# INVESTIGATING HIGH MEMORY CHURN VIA OBJECT LIFETIME ANALYSIS TO IMPROVE SOFTWARE PERFORMANCE

**SSP 2020**

**Markus Weninger**, Elias Gander, Hanspeter Mössenböck

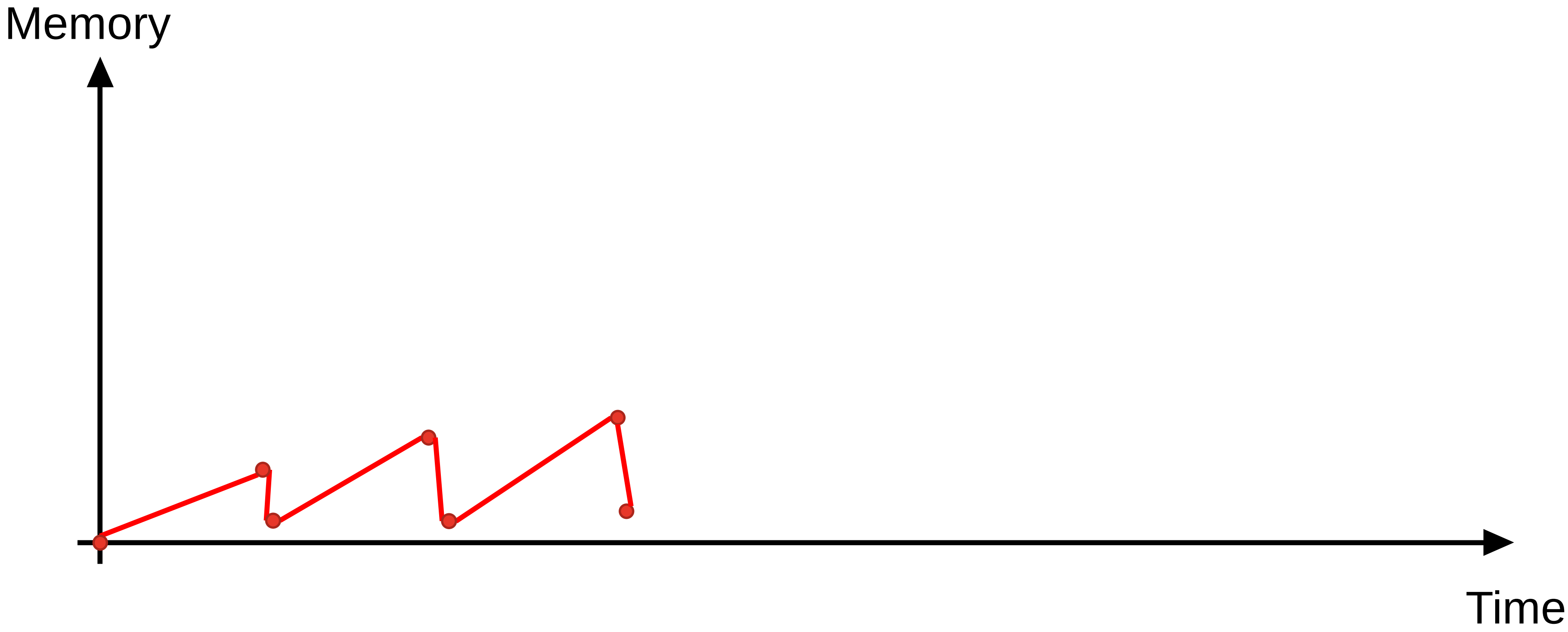
*Johannes Kepler University Linz, Austria*

*Institute for System Software, Christian Doppler Laboratory MEVSS*

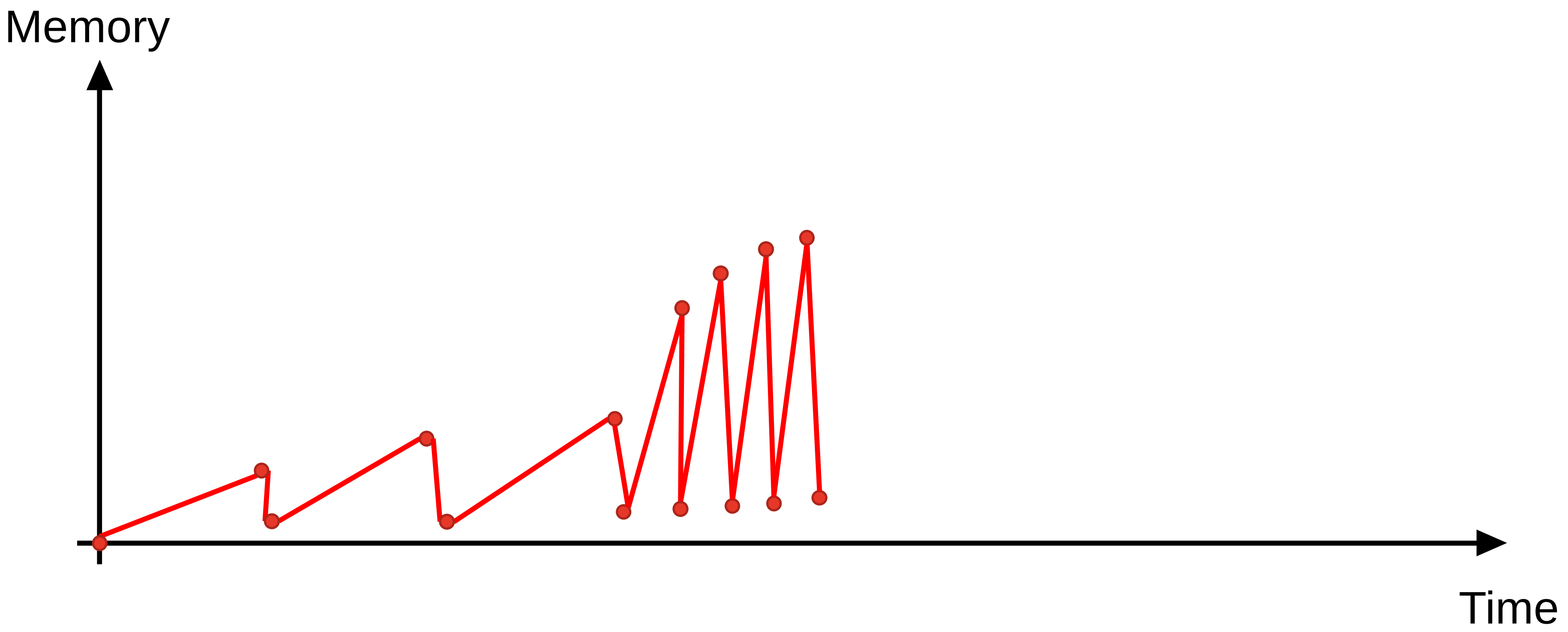
JOHANNES KEPLER UNIVERSITY LINZ

# MOTIVATION: MEMORY ANOMALIES

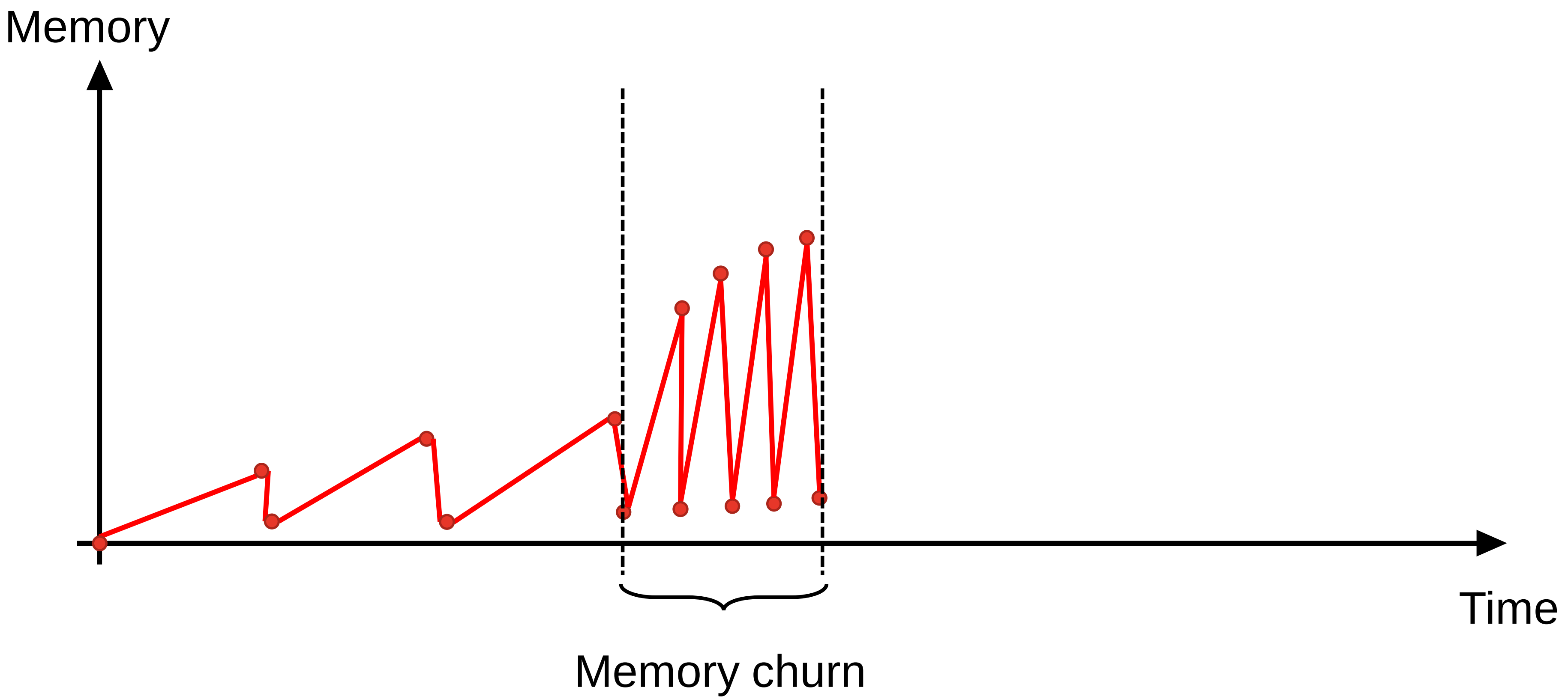
# MOTIVATION: MEMORY ANOMALIES



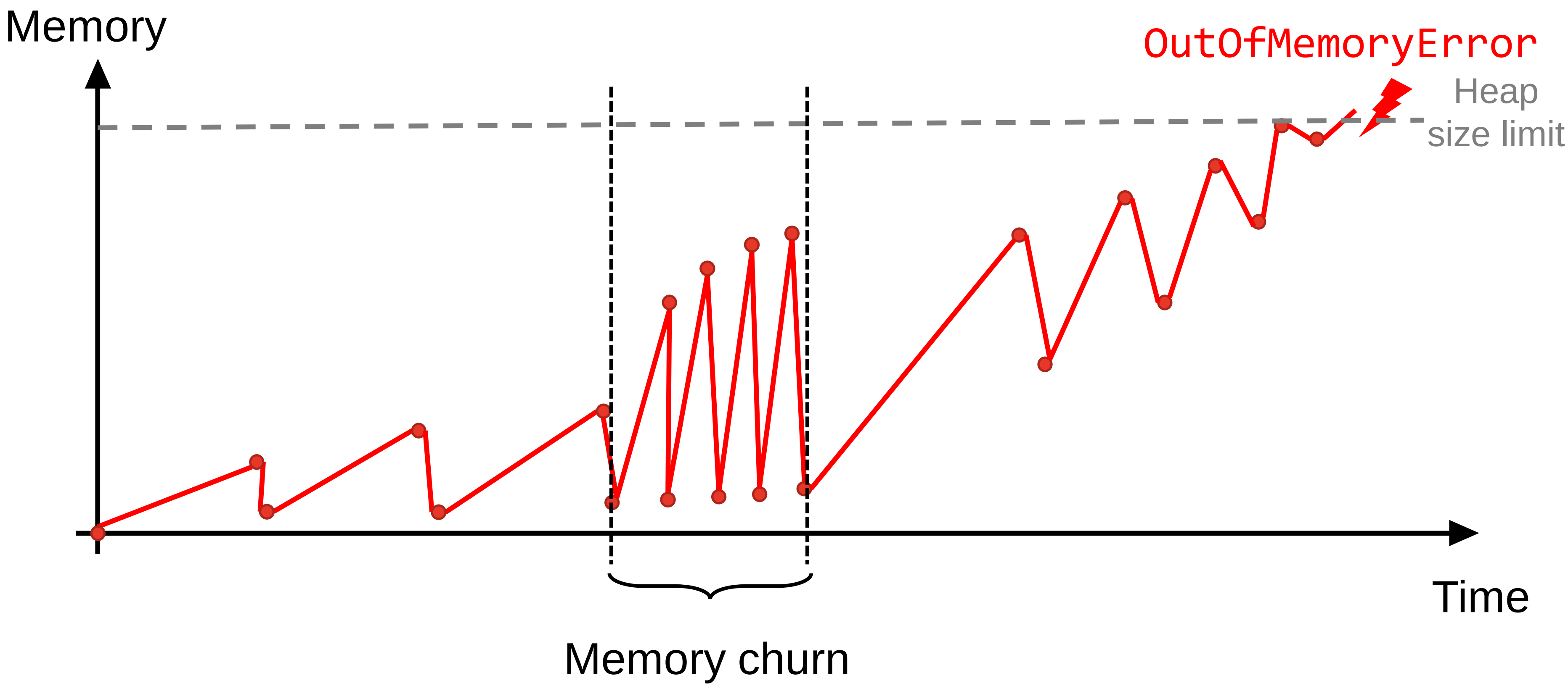
# MOTIVATION: MEMORY ANOMALIES



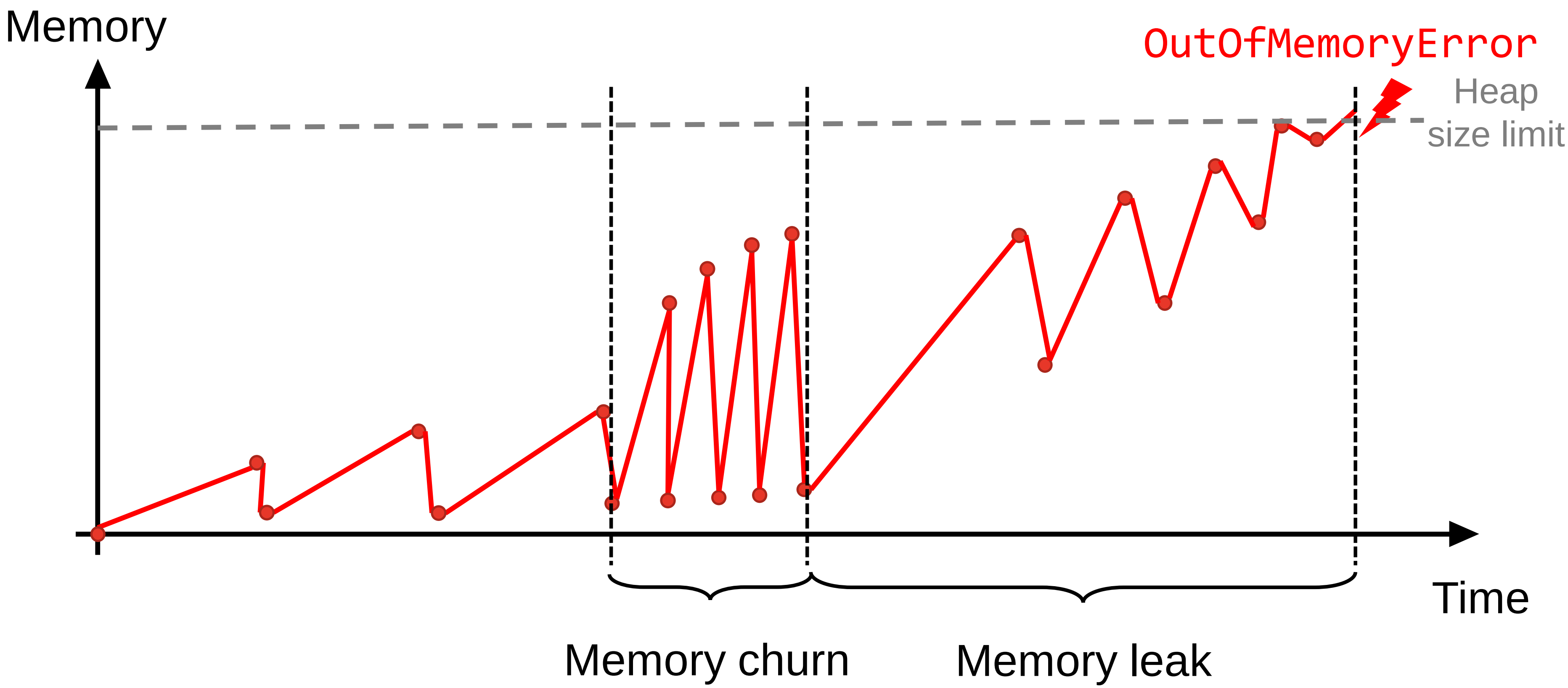
# MOTIVATION: MEMORY ANOMALIES



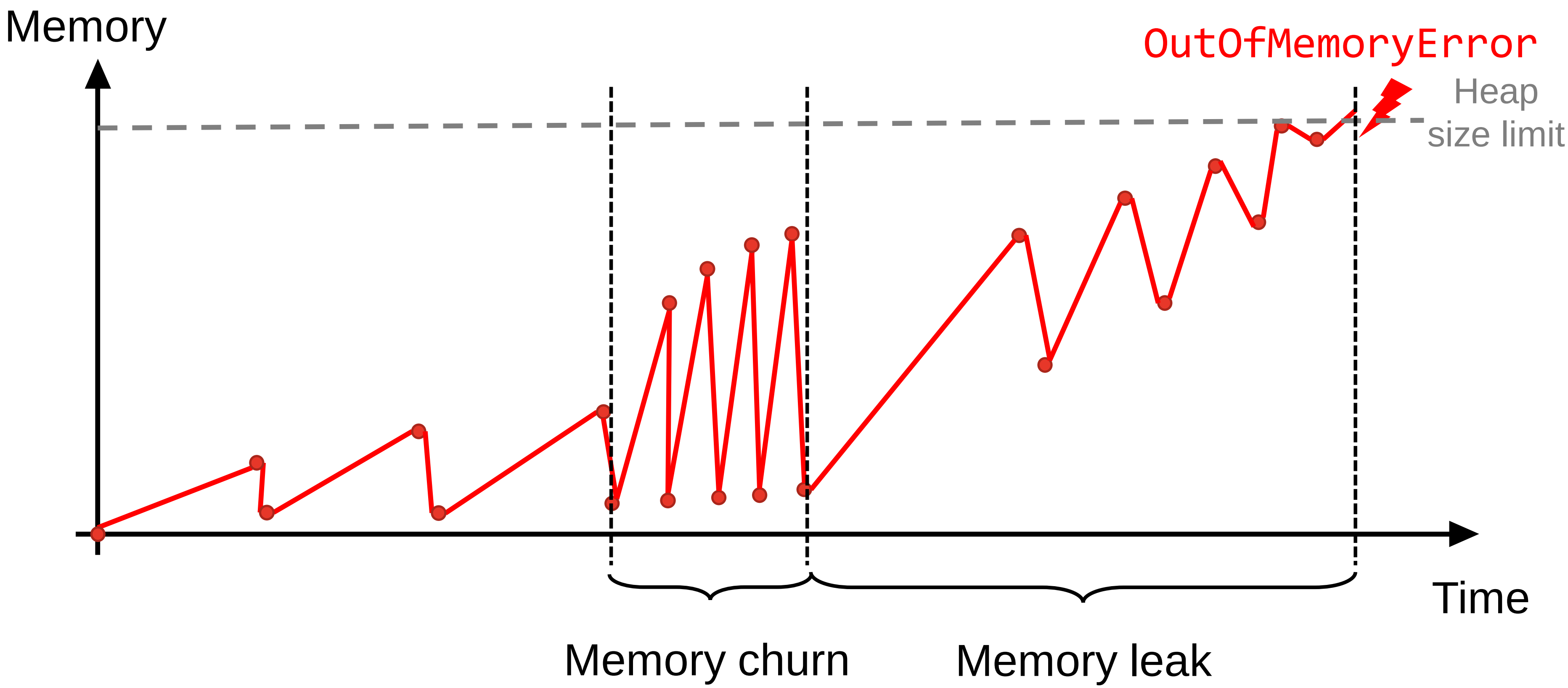
# MOTIVATION: MEMORY ANOMALIES



# MOTIVATION: MEMORY ANOMALIES

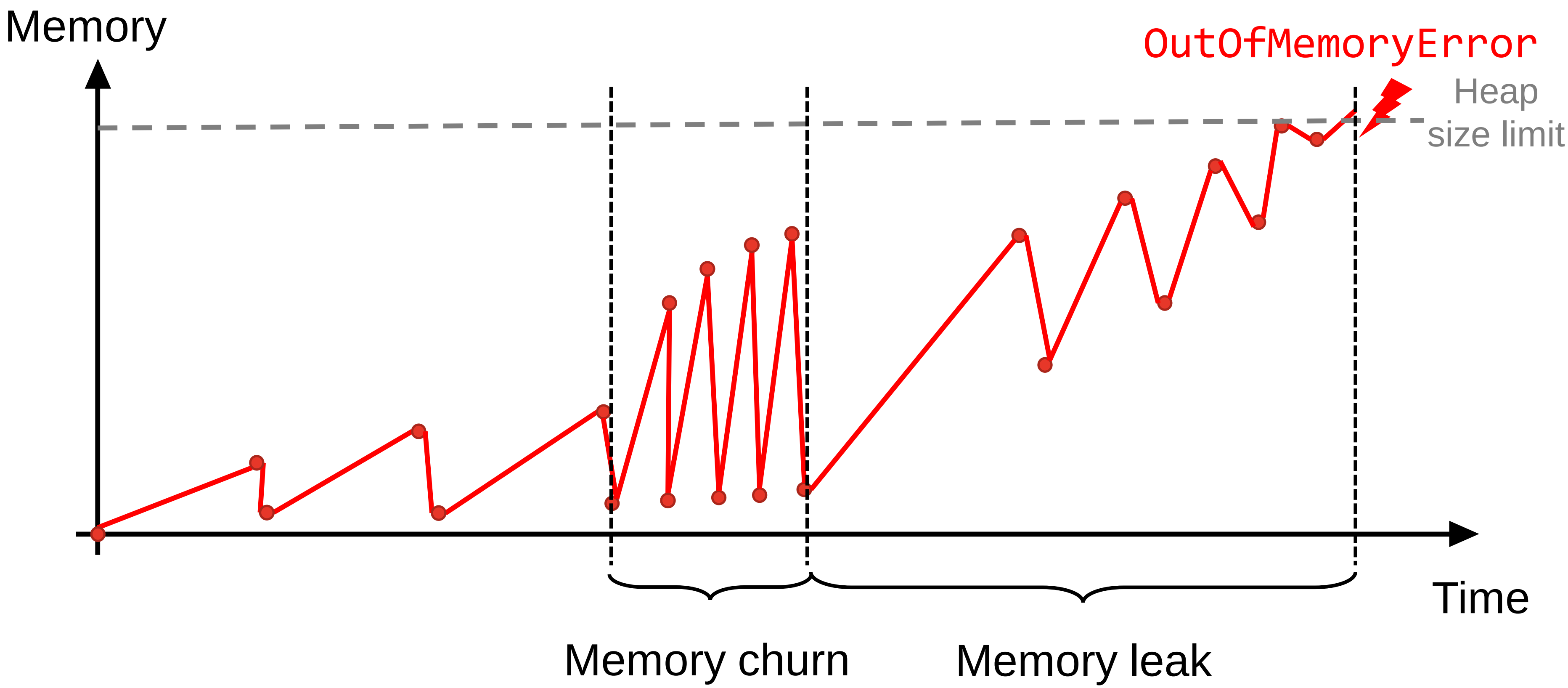


# MOTIVATION: MEMORY ANOMALIES



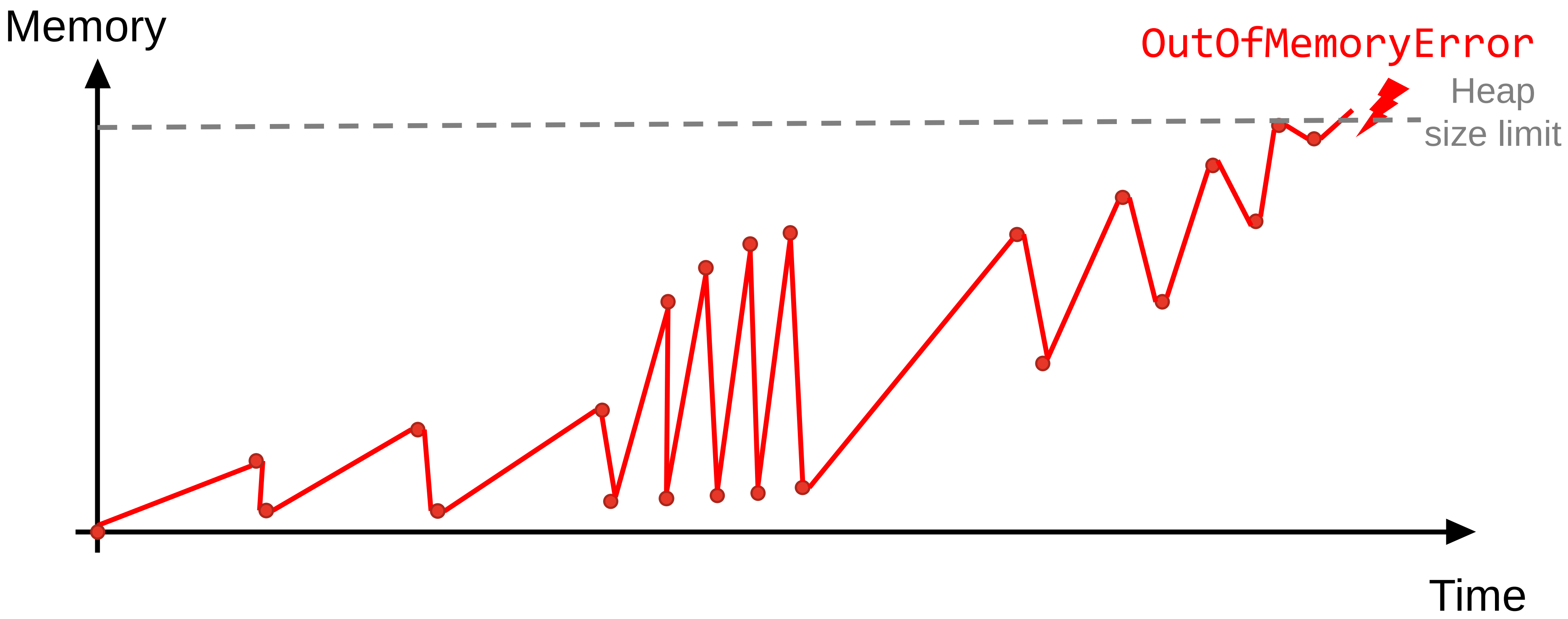

**Investigate!**

# MOTIVATION: MEMORY ANOMALIES

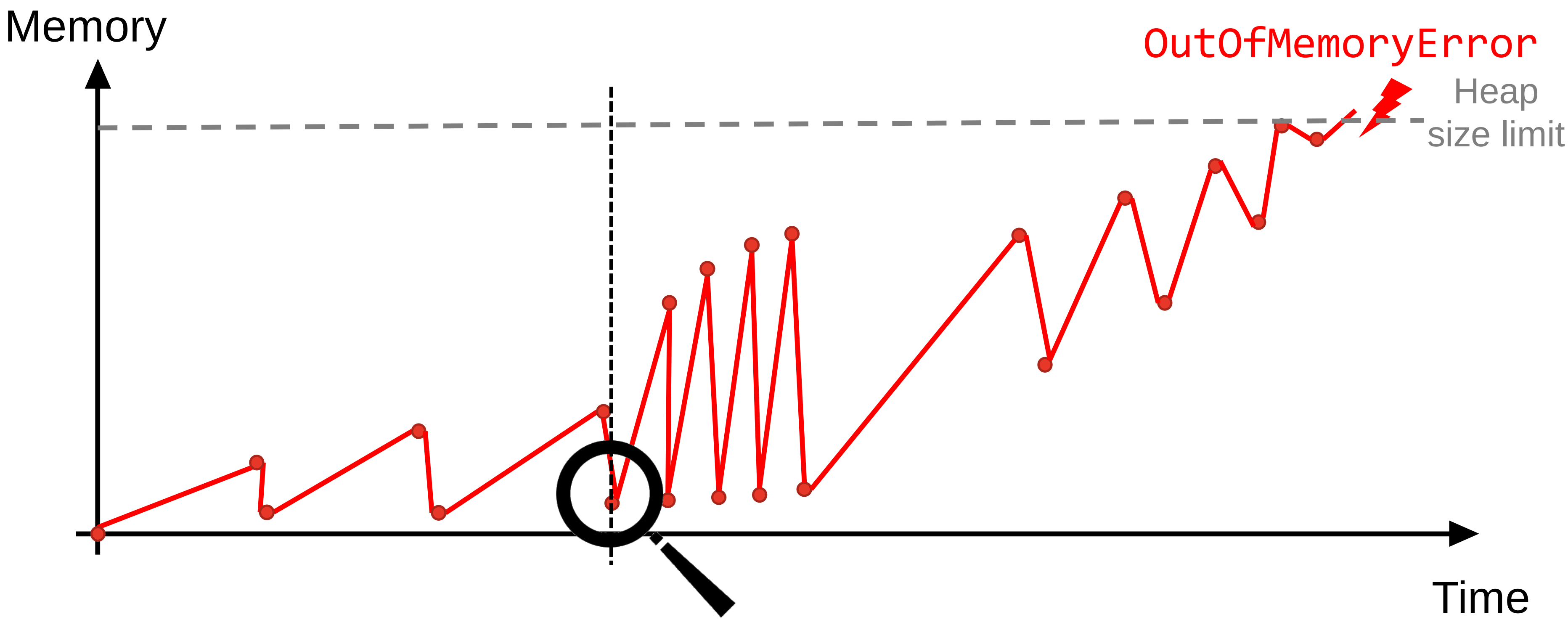



**How?**

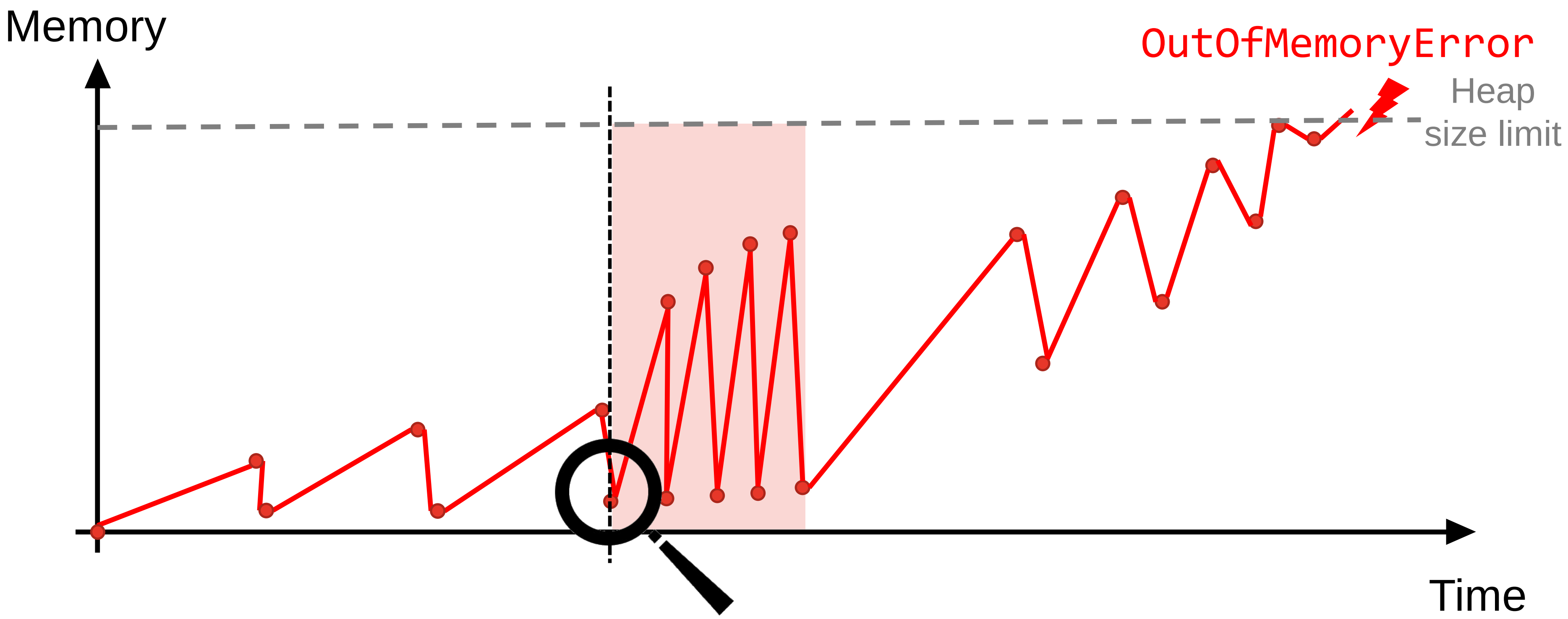
# MOTIVATION: MEMORY ANOMALIES



# MOTIVATION: MEMORY ANOMALIES

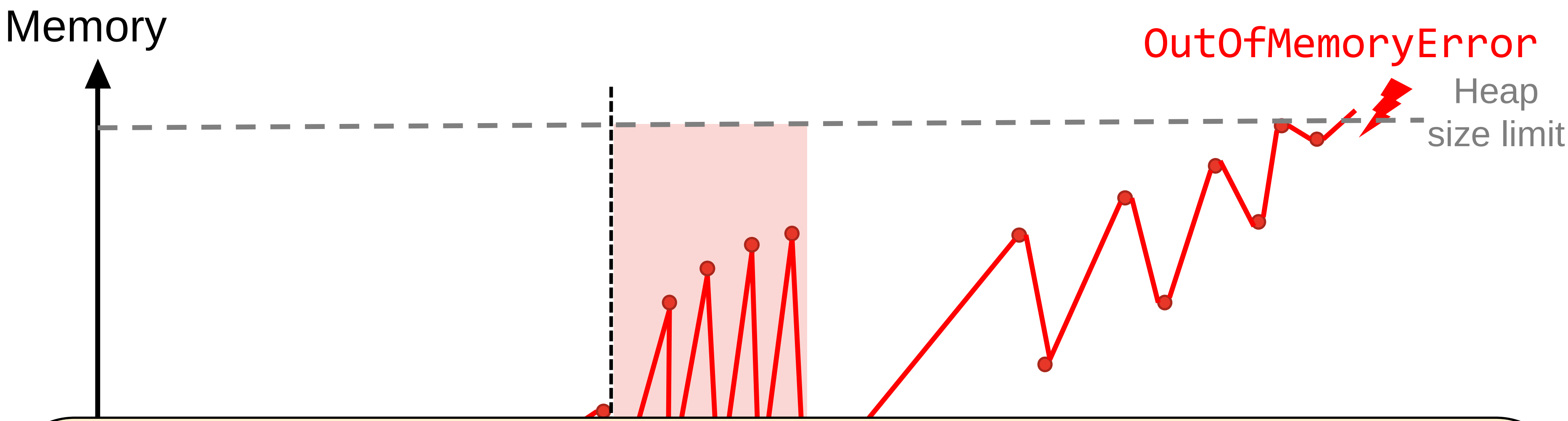


# MOTIVATION: MEMORY ANOMALIES



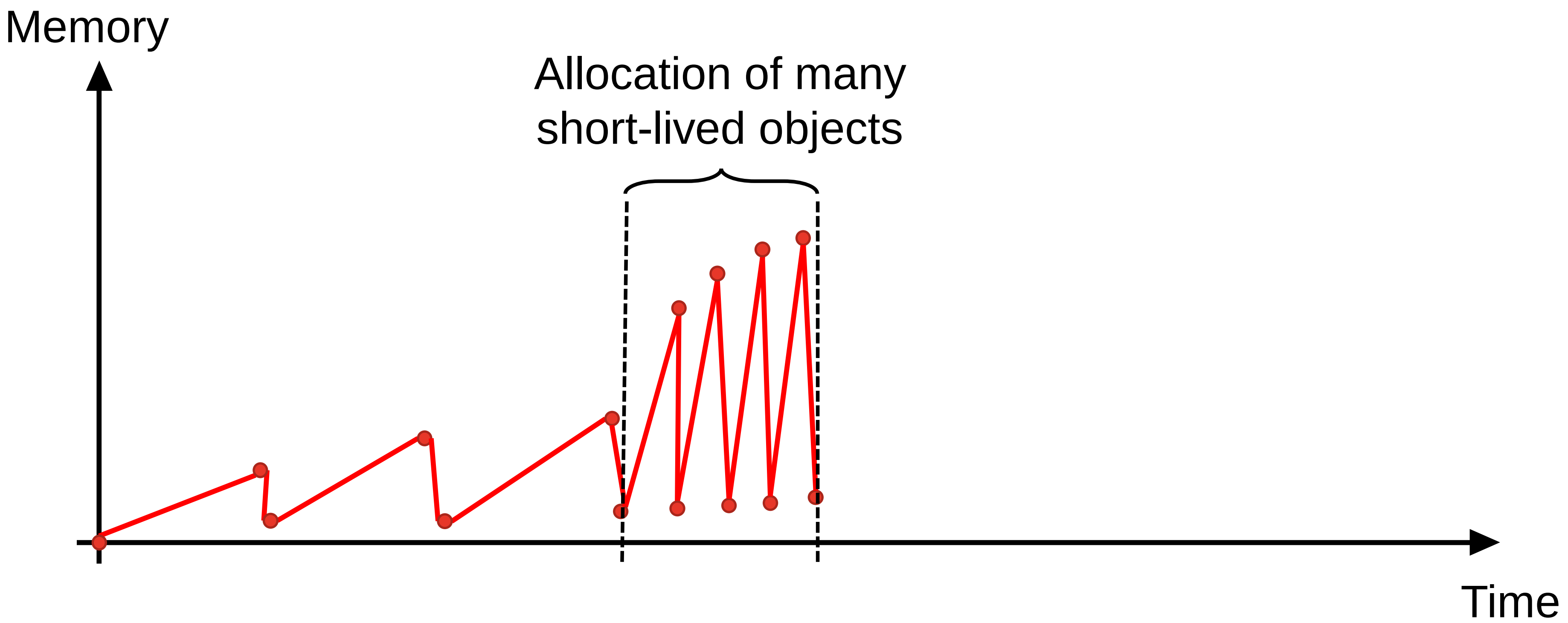

JKU

# MOTIVATION: MEMORY ANOMALIES

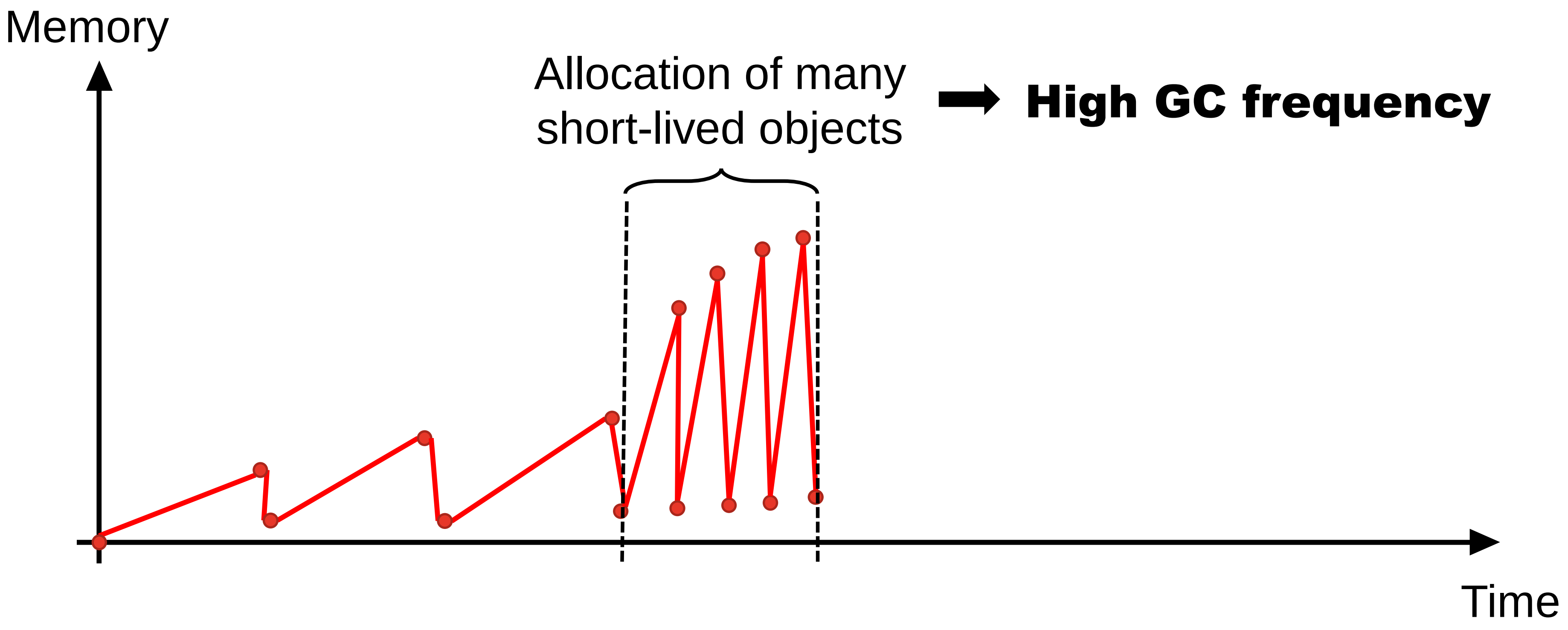



**Idea:** Highlight ***memory churn hotspots*** and provide ***information*** on objects that generate the ***most garbage***.

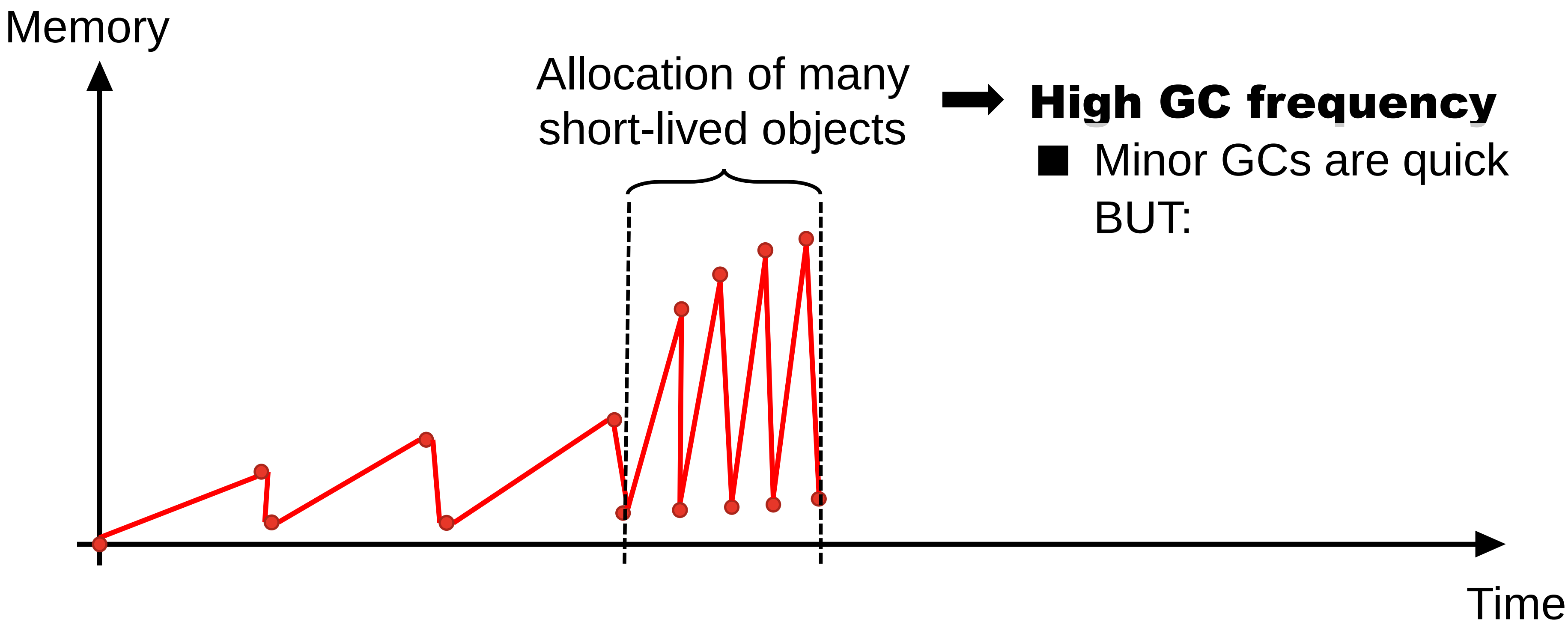
# MEMORY CHURN



# MEMORY CHURN



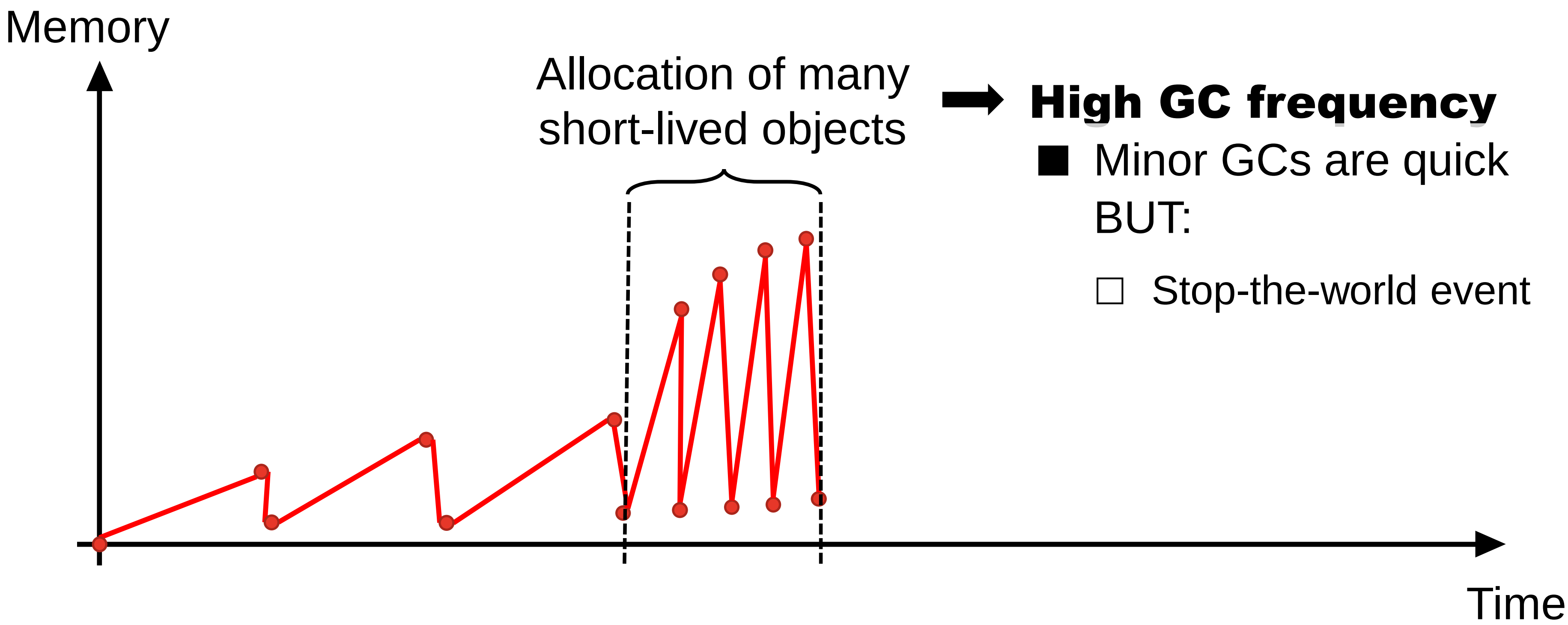
# MEMORY CHURN




➡ **High GC frequency**

- Minor GCs are quick
  BUT:

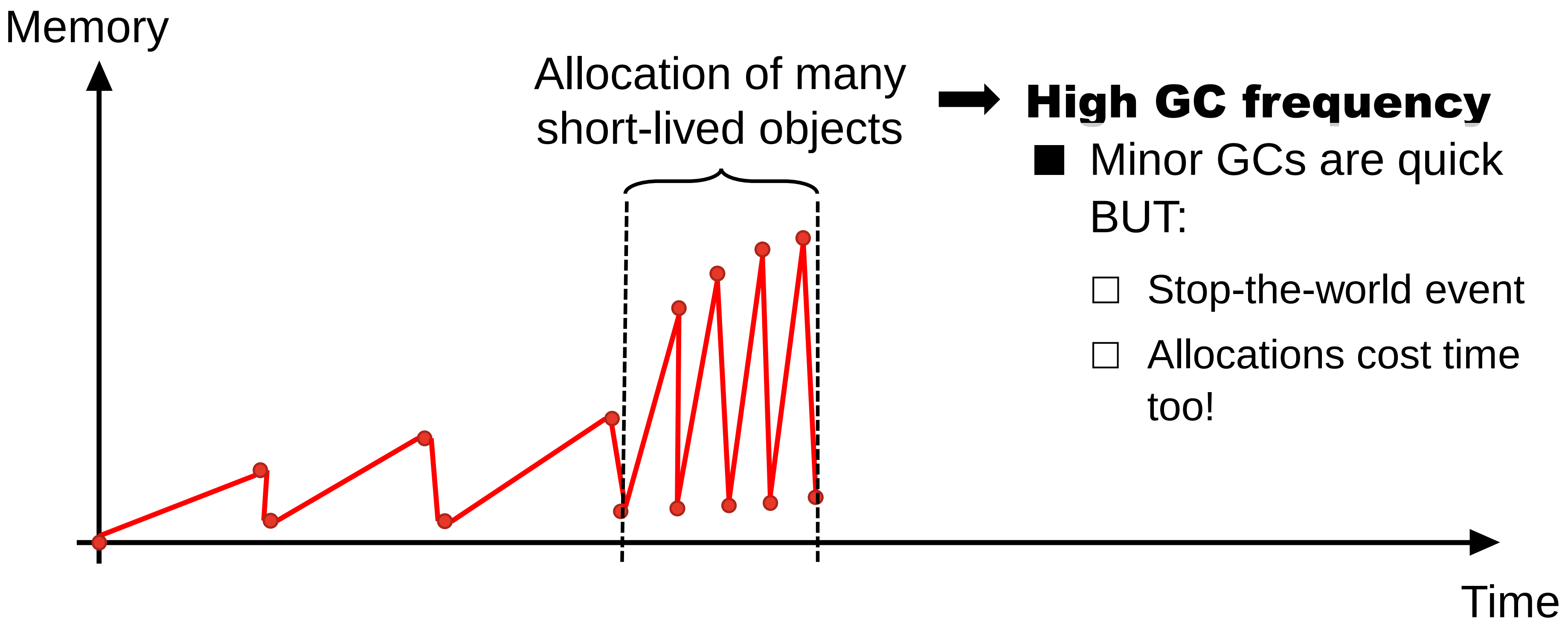
# MEMORY CHURN




➡ **High GC frequency**

- Minor GCs are quick BUT:
  - Stop-the-world event

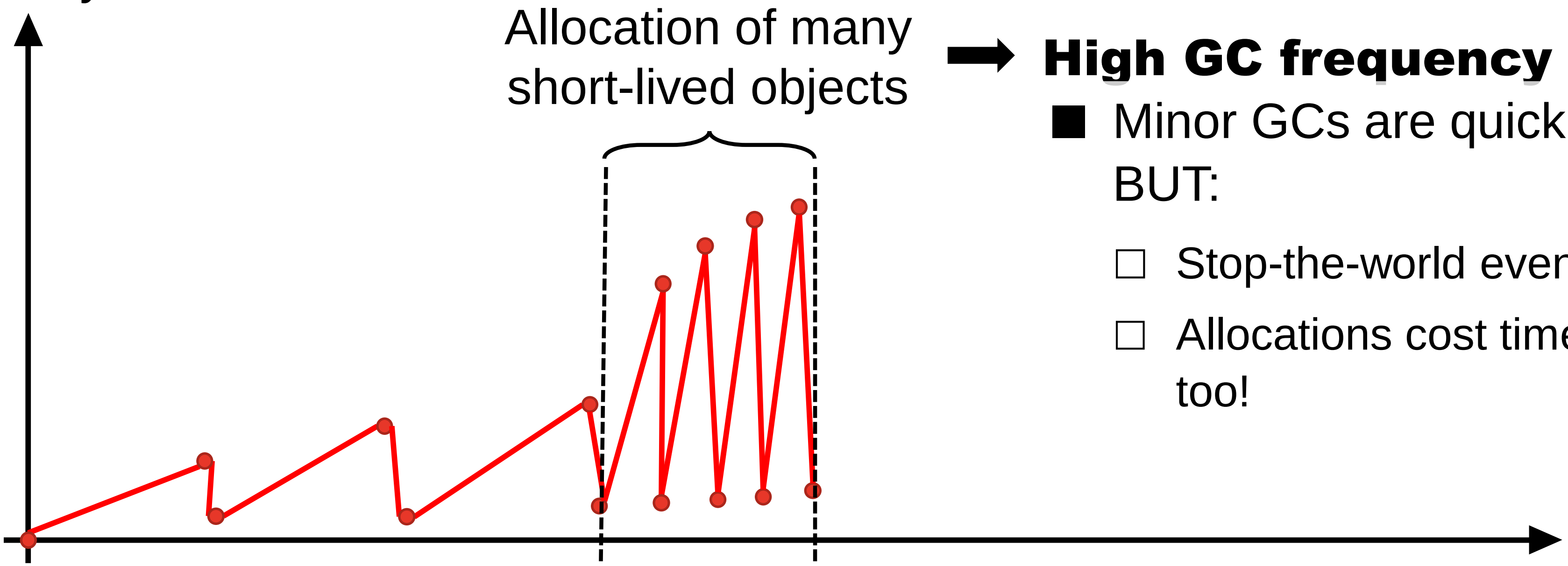
# MEMORY CHURN




➡ **High GC frequency**

- Minor GCs are quick BUT:
  - Stop-the-world event
  - Allocations cost time too!

# MEMORY CHURN

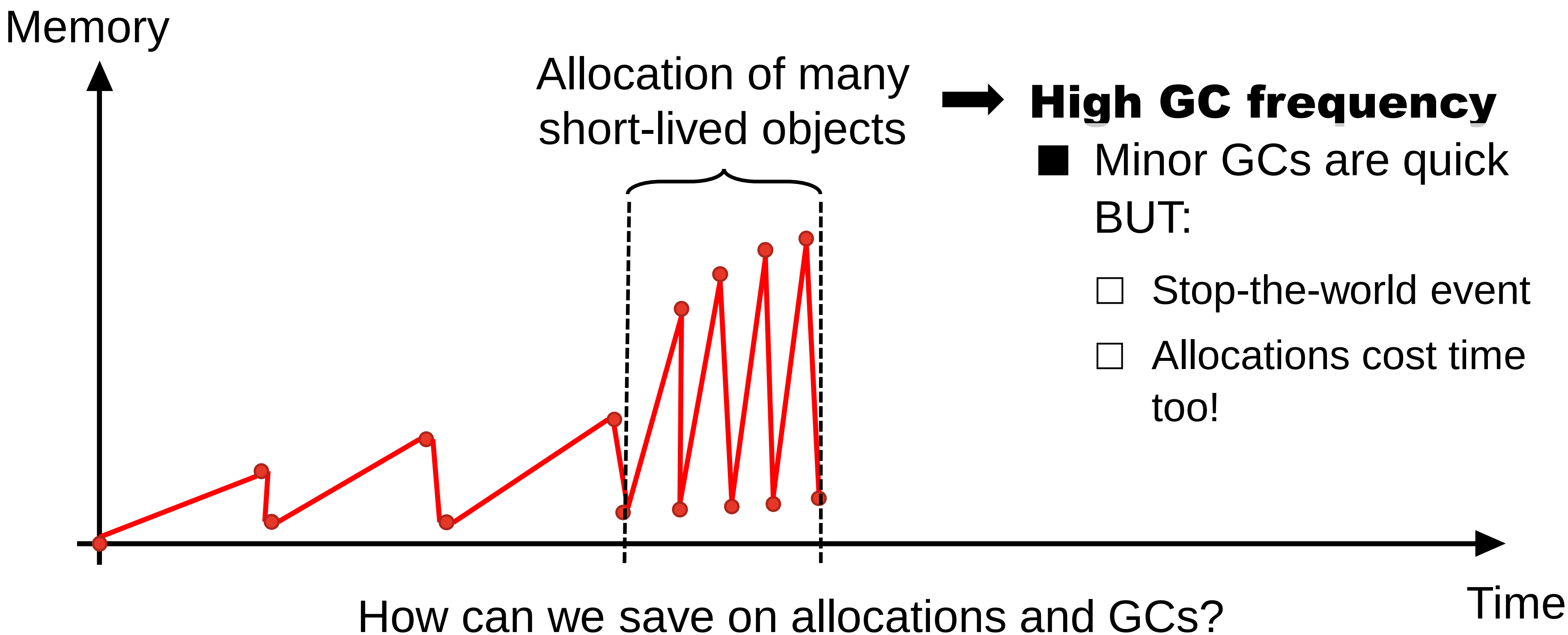



➡ **High GC frequency**

- Minor GCs are quick BUT:
  - ☐ Stop-the-world event
  - ☐ Allocations cost time too!

How can we save on allocations and GCs?

# MEMORY CHURN




➡ **High GC frequency**

- Minor GCs are quick BUT:
  - Stop-the-world event
  - Allocations cost time too!

How can we save on allocations and GCs?

Find out which objects survive only few GCs and where they are allocated!

# REASONS FOR MEMORY CHURN

# REASONS FOR MEMORY CHURN

Allocations in heavily executed loops

# REASONS FOR MEMORY CHURN

Allocations in heavily executed loops

Boxed primitives
(e.g., `ArrayList<Integer>`)

# REASONS FOR MEMORY CHURN

Allocations in heavily executed loops

Boxed primitives
(e.g., `ArrayList<Integer>`)

Streams (multiple `map` operations, late `filter` operations, etc.)

JKU

# REASONS FOR MEMORY CHURN

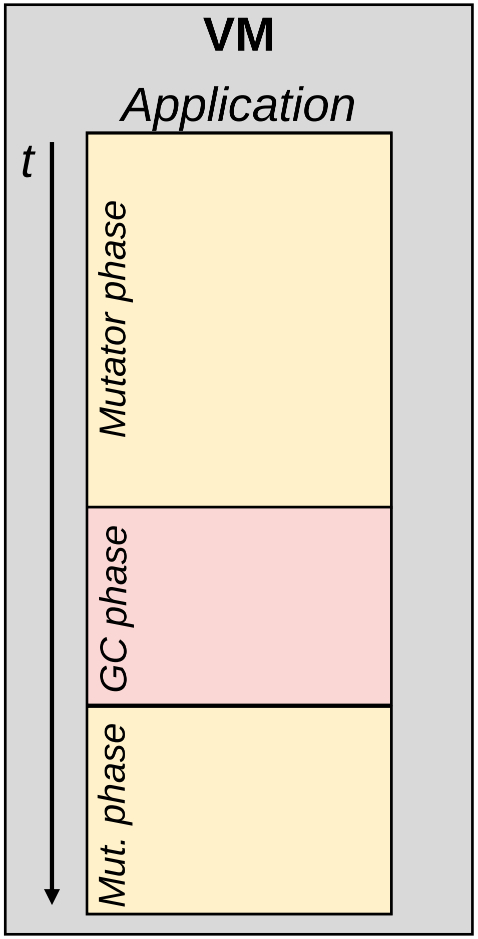
Allocations in heavily executed loops

Boxed primitives
(e.g., `ArrayList<Integer>`)

Streams (multiple `map` operations, late `filter` operations, etc.)

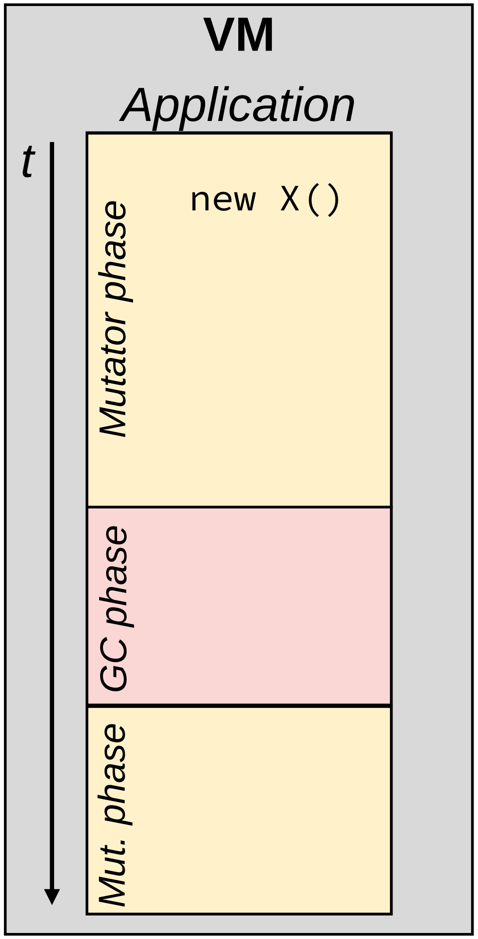
Inefficient database accesses

# DATA: MEMORY TRACES



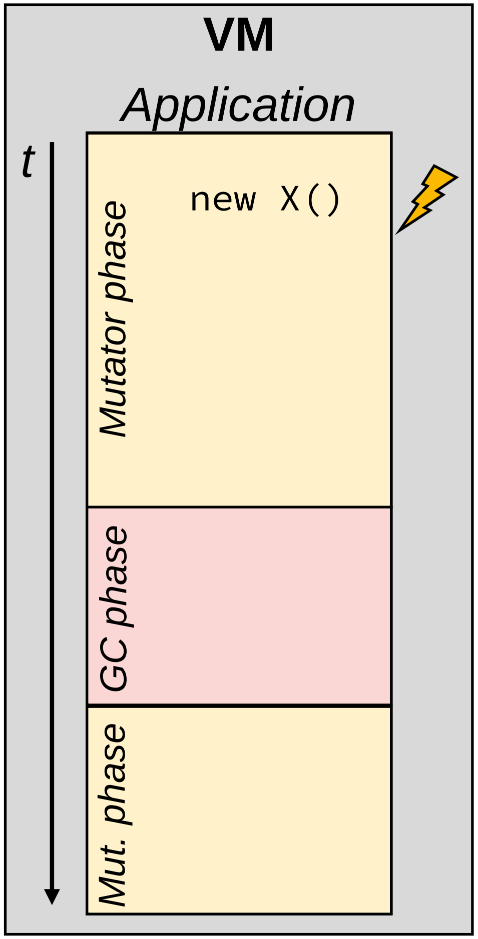

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



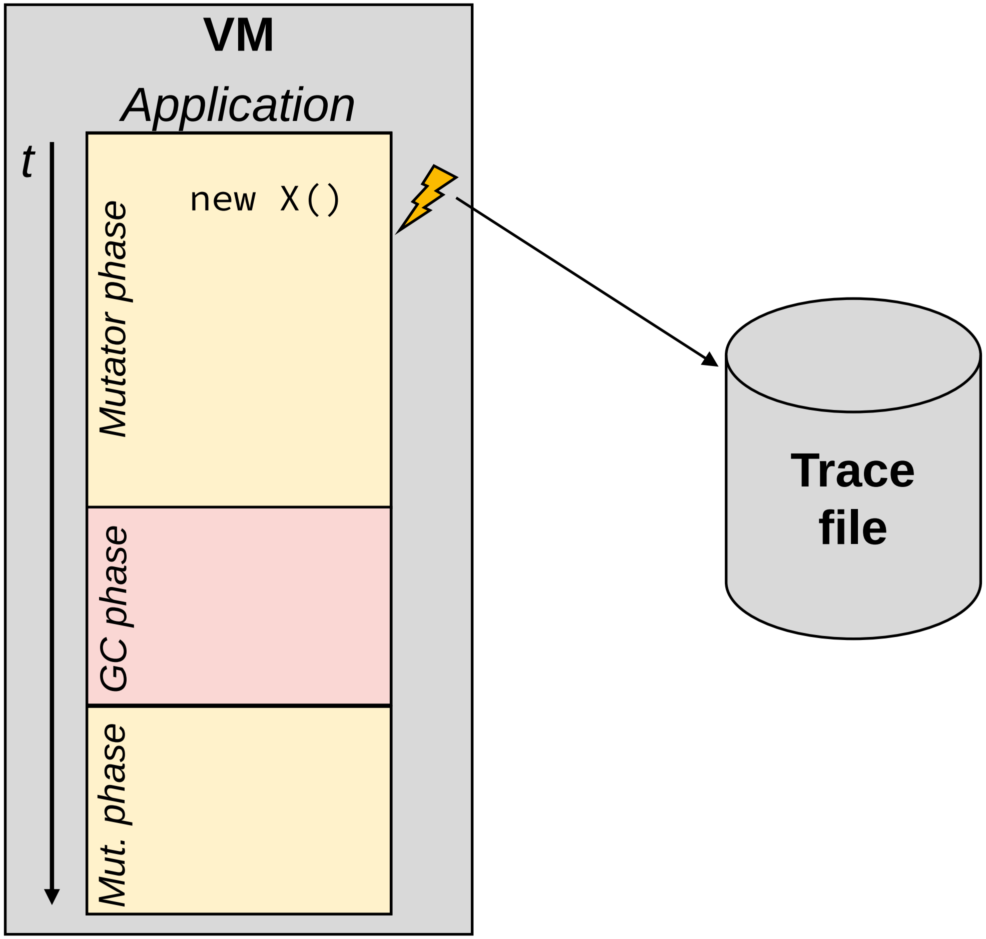

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



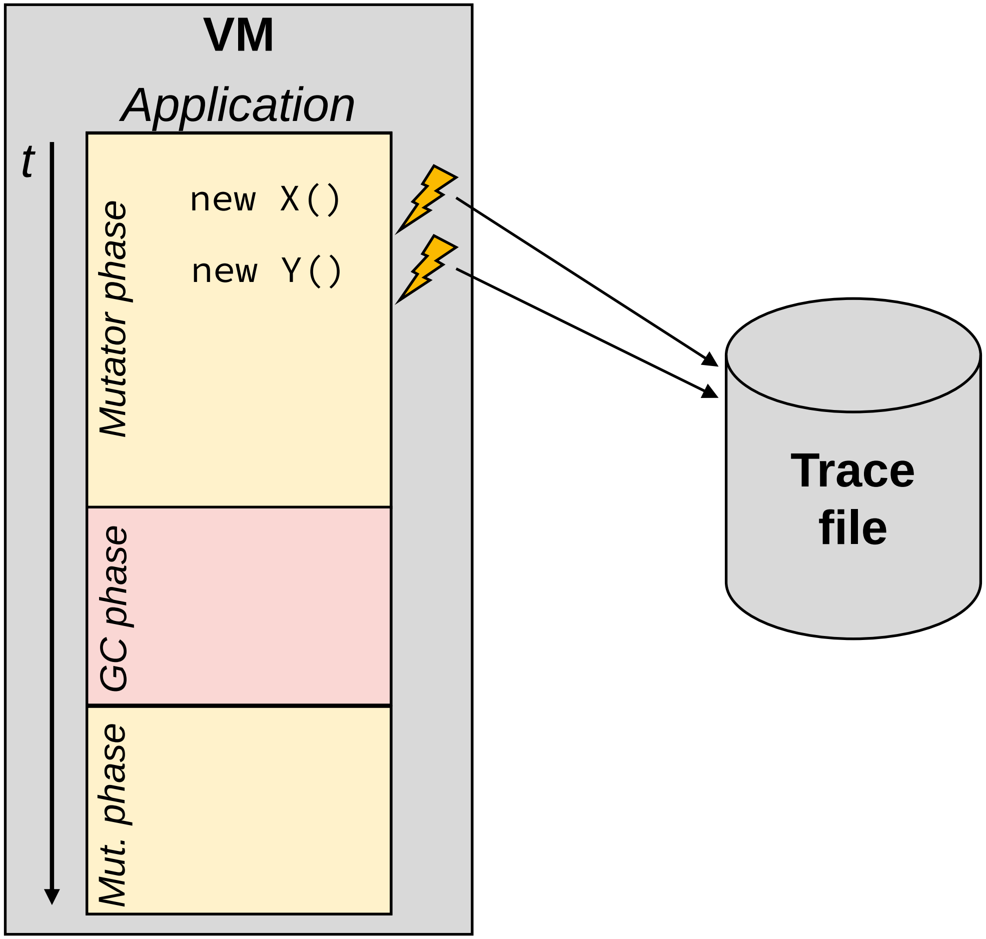

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



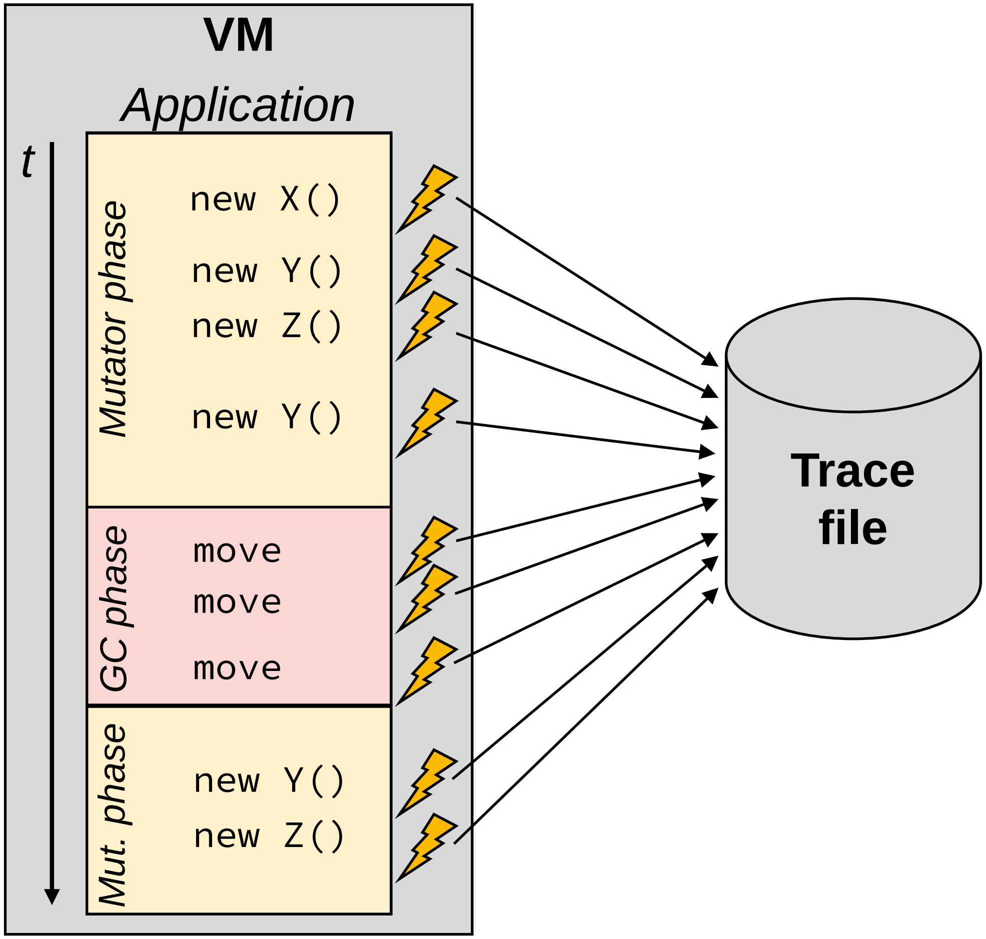

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



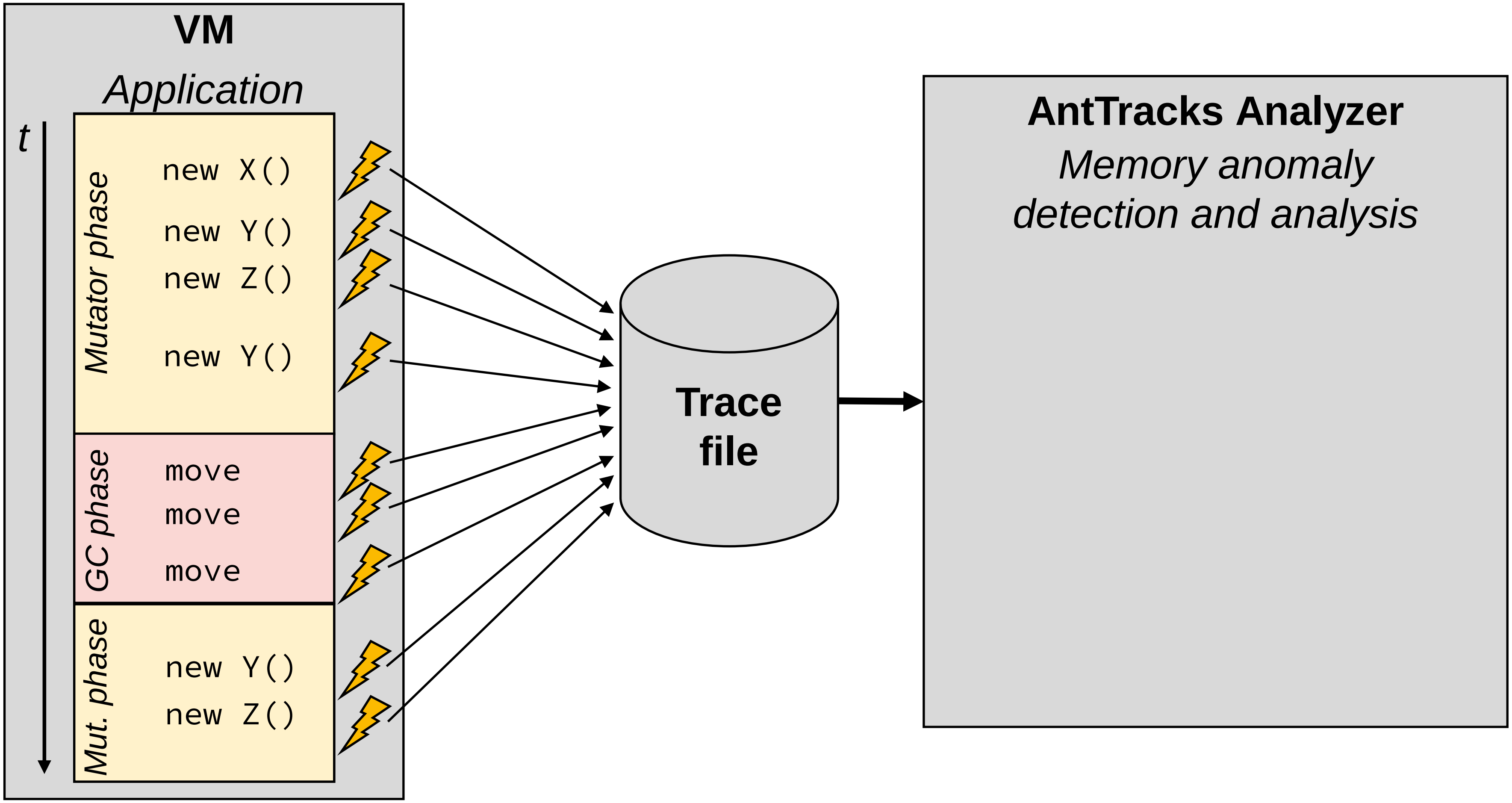

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



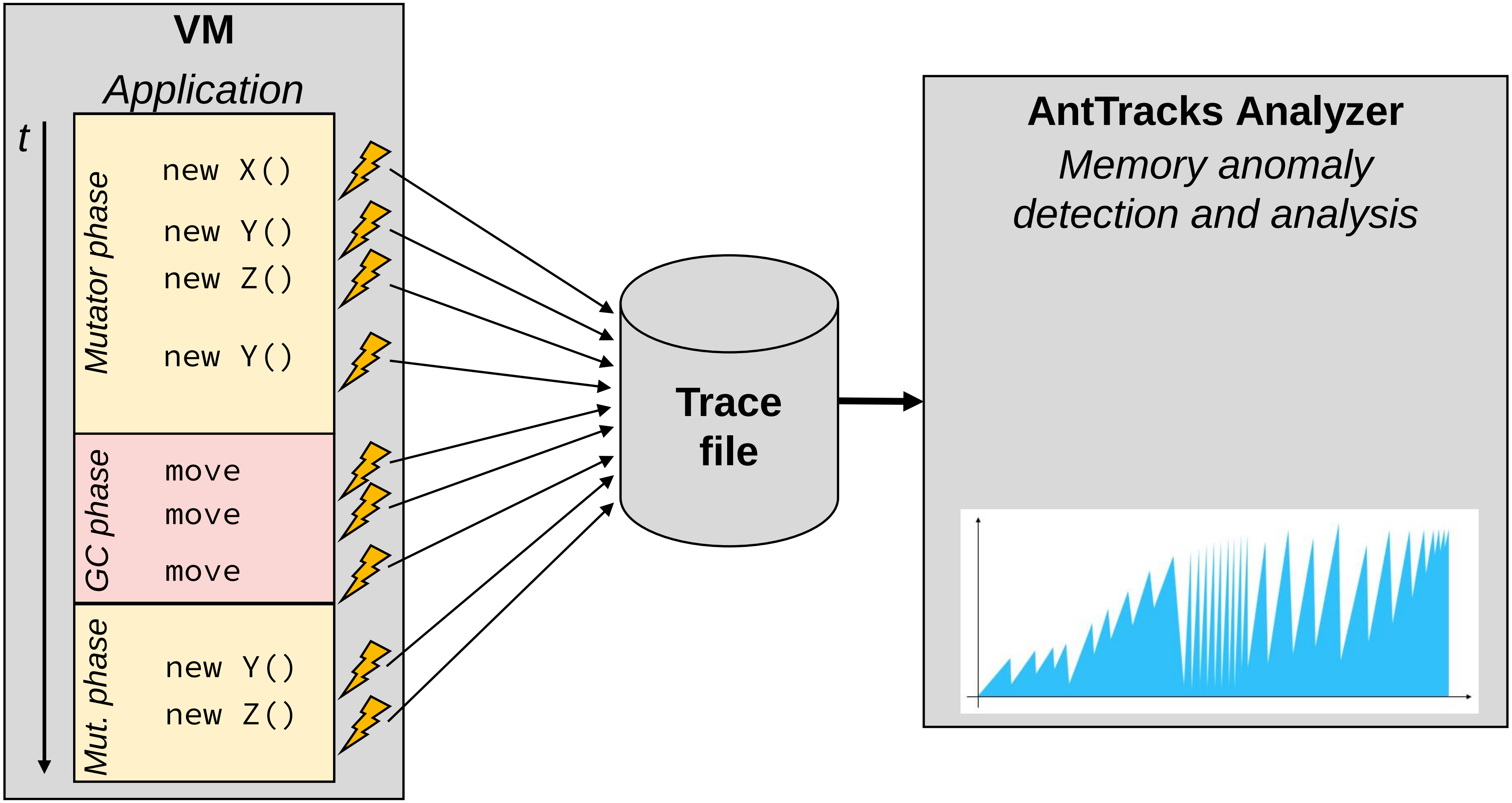

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



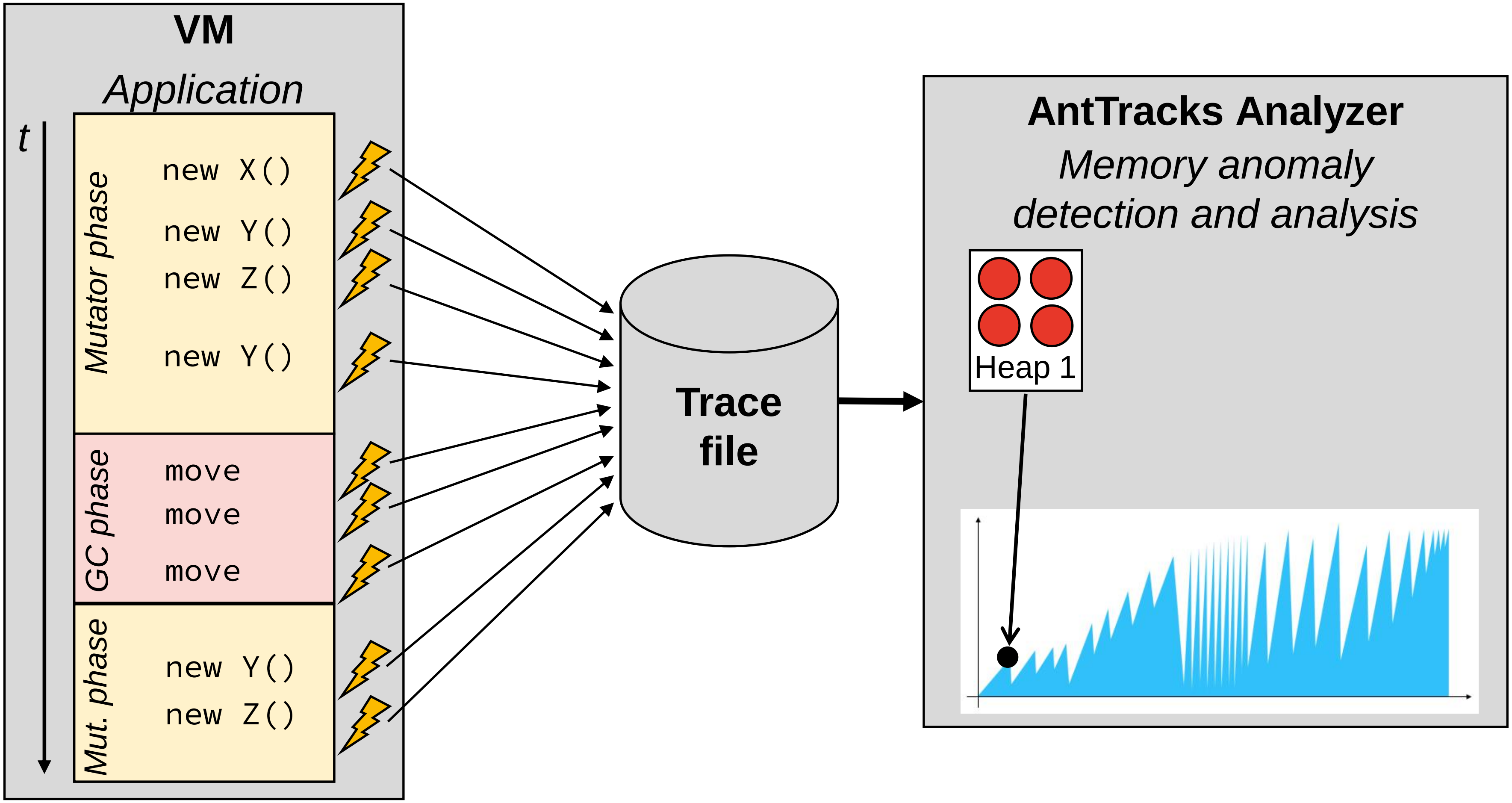

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



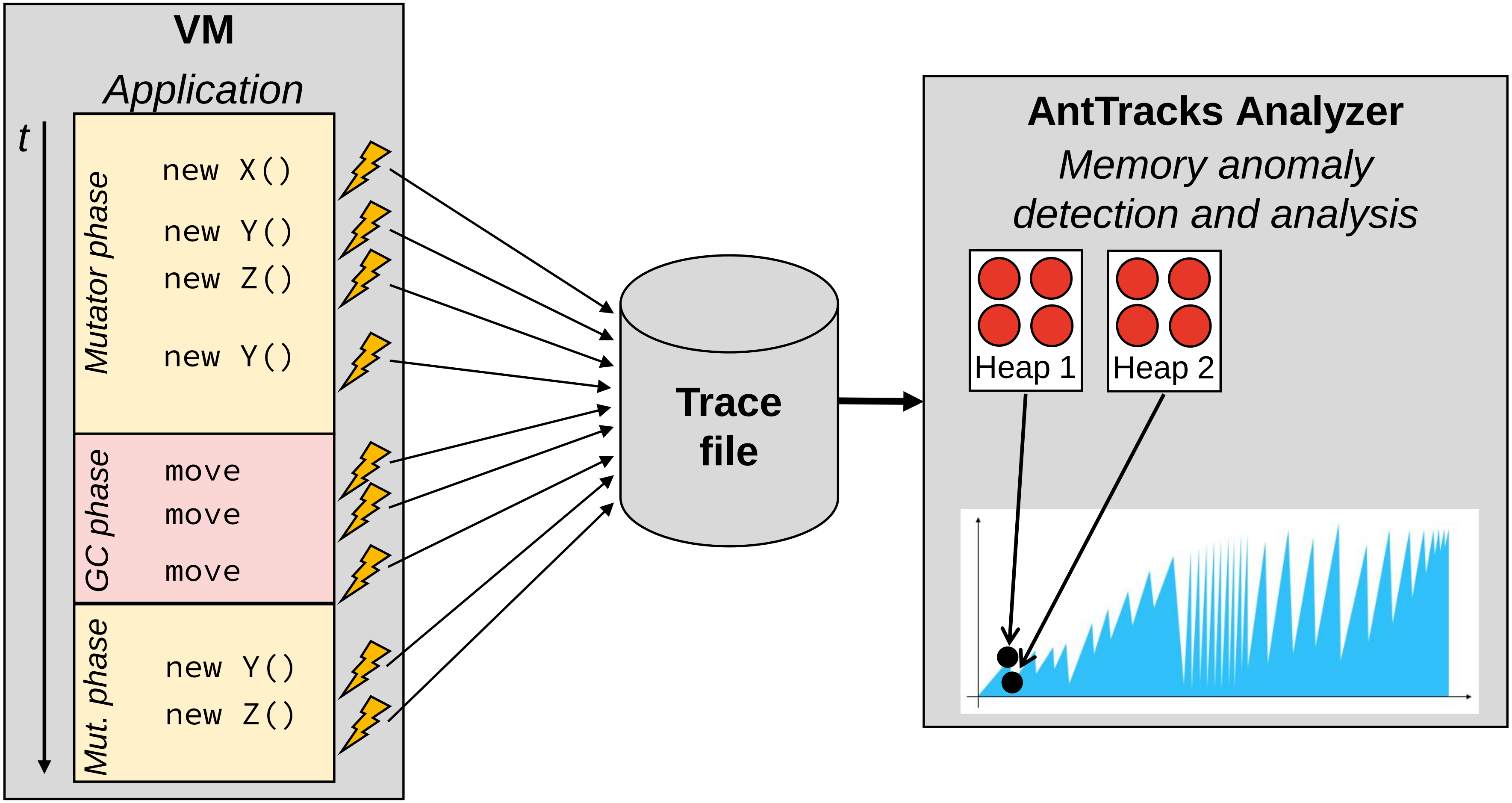

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



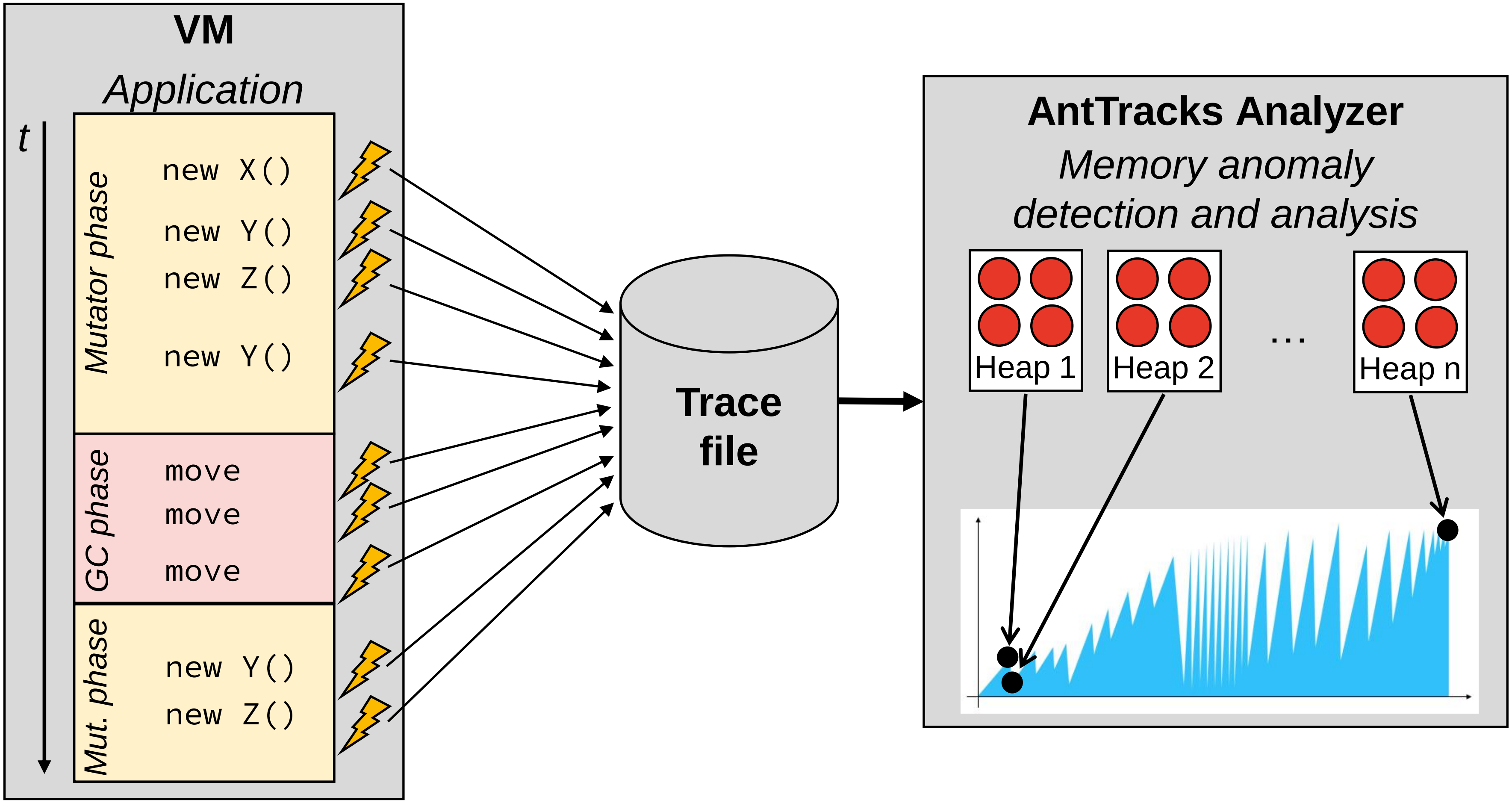

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



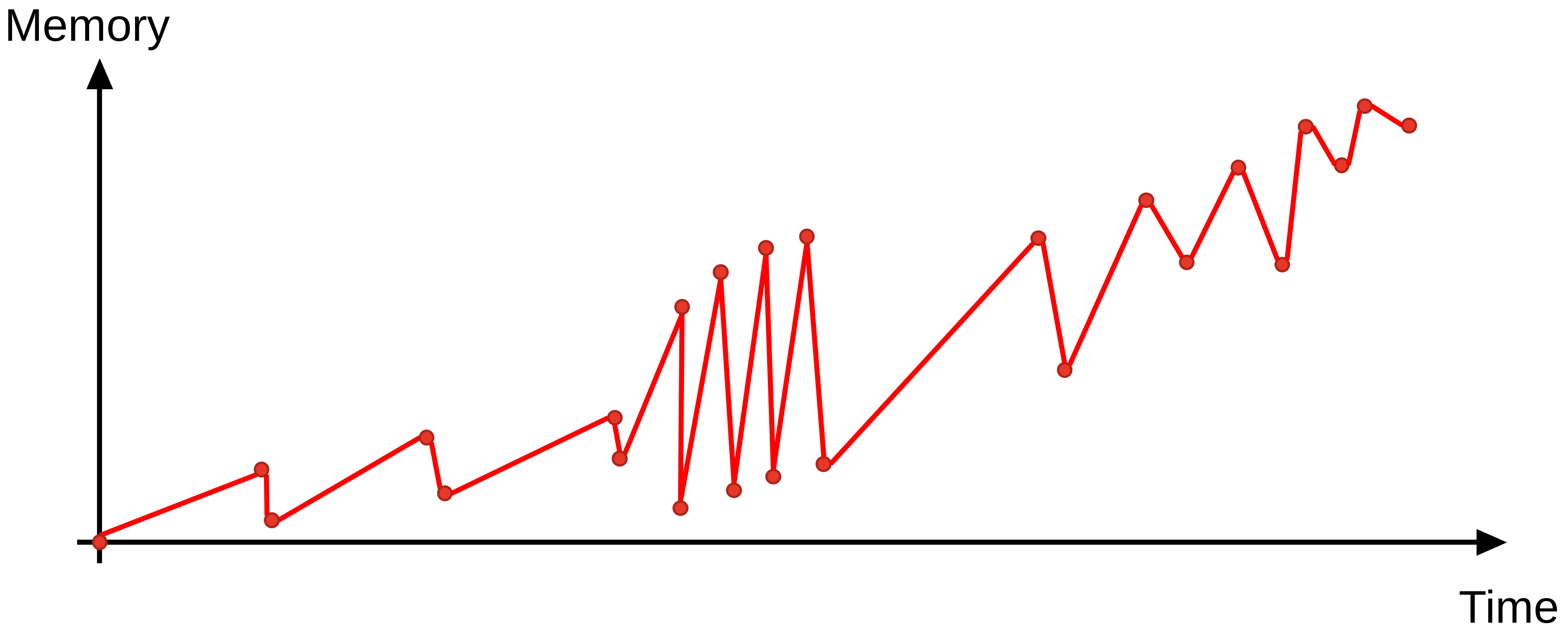

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# DATA: MEMORY TRACES



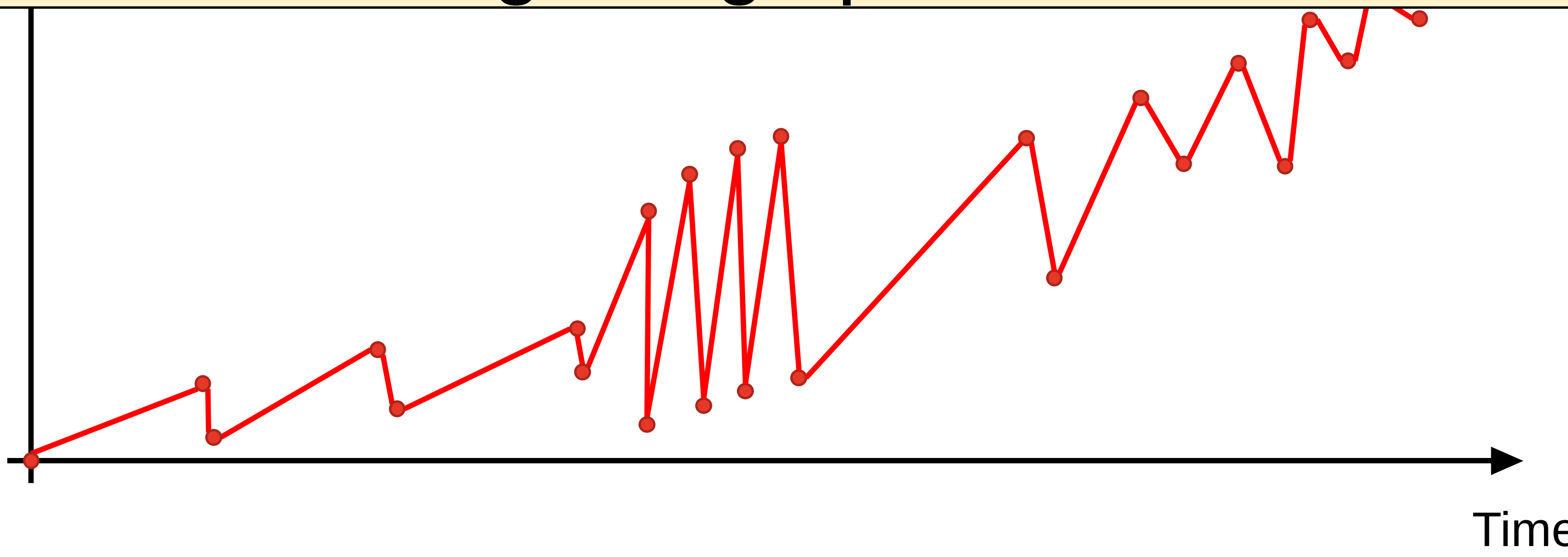

*Lengauer, Bitto, Mössenböck: Accurate and Efficient Object Tracing for Java Applications, ICPE 2015*
*Lengauer et al.: Efficient Memory Traces with Full Pointer Information, PPPJ 2016*

# TIME-WINDOW DETECTION

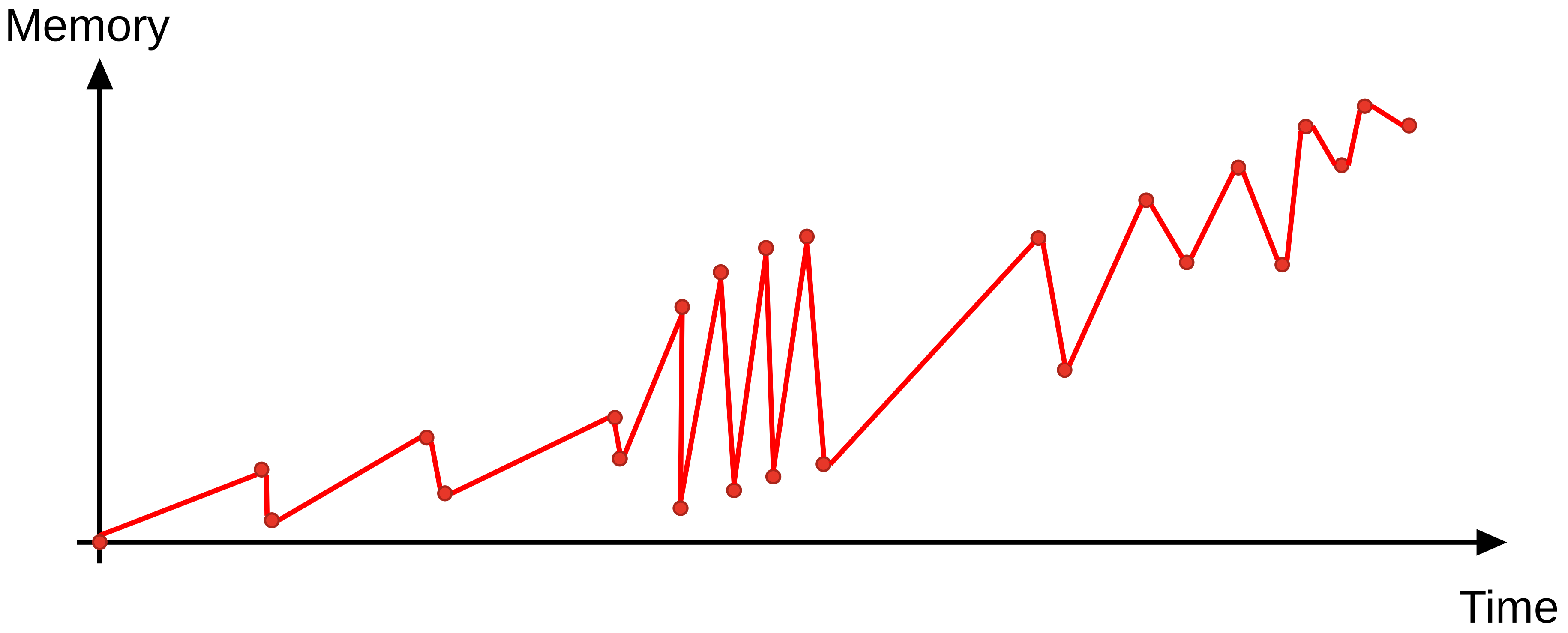


# TIME-WINDOW DETECTION

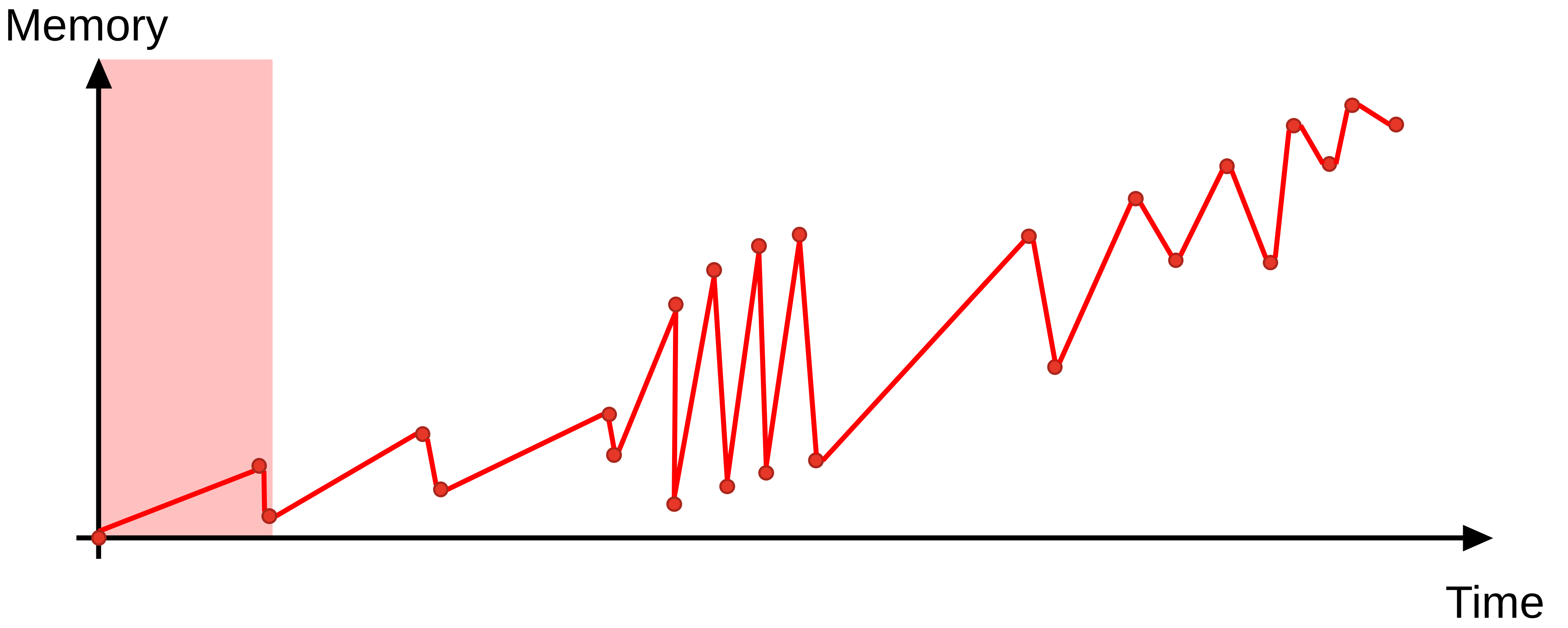
Find time window with
most garbage per second



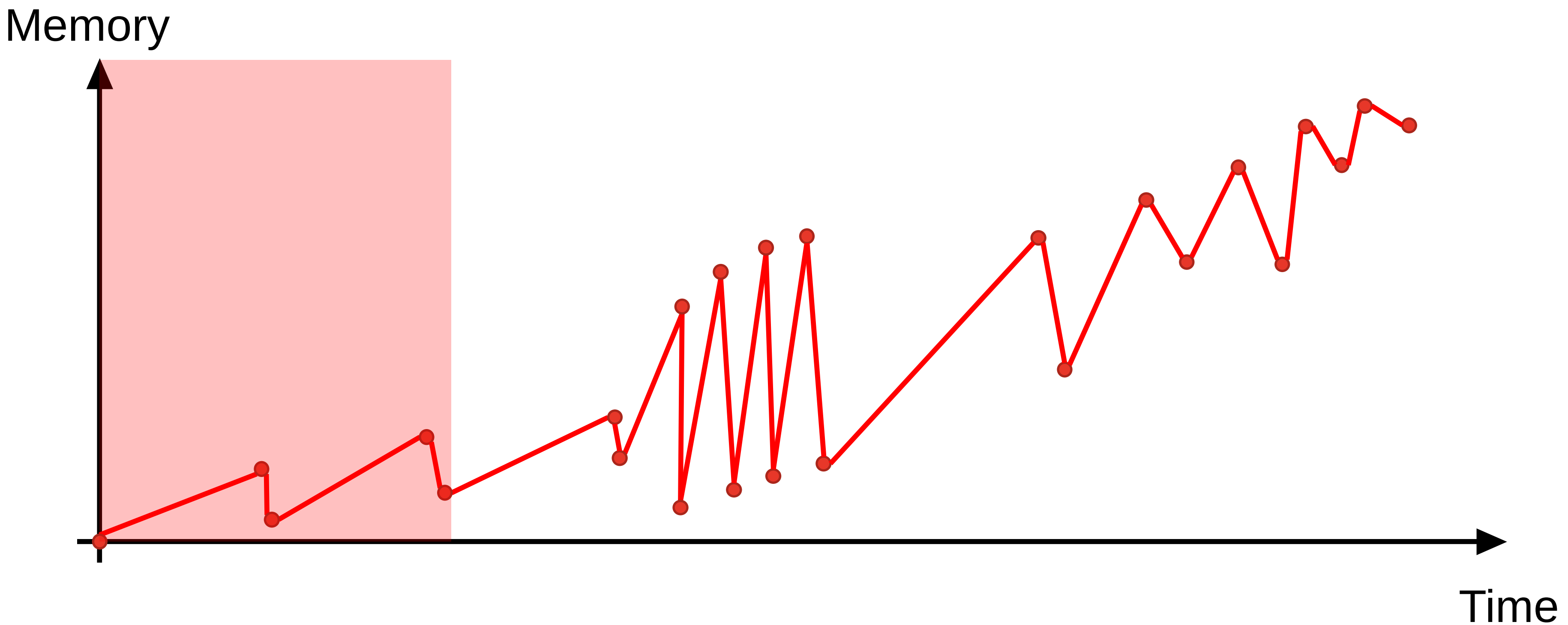
# TIME-WINDOW DETECTION



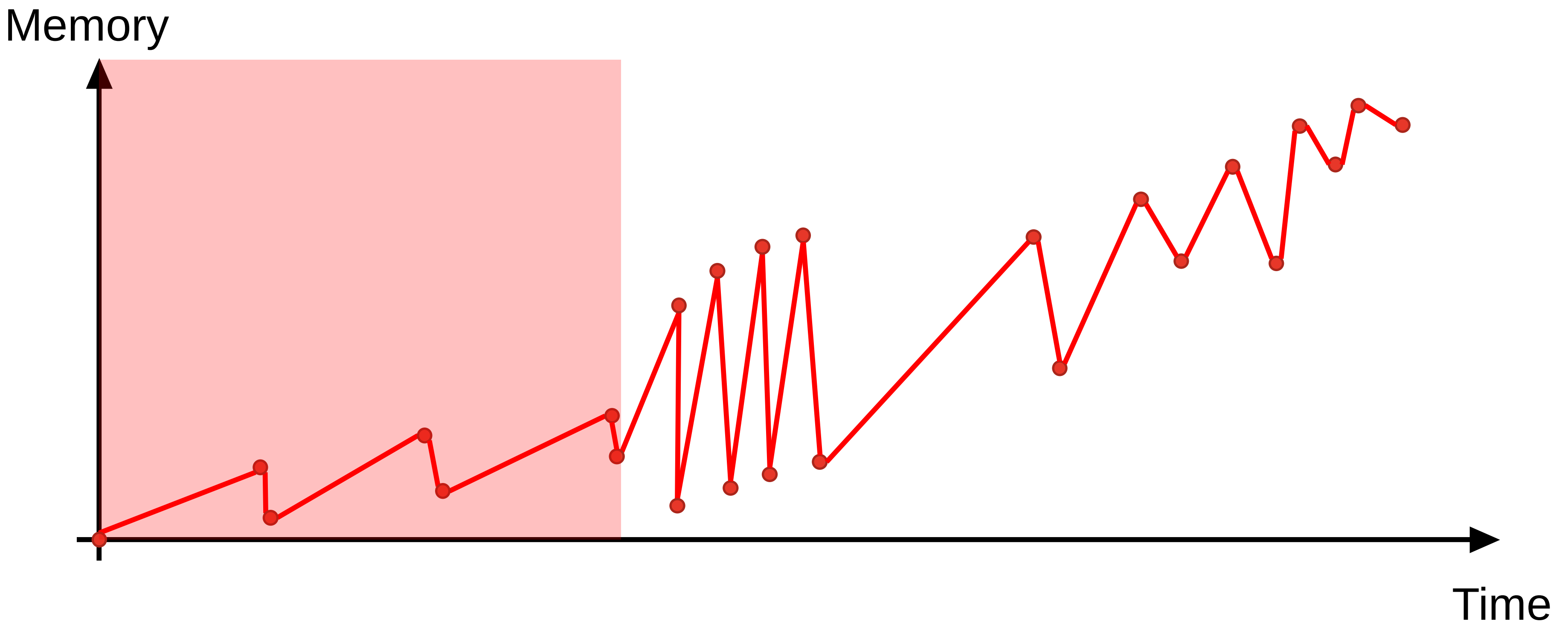
# TIME-WINDOW DETECTION



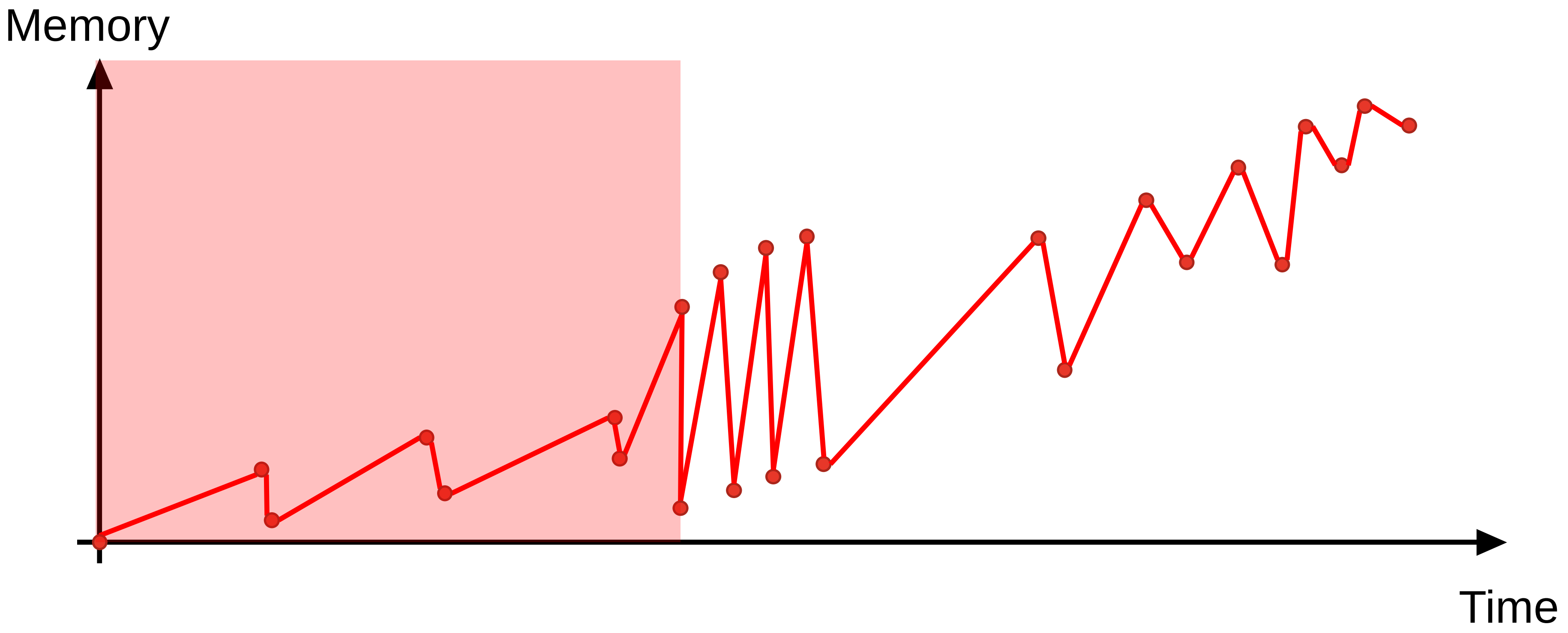
# TIME-WINDOW DETECTION



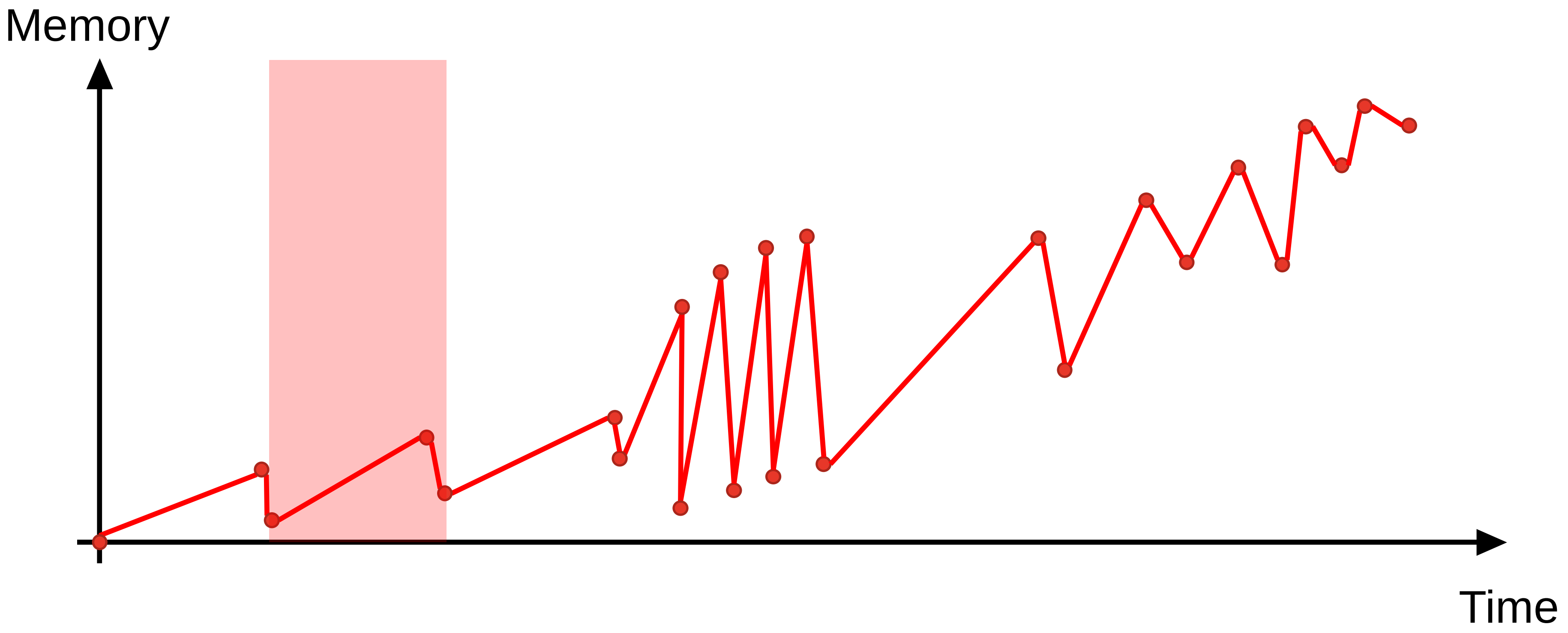
# TIME-WINDOW DETECTION



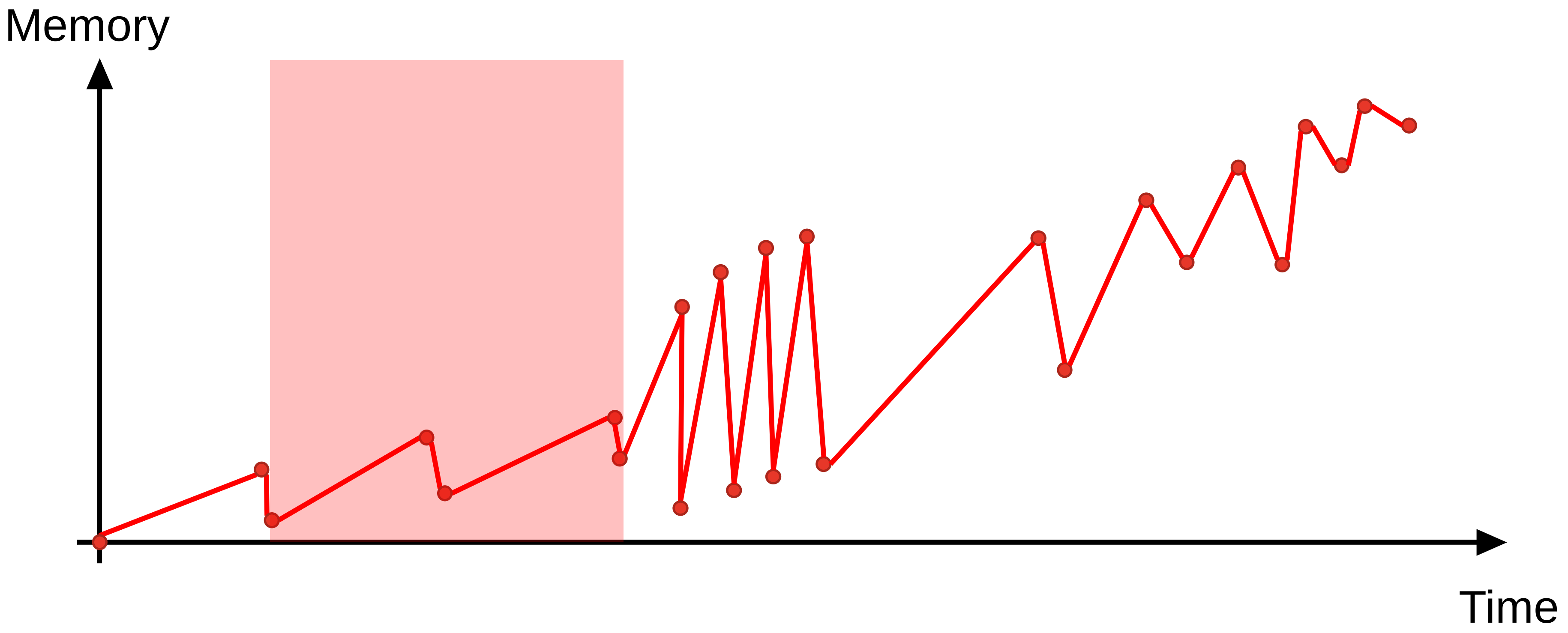
# TIME-WINDOW DETECTION



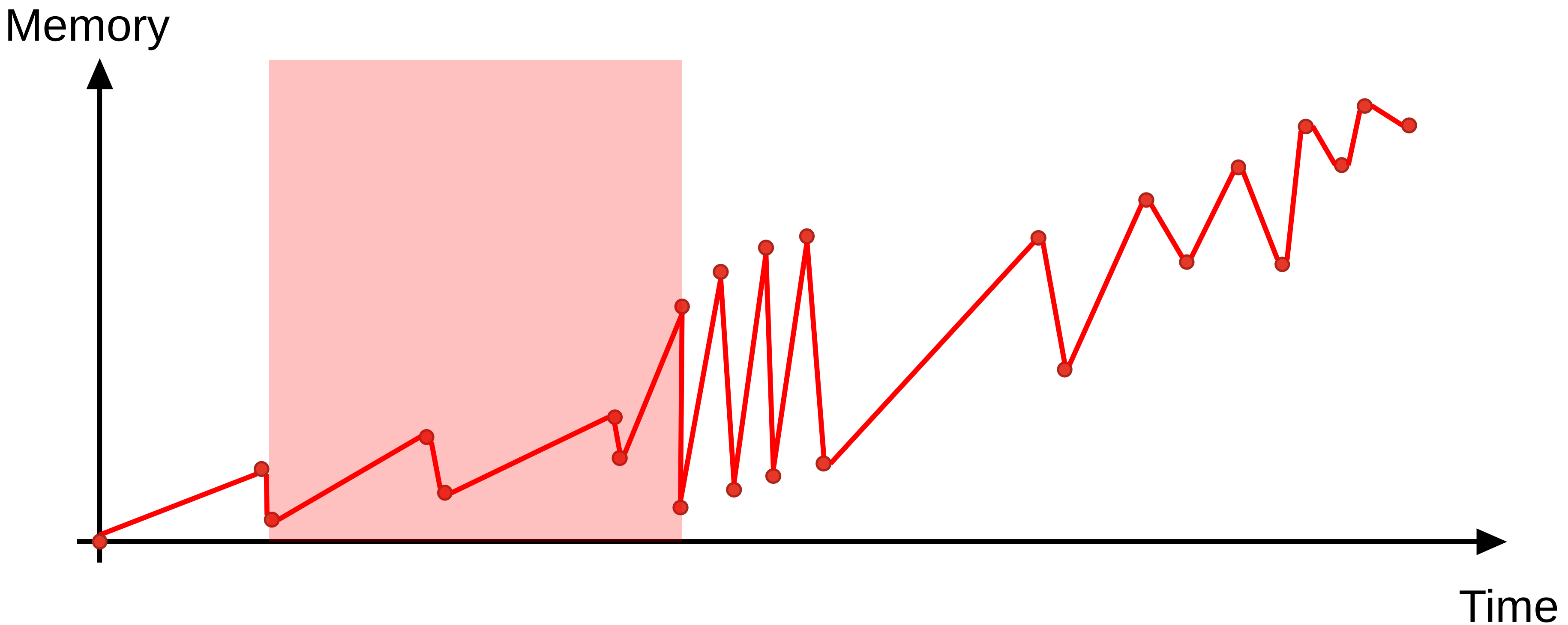
# TIME-WINDOW DETECTION



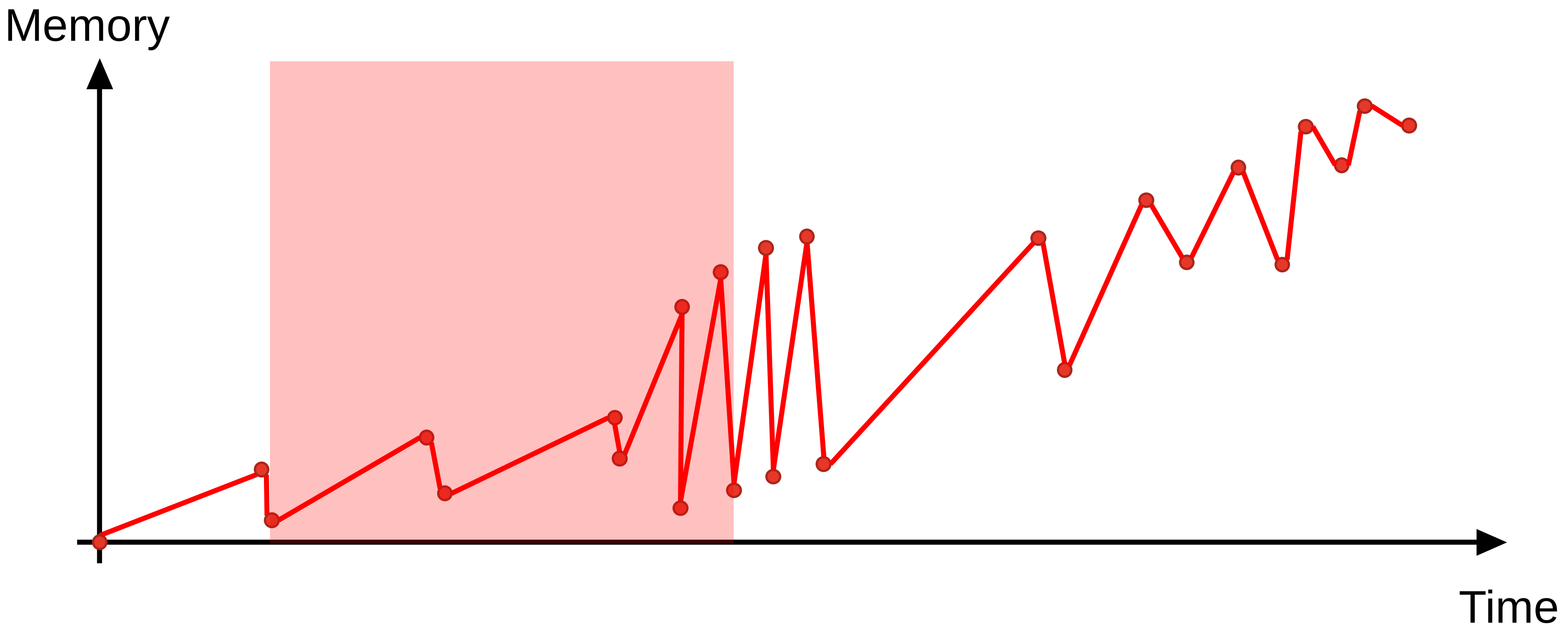
# TIME-WINDOW DETECTION



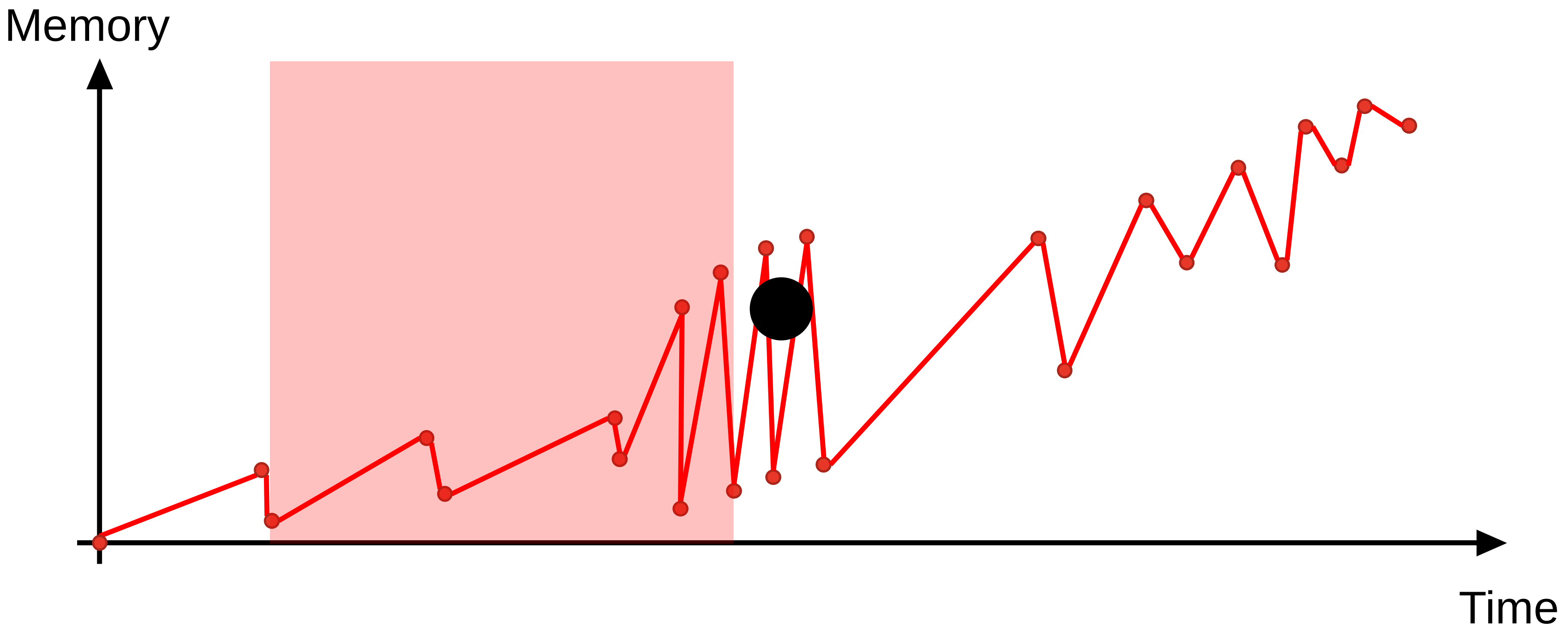
# TIME-WINDOW DETECTION



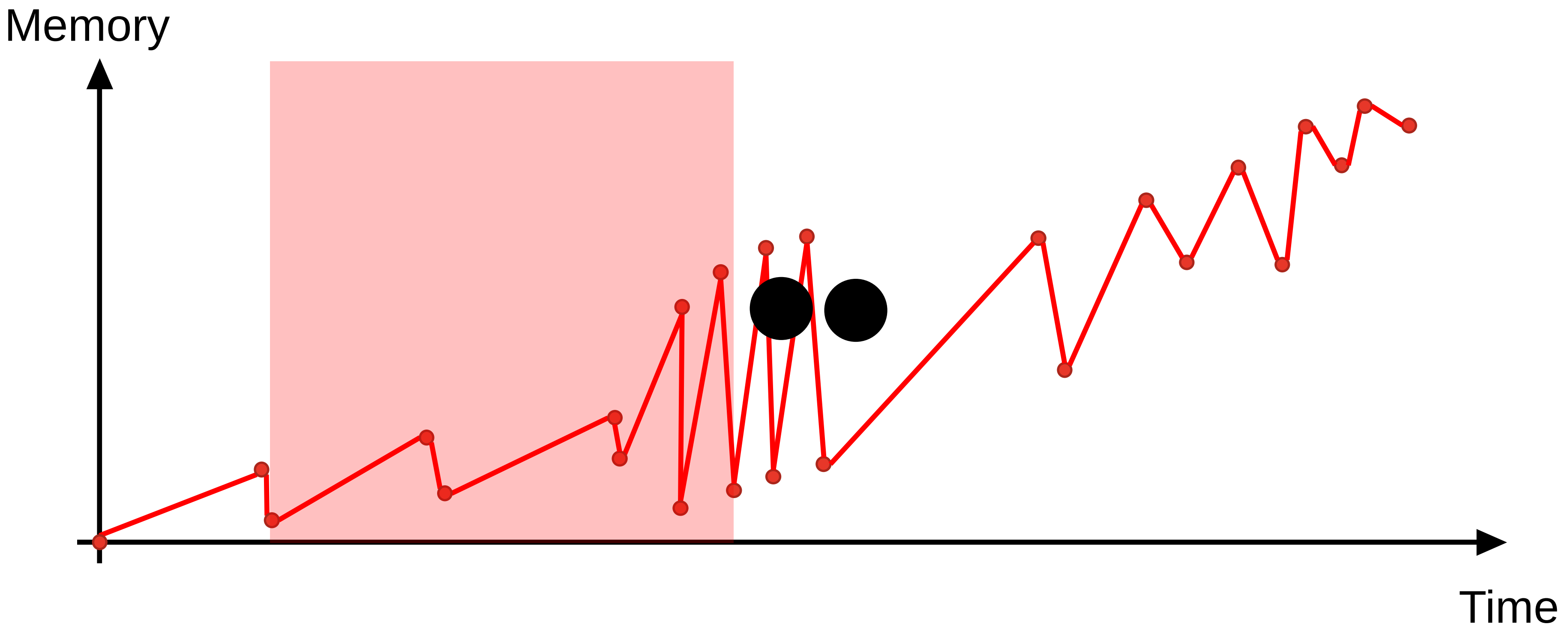
# TIME-WINDOW DETECTION



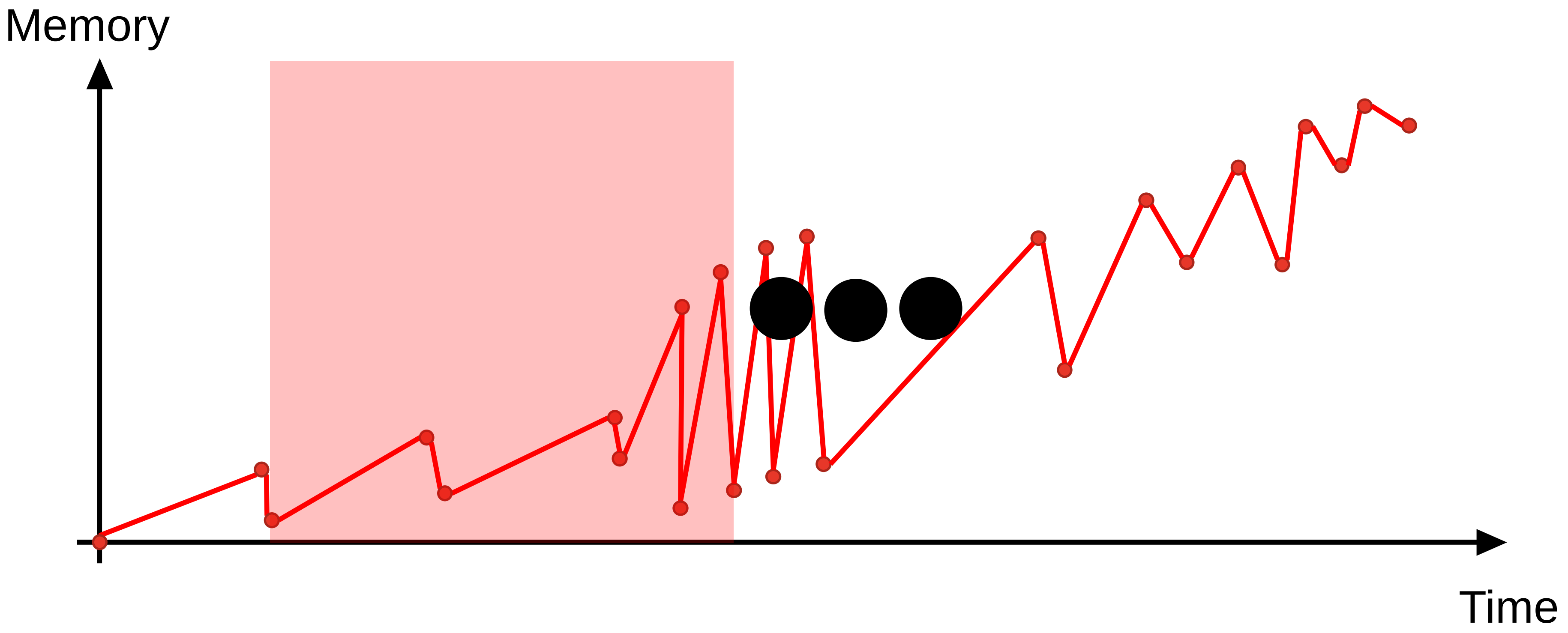
# TIME-WINDOW DETECTION



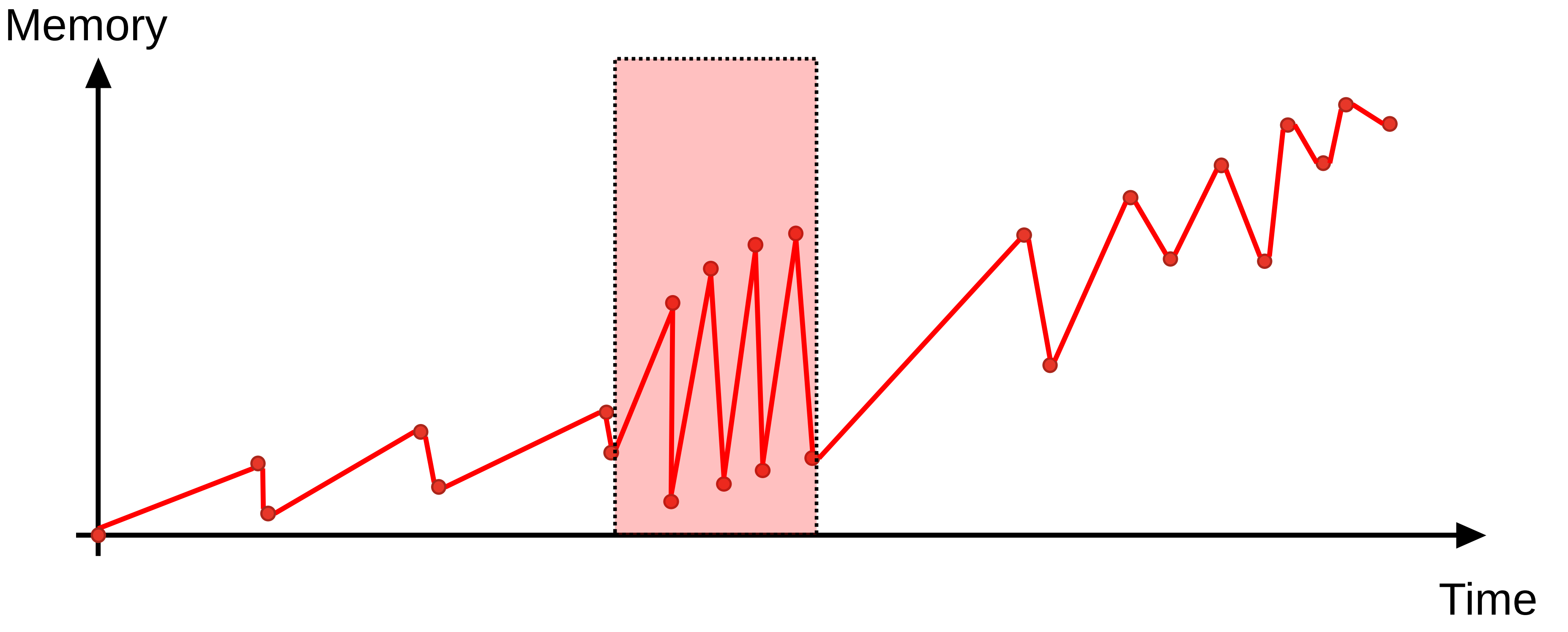
# TIME-WINDOW DETECTION



# TIME-WINDOW DETECTION

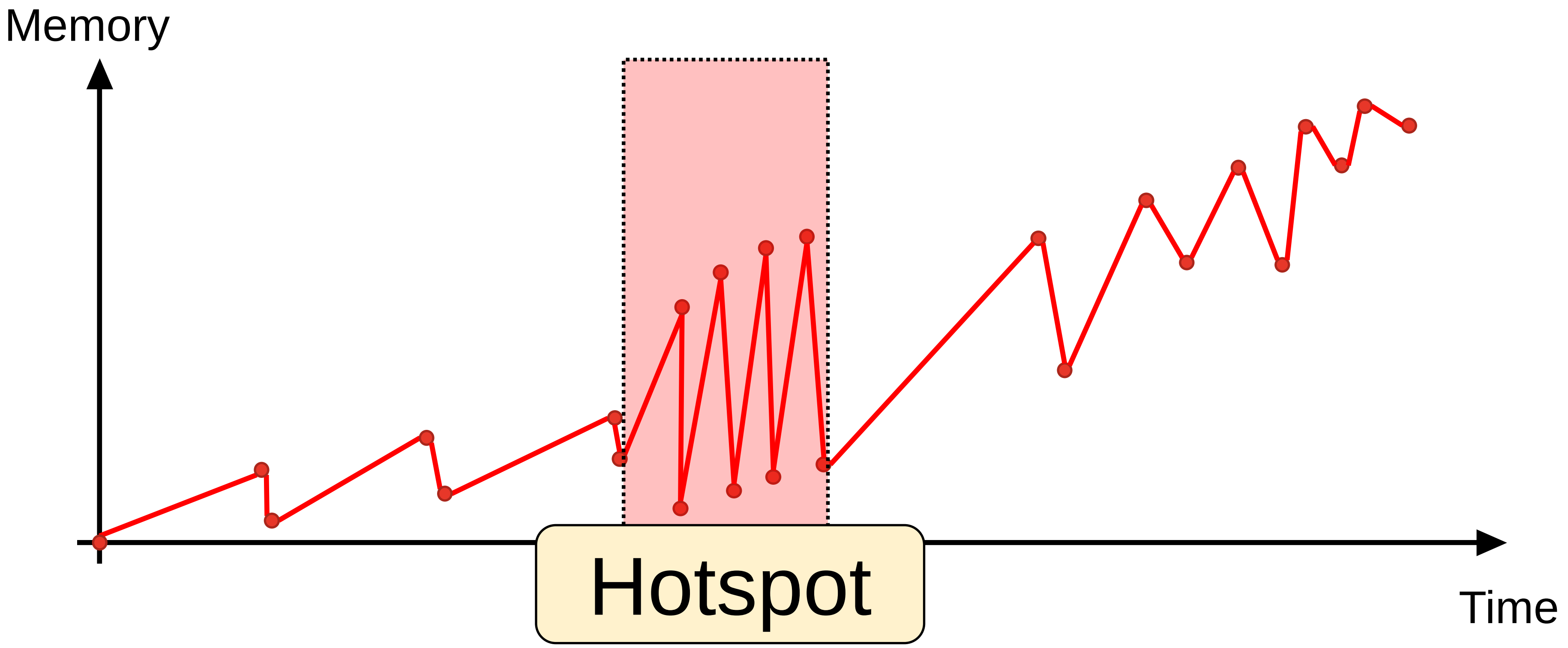


# TIME-WINDOW DETECTION

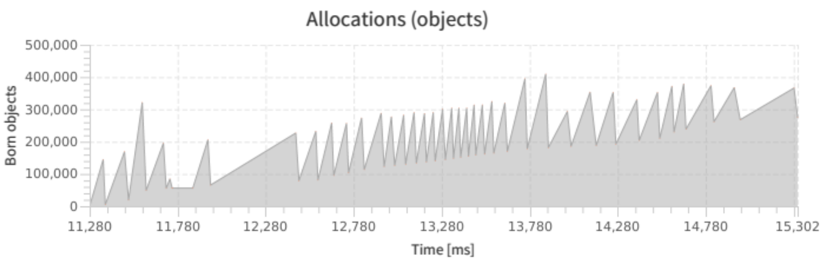


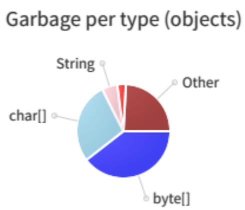

JKU

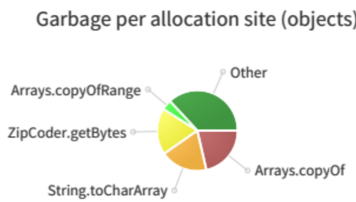
# TIME-WINDOW DETECTION

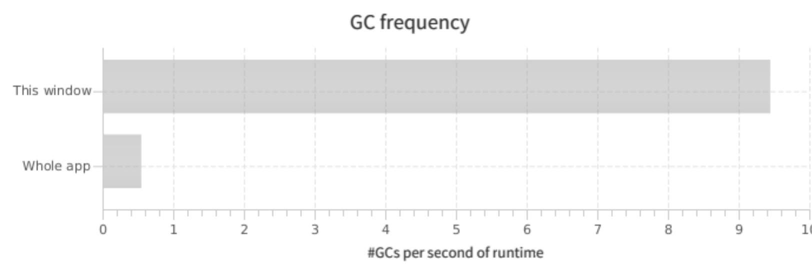


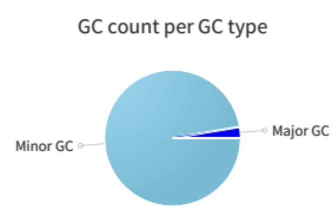
# SHORT-LIVED OBJECTS OVERVIEW

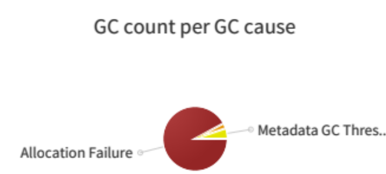


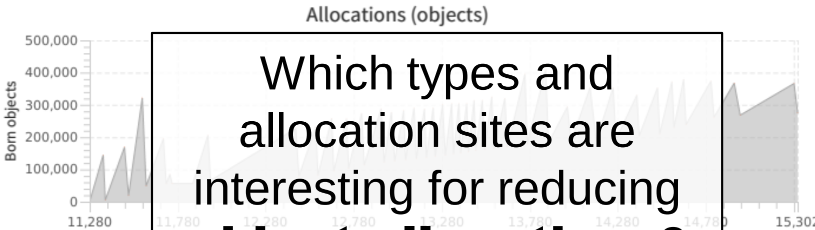



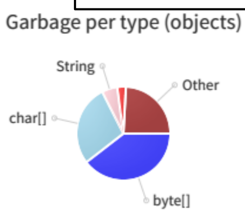



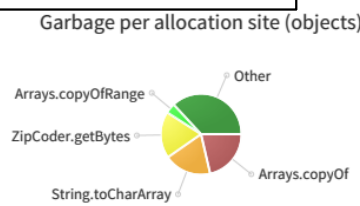



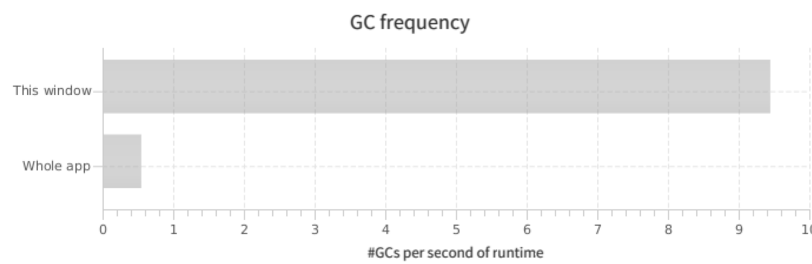
# SHORT-LIVED OBJECTS OVERVIEW

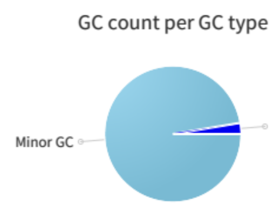


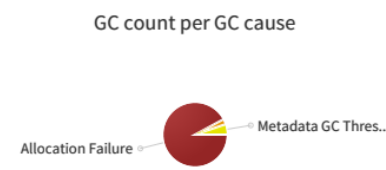

Which types and allocation sites are interesting for reducing **object allocations**?



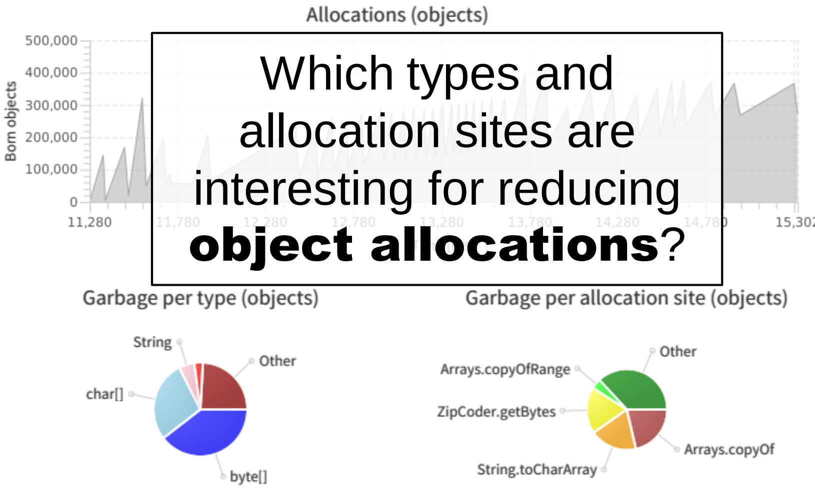



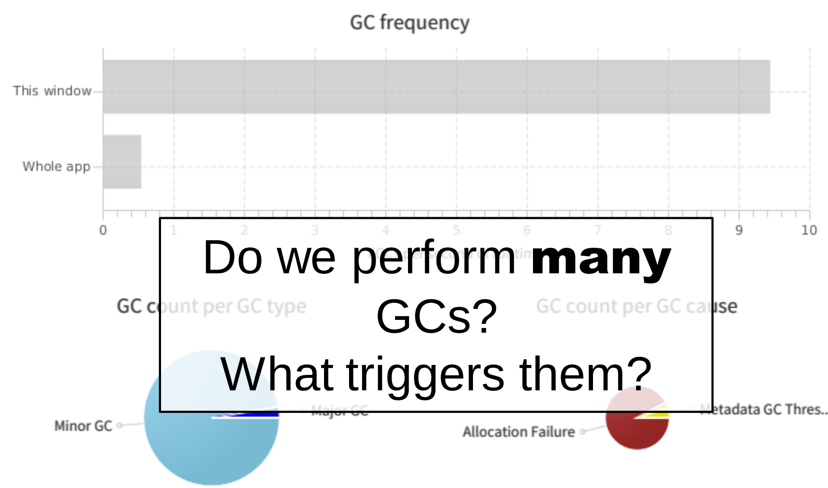



# SHORT-LIVED OBJECTS OVERVIEW

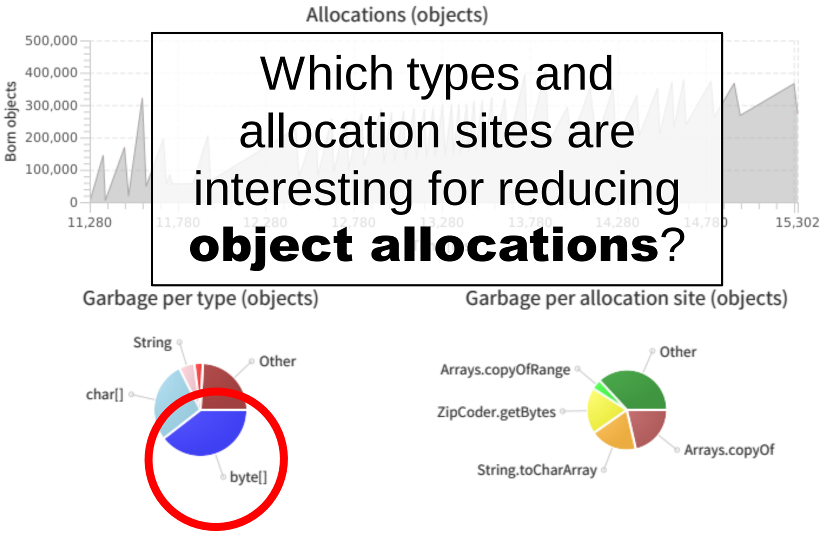



Which types and allocation sites are interesting for reducing **object allocations**?

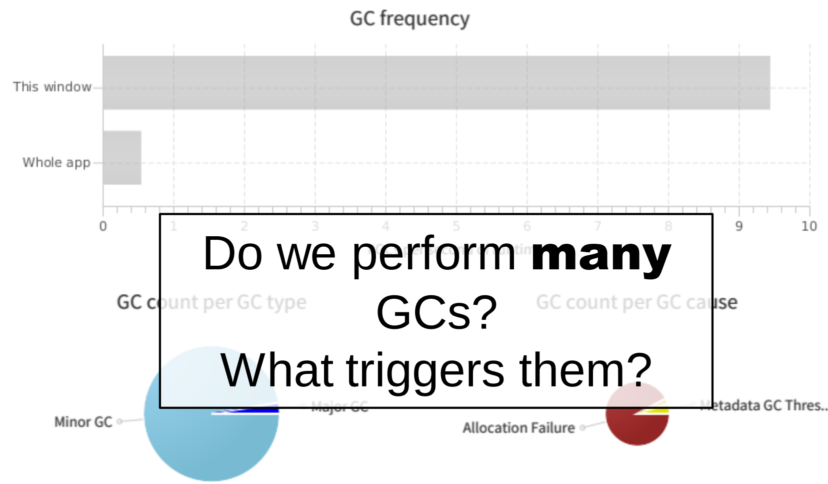



Do we perform **many** GCs?
What triggers them?

# SHORT-LIVED OBJECTS OVERVIEW

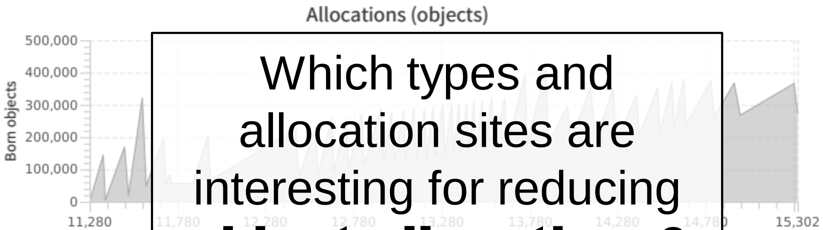



Which types and allocation sites are interesting for reducing **object allocations**?

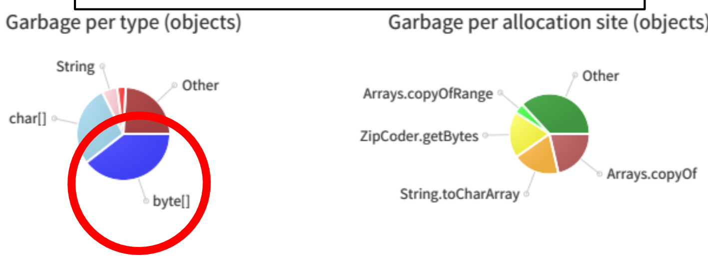



Do we perform **many** GCs?
What triggers them?

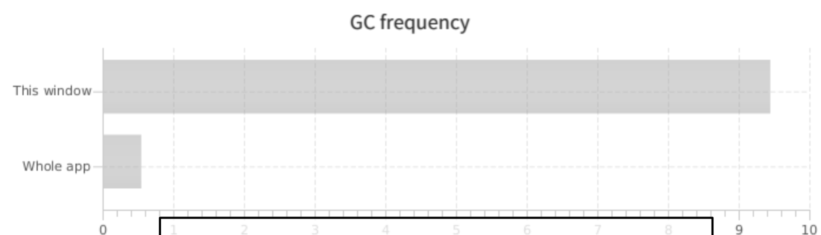
# SHORT-LIVED OBJECTS OVERVIEW



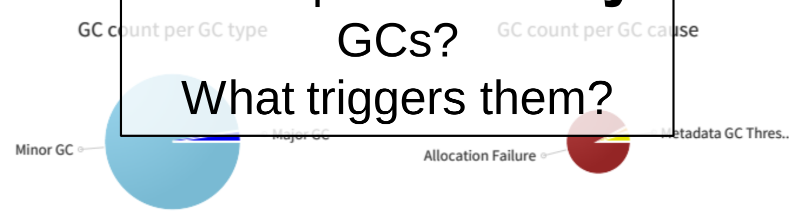

Which types and allocation sites are interesting for reducing **object allocations**?



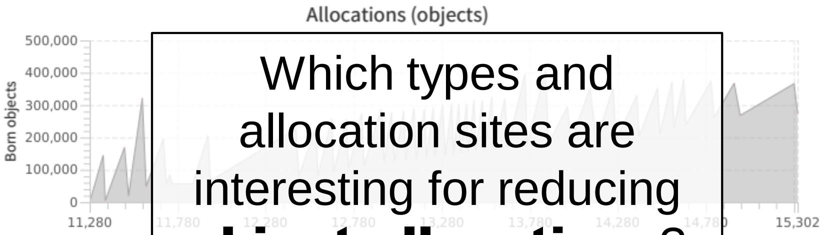

1/3 of all died objects are of a single type



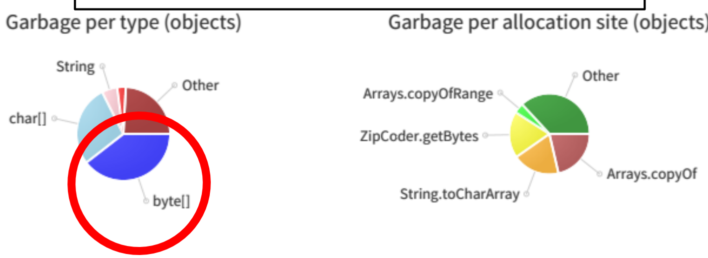

Do we perform **many** GCs?
What triggers them?



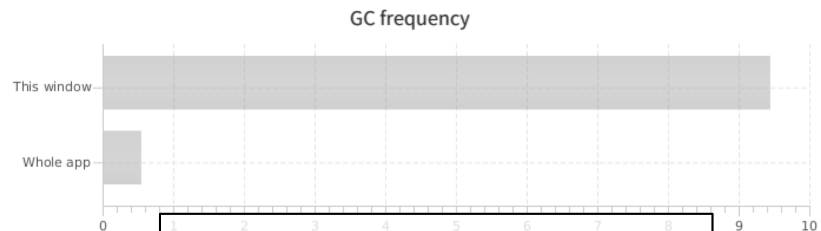
# SHORT-LIVED OBJECTS OVERVIEW




Which types and allocation sites are interesting for reducing **object allocations**?




1/3 of all died objects are of a single t

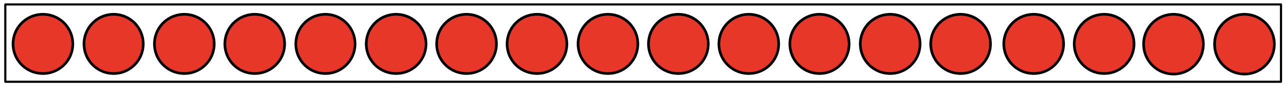



Do we perform **many** GCs?
What triggers them?

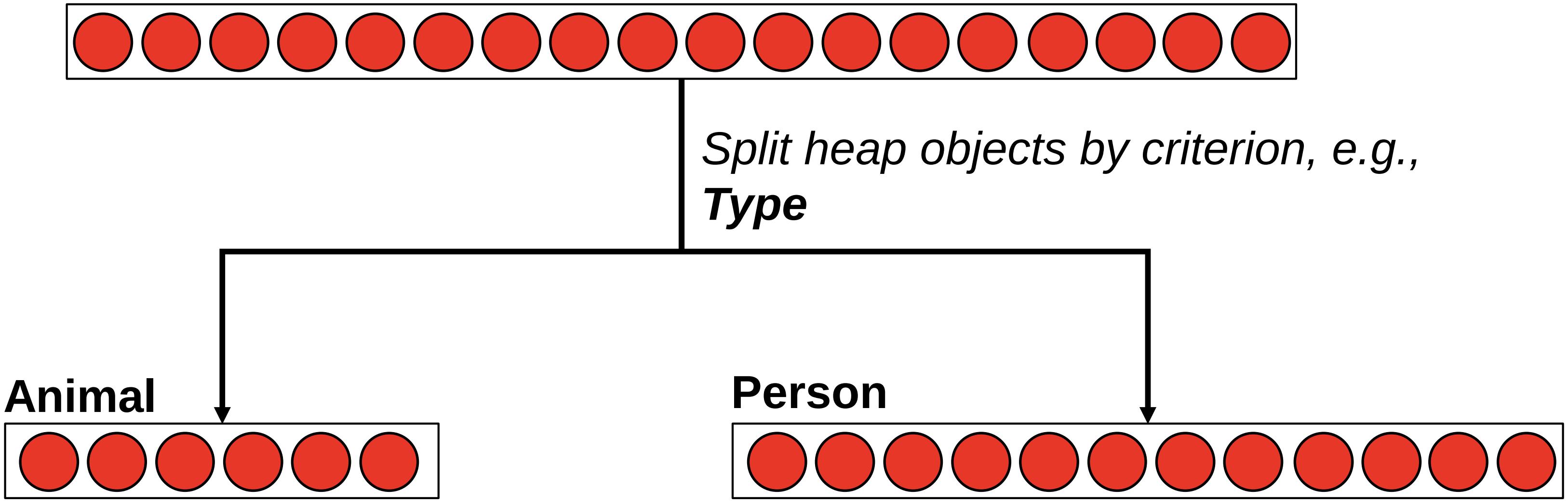

Investigate!

# OBJECT CLASSIFICATION

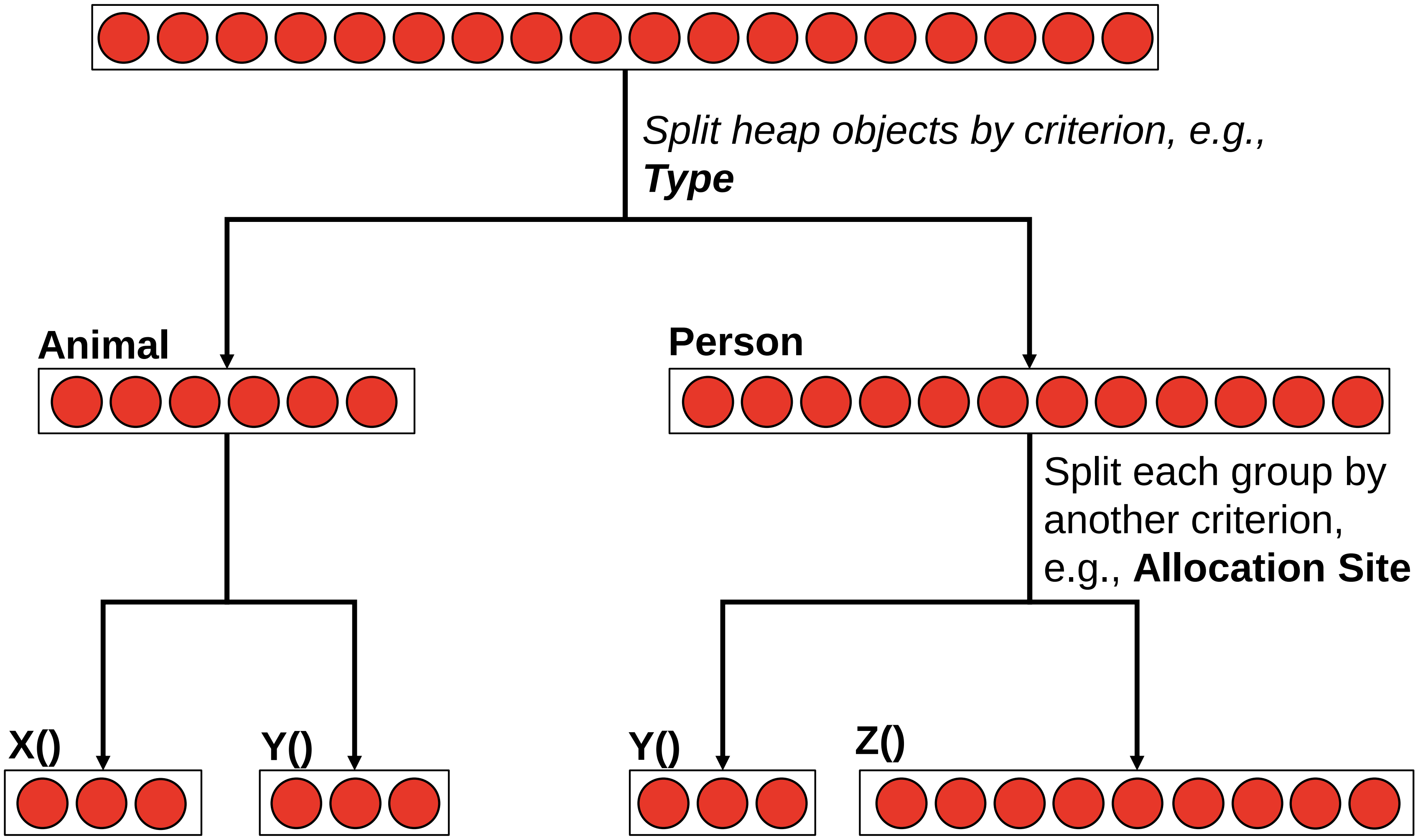
# OBJECT CLASSIFICATION



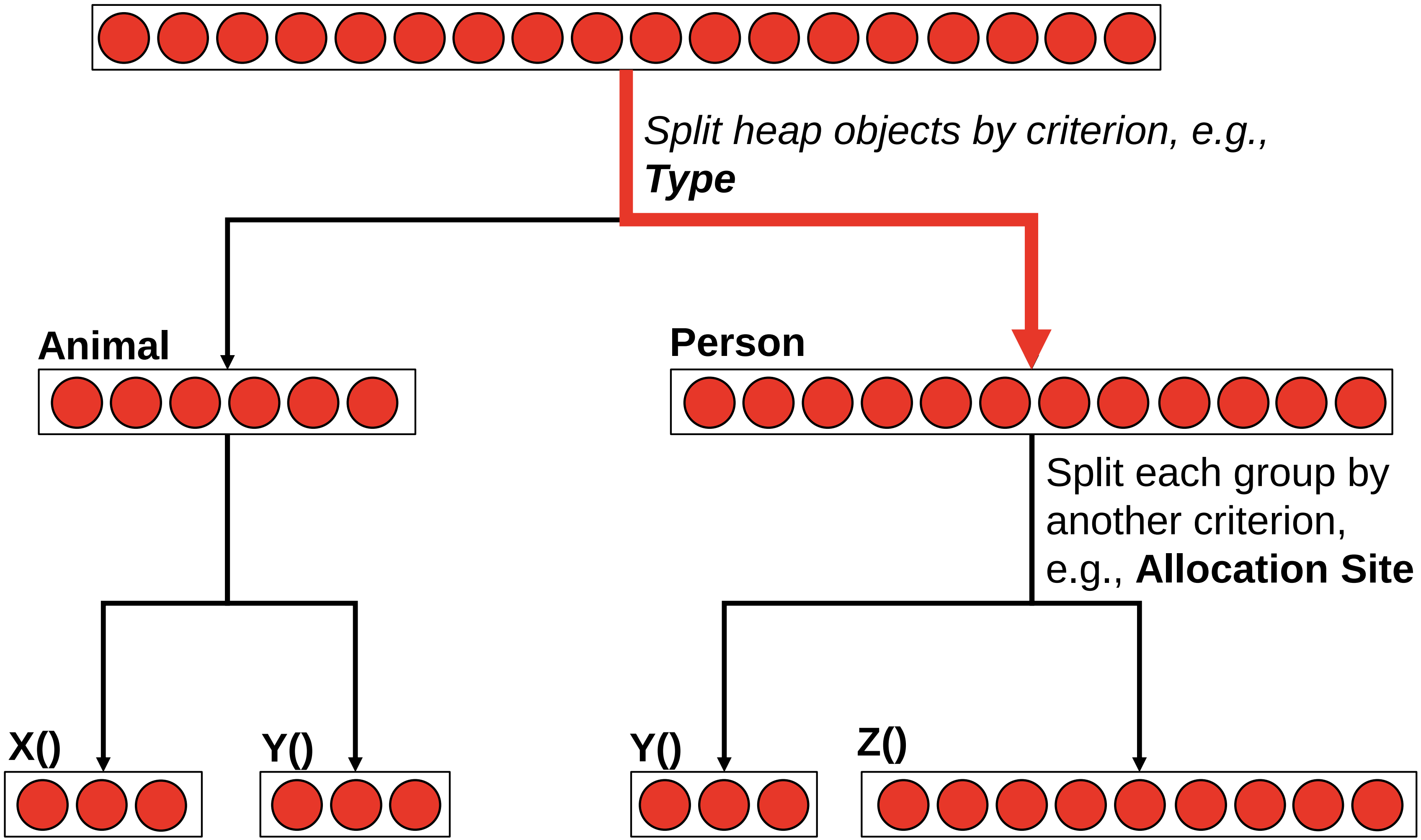
# OBJECT CLASSIFICATION



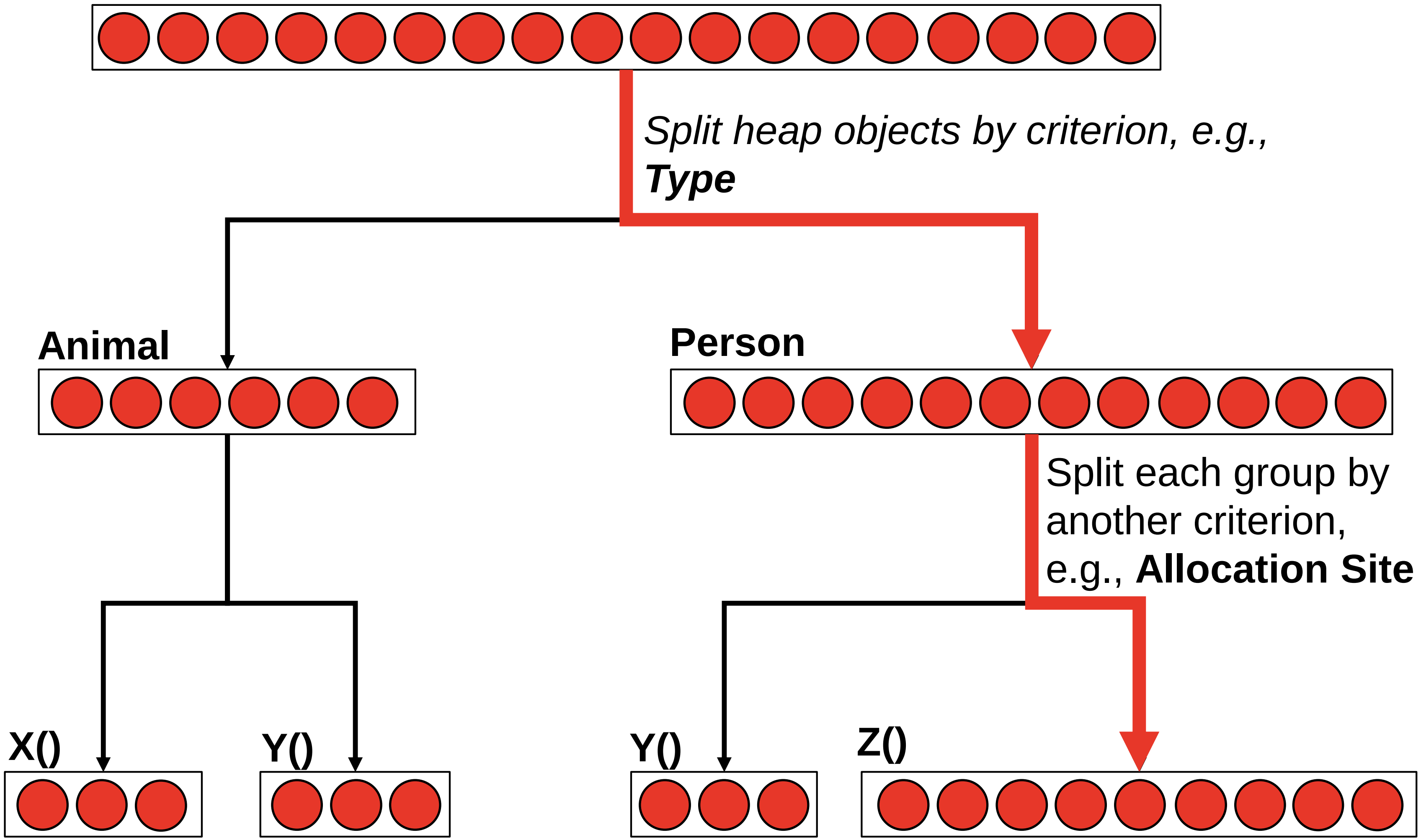
# OBJECT CLASSIFICATION



# OBJECT CLASSIFICATION

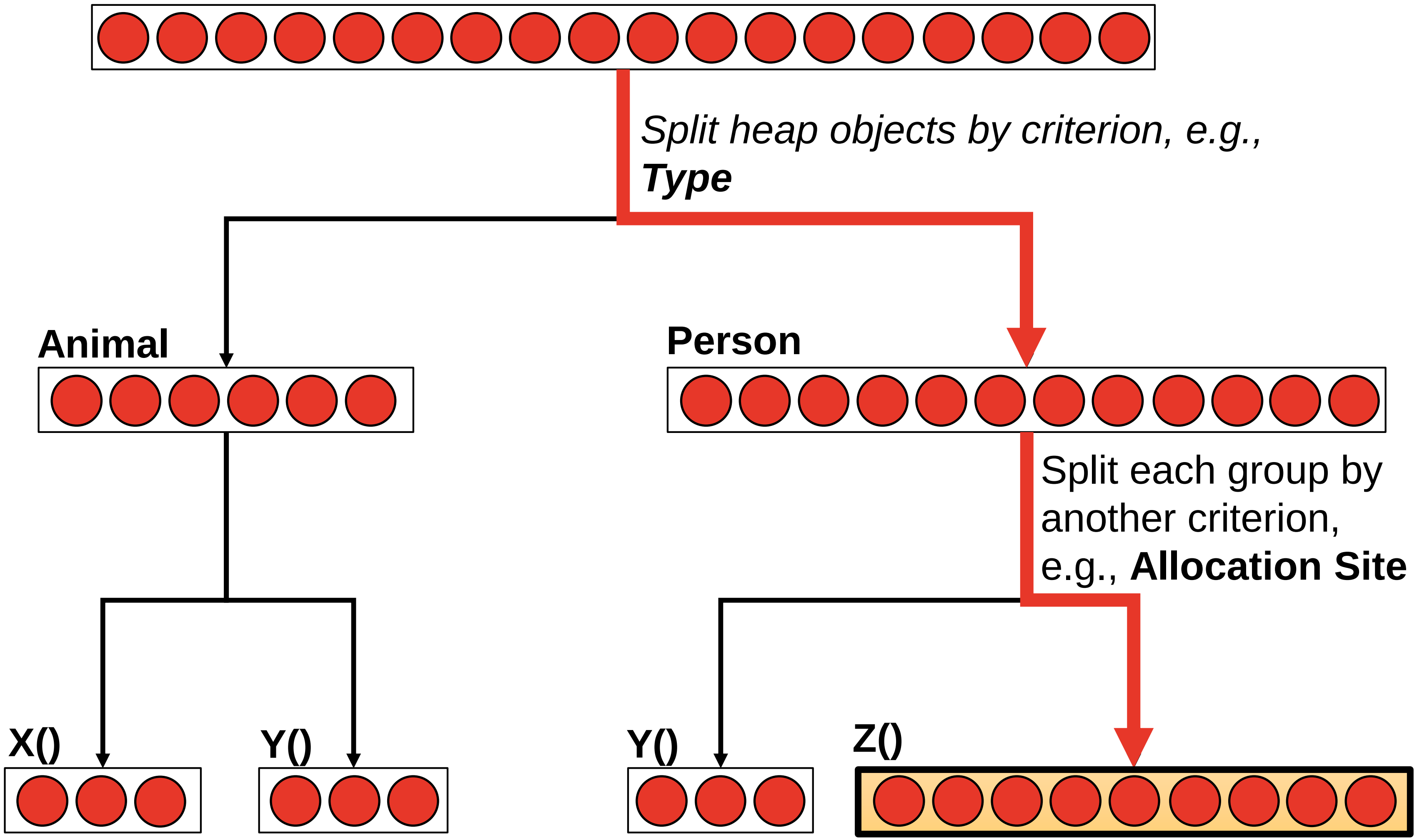


# OBJECT CLASSIFICATION

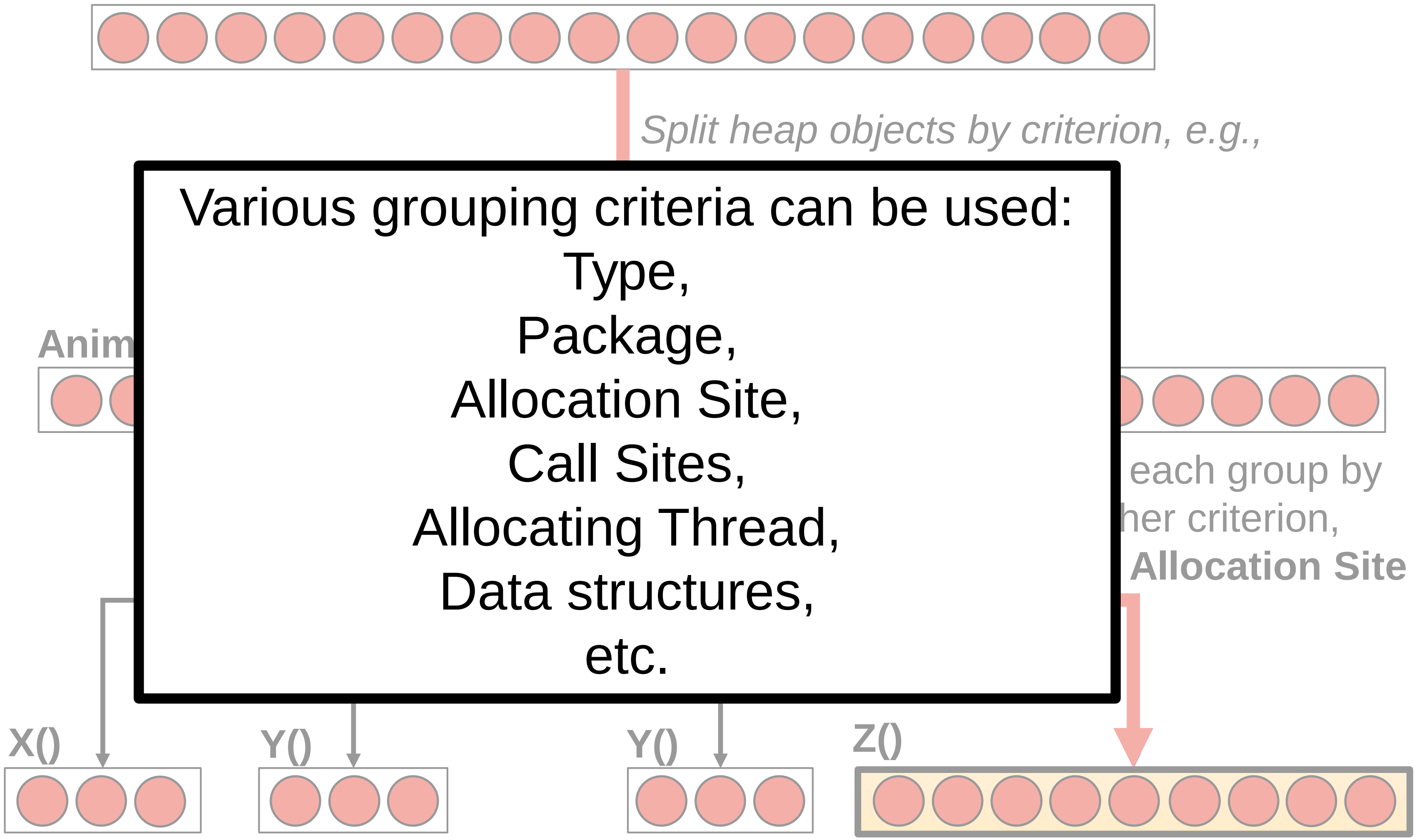



JKU

# OBJECT CLASSIFICATION



# OBJECT CLASSIFICATION




Various grouping criteria can be used:
Type,
Package,
Allocation Site,
Call Sites,
Allocating Thread,
Data structures,
etc.

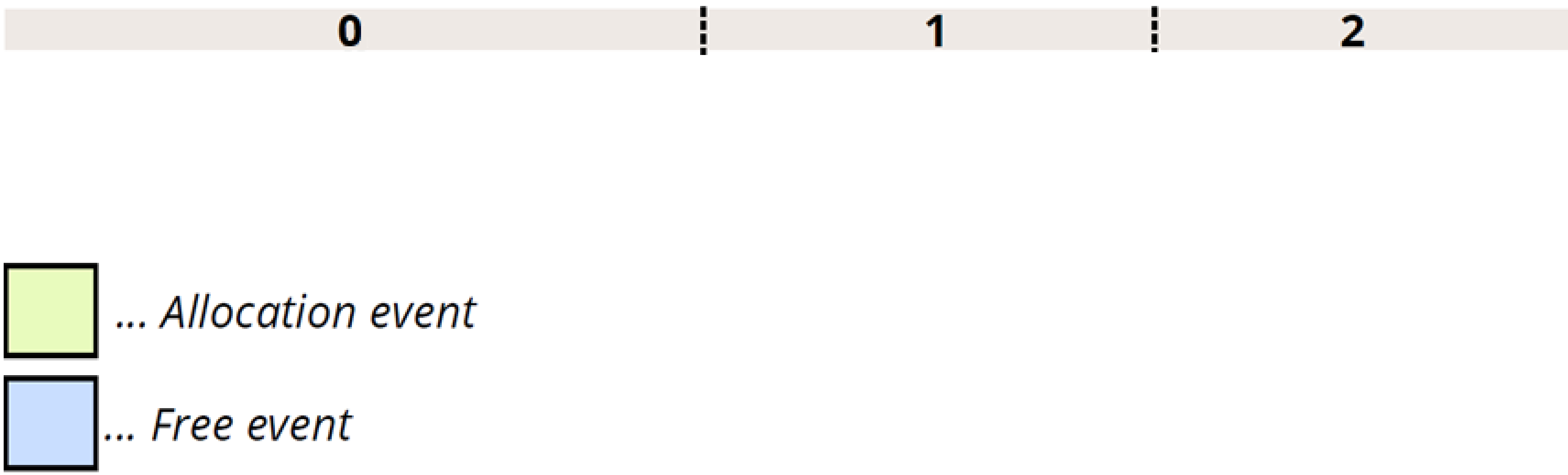
JKU

# OBJECT LIFETIME

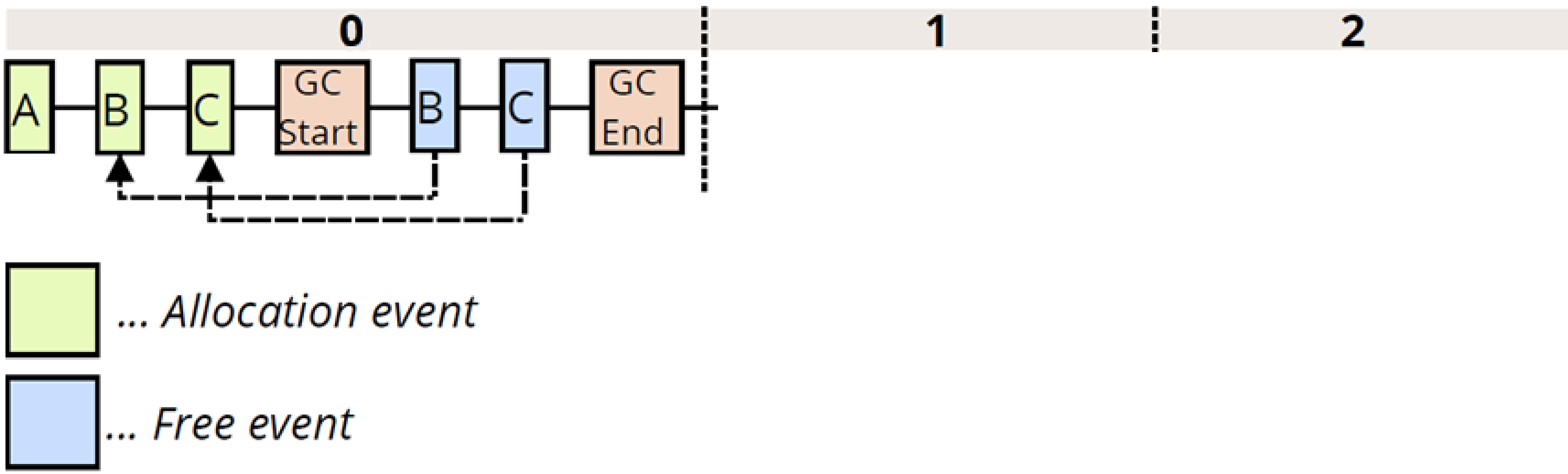
# OBJECT LIFETIME

| 0 | 1 | 2 |
|---|---|---|

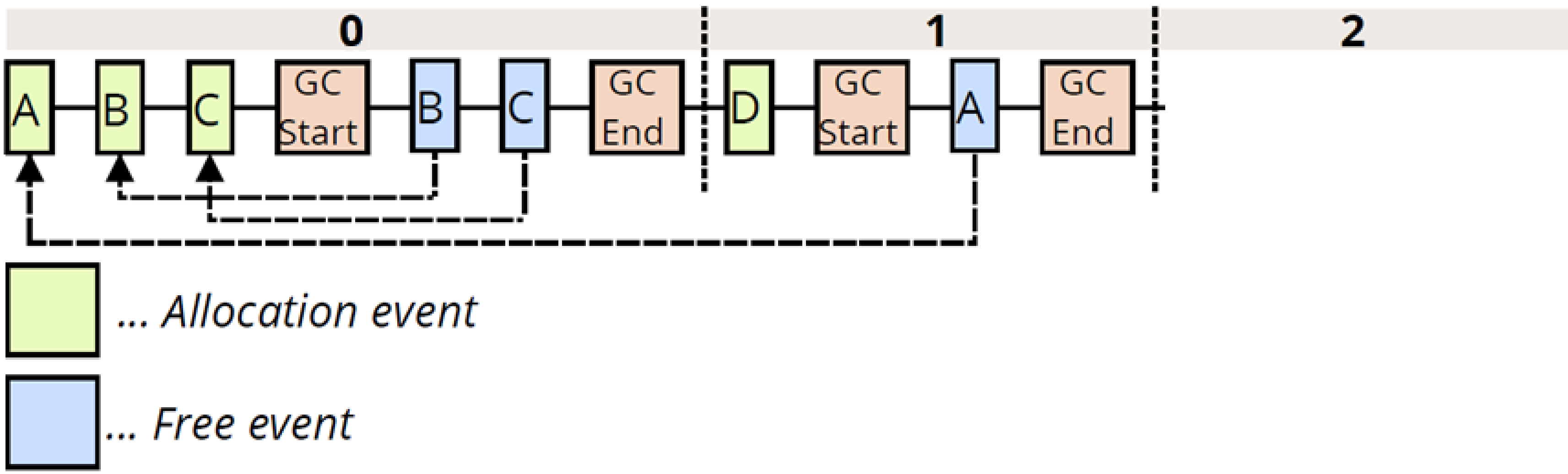
# OBJECT LIFETIME



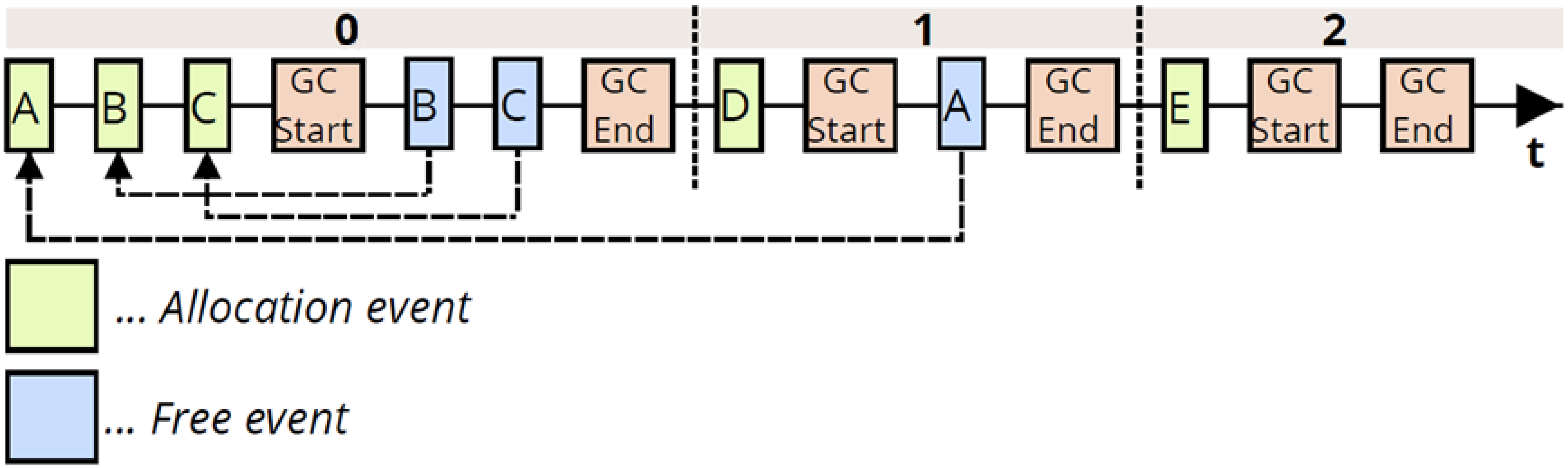
# OBJECT LIFETIME



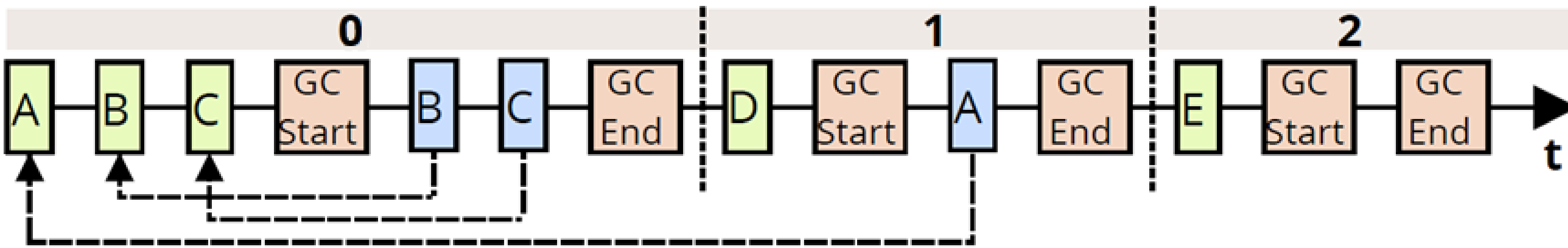
# OBJECT LIFETIME

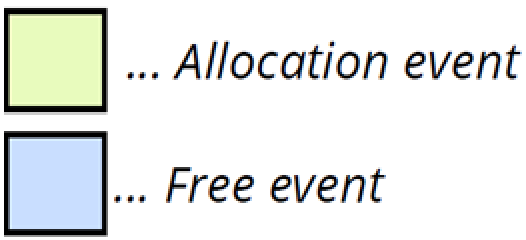


# OBJECT LIFETIME



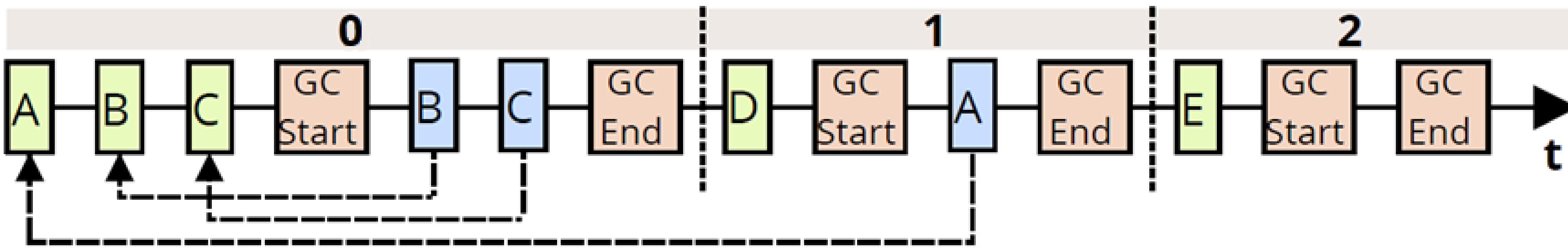
# OBJECT LIFETIME

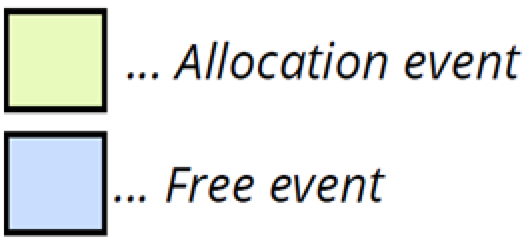



| | A | B | C | D | E |
|---|---|---|---|---|---|
| ***Birth time:*** | 0 | 0 | 0 | 1 | 2 |
| ***Free time:*** | 1 | 0 | 0 | - | - |

# OBJECT LIFETIME

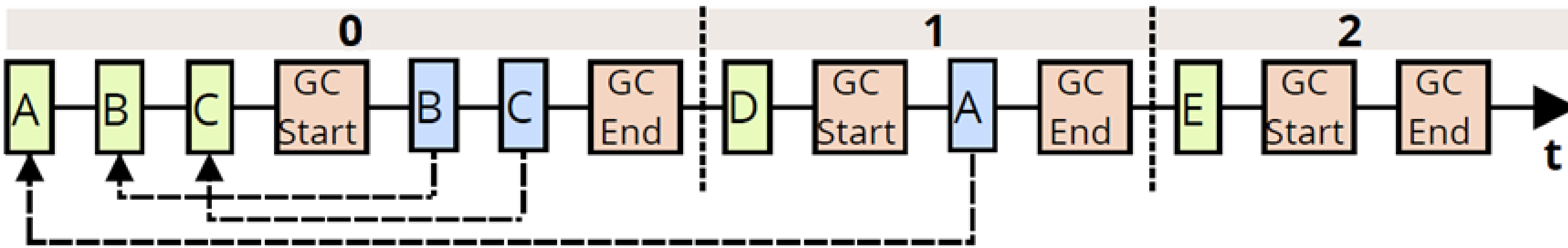



| | A | B | C | D | E |
|---|---|---|---|---|---|
| ***Birth time:*** | 0 | 0 | 0 | 1 | 2 |
| ***Free time:*** | 1 | 0 | 0 | - | - |

**Age:**

# OBJECT LIFETIME

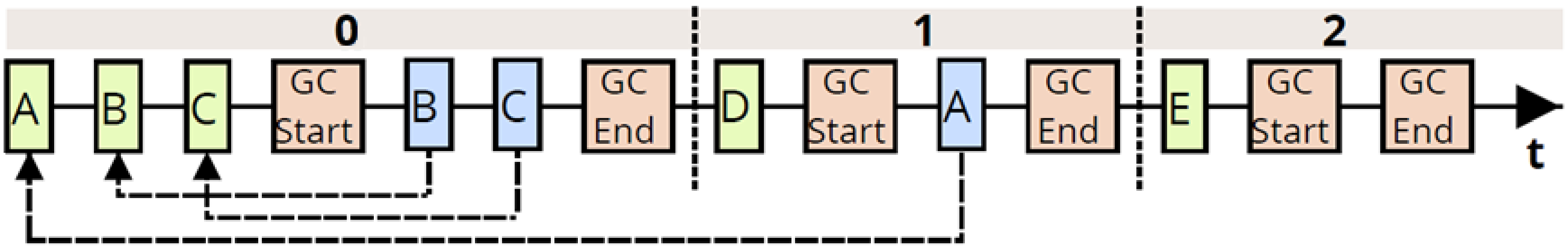


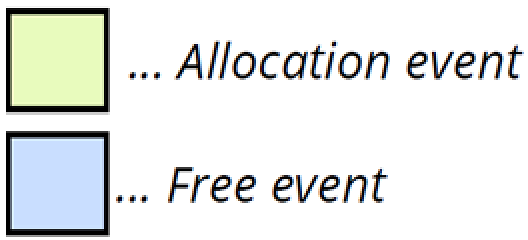

... *Allocation event*

... *Free event*

| | A | B | C | D | E |
|---|---|---|---|---|---|
| ***Birth time:*** | 0 | 0 | 0 | 1 | 2 |
| ***Free time:*** | 1 | 0 | 0 | - | - |
| **Age:** | **1** | **0** | **0** | | |

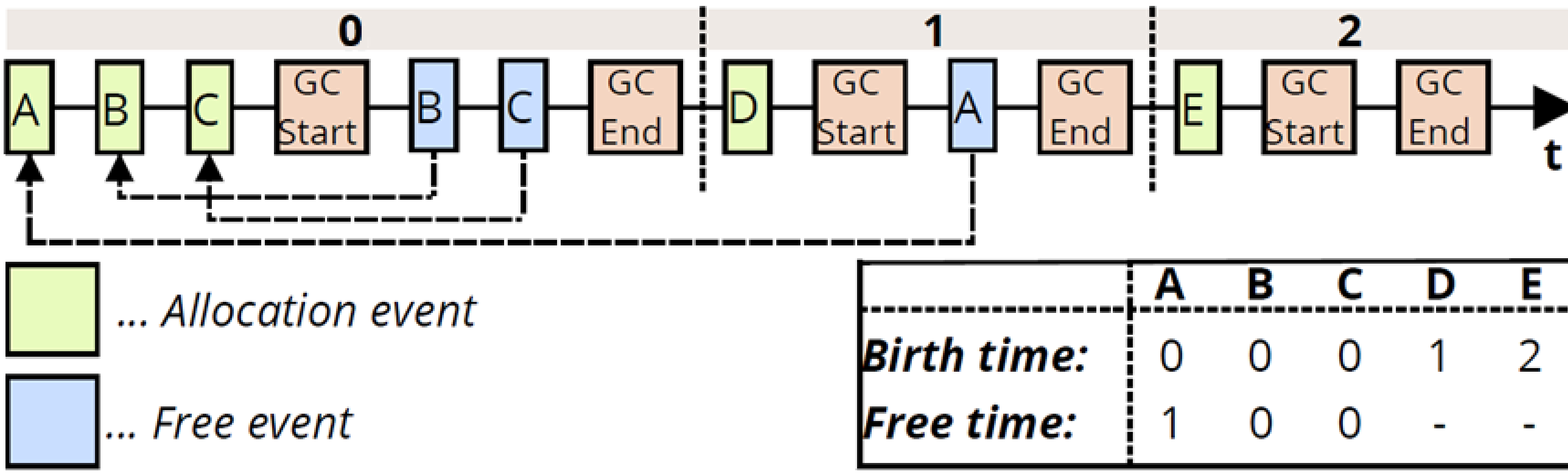
# OBJECT LIFETIME




| | A | B | C | D | E |
|---|---|---|---|---|---|
| ***Birth time:*** | 0 | 0 | 0 | 1 | 2 |
| ***Free time:*** | 1 | 0 | 0 | - | - |
| **Age:** | **1** | **0** | **0** | ***2*** | ***1*** |

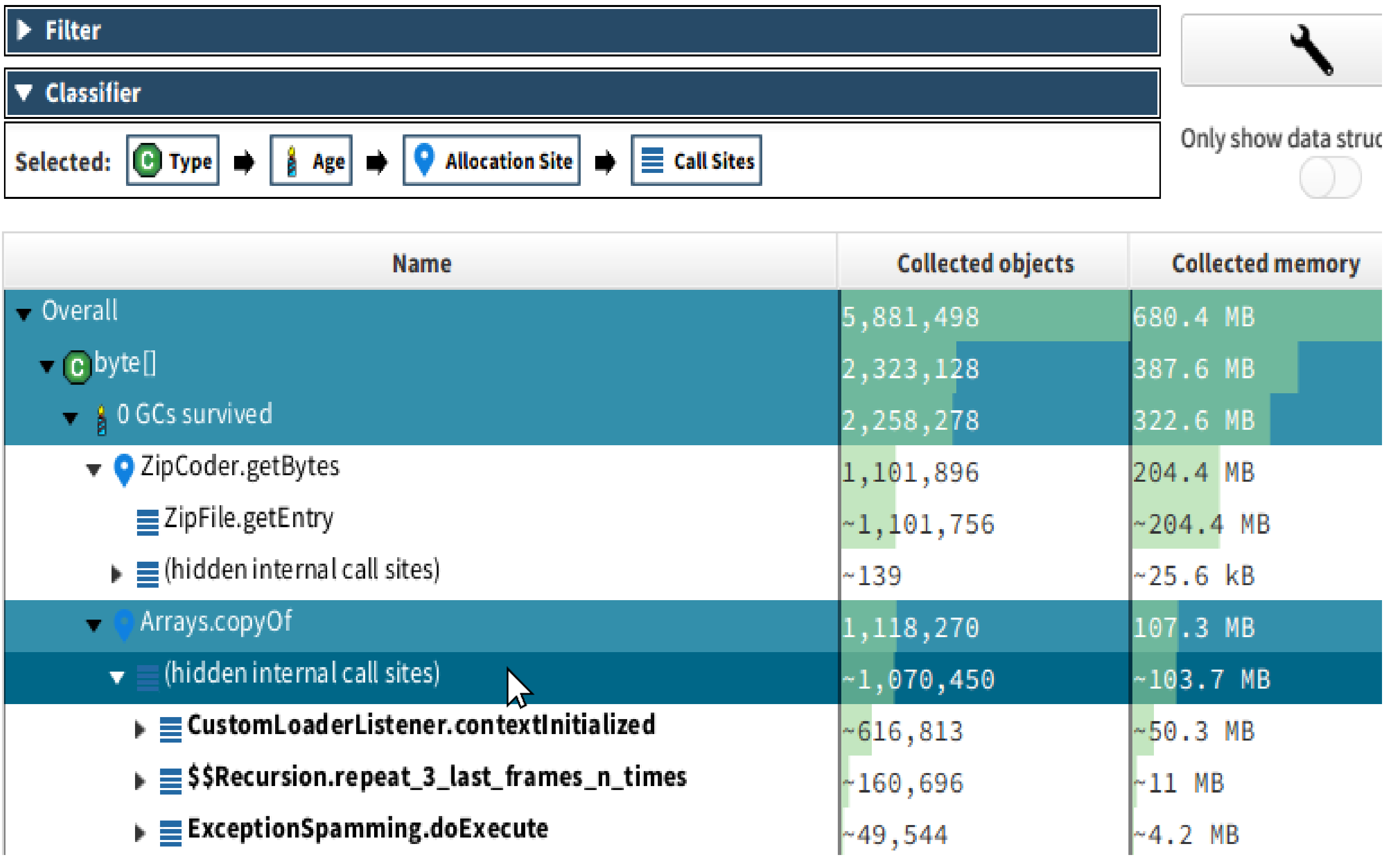
# OBJECT LIFETIME



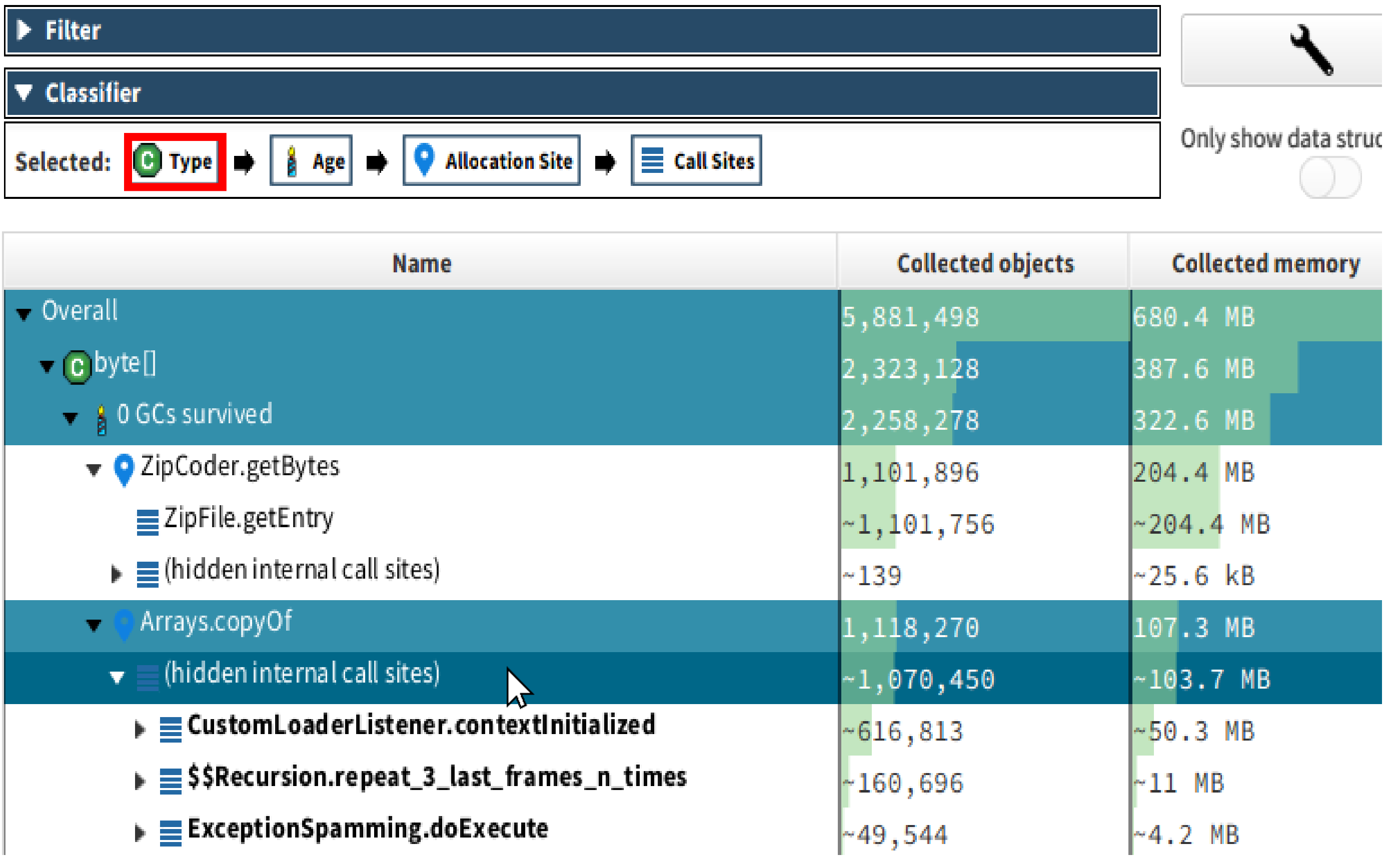

| | A | B | C | D | E |
|---|---|---|---|---|---|
| ***Birth time:*** | 0 | 0 | 0 | 1 | 2 |
| ***Free time:*** | 1 | 0 | 0 | - | - |
| **Age:** | **1** | **0** | **0** | ***2*** | ***1*** |

New classifier based on age

# DRILL-DOWN

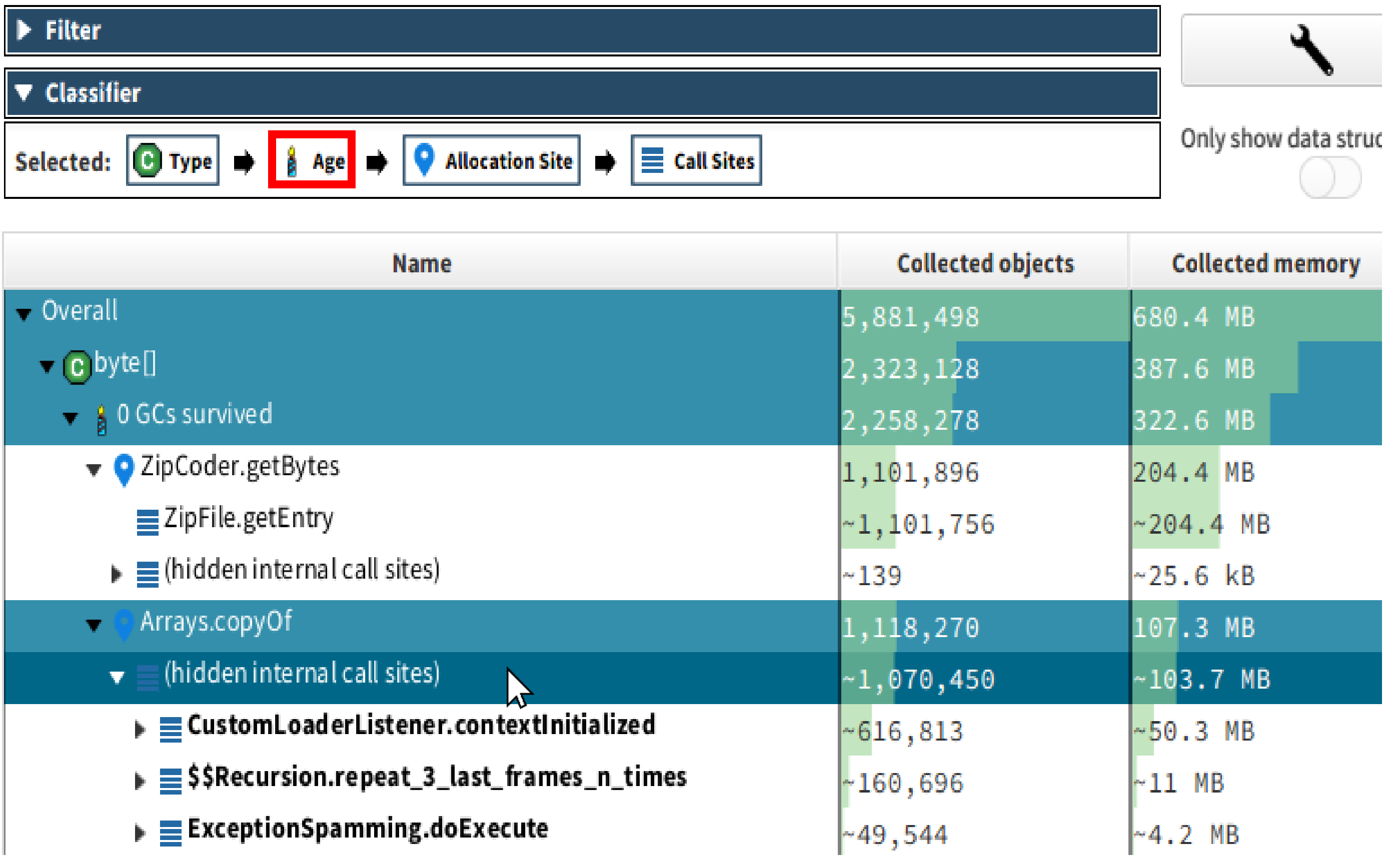

Filter
Classifier
Selected: Type Age Allocation Site Call Sites
Only show data struc
Name
Collected objects
Collected memory
Overall 5,881,498 680.4 MB
byte[] 2,323,128 387.6 MB
0 GCs survived 2,258,278 322.6 MB
ZipCoder.getBytes 1,101,896 204.4 MB
ZipFile.getEntry ~1,101,756 ~204.4 MB
(hidden internal call sites) ~139 ~25.6 kB
Arrays.copyOf 1,118,270 107.3 MB
(hidden internal call sites) ~1,070,450 ~103.7 MB
CustomLoaderListener.contextInitialized ~616,813 ~50.3 MB
$$Recursion.repeat_3_last_frames_n_times ~160,696 ~11 MB
ExceptionSpamming.doExecute ~49,544 ~4.2 MB

# DRILL-DOWN



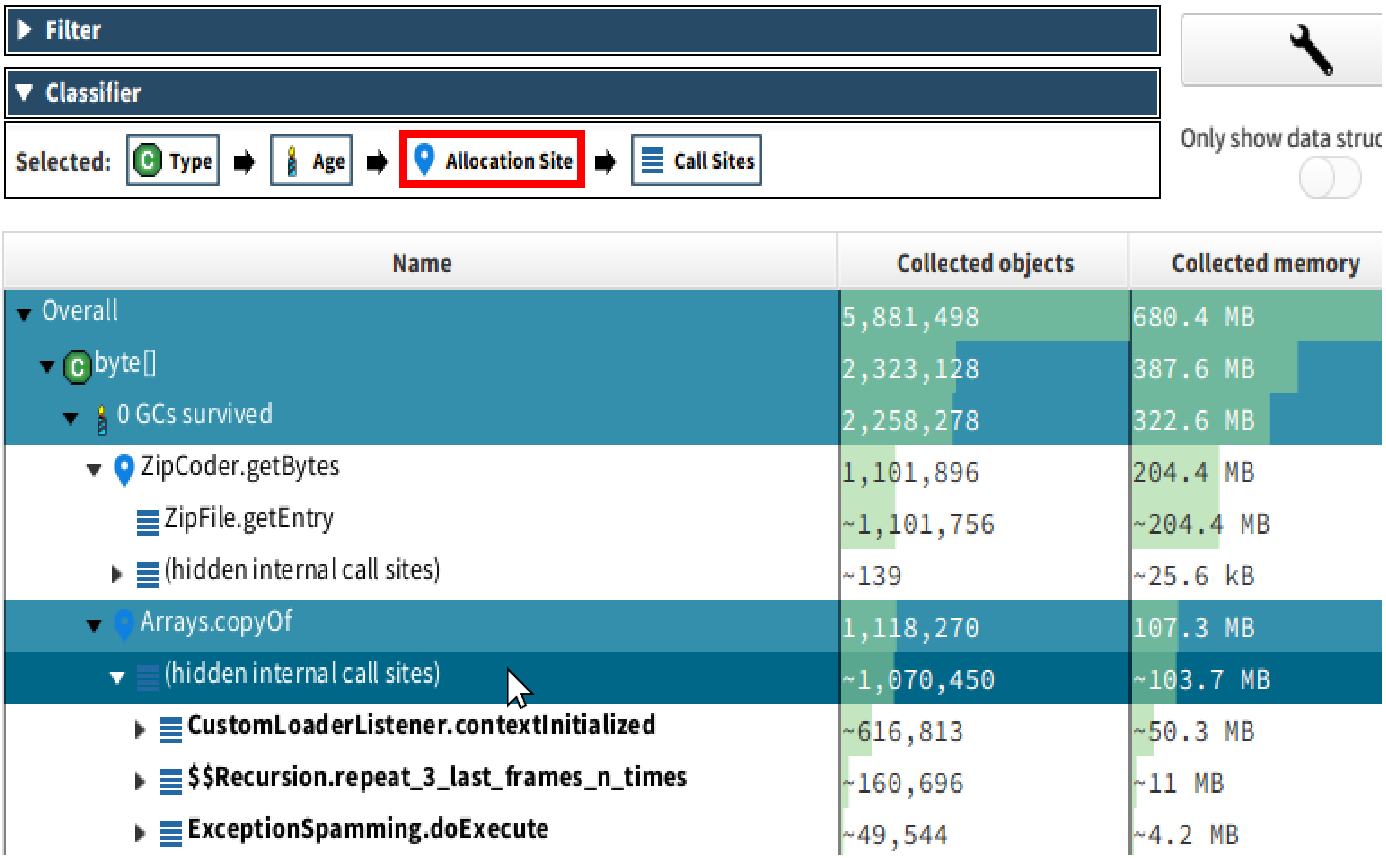
Filter
Classifier
Selected:
Type
Age
Allocation Site
Call Sites
Only show data struc
Name
Collected objects
Collected memory
Overall
5,881,498
680.4 MB
byte[]
2,323,128
387.6 MB
0 GCs survived
2,258,278
322.6 MB
ZipCoder.getBytes
1,101,896
204.4 MB
ZipFile.getEntry
~1,101,756
~204.4 MB
(hidden internal call sites)
~139
~25.6 kB
Arrays.copyOf
1,118,270
107.3 MB
(hidden internal call sites)
~1,070,450
~103.7 MB
CustomLoaderListener.contextInitialized
~616,813
~50.3 MB
$$Recursion.repeat_3_last_frames_n_times
~160,696
~11 MB
ExceptionSpamming.doExecute
~49,544
~4.2 MB

# DRILL-DOWN



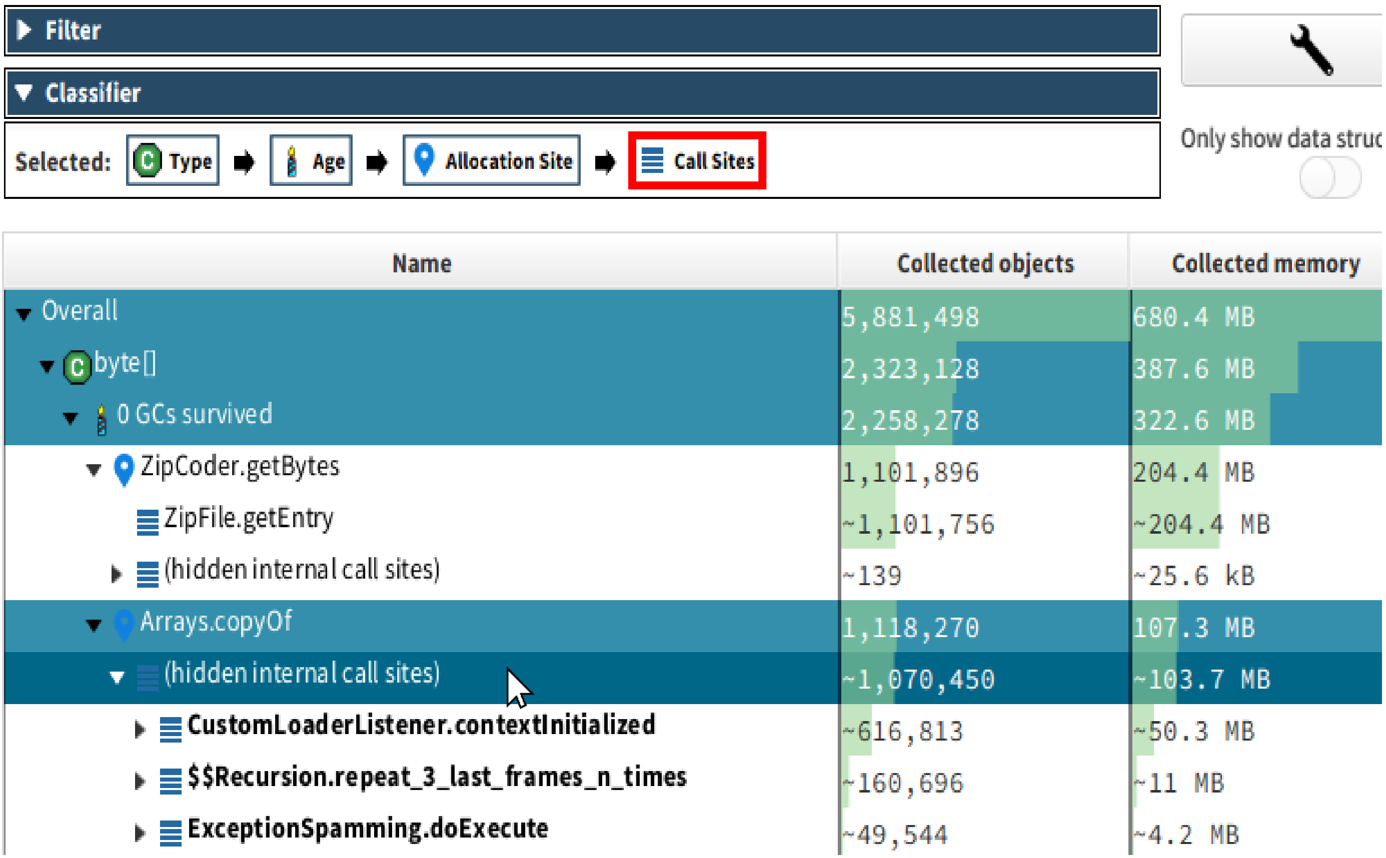
Filter
Classifier
Selected: Type Age Allocation Site Call Sites
Only show data struc
Name
Collected objects
Collected memory
Overall 5,881,498 680.4 MB
byte[] 2,323,128 387.6 MB
0 GCs survived 2,258,278 322.6 MB
ZipCoder.getBytes 1,101,896 204.4 MB
ZipFile.getEntry ~1,101,756 ~204.4 MB
(hidden internal call sites) ~139 ~25.6 kB
Arrays.copyOf 1,118,270 107.3 MB
(hidden internal call sites) ~1,070,450 ~103.7 MB
CustomLoaderListener.contextInitialized ~616,813 ~50.3 MB
$$Recursion.repeat_3_last_frames_n_times ~160,696 ~11 MB
ExceptionSpamming.doExecute ~49,544 ~4.2 MB

# DRILL-DOWN

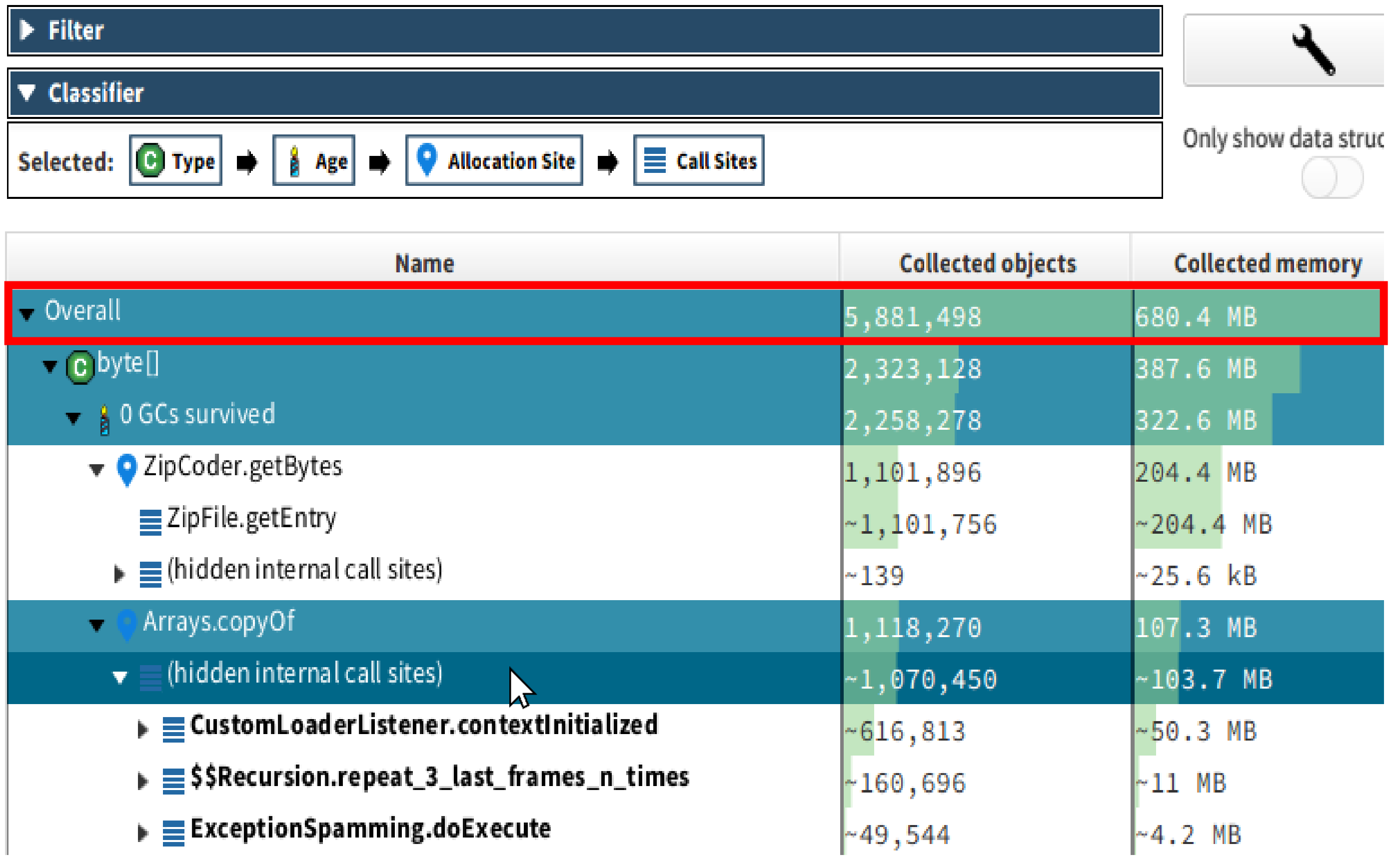



JKU

# DRILL-DOWN

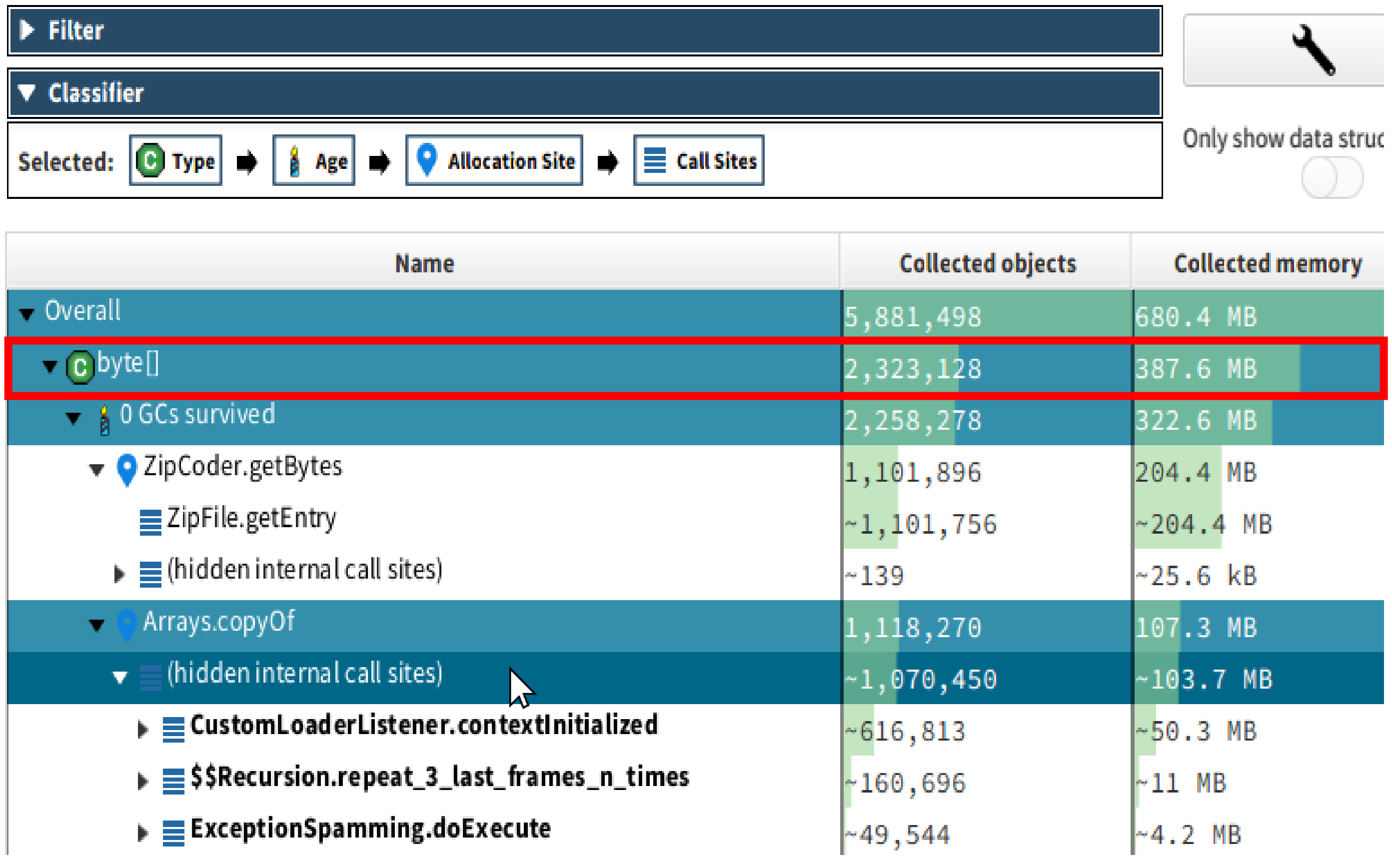

Filter
Classifier
Selected: Type Age Allocation Site Call Sites
Only show data struc
Name
Collected objects
Collected memory
Overall 5,881,498 680.4 MB
byte[] 2,323,128 387.6 MB
0 GCs survived 2,258,278 322.6 MB
ZipCoder.getBytes 1,101,896 204.4 MB
ZipFile.getEntry ~1,101,756 ~204.4 MB
(hidden internal call sites) ~139 ~25.6 kB
Arrays.copyOf 1,118,270 107.3 MB
(hidden internal call sites) ~1,070,450 ~103.7 MB
CustomLoaderListener.contextInitialized ~616,813 ~50.3 MB
$$Recursion.repeat_3_last_frames_n_times ~160,696 ~11 MB
ExceptionSpamming.doExecute ~49,544 ~4.2 MB

# DRILL-DOWN




...garbage over analyzed time window

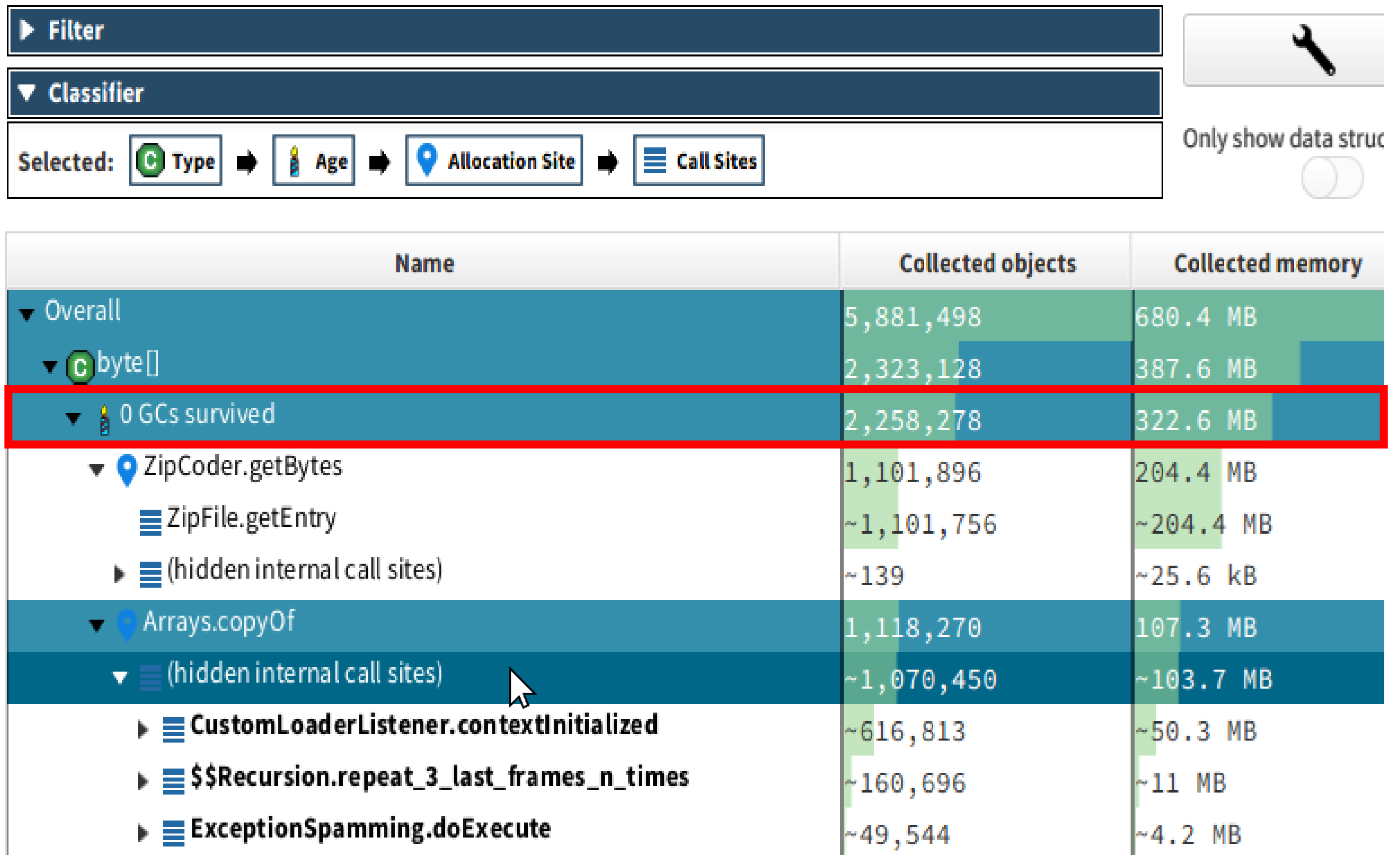
# DRILL-DOWN




...garbage over analyzed time window

…of which are `byte` arrays

# DRILL-DOWN



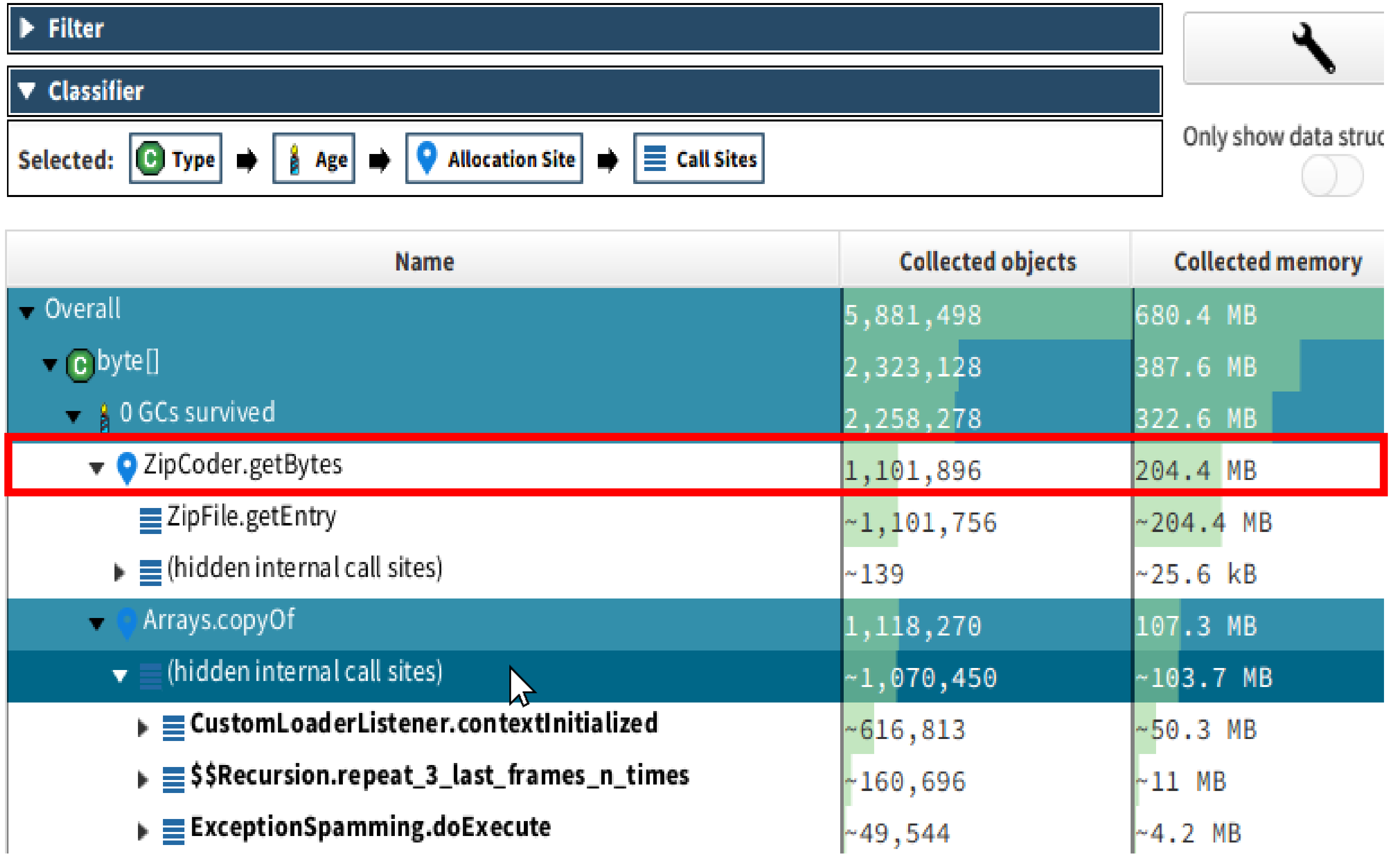

...garbage over analyzed time window

…of which are `byte` arrays

…of which survived no GC

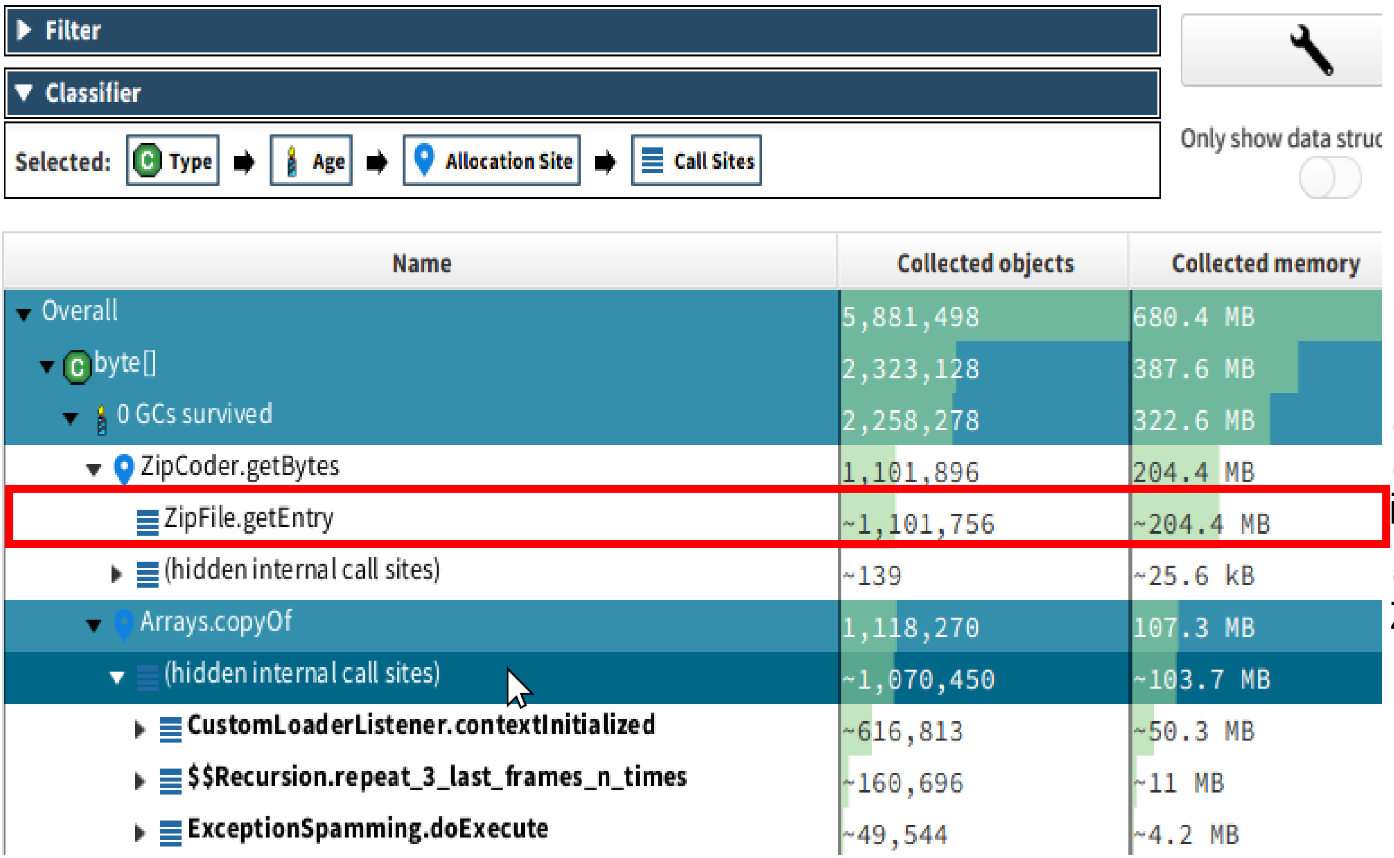
JKU

# DRILL-DOWN




...garbage over analyzed time window
…of which are `byte` arrays
…of which survived no GC
…of which were allocated in `ZipCoder.getBytes`

# DRILL-DOWN




...garbage over analyzed time window

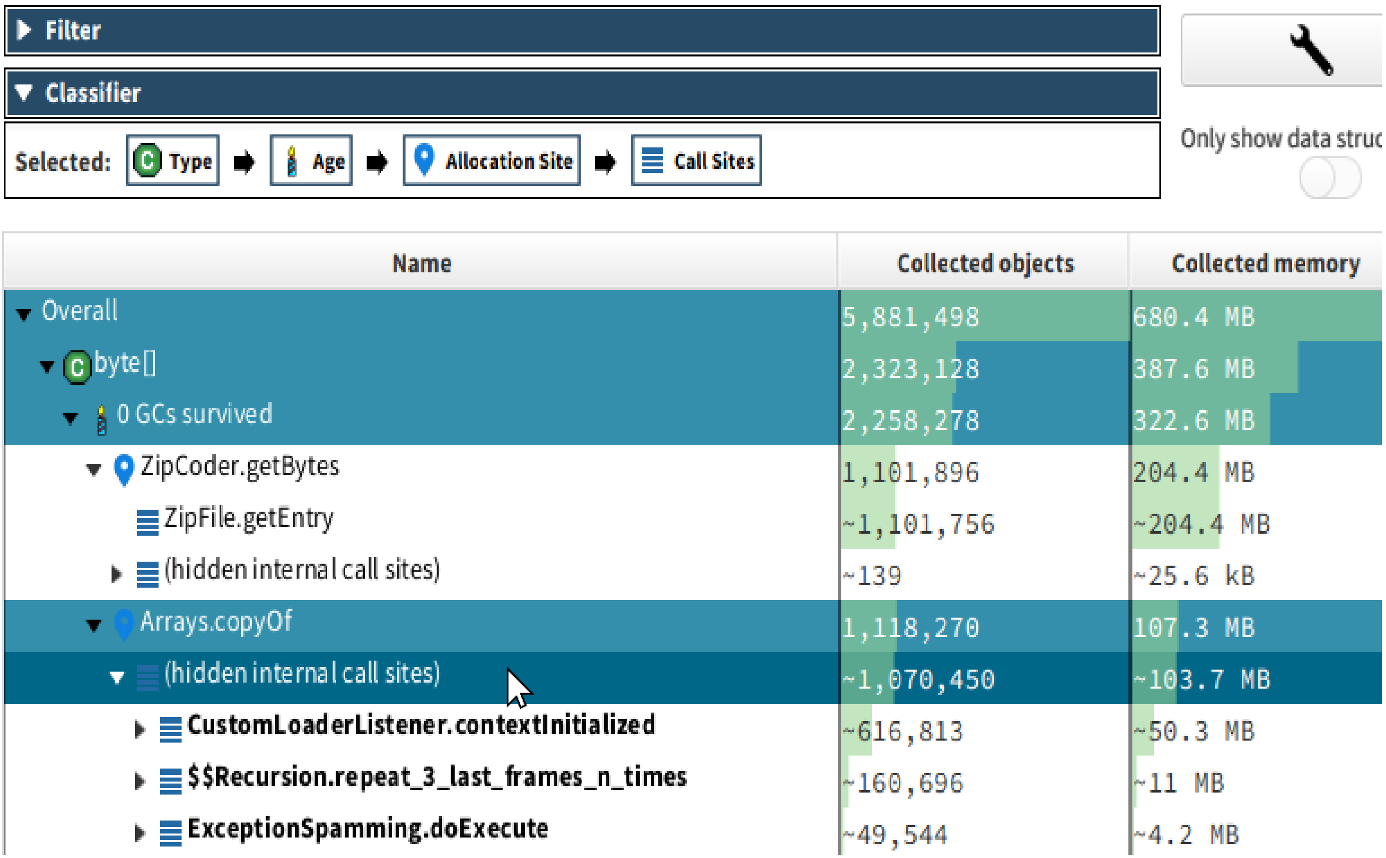
…of which are `byte` arrays

…of which survived no GC

…of which were allocated in `ZipCoder.getBytes`

…while it was called by `ZipFile.getEntry`

# DRILL-DOWN




...garbage over analyzed time window

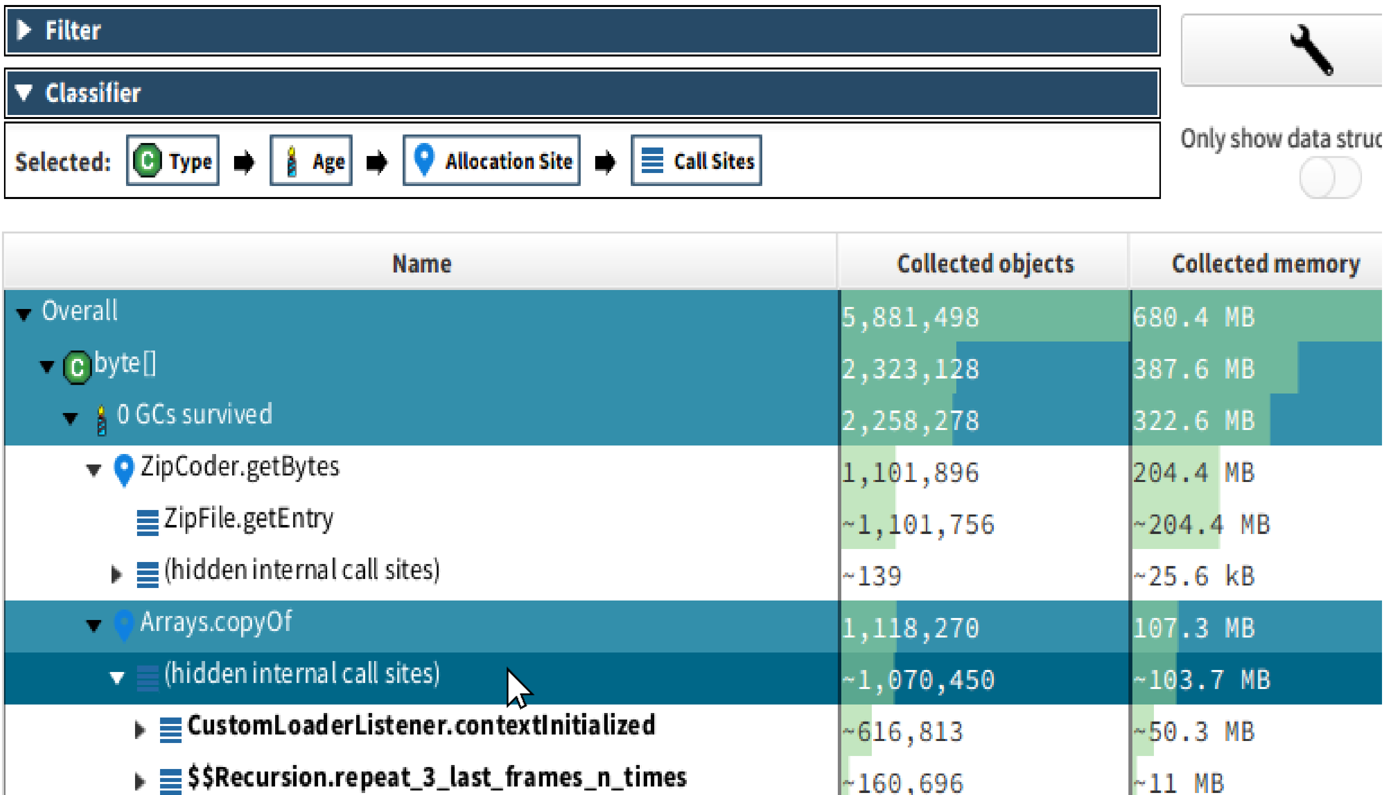
…of which are `byte` arrays

…of which survived no GC

…of which were allocated in `ZipCoder.getBytes`

…while it was called by `ZipFile.getEntry`

# DRILL-DOWN




...garbage over analyzed time window

…of which are `byte` arrays

…of which survived no GC

…of which were allocated in `ZipCoder.getBytes`

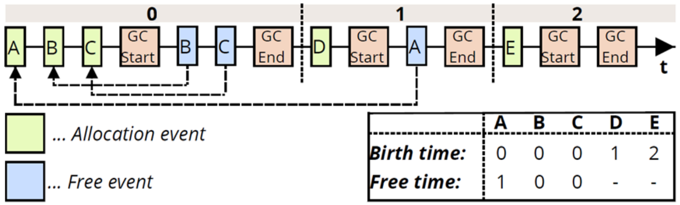
…while it was called by `ZipFile.getEntry`

**Open IDE and check whether the number of allocations can be reduced.**

# FUTURE WORK

# FUTURE WORK

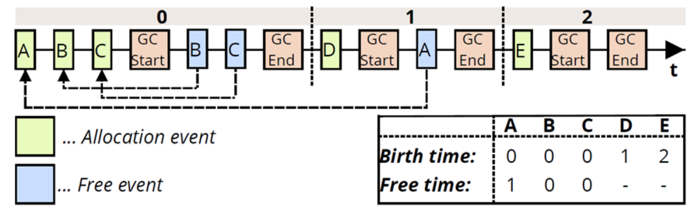
- Use lifetime information in other analyses




| | A | B | C | D | E |
|---|---|---|---|---|---|
| ***Birth time:*** | 0 | 0 | 0 | 1 | 2 |
| ***Free time:*** | 1 | 0 | 0 | - | - |

# FUTURE WORK

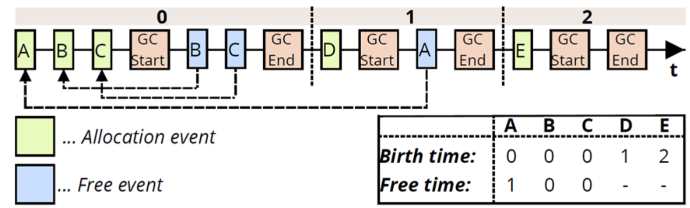
- Use lifetime information in other analyses




- Guidance

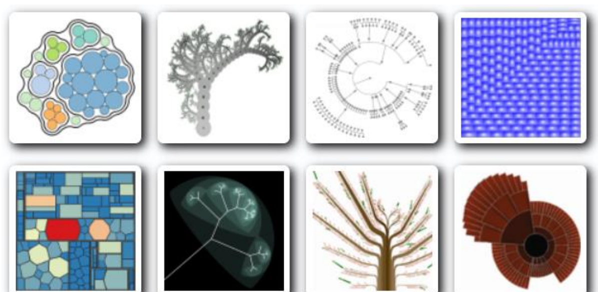
# FUTURE WORK

- Use lifetime information in other analyses




- Guidance

- Visualization



# TAKE-AWAYS

# TAKE-AWAYS

**Problem**

**High memory churn**

**Freq. allocations**

**Freq. garbage collections**

# TAKE-AWAYS

| Problem | Memory Churn Hotspot |
| --- | --- |
| High memory churn<br><br>Freq. allocations<br><br>Freq. garbage collections | Detect time window with highest garbage per second |

# TAKE-AWAYS

| Problem | Memory Churn Hotspot | Object Lifetime |
| --- | --- | --- |
| High memory churn<br><br>Freq. allocations<br><br>Freq. garbage collections | Detect time window with highest garbage per second | Birth time<br><br>Free time<br><br>Age<br><br>New grouping classifier |

JKU

# TAKE-AWAYS

| Problem | Memory Churn Hotspot | Object Lifetime | Inspection |
| --- | --- | --- | --- |
| High memory churn<br><br>Freq. allocations<br><br>Freq. garbage collections | Detect time window with highest garbage per second | Birth time<br><br>Free time<br><br>Age<br><br>New grouping classifier | Which objects die without survinging a single GC?<br><br>Type<br><br>Allocation Site |

JKU

# TAKE-AWAYS

| Problem | Memory Churn Hotspot | Object Lifetime | Inspection |
|---|---|---|---|
| High memory churn<br><br>Freq. allocations<br><br>Freq. garbage collections | Detect time window with highest garbage per second | Birth time<br><br>Free time<br><br>Age<br><br>New grouping classifier | Which objects die without survinging a single GC?<br><br>Type<br><br>Allocation Site |

**Markus Weninger**

Johannes Kepler University
Linz, Austria

markus.weninger@jku.at

JKU