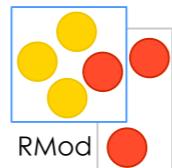


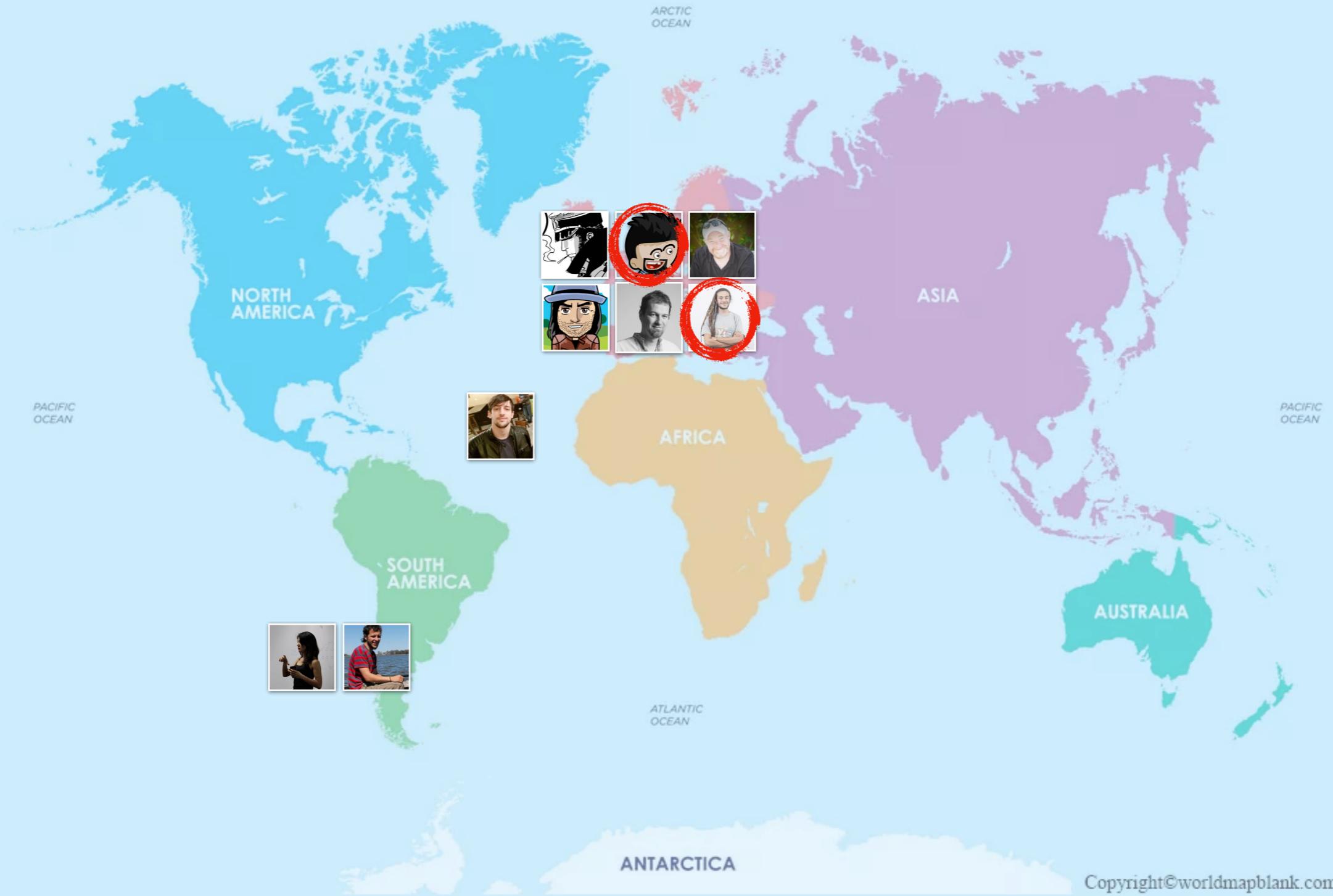
Improving Pharo Snapshots

P. Tesone - G. Polito - N. Palumbo - **ESUG'22**

@tesonep pablo.tesone@inria.fr
@guillep guillermo.polito@univ-lille.fr
@noTwitter nahuel.palumbo@inria.fr



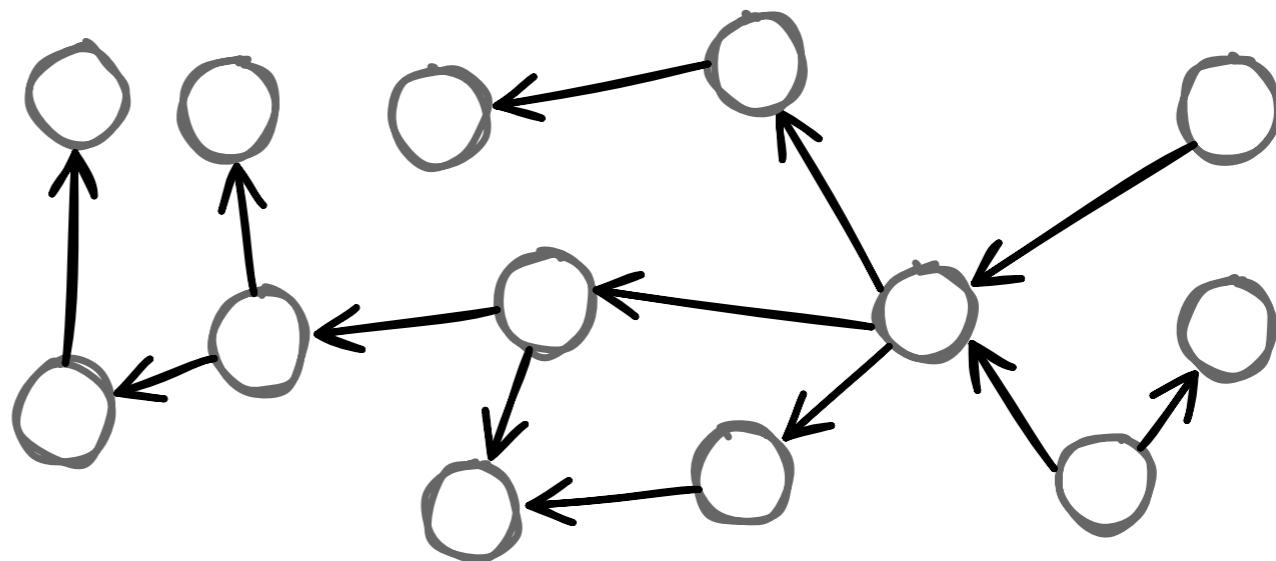
2022 VM+ Team



Copyright©worldmapblank.com

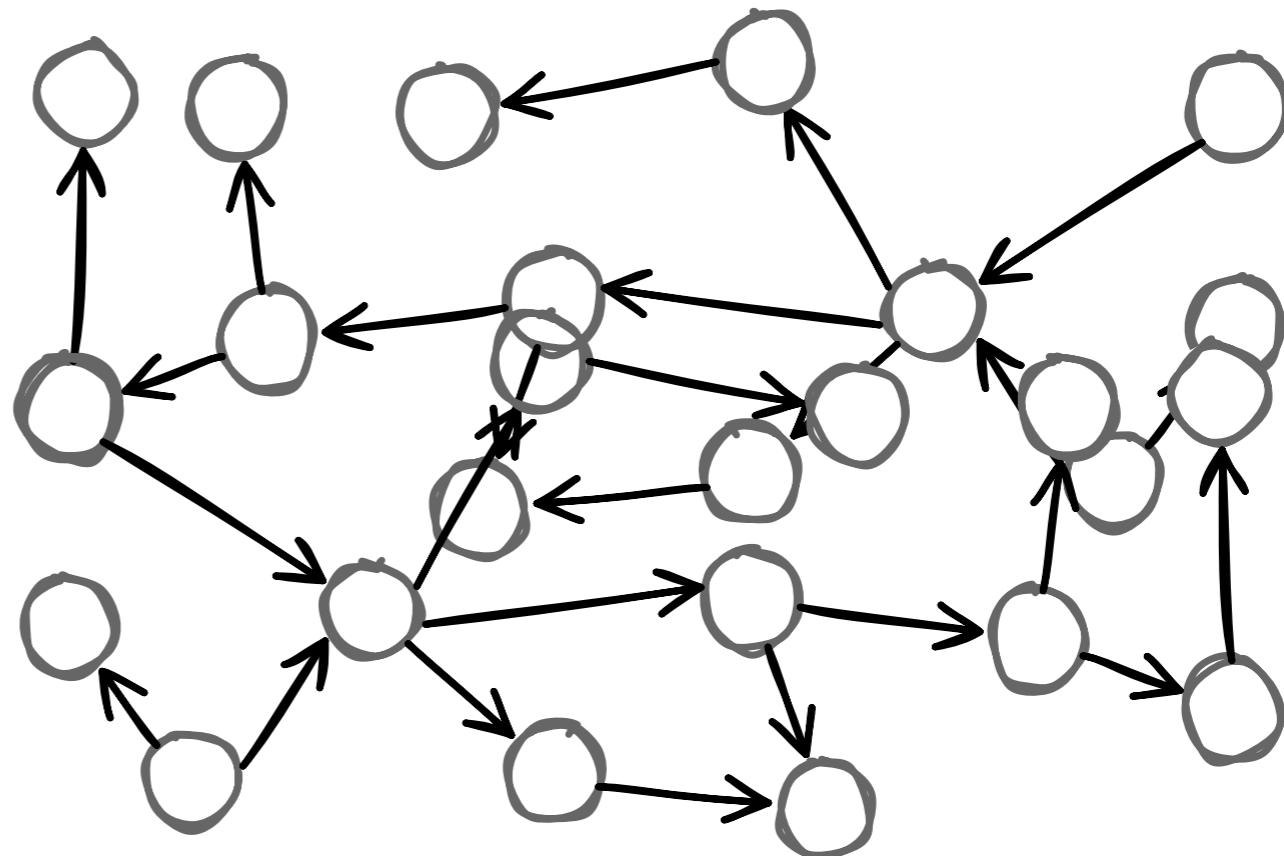
Everything is an Object

- Numbers
- Characters
- Strings
- Arrays



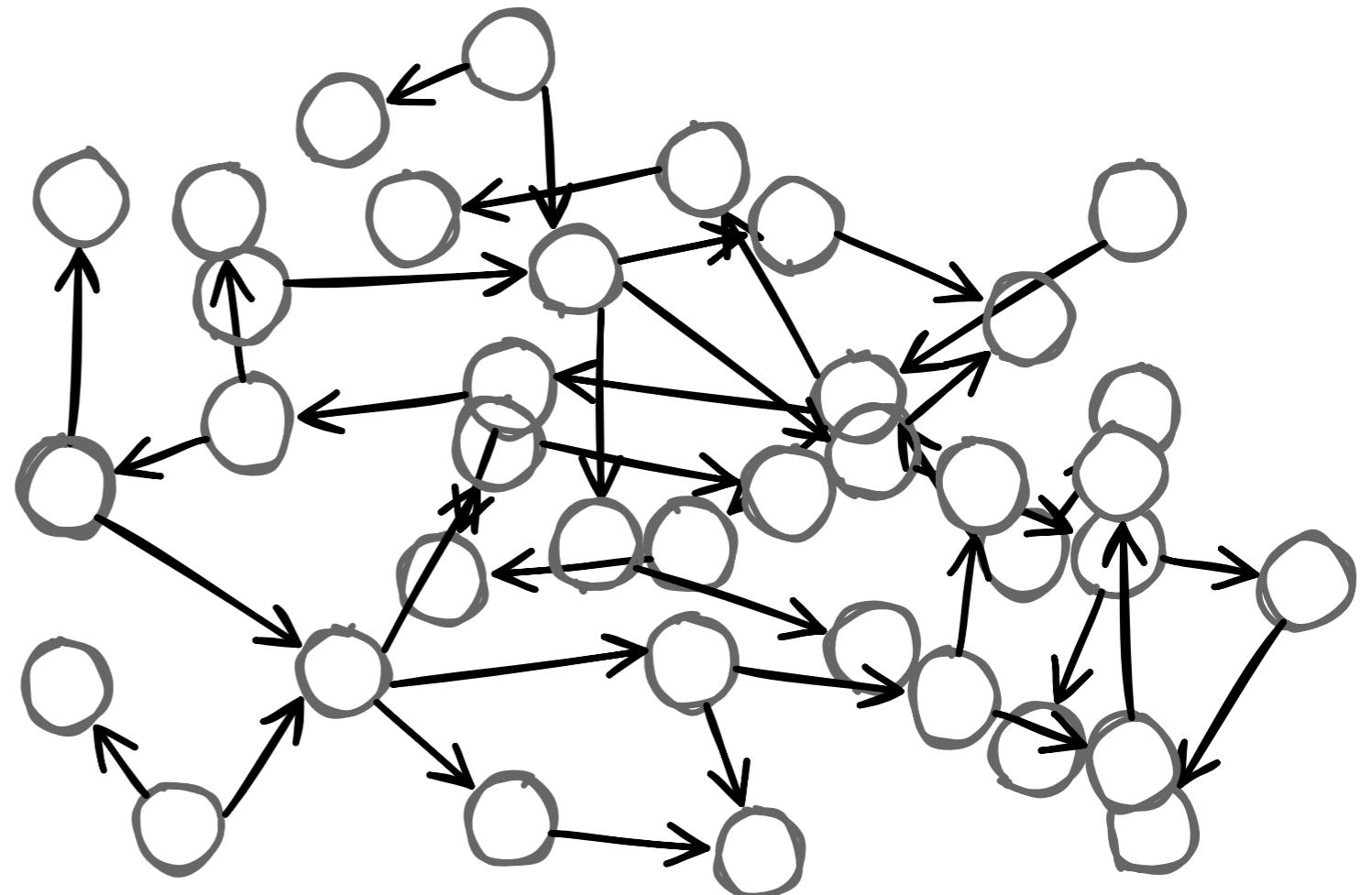
Everything is an Object

- Numbers
- Characters
- Strings
- Arrays
- Closures



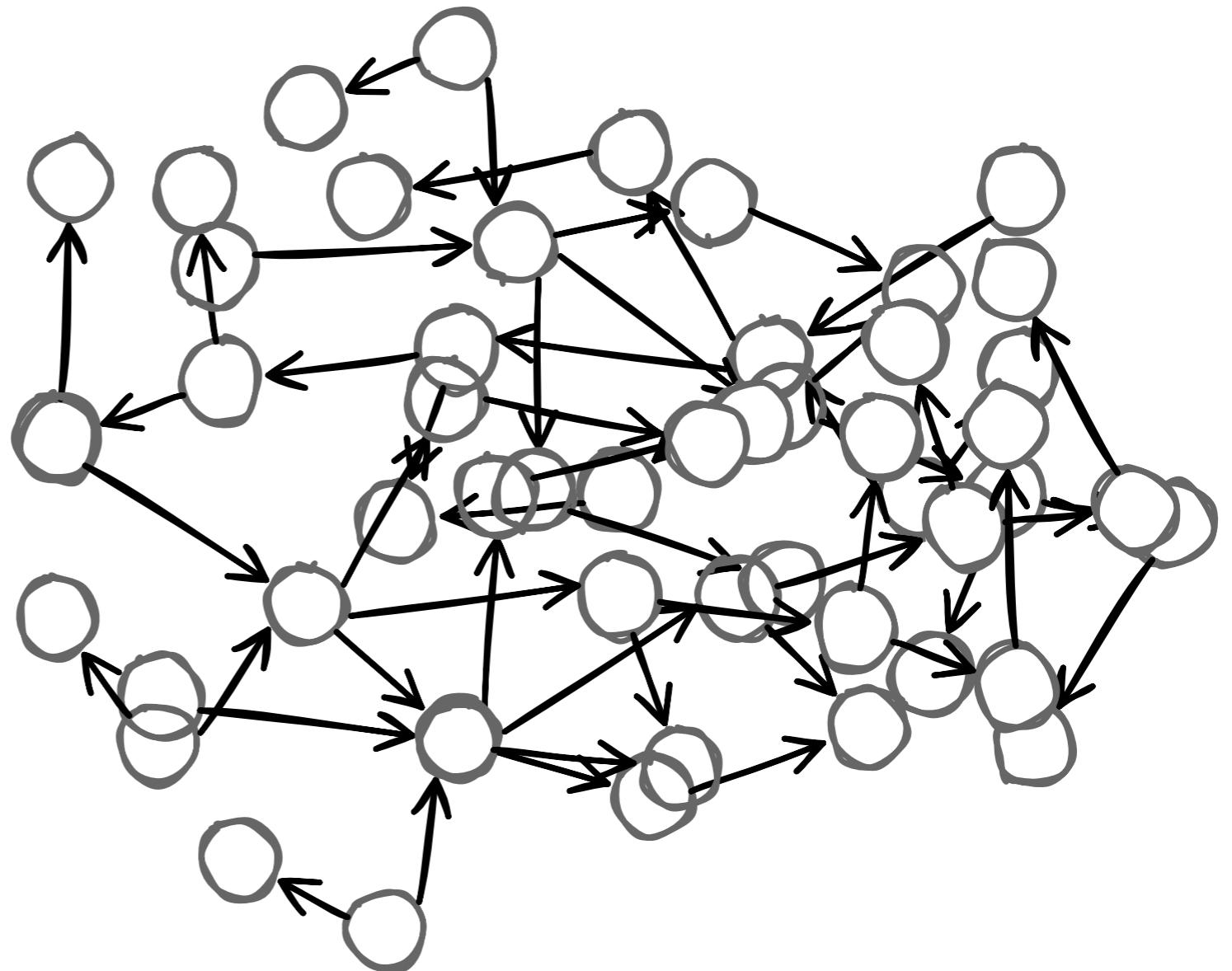
Everything is an Object

- Numbers
- Characters
- Strings
- Arrays
- Closures
- Classes



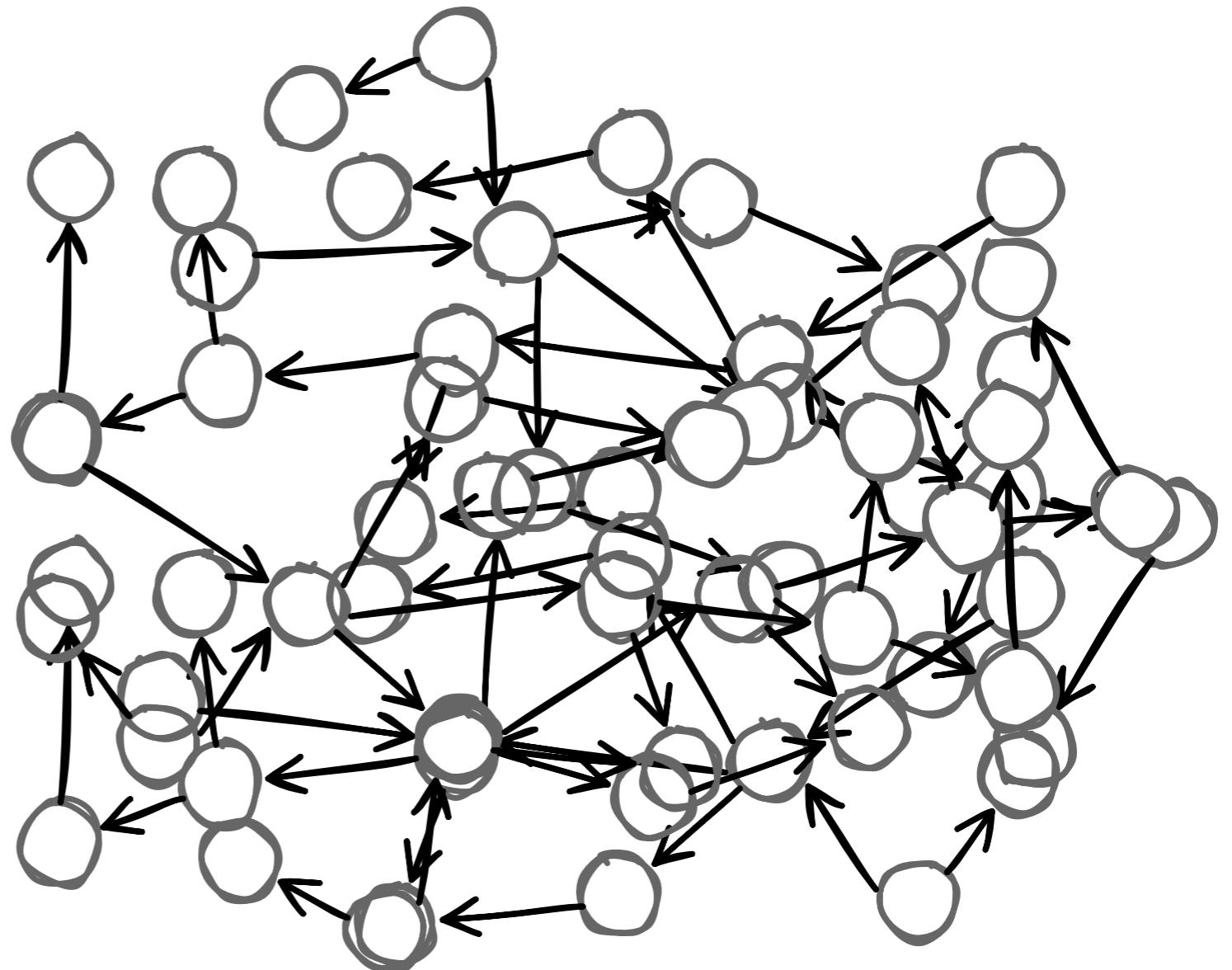
Everything is an Object

- Numbers
- Characters
- Strings
- Arrays
- Closures
- Classes
- Methods



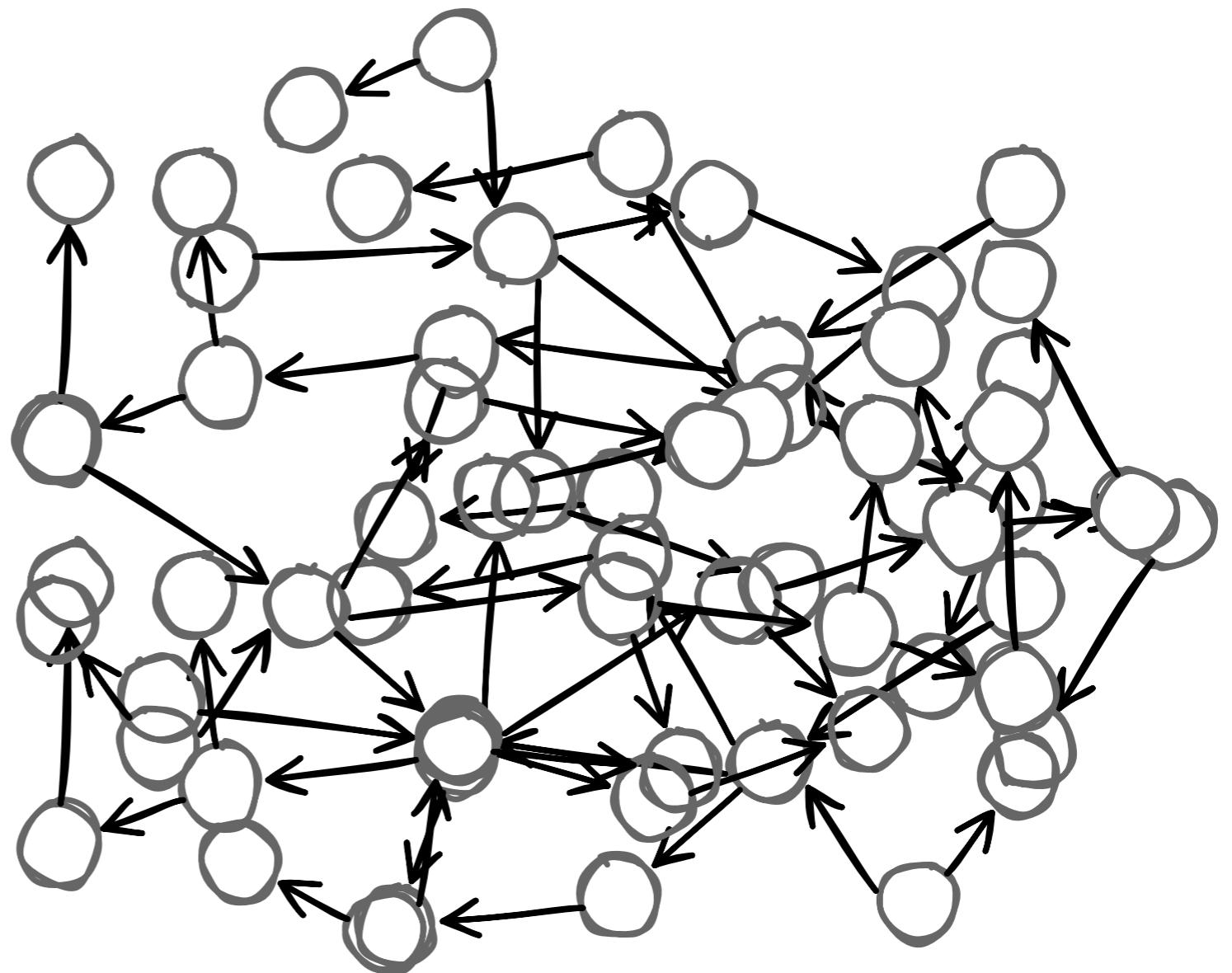
Everything is an Object

- Numbers
- Characters
- Strings
- Arrays
- Closures
- Classes
- Methods
- ...



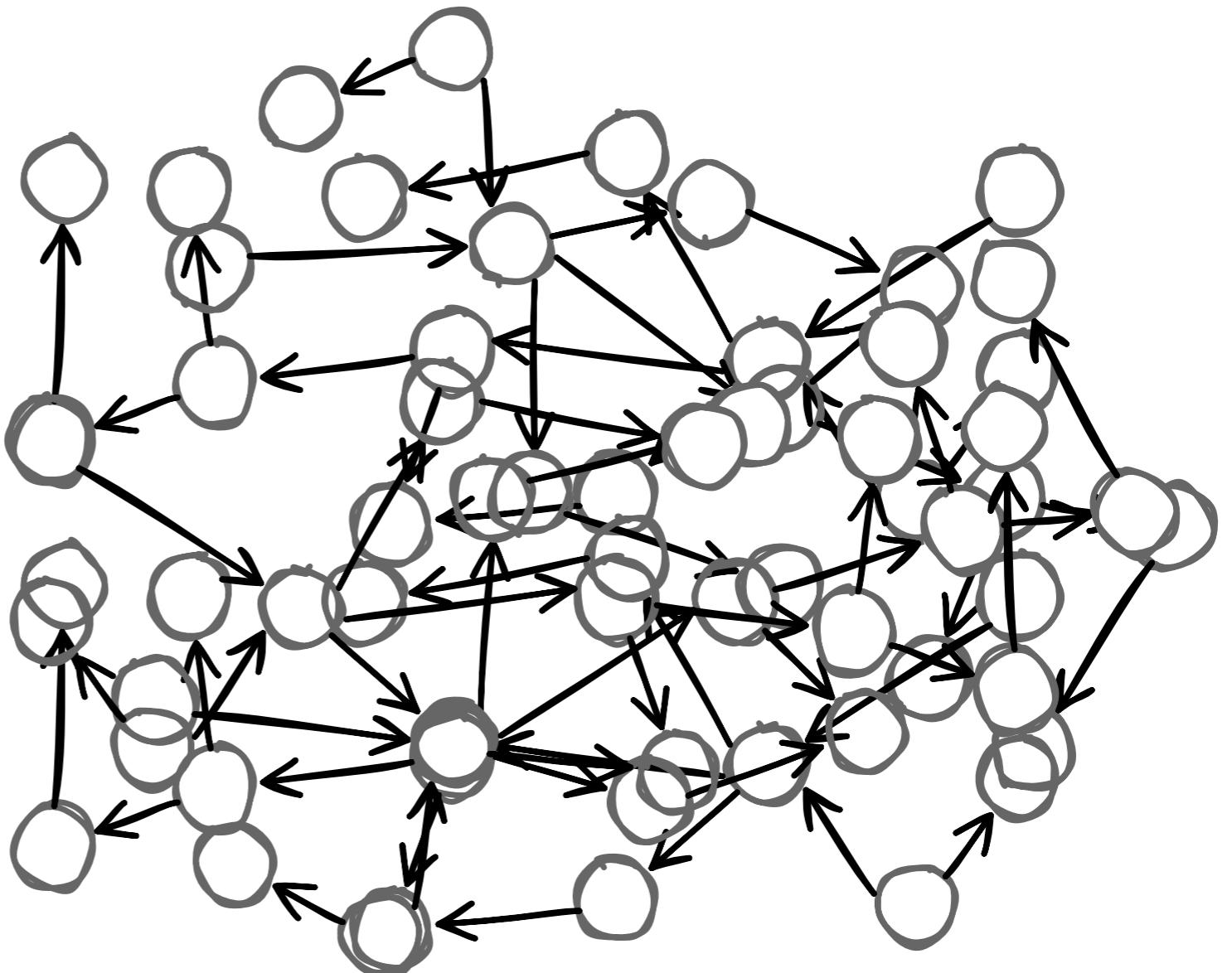
Lots of Objects

- Numbers
- Characters
- Strings
- Arrays
- Closures
- Classes
- Methods
- ...



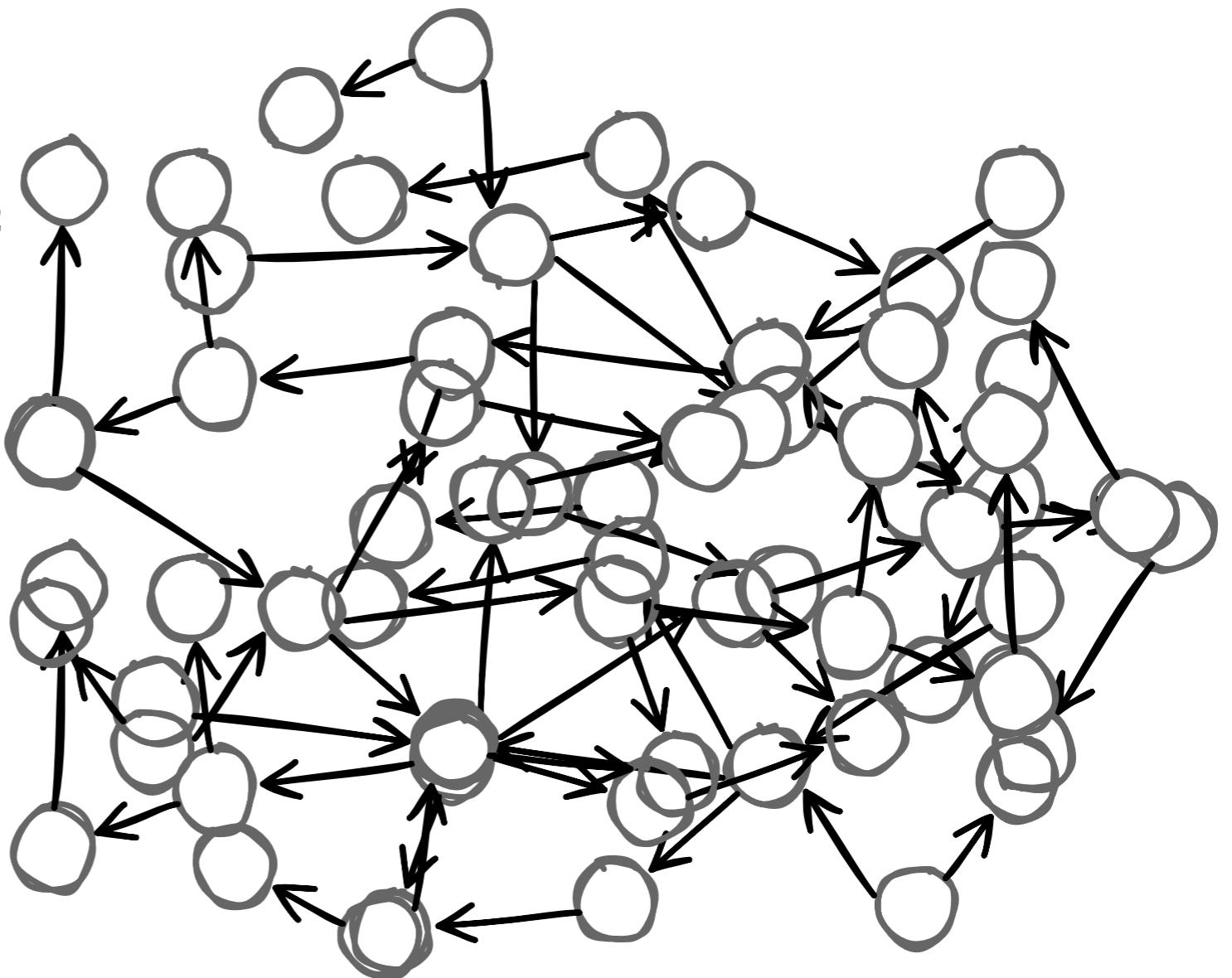
~~Lots of Objects~~ Lots of Stress

- Numbers
- Characters
- Strings
- Arrays
- Closures
- Classes
- Methods
- ...



~~Lots of Objects~~ Lots of Stress

- GC **Stress**
- Autocompletion **Stress**
- Search **Stress**
- Spotter **Stress**
- Startup **Stress**



Large Image Support



- <https://github.com/pharo-project/largelimages>
- MIT Licenced

A screenshot of a web browser window displaying the GitHub repository page for "pharo-project/largelimages".

The browser's address bar shows the URL <https://github.com/pharo-project/largelimages>. The page content includes:

- README.md**: A file containing the project's description and dependencies.
- Large Image Support**: The main heading of the README file.
- Description**: A paragraph explaining the project's purpose: "This project includes a baseline to load a series of enhancements to Pharo. These enhancements provide a better user experience when coping with large images. Large images are images with a lot of objects, this objects are not only objects representing our data but also it applies to images with a lot of code."
- Dependencies**: A list of two projects:
 - [Complishon](#) a new completion engine for Pharo that provides better contextual answers and it is implemented to minimize the queries to the global system.
 - [Spotter](#) an iteration on the processor model of GTSpotter adding new processors that uses a set of composable iterator to perform the queries incrementally.
- Contributors**: A section listing three contributors with their GitHub profiles:
 - [tesonep](#) Pablo Tesone
 - [estebanlm](#) Esteban Lorenzano
 - [guillep](#) Guille Polito

Large Image Support: Highlights

- Generator based searches

- Spotter

- Code Completion

```
generator
```

```
  ^ generator ifNil: [
    generator := Generator on: [ :g |
      self entriesDo: [ :entry |
        (self acceptsEntry: entry)
          ifTrue: [ g yield: entry ] ] ] ]
```

- GC Fine Tuning API



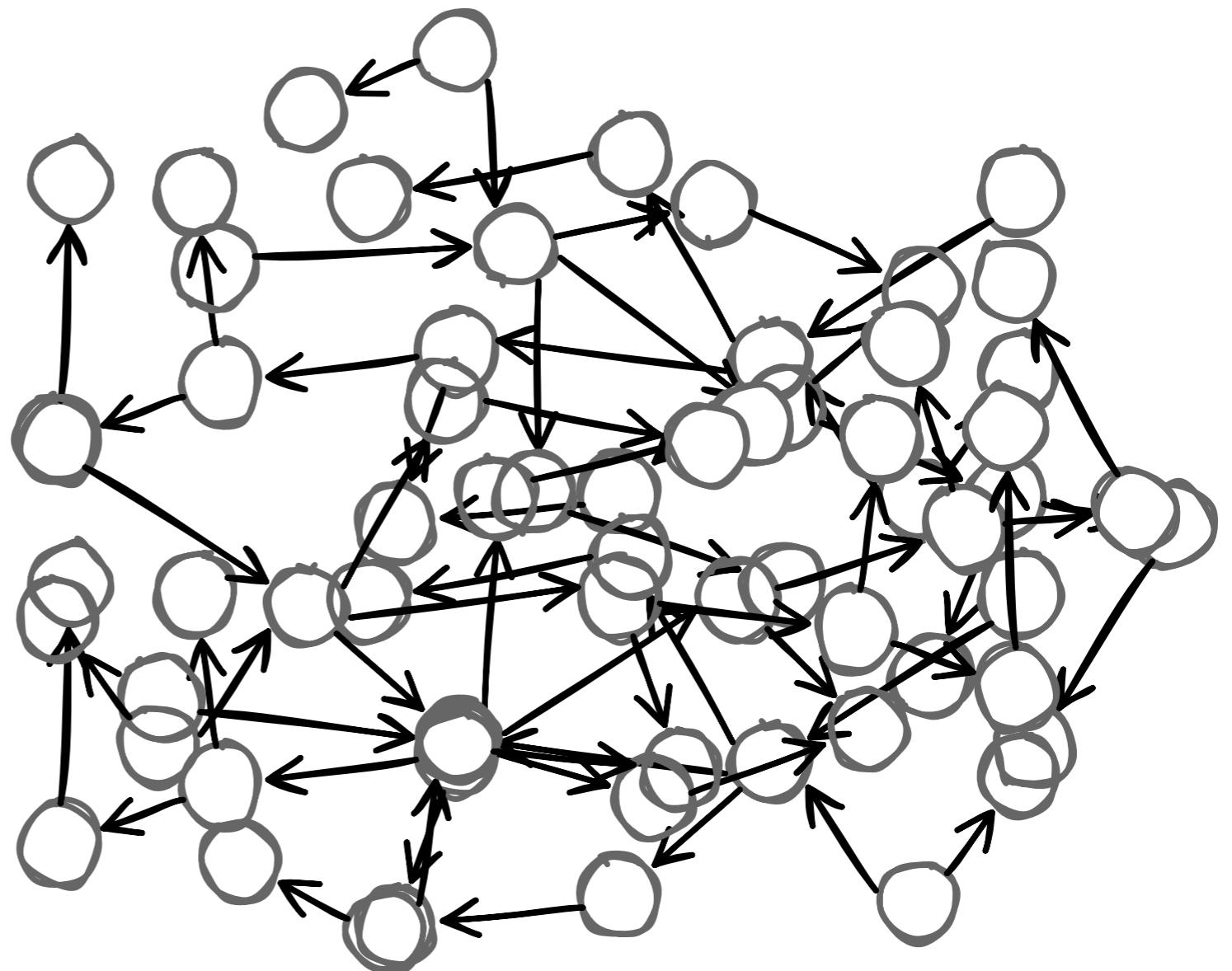
GC Fine Tuning

- Configure
 - Eden Size
 - Full GC Ratio
 - Growth Headroom
 - Shrink Threshold

```
GCConfiguration readFromVM
    fullGCRatio: 1.0;
    activeDuring: [ "something" ].
```

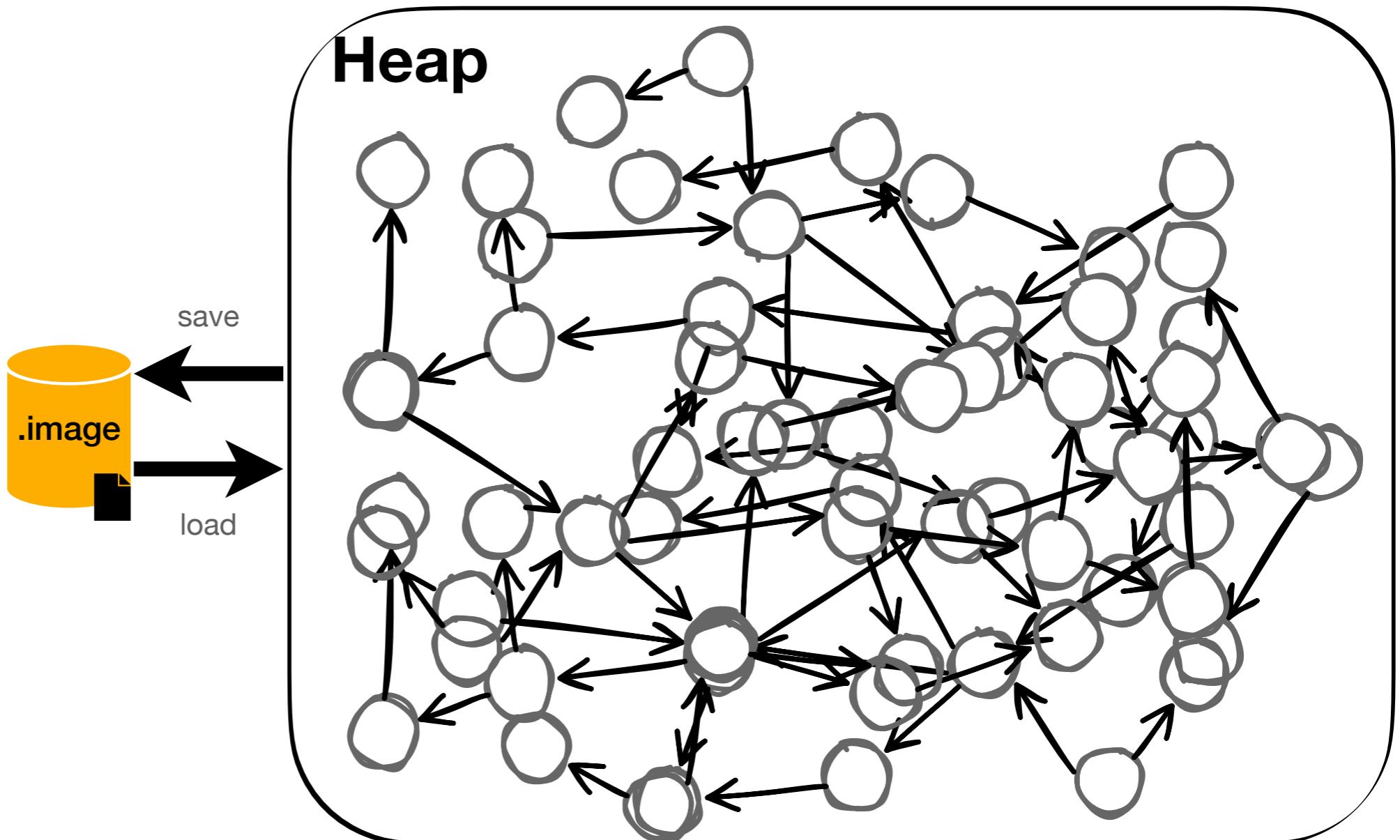
Lots of Objects

- Numbers
- Characters
- Strings
- Arrays
- Closures
- Classes
- Methods
- ...



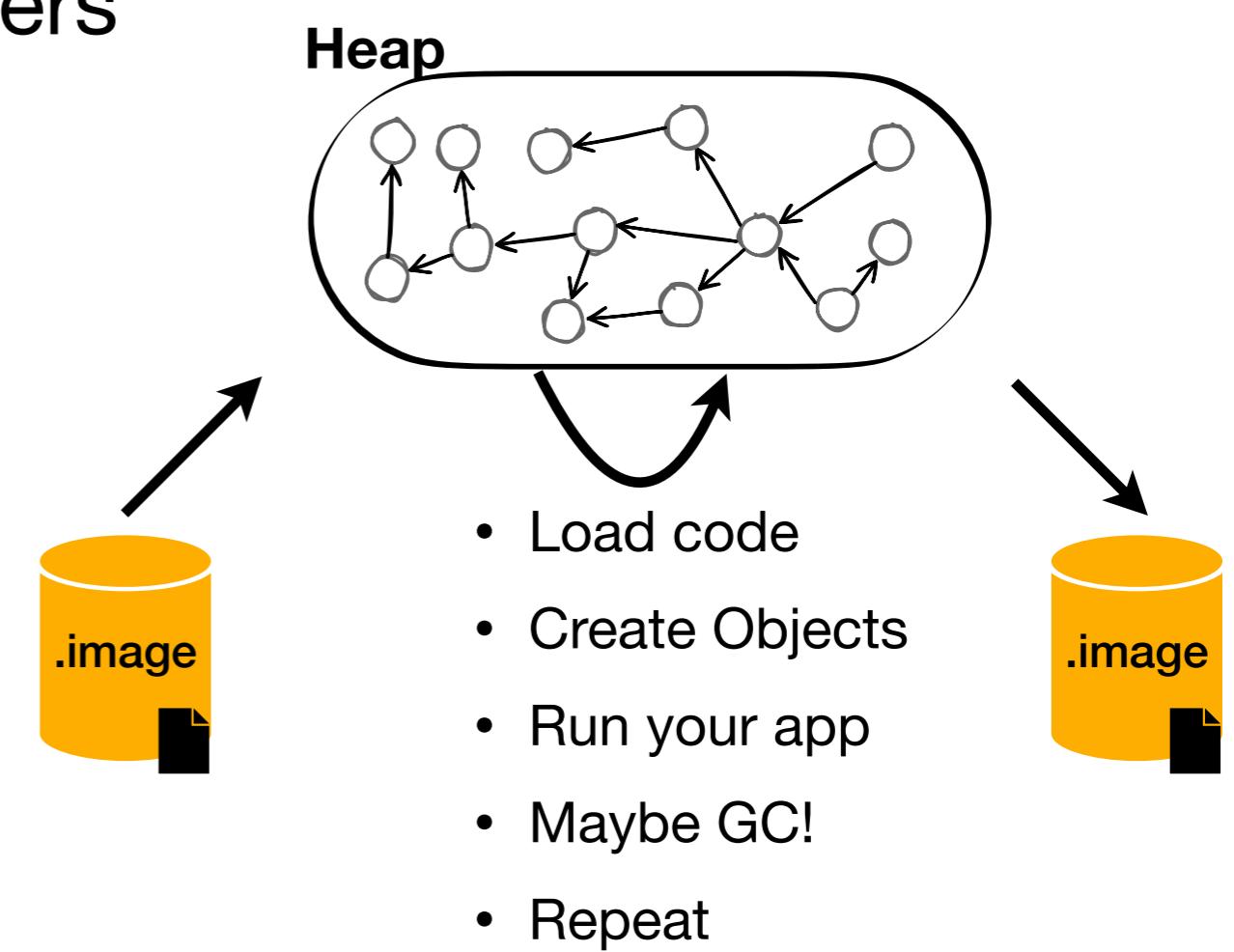
Images = Heap Snapshots

- Numbers
- Characters
- Strings
- Arrays
- Closures
- Classes
- Methods
- ...



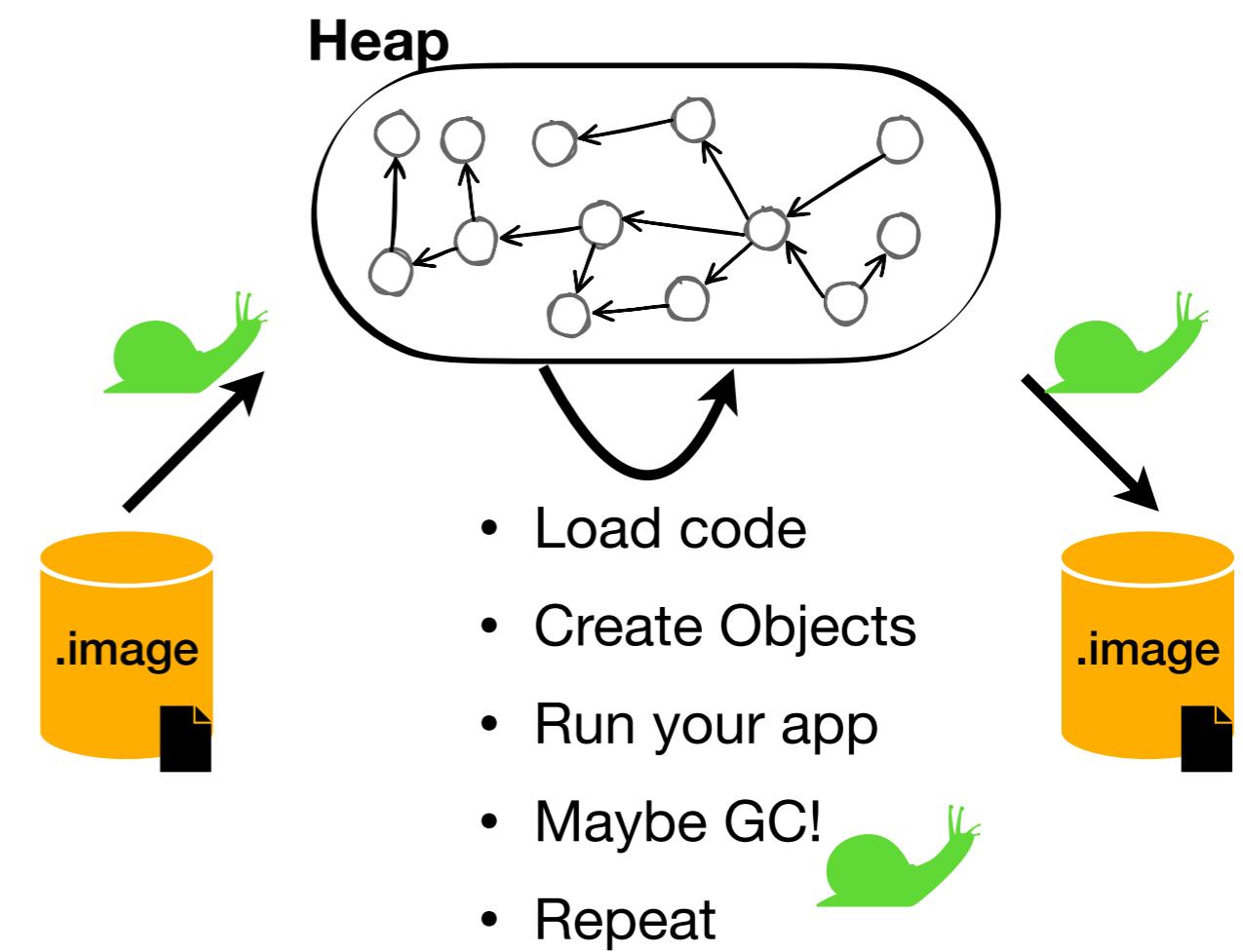
Snapshot Current Design Points

- Bootstrap once, then mutate
- Portable
- Object References are pointers



But it could be better...

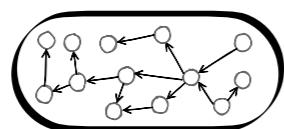
- VM startup is bound by disk!
- Large heaps take long to load/save
- 3-4GB heaps = seconds to GC
 - pauses
 - long pauses



Snapshots vs Runtime Memory Mismatch

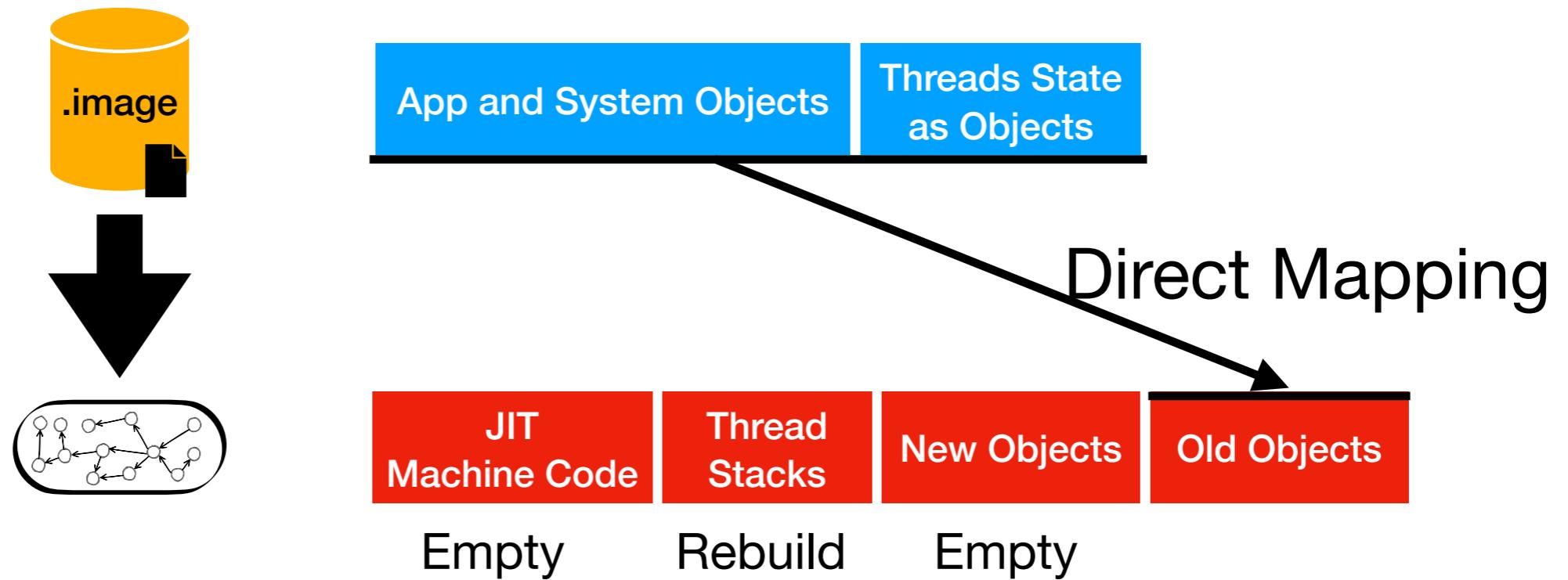


App and System Objects
Threads State
as Objects



JIT
Machine Code
Thread
Stacks
New Objects
Old Objects

Current Loading Snapshot to Memory



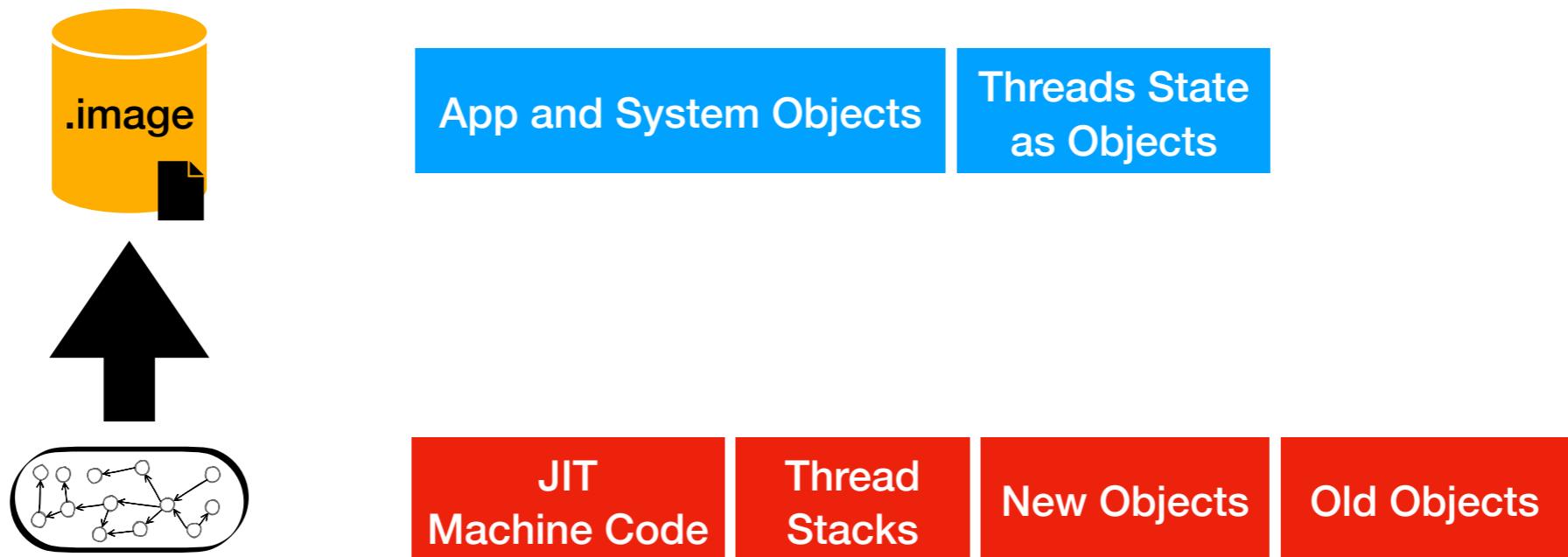
Start with a cold VM,
startup is slow

Reference Swizzling

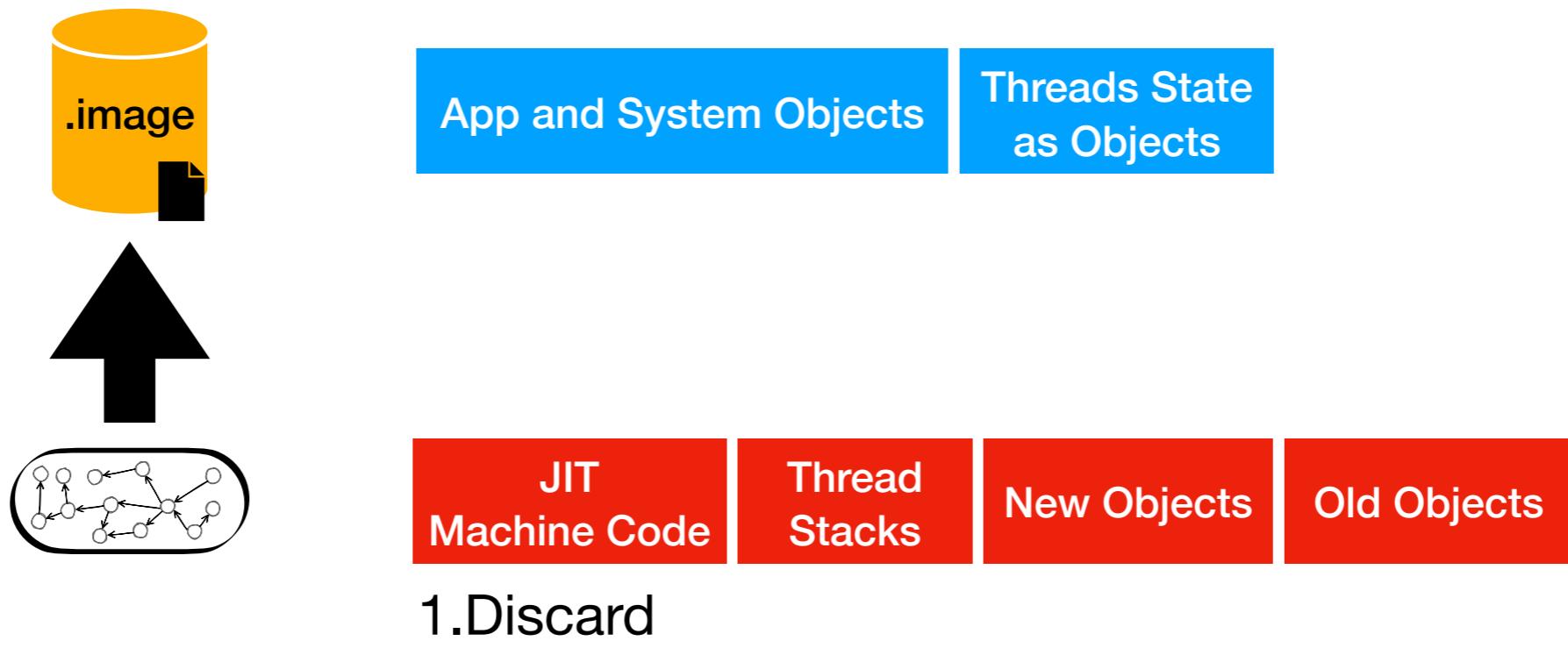


- Traverse the heap to remap old references by *delta*
- Slow for large heaps (2/4GB)

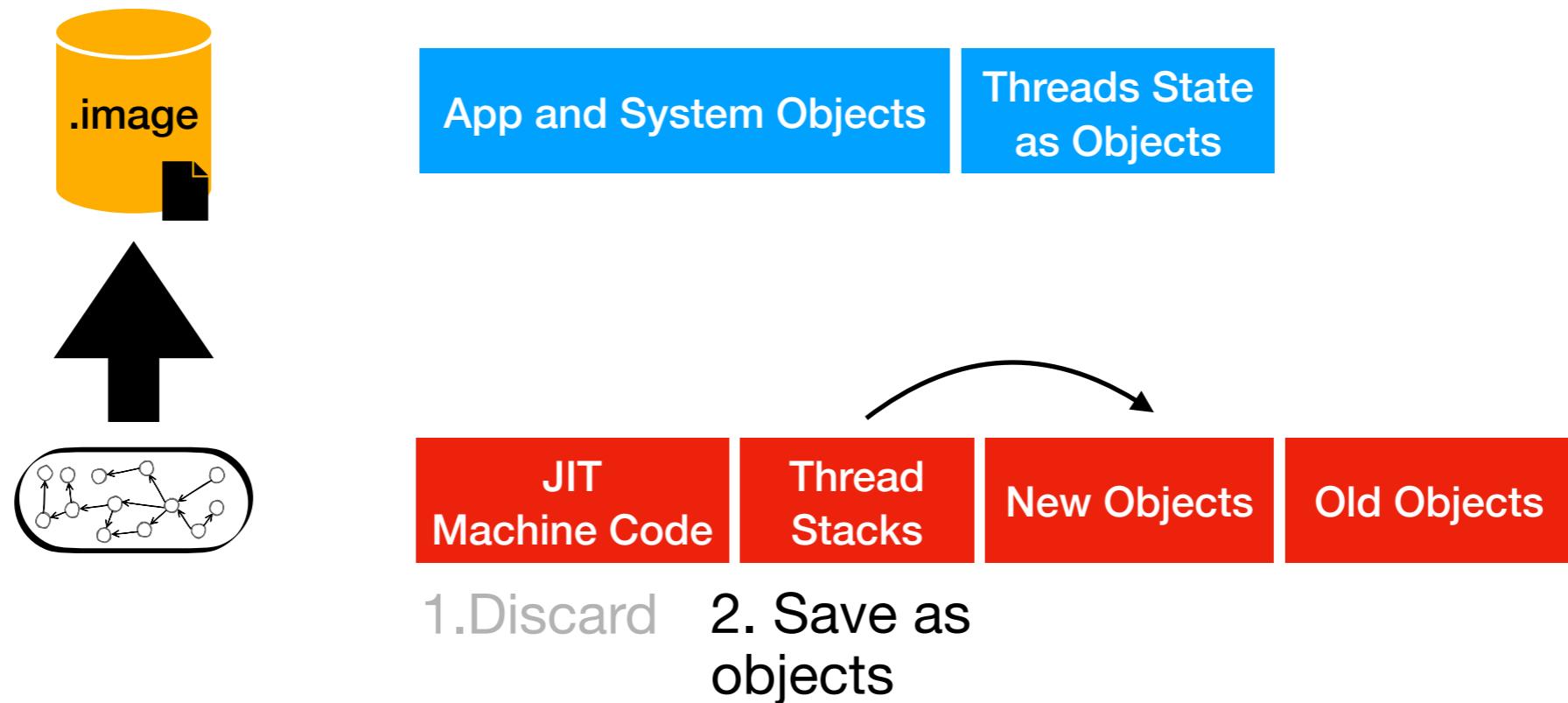
Current Snapshot to Disk



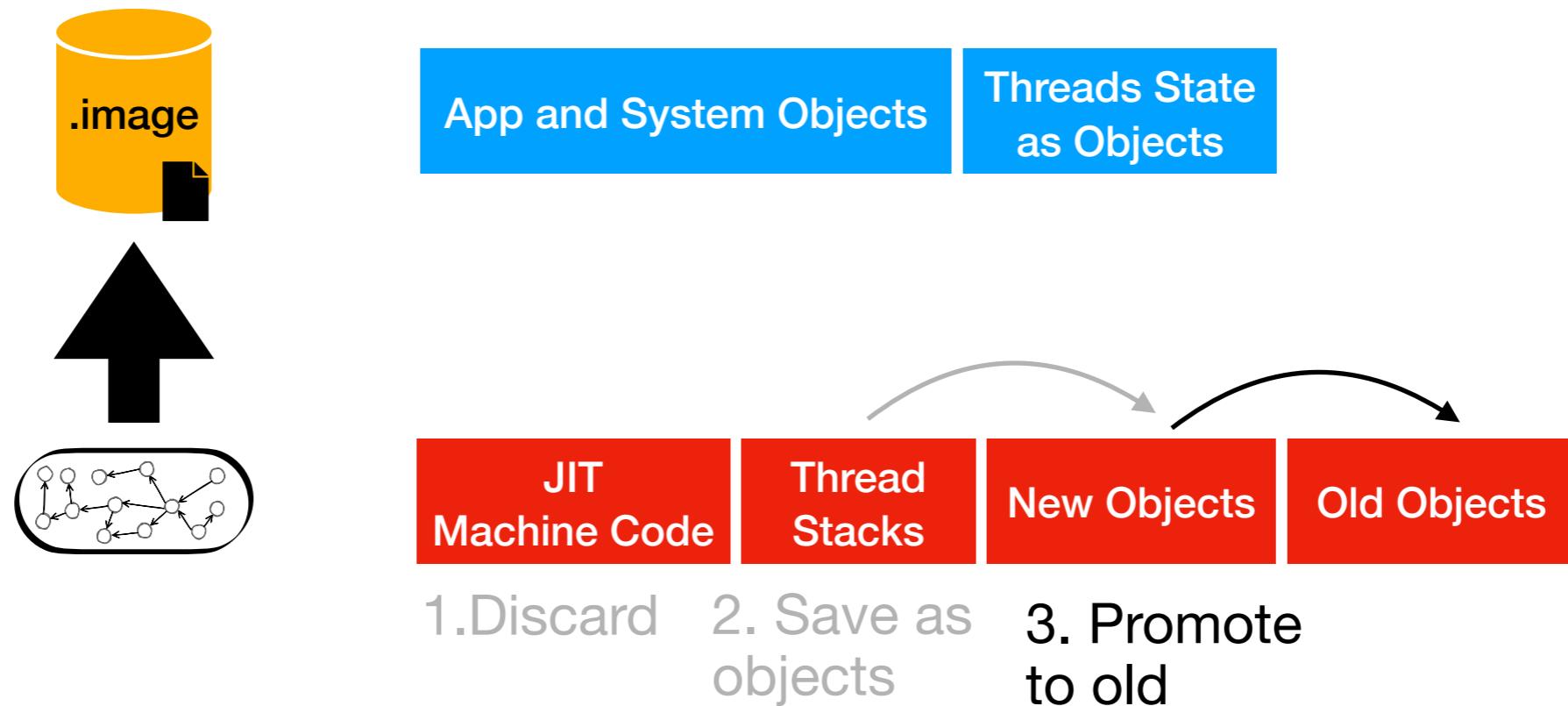
Current Snapshot to Disk



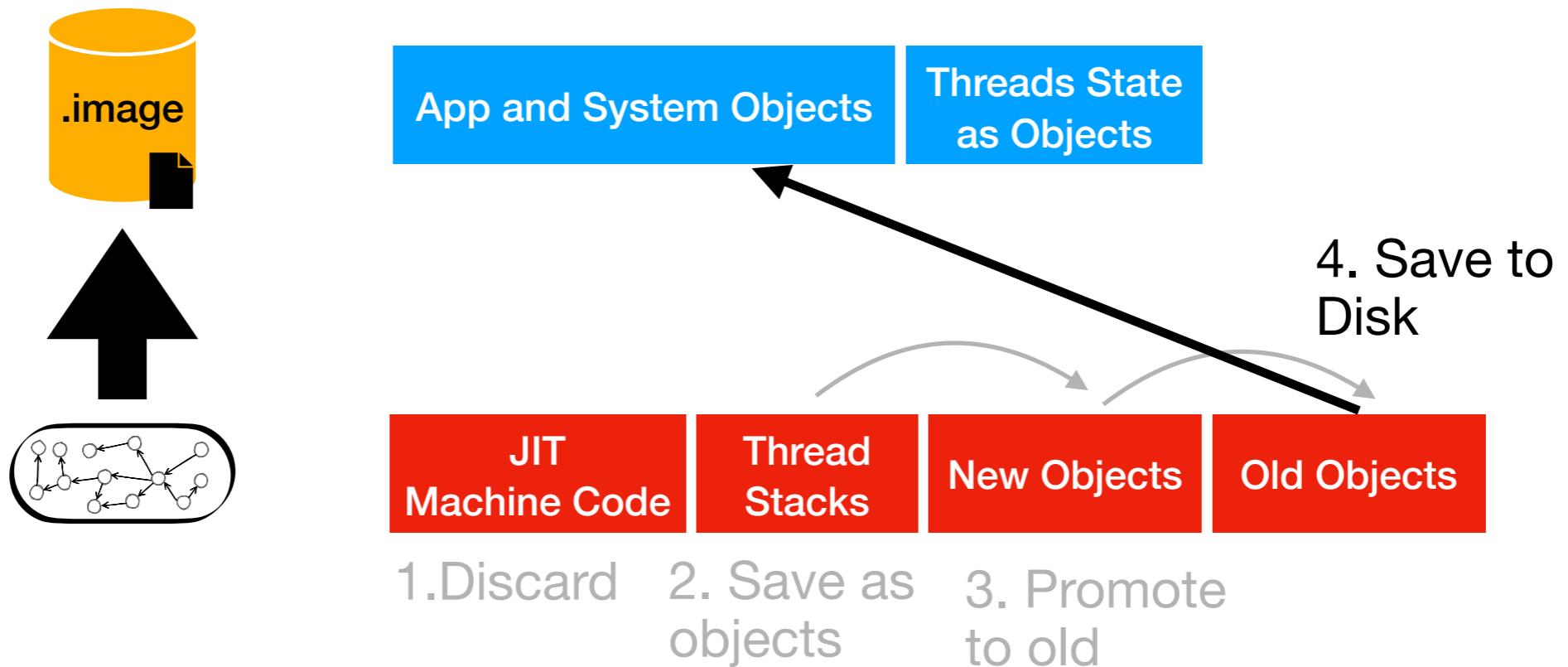
Current Snapshot to Disk



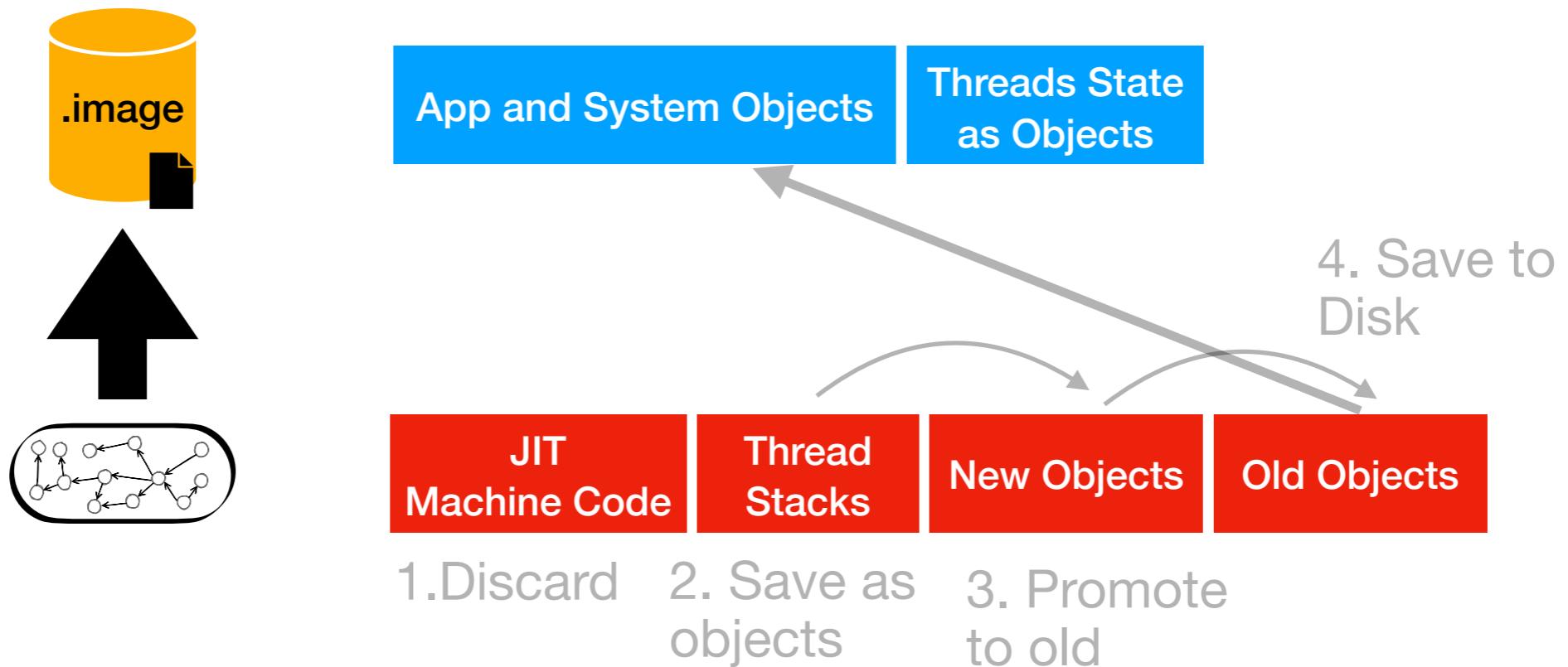
Current Snapshot to Disk



Current Snapshot to Disk



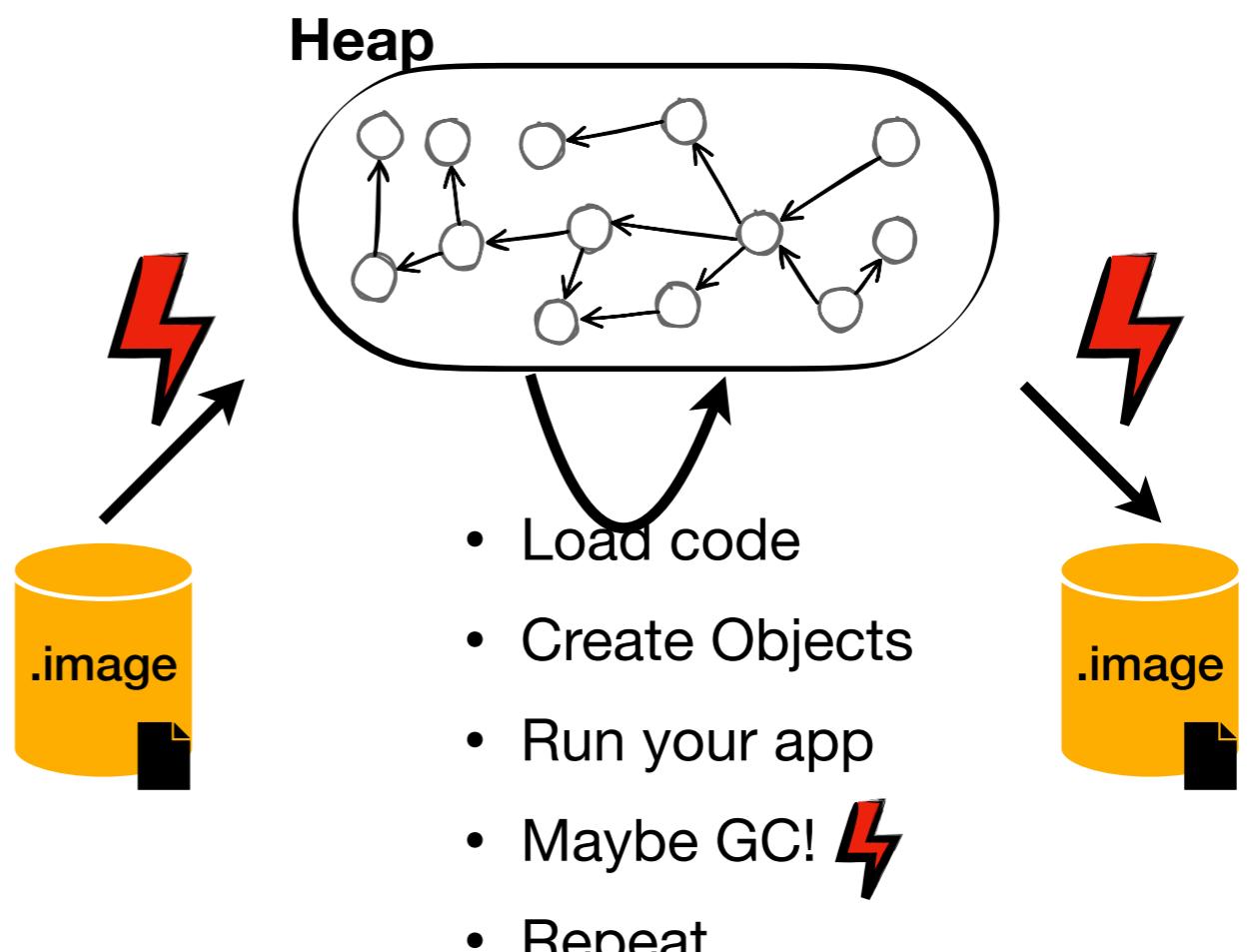
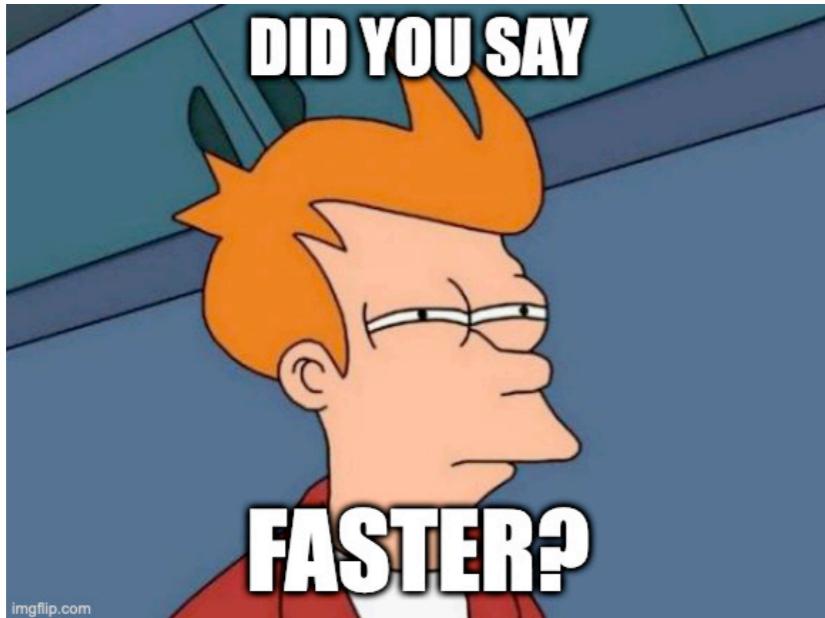
Current Snapshot to Disk



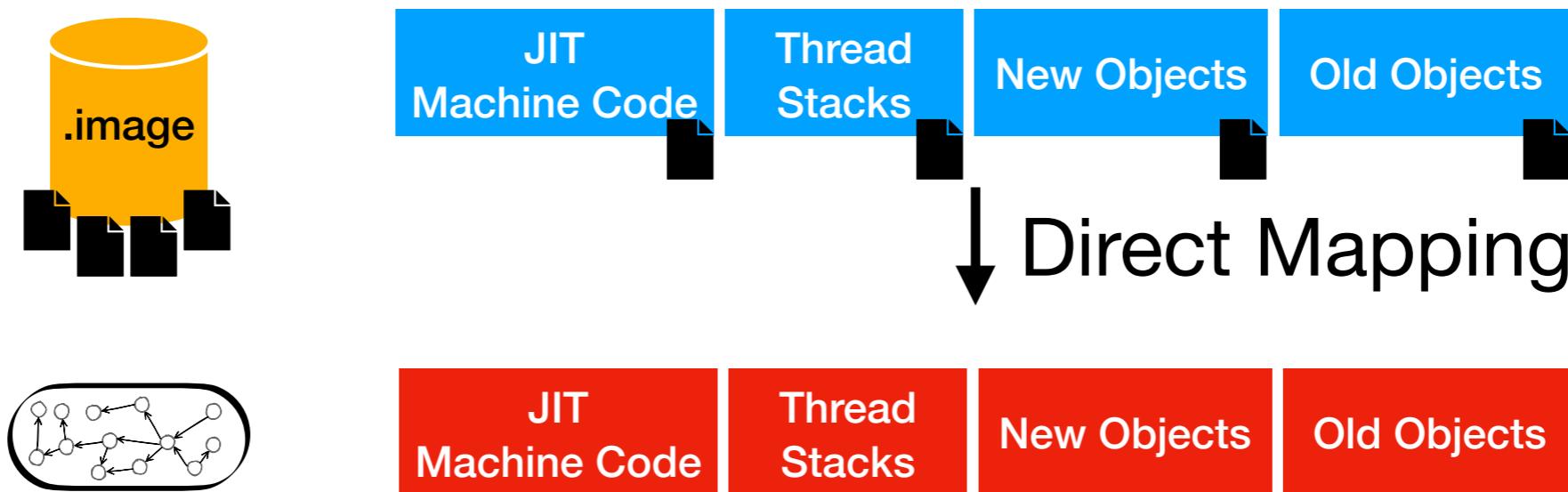
**Discards all optimisations:
slow shutdown => slow startup**

Goals

- *Faster* loading
- *Faster* snapshot
- *Faster* Multi-GB Heaps

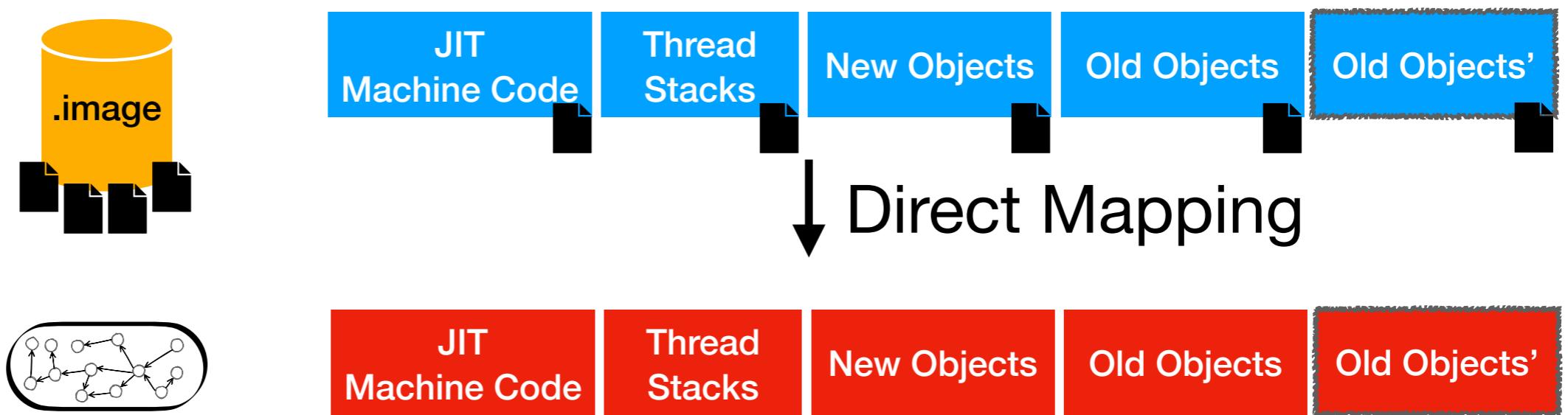


Towards a Multi-file Snapshot Format



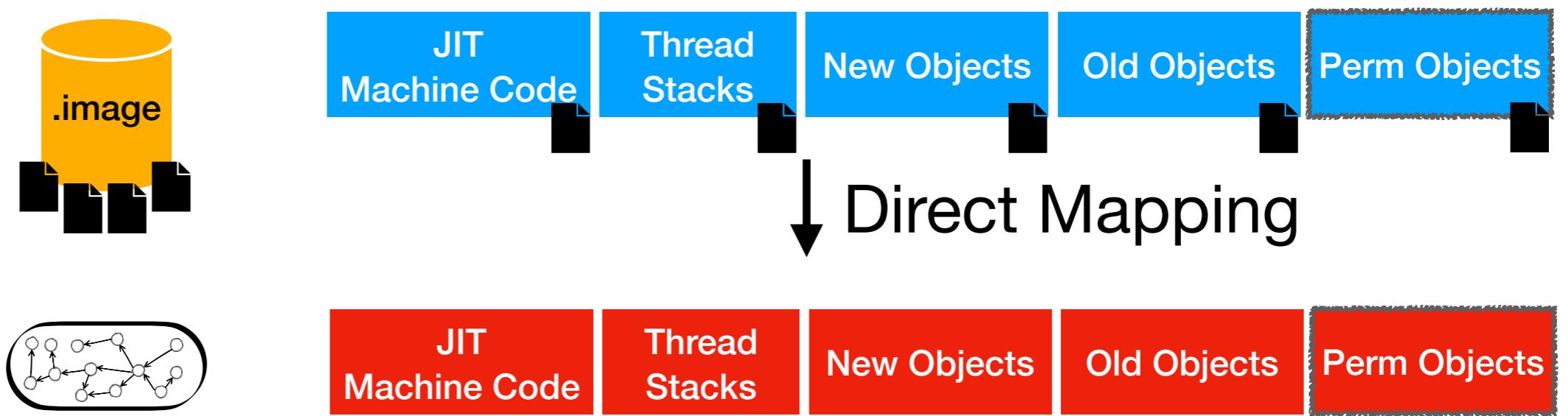
- System memory mapping
- Minimize Swizzling
- Lazy loading of memory segments

Multiple Memory Segments



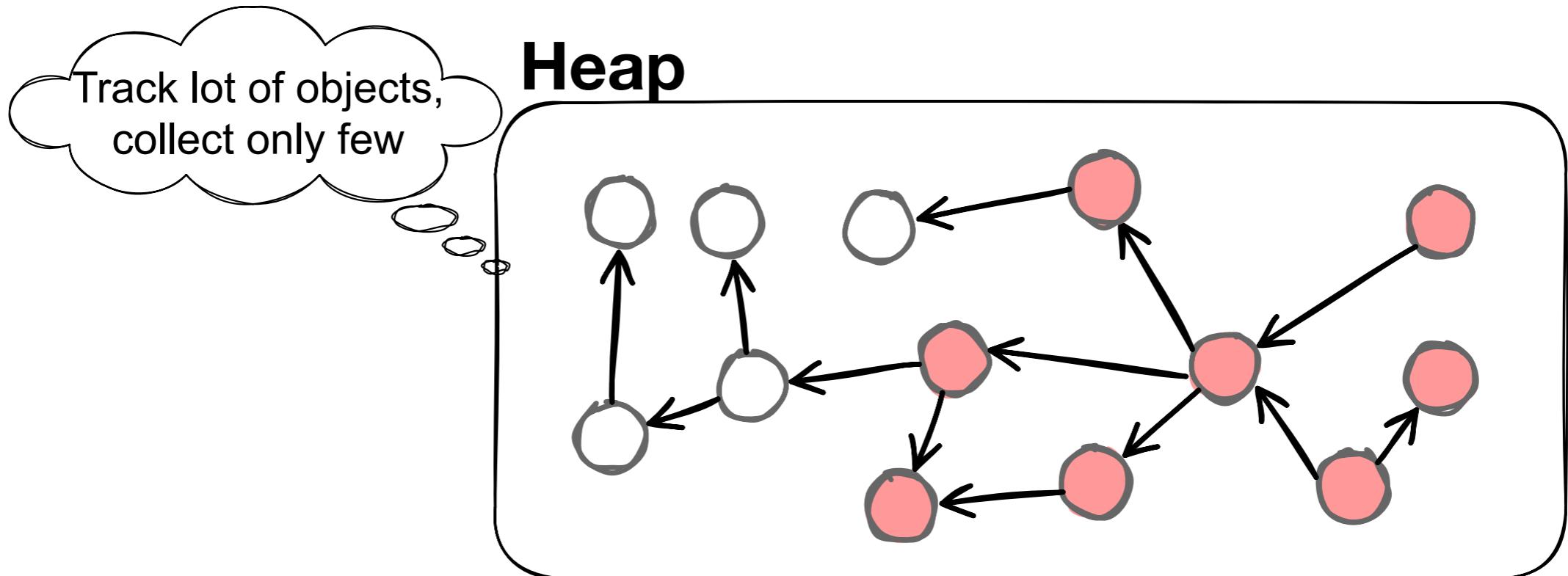
- Independently and *lazy* loadable ⚡
- Independently storable ⚡

New Memory Segments



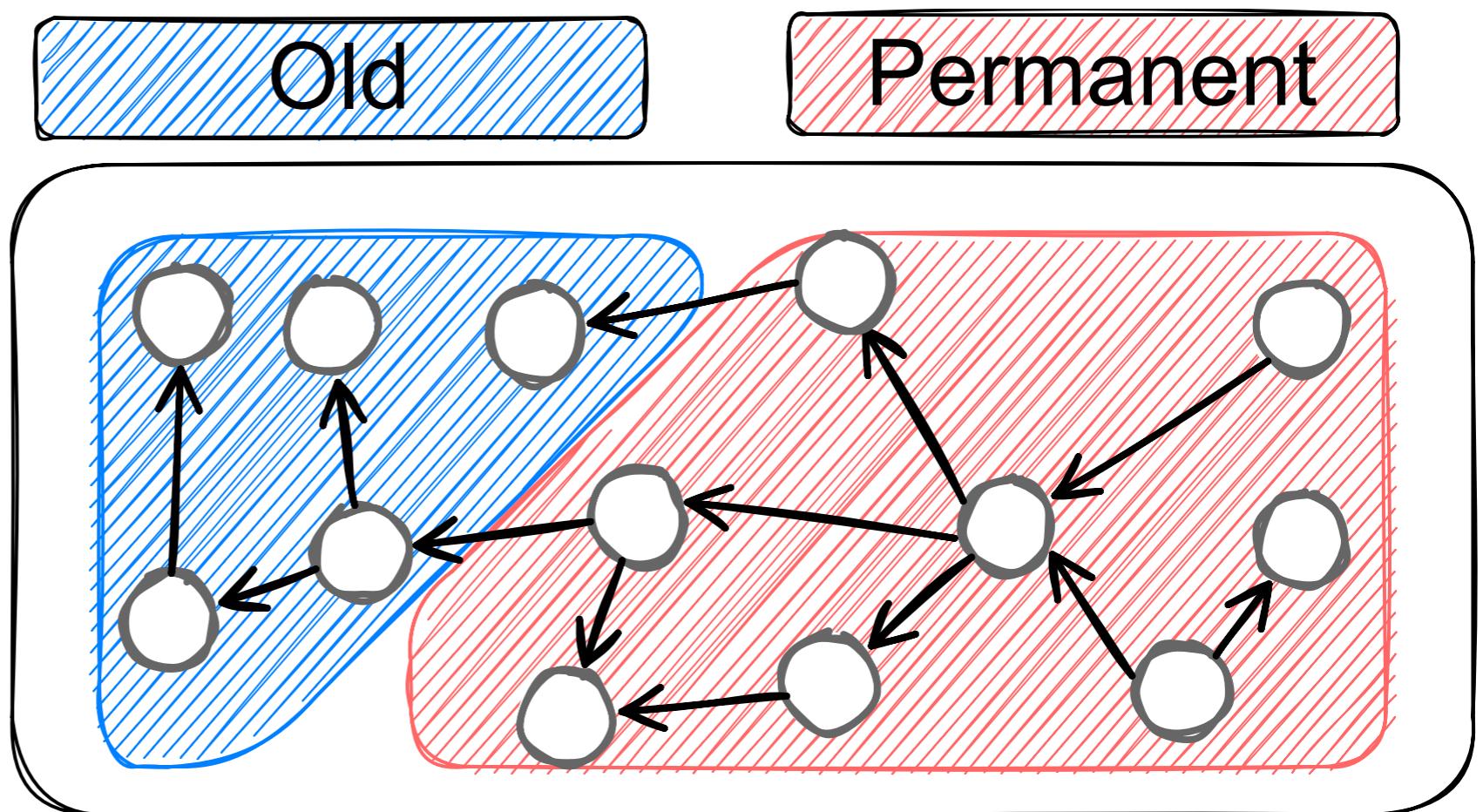
- Reduced garbage collection pressure ⚡
- Great for opaque objects, and rarely changing objects (code, literals...)

Semi-permanent Heap Segments



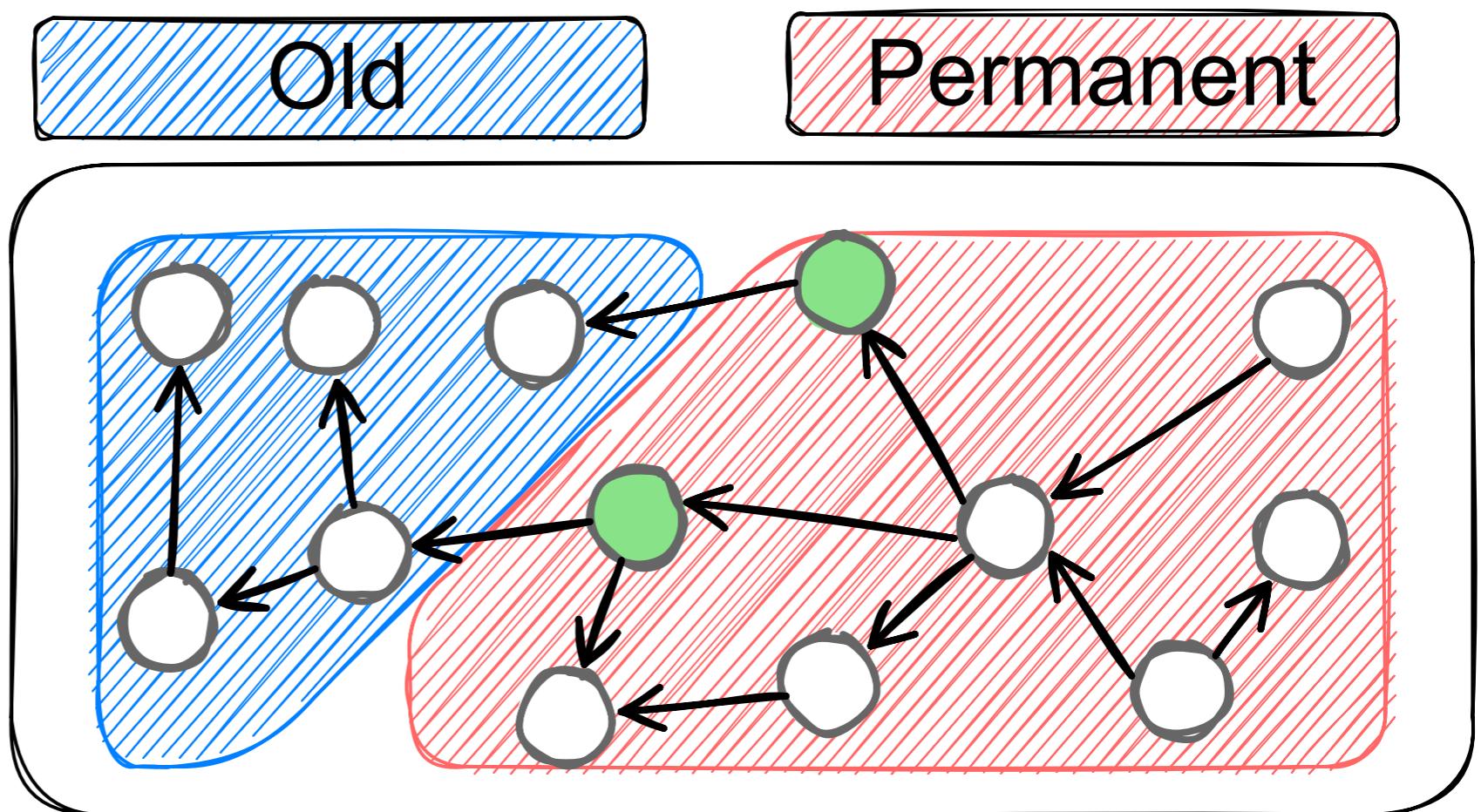
Separating Permanent Objects

- Permanent objects are **roots**
- But ***not all of them*** are roots
- We don't want to iterate all permanent objects!

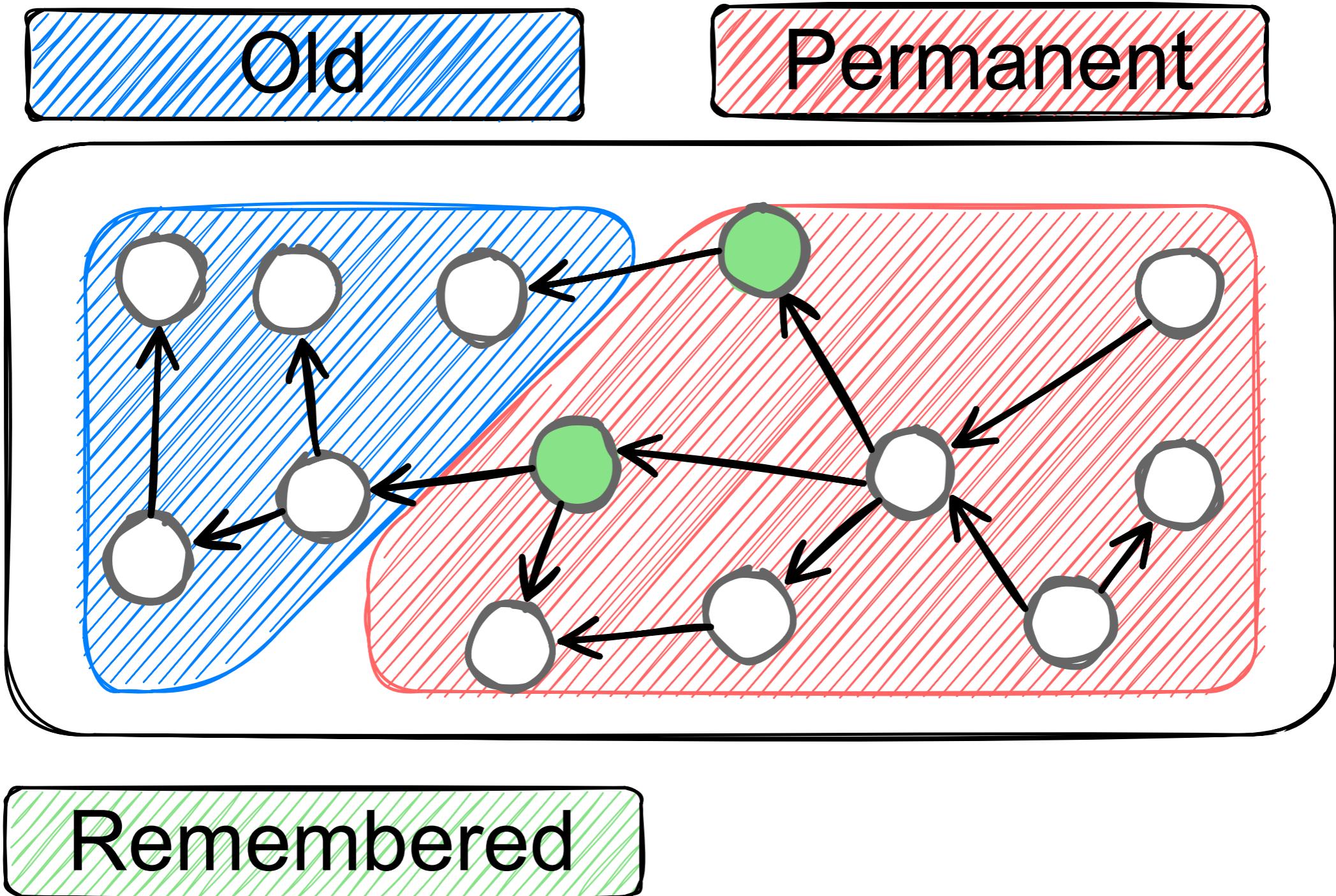


Maintaining a Remembered Set

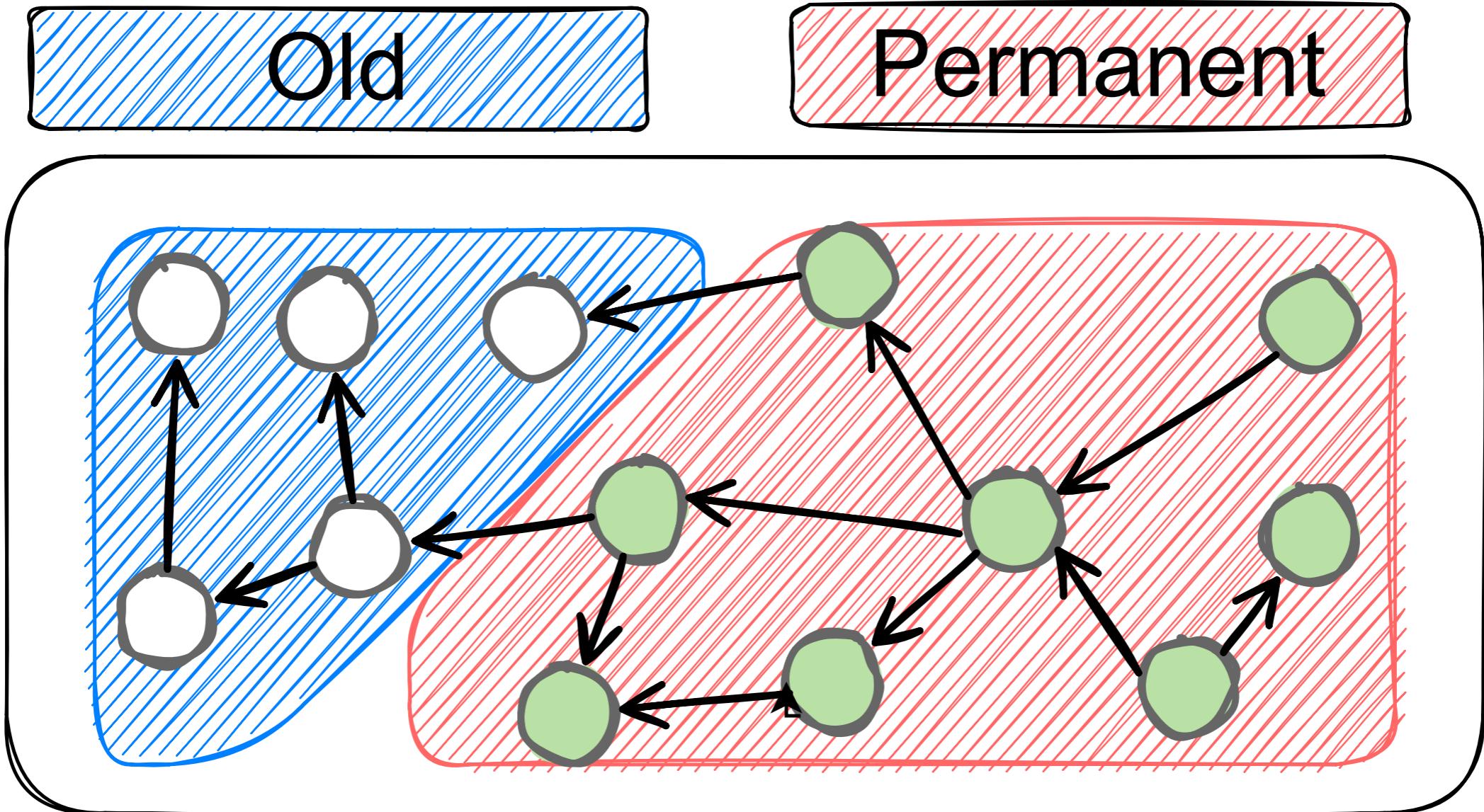
- Get the real roots in a *remembered set*
- Updated with a write barrier and cleaned at GC



Semi-permanent Object Selection



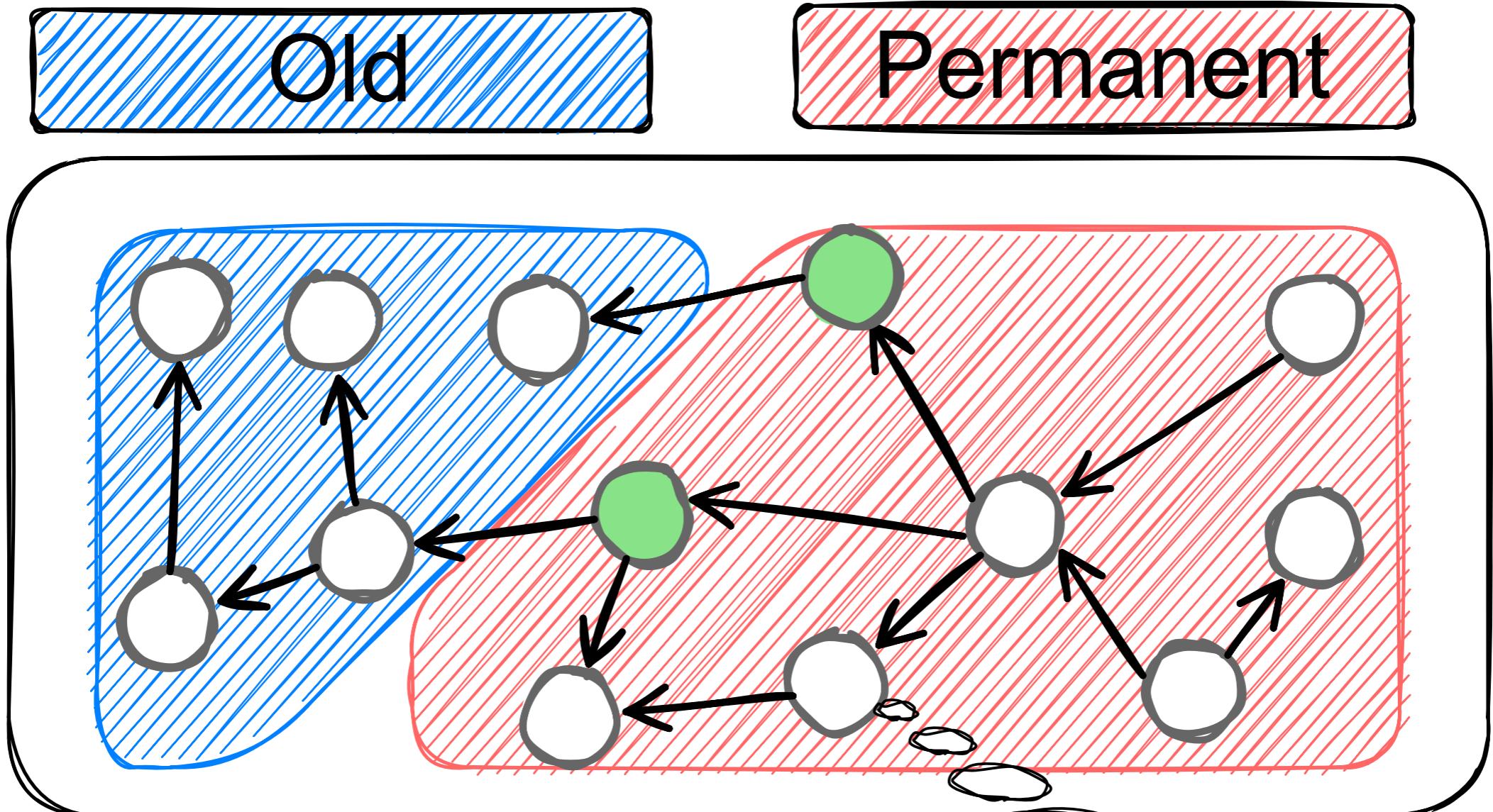
Bad Semi-permanent Object Selection



Remembered

**Worst case: all permanent
are remembered (!!)**

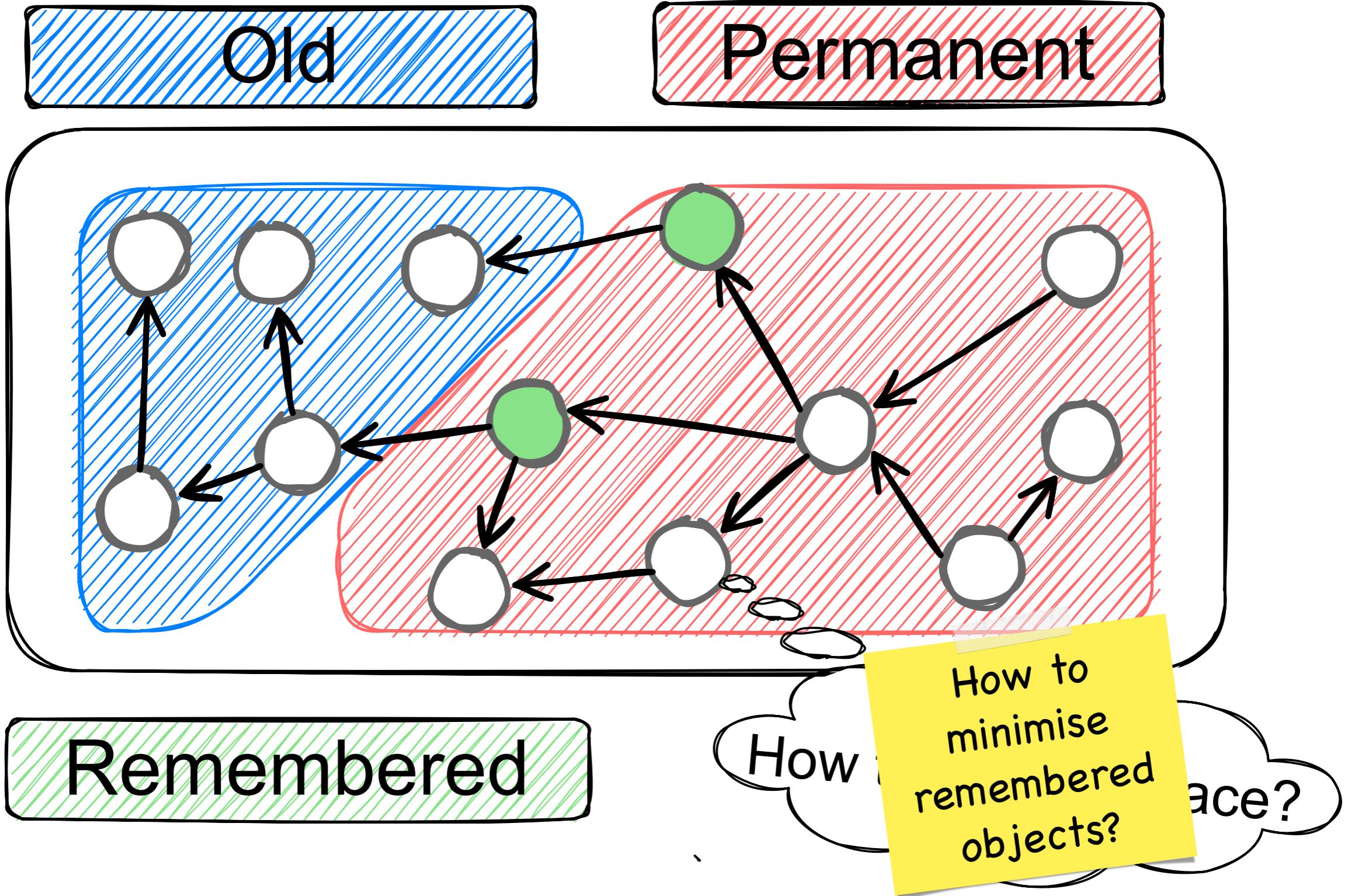
What objects should be permanent?



Remembered

How to cut the space?

What objects should be permanent?

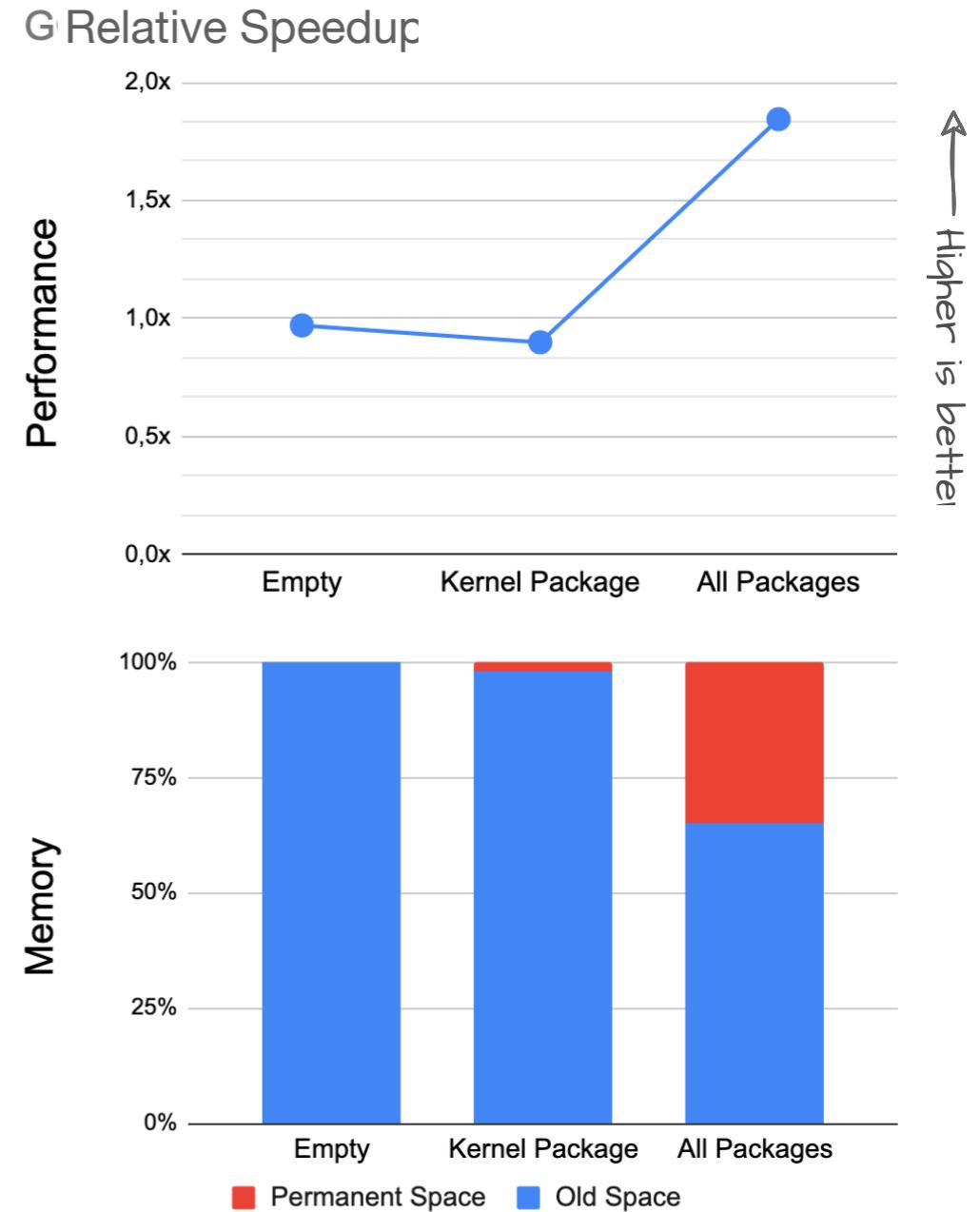


Pitfalls of Semi-permanent Object Selection

- The remembered set can *explode* easily. E.g.,
 - Objects that reference `nil`, `true`, `false` are ***always remembered***
 - If you make a *class* permanent
 - => you probably want to make its method dictionary too
 - => and its methods, and literals
 - => and ...

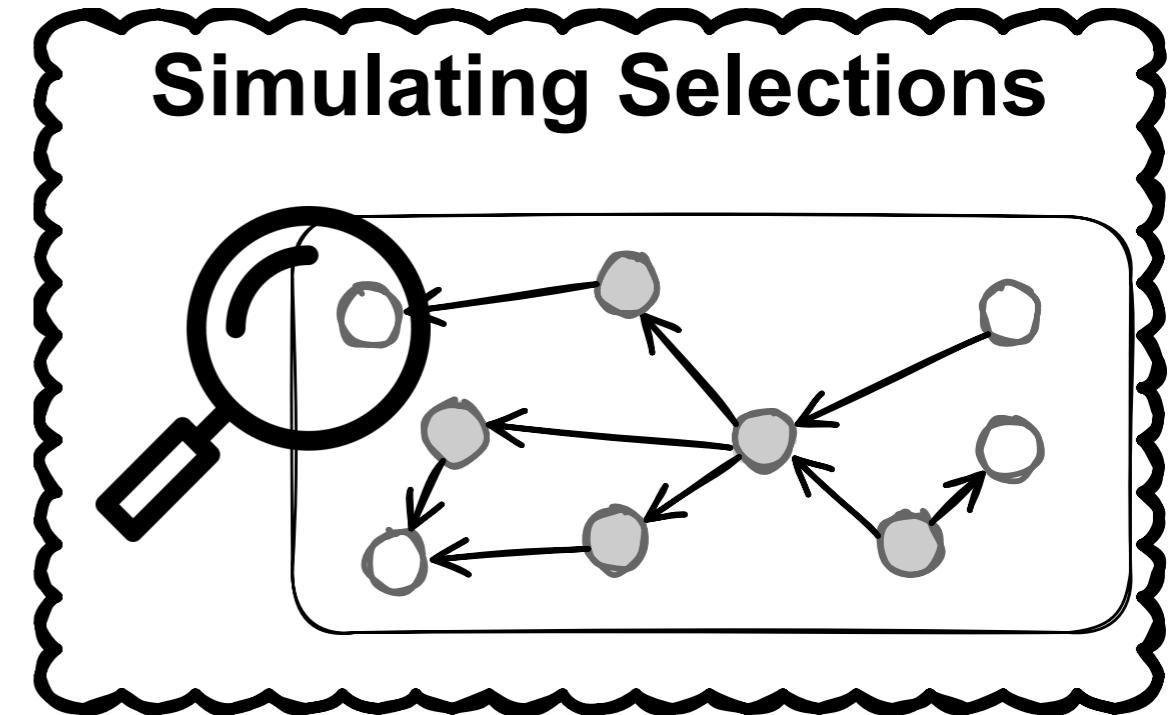
Potential: GC cut by half

- For production Apps!
- Some Heuristics:
 - Code (+related) is semi-permanent
 - Collections go with their inner array
 - Association values are not (!!)



Automatic Object Selection via Simulations

- Estimate
 - permanent segment size
 - remembered set size
- Understand the ***leaking*** reasons
- And extract better heuristics for production code
(e.g., better move all classes with all method dictionaries...)



Future Perspectives

- **Sharing** permanent immutable objects, copy on write
- **Scaling** multi-process applications
- **Application-specific** permanent object selection

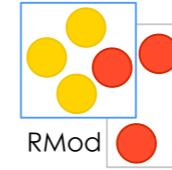


We are hiring!

- We have
 - Engineer Positions
 - Phd Positions
- Keywords: *Compilers, Interpreters, Memory Management, Security*
- **Come talk to us!**



Conclusion



- Multi-file snapshot format
- Permanent Objects and Selection
- 2x GC improvements

