Promotion policies

How exactly do we promote objects

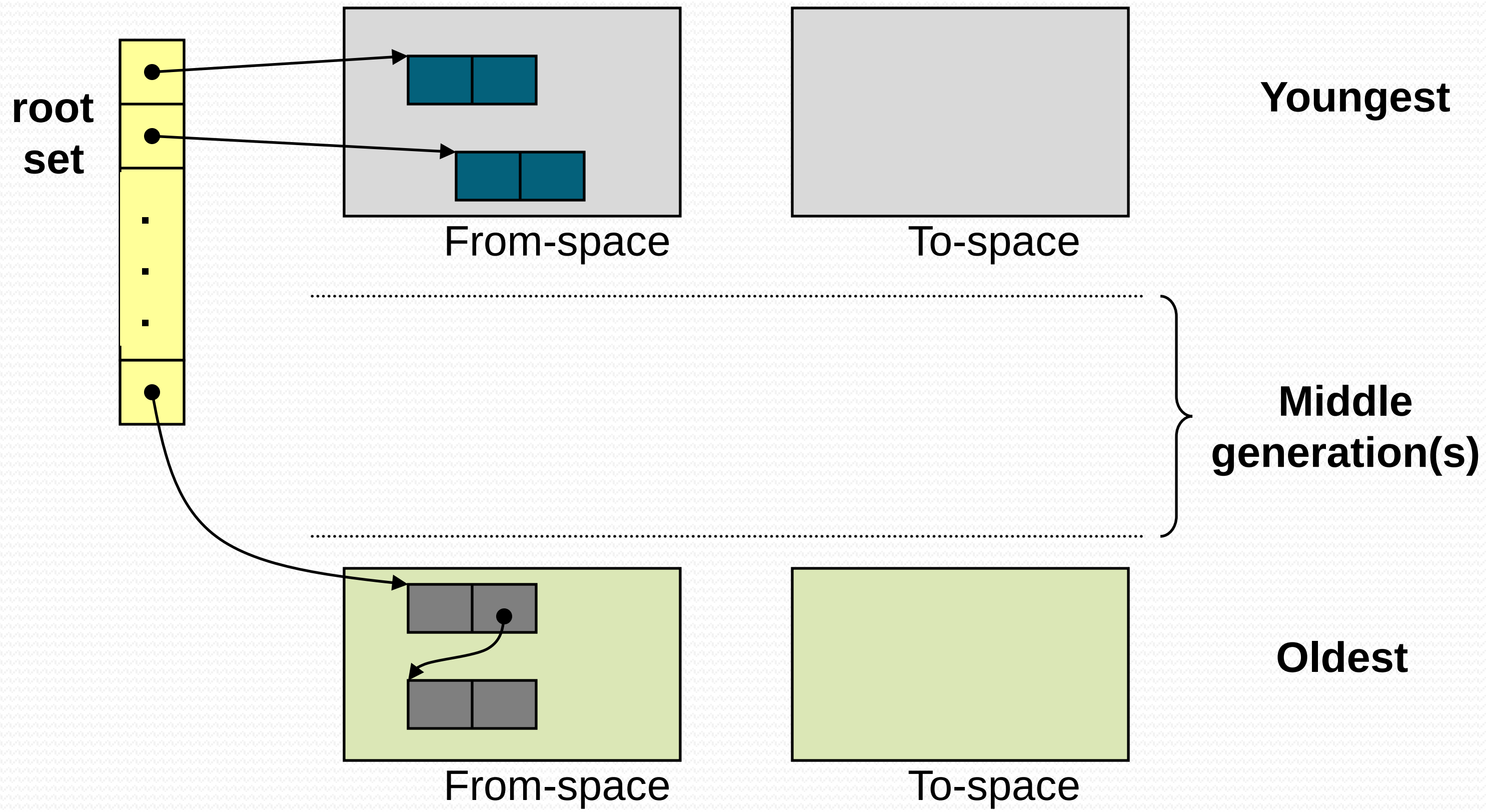
Benefits of generational collection

- Benefits:
  - Collect only a part of the heap
  - Pause time diminish
  - GC becomes feasible for interactive systems
    - “Can I garbage collect while tracking the mouse?”
  - Avoid repeatedly processing objects that remain alive
  - Overall effort of GC can be reduced
  - Locality of the collector can be improved

Cost of generational collection

- Cost:
  - System must be able to distinguish old from young objects
  - Cost associated with storing in old object pointer to young object can be very expensive

Generational copy collector



Inter-generational pointers

- Created in 2 ways
  - storing pointers in object (assignment)
  - Object containing pointers promoted to older gen.
- Burden on mutator or collector to track
  - Promotion: can be easily tracked by collector
  - Assignment: need write barrier to trap and record
    - Recall most stores are in local variables
    - Only need to record old-young pointers, Why?
      - They are rare
      - They become roots for minor collection

Goals of generational collection

- Aims of generational GC:
  - Reduce cost of dealing with long lived objects
  - Reduce garbage collection pause time
    - Interactive program test
    - Depends on amount of data that survives a collection
    - Depends on size of generation
      - small → more frequent collection
      - Large → less frequent collection
  - Achievable by segregating objects by age

Effects of premature promotion

- Objects should not be promoted prematurely
  - Basis of generational GC is to allow as many objects as possible to die in youngest generation
- Need promotion threshold
  - If too low:
    - promote soon-to-be tenured garbage
    - Old generation fills quickly → major collection
    - Major collection → longer pause
    - More inter-generational pointers
    - What about write-barrier cost?

What policies to use for promotion?

- Multiple generations
- Promotion threshold
- Adaptive tenuring (promotion)

Multiple generations

- Two generations offer
  - Reduced pause time
  - Reduce copying overhead
- What about multiple generations?
  - Filter objects prematurely promoted from youngest gen.
  - Increase chances that they will die before promotion to oldest generation
  - Fill up more slowly than youngest generation
  - Will be collected less often

Multiple generation: Other effects

- Allow new objects to be promoted quickly
- Keeps youngest generation fairly small
  - Reduces pause incurred when scavenging it
- Does not increase volume of permanent garbage

Multiple generations: limitations

- Pause time for collecting intermediate generation may still be disruptive
- More pointers from old to younger generations will be created
- Size of root set for younger generations increases

Promotion threshold

- Promotion rates also depend on number of minor collections object must survive before promotion
  - Copy count of 1 → *en masse* promotion even though some objects are extremely young
    - Leads to promotion rates that are 50% to 100% higher
  - Copy count of 2 has following properties
    - Denies promotion to recently created objects
    - Highly effective
    - Reducing survivors by a factor of 2 while increasing copy cost by < ½
  - Beyond 2 produces very little benefit

Dilema for fixed promotion policies

- Consider small youngest generation
  - Shortens interval between scavenges
  - Shortens pause length
- Consider larger generations
  - Reduces promotion rates
  - Gives objects longer to die
  - Scavenges less often → copying overhead is reduced
    - But pause length is increased
- So how does fixed promotion policies handle this dilema?

Adaptive tenuring

- Tuning generational collection is complex and time consuming
- What if program has varying allocation rates?
  - Fixed policies does not have a way to adjust tenure rate and prevent collector from thrashing
- Adaptive tenuring:
  - Invoke collector when volume of data allocated since last collection exceeds an allocation threshold
  - Dynamically vary size of semi-spaces if necessary
  - Threshold-based policy more stable than fixed-size generation policy

Two flavors of adaptive tenuring

- Only tenure when it is necessary
- Only tenure as many objects as necessary

- Objects' age given in bytes allocated
  - More memory allocated since object creation → older object
  - Less memory allocated since object creation → younger object
- Pause time given as bytes copied

Tenure only when necessary

- # of objects that survive a scavenge is used to predict pause time of next scavenge
  - Definition of pause time
  - Time measured in bytes
  - If few objects survive a scavenge (less than threshold)
    - Probably not worth promoting them
    - GC pause less than max acceptable pause
    - Consider write-barrier cost

Tenure # of objects as necessary

- If survivor size suggests maximum pause time (in bytes) would be exceeded at next scavenge
  - Set age threshold to value to allow excess data to be promoted
  - Survivors scanned to produce table recording volume of object of each age
  - Table then scanned (descending order) to look for promotion threshold for next minor collection

Pioneers of adaptive tenuring

- Ungar and Jackson → feedback mediation
- Barett and Zorn → threaten boundary and remembered set
- Next class
  - Generation organization
  - Age recording
  - Inter-generational pointers