Relational Debugging — Pinpointing Root Causes of Performance Problems

Xiang (Jenny) Ren, Sitao Wang, Zhuqi Jin,
David Lion, Adrian Chiu, Tianyin Xu, Ding Yuan

UNIVERSITY OF TORONTO

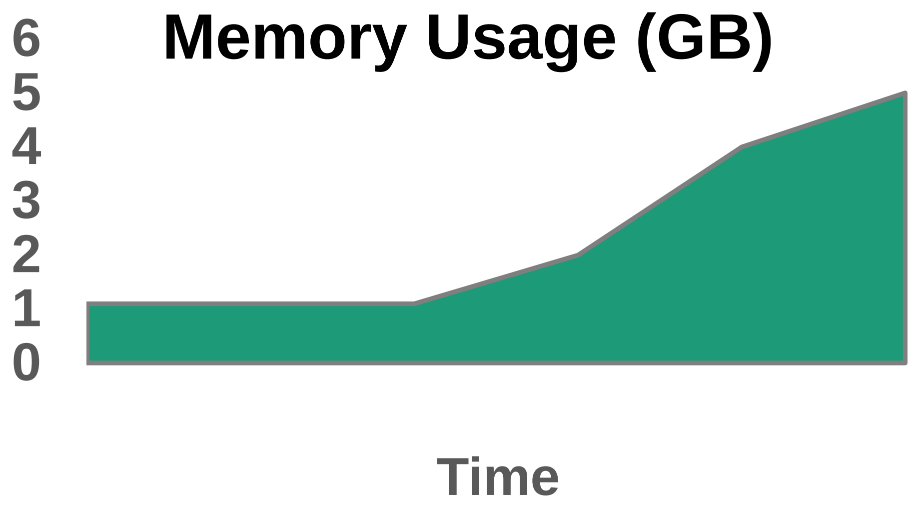
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Performance issues are costly

“Google found a 0.5 seconds delay (in page load time) caused a 20% decrease in repeat traffic”

“the Go process has been crashing every other hour … it was such a memory hog”

*Performance is **relative***



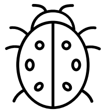
Request rate changed?

- More requests/period

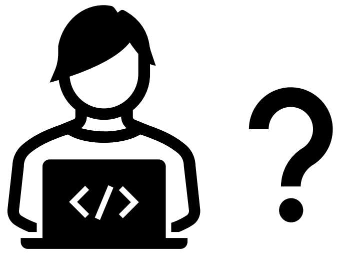
Request type changed?

- More allocation/request

Memory leak?

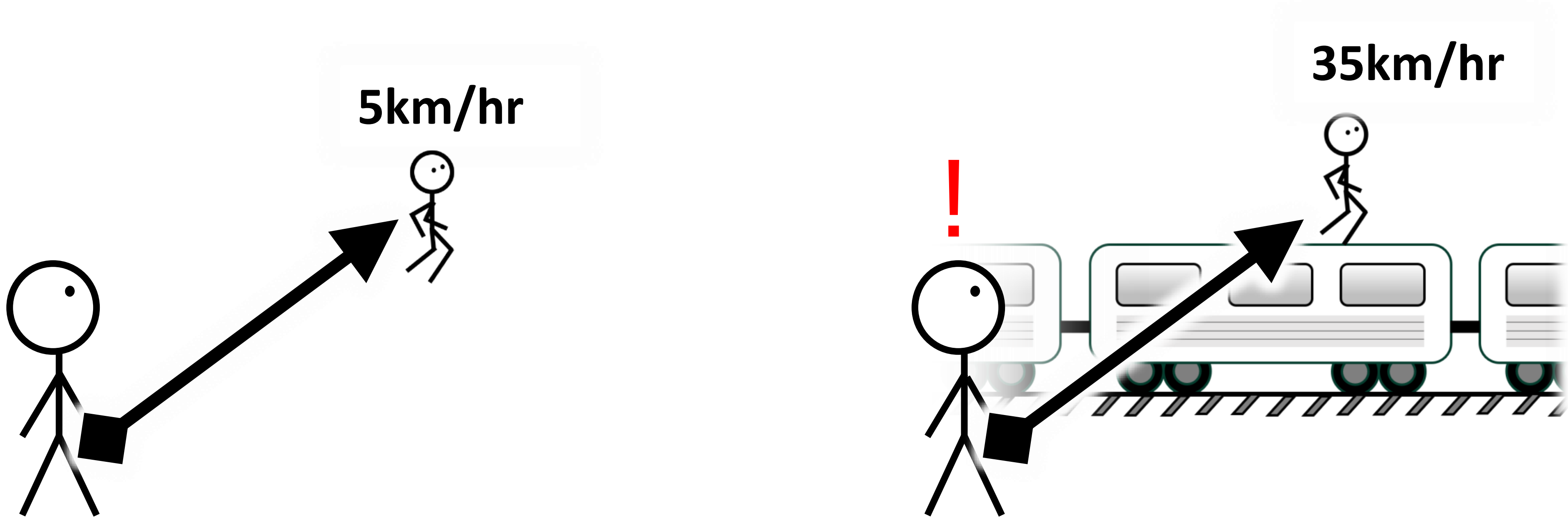


- Fewer deallocation/allocation



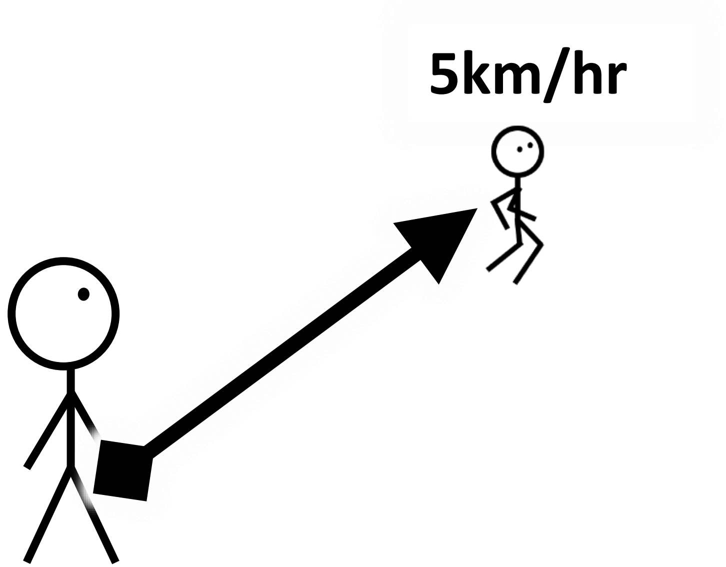
*Performance is **relative***

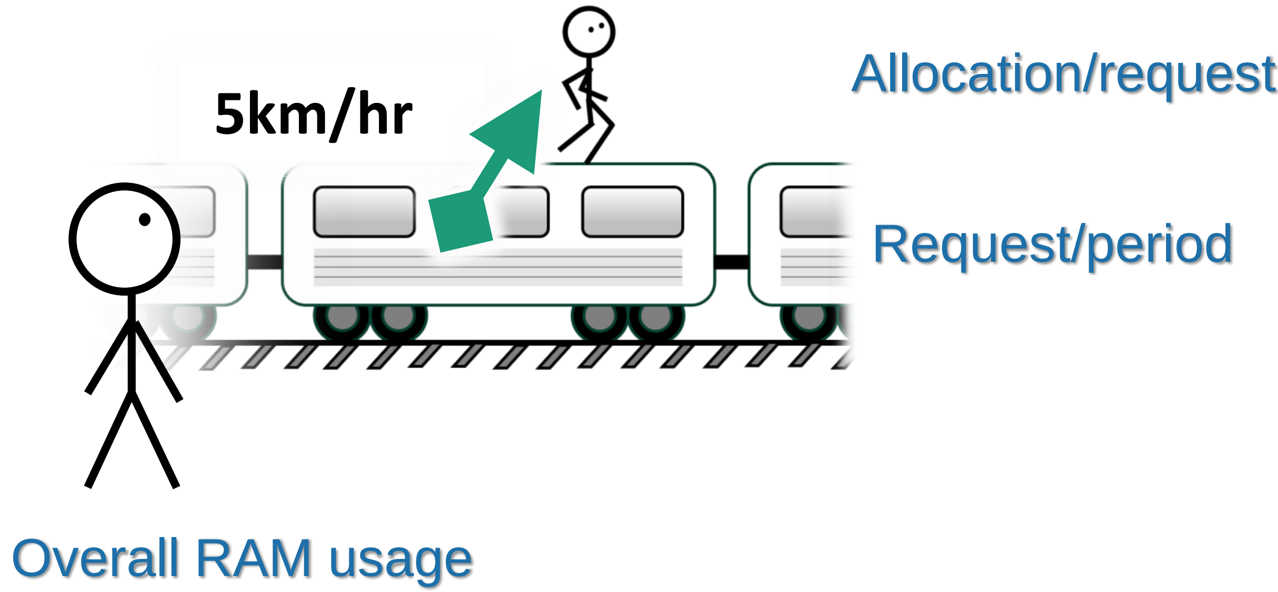
Idea: locate most specific reference point to captures the root cause



*Performance is **relative***

Idea: locate most specific reference point to captures the root cause

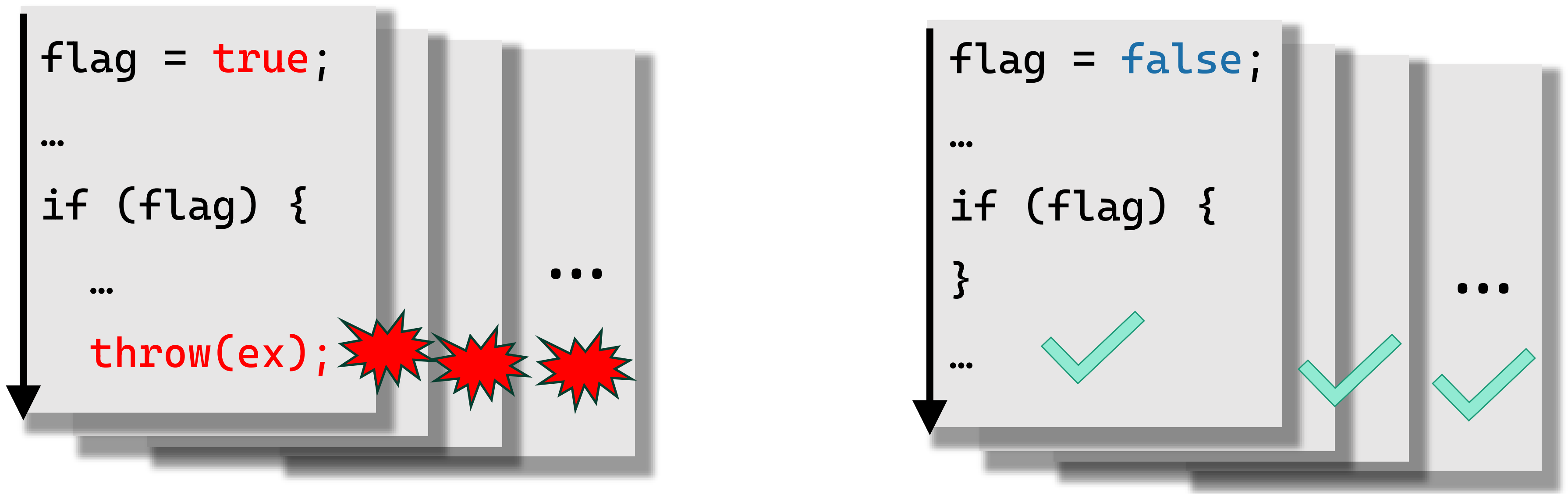




Existing solutions are limited

Statistical debugging

- Identifies ***absolute*** predicates correlated with failure
- Requires labeling ***many*** executions as fail or success



Failure: `throw(ex)`   Predicate: `flag == true`   Correlation: 100%

Relational Debugging
– pinpoints root causes of performance problems

Relations between events represents relative performance

& general representation of performance root causes.

```
Request#1              Request#2
malloc()  ┐            malloc()  ┐
…         ├ 20         …         ├ 40
malloc()  ┘            malloc()  ┘
```

$R(\text{malloc}|\text{request})$ = {20, 40}

Relational Debugging
– pinpoints root causes of performance problems

Relations between events represents relative performance
& general representation of performance root causes.

```
Request#1          Request#2
Mem usage:         Mem usage:
200MB              400MB
```

$$\boldsymbol{R}(\text{malloc}(\text{size})|\text{request})$$

$$= \{200\text{MB},\ 400\text{MB}\}$$

Relations can represent:
- Memory usage
- CPU cyces
- Network bandwidth
- Disk usage

…

Relational Debugging
– pinpoints root causes of performance problems

Relations between events represents relative performance
& general representation of performance root causes.

Request#1

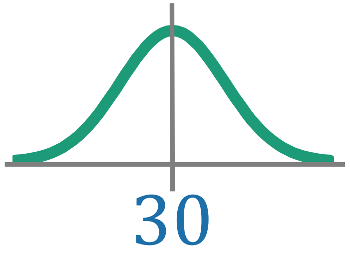
$\left.\begin{array}{l}\texttt{malloc()}\\ \dots\\ \texttt{malloc()}\end{array}\right\}20$

…

Request#2

$\left.\begin{array}{l}\texttt{malloc()}\\ \dots\\ \texttt{malloc()}\end{array}\right\}40$

$\boldsymbol{R}(\text{malloc}|\text{request}) =$

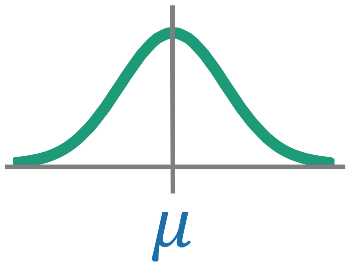


Relational Debugging
– pinpoints root causes of performance problems

Relations between events represents relative performance

& general representation of performance root causes.

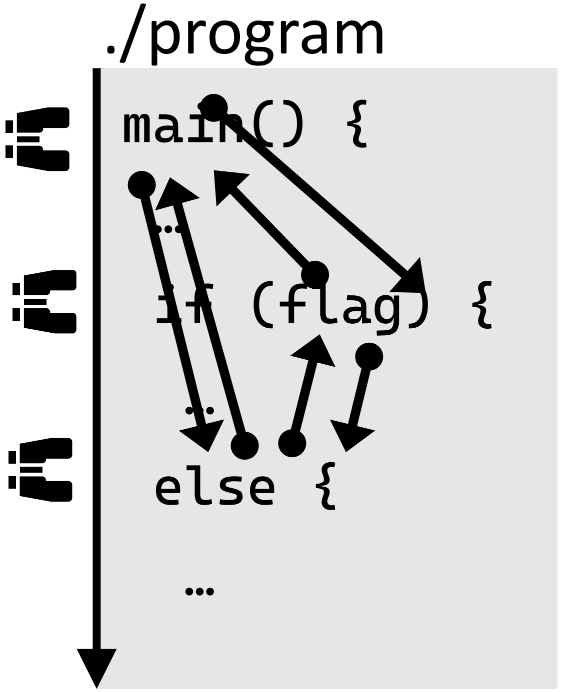
$$\boldsymbol{R}(\mathrm{B}|\mathrm{A}) =$$



"The # of event B's that causally dependent on an event A.

Relational Debugging
– pinpoints root causes of performance problems

Challenges
- Possibles relations in an execution are combinatorial
- Which ones capture the root cause of performance bug?



Relational Debugging
– pinpoints root causes of performance problems

Core idea:

locate most specific reference point to capture the root cause

```
main() {
  while (true) {
     handle_request();
  }
}
```

$\boldsymbol{R}(\text{malloc}|\text{main()})$ = 2GB ➔ 6GB

main() handle_request malloc

Relational Debugging
– pinpoints root causes of performance problems

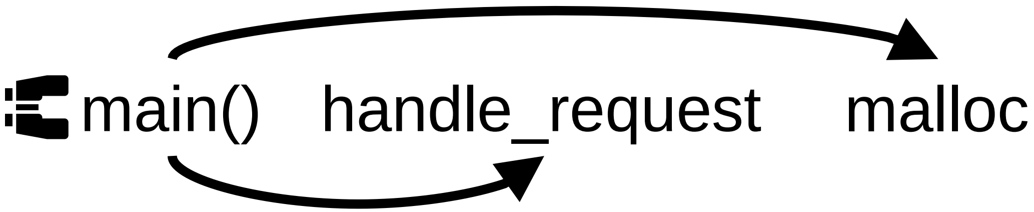
Core idea:

locate most specific reference point to capture the root cause

```
main() {
  while (true) {
     handle_request();
  }
}
```

$$\boldsymbol{R}(\text{malloc}|\text{main()}) \quad = 2\text{GB} \rightarrow 6\text{GB}$$

$$\text{Given } \boldsymbol{R}(\text{handle\_request}|\text{main()}) = 10 \rightarrow 10$$



Relational Debugging
– pinpoints root causes of performance problems

Core idea:

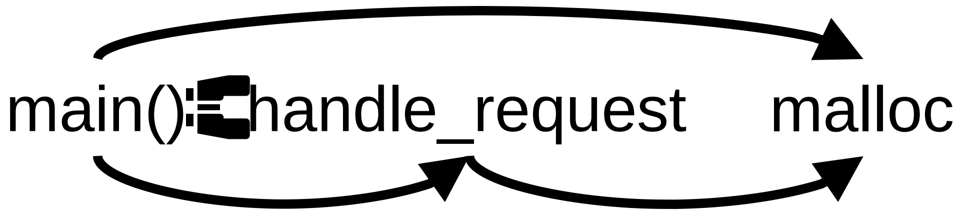
locate most specific reference point to capture the root cause

```
main() {
  while (true) {
     handle_request();
  }
}
```

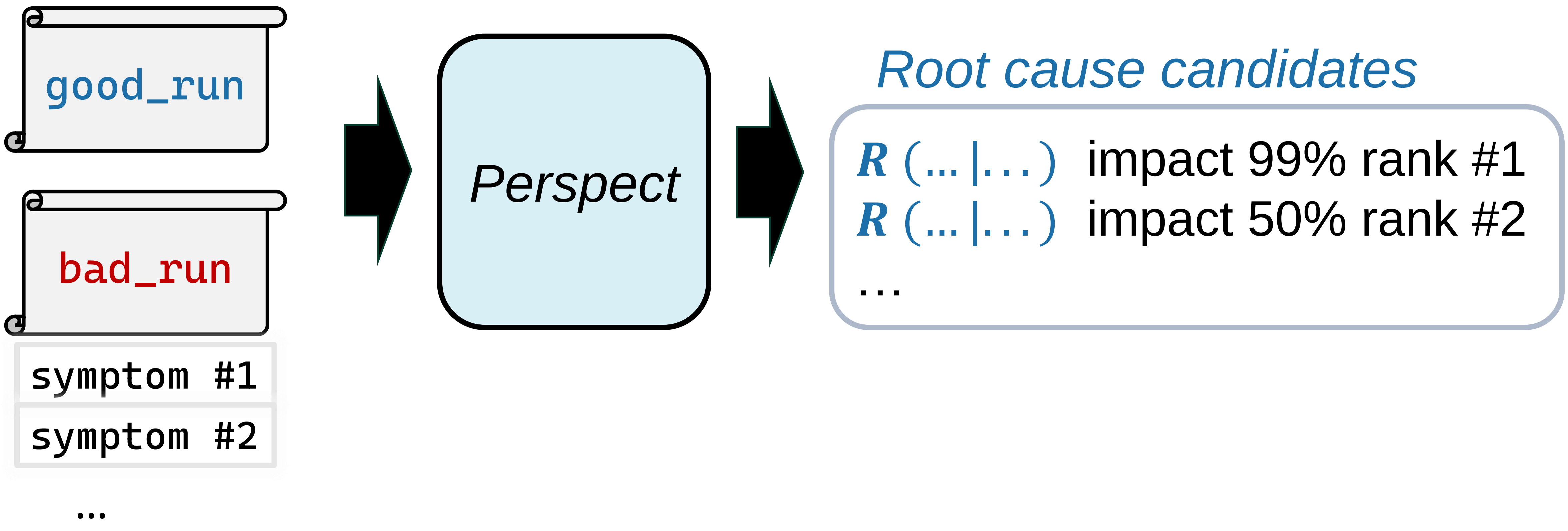
$\boldsymbol{R}(\text{malloc}|\text{main()})$ = 2GB → 6GB

Given $\boldsymbol{R}(\text{handle\_request}|\text{main()})$ = 10 → 10

Refine to $\boldsymbol{R}(\text{malloc}|\text{hand\_request})$ = 205MB → 315MB



Perspect implements Relational Debugging



Perspect implements Relational Debugging

Causal analysis

- Bootstrap with performance symptoms
- Identify causal predecessors of the symptoms

Relational debugging

Step1. Build relations at most general reference points

Step2. Filter relations that have not changed

Step3. Refine relations - move ref. points closer to symptom

Step4. Rank root cause candidates based on impact on perf.

Repeat

Go-909 – A memory leak bug

Go-909 causes “Severe memory problems on 32bit Linux”

```
for i := 0; i < 1000; i++ {
  r := make([]float64, 923521)
}
```

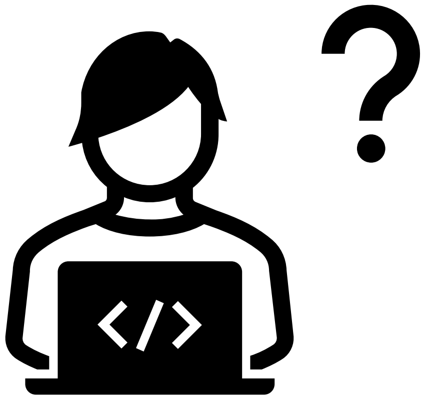
```
$./64bit_run
$ heap_size:   29MB

$./32bit_run
$ heap_size:  2075MB
```

- Impacted many workloads & Extensively discussed

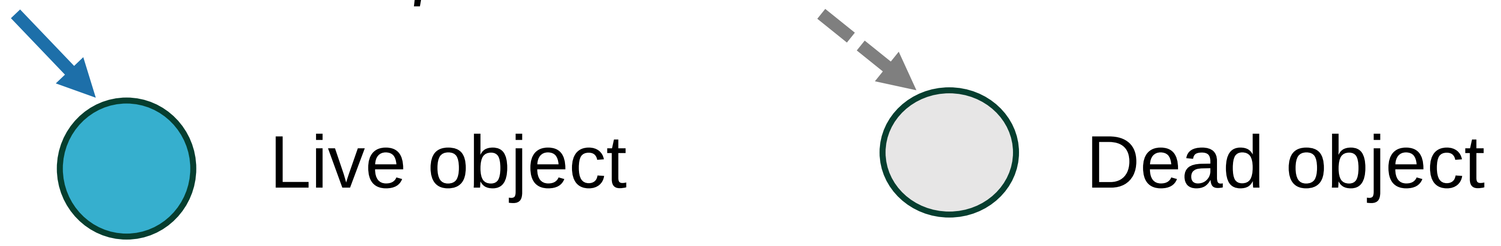
Diagnosing Go-909 was challenging

- Diagnosed through trial-and-error after more than a year
- Root cause breaks no program invariants/absolute predicates



The root cause of Go-909

```
void *p = malloc(…);        const int q = 0x8126890;
```



The root cause of Go-909

`void *p = malloc(…);`

`const int q = 0x8126890;`

Live object

GC: `mark` *object as reachable*

Dead object

GC: reclaim

The root cause of Go-909

```
void *p = malloc(…);
```

Live object

GC: `mark` *object as reachable*

```
const int q = 0x8126890;
```

Dead object

GC: `mark` *object as reachable*

Perspect pinpoints the root cause of Go-909

```
void *p = malloc(…);
```

```
const int q = 0x8126890;
```

Live object

GC: `mark` *object as reachable*

Dead object

GC: `mark` *object as reachable*

Root cause relation:

Good run: $\boldsymbol{R}(\text{malloc}|\text{mark\_object}) = \{1, 1, 1, 1, \ldots 1, 0\}$

Bad run: $\boldsymbol{R}(\text{malloc}|\text{mark\_object}) = \{0,0, 0, 0, \ldots 0, 1\}$

“The # of malloc events each mark event depends on.”

Perspect pinpoints the root cause of Go-909

```
mark() {… if points_to_heap(ptr) {mark(*ptr)}}
```

```
void *p = malloc(…);        const int q = 0x8126890;
```

Live object

Dead object

GC: `mark` *object as reachable*

GC: `mark` *object as reachable*

Root cause relation:

Good run: $\boldsymbol{R}(\text{malloc}|\text{mark\_object}) = \{1, 1, 1, 1, \ldots 1, 0\}$

Bad run: $\boldsymbol{R}(\text{malloc}|\text{mark\_object}) = \{0,0, 0, 0, \ldots 0, 1\}$

"The # of malloc events each mark event depends on."

Perspect pinpoints the root cause of Go-909

```
mark() {… if points_to_heap(ptr) {mark(*ptr)}}
```

```
void *p = malloc(…);        const int q = 0x8126890;
```

Live object

GC: `mark` *object as reachable*

Dead object

GC: `mark` *object as reachable*

Root cause relation:

Good run: $\boldsymbol{R}(\text{malloc}|\text{mark\_object}) = 0.99$

Bad run: $\boldsymbol{R}(\text{malloc}|\text{mark\_object}) = 0.01$

`Impact: 99% rank: 1/1`

Perspect on Go-909

Causal analysis

- Bootstrap with performance symptoms
- Identify causal predecessors of the symptoms

Relational debugging

Step1. Build relations

Step2. Filter relations

Step3. Refine relations

Repeat

Step4. Rank root cause candidates

Bootstrap with performance symptoms

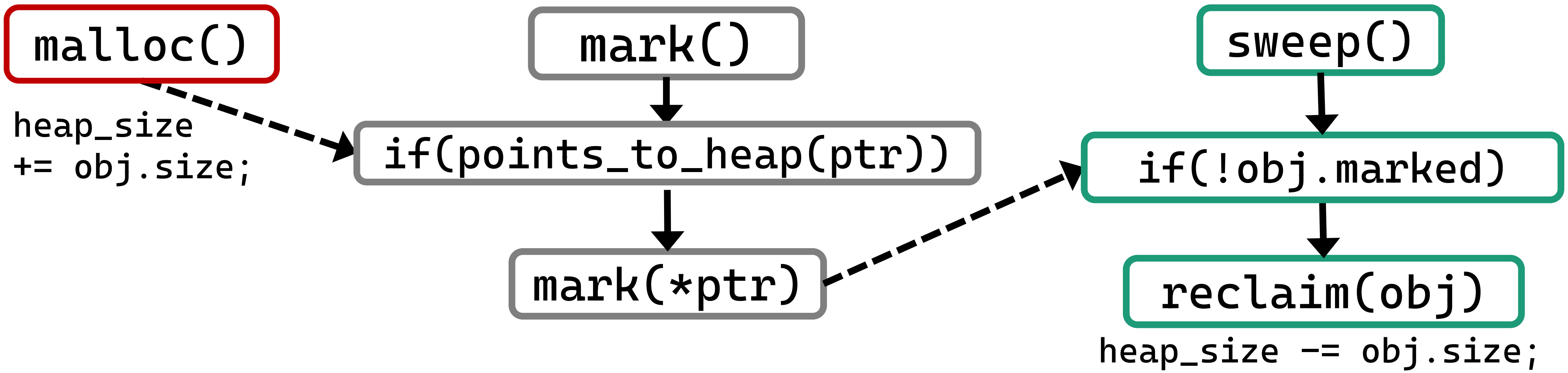
`malloc()`

```
heap_size
+= obj.size;
```

`reclaim(obj)`

```
heap_size -= obj.size;
```

Identify causal dependencies of the symptoms



Perspect automates relational debugging

Causal analysis

- Bootstrap with performance symptoms
- Identify causal predecessors of the symptoms

Relational debugging

Step1. Build relations

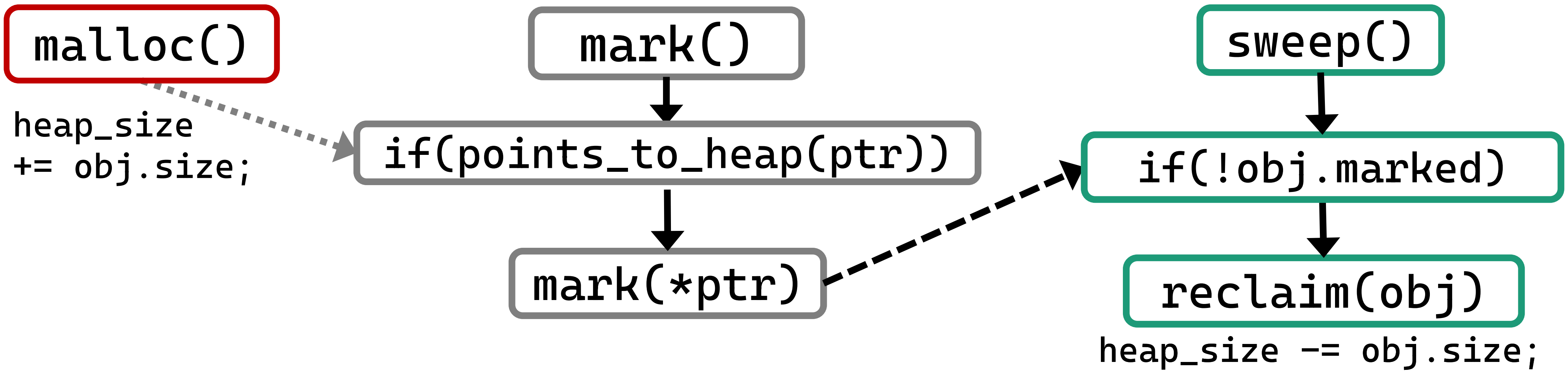
Step2. Filter relations

Step3. Refine relations

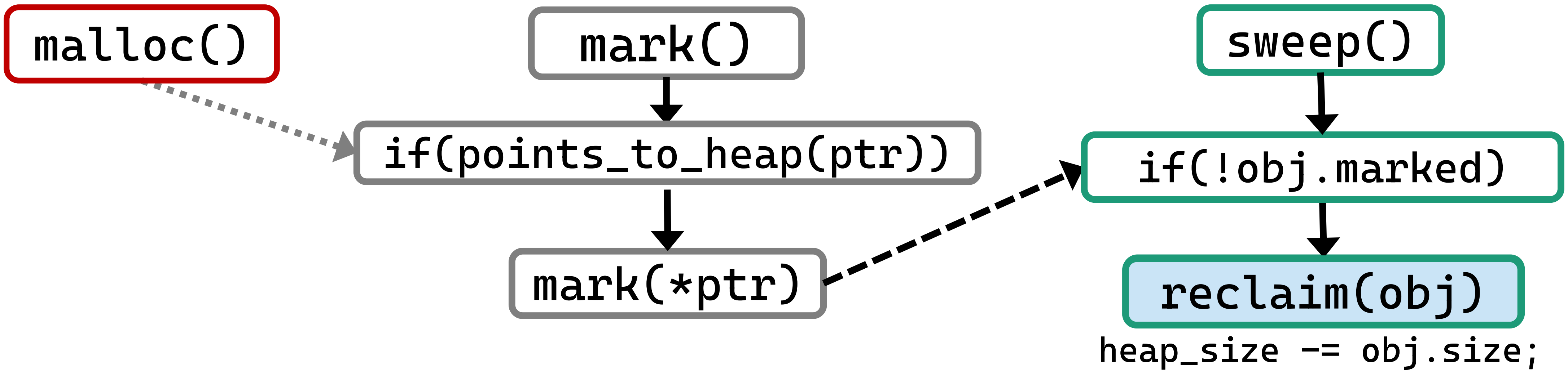
Repeat

Step4. Rank root cause candidates

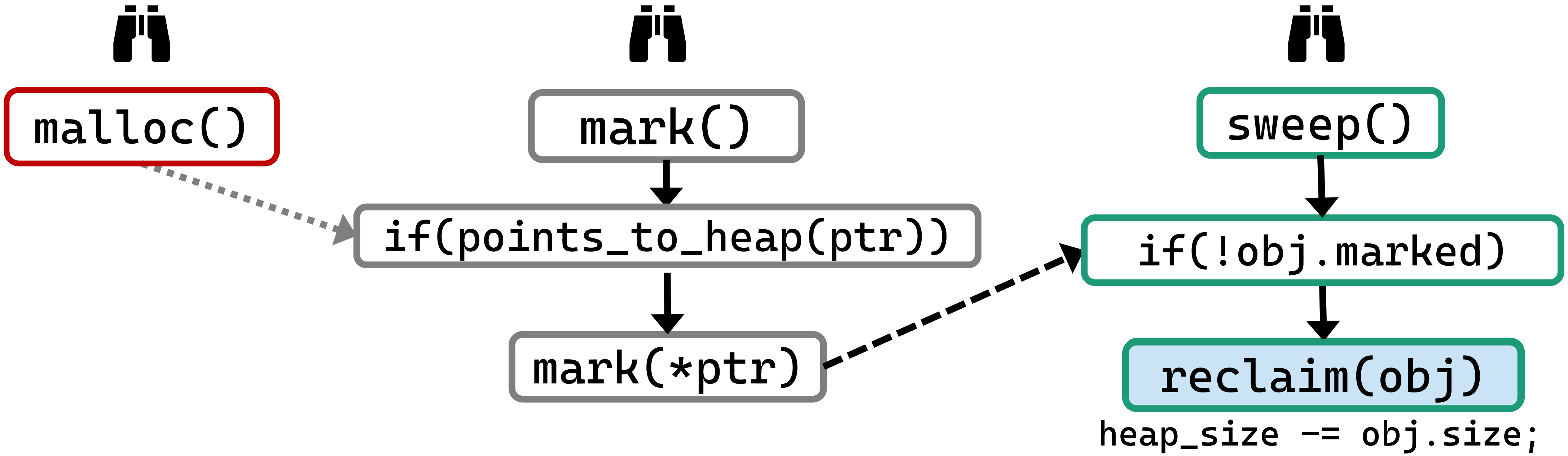
Step1. Build relations at most general reference points



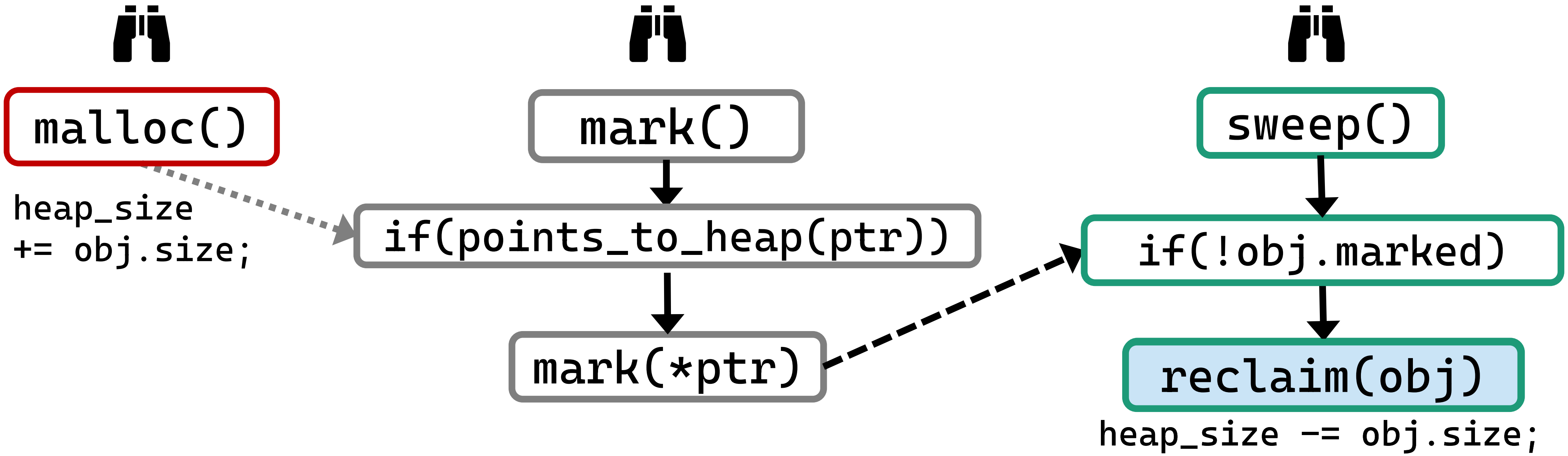
Step1. Build relations at most general reference points



Step1. Build relations at most general reference points

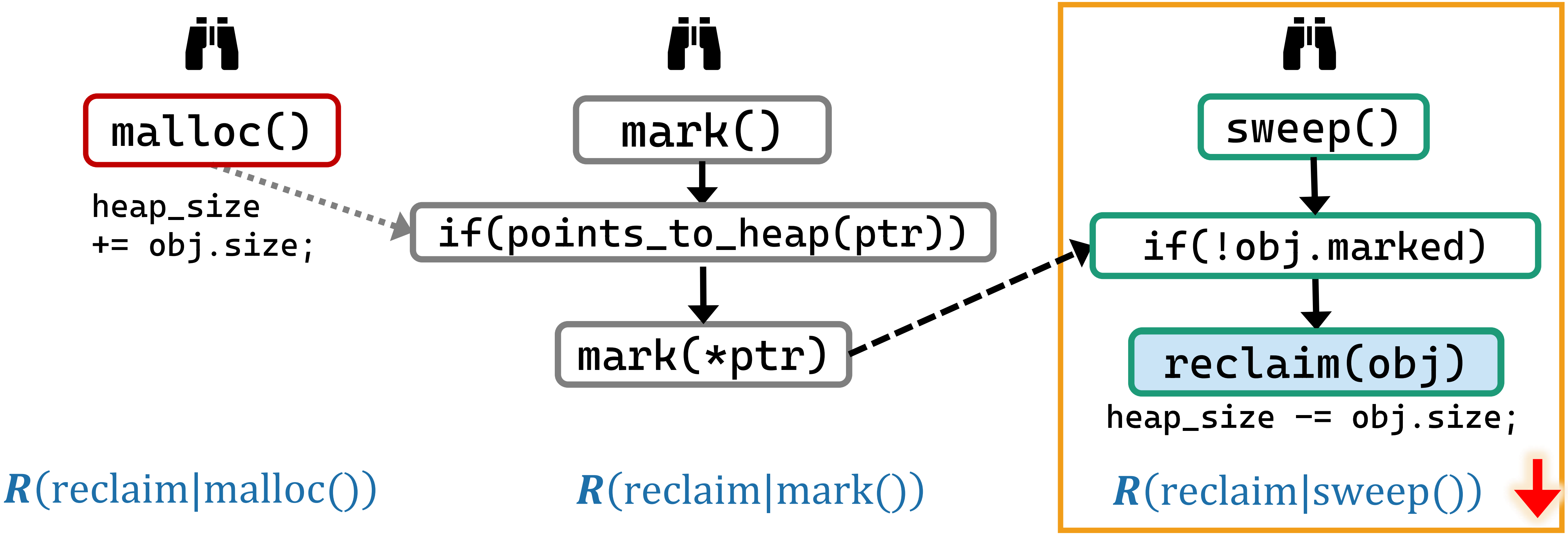


Step1. Build relations at most general reference points

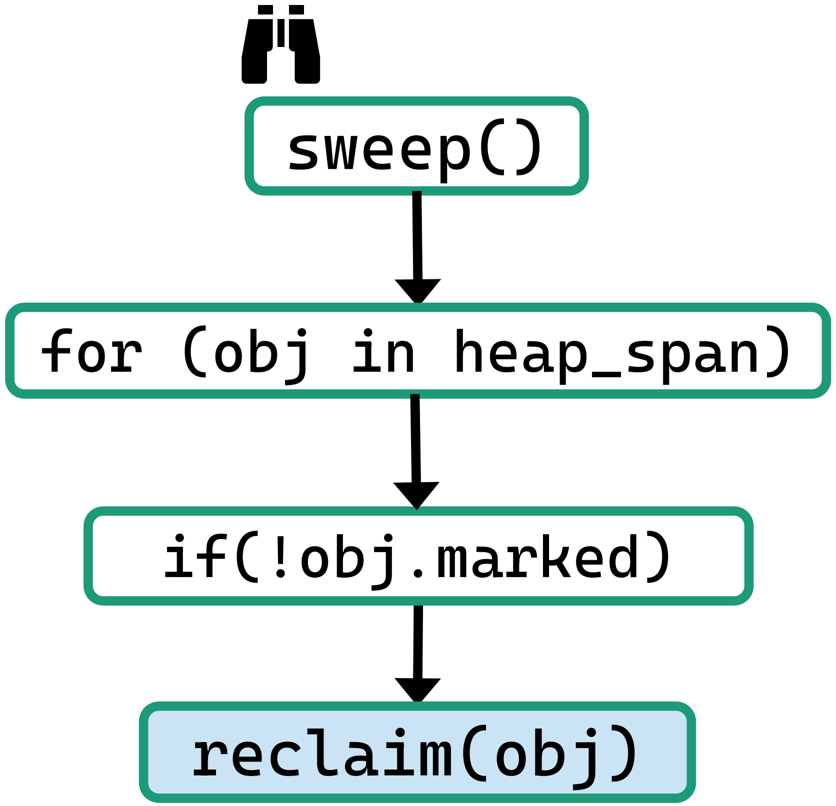


$\boldsymbol{R}(\text{reclaim}|\text{malloc()})$ $\boldsymbol{R}(\text{reclaim}|\text{mark()})$ $\boldsymbol{R}(\text{reclaim}|\text{sweep()})$

Step1. Build relations at most general reference points

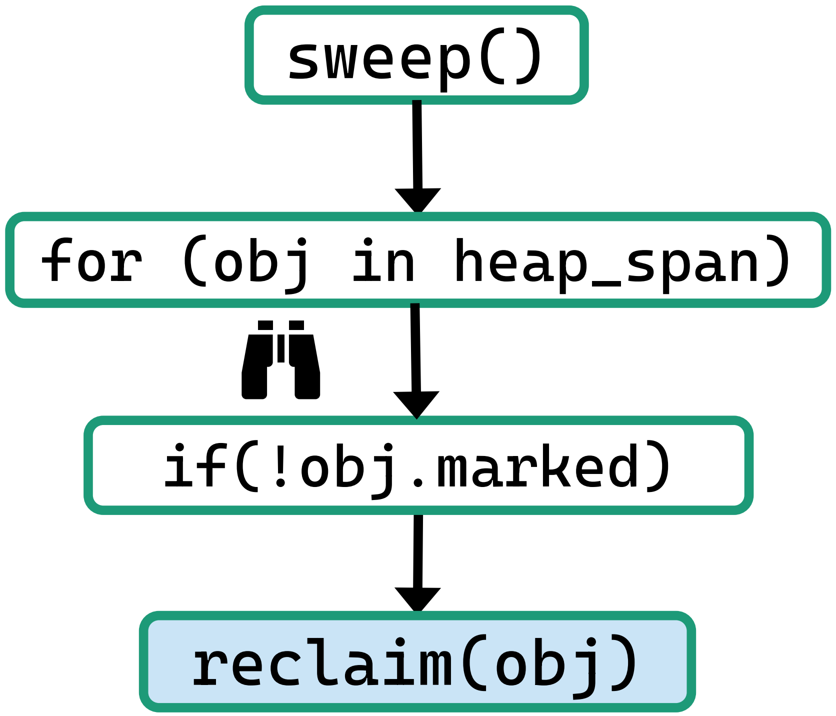


Refine relations – rule #1



$\boldsymbol{R}(\text{reclaim}|\text{sweep}())$

Refine relations – rule #1

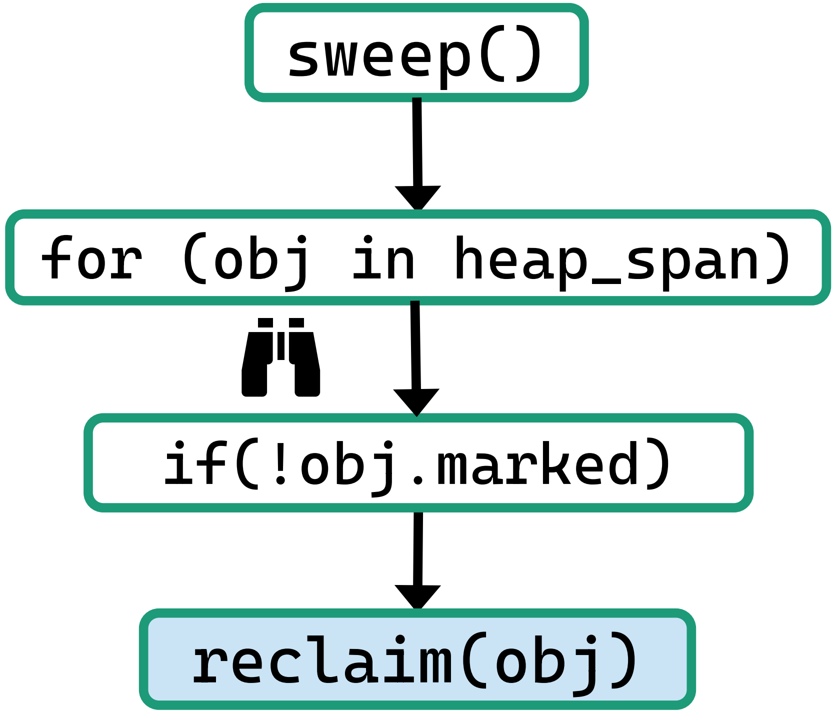


$$\boldsymbol{R}(\text{reclaim}|\text{sweep()})$$

Refine

$$\boldsymbol{R}(\text{reclaim}|\text{if(! obj. marked)})$$

Refine relations – rule #1



$$\boldsymbol{R}(\text{reclaim}|\text{sweep}())$$

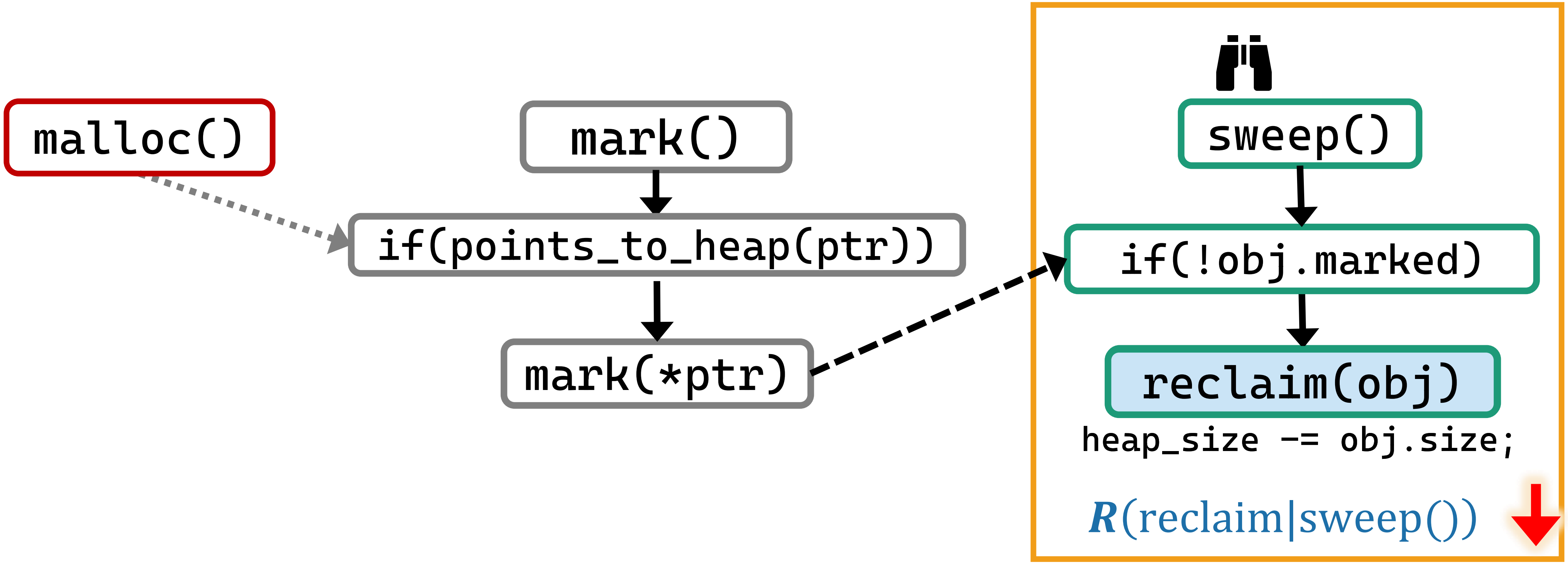
Refine

$$\boldsymbol{R}(\text{reclaim}|\text{if}(!\,\text{obj}.\,\text{marked}))$$

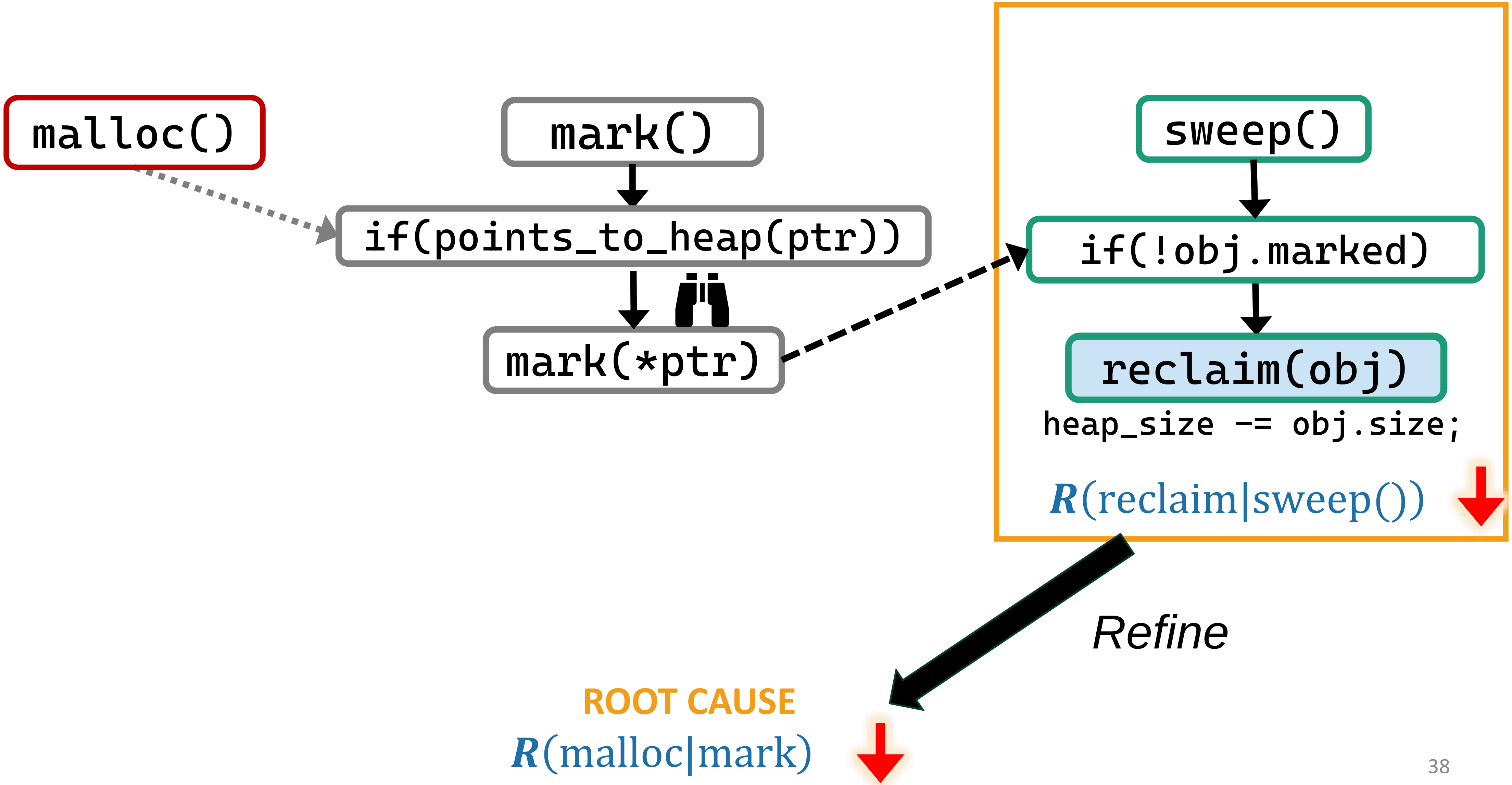
OK since

$$\boxed{\boldsymbol{R}(\text{if}(!\,\text{obj}.\,\text{marked})|\text{sweep}())}$$

Step2. Refine relations to capture root cause



Step2. Refine relations to capture root cause



Evaluating ***Perspect***'s effectiveness

- Evaluated on 12 bugs from Golang, Mongodb, Redis, Coreutils:
  10 bugs: *Perspect* diagnosed the root cause *successfully*
  1 bug: root cause in kernel, excluded from go system
  1 bug: unsuccessful due to significant code change

- Diagnosed two *open* bugs

> *"[Perspect's result] ties all the pieces together into a nice explanation."*
>
> —MongoDB developer's comment

Perspect’s usability and scalability

- Partcipants diagnose 2 cases **10.87 X faster** with *Perspect:* Go-909 and Mongodb-44991

- *Perspect* takes an average of **8 minutes** to run on most cases

Related work

Statistical debugging

- Identifies ***absolute*** predicates correlated with failure
- Requires labeling many executions as fail or success

X-Ray

- Captures root causes in input parameters & configurations

Other solutions

- Designed for specific patterns of bad performance

Conclusion

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Relational Debugging

- *Relation* btw. events captures relativeness of performance bugs
- Refine relations to narrow down to most specific root causes

Perspect (implements relational debugging)

- Pinpoints root causes of complex real-world bugs efficiently
- Helped diagnose two *open bugs*

https://gitlab.dsrg.utoronto.ca/dsrg/perspect