

Garbage Collection

CS351 : Saelee

implicit deallocation

garbage = unreachable
memory

2 essential approaches

1. reference counting

2. *tracing* garbage collector

<1>

manual vs. automatic
reference counting

references = 0



deallocate

case study: Objective-C 1.0

object-oriented superset of C

messaging:

[obj method]

[obj method:arg]

Message	Description
<code>alloc</code>	allocate object with refcount = 1
<code>release</code>	decrease object's refcount by 1
<code>retain</code>	increase object's refcount by 1

```
@interface Widget : NSObject {
    NSString* name;
}

- (NSString*) name;
- (void) setName: (NSString*)new_name;
- (void) dealloc;
@end
```

```
Widget* w = [Widget alloc];
/* use widget */
[w release];
```

```
@implementation Widget
- (NSString*) name {
    return name;
}

- (void) setName: (NSString*)new_name {
    [name release];
    name = [new_name retain];
}

- (void) dealloc {
    [name release];
    [super dealloc];
}
@end
```

```
Widget* w = [Widget alloc];
NSString* str = [[NSString alloc] initWithCString:"Foo"];
```

```
annoying → [w setName:str];
            [str release];
```

```
/* use widget */
[w release];
```

```
Widget* w = [Widget alloc];
NSString* str = [NSString stringWithCString:"Foo"];

[w setName:str];
/* no need to release str -- done automatically */

/* use widget */
[w release];
```

```
/* convenience constructor for Widget class */  
+ (Widget*) widgetWithName: (NSString*)my_name {  
    Widget* w = [Widget alloc];  
    [w setName:my_name];  
    [w release];  
    return w;  
}
```

Message	Description
<code>autorelease</code>	decrease refcount by 1 <i>at some time in the future</i>

```
/* convenience constructor for Widget class */  
+ (Widget*) widgetWithName: (NSString*)my_name {  
    Widget* w = [Widget alloc];  
    [w setName:my_name];  
    [w autorelease];  
    return w;  
}
```

```
w = [Widget widgetWithName:@"Foo"];
```

```
/* no release necessary, but need to do *  
* [w retain] *  
* if we want to keep w around for later */
```

circular references?

e.g., doubly-linked lists, trees

conventions for:

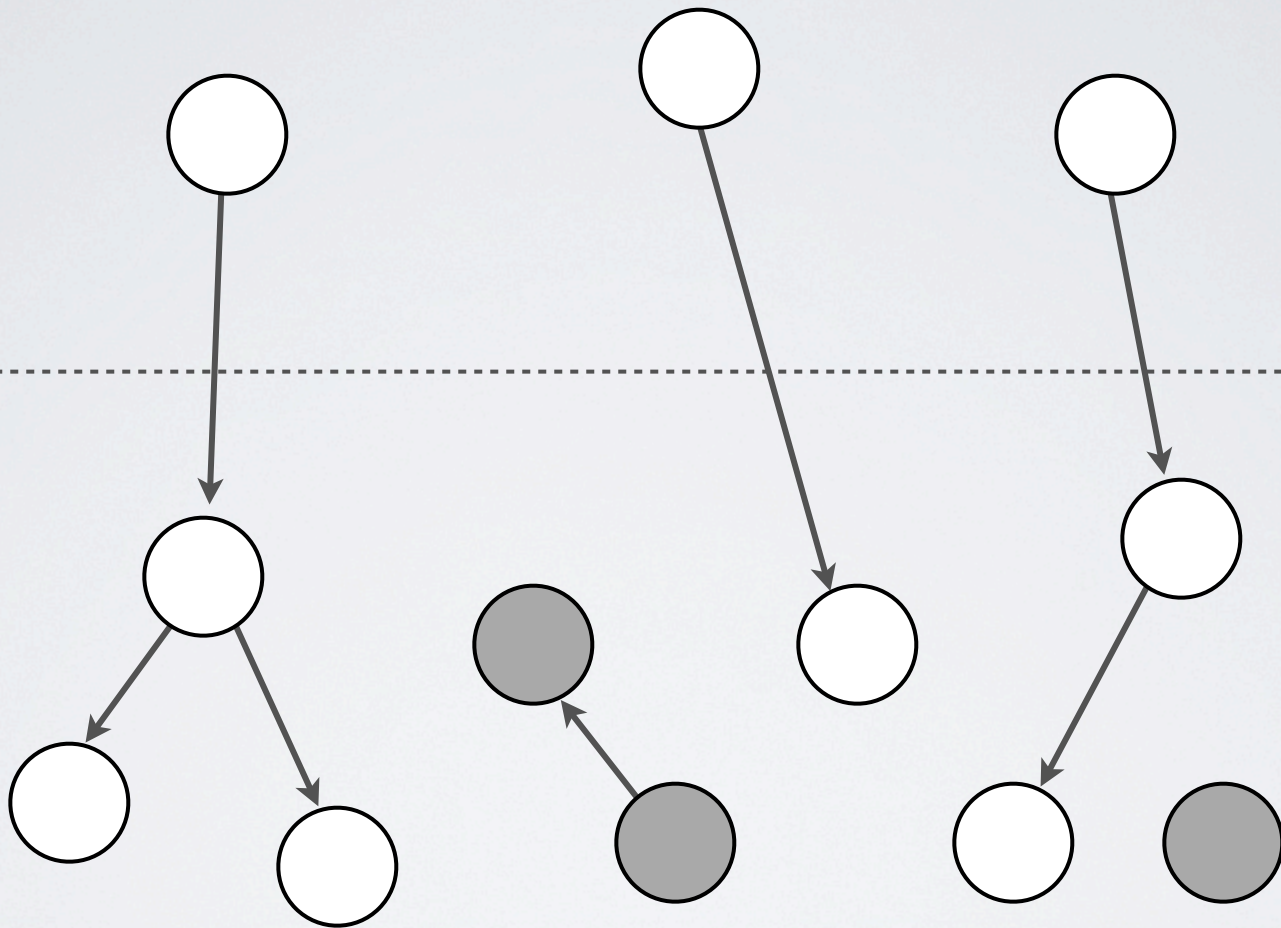
- parent/child
- peer nodes

</1>

<2>

identify *unreachable* memory

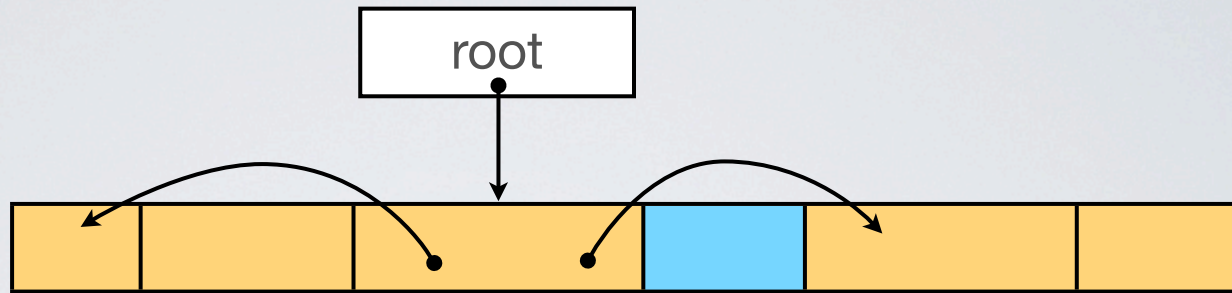
Static & Local
"Root" space



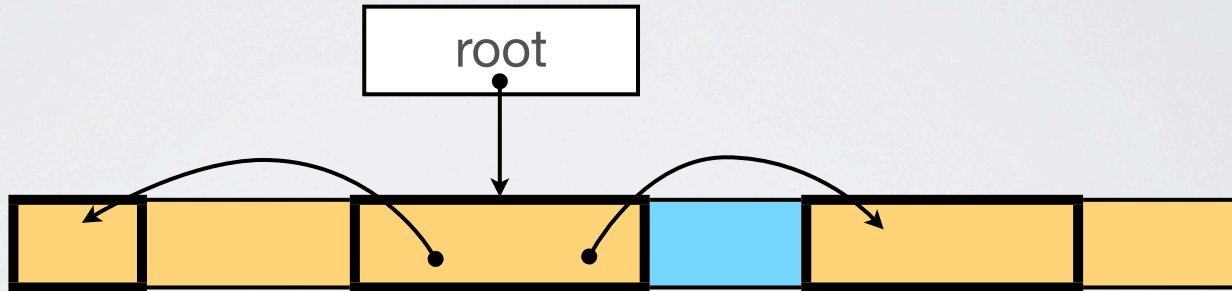
Heap space

mark and sweep

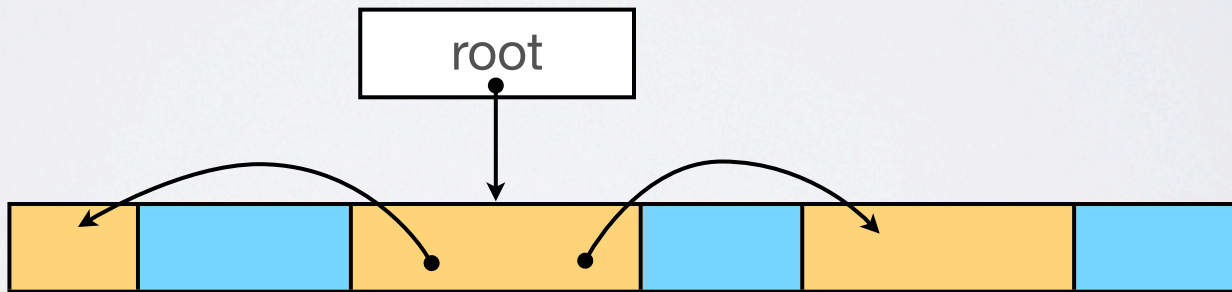
Before:



Mark:



Sweep:



```
void mark (void *p) {
    if (!is_ptr(p) || markBitSet(p)) {
        return;
    }
    setMarkBit(p);
    for (int i=0; i < length(p); i++) {
        mark(p[i]);
    }
    return;
}
```

```
void sweep (void *p, void *end) {
    while (p < end) {
        if markBitSet(p)
            clearMarkBit(p);
        else if (allocateBitSet(p))
            free(p);
        p += length(p);
    }
}
```

is_ptr?

every bit pattern = pointer

conservative GC

problem: need to “freeze”
memory for GC

tri-color **marking**

3 sets:

- white: “condemned” set
- black: reachable nodes that don't refer to white nodes
- gray: reachable nodes, but not yet scanned

partition memory

periodically “blacken”
a gray object

white → gray → black
(never other direction)

GC white set
when gray set is empty

on-the-fly marking

$\langle /2 \rangle$

reference counting
vs.
tracing GC

deterministic
vs.
non-deterministic

GC → performance hiccups

random!

real-time / embedded
systems

GC optimizations

opaque / internal pointers

Moving GC

memory compaction

reduce fragmentation

Generational GC

recent alloc likely to be freed
“infant mortality”

track generations of objects
mark only some generations

heuristic approach