Lecture #37

e side excursion into nitty-gritty stuff: Storage man-

Scope and Lifetime

eclaration is portion of program text to which it applies

be contiguous.

s static: independent of data.

extent of storage is portion of program execution dur-
exists.

ntiguous

dynamic: depends on data

xtent:

ntire duration of program

utomatic: duration of call or block execution (local vari-

From time of allocation statement (**new**) to dealloca-
y.

Explicit vs. Automatic Freeing

explicit means to free dynamic storage.

en no expression in any thread can possibly be influ-
change an object, it might as well not exist:

```
steful()

c = new IntList(3, new IntList(4, null));
.tail;
ble c now deallocated, so no way
t to first cell of list
```

t, Java runtime, like Scheme's, recycles the object c
arbage collection.

Under the Hood: Allocation

s (references) are represented as integer addresses.

to machine's own practice.

not convert integers ↔ pointers,

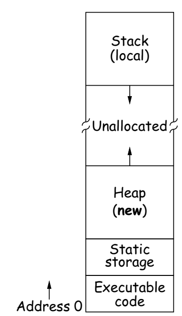
parts of Java runtime implemented in C, or sometimes
e, where you can.

tor in C:

```
[STORAGE_SIZE];  // Allocated array
inder = STORAGE_SIZE;

ter to a block of at least N bytes of storage */
leAlloc(size_t n) { // void*: pointer to anything
remainder) ERROR();
r = (remainder - n) & ~0x7;  // Make multiple of 8
void*) (store + remainder);
```


xample of Storage Layout: Unix



to turn chunks of unallocated region into heap.

omatically for stack.

Explicit Deallocating

lly require explicit deallocation, because of

n-time information about what is array

of converting pointers to integers.

n-time information about *unions:*

```
Various {
Int;
* Pntr;
le Double;
// X is either an int, char*, or double
```

all three problems; automatic collection possible.

ing can be somewhat faster, but rather error-prone:

orruption

eaks

Free List Strategies

uests generally come in multiple sizes.

ks on the free list are big enough, and one may have to chunk and break it up if too big.

tegies to find a chunk that fits have been used:

l fits:

cks in LIFO or FIFO order, or sorted by address.

e adjacent blocks.

for *first fit* on list, *best fit* on list, or *next fit* on list st-chosen chunk.

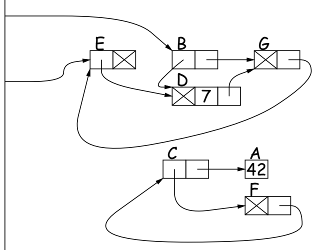
ed fits: separate free lists for different chunk sizes.

stems: A kind of segregated fit where some newly ad-e blocks of one size are easily detected and combined r chunks.

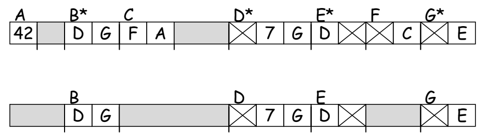
ocks reduces *fragmentation* of memory into lots of lit-d chunks.

rbage Collection: Mark and Sweep

atics)



1. Traverse and mark graph of objects.
2. Sweep through memory, freeing unmarked objects.



Copying Garbage Collection

roach: *copying garbage collection* takes time propor-unt of active storage:

the graph of active objects breadth first, *copying* them e contiguous area (called "to-space").

py each object, mark it and put a *forwarding pointer* t points to where you copied it.

time you have to copy an already marked object, just rwarding pointer instead.

e, the space you copied from ("from-space") becomes o-space; in effect, all its objects are freed in constant

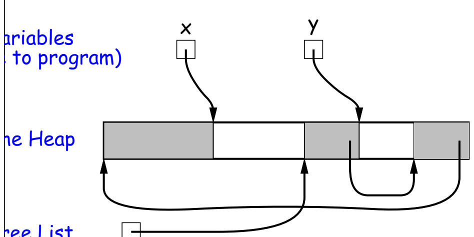
Free Lists

cator grabs chunks of storage from OS and gives to

ycled storage, when available.

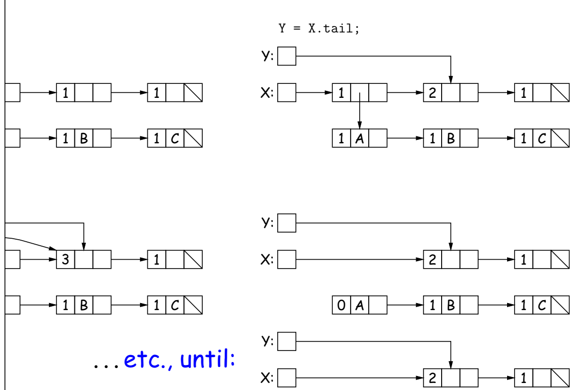
e is freed, added to a *free list* data structure to be

or explicit freeing and some kinds of automatic garbage



age Collection: Reference Counting

count of number of pointers to each object. Release oes to 0.



Cost of Mark-and-Sweep

eep algorithms don't move any exisiting objects—pointers e.

ount of work depends on the amount of memory swept—l amount of active (non-garbage) storage + amount of t necessarily a big hit: the garbage had to be active at d hence there was always some "good" processing in the h byte of garbage scanned.

jects Die Young: Generational Collection

bjects stay active, and need not be collected.

:e to avoid copying them over and over.

garbage collection schemes have two (or more) from for newly created objects (*new space*) and one for jects that have survived garbage collection (*old space*).

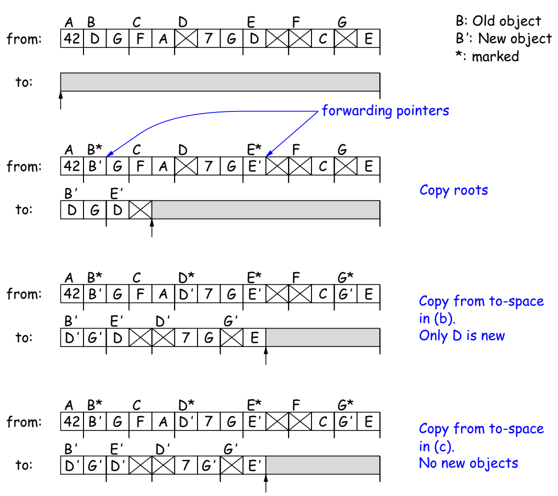
bage collection collects only in new space, ignores point- to old space, and moves objects to old space.

:s usual roots plus pointers in old space that have changed might be pointing to new space).

ace full, collect all spaces.

:h leads to much smaller *pause times* in interactive sys-

ying Garbage Collection Illustrated



There's Much More

st highlights.

on how to implement these ideas efficiently.

garbage collection: What if objects scattered over many

llection: where predictable pause times are important, *emental* collection, doing a little at a time.