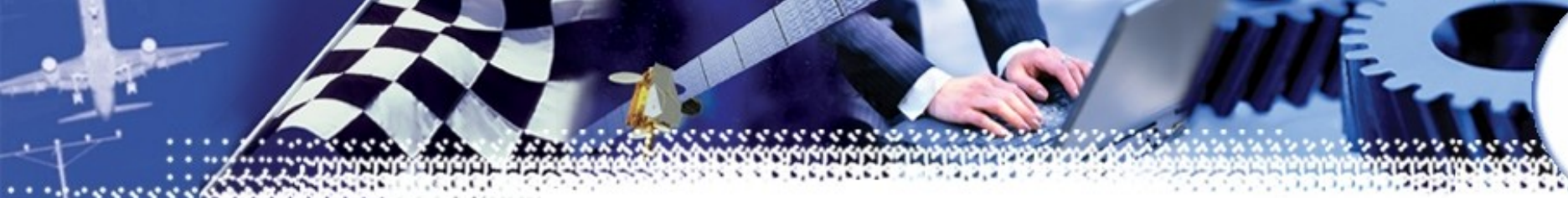


Limits of Parallel Marking Garbage Collection

...how parallel can a GC become?

Dr. Fridtjof Siebert
CTO, aicas
ISMM 2008, Tucson, 7. June 2008

Introduction

Parallel Hardware is becoming the norm

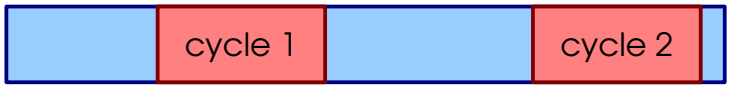
- even for embedded computers
- even for real-time systems

We need parallel garbage collection

- That is not only optimized for max. throughput
- But that gives **guarantees** on its performance
- The worst-case GC **timing** must **predictable** and **fast**

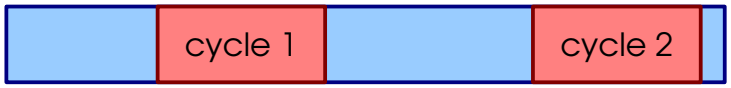
Terminology

blocking GC



Terminology

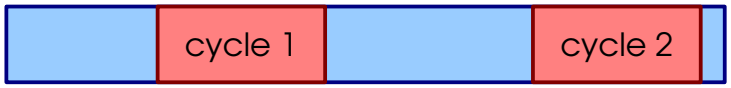
blocking GC



Incremental GC

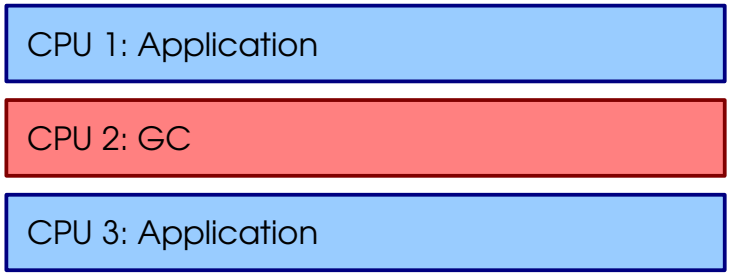
Terminology

blocking GC



Incremental GC

Concurrent GC

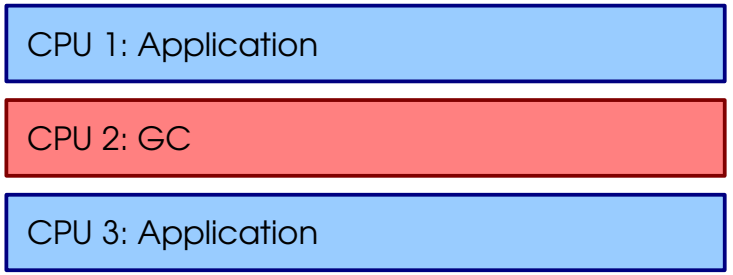


Terminology

blocking GC

Incremental GC

Concurrent GC



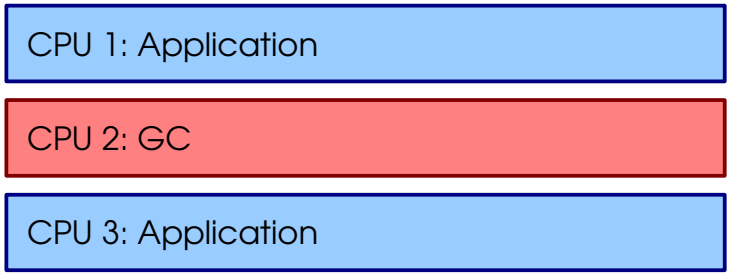
parallel GC

Terminology

blocking GC

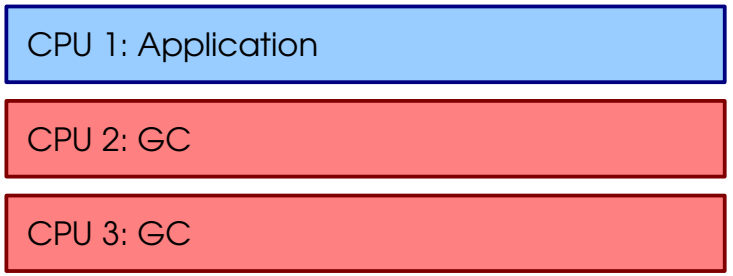
Incremental GC

Concurrent GC



parallel GC

Parallel & Concurrent

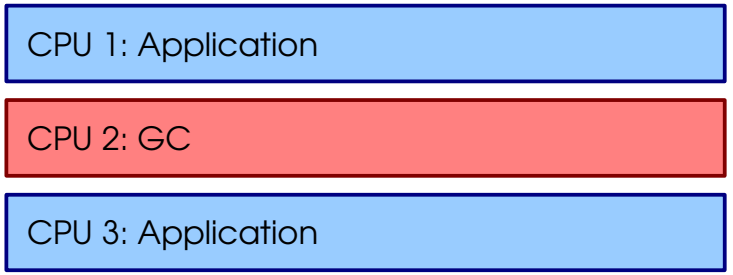


Terminology

blocking GC

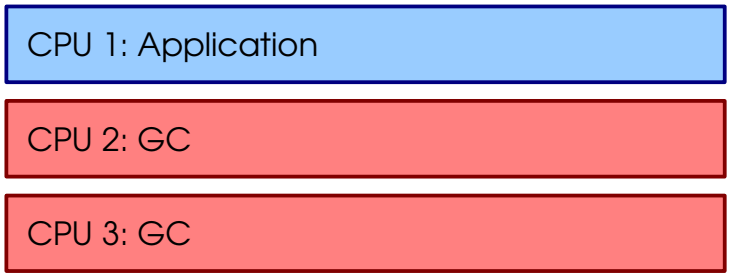
Incremental GC

Concurrent GC

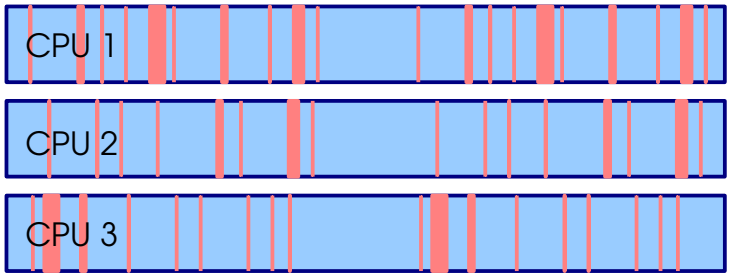


parallel GC

Parallel & Concurrent



Parallel & Concurrent



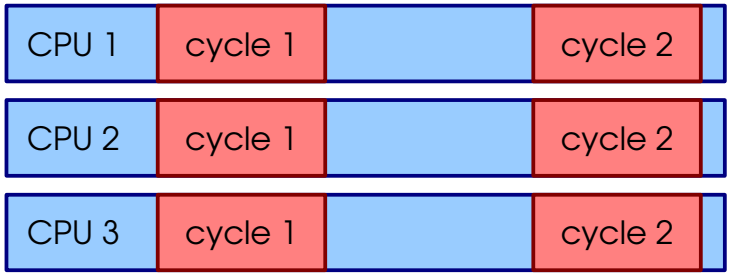
Terminology

blocking GC

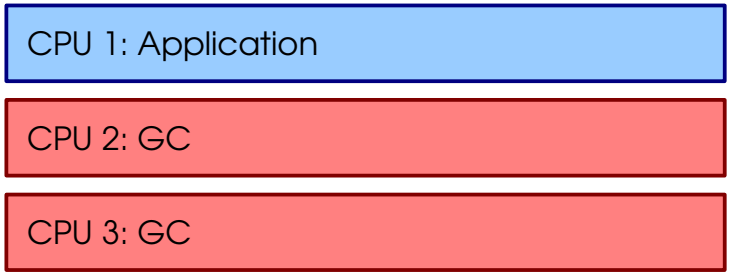
Incremental GC

Concurrent GC

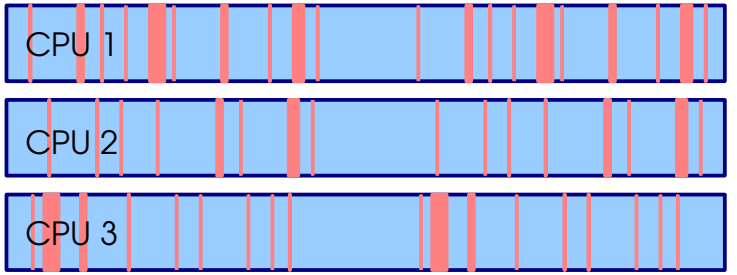
parallel GC



Parallel & Concurrent



Parallel & Concurrent



Parallel Mark & Sweep

Incremental Mark & Sweep

- uses three color marking: *white*, *grey* and *black*
- mark phase step is
  - find take grey object o
  - mark all white objects referenced by o grey
  - mark o black
- sweep phase step is
  - take white object
  - free its memory

Parallel Mark & Sweep

Parallel Sweep Steps

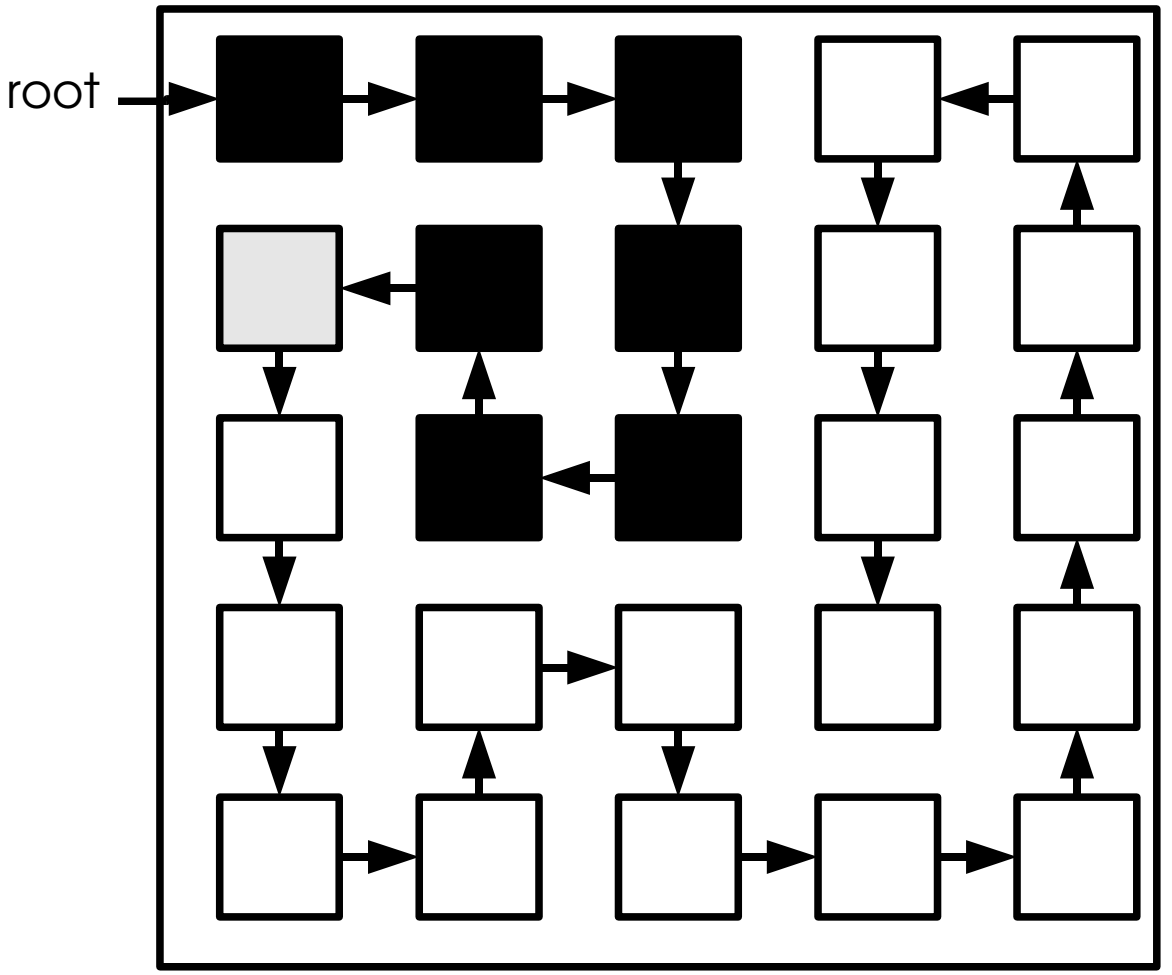
- not addressed here
- sweeping can be performed fully in parallel by
  - sweeping different regions of the heap by different CPUs
  - need parallel access to the free lists

Parallel Mark & Sweep

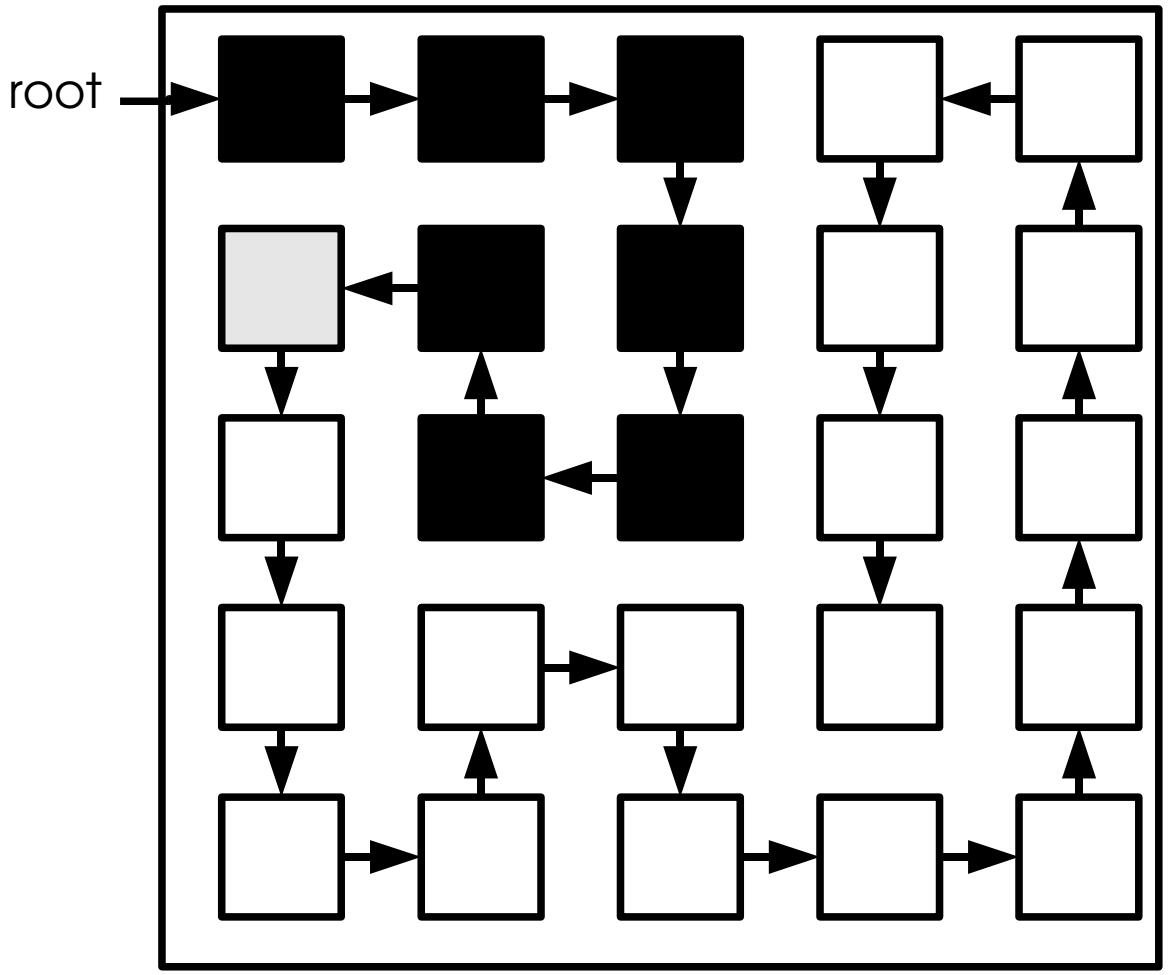
Parallel Mark

- several threads may scan grey objects in parallel
- new color *anthracite* for grey object that is being scanned by one CPU
- stalls possible if grey set temporarily empty!

Worst Case: Linked List



Worst Case: Linked List



CPU1

CPU2

CPU3

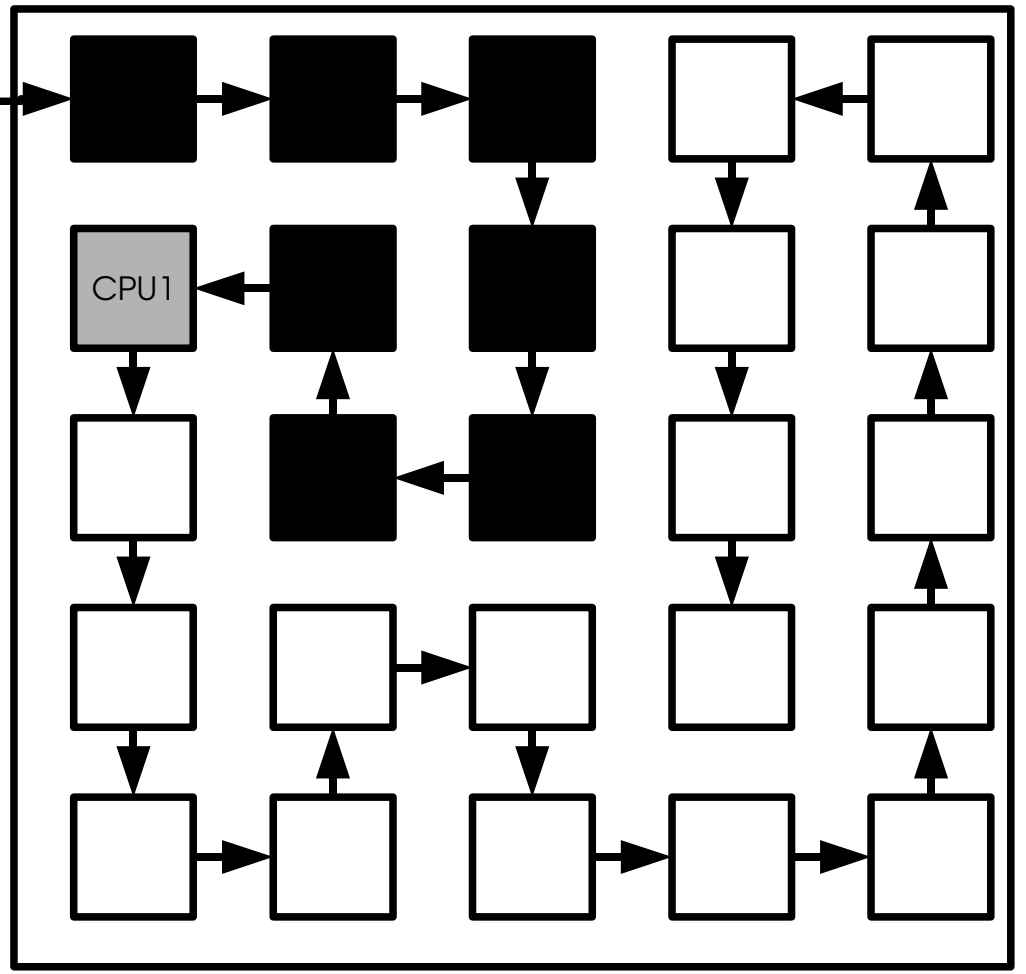
Worst Case: Linked List

CPU1

starts mark step

CPU2

CPU3



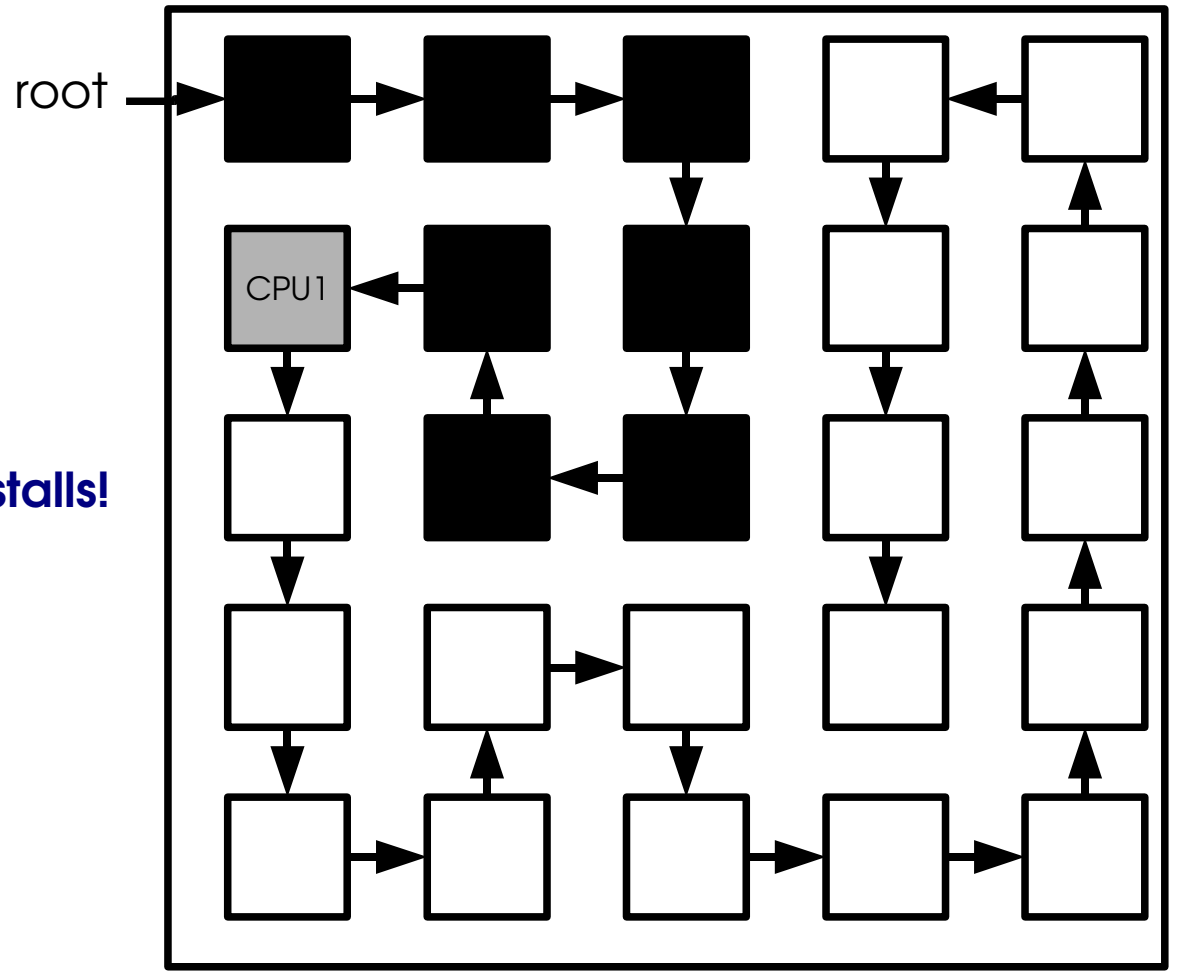
Worst Case: Linked List

CPU1

CPU2

CPU3

no grey object, stalls!



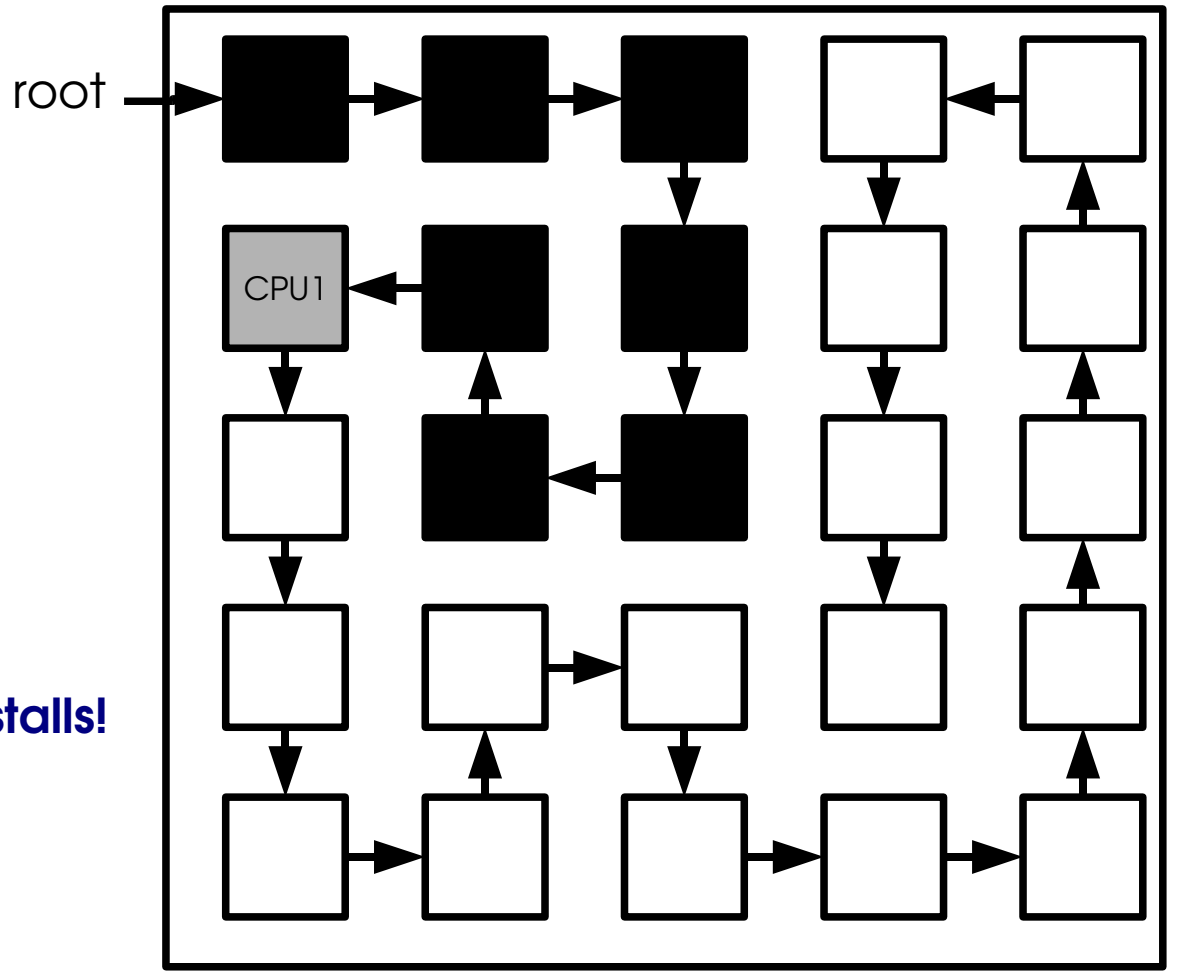
Worst Case: Linked List

CPU1

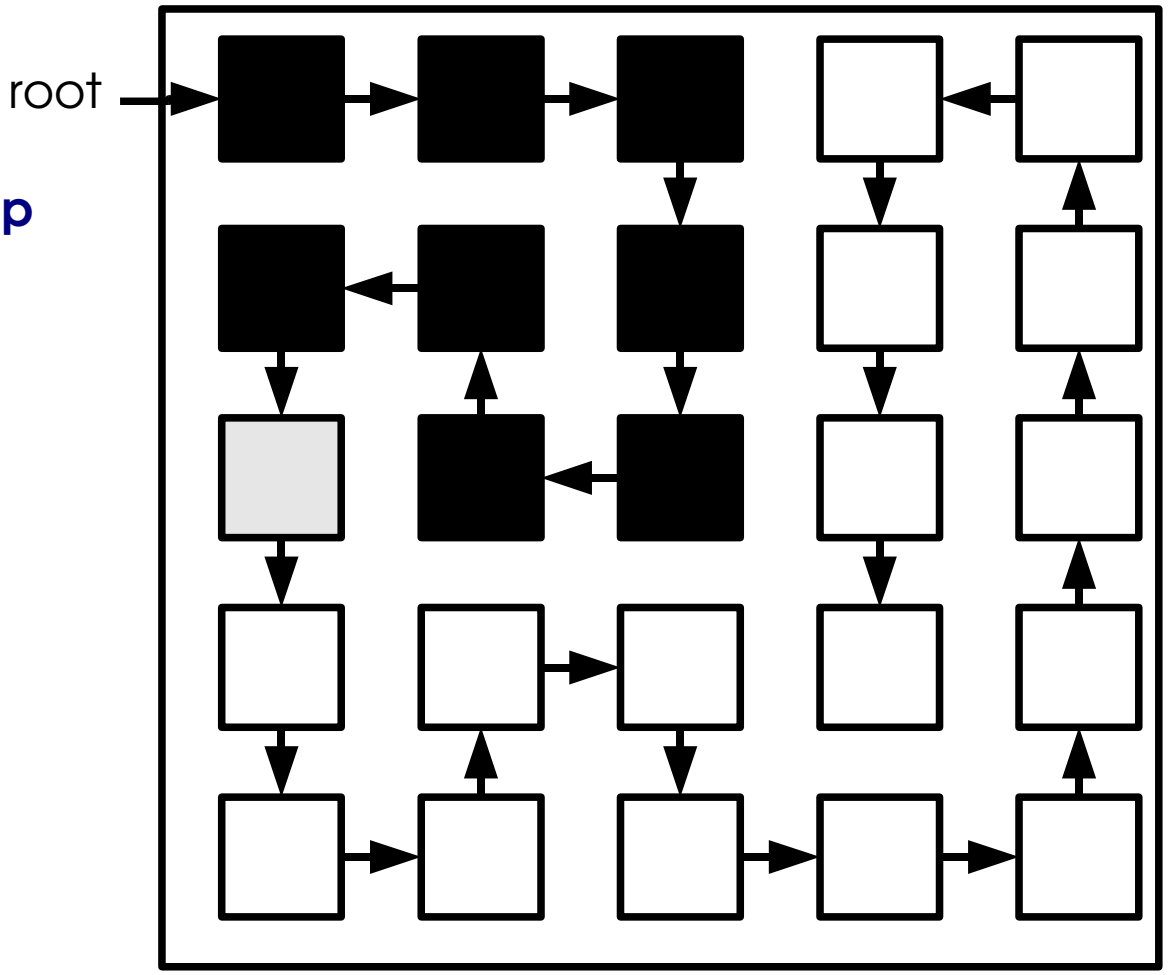
CPU2

CPU3

no grey object, stalls!



Worst Case: Linked List



CPU1

mark step finished

CPU2

CPU3

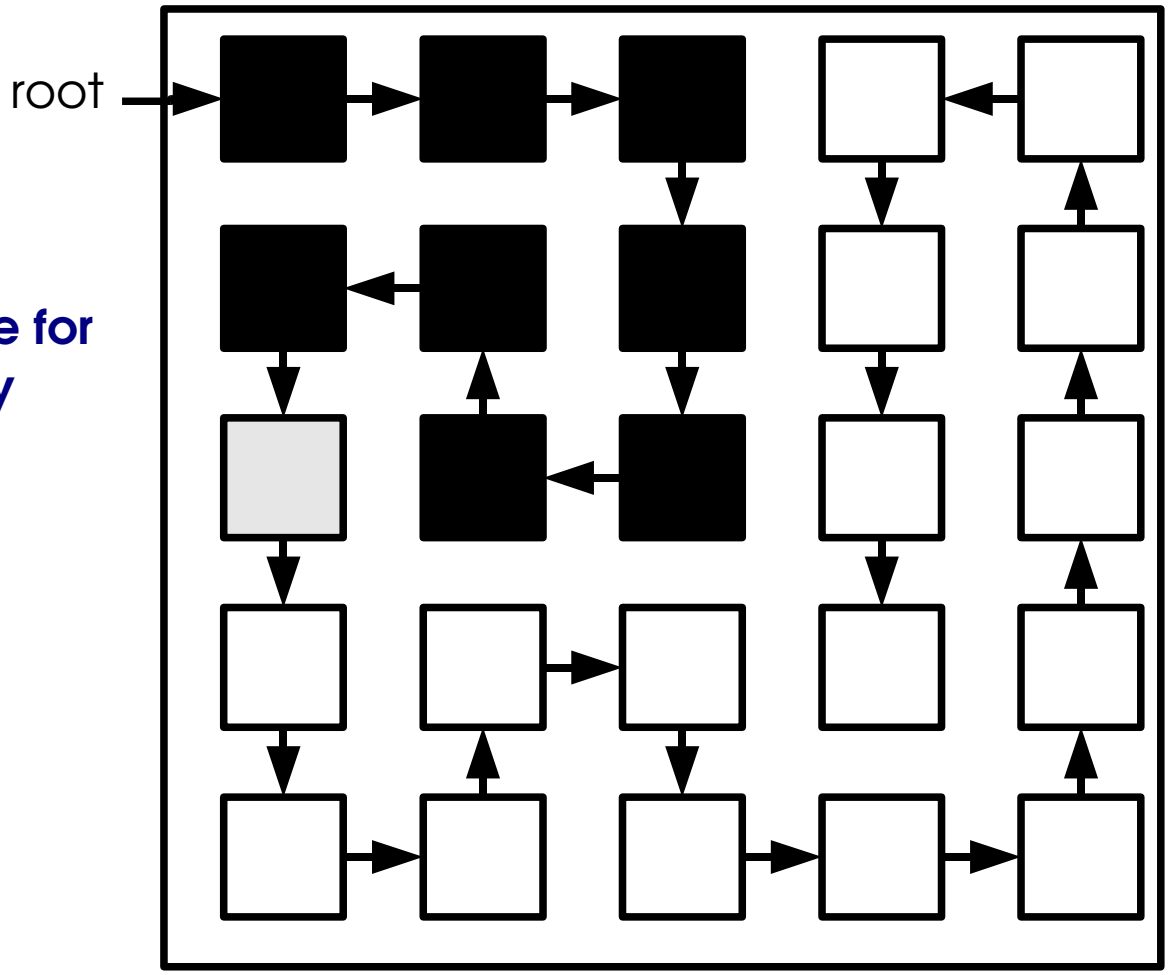
Worst Case: Linked List

CPU1

CPU2

CPU3

all CPUs compete for one grey object!



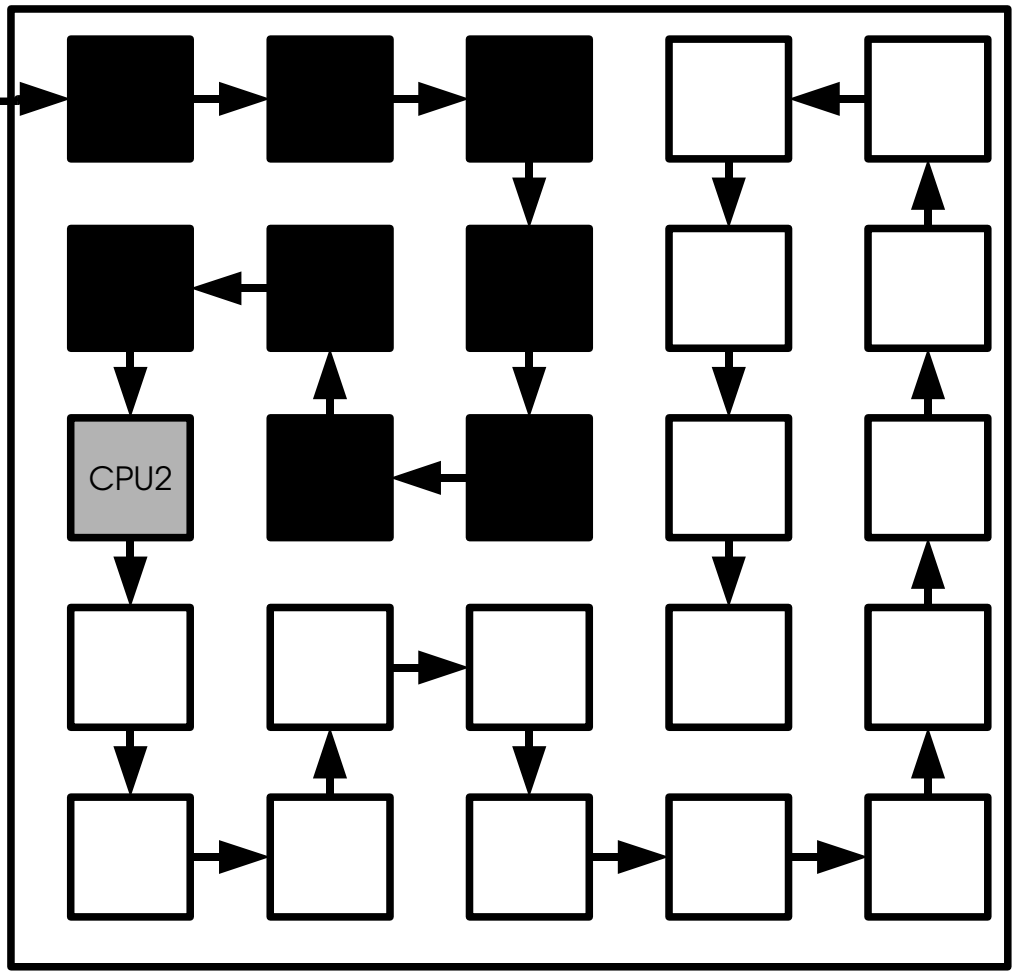
Worst Case: Linked List

CPU1

CPU2

CPU3

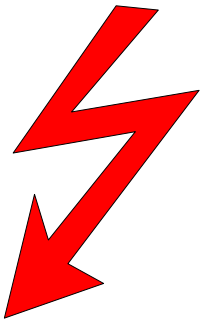
eg., CPU2 successful, CPU1 + CPU3 stall!



Worst Case: Linked List

With *n* CPUs performing mark in parallel

- there might be ***n-1*** stalls for each mark step
- only one CPU is performing a mark step at any time



Worst-case performance equal to non-parallel GC!

Can we find a better limit for real applications?

First, look at two processor parallel mark only

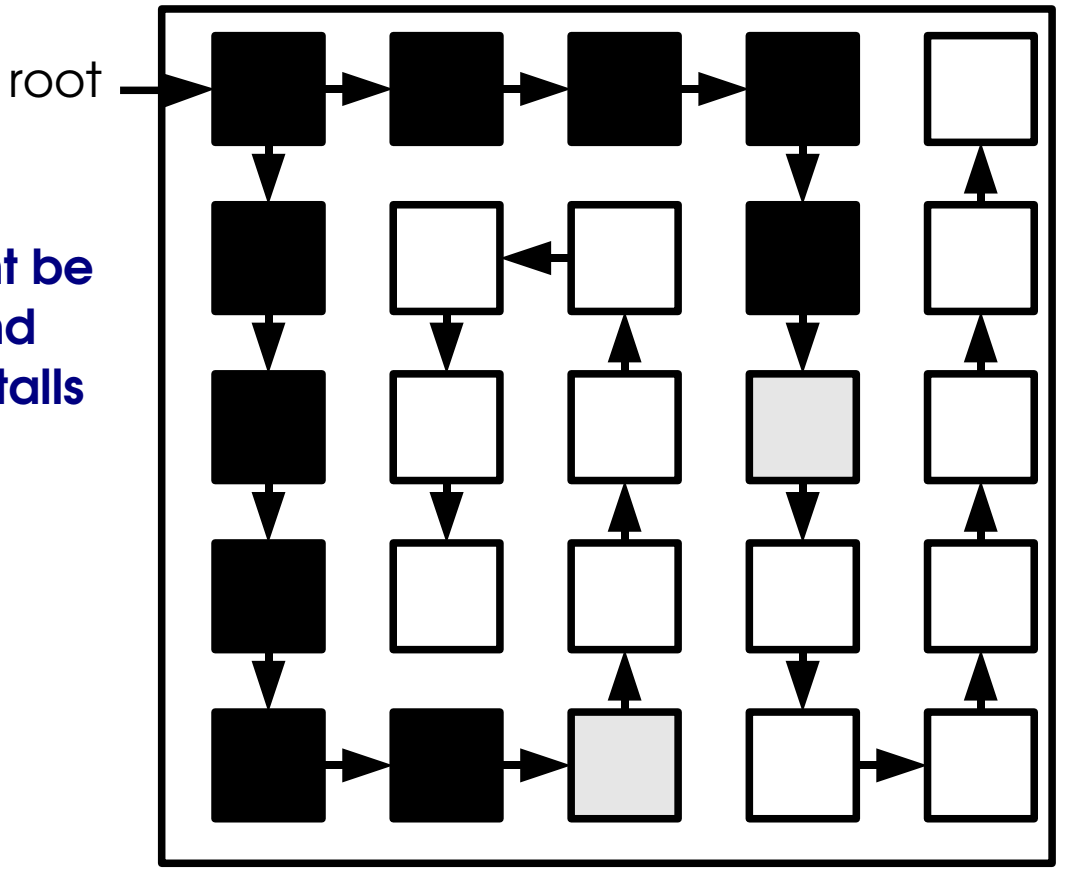
- what if memory graph consists of two linked lists?

Two Linked Lists with two CPUs

CPU1

CPU2

we might be lucky and see no stalls

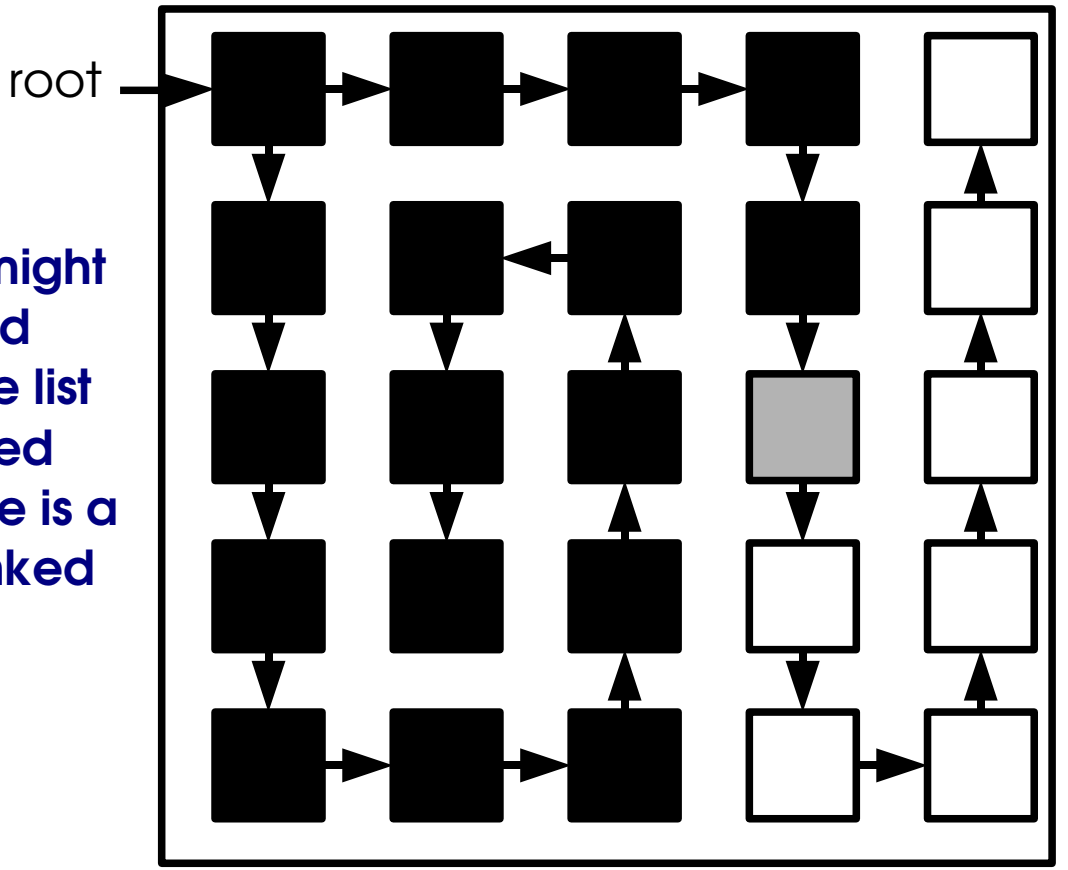


Two Linked Lists with two CPUs

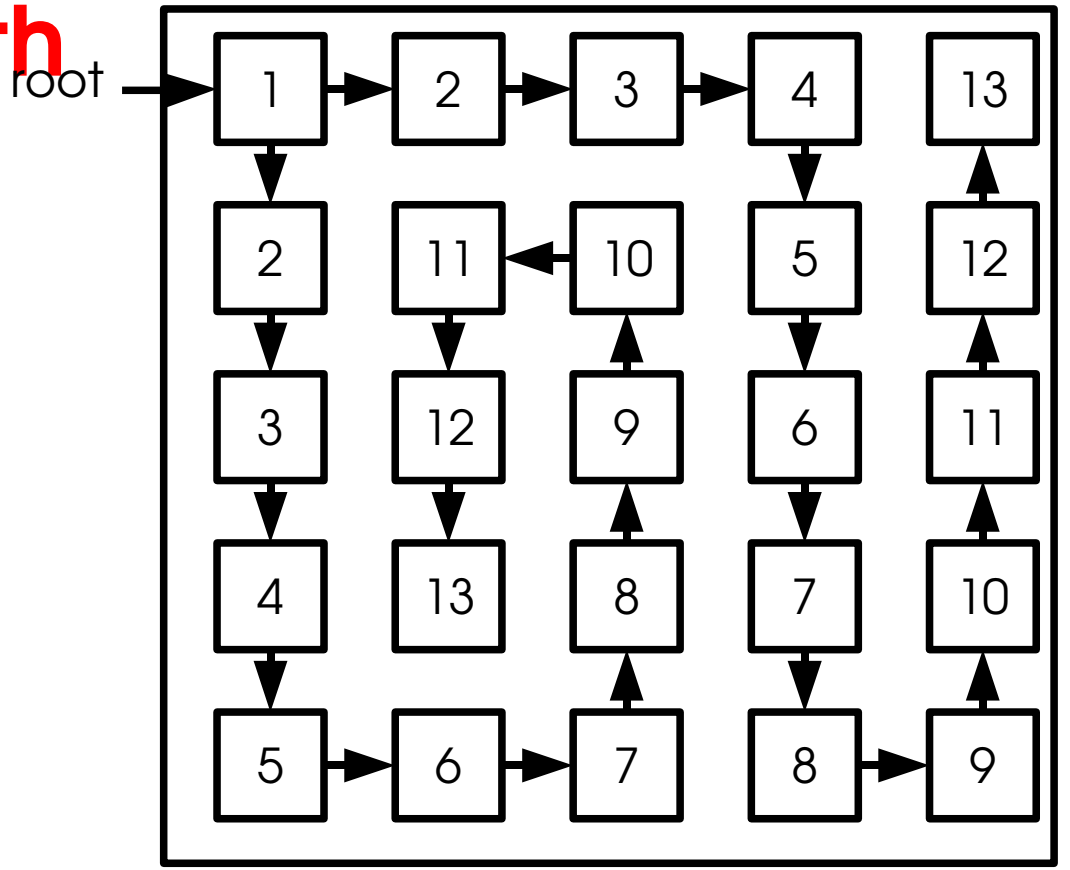
CPU1

CPU2

but we might have bad luck: one list is scanned first, there is a single linked list left!



Limit on stalls depends on object depth



Limit on stalls depends on object depth (2-processors)

- after 1st stall, all objects with depth ≤ 1 are black
- after 2nd stall, all objects with depth ≤ 2 are black
- etc.
- after *n*th stall, all objects with depth ≤ *n* are black

Limit on stalls depends on object depth (2-processors)

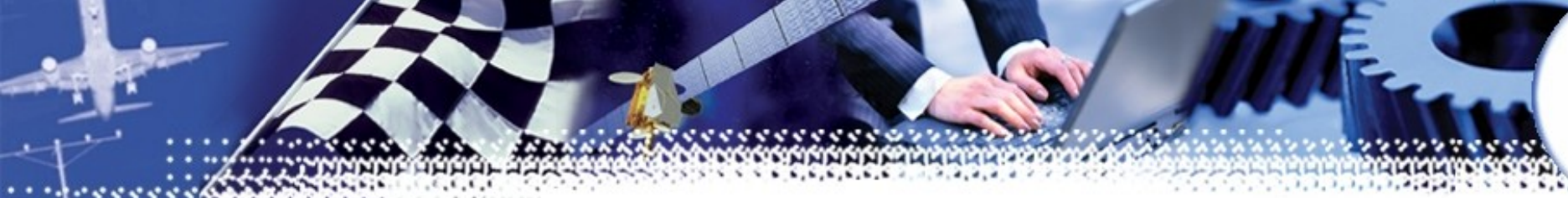
# of stalls *s* on two-processor parallel mark is limited by max. depth of the memory graph H:

$$s \leq \max_{b \in R(H)} depth(b, H)$$

Generalization for more processors

# of stalls *s* on *p*-processor parallel mark is limited by:

$$s \leq (p-1) \cdot \max_{b \in R(H)} depth(b, H)$$



Analysis and Measurements

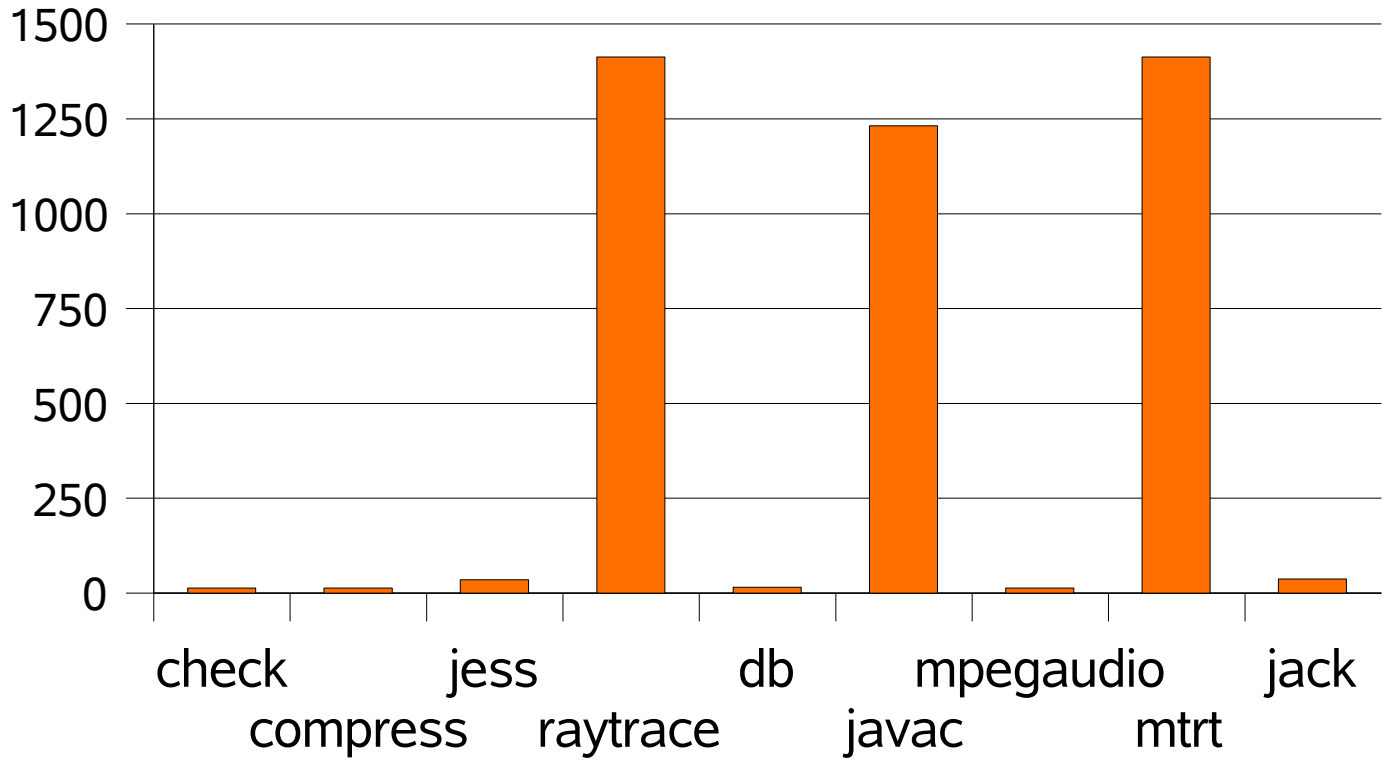
Instrumented JamaicaVM Java implementation to

- measure the maximum depth of the heap graph,
- make samples of the current heap graph all 10,000 reference store operations, and
- output the maximum depths and the maximum ratios depth / heap size in # of objects

The instrumented VM was then used to run the SPECjvm98 benchmark suite

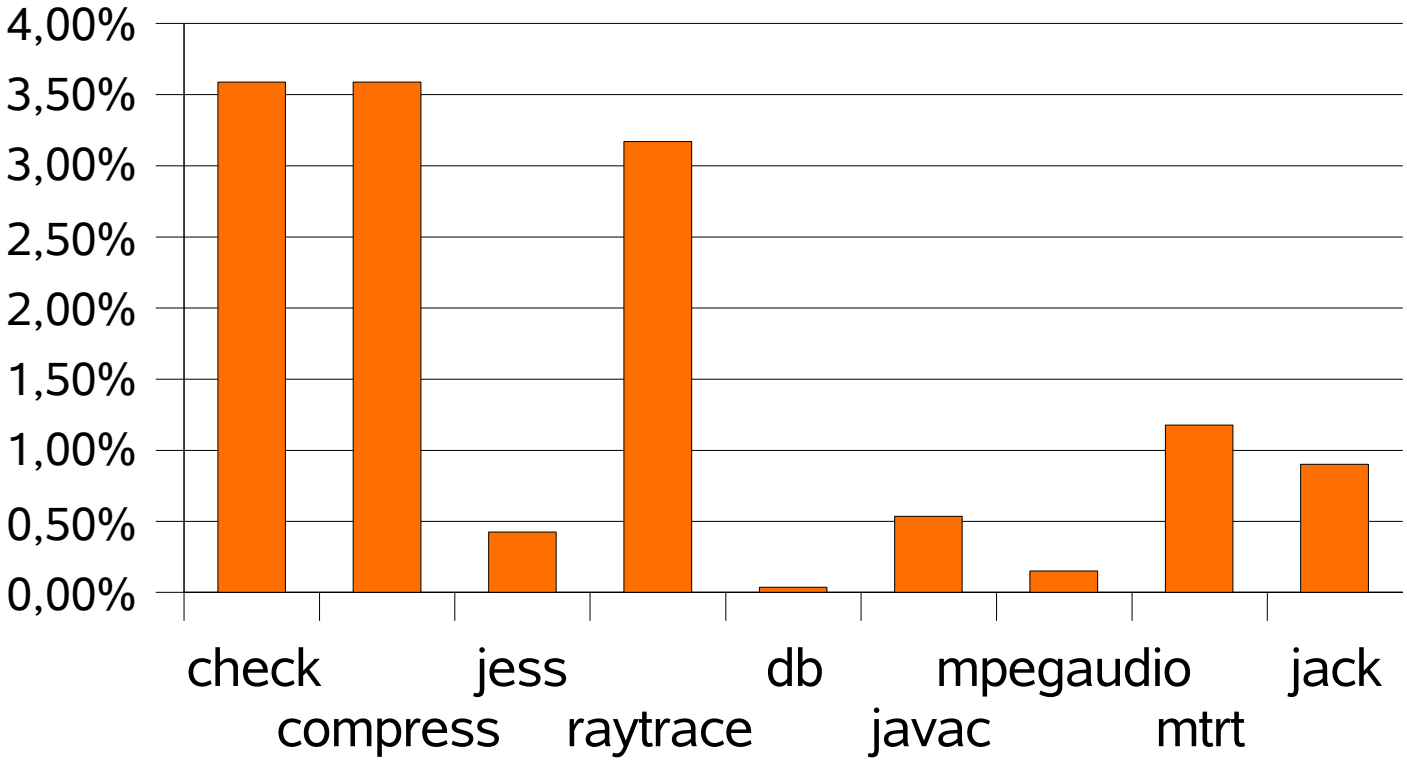
Measurements

Maximum depths of SPECjvm98 benchmarks



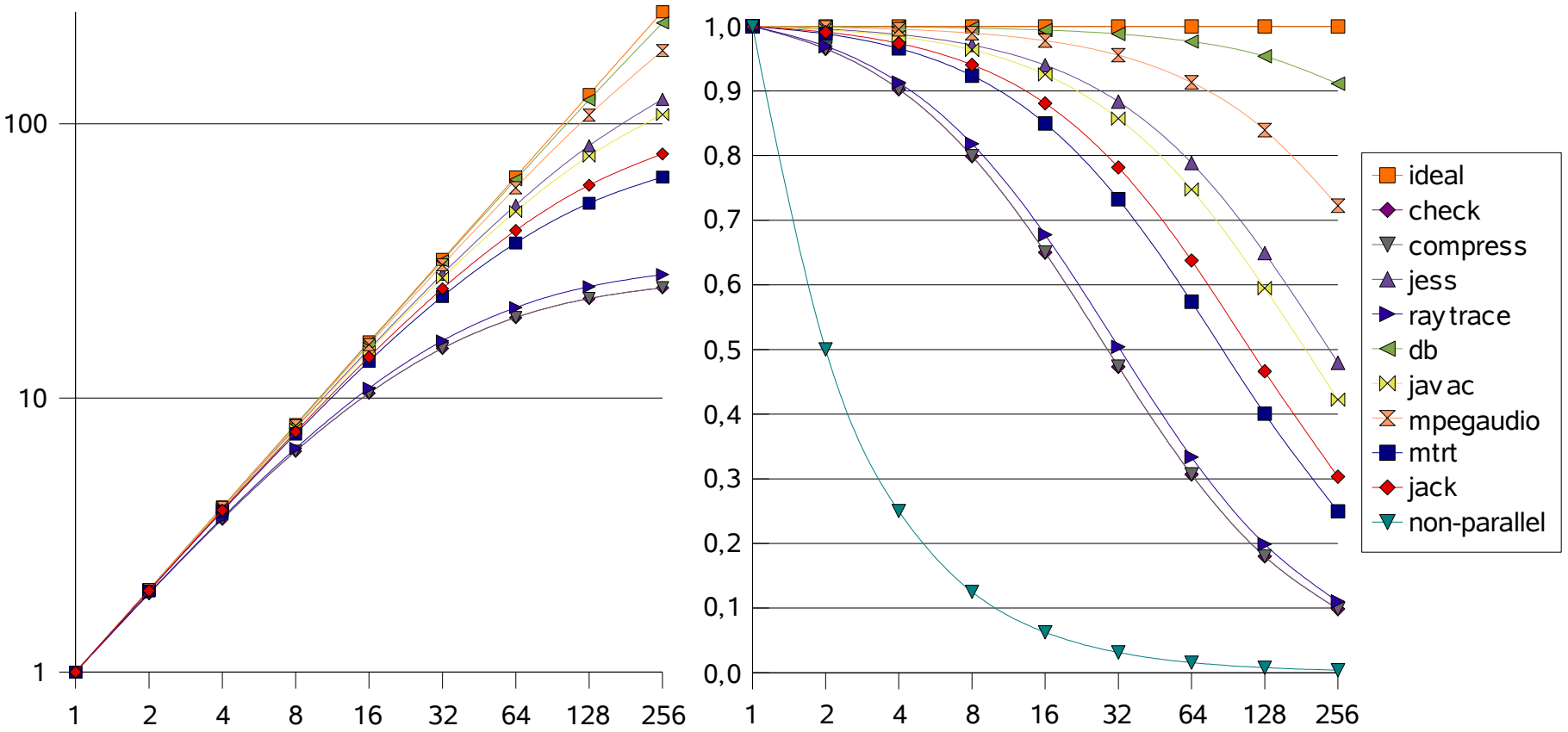
Measurements

Maximum relative depths of SPECjvm98 benchmarks



Measurements

Worst-case scalability of SPECjvm98 benchmarks



Conclusions

In the general case, parallel marking garbage collection can not be parallelized.

However, if the depth of the memory graph is limited, then parallel mark phase generally works well.

To be able to give realtime guarantees on the performance of the mark phase, we need a guarantee from the application on its maximum heap depth.