

# Java Memory Management

Oleg Šelajev  
Java Fundamentals  
03.04.2017

# Agenda

- JVM memory
- Reference objects
- Monitoring
- Garbage collectors
  - ParallelGC
  - G1GC

# JVM memory

- Heap (user objects)
- Non-heap
  - Stack (per thread: call stack, local variables)
  - Metaspace (class metadata)
  - Direct Byte Buffers
  - Native stuff (JNI, Java internals)

| Method call    | Locals                               |
|----------------|--------------------------------------|
| printSubstring | s -> 0x1<br>ss -> 0x33<br>offset = 6 |
| main           | args -> 0x27<br>info -> 0x1          |

| Address | Value          |
|---------|----------------|
| 0x1     | "luke ..."     |
| 0x27    | String[0]      |
| 0x33    | "i'm your ..." |

```

public class Example {

    public static void main(String[] args) {
        String info = new String("luke, i'm your father");
        printSubstring(info, 6);
    }

    private static void printSubstring(String s, int offset) {
        String ss = s.substring(offset);
        System.out.println(ss);
    }
}

```

# Tuning options

- -Xms512M (initial heap size)
- -Xmx2G (max heap size)
- -Xss2M (max stack size, per thread)
- java [options] classname [args]
- All options at  
<https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html>  
<https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/>

# PermGen

- Java 8+: class metadata lives in Metaspace
- `-XX:MaxMetaspaceSize=size` (default: unlimited)
- Older versions: classes live in PermGen, PermGen is a special part of the heap
- `OutOfMemoryError: PermGen space`
- `-XX:MaxPermSize=size` (default: limited)

# Generating garbage

- Primitives in the stack, objects in the heap
- Using *new* allocates objects in the heap
- When and how are objects “deleted” and heap space freed up?

# Garbage collection (GC)

- Sorts garbage / live objects
- Reclaims heap space
- Fully automatic, no manual deallocation  
Java GC vs C++ new/delete
- Different GC algorithms exist



# GC advantages

- Avoiding bugs
  - forgetting to free the memory
  - double freeing memory
  - using already freed memory
- Java specific
  - No direct memory access
  - Can't accidentally overwrite unrelated memory

# GC disadvantages

- Consumes resources
- Automatic, no manual control
- Unpredictable stalls
- Harder to understand

# How does it work?

- Basic principle
  - Find referenced objects
  - Everything else is garbage
- Reachability (GC roots)
  - Classes loaded by system classloader (static fields!)
  - Stack locals (local variables, parameters)
  - Active threads
  - JNI References

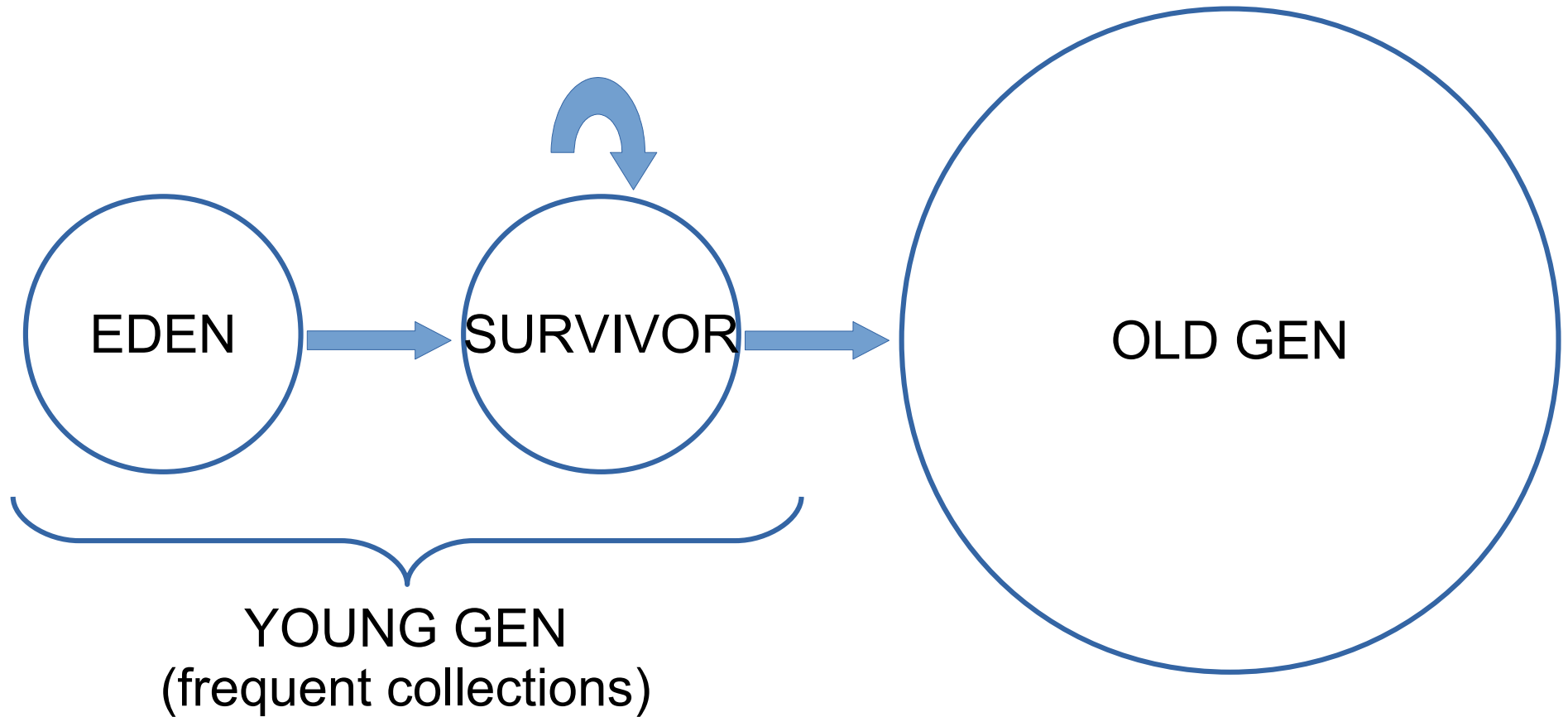
# Consumes resources?

- Extra memory + CPU for bookkeeping
- Stop The World pauses all threads
- Some applications need to tune GC:  
pause duration vs pause frequency

# Generational GC

- Most objects die young
- Generations: memory pools holding objects of different ages
  - Young generation: eden, survivors
  - Old/tenured generation
- Young-Old default size ratio 1:2

# Young/Old



# GC algorithms

- Serial
- **Parallel**
- Concurrent Mark-Sweep
- **Garbage first (G1)**
- IBM, Azul special stuff

# Agenda

- JVM memory
- **Reference objects**
- Monitoring
- Garbage collectors
  - ParallelGC
  - G1GC



# Reference objects

- `java.lang.ref` package docs are useful
- `WeakReference<T>`
- `PhantomReference<T>`
- `ReferenceQueue<T>`

# Detour: memory leaks

```
interface Passenger {
    void trainArrived();
}

class TrainStation {
    private final List<Passenger> passengers = new...
    public void startWaiting(Passenger passenger) {
        passengers.add(passenger);
    }
    public void leave(Passenger passenger) {
        passengers.remove(passenger);
    }
    public void onTrainArrived() {
        passengers.forEach(Passenger::trainArrived);
    }
}
```

# Detour: memory leaks

```
interface Passenger {
    void trainArrived();
}

class TrainStation {
    private final List<Passenger> passengers = new...
    public void startWaiting(Passenger passenger) {
        passengers.add(passenger);
    }
    public void leave(Passenger passenger) {
        passengers.remove(passenger);
    }
    public void onTrainArrived() {
        passengers.forEach(Passenger::trainArrived);
    }
}
```

# WeakReference<T>

Keep a reference without preventing GC

```
private final WeakReference<SomethingBig> weakRef;  
  
public Example(SomethingBig sb) {  
    this.weakRef = new WeakReference<>(sb);  
}  
  
private void tryPrint() {  
    SomethingBig strongRef = weakRef.get();  
    System.out.println(strongRef != null  
        ? strongRef  
        : "collected");  
}
```

# Make it foolproof

```
interface Passenger {
    void trainArrived();
}

class TrainStation {
    private final List<Passenger> passengers = new...
    public void startWaiting(Passenger passenger) {
        passengers.add(passenger);
    }
    public void leave(Passenger passenger) {
        passengers.remove(passenger);
    }
    public void onTrainArrived() {
        passengers.forEach(Passenger::trainArrived);
    }
}
```

# Weaker TrainStation

```
class TrainStation {
    private List<WeakReference<Passenger>> passengers;

    public void startWaiting(Passenger passenger) {
        passengers.add(new WeakReference<>(passenger));
    }

    public void onTrainArrived() {
        for (WeakReference<Passenger> ref : passengers) {
            Passenger passenger = ref.get();
            if (passenger != null)
                passenger.trainArrived();
        }
    }
}
```

# Detour: finalizers

From `java.lang.Object` JavaDoc

- protected void `finalize()`

Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.

- Safety net for file streams, network sockets, JDBC connections, etc.

# Detour: finalizers

From “Effective Java” by Joshua Bloch

- Finalizers are unpredictable, often dangerous, and generally unnecessary.
- Not only does the language specification provide no guarantee that finalizers will get executed promptly; it provides no guarantee that they’ll get executed at all.



# Detour: finalizers

Trolling the garbage collector:

```
public class Test {  
    static Test t;  
    @Override  
    public void finalize() {  
        t = this; // I refuse to die  
    }  
}
```



# PhantomReference<T>

- Not a reference, but a GC token
- Only usable with a ReferenceQueue
- Enqueued by the garbage collector, only after referent is collected
- `get()` -> null always!

# PhantomReference<T>

```
Example e = new Example();
ReferenceQueue<Example> queue =
    new ReferenceQueue<>();
PhantomReference<Example> phantom =
    new PhantomReference<>(e, queue);
e = null;
```

```
// generate garbage, cause a GC
Reference<?> collected =
queue.remove();
if (collected == phantom) {
    // our e has been collected
}
```

# Agenda

- JVM memory
- Reference objects
- **Monitoring**
- Garbage collectors
  - ParallelGC
  - G1GC

# GC logging

- `-XX:+PrintGCTimeStamps`
- `-XX:+PrintGCDetails`
- `-Xloggc:filename`
- Output depends heavily on GC algo
- Read the fine manual:  
[plumbr.eu/java-garbage-collection-handbook](http://plumbr.eu/java-garbage-collection-handbook)  
“GC Algorithms: Implementations”

# ParallelGC minor

2015-05-26T14:27:40.915-0200: 116.115:

[GC (Allocation Failure)

[PSYoungGen: 2 694 440K -> 1 305 132K (2 796  
544K)]

9 556 775K -> 8 438 926K (11 185 152K), 0.24066 secs  
]

[Times: user=1.77 sys=0.01, real=0.24 secs]

# ParallelGC full

2015-05-26T14:27:41.155-0200: 116.356:

[Full GC (Ergonomics)

[PSYoungGen: 1 305 132K -> 0K(2 796 544K)]

[ParOldGen: 7 133 794K -> 6 597 672K (8 388 608K)]

8 438 926K -> 6 597 672K (11 185 152K),

[Metaspace: 6 745K -> 6 745K (1 056 768K)] ,

0.91588 secs

]

[Times: user=4.49 sys=0.64, real=0.92 secs]

# G1 minor

```
0.134: [GC pause (G1 Evacuation Pause)
(young), 0.0144119 secs] ... [
  Eden: 24.0M (24.0M) -> 0.0B (13.0M)
  Survivors: 0.0B -> 3072.0K
  Heap: 24.0M (256.0M) -> 21.9M (256.0M)
]
[Times: user=0.04 sys=0.04, real=0.02 secs]
```



# GC overhead

116.356: [Full GC ...

117.331: [Full GC ...

118.378: [Full GC ...

119.316: [Full GC ...

java.lang.OutOfMemoryError:  
GC overhead limit exceeded

Frequent+quick minor collections expected

# Heap dumps

- JVisualVM
- Eclipse memory analyzer (MAT)
- `jmap -heap / -histo / -dump:... <pid>`
- `-XX:+HeapDumpOnOutOfMemory`  
`-XX:HeapDumpPath=path`

# jmap

```
$ jmap -heap 1244
```

Heap Usage:

PS Young Generation  
capacity = (930.0MB)  
used = (595.2MB)  
free = (334.7MB)  
64.0% used

PS Old Generation

capacity = (167.0MB)  
used = (2.9MB)  
free = (164.0MB)  
1.7% used

# jmap

```
$ jmap -histo 1244
```

| num | #instances | #bytes    | class name          |
|-----|------------|-----------|---------------------|
| 1:  | 250080     | 217038824 | [C                  |
| 2:  | 874        | 11646608  | [I                  |
| 3:  | 250040     | 6000960   | java.lang.String    |
| 4:  | 577        | 85968     | [Ljava.lang.Object; |
| 5:  | 582        | 66144     | java.lang.Class     |
| 6:  | 22         | 25312     | [B                  |
| 7:  | 109        | 7848      | j.l.r.Field         |

# jps + jmap

```
$ jps -lv
```

```
30086 com.intellij.idea.Main ...
```

```
1738 sun.tools.jps.Jps ...
```

```
1659 org.jetbrains.jps.cmdline.Launcher ...
```

```
1660
```

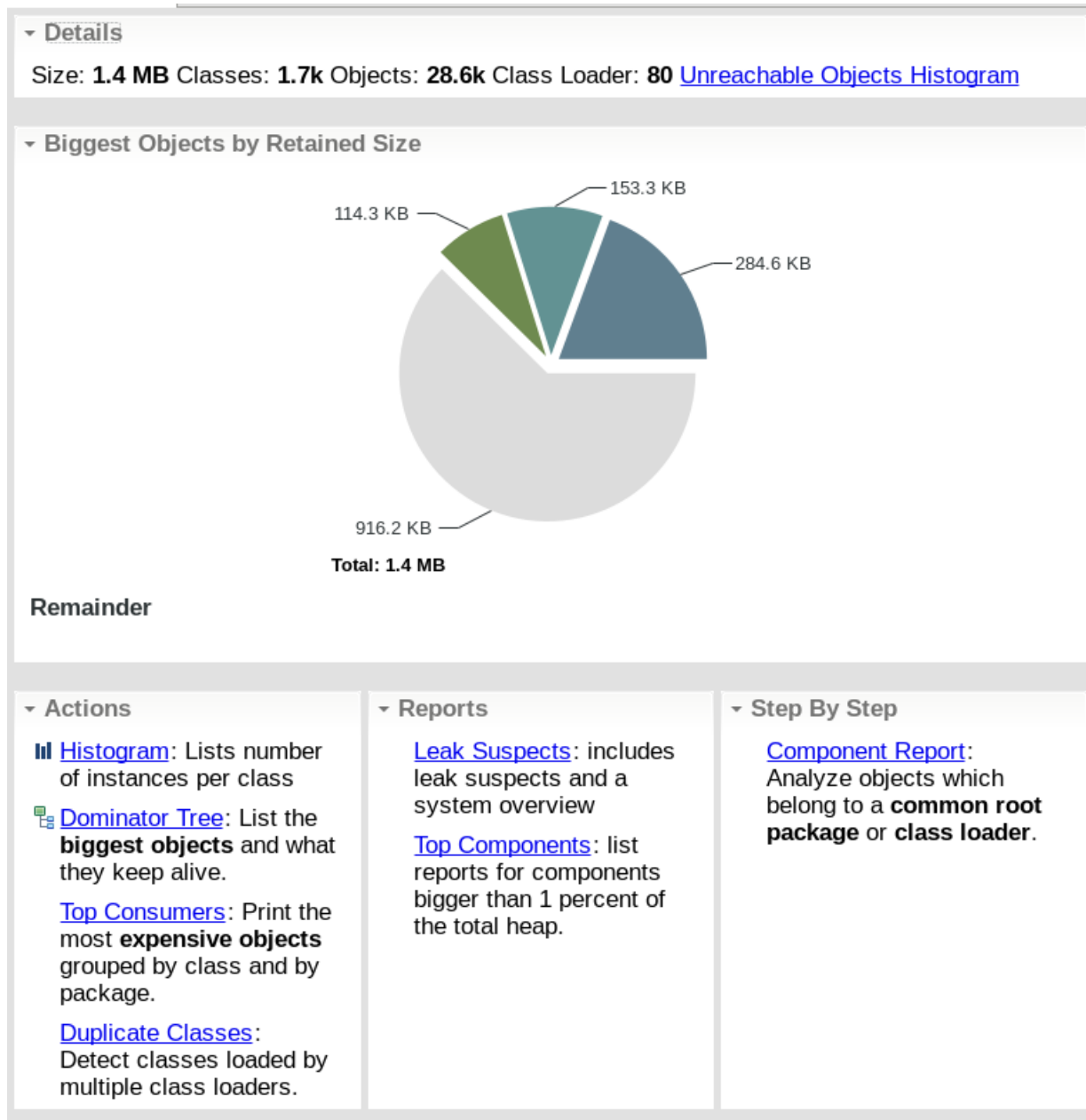
```
com.intellij.rt.execution.application.AppMain ...
```

```
$ jmap -dump:format=b,file=dump.bin 1660
```

```
Dumping heap to /tmp/dump.bin ...
```

```
Heap dump file created
```

# Eclipse memory analyzer



# Eclipse memory analyzer

Inspector

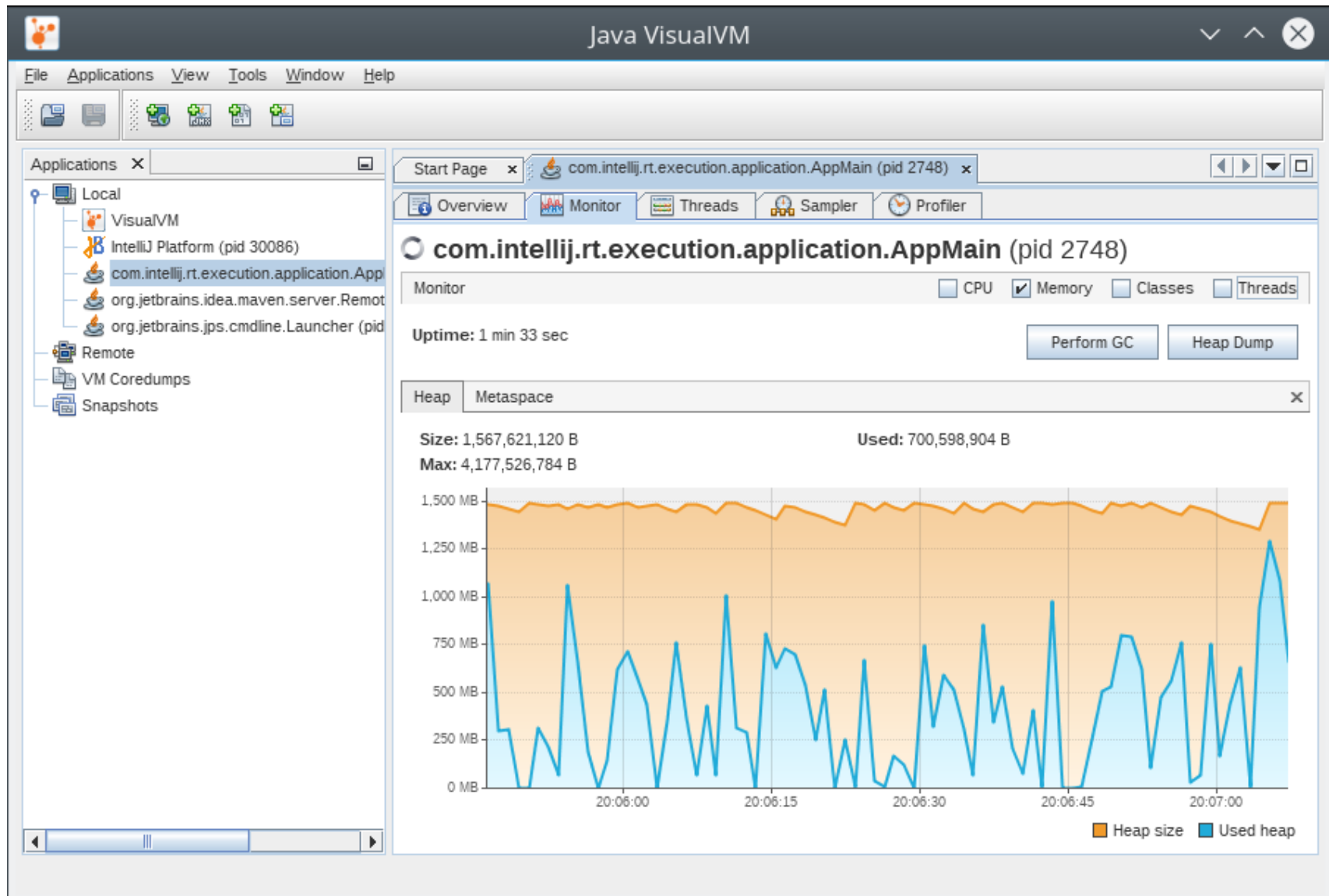
@ 0x6c701a020

- Object[]
- java.lang
- class java.lang.Object[] @ 0x6c701a020
- java.lang.Object
- java.lang.ClassLoader @ 0x0
- 56,232 (shallow size)
- 290,368 (retained size)
- no GC root

| Class Name  | Shallow   | Retained  | Retained Heap |
|---|-----------|-----------|---------------|
| <Regex>   | <Numeric> | <Numeric> | <Numeric>     |
| java.lang.Thread @ 0x6c70129f8 main Thread                |           |           |               |
| <Java Local> java.util.ArrayList @ 0x6c7012870            |           |           |               |
| <b>elementData java.lang.Object[14053] @ 0x6c701a020</b>  |           |           |               |
| [285] java.lang.String @ 0x6c7009c48 Non qui distinctio   |           |           |               |
| [283] java.lang.String @ 0x779cf30c0 n qui distinctio lil |           |           |               |
| [284] java.lang.String @ 0x779cf34b0 on qui distinctio    |           |           |               |
| [279] java.lang.String @ 0x779cf2120 i distinctio libero  |           |           |               |
| [280] java.lang.String @ 0x779cf2508 ui distinctio liber  |           |           |               |
| [281] java.lang.String @ 0x779cf28f0 qui distinctio liber |           |           |               |
| [282] java.lang.String @ 0x779cf2cd8 qui distinctio lib   |           |           |               |
| [275] java.lang.String @ 0x779cf11a0 stinctio libero es   |           |           |               |
| [276] java.lang.String @ 0x779cf1580 istinctio libero es  |           |           |               |
| [277] java.lang.String @ 0x779cf1960 distinctio libero e  |           |           |               |
| [278] java.lang.String @ 0x779cf1d40 distinctio libero    |           |           |               |
| [271] java.lang.String @ 0x779cf0240 ctio libero est qu   |           |           |               |
| [272] java.lang.String @ 0x779cf0618 nctio libero est q   |           |           |               |
| [273] java.lang.String @ 0x779cf09f0 inctio libero est q  |           |           |               |

| Type | Name | Value            |
|------|------|------------------|
| ref  | [0]  | uo.Non qui disti |
| ref  | [1]  | quo.Non qui dis  |
| ref  | [2]  | quo.Non qui di   |
| ref  | [3]  | i quo.Non qui di |
| ref  | [4]  | ui quo.Non qui e |
| ref  | [5]  | qui quo.Non qui  |

# JVisualVM





# JVisualVM

The screenshot displays the JVisualVM application window. The title bar reads "Java VisualVM". The menu bar includes "File", "Applications", "View", "Tools", "Window", and "Help". The main window shows a tab for "com.intellij.rt.execution.application.AppMain (pid 3190)". Below the tabs, there are buttons for "Overview", "Monitor", "Threads", "Sampler", "Profiler", and "[heapdump] 20:14:08".

The central area displays the "Heap Dump" for "com.intellij.rt.execution.application.AppMain (pid 3190)". It includes navigation buttons for "Summary", "Classes", "Instances", and "OQL Console". The selected class is "java.lang.String", with statistics: "Instances: 10,430 | Instance size: 28 | Total size: 292,040 | [Compute Retained Sizes](#)".

Below the statistics, there are two main panels:

- Instances:** A list of instances for the selected class. The first instance is highlighted: "#10001 cusamus ut suscipit et recusandae".
- Fields:** A table showing the fields of the selected instance.

| Field                  | Type                              | Value                                    |
|------------------------|-----------------------------------|--|
| this                   | String                            | #10001 cusamus ut suscipit et recusandae |
| hash                   | int                               | 0  |
| value                  | char[]                            | #10048 cusamus ut suscipit et recusandae |
| CASE_INSENSITIVE_ORDER | String\$CaseInsensitiveComparator | #1                                       |
| serialPersistentFields | ObjectStreamField[]               | #11 0 items                              |
| serialVersionUID       | long                              | -6849794470754667710                     |
| <classLoader>          | <object>                          | null                                     |

Below the Fields panel is the "References" panel, which shows references to the selected instance:

| Field                  | Type      | Value                                    |
|------------------------|-----------|--|
| this                   | String    | #10001 cusamus ut suscipit et recusandae |
| [430]                  | Object[]  | #7 14,053 items                          |
| elementData            | ArrayList | #3                                       |
| myGarbage (Java frame) | Example   | class Example                            |

At the bottom of the window, there is a legend for field types: "Array type" (blue square), "Object type" (red circle), "Primitive type" (blue square), "Static field" (red triangle), "GC Root" (blue square), and "Loop" (green circle).

# JVisualVM

- Bundled with JDK
- Windows:  
C:\Program Files\Java\jdk1.8.x\bin\jvisualvm.exe
- Linux:  
jvisualvm or visualvm  
(apt-get install visualvm)
- Compile & Run: <https://goo.gl/tGO4p8>

# STW

IT'S LIKE THERE'S  
CONTINUOUS  
PRESSURE ON ME  
TO WORK.



# Agenda

- JVM memory
- Reference objects
- Monitoring
- **Garbage collectors**
  - ParallelGC
  - G1GC

# ParallelGC

- GC roots
  - static fields
  - stack locals
  - threads
- Young gen: eden, survivor to/from
- Old gen
- Stop The World pauses

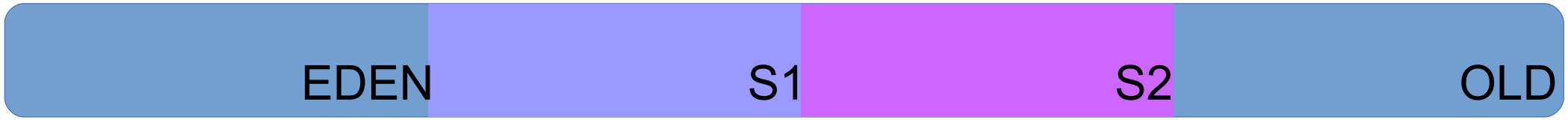
GC ROOTS   **STACK**   **STATIC FIELDS**

**RUNNING/  
STOP THE WORLD**





**RUNNING**

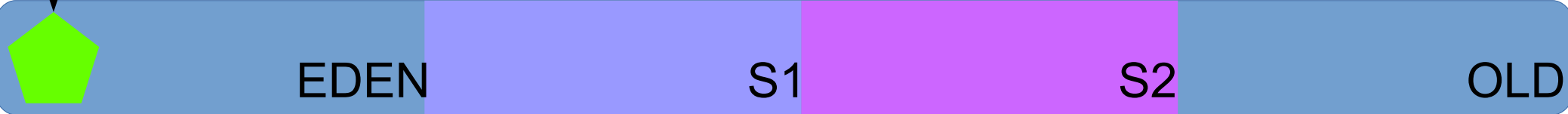
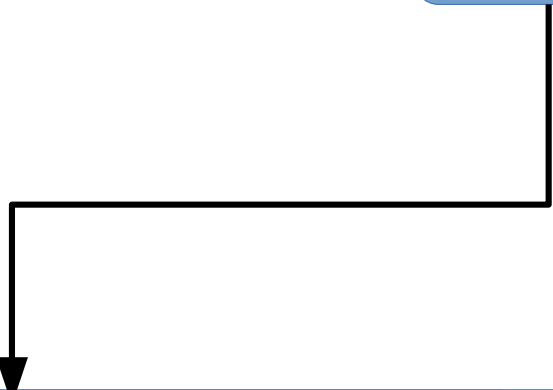


**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

S1

S2

OLD

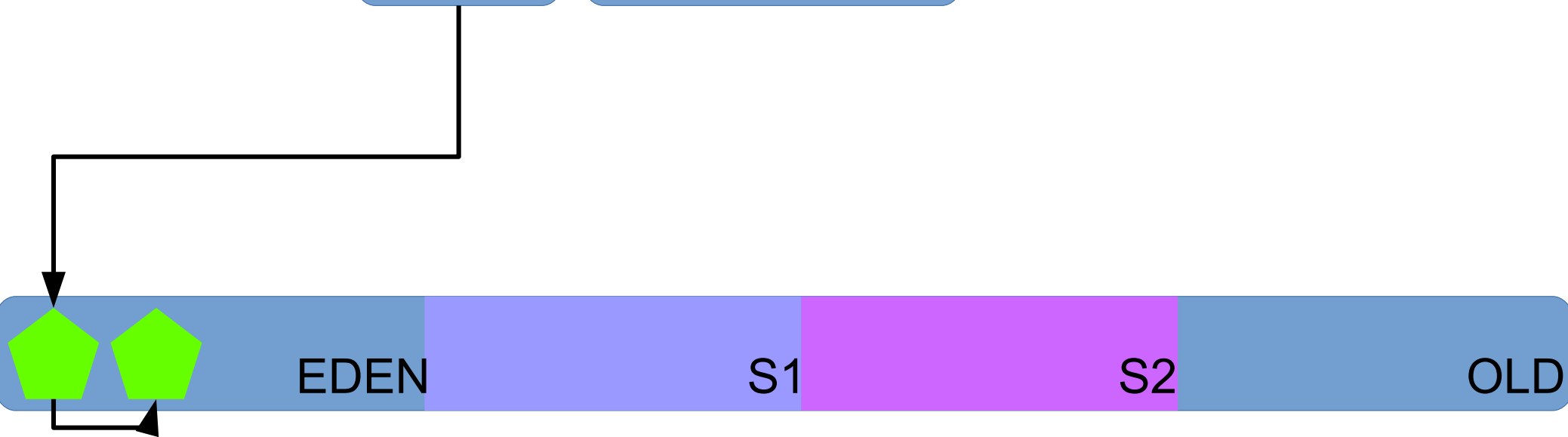


**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

S1

S2

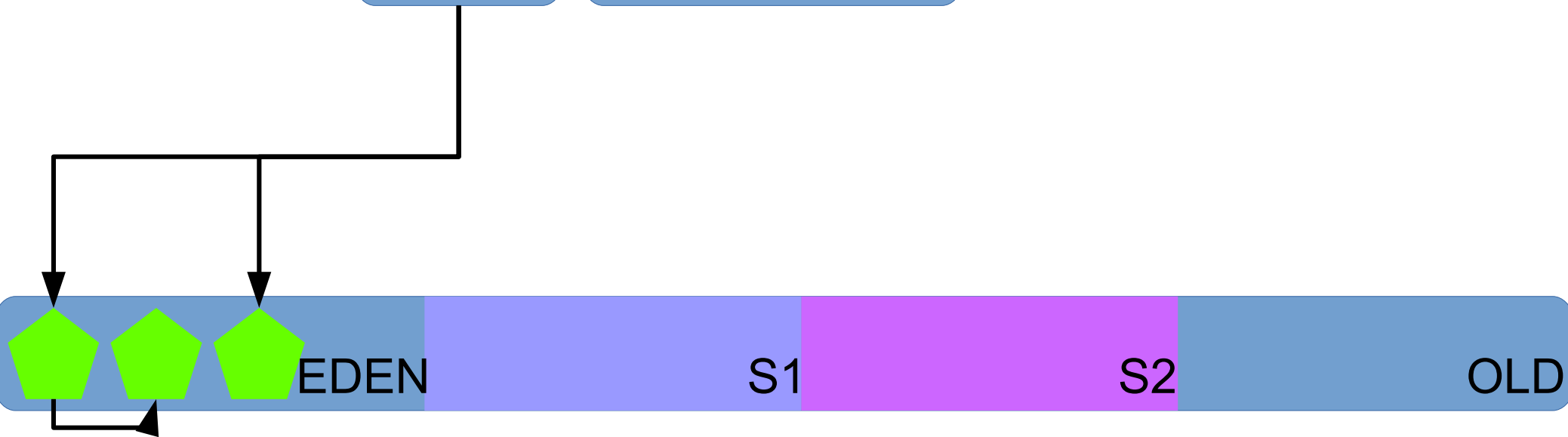
OLD

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

S1

S2

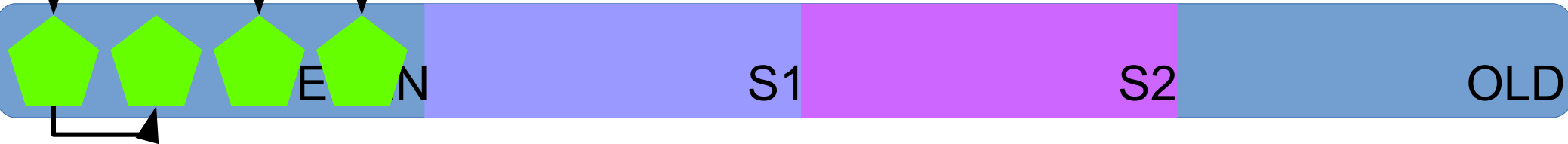
OLD

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS

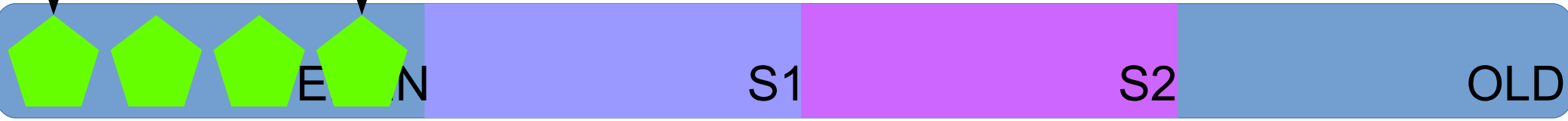


**RUNNING**

GC ROOTS

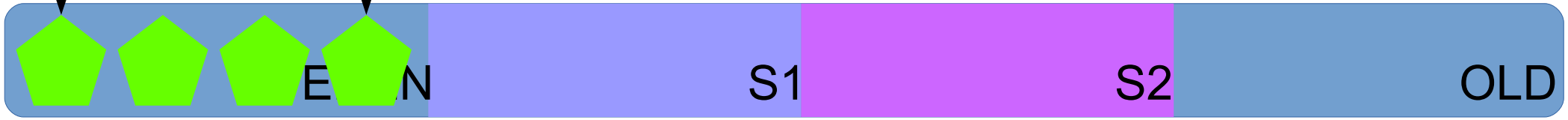
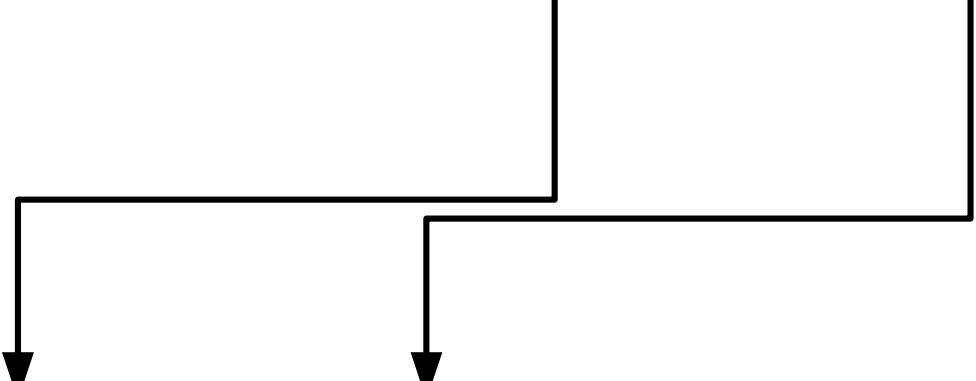
STACK

STATIC FIELDS



**STOP THE WORLD**

GC ROOTS



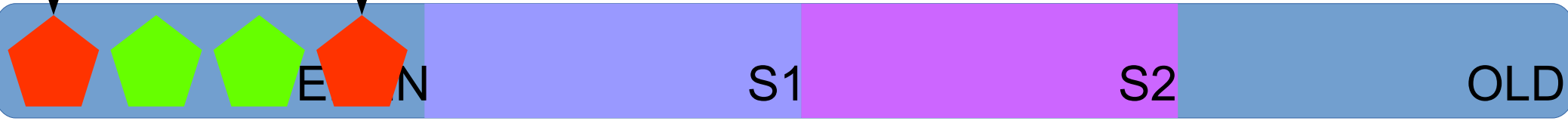
2015-05-26T14:27:40.915-0200: 116.115:  
[GC (Allocation Failure) ... ]

**STOP THE WORLD**

GC ROOTS

STACK

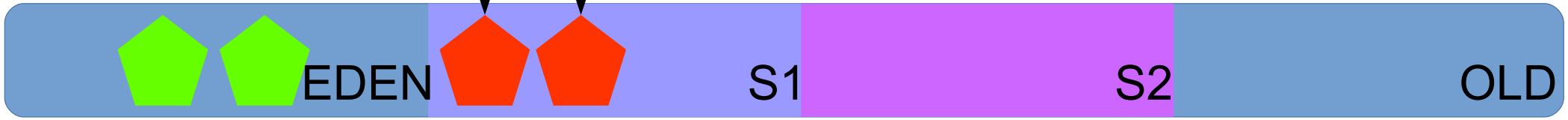
STATIC FIELDS



Find live objects, starting from GC roots (mark)

**STOP THE WORLD**

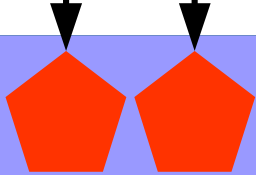
GC ROOTS    **STACK**    **STATIC FIELDS**



Move live objects to survivors (compacting)

**STOP THE WORLD**

GC ROOTS



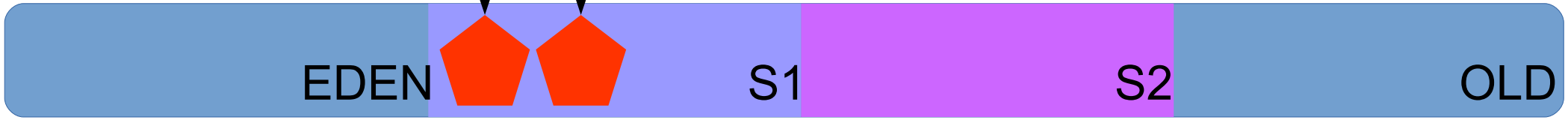
EDEN

S1

S2

OLD

Mark EDEN as clean





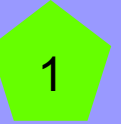
**RUNNING**

GC ROOTS

STACK

STATIC FIELDS

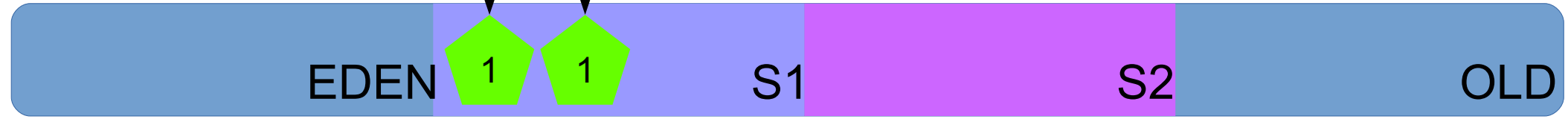
EDEN



S1

S2

OLD

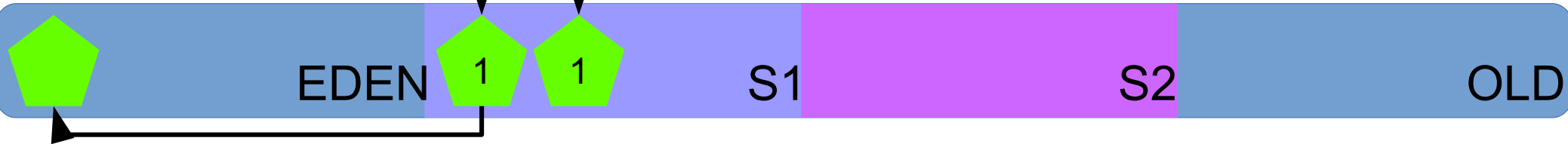


**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

1

1

S1

S2

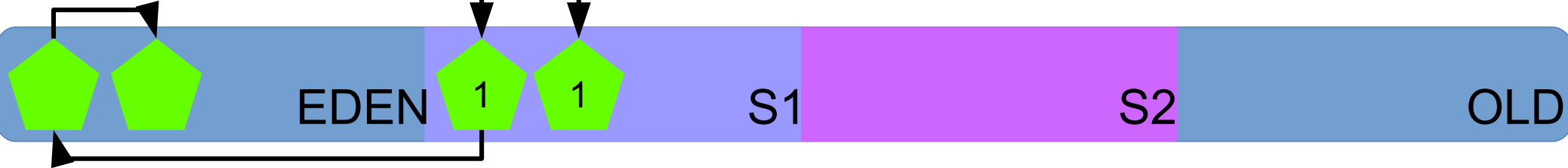
OLD

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

1

1

S1

S2

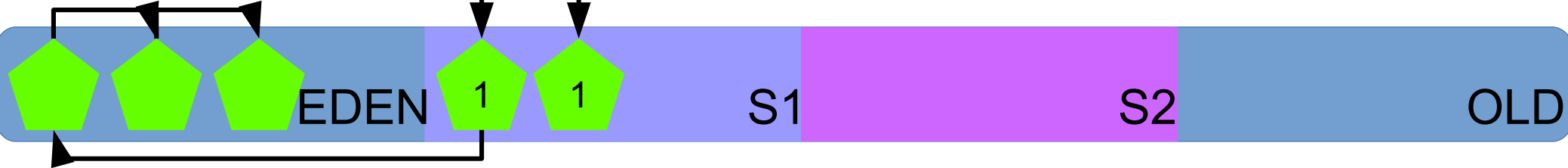
OLD

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

1

1

S1

S2

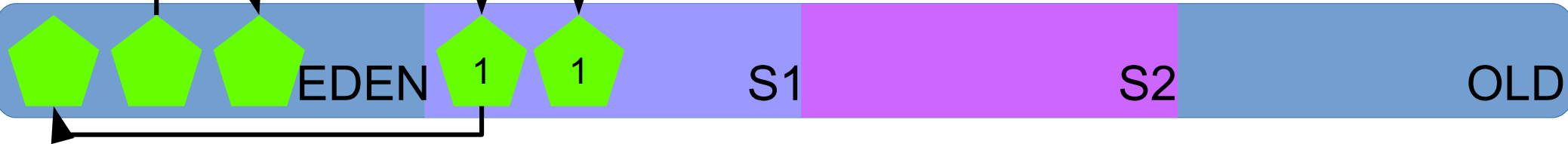
OLD

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

1

1

S1

S2

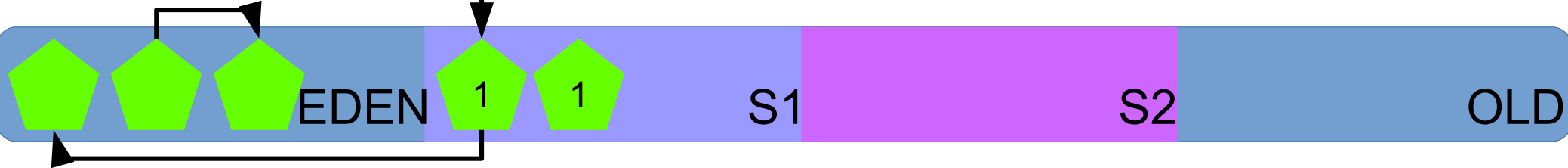
OLD

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

1

1

S1

S2

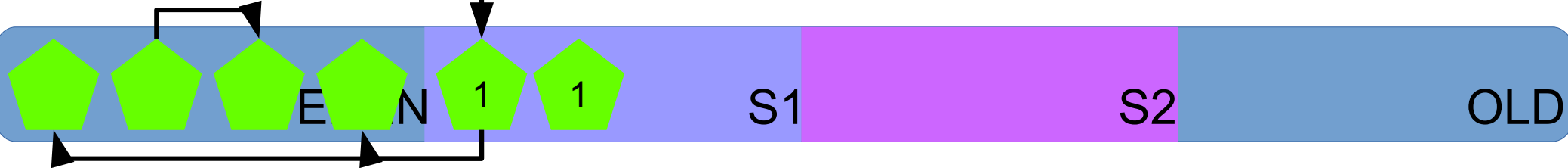
OLD

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS

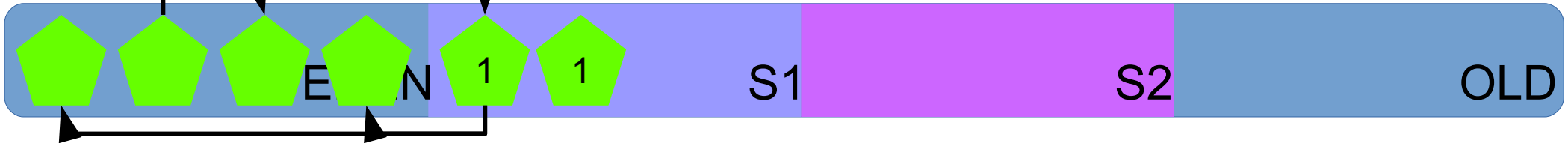


**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS

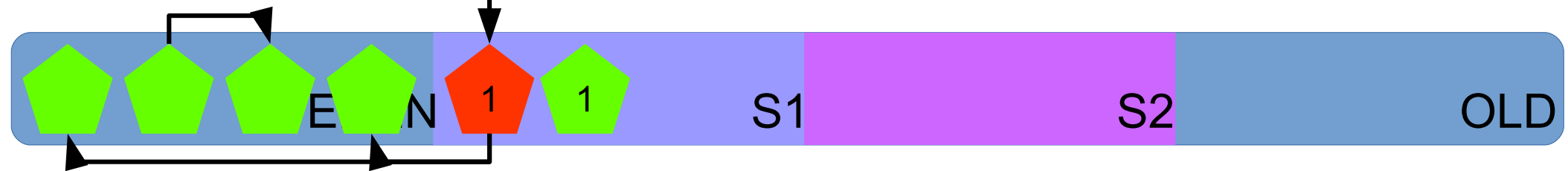


2015-05-26T14:27:41.915-0200: 117.115:  
[GC (Allocation Failure) ... ]



**STOP THE WORLD**

GC ROOTS



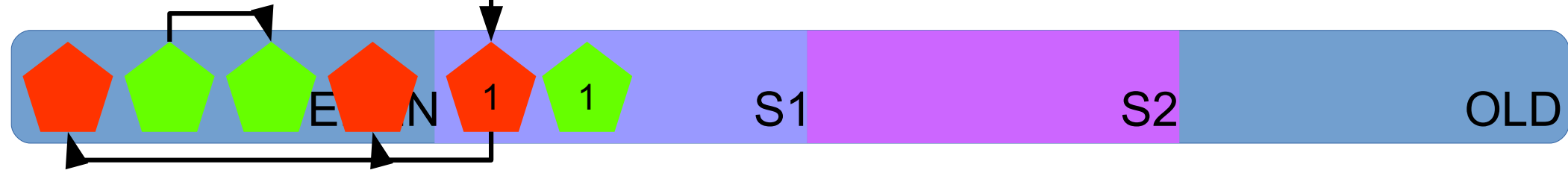
Find live objects, starting from GC roots (mark)

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



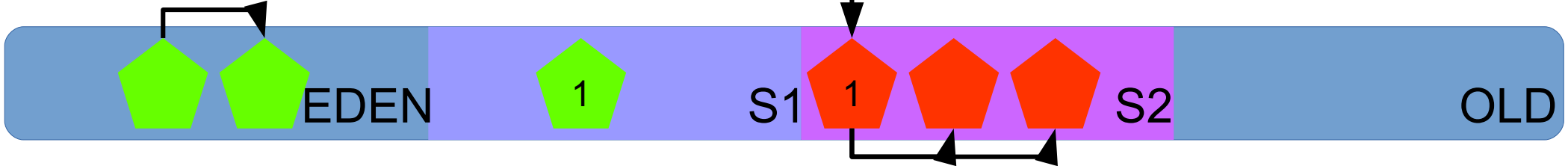
Find live objects, starting from GC roots (mark)

**STOP THE WORLD**

GC ROOTS

STACK

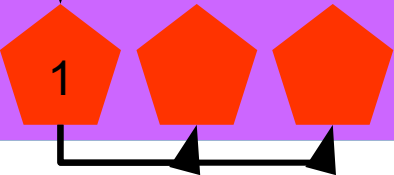
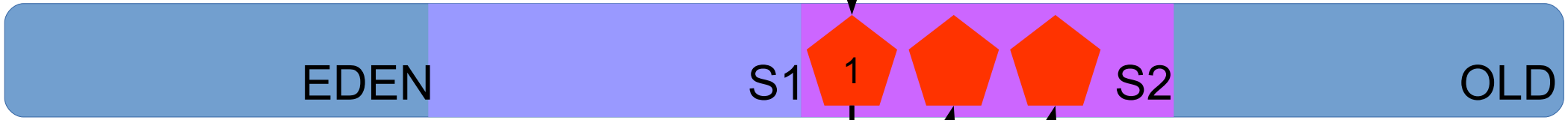
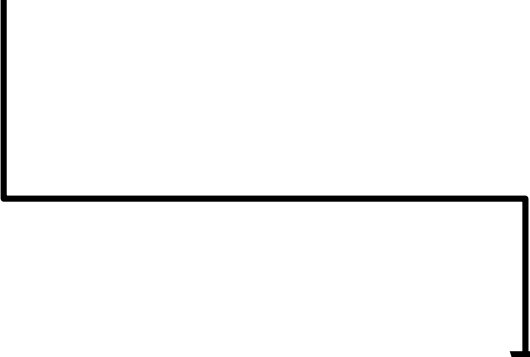
STATIC FIELDS



Move live objects to survivors (compacting)

**STOP THE WORLD**

GC ROOTS



Mark EDEN+S1 as clean  
S1/S2 compaction

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS

EDEN

S1

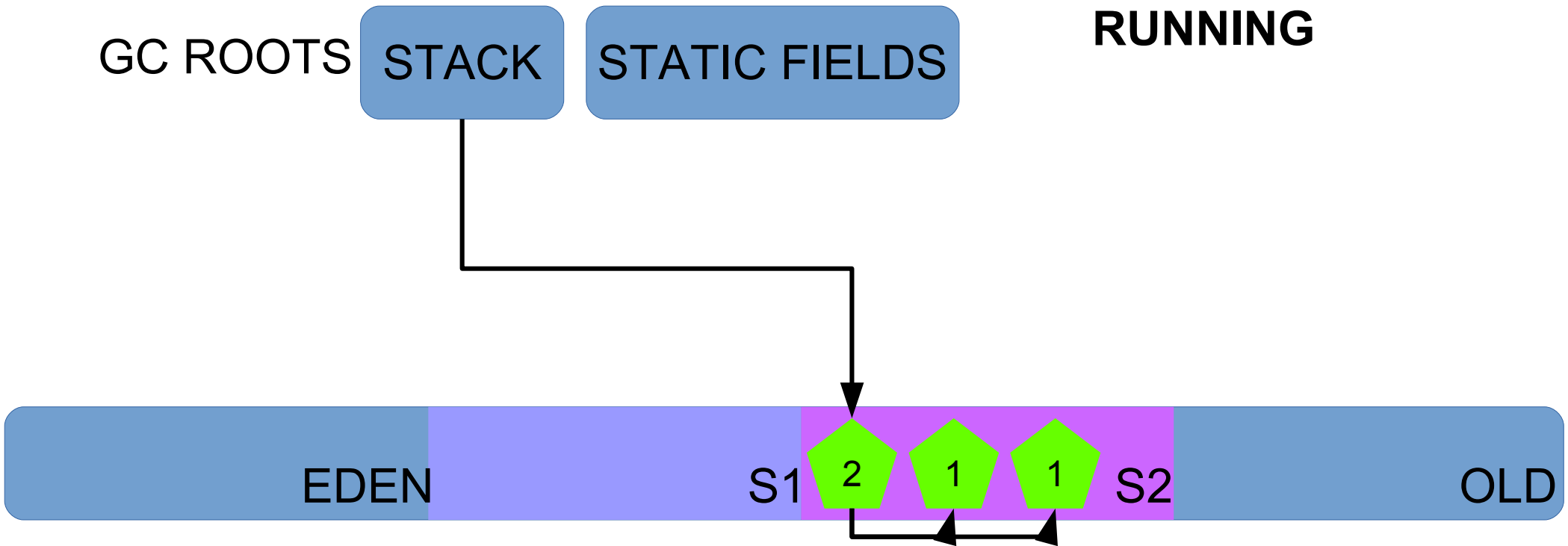
2

1

1

S2

OLD

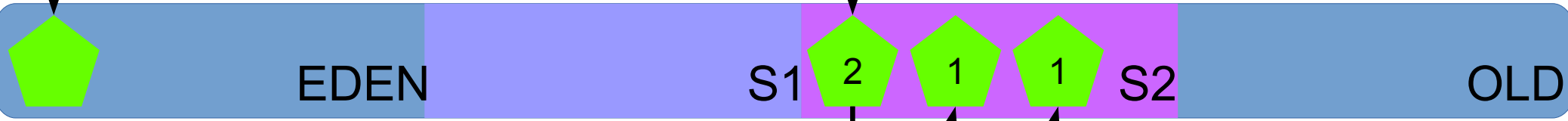
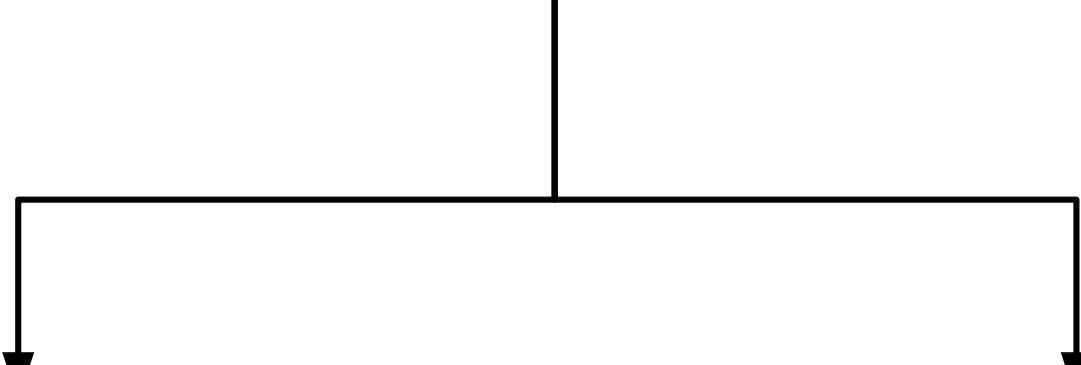


**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

S1

2

1

1

S2

OLD

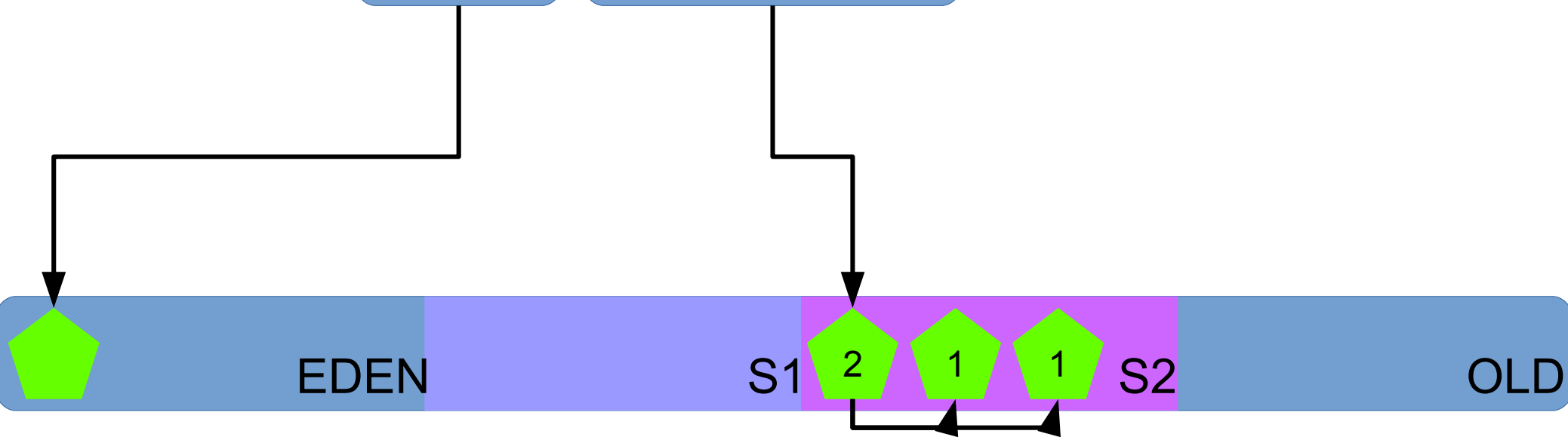


**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

S1

2

1

1

S2

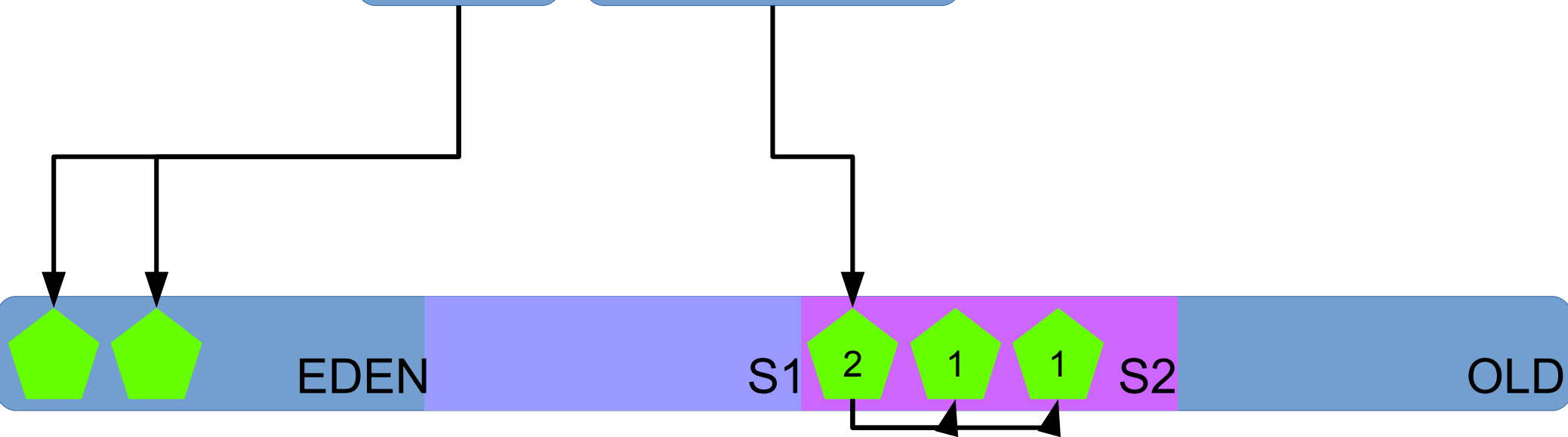
OLD

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



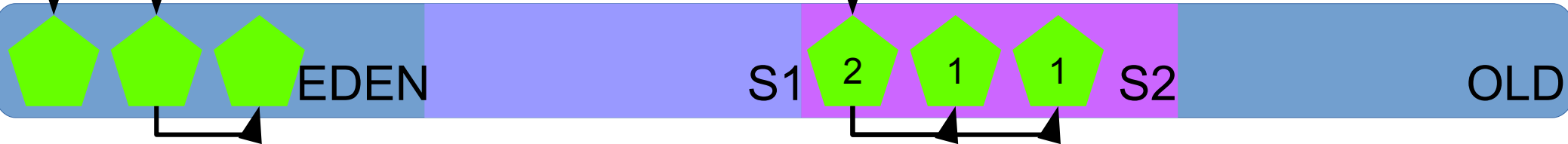


**RUNNING**

GC ROOTS

STACK

STATIC FIELDS

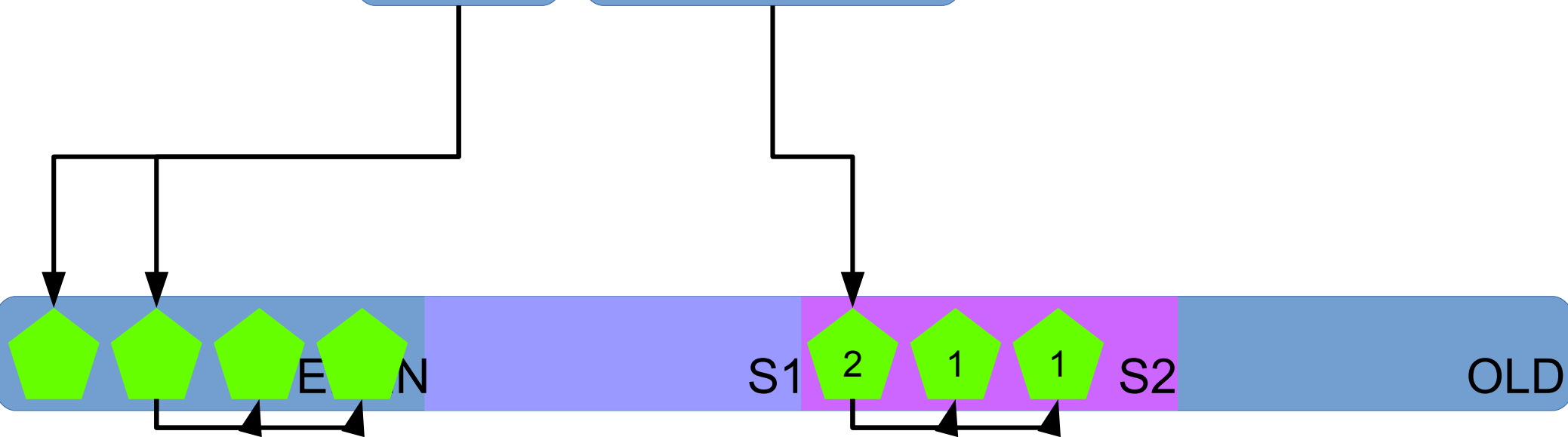


**RUNNING**

GC ROOTS

STACK

STATIC FIELDS

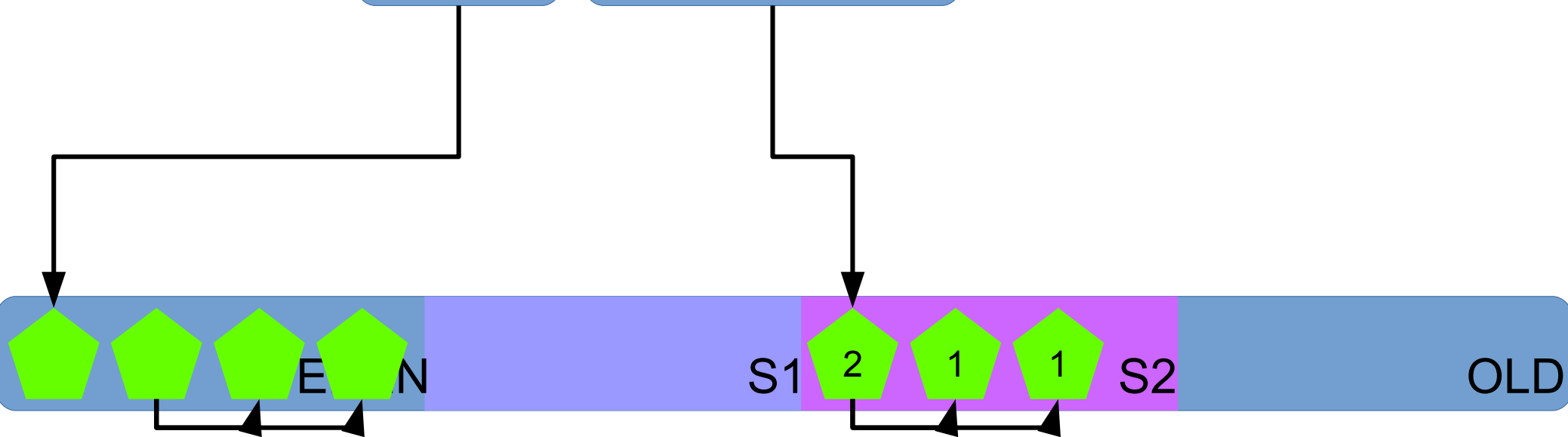


**RUNNING**

GC ROOTS

STACK

STATIC FIELDS

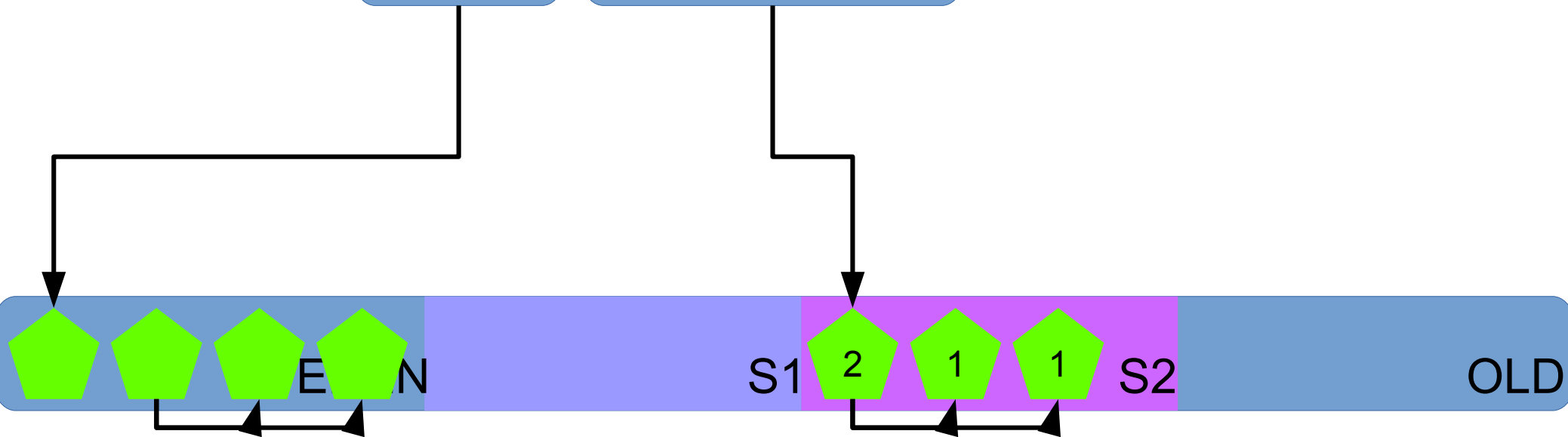


**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



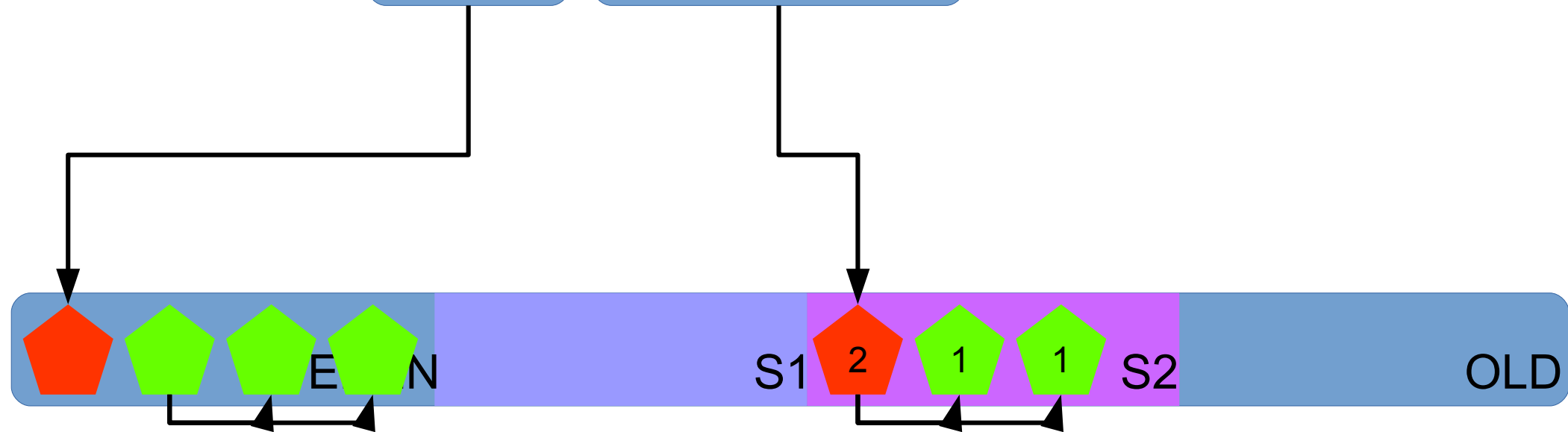
2015-05-26T14:27:43.915-0200: 119.115:  
[GC (Allocation Failure) ... ]

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



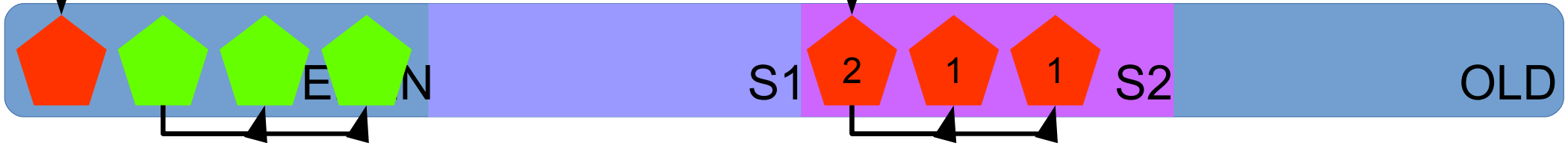
Find live objects, starting from GC roots (mark)

**STOP THE WORLD**

GC ROOTS

STACK

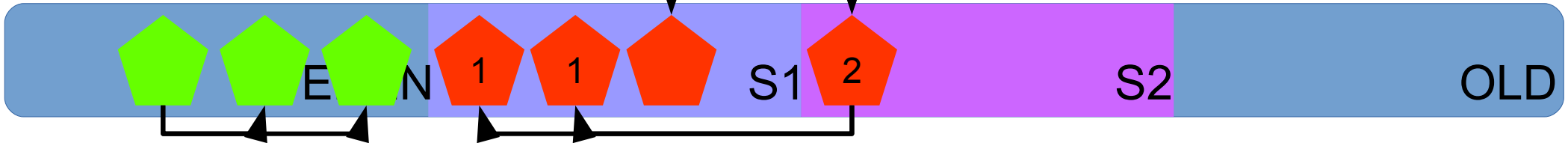
STATIC FIELDS



Find live objects, starting from GC roots (mark)

**STOP THE WORLD**

GC ROOTS    **STACK**    **STATIC FIELDS**



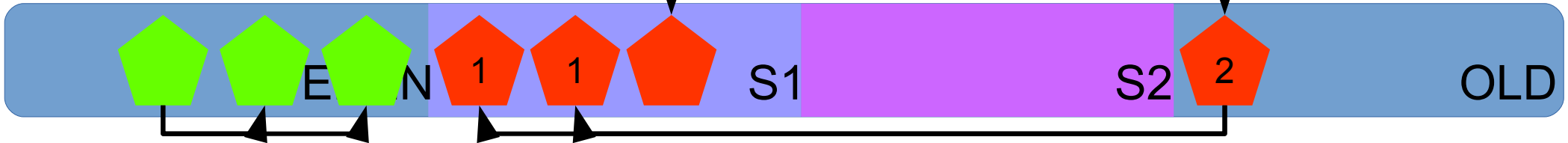
Move live objects to survivors (compacting)

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



Move live objects to survivors or old (compacting)

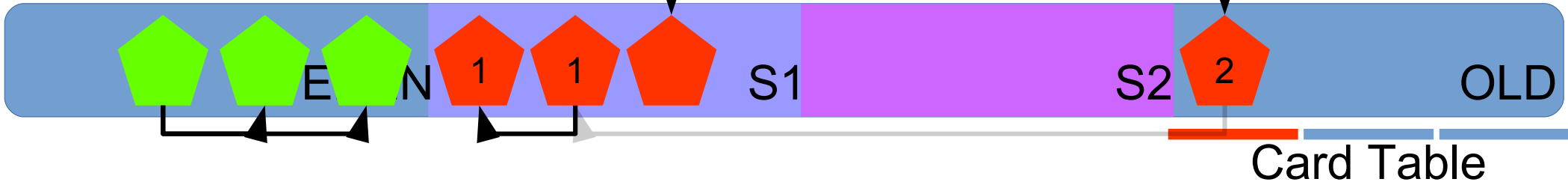


**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



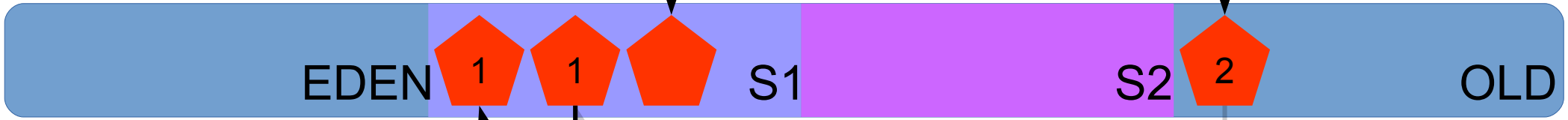
Old Gen has a “Card Table” (card region ~2M)  
Identifies regions that reference Young Gen

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



EDEN

1 1

S1

S2

2

OLD

Card Table

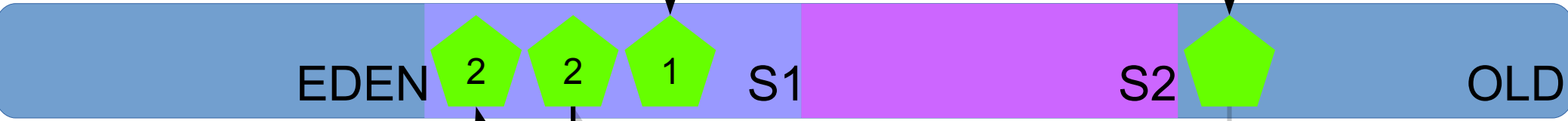
Mark EDEN+S2 as clean

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

2

2

1

S1

S2

OLD

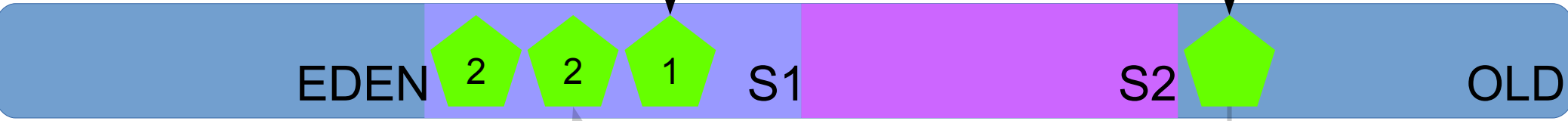
Card Table

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

2

2

1

S1

S2

2

OLD

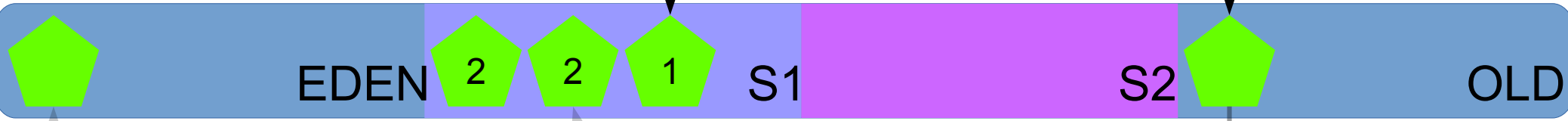
Card Table

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

2

2

1

S1

S2

OLD

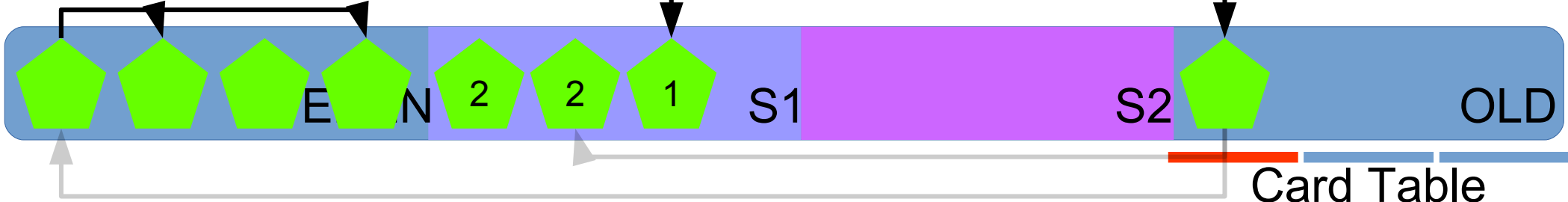
Card Table

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS

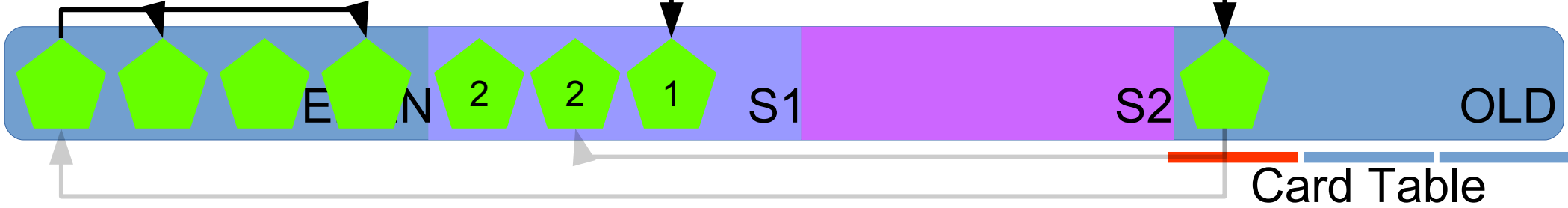


**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



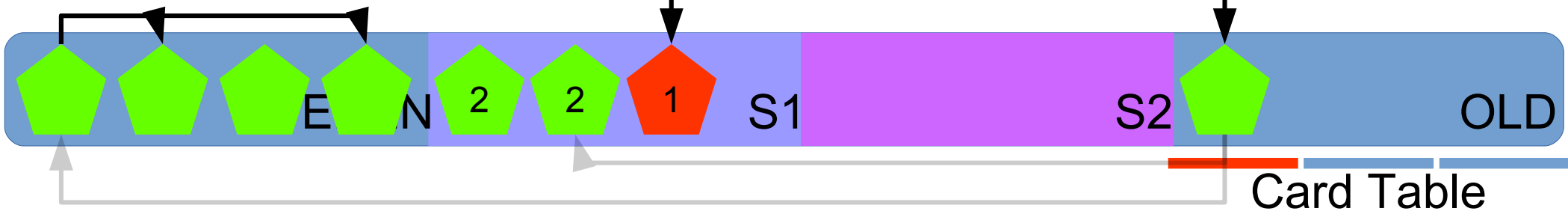
2015-05-26T14:27:44.915-0200: 120.115:  
[GC (Allocation Failure) ... ]

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



Find live objects, starting from GC roots (mark)  
\* Don't look for live objects in Old Gen

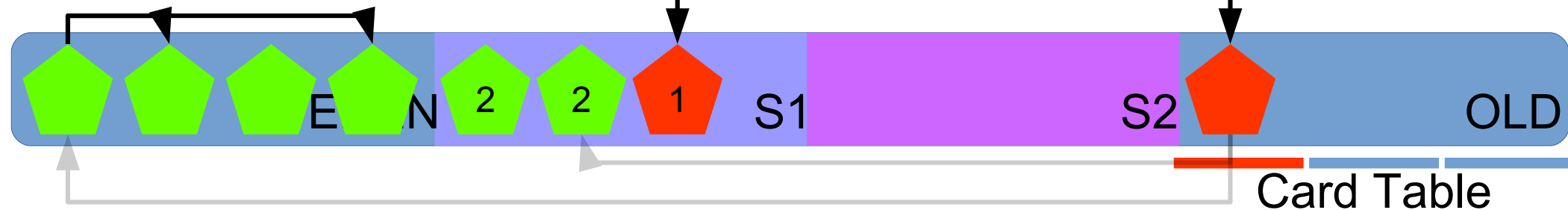


**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



Find live objects, starting from GC roots (mark)

\* Don't look for live objects in Old Gen

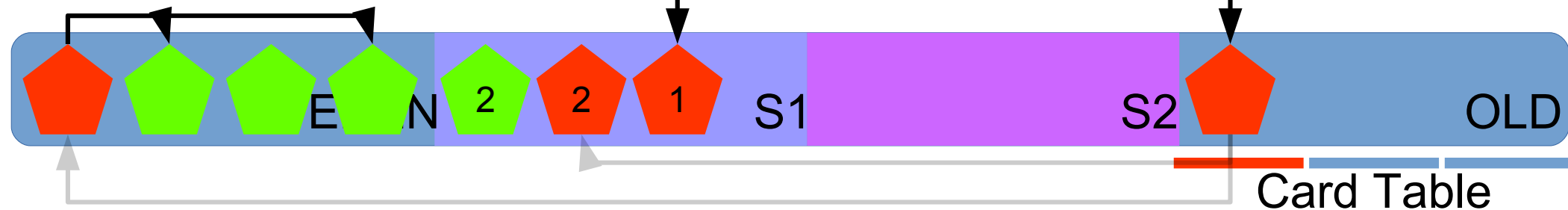
\* Scan Card Table regions for extra references

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



Find live objects, starting from GC roots (mark)

\* Don't look for live objects in Old Gen

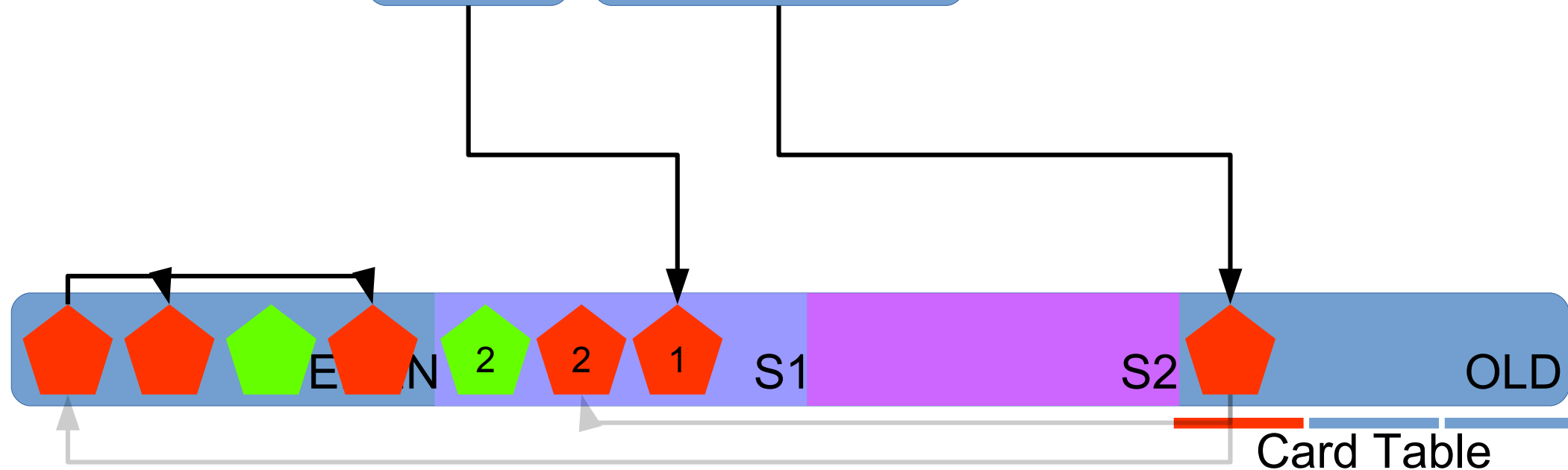
\* Scan Card Table regions for extra references

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



Find live objects, starting from GC roots (mark)

\* Don't look for live objects in Old Gen

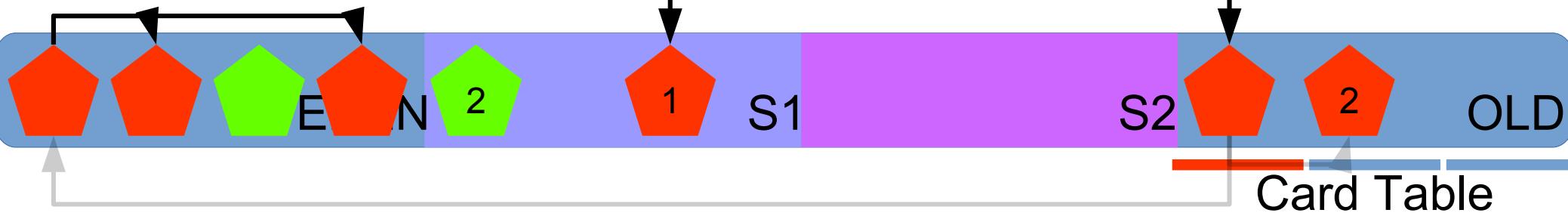
\* Scan Card Table regions for extra references

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



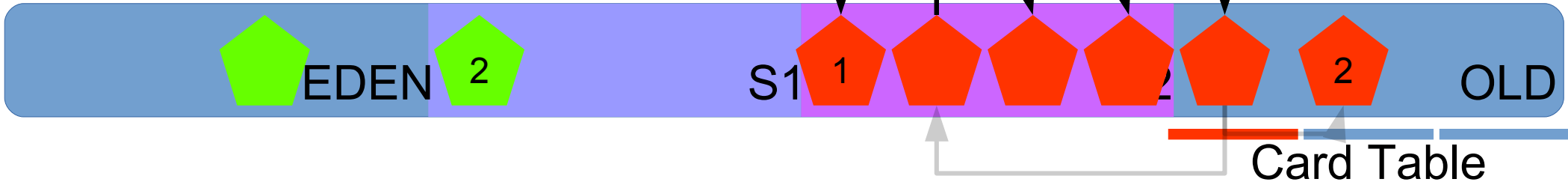
Move live objects to survivors or old (compacting)

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



Move live objects to survivors or old (compacting)

**STOP THE WORLD**

GC ROOTS

STACK

STATIC FIELDS



EDEN

S1

1

2

2

OLD

Card Table

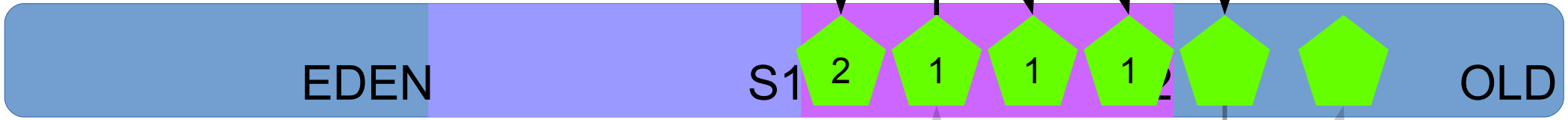
Mark EDEN+S1 as clean

**RUNNING**

GC ROOTS

STACK

STATIC FIELDS



EDEN

S1

OLD

Card Table

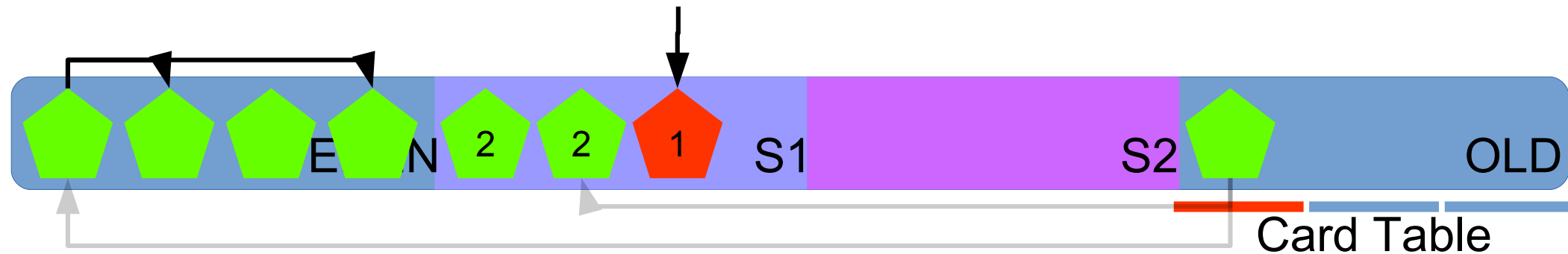
# Recap: ParallelGC

- Allocate to eden
- Copy live objects to survivors or old
- Clear entire eden + cleared survivor space
- Promote repeat-survivors to Old gen
- Use Card Tables to avoid scanning Old gen
- Use Old objects as GC roots in minor collection
- Full GC when everything is full



# ParallelGC insights (1)

- Reference scanning expensive
- $\text{size}(\text{Old}) > \text{size}(\text{Young})$
- Card Table  $\rightarrow$  avoid most scanning



# ParallelGC insights (2)

- Objects die young
- Copy only live objects
- Don't touch Old Gen until Full GC

# G1

- Generational: young (eden, survivor), old
- Aims for short Stop The World pauses
- Thousands of non-contiguous regions
- Concurrent marking

# RUNNING/ STOP THE WORLD

GC ROOTS

STACK

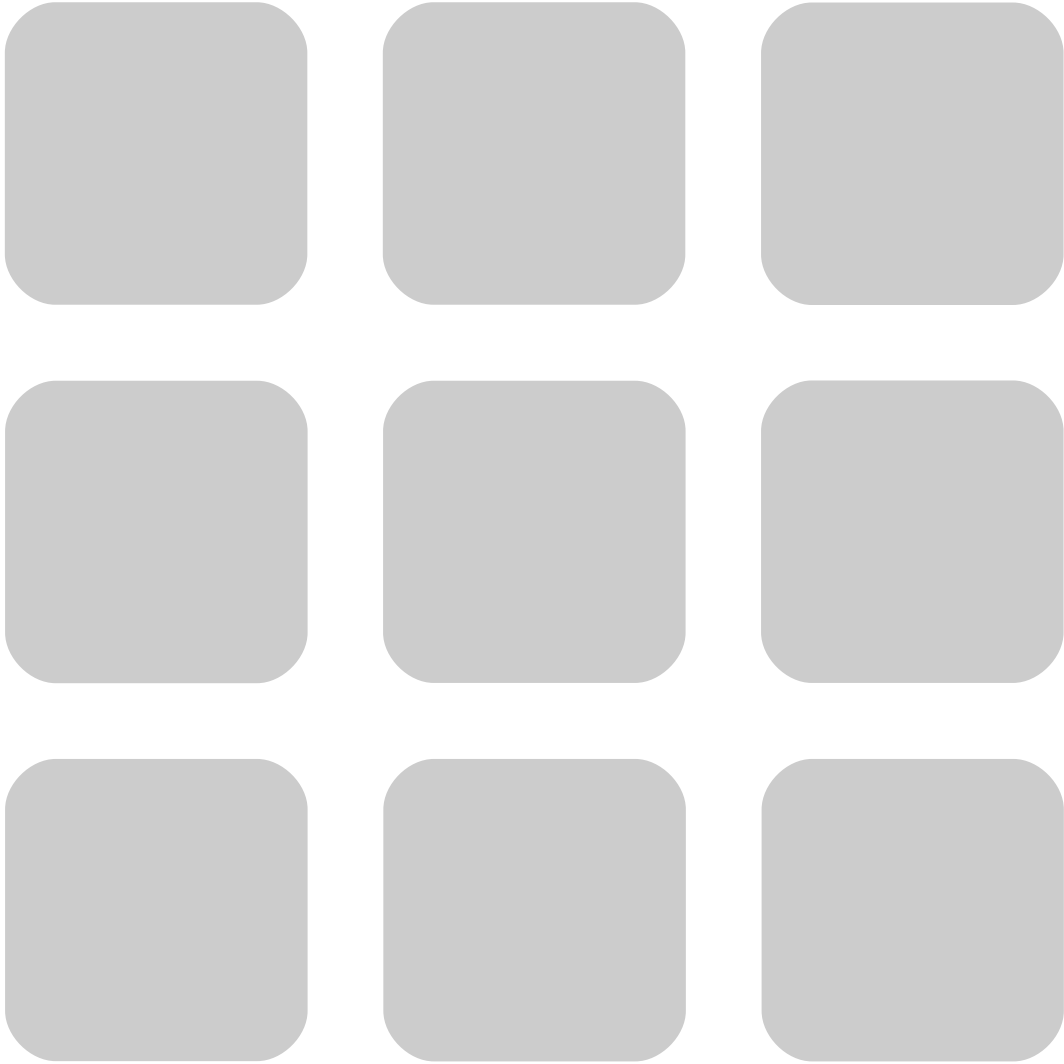
STATIC FIELDS

FREE

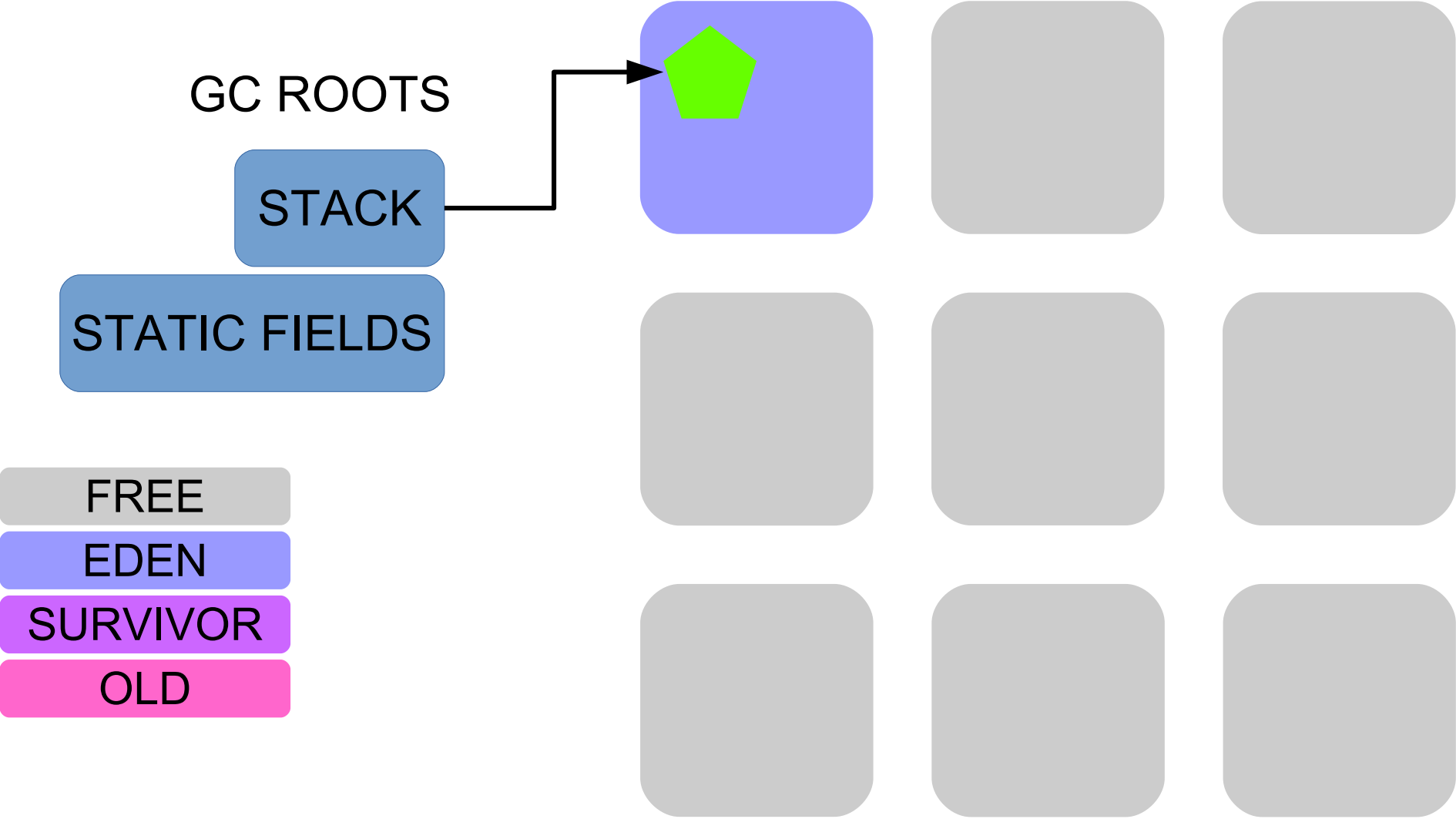
EDEN

SURVIVOR

OLD



# RUNNING



GC ROOTS

STACK

STATIC FIELDS

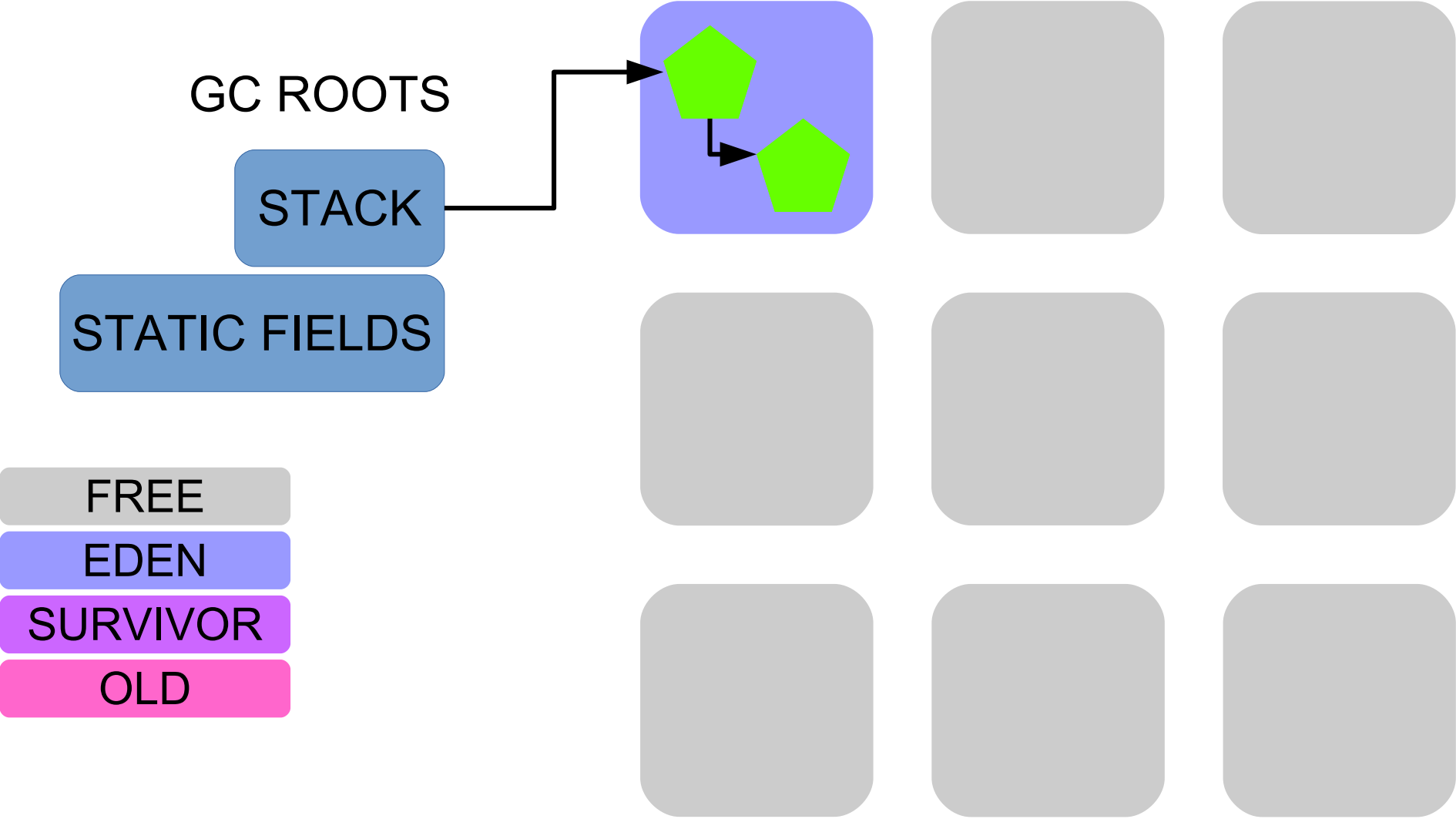
FREE

EDEN

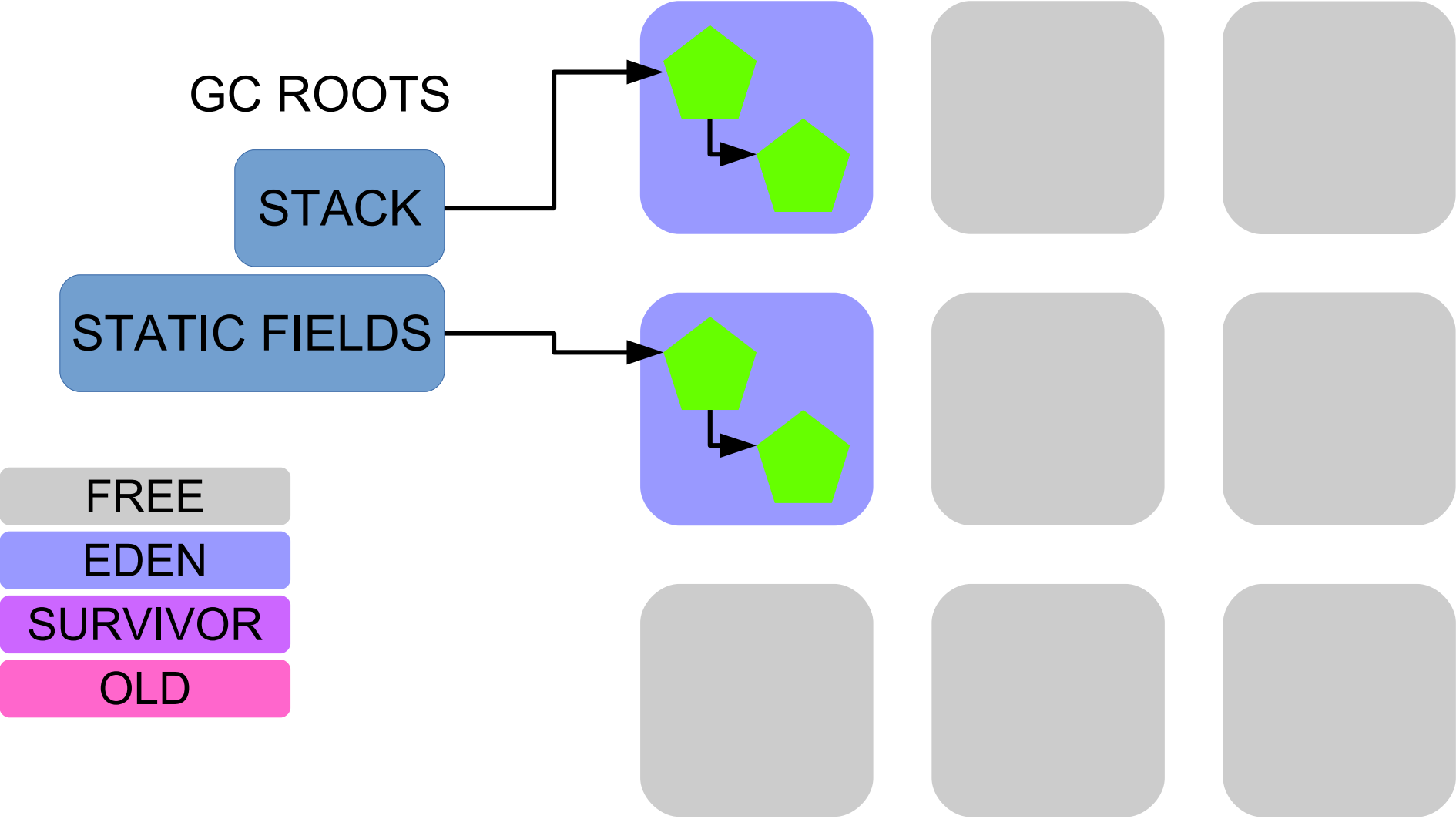
SURVIVOR

OLD

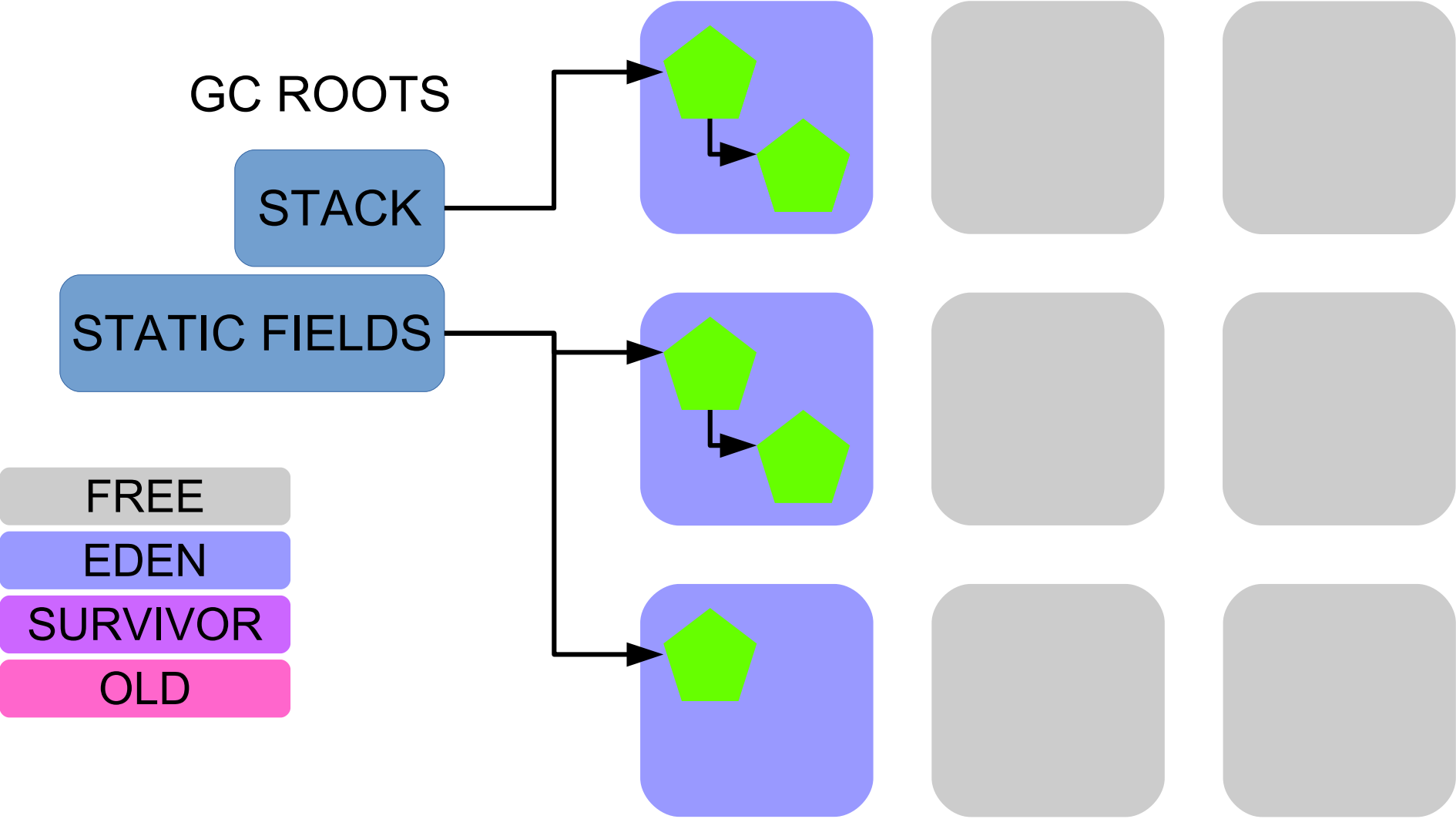
# RUNNING



# RUNNING



# RUNNING





# RUNNING

GC ROOTS

STACK

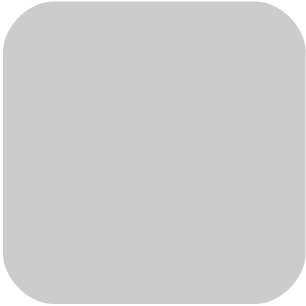
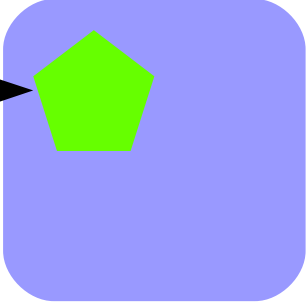
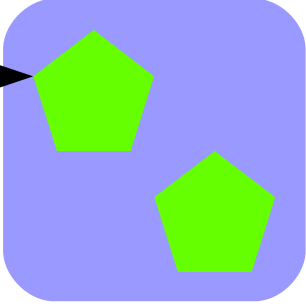
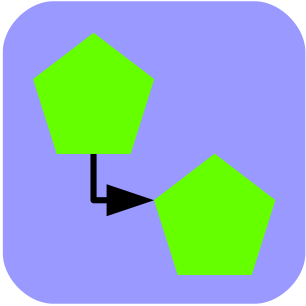
STATIC FIELDS

FREE

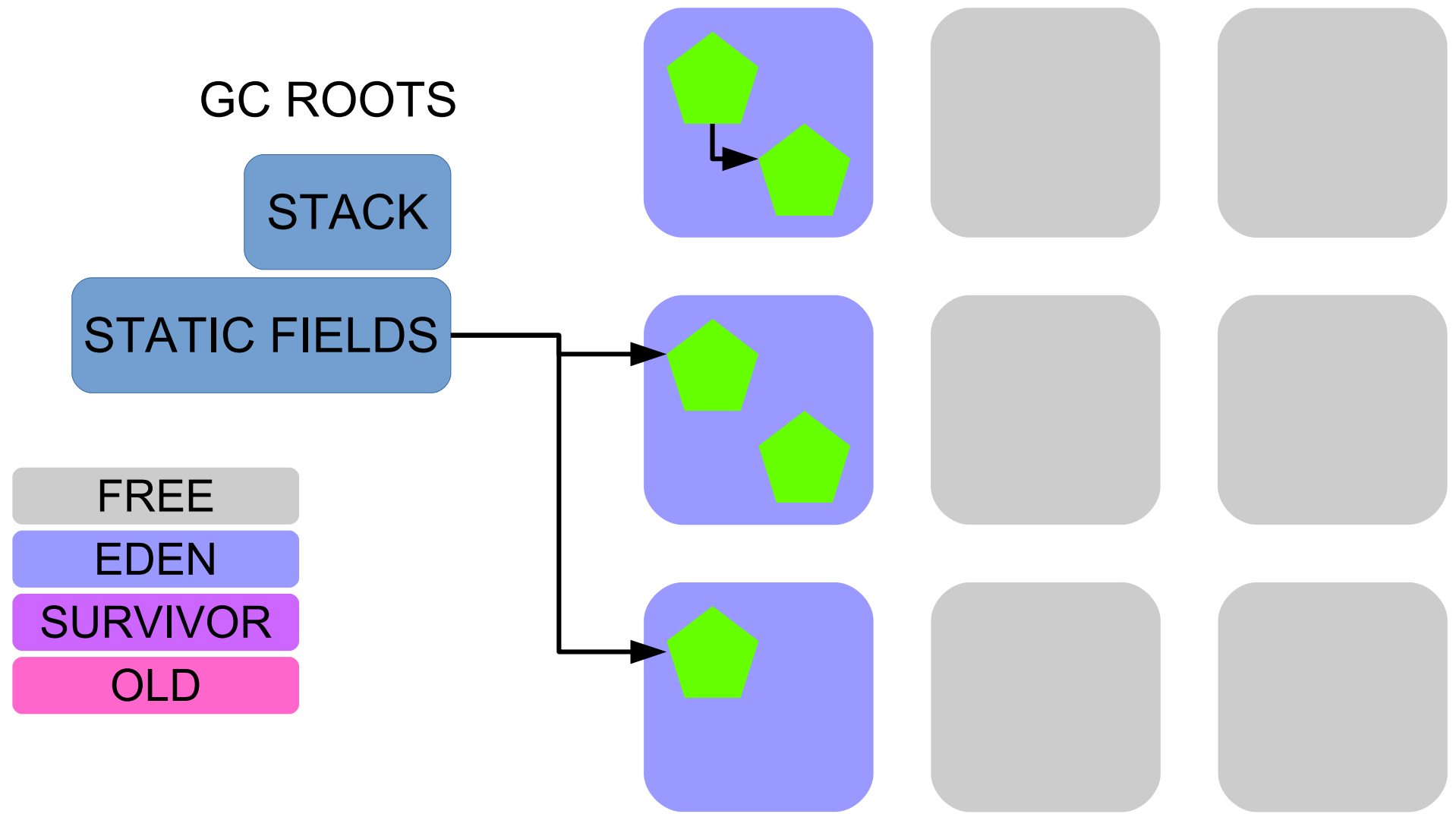
EDEN

SURVIVOR

OLD

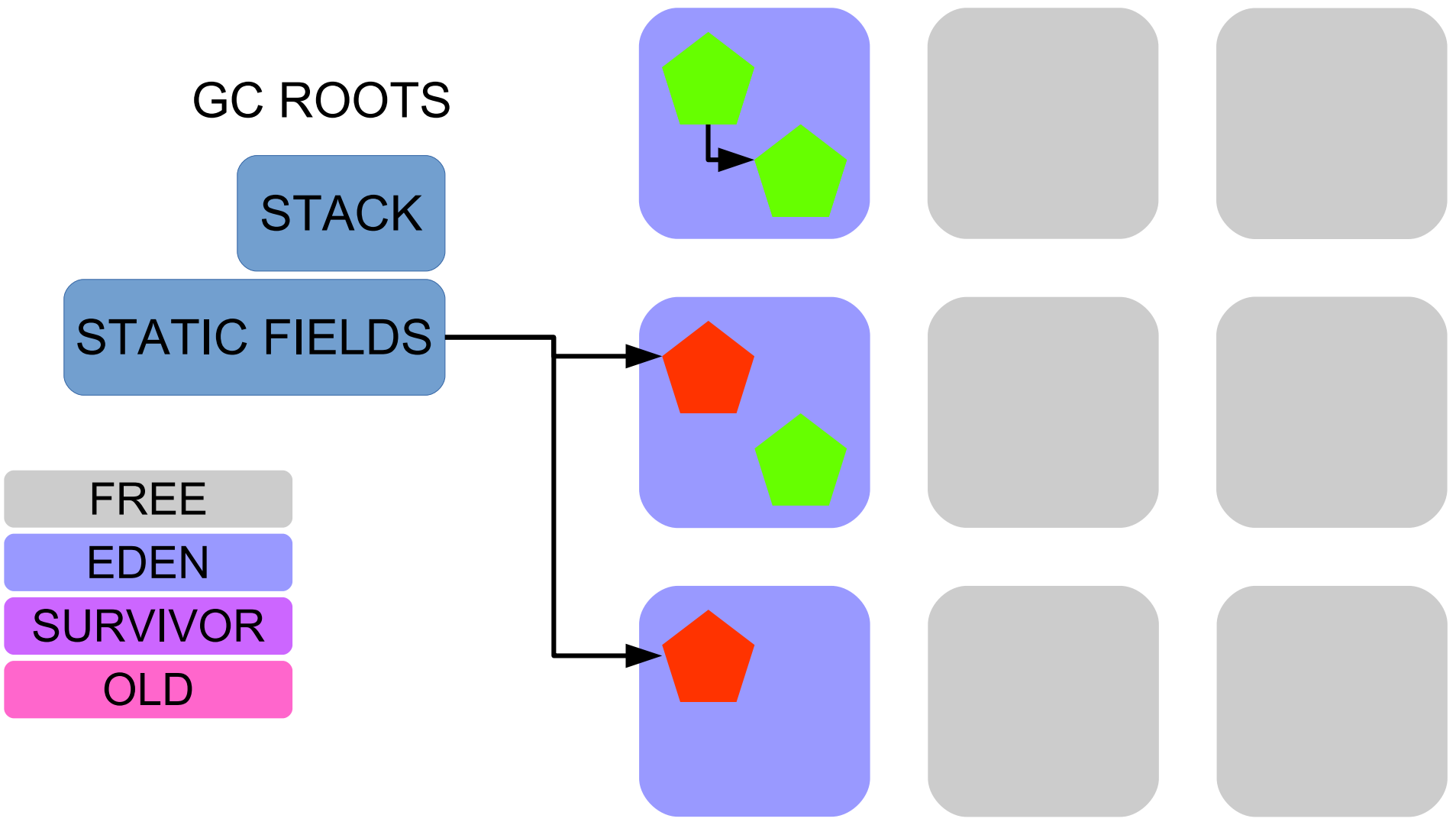


# STOP THE WORLD



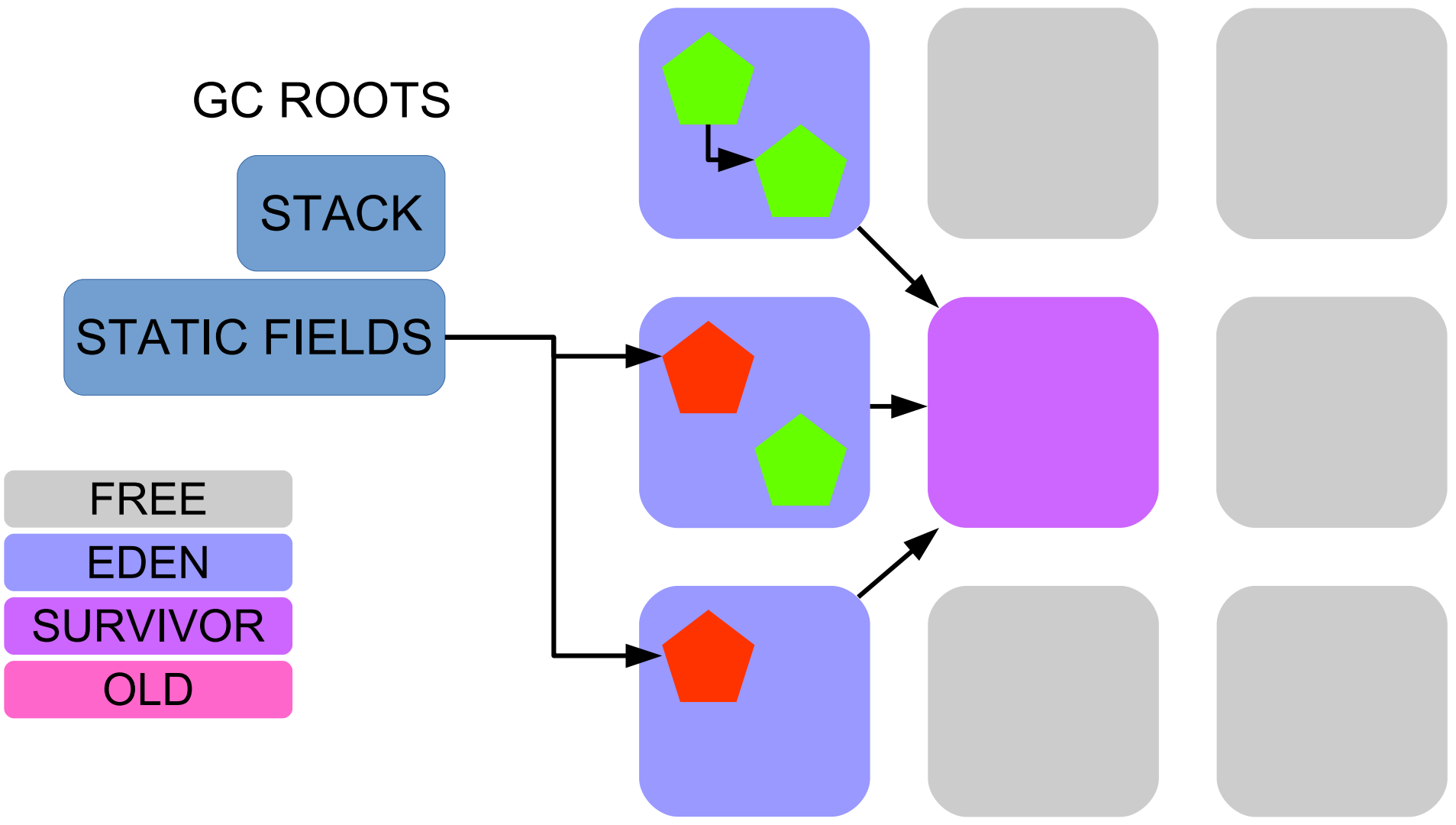
[GC pause (G1 Evacuation Pause) (young) ..]  
Collection Set: all young

# STOP THE WORLD

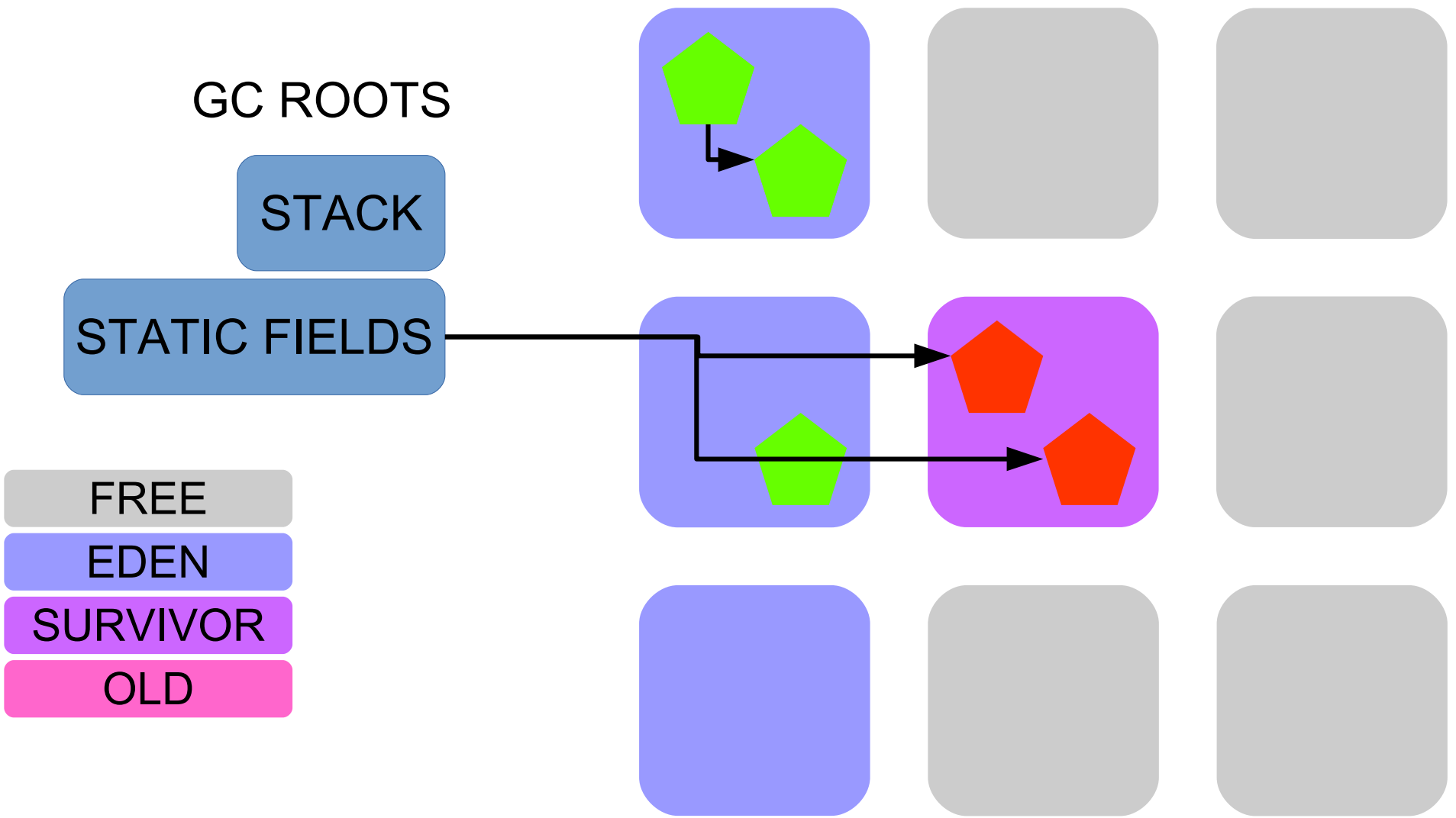


Mark objects reachable from roots

# STOP THE WORLD

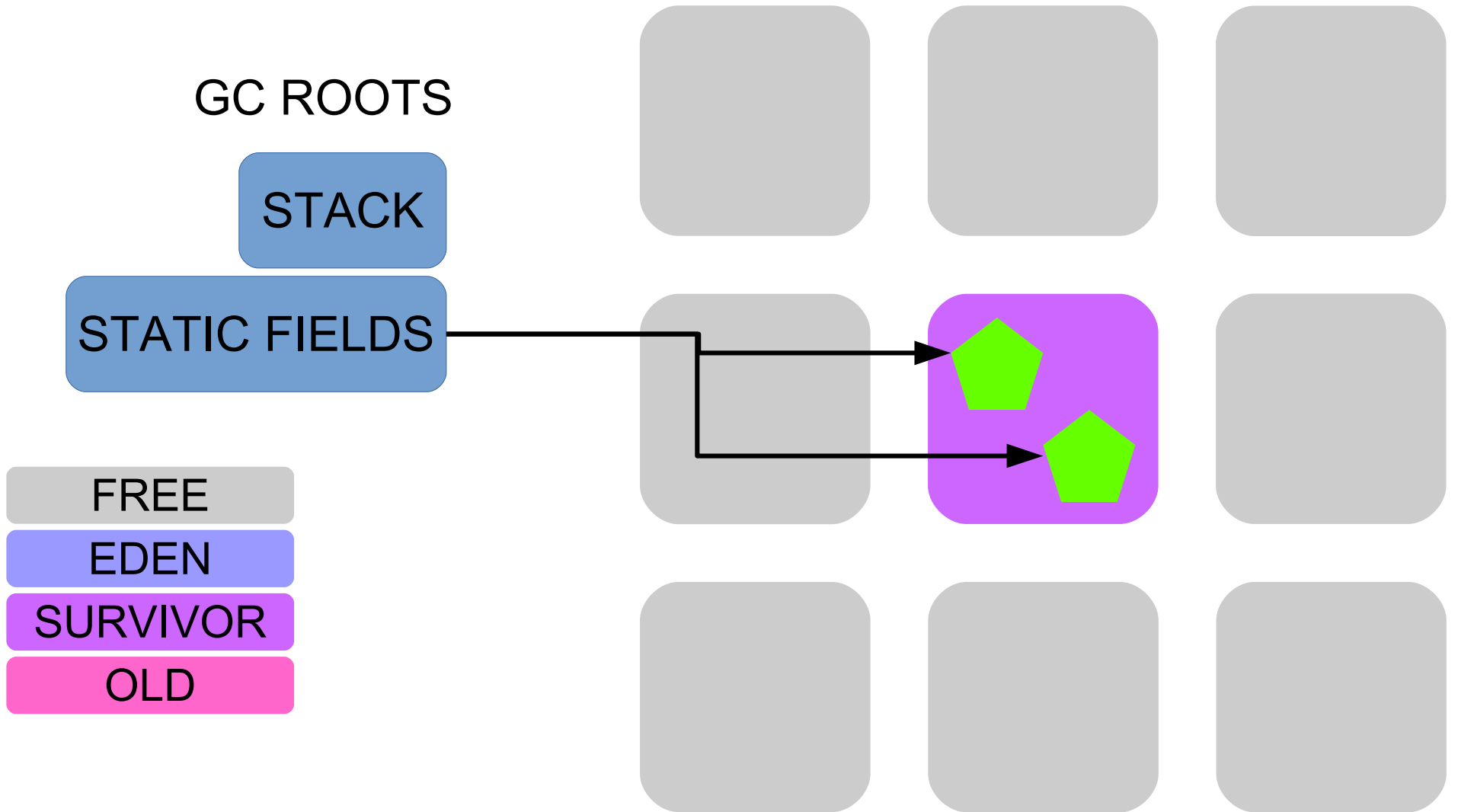


# STOP THE WORLD



Move to new survivor region

# STOP THE WORLD



Free evacuated regions

# RUNNING

GC ROOTS

STACK

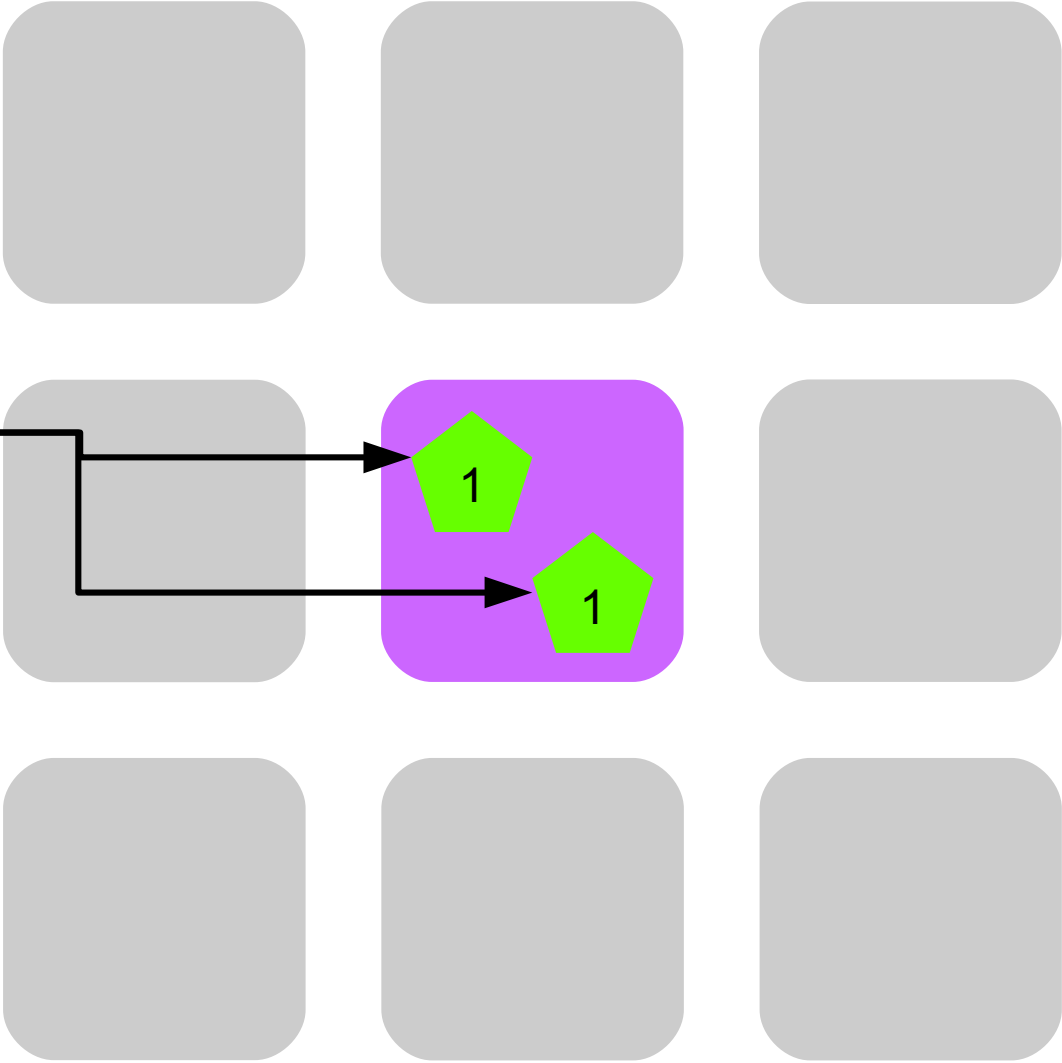
STATIC FIELDS

FREE

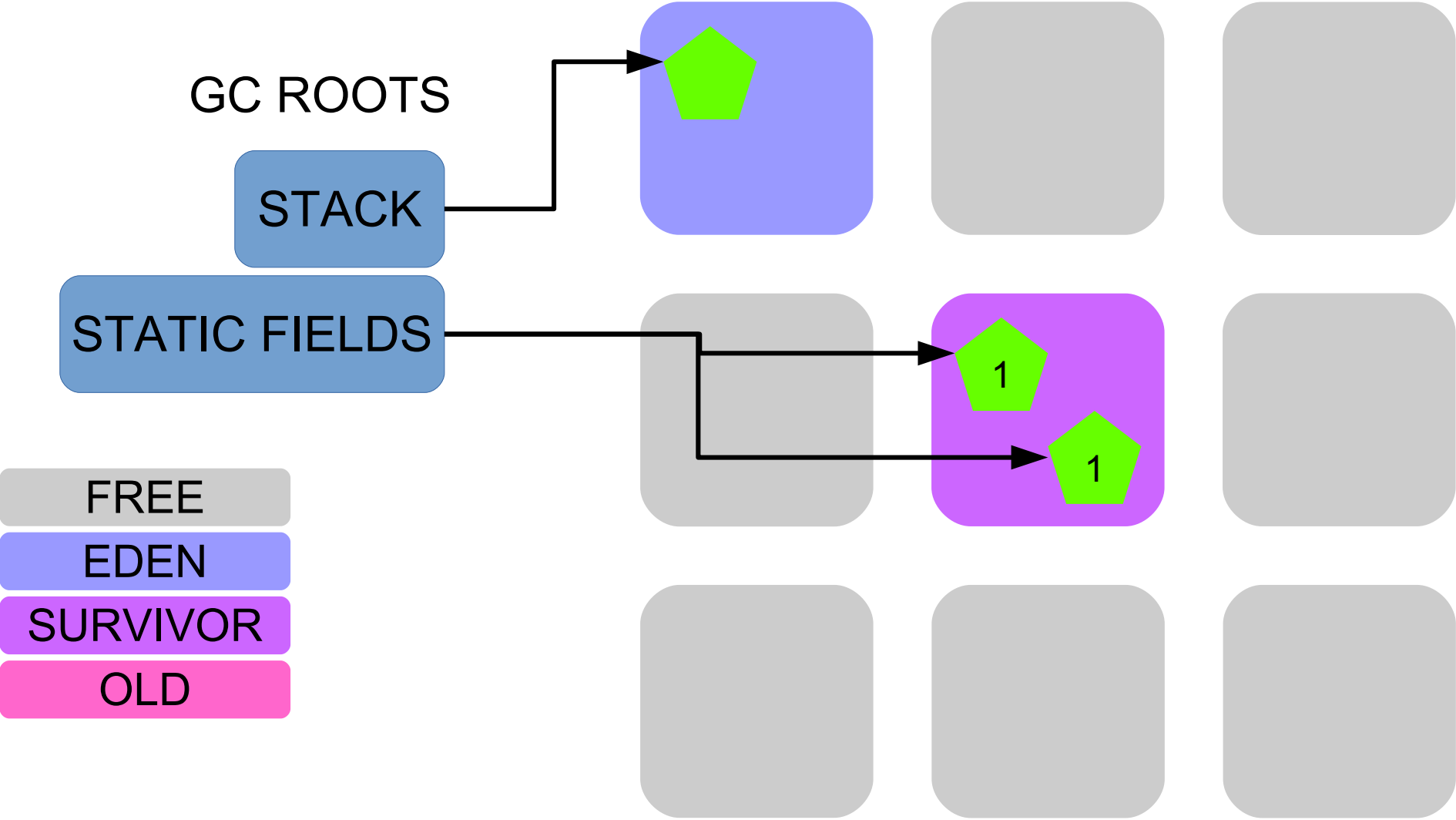
EDEN

SURVIVOR

OLD

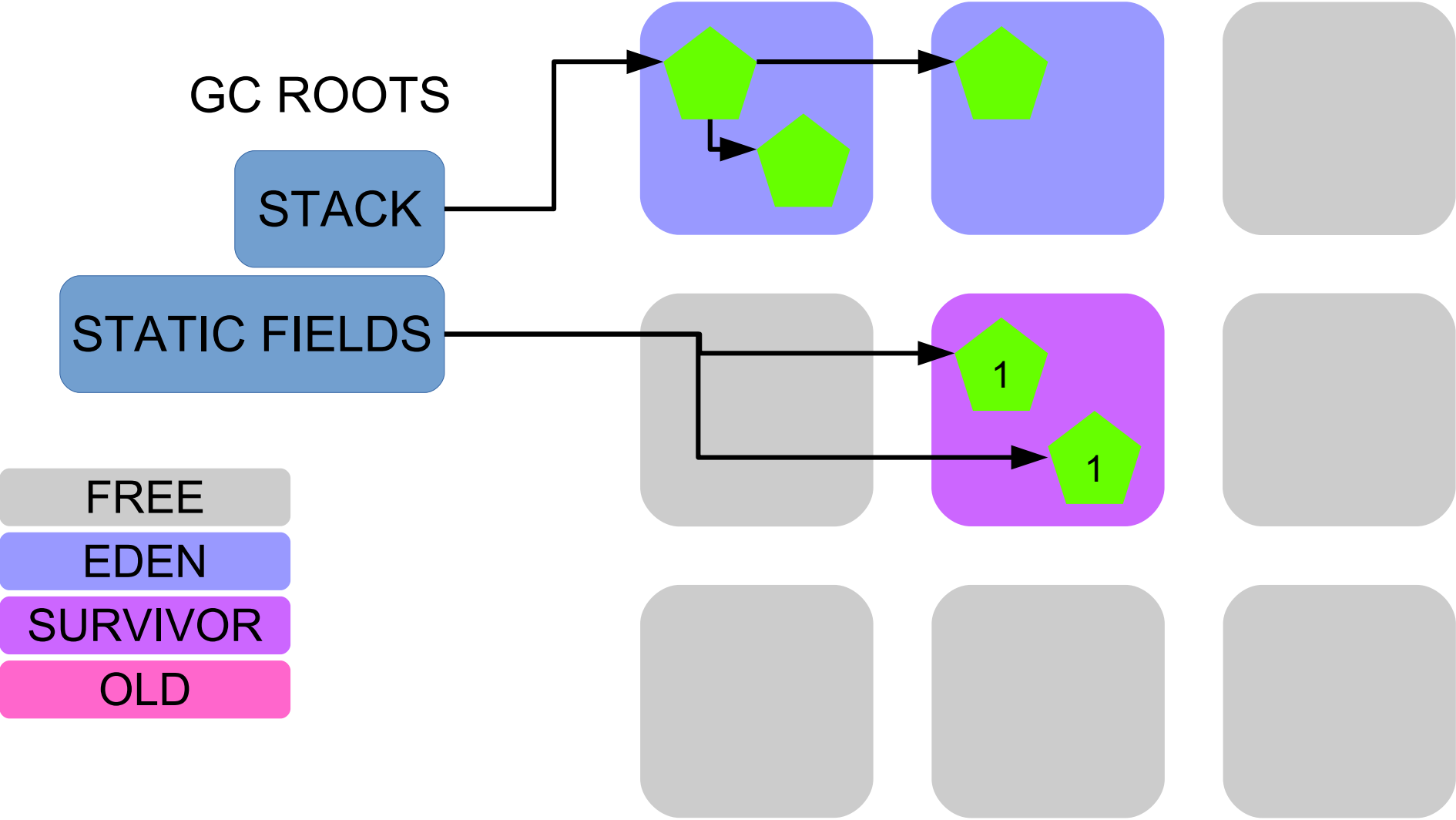


# RUNNING

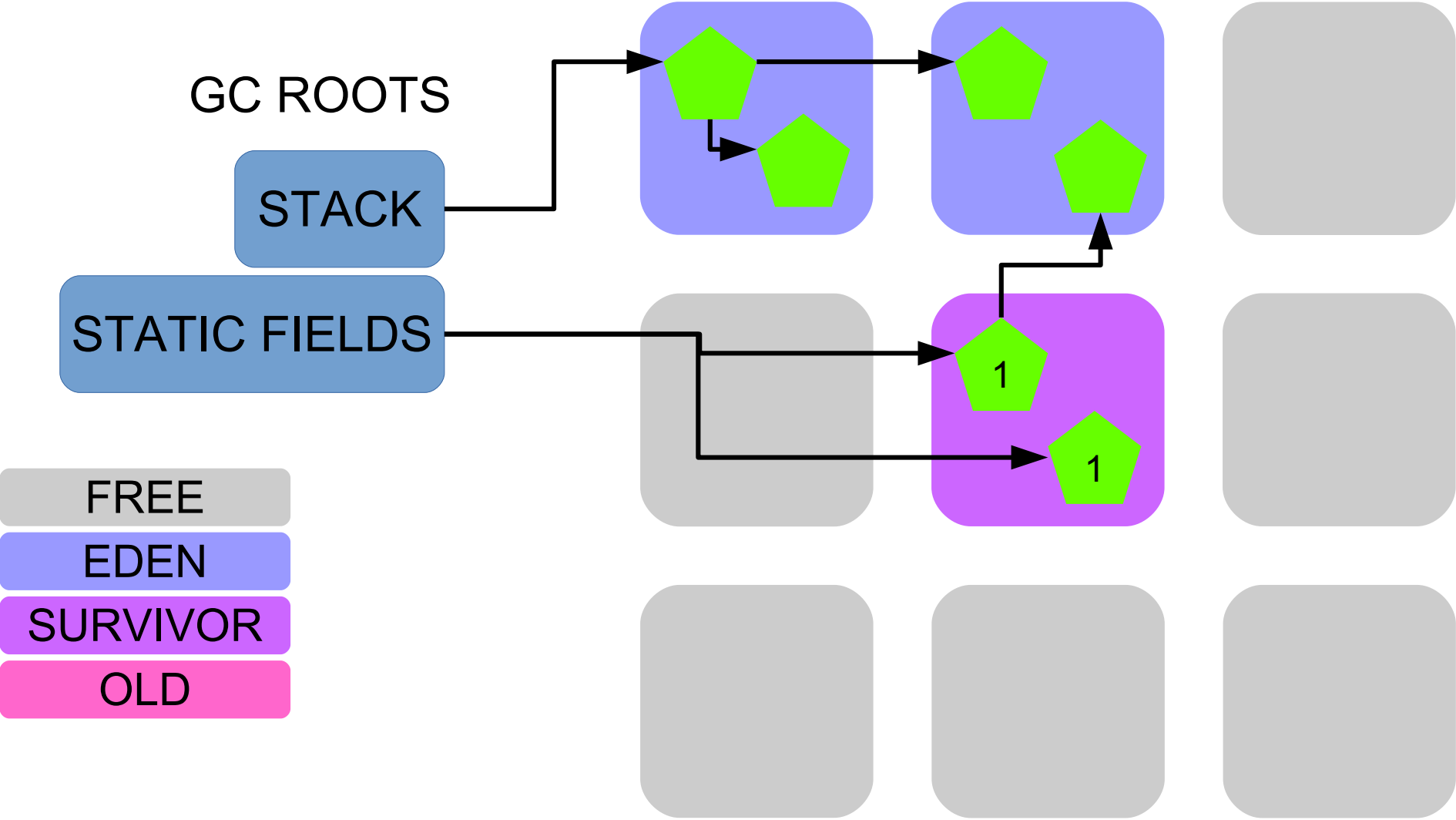




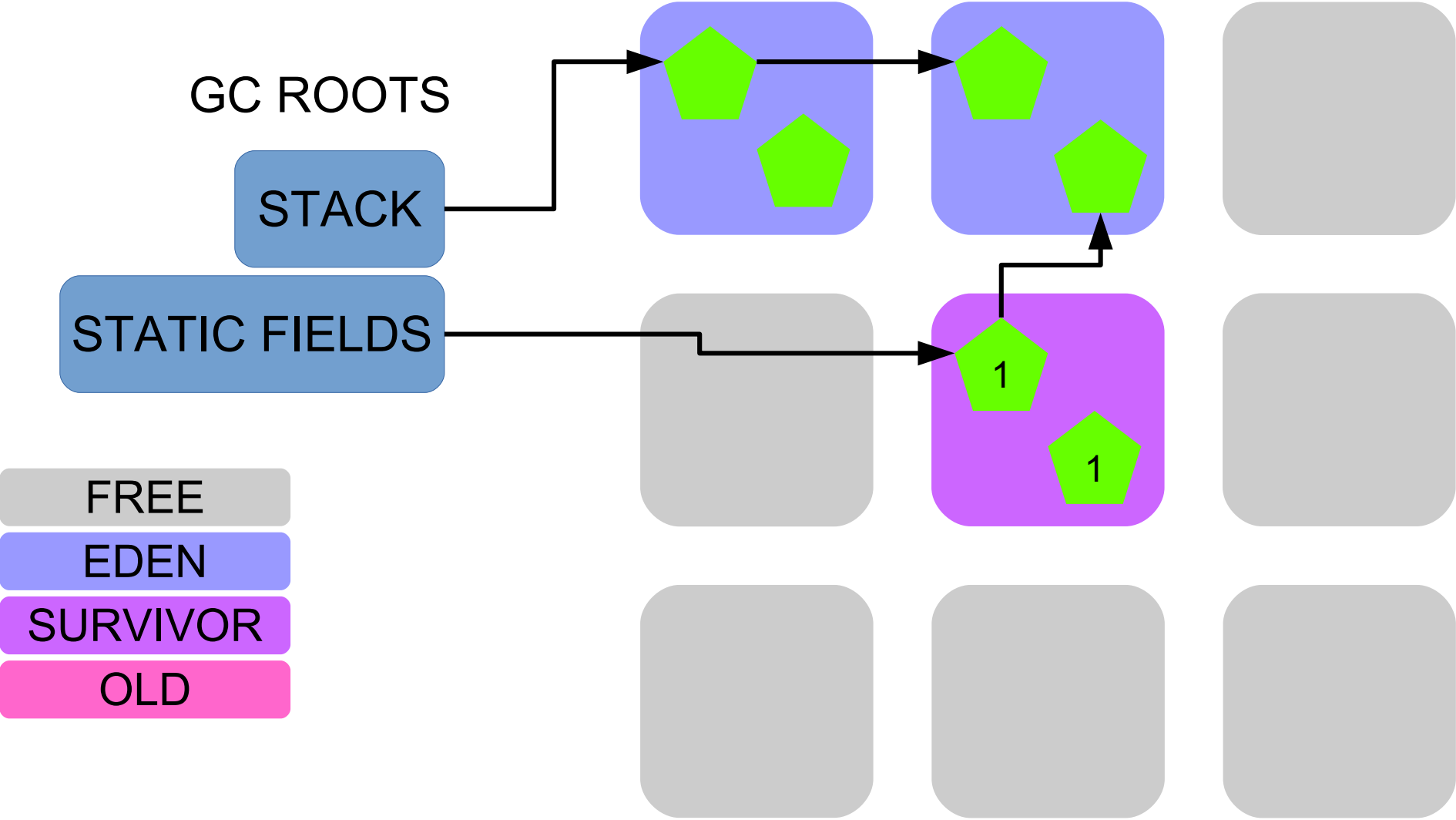
# RUNNING



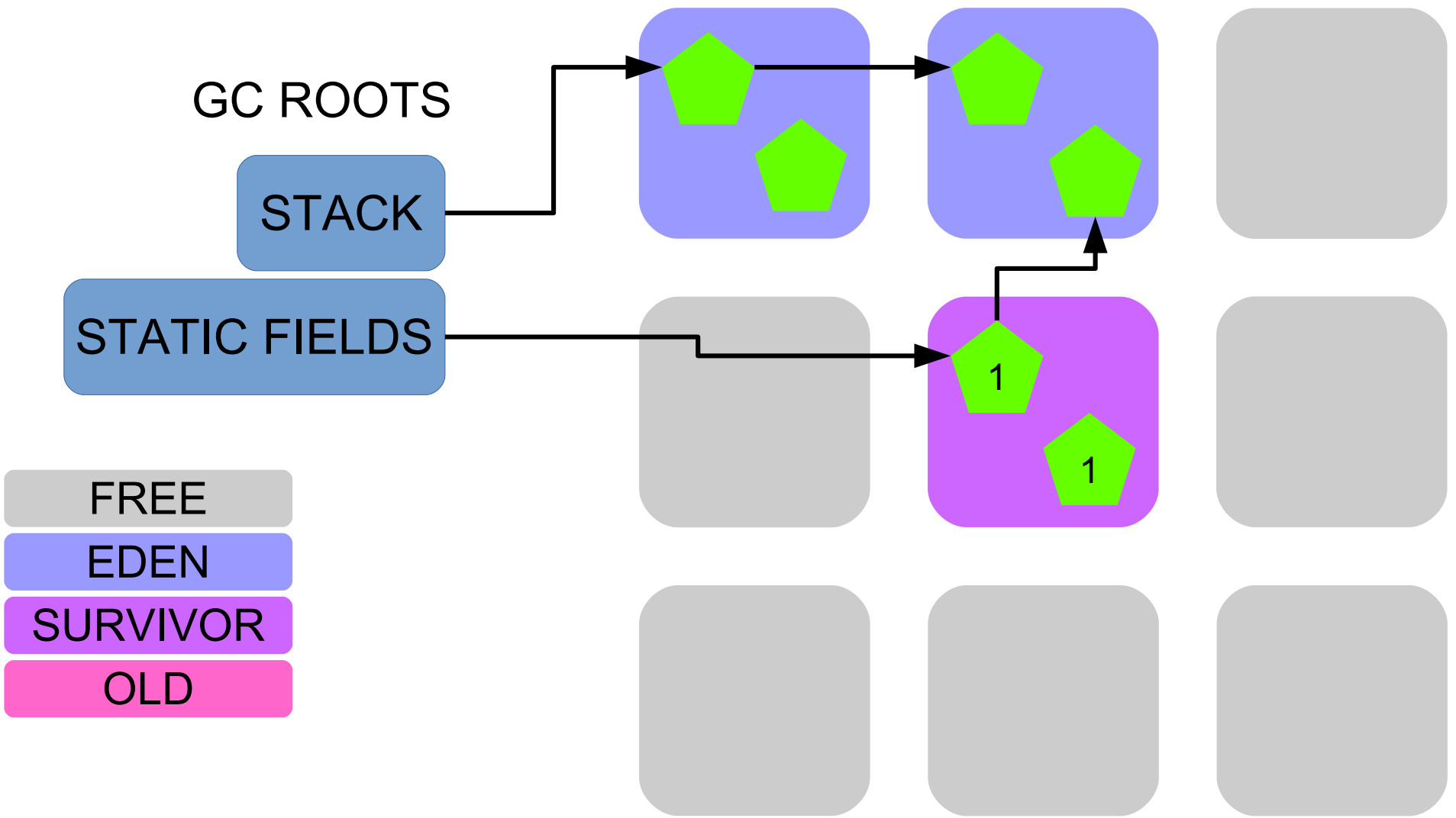
# RUNNING



# RUNNING

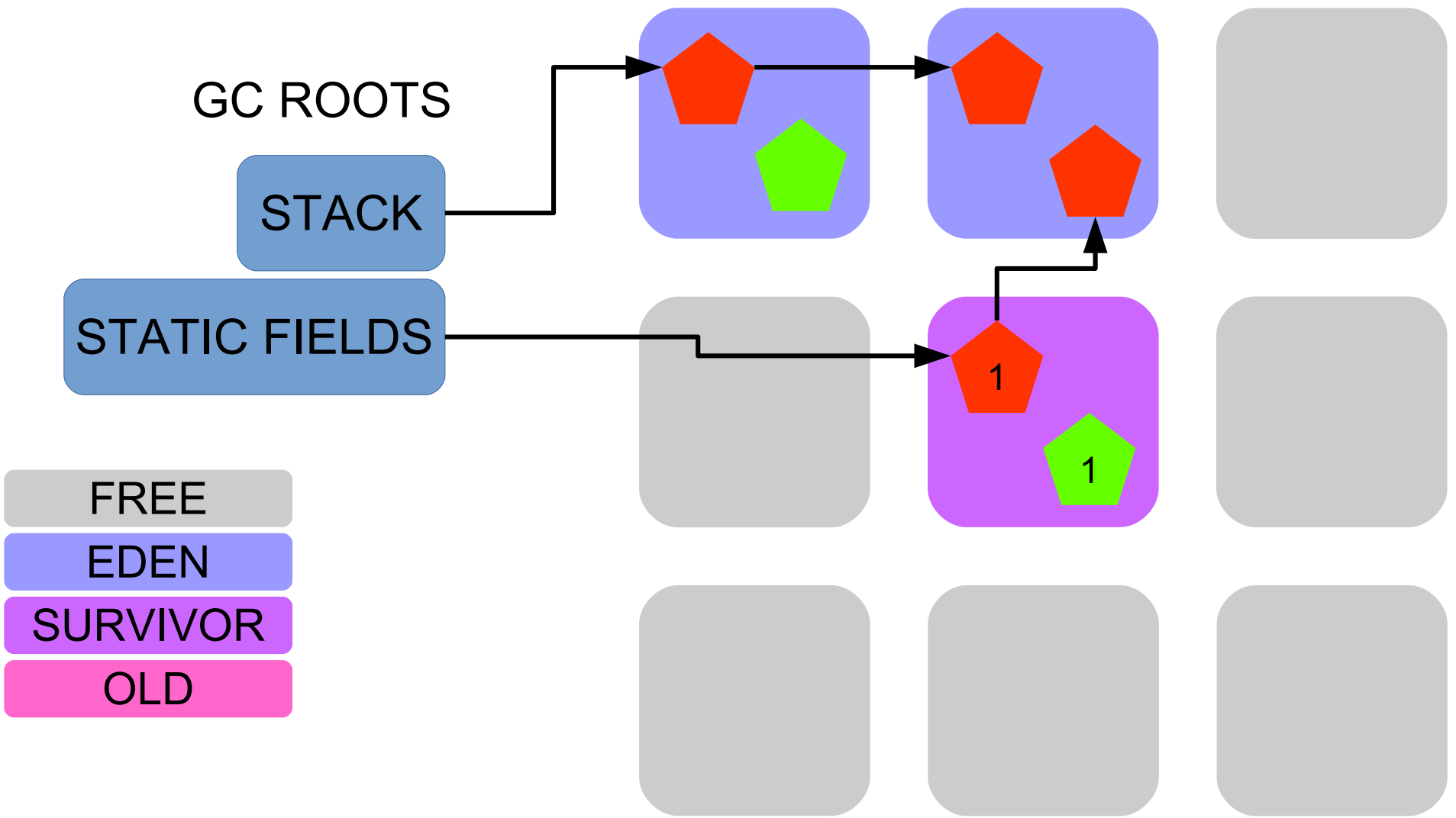


# STOP THE WORLD



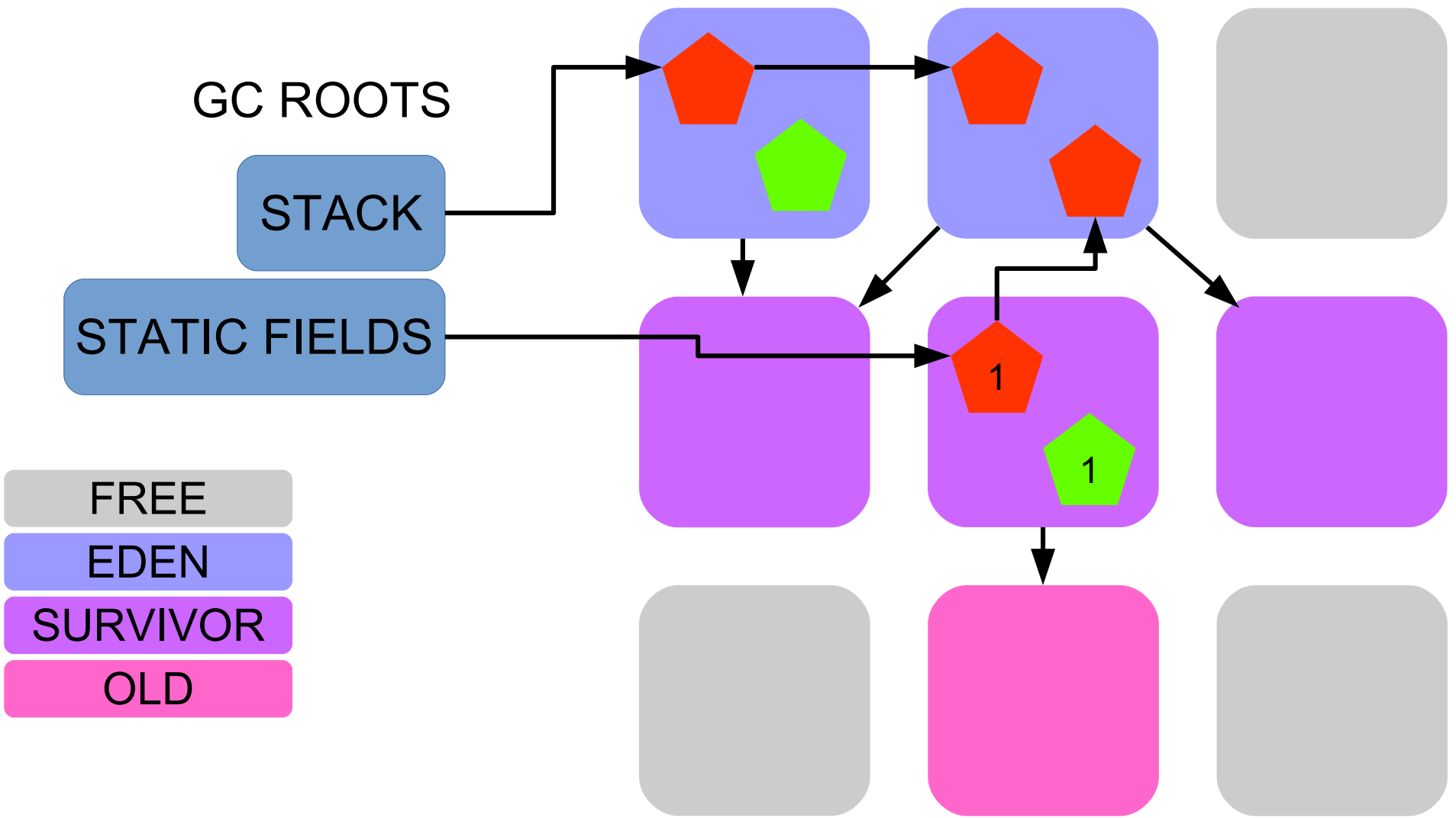
[GC pause (G1 Evacuation Pause) (young) ..]  
Collection Set: all young

# STOP THE WORLD



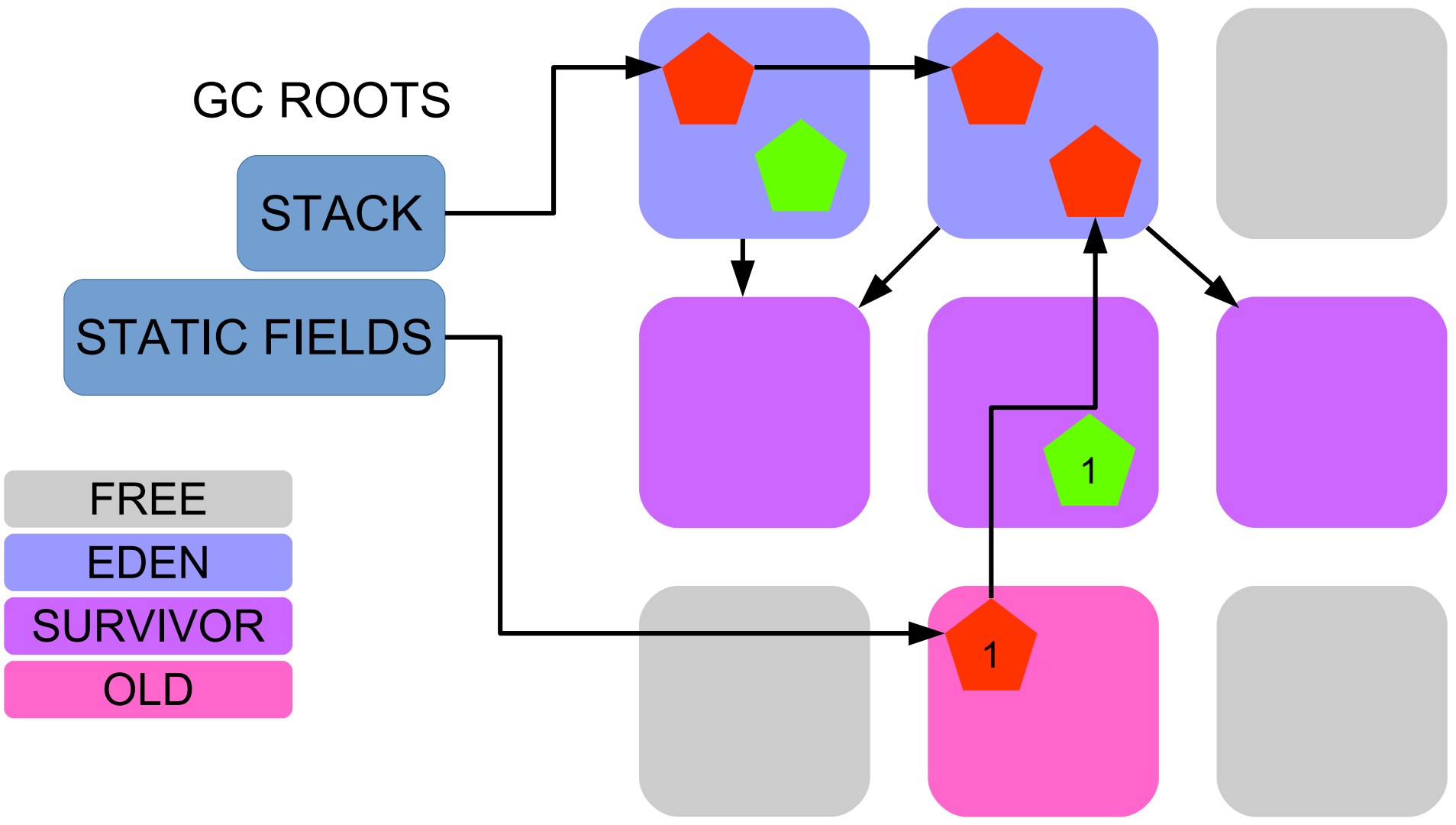
Mark objects reachable from roots

# STOP THE WORLD



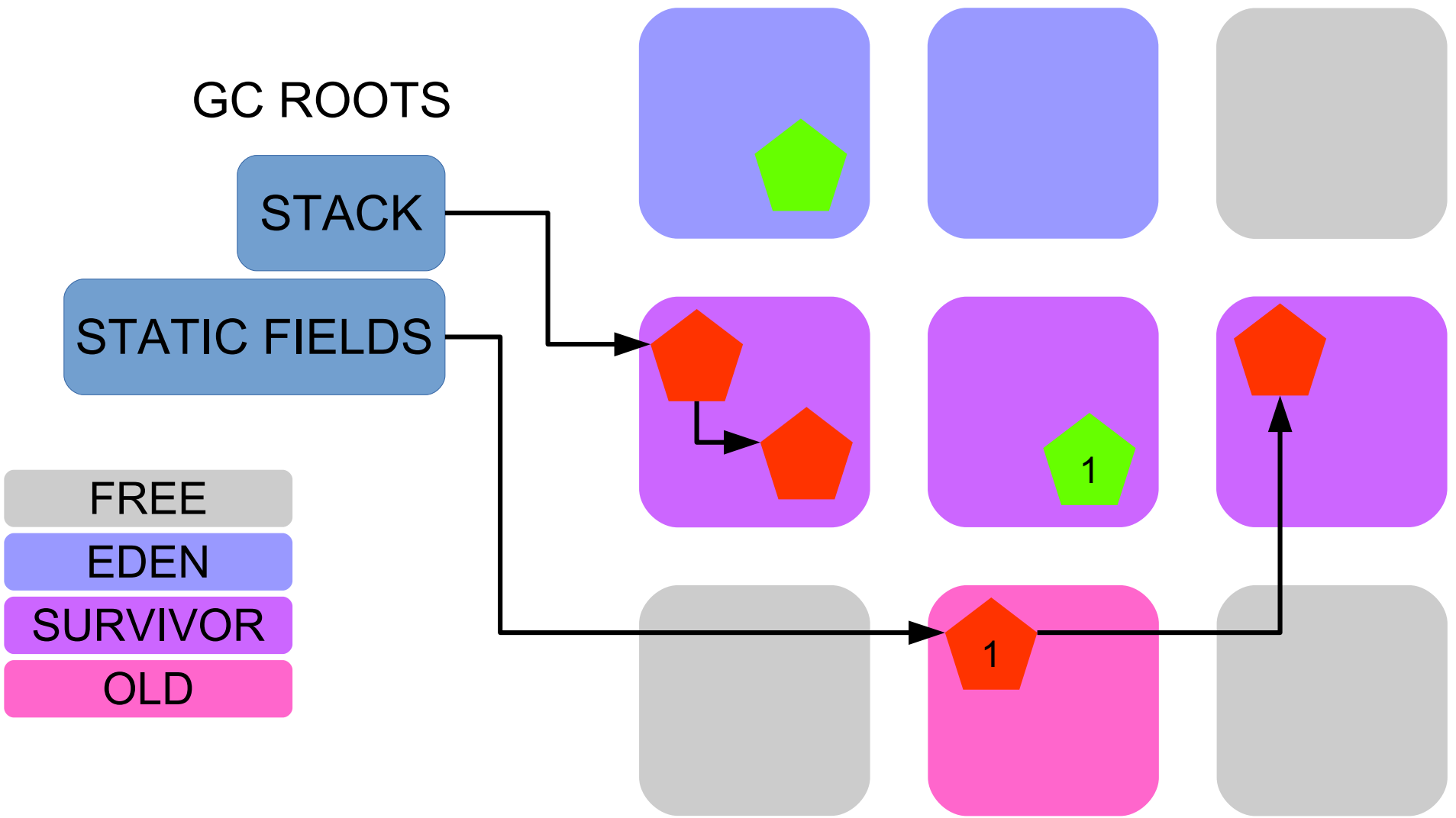
Move to new survivor region / old

# STOP THE WORLD



Move to new survivor region / old

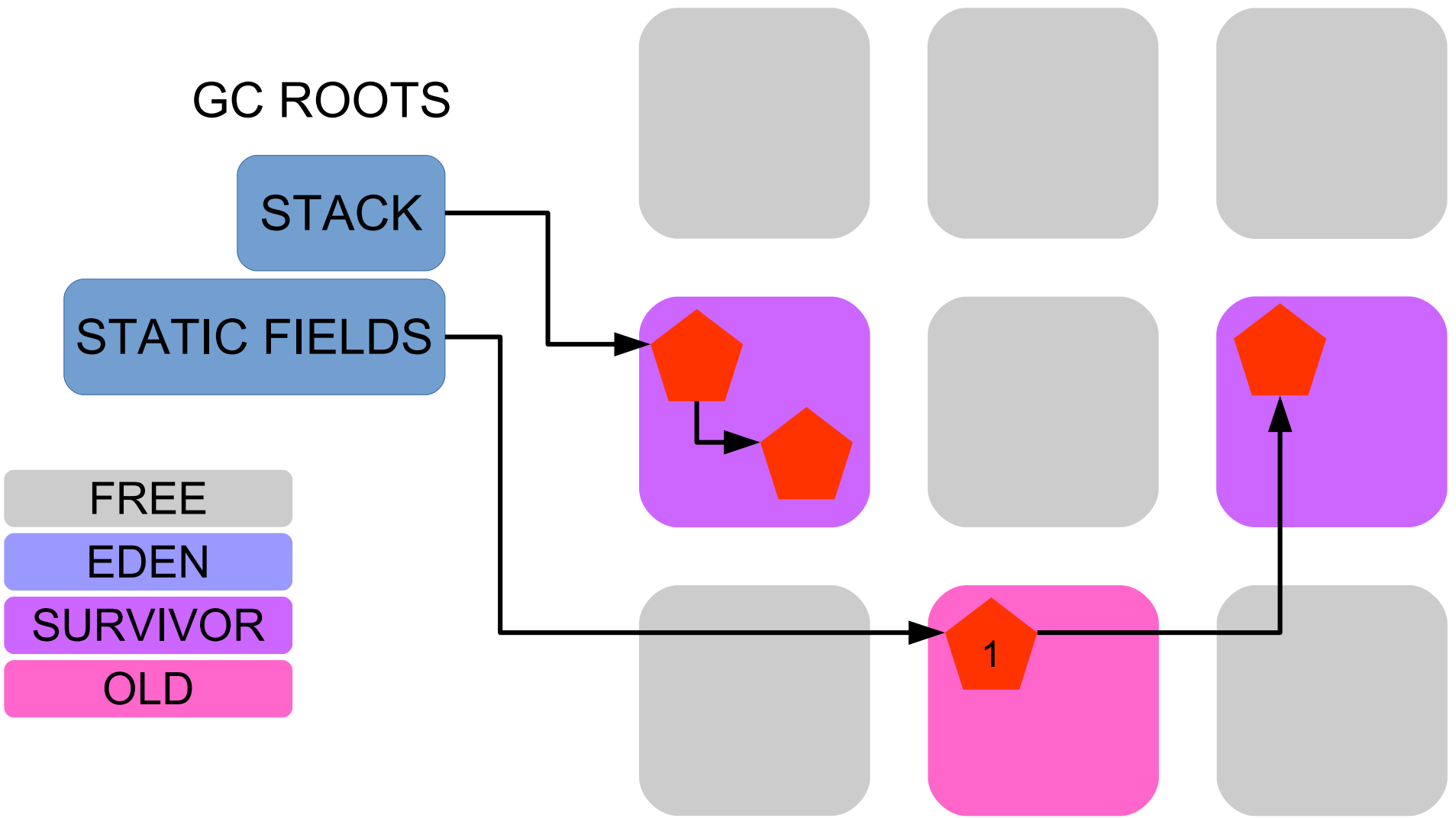
# STOP THE WORLD



Move to new survivor region / old

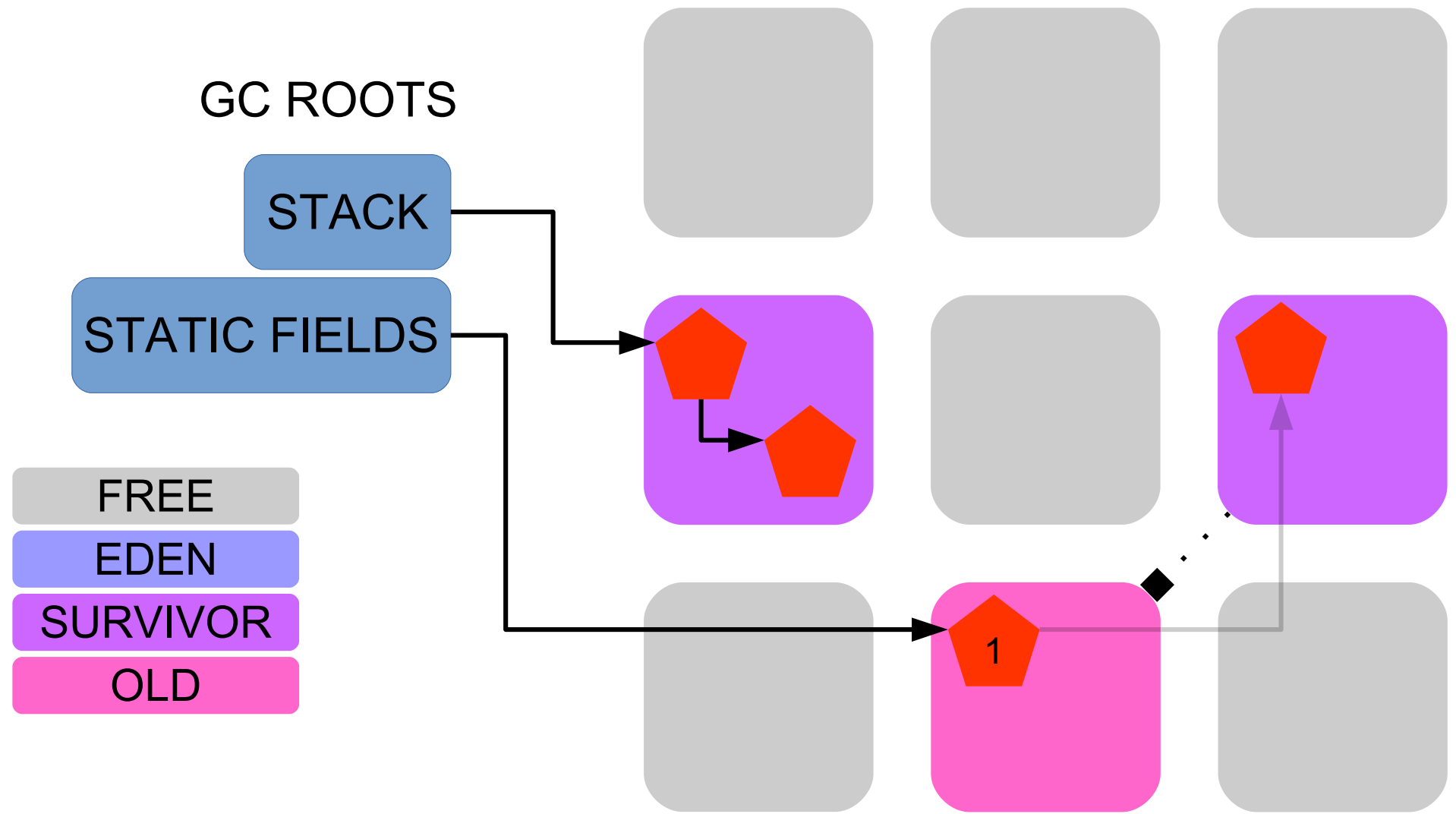


# STOP THE WORLD



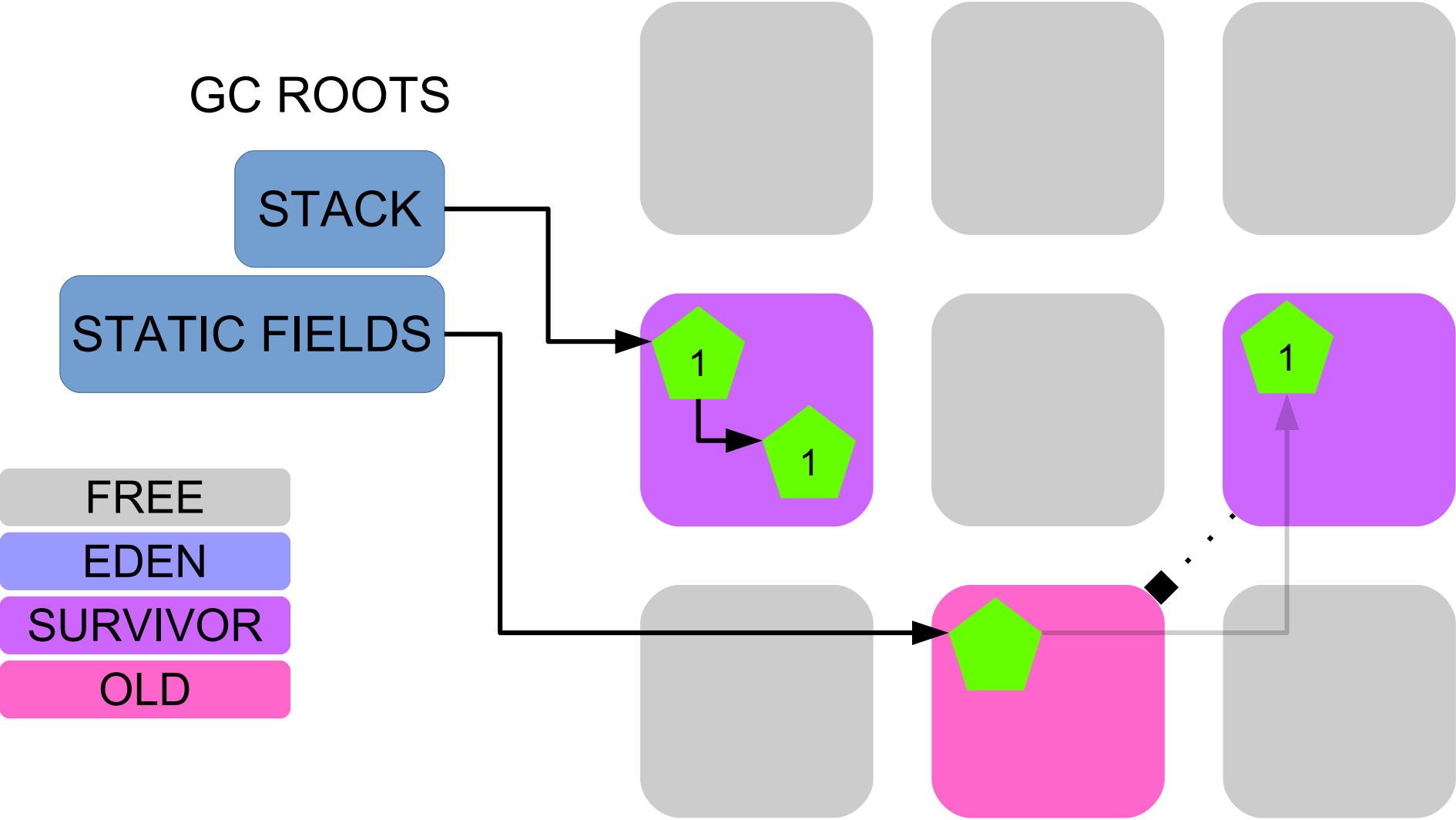
Free evacuated regions

# STOP THE WORLD



Update RSets (Per region Card Table on steroids)  
old->old, old->young

# RUNNING



GC ROOTS

STACK

STATIC FIELDS

FREE

EDEN

SURVIVOR

OLD

# RUNNING

GC ROOTS

STACK

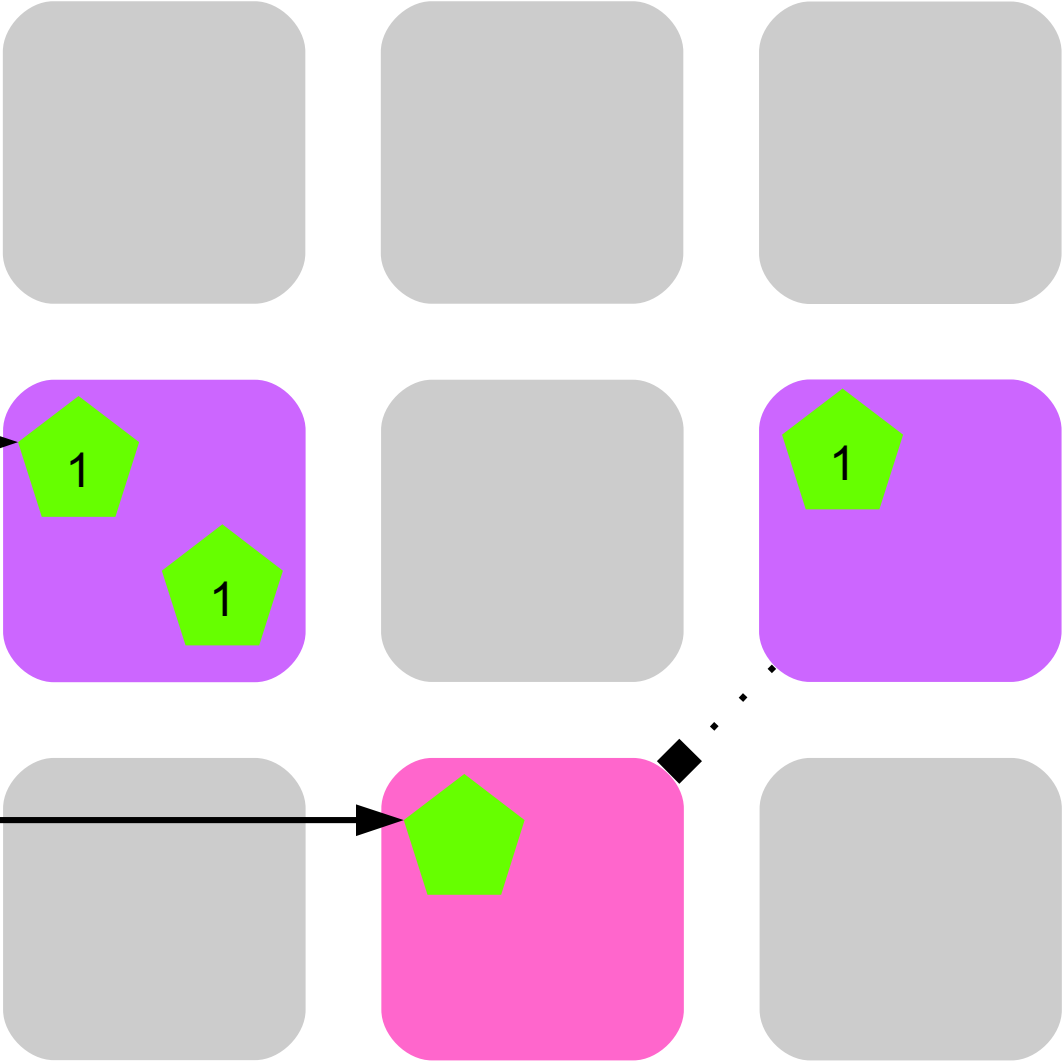
STATIC FIELDS

FREE

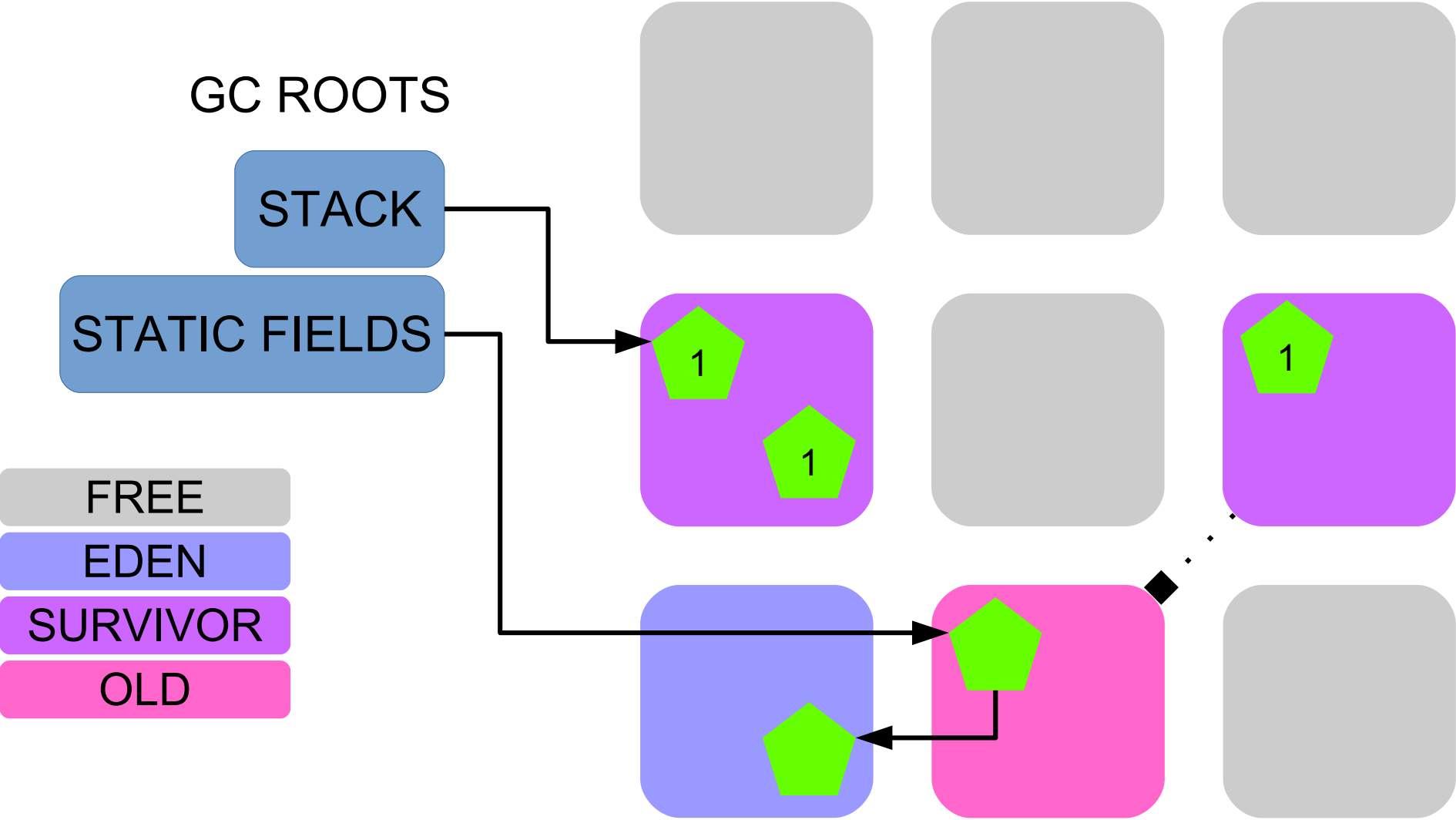
EDEN

SURVIVOR

OLD



# RUNNING



# RUNNING

GC ROOTS

STACK

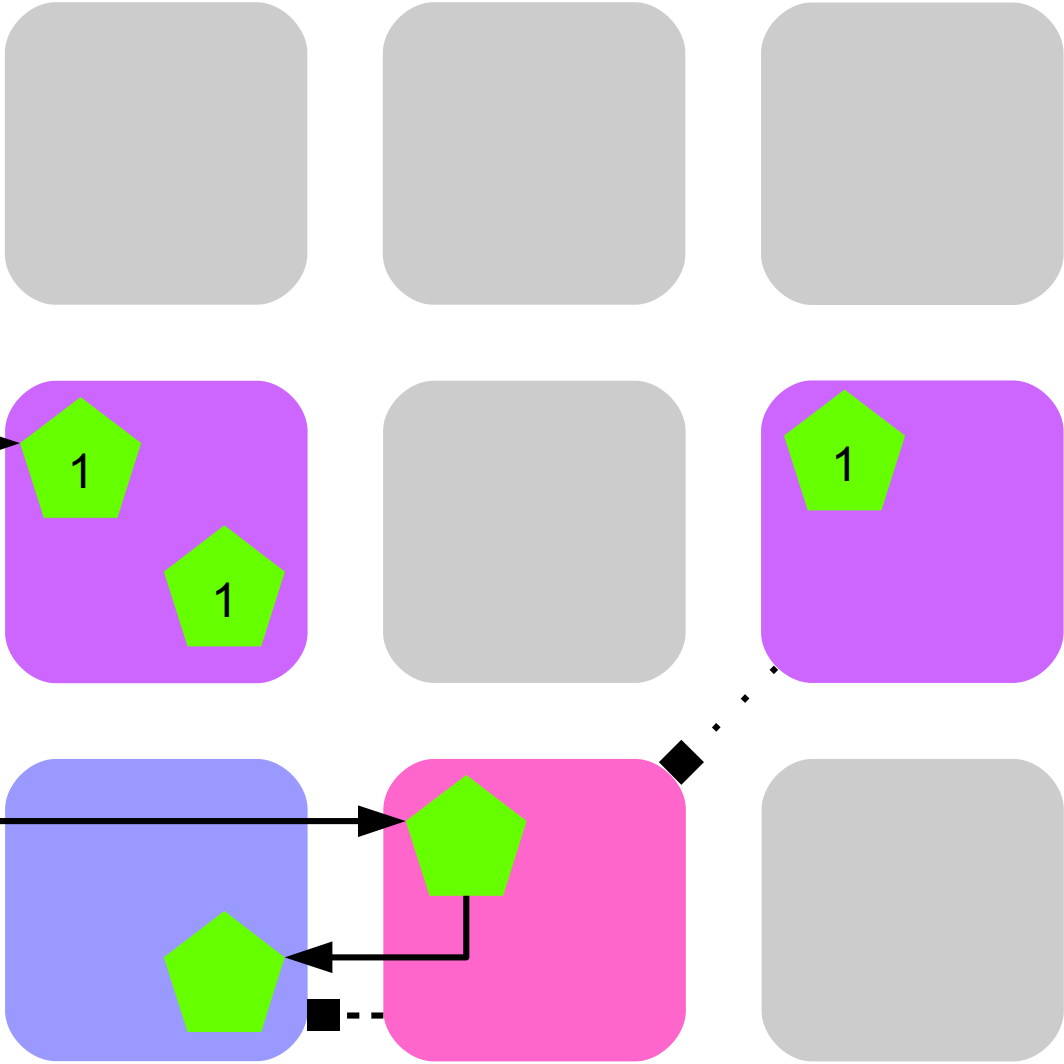
STATIC FIELDS

FREE

EDEN

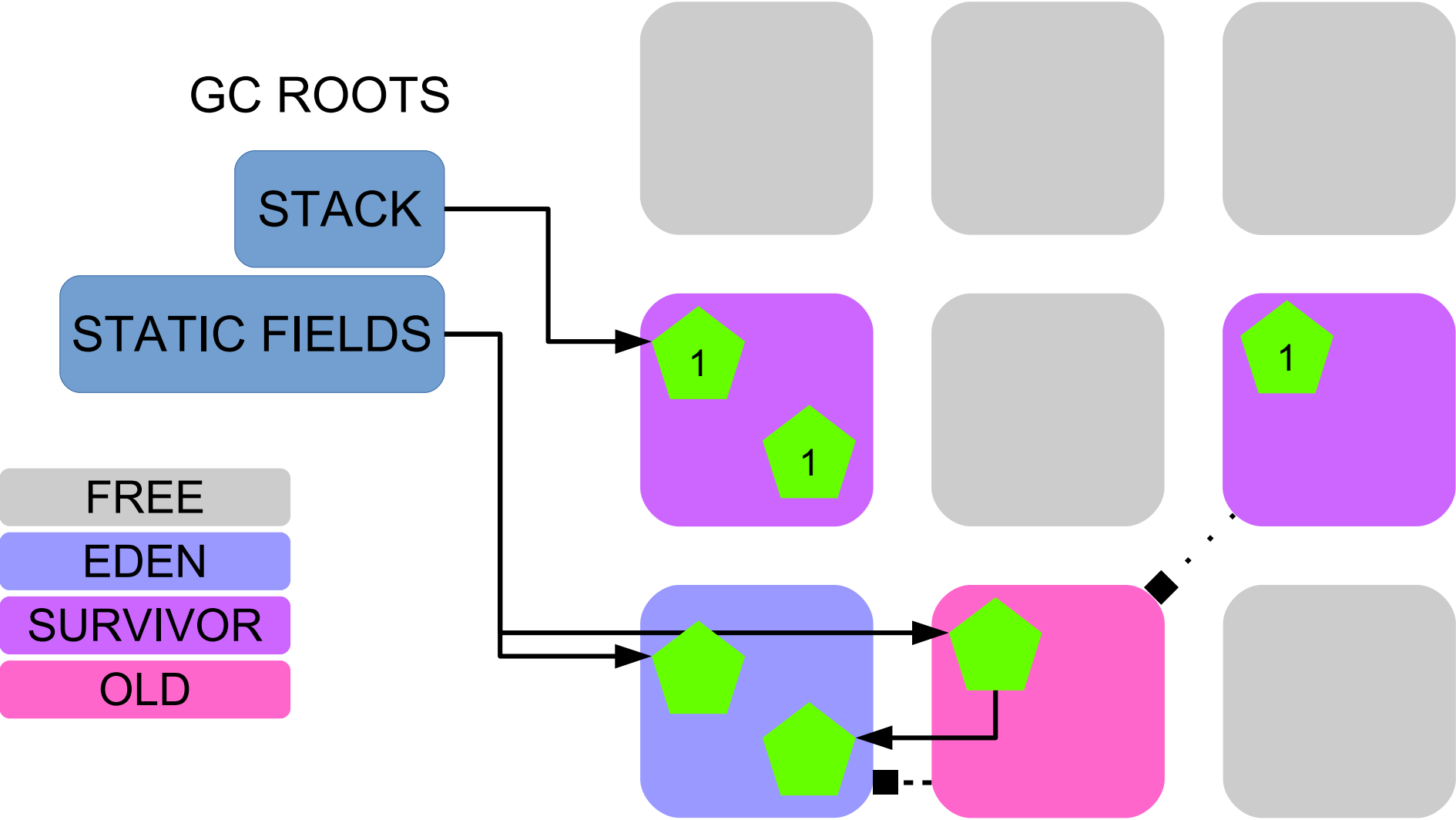
SURVIVOR

OLD



Update RSet: old->young

# RUNNING



GC ROOTS

STACK

STATIC FIELDS

FREE

EDEN

SURVIVOR

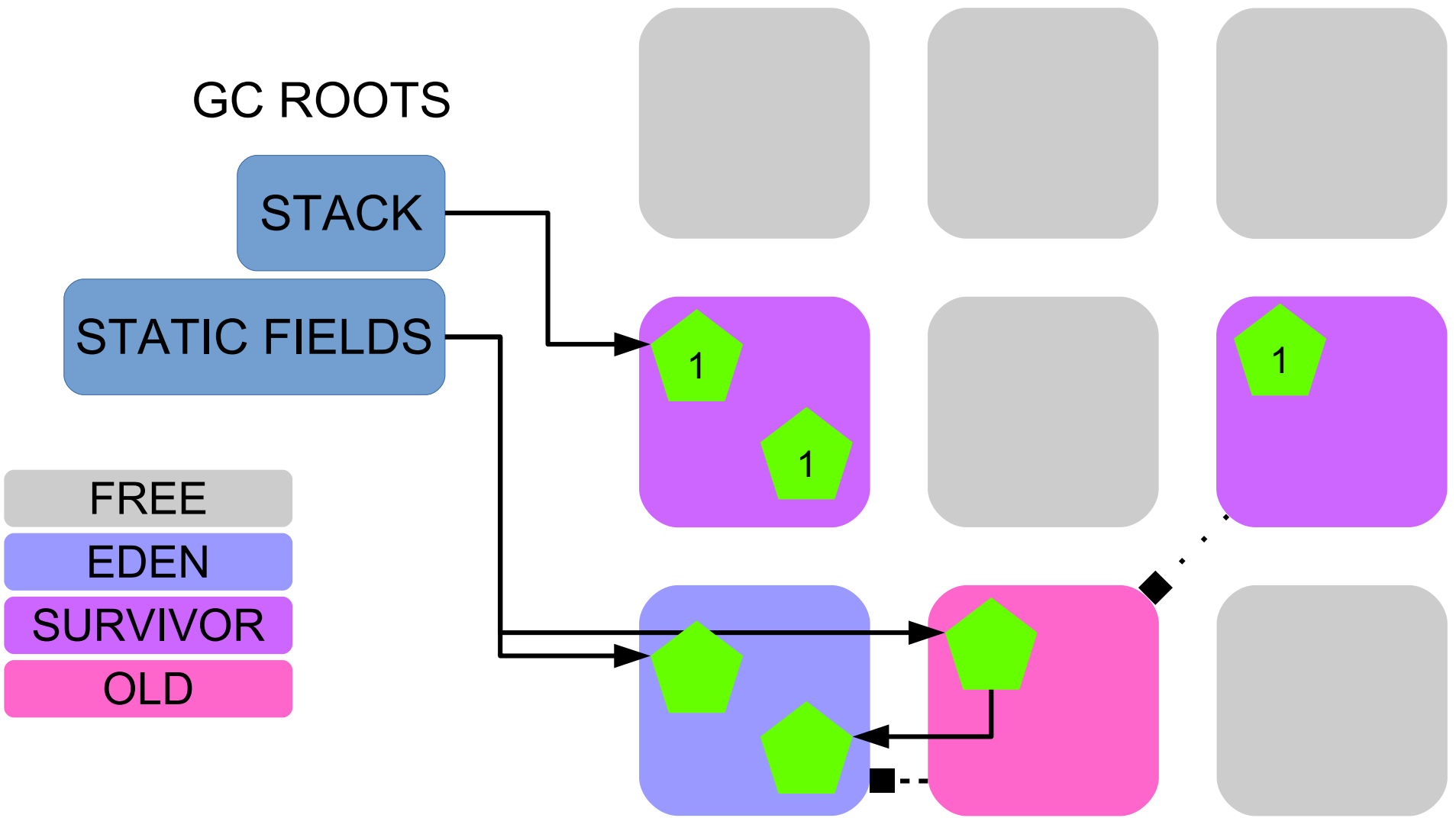
OLD

1

1

1

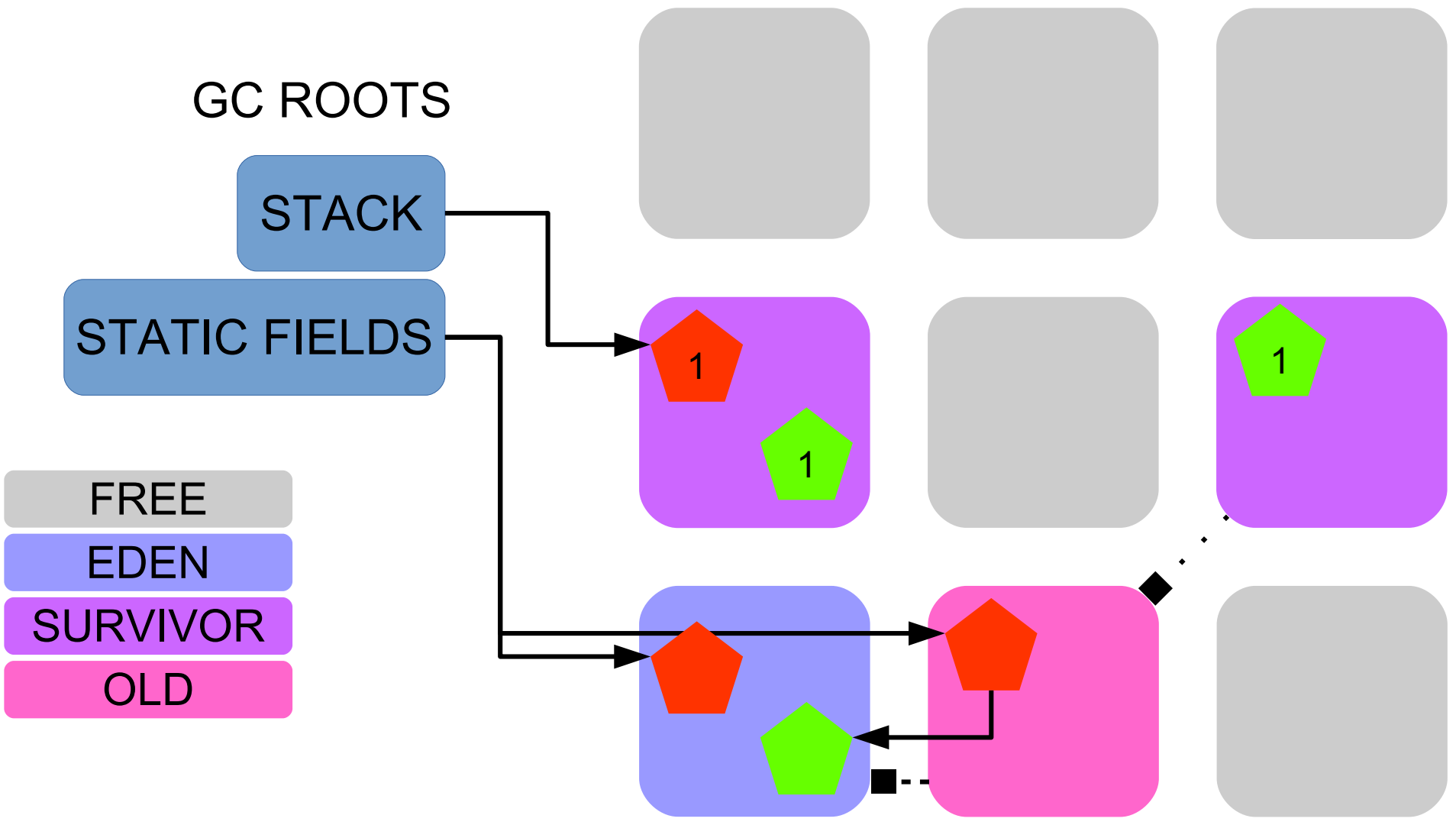
# STOP THE WORLD



[GC pause (G1 Evacuation Pause) (young) ..]  
Collection Set: all young

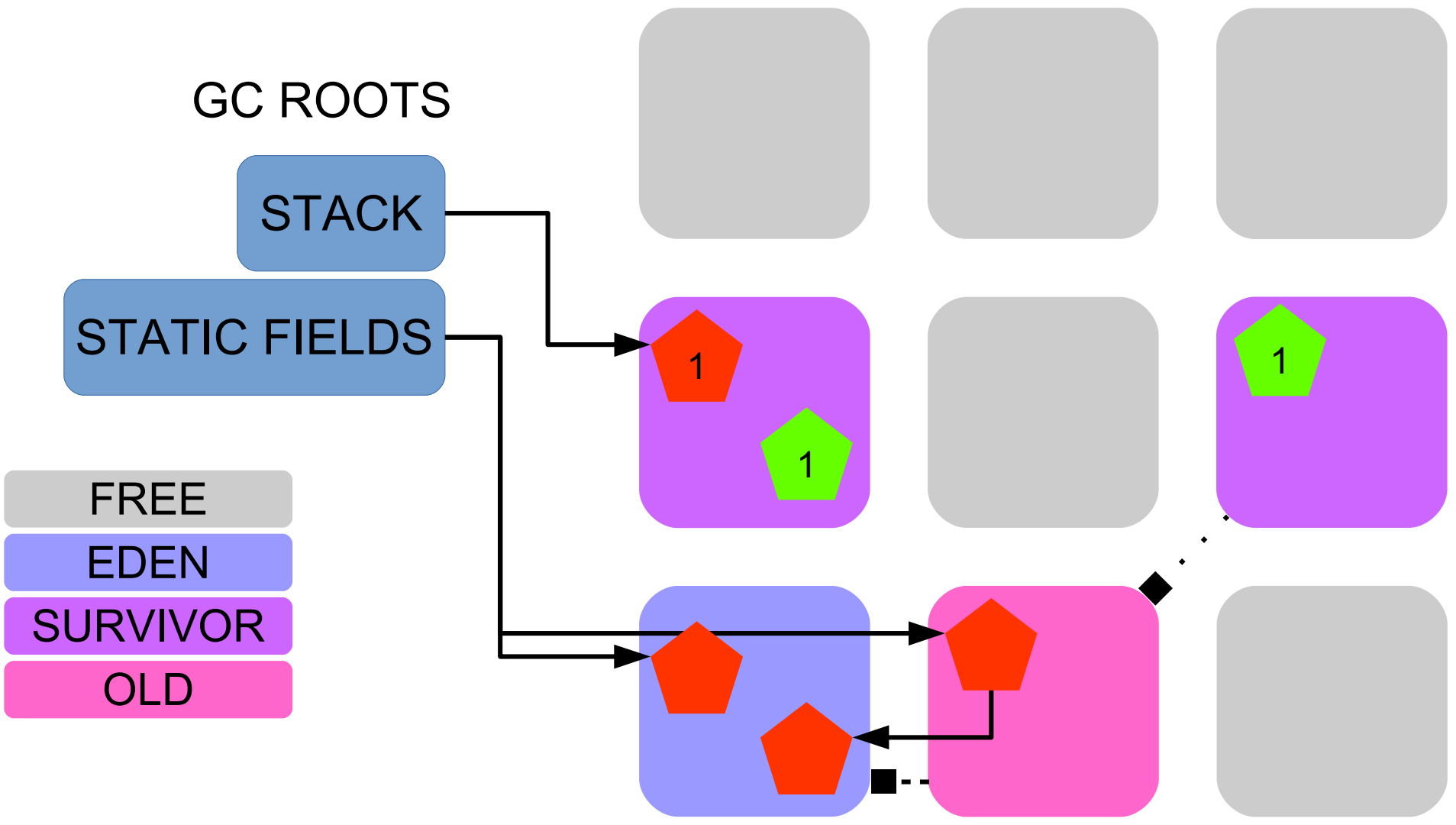


# STOP THE WORLD



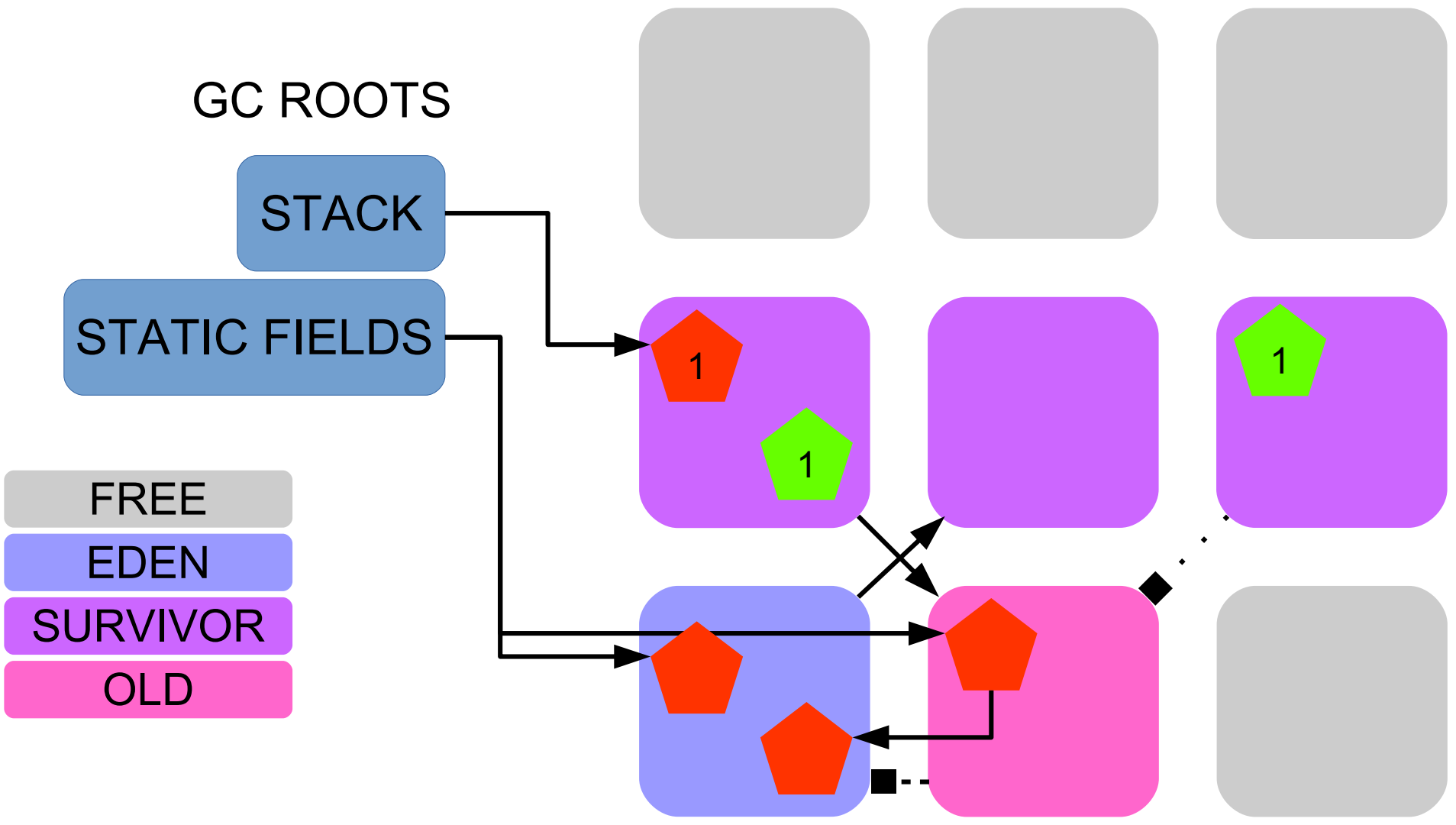
Mark objects reachable from roots

# STOP THE WORLD



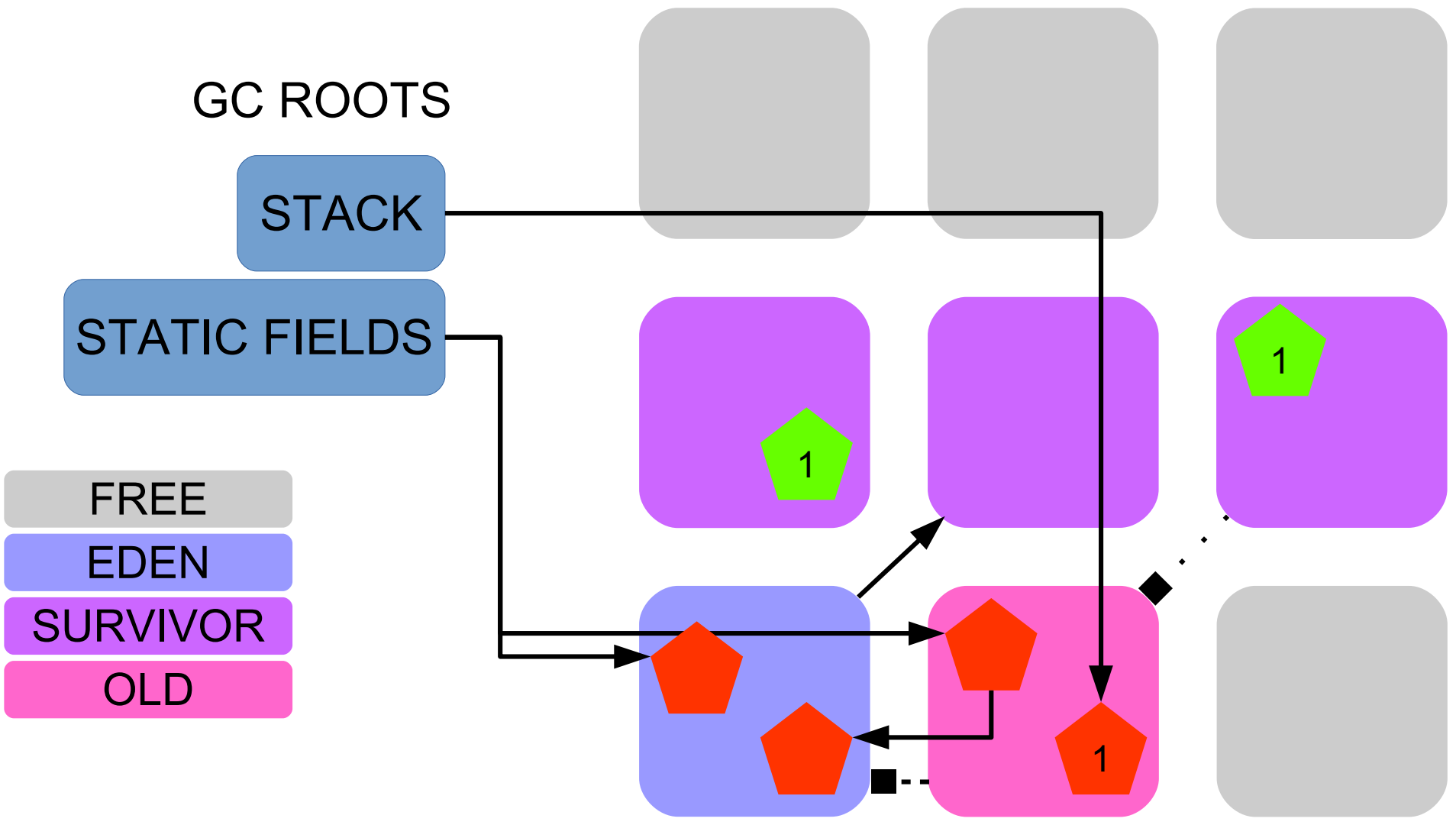
Mark objects reachable from roots and RSets

# STOP THE WORLD



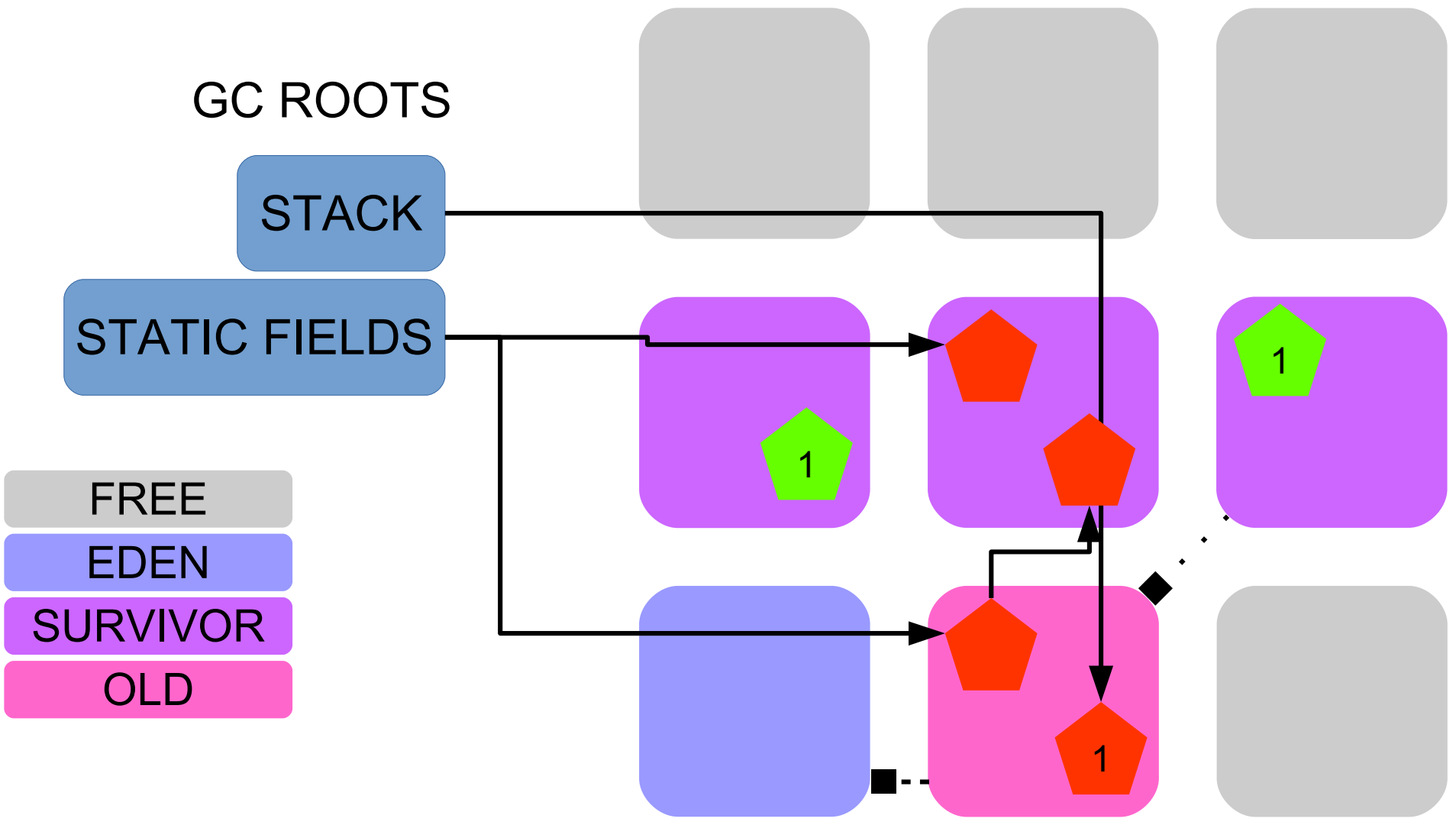
Move to new survivor region / old

# STOP THE WORLD



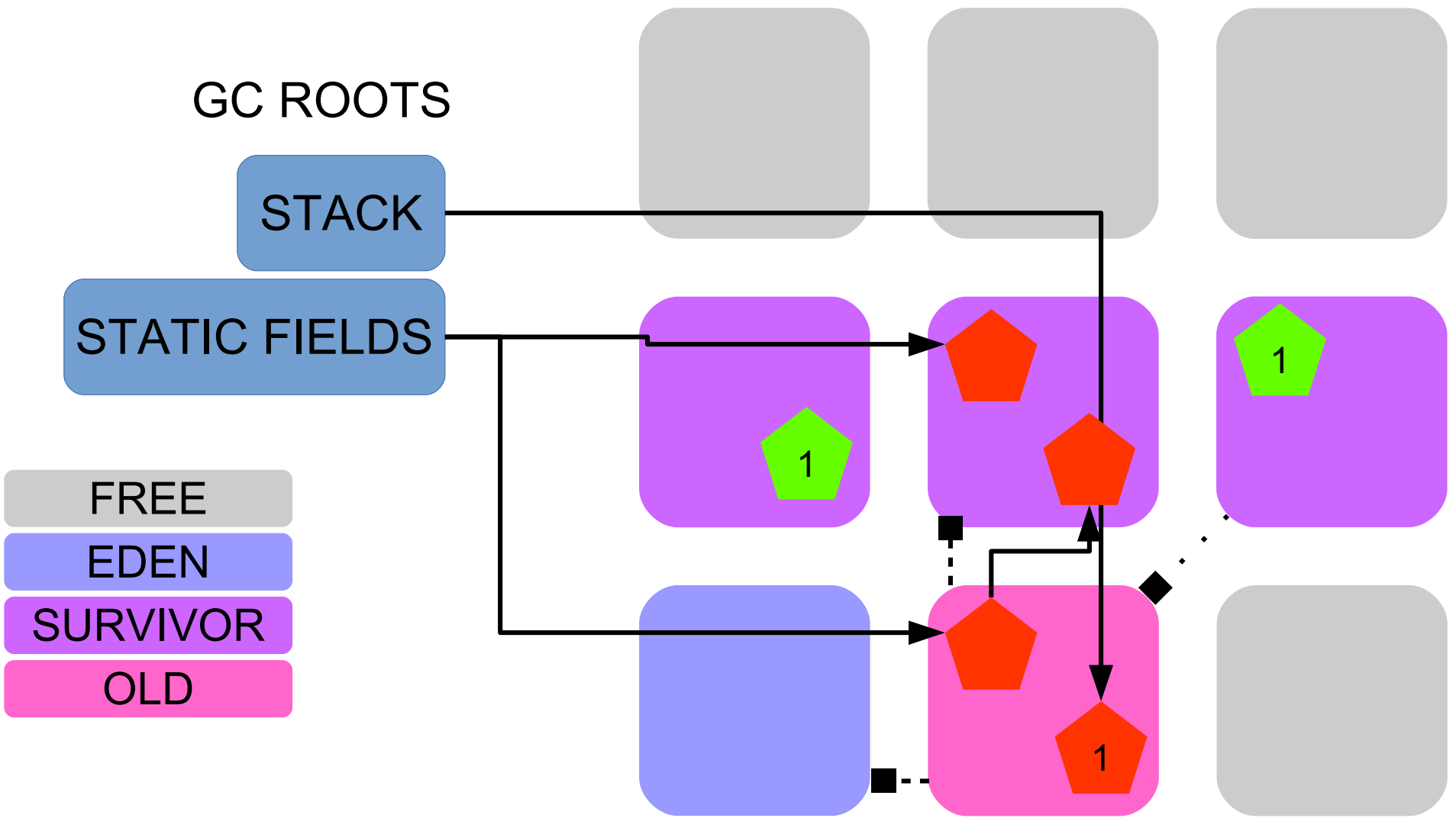
Move to new survivor region / old

# STOP THE WORLD



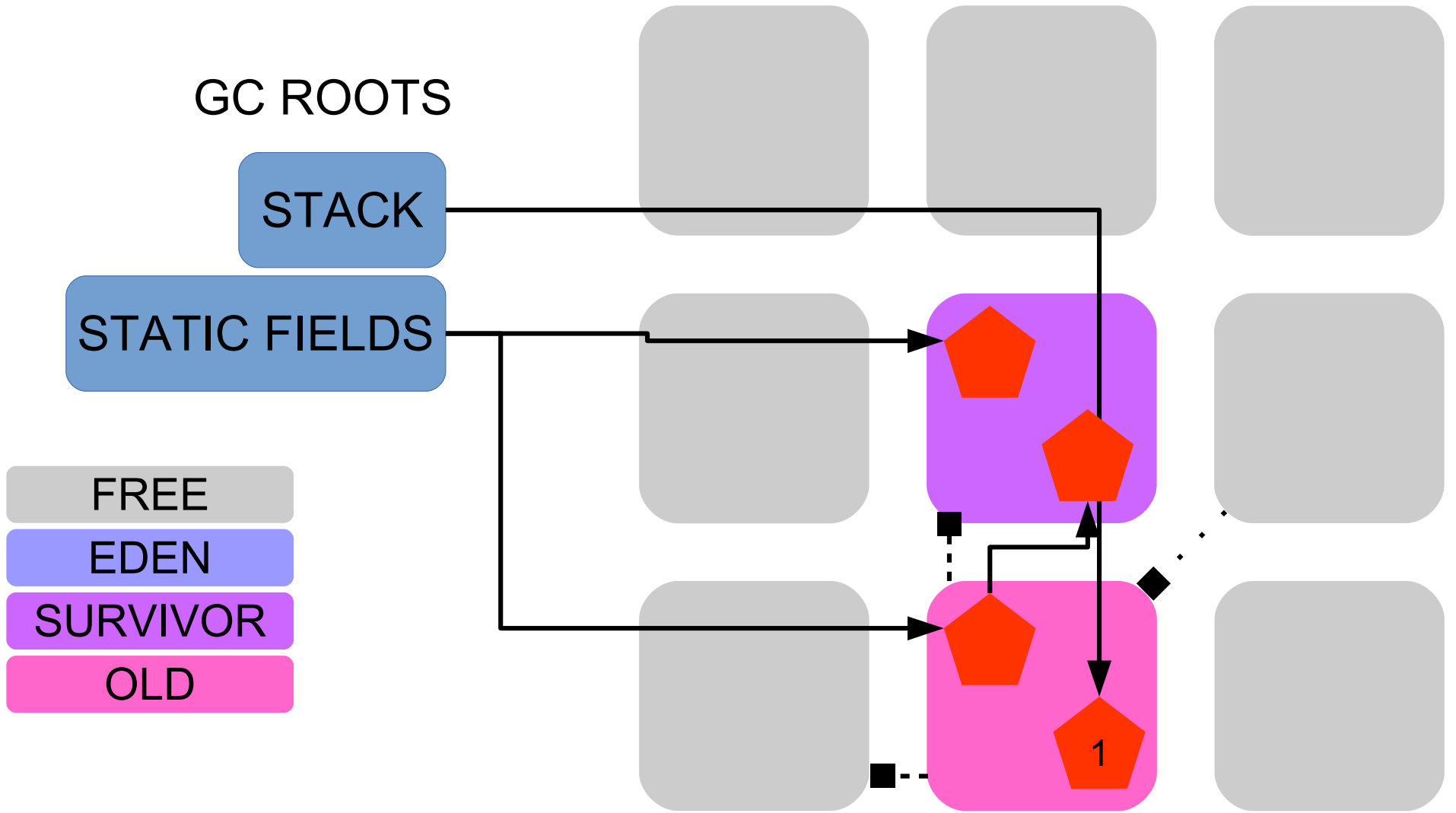
Move to new survivor region / old

# STOP THE WORLD



Update RSet: old->young

# STOP THE WORLD



Free evacuated regions

# RUNNING

GC ROOTS

STACK

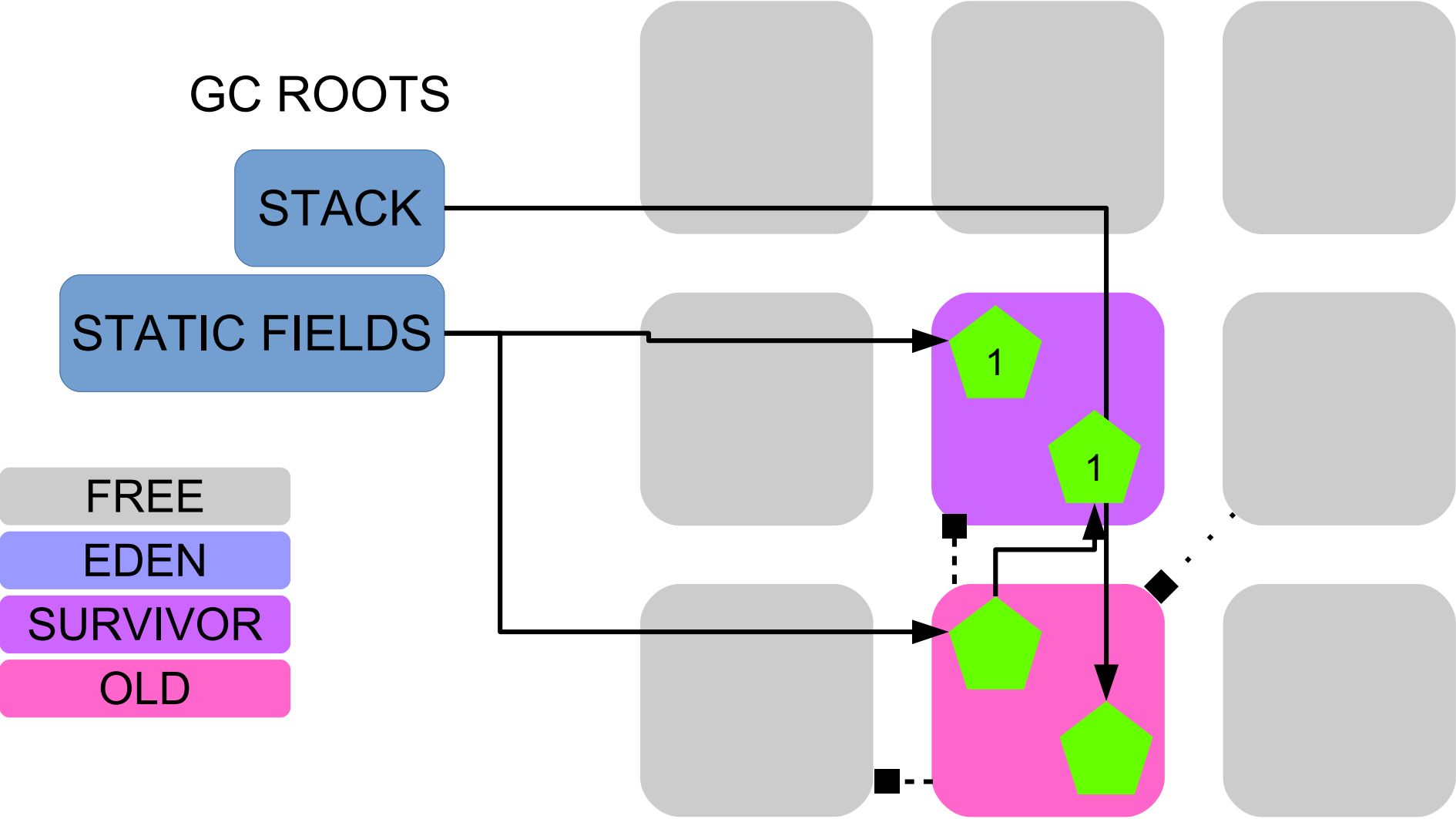
STATIC FIELDS

FREE

EDEN

SURVIVOR

OLD





# Meanwhile..

- Concurrent marking
- Start from GC roots
- Find all live objects
- Sort regions by “liveness”

# After concurrent marking

- Stop The World
- Scrub RSets
- Collection set: all young  
+ Old with least live objects
- 1.269: [GC pause (**mixed**) ... ]
- Amount of old regions selected ~ pause time

# Recap: G1

- Use concurrent marking
- Collection set: all young  
+ hand picked old (most garbage)
- Find live objects from GC roots + RSets
- Compact to new survivors / old regions
- Free entire evacuated regions

# G1 insights

- Avoid reference scanning with RSets
- Avoid long pauses with mixed collection:  
never clean entire Old Gen
- Only collect Old regions with most garbage
  - -> don't touch live Old objects
  - -> more time to become garbage
  - Fewer live objects -> less copying

# Homework (setup)

- <https://github.com/shelajev/jf-hw-jmm>
- Implement post-GC task execution:
  - Implement methods marked with `//TODO`

# Homework (1)

- Use PhantomReferences to write a finalize() replacement
- User can register several cleanup tasks for any object
- PostCollectionTaskRunner runs cleanup tasks in the background

```
interface PostCollectionTaskRunner {  
    void register(Object o, Runnable  
task);  
    void shutdown() throws Exception;  
}
```

# Homework (2)

- Run with ParallelGC (Java 8 default).  
Enable detailed GC logging. Cause a Full GC.
  - Use file: **gclog.log**
- **Submit the GC log + following comments:**
  - for one minor collection: time since last collection and bytes freed for young gen
  - for one Full GC: bytes freed for young gen, old gen, total heap size
  - Use file: **gccomments.txt**
- **Use max heap size 64M**
- Submit code for triggering Full GC

# Homework (3)

- Phantoms expensive ->  
    use 1 per managed object
- Don't keep stuff for dead objects
- Reasonably efficient code:  
    no Thread#sleep, crazy list iterations, etc.
- shutdown() -> stop thread, clear data
- Deadline: 09 Apr 23:59 local time



# Read more..

- <https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/>
- <https://plumbr.eu/handbook/garbage-collection-algorithms-implementations>
- <https://vimeo.com/181948157>
- <https://stackoverflow.com/q/19154607>
- <http://insightfullogic.com/2013/Mar/06/garbage-collection-java-2/>