



Universität Stuttgart

INSTITUT FÜR  
KOMMUNIKATIONSNETZE  
UND RECHNERSYSTEME  
Prof. Dr.-Ing. Dr. h. c. mult. P. J. Kühn

# A True Hardware Read Barrier

Matthias Meyer

Institute of Communication Networks and Computer Engineering  
University of Stuttgart, Germany  
[matthias.meyer@ikr.uni-stuttgart.de](mailto:matthias.meyer@ikr.uni-stuttgart.de)

International Symposium on Memory Management

June 10–11, 2006

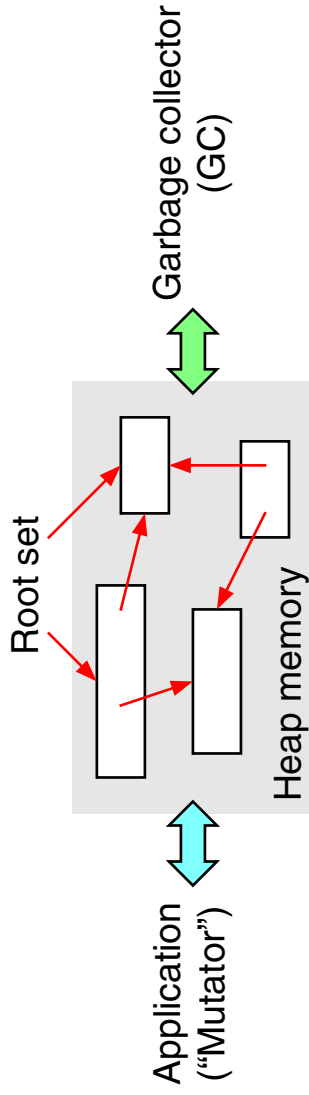
Ottawa, Canada

## A True Hardware Read Barrier

- ❑ Real-time garbage collection: The synchronization problem
- ❑ A hardware-supported approach
  - x Novel processor architecture
  - x Garbage collection coprocessor
  - x Prototype
- ❑ The read barrier
  - x Effect on mutator progress
  - x A closer look at the read barrier fault handler
  - x Novel hardware read barrier design
- ❑ Conclusions and further work

# Real-Time Garbage Collection

## The synchronization problem



- ❑ Mutator and GC modify graph of objects
  - ➡ read or write barriers
- ❑ Mutator and GC access same object
  - ➡ mechanisms for mutual exclusion
  - ➡ or atomic processing of objects
- ❑ Critical regions (root set processing)
  - ➡ unbounded pauses

➡ high synchronization overhead ➡ no hard real-time capabilities

## Basic idea

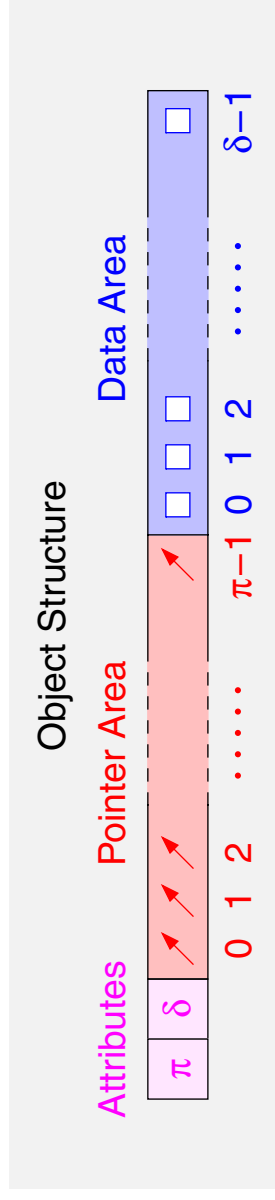
- ❑ Hide garbage collection at the assembly language level
- ❑ Efficiently realize garbage collection and synchronization in hardware

## Precondition

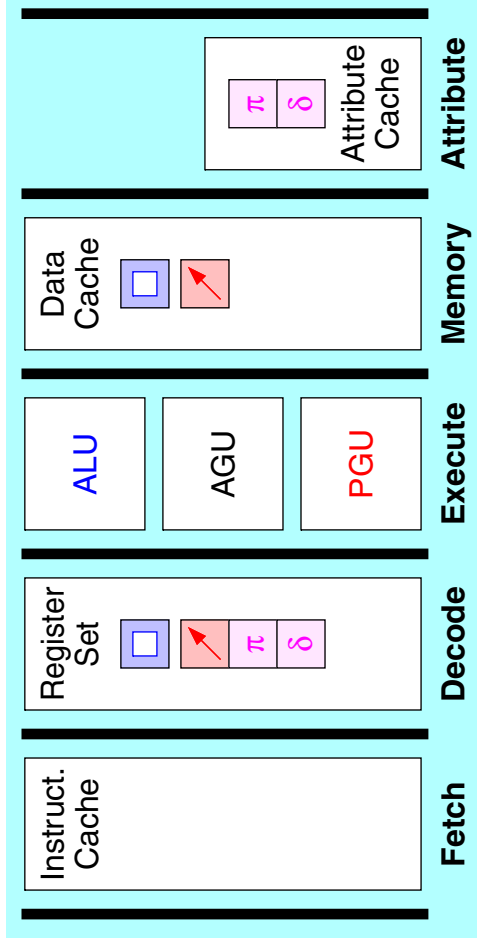
- ❑ Knowledge of pointers and objects in hardware

## Novel approach

- ❑ Strictly separate pointers from non-pointer data
  - $x$  in the register file
  - $x$  in the instruction set
  - $x$  in memory

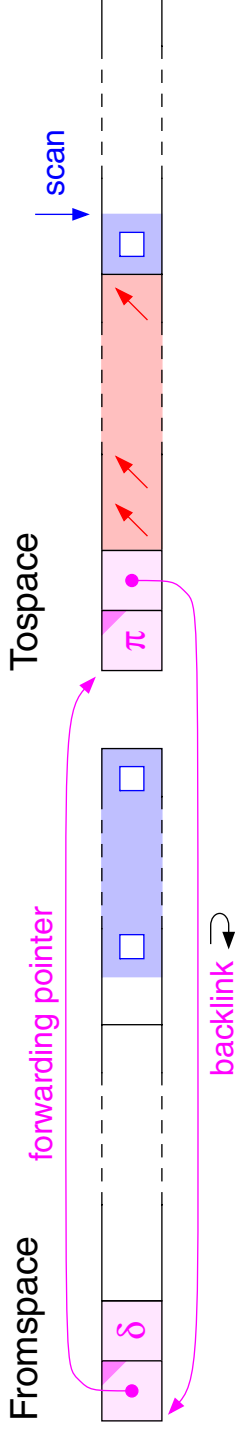




### Extensions to a classical RISC pipeline

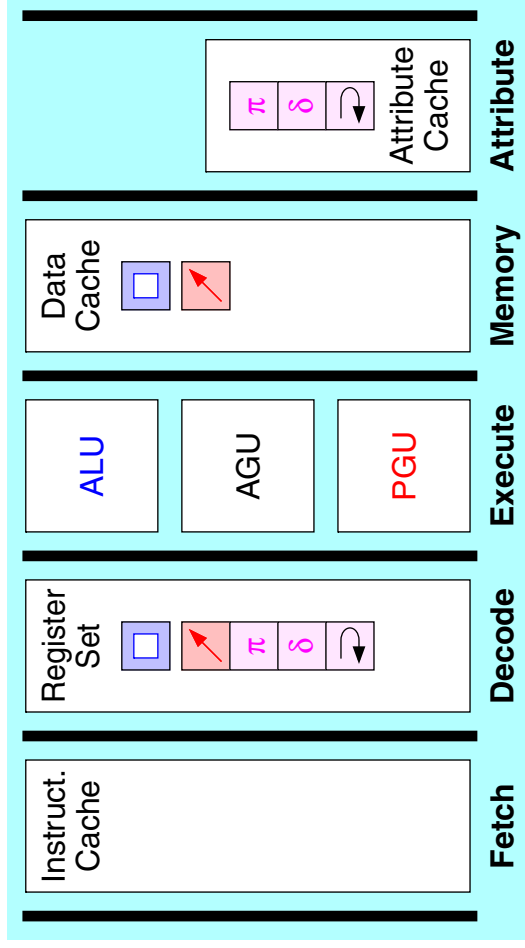


- ❑ Separate data and pointer registers
- ❑ Extend pointer registers by attributes
- ❑ Add PGU for operations that generate pointers (allocate, copy pointer)
- ❑ Add attribute stage for efficient attribute access

## Support for concurrent compaction

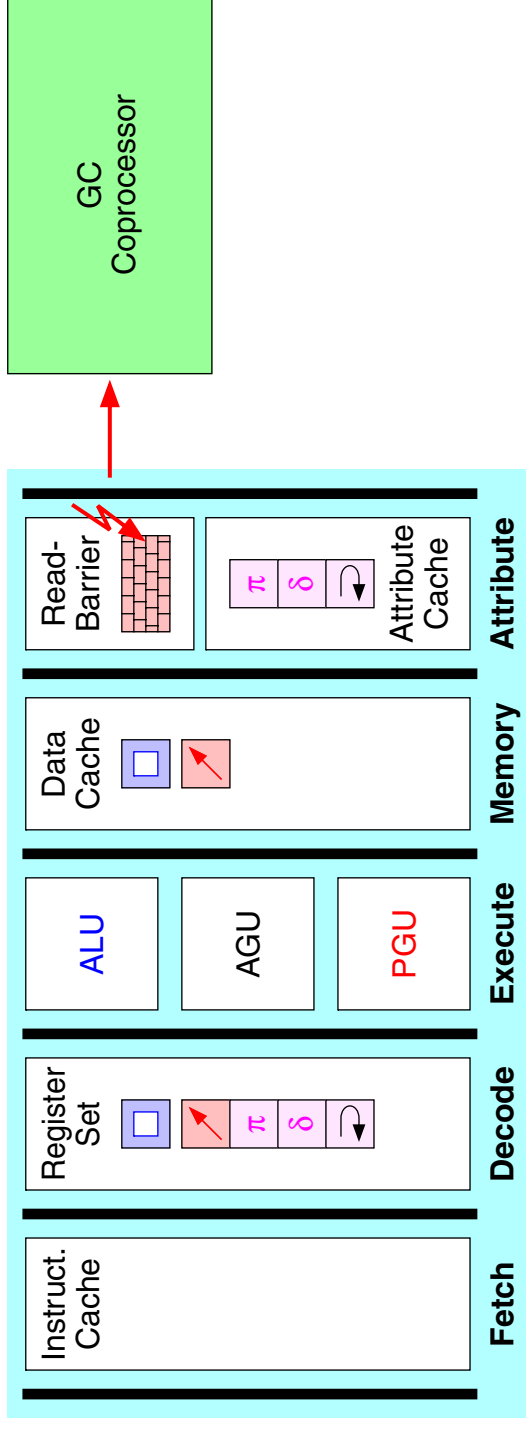


- ❑ Extend pointer register set by backlink entry
- ❑ Extend attribute cache by backlink entry
- ❑ AGU dynamically uses tospace pointer  or backlink  for address generation



### The read barrier

- ❑ Two comparators check loaded pointers (hardware read barrier)
- ❑ Read barrier will trigger interrupt if loaded pointer refers to fromspace
- ❑ Interrupt handled by a dedicated garbage collection coprocessor



# Garbage Collection Coprocessor

## Features

- ✗ performs garbage collection concurrently with application processing
- ✗ low cost device, specialized for garbage collection

## Integration

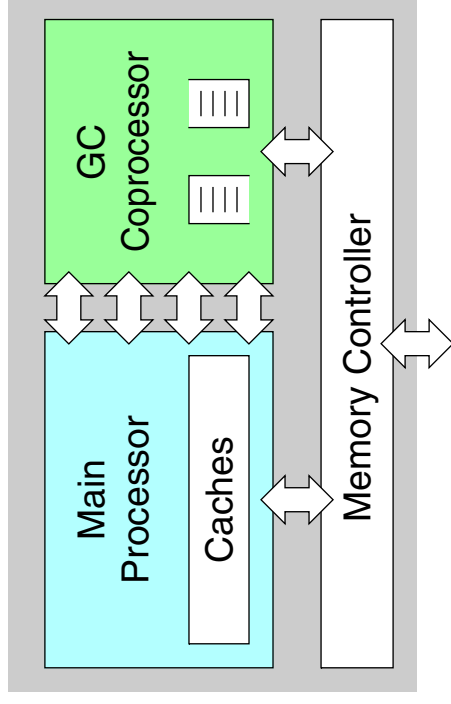
- ✗ tightly coupled to main processor
- ✗ realized on same device
- ✗ separate ports to memory controller

## Memory interface

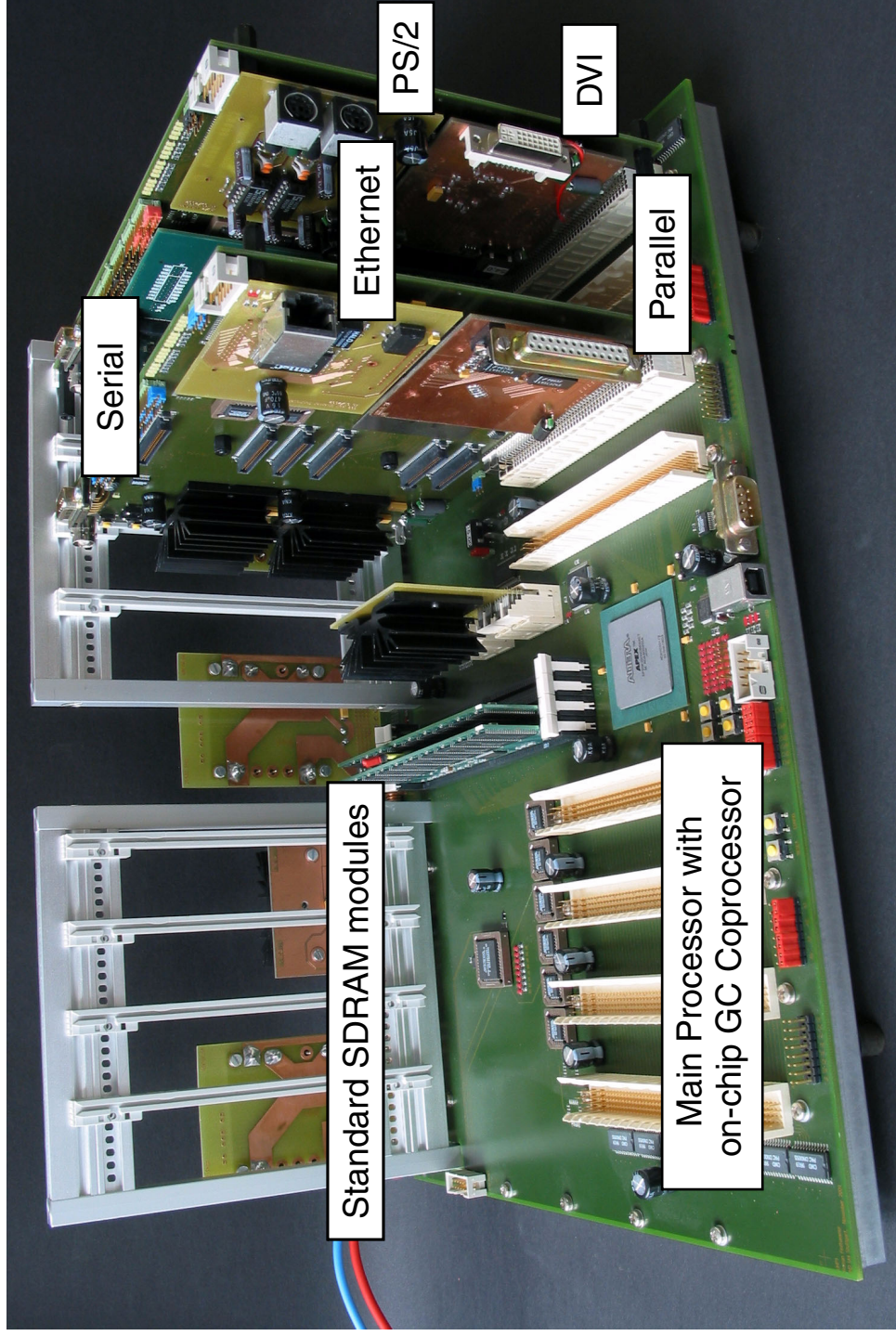
- ✗ no temporal locality: **no cache!**
- ✗ spatial locality: **burst registers!**

## Algorithm

- ✗ based on Baker's algorithm
- ✗ directly implemented in microcode



# Prototype



# Prototype

## Hardware

- Main processor: 3-way multiple issue, “in order”
- GC coprocessor: 256 x 80 bit microcode memory
- Synchronously operated at 25 MHz

## Software

- Static Java compiler (bytecode to machine code)
- Subset of the Java class libraries

## Features

- Low-cost fine-grained synchronization
  - independent of compiler and runtime system
  - no code size overhead, little runtime overhead
- First known system that **limits any GC-related pause to max. 500 clock cycles**

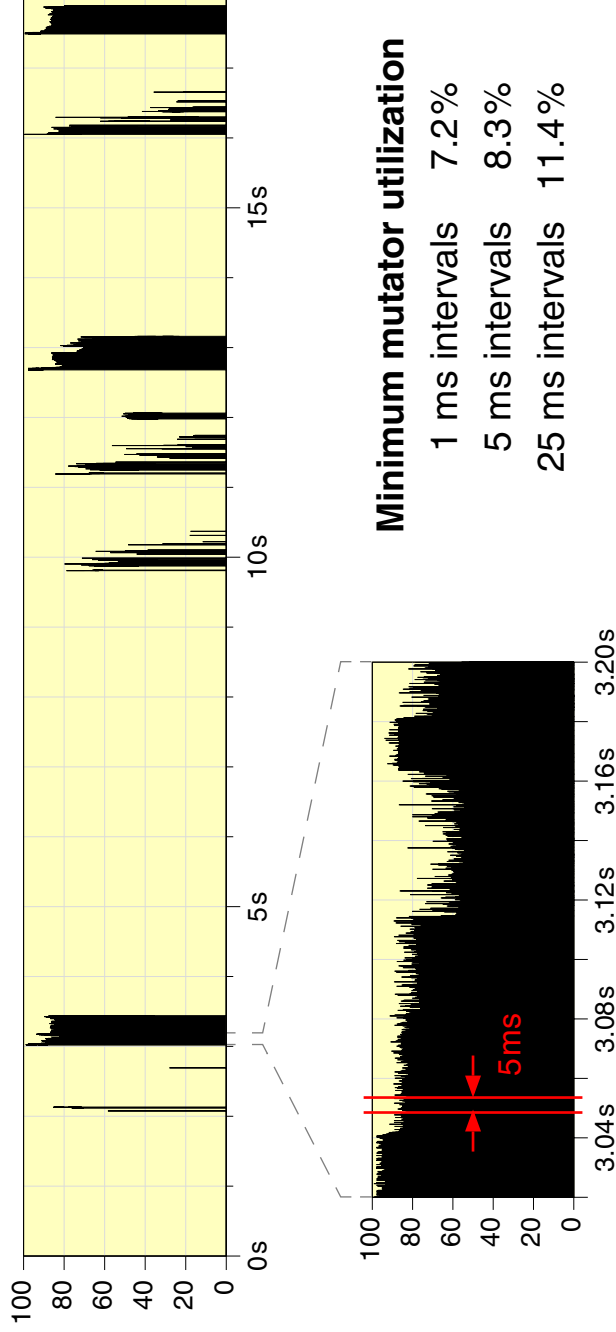
## Question

**How are the pauses distributed over time?**

# Read barrier effect on mutator progress

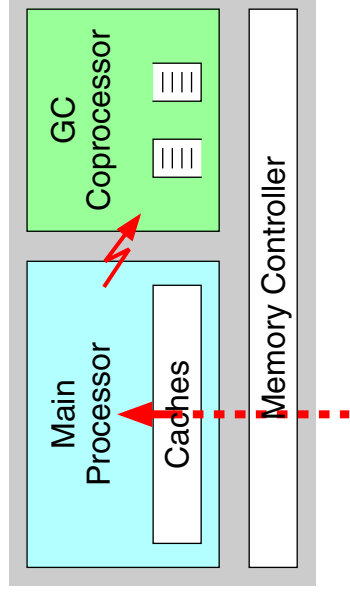
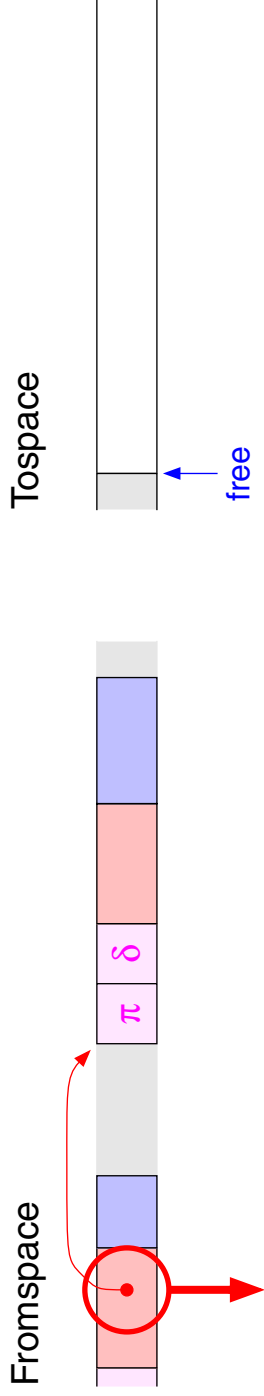
## Experimental results

Percentage of pause cycles (in intervals of 500 clock cycles, benchmark “database”)



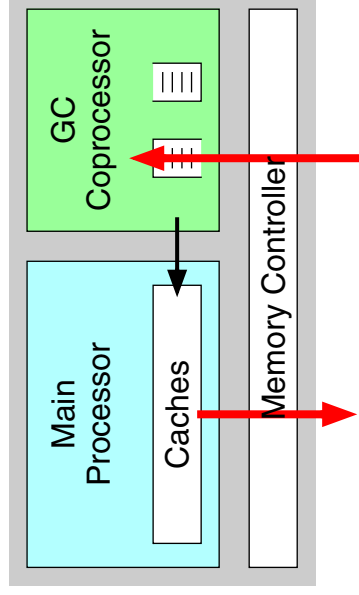
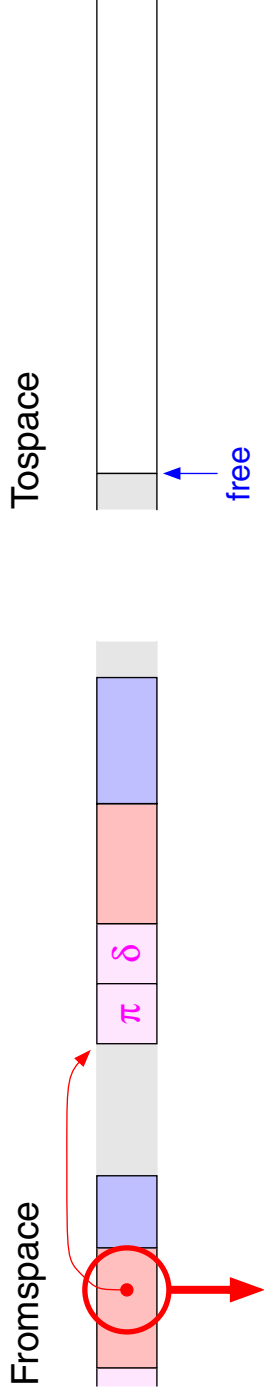
# A closer look at the read barrier fault handler

Trigger: Processor reads fromspace pointer



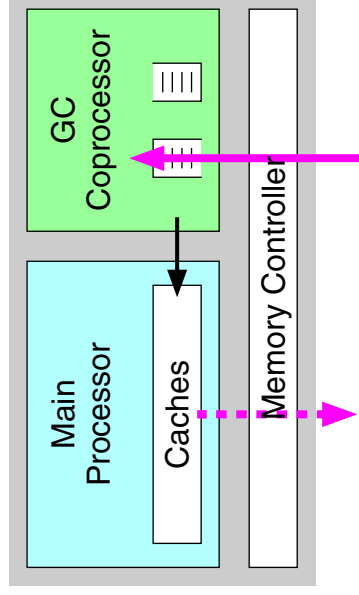
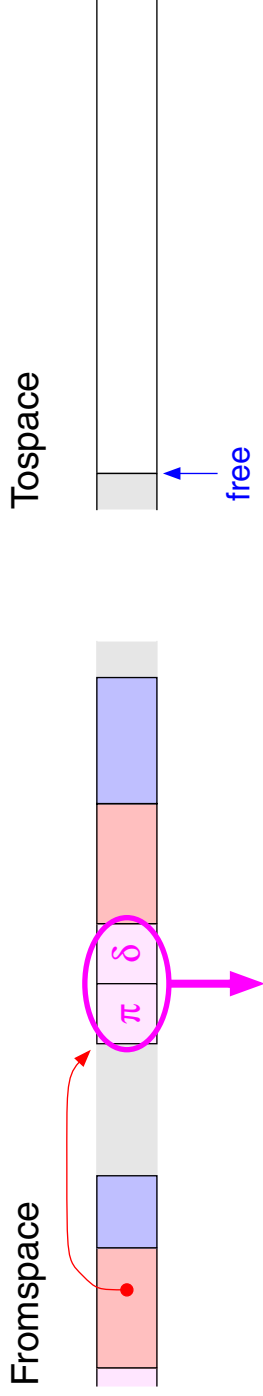
# A closer look at the read barrier fault handler

## Step 1: Coprocessor reads faulting pointer



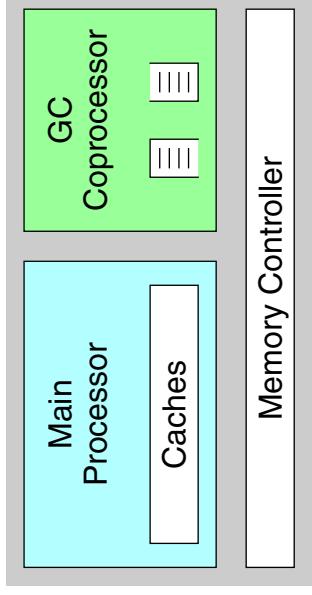
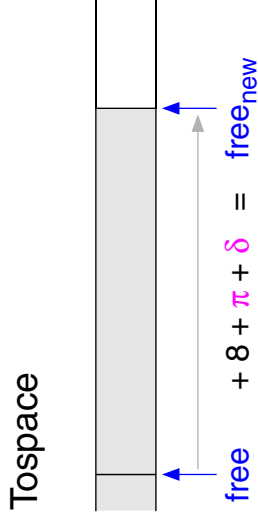
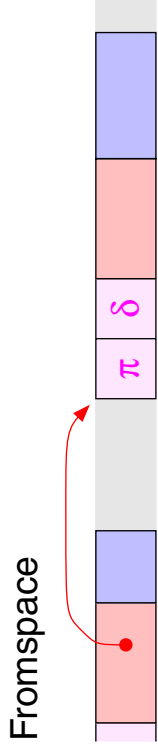
# A closer look at the read barrier fault handler

## Step 2: Coprocessor reads object attributes



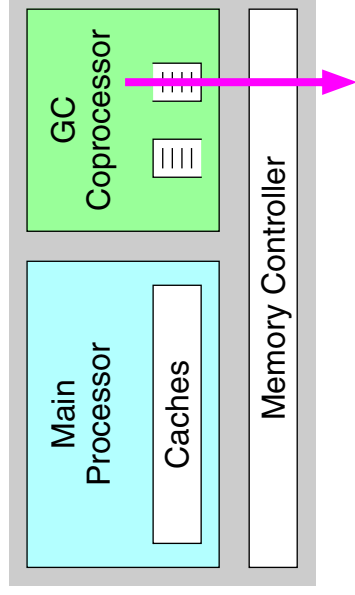
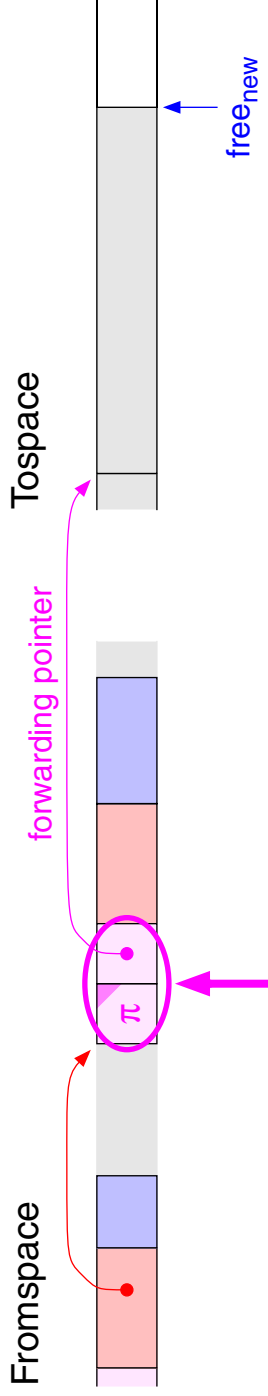
# A closer look at the read barrier fault handler

## Step 3: Coprocessor advances free



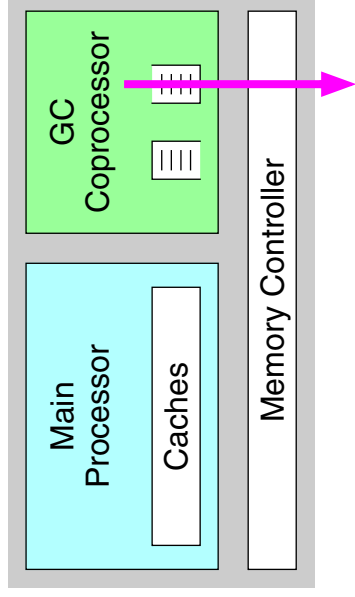
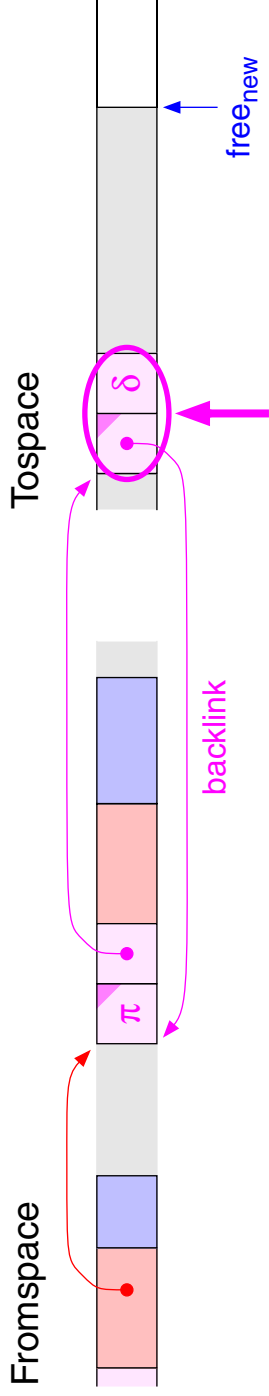
# A closer look at the read barrier fault handler

## Step 4: Coprocessor overwrites fromspace attributes



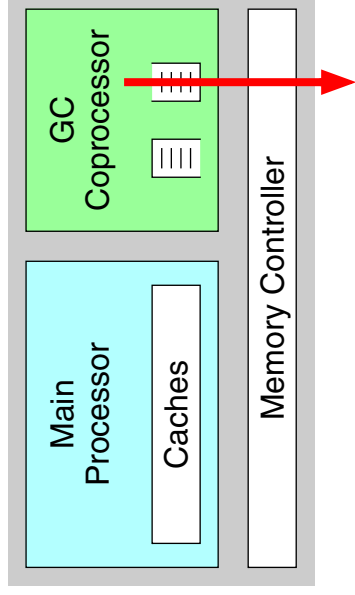
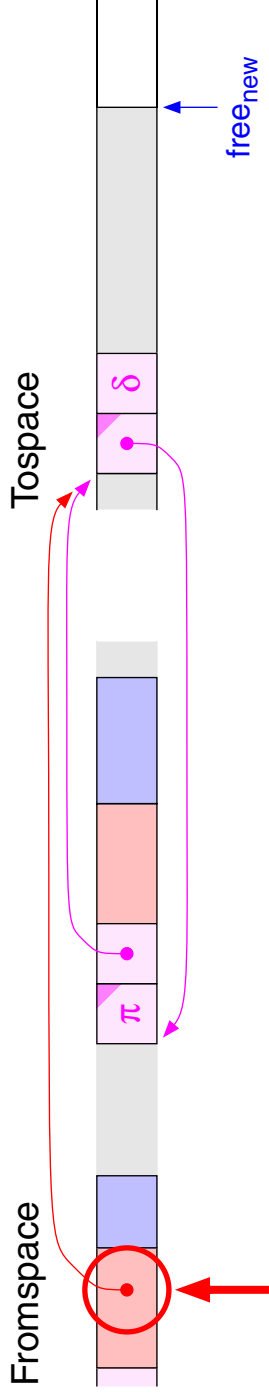
# A closer look at the read barrier fault handler

## Step 5: Coprocessor initializes tospace attributes



# A closer look at the read barrier fault handler

## Step 6: Coprocessor updates fromspace pointer



# A novel hardware read barrier design

## Analysis

- ❑ Read barrier fault handling expensive despite hardware support
- ❑ Necessary to **sacrifice the tospace invariant** to avoid clustering?

## Insights

1. Read barrier in hardware  
*... but read barrier fault handling still in software*
2. Processors expensively communicate via main memory  
*... because faulting pointer local to main processor, not to garbage collector*

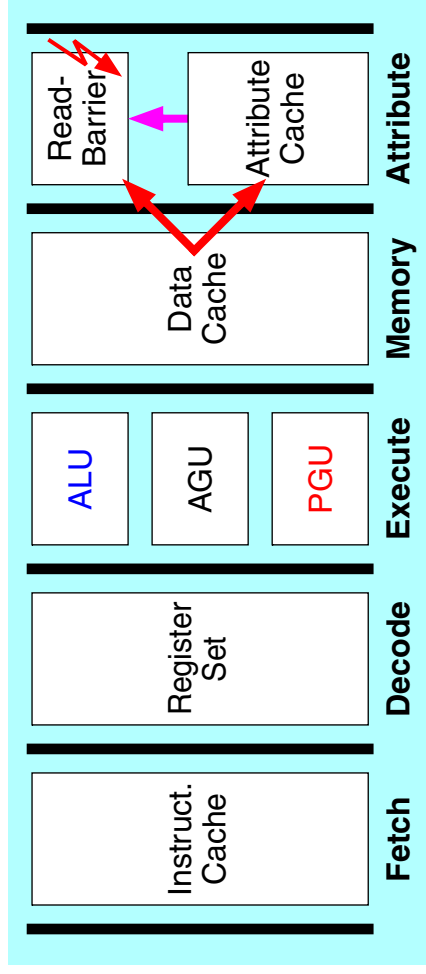
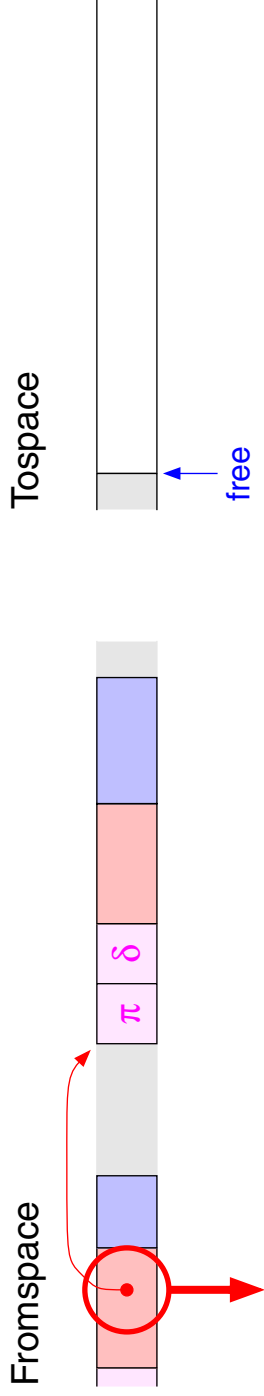
## Novel idea

Live with the clustering, **save the tospace invariant**

1. Increase efficiency of the handler
  - *Realize fault handling completely in hardware!*
2. Resolve the locality issue
  - *Move fault handling to main processor!*

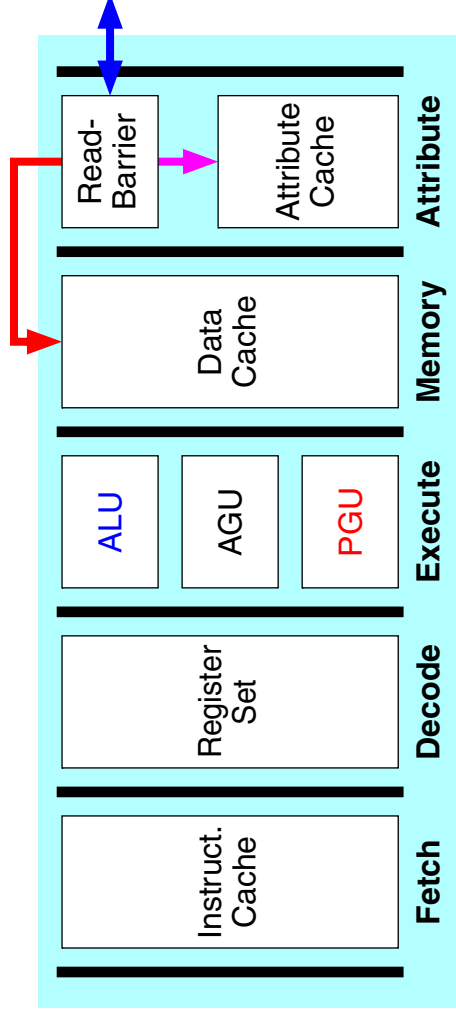
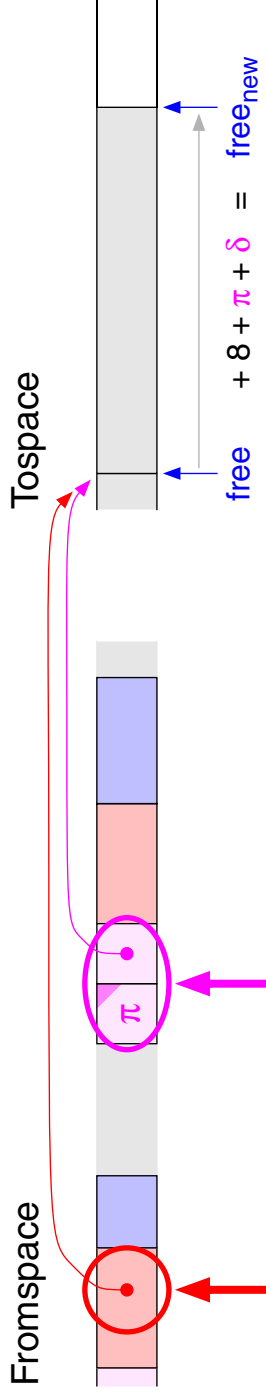
# A novel hardware read barrier design

Trigger: Processor reads fromspace pointer



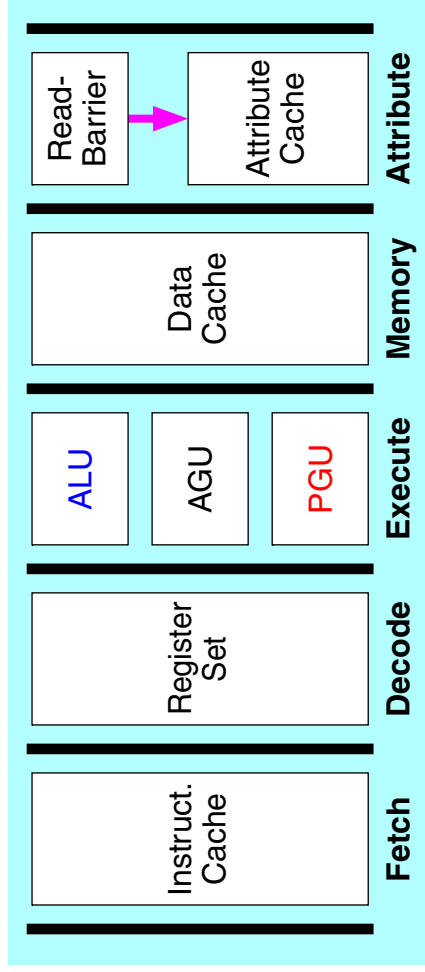
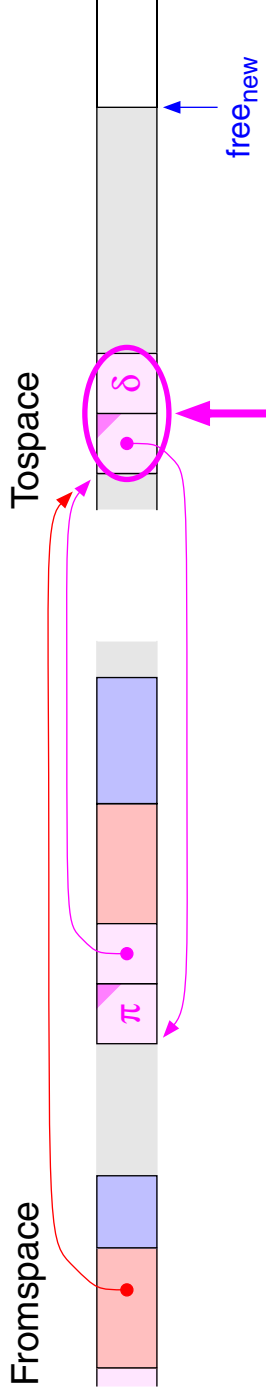
# A novel hardware read barrier design

Step 1: Advance free, write fromspace attributes, update fromspace pointer



# A novel hardware read barrier design

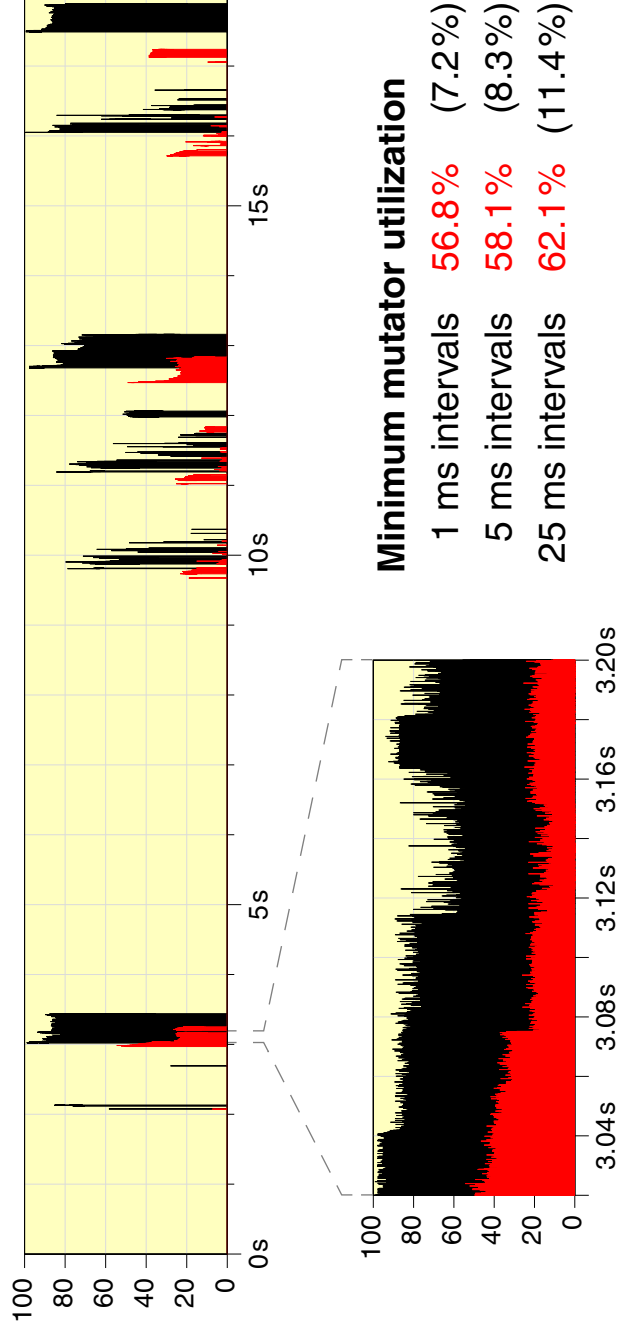
## Step 2: Initialize tospace attributes



# A novel hardware read barrier design

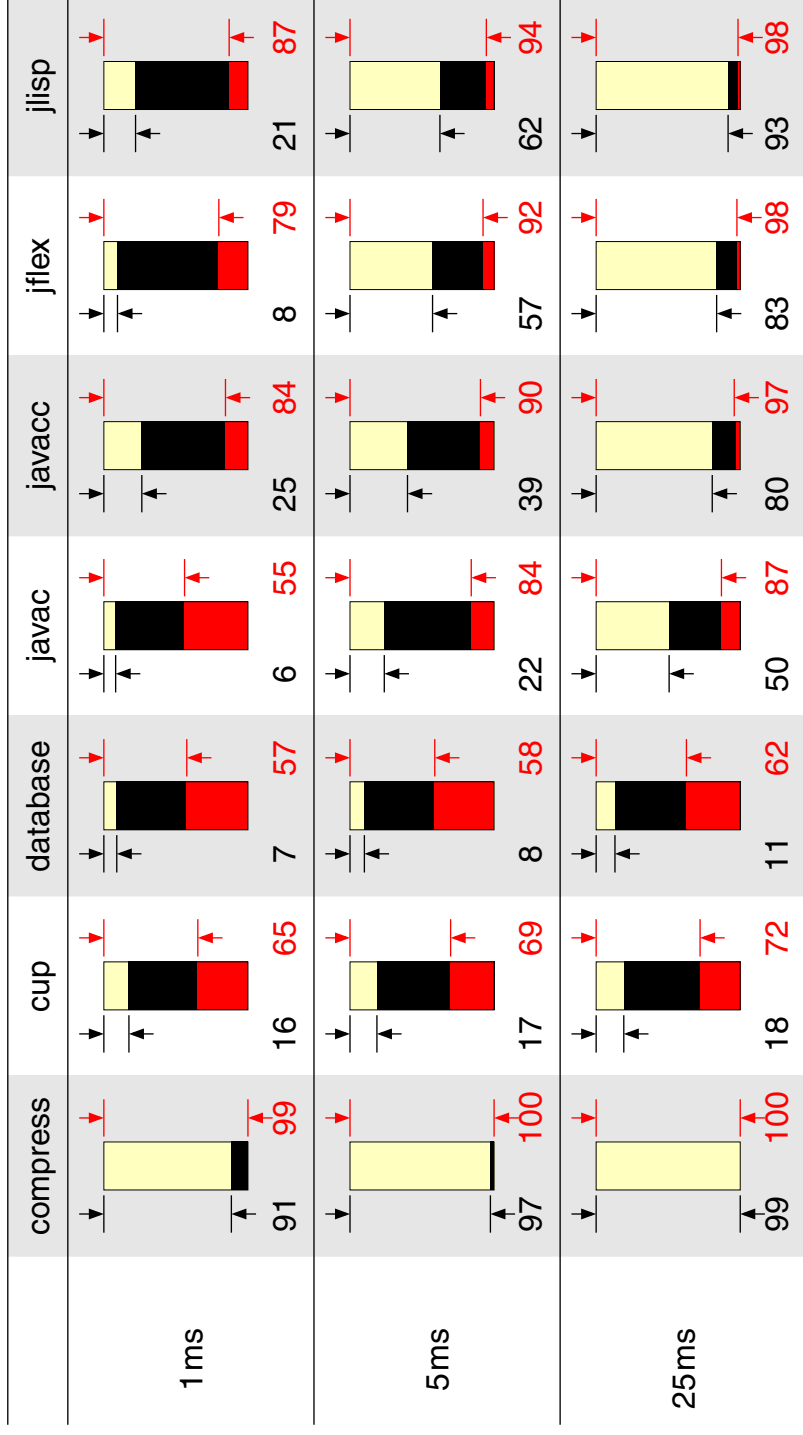
## Experimental results

Percentage of stall cycles within intervals of 500 clock cycles (benchmark “database”)



# A novel hardware read barrier design

Minimum mutator utilization [%] for var. benchmarks and time quantumms



# Conclusions and Further Work

## **The true hardware read barrier**

- ❑ Average handling time: Less than 20 clock cycles
- ❑ Minimum mutator utilization for a time quantum of 1 ms: Greater than 55%
- ❑ Maintains elegance and simplicity of Baker's tospace invariant

## **Processor architecture and GC coprocessor**

- ❑ Exact garbage collection without support from compiler or runtime system
- ❑ High robustness at the machine code level
- ❑ Maximum pause time less than 500 clock cycles
- ❑ Total runtime overhead of garbage collection a few percent only

## **Further work**

- ❑ Generational collector with "true hardware write barrier"
- ❑ Out-of-order implementation of the processor architecture

## Distribution of pause lengths

