C Memory API

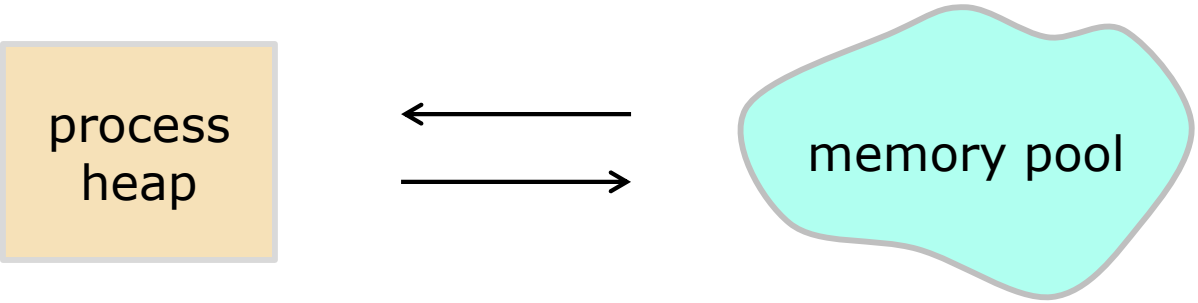
Glenn Bruns
CSUMB

Lecture Objectives

At the end of this lecture, you should be able to:

- Explain what garbage collection is
- Be able to use malloc() and free() correctly in C code

Dynamically-allocated memory

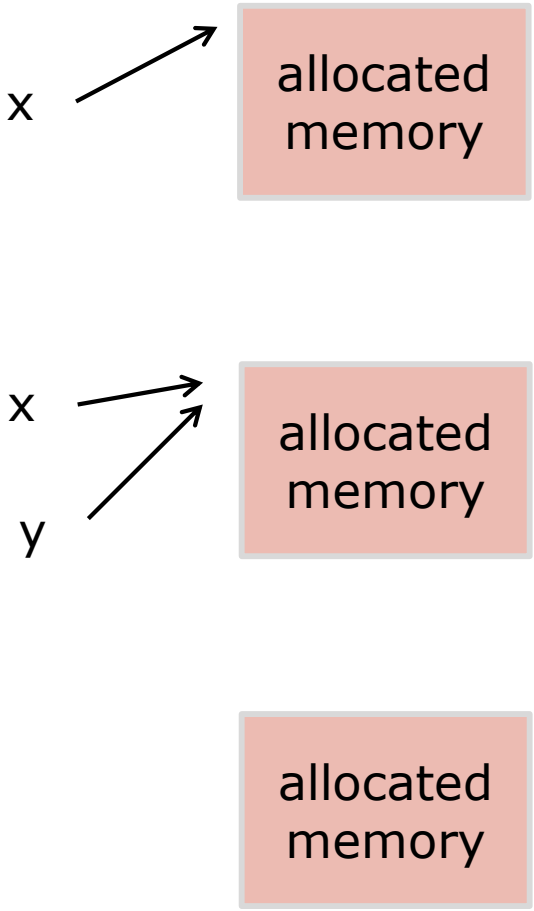


	Java	C
Allocating memory	List<Int> x = new ArrayList<Int>();	char *x = (char *)malloc(10 * sizeof(char));
Releasing memory	happens automatically	free(x);

Garbage Collection

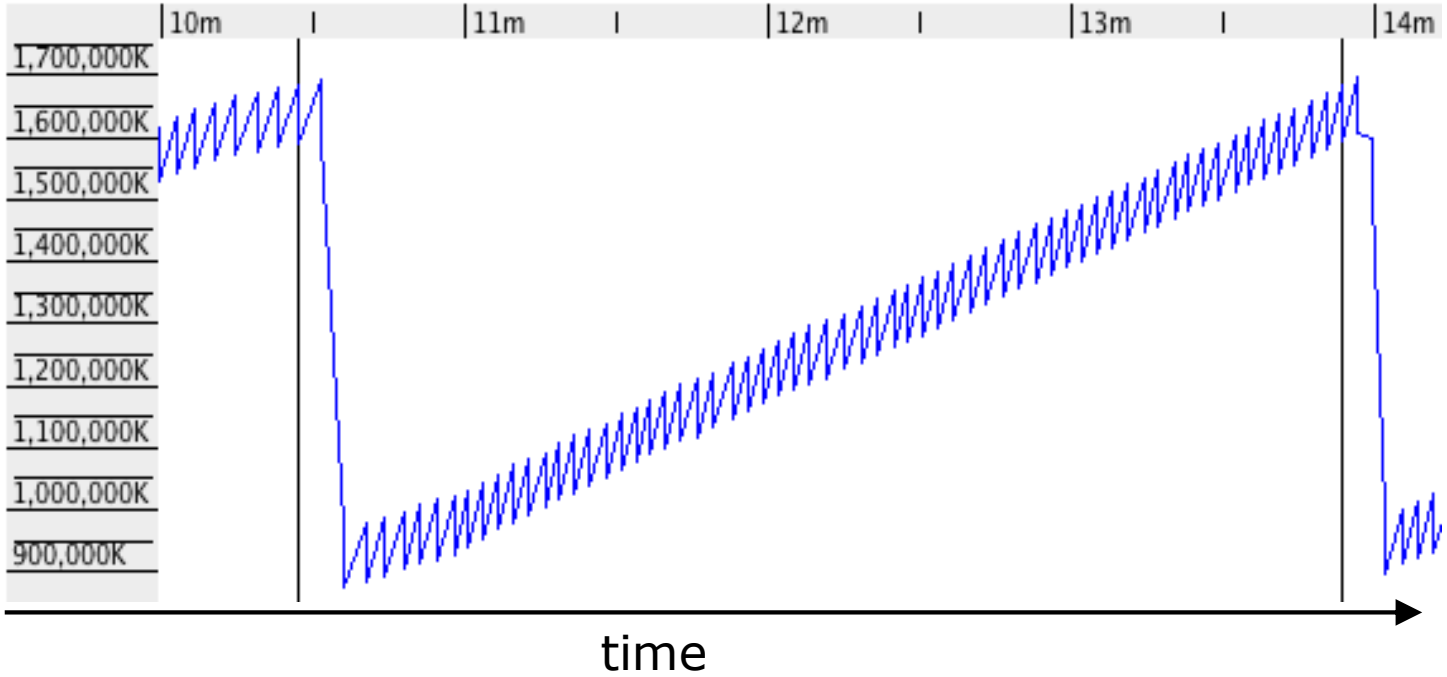
(Java code)

```
function foo() {
    // create x
    x = new ArrayList<String>();
    x.add("foo");

    y = x;
    y.add("bar");

    System.print(y);
}
```



Visualizing garbage collection

program heap size



Minor and major collections can be seen

diagram from: http://liveramp.com/engineering/adventures-in-java-garbage-collection-tuning/

Pros and cons of Garbage Collection

Pros

- Better programmer productivity
- Fewer program problems:
  - memory leaks
  - crashes caused by bad mallocs, frees

Cons

- Program responsiveness can be unpredictable

Memory allocation in C

```
void func() {
     int *x = (int *)malloc(sizeof(int));
     …
}
```

Exercise: is memory being allocated:

1. on the stack?
2. on the heap?
3. on both?

(example from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Calling malloc()

```
double *d = (double *)malloc(sizeof(double));

char *s = (char *)malloc(10);

char *s = (char *)malloc(strlen(t) + 1);
```

(example from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Calling free()

```
char *s = (char *)malloc(10);
…
free(s);
```

(example from Operating Systems: Three Easy Pieces, Arpaci-Dusseau & Arpaci-Dusseau)

Common mistakes: 1

```
char *s = “hello”;
char *t;
strcpy(t, s);
```

What is the problem?

forgot to allocate memory

Common mistakes: 2

```
char *s = “hello”;
char *t = (char *)malloc(strlen(s));
strcpy(t, s);
```

What is the problem?

didn’t allocate enough memory

Common mistakes: 3

```
char *s = “hello”;
char *t = (char *)malloc(strlen(s)+1);
printf(“t = %s\n”, t);
```

What is the problem?

didn’t initialize allocated memory

Common mistakes: 4

```
char *s = (char *)malloc(10);
char *t = (char *)malloc(10);
…
s = t;
```

What is the problem?

One of many ways to create a **memory leak**

A long-running program with small leaks will eventually use up all system memory.

Program analysis tools can discover memory leaks.

Summary

- In most modern languages, memory is allocated and deallocated automatically
- Garbage collection algorithms find unused memory and deallocate it
- In C, the programmer must explicitly allocate and deallocate memory
- Misery ensues: memory leaks, etc.

Bonus content: how does malloc work?

- It is a *dynamic storage allocation algorithm*
- OS has a pool of memory, some of which has been allocated
- Tries to match requests with blocks of available memory
- Strategies:
  - first fit
  - best fit
  - worst fit