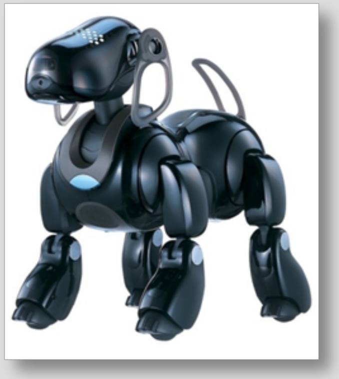
Linked lists and memory allocation



Prof. Noah Snavely

CS1114

http://cs1114.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- Assignment 3 has been posted
  - Due next Friday, March 6

- Prelim 1 Thursday in class
  - Review session this evening at 7:15pm, Upson 315
  - Review session tomorrow, 8:30pm?
  - Topics include: running time, sorting, selection, graphs, connected components, linked lists
  - Closed-book, closed-note

Administrivia

- Homework policy:
- You can discuss problems in general with other students
- You must write the code on your own – you may not share code

Bubble sort

- What is the running time?

- Which is faster?
  a) Bubble sort
  b) Quicksort

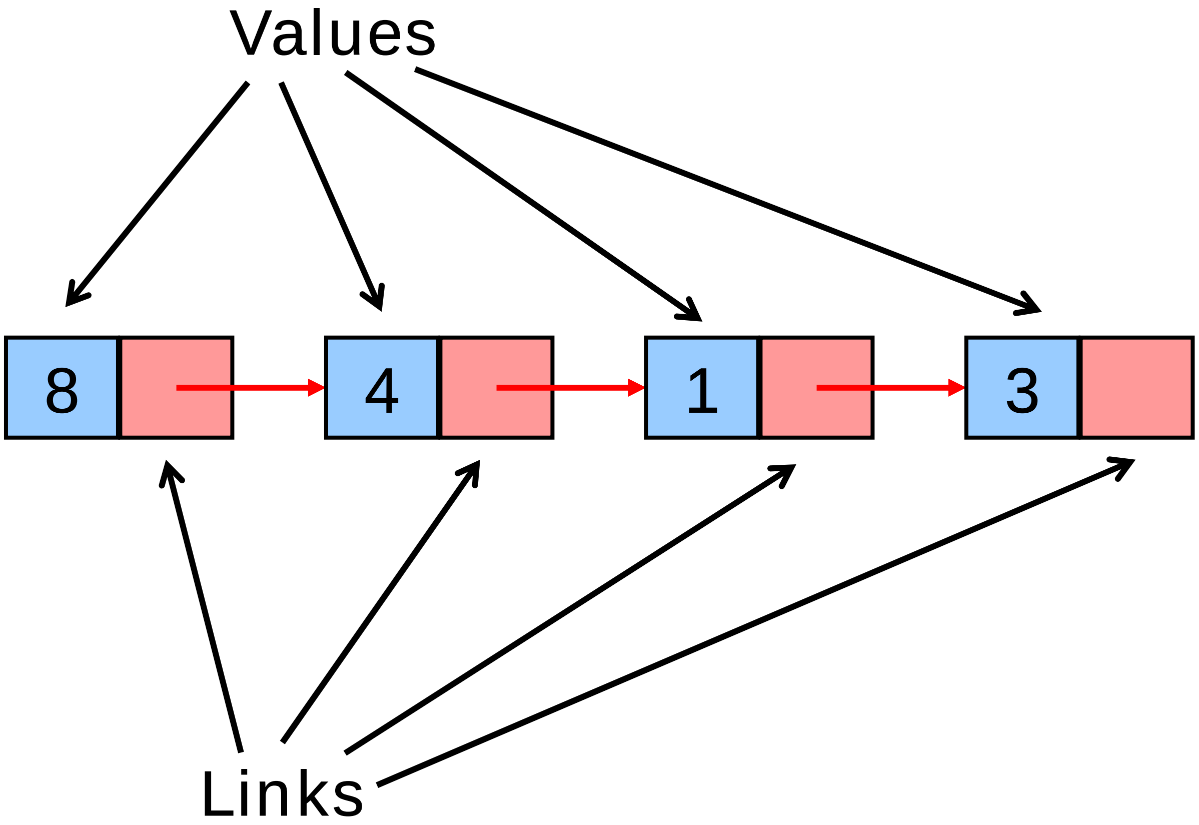
Linked lists

- Alternative to an array

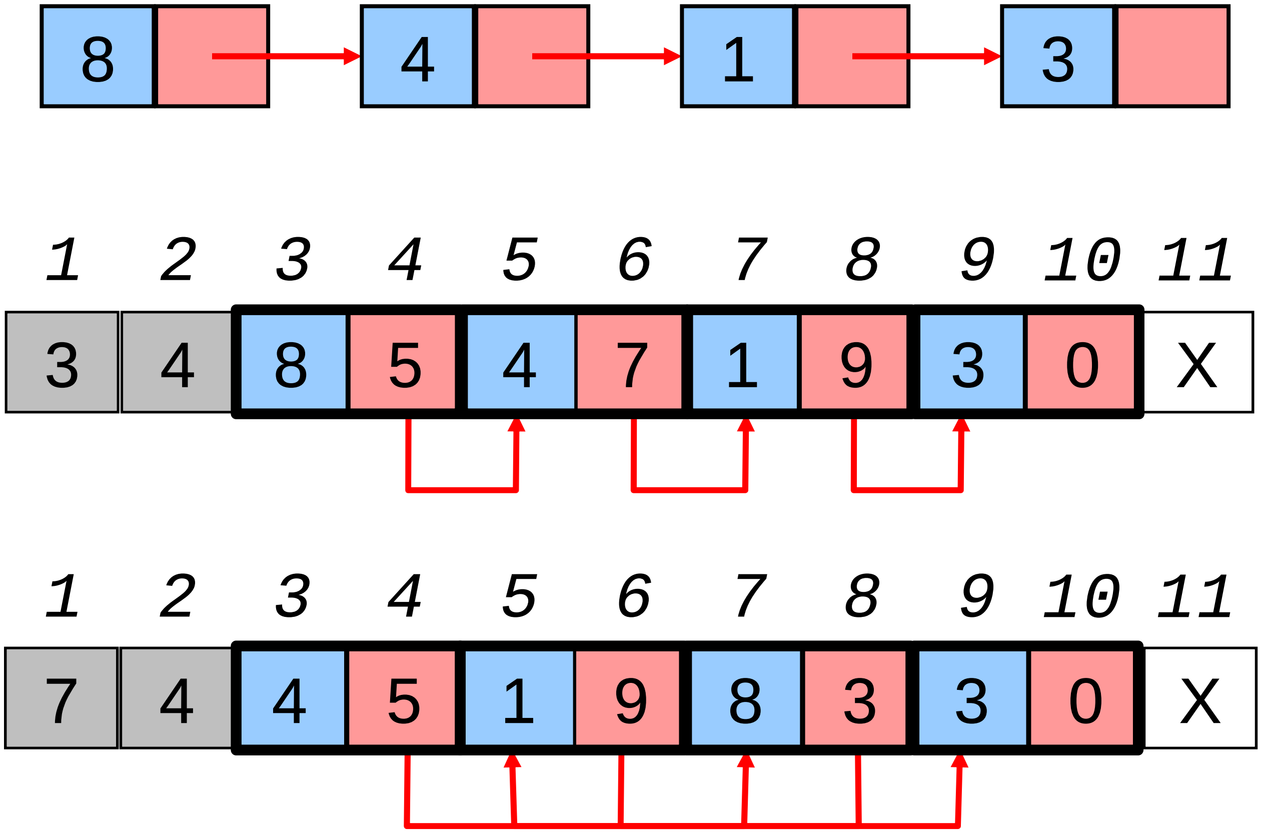
- Every element (cell) has two parts:
  1. A value (as in an array)
  2. A link (address, pointer) to the next cell

     This pointer will always point to the start of a cell

Linked lists



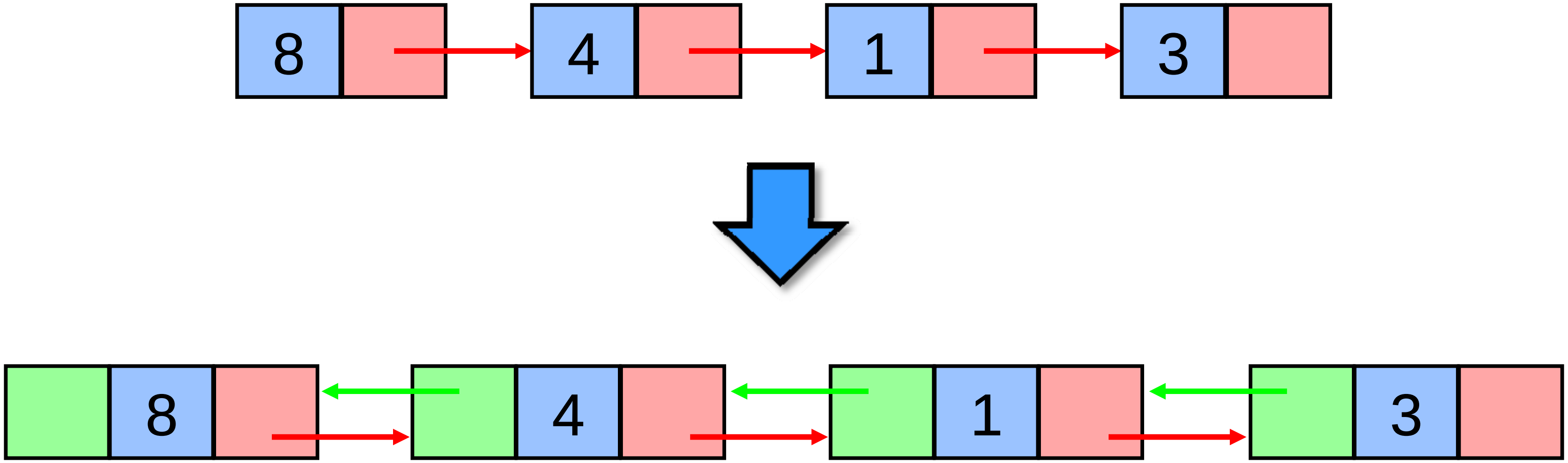
Example



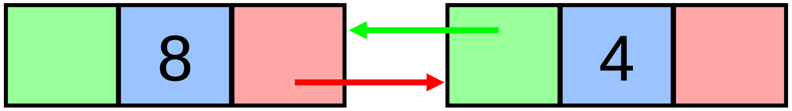
1	2	3	4	5	6	7	8	9	10	11
3	4	8	5	4	7	1	9	3	0	X

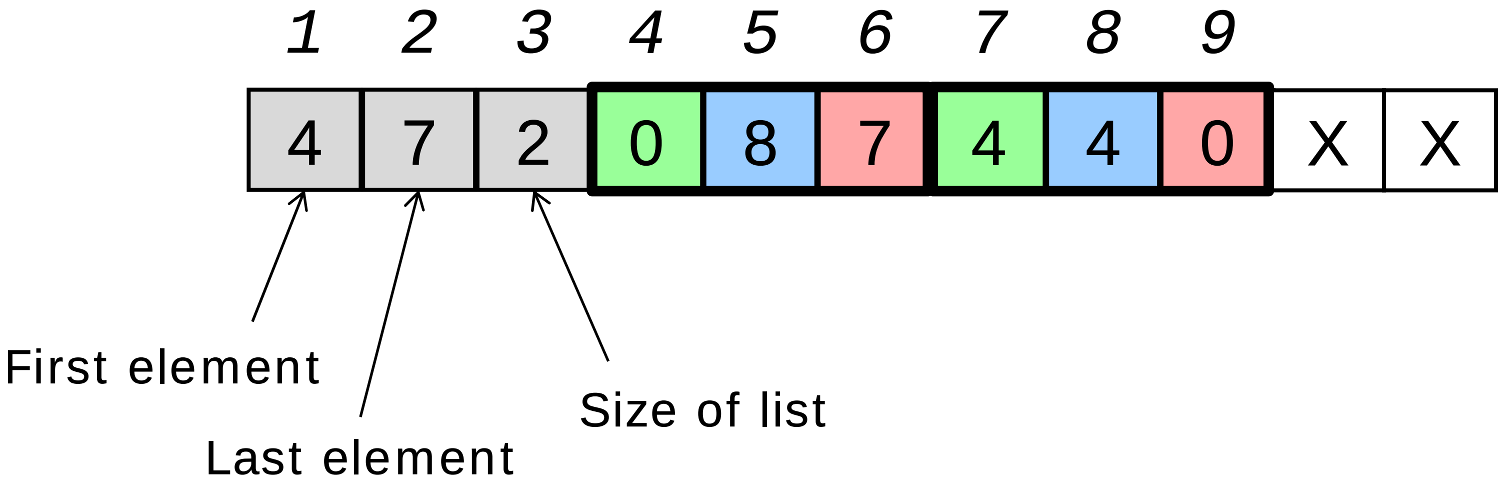
1	2	3	4	5	6	7	8	9	10	11
7	4	4	5	1	9	8	3	3	0	X

Doubly linked lists



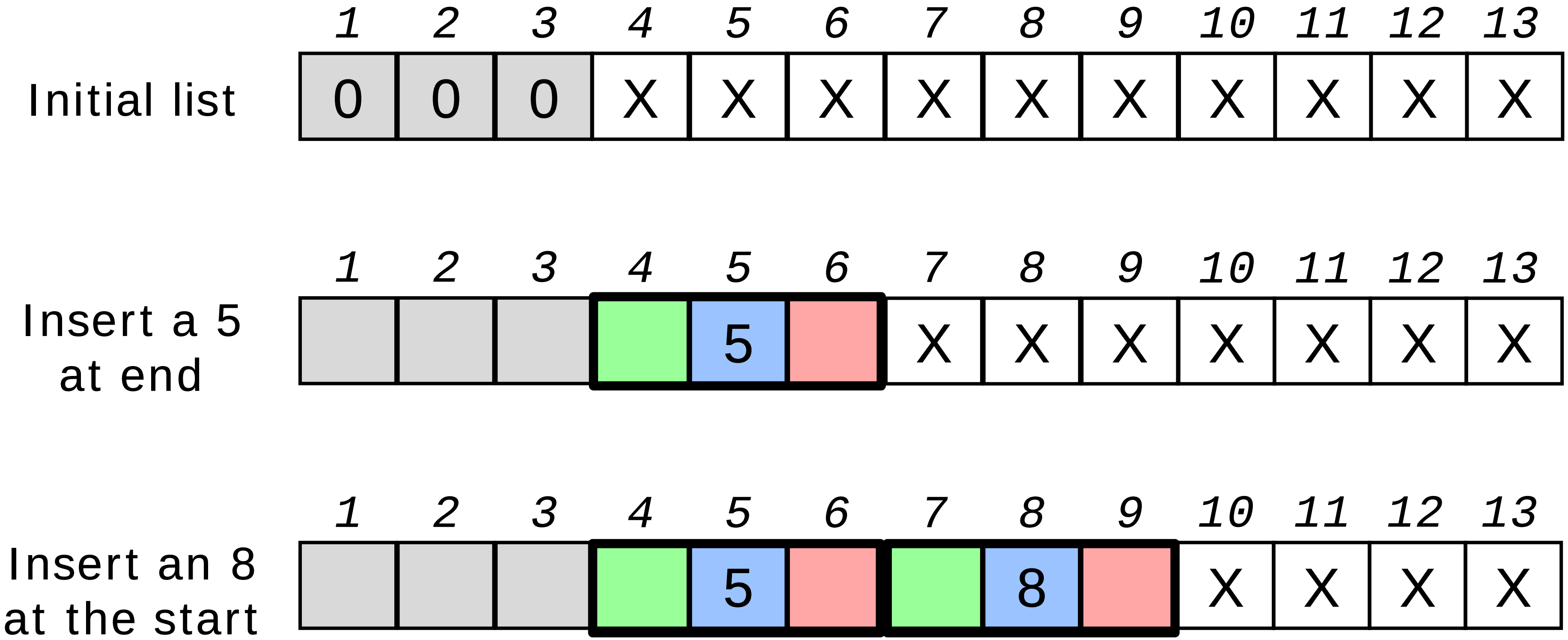
A doubly-linked list in memory





1	2	3	4	5	6	7	8	9		
4	7	2	0	8	7	4	4	0	X	X

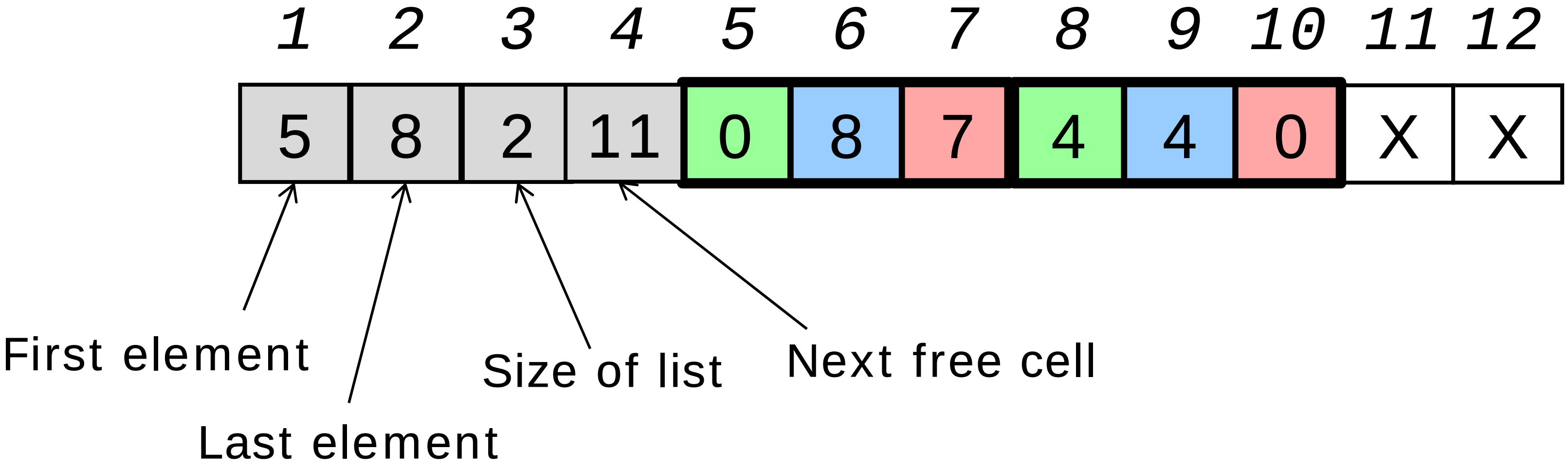
Doubly-linked list insertion



	1	2	3	4	5	6	7	8	9	10	11	12	13
Initial list	0	0	0	X	X	X	X	X	X	X	X	X	X
Insert a 5 at end				🟩	🟦 5	🟥	X	X	X	X	X	X	X
Insert an 8 at the start				🟩	🟦 5	🟥	🟩	🟦 8	🟥	X	X	X	X

Memory allocation

- When we need a new cell, how do we know where to find it?
- We’ll keep track of a “free pointer” to the next unused cell after the list



1	2	3	4	5	6	7	8	9	10	11	12
5	8	2	11	0	8	7	4	4	0	X	X

First element

Last element

Size of list

Next free cell

Doubly-linked list insertion

1	2	3	4	5	6	7	8	9	10	11	12	13	14
5	5	1	8	0	5	0	X	X	X	X	X	X	X

Insert a 9 at the start

1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	5	2	11	8	5	0	0	9	5	X	X	X	X

Delete the last element

1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	8	1		8	5	0	0	9	0	X	X	X	X

Memory allocation

- Current strategy: when we need more storage, we just grab locations at the end
- What can go wrong?

- When we delete items from a linked list we change pointers so that the items are inaccessible
  - But they still waste space!

Memory allocation

- Strategy 1: Computer keep tracks of free space at the end

- Strategy 2: Computer keeps a linked list of free storage blocks (“freelist”)
  - For each block, stores the size and location
  - When we ask for more space, the computer finds a big enough block in the freelist
  - What if it doesn’t find one?

Maintaining a freelist

1	2	3	4	5	6	7	8	9	10	11	12	13
7	4	2	7	5	0	0	9	4	X	X	X	X

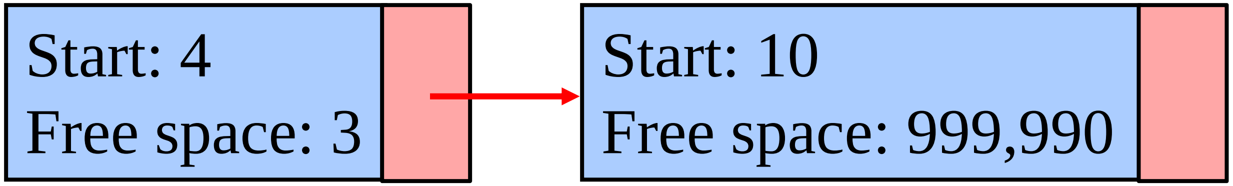
Free list

Start: 10
Free space: 999,990

Delete the last element

1	2	3	4	5	6	7	8	9	10	11	12	13
7	7	1	X	X	X	0	9	0	X	X	X	X

Free list


Start: 4
Free space: 3

Start: 10
Free space: 999,990

Allocation issues

- Surprisingly important question:
  - Which block do you supply?
  - The smallest one that the users request fits into?
  - A larger one, in case the user wants to grow the array?

Memory deallocation

- How do we give the computer back a block we’re finished with?

- Someone has to figure out that certain values will never be used ever (“garbage”), and should be put back on the free list
  - If this is too conservative, your program will use more and more memory (“memory leak”)
  - If it’s too aggressive, your program will crash (“blue screen of death”)

Memory deallocation

- Two basic options:

1. Manual storage reclamation
   - Programmer has to explicitly free garbage
   - Languages: C, C++, assembler

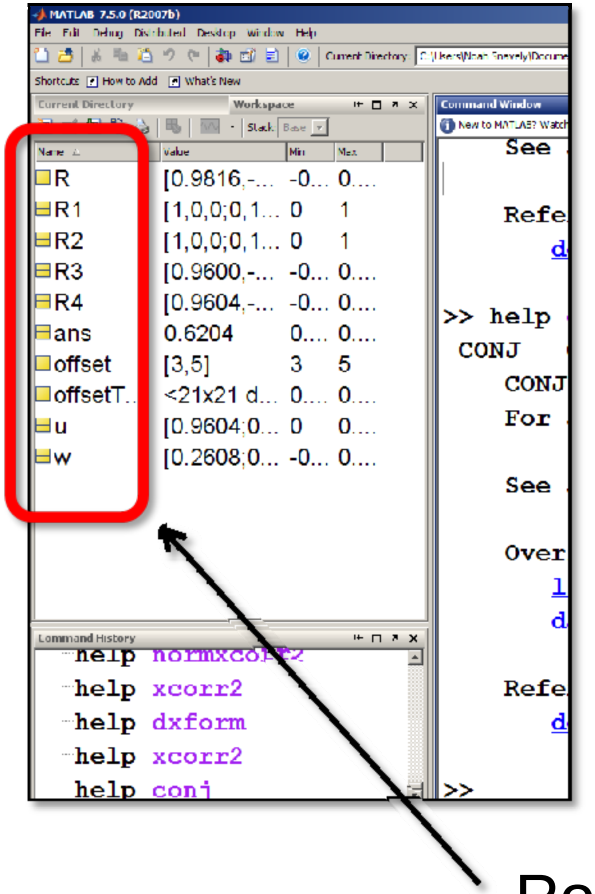
2. Automatic storage reclamation
   - Computer will notice that you're no longer using cells, and recycle them for you
   - Languages: Matlab, Java, C#, Scheme

Manual storage reclamation

- Programmers always ask for a block of memory of a certain size
  - In C, explicitly declare when it is free

- Desirable but complex invariants:
  1. Everything should be freed when it is no longer going to be used
  2. If we free something, we shouldn't try to use it again
  3. And, it should be freed exactly once!
  4. Minimize fragmentation

Automatic storage reclamation


MATLAB 7.5.0 (R2007b)
File Edit Debug Distributed Desktop Window Help
Current Directory
Shortcuts How to Add What's New
Current Directory Workspace
Stack: Base
Command Window
New to MATLAB? Watch
Command History

Name	Value	Min	Max
R	[0.9816,-...	-0...	0....
R1	[1,0,0;0,1...	0	1
R2	[1,0,0;0,1...	0	1
R3	[0.9600,-...	-0...	0....
R4	[0.9604,-...	-0...	0....
ans	0.6204	0....	0....
offset	[3,5]	3	5
offsetT...	<21x21 d...	0....	0....
u	[0.9604;0...	0	0....
w	[0.2608;0...	-0...	0....

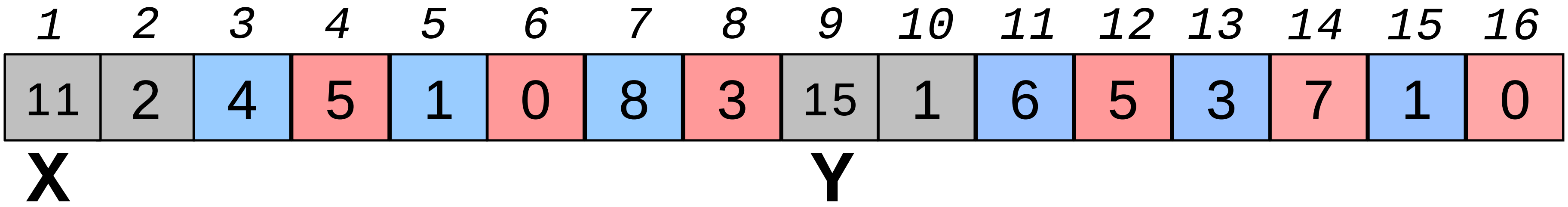
```
help normxcorr2
help xcorr2
help dxform
help xcorr2
help conj
```

```
   See
   Refe
      d
>> help
 CONJ
   CONJ
   For
   See
   Over
      l
      d
   Refe
      d
>>
```

Root set in Matlab

- "Garbage collection"
- 1st challenge: find memory locations that are still in use by the programmer ("live")
  1. Anything that has a name the programmer can get to (the "root set")
  2. Anything pointed to by a live object

Garbage collection



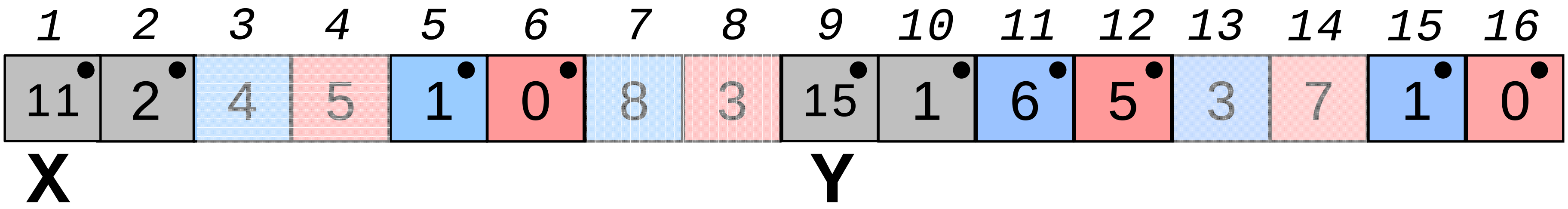
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
11	2	4	5	1	0	8	3	15	1	6	5	3	7	1	0
X								**Y**							

- Two lists, X and Y
- Which cells are live?
- Which cells are garbage?

Simple algorithm: mark-sweep

- Mark: Chase the pointers from the root set, marking everything as you go
- Sweep: Scan all of memory – everything not marked is garbage, and can go back on the free list

Mark and sweep



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
11 ●	2 ●	4	5	1 ●	0 ●	8	3	15 ●	1 ●	6 ●	5 ●	3	7	1 ●	0 ●
X								Y							

Root set: { X, Y }

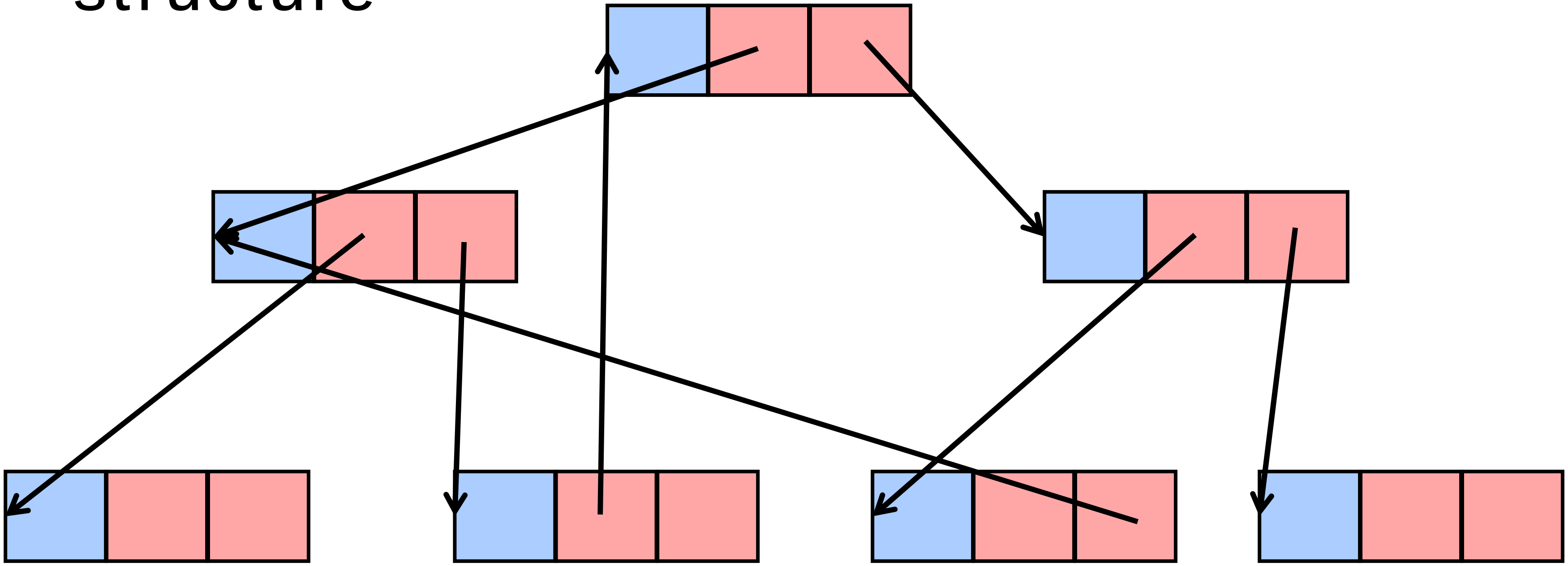
- Mark phase
- Sweep phase

Mark and sweep

- The machine needs to be able to tell where the pointers are (we’ll assume that it’s up to the programmer to do that)
  - For instance, the programmer will say that the second entry in a cell is a pointer (for singly-linked list)
  - Or, for a doubly-linked list, the first and third entries in a cell are pointers

Mark and sweep

- In general, pointers may have a complex structure



How do we mark, in the general case?

When to do garbage collection?

- Option 1 (“stop the world”): Once memory is full, stop everything and run garbage collection
  - Your program will freeze while the garbage is being collected
  - Not good if you’re coding the safety monitoring system for a nuclear reactor

- Option 2 (“incremental GC”): Collect garbage in parallel with the main program
  - Needs to be careful not to step on the program

Assignment 3

- Implementing stacks and queues using linked lists
- Using DFS and BFS to find connected components
- Guiding the robot with the lightstick