CIS241

System-Level Programming and Utilities

C Dynamic Memory Allocation

Erik Fredericks, frederer@gvsu.edu

Fall 2025

> **Based on material provided by Erin Carrier, Austin Ferguson, and Katherine Bowers**

Memory allocation!

`malloc(size)`

- Allocates size bytes
- On the heap
- Contiguous

`void* malloc(size);`

- Returns a void pointer
- Think of it like a generic pointer
- Needs to be cast to the type you need
- Pointer points to start of allocated block

Example:

```
int* p = (int*)malloc(4 * 100);
```

#include

Note, `malloc` (and those like it), require a new include:

- `#include <stdlib.h>`

What if we don't know the size?

What if we don’t know the size of our type?

- Use `sizeof`!

```
long* p = (long*)malloc(sizeof(long) * 10);
```

When using `malloc` it may show a 0 when you print without initializing - this is either security or luck of the draw but regardless it is **NOT** guaranteed!

Alternatively...

```
calloc(count, size);
```

- Allocates `count * size` bytes
- Initializes values to zero (`malloc` does not)

Example:

```
float* p = calloc(64, sizeof(float));
```

What if we want more space?

Imagine we have an array of 16 ints.

- Now we need to double it to 32.
- But we don’t want to copy them by hand…
- `int* p = (int*)malloc(sizeof(int) * 16);`
- `p = realloc(p, sizeof(int) * 32);`

Allocates new memory!

- Copies the old over
- "Frees" old memory
- Not guaranteed to initialize new mem to zero

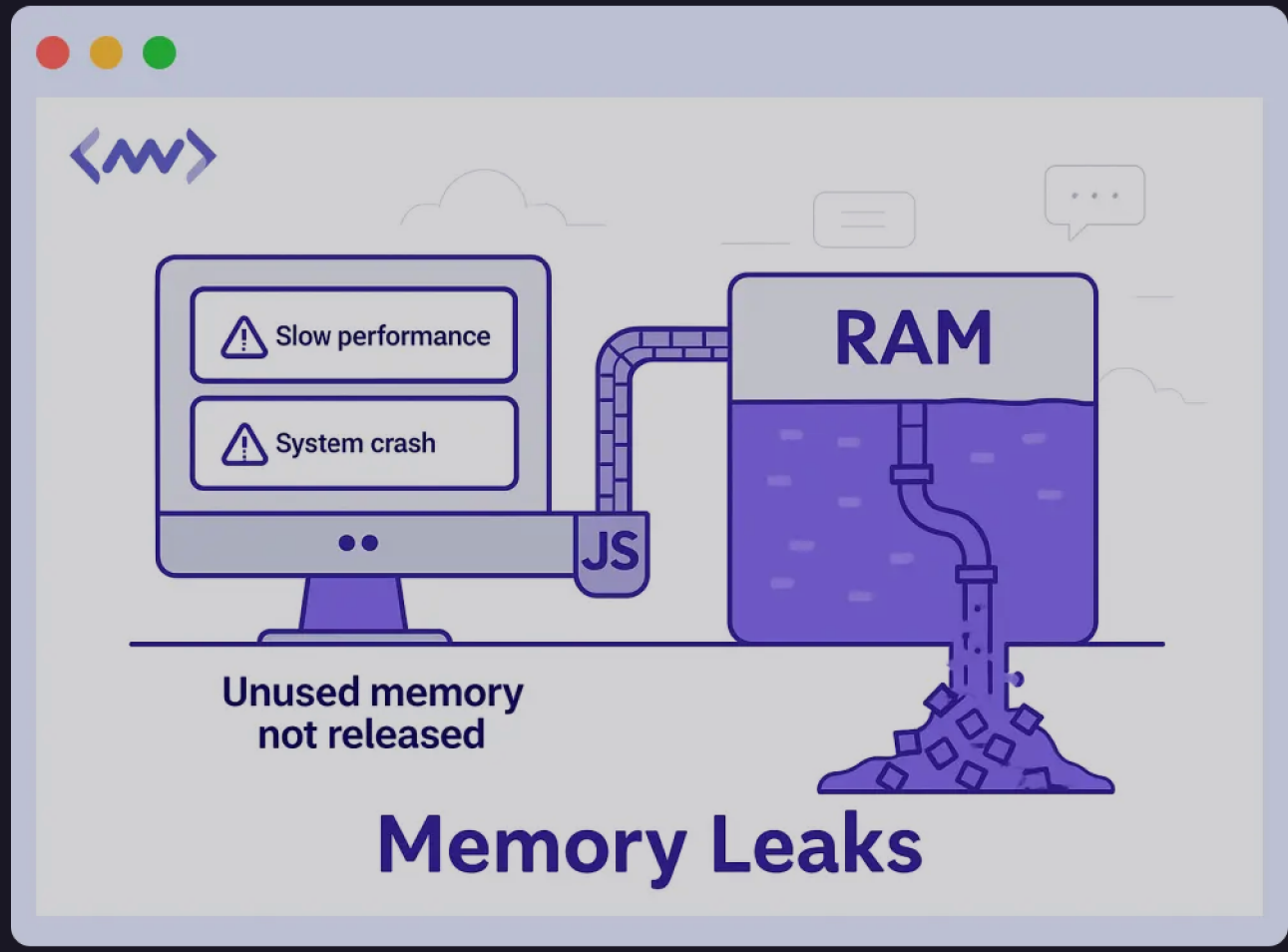
Old memory / memory leaks?

We need to free it when we're done!

- `free(pointer);`

If we forget, we can have memory leaks!

- Memory leak bad!



Example:

https://www.geeksforgeeks.org/c/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/