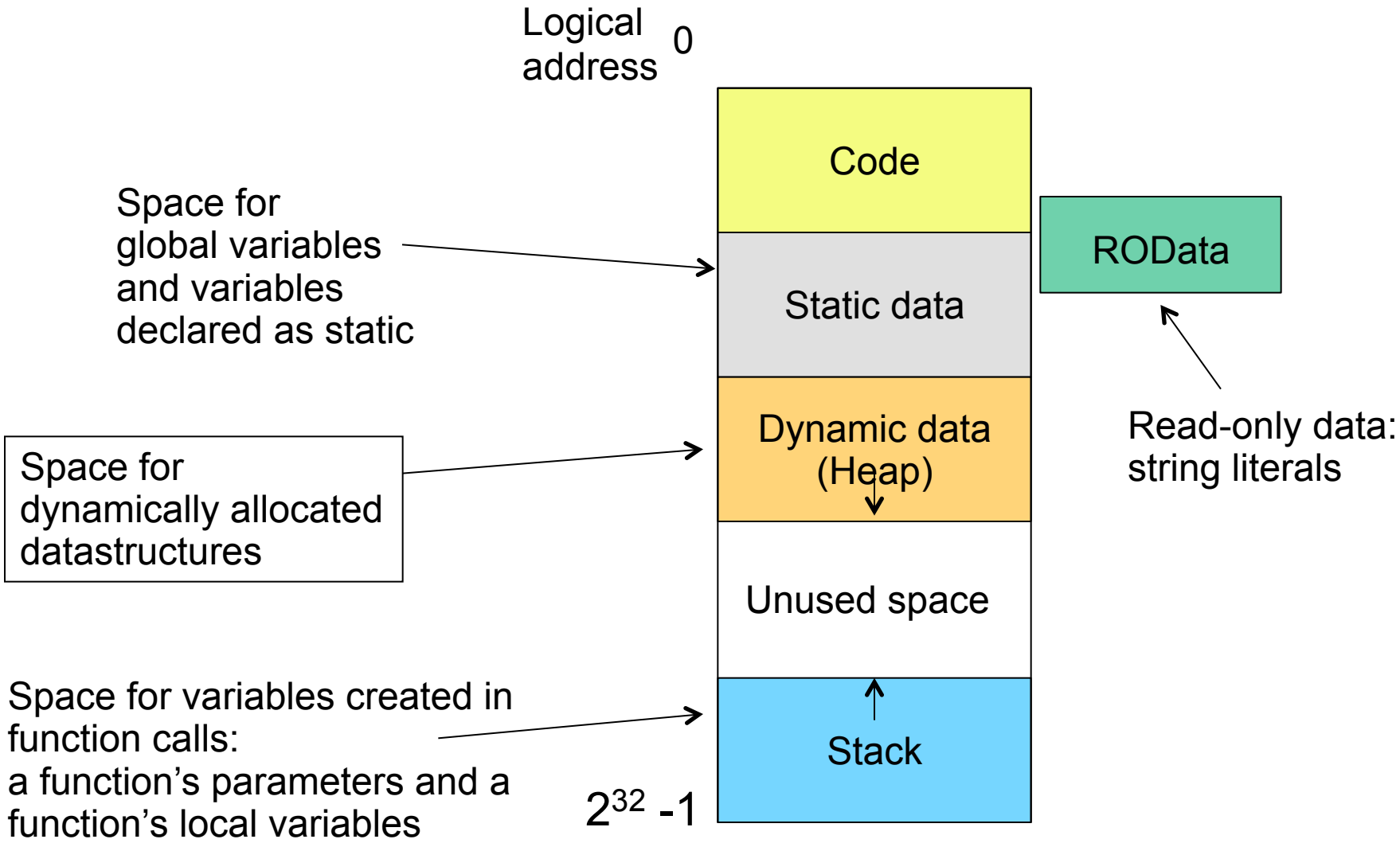
CSCB09: Software Tools and Systems Programming

Bianca Schroeder
bianca@cs.toronto.edu
IC 460

Recap..


Logical address
0
Code
Static data
Dynamic data (Heap)
Unused space
Stack
2^{32} -1
RODataa
Space for global variables and variables declared as static
Space for dynamically allocated datastructures
Space for variables created in function calls:
a function's parameters and a function's local variables
Read-only data: string literals

Memory can leak …

Memory can leak …

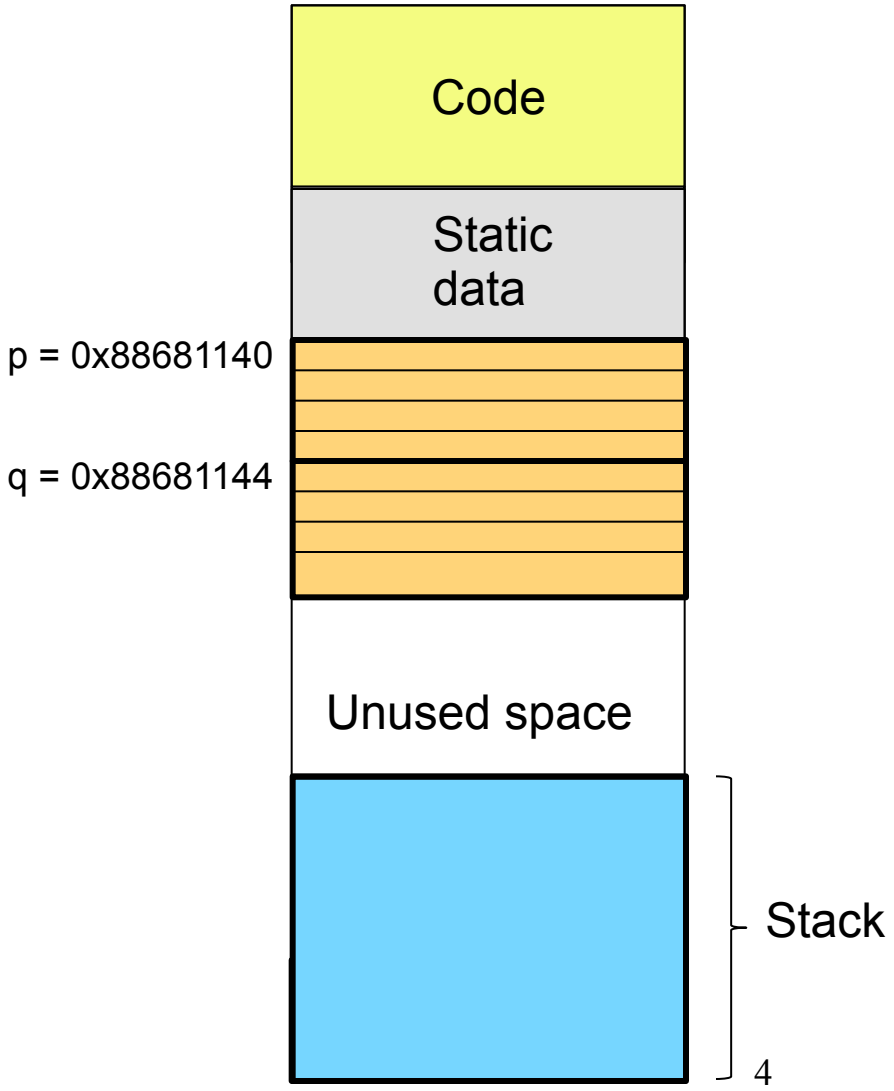
- Consider:

```
p = malloc(…);
q = malloc(…);
```

- Next consider:

```
p=q;
```

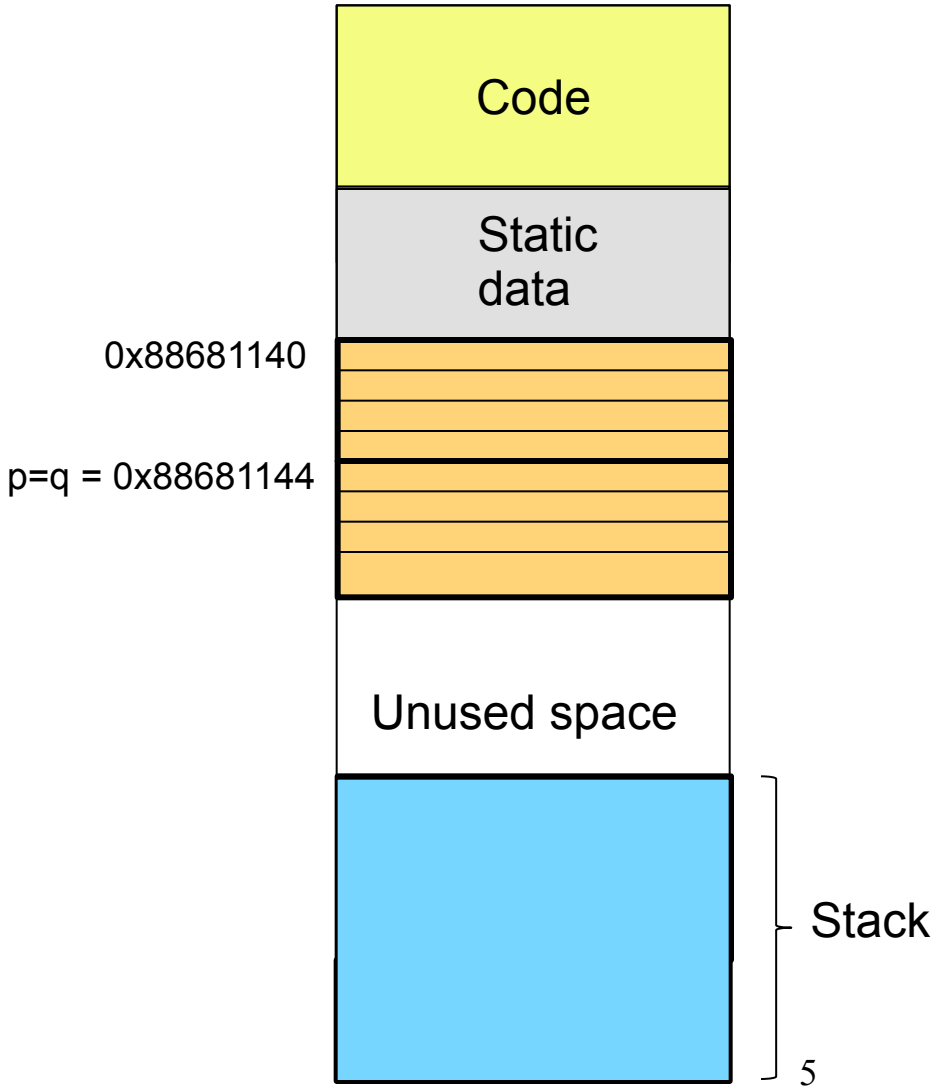
What happens?



Memory can leak …

- Problem we have no way of accessing p's old block.
- We also have no way of freeing p's old block.

- This is called a memory leak.
- One of the most common programming errors.



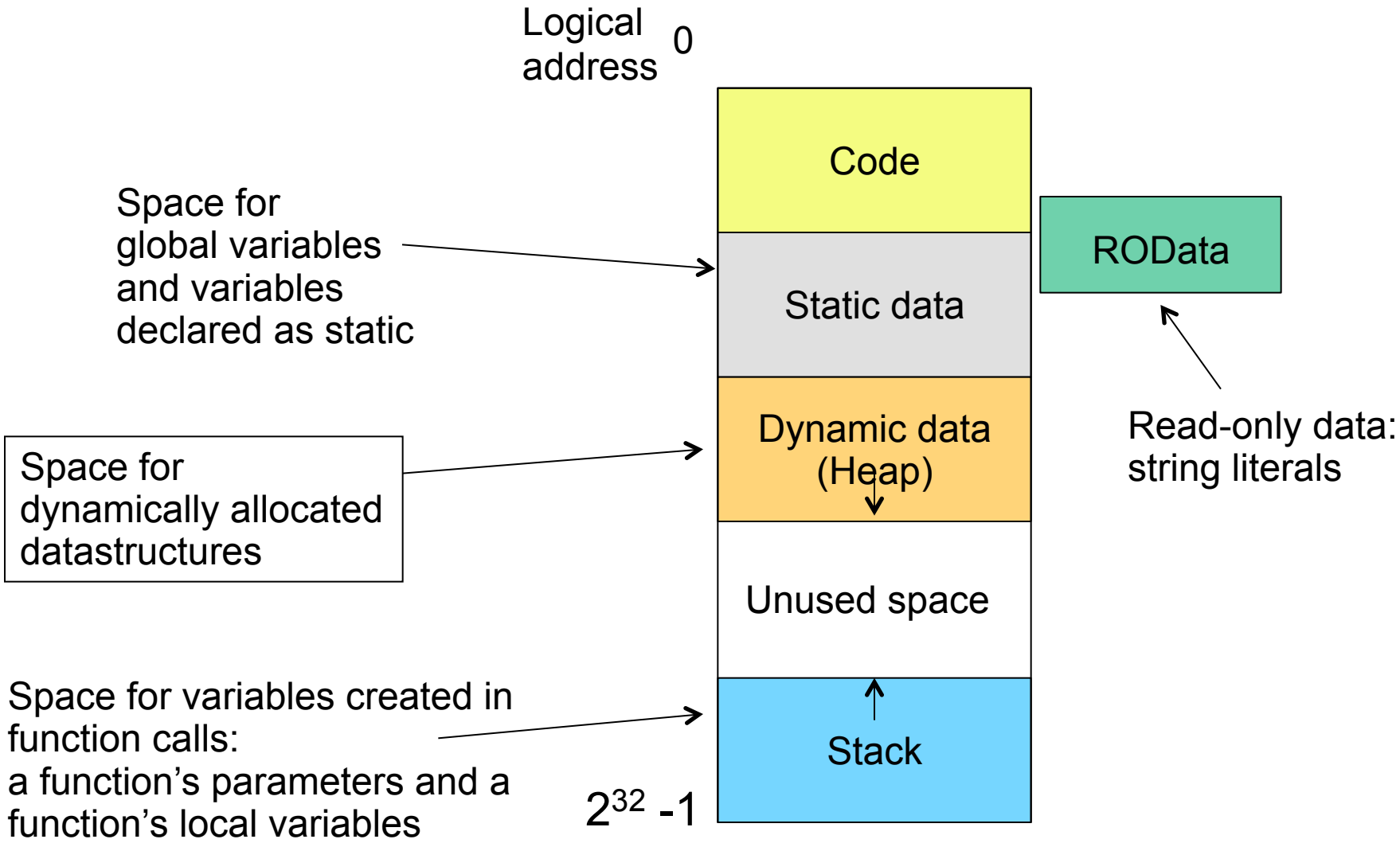
Dangling pointers

```
char *p = malloc(5);
…
free (p)
```

- p is now a pointer to memory it does not own.

```
strcpy(p,"abcd");   // Bad things can happen
```

Recap..


Logical address
0
Code
Static data
Dynamic data (Heap)
Unused space
Stack
2^{32} -1
RODataa
Space for global variables and variables declared as static
Space for dynamically allocated datastructures
Space for variables created in function calls:
a function's parameters and a function's local variables
Read-only data: string literals

(I)

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  int i;`
`}`
	sizeof(int)	stack	when program ends

`int fun() {`
`  float i;`
`}`
`int main() {`
`  fun();`
`}`			

`int fun(char i) {`
 	
`  ...`
`}`
`int main() {`
`  fun('a');`
`}`			

(I)

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  int i;`
`}`
	sizeof(int)	stack	when program ends

`int fun() {`
`  float i;`
`}`
`int main() {`
`  fun();`
`}`
	sizeof(float)	stack	when fun returns

`int fun(char i) {`
 	
`  ...`
`}`
`int main() {`
`  fun('a');`
`}`
	1 == sizeof(char)	stack	when fun returns

(II)

```
int main() {
    char i[10] = "hello";

}
```

```
int main() {
    char *i;
}
```

```
int main() {
    int *i;
}
```

```
int main() {
    char *i = "hello";


}
```

(II)

`int main() {`
`  char i[10] = "hello";`
 	
`}`			

`int main() {`
`  char *i;`
`}`
	sizeof(char *)	stack	when program ends

`int main() {`
`  int *i;`
`}`
	sizeof(int *)	stack	when program ends

`int main() {`
`  char *i = "hello";`
 	
`}`			

(II)

`int main() {`
`  char i[10] = "hello";`
 	
`}`
	10 == 10*sizeof(char)	stack	when program ends

`int main() {`
`  char *i;`
`}`
	sizeof(char *)	stack	when program ends

`int main() {`
`  int *i;`
`}`
	sizeof(int *)	stack	when program ends

`int main() {`
`  char *i = "hello";`
 	
`}`
	sizeof(char *)
	6 == 6 * (sizeof(char)
		stack
		read-only memory
			when program ends
			when program ends

(III)

```
int fun(int *i) {
    ...
}
int main() {
    int i[5] = {4,5,2,5,1};
    fun(i);
}
```

(III)

`int fun(int *i) {`
`  ...`
`}`
`int main() {`
`  int i[5] = {4,5,2,5,1};`
`  fun(i);`
`}`
	sizeof(int *)
	
	5 * sizeof(int)
		stack
		
		stack
			when fun returns
			
			when program ends

(IV)

```
int main() {
  int *i;
  i = malloc(sizeof(int));

}
```

(IV)

`int main() {`
`  int *i;`
`  i = malloc(sizeof(int));`
 	
`}`
	sizeof(int *)
	sizeof(int)
		stack
		heap
			when program ends
			when free called or when program ends

(V)

```
void fun(int **i) {
  *i = malloc(sizeof(int)*7);

}
int main() {
    int *i;
    fun(&i);
    free(i);
}
```

(V)

`void fun(int **i) {`
`  *i = malloc(sizeof(int)*7);`
 	
`}`
`int main() {`
`  int *i;`
`  fun(&i);`
`  free(i);`
`}`
	sizeof(int **)
	7 * sizeof(int)
	
	sizeof(int *)
		stack
		heap
		
		stack
			when function returns
			when free called or prog ends
			
			when program ends

And off to the second page of the worksheet…

Next up on the worksheet …

- Write a program that declares 3 strings
  - “Monday” on the stack,
  - “Tuesday” string literal,
  - “Wednesday” on the heap.
- Shorten the strings to the abbreviations of the weekday names, e.g. “Monday” becomes “Mon”.
- Add an array, where each element points to one of the strings above.
- Draw the memory model.