Heap Heap Hooray: Memory Management

Tyler Gutowski, Trevor Schiff,
Dr. Ryan Stansifer (client)

Task	Description	Tyler	Trevor
Implement memory slab system	Contiguous, resizable memory chunks for use with copying garbage collection, and other algorithms that require low-level heap access.	0.3	0.7
Implement copying garbage collection	“Copying” garbage collection algorithm	0.6	0.4
Design compiler flag/configuration user interface	Design compiler user-interface, including flags that can be used when invoking the compiler (--gc, --opt, --verbose, etc.)	0.8	0.2
Implement compiler flag/configuration functionality	Implement the required functionality behind the compiler flags	0.2	0.8
Implement compiler flag for setting garbage collection method	Bake the garbage collection type into the executable, where the C runtime can see and then toggle the appropriate features	0.0	1.0
Run tests across refcount/marksweep/copying garbage collection methods	Create and run thorough test cases across all garbage collection implementations	N/A	N/A
Gather metrics for refcount/marksweep/copying garbage collection methods	Analyze test case results and gather further metrics	N/A	N/A

Basic Overview: Copying

When we try to allocate memory, but none is available, perform:

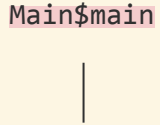
1. Mark Phase
   a. Same functionality as the mark phase of mark-sweep GC
   b. Traverse object graph & mark all reachable allocations
   c. Start from the program roots, or objects reachable from outside the heap (such as local variables)
2. Copy Phase
   a. All memory allocations exist in one of two “slabs” (contiguous chunks) of memory: the “from” slab, or the “to” slab
      i. “From” slab contains all allocations
   b. Copy all live allocations from the “from” slab to the “to” slab
      i. “To” slab contains only live (reachable) allocations
   c. Remove allocations from the “from” slab that have just been copied
      i. “From” slab contains only garbage (unreachable) allocations
   d. Release allocations which still exist in the “from” slab
      i. All memory associated with garbage has been released
   e. Swap handles to the “from” and “to” slabs
      i. “From” slab contains only live (reachable allocations)

Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

Graph

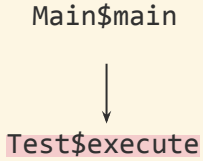


Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

Graph

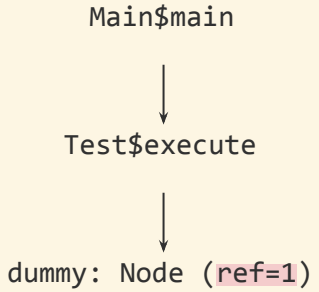


Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

Graph

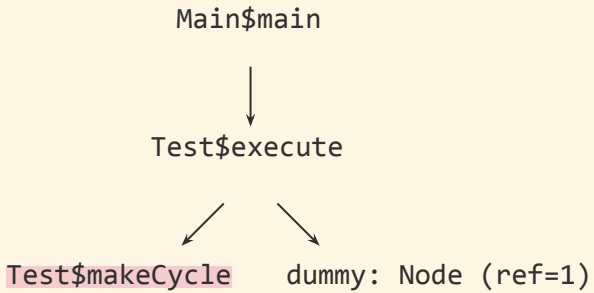


Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

Graph

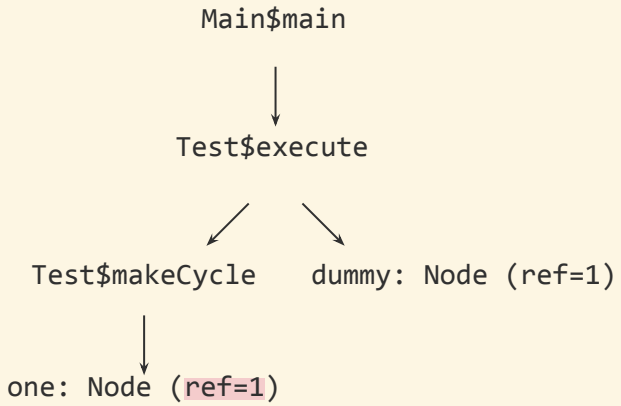


Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

Graph

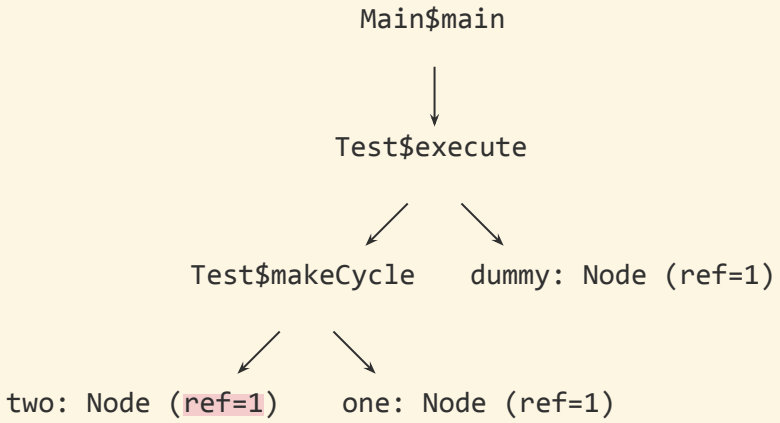


Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

Graph

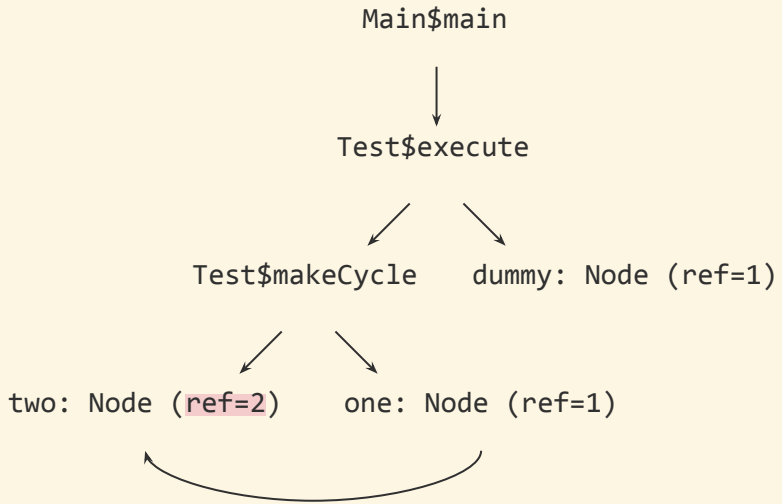


Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

Graph

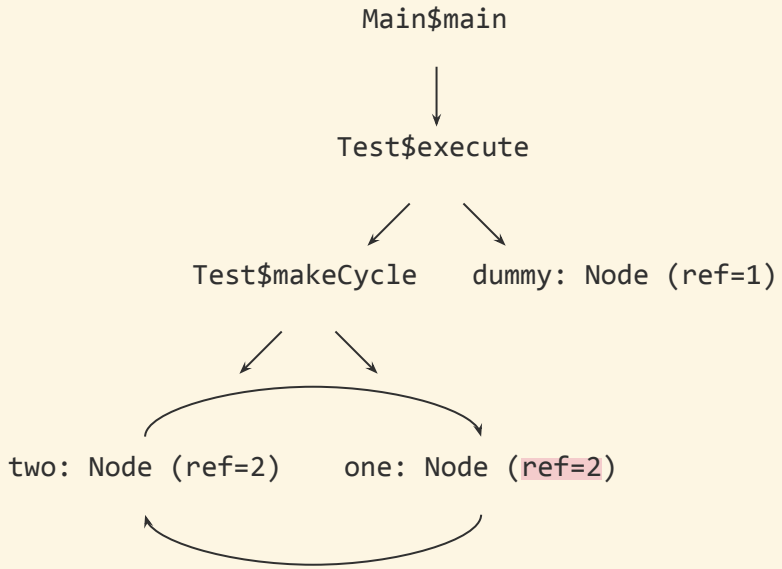


Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

Graph



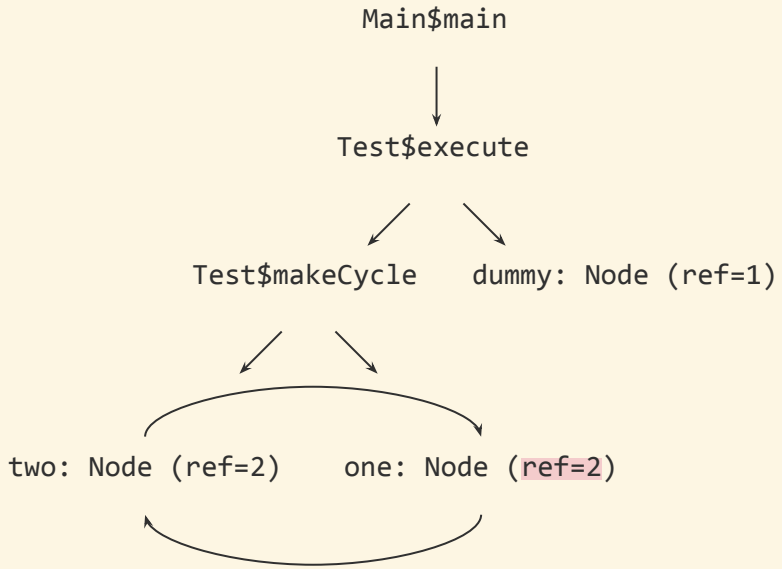
Implementation: Copying

Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

End of scope, decrement refcount of ‘one’ & ‘two’

Graph

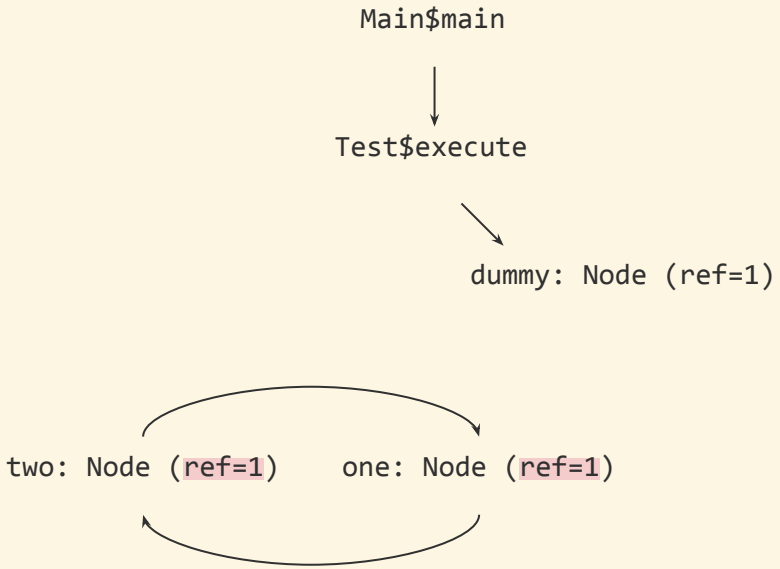


Implementation: Copying

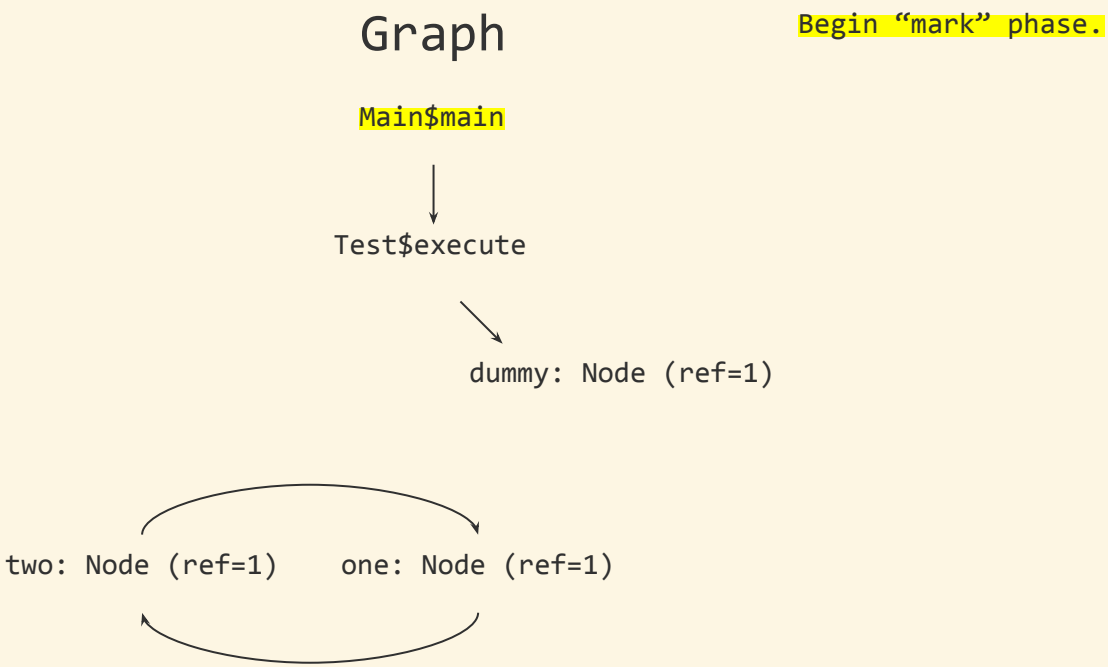
Program

```
class Main {
    public static void main(String[] a) {
        new Test().execute();
    }
}

class Test {
    public int execute() {
        Node dummy = new Node();
        makeCycle();
    }

    public void makeCycle() {
        Node one = new Node();
        Node two = new Node();

        one.setNext(two);
        two.setNext(one);
    }
}
```

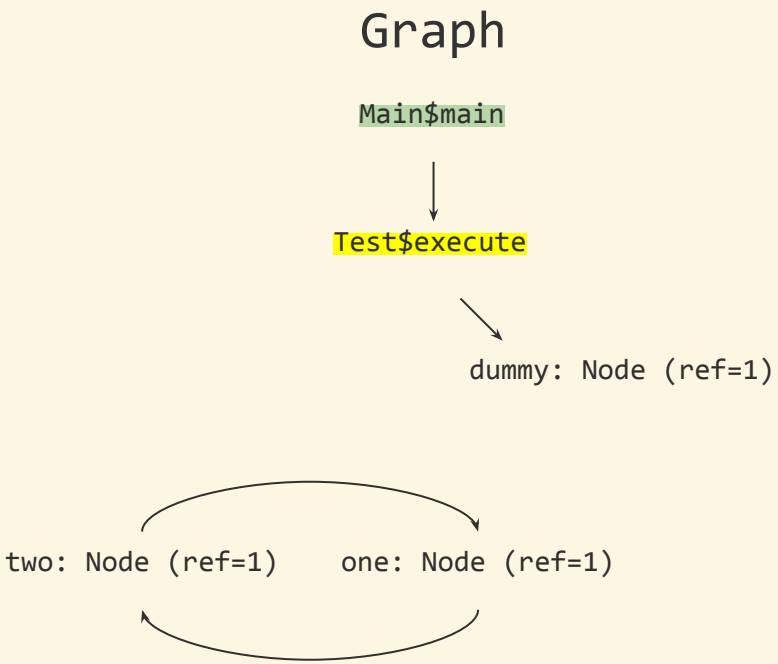
Graph



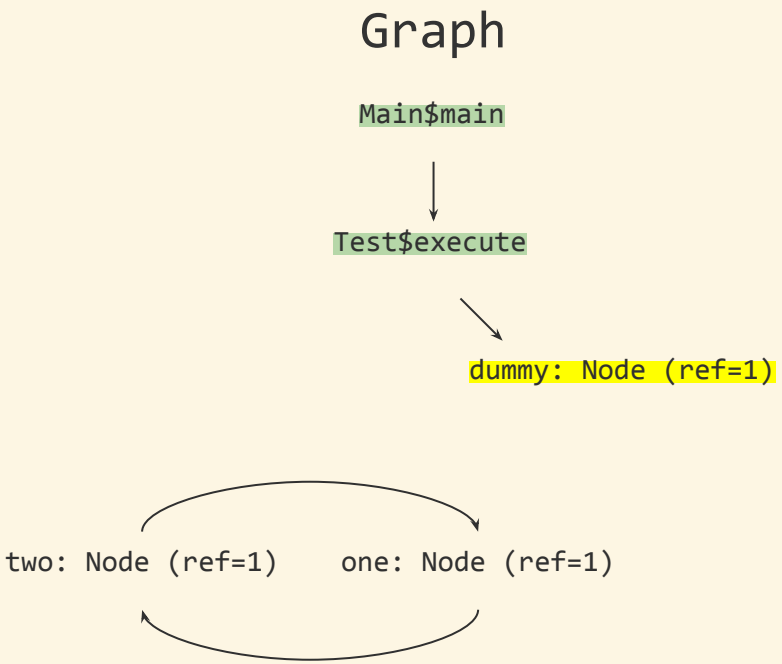
Implementation: Copying



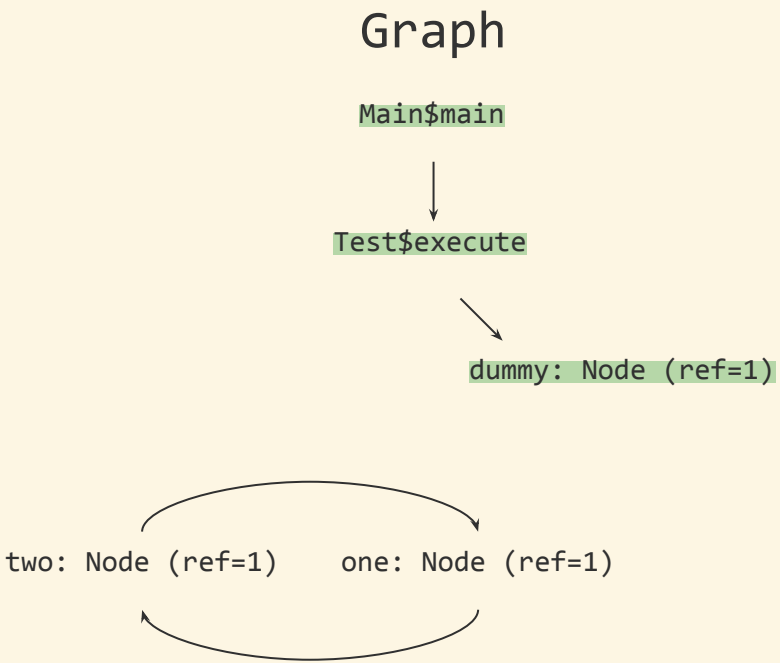
Implementation: Copying



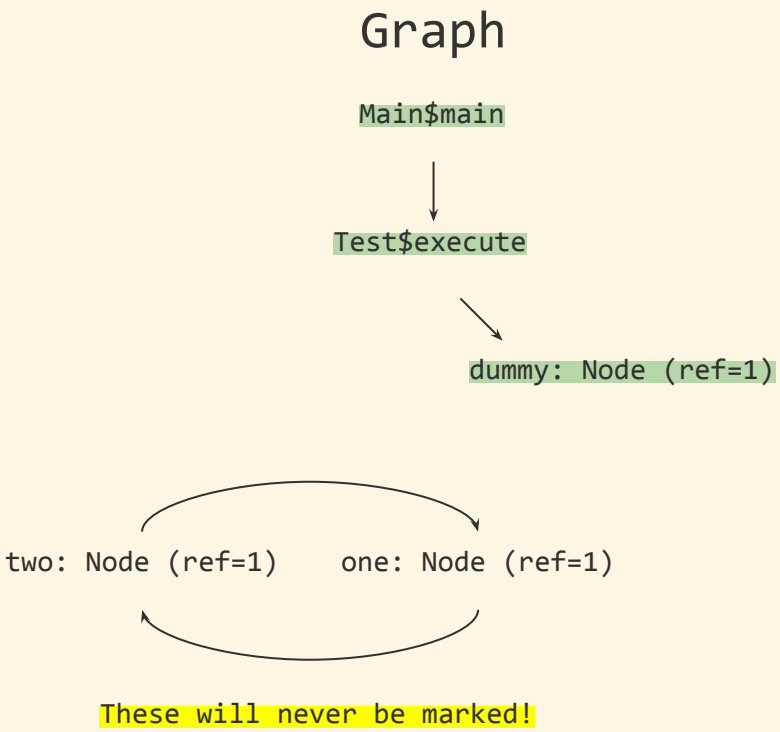
Implementation: Copying



Implementation: Copying



Implementation: Copying



Implementation: Copying

“From” slab

Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
one	0x25068	0x000C	No
two	0x25088	0x000C	No
(free space)	N/A	0xFFDB	N/A

Begin “copy” phase.

“To” slab

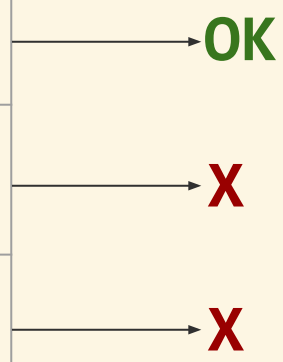
Name	Address	Size	Marked?
(free space)	N/A	0xFFFF	N/A

Implementation: Copying

“From” slab

Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
one	0x25068	0x000C	No
two	0x25088	0x000C	No
(free space)	N/A	0xFFDB	N/A

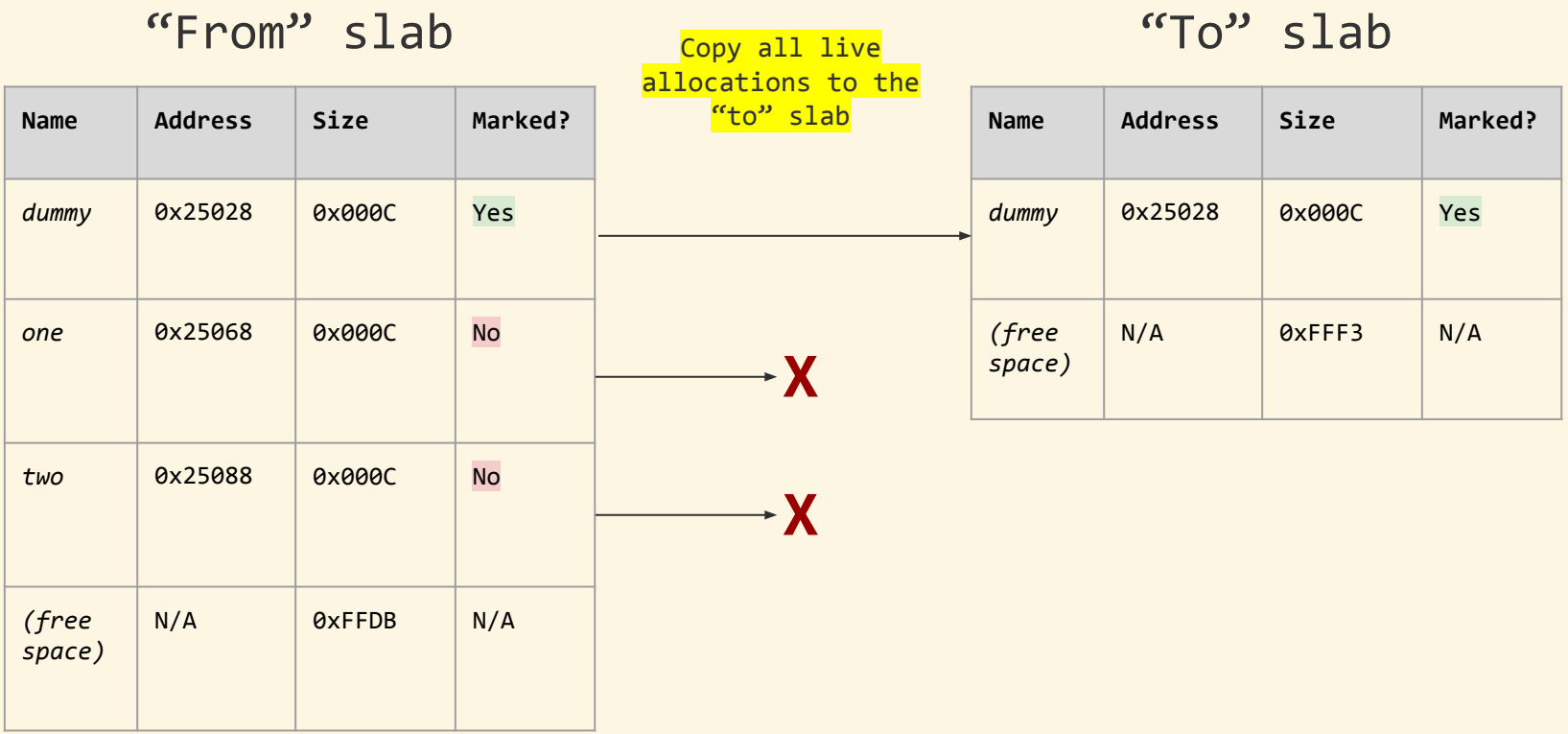
Copy all live allocations to the “to” slab



“To” slab

Name	Address	Size	Marked?
(free space)	N/A	0xFFFF	N/A

Implementation: Copying



Implementation: Copying

“From” slab

Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
one	0x25068	0x000C	No
two	0x25088	0x000C	No
(free space)	N/A	0xFFDB	N/A

Release allocations in the “from” slab that were just copied

“To” slab

Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
(free space)	N/A	0xFFF3	N/A

Implementation: Copying

“From” slab

Name	Address	Size	Marked?
one	0x25068	0x000C	No
two	0x25088	0x000C	No
(free space)	N/A	0xFFE7	N/A

Now only garbage

Release allocations in the “from” slab that were just copied

“To” slab

Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
(free space)	N/A	0xFFF3	N/A

Now only non-garbage

Implementation: Copying

“From” slab

Name	Address	Size	Marked?
one	0x25068	0x000C	No
two	0x25088	0x000C	No
(free space)	N/A	0xFFE7	N/A

Release allocations which still exist in the “from” slab

“To” slab

Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
(free space)	N/A	0xFFF3	N/A

Implementation: Copying

“From” slab

Name	Address	Size	Marked?
(free space)	N/A	0xFFFF	N/A

All garbage collected!

Release allocations which still exist in the “from” slab

“To” slab

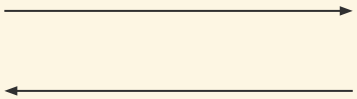
Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
(free space)	N/A	0xFFF3	N/A

Implementation: Copying

“From” slab

Name	Address	Size	Marked?
(free space)	N/A	0xFFFF	N/A

Swap handles to the “from” and “to” slabs



“To” slab

Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
(free space)	N/A	0xFFF3	N/A

Implementation: Copying

“From” slab

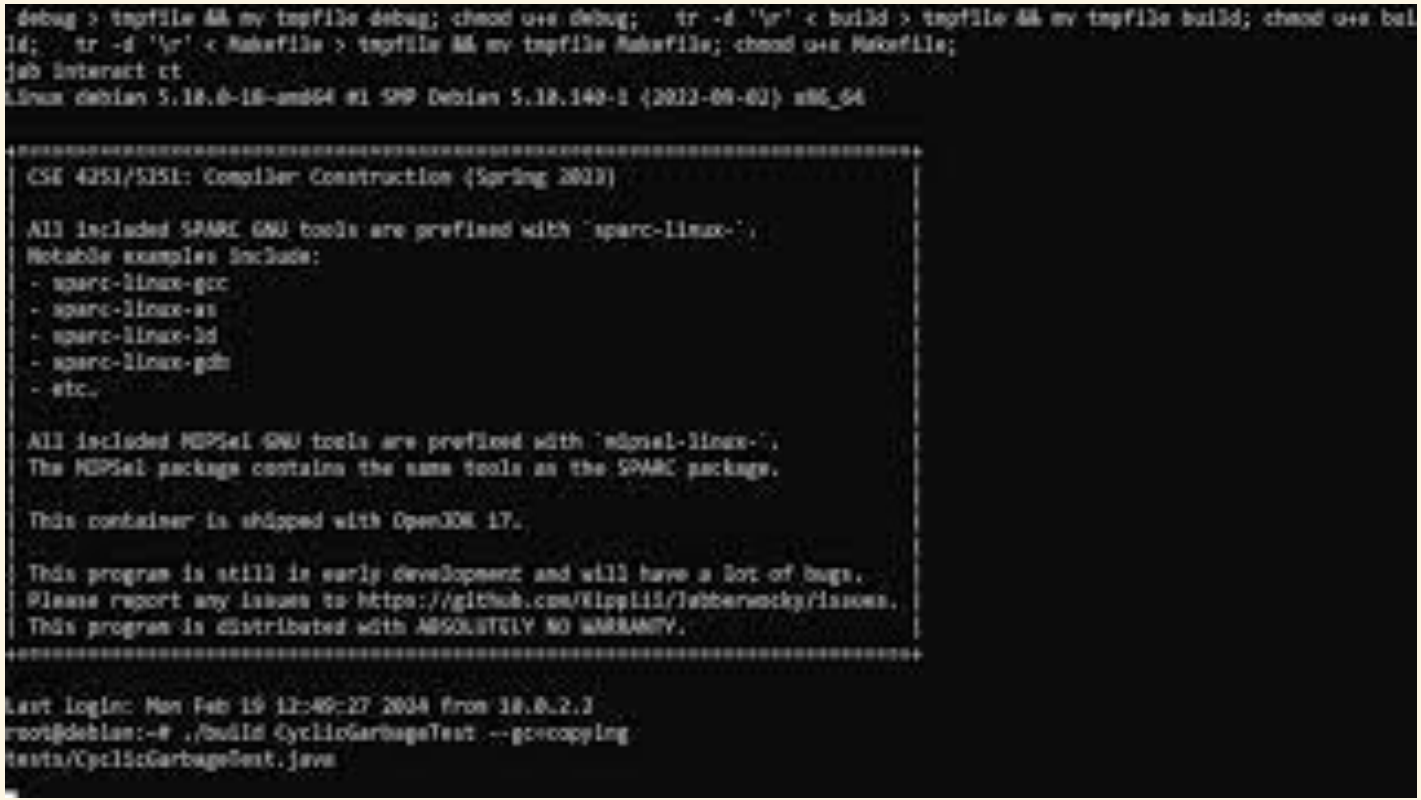
Name	Address	Size	Marked?
dummy	0x25028	0x000C	Yes
(free space)	N/A	0xFFF3	N/A

Garbage collection cycle is finished.

“To” slab

Name	Address	Size	Marked?
(free space)	N/A	0xFFFF	N/A

Demo: Copying



Implementation: Copying

```
root@debian:~# ./CyclicGarbageTest
[runtime/config.c:42] setting gctype to Copying
[runtime/marksweep.c:52] push_stack 0x40800660 (size:96)
[runtime/heap.c:143] alloc 0x25040 (size:4), userptr: 0x25048
[runtime/marksweep.c:52] push_stack 0x408005f8 (size:100)
[runtime/heap.c:143] alloc 0x2504c (size:4), userptr: 0x25054
[runtime/heap.c:143] alloc 0x25058 (size:4), userptr: 0x25060
[runtime/marksweep.c:52] push_stack 0x40800598 (size:92)
[runtime/marksweep.c:70] pop_stack
[runtime/marksweep.c:52] push_stack 0x40800598 (size:92)
[runtime/marksweep.c:70] pop_stack
[runtime/marksweep.c:70] pop_stack
[runtime/marksweep.c:144] search stack frame: 0x40800660
(size:96)
```

```
class CyclicGarbageTest {
    public int execute() {
        Node dummy;

        dummy = new Node();

        this.leak();
        System.gc();

        return 0;
    }

    public int leak() {
        Node one;
        Node two;

        one = new Node();
        two = new Node();

        one.setNext(two);
        two.setNext(one);

        return 0;
    }
}
```

Implementation: Copying

```
[runtime/marksweep.c:148]   sp->lreg[0]=408006BC
[runtime/marksweep.c:148]   sp->lreg[1]=00000000
[runtime/marksweep.c:148]   sp->lreg[2]=00000000
[runtime/marksweep.c:148]   sp->lreg[3]=00000000
[runtime/marksweep.c:148]   sp->lreg[4]=00000000
[runtime/marksweep.c:148]   sp->lreg[5]=00000000
[runtime/marksweep.c:148]   sp->lreg[6]=00000000
[runtime/marksweep.c:148]   sp->lreg[7]=000240C8
[runtime/marksweep.c:155]   sp->ioreg[0]=408006BC
[runtime/marksweep.c:155]   sp->ioreg[1]=00000000
[runtime/marksweep.c:155]   sp->ioreg[2]=00000000
[runtime/marksweep.c:155]   sp->ioreg[3]=00000000
[runtime/marksweep.c:155]   sp->ioreg[4]=00000000
[runtime/marksweep.c:155]   sp->ioreg[5]=00000000
[runtime/marksweep.c:162]   local_num=1 ← found dummy
[runtime/marksweep.c:192]   sp->locals[0 (align:0)] = 00025048
[runtime/marksweep.c:103] mark 0x25040
[runtime/marksweep.c:107] search alloced block 0x25048
[runtime/copying.c:126] copying 0x25040 from the "from" slab ← copying dummy
[runtime/heap.c:143] alloc 0x35080 (size:4), userptr: 0x35088  ← for copy
[runtime/slab.c:62] destroy: freeing something (0x450a8, begin->0x2504c)
[runtime/slab.c:62] destroy: freeing something (0x450f8, begin->0x25058)
[runtime/marksweep.c:70] pop_stack
[runtime/heap.c:101] alloced:
[runtime/heap.c:105]    addr:0x35080 size:4 ref:0 ← dummy was preserved
```

```
class CyclicGarbageTest {
    public int execute() {
        Node dummy;

        dummy = new Node();

        this.leak();
        System.gc();

        return 0;
    }

    public int leak() {
        Node one;
        Node two;

        one = new Node();
        two = new Node();

        one.setNext(two);
        two.setNext(one);

        return 0;
    }
}
```

Milestone 5 Goals

- Implement “generational” GC method
  - Relies on copying GC
  - Improves efficiency of GC cycles by less frequently trying to collect objects that could not previously be freed
- Conduct evaluation and analyze results
  - Compare implementations
  - Gather metrics and meaningful conclusions
- Create poster and ebook page for Senior Design Showcase
  - Begin preparing display material for the showcase