Incremental Garbage Collection

Part II

Roland Schatz

Baker – Disadvantages

- not concurrent

Baker – Disadvantages

- not concurrent
- not even threadsafe

Baker – Disadvantages

- not concurrent
- not even threadsafe
- expensive read trap

Baker – Disadvantages

- not concurrent
- not even threadsafe
- expensive read trap
- requires hardware support

Appel-Ellis-Li Collector

- no black $\Rightarrow$ white pointers

Appel-Ellis-Li Collector

- no black $\Rightarrow$ white pointers
- Baker's invariant

Appel-Ellis-Li Collector

- no black $\Rightarrow$ white pointers
- Baker's invariant
  - mutator never sees white pointers

Appel-Ellis-Li Collector

- no black $\Rightarrow$ white pointers
- Baker's invariant
  - mutator never sees white pointers
  - intercept all pointer reads

Appel-Ellis-Li Collector

- no black $\Rightarrow$ white pointers
- Baker's invariant
  - mutator never sees white pointers
  - intercept all pointer reads
- slightly stricter constraint

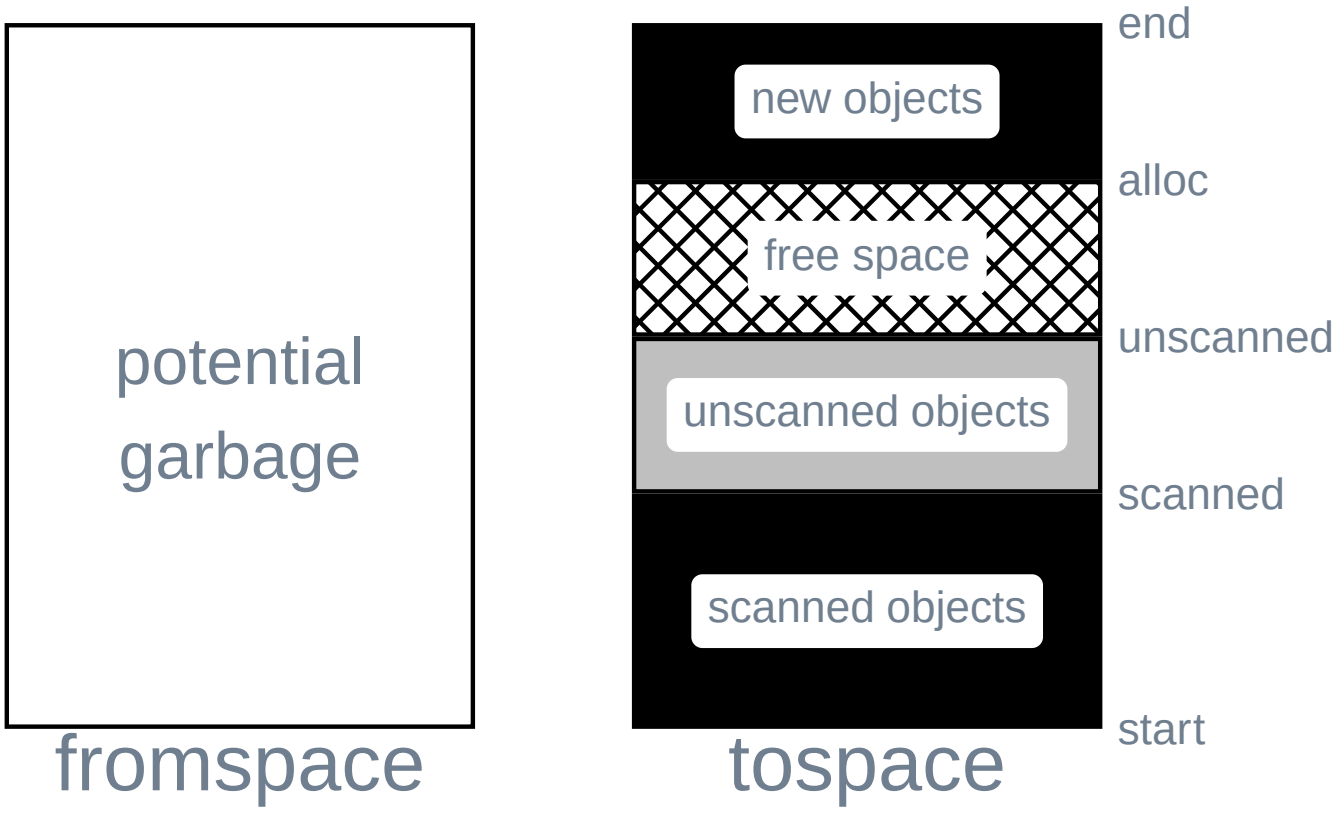
Appel-Ellis-Li Collector

- no black $\Rightarrow$ white pointers
- Baker's invariant
  - mutator never sees white pointers
  - intercept all pointer reads
- slightly stricter constraint
  - mutator only sees black objects

Appel-Ellis-Li Collector

- no black $\Rightarrow$ white pointers
- Baker's invariant
  - mutator never sees white pointers
  - intercept all pointer reads
- slightly stricter constraint
  - mutator only sees black objects
  - use page protection hardware

Memory Layout



Page Protection

- black pages

Page Protection

- black pages
  - contain only tospace pointers

Page Protection

- black pages
  - contain only tospace pointers
  - no problem

Page Protection

- black pages
  - contain only tospace pointers
  - no problem
- white pages

Page Protection

- black pages
  - contain only tospace pointers
  - no problem
- white pages
  - mutator never sees white pointers

Page Protection

- black pages
  - contain only tospace pointers
  - no problem
- white pages
  - mutator never sees white pointers
  - so we don’t care

Page Protection

- gray pages

Page Protection

- gray pages
  - mutator may see gray pointers

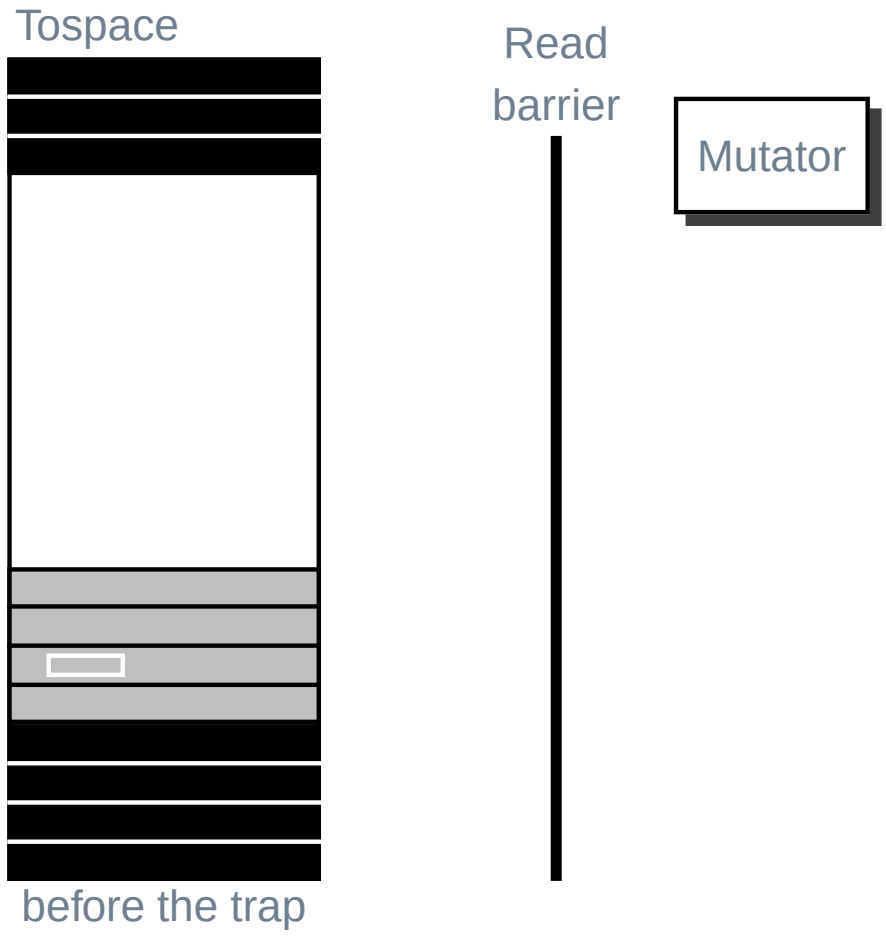
Page Protection

- gray pages
  - mutator may see gray pointers
  - mutator must not see gray objects

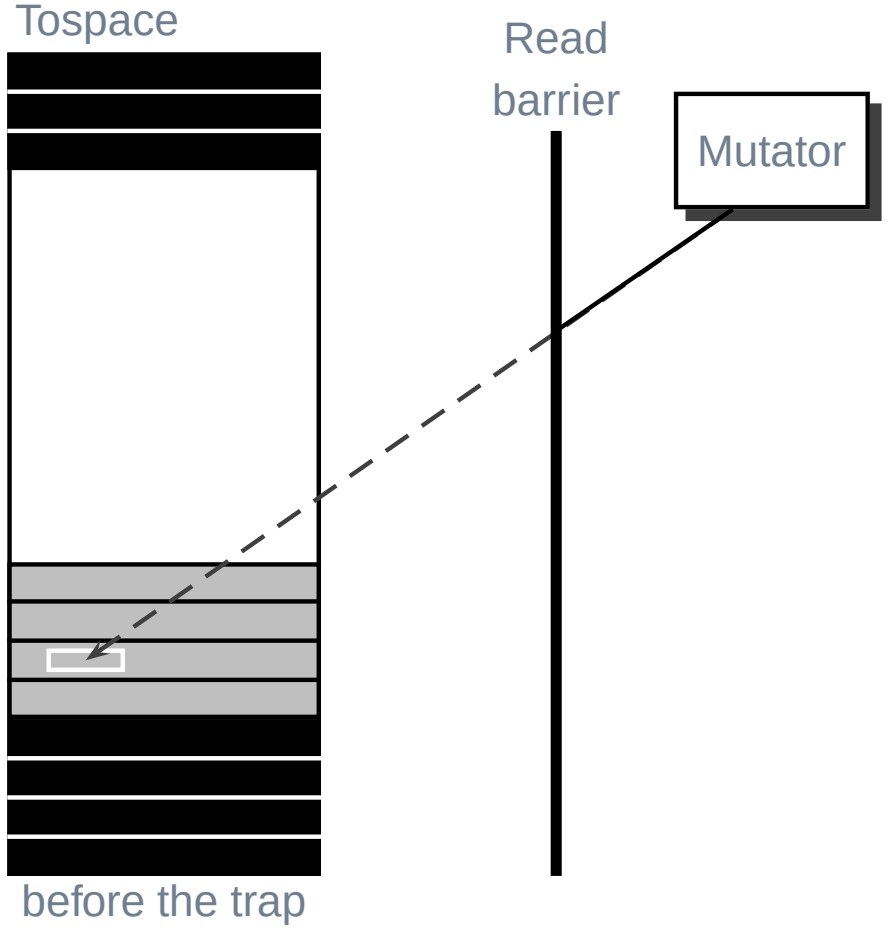
Page Protection

- gray pages
  - mutator may see gray pointers
  - mutator must not see gray objects
  - set to no access

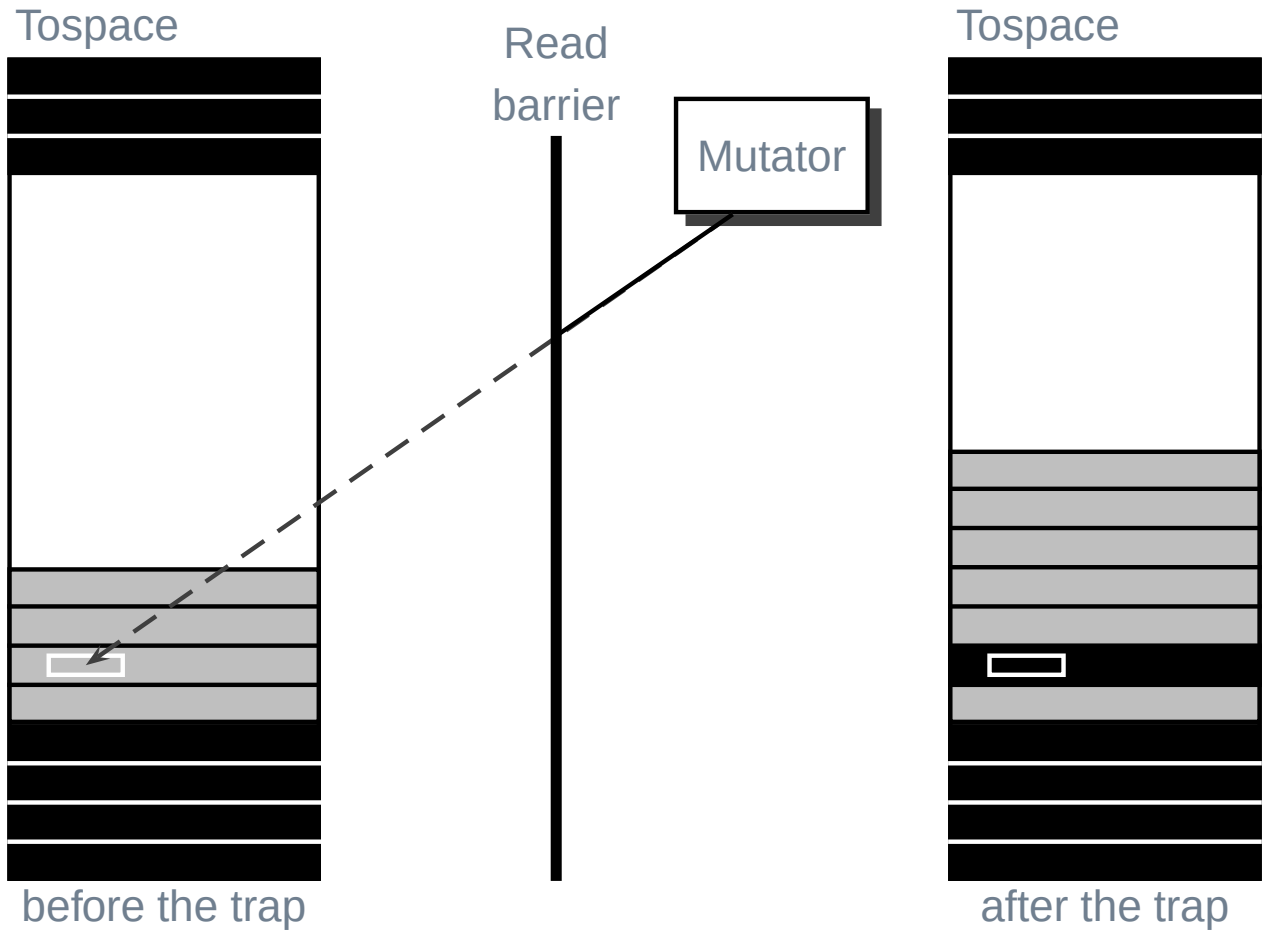
Handling Page Faults



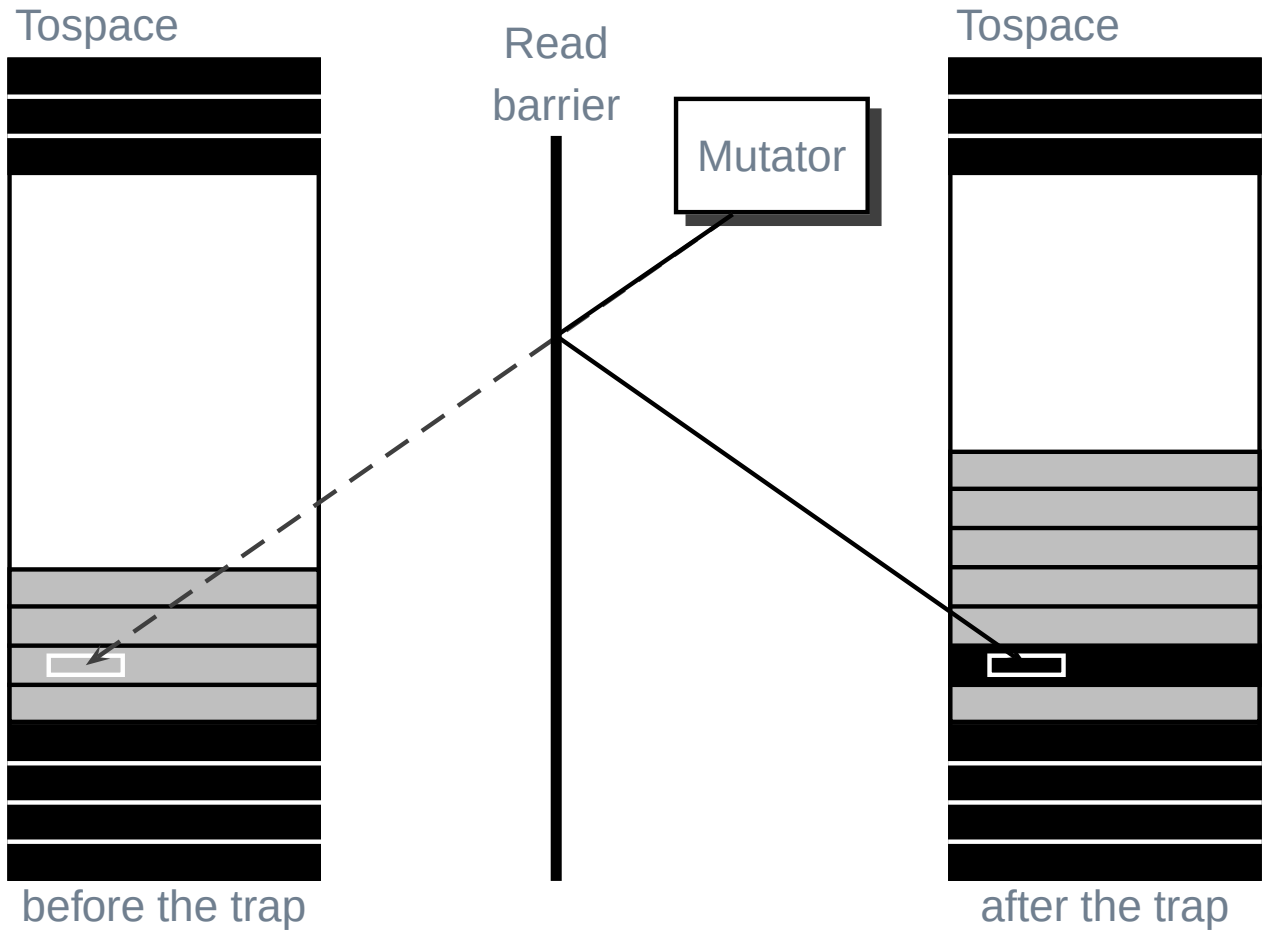
Handling Page Faults



Handling Page Faults



Handling Page Faults



Summary

- advantages

Summary

- advantages
  - concurrent scanning possible

Summary

- advantages
  - concurrent scanning possible
  - cheaper read trap

Summary

- advantages
  - concurrent scanning possible
  - cheaper read trap
- problems

Summary

- advantages
  - concurrent scanning possible
  - cheaper read trap
- problems
  - relies on virtual memory

Summary

- advantages
  - concurrent scanning possible
  - cheaper read trap
- problems
  - relies on virtual memory
  - global lock is bottleneck

Summary

- advantages
  - concurrent scanning possible
  - cheaper read trap
- problems
  - relies on virtual memory
  - global lock is bottleneck
  - flip is still stop-the-world

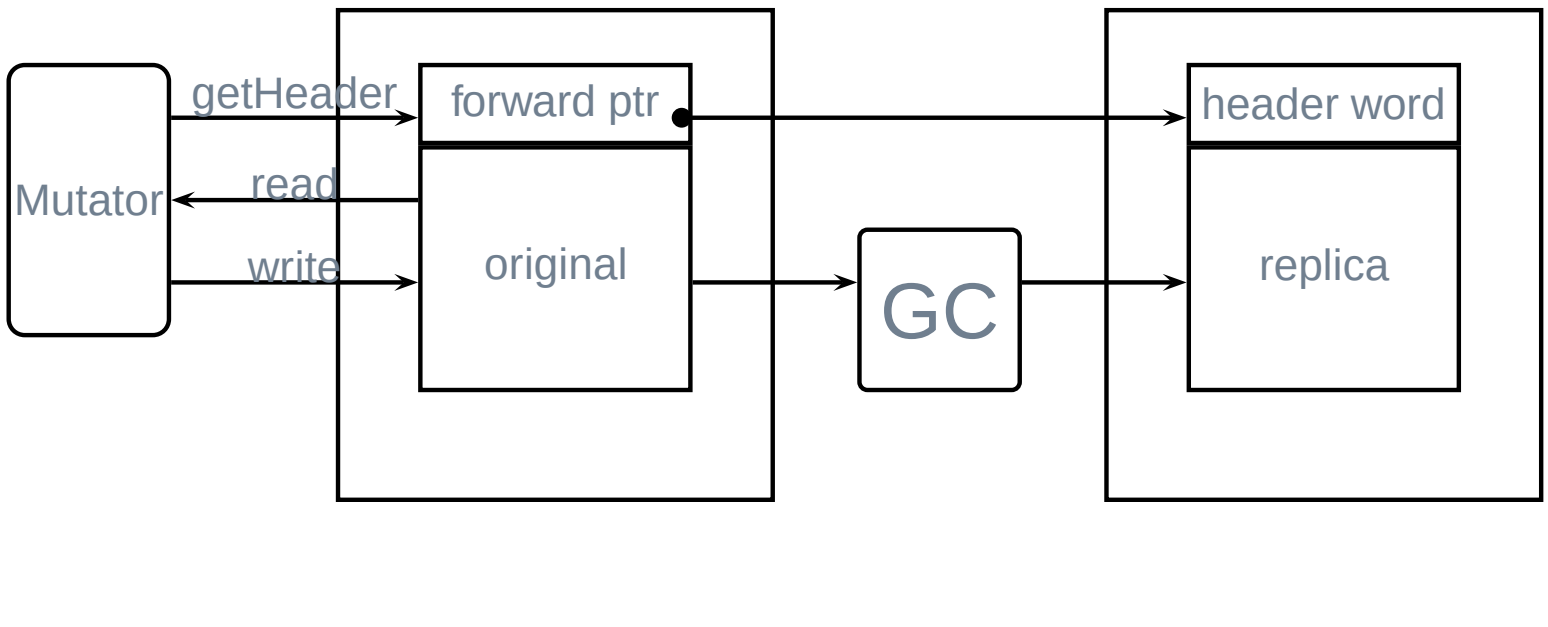
Nettle's Replicating Collector

- read barrier is still expensive

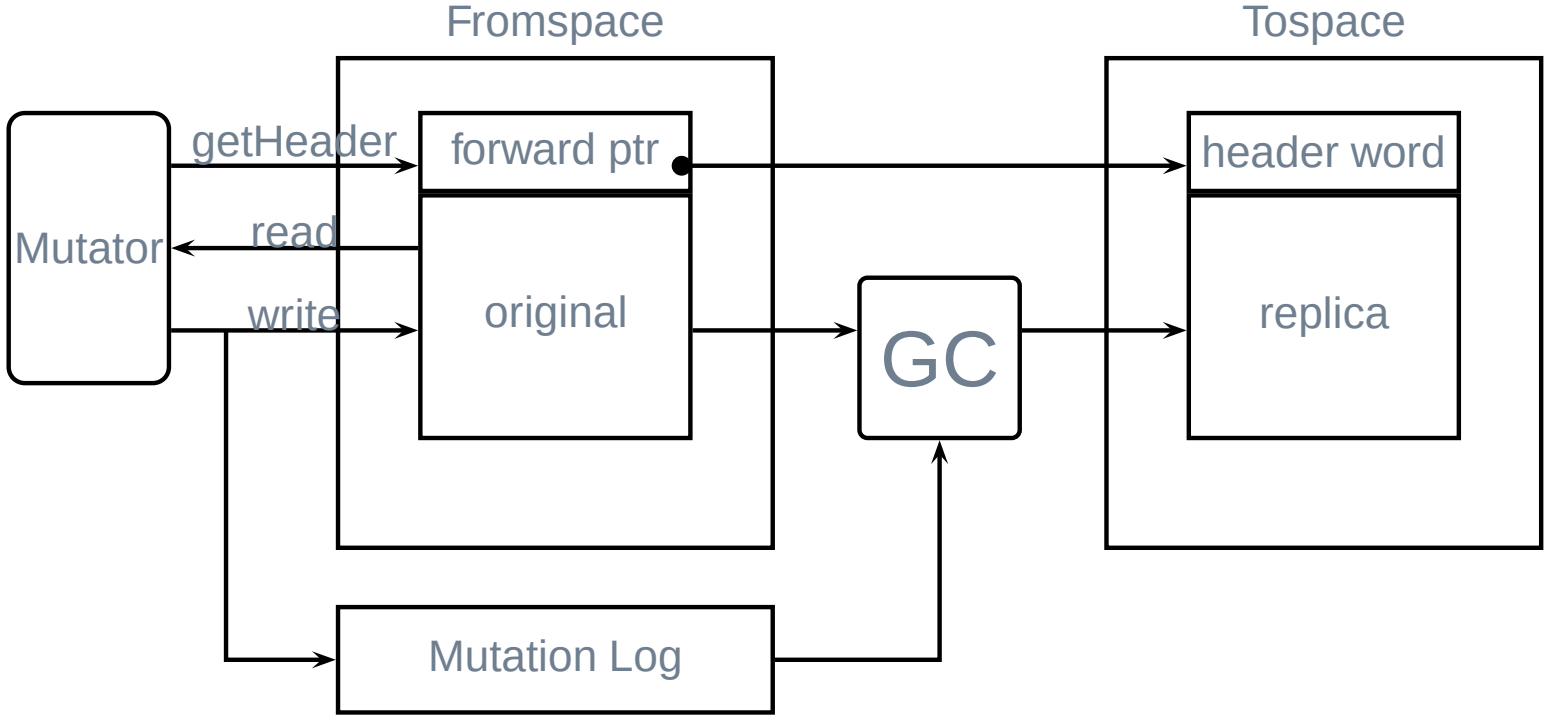
Nettle's Replicating Collector

- read barrier is still expensive
- use write barrier instead

Nettle's Replicating Collector



Nettle's Replicating Collector



Huelsbergen-Laurus Collector

- distinguish mutable/immutable objects

Huelsbergen-Laurus Collector

- distinguish mutable/immutable objects
  - e.g. ML

Huelsbergen-Laurus Collector

- distinguish mutable/immutable objects
  - e.g. ML
- immutable objects

Huelsbergen-Laurus Collector

- distinguish mutable/immutable objects
  - e.g. ML
- immutable objects
  - replicating collection

Huelsbergen-Laurus Collector

- distinguish mutable/immutable objects
  - e.g. ML
- immutable objects
  - replicating collection
- mutable objects

Huelsbergen-Laurus Collector

- distinguish mutable/immutable objects
  - e.g. ML
- immutable objects
  - replicating collection
- mutable objects
  - classic copying collection

Doligez-Leroy Collector

- Problem: heavy synchronization

Doligez-Leroy Collector

- Problem: heavy synchronization
  - seperate thread-local data

Doligez-Leroy Collector

- Problem: heavy synchronization
  - seperate thread-local data
- young generation

Doligez-Leroy Collector

- Problem: heavy synchronization
  - seperate thread-local data
- young generation
  - thread-local

Doligez-Leroy Collector

- Problem: heavy synchronization
  - seperate thread-local data
- young generation
  - thread-local
  - immutable

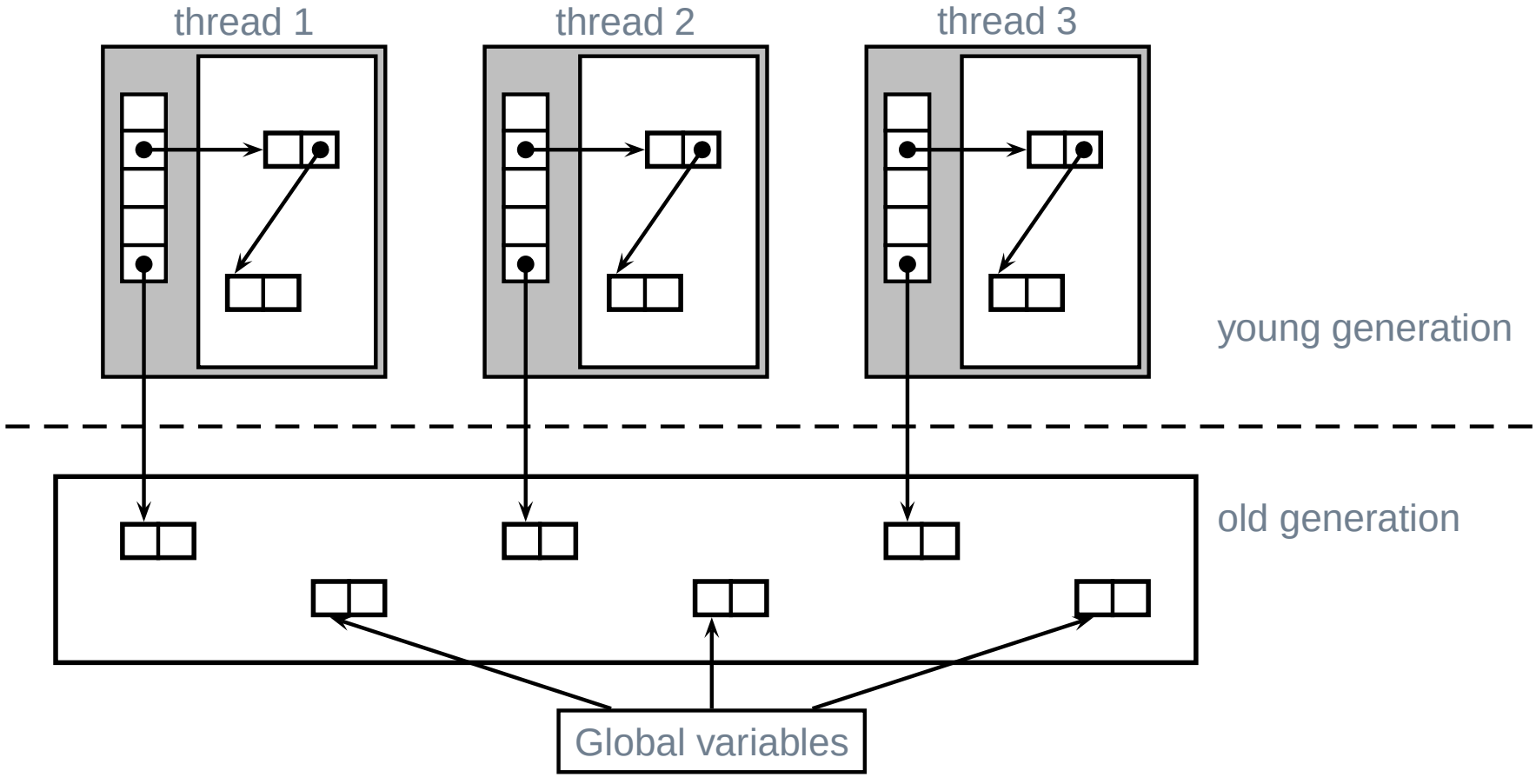
Doligez-Leroy Collector

- Problem: heavy synchronization
  - seperate thread-local data
- young generation
  - thread-local
  - immutable
- old generation

Doligez-Leroy Collector

- Problem: heavy synchronization
  - seperate thread-local data
- young generation
  - thread-local
  - immutable
- old generation
  - the rest

Thread Local Heaps



In-Place Garbage Collection

- uncooperative environment

In-Place Garbage Collection

- uncooperative environment
- relaxed consistency requirements

In-Place Garbage Collection

- uncooperative environment
- relaxed consistency requirements
- Fragmentation!

Four-Color Abstraction

- black: scanned

Four-Color Abstraction

- black: scanned
- gray: marked, not scanned

Four-Color Abstraction

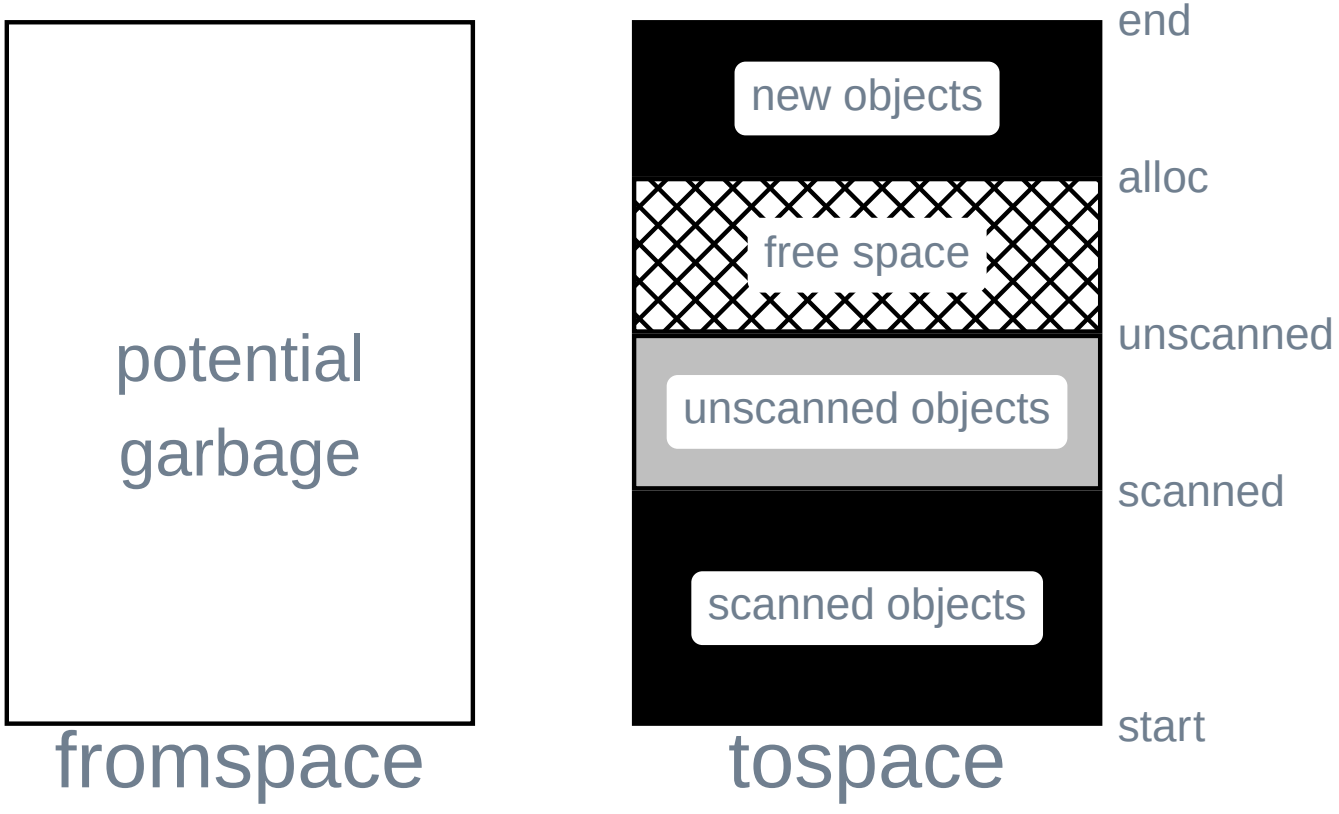
- black: scanned
- gray: marked, not scanned
- white: not marked

Four-Color Abstraction

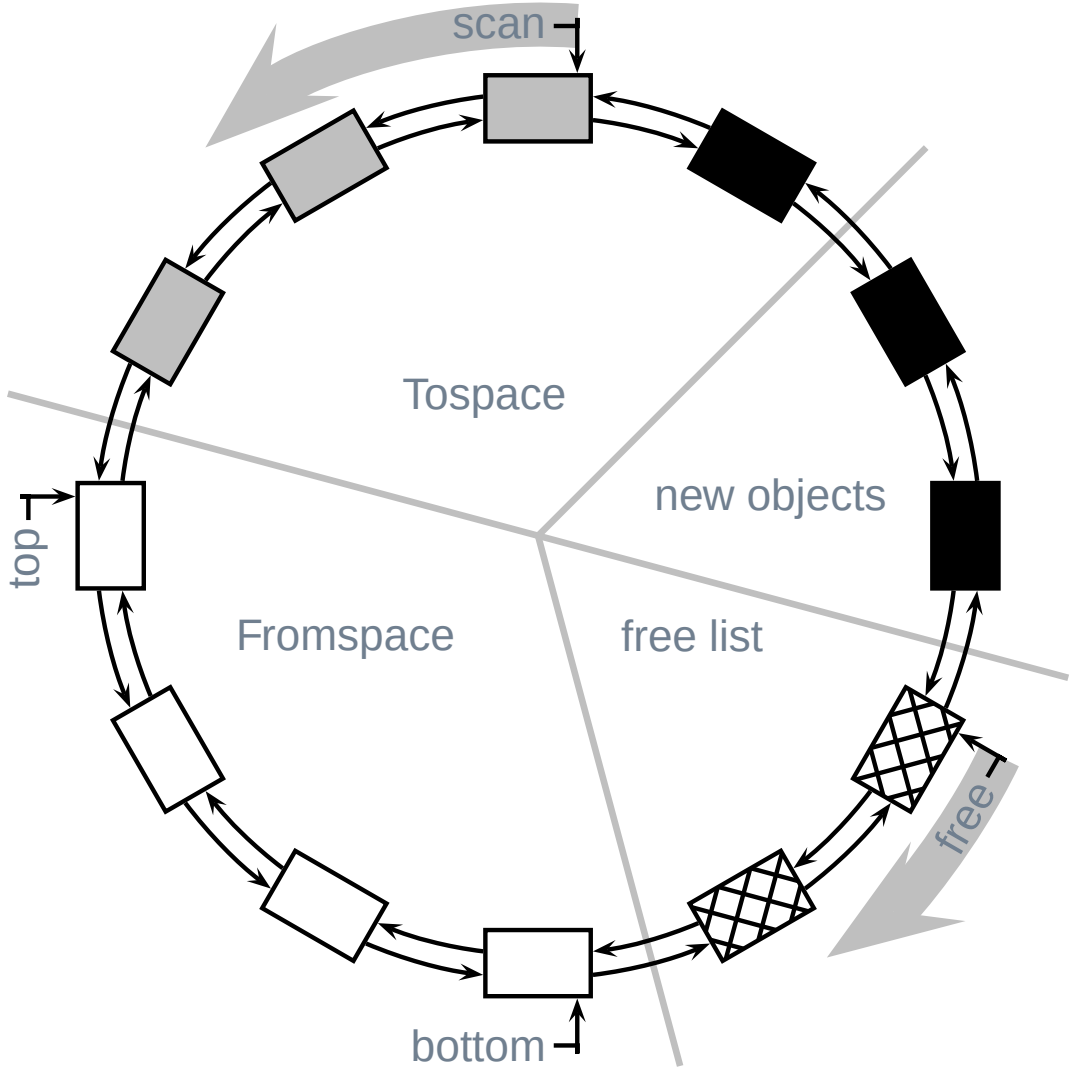
- black: scanned
- gray: marked, not scanned
- white: not marked

- dead-white: free objects

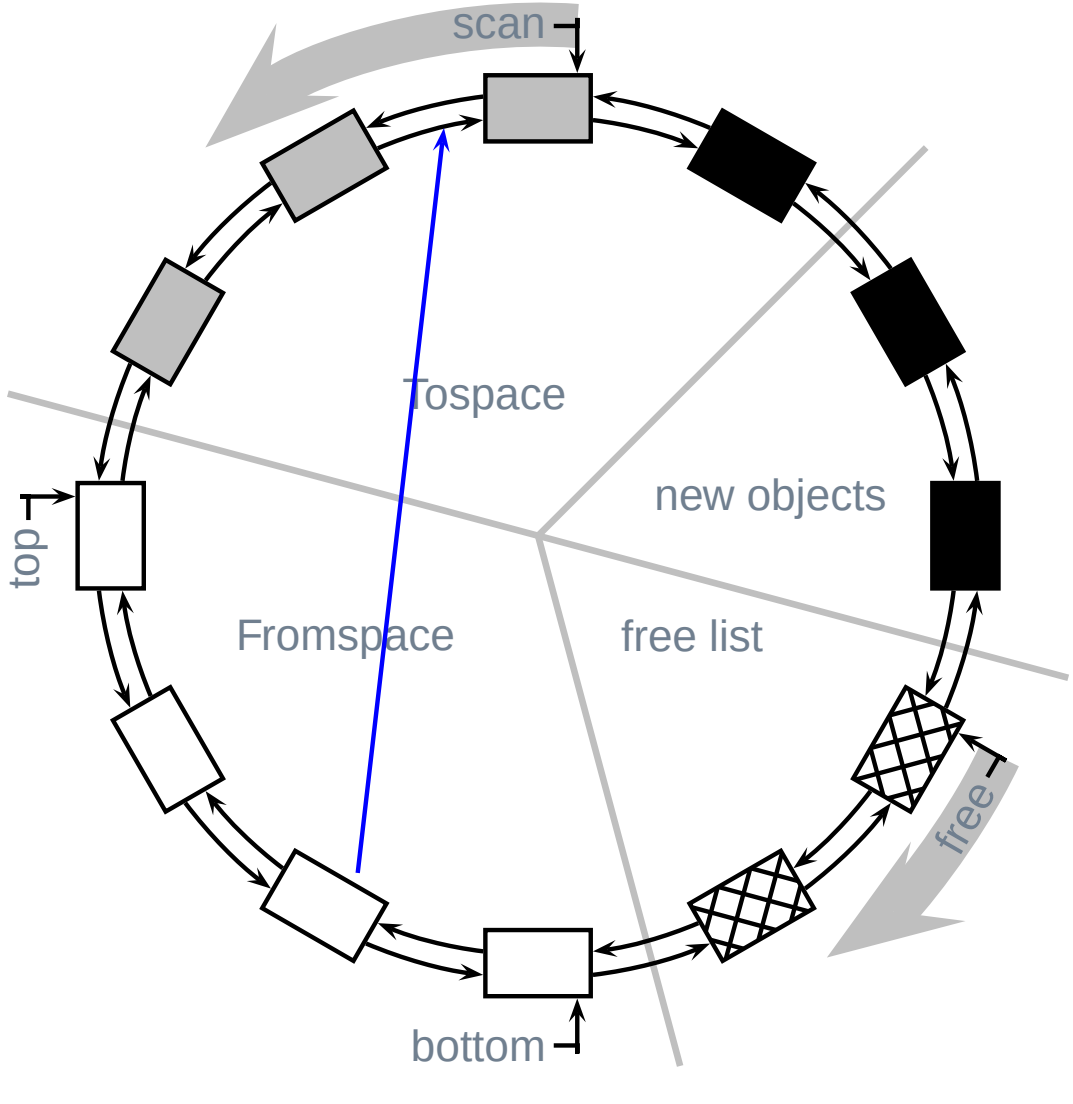
Four-Color Implementation



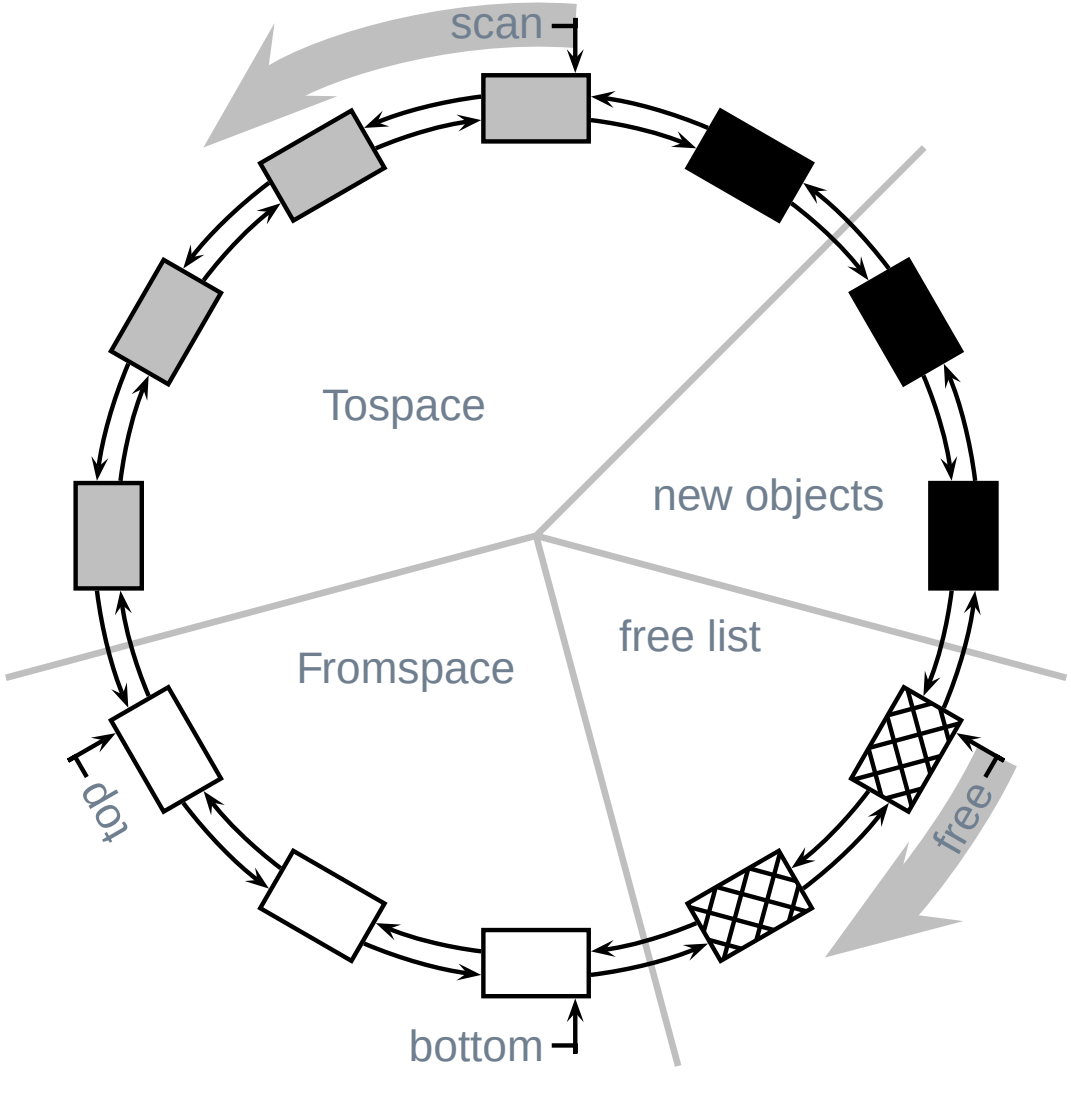
Baker's Treadmill



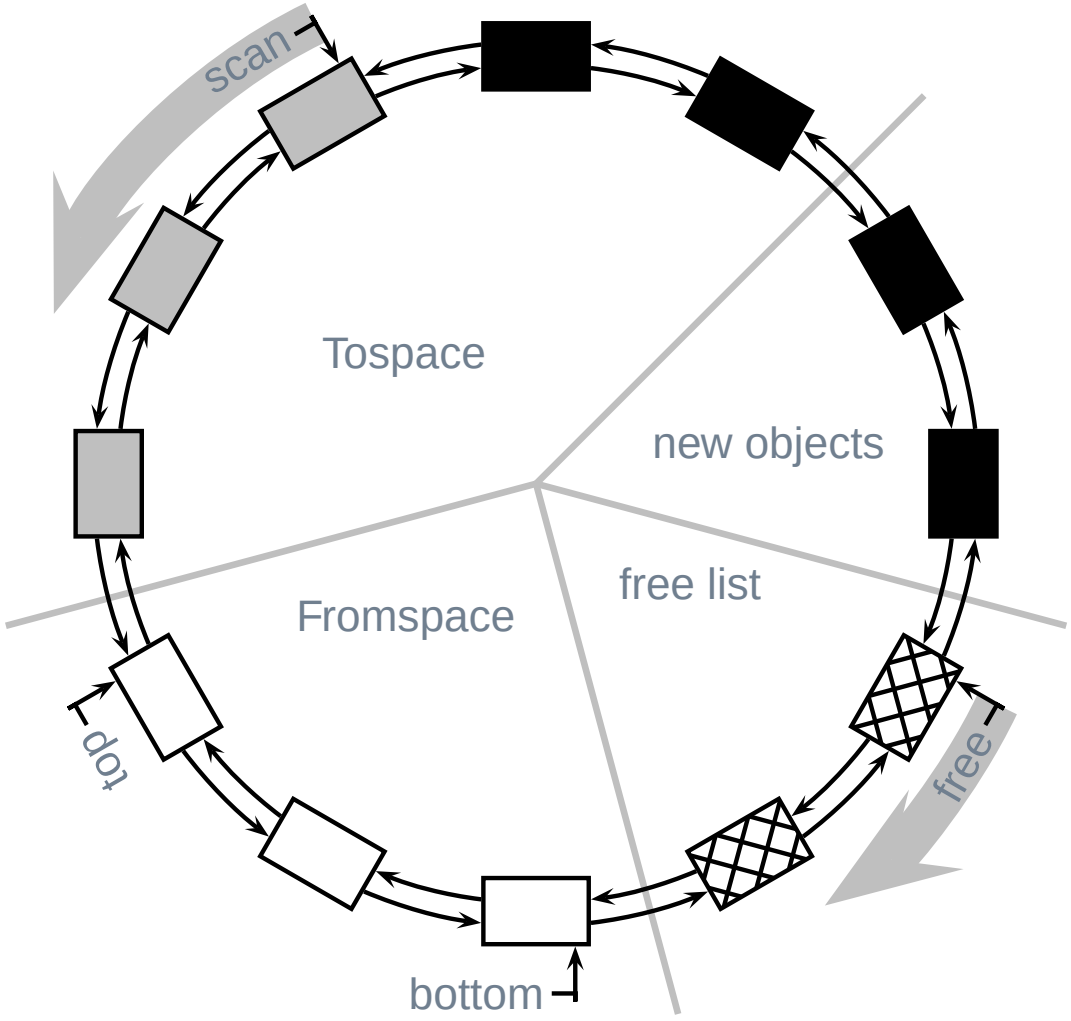
Baker's Treadmill



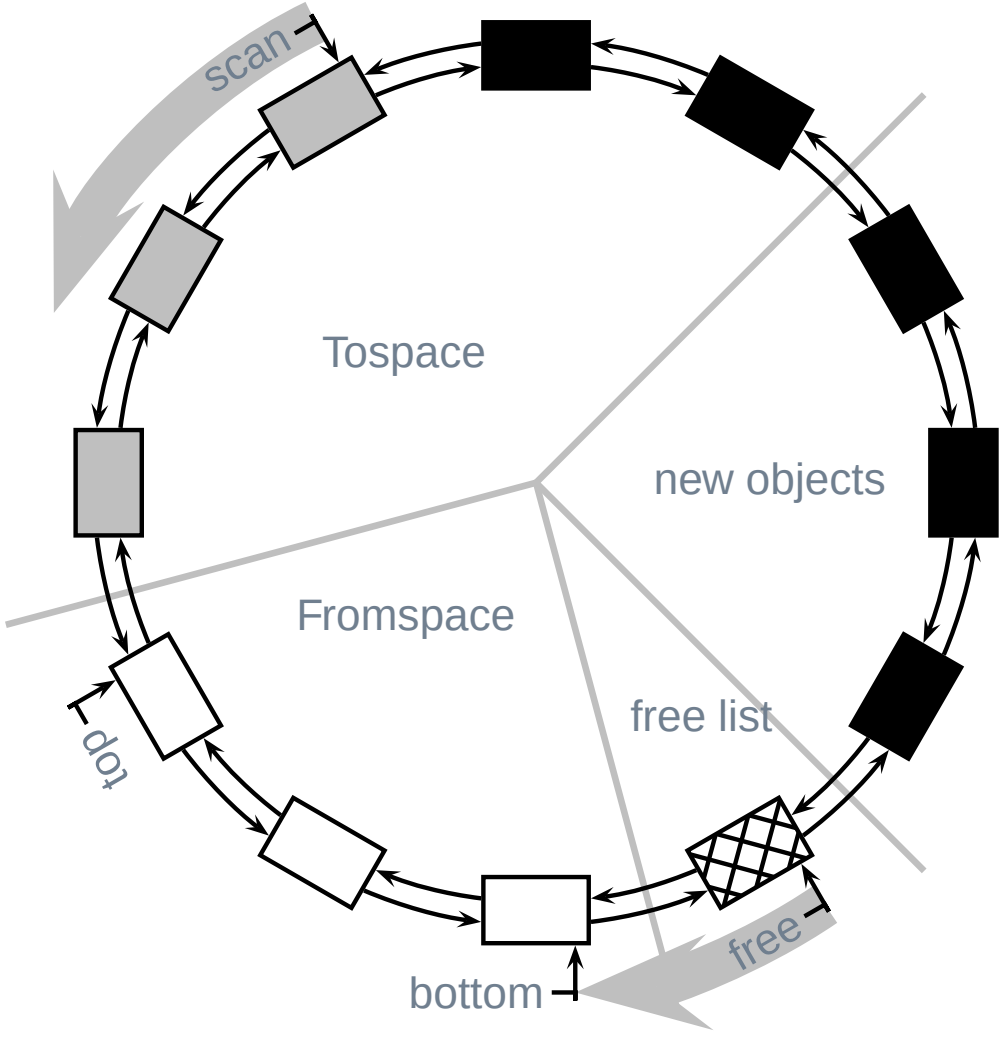
Baker's Treadmill



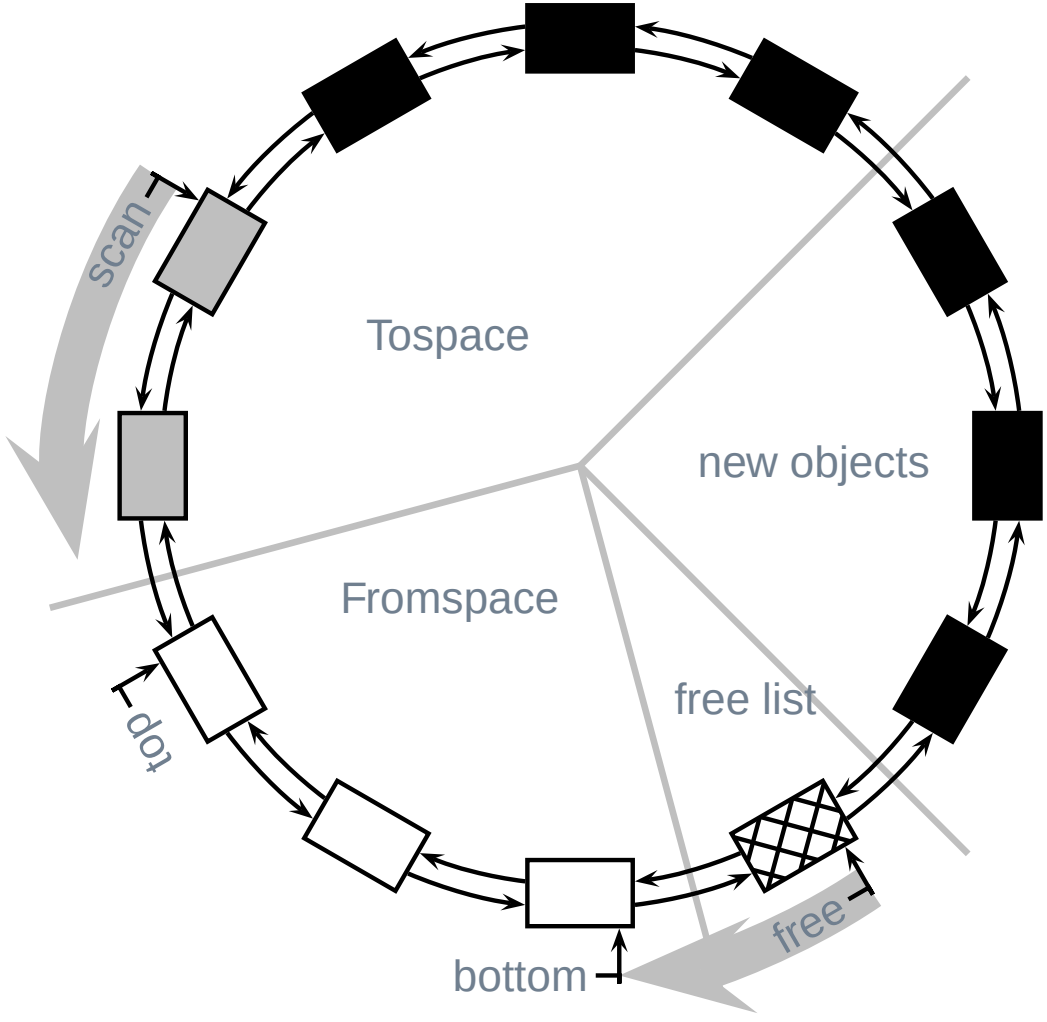
Baker's Treadmill



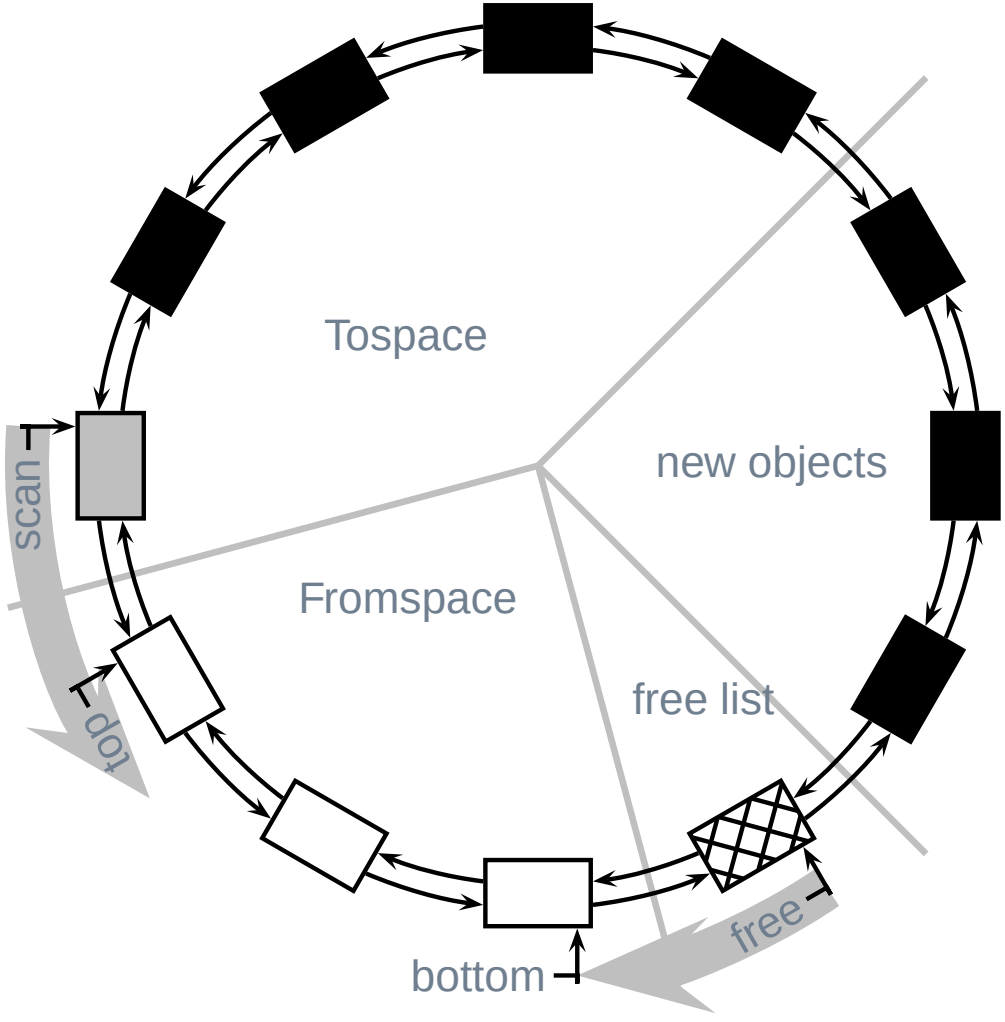
Baker's Treadmill



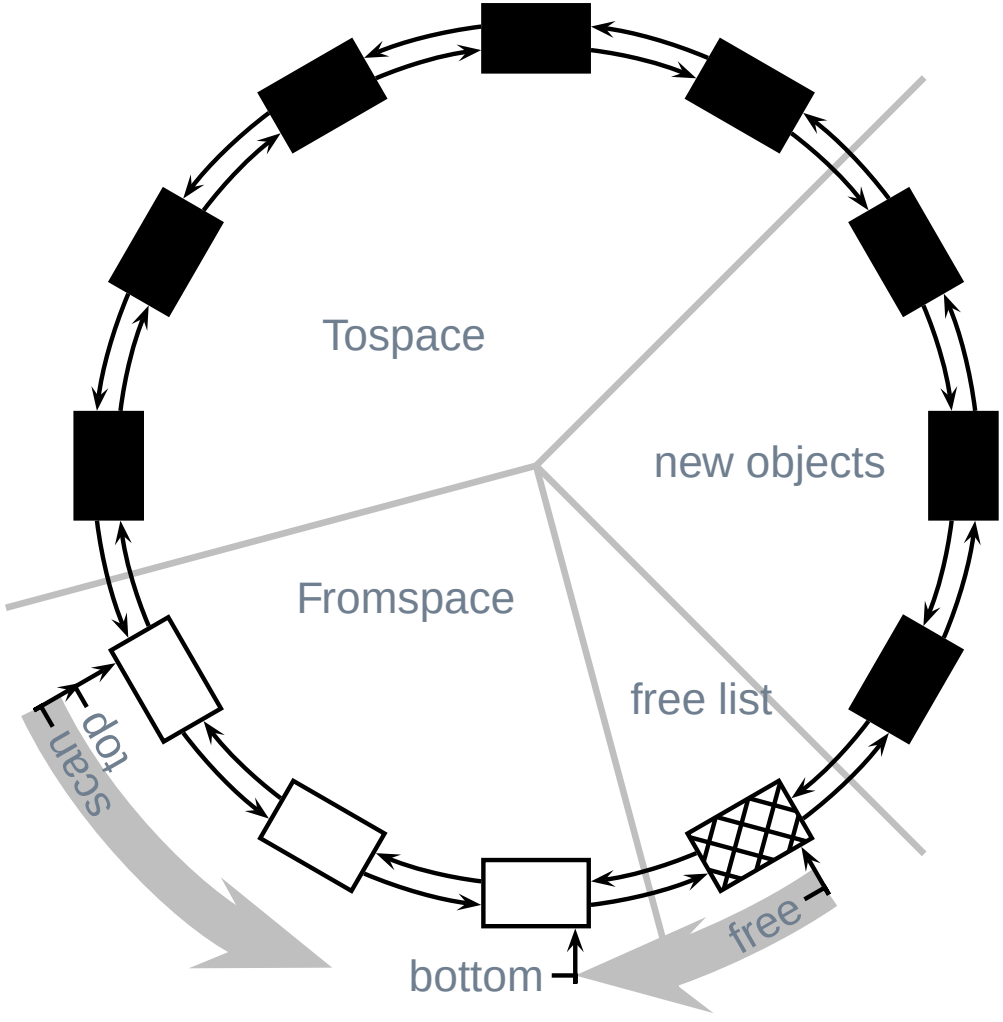
Baker's Treadmill



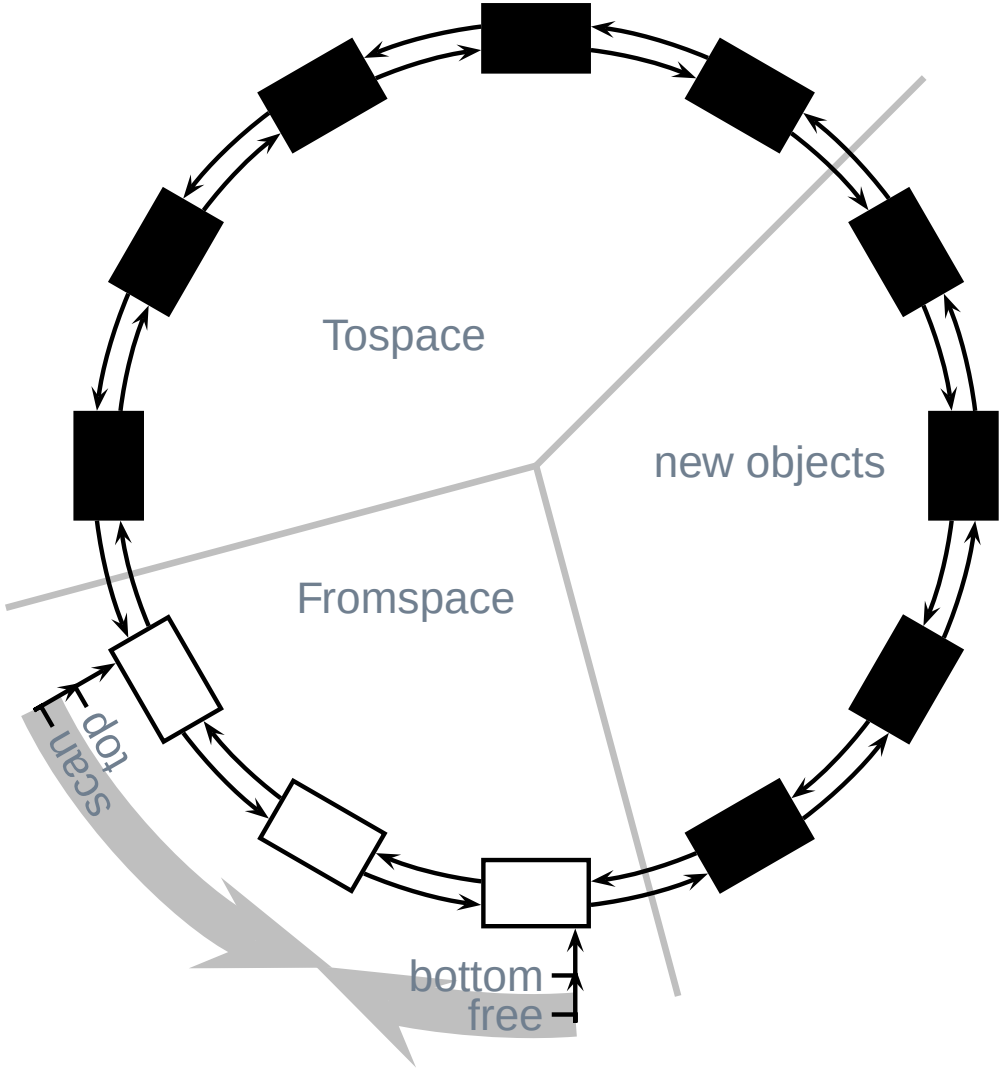
Baker's Treadmill



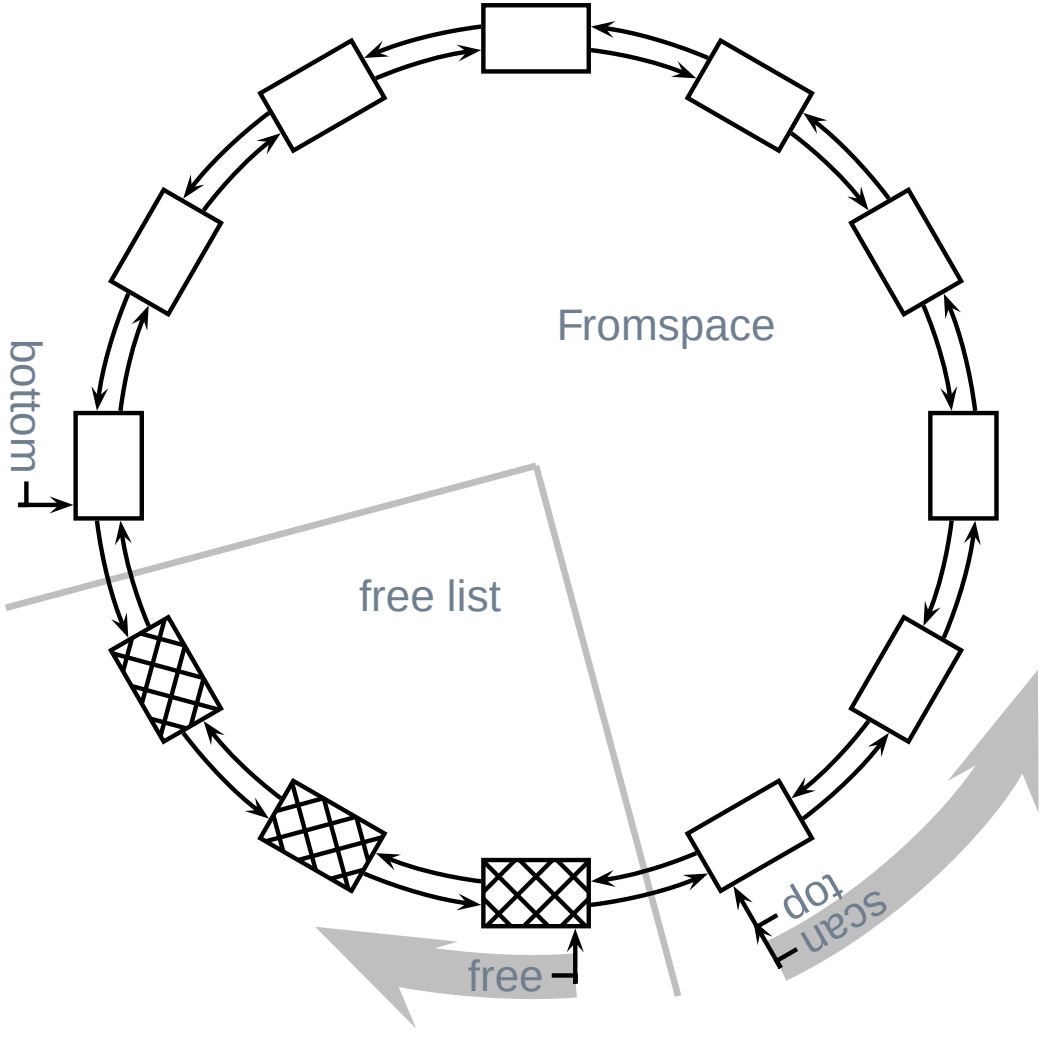
Baker's Treadmill



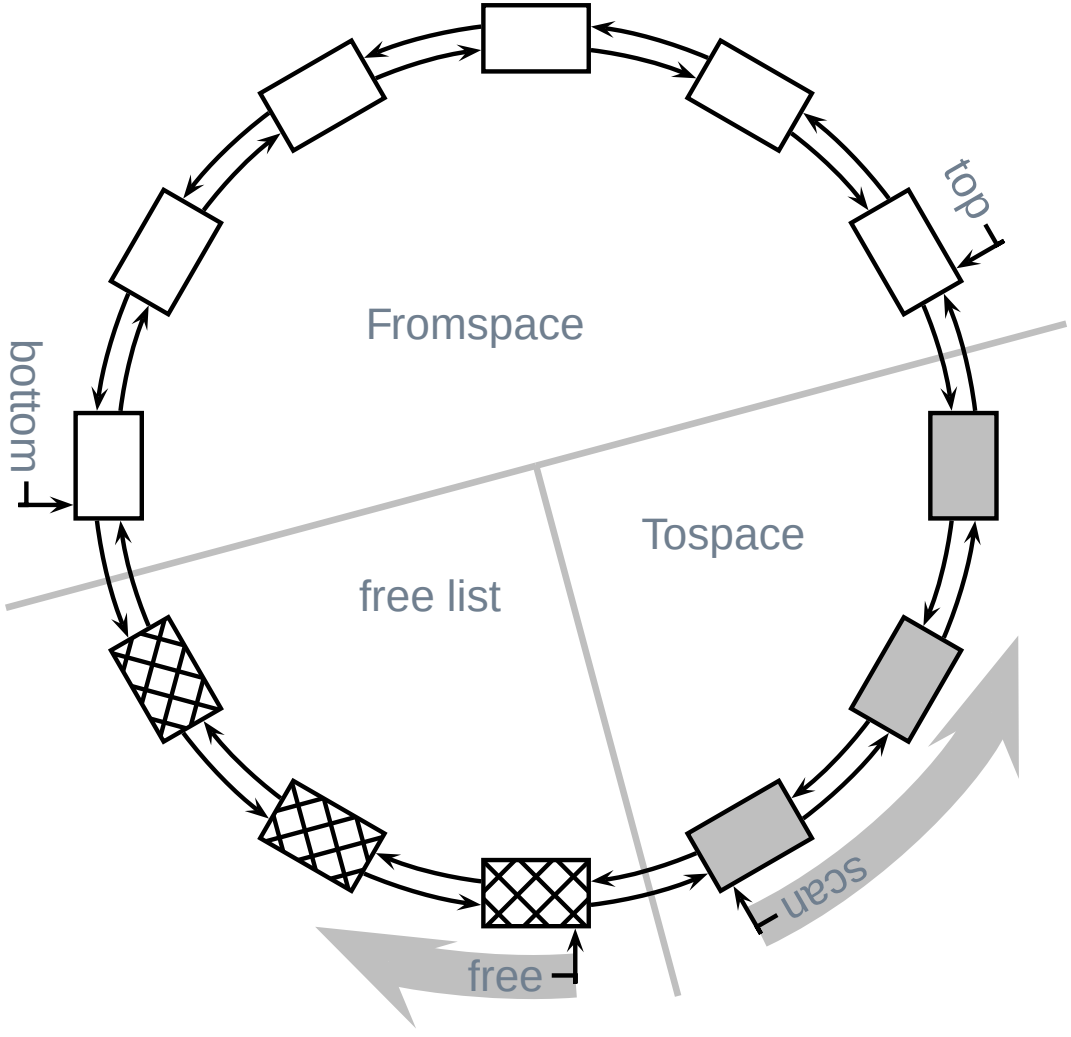
Baker's Treadmill



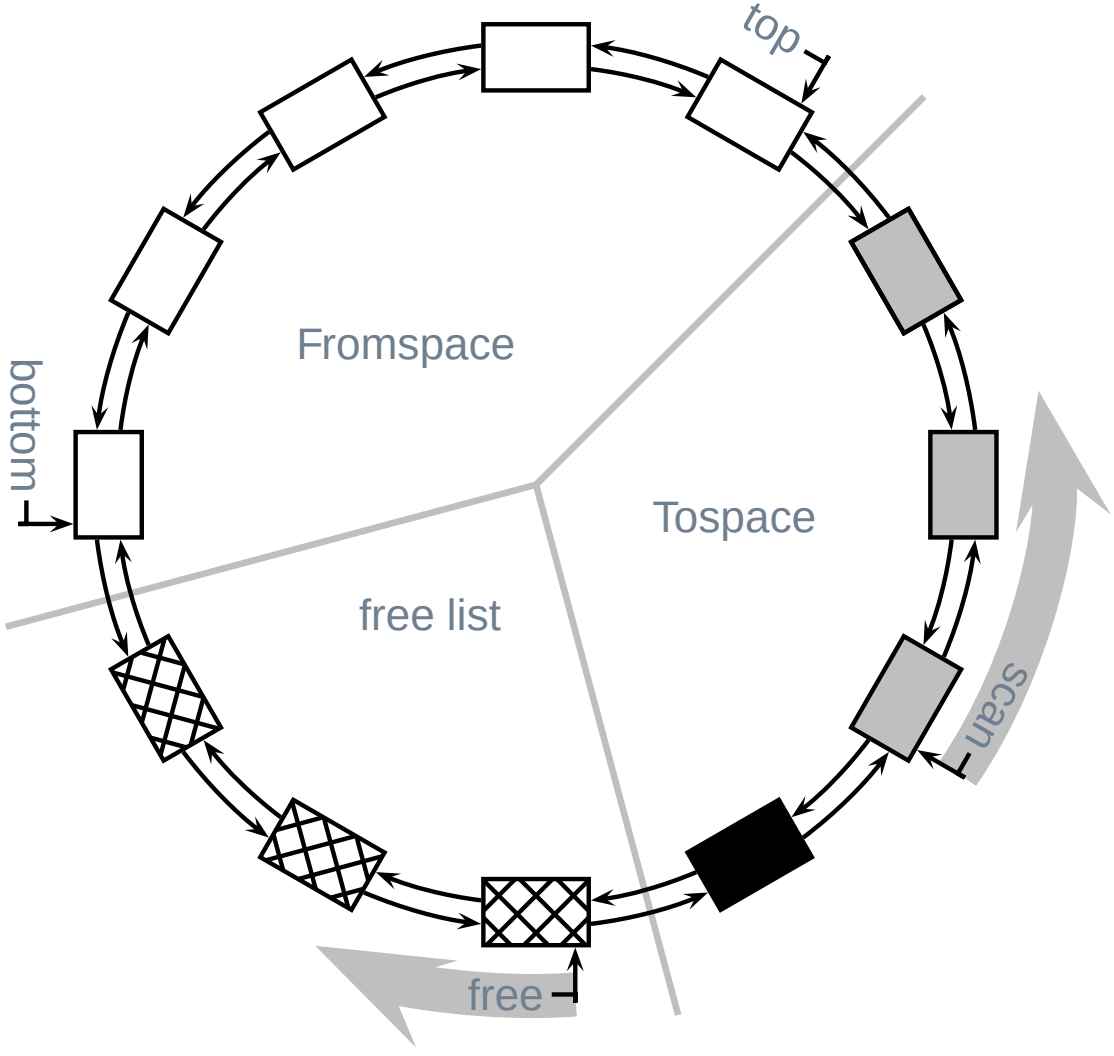
Baker's Treadmill



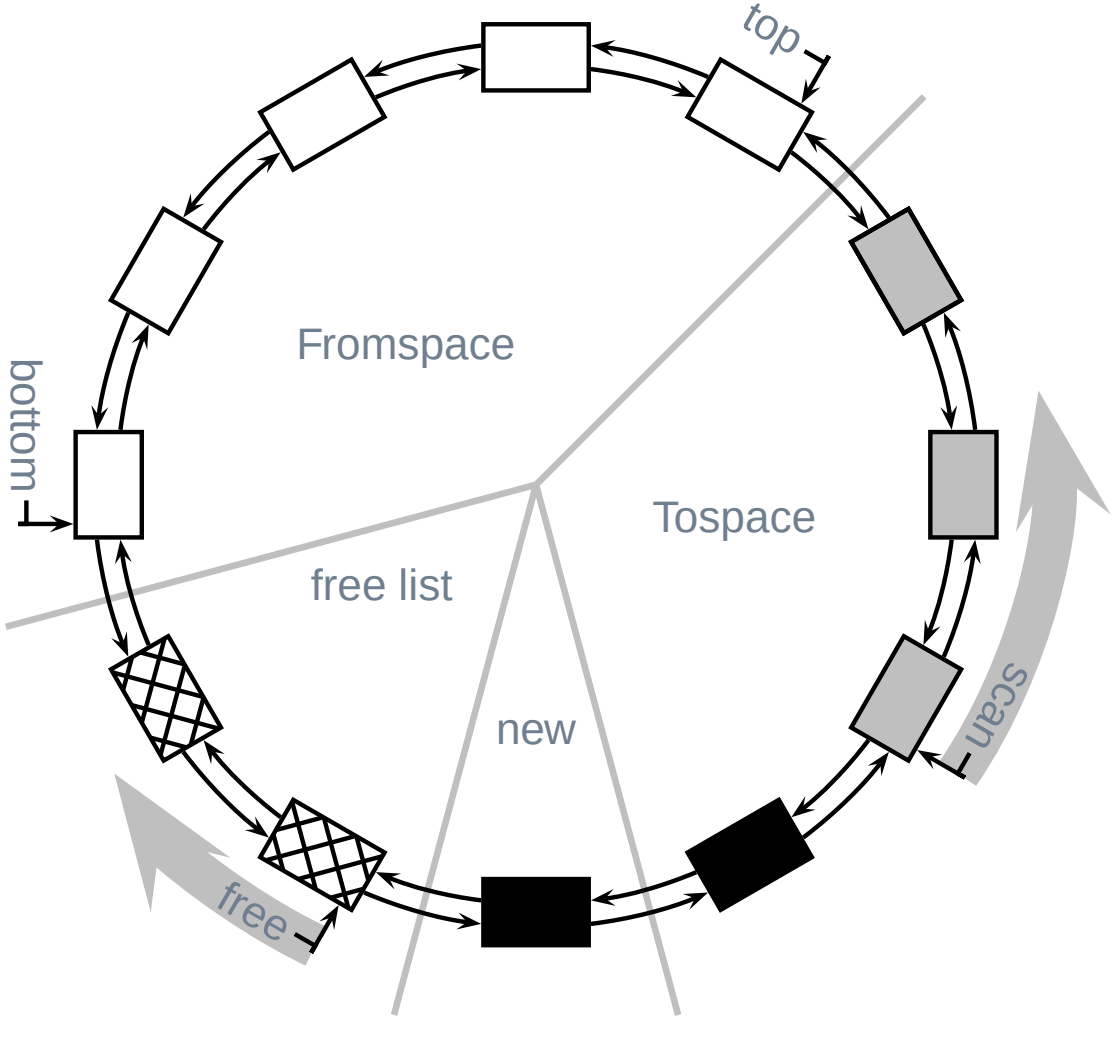
Baker's Treadmill



Baker's Treadmill



Baker's Treadmill



Advantages

- compared to incremental mark/sweep

Advantages

- compared to incremental mark/sweep
  - easy allocation

Advantages

- compared to incremental mark/sweep
  - easy allocation
  - easy to “blacken” object

Advantages

- compared to incremental mark/sweep
  - easy allocation
  - easy to “blacken” object
  - easy flip

Advantages

- compared to incremental mark/sweep
  - easy allocation
  - easy to “blacken” object
  - easy flip
- compared to copying collector

Advantages

- compared to incremental mark/sweep
  - easy allocation
  - easy to “blacken” object
  - easy flip
- compared to copying collector
  - write barrier

Advantages

- compared to incremental mark/sweep
  - easy allocation
  - easy to “blacken” object
  - easy flip
- compared to copying collector
  - write barrier
  - equal or better memory utilization

Advantages

- compared to incremental mark/sweep
  - easy allocation
  - easy to “blacken” object
  - easy flip
- compared to copying collector
  - write barrier
  - equal or better memory utilization
- compared to both

Advantages

- compared to incremental mark/sweep
  - easy allocation
  - easy to “blacken” object
  - easy flip
- compared to copying collector
  - write barrier
  - equal or better memory utilization
- compared to both
  - depth first traversal for free

Problems

- heterogenous objects

Problems

- heterogenous objects
  - multiple Treadmills

Problems

- heterogenous objects
  - multiple Treadmills
- memory fragmentation

Improvements

- one Treadmill for big objects

Improvements

- one Treadmill for big objects
- seperate Treadmills for small objects

Improvements

- one Treadmill for big objects
- seperate Treadmills for small objects
- free page migration between Treadmills

Improvements

- one Treadmill for big objects
- seperate Treadmills for small objects
- free page migration between Treadmills
- page-wise collection for fast migration

Improvements

- one Treadmill for big objects
- seperate Treadmills for small objects
- free page migration between Treadmills
- page-wise collection for fast migration
- free page remapping for large allocations

Improvements

- one Treadmill for big objects
- seperate Treadmills for small objects
- free page migration between Treadmills
- page-wise collection for fast migration
- free page remapping for large allocations
- page-filling allocation

Questions?

Thanks for your attention!