Pointers & Memory

Memory Allocation

- Types of memory allocation in C++
    - Automatic memory allocation
    - Dynamic memory allocation
    - Static memory allocation

Memory Allocation

- Automatic memory allocation is an automated process of allocating and freeing memory that affects any data values of variable that are know at compile time

- Dynamic memory allocation is a dynamic process of allocating and freeing of memory that affects any data values that can only be known at run-time

Automatic Memory Management

- The Stack manages automatic memory allocation for local variables
- Memory blocks are pushed to the stack during allocation and popped form the stack during de-allocation
- Memory blocks are stored in a contiguous order

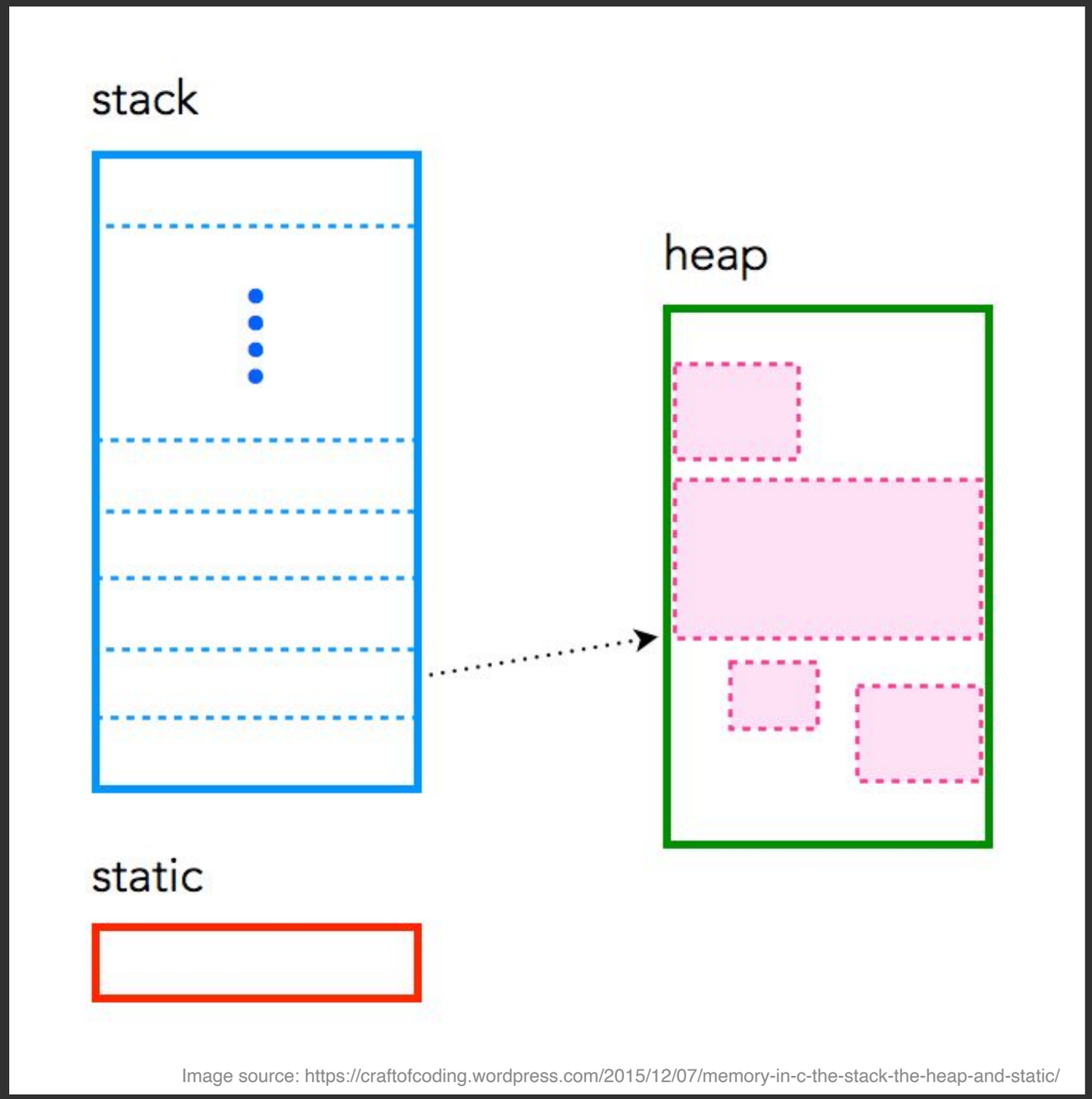


Image source: https://craftofcoding.wordpress.com/2015/12/07/memory-in-c-the-stack-the-heap-and-static/

Dynamic Memory Management

- The Heap or “free-store” manages dynamic memory allocation in combination with the stack for any local variables

- The heap comprises a large pool of memory and stores memory where suitable

- Memory blocks are NOT stored in a contiguous order

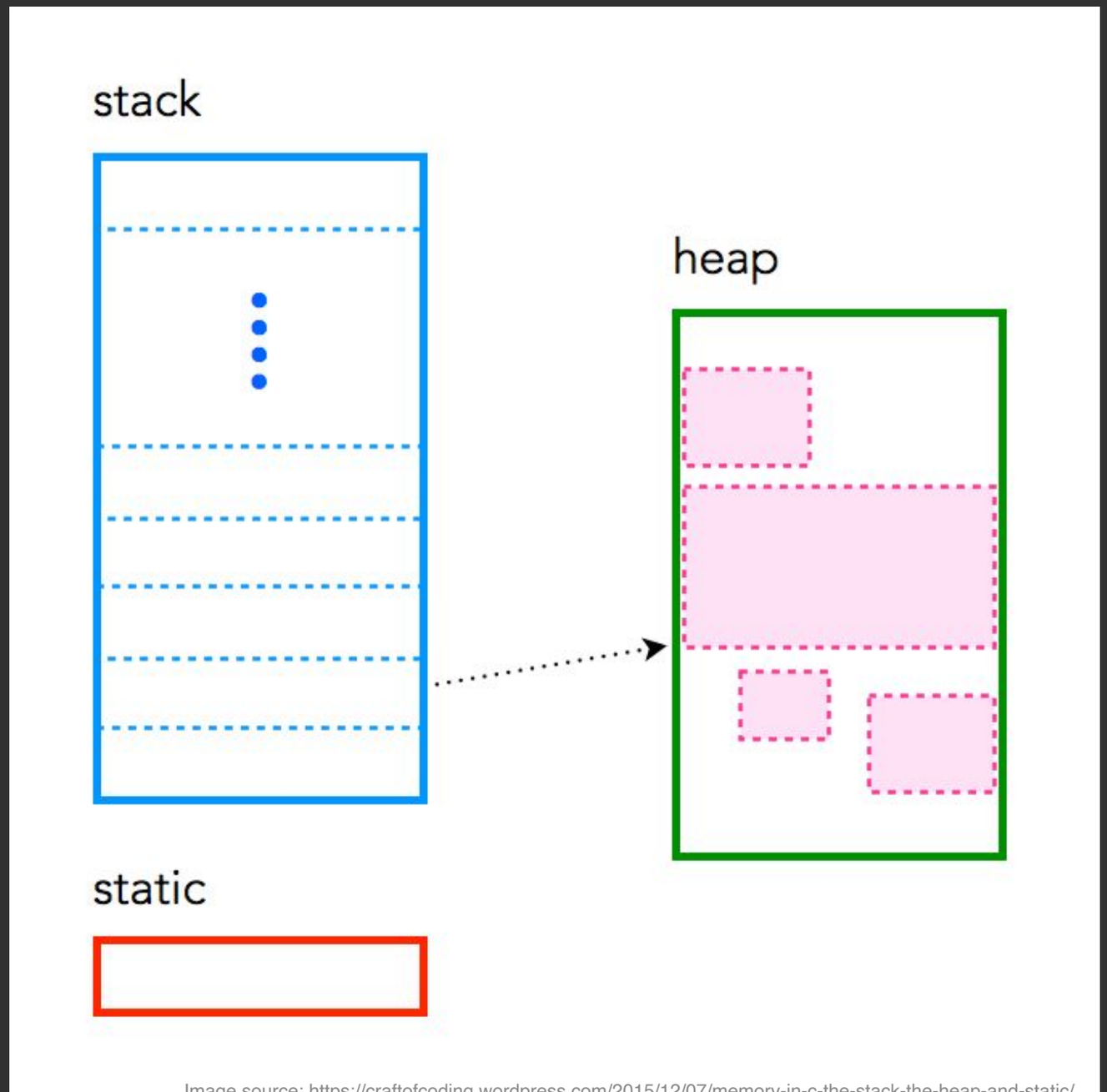


Image source: https://craftofcoding.wordpress.com/2015/12/07/memory-in-c-the-stack-the-heap-and-static/

Static Memory Management

- The Static manages any global variables
- This memory blocks are allocated permanently for the entire run of the program
- *[References: https://craftofcoding.wordpress.com]*

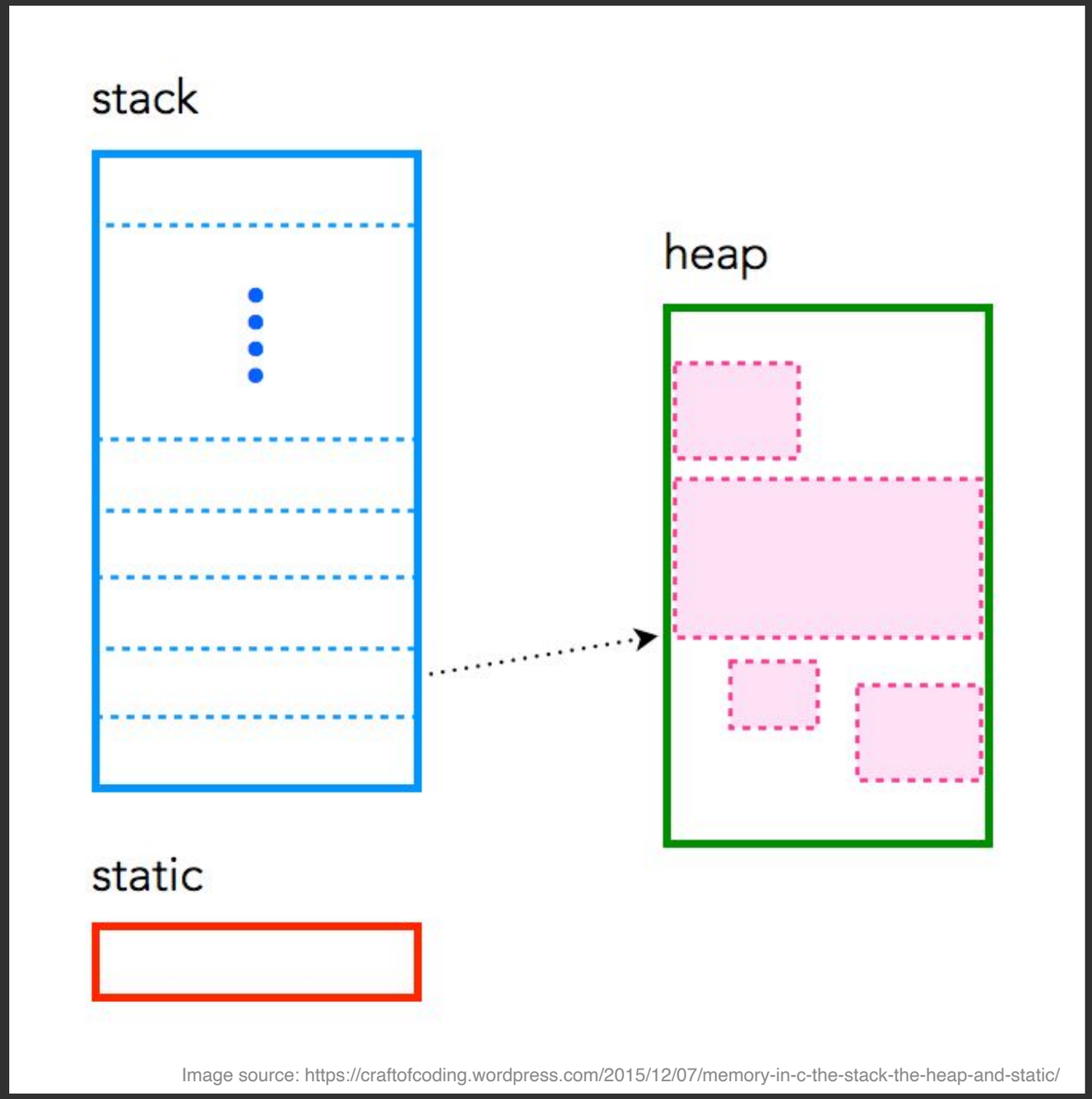


Image source: https://craftofcoding.wordpress.com/2015/12/07/memory-in-c-the-stack-the-heap-and-static/

Pointers & Dynamic Memory Allocation

- Pointers are the tools provided by C++ to manually initiate the dynamic allocation, use, and de-allocation during run-time

- Dynamic memory allocation requires manual memory management

- The developer has to take care of allocating AND freeing the memory with new & delete — no automated process

- This makes dynamic memory allocation error-prone

Pointers & Dynamic Memory Allocation

- To allocate memory dynamically in C++, specific keywords are used to request the memory
  - new & delete
  - new & delete[ ] — for C-arrays | use std::vector instead!

- … and the allocated memory is accessed via a pointer

```
myApp* myPtr = new myApp( ); // allocate memory & assign
myPtr->setValue( … );        // do something
delete myPtr;                // free the memory
```

Pointers & Dynamic Memory Allocation

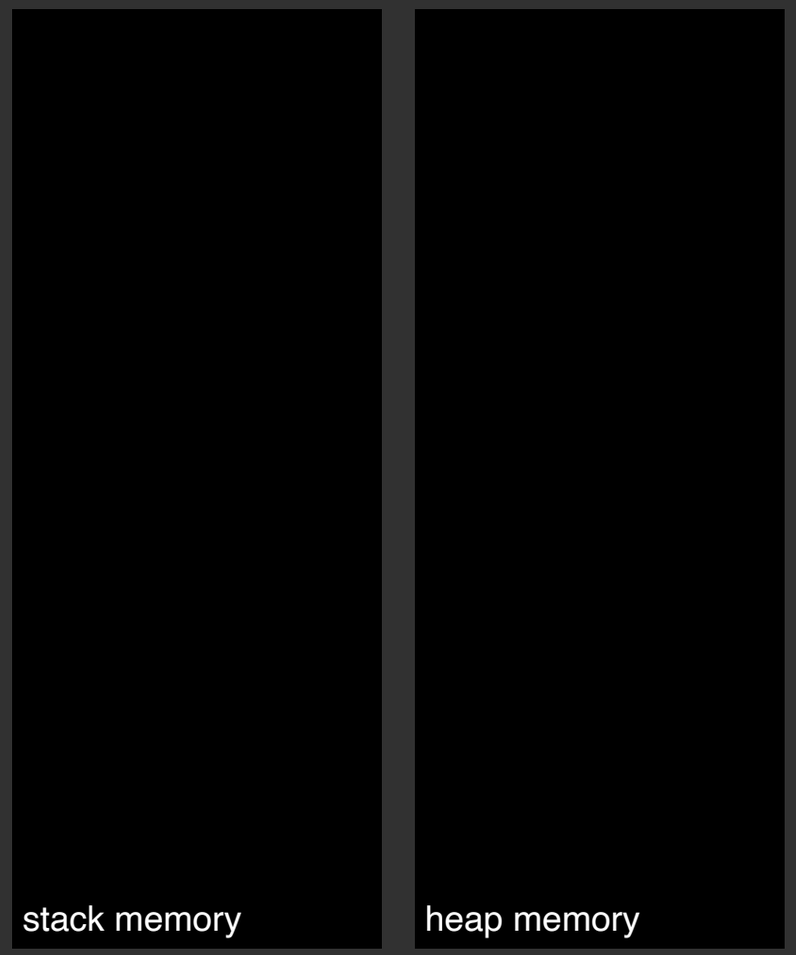
```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };      // allocating memory dynamically with new
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();    // the derived classes inherit the
21
22      delete aPtrToFruit;    // freeing or deallocating the memory with delete
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

Pointers & Dynamic Memory Allocation

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | line 15

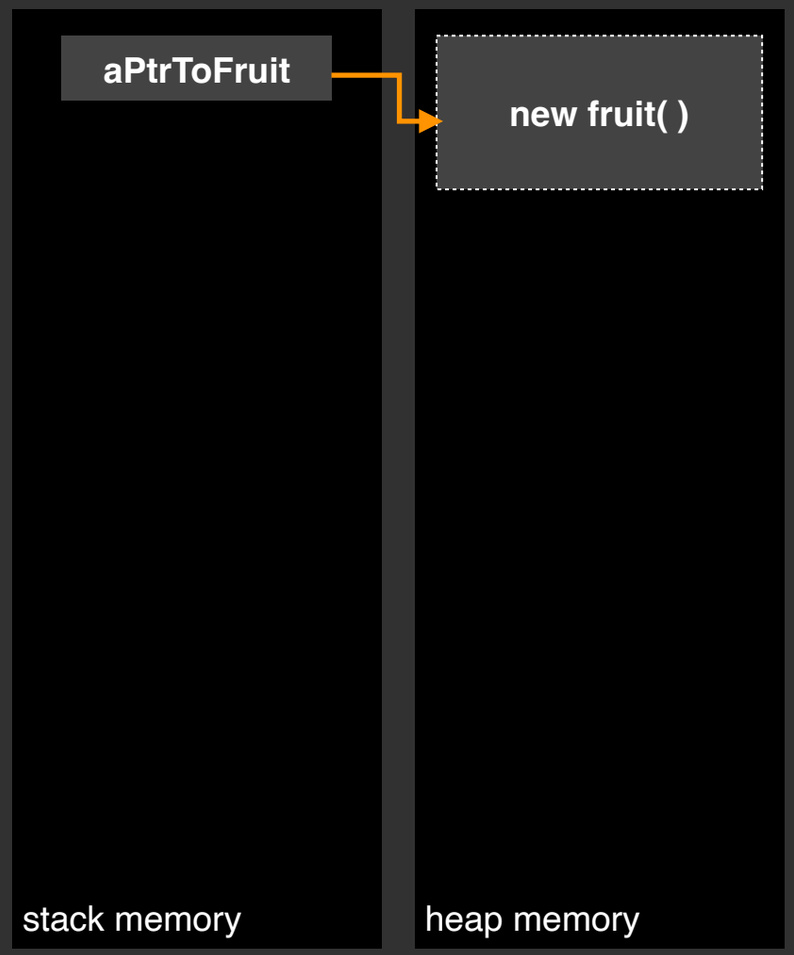


Pointers & Dynamic Memory Allocation

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | line 16

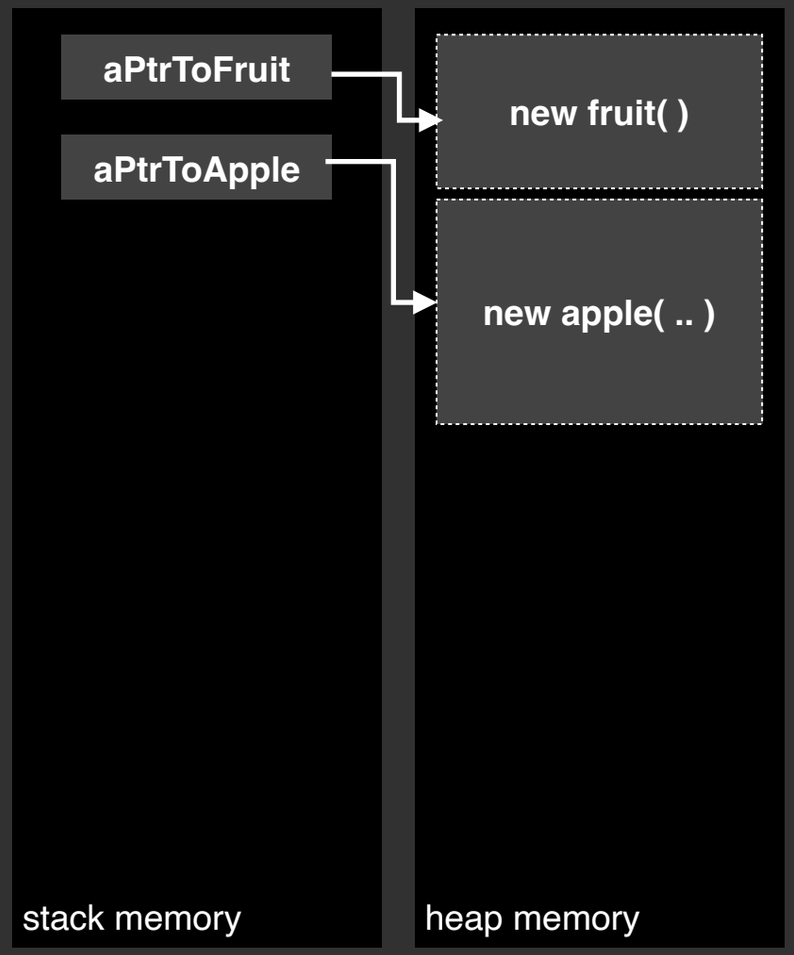


Pointers & Dynamic Memory Allocation

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | line 17

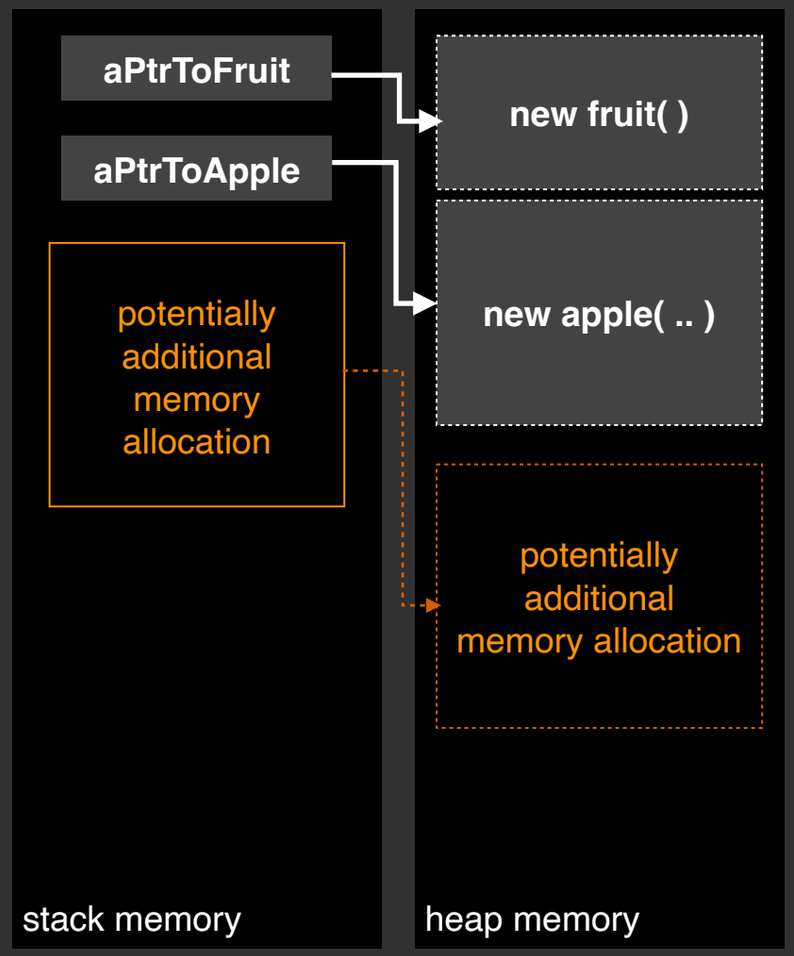


Pointers & Dynamic Memory Allocation

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | lines 19, 20

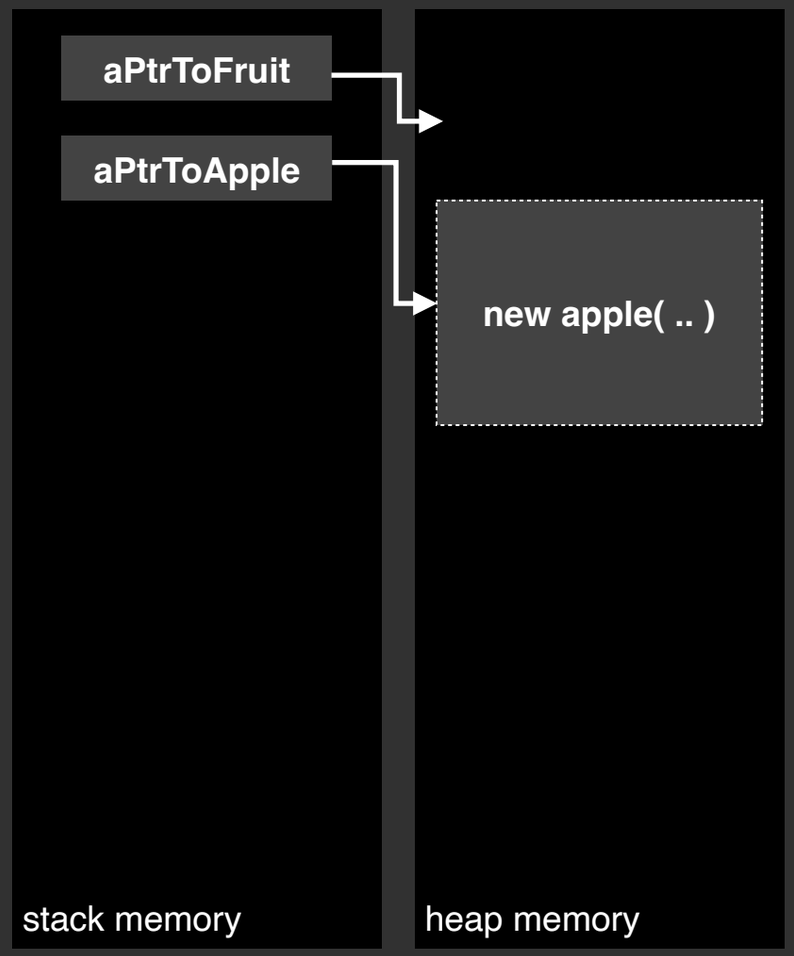


Pointers & Dynamic Memory Allocation

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | line 22

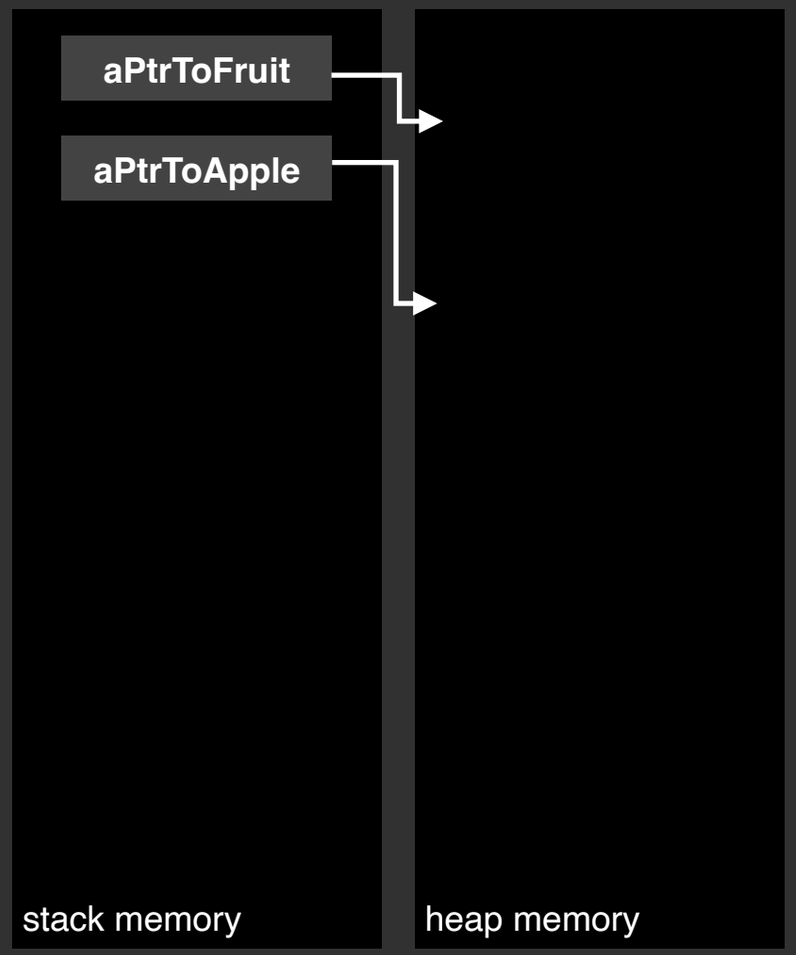


Pointers & Dynamic Memory Allocation

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | line 22

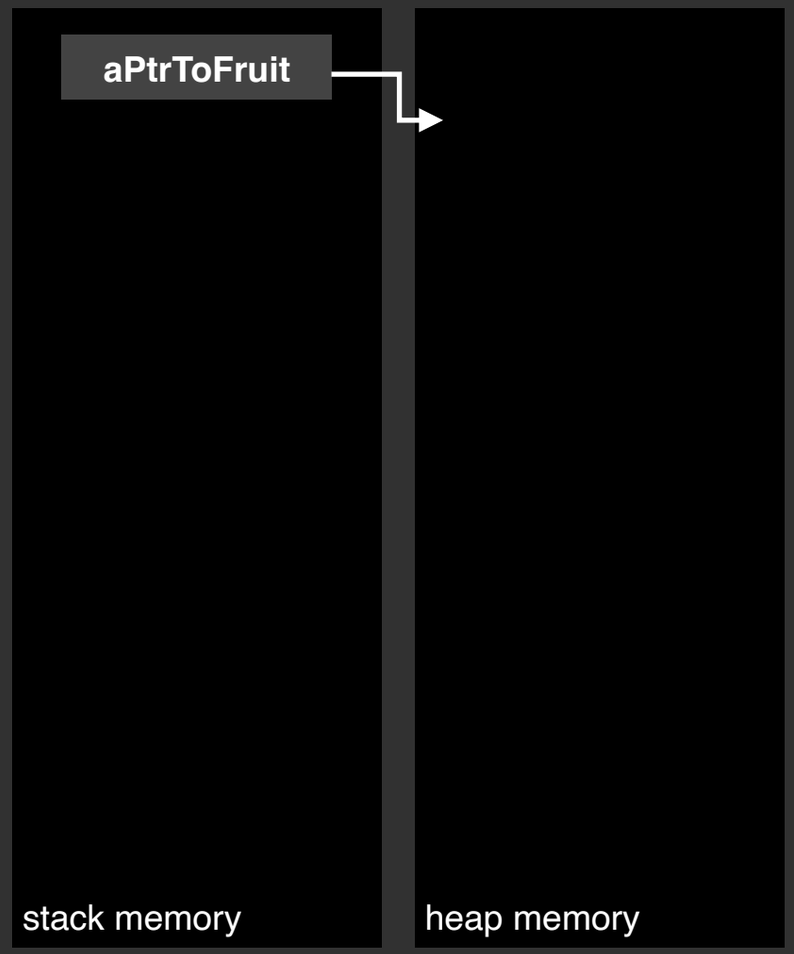


Pointers & Dynamic Memory Allocation

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | line 22

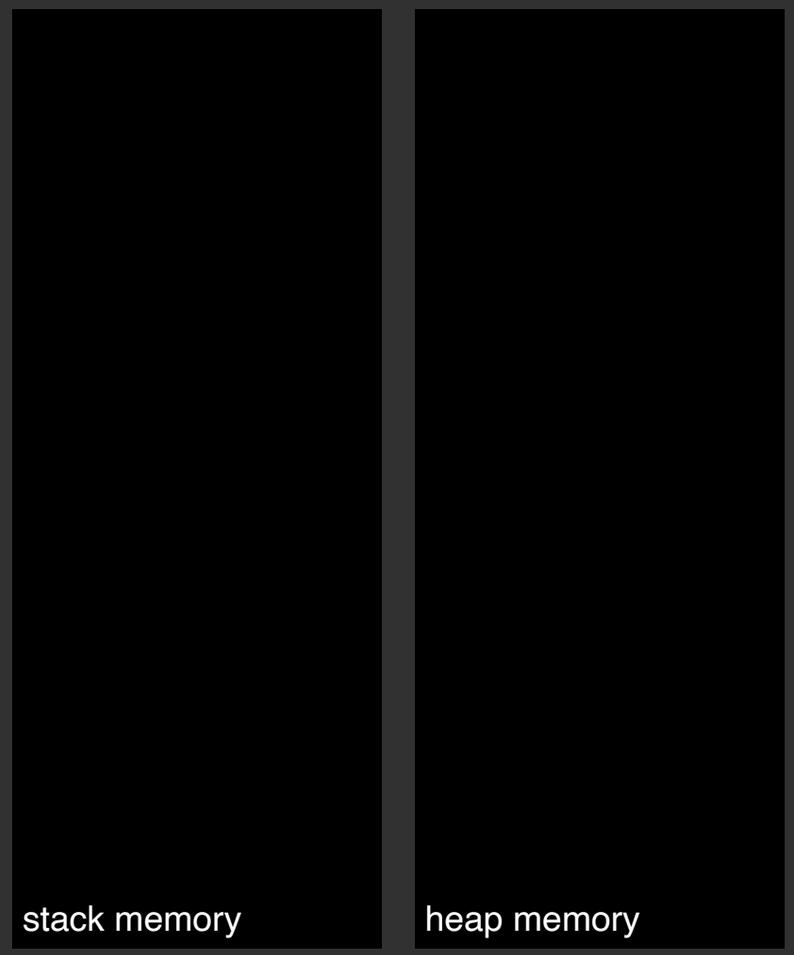


Pointers & Dynamic Memory Allocation

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | line 26



The Heap

- Generally, dynamic memory allocation is a great way to allocate memory on the fly when needed during runtime

- The memory area used for dynamic memory allocation is referred to as “The Heap” because memory piles up on-the-fly

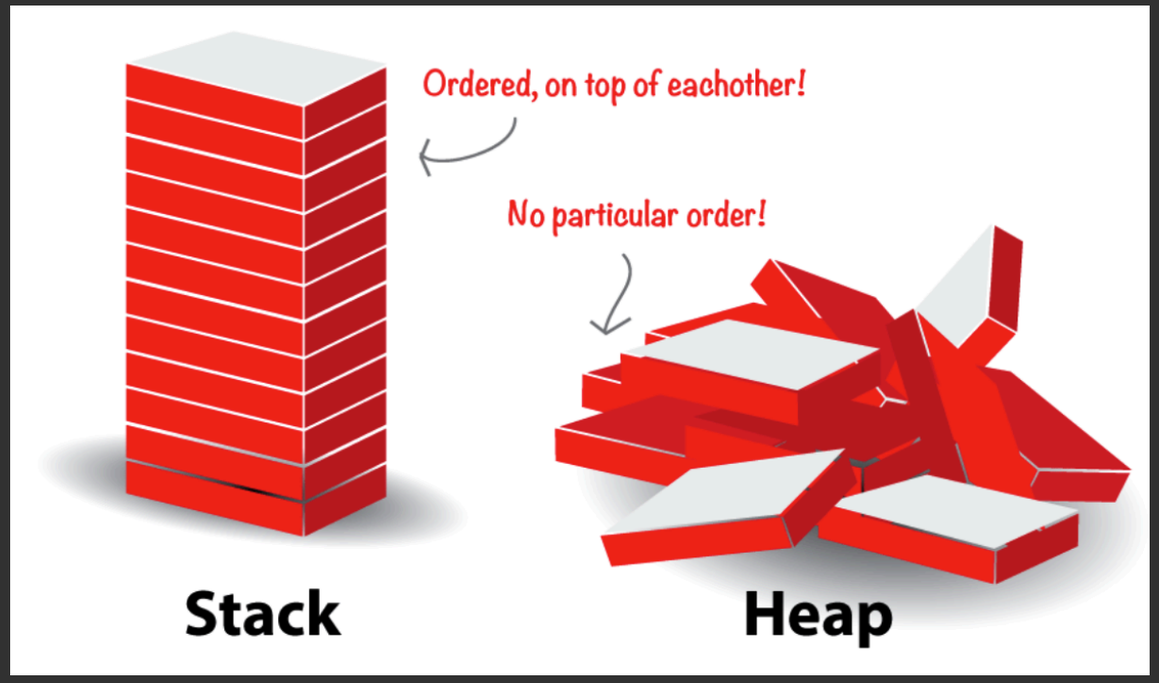


Image credit: https://stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap

Issues with Pointers

- Pointers can be used to point to & access dynamically allocated memory blocks

- However, two major difficulties come with it
  - Dangling pointers
  - Memory leaks

Dangling Pointers

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

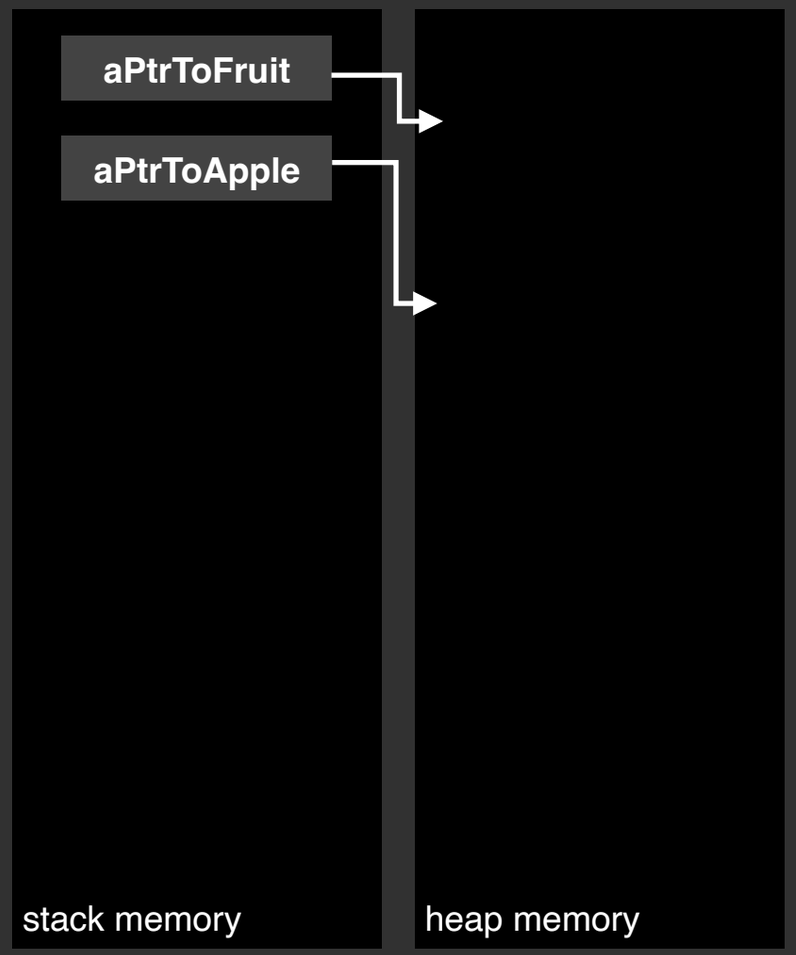
Dangling Pointers

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

Using a pointer after freeing the memory they point to causes undefined behavior

computer memory | line 22

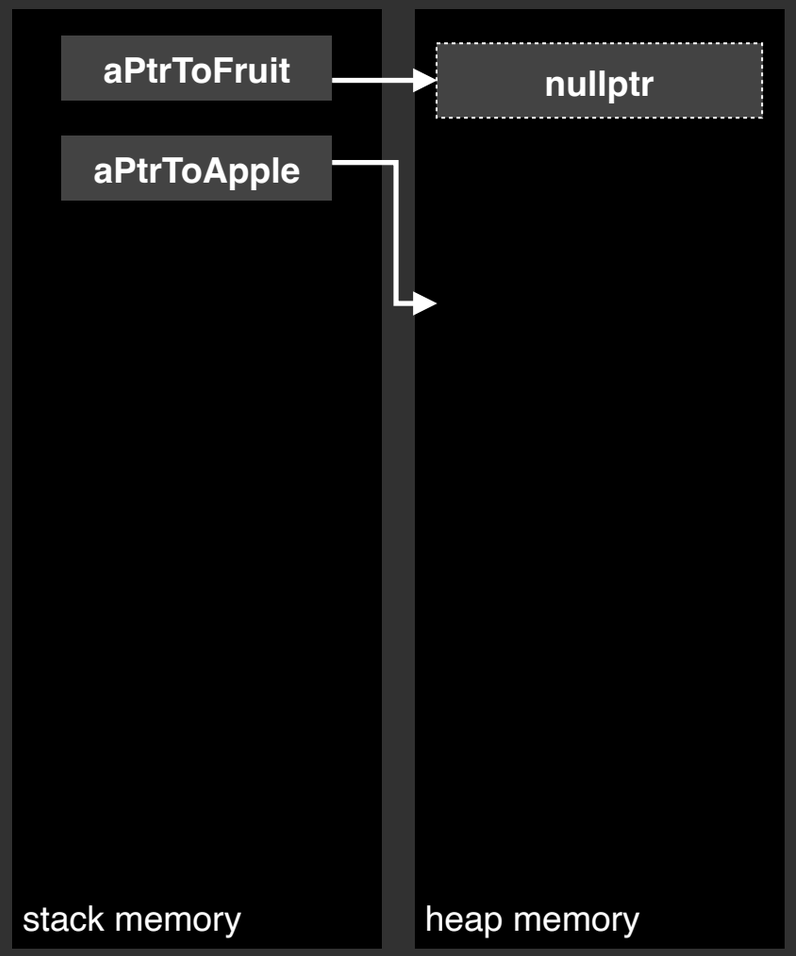


Dangling Pointers

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;   // if we do not free the allocated
23      aPtrToFruit = nullptr;
24
25      delete aPtrToApple;     // memory properly, it remains occupied
26      aPtrToApple = nullptr;
27
28      return 0;
29  }
30
```

computer memory | line 22

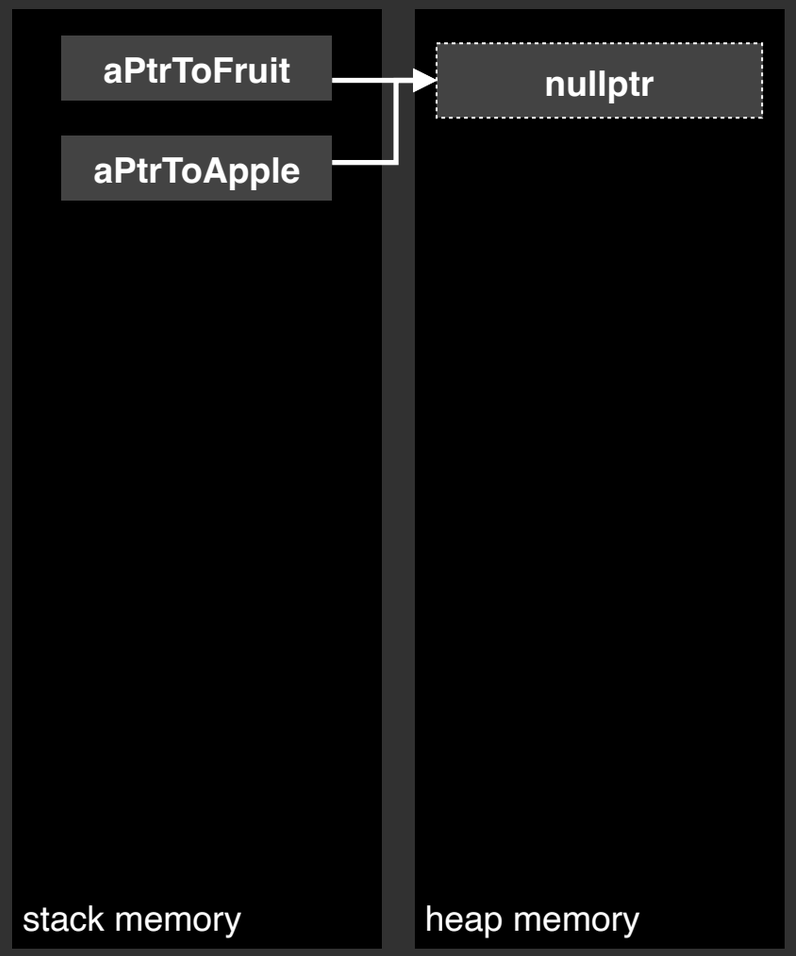


Dangling Pointers

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;   // if we do not free the allocated
23      aPtrToFruit = nullptr;
24
25      delete aPtrToApple;     // memory properly, it remains occupied
26      aPtrToApple = nullptr;
27
28      return 0;
29  }
30
```

computer memory | line 22



Memory Leaks

source code instructions

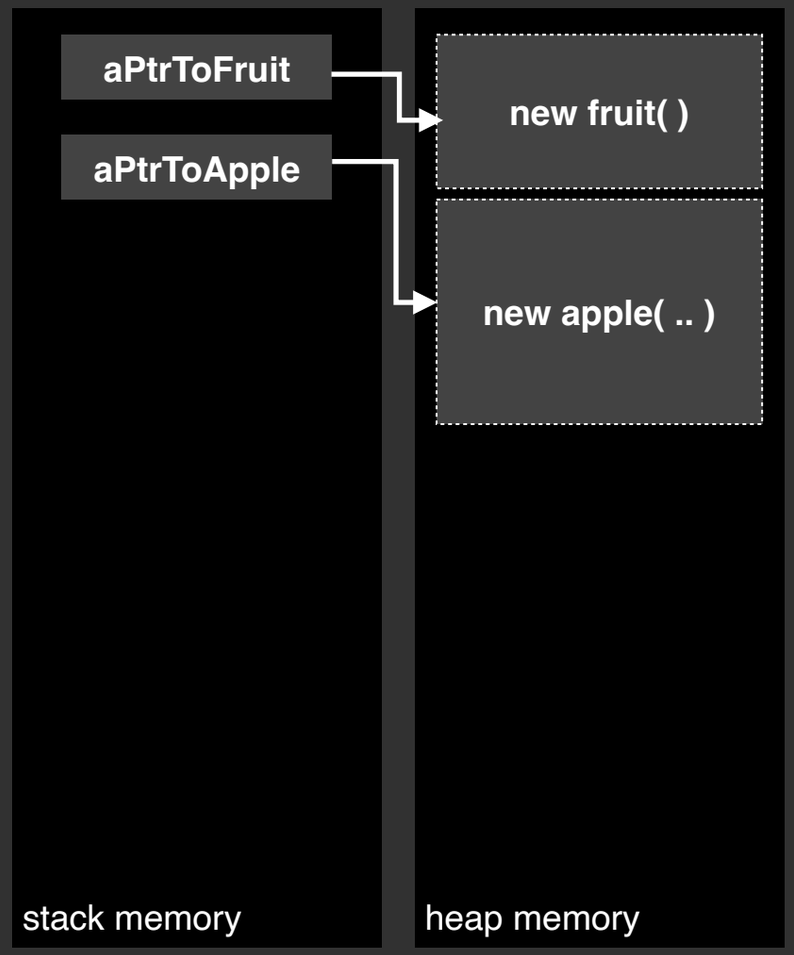
```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

Memory Leaks

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      delete aPtrToFruit;
23      delete aPtrToApple;
24
25      return 0;
26  }
27
```

computer memory | lines 21

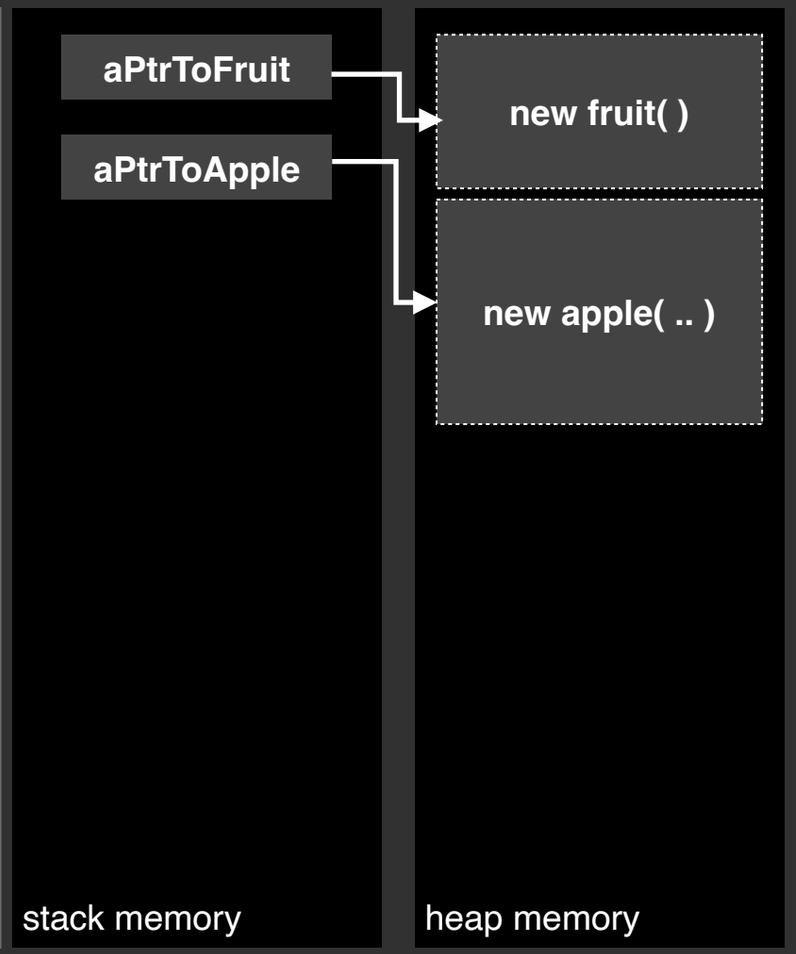


Memory Leaks

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      //delete aPtrToFruit;   // if we do not free the allocated
23      delete aPtrToApple;     // memory properly, it remains occupied
24
25      return 0;
26  }
27
```

computer memory | lines 21

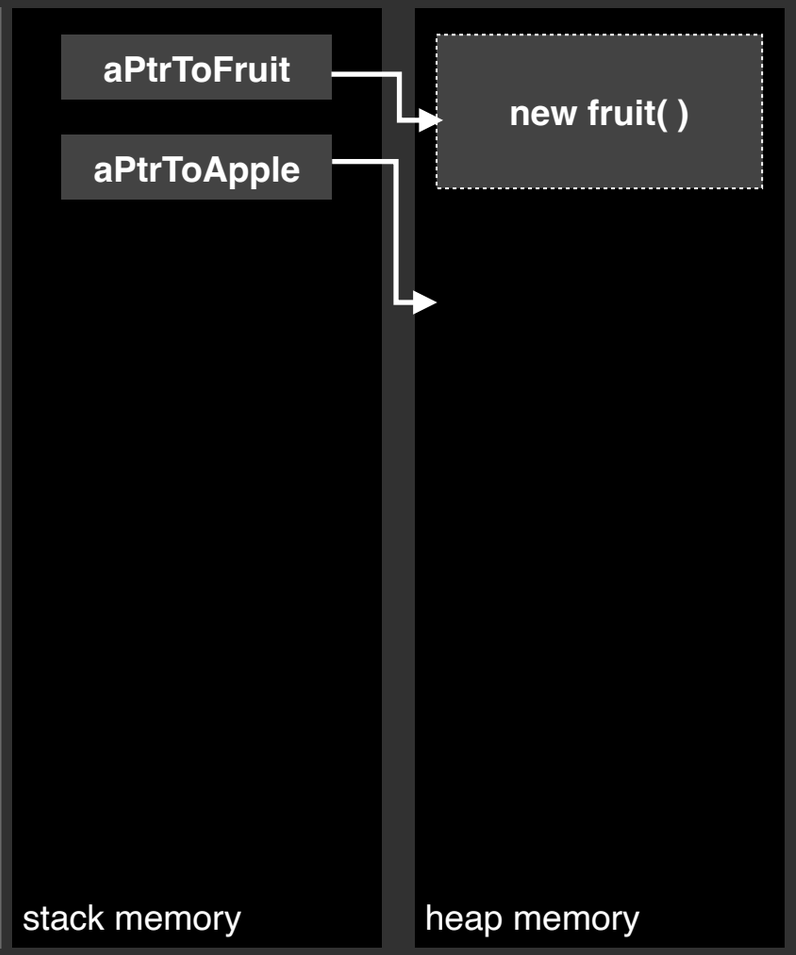


Memory Leaks

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      //delete aPtrToFruit;   // if we do not free the allocated
23      delete aPtrToApple;     // memory properly, it remains occupied
24
25      return 0;
26  }
27
```

computer memory | lines 21



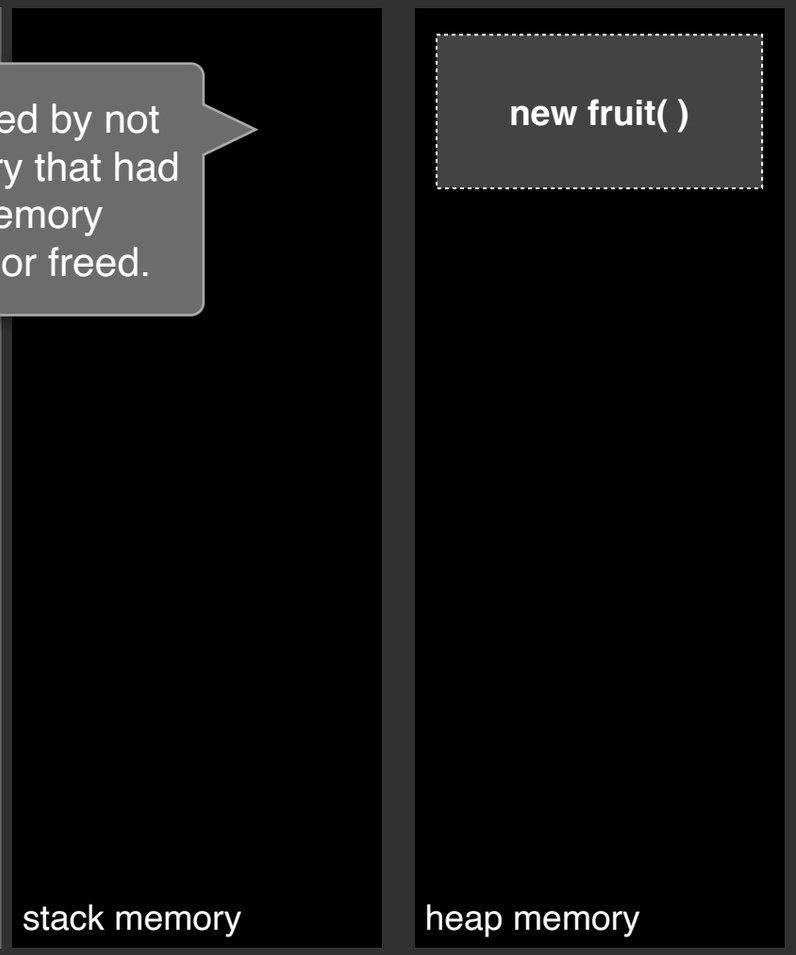
Memory Leaks

source code instructions

```
 9  #include "fruit.h"
10  #include "apple.h"
11  #include "banana.h"
12
13
14  int main()
15  {
16      fruit* aPtrToFruit{ new fruit() };
17      apple* aPtrToApple{ new apple(apple::appleType::BRAEBURN) };
18
19      aPtrToFruit->printName();
20      aPtrToApple->printName();
21
22      //delete aPtrToFruit;   // if we do not free the allocated
23      delete aPtrToApple;     // memory properly, it remains occupied
24
25      return 0;
26  }
27
```

A memory leak is created by not properly freeing memory that had been allocated. This memory block cannot be used nor freed.

computer memory | lines 21



Take Away

- Dynamic memory allocation is not as fast & easy to use as automatic memory allocation

- Dynamic memory allocation however is necessary whenever you have to handle dynamic data that needs to be stored during the runtime of your program

- Smart pointers circumvent most of the issues with raw pointers and let you deal with dynamic data more easily