Mark-Compact Garbage Collection

Saarland University, WS 2001/2002
Jens Regenberg, *jens@regenberg.org*

Overview

- motivation
- general properties
- four compaction algorithms
- comparison of these algorithms

Motivation

- Mark-Sweep fragments heap
- fragmentation may cause memory flaw
- idea: compact after marking

Generic compaction Algorithm

- **mark**: the live nodes.
- **compact**: move the live nodes to the beginning of the heap.
- **update**: the pointer values that referred to moved nodes.

Compaction characteristics

- **object size**: is algorithm capable of processing objects of different sizes
- **heap passes**: number of passes through the heap to relocate objects and update pointers
- **space requirements**: does algorithm need additional space
- **styles of compaction**
  - arbitrary
  - sliding

Two-Finger Algorithm

Requirements

- **fixed-sized objects**
- **contiguous heap**

Edwards [Saunders, 1974]

Algorithm

- **mark** and count `no_of_live_nodes`
- **compact** heap
- **update** pointers to addresses $>$ `no_of_live_nodes`.

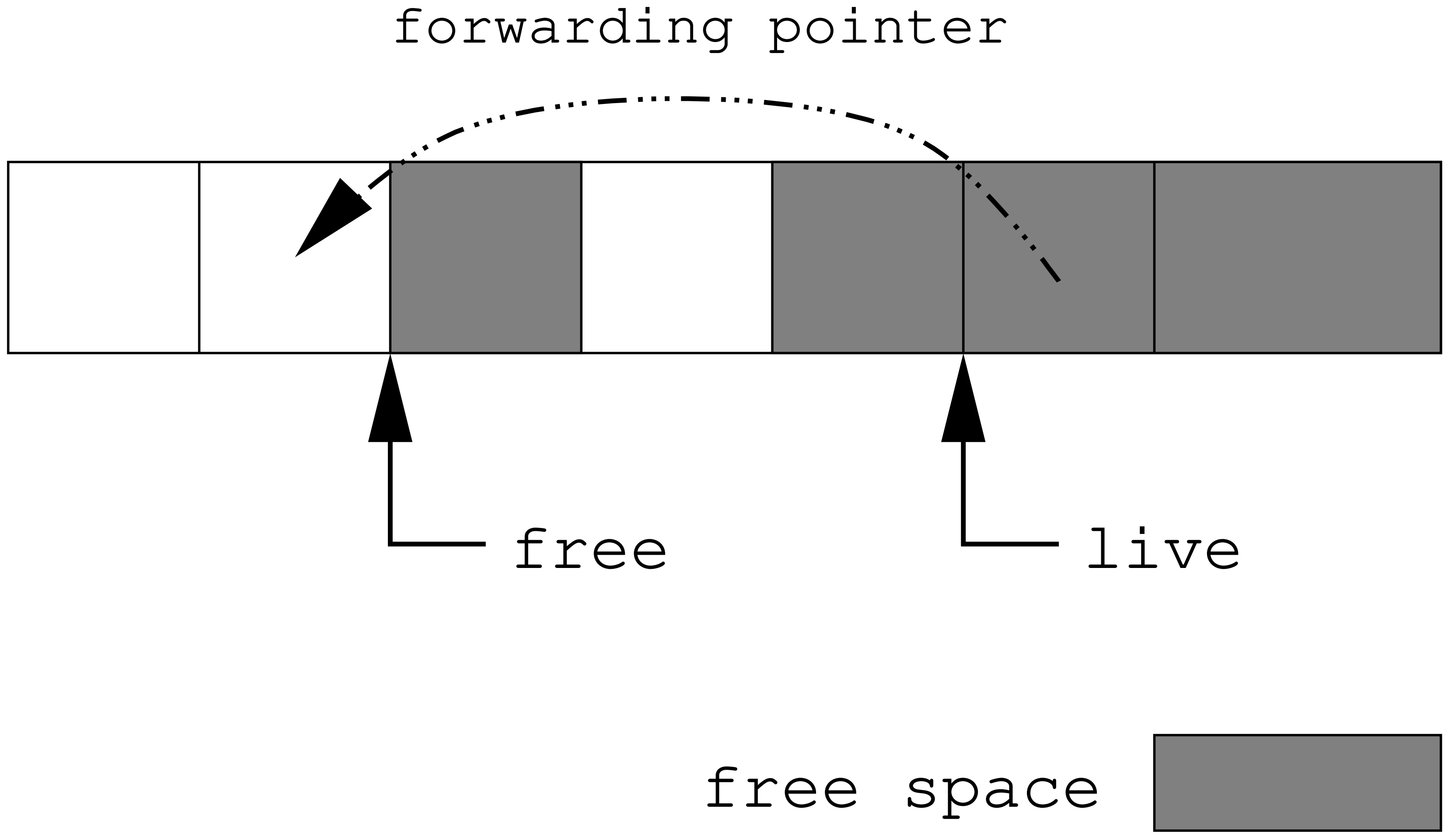
Compaction-Phase

Partitioning:

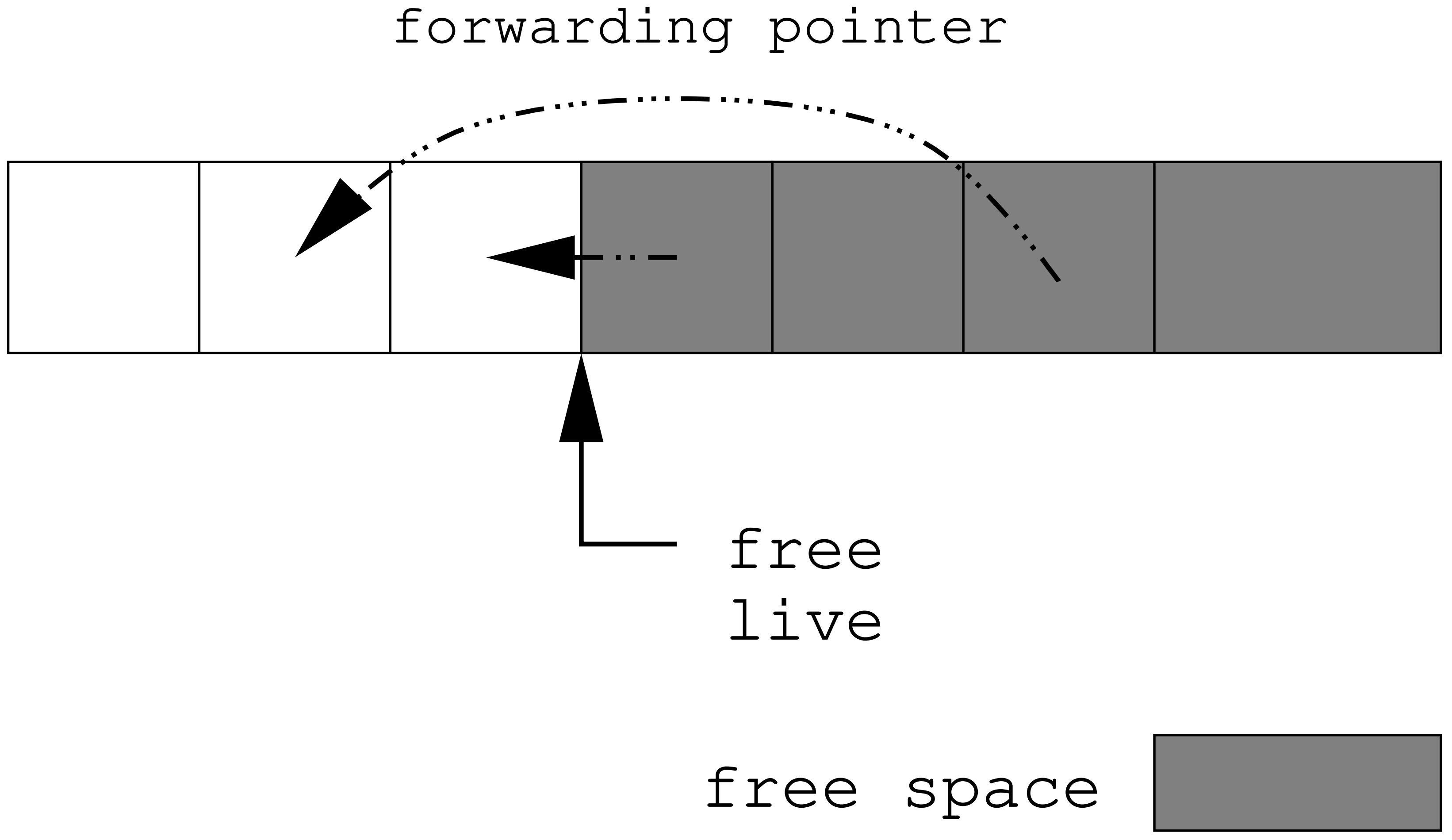
- **Two pointers (finger)**
  - `free` scans heap from bottom to top. Initialy points to the first free node
  - `live` scans heap from top to bottom
- **compacting**: whenever `live` encounters a live node it is copied to `free`
- **terminates**: when `free` and `live` meet

Compact-Phase

Compact-Phase



Compact-Phase



Analysis

- **Advantages**
  - $O(|Heap|)$ complexity
  - no additional space needed
- **Drawbacks**
  - works only on fixed-sized objects
  - locality
- **Variations**
  - variable object size

Lisp2 Algorithm

Requirements

- **Additional pointer** `new_address` in each object header
- used for marking nodes
- if `new_address` = *nil*: node is garbage

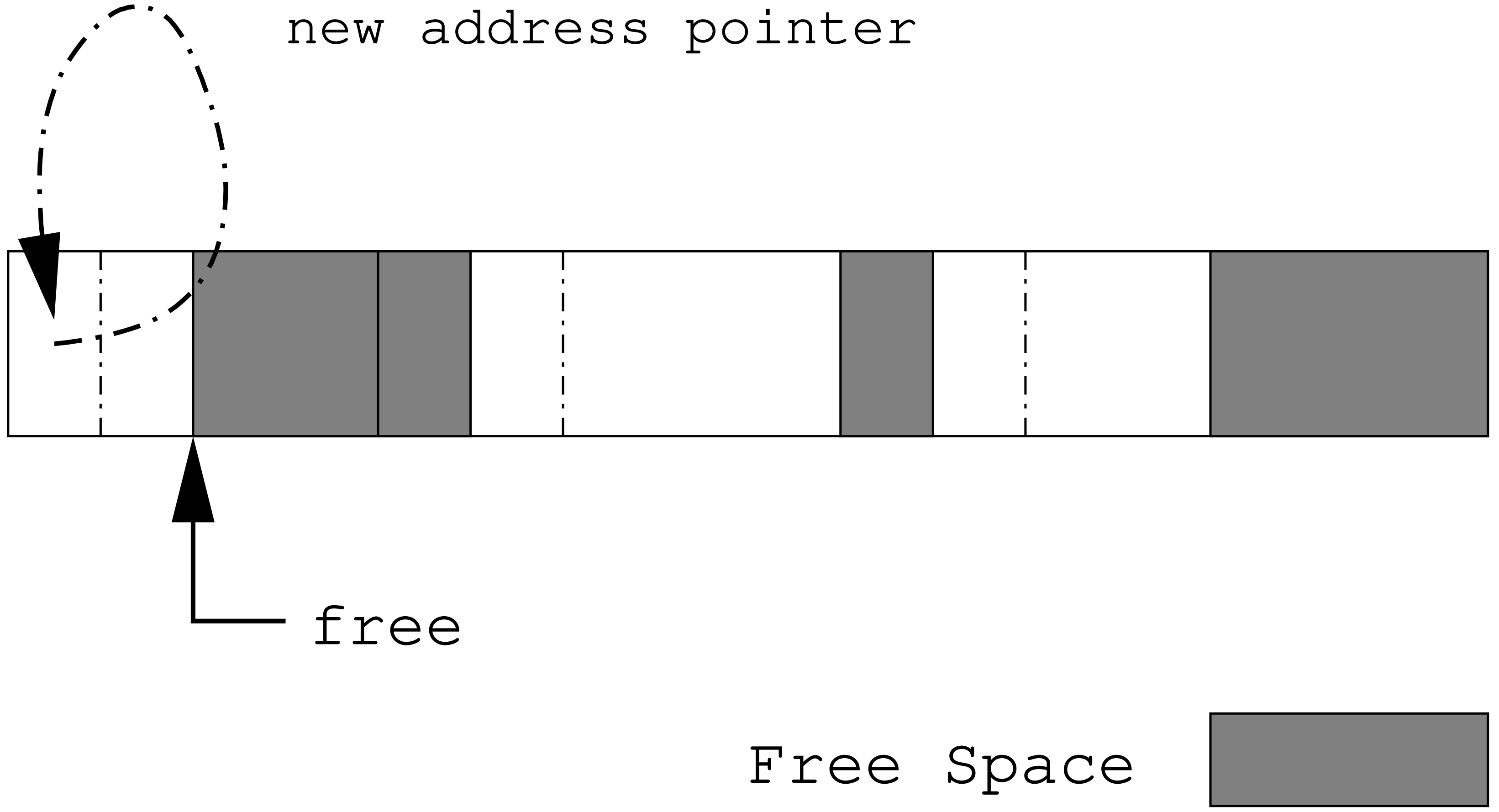
[Cohen and Nicolau, 1983]

Algorithm

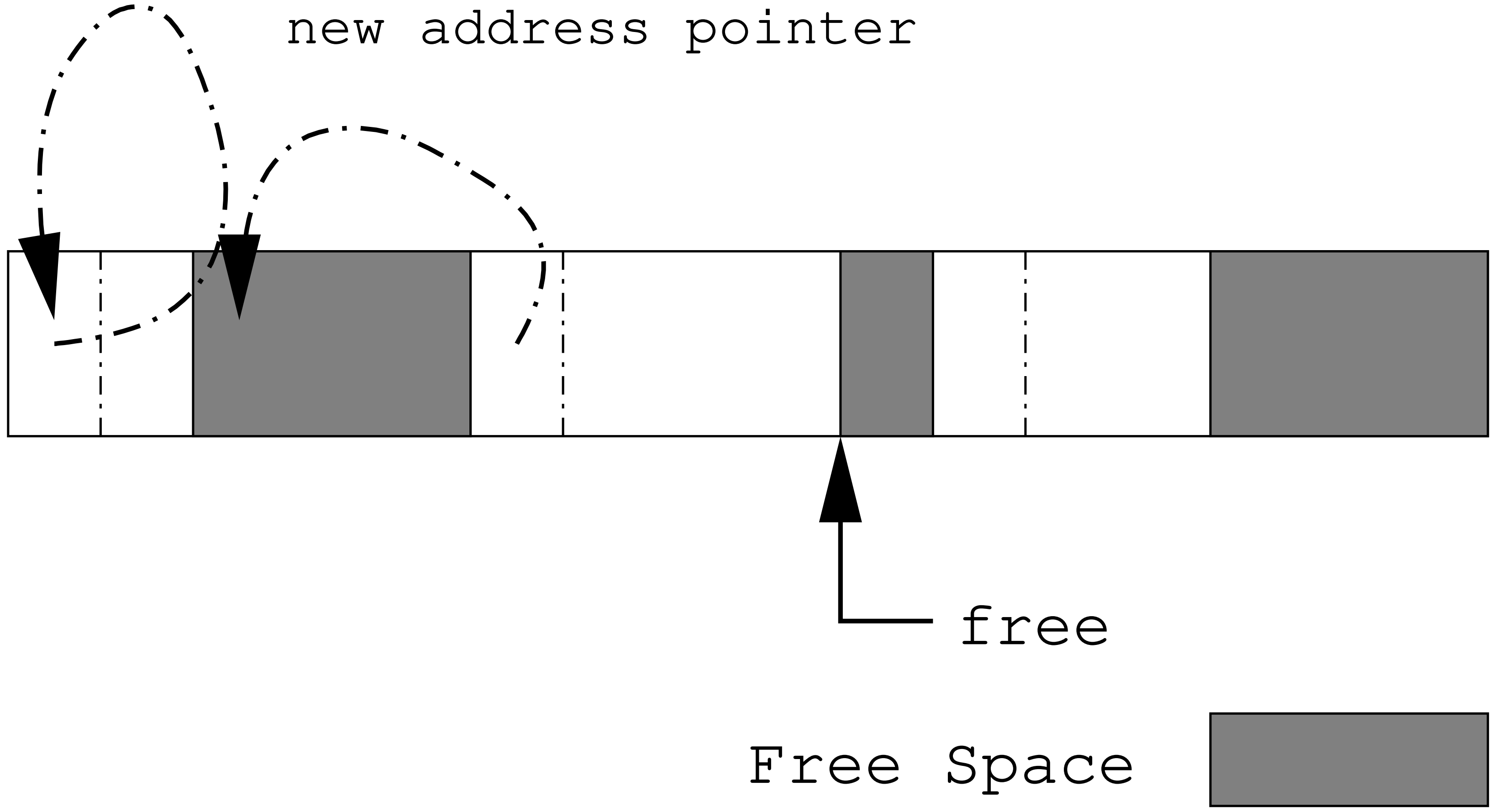
- **mark**: set `new_adress` $\neq$ nil
- **compute_address**: `new_address` = *new_address*
- **update_pointers**: to the new loactions
- **relocate**: compact the heap

Compute_address

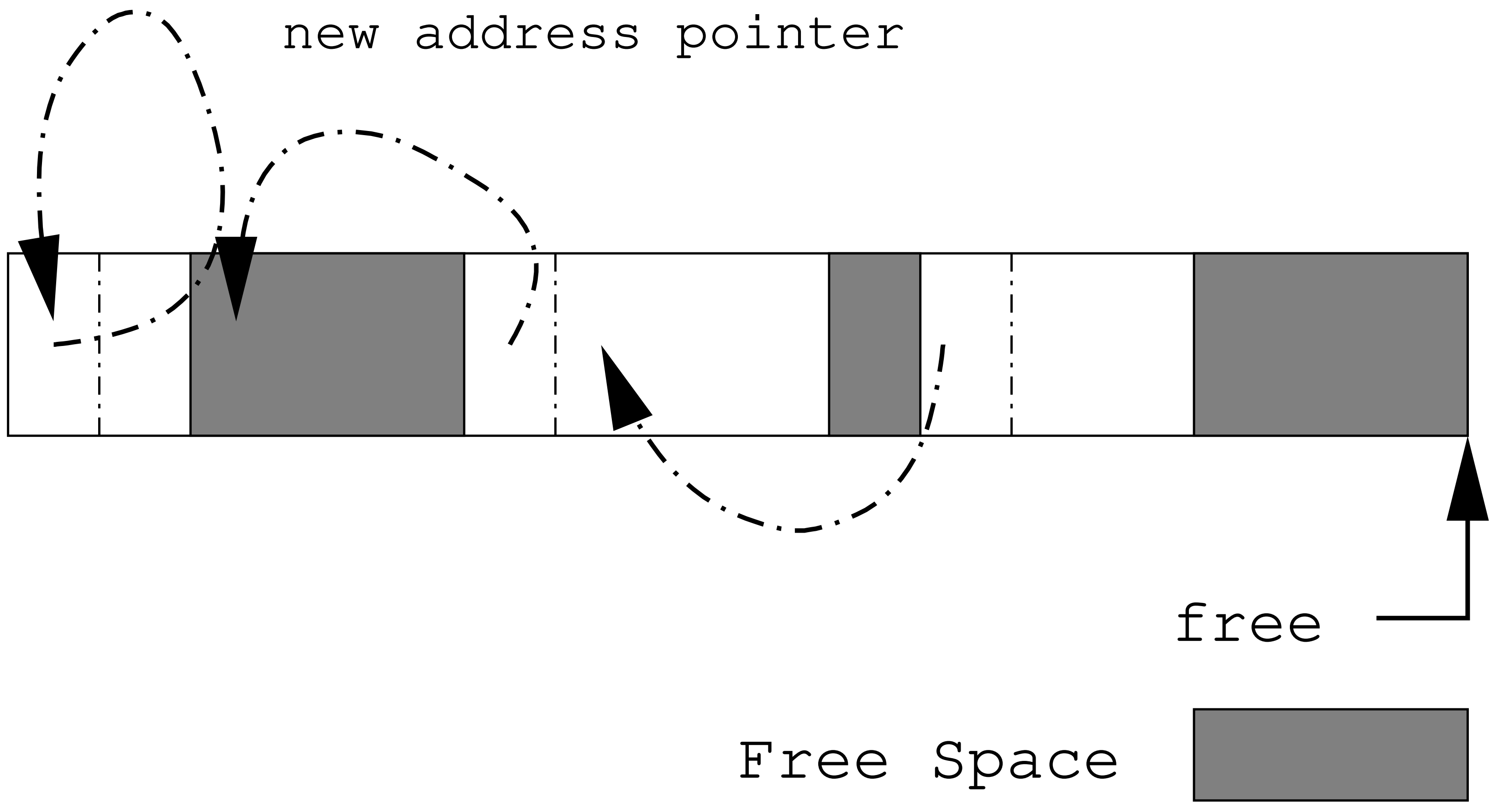
Compute_address



Compute_address



Compute_address



Analysis

- **Advantages**
  - ◇ maintains locality
  - ◇ $O(|Heap|)$ complexity
- **Drawbacks**
  - ◇ additional pointer needed
  - ◇ three heap passes

Haddon - Waite Algorithm

Requirements

- minimum size of an object is 2 words.

[Haddon and Waite, 1967]

Algorithm

- **mark**
- **relocate**
  - compact live nodes
  - create table
- **update**

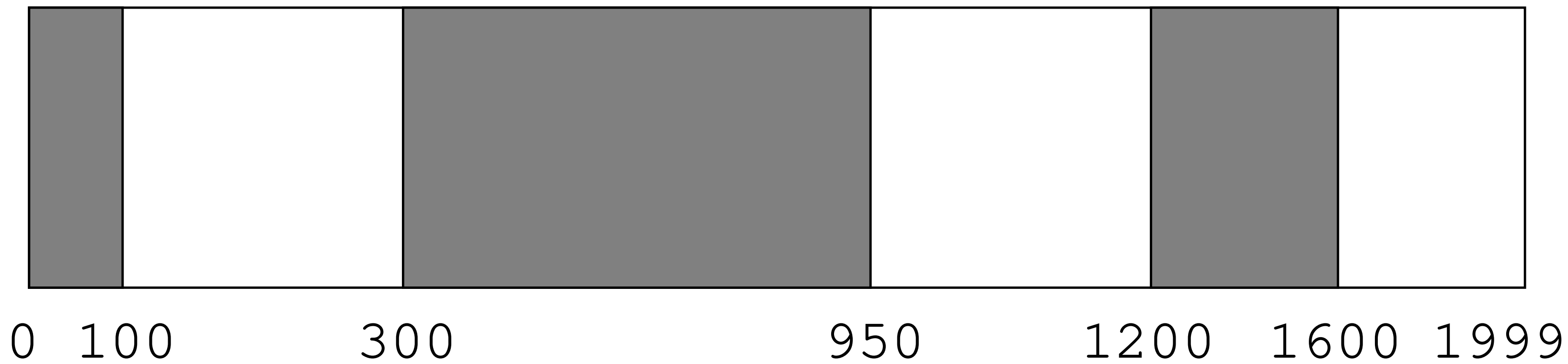
Relocation-Phase

- scan heap from bottom to top
- for every encountered live node
  - ◇ move node to the first free node ($\rightarrow$bottom)
  - ◇ create a table entry (a, s)

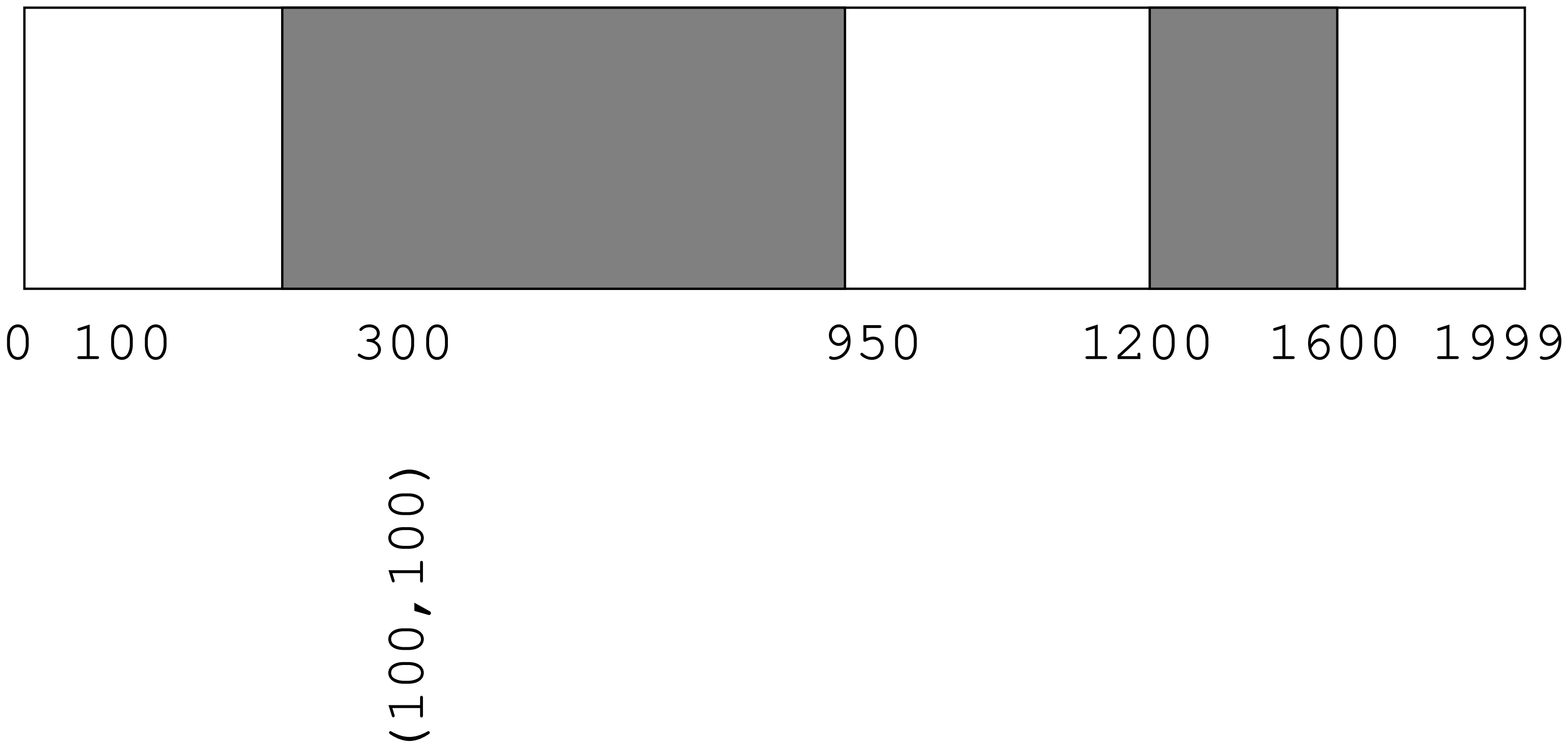
  a - old address

  s - free space encountered so far

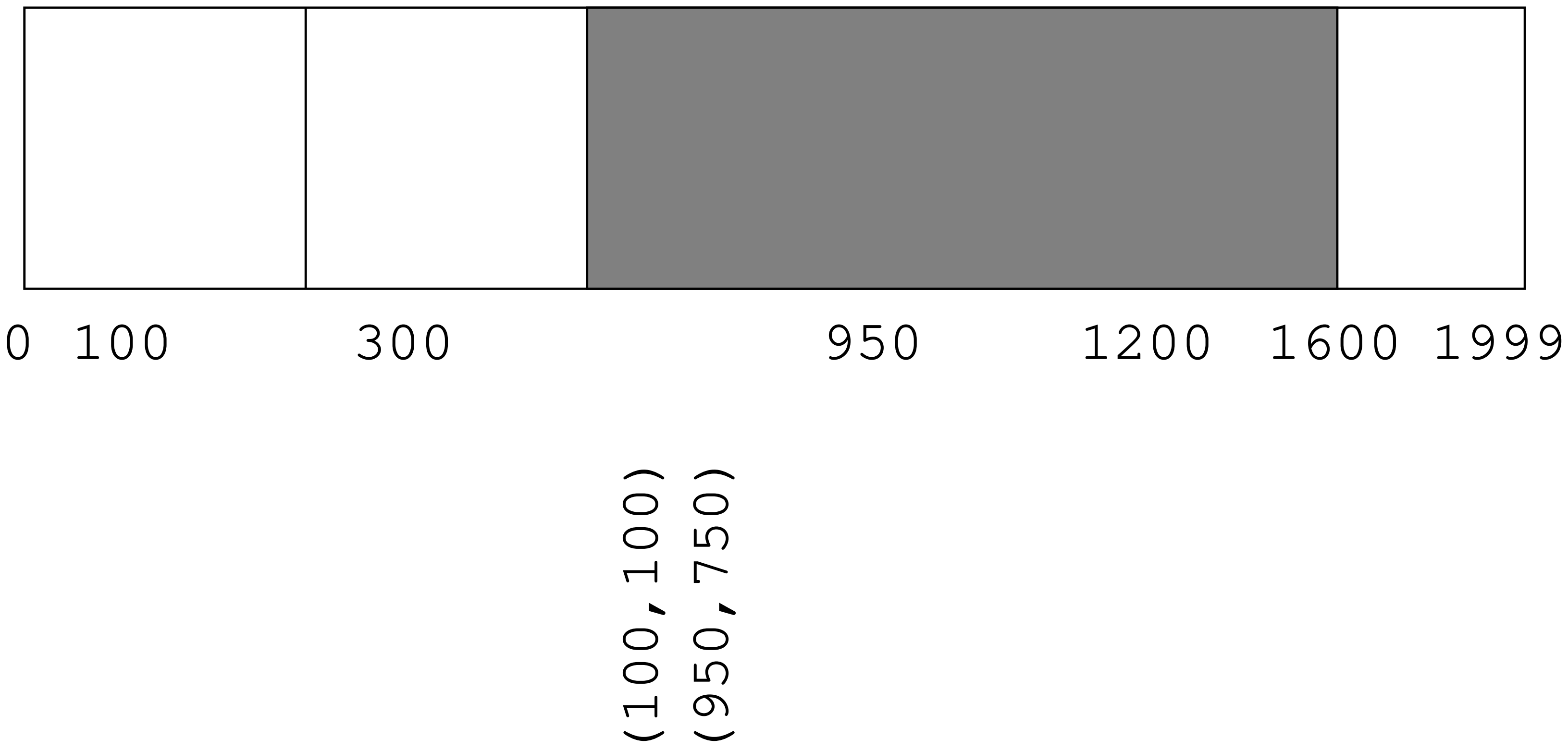
Relocation-Phase



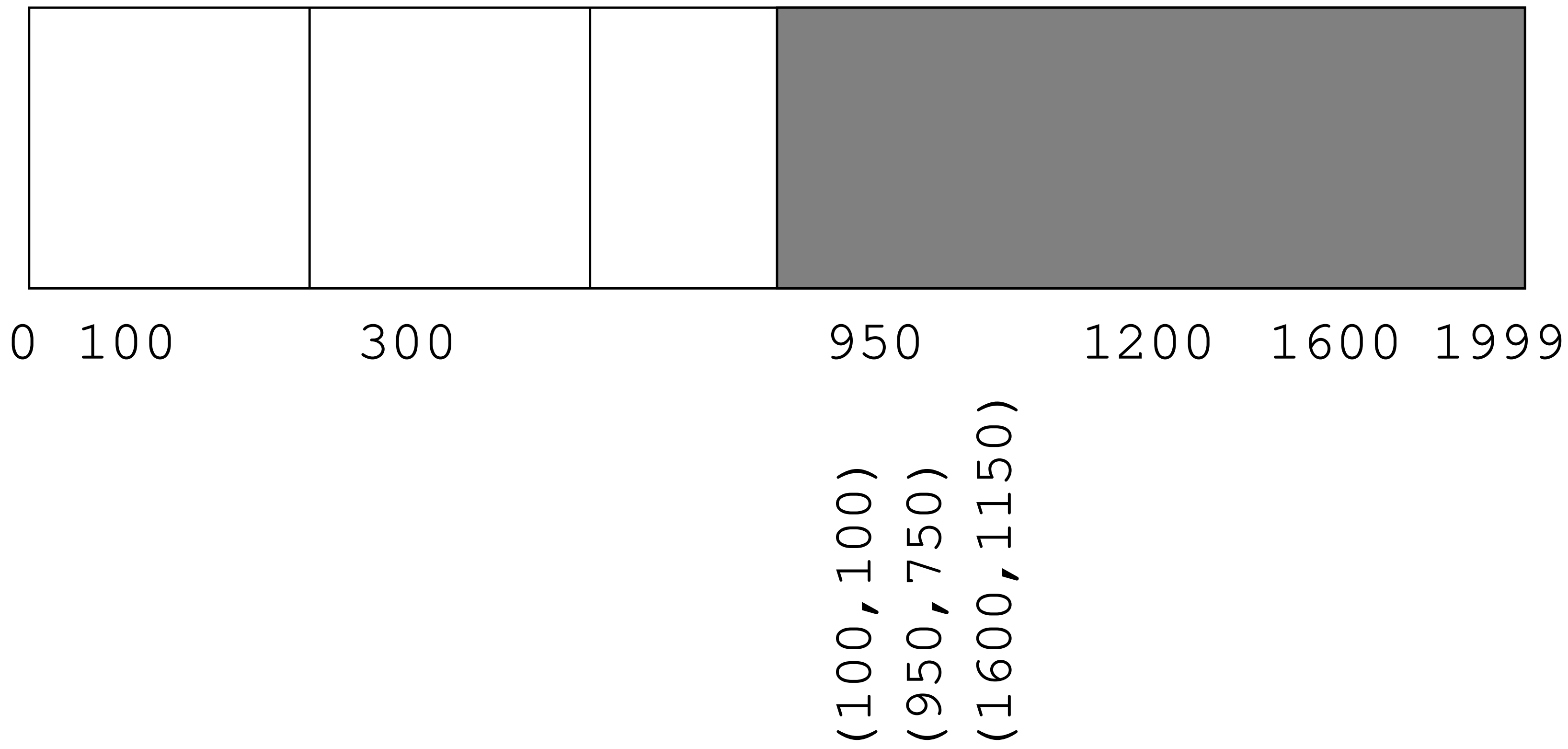
Relocation-Phase



Relocation-Phase



Relocation-Phase



Update-Phase

- for each pointer p
  - find (a, s) , (a', s') such that $a \leq p < a'$
  - $p = p - s$

Analysis

- **Advantages**
  - maintains locality
  - no additional space required
- **Drawbacks**
  - $O(|Heap| log(|Heap|))$ complexity
- **Variations**
  - Hashtables

Threaded Methods

threading - all pointers refering to P can be found from P.

Requirements

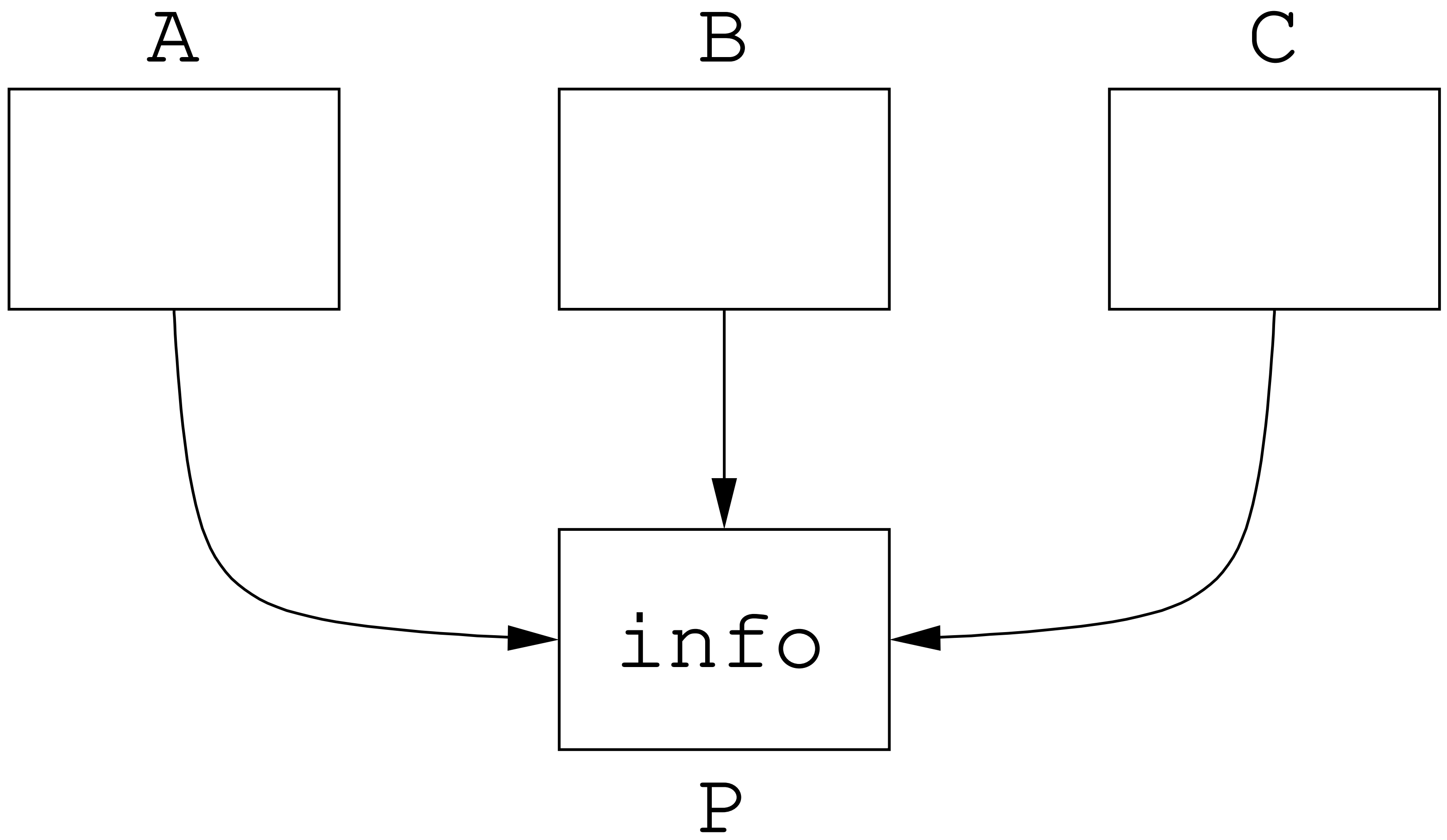
- pointer may only refer to the header of a node.
- header must be large enough to contain a pointer.
- data must be distinguishable from heap pointers.

[Fisher, 1974] [Jonkers, 1979]

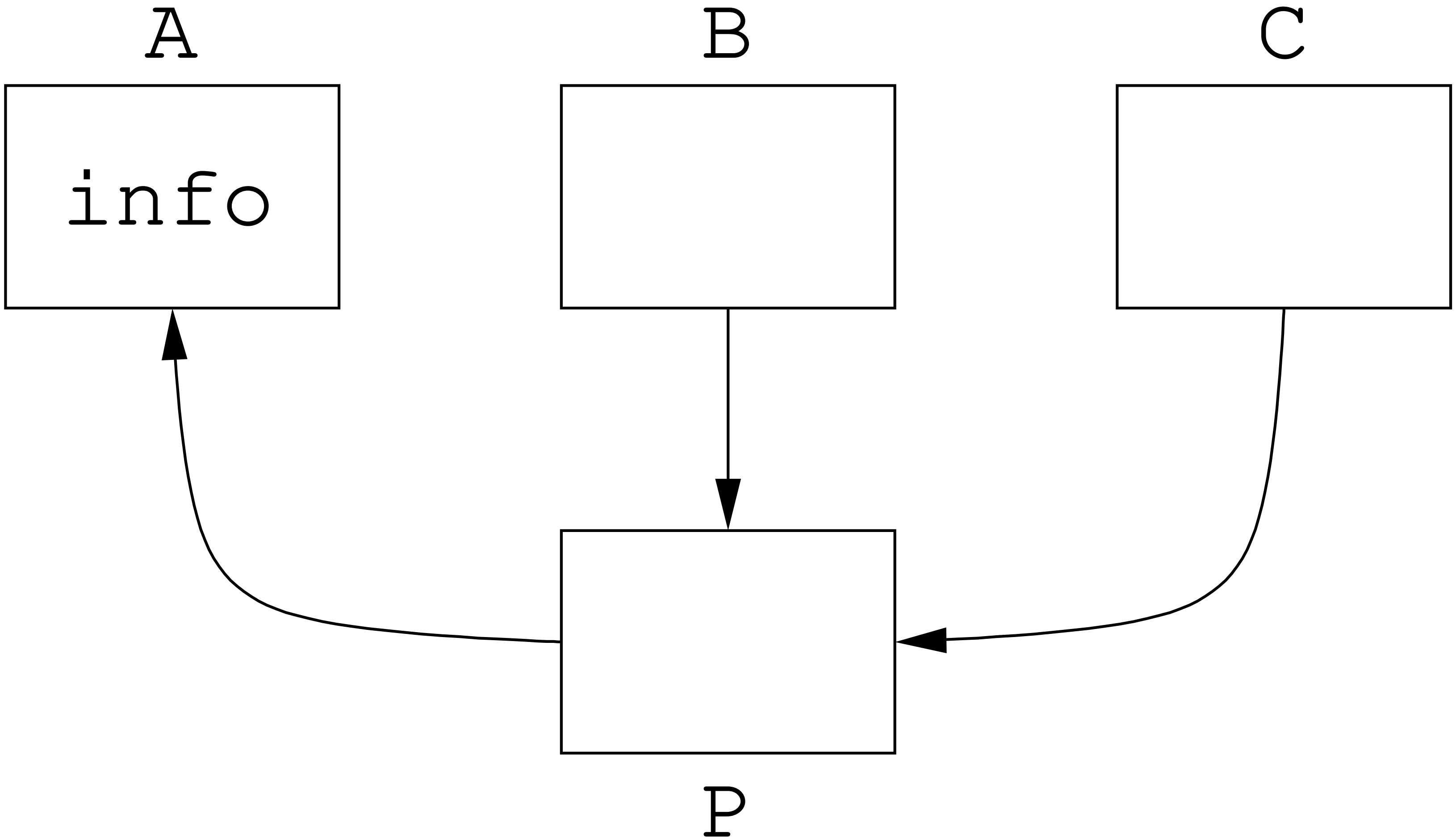
Algorithm

- **mark**
- **update_forward_pointers**
- **update_backward_pointers**

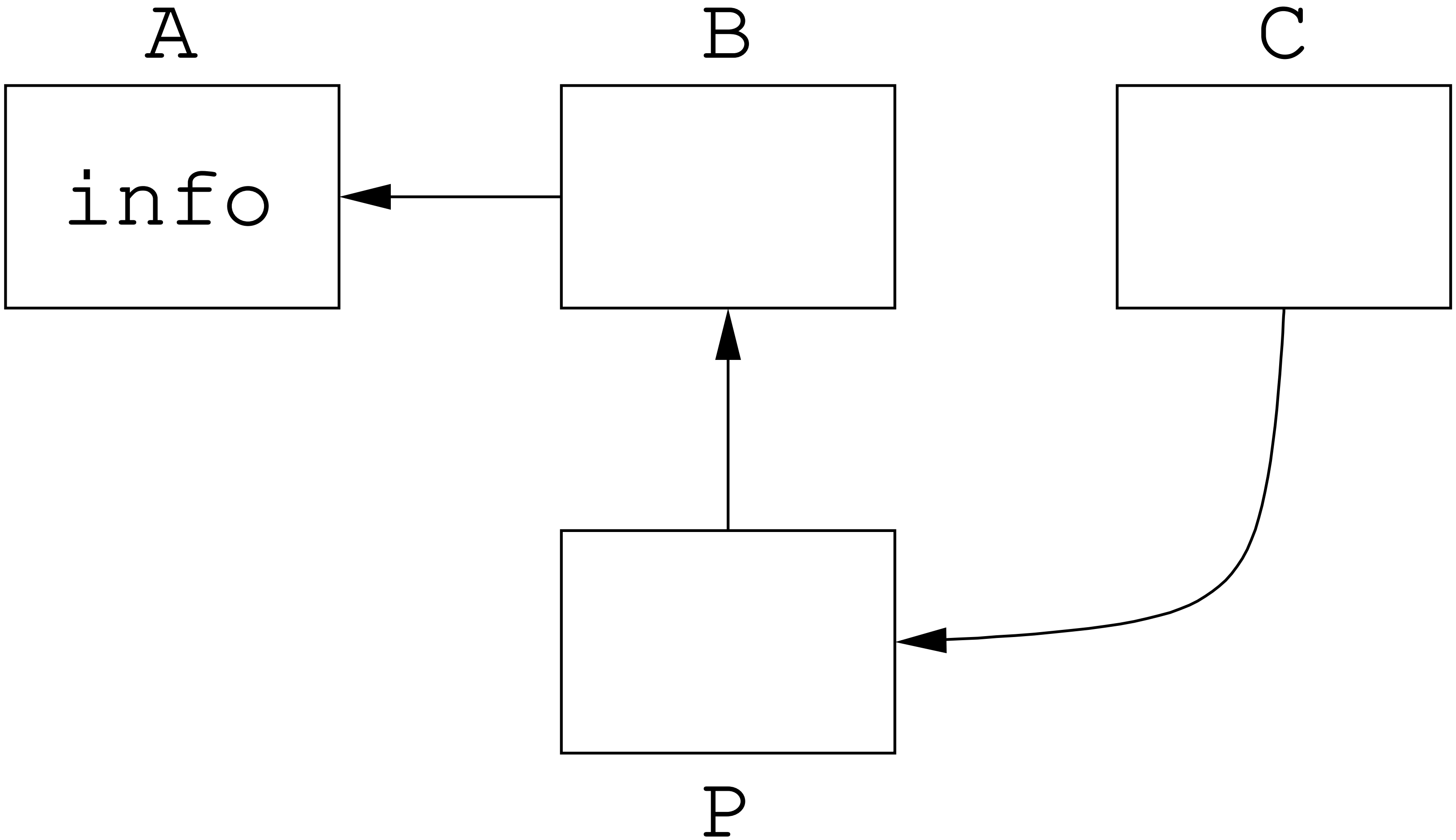
Threading



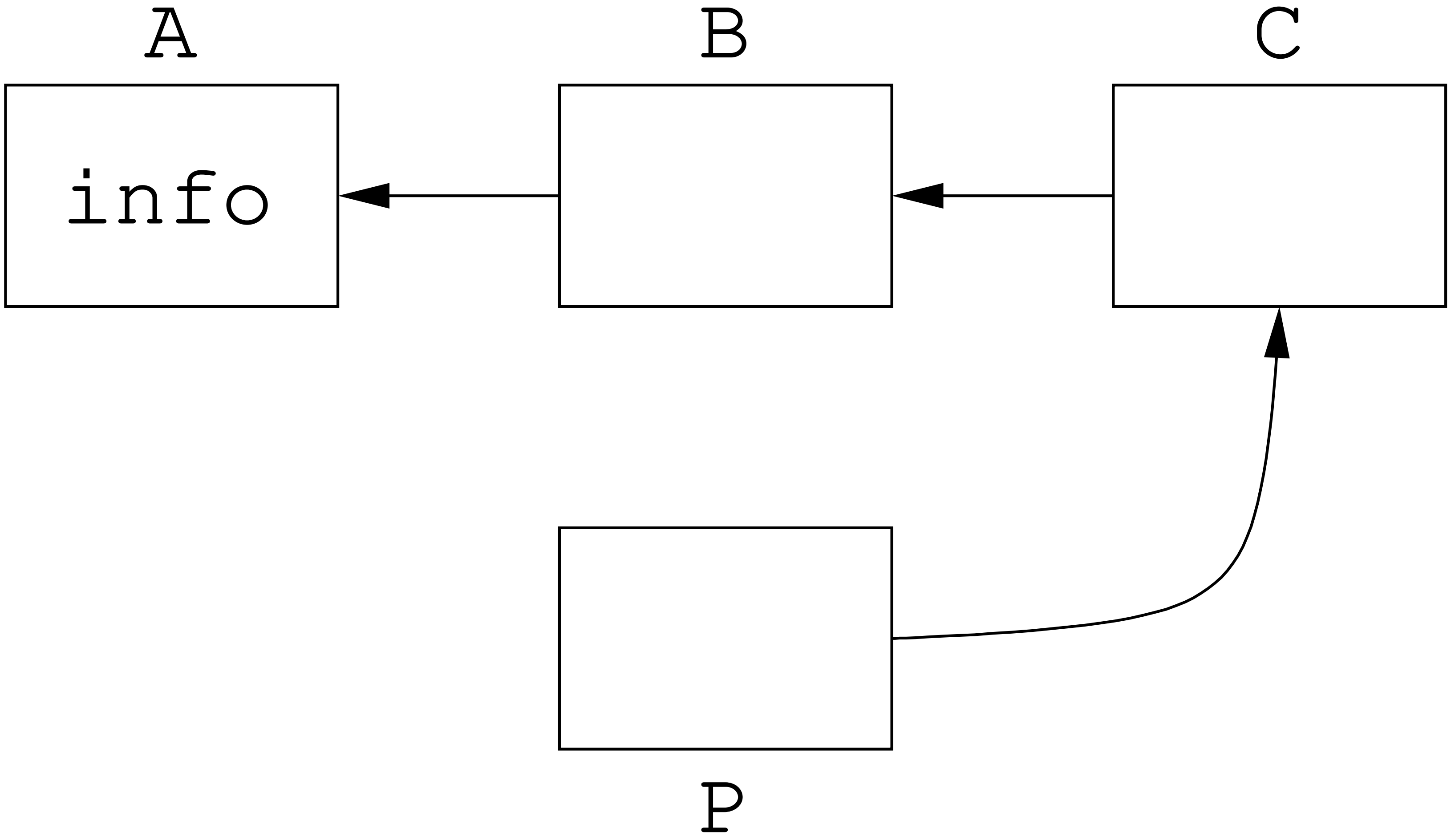
Threading



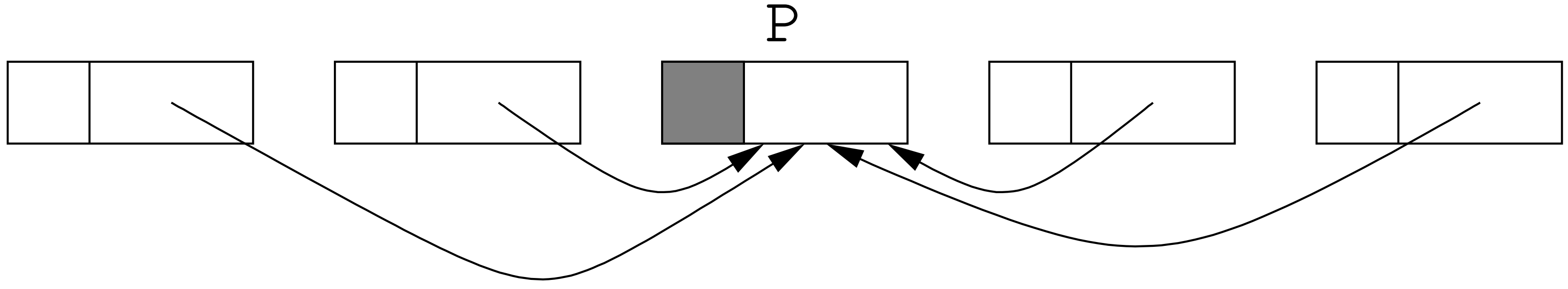
Threading



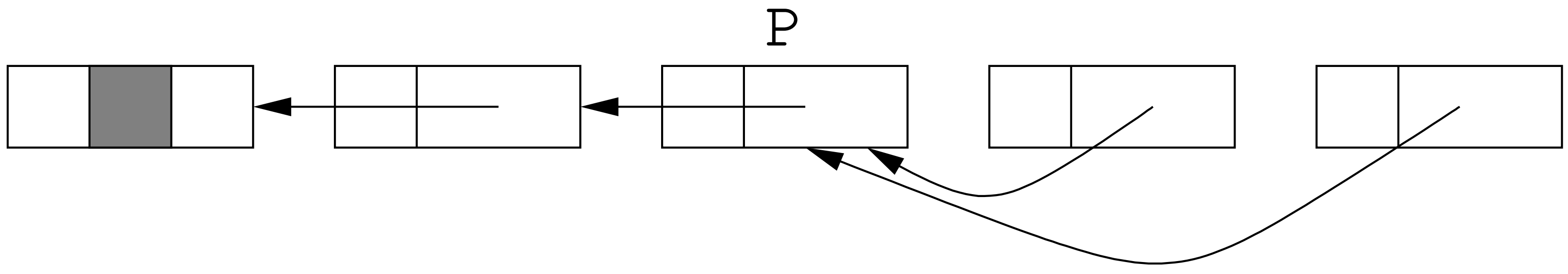
Threading



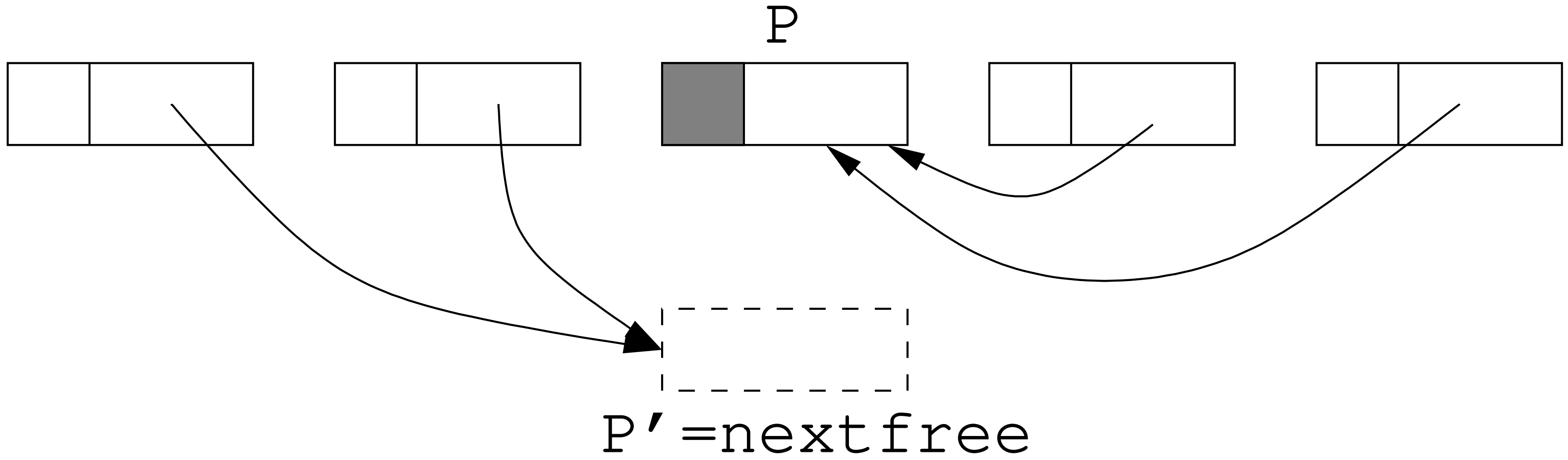
update_forward_pointers



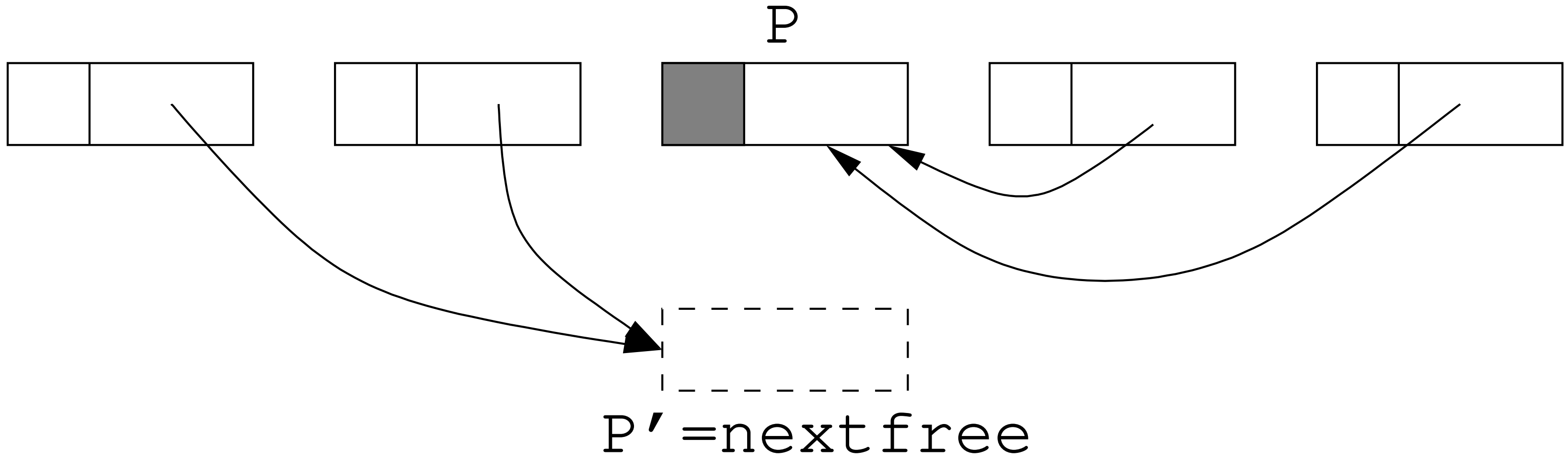
update_forward_pointers



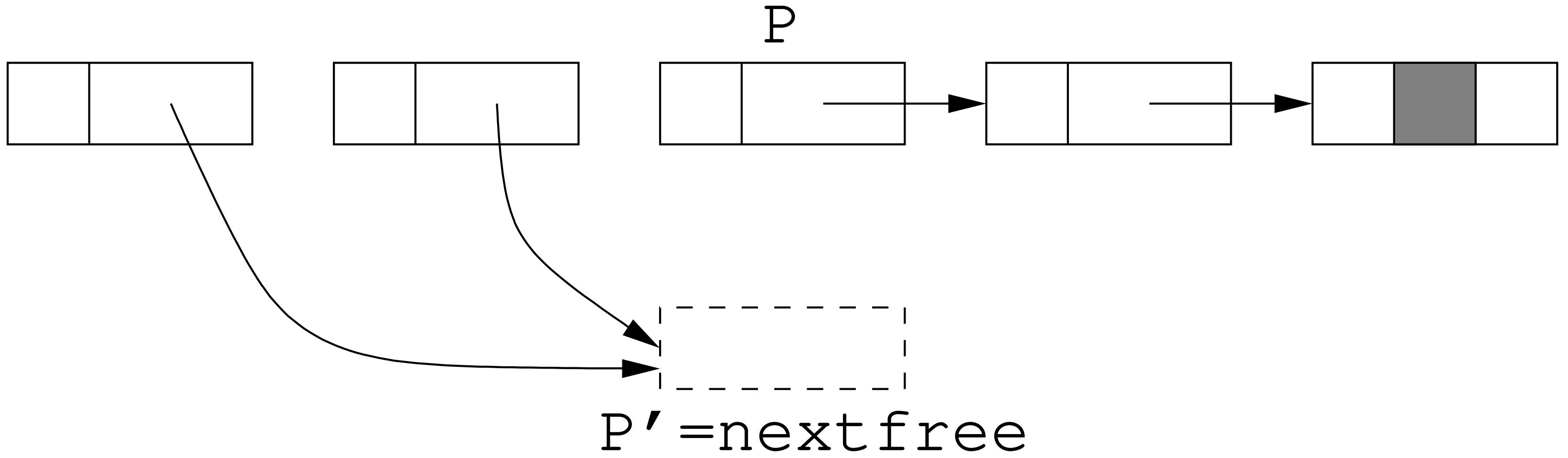
update_forward_pointers



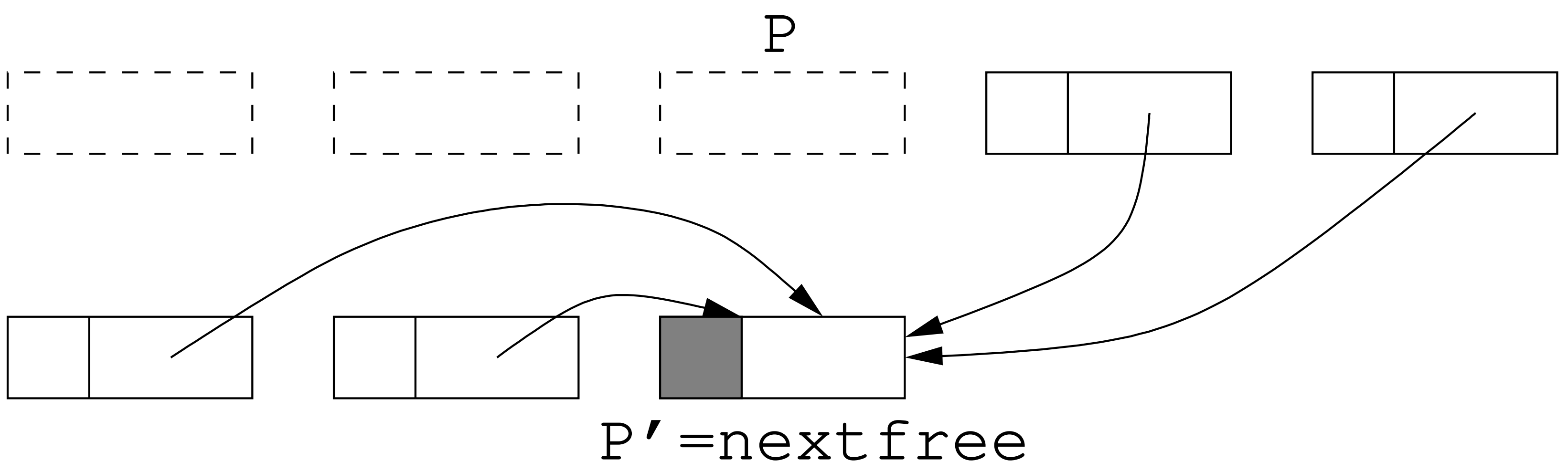
update_backward_pointers



update_backward_pointers



update_backward_pointers



Analysis

- **Advantages**
  - no additional space
  - $O(|Heap|)$ complexity
  - only 2 passes through the heap
- **Drawbacks**
  - much overhead at each iteration

Algorithm Overview

Algorithm	Style	Cell size	Passes	Space	Time
Two-Finger	arbitrary	fixed	2	none	M
Lisp2	sliding	variable	3	pointer sized field per cell	M
Table-based	sliding	variable	2	none	Mlog(M)
Threaded	sliding	variable	2	headers at least pointer sized	M

M = $|Heap|$

Summary

- compaction is expensive, but reduces allocation costs.
- repeated copying unlikely
- locality can be preserved
- smaller address space (copying collector)
- performance improvement
  - don’t compact at every collection

Thank You