GC Design Choices

What about other storage reclamation schemes?

Memory management options

- Manual /explicit memory management
  - Strengths?
  - Challenges?
- Automated memory management (garbage collection)
  - Strengths?
  - Challenges?
- Any others?

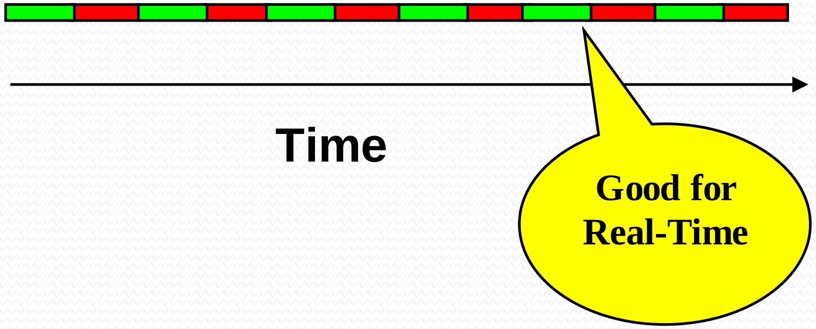
Real-time garbage collection (RTGC)

- Real-time system
  - A system that meets real-time requirements.
- Real-time requirements
  - As expected, operations must be logically correct
  - Additionally, operations must be completed within deadline
- RTGC
  - Bounded-time allocation
  - Predictable deallocation
  - Must be incremental

Real-time garbage collection (RTGC)

```
public void f(){
    startLaser();
    Obj o = new Obj();
    stopLaser();
}

public static void main(…){
    f();
}
```



Good for Real-Time

RTGC strengths and challenges

- Need extra storage
  - Store state of application when collector runs
- Application can allocate memory during garbage collection
- Space-time trade-off

RTSJ scoped-memory

- RTSJ – Real-time specification for Java proposed by the Real-time for Java expert group (RTJEG).
- Semi-manual with scopes
  - Scopes: regions of memory
  - Scopes: limited life times
  - Threads allocate from current scope
  - Predictable allocation
  - Predictable deallocation
  - No dangling pointers

RTSJ scoped-memory

```
ScopedMemory
 scope = new ScopedMemory(1024);
scope.enter(new Runnable() {
 public void run(){
   // do some stuff
   someObj o = new someObj();
   // do some more stuff
   someObj s = new someObj();
 }
});
// scope is collected (no threads)
```

RTSJ scoped-memory challenges

- Restrictive memory model
- Difficult to use
- Can leak memory

Memory management options

- Manual/explicit memory management
- Automated memory management (GC)
- Real-time garbage collection
- RTSJ scoped-memory

Garbage collection design choices

- Stop-the-world
- Incrementality
- Hybrid
- Concurrency
- Parallelism

Stop-the-world collectors

- Typically used on uniprocessor systems
- Suspend application
- Run collector from start to finish
- Resume application

Stop-the-world collectors

- Execution costs?
  - Pause time
  - Discovery of live objects (how long does it take?)
  - Instruction overhead (per instruction)
  - Delay between object death and collection
  - Number of collectible objects collected
  - Overall execution time
  - Worst-case vs average case performance
  - frequency

Incremental collection

- Interleave GC with application
- Note: for full heap tracing
  - Pause time increases with heap size
- Incremental tracing
  - Bounded tracing time
  - Conservative assumption
    - All other objects in heap are live
  - Remember pointers from objects in heap
    - Add such pointers to root set for tracing

Hybrid collection

- Generational collectors
  - Collect young objects frequently
    - Young objects die quickly
- Example
  - Copy collection for young objects
  - Non-copy collection for older objects
- Partitioning
  - Copy intra-partition incrementally
  - Reference count inter-partition

Concurrent collection

- Application is called a *mutator*
- GC regards application as such because it is mutating the heap
- *Mutator* and GC function at the same time except when GC needs info from *mutator*
  - Synchronization

Parallel collection

- Concurrency among multiple GC threads
  - Load balancing
  - Synchronization
  - Race condition when tracing