

Optimizing Flex Applications

*most optimizations can
be split into 2 categories*

Memory Performance

*but garbage collection
affects both*

Garbage Collection Empirical Model

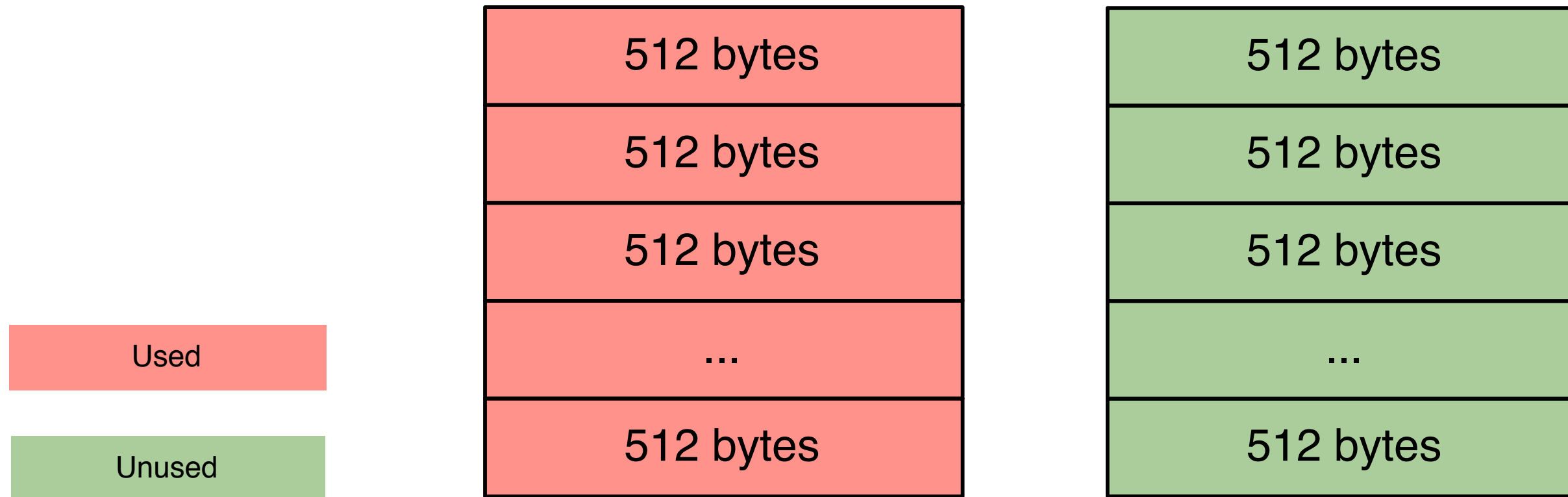
Why empirical?

- this is a description of *how we think* the garbage collection works in the player
- this is *not* an exact technical description
- the actual behavior is complex and difficult to describe
- the player may change it at some point
- this model worked for us so far

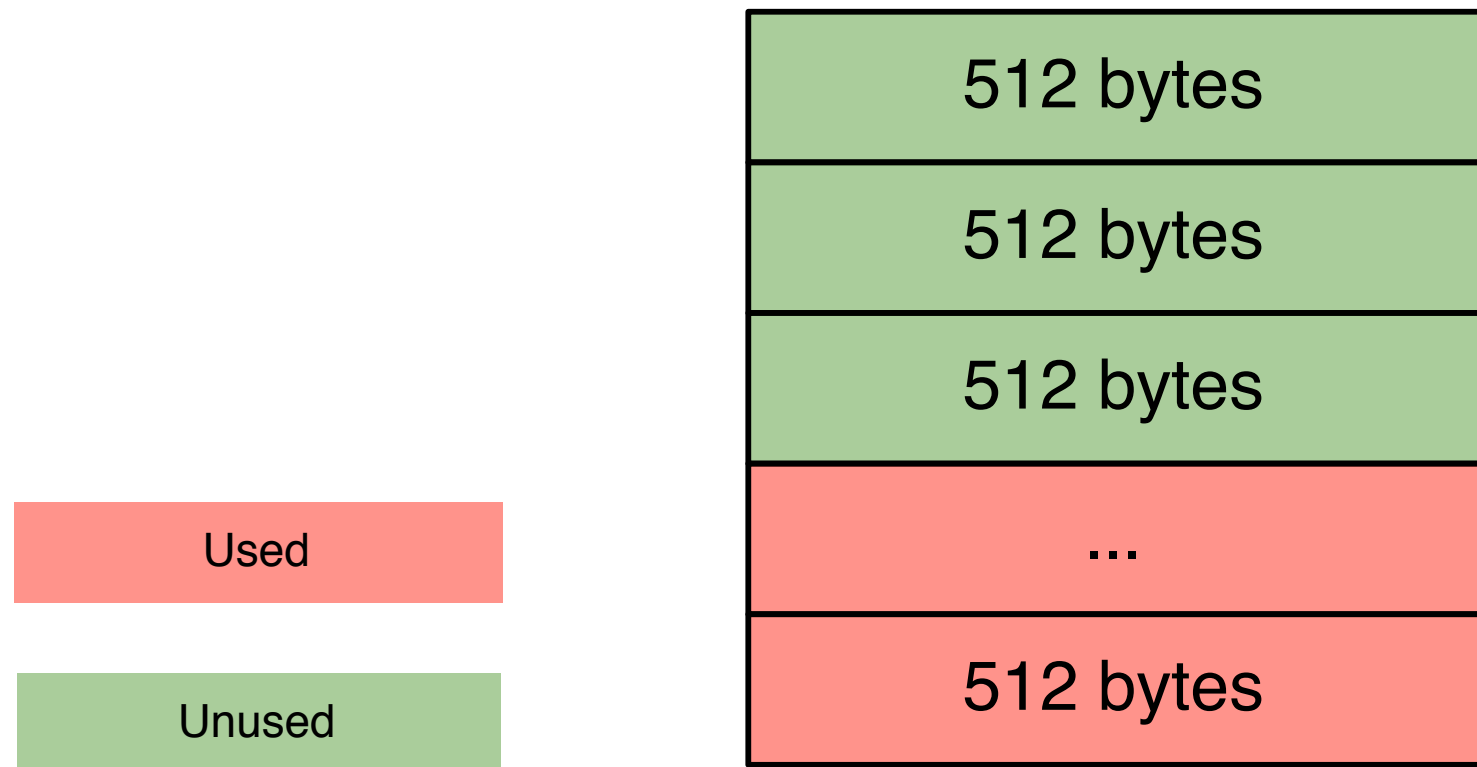
Memory Allocation

- **most Flash applications need to allocate small chunks of memory of common sizes, but...**
- **small frequent OS memory allocations can be slow**
- **Flash grabs large chunks of memory from the OS less often**
- **single large chunk is split into a pool of small blocks of a fixed size**
- **big chunks for Bitmaps, Files, etc. are not pooled**

after a pool is used up another large chunk is allocated from the OS



GC doesn't run Interactively

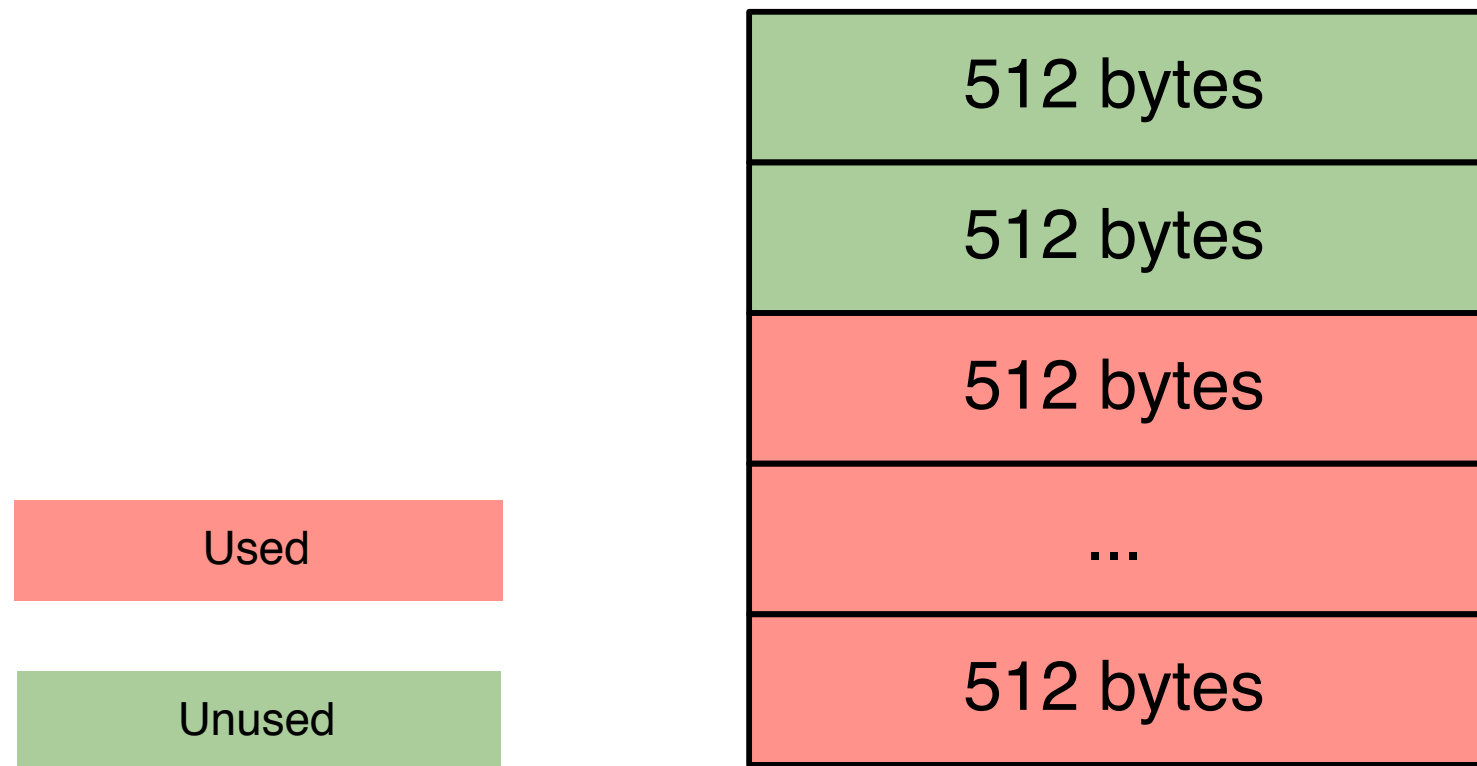


- lets assume that Foo's instance is 512b

```
var foo : Foo = new Foo();
```

- when allocated another 512b are used from the pool

GC doesn't run Interactively

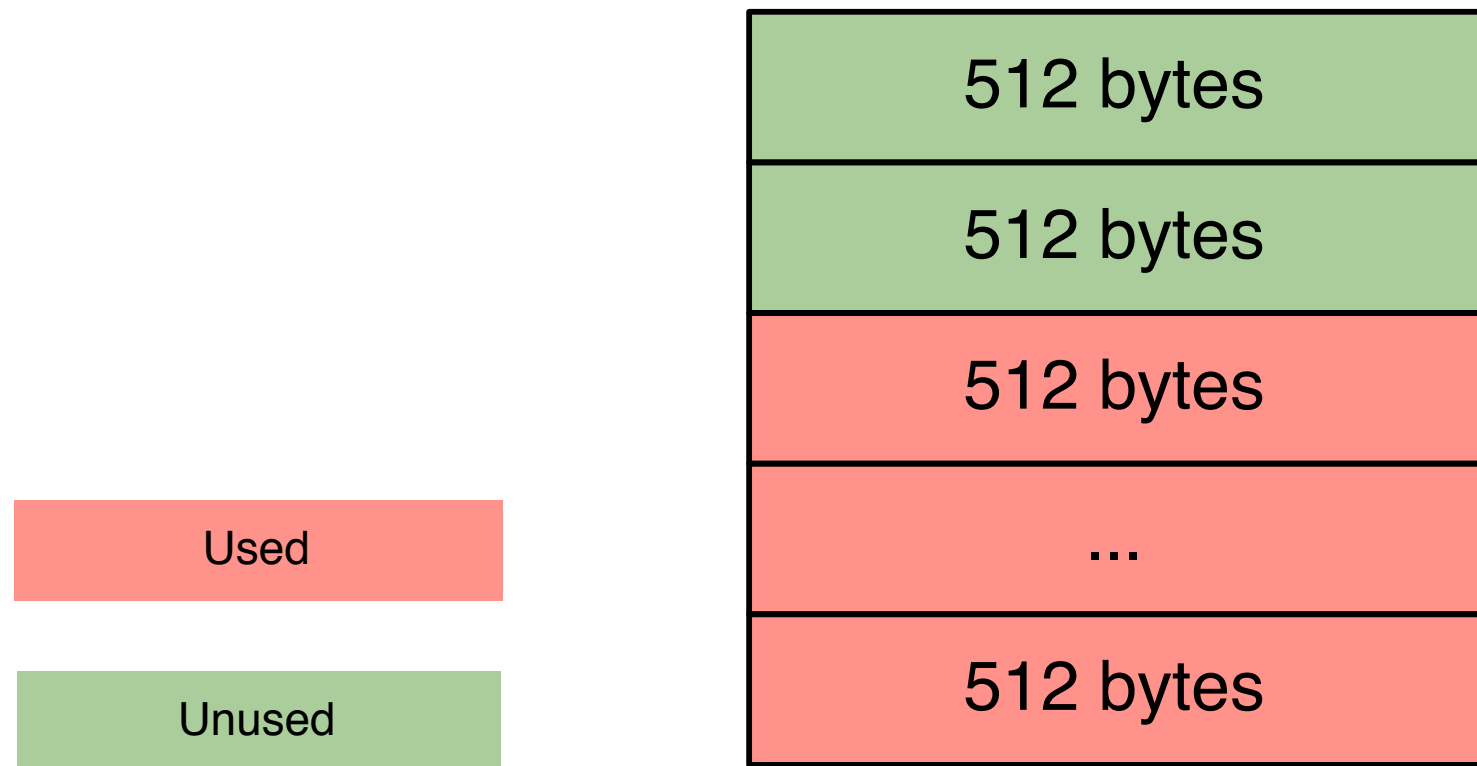


- lets assume that Foo's instance is 512b

```
var foo : Foo = new Foo();
```

- when allocated another 512b are used from the pool

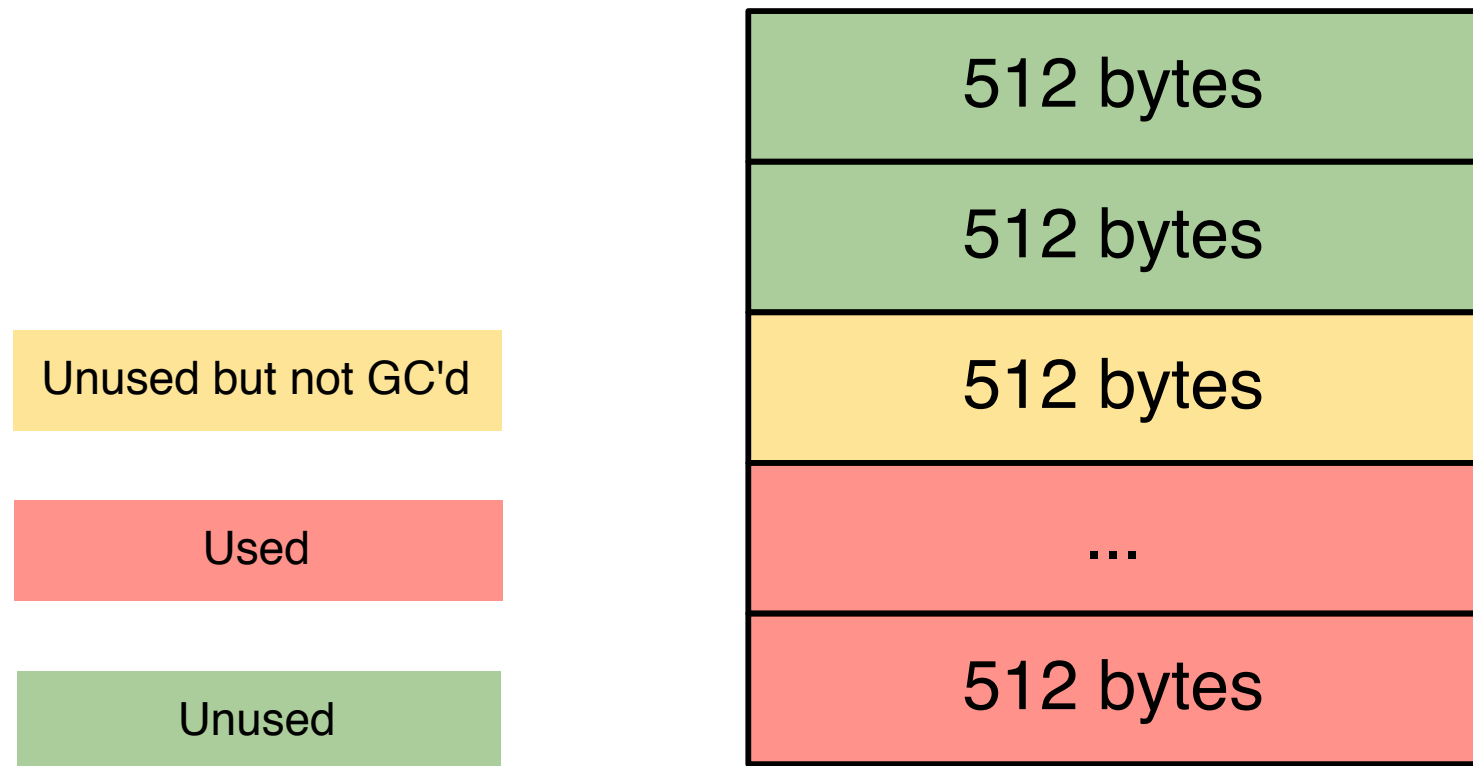
GC doesn't run Interactively



```
foo = null;
```

- when “freed” memory is not marked unused
- only GC will mark it unused

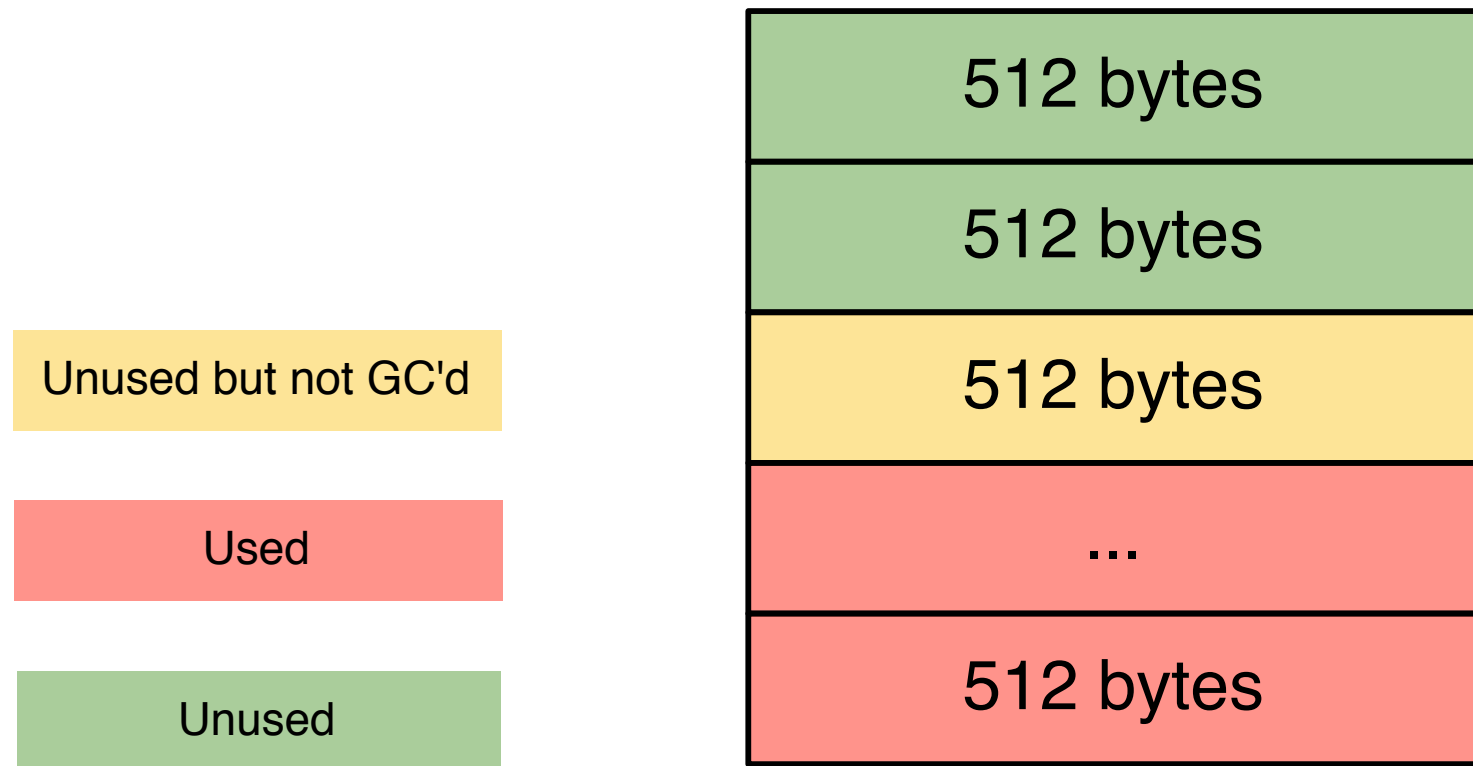
GC doesn't run Interactively



```
foo = null;
```

- when “freed” memory is not marked unused
- only GC will mark it unused

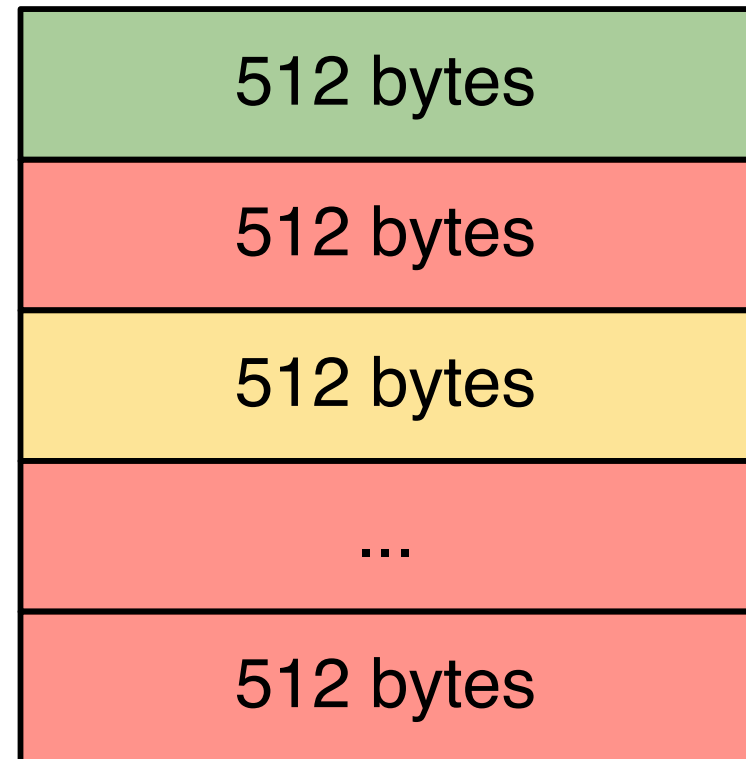
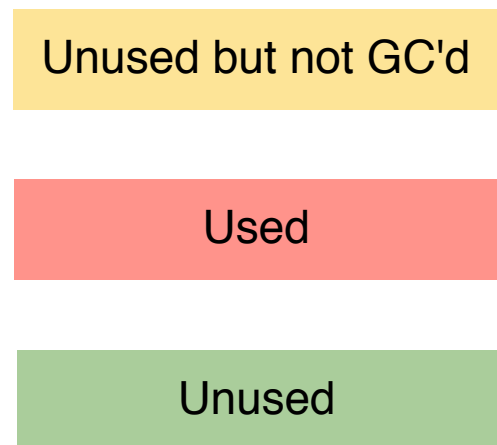
GC doesn't run Interactively



```
foo = new Foo();
```

- **when another Foo is allocated it might take a new block from the pool**

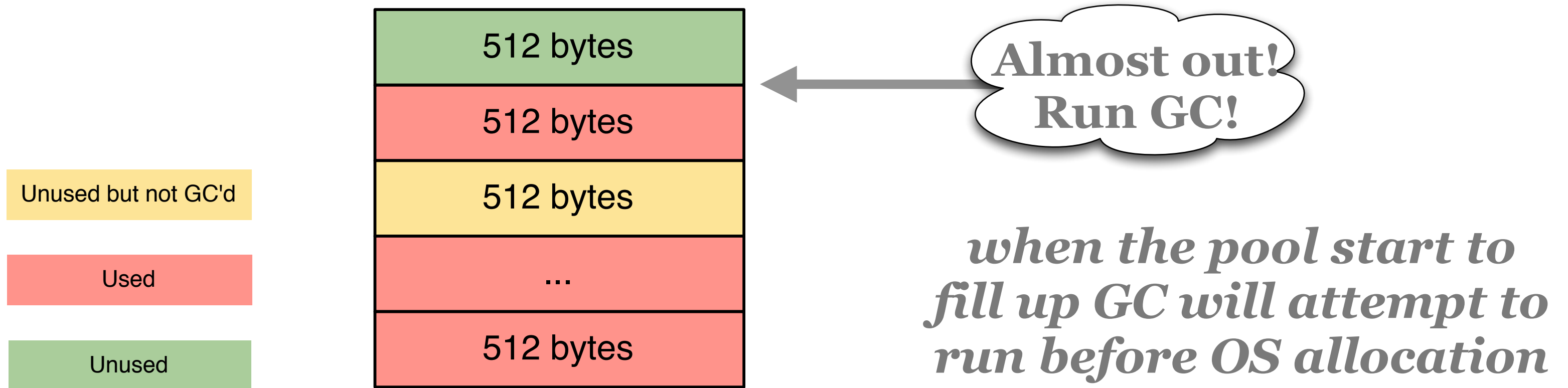
GC doesn't run Interactively



```
foo = new Foo();
```

- **when another Foo is allocated it might take a new block from the pool**
- **memory consumption grows**

GC is only triggered by Allocation



**Since GC is only triggered
by allocations the memory
usage of an idle application
will never change**

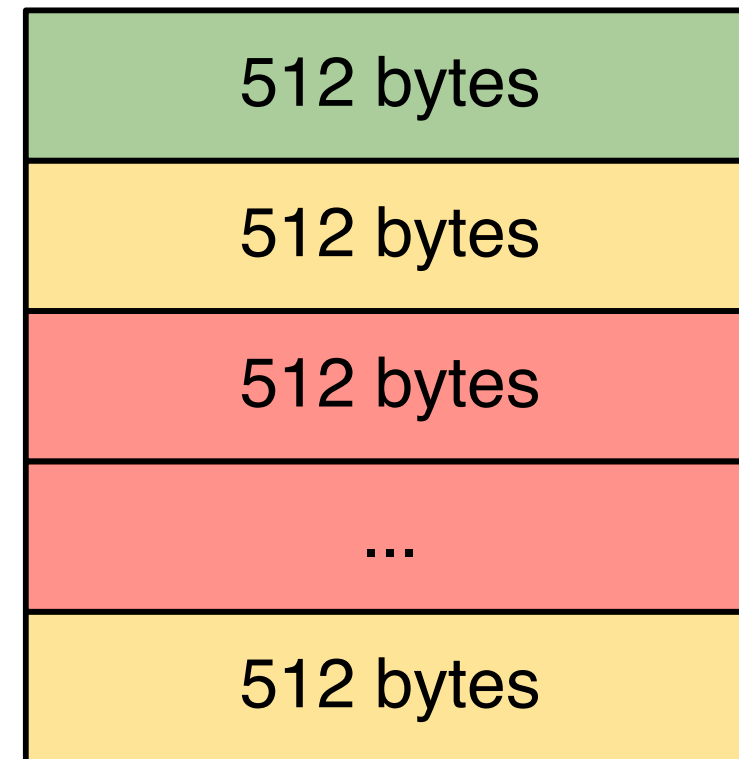
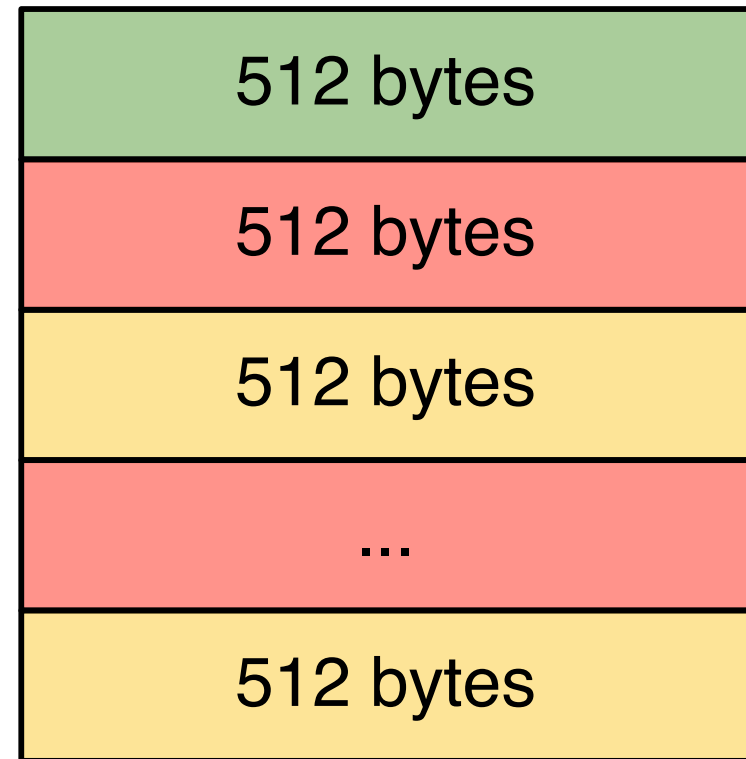
GC doesn't run Completely

memory before GC

Unused but not GC'd

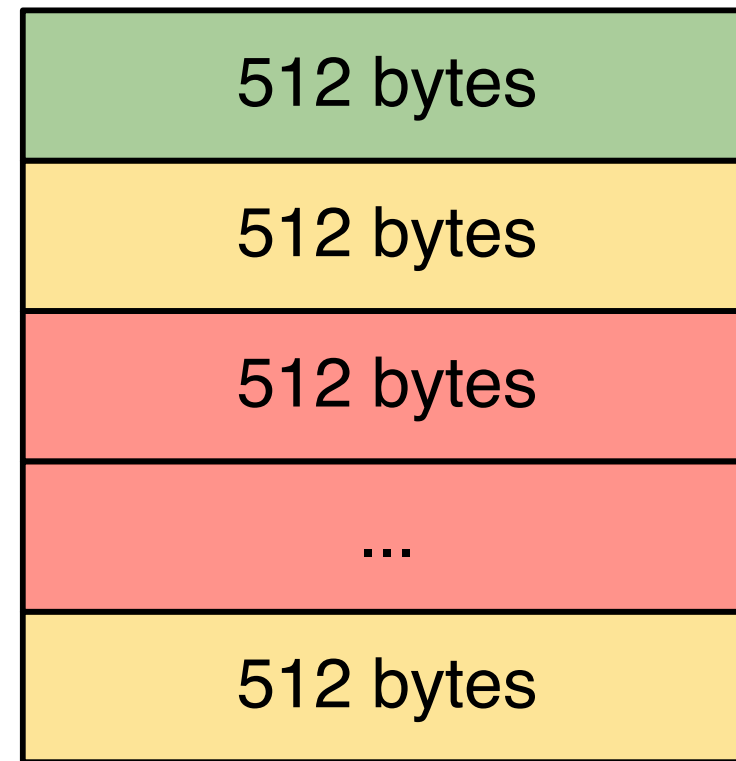
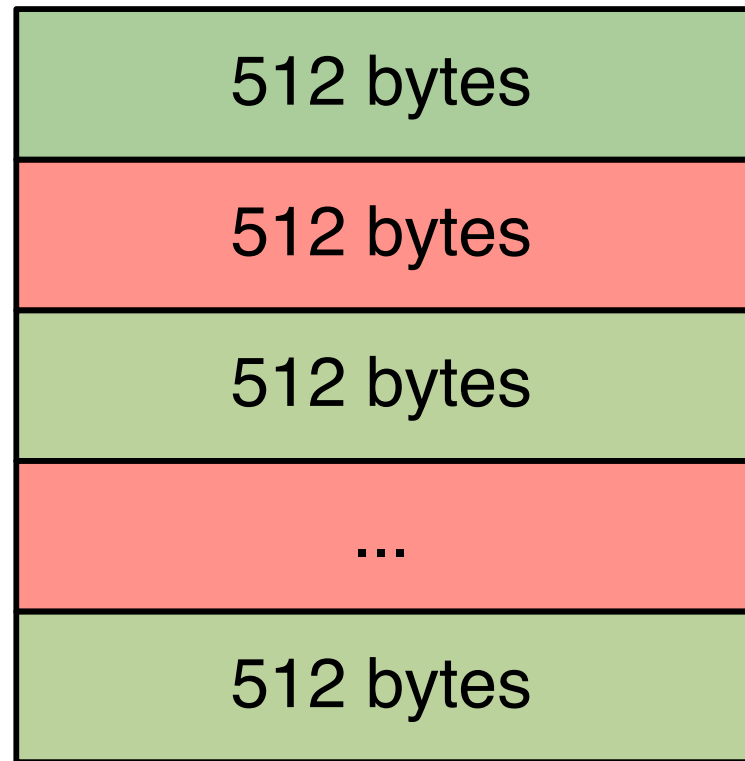
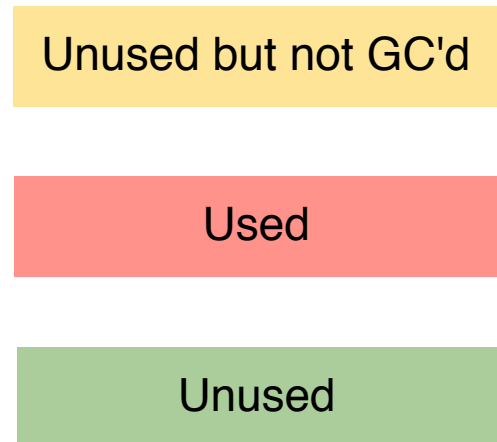
Used

Unused



GC doesn't run Completely

memory after GC

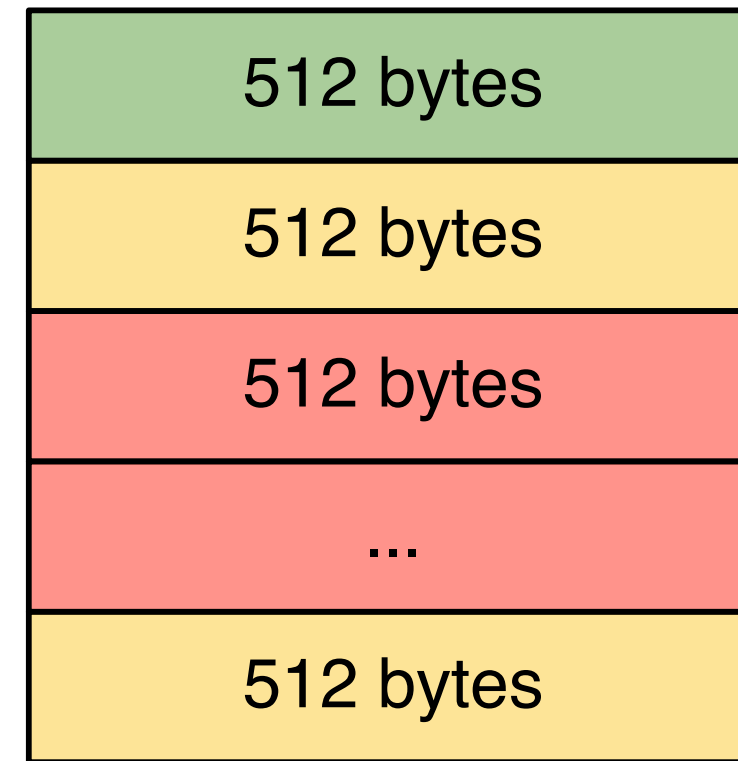
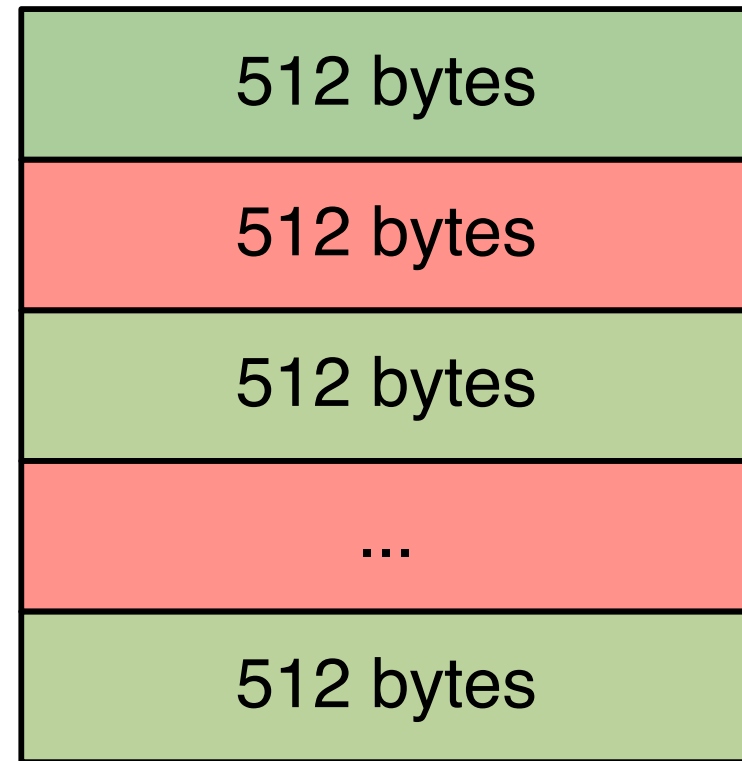
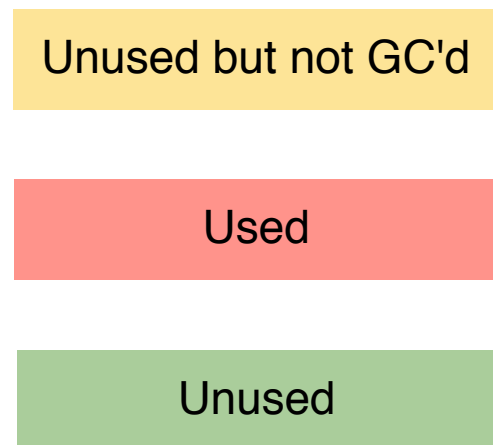


GC doesn't run Completely

- **the collection is not guaranteed to find all collectible blocks in one pass**
- **the GC must not interfere with rendering and interaction**
- **memory may never return to the initial point**

GC doesn't always free OS memory

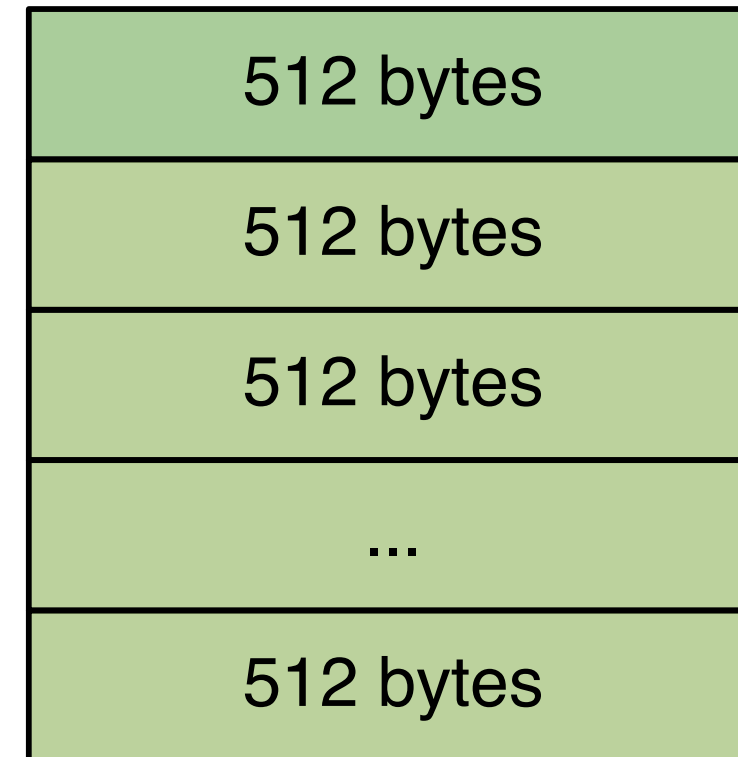
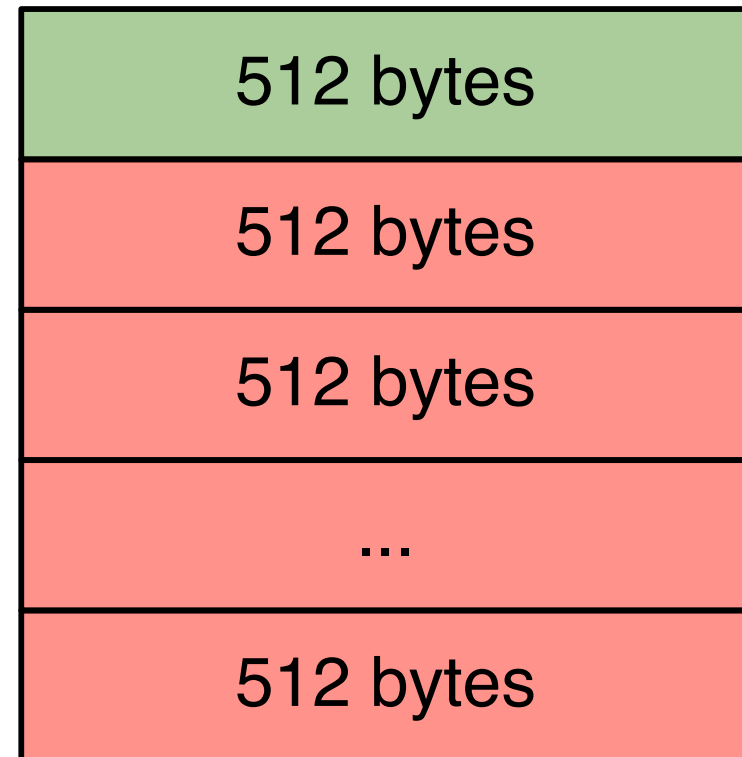
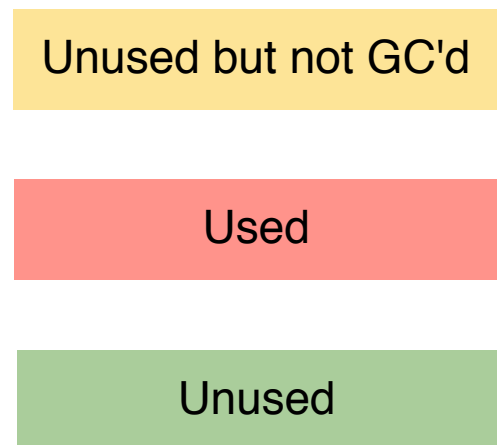
memory before GC



*GC will attempt to move blocks
from one big chunk to another*

GC doesn't always free OS memory

memory after GC

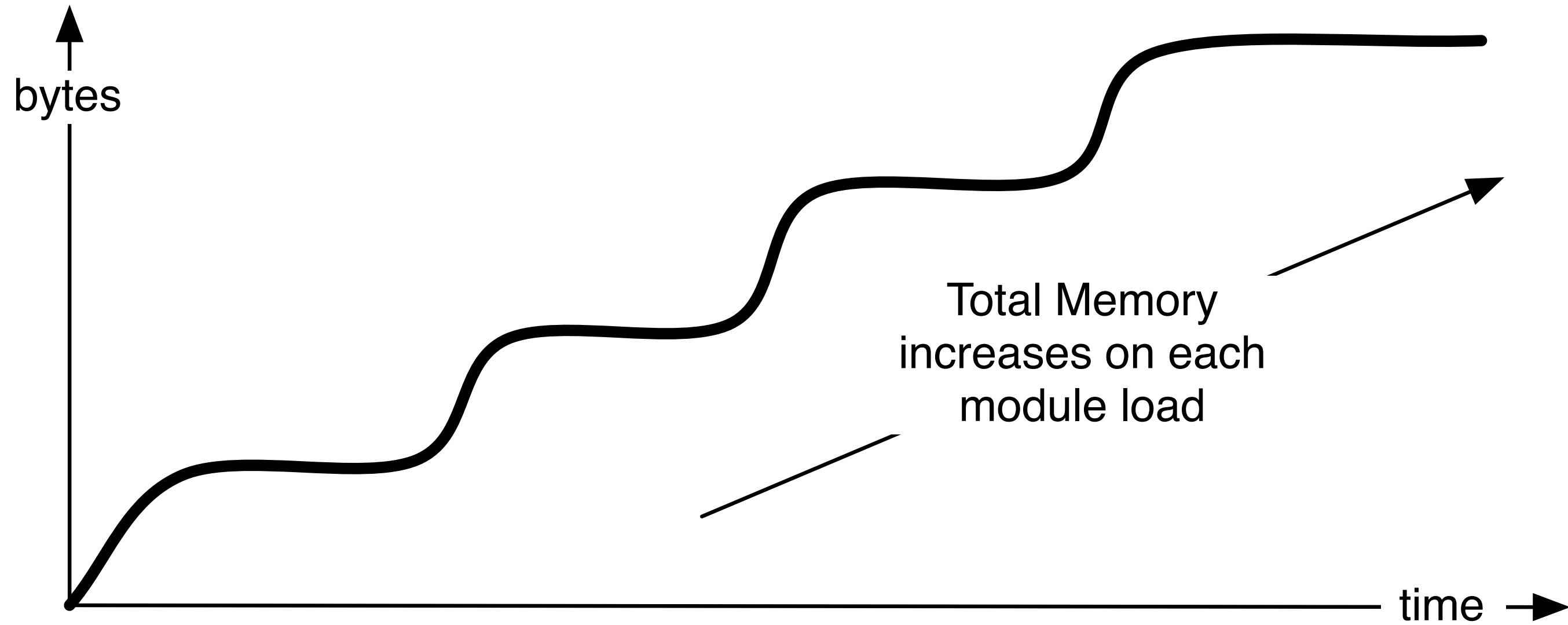


**GC is not predictable, so
how to detect memory
leaks?**

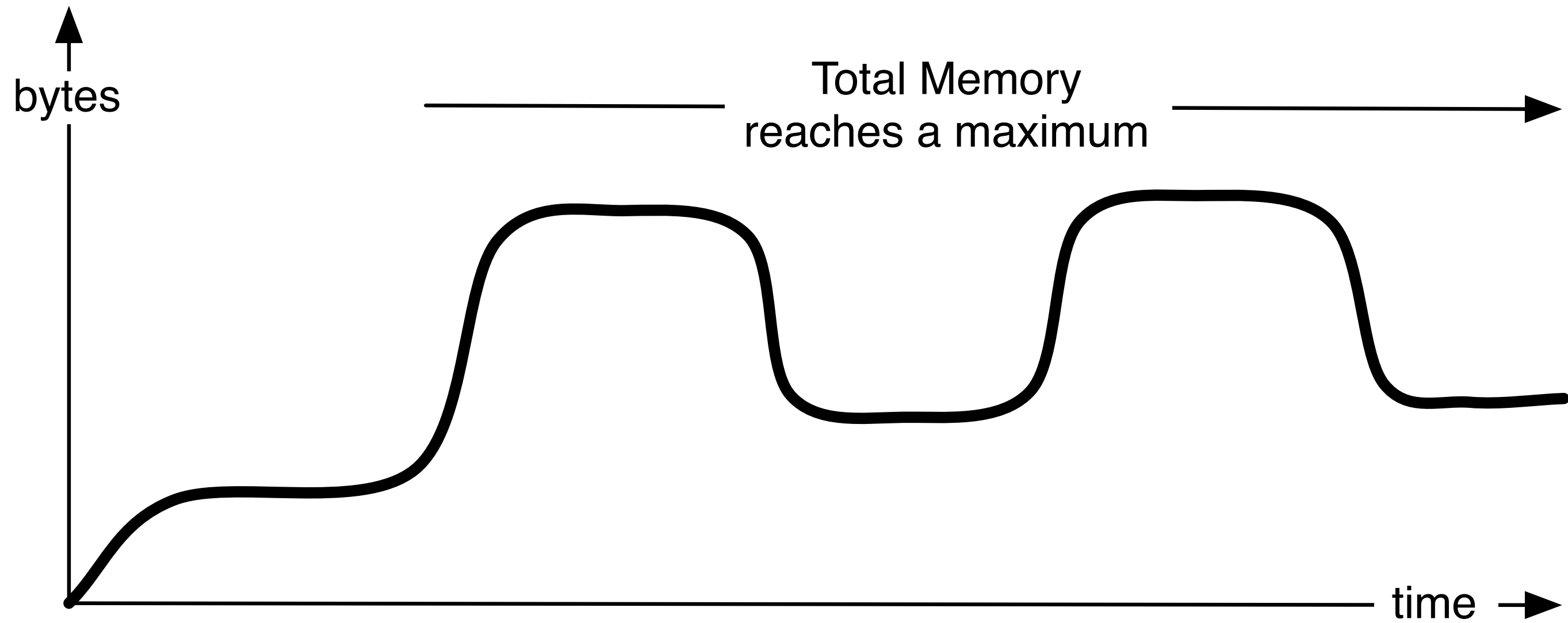
Detecting Memory Leaks

- **even small changes like location of mouse and keyboard events can affect the total memory used**
 - **therefore user interaction is not a good test**
- **you need to be concerned about repeatable sequences**
 - **popups coming and going**
 - **switching between various views**
 - **loading and unloading modules**
- **repeat these sequence long enough and observe how it does affect total memory used**

Leaking Memory Pattern



Without Memory Leaks



The Usual Suspects

- **Array**
- **Object used as map**
- **Dictionary with strong references**
- **Failure to remove event listeners**

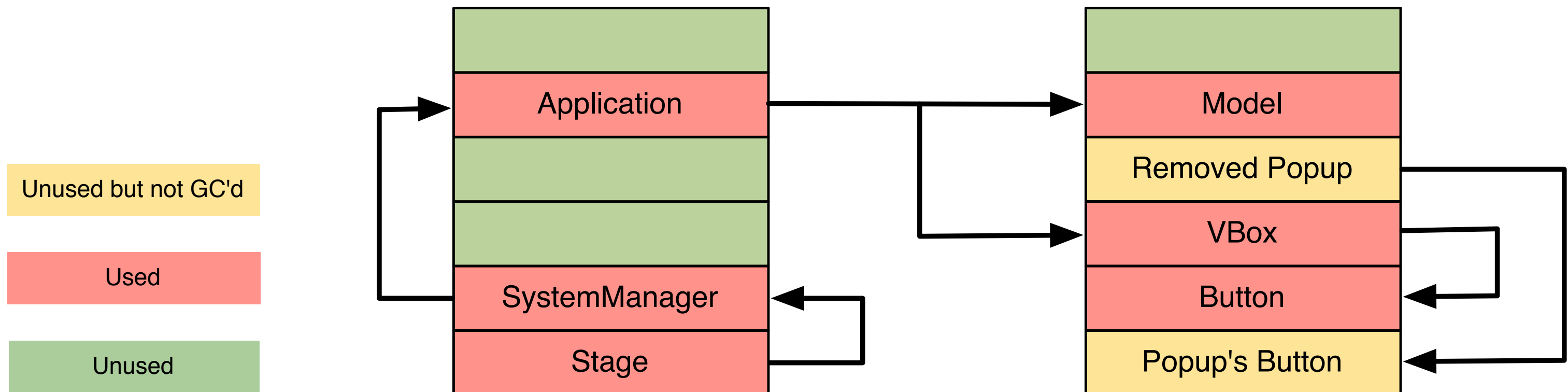
How Garbage Collection Works?

```

<mx:Application>
  <mx:Model id="model"/>
  <mx:VBox>
    <mx:Button/>
    ...
</mx:Application>

```

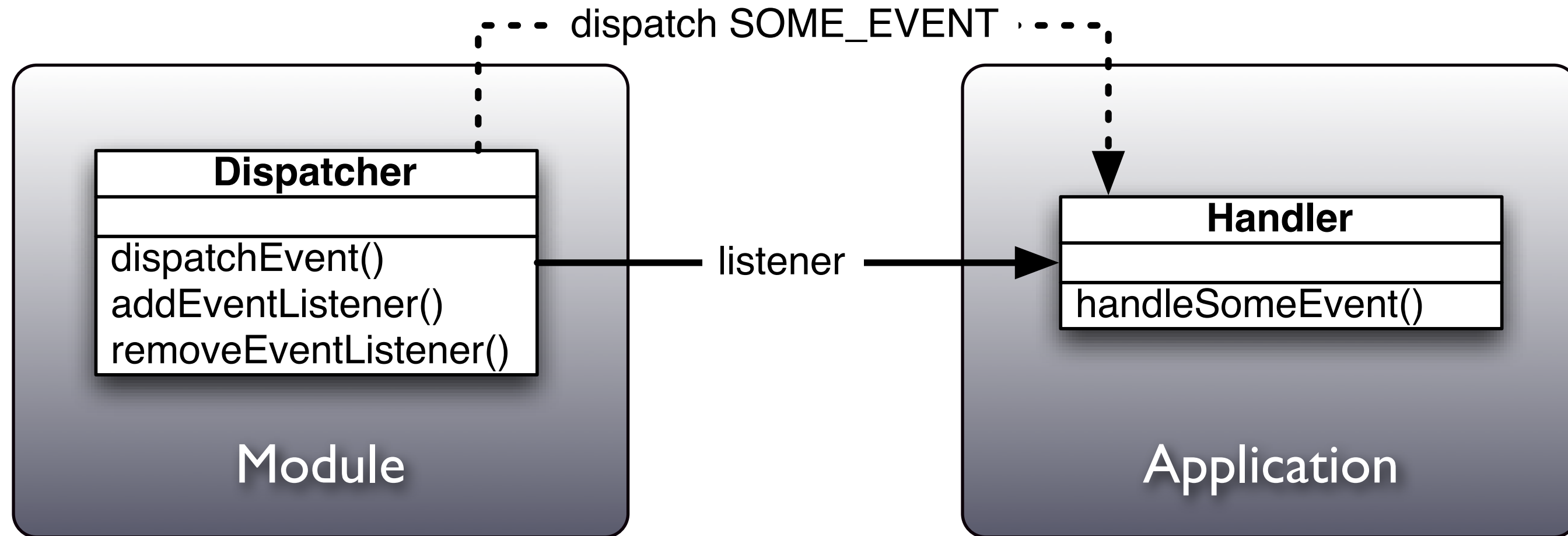
- GC starts at the roots of objects trees
- marks them and all objects they refer to
- then go through the heap and free unmarked objects
- top objects are ApplicationDomain, Stage, Stack for local variables



Removing Event Listeners

Child

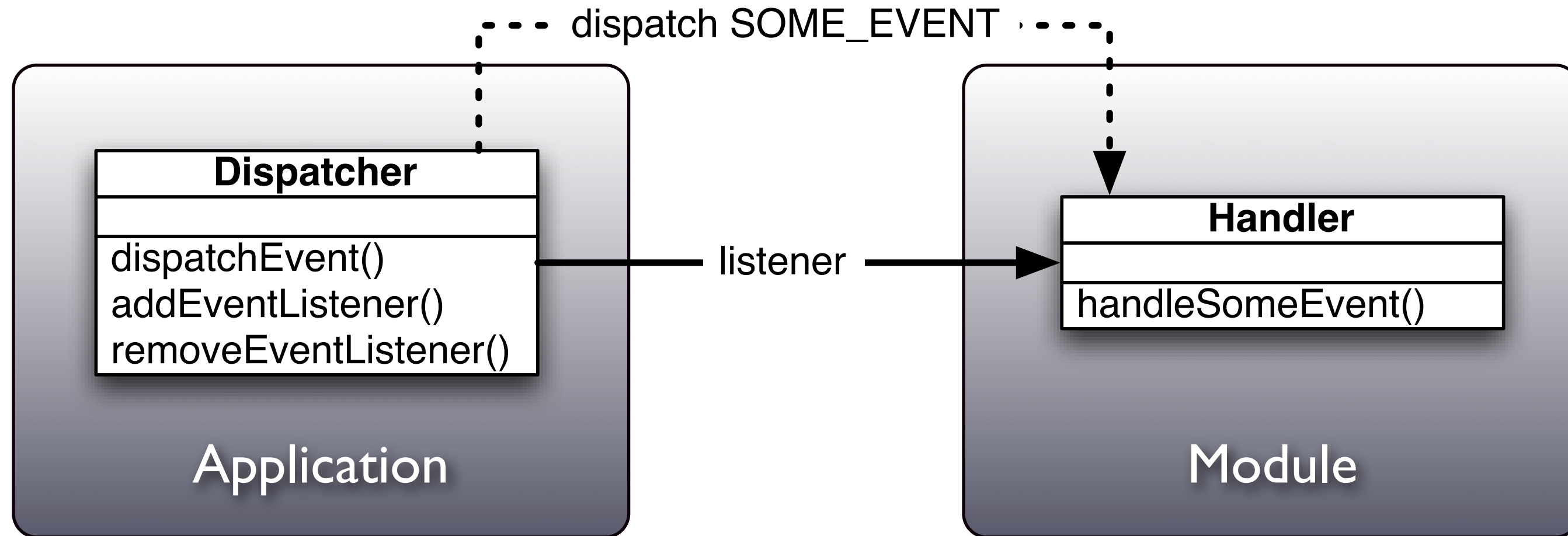
Parent



*doesn't cause memory leak
- it is not necessary to
remove the event listener*

Parent

Child



causes memory leak - the application keeps referencing the Module - event listener needs to be removed or use weak reference

Reuse Objects

- **instead of freeing unused objects, you can store them in a special cache for later use**
- **thus you can reuse renderers just like the List components reuses its items**

Unloading Checklist

When unloading modules or third-party content, be sure to:

- **free bitmap memory**
 - **stop video streams**
 - **stop audio streams**
 - **stop all MovieClips from animating**
 - **remove event listeners to global list of enterFrame, exitFrame, etc.**
 - **stop any downloads (http, sockets, FileReference)**
 - **clear any fonts from the font table**
- ...or use *Loader.unloadAndStop()* (only in Flash 10)**

Optimization Rules of Thumb

- **always compile in strict mode**
- **use typed data structures**
- **use sealed classes instead dynamic classes**
- **avoid globals when code is deeply nested - the VM will lookup for globals in each scope chain from the bottom to the top**
- **use *vector*<> instead Array (only in Flash 10)**

Other optimization techniques

```
var copy : Array = sourceArray.concat();
```

```
for (var i : int = 0; i < n; i++)  
/* not */  
for (var i : Number = 0; i < n; i++)
```

```
5000 * 0.001  
/* instead of */  
5000 / 1000
```

*the fastest way to copy
an Array*

*use integers for
iterations*

Multiply vs. Divide

```
comp.setStyle("color", 0xff00ff);
```

```
<mx:Panel>  
  <mx:VBox>  
    <mx:HBox>  
      <mx:Label text="Label 1"/>  
      <mx:VBox>  
        <mx:Label text="Label 2"/>  
      </mx:VBox>  
      <mx:HBox>  
        <mx:Label text="Label 3"/>  
        <mx:VBox>  
          <mx:Label text="Label 4"/>  
        </mx:VBox>  
      </mx:HBox>  
    </mx:HBox>  
  </mx:VBox>  
</mx:Panel>
```

avoid the `setStyle` method - one of the most expensive calls in the framework

too many nested containers dramatically reduces the performance

Summary

- **Flash memory allocation - big chunks less often, instead of small chunks frequently**
- **the GC is only triggered by allocation**
- **the GC doesn't run completely - all unused memory is not released in one pass**
- **GC is not predictable**
- **detecting memory leaks**
- **how GC works**
- **removing event dispatchers**
- **various optimization techniques**