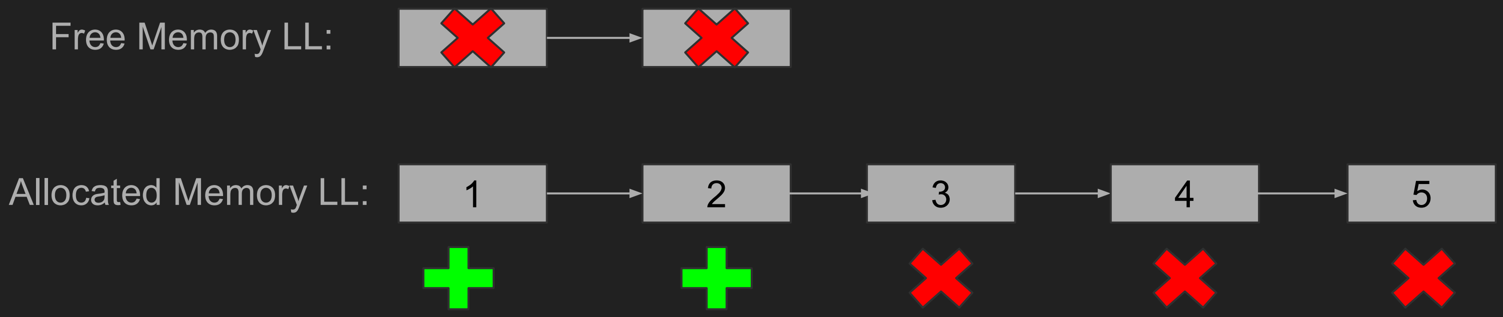
Heap Memory Recovery: Explicit vs Lazy Implicit

By: Noah Dunn

Lazy Implicit Memory Collection

- Used by languages like Java
- Utilizes two separate Linked Lists of Memory: Allocated and Free
- The Garbage Collector process waits until memory is needed, runs through all Allocated Memory to see what is not currently in use, and returns all that memory back to free memory



Lazy Implicit Memory Collection

Pros:

- Requires minimal effort on programmer’s end, it’s all handled for you
- In many cases, the default allocated Heap memory will not need to garbage collect during the runtime of the program, meaning the process will not trigger to begin with

Cons:

- When the garbage collector does have to activate, it takes significant effort to determine what is and is not in use
- The garbage collector may have to fight with the ‘CPU Scheduler’, increasing the runtime of a program further

Explicit Heap Memory Collection

- Most prominent examples are C++ and C
- The programmer specifies when memory is ready to be deleted using programmatic syntax (In C++ this is in the form of destructors and the delete clause)
- Memory management is user instigated not automatic
- The Philosophy of these languages is to avoid creating garbage when possible

Ex Code:

```
int *p1 = new int;

delete p1;
```

Explicit Heap Memory Collection

Pros:

- The programmer specifies exactly when they want memory removed
- The overhead of the garbage collector process is gone completely, leading to faster runtimes

Cons:

- Nothing is done automatically, the user bears full responsibility for memory cleanup
- A memory leak caused by negligent programming can lead to stealing away precious processing power over time