Fast startup and low latency: pick two

Denys Shabalin and Lukas Kellenberger

EPFL

Scala Native

- Announced a year ago with a first prototype at ScalaDays ’16 in New York.

- An ahead-of-time compiler for Scala build on top of the LLVM compiler infrastructure.

- Developed by EPFL and Scala Center.

Scala Native

- As of today:
  - 55 contributors
  - 383 pull requests closed
  - 246 issues fixed

Scala Native

Thanks to everyone who contributed!

Guillaume Massé, Martin Duhem, Hugo Kapp, Hiroyoshi Takahashi, Jonas Fonseca, Lukas Kellenberger, Francois Bertrand, AndreaTP, Eric K Richardson, Marius B. Kotsbak, Kota Mizushima, Łukasz Indykiewicz, Timothy Klim, Paweł Batko, Shunsuke Otani, Nick Pavlov, Cedric Viaccoz, Andrzej Sołtysik, Ankit Soni, Hanns Holger Rutz, Kamil Tomala, Philipp Dörfler, Simon Ochsenreither, Zack Powers, Musabilal, Hubert Plociniczak, Mike Samsonov, Greg Dorrell, Pablo Guerrero, Florian Duraffourg, Alex Dupre, Ragavendar Ramamurthi, Richard Whaling, Roman Zoller, Ruben Berenguel Montoro, Saleem Ansari, Sam Halliday, Felix Mulder, Ragnr, Stefan Ollinger, The Gitter Badger, Tim Nieradzik, Brad Rathke, Viacheslav Blinov, Vincent Munier, Remi, Alexey Kutepov, Adam Voss, Gregor-i, Gute-ist-tot, Kenji Yoshida, Joseph Price, Jentsch, Ignat Loskutov, Martin Mauch

Road towards 0.1

(March 14, 2017)

Goals for 0.1

- All Scala language features are supported
- Sbt integration is sufficient to build and publish existing cross-platform projects
- Enough core libraries to cover for basic standard library usage

Improving the standard library story in 0.2

(April 26, 2017)

Goals for 0.2

- Support for file i/o from java.io.*
- Support for regex from java.util.regex.*
- Event-loop-based Futures

Laying down the foundation for *better* garbage collection in 0.3

(June 6, 2017)

Goal for 0.3

Better GC #128

Open **ebruchez** opened this issue on May 23, 2016 · 21 comments

ebruchez commented on May 23, 2016

Adding this as a backlog idea.

sgen is Mono's newish garbage collector, introduced to replace the old Boehm. It is a generational gc, and since it works for Mono, I would think it could fit the bill for Scala Native as well.

Here is the source code. It appears to under the MIT license.

👍 5

```
nativeGC := “boehm”
```

Boehm GC

- Conservative garbage collector
- Originally designed for C/C++ environment
- Good starting point for a new language implementation due to it simple GC interface

Conservative GC

- Conservative roots:

  GC doesn't *precisely* know which values on the stack are heap references, but object layout is known.

- Fully conservative:

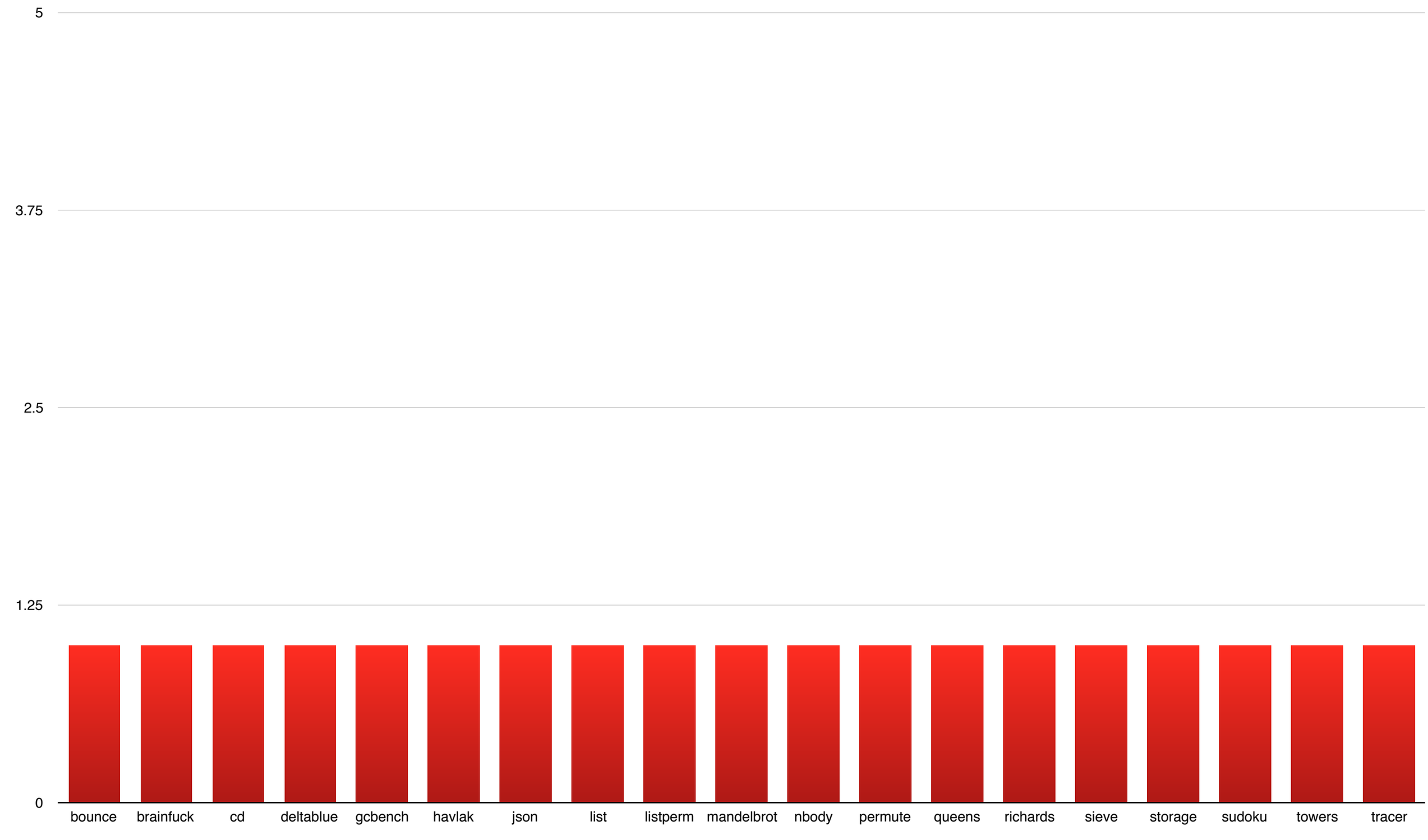
  GC doesn't *precisely* know which values on the stack or heap are references.

Boehm GC

- How fast/slow are we exactly?

- github.com/smarr/are-we-fast-yet

Boehm GC


5
3.75
2.5
1.25
0
bounce
brainfuck
cd
deltablue
gcbench
havlak
json
list
listperm
mandelbrot
nbody
permute
queens
richards
sieve
storage
sudoku
towers
tracer

```
nativeGC := “none”
```

Running without GC

- The simplest form of garbage collection: allocate and never free allocated memory
- Practical for short-lived command-line tools
- Sometimes used in applications with insane requirements for application predictability

From: k...@rational.com (Kent Mitchell)
Subject: Re: Does memory leak?
Date: 1995/03/31
newsgroups: comp.lang.ada

This sparked and interesting memory for me. **I was once working with a customer who was producing on-board software for a missile.** In my analysis of the code, I pointed out that they had a number of problems with storage leaks. Imagine my surprise when the customers chief software engineer said "Of course it leaks". He went on to point out that they had calculated the amount of memory the application would leak in the total possible flight time for the missile and then doubled that number. **They added this much additional memory to the hardware to "support" the leaks. Since the missile will explode when it hits it's target or at the end of it's flight, *the ultimate in garbage collection* is performed without programmer intervention.**

--
Kent Mitchell | One possible reason that things aren't
Technical Consultant | going according to plan is .....
Rational Software Corporation | that there never *was* a plan!

https://groups.google.com/forum/message/raw?msg=comp.lang.ada/E9bNCvDQ12k/1tezW24ZxdAJ

Running without GC

OpenJDK

OpenJDK FAQ
Installing
Contributing
Sponsoring
Developers' Guide

Mailing lists
IRC · Wiki

Bylaws · Census
Legal

JEP Process

search

Source code
Mercurial
Bundles (6)

Groups
(overview)
2D Graphics
Adoption
AWT
Build
Compiler
Conformance
Core Libraries
Governing Board
HotSpot

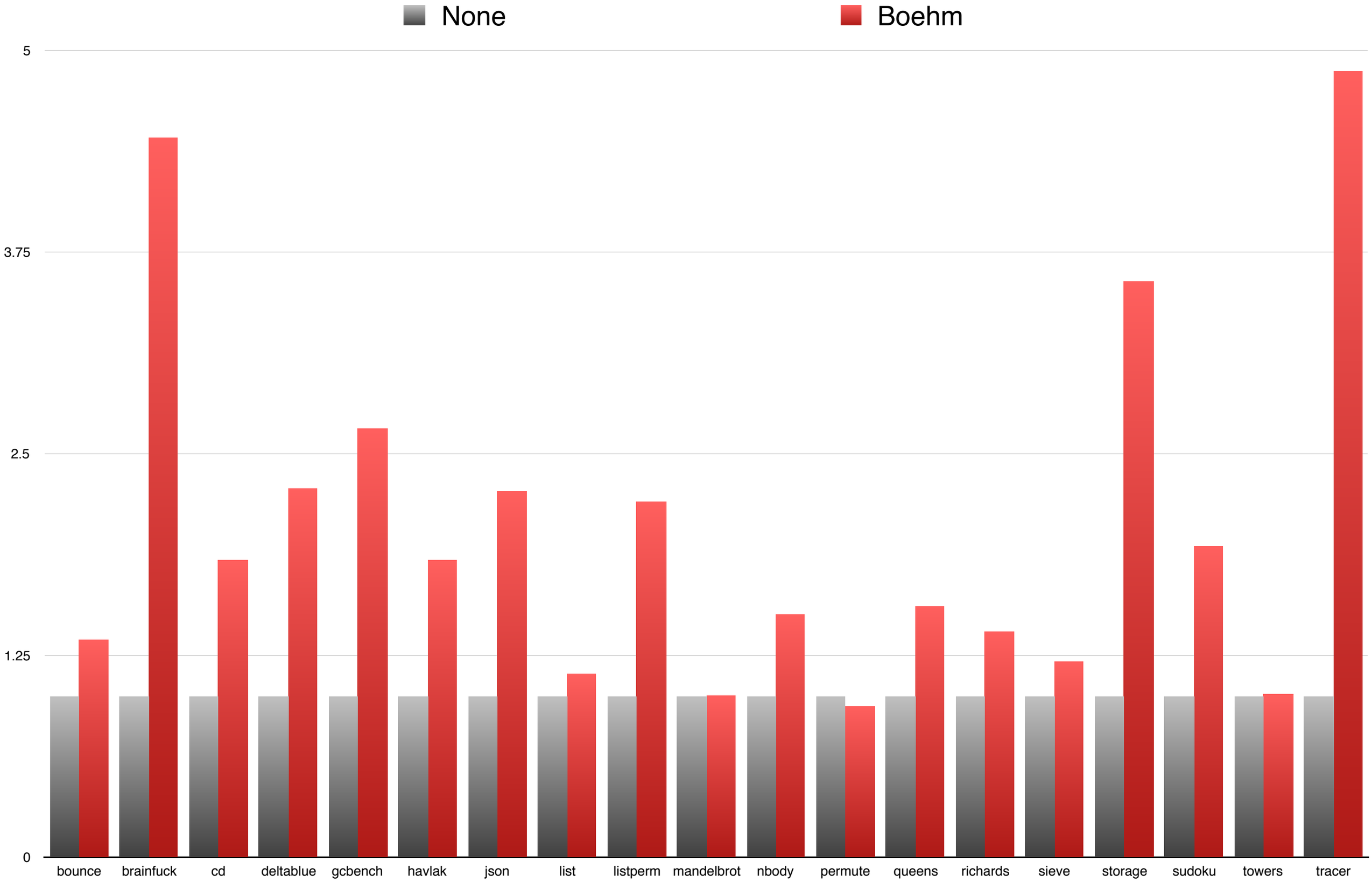
JEP draft: Epsilon GC: The Arbitrarily Low Overhead Garbage (Non-)Collector

Owner	Aleksey Shipilev
Created	2017/02/14 08:23
Updated	2017/04/01 16:29
Type	Feature
Status	Draft
Component	hotspot / gc
Scope	Implementation
Effort	S
Duration	S
Priority	4
Issue	8174901

Summary

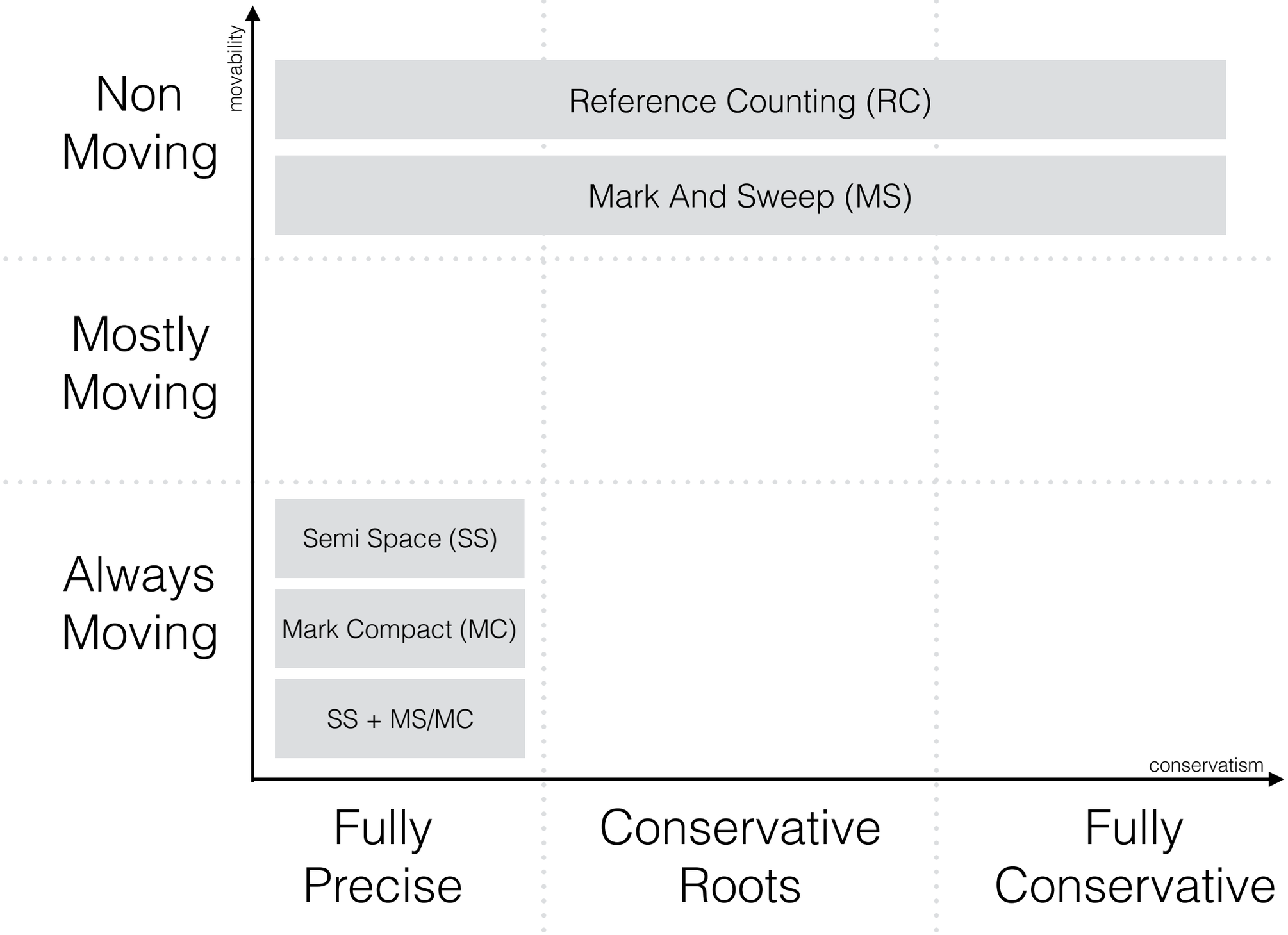
Develop a GC that only handles memory allocation, but does not implement any actual memory reclamation mechanism. Once available Java heap is exhausted, perform the orderly JVM shutdown.

Evaluating cost of GC

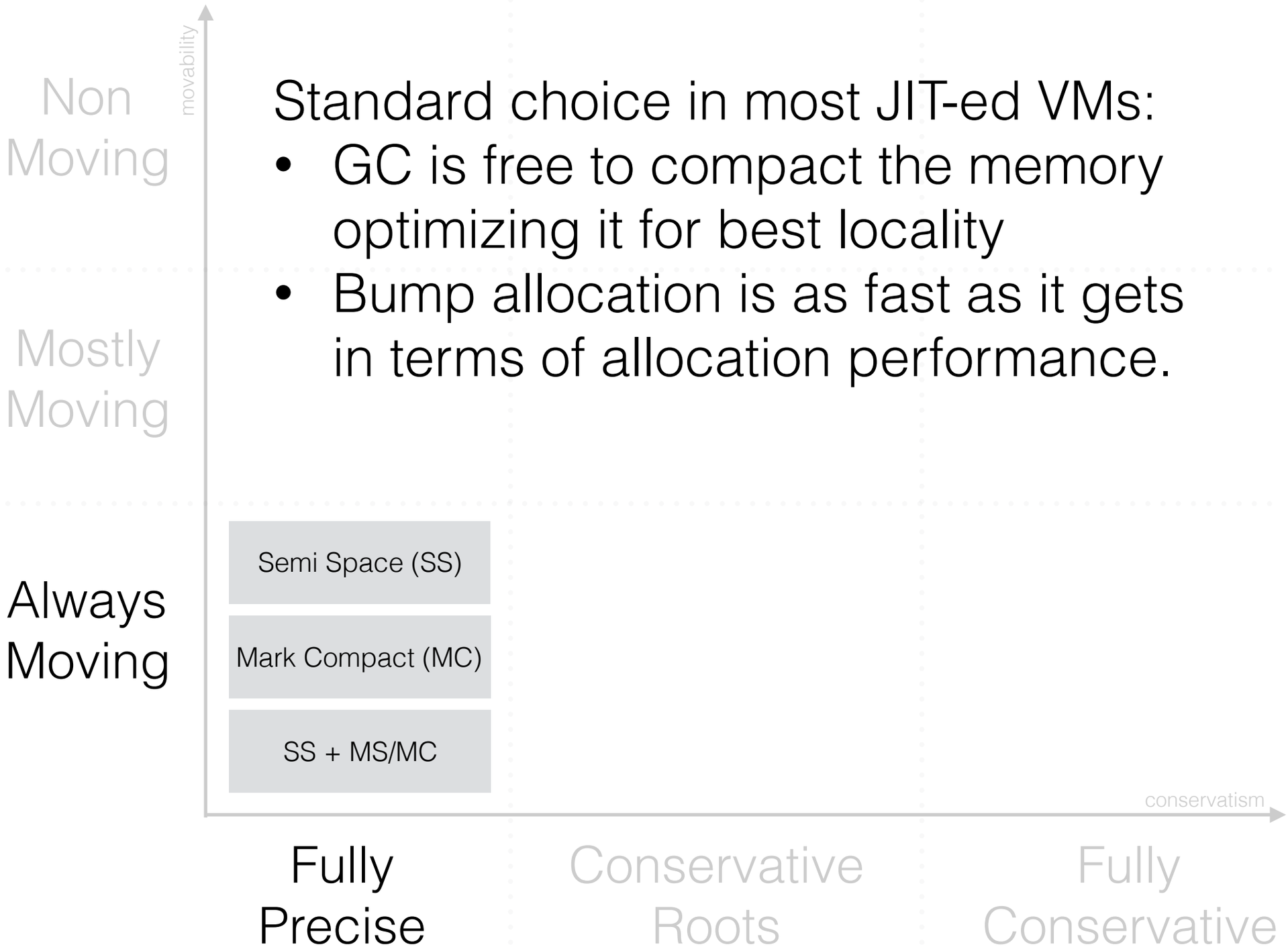


nativeGC := “?“

GCs in the wild



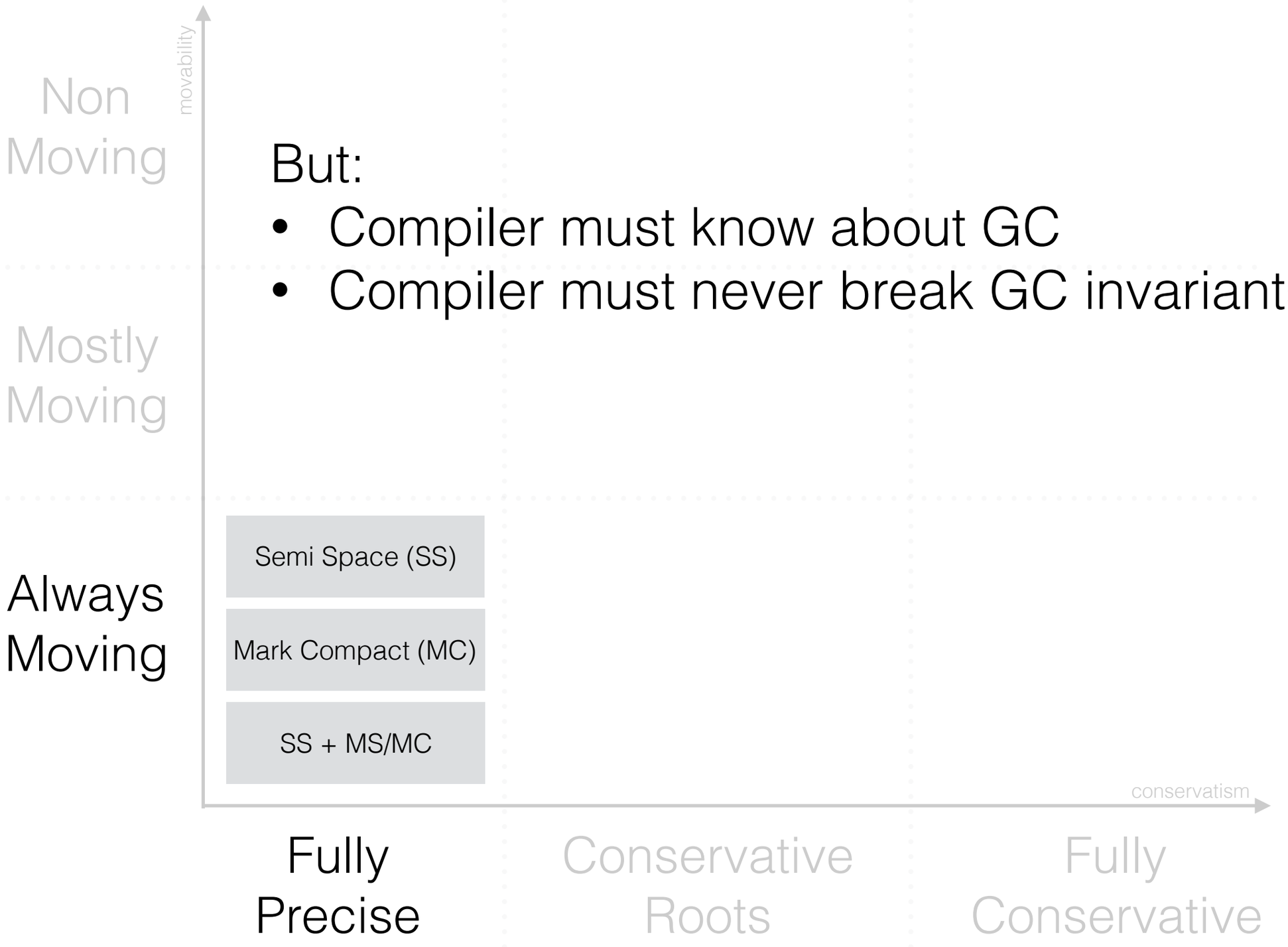
Always Moving GCs



Standard choice in most JIT-ed VMs:

- GC is free to compact the memory optimizing it for best locality
- Bump allocation is as fast as it gets in terms of allocation performance.

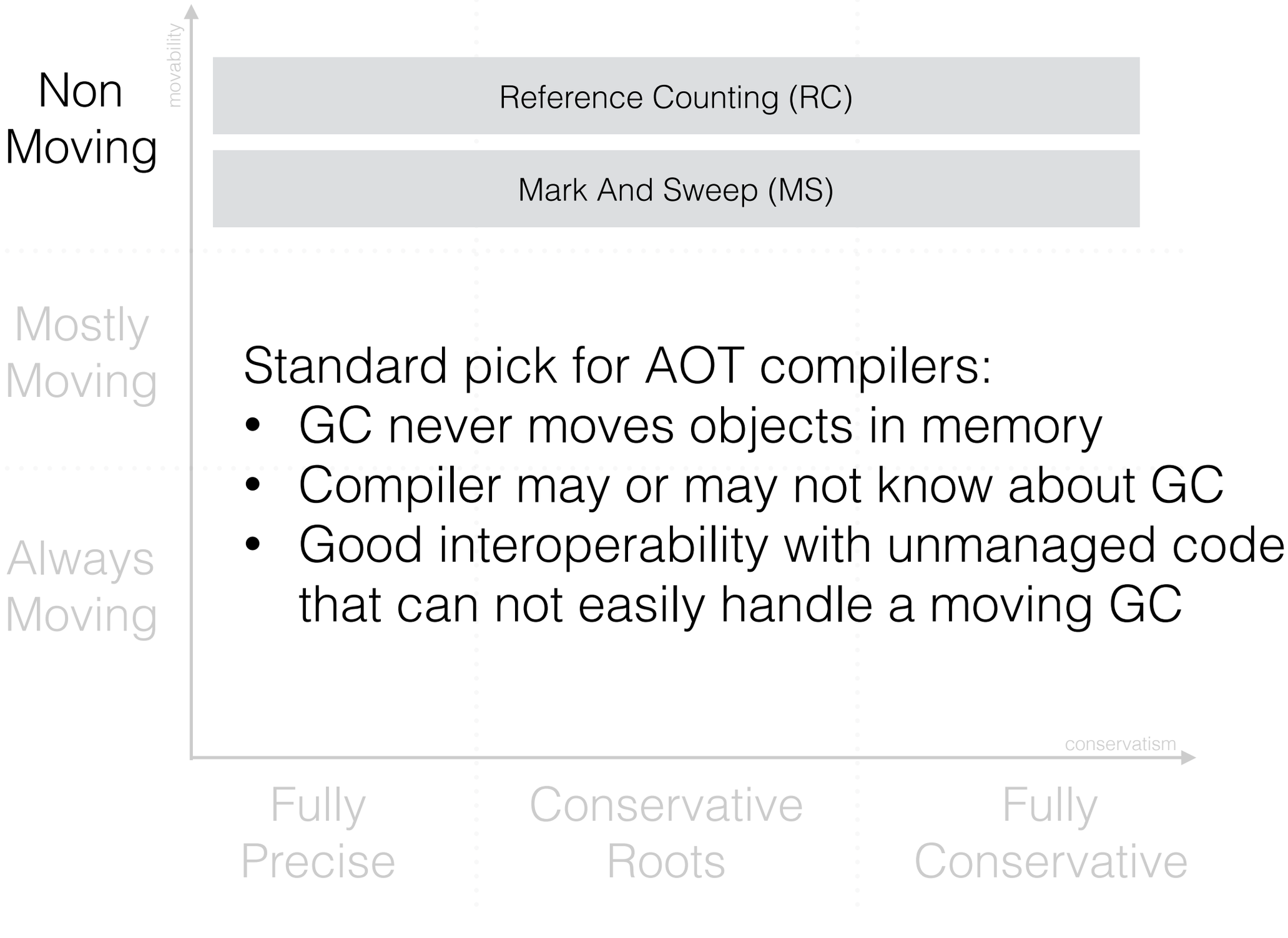
Always Moving GCs



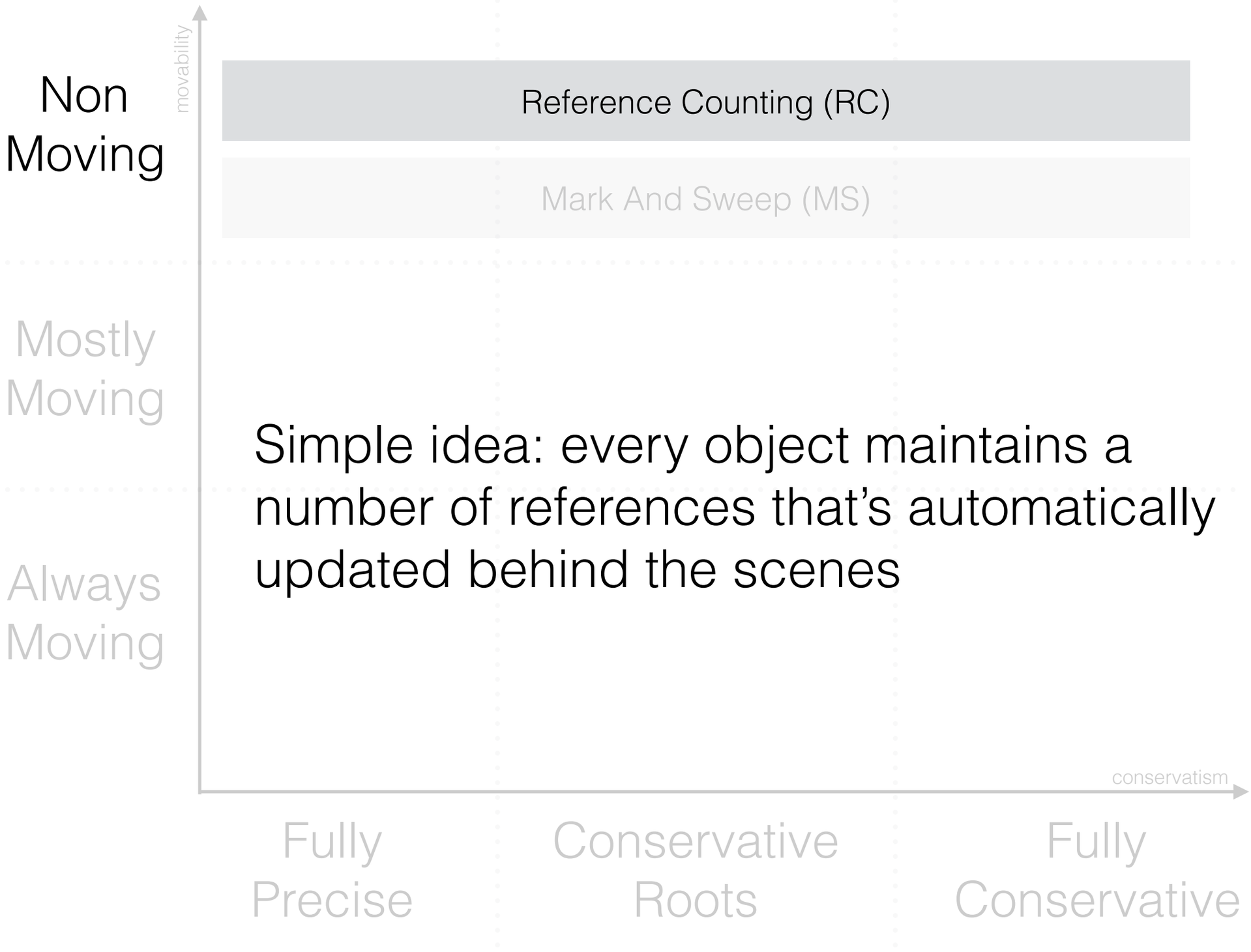
But:

- Compiler must know about GC
- Compiler must never break GC invariants

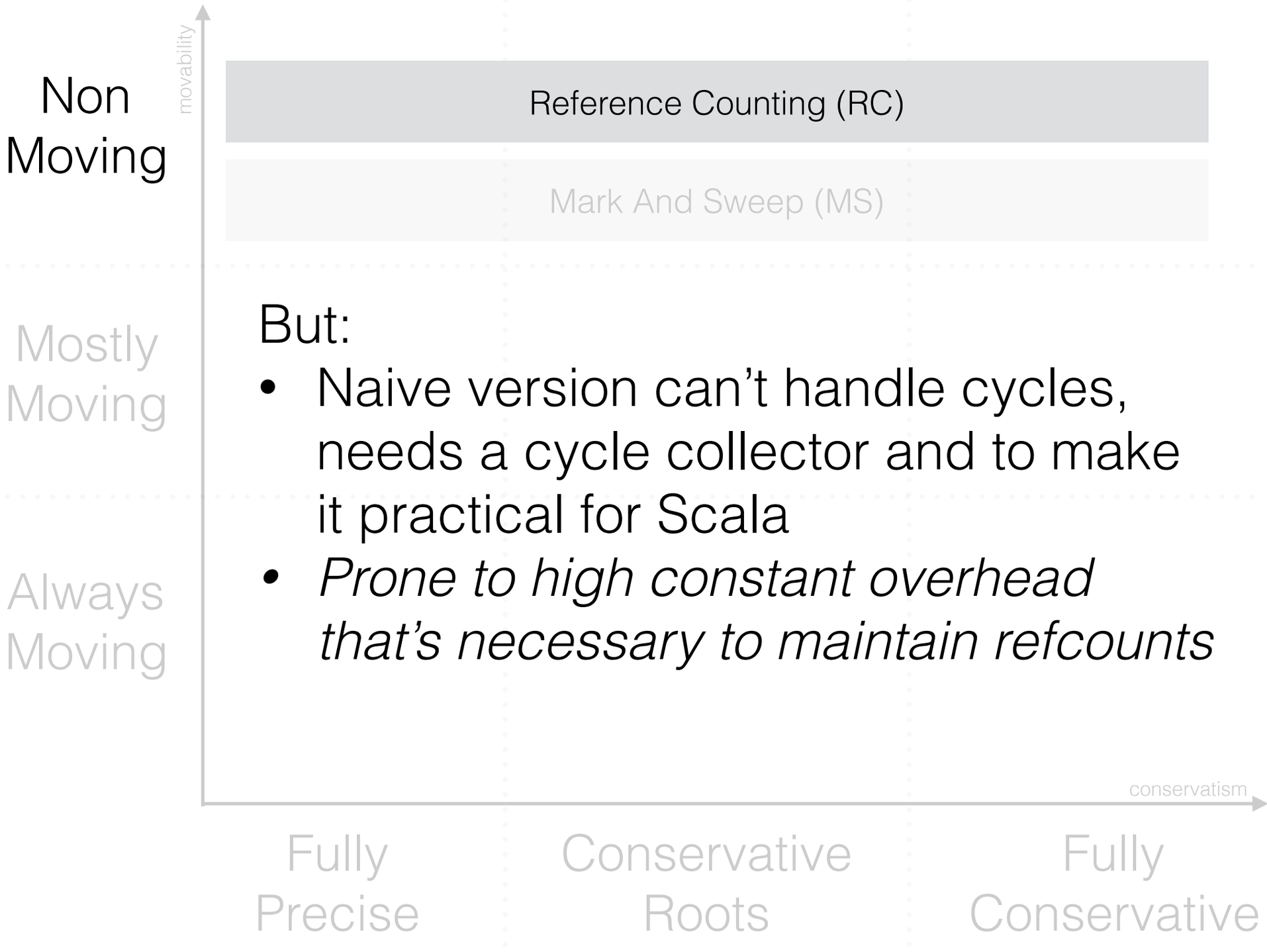
Non-Moving GCs



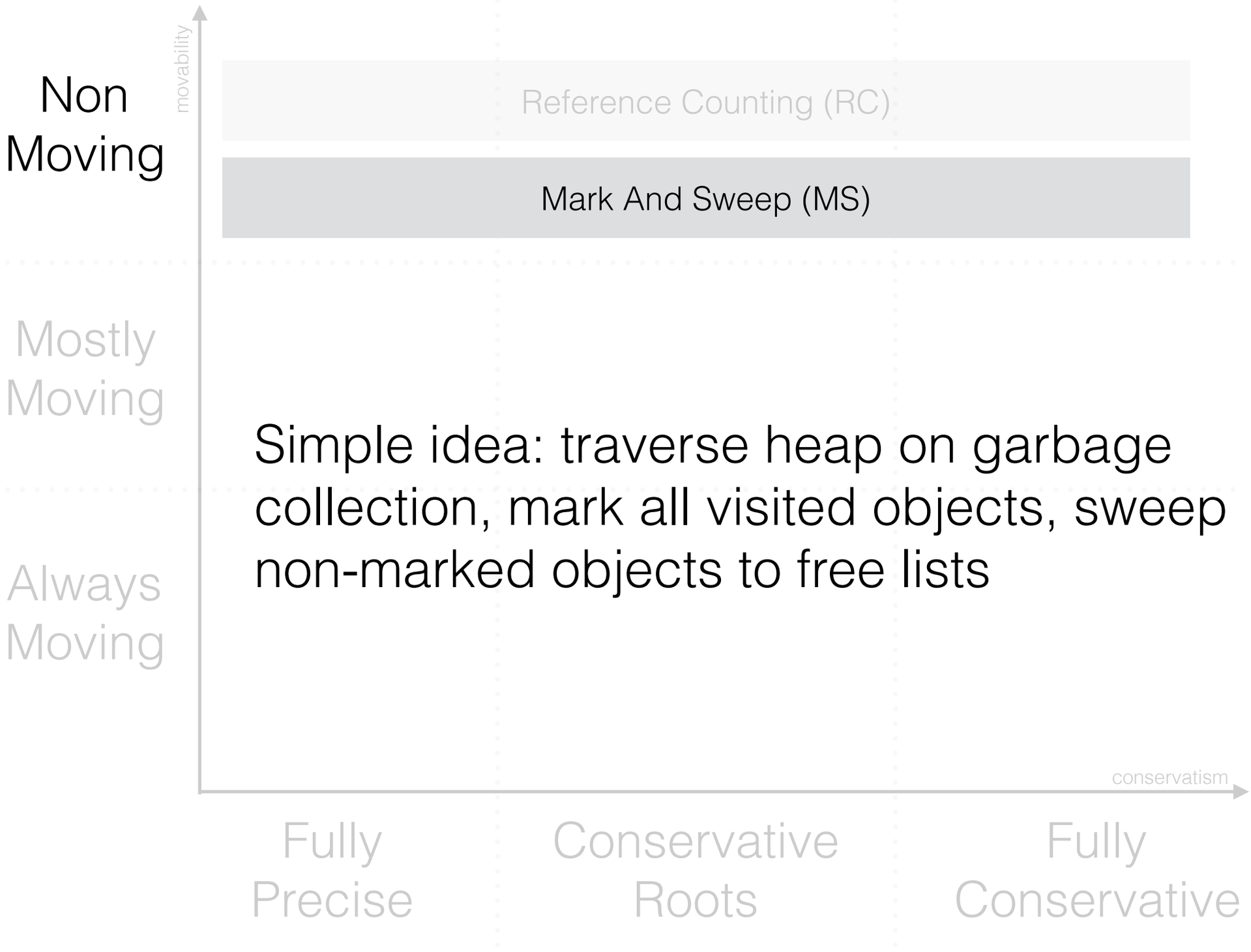
Reference Counting



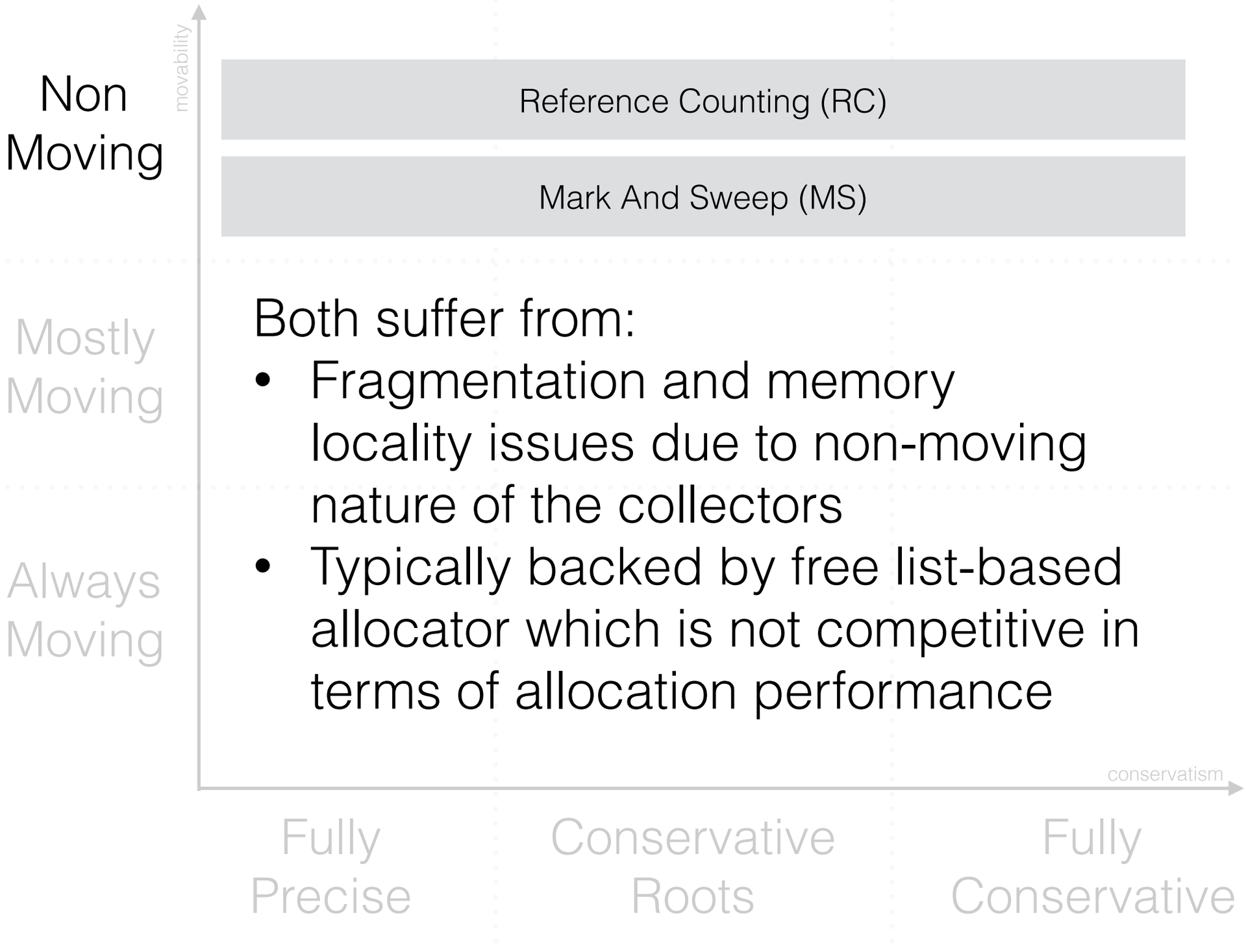
Reference Counting



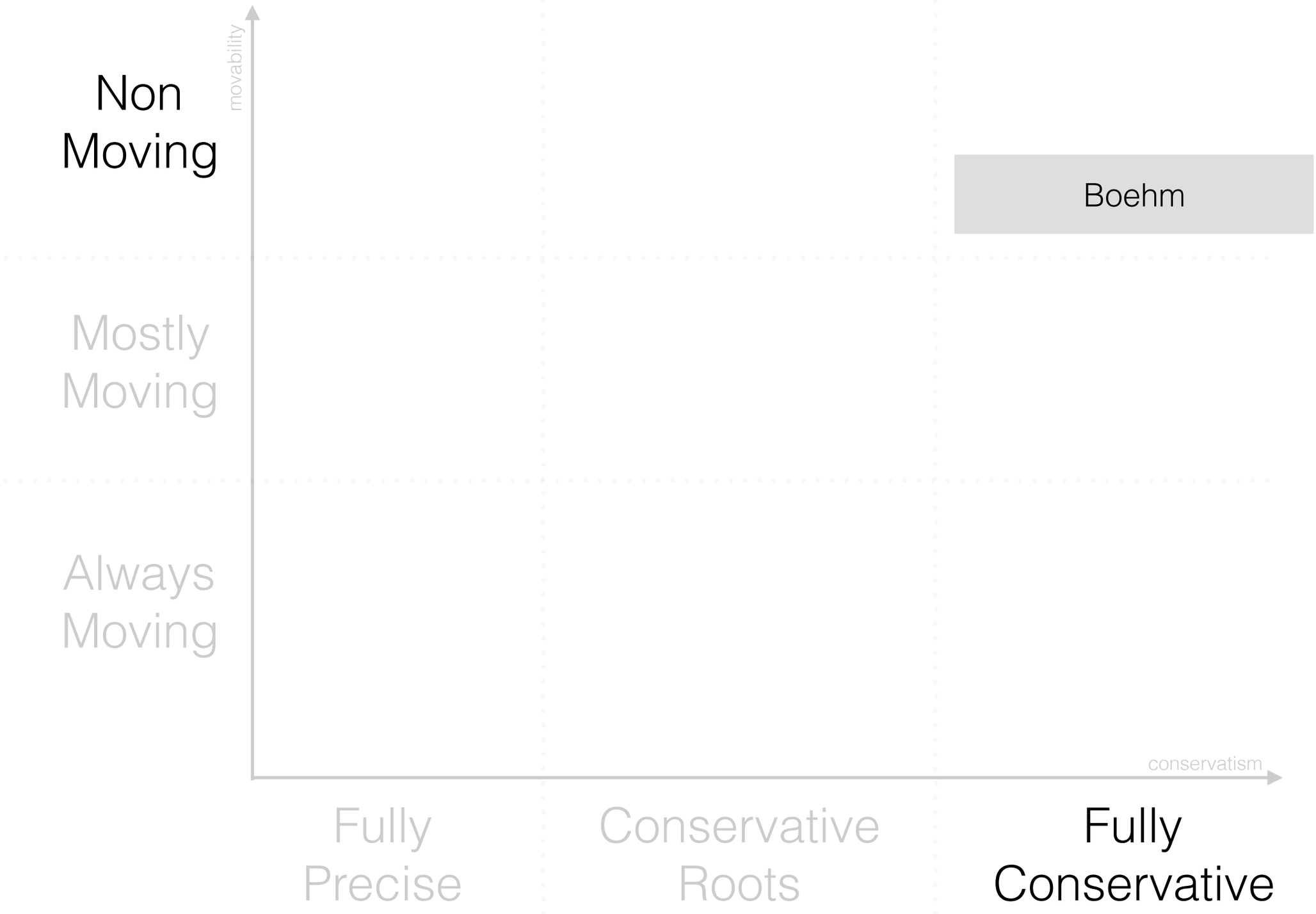
Reference Counting



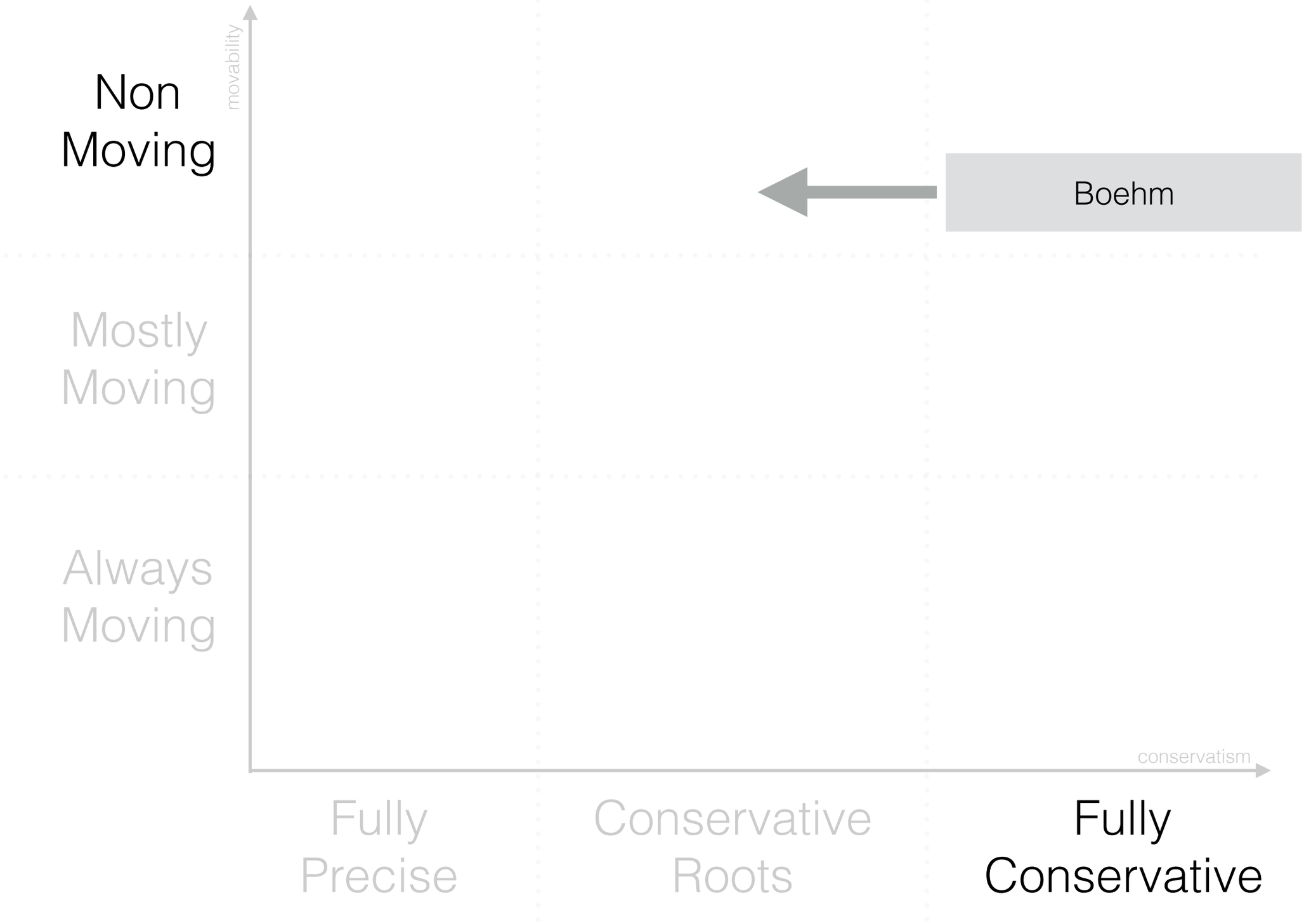
Non Moving GCs



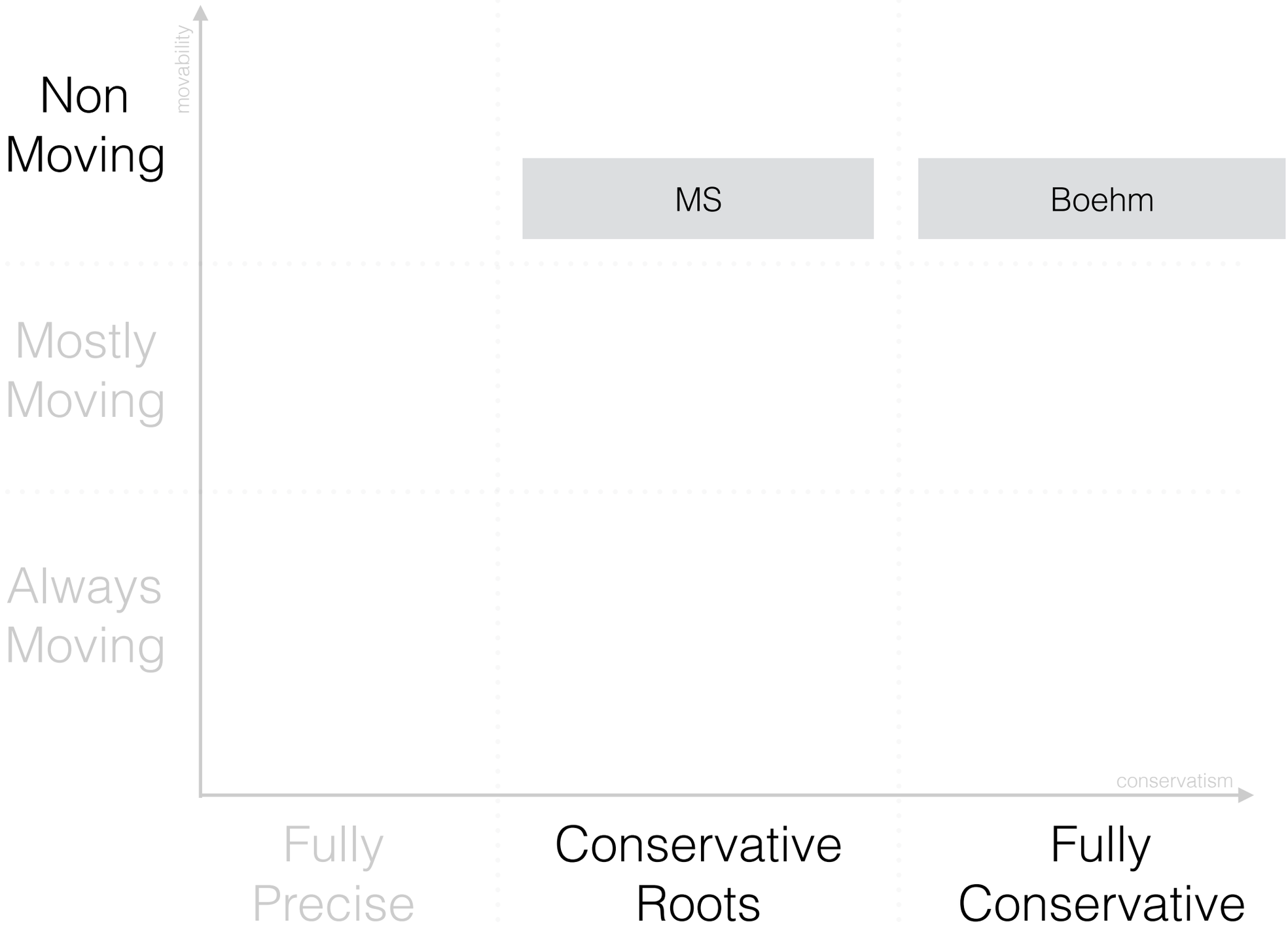
Status quo revisited



Status quo revisited

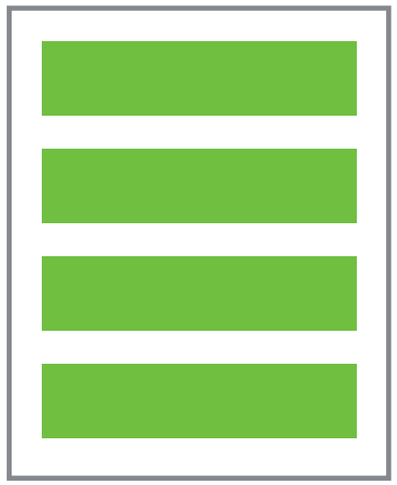


Initial experiments

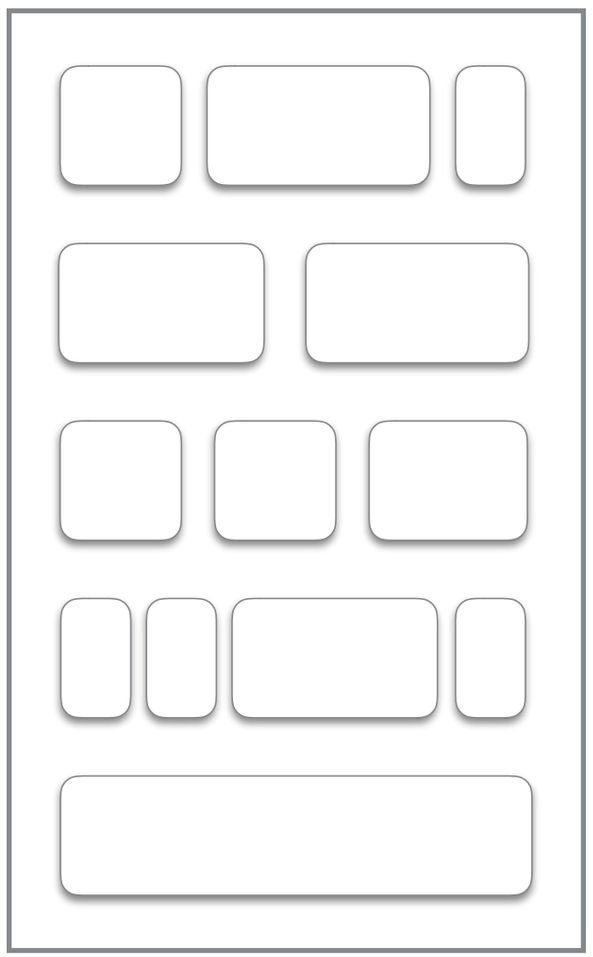


MS

Stack

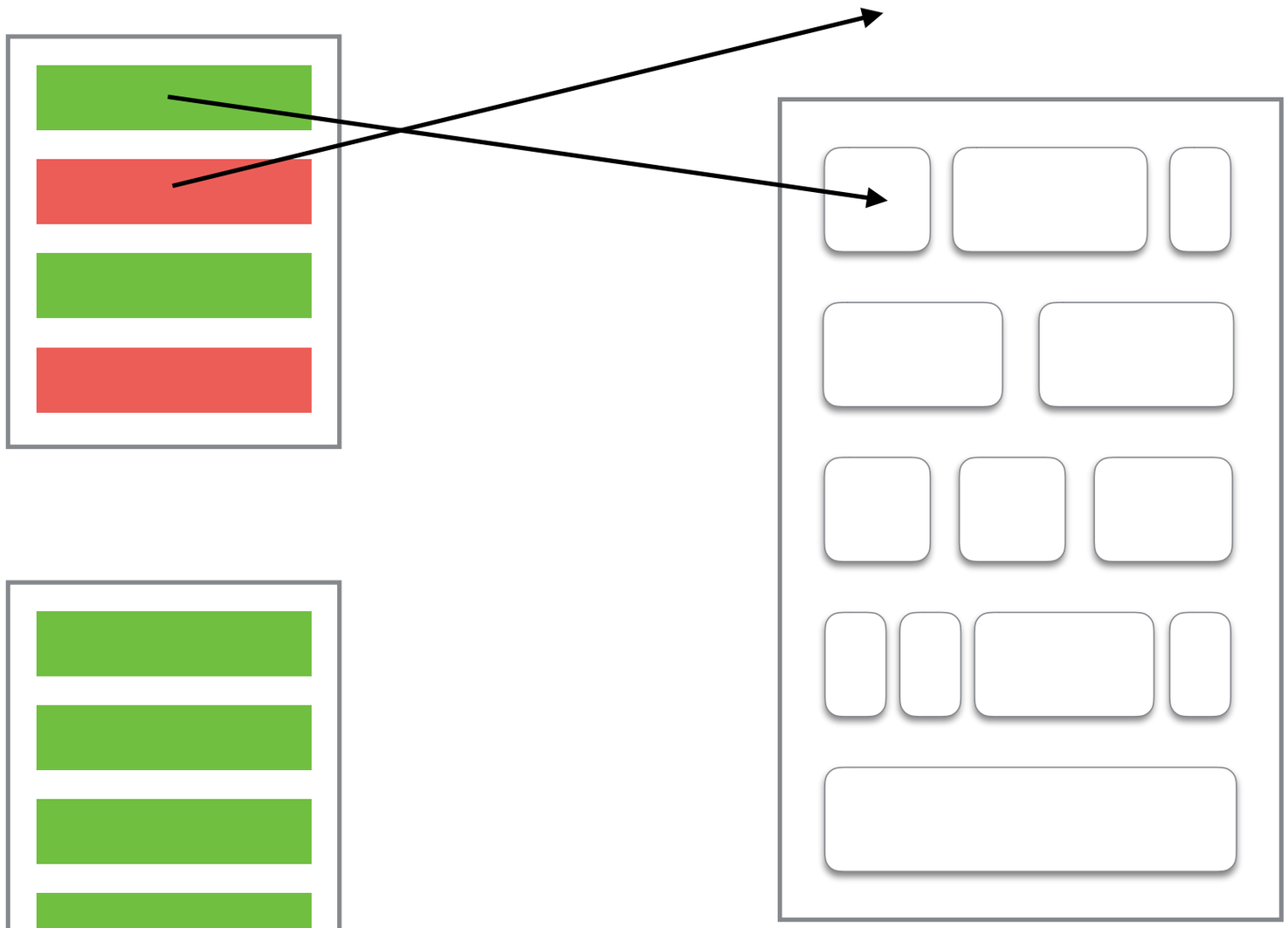


Modules



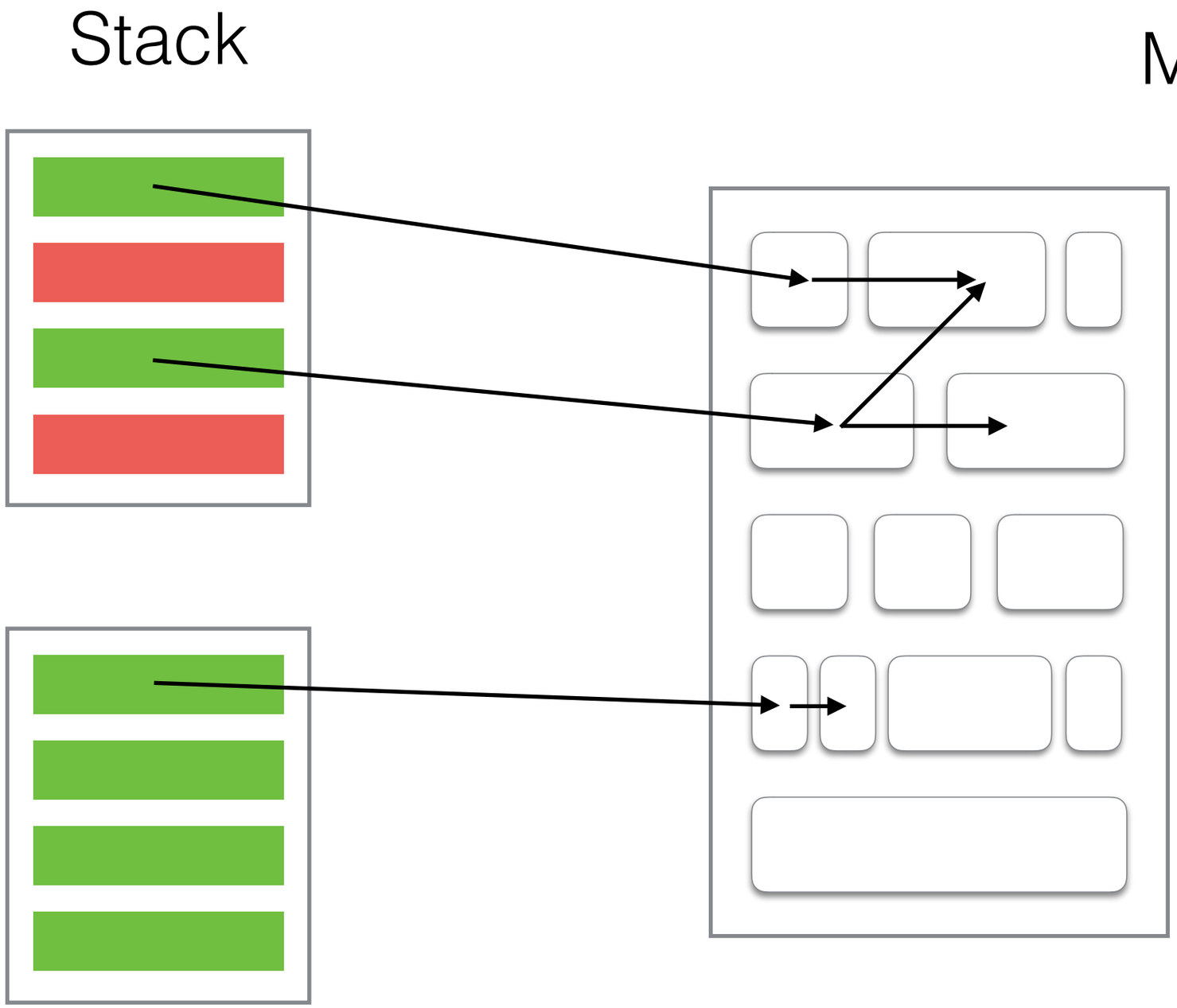
MS

Stack

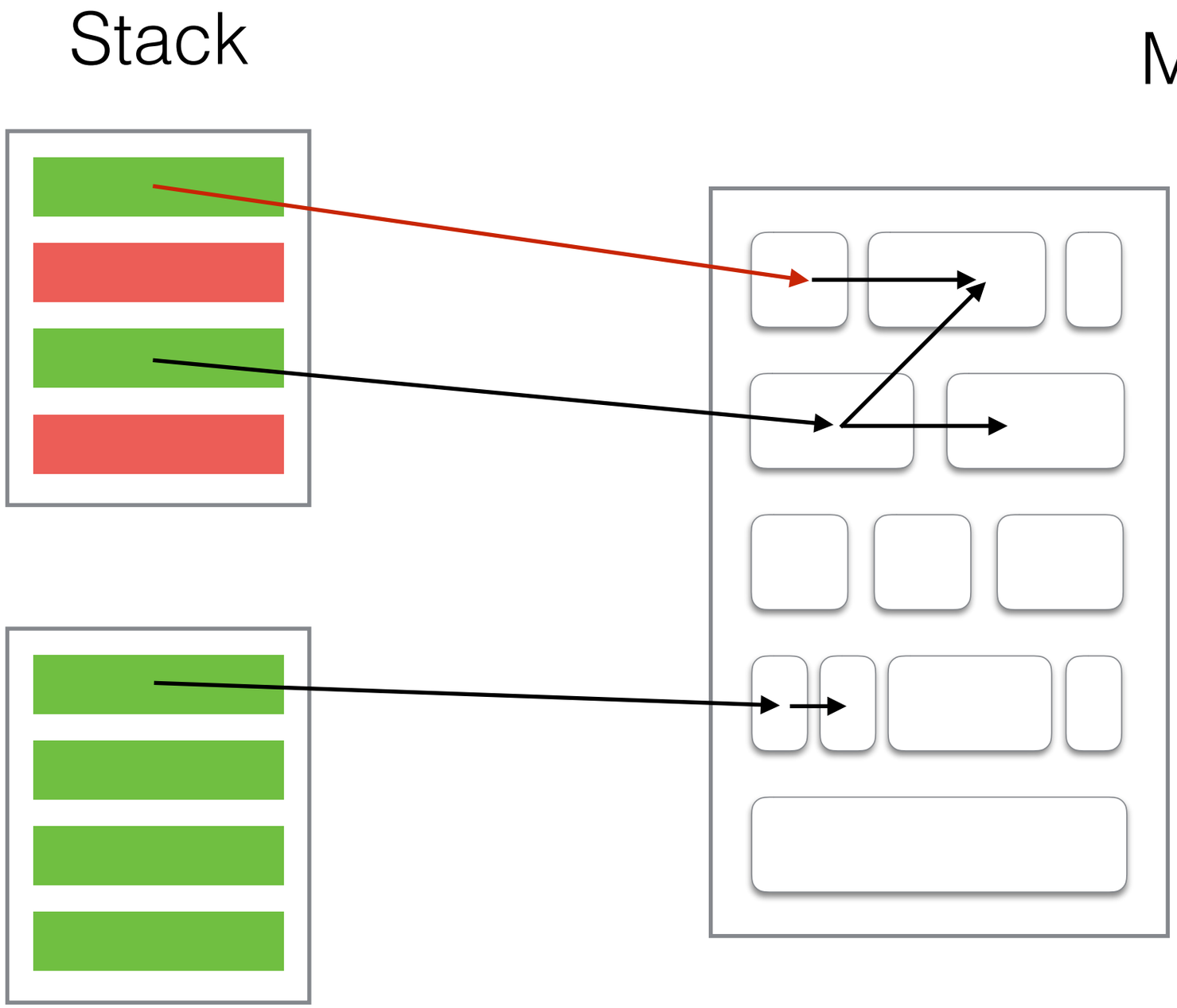


Modules

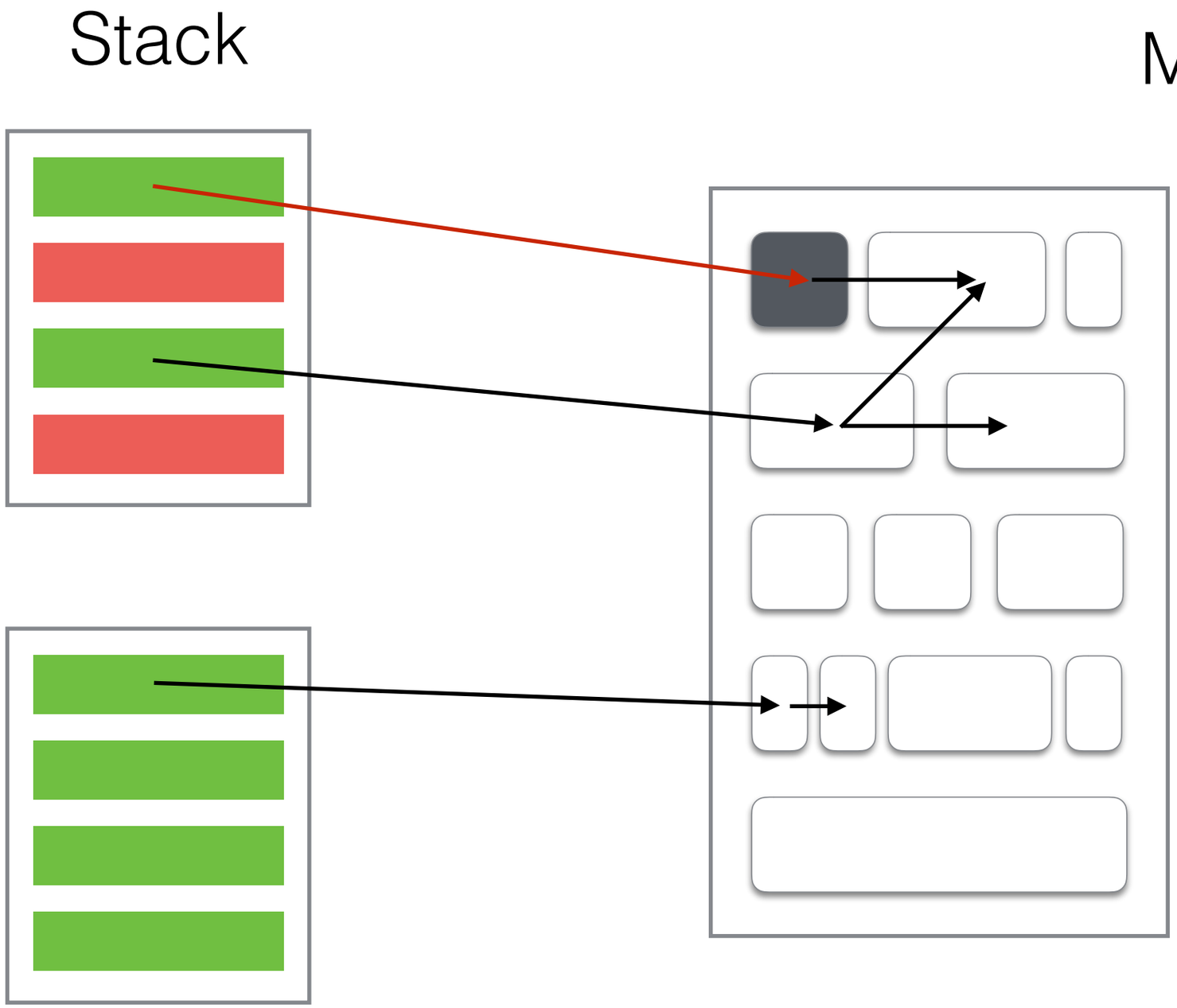
MS



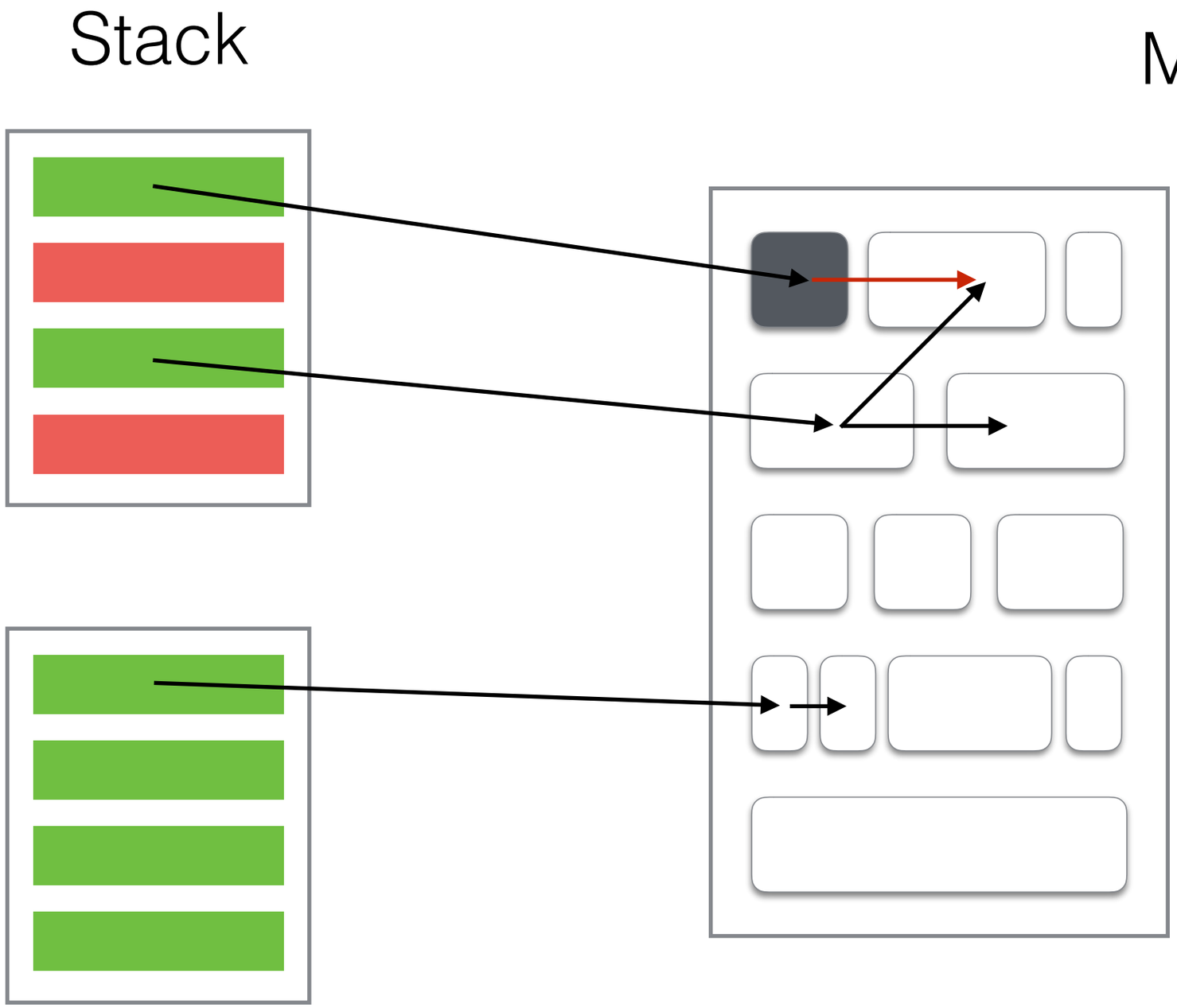
MS



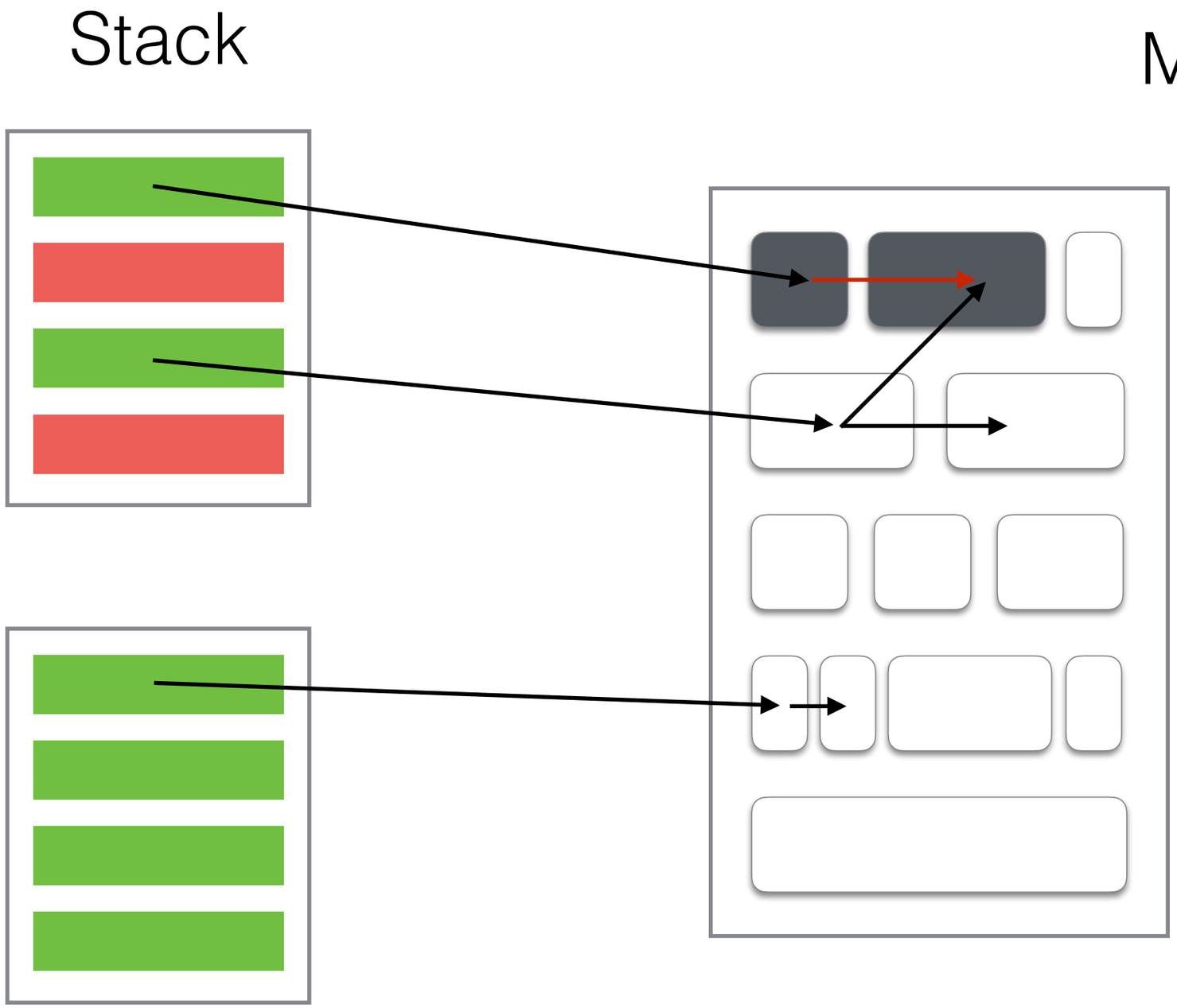
MS



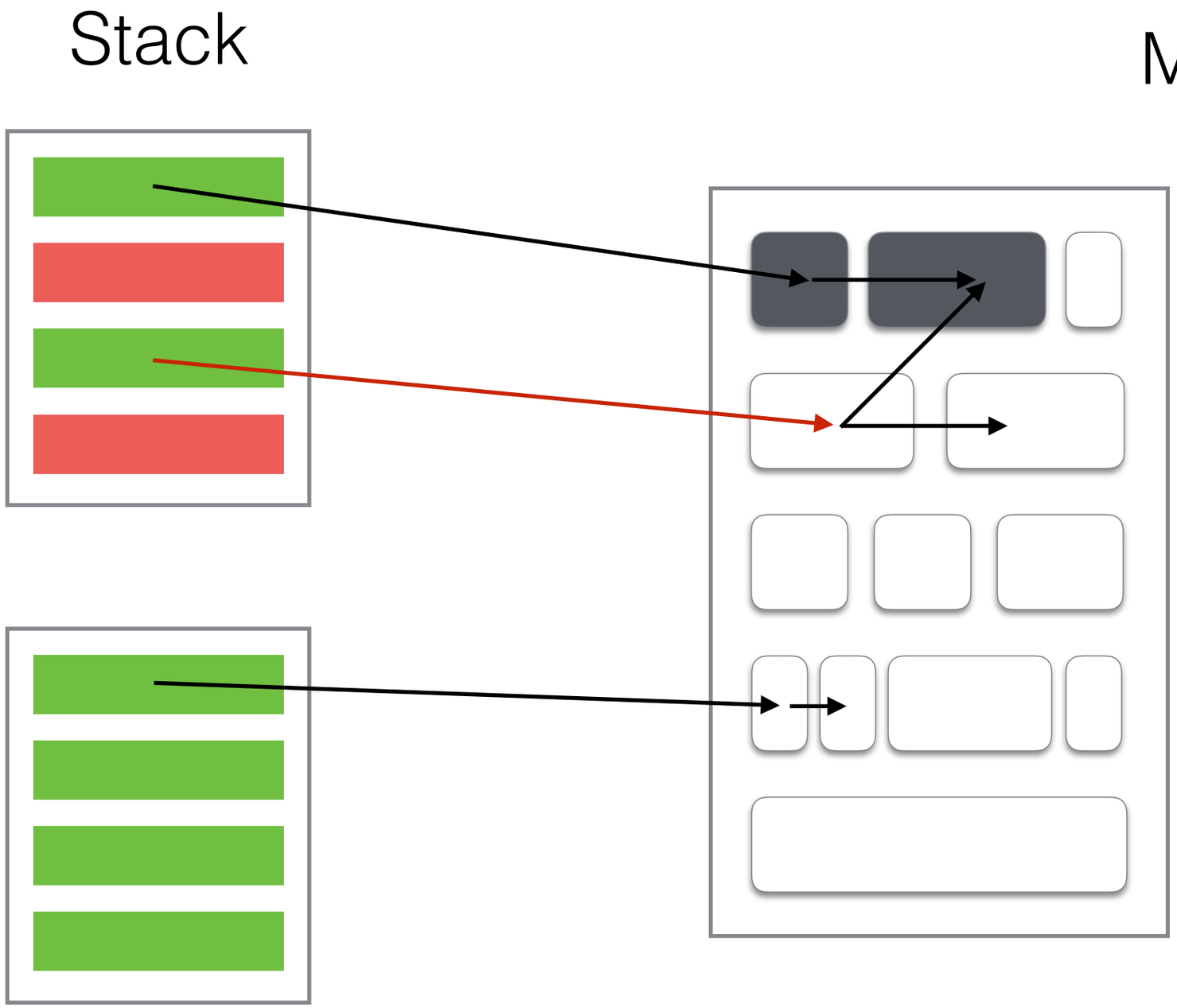
MS



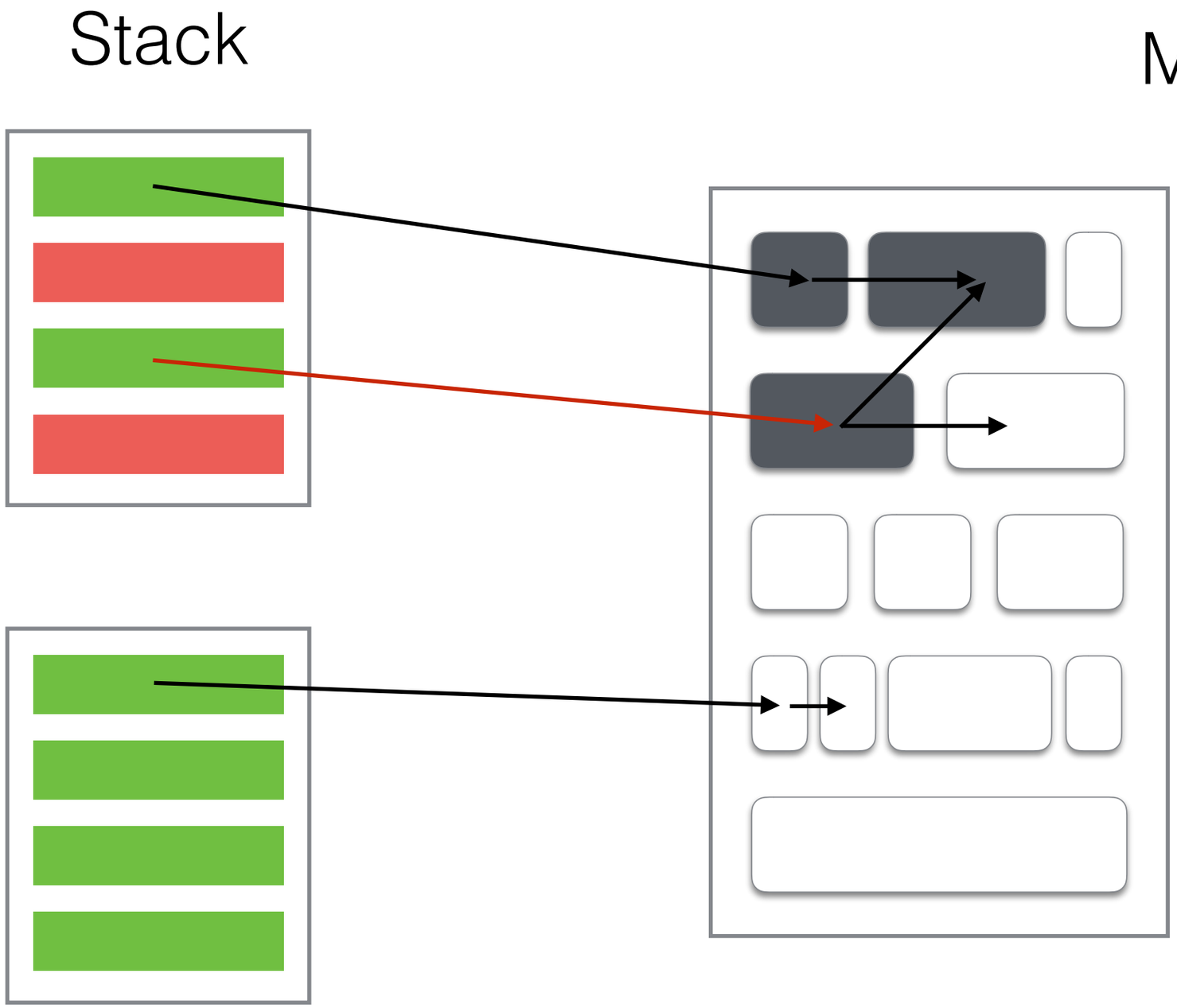
MS



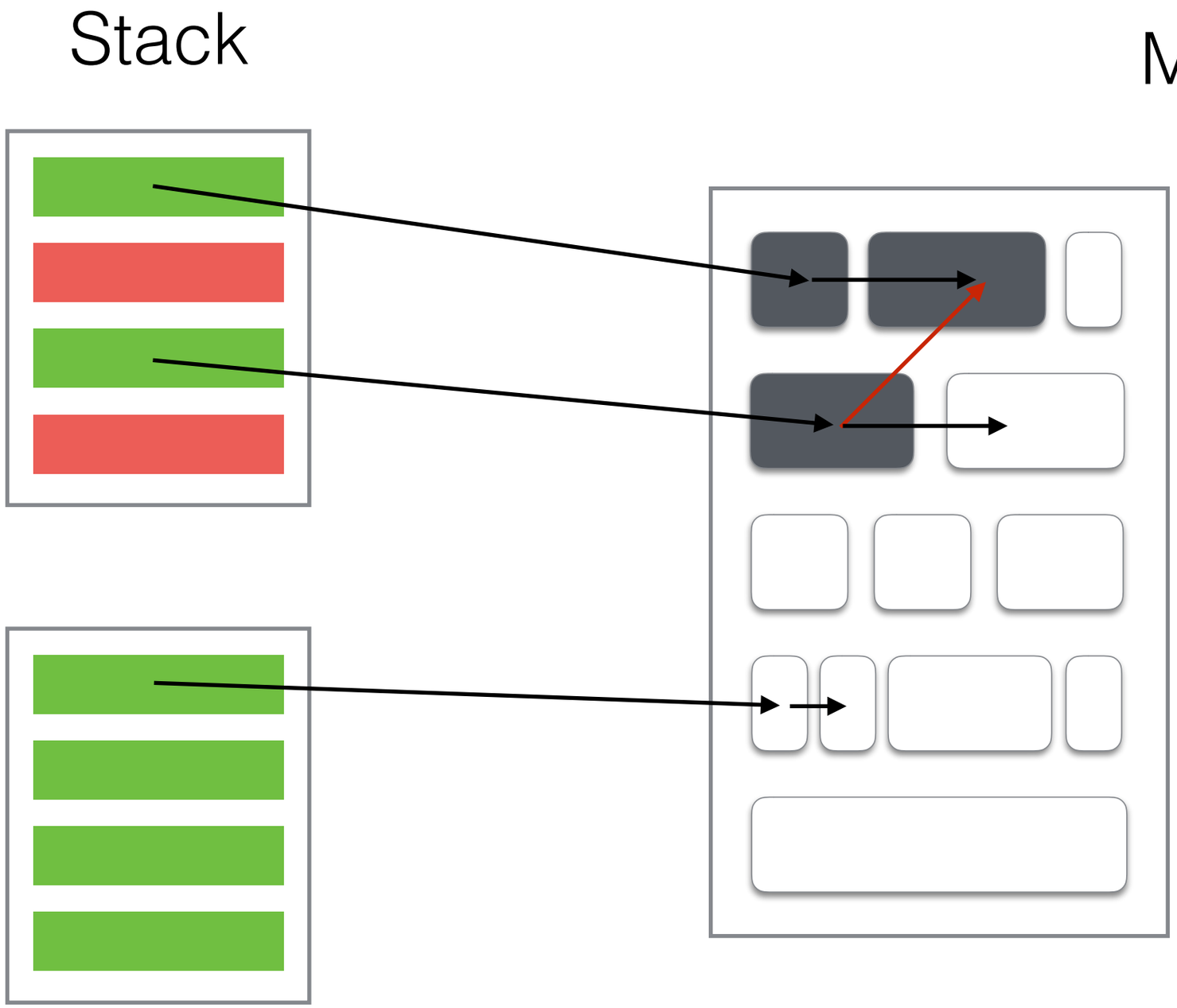
MS



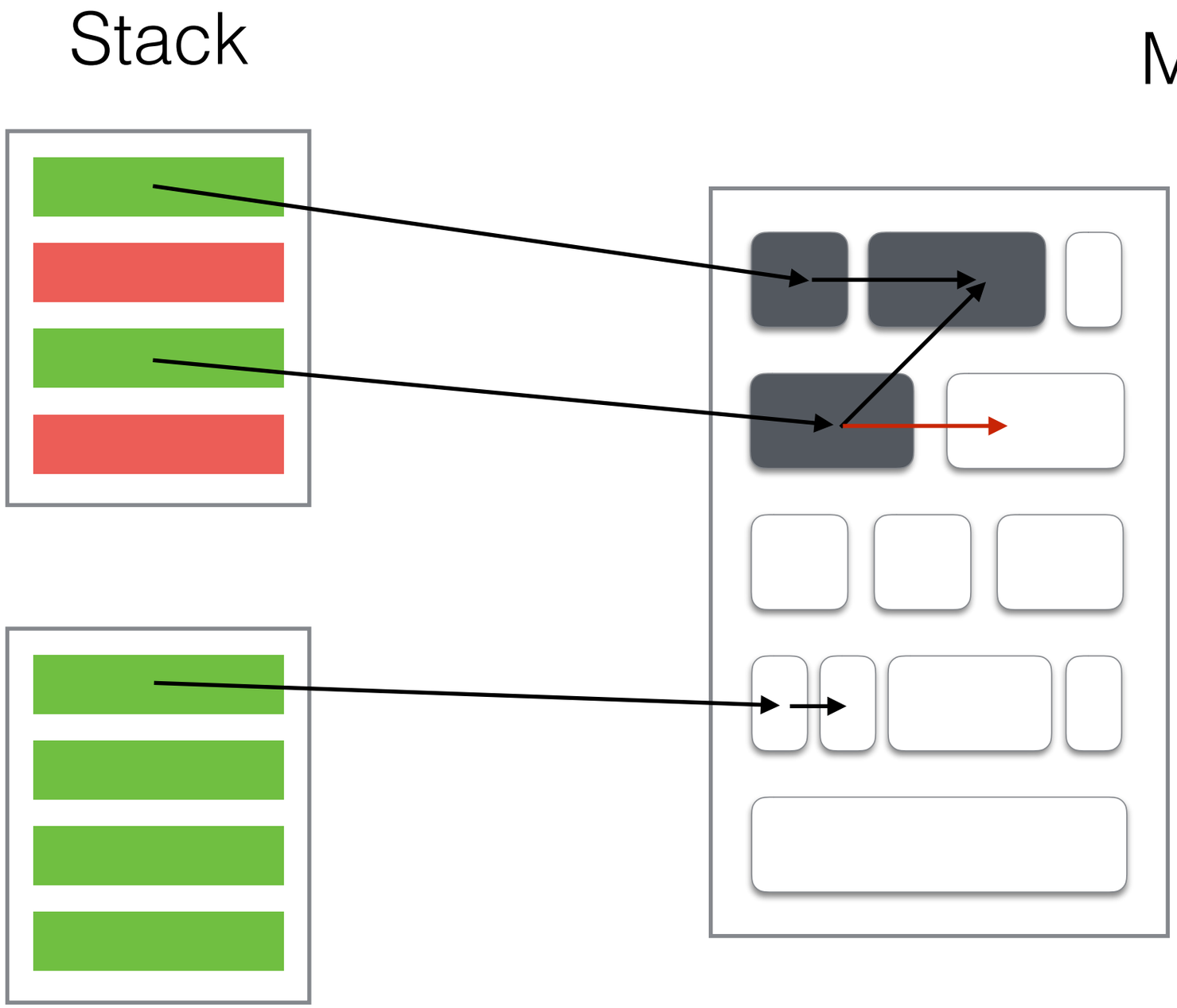
MS



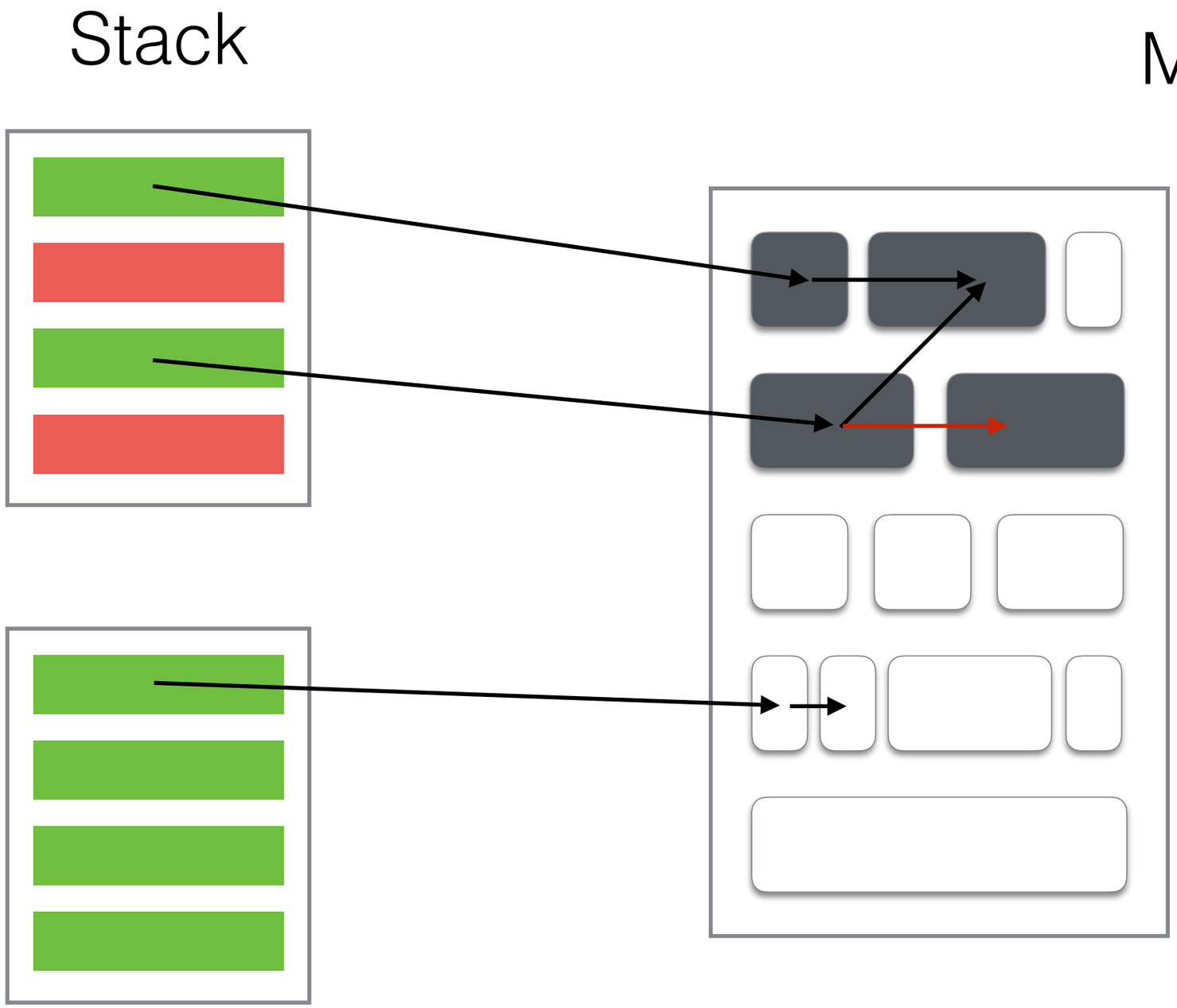
MS



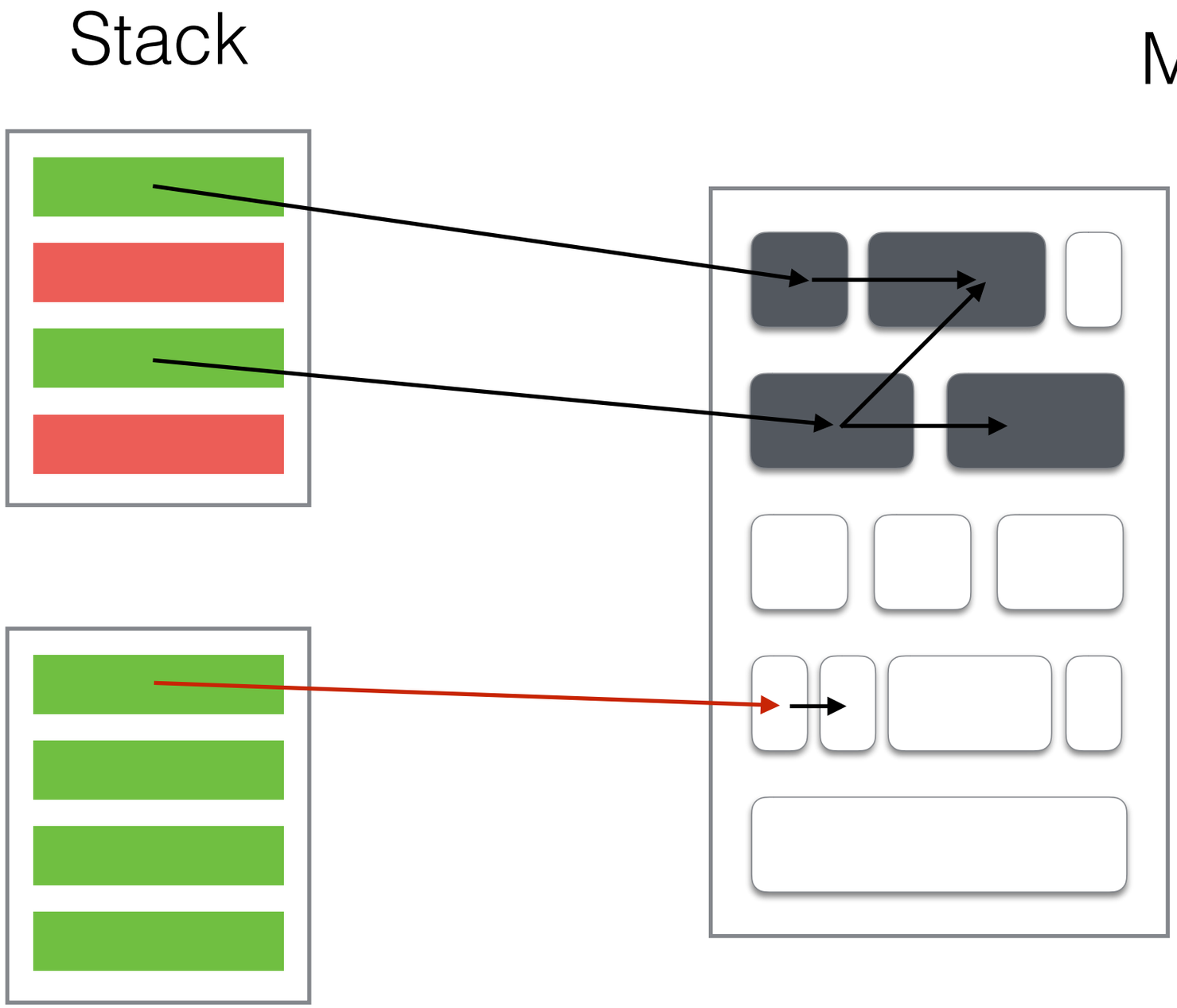
MS



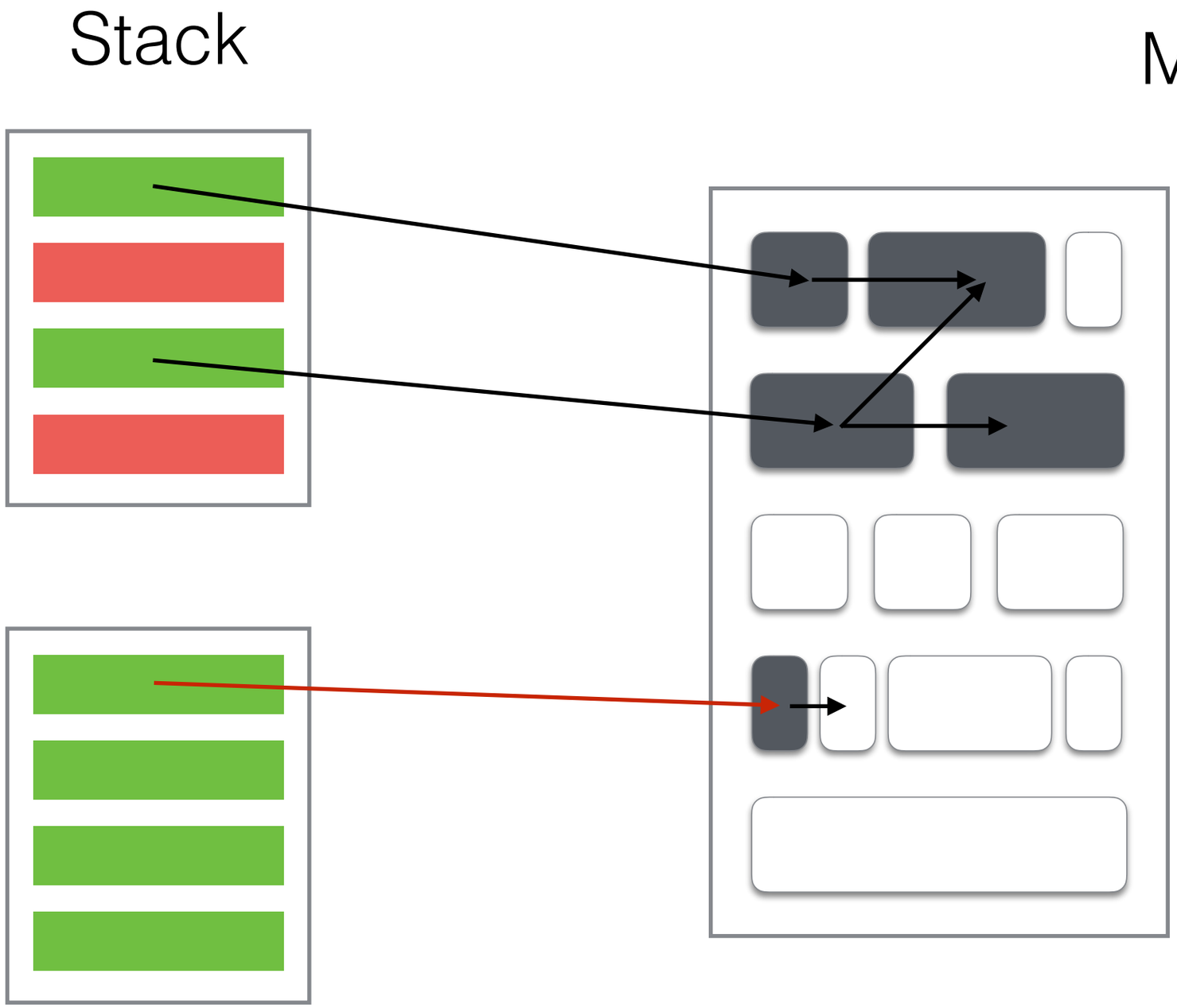
MS



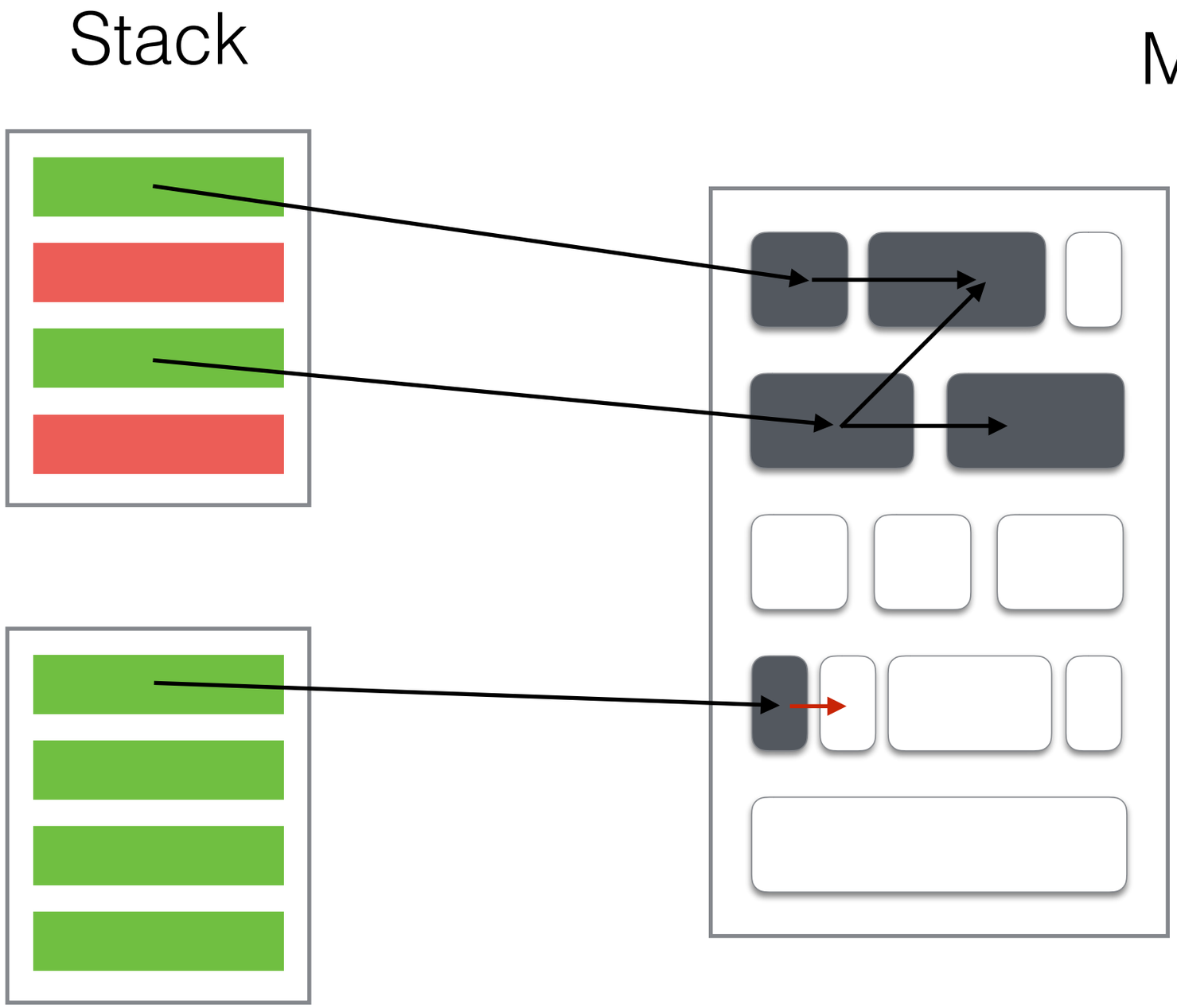
MS



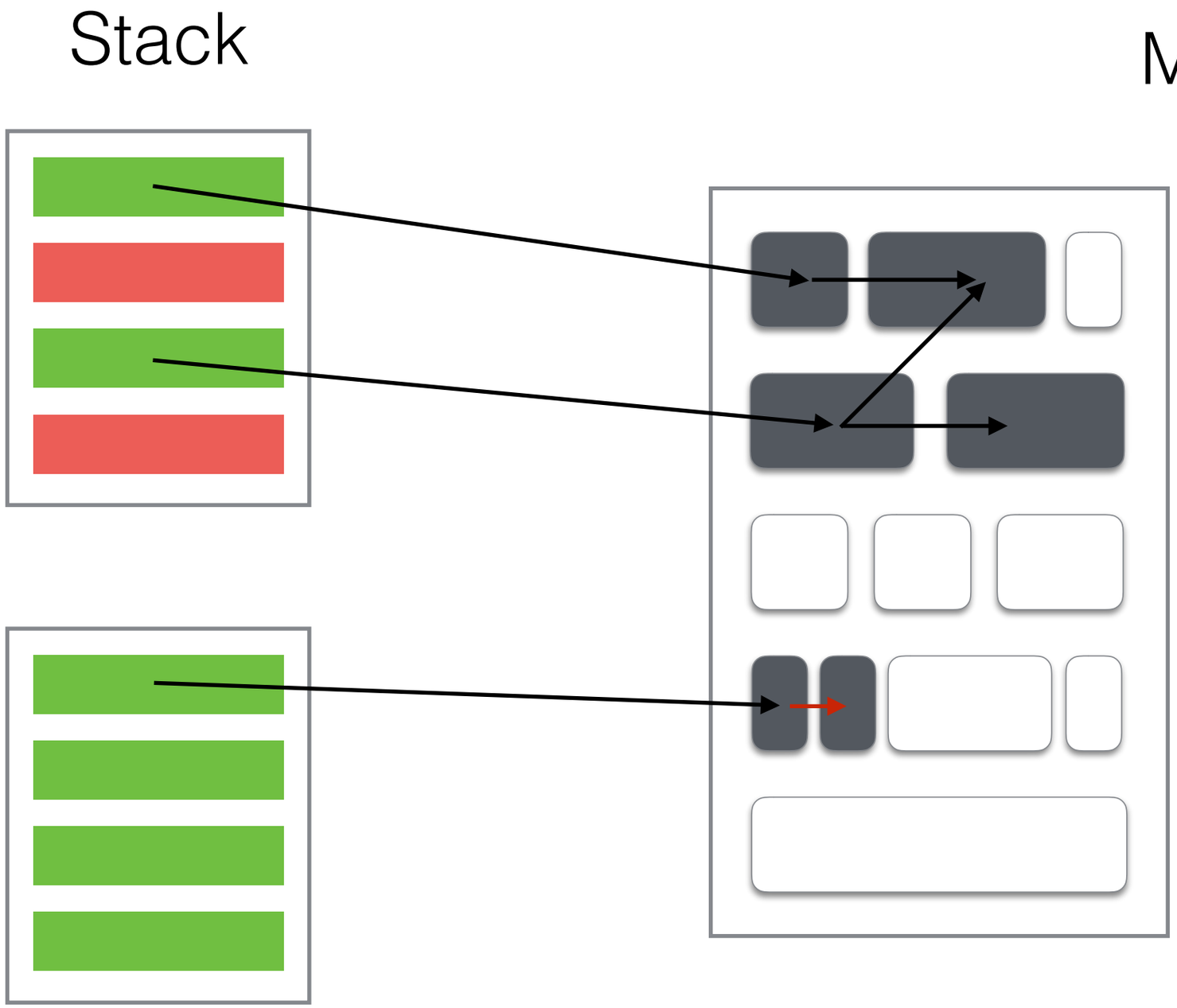
MS



MS

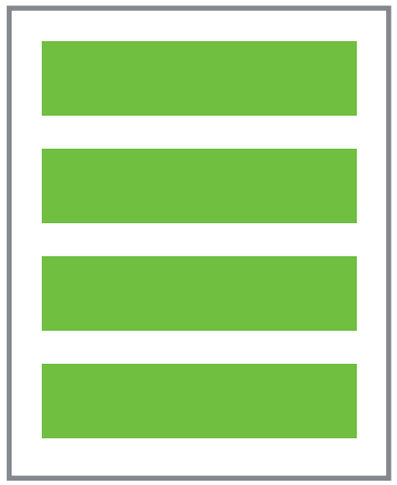


MS



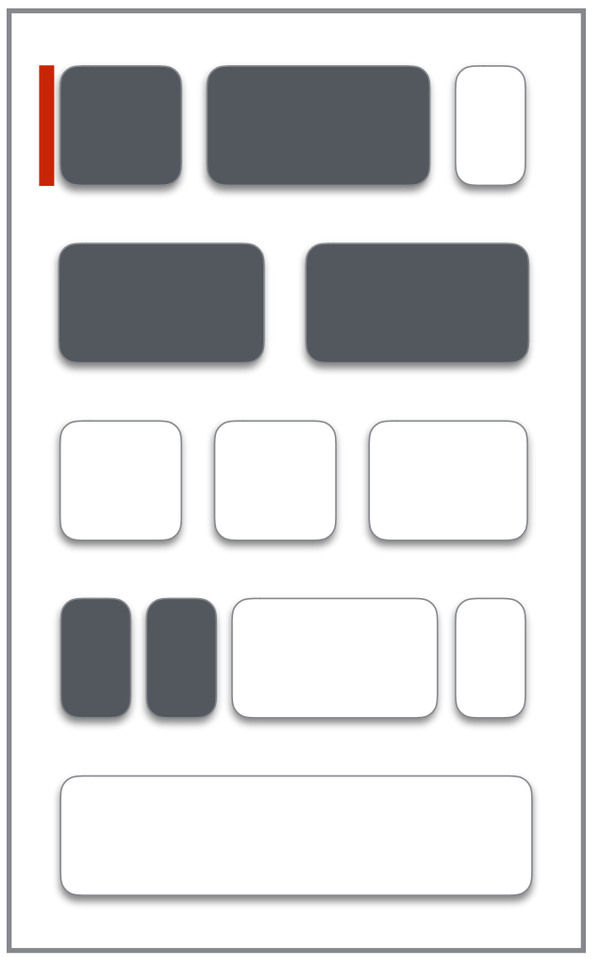
MS

Stack



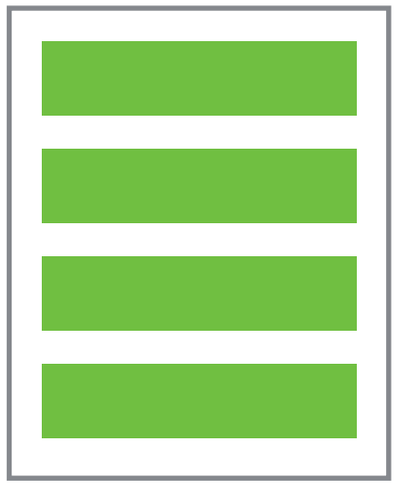
Modules

Sweeping Phase



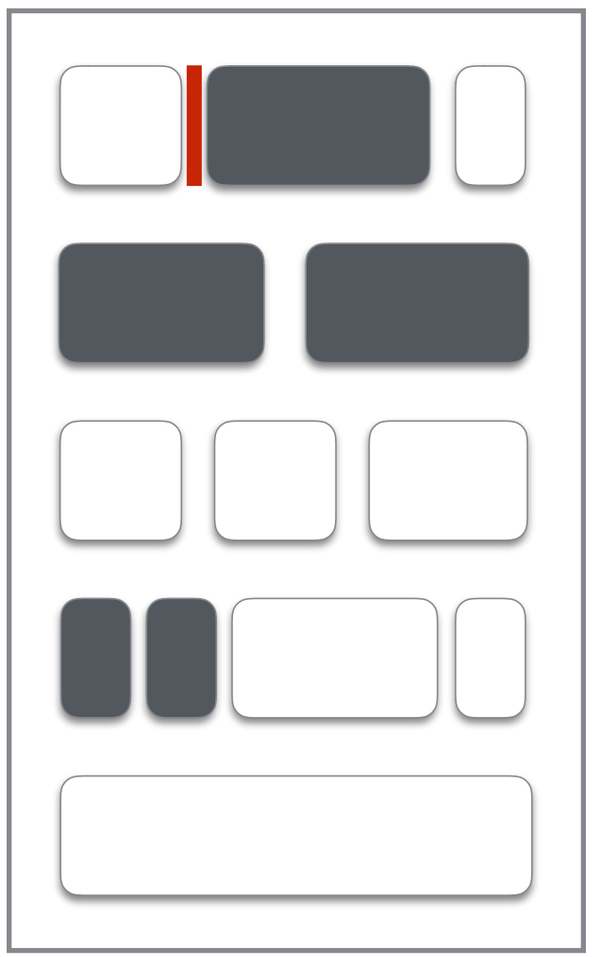
MS

Stack



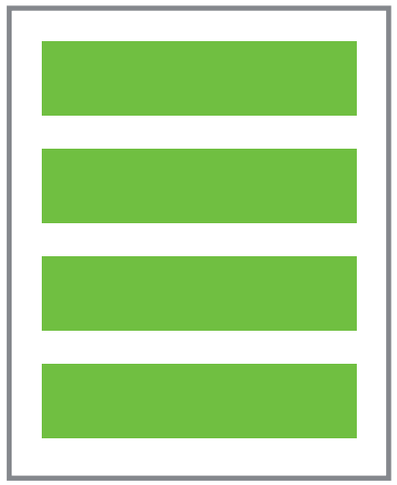
Modules

Sweeping Phase



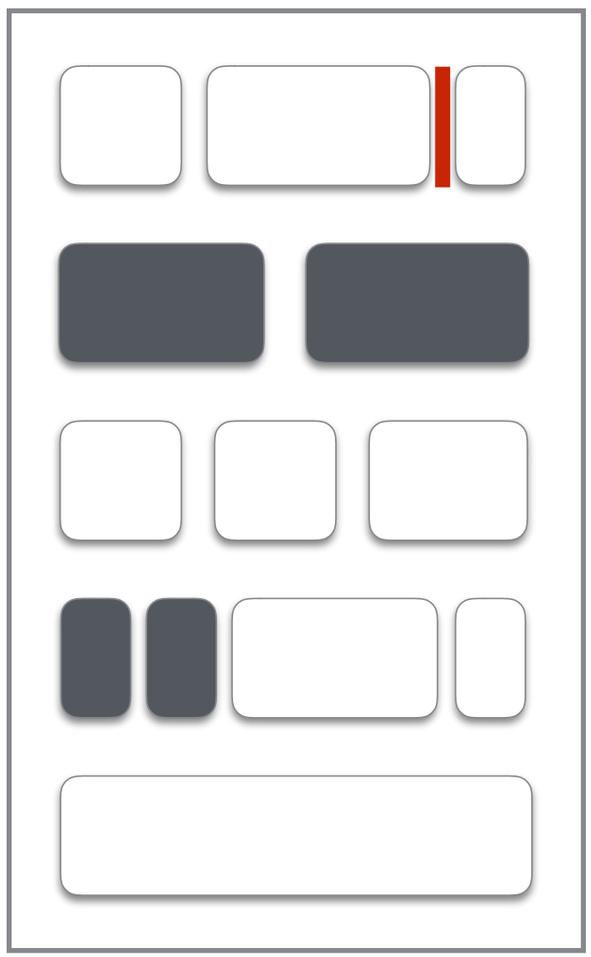
MS

Stack



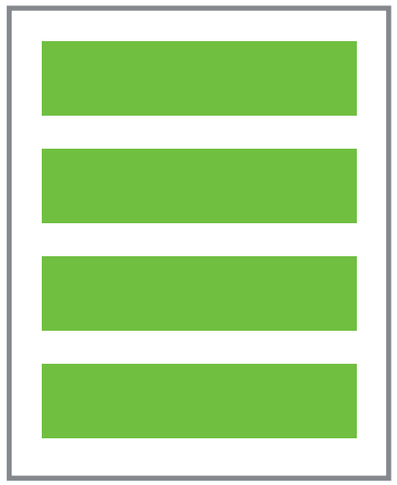
Modules

Sweeping Phase



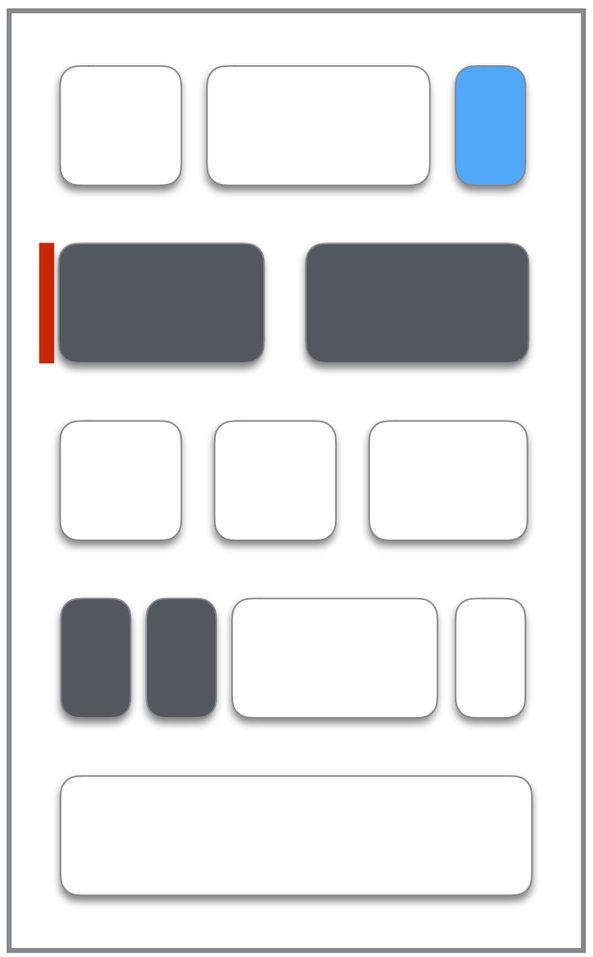
MS

Stack



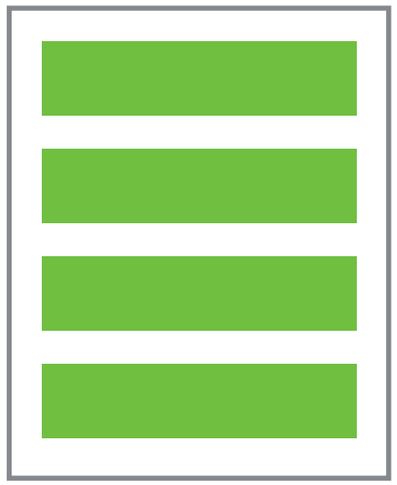
Modules

Sweeping Phase



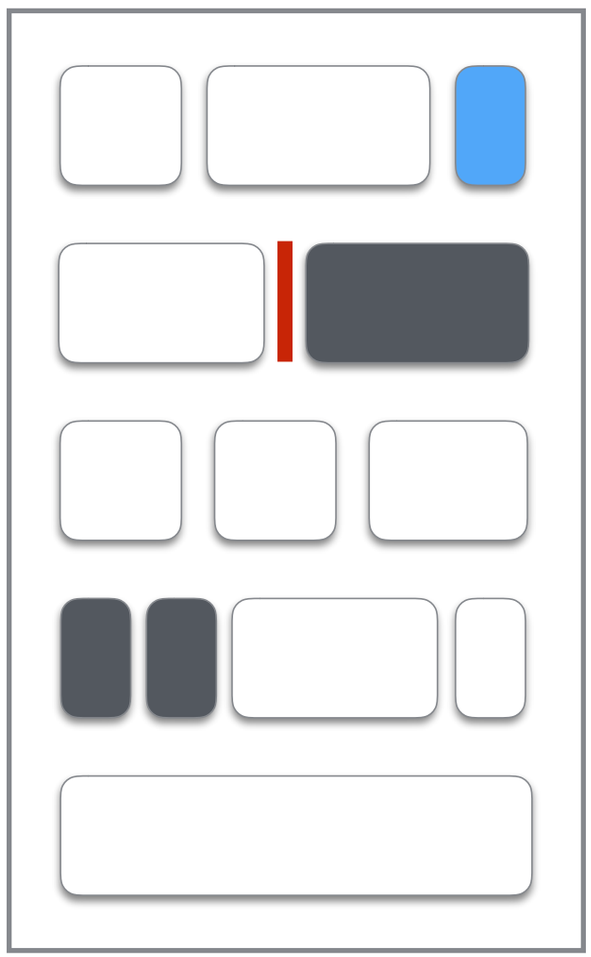
MS

Stack



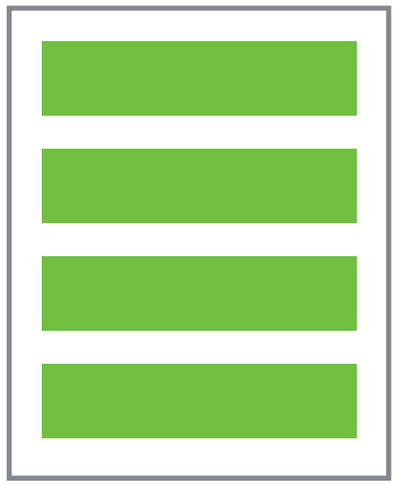
Modules

Sweeping Phase



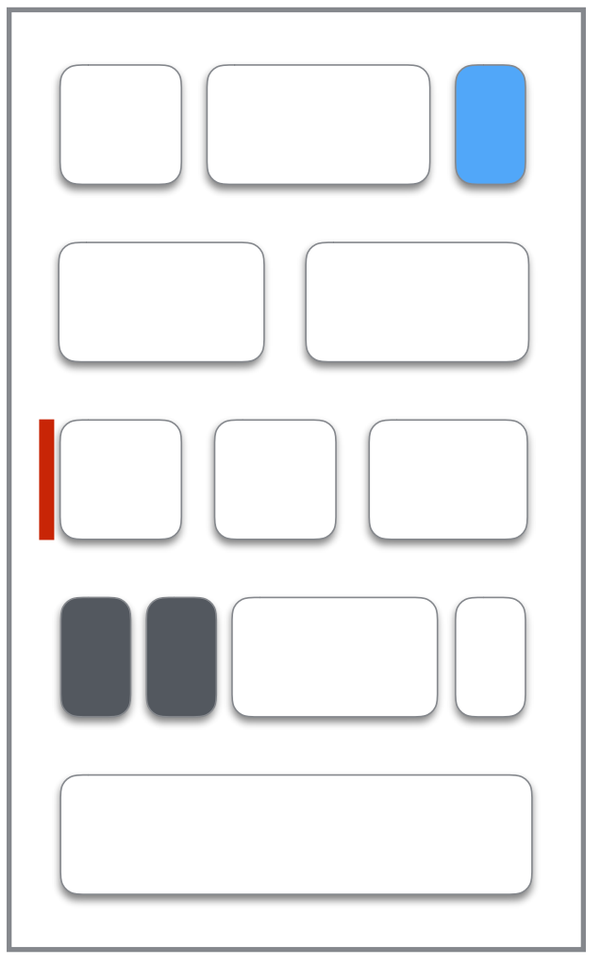
MS

Stack



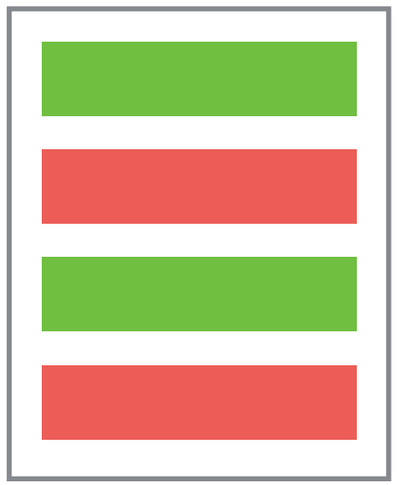
Modules

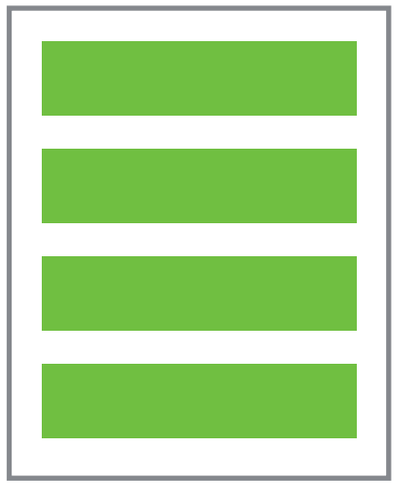
Sweeping Phase



MS

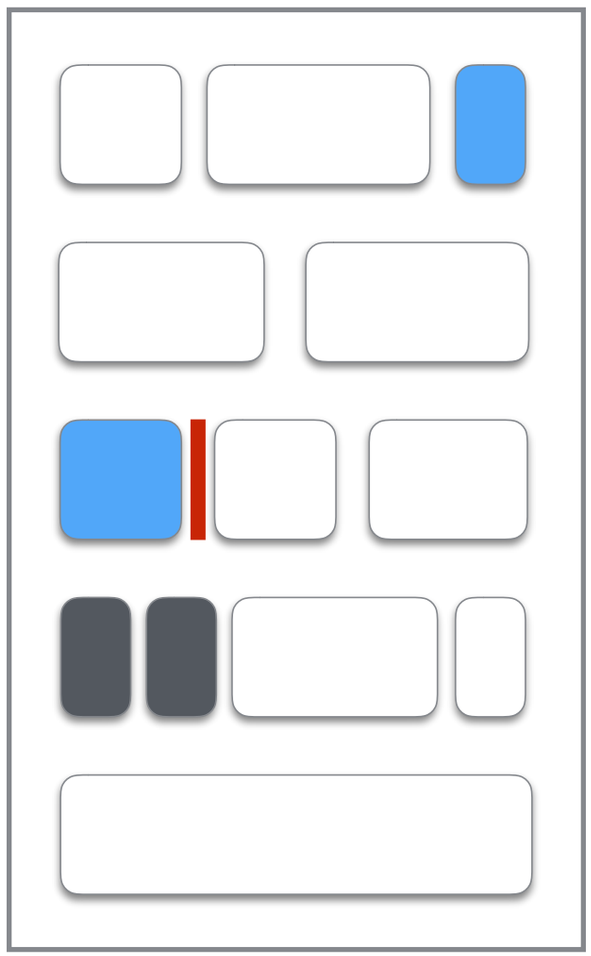
Stack





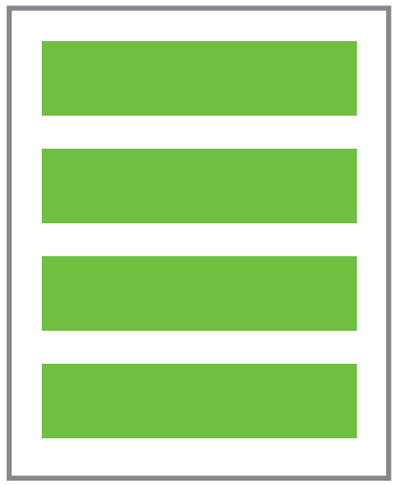
Modules

Sweeping Phase



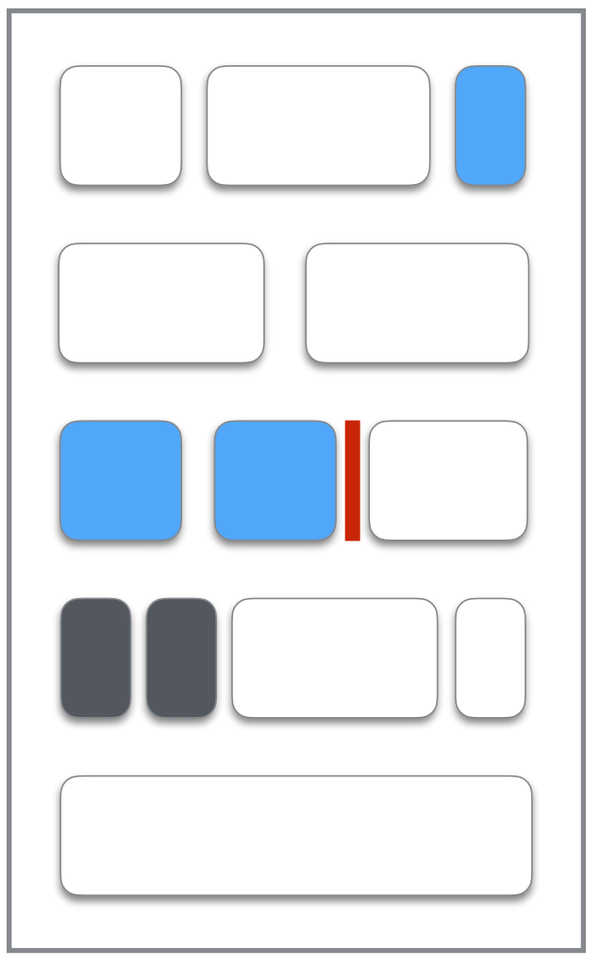
MS

Stack



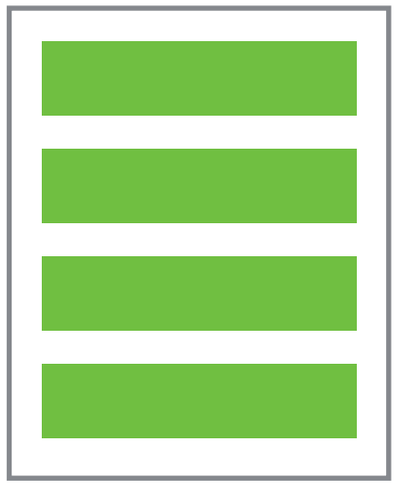
Modules

Sweeping Phase



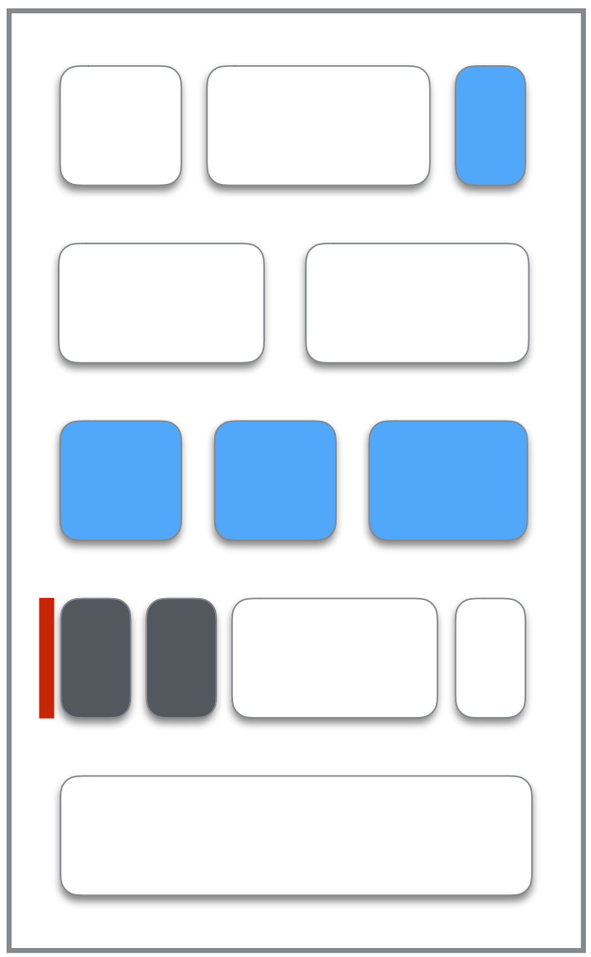
MS

Stack



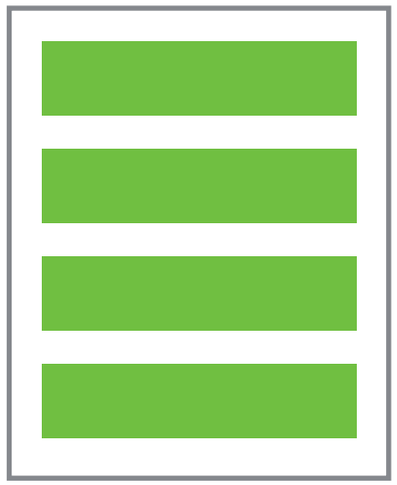
Modules

Sweeping Phase



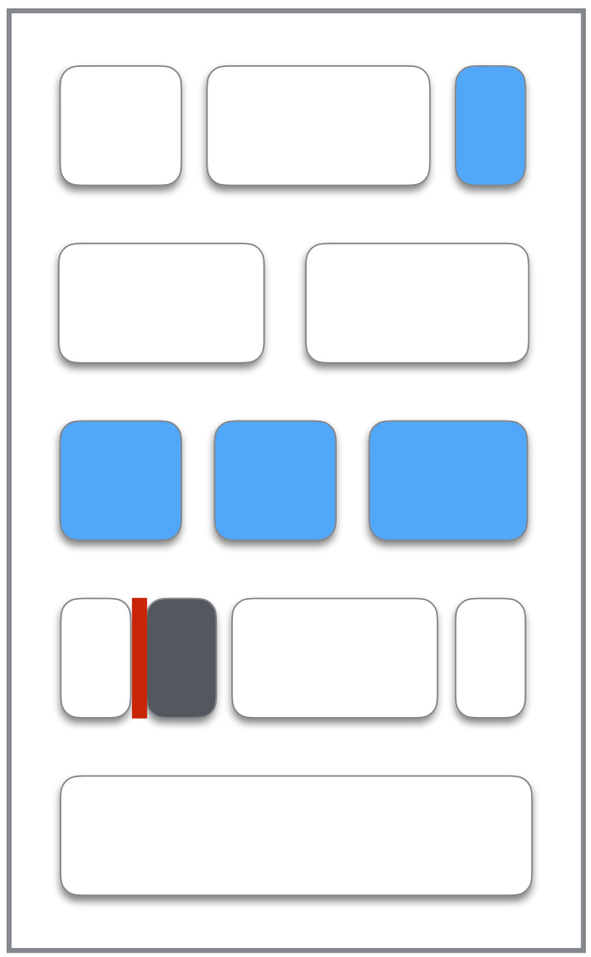
MS

Stack



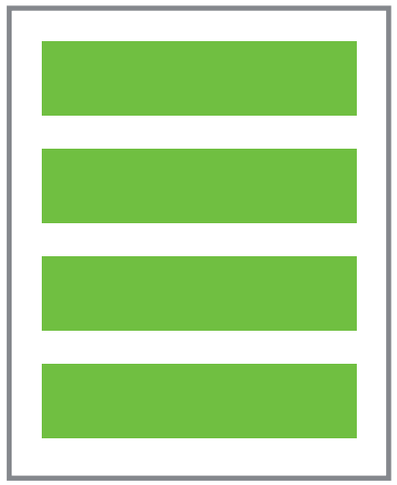
Modules

Sweeping Phase



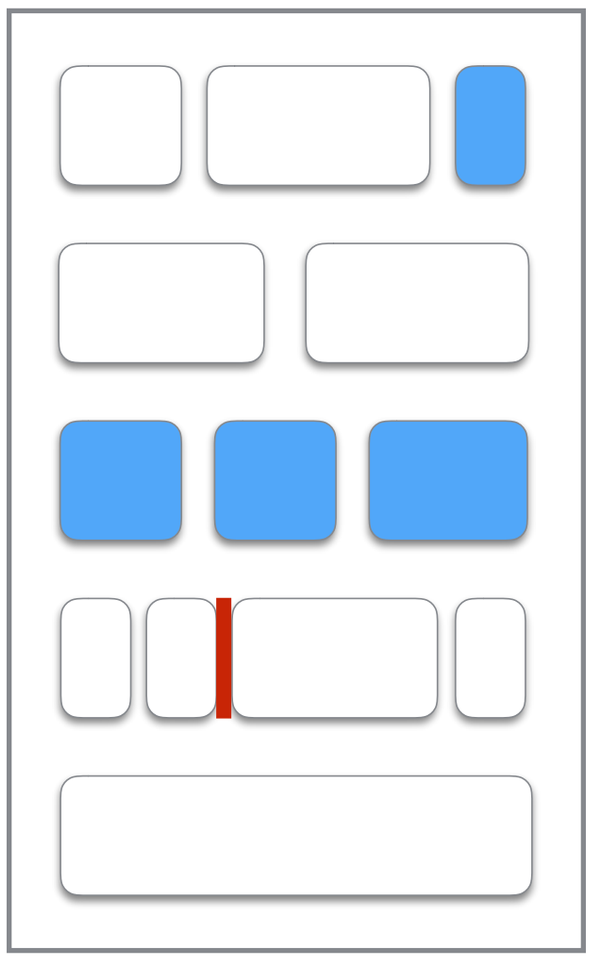
MS

Stack



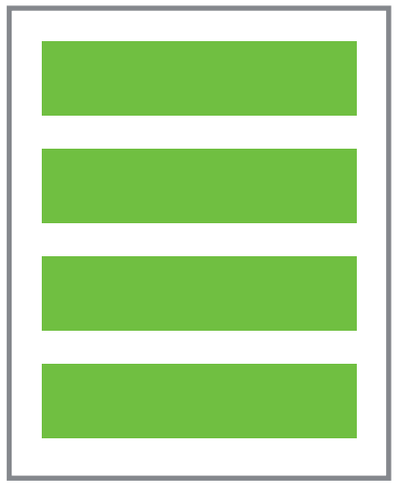
Modules

Sweeping Phase



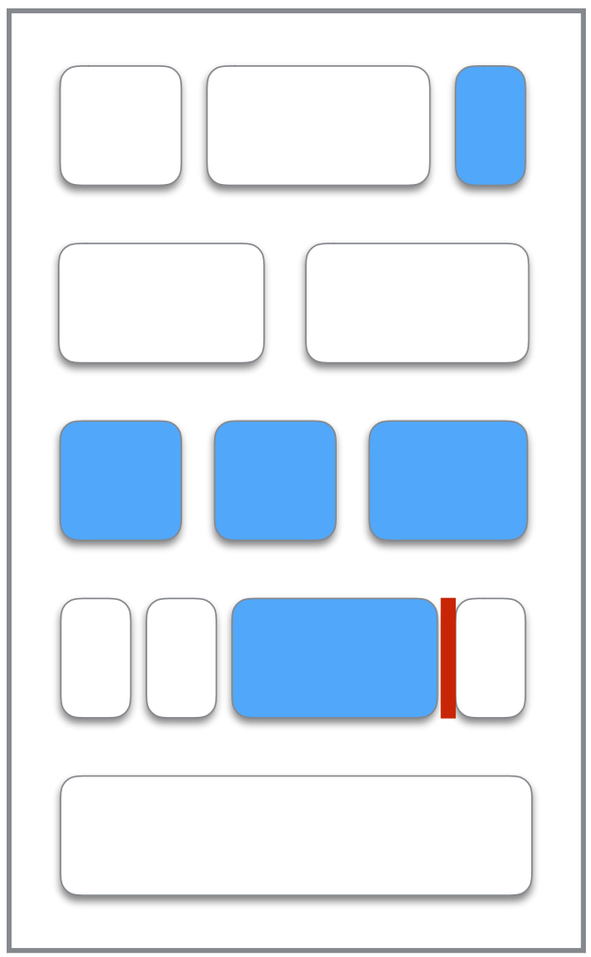
MS

Stack



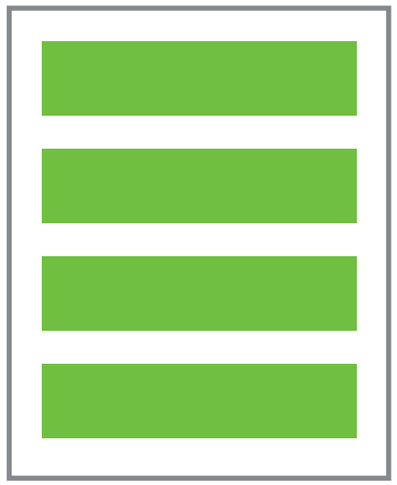
Modules

Sweeping Phase



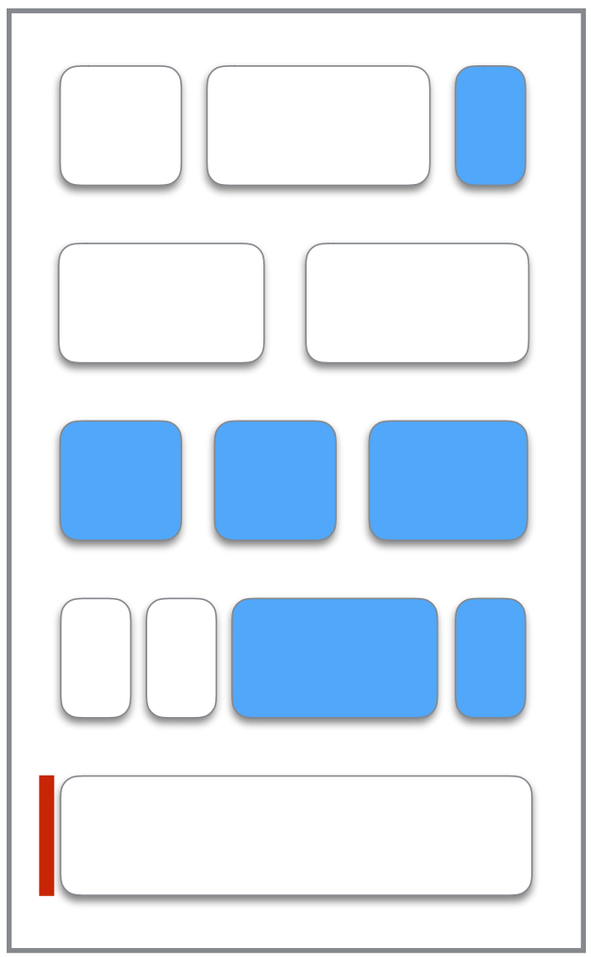
MS

Stack



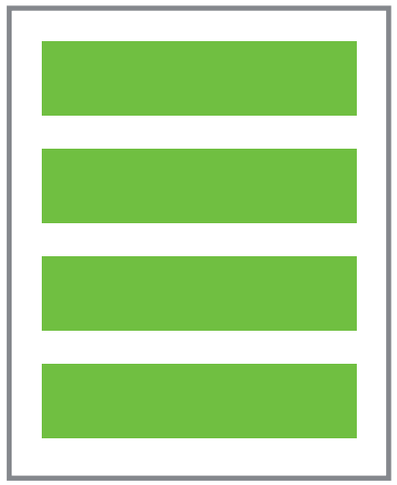
Modules

Sweeping Phase



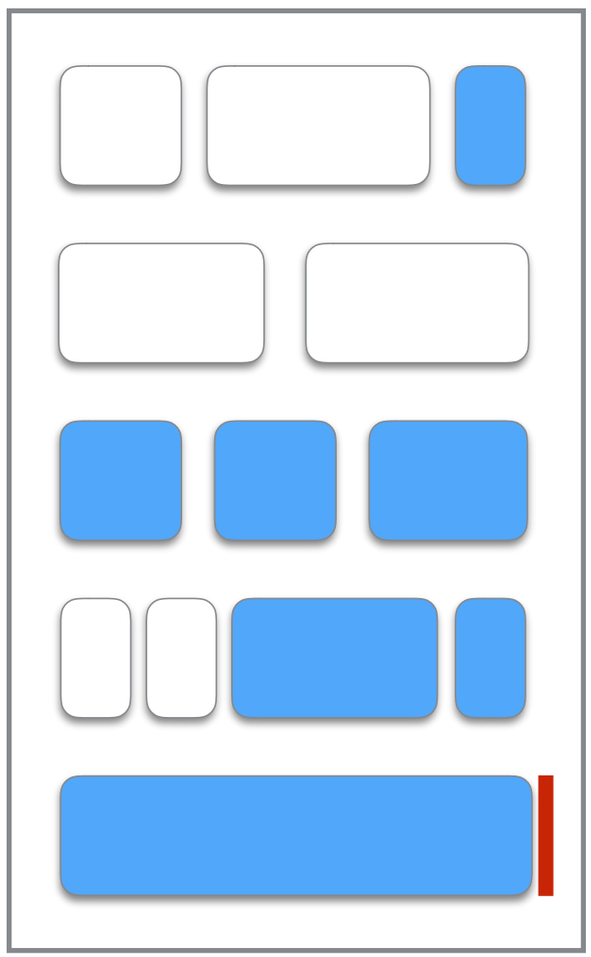
MS

Stack

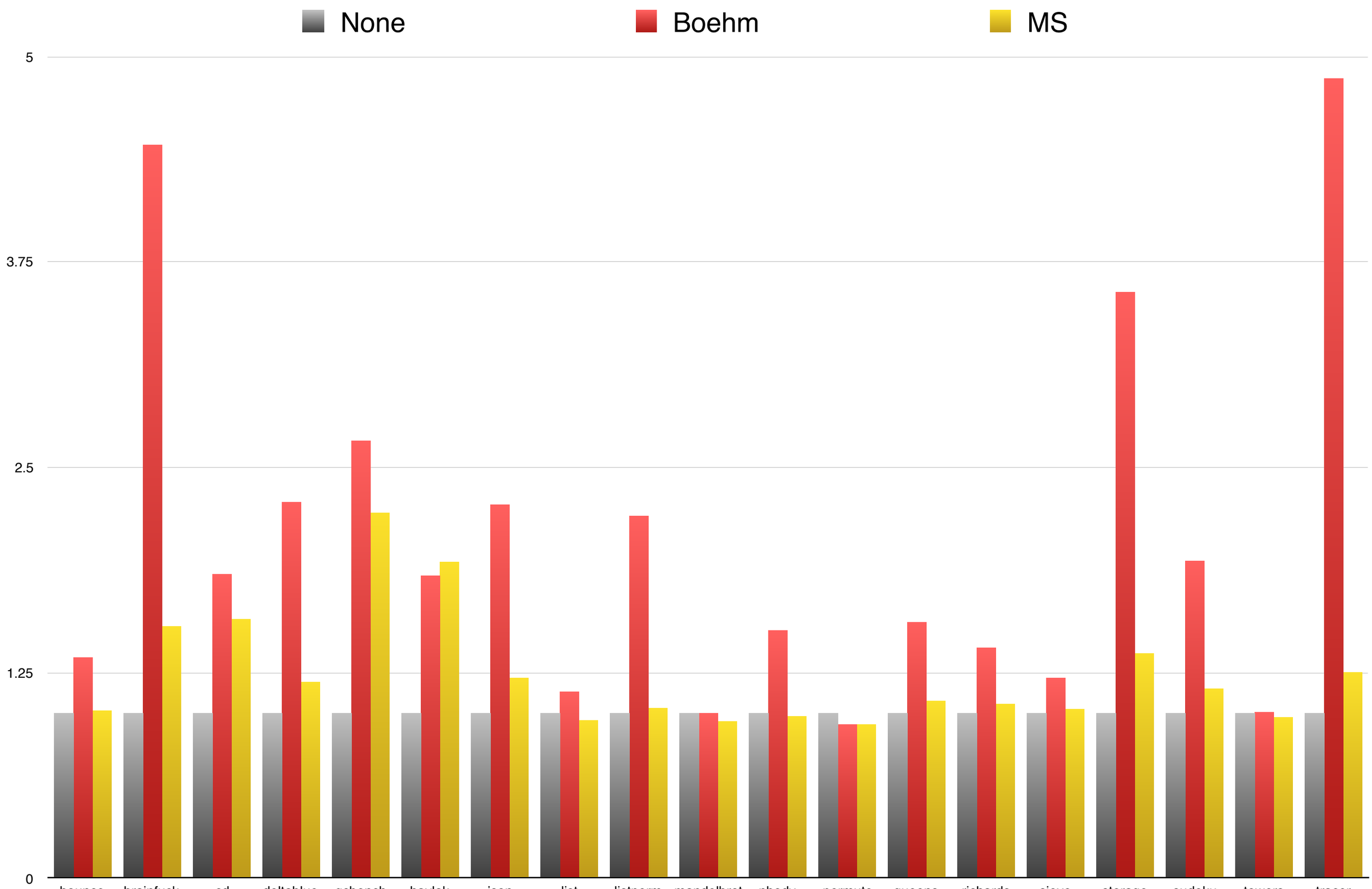


Modules

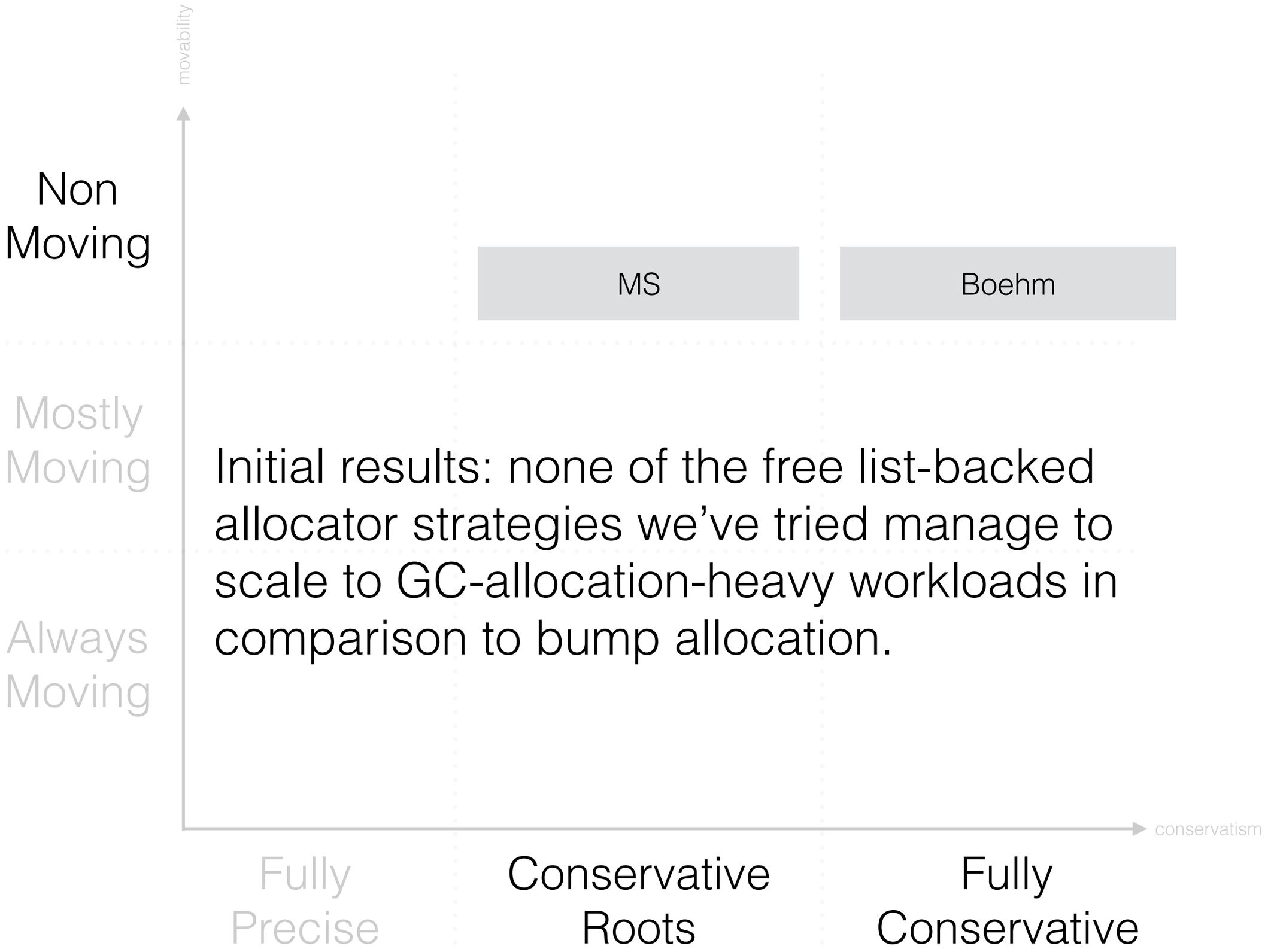
Sweeping Phase



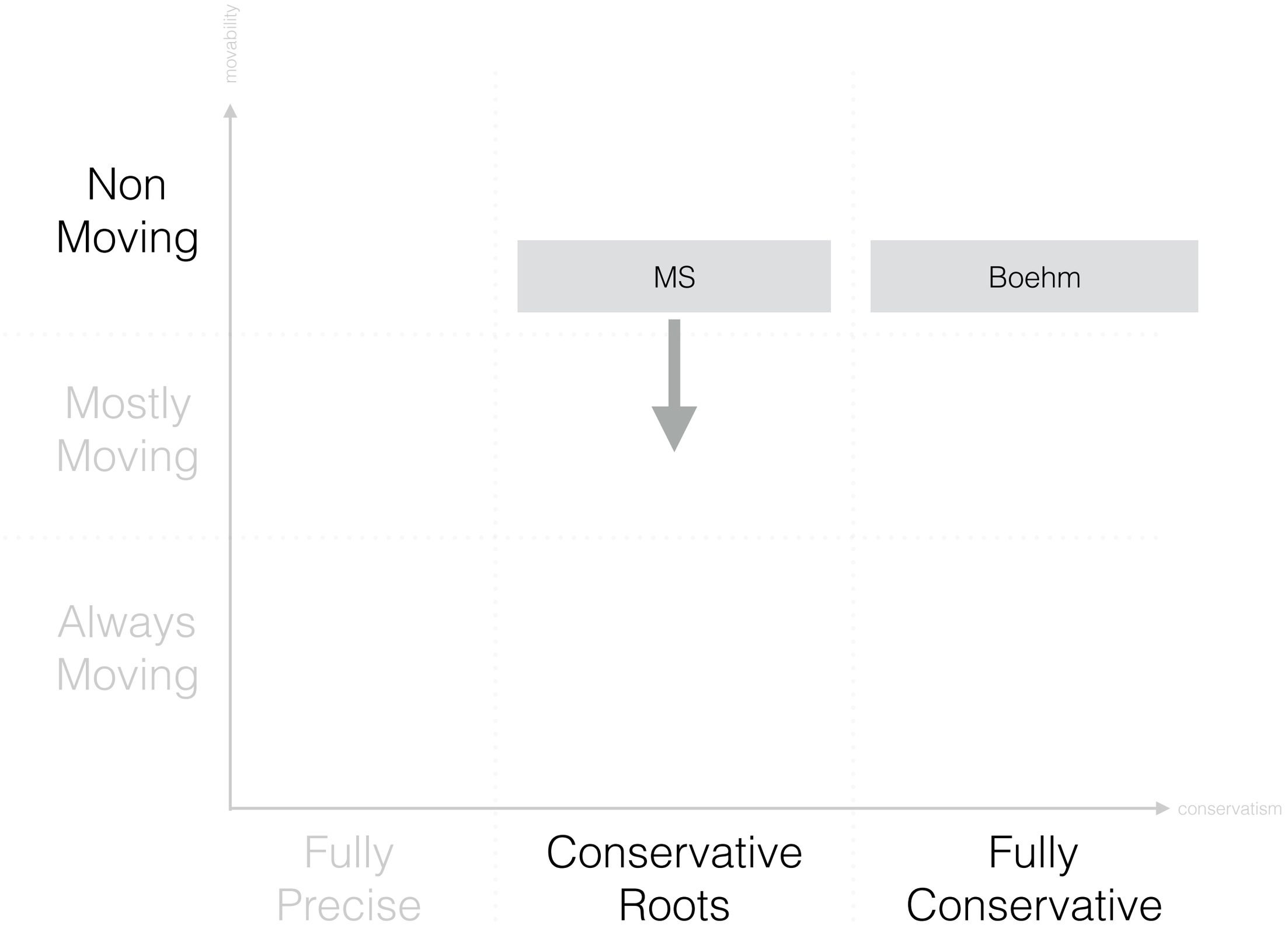
MS Performance



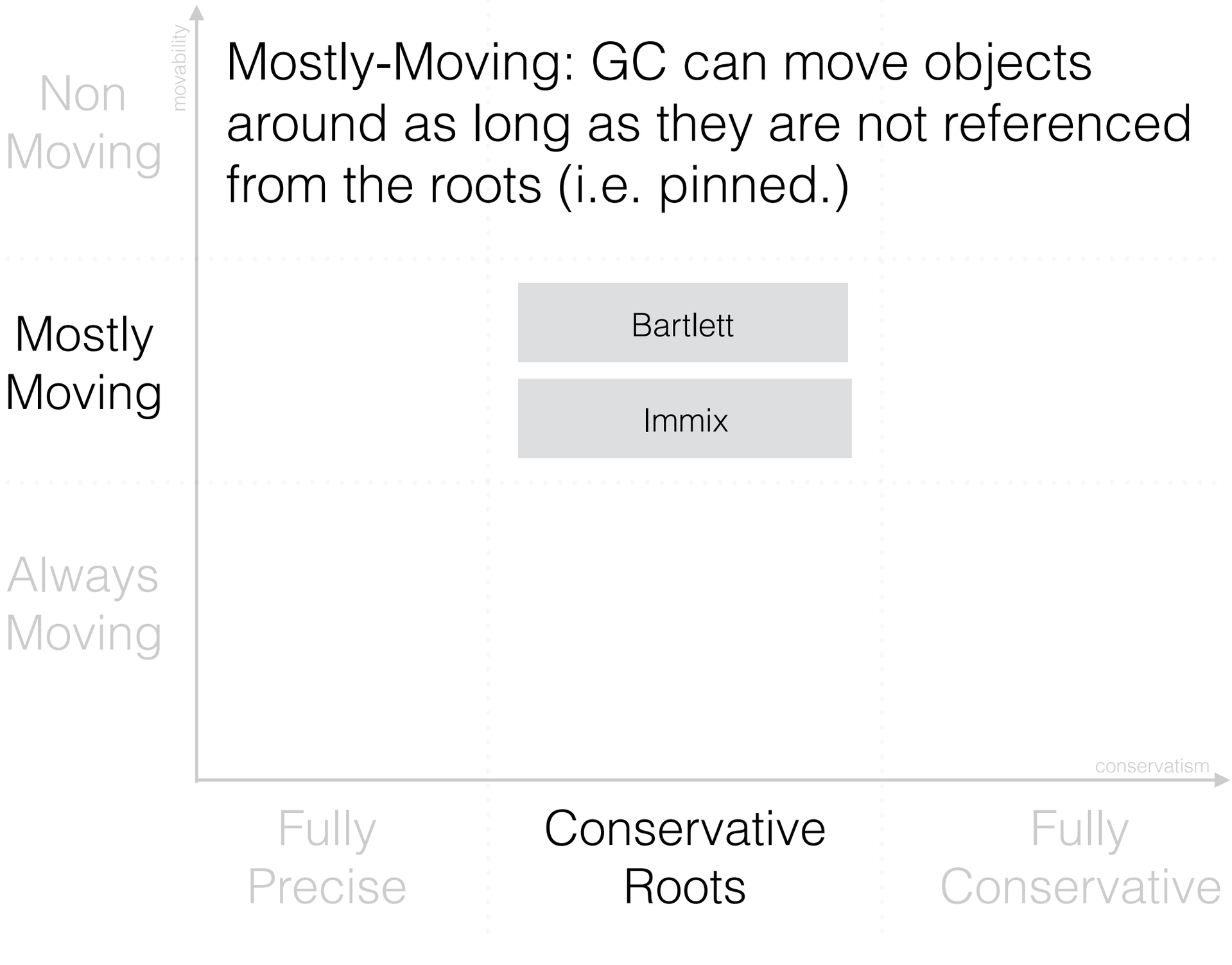
Initial experiments



Initial experiments



Mostly-Moving Sweet Spot



Bartlett

Non Moving

Mostly Moving

Always Moving

movability

Copying GC that's able to pin objects which are referred to from the roots. Used in Safari's WebKit engine.

Bartlett

"Compacting Garbage Collection with Ambiguous Roots"
Joel F. Bartlett

conservatism

Fully Precise

Conservative Roots

Fully Conservative

Immix

movability

Non Moving

Mostly Moving

Always Moving

Mark-region garbage collector with opportunistic one-pass defragmentation.

Immix

"Immix: a mark-region garbage collector with space efficiency, fast collection, and mutator performance"
Stephen M. Blackburn, Kathryn S. McKinley

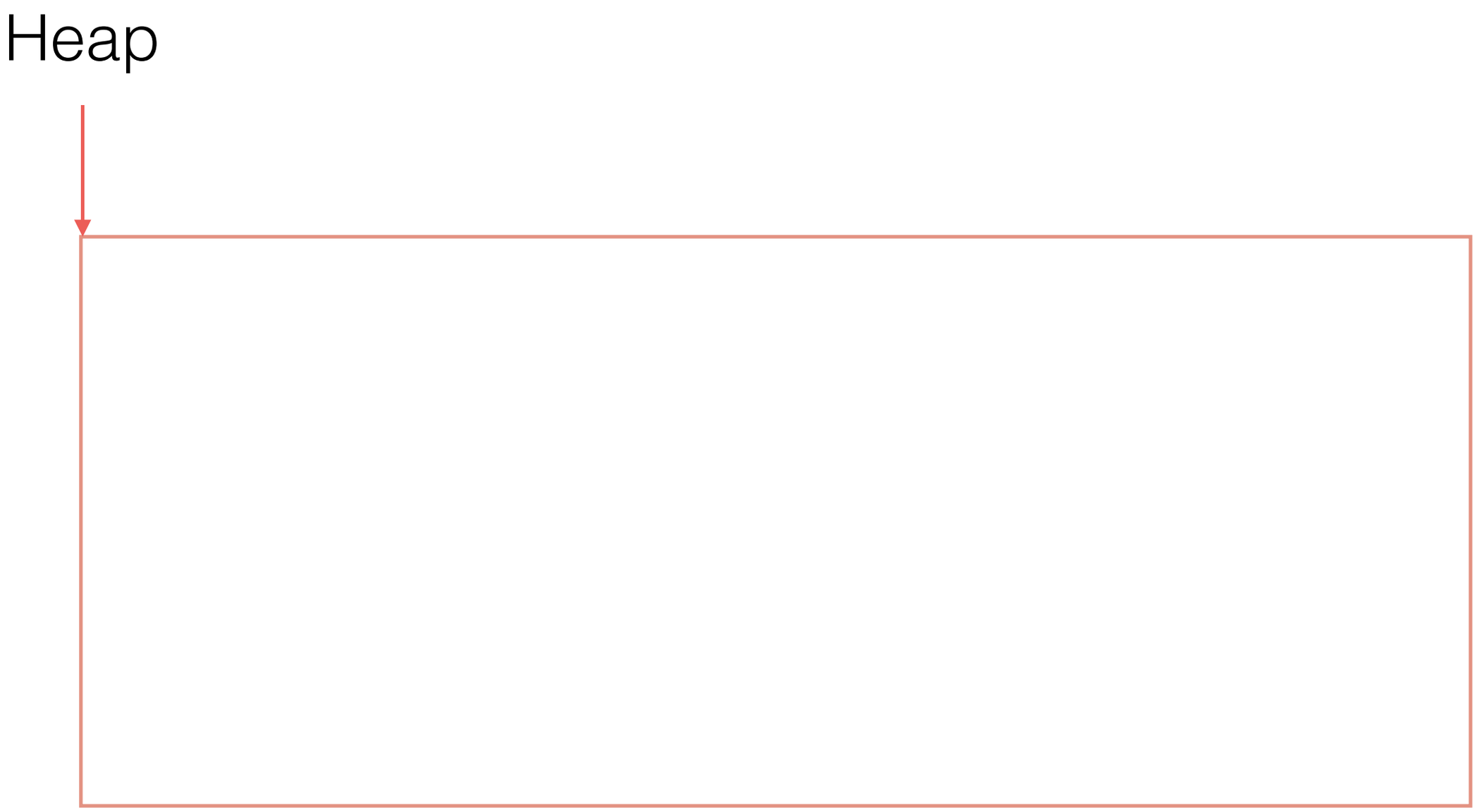
conservatism

Fully Precise

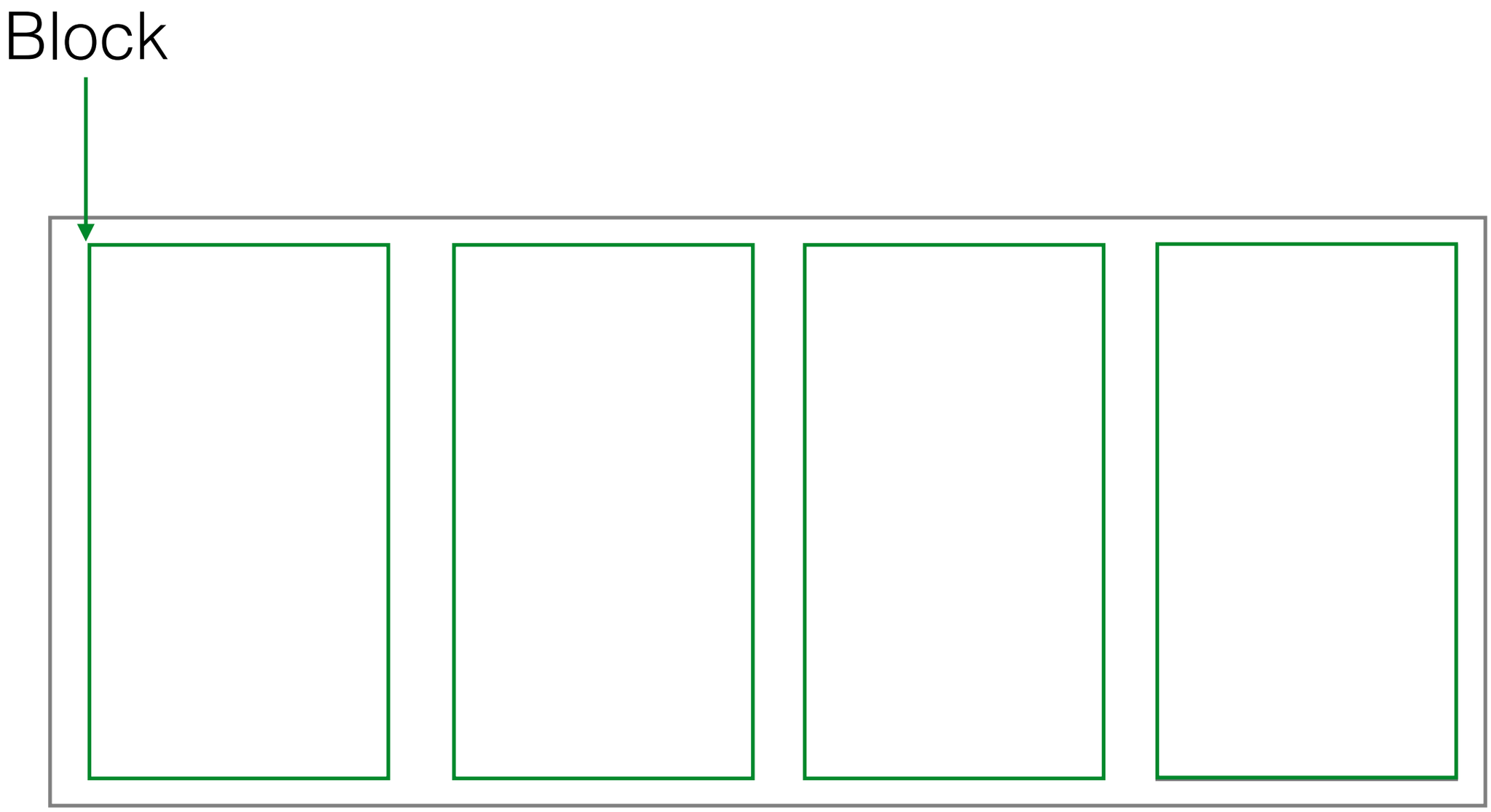
Conservative Roots

Fully Conservative

Immix

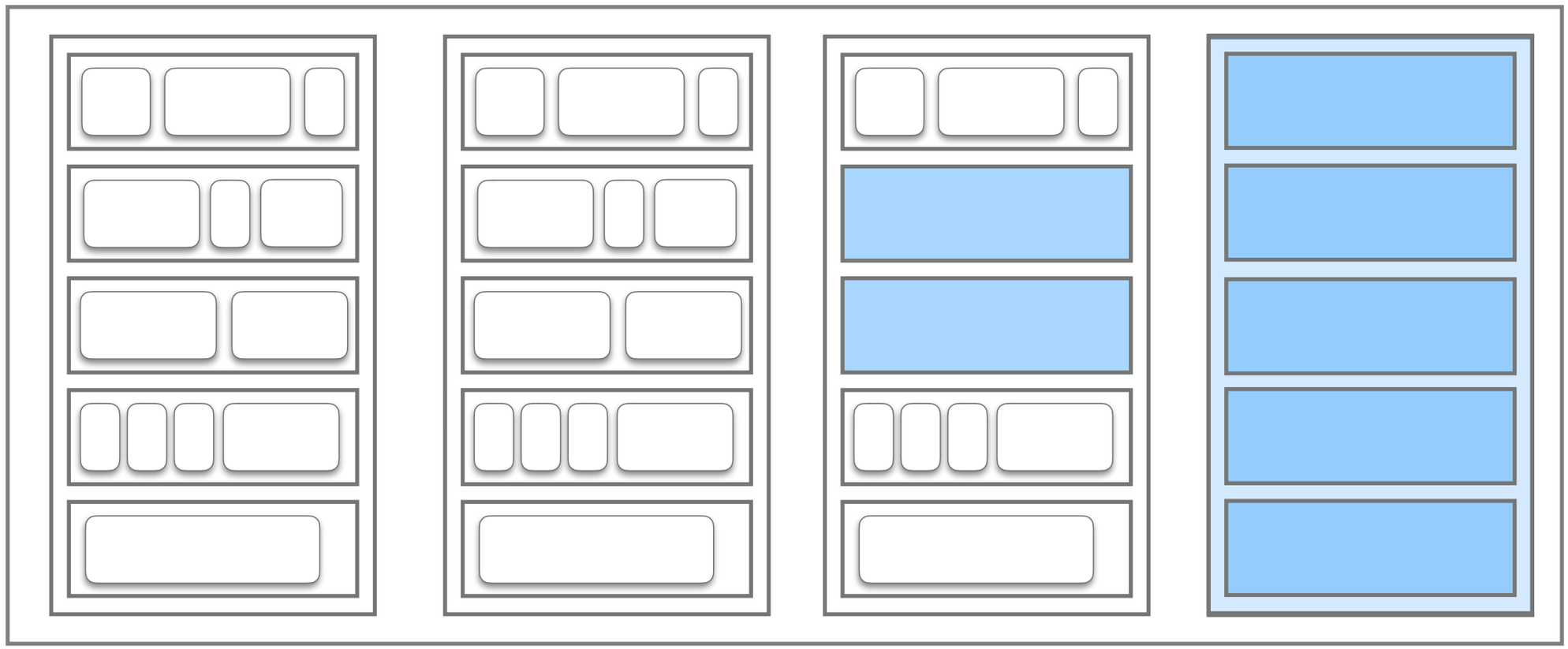


Immix

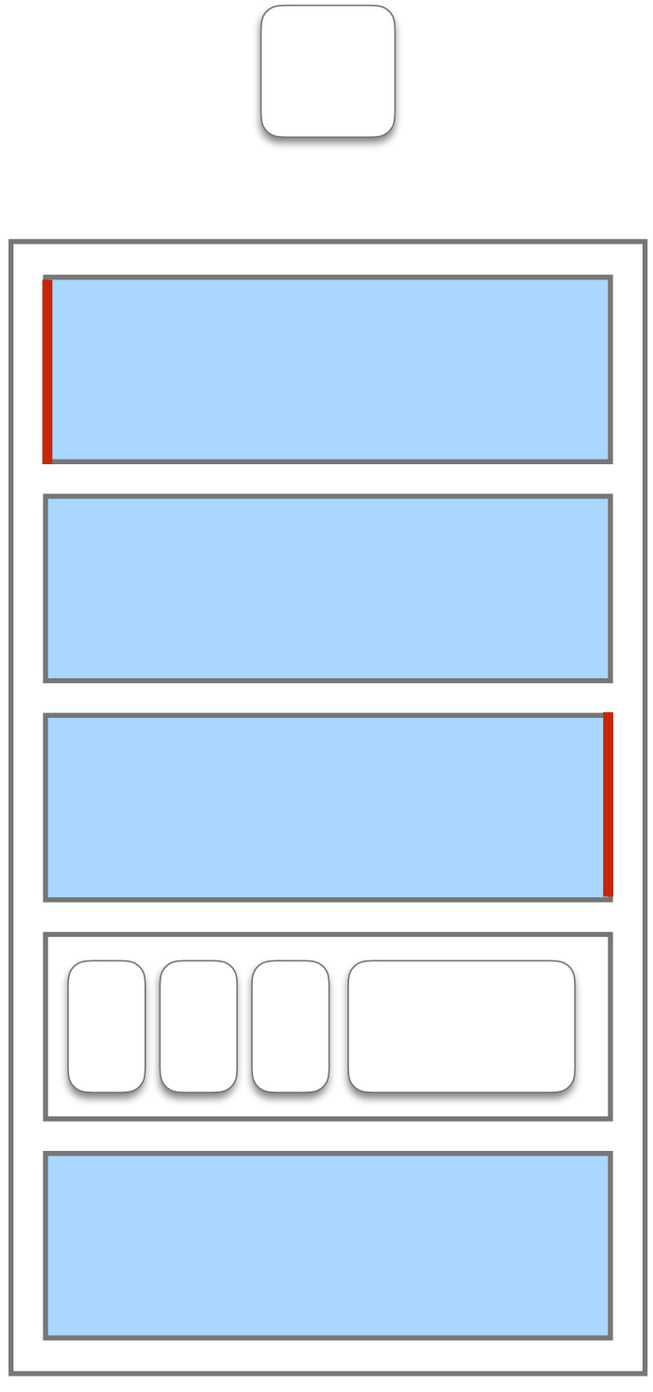


Immix

Immix

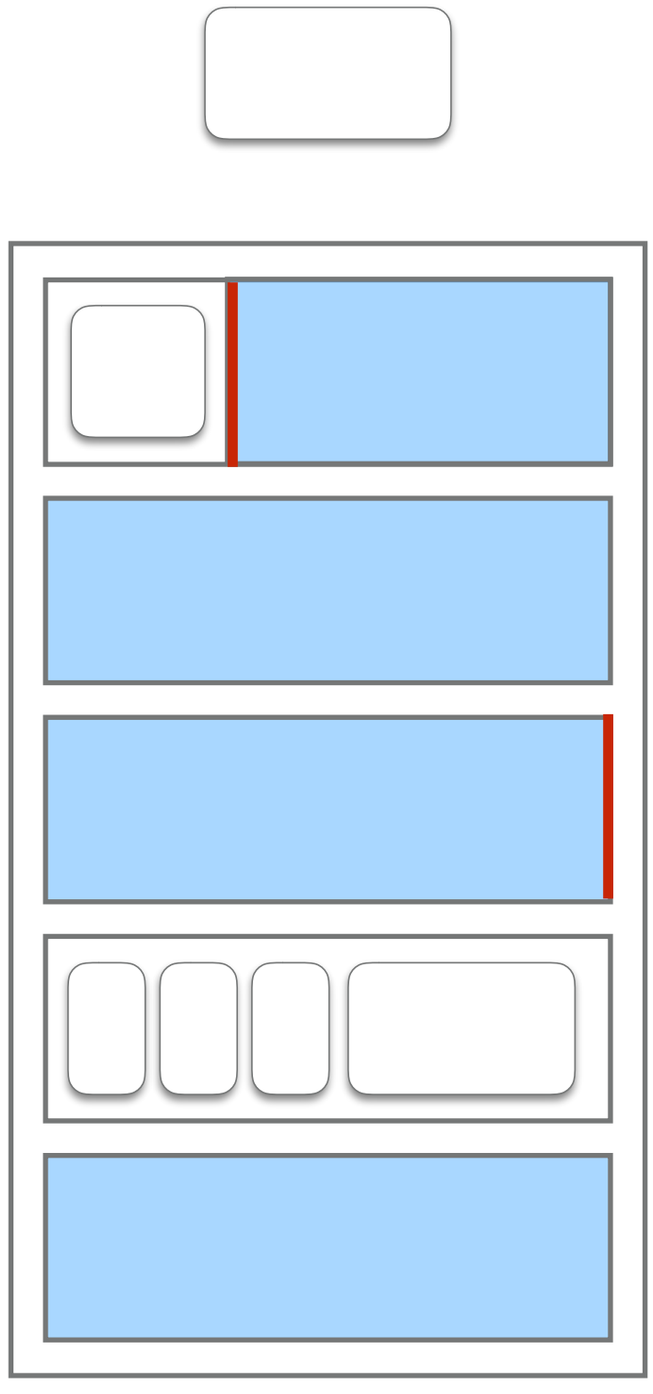


Immix



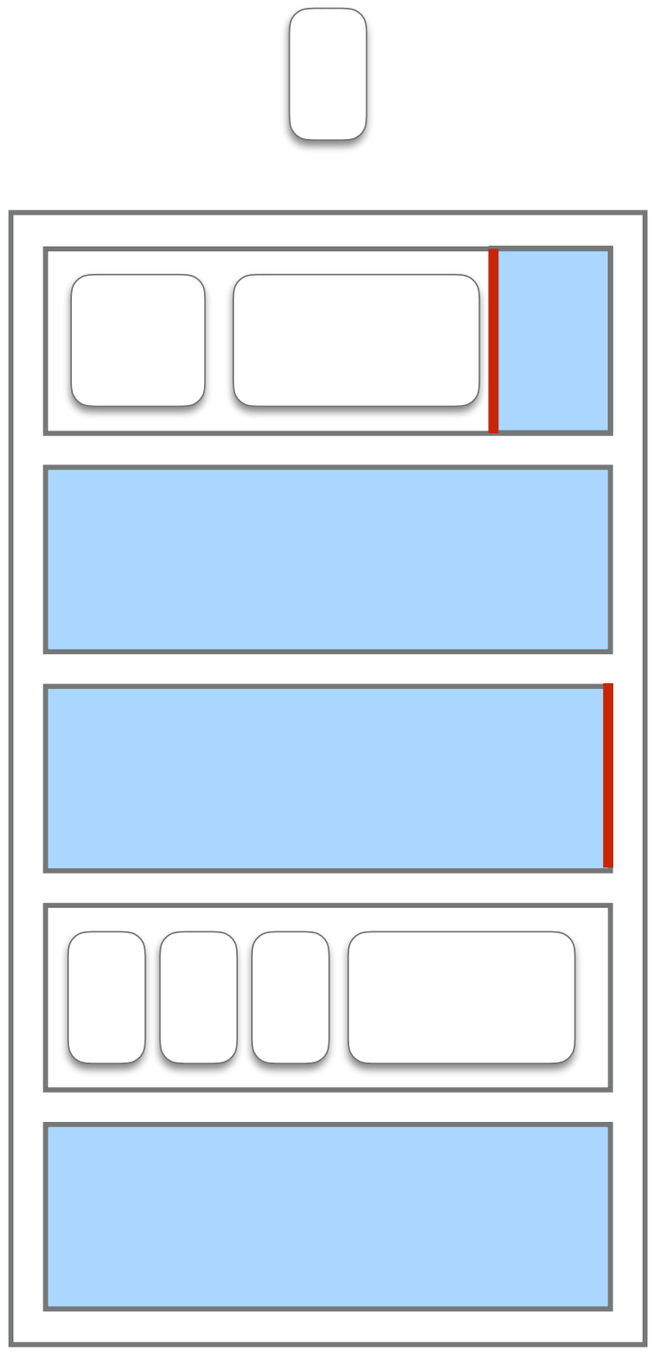
Allocation

Immix



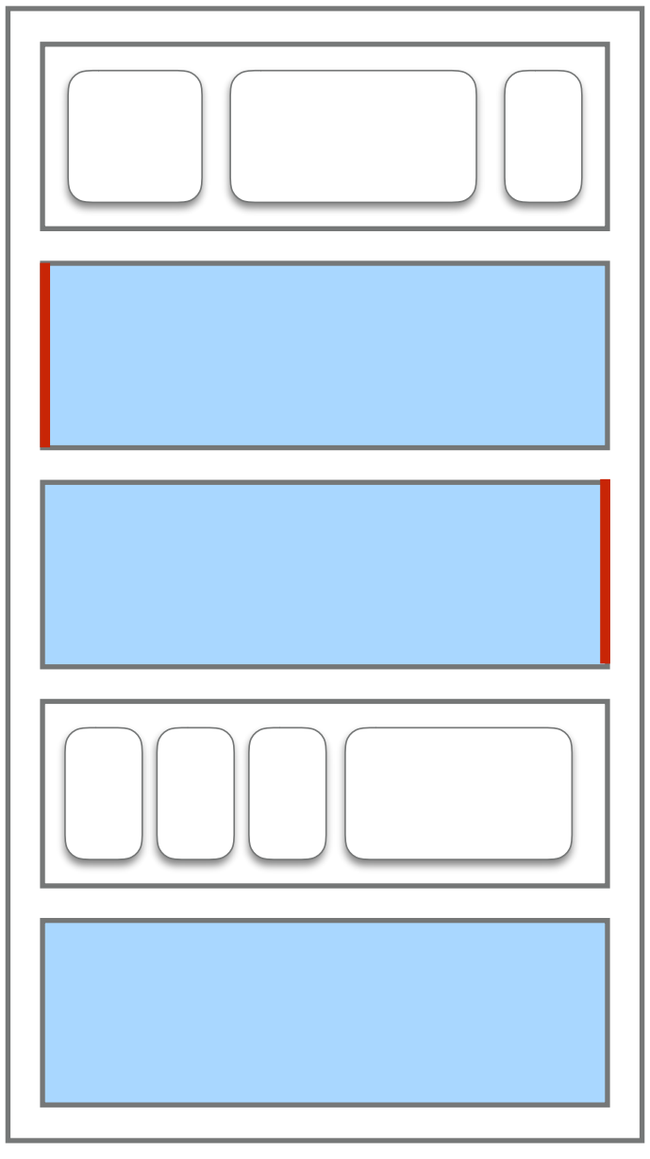
Allocation

Immix



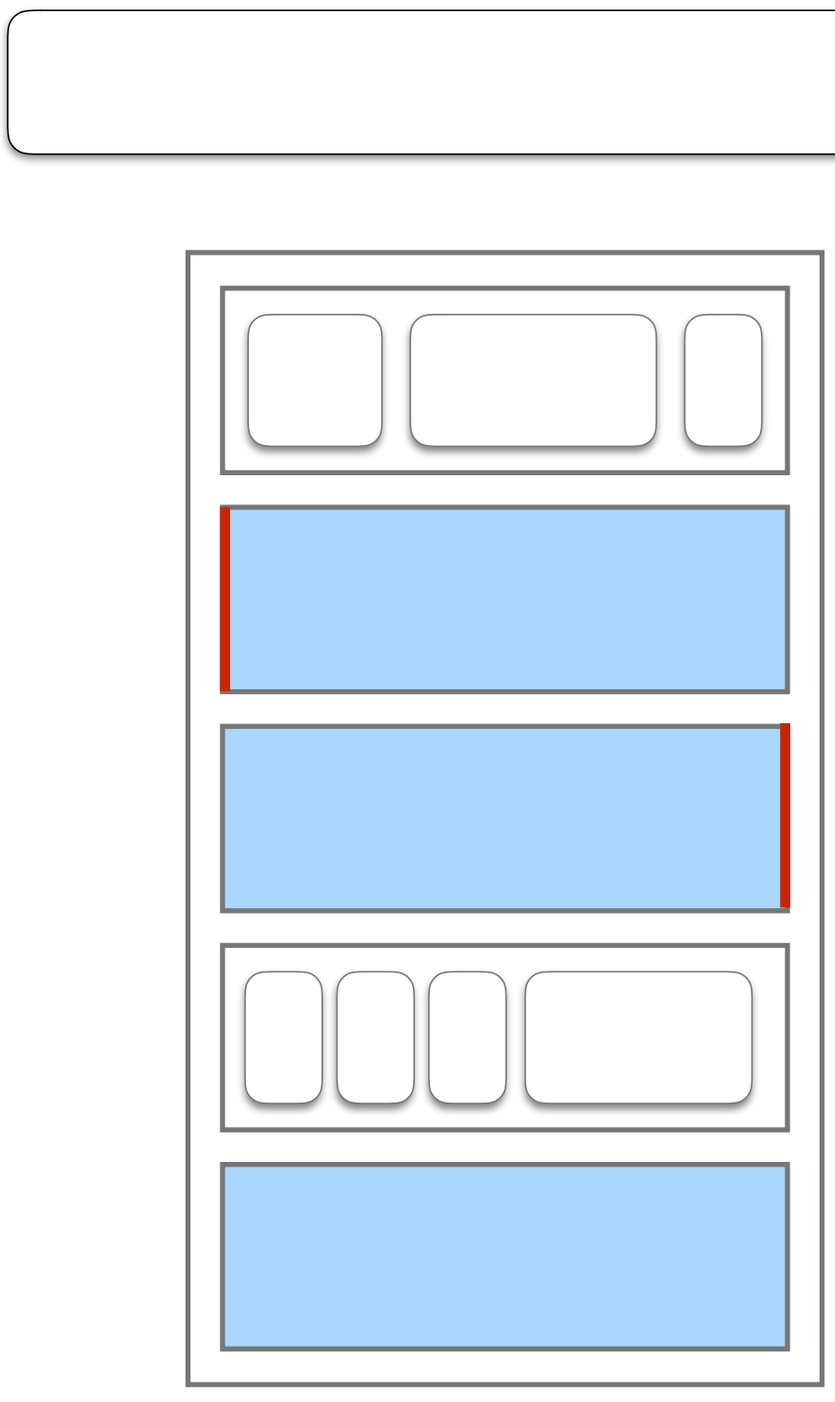
Allocation

Immix



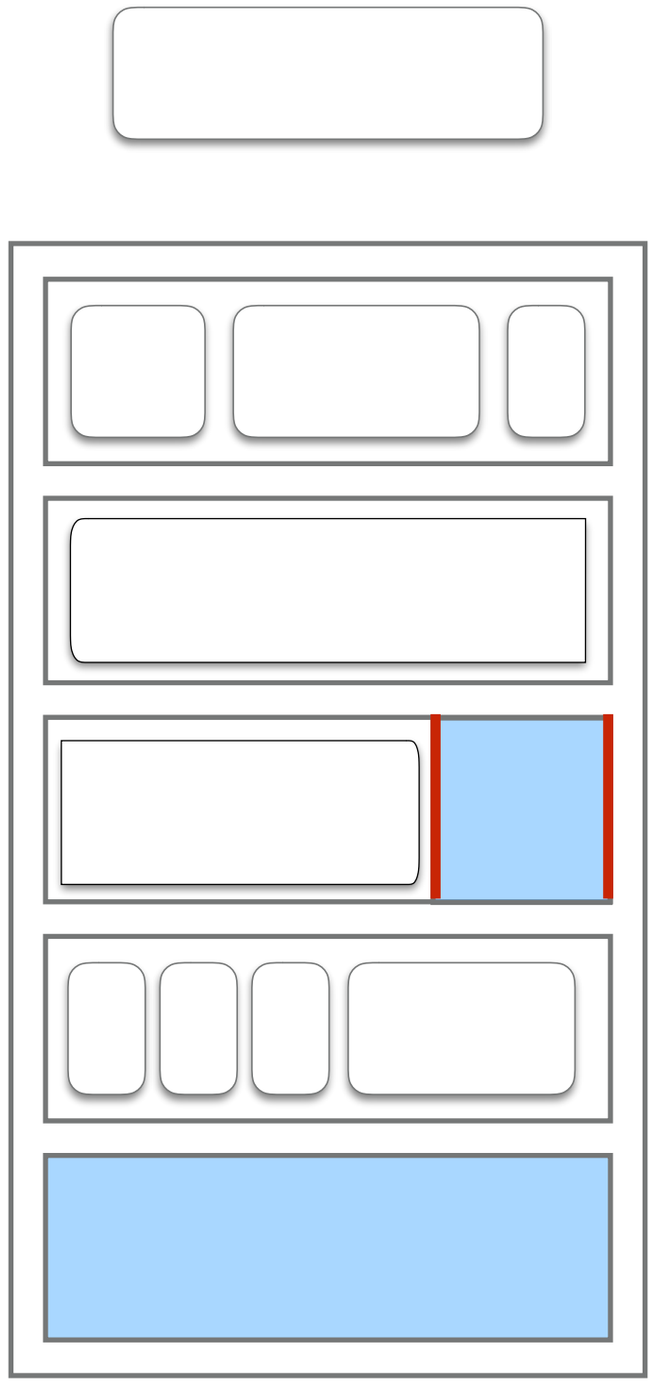
Allocation

Immix



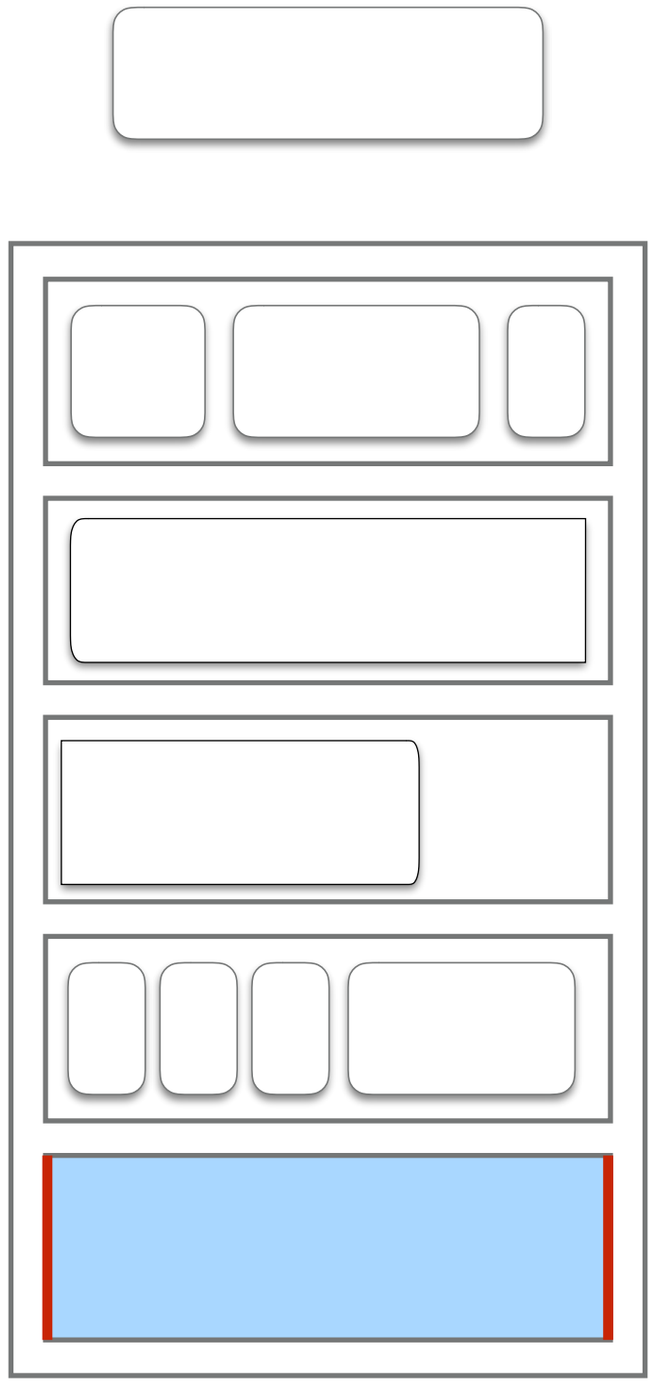
Allocation

Immix



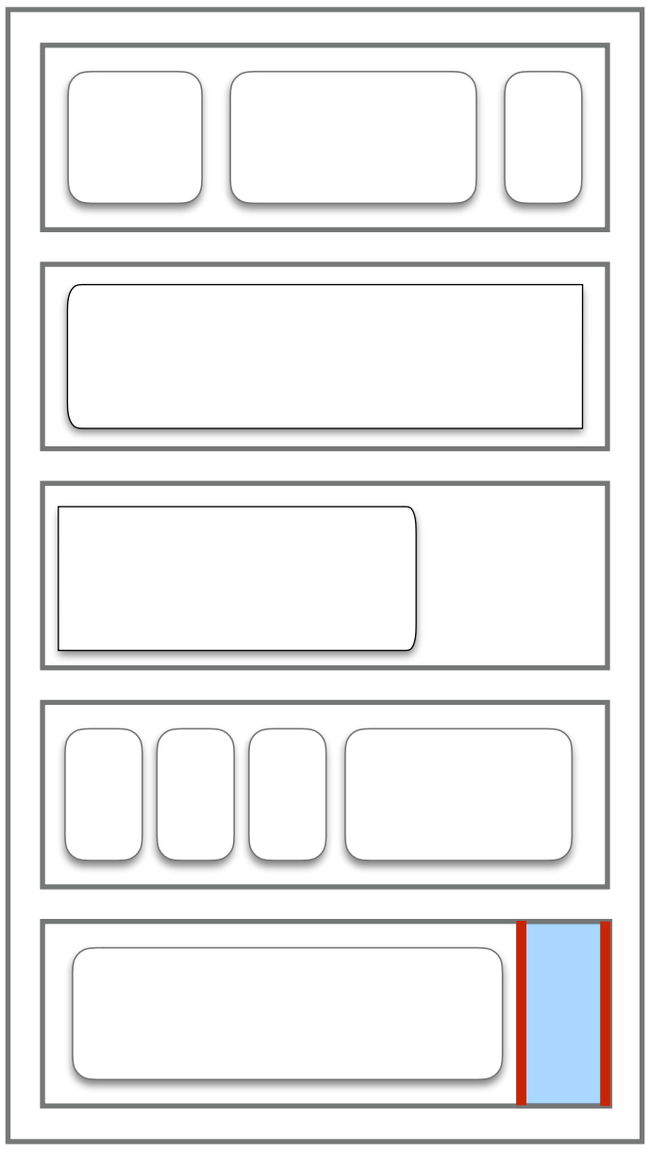
Allocation

Immix



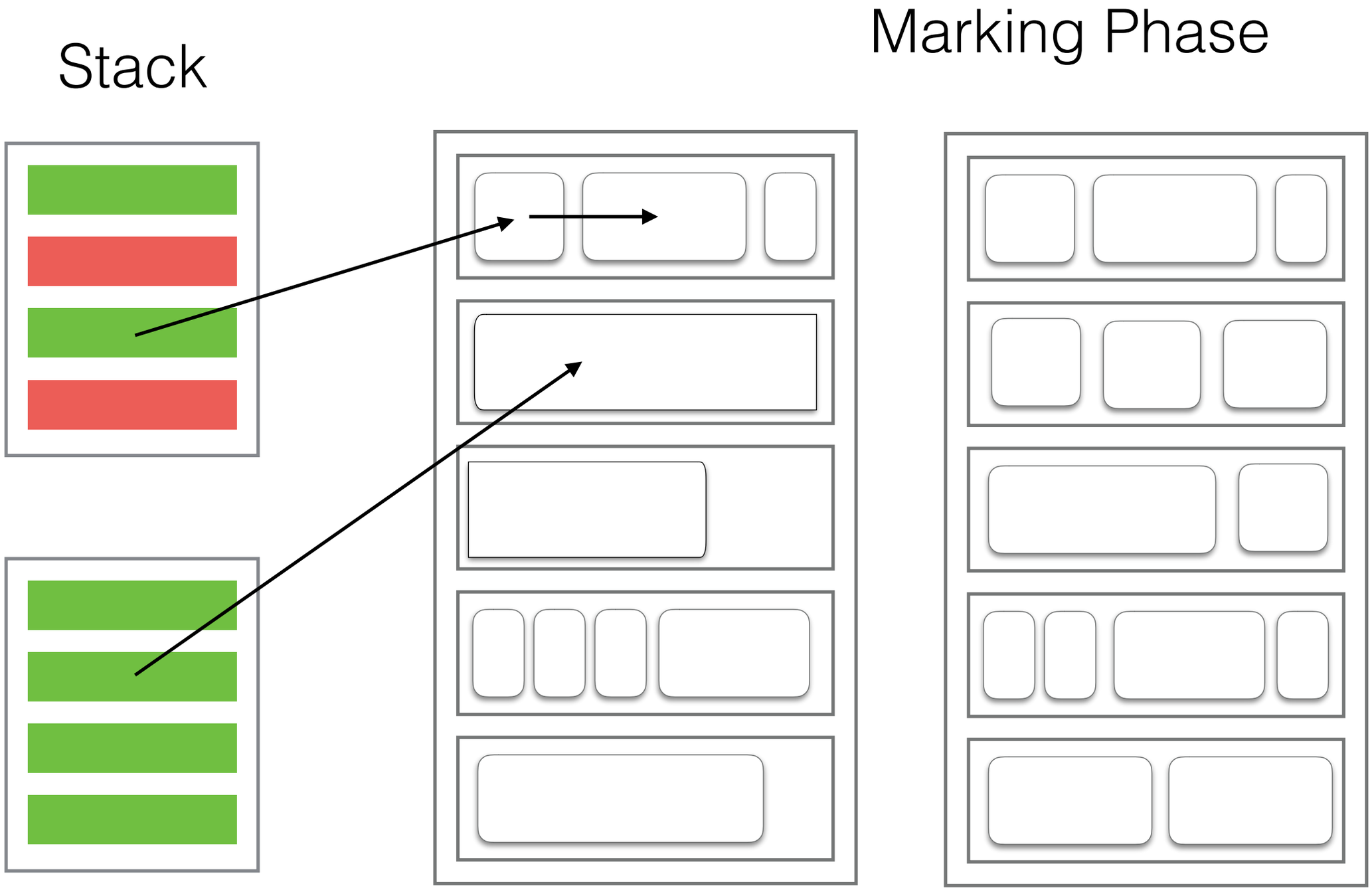
Allocation

Immix

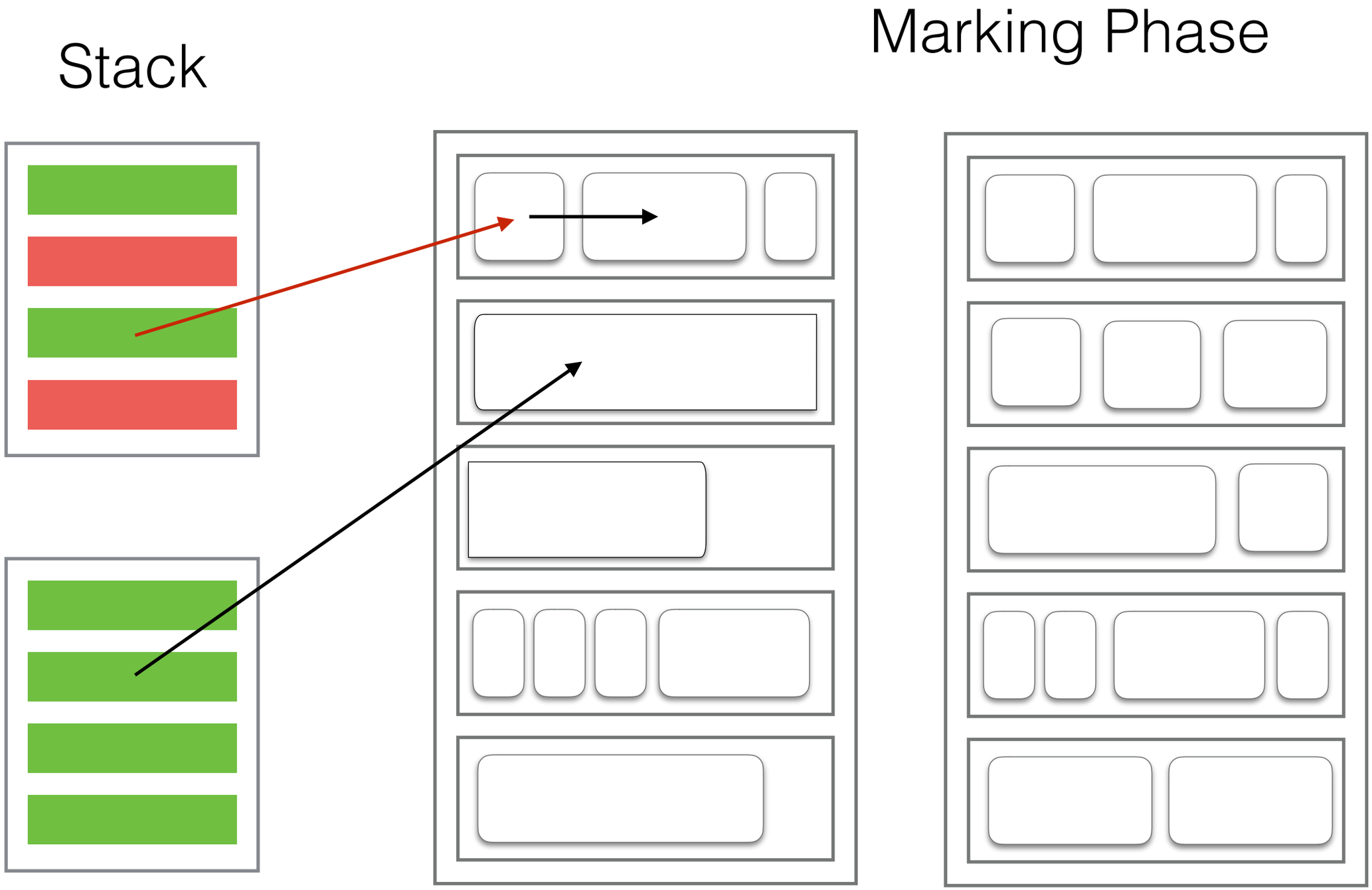


Allocation

Immix



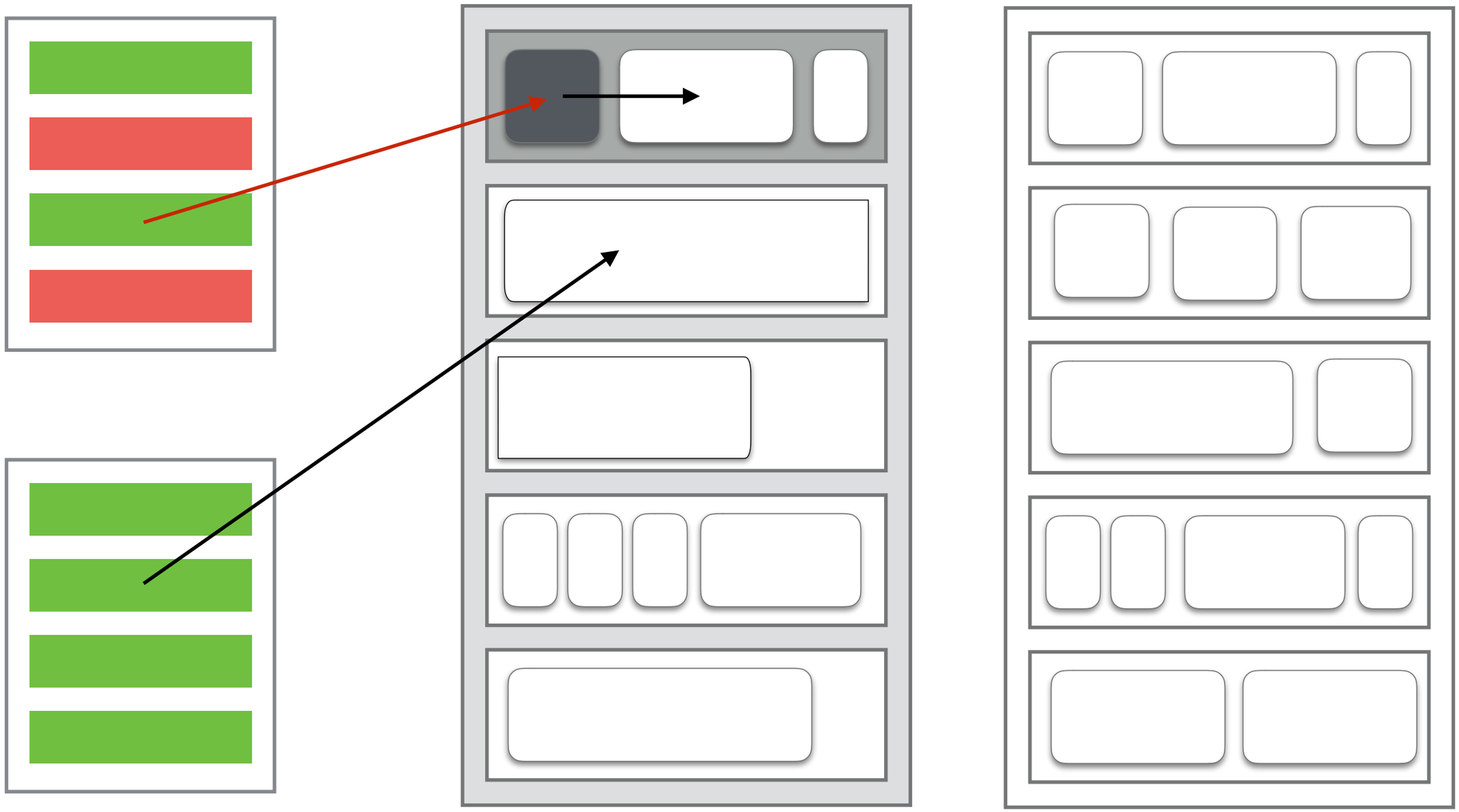
Immix



Immix

Stack

Marking Phase

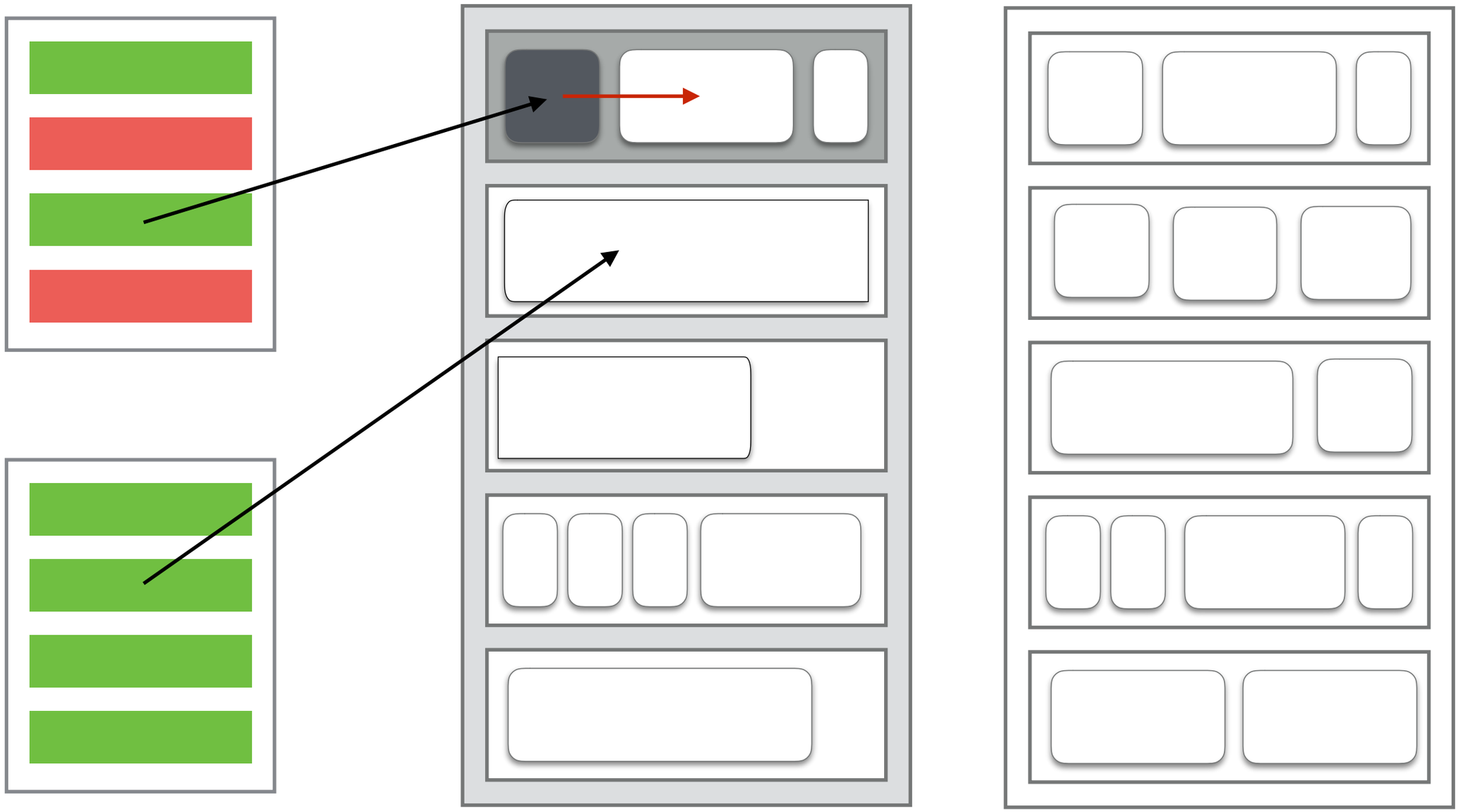


Modules

Immix

Stack

Marking Phase

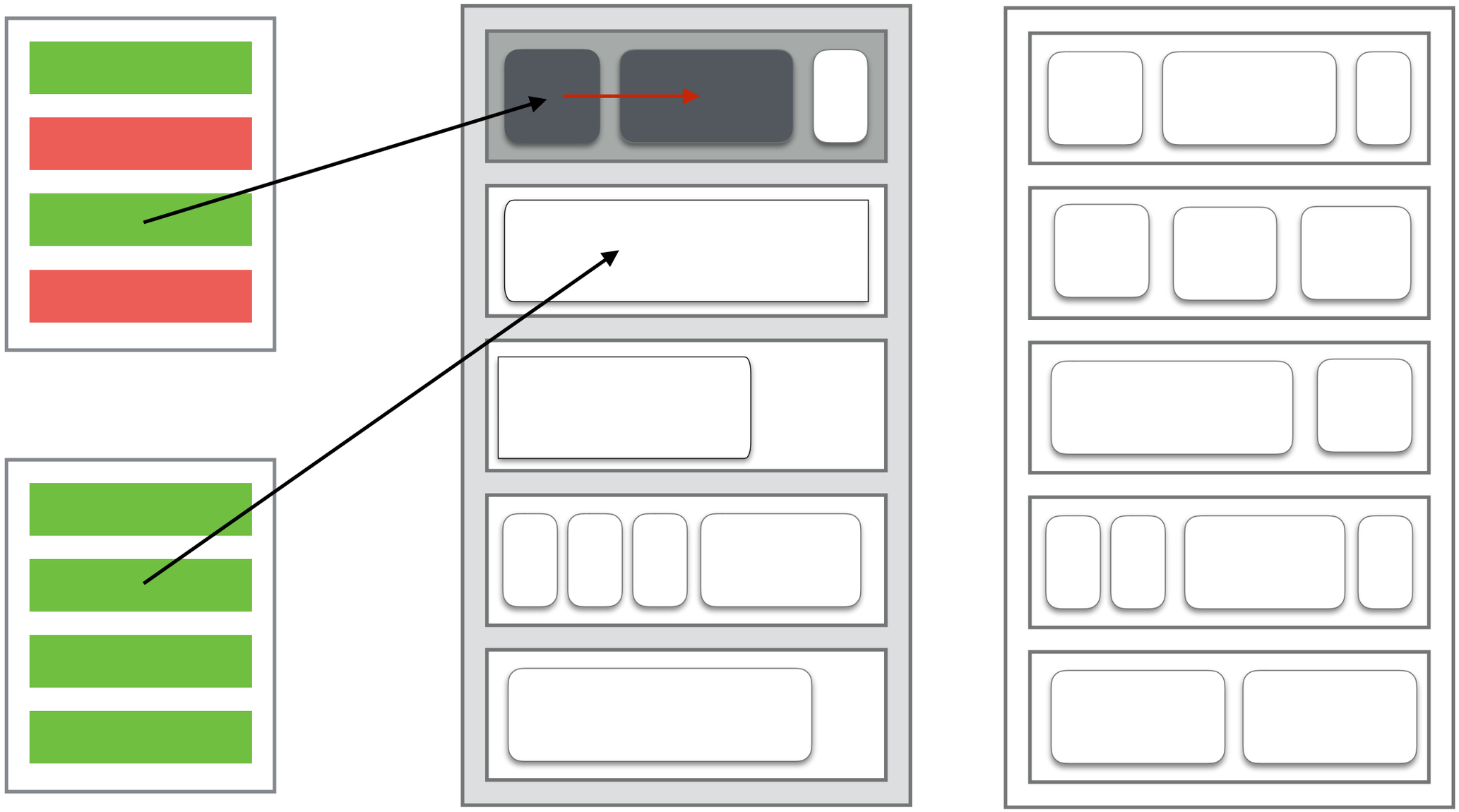


Modules

Immix

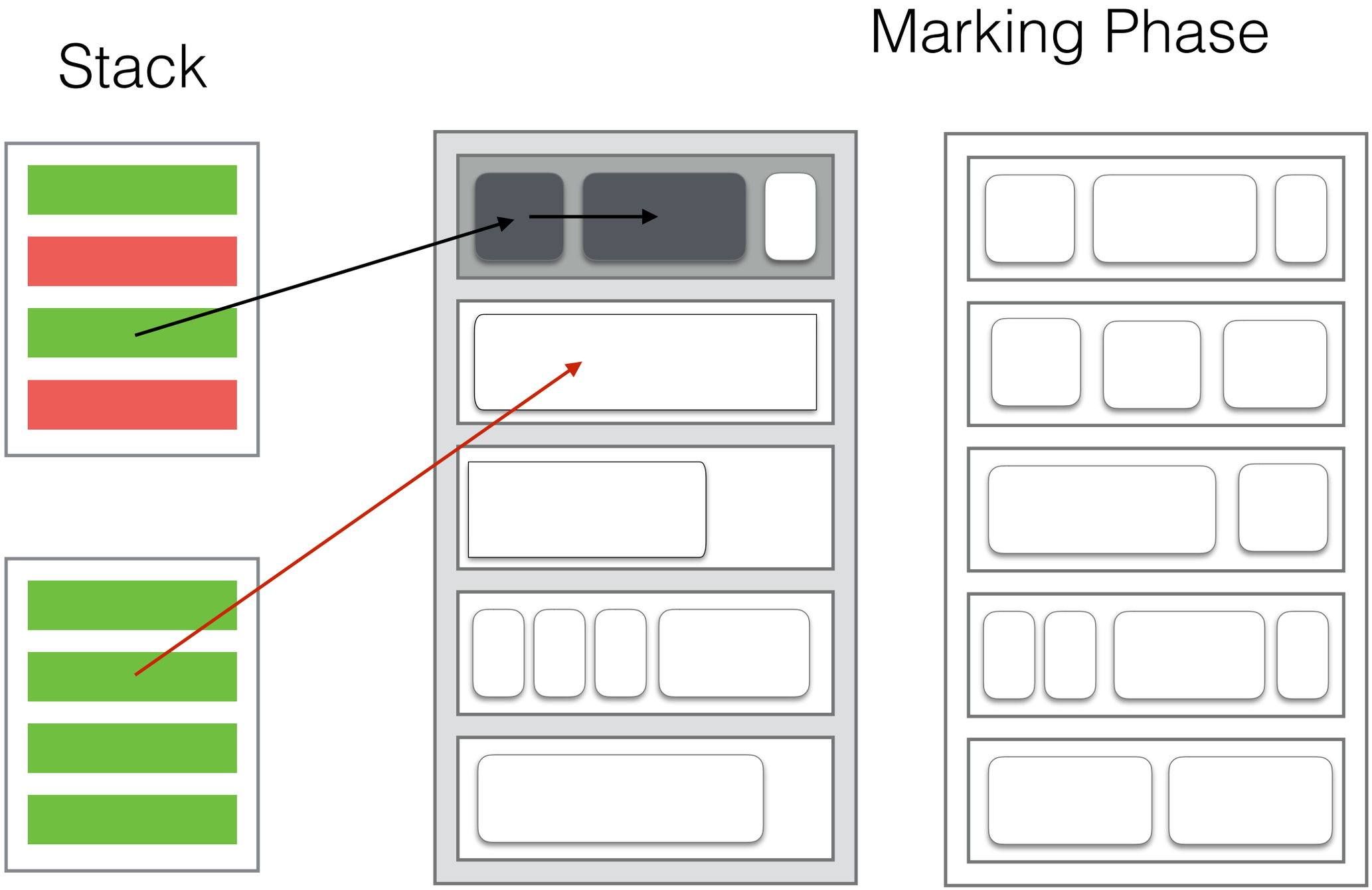
Stack

Marking Phase



Modules

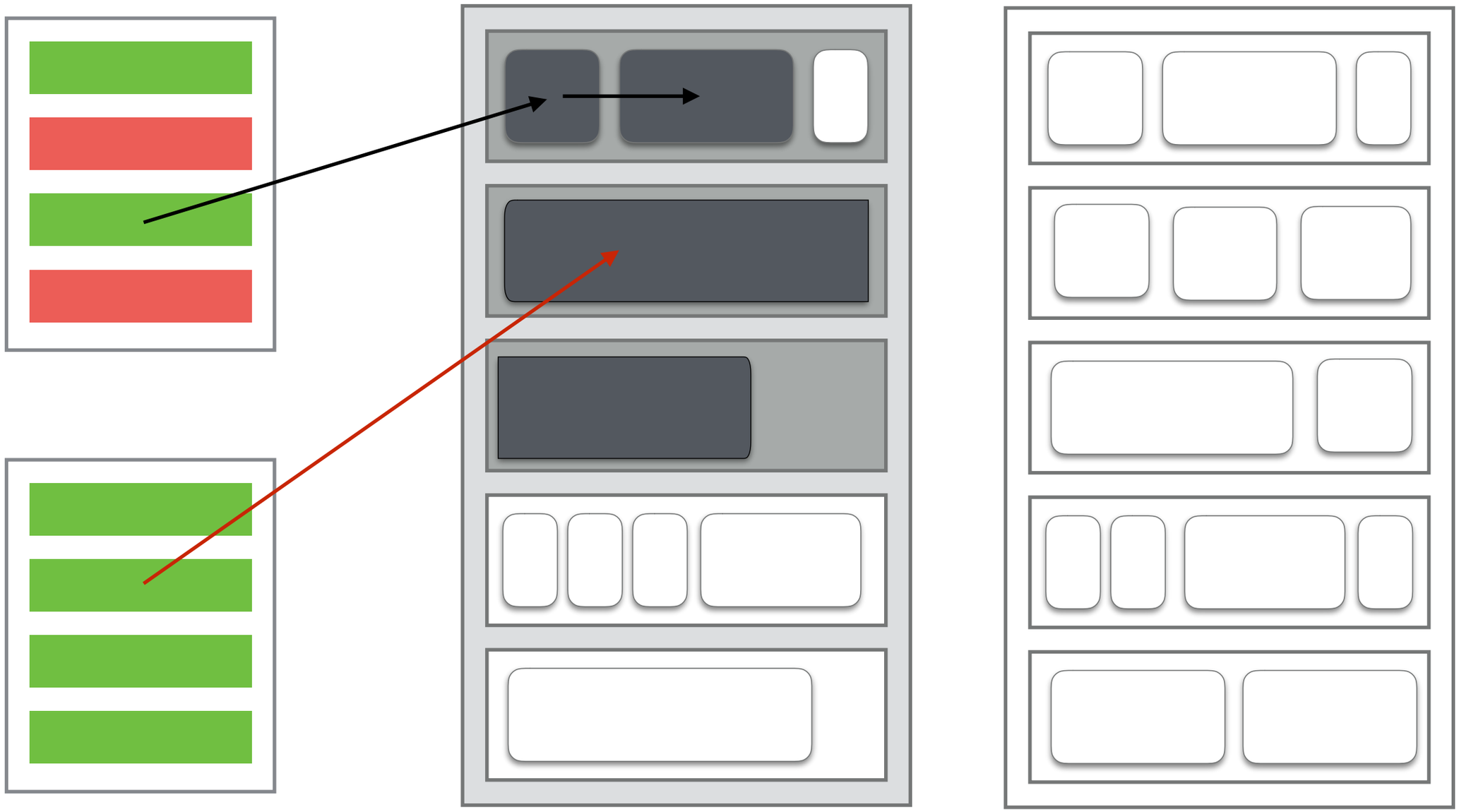
Immix



Immix

Stack

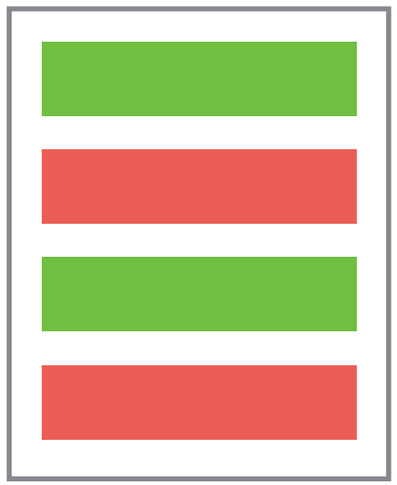
Marking Phase

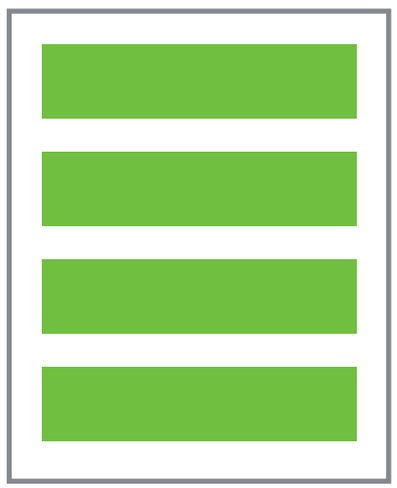


Modules

Immix

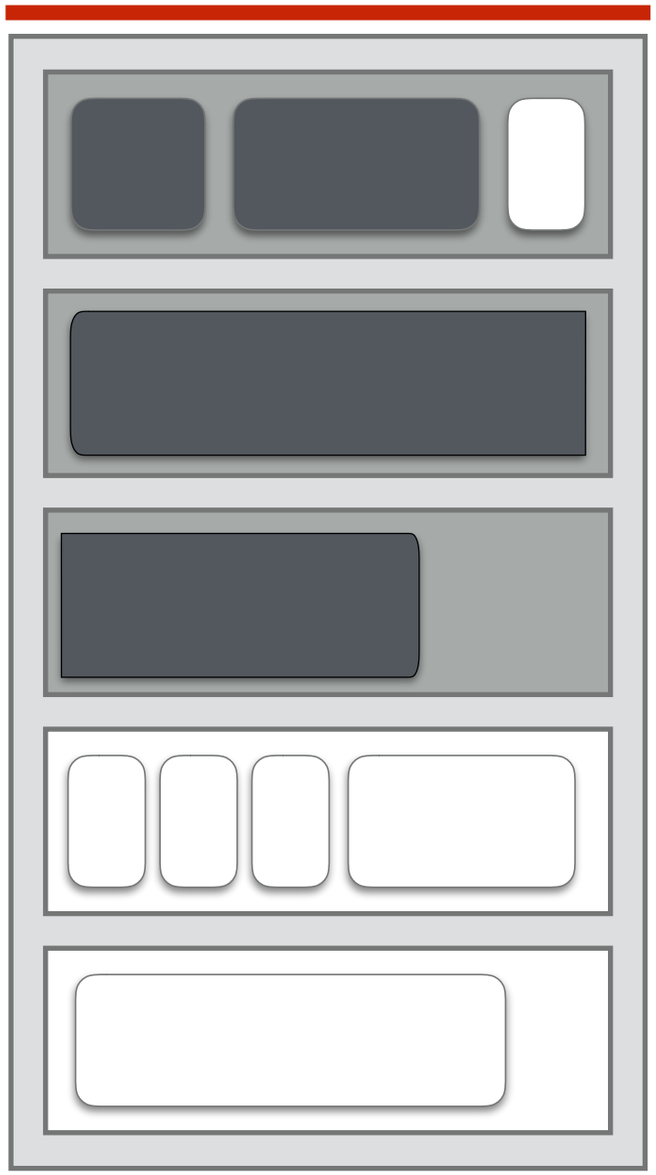
Stack

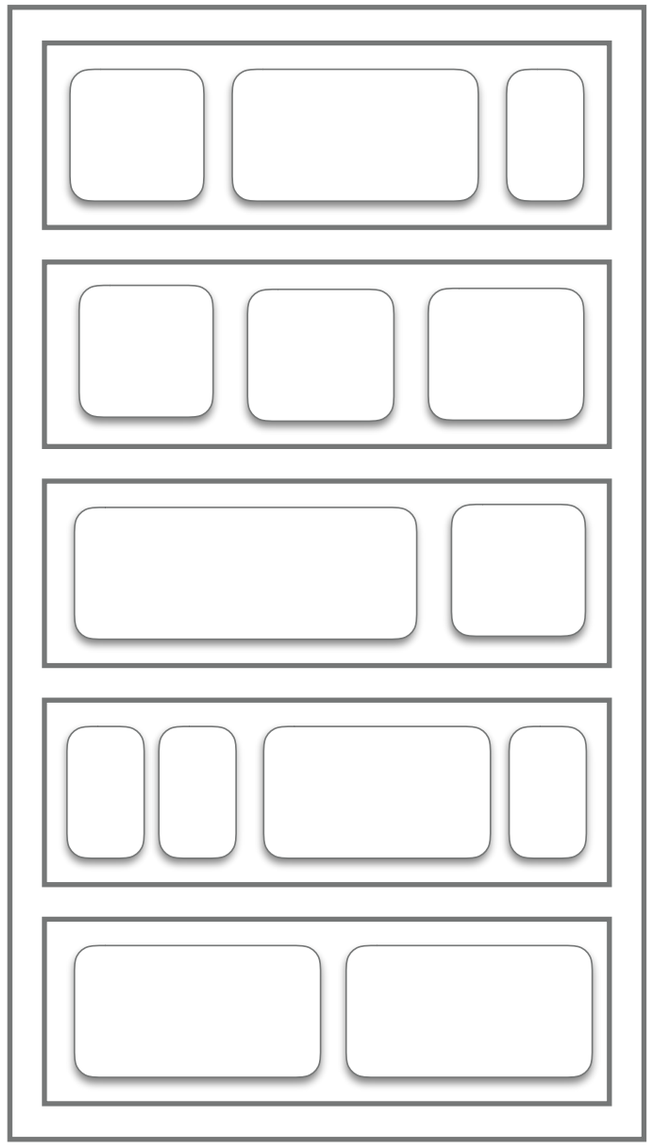




Modules

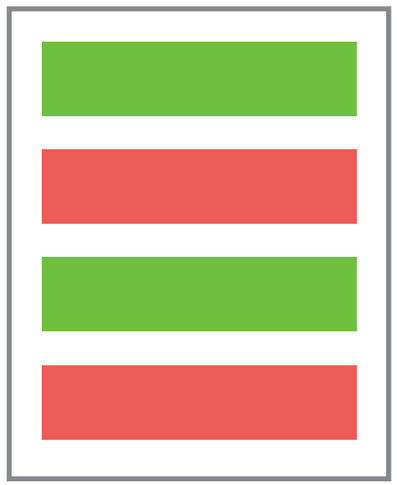
Sweeping Phase

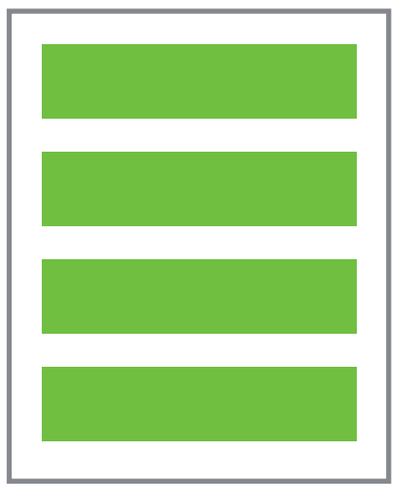




Immix

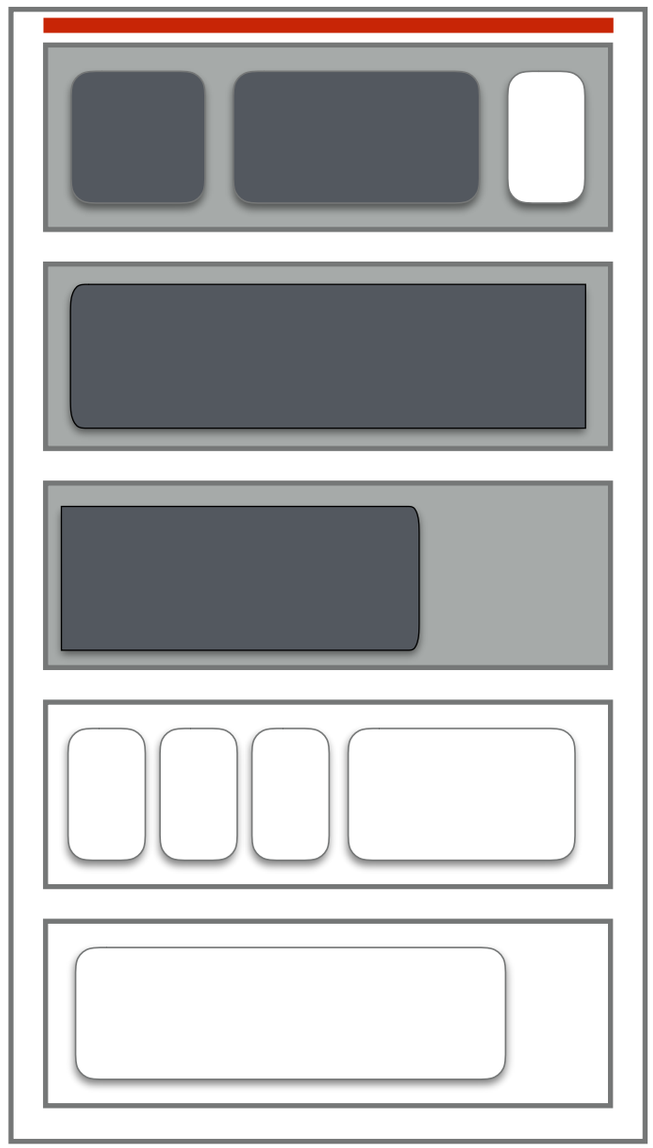
Stack

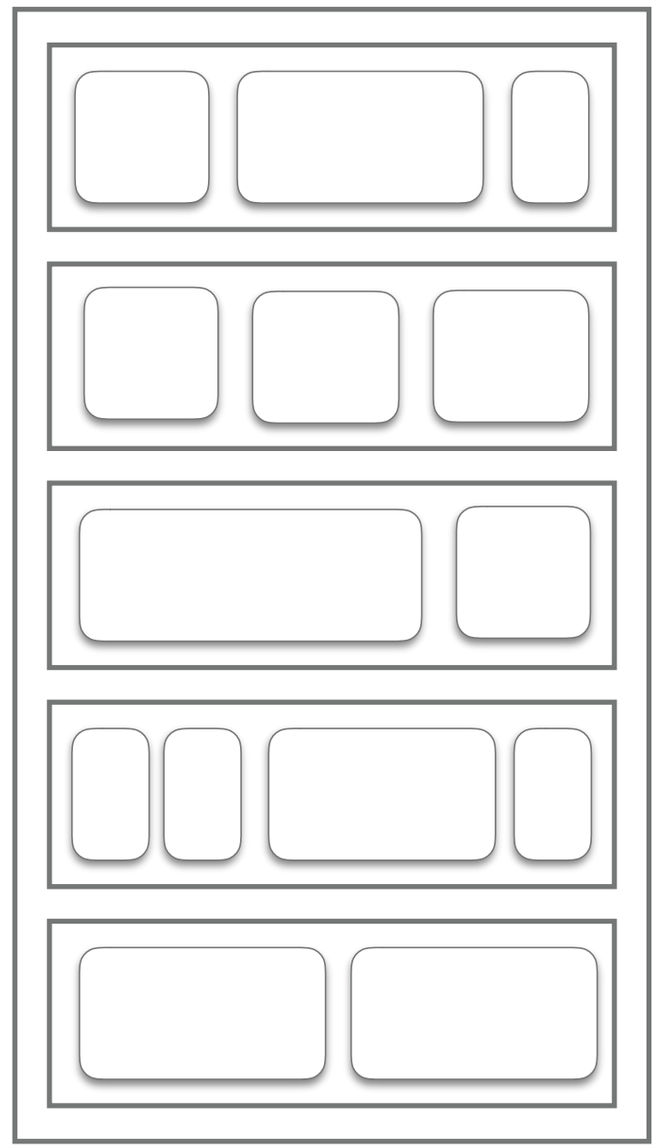




Modules

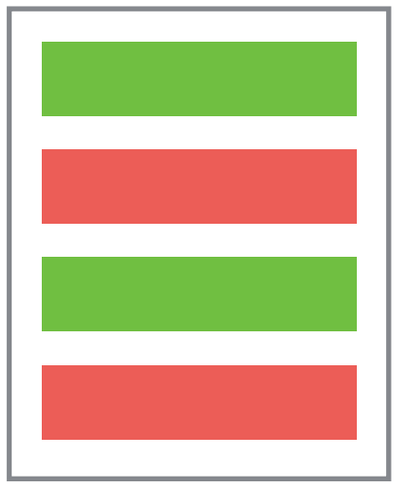
Sweeping Phase

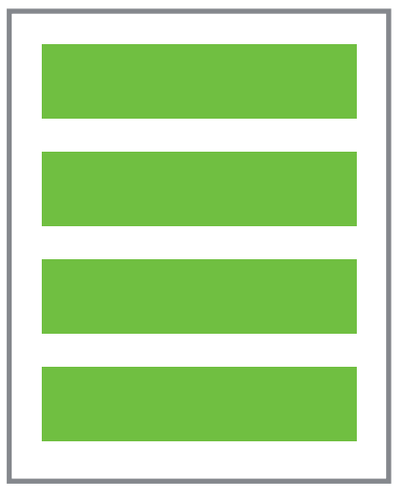




Immix

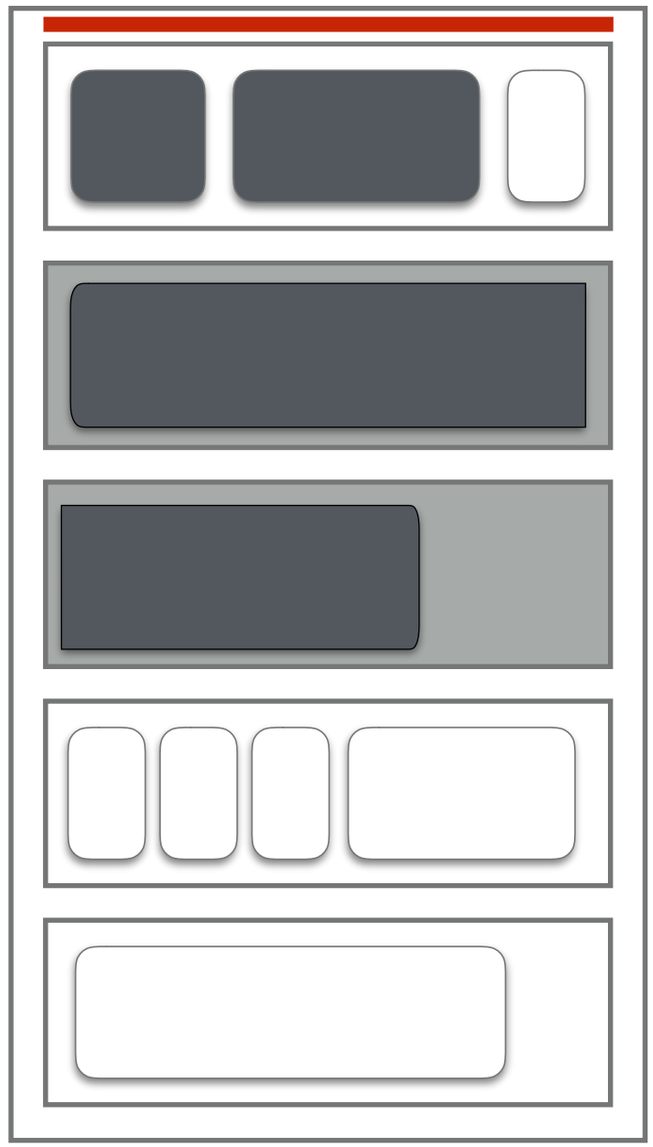
Stack

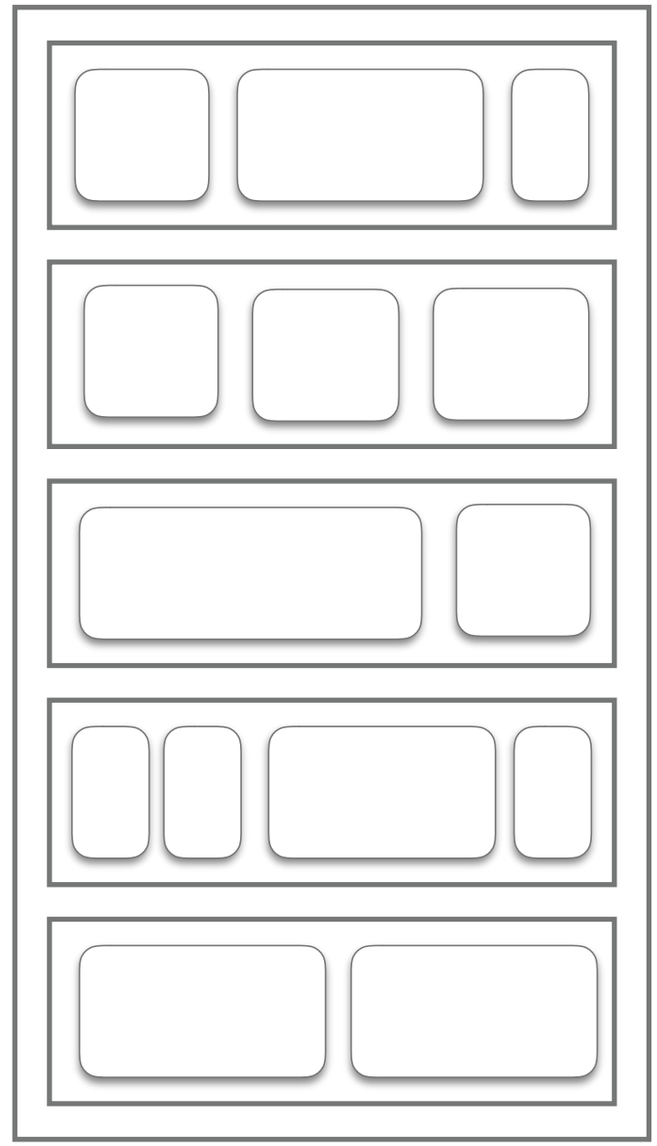




Modules

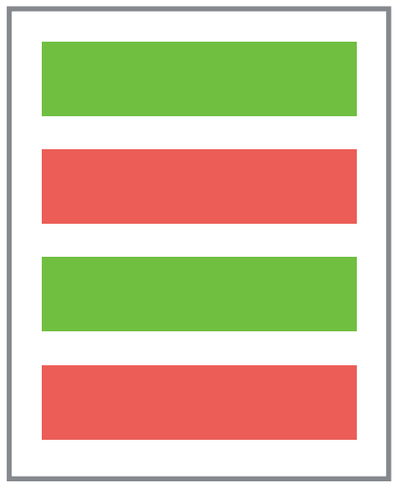
Sweeping Phase

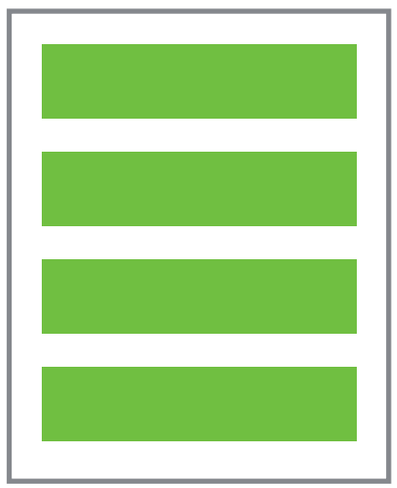




Immix

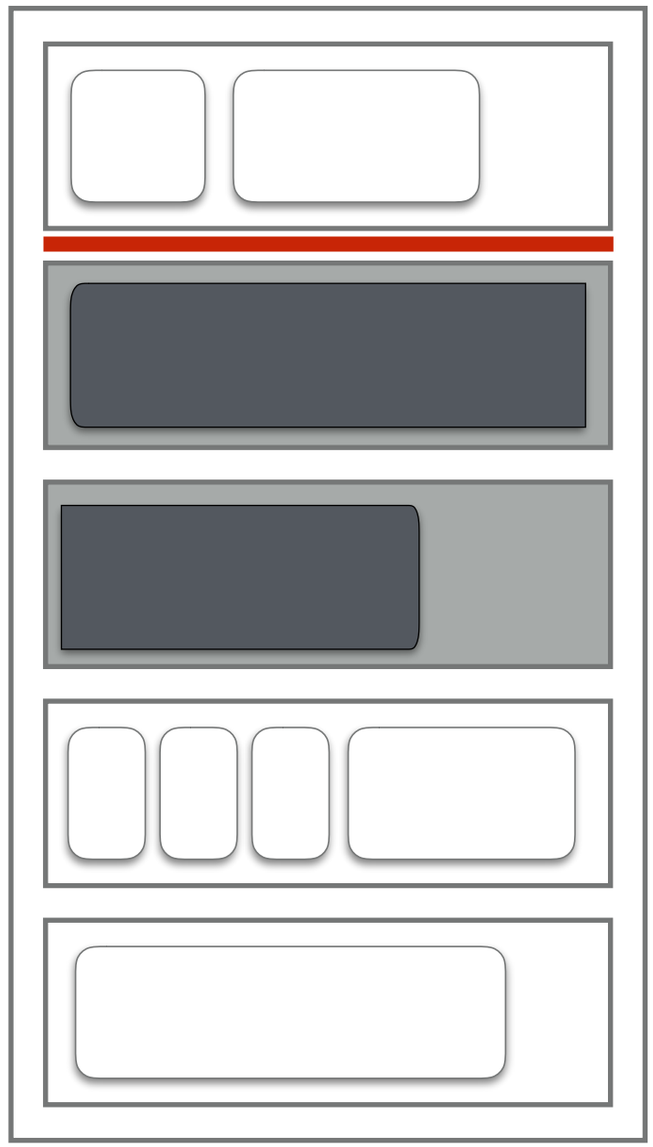
Stack

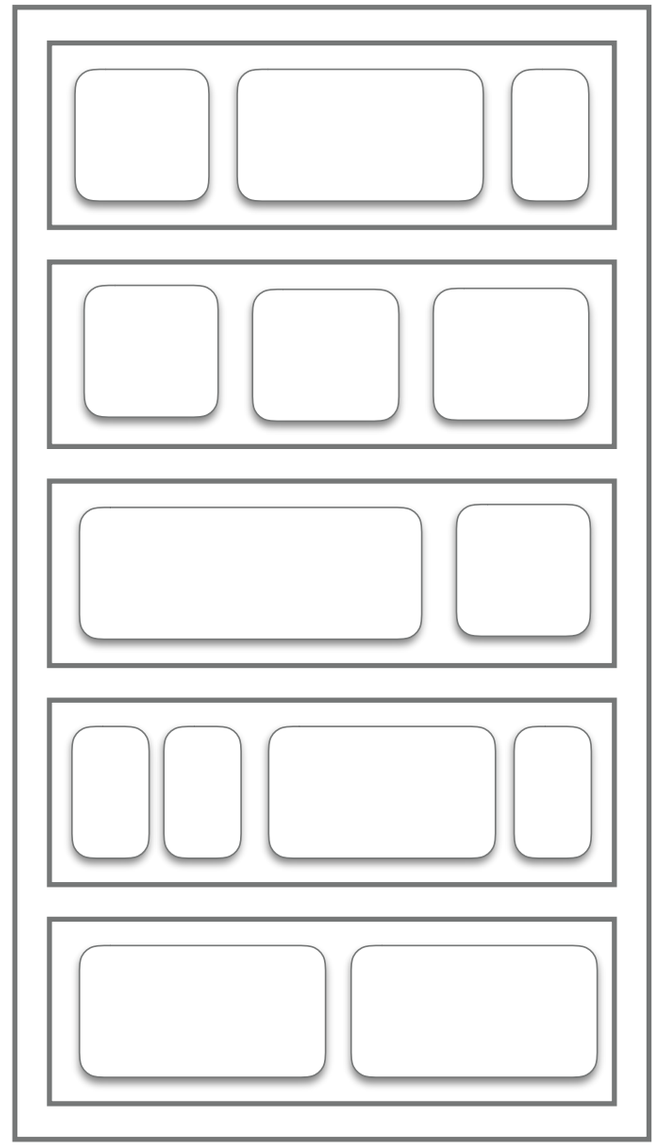




Modules

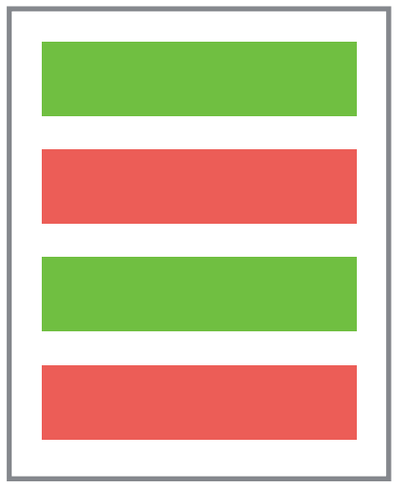
Sweeping Phase

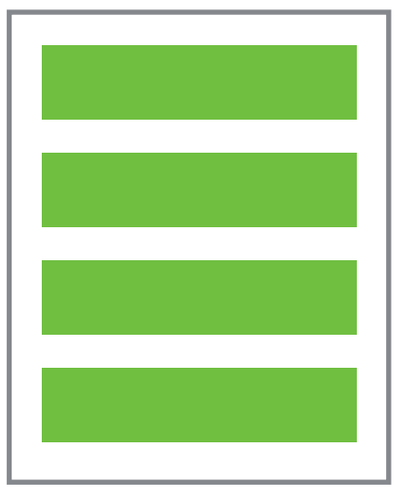




Immix

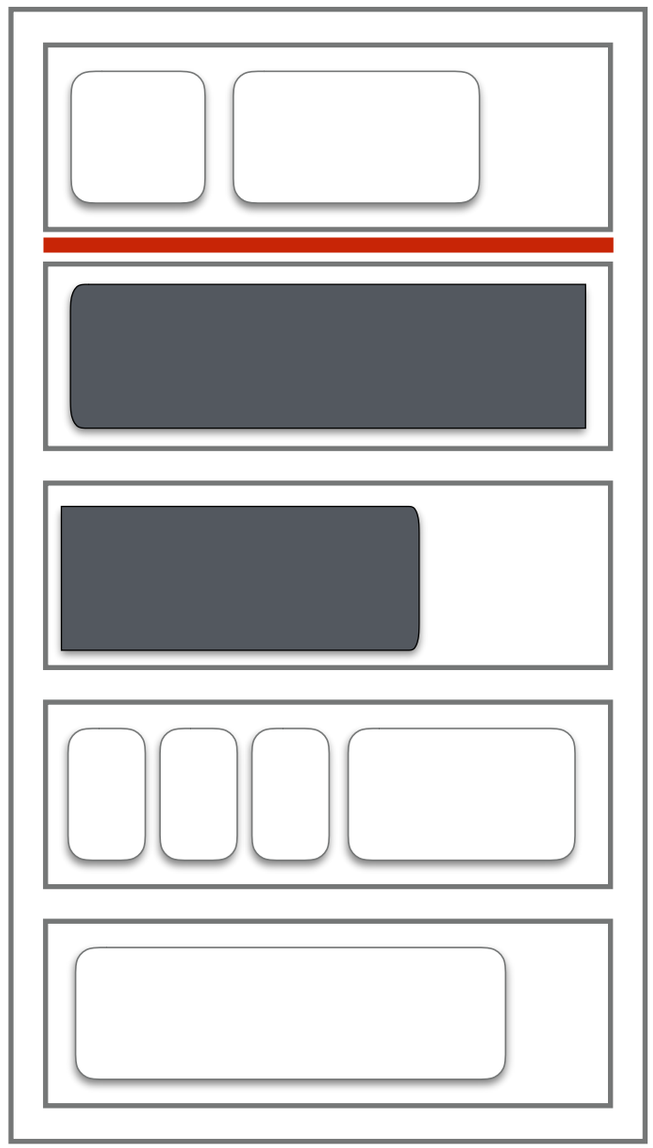
Stack

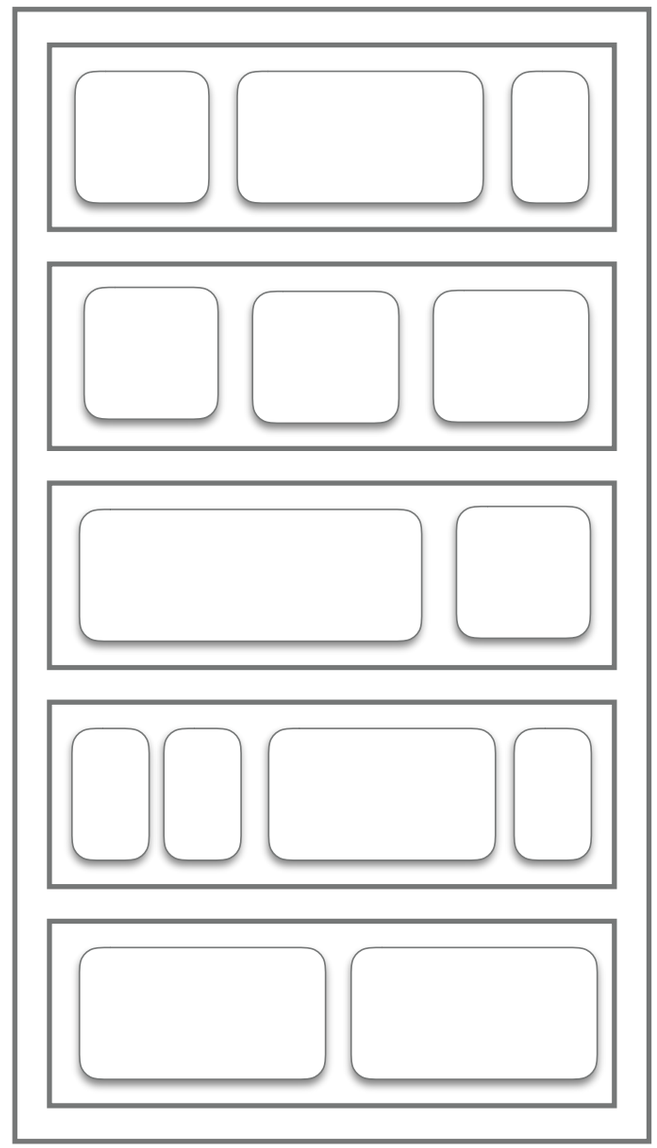




Modules

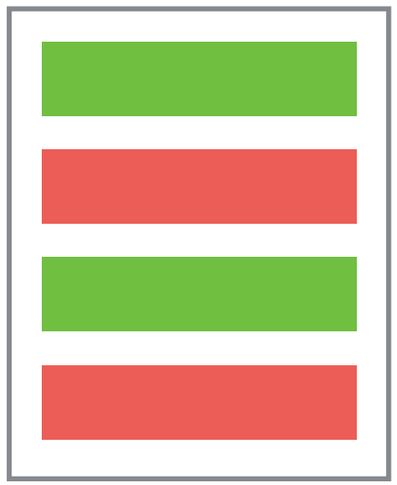
Sweeping Phase

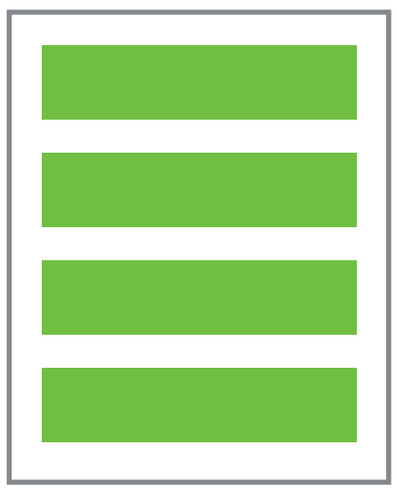




Immix

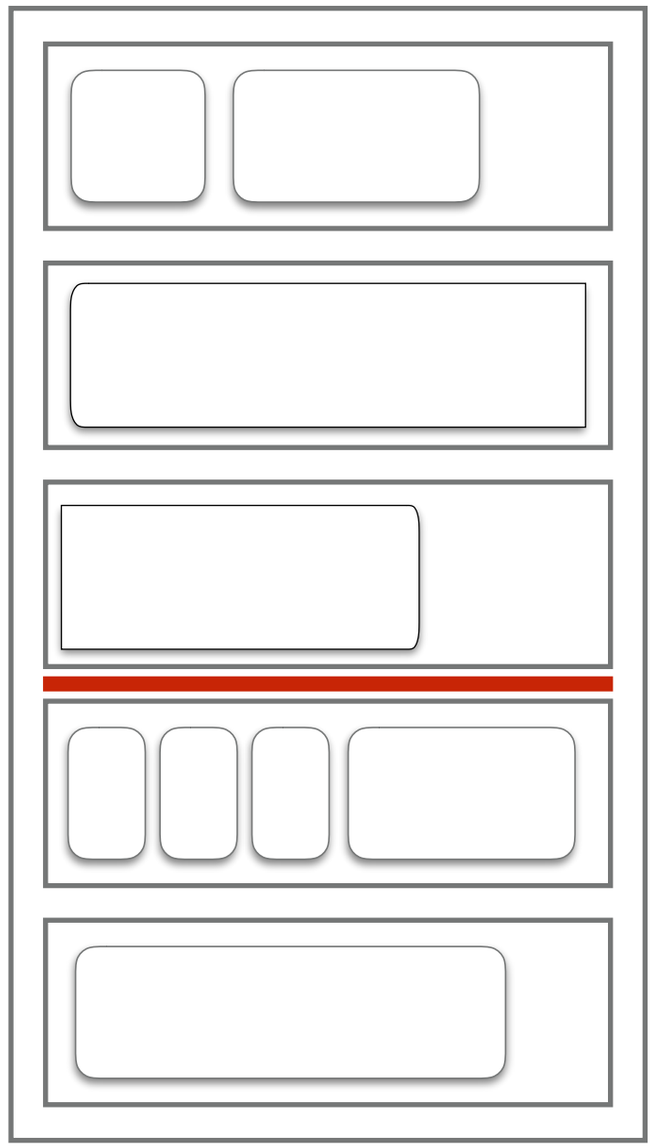
Stack

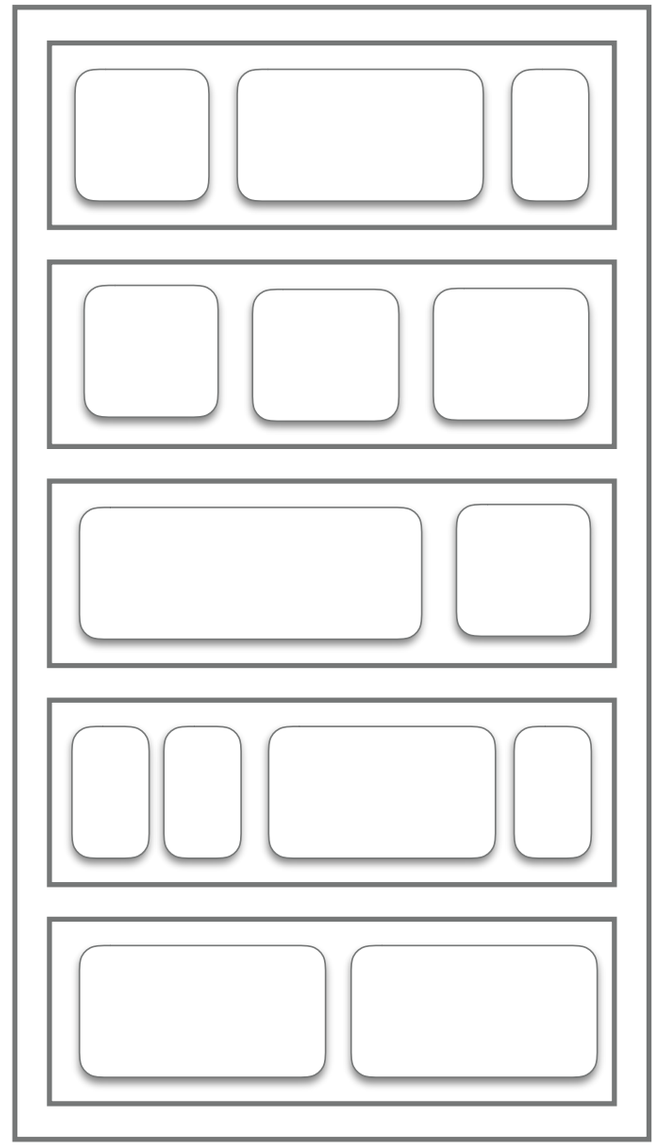




Modules

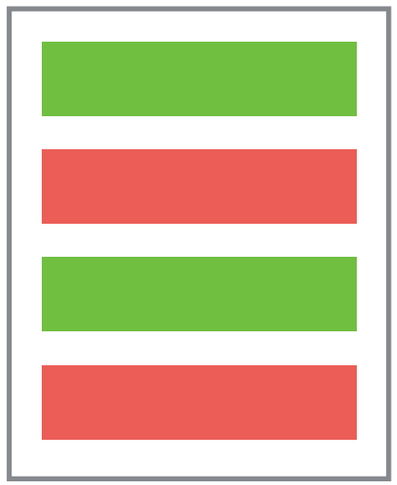
Sweeping Phase

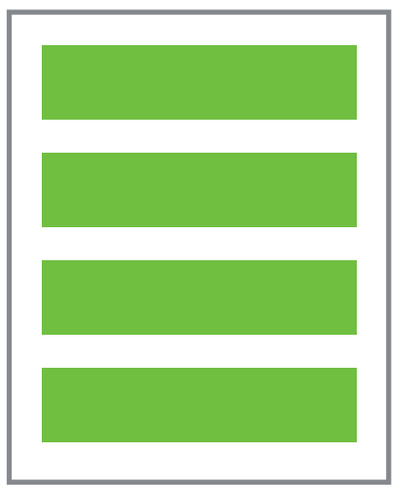




Immix

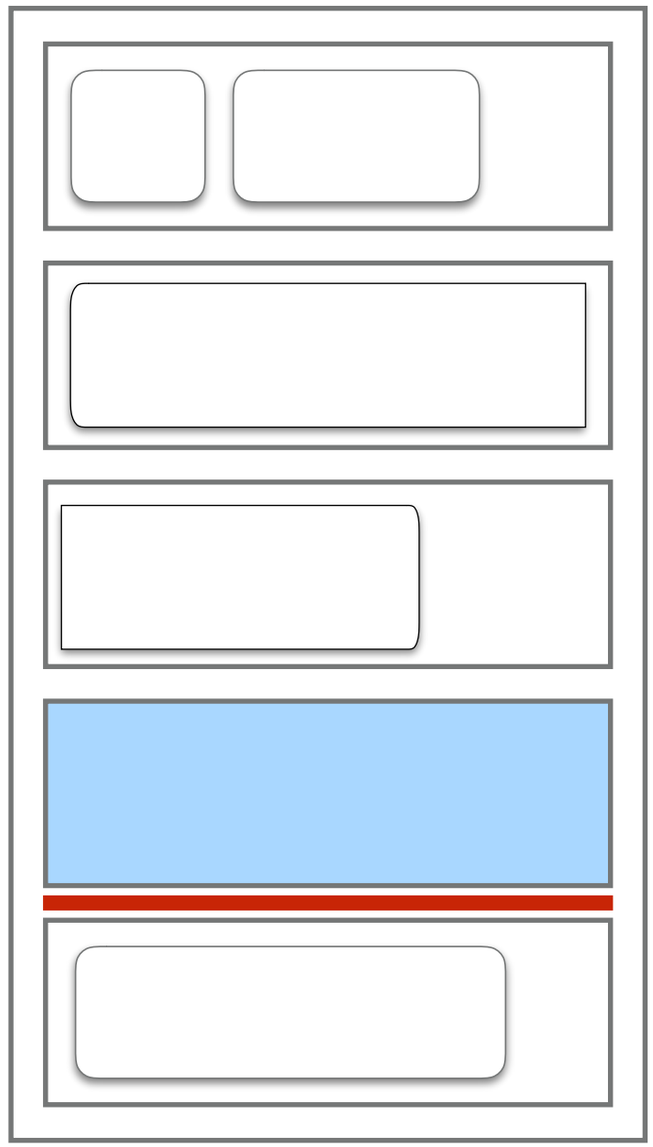
Stack

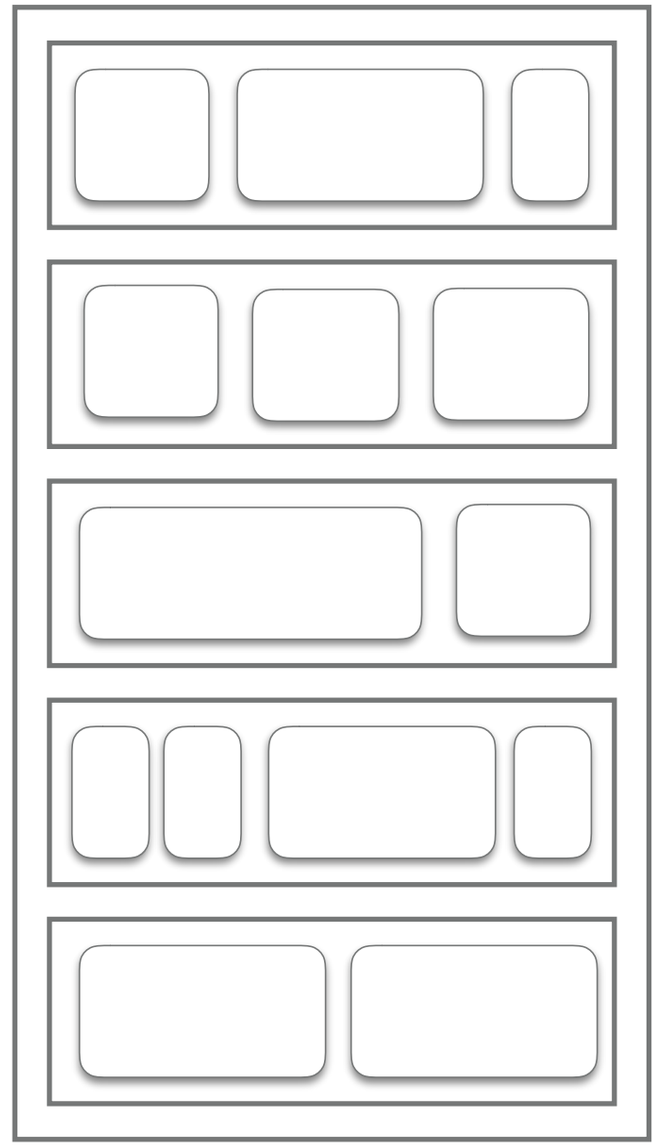




Modules

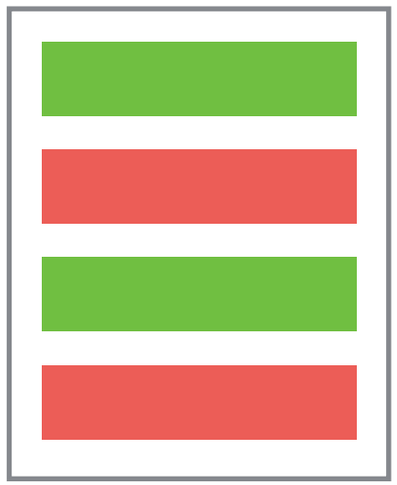
Sweeping Phase

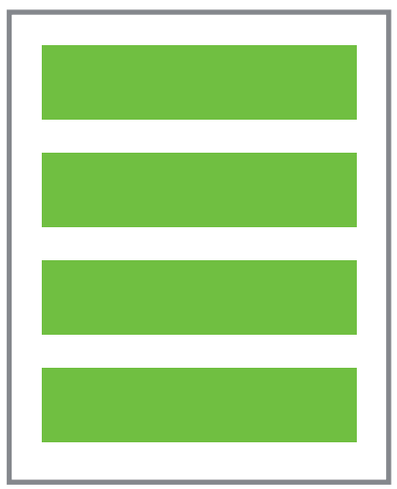




Immix

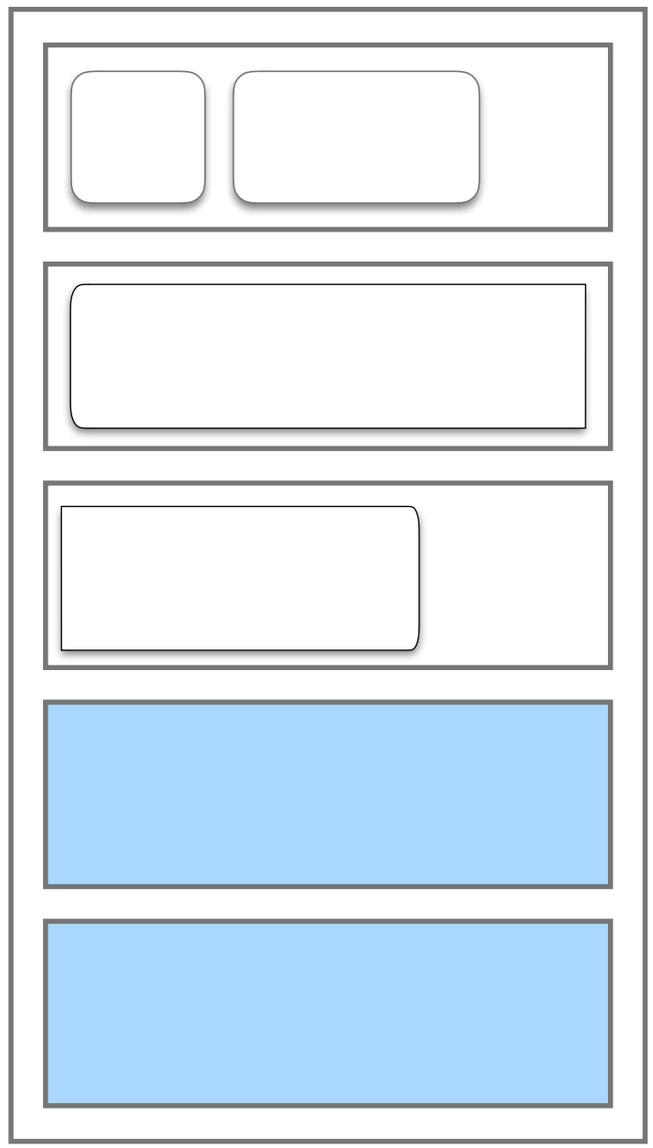
Stack

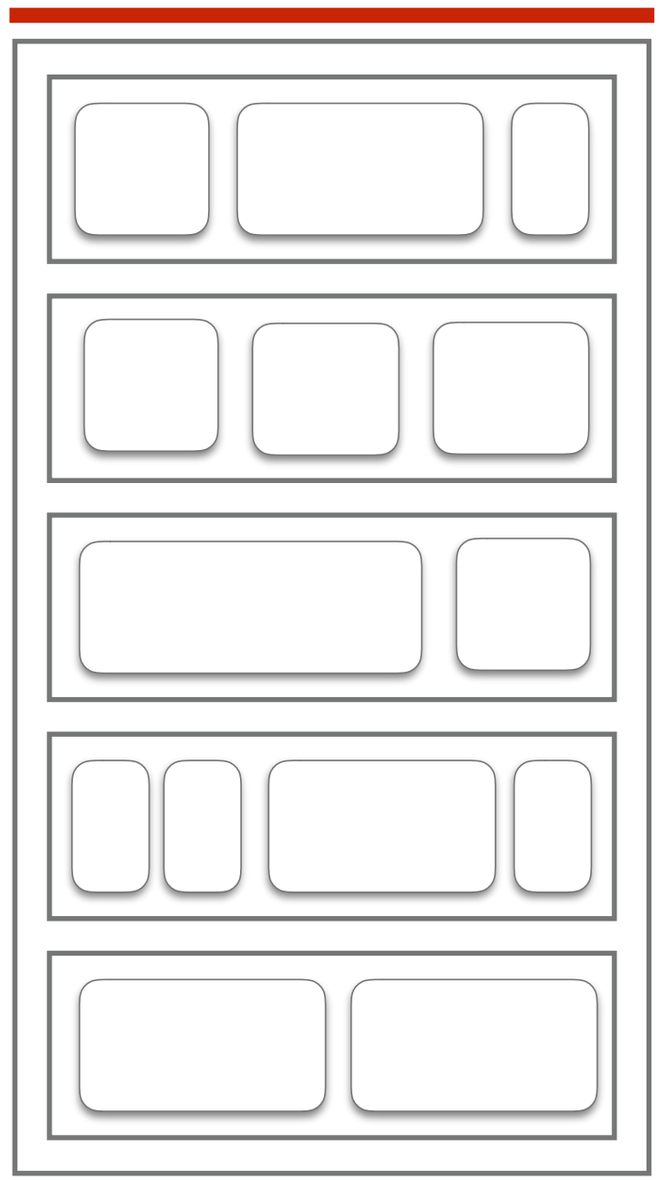




Modules

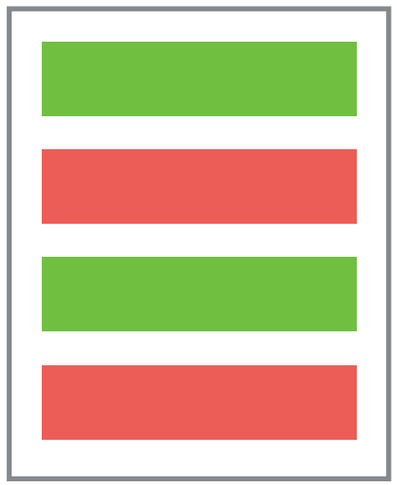
Sweeping Phase

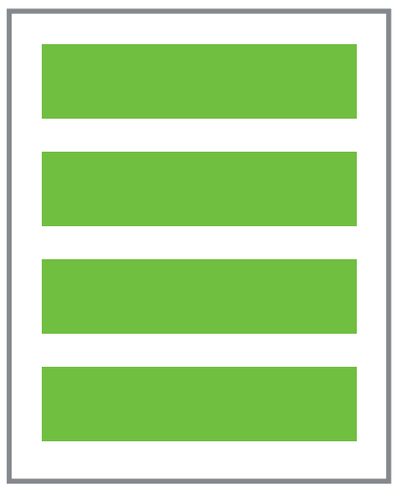




Immix

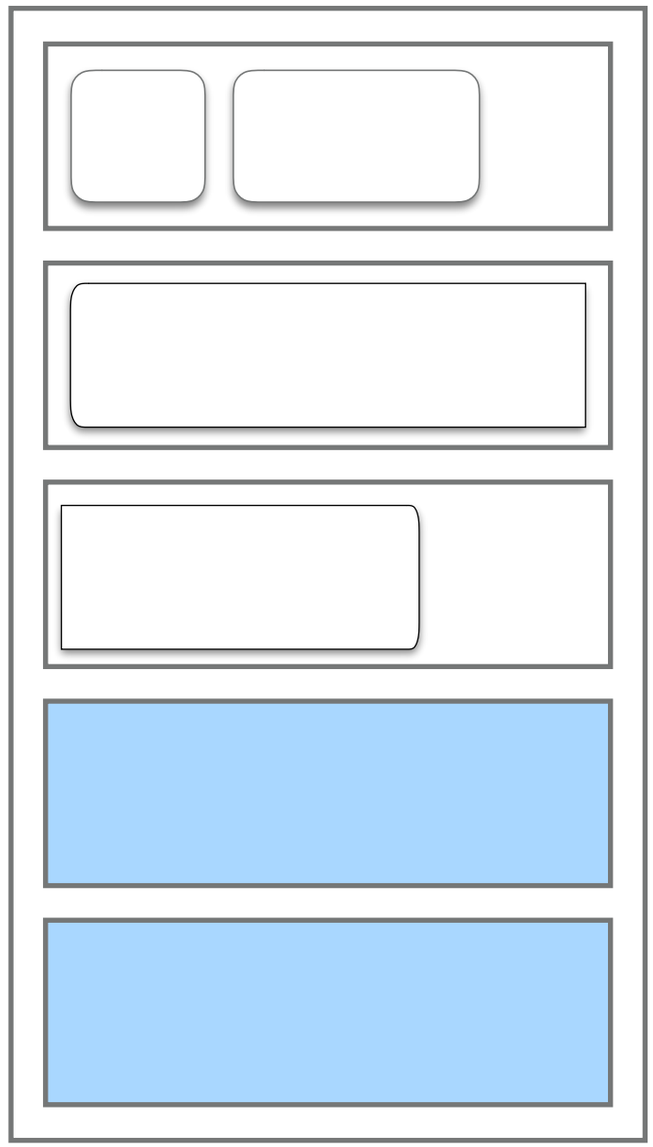
Stack

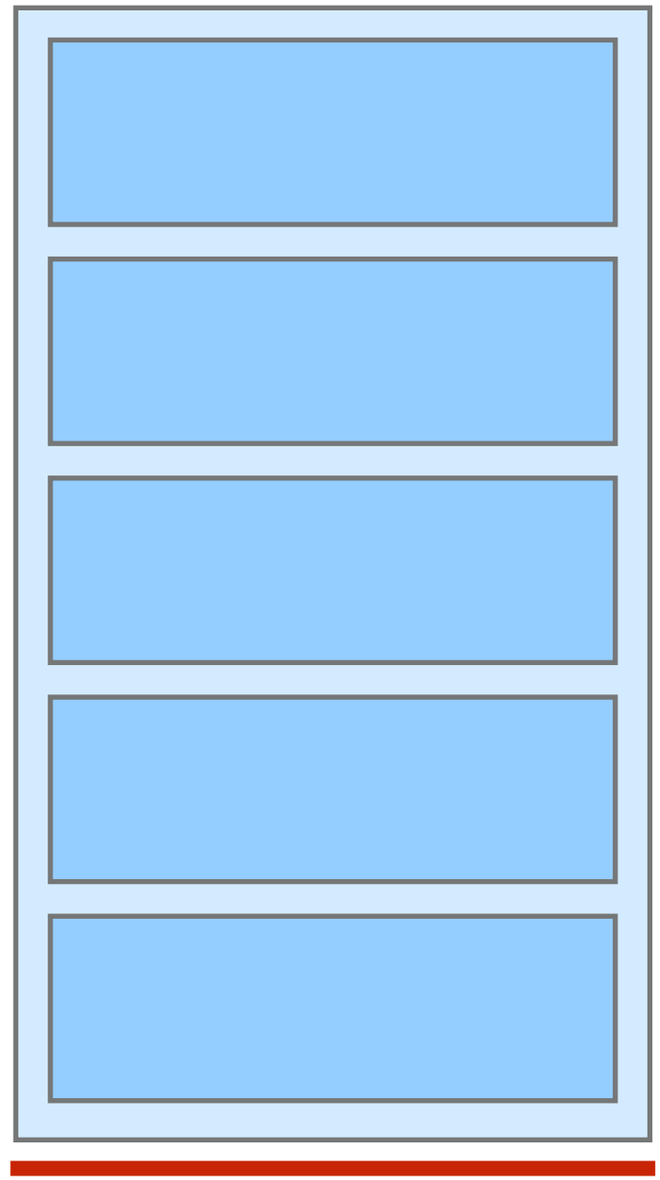




Modules

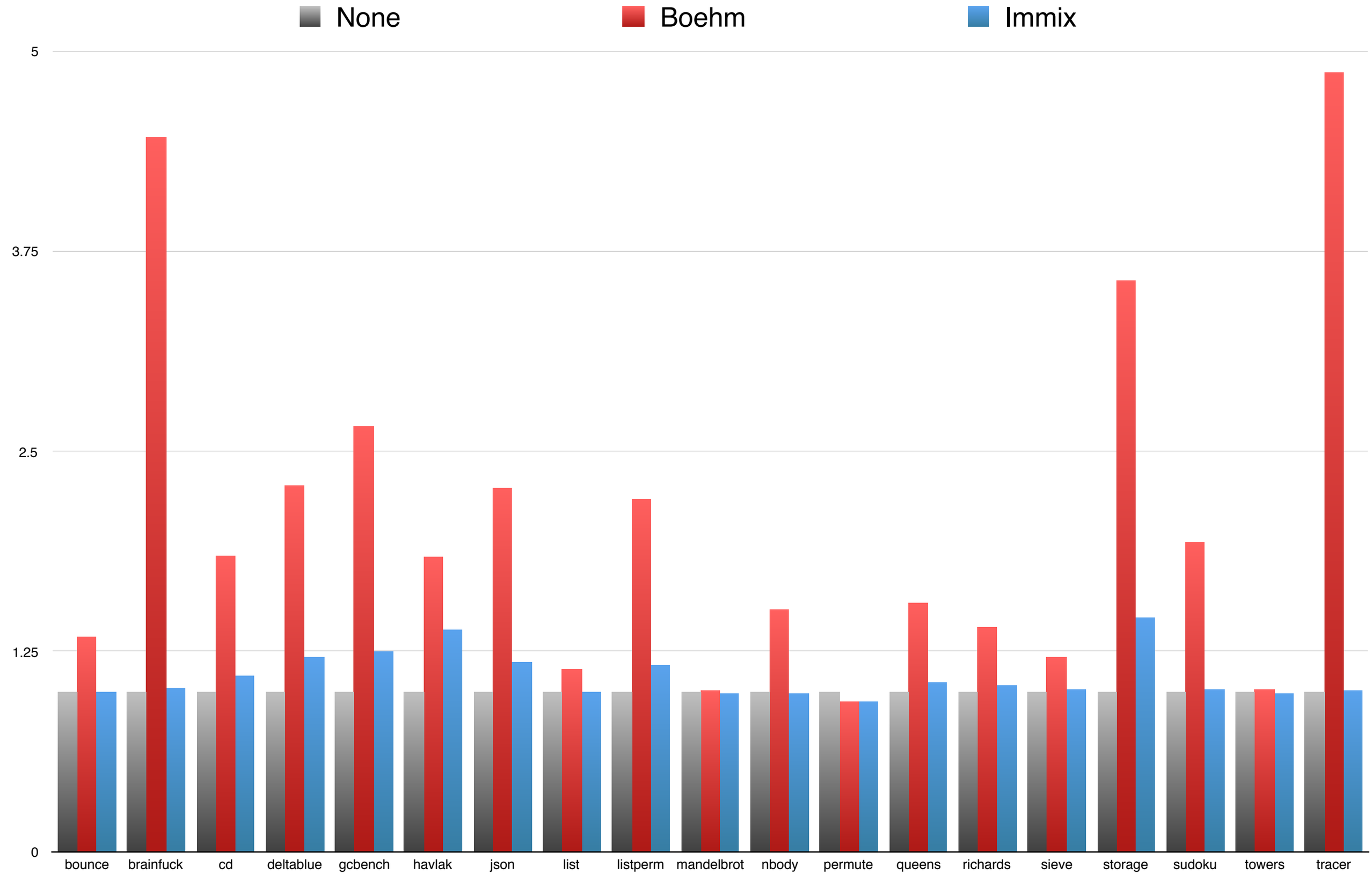
Sweeping Phase





Immix Performance

Immix Performance



Immix

- First prototype is coming in Scala Native 0.3
- Opt-in via `nativeGC := “immix”`
- We’re going to support Boehm until the immix implementation matures and becomes the default

Bonus features in 0.3

- Sbt testing framework integration
- Initial support for file i/o from java.nio.*
- Initial support for zip/jar from java.util.*
- Smaller binaries

Questions?