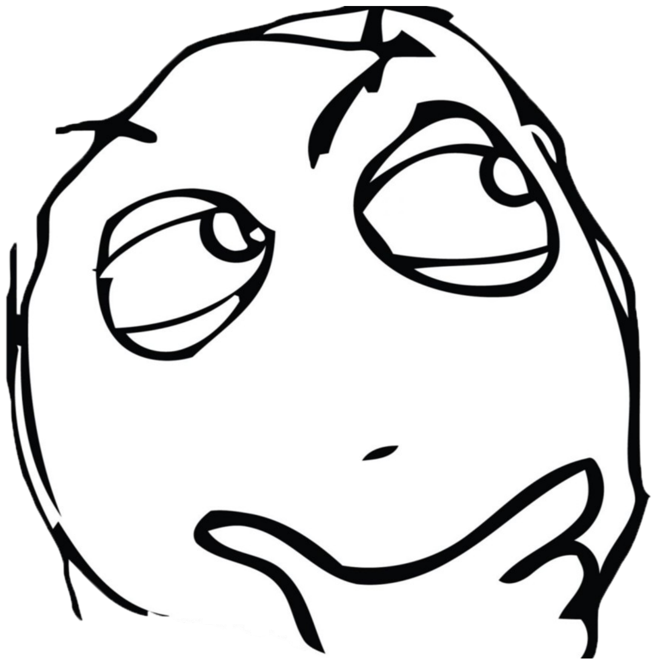
Waht is Heap Memory



Why We Avoid Dynamic Memory Management

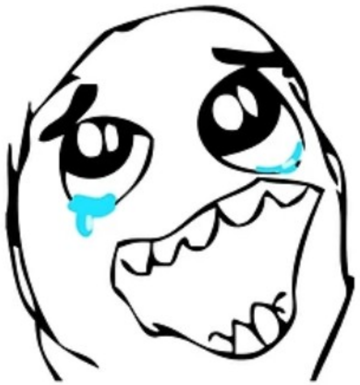
Limited Heap Size:

Embedded systems often have limited resources, including RAM.
The heap, where dynamic memory is allocated, is typically much smaller than in regular desktop applications.
As a result, excessive or inefficient use of dynamic memory can quickly deplete the available heap, leading to system instability.

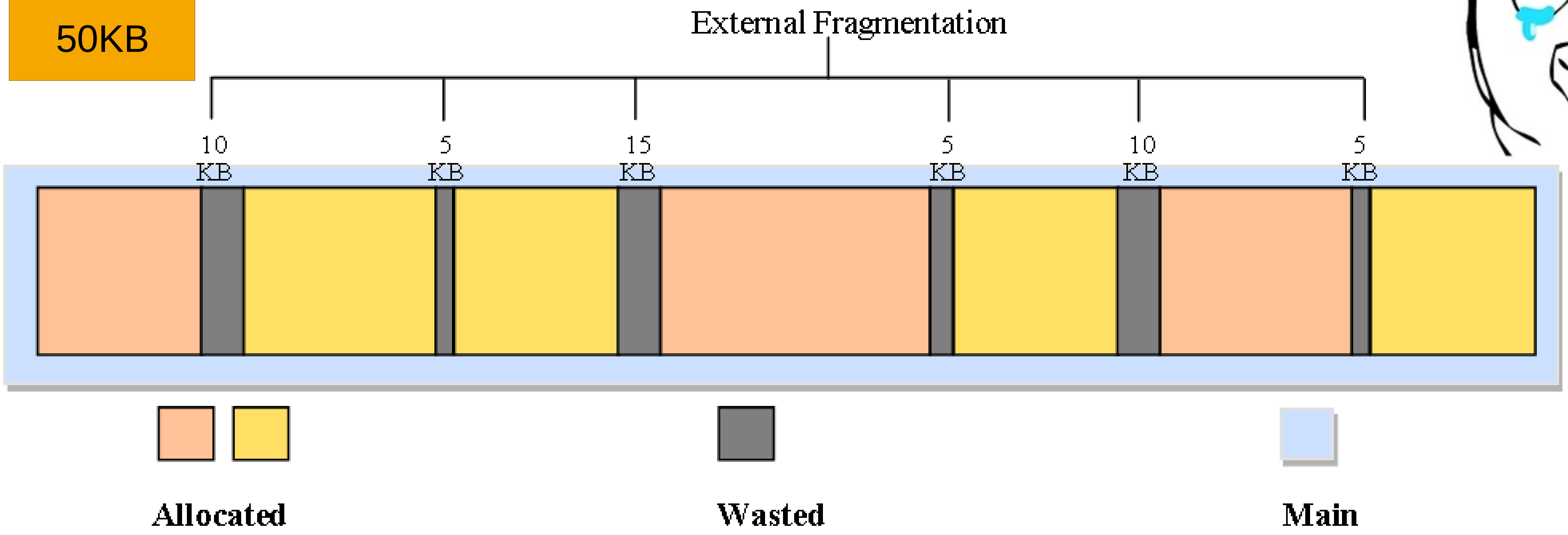


Memory Fragmentation

We have 50 KB of free memory, but we can’t even allocate 20 KB of it!



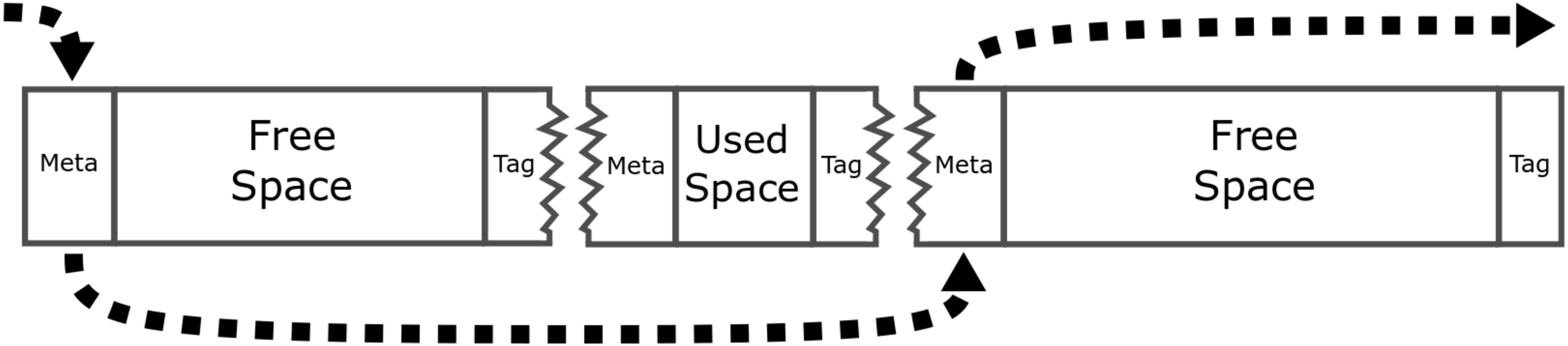
50KB



Memory Management Overhead

Memory Overhead

Performance overhead



And so many reasons

User data memory fragmentation

The problem with having user data fragmented all over the place is mainly an issue on mid-range MCUs like for example >Cortex M4 or PowerPC e200. Such MCUs utilize branch prediction at some extent and may use both data and/or instruction cache. The MCU cannot effectively load our data into data cache lines if it is fragmented all over the place.

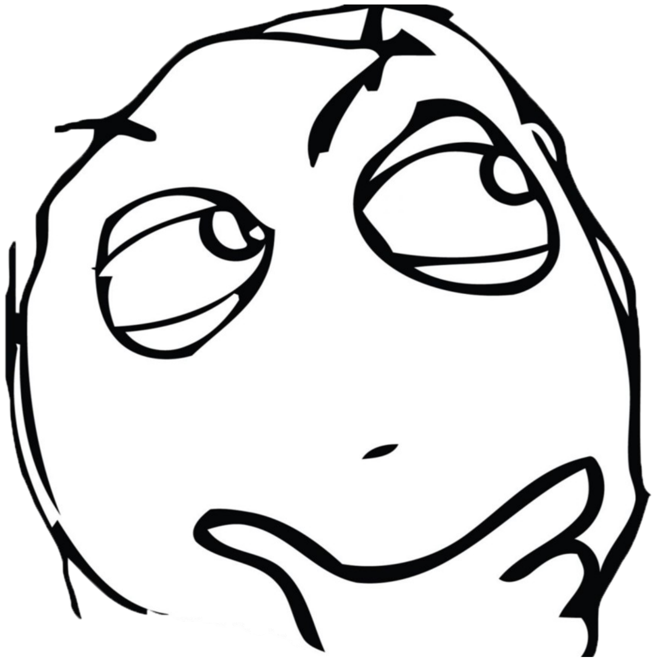
Memory leaks

Memory leaks is mostly a problem in case you repeatedly call a function which leaks, in which case the heap will soon run out of memory and the whole program will fail.

Dangling pointers.

A dangling pointer is a pointer that used to point at valid dynamic memory. After that memory got freed, the pointer still points at the same address, but it's now marked as available for the heap, so something else might get stored there at any time.

We need dynamic memory allocation



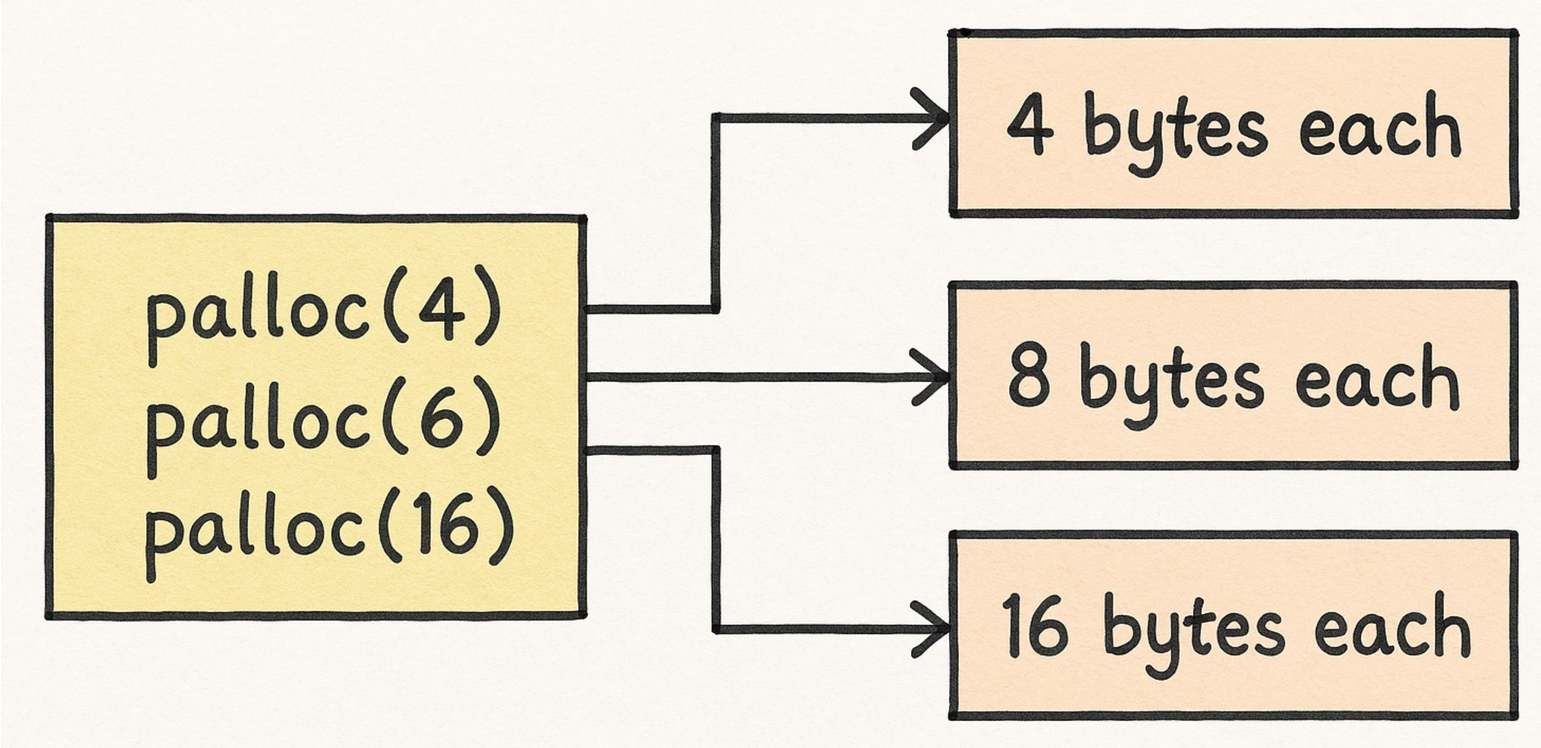
0x00 : Static memory preallocation

```c
#define SALLOC_BUFFER_SIZE 90000

static unsigned char GS_sallocBuffer[SALLOC_BUFFER_SIZE];
int GS_sallocFree = 0;

void *salloc(int size)
{
   void *nextBlock;
   assert(FS_enabled);
   if((GS_sallocFree + size) > SALLOC_BUFFER_SIZE)
   {
      assert(FALSE);
   }
   nextBlock = &GS_sallocBuffer[GS_sallocFree];
   GS_sallocFree += size;
   return nextBlock;
}
```

0x01 : Memory pools



Thank you

Twitter.com/sisoog

t.me/sisoog

zeus@sisoog.com

www.sisoog.com