ACDC4JS

How to analyze a JavaScript garbage collector

Mario Preishuber

Department of Computer Sciences
University of Salzburg

January 13, 2014

Related work

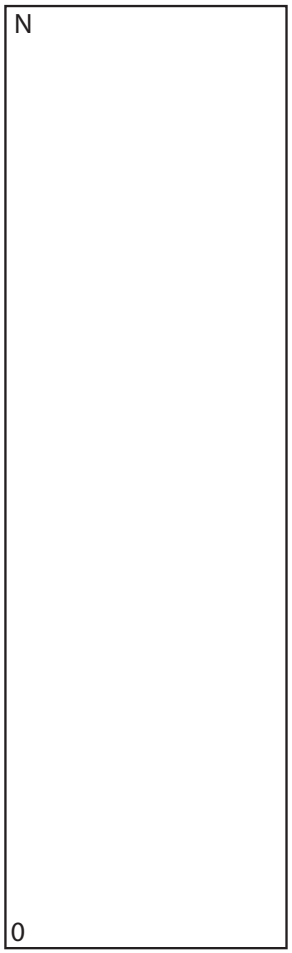
ACDC: Towards a Universal Mutator for Benchmarking Heap Management Systems

Authors
Martin Aigner
Christoph Kirsch

Memory management

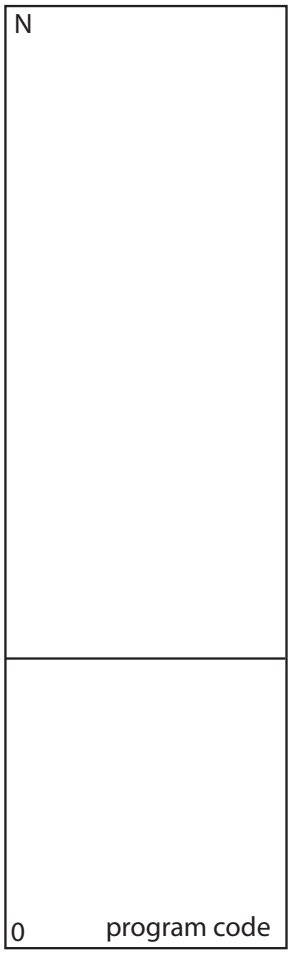
- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack

Memory management



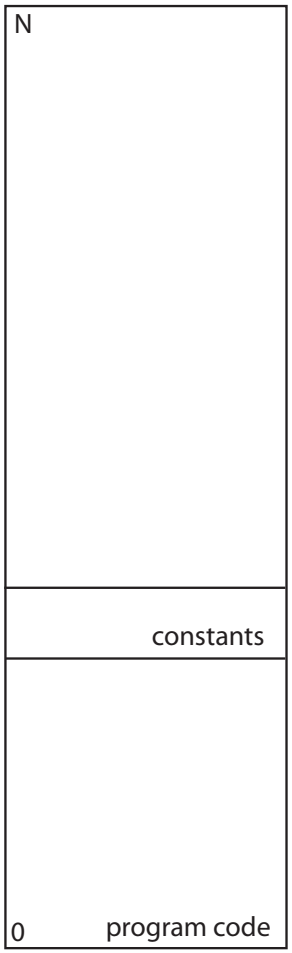
- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack

Memory management



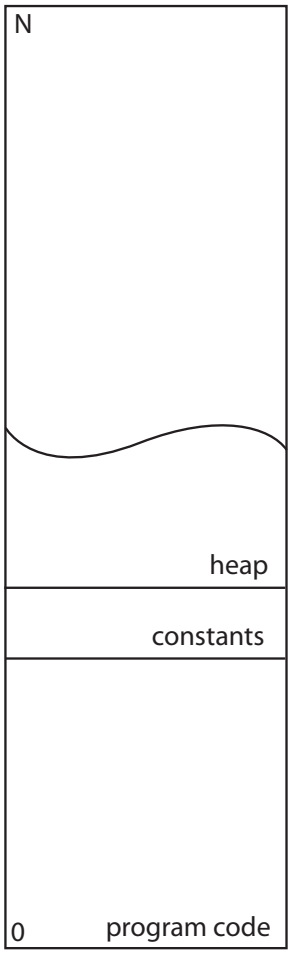
- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack

Memory management



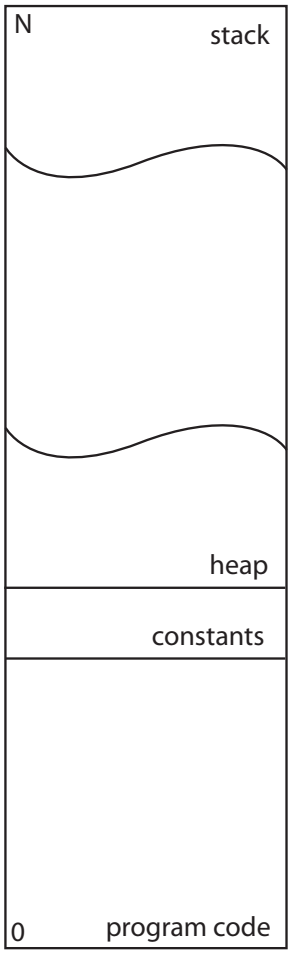
- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack

Memory management



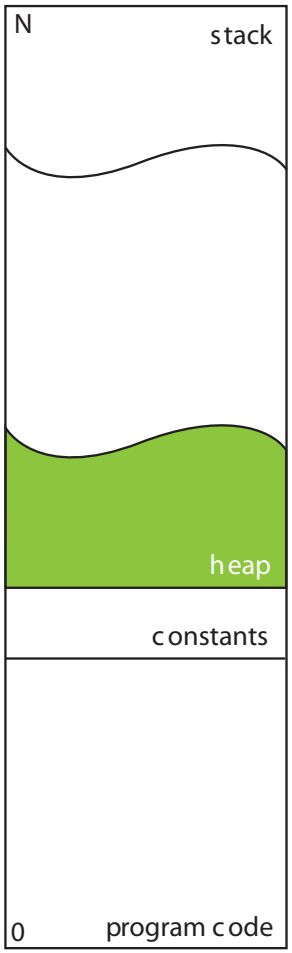
- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack

Memory management



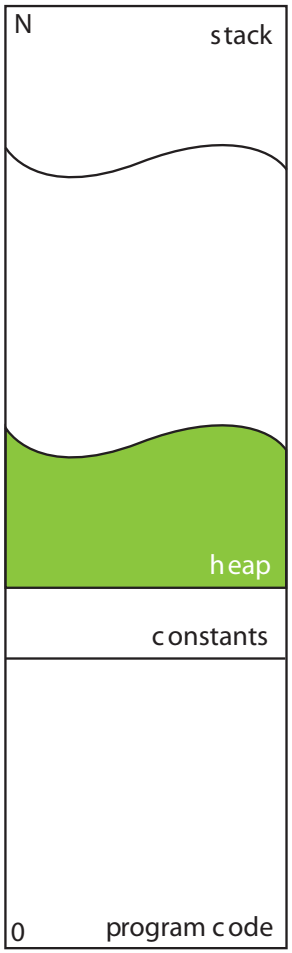
- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack

MEMORY MANAGEMENT



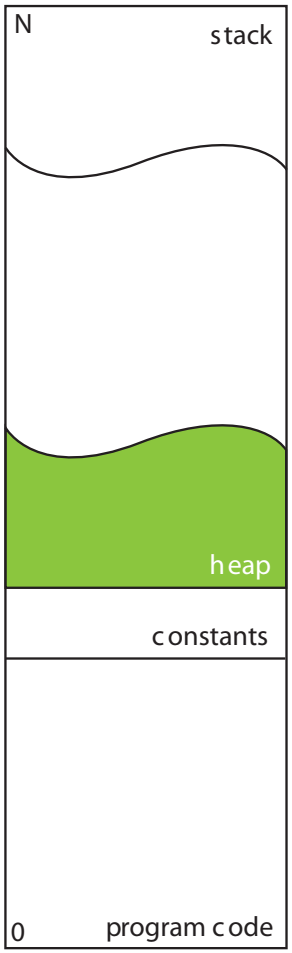
- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack

Memory management



- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack
- Memory management in C
  - **Allocation** explicit with `malloc()`
  - **Deallocation** explicit with `free()`

Memory management



- Typical memory structure
  1. Program code
  2. Constants (e.g. strings)
  3. Heap
  4. Stack
- Memory management in C
  - **Allocation** explicit with `malloc()`
  - **Deallocation** explicit with `free()`
- Memory management in JavaScript
  - **Allocation** explicit / implicit
  - **Deallocation** implicit

Memory management

```
1  // C
2  char *s = malloc(4*sizeof(char)); // explicit allocation
3  strncpy(s, "abc\0", 4);
4  ...
5  access(s);
6  ...
7  free(s);                          // explicit deallocation
```

Memory management

```c
// C
char *s = malloc(4*sizeof(char)); // explicit allocation
strncpy(s, "abc\0", 4);
...
access(s);
...
free(s);                          // explicit deallocation
```

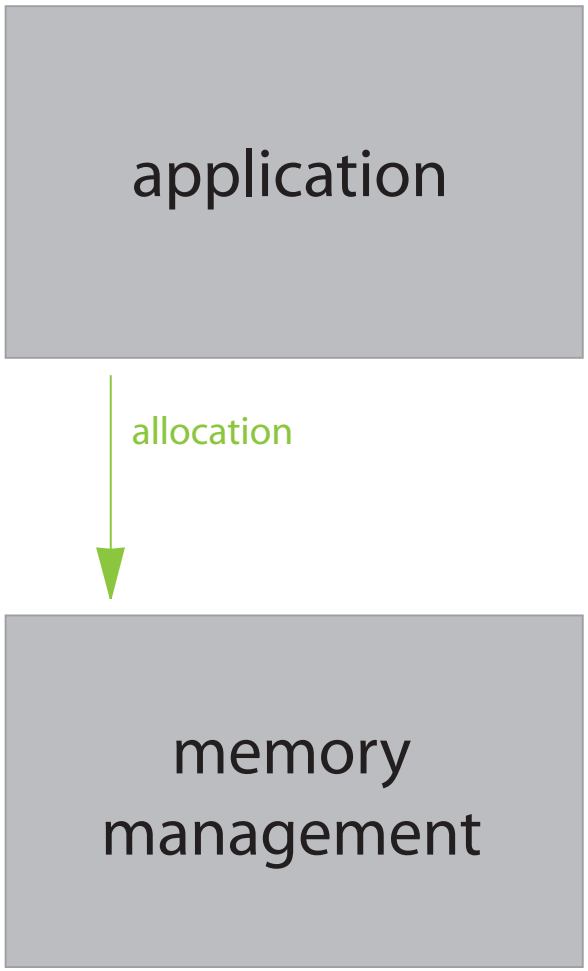
```javascript
// JS
function dosomething() {
  var s = "abc";                  // implicit allocation
  ...
}                                 // destroy references

dosomething();
```

MEMORY MANAGEMENT

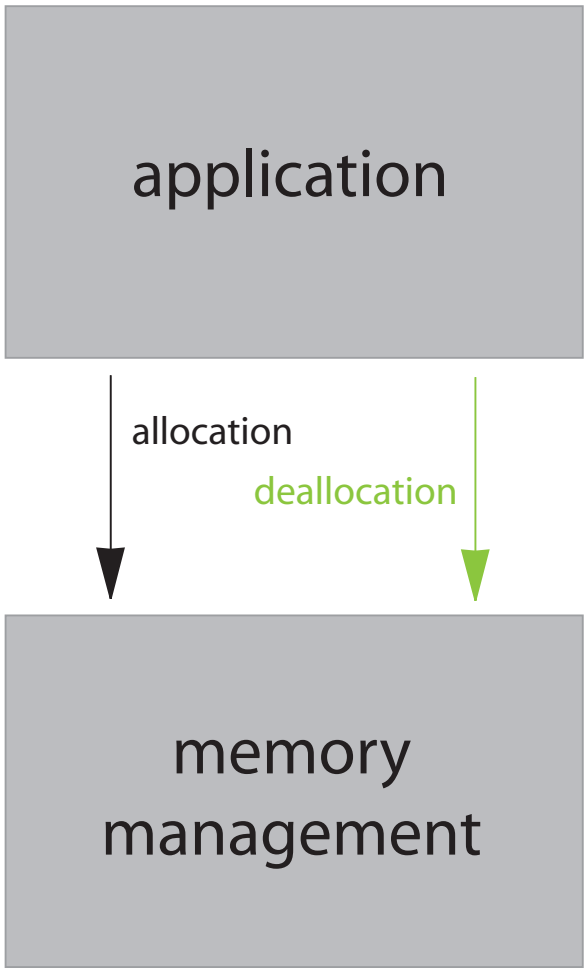
application

memory management

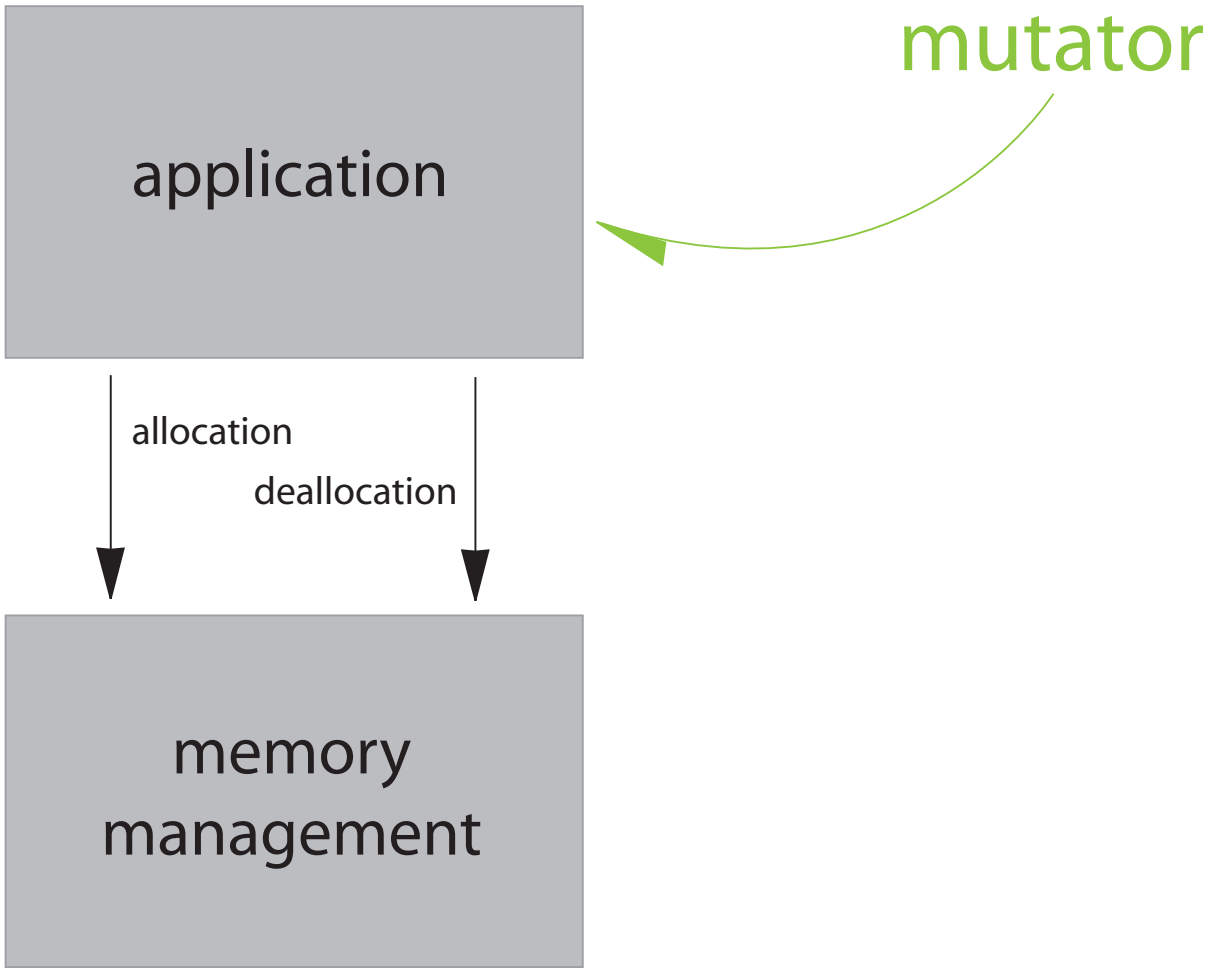
Memory management



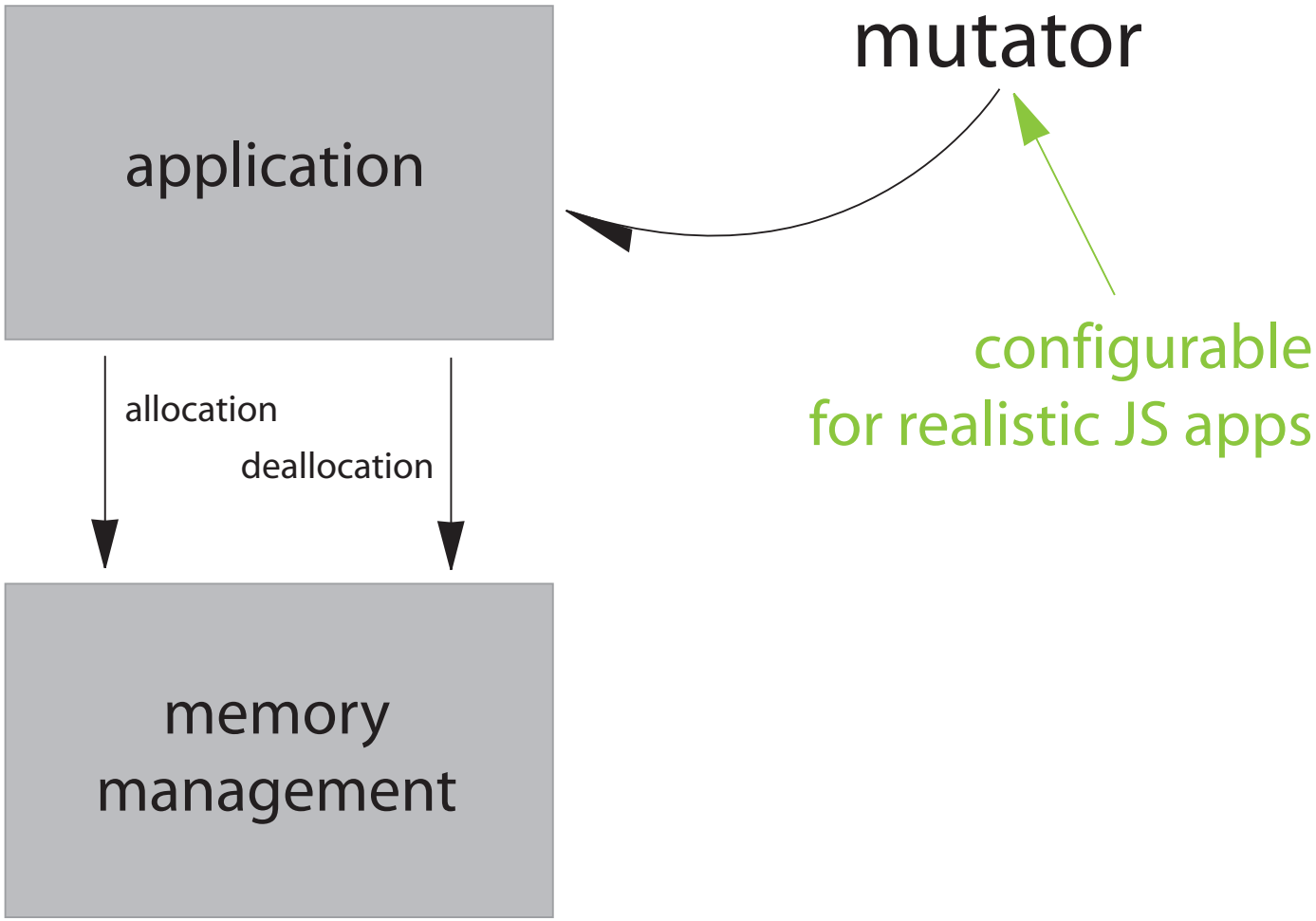
Memory management



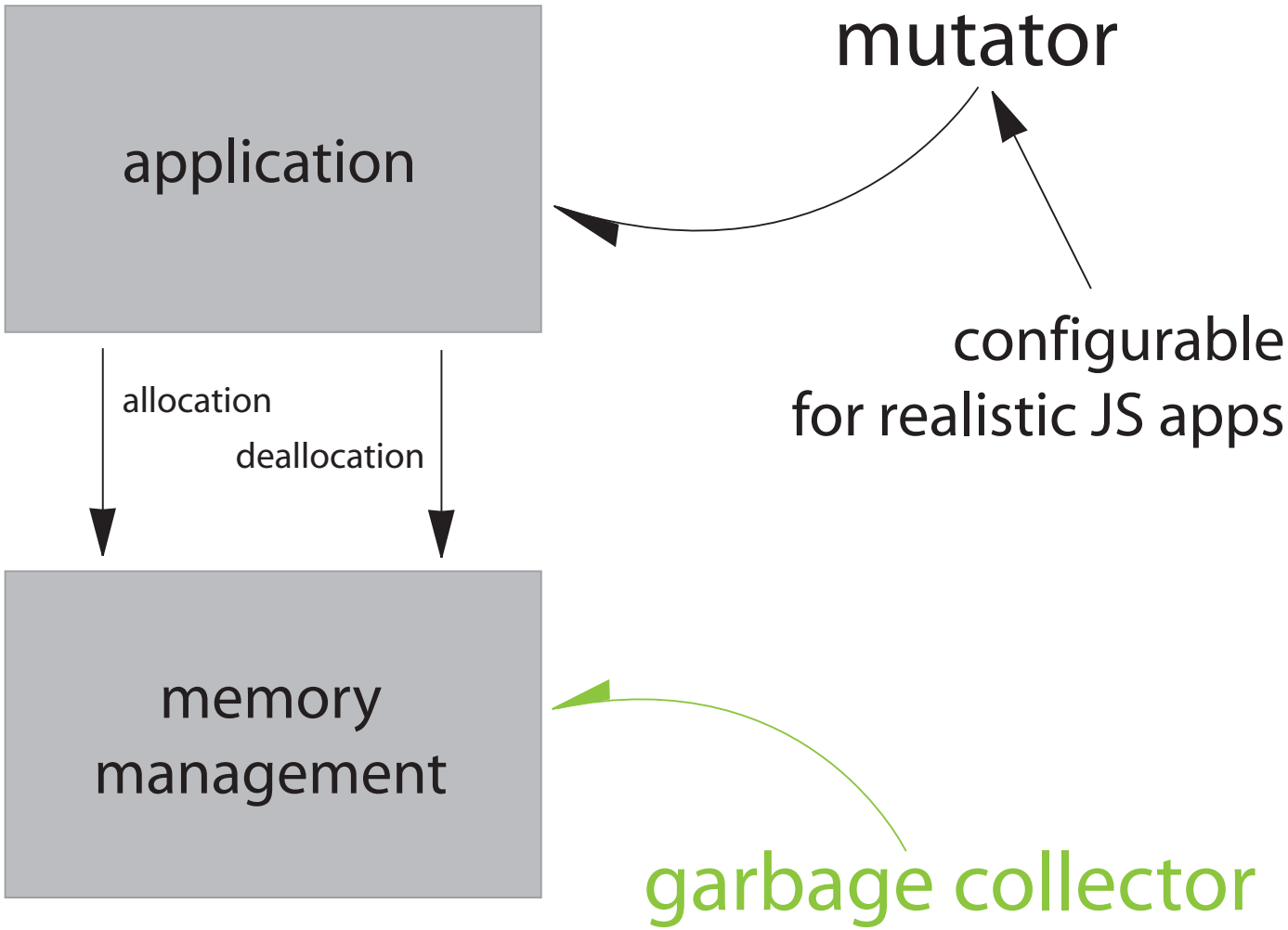
Memory management



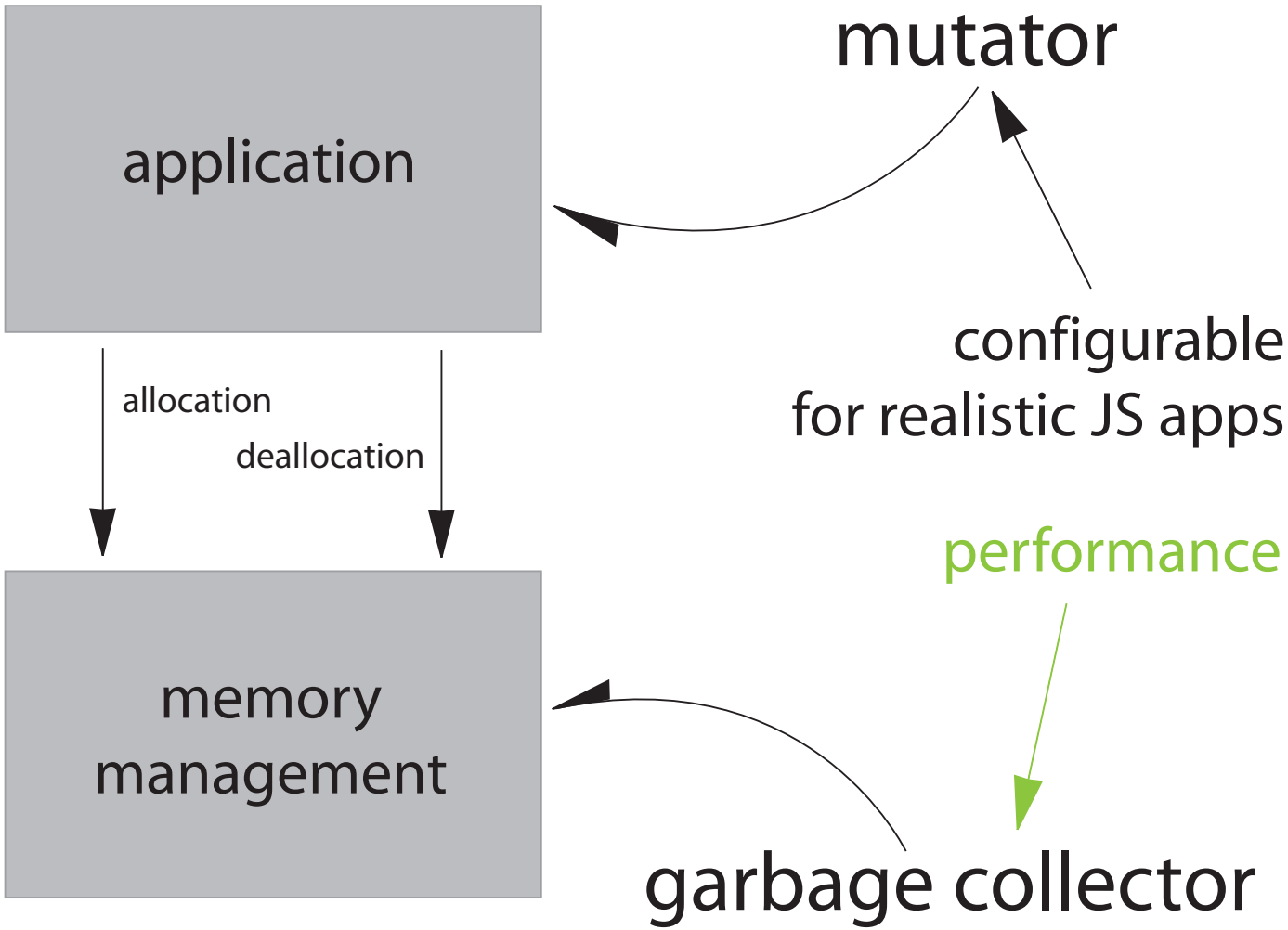
Memory management



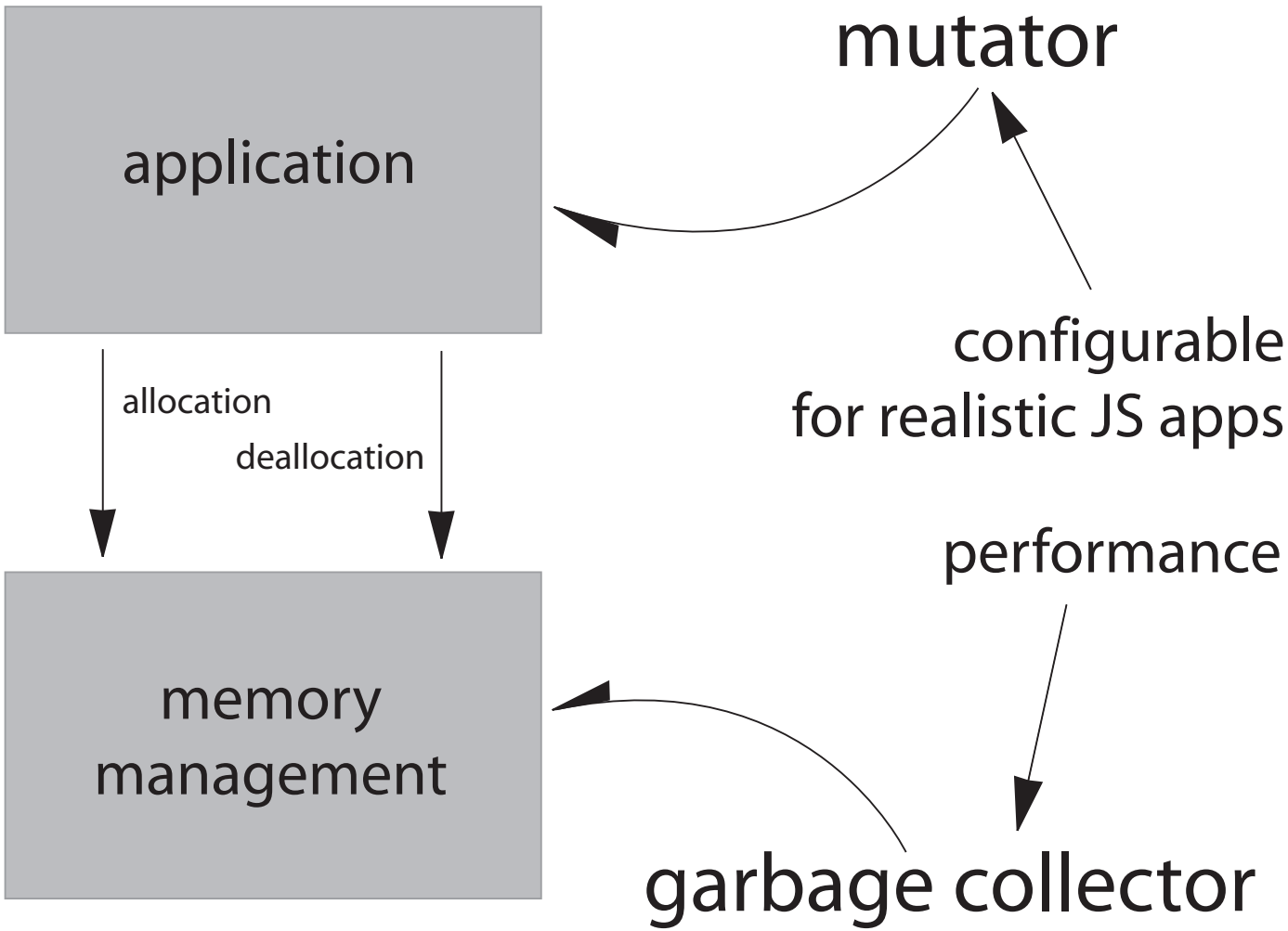
MEMORY MANAGEMENT



MEMORY MANAGEMENT



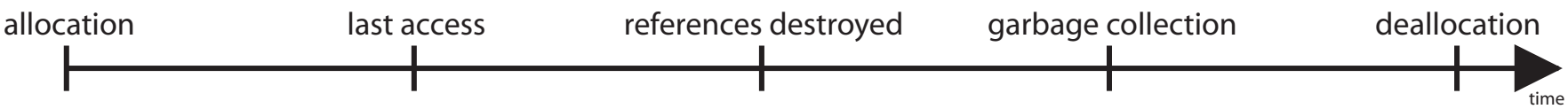
MEMORY MANAGEMENT



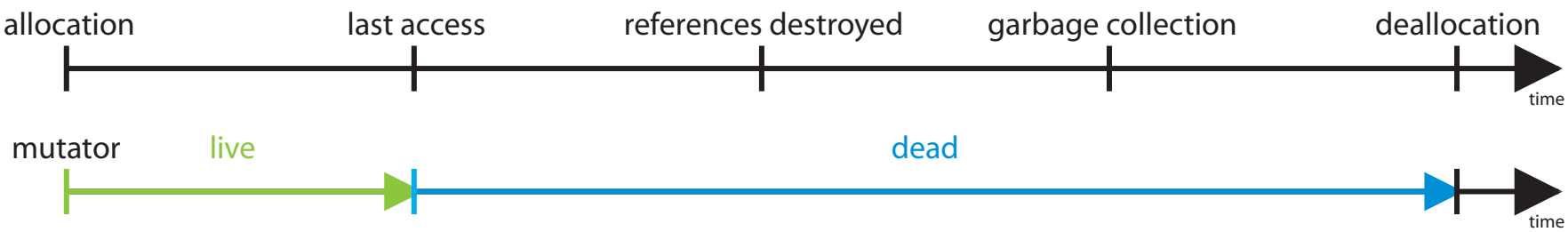
Garbage Collector

- differs between live memory and dead memory
- deallocates dead memory for reuse

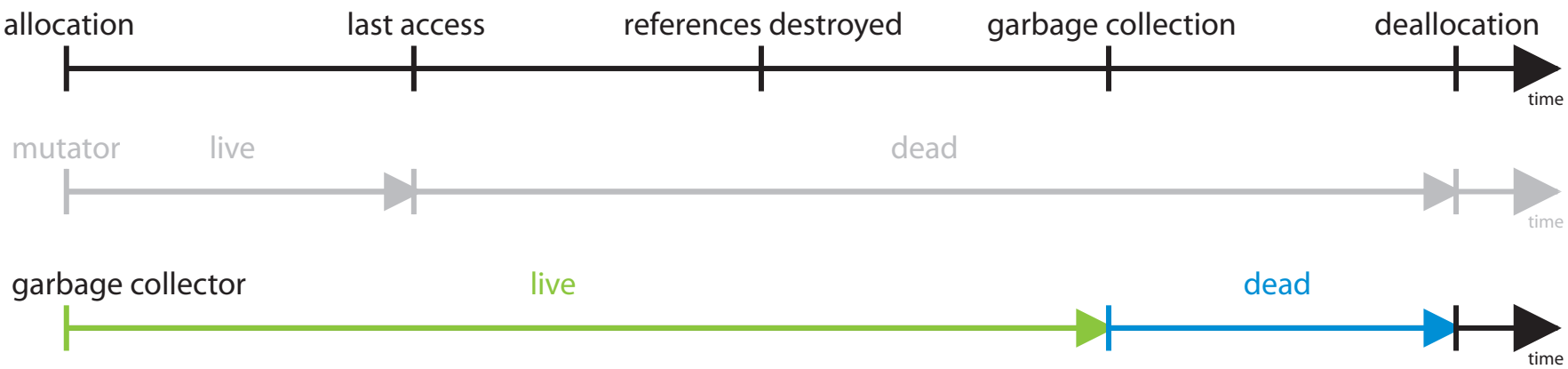
DIFFERENCE BETWEEN LIVE AND DEAD MEMORY



DIFFERENCE BETWEEN LIVE AND DEAD MEMORY

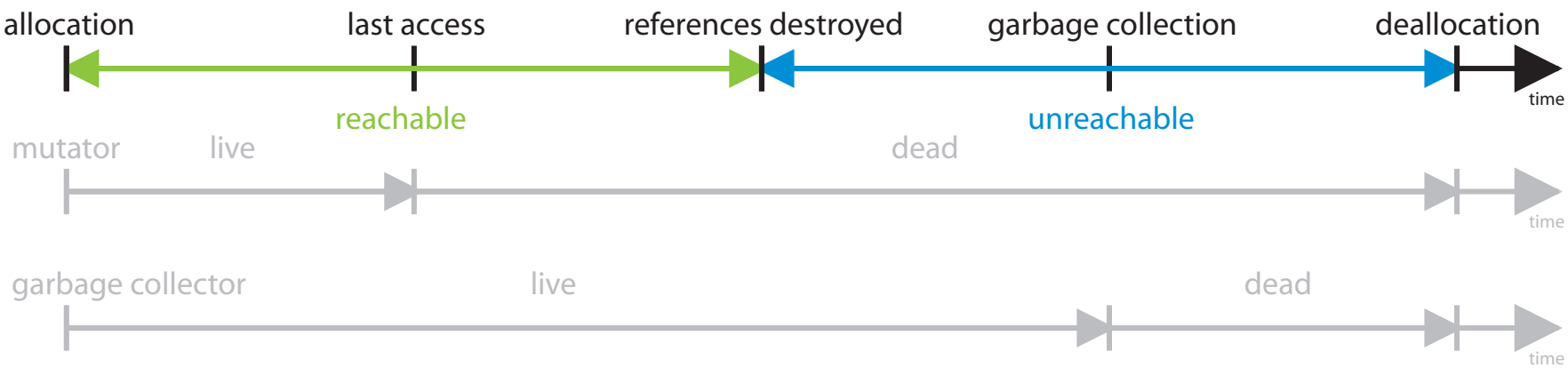


DIFFERENCE BETWEEN LIVE AND DEAD MEMORY

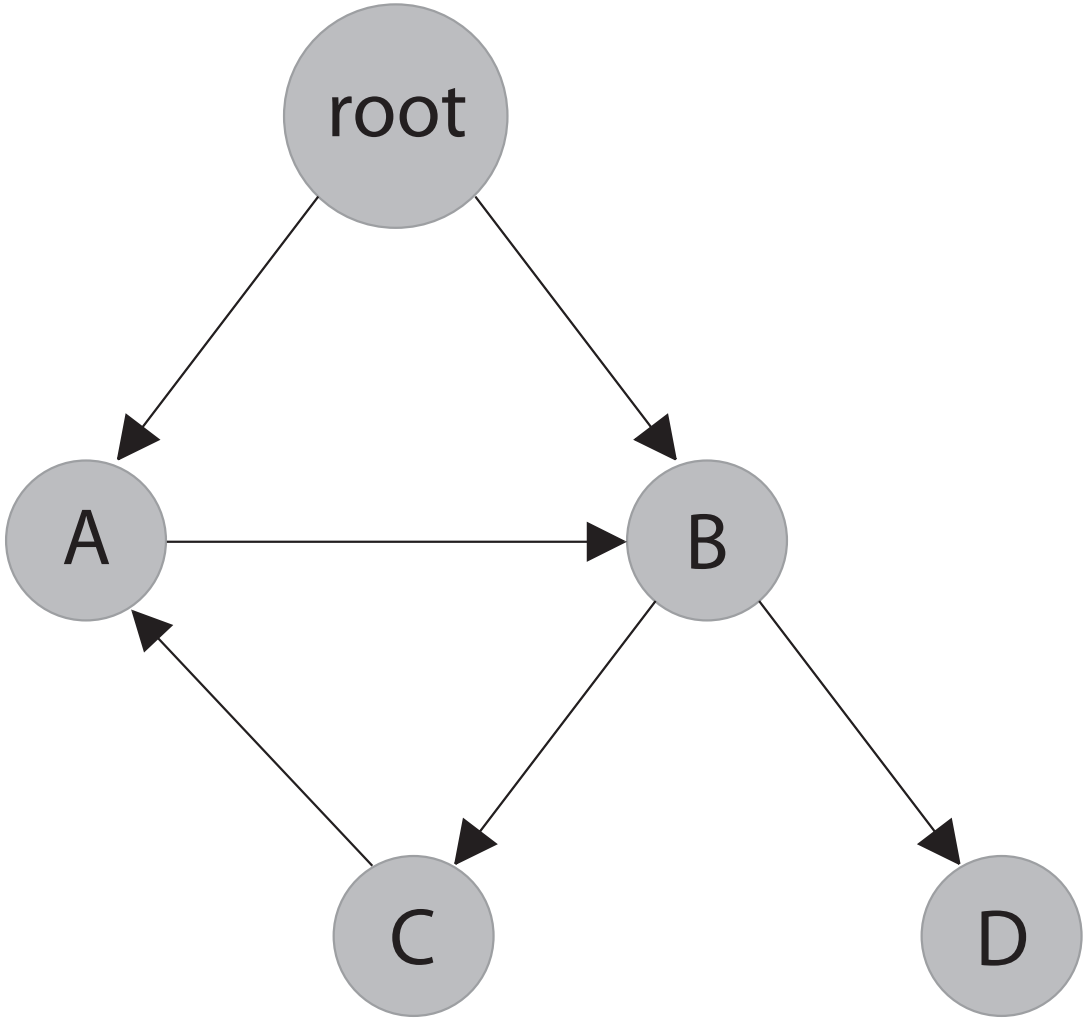


DIFFERENCE BETWEEN LIVE AND DEAD MEMORY

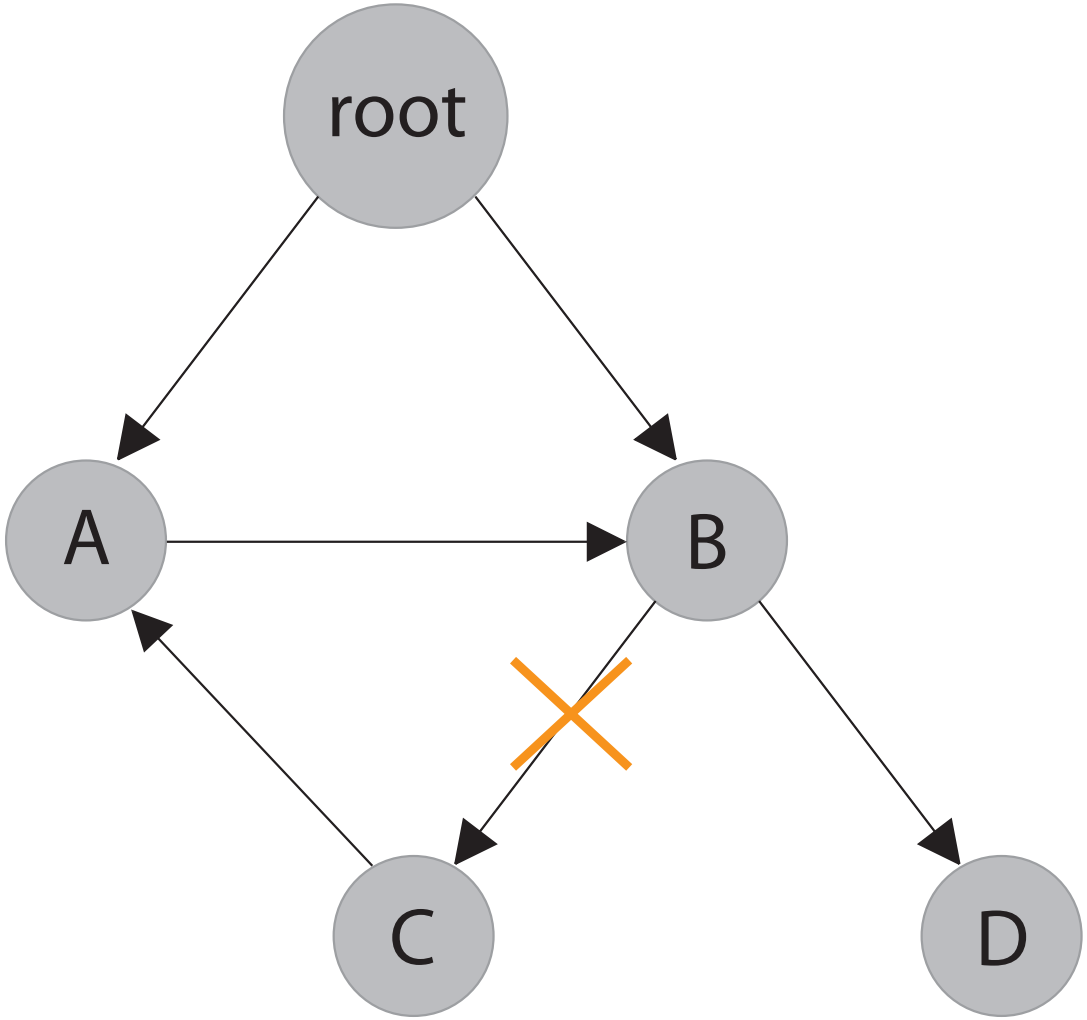
DIFFERENCE BETWEEN LIVE AND DEAD MEMORY



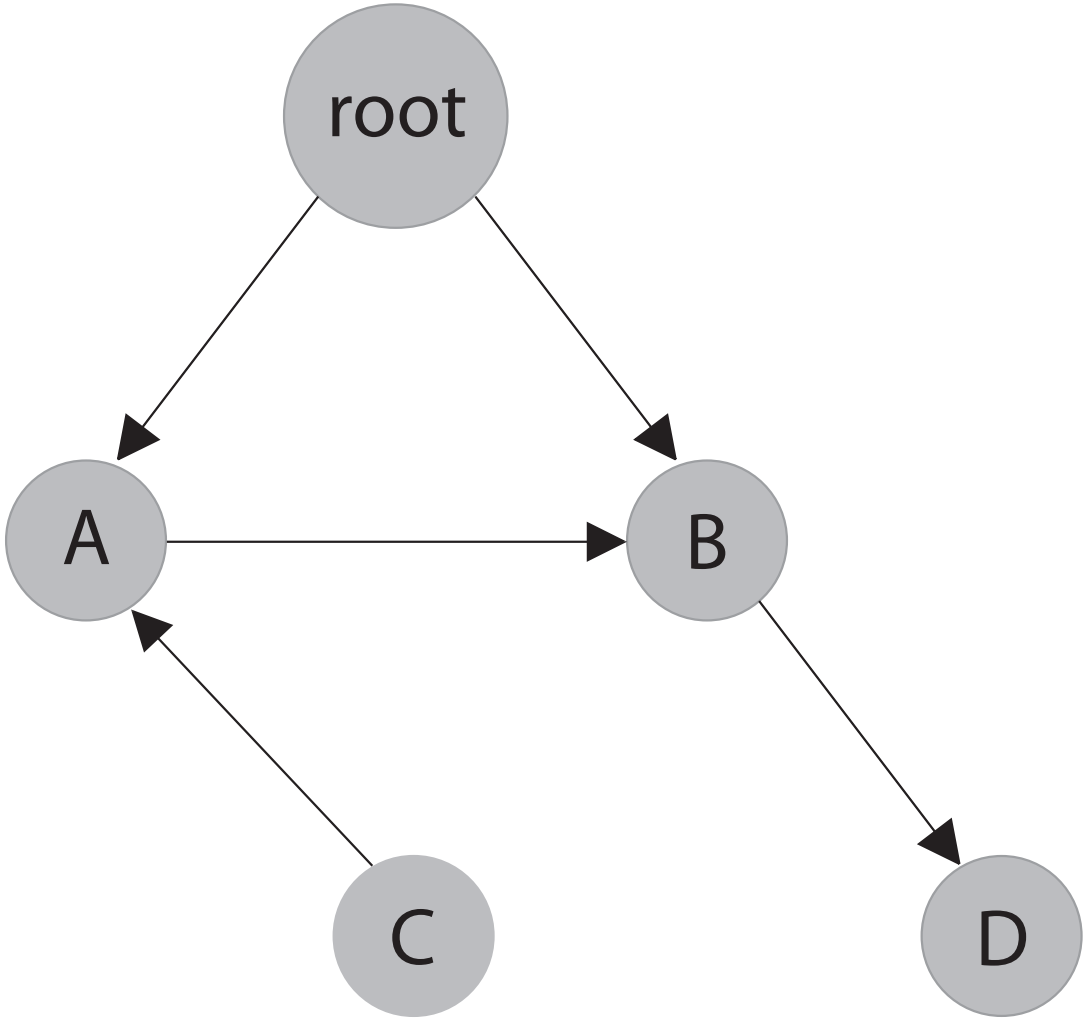
TRACING



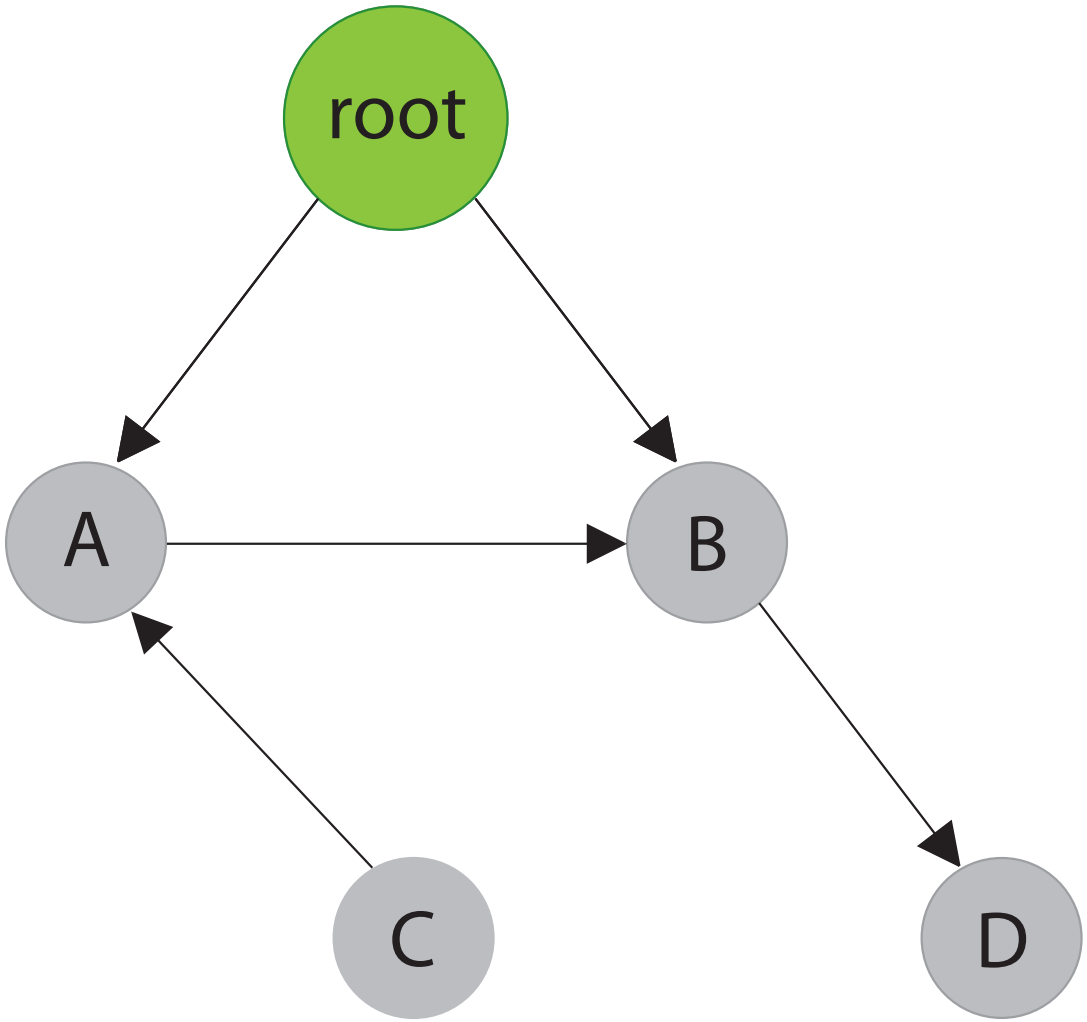
TRACING



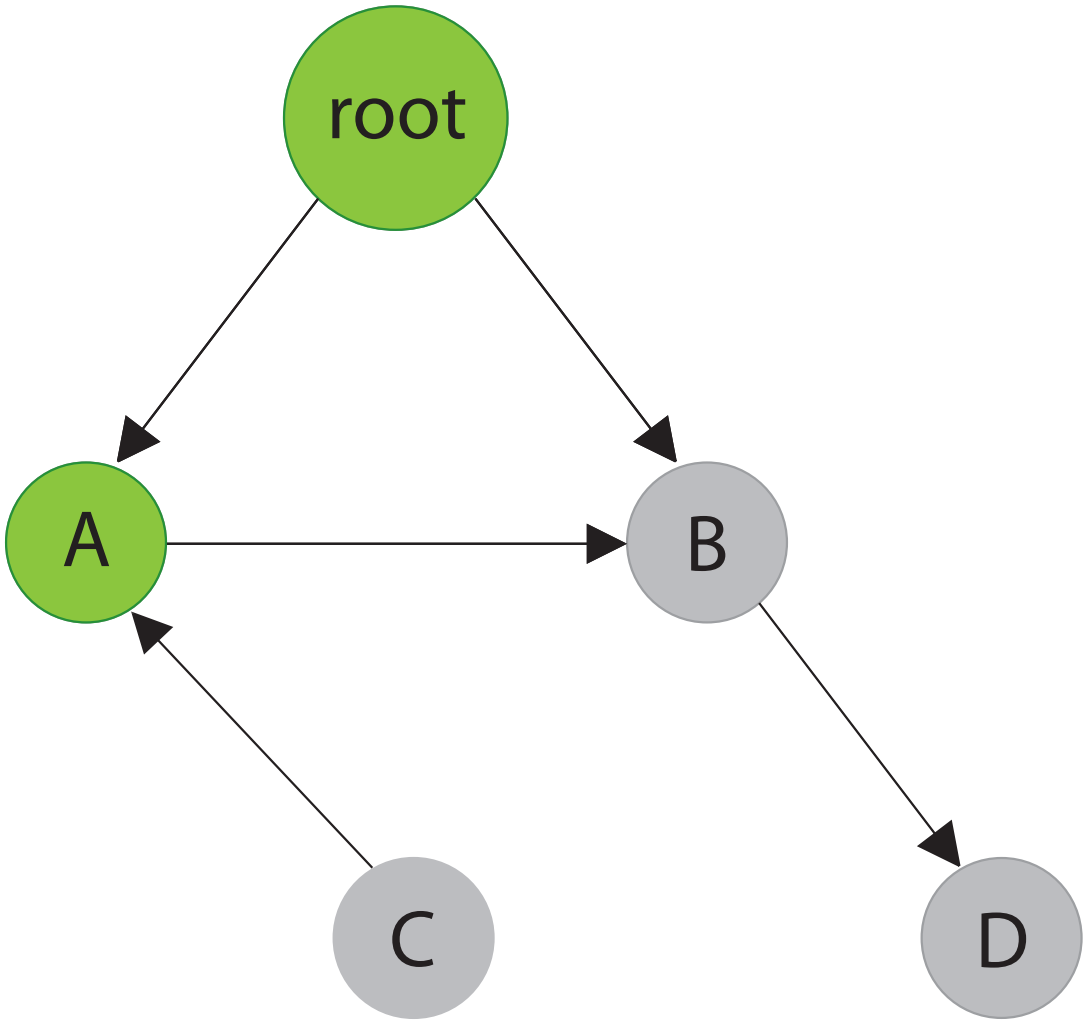
TRACING



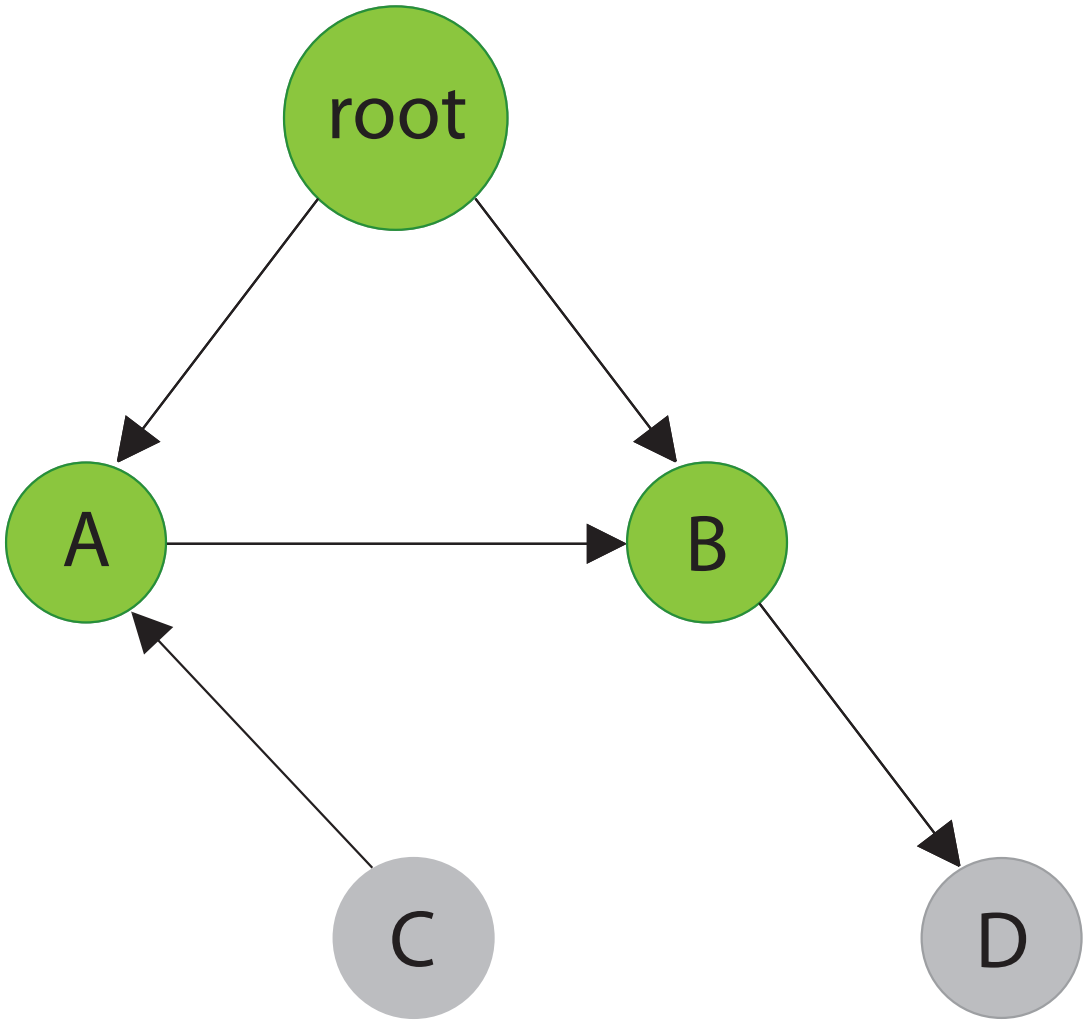
TRACING



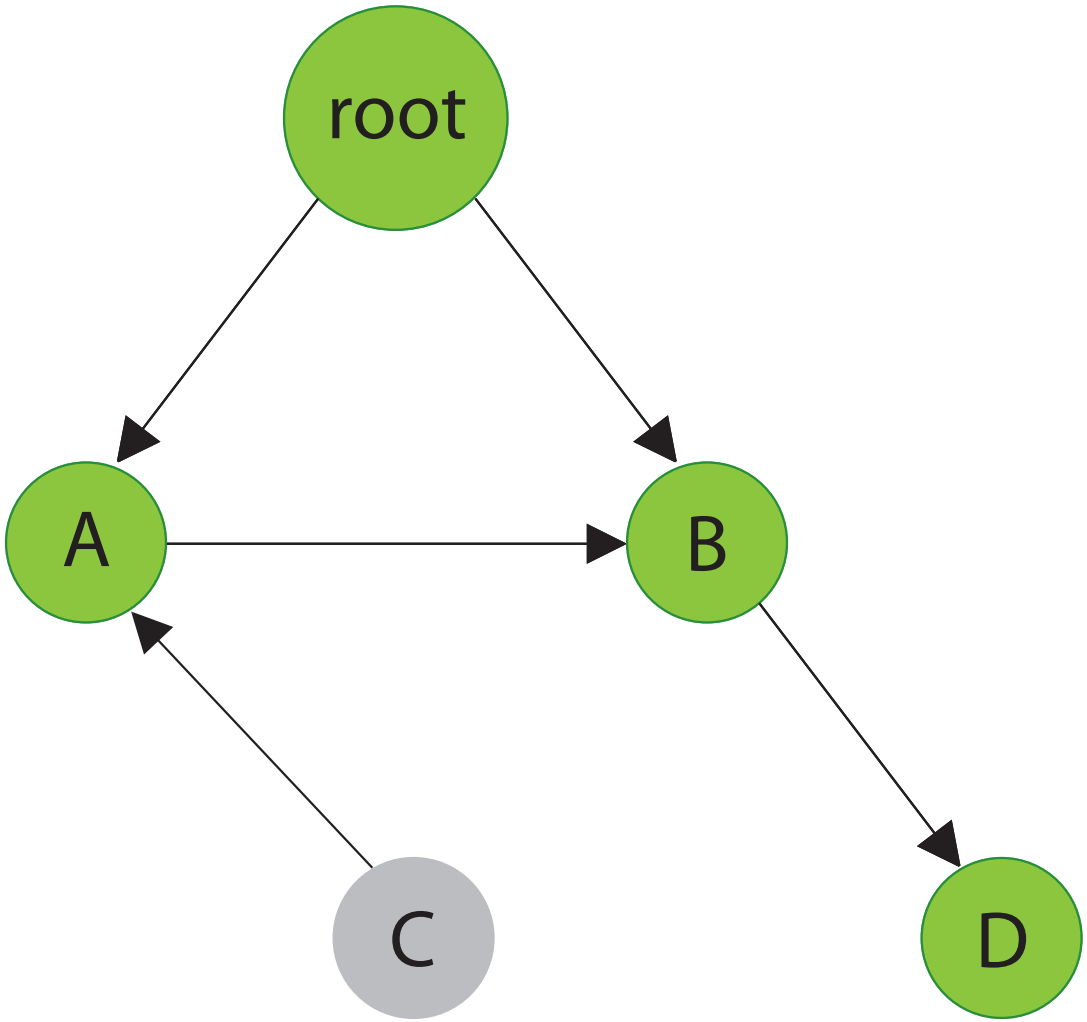
TRACING



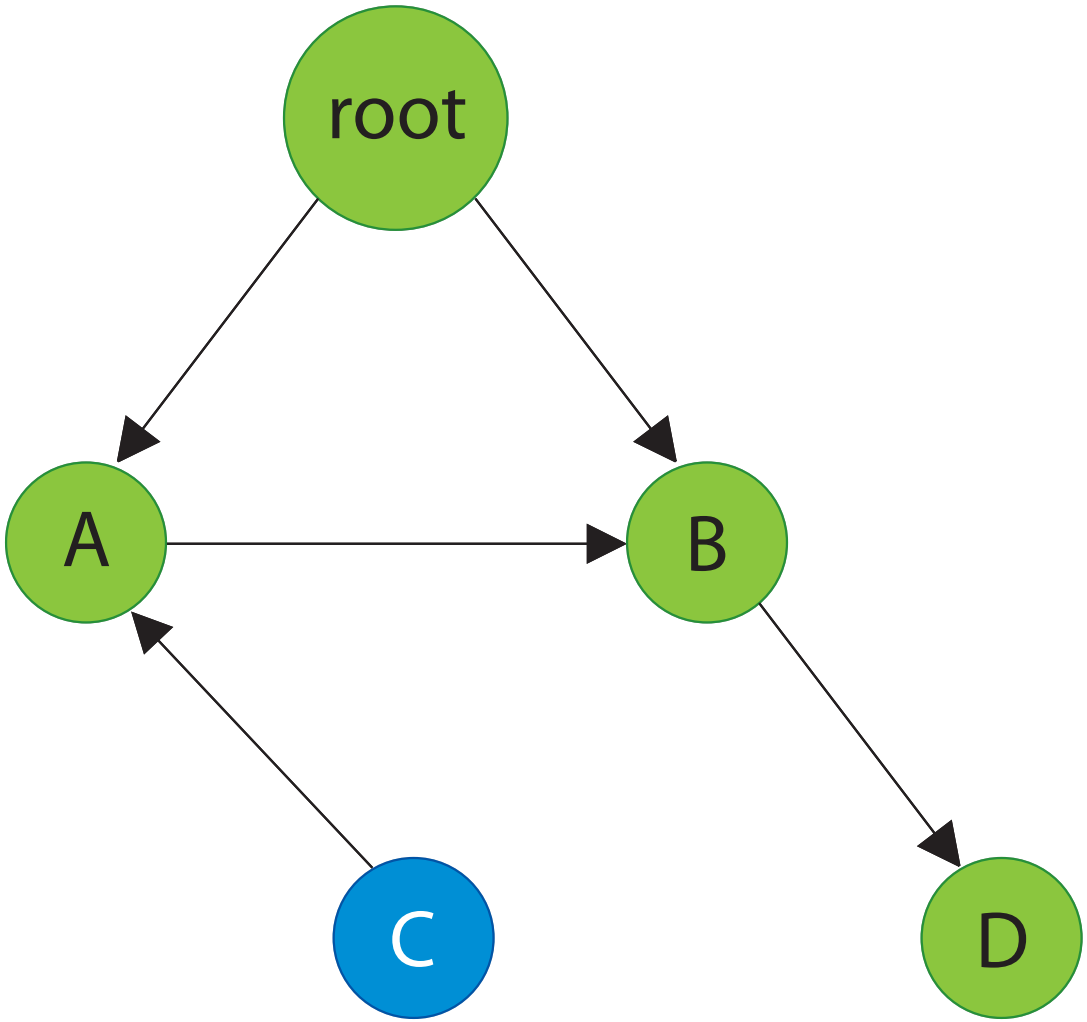
TRACING



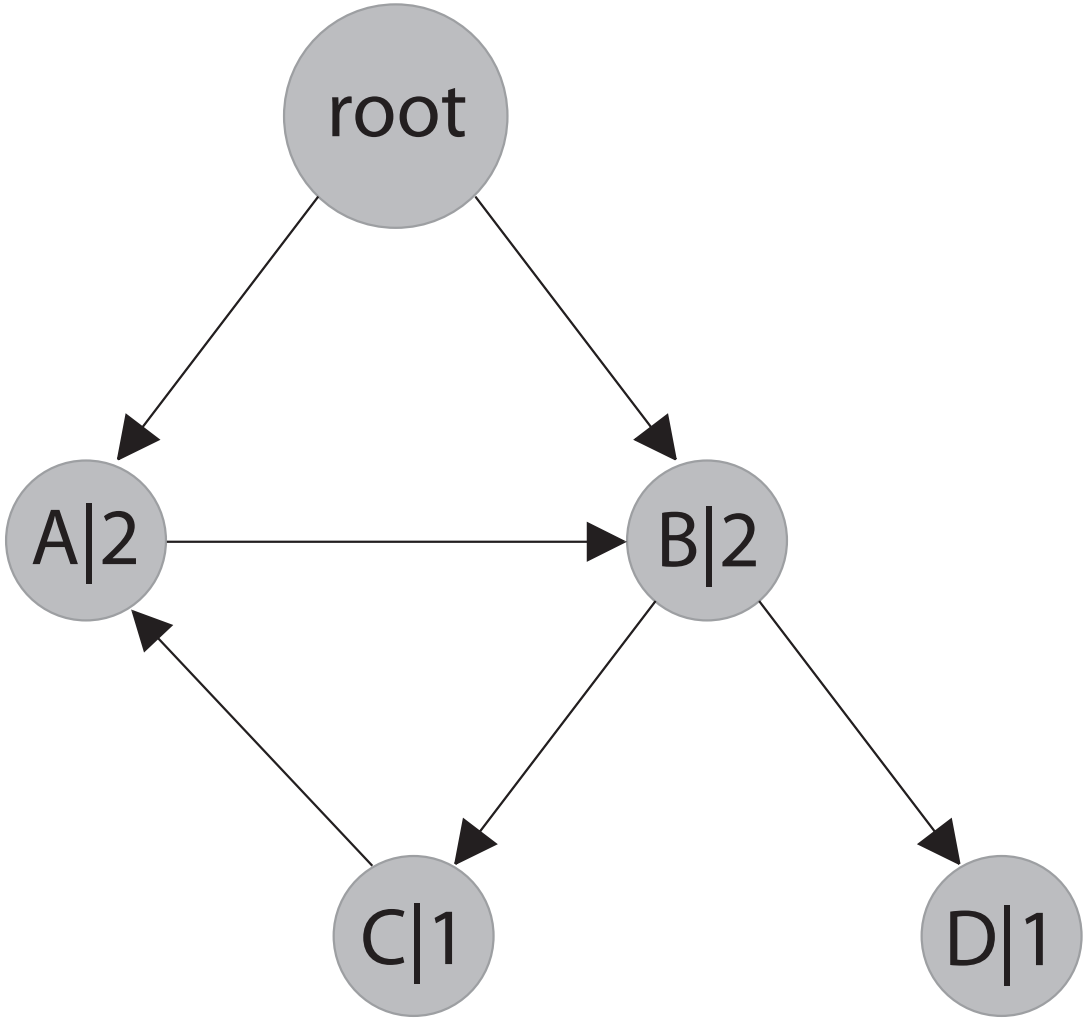
TRACING



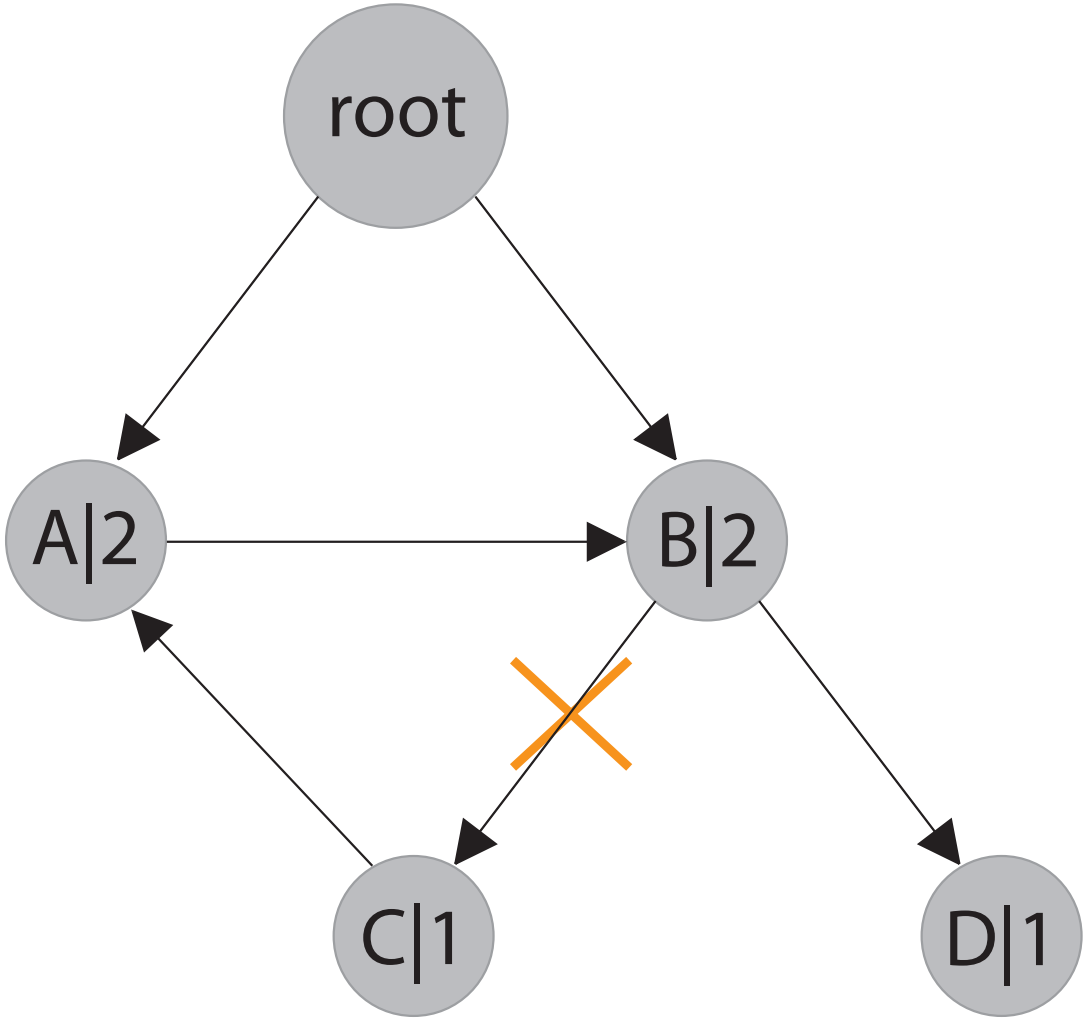
TRACING



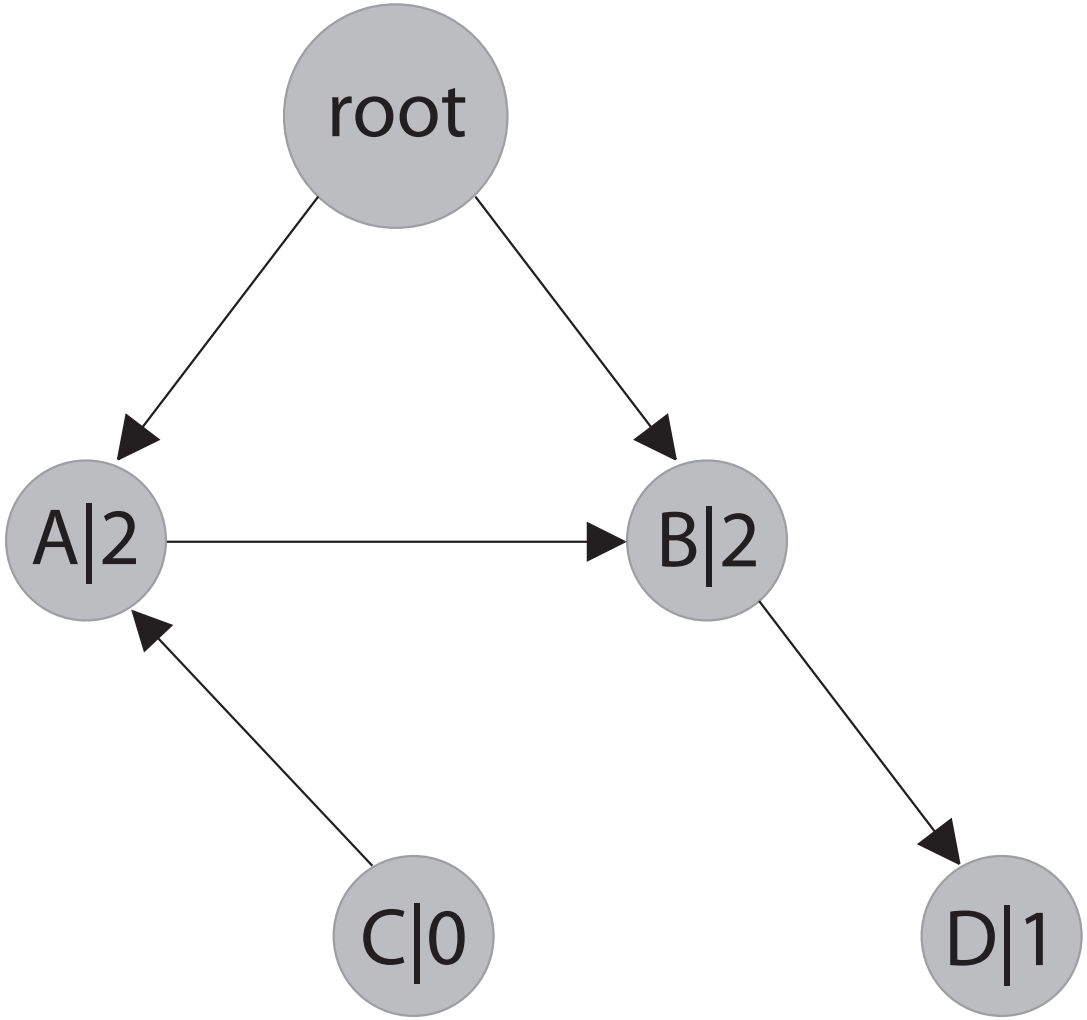
Reference counting



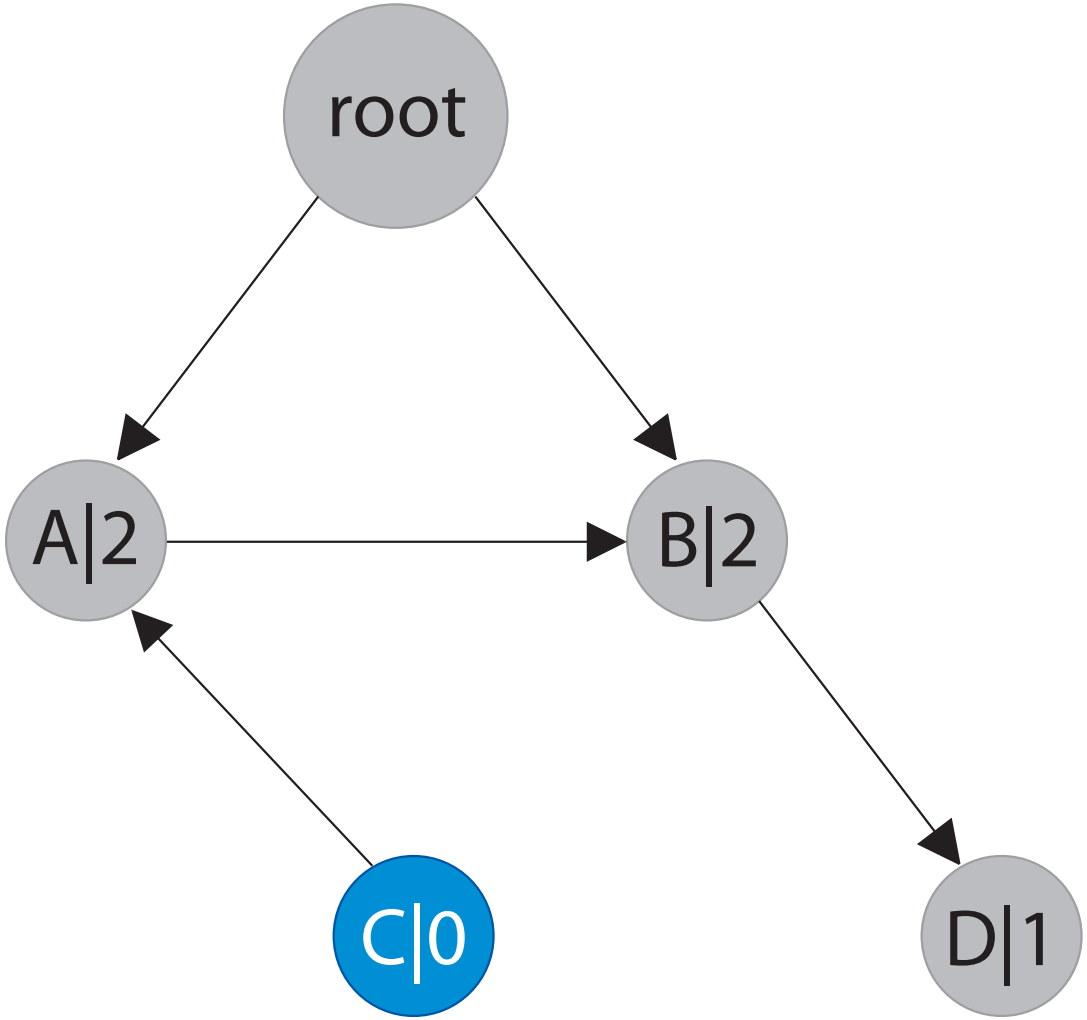
Reference counting



Reference counting



Reference counting



Problem definition

The purpose of ACDC4JS is to analyze the efficiency of the garbage collector in JavaScript virtual machines, especially Google's V8.

Experiments

- Step 1: Simple but artificial mutator
- Step 2: Obtaining a realistic heap model
- Step 3: Developing a realistic JavaScript mutator (TODO)

Step 1: Simple but artificial mutator

- Get information about the garbage collector
  - Collection frequency
  - Quantity of collected memory
  - Number of collected objects

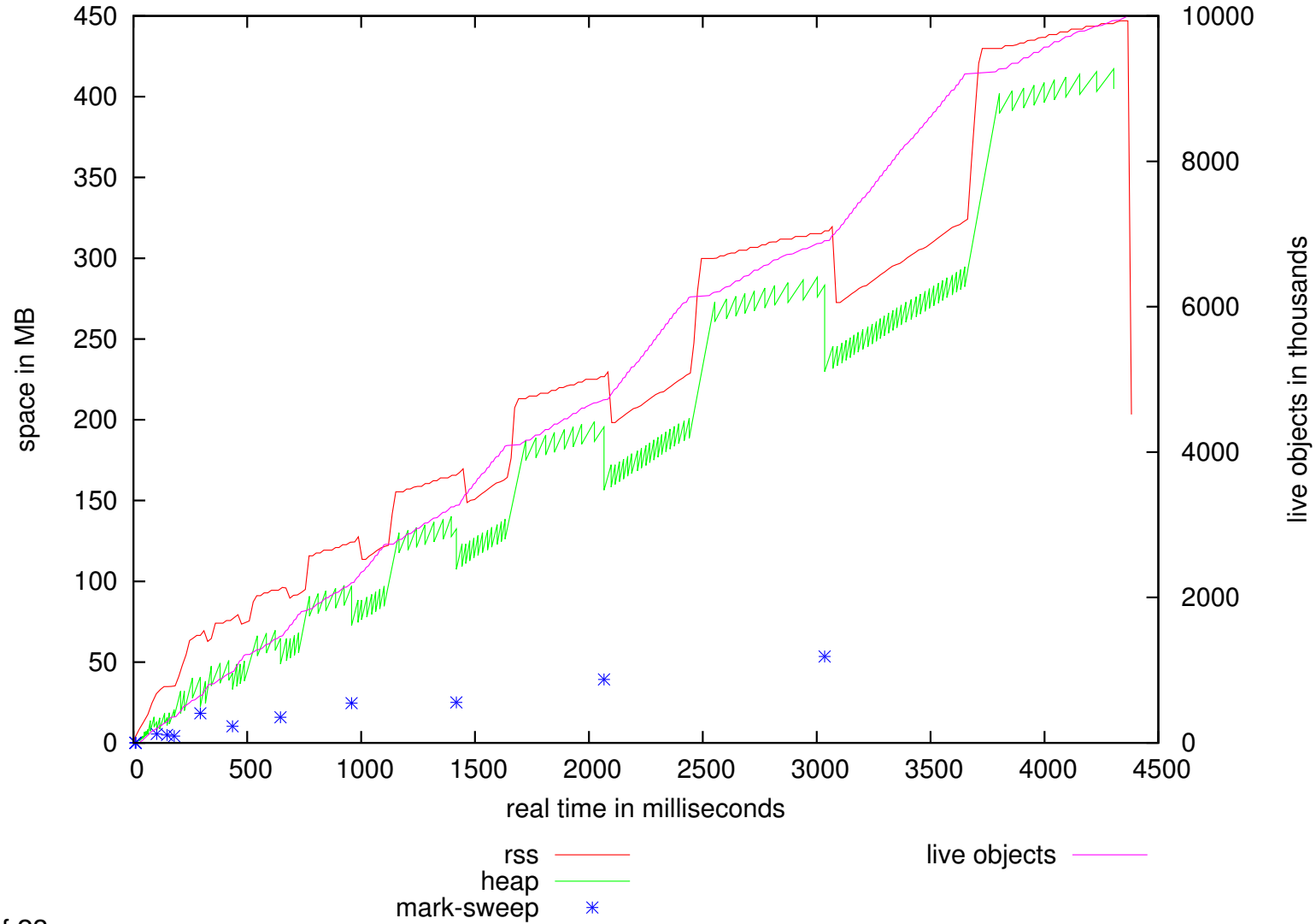
Step 1: Simple but artificial mutator

- Get information about the garbage collector
  - Collection frequency
  - Quantity of collected memory
  - Number of collected objects
- Prepare a simple mutator
  - Number of allocated objects
  - Number of live objects
  - Size of an object

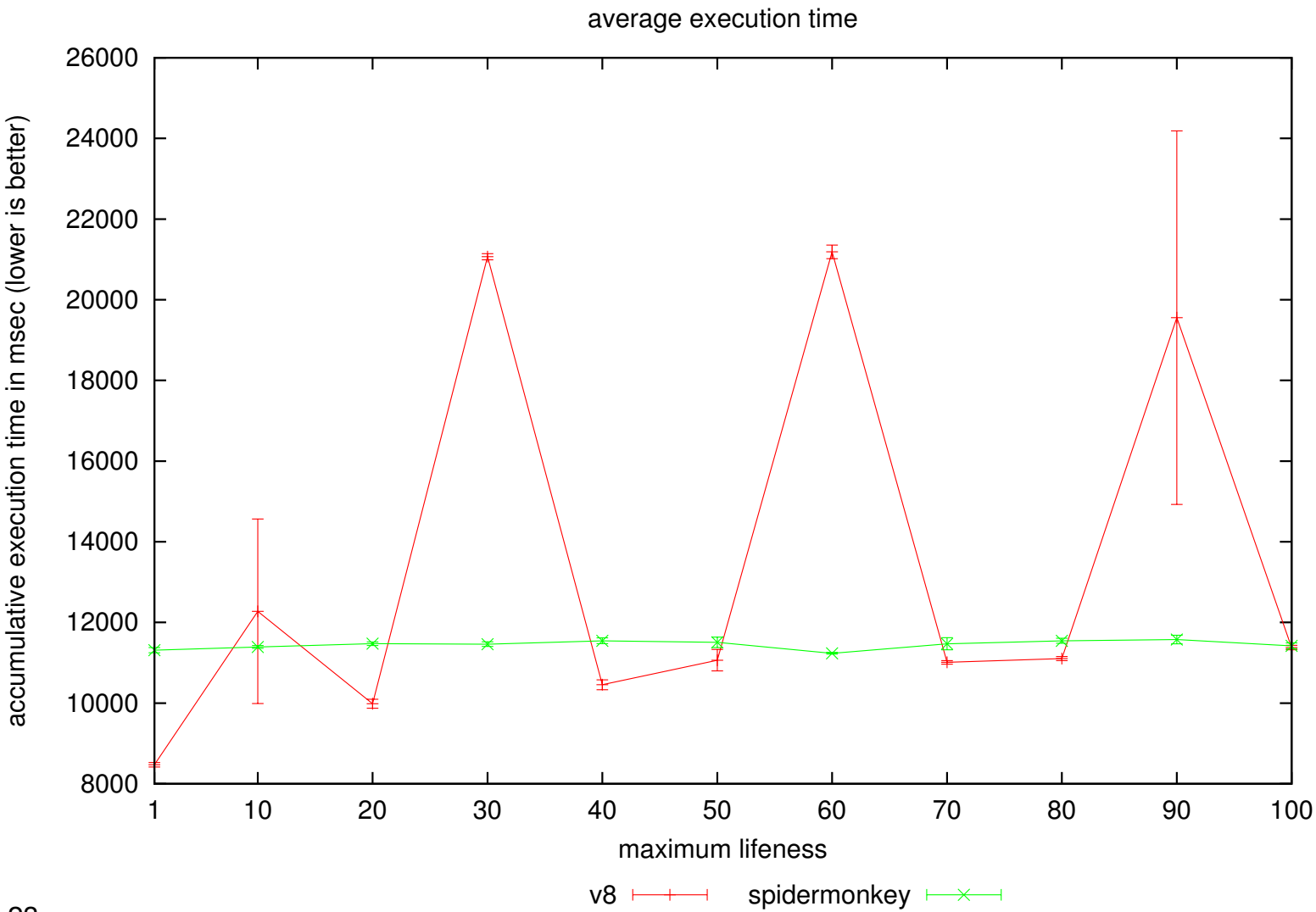
Step 1: Simple but artificial mutator

- Get information about the garbage collector
  - Collection frequency
  - Quantity of collected memory
  - Number of collected objects
- Prepare a simple mutator
  - Number of allocated objects
  - Number of live objects
  - Size of an object
- Wrap system measurements
  - Execution time
  - Real memory (resident set size ... rss)

STEP 1: MEASUREMENTS - ALLOCATION ONLY



STEP 1: MEASUREMENTS - EXECUTION TIME



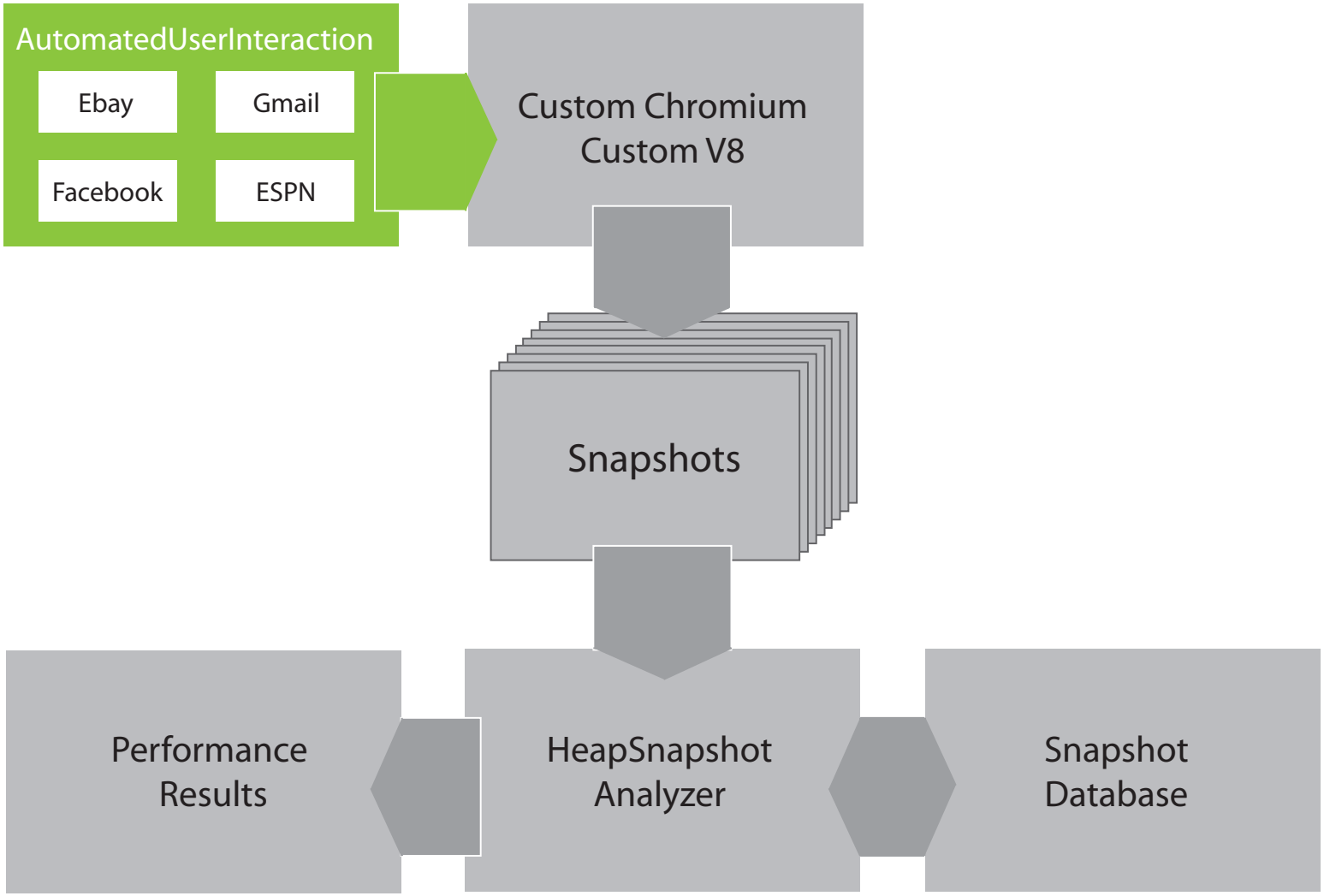
Step 2: Obtaining a realistic heap model

- Important for mutator implementation
- Customizations
  - Custom Chromium binary
  - Custom V8 binary

Step 2: Obtaining a realistic heap model

- Important for mutator implementation
- Customizations
  - Custom Chromium binary
  - Custom V8 binary
- Tools for JavaScript heap analysis
  - AutomatedUserInteraction
  - HeapSnapshotAnalyzer

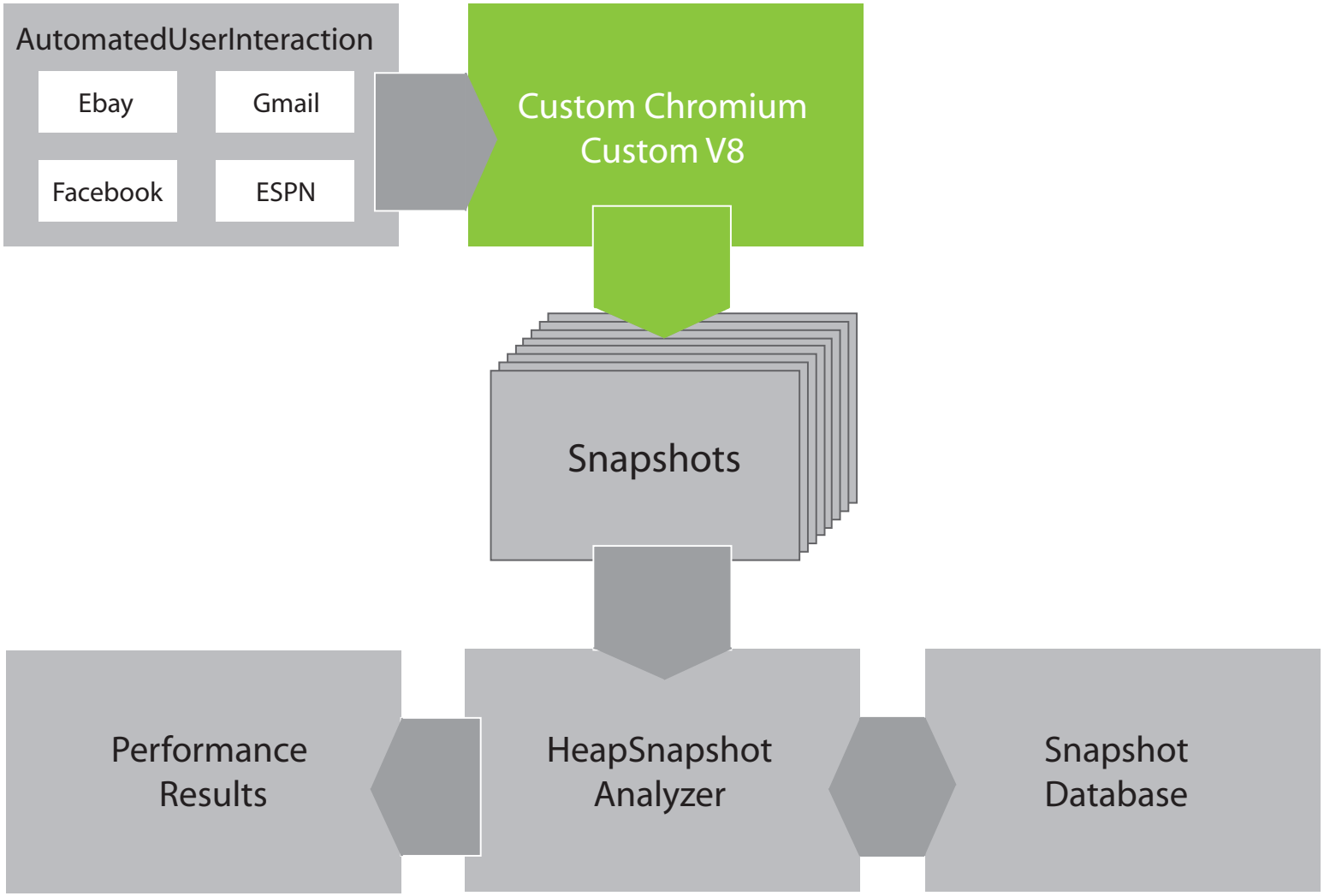
Step 2: Obtaining a realistic heap model



AUTOMATED USER INTERACTION

- Java application
- SeleniumHQ framework for automated user interaction
  - http://www.seleniumhq.org
- Web applications
  - **News:** CNN, ESPN, The Economist
  - **Email:** Gmail, Hotmail
  - **Shops:** Ebay, Amazon
  - **Maps:** Google, Bing
  - **Search:** Google, Bing
  - **Social:** Facebook, Google Plus

Step 2: Obtaining a realistic heap model

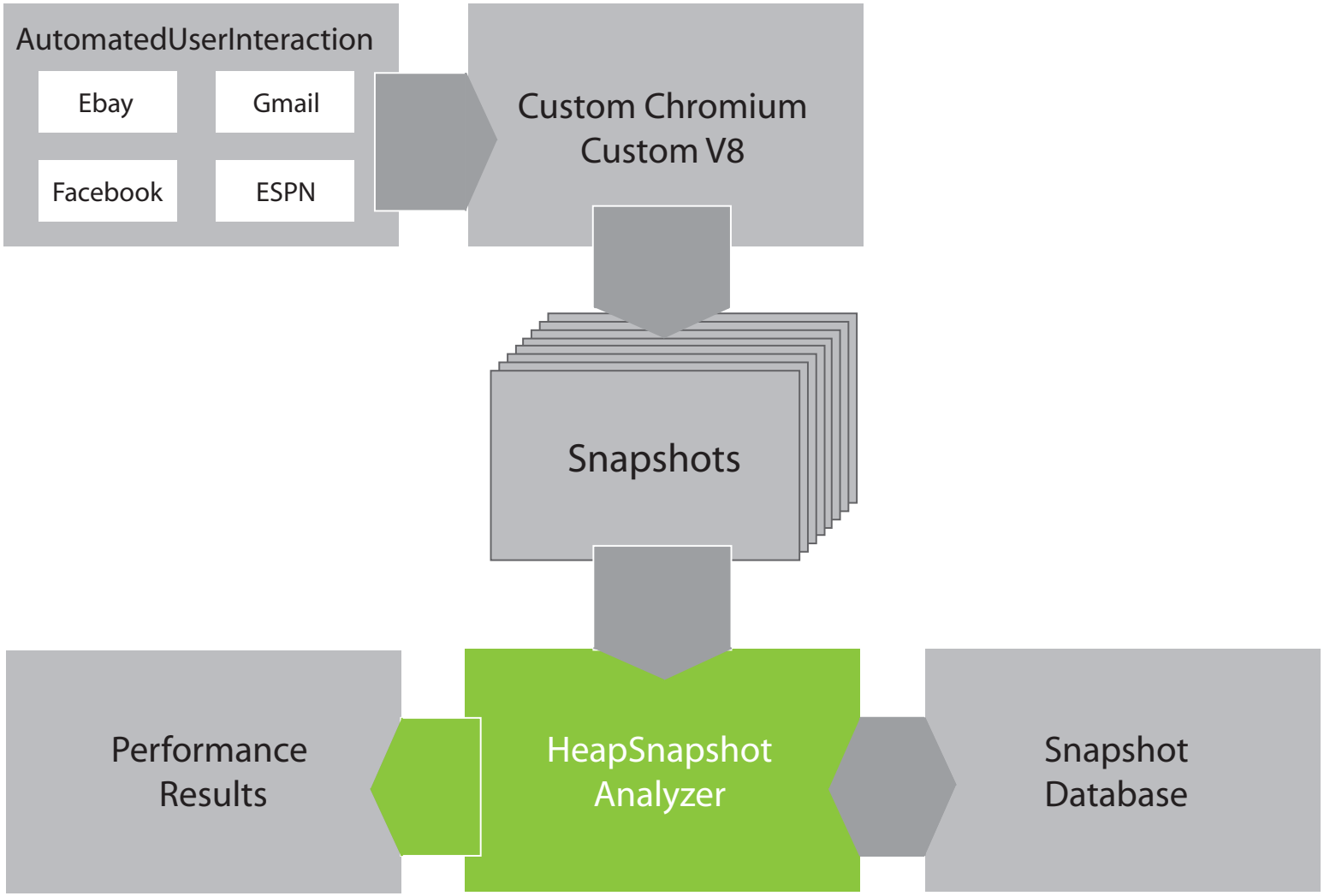


CUSTOMIZATIONS

Custom V8 binary

- New flags
  - `automatic_heap_snapshots`
  - `heap_snapshot_interval`
  - `heap_snapshot_prefix`
- Used flags
  - `gc_interval`

Step 2: Obtaining a realistic heap model



HeapSnapshotAnalyzer

- Java application
- PostgreSQL 9.3
- Write snapshots into database

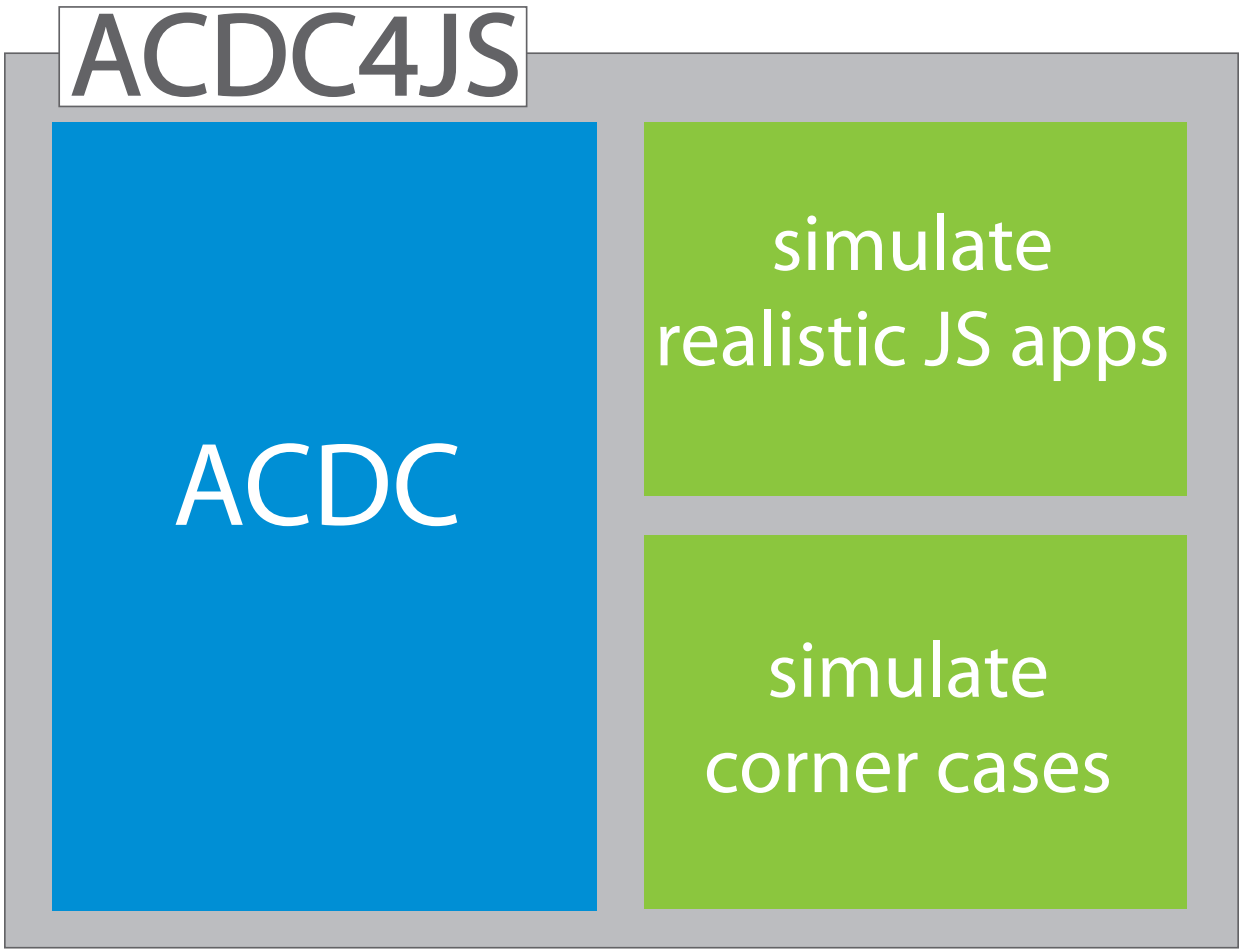
HeapSnapshotAnalyzer

- Java application
- PostgreSQL 9.3
- Write snapshots into database
- Heap graph
  - Number of leafs
  - Number of nodes
  - Number of edges
  - Number of strongly connected components

HeapSnapshotAnalyzer

- Java application
- PostgreSQL 9.3
- Write snapshots into database
- Heap graph
  - Number of leafs
  - Number of nodes
  - Number of edges
  - Number of strongly connected components
- Node characteristics
  - In-degree
  - Out-degree
  - Root distance
  - Node size

Step 3: Developing a realistic JavaScript mutator (TODO)



Thank you for your attention!

Questions?