Data-Race Exceptions Have Benefits Beyond the Memory Model

Benjamin P. Wood, Luis Ceze, Dan Grossman

University of Washington

Safe MultiProcessing Architectures at the University of Washington

Why data-race exceptions?

[Elmas et al., PLDI 2007; Adve and Boehm, CACM Aug. 2010;
Marino et al., PLDI 2010; Lucia et al., ISCA 2010; ...]

Why data-race exceptions?

Find bugs.

[Elmas et al., PLDI 2007; Adve and Boehm, CACM Aug. 2010;
Marino et al., PLDI 2010; Lucia et al., ISCA 2010; ...]

Why data-race exceptions?

Find bugs.

Simplify memory models.

(DRF $\Rightarrow$ SC)

[Elmas et al., PLDI 2007; Adve and Boehm, CACM Aug. 2010;
Marino et al., PLDI 2010; Lucia et al., ISCA 2010; ...]

Why data-race exceptions?

Find bugs.

Simplify memory models.

(DRF ⇒ SC)

Avoid reasoning about memory reorderings.

[Elmas et al., PLDI 2007; Adve and Boehm, CACM Aug. 2010;
Marino et al., PLDI 2010; Lucia et al., ISCA 2010; ...]

Why *not* data-race exceptions?

Why *not* data-race exceptions?

Lock-free algorithms

Why *not* data-race exceptions?

"Benign" races

Lock-free algorithms

Why *not* data-race exceptions?

~~“Benign” races~~

~~Lock-free algorithms~~

unchecked **annotations**

Why *not* data-race exceptions?

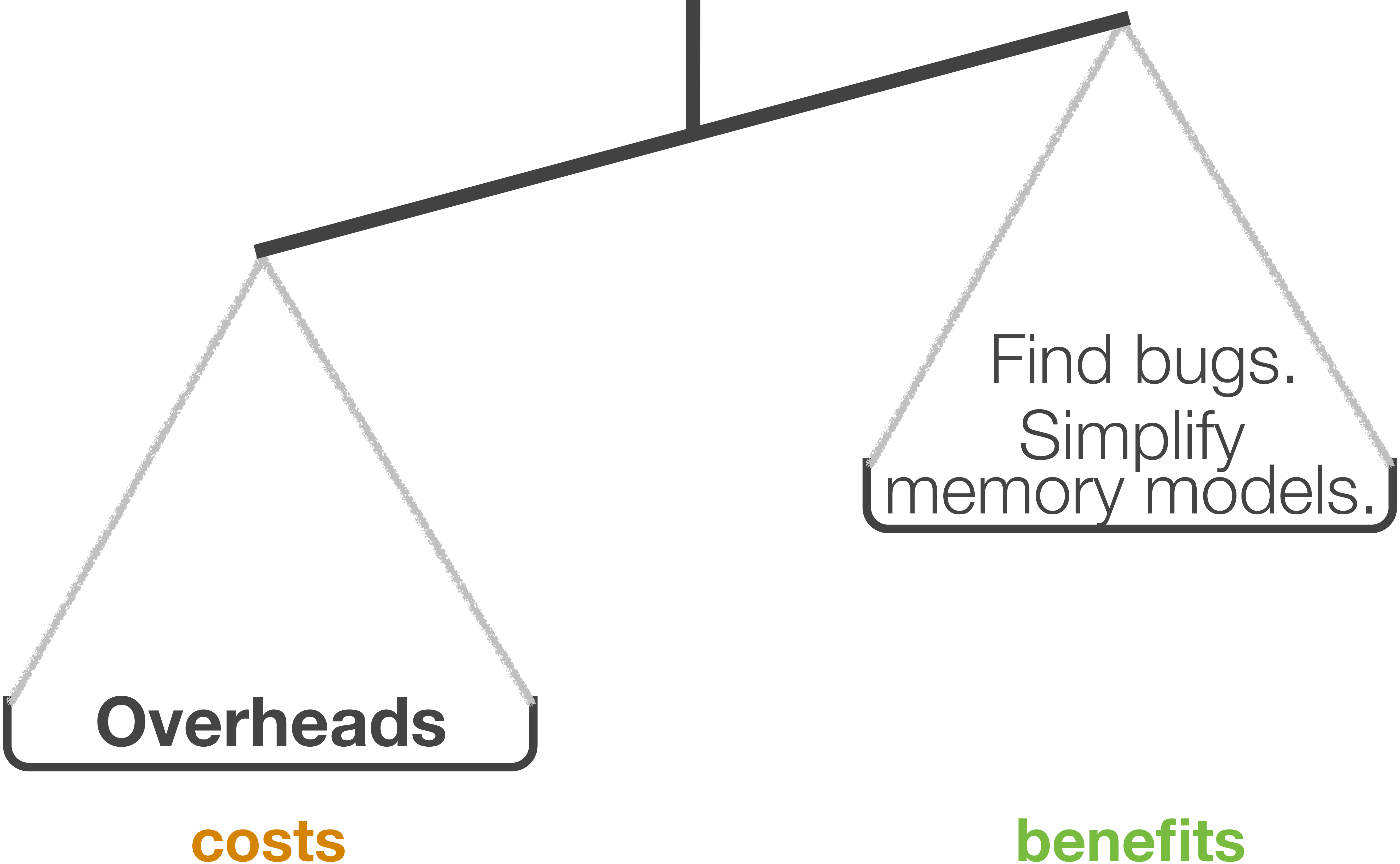
~~“Benign” races~~

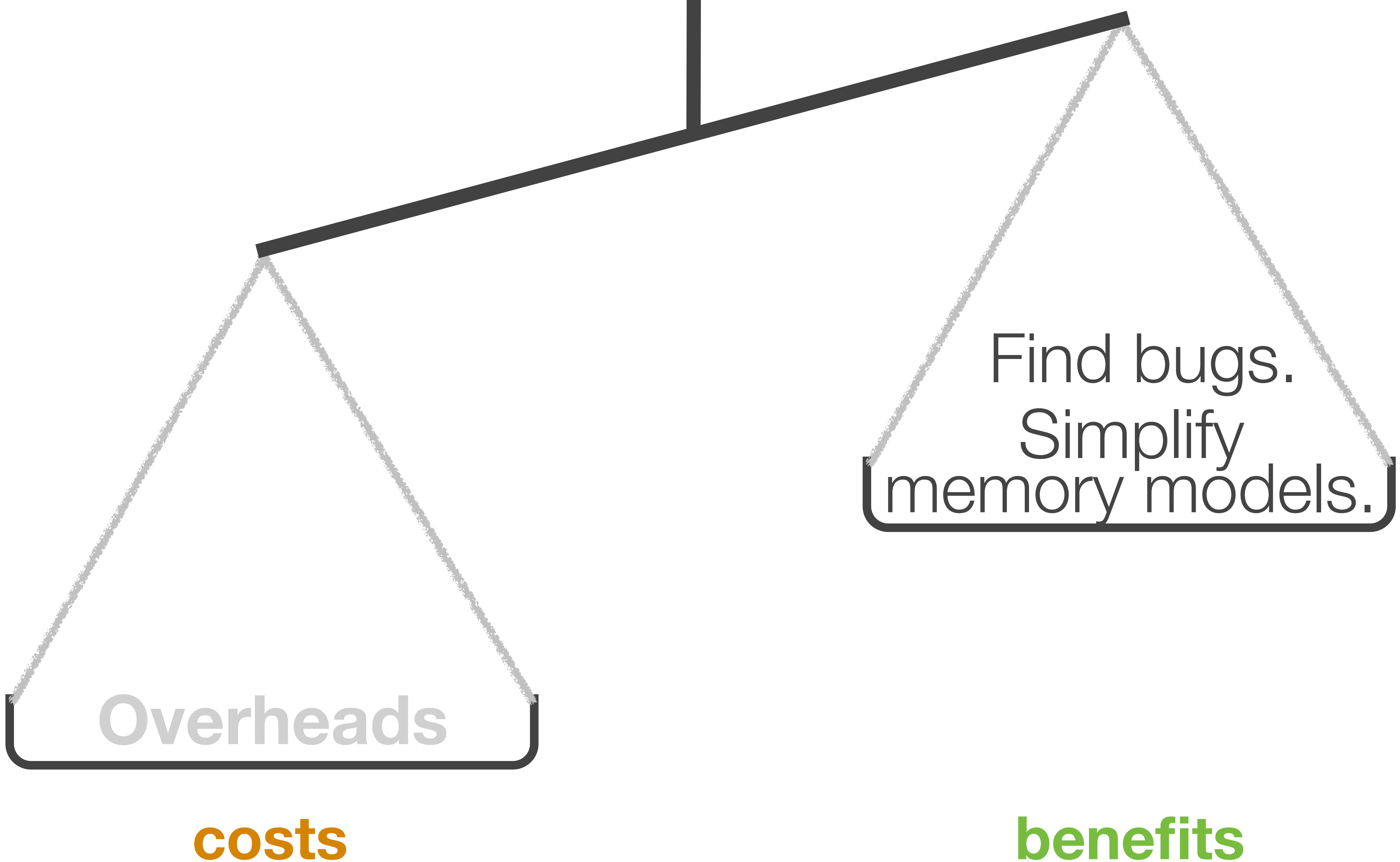
~~Lock-free algorithms~~

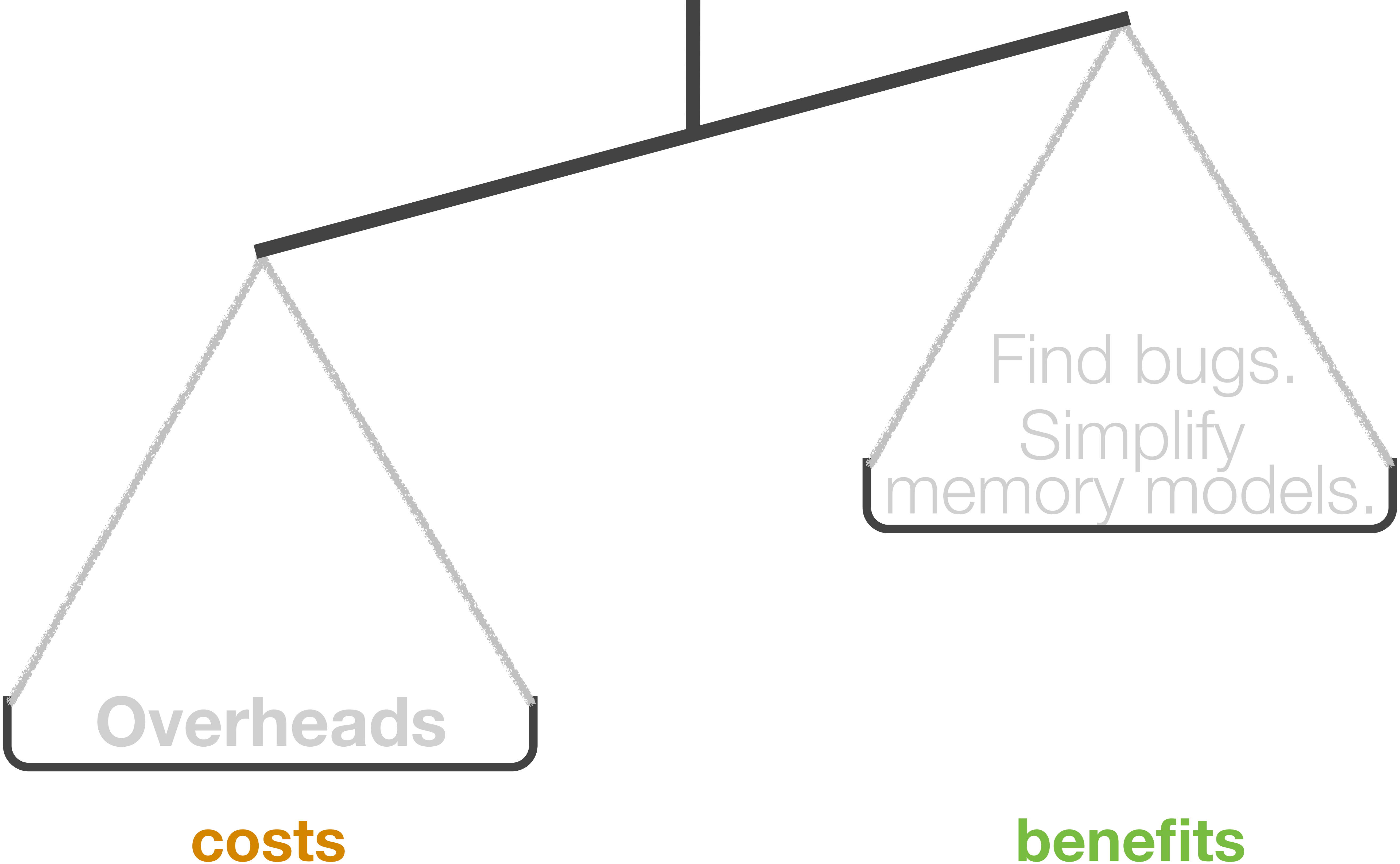
unchecked **annotations**

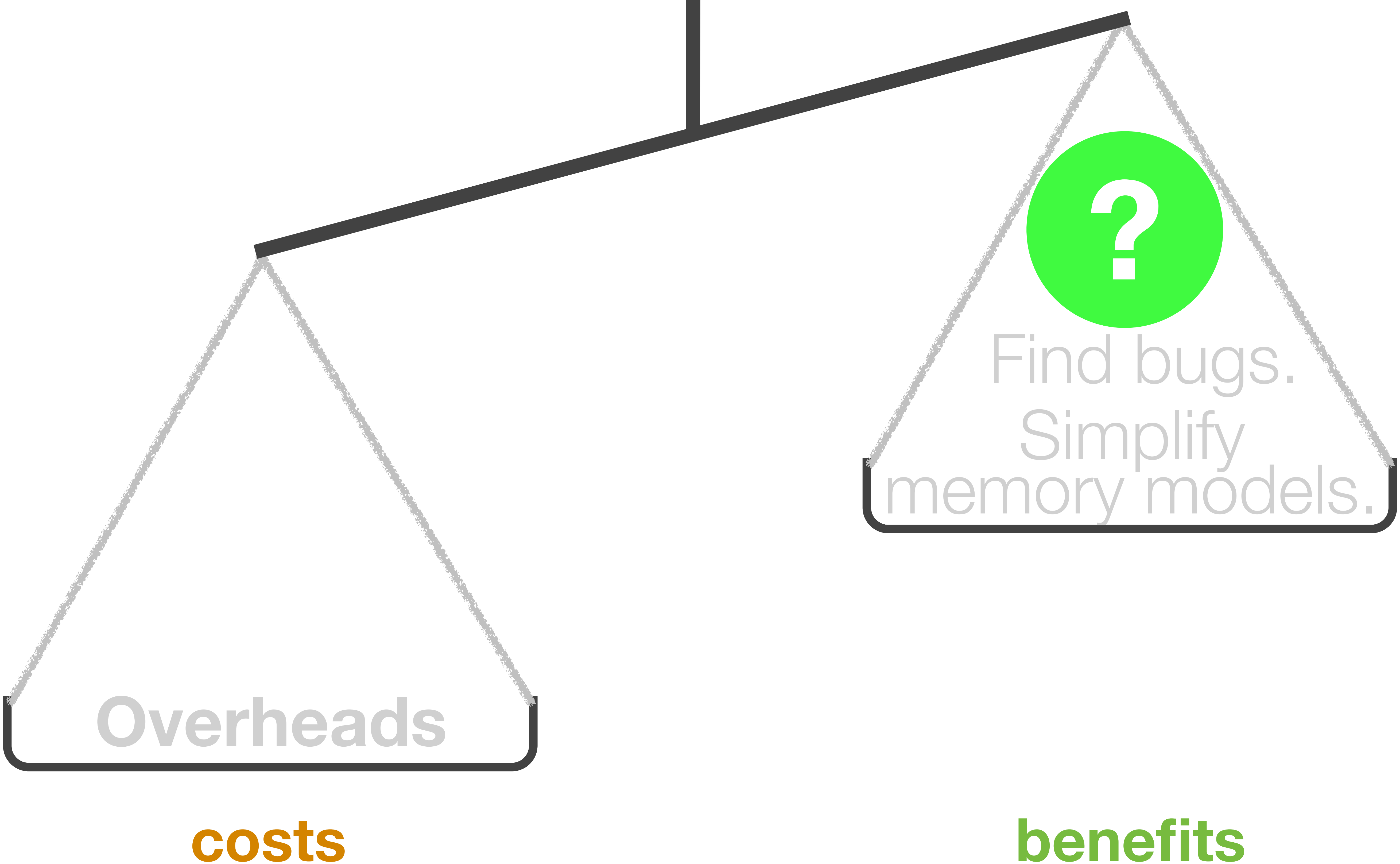
Performance overheads

ongoing research

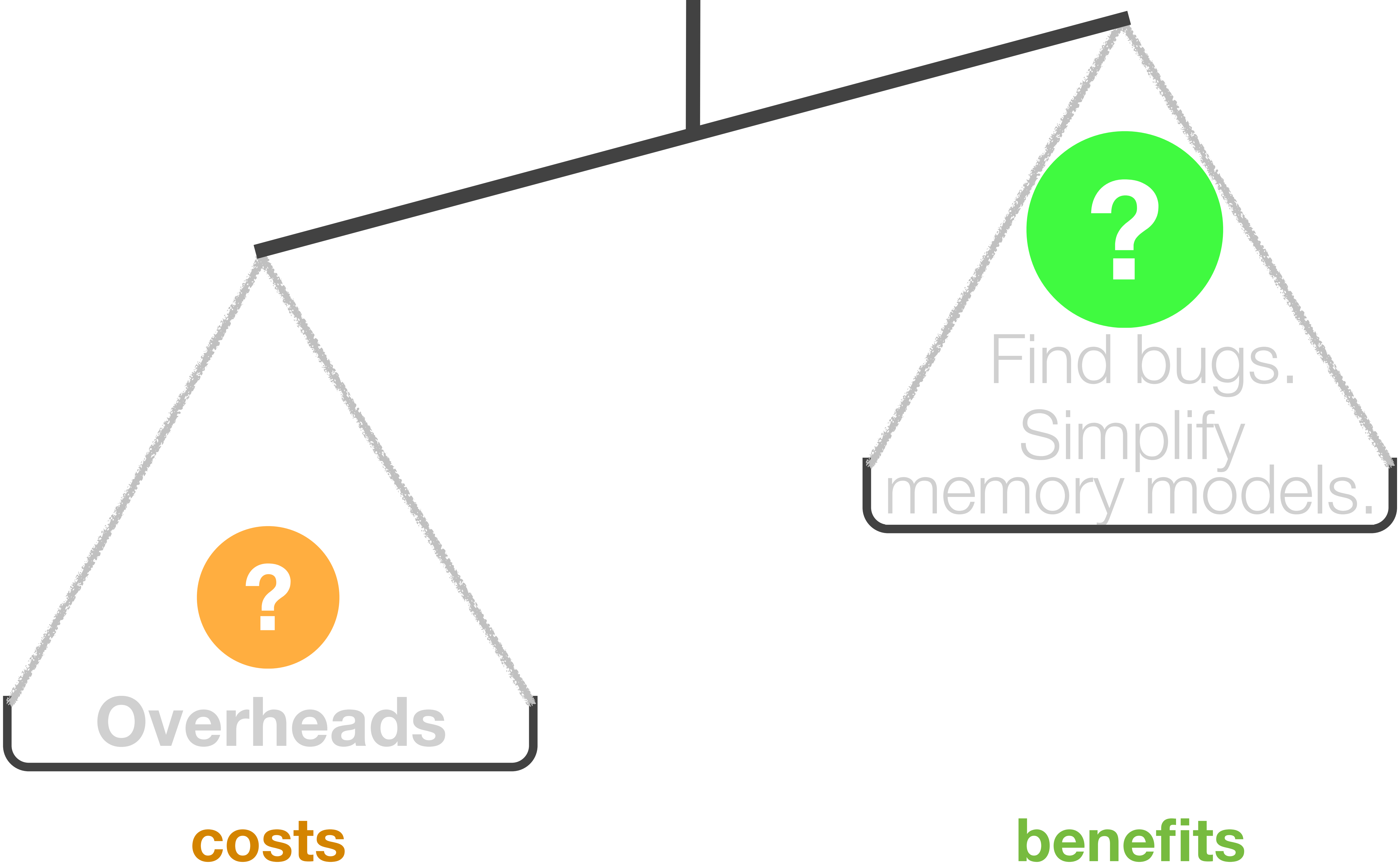

Find bugs.
Simplify memory models.
Overheads
costs
benefits


Find bugs.
Simplify
memory models.
Overheads
costs
benefits

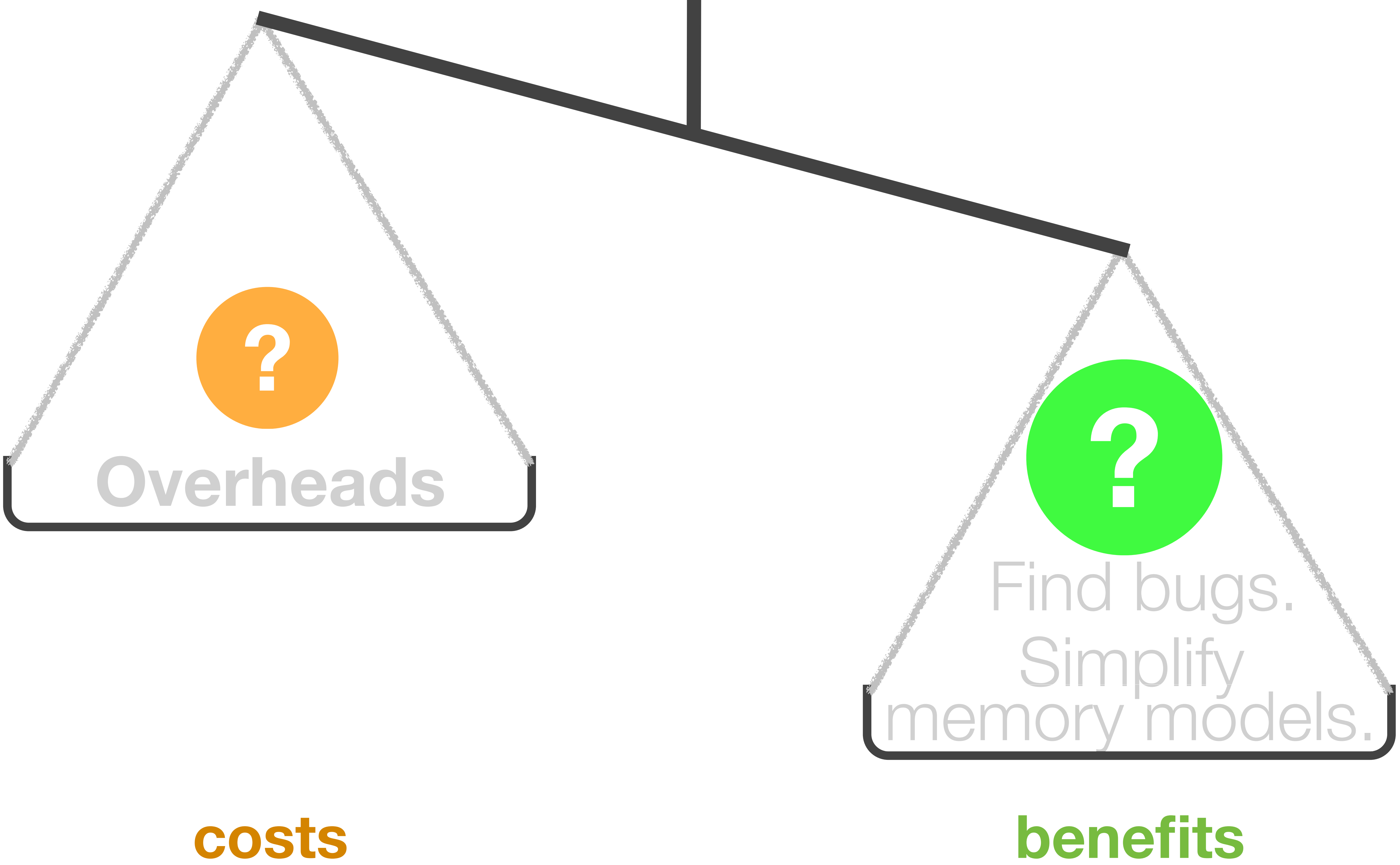

Overheads
Find bugs.
Simplify
memory models.
costs
benefits



This talk explores hidden benefits
of data-race exceptions **in runtime systems**.



This talk explores hidden benefits (and costs) of data-race exceptions **in runtime systems**.



This talk explores hidden benefits (and costs) of data-race exceptions **in runtime systems**.

All races are inherently wrong.

unannotated

All races are inherently wrong.

~~All unannotated races are [illegible] wrong.~~

Exceptions ensure **all unannotated races are impossible**.

Attempts at unannotated racy accesses
are **exceptional, but legal**.

contributions

Properties of data-race exceptions enable **conflict detection**.

Exploit data-race exceptions in **concurrent garbage collection**.

Low-level data-race exceptions have **subtle implications**.

outline

Review data races, exceptions, and sequential consistency.

Properties of data-race exceptions enable **conflict detection**.

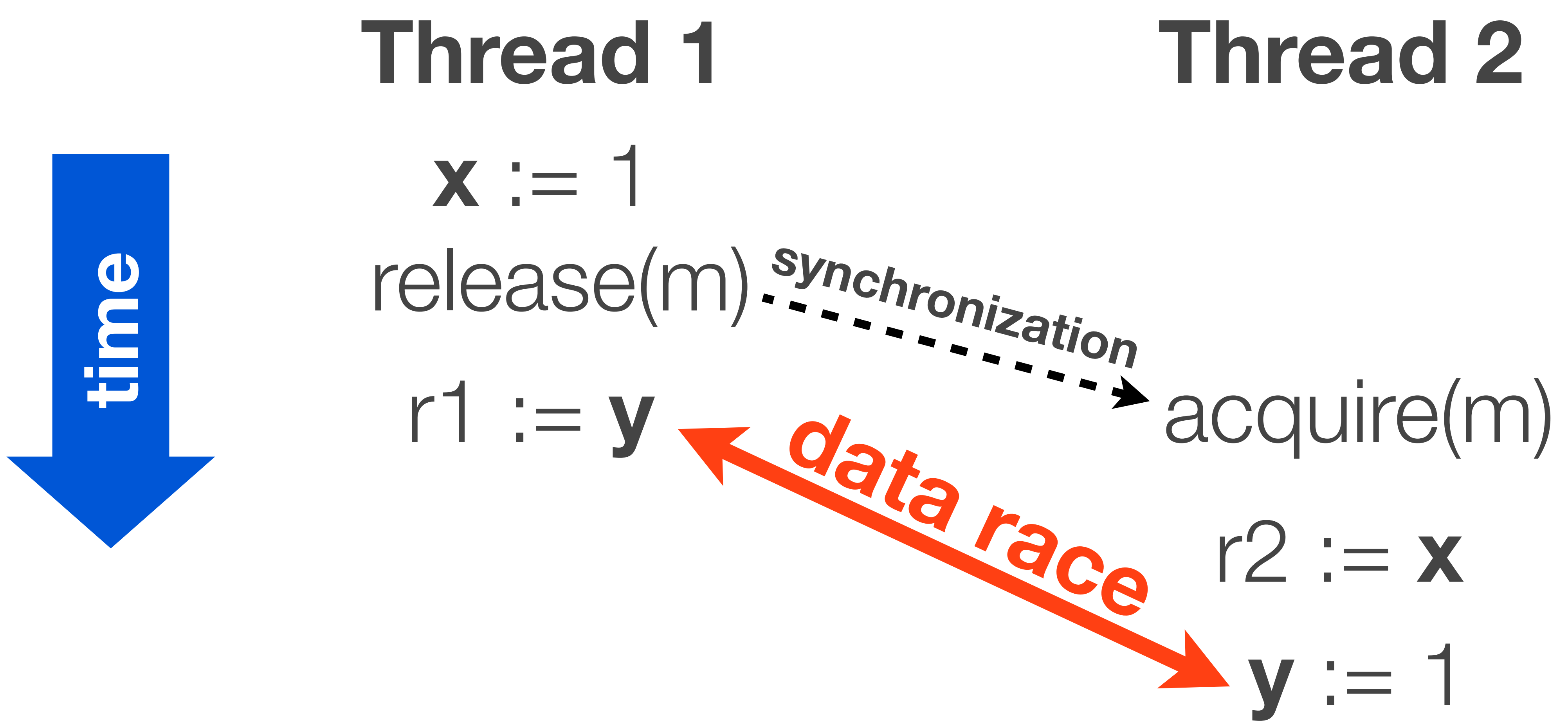
Exploit data-race exceptions in **concurrent garbage collection**.

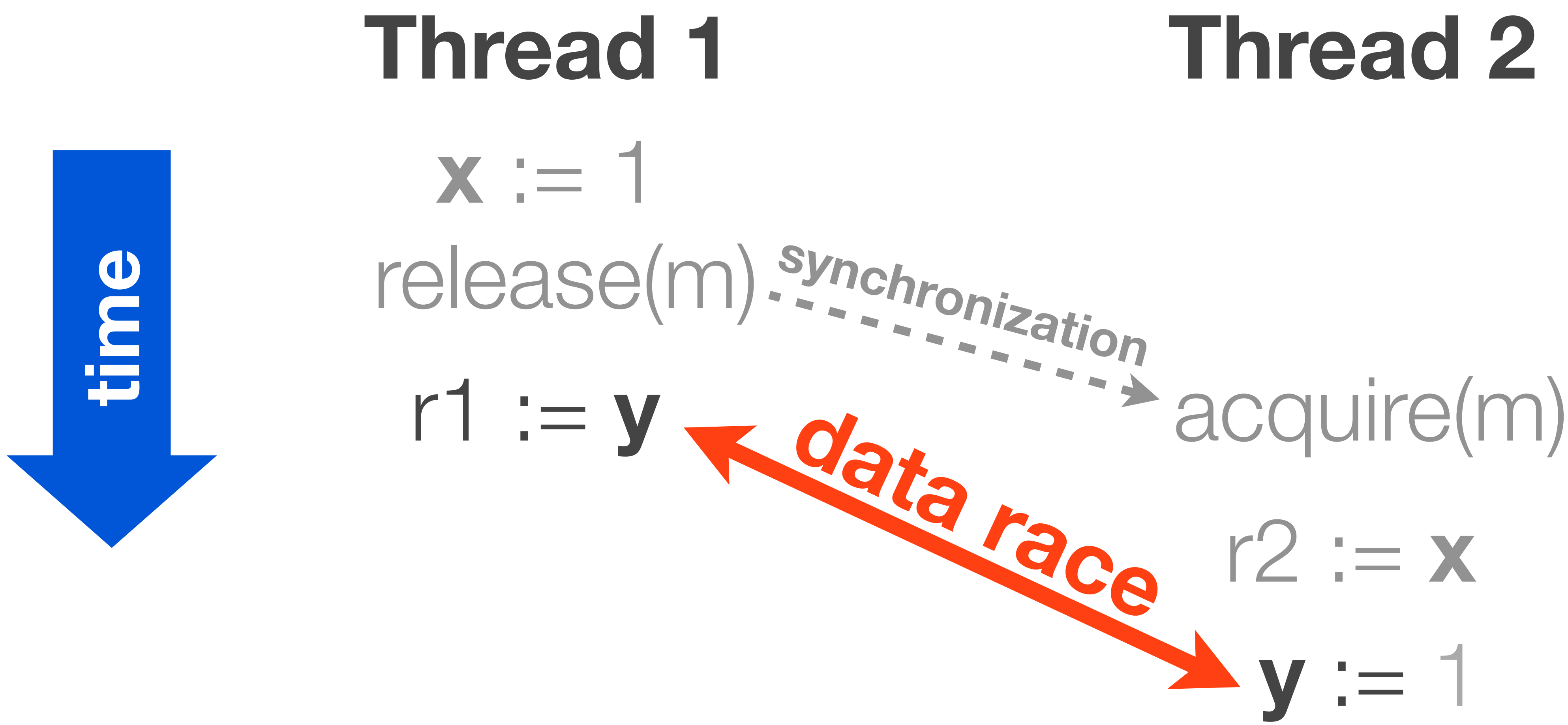
Low-level data-race exceptions have **subtle implications**.

Conclusions

data race

a pair of ***concurrent***, ***conflicting*** accesses

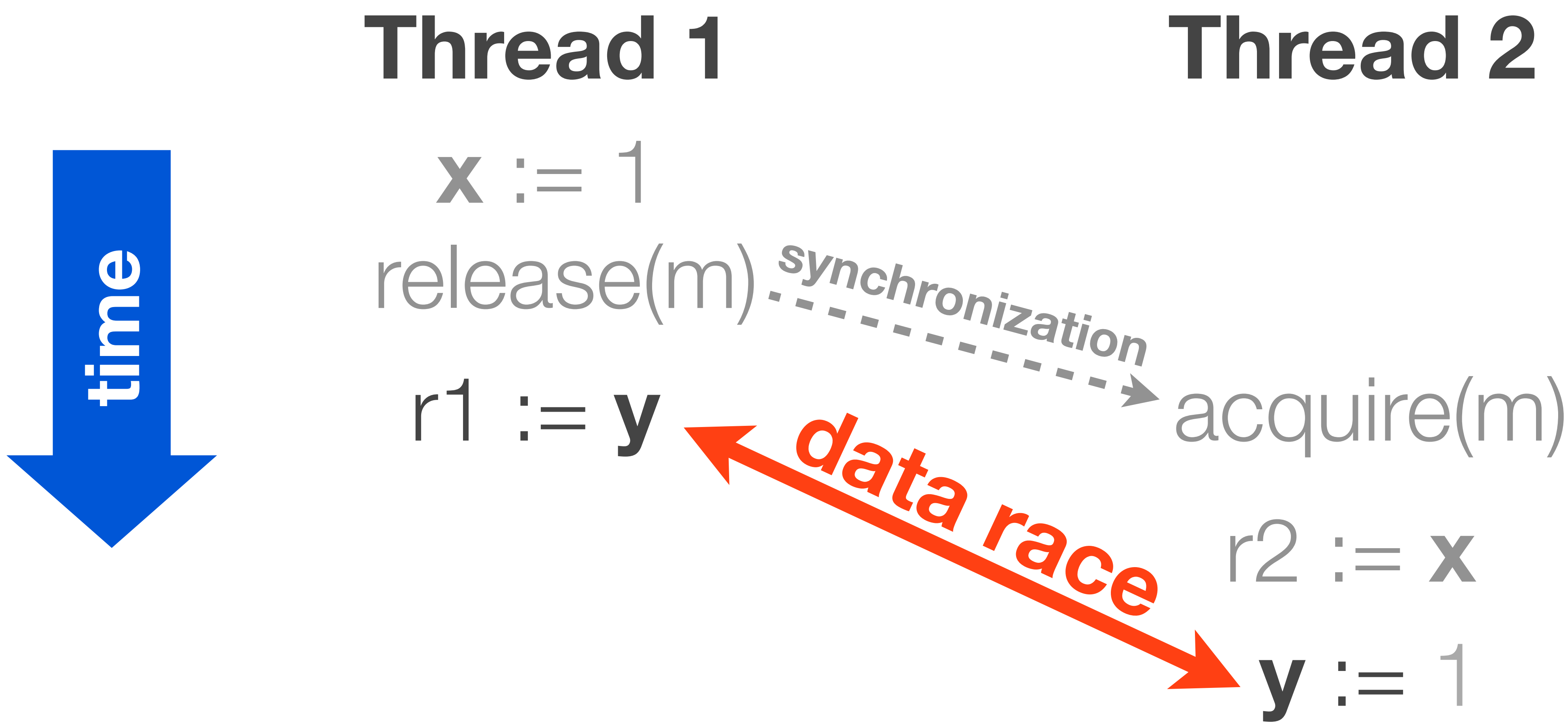




exception on second access

data-race exceptions

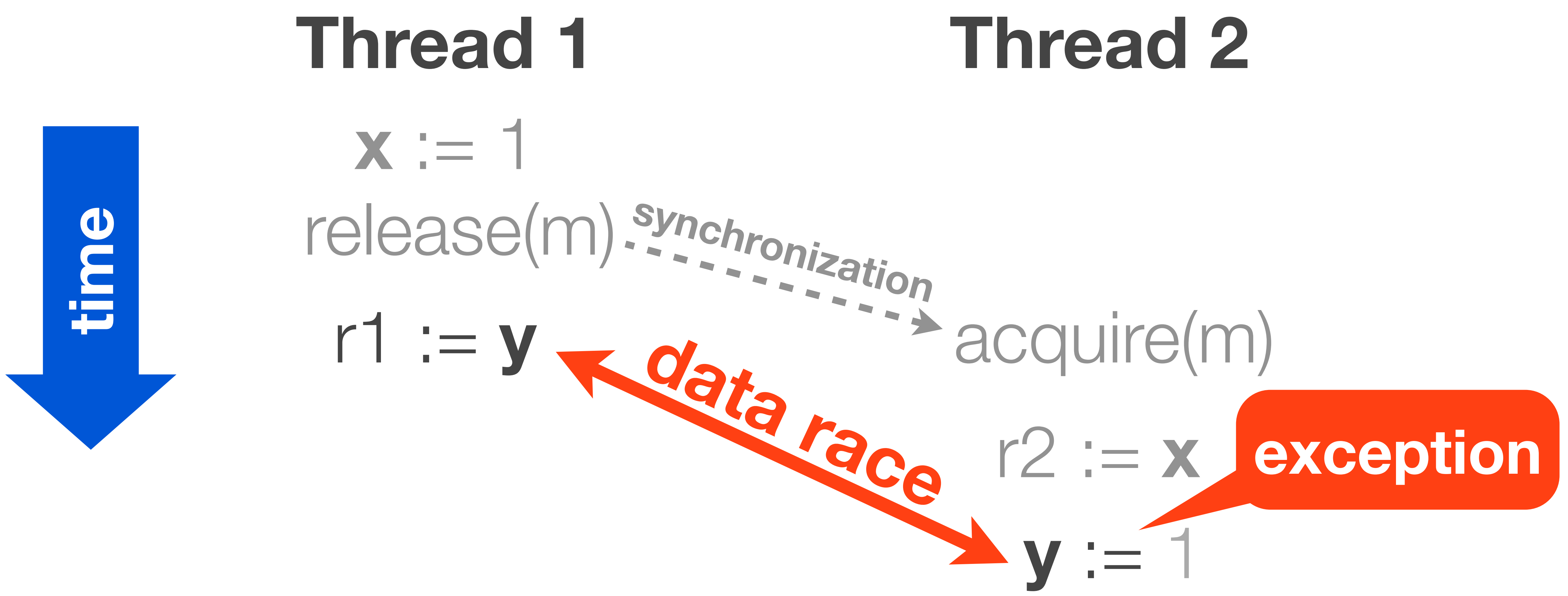
guarantee either data-race-free or exception



exception on second access

data-race exceptions

guarantee either data-race-free or exception



exception on second access

Data-race exceptions

DRF $\oplus$ exception

and

Java/C++

DRF $\Rightarrow$ SC

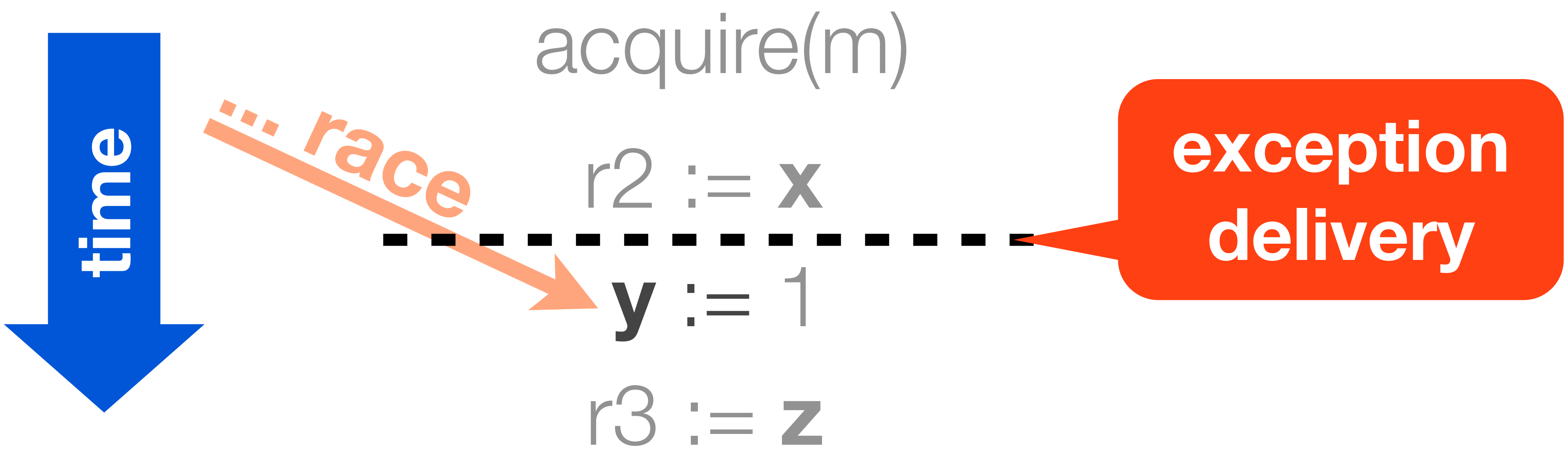
$\Downarrow$

SC or exception

data-race exceptions in HW

precise

Suspend the thread just before its racy access.
Respect program order.



handleable

Deliver a trap with information about the race.

concurrent GC

Concurrent mark-sweep: atomic/consistent heap traversal

tri-color marking and *write barriers*

Concurrent copying/moving: atomic object copying

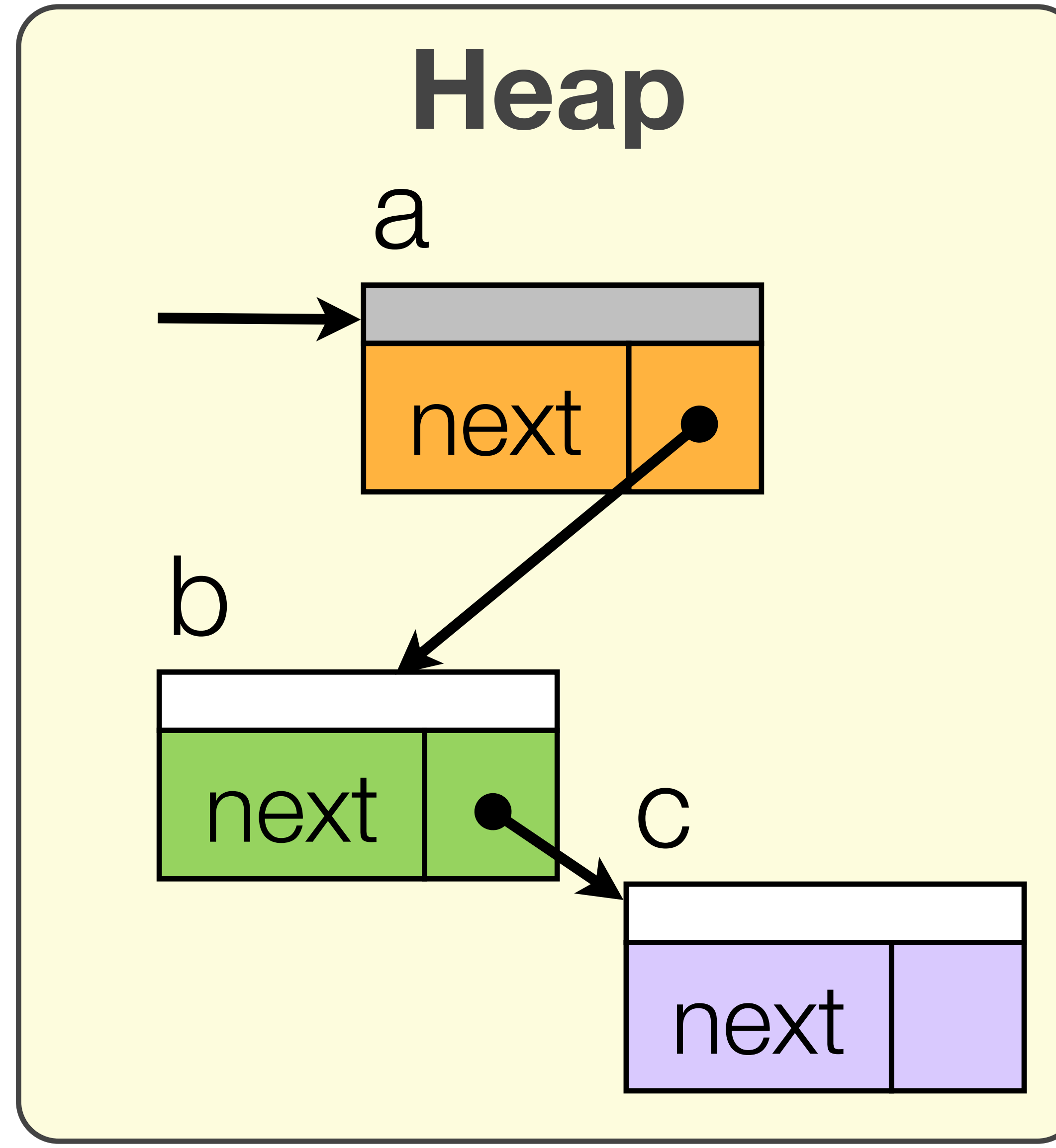
Piggyback on STM runtime [McGachey et al., PPoPP 2008]

Lock-free algorithms [Pizlo et al., ISMM 2007, PLDI 2008]

mark-sweep

GC thread **Mutator**

gray(a)



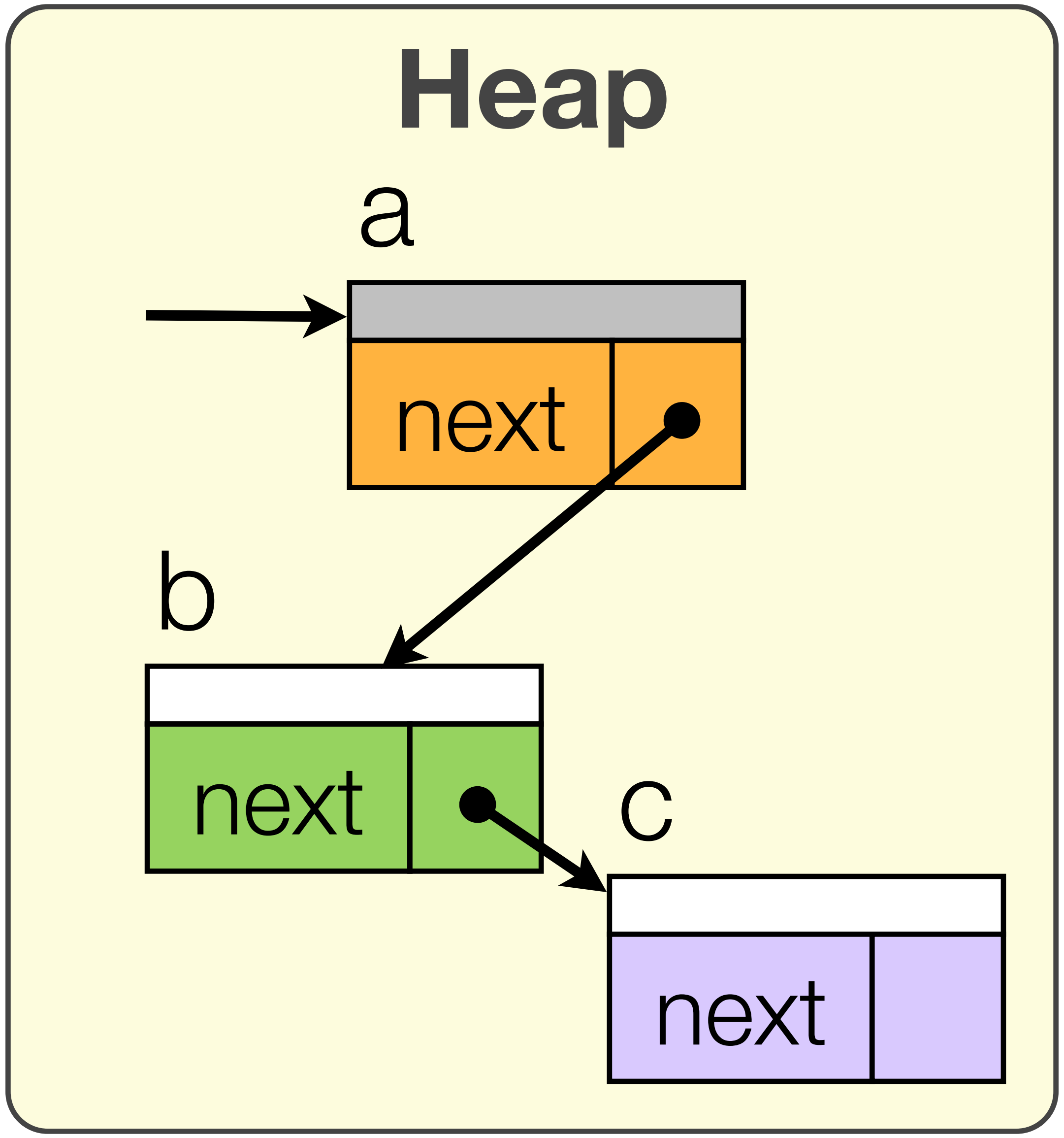
■ reachable, refs visited
□ reachable, refs unvisited
□ unreachable/unknown

mark-sweep

GC thread **Mutator**

gray(**a**)

gray(**a**.next)



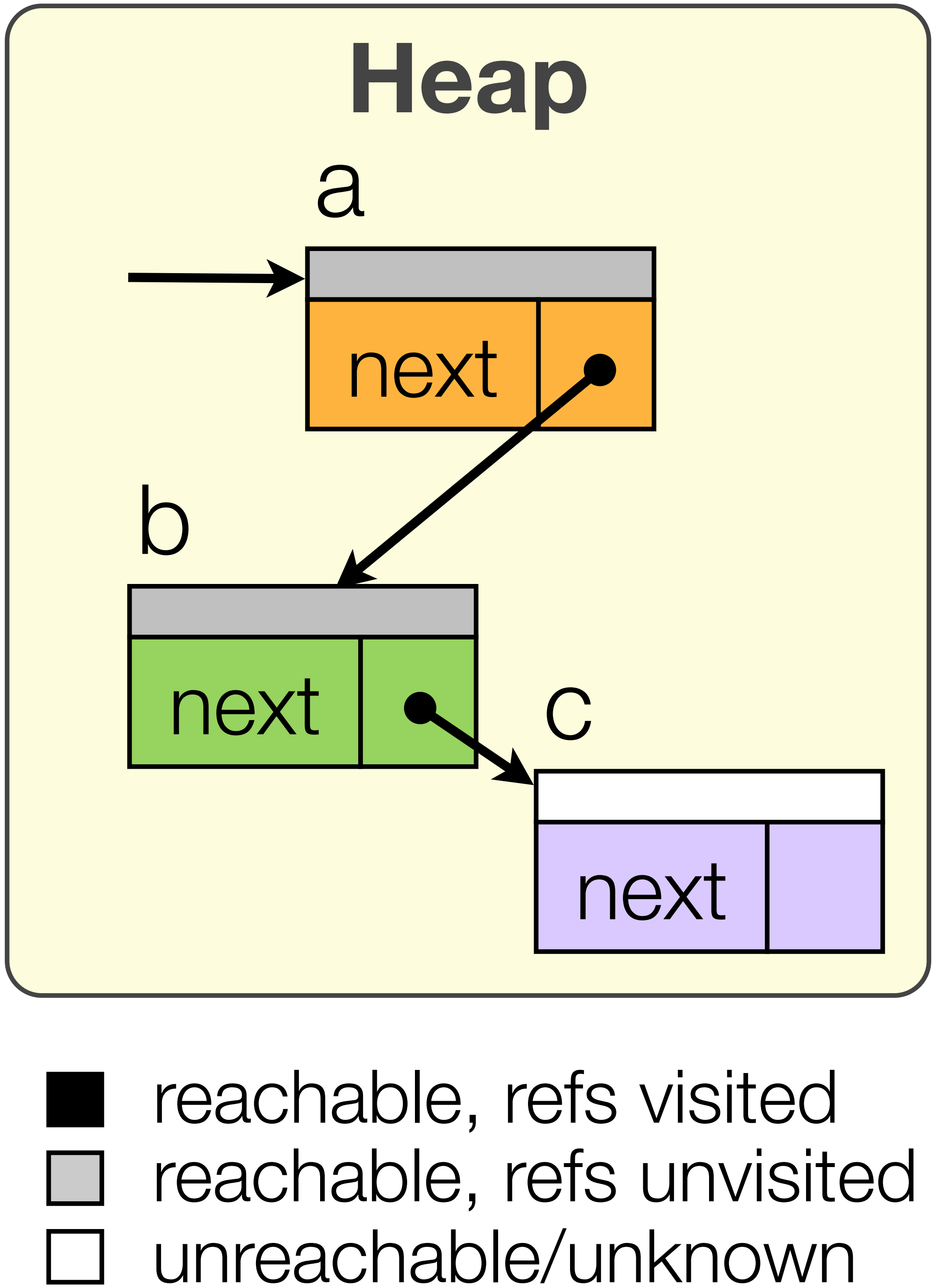
■ reachable, refs visited
▨ reachable, refs unvisited
□ unreachable/unknown

mark-sweep

GC thread **Mutator**

gray(**a**)

gray(**a**.next)



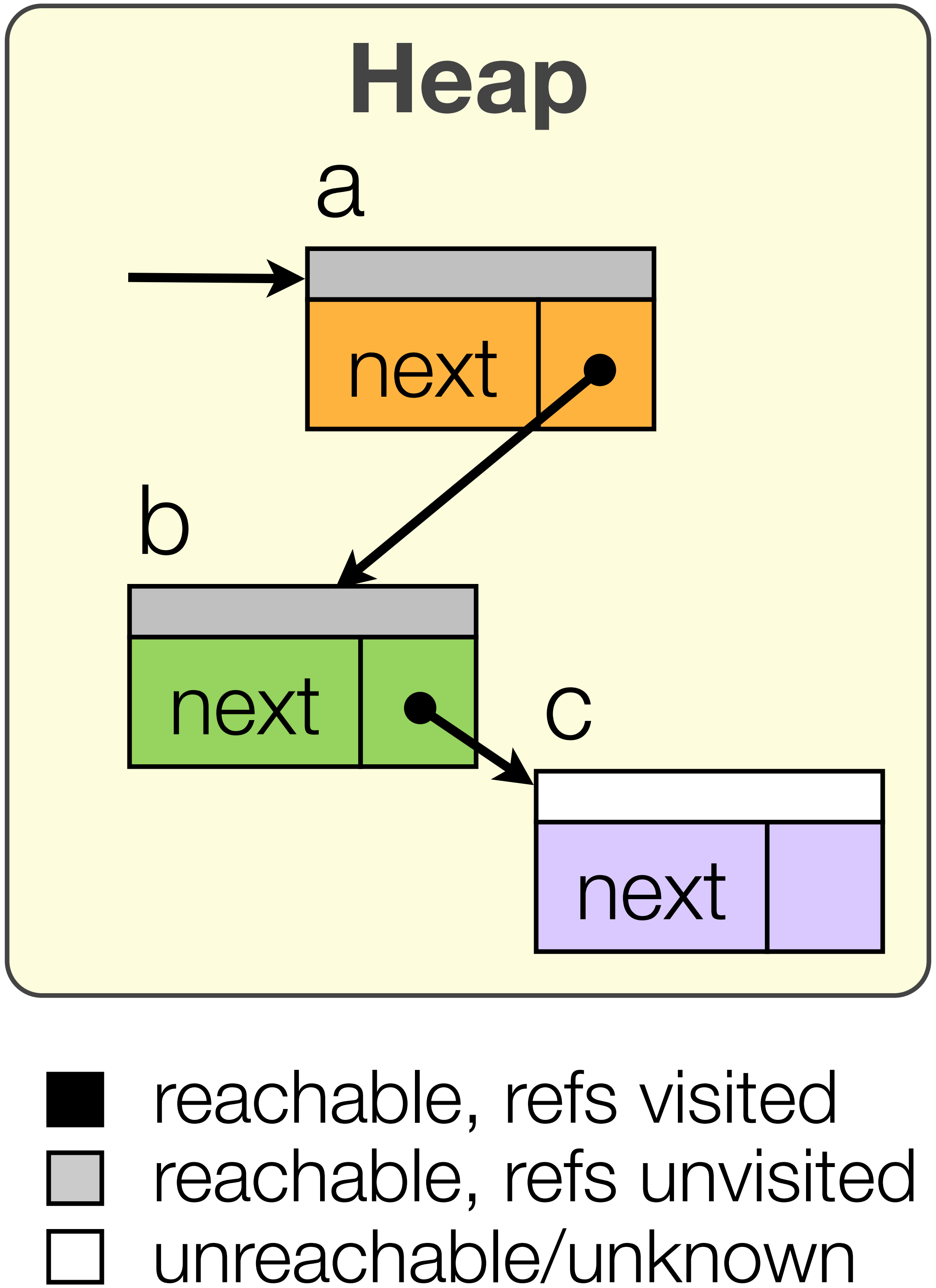
mark-sweep

GC thread **Mutator**

gray(**a**)

gray(**a**.next)

black(**a**)



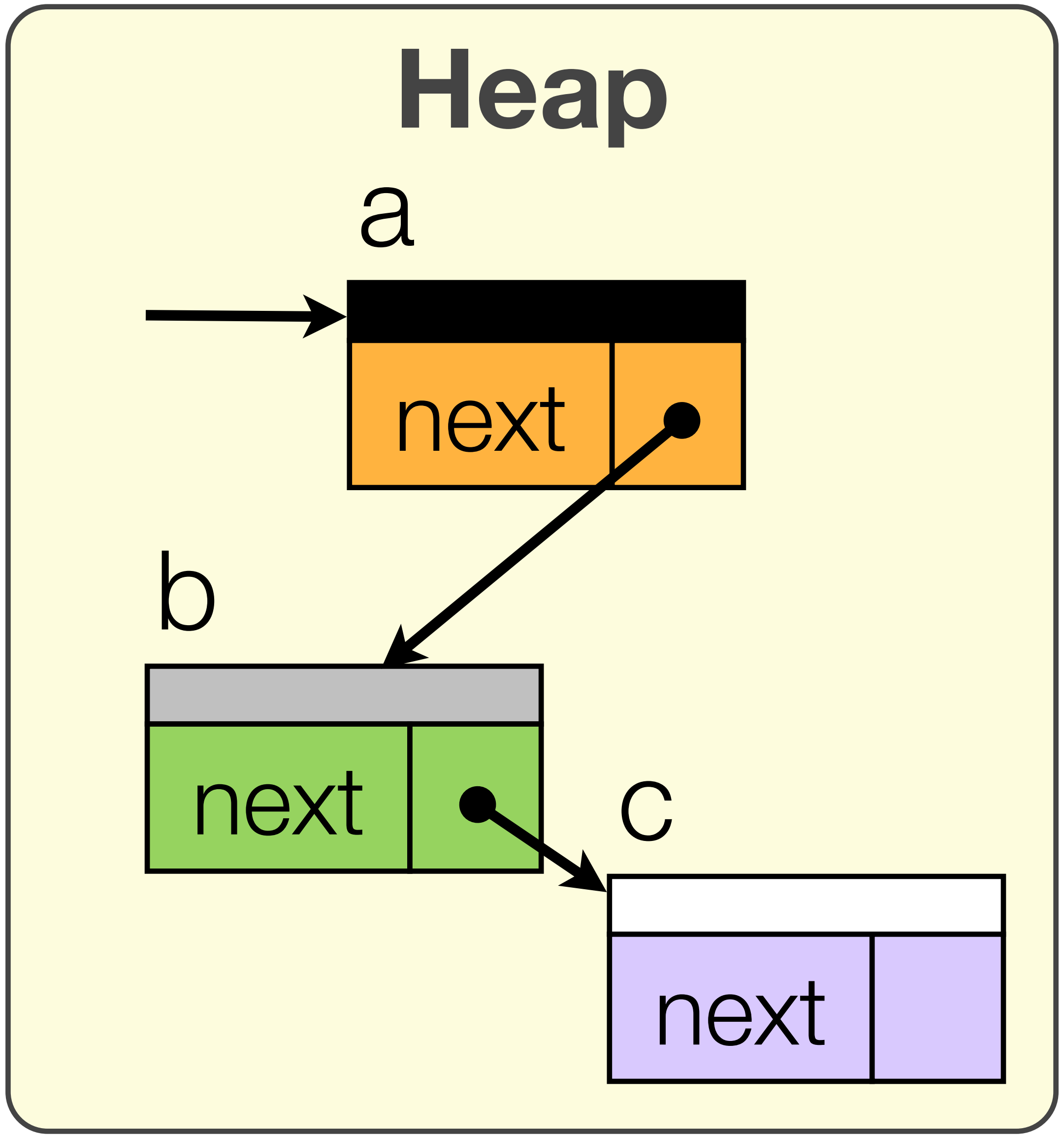
mark-sweep

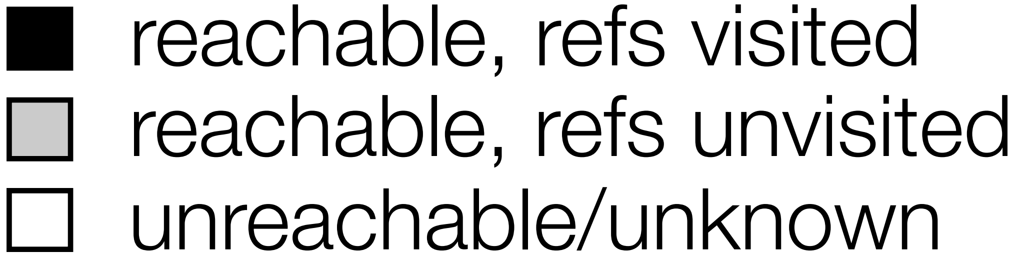
GC thread **Mutator**

gray(**a**)

gray(**a**.next)

black(**a**)





mark-sweep

GC thread **Mutator**

gray(a)

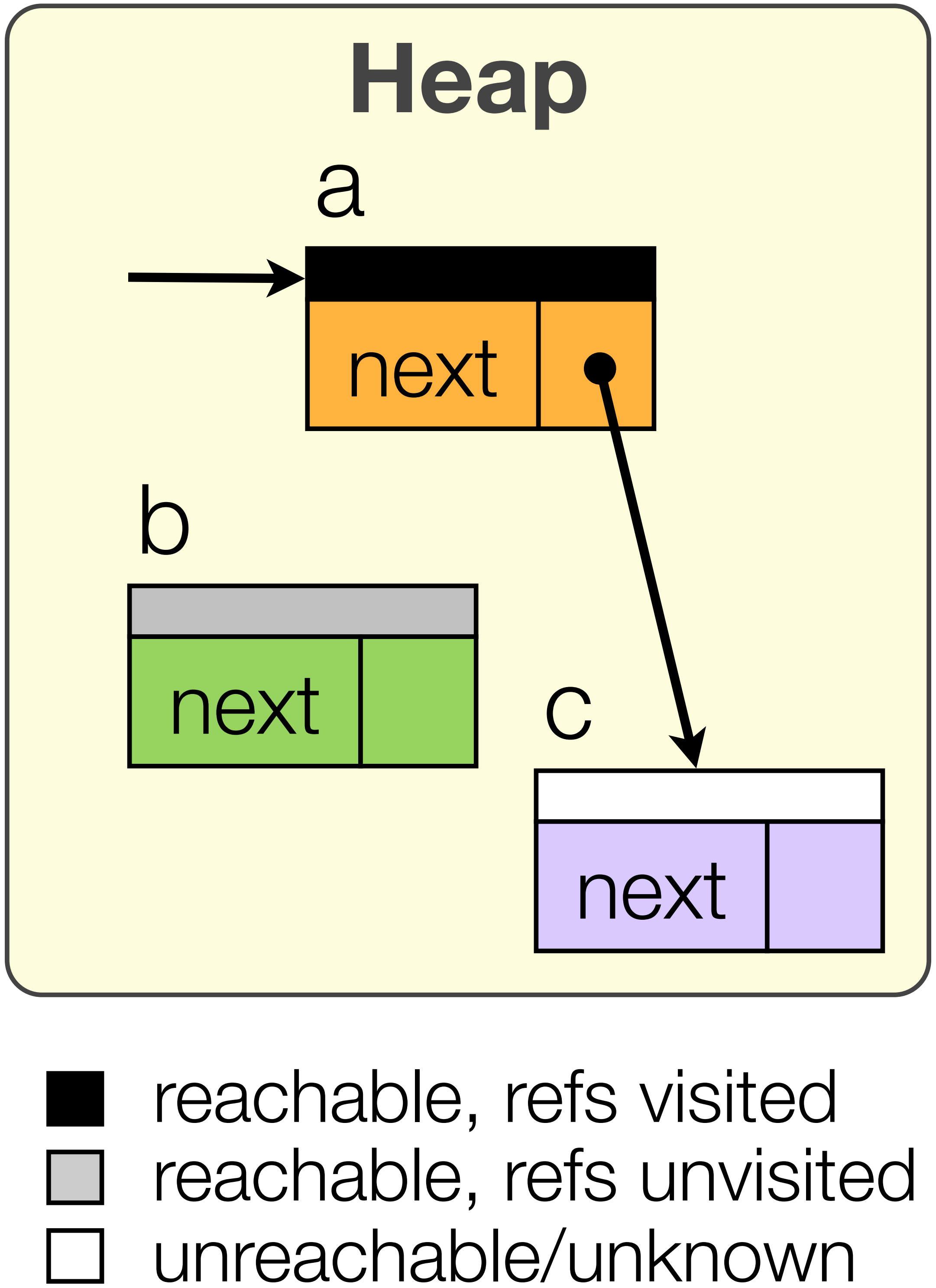
gray(a.next)

black(a)

n := a.next

a.next := a.next.next

n.next := null



mark-sweep

GC thread **Mutator**

gray(**a**)

gray(**a**.next)

black(**a**)

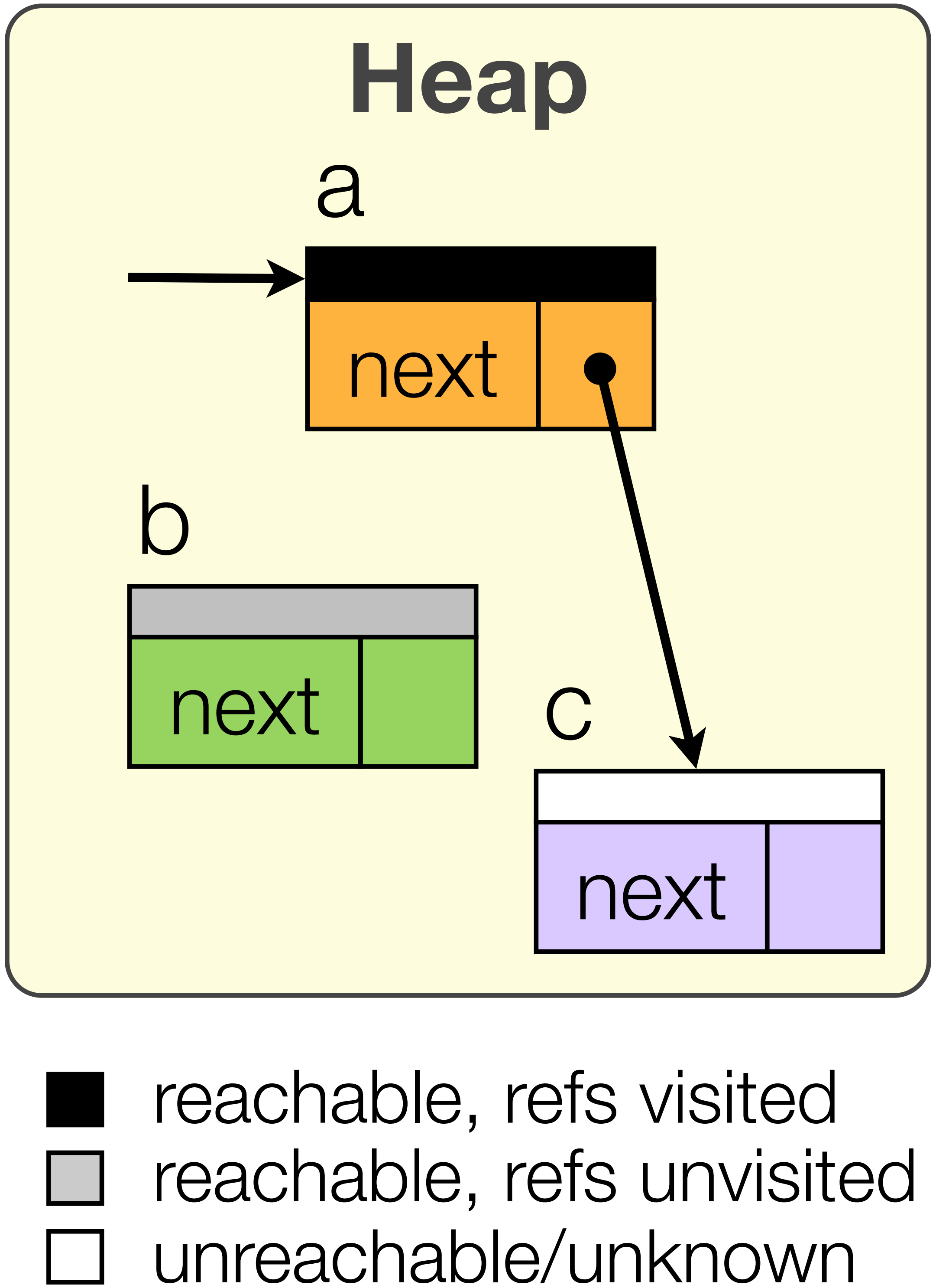
n := **a**.next

a.next := **a**.next.next

n.next := null

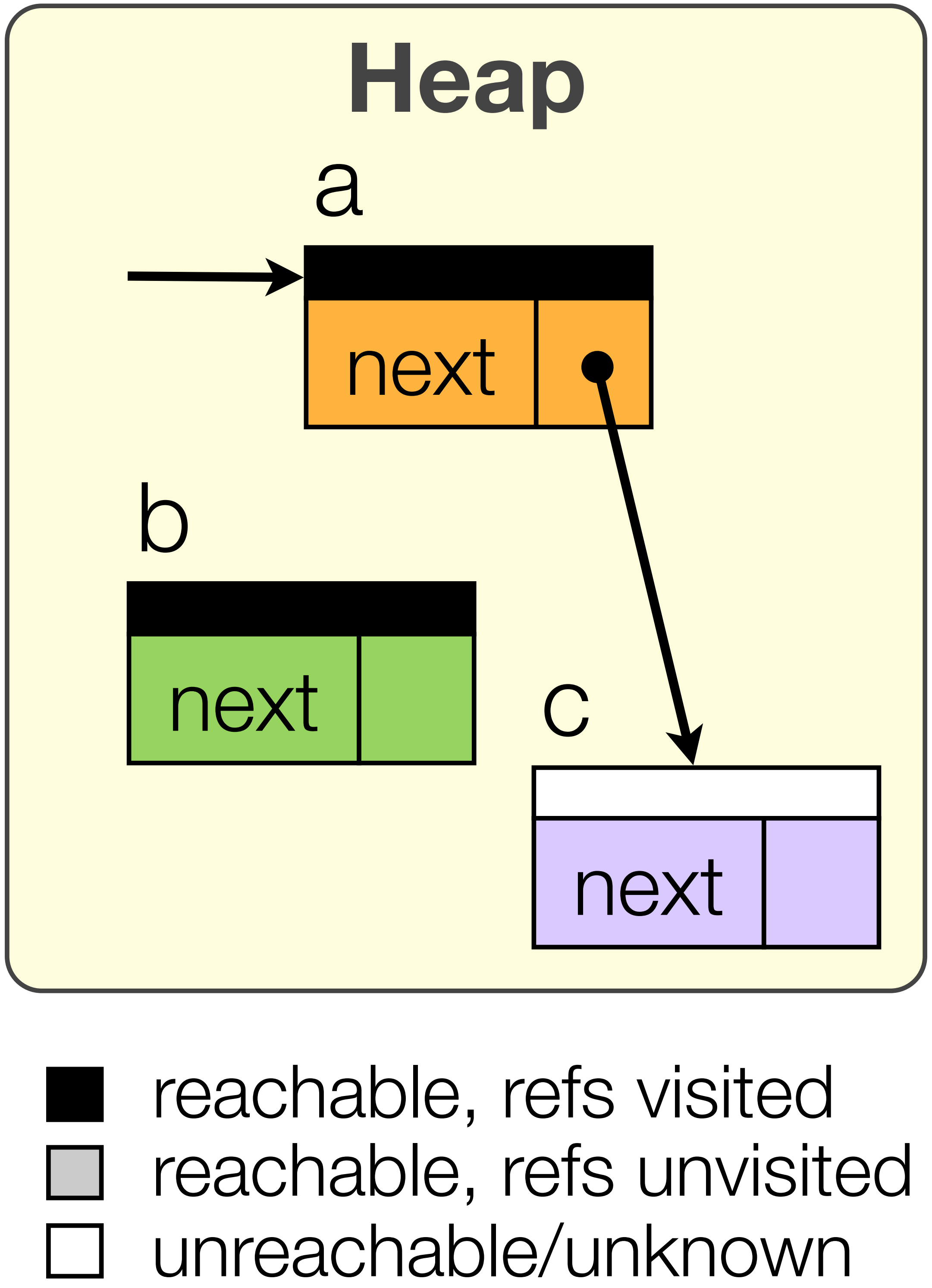
gray(**b**.next)

black(**b**)



mark-sweep

GC thread	Mutator
gray(a)	
gray(a.next)	
black(a)	
	n := a.next
	a.next := a.next.next
	n.next := null
gray(b.next)	
black(b)	



mark-sweep

GC thread | **Mutator**

gray(**a**)

gray(**a**.next)

black(**a**)

n := a.next

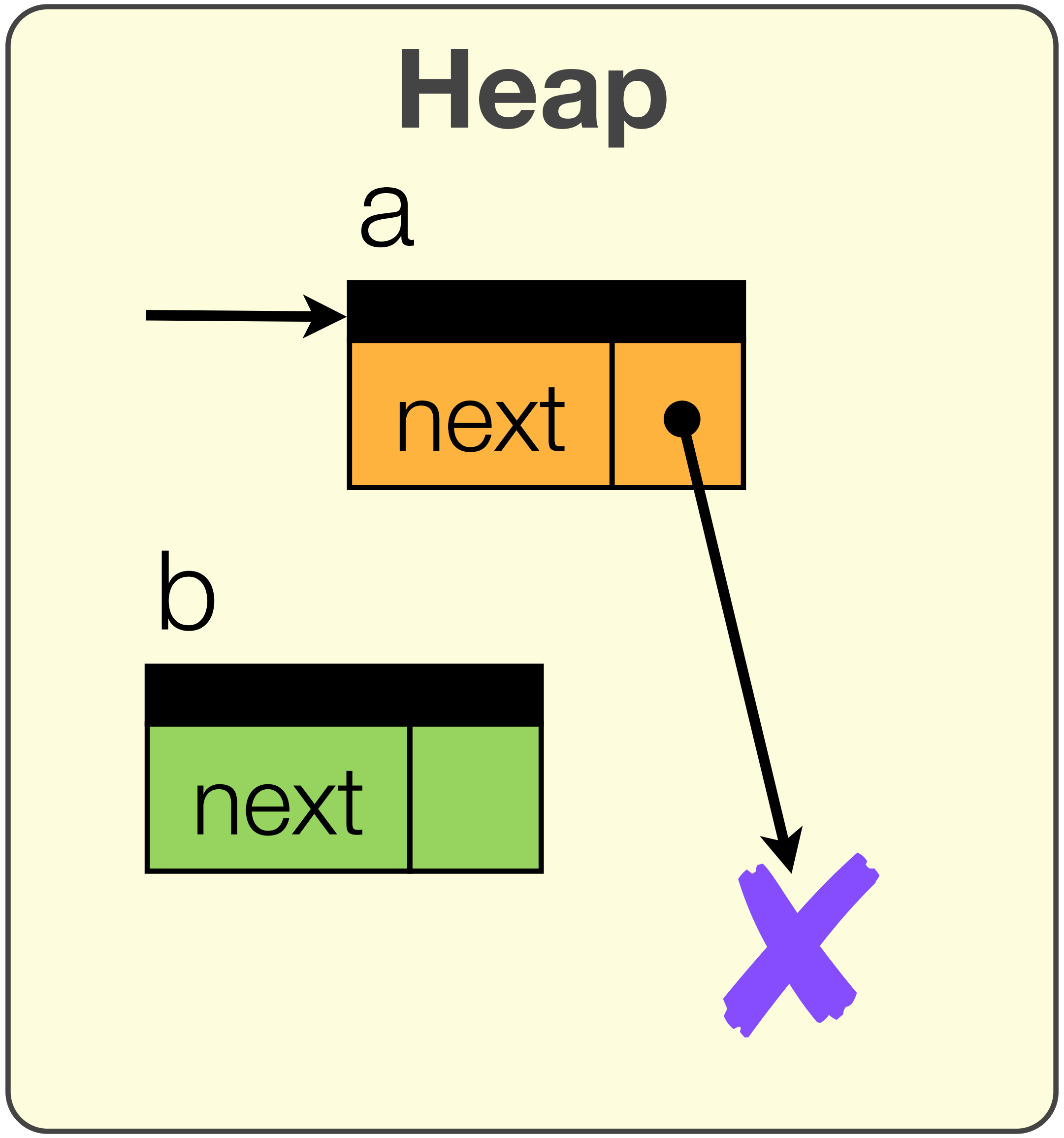
a.next := a.next.next

n.next := null

gray(**b**.next)

black(**b**)

collect(**c**)

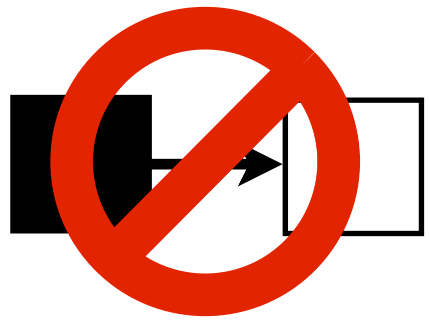


■ reachable, refs visited

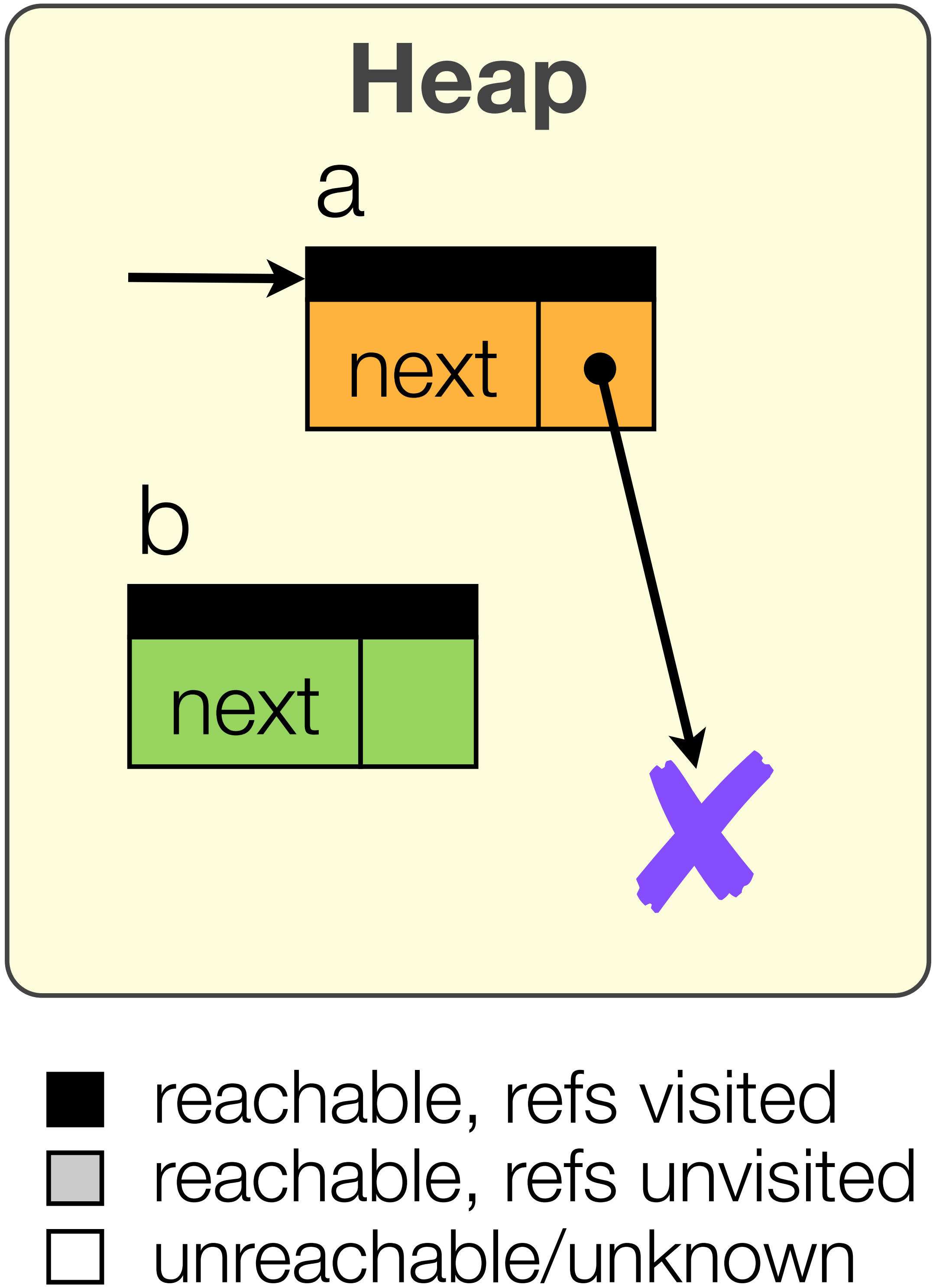
▢ reachable, refs unvisited

□ unreachable/unknown

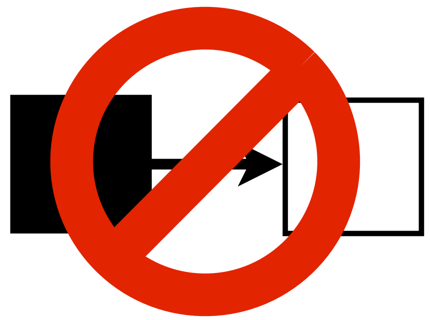
mark-sweep



GC thread	Mutator
gray(a)	
gray(a.next)	
black(a)	
	n := a.next
	a.next := a.next.next
	n.next := null
gray(b.next)	
black(b)	
collect(c)	



mark-sweep



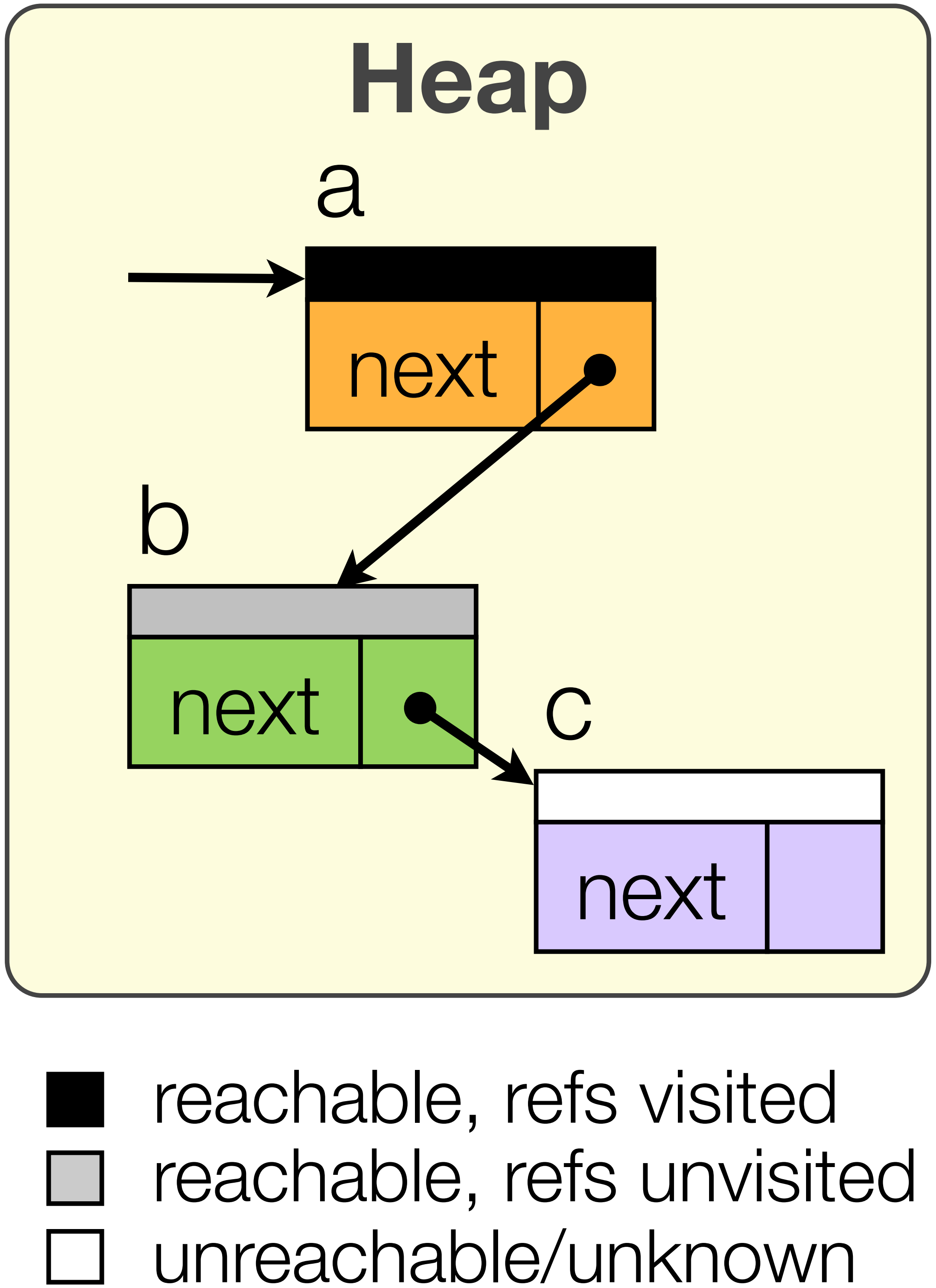
GC thread | **Mutator**

gray(**a**)

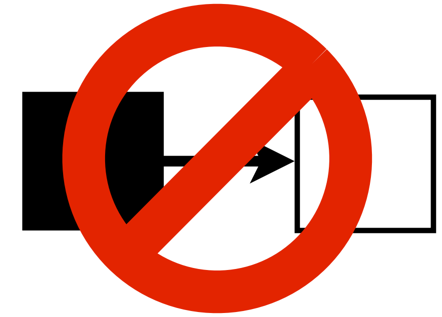
gray(**a**.next)

black(**a**)

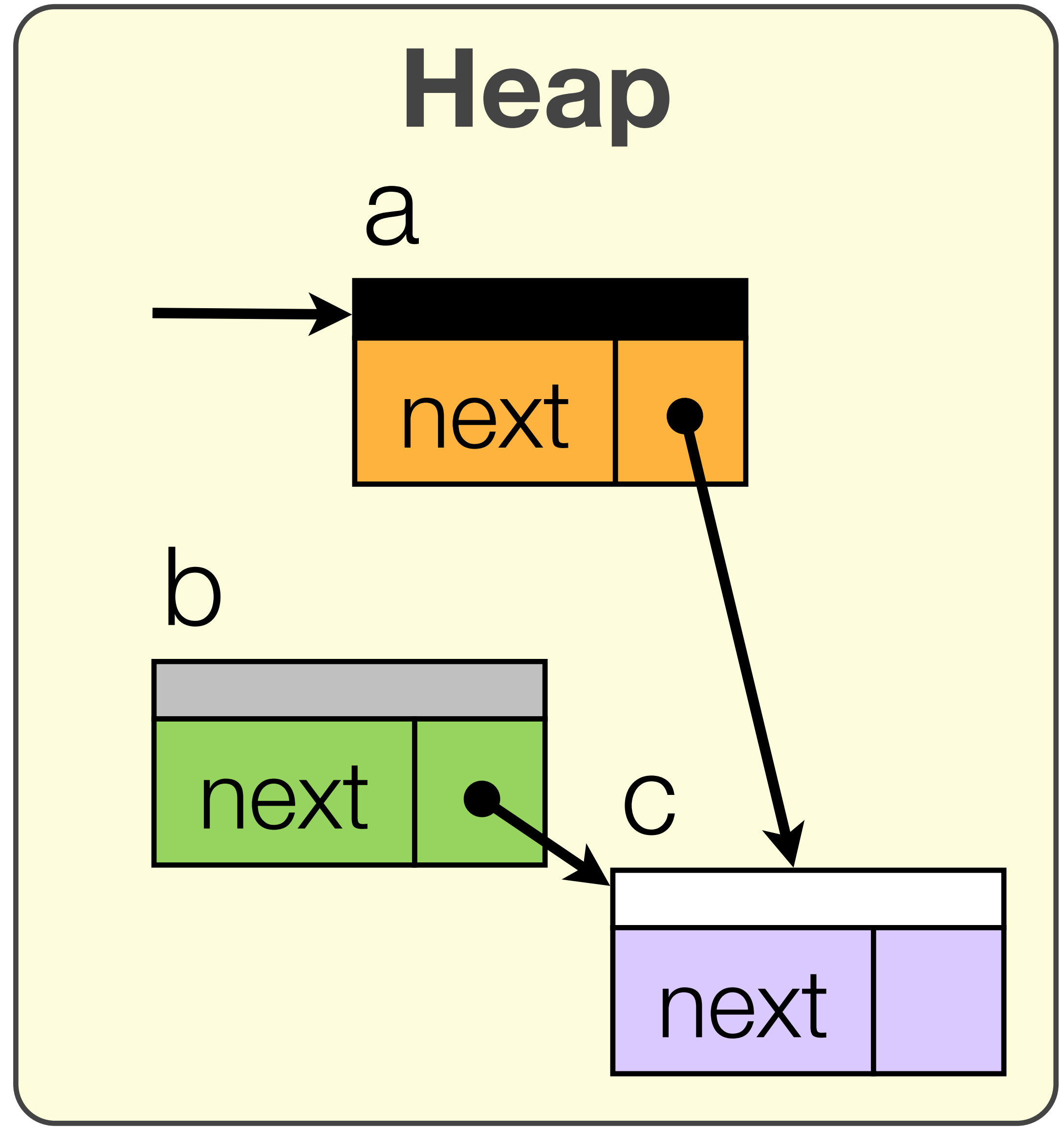
n := **a**.next



mark-sweep

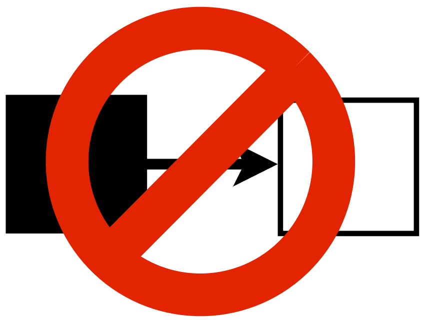


GC thread	Mutator
gray(a)	
gray(a.next)	
black(a)	
	n := a.next
	a.next := a.next.next
	gray(c)

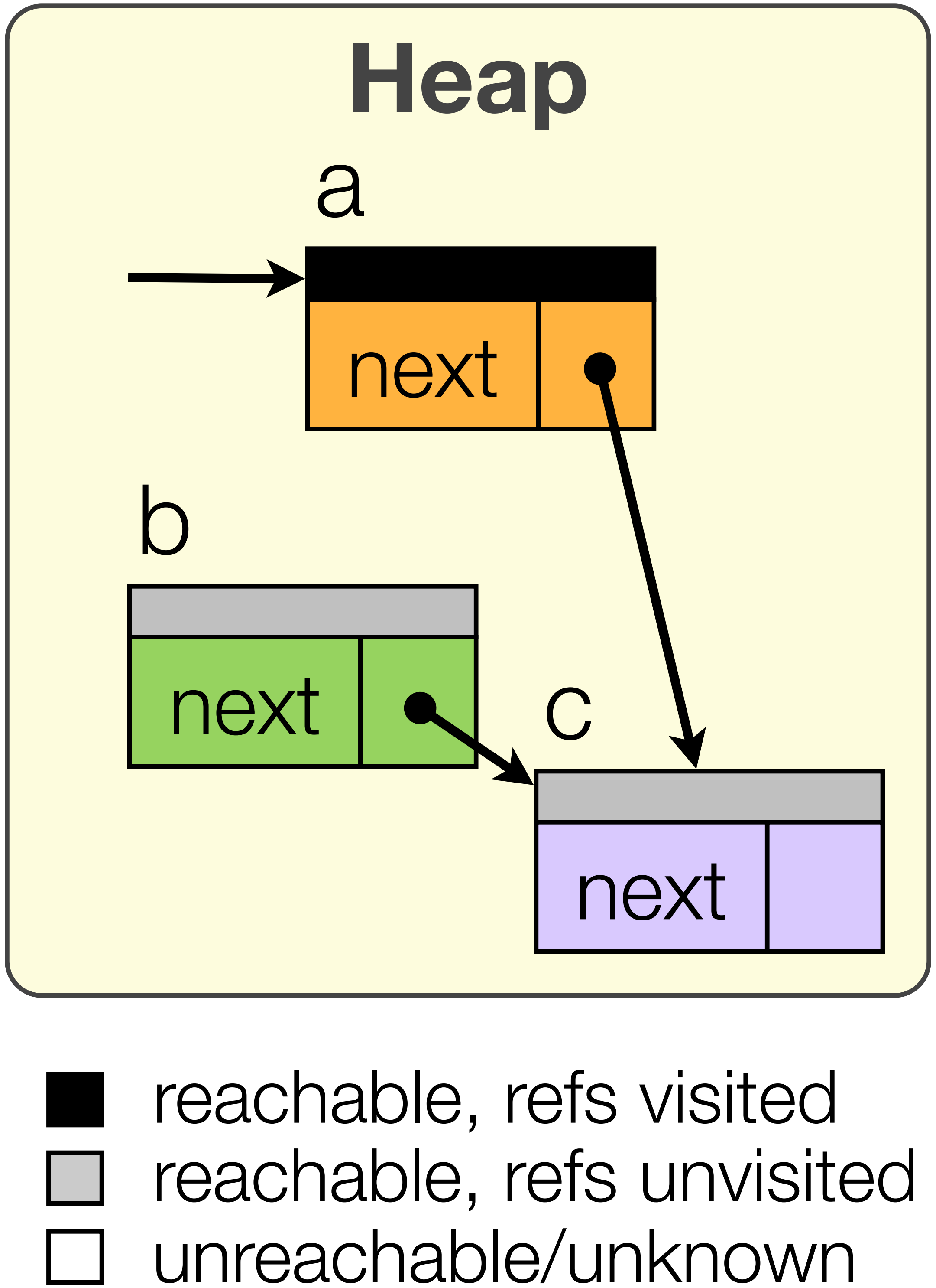


■ reachable, refs visited
■ reachable, refs unvisited
□ unreachable/unknown

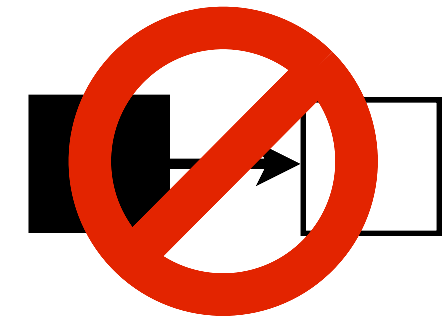
mark-sweep



GC thread	**Mutator**
gray(a)
gray(a.next)
black(a)
n := a.next
a.next := a.next.next
gray(c)



mark-sweep



GC thread **Mutator**

gray(a)

gray(a.next)

black(a)

n := a.next

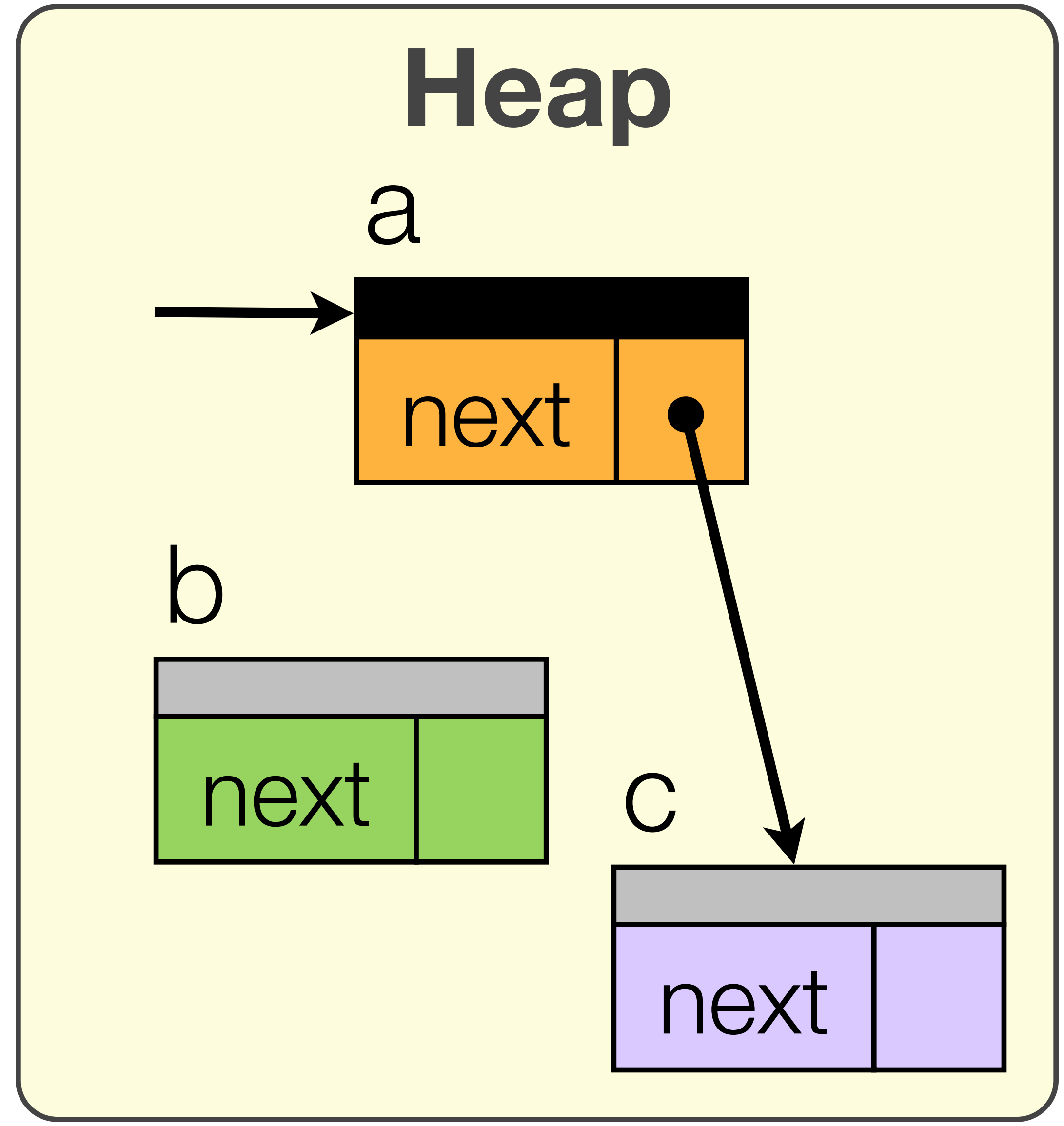
a.next := a.next.next

gray(c)

n.next := null

gray(b.next)

black(b)

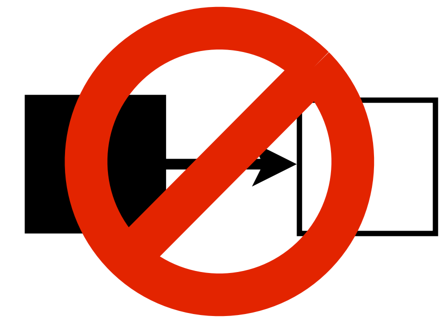


reachable, refs visited

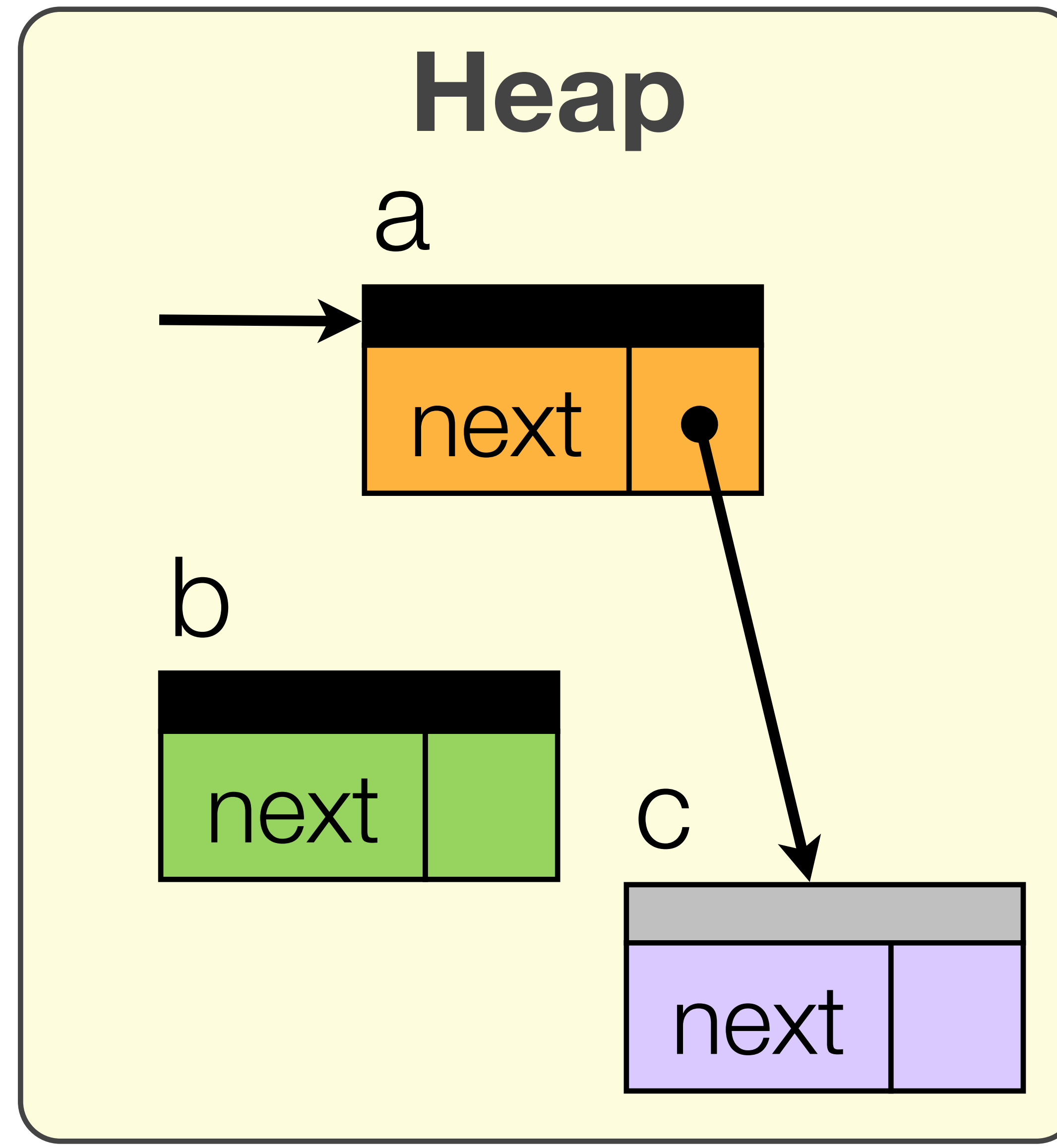
reachable, refs unvisited

unreachable/unknown

mark-sweep

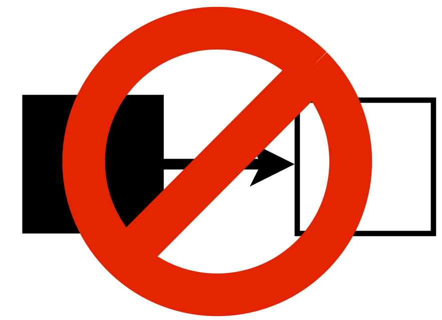


GC thread	**Mutator**
gray(a)	
gray(a.next)	
black(a)	
	n := a.next
	a.next := a.next.next
	gray(c)
	n.next := null
gray(b.next)	
black(b)	

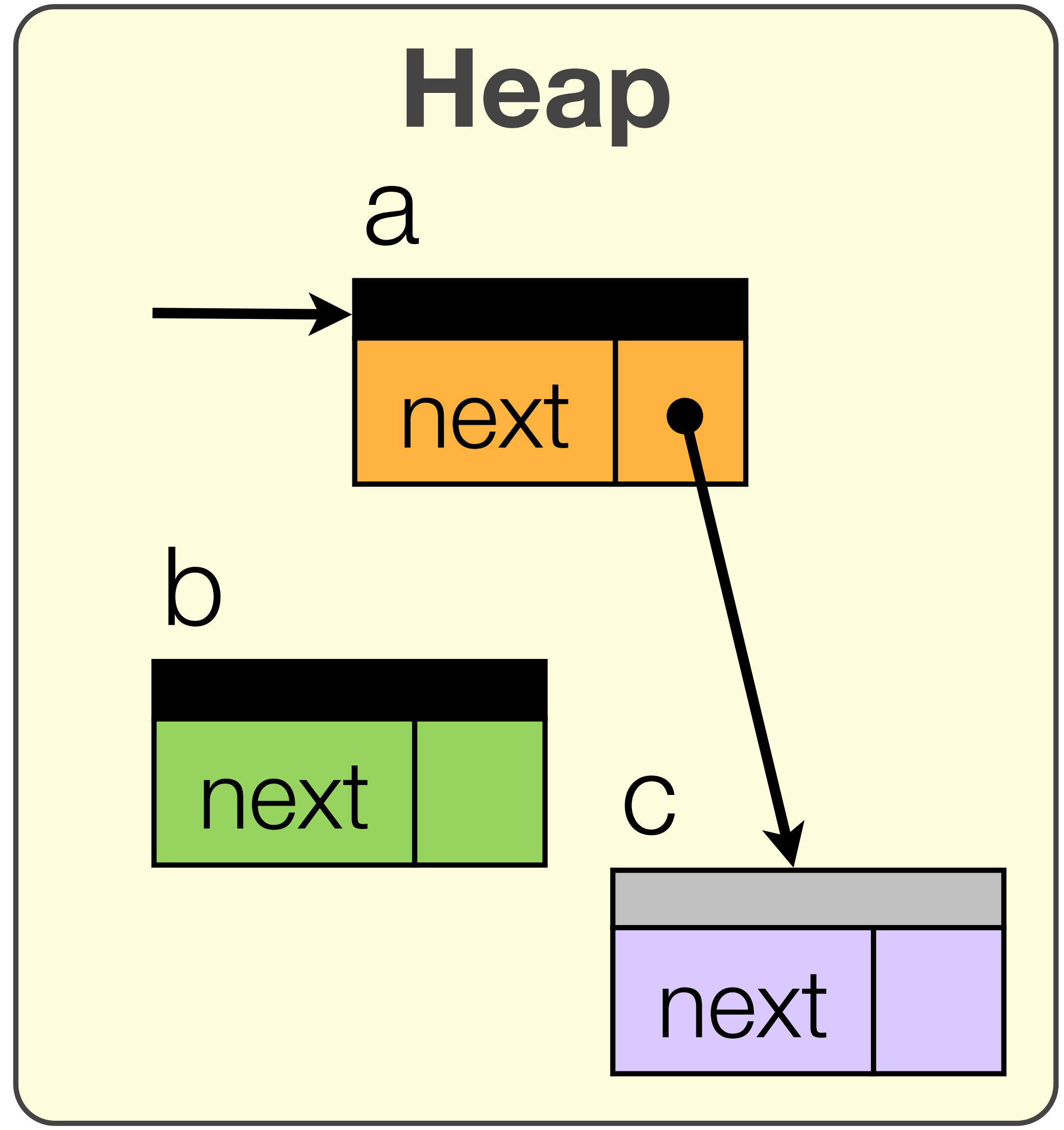


- ■ reachable, refs visited
- ▩ reachable, refs unvisited
- □ unreachable/unknown

mark-sweep

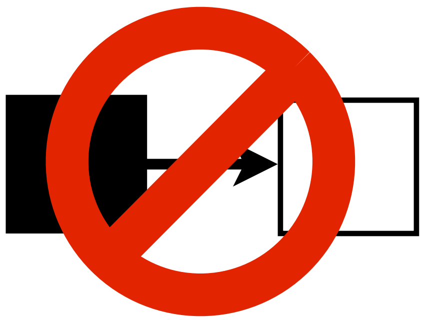


GC thread	Mutator
gray(**a**)	
gray(**a**.next)	
black(**a**)	
	n := **a**.next
	a.next := **a**.next.next
	gray(**c**)
	n.next := null
gray(**b**.next)	
black(**b**)	
gray(**c**.next)	
black(**c**)	

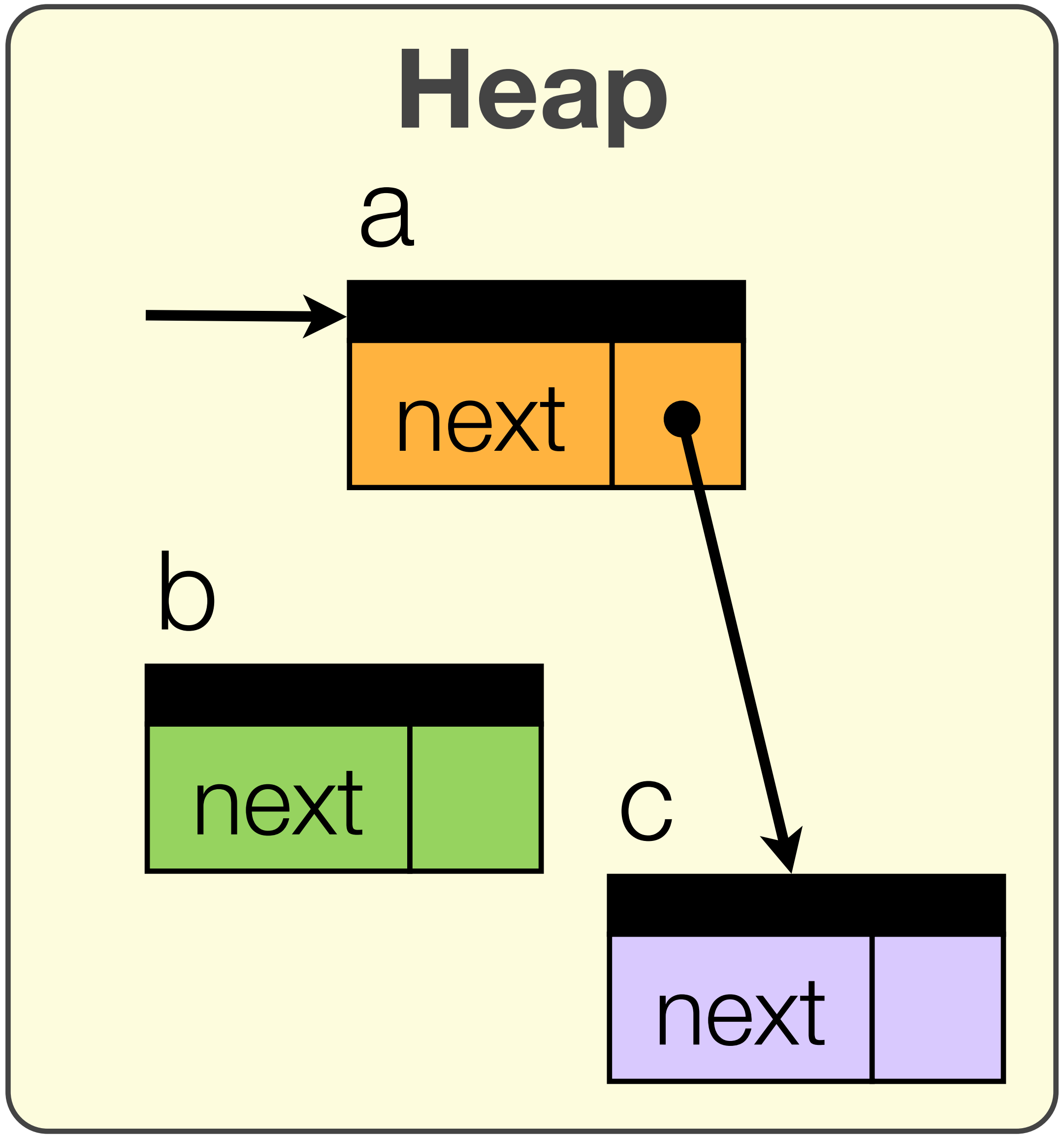


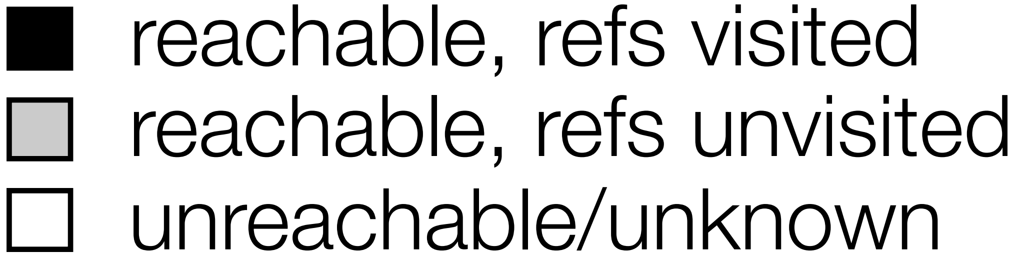
■ reachable, refs visited
▨ reachable, refs unvisited
□ unreachable/unknown

mark-sweep

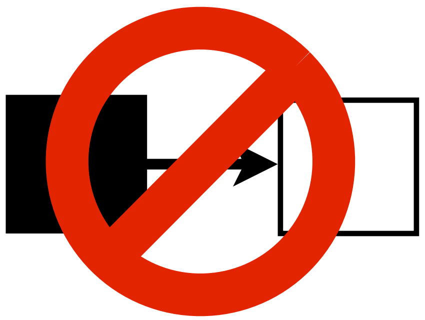


GC thread	Mutator
gray(a)	
gray(a.next)	
black(a)	
	n := a.next
	a.next := a.next.next
	gray(c)
	n.next := null
gray(b.next)	
black(b)	
gray(c.next)	
black(c)	





mark-sweep



GC thread **Mutator**

gray(a)

gray(a.next)

black(a)

n := a.next

a.next := a.next.next

gray(c)

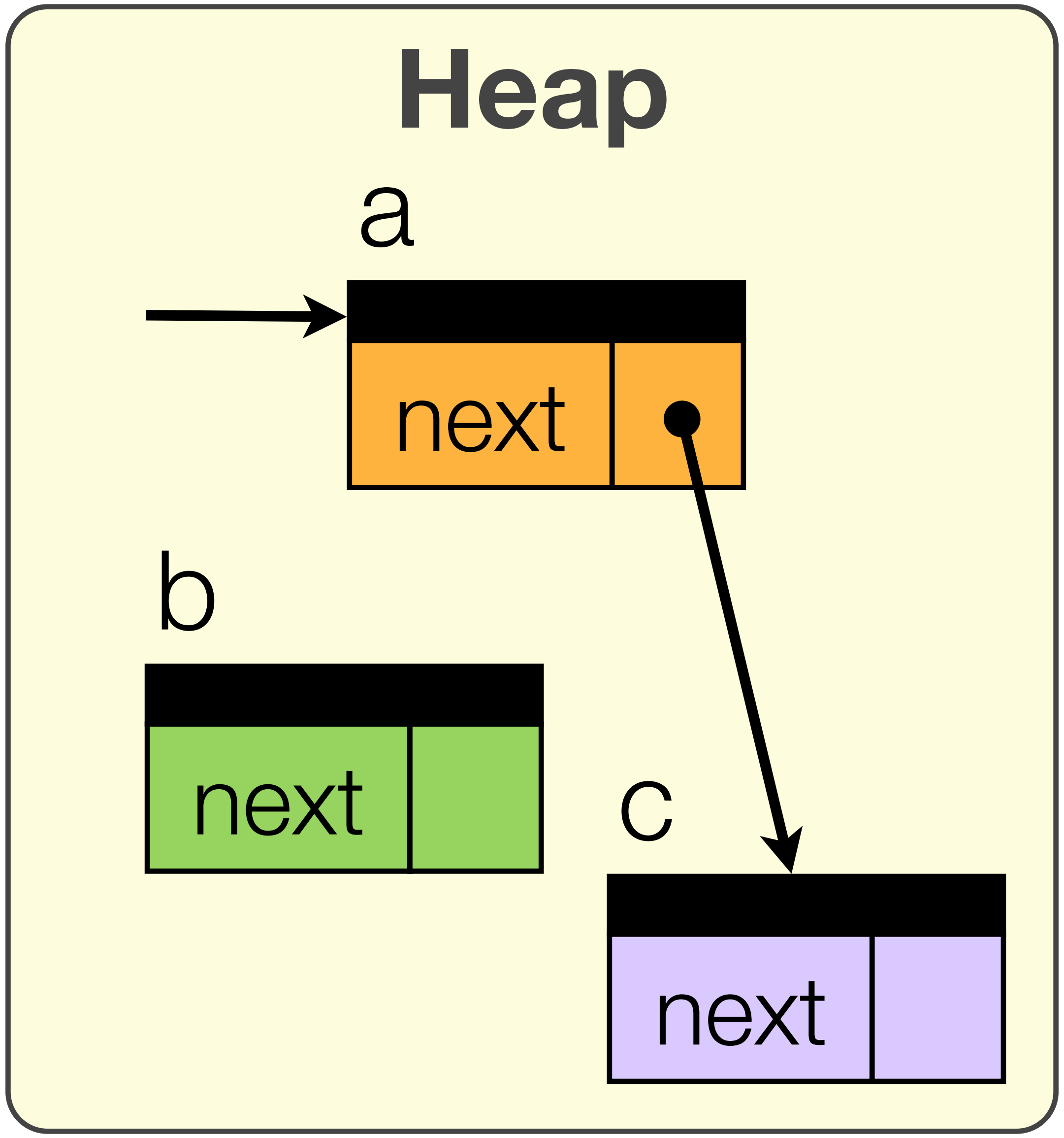
n.next := null

gray(b.next)

black(b)

gray(c.next)

black(c)

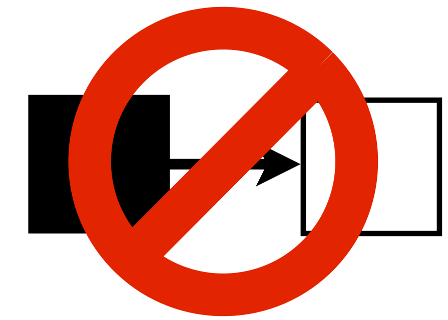


■ reachable, refs visited

▢ reachable, refs unvisited

□ unreachable/unknown

mark-sweep



GC thread | **Mutator**

gray(a)

gray(a.next)

black(a)

exception: gray(c)

n := a.next

a.next := a.next.next

gray(c)

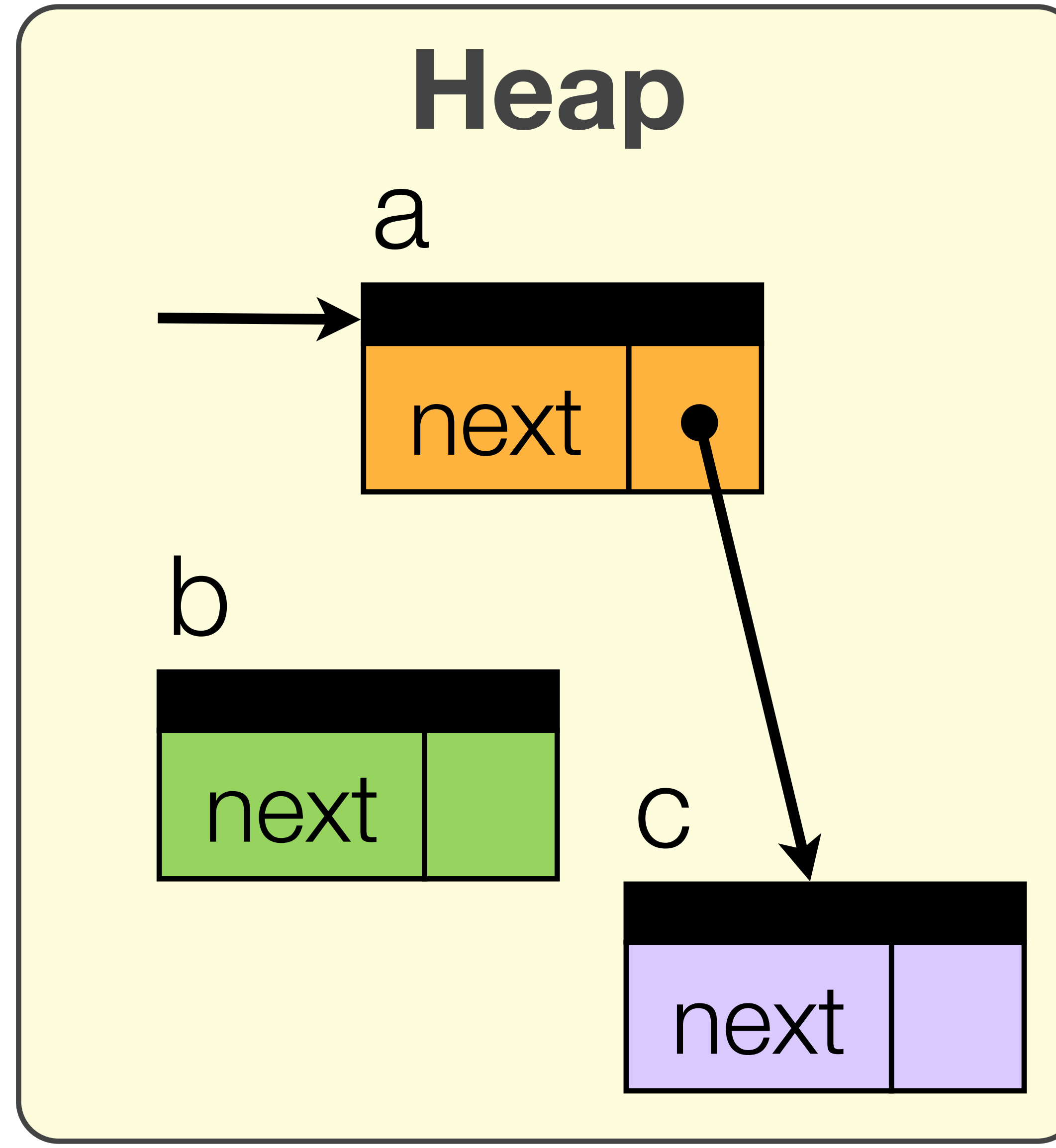
n.next := null

gray(b.next)

black(b)

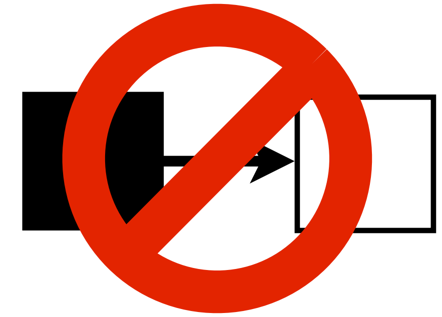
gray(c.next)

black(c)



■ reachable, refs visited
▢ reachable, refs unvisited
□ unreachable/unknown

mark-sweep



GC thread **Mutator**

gray(a)

gray(a.next)

black(a)

exception: gray(c)

n := a.next

a.next := a.next.next

gray(c)

n.next := null

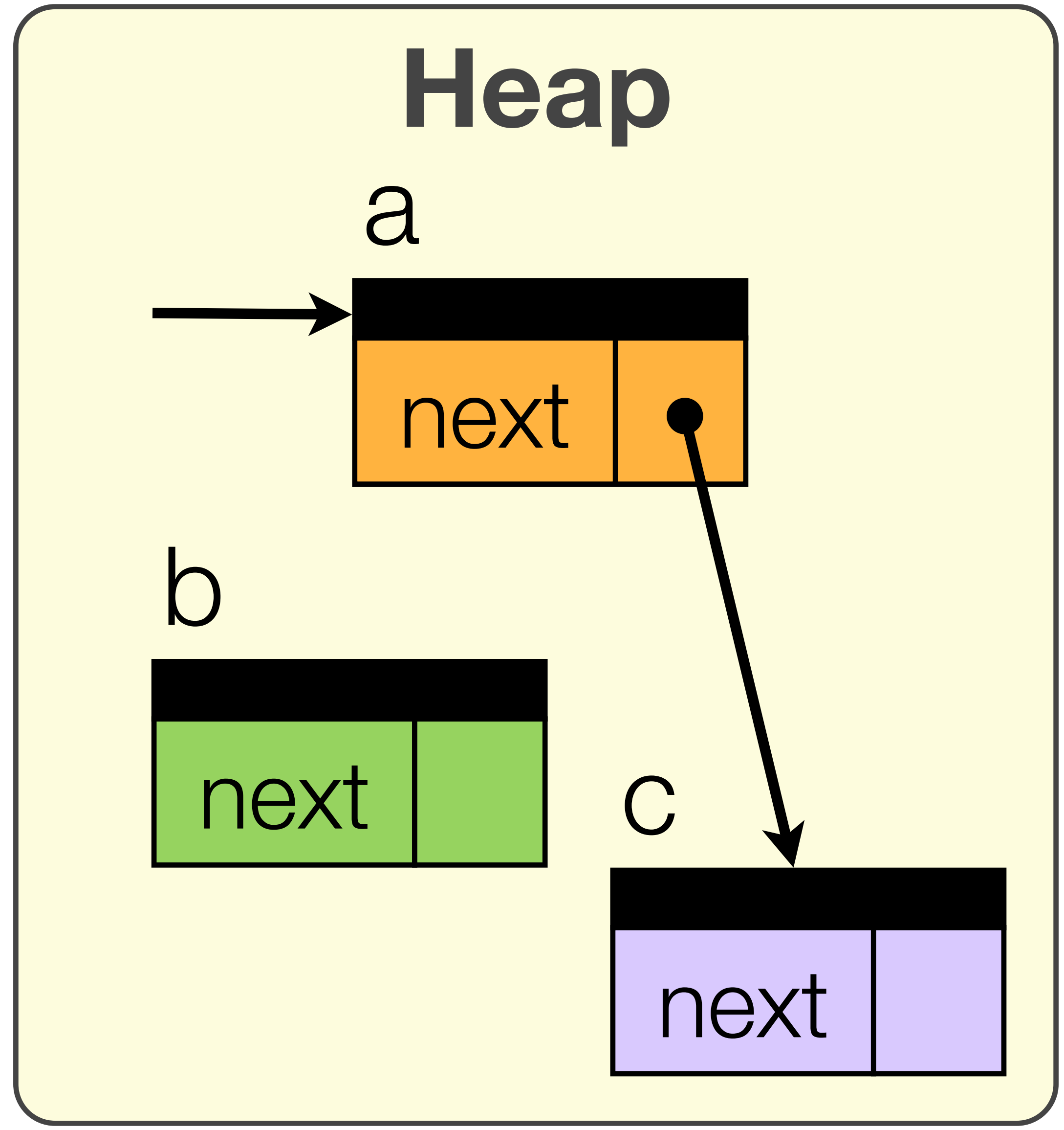
gray(b.next)

black(b)

gray(c.next)

black(c)

exception: ???



■ reachable, refs visited

□ reachable, refs unvisited

□ unreachable/unknown

extensions for GC

Erase tracks
Remove and replace last reads and last writes.

Racy read
Record and execute a read even if it races.

Data-carrying exceptions
Deliver a racing write value to a racing read.

…

SO FAR

Use HW data-race exceptions in runtime systems.

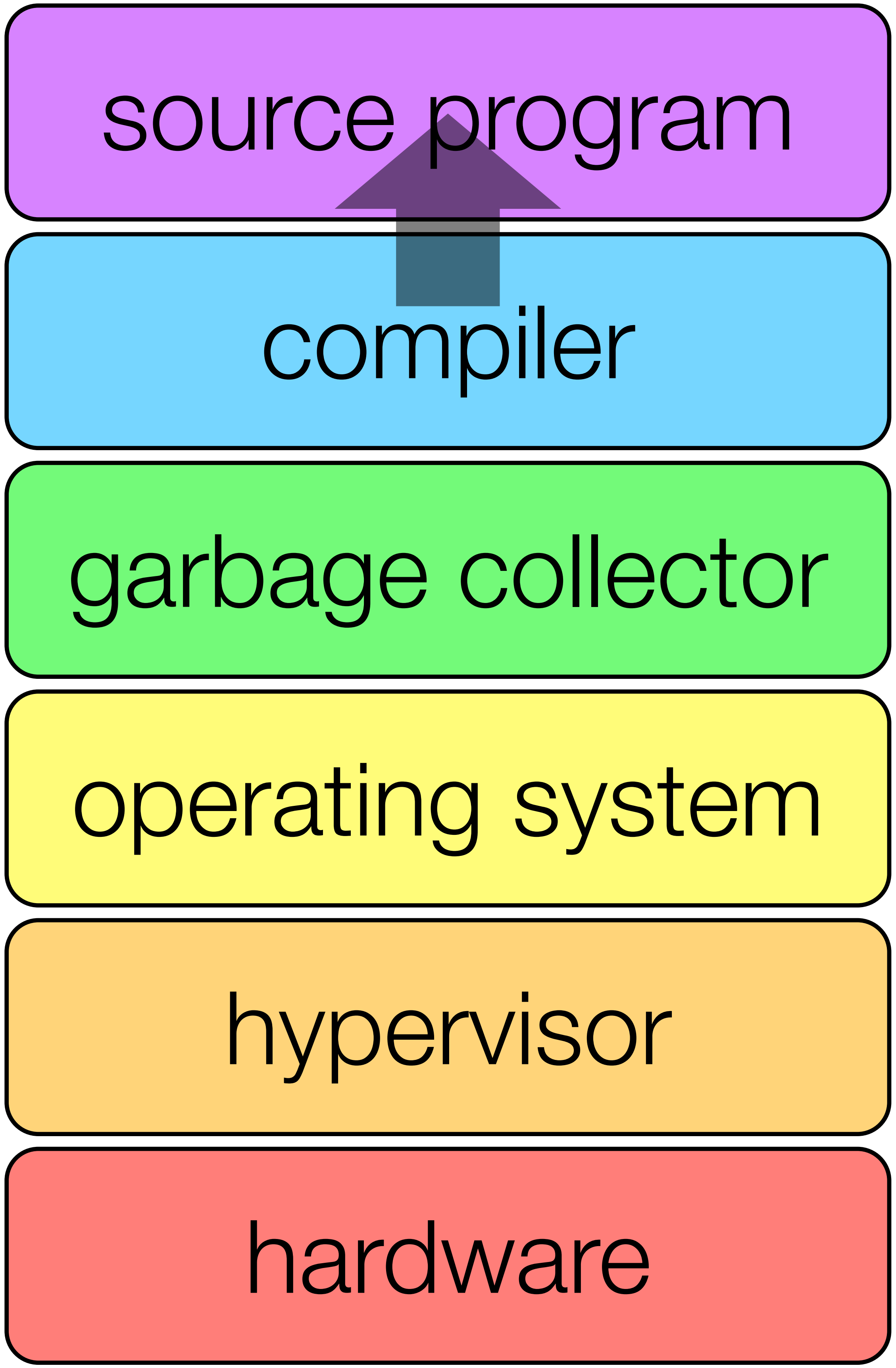
SO FAR

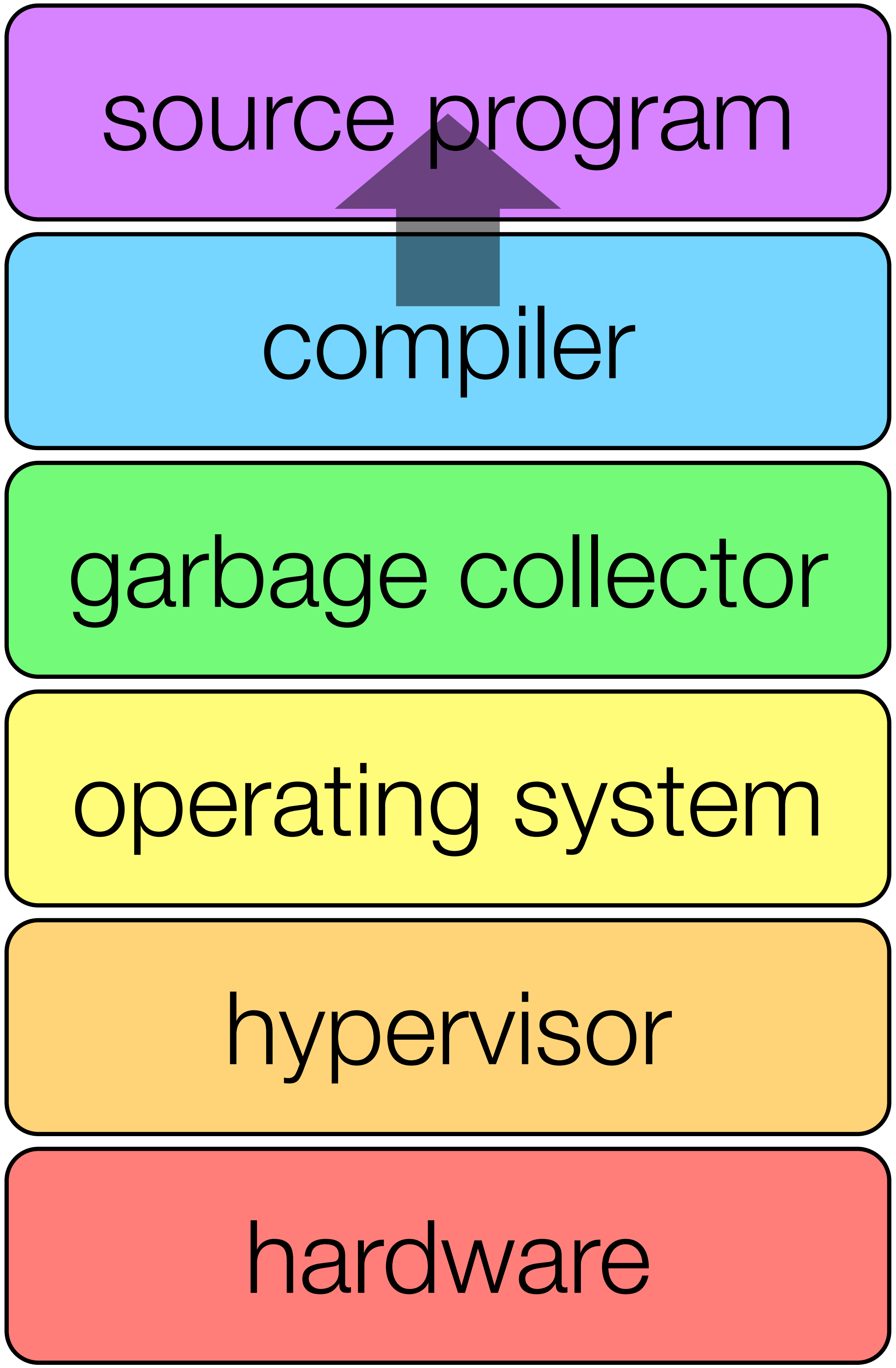
Use HW data-race exceptions in runtime systems.

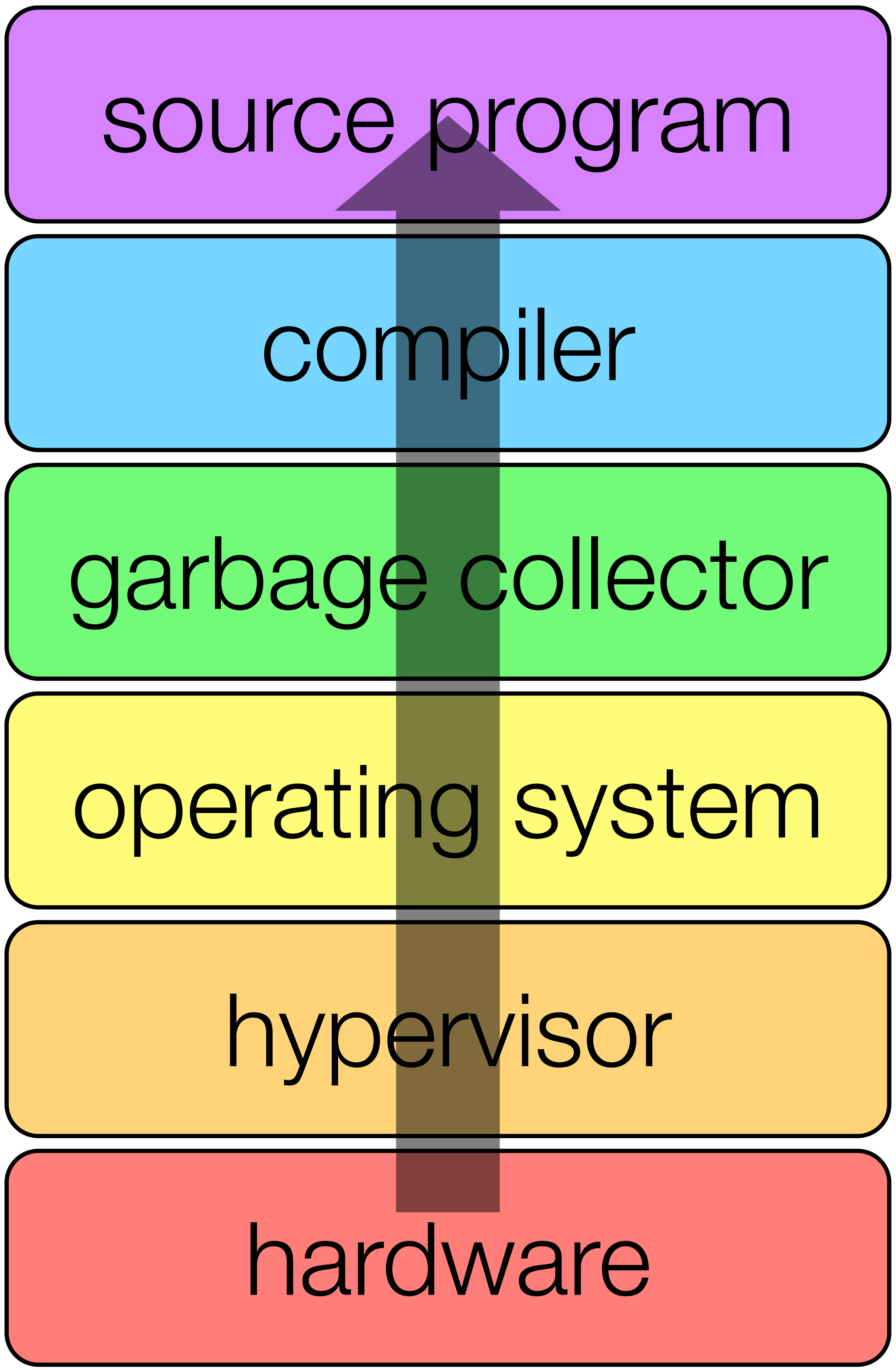
NOW

Do guarantees from HW data-race exceptions apply at the program level?

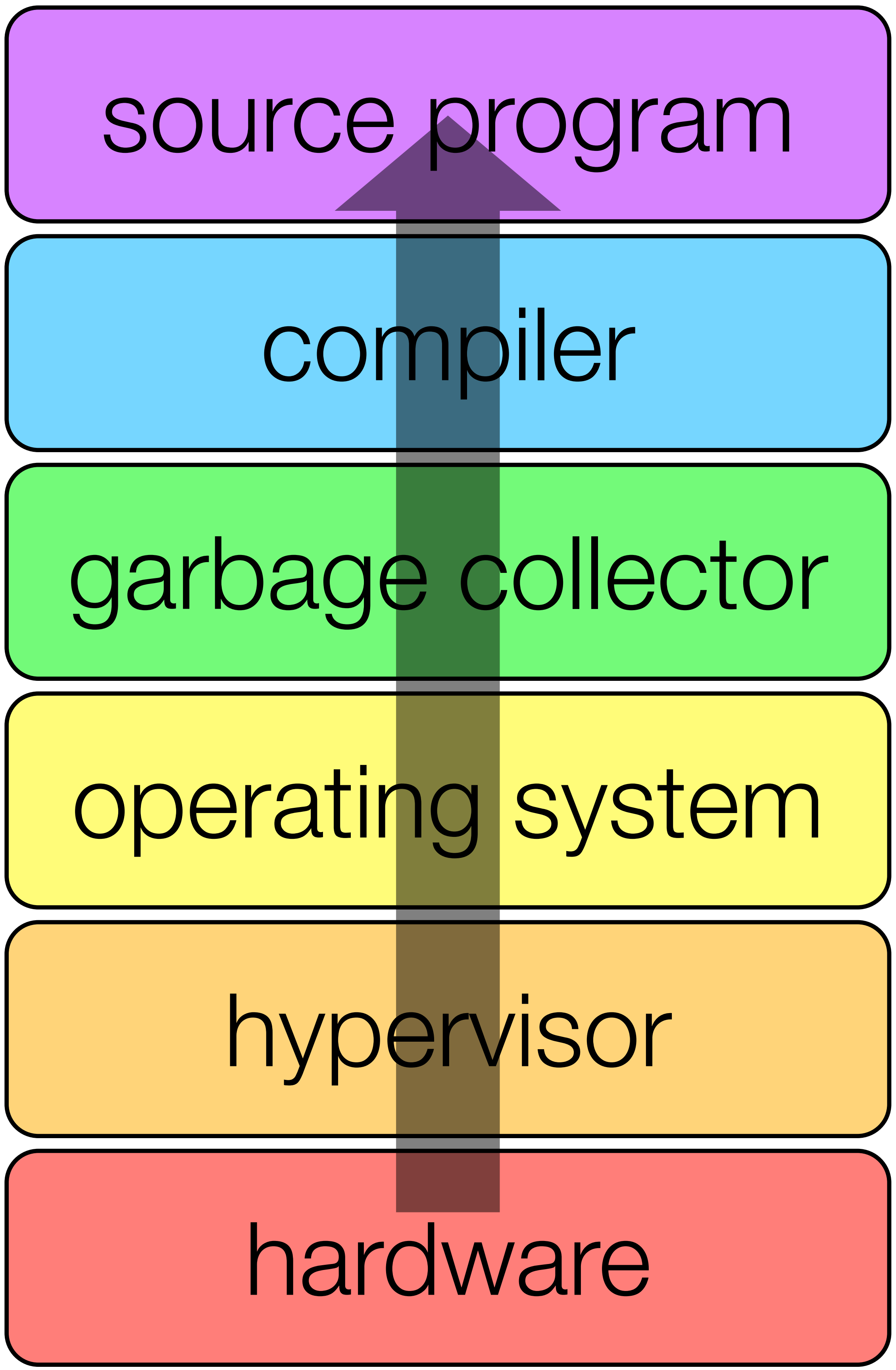
high-level support




source program
compiler
garbage collector
operating system
hypervisor
hardware


source program
compiler
garbage collector
operating system
hypervisor
hardware

non-program accesses and sync.



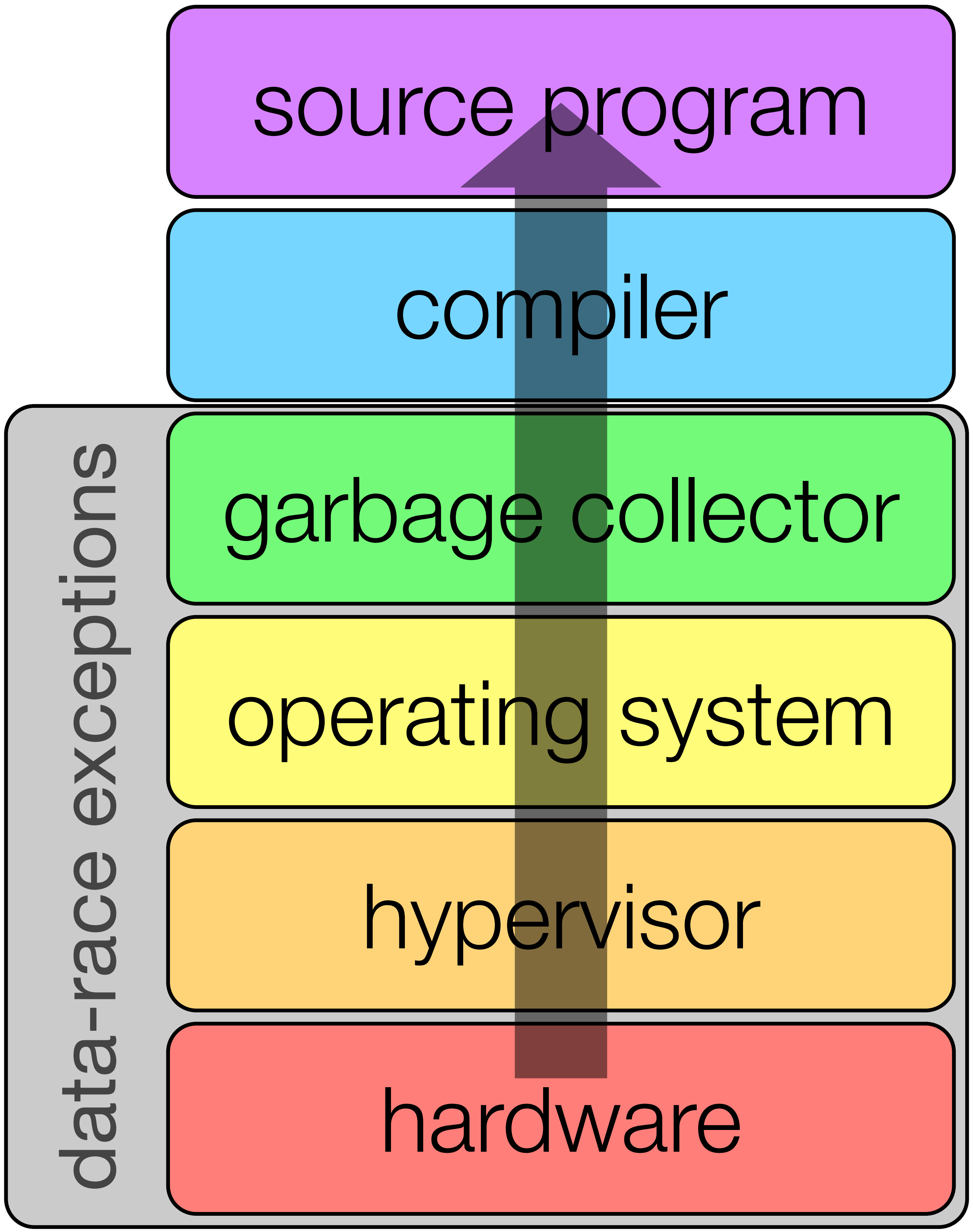
caveats for *any* GC

Movement
Race-detector state must follow moved objects atomically.

Invisibility
Only program heap writes should be seen by the race detector.
GC writes to the heap should be ignored.

No transitive ordering
GC must not induce ordering between mutators.

cooperation vs. abstraction



also in the paper...

More details on concurrent GC using data-race exceptions

Lock elision with no rollback support

Conflict races and conflict-race exceptions

conclusions

Data-race exceptions have benefits beyond the memory model.

Exceptions make races *impossible*.
Attempted races are exceptional, but legal.

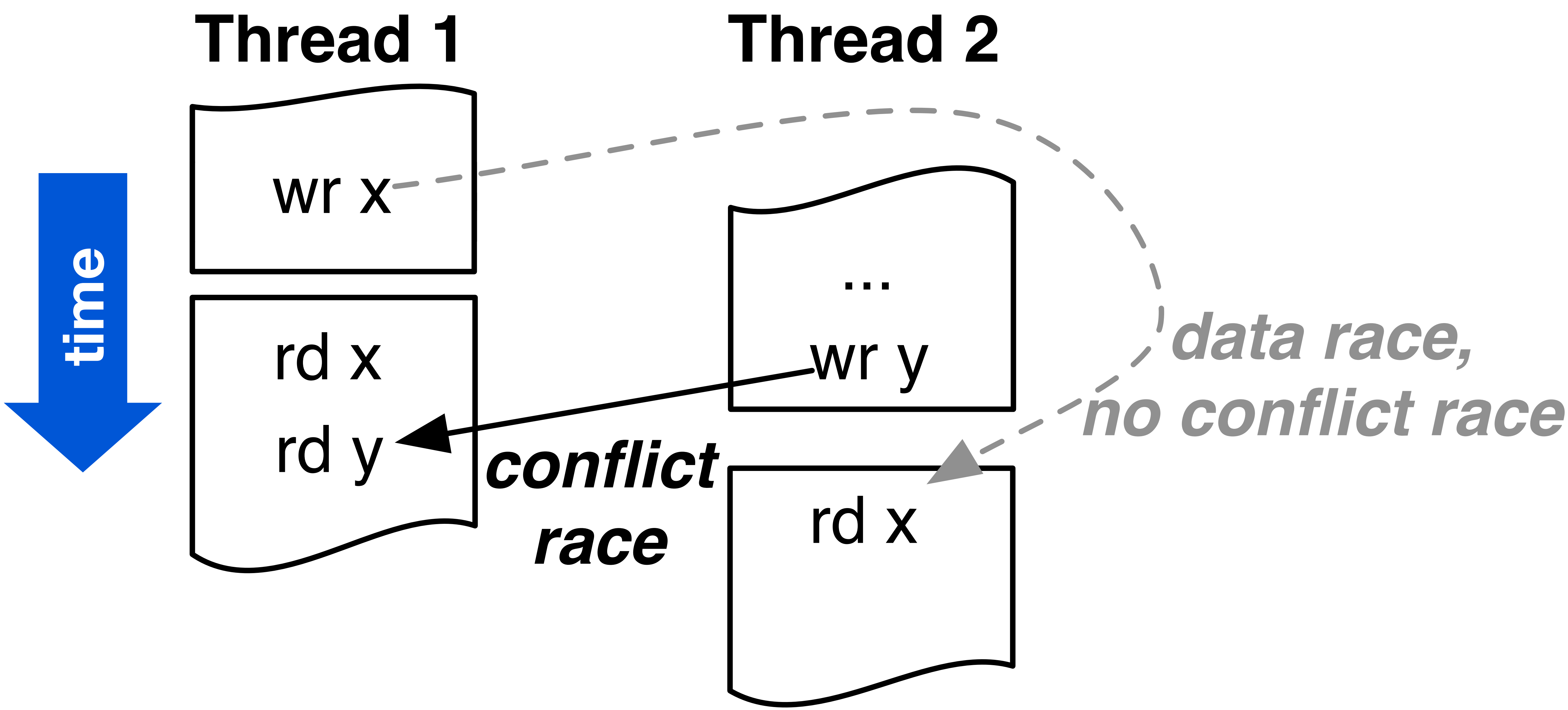
Data-race exceptions enable general conflict detection
for runtime systems like concurrent GC.

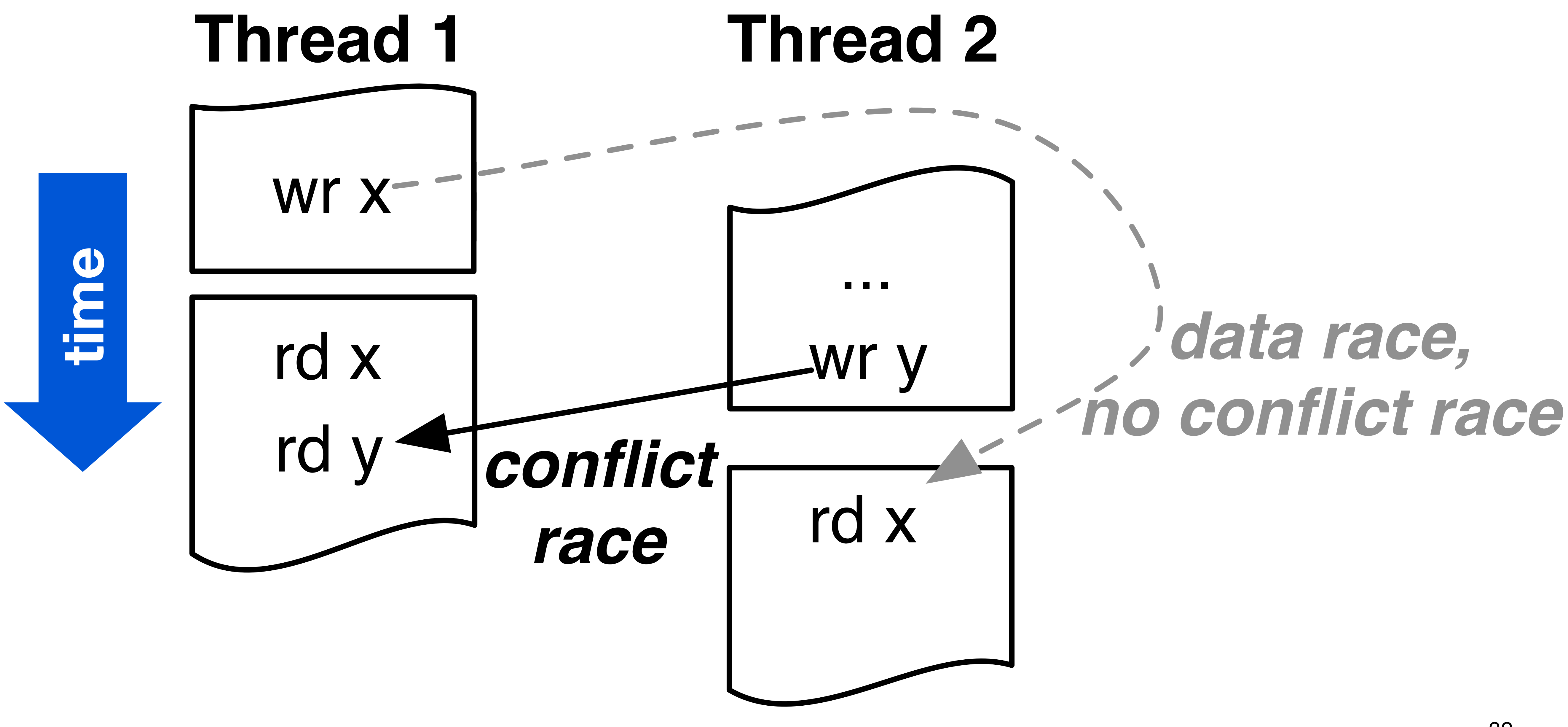
Low-level data-race detection has subtle implications.

This slide intentionally not left blank.

conflict race

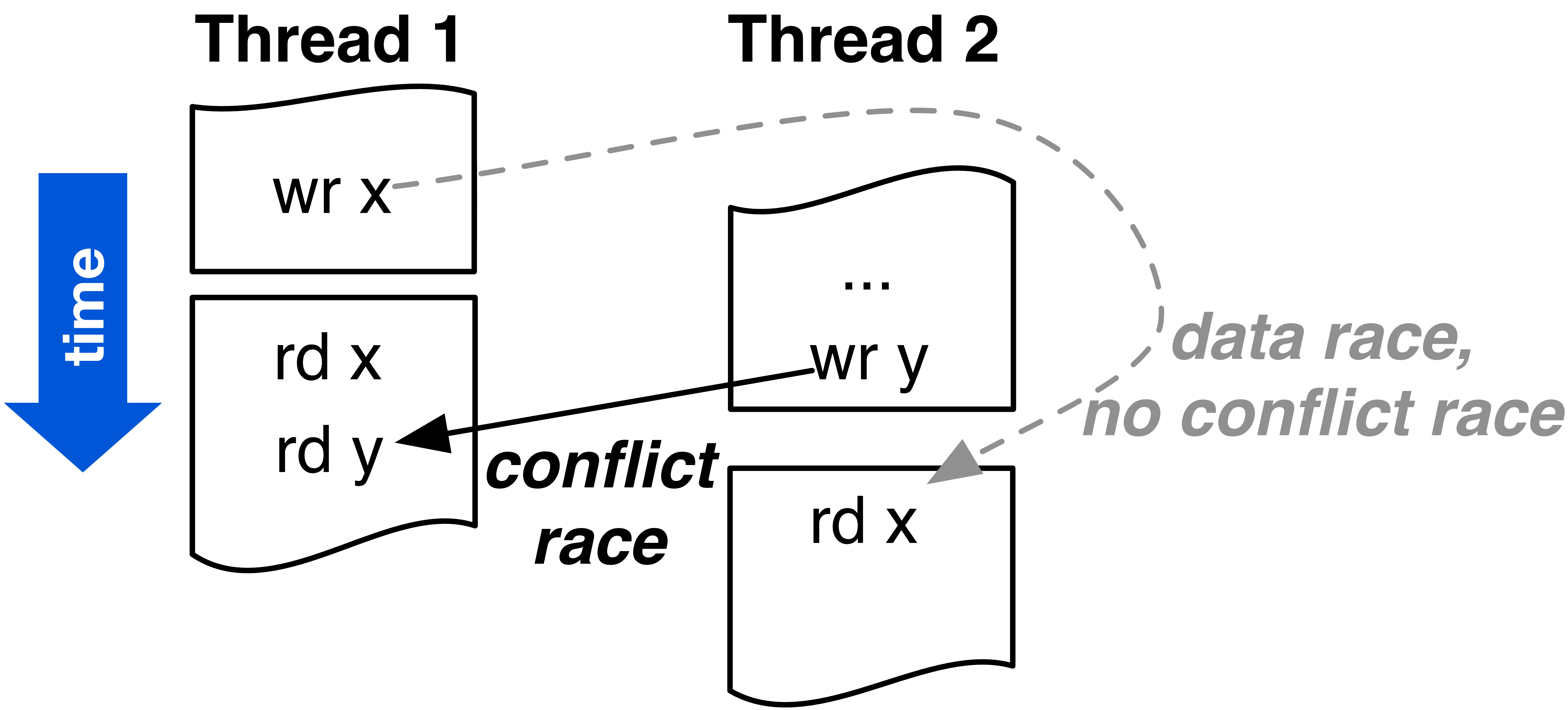
data race across synchronization-free regions running concurrently in real time




Thread 1
Thread 2
time
wr x
rd x
rd y
...
wr y
rd x
conflict race
data race, no conflict race

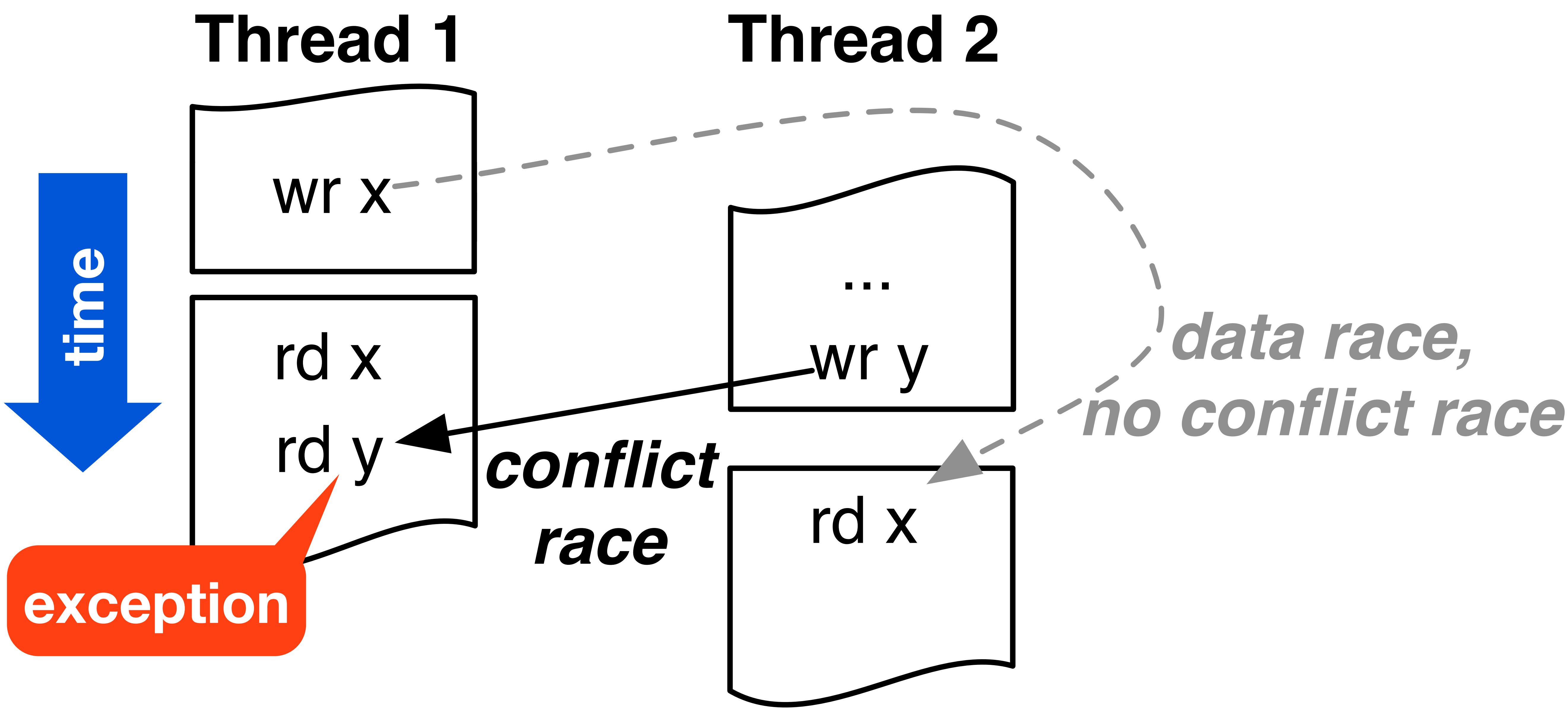
conflict-race exceptions

guarantee sequential consistency or exception

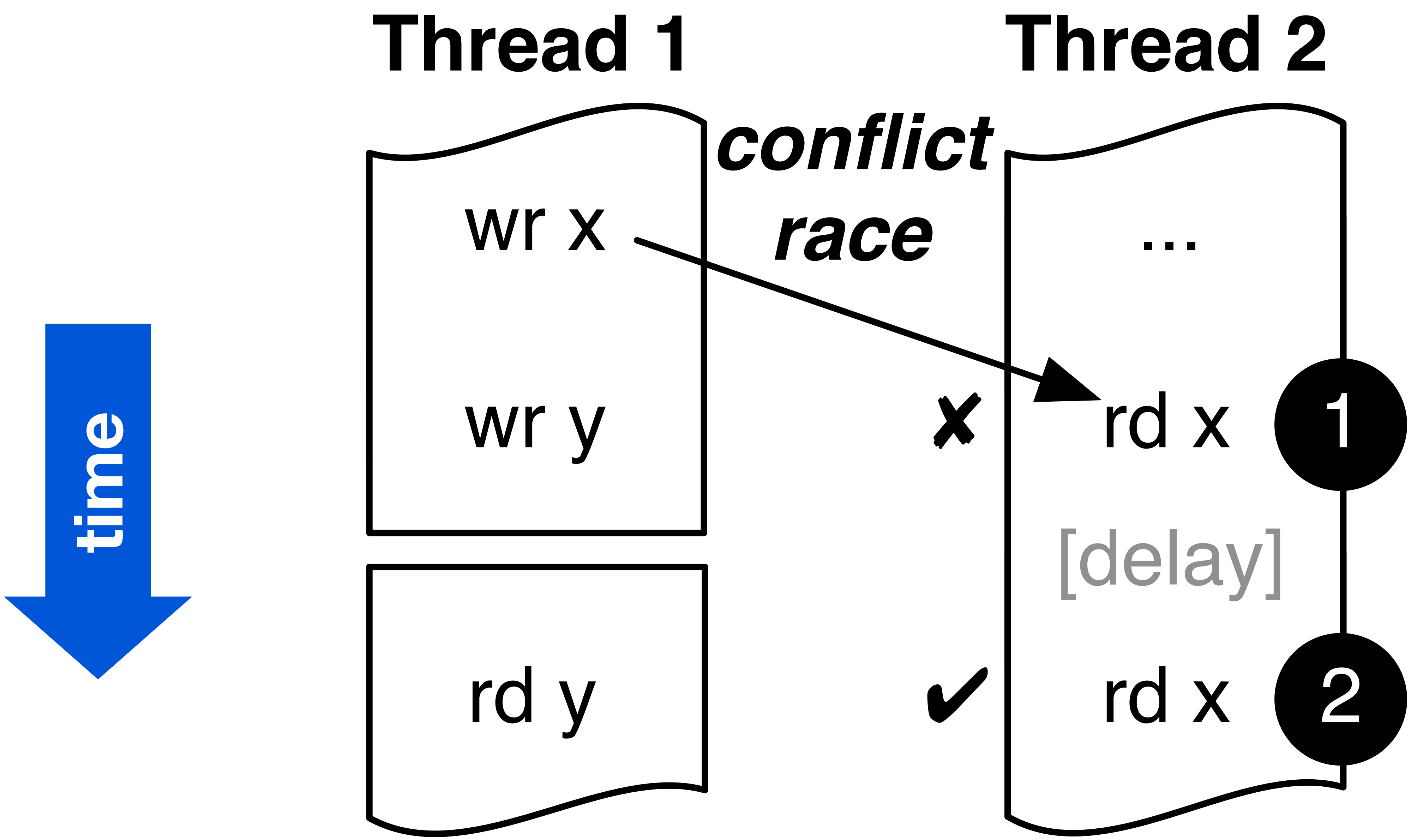


conflict-race exceptions

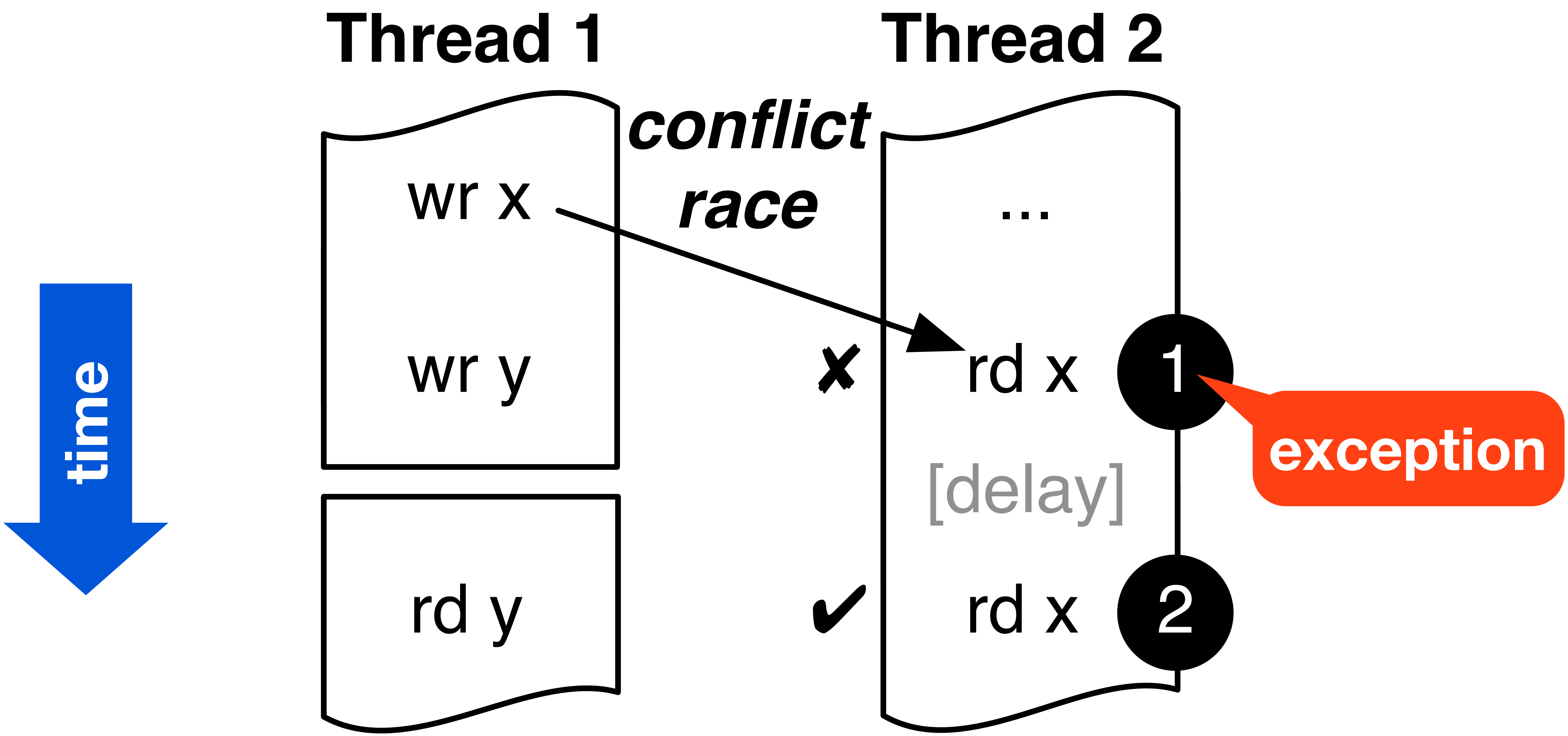
guarantee sequential consistency or exception



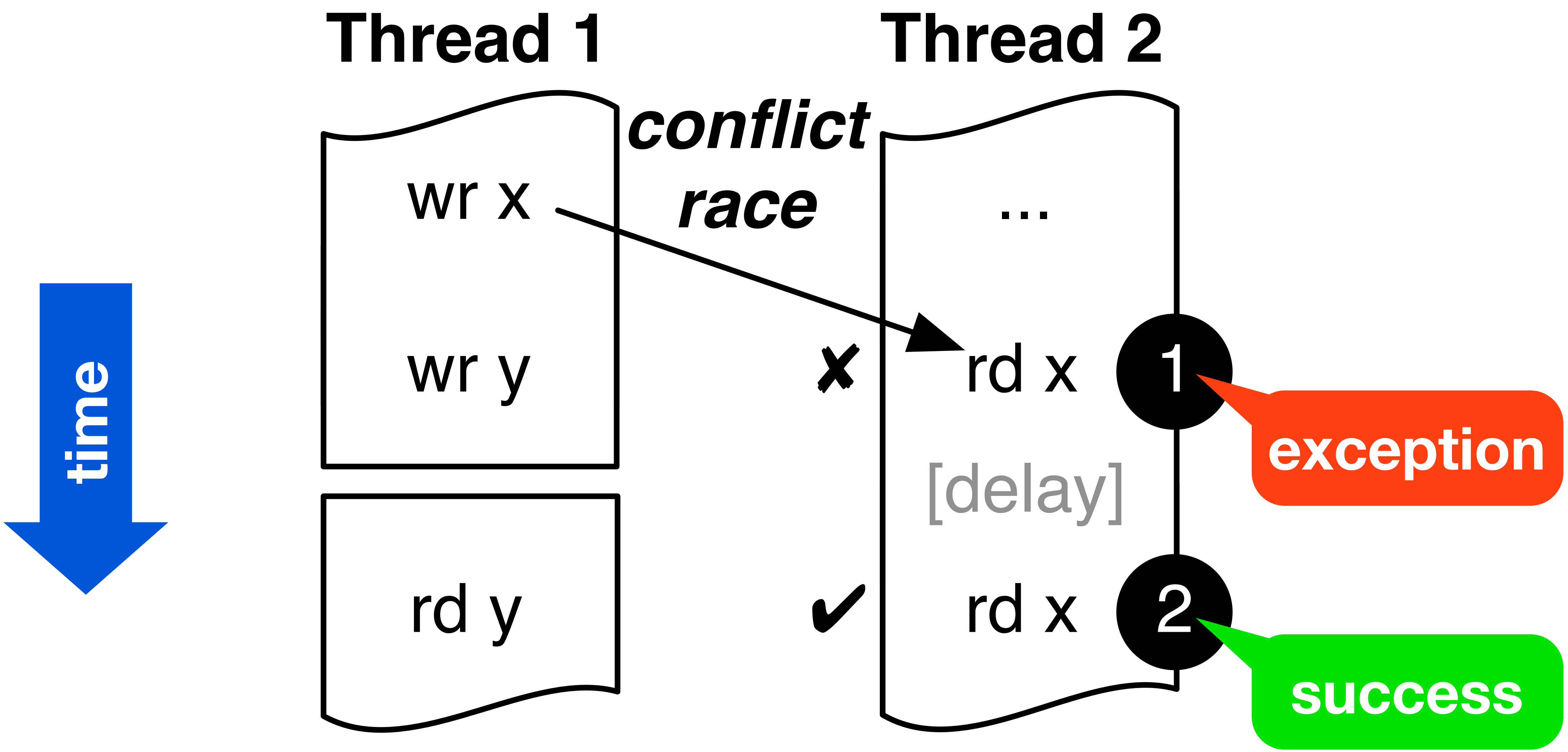
Best-effort automatic recovery from conflict races



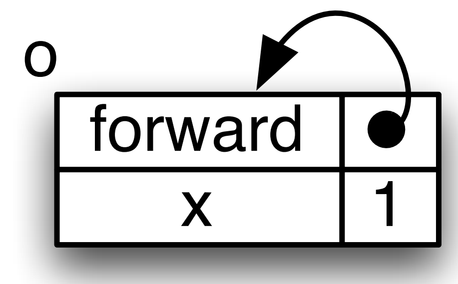
Best-effort automatic recovery from conflict races



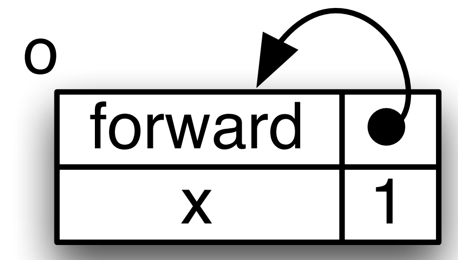
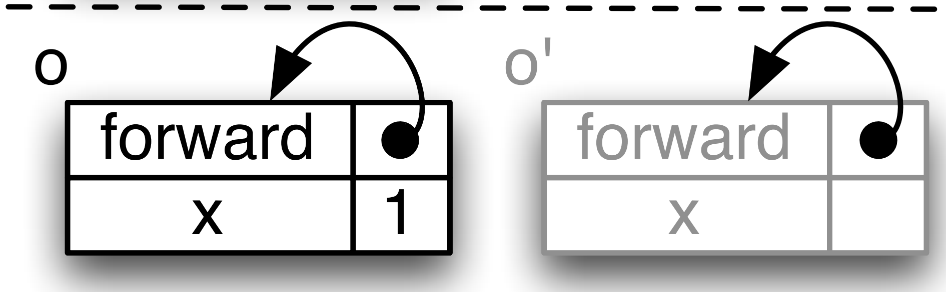
Best-effort automatic recovery from conflict races



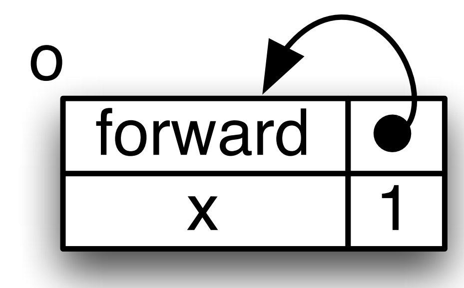
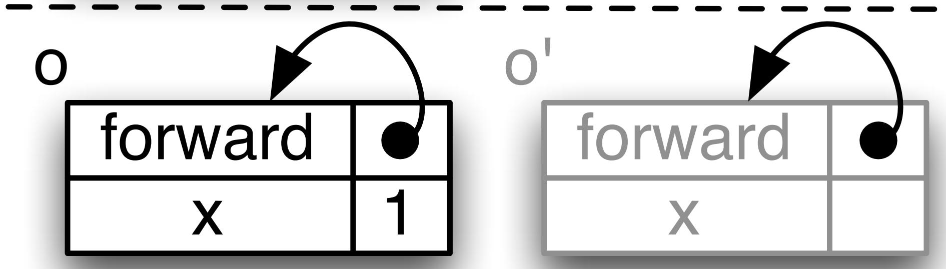
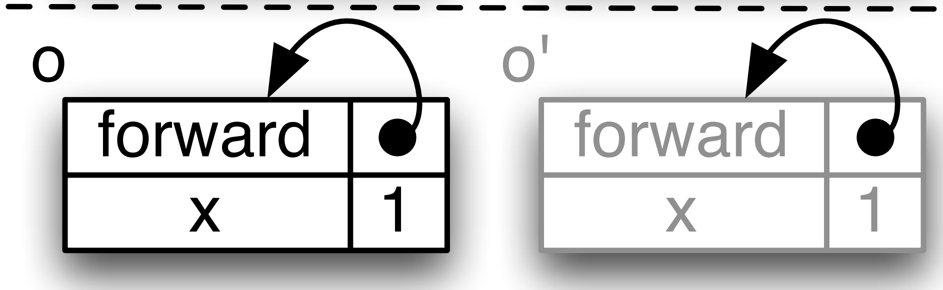
moving/copying

	GC Thread	**Mutator**	**Resulting Heap**
0		p = o.forward p.x = 1	

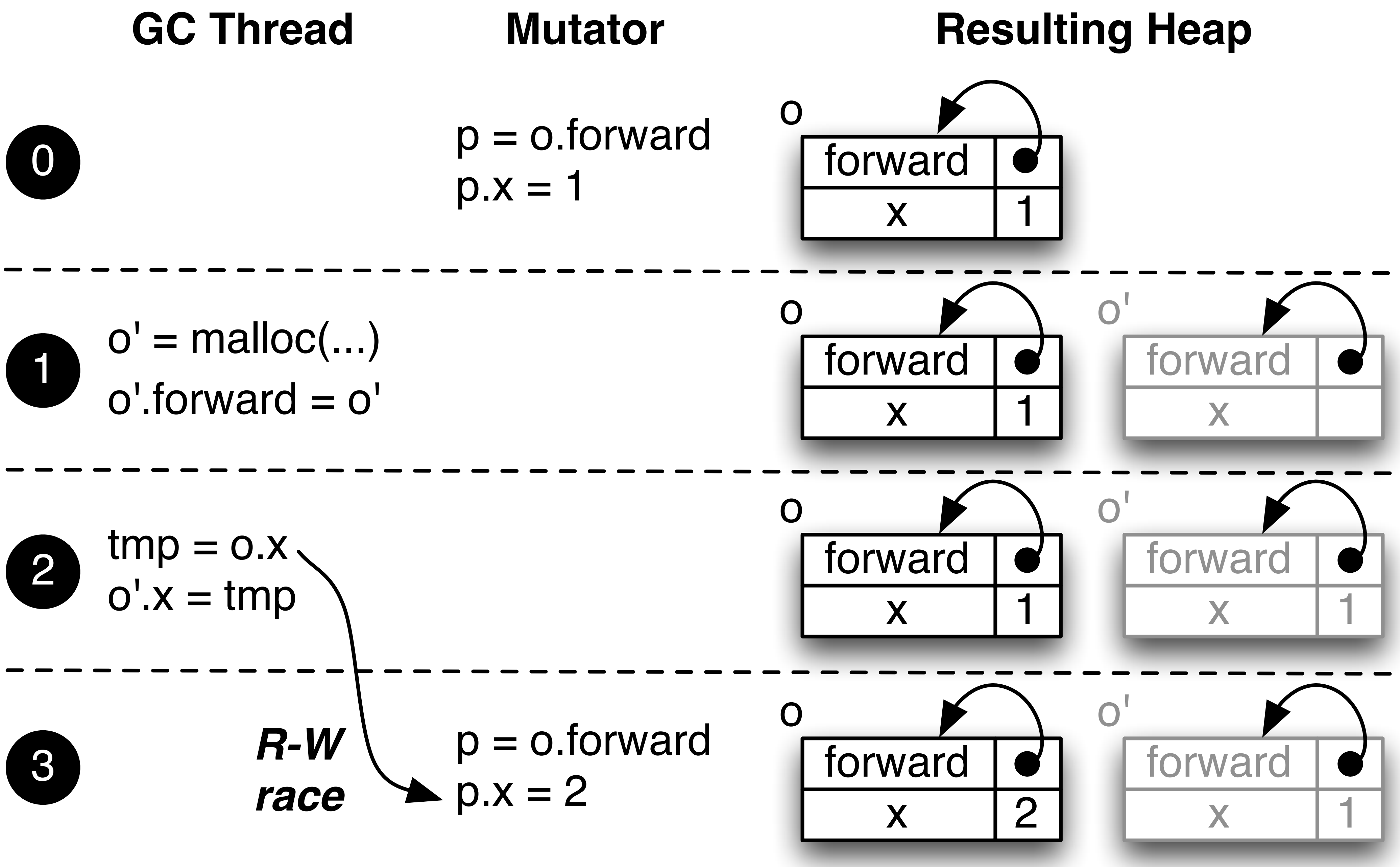
moving/copying

	GC Thread	Mutator	Resulting Heap
0		p = o.forward p.x = 1	
1	o' = malloc(...) o'.forward = o'		

moving/copying

	GC Thread	Mutator	Resulting Heap
0		p = o.forward p.x = 1	
1	o' = malloc(...) o'.forward = o'		
2	tmp = o.x o'.x = tmp		

moving/copying



moving/copying