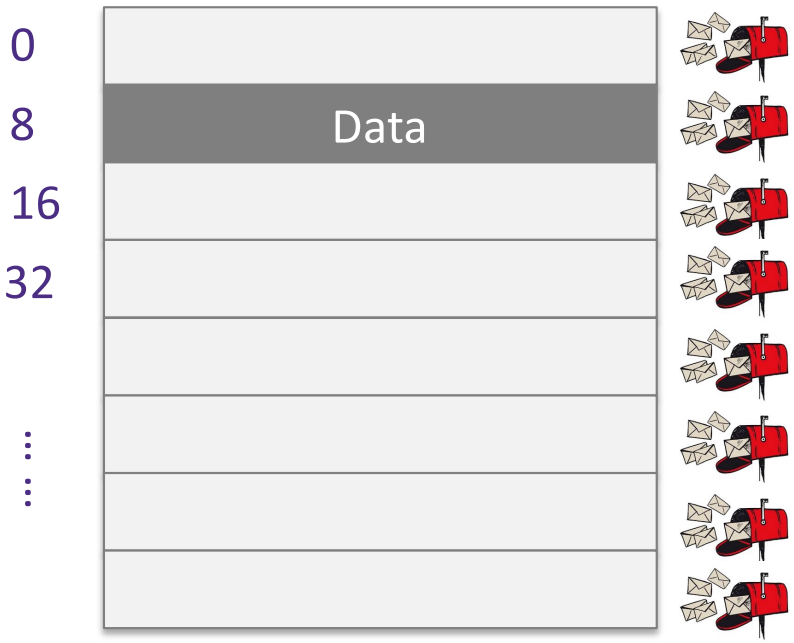
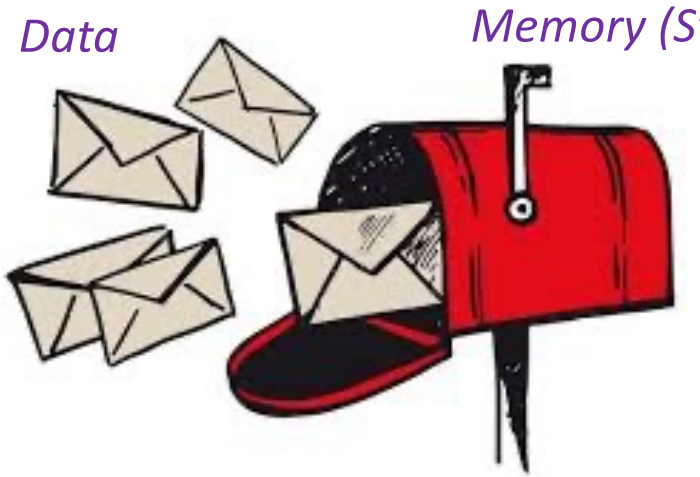
2024 CSS 342 BootUp

UNIVERSITY of WASHINGTON

Memory

Computer Memory





Memory

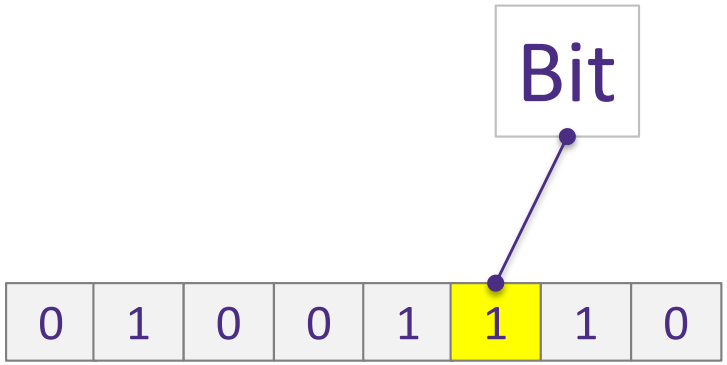
0

Memory

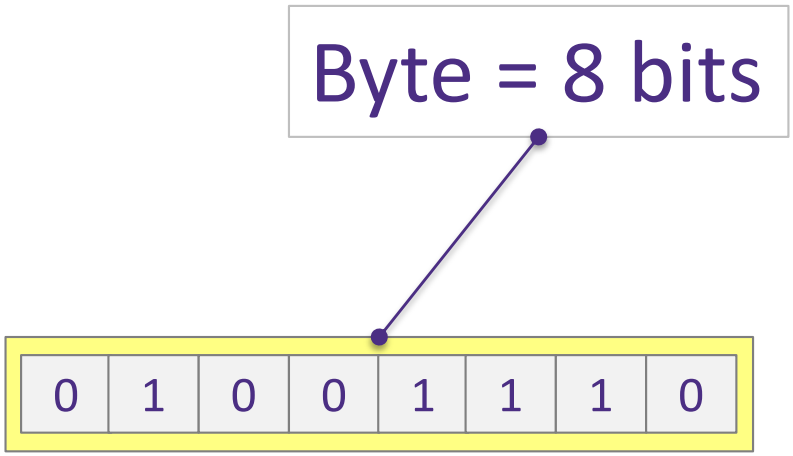
Memory

0	1	0	0	1	1	1	0

Memory



Memory



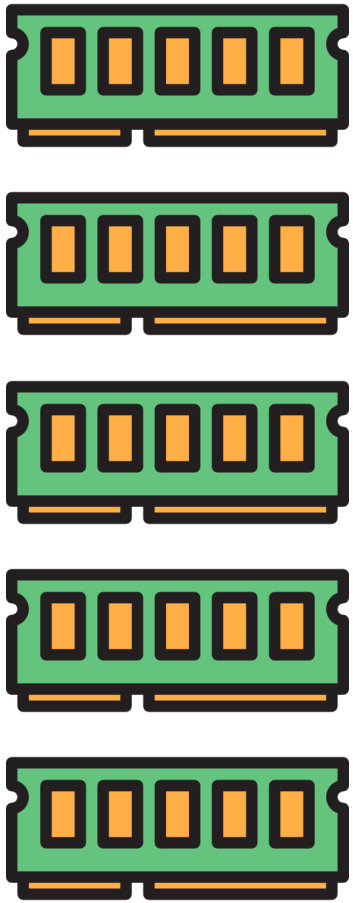
Memory Address

0	0	1	1	0	1	0	0	0
1	0	1	1	0	0	1	0	1
2	0	1	1	0	1	1	0	0
3	0	1	1	0	1	1	0	0
4	0	1	1	0	1	1	1	1

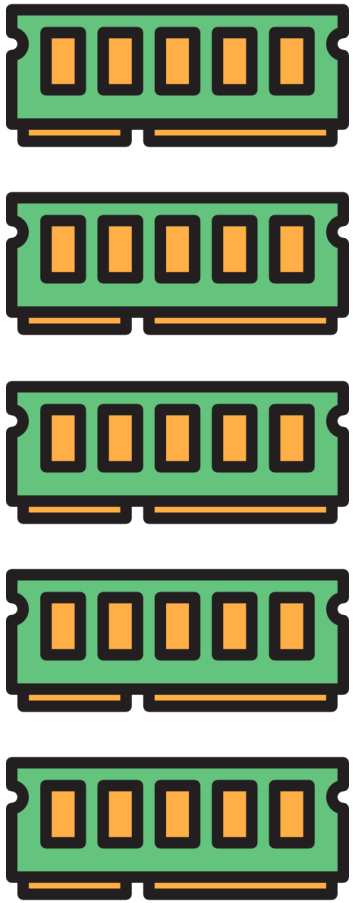
Code

```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```

Memory



Memory

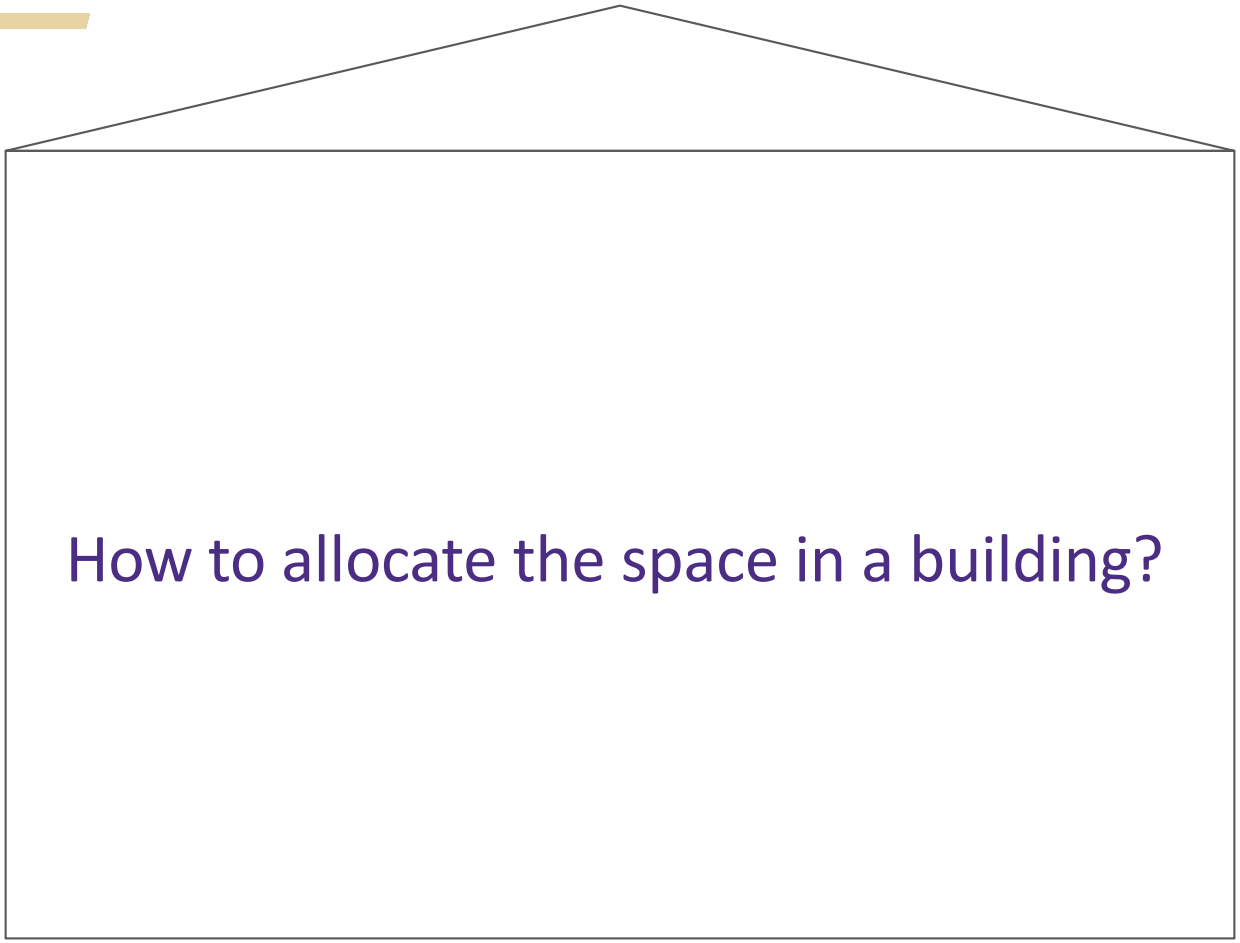


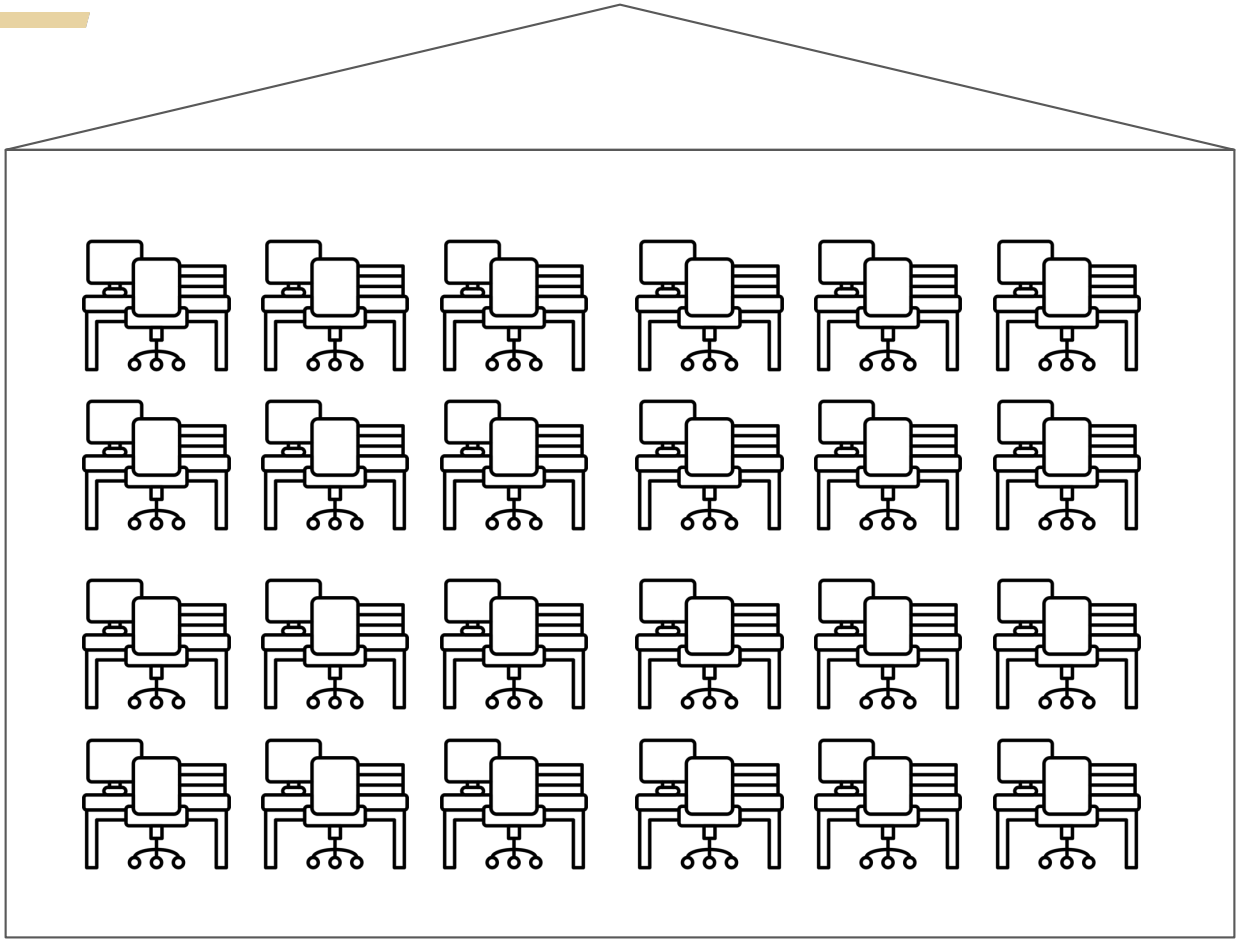
Code

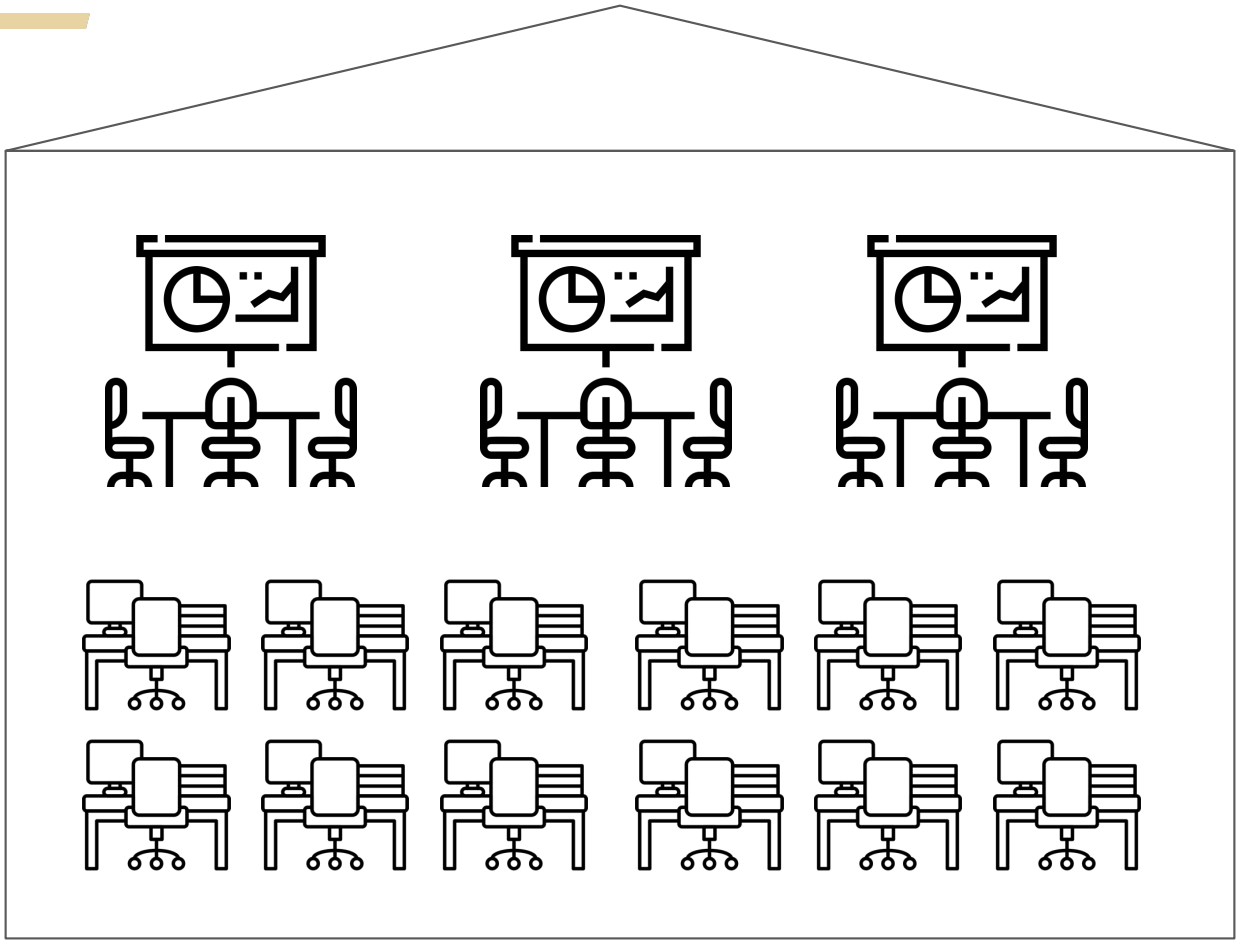
```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```

Not all memory are used equally

Office Space



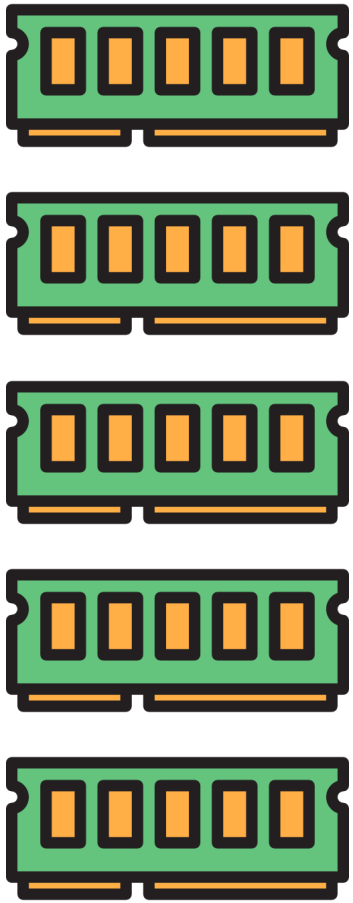




Shared Space

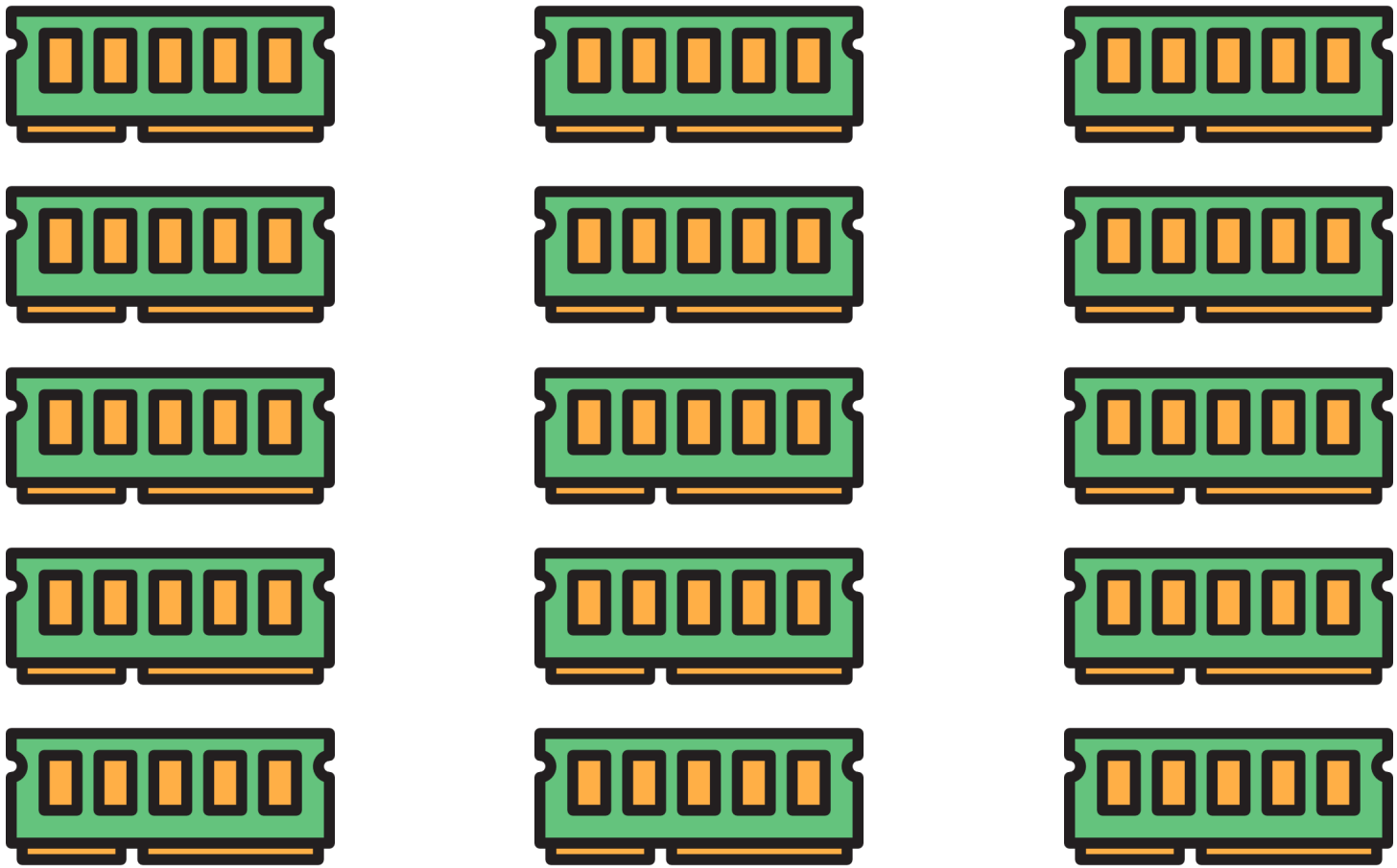
Private Space

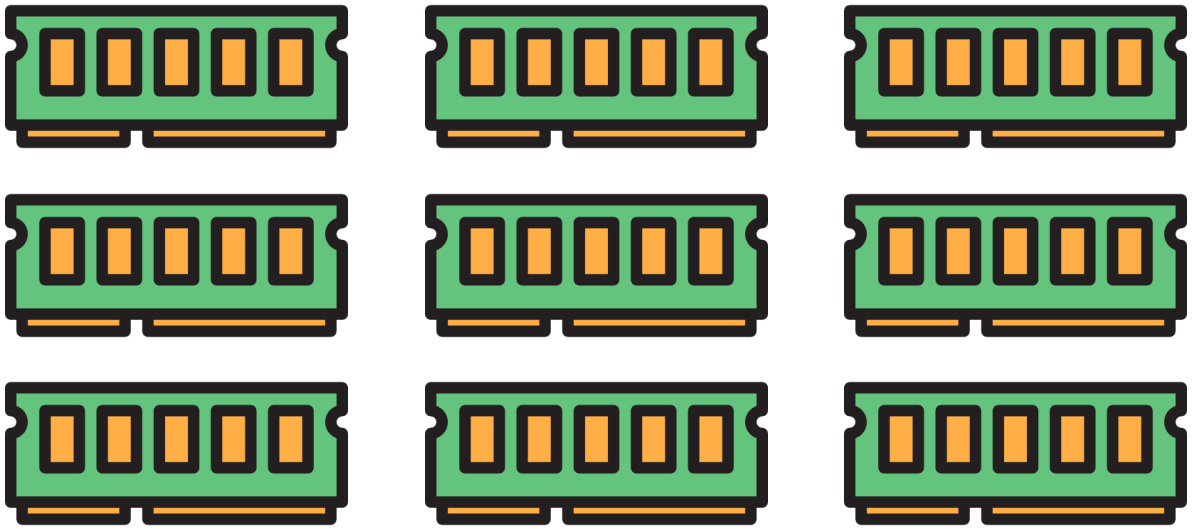
Group 1: Shared By All Program



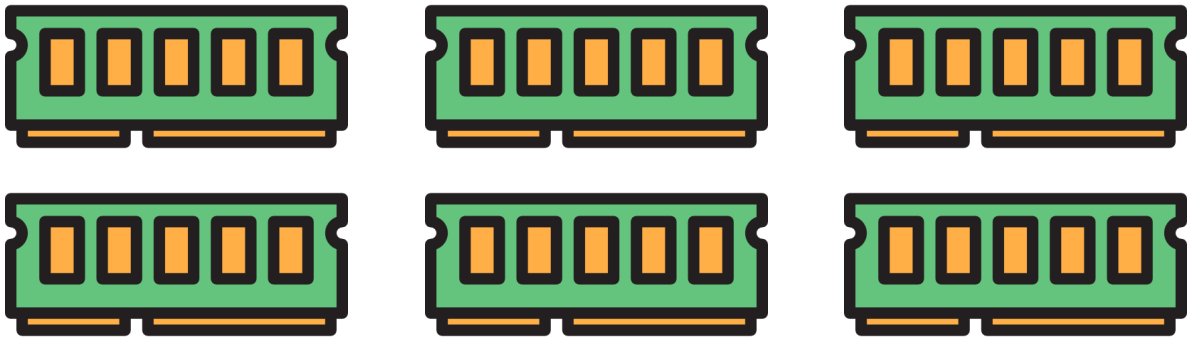
Code

```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```



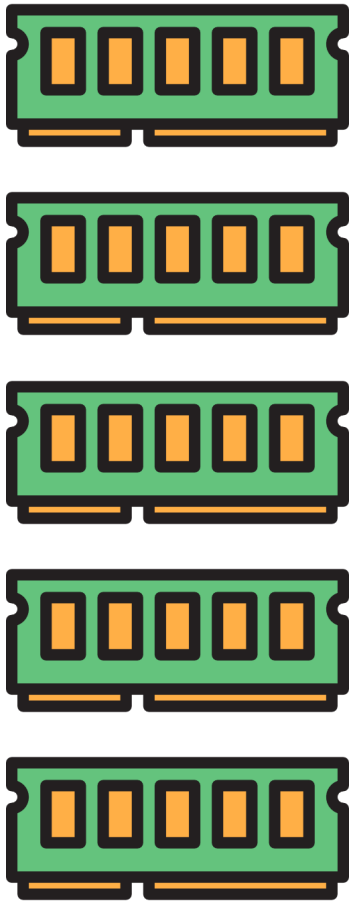


Shared Space



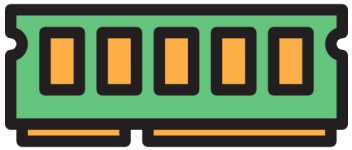
Private Space

Group 1: Shared By All Program



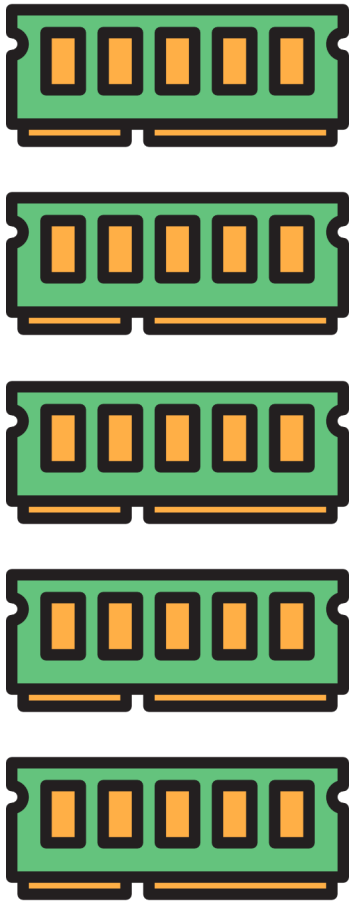
Code

```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```



Group 2: Private for each function call

“Heap Memory”



Code

```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```

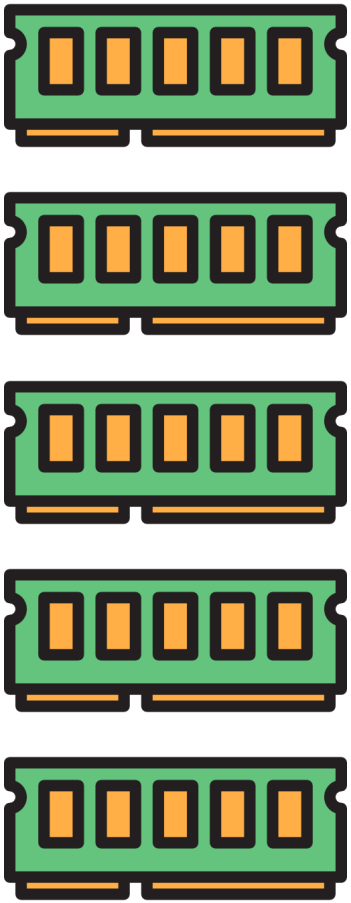
“Stack Memory”

Code

```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```

“Stack”

“Heap”

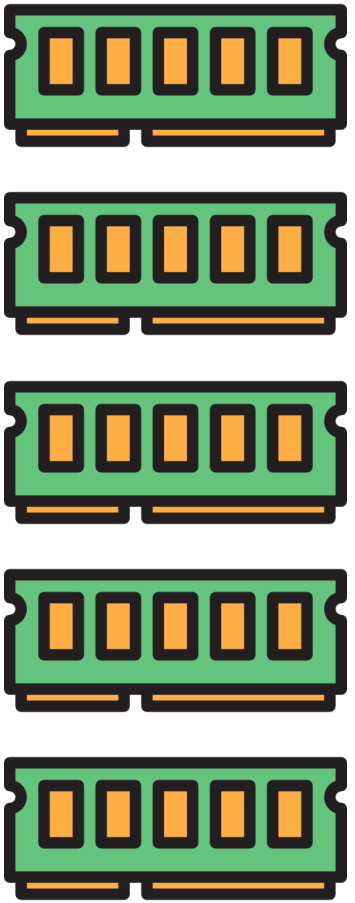


Code

```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```

“Stack”

“Heap”



Cost: 1 Million Dollar

100k/yr

Cost: 1 Million Dollar

100k/yr

Afford a House?

Cost: 1 Million Dollar

Cost: 1 Million Dollar

BORROW

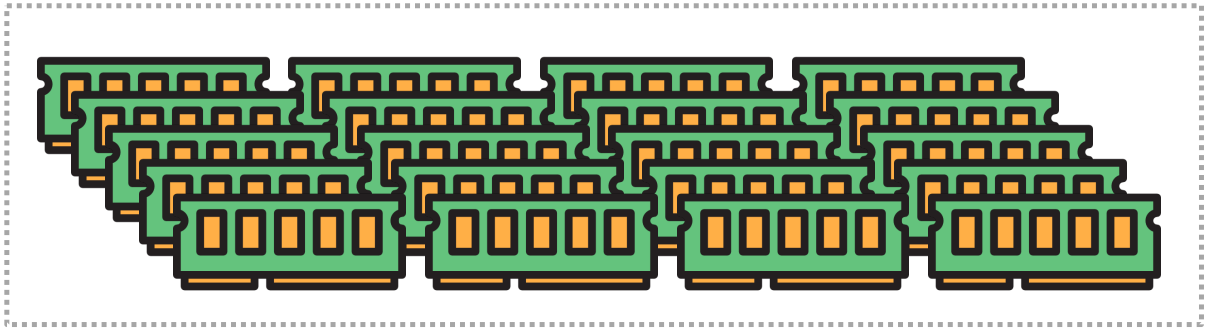
Loan

Cost: 1 Million Dollar

Buy A House

Cost: 1 Million Dollar

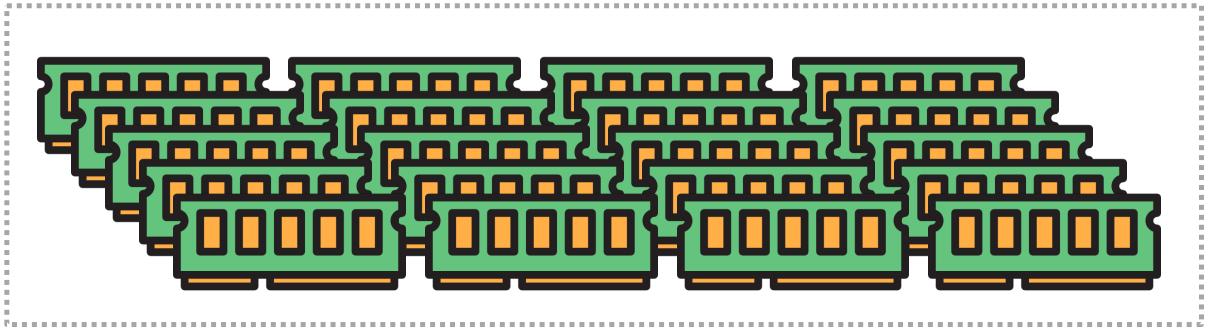
RETURN



“bank”

“You”

```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```



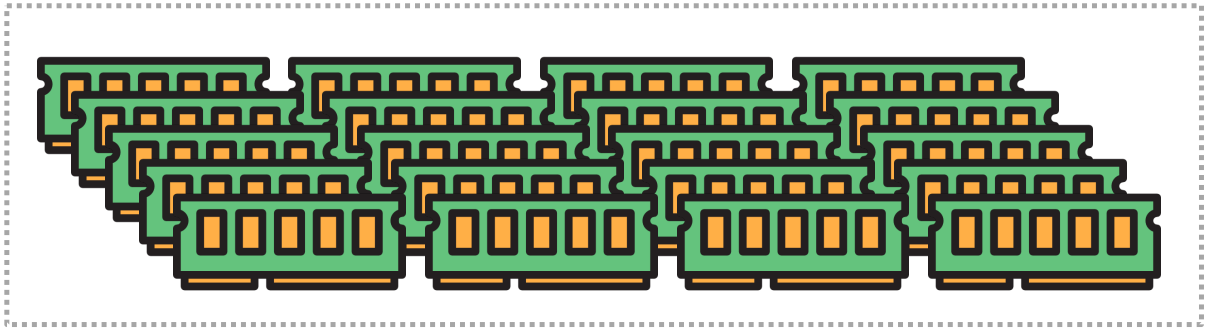
```c
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```

"***stack*** memory"

A little pool of memory for each function call to use

“***heap*** memory”

A large pool of memory to use



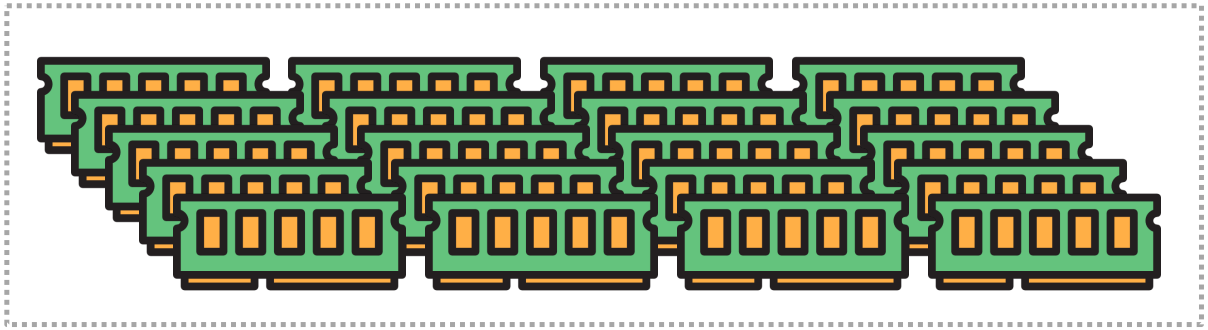
```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```

“***stack*** memory”

A little pool of memory for each function call to use

"***heap*** memory"

A large pool of memory to use



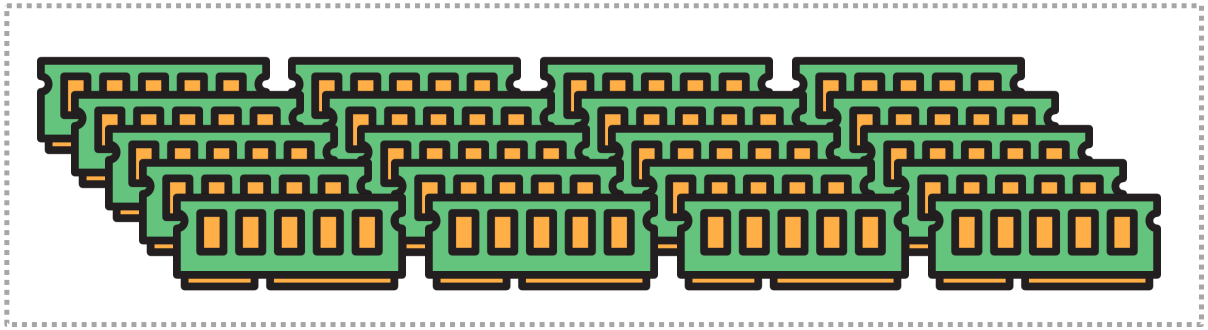
```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int *L = new int[n1];
    int *R = new int[n2];

    {...}

    // Free the memory of temporary arrays
    delete[] L;
    delete[] R;
}
```

"***stack*** memory"

A little pool of memory for each function call to use

"***heap*** memory"

A large pool of memory to use



```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int *L = new int[n1];
    int *R = new int[n2];

    {...}

    // Free the memory of temporary arrays
    delete[] L;
    delete[] R;
}
```

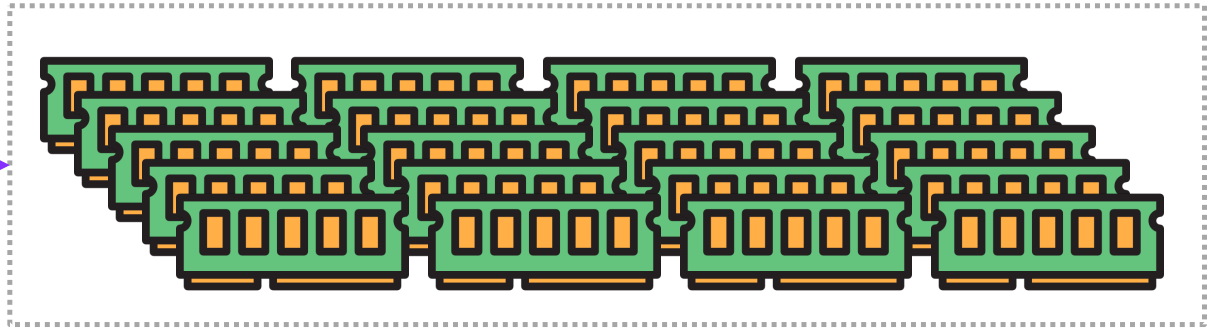
"***stack*** memory"

A little pool of memory for each function call to use

BORROW

"***heap*** memory"

A large pool of memory to use



*I need to "**borrow**" memory for (n1 + n2) integers*

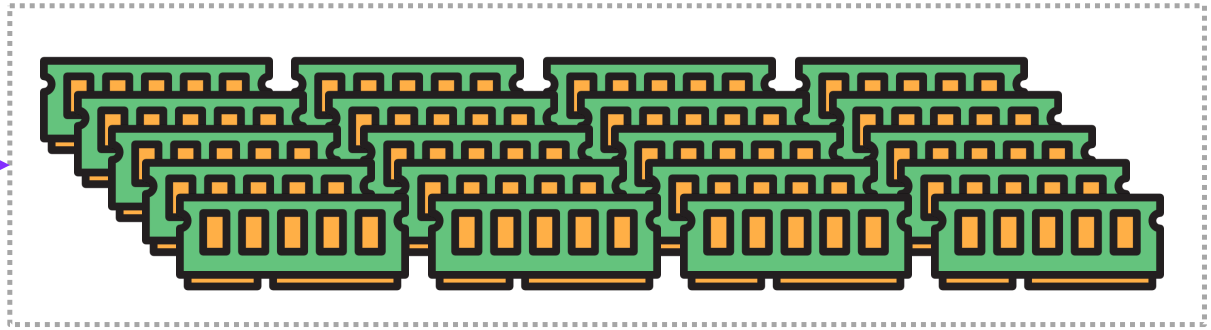
```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int *L = new int[n1];
    int *R = new int[n2];

    {...}

    // Free the memory of temporary arrays
    delete[] L;
    delete[] R;
}
```

"***stack*** memory"

RETURN

"***heap*** memory"

A large pool of memory to use



*"Done with the memory. **Return** it to the heap so others can use it."*

```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int *L = new int[n1];
    int *R = new int[n2];

    {...}

    // Free the memory of temporary arrays
    delete[] L;
    delete[] R;
}
```

"***stack*** memory"

```
➜  ~ ulimit -s
8176
```

Stack Memory: **8MB**/program(process)

8MB for all functions call of a thread

Stack Memory: **8MB**/program(process)

8MB for all functions call of a thread

Heap Memory: ?MB

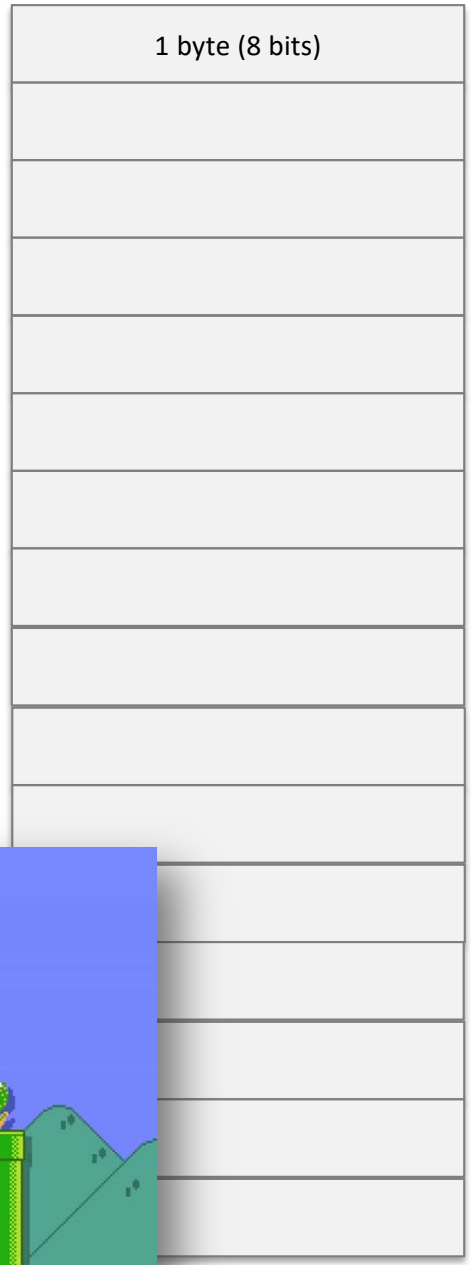
8-bit to store each data (byte)

0000 0000 0000 0000
address

8-bit to store each data (byte)

16-bit to represent memory address

0000 0000 0000 0000



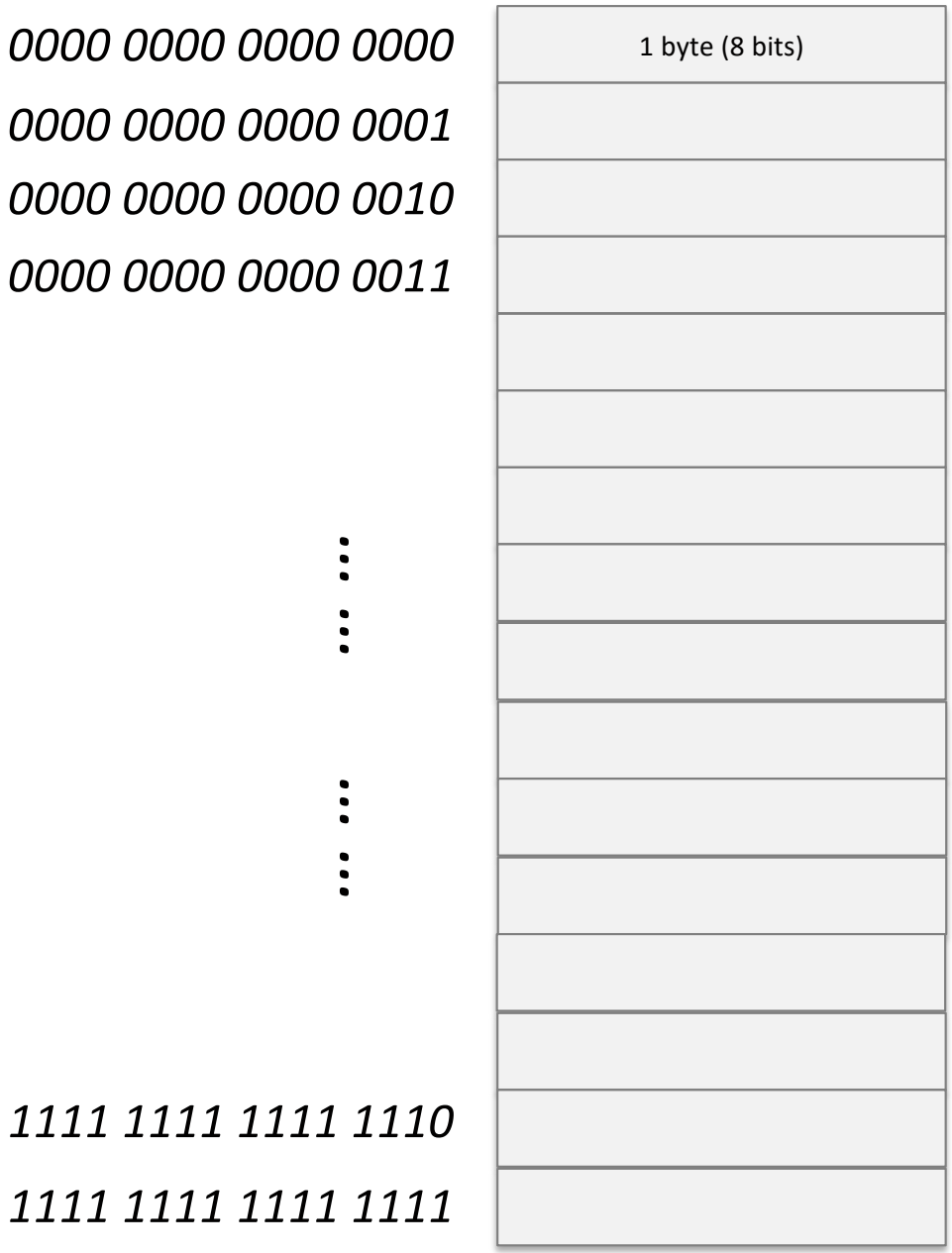
8-bit to store each data (byte)

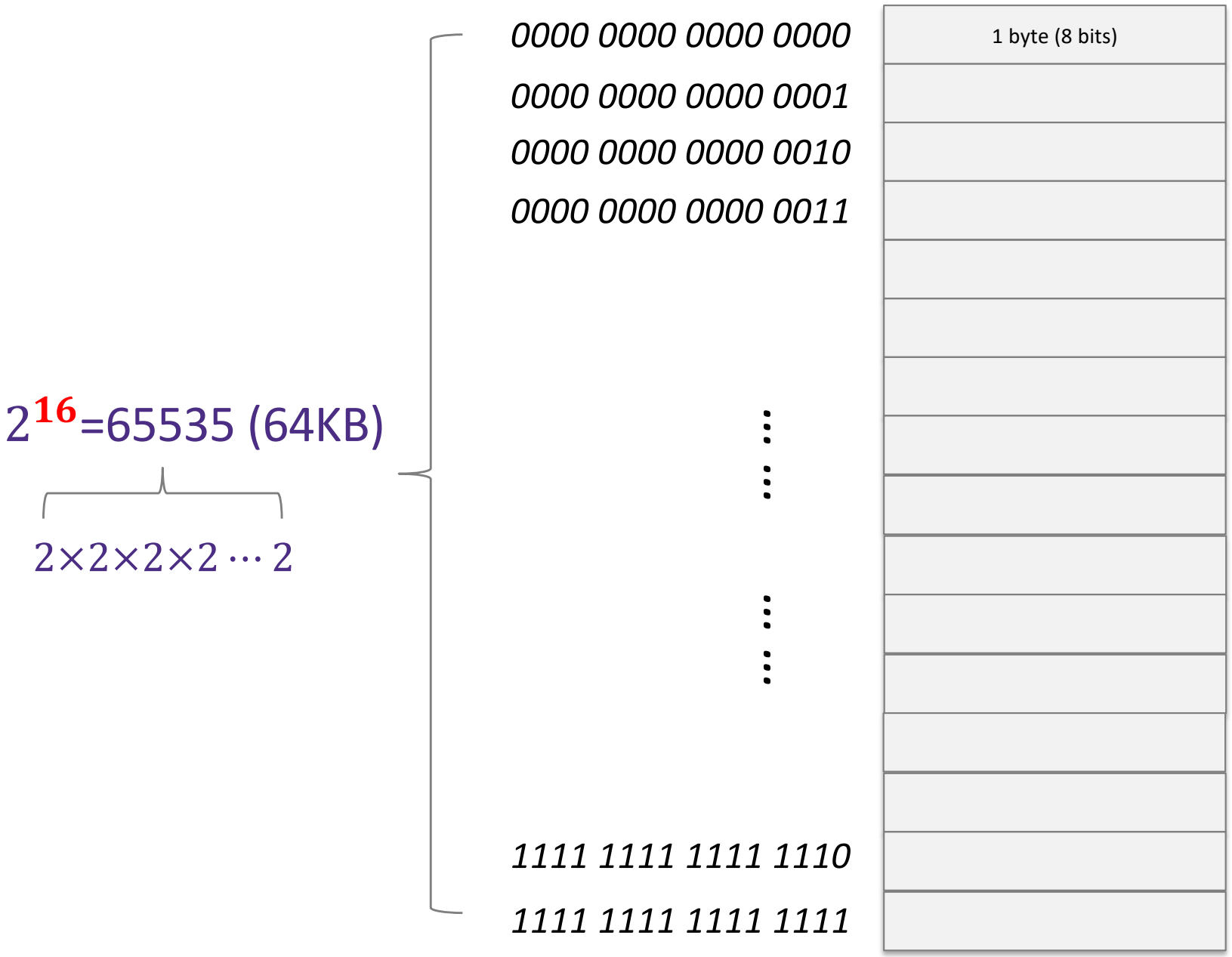
16-bit to represent memory address

0000 0000 0000 0000

0000 0000 0000 0000

0000 0000 0000 0001


0000 0000 0000 0000
0000 0000 0000 0001
0000 0000 0000 0010
0000 0000 0000 0011
1111 1111 1111 1110
1111 1111 1111 1111
1 byte (8 bits)


0000 0000 0000 0000
0000 0000 0000 0001
0000 0000 0000 0010
0000 0000 0000 0011
1111 1111 1111 1110
1111 1111 1111 1111
1 byte (8 bits)
2^{16}=65535 (64KB)
$2\times2\times2\times2\cdots2$

64KB

64KB

256 pixels wide
× 240 pixels tall

61,440 pixels
per screen
÷ 4 pixels per byte

15 KB
per screen

To store one screen, it takes 15KB already

https://youtu.be/TPbroUDHG0s?si=wbuJeZbVah0E6YAj

1 byte (8 bits)

256 pixels wide
× 240 pixels tall

61,440 pixels
per screen
÷ 4 pixels per byte

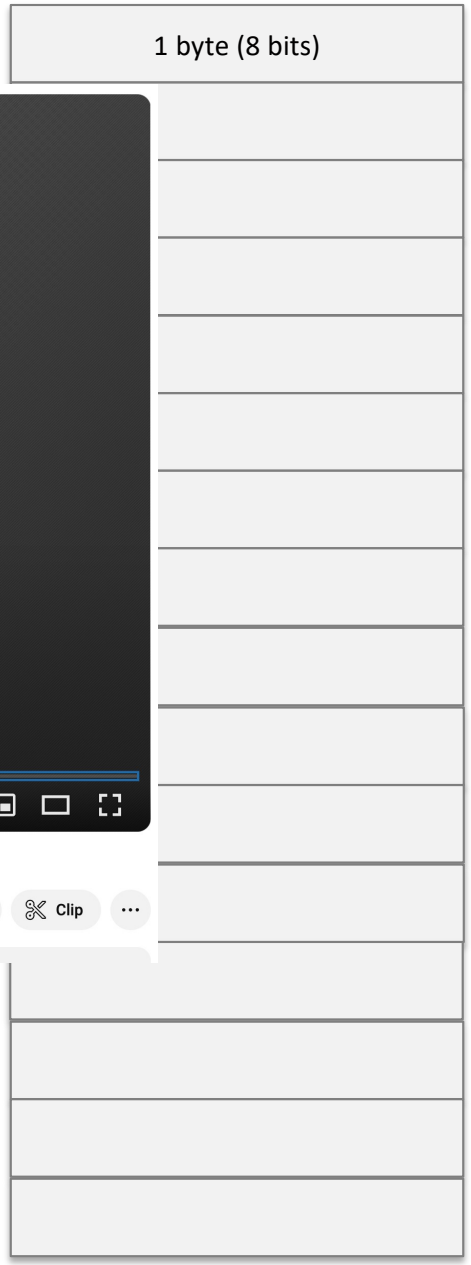
15 KB
per screen

How we fit an NES game into 40 Kilobytes

Morphcat Games
52K subscribers
Subscribe

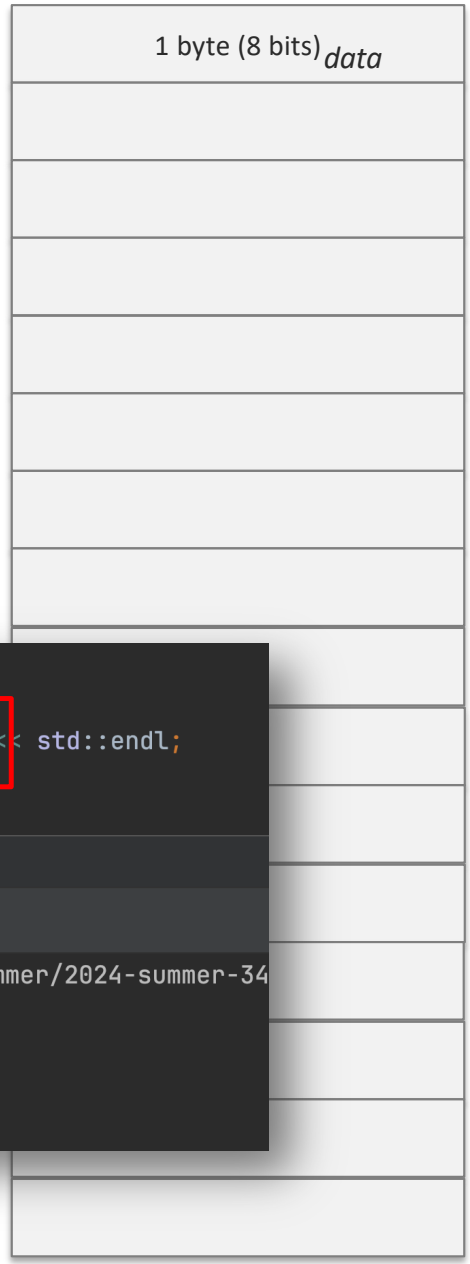
161K | Share | Download | Clip

https://youtu.be/ZWQ0591PAxM?si=MHAJTk384eJG5Fbg



0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

address



8-bit: 64K bytes

What about 64-bit address?

```
void foo() {
    std::cout << sizeof(int *) << std::endl;
}
```

```
my_executable
/Users/pengdu/teaching/2024/342-Summer/2024-summer-34
8

Process finished with exit code 0
```

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

address

8-bit: 64K bytes

What about 64-bit address?

2^{64} byte = ? GB

$$\frac{2^{64}}{2^{(10+10+10)}} = 2^{34}GB$$

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

address

8-bit: 64K bytes

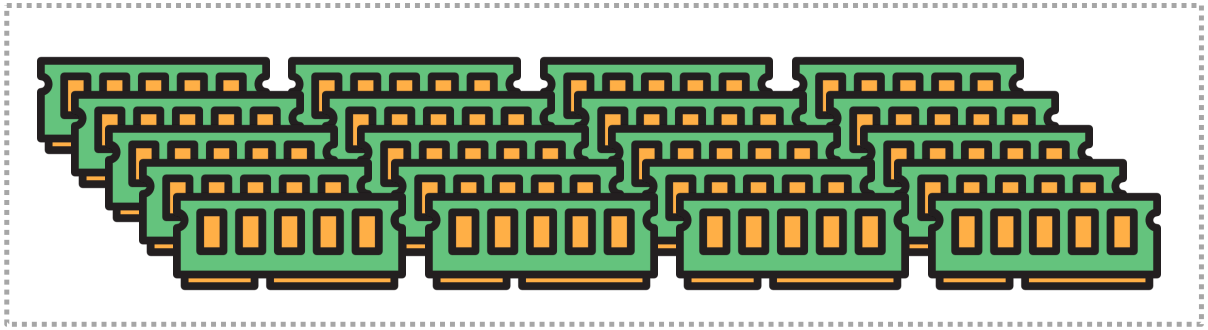
What about 64-bit address?

2^{64} byte = ? GB

$$\frac{2^{64}}{2^{(10+10+10)}} = 2^{34}GB$$

Max heap memory is limited by your hardware and OS

"***heap*** memory", GBs for all programs



```
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int *L = new int[n1];
    int *R = new int[n2];

    {...}

    // Free the memory of temporary arrays
    delete[] L;
    delete[] R;
}
```

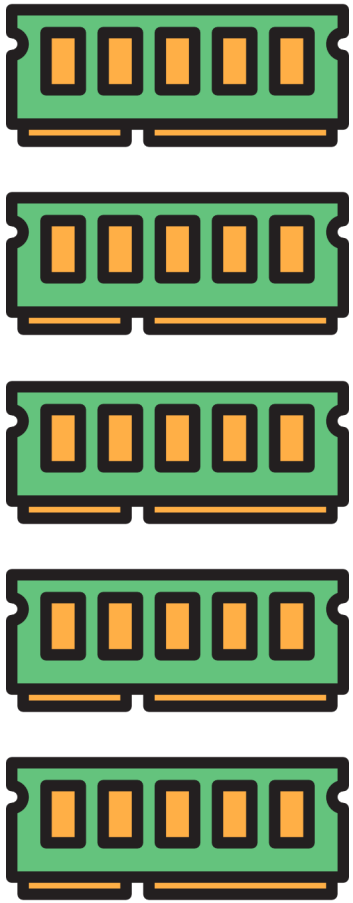
"***stack*** memory", *8MB/function call*

Code

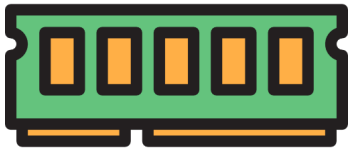
```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c : (c + 1);
}
```

Stack

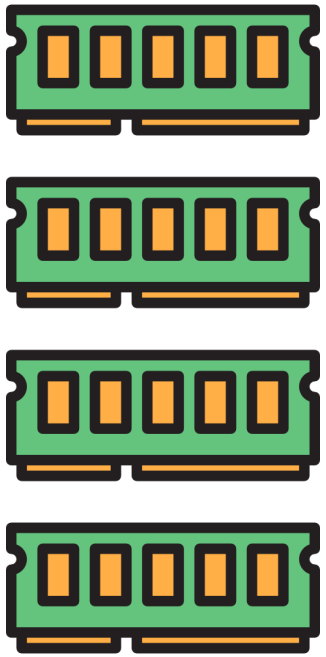
Heap



Stack



Heap



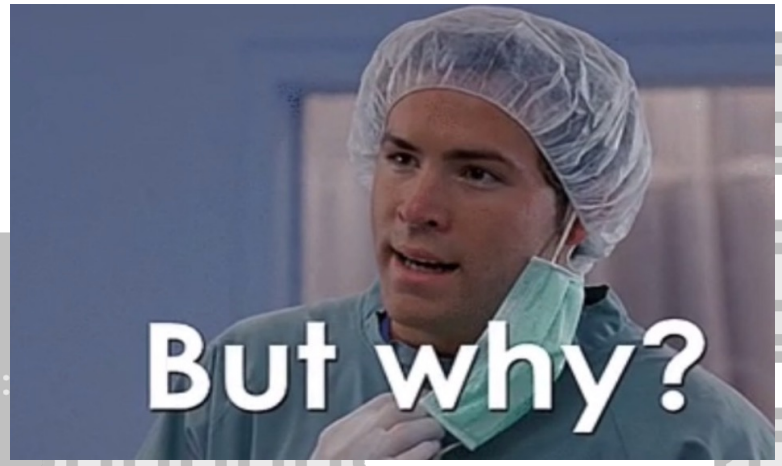
```
void hello() {
    char *word = "hello";
    std::cout << word << std::endl;
}
```

Is the variable *word* allocated on stack memory or heap memory?

Heap

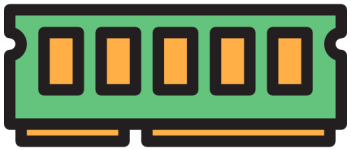
Code

```
int foo(int a, int b) {
    int c = a + b;
    return c % 2 == 0 ? c :
}
```



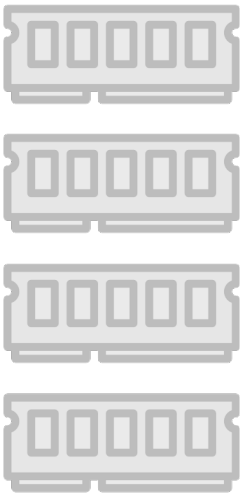
Stack

Stack



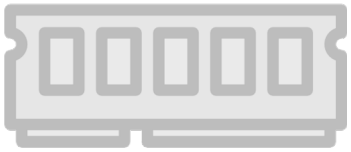
- Static allocation
- Automatically managed (Use and discard)
- Faster operation
- Limited size
- Private to each program (thread)
- Used for local variables, function calls

Heap



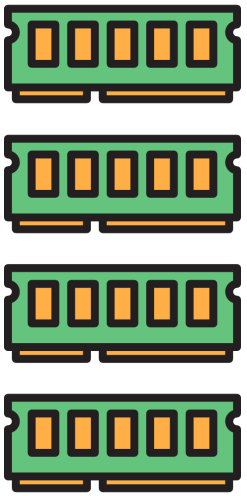
- Dynamic allocation
- Manually managed (Use and return)
- Slower operation
- ”Unlimited” size
- Shared among all programs
- Large and long-lived data structures (such as linked list, trees)

Stack



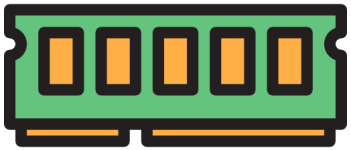
- Static allocation
- Automatically managed (Use and discard)
- Faster operation
- Limited size
- Private to each program (thread)
- Used for local variables, function calls

Heap



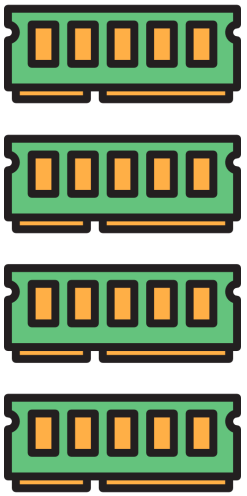
- Dynamic allocation
- Manually managed (Use and return)
- Slower operation
- ”Unlimited” size
- Shared among all programs
- Large and long-lived data structures (such as linked list, trees)

Stack



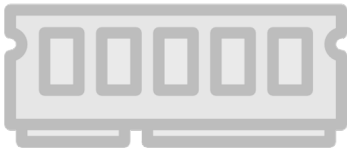
- Static allocation
- Automatically managed (Use and discard)
- Faster operation
- Limited size
- Private to each program (thread)
- Used for local variables, function calls

Heap

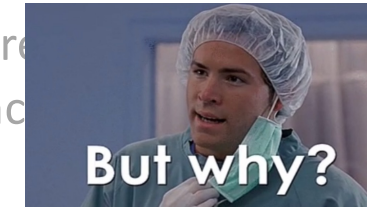


- Dynamic allocation
- Manually managed (Use and return)
- Slower operation
- ”Unlimited” size
- Shared among all programs
- Large and long-lived data structures (such as linked list, trees)

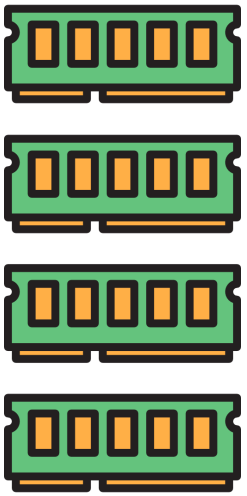
Stack



- Static allocation
- Automatically managed (Use and discard)
- Faster operation
- Limited size
- Private to each program (thr
- Used for local variables, func



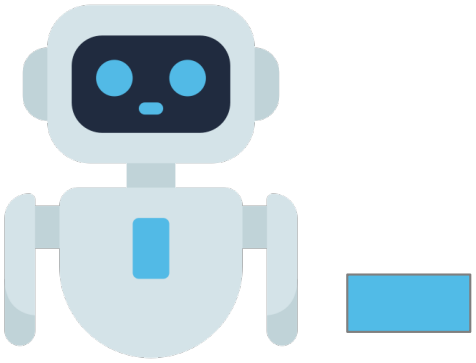
Heap

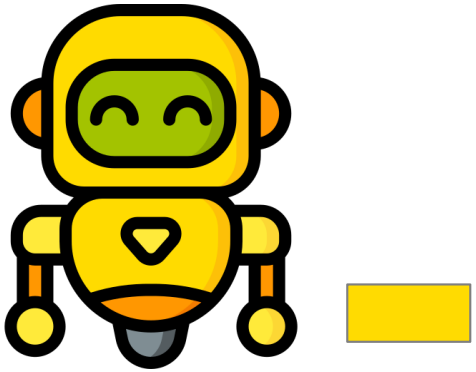


- Dynamic allocation
- Manually managed (Use and return)
- Slower operation

"Unlimited" size
Shared among all programs
Large and long-lived data structures
(such as linked list, trees)

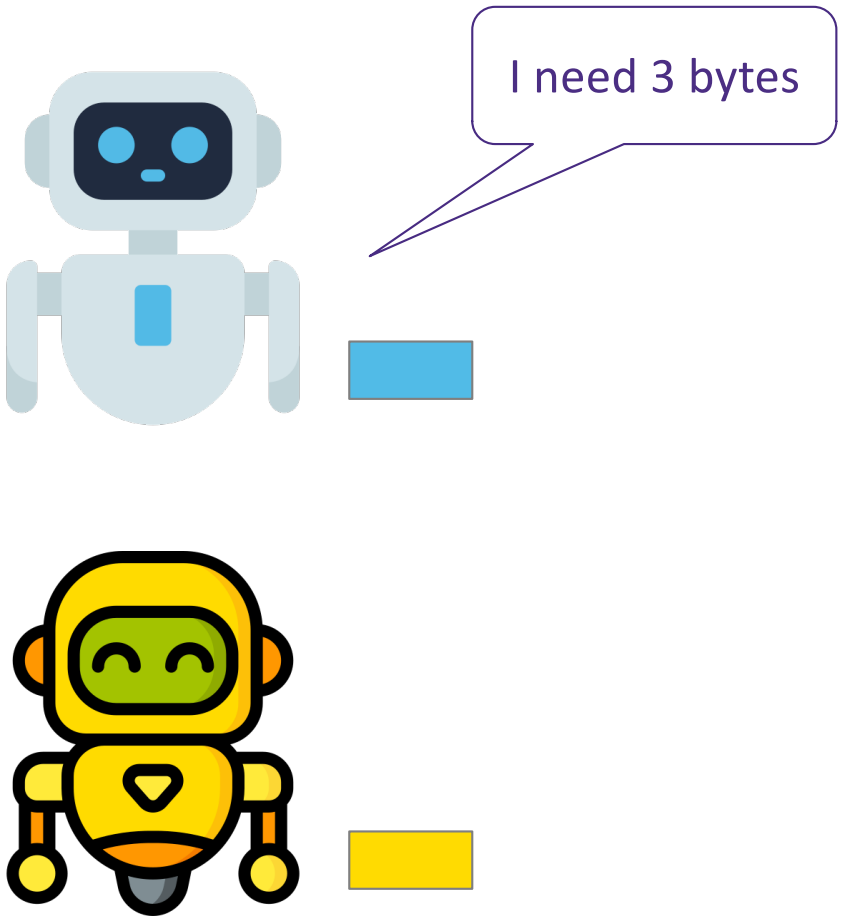
Shared by all program



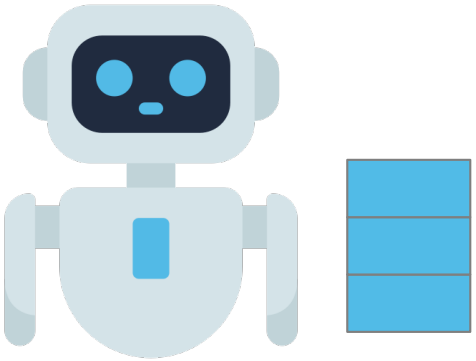


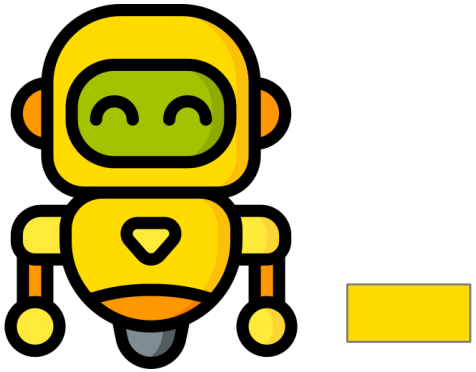
1 byte (8 bits)

Shared by all program

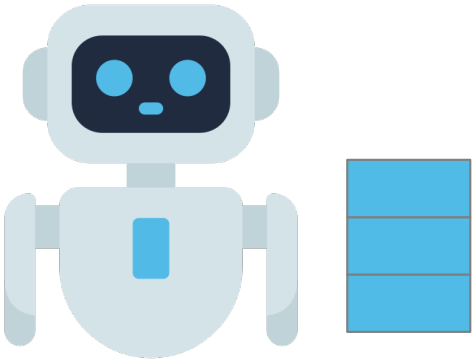


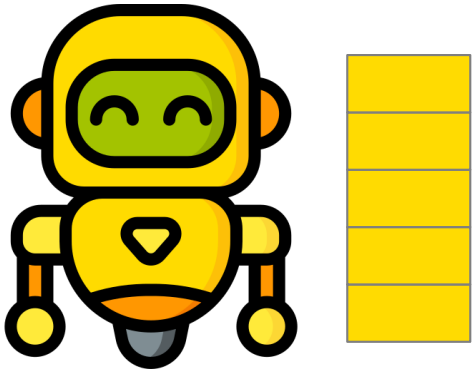
Shared by all program

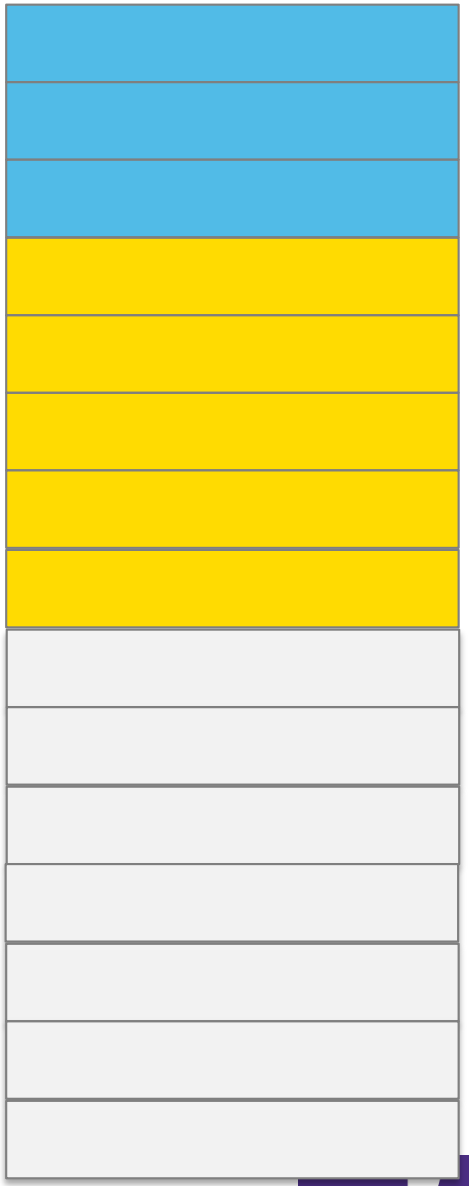




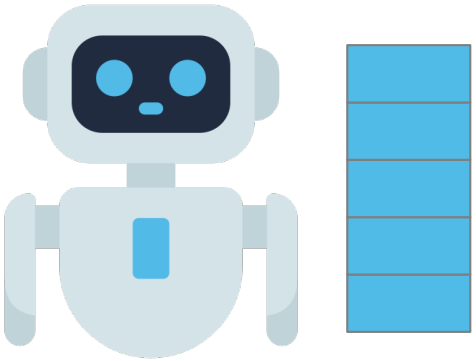
Shared by all program

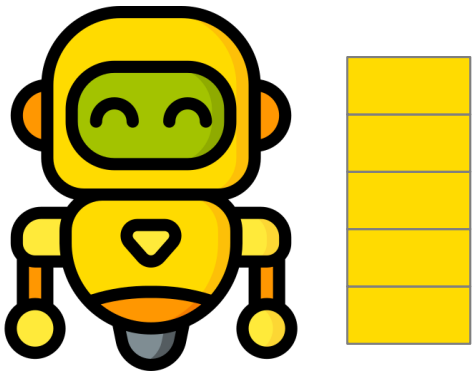


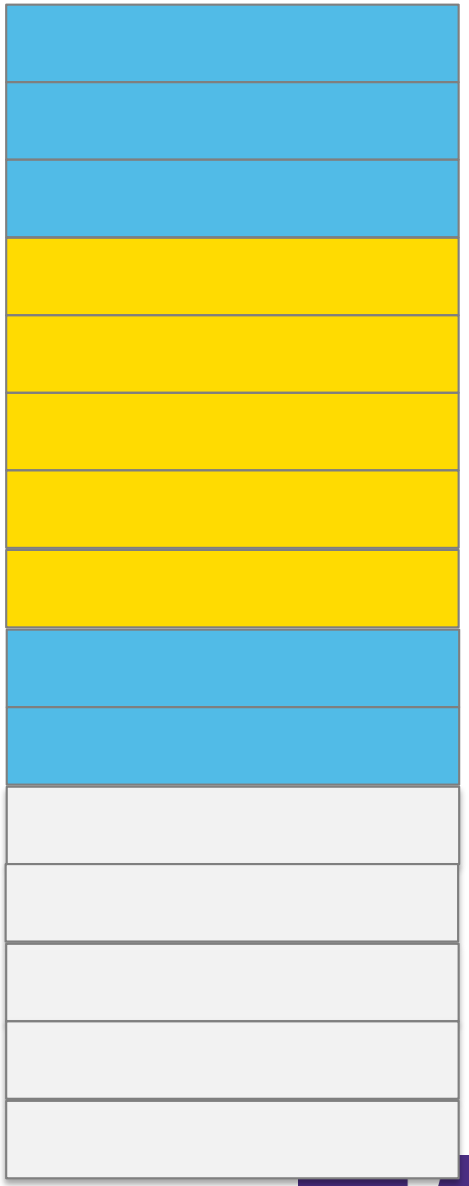




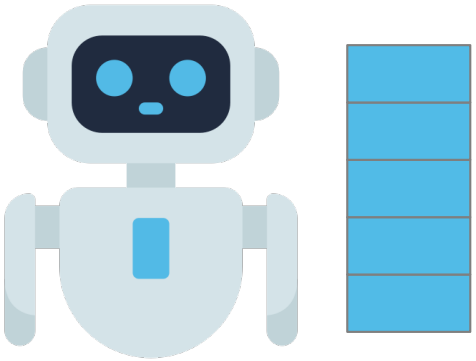
Shared by all program

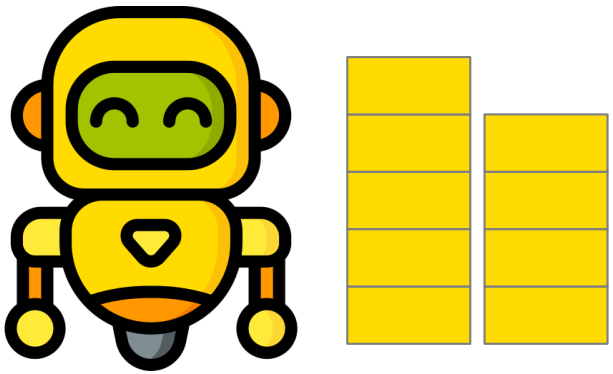


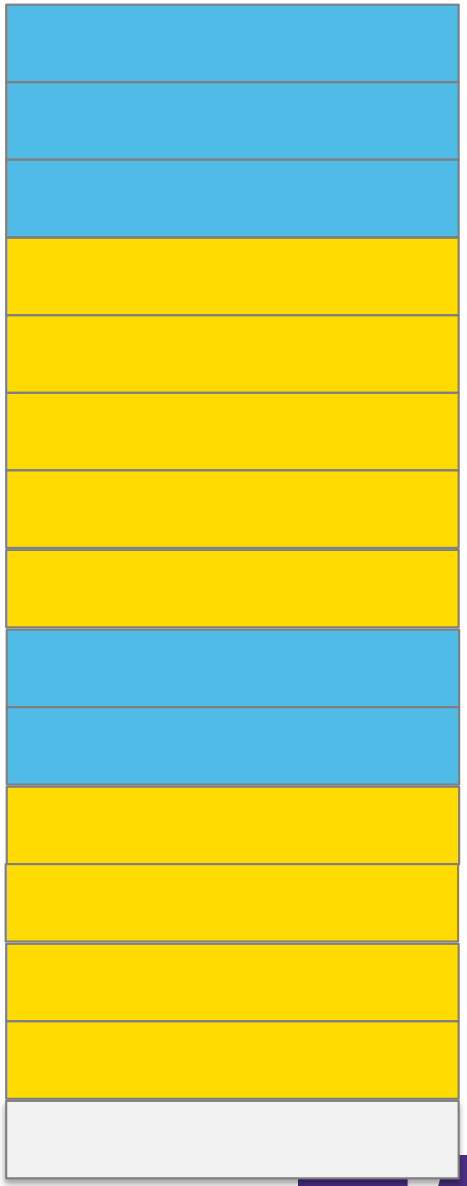




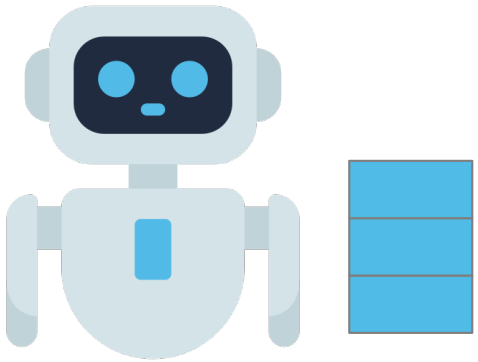
Shared by all program

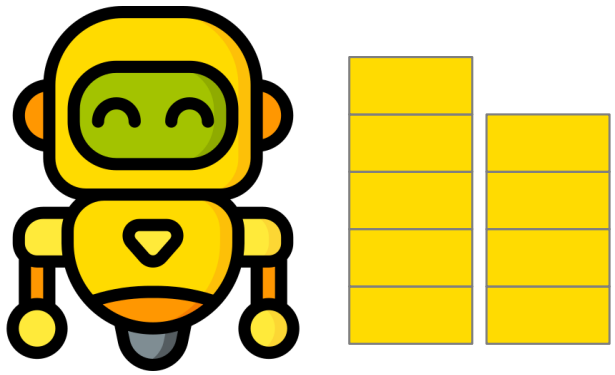


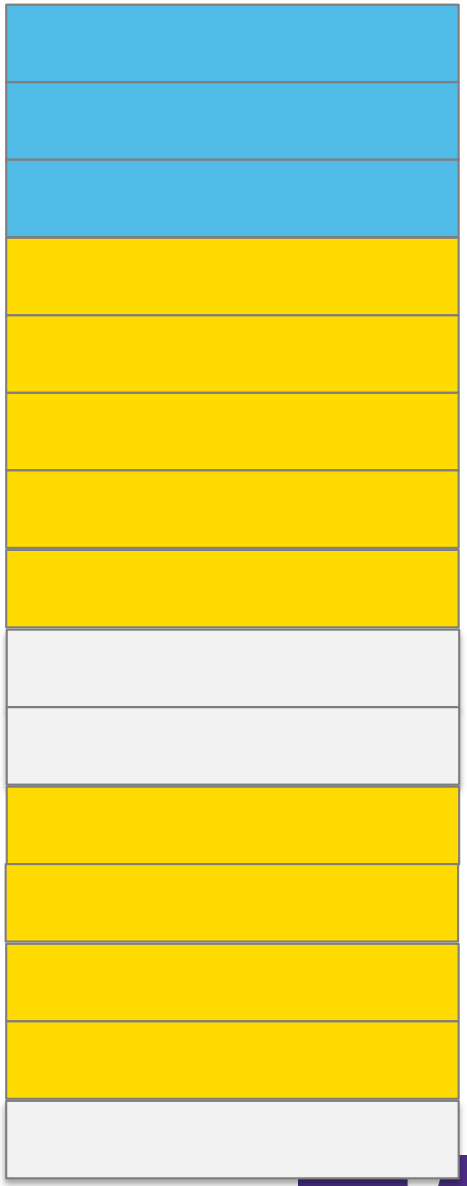




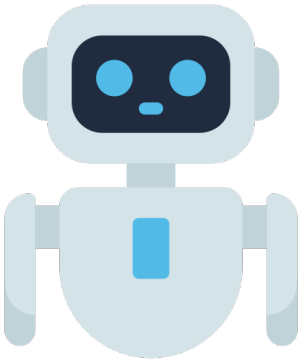
Shared by all program

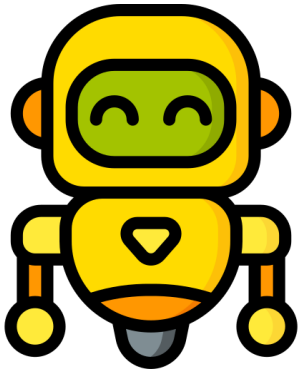


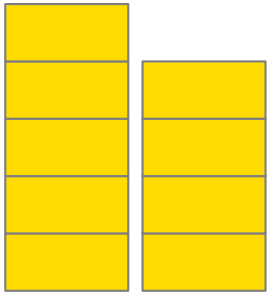


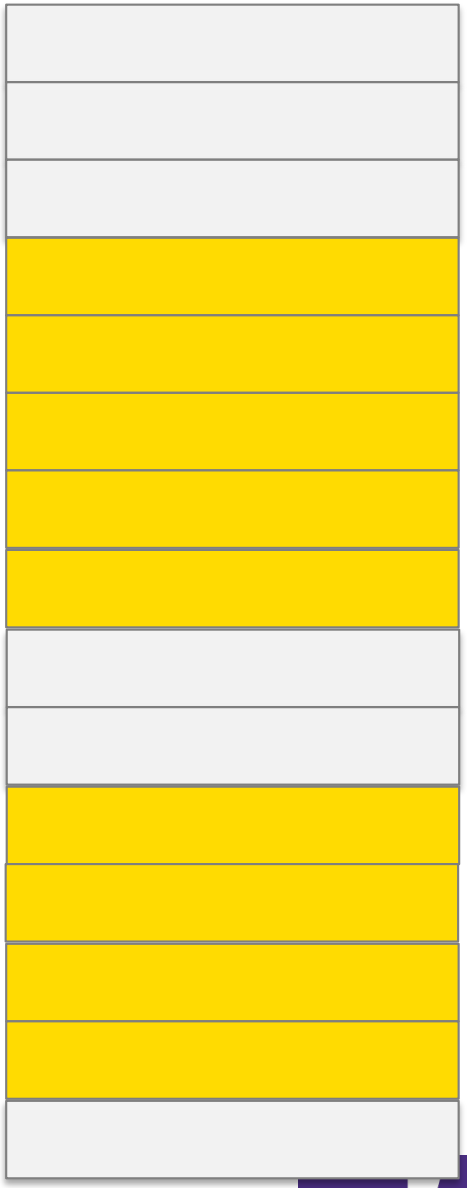


Shared by all program

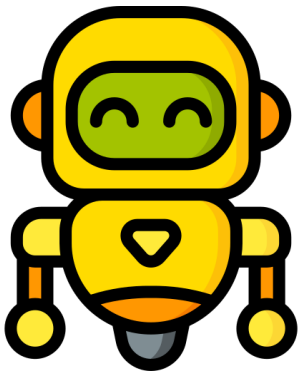


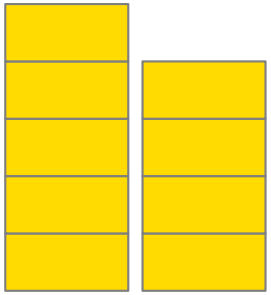


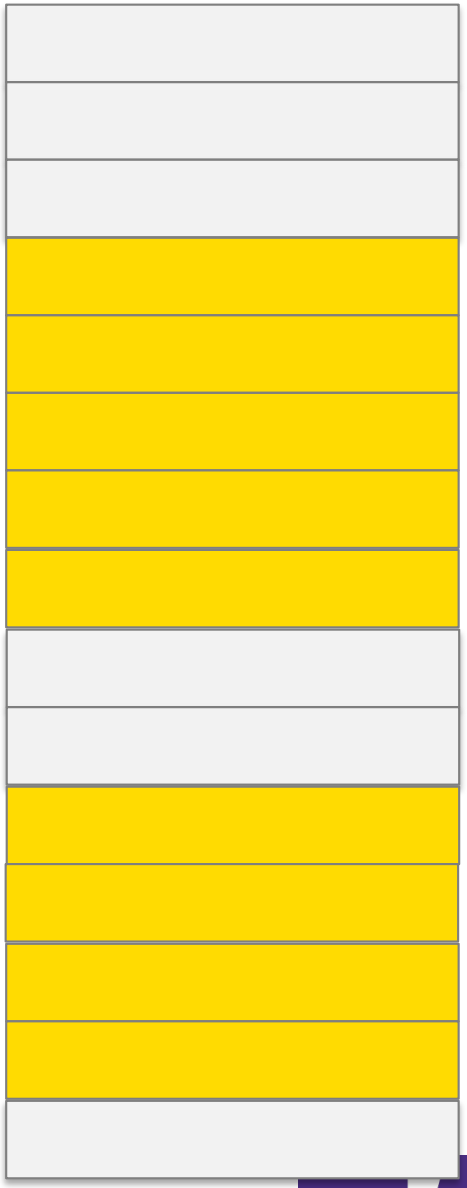




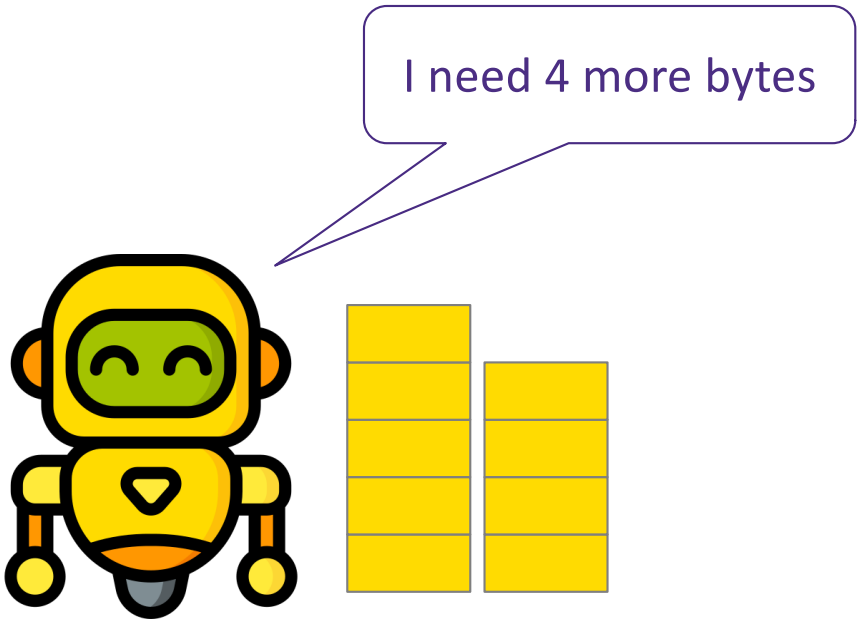
Shared by all program

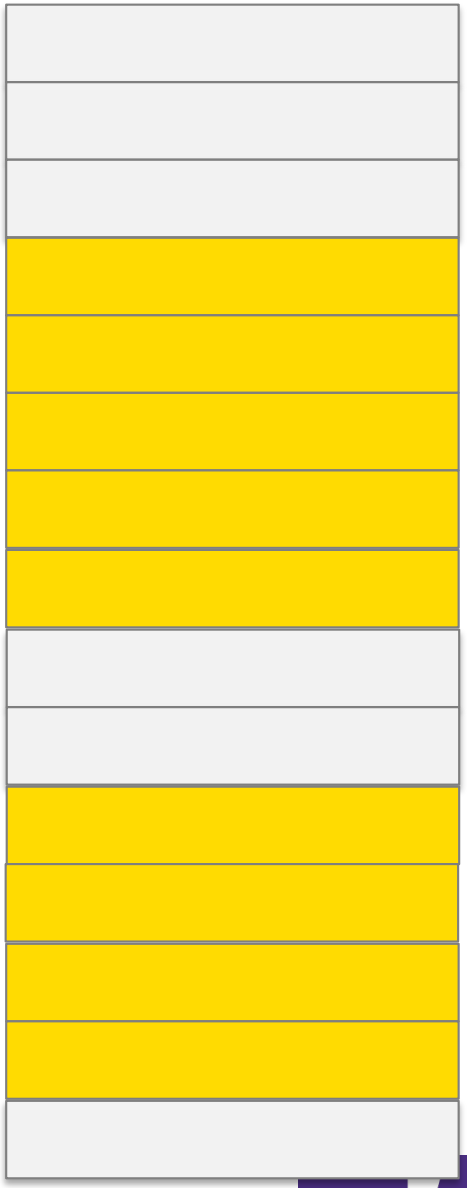




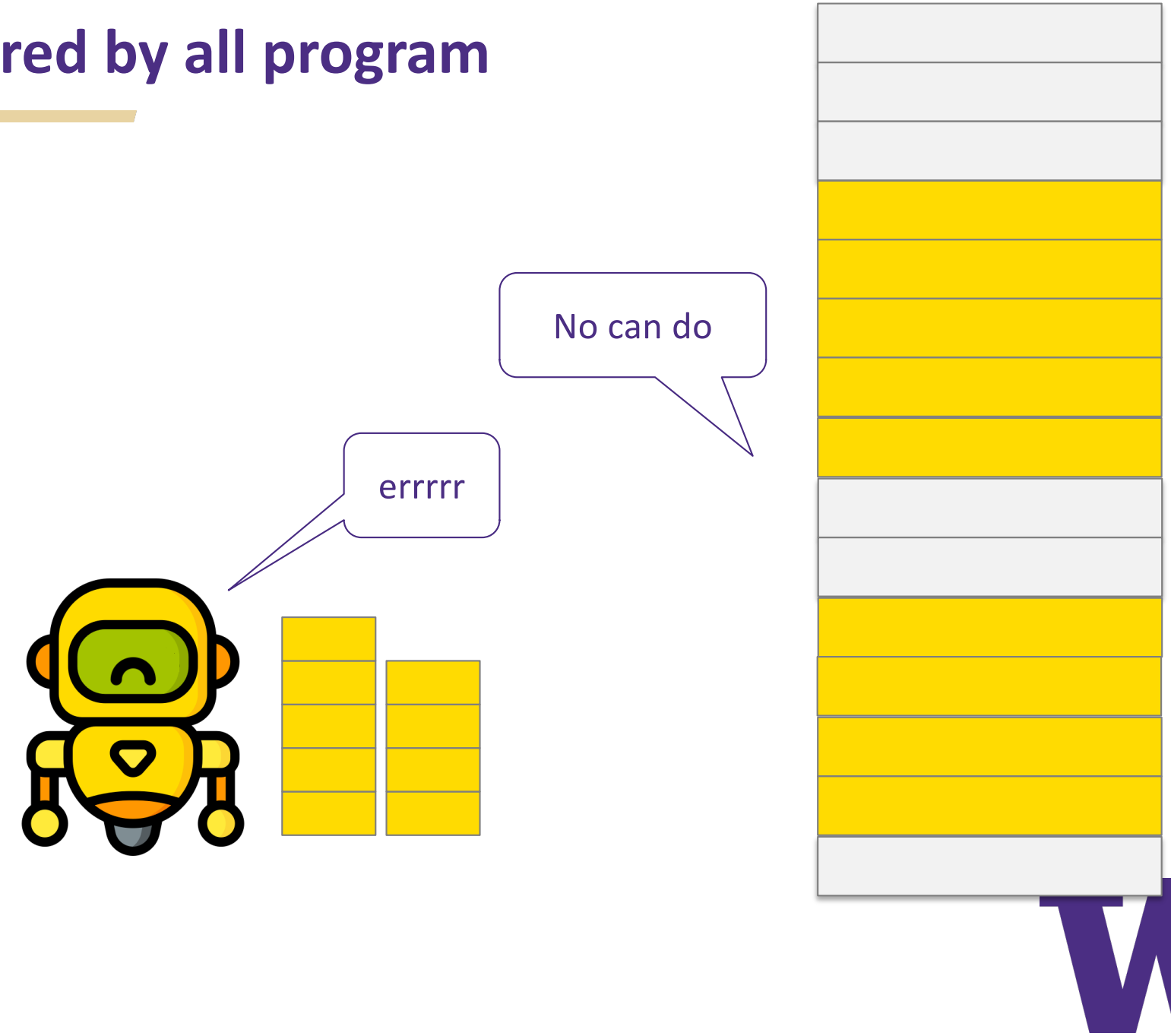


Shared by all program



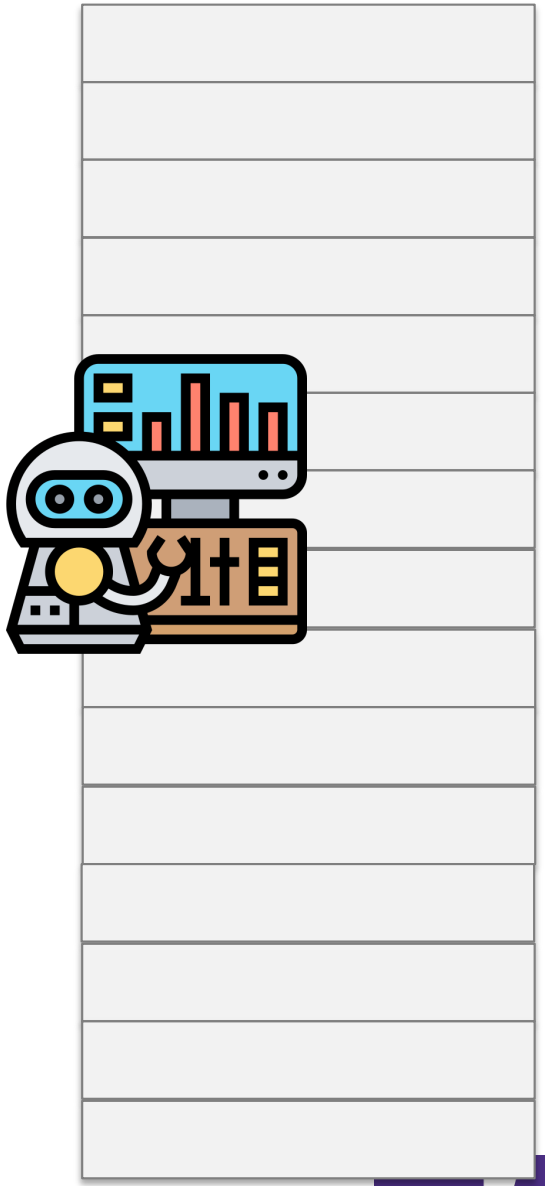


Shared by all program



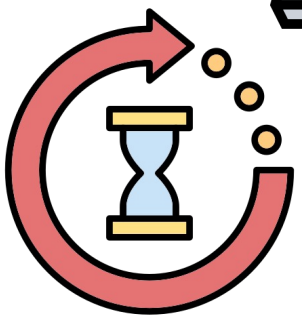
Shared by all program

Memory Management by OS



Shared by all program

Memory Management by OS



Slow……