

# Garbage-First: Low Latency, High Throughput Garbage Collection

Garbage First!

David Detlefs

Java™ Technology Research Group

Sun Microsystems Laboratories (Burlington, MA)

*(Steve Heller, Christine Flood, Tony Printezis)*

# Outline

- ◆ Goals.
- ◆ Mechanisms.
- ◆ Results.
- ◆ Related work.
- ◆ Future work.

# Garbage-First Goals

## Garbage First!

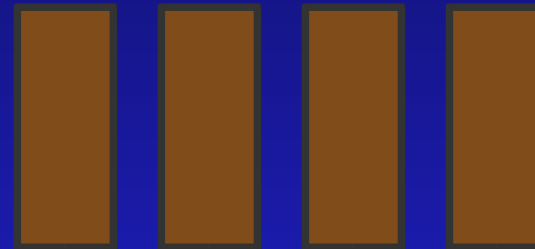
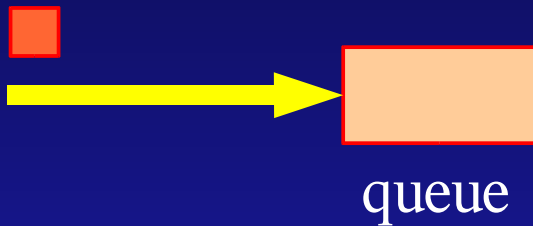
- ◆ Scale to large heaps, large #'s of threads.
- ◆ High-probability compliance with a *real-time goal* (aka *MMU* specification):
  - ◆ “GC should consume no more than  $x$  ms in any  $y$  ms interval...”
- ◆ High throughput
  - ◆ On machines with large numbers of processors
    - ◆ Implies effective parallelism when the mutator is stopped, and...
    - ◆ Use of concurrent collection techniques when the mutator is running.

# Why (some) customers care about MMU

- ◆ Typical application:

Garbage First!

Incoming task



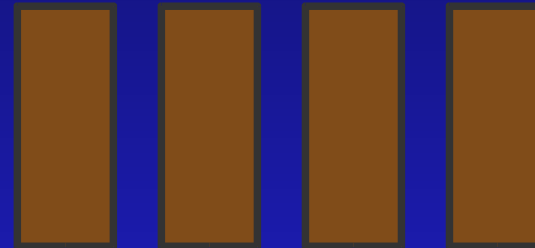
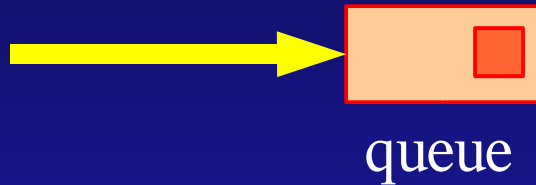
Application threads

# Why (some) customers care about MMU

- ◆ Typical application:

Garbage First!

Incoming task



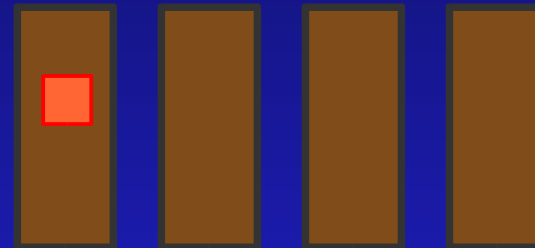
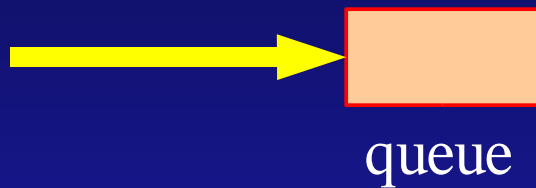
Application threads

# Why (some) customers care about MMU

- ◆ Typical application:

Garbage First!

Incoming task



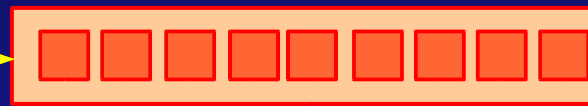
Application threads

# Why (some) customers care about MMU

- ◆ If GC stops mutator thread, queue grows:
  - ◆ Max latency to last service last task (+ processing rate) determines real-time goal (max ms gc/ms elapsed).

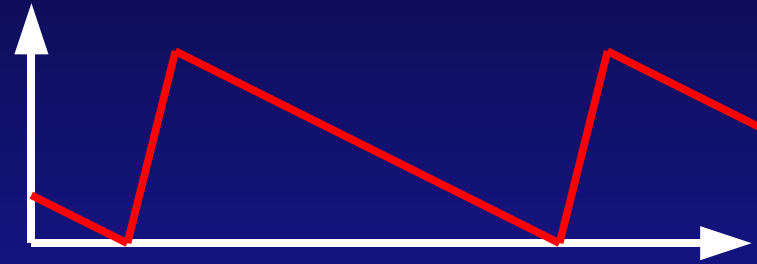
Garbage First!

Incoming task

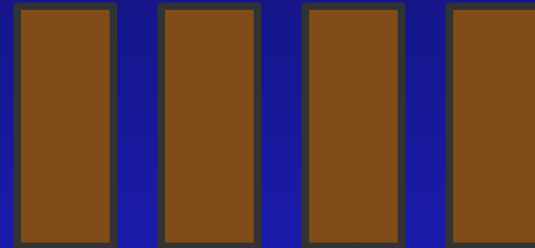


queue

Q len



Time



Application threads

# Garbage-First Mechanisms

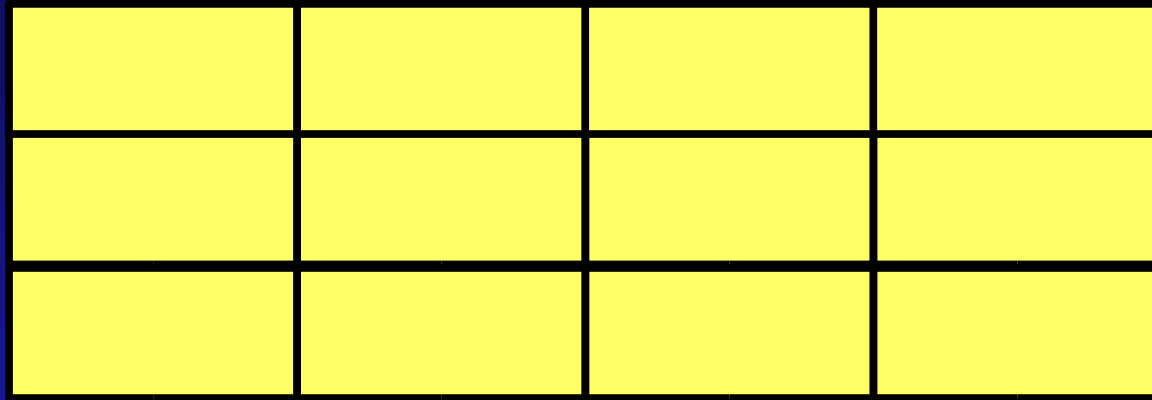
## Garbage First!

- ◆ Heap regions and rem sets.
- ◆ Evacuation pauses.
- ◆ Generational Garbage-First.
- ◆ Concurrent rem set maintenance.
- ◆ Concurrent marking.
- ◆ Collection set choice:
  - ◆ Garbage-First heuristic: choose best regions.
  - ◆ Pause-time modeling to meet real-time goal.

# Garbage-First Heap Layout

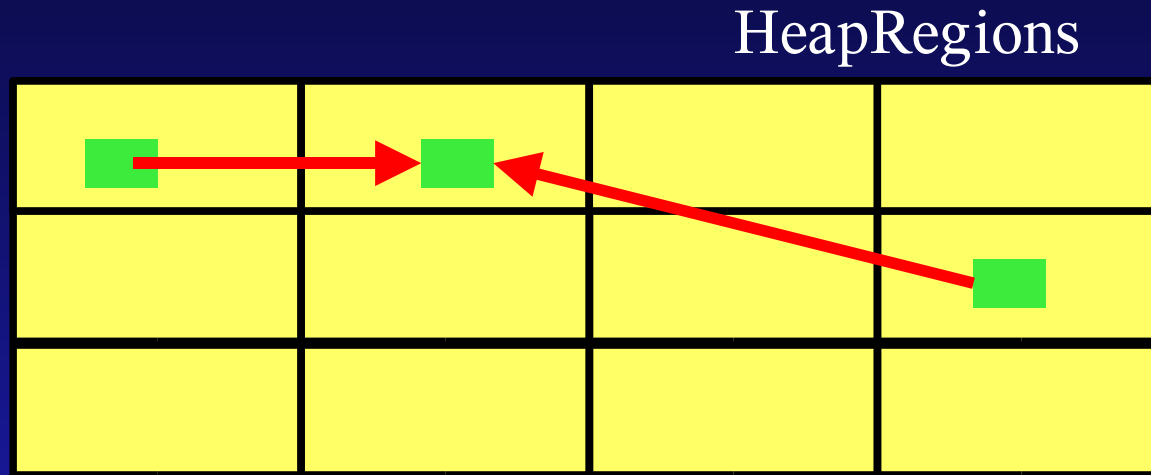
Garbage First!

HeapRegions



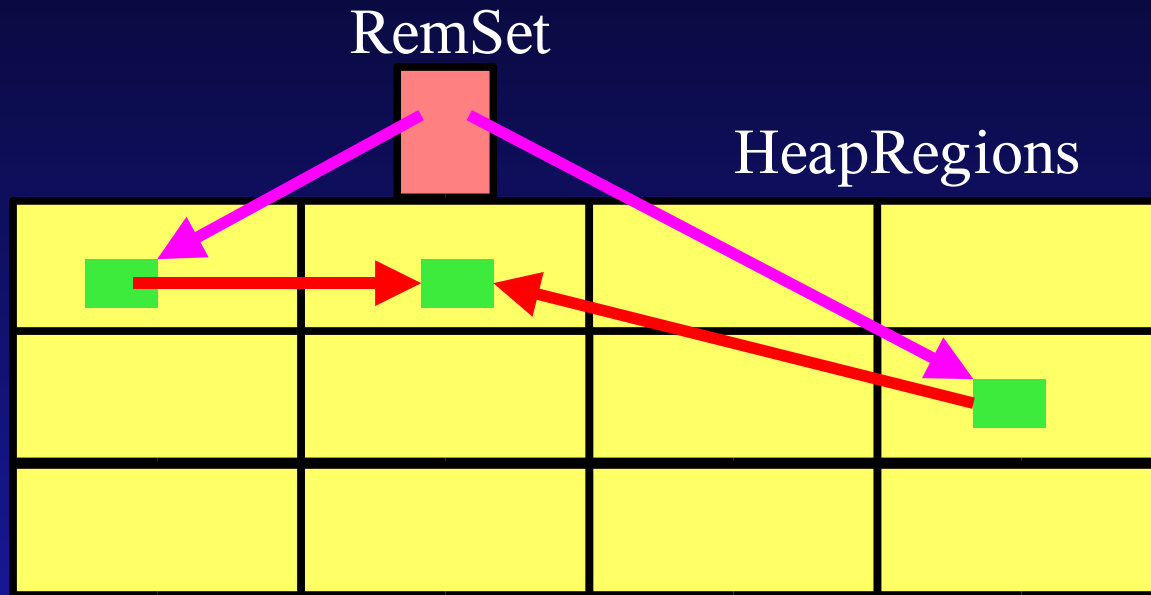
# Garbage-First Heap Layout

Garbage First!



# Garbage-First Heap Layout

Garbage First!



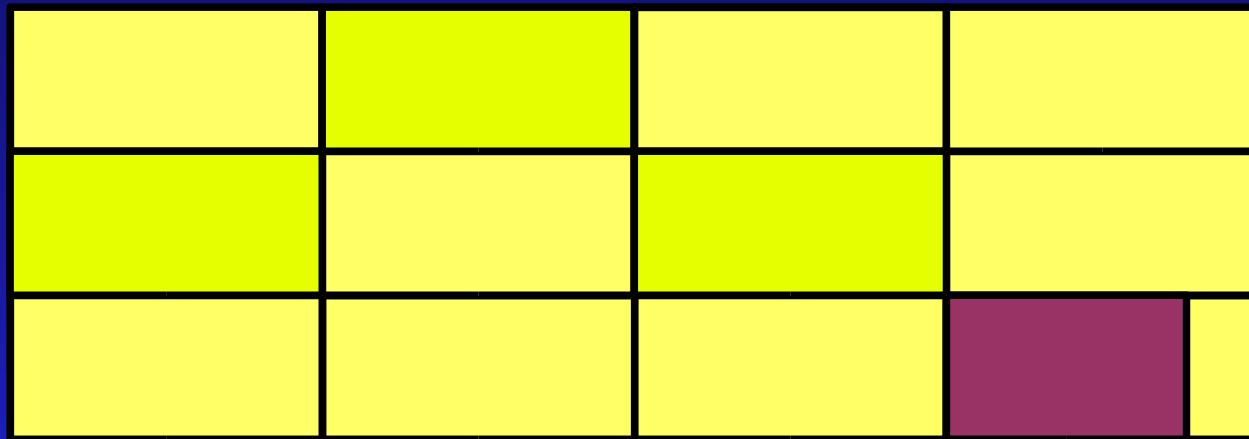
# Evacuation Pause

- ◆ Pick a *Collection Set* of regions.
- ◆ Evacuate live data elsewhere:
  - ◆ Live as indicated by roots, rem sets, previous marking information.



# Evacuation Pause

- ◆ Pick a *Collection Set* of regions.
- ◆ Evacuate live data elsewhere:
  - ◆ Live as indicated by roots, rem sets, previous marking information.



Garbage First!

# Parallelism in Evac Pauses

- ◆ Based on [Flood+01]:
  - ◆ Static partitioning + fine-grained work-stealing for load-balancing.
  - ◆ (See also [Endo97].)
- ◆ Different phases combined to avoid barrier synchronizations (Flood).

# Remembered Set Maintenance

- ◆ RemSet = hash tables of *cards*.
  - ◆ Would be *very* expensive to update at each mutator pointer write!
- ◆ So the mutator logs updates, a *concurrent remembered set* thread reads the logs and updates the remembered sets.
- ◆ Mutator barrier filters for intra-region, NULL writes, writes to already-dirty cards.
- ◆ We *must* do RS updating concurrently, or pause times would be too long.

# Generational Garbage-First

- ♦ Mutator allocation region can be declared *young*:
  - ♦ Will be included in the next collection set.
  - ♦ (Therefore) don't need to track pointer modifications.
  - ♦ But young regions are not physically segregated.
- ♦ Choose number of young regions to fit pause time (via dynamic feedback).
  - ♦ Might choose *not* to make a region young.

# Concurrent Marking

Garbage First!

- ◆ Eventually surviving objects fill up the heap (the “old gen”).
- ◆ Mark live objects concurrently.
  - ◆ We use *snapshot-at-the-beginning* [Yuasa90] marking (vs *incremental update*).
  - ◆ Some extra barrier overhead, but much shorter marking-related pause times, less total work.
    - ◆ Related work: static analysis to eliminate barriers (Detlefs&Nandivada)
- ◆ Marking also records live data in regions.

# Marking, Evacuation, and Allocation

## Garbage First!

- ◆ Objects are efficiently “allocated black” by recording allocation points in all heap regions at start of marking.
- ◆ Evacuation can occur during (and independently of) a marking cycle:
  - ◆ Evacuation uses results of last completed marking cycle.
  - ◆ Evacuation preserves partial results of in-progress marking.

# Using Marking Data

- ♦ Marking completes. Regions are labeled with live data ( $\Rightarrow$  *known garbage*).
- ♦ Reclaim any completely-free regions (*cleanup*).

50%	3%	0%	0%
10%	15%	5%	25%
60%	80%	100%	100%

# Using Marking Data

- ◆ Now continue allocation....

Garbage First!

50%	3%		
10%	15%	5%	25%
60%	80%	100%	100%

# Using Marking Data

- ◆ Do collections as often as real-time spec allows.
- ◆ Pick a collection set:
  - ◆ Young + most-efficient non-young.

50%	3%	100%	
10%	15%	5%	25%
60%	80%	100%	100%

Garbage First!

# Using Marking Data

- ◆ (Hence the name: *Garbage-First!*)

Garbage First!

50%	3%	100%	
10%	15%	5%	25%
60%	80%	100%	100%

# Using Marking Data

- ◆ (Hence the name: *Garbage-First!*)

Garbage First!

50%	3%	100%	
10%	15%	5%	25%
60%	80%	100%	100%



The diagram shows a 3x4 grid of cells. The top row has cells with 50%, 3%, 100%, and an empty cell. The middle row has cells with 10%, 15%, 5%, and 25%. The bottom row has cells with 60%, 80%, 100%, and 100%. Red arrows indicate flow: one from the 3% cell to the empty cell, one from the 10% cell to the empty cell, and one from the 5% cell to the empty cell.

# Using Marking Data

- ◆ (Hence the name: *Garbage-First!*)

Garbage First!

50%			100%	
	15%		25%	
60%	80%	100%	100%	

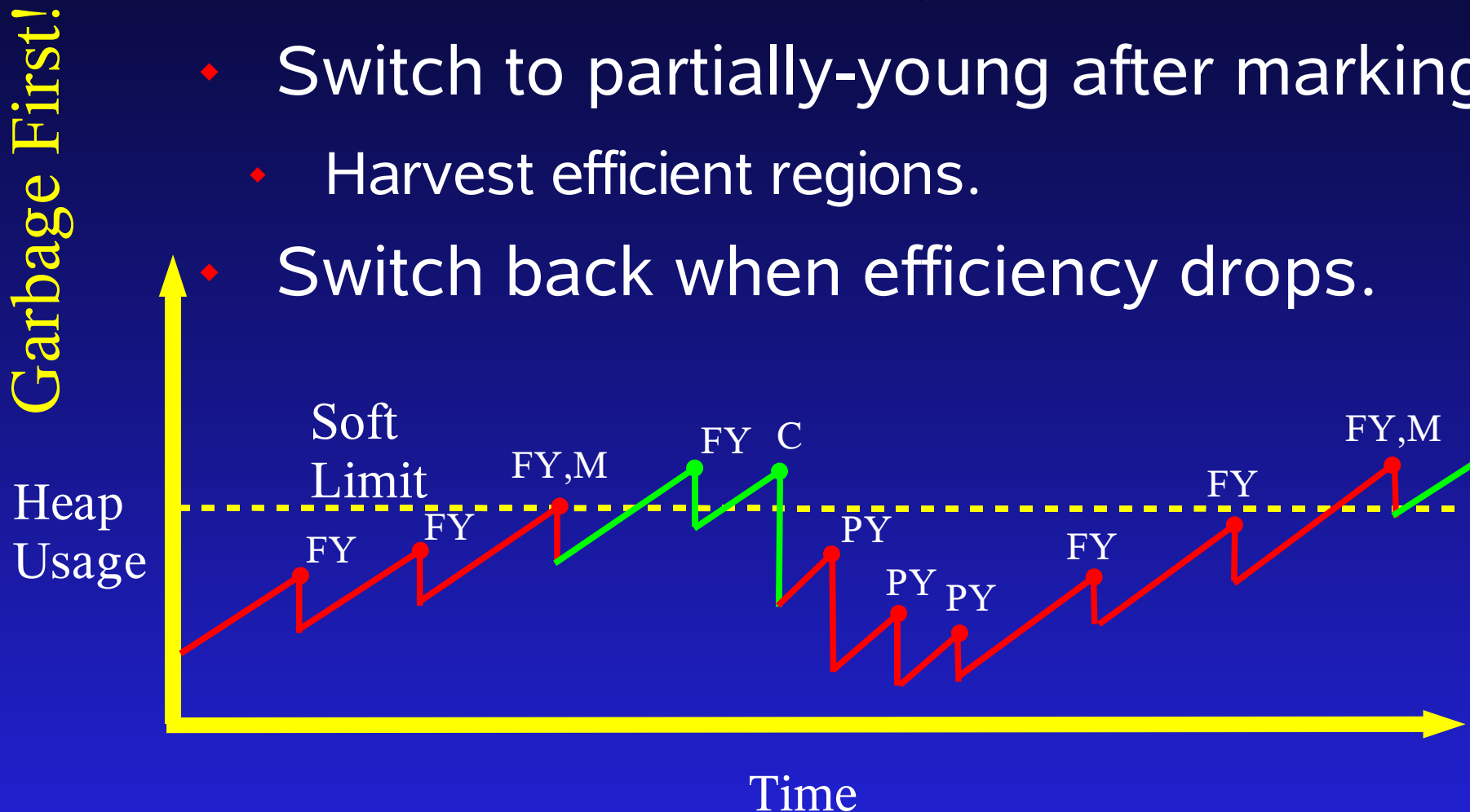
# Pause Time Compliance (Printezis)

Garbage First!

- ◆ How many non-young regions can we collect in  $x$  ms?
- ◆ Model predicts the cost of adding a heap region  $r$  to a collection set:
  - ◆  $C(r) = a*r.live\_data + b*r.rem\_set\_size + c$
  - ◆  $a$ ,  $b$ , and  $c$  are estimated initially, tracked dynamically.
- ◆ Add regions in efficiency (*garbage/cost*) order until you 'overflow the bucket'.

# A Garbage-First Execution

- ◆ Usually in fully-young mode.
- ◆ Switch to partially-young after marking
  - ◆ Harvest efficient regions.
- ◆ Switch back when efficiency drops.



# Benchmarks

- ◆ “Telco”
  - ◆ A call setup application (SIP server).
  - ◆ Trying to get permission to tell you their name.
- ◆ SPECjbb
  - ◆ Java version of a TPC benchmark.
- ◆ Compare with ParNew+Concurrent  
MarkSweep
  - ◆ best Sun product solution for low pause  
time/high throughput.

# MMU Specification Compliance

Garbage First!

- ◆ 3 Metrics
  - ◆ % of *timeslices* exceeding constraint (via sampling).
  - ◆ Avg % of excess.
  - ◆ Worst excess (as % of specified mutator time).

Benchmark	MMU	Garbage-First			CMS		
		V%	AvgV%	wV%	V%	AvgV%	wV%
SPECjbb	150/450	3.1%	2.3%	6.5%	13.4%	67.7%	100%
telco	75/225	0.2%	3.3%	3.7%	0.5%	32.8%	100%

# Maximum Marking Pause

- ♦ SATB is very good at limiting marking-related pauses.

App	Garbage-First	CMS
SPECjbb	24 ms	935 ms
Telco	49 ms	382 ms

Garbage First!

# Throughput

- ◆ ~ Parity on telco.
- ◆ A little behind on SPECjbb
  - ◆ Working on it!

Garbage First!

App	Garbage-First	CMS	units
SPECjbb	27-30	32	Kops/sec
telco	1160-1170	1190	Calls/sec

# Related Work

- ◆ Independent collection of heap regions:
  - ◆ *[Bishop77], Train algorithm [Hudson&Moss92], MC<sup>2</sup> [Sachindren&Moss03]*
- ◆ Concurrent Marking:
  - ◆ *[Dijkstra+78], [Yuasa90], [Boehm+91], [Printezis&Detlefs00]*
- ◆ Marking + Evacuation:
  - ◆ *[Lang&Dupont87], [Ben-Yitzhak+02]*

## Related Work

- ◆ Oldest-First
  - ◆ *[Stefanovic+02], [Hansen&Clinger02]*
- ◆ Real-time GC
  - ◆ *[Baker78], Metronome [Bacon+03], [Henriksson98]*

# Things I didn't have time to mention :-)

- ◆ Popular object handling
- ◆ “Pure” Garbage-First.
- ◆ Age segregation...

Garbage First!

# Future Work

- ◆ Fix remaining problems:
  - ◆ Rem set space overhead (Flood/Detlefs).
  - ◆ Make marking parallel as well as concurrent (Printezis).
  - ◆ Taking concurrent process overhead into account in MMU compliance (Printezis).
- ◆ Dynamic Pretenuring (Fabio Rojas, NU):
  - ◆ We claim it is well suited for Garbage-First:
    - ◆ Shorter, more predictable young-gen pauses.
    - ◆ Collect more via marking.

# Questions?

david.detlefs@sun.com  
tony.printezis@sun.com

# Snapshot-at-the-Beginning Concurrent Marking

Garbage First!

- ♦ Goal: mark all objects reachable at the beginning of marking cycle.
- ♦ Barrier for  $o.f = x$   
logs *pre-val*  $o.f$

