Short-term Memory for Self-collecting Mutators

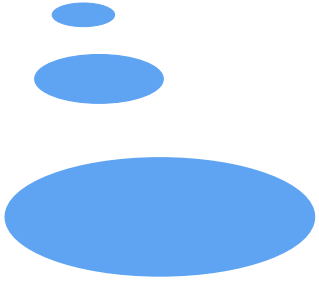
Martin Aigner, **Andreas Haas**, Christoph M. Kirsch, Michael Lippautz,
Ana Sokolova, Stephanie Stroka, Andreas Unterweger

University of Salzburg

International Symposium on Memory Management 2011
June 5, San Jose

Overview

Short-term Memory



new memory model
for heap management

Overview

Short-term Memory

for

Self-collecting Mutators

new memory model for heap management

new explicit heap management system

Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused

Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused

Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused

Idea

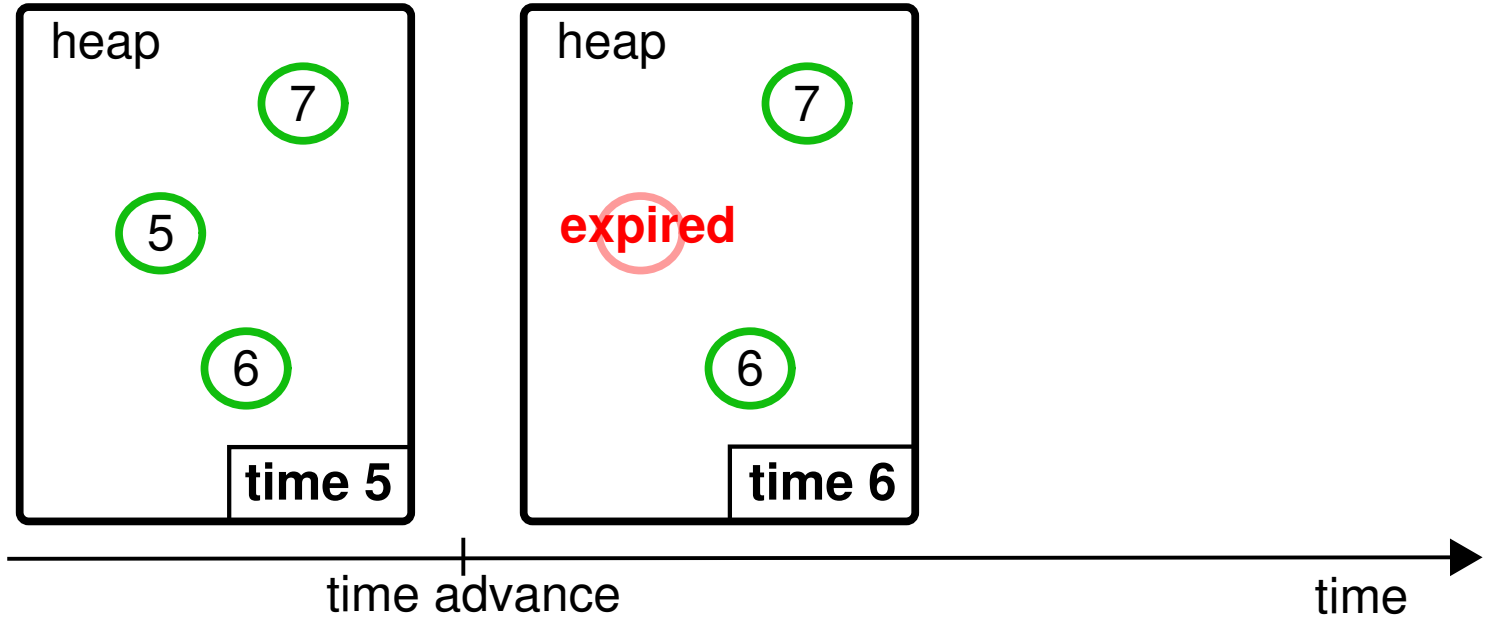
- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused

Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused

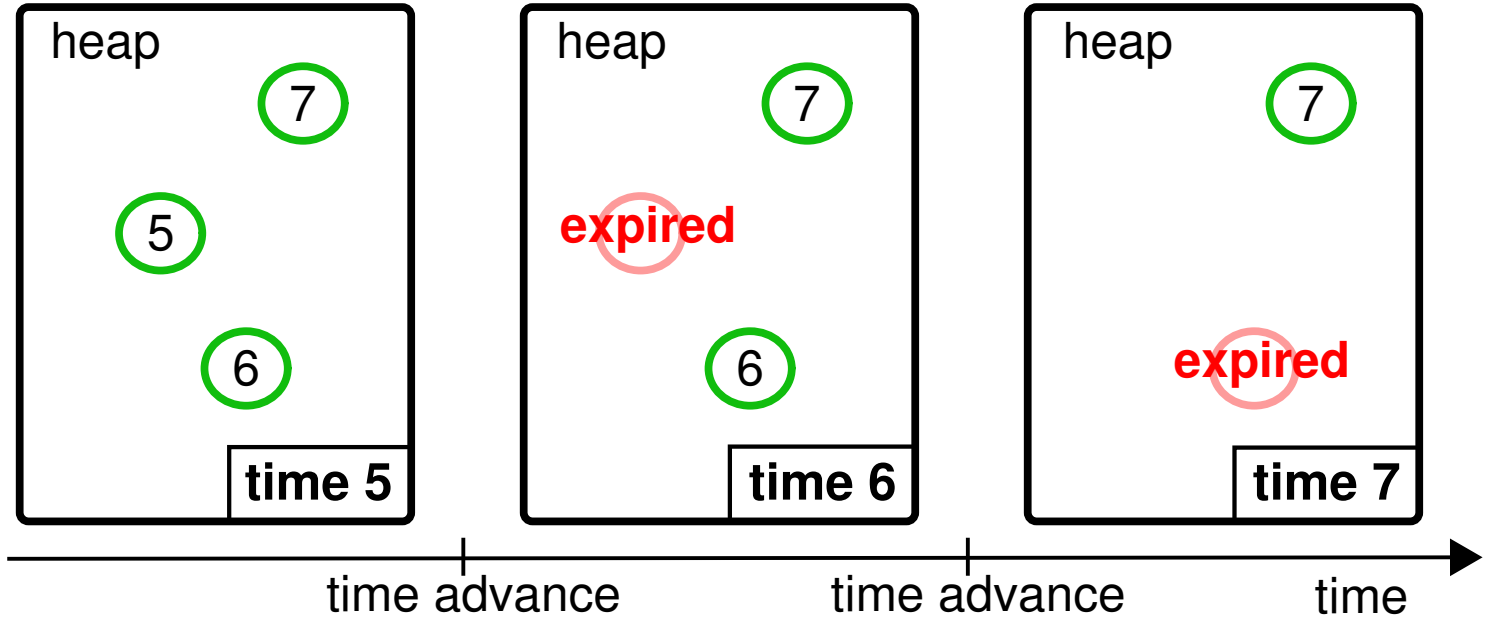
Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused



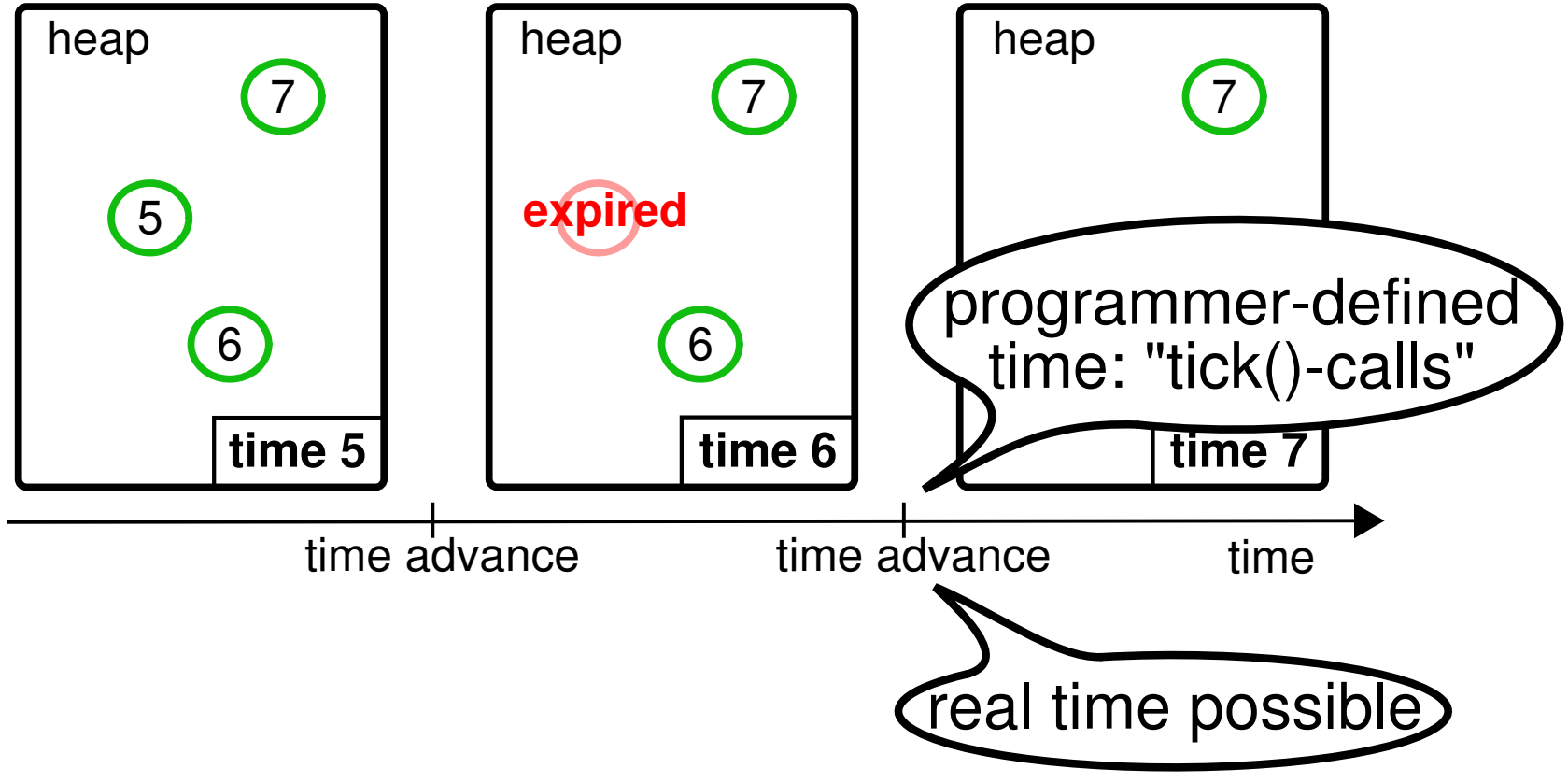
Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused



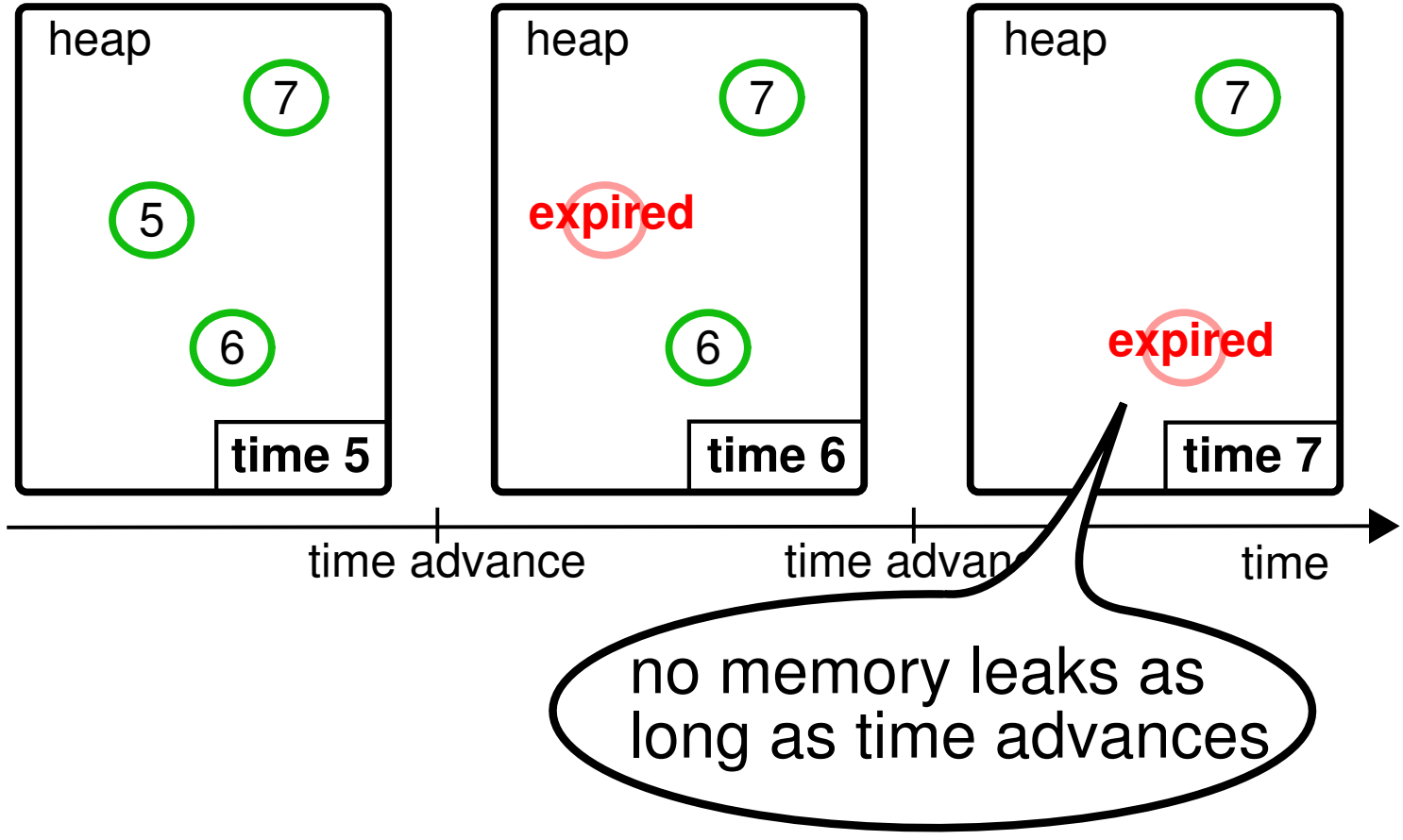
Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused



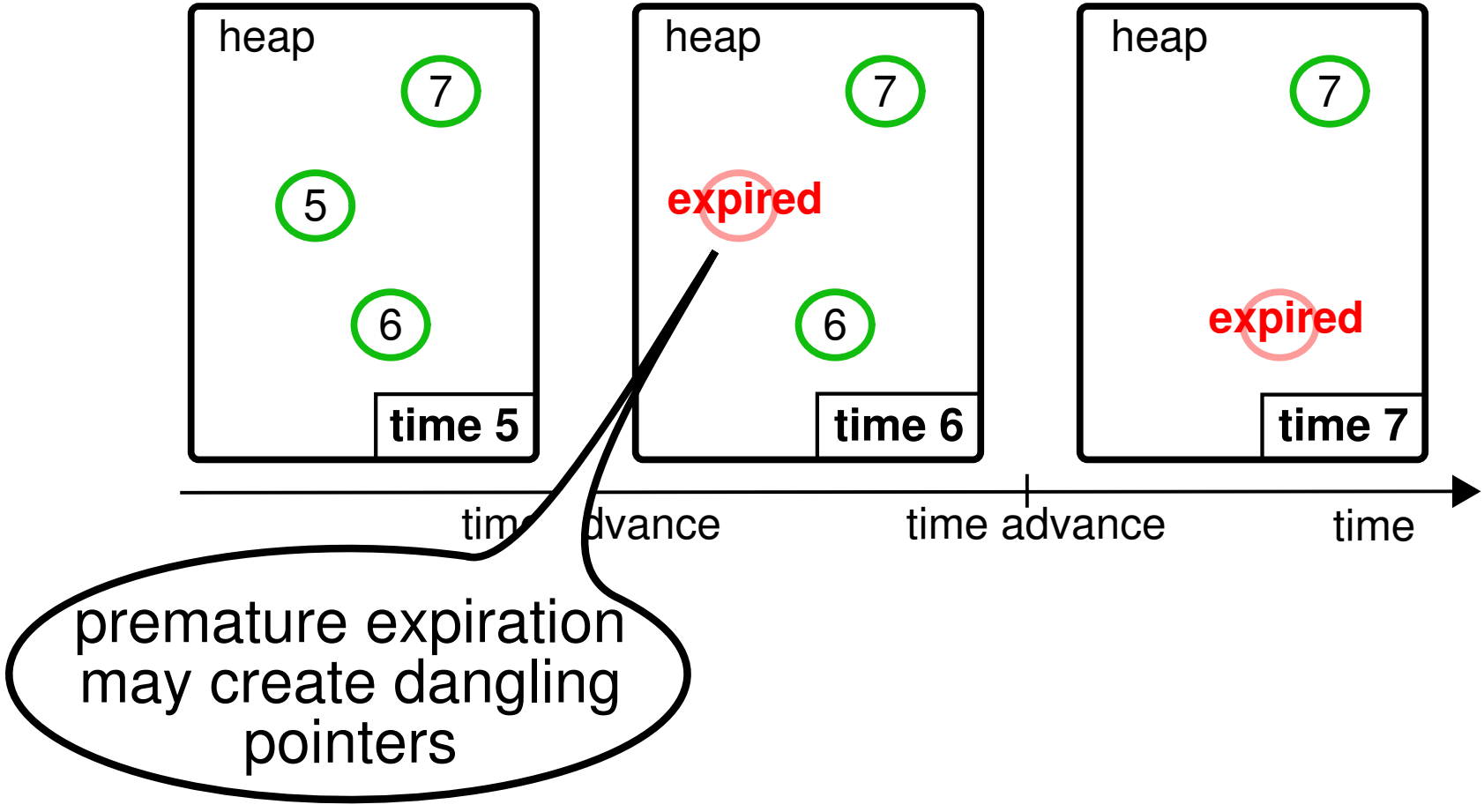
Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused



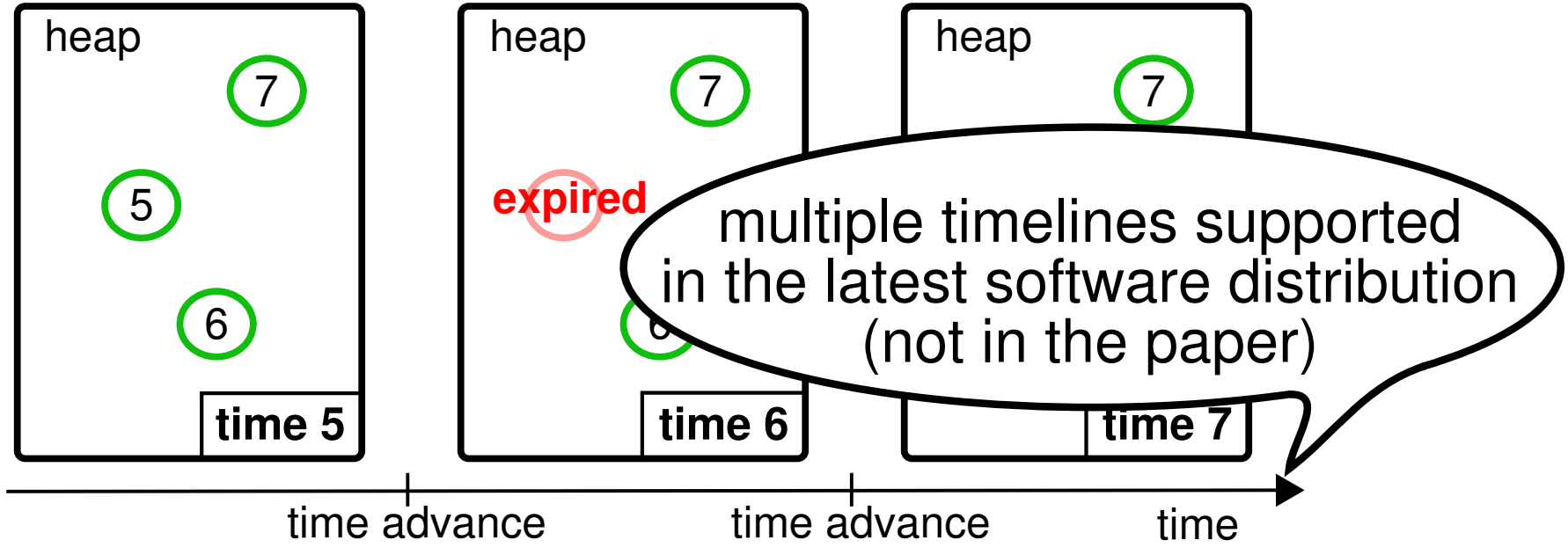
Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused



Idea

- Each memory object allocated in short-term memory has an **expiration date**
  - When an object **expires**, its memory may be reused



Refresh

- Extend the expiration date of an object by refreshing

Refresh

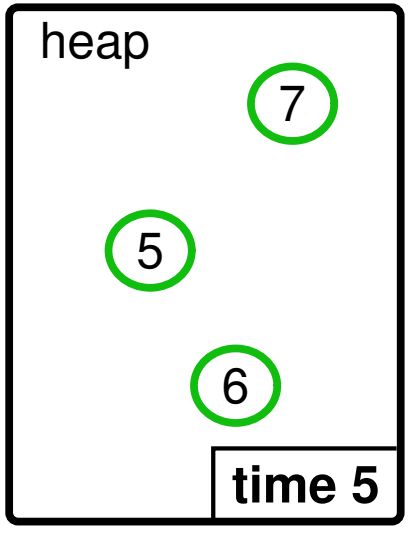
- Extend the expiration date of an object by refreshing

$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$

Refresh

- Extend the expiration date of an object by refreshing

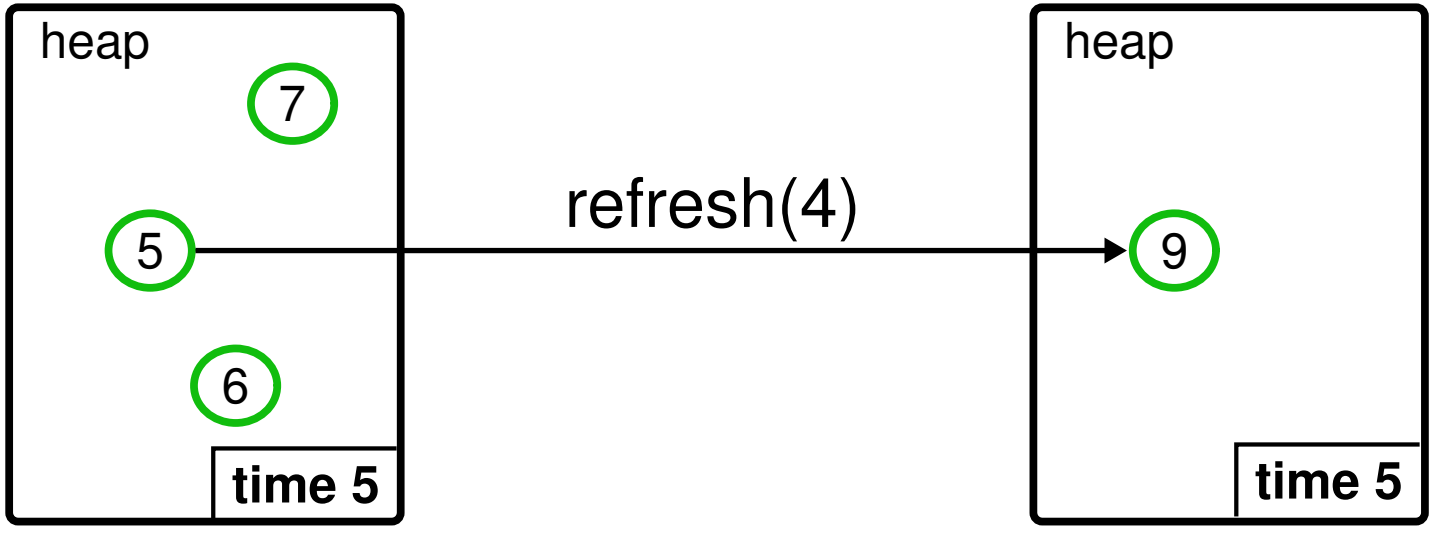
$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$



Refresh

- Extend the expiration date of an object by refreshing

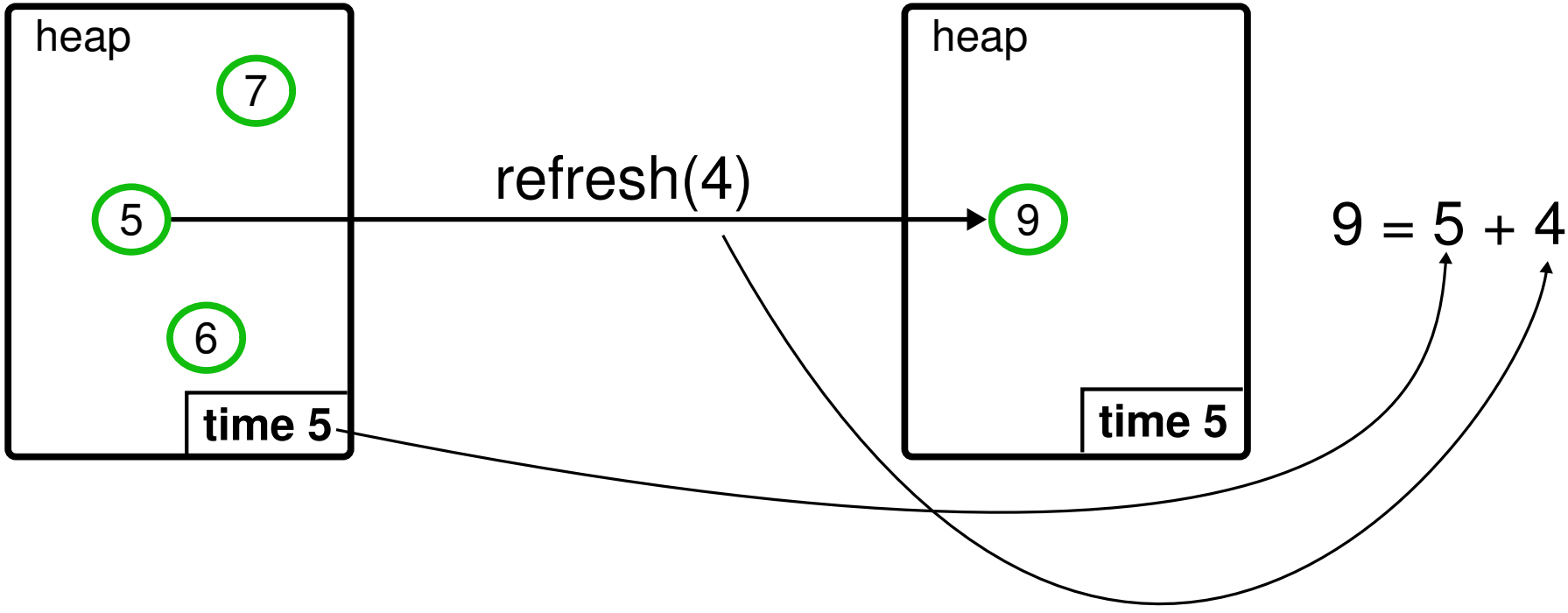
$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$



Refresh

- Extend the expiration date of an object by refreshing

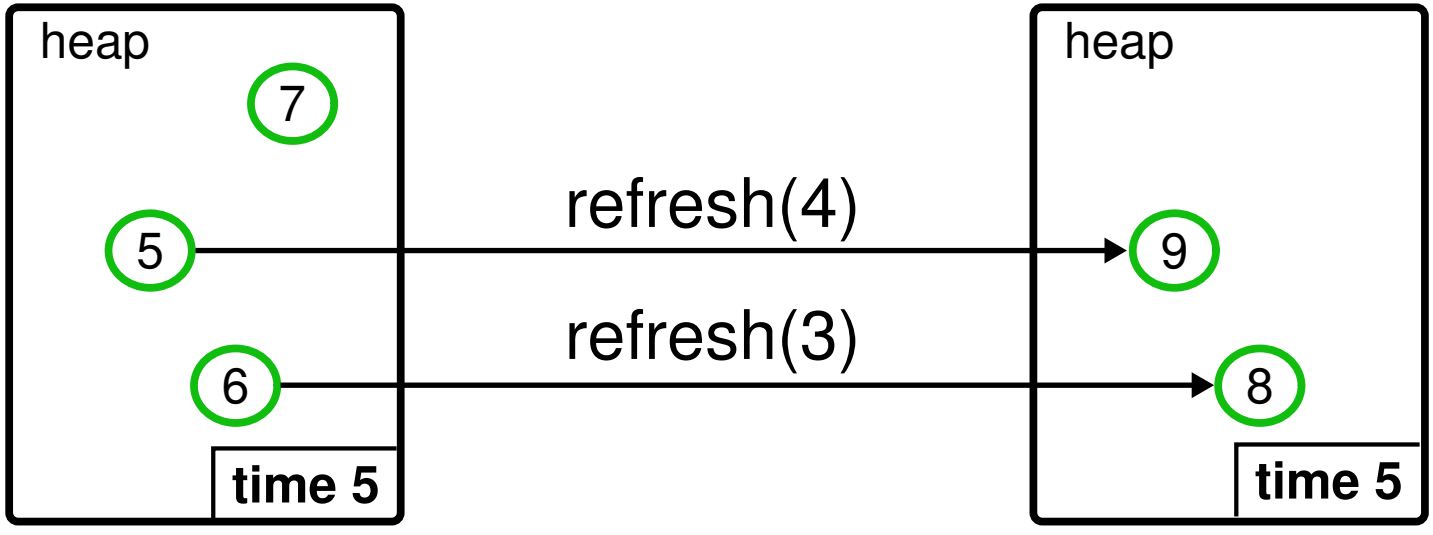
$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$



Refresh

- Extend the expiration date of an object by refreshing

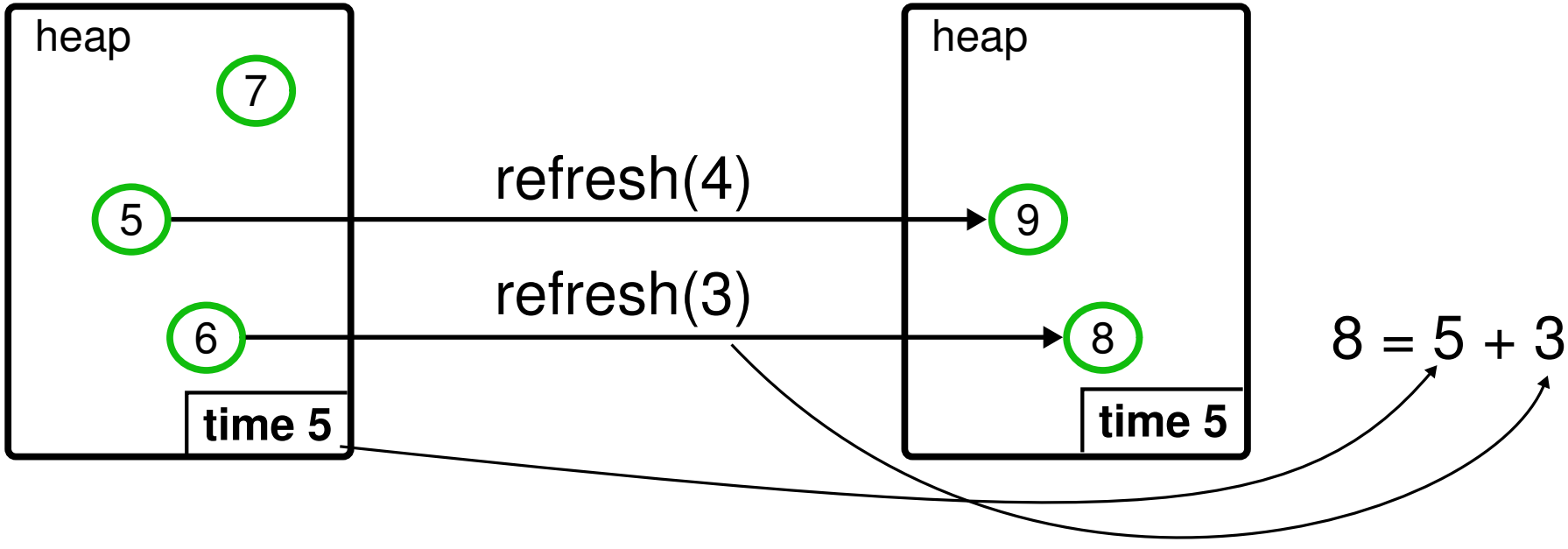
$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$



Refresh

- Extend the expiration date of an object by refreshing

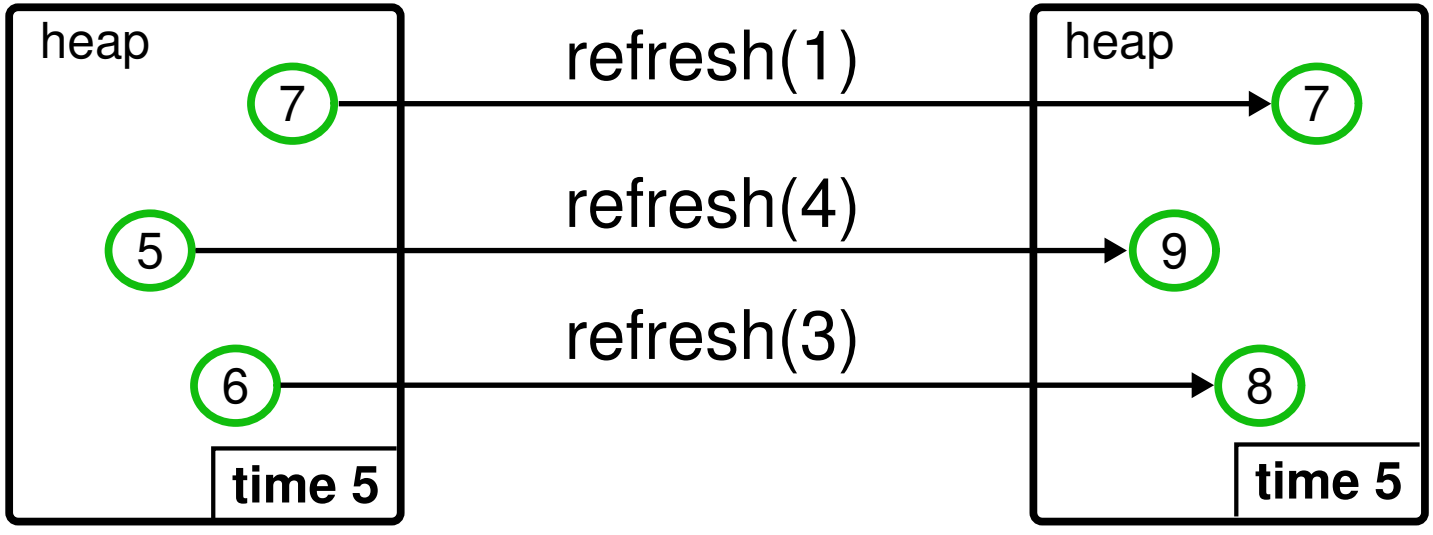
$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$



Refresh

- Extend the expiration date of an object by refreshing

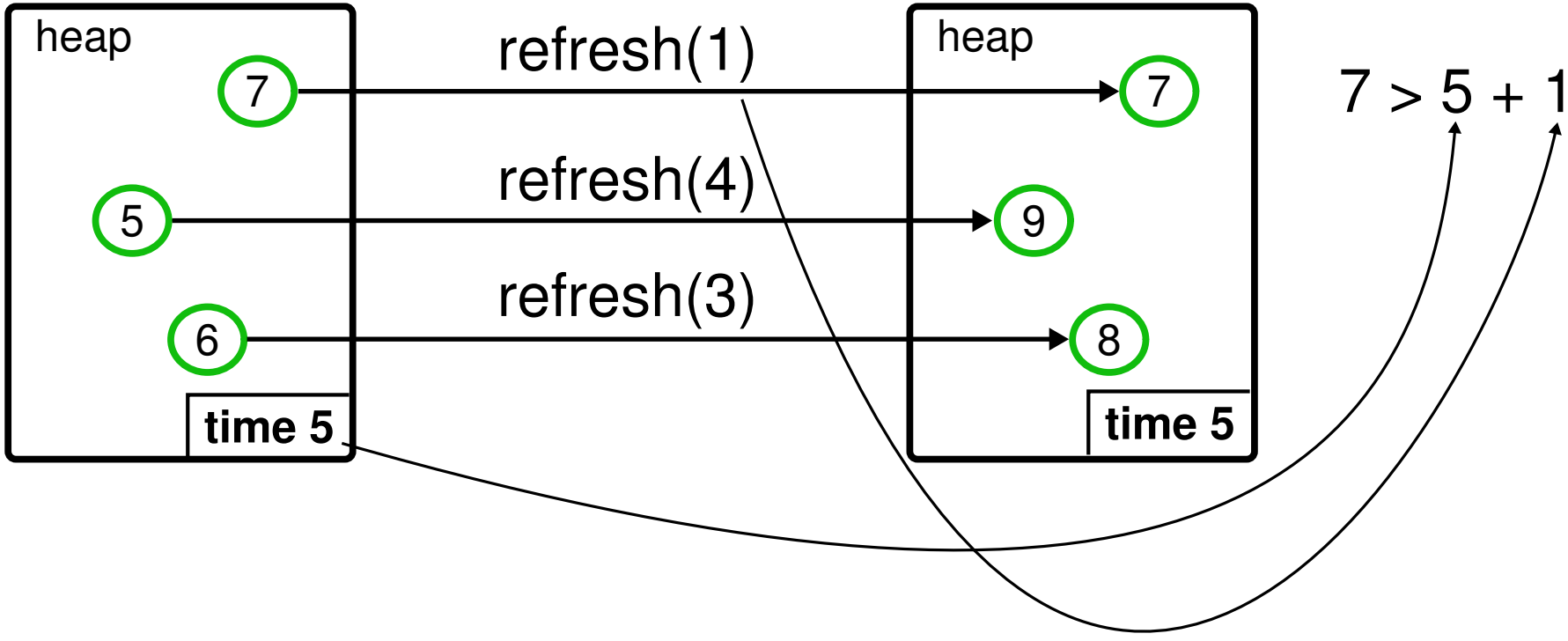
$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$



Refresh

- Extend the expiration date of an object by refreshing

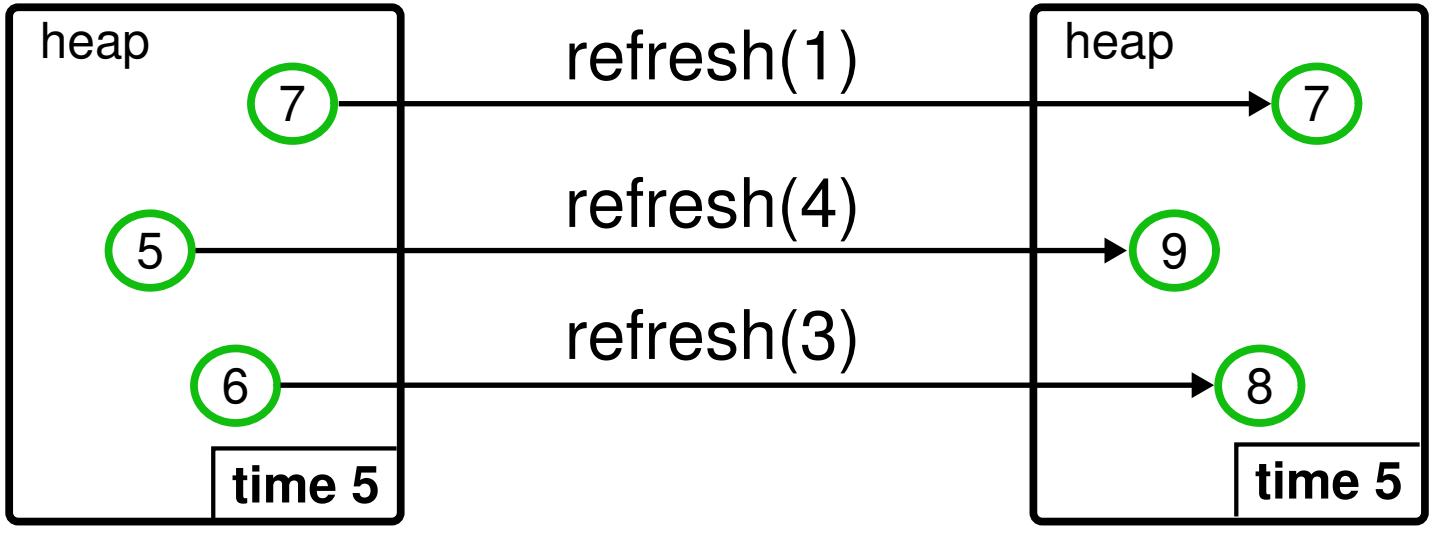
$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$



Refresh

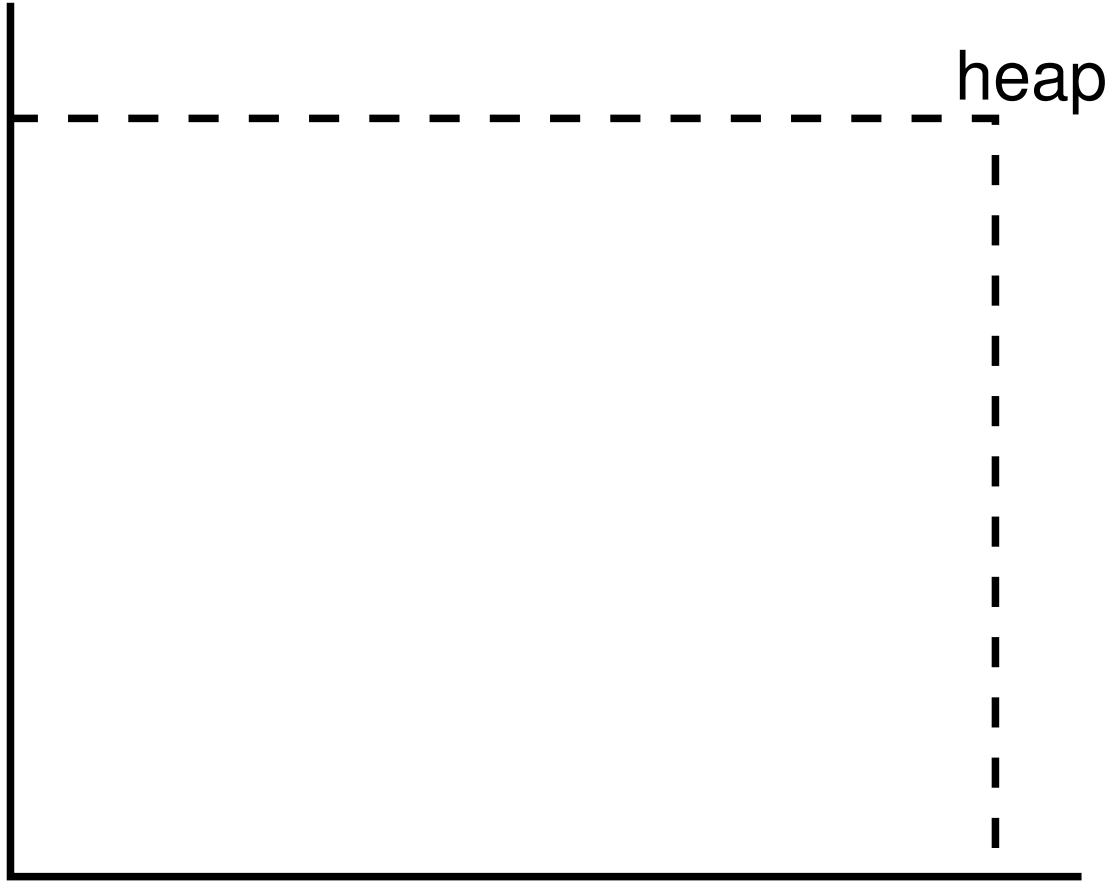
- Extend the expiration date of an object by refreshing

$$\textbf{new expiration date} = \max \begin{cases} \textbf{current time + extension} \\ \textbf{current expiration date} \end{cases}$$

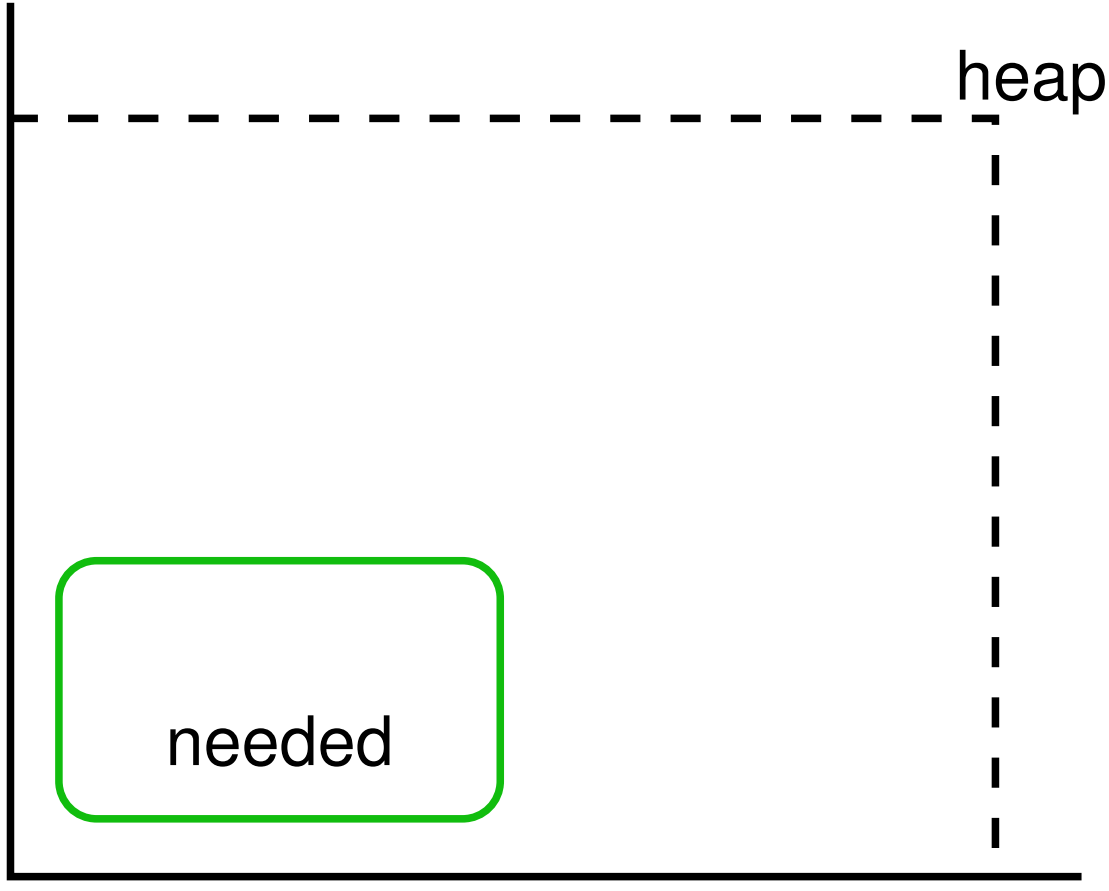


- Expiration extensions do not accumulate
- Objects may be refreshed multiple times as long as they have not expired

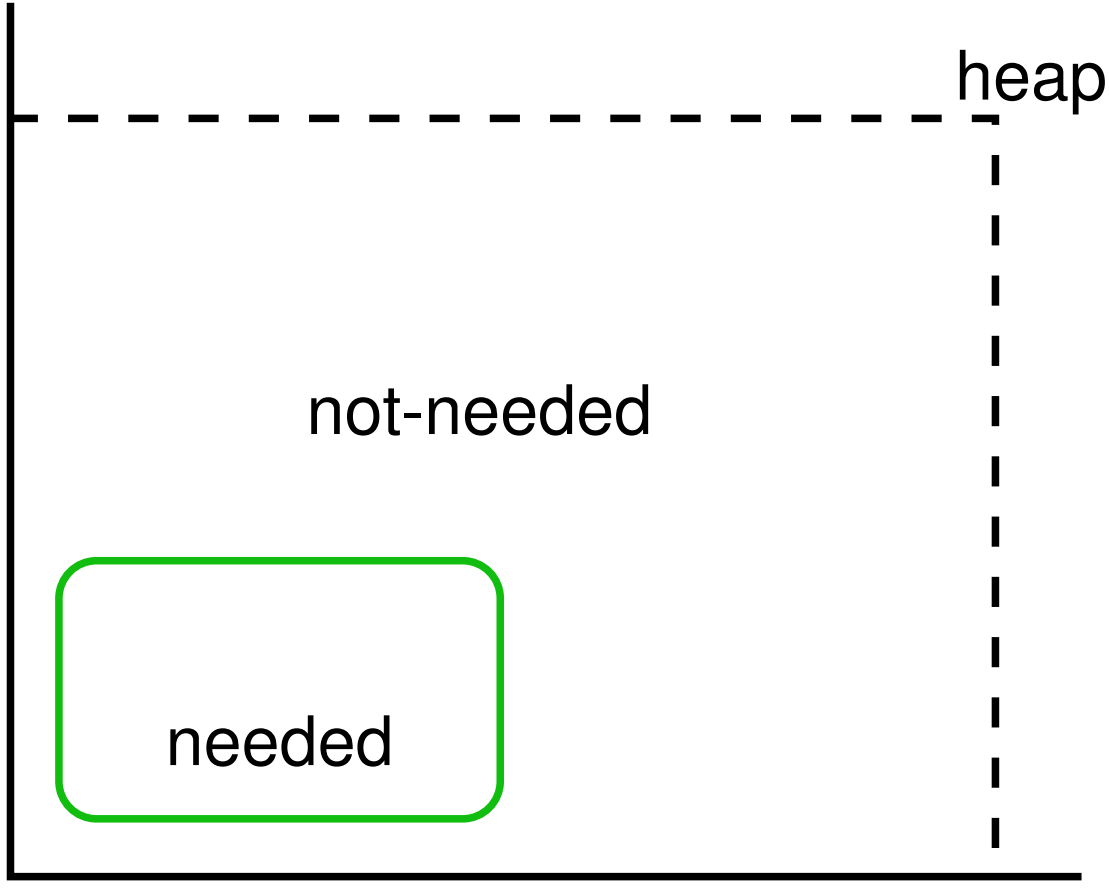
Heap Management



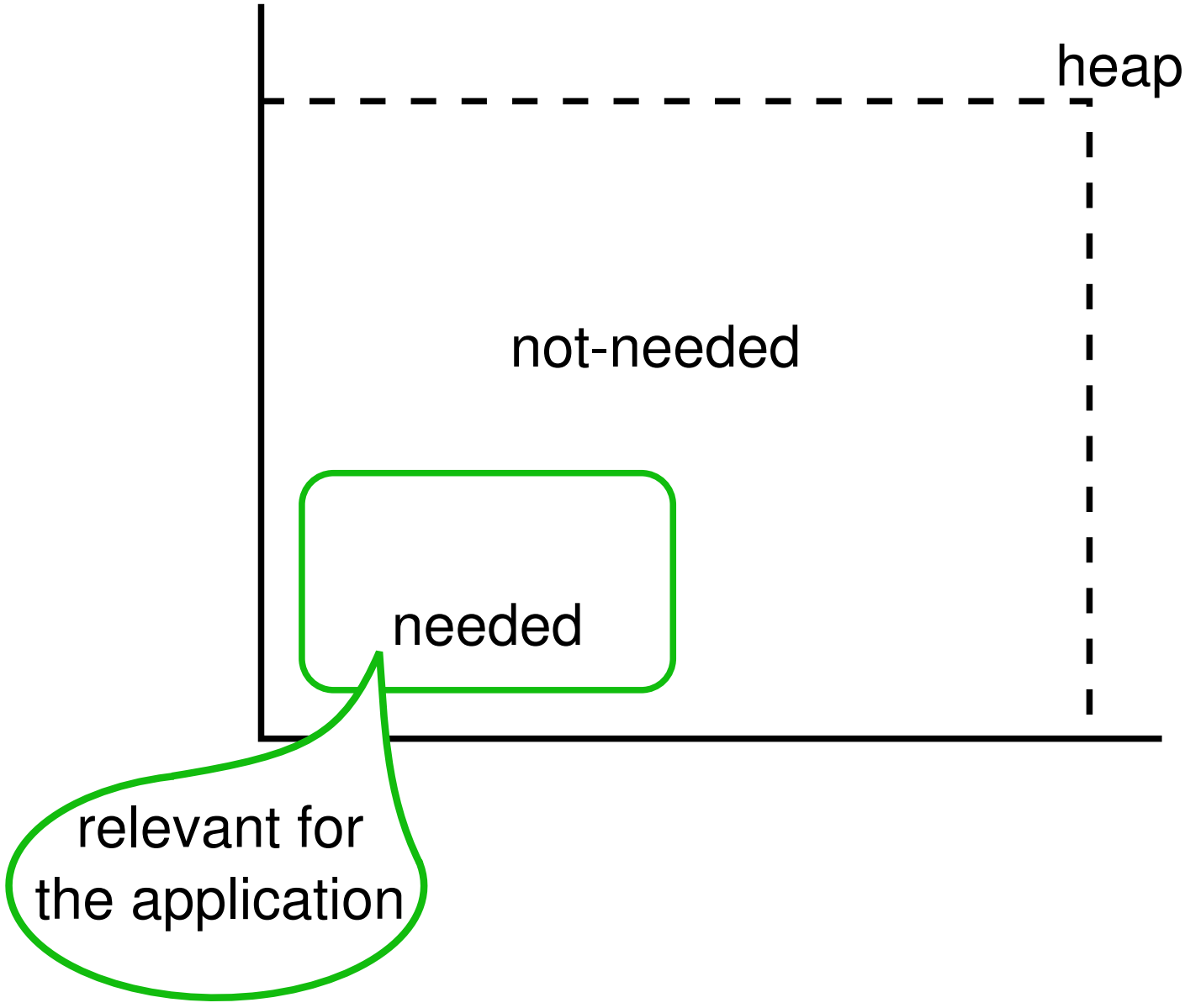
Heap Management



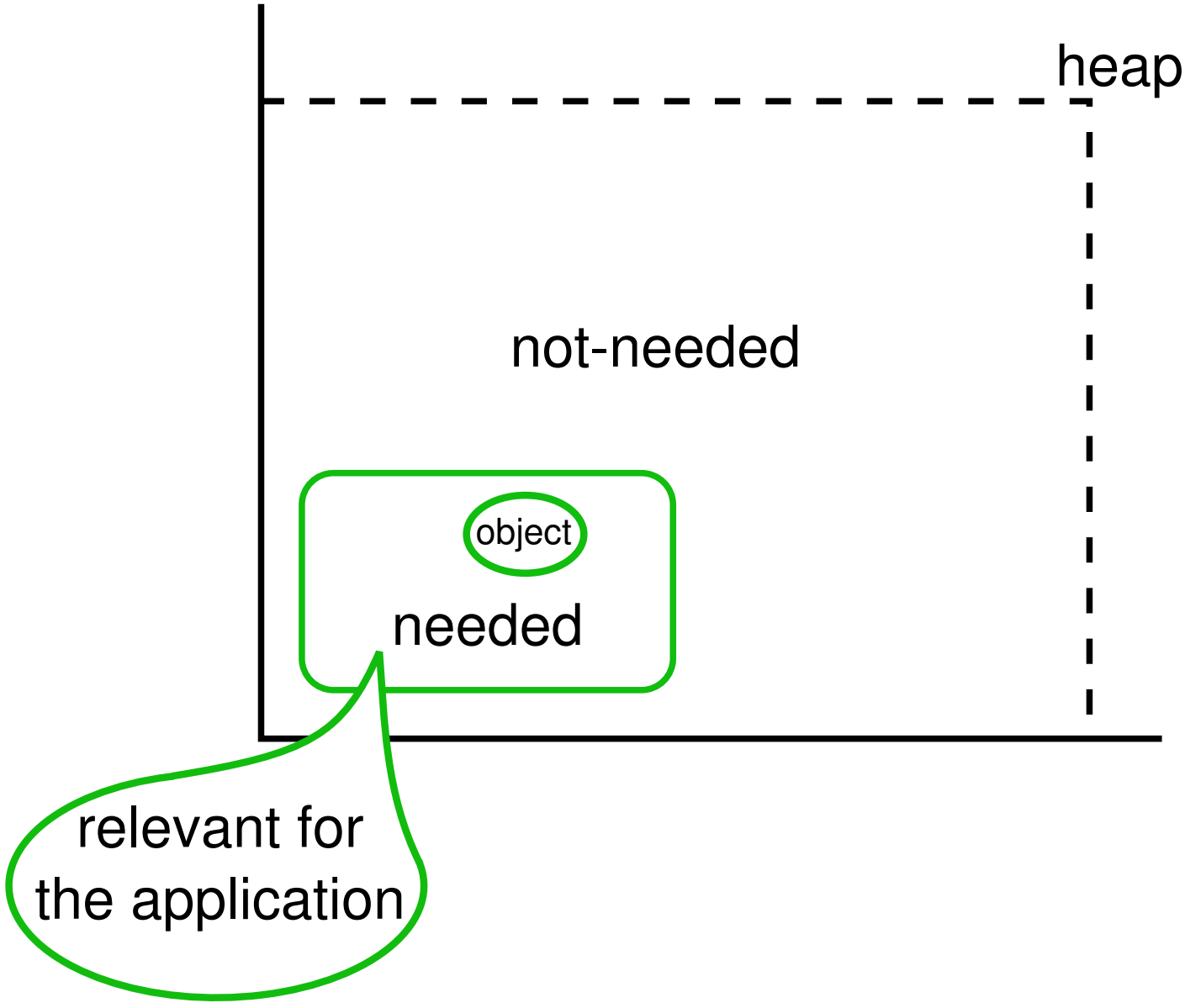
Heap Management



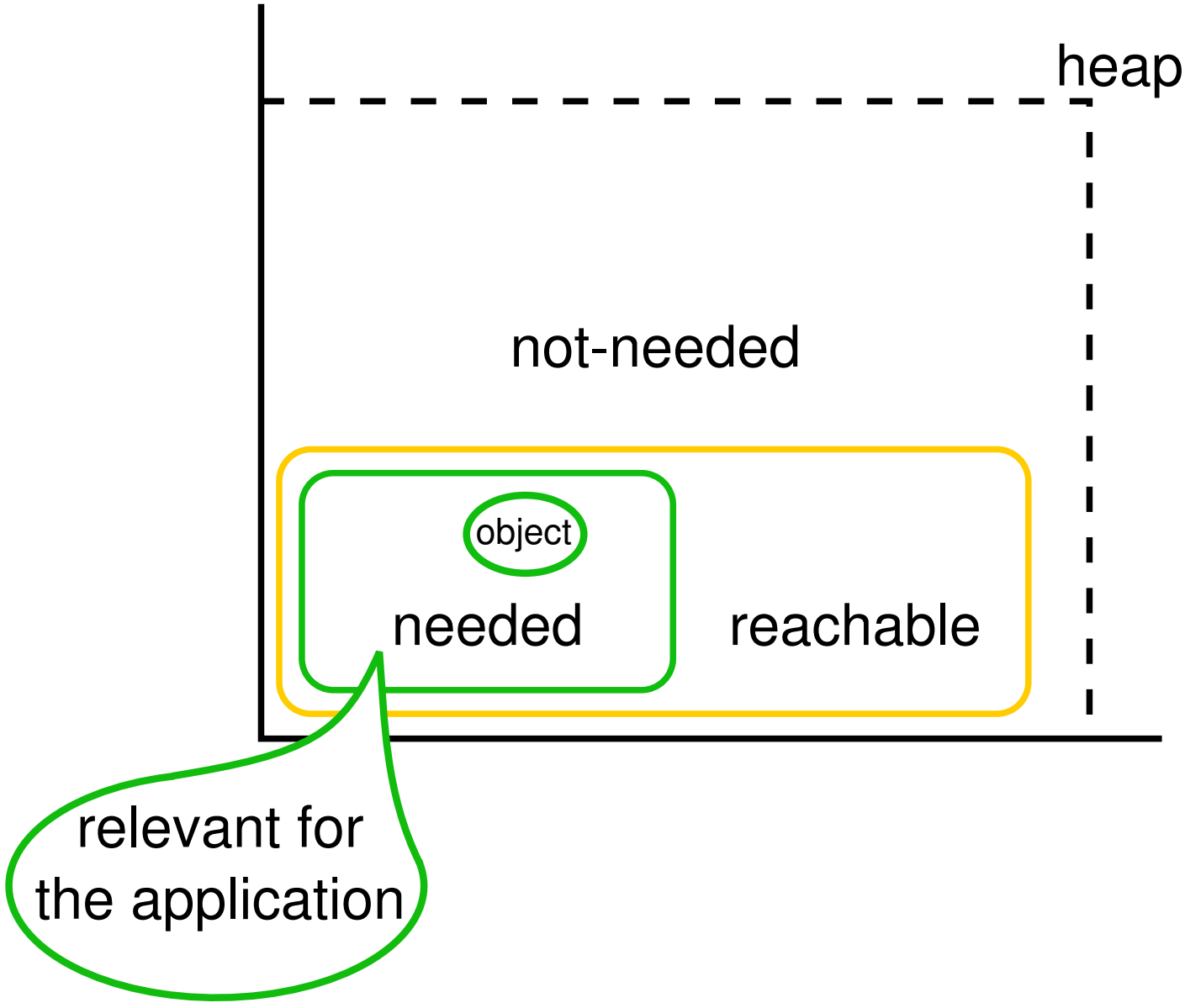
Heap Management



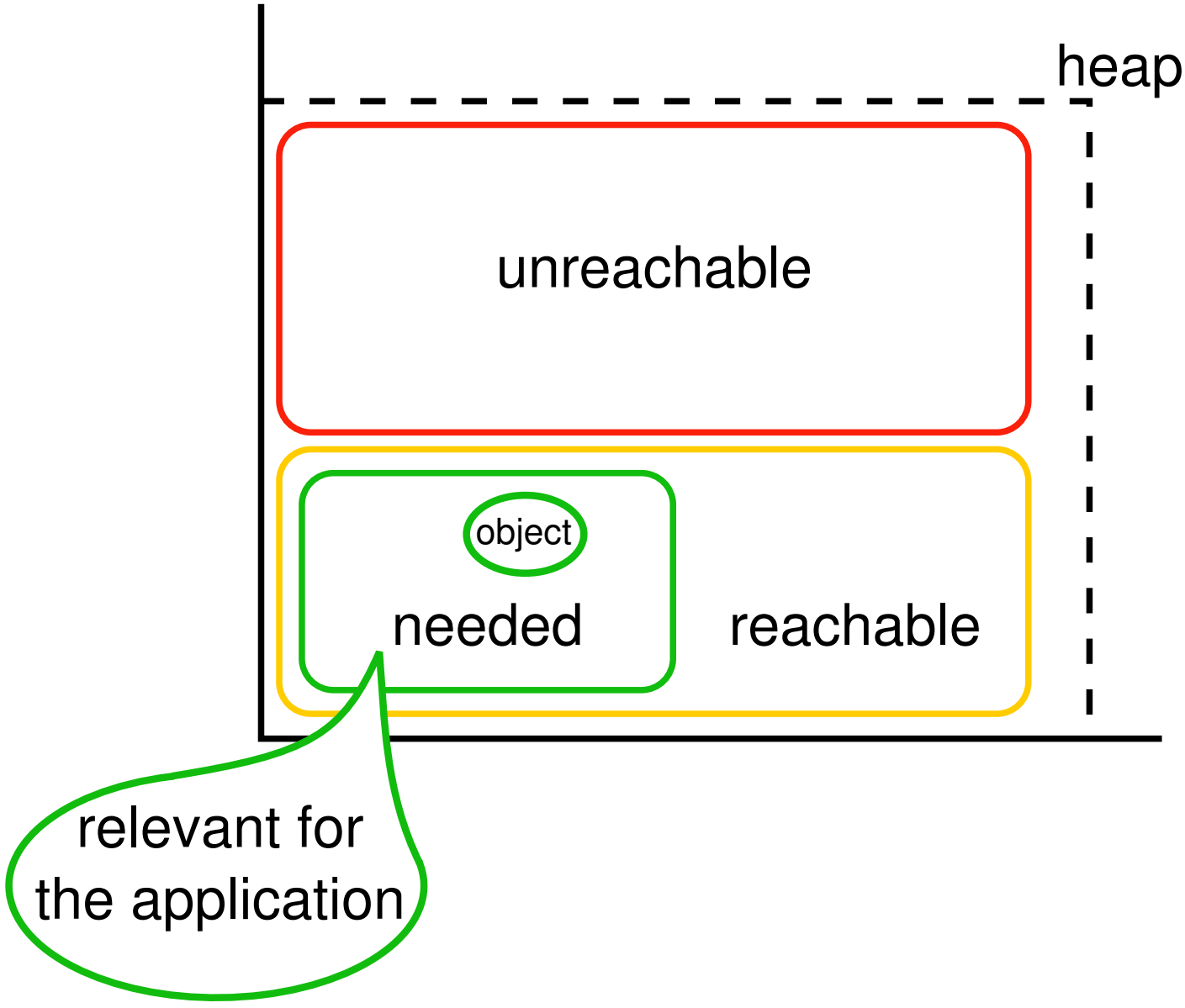
Heap Management



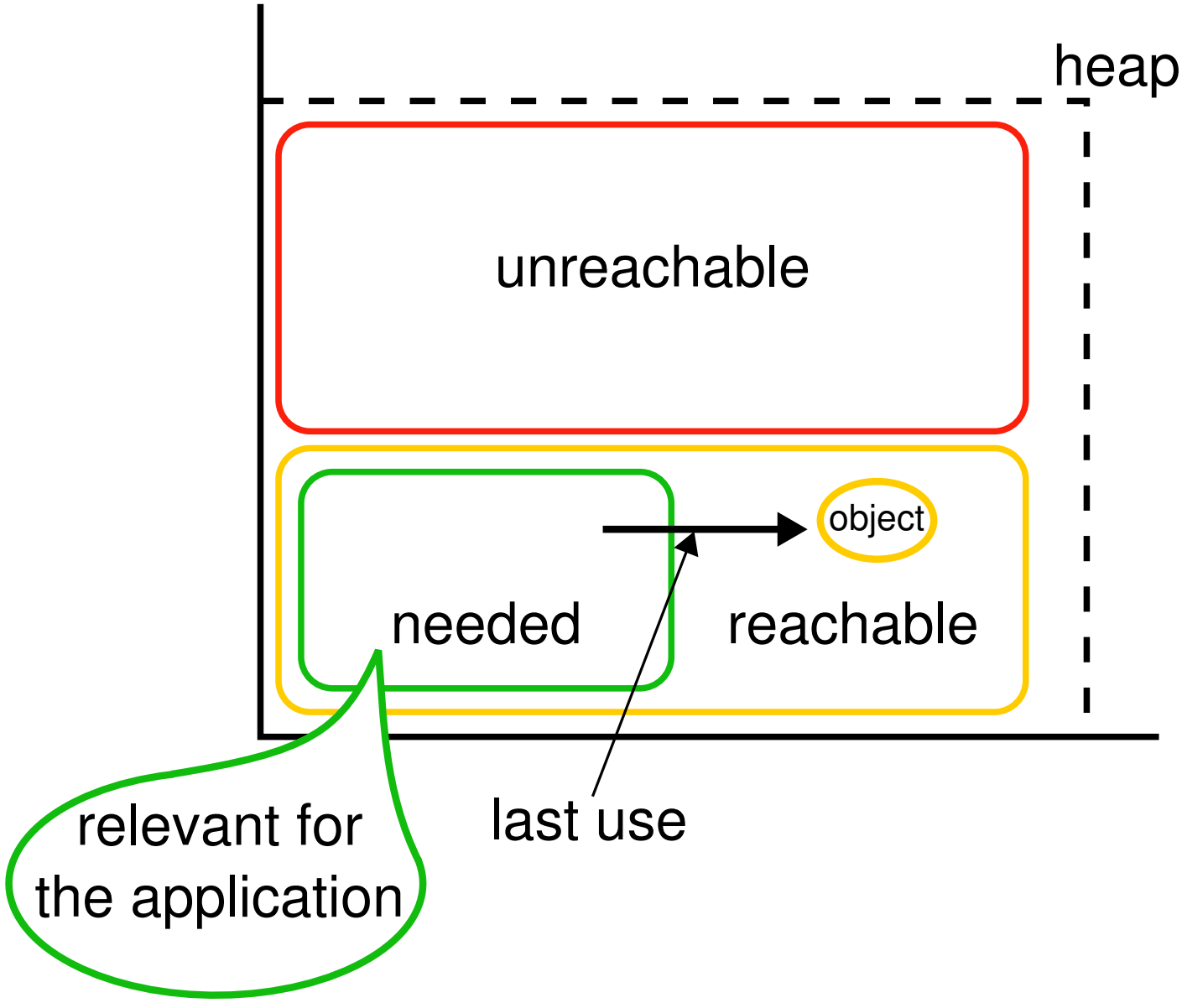
Heap Management



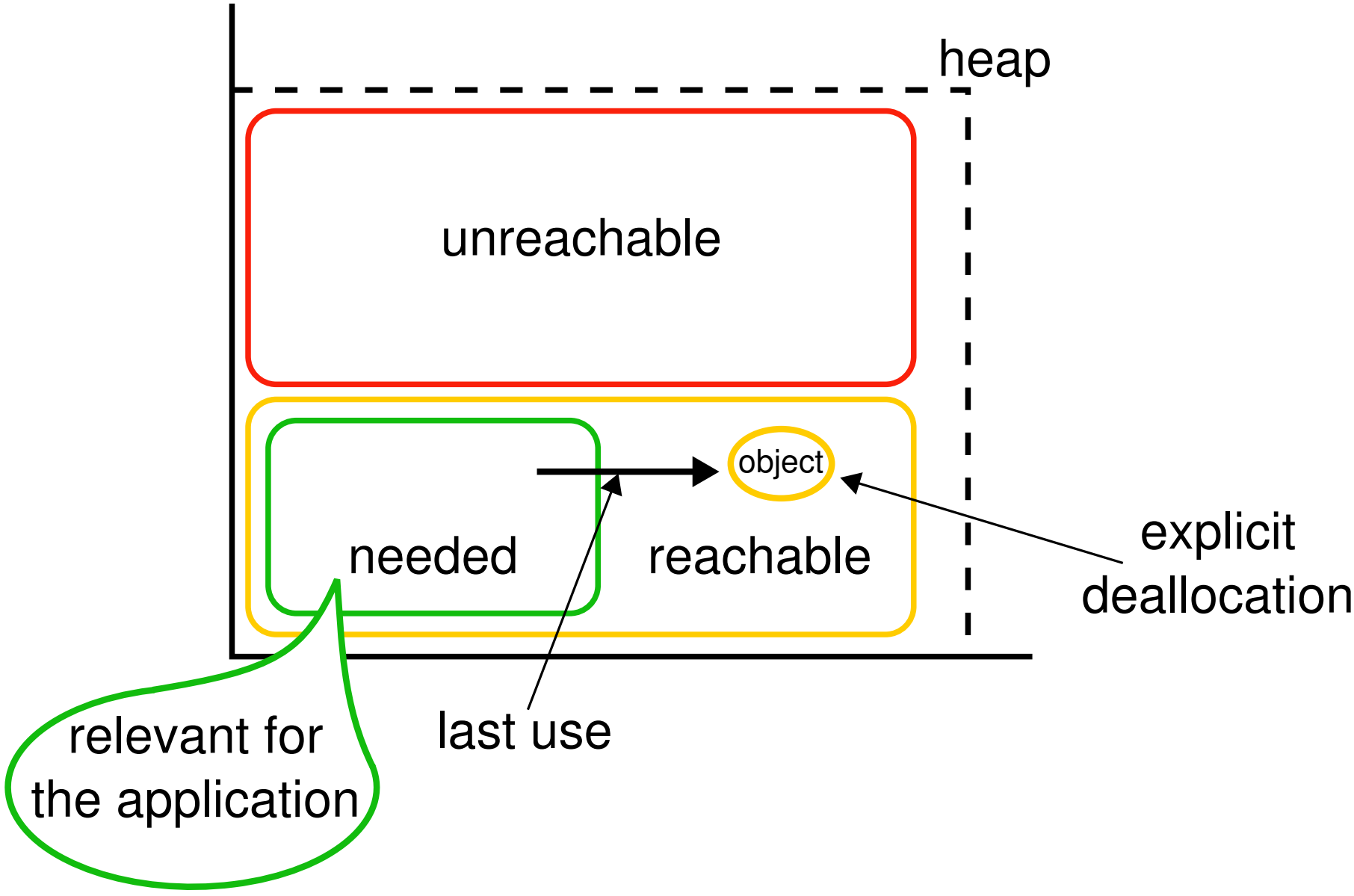
Heap Management



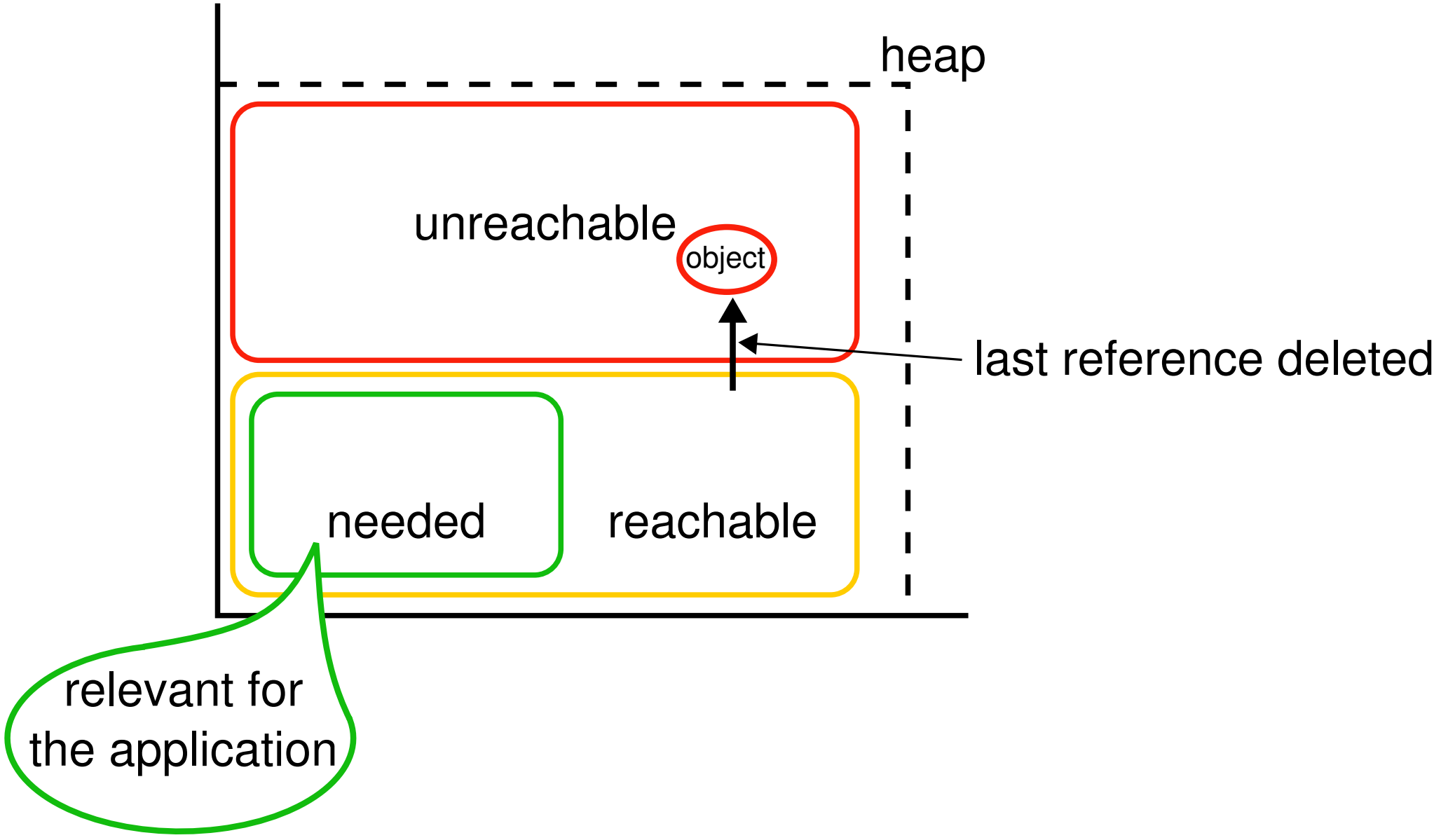
Heap Management



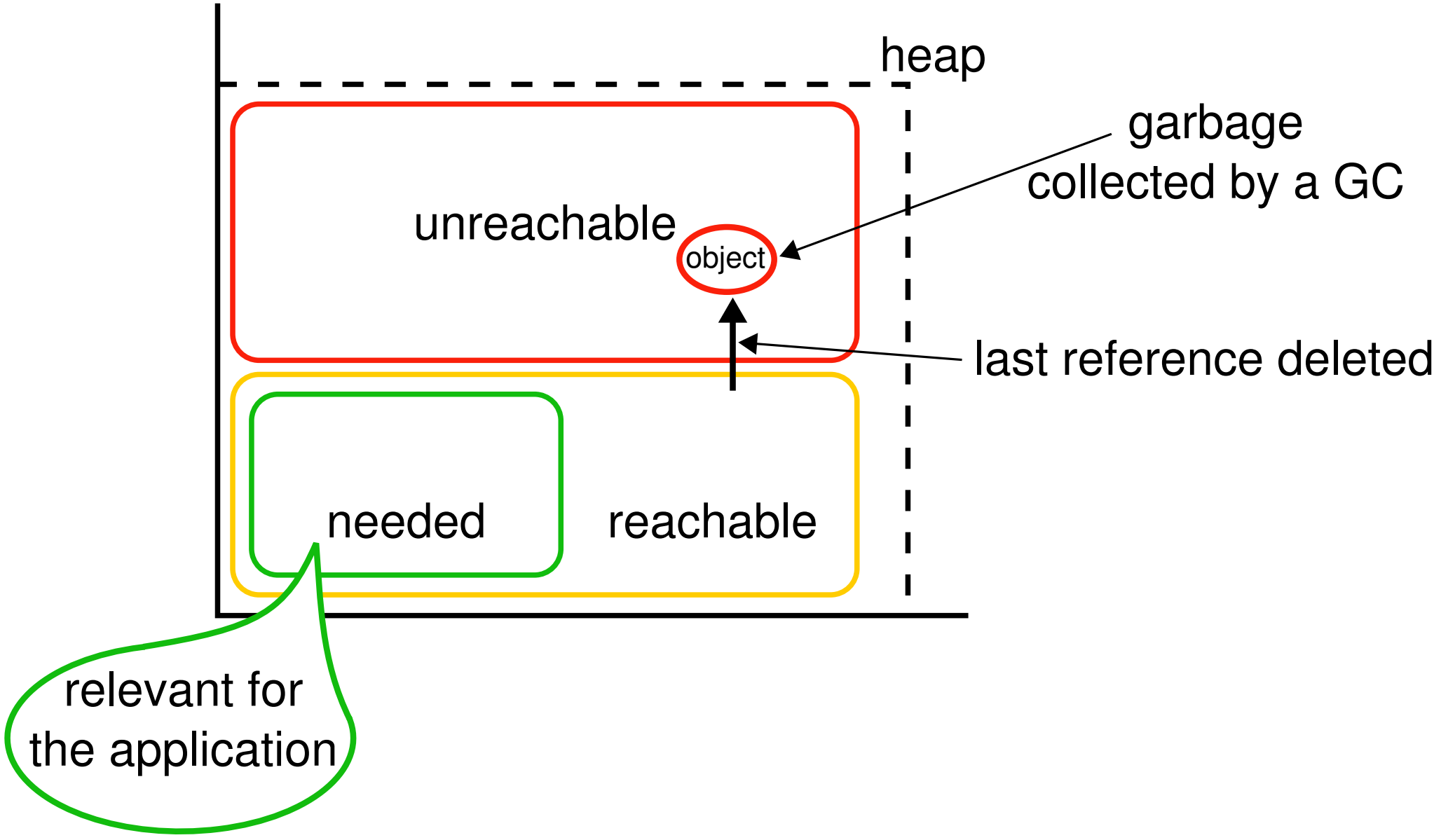
Heap Management



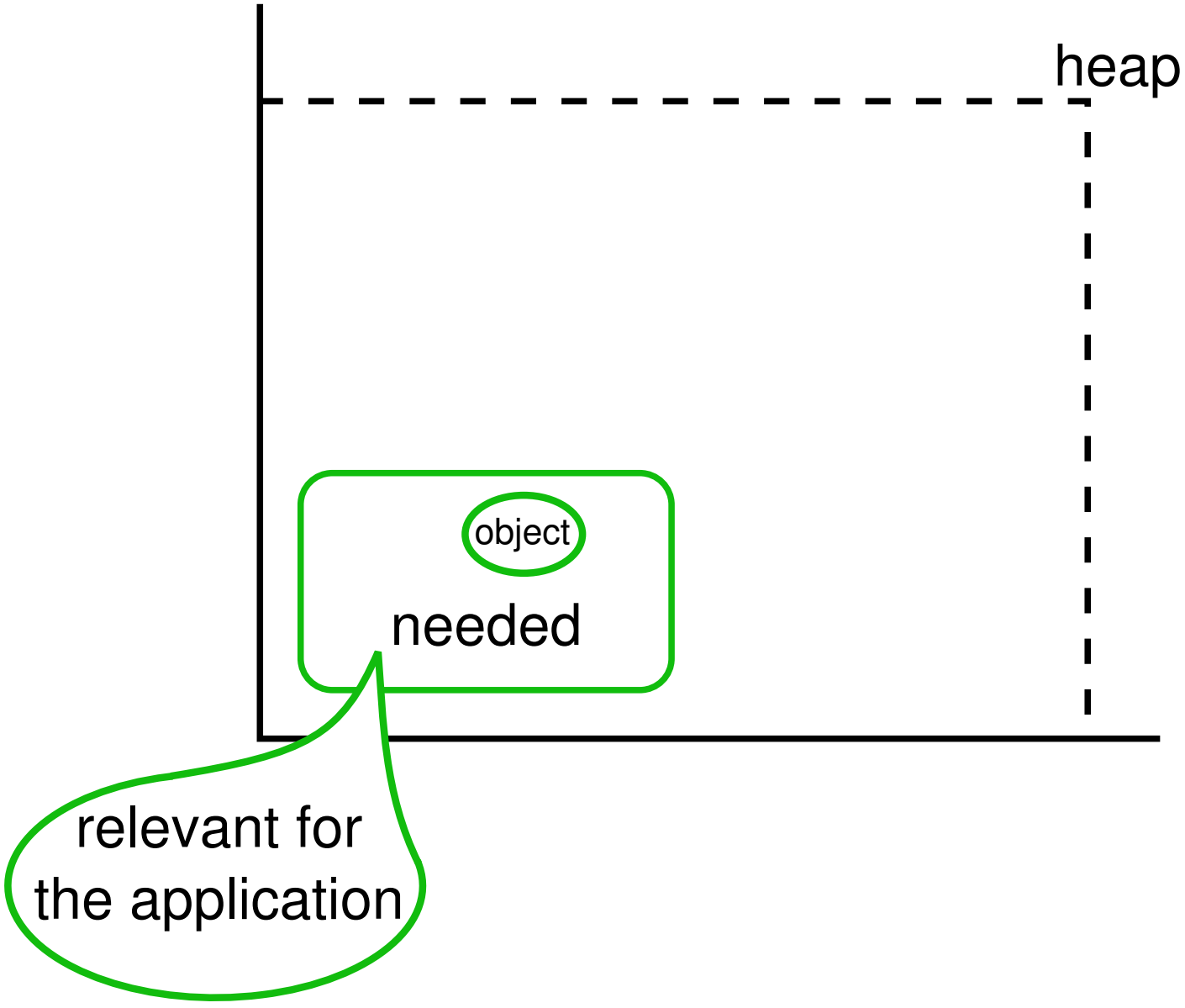
Heap Management



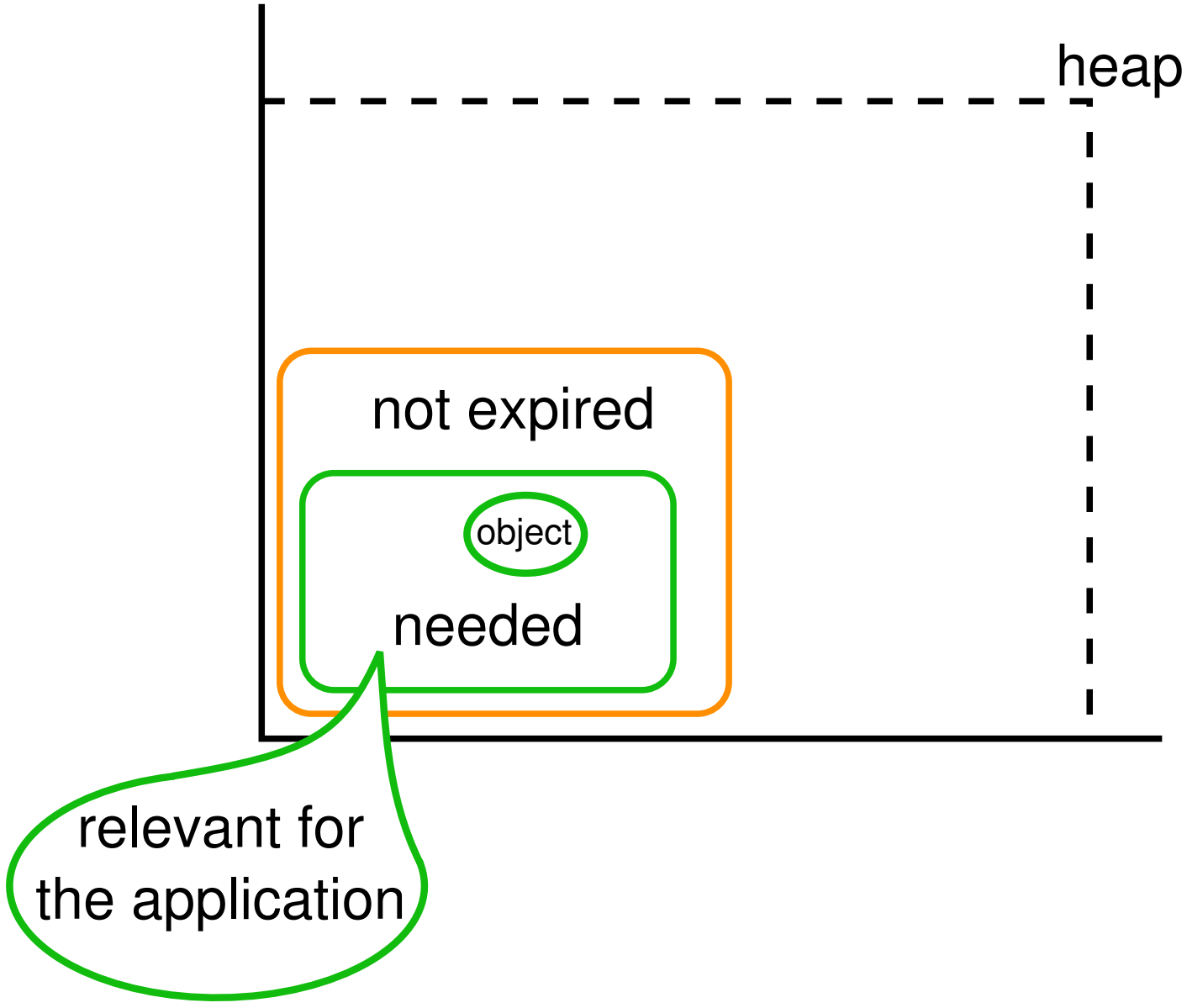
Heap Management



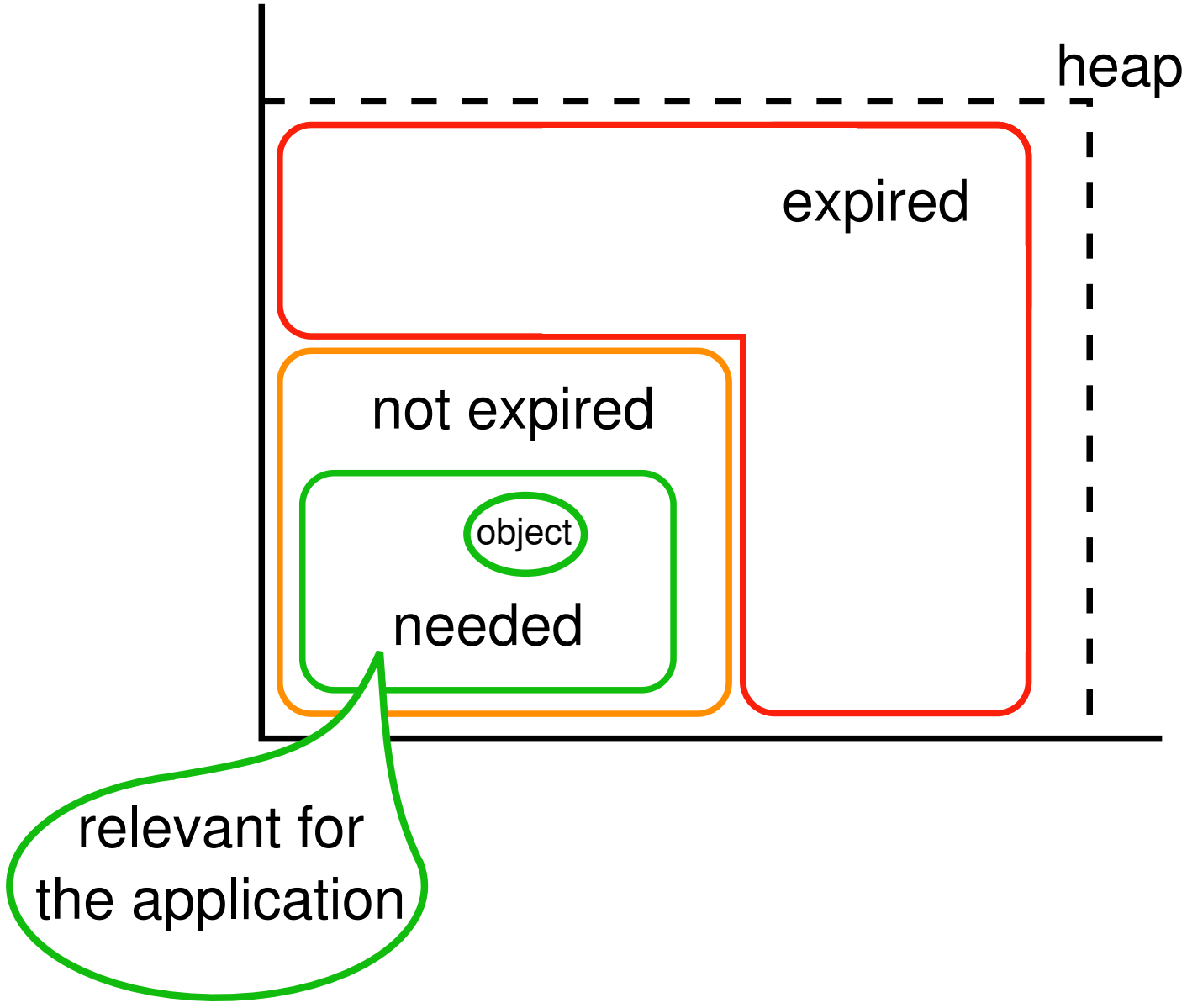
Short-term memory



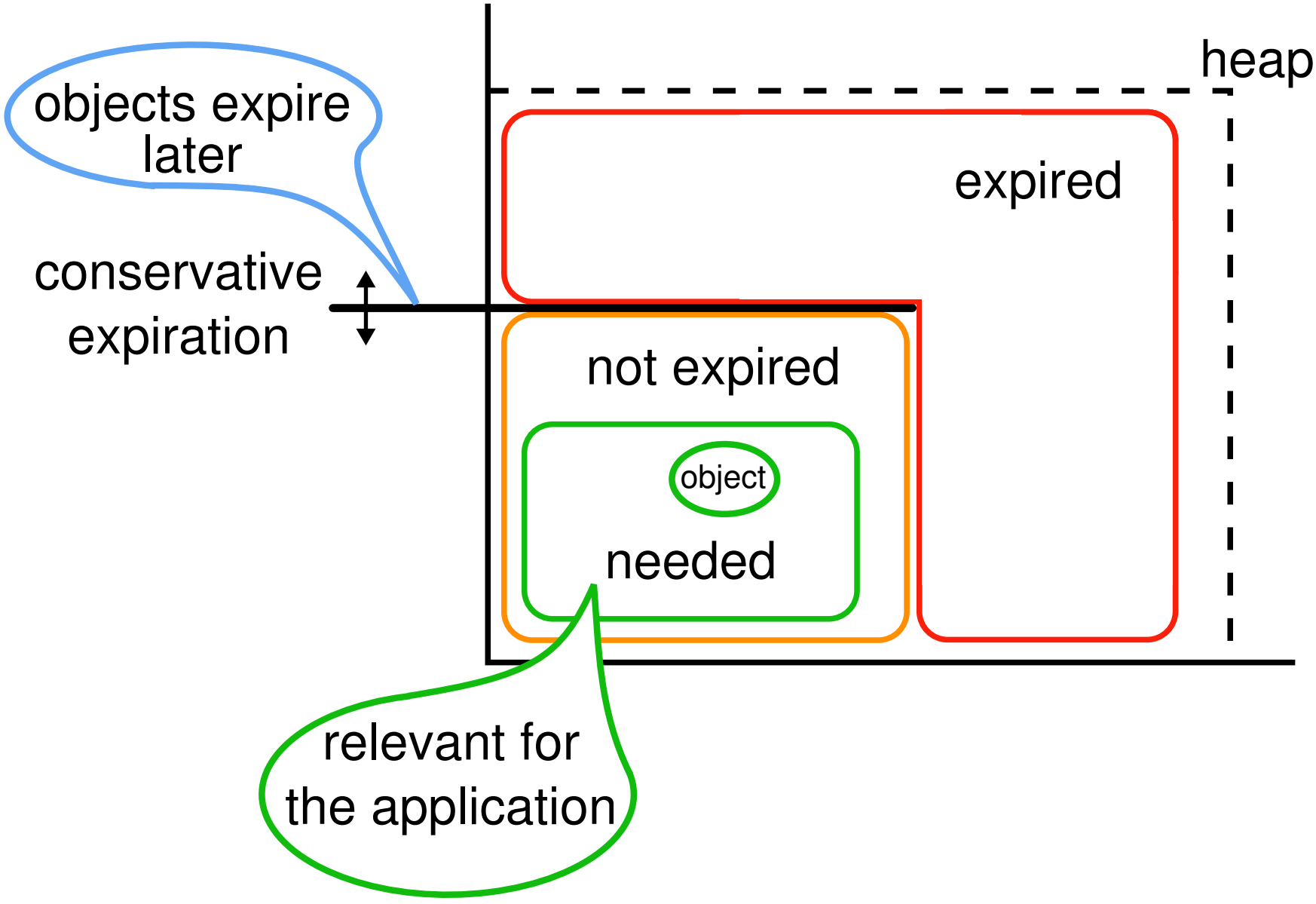
Short-term memory



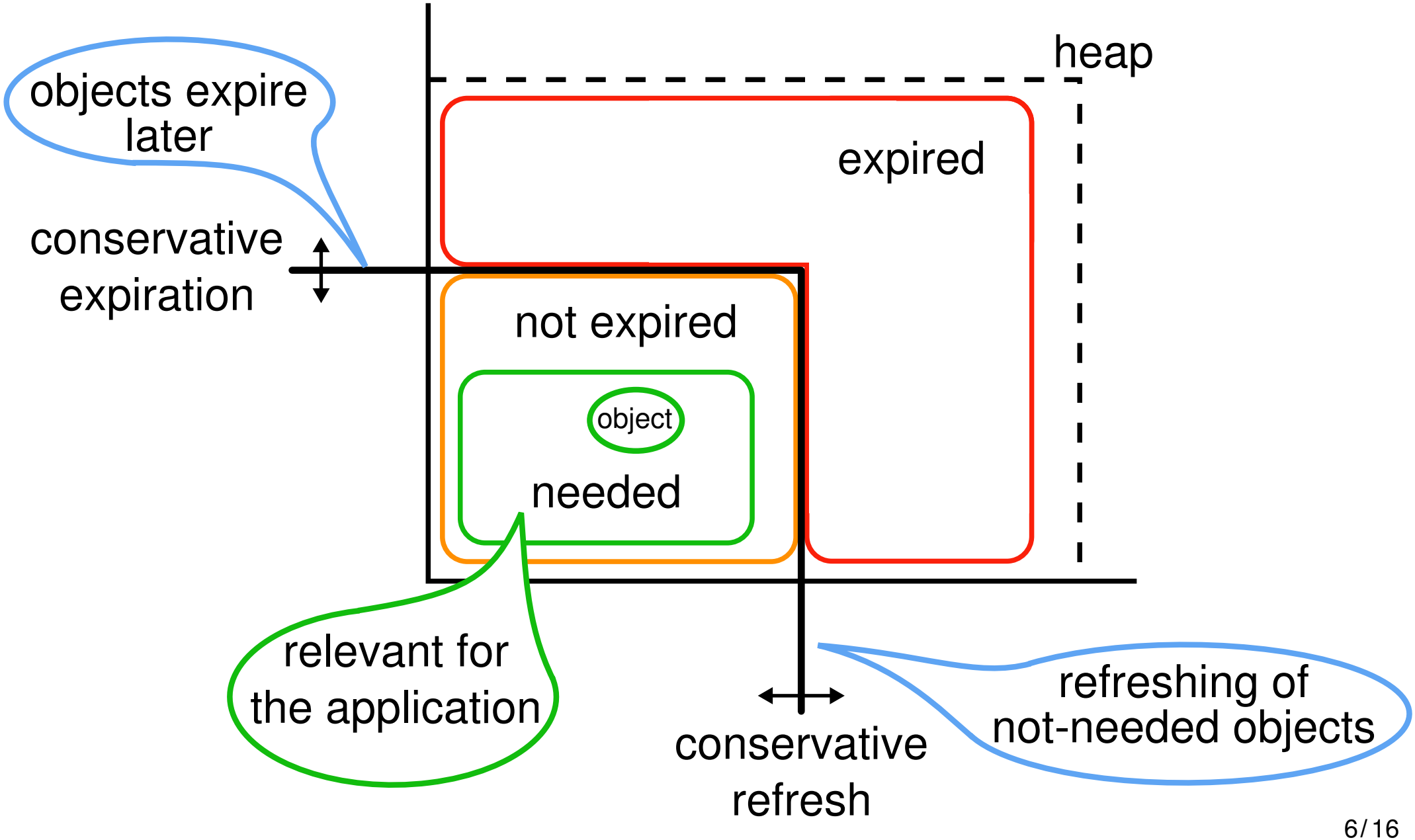
Short-term memory



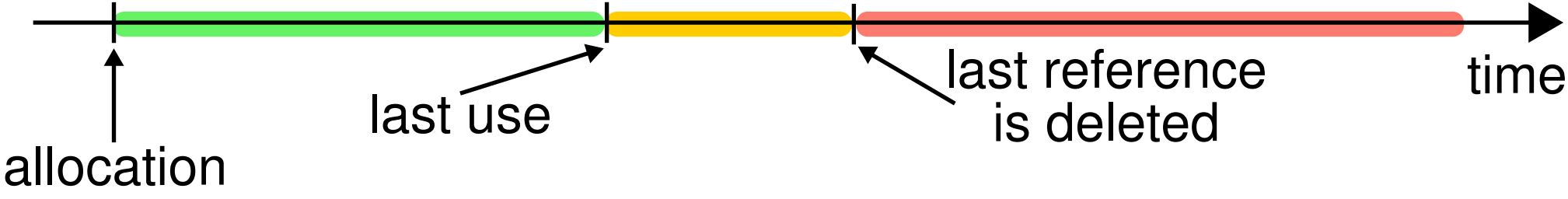
Short-term memory



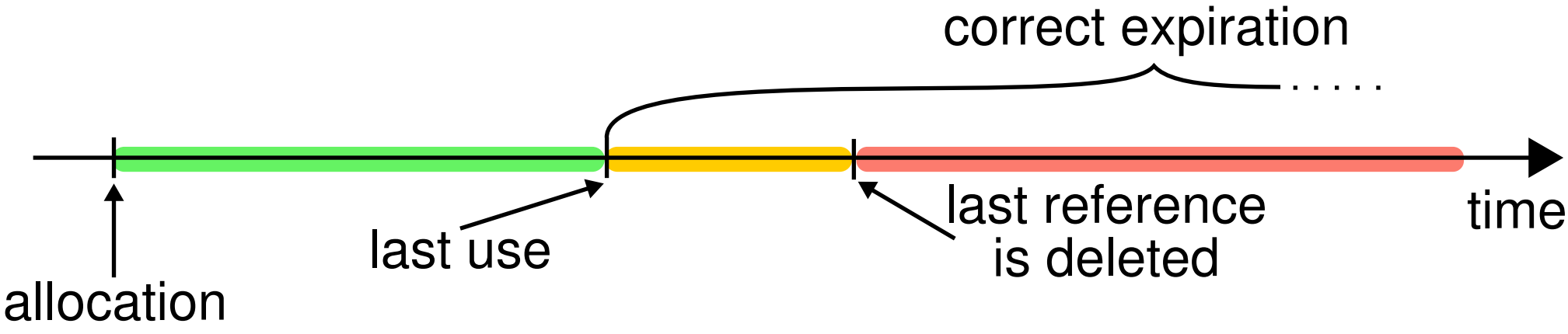
Short-term memory



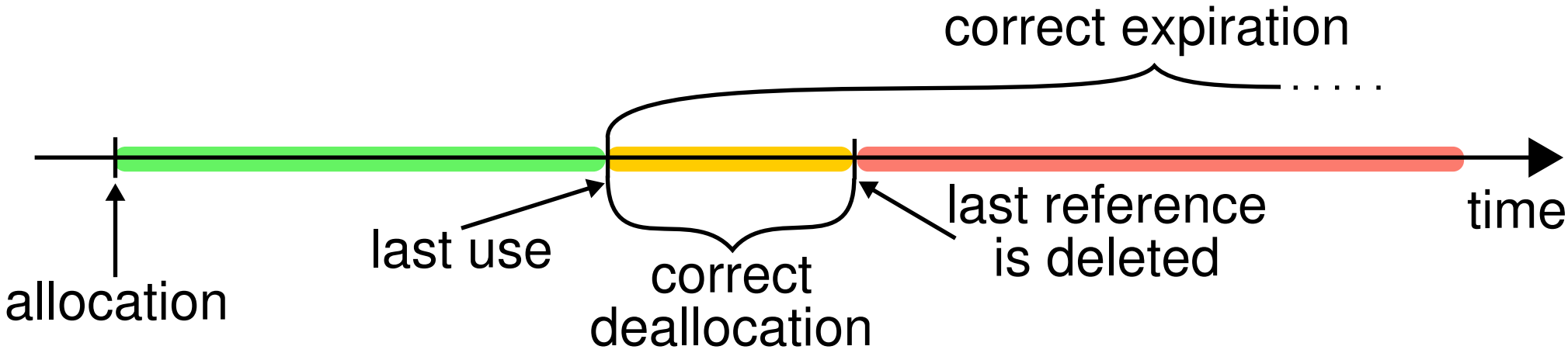
Comparison



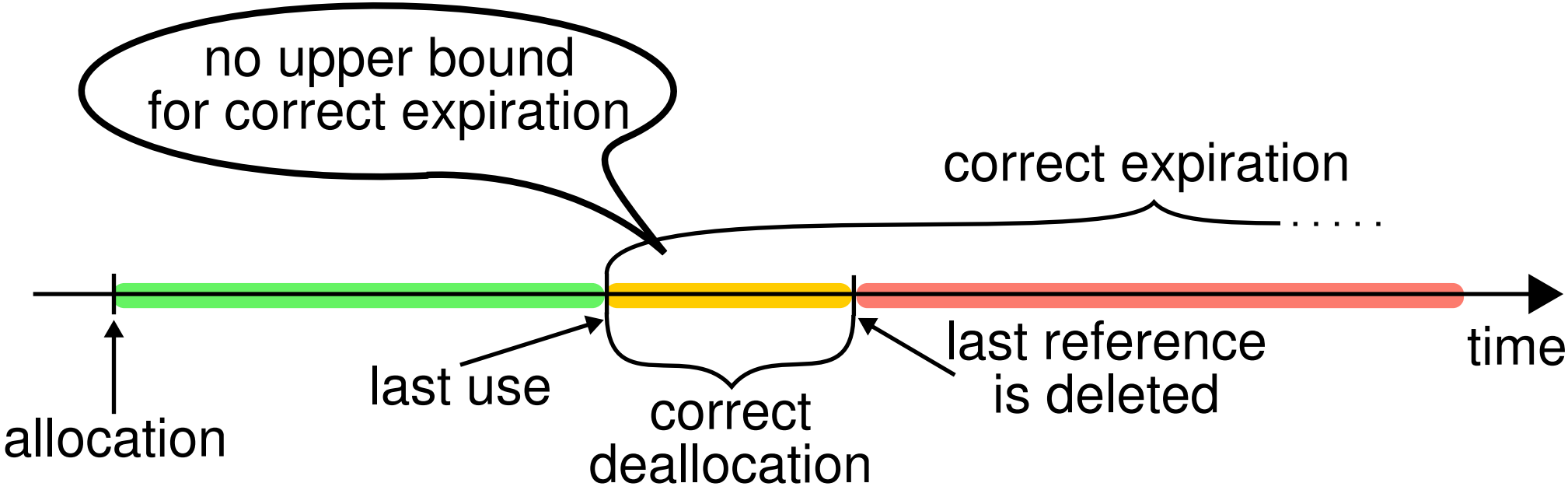
Comparison


correct expiration
allocation
last use
last reference
is deleted
time

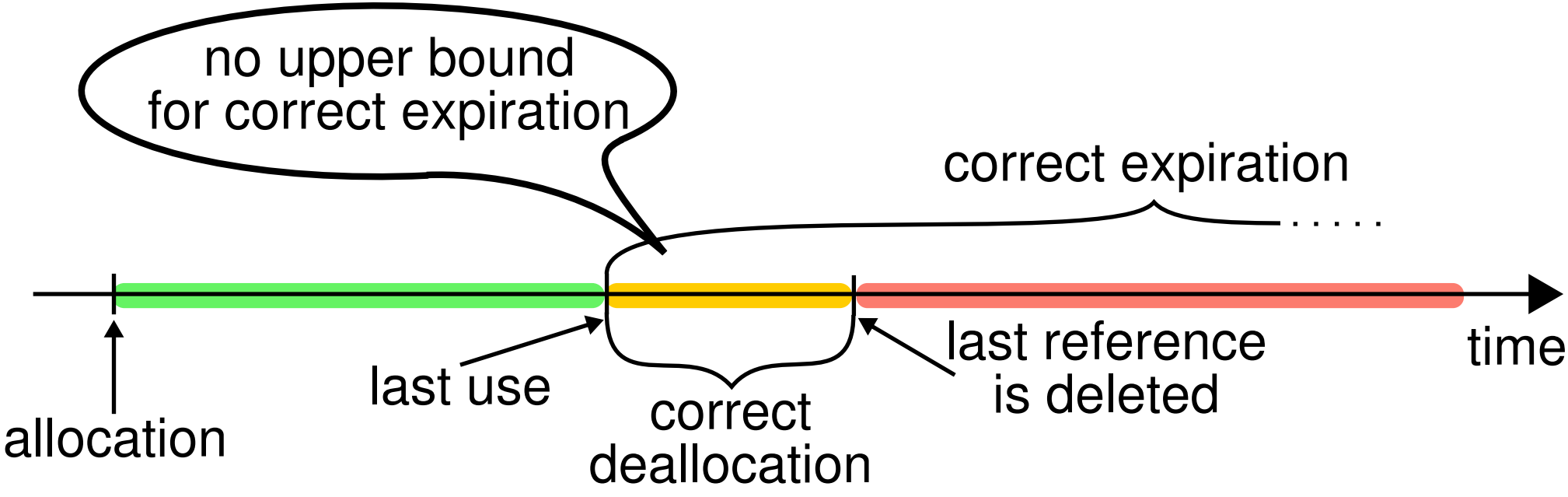
Comparison


correct expiration
allocation
last use
correct
deallocation
last reference
is deleted
time

Comparison


no upper bound
for correct expiration
correct expiration
allocation
last use
correct
deallocation
last reference
is deleted
time

Comparison

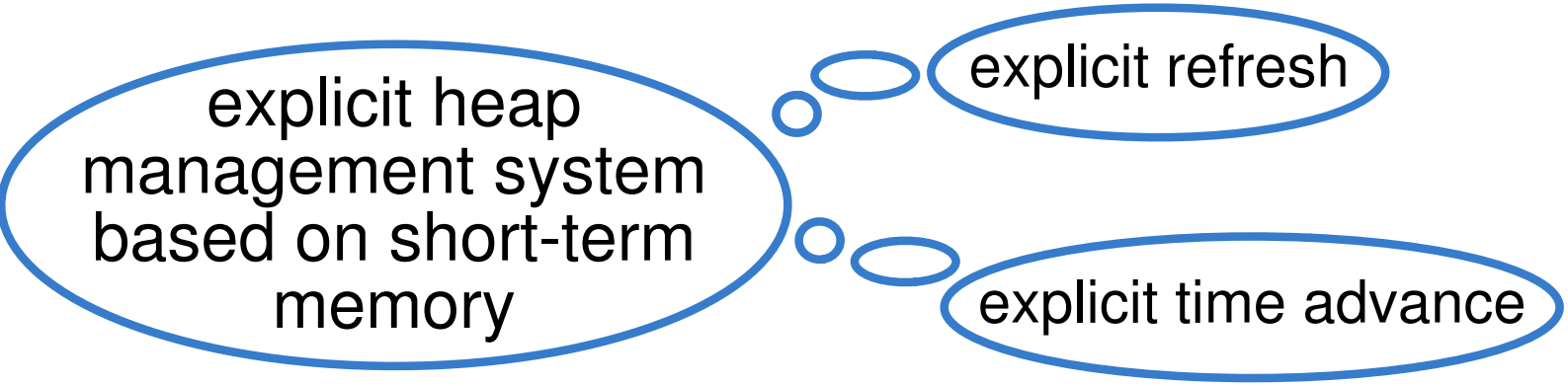


- Conservative lifetime approximations possible in short-term memory
  - May be easier to determine

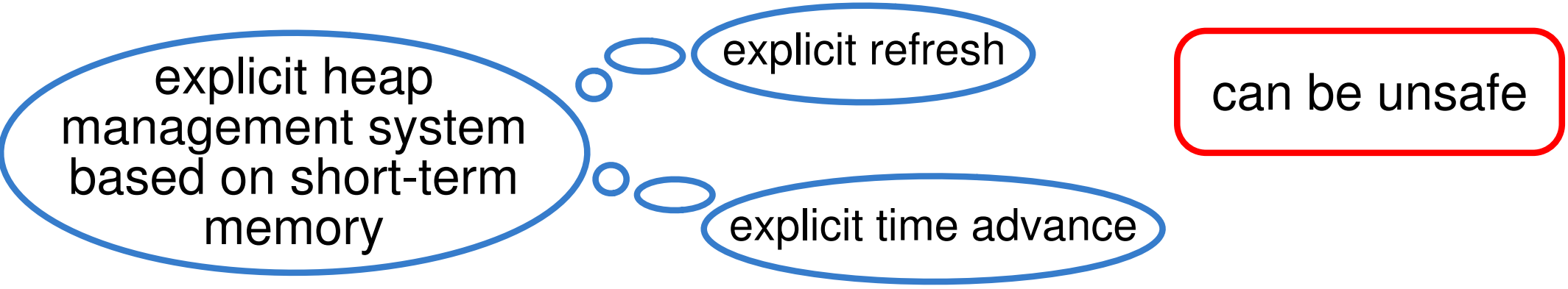
Self-collecting Mutators

explicit heap management system based on short-term memory

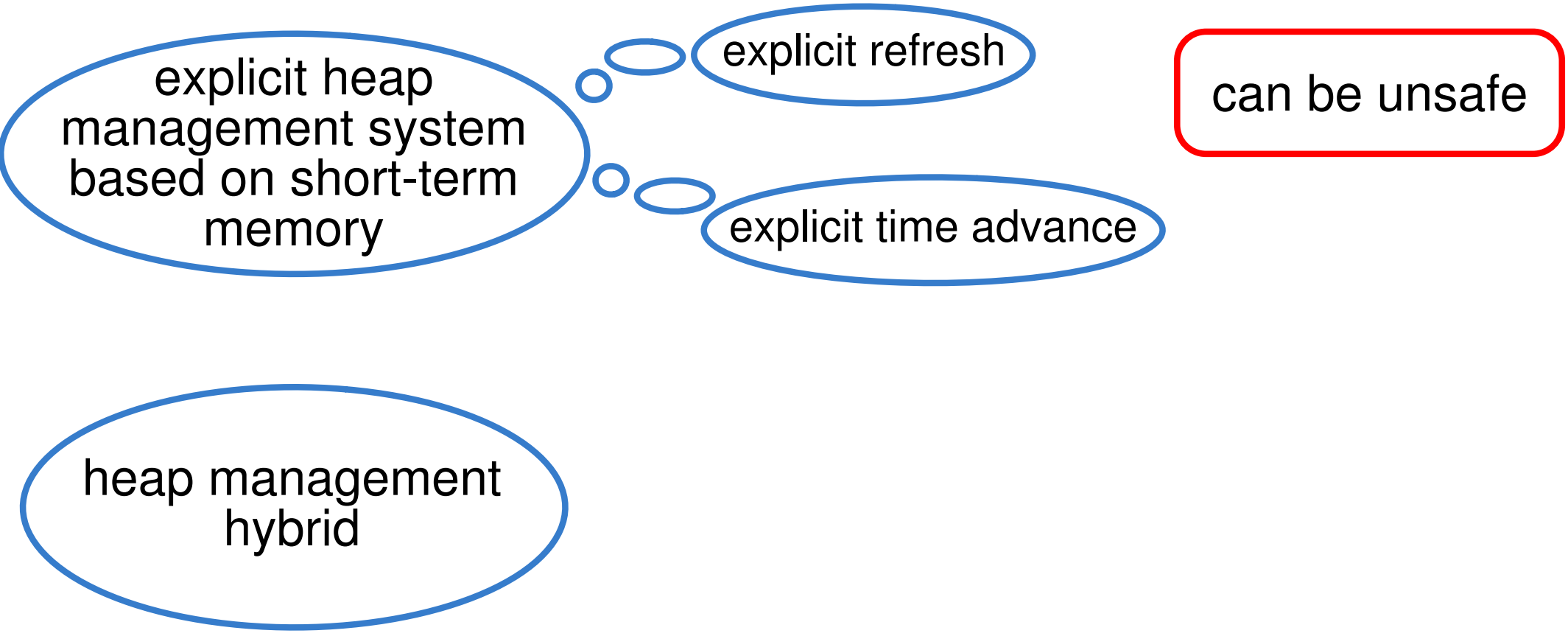
Self-collecting Mutators



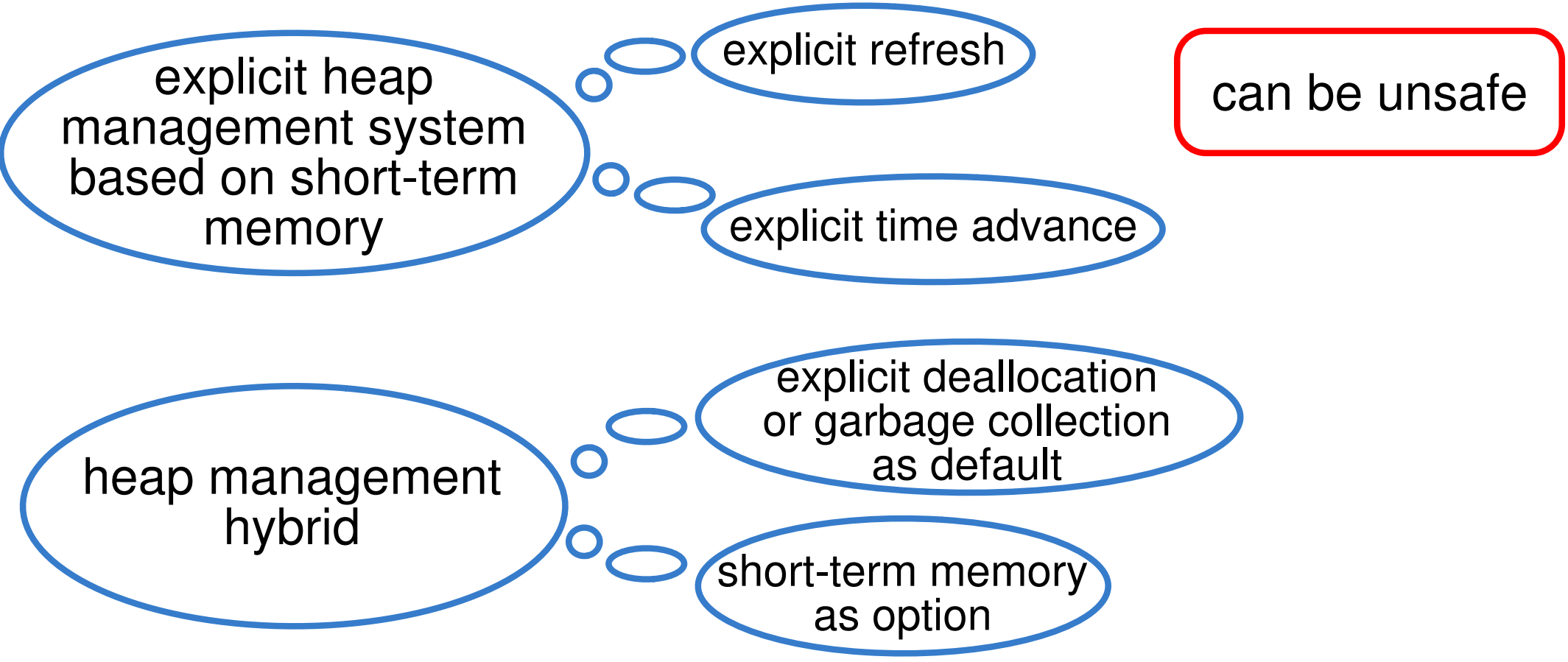
Self-collecting Mutators



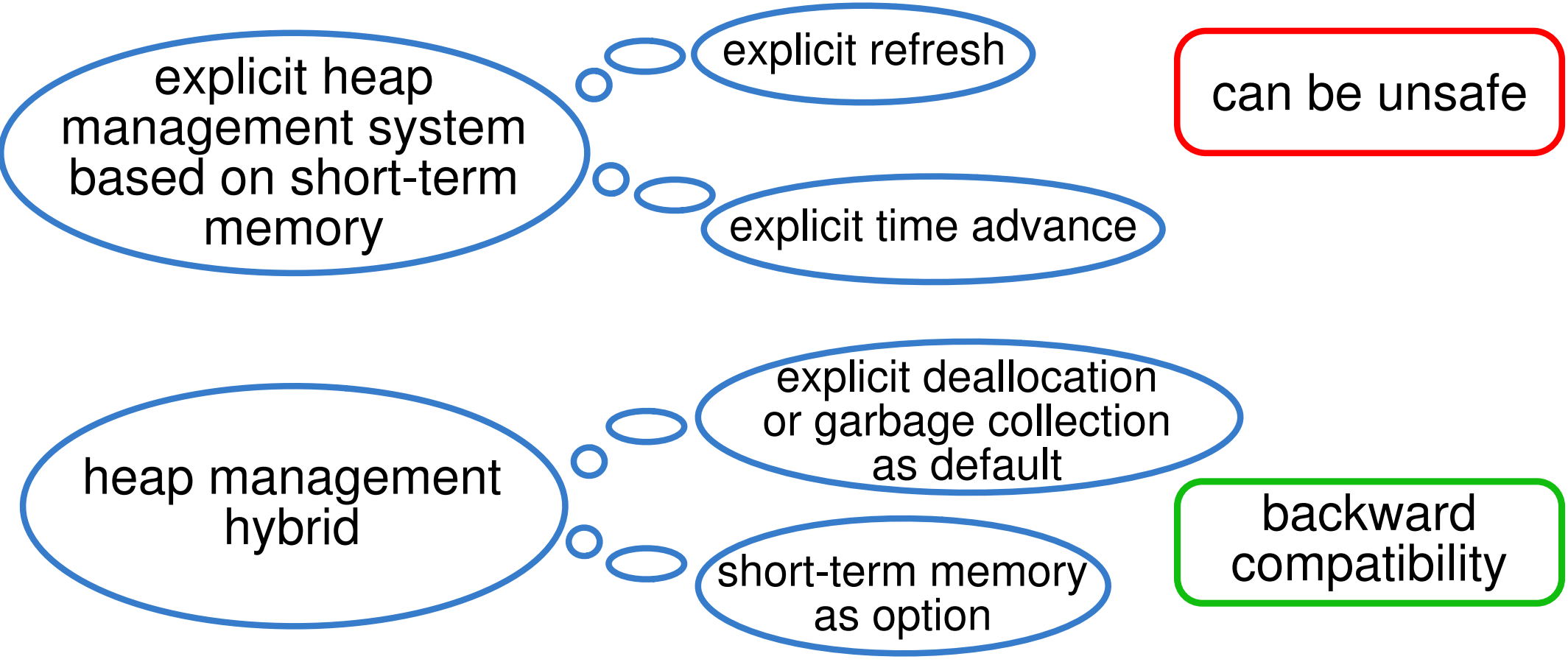
Self-collecting Mutators



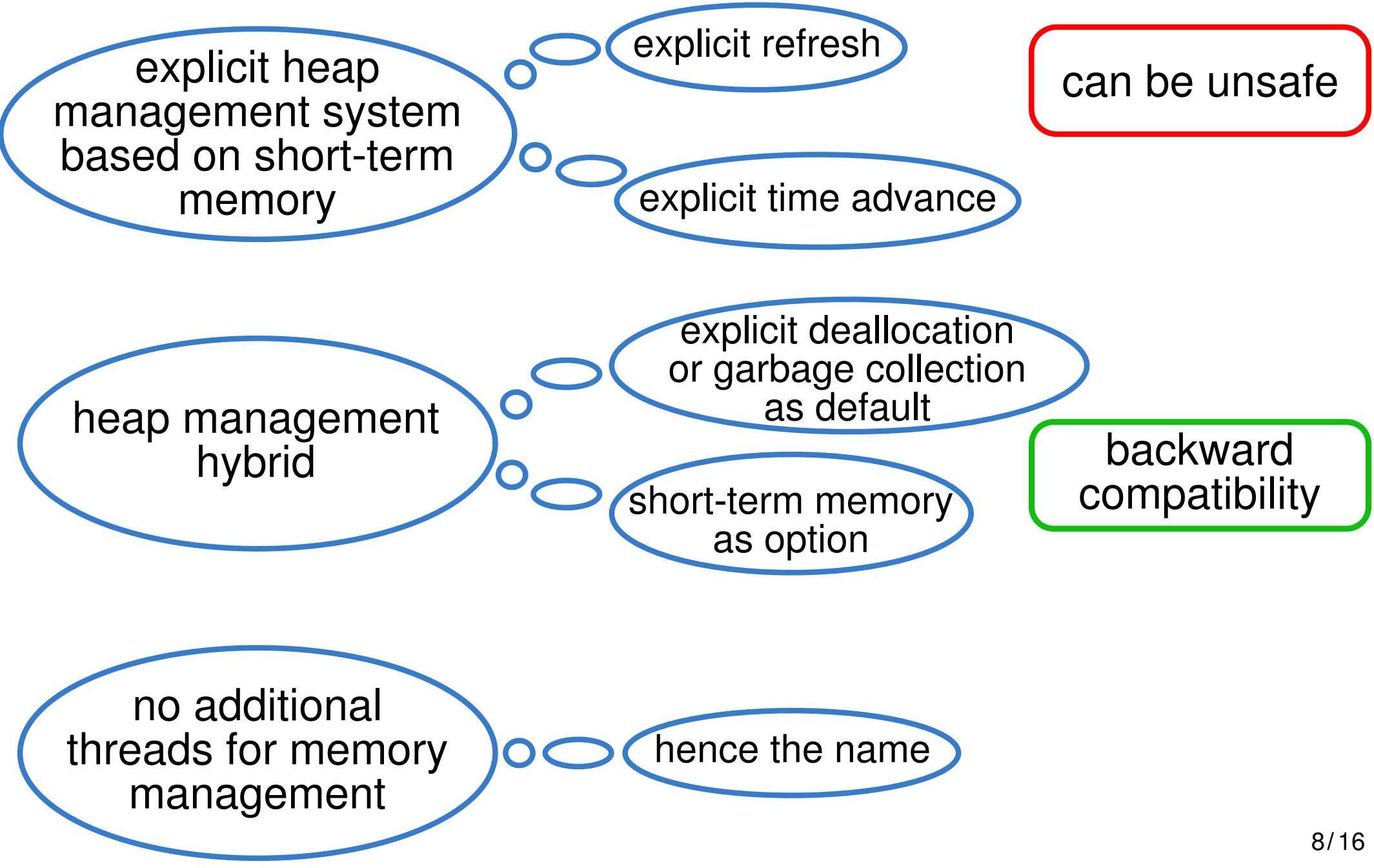
Self-collecting Mutators



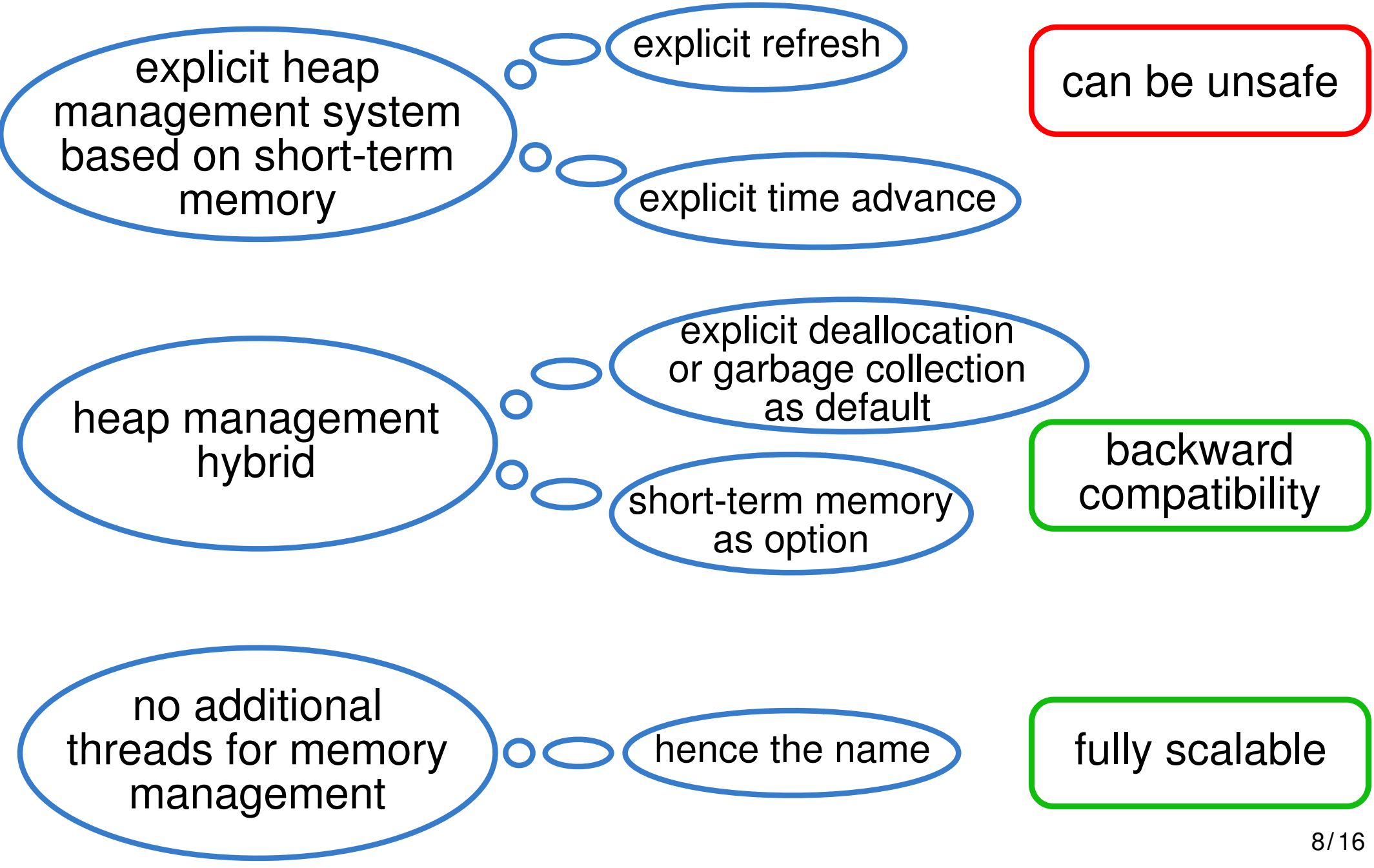
Self-collecting Mutators



Self-collecting Mutators



Self-collecting Mutators

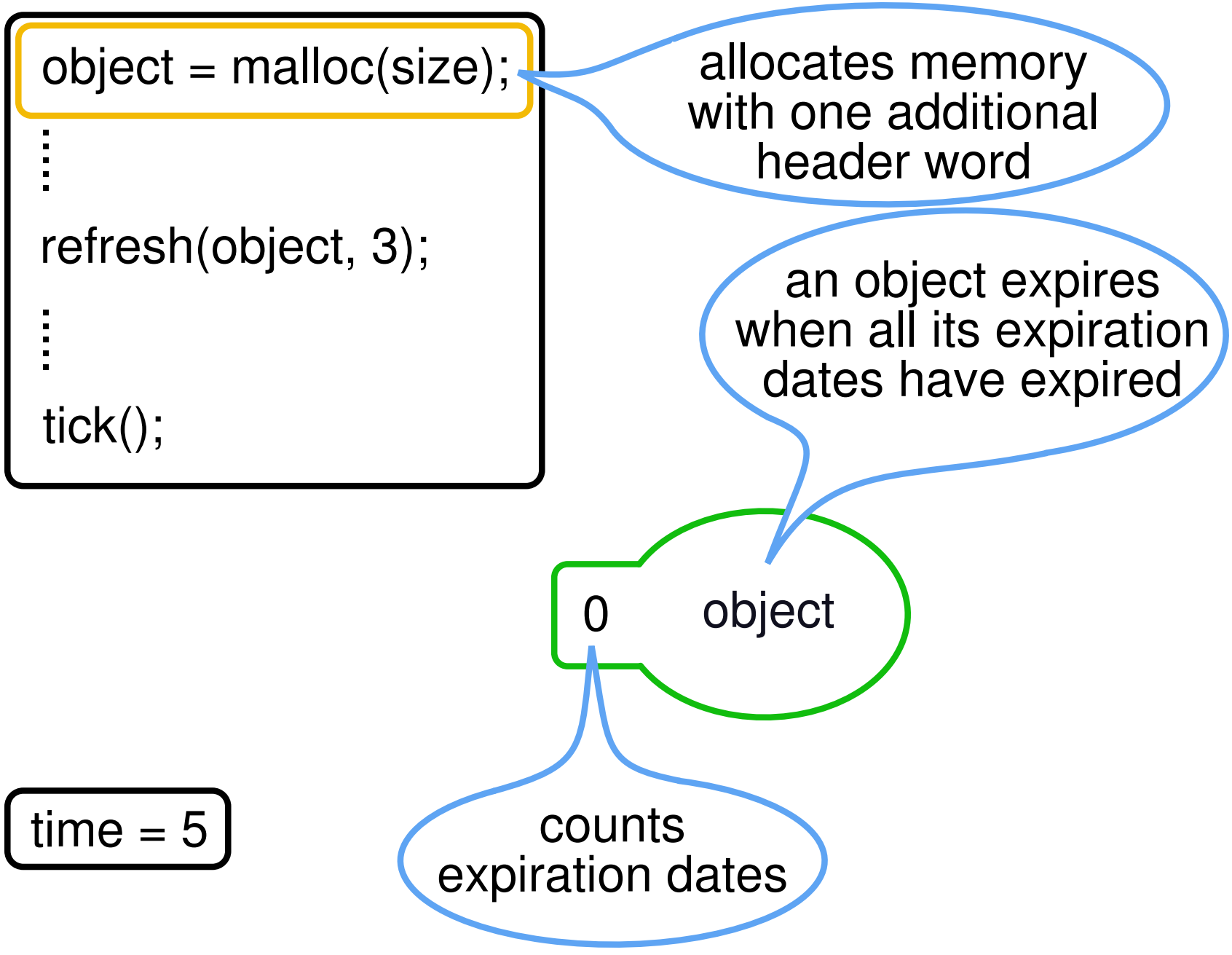


Programming Model

```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```

time = 5

Programming Model



Programming Model

```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```

hybrid heap management: the object can be deallocated here explicitly

0 object

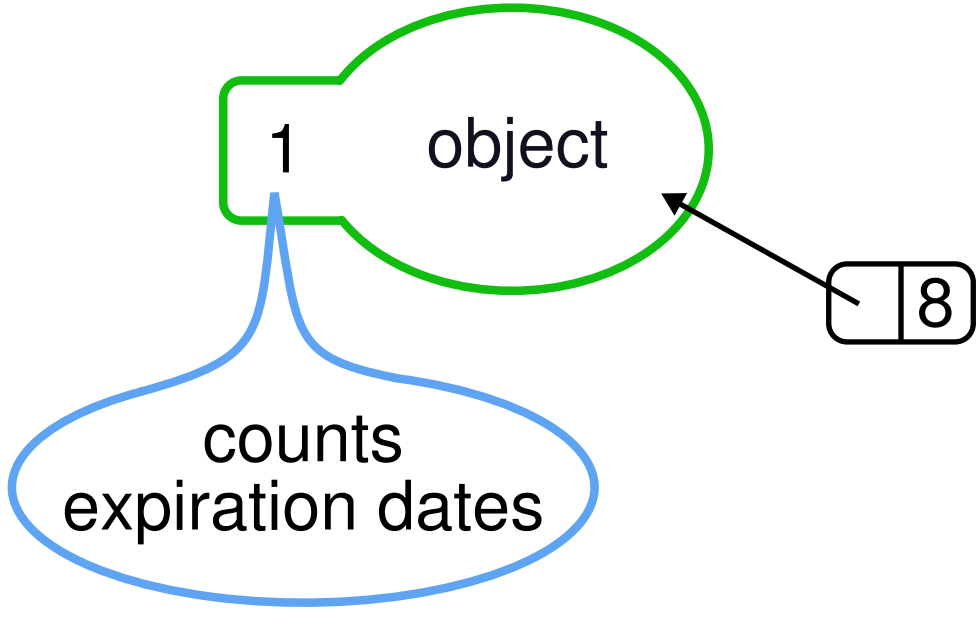
counts expiration dates

time = 5

Programming Model

```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```

creates an expiration date for the object

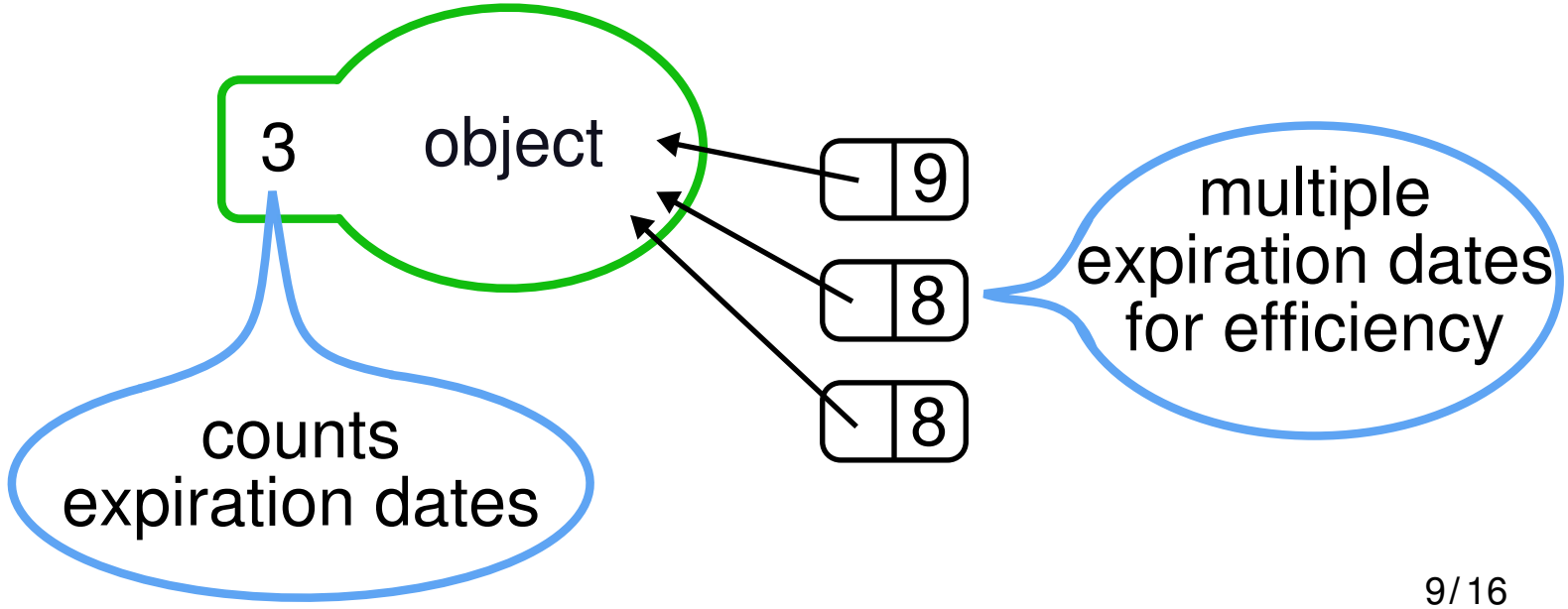


time = 5

Programming Model

```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```

creates an expiration date for the object

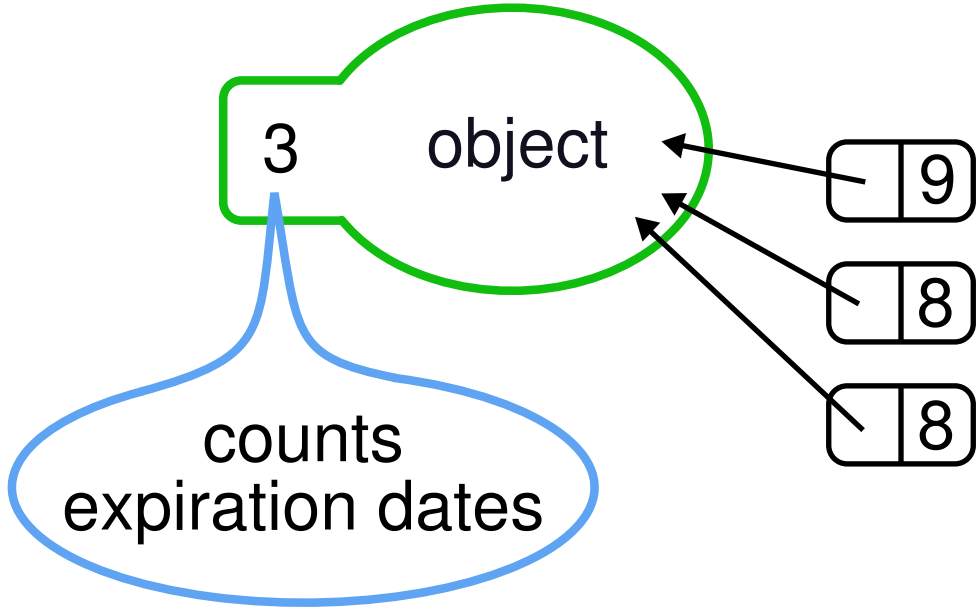


time = 5

Programming Model

```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```

object is short-term:
deallocation-calls
are ignored



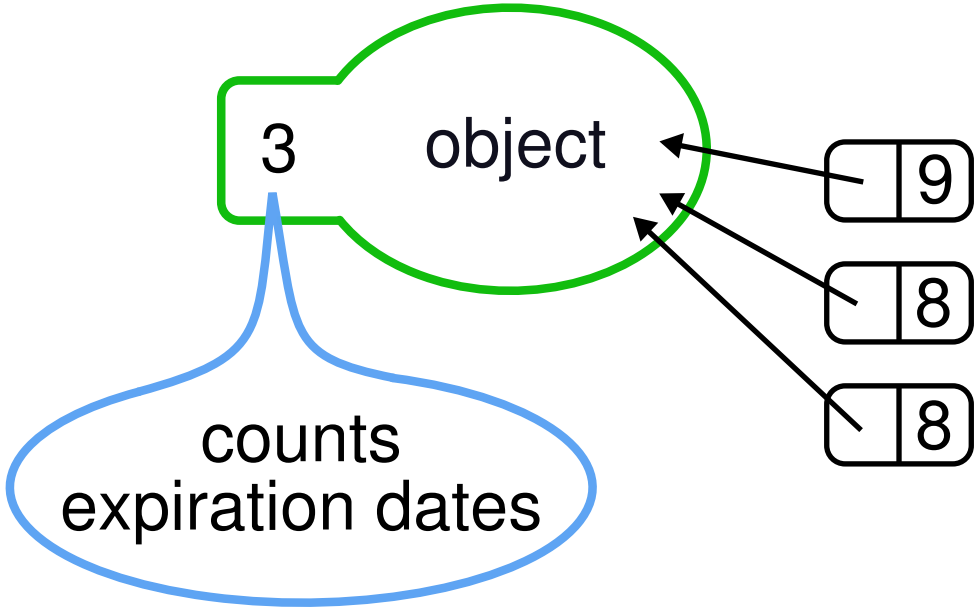
time = 5

Programming Model

```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```

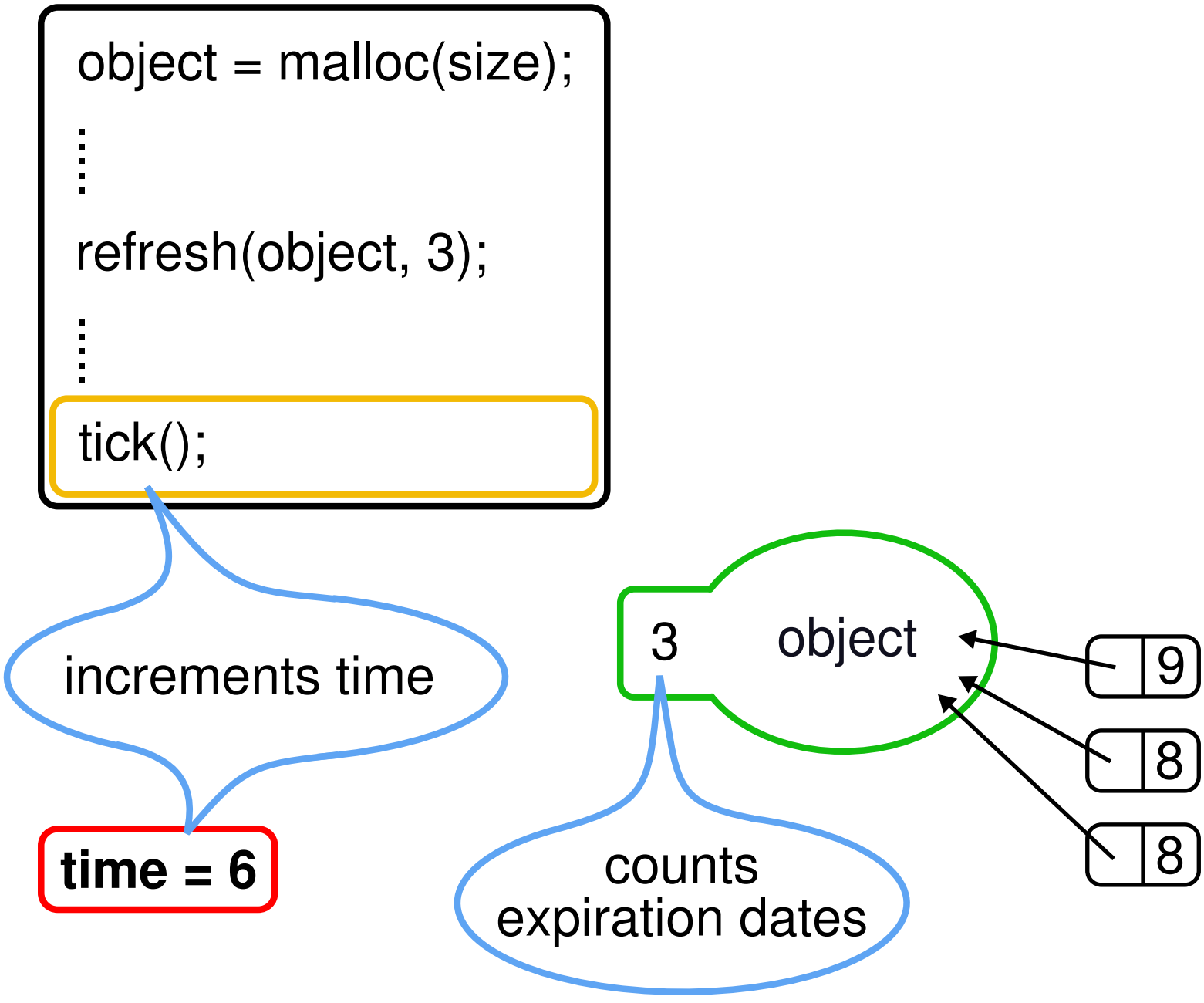
port an application without removing free-calls

object is short-term: deallocation-calls are ignored



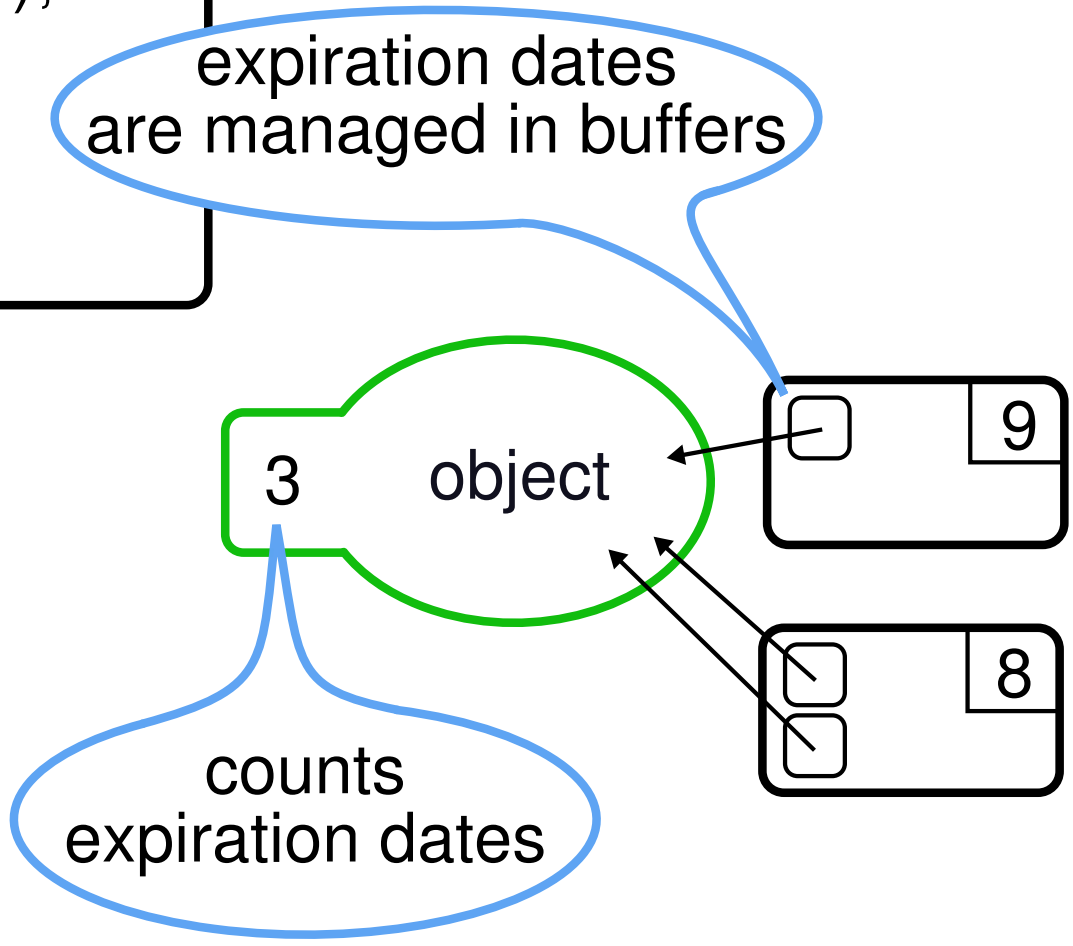
time = 5

Programming Model



Programming Model

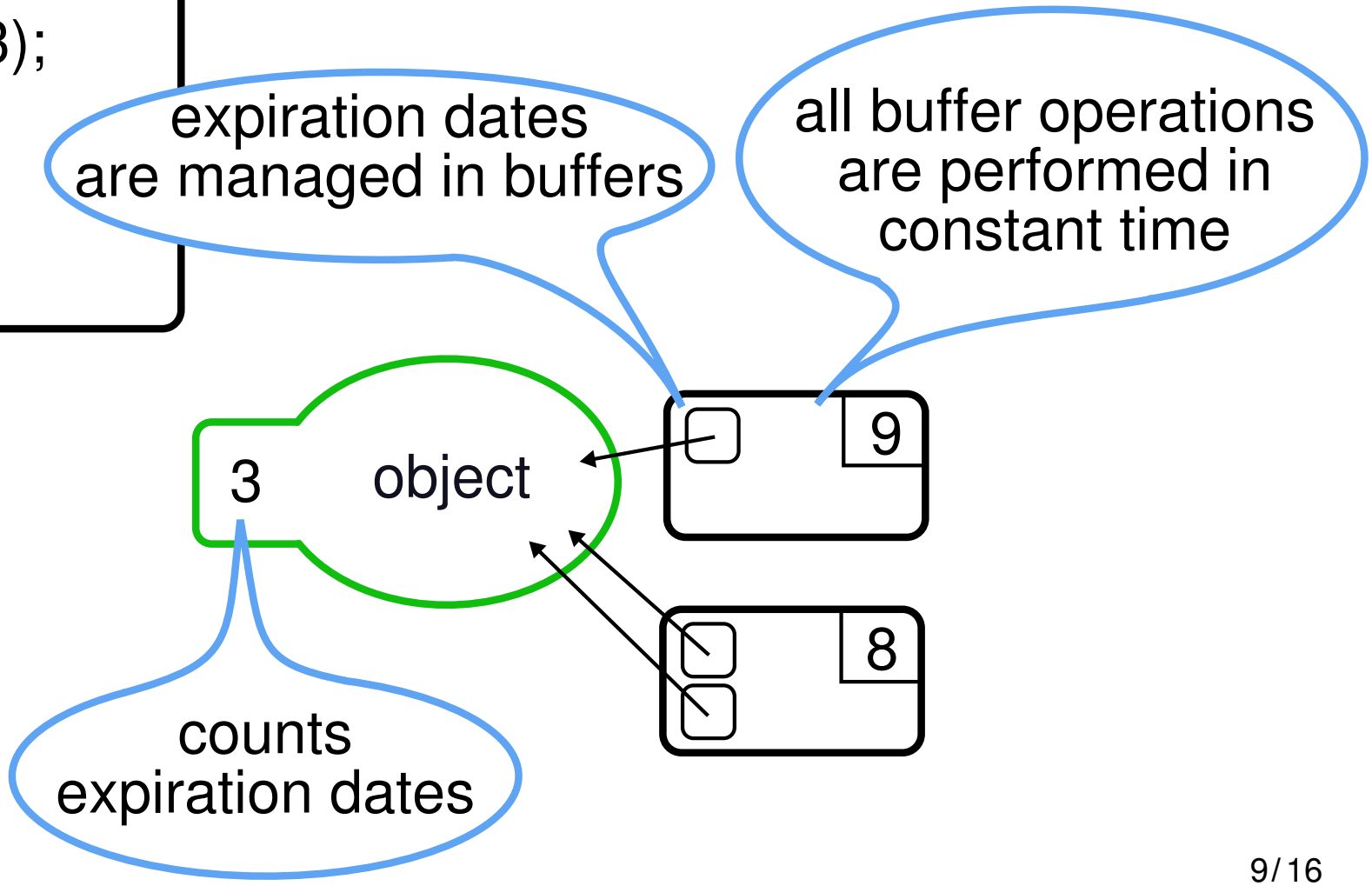
```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```



time = 6

Programming Model

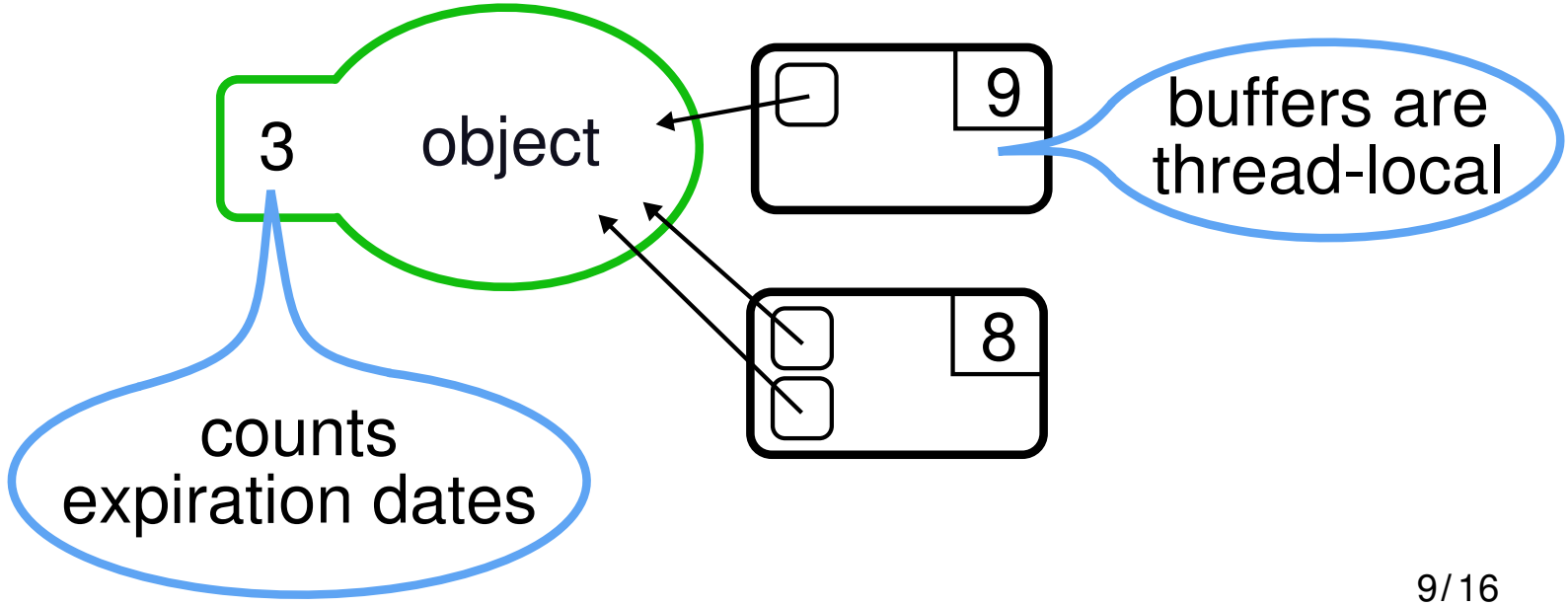
```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```



time = 6

Programming Model

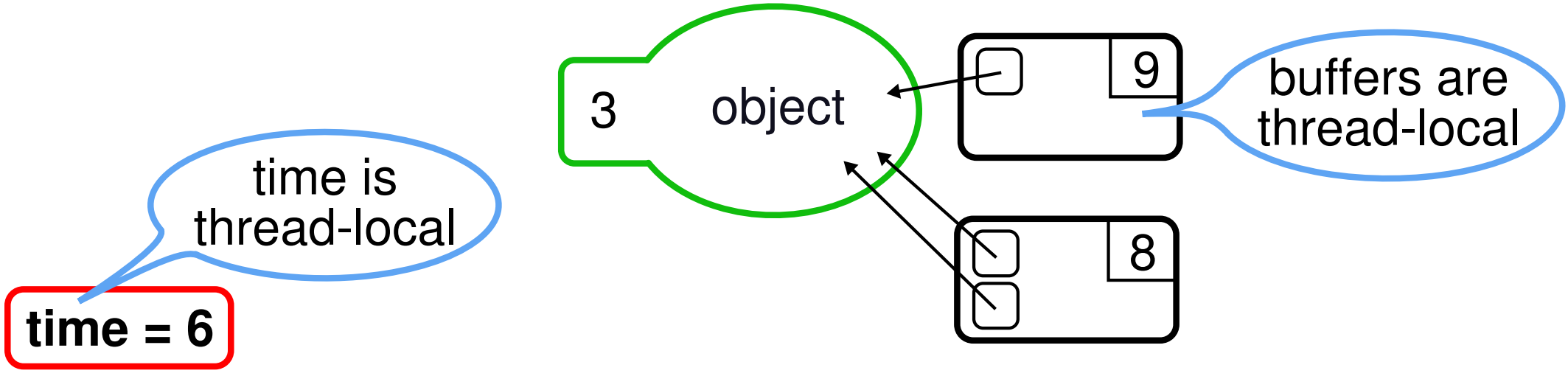
```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```



time = 6

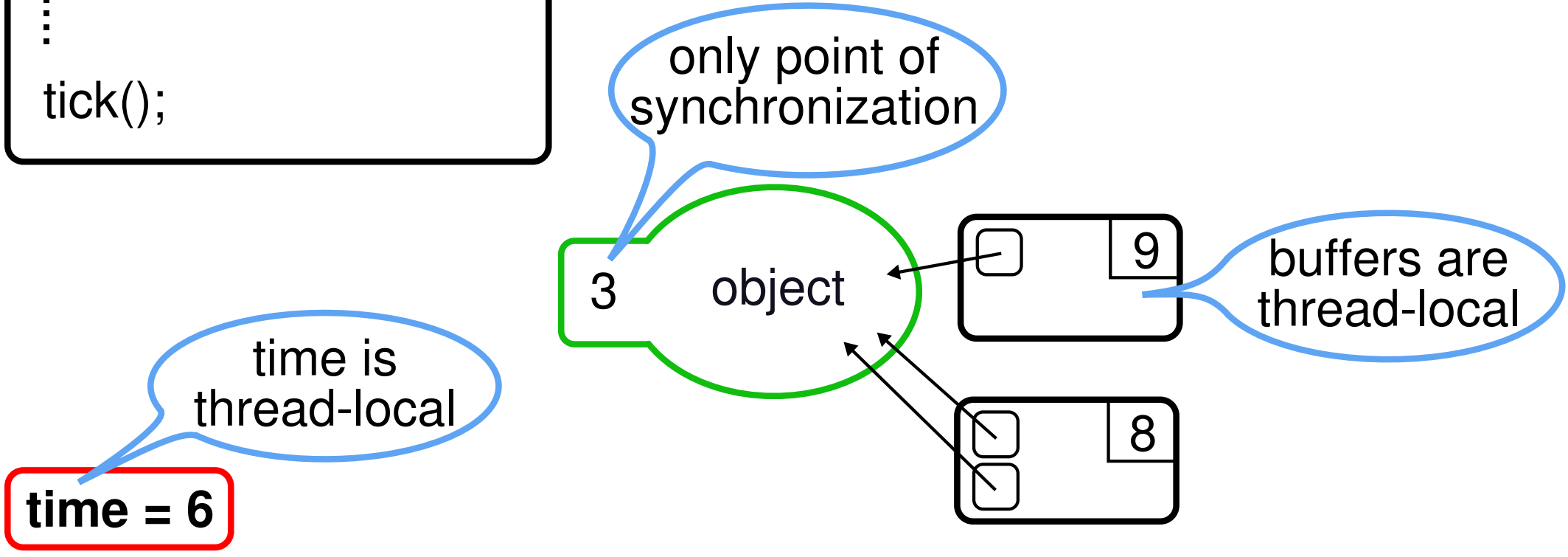
Programming Model

```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```

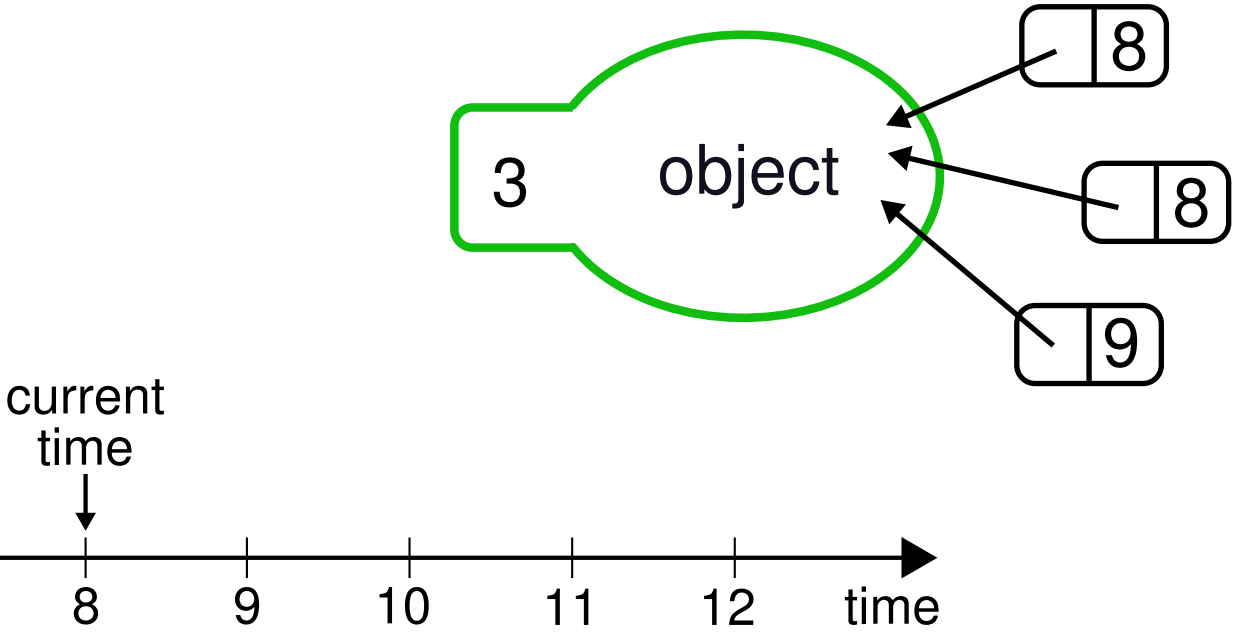


Programming Model

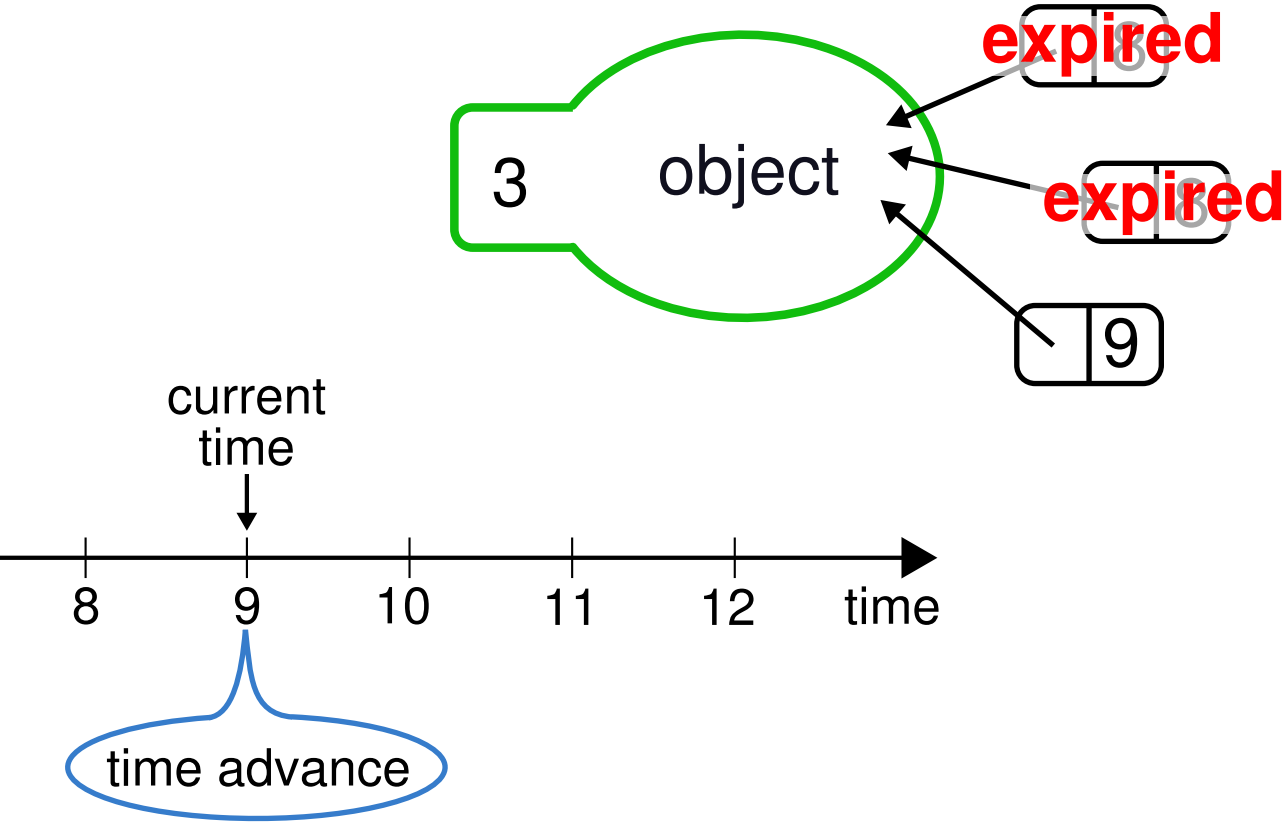
```
object = malloc(size);
⋮
refresh(object, 3);
⋮
tick();
```



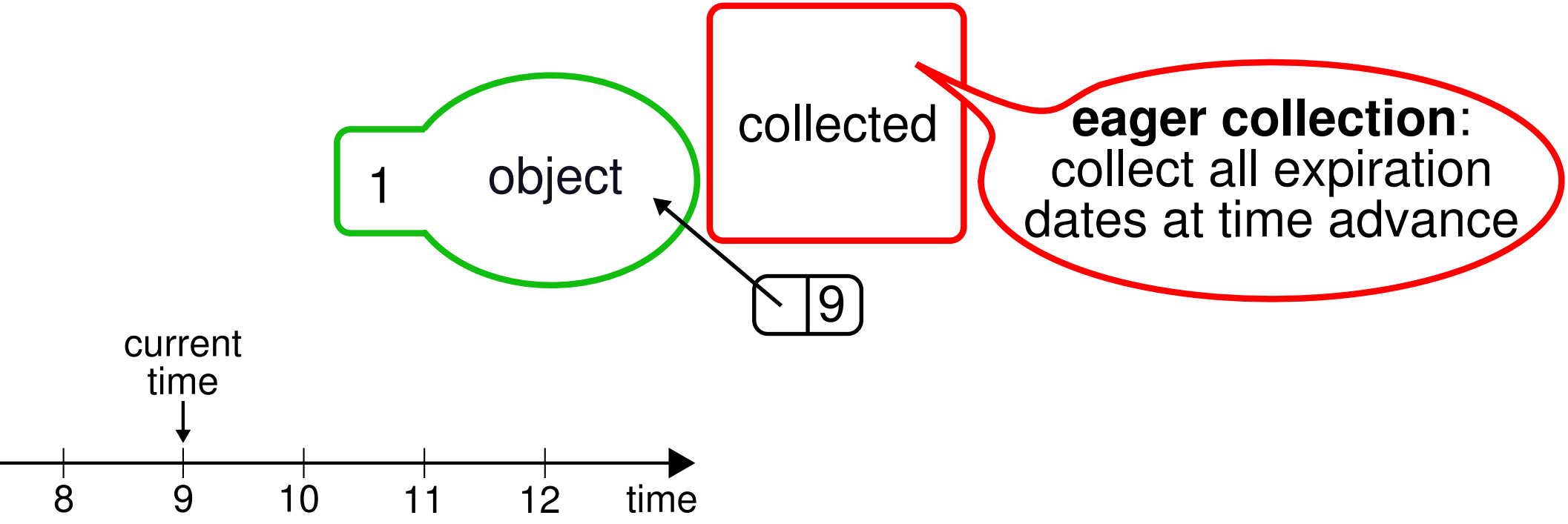
Collection of Expired Objects



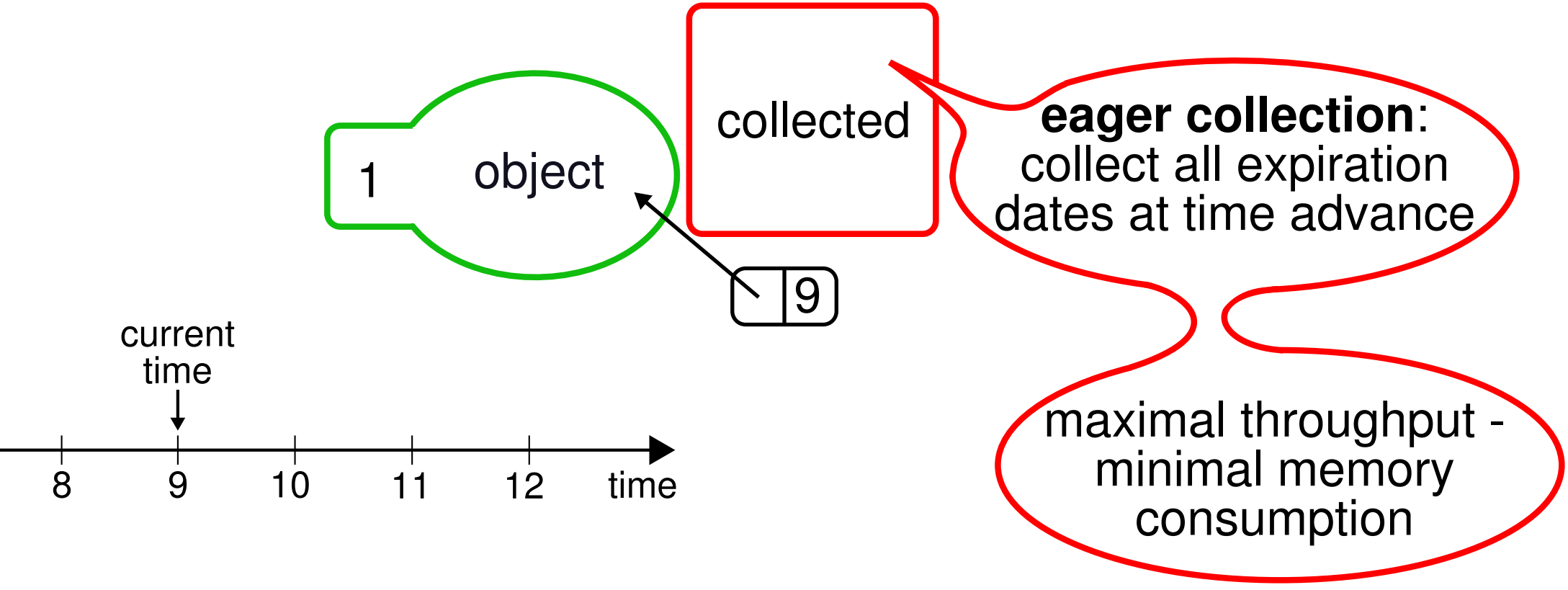
Collection of Expired Objects



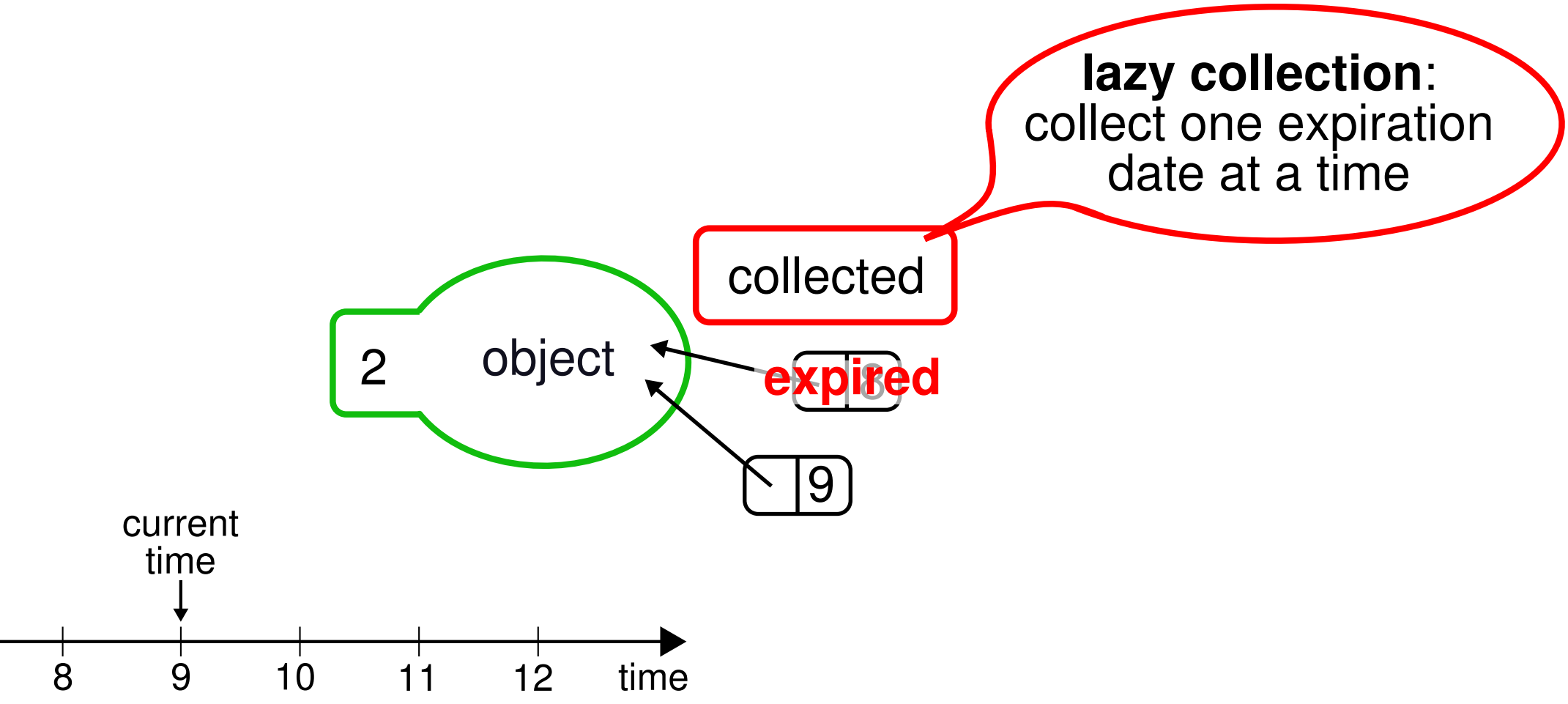
Collection of Expired Objects



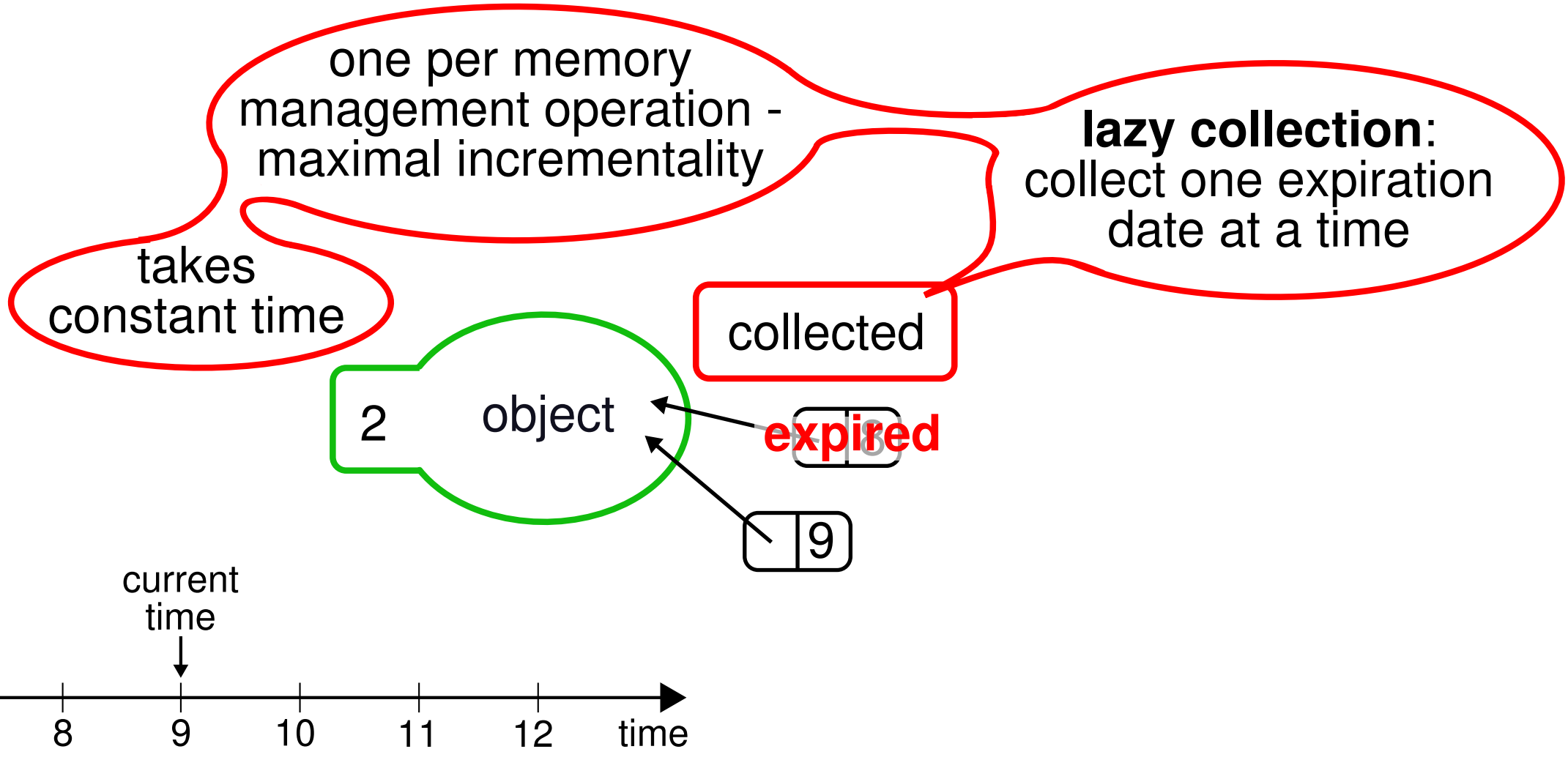
Collection of Expired Objects



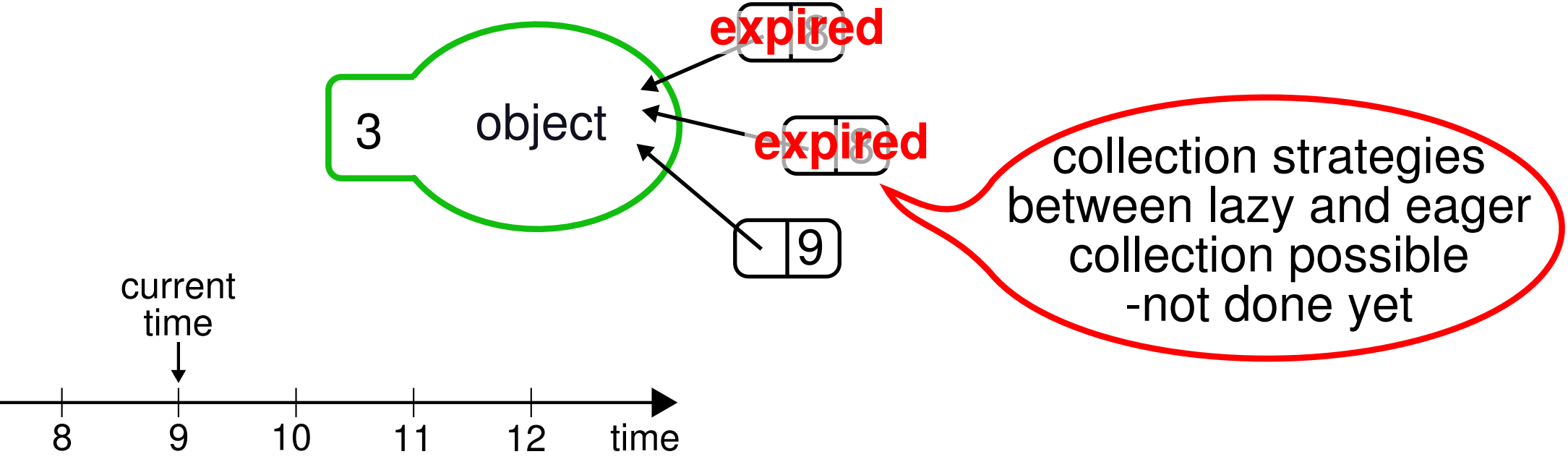
Collection of Expired Objects



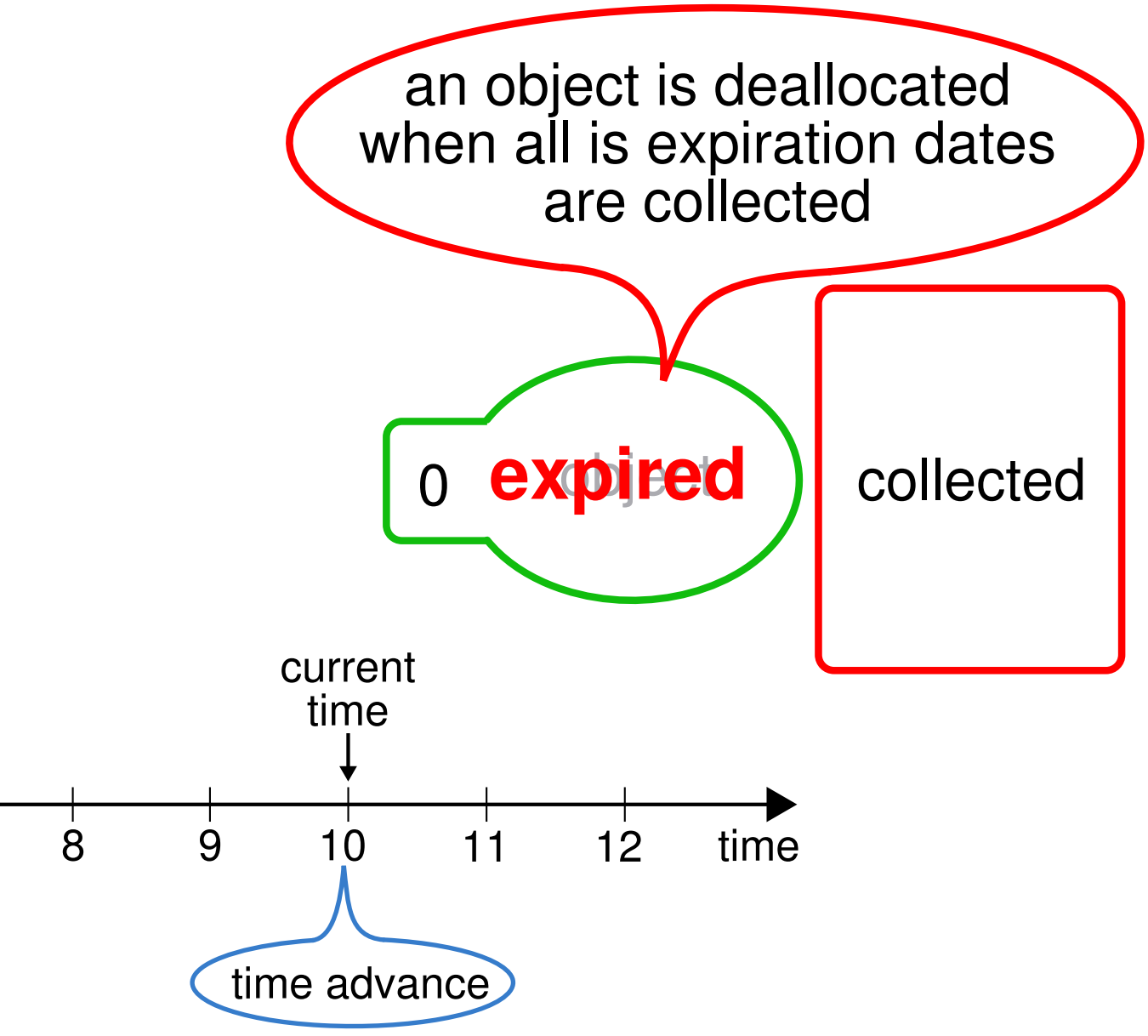
Collection of Expired Objects



Collection of Expired Objects



Collection of Expired Objects



Implementation and Evaluation

Java
in Jikes RVM

C
based on ptmalloc2

Go
in the 6g runtime

Implementation and Evaluation

Java
in Jikes RVM

Monte Carlo

C
based on ptmalloc2

mpg123
x264

Go
in the 6g runtime

tree benchmark
webserver

Implementation and Evaluation

Java
in Jikes RVM

Monte Carlo

C
based on ptmalloc2

mpg123
x264

Go
in the 6g runtime

tree benchmark
webserver

performance

- total execution time (throughput)
- execution time per operation (latency)
- memory consumption

ease of use

- lines of code
- difficulty of placing code

Implementation and Evaluation

Java
in Jikes RVM

Monte Carlo

- 1450 LoC
- 1 tick-call
- refresh all short-living objects right after allocation

C
based on ptmalloc2

mpg123
x264

Go
in the 6g runtime

tree benchmark
webserver

Implementation and Evaluation

Java
in Jikes RVM

Monte Carlo

C
based on ptmalloc2

mpg123
x264

Go
in the 6g runtime

tree benchmark
webserver

- 1450 LoC
- 1 tick-call
- refresh all short-living objects right after allocation

to make all
short-living objects
short-term

Implementation and Evaluation

Java
in Jikes RVM

Monte Carlo

C
based on ptmalloc2

mpg123
x264

- 16043 LoC
- 1 tick-call
- refresh all objects right after allocation

Go
in the 6g runtime

tree benchmark
webserver

Implementation and Evaluation

Java
in Jikes RVM

Monte Carlo

C
based on ptmalloc2

mpg123
x264

- 61722 LoC
- 1 tick-call

Go
in the 6g runtime

tree benchmark
webserver

or

- **single refresh**: refresh all frames after allocation with a conservative expiration date
- **continuous refresh**: refresh all needed frames before time advance with extension '1'

Implementation and Evaluation

Java
in Jikes RVM

Monte Carlo

C
based on ptmalloc2

mpg123
x264

- 61722 LoC
- 1 tick-call

Go
in the 6g runtime

tree benchmark
webserver

2 tick-calls in the multi- threaded version: in the paper

or

- **single refresh**: refresh all frames after allocation with a conservative expiration date
- **continuous refresh**: refresh all needed frames before time advance with extension '1'

Implementation and Evaluation

Java
in Jikes RVM

Monte Carlo

C
based on ptmalloc2

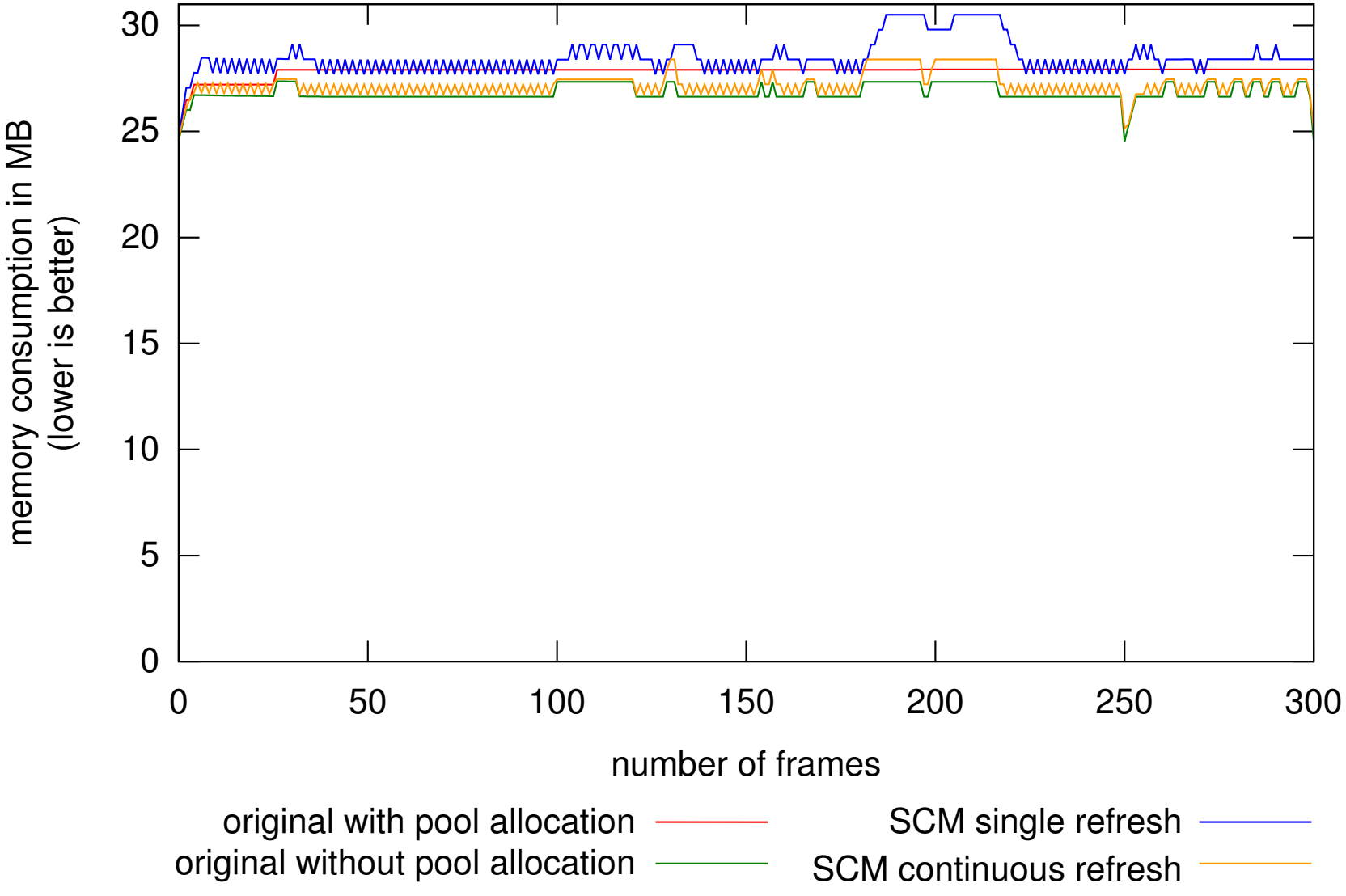
mpg123
x264

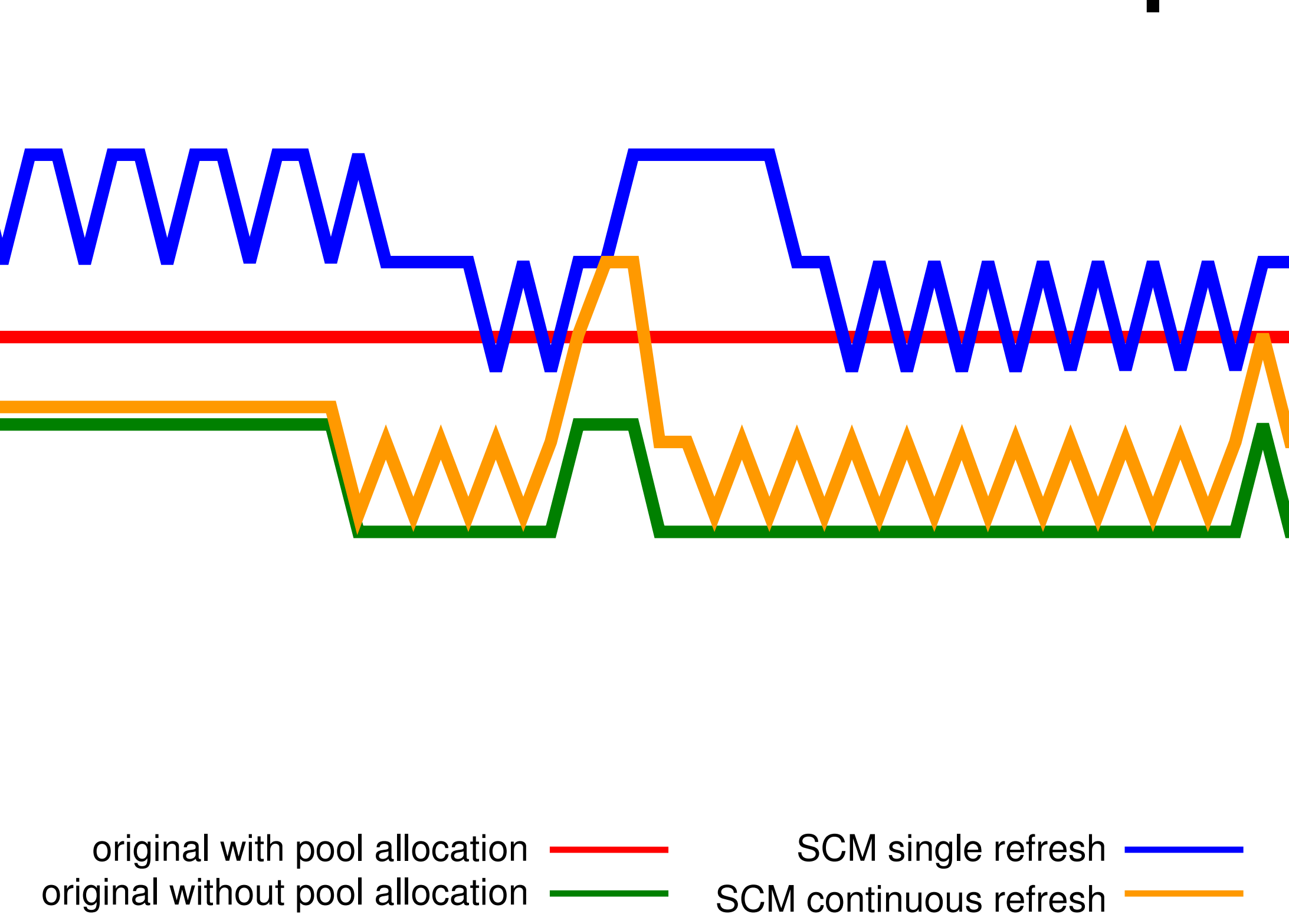
Go
in the 6g runtime

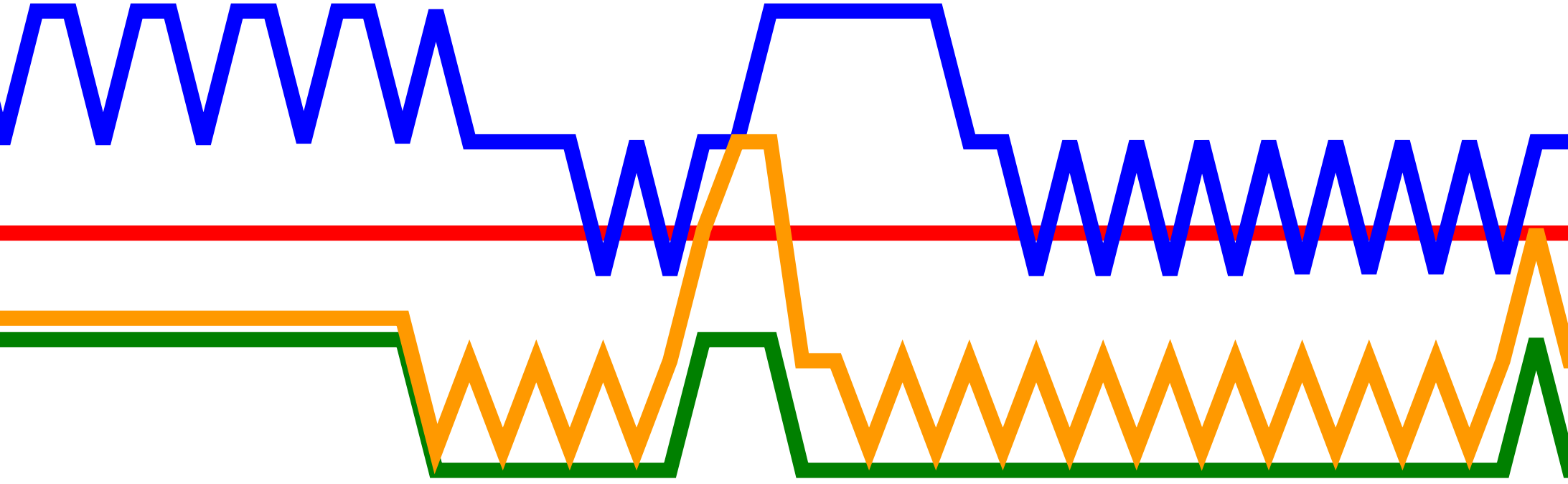
tree benchmark
webserver

in the paper

Memory Consumption: x264




original with pool allocation
original without pool allocation
SCM single refresh
SCM continuous refresh



- Bounded memory overhead
- Easier to use than explicit deallocation (no reference counting)

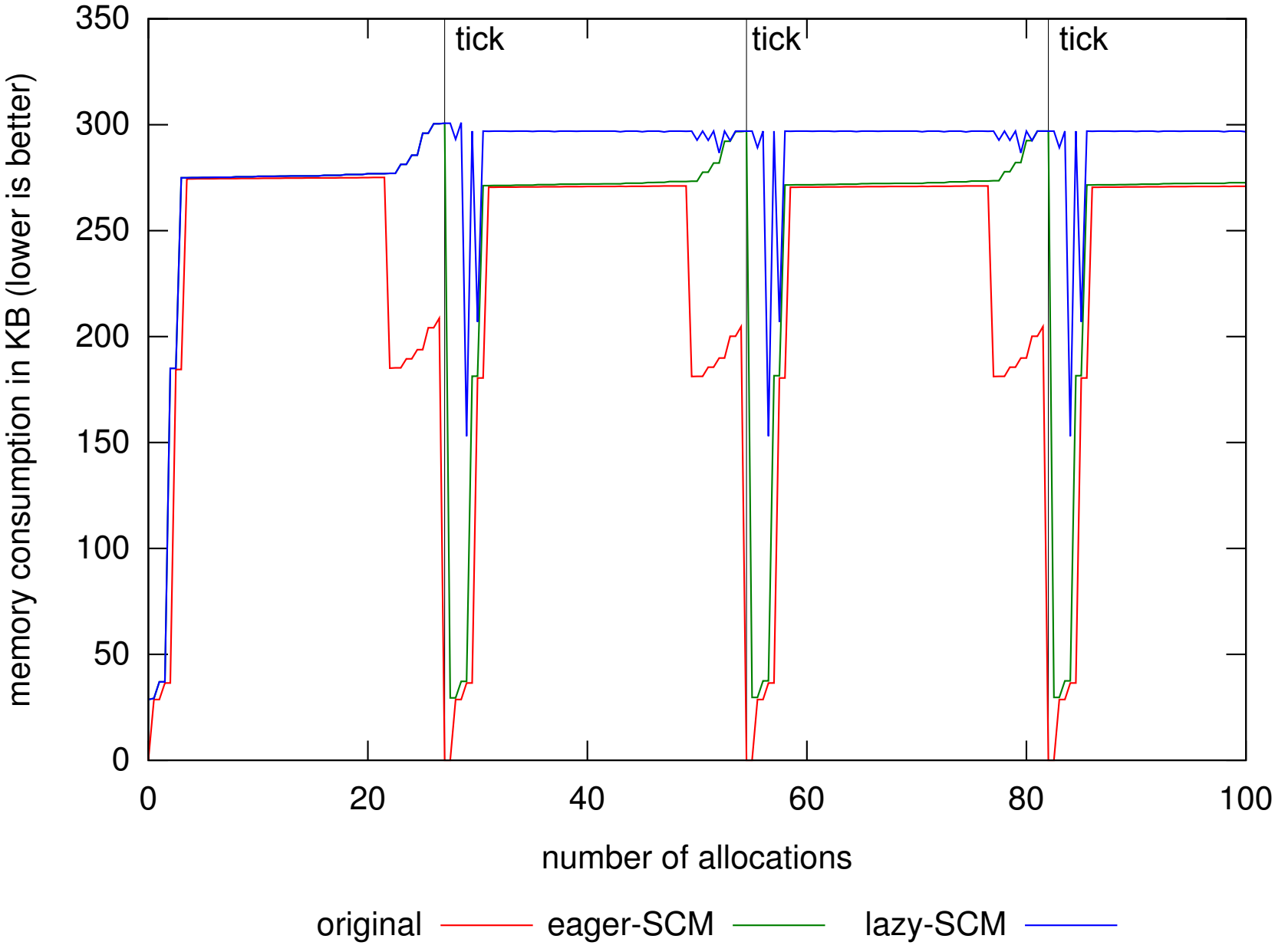
original with pool allocation

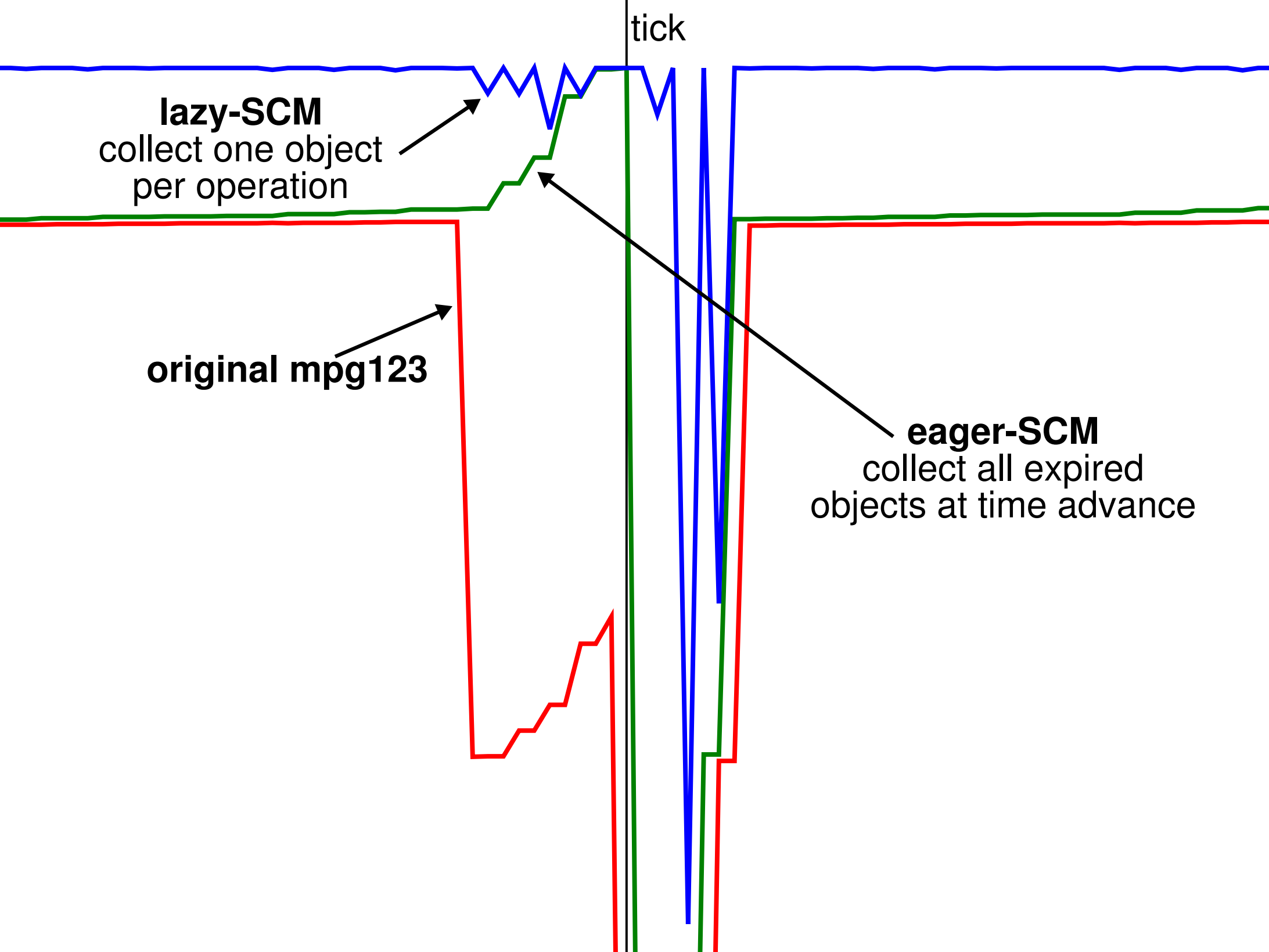
original without pool allocation

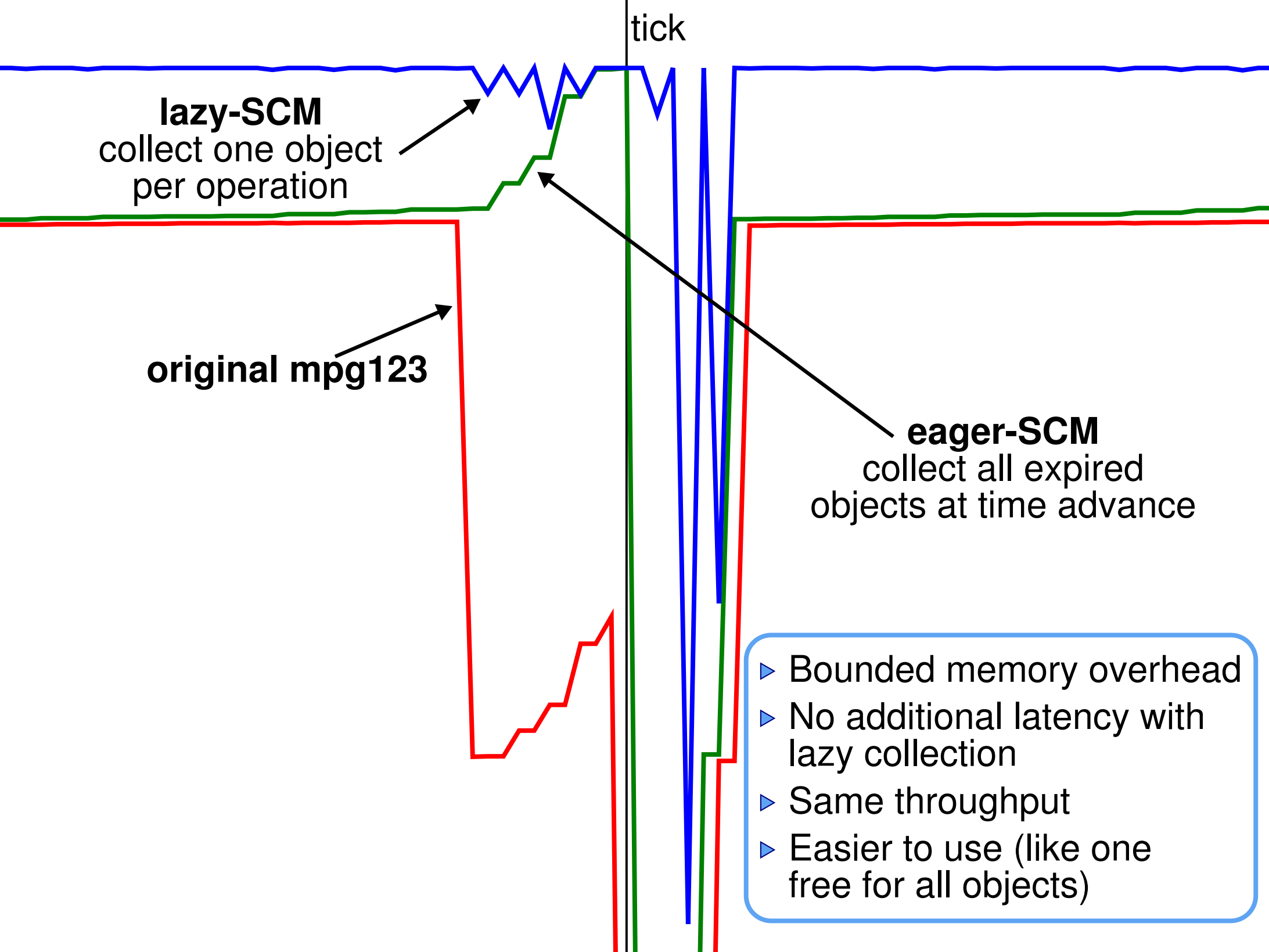
SCM single refresh

SCM continuous refresh

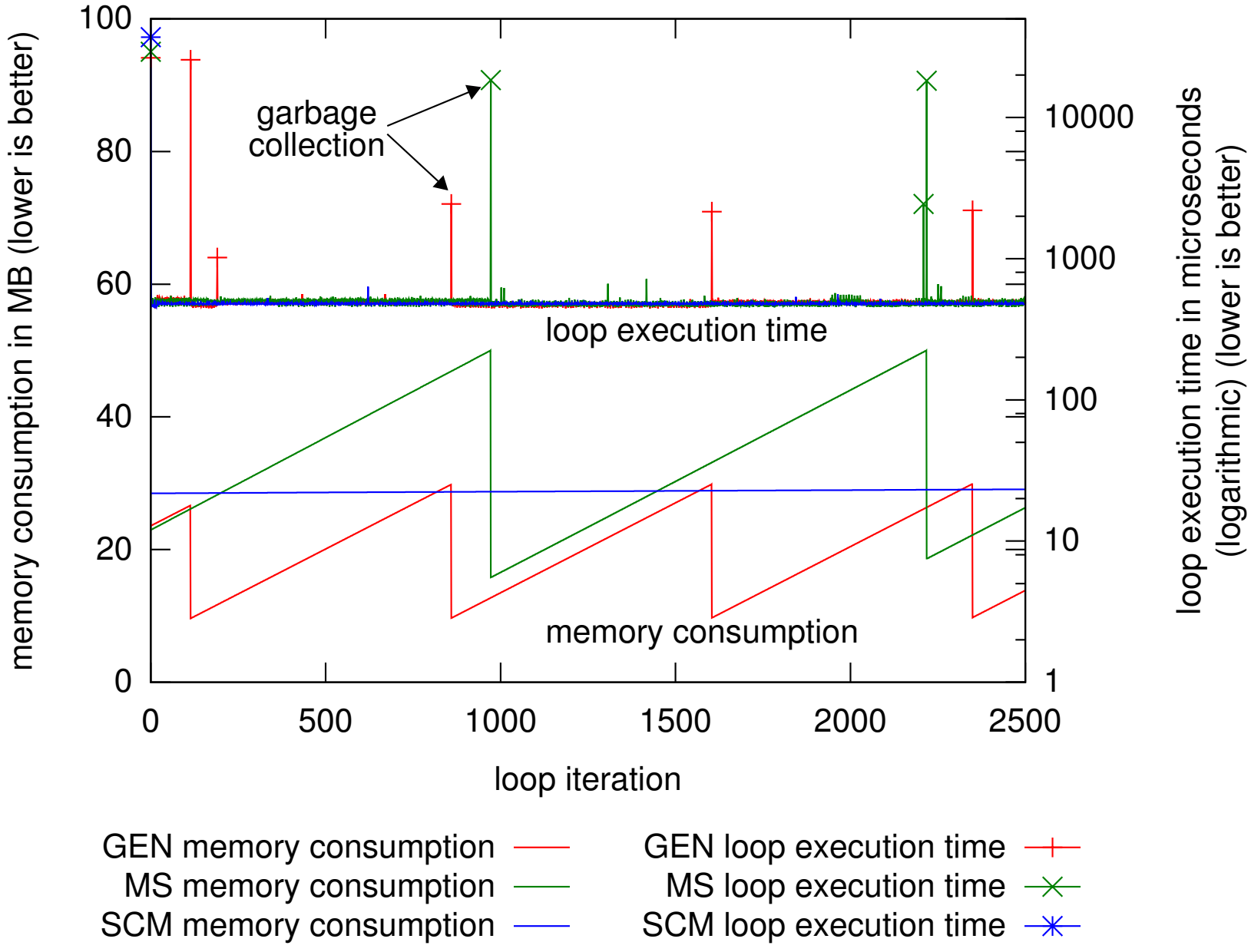
Lazy vs Eager Collection: mpg123

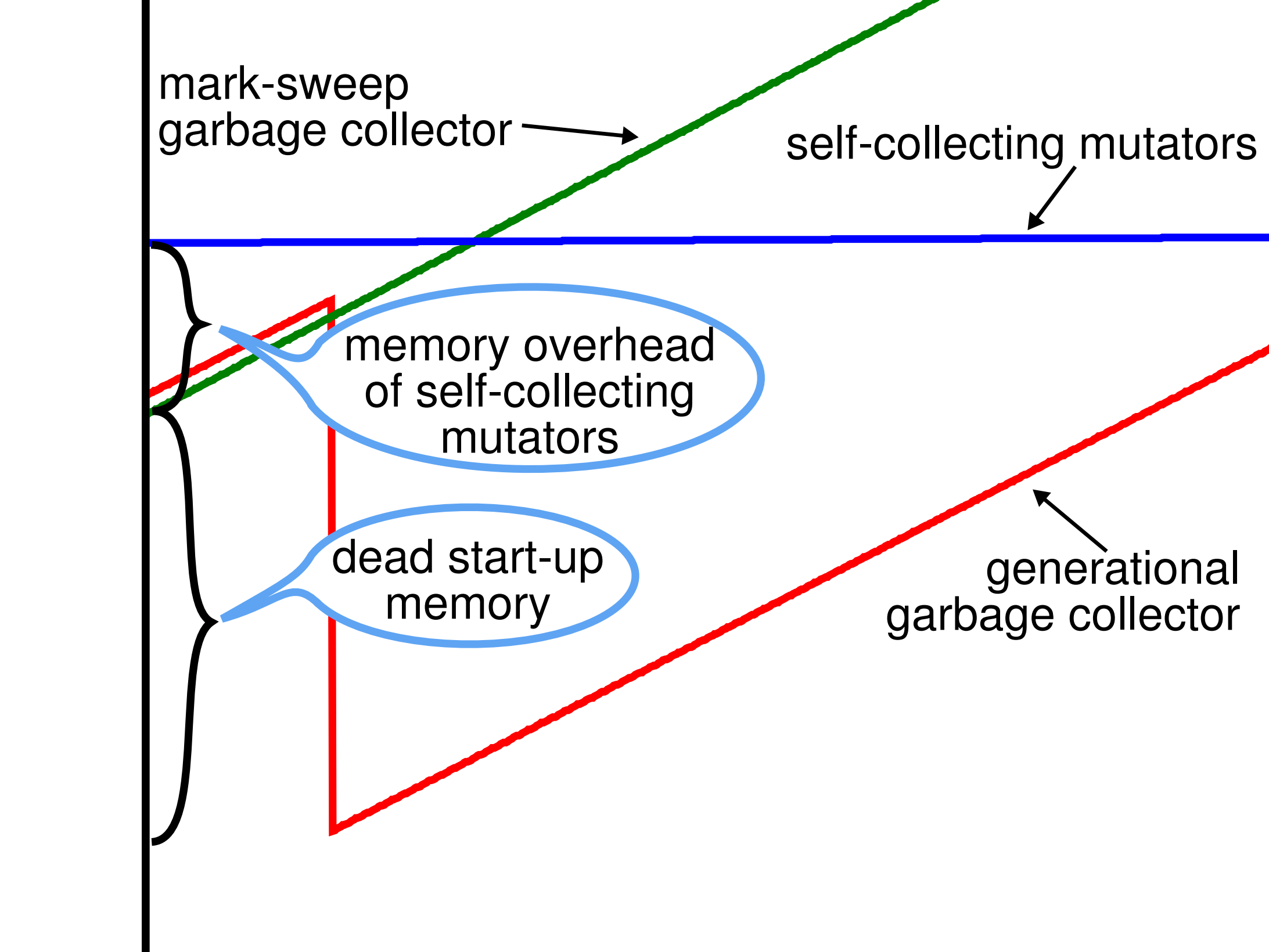


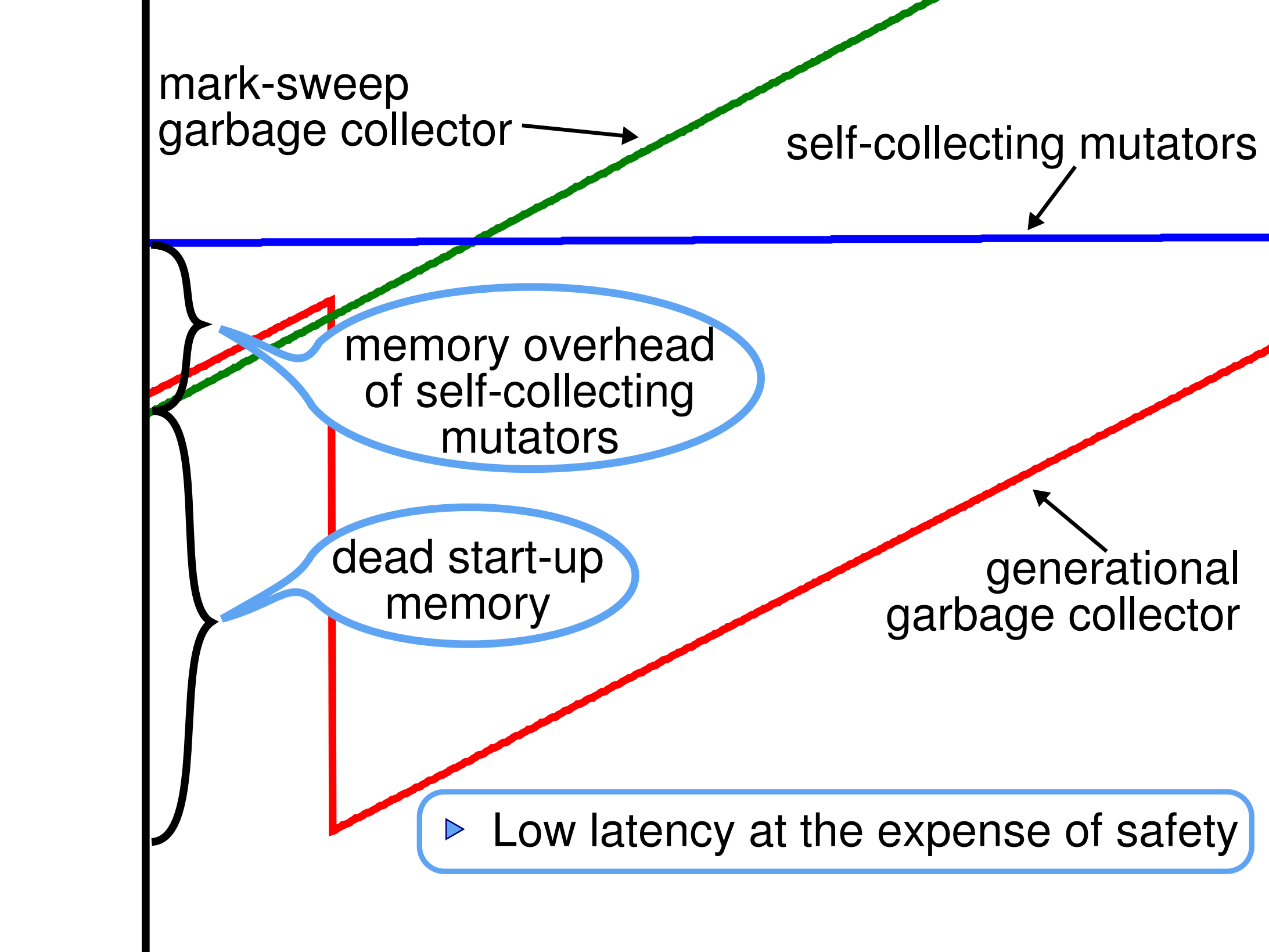

tick
lazy-SCM
collect one object
per operation
original mpg123
eager-SCM
collect all expired
objects at time advance


tick
lazy-SCM
collect one object
per operation
original mpg123
eager-SCM
collect all expired
objects at time advance
Bounded memory overhead
No additional latency with lazy collection
Same throughput
Easier to use (like one free for all objects)

Latency: Monte Carlo




mark-sweep
garbage collector
self-collecting mutators
memory overhead
of self-collecting
mutators
dead start-up
memory
generational
garbage collector


mark-sweep
garbage collector
self-collecting mutators
memory overhead
of self-collecting
mutators
dead start-up
memory
generational
garbage collector
Low latency at the expense of safety

Conclusion

- Get the code from here:
  - http://tiptoe.cs.uni-salzburg.at/short-term-memory/
- Future work (memory management community)
  - **Efficiency**: integrate short-term memory into other allocators
  - **Correctness**: make short-term memory safe
- Future work (embedded software community)
  - **Time as first class citizen**: capture execution progress relevant for memory management

Thank You

http://tiptoe.cs.uni-salzburg.at/short-term-memory/