Memory management for big data

Gaël Thomas, professor at Telecom SudParis

I'm a researcher in system

- 2015 – today: Professor at Telecom SudParis/Paris
  Language runtimes, multicore, parallelism, HPC, hypervisors
- 2006 – 2015: Ass. prof. at UPMC Sorbonne Univ./Paris
  Language runtimes, multicore, parallelism
- 2005 – 2006: PostDoc at LIG/Grenoble
  Distributed systems
- 2001 – 2005: PhD at UPMC Sorbonne Univ./Paris
  Design and implementation of Java virtual machines

And I like doing systems!

gthomas@archlinux:~/research/vrack/src

```
  ACPI_OBJECT obj;

  parms.Count = 1;
  parms.Pointer = &obj;

  obj.Type = ACPI_TYPE_INTEGER;
  obj.Integer.Value = 1;

  if(ACPI_FAILURE(s = AcpiEvaluateObject(ACPI_ROOT_OBJECT, (char*)"_PIC", &parms,\
 0)))
    panic("unable to switch to acpi apic mode (error %d)\n", s);
}

void ACPIDriver::add(Device* parent, Domain* domain, void* info) {
  printk("Attaching ACPI bus to ");
  parent->printName();
  printk("\n");

  /* inform acpi that we are using IO-Apic  */
  ACPIDevice* dev = new ACPIDevice(parent, domain);

  enterAcpiApicMode();

  void* res;
  AcpiWalkNamespace(ACPI_TYPE_DEVICE, ACPI_ROOT_OBJECT, 100, visitDescending, vis\
-UU-:----F1  acpi-driver.cc    6% (38,43)   Git:master  (C++/l Abbrev) ----------
Find file: ~/research/vrack/src/drivers/acpi/
```

Memory management for big data

Data, data, data

- Amount of data increases exponentially
  - Web (facebook, gmail, google, Amazon…)
  - Devices (Waze, healthcare monitoring, banking…)
  - Science (Large Hadron Collider…)

Data analytics

- Analyzing this data is a key ingredient in many domains
  - Market analysis, banking, scientific computations…

Analyzing big data requires
efficient and powerful computing infrastructures

To illustrate, the Large Hadron Collider generates 1 PB each day
(~ 1,000 hard drives)

But achieving performance is difficult

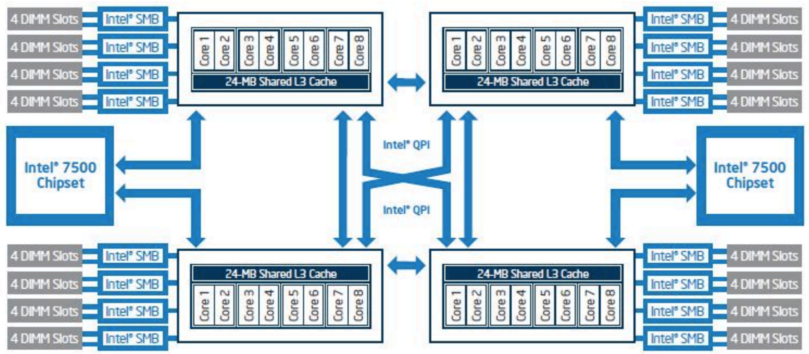
(even with data analytics algorithms of genius)

Because infrastructures are complex…

- Data centers are geo-distributed

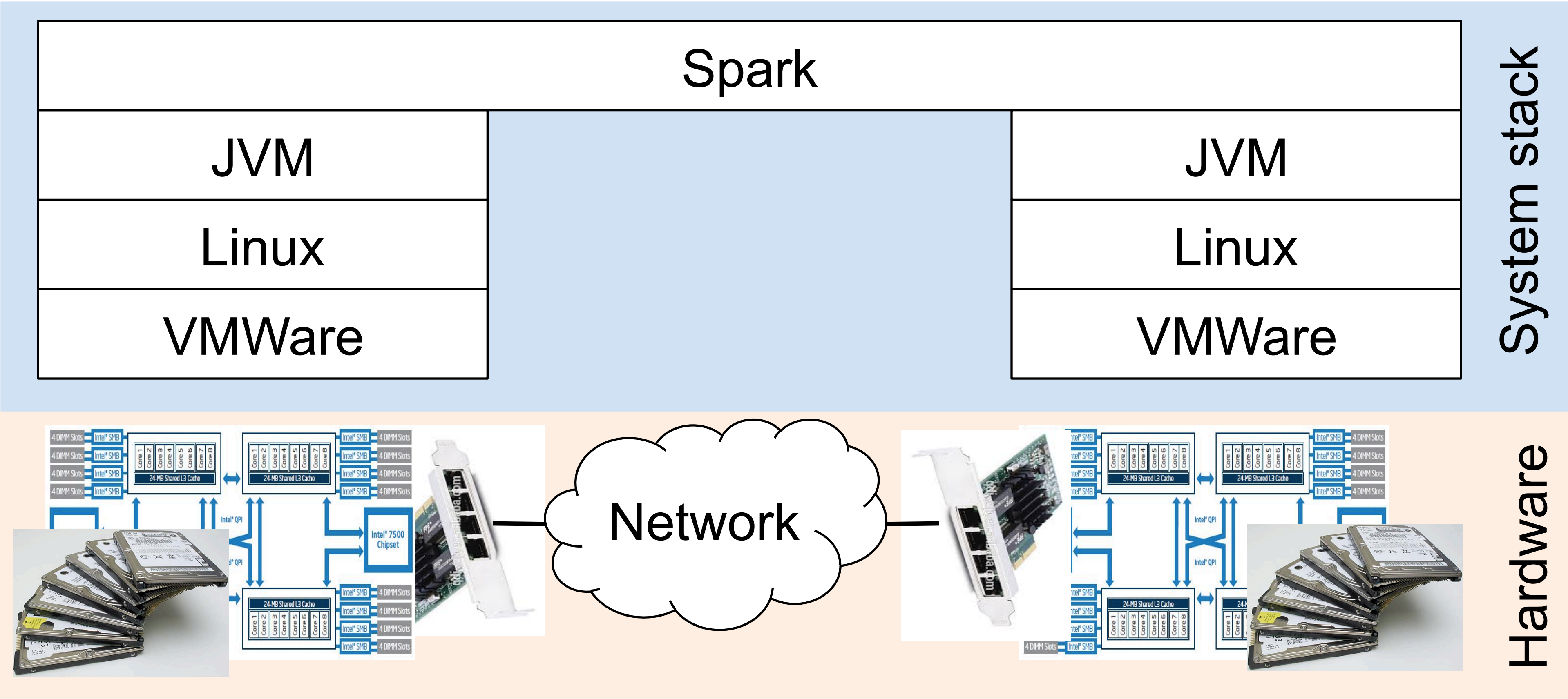
- Each data center contains a complex computers infrastructure

- Each computer is itself a distributed network



… and system stacks are complex

A typical system stack includes more than 10^7 lines of code



How can we achieve better efficiency?

By building efficient system stacks
for big-data analytics ☺

A typical research work

Work of Lokesh Gidra (defense the 2015 28th september)
Now research engineer at HP labs at Palo Alto, CA, USA

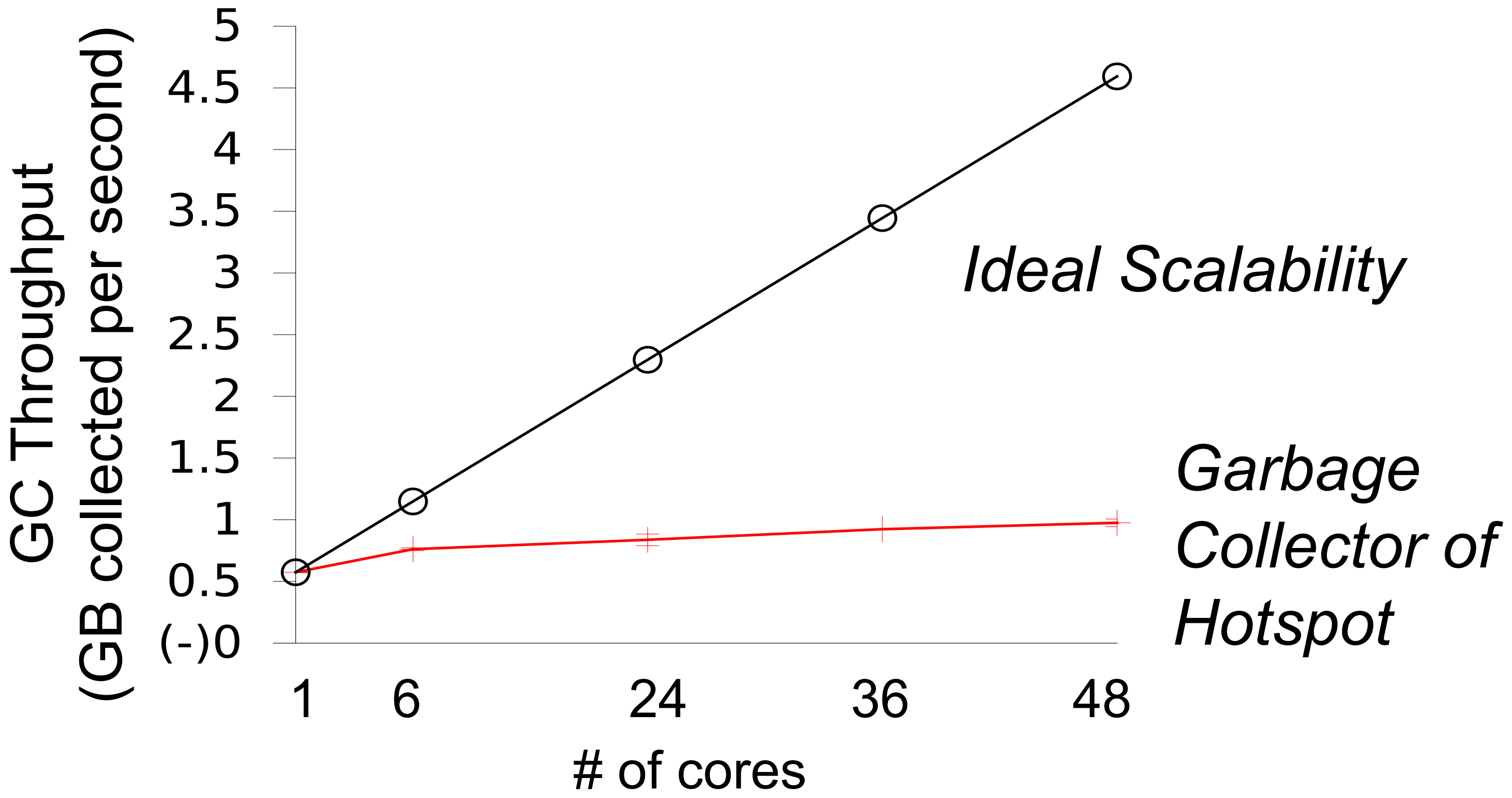
Big data and memory management

Page rank computation with Spark on 10^8 nodes

Memory management of the JVM on a modern 48-core
takes roughly 60% of execution time
while it takes less than 10% on a 4-core
(heap size is 40GB)

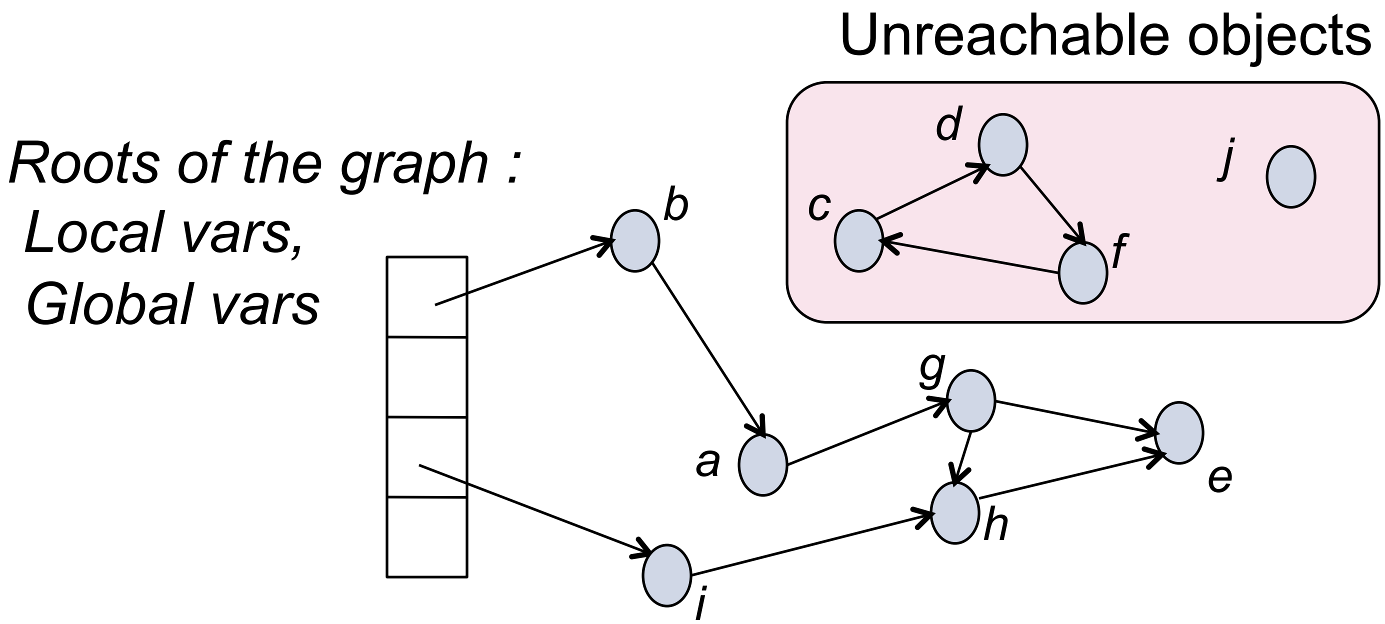
The problem: the GC does not scale

Page rank computation with Spark on 10^8 nodes



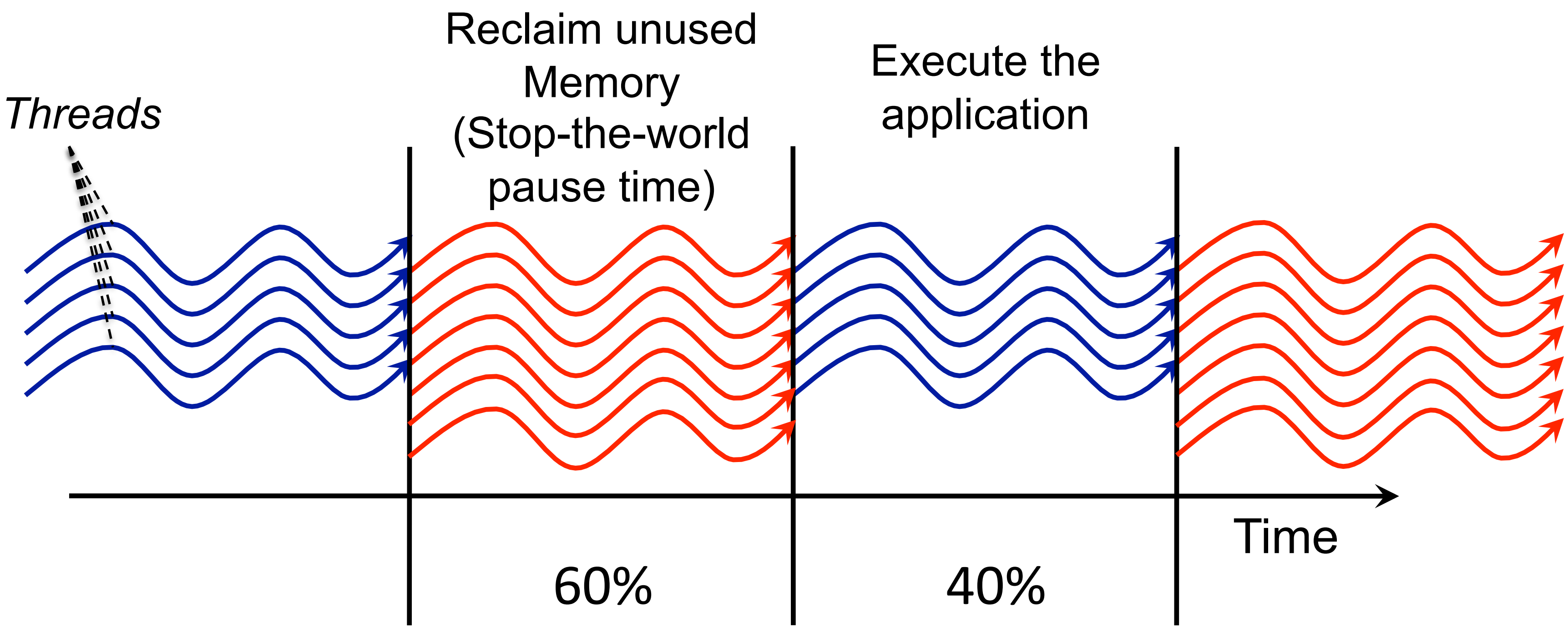
Background: Java garbage collector

- Automatically reclaims unused objects by considering the Java heap as a directed graph
  - Nodes are the Java objects
  - Edges are the Java reference
  - Traverse the graph in order to find live objects



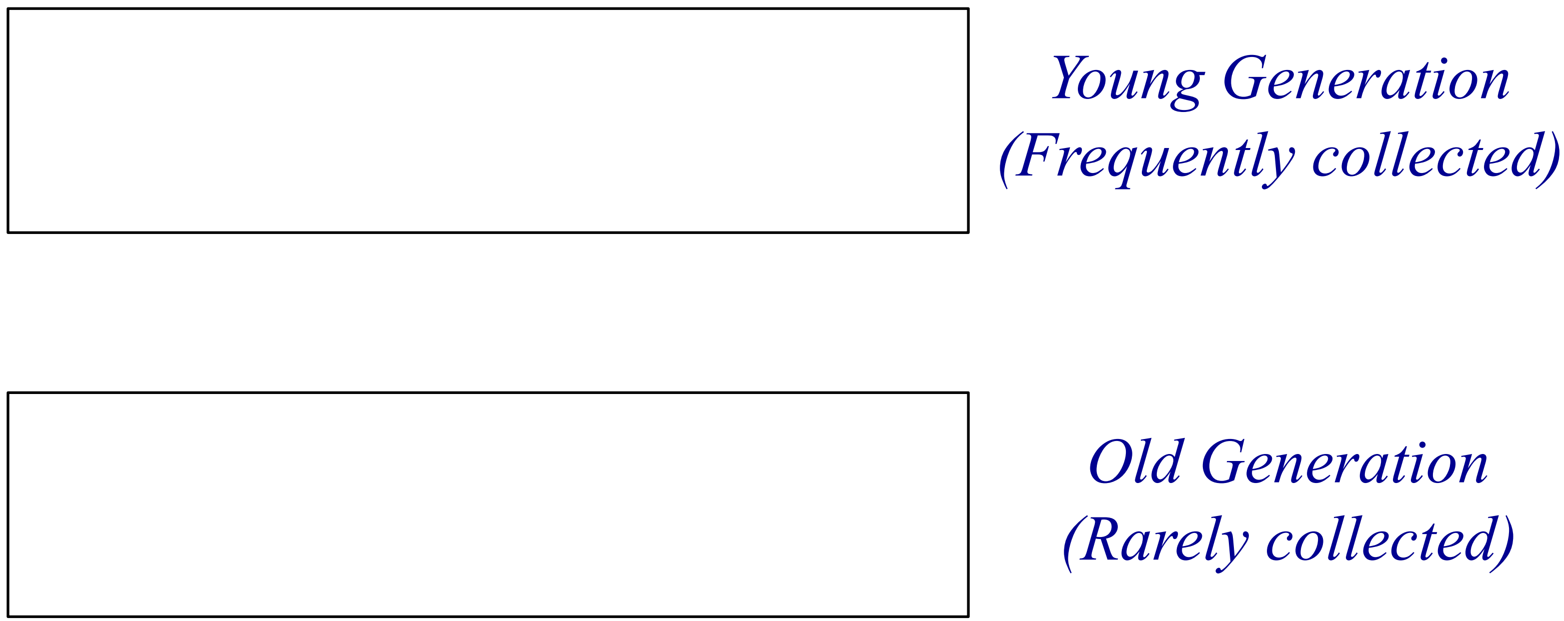
Background: Java garbage collector

- At each time, a Java process is either
  - Executing the application
  - Reclaiming unused memory (GC pause)

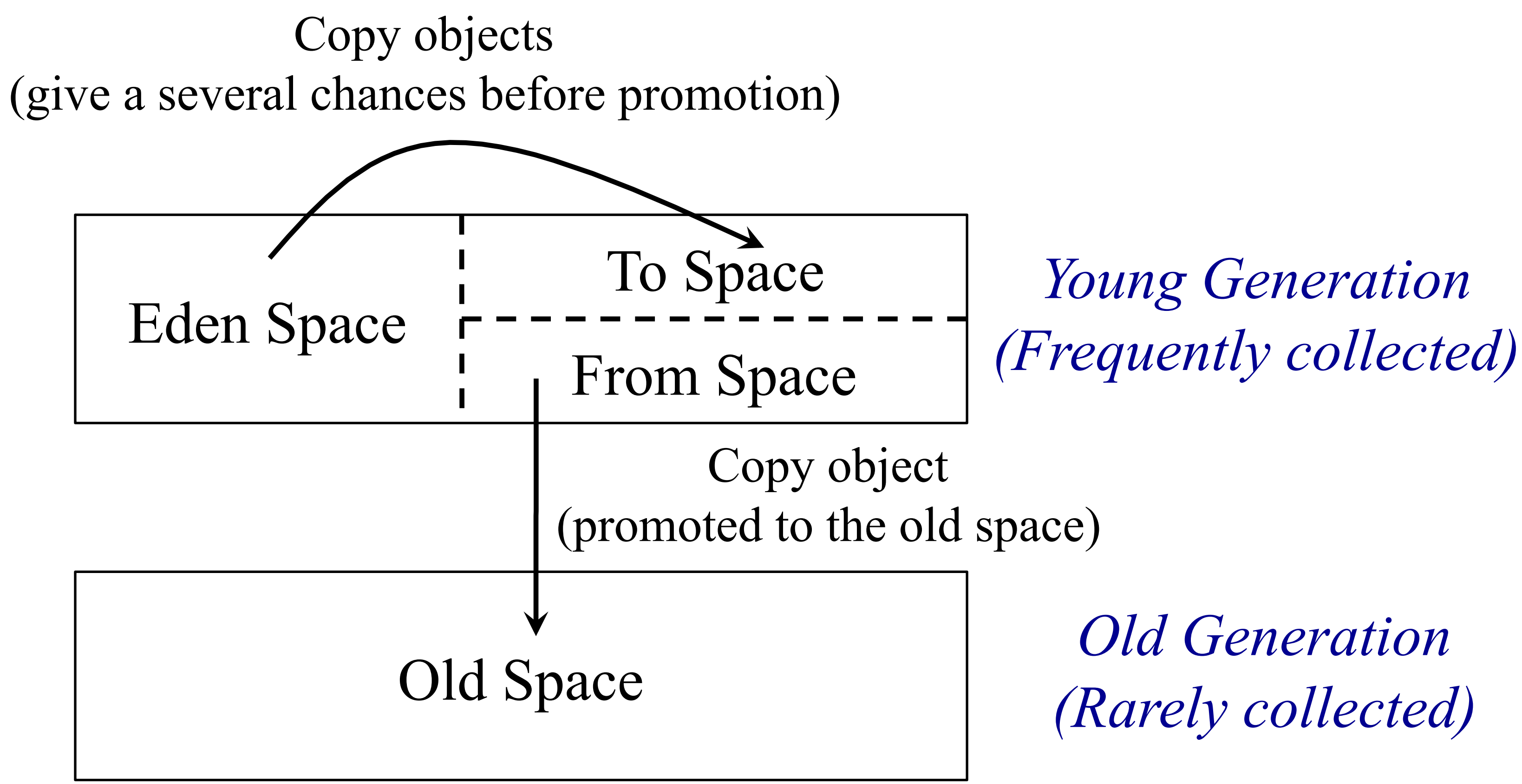


Baseline GC: Parallel Scavenge

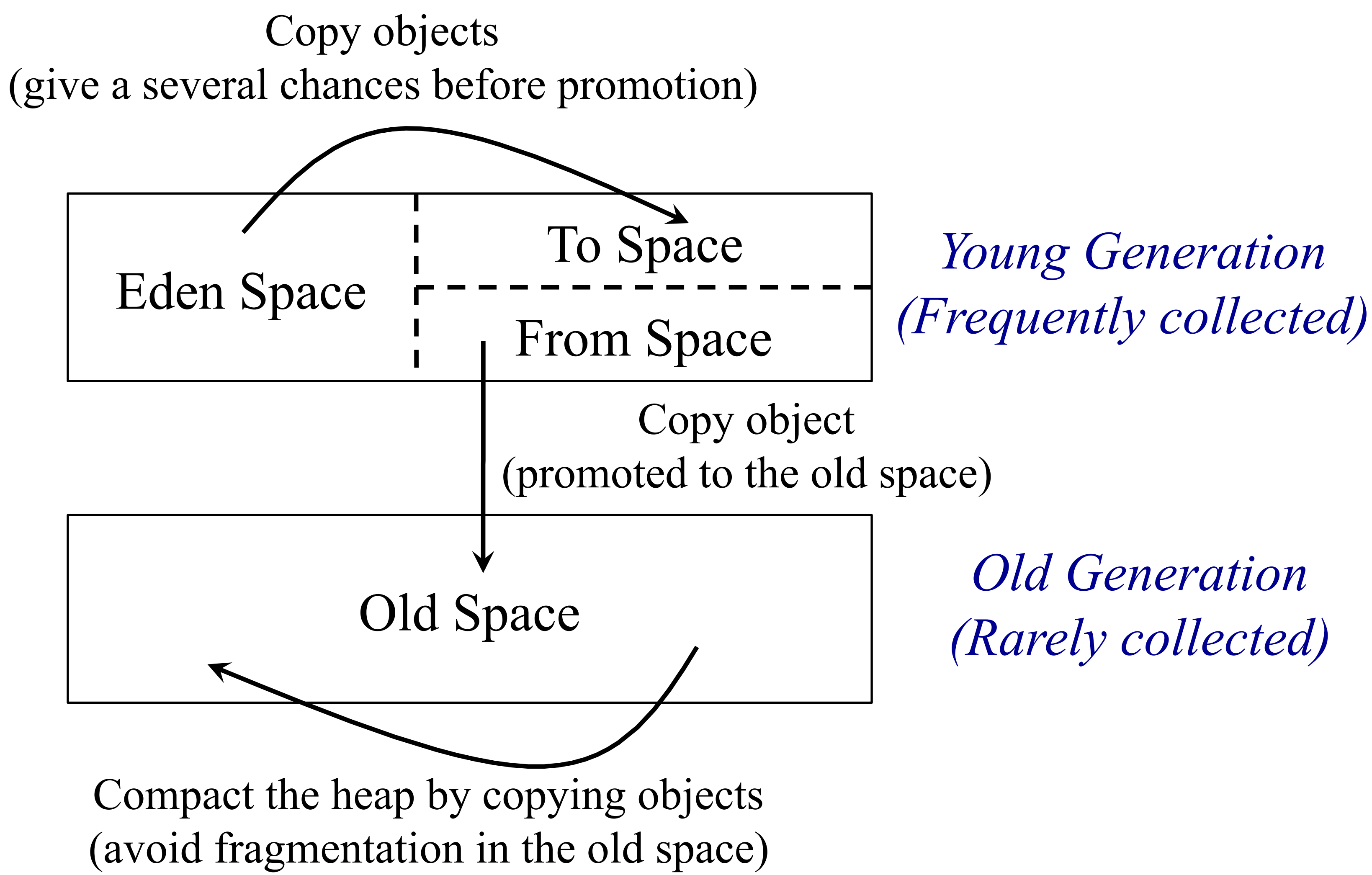
- Generationnal hypothesis: objects die young



Baseline GC: Parallel Scavenge

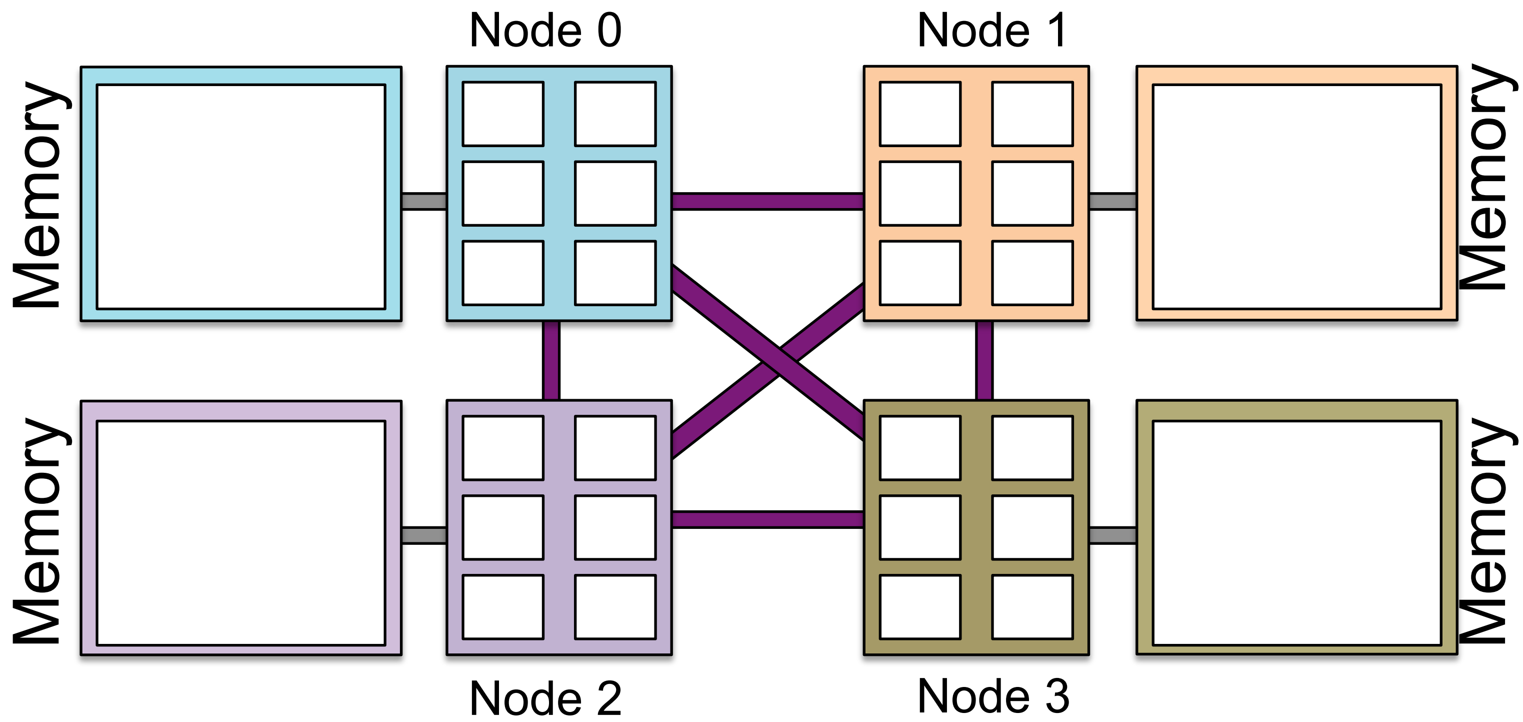


Baseline GC: Parallel Scavenge



Analysis of the GC bottleneck

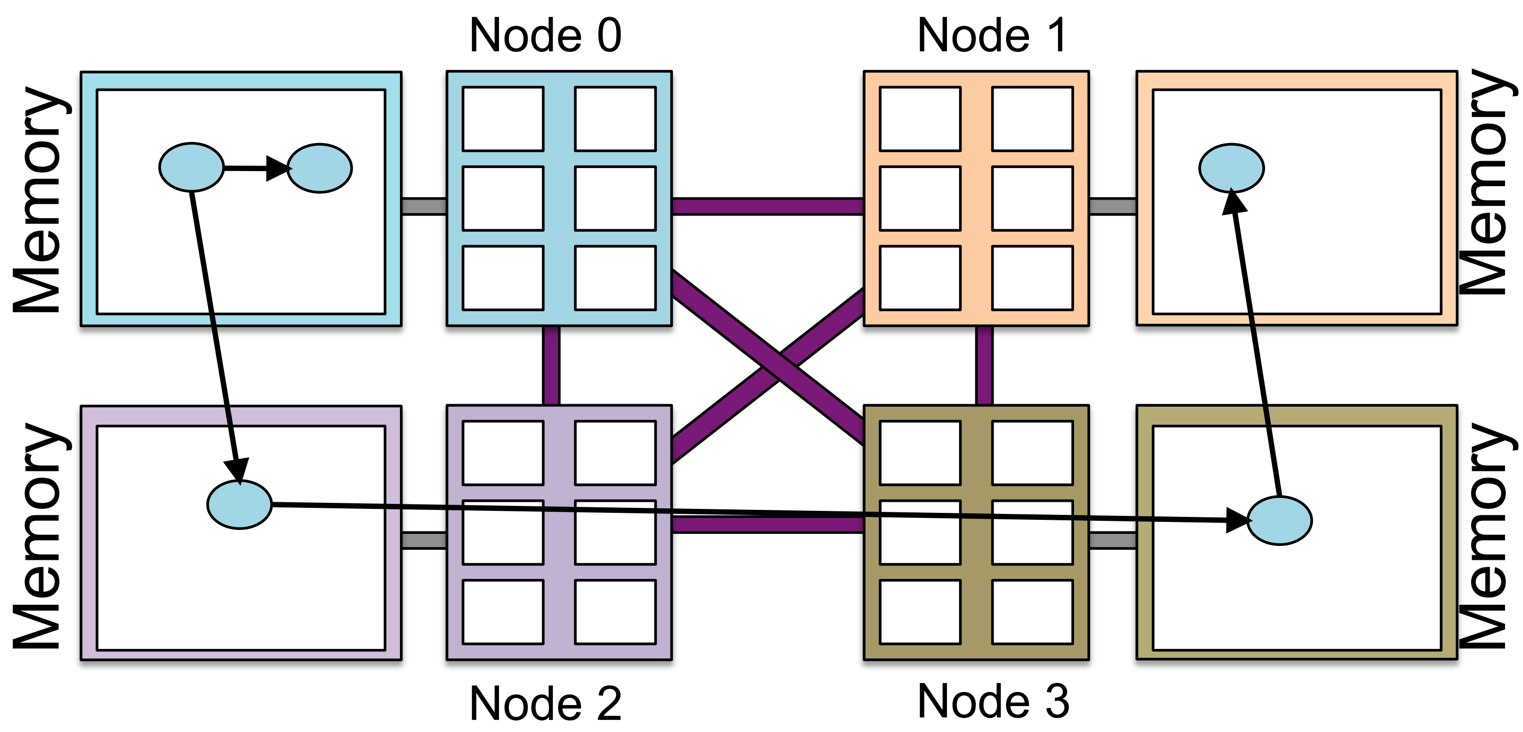
- A modern multicore is a small distributed system



Analysis of the GC bottleneck

- A modern multicore is a small distributed system

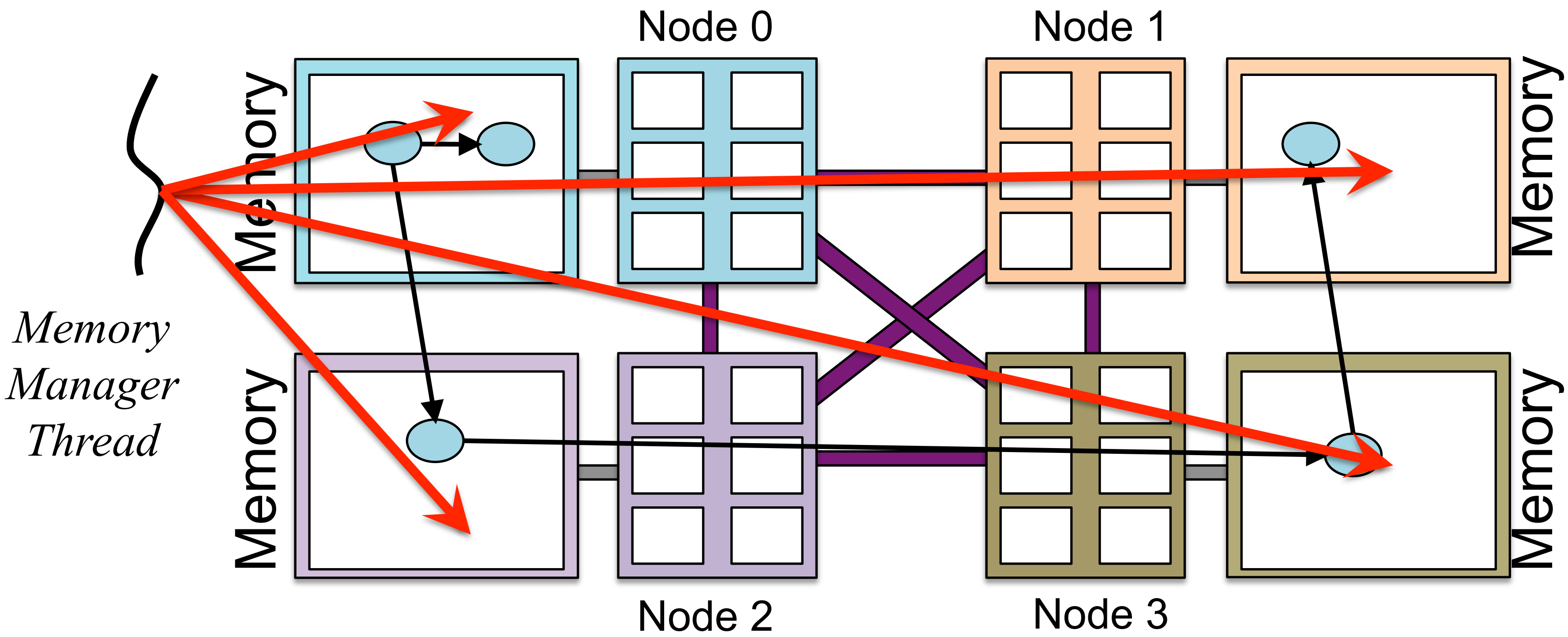
Application silently creates inter-node references



Analysis of the GC bottleneck

- A modern multicore is a small distributed system

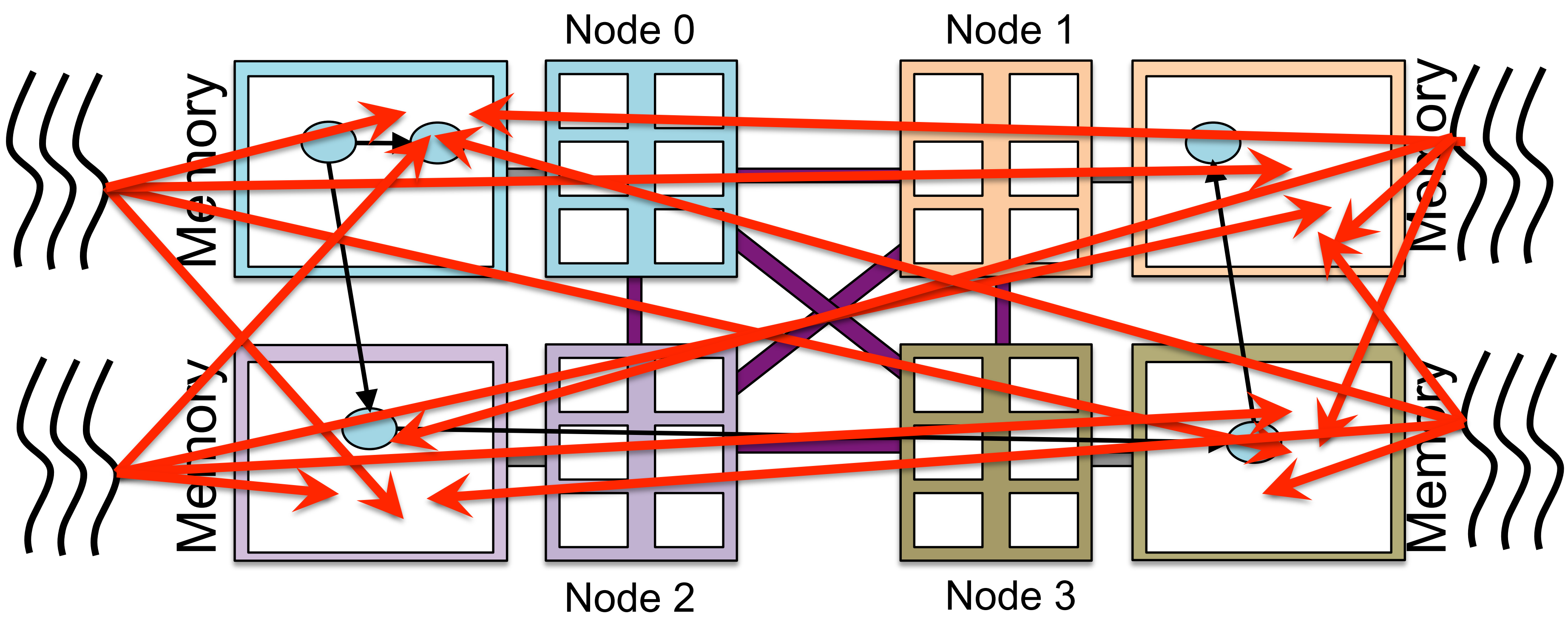
Threads of the memory manager perform random accesses



Analysis of the GC bottleneck

- A modern multicore is a small distributed system

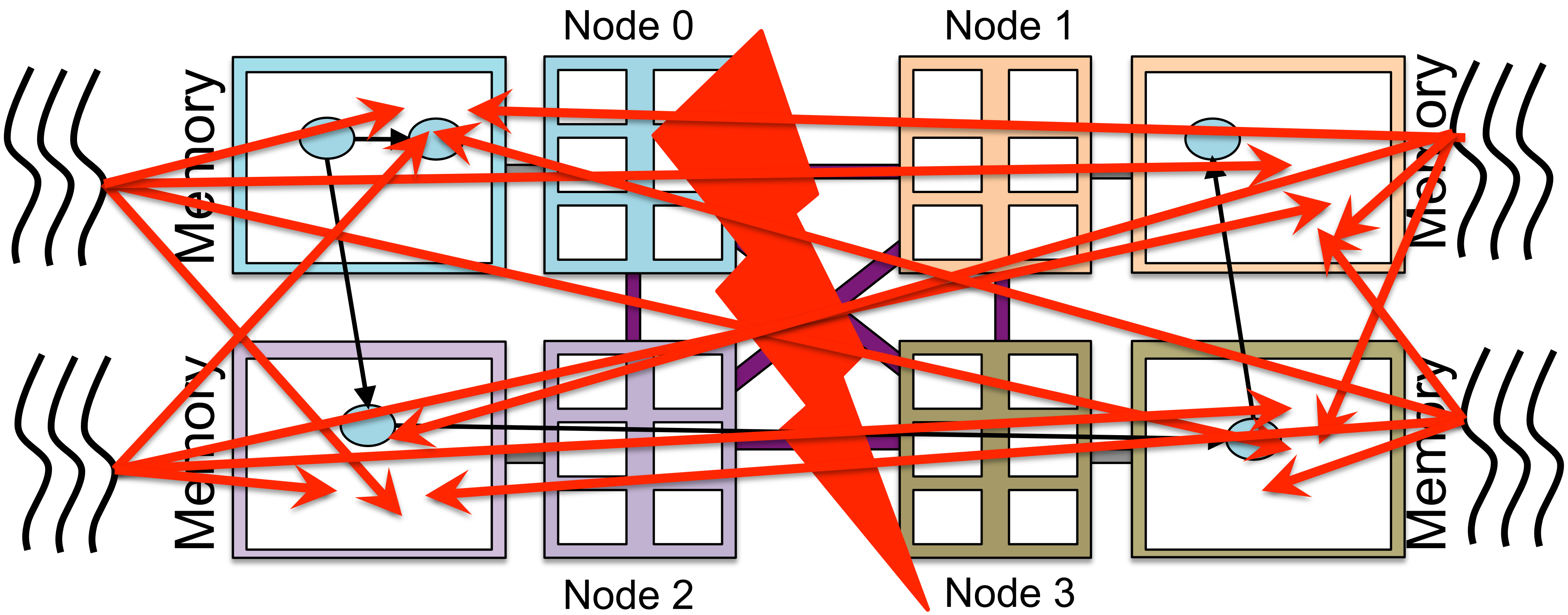
The memory manager uses many threads



Analysis of the GC bottleneck

- A modern multicore is a small distributed system

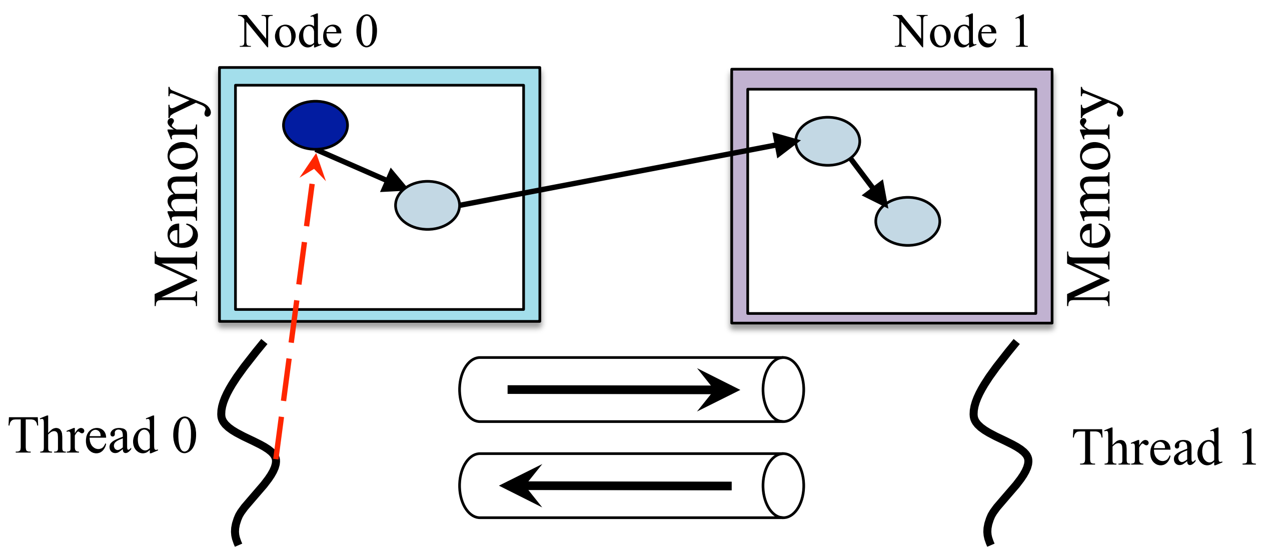
And eventually the network between the nodes saturates
⇒ drastically slows down memory access time



Solution: a new memory manager

NUMAGiC: a memory manager with a distributed design

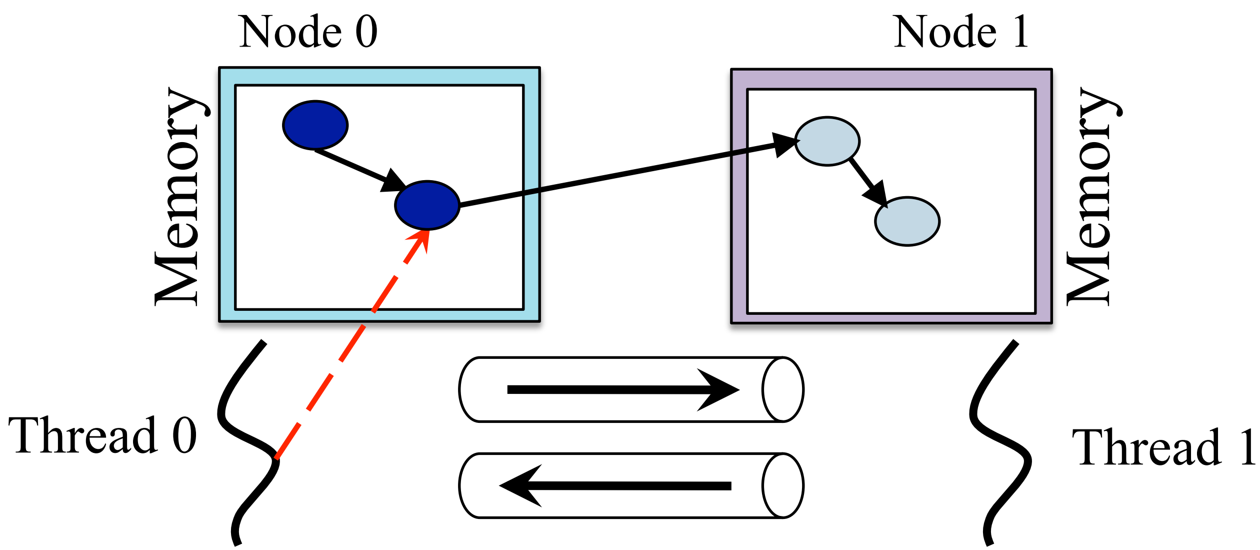
Idea: instead of accessing memory on a remote node, a thread notifies another thread on the remote node



Solution: a new memory manager

NUMAGiC: a memory manager with a distributed design

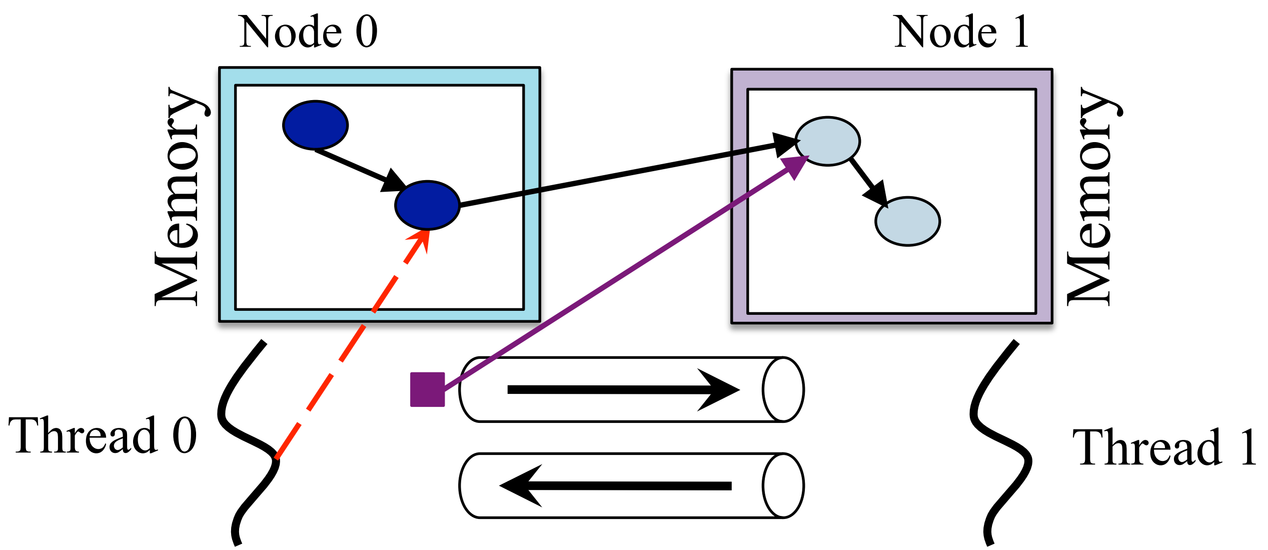
Idea: instead of accessing memory on a remote node,
a thread notifies another thread on the remote node



Solution: a new memory manager

NUMAGiC: a memory manager with a distributed design

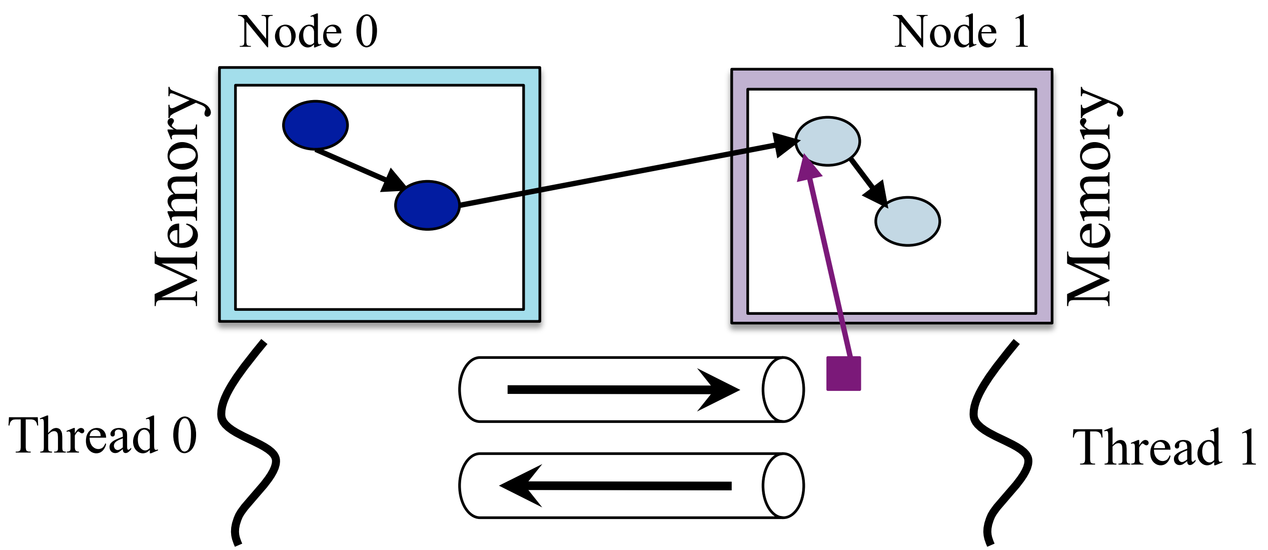
Idea: instead of accessing memory on a remote node,
a thread notifies another thread on the remote node



Solution: a new memory manager

NUMAGiC: a memory manager with a distributed design

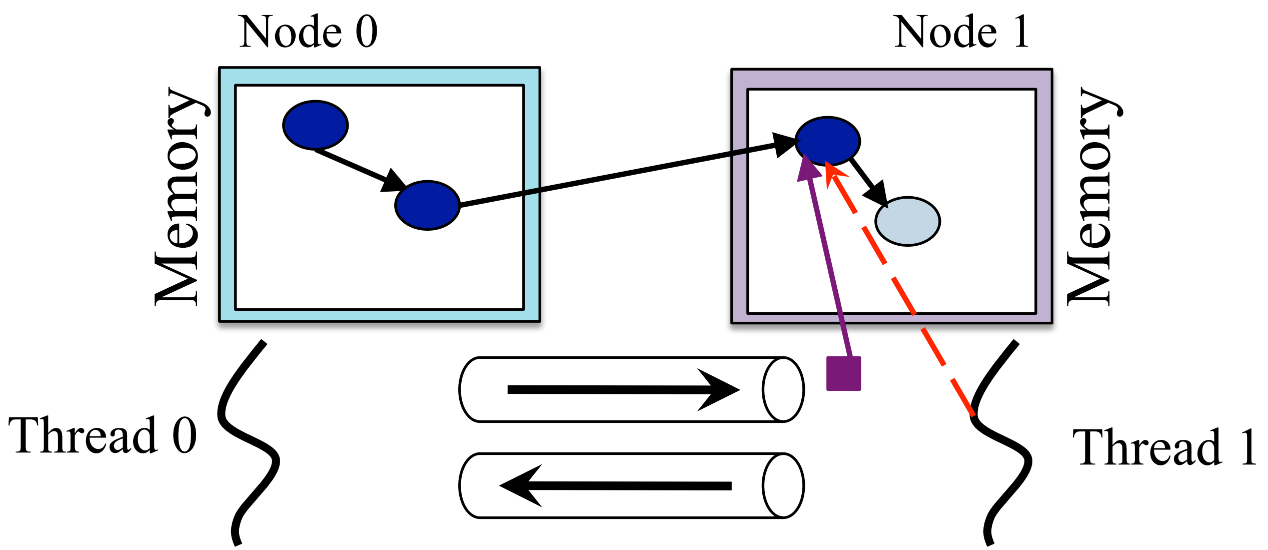
Idea: instead of accessing memory on a remote node,
a thread notifies another thread on the remote node



Solution: a new memory manager

NUMAGiC: a memory manager with a distributed design

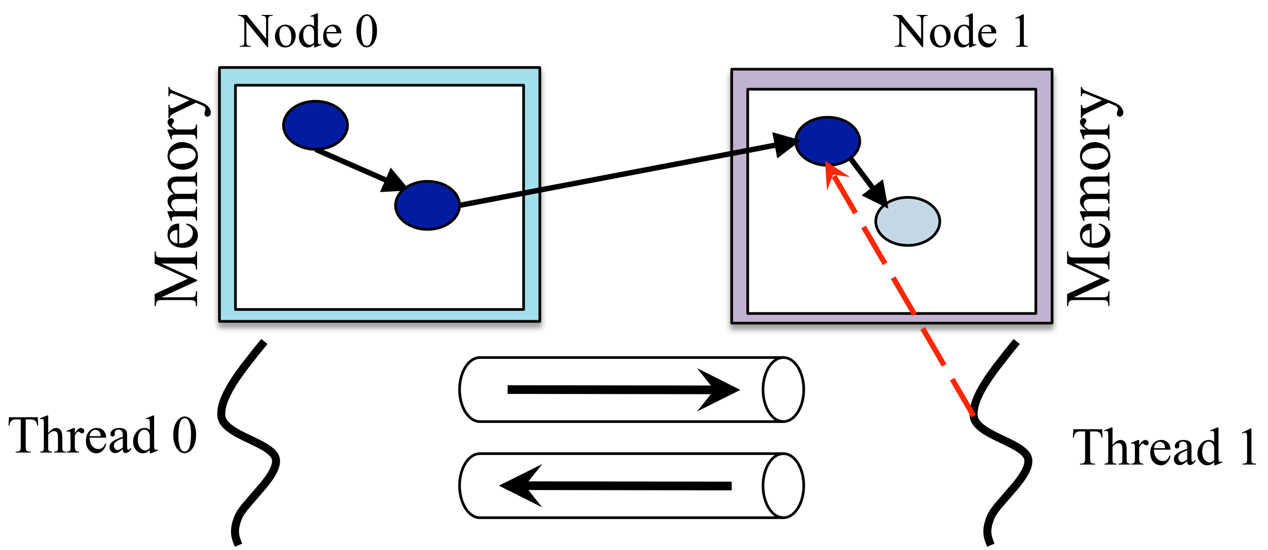
Idea: instead of accessing memory on a remote node,
a thread notifies another thread on the remote node



Solution: a new memory manager

NUMAGiC: a memory manager with a distributed design

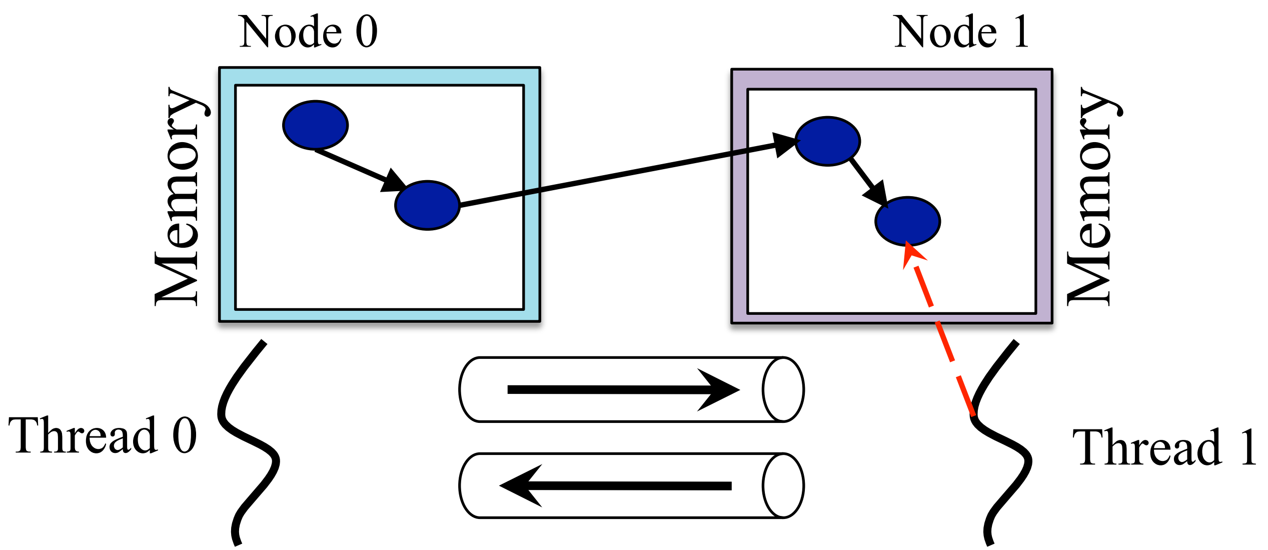
Idea: instead of accessing memory on a remote node, a thread notifies another thread on the remote node



Solution: a new memory manager

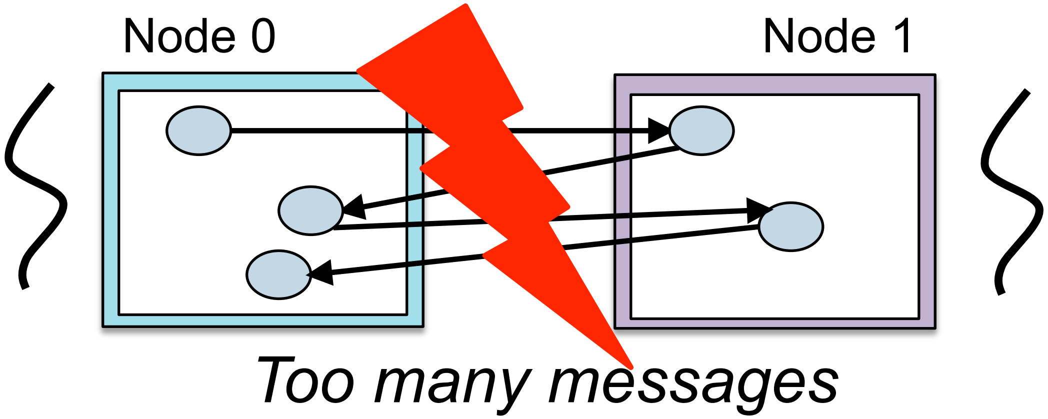
NUMAGiC: a memory manager with a distributed design

Idea: instead of accessing memory on a remote node,
a thread notifies another thread on the remote node



NUMA-friendly placement heuristics

- Problem: 1 message is more costly than 1 remote access
  => Inter-node references must be minimized



- Observation: a thread mostly connects objects it has allocated
- Heuristics: let objects allocated by a thread on its node
  ⇒ side effect: improve memory access locality for the application

NUMA-friendly placement heuristics

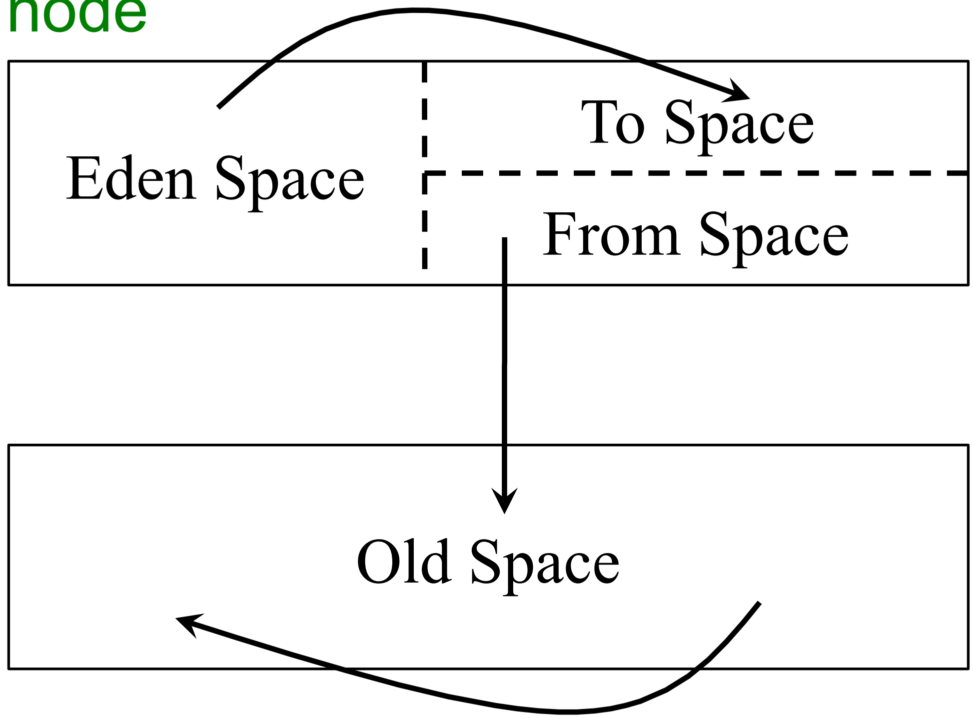
But “Let objects allocated by a thread on its node” raises a problem

- If only one node allocate the memory,
- All the GC threads accesses the allocation node

⇒ the node collapse

NUMA-friendly placement heuristics

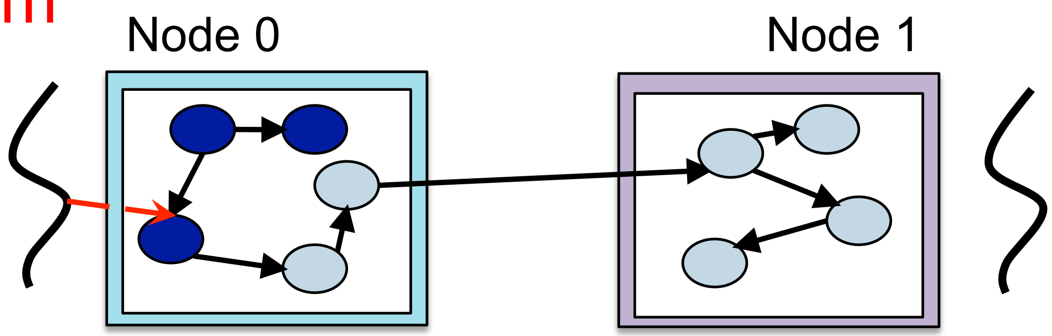
- Eden :
  Allocate in the current NUMA node
- Eden → To Space
  Local copy
  Steal from remote
- From Space → Old Space
  Local copy
  Steal from remote
- Old Space → Old Space
  Local copy only



Local copy ⇒ prevents remote references
Steal from remote ⇒ balance memory on all the nodes
(important in order to avoid overloaded nodes)

Adaptive algorithm

- Problem: strictly avoiding remote access degrades parallelism



Node 1 idles while node 0 collects its memory

- Happen often because we minimize inter-node references!

- Solution: adaptive algorithm
  - Local mode: send messages when not idling
  - Thief mode: steal and access remote objects when idling

Experiments – hardware setting

- Amd48 : AMD Magny Cour with
  - 8 nodes
  - 48 threads
  - 256 GB of RAM

- Intel80 : Xeon E7-2860 with
  - 4 nodes
  - 160 threads
  - 512 GB of RAM

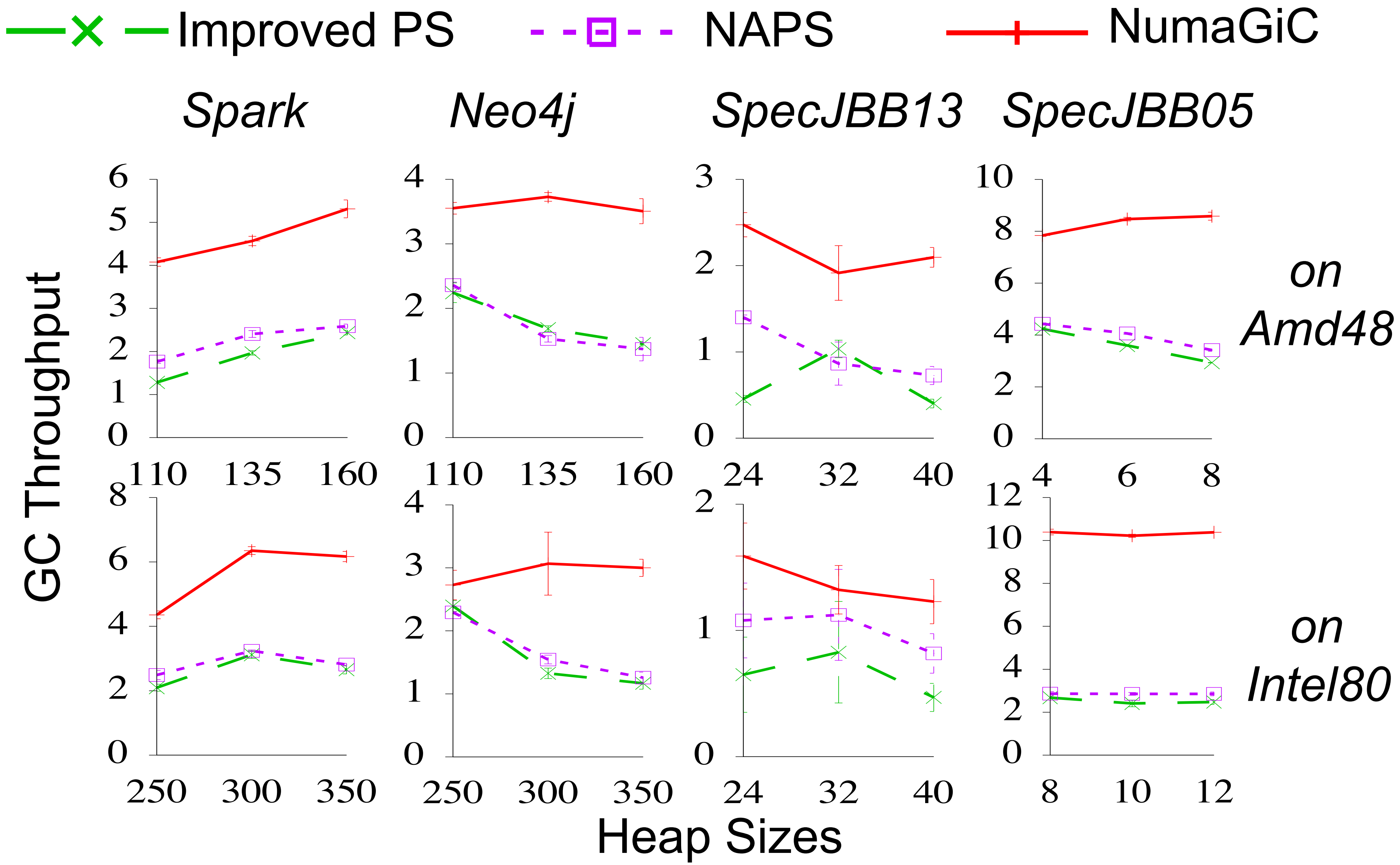
Experiments – software setting

Name	Description	Heap Size	
		Amd48	Intel80
Spark	In-memory map-reduce (page rank computation)	110 to 160GB	250 to 350GB
Neo4j	Object graph database (page rank computation)	110 to 160GB	250 to 350GB
SPECjbb2013	Business-logic server	24 to 40GB	24 to 40GB
SPECjbb2005	Business-logic server	4 to 8GB	4 to 12GB

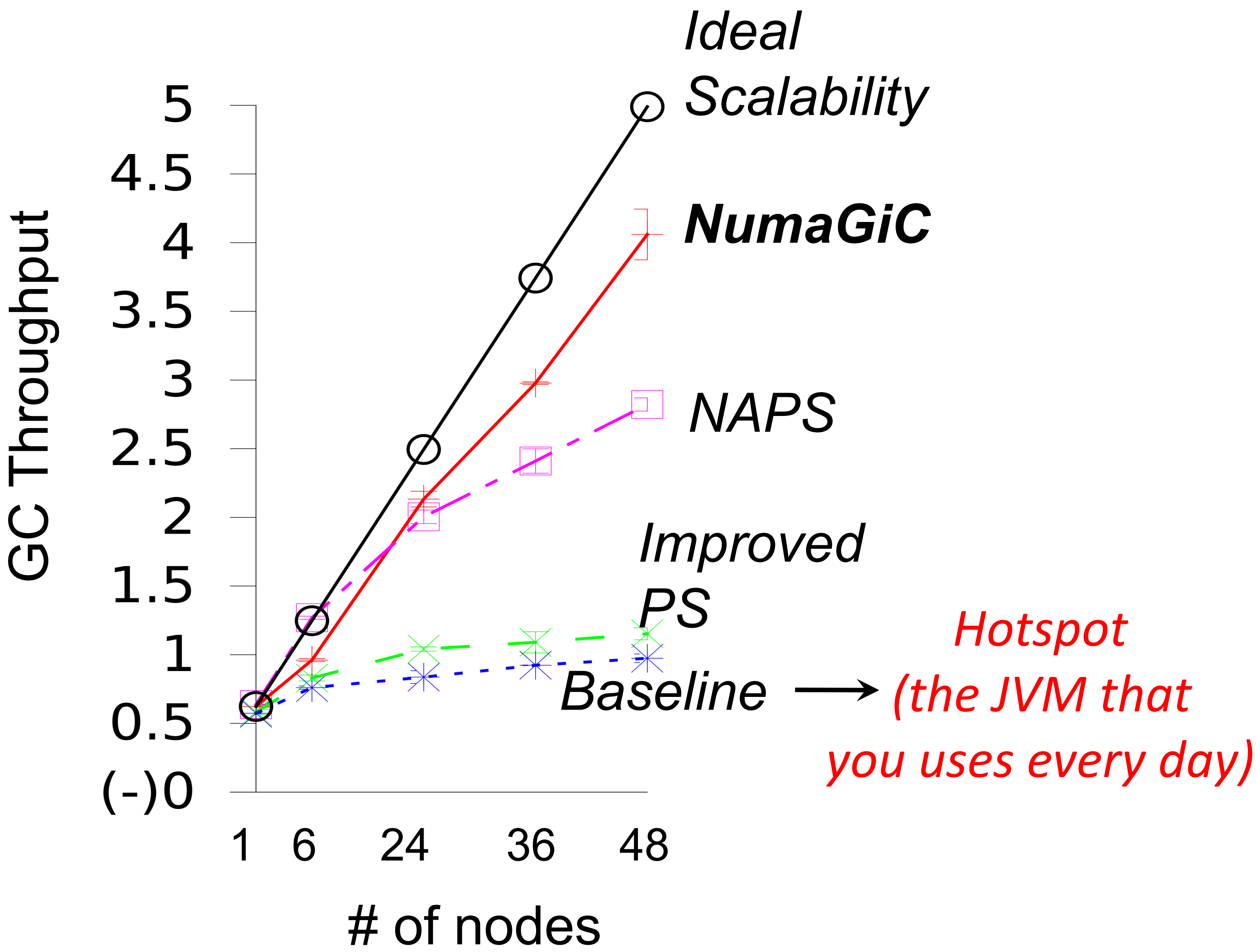
1 billions node
from the Friendster dataset

The 1.8 billions node
of the Friendster dataset

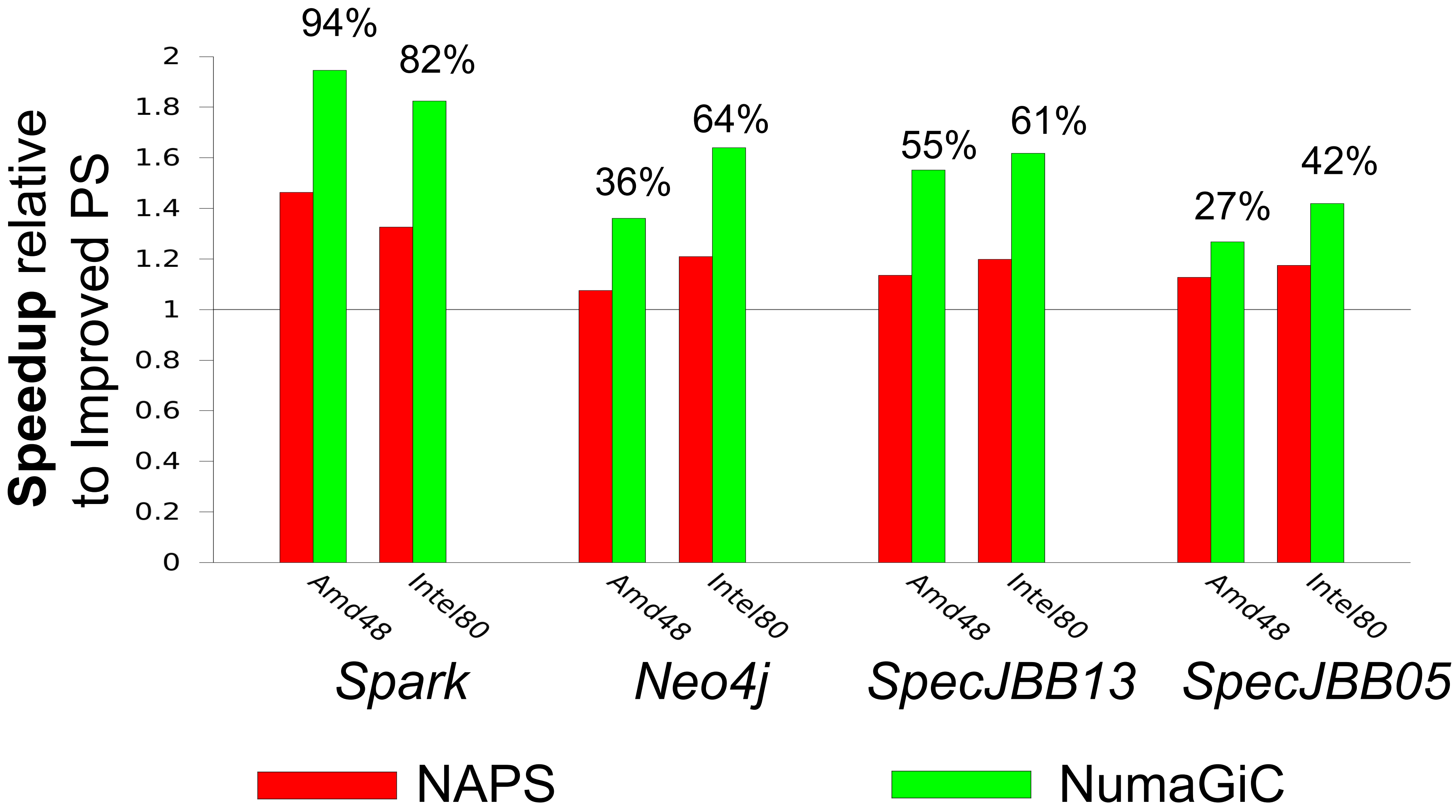
GC Throughput from x2 to x5



NUMAGiC scalability



Improvement for the applications



Heap size of 160GB on Amd48 and 350GB on Intel80

To take away

- Performance of big-data analytics relies on GC performance
- Memory access locality has huge effect on GC performance
- Enforcing locality can be detrimental for parallelism in GCs
- No big difference between Intel and Amd NUMA architectures

To take away

- Performance of big-data analytics relies on GC performance
- Memory access locality has huge effect on GC performance
- Enforcing locality can be detrimental for parallelism in GCs
- No big difference between Intel and Amd NUMA architectures

Thank You ☺