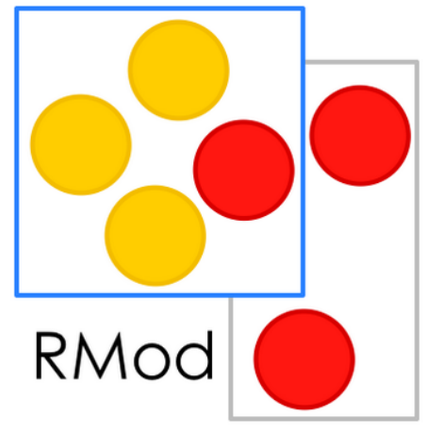
Pharo Scavenger



Pierre Misse-Chanabier
pierre.misse-chanabier@inria.fr

A plan of the VM

Object memory

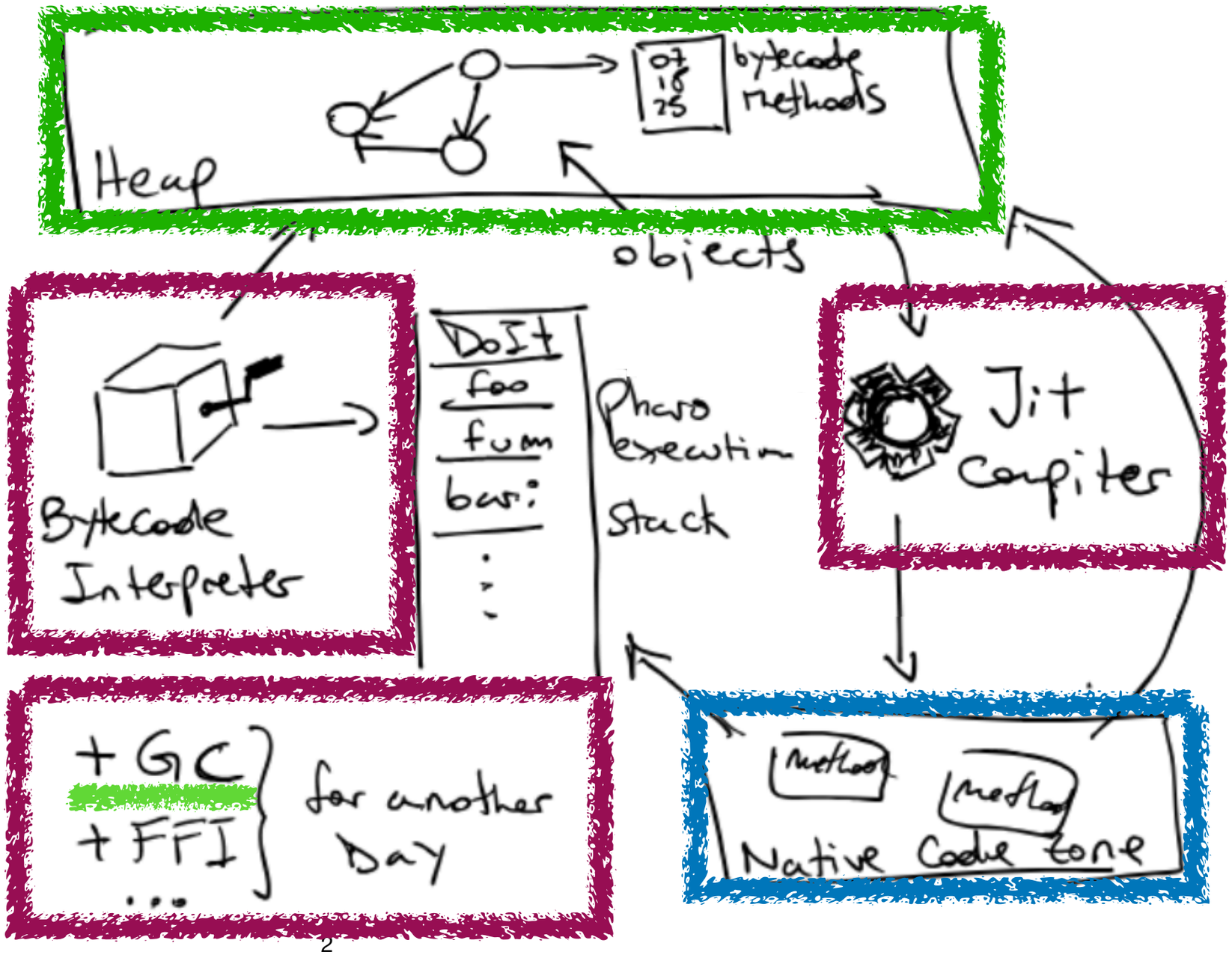
Memory managed by the VM
Garbage collected

VM C Runtime

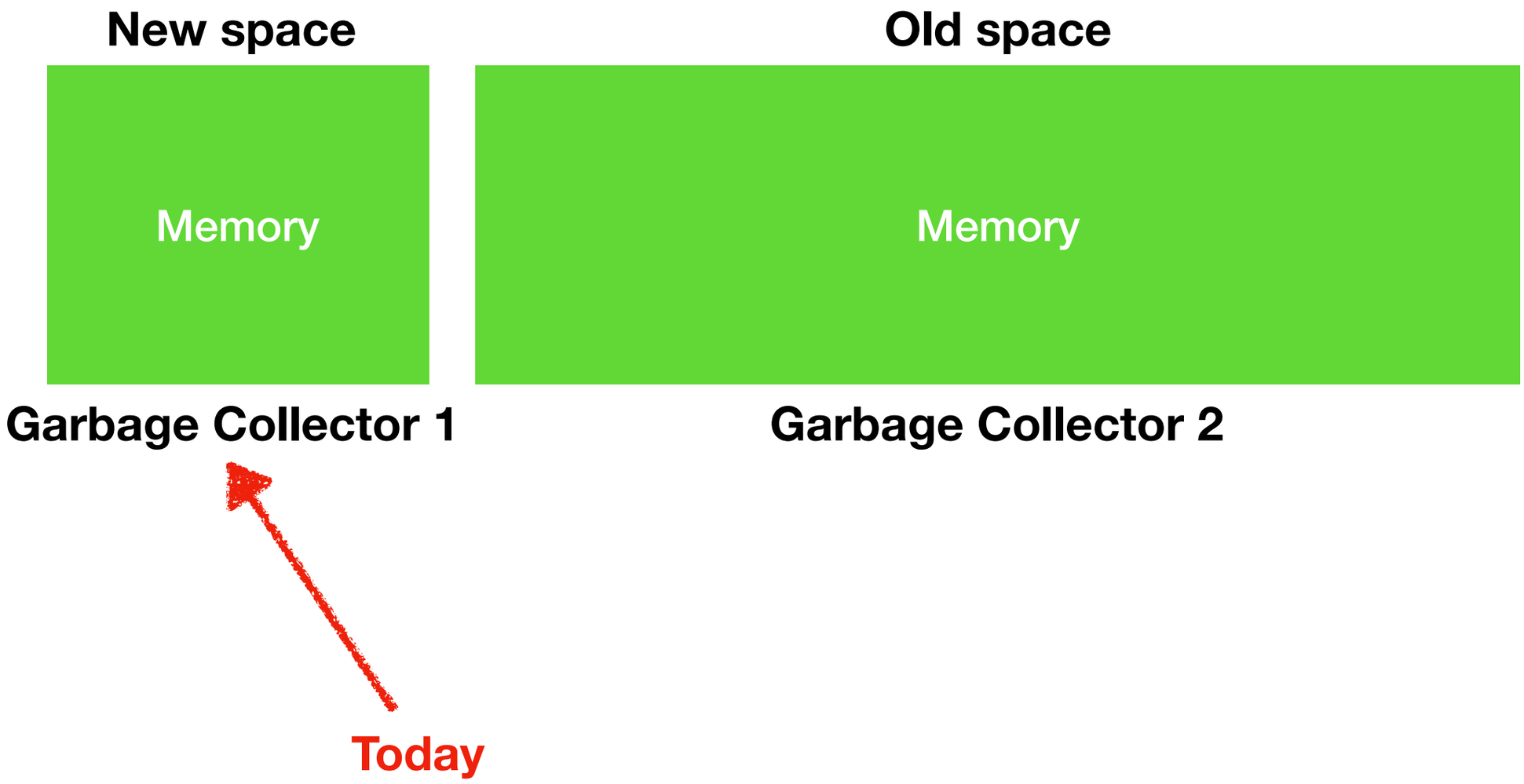
Compiled Ahead of Time
Slang -> C -> Native code

VM Generated Runtime

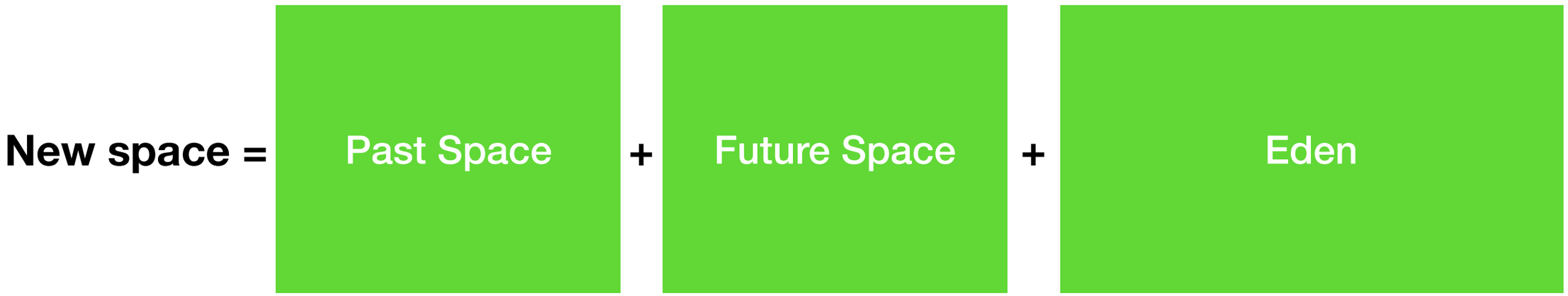
Compiled at Run Time
Bytecode -> Native code



The (managed) memory



The new space



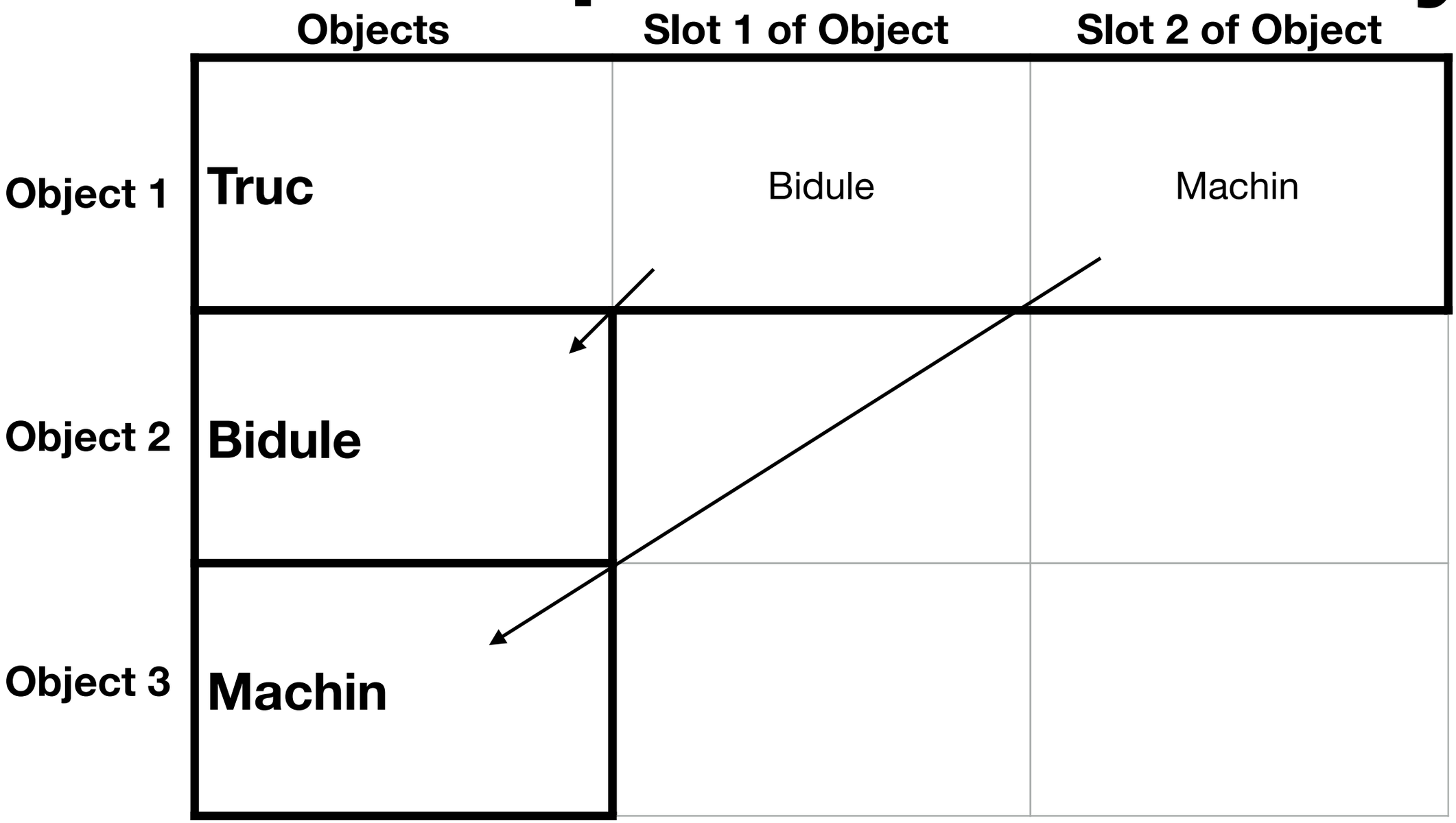
Scavenger = Copy Garbage collector

Memory representation

	Objects	Slot 1 of Object	Slot 2 of Object
Object 1	**Object1**	Slot1	Slot2
Object 2	**Object2**	Slot1	Slot2
Object 3	**…**	…	…

This is just a representation to be able to better follow the objects

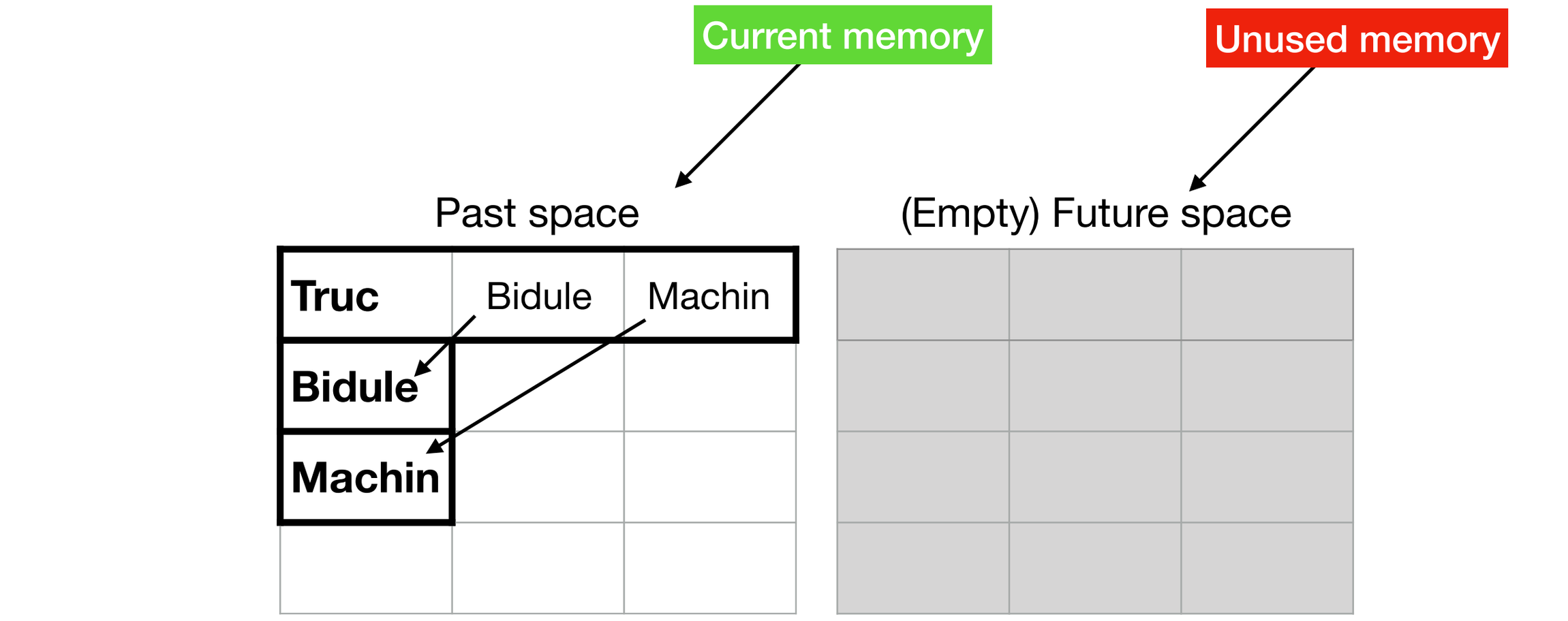
Exemple of memory

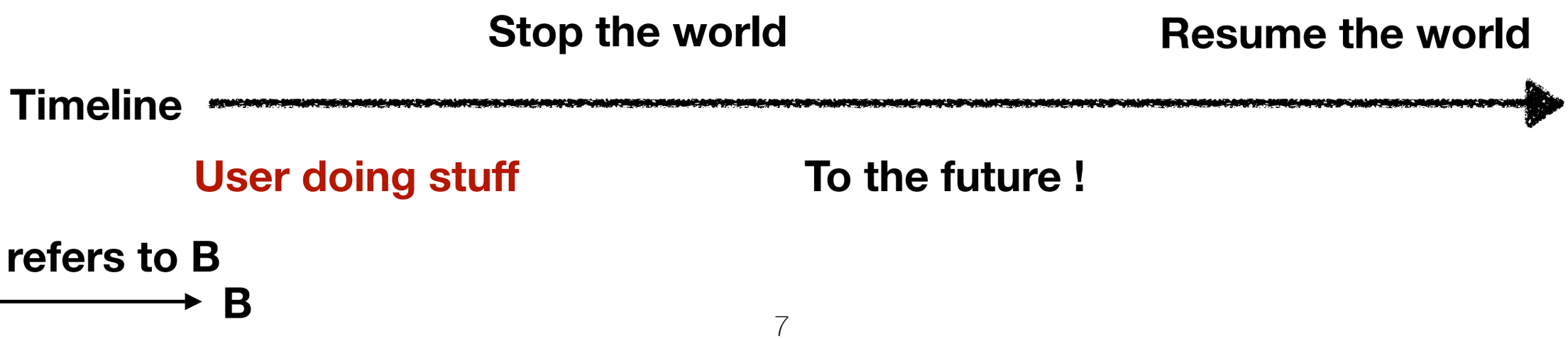


A refers to B

A → B

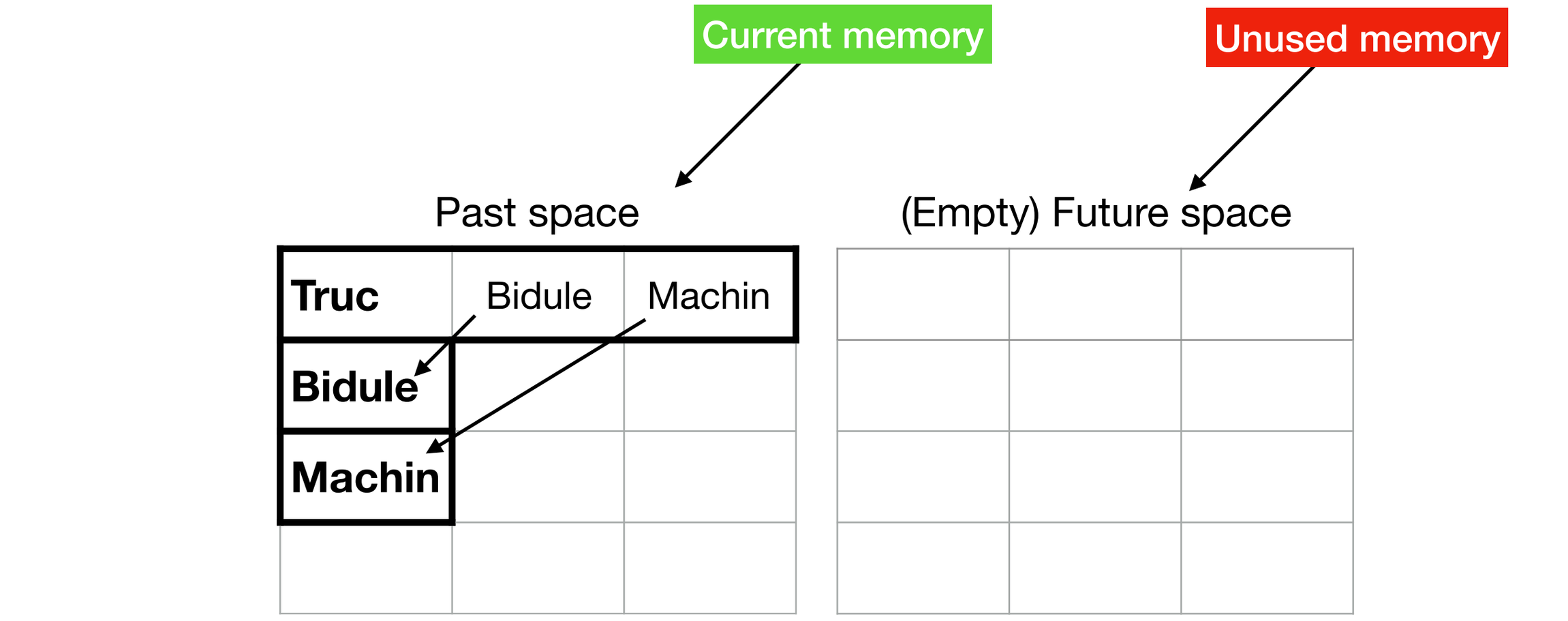
Exemple of memory

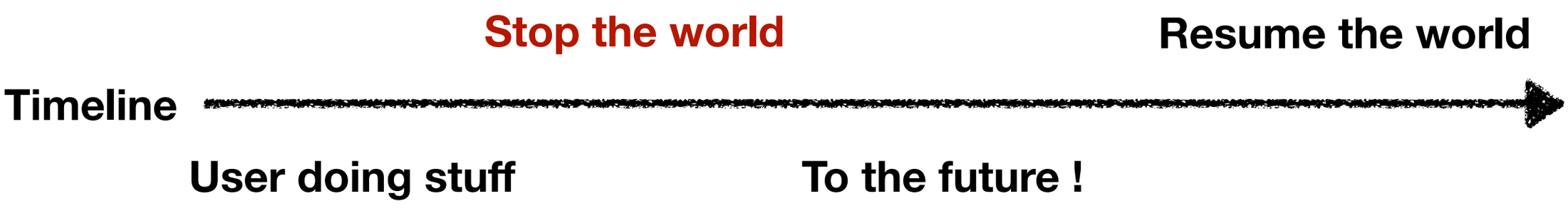




A refers to B

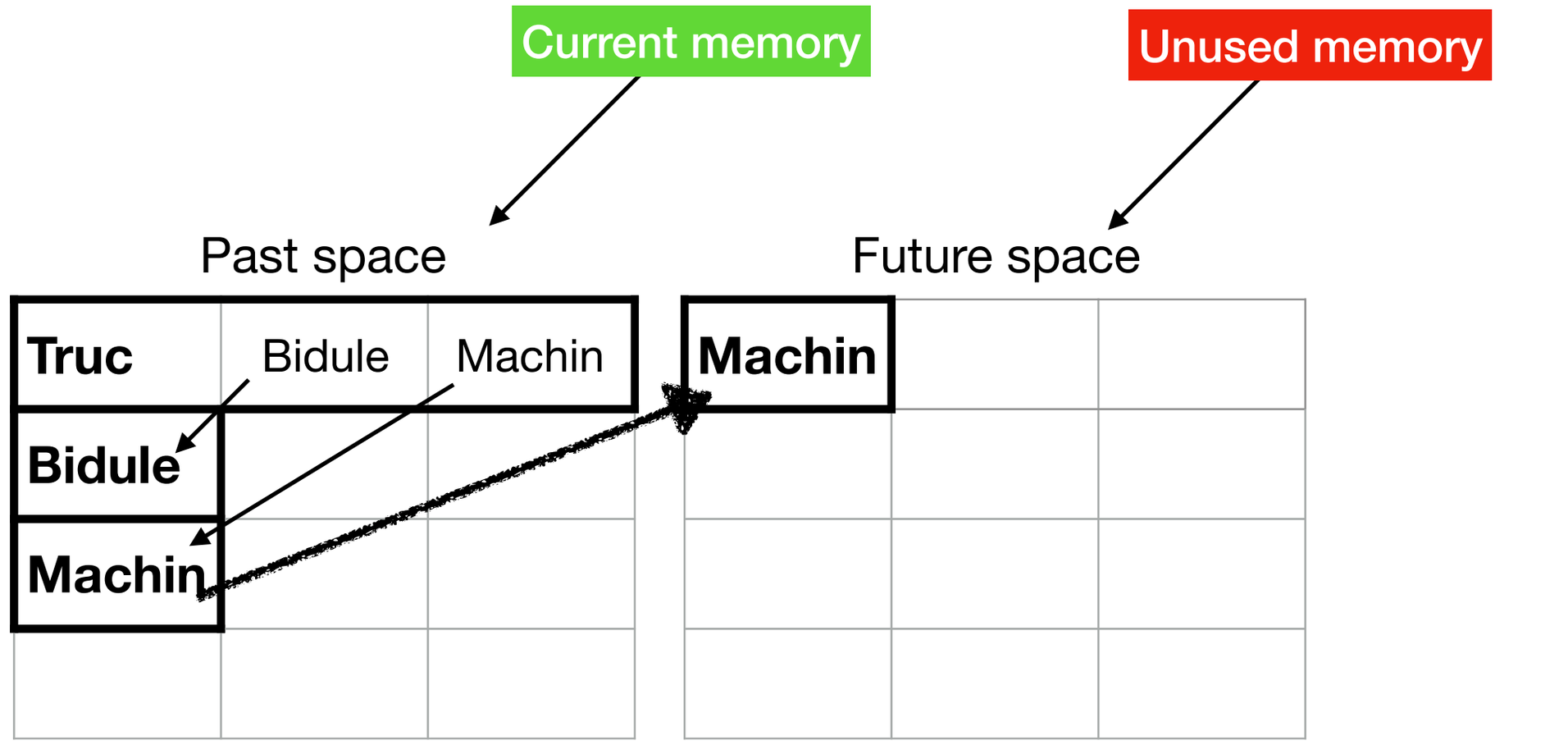
Let’s garbage collect it !





A refers to B

Copy objects



Stop the world

Resume the world

Timeline

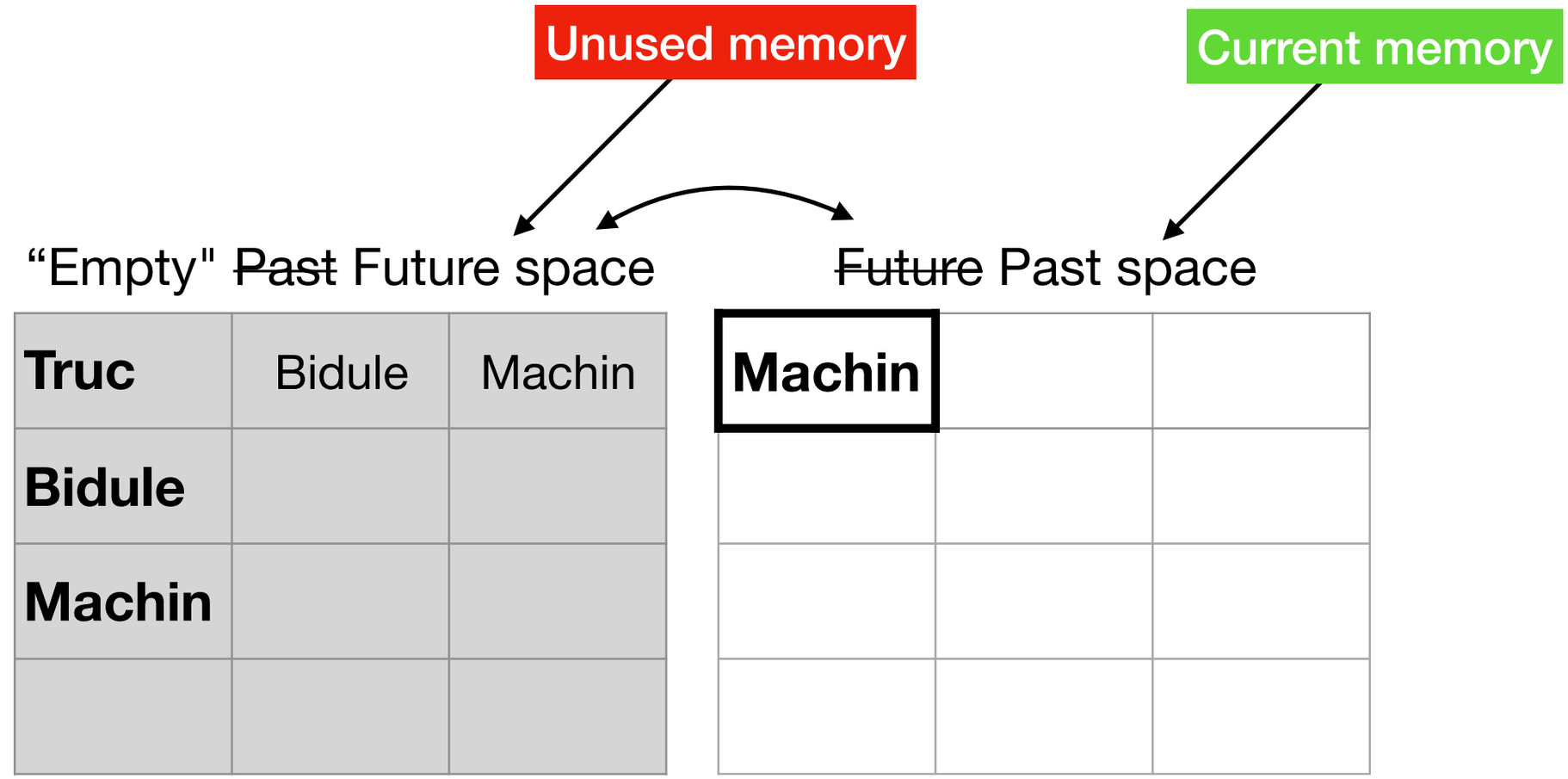
User doing stuff

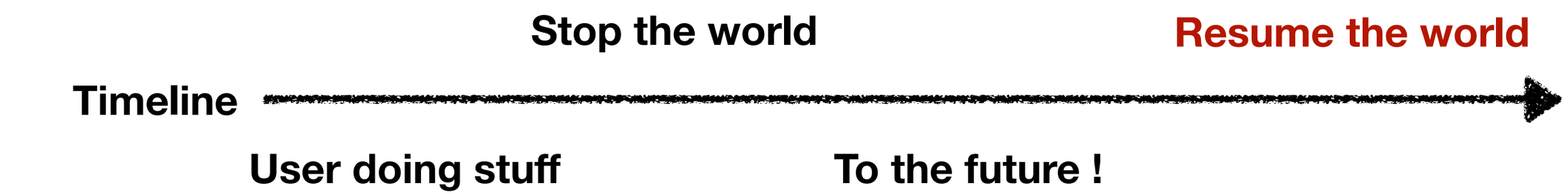
To the future !

Copy cell A contents to cell B

A → B

Invert the spaces





A refers to B

Important questions

- What’s the eden space ?
- When is garbage collection triggered ?
- What happens when future space is full ?
- How do we clean memory?
- How to choose what to copy to the future?
- How to reference the right object after it has been copied at a different location?
- How do we handle Ephemeron and Weak objects?

Future dedicated presentation

Eden

Current memory

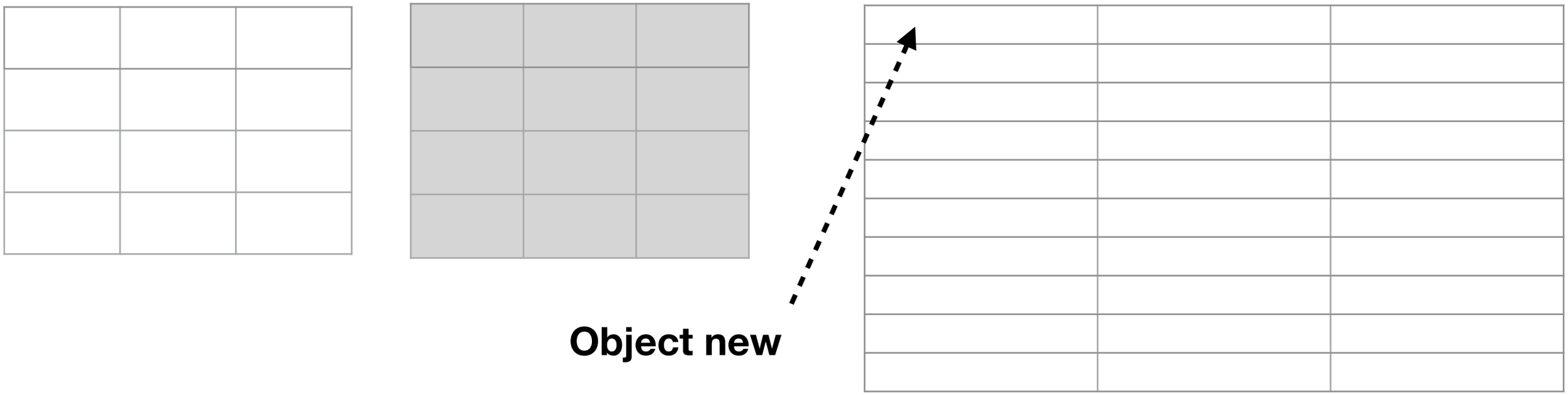
Unused memory

Allocation space

Past space

Future space

Eden



Object new

Stop the world

Resume the world

Timeline

User doing stuff

To the future !

Allocation into cell A

A

Eden

Current memory

Unused memory

Allocation space

Past space

Future space

Eden

anObj

Object new

Stop the world

Resume the world

Timeline

User doing stuff

To the future !

Allocation into cell A

A

Eden's full !

Current memory

Unused memory

Allocation space

Past space

Future space

Eden

anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		

Object new

Stop the world

Resume the world

Timeline

User doing stuff

To the future !

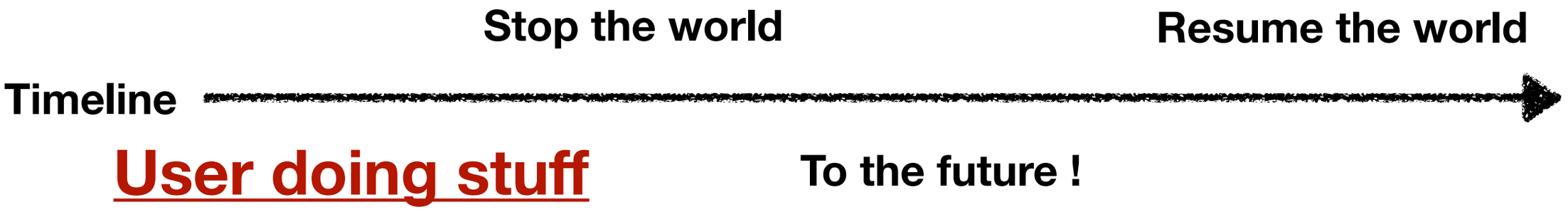
Allocation into cell A

A

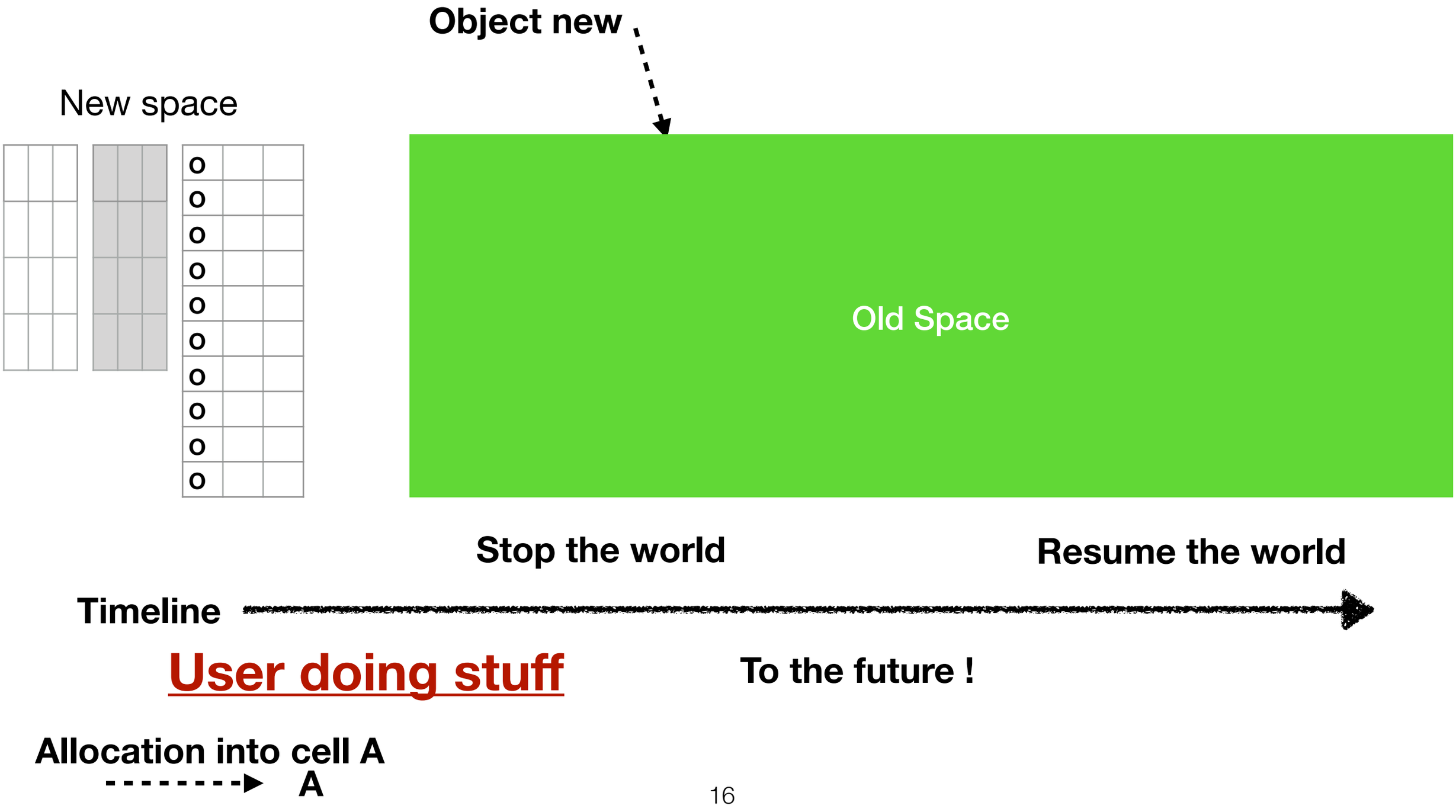
We ask for a scavenge whenever possible

SpurMemoryManager >> #allocateSlots:format:classIndex:isPinned:

```
freeStart + numBytes > scavengeThreshold
  ifTrue: [
    self scheduleScavenge.
    ^self
      allocateSlotsInOldSpace: numSlots
      bytes: numBytes format: formatField classIndex: classIndex ].
```



And allocate in the old space



Later, scavenge is triggered

checkForEventsMayContextSwitch: mayContextSwitch

```
objectMemory needGCFlag ifTrue:
  ["sufficientSpaceAfterGC: runs the incremental GC and
   then, if not enough space is available, the fullGC."
   (objectMemory sufficientSpaceAfterGC: 0) ifFalse:
    [self setSignalLowSpaceFlagAndSaveProcess]].
```

Happens mostly during method/block activations

Now future space is full !

Current memory

Unused memory

Allocation space

Past space

Future space

Eden

anObj		
anObj		
anObj		

anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		

Overflow !

Stop the world

Resume the world

Timeline

User doing stuff

To the future !

Copy cell A contents to cell B

A → B

Copy to the old space

Stop the world

Resume the world

Timeline

User doing stuff

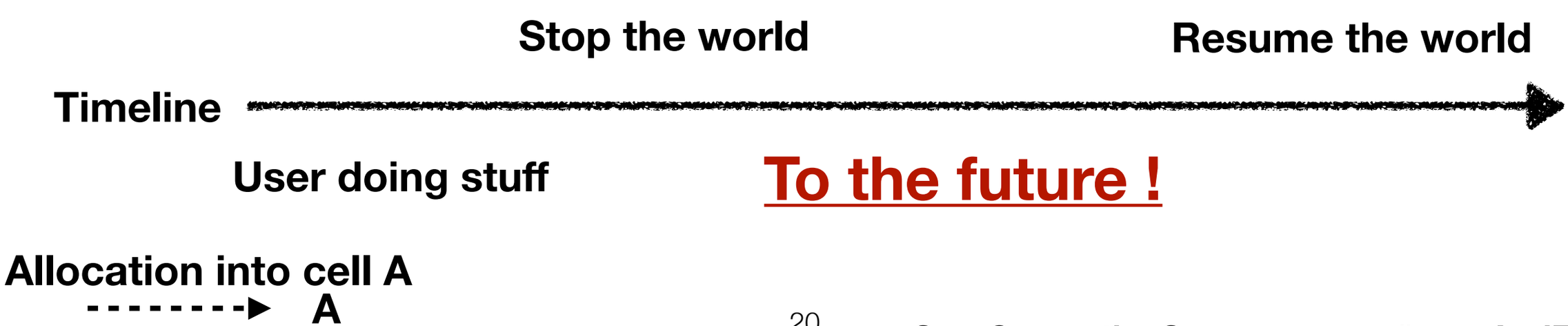
To the future !

Copy cell A contents to cell B

A → **B**

Tenuring

```
(tenure or: [futureSurvivorStart + bytesInObj > futureSpace limit])
    ifTrue: [self copyToOldSpace: survivor bytes: bytesInObj format: format]
    ifFalse: [self copyToFutureSpace: survivor bytes: bytesInObj].
```



SpurGenerationScavenger >> #copyAndForward:

No time to clean !

Unused memory

Current memory

Allocation space

~~Past~~ Future space

~~Future~~ Past space

Eden

anObj		
anObj		
anObj		
anObj		

anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		

Array new

Stop the world

Resume the world

Timeline

User doing stuff

To the future !

Allocation into cell A

A

Override the memory

Unused memory

Current memory

Allocation space

~~Past~~ Future space

~~Future~~ Past space

anObj		
anObj		
anObj		

Eden

anArray		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		

Stop the world

Resume the world

Timeline →

User doing stuff

To the future !

Allocation into cell A

- - - - - - - - ▶ **A**

And repeat !

Unused memory

Current memory

Allocation space

~~Past~~ Future space

~~Future~~ Past space

Eden

anObj		
anObj		
anObj		

anArray		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		
anObj		

Array new

Stop the world

Resume the world

Timeline

User doing stuff

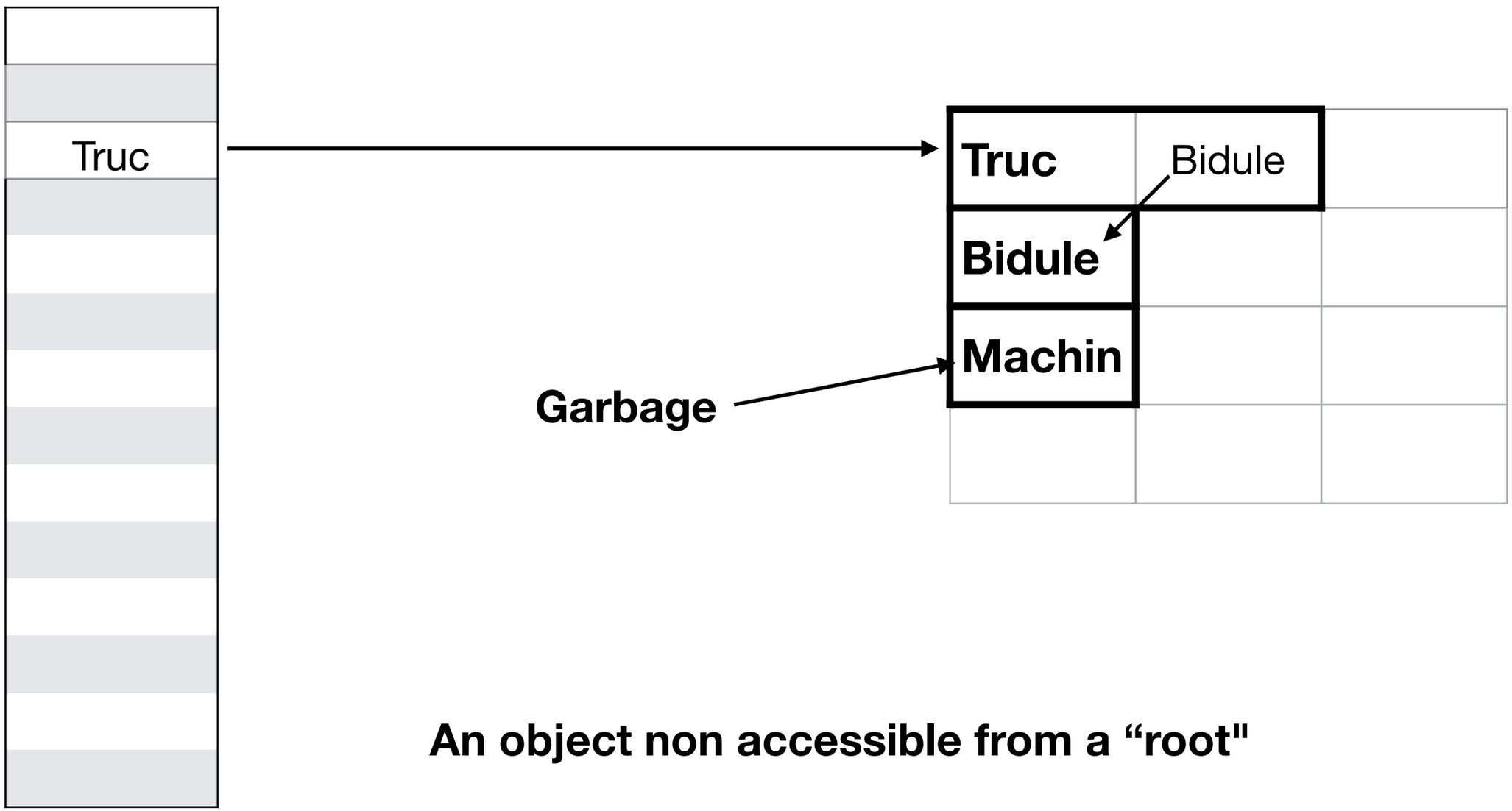
To the future !

Allocation into cell A

A

What is garbage?

Execution stack



An object non accessible from a “root"

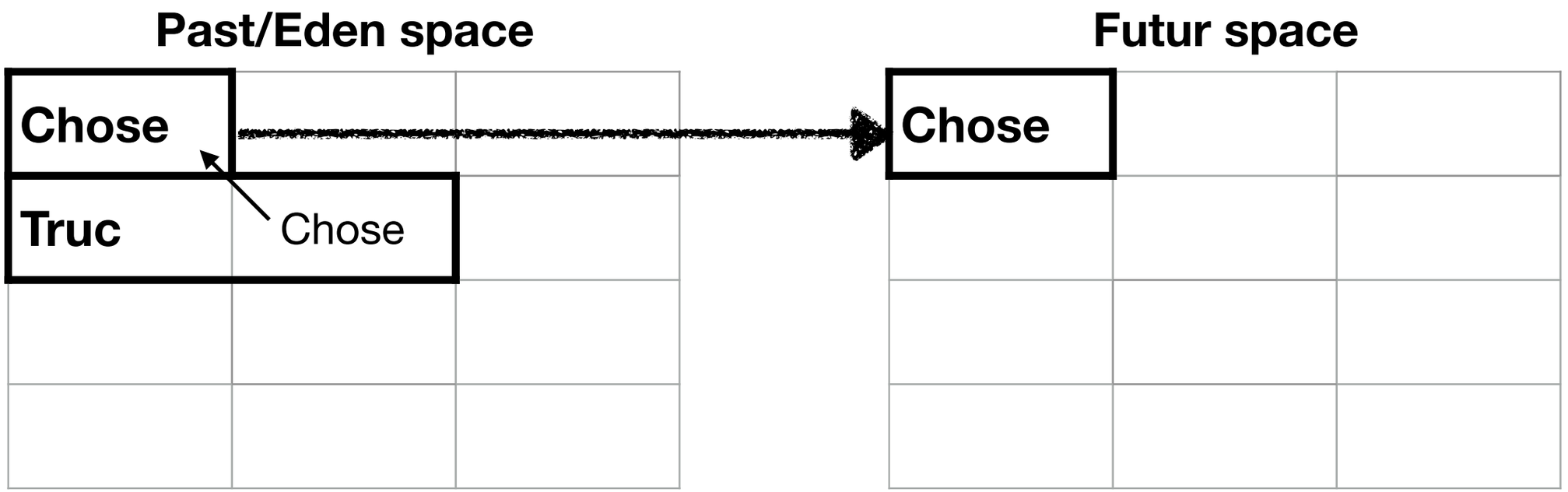
Other Roots

- Interpreter variables (newMethod, tempOop..)
- Stack pages
- Class tables pages
- …

Move an object

With *forwarders*

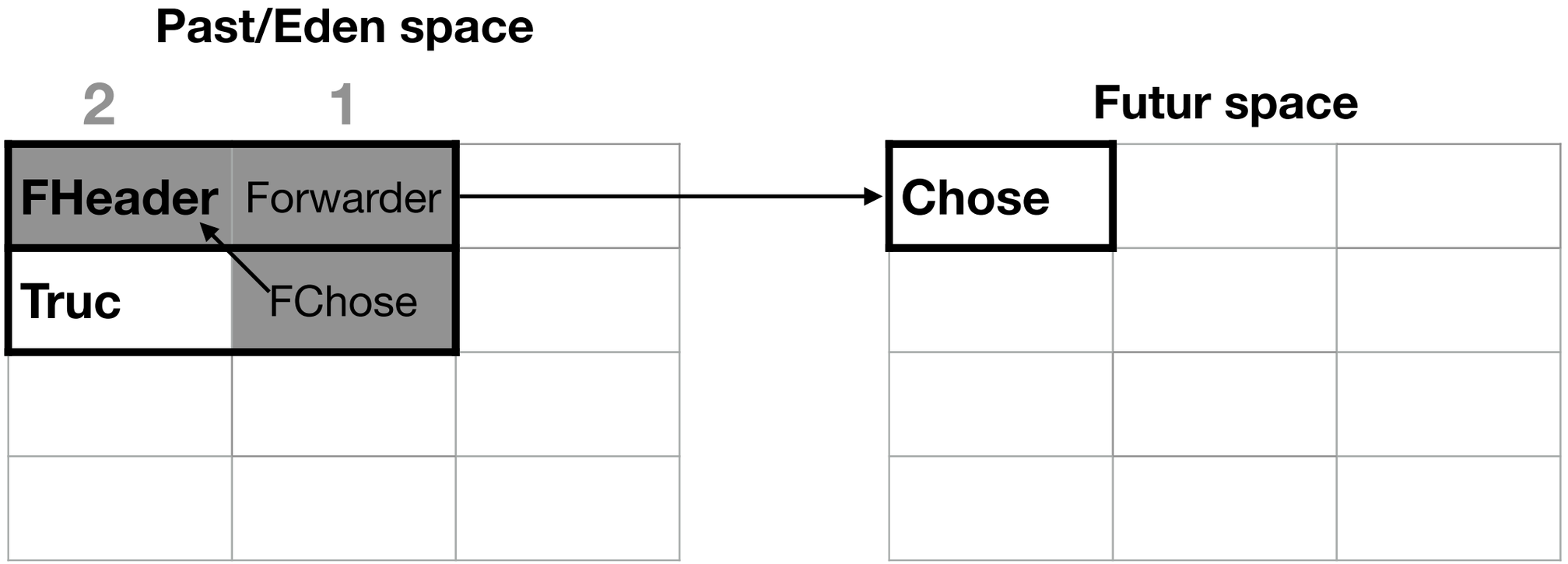
Copy an object



Copy

Refers to

Leaves a forwarder behind

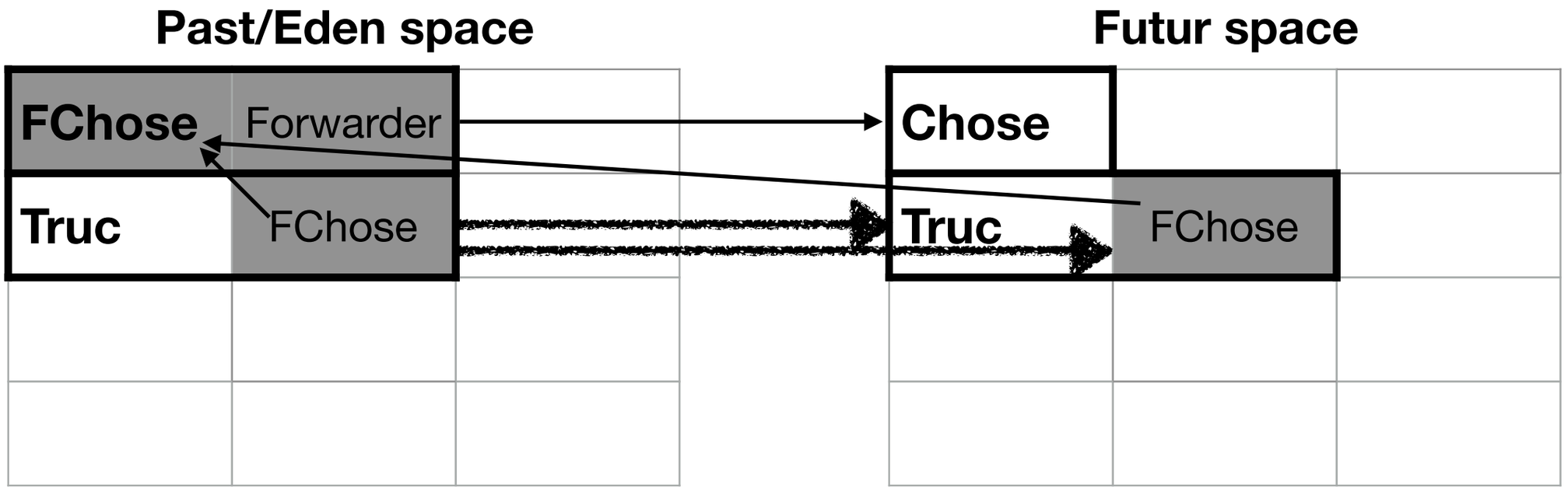



```
forwardSurvivor: obj1 to: obj2
1  self storePointerUnchecked: 0 ofObject: obj1 withValue: obj2.
   self set: obj1
2    classIndexTo: self isForwardedObjectClassIndexPun
     formatTo: self forwardedFormat
```

Copy

Refers to

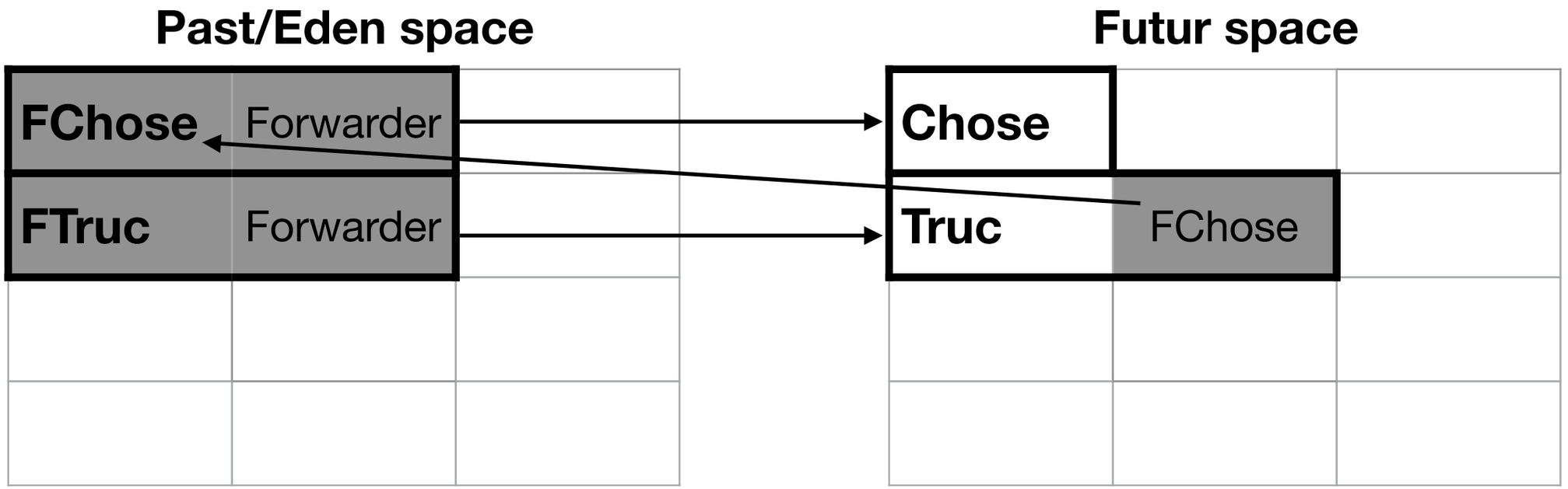
Copy another object…



Copy

Refers to

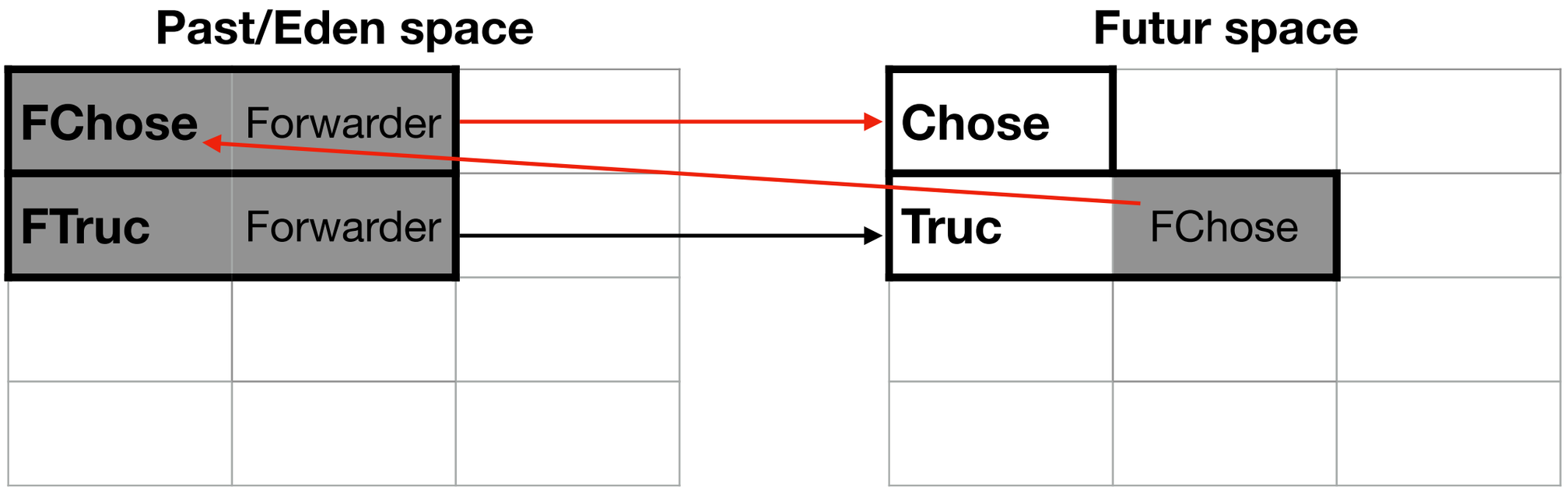
…Also leaves a forwarder



Copy

Refers to

Scavenge Futur space

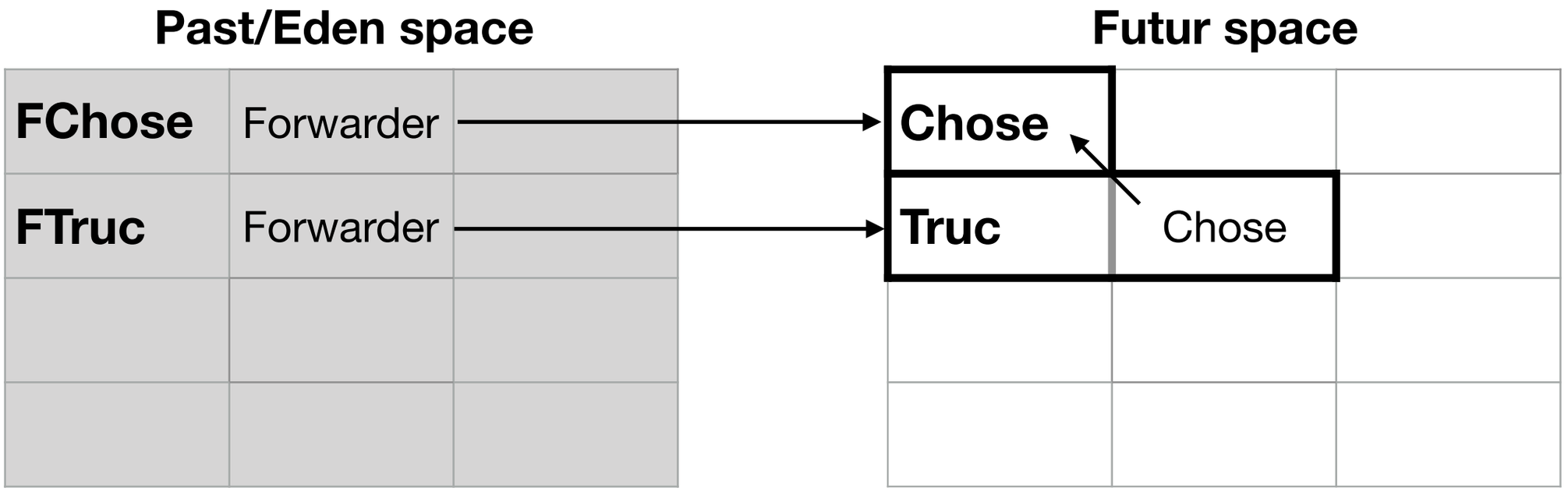


Replace forwarder by direct reference to chose

Copy

Refers to

Done !



There is no forwarder left after garbage collection

Copy

Refers to

Flashback to Layout presentation

Objects size

- Every object must have at least one slot
  - Minimum size is 16 bytes
  - It is hidden from the image side (smalltalk code)
  - In case it becomes into a Forwarder

- Even nil, true and false have one slot!

http://rmod-files.lille.inria.fr/Team/Presentations/2020-09-29-VM-Hernandez-objectsLayout.pdf

Other usages of forwarders

- (efficient) become
- Compaction
- …

Copy Garbage Collector and eden size

Hypothesis:

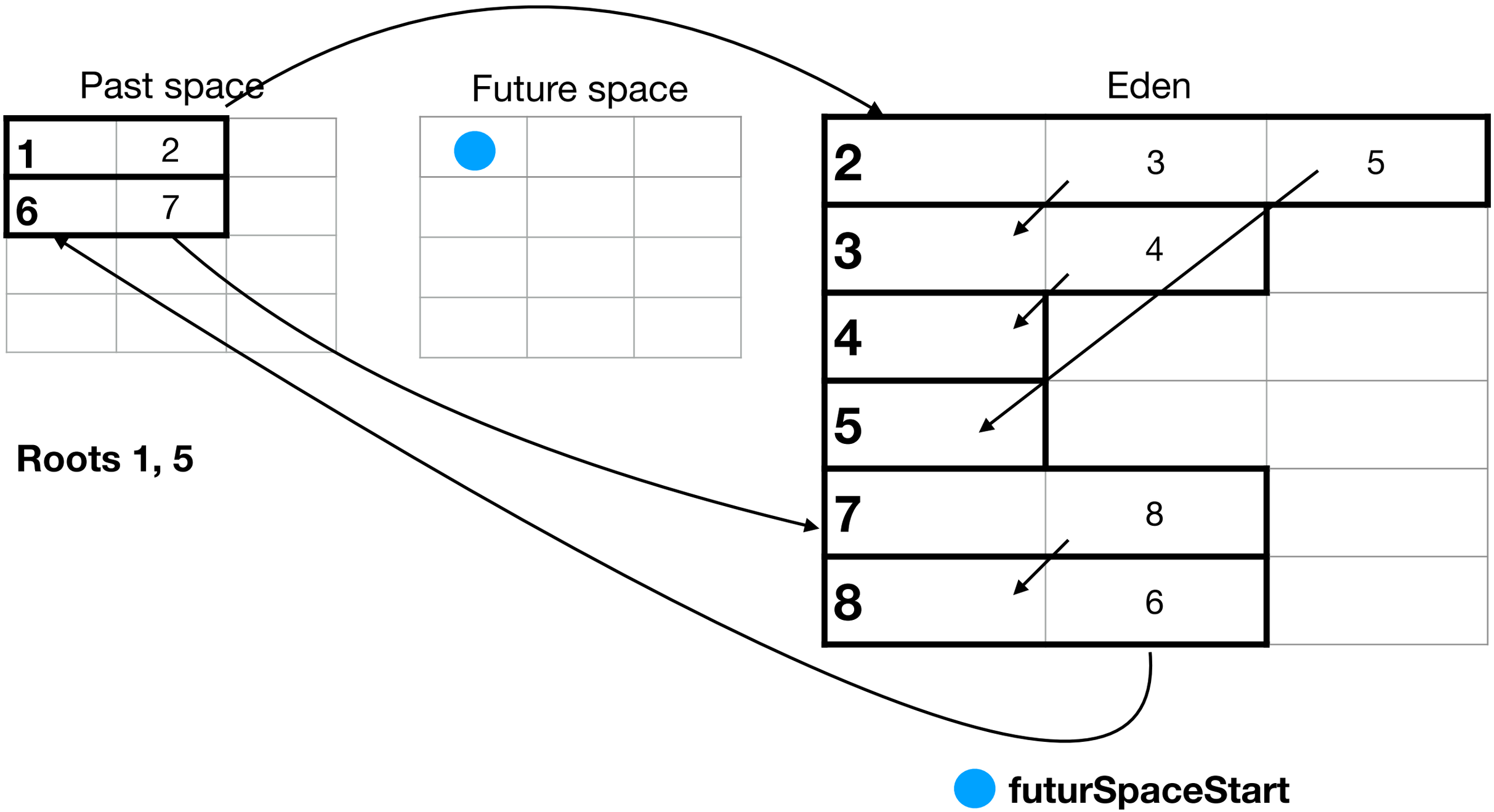
- Young objects die quickly.
- Many objects die quickly.

Results in:

- Few copies are made
- Eden is big to scavenge less often

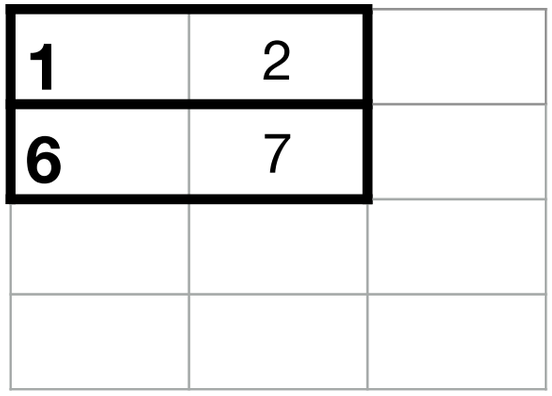
Now a more complex example !

The setup


Past space
Future space
Eden
1
2
6
7
2
3
5
3
4
4
5
7
8
8
6
Roots 1, 5
futurSpaceStart

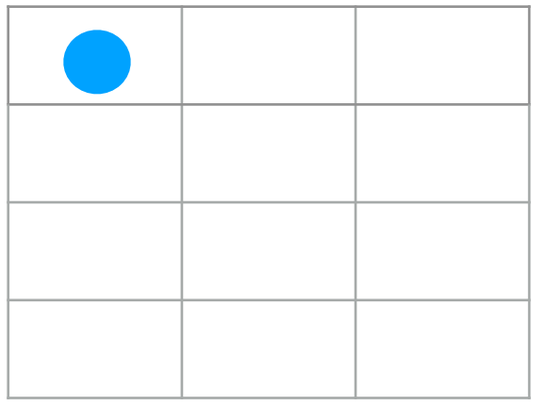
Copy the roots

Past space

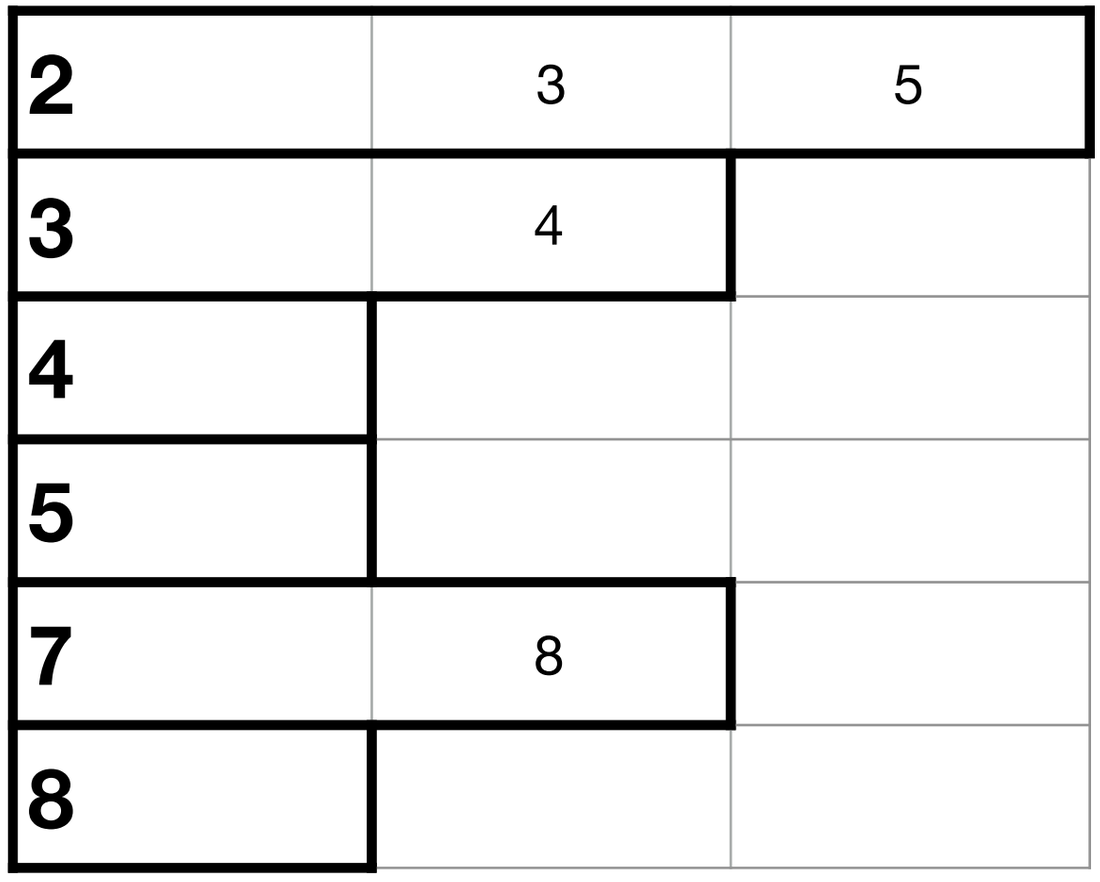


Roots 1, 5

Future space

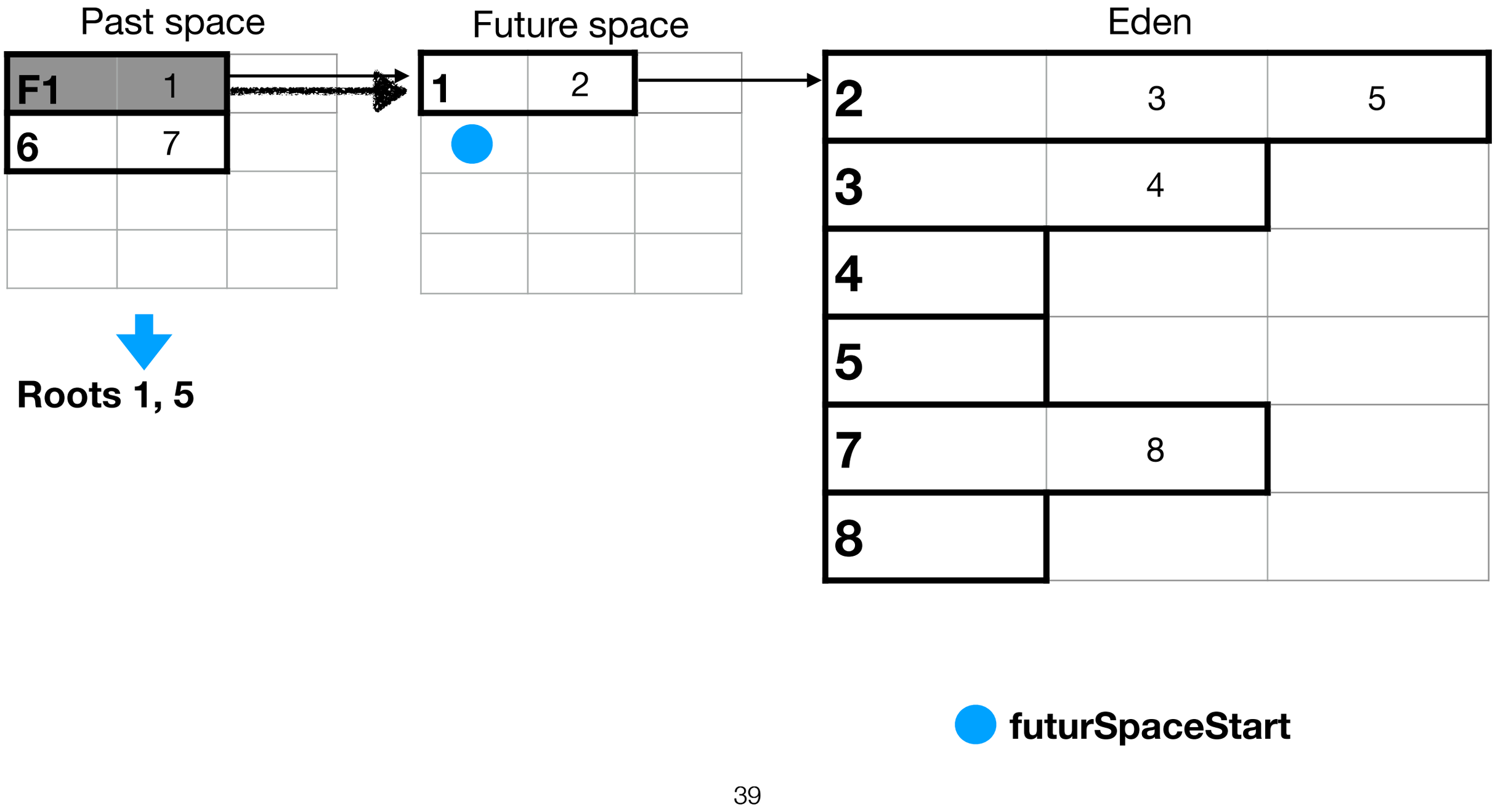


Eden

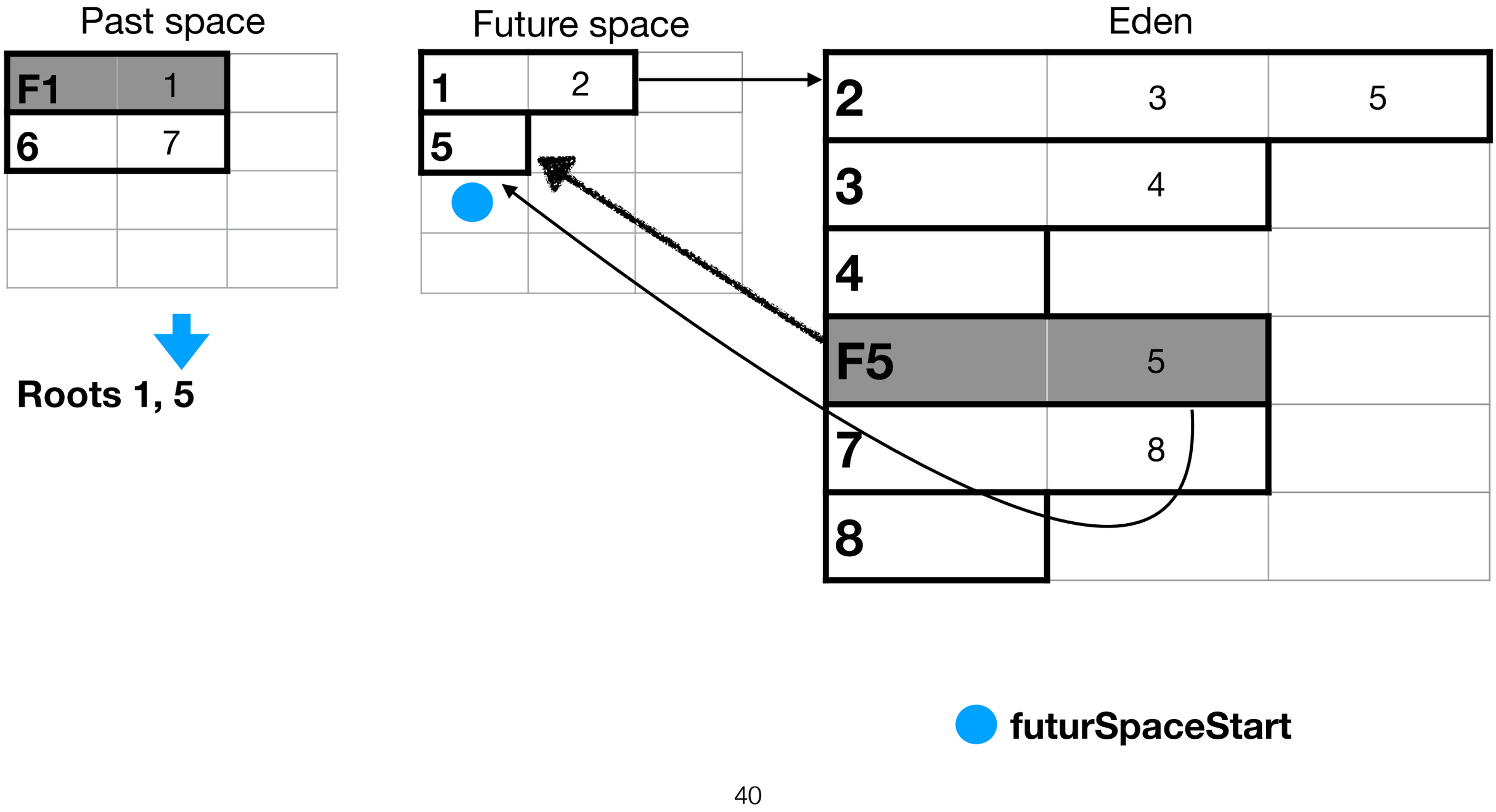


futurSpaceStart

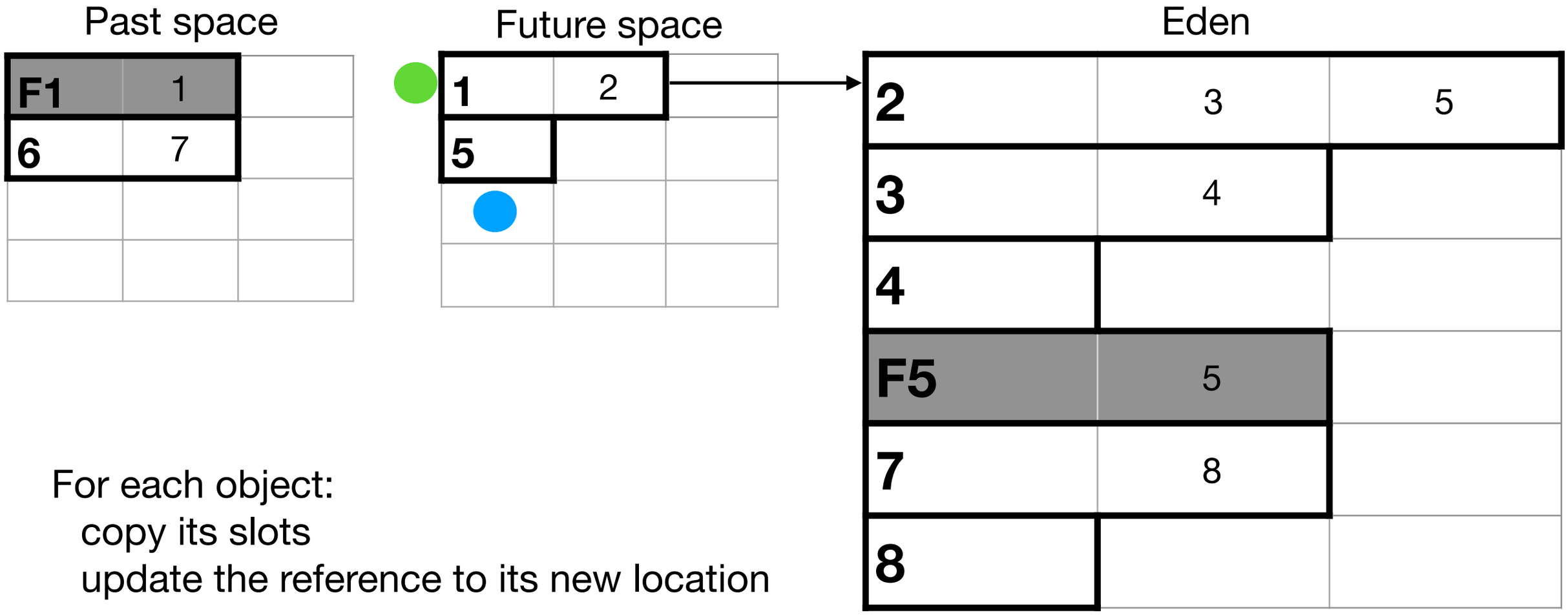
Copy 1 to the future



Copy 5 to the future

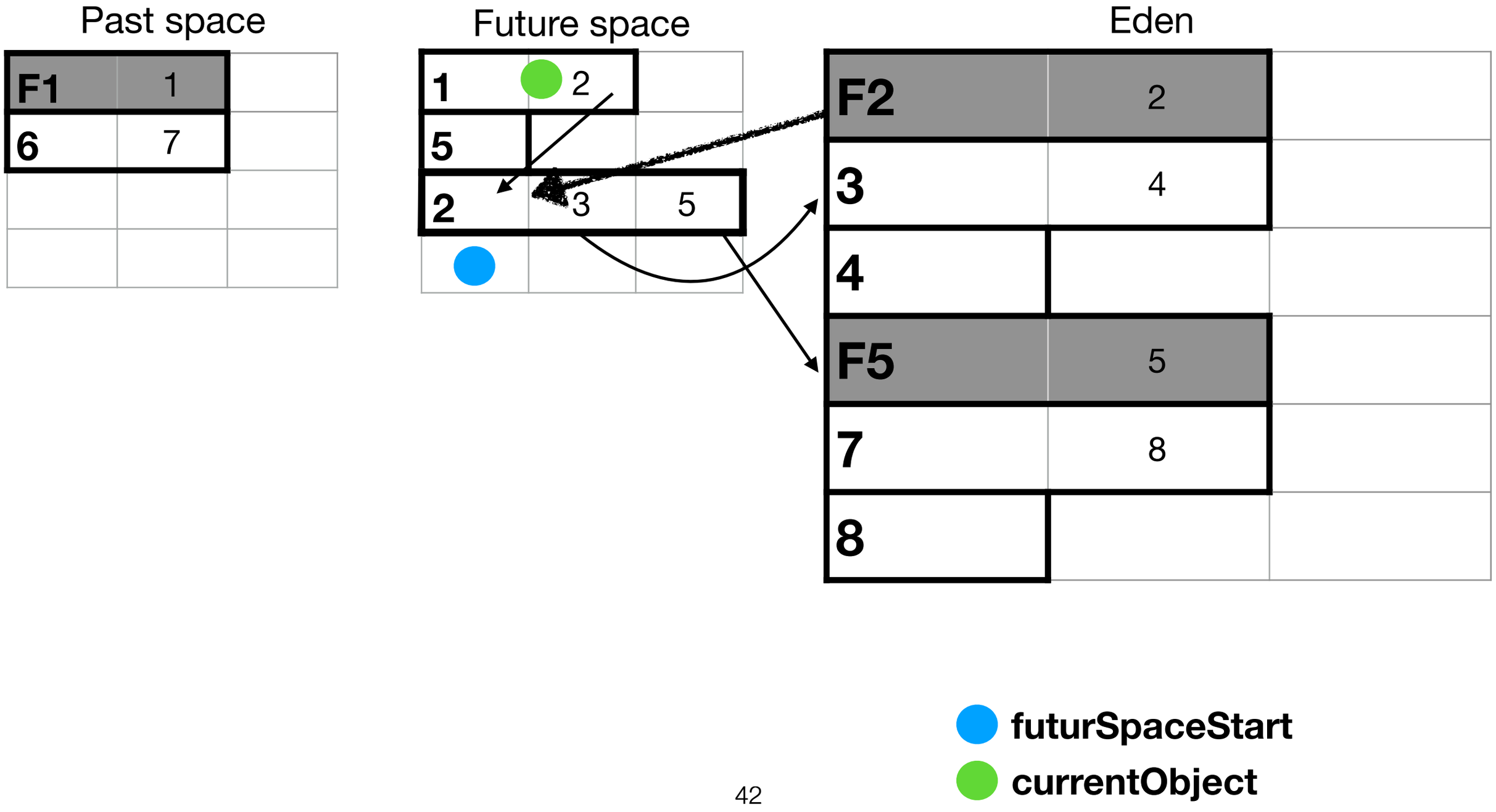


Scavenge the future space

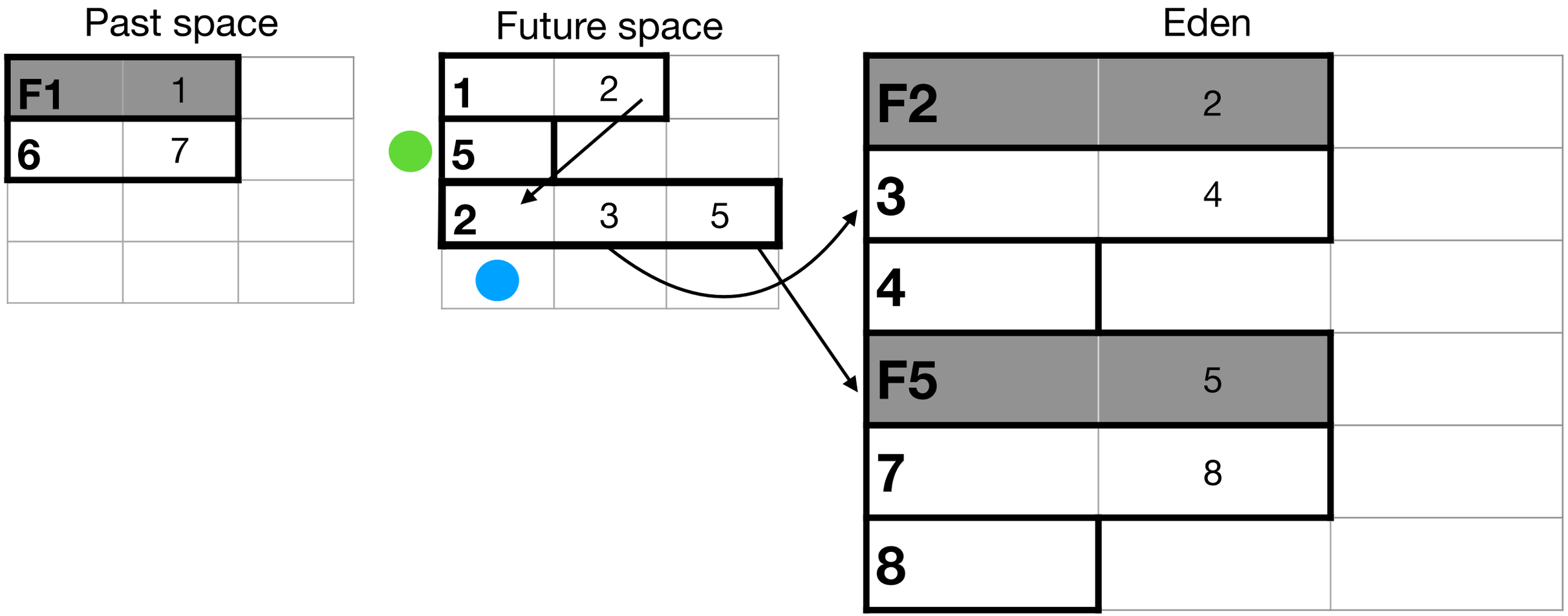


For each object:
  copy its slots
  update the reference to its new location

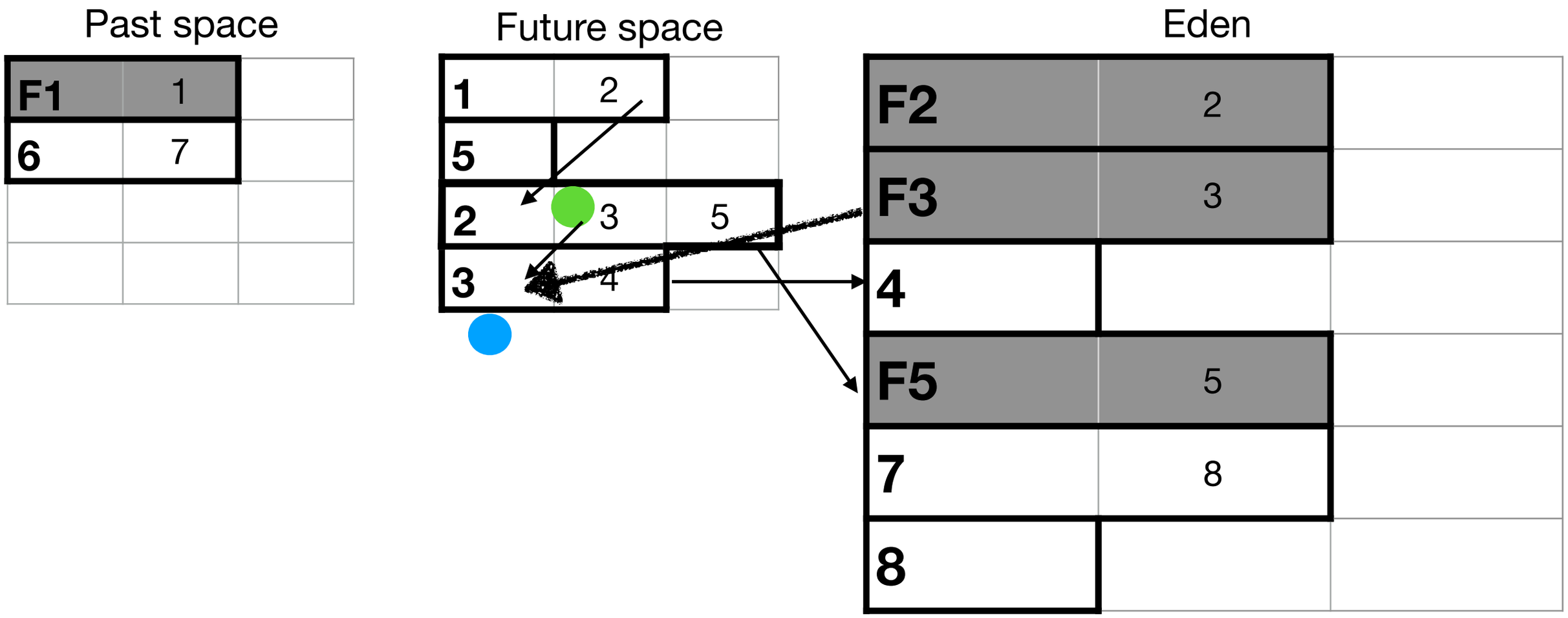
Copy 2 to the future



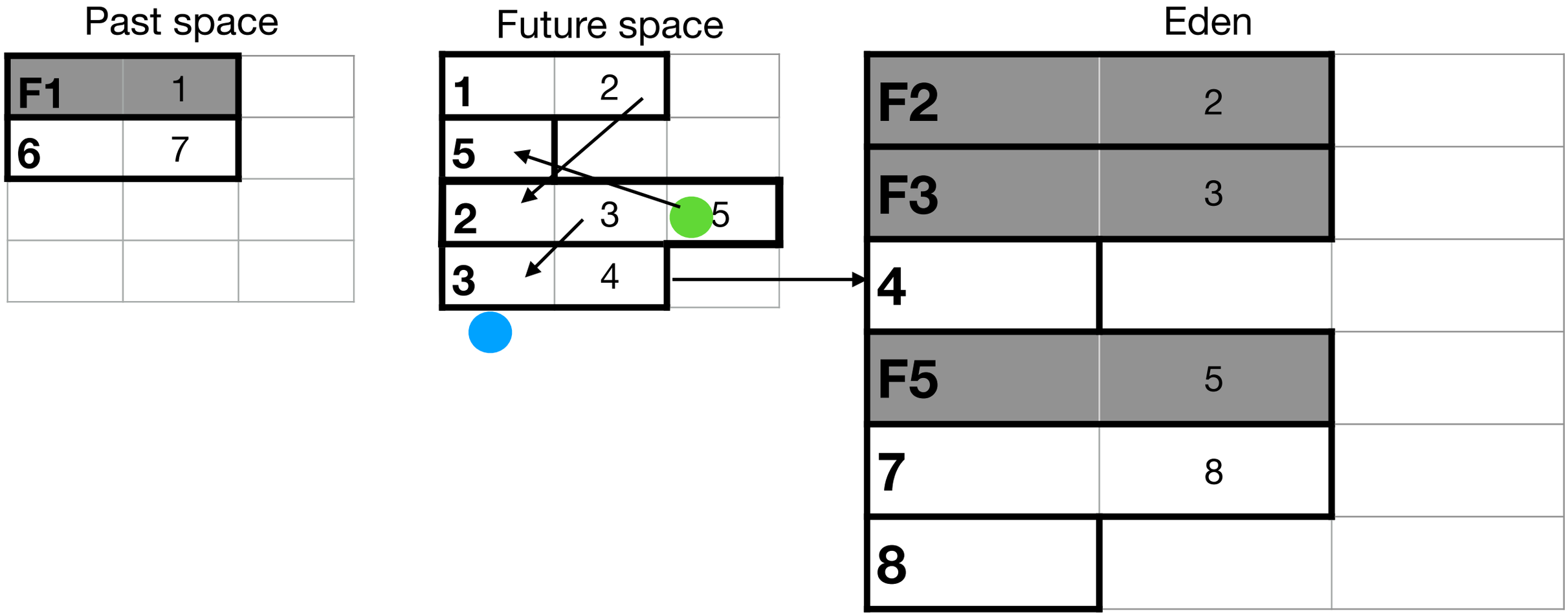
5 has no slots



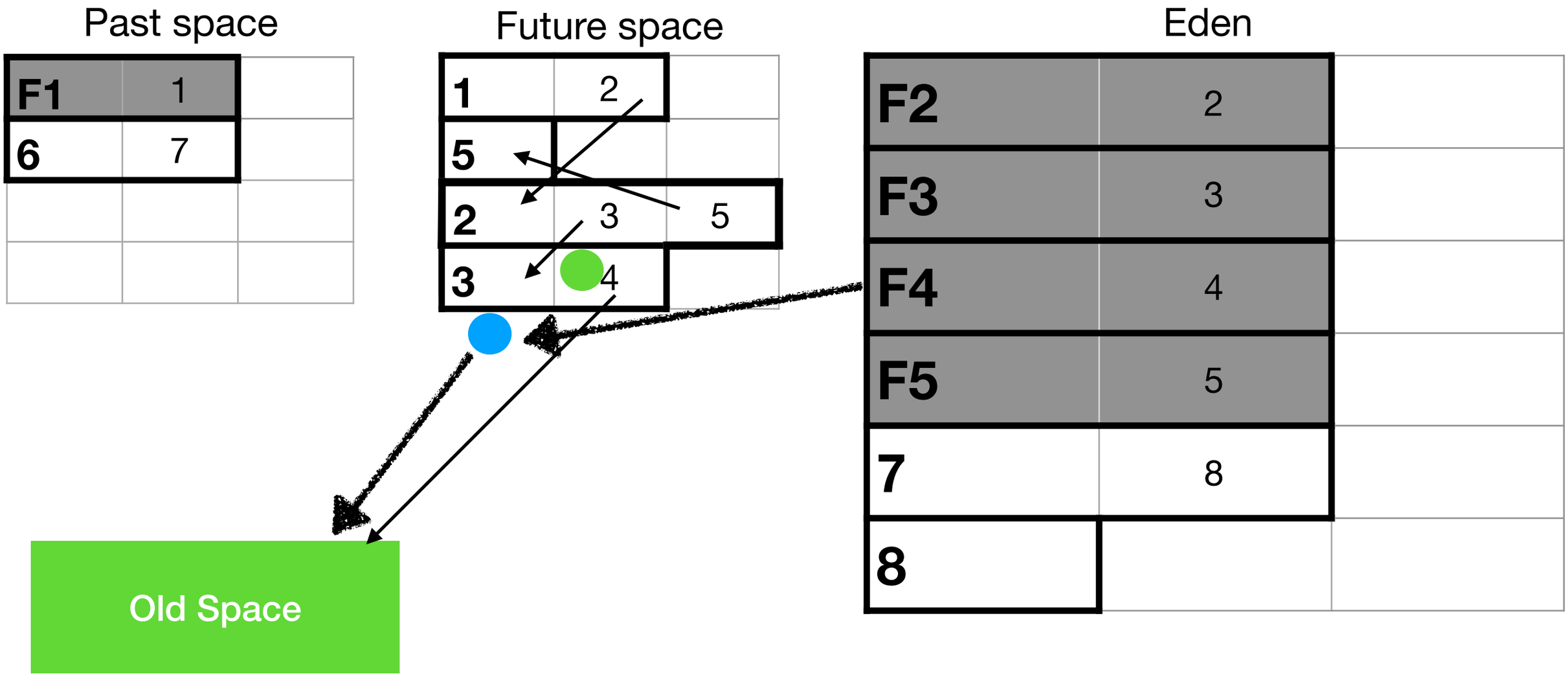
Copy 3 to the future



Copy 5 to the future
Already done !



Copy 4 to the future
Future is full !

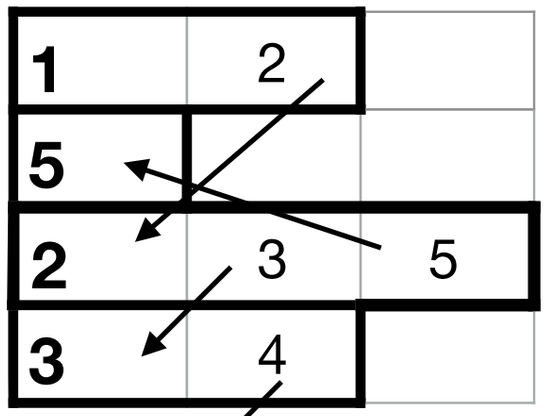


Switch the spaces

~~Past~~ Future space

F1	1	
6	7	

~~Future~~ Past space



(Empty) Eden

F2	2	
F3	3	
F4	4	
F5	5	
7	8	
8		

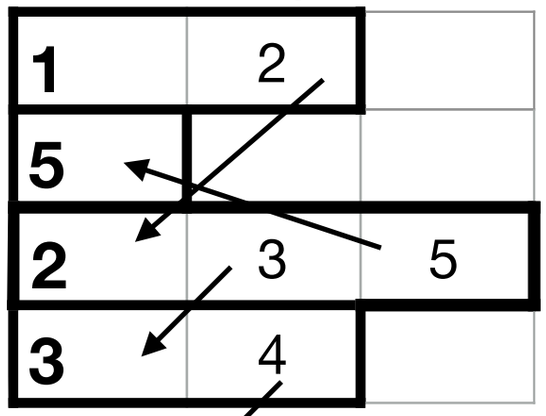
Old Space

Ready to resume working !

(Empty) Future space

F1	1	
6	7	

Past space

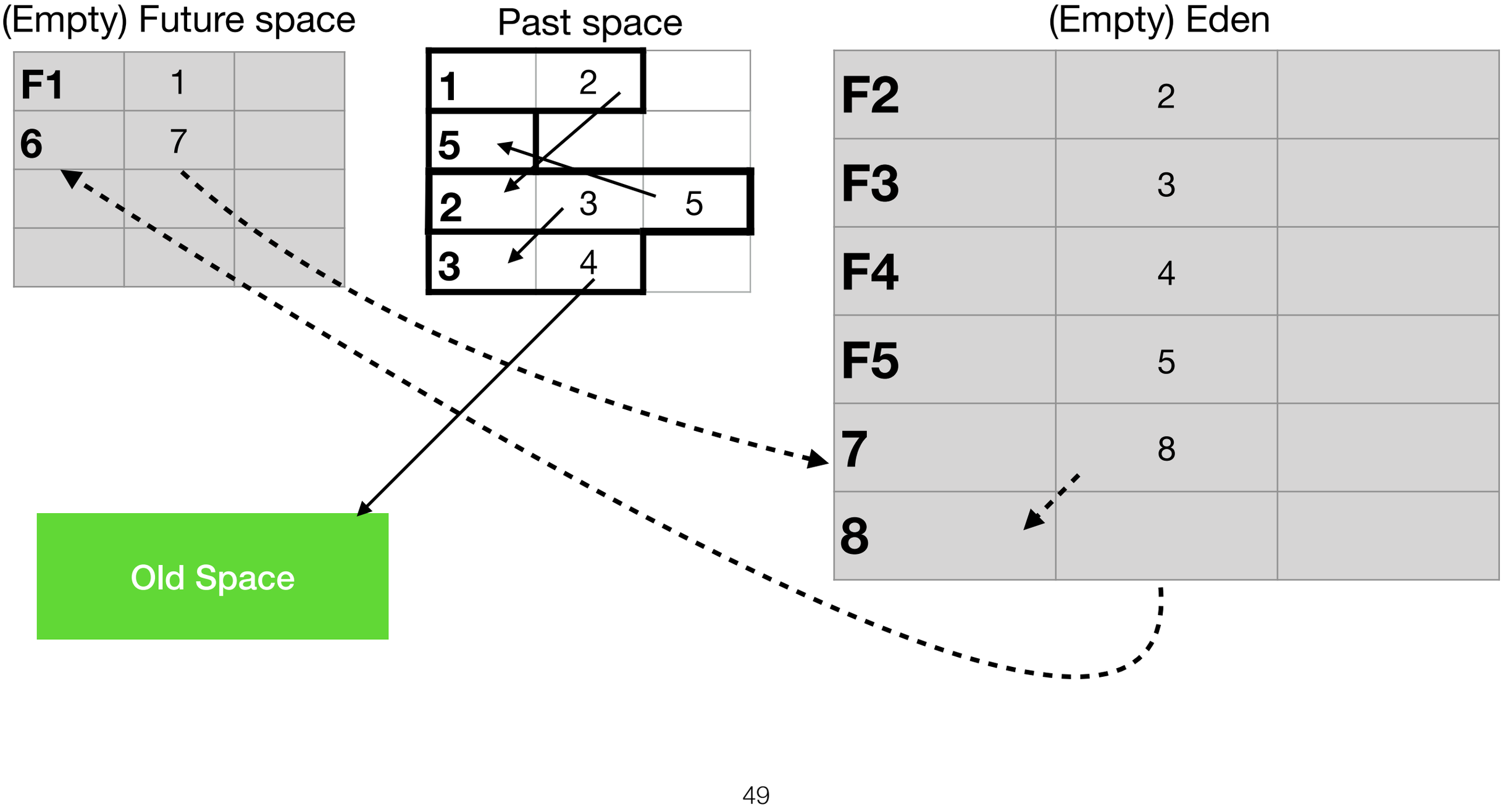


(Empty) Eden

F2	2	
F3	3	
F4	4	
F5	5	
7	8	
8		

Old Space

Reclaimed a cycle too!



Recap

- The Scavenger is a copy garbage collector
- Used for the new space
- New objects are allocated in the eden space
- Spaces are not cleaned, their memory is overridden
- The GC is triggered when the eden is full
- Forwarders are used to copy objects properly by the scavenger
- The scavenger first copies the objects in the roots
- Then it copies by breadth first order the objects to the future space
- If the future space is full, the objects are copied in the old space instead