

Non-copying Generational Collection, Scheduling GC

Not every generational GC is a copying generational GC. Knowing when to schedule GC can also impact performance

Which pointer writes to trap

- Writes that may not need to be trapped
 - Stores to registers
 - Stores to stack
 - Initializing stores
 - Not easy to detect in many languages
 - These stores form the majority of pointer stores
- Stores that should be trapped
 - Non-initializing stores
 - ~ 1 % of instructions generated by Lisp or ML compilers

Trapping & recording inter-gen ptrs

- Entry tables
- Remembered sets
- Sequential store buffer
- Page marking with hardware support
- Page marking with virtual memory support
- Card marking

Card marking

- Divide heap into # regions called cards
 - Smaller than pages (~ 128 Bytes)
 - Requires card table
 - Bit is set in card table for a card whenever a word in the card modified
 - Scans dirty cards for inter-generational ptrs at collection time
 - Advantages
 - Portable
 - Independent of virtual memory system
 - Flexible → card size can be picked to optimize locality

Card marking

- Cost of scanning
 - Proportional to number of cards marked
 - Proportional to size of card
- GC can use dirtiness info to
 - Segregate objects on written-to-cards from clean cards
 - Gather dirty cards on same virtual memory page
 - # pages holding cards to be scanned is reduced

Non-copying generational GC

- Mark-sweep based generational GC
 - Yields no worse performance than copying base collectors
- Zorn's system
 - Four generations, each containing a
 - mark bitmap
 - fixed-sized-object region
 - variable-sized-object region

Zorn's fixed-sized-object region

- Divided into # areas
- Each area stores objects of a fixed size
- Uses mark-and-deferred-sweep GC
- Promotes objects en masse by copying, after 3 collections

Zorn's variable-sized object region

- Holds object that cannot fit in any of the fixed-sized object areas
- Collects with two semi-space copying collector
- Zorn's finding
 - Mark-sweep collector suffered from greater CPU overhead
 - Mark-sweep collector required 30% to 40% less real memory than the copying collector
 - Promotion thresholds, pause times, cache miss ratio were important

When do we schedule GC

- Two options:
 - Hide collections at times when user least likely to notice pauses
 - Trigger efficient collections when there is likely to be most garbage collected

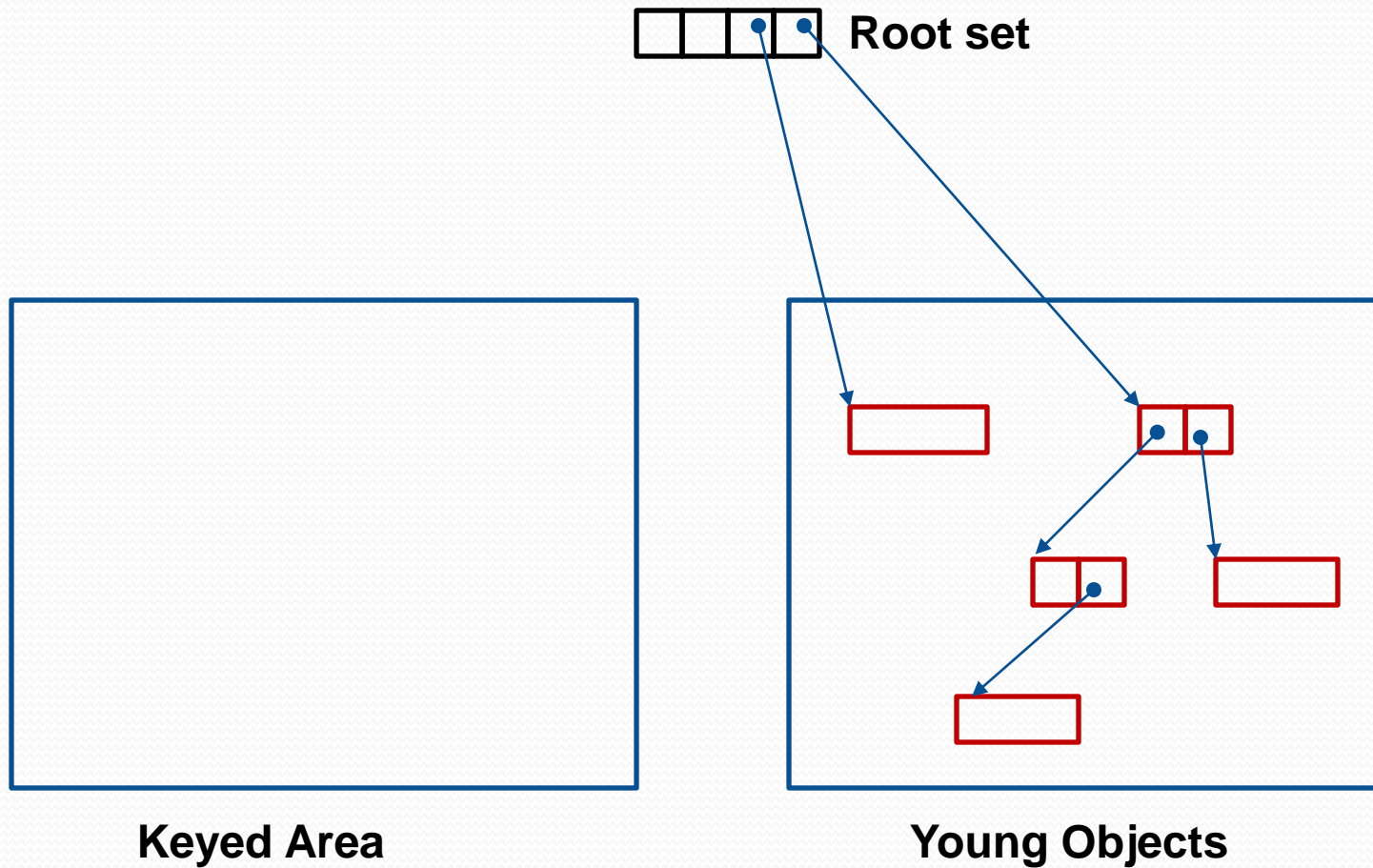
Hiding garbage collection

- In long lived systems
 - Perform GC overnight
 - Perform GC when machine is idle
- Perform GC at points in program when pause is less likely to be disruptive
 - End of compute-bound periods
 - Volume of live data is low
 - User given opportunity to interact with program and does not do so
 - Can attach code to monitor user interaction (good heuristic)
 - Emacs uses this strategy

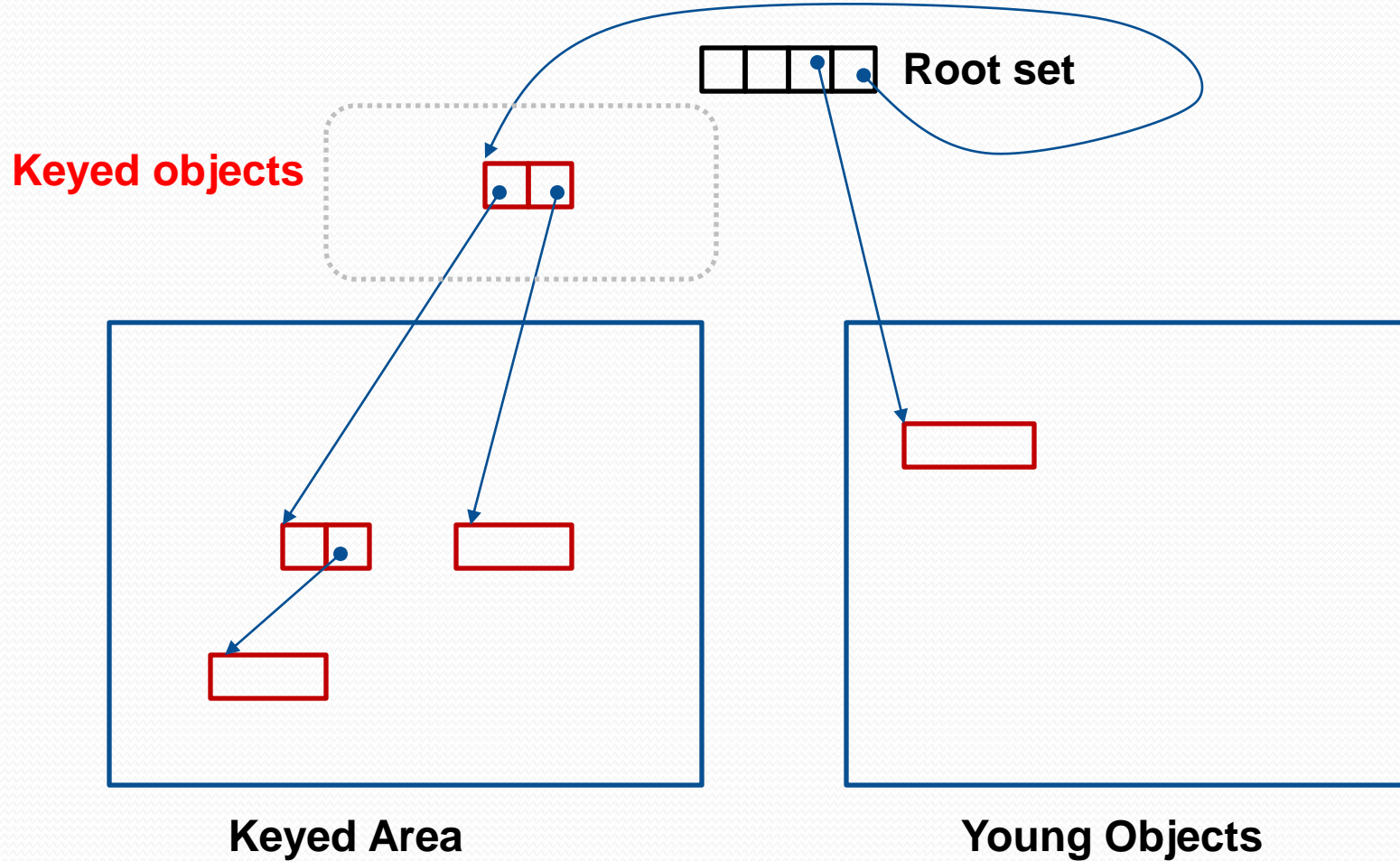
Key objects as indicators

- Hayes observed that death of objects allocated at roughly the same time closely correlate
 - Object demographics arise from typical programming style
 - Few static pointers to large data structures
 - e.g., root of a tree data structure
 - When program finish with tree it is only accessible from root
 - When ptr to root is deleted, entire tree is garbage
 - Use these key objects as indicators for GC
 - When death of a cluster no longer predictable by their age, should be promoted out of time-based generation
 - One key object (root of tree) retained in generational scheme

Key objects before promotion



Key objects after promotion



Using a mature object space

- Promote very old objects out of time-based generation scheme into mature object space, all at once
 - Avoid disruptive collections
- Mature object space divided into areas, each with remembered set
 - Each collected one at a time in round-robin fashion
 - Area is collected when its remembered set is empty
- What if objects too big to fit in one area?
 - Use carriages and train analogy