Enhancing Checked C with Temporal Memory Safety

Jie Zhou
University of Rochester

John Criswell
University of Rochester

David Tarditi
Microsoft

Checked C: A safe dialect of C

A little background on Checked C

Initiator: Galen Hunt (URCS alumnus) at Microsoft Research in 2015

Project leader: David Tarditi

Major collaborators outside Microsoft:
Michael Hicks, Andrew Ruef (graduated), Ray Chen, and Hasan Touma (UMD)
Aravind Machiry (UCSB), Jorge Navas (SRI), Arie Gurfinkel (U of Waterloo)
Sam Elliott (graduated) and Anna Kornfeld Simpson (U of Washington)

A little background on Checked C

Initiator: Galen Hunt (URCS alumnus) at Microsoft Research in 2015

Project leader: David Tarditi

Major collaborators outside Microsoft:
Michael Hicks, Andrew Ruef (graduated), Ray Chen, and Hasan Touma (UMD)
Aravind Machiry (UCSB), Jorge Navas (SRI), Arie Gurfinkel (U of Waterloo)
Sam Elliott (graduated) and Anna Kornfeld Simpson (U of Washington)

"The world builds on C. …… We cannot get rid of C. ……
I make long-term bets and history shows I'm good at it. …..
I bet the future of C is a safe dialect of C."

- Galen Hunt

Memory Safety

Spatial Memory Safety

- out-of-bounds access
- null-pointer dereference

Temporal Memory Safety

- Use-After-Free (UAF)
- Double Free

Focus of current Checked C

Aspects of Memory Safety Solutions

Aspects	What we want
Performance Overhead	as low as possible
Memory Overhead	as low as possible
Backward Compatibility	compatible with legacy code
Generality	general
Programmer Control	high
Programmer Efforts	as little as possible
Soundness	sound
Completeness	complete
Require Source Code	no

Checked C's Choices

Aspects	What we want	Checked C's Choice
Performance Overhead	as low as possible	✓
Memory Overhead	as low as possible	✓
Backward Compatibility	compatible with legacy code	✓
Generality	general	✓
Programmer Control	high	✓
Programmer Efforts	as little as possible	medium
Soundness	sound	✓
Completeness	complete	✗
Require Source Code	no	✗

Memory Safety

Spatial Memory Safety

- out-of-bounds access
- null-pointer dereference

Focus of current Checked C

Temporal Memory Safety

- Use-After-Free (UAF)
- Double Free

Focus of this talk

Use-After-Free (UAF): dereferencing a pointer after the pointed memory object has been freed.

Use-After-Free (UAF) Vulnerability

portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-1199

Microsoft | MSRC Report an issue Customer guidance Engage Who we are Blogs

Security Update Guide > Details

CVE-2019-1199 | Microsoft Outlook Memory Corruption Vulnerability

Security Vulnerability

Published: 08/13/2019 | Last Updated : 08/13/2019
MITRE CVE-2019-1199

A remote code execution vulnerability exists in Microsoft Outlook when the software fails to properly handle objects in memory. An attacker context of the current user. If the current user is logged on with administrative user rights, an attacker could take control of the affected sys create new accounts with full user rights. Users whose accounts are configured to have fewer user rights on the system could be less impact

lares.com/use-after-free-uaf-vulnerability-cve-2019-1199-in-microsoft-outlook/

LARES SERVICES INDUSTRIES RESOURCES ABOUT CONTACT

BLOG RESEARCH

Use-After-Free (UAF) Vulnerability CVE-2019-1199 in Microsoft Outlook

RJ MCDOWN / AUGUST 14, 2019 / NO COMMENTS

Use-After-Free (UAF) Vulnerability

Analysis of a Chrome Zero Day: CVE-2019-5786

By Philippe Laulheret on Mar 20, 2019

1. Introduction

On March 1st, Google published an advisory [1] for a use-after-free in the Chrome implementation of the FileReader API (CVE 2019-5786). Clement Lecigne from Google Threat Analysis Group reported the bug as being exploited in the wild and targeting Windows 7, 32-bit platforms. The exploit leads to code execution in the Renderer process, and a second exploit was used to fully compromise the host system [2]. This blog is a technical write-up detailing the first bug and how to find more information about it. At the time of writing, the bug report [2b] is still sealed. Default installation of Chrome will install updates automatically, and users running the latest version of Chrome are already protected against that bug. To make sure you're running the patched version, visit chrome://version, the version number displayed on the page should be 72.0.3626.121 or greater.

nvd.nist.gov/vuln/search/results?form_type=Basic&results_type=overview&query=use+after+free&search_type=all

Information Technology Laboratory

NATIONAL VULNERABILITY DATABASE

VULNERABILITIES | SEARCH AND STATISTICS

Search Results (Refine Search)

Sort results by: Publish Date Descending Sort

Search Parameters:

- Results Type: Overview
- Keyword (text search): use after free
- Search Type: Search All

There are **2,162** matching records.
Displaying matches **1** through **20**.

Vuln ID	Summary	CVSS Severity
CVE-2019-2215	A use-after-free in binder.c allows an elevation of privilege from an application to the Linux Kernel. No user interaction is required to exploit this vulnerability, however exploitation does require either the installation of a malicious local application or a separate vulnerability in a network facing application.Product: AndroidAndroid ID: A-141720095 **Published:** October 11, 2019; 03:15:10 PM -04:00	(not available)
CVE-2019-5527	ESXi, Workstation, Fusion, VMRC and Horizon Client contain a use-after-free vulnerability in the virtual sound device. VMware has evaluated the severity of this issue to be in the Important severity range with a maximum CVSSv3 base score of 8.5. **Published:** October 10, 2019; 01:15:18 PM -04:00	(not available)
CVE-2019-5053	An exploitable use-after-free vulnerability exists in the Length parsing function of NitroPDF. A specially crafted PDF can cause a type confusion, resulting in a use-after-free condition. An attacker can craft a malicious PDF to trigger this vulnerability. **Published:** October 09, 2019; 05:15:13 PM -04:00	*V3.1:* 7.8 HIGH *V2:* 6.8 MEDIUM
CVE-2019-5047	An exploitable Use After Free vulnerability exists in the CharProcs parsing functionality of NitroPDF. A specially crafted PDF can cause a type confusion, resulting in a Use After Free. An attacker can craft a malicious PDF to trigger this vulnerability. **Published:** October 09, 2019; 05:15:13 PM -04:00	*V3.1:* 7.8 HIGH *V2:* 6.8 MEDIUM
CVE-2019-2284	Possible use-after-free issue due to a race condition while calling camera ioctl concurrently in Snapdragon Compute, Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MSM8909W, QCS405, QCS605, Qualcomm 215, SD 425, SD 439 / SD 429, SD 450, SD 625, SD 632, SD 665, SD 675, SD 712 / SD 710 / SD 670, SD 730, SD 845 / SD 850, SD 855, SDM439, SDX24 **Published:** September 30, 2019; 12:15:11 PM -04:00	*V3.1:* 7.0 HIGH *V2:* 4.4 MEDIUM
CVE-2019-10509	Device record of the pairing device used after free during ACL disconnection in Snapdragon Auto, Snapdragon Compute, Snapdragon Connectivity, Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MSM8909W, MSM8996AU, QCA6574AU, QCS405, QCS605, SD 425, SD 427, SD 430, SD 435, SD 439 / SD 429, SD 450, SD 625, SD 632, SD 636, SD 665, SD 675, SD 712 / SD 710 / SD 670, SD 730, SD 820, SD 820A, SD 835, SD 845 / SD 850, SD 855, SDA660, SDM439, SDM630, SDM660, Snapdragon_High_Med_2016 **Published:** September 30, 2019; 12:15:10 PM -04:00	*V3.1:* 9.8 CRITICAL *V2:* 10.0 HIGH
CVE-2019-10501	Possible use after free issue due to improper input validation in volume listener library in Snapdragon Auto, Snapdragon Compute, Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MDM9150, MDM9206, MDM9607, MDM9650, MSM8909W, MSM8996AU, QCS405, QCS605, Qualcomm 215, SD 210/SD 212/SD 205, SD 425, SD 427, SD 430, SD 435, SD 439 / SD 429, SD 450, SD 625, SD 632, SD 636, SD 665, SD 675, SD 712 / SD 710 / SD 670, SD 730, SD 820, SD 820A, SD 835, SD 845 / SD 850, SD 855, SDA660, SDM439, SDM630, SDM660, SDX20, SDX24 **Published:** September 30, 2019; 12:15:10 PM -04:00	*V3.1:* 7.8 HIGH *V2:* 4.6 MEDIUM

Vuln ID	Summary
CVE-2019-2215	A use-after-free in binder.c allows an elevation of privilege from an application to the Linux Kernel. No user interaction is required to exploit this vulnerability, however exploitation does require either the installation of a malicious local application or a separate vulnerability in a network fa application.Product: AndroidAndroid ID: A-141720095 **Published:** October 11, 2019; 03:15:10 PM -04:00
CVE-2019-5527	ESXi, Workstation, Fusion, VMRC and Horizon Client contain a use-after-free vulnerability in the virtual sound device. VMware has evaluated the severity of this issue to be in the Important severity range with a maximum CVSSv3 base score of 8.5. **Published:** October 10, 2019; 01:15:18 PM -04:00
CVE-2019-5053	An exploitable use-after-free vulnerability exists in the Length parsing function of NitroPDF. A specially crafted PDF can cause a type confusion, resulting in a use-after-free condition. An attacker can craft a malicious PDF to trigger this vulnerability. **Published:** October 09, 2019; 05:15:13 PM -04:00
CVE-2019-5047	An exploitable Use After Free vulnerability exists in the CharProcs parsing functionality of NitroPDF. A specially crafted PDF can cause a type confusion, resulting in a Use After Free. An attacker can craft a malicious PDF to trigger this vulnerability. **Published:** October 09, 2019; 05:15:13 PM -04:00
CVE-2019-2284	Possible use-after-free issue due to a race condition while calling camera ioctl concurrently in Snapdragon Compute, Snapdragon Consumer IOT Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MSM8909W, QCS405, QCS605, Qualcomm SD 425, SD 439 / SD 429, SD 450, SD 625, SD 632, SD 665, SD 675, SD 712 / SD 710 / SD 670, SD 730, SD 845 / SD 850, SD 855, SDM439, SDX24 **Published:** September 30, 2019; 12:15:11 PM -04:00
CVE-2019-10509	Device record of the pairing device used after free during ACL disconnection in Snapdragon Auto, Snapdragon Compute, Snapdragon Connectiv Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MSM8909W, MSM8996AU, QCA6574AU, QCS405, QCS605, SD 425, SD 427, SD 430, SD 435, SD 439 / SD 429, SD 450, SD 625, SD 632, SD 636, SD 665, SD 675, SD SD 710 / SD 670, SD 730, SD 820, SD 820A, SD 835, SD 845 / SD 850, SD 855, SDA660, SDM439, SDM630, SDM660, Snapdragon_High_Med_2016 **Published:** September 30, 2019; 12:15:10 PM -04:00
CVE-2019-10501	Possible use after free issue due to improper input validation in volume listener library in Snapdragon Auto, Snapdragon Compute, Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MDM9150, MDM9206, MDM9607, MDM9650, MSM8909W, MSM8996AU, QCS405, QCS605, Qualcomm 215, SD 210/SD 212/SD 205, SD 425, SD 427, SD 430, SD 435, SD 43 429, SD 450, SD 625, SD 632, SD 636, SD 665, SD 675, SD 712 / SD 710 / SD 670, SD 730, SD 820, SD 820A, SD 835, SD 845 / SD 850, SD 855, SDA660 SDM439, SDM630, SDM660, SDX20, SDX24 **Published:** September 30, 2019; 12:15:10 PM -04:00

Vuln ID	Summary
CVE-2019-2215	A use-after-free in binder.c allows an elevation of privilege from an application to the Linux Kernel. No user interaction is required to exploit this vulnerability, however exploitation does require either the installation of a malicious local application or a separate vulnerability in a network fa application.Product: AndroidAndroid ID: A-141720095 **Published:** October 11, 2019; 03:15:10 PM -04:00
CVE-2019-5527	ESXi, Workstation, Fusion, VMRC and Horizon Client contain a use-after-free vulnerability in the virtual sound device. VMware has evaluated the severity of this issue to be in the Important severity range with a maximum CVSSv3 base score of 8.5. **Published:** October 10, 2019; 01:15:18 PM -04:00
CVE-2019-5053	An exploitable use-after-free vulnerability exists in the Length parsing function of NitroPDF. A specially crafted PDF can cause a type confusion, resulting in a use-after-free condition. An attacker can craft a malicious PDF to trigger this vulnerability. **Published:** October 09, 2019; 05:15:13 PM -04:00
CVE-2019-5047	An exploitable Use After Free vulnerability exists in the CharProcs parsing functionality of NitroPDF. A specially crafted PDF can cause a type confusion, resulting in a Use After Free. An attacker can craft a malicious PDF to trigger this vulnerability. **Published:** October 09, 2019; 05:15:13 PM -04:00
CVE-2019-2284	Possible use-after-free issue due to a race condition while calling camera ioctl concurrently in Snapdragon Compute, Snapdragon Consumer IOT Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MSM8909W, QCS405, QCS605, Qualcomm SD 425, SD 439 / SD 429, SD 450, SD 625, SD 632, SD 665, SD 675, SD 712 / SD 710 / SD 670, SD 730, SD 845 / SD 850, SD 855, SDM439, SDX24 **Published:** September 30, 2019; 12:15:11 PM -04:00
CVE-2019-10509	Device record of the pairing device used after free during ACL disconnection in Snapdragon Auto, Snapdragon Compute, Snapdragon Connectiv Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MSM8909W, MSM8996AU, QCA6574AU, QCS405, QCS605, SD 425, SD 427, SD 430, SD 435, SD 439 / SD 429, SD 450, SD 625, SD 632, SD 636, SD 665, SD 675, SD SD 710 / SD 670, SD 730, SD 820, SD 820A, SD 835, SD 845 / SD 850, SD 855, SDA660, SDM439, SDM630, SDM660, Snapdragon_High_Med_2016 **Published:** September 30, 2019; 12:15:10 PM -04:00
CVE-2019-10501	Possible use after free issue due to improper input validation in volume listener library in Snapdragon Auto, Snapdragon Compute, Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables in MDM9150, MDM9206, MDM9607, MDM9650, MSM8909W, MSM8996AU, QCS405, QCS605, Qualcomm 215, SD 210/SD 212/SD 205, SD 425, SD 427, SD 430, SD 435, SD 43 429, SD 450, SD 625, SD 632, SD 636, SD 665, SD 675, SD 712 / SD 710 / SD 670, SD 730, SD 820, SD 820A, SD 835, SD 845 / SD 850, SD 855, SDA660 SDM439, SDM630, SDM660, SDX20, SDX24 **Published:** September 30, 2019; 12:15:10 PM -04:00

Outline

- ❖ **Existing Solutions to UAF**
- ❖ **Checked C Solution to UAF**
- ❖ **Demo (if time permits)**

Outline

- ❖ **Existing Solutions to UAF**
- ❖ **Checked C Solution to UAF**
- ❖ **Demo (if time permits)**

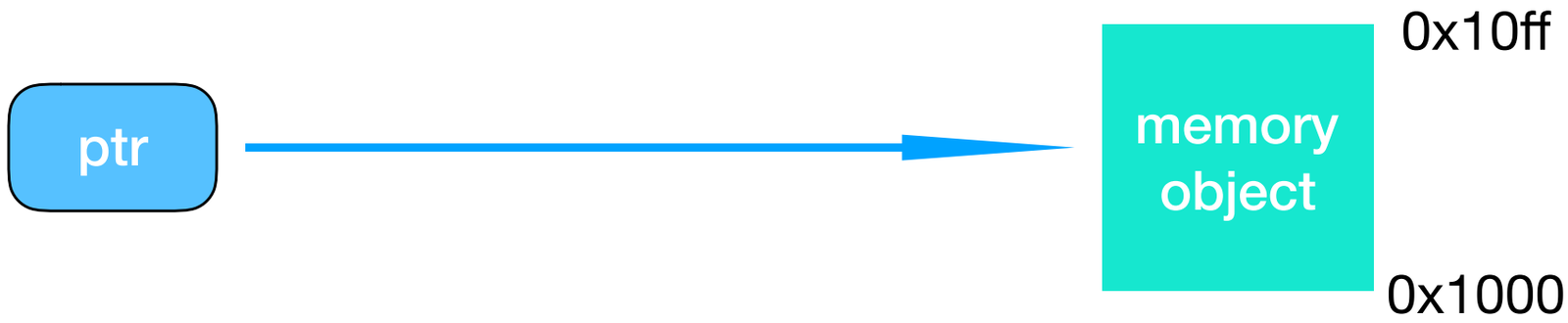
Existing Solutions to UAF

- Safe memory allocators
- Invalidating dangling pointers
- Dynamic checking on dereference

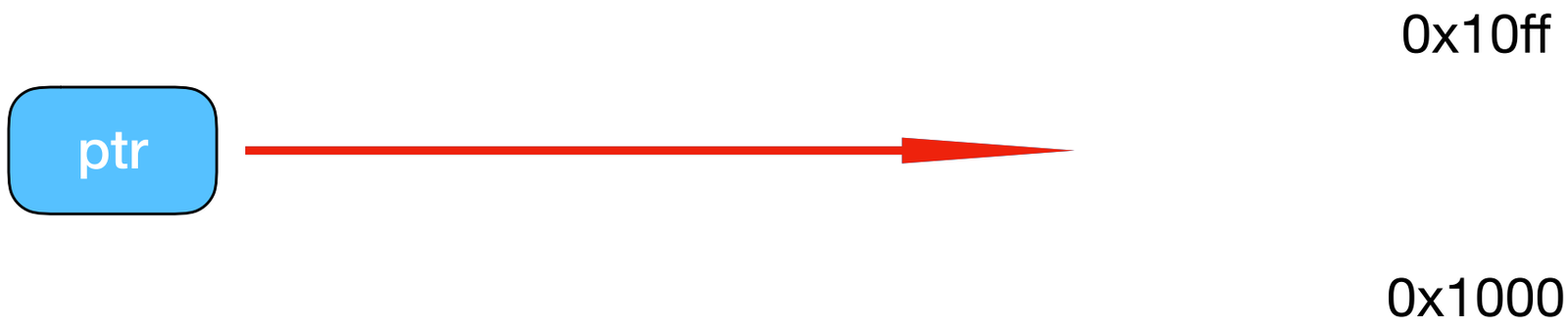
Existing Solutions to UAF

- Safe memory allocators
- Invalidating dangling pointers
- Dynamic checking on dereference

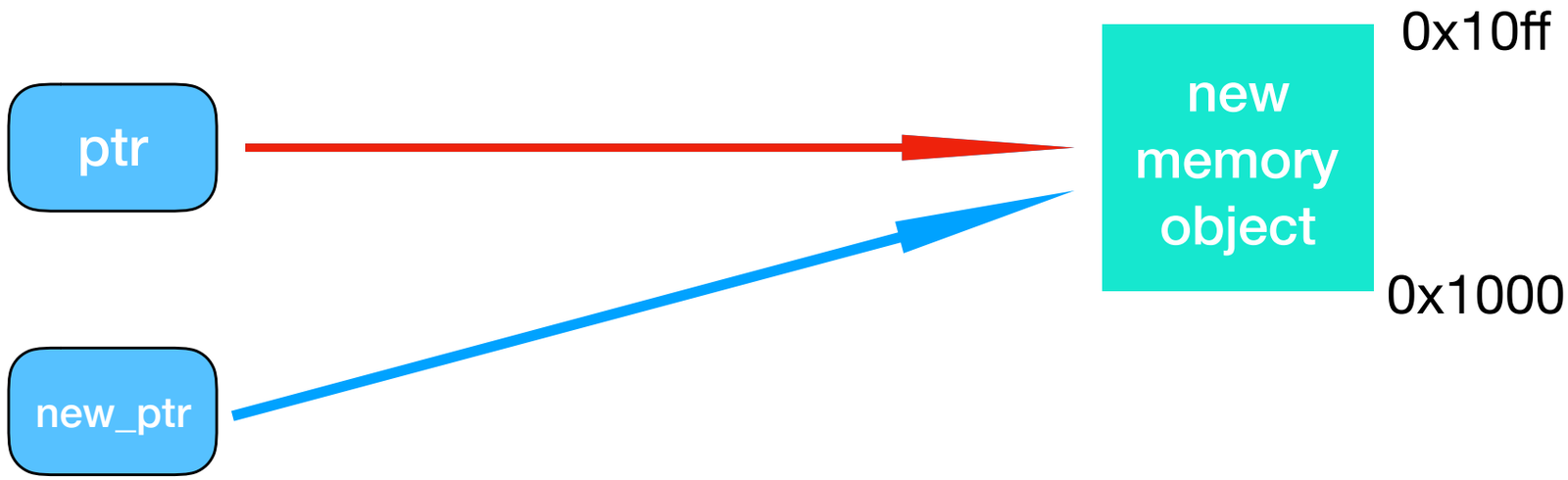
Safe Memory Allocator



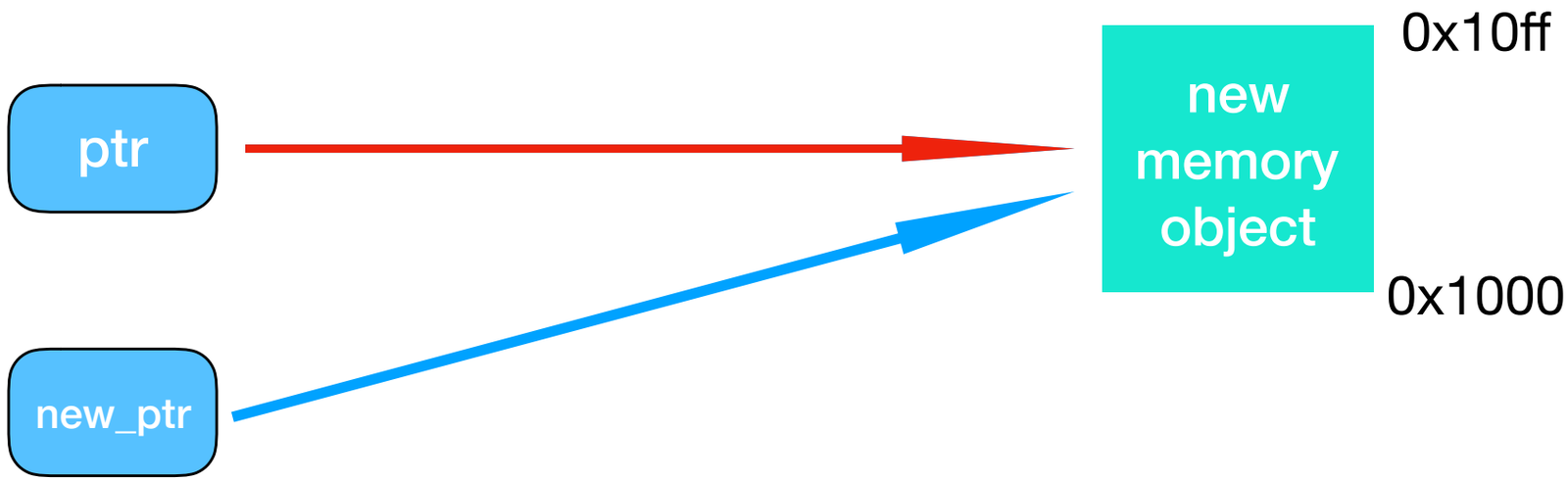
Safe Memory Allocator



Safe Memory Allocator



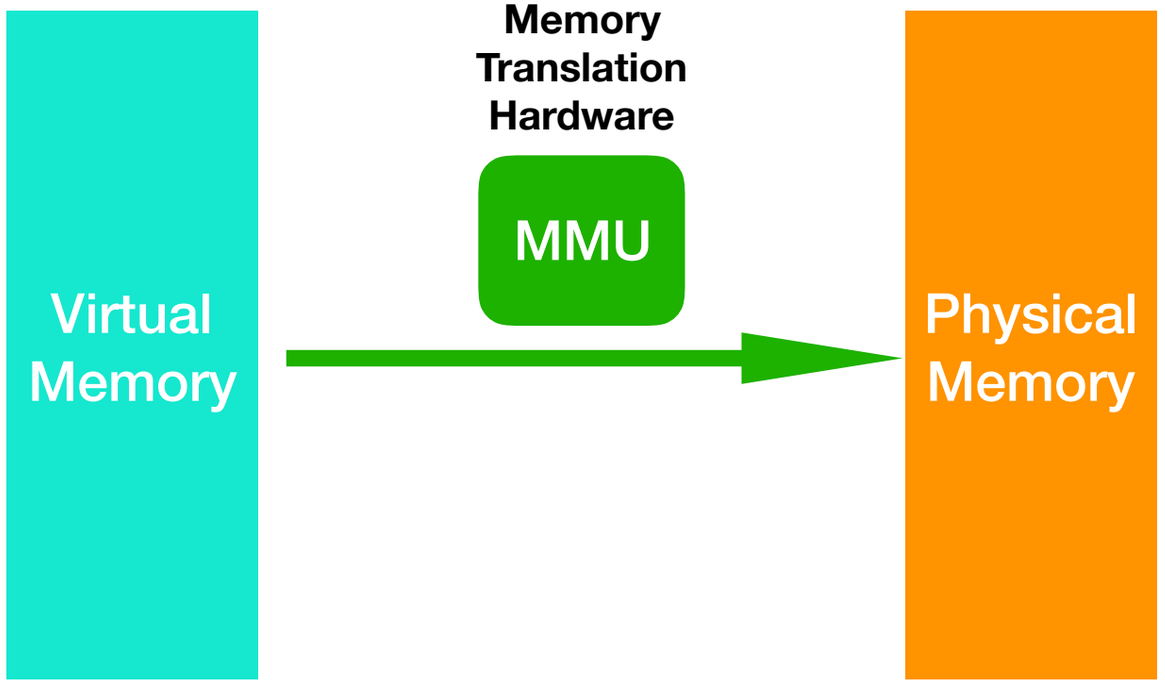
Safe Memory Allocator



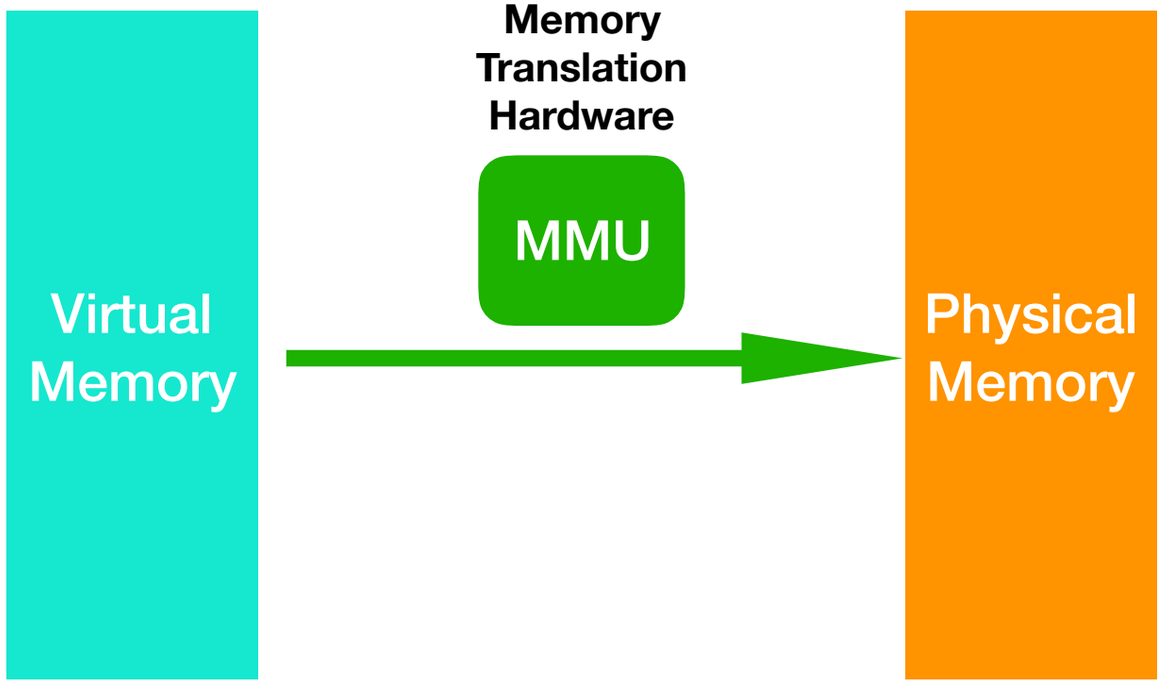
Direction: Forbid / Minimize reuse of freed memory regions

- forbid/minimize reuse of virtual page (**D & V DSN'16**, **Oscar**)
- allocate memory region by data Type (**Cling**)
- randomize locations of allocated memory objects (**DieHard**)

Minimize Reuse of Virtual Memory Page



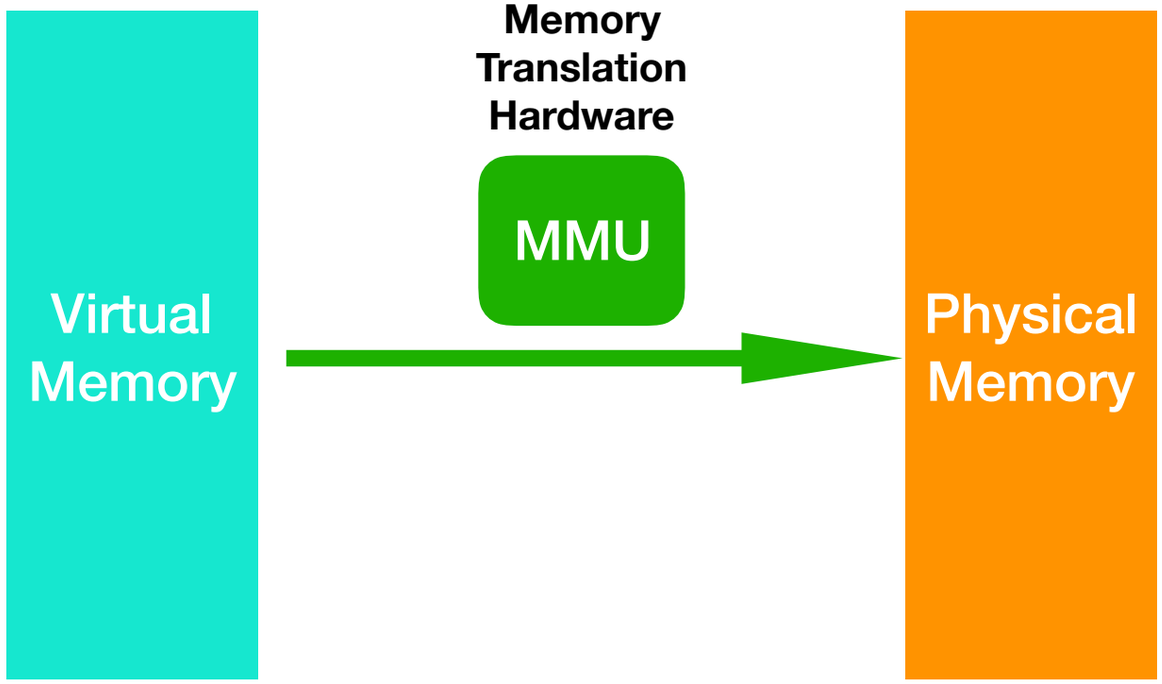
Minimize Reuse of Virtual Memory Page



Basic idea:

- Put each newly allocated memory object on a new virtual page
- Set the page access permission bit to be invalid after free operation
- Rely on MMU to do the checking for access after free

Minimize Reuse of Virtual Memory Page



Basic idea:

- Put each newly allocated memory object on a new virtual page
- Set the page access permission bit to be invalid after free operation
- Rely on MMU to do the checking for access after free

Limitations:

- **Electric Fence** (1994), **PageHeap** (2000): impractical memory consumption
- **Dhurjati & Adve** [DSN'06]: perform bad (up to 11x overhead) on allocation-intensive programs
- **Oscar** [USENIX Security'17]: memory overhead 62% on average; up to > 400%

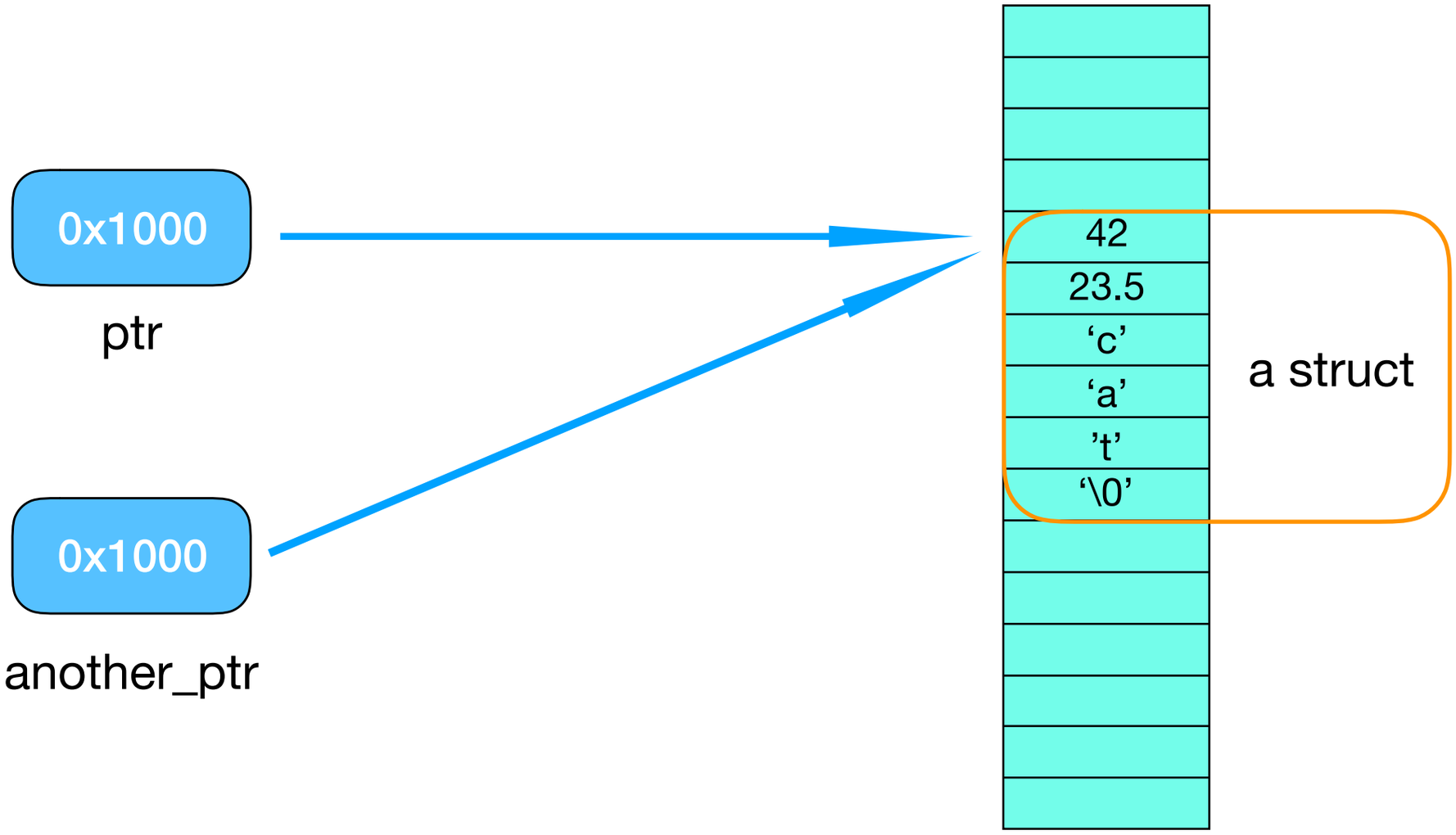
Existing Solutions to UAF

- Safe memory allocators

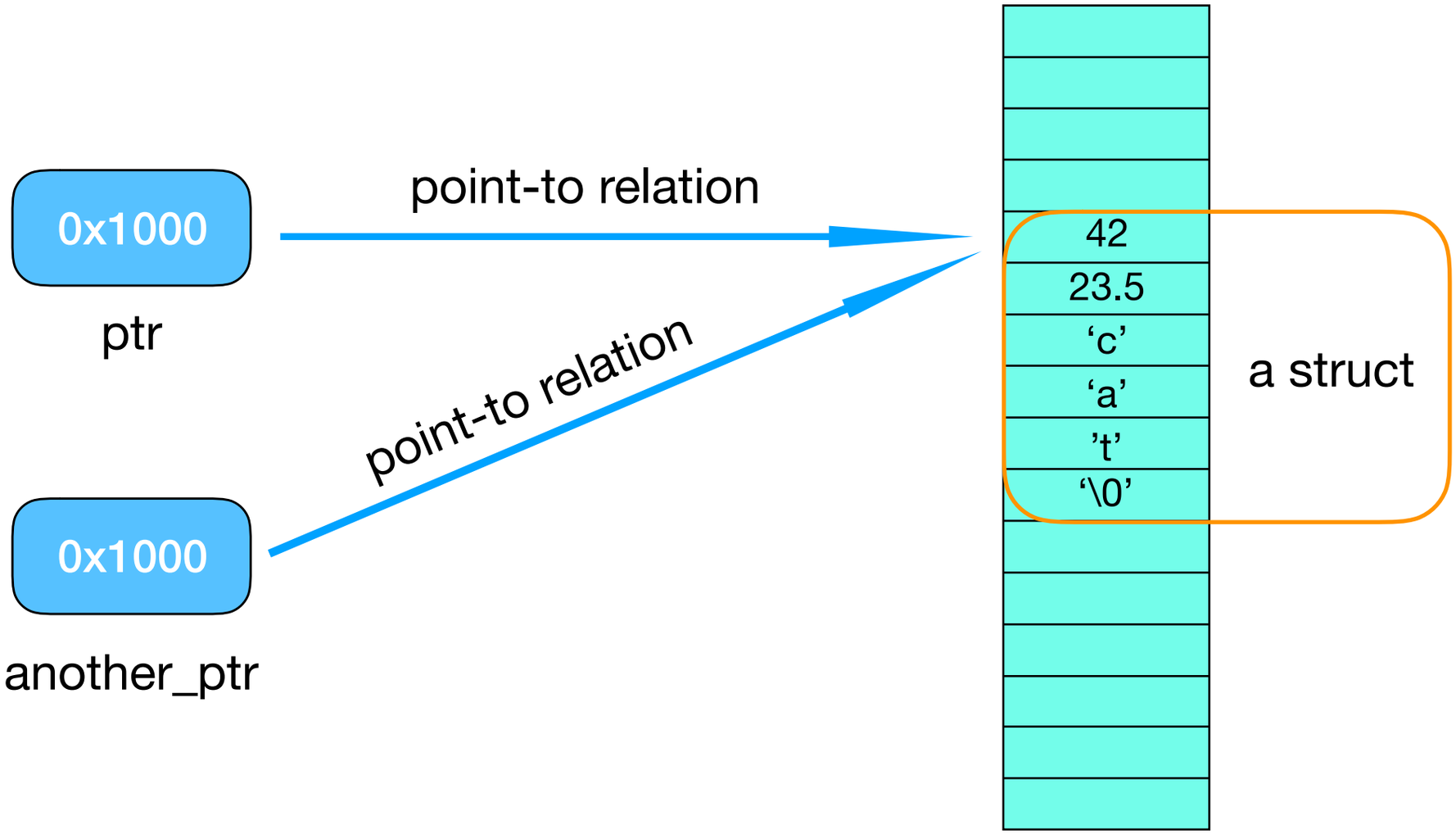
- Invalidating dangling pointers

- Dynamic checking on dereference

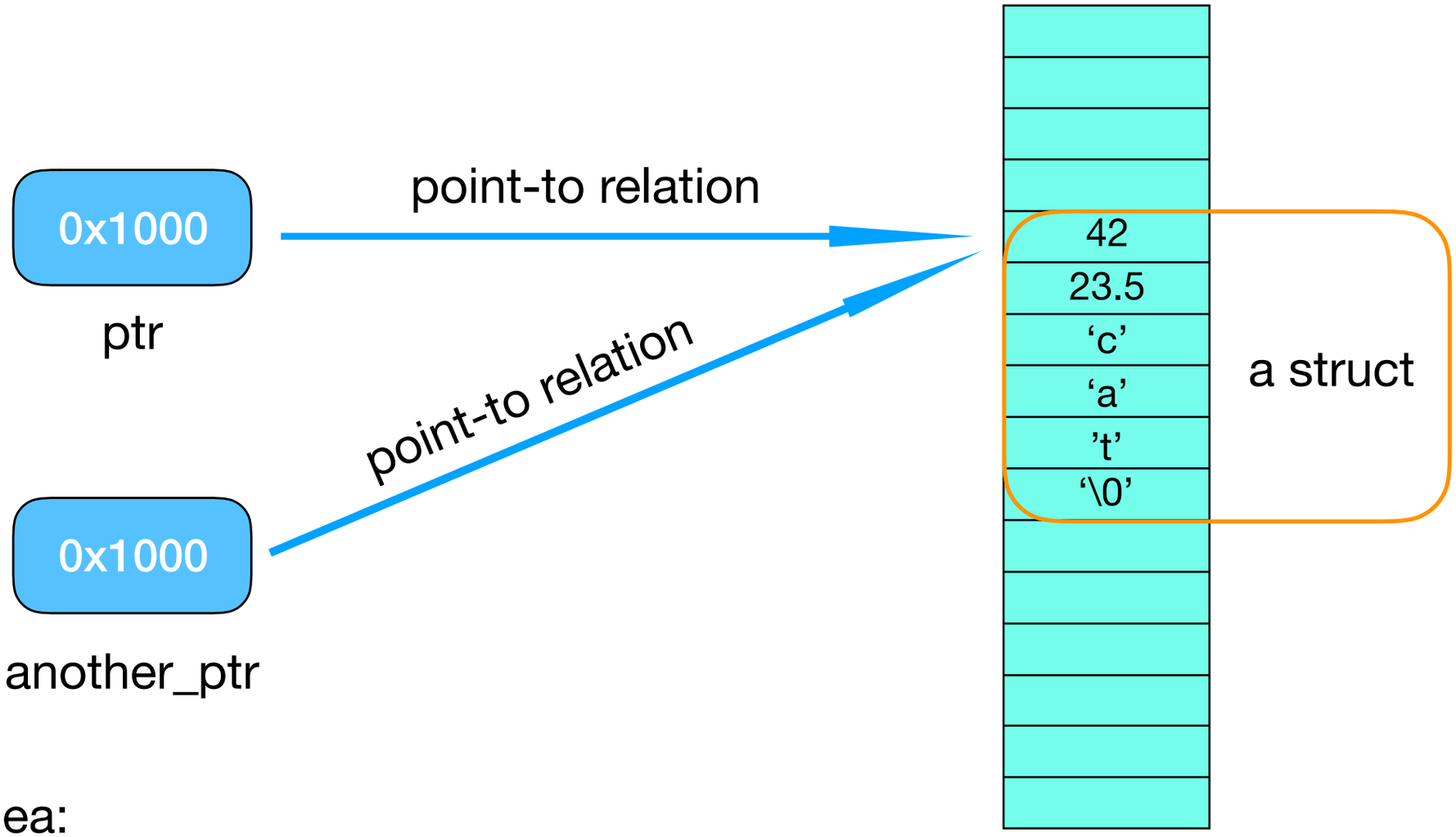
Invalidating Pointers After Deallocation



Invalidating Pointers After Deallocation



Invalidating Pointers After Deallocation



Basic idea:

- keep track of the point-to relations
- invalidate pointers upon memory deallocation

Examples: **DANGNULL** [NDSS’15], **FreeSentry** [NDSS’15], **DangSan** [EuroSys’17]

Invalidating Pointers After Deallocation

Solutions	Performance Overhead	Memory Overhead
DANGNULL	Average: 80% Maximum: > 400%	Average: 127% Maximum: 1,700%
FreeSentry	Average: 42% Maximum: > 460%	Not reported in the paper
DangSan	Average: 41% Maximum: 772%	Average: 240% Maximum: 13,465%

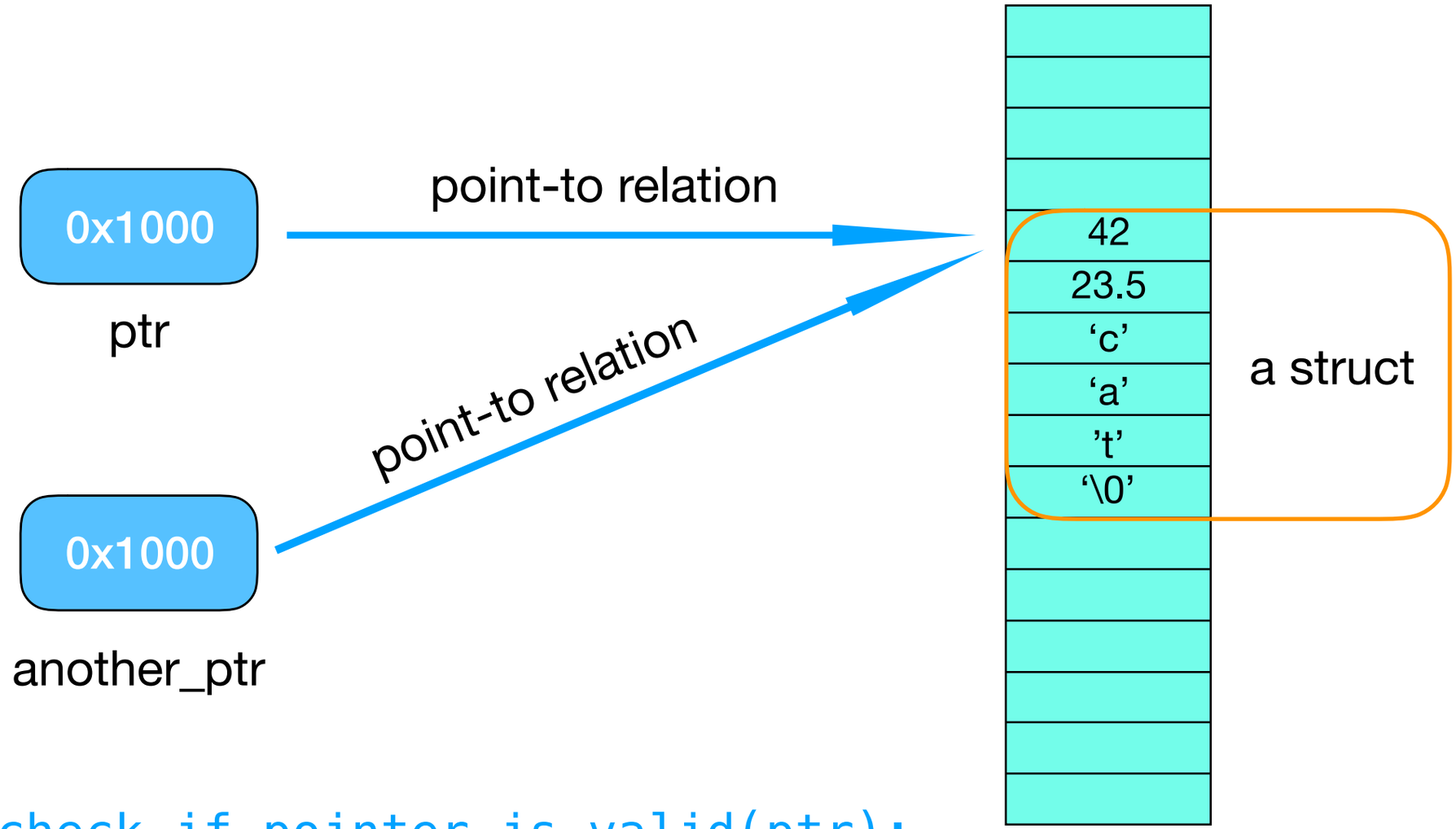
Overhead summary of three recent works

Other problems: false positive, false negative, lack of multithreading support

Existing Solutions to UAF

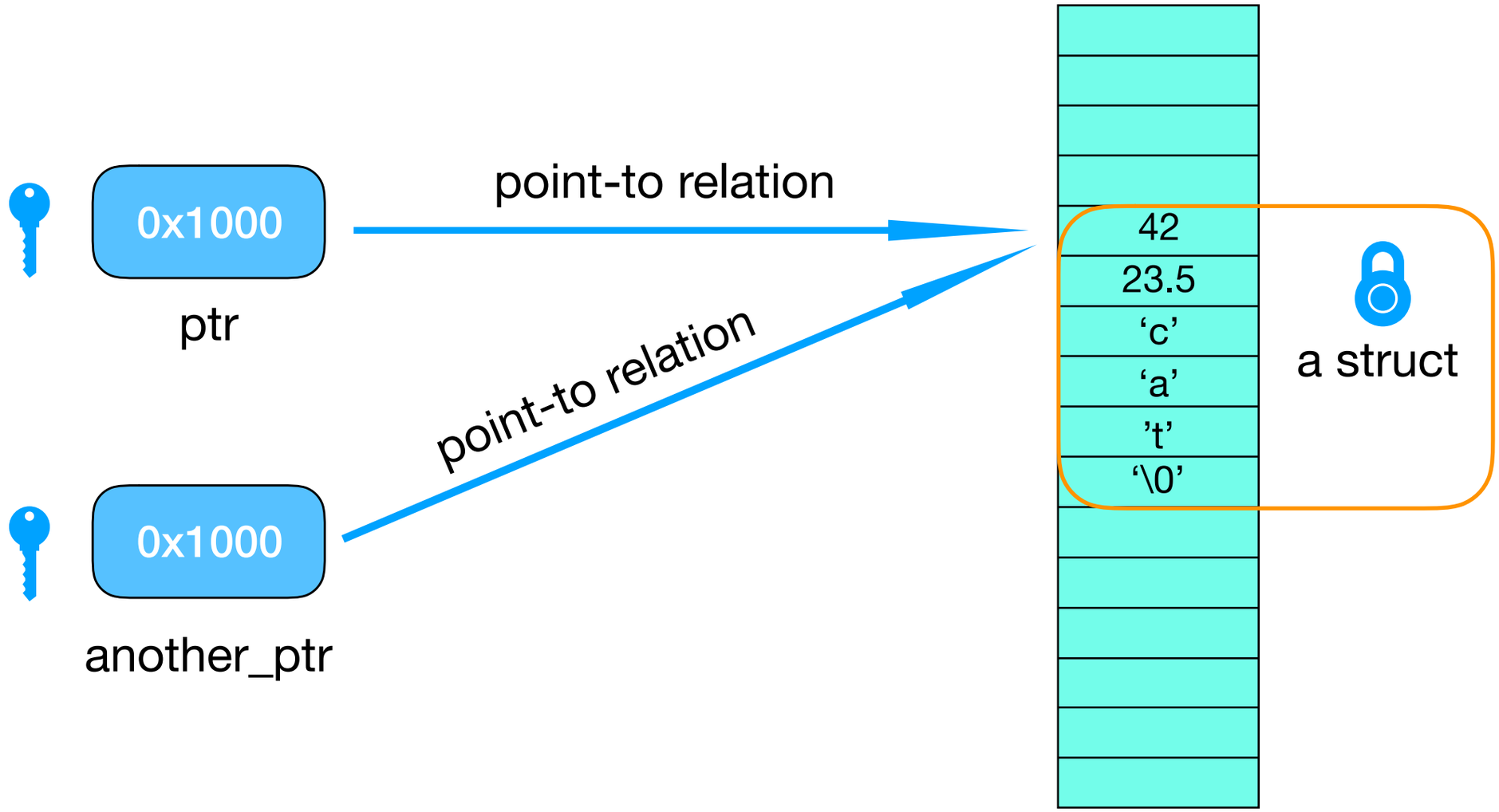
- Safe memory allocators
- Invalidating dangling pointers
- Dynamic checking on dereference

Dynamic Checking on Pointer Dereference

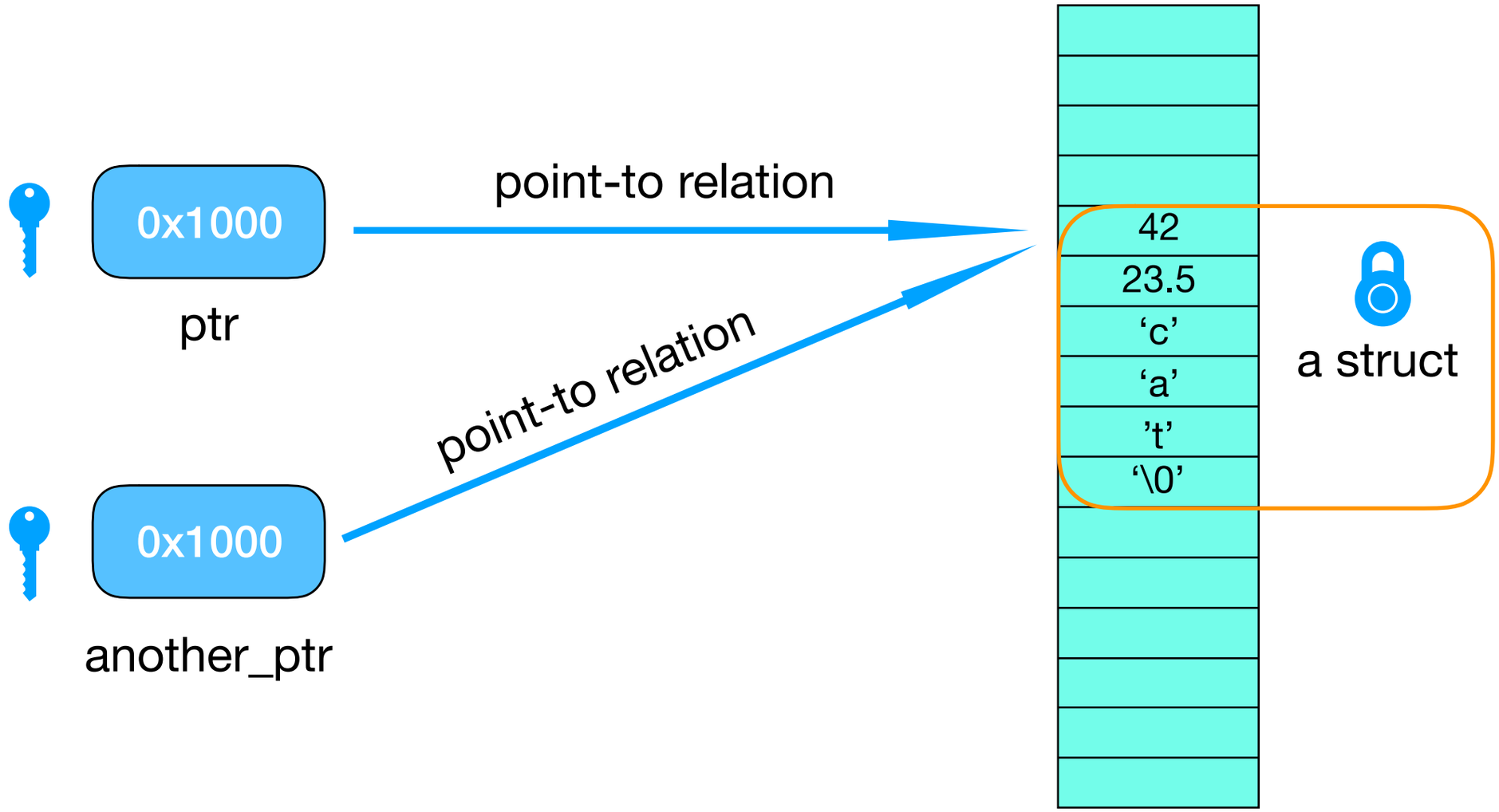



```
check_if_pointer_is_valid(ptr);
ptr->num = 30;
```

Key-lock Based Dynamic Checking



Key-lock Based Dynamic Checking



Basic idea:

- associate each pointer with a key and each memory object with a lock
- invalidate the lock when a memory object is freed
- check if the key matches the lock when a pointer is dereferenced

Key-lock Based Dynamic Checking

[ISMM'10] *CETS: Compiler-Enforced Temporal Safety for C*

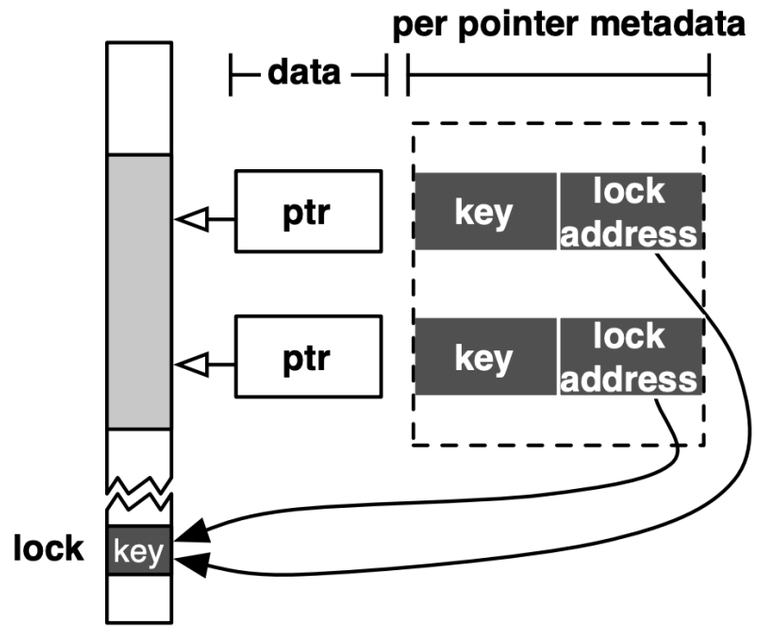


Figure 2. CETS pointer metadata for two pointers in memory. In this example, the pointers point to locations within the same object and thus have the same lock address and have the same key value.

Key-lock Based Dynamic Checking

[ISMM'10] *CETS: Compiler-Enforced Temporal Safety for C*

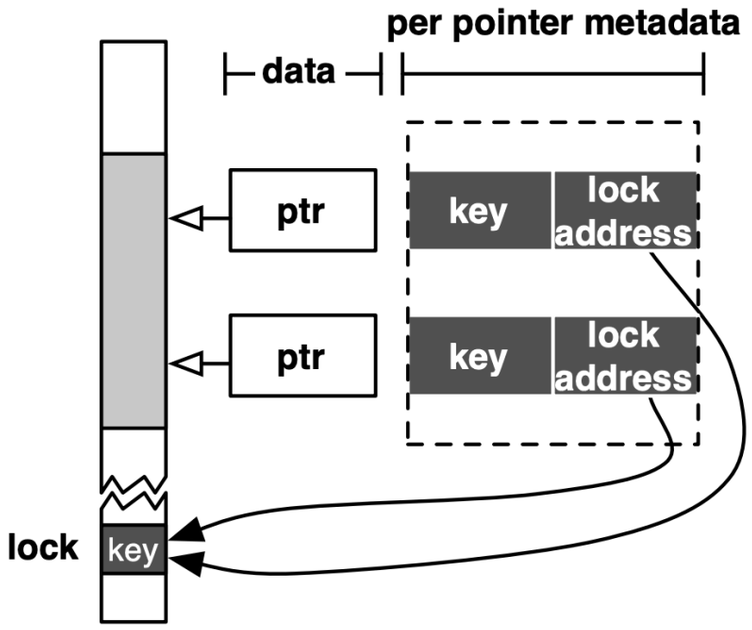


Figure 2. CETS pointer metadata for two pointers in memory. In this example, the pointers point to locations within the same object and thus have the same lock address and have the same key value.

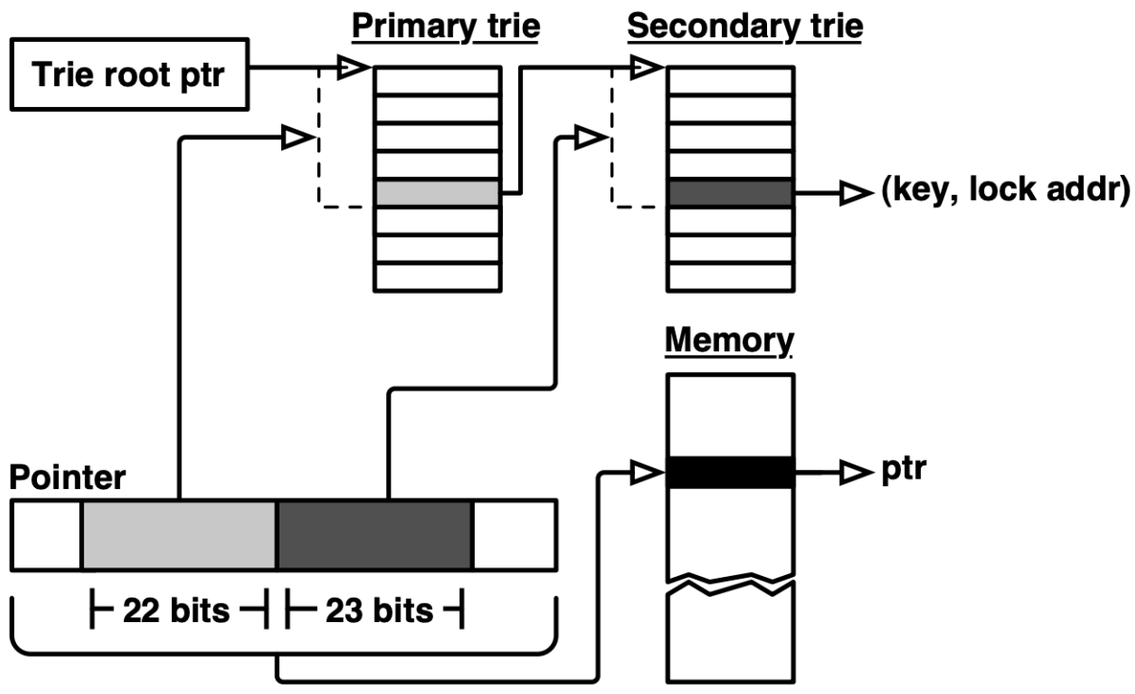


Figure 3. Organization of the trie for pointer metadata

Key-lock Based Dynamic Checking

[ISMM'10] *CETS: Compiler-Enforced Temporal Safety for C*

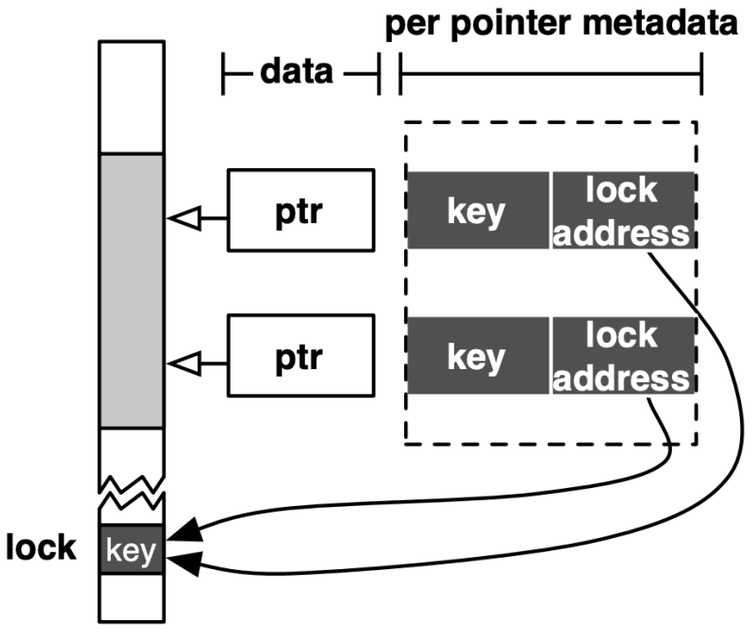


Figure 2. CETS pointer metadata for two pointers in memory. In this example, the pointers point to locations within the same object and thus have the same lock address and have the same key value.

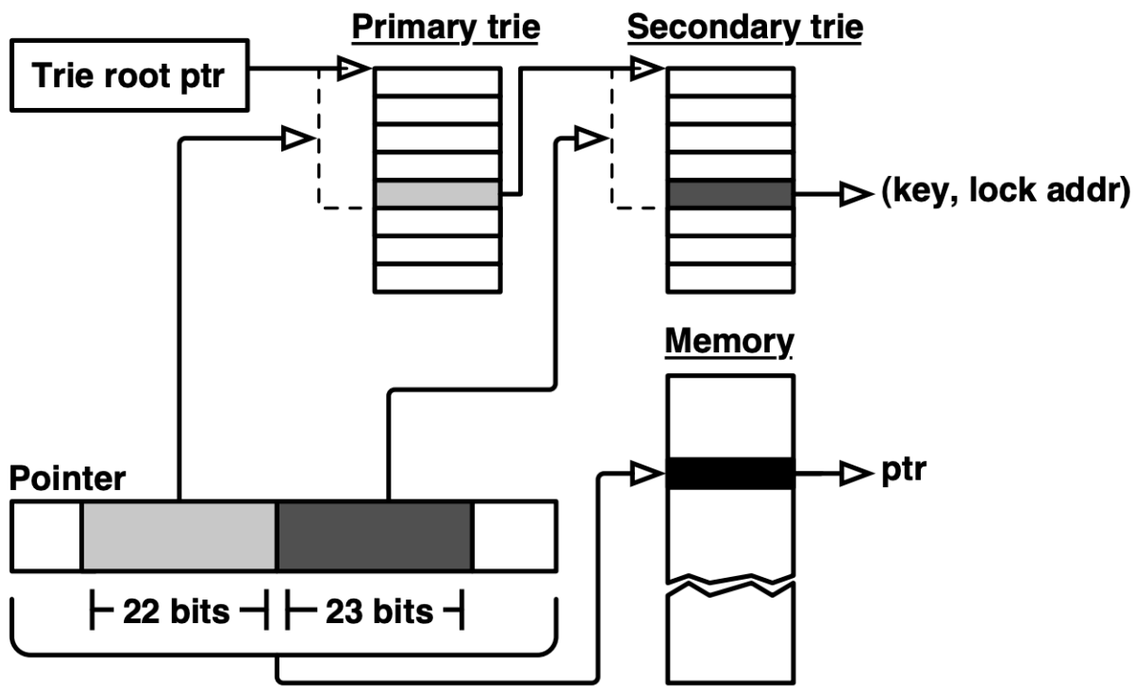


Figure 3. Organization of the trie for pointer metadata

Not cheap:

each trie lookup for a pointer load/store: roughly 11 x86 instructions, including 4 loads

performance overhead: 48% on SPEC 2006

Outline

- ❖ Existing Solutions to UAF
- ❖ **Checked C Solution to UAF**
- ❖ Demo (if time permits)

Checked C Solution to UAF

Observation: using *disjoint data structures* to keep track of the point-to relations (metadata) is slow.

Checked C Solution to UAF

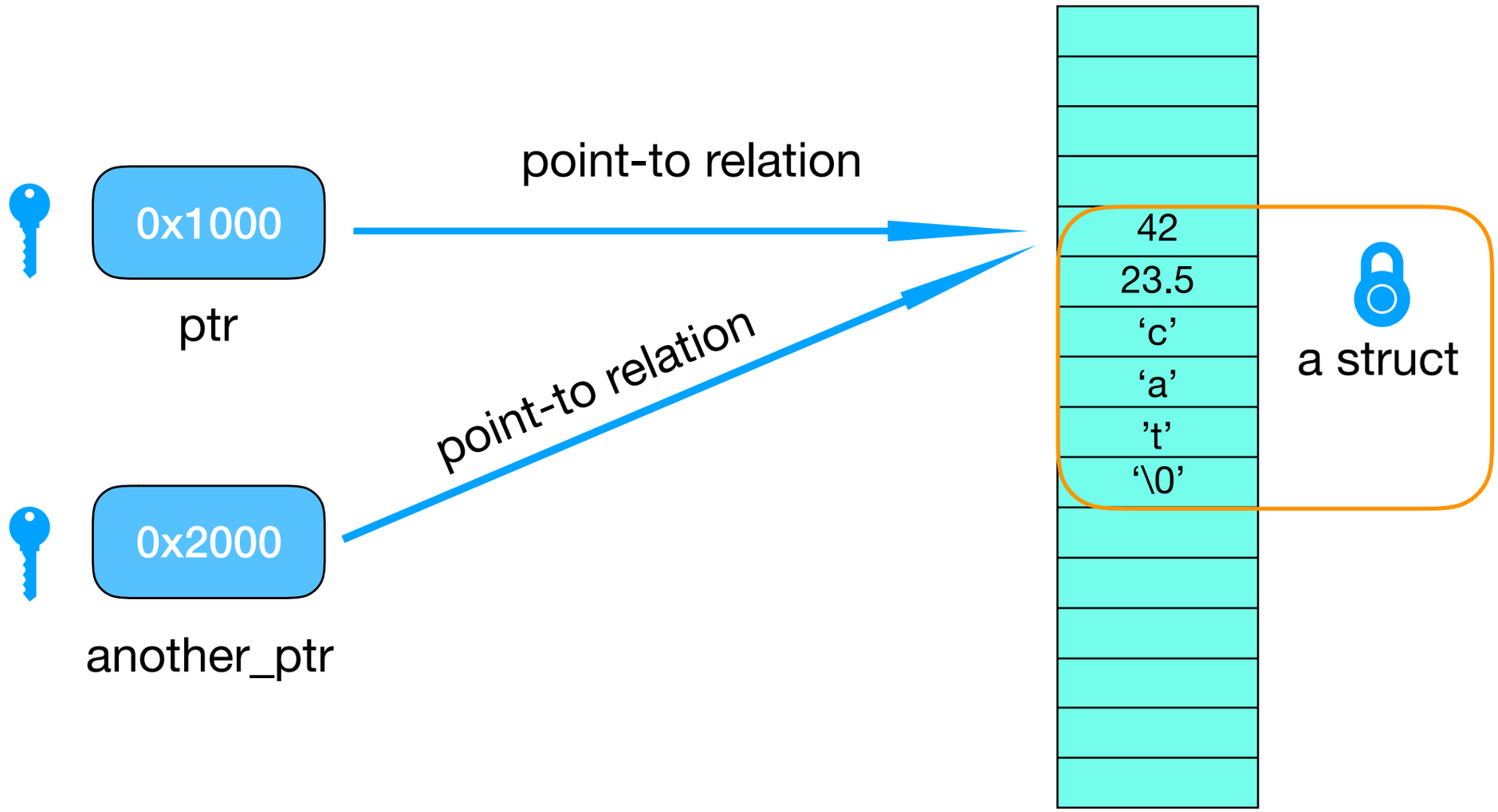
Observation: using *disjoint data structures* to keep track of the point-to relations (metadata) is slow.

Make the metadata travel with pointers and memory objects

C. N. Fischer and R. J. LeBlanc. The Implementation of Run-Time Diagnostics in Pascal. *IEEE Transactions on Software Engineering*, SE-6(4):313–319, July 1980.

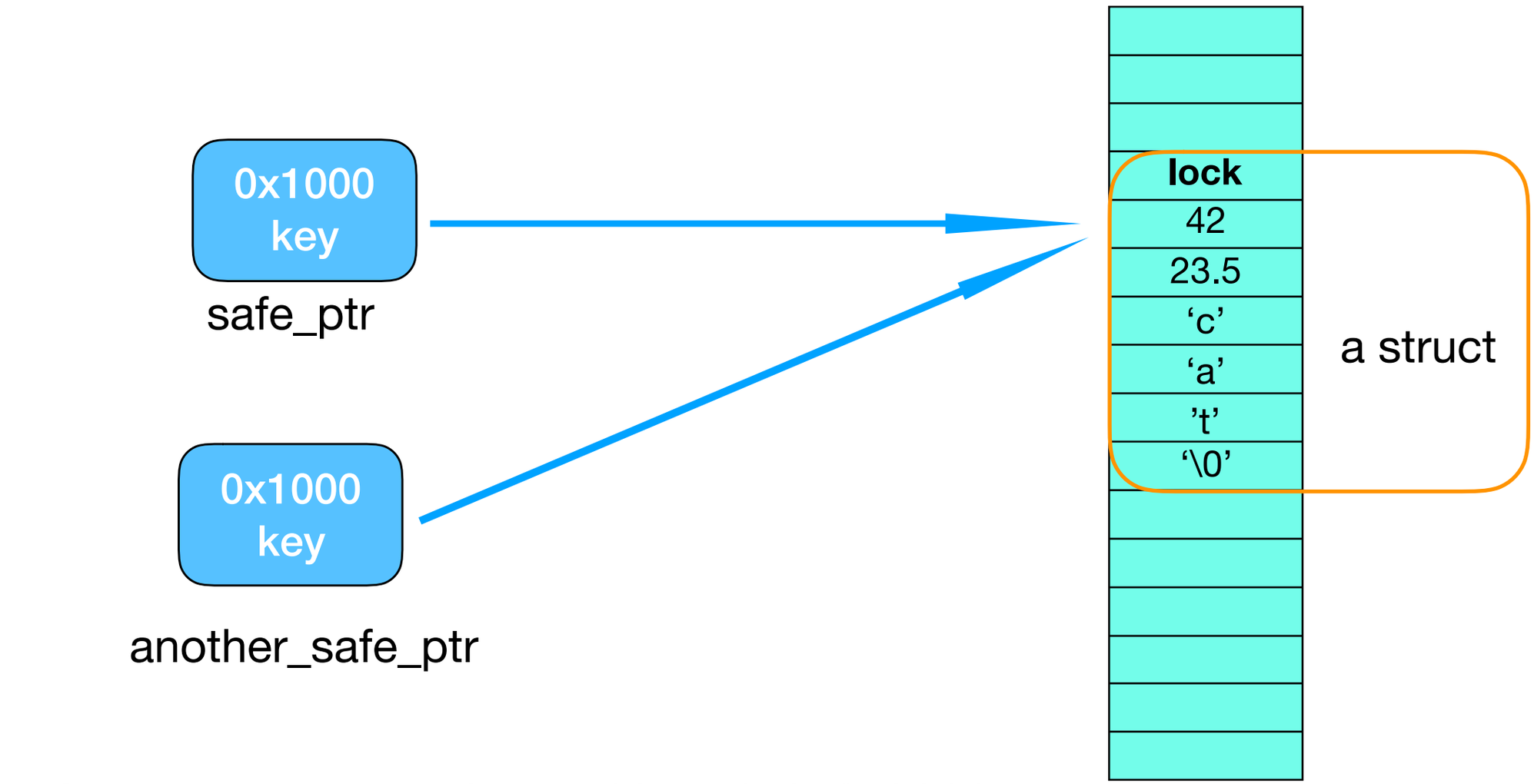
New Types of Pointers

New safe pointers : dereference is free from UAF errors



New Types of Pointers

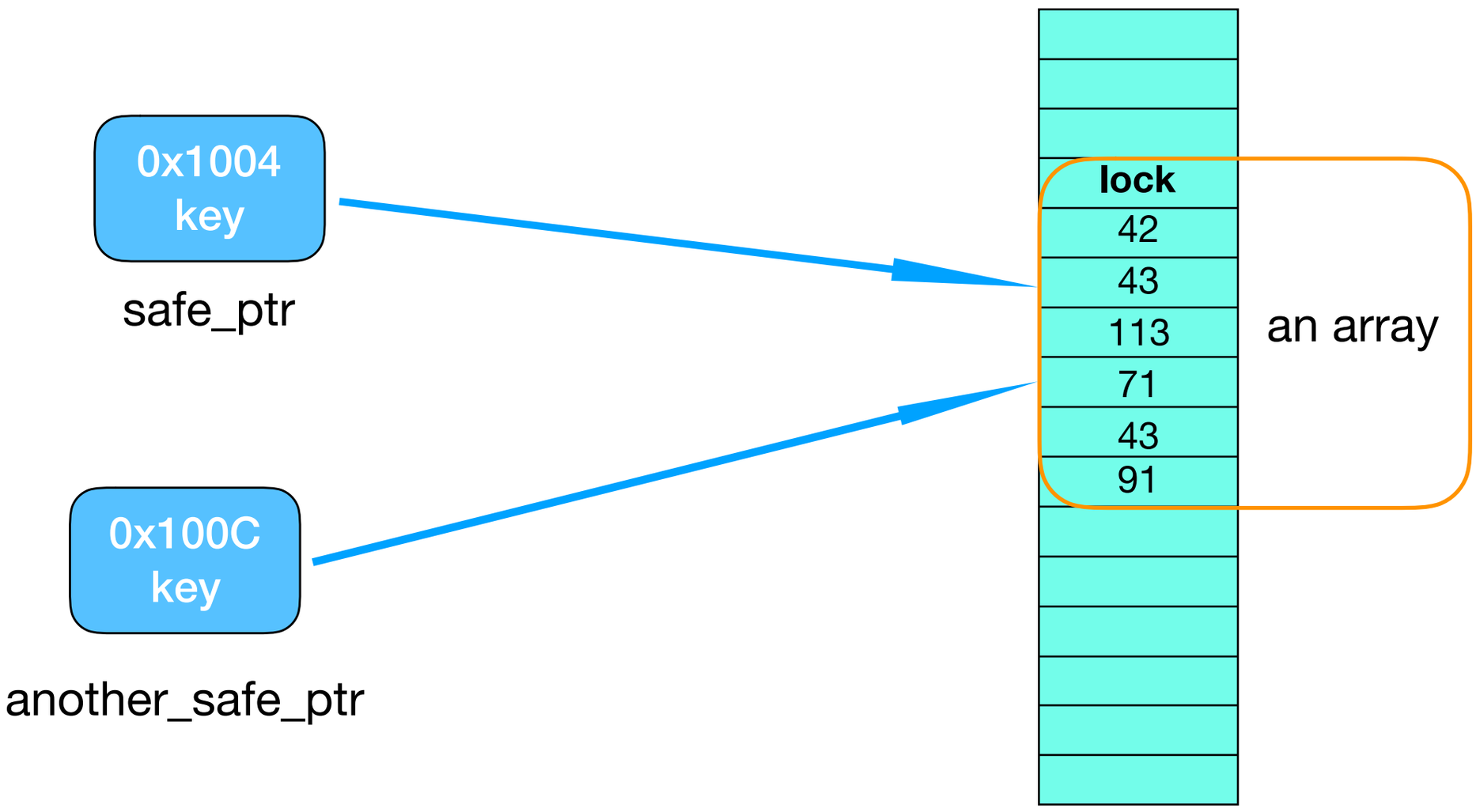
New safe pointers : dereference is free from UAF errors



pointers to struct: pointer arithmetic is forbidden

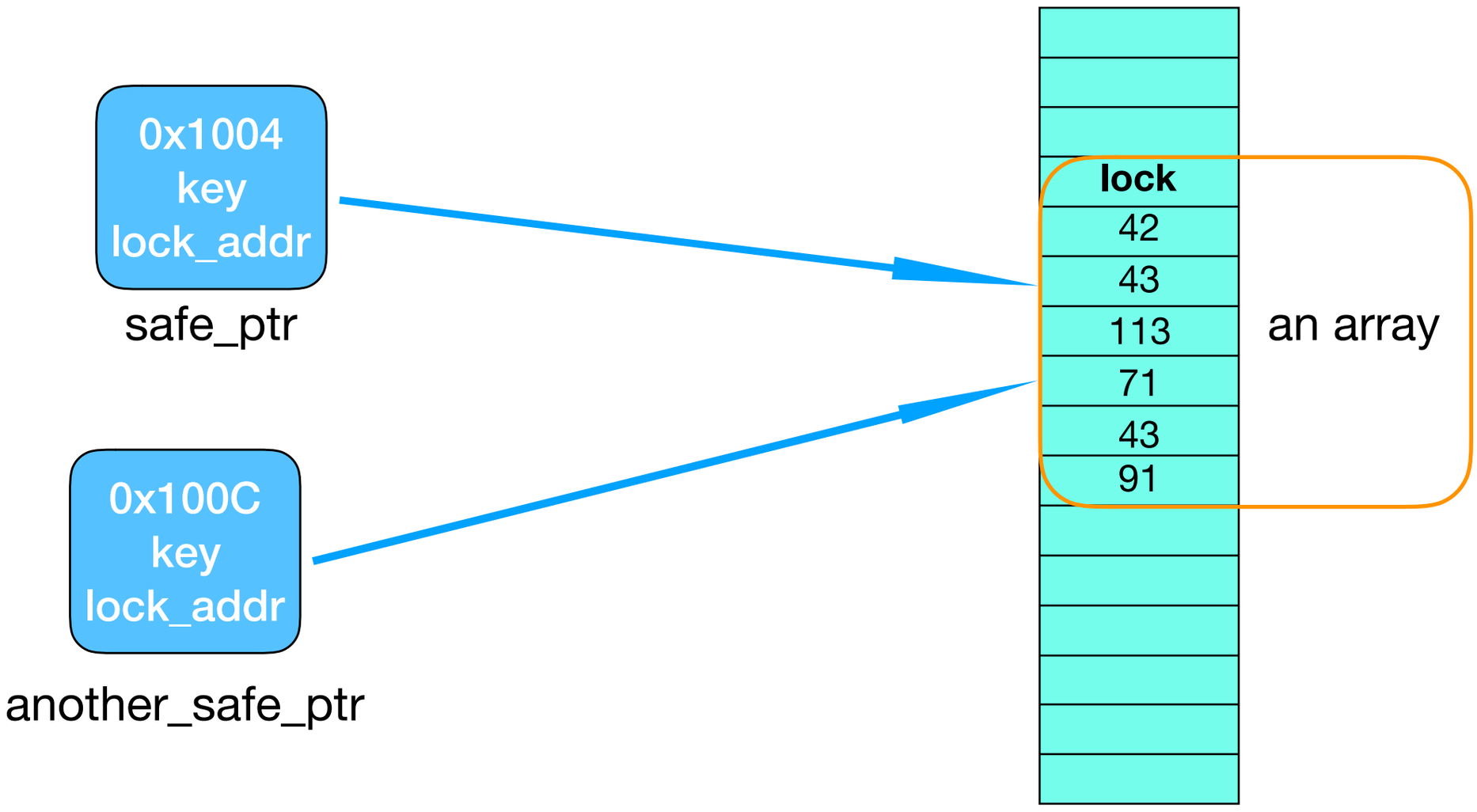
Safe Array Pointers

New safe pointers : dereference is free from UAF errors



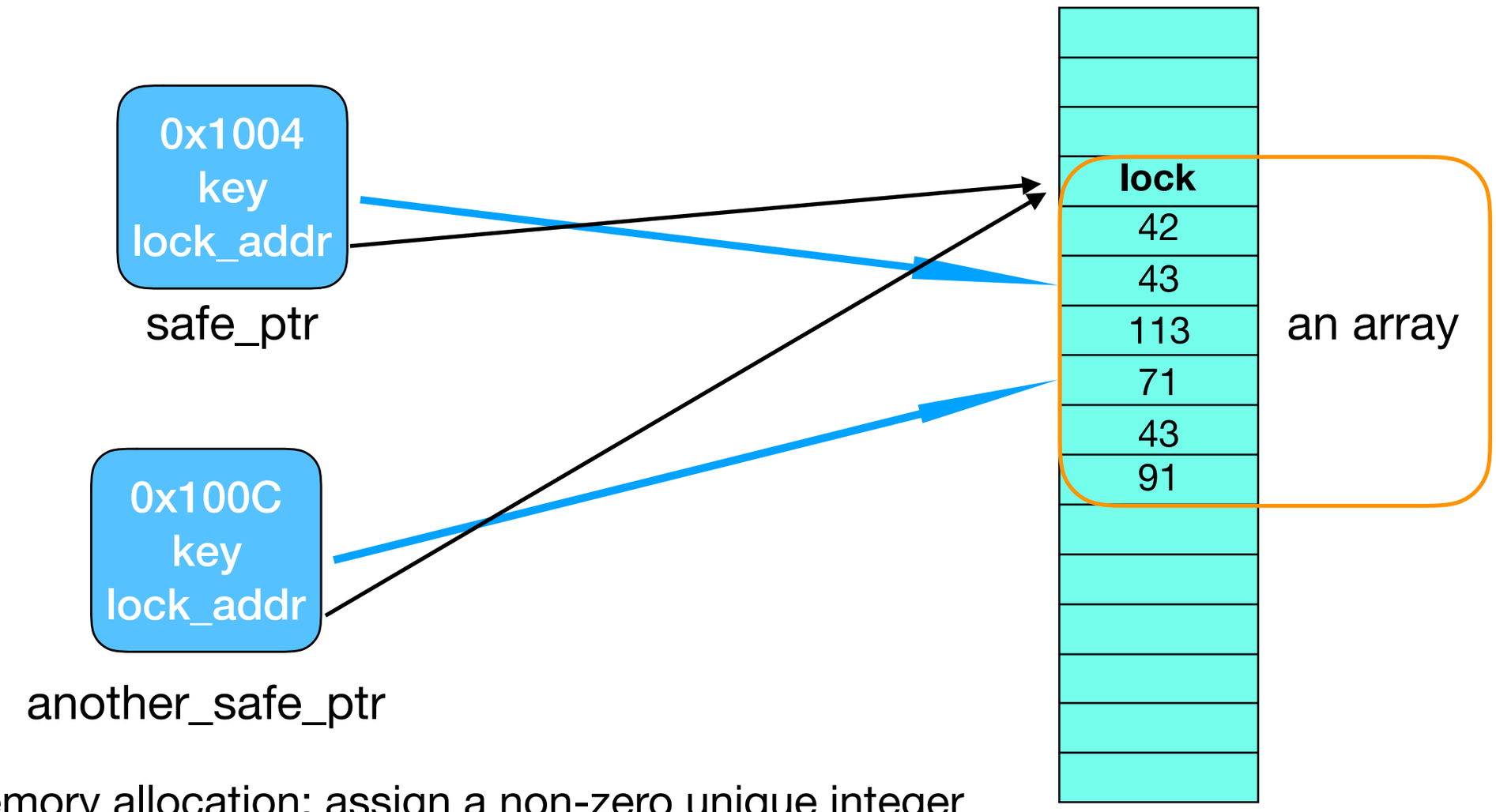
Safe Array Pointers

New safe pointers : dereference is free from UAF errors



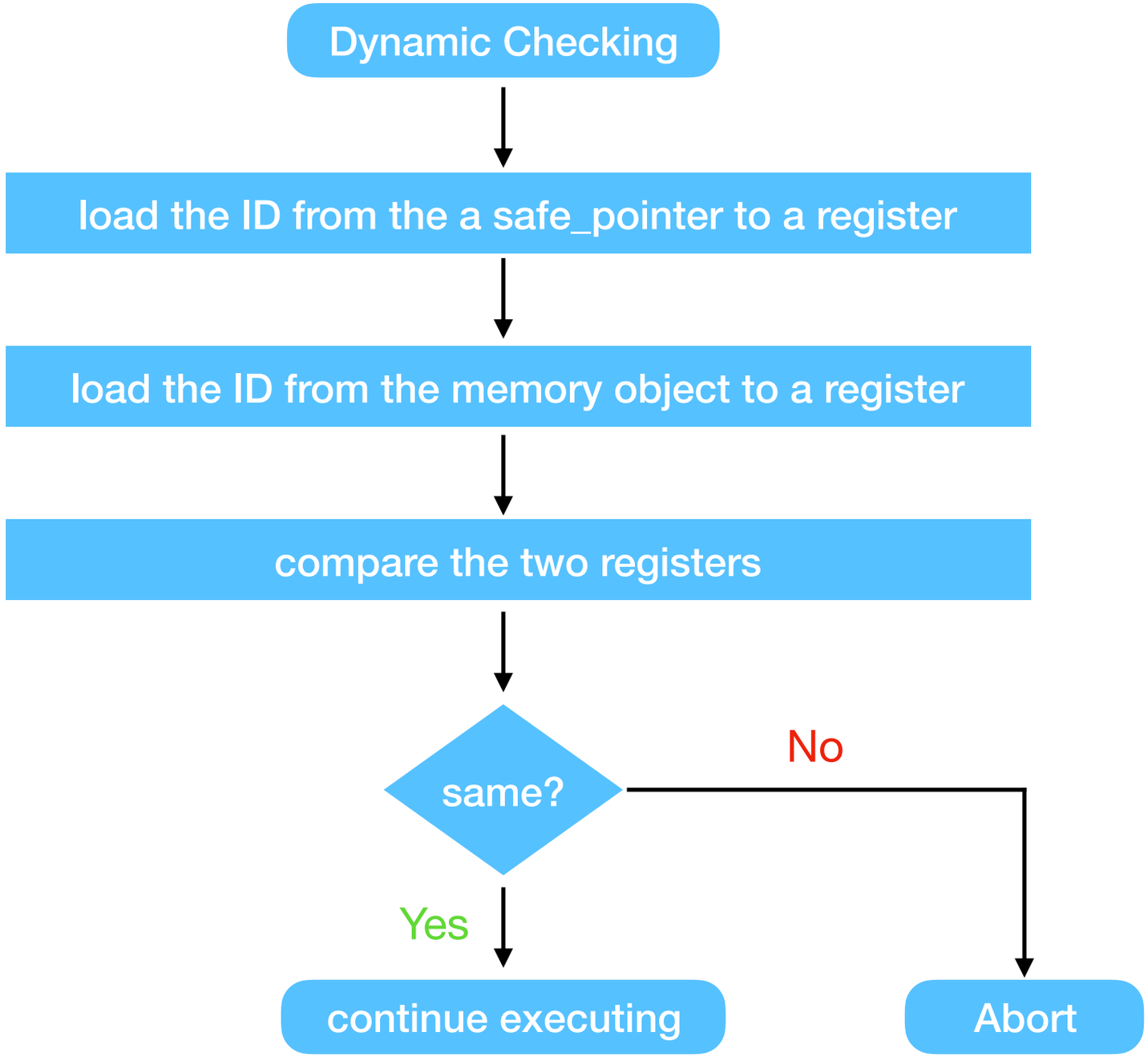
Safe Array Pointers

New safe pointers : dereference is free from UAF errors

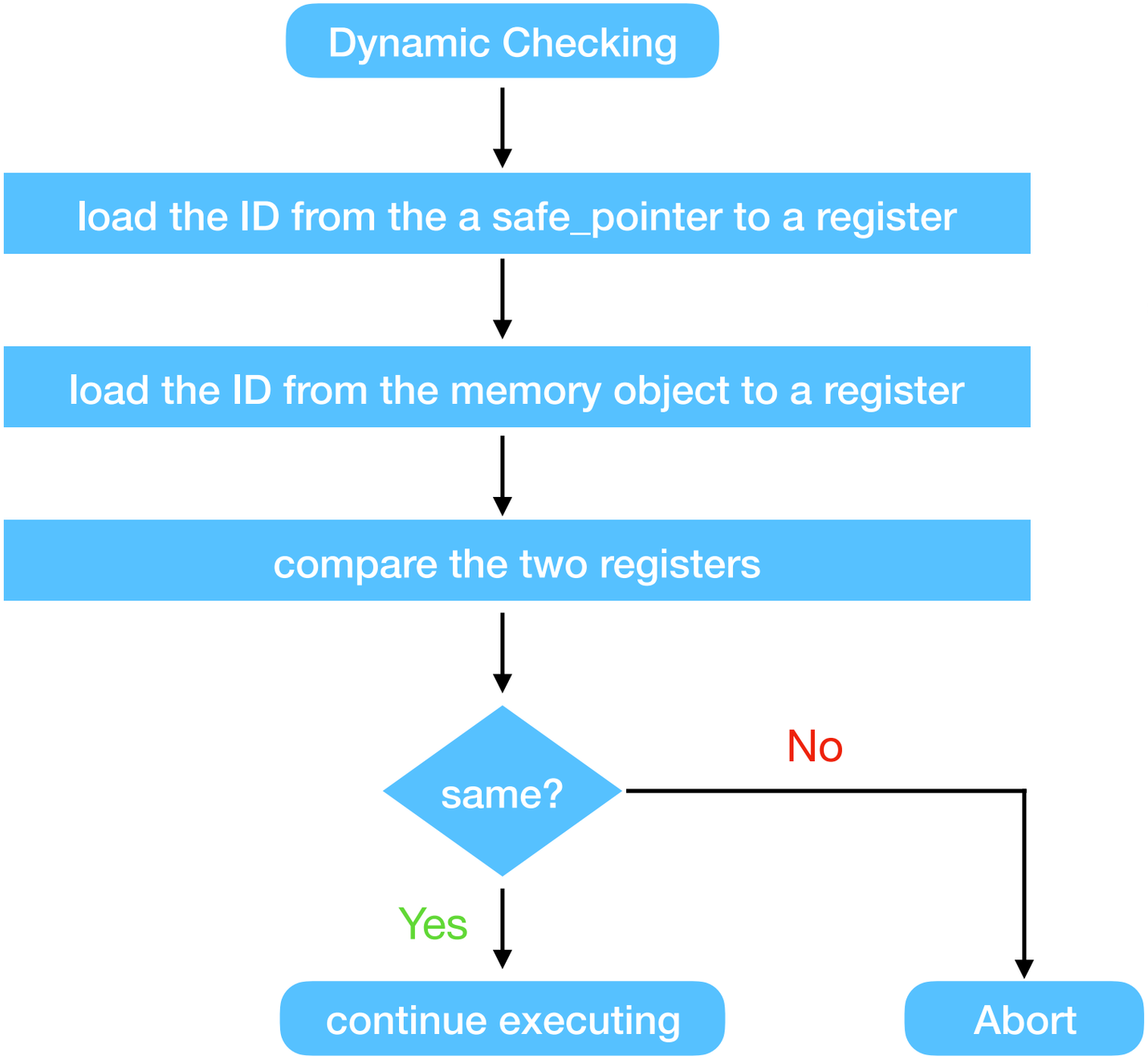


- on memory allocation: assign a non-zero unique integer to both the pointer key and the object lock.
- on memory deallocation: set the object's lock to 0.
- on pointer dereference: check if the key and lock match

Algorithm of Dynamic Checking



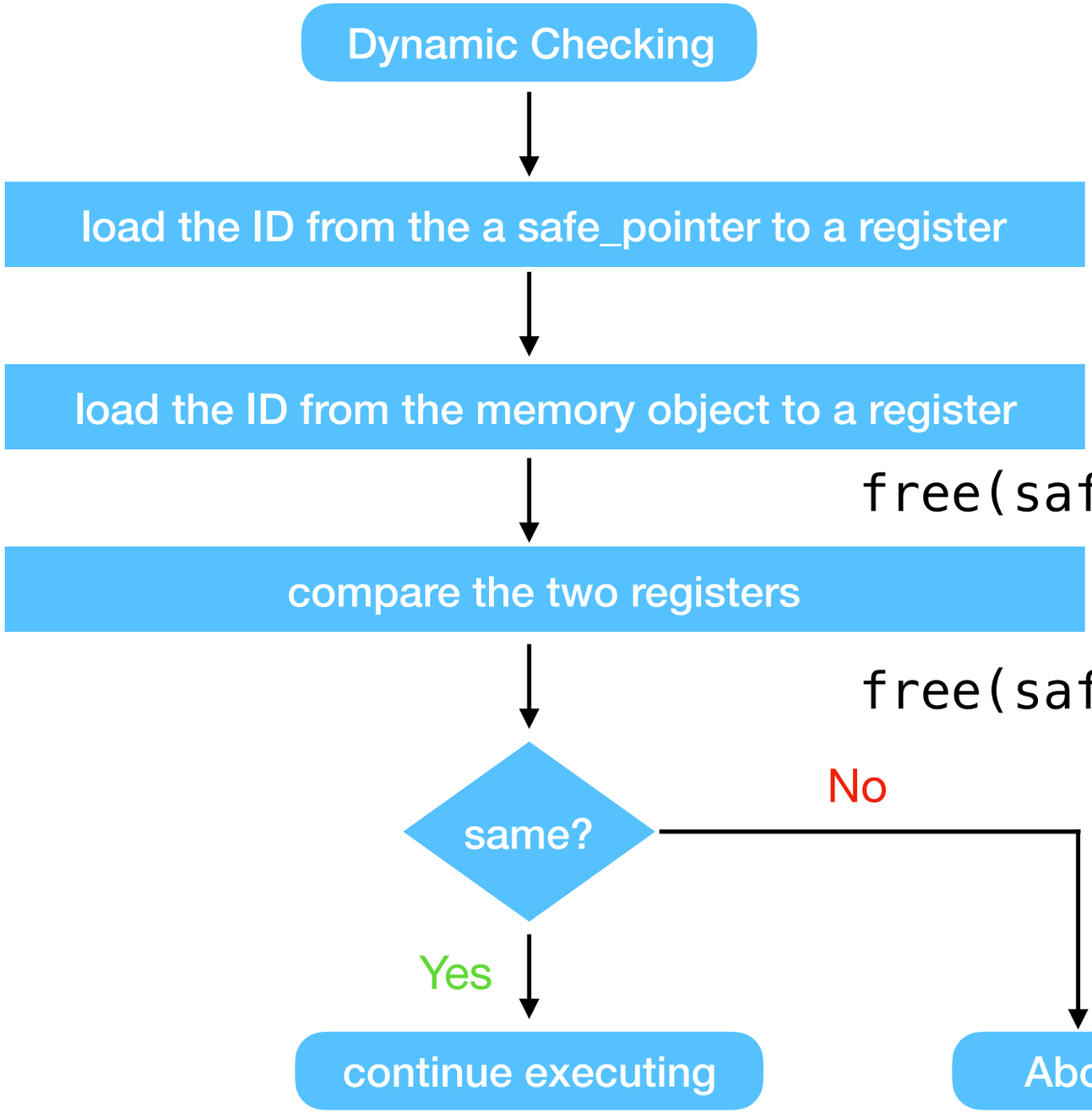
Algorithm of Dynamic Checking



execution overhead:
less than 6 instructions

memory overhead:
one word for each object
at most two words for each pointer

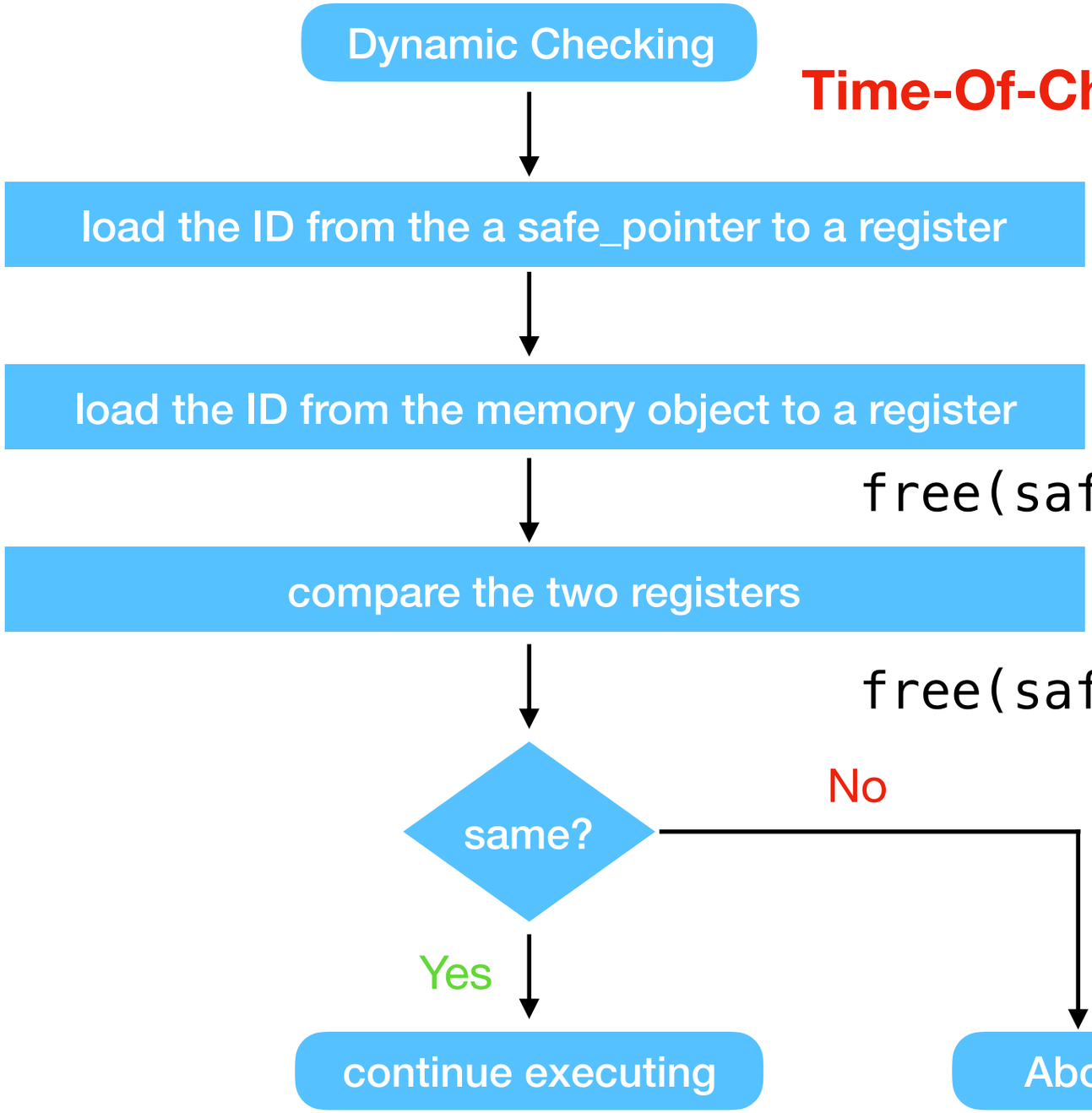
Multithreading Issue




```
free(safe_pointer);
```

```
free(safe_pointer);
```

Multithreading Issue



Time-Of-Check-Time-Of-Use (TOCTOU) bug!

```
free(safe_pointer);
```

```
free(safe_pointer);
```

Prevent TOCTOU Bugs

Option 1:

Transactional Memory: make pointer dereference atomic

Prevent TOCTOU Bugs

Option 1:

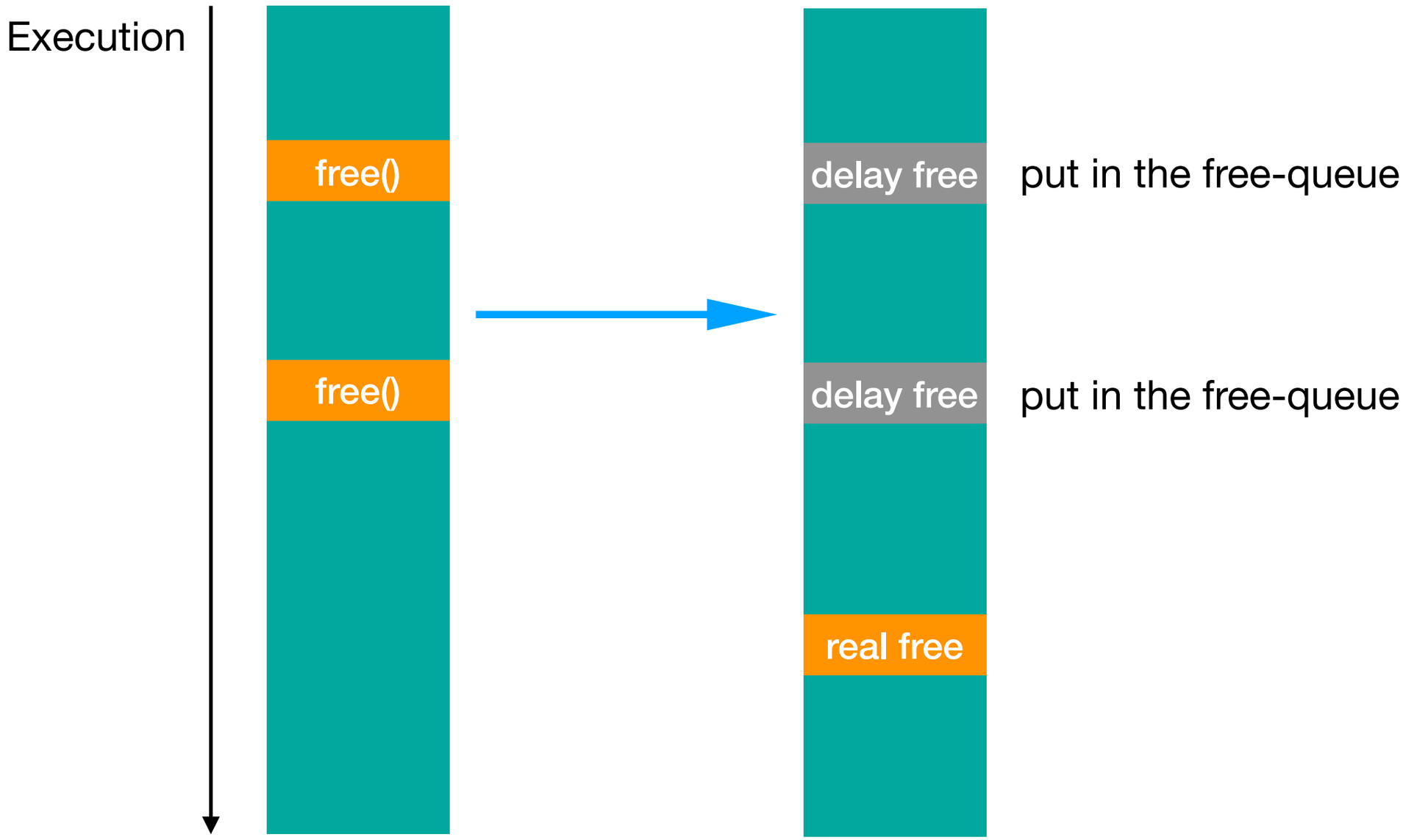
Transactional Memory: make pointer dereference atomic

Option 2:

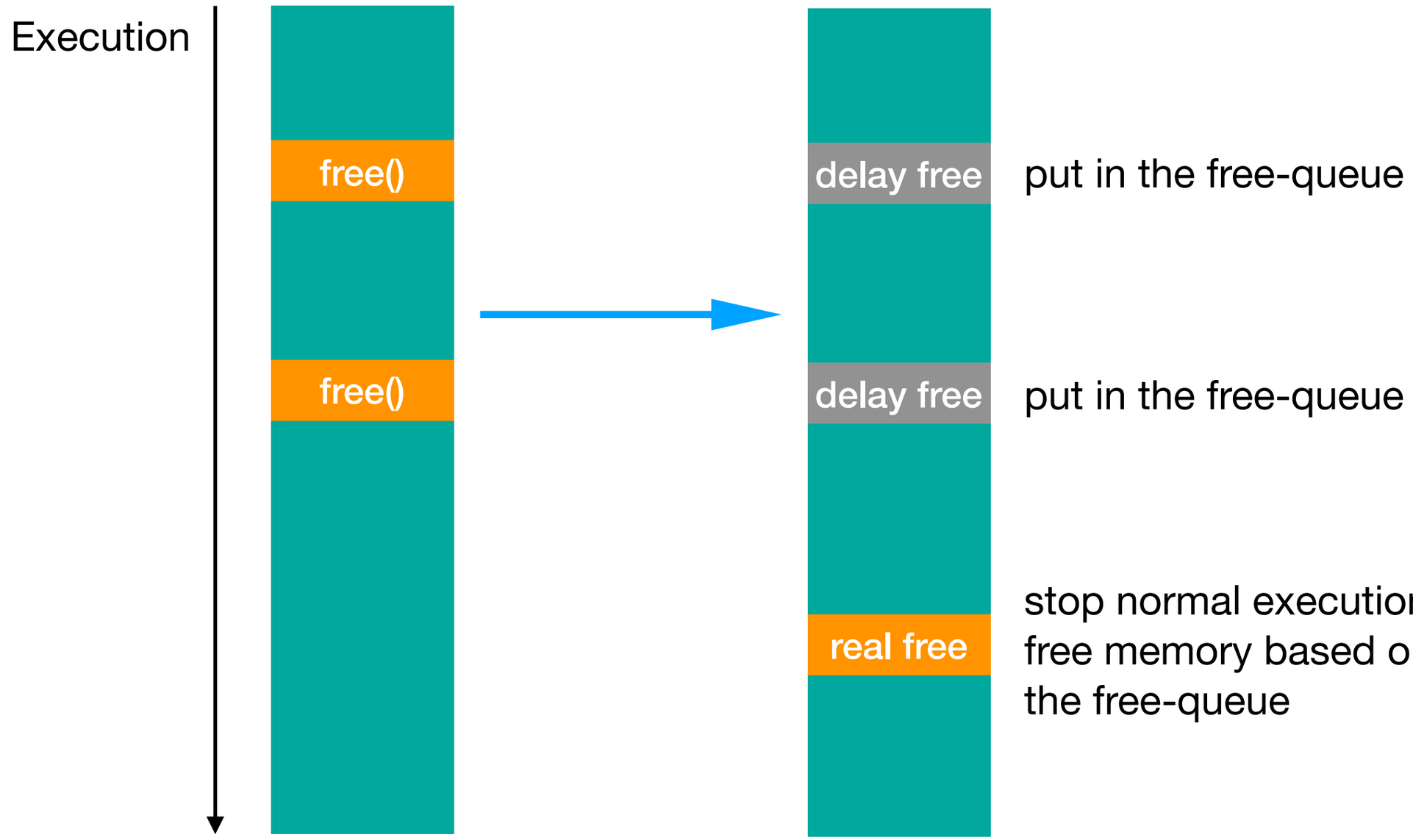
A fact: `free()` is much less frequent than pointer dereference.

Periodic Free: delay normal free, and periodically free the accumulated memory

Periodic Free



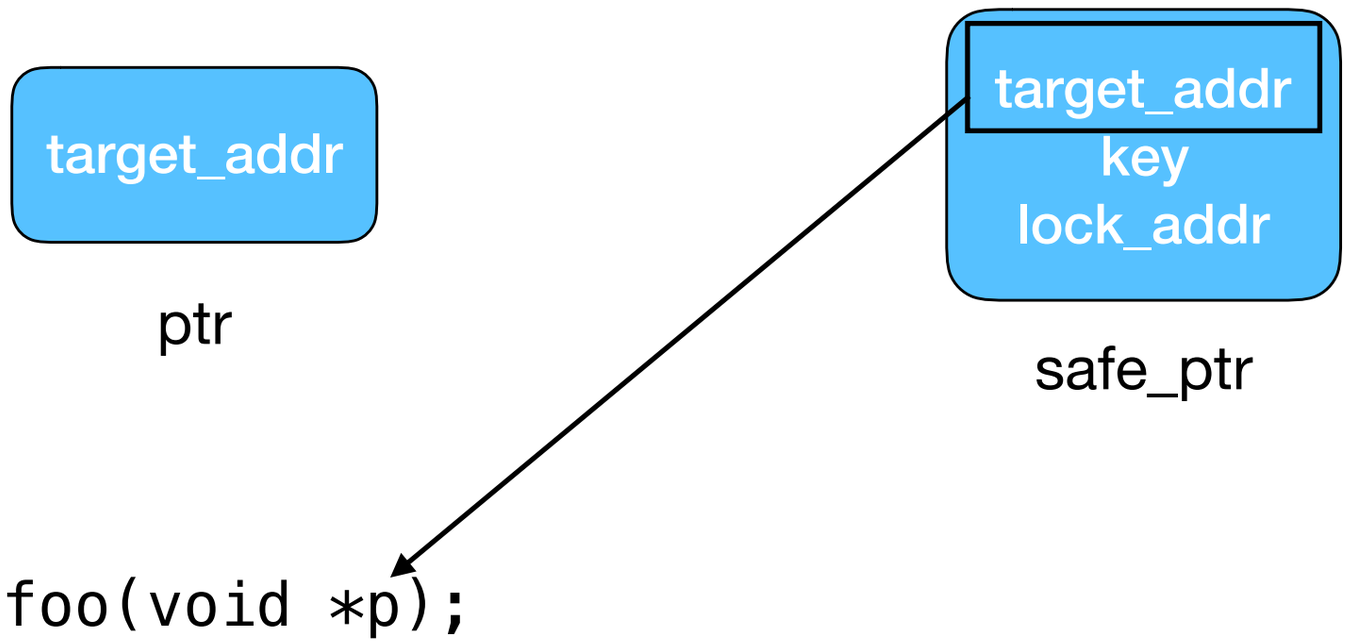
Periodic Free



Backward Compatibility

Aspects	What we want	Checked C's Choice
Performance Overhead	as low as possible	✓
Memory Overhead	as low as possible	✓
Backward Compatibility	compatible with legacy code	✗
Generality	general	✓
Programmer Control	high	✓
Programmer Efforts	as little as possible	medium
Soundness	sound	✓
Completeness	complete	✗
Require Source Code	no	✗

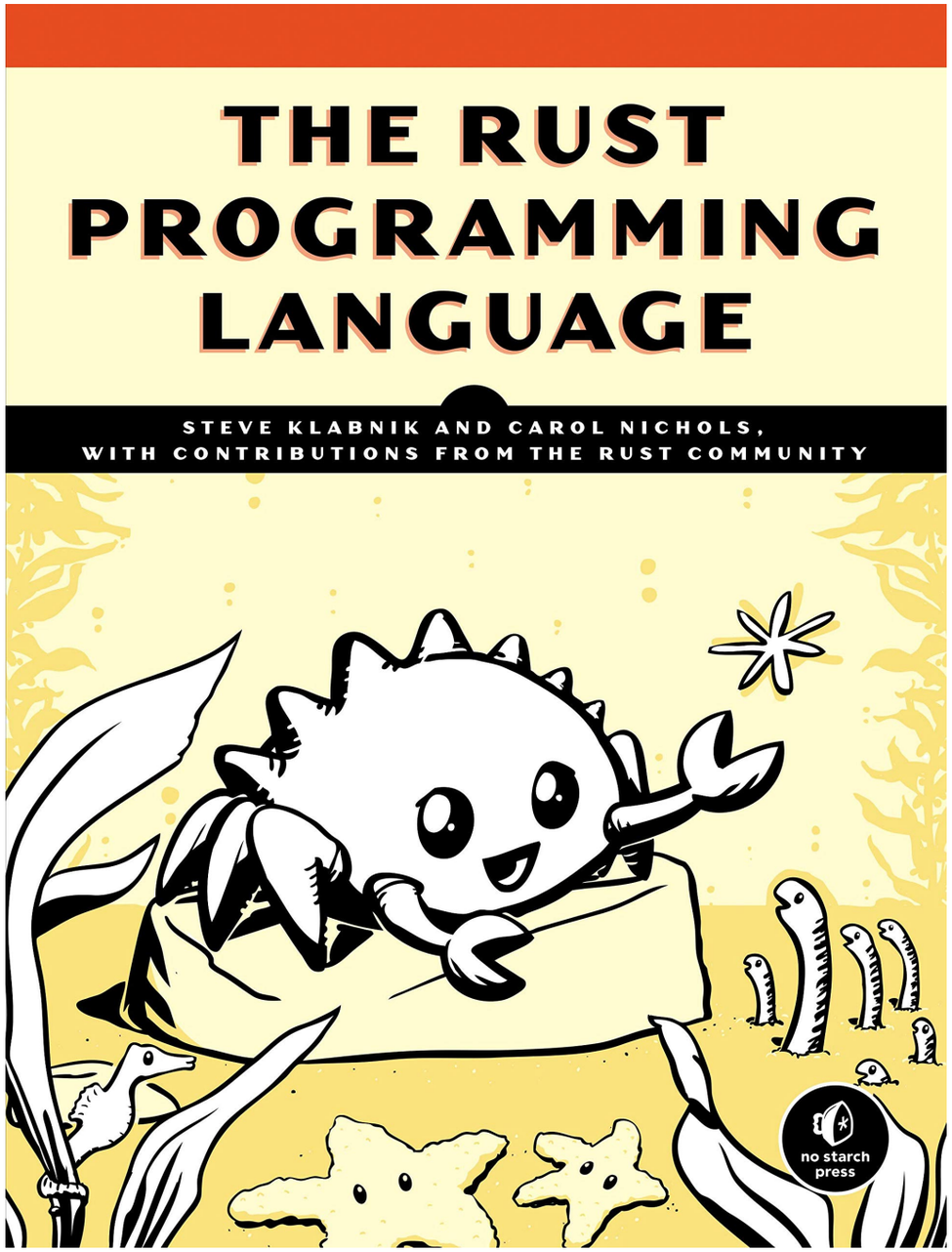
Backward Compatibility



Outline

- ❖ Existing Solutions to UAF
- ❖ Checked C Solution to UAF
- ❖ Demo (if time permits)

Why Not Using Rust?



- backward compatibility
- programmer efforts (financial concerns)