Forgetful Memoization in Ocaml

François Bobot
joint work with Jean-Christophe Filliâtre and Andrei Paskevich
ProVal team, Orsay, France

OcamlMeeting
April 2011

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche SACLAY - ÎLE-DE-FRANCE

LABORATOIRE DE RECHERCHE EN INFORMATIQUE

UNIVERSITÉ PARIS-SUD 11

Memoization

Computing a function with a cache

```
let memo_f =
  let cache = H.create () in
  fun k →
    try H.find cache k
    with Not_found →
      let v = f k in
      H.add cache k v;
      v

let v1 = memo_f k1 in
...
let v2 = memo_f k2 in (* k2=k1 => O(1) *)
```

Avoid Memory Leaks

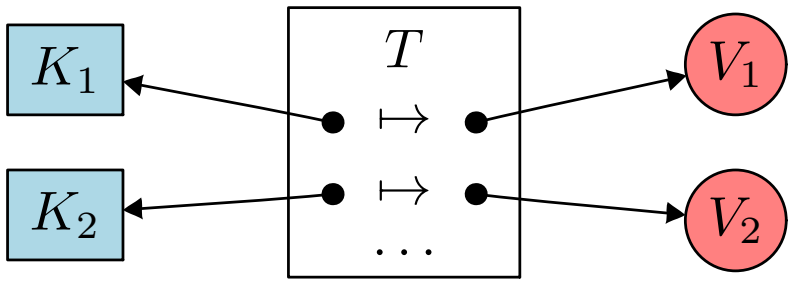
If a key is not needed anymore,
we want to remove the entry from the cache

Particular case: heap-allocated keys
not needed anymore means not reachable (apart from the cache)

Some (Partial) Solutions

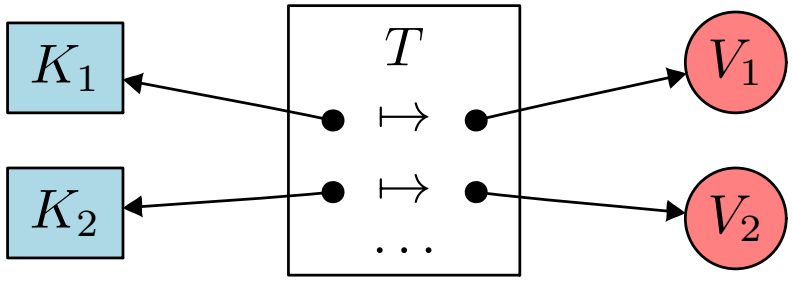
Naive Solution

T is a traditional dictionary data structure
(hash table, balanced tree, etc.)



Naive Solution

T is a traditional dictionary data structure
(hash table, balanced tree, etc.)



major drawback

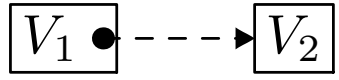
T reachable $\Rightarrow$ all keys and values bound in T are also reachable

conclusion

T should not hold pointers to keys

Weak Pointers

a value can weakly point to another value, depicted

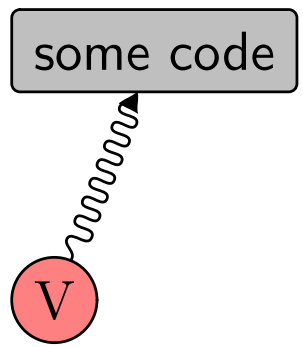


a value not yet reclaimed can be accessed via a weak pointer

```
val get : α Weak.t → int → α option
```

Finalizers

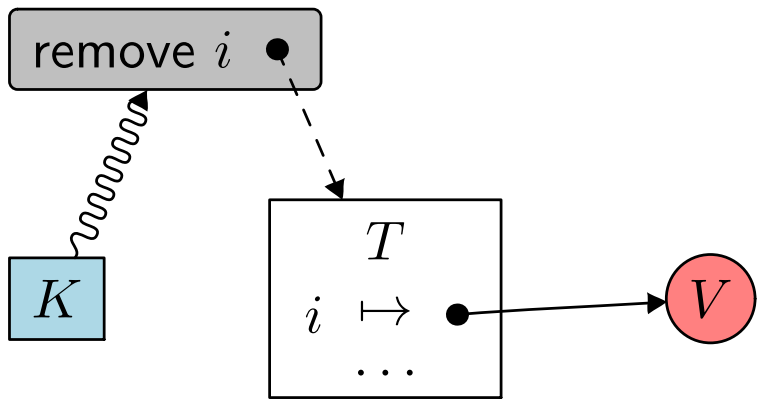
one or several finalizers can be attached to a value



a finalizer is a closure which is executed whenever the corresponding value is going to be reclaimed

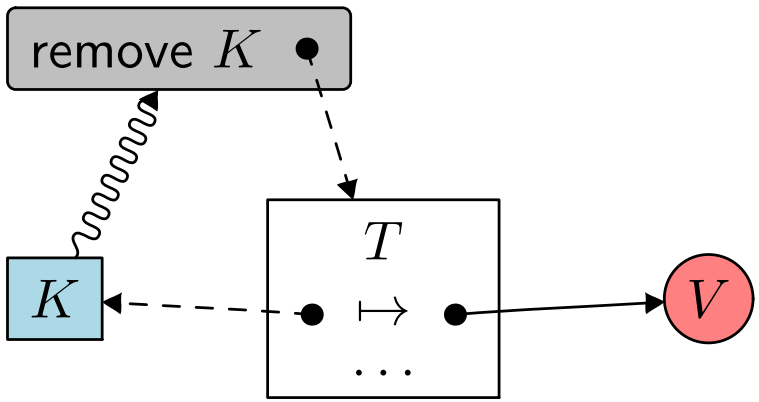
A Better Solution?

K is not used directly as index in T



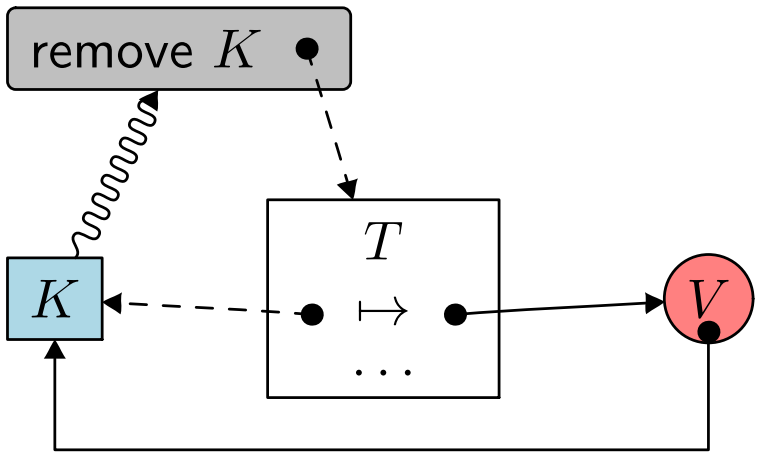
A Better Solution?

K is not used directly as index in T



A Better Solution?

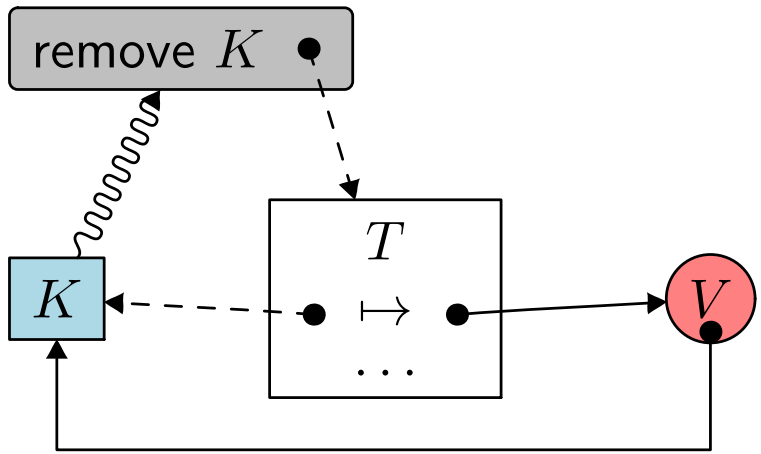
it seems to be a good solution...
but a key can be reachable from a value



preventing K to be reclaimed

A Better Solution?

it seems to be a good solution...
but a key can be reachable from a value



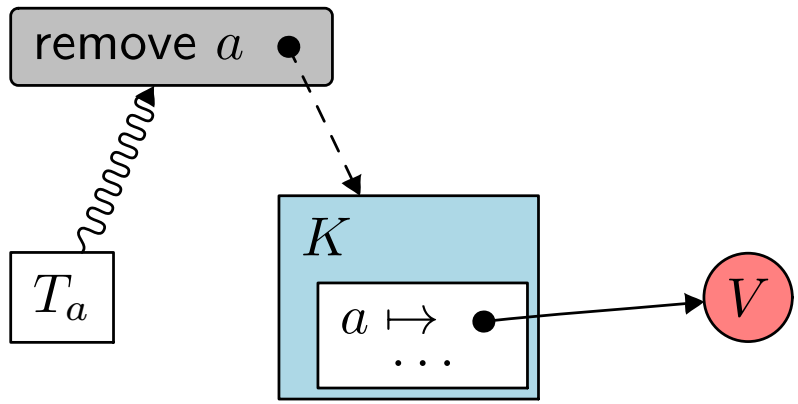
preventing K to be reclaimed

conclusion T should not hold pointers to values either

A Better Solution!

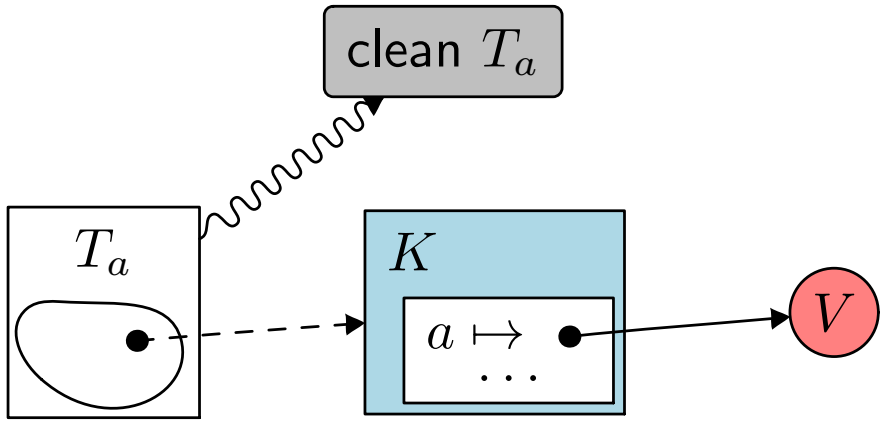
we cannot store bindings inside tables

$\Rightarrow$ let us keep them in keys instead



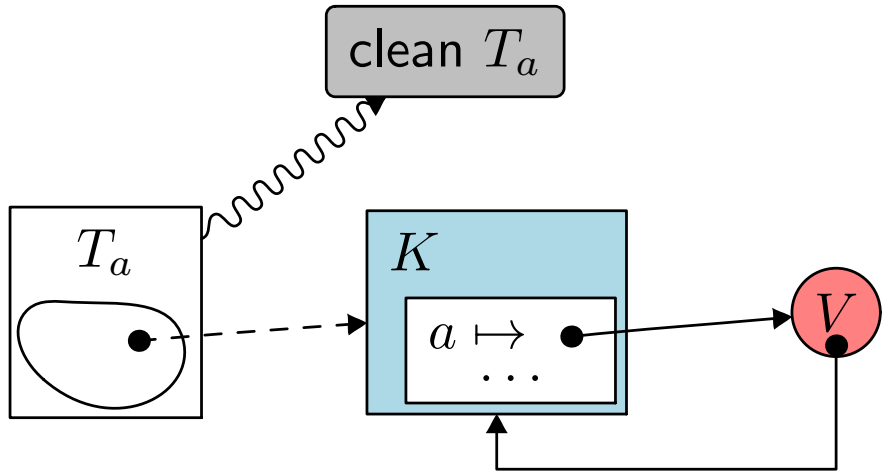
A Better Solution!

improvement: only one finalizer instead of one per key



A Better Solution!

K reachable from V is not a problem anymore



(note: you can implement a similar solution in Haskell using `System.Mem.Weak`)

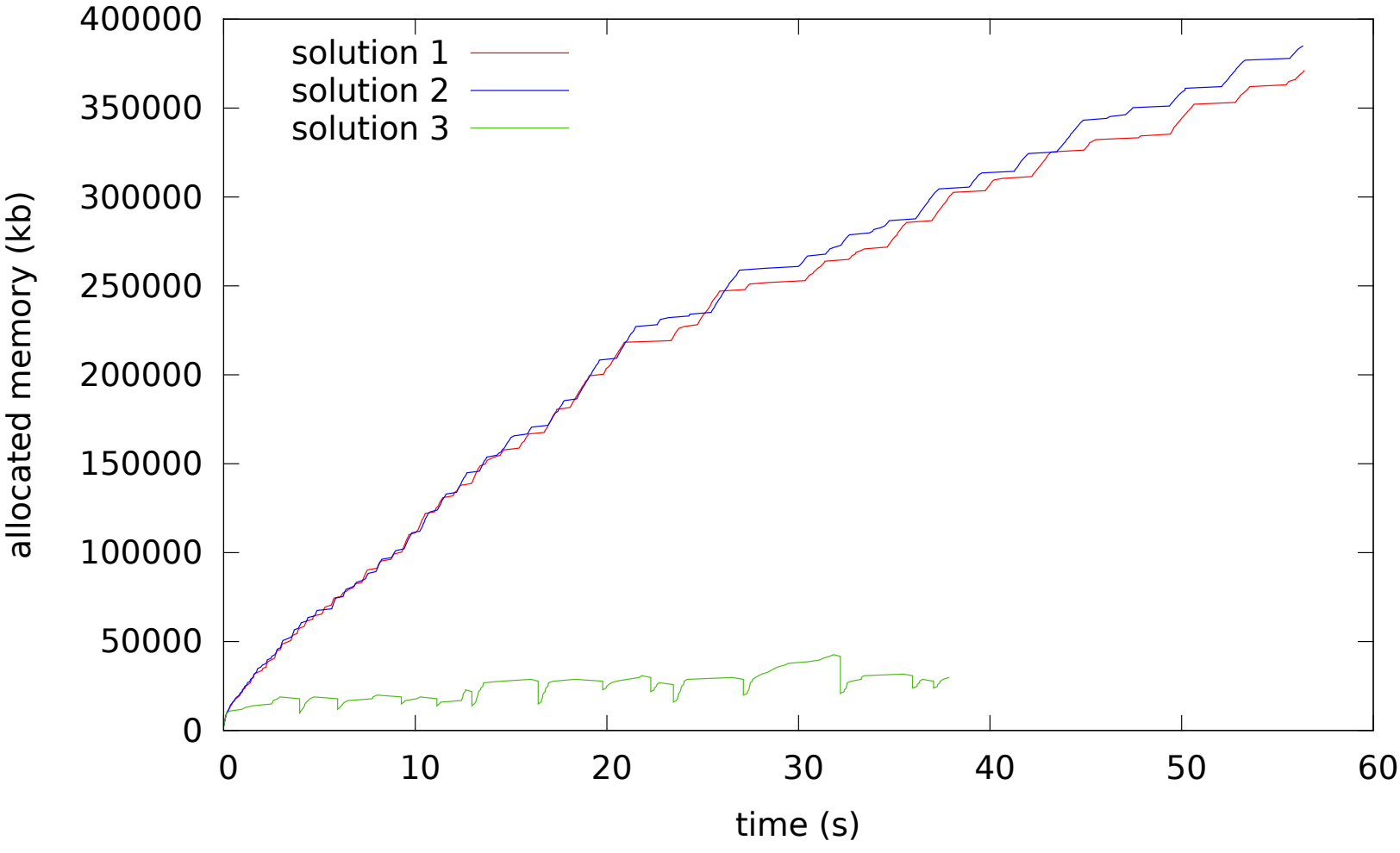
Implementation

implemented as an Ocaml library

```
type cache (* = (int, Obj.t) Hashtbl.t *)

type α key = private {
  node  : α;
  cache : cache;
}

val create : α → α key

val memoize : (α key → β) → (α key → β)
```

Benchmarks

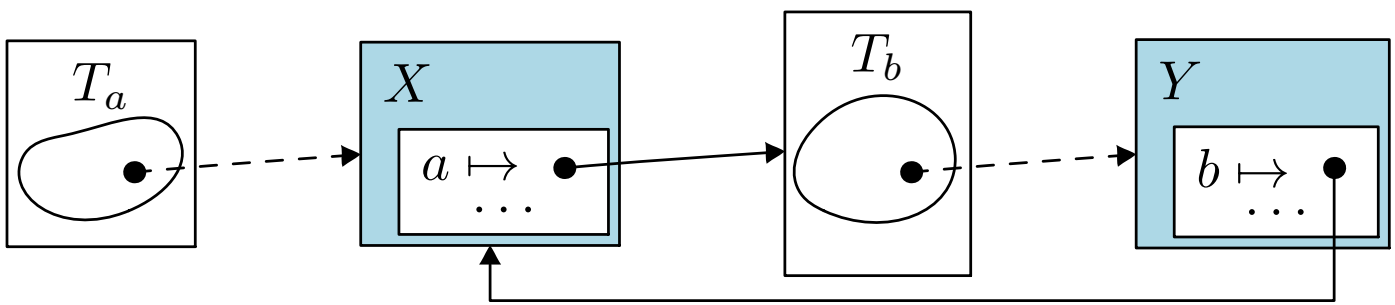
transformation of 1,448 proof tasks sharing subterms



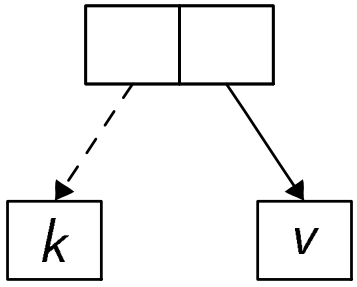
Still not Perfect

of course, the roles of K and T being symmetric,
if T is reachable from V the "cycle issue" is still there

example: we want to memoize the **K** combinator $\mathbf{K}(X, Y) = X$
we first memoize the partial application to X, the result being another memoization table



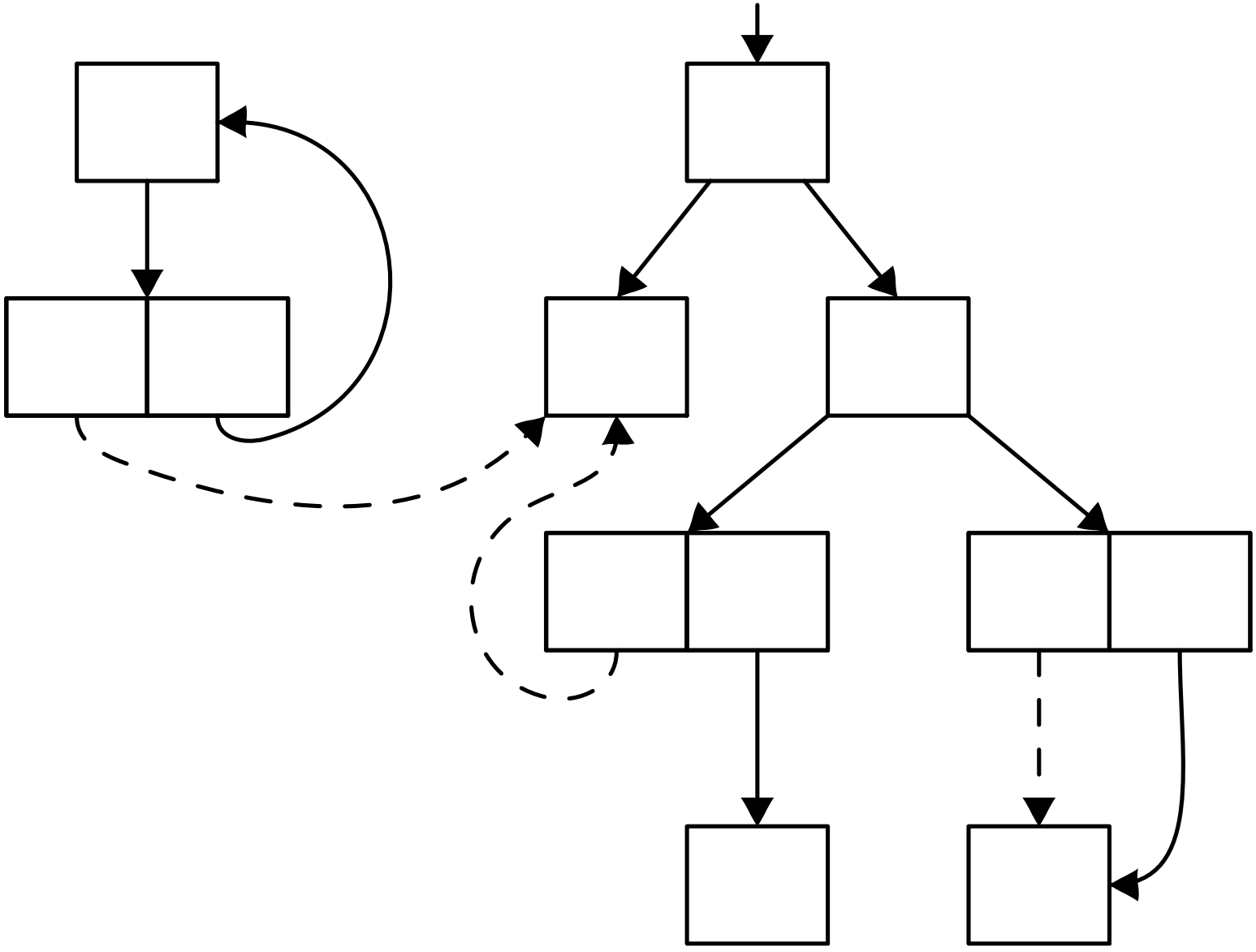
Ephemerons [Hayes 1997]



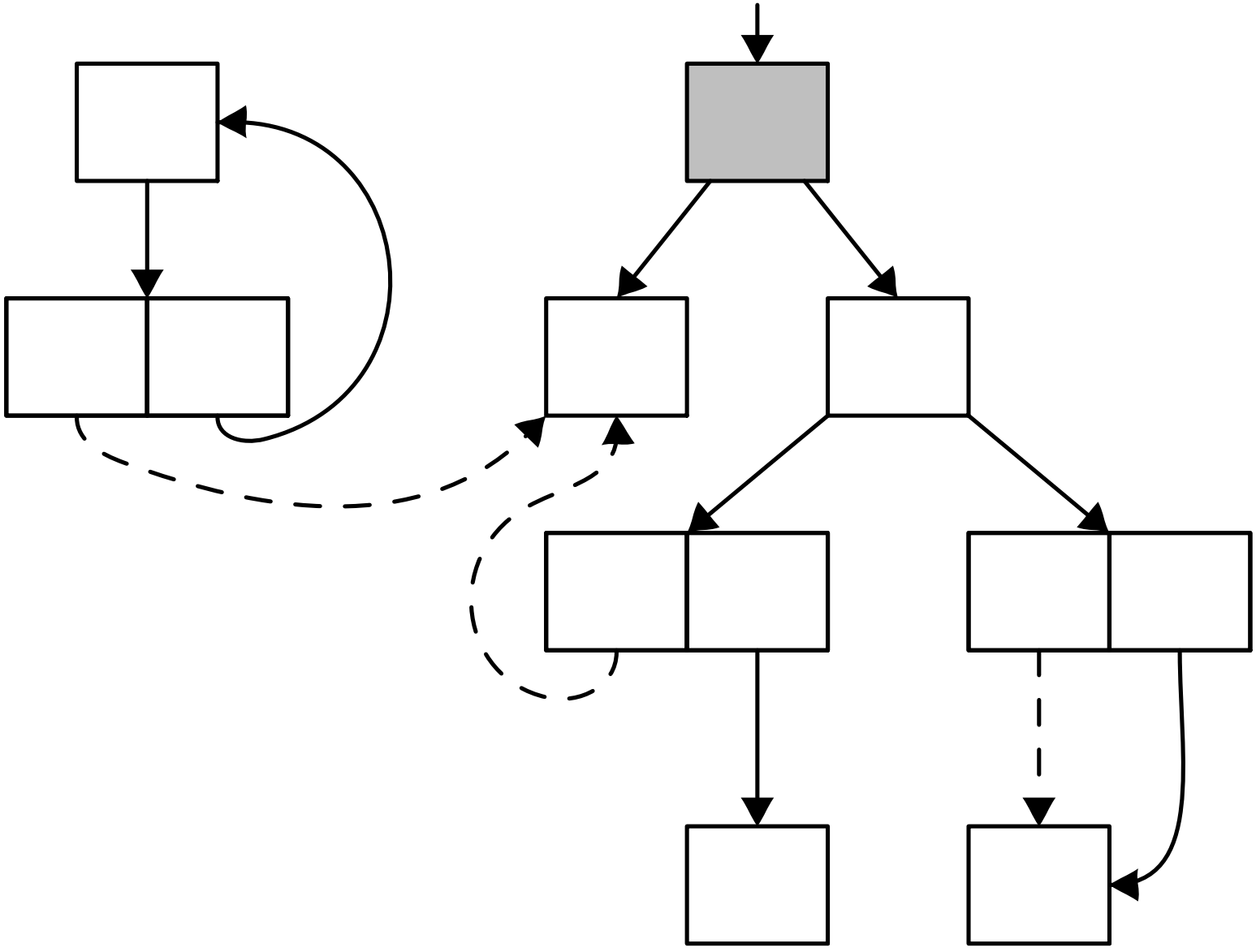
v can be reclaimed
if *k* or the ephemeron can be reclaimed

```
module Ephemeron : sig

    type (α,β) t

    val create : α → β → (α,β) t

    val check : (α,β) t → bool
    val get : (α,β) t → β option
    val get_key : (α,β) t → α option

end
```

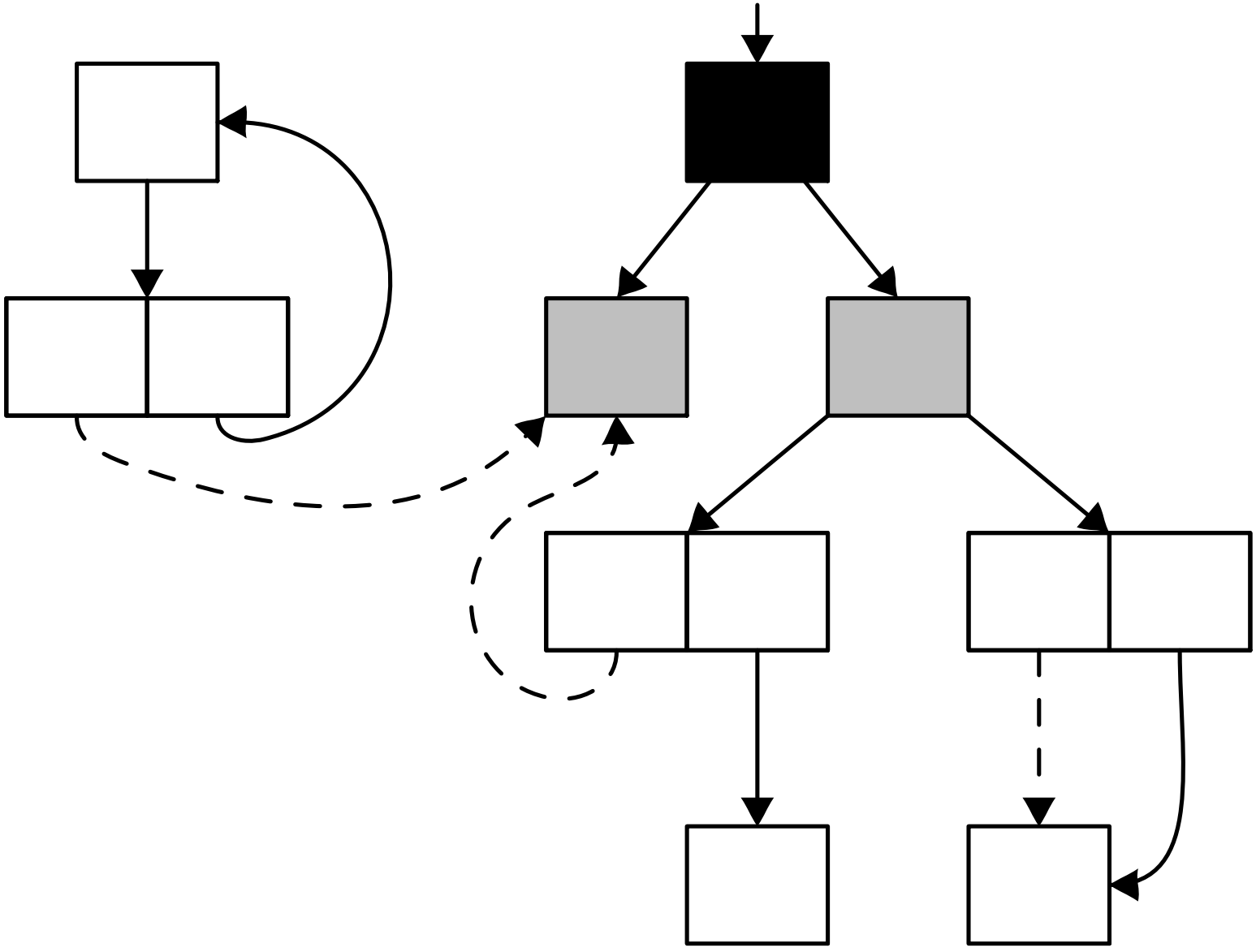
Garbage Collection with Ephemerons



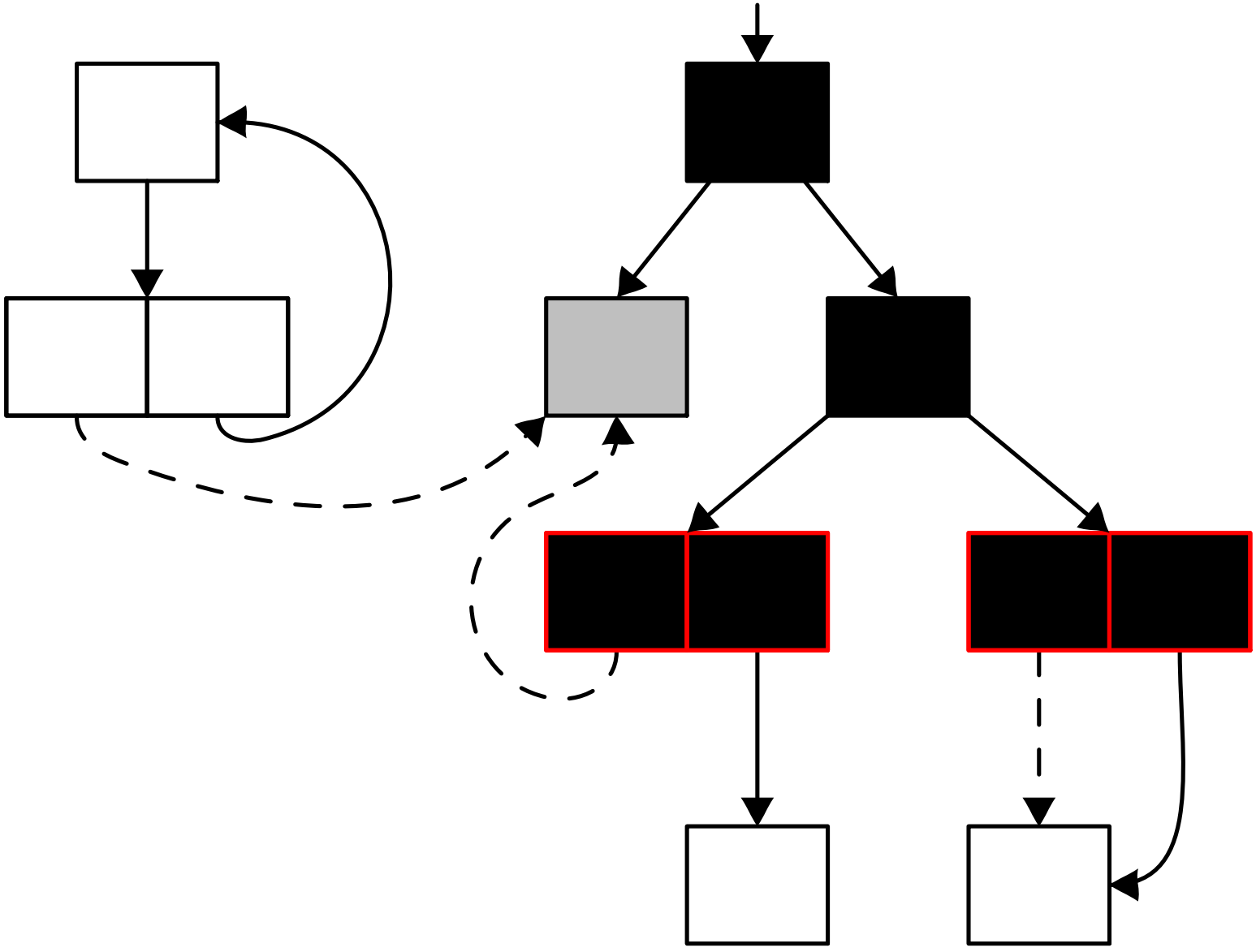
Garbage Collection with Ephemerons



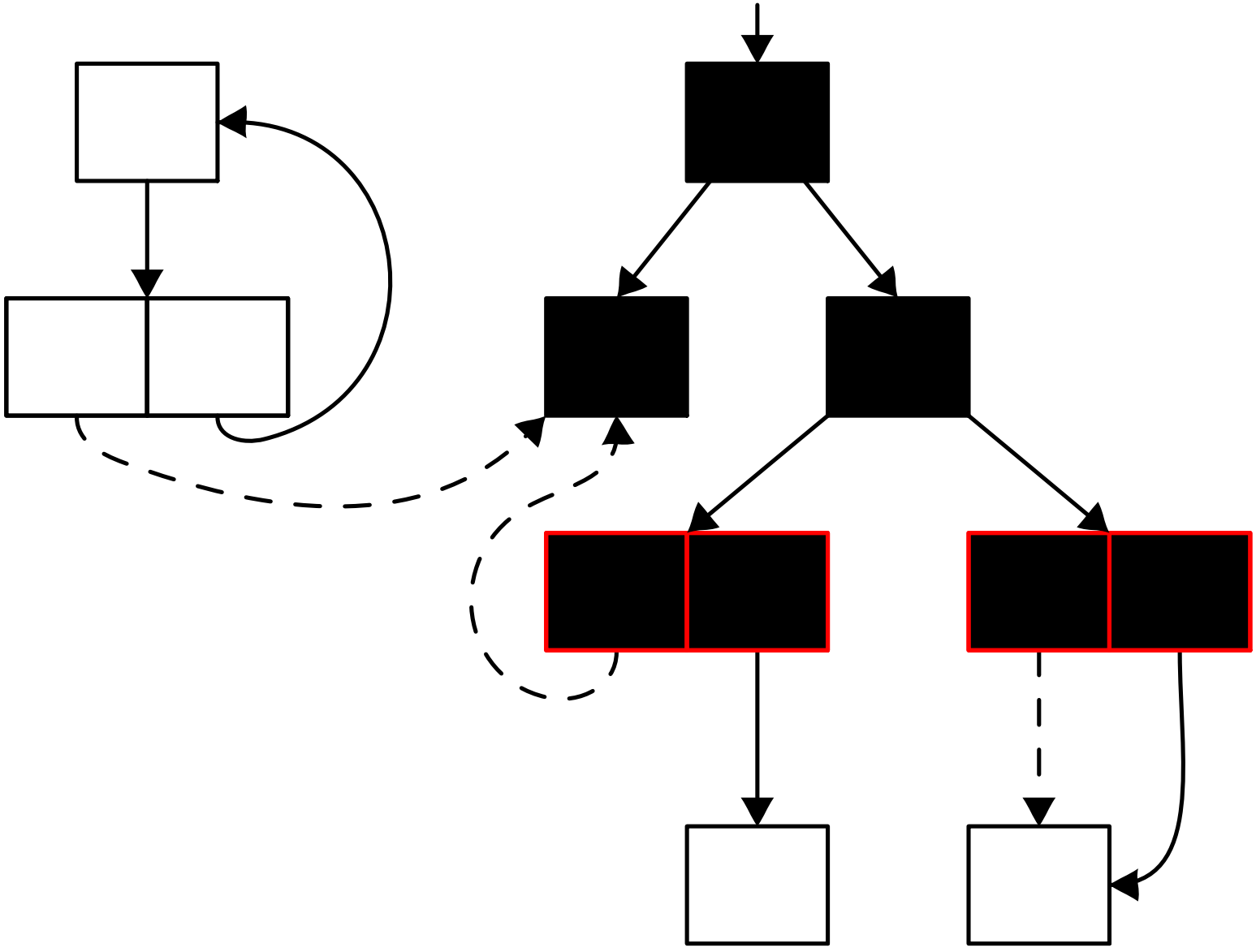
Garbage Collection with Ephemerons



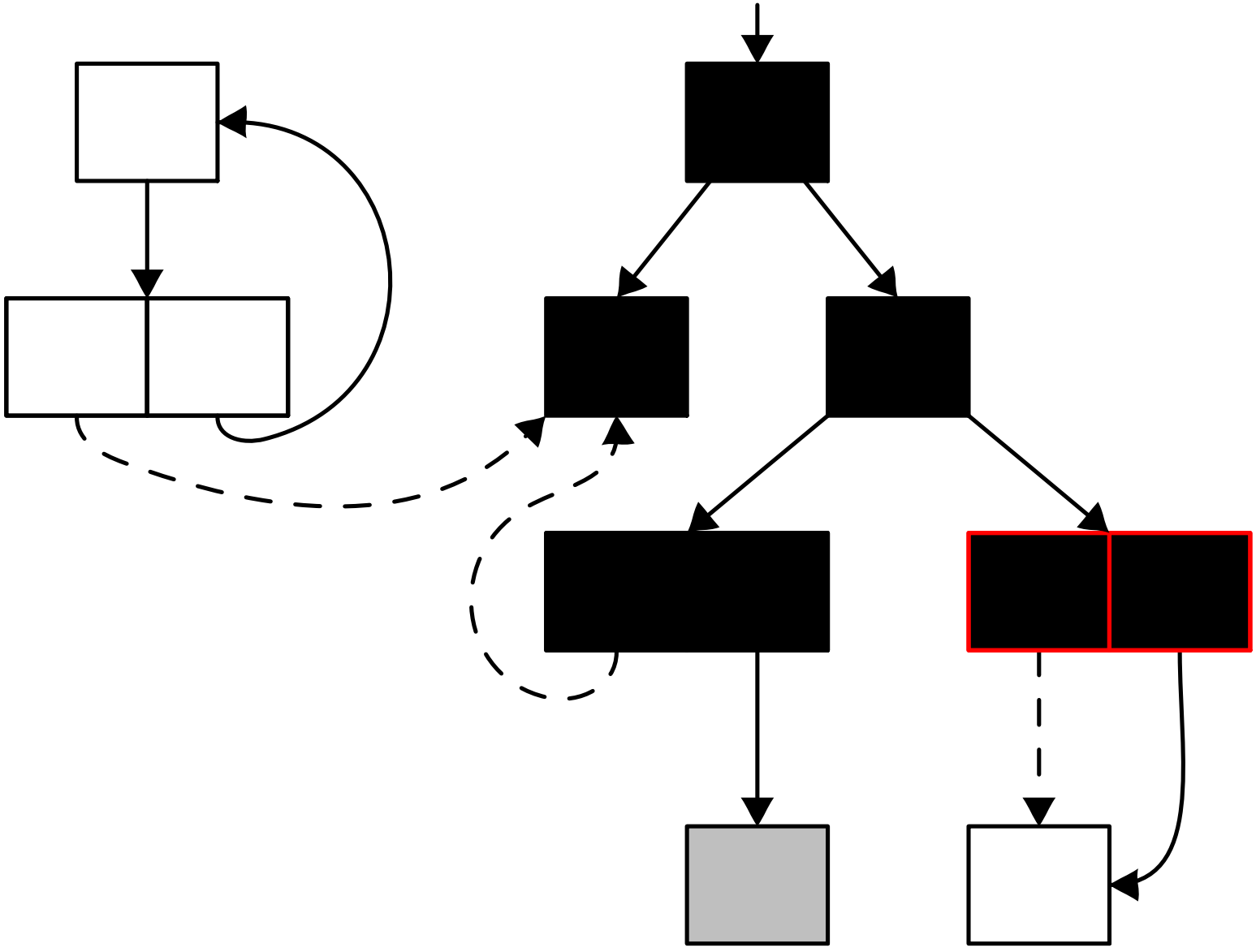
Garbage Collection with Ephemerons



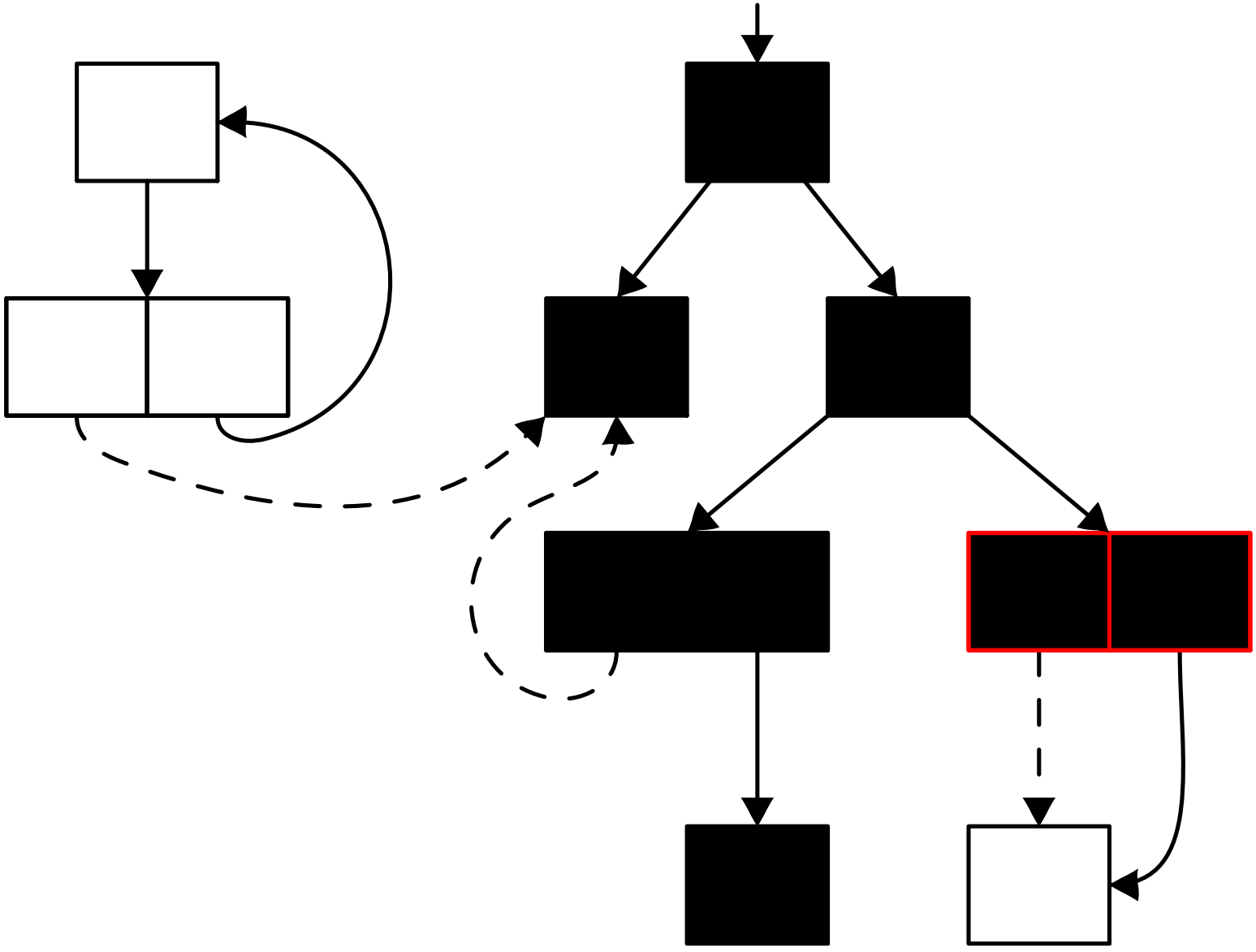
Garbage Collection with Ephemerons



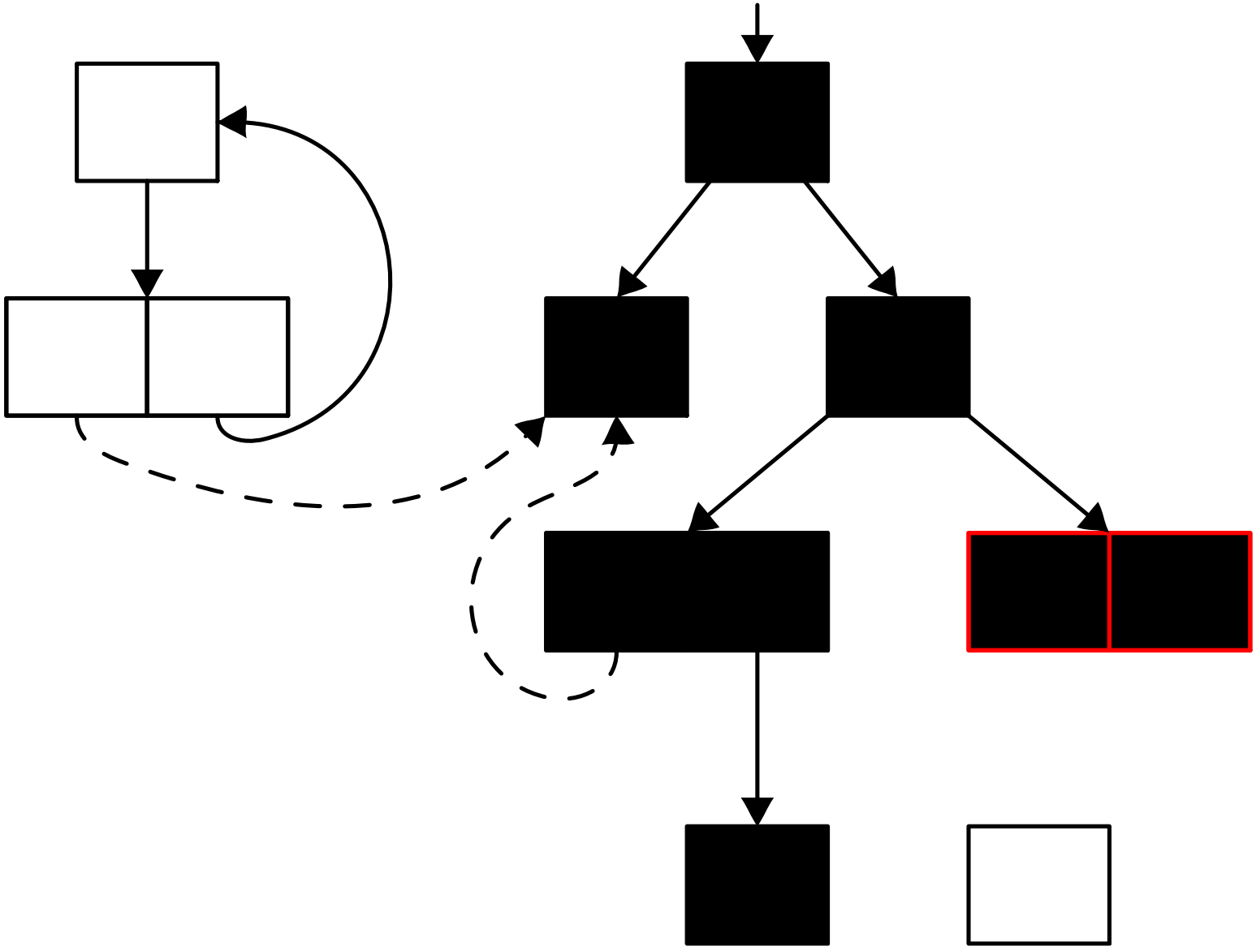
Garbage Collection with Ephemerons



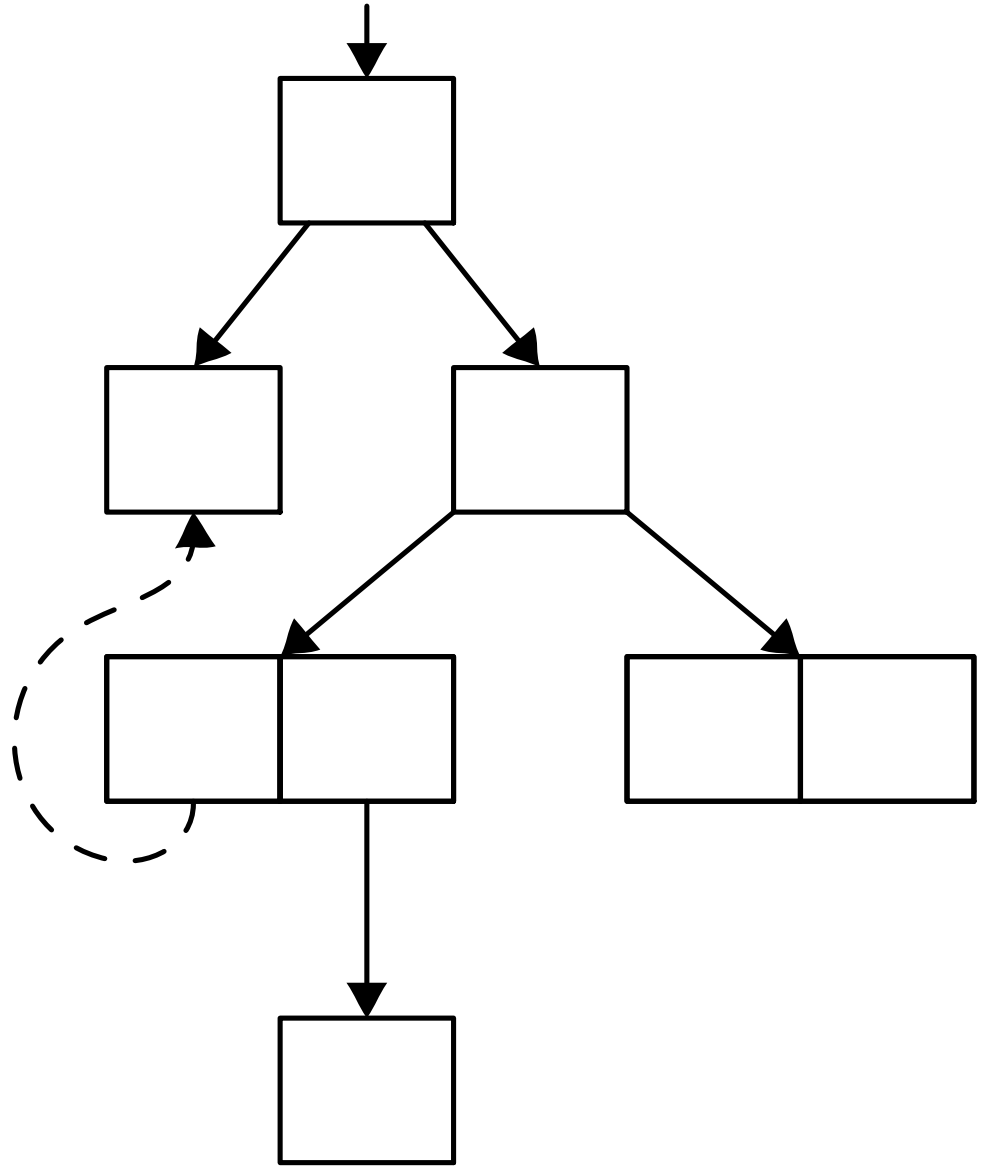
Garbage Collection with Ephemerons



Garbage Collection with Ephemerons



Garbage Collection with Ephemerons



Patched Ocaml Runtime

- use a new tag to represent ephemeron values
- use 4 words

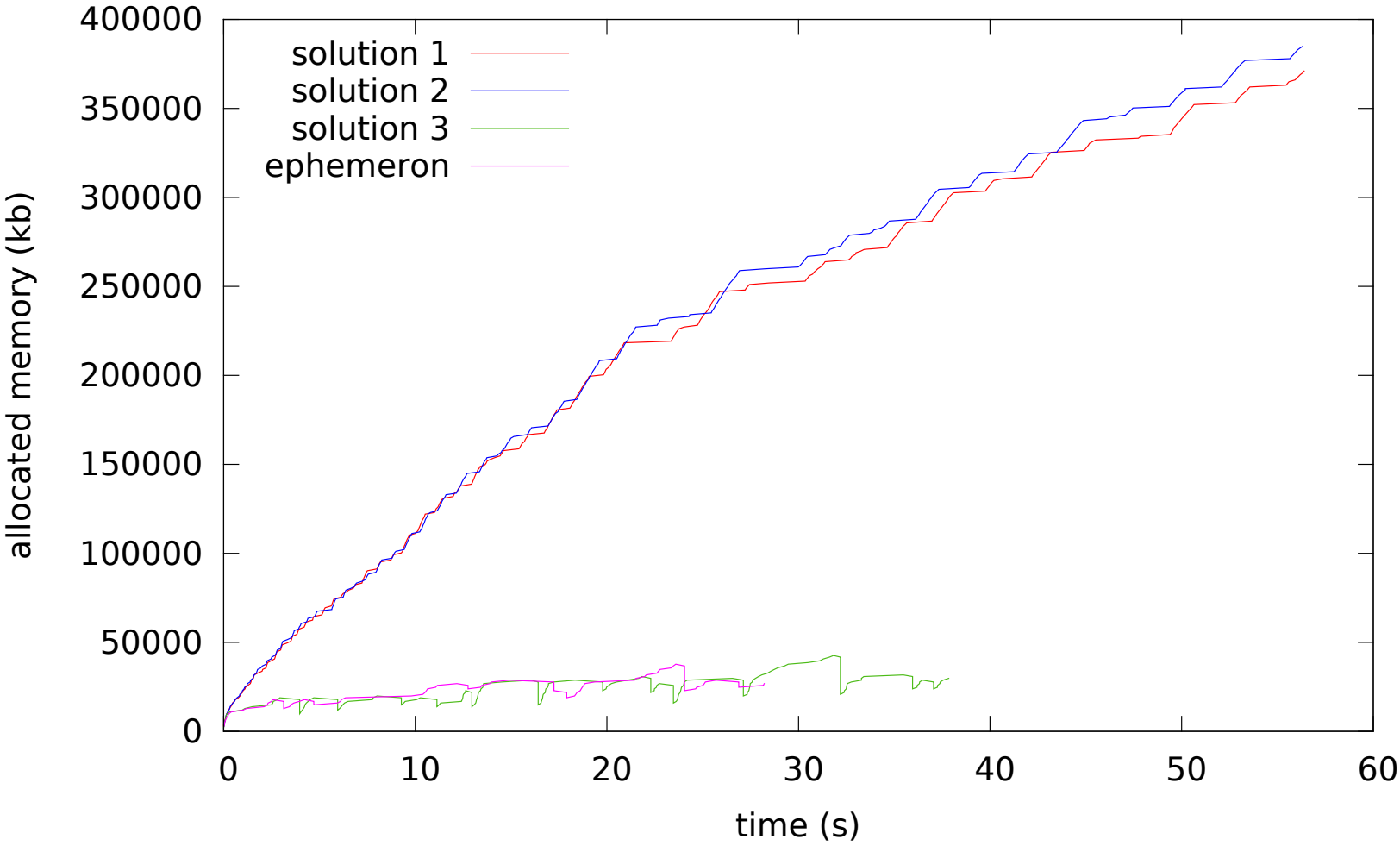
- add $O(n \cdot c)$ operations to mark phase. n number of ephemerons, c the longest chain of ephemerons.
- +84 lines in `major_gc.c` in `mark_slice`
- +70 lines in `weak.c`

Weak Hash Tables with Ephemerons

```
type (α,β) bucketlist =
| Empty
| Cons of (α,β) Ephemeron.t × (α,β) bucketlist
```

Benchmarks

transformation of 1,448 proof tasks sharing subterms



References

- B. Hayes, Ephemerons: a New Finalization Mechanism, OOPSLA'97
- Remi Vanicat 2002, Hweak: the key and the value are weak, so not fit for memoization
- Zheng Li 2007, Weaktbl: doesn't release a key when there is a cycle from value to key