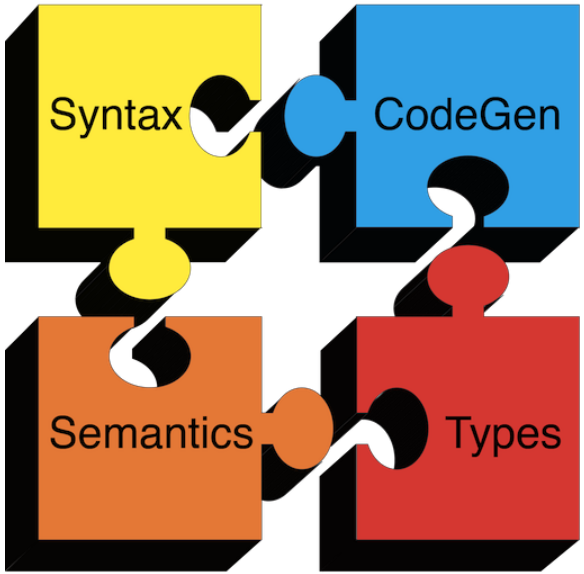


Compilers

Reference Counting

Reference Counting

- Rather that wait for memory to be exhausted, try to collect an object when there are no more pointers to it
- Store in each object the number of pointers to that object
  - this is the reference count

- Each assignment operation manipulates the reference count

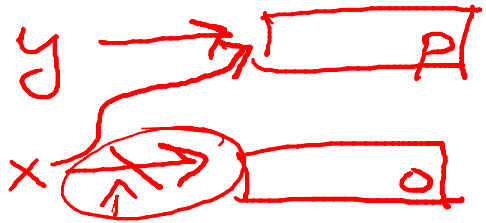
Reference Counting

- new returns an object with reference count 1
- Let rc(x) be the reference count of x

- Assume x, y point to objects o, p

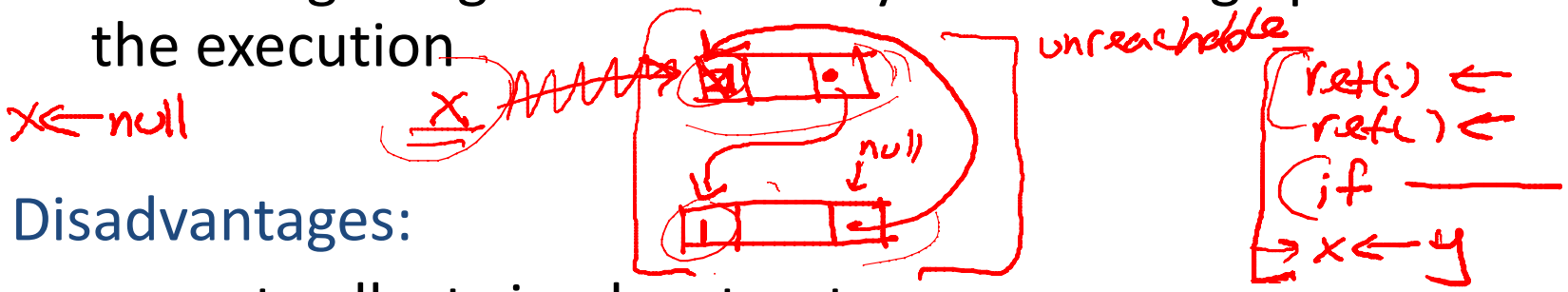
- Every assignment $x \leftarrow y$ becomes:

  $rc(p) \leftarrow rc(p) + 1$
  $rc(o) \leftarrow rc(o) - 1$
  if($rc(o) == 0$) then free o
  $x \leftarrow y$

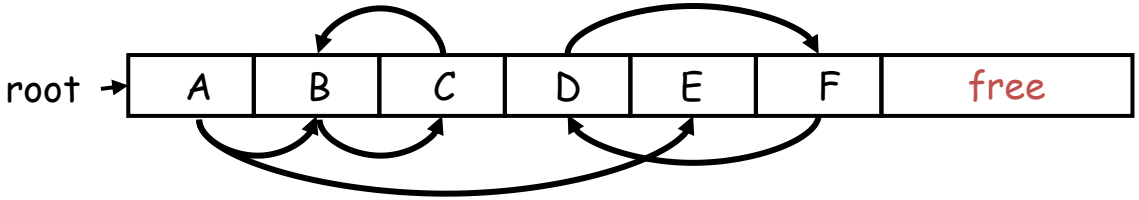


Reference Counting

- Advantages:
  - – easy to implement
  - – collects garbage incrementally without large pauses in the execution
- Disadvantages:
  - – cannot collect circular structures
  - – manipulating reference counts at each assignment is very slow



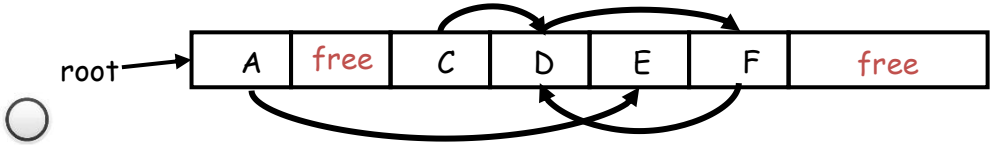
Reference Counting

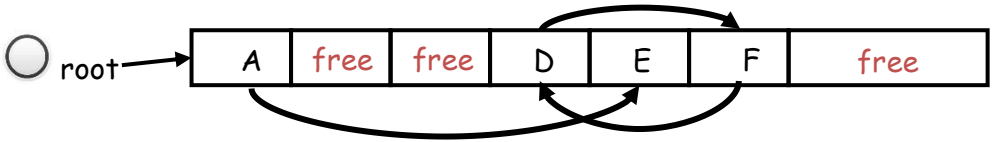


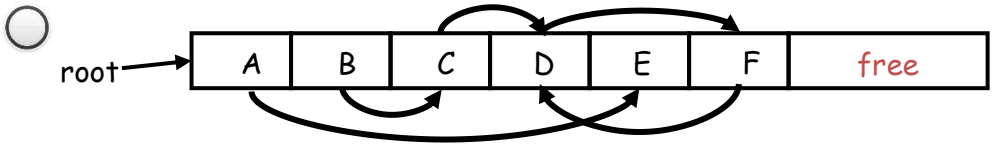
Choose the final heap after executing the following two assignments and updating reference counts:

C.ptrToB = D

A.ptrToB = NULL

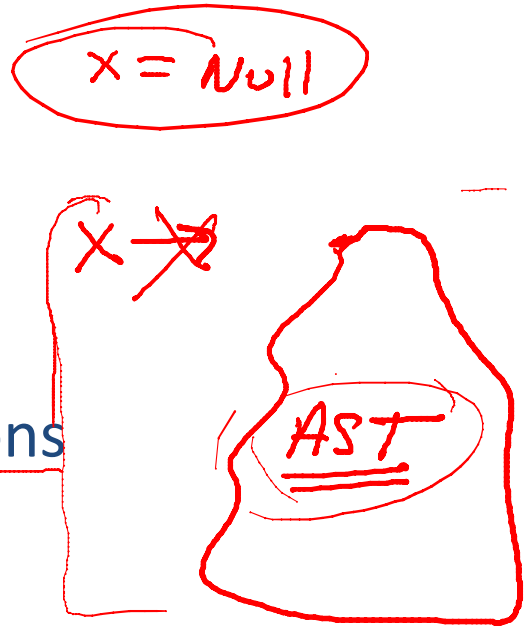






Reference Counting

- Automatic memory management prevents serious storage bugs
- But reduces programmer control
  - e.g., layout of data in memory
  - e.g., when is memory deallocated
- Pauses problematic in real-time applications
- Memory leaks possible (even likely)



Reference Counting

- Garbage collection is very important

- There are more advanced garbage collection algorithms:
  - concurrent: allow the program to run while the collection is happening
  - generational: do not scan long-lived objects at every collection
  - real time: bound the length of pauses
  - parallel: several collectors working at once