Generational Garbage Collection

Mirko Jerrentrup, *Jerrentrup@interactive-software.de*

Overview

- motivation
- at a glance
- issues
- problems and limitations
- conclusion

Motivation

„classical“, e.g. copying GC:

- no improvement of locality
- repeated handling of long-lived objects

The weak generational hypothesis

„Most objects die young.“

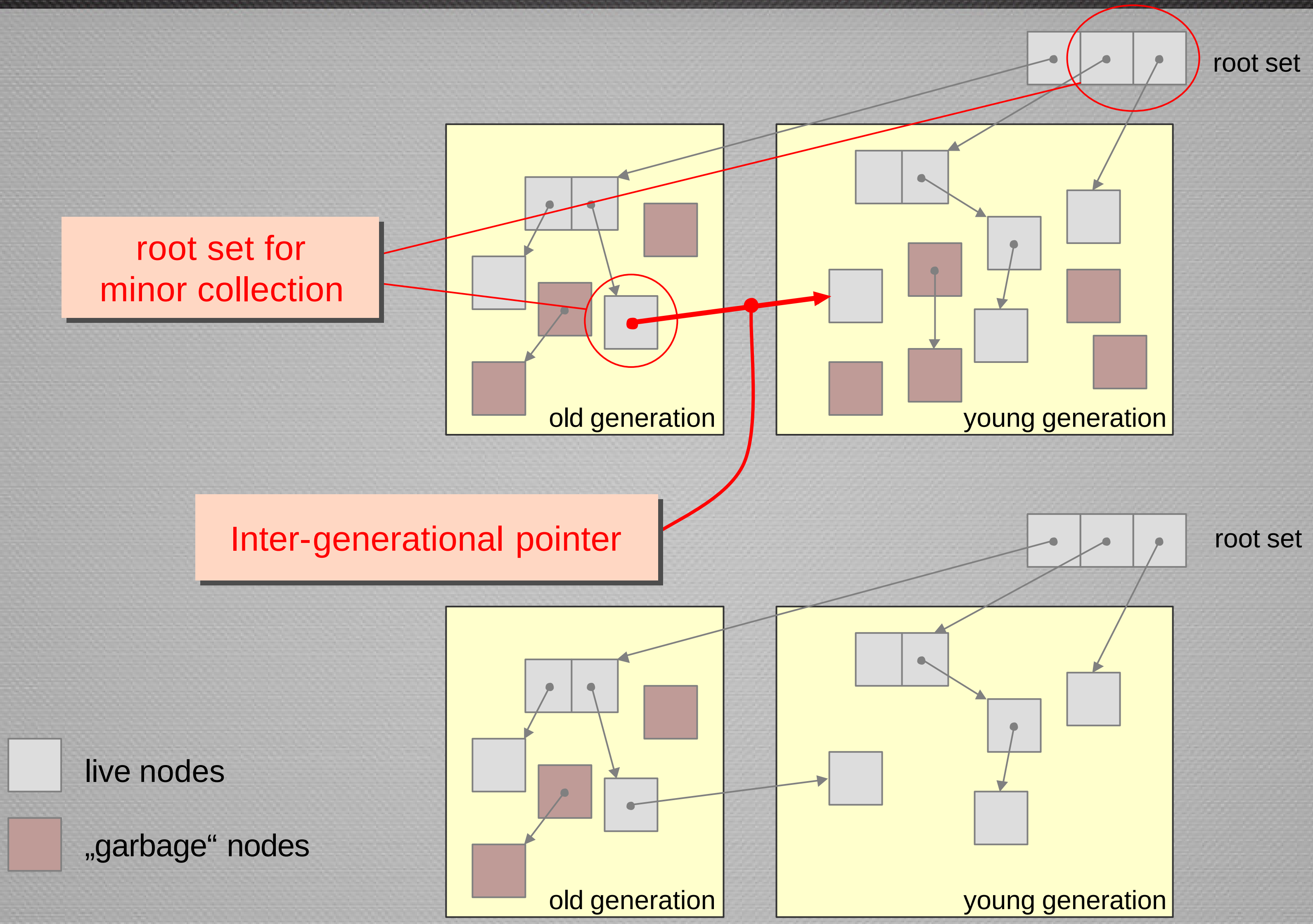
- partition objects into *generations*
- special handling of *young* objects:
  - → reduced pause times
  - → better collection efficiency

[Ungar, 1984]

Generational garbage collection

- objects partitioned into *multiple generations*
- young generation *frequently collected*
  (minor collection)
- old generation *seldomly collected*
  (major collection)
- surviving young objects *promoted* into old generation

Example: minor collection



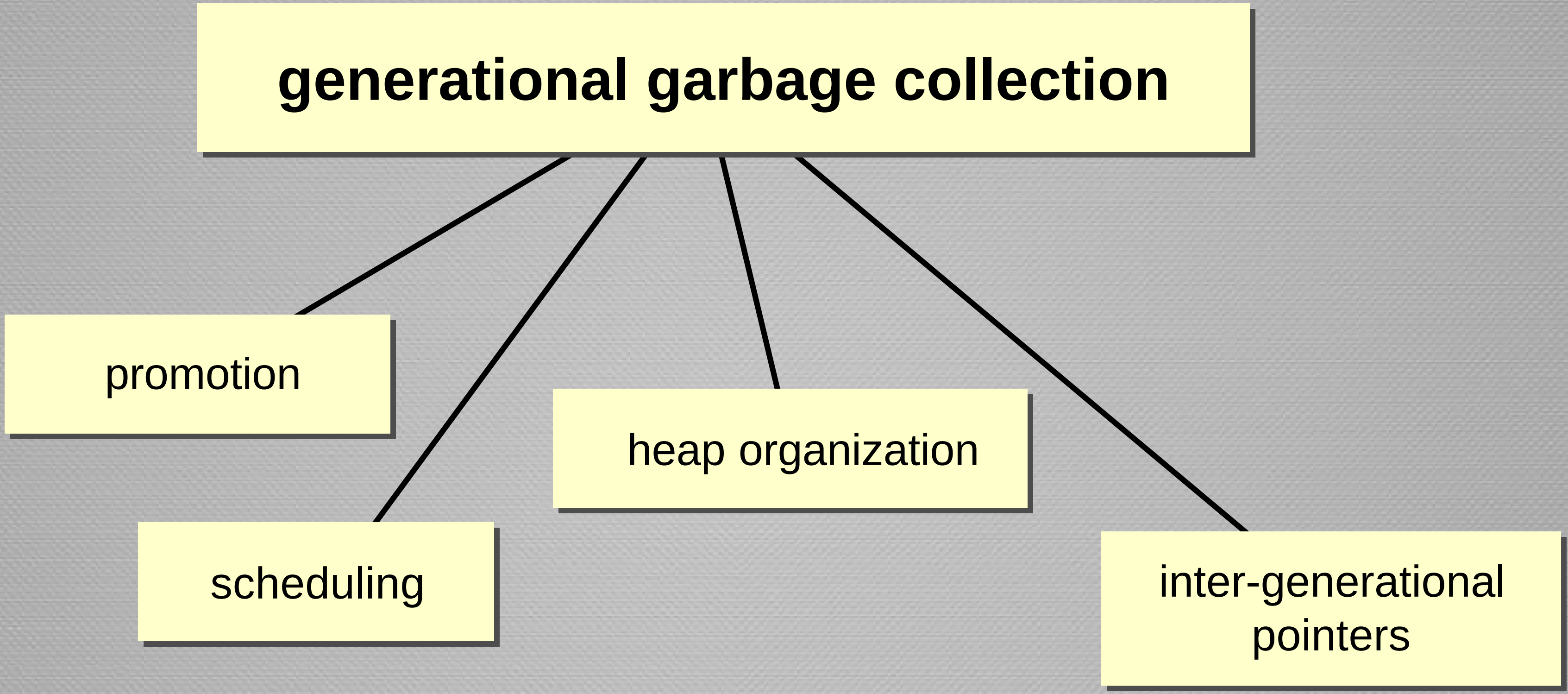
root set for minor collection

Inter-generational pointer

⬜ live nodes

🟫 „garbage“ nodes

Issues in generational garbage collection



generational garbage collection

promotion

scheduling

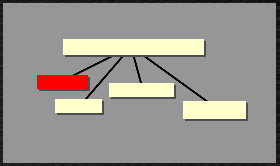
heap organization

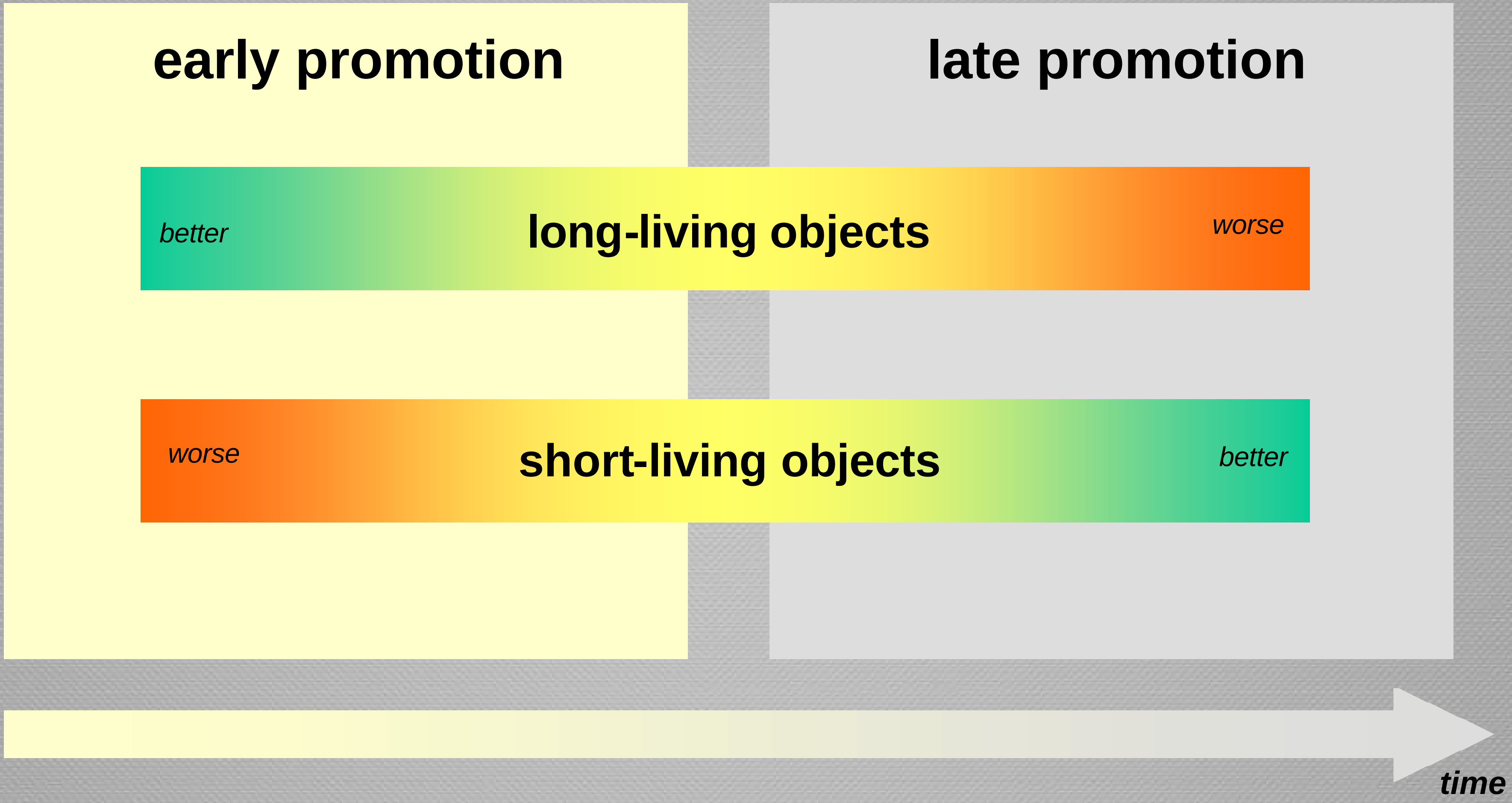
inter-generational pointers

generational garbage collection

- **promotion**
- scheduling
- heap organization
- inter-generational pointers

Promotion





early promotion

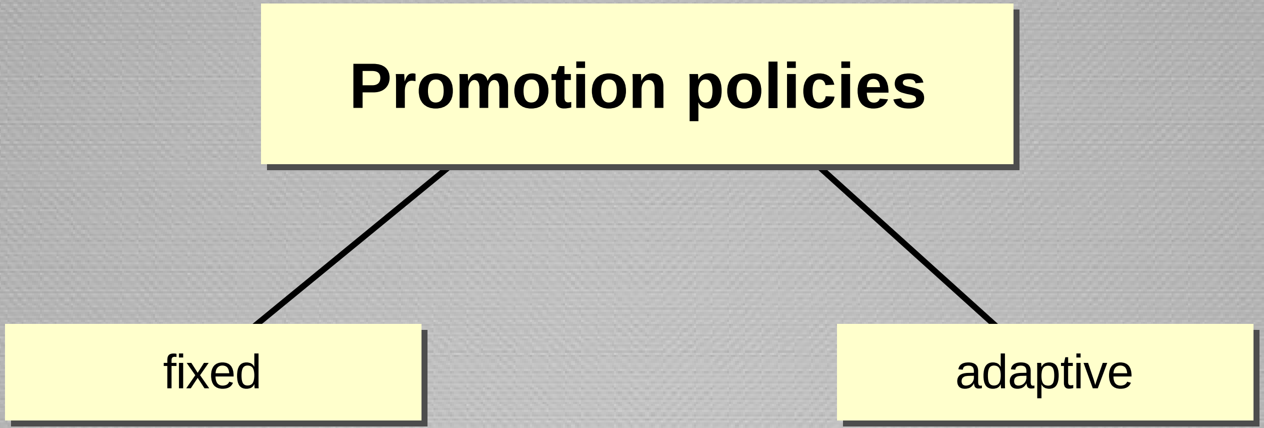
late promotion

better **long-living objects** *worse*

worse **short-living objects** *better*

time

Promotion policies



Promotion policies

fixed

adaptive

Appel‘s collector for SML/NJ

demographic feedback-mediated tenuring

Appel‘s collector for SML/NJ (1)

Idea: manage promotion rates by
fixing heap occupancy of young objects

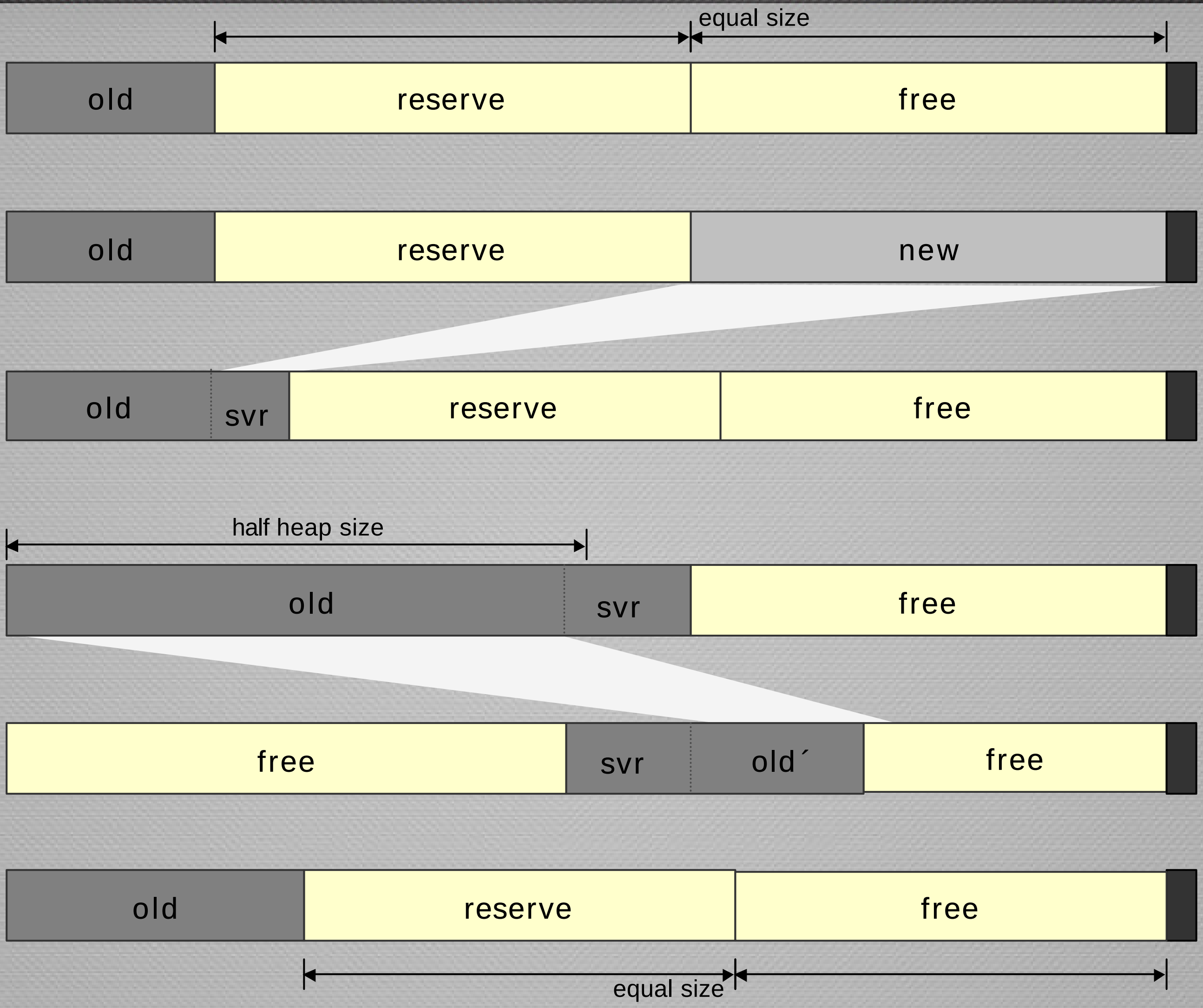
- two generations,
  very large young generation

- major collections only if old objects
  occupy *half size of heap*

- precondition: *contigious heap*

[Appel, 1989]

Appel‘s collector for SML/NJ (2)



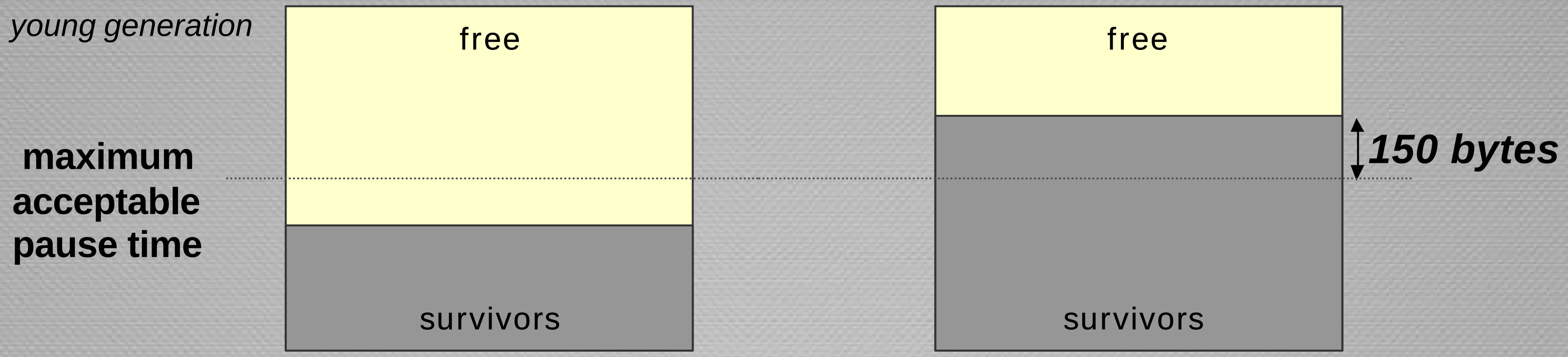
Demographic feedback-mediated tenuring (1)

Idea: promote only when necessary
to hold maximum pause time

- only promote if *pause time* will be *acceptable*
- generate *space-age table*
- promote *only as many objects* to make pause time acceptable

[Ungar, Jackson 1992]

Demographic feedback-mediated tenuring (2)



No promotion at next collection

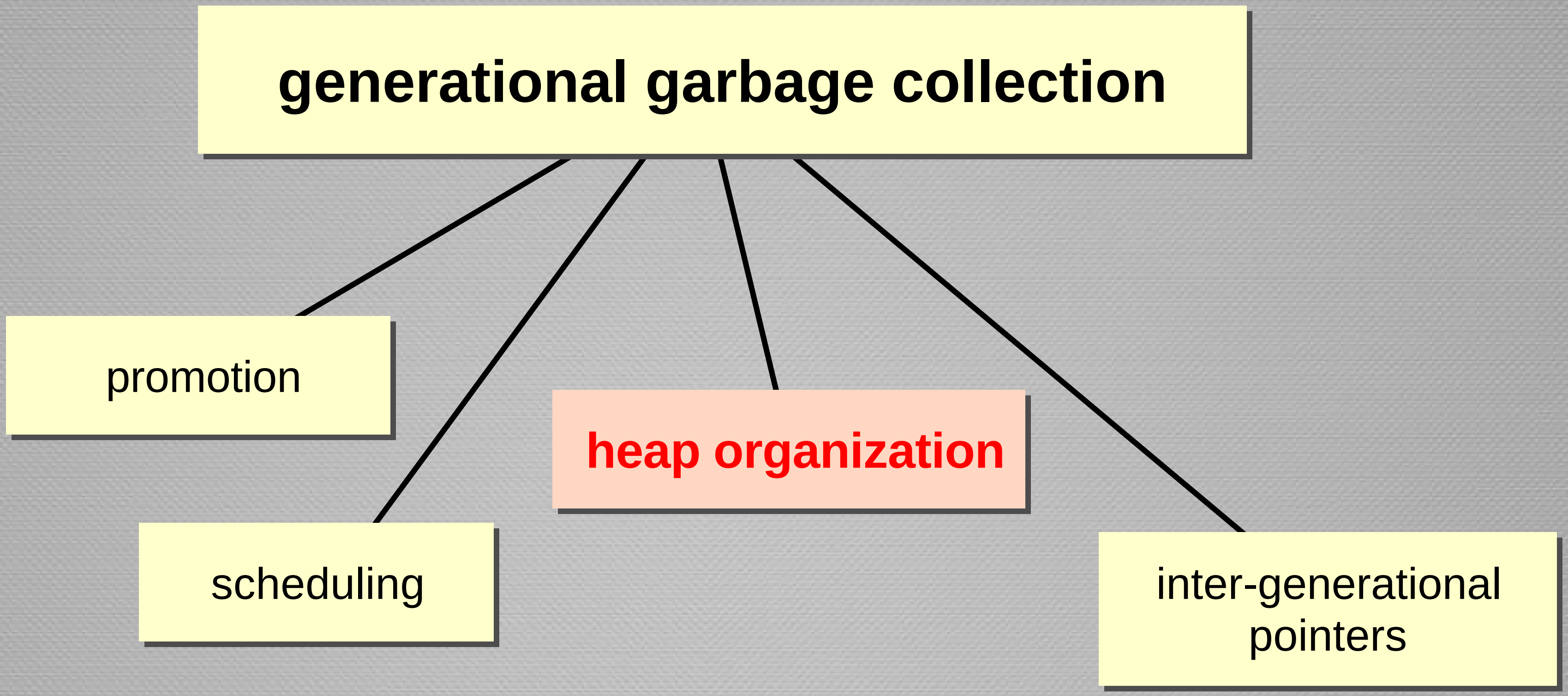
promotion of 150 bytes „oldest“ objects

age	size of age group
1	300 bytes
2	200 bytes
3	100 bytes

number of survived collections

size of objects in age group

[Ungar, Jackson 1992]



generational garbage collection

promotion

scheduling

heap organization

inter-generational pointers

Heap organization

Determination of object‘s generation

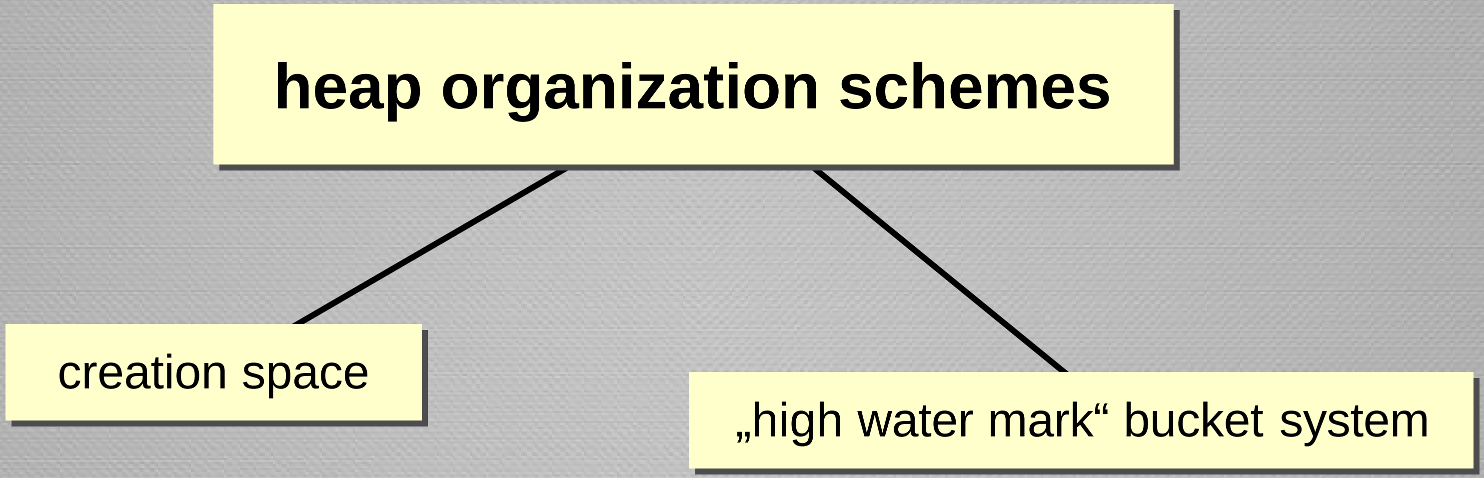
- copying collectors: *subheaps* for generation

  contigious heaps → object adress

  non-contigious heaps → header field or page-table

- non-copying collectors → header field

Heap organization schemes

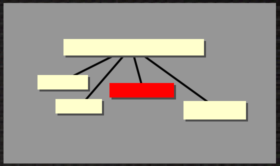


heap organization schemes

creation space

„high water mark“ bucket system

Creation space (1)



Goals:

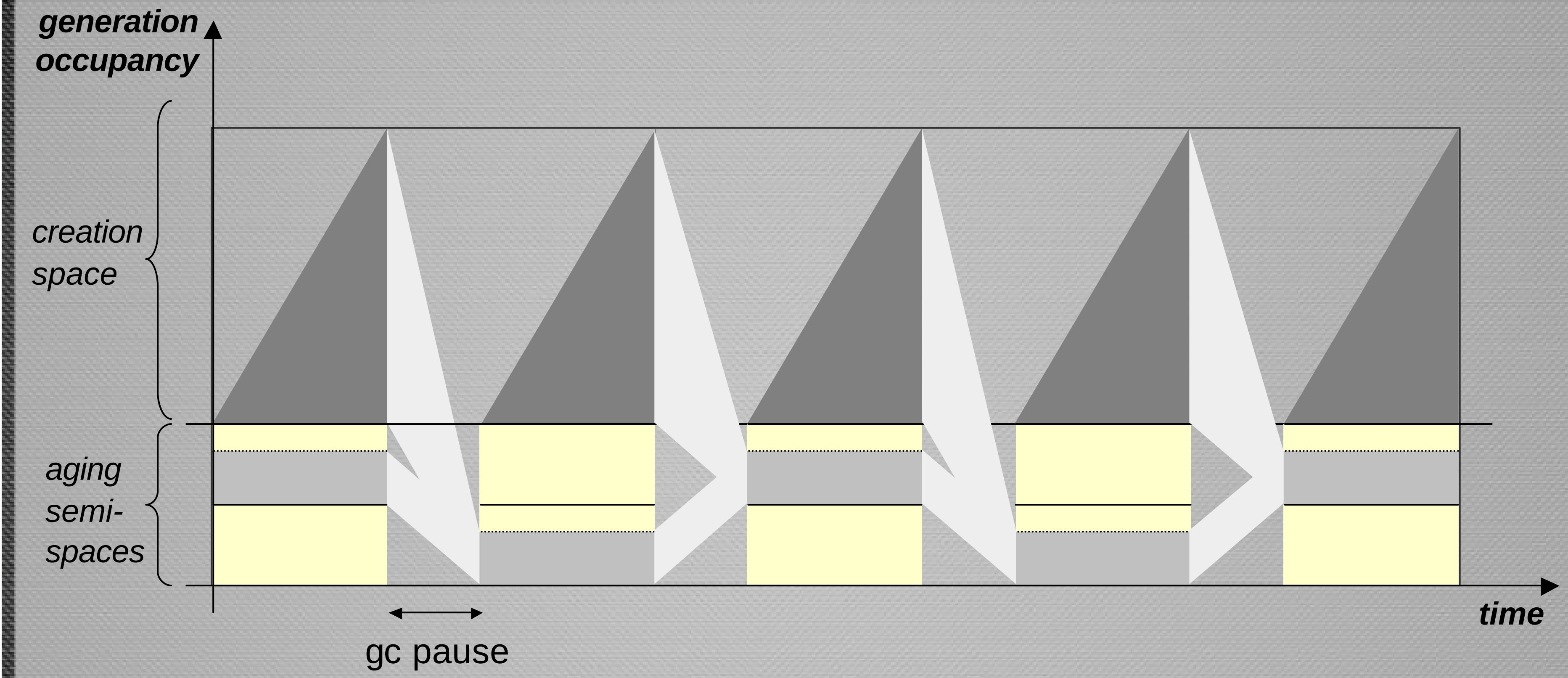
- no large semi-spaces
- improve locality

Organization of a generation:

- (small) *aging area* in two semi-spaces
- (large) *new object area* in one *creation space*

[Ungar, 1984]

Creation space (2)



[Ungar, 1984]

„High water mark“ bucket system (1)

Goals:

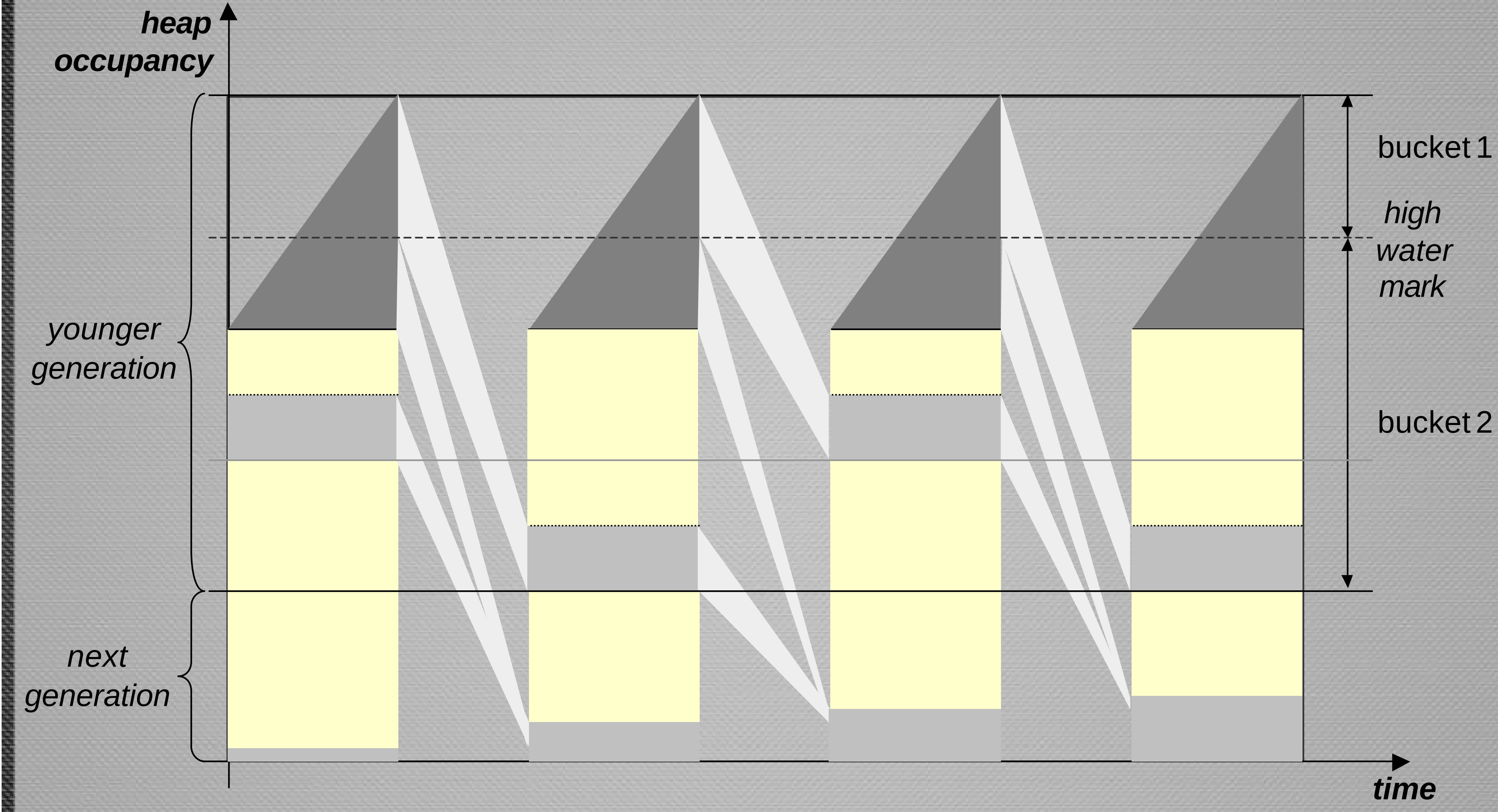
- avoid age field in object header
- adaptive promotion threshold

Organization of generations:

- creation and aging spaces per generation
- two *buckets* per generation
- creation space partly holds first bucket
- *high water mark* seperates buckets

[Wilson, Moher 1989]

„High water mark“ bucket system (2)



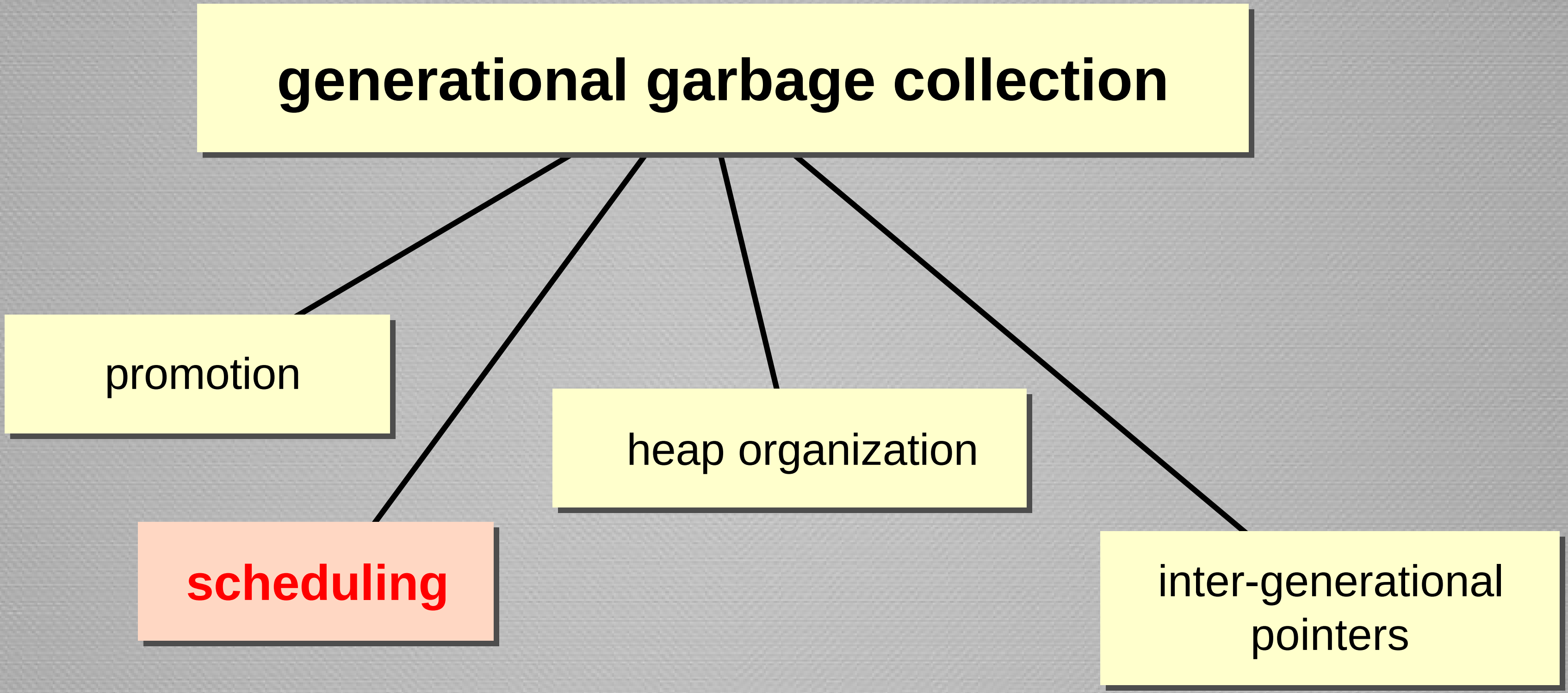
[Wilson, Moher 1989]

„High water mark“ bucket system (3)

„high water mark” effect:

- objects from bucket 2 are promoted into older generation
- objects from bucket 1 are stored in bucket 2
- „high water mark“ position determines promotion threshold
- promotion threshold between 1 and 2

[Wilson, Moher 1989]



generational garbage collection

promotion

scheduling

heap organization

inter-generational pointers

Collection scheduling

Perform collection when :

- pause is not interruptive

hide collection from user

- large amount of garbage can be expected

efficent collection

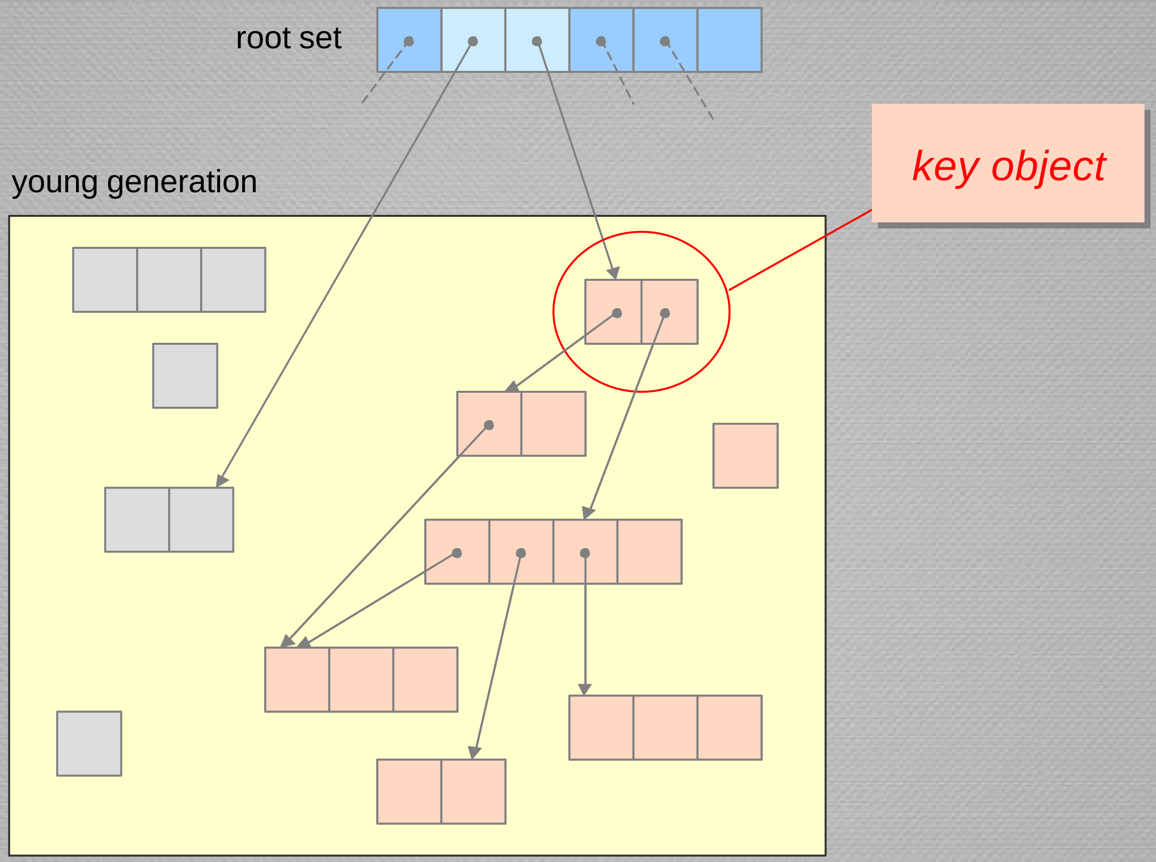
Efficient collections

Key (large) objects :

- objects whose „death“ produces much garbage (root of a large tree etc.)
- exclude key objects from generational scheme
- store key objects „descendants“ in special *large object area*
- reclaiming of key objects trigger collection in key area

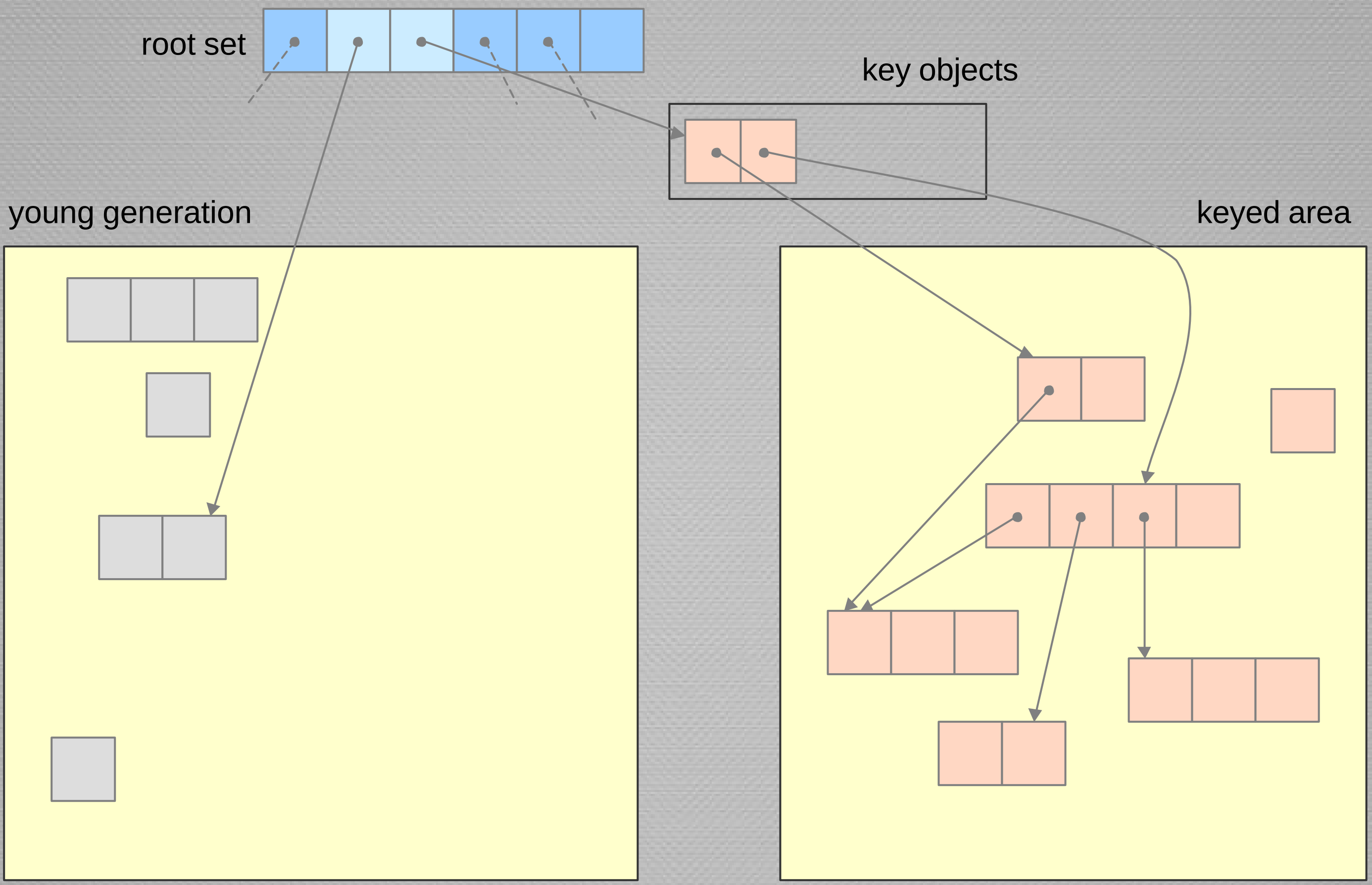
[Hayes 1991]

Efficient collections – key objects (1)

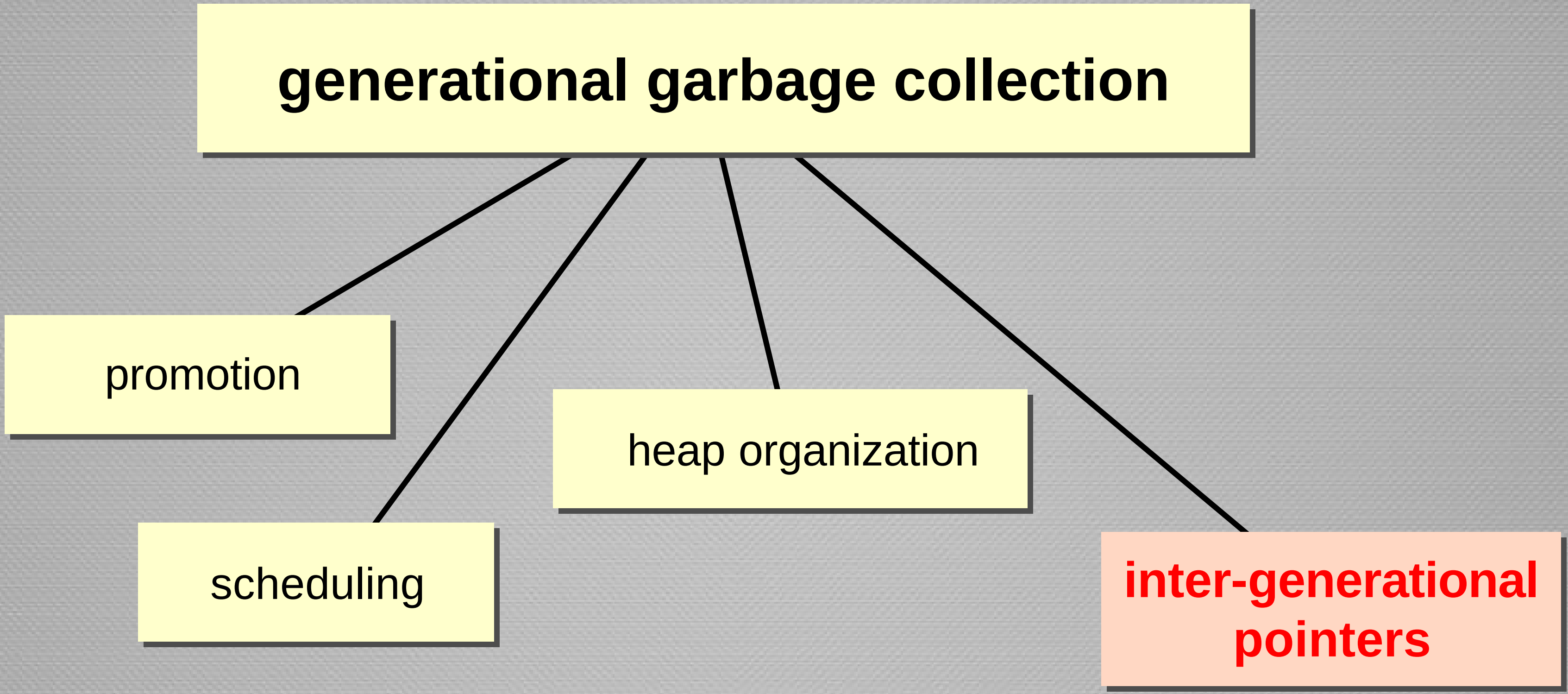


[Hayes 1991]

Efficient collections – key objects (2)



[Hayes 1991]



generational garbage collection

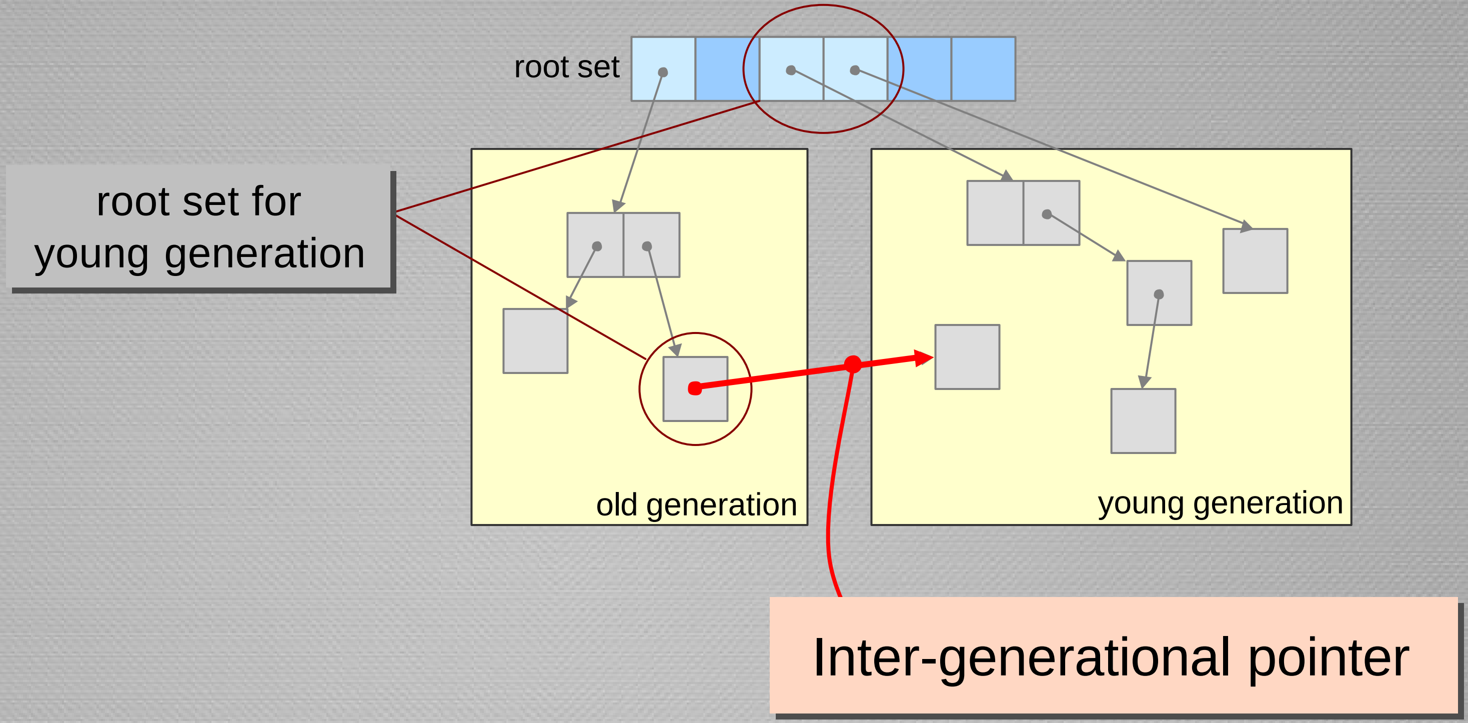
promotion

scheduling

heap organization

inter-generational pointers

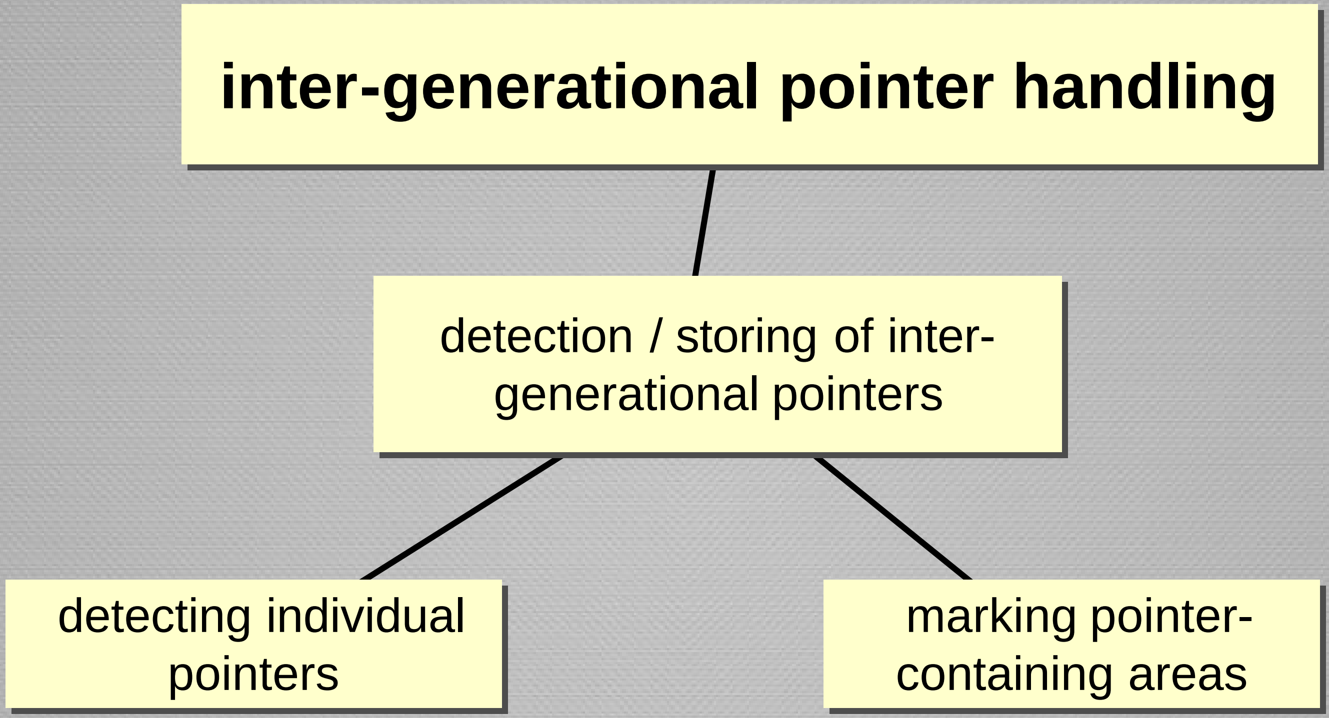
Inter-generational pointers



Issues:

- detecting creation
- including into root set upon collection
- cost of detection / inclusion

Inter-generational pointer handling



inter-generational pointer handling

detection / storing of inter-generational pointers

detecting individual pointers

marking pointer-containing areas

- *pointer indirection*
- *remembered sets*

page / word / card marking

Detection

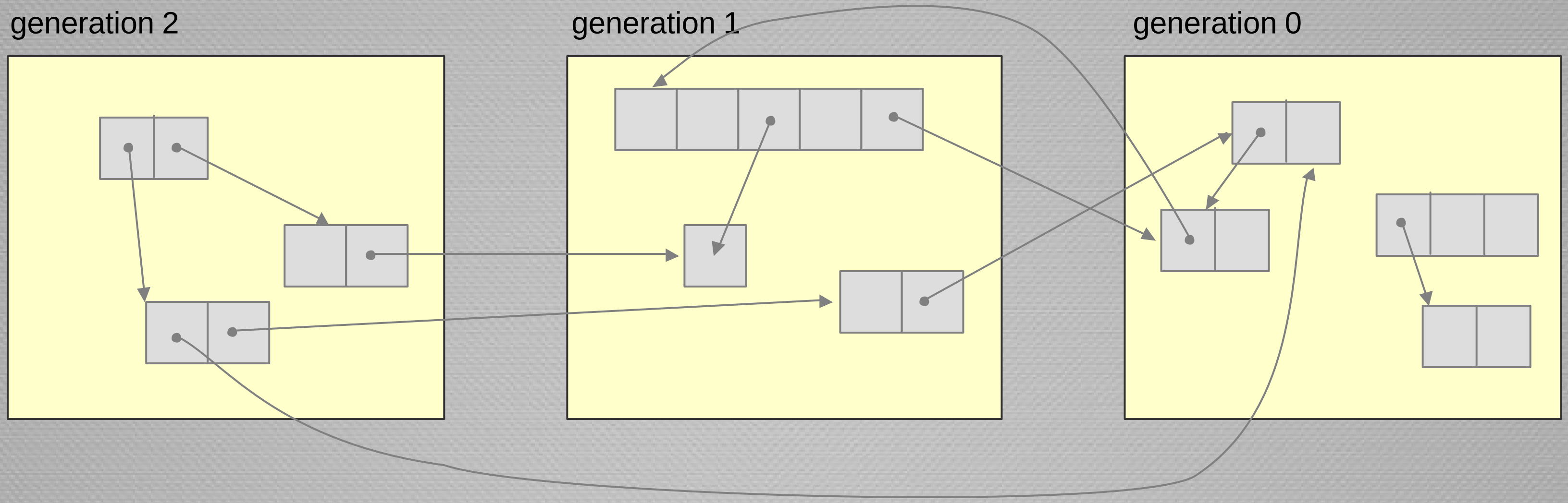
Detection of inter-generational pointers:

- trapping pointer stores
- only necessary to check objects in old generation(s)
- generally: only non-initializing stores

write-barrier

Individual pointers – entry table

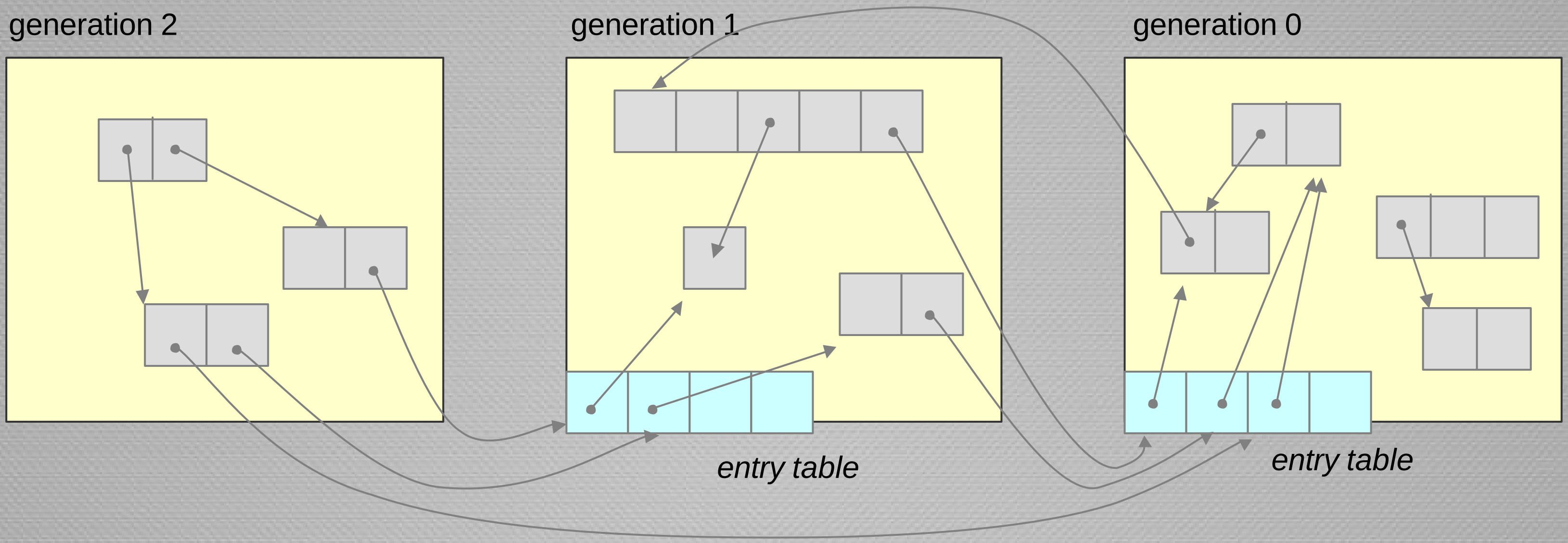
Idea: indirect pointers through an *entry table*



[Liebermann, Hewitt 1983]

Individual pointers – entry table

Idea: indirect pointers through an *entry table*



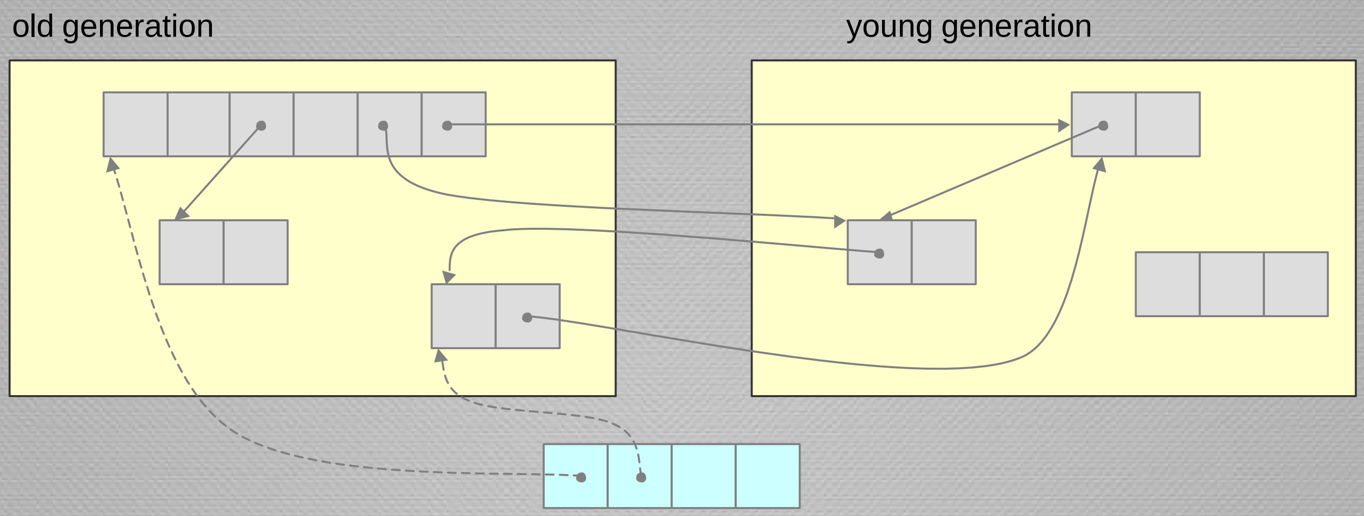
	time	**space**
trapping stores (by mutator)	small constant for *every* store	O(#stores), small constant
inter-generational pointer-handling at collection time	O(#stores)	N/A

all pointers in the entry table are added to the root set

[Liebermann, Hewitt 1983]

Individual pointers – remembered set

Idea: *remember* objects with old-young pointers



	time	**space**
trapping stores (by mutator)	small constant for *every* store	O(# pointer-containers)
inter-generational pointer-handling at collection time	O(# pointer-containers)	N/A

no duplicate entries in set due to bit in object header

Objects are scanned for pointers:
O(size of objects)

[Ungar 1984]

Pointer areas – page marking

Idea: *mark* (virtual) memory *pages* containing objects with inter-generational pointers

- hardware / virtual memory management support
- only slight overhead for write barrier
- problems intercepting VM signals
- large pages → high collection cost

	time	**space**
trapping stores (by mutator)	small constant for *every* store	N/A
inter-generational pointer-handling at collection time	O(# i.g.p. • size of page)	N/A

→ pages are scanned for pointers

[Moon 1984]

[Shaw 1988]

Pointer areas – card marking

Idea: don´t mark to-large pages
or to-small words

- divide adress space into *cards* (~128 bytes)
- lower collection cost if card size is near object size
- very low cost for write barrier: 2-3 instructions

	time	**space**
trapping stores (by mutator)	small constant for *every* store	small portion of heap
inter-generational pointer-handling at collection time	$O(\#\,\text{i.g.p.} \bullet \text{size of card})$	N/A

→ cards are scanned for pointers

[Sobalvarro 1988]

[Wilson, Moher 1989]

Remembered sets vs. card marking

remembered sets (with *sequential store buffers*):

- no scanning upon collection ✓
- collection overhead *O(#pointer stores)* ✗
- duplicates in sequential store buffers ✗
- small overhead for write barrier (2-3 instructions) ✓

Remembered sets vs. card marking

card marking

- scanning upon collection ✗
- collection overhead *O(#inter-gen. pointers)* ✓
- small overhead for write barrier (2-3 instructions) ✓
- long-lived object´s cards must be scanned repeatedly ✗

Remembered sets vs. card marking

hybrid card marking / remembered set GC

- write barrier like card-marking
- after collection, old-young-pointers
  are added to remembered set

[Hosking, Hudson 1993]

Problems and limitations

heuristic failure:

- cluster of long-lived objects „*pig in the snake*“
- small heap-allocated objects
- large root sets

Conclusion

- **improvement, if assumptions hold**

- **highly variable**

Thank you !