UCI University of California, Irvine

Yak: A High-Performance Big-Data-Friendly Garbage Collector

OSDI '16

Khanh Nguyen, Lu Fang, Guoqing Xu, Brian Demsky, Sanazsadat Alamian
University of California, Irvine

Shan Lu
University of Chicago

Onur Mutlu
ETH Zürich

BIG DATA

cloudera IMPALA

APACHE GIRAPH

hadoop MapReduce

Apache Hama

BIG DATA

HIVE

Hortonworks

Naiad

mahout

FLUME

Spark

AsterixDB

Dryad

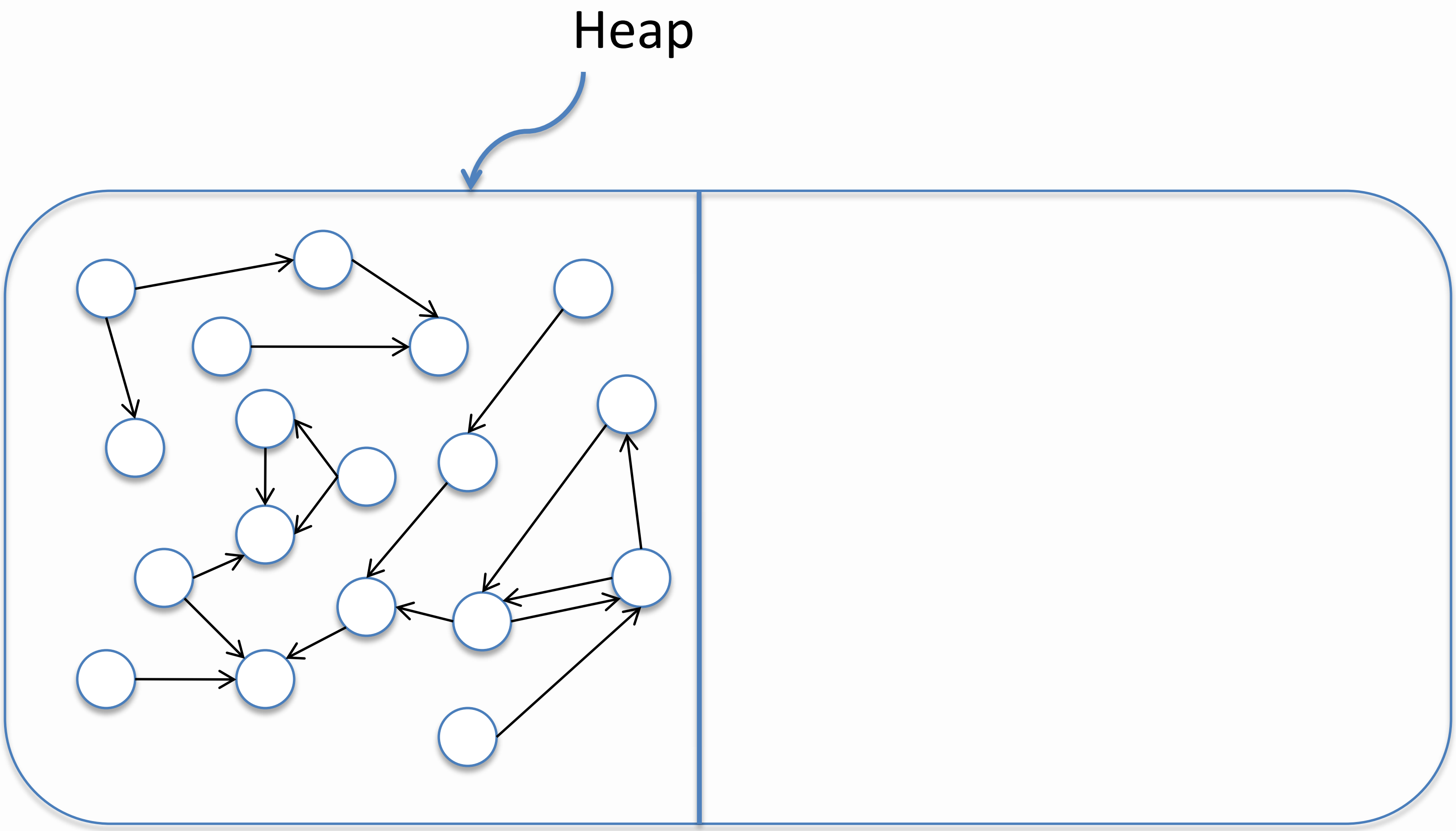
APACHE STORM™

Java™

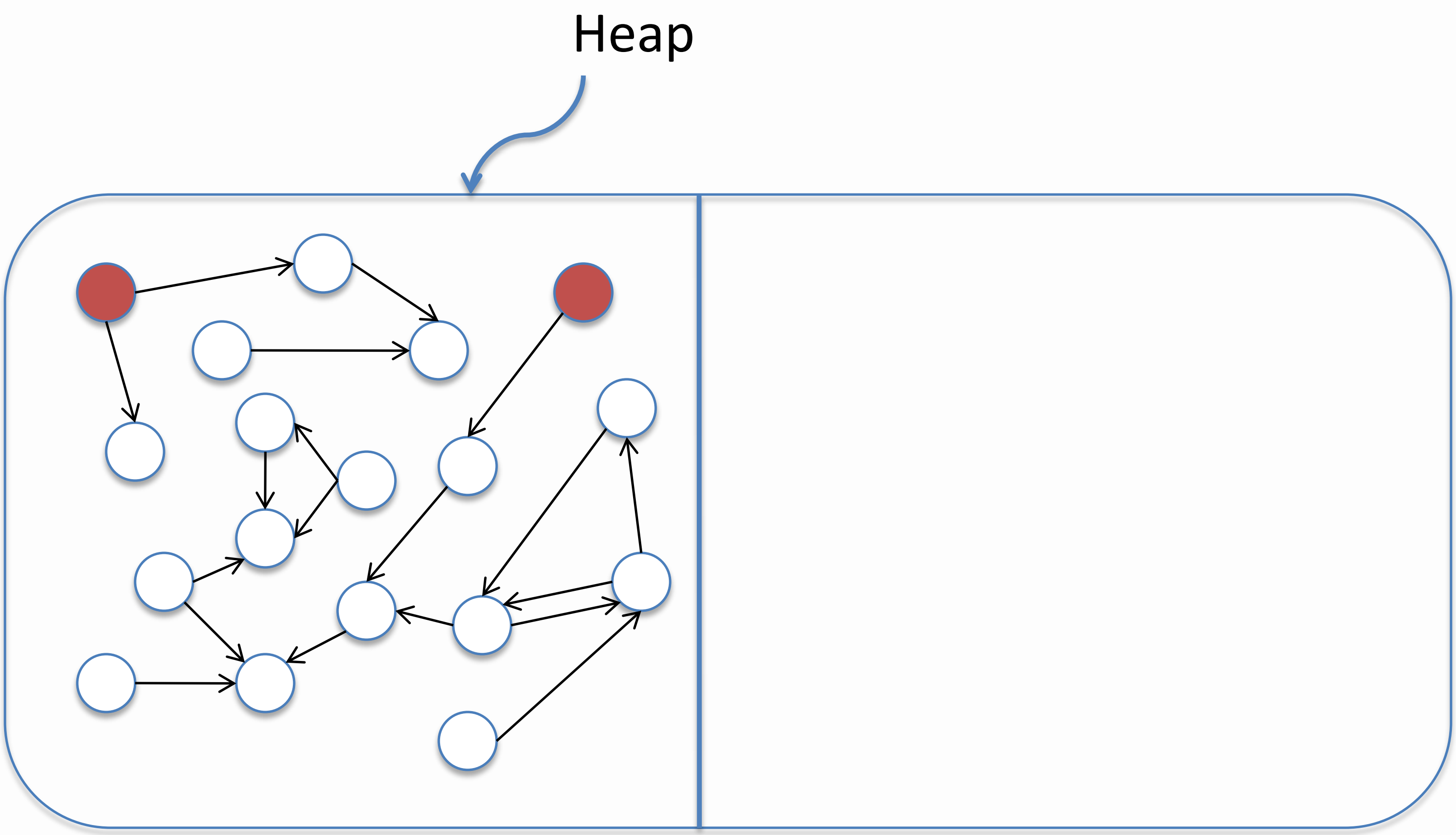
C#

Scala

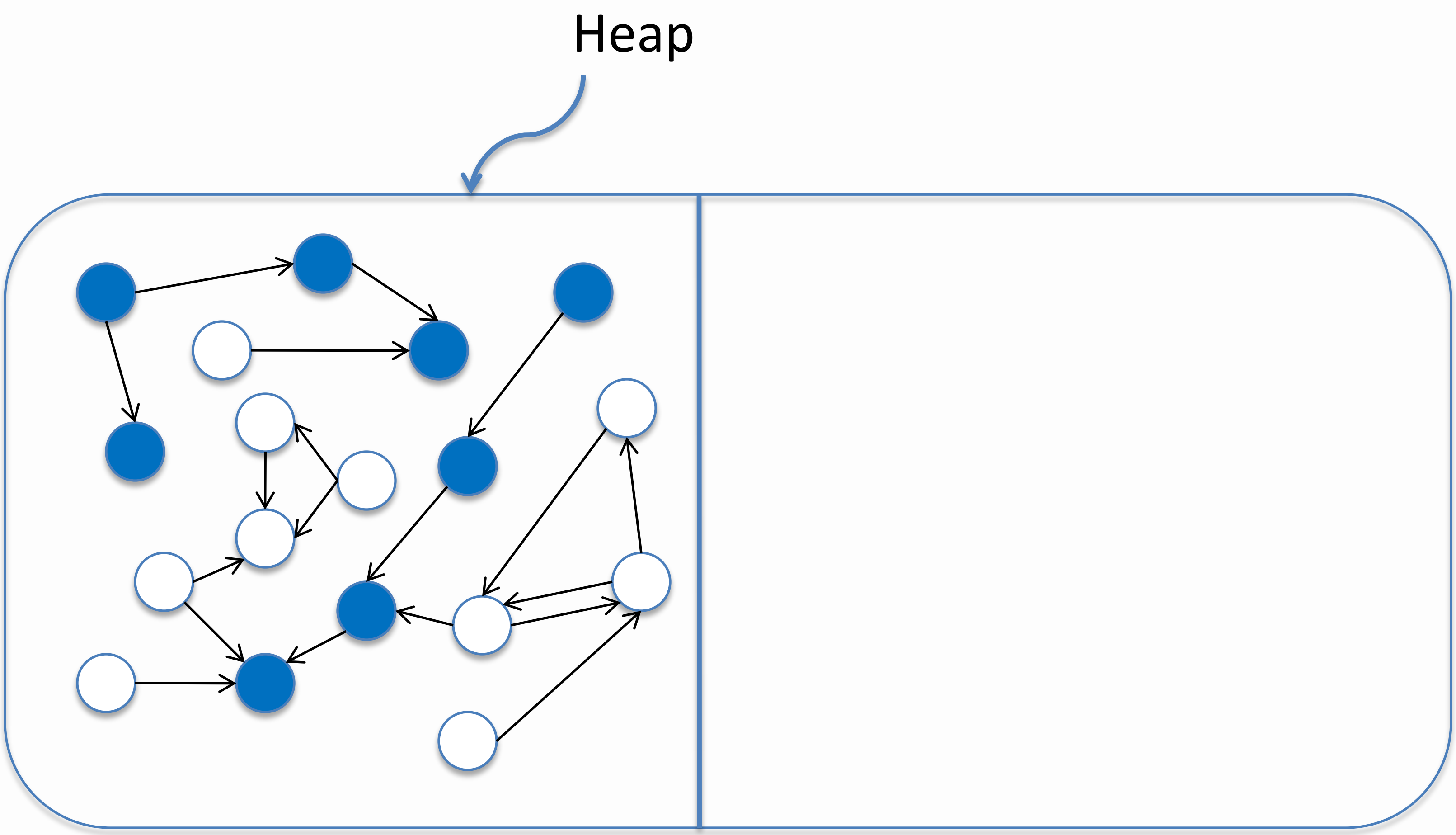
Background: GC



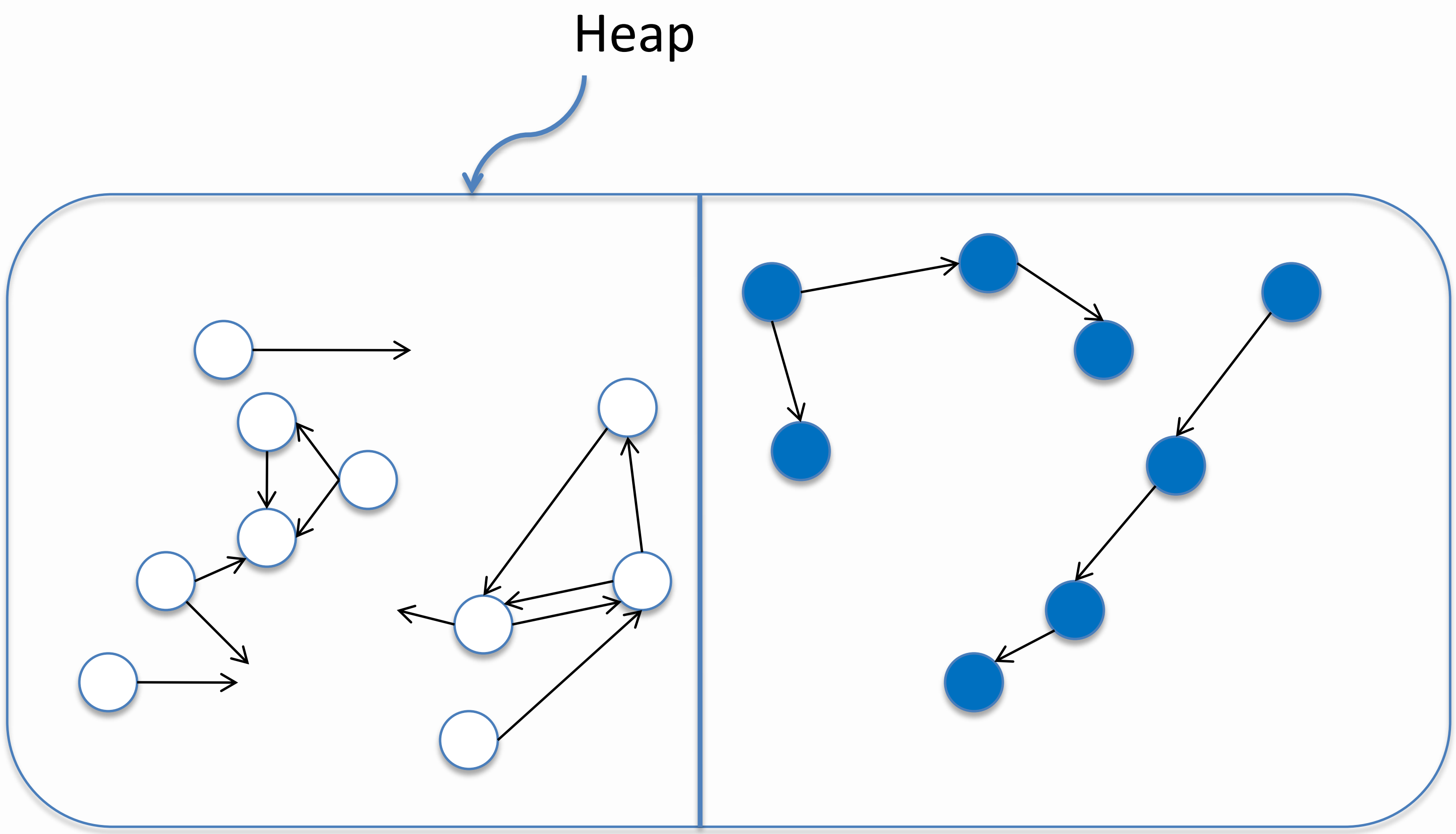
Background: GC



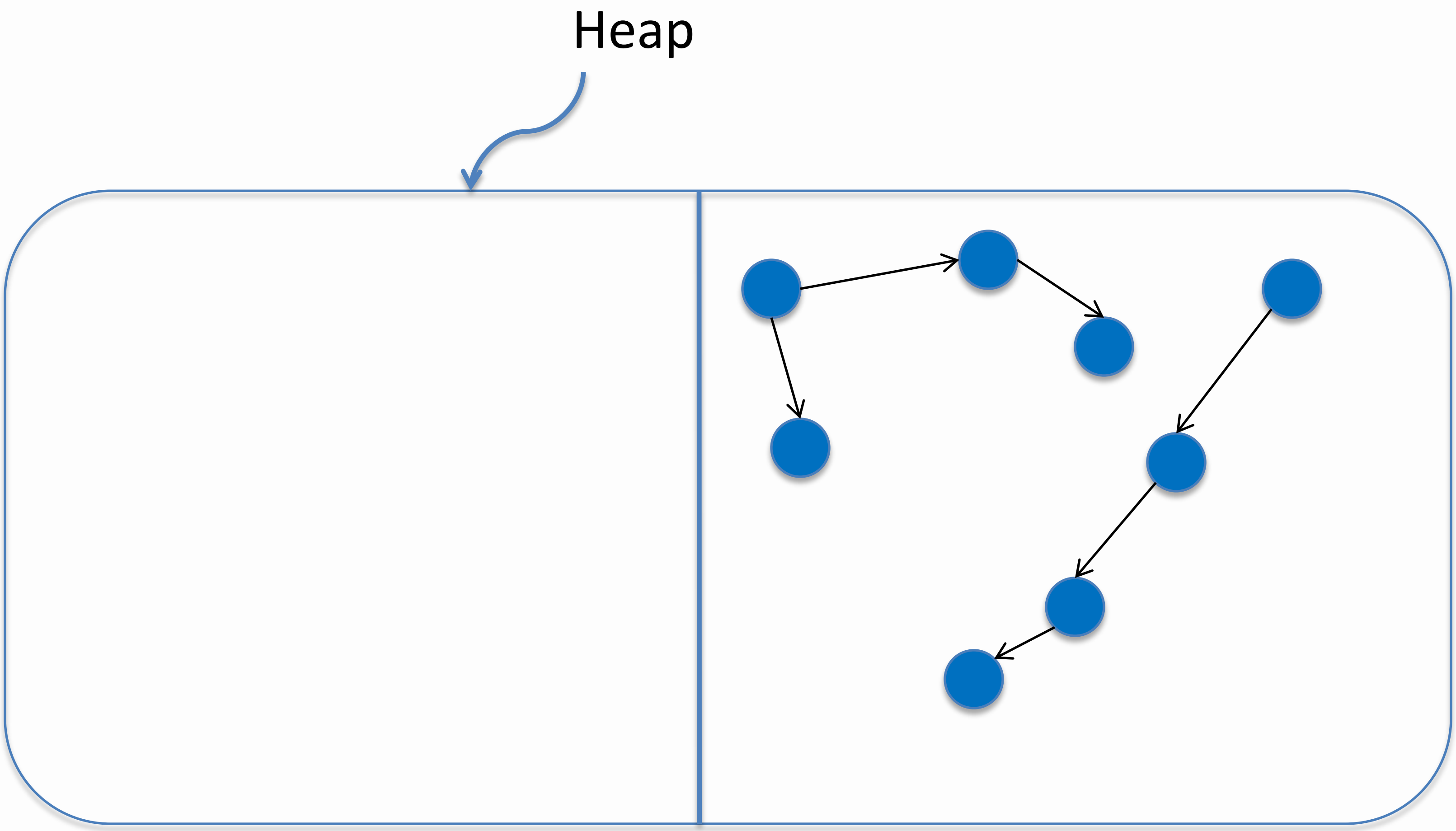
Background: GC



Background: GC



Background: GC



Java™

C#

Scala

Java™ C# Scala

Scalability

JVM crashes

due to OutOfMemory

error at early stage

[Fang et al., SOSP'15]

Scalability

JVM crashes due to OutOfMemory error at early stage

[Fang et al., SOSP'15]

Management cost

GC time accounts for up to **50%** of the execution time

[Bu et al., ISMM'13]

Scalability

JVM crashes due to OutOfMemory error at early stage

[Fang et al., SOSP'15]

Management cost

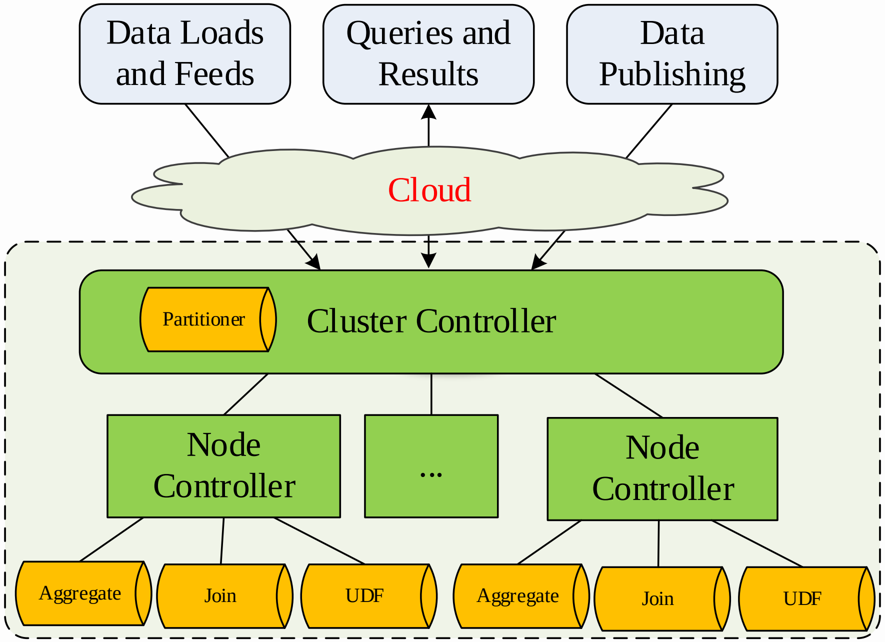
GC time accounts for up to **50%** of the execution time

[Bu et al., ISMM'13]

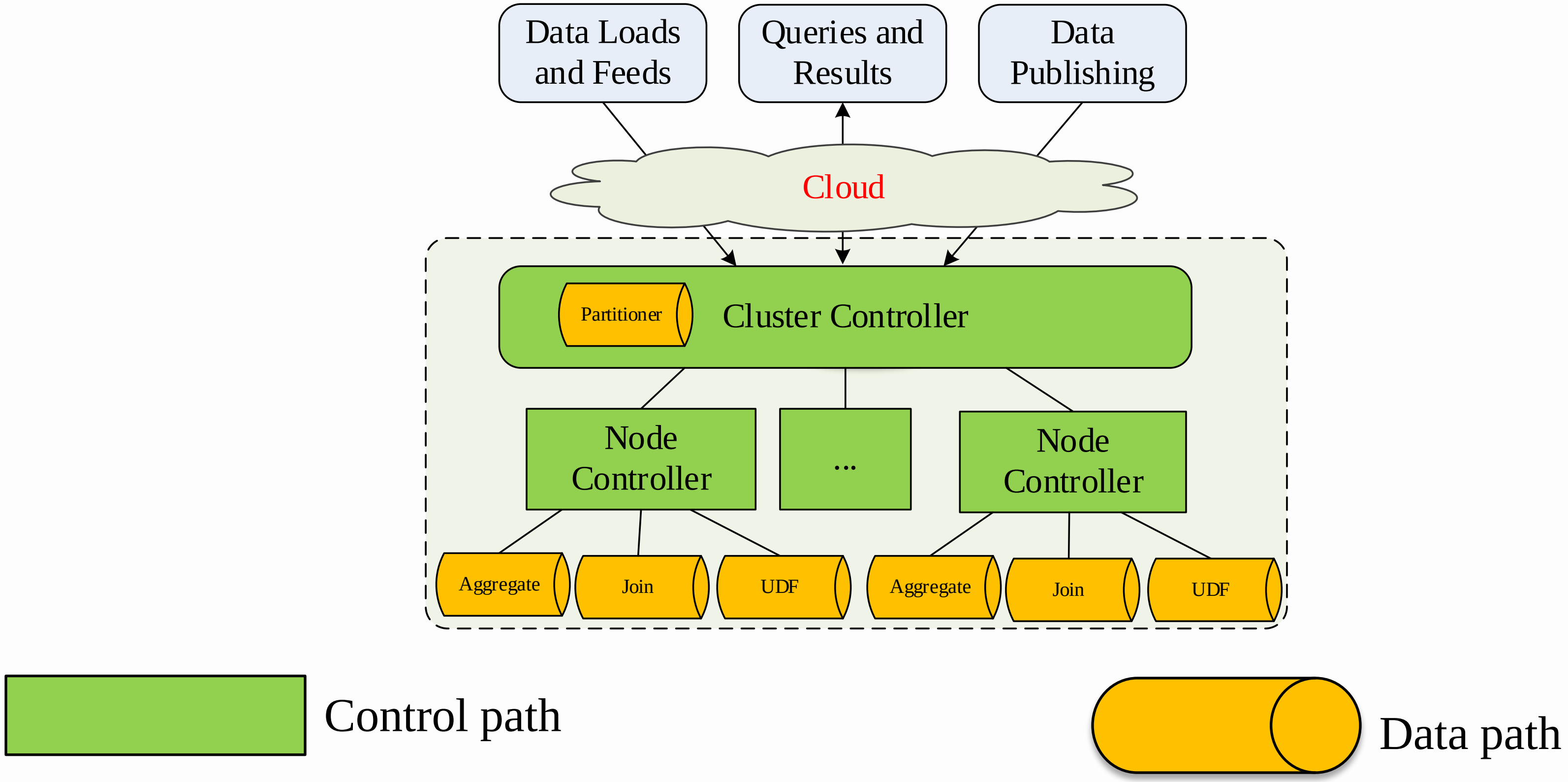
High cost of the managed runtime is a fundamental problem!

Two Paths, Two Hypotheses

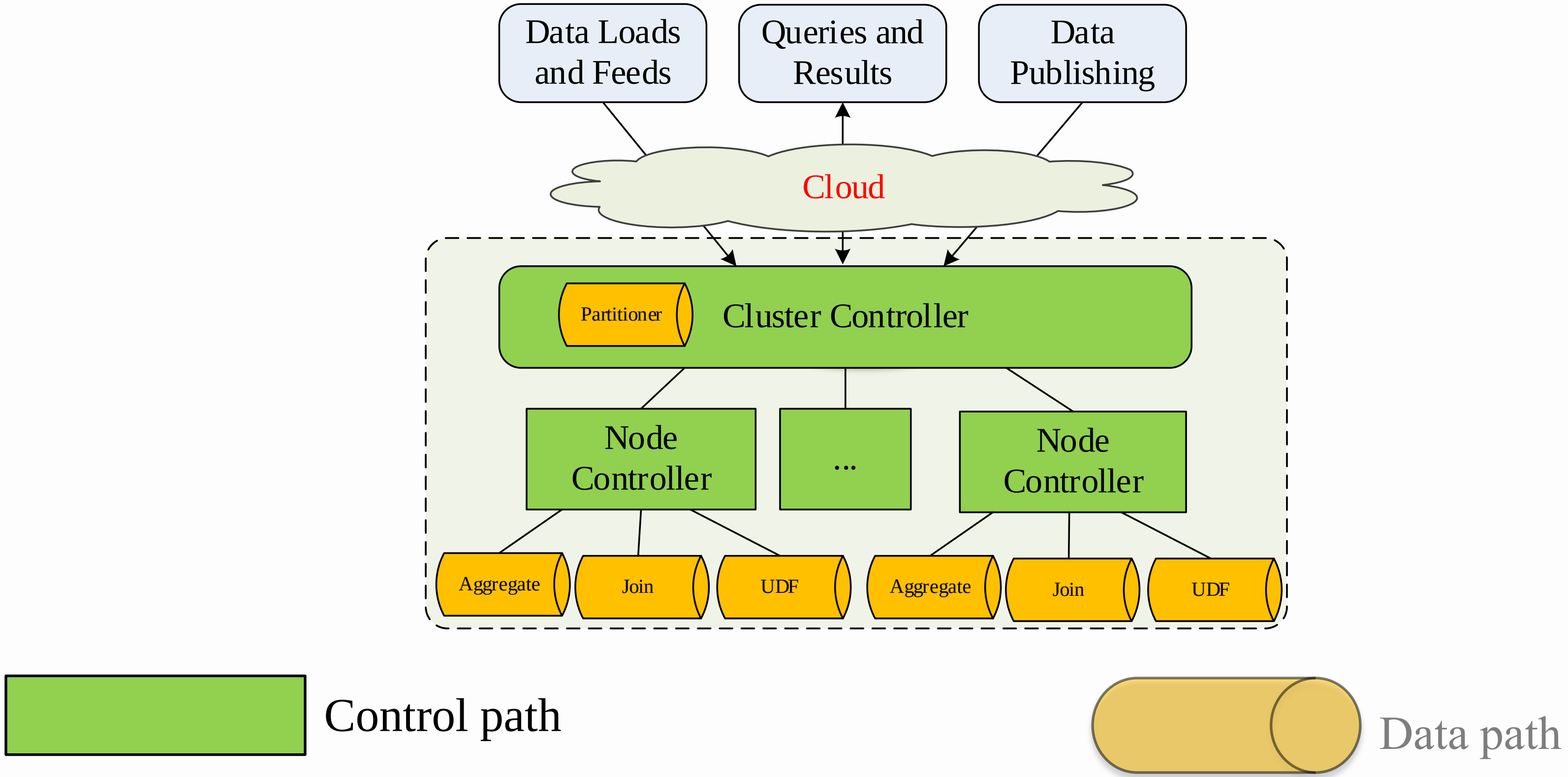
Two Paths, Two Hypotheses



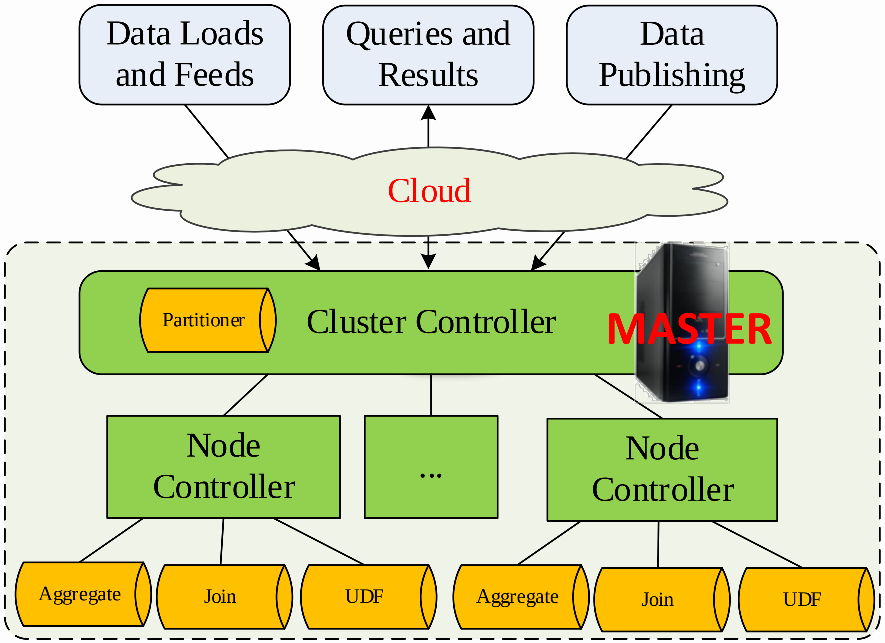
Two Paths, Two Hypotheses



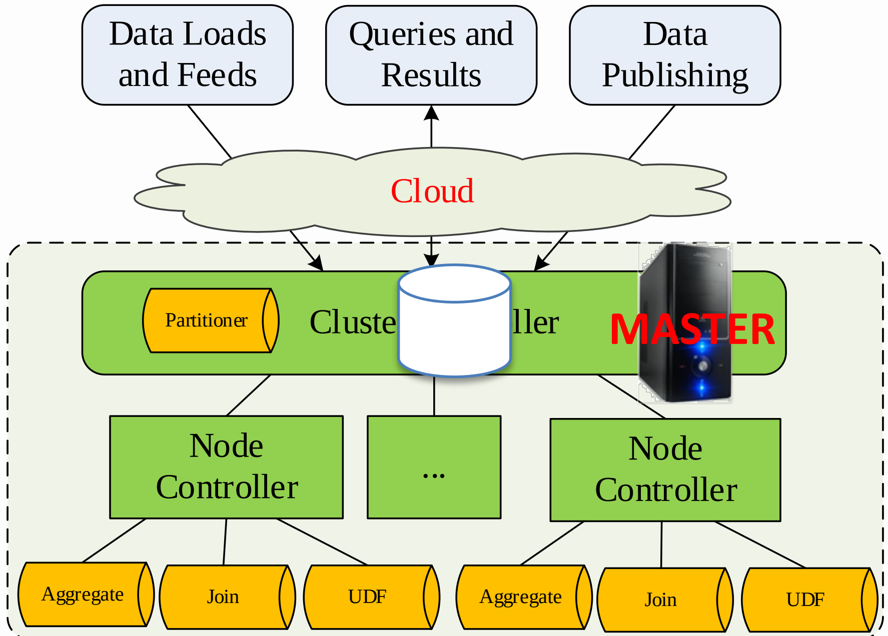
Two Paths, Two Hypotheses



Two Paths, Two Hypotheses



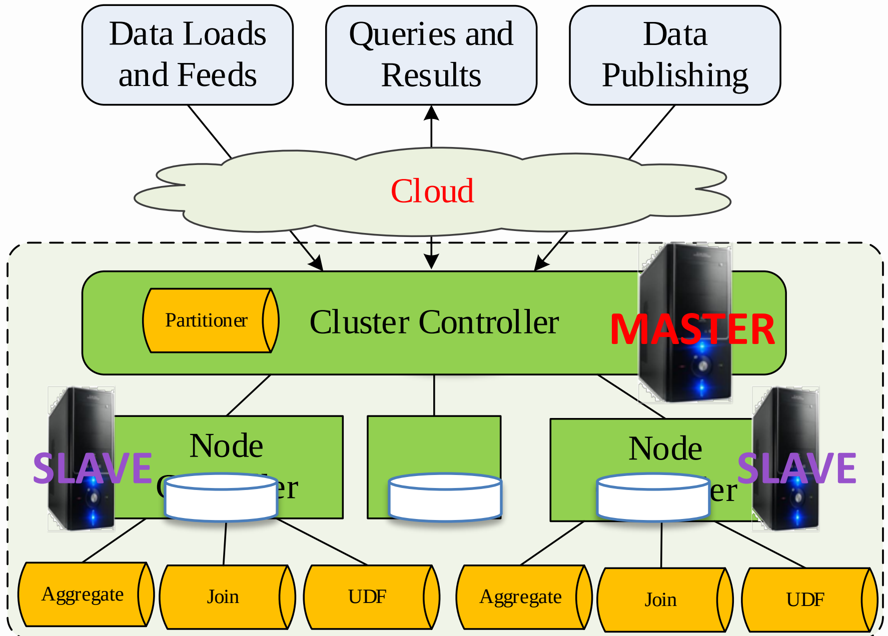
Two Paths, Two Hypotheses



Control path

Data path

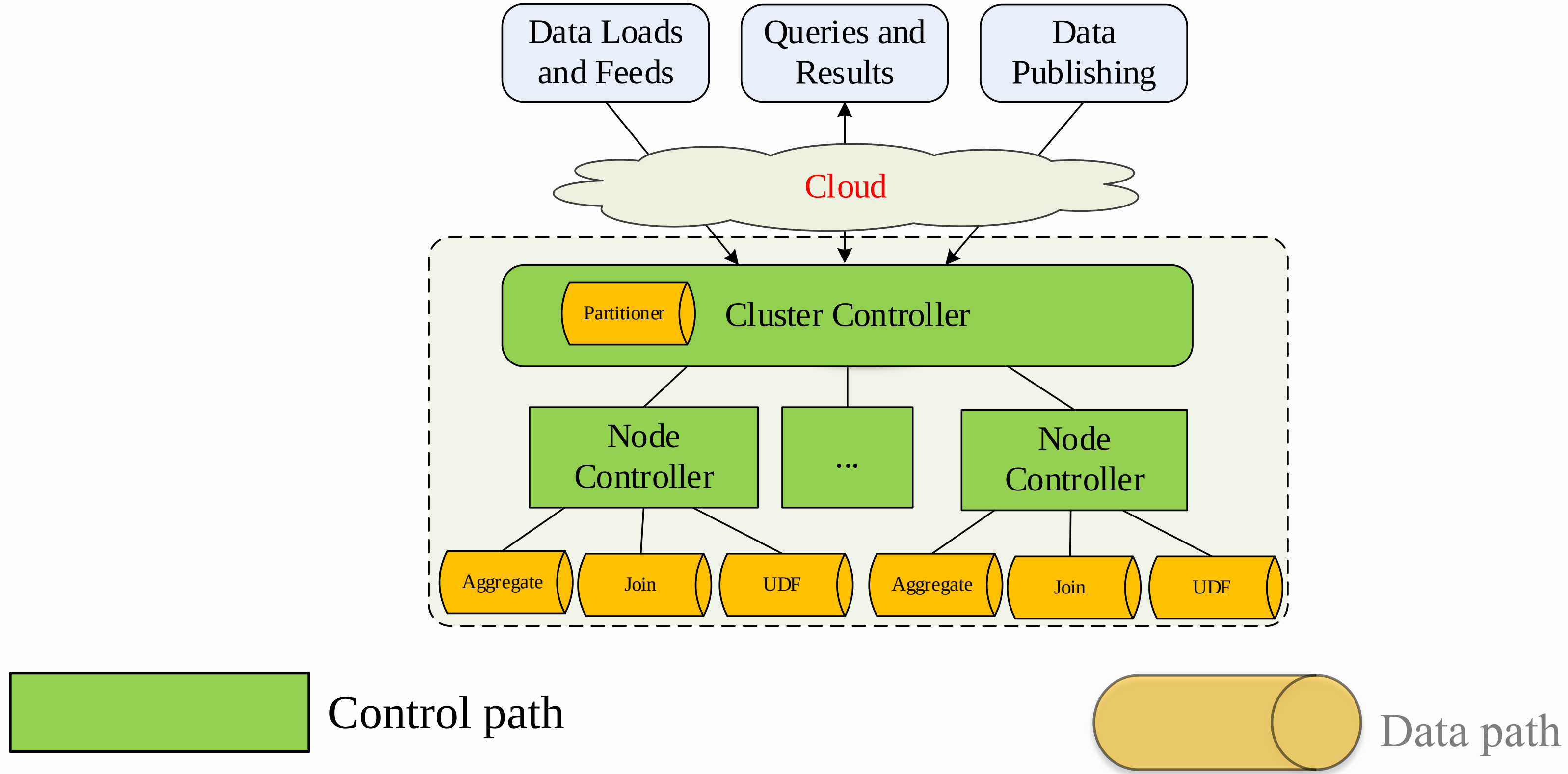
Two Paths, Two Hypotheses



Control path

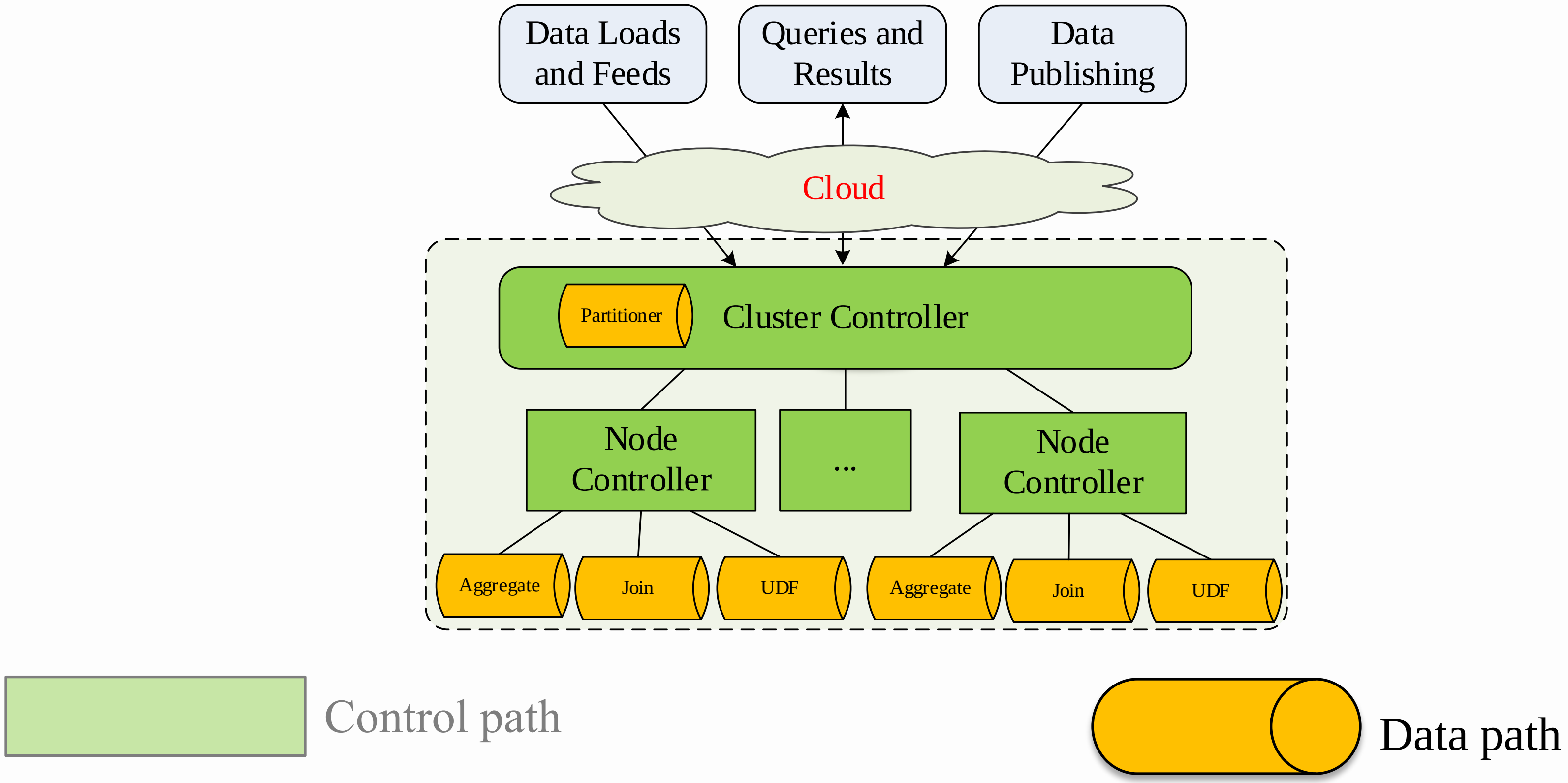
Data path

Two Paths, Two Hypotheses



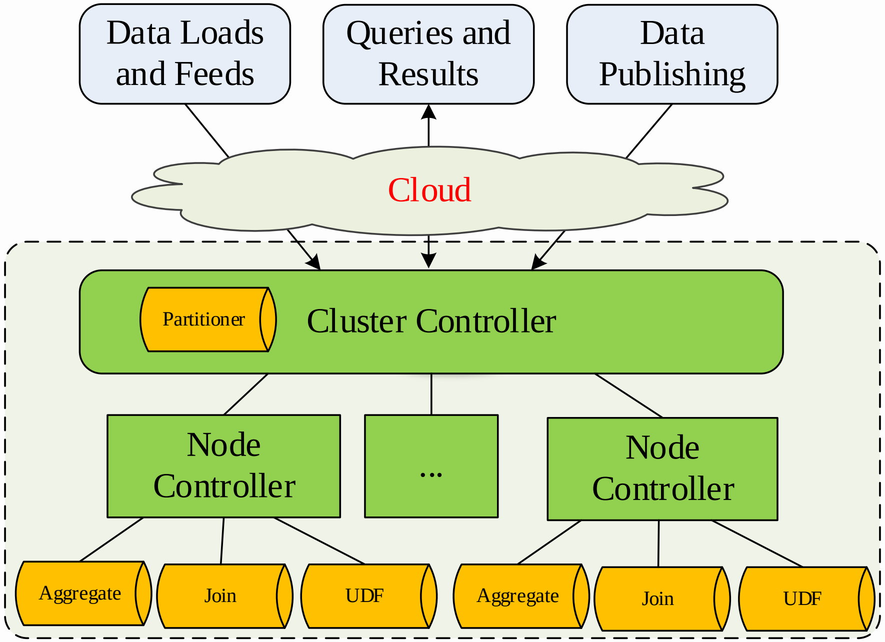
Generational Hypothesis

Two Paths, Two Hypotheses



Generational Hypothesis

Two Paths, Two Hypotheses

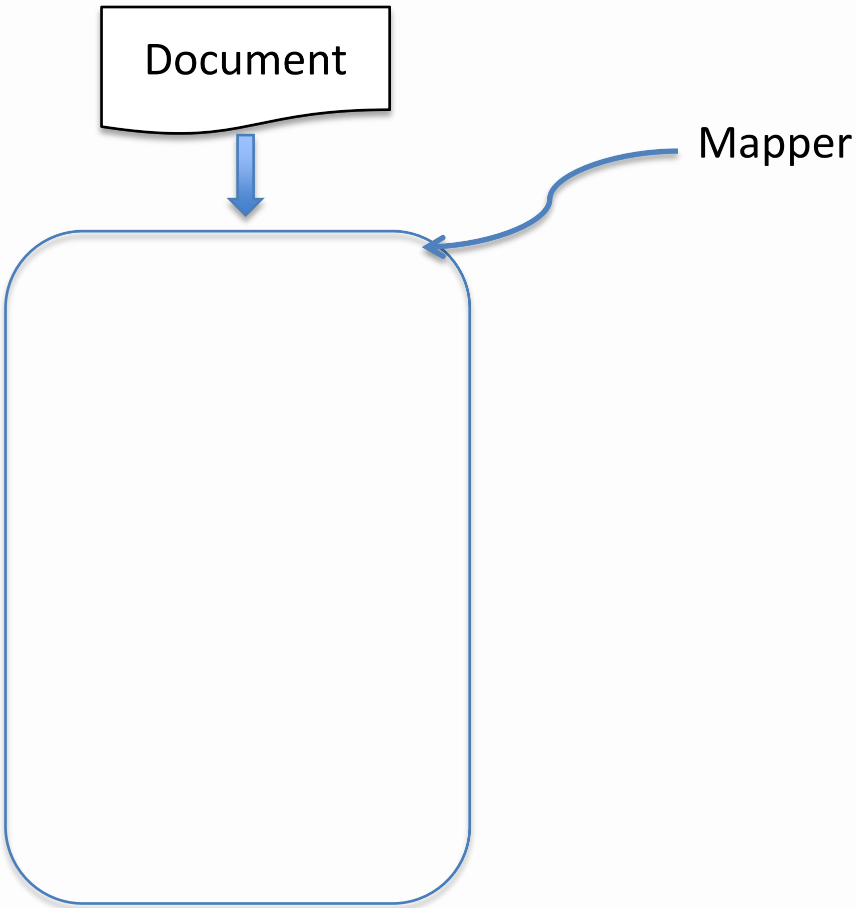


Generational Hypothesis

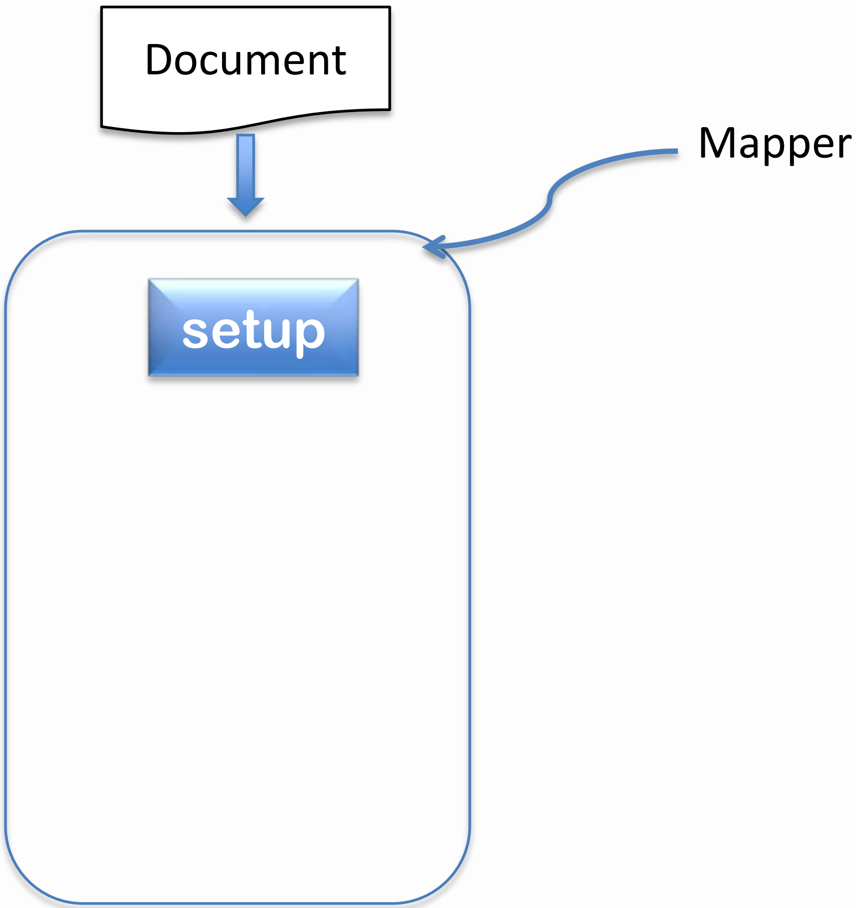
Epochal Hypothesis

WordCount

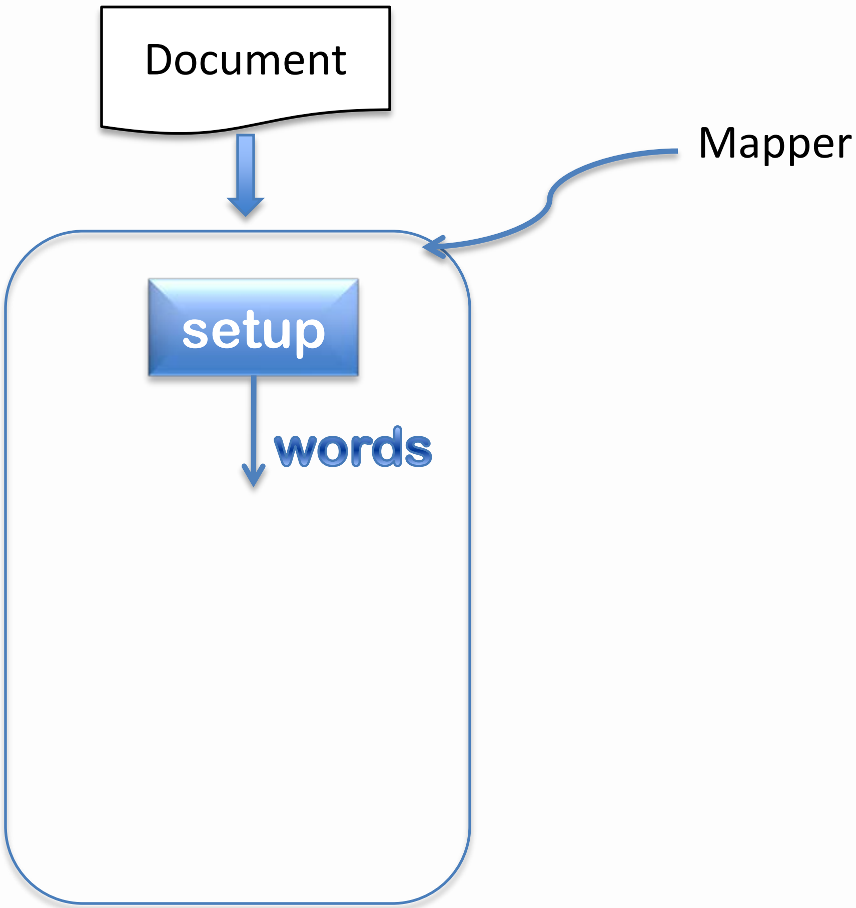
WordCount



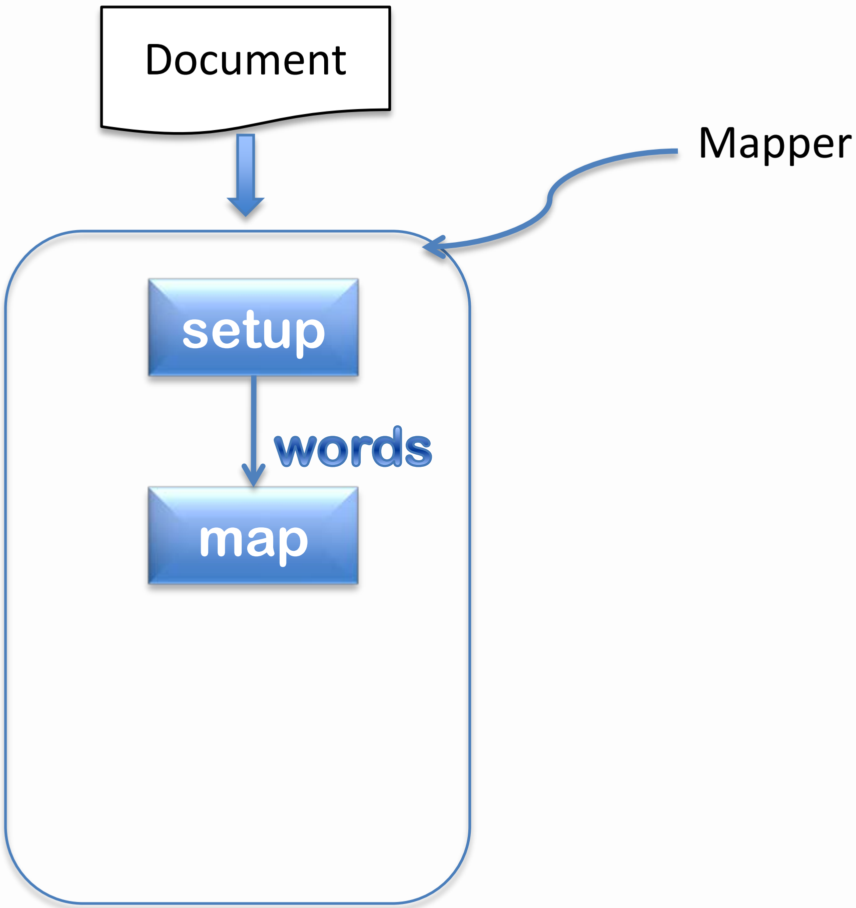
WordCount



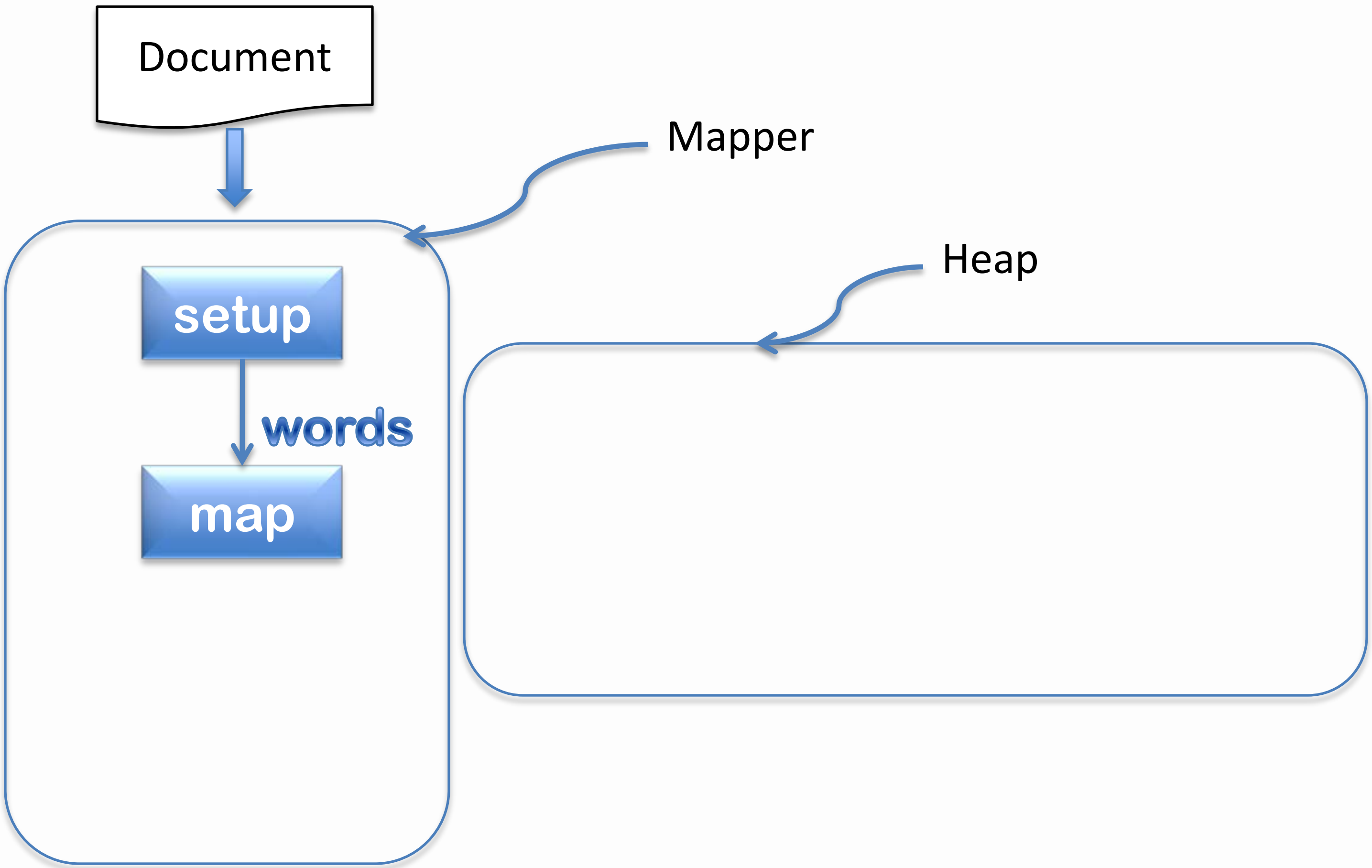
WordCount



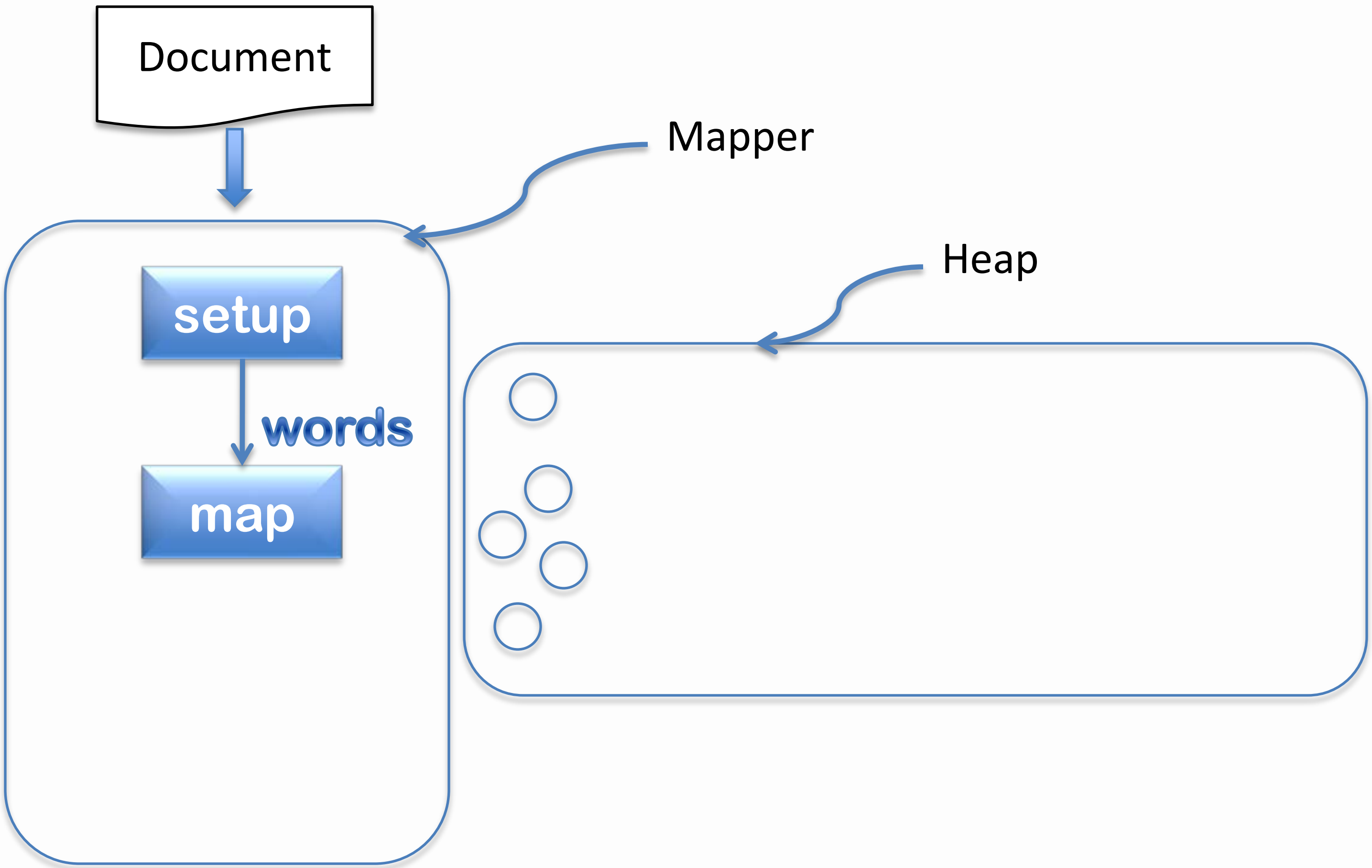
WordCount



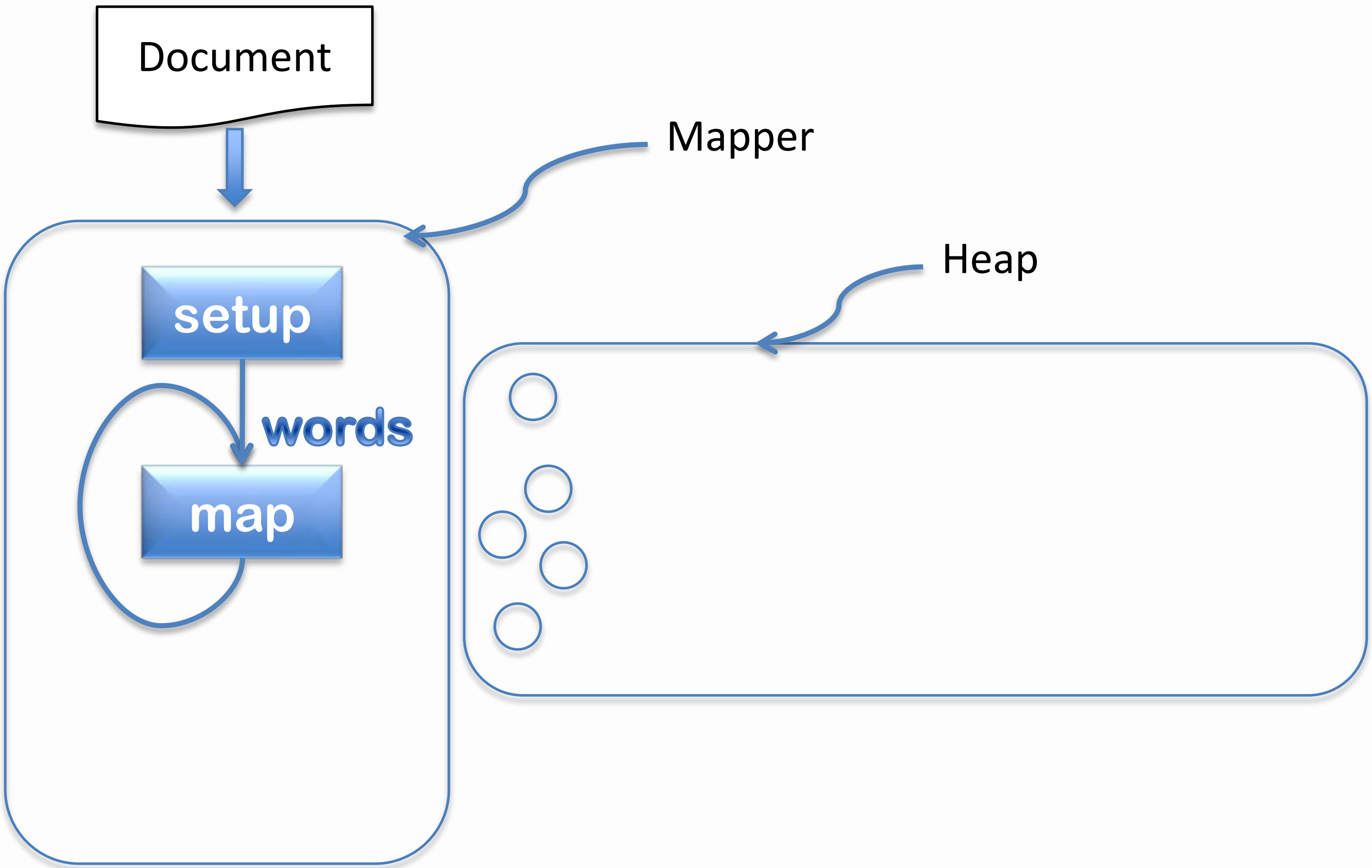
WordCount



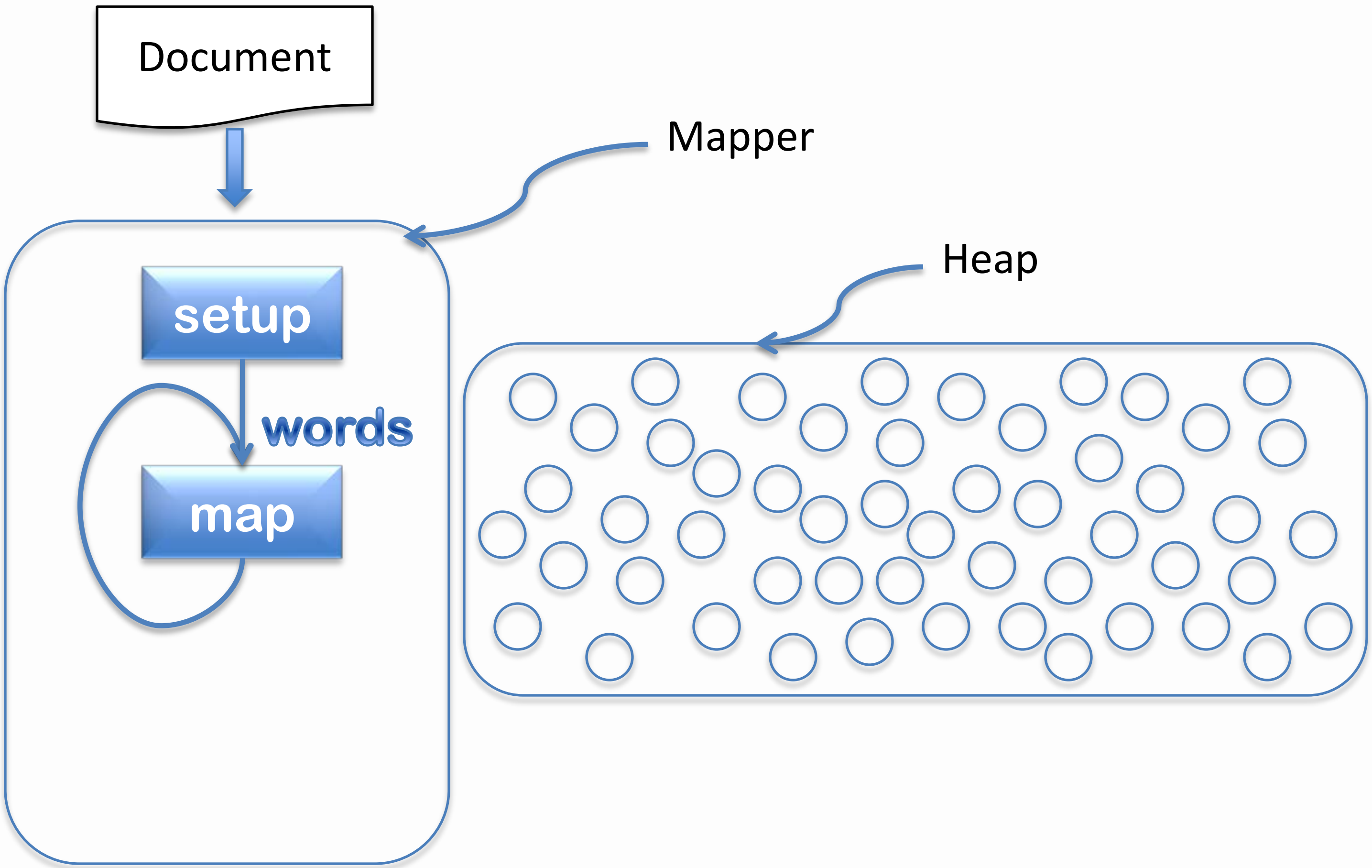
WordCount



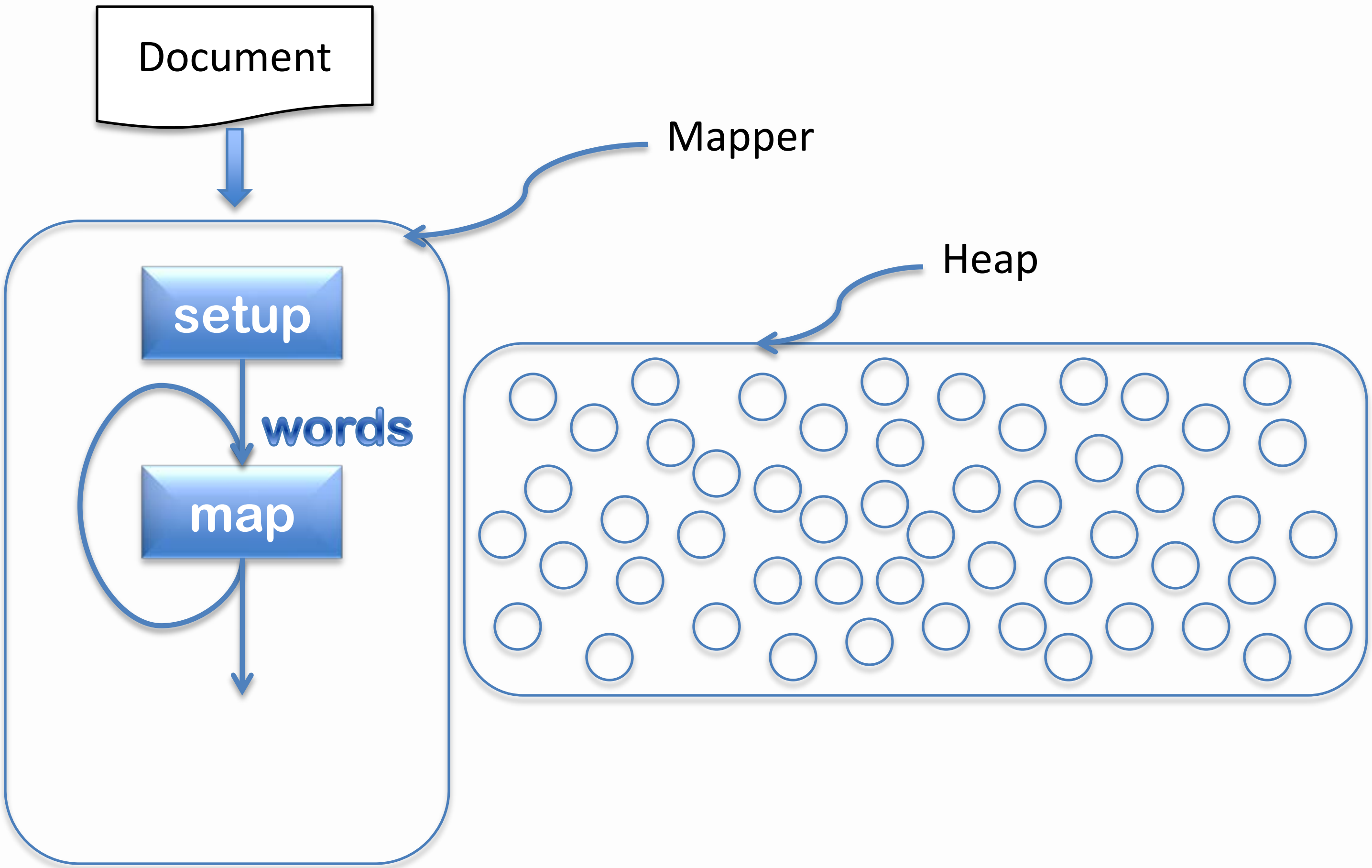
WordCount



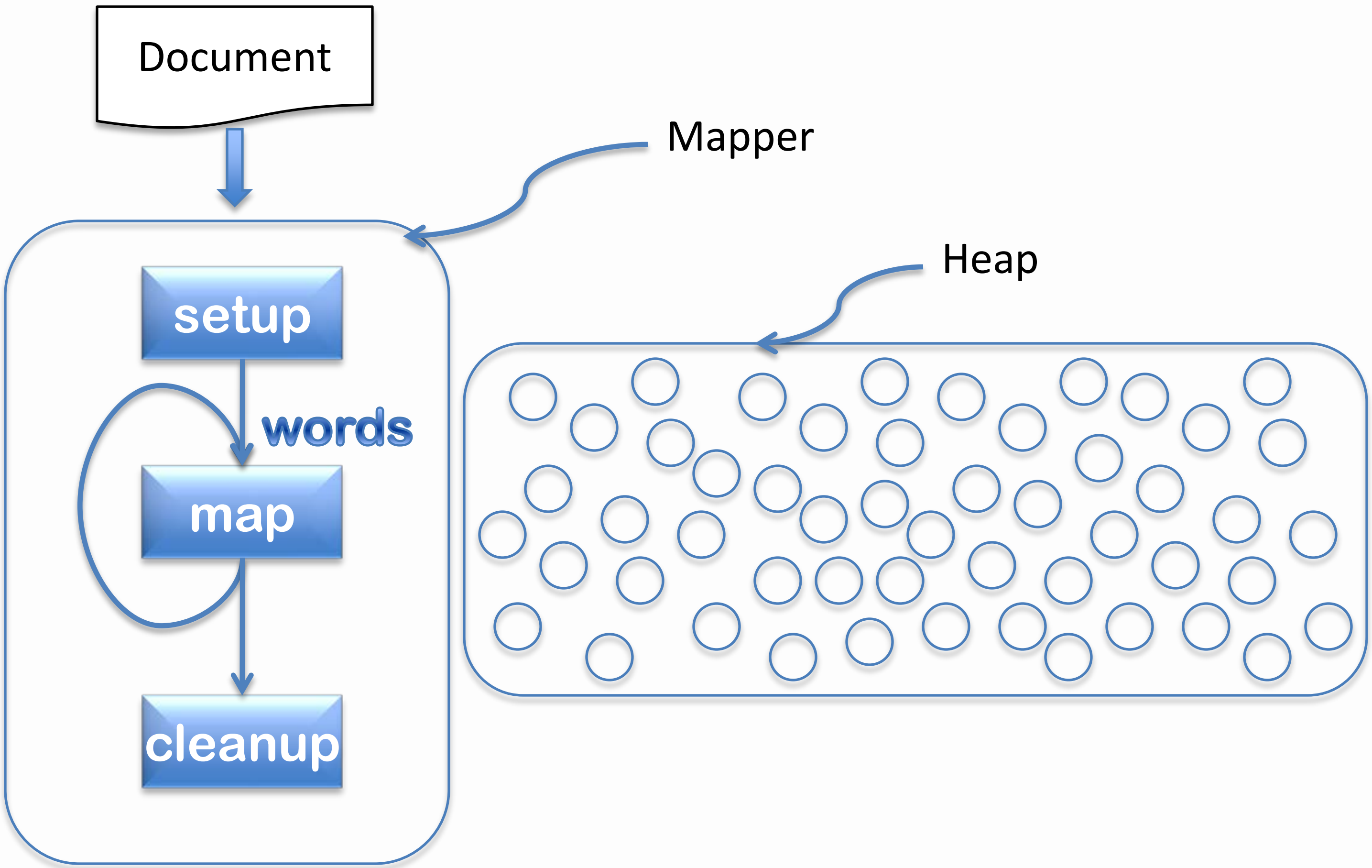
WordCount



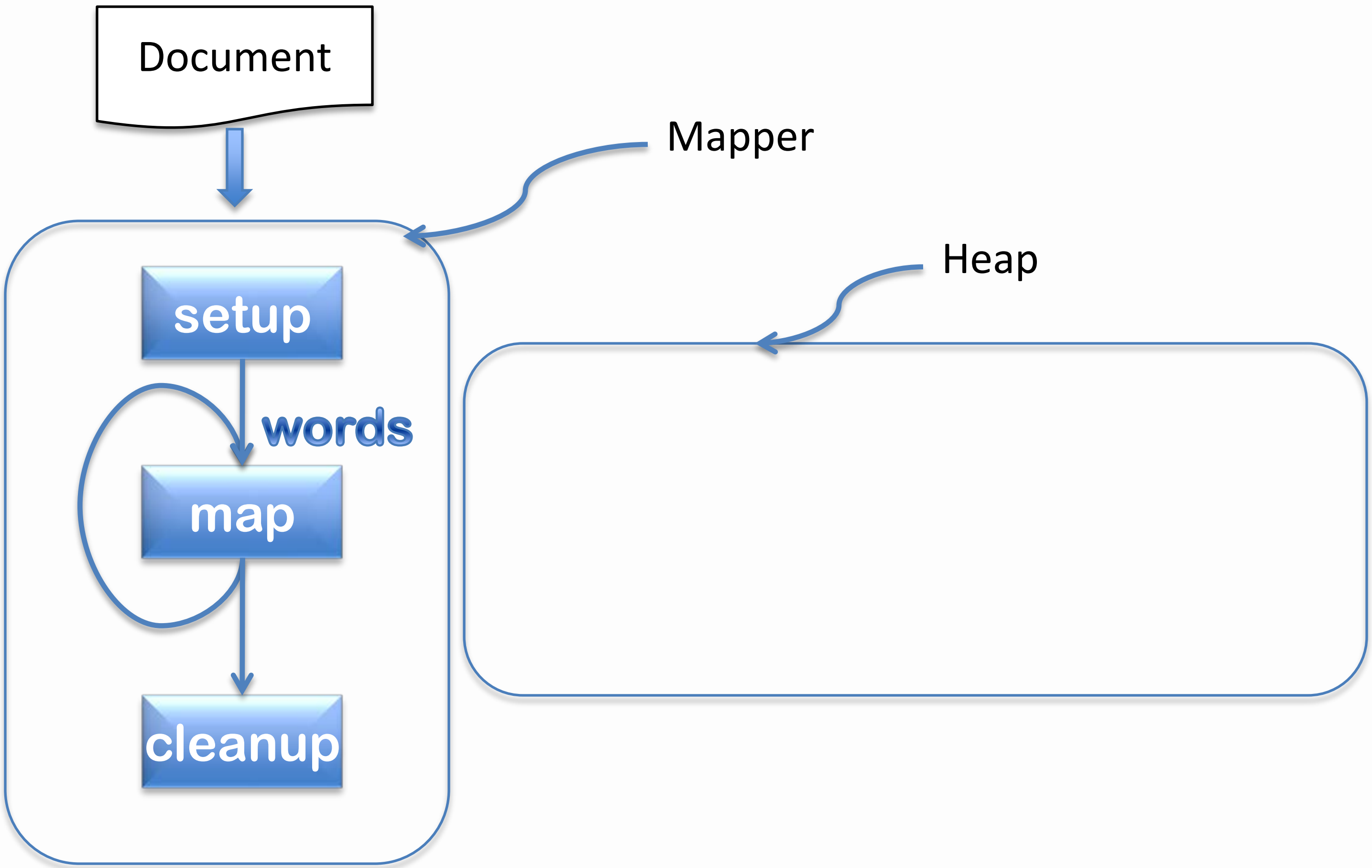
WordCount



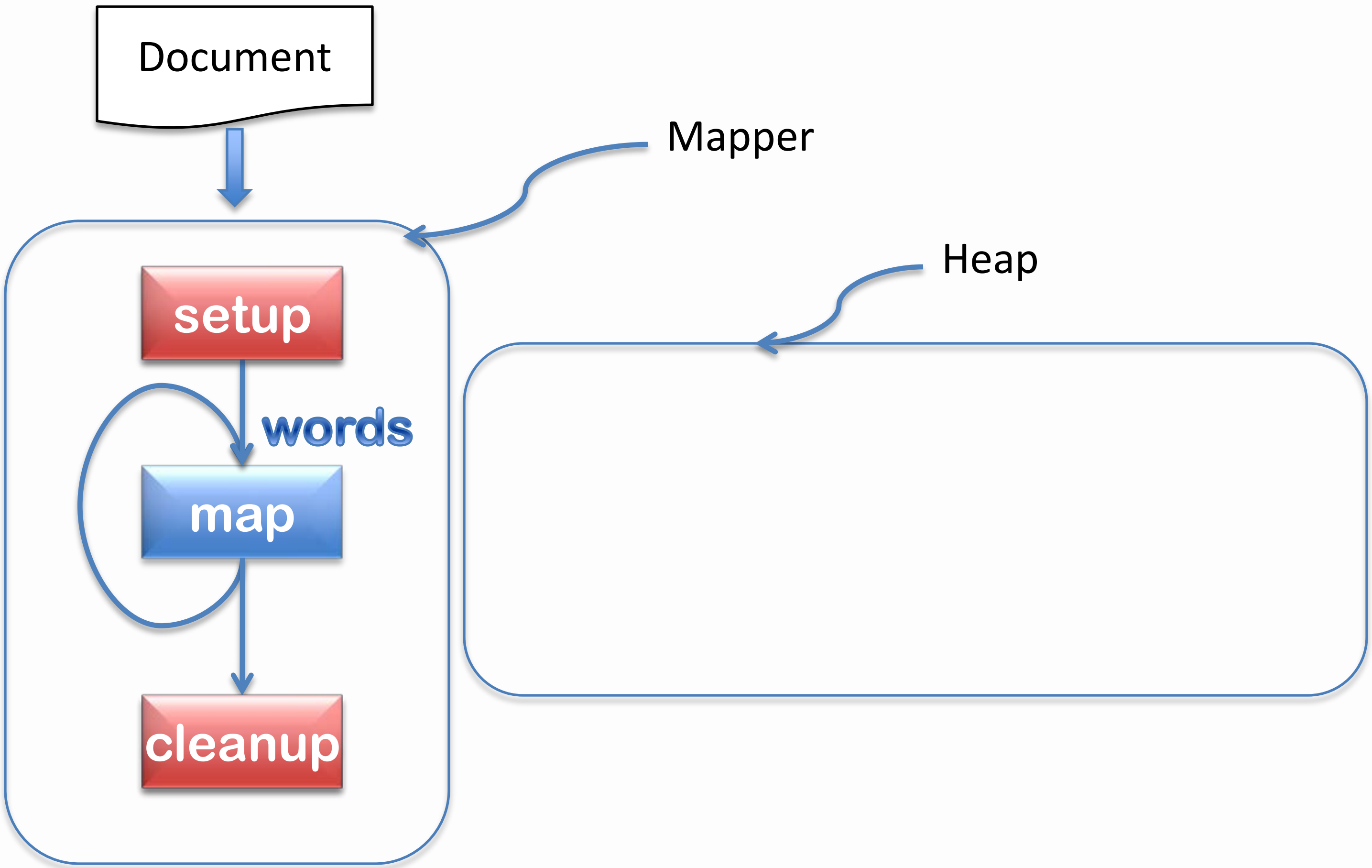
WordCount



WordCount

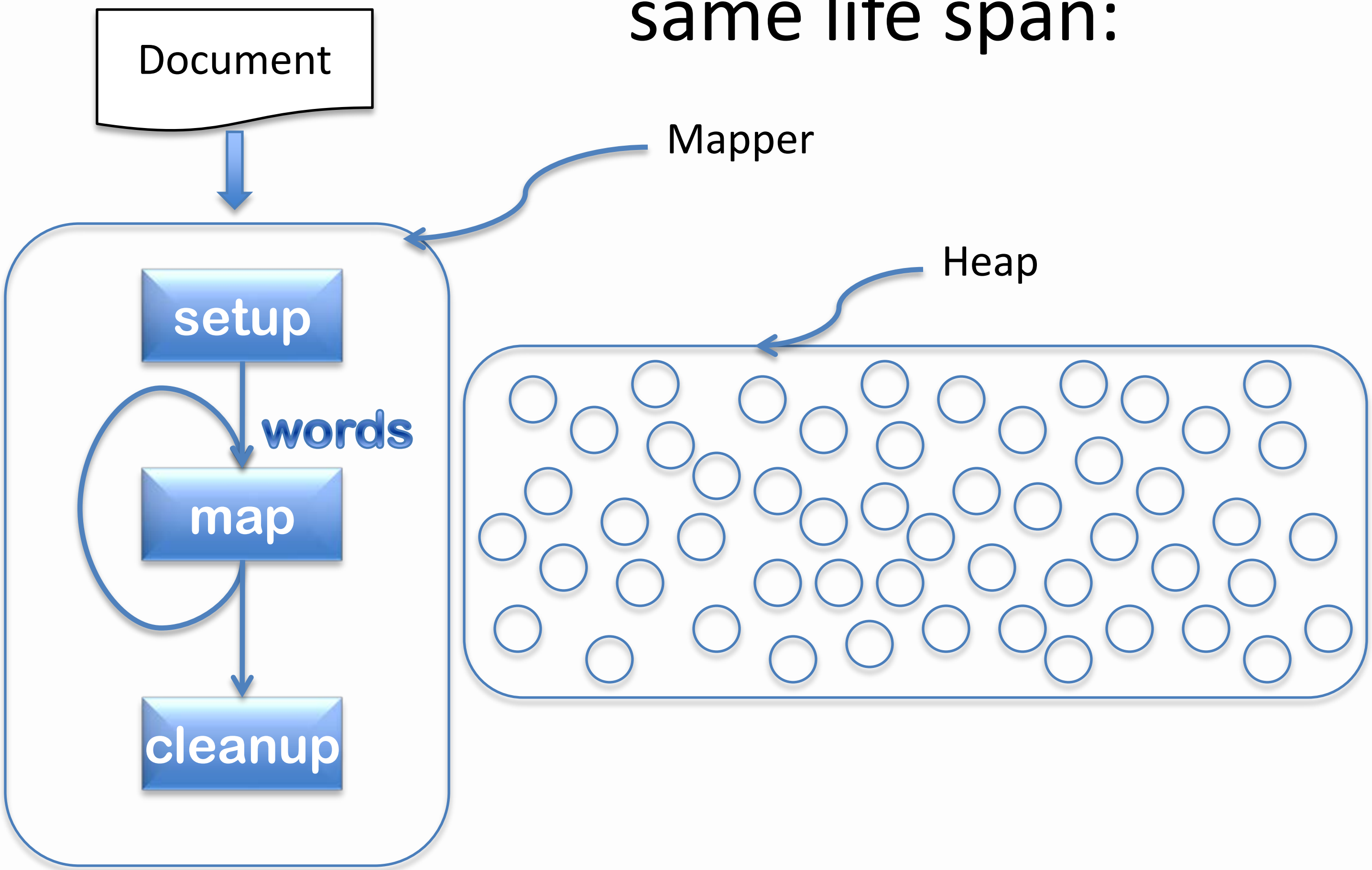


WordCount



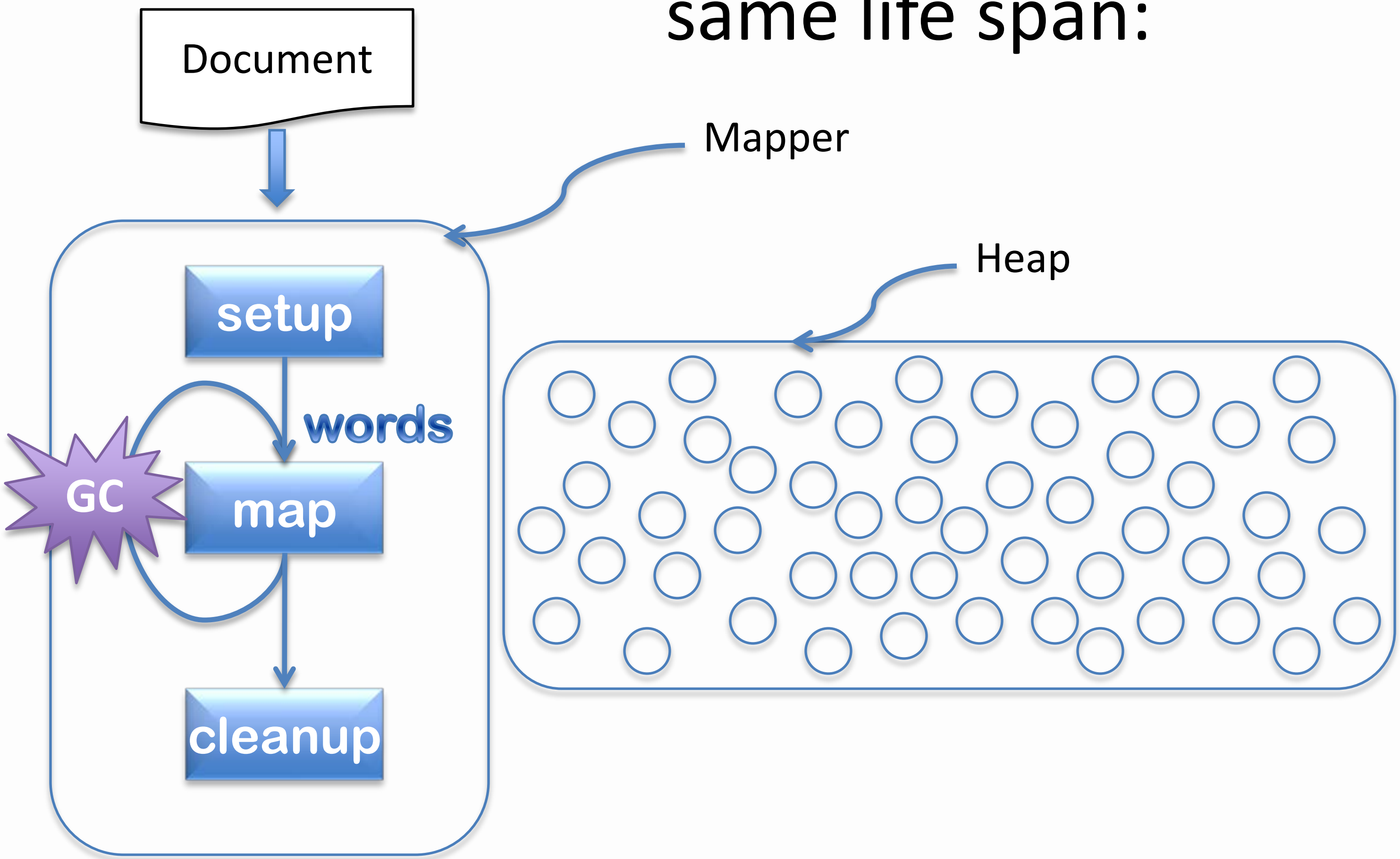
WordCount

- Many data objects have the same life span:



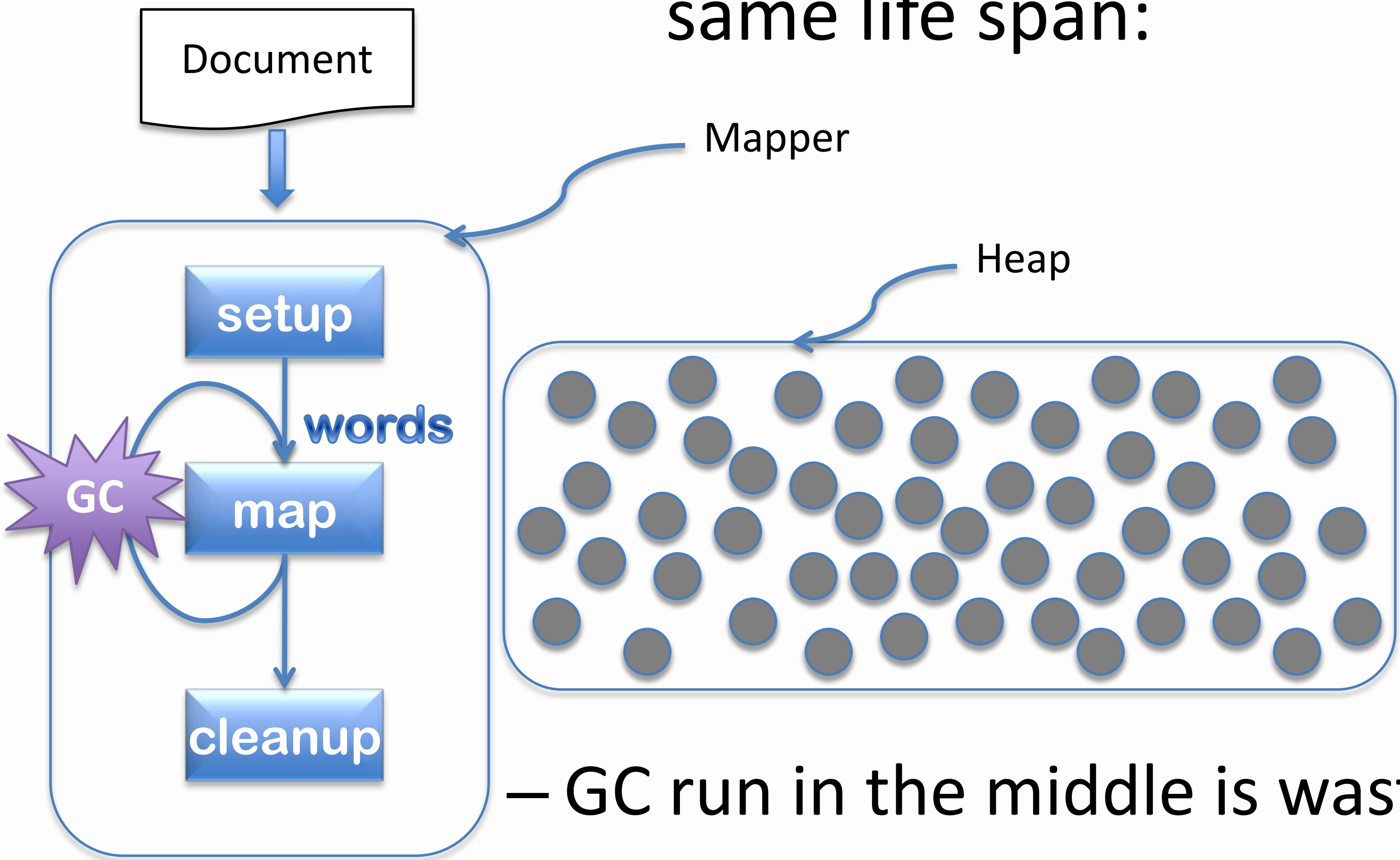
WordCount

- Many data objects have the same life span:



WordCount

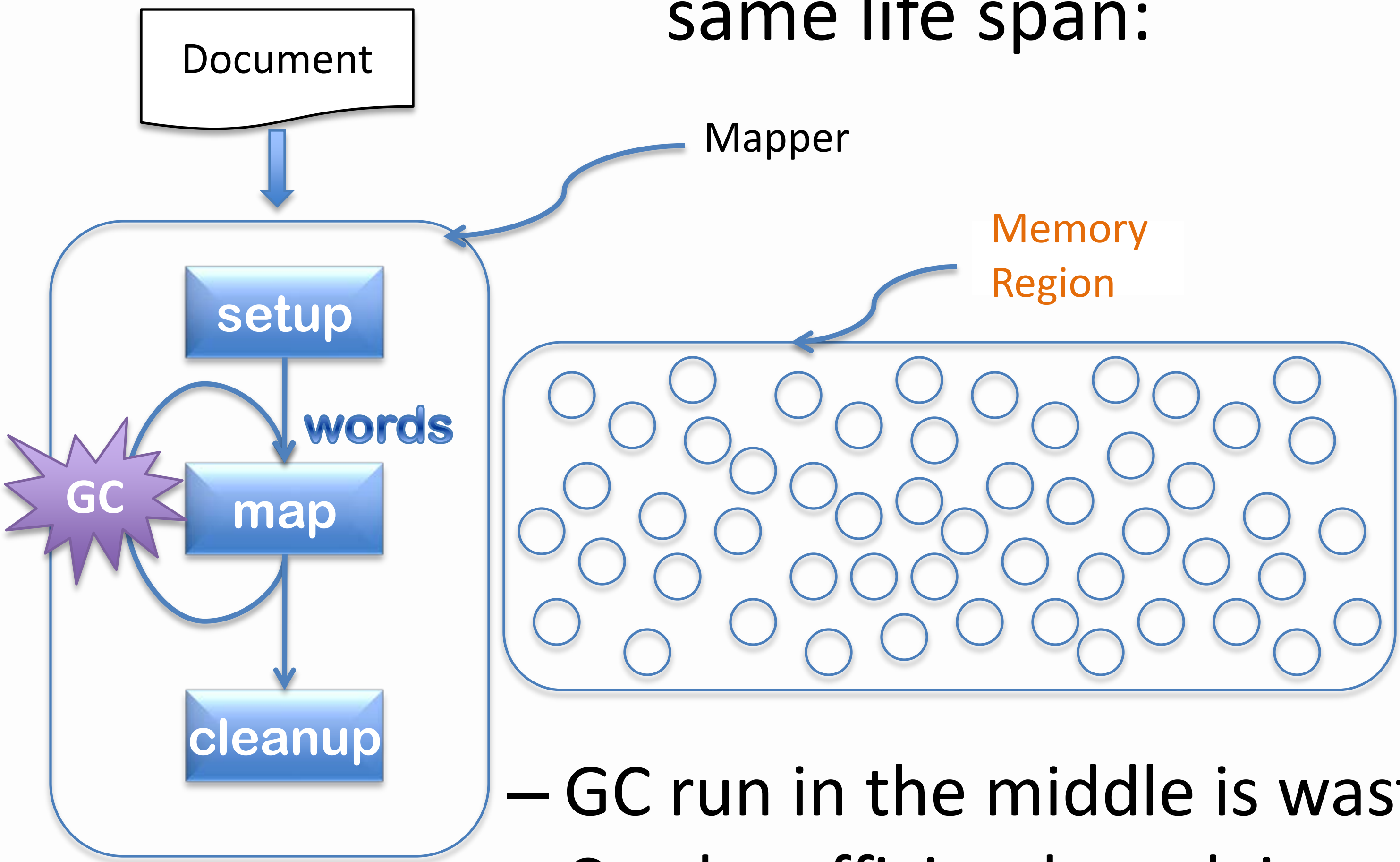
- Many data objects have the same life span:



  – GC run in the middle is wasted

WordCount

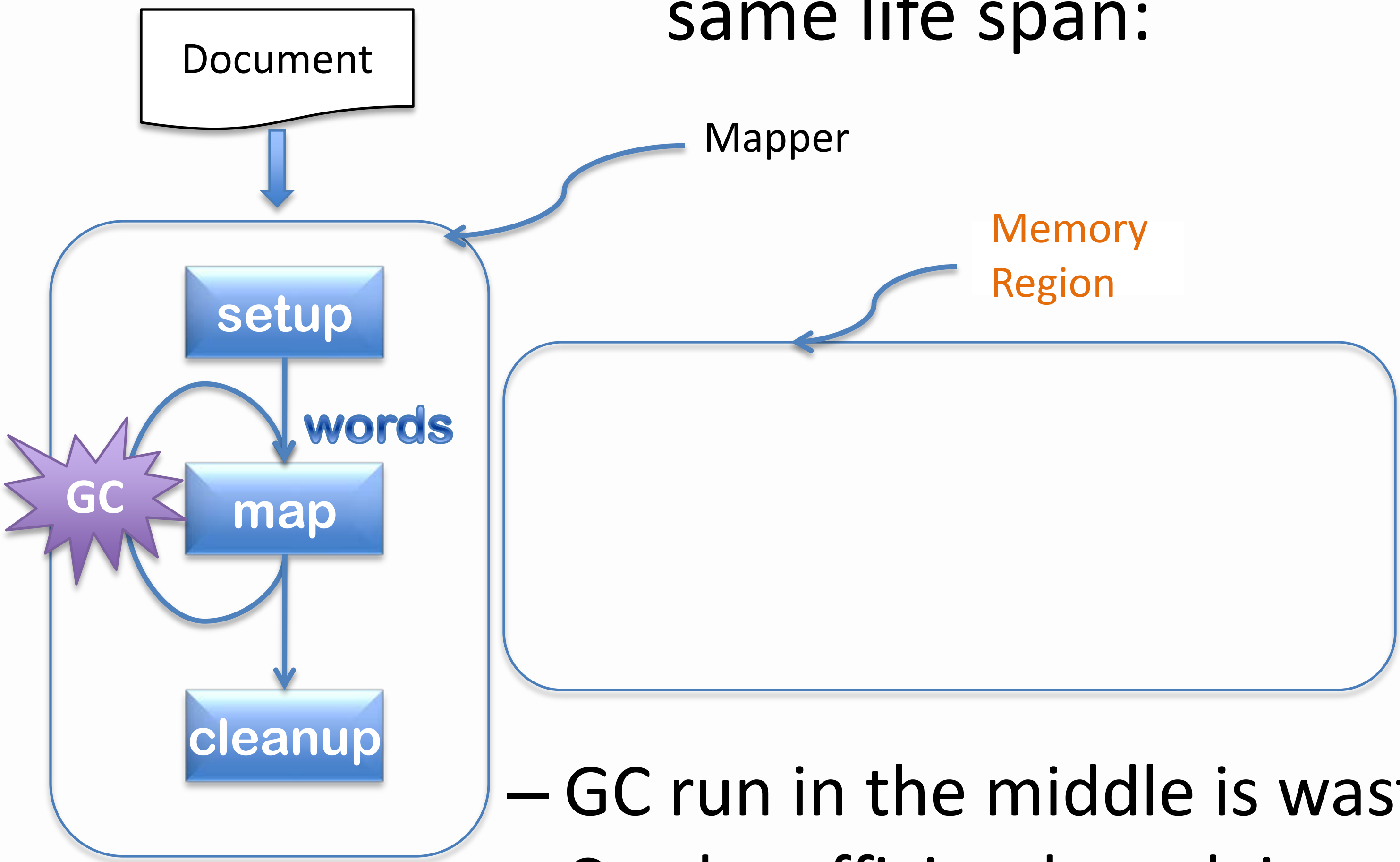
- Many data objects have the same life span:



  - GC run in the middle is wasted
  - Can be efficiently reclaimed together

WordCount

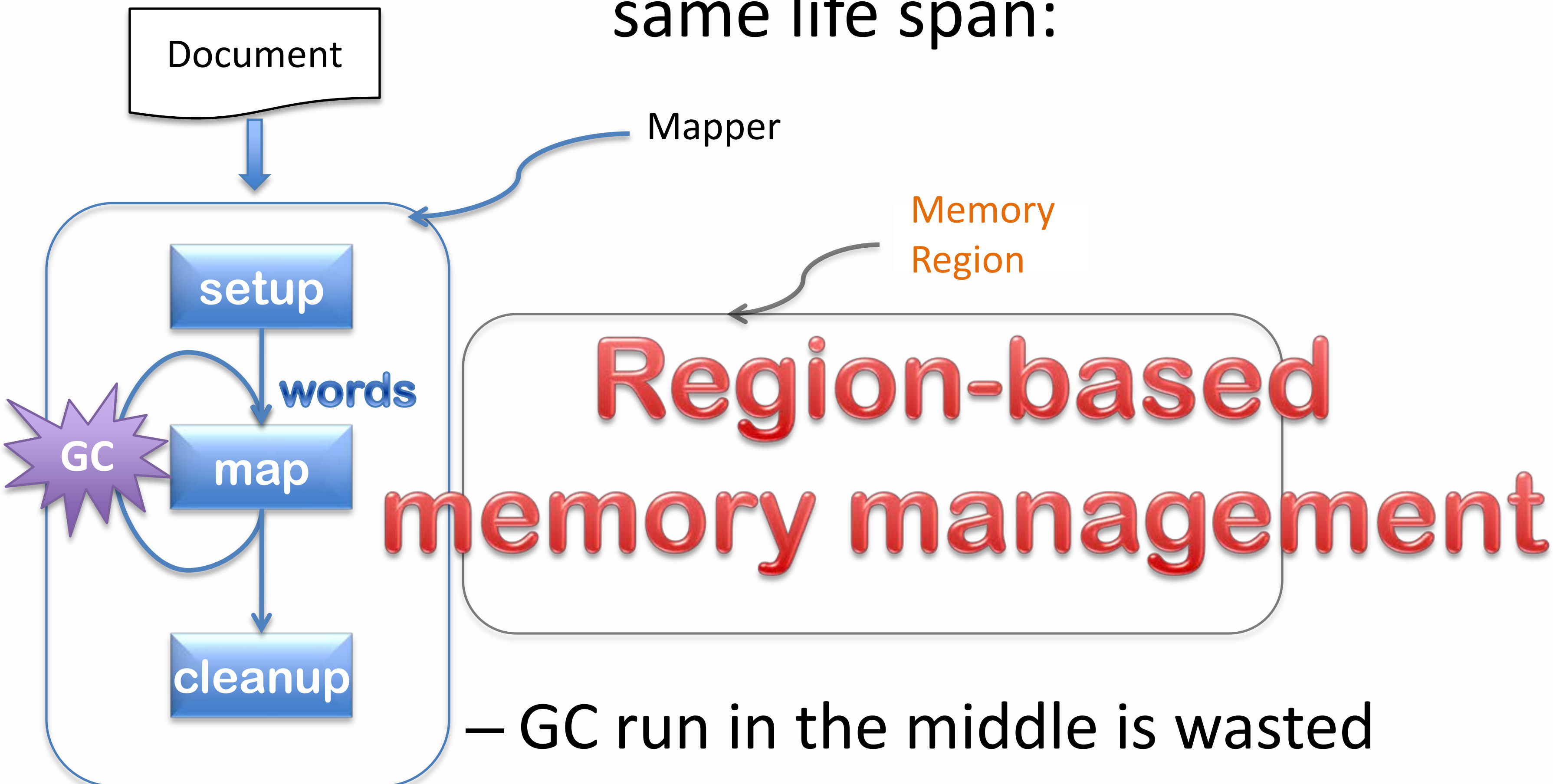
- Many data objects have the same life span:



  - GC run in the middle is wasted
  - Can be efficiently reclaimed together

WordCount

- Many data objects have the same life span:



  – GC run in the middle is wasted
  – Can be efficiently reclaimed together

Region-based Memory Management

- Sophisticated static analysis won't work for data-intensive systems

Region-based Memory Management

- Sophisticated static analysis won't work for data-intensive systems
- What about control path?

Region-based Memory Management

- Sophisticated static analysis won't work for data-intensive systems
- What about control path?

generational GC ➕ region-based memory management

Yak Approach

Heap

Yak Approach

Control Space

Data Space

Yak Approach

Control Space

Generational GC

Data Space

Yak Approach

Control Space

Generational GC

Data Space

Region-based
Memory Management

Yak Approach

Control Space

Generational GC

Data Space

Region-based Memory Management

Reduced memory management cost

Yak Approach

Control Space

Generational GC

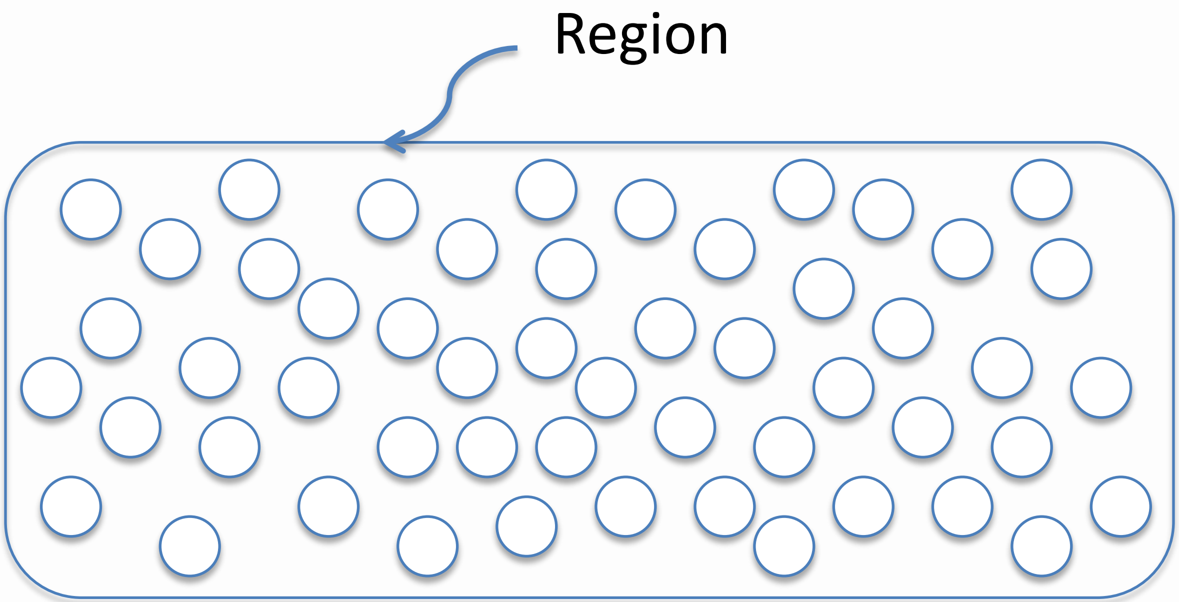
Data Space

Region-based
Memory Management

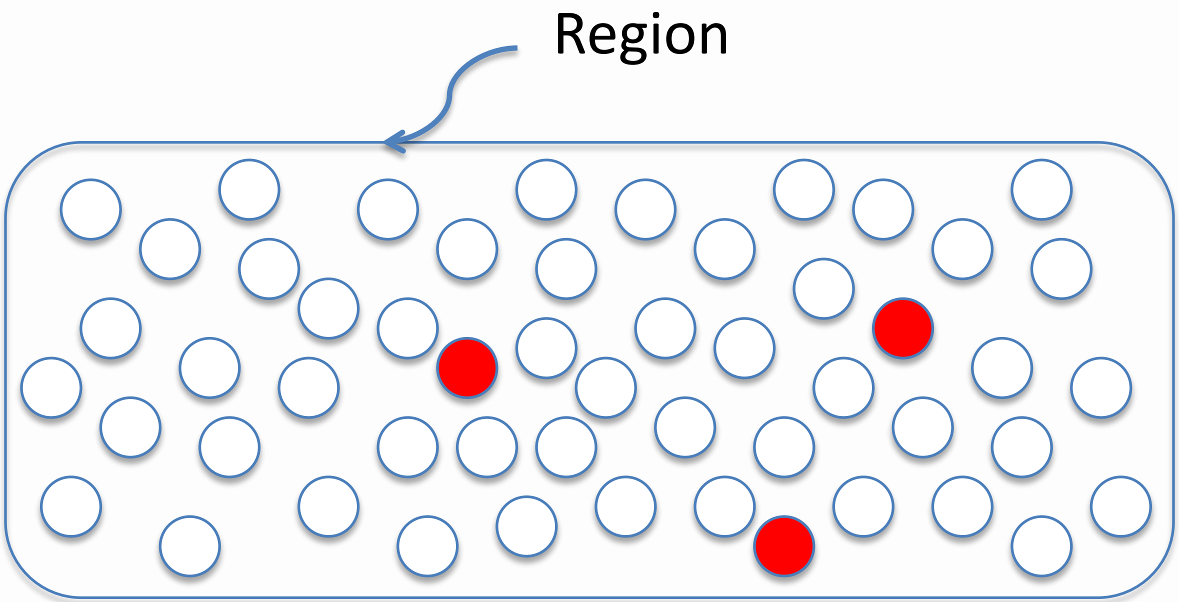
Reduced memory management cost

annotate epoch boundary:
- epoch_start()
- epoch_end()

Correctness



Correctness



Correctness

- User-based approach solution:
  - Facade [Nguyen et al., ASPLOS'15]

Correctness

- User-based approach solution:
  - Facade [Nguyen et al., ASPLOS'15]

annotation & refactoring

Correctness

- User-based approach solution:
  - Facade [Nguyen et al., ASPLOS'15]

annotation & refactoring

Developers

Correctness

- User-based approach solution:
  - Facade [Nguyen et al., ASPLOS'15]

annotation & refactoring

Developers → Yak

Yak: An Automatic Solution

- **Yak**: the first hybrid GC

Yak: An Automatic Solution

- **Yak**: the first hybrid GC
  - Implemented in OpenJDK 8
    - Modified the interpreter, two JIT compilers, the heap layout, the Parallel Scavenge GC
  - NO code refactoring needed;

    correctness guaranteed by the system

Yak: An Automatic Solution

- **Yak**: the first hybrid GC
  - Implemented in OpenJDK 8
    - Modified the interpreter, two JIT compilers, the heap layout, the Parallel Scavenge GC
  - NO code refactoring needed;

    correctness guaranteed by the system
  - On average, vs. default production GC (PS):
    - Reduce **33%** execution time
    - Reduce **78%** GC time

Challenges

- How to create regions?
- How to reclaim regions correctly?

How to Create Regions?

- A region starts at a call to epoch-start and ends at a call to epoch-end
  - The location of epochs affects *performance* but not *correctness*
- Regions are thread-local
- Regions can be nested

How to Create Regions?

```
void main() {

} //end of main
```

How to Create Regions?

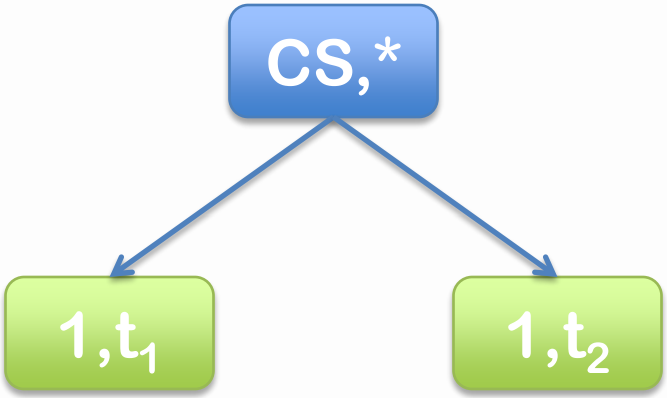
```
void main() {

} //end of main
```

CS,*

How to Create Regions?

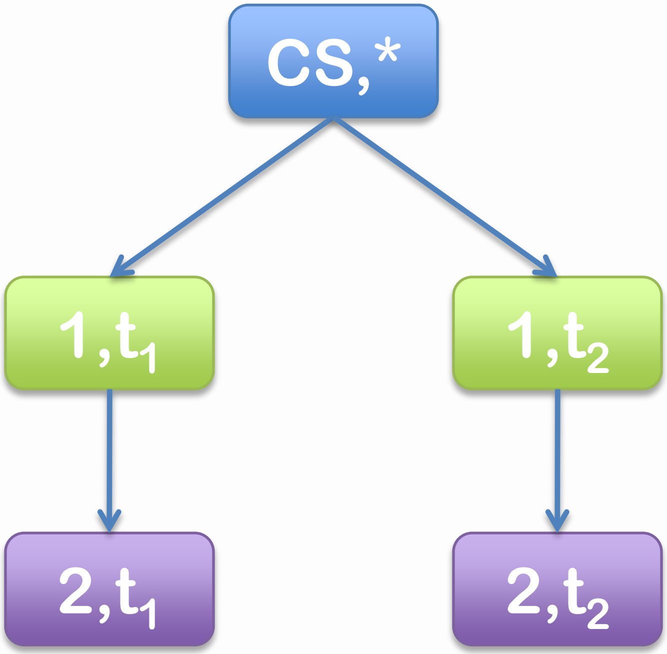
```
void main() {

epoch_start();
for(    ) {



}
epoch_end();

} //end of main
```

epoch #1



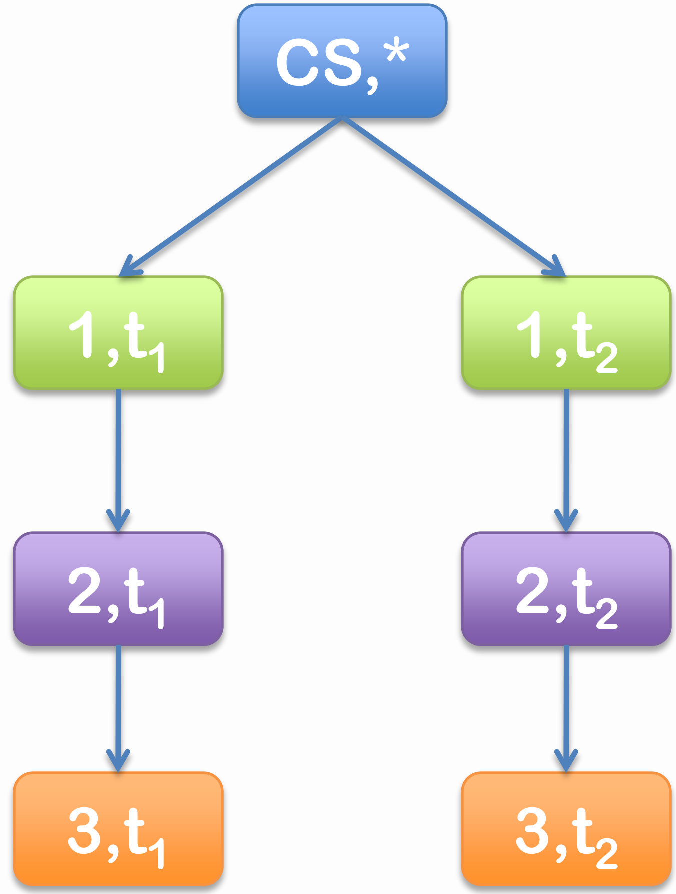
How to Create Regions?

```
void main() {

  epoch_start();                 epoch #1
  for(    ) {
    epoch_start();               epoch #2
    for(    ) {



    }
    epoch_end();
  }
  epoch_end();
} //end of main
```



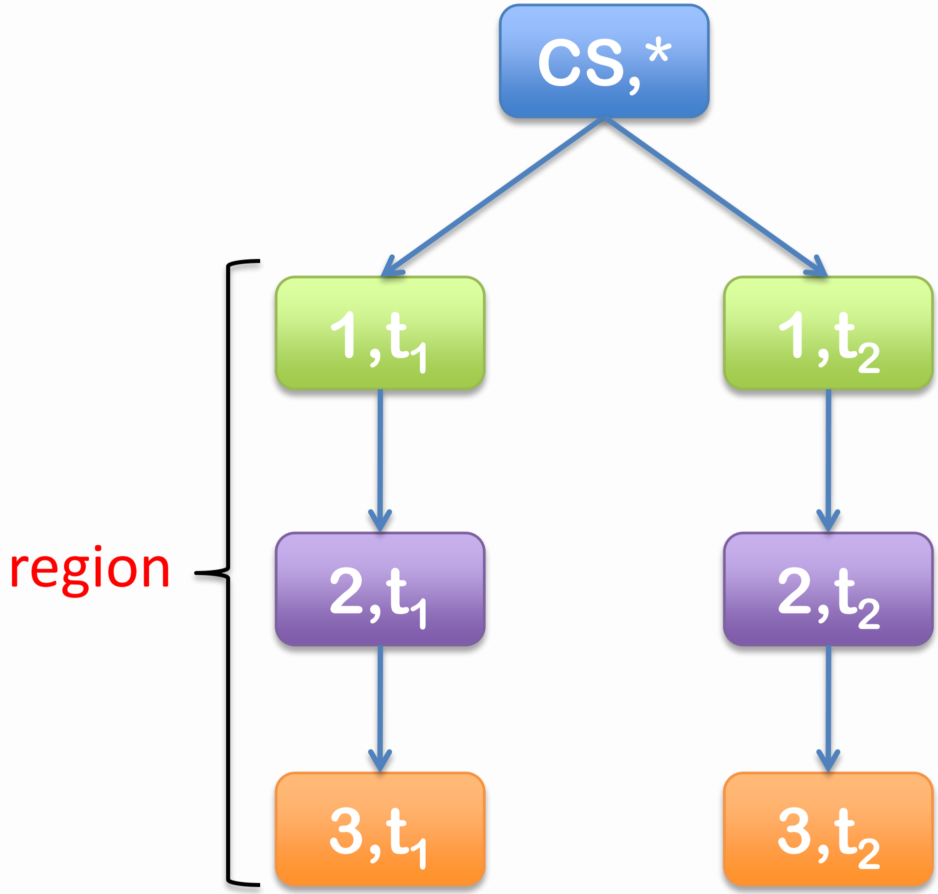
How to Create Regions?

```
void main() {

epoch_start();              epoch #1
for(    ) {
    epoch_start();          epoch #2
    for(    ) {
        epoch_start();      epoch #3
        for(    ) {

        }
        epoch_end();

    }
    epoch_end();
}
epoch_end();
} //end of main
```



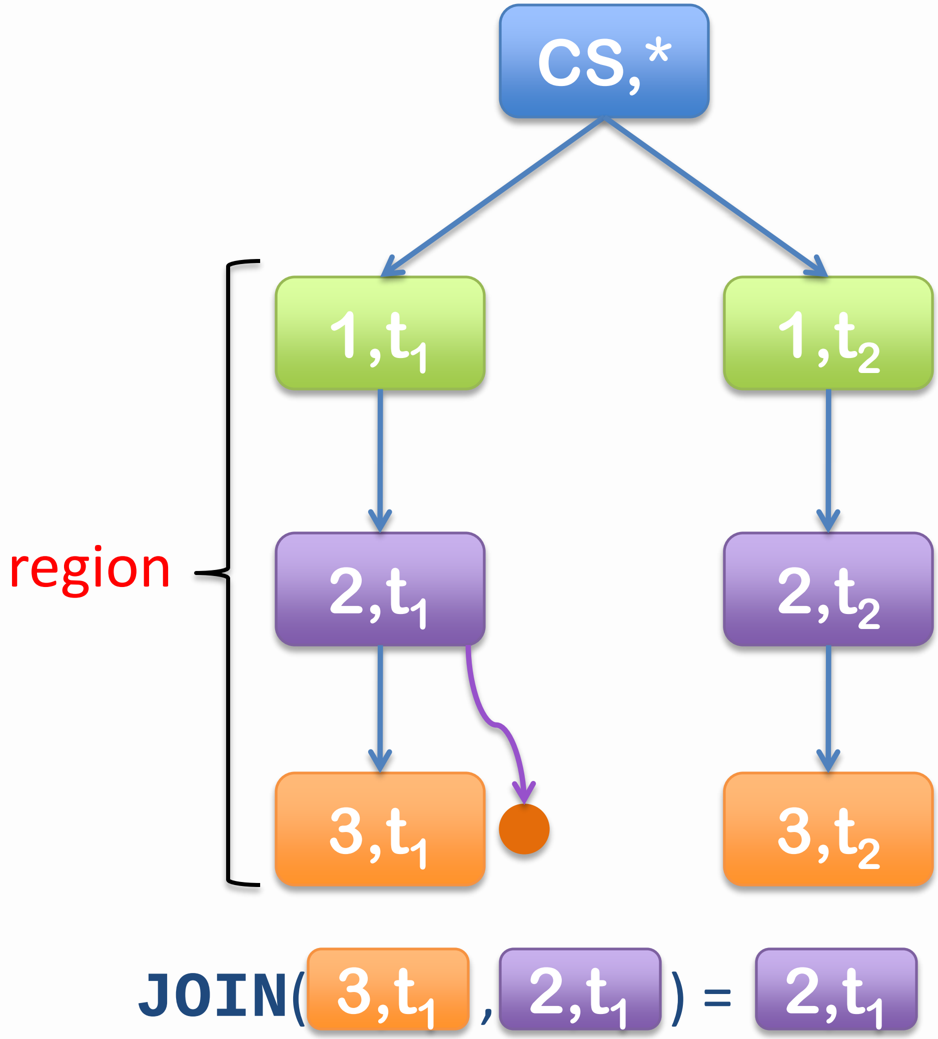
Region Semilattice

```
void main() {

  epoch_start();                epoch #1
  for(    ) {
    epoch_start();              epoch #2
    for(    ) {
      epoch_start();            epoch #3
      for(    ) {

      }
      epoch_end();

    }
    epoch_end();

  }
  epoch_end();

} //end of main
```



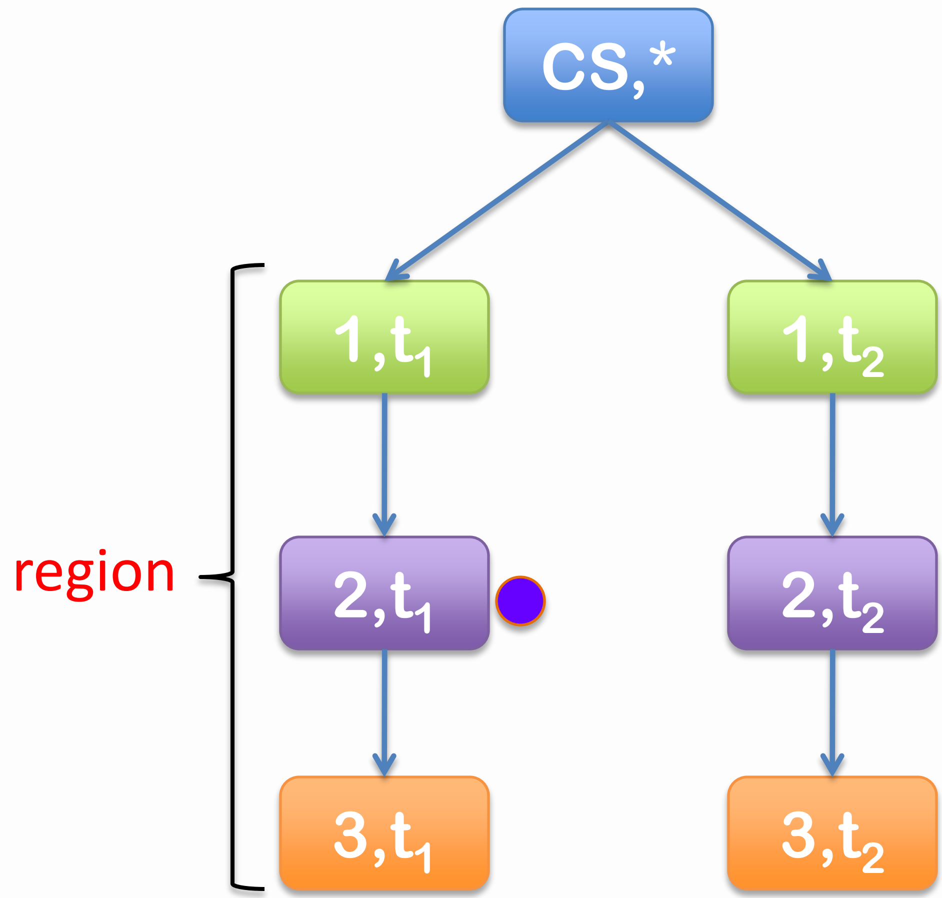
Region Semilattice

```
void main() {

epoch_start();                epoch #1
for(    ) {
   epoch_start();             epoch #2
   for(    ) {
      epoch_start();          epoch #3
      for(    ) {

      }
      epoch_end();

   }
   epoch_end();
}
epoch_end();
} //end of main
```



Region Semilattice

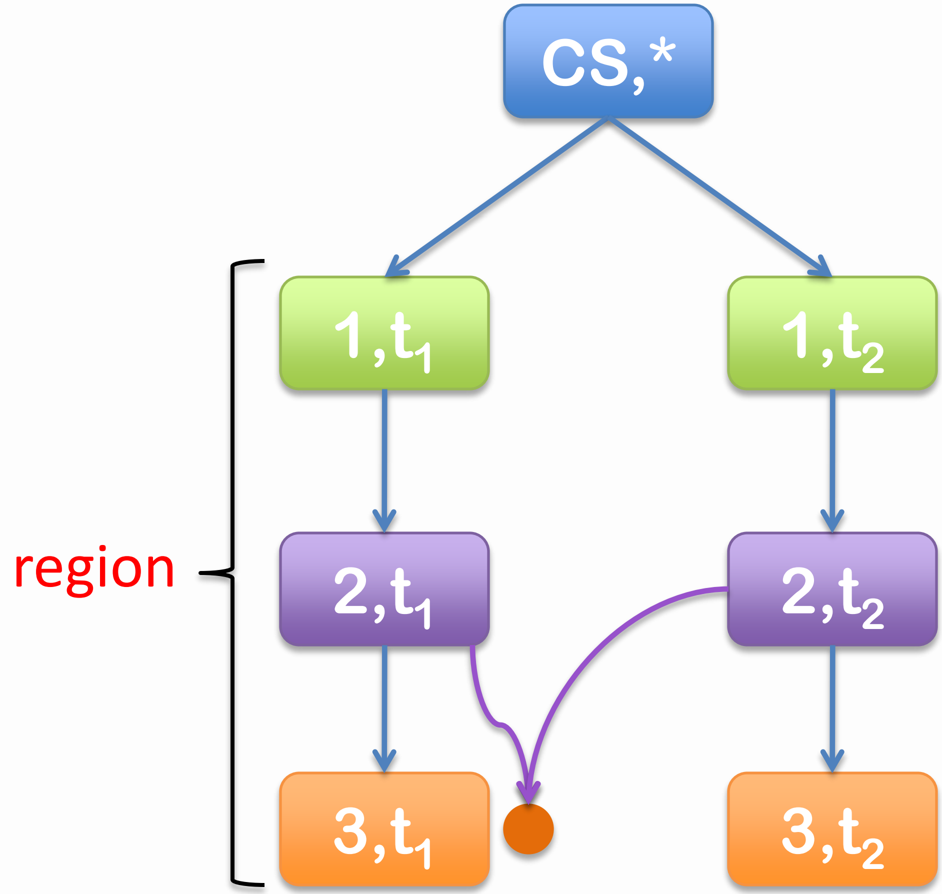
```
void main() {

epoch_start();                 epoch #1
for(    ) {
    epoch_start();             epoch #2
    for(    ) {
        epoch_start();         epoch #3
        for(    ) {

        }
        epoch_end();

    }
    epoch_end();
}
epoch_end();

} //end of main
```



$\mathrm{JOIN}(3,t_1\,,\ 2,t_1) = 2,t_1$

Region Semilattice

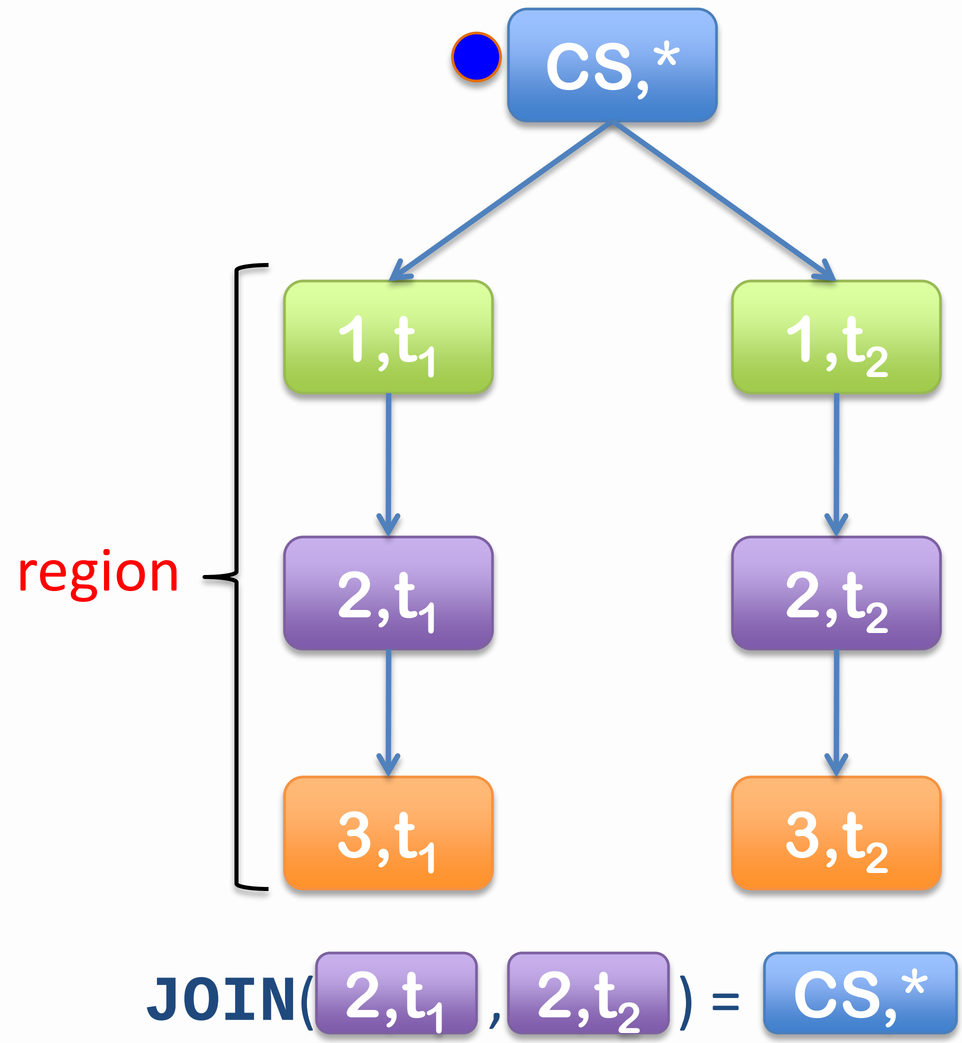
```
void main() {

epoch_start();                 epoch #1
for(    ) {
    epoch_start();             epoch #2
    for(    ) {
        epoch_start();         epoch #3
        for(    ) {

        }
        epoch_end();

    }
    epoch_end();
}
epoch_end();

} //end of main
```



JOIN(2,t_1 , 2,t_2) = CS,*

Region Semilattice

```
void main() {

epoch_start();                epoch #1
for(    ) {
    epoch_start();            epoch #2
    for(    ) {
        epoch_start();        epoch #3
        for(    ) {

        }
        epoch_end();

    }
    epoch_end();
}
epoch_end();
} //end of main
```



How to Reclaim Regions Correctly?

Object Promotion Algorithm

How to Reclaim Regions Correctly?

Object Promotion Algorithm

Two key tasks:

- What: Identify escaping objects:

How to Reclaim Regions Correctly?

Object Promotion Algorithm

Two key tasks:

- What: Identify escaping objects:

  Tracking of cross-region/space references in write barrier

  - A fast path for intra-region references
  - Inter-region references are recorded in the remember sets of their destination regions

How to Reclaim Regions Correctly?

Object Promotion Algorithm

Two key tasks:

- What: Identify escaping objects:

  Tracking of cross-region/space references in write barrier

  - A fast path for intra-region references
  - Inter-region references are recorded in the remember sets of their destination regions

- Where: Decide the relocation destination:

How to Reclaim Regions Correctly?

Object Promotion Algorithm

Two key tasks:

- What: Identify escaping objects:

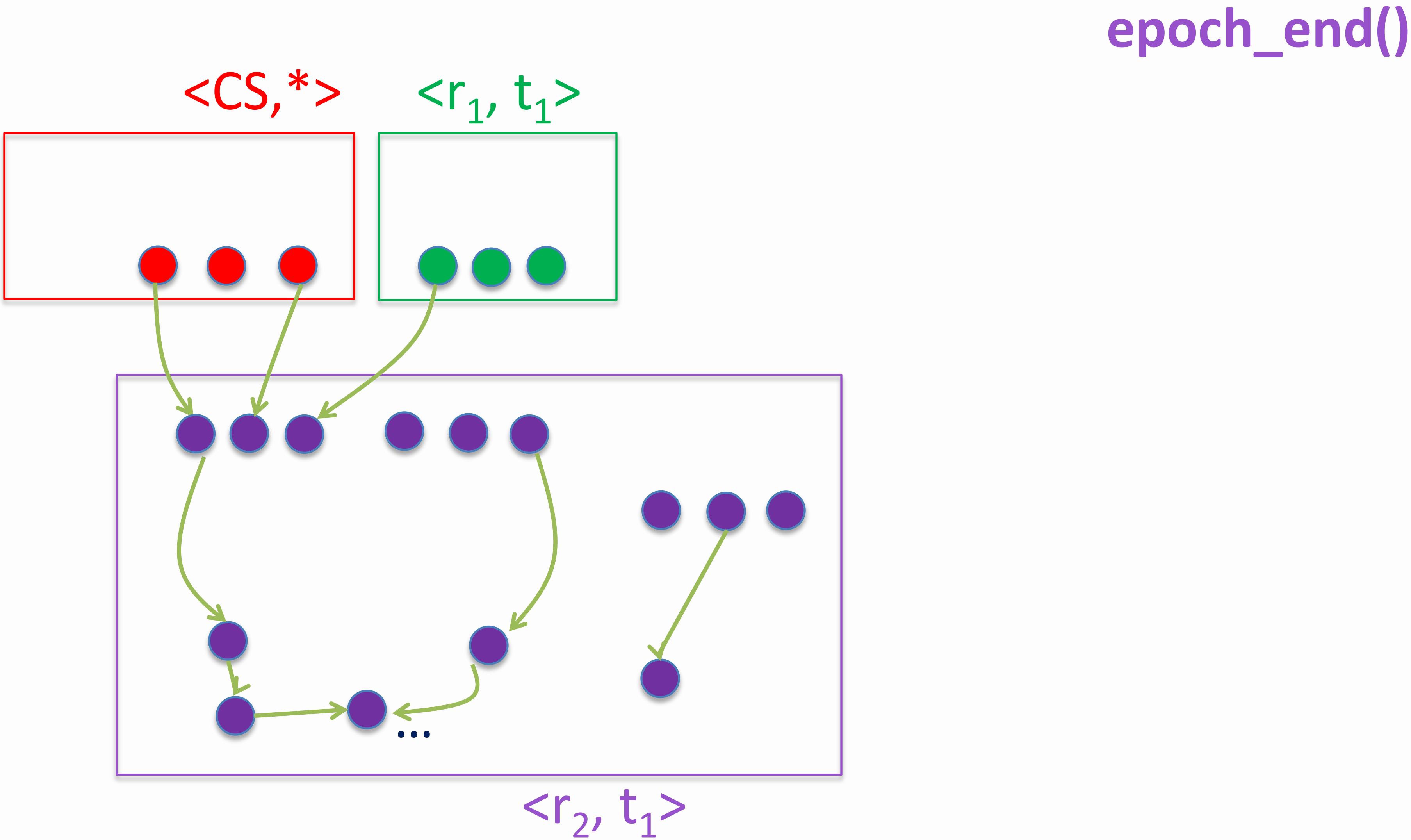
  Tracking of cross-region/space references in write barrier

  - A fast path for intra-region references
  - Inter-region references are recorded in the remember sets of their destination regions

- Where: Decide the relocation destination:

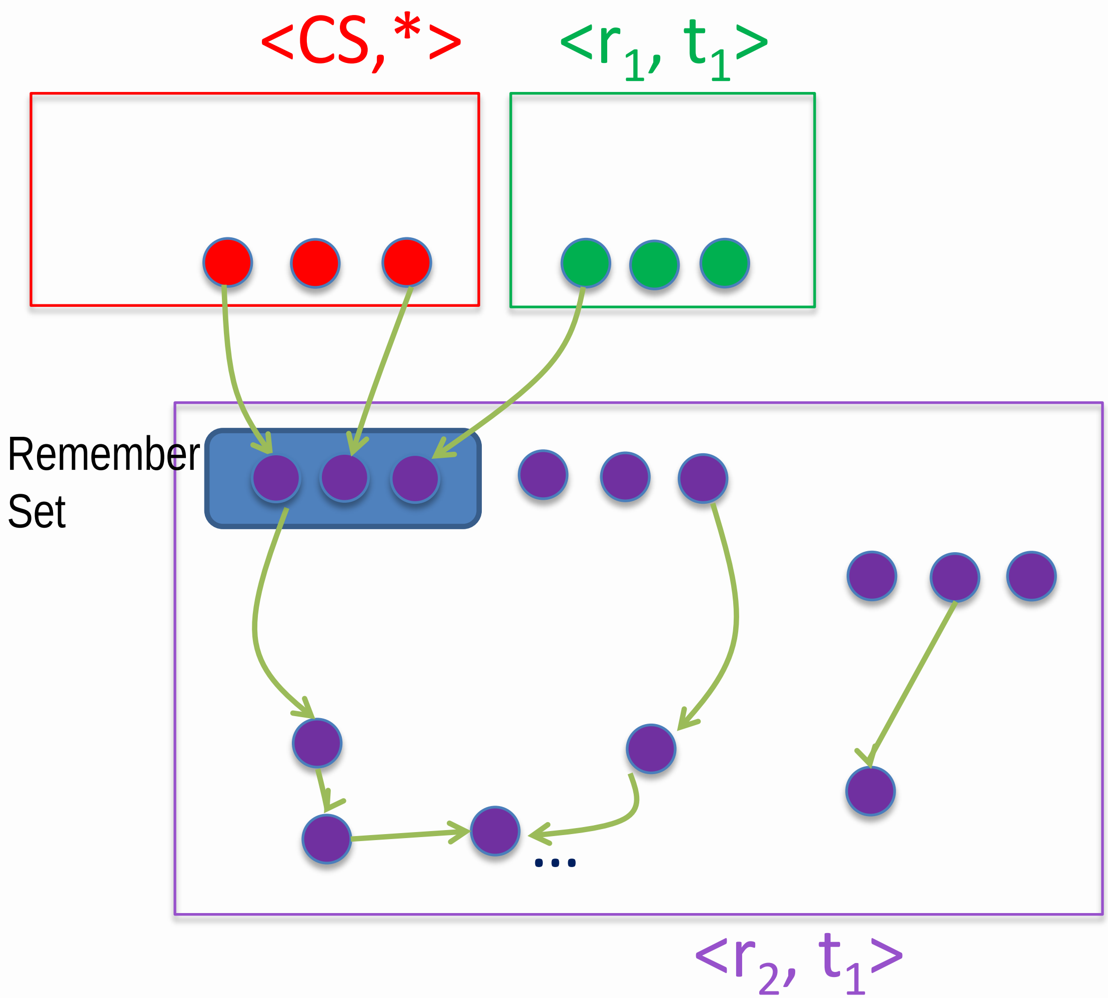
  Query region semilattice

Region Deallocation



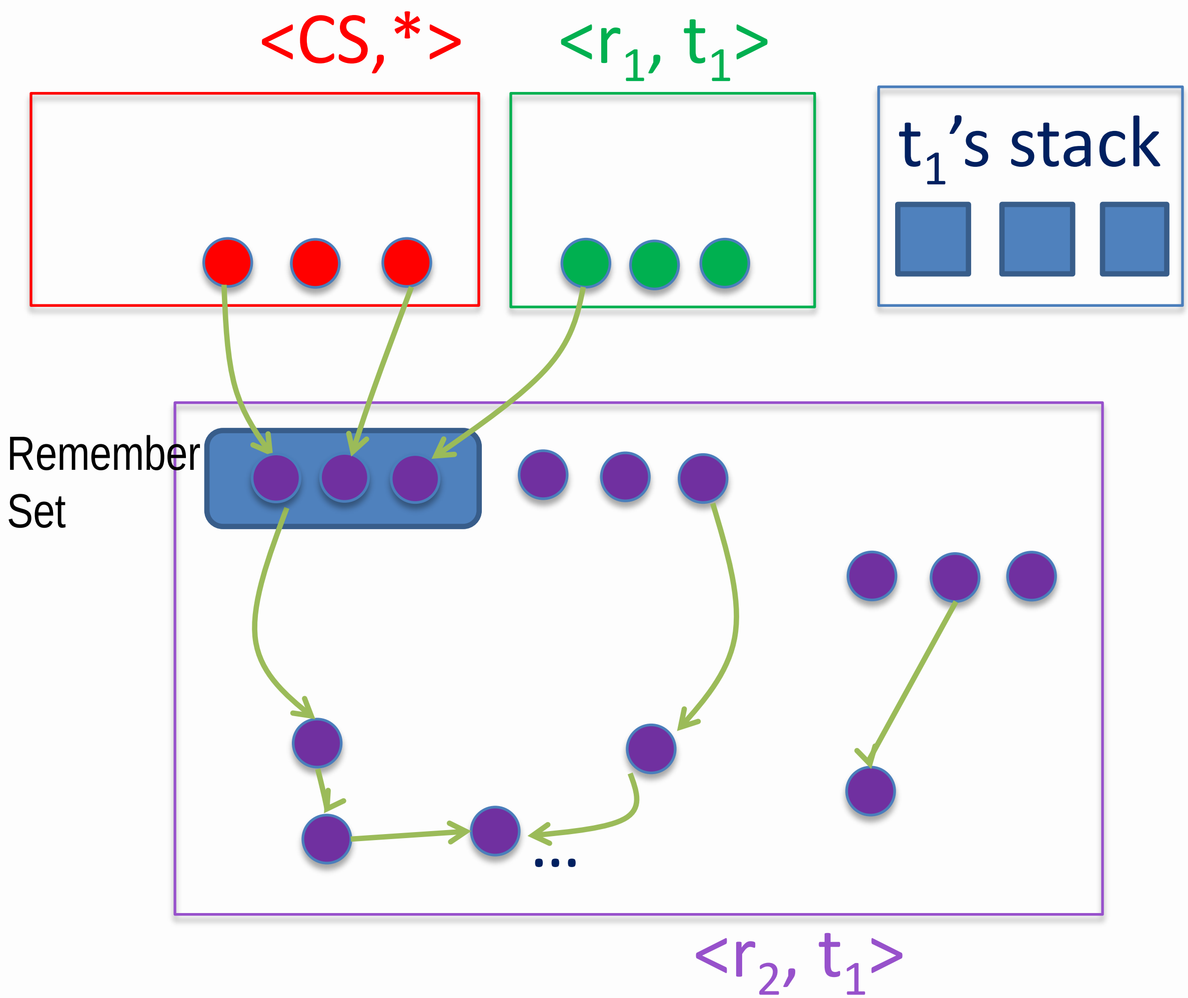
Region Deallocation

epoch_end()



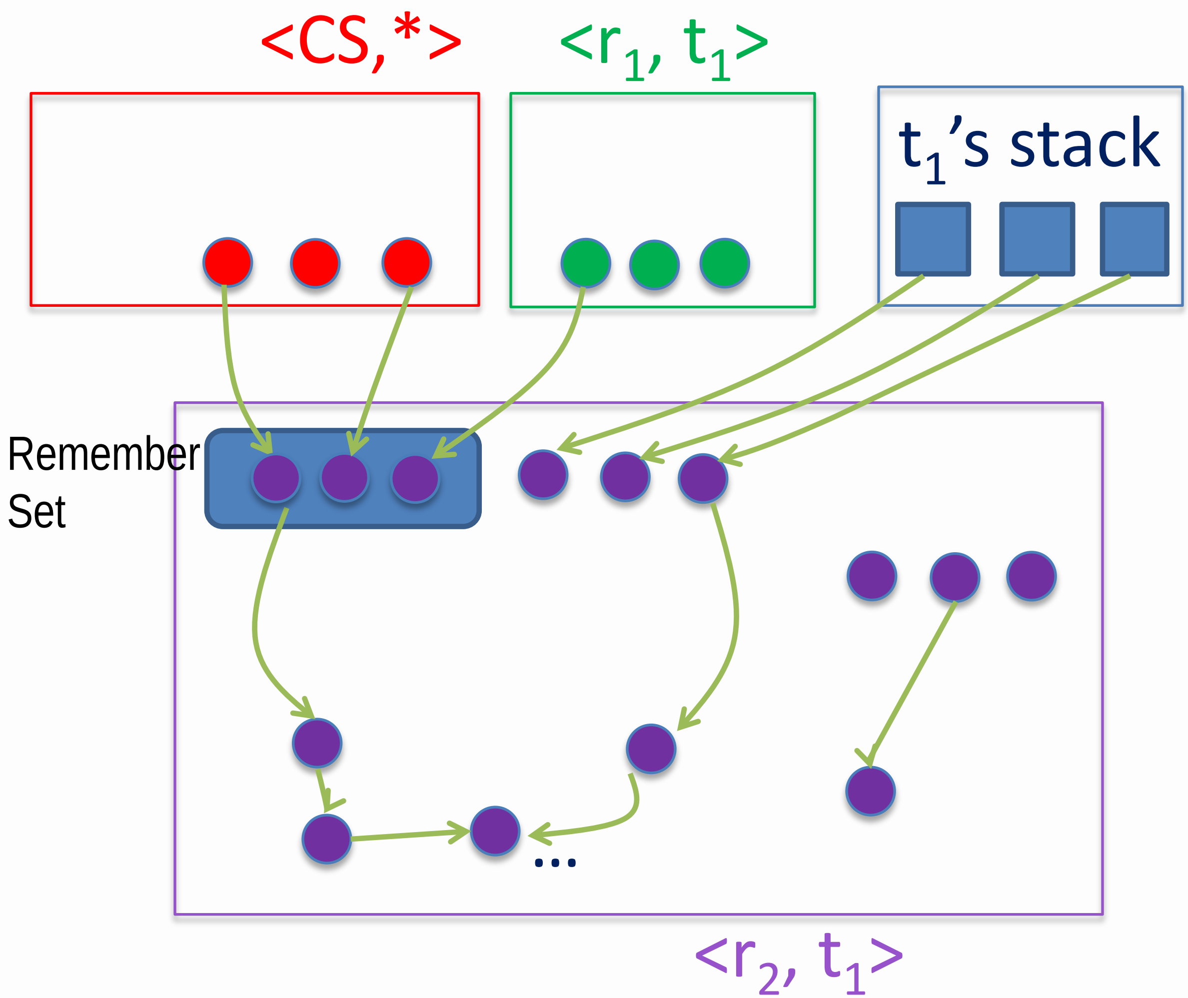
Region Deallocation

epoch_end()



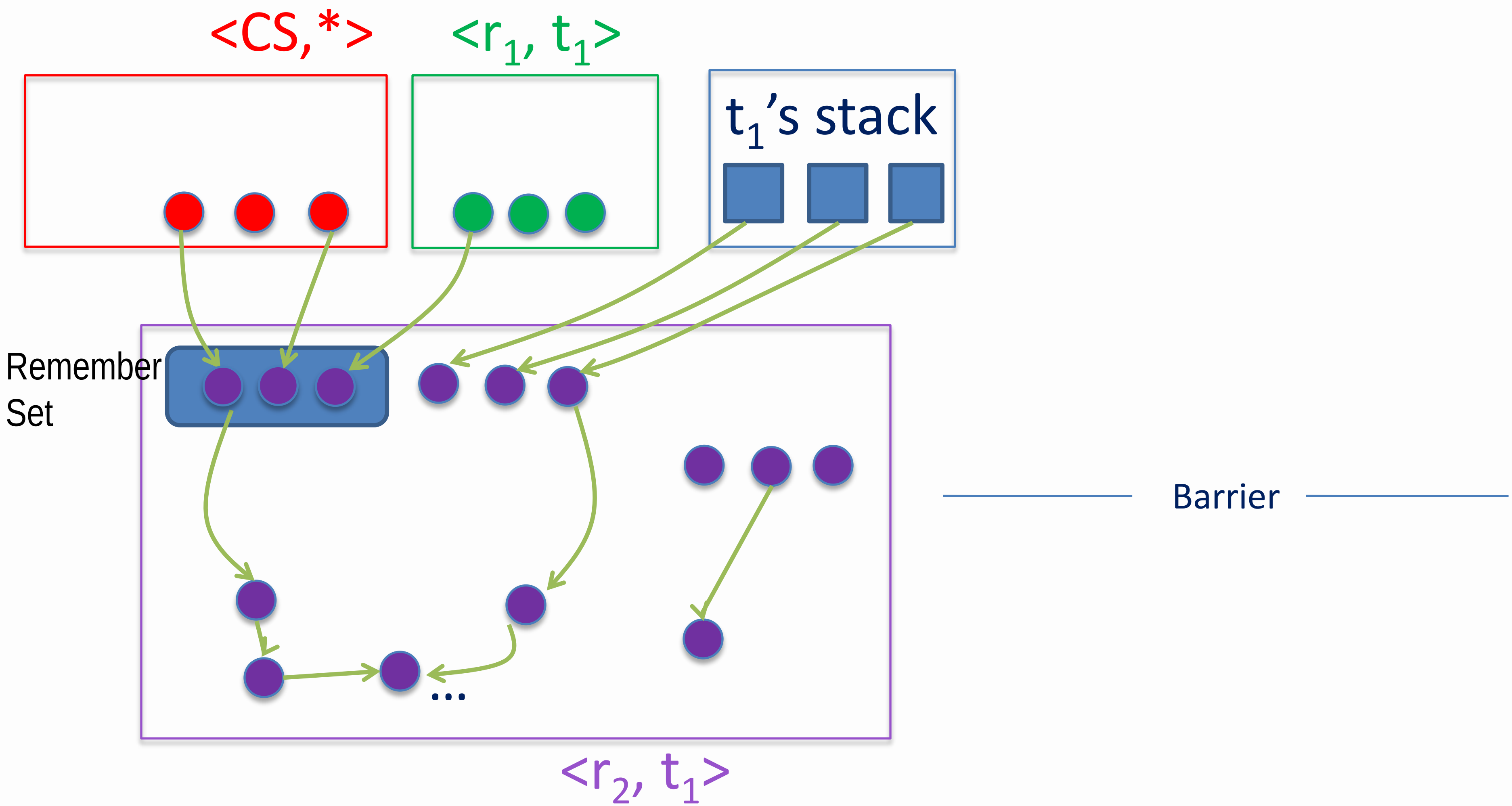
Region Deallocation

epoch_end()

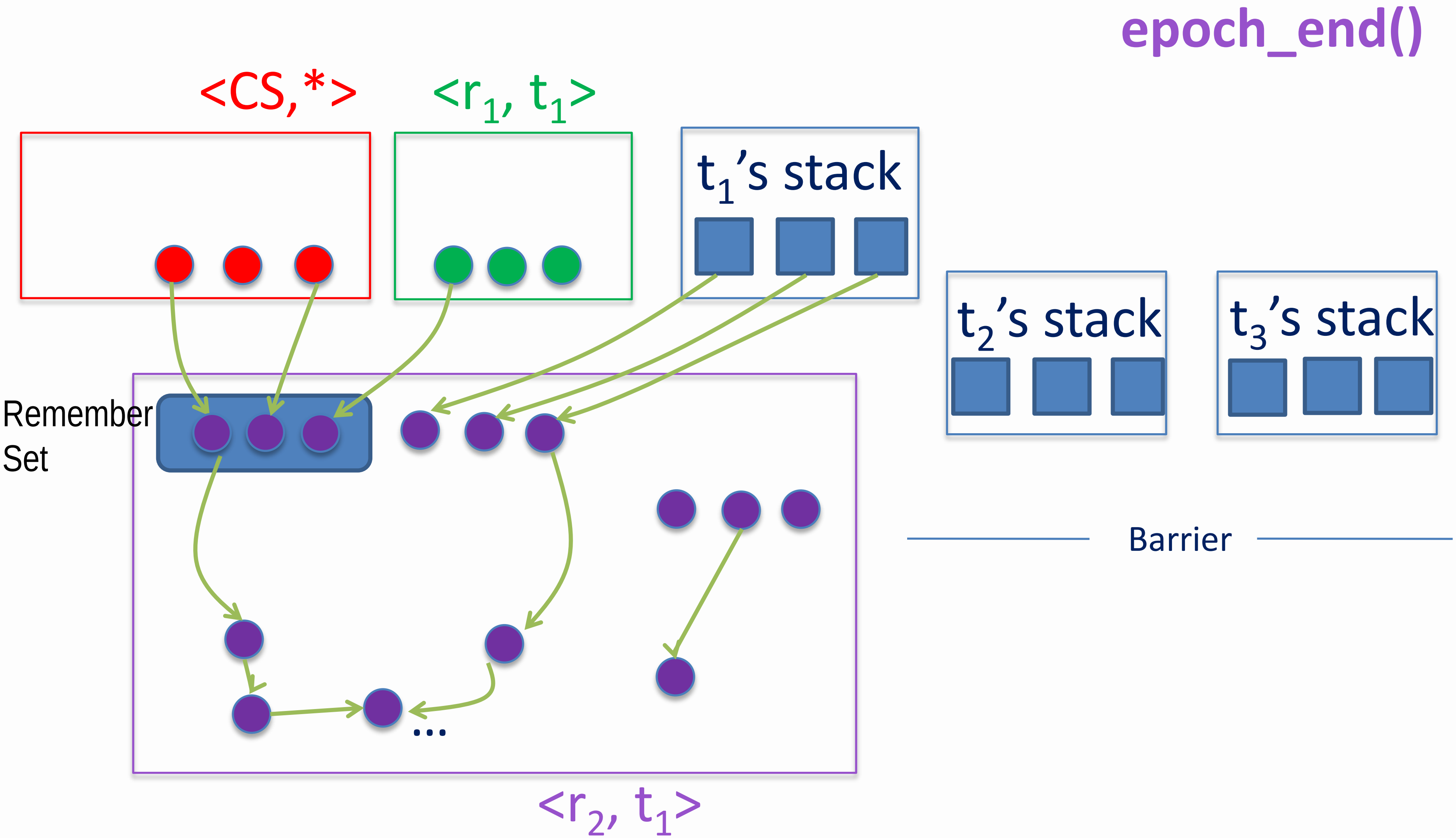


Region Deallocation

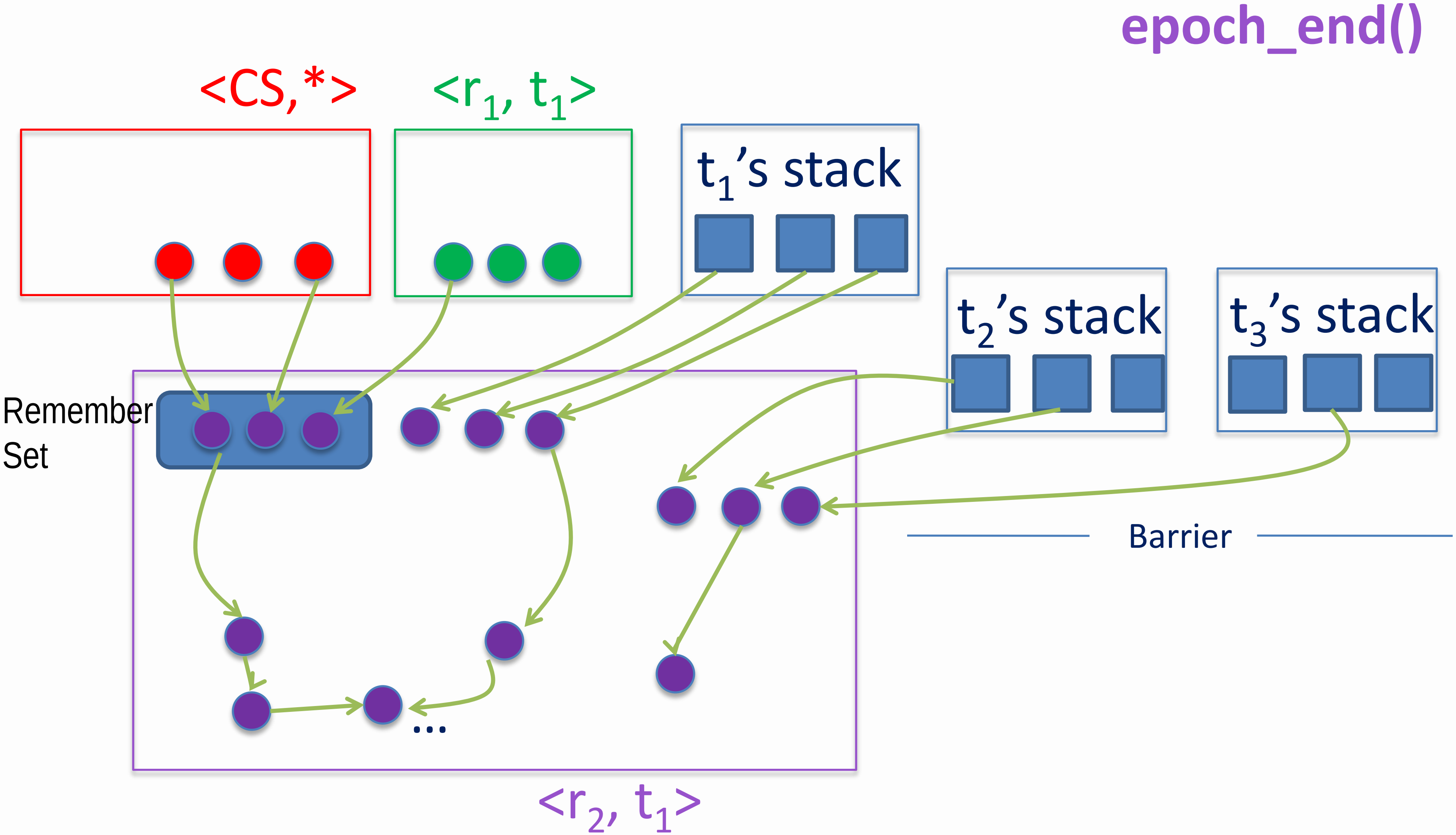
epoch_end()



Region Deallocation

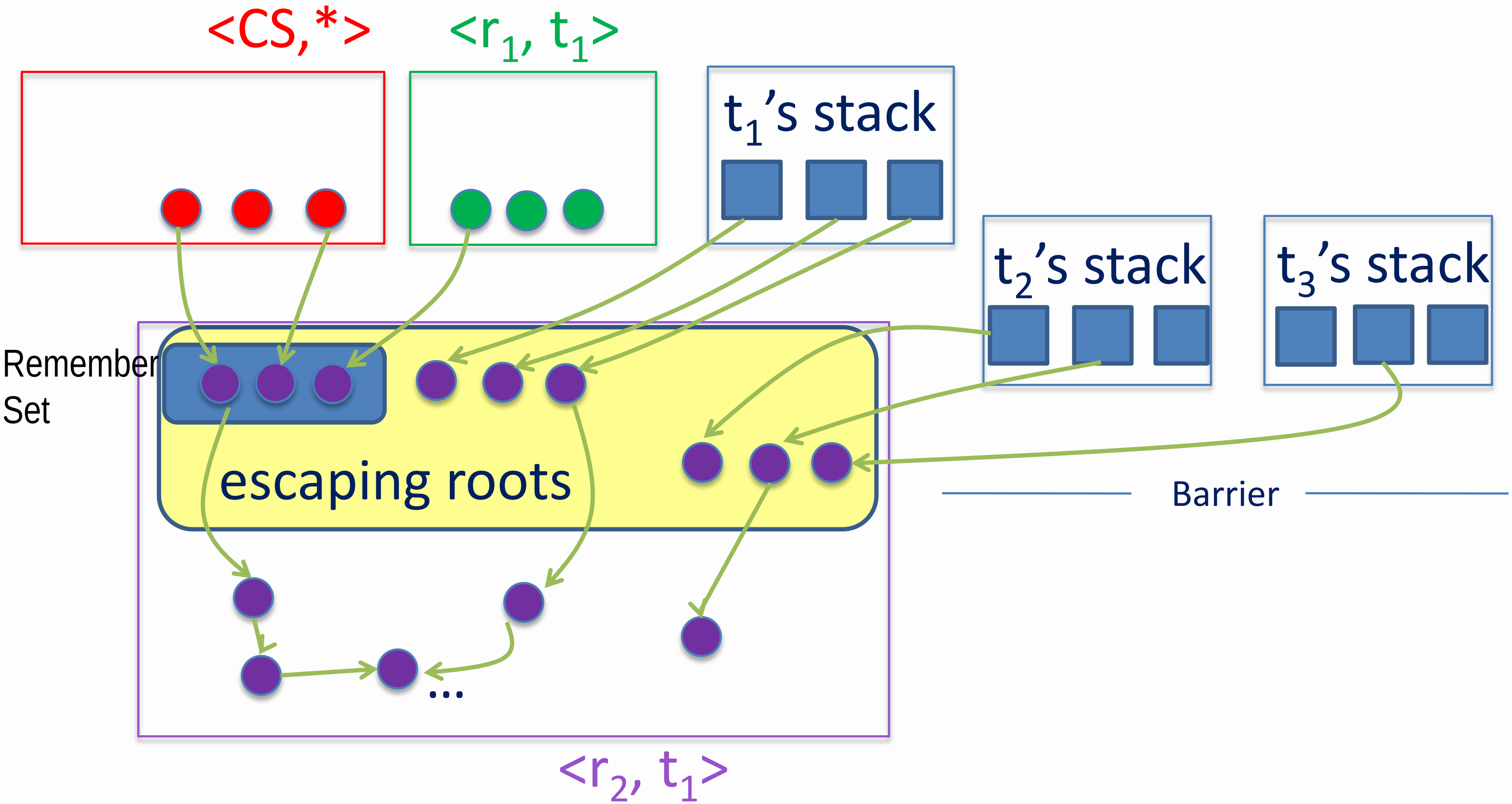


Region Deallocation

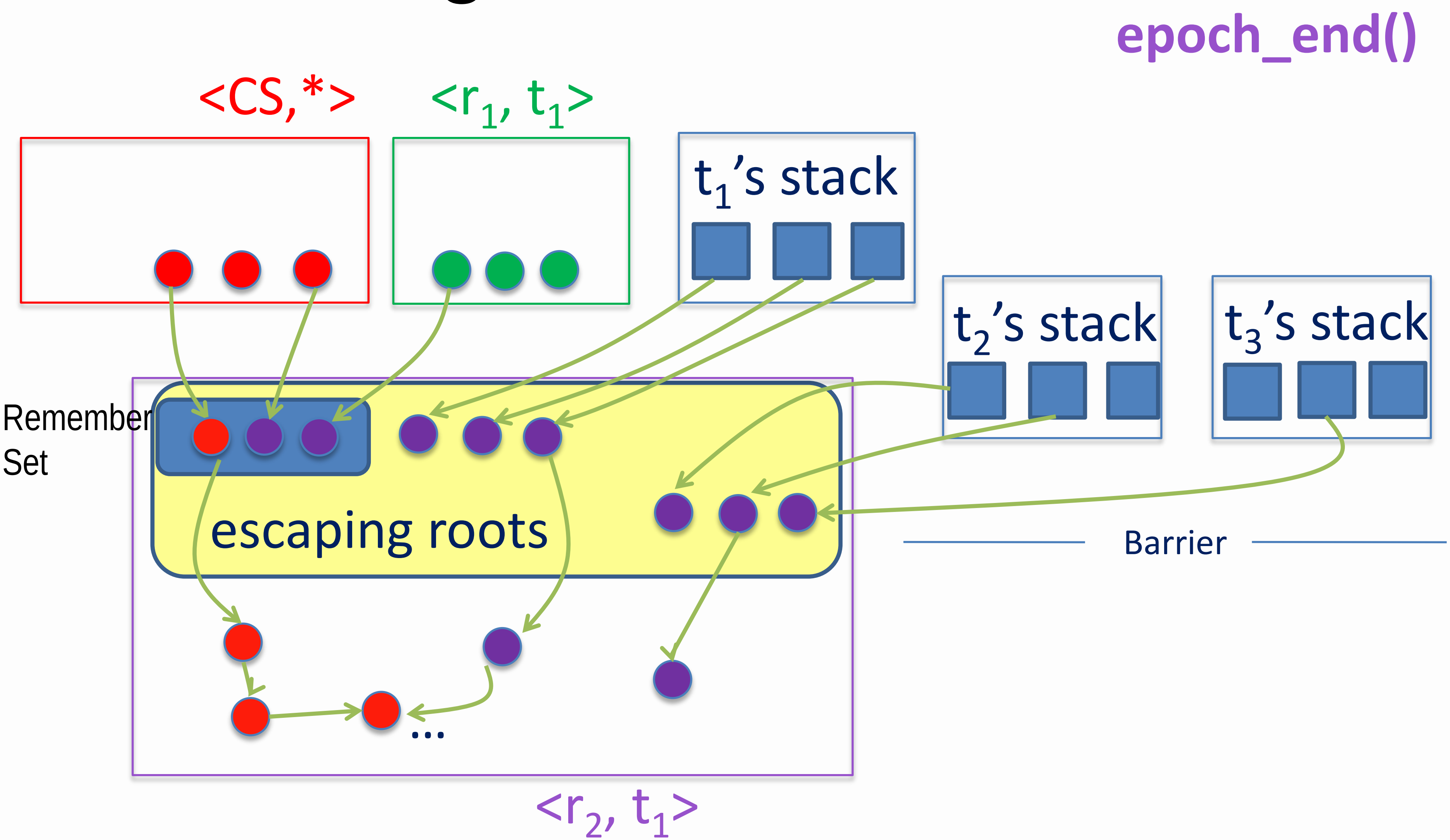


Region Deallocation

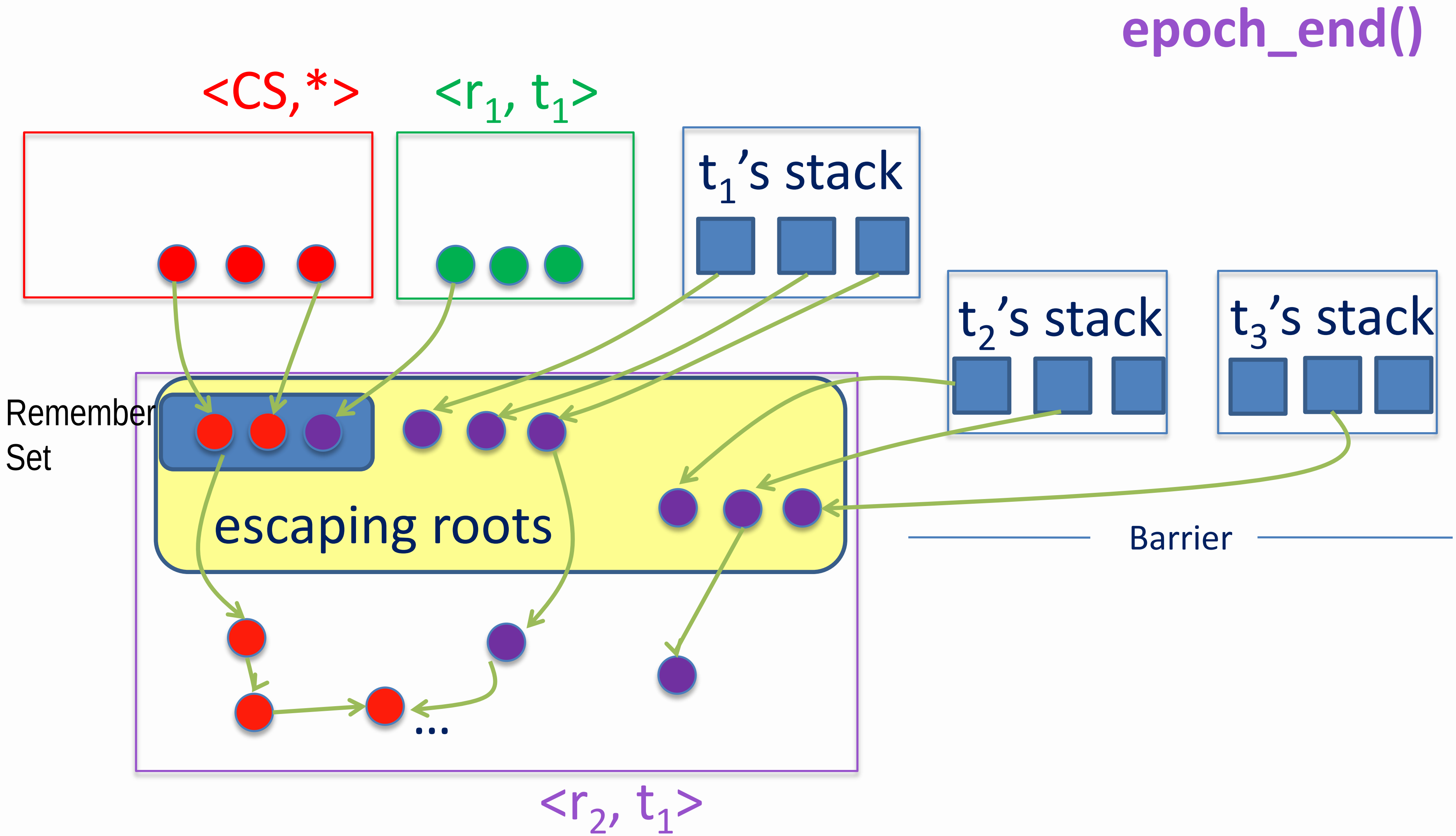
epoch_end()



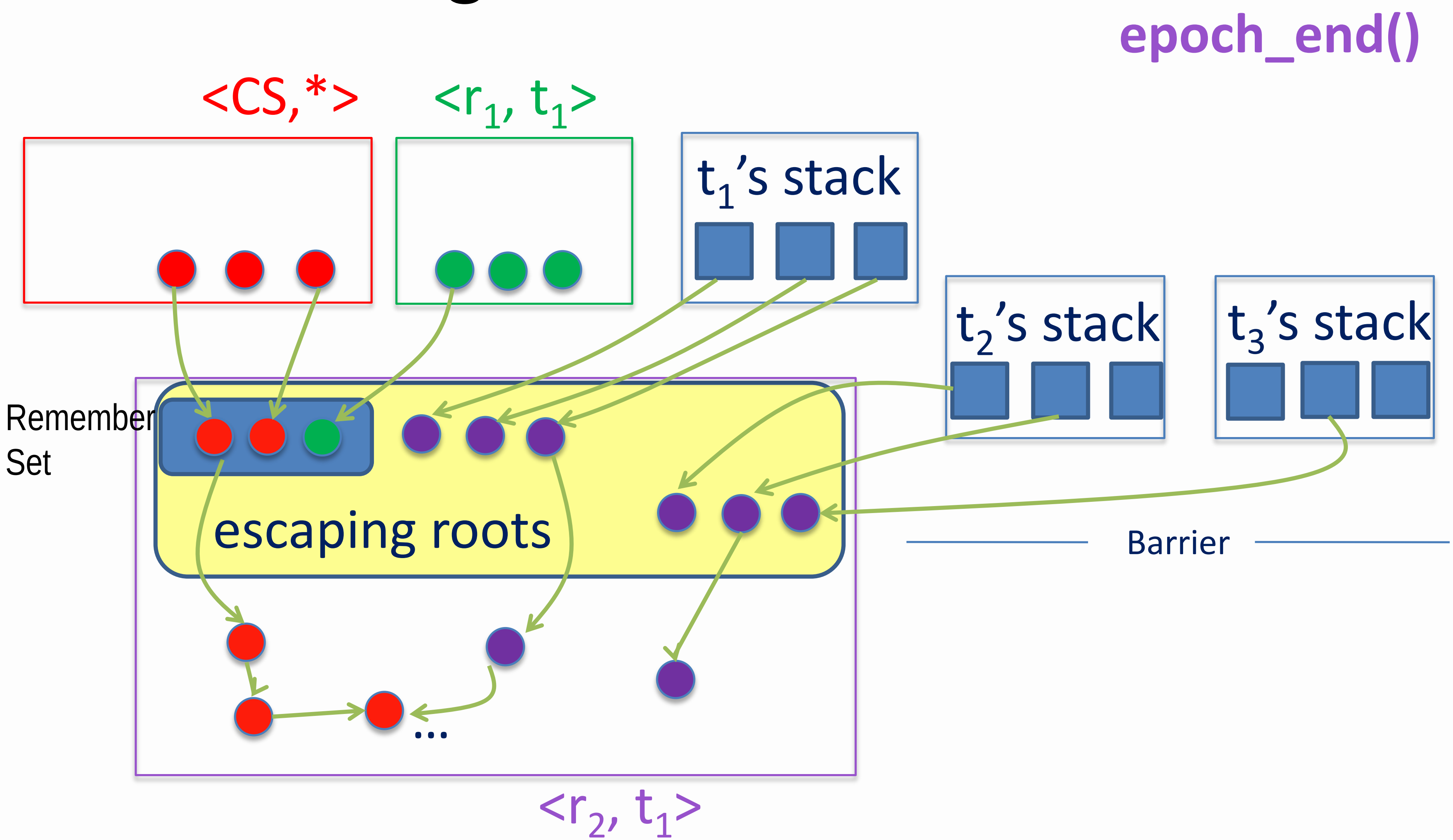
Region Deallocation



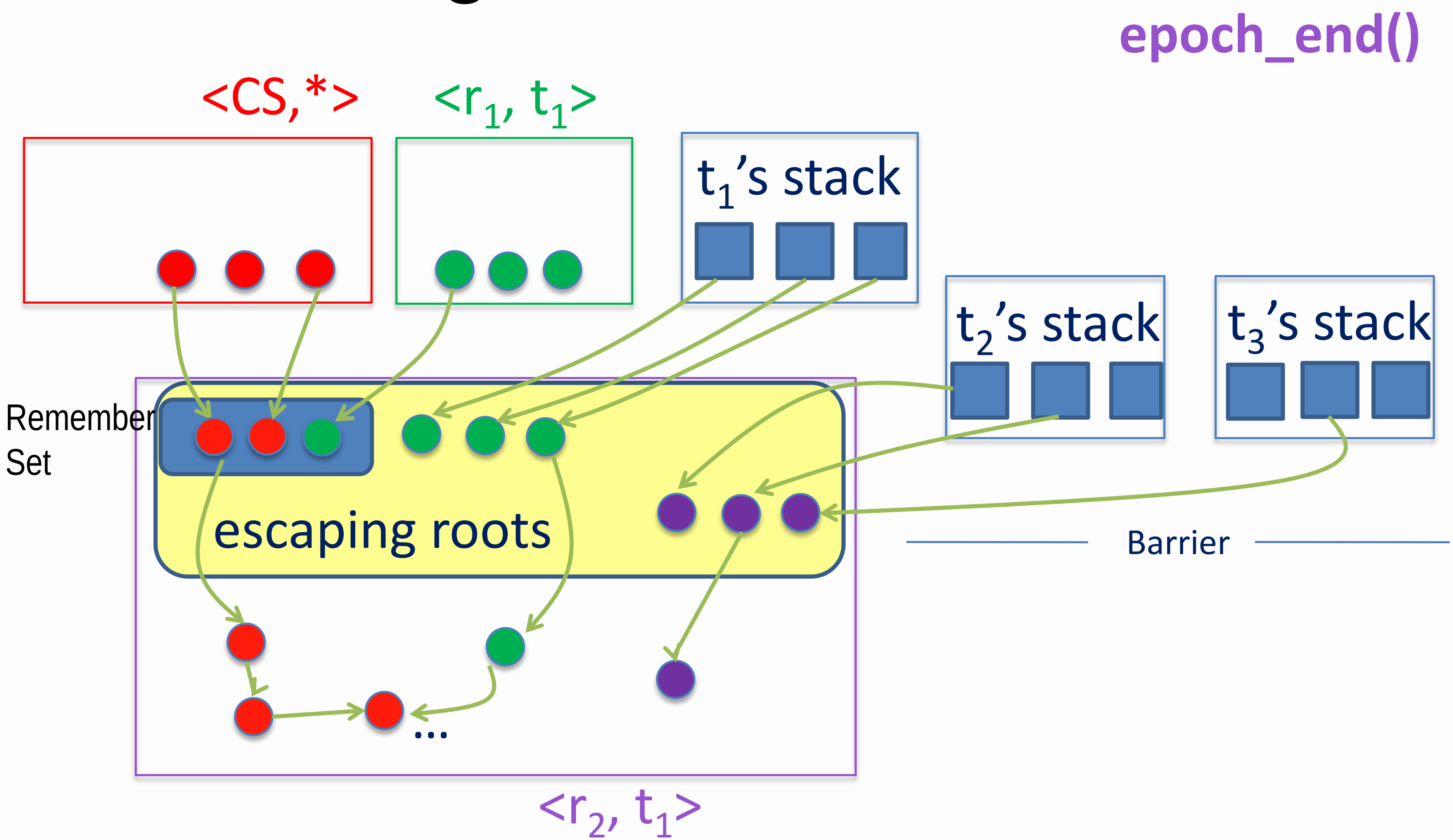
Region Deallocation



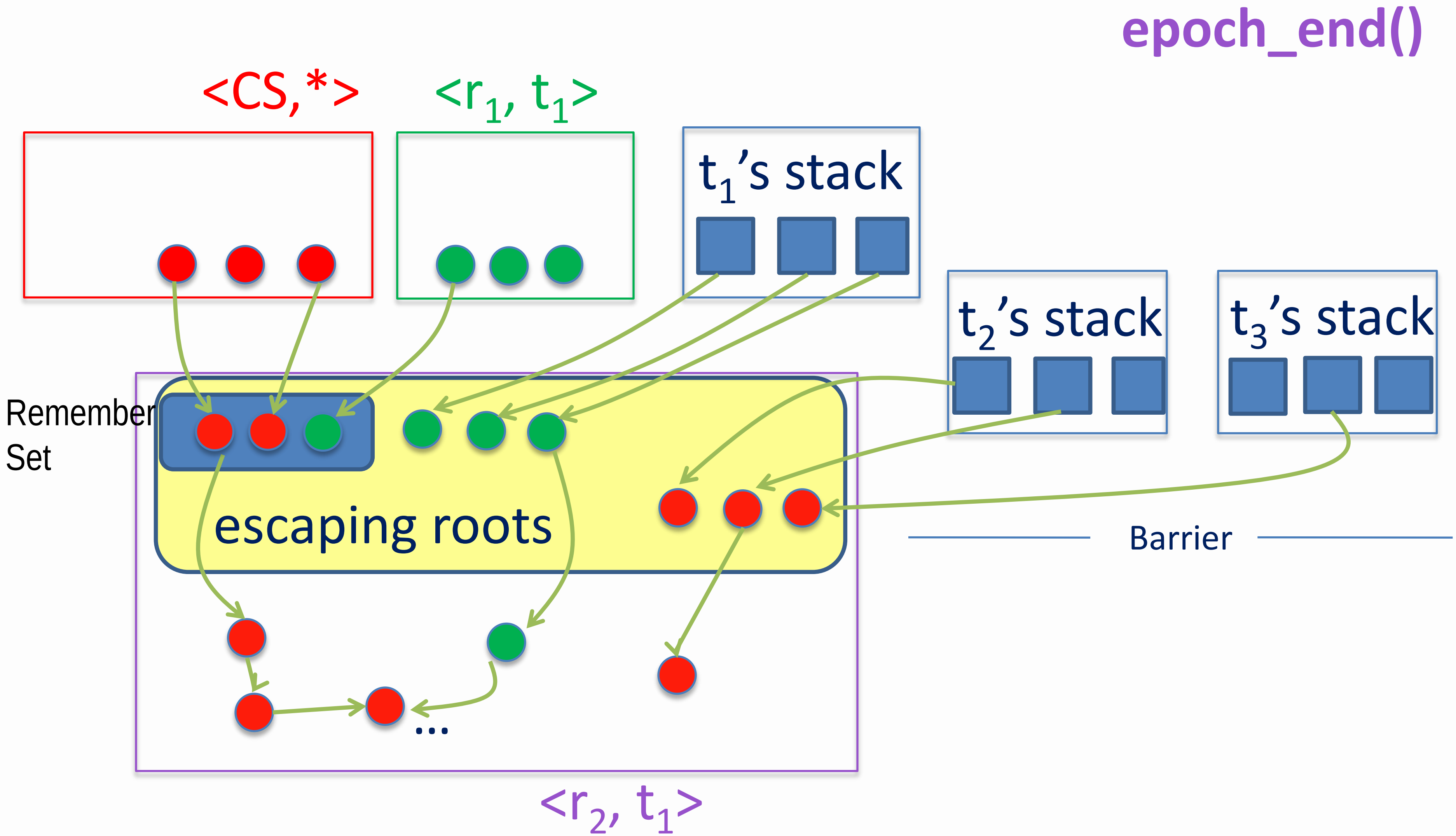
Region Deallocation



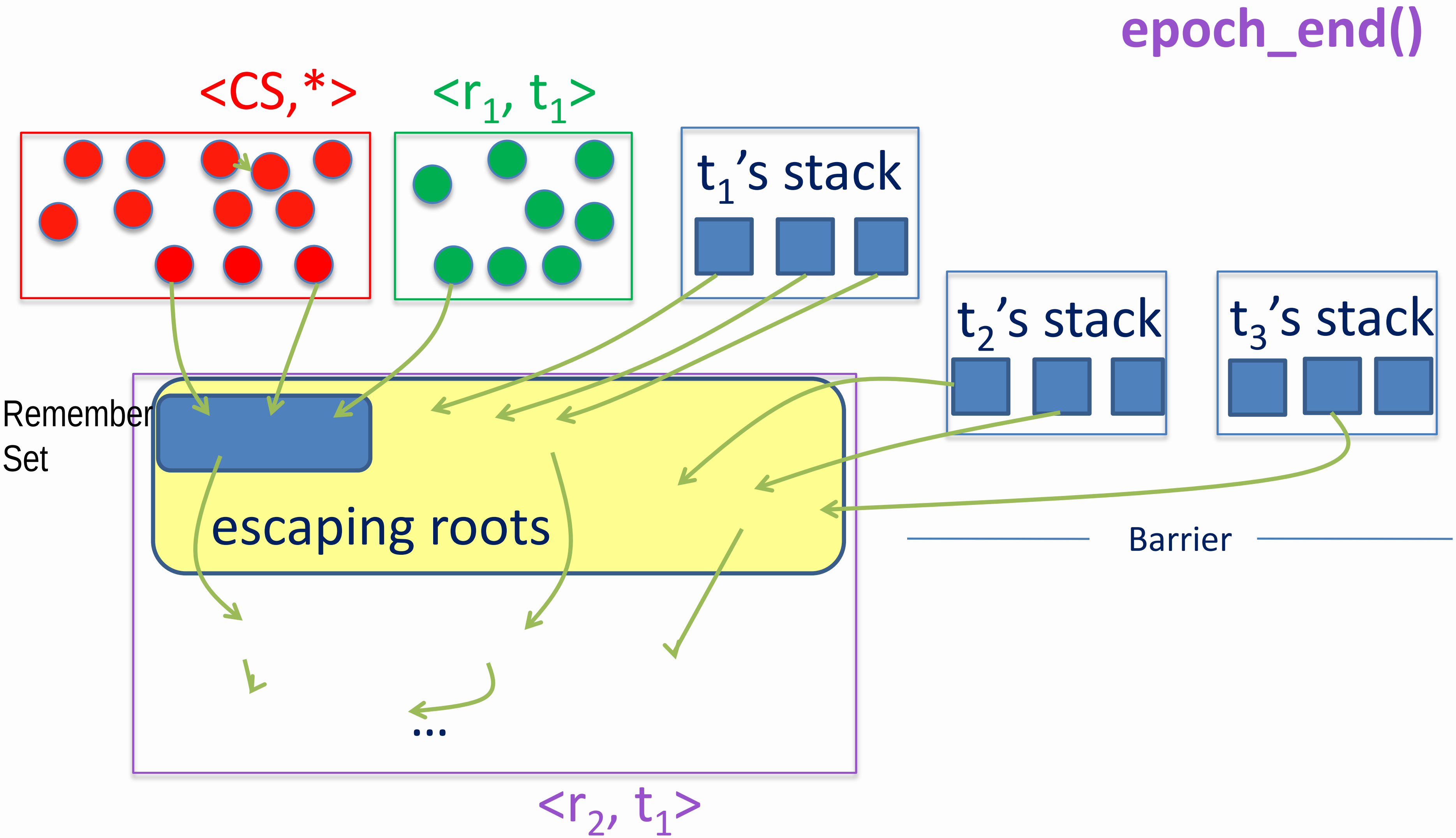
Region Deallocation



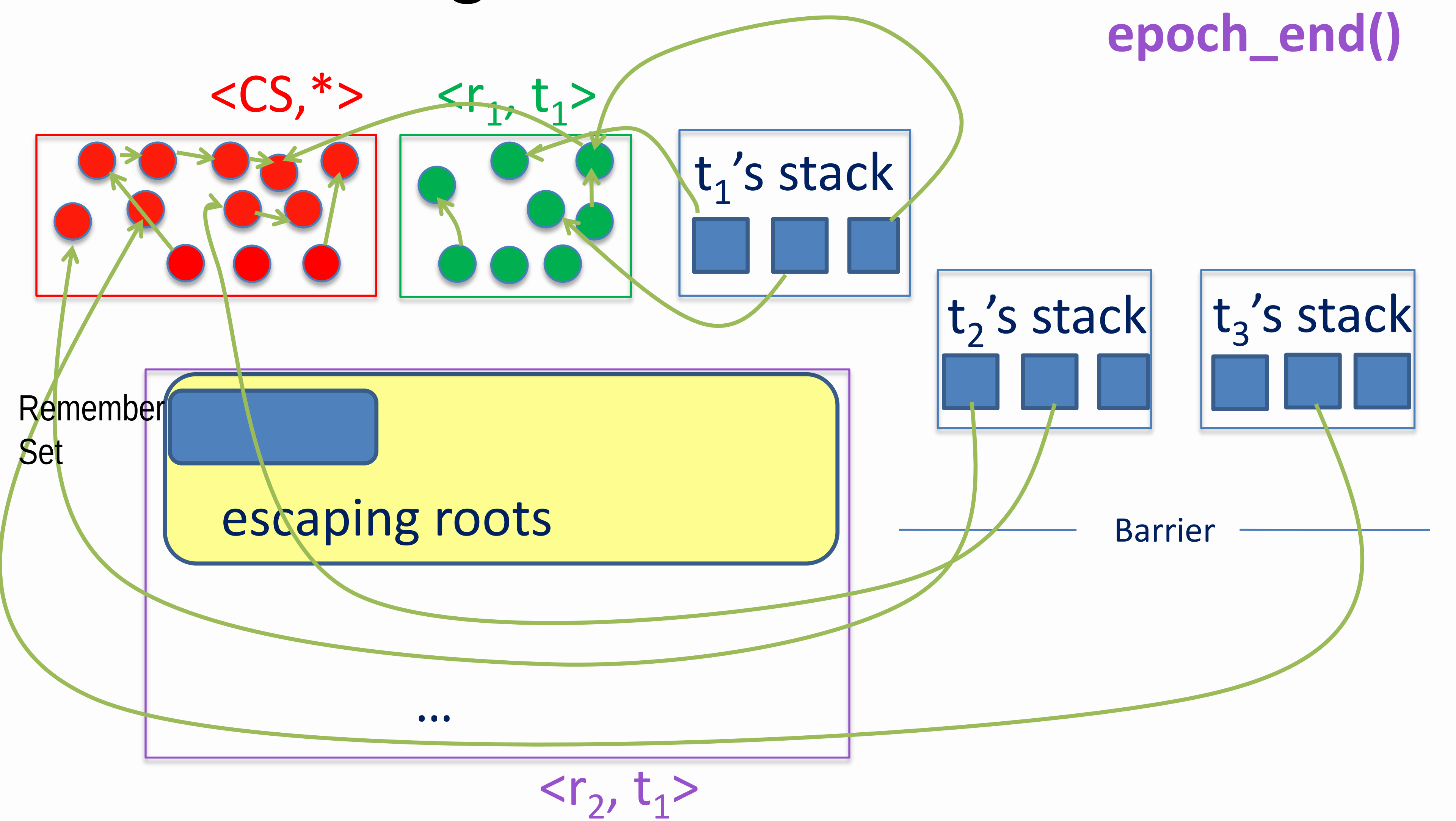
Region Deallocation



Region Deallocation

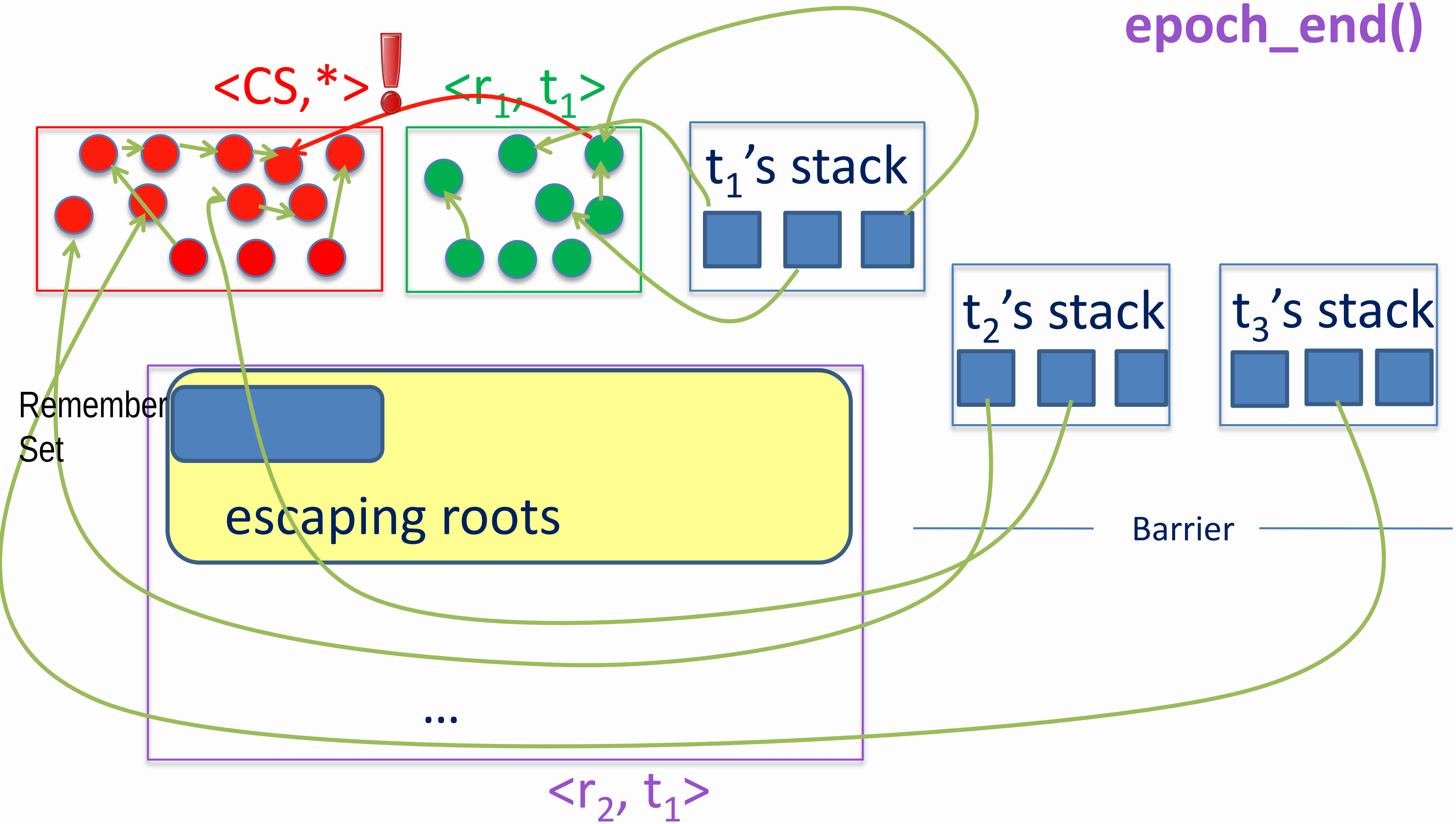


Region Deallocation

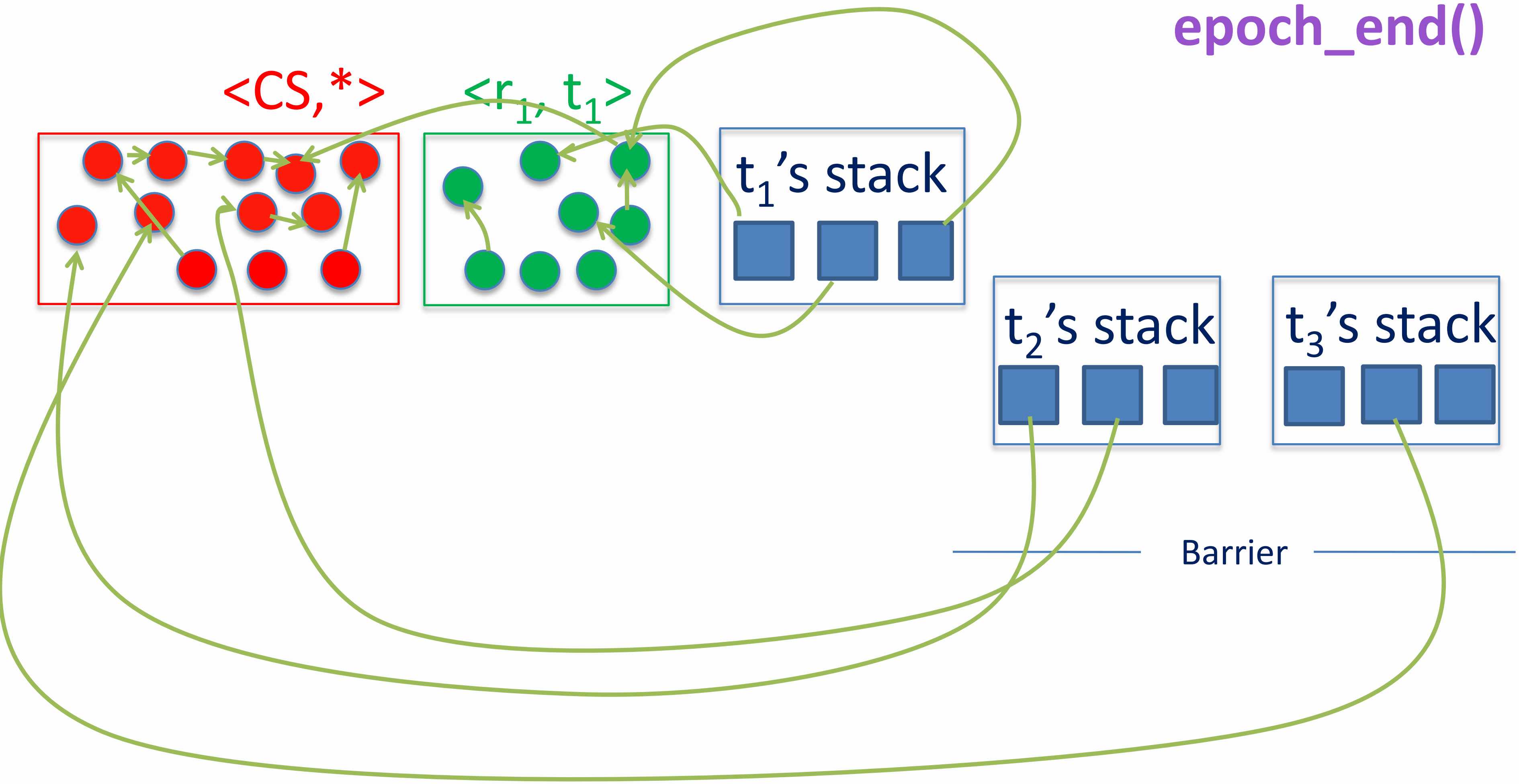


Region Deallocation

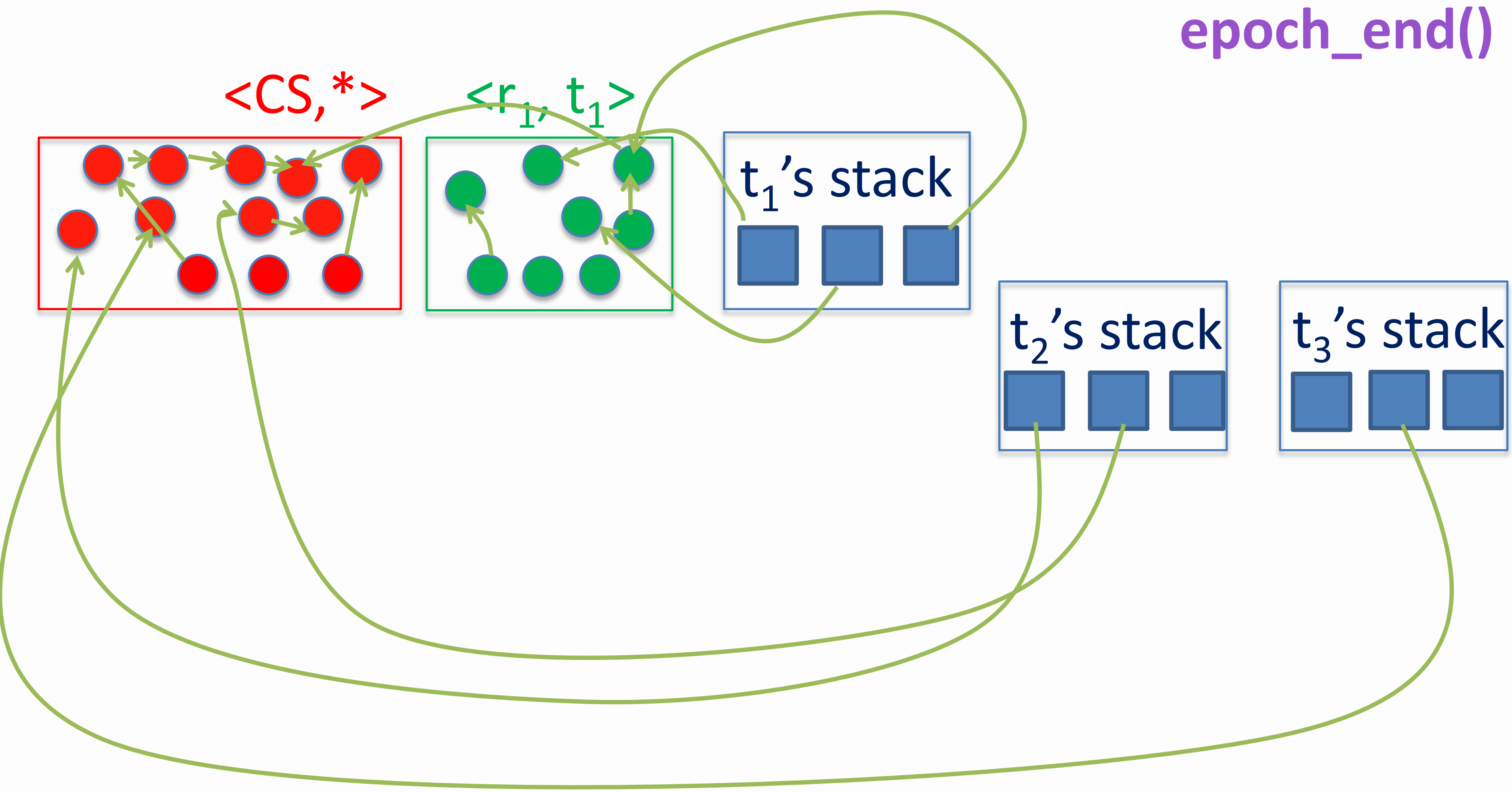
epoch_end()



Region Deallocation

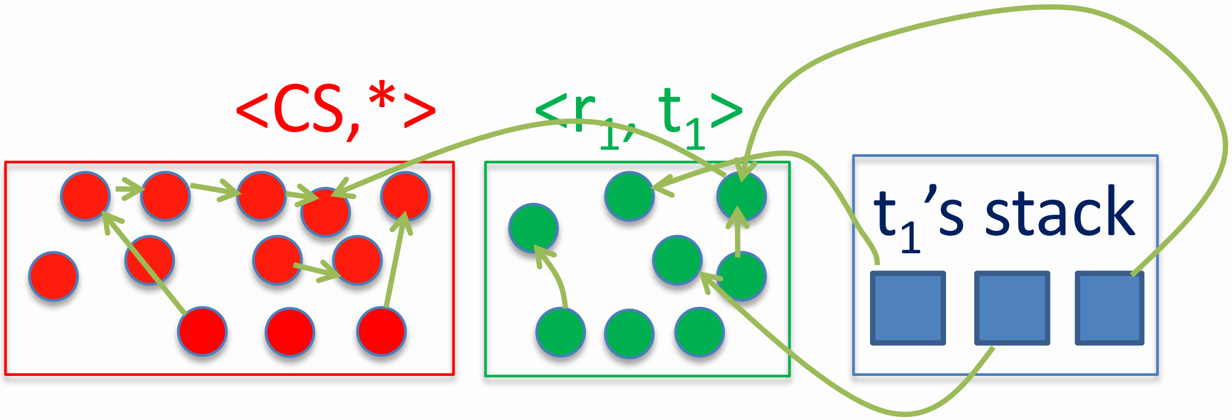


Region Deallocation

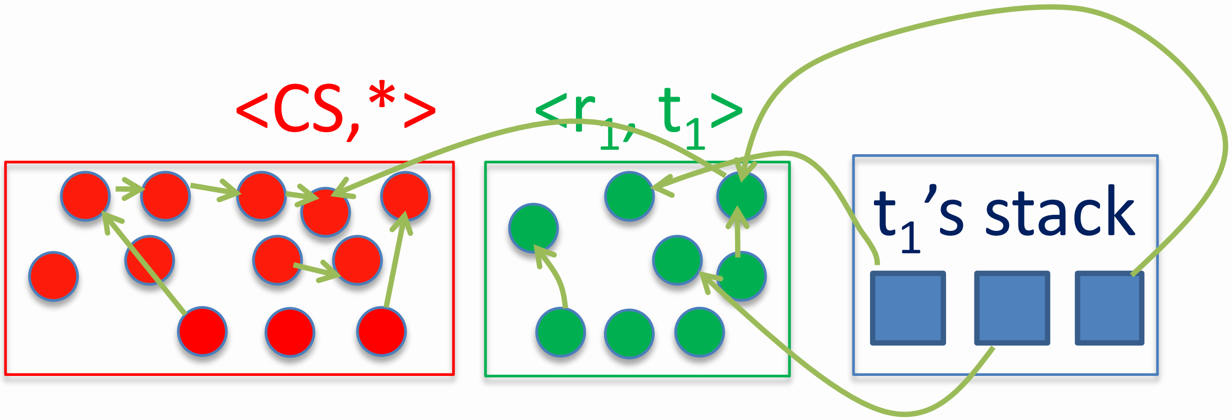


Region Deallocation

epoch_end()



Region Deallocation

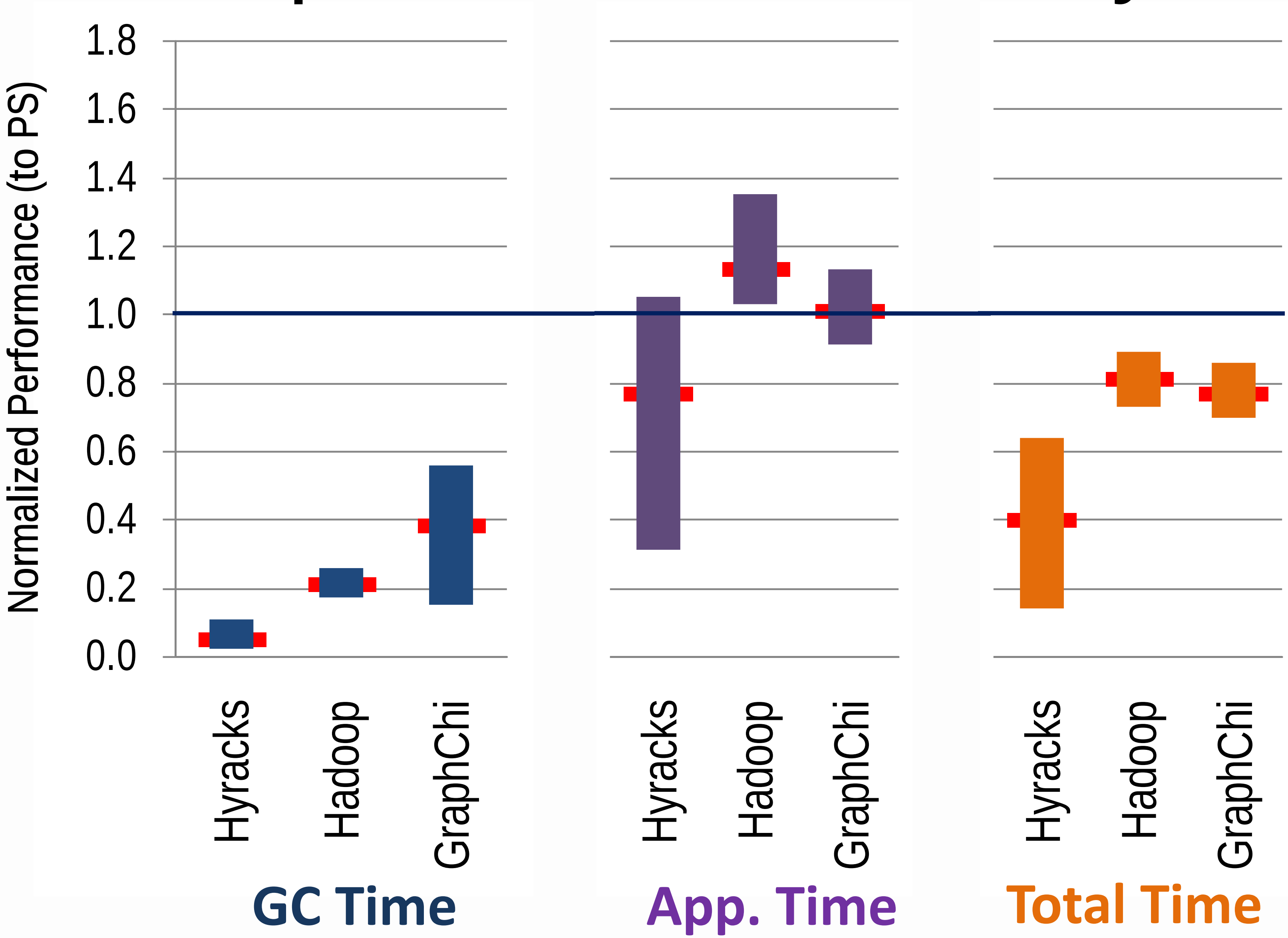


Evaluations

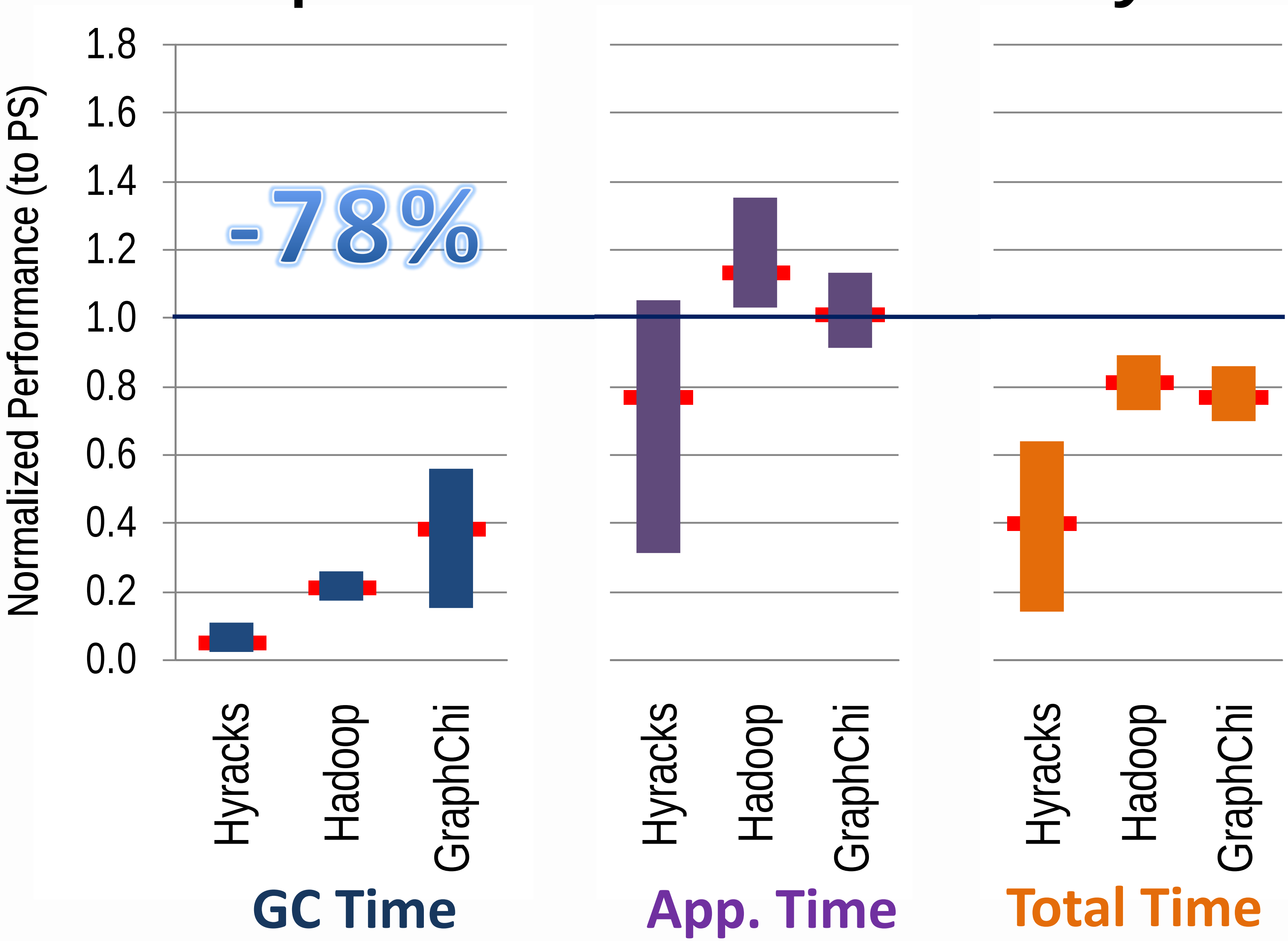
3 real-world systems, 9 applications:

- Hadoop
  - Popular distributed MapReduce implementation
- Hyracks [Borkar et al., ICDE'11]
  - Data-parallel platform to run data-intensive jobs on a cluster of shared-nothing machines
- GraphChi [Kyrola et al., OSDI'12]
  - High-performance graph analytical framework for a single machine

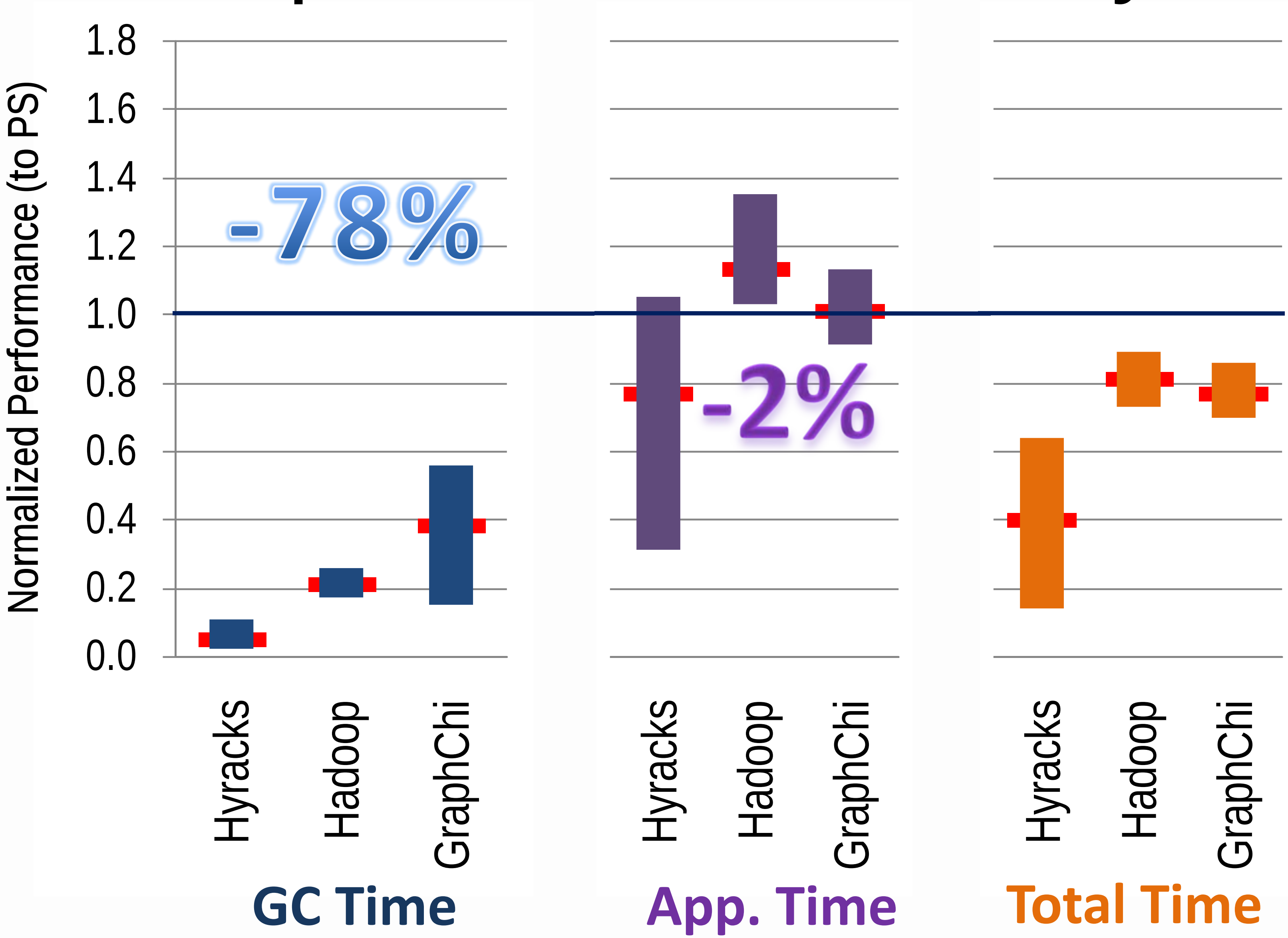
Improvement Summary



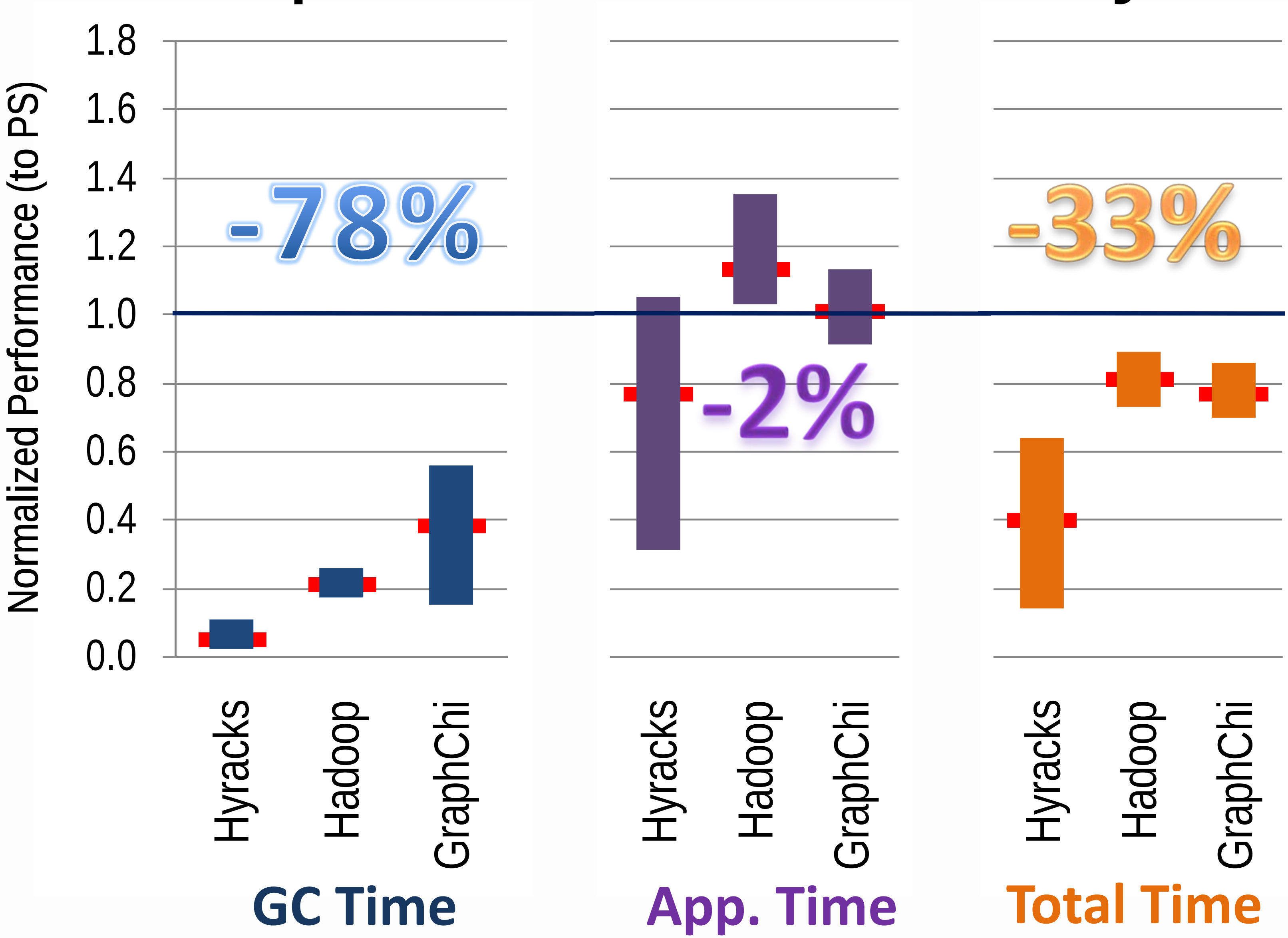
Improvement Summary



Improvement Summary



Improvement Summary



More Data in the Paper

- Statistics on Yak's heap
- Overhead breakdown
  - Write barrier cost
  - Extra space cost

Conclusion

- *Goal:* Reduce memory management cost in Big Data systems
- *Solution:* **Yak**, the first hybrid GC
  - **33%** execution time savings
    78% GC time reduction
  - Requires almost *zero* user effort

Thank You!