

Controlling Fragmentation and Space Consumption in the Metronome

David F. Bacon

Perry Cheng

V.T. Rajan

IBM T.J. Watson Research Center

Problem Domain

- Hard real-time garbage collection
 - Implemented for Java
- Uniprocessor
 - Multiprocessors rare in real-time systems
 - Complication: collector must be finely interleaved
 - Simplification: memory model easier to program
 - No truly concurrent operations
 - Sequentially consistent

Metronome Project Goals

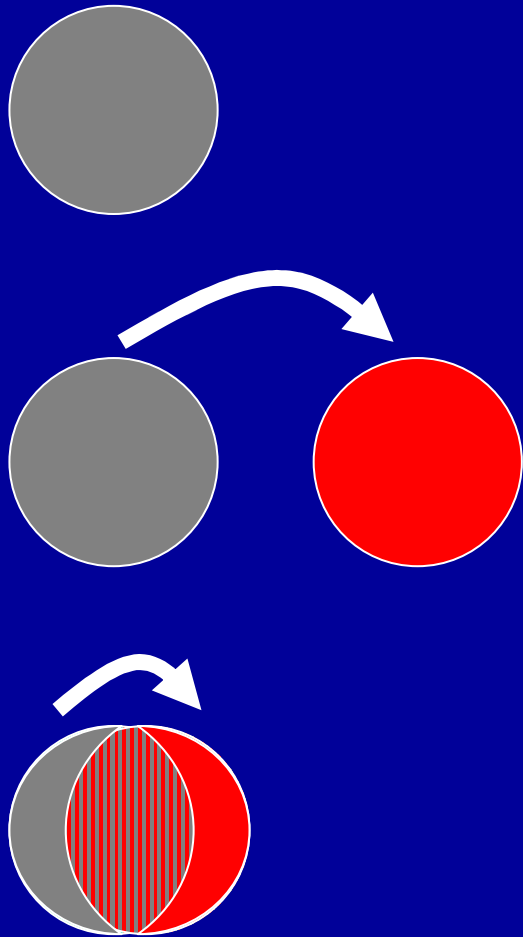
- Make GC feasible for hard real-time systems
- Provide simple application interface
- Develop technology that is efficient:
 - Throughput, Space comparable to stop-the-world
- **BREAK THE MILLISECOND BARRIER**
 - While providing even CPU utilization

Outline

- Overview of the Metronome
- Empirical Results
- What is Fragmentation?
 - Static and dynamic measures
- How do we control space consumption?
- Conclusions

The Metronome Collector

Real-time GC Approaches



- **Mark-sweep (non-copying)**
 - Fragmentation avoidance, coalescing
 - Subject to space explosion
- **Semi-space Copying**
 - Concurrently copies between spaces
 - Cost of copying, consistency, 2x space
- ***The Metronome***
 - Mark-sweep, selective defragmentation
 - Best of both, adaptively adjusts

Components of the Metronome

- Incremental mark-sweep collector
 - Mark phase fixes stale pointers
- Selective incremental defragmentation
 - Moves < 2% of traced objects
- Time-based scheduling
- Segregated free list allocator
 - Geometric size progression limits internal fragmentation

Old

New

Support Technologies

- Read barrier: to-space invariant [Brooks]
 - New techniques with only 4% overhead
- Write barrier: snapshot-at-the-beginning [Yuasa]
- Arraylets: bound fragmentation, large object ops

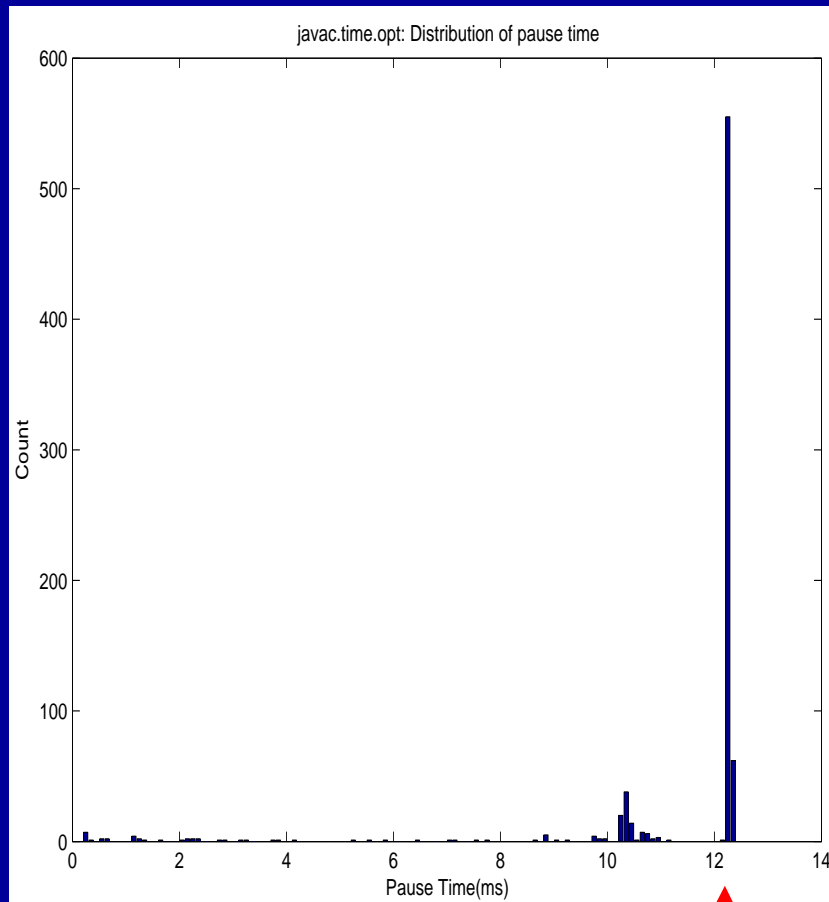
Old

New

Empirical Results

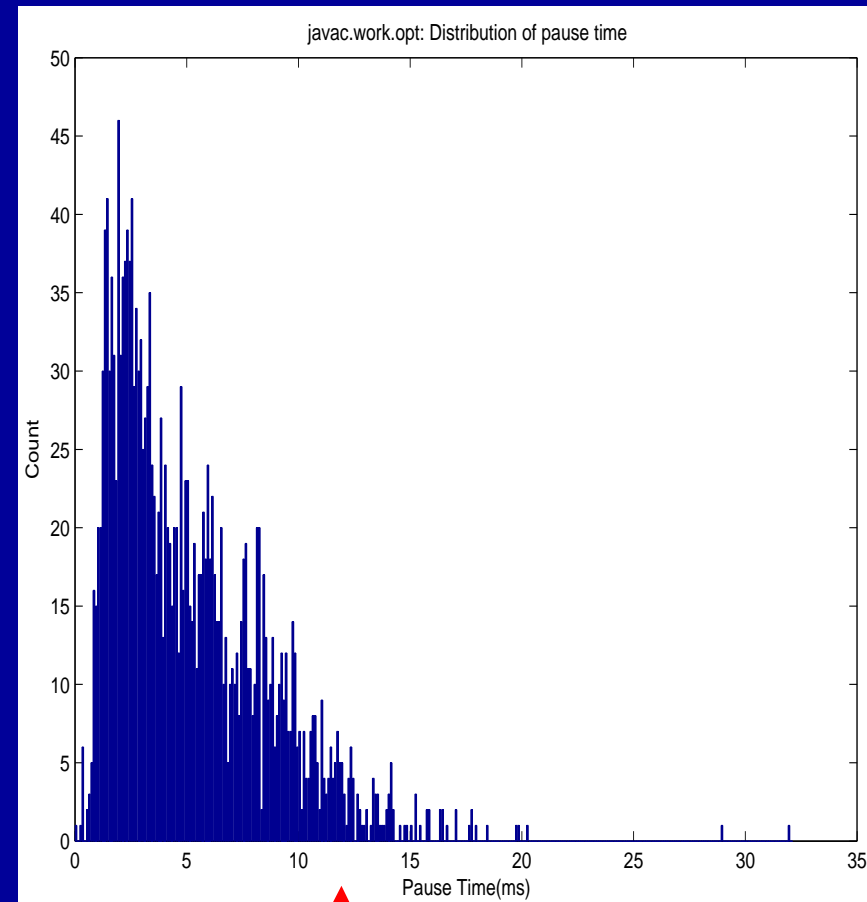
Pause time distribution: javac

Time-based Scheduling



12 ms

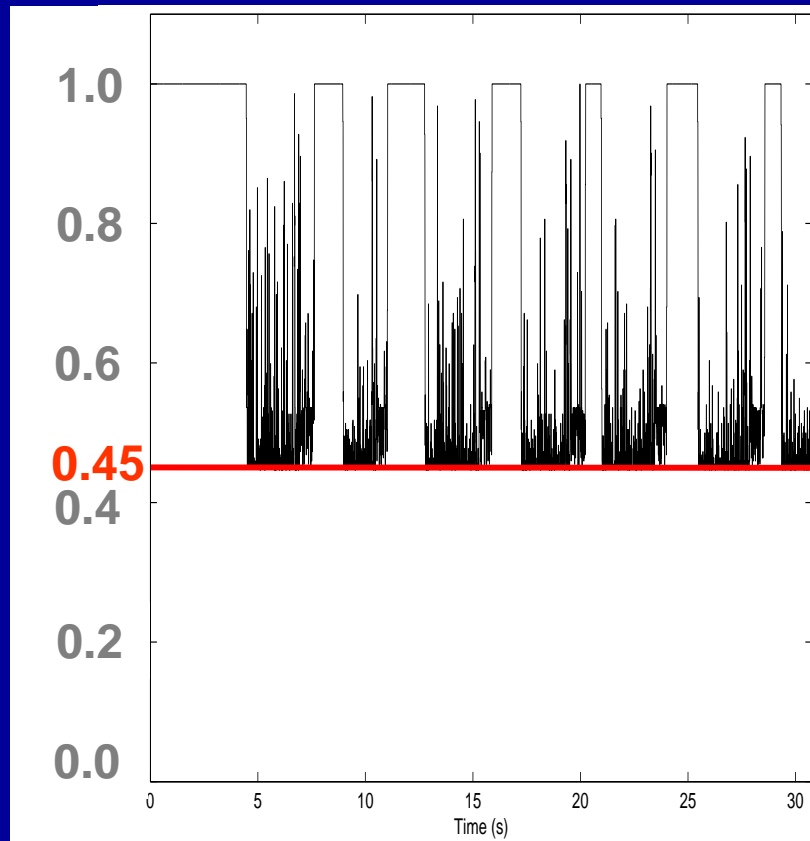
Work-based Scheduling



12 ms

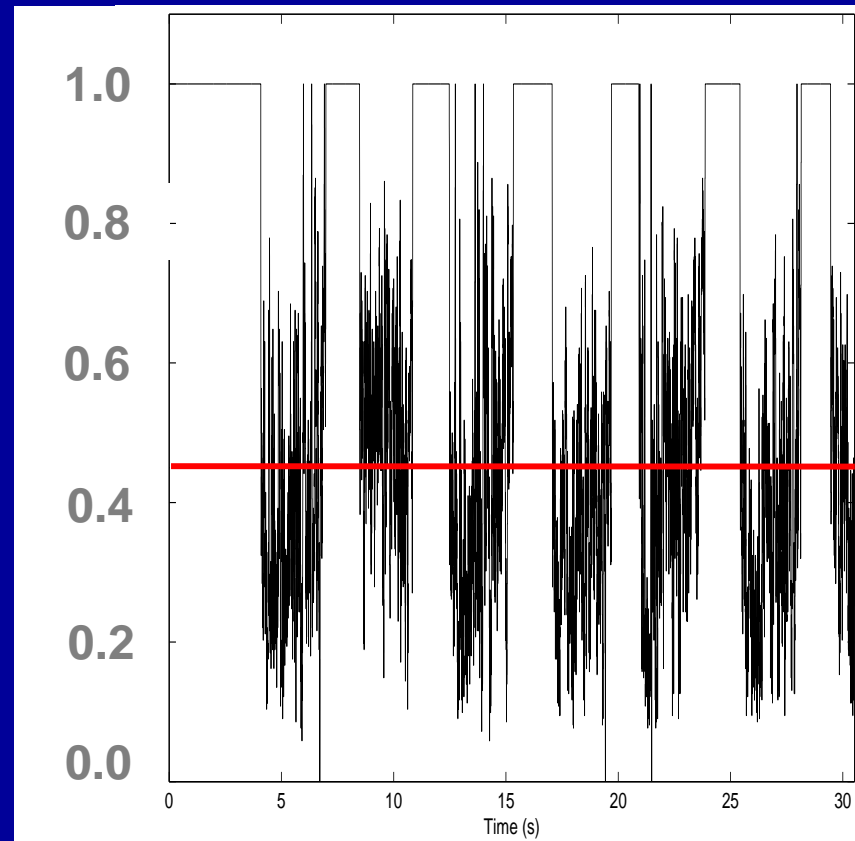
Utilization vs. Time: javac

Time-based Scheduling



Time (s)

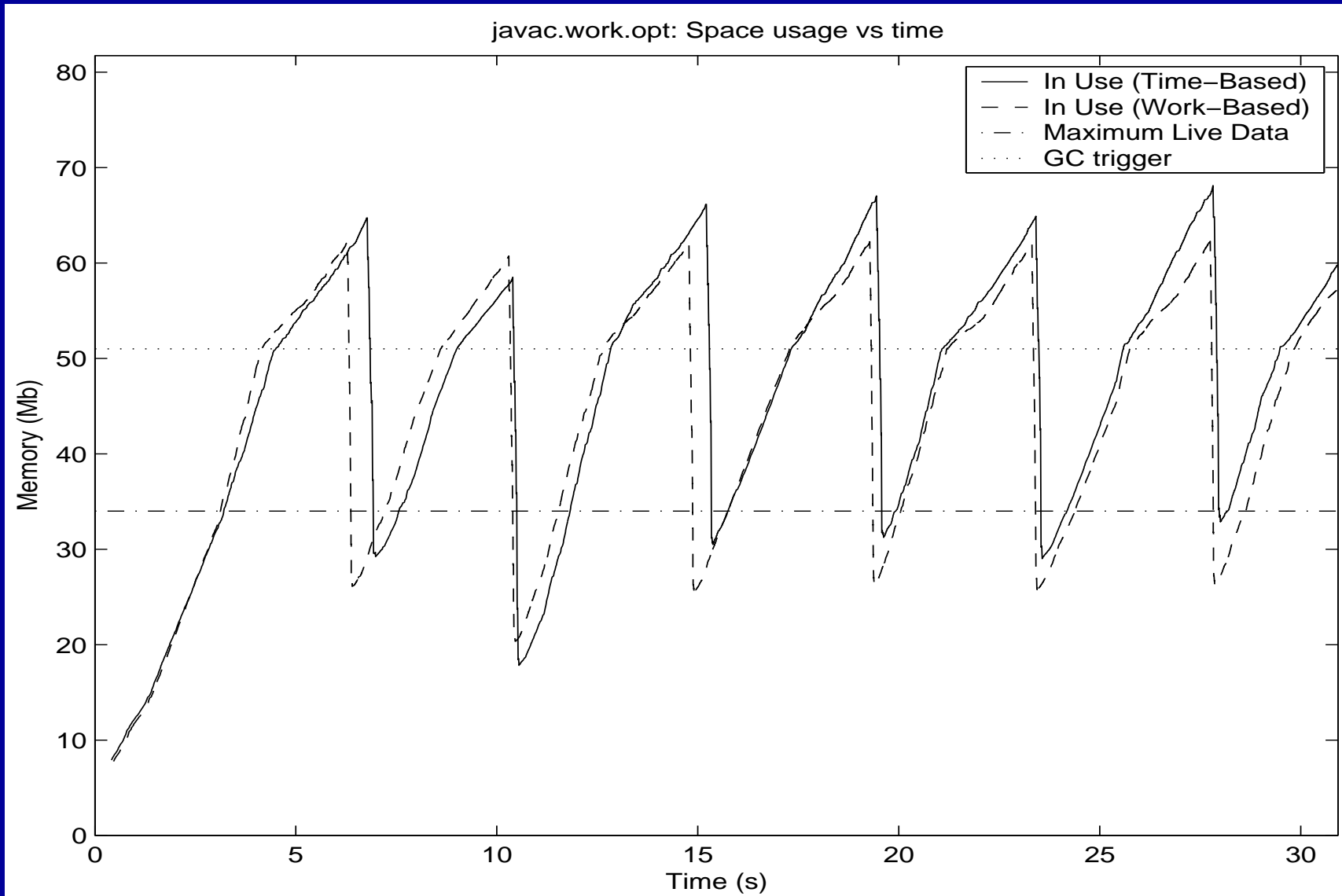
Work-based Scheduling



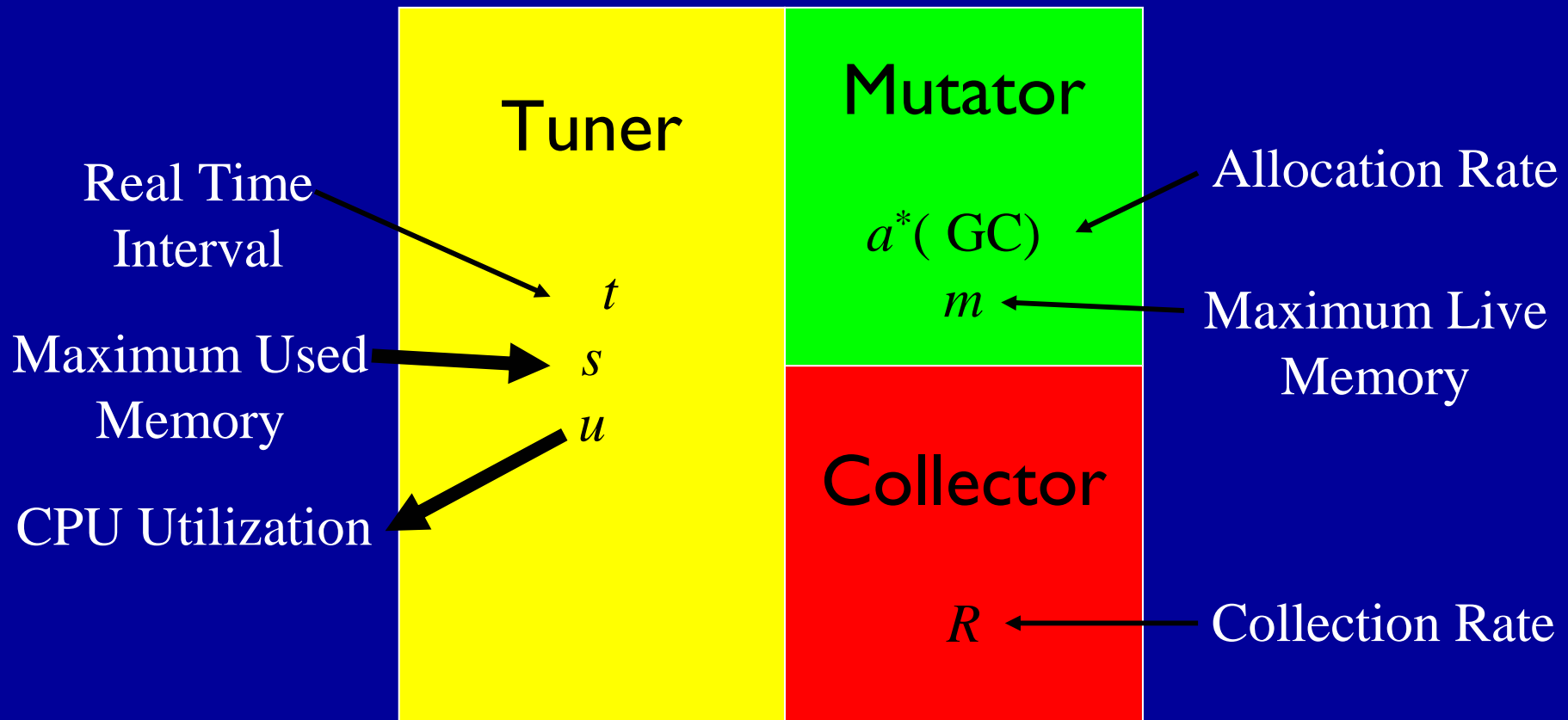
Time (s)

Utilization (%)

Space Usage: javac



Parameterization



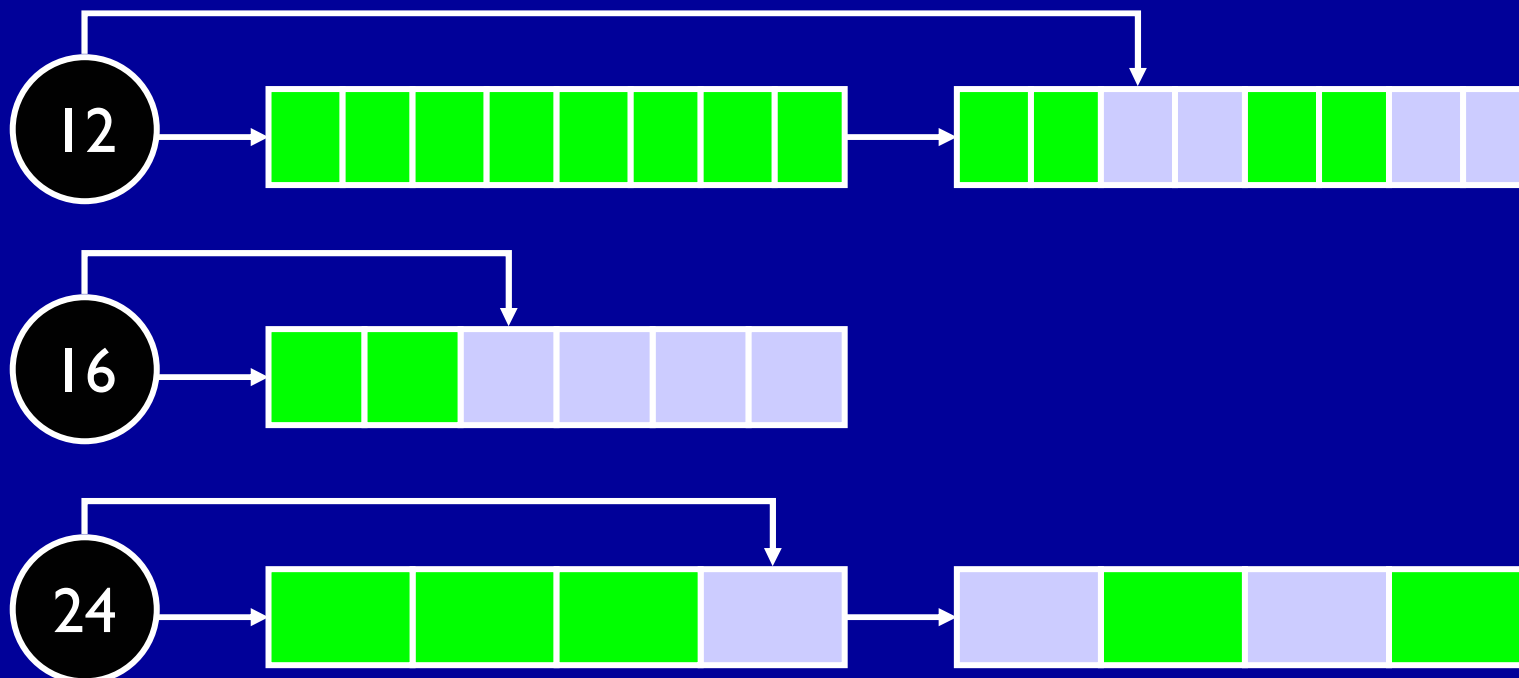
Is it real-time? Yes

- Maximum pause time < 4 ms [currently]
- MMU $> 50\% \pm 2\%$
- Memory requirement $< 2 \times$ max live

Static Fragmentation

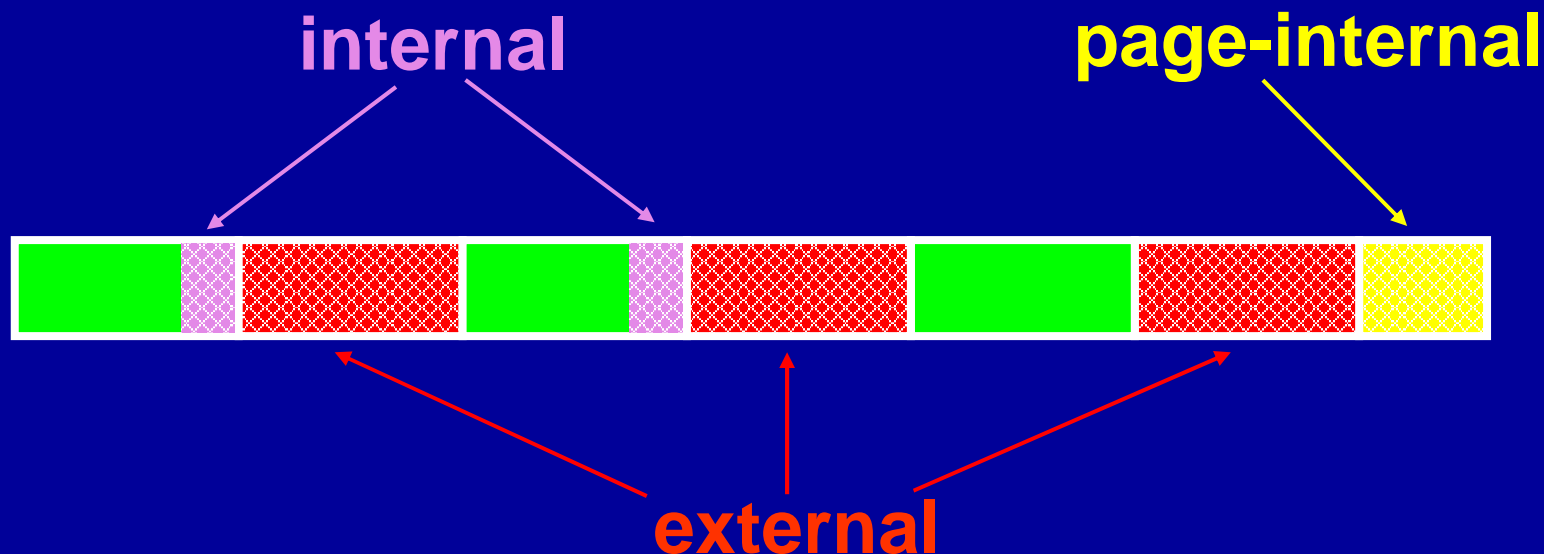
Segregated Free List Allocator

- Heap divided into fixed-size pages
- Each page divided into fixed-size blocks
- Objects allocated in smallest block that fits

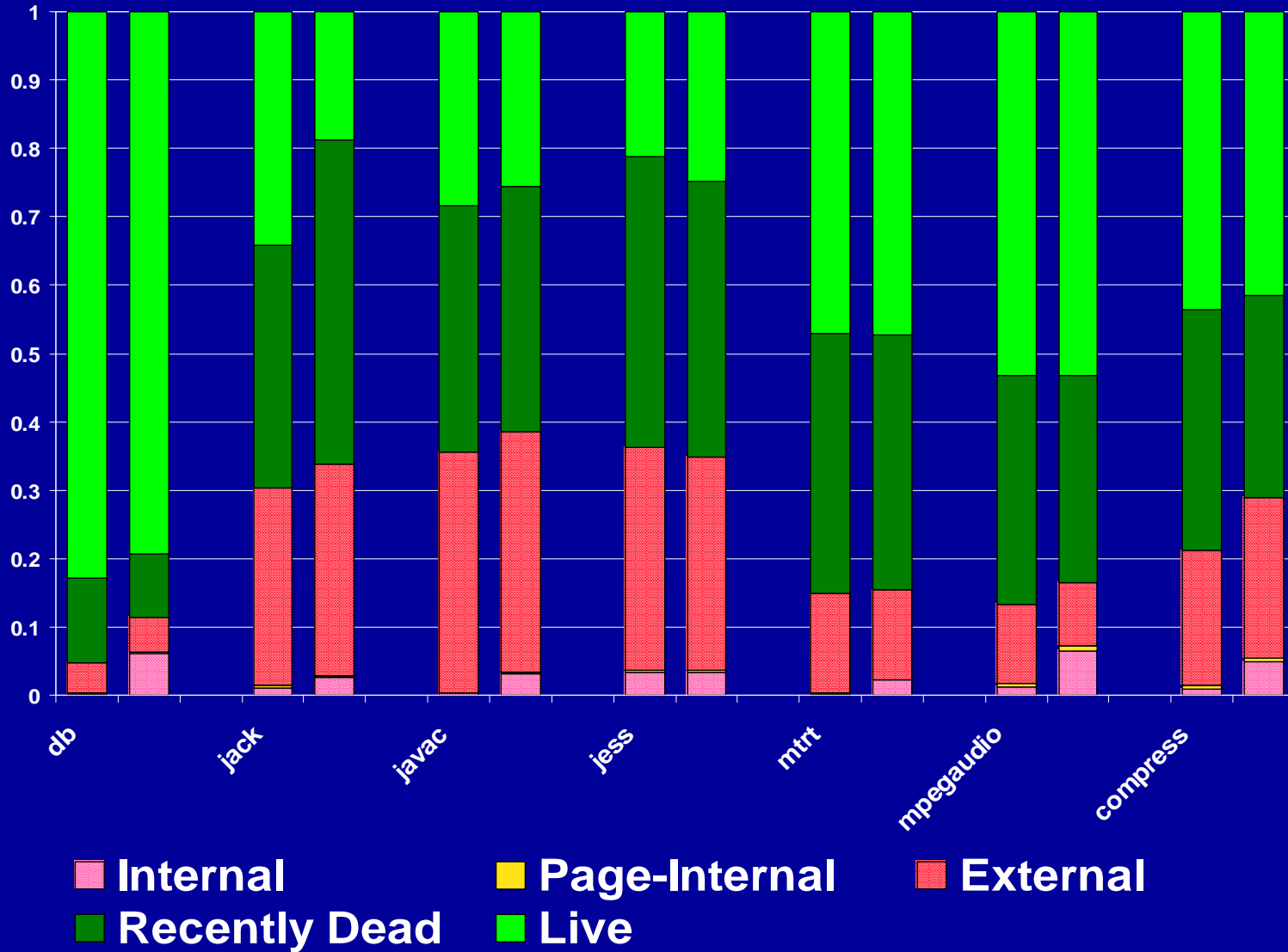


Fragmentation on a Page

- Internal: wasted space at end of object
- Page-internal: wasted space at end of page
- External: blocks needed for other size



Fragmentation: $\rho=1/8$ vs. $\rho=1/2$



Dynamic Fragmentation

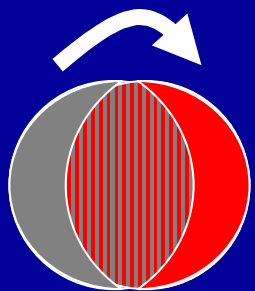
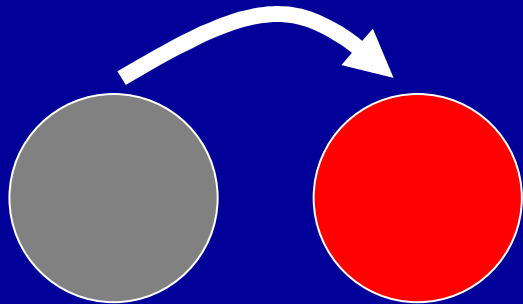
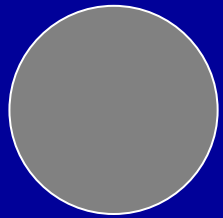
Locality of Size: λ



- Measures reuse
- Normalized: $0 \leq \lambda \leq 1$
- Segregated by size

$$\lambda = \sum_i \min (f_i / f, a_i / a)$$

λ in Real-time GC Context



- Mark-sweep (non-copying)

Assumes $\lambda=1$

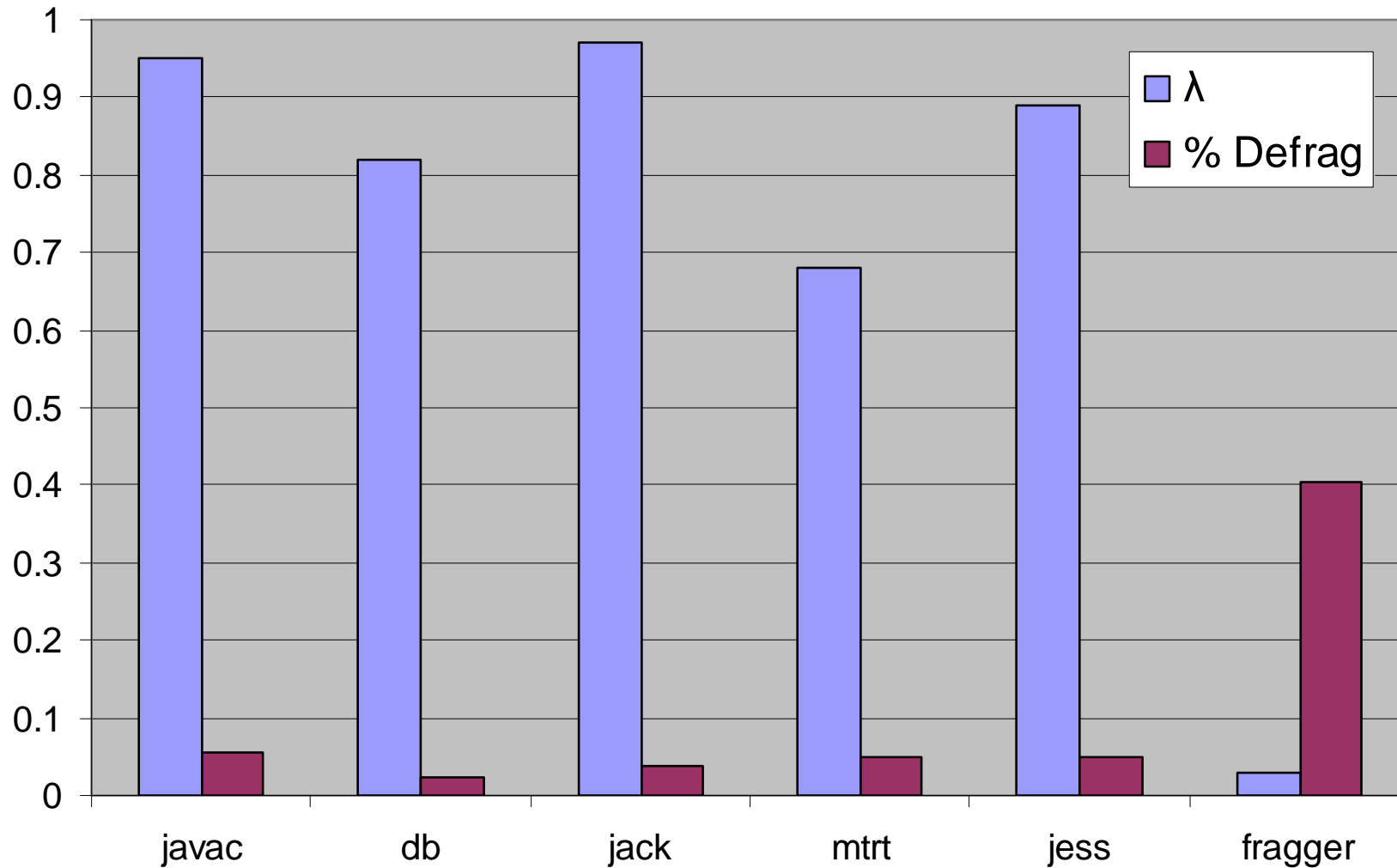
- Semi-space Copying

Assumes $\lambda=0$

- *The Metronome*

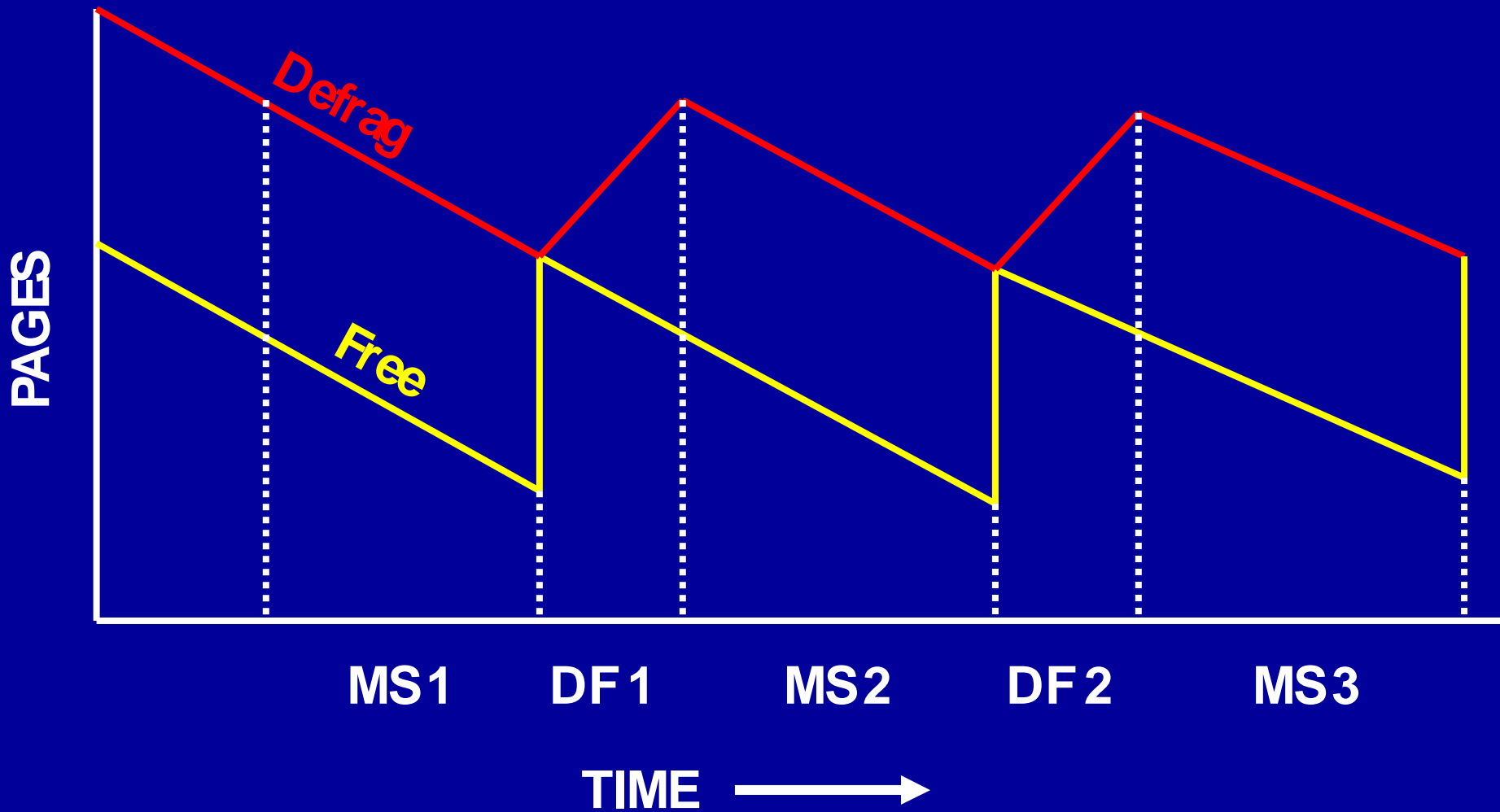
Adapts as λ varies

λ in Practice

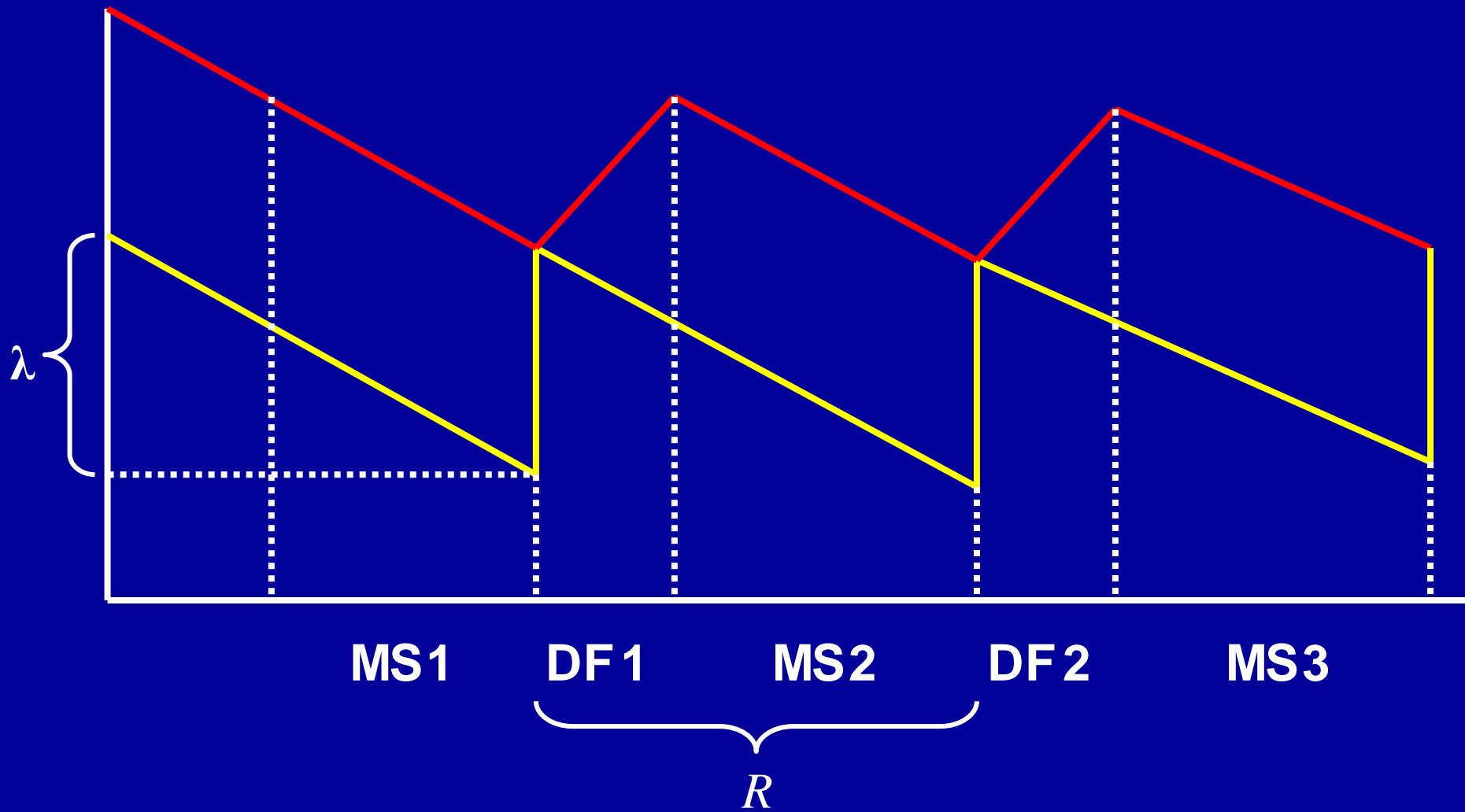


Controlling Space Consumption

Triggering Collection



Factors in Space Consumption



Reducing Space Consumption

- Collection Rate R
 - Higher rate: less memory reserve needed
 - Speed up collector
 - Specify rate more precisely (bound worst-case)
- Locality of Size λ
 - Higher locality: less free page reserve needed
 - Specify page requirement more precisely

Conclusions

Conclusions

- Contributions
 - Time-based scheduling (Tunable)
 - Mostly non-copying collection (λ varies)
 - Efficient software read barrier
 - Precise definition of fragmentation
 - Precise specification of collection triggers
- The Metronome provides true real-time GC
 - First collector to do so without major sacrifice
 - Short pauses (4 ms)
 - High MMU during collection (50%)
 - Low memory consumption (2x max live)