46th IEEE Symposium on Security and Privacy

CMASan: Custom Memory Allocator-aware Address Sanitizer

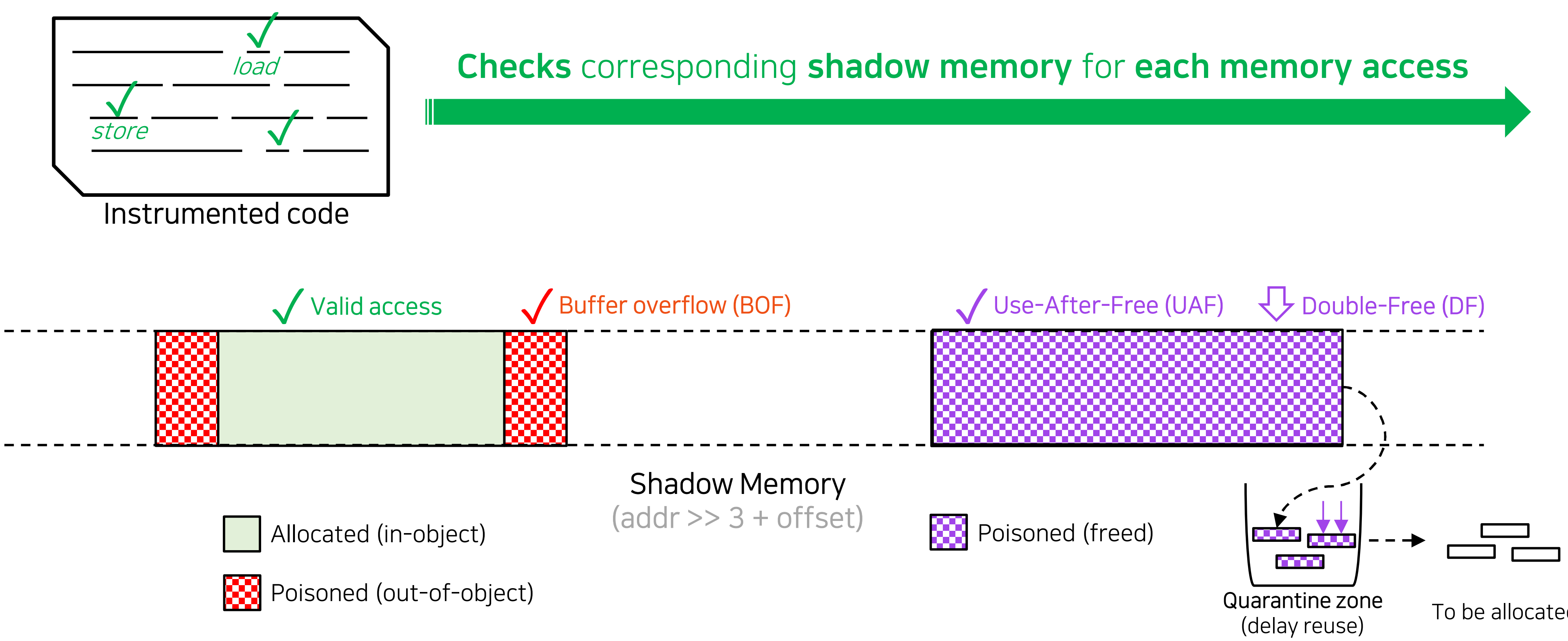
Junwha Hong[1], Wonil Jang[1], Mijung Kim[1], Lei Yu[2], Yonghwi Kwon[3], Yuseok Jeon[1]

[1] UNIST, [2] Rensselaer Polytechnic Institute, [3] University of Maryland

Junwha Hong qbit@unist.ac.kr
Wonil Jang dnjsdlf51@unist.ac.kr
Mijung Kim mijungk@unist.ac.kr
Lei Yu yul9@rpi.edu
Yonghwi Kwon yongkwon@umd.edu
Yuseok Jeon ys_jeon@korea.ac.kr

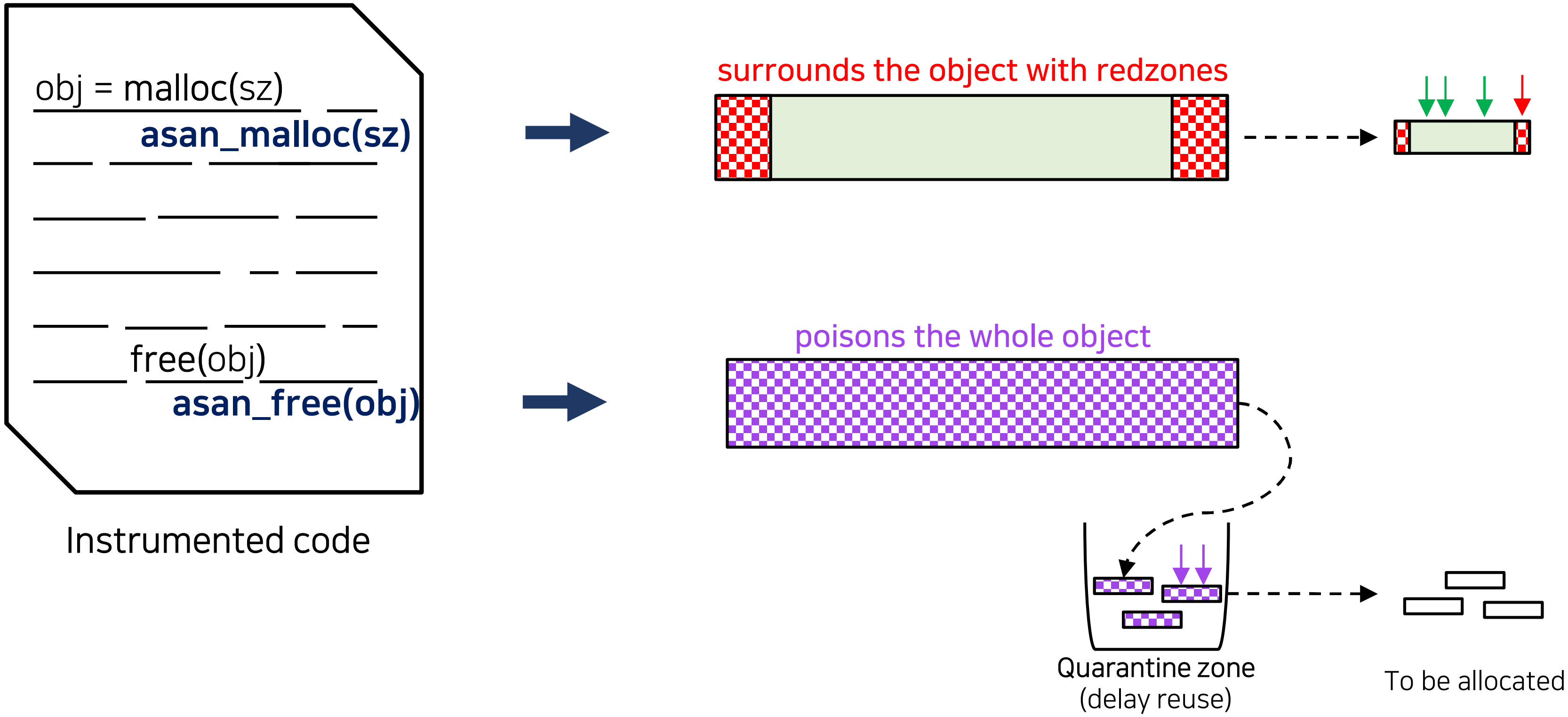
Address Sanitizer (ASan)

Address Sanitizer: runtime detection tool for detecting **memory bugs** with **shadow memory**



ASan's Internal Allocator

- ASan **replaces standard allocators with its internal allocator** to manage objects and redzones



What happens if the target application uses
a Custom Memory Allocator that ASan cannot replace?

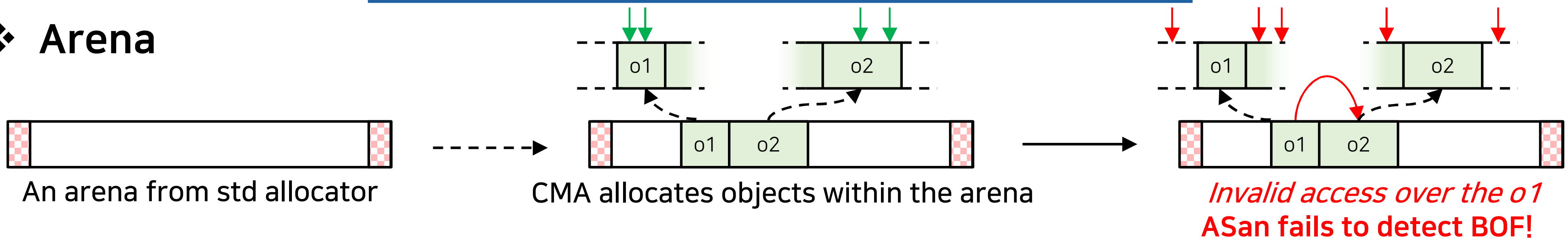
ASan will miss CMA-allocated objects,
so bugs affecting them go undetected

CMA Patterns Against ASan

- From 100 C/C++ GitHub projects, **identified 78 CMAs in 44 applications**
- Two **false negative** patterns: **Arena** (69%) & **Recycler** (45%)

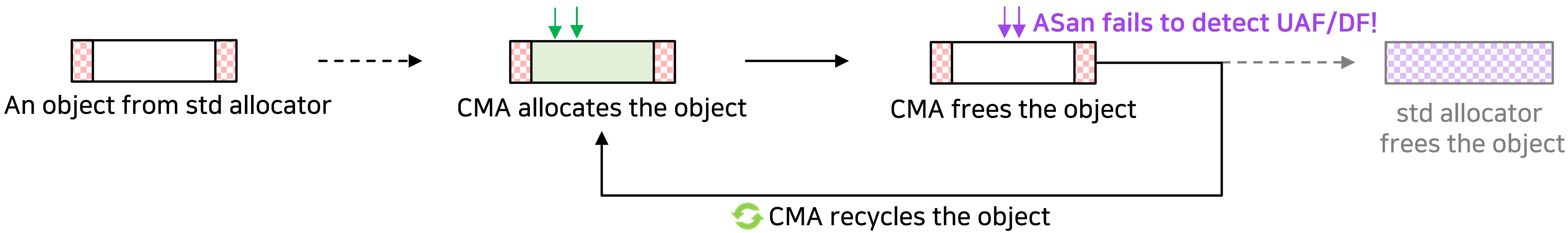
Need **redzones** between the **adjacent objects!**

- **Arena**



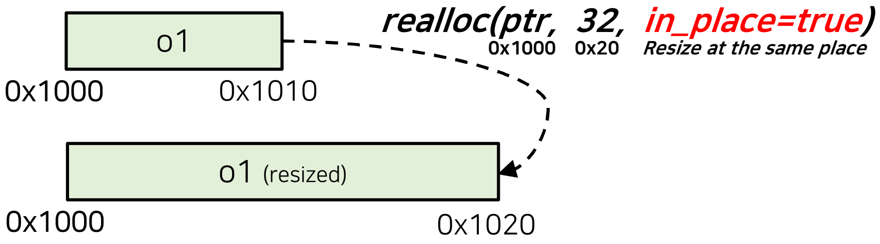
Need **Poisoning & Quarantine Zone** before they're **recycled!**

- **Recycler**

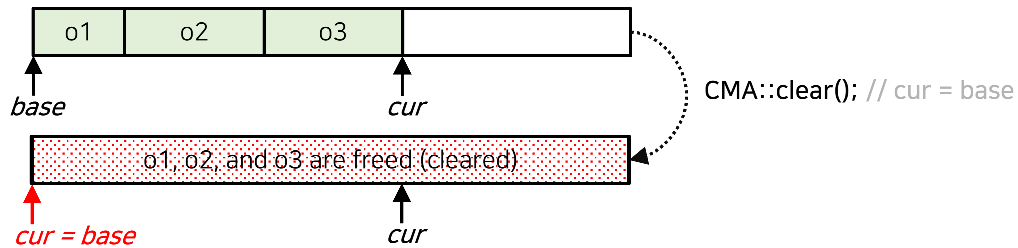


Existing Approaches: Shim & ASan API

- **Shim:** provide a mode to switch CMAs to standard allocators
  - **Not all CMAs are compatible** with standard allocators



An example of the inplace-realloc API

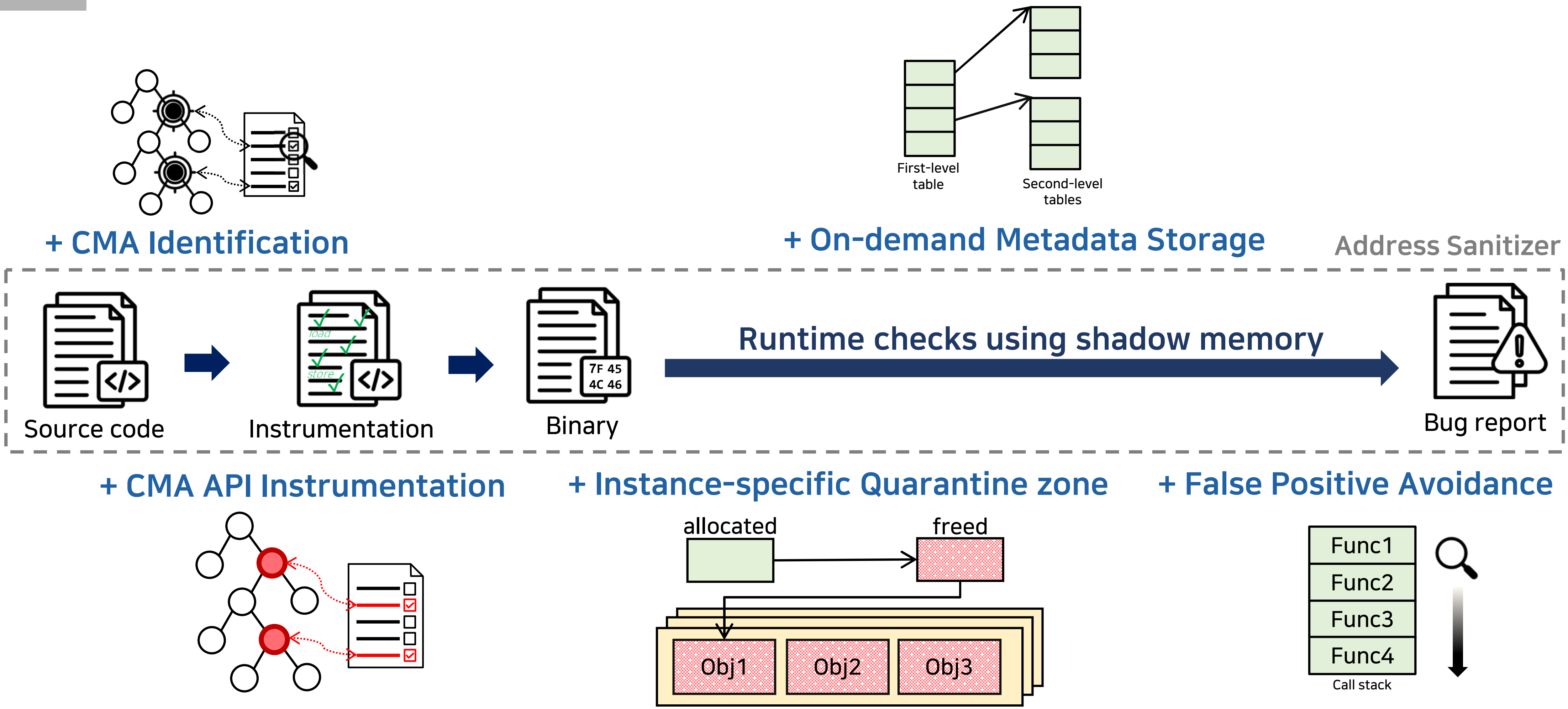


An example of the clear API

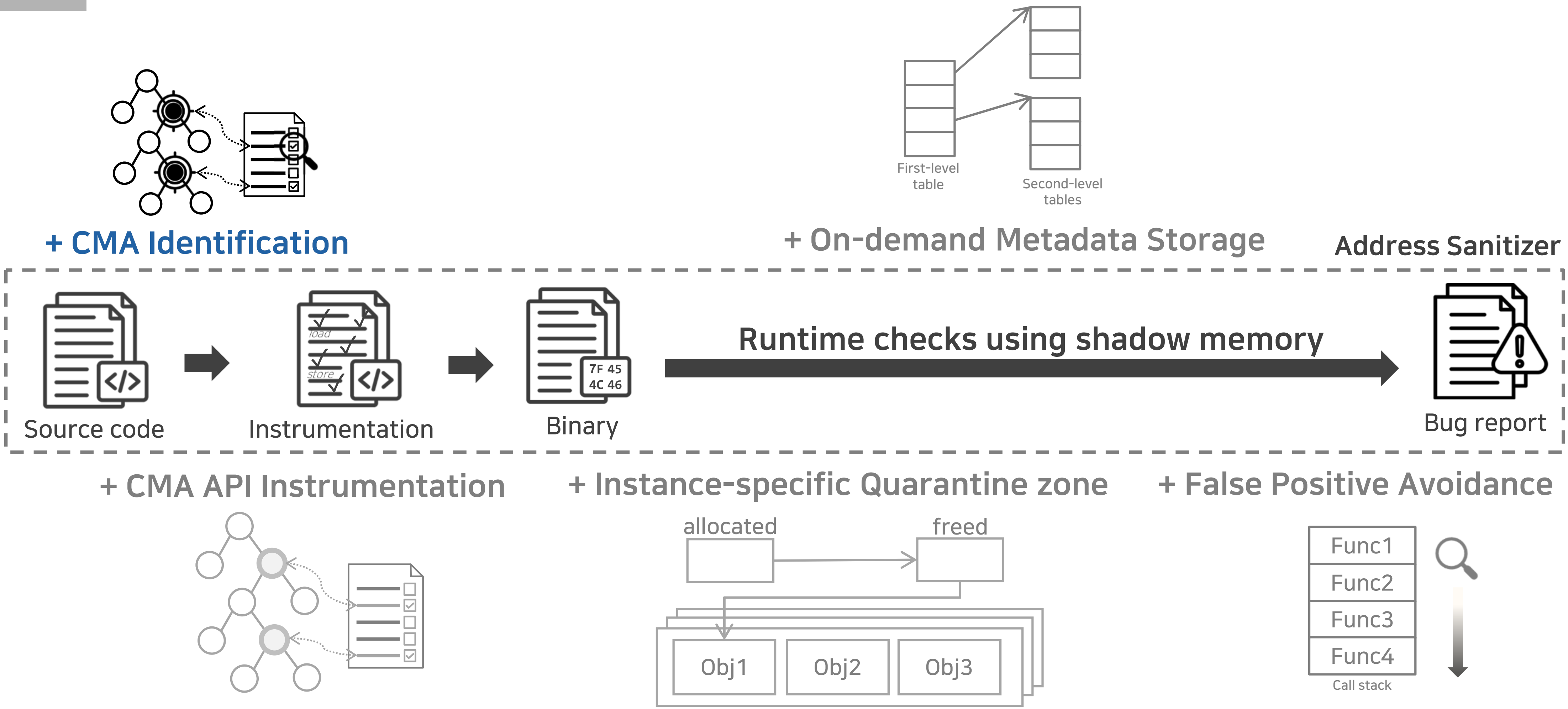
- **ASan Poisoning API:** API for manually poisoning ASan's shadow memory
  - **Redzone space** must be manually secured
  - **No Quarantine Zone** support

Both require **manual modification** with a **deep understanding of CMAs**

CMASan Overview



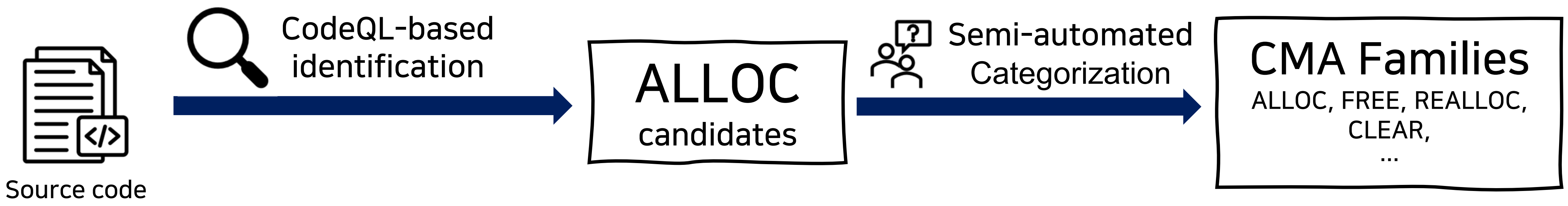
CMASan Overview



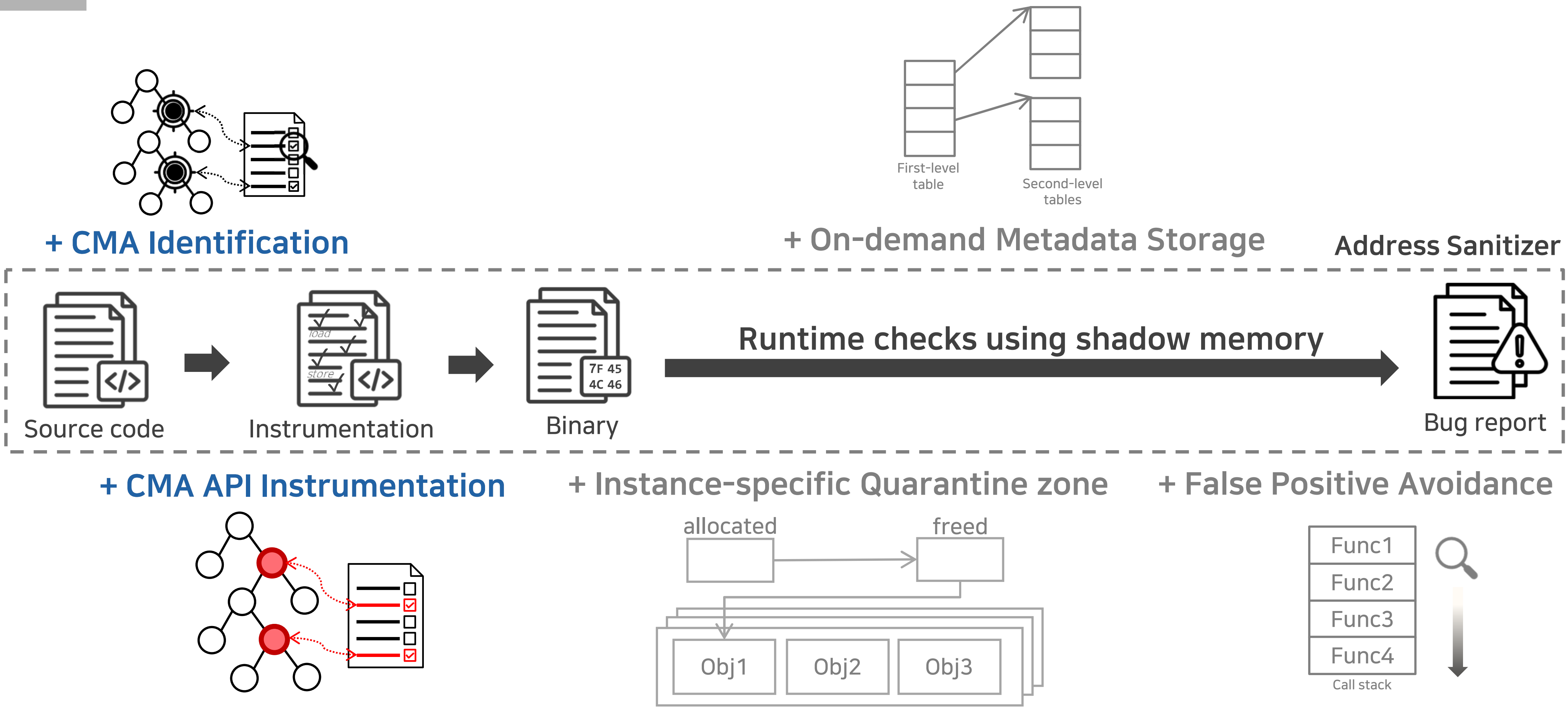
CMA Identification

- Extends CodeQL HeuristicAllocationFunction Query ⟶ ALLOC candidates
  (size argument flow, additional keywords)

- Semi-automated Categorization Procedure ⟶ collect family functions
  (with helper script for user convenience)



CMASan Overview

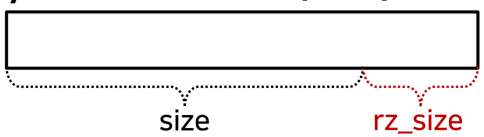


CMA API Instrumentation (1/2)

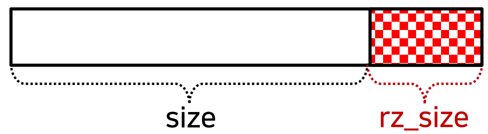
❖ ALLOC: secure redzone space

1. **Extend size** before the call

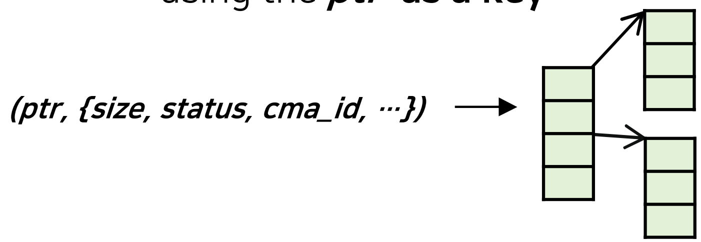
ptr = cma_alloc (size);+rz_size



2. **Poison the right side** of the returned object



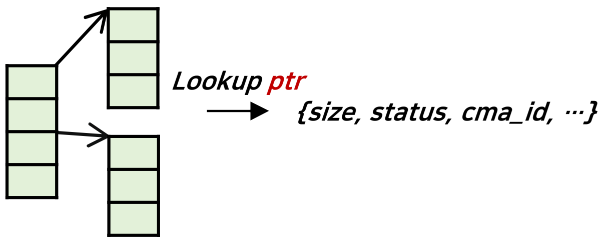
3. **Save metadata** into the metadata table using the ***ptr*** **as a key**



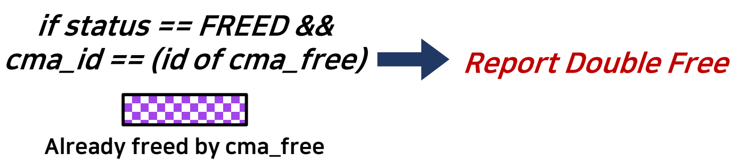
❖ FREE: retrieve size & poison

1. **Retrieve size** using the argument ***ptr***

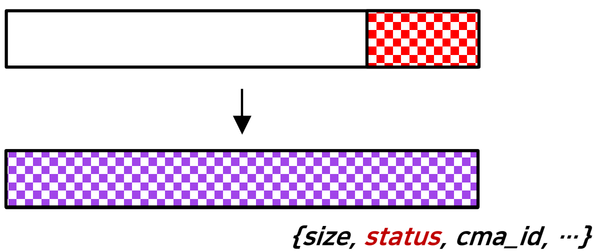
cma_free (ptr);



2. **Check double free** with status



3. **Poison** and **update metadata**

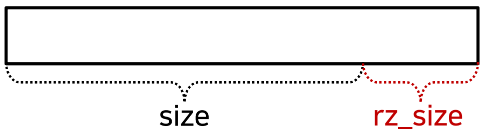


CMA API Instrumentation (2/2)

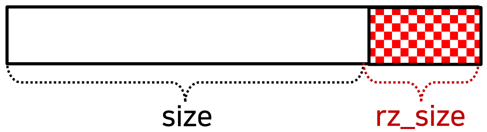
❖ REALLOC: Compare **$\mathbf{obj}_{\mathbf{old}}$** ↔ **$\mathbf{obj}_{\mathbf{new}}$** to distinguish **in-place behavior**

1. **Extend size** before the call

new_ptr = cma_realloc (old_ptr, size + rz_size);



2. **Poison the right side** of the returned object



3. **Compare *old_ptr*** and ***new_ptr*** and **poison (*old_ptr, size)*** if **different**

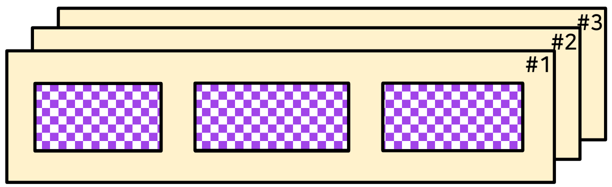
❖ CLEAR: record objects & poison them on a CLEAR call

o1 = $cma_{\#1}$::alloc (size1);

o2 = $cma_{\#1}$:: alloc (size2);

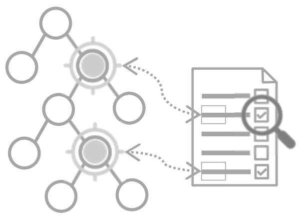
o3 = $cma_{\#1}$:: alloc (size3);

$cma_{\#1}$:: clear();

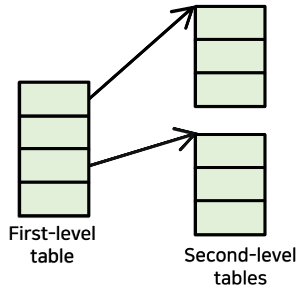


Allocation Zone #1~#N
(push at ALLOC, pop at CLEAR)

CMASan Overview



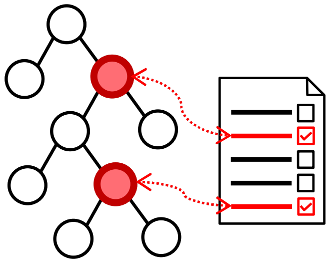
+ CMA Identification



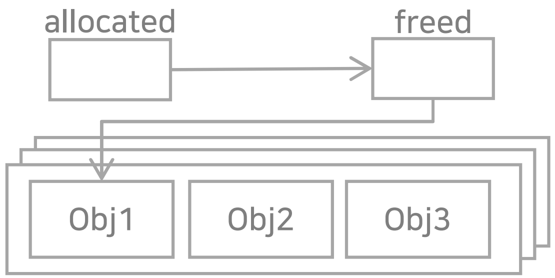
+ On-demand Metadata Storage

Address Sanitizer

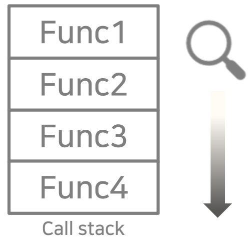
+ CMA API Instrumentation



+ Instance-specific Quarantine zone

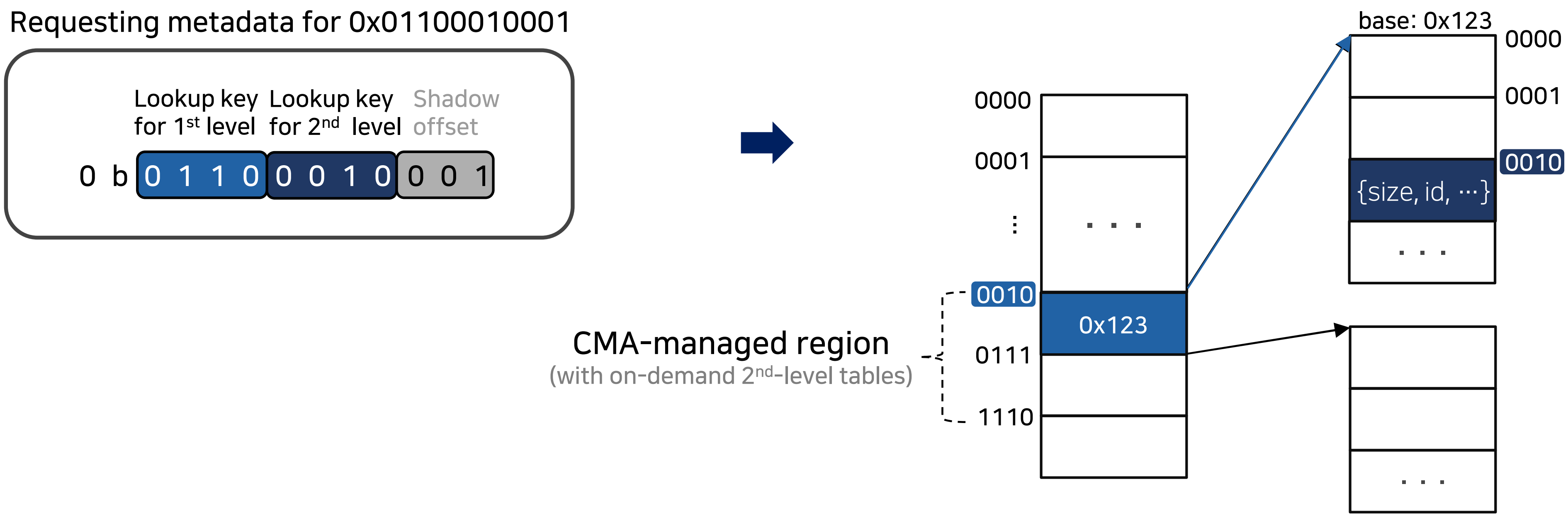


+ False Positive Avoidance

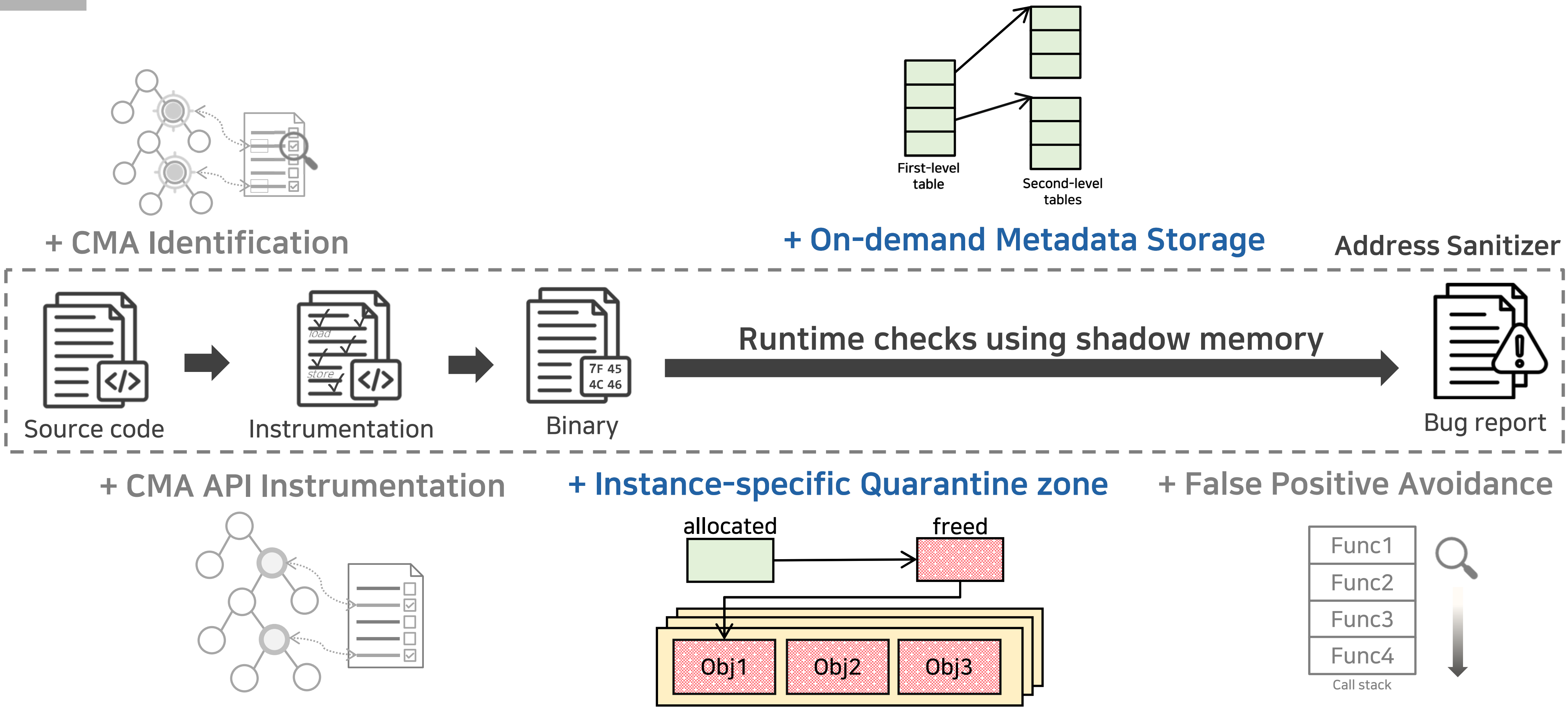


On-demand Metadata Storage

- **Observation**: CMAs **manage their objects** within a relatively **narrow memory region** (e.g., arena)
  - CMASan utilizes a **two-level table** to manage the metadata of CMA objects
  - The 2nd-level tables are **allocated on demand** for CMA-managed regions



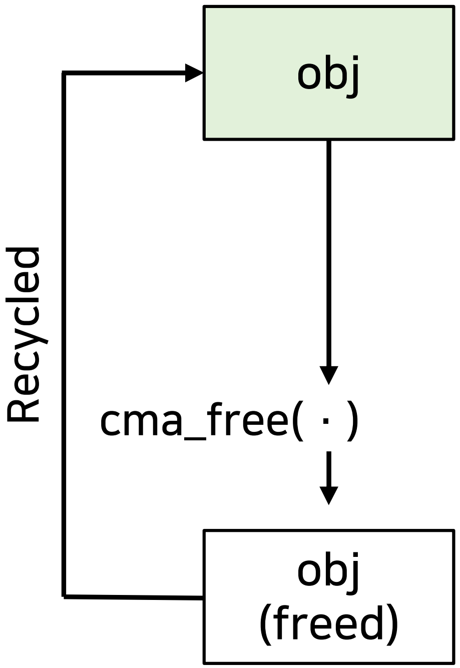
CMASan Overview



Instance-specific Free-delaying Quarantine Zone (1/2)

❖ **Objective 1 (Free-delaying):** Delay the recycling of objects while preserving CMAs

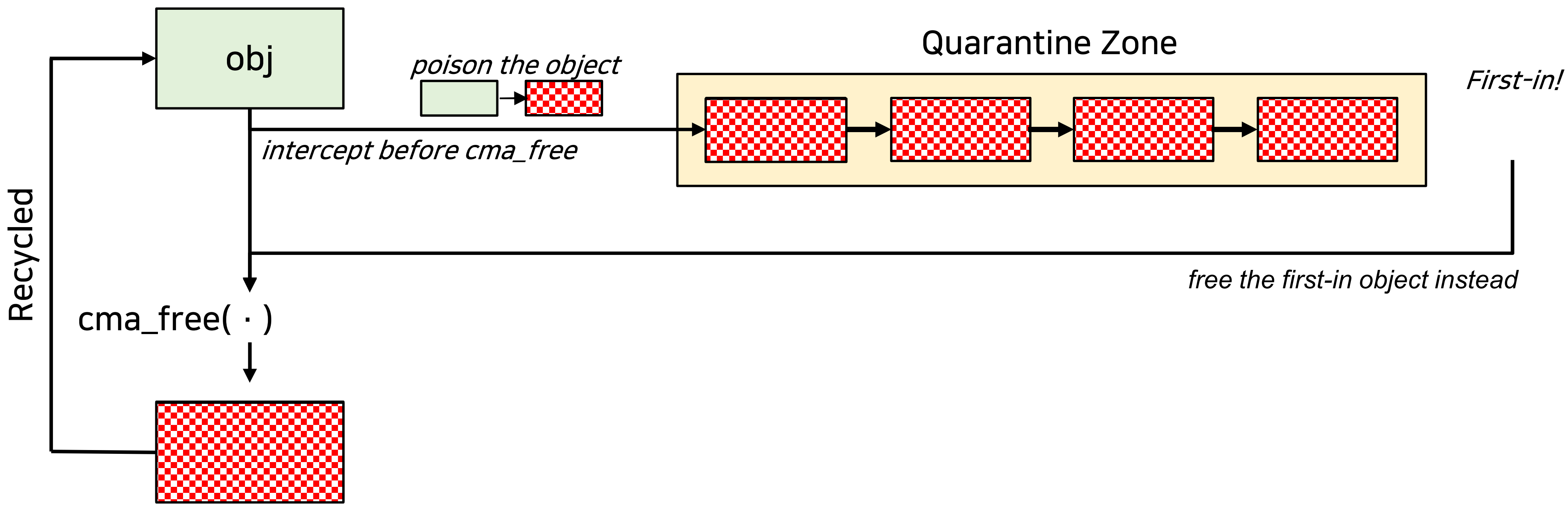
Lifecycle of CMA object



Instance-specific Free-delaying Quarantine Zone (1/2)

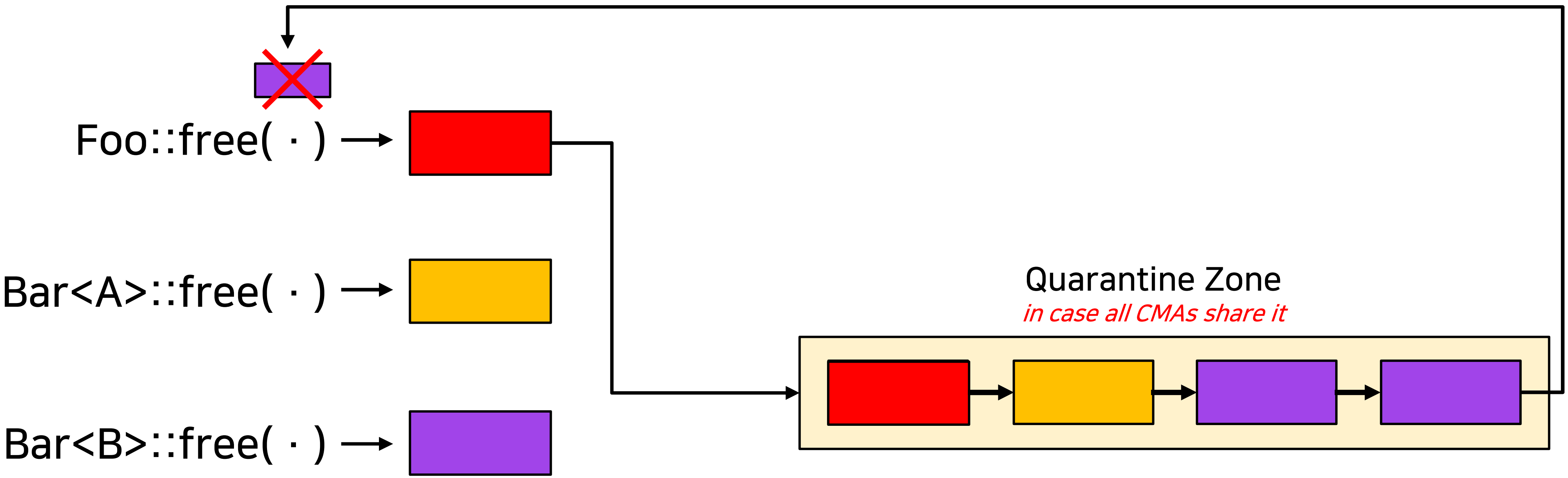
- **Objective 1 (Free-delaying):** Prevent the recycling of objects while preserving CMAs

Lifecycle of CMA object w/ Free-delaying



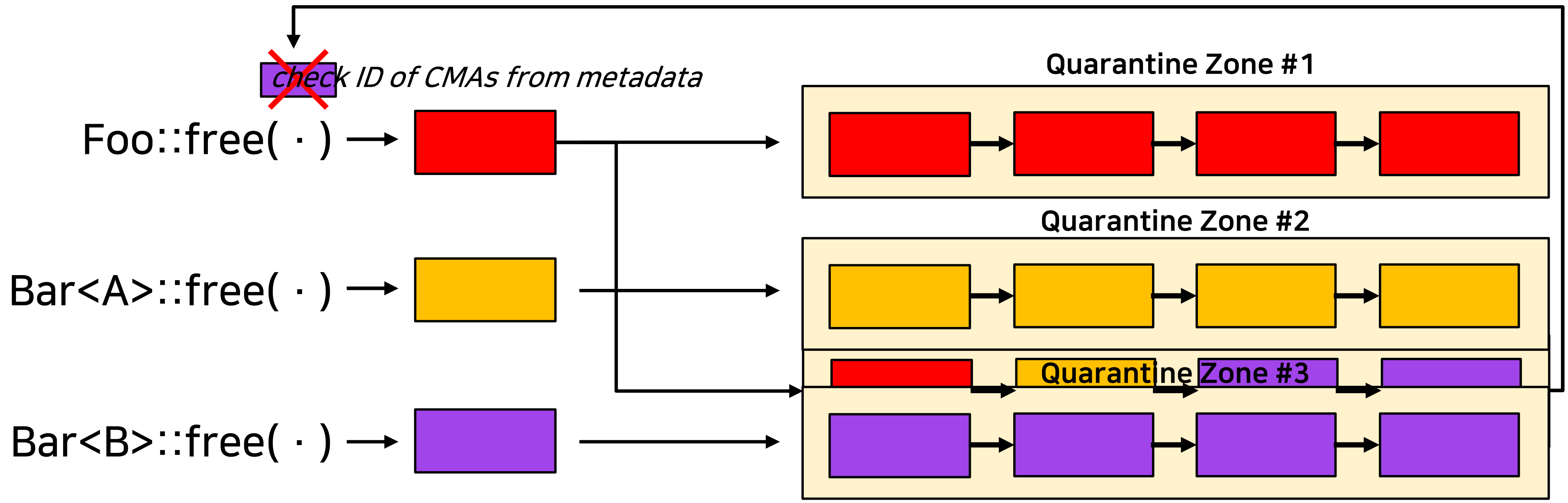
Instance-specific Free-delaying Quarantine Zone (2/2)

❖ **Objective 2 (Instance-specific):** Distinguish QZ between different CMA instances

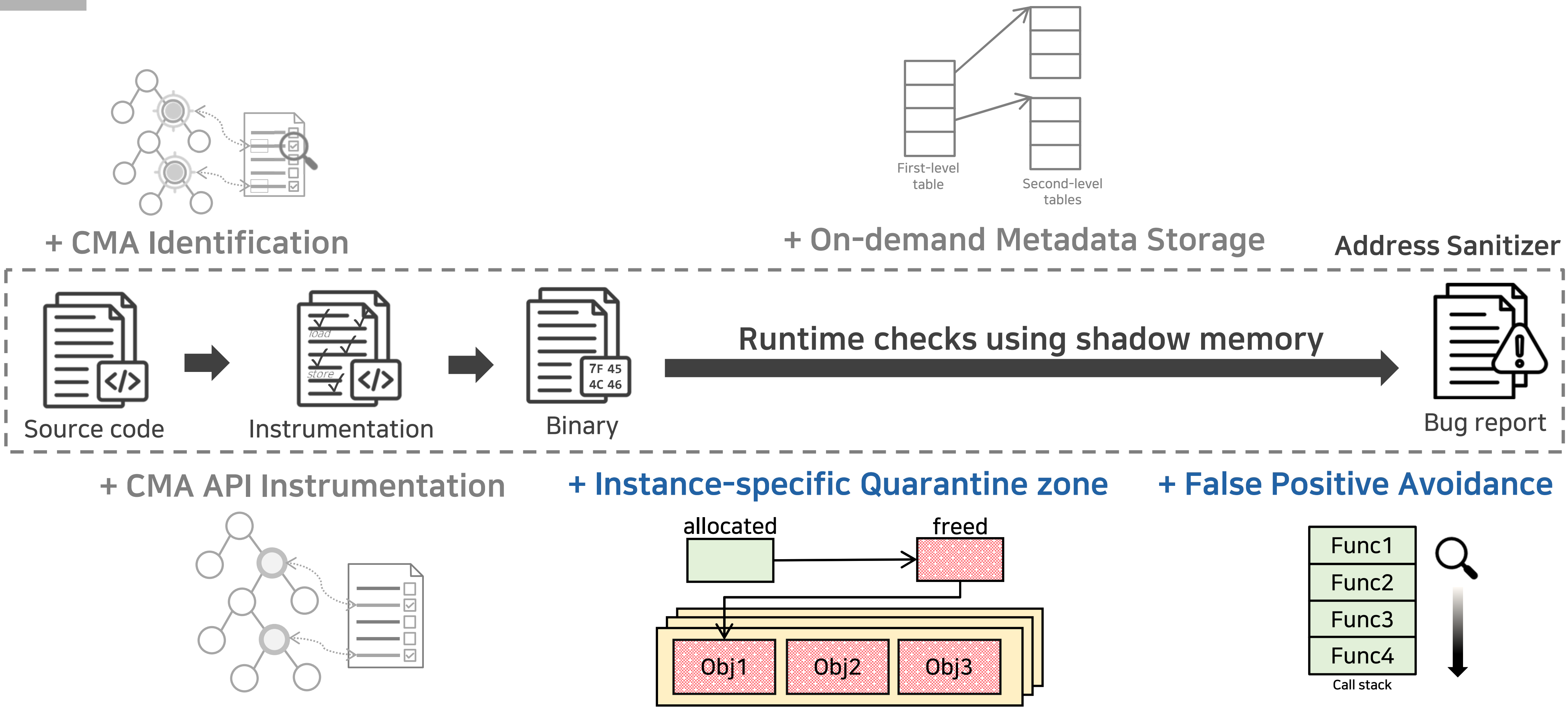


Instance-specific Free-delaying Quarantine Zone (2/2)

❖ **Objective 2 (Instance-specific):** Distinguish QZ between different CMA instances

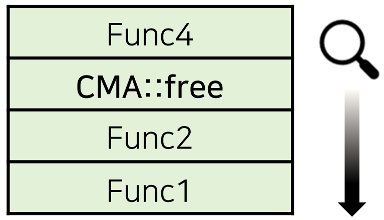


CMASan Overview



False Positive Avoidance

- **Suppress reports in CMA APIs** using call stack
  - CMAs often legitimately access poisoned objects (e.g., metadata)



- Activate only **the outermost CMA's instrumentation** using call stack
  - REALLOC often calls ALLOC and FREE (incorrect redzone sizes/double-free FP)

- Return the **original object size** on **size-querying API** calls
  - To CMAs, redzones are part of the object

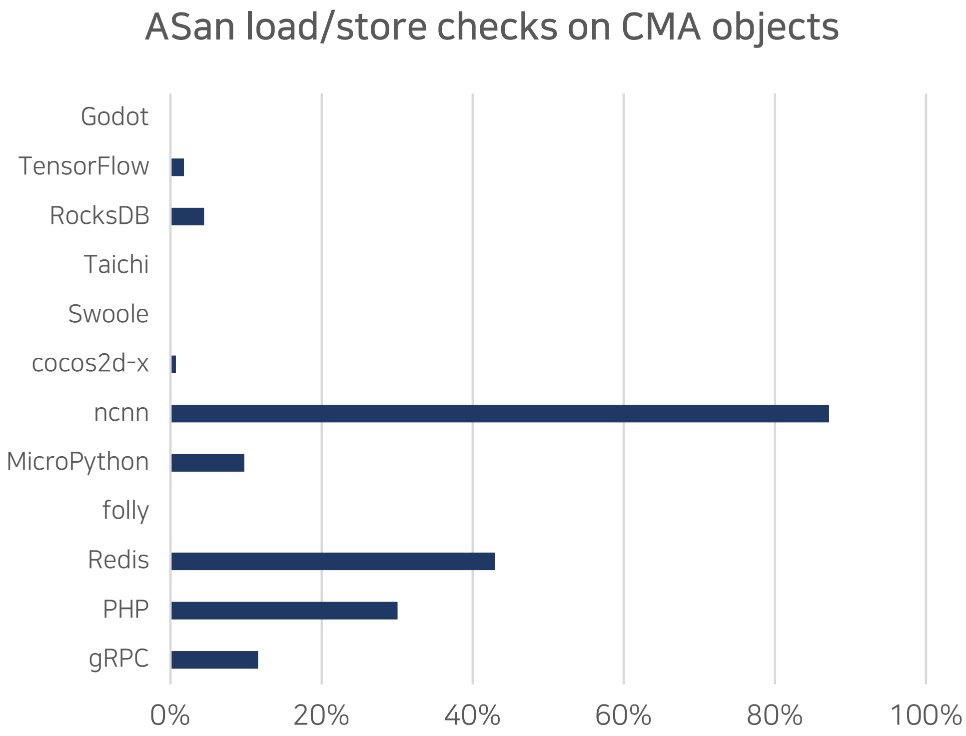
CMA::get_object_size(obj) – **rz_size**

Evaluation Setup

- **Target**: Top **12 C/C++ applications** that utilize CMAs from GitHub
- **Comparison Targets**: **ASan** and **Goshawk** (Static Analyzer for UAF/DF detection)
- **Evaluation Metrics**:
  - Detection Coverage
  - Performance Overhead
  - False Positive Avoidance
  - Bug Detection Capability
  - Unknown Bug Detection

Detection Coverage

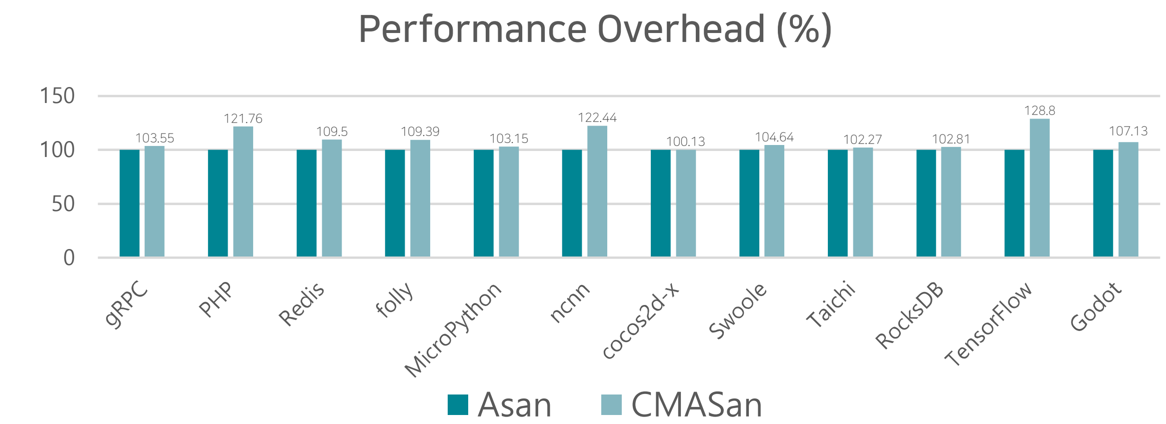
Application	CMA Objects
gRPC	623,109,396
PHP	66,431,503
Redis	26,867,307
folly	1,087,644
MicroPython	839,561
ncnn	95,741
cocos2d-x	29,504
Swoole	704
Taichi	468
RocksDB	97,101,151
TensorFlow	92,900
Godot	298

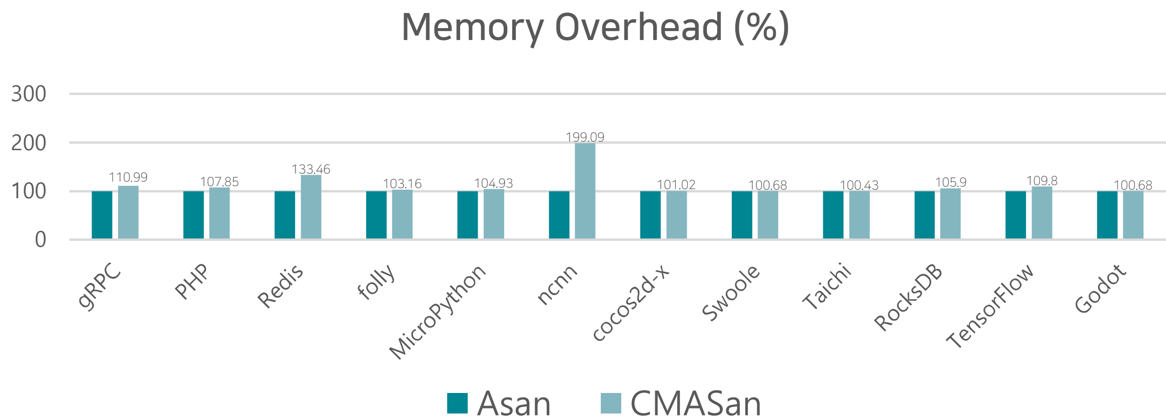


CMASan recognizes **many CMA objects** overlooked by ASan

Up to **87%** of **load/store checks** are on CMA objects

Performance Overhead





CMASan incurs **1.096x** performance and **1.148x** memory overhead compared to ASan

Unknown Bug Detection

Bug ID	Application	Bug Type
CVE-2023-7152	MicroPython	UAF
CVE-2023-7158	MicroPython	BOF
CVE-2023-8946	MicroPython	BOF
CVE-2024-8947	MicroPython	UAF
CVE-2024-8948	MicroPython	BOF
Issue #13004	MicroPython	BOF
Issue #13046	MicroPython	BOF
Issue #13220	MicroPython	BOF
Issue #13428	MicroPython	BOF
Issue #136-1	qhull	UAF
Issue #136-2	qhull	UAF
PR #2213	RapidJSON	BOF
PR #2244	RapidJSON	UAF
PR #2256	RapidJSON	UAF
CVE-2023-7104	SQLite3	BOF
Issue #13230	PHP	UAF
GHSA-rwp7-7vc6-8477	PHP	UAF
Issue #8501	Taichi	BOF
Issue #5734	ncnn	BOF

CMASan detects
19 previously unknown bugs
including ones undetected for
9 years and **2 years** in SQLite3 & PHP
(6 CVEs , 12 confirmed, 7 patched)

ASan and Goshawk miss all 19 bugs
(no CMA support / missing CMA APIs / complex paths)

Conclusion

- **CMASan effectively detects all types of CMA-related memory bugs**
  - Identifies 19 previously Unknown Bugs
  - 9.63% performance overhead compared to native ASan
  - Extends ASan's coverage without replacing CMAs

46th IEEE Symposium on Security and Privacy

Thank You

[Paper]

CMASan : Custom Memory Allocator-aware Address Sanitizer

Junwha Hong, Wonil Jang, Mijung Kim, Lei Yu, Yonghwi Kwon, Yuseok Jeon

[Open Source]

CMASan GitHub Repository

https://github.com/S2-Lab/CMASan