# Structs and the Heap

Grouping data and dynamic memory

**Paul Solt**
iPhoneDev.tv

# Outline

- `struct`
- The Heap
- `malloc()`
- `free()`

# struct

- Custom type
- Group related data
- Share between functions

## struct

```
 struct Point {
     int x;
     int y;
};
```

```
// main()
struct Point start;
start.x = 500;
start.y = 40;
printf("(%d, %d)",
    start.x, start.y);
```

Paul Solt
iPhoneDev.tv

# struct

```c
 struct Point {
     int x;
     int y;
};
```

```c
// main()
struct Point start;
start.x = 500;
start.y = 40;
printf("(%d, %d)",
    start.x, start.y);
```

```
(500, 40)
```

Paul Solt
iPhoneDev.tv

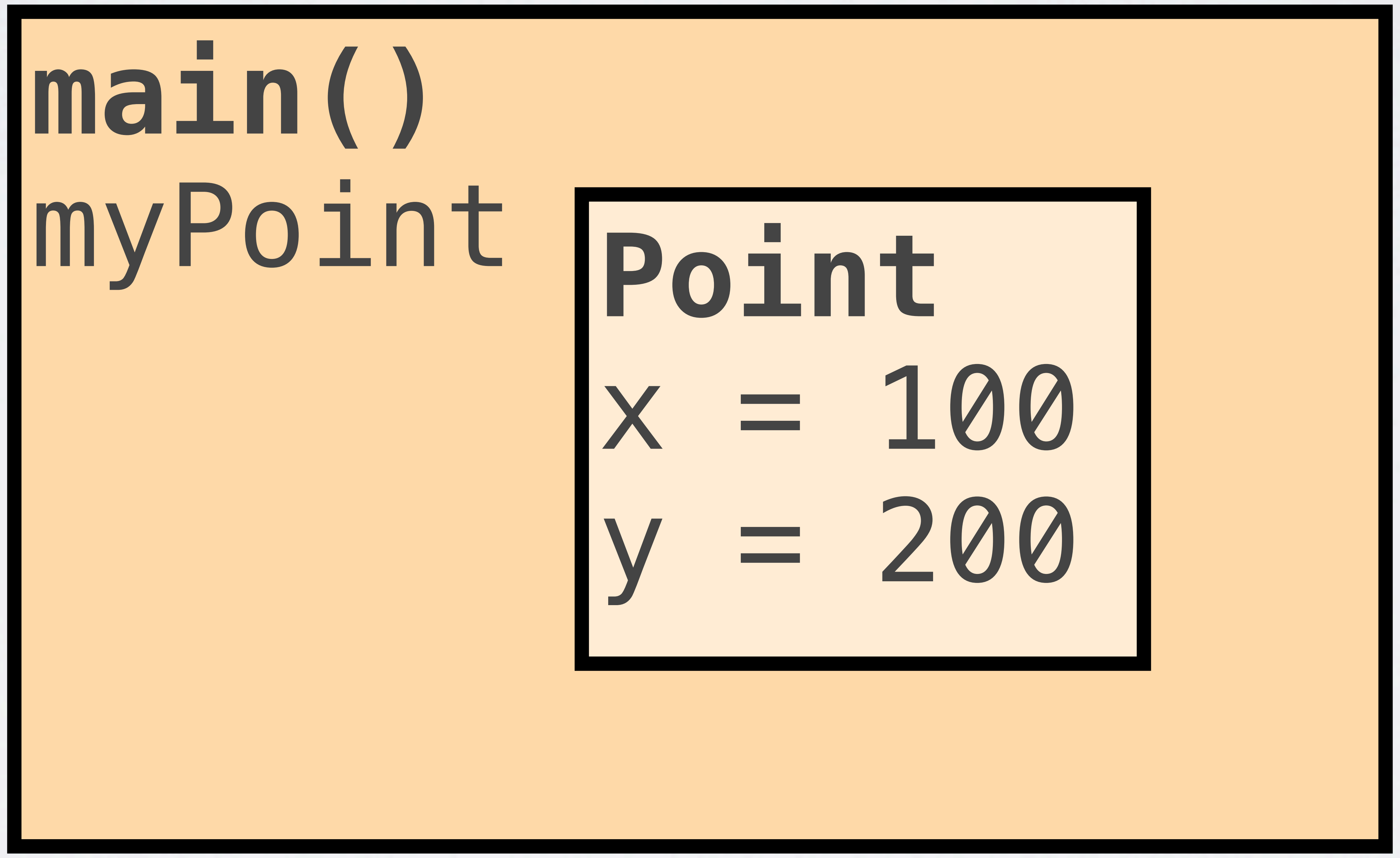
# typedef

```
 struct Point {
     int x;
     int y;
};

// main()
struct Point start;
```

```
typedef struct {
    int x;
    int y;
} Point;

// main()
Point start;
```

## Stack Frame

## Stack Frame

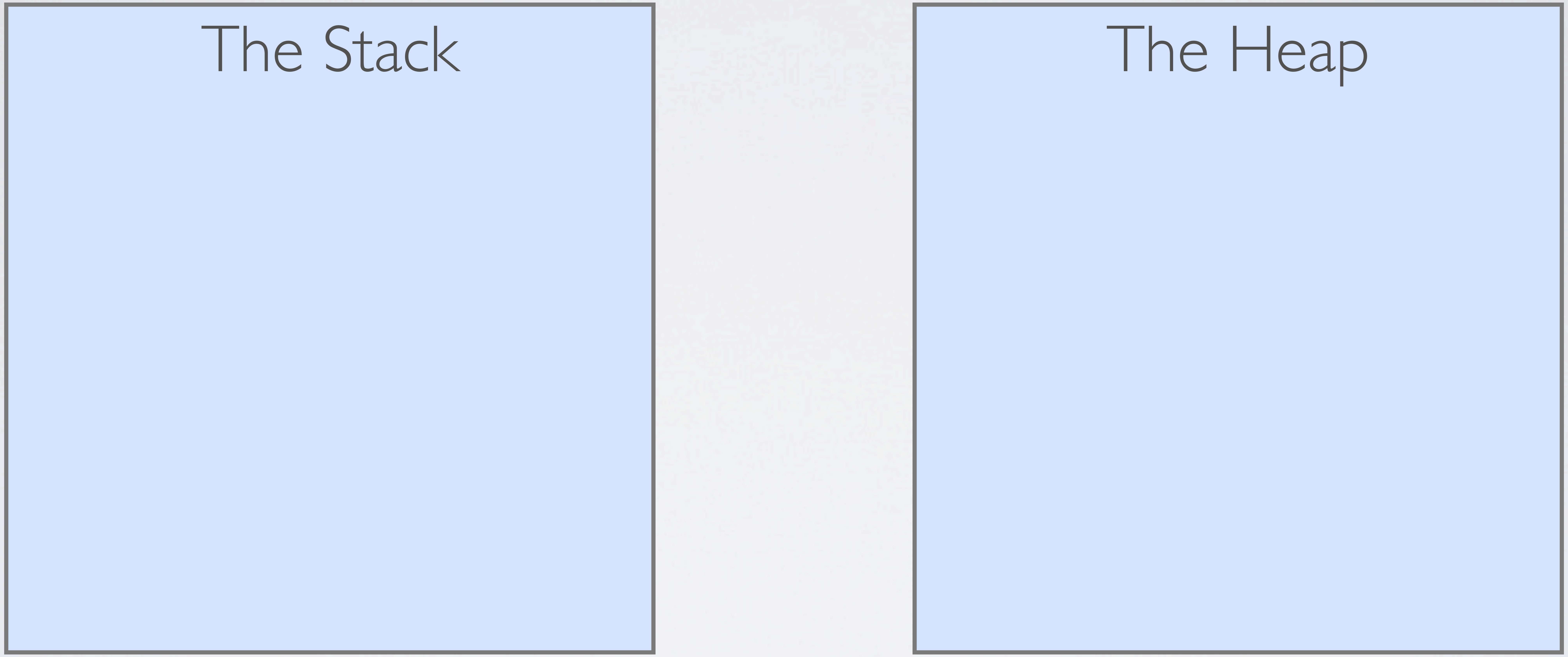


Paul Solt
iPhoneDev.tv

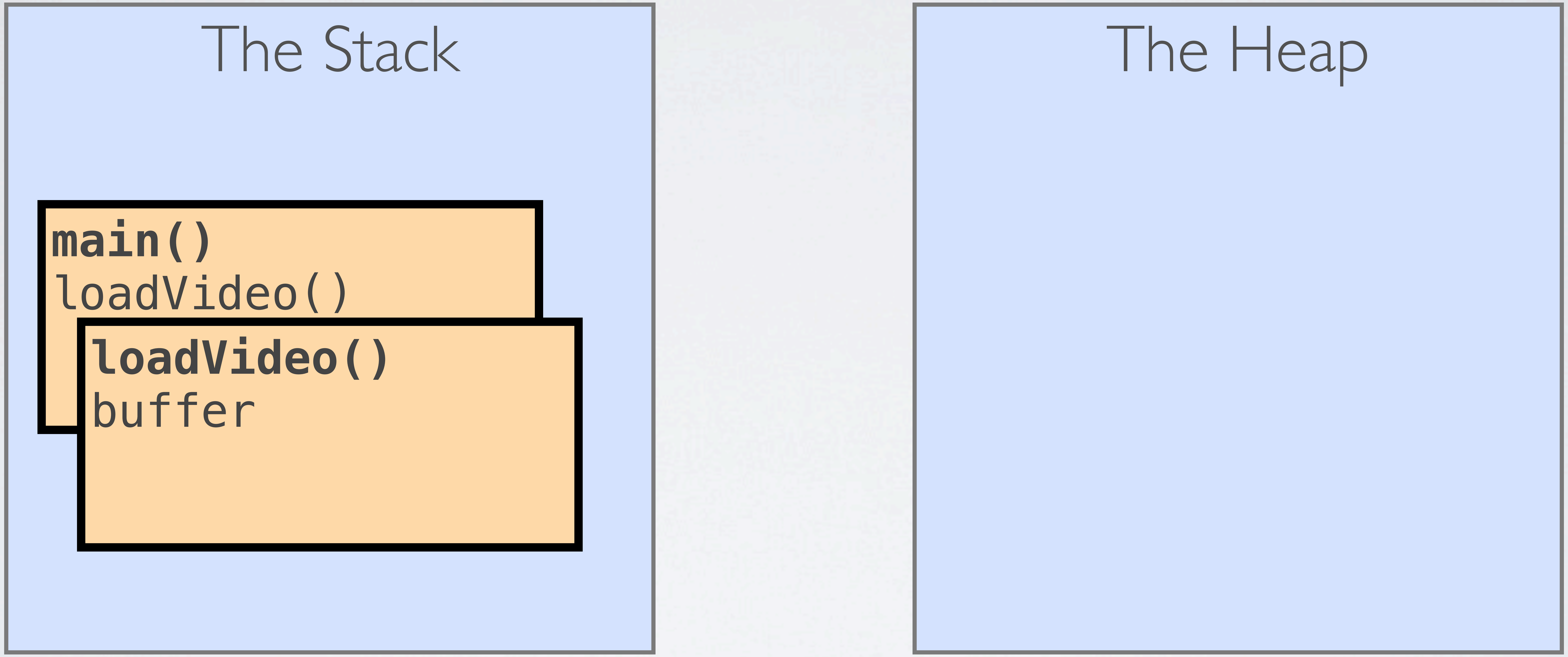
## The Heap

- Dynamic memory
- Separate from Stack

# Memory

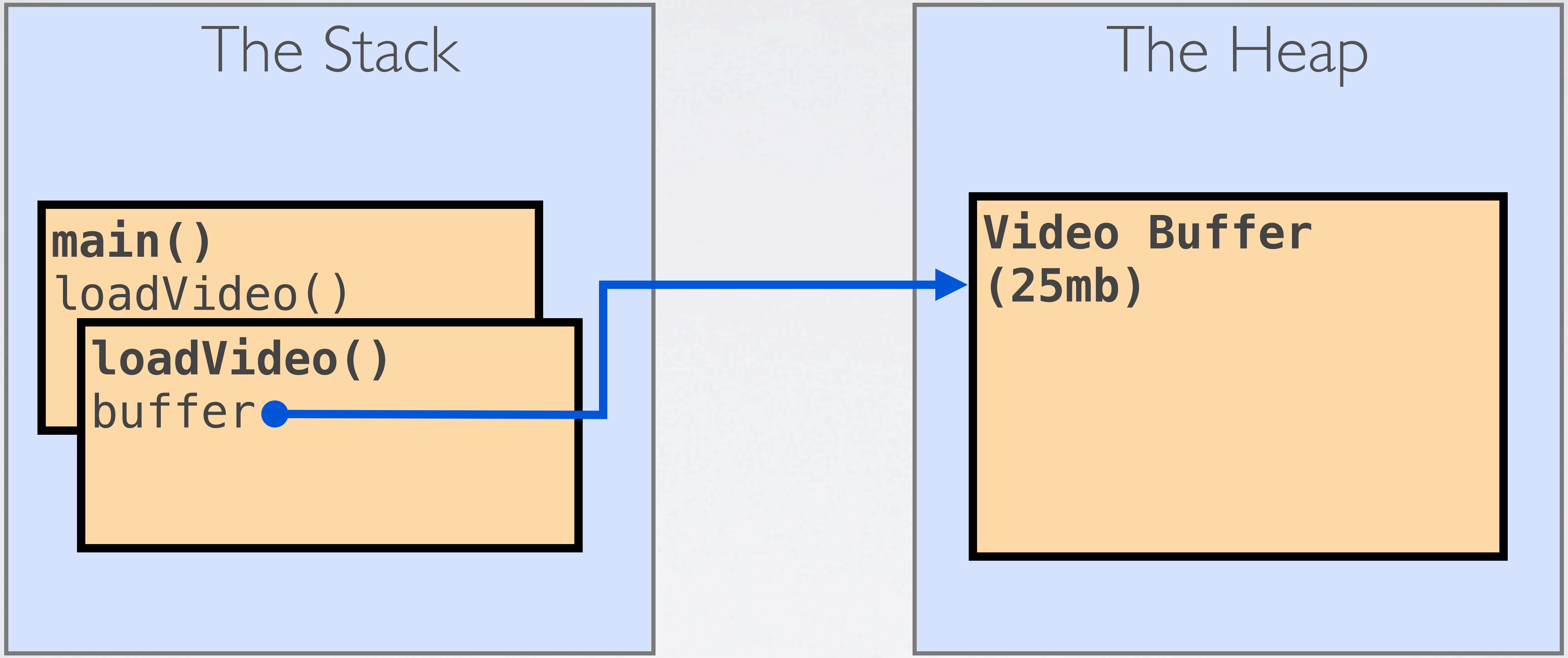


# Memory



Paul Solt
iPhoneDev.tv

# Memory



Paul Solt
iPhoneDev.tv

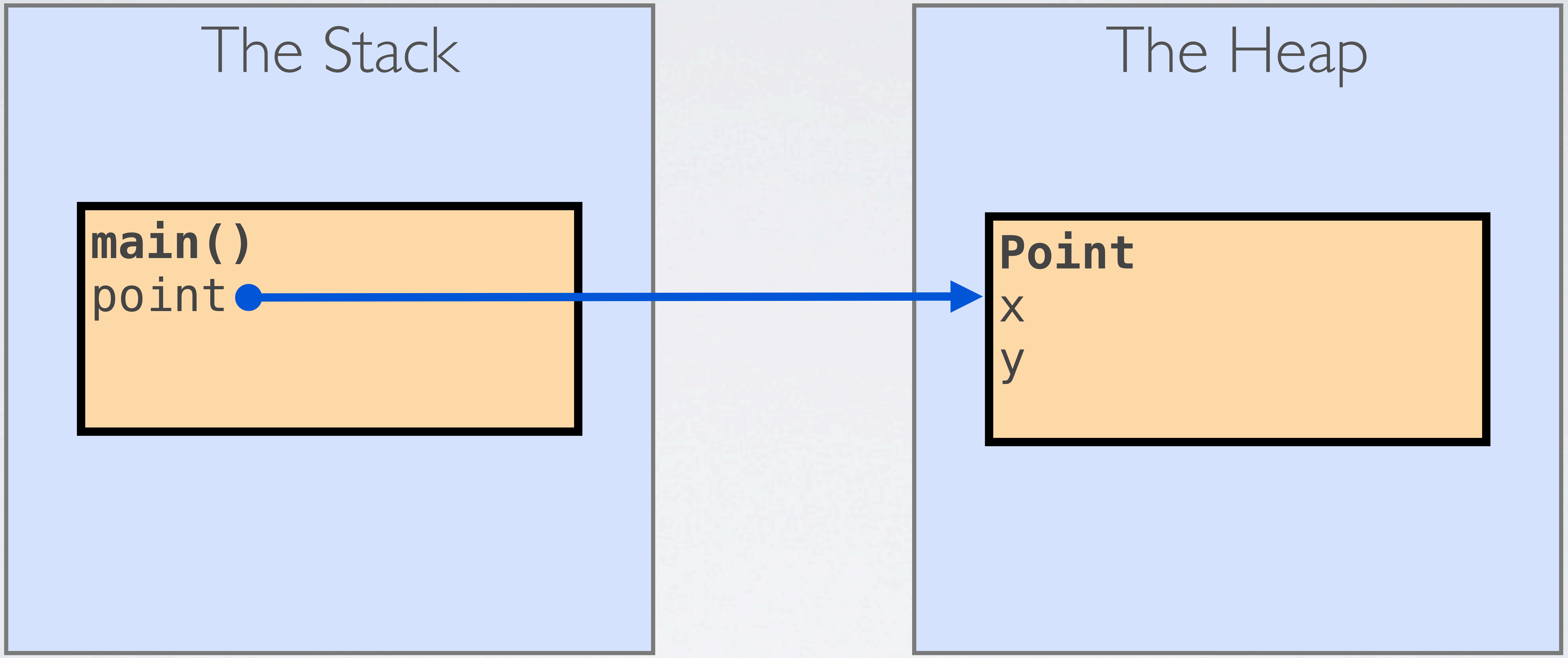
# `malloc()`

- “Check-in”
- Request memory block
    - **`sizeof()`**
    - Returns pointer address

## free()

- “Check-out”
- Release memory block
  - CPU cleans up
  - Can be “rented” again

```c
 // Create a buffer for text
char *textBuffer = malloc(2000 * sizeof(char));

// Use buffer

// Cleanup
free(textBuffer);

// Clear memory address
textBuffer = NULL;
```


The Stack
main()
point
The Heap
Point
x
y

# Review

- `struct`
- The Heap
- `malloc()`
- `free()`

**Paul Solt**
iPhoneDev.tv