Unobtrusive Garbage Collection

GC Requirements:

- **batch programs:** We want short total GC time.
- **interactive programs:** We want unnoticeable GCs.

Unobtrusive GC:

- Incremental Collection
  - Do a little GC-work every time an object is allocated, or a pointer is changed.
- Concurrent Collection
  - Run the collector and the program in different processes, or on different processors.

Incremental GC

- Use *copying collection*, but rather than stop when you run out of memory and then do all the GC work in one shot, do a little bit whenever a pointer variable is referenced or when a new object is allocated.
- We start out by forwarding (copying) the objects pointed to by global variables.
- Then, instead of continuing forwarding recursively, we resume the program.
- Every time a pointer is referenced we check to see whether it is pointing into from-space. If it is, we forward that object too.
Incremental GC...

Even objects which are not explicitly referenced have to be checked, to see if they have become garbage. Therefore, every time we allocate a new object we forward $k$ pointers. A good value for $k$ has to be determined by experimentation.

Eventually scan will catch up with next and we switch from-space and to-space and start an new cycle.

Baker's algorithm (on the next slide) is a variant of copying collection.

Exam Problem

1. Why is generational collection more appropriate for functional and logic languages (such as LISP and Prolog), than for object-oriented languages (such as Eiffel and Modula-3)?

2. The heap in the figure on the next slide holds 7 objects. All objects have one integer field and one or two pointer fields (black dots). The only roots are the three global variables $X$, $Y$, and $Z$. Free space is shaded. Show the state of To-Space after a copying garbage collection has been performed on From-Space. Note that several answers are possible, depending on the visit strategy (Depth-First or Breadth-First Search) you chose.

Incremental GC...

1. Copy and update objects pointed to by global pointers to to-space.
2. Resume program.
3. When an object in from-space is referenced, first copy it to to-space.
   
   \[
   p := x^\uparrow\.next;
   \]
   
   (implemented as)
   
   \[
   \text{IF } x \in \text{from-space } \text{THEN}
   \]
   
   copy $x$ to to-space;
   update $x$, scan, and next;
   $x := x$’s new address in to-space;
   
   END;
   
   $p := x^\uparrow\.next$;

4. Every time NEW is called, $k$ pointers are forwarded.

Exam Problem I...

```latex
\begin{abstract}

1. Why is generational collection more appropriate for functional and logic languages (such as LISP and Prolog), than for object-oriented languages (such as Eiffel and Modula-3)?

2. The heap in the figure on the next slide holds 7 objects. All objects have one integer field and one or two pointer fields (black dots). The only roots are the three global variables $X$, $Y$, and $Z$. Free space is shaded. Show the state of To-Space after a copying garbage collection has been performed on From-Space. Note that several answers are possible, depending on the visit strategy (Depth-First or Breadth-First Search) you chose.

\end{abstract}
```
Exam Problem...

1. Name five garbage collection algorithms!
2. Describe the Deutsch-Schorr-Waite algorithm! When is it used? Why is it used? How does it work?
3. What are the differences between stop-and-copy, incremental and concurrent garbage collection? When would we prefer one over the other?

Readings and References

- Read Scott, pp. 395–401.
- Aho, Hopcroft, Ullman. Data Structures and Algorithms, Chapter 12, Memory Management.

Readings and References...