pictures I took recently

COMP1511/COMP1911
Week 8!

M13B: 1pm – 4pm || T11X: 11am – 2pm

Tutors: William (me!) + Jason || Daniel

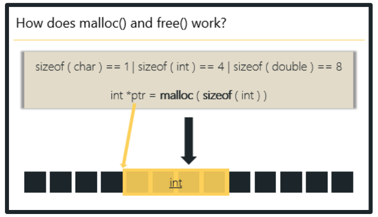
My GitHub:

https://github.com/william-o-s/unsw_comp1511_tutoring

Time to start Assignment 2!

Tutorial Agenda:

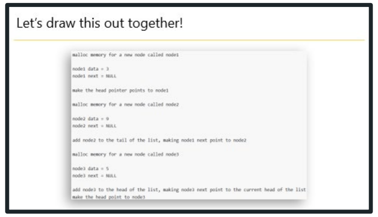
Part 1
Part 2
Part 3



How does malloc() and free() work?

sizeof (char) == 1 | sizeof (int) == 4 | sizeof (double) == 8

int *ptr = **malloc** (**sizeof** (int))



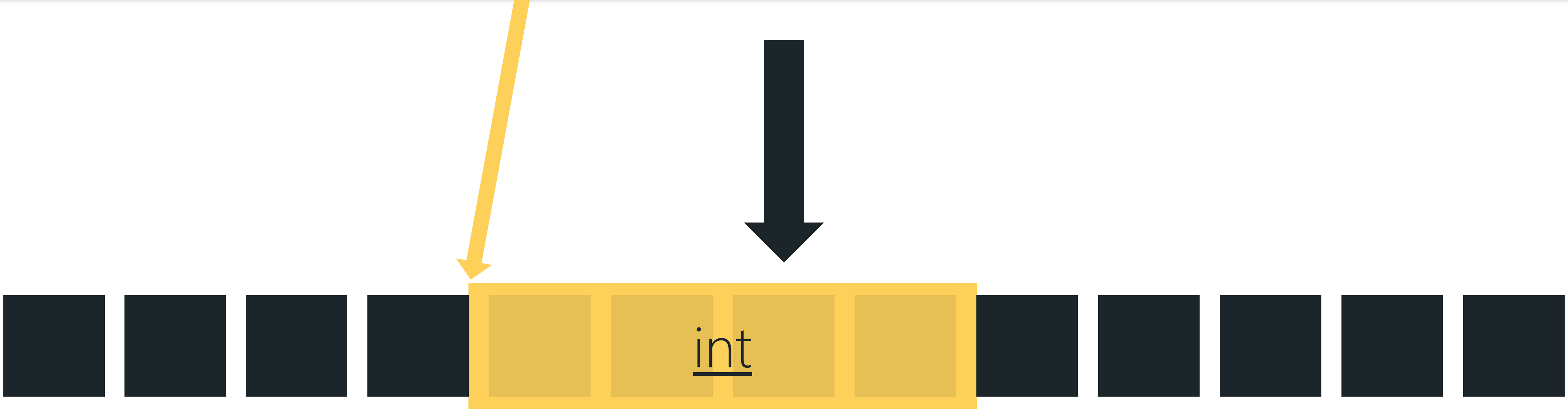
Let’s draw this out together!

```
malloc memory for a new node called node1

node1 data = 3
node1 next = NULL

make the head pointer points to node1

malloc memory for a new node called node2

node2 data = 9
node2 next = NULL

add node2 to the tail of the list, making node1 next point to node2

malloc memory for a new node called node3

node3 data = 5
node3 next = NULL

add node3 to the head of the list, making node3 next point to the current head of the list
make the head point to node3
```

How does malloc() and free() work?

sizeof (char) == 1 | sizeof (int) == 4 | sizeof (double) == 8

int *ptr = **malloc** (**sizeof** (int))



Using malloc() and free(), allocate the following

Integer:	int *ptr = malloc(sizeof (int))
Double:	double *ptr = malloc(sizeof(double))
Character:	char *ptr = malloc(sizeof(char))
Array of 10 characters:	char *ptr = malloc(10 * sizeof(char))
A struct called my_struct:	struct my_struct *ptr = malloc(sizeof(struct my_struct))

When to use what?

Stack

Advantages

- Speed:
  - Faster memory access due to stack frame pointer (look only at the stack compared to everywhere in heap)
- Deterministic:
  - Automatic deallocation when memory out-of-scope, more convenient for programmer

Disadvantages

Heap

Advantages

- Dynamic memory:
  - Adaptable to memory requirements (capped by total memory)
- Flexible memory:
  - Memory can be resized as needed (for arrays)
- Large memory space
  - Offers more memory compared to stack (depends on compile flags)

Disadvantages

When to use what?

Stack

Advantages

Disadvantages

- Limited size:
  - Bound by OS, risks stack overflow with excess memory allocation
- Static size:
  - Stack-allocated variables must be constant and known at compile-time (also why normal arrays can't adapt)

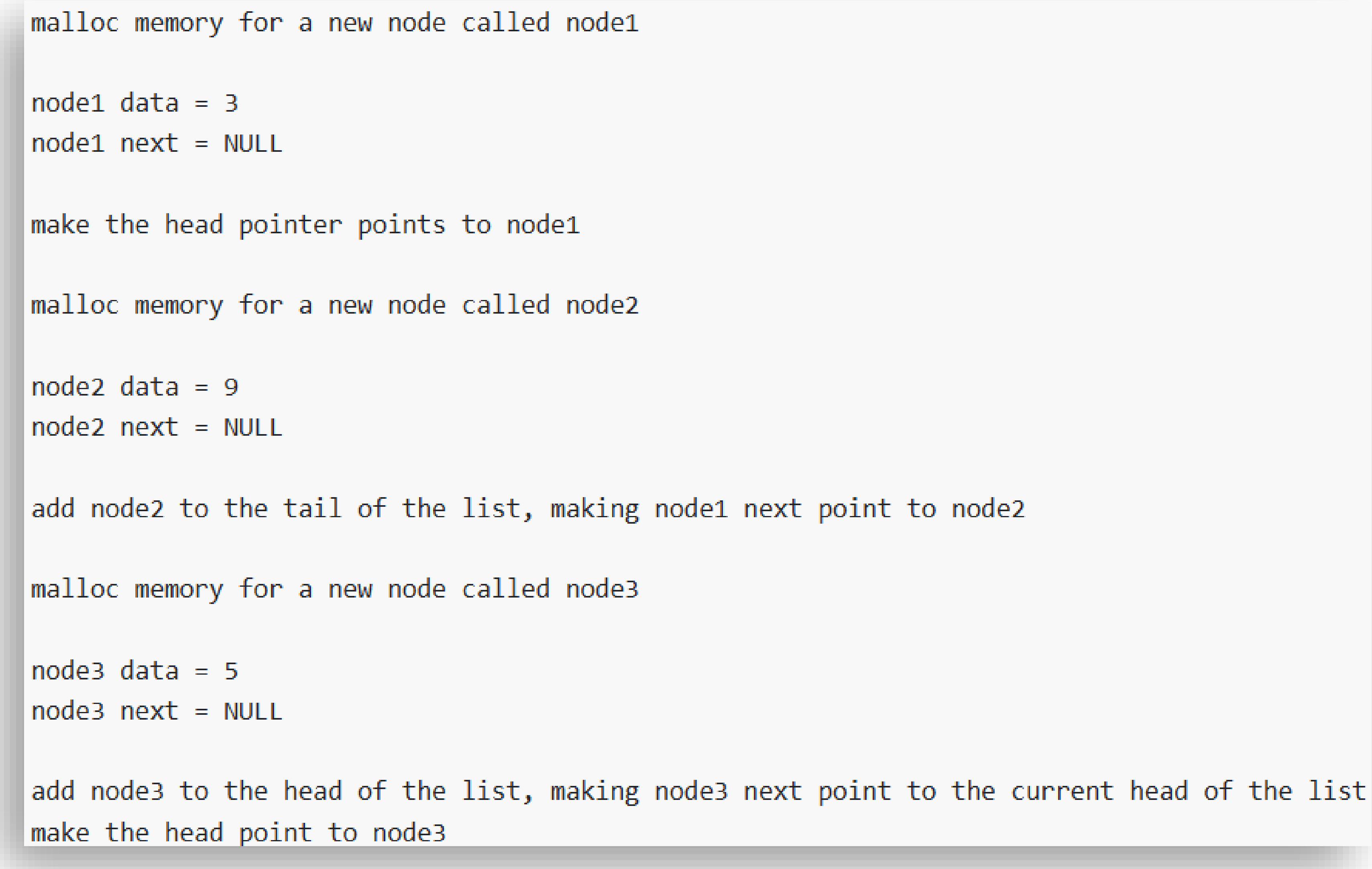
Heap

Advantages

Disadvantages

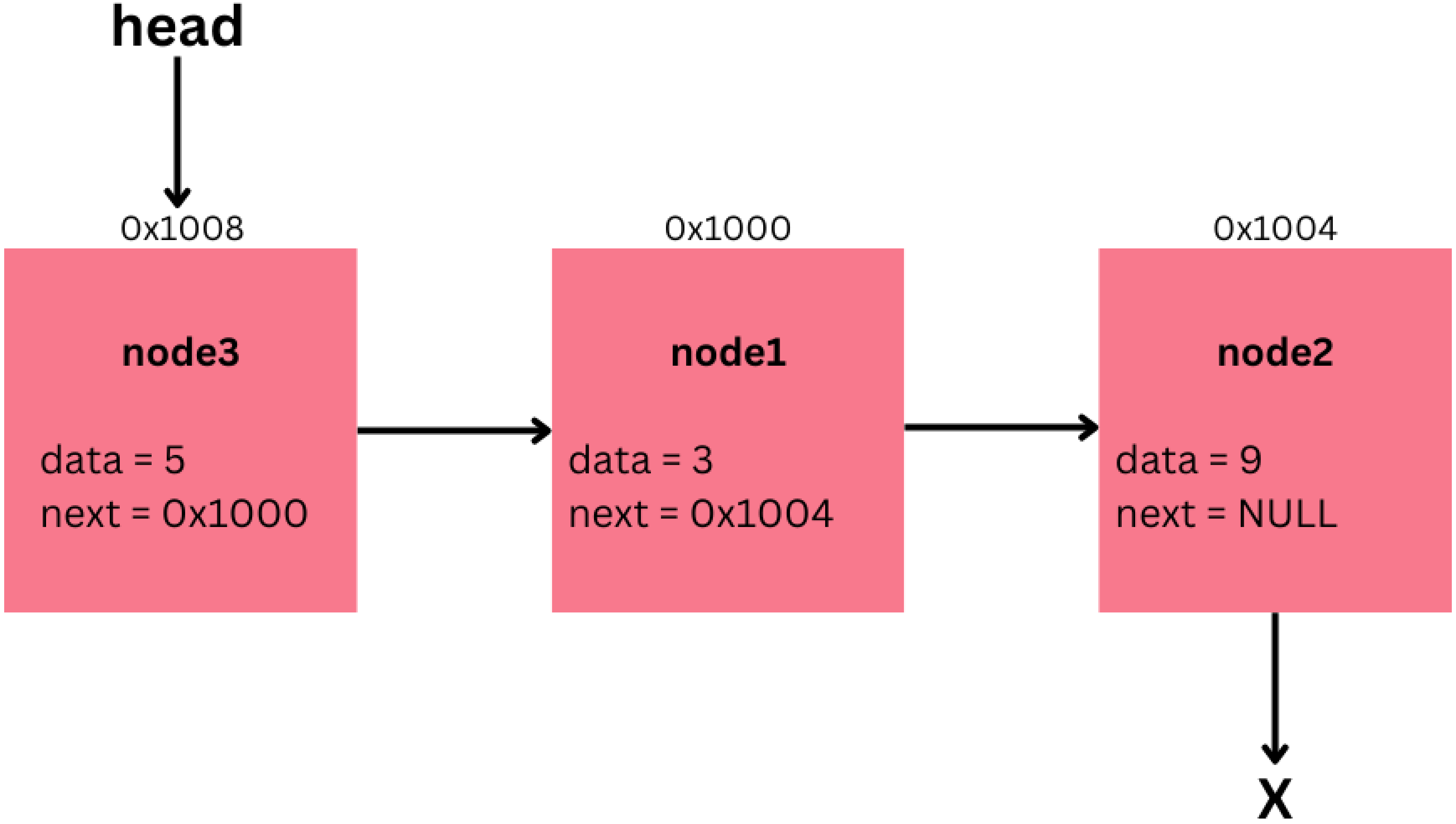
- Manual management:
  - Requires deallocation (using free) to prevent memory leaks
- Slower performance:
  - More overhead involved with memory management
- Fragmentation
  - Heap may become fragmented – imagine multiple small spaces vs. one large space

Let’s draw this out together!



```
malloc memory for a new node called node1

node1 data = 3
node1 next = NULL

make the head pointer points to node1

malloc memory for a new node called node2

node2 data = 9
node2 next = NULL

add node2 to the tail of the list, making node1 next point to node2

malloc memory for a new node called node3

node3 data = 5
node3 next = NULL

add node3 to the head of the list, making node3 next point to the current head of the list
make the head point to node3
```

Did we get something like this?



VSCode Shortcuts

- Start with Ctrl+Shift+P
  - "Toggle Multi-Cursor Editor"
  - Convert text casing: (highlight text) → Ctrl + Shift + P → "Transform to …"
- Multiple Cursors: Ctrl + Click anywhere
  - Cursor over multiple lines vertically: Shift + Alt + Click on line
- Duplicate Line: Ctrl + Shift + Alt + Up/Down Arrow
- Move Lines: Alt + Up/Down Arrow
- Change All Occurrences: Ctrl + Shift + L or Ctrl + D
- Indentation: (Highlight line/lines) → Ctrl + Left/Right Square Bracket
- Find and Replace: Ctrl + F → (click dropdown) → Replace next