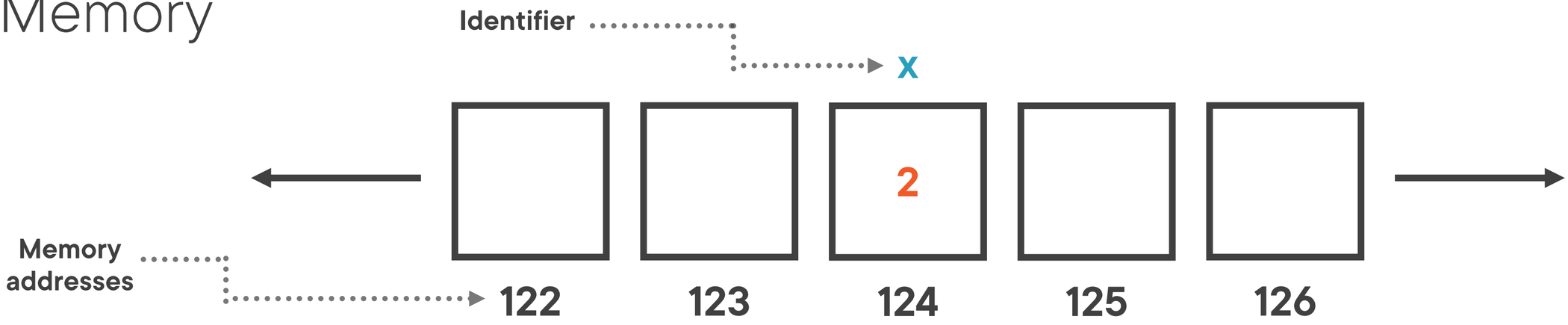
Dynamic Memory Allocation

Mateo Prigl
Software Developer

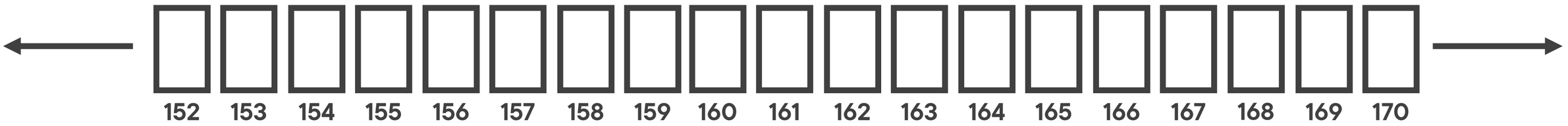
```
int x = 2;

&x;      // 124
// & - address-of operator
```

Memory



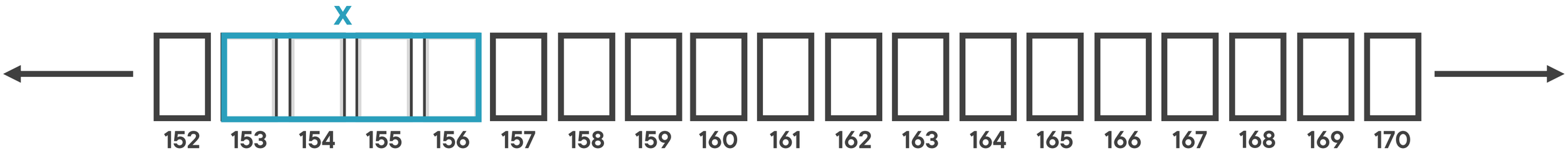
```
int x = 2;
```

Memory (Byte Sequence)



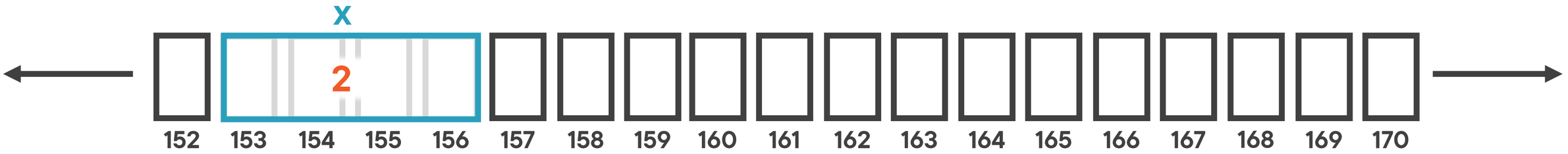
```
int x = 2;
```

Memory (Byte Sequence)



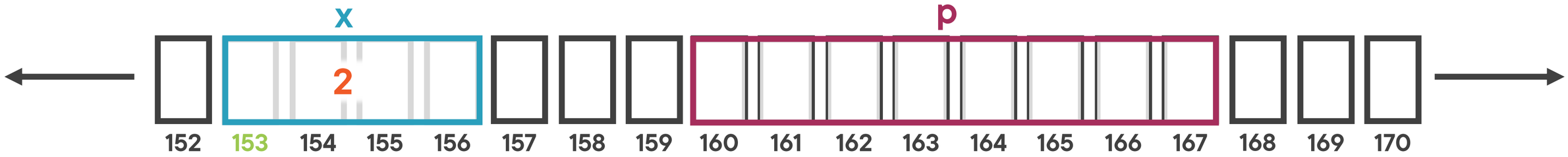
```
int x = 2;

&x;      // 153
```

Memory (Byte Sequence)



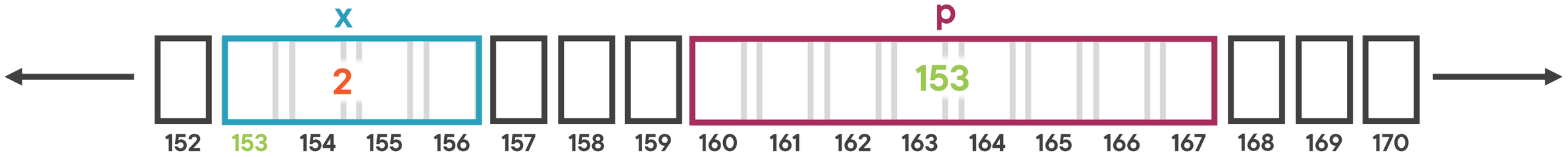
```
int x = 2;

&x;      // 153

int *p = &x;
```

Memory (Byte Sequence)



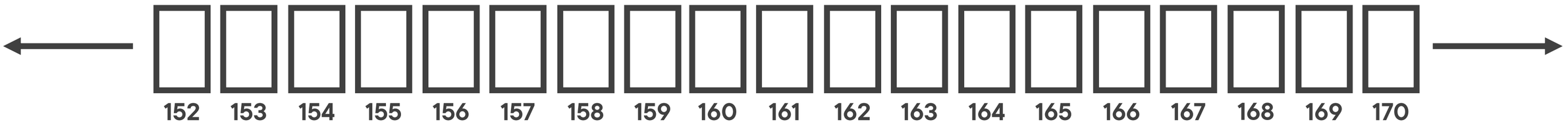
```
int x = 2;

&x;      // 153

int *p = &x;

*p;      // 2
```

Memory (Byte Sequence)



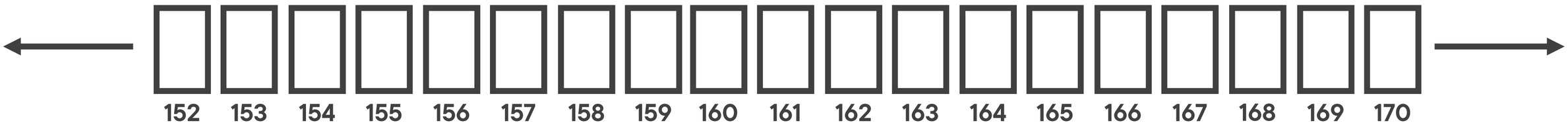
```
int x = 2;

&x;      // 153

int *p = &x;

*p;      // 2
```

Memory (Byte Sequence)



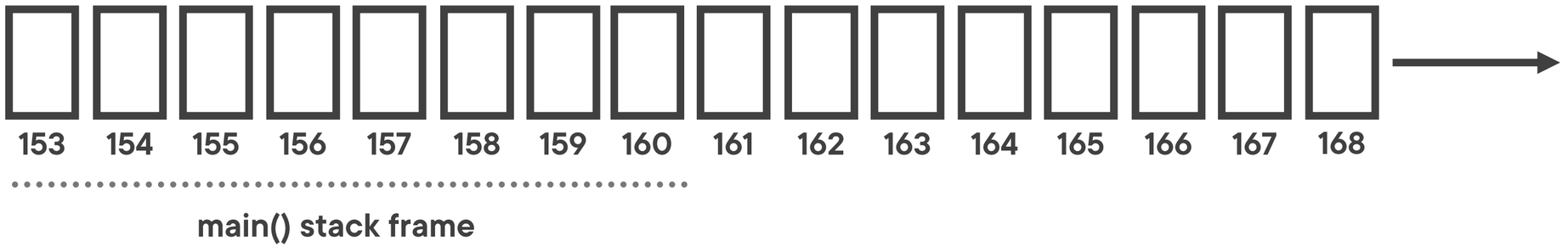
```
int x = 2;

&x;      // 153
```

Memory (Byte Sequence)



Stack

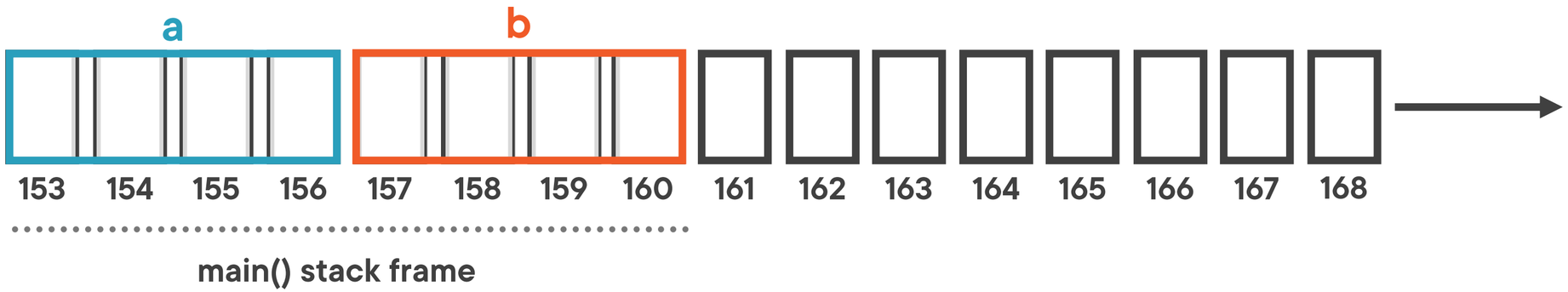
```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```



Stores local variables

Stack

```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

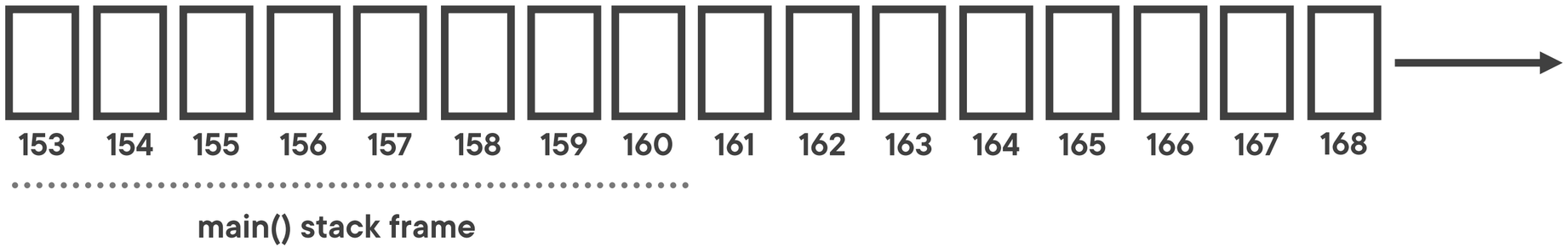


Stores local variables

Managed by the CPU

Stack

```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

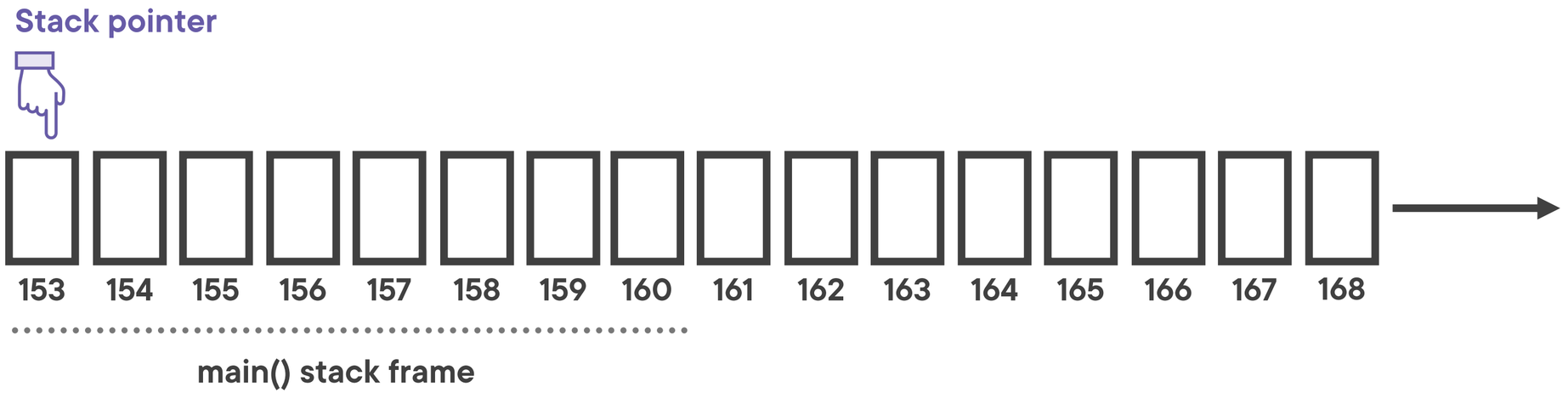


Stores local variables

Managed by the CPU

```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

Stack

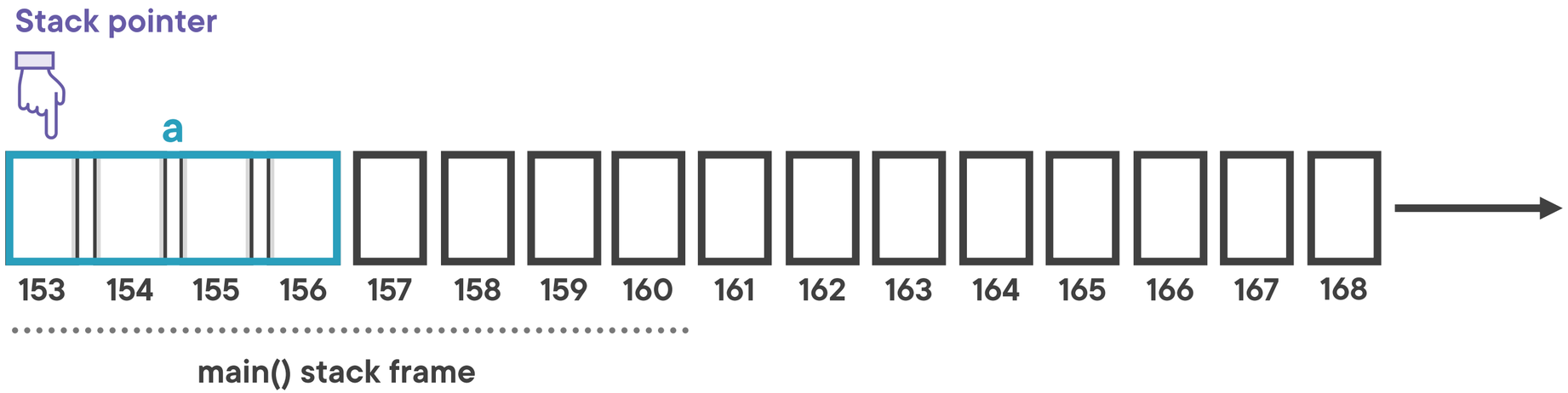


Stores local variables

Managed by the CPU

```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

Stack

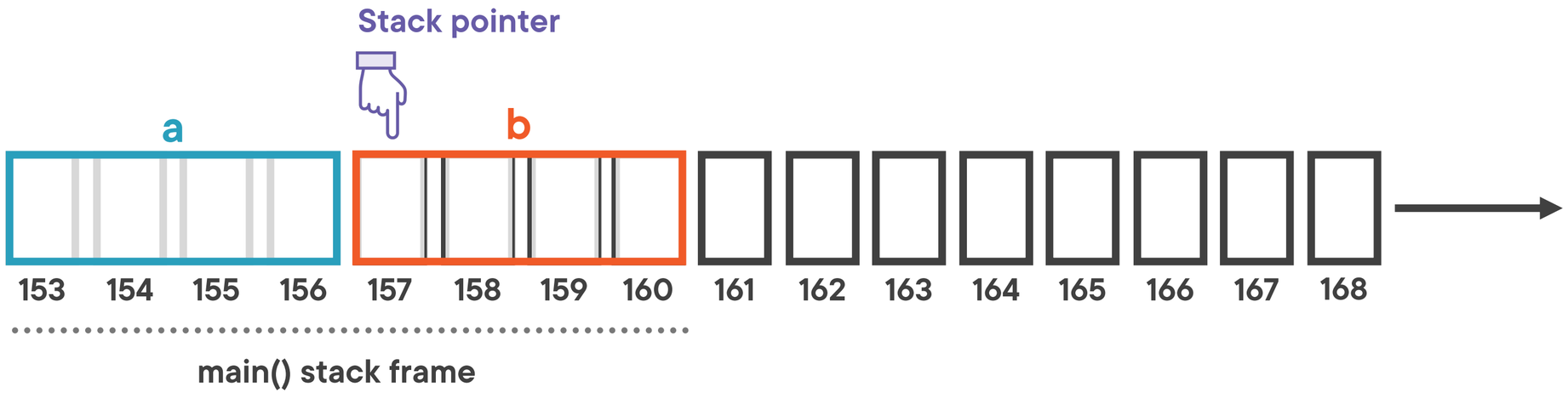


Stores local variables

Managed by the CPU

```c
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

Stack

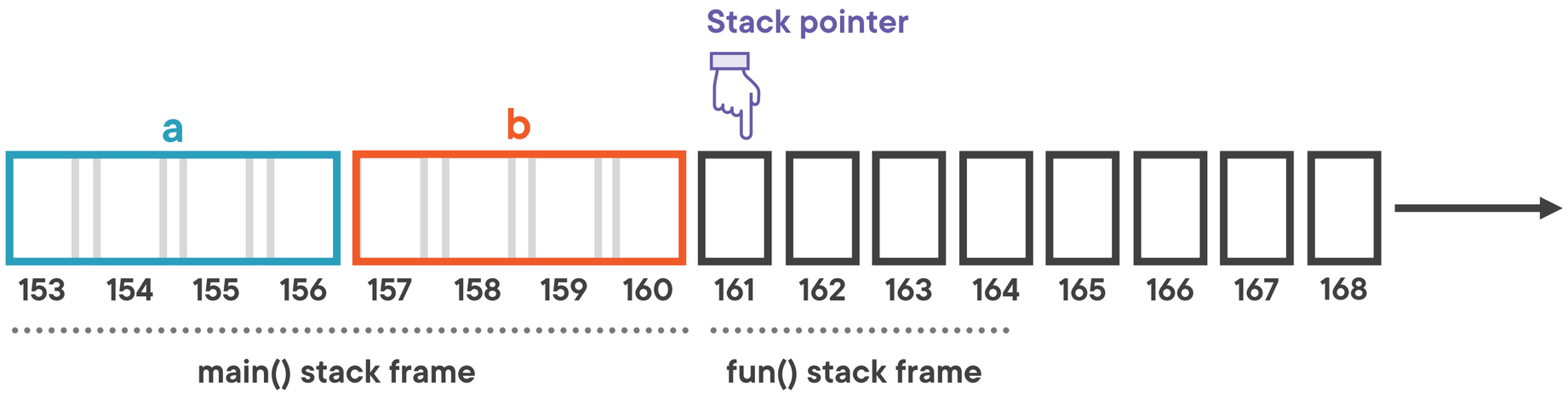


Stores local variables

Managed by the CPU

```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

Stack

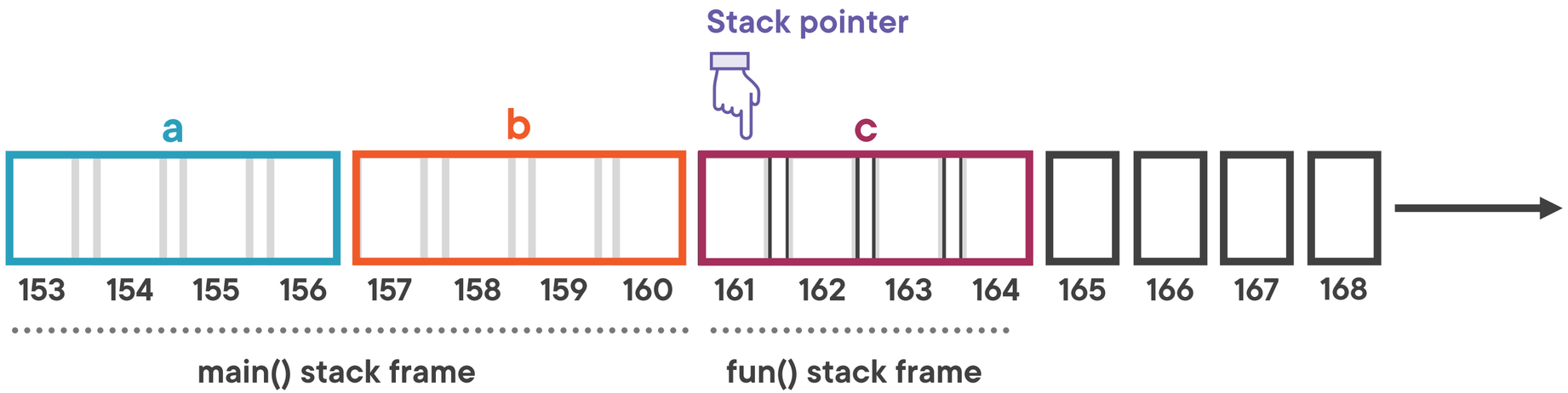


Stores local variables

Managed by the CPU

Stack

```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

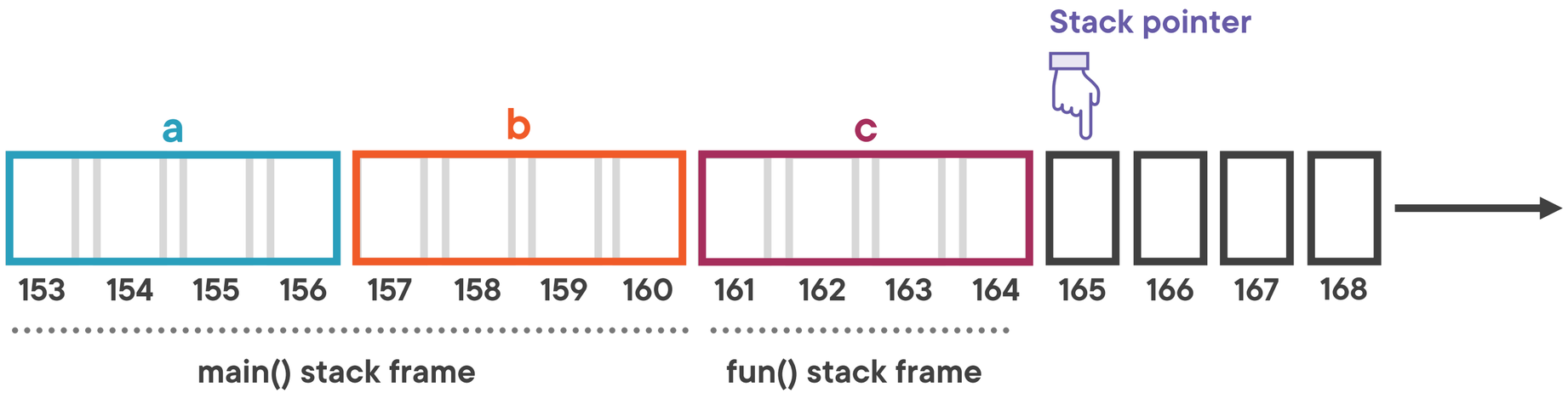


Stores local variables

Managed by the CPU

Stack

```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

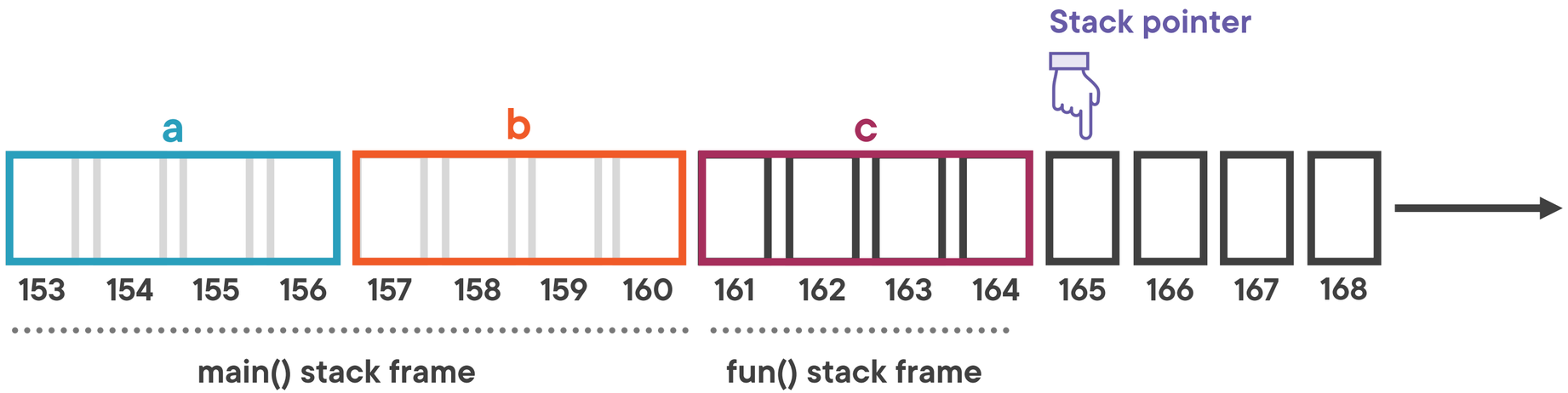


Stores local variables

Managed by the CPU

Stack

```
void fun()
{
    int c;
}

int main()
{
    int a;
    int b;
    fun();
}
```

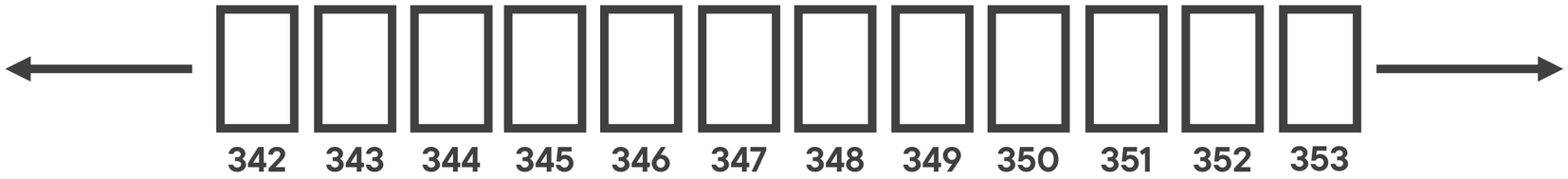


Stores local variables

Managed by the CPU

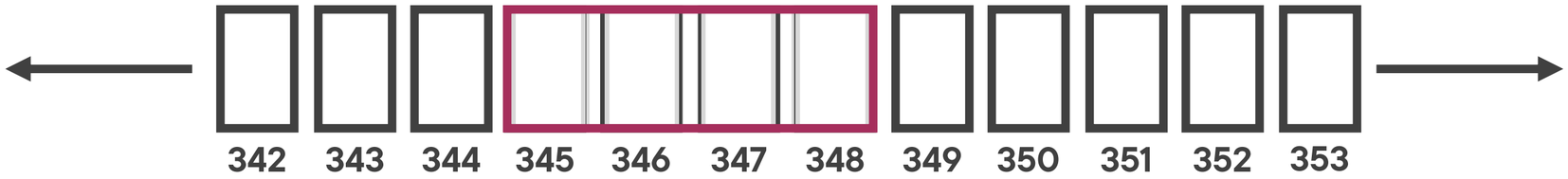
Heap (Free Store)

```
int main()
{


        new


}
```



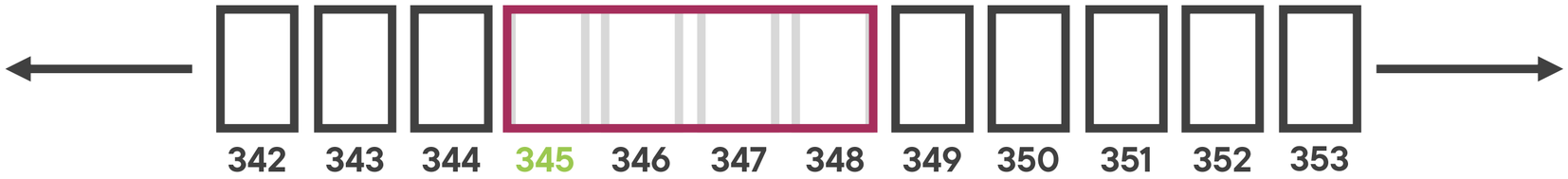
```
int main()
{


        new int;


}
```

Heap (Free Store)



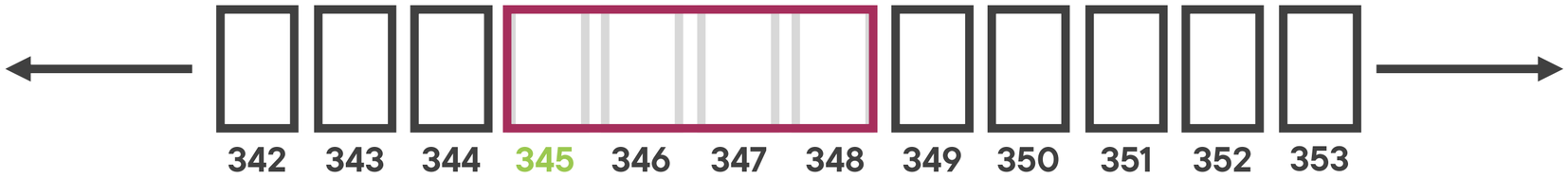
```
int main()
{


        new int;


}
```

Heap (Free Store)

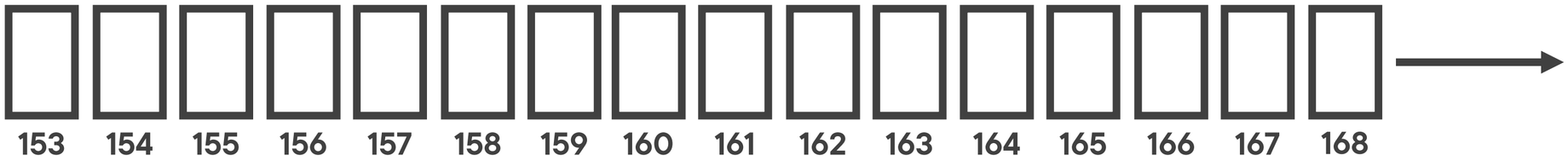


```
int main()
{


int *x = new int;


}
```

Heap (Free Store)

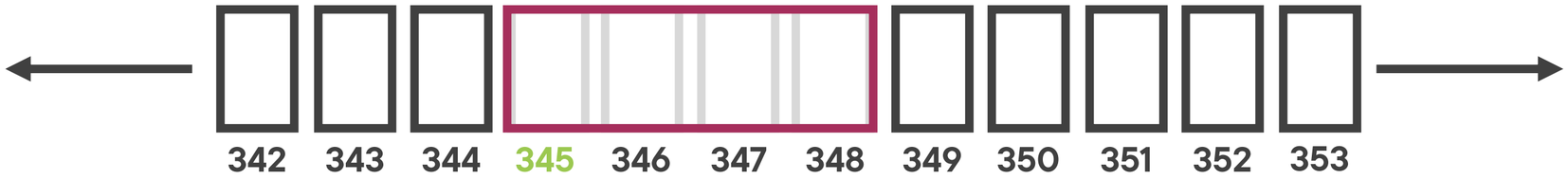


Stack

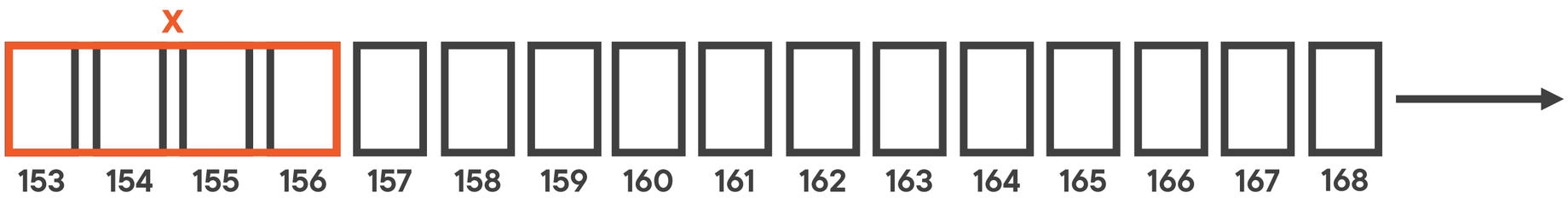


```
int main()
{


int *x = new int;


}
```

Heap (Free Store)

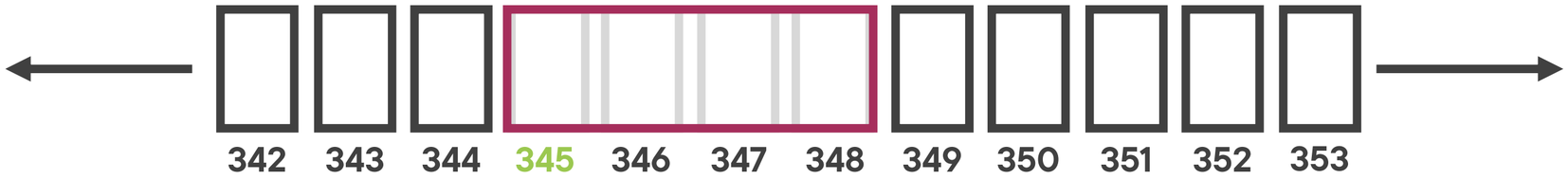


Stack

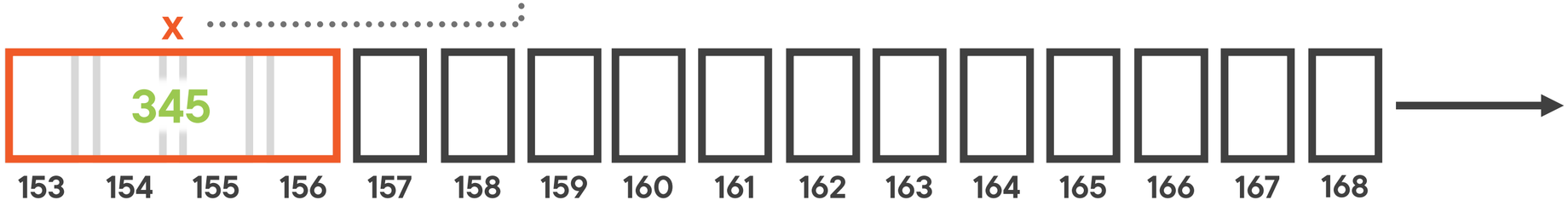


```
int main()
{


int *x = new int;


}
```

Heap (Free Store)

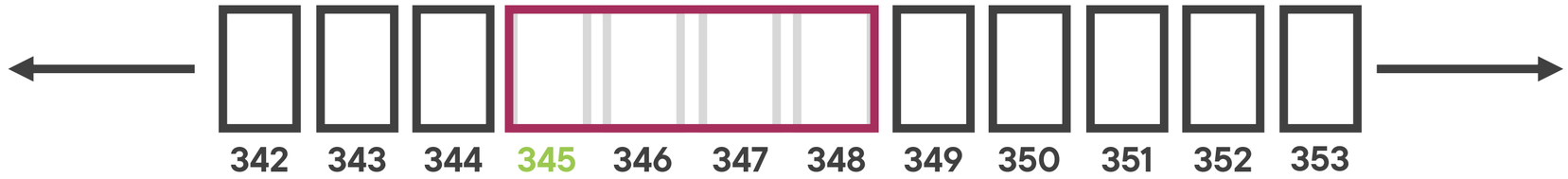


Stack

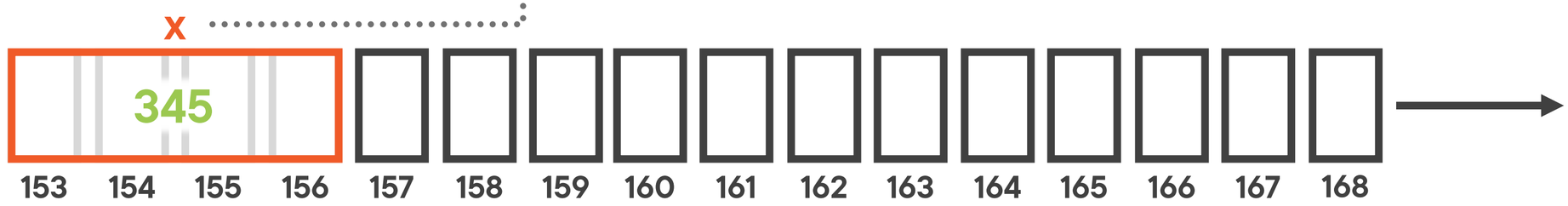


```
int main()
{

  {

int *x = new int;

  }

}
```

Heap (Free Store)

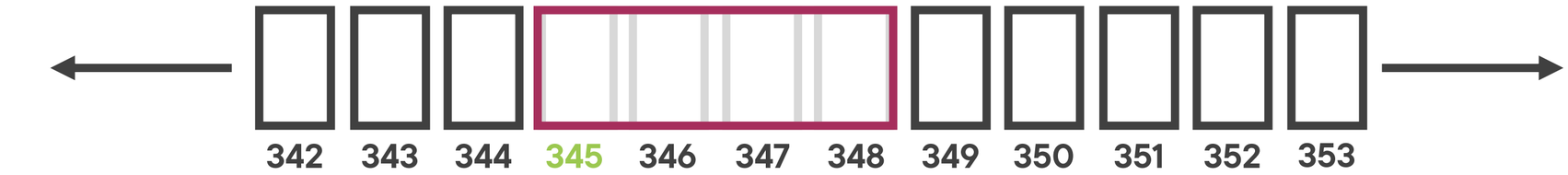


Stack

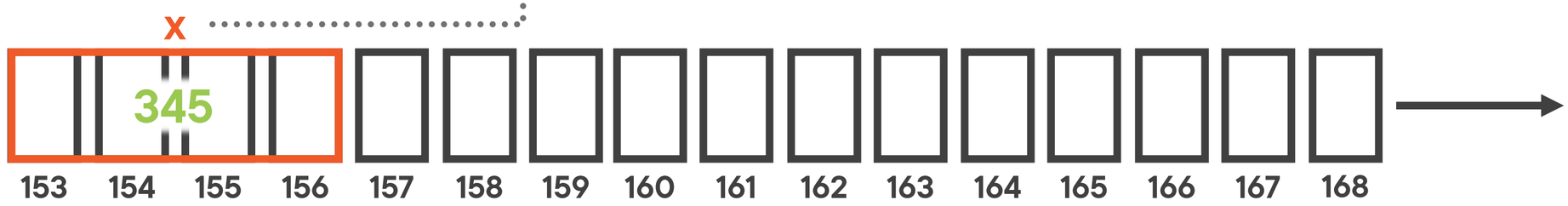


```
int main()
{

  {

int *x = new int;

  }

}
```

Heap (Free Store)

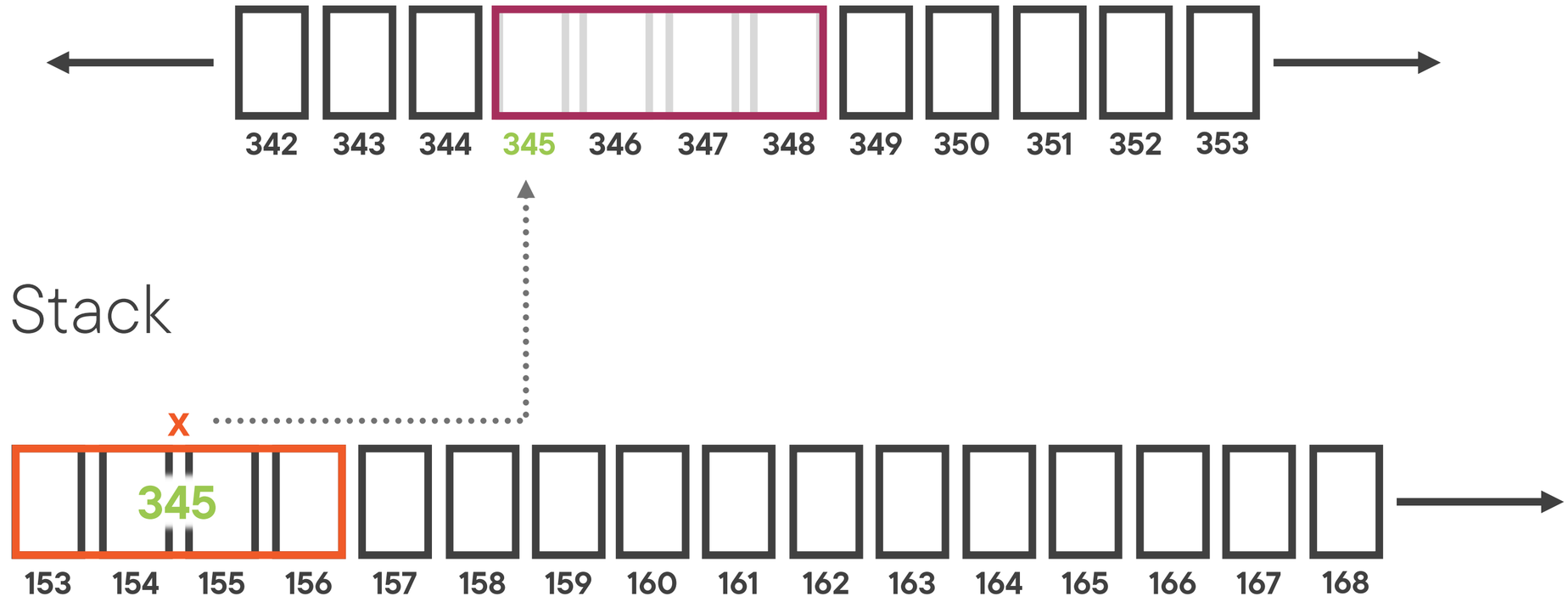


Stack



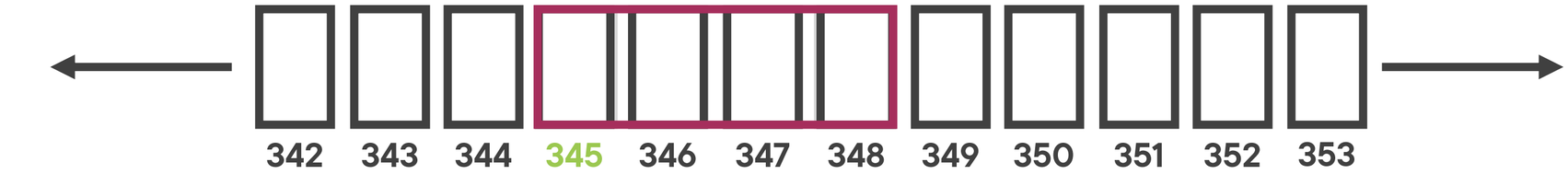
```
int main()
{

  {

int *x = new int;
delete x;
  }

}
```

Heap (Free Store)

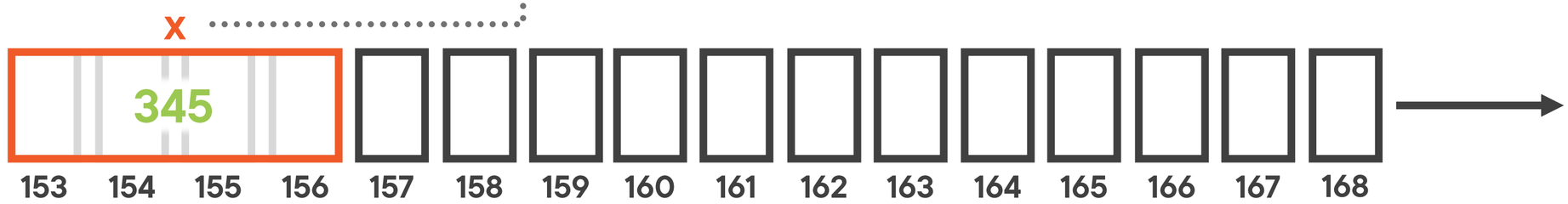


```
int main()
{

  {

int *x = new int;
delete x;
  }

}
```

Heap (Free Store)

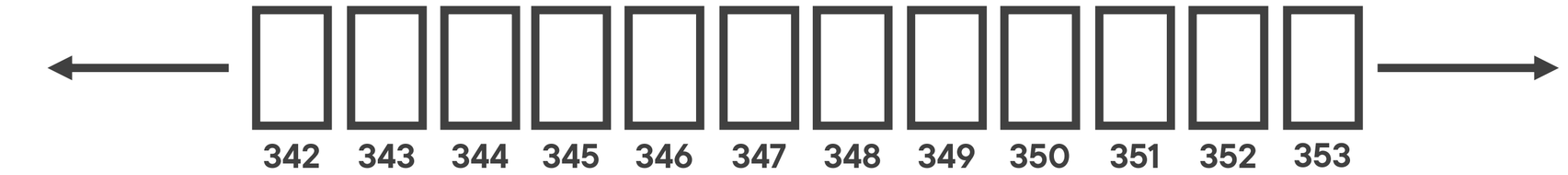


Stack

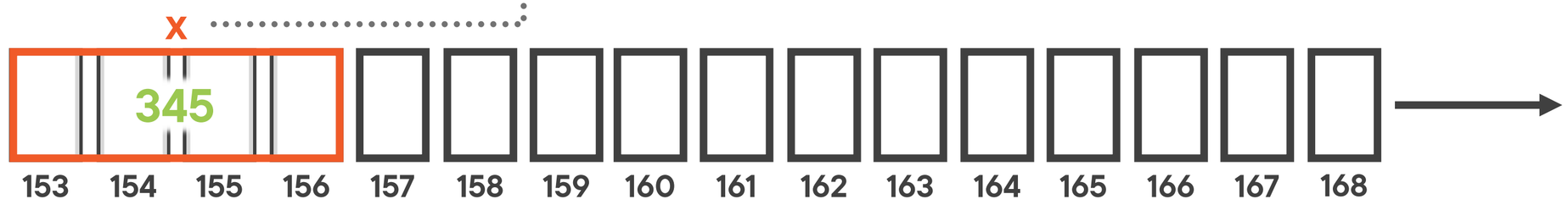


```
int main()
{

  {

int *x = new int;
delete x;
  }

}
```

Heap (Free Store)



Stack



Stack vs. Heap

Stack

Predefined size ~2MB

Scope based resource management

Memory allocations are fast

Heap

No predefined size, substantially larger

Memory allocations are slower with inconsistent speed

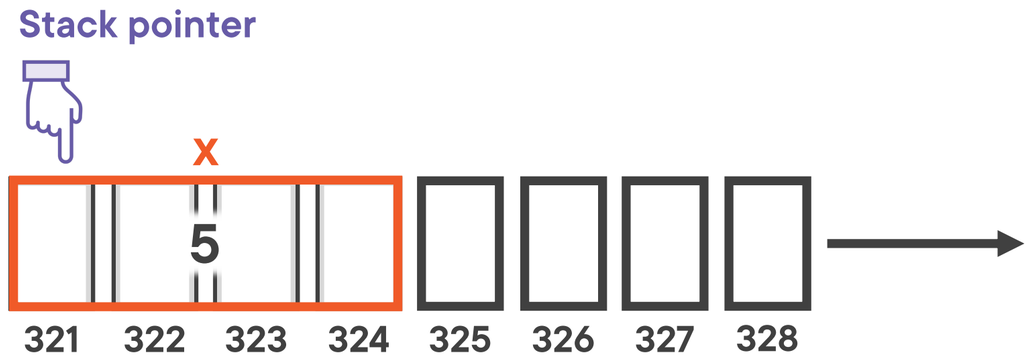
Supports large memory allocations

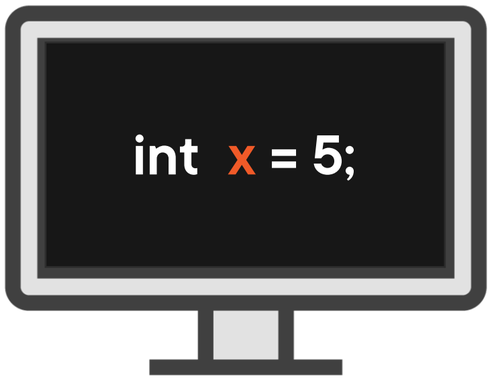
Allocation flexibility (e.g. dynamic data structures)

A lot of STL containers use heap

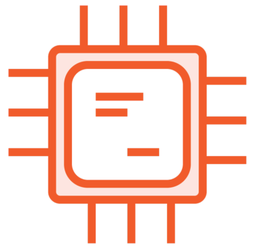
Dynamic memory allocation (at runtime)

Stack

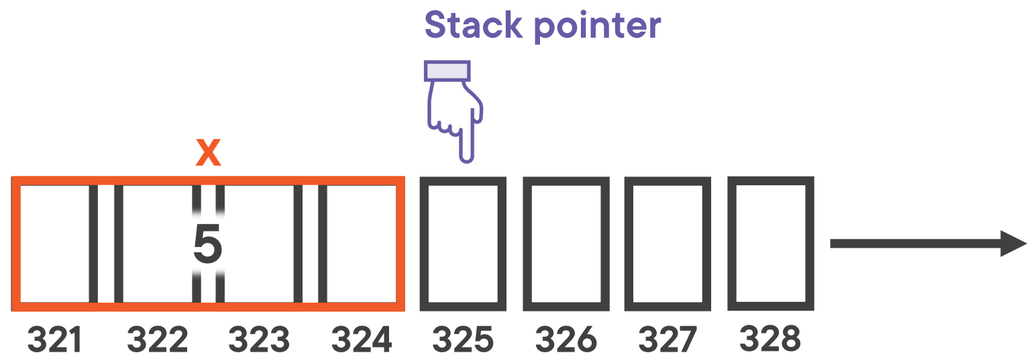


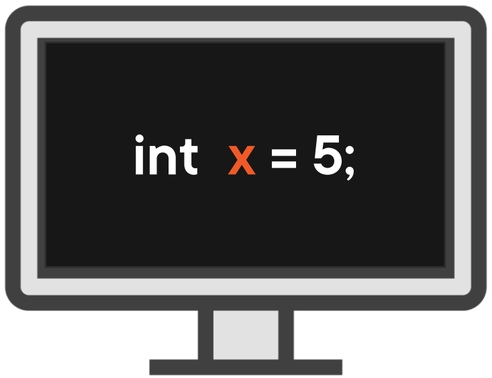



```
mov     DWORD PTR [rbp-4], 5
```

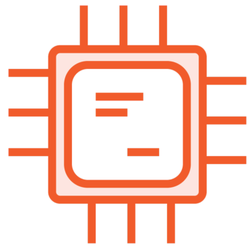


Stack

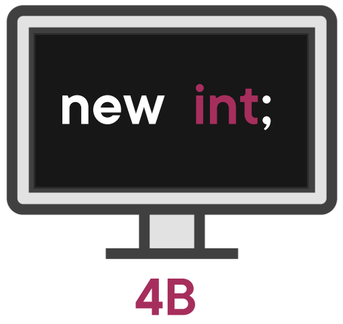


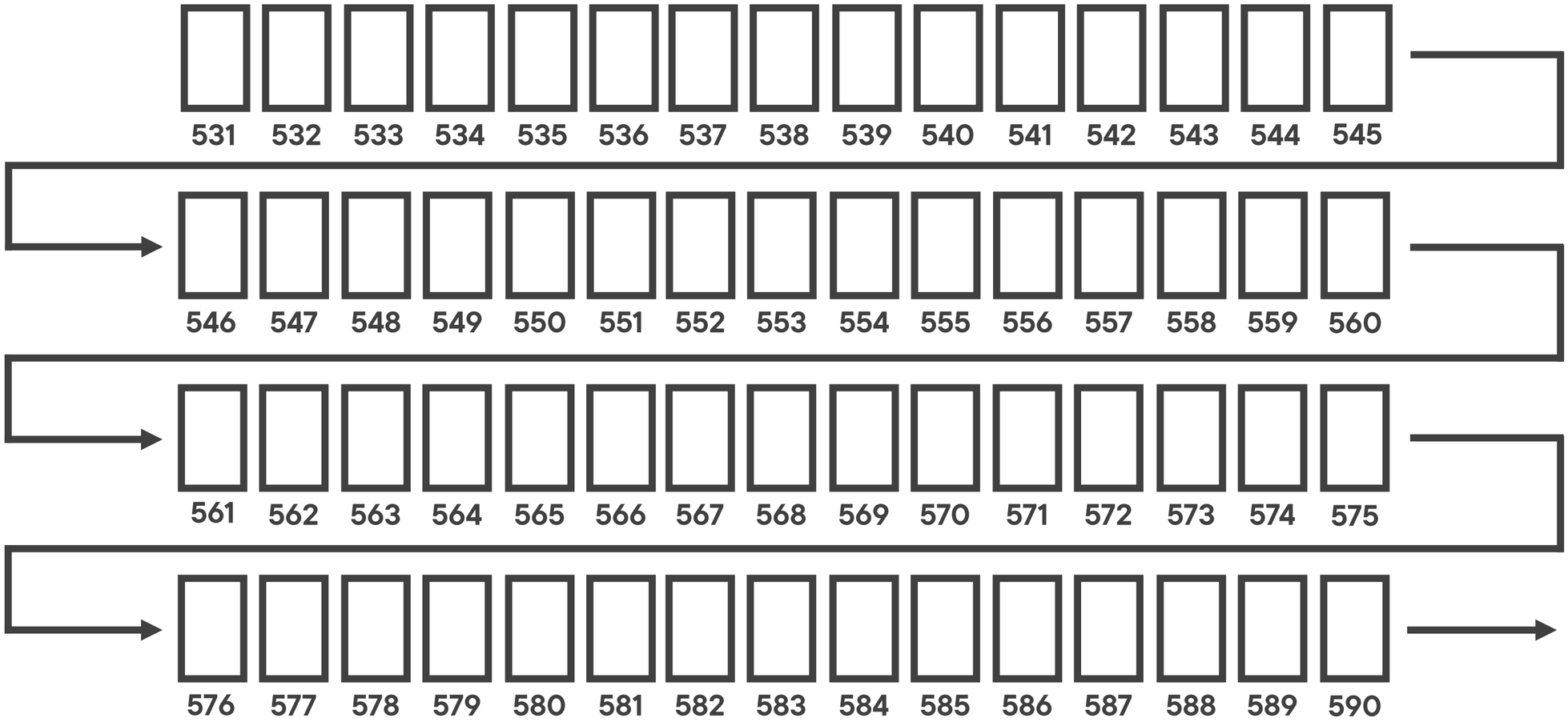


pop rbp

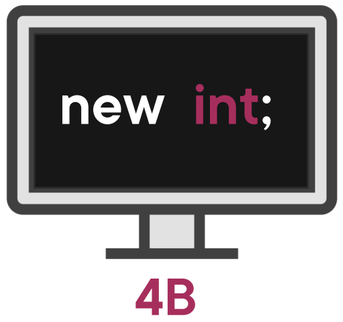


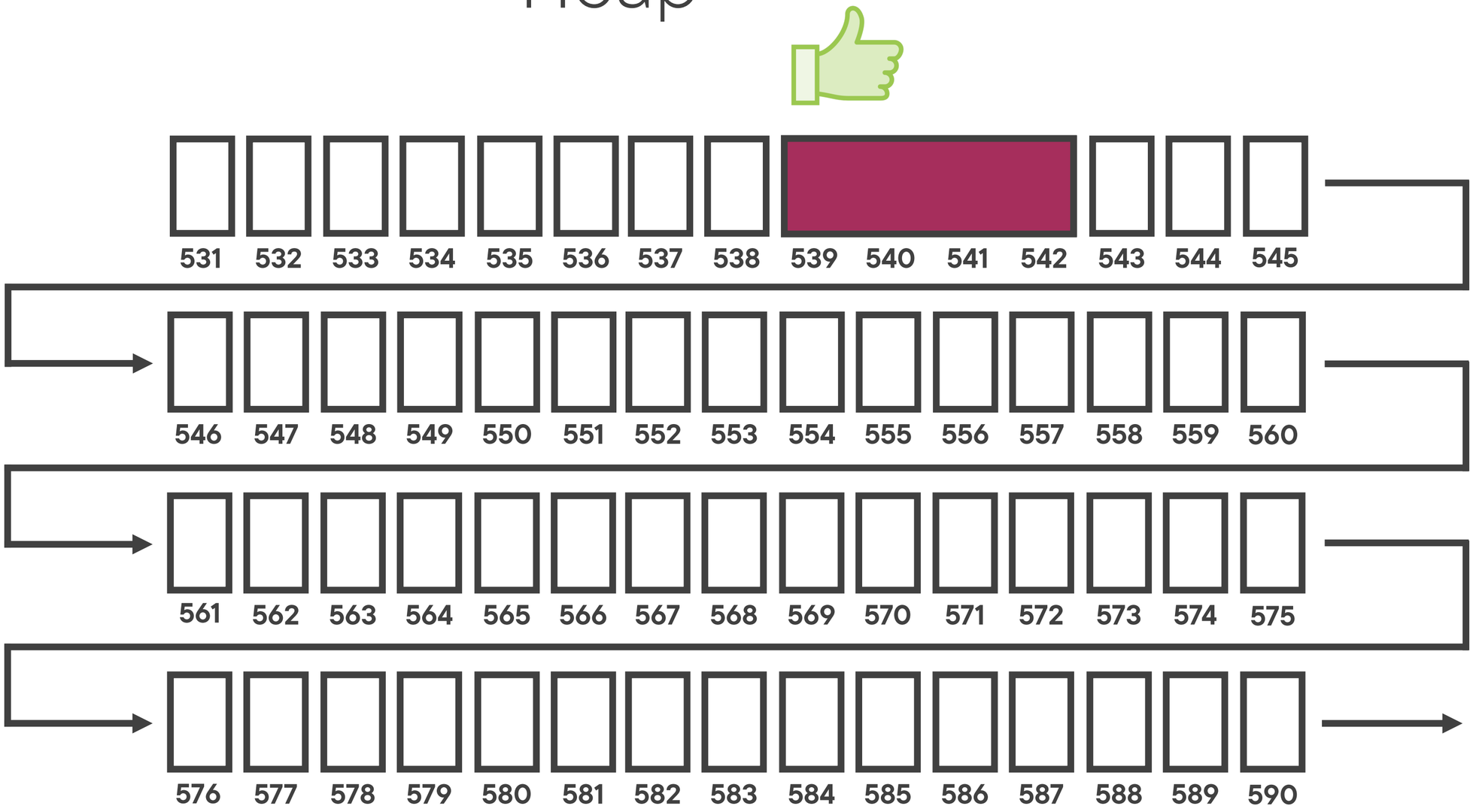
Heap


new int;
4B

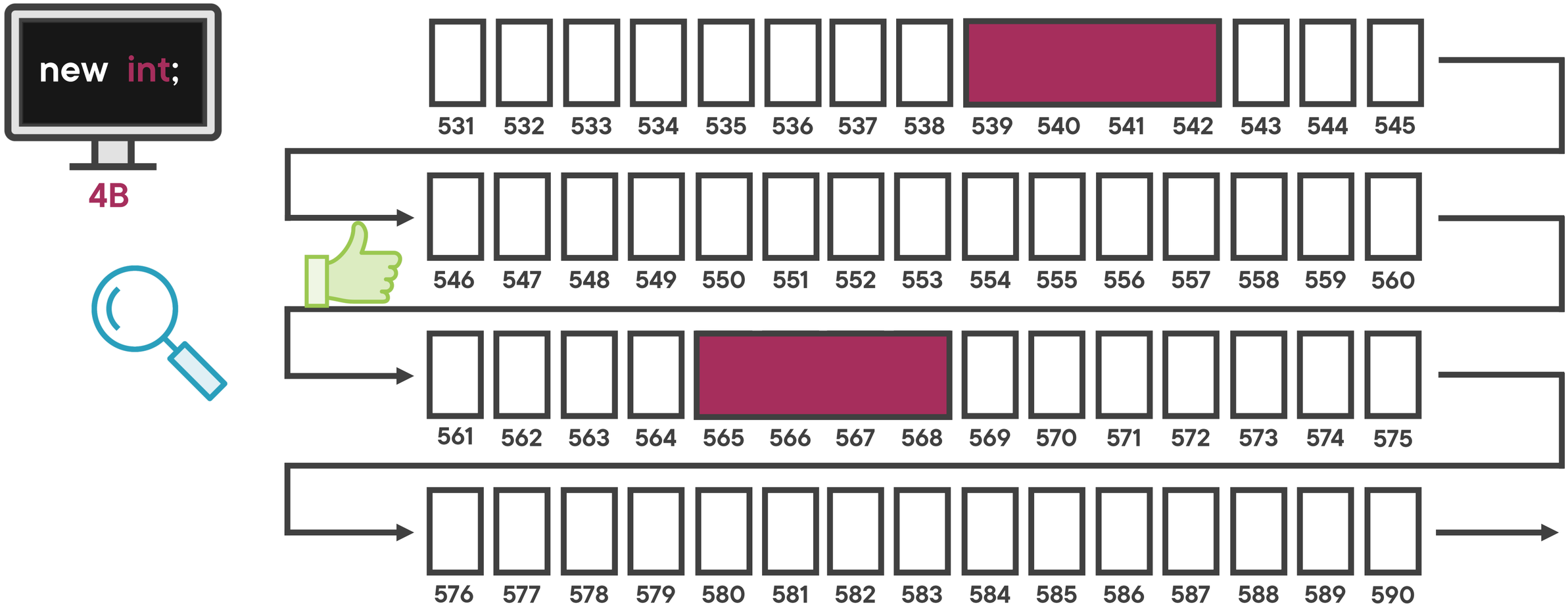

531 532 533 534 535 536 537 538 539 540 541 542 543 544 545
546 547 548 549 550 551 552 553 554 555 556 557 558 559 560
561 562 563 564 565 566 567 568 569 570 571 572 573 574 575
576 577 578 579 580 581 582 583 584 585 586 587 588 589 590

Heap

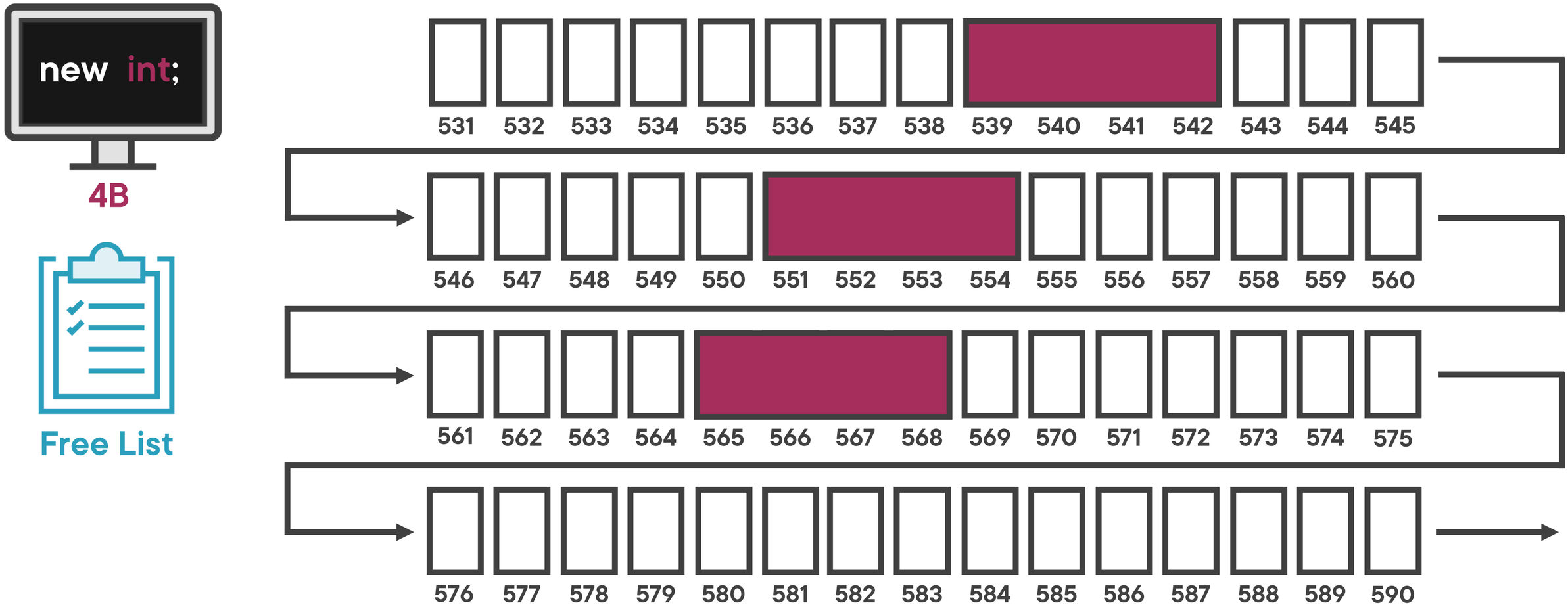

new int;
4B


531 532 533 534 535 536 537 538 539 540 541 542 543 544 545
546 547 548 549 550 551 552 553 554 555 556 557 558 559 560
561 562 563 564 565 566 567 568 569 570 571 572 573 574 575
576 577 578 579 580 581 582 583 584 585 586 587 588 589 590

Heap


new int;
4B
531 532 533 534 535 536 537 538 539 540 541 542 543 544 545
546 547 548 549 550 551 552 553 554 555 556 557 558 559 560
561 562 563 564 565 566 567 568 569 570 571 572 573 574 575
576 577 578 579 580 581 582 583 584 585 586 587 588 589 590

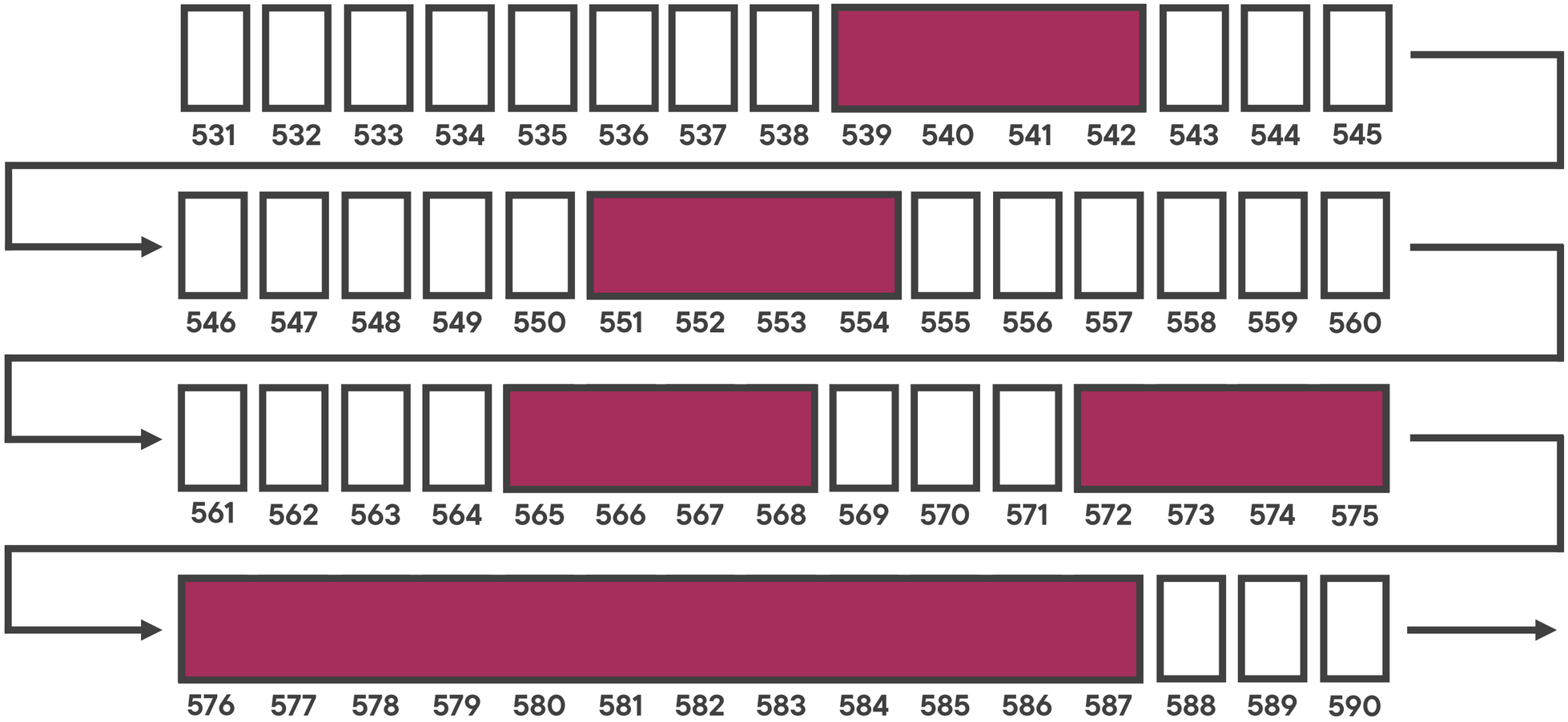
Heap


new int;
4B
Free List
531 532 533 534 535 536 537 538 539 540 541 542 543 544 545
546 547 548 549 550 551 552 553 554 555 556 557 558 559 560
561 562 563 564 565 566 567 568 569 570 571 572 573 574 575
576 577 578 579 580 581 582 583 584 585 586 587 588 589 590

Heap

16B

Free List



Video Game

HIGHSCORE: 0

Player

Enemy

Video Game

HIGHSCORE: 00

Player

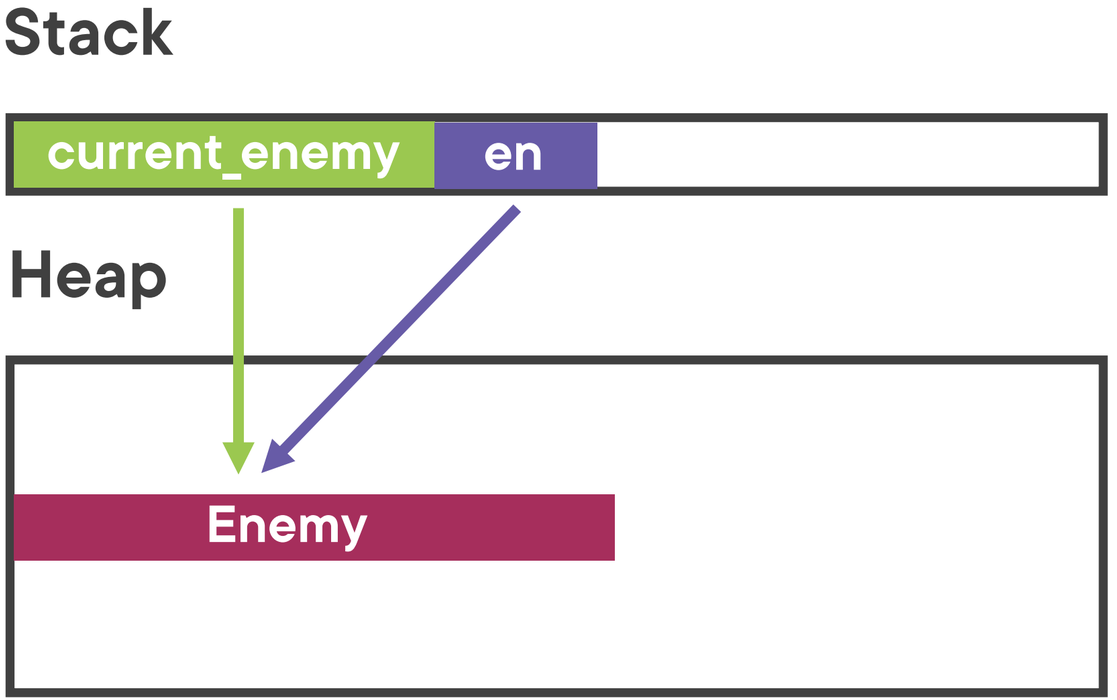
Enemy

Video Game

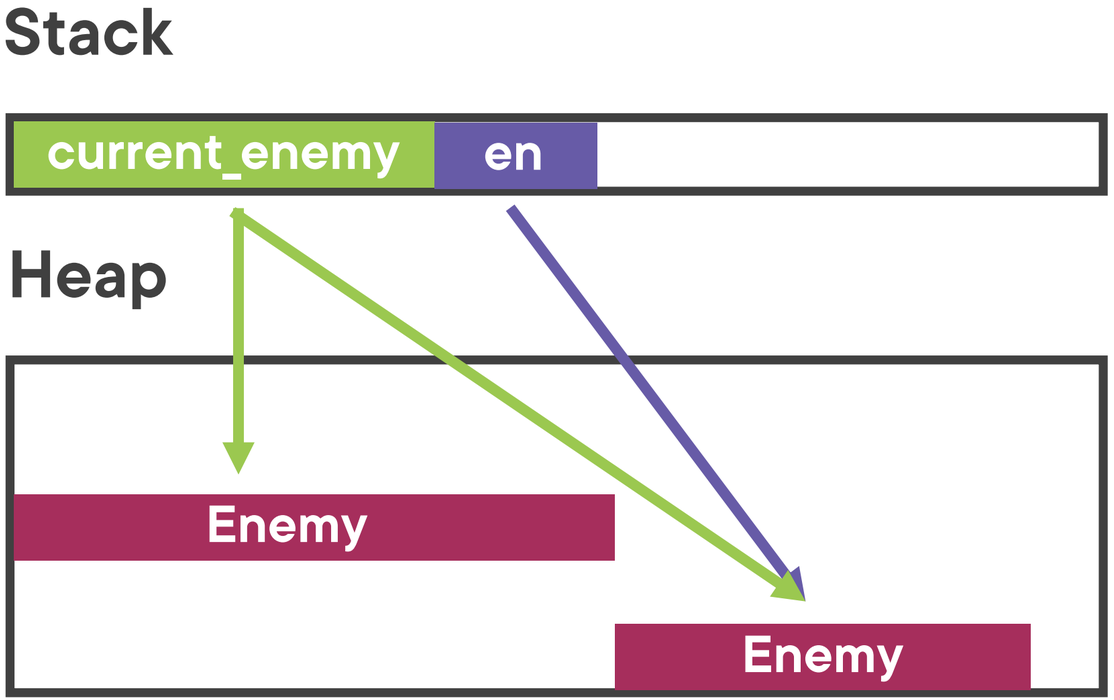
HIGHSCORE: 200

Player

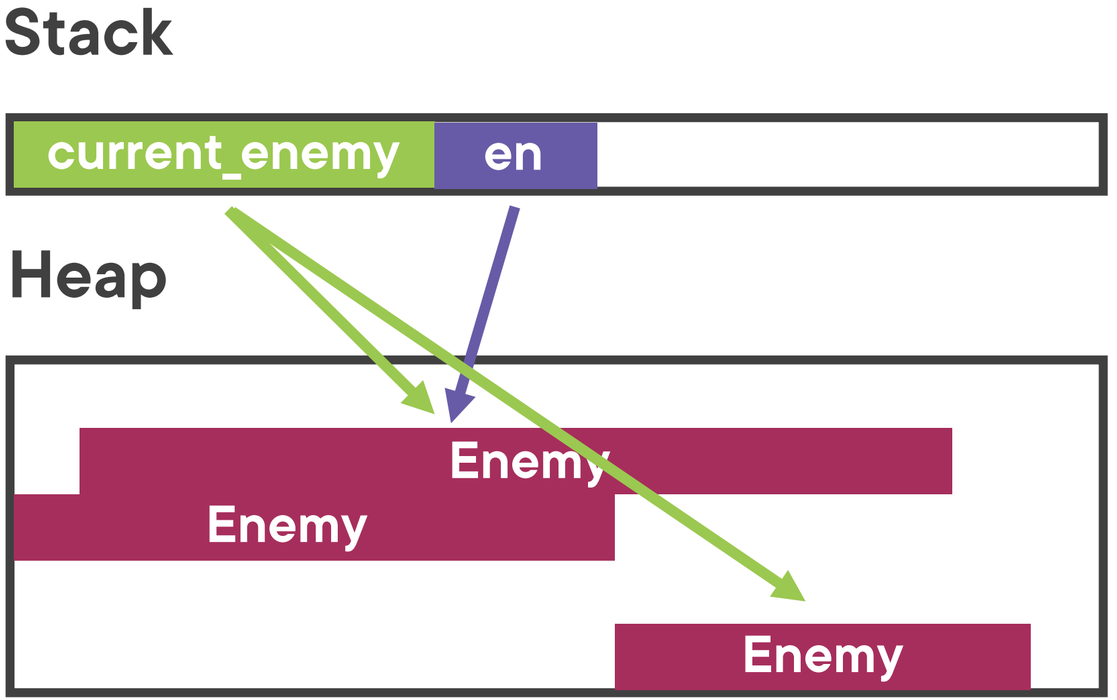
```cpp
#include <iostream>
#include <gameengine>

Enemy *loadEnemy()
{
  Enemy *en = new Enemy();
  (*en).power = random_power();
  return en;
}

int main()
{
  Enemy *current_enemy;
  while (true)
  {
    // Taking in input

    if (enemy_in_sight)
      current_enemy = loadEnemy();

    // Rendering

    if (gameover)
      break;
  }
}
```



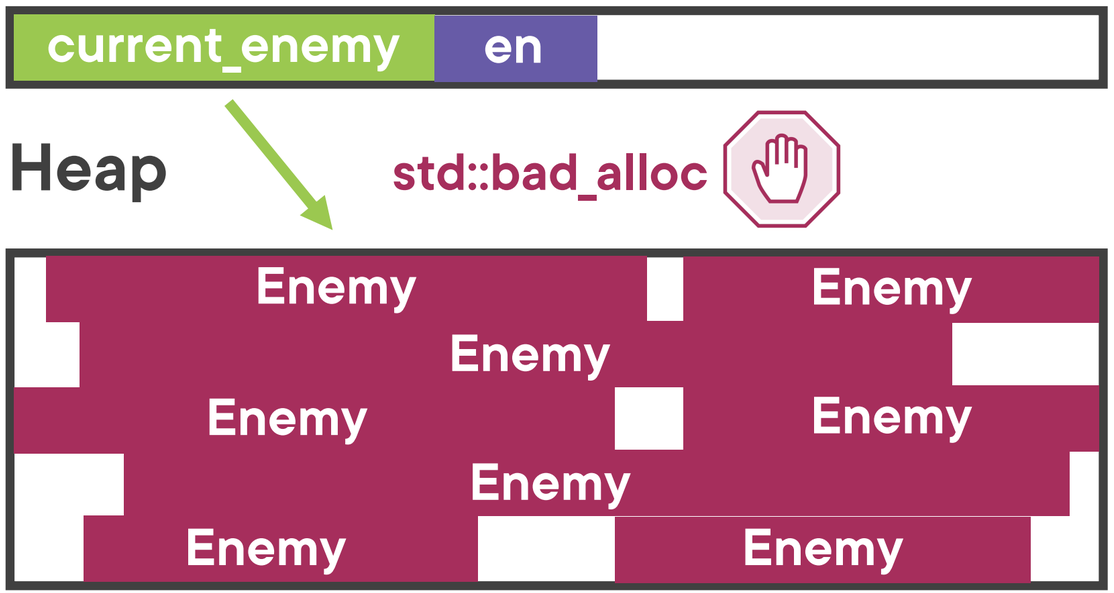
```cpp
#include <iostream>
#include <gameengine>

Enemy *loadEnemy()
{
  Enemy *en = new Enemy();
  (*en).power = random_power();
  return en;
}

int main()
{
  Enemy *current_enemy;
  while (true)
  {
    // Taking in input

    if (enemy_in_sight)
      current_enemy = loadEnemy();

    // Rendering

    if (gameover)
      break;
  }
}
```



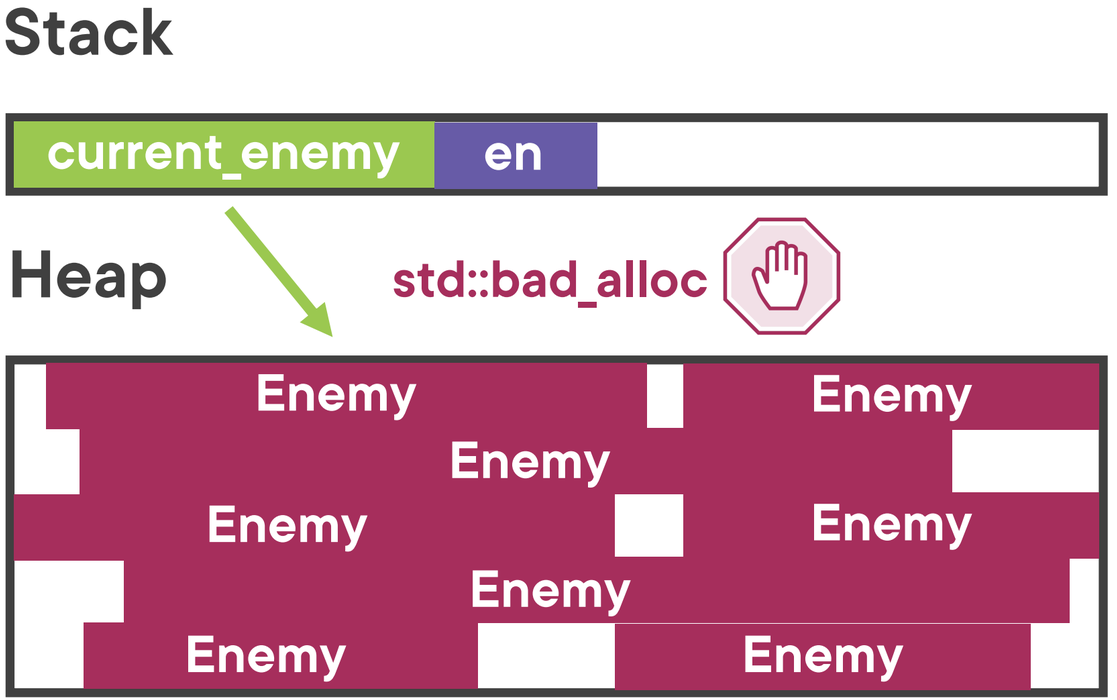
```cpp
#include <iostream>
#include <gameengine>

Enemy *loadEnemy()
{
  Enemy *en = new Enemy();
  (*en).power = random_power();
  return en;
}

int main()
{
  Enemy *current_enemy;
  while (true)
  {
    // Taking in input

    if (enemy_in_sight)
      current_enemy = loadEnemy();

    // Rendering

    if (gameover)
      break;
  }
}
```



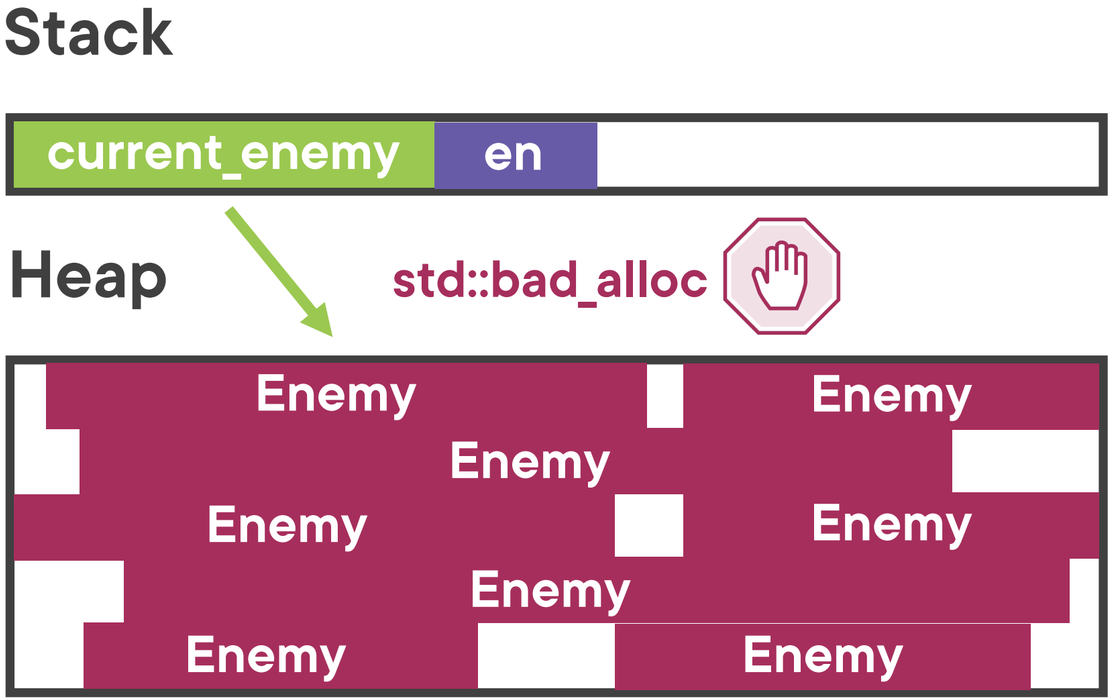
```cpp
#include <iostream>
#include <gameengine>

Enemy *loadEnemy()
{
  Enemy *en = new Enemy();
  (*en).power = random_power();
  return en;
}

int main()
{
  Enemy *current_enemy;
  while (true)
  {
    // Taking in input

    if (enemy_in_sight)
      current_enemy = loadEnemy();

    // Rendering

    if (gameover)
      break;
  }
}
```



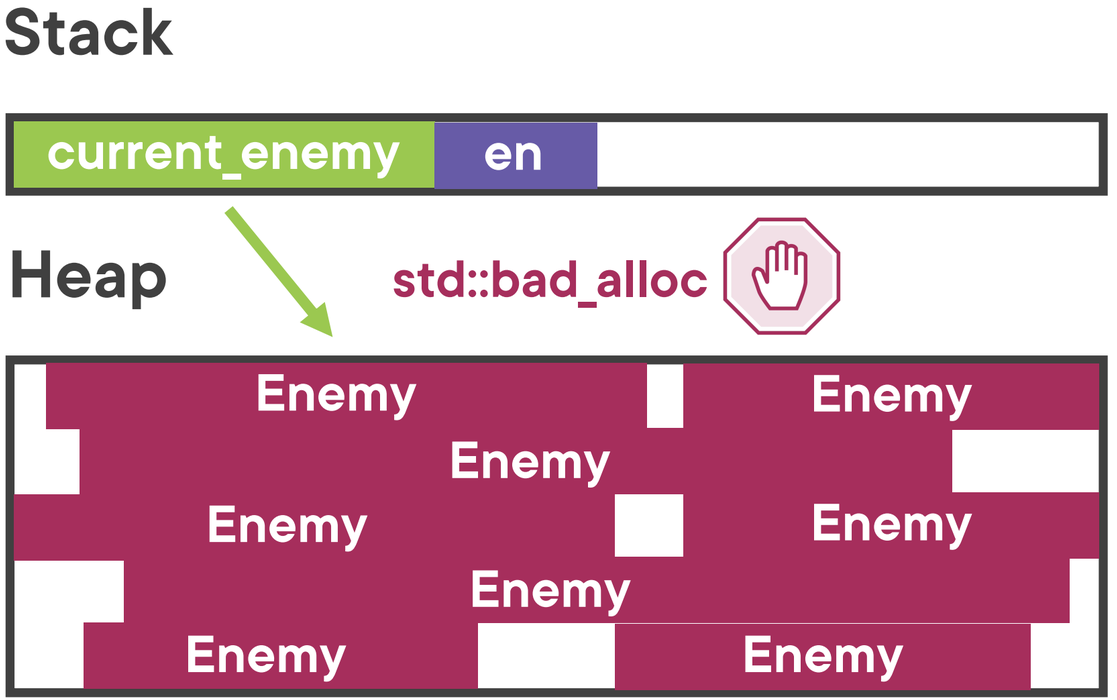
```cpp
#include <iostream>
#include <gameengine>

Enemy *loadEnemy()
{
  Enemy *en = new Enemy();
  (*en).power = random_power();
  return en;
}

int main()
{
  Enemy *current_enemy;
  while (true)
  {
    // Taking in input

    if (enemy_in_sight)
      current_enemy = loadEnemy();

    // Rendering

    if (gameover)
      break;
  }
}
```



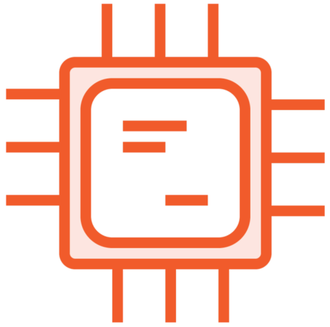
```cpp
#include <iostream>
#include <gameengine>

Enemy *loadEnemy()
{
  Enemy *en = new Enemy();
  (*en).power = random_power();
  return en;
}

int main()
{
  Enemy *current_enemy;
  while (true)
  {
    // Taking in input

    if (enemy_in_sight)
      current_enemy = loadEnemy();

    // Rendering

    if (gameover)
      break;
  }
}
```



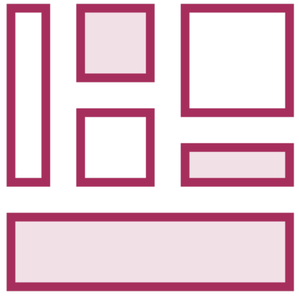
```cpp
#include <iostream>
#include <gameengine>

Enemy *loadEnemy()
{
  Enemy *en = new Enemy();
  (*en).power = random_power();
  return en;
}

int main()
{
  Enemy *current_enemy;
  while (true)
  {
    // Taking in input

    if (enemy_in_sight)
      current_enemy = loadEnemy();

    // Rendering

    if (gameover)
      break;
  }
}
```



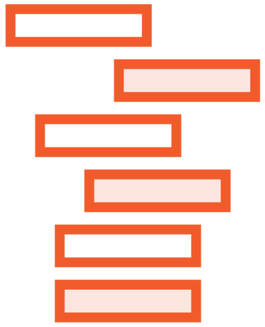
Why Overload new and delete Operators?



Embedded systems

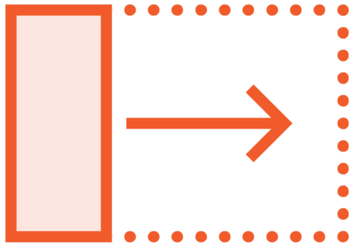


Preallocating memory

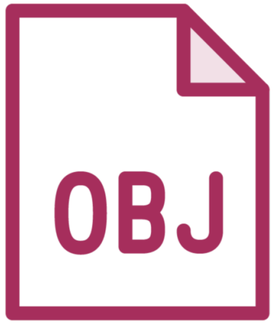


Multiple overloads

Handle exceptions



Reuse or re-allocate memory



Restrict overloads to specific classes

Summary

Up Next:
Using Pointers to Access Array Elements