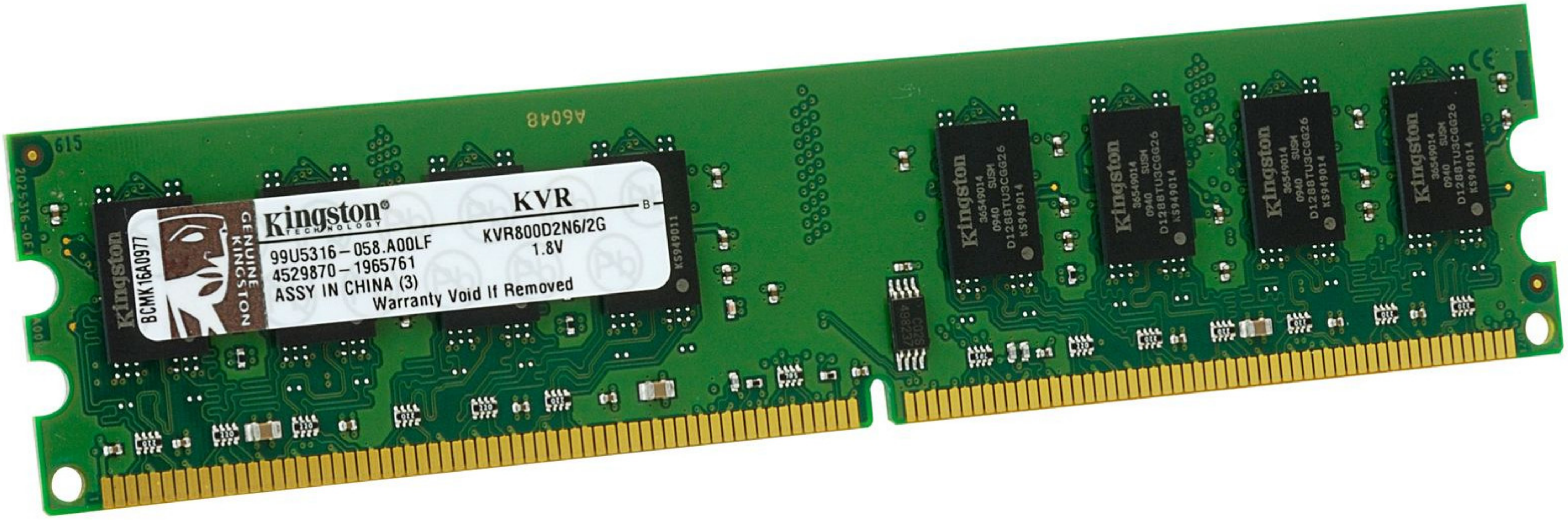
# Model of Execution

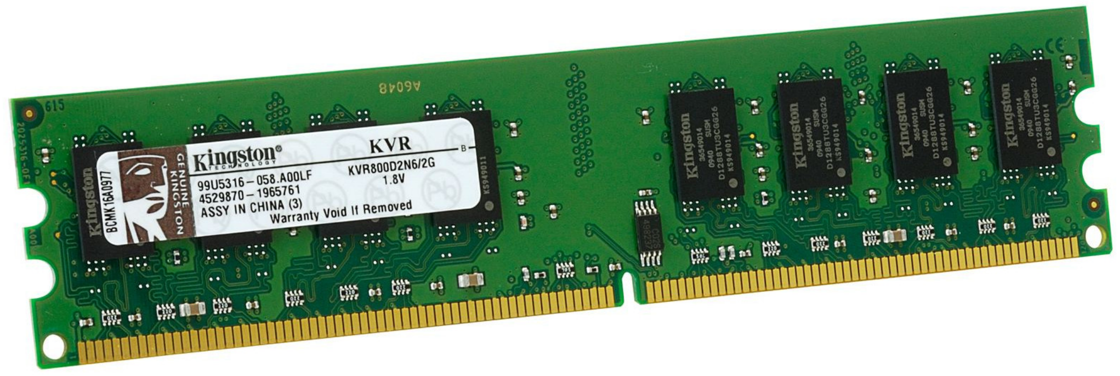
# Let's Talk About Memory

- Random Access Memory (RAM)
  - Access any value by index
  - Effectively a giant array
- All values in your program is stored here



# Let's Talk About Memory

- Significantly faster than reading/writing to disk
  - Even with an SSD
- Significantly more expensive than disk space



>

# Let's Talk About Memory

- Operating System (OS) controls memory
- On program start, OS allocates a section of memory for our program
  - Gives access to a range of memory addresses/indices

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | <Our Program Memory> |
| 27171 | <Our Program Memory> |
| 27170 | <Our Program Memory> |
| 27169 | <Our Program Memory> |
| 27168 | <Our Program Memory> |
| 27167 | <Our Program Memory> |
| 27166 | <Our Program Memory> |
| 27165 | <Our Program Memory> |
| 27164 | <Our Program Memory> |
| 27163 | <Our Program Memory> |
| 27162 | <Our Program Memory> |
| 27161 | <Used by another program> |
| ... | ... |

# Program Memory

- Some space is reserved for program data
- Details not important to CSE116
- The rest will be used for our data
- Data stored in the memory stack

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | <Command line args> |
| 27171 | <Our Program Memory> |
| 27170 | <Our Program Memory> |
| 27169 | <Our Program Memory> |
| 27168 | <Our Program Memory> |
| 27167 | <Our Program Memory> |
| 27166 | <Our Program Memory> |
| 27165 | <Our Program Memory> |
| 27164 | <Program Data> |
| 27163 | <Program Data> |
| 27162 | <Program Data> |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

**Note: This is example is language independent and will focus on the concept of memory. Each language will have differences in how memory is managed**

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Command line arguments added to the stack

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | |
| 27170 | |
| 27169 | |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
➔ i = 5
  n = computeFactorial(i)
  print(n)
}
```

- A variable named i of type Int is added to the stack
- The variable i is assigned a value of 5

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | |
| 27169 | |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Add n to the stack and assign it the value from the input argument

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | <computeFactorial stack frame> |
| 27169 | name:n, value:5 (computeFactorial) |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
➔ result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Add result to the stack and assign it the value 1

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | <function call stack frame> |
| 27169 | name:n, value:5 (function) |
| 27168 | name:result, value:1 (function) |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Begin loop block
- Add i to the stack and assign it the value 1
  - This is different from the i declared in main

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | <function call stack frame> |
| 27169 | name:n, value:5 (function) |
| 27168 | name:result, value:1 (function) |
| 27167 | <loop block> |
| 27166 | name:i, value:1 (function) |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Iterate through the loop
- look for variable named result in current stack frame
  - Found it outside the loop block
  - Update it's value (remains 1 on first iteration)

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | <function call stack frame> |
| 27169 | name:n, value:5 (function) |
| 27168 | name:result, value:1 (function) |
| 27167 | <loop block> |
| 27166 | name:i, value:1 (function) |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
 function computeFactorial(n){
   result = 1
➔  for (i=1; i<=n; i++) {
     result *= i
   }
   return result
 }

 function main(commandLineArgs){
   i = 5
   n = computeFactorial(i)
   print(n)
 }
```

- Iterate through the loop
- look for variable named i in current stack frame
    - Found it inside the loop block
    - *Some languages look outside the current frame

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | <function call stack frame> |
| 27169 | name:n, value:5 (function) |
| 27168 | name:result, value:1 (function) |
| 27167 | <loop block> |
| 27166 | name:i, value:2 (function) |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
➔ }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Iterate through the loop until condition is false

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | <function call stack frame> |
| 27169 | name:n, value:5 (function) |
| 27168 | name:result, value:120 (function) |
| 27167 | <loop block> |
| 27166 | name:i, value:5 (function) |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
→ return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- End of a code block is reached
- Delete ALL stack storage used by that block!
  - The variable i fell out of scope and no longer exists

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | <function call stack frame> |
| 27169 | name:n, value:5 (function) |
| 27168 | name:result, value:120 (function) |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
➔ return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- End of a function is reached
- Delete ALL stack storage used by that stack frame!
- Replace function call with its return value

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | function returned: 120 |
| 27169 | |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
➜ n = computeFactorial(i)
  print(n)
}
```

- Declare n
- Assign return value to n

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | name:n, value:120 (main) |
| 27169 | |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
➔ print(n)
}
```

- Print n to the screen
- At this point
  - No memory of variables n (function), i (function), or result

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | name:n, value:120 (main) |
| 27169 | |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- End of program
- Free memory back to the OS

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | |
| 27171 | |
| 27170 | |
| 27169 | |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

# Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- No memory of our program

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | |
| 27171 | |
| 27170 | |
| 27169 | |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

## Memory Heap

What is our data Needs to change size?

# Memory Heap

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | name:data, value:List of Ints (main) |
| 27169 | name:n, value:120 (main) |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

```
...
data.addValue(78)
...
```

- Variable data has values before and after it in memory
- Where do we store 78?
  - On the heap

# Memory Heap

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | name:data, value:97197 (main) |
| 27169 | name:n, value:120 (main) |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

| Index | Value |
|---|---|
| ... | ... |
| 97197 | List of Ints |
| 97198 | List of Ints |
| 97199 | |
| ... | ... |

- Heap memory is dynamic
- Can be anywhere in RAM
  - Location not important
  - Location can change
- Use references to find data
  - Variable data only stores a reference to the List of Ints

# Memory Heap

| Index | Value |
|---|---|
| ... | ... |
| 27173 | <Used by another program> |
| 27172 | commandLineArgs |
| 27171 | name:i, value:5 (main) |
| 27170 | name:data, value:97197 (main) |
| 27169 | name:n, value:120 (main) |
| 27168 | |
| 27167 | |
| 27166 | |
| 27165 | |
| 27164 | |
| 27163 | |
| 27162 | |
| 27161 | <Used by another program> |
| ... | ... |

| Index | Value |
|---|---|
| ... | ... |
| 97197 | List of Ints |
| 97198 | List of Ints |
| 97199 | |
| ... | ... |

- Heap memory is dynamic
- Can be anywhere in RAM
    - Location not important
    - Location can change
- Variable data only stores location
- Objects *usually* stored in heap memory

# Memory Heap Example

| Index | Value |
|---|---|
| ... | ... |
| 63051 | <Used by another program> |
| 63052 | commandLineArgs |
| 63053 | name:data, value:38772 (main) |
| 63054 | |
| 63055 | |
| 63056 | |
| 63057 | <Used by another program> |
| ... | ... |

| Index | Value |
|---|---|
| ... | ... |
| 38772 | ClassWithStateObject |
| 38773 | -stateVar value:0 |
| 38774 | |
| ... | ... |

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState
  addToState(data)
  println(data.stateVar)
}
```

- Create instance of ClassWithState on the heap
- Store **memory address** of the new object in data

# Memory Heap Example

| Index | Value |
|---|---|
| ... | ... |
| 63051 | <Used by another program> |
| 63052 | commandLineArgs |
| 63053 | name:data, value:38772 (main) |
| 63054 | <function call stack frame> |
| 63055 | name:input, value:38772 (function) |
| 63056 | |
| 63057 | <Used by another program> |
| ... | ... |

| Index | Value |
|---|---|
| ... | ... |
| 38772 | ClassWithStateObject |
| 38773 | -stateVar value:0 |
| 38774 | |
| ... | ... |

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState
  addToState(data)
  println(data.stateVar)
}
```

- Create a stack frame for the function call
- input is assigned the value in data
    - Which is a **memory address**

# Memory Heap Example

| Index | Value |
|---|---|
| ... | ... |
| 63051 | <Used by another program> |
| 63052 | commandLineArgs |
| 63053 | name:data, value:38772 (main) |
| 63054 | <function call stack frame> |
| 63055 | name:input, value:38772 (function) |
| 63056 | |
| 63057 | <Used by another program> |
| ... | ... |

| Index | Value |
|---|---|
| ... | ... |
| 38772 | ClassWithStateObject |
| 38773 | -stateVar value:1 |
| 38774 | |
| ... | ... |

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState
  addToState(data)
  println(data.stateVar)
}
```

- Add 1 to input.state variable
- Find the object at memory address 38772
- Alter the state of the object at that address

# Memory Heap Example

| Index | Value |
|---|---|
| ... | ... |
| 63051 | <Used by another program> |
| 63052 | commandLineArgs |
| 63053 | name:data, value:38772 (main) |
| 63054 | |
| 63055 | |
| 63056 | |
| 63057 | <Used by another program> |
| ... | ... |

| Index | Value |
|---|---|
| ... | ... |
| 38772 | ClassWithStateObject |
| 38773 | -stateVar value:1 |
| 38774 | |
| ... | ... |

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState
  addToState(data)
  println(data.stateVar)
}
```

- Function call ends
- Destroy all data in the stack frame
- input is destroyed
- Change to the object remains

# Memory Heap Example

| Index | Value |
|---|---|
| ... | ... |
| 63051 | <Used by another program> |
| 63052 | commandLineArgs |
| 63053 | name:data, value:38772 (main) |
| 63054 | |
| 63055 | |
| 63056 | |
| 63057 | <Used by another program> |
| ... | ... |

| Index | Value |
|---|---|
| ... | ... |
| 38772 | ClassWithStateObject |
| 38773 | -stateVar value:1 |
| 38774 | |
| ... | ... |

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState
  addToState(data)
  println(data.stateVar)
}
```

- Access data.stateVar
- Find the object at memory address 38772
- Access the state of the object at that address

# Memory Heap Example

| Index | Value |
|---|---|
| ... | ... |
| 63051 | <Used by another program> |
| 63052 | |
| 63053 | |
| 63054 | |
| 63055 | |
| 63056 | |
| 63057 | <Used by another program> |
| ... | ... |

| Index | Value |
|---|---|
| ... | ... |
| 38772 | |
| 38773 | |
| 38774 | |
| ... | ... |

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState
  addToState(data)
  println(data.stateVar)
}
```

- All memory freed when program ends

# Lecture Question

**Question**: In a package named "oop" create a Scala class named "Team" and a Scala object named "Referee".

Team will have:

- State values of type Int representing the strength of team's offense and defense with a constructor to set these values. The parameters for the constructor should be offense then defense
- A third state variable named "score" of type Int

Referee will have:

- A method named "playGame" that takes two Team objects as parameters and does not return a value. This method will alter the state of each input Team by setting their scores equal to their offense minus the other Team's defense. If a Team's offense is less than the other Team's defense their score should be 0 (no negative scores)
- A method named "declareWinner" that takes two Teams as parameters and returns the Team with the higher score. If both Teams have the same score, return a new Team object with offense and defense both set to 0

* This question will be open until midnight

# Lecture Question

## Sample Usage

```scala
val t1: Team = new Team(7, 3)
val t2: Team = new Team(4, 20)

Referee.playGame(t1, t2)
assert(Referee.declareWinner(t1, t2) == t2)
assert(Referee.declareWinner(t2, t1) == t2)
```

## Commentary

We create Team as a **class** since we want to create many objects of type team that will compete against each other. Each team will have different state (offense, defense, score), but will be the same type (Team)

Referee is an **object** since there only needs to be one of them and the object has no state. The same referee can officiate every game between any two teams

We pass **references** of objects of type Team to the Referee. Since the Referee has the references, when it changes the score of a Team that change is made to the state of that Team throughout the program