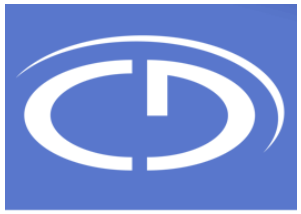
ANALYZING THE EVOLUTION OF DATA STRUCTURES IN TRACE-BASED MEMORY MONITORING

Markus Weninger
Elias Gander
Hanspeter Mössenböck

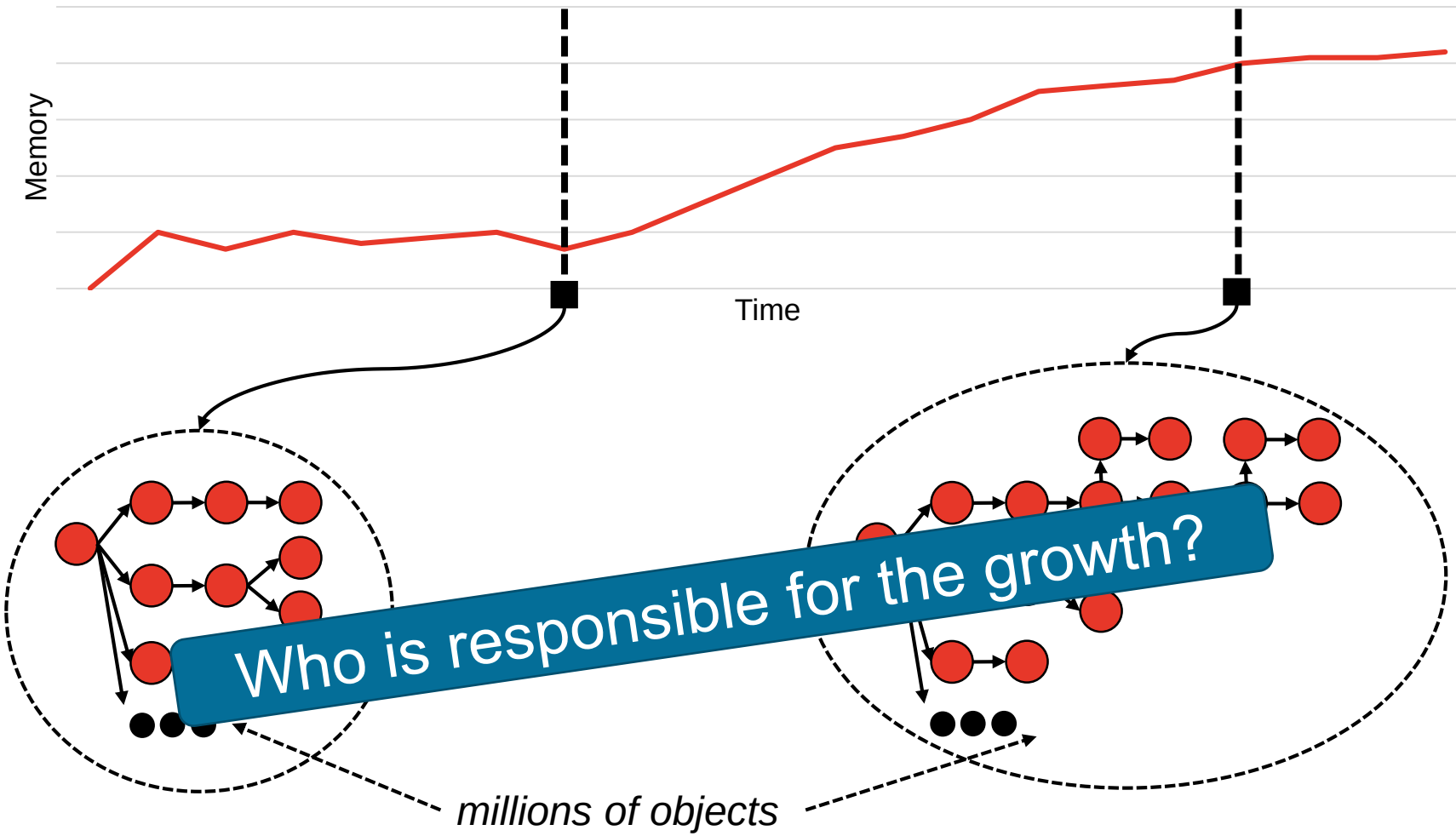
SSP 2018 – Hildesheim, Germany

JOHANNES KEPLER
UNIVERSITY LINZ



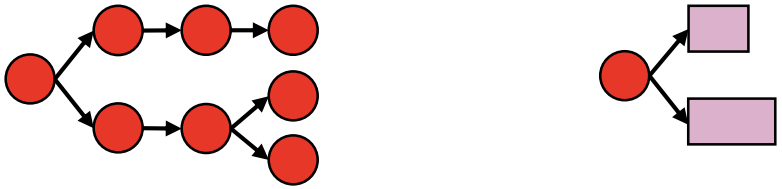
dynatrace

MOTIVATION

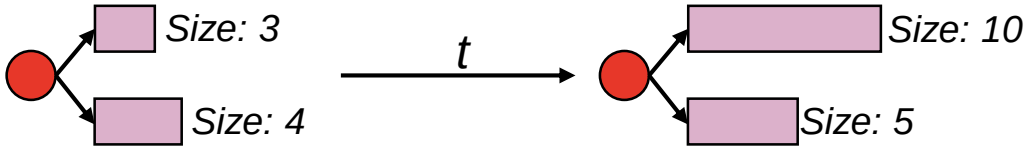

Memory
Time
Who is responsible for the growth?
millions of objects

APPROACH

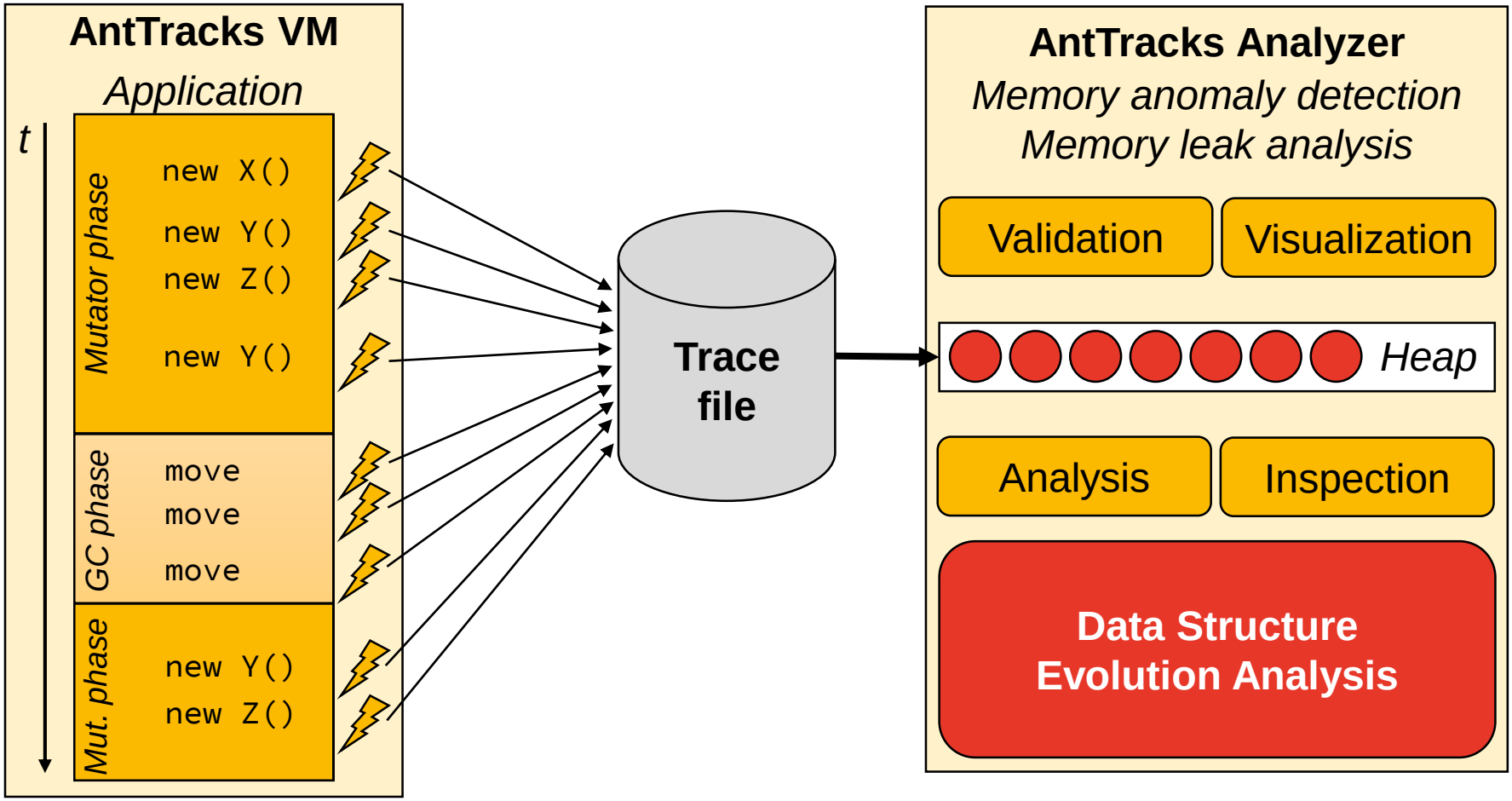
- Analysis on the object-level hard to impossible
  - Millions of objects
  - **Abstraction** needed → **Data structures**



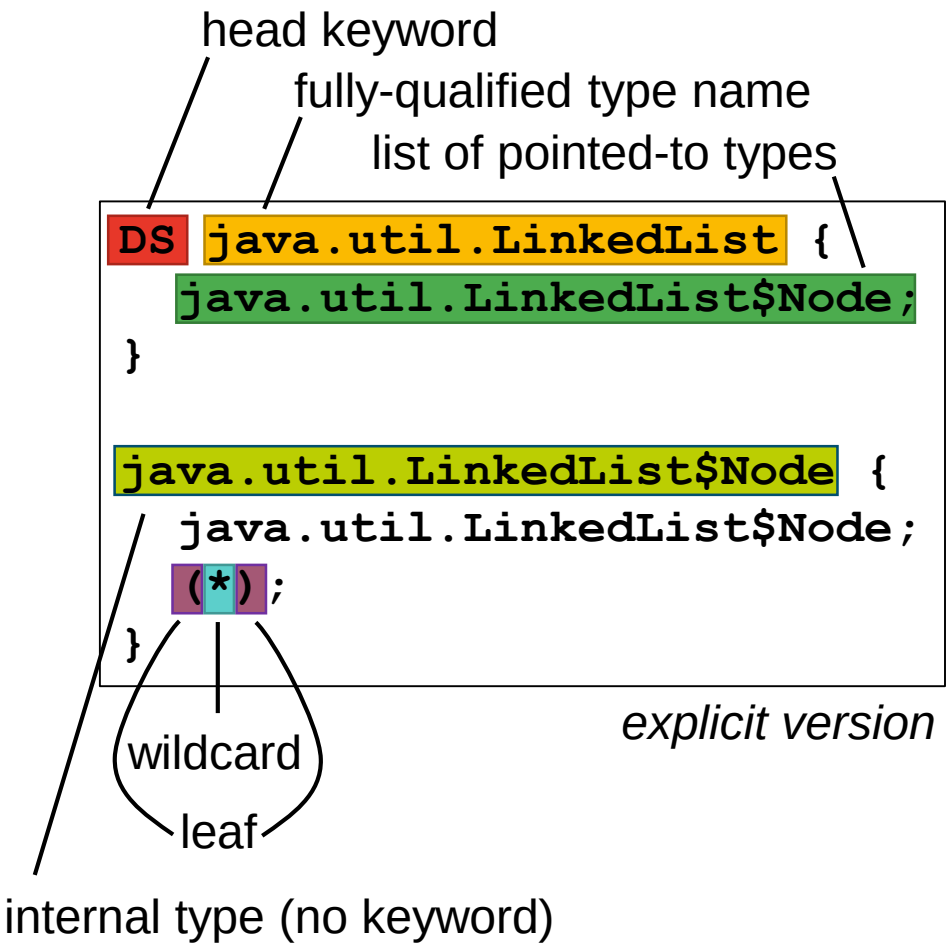
- **Evolution** over time may be the key to detect memory leaks
  - But how to obtain the needed information?
    → **Trace-based memory monitoring**
- **Combine** data structure information and temporal information
  - Quantify growth? → **Metrics**



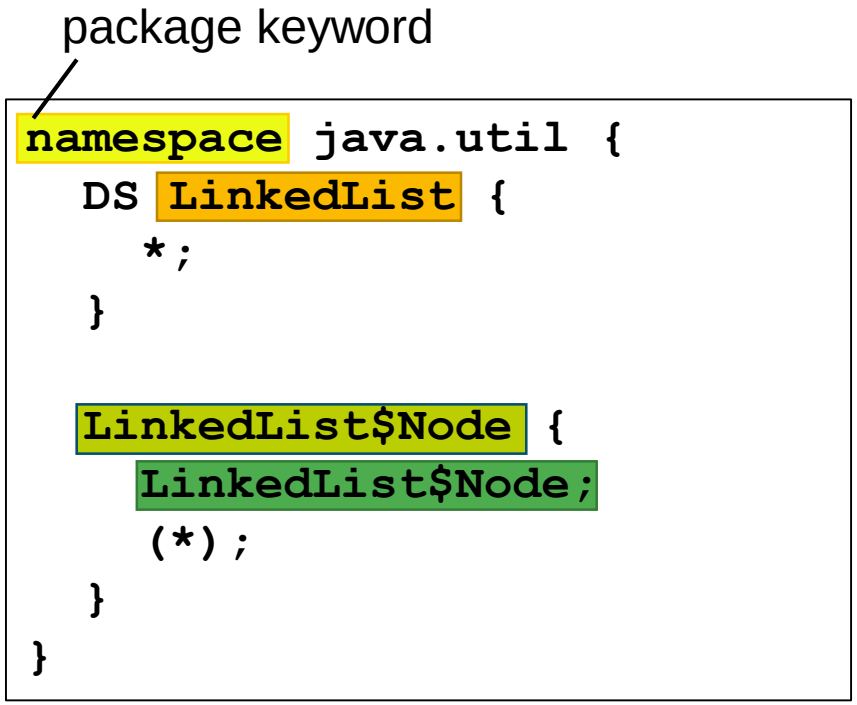
SYSTEM OVERVIEW



DATA STRUCTURE DESCRIPTION



explicit version



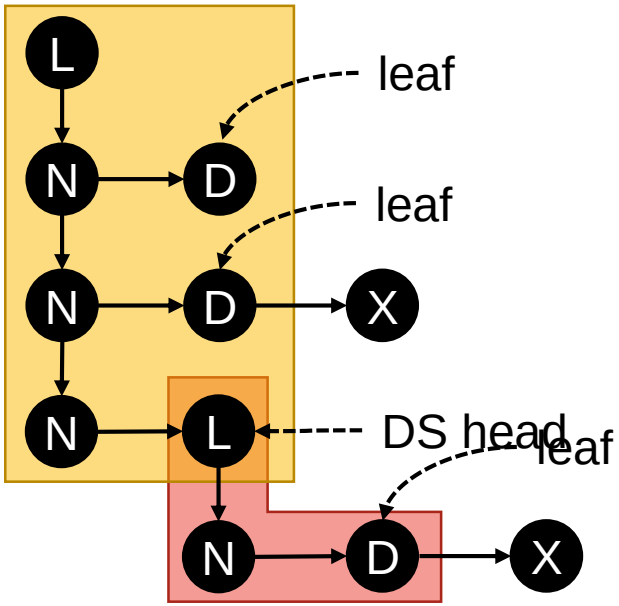
namespace + wildcard version

DATA STRUCTURE DETECTION

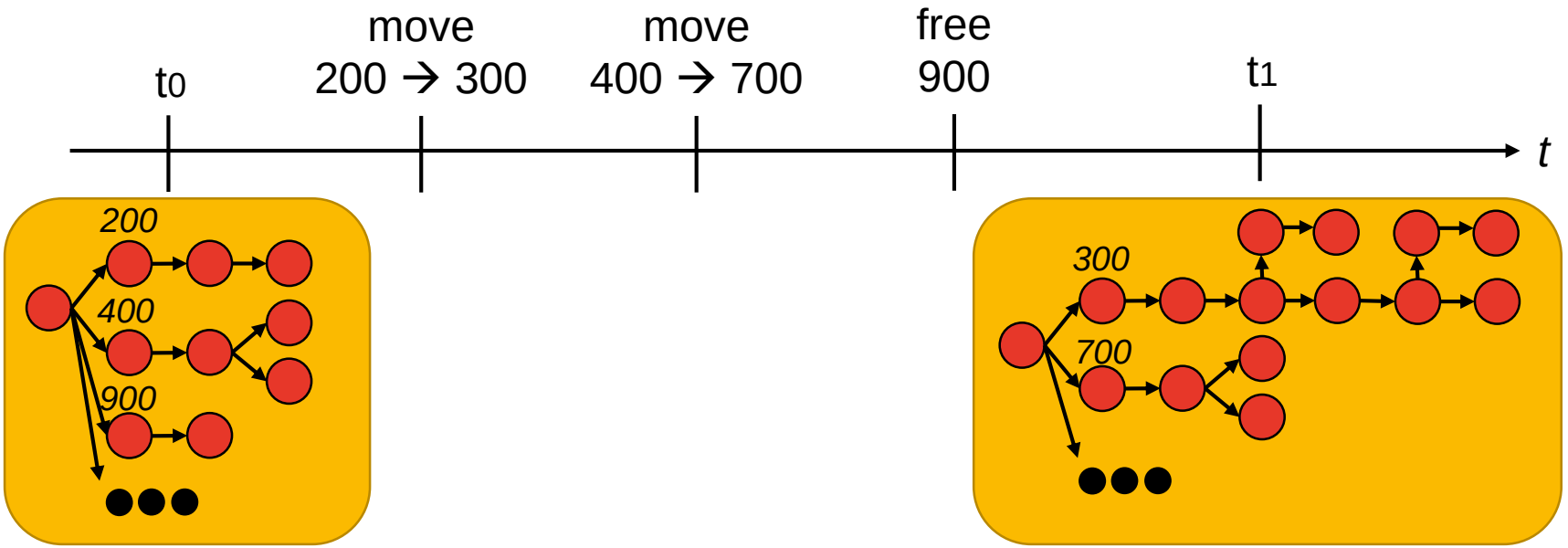
- Detect all head objects
- Start at heads and
  - visit referenced object if its type is part of pointed-to types
    - Follow references if
      - (1) its type is part of non-leaf pointed-to types *and*
      - *(2)* it is not a data structure head itself

```
DS java.util.LinkedList {
  java.util.LinkedList$Node;
}

java.util.LinkedList$Node {
  java.util.LinkedList$Node;
  (*);
}
```



DATA STRUCTURE TRACKING



start	cur
200	200
400	400
900	900
...	...

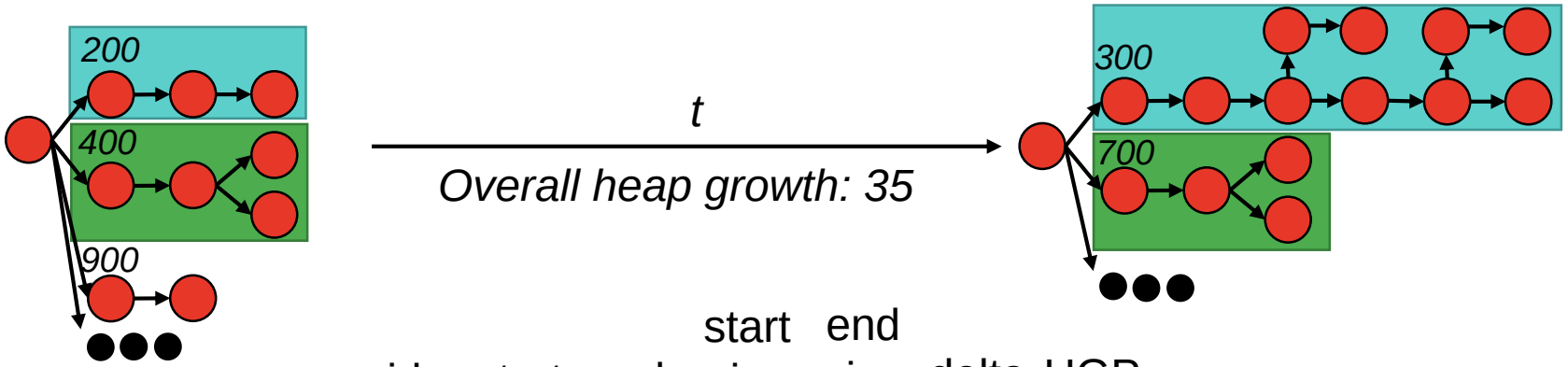
start	cur
200	300
400	400
900	900
...	...

start	cur
200	300
400	700
900	900
...	...

start	cur
200	300
400	700
...	...

start	end
200	300
400	700
...	...

METRICS



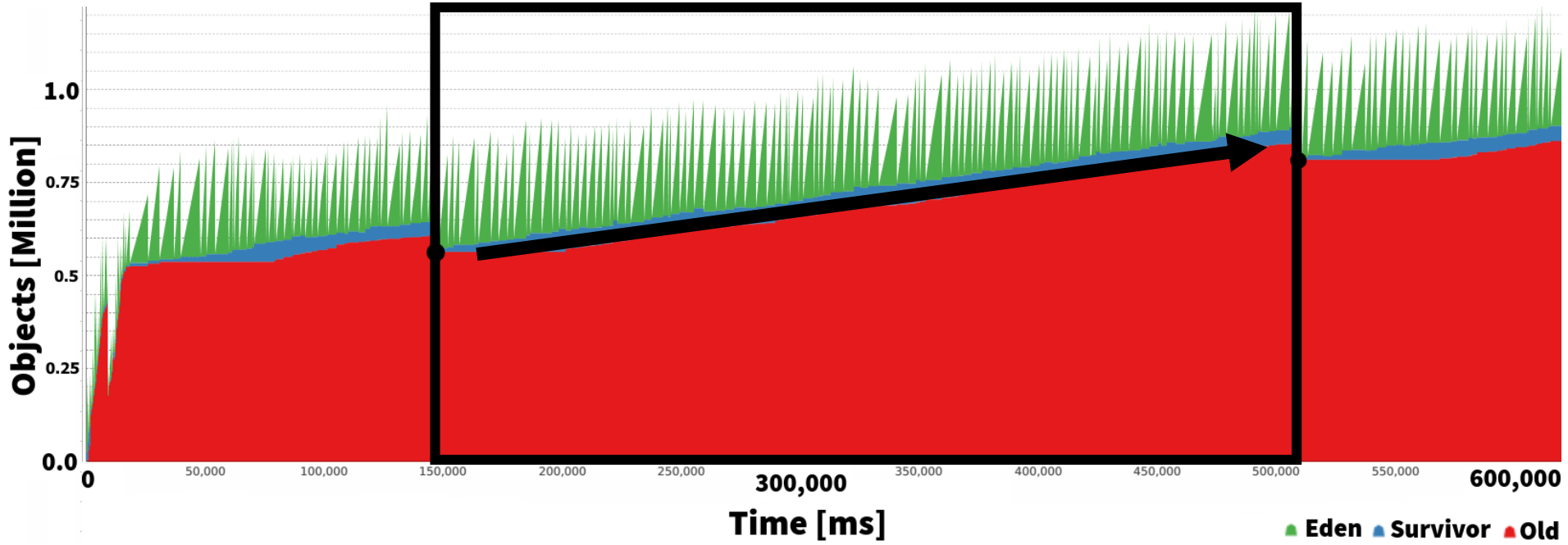
id	start	end	start size	end size	delta	HGP
1	200	300	3	10	+7	
2	400	700	4	4	+0	0
...	...	...	...	...	...	...

Is this a lot?

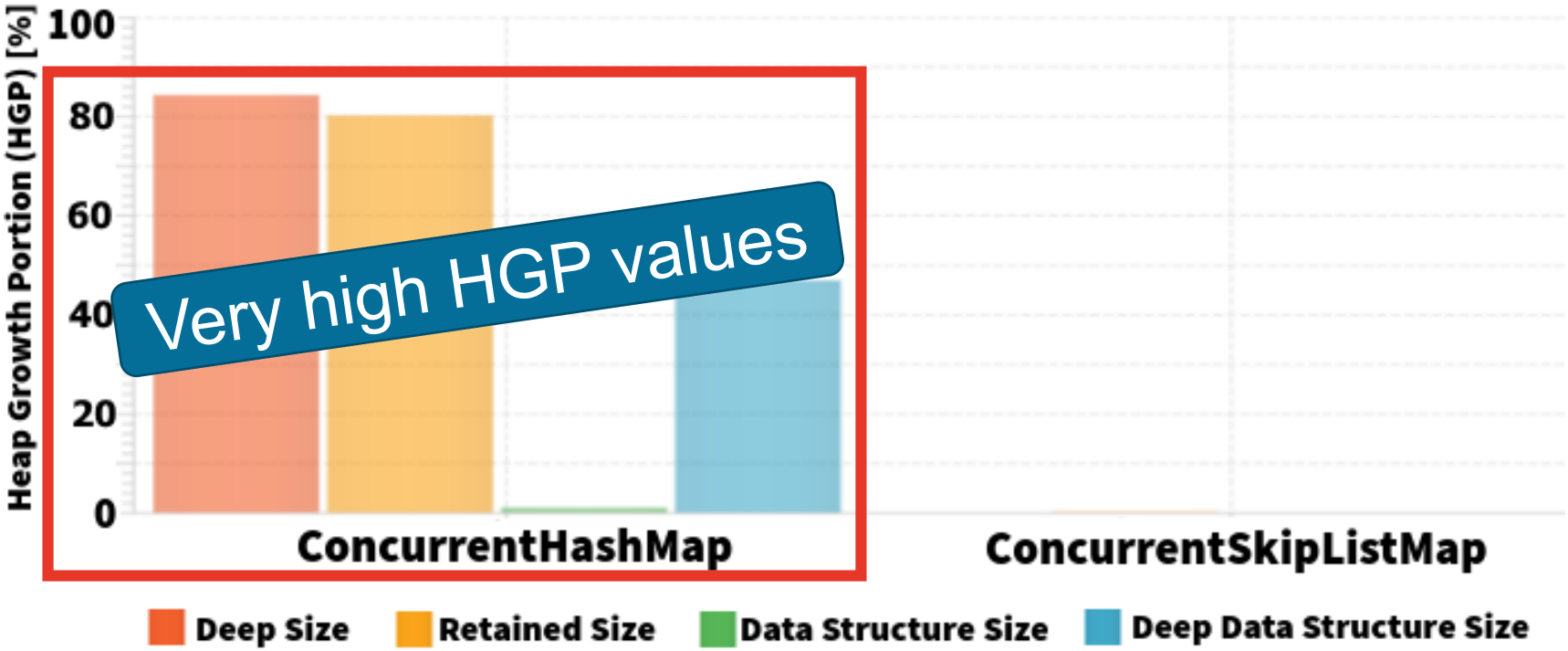
Heap growth portion in %

Sorting by this metric highlights the most conspicious DS

EXAMPLE: BROKEN CACHE



EXAMPLE: BROKEN CACHE



EXAMPLE: BROKEN CACHE

Name	Objects	
	Before	After
Overall	25,748	25,748
java.util.concurrent.ConcurrentHashMap [4,253,574,384]	1	1
JourneyService.findLocations(String, int, boolean) : Location[] : 106	1	1
Data structure leaves	3[illegible],058	13[illegible],224
Location	31,781	135,026
GeneratedConstructorAccessor33.newInstance(Object[]) : Object : 0	31,769	135,014
Constructor.newInstance(Object[]) : Object : -1	12	12
JourneyService$QueryKey	276	1,196
JourneyService.findLocations(String, int, boolean) : Location[] : 128	276	1,196
ConcurrentHashMap$CounterCell	0	1
Collections$EmptyList	1	1

Check JourneyService.findLocations()

FUTURE WORK

- Graph-based visualization
  - Aggregation based on data structure information
- Automatic data structure descriptions
  - Using static + dynamic information (such as analyzing the evolution of points-to graphs)
- Automatic pattern detection
  - Detect certain combinations of metrics
  - Automatically apply classification steps

TAKE-AWAYS

Description	Detection	Tracking	Metrics & Visualization
DSL + Namespace + Wildcards + Leafs Describes pointed-to relation	Recursive descent Allows to detect DS composition	Track DSs through application's run time Stop tracking died DSs	Quantify growth (transitive, ...) Sort and select based on growth