KU LEUVEN GENT

N00B: Bounds Checking for the Masses

DistriNet

Adriaan Jacobs, Carlo Ramponi, Jonas Roels, Bruno Crispo, Silviu Vlasceanu, Mahmoud Ammar, Stijn Volckaert

CCS'26, 2026/11/17

FBI

NEWS

The M Worm

30 Yea

Half a million widely trusted websites vulnerable to Heartbleed bug

A serious overrun vulnerability in the OpenSSL crypto of SSL web servers which use certificate

The Guardian

Heartbleed: catastrophic

Code error means 'heartbeat' function u

DAN GOODIN -

SC Magazine

Google patches new zero-day actively exploited in the Chrome browser

Google's most recent patch for Chrome is the fifth actively exploited zero-day targeted by threat actors this year in the popular browser.

3 days ago



wp Washington P

U.S. declares North Ko

In a statement Tuesda

WannaCry rans

19 De

SiliconANGLE

Chrome, Firefox and other br WebP vulnerability

Google LLC, the Mozilla Foundation to fix a zero-day vulnerability affecting the

3 weeks ago

Critical libwebp Vulnerability Under Active Exploitation - Gets Maximum CVSS Score

Google has assigned a new CVE identifier for a critical security flaw in the libwebp image library for rendering images in the WebP format...

5 days ago

nerability

4034



Discovered in polkit


FBI
NEWS
Half a million widely trusted websites vulnerable to Heartbleed bug
A serious overrun vulnerability in the OpenSSL crypto
of SSL web servers which use certificate
The Guardian
SC Magazine
sonatype
CVE-2026-109618 | CVE-2026-109619
PHP2SHELL RCE FLAWS
THREAT UPDATE CVE-2026-111497
Node2Shell CVE evolves to HTTP2shell affecting many libraries & apps using HTTPs
aikido/intel
Chrome, Fire
WebP vulnerability
Google LLC, the Mozilla Foundation
to fix a zero-day vulnerability affecting the
3 weeks ago
5 days ago
for rendering images in the WebP format...
a new CVE identifier for a critical security flaw in the libwebp
Exploitation - Gets
webp
Discovered in polkit

With great power comes great responsibility

- Uncle Ben

```
buffer[index] = content;
```

Random pointer/buffer getting exploited

Attacker-controlled offset → **attacker chooses what memory gets accessed**

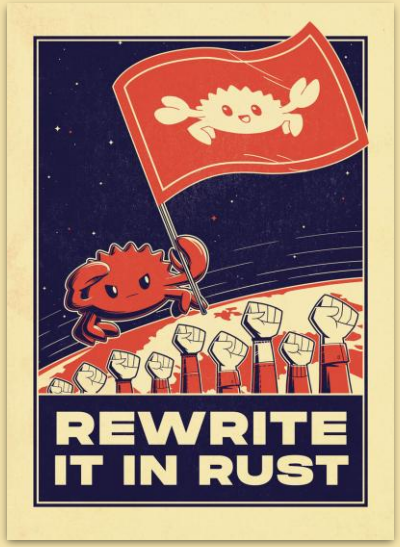
Data that the attacker wants to write to the chosen location

```
bool isAdmin;
std::function<void()> callback_func;
Person* personReceivingAMillionDollars;
```

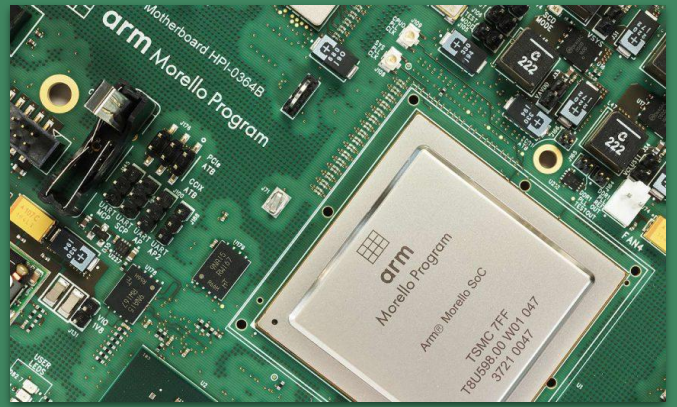

KU LEUVEN

Didn’t We Solve This Problem Already?

Memory-Safe Software



Memory-Safe Hardware

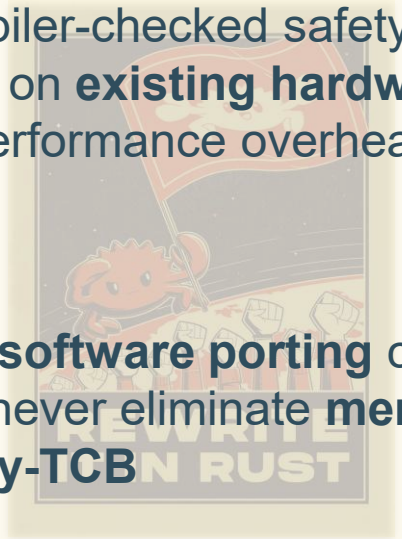


Didn’t We Solve This Problem Already?

Memory-Safe Software

- ✅ Compiler-checked safety
- ✅ Runs on **existing hardware**
- ✅ No performance overhead

- ❌ High **software porting** cost
- ❌ Can never eliminate **memory-safety-TCB**



Memory-Safe Hardware

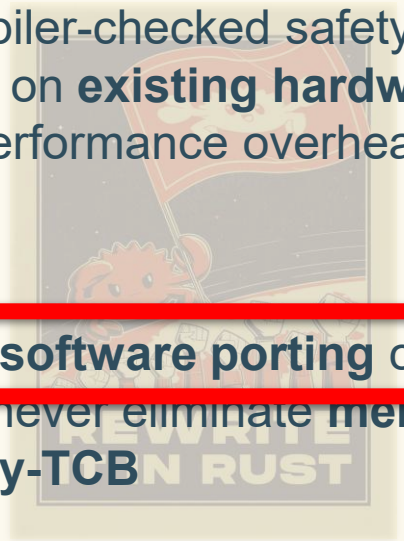
- ✅ Hardware-enforced safety
- ✅ Low software porting cost
- ✅ **Memory-safety-TCB** eliminated

- ❌ High **hardware replacement** cost
- ❌ Non-zero **performance overhead**



Didn’t We Solve This Problem Already?

Memory-Safe Software	Memory-Safe Hardware
✅ Compiler-checked safety	✅ Hardware-enforced safety
✅ Runs on **existing hardware**	✅ Low software porting cost
✅ No performance overhead	✅ **Memory-safety-TCB** eliminated
❌ High **software porting** cost	❌ High **hardware replacement** cost
❌ Can never eliminate **memory-safety-TCB**	❌ Non-zero **performance overhead**

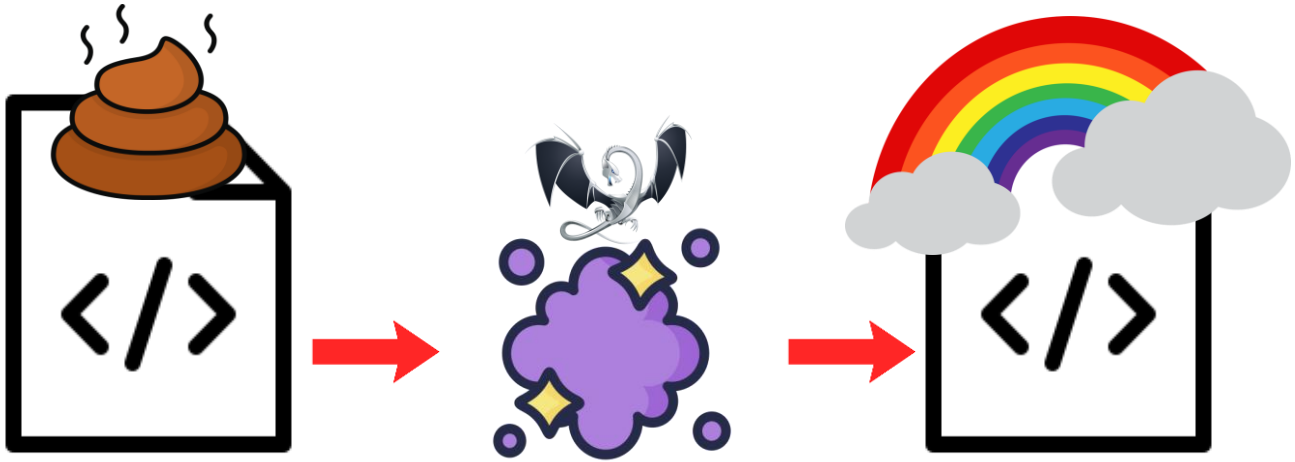




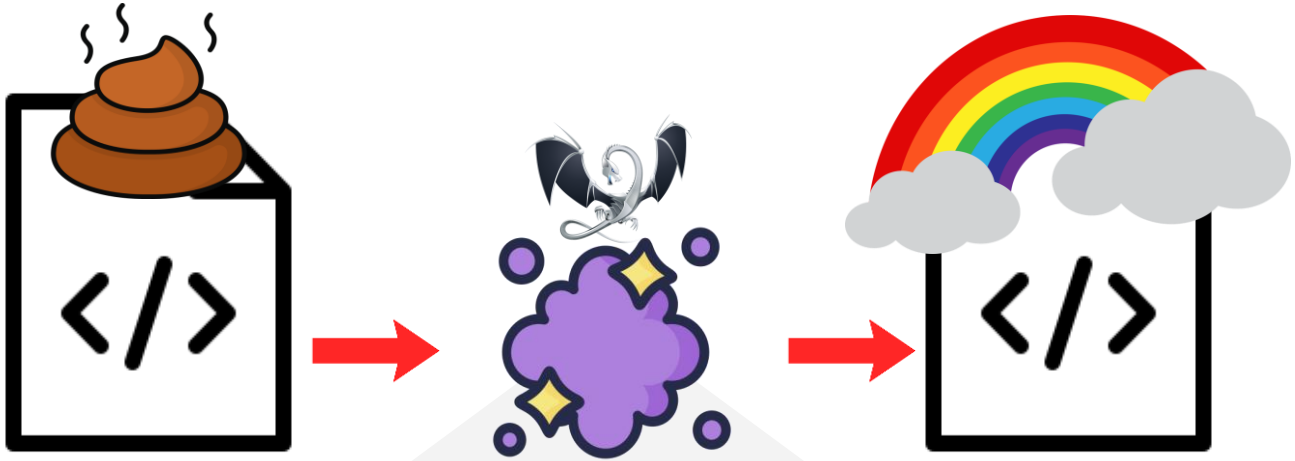


What Will Protect Us in the Meantime?

Bounds Checking to the Rescue!



Bounds Checking to the Rescue!




```
if (i < 0 || i >= size)
    trap();
buf[i] = data;
```

Bounds Checking to the Rescue ?

Approach	Type	Hardware Assistance	Pointer Instrumentation				Target Memory			Compatibility					Run-Time Overhead
			Deref.	Arith.	Escape	Copy	Stack	Heap	Global	Ext. code interop	Benign OoB ptrs	Maximal Address Space (2^x)	Max. #Live Objects	Max. Alloc. Size (GB)	
Fat Pointers [15–18]	P	-	✓	-	-	-	✓	✓	✓	○	●	▲	▲	▲	🔴
SoftBound [19]	P	-	✓	-	✓	✓	✓	✓	✓	◐	●	▲	▲	▲	🔴
Intel MPX [39, 122]	P	Intel MPX	✓	-	✓	✓	✓	✓	✓	◐	●	▲	▲	▲	🔴
MESH [41]	P	-	✓	✗	✓	-	✗	✓	✗	◐	●	48	2^{16}	▲	🔴
PACMem [20]	P	ARM PA	✓	✗	✓	-	✓	✓	✓	◐	●	39	2^{24}	4 GB	🔴
CUP [27]/Midfat [40]	P	-	✓	✓	✓	-	✓	✓	✓	◐	●	▲	2^{31}	4 GB	🟠
CGuard [30]	P	-	✓	✓	✓	-	✓	✓	✓	○	◐‡	48	▲	▲	🟠
Delta Pointers [28]	P	-	✓	✓	✓	-	✓	✓	✓	○	◐‡	32	▲	2 GB	🟠
J&K [14]	O	-	-	✓	-	-	✓	✓	✓	●	○†	▲	▲	▲	🔴
CRED [24]	O	-	✓	✓	-	-	✓	✓	✓	●	◐	▲	▲	▲	🔴
PariCheck [34]	O	-	-	✓	-	-	✓	✓	✓	●	◐	38	▲	▲	🔴
Baggy Bounds$^{64\text{-bit}}$ [22]	O	-	-	✓	-	-	✓	✓	✓	●	◐	38	▲	▲	🔴
LowFat [23, 38, 93]	O	-	-	✓	-	-	✓	✓	✓	●	○	▲	▲	32 GB	🟠
CAMP [25]	O	-	-	✓	-	-	✗	✓	✗	●	○†	▲	▲	▲	🔴
ShadowBound [56]	O	-	-	✓	-	-	✗	✓	✗	●	○	▲	▲	8 GB	🟠
TailCheck [31]	R	-	✓	✗	✓	-	✗	✓	✓	◐	●	48	▲	▲	🟢
StickyTags [76]	R	ARM MTE	✓	✗	-	-	✓	✓	✗	●	●	▲	▲	▲	🟢
ZOMETag [121]	P+O+R	ARM MTE	✓	✓	-	-	✓	✓	✓	●	●‡	48	$16 \times 2^{48-32} = 2^{20}$	4 GB	🟠
FRAMER [21, 70]	P+O	-	✓	✓	✓	-	✓	✓	✓	◐	◐‡	▲	▲	▲	🔴

Legend: **P** Pointer-based, **O** Object-based, **R** Red-zone-based (aka Tripwire), ✓ Has this feature, ✗ Lacks this feature, ✓ Has this feature (but is fundamentally flawed), ✗ Lacks this feature (but no fundamental issue in the design), - Feature is not required, ● Full Compatibility, ◐ Partial Compatibility, ○ No Compatibility, 🟢 <30% (average) overhead, 🟠 <50% (average) overhead, 🔴 >50% (average) overhead, ▲ The maximum possible by the underlying OS/architecture, † Only off by One, ‡ Within limited range. If the defense includes more than just spatial checks, we model their spatial component alone.

KU LEUVEN

Bounds Checking to the Rescue ?

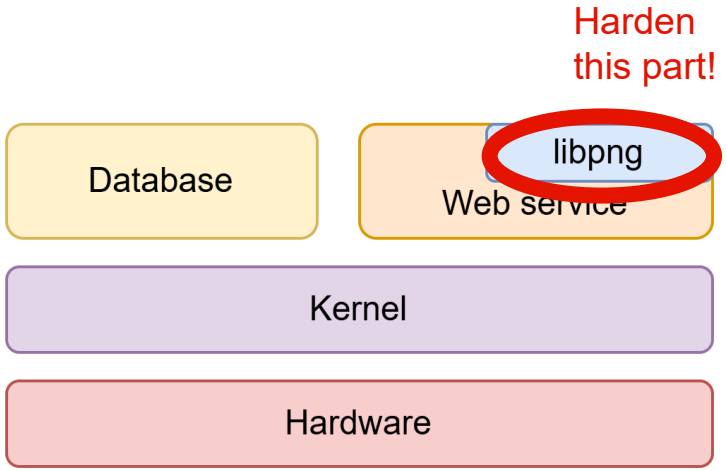


Approach	Type	Hardware Assistance	Pointer Instrumentation				Target Memory			Compatibility					Run-Time Overhead
			Deref.	Arith.	Escape	Copy	Stack	Heap	Global	Ext. code interop	Benign OoB ptrs	Maximal Address Space (2^x)	Max. #Live Objects	Max. Alloc. Size (GB)	
Fat Pointers [15–18]	P	-	✓	-	-	-	✓	✓			●	▲	▲	▲	●
SoftBound [19]	P	-	✓	-	✓	✓	✓	✓		◐	●	▲	▲	▲	●
Intel MPX [39, 122]	P	Intel MPX	✓	-	✓	✓		✓	✓	◐	●	▲	▲	▲	●
MESH [41]	P	-	✓	✗	✓	-	✓	✓	✗	●	●	48	2^{16}	▲	●
PACMem [20]	P	ARM PA	✓	✗	✓	-	✓	✓	✗	◐	●	39	2^{24}	4 GB	●
CUP [27]/Midfat [40]	P	-	✓	✓	✓	-	✓	✓	✓	◐	●	▲	2^{31}	4 GB	●
CGuard [30]	P		✓	✓	✓	-	✓	✓	✓	○	●	48	▲	▲	●
Delta Pointers [28]	P		✓	✓	✓	-	✓	✓	✓	○	◐‡	32	▲	2 GB	●
J&K [14]	O	-	✓	✓	✓	-	✓	✓	✓	●	○†	▲	▲	▲	●
CRED [21]	O	-	✓	✓	✓	-	✓	✓	✓	●	◐	▲	▲	▲	●
PariCheck [34]	O			✓			✓	✓	✓	●	◐	38	▲	▲	●
Baggy Bounds$^{64\text{-bit}}$ [2]	O			✓			✓	✓	✓	●	◐	38	▲	▲	●
LowFat [23, 38, 93]	O			✓			✓	✓	✓	●	●	▲	▲	32 GB	●
CAMP [25]	O			✓			✗	✓	✗	●	○†	▲	▲	▲	●
ShadowBound [56]	O			✓			✗	✓	✗	●	○	▲	▲	8 GB	●
TailCheck [31]	R		✓	✗	✓	-		✓	✓	◐	●	48	▲	▲	●
StickyTags [76]	R	ARM MTE	✓	✗	-		✓	✓	✗	●	●	▲	▲	▲	●
ZOMETag [121]	P+O+R	ARM MTE	✓	✓	✓		✓	✓	✓	●	●‡	48	$16 \times 2^{48-32} = 2^{20}$	4 GB	●
FRAMER [21, 70]	P+O	-	✓	✓	✓	-	✓	✓	✓	◐	◐‡	▲	▲	▲	●

Legend: **P** Pointer-based, **O** Object-based, **R** Red-zone-based (aka Tripwire), ✓ Has this feature, ✗ Lacks this feature, ✓ Has this feature (but is fundamentally flawed), ✗ Lacks this feature (but no fundamental issue in the design), - Feature is not required, ● Full Compatibility, ◐ Partial Compatibility, ○ No Compatibility, ● <30% (average) overhead, ● <50% (average) overhead, ● >50% (average) overhead, ▲ The maximum possible by the underlying OS/architecture, † Only off by One, ‡ Within limited range. If the defense includes more than just spatial checks, we model their spatial component alone.

KU LEUVEN

Is Performance Really The (Only) Problem?



Bounds Checkers Have *Functional* Challenges, Too

How to recover original bounds
during dereference?

Bounds Checkers Have *Functional* Challenges, Too

How to recover original bounds during dereference?

Propagate bounds information with all pointers throughout the program.

POINTER-BASED

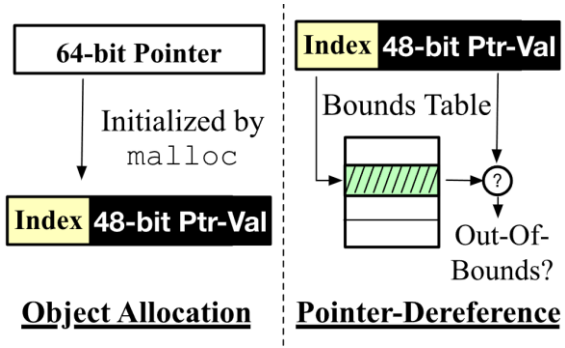
```
struct fat_ptr {
    void* ptr;
    void* upper_bound;
    void* lower_bound;
};
```

Bounds Checkers Have *Functional* Challenges, Too

How to recover original bounds during dereference?

Propagate bounds information with all pointers throughout the program.

POINTER-BASED



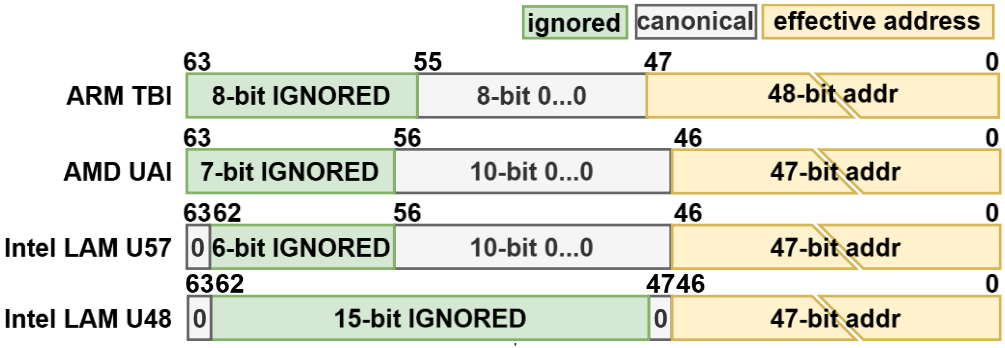
 Saileshwar, Gururaj, et al. "Heapcheck: Low-cost hardware support for memory safety." TACO'22.

Bounds Checkers Have *Functional* Challenges, Too

How to recover original bounds during dereference?

Propagate bounds information with all pointers throughout the program.

POINTER-BASED



 Saileshwar, Gururaj, et al. "Heapcheck: Low-cost hardware support for memory safety." TACO'22.

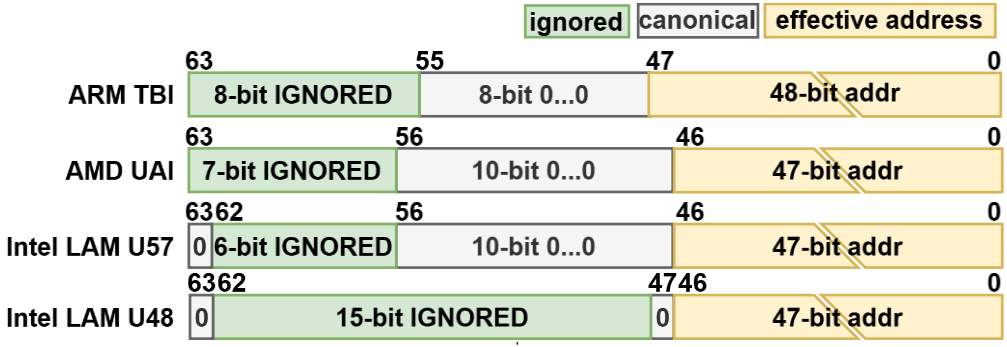
KU LEUVEN

Bounds Checkers Have *Functional* Challenges, Too

How to recover original bounds during dereference?

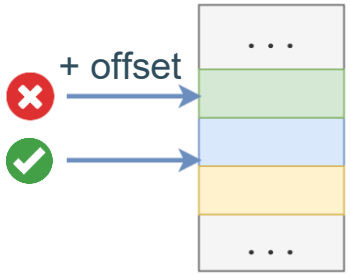
Propagate bounds information with all pointers throughout the program.

POINTER-BASED



Never let pointers go out of bounds in the first place.

OBJECT-BASED



Saileshwar, Gururaj, et al. "Heapcheck: Low-cost hardware support for memory safety." TACO'22.

Problem: Benign Out-of-Bounds Pointers

```
for (ptr = v.begin(); ptr != v.end(); ptr++)
  *ptr = ...
```

```
enum CHAR {
  CHAR_MIN = 100,
  C, D, E, F,
}

chars->encoding = malloc(size) - CHAR_MIN
// ...
chars->encoding[input_char()]
```



Problem: Benign Out-of-Bounds Pointers

```
for (ptr = v.begin(); ptr != v.end(); ptr++)
```

Problem : *Object-based approaches cannot distinguish between intentional and unintentional out-of-bounds pointer arithmetic, possibly leading to false positives.*

```
// ...
chars->encoding[input_char()]
```





N00B: Hybrid Bounds Checking

"Allow unrestricted benign OoB pointers"

"Crash on all benign OoB pointers"



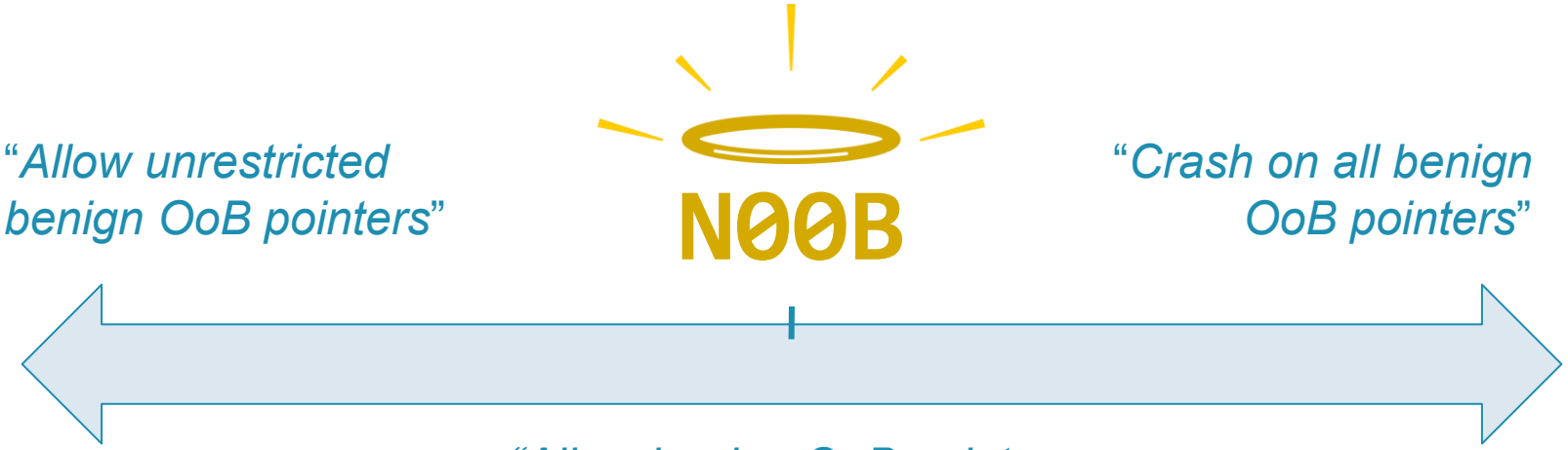
POINTER-BASED

OBJECT-BASED





N00B: Hybrid Bounds Checking



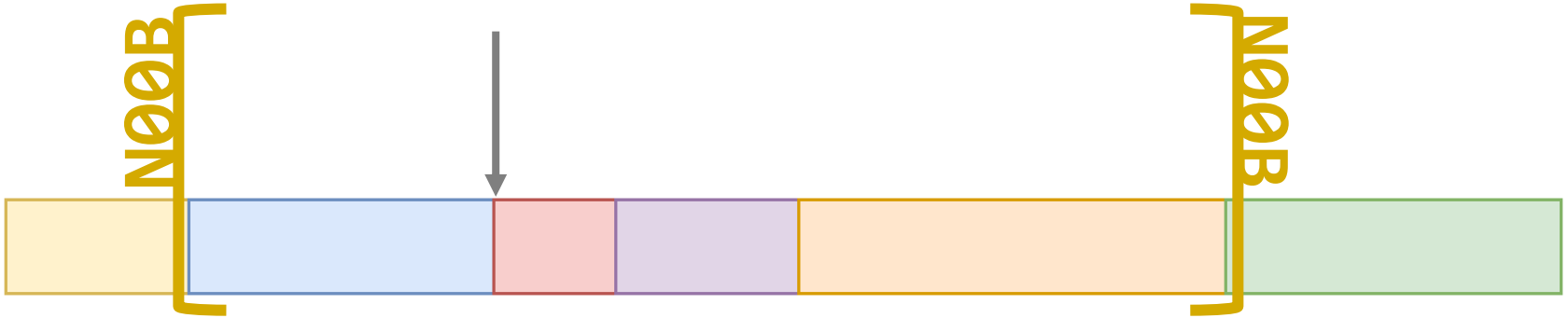




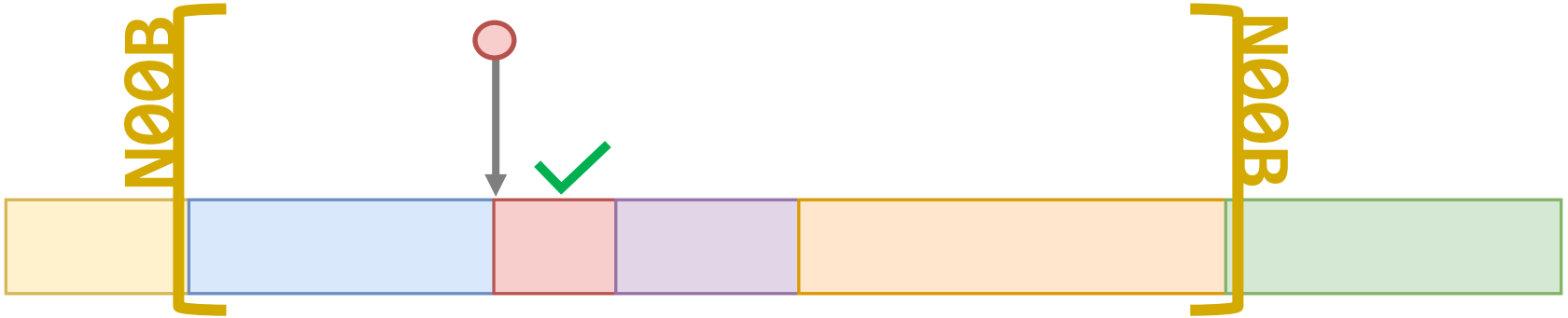


KU LEUVEN

The N00B Idea

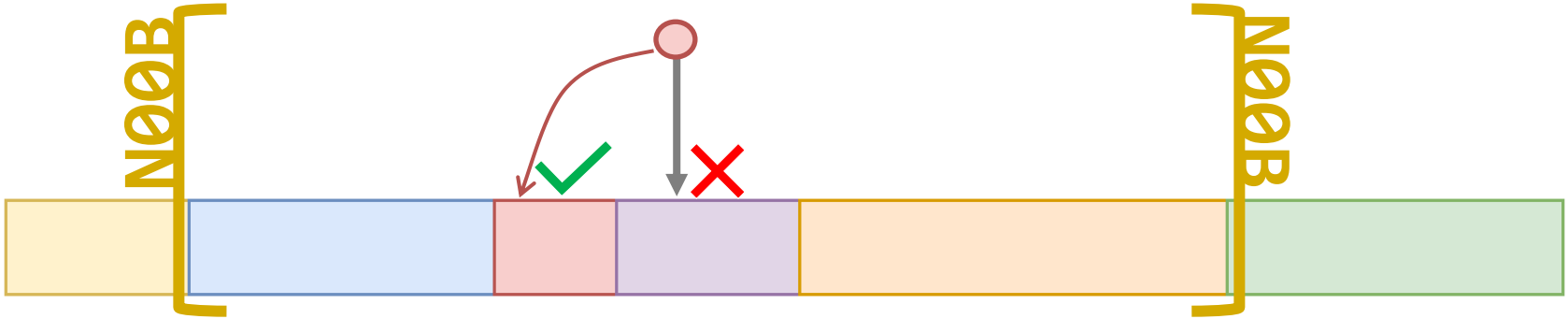


The N00B Idea



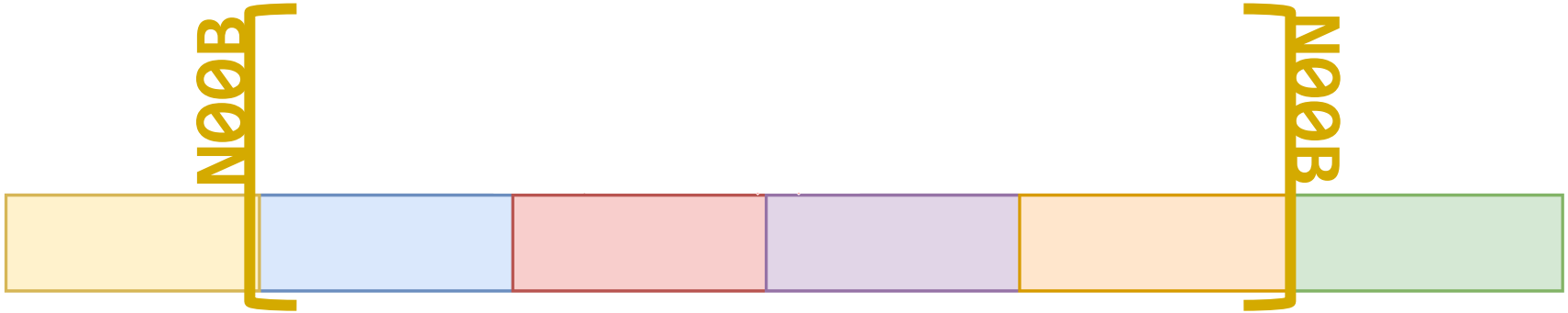
KU LEUVEN

The N00B Idea

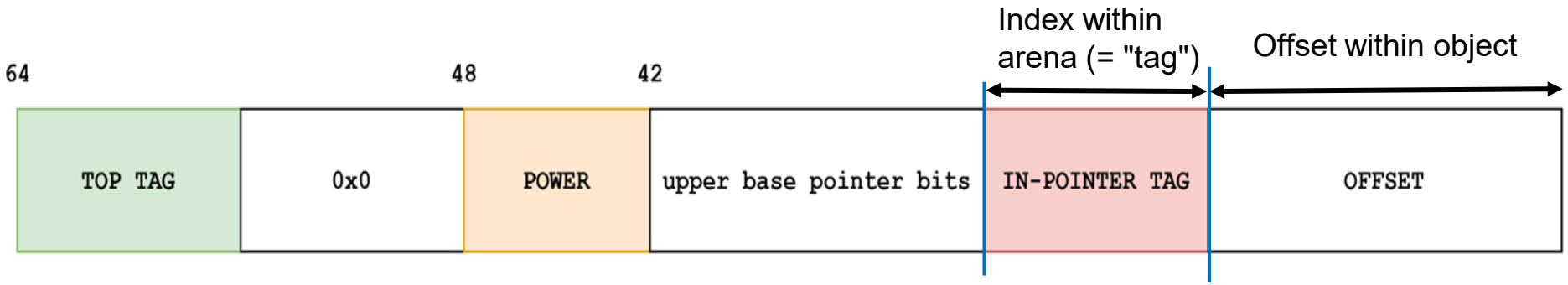


KU LEUVEN

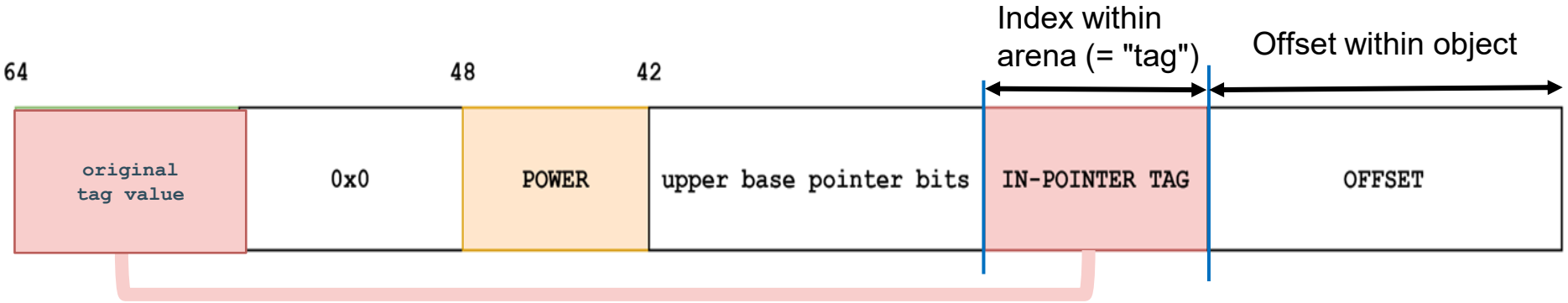
The N00B Idea



N00B Checks & Pointer Layout

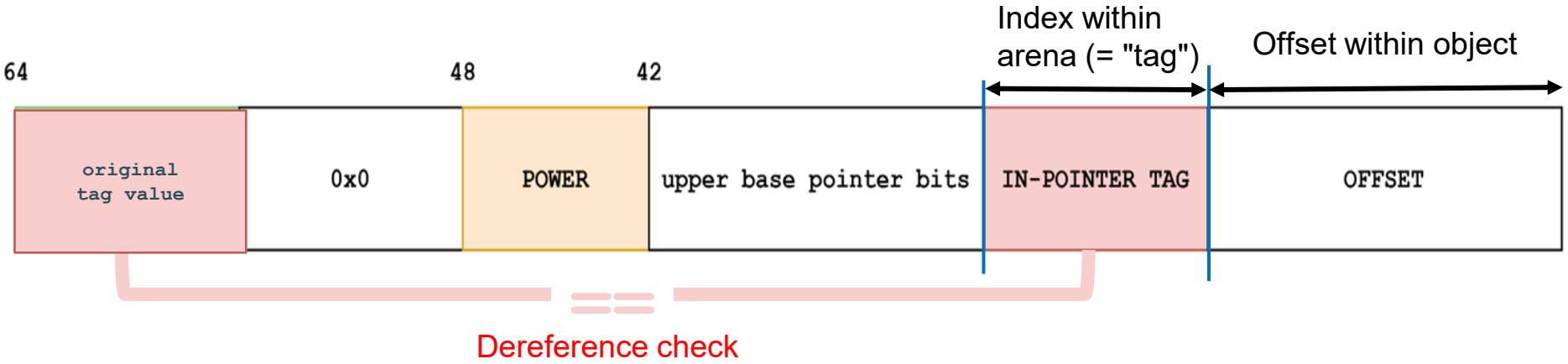


N00B Checks & Pointer Layout

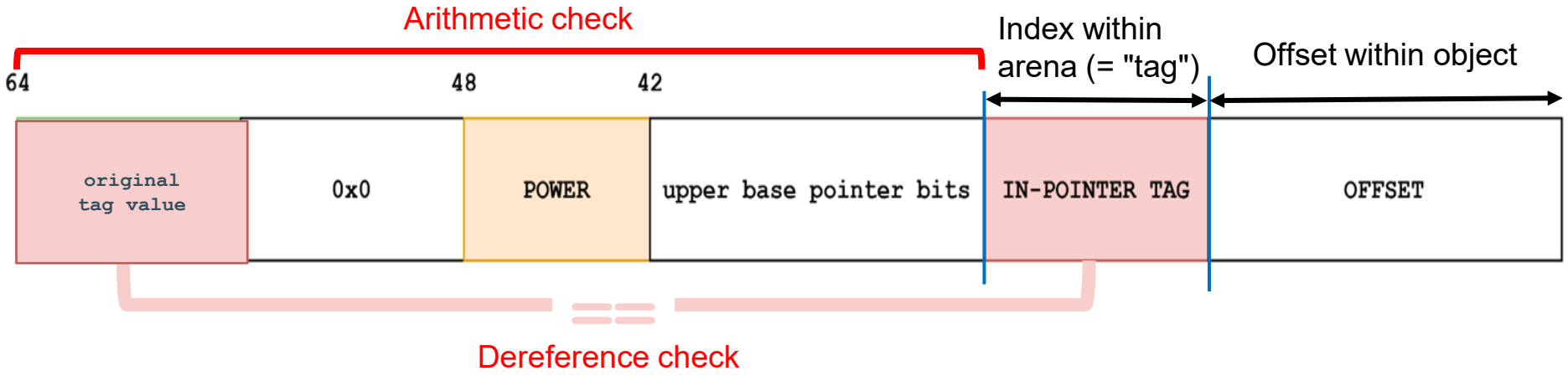


KU LEUVEN

N00B Checks & Pointer Layout

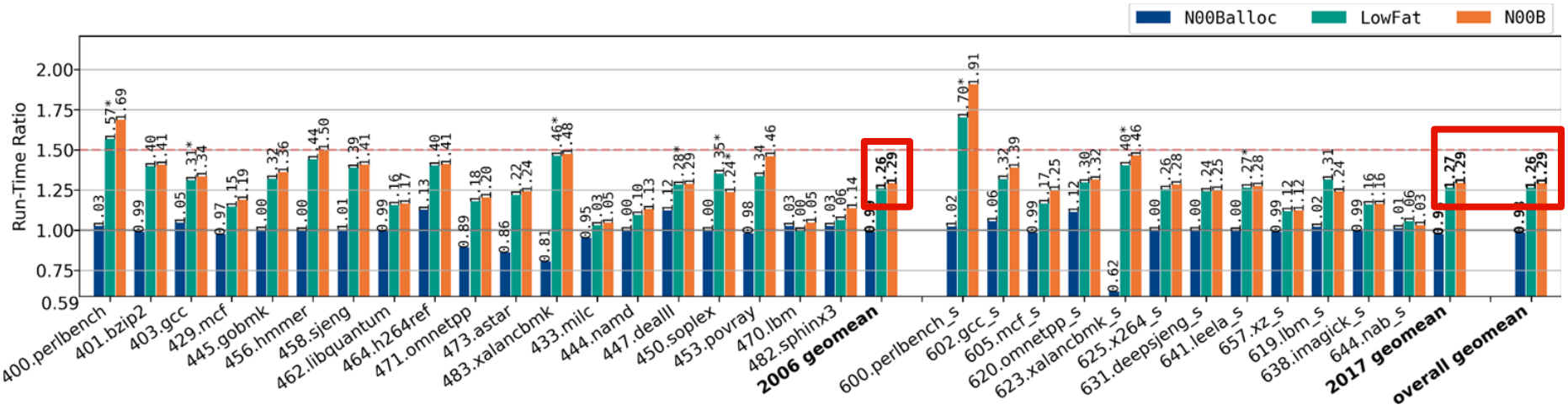


N00B Checks & Pointer Layout



KU LEUVEN

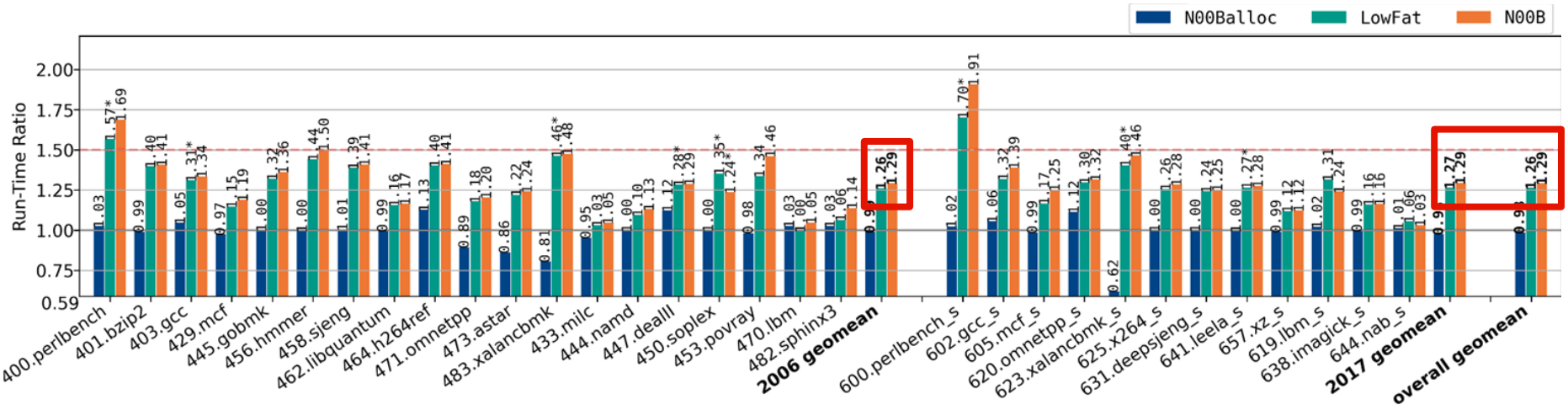
Twice the Checks, Twice the Overhead?



Twice the Checks, Twice the Overhead?

Table 4: Run-time overhead comparison of **N00B** versus the fastest current software-based bounds checkers.

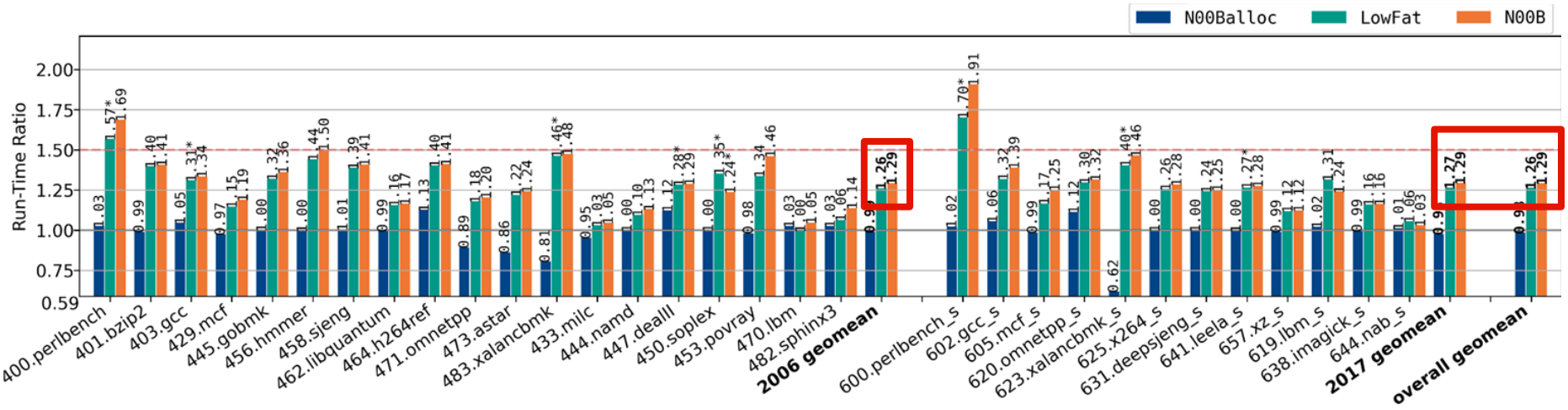
Bounds Checker	SPEC CPU2006	SPECspeed 2017
CAMP [26]	76.1% [76]	48.6% [76]
ShadowBound [53]	20.1% [76]	5.72%
Low-Fat [24]	26–43%	27–42%
Delta Pointers [29]	35% [61]	–
CGuard [31]	–	42.1%
N00B	**29–54%**	**29–54%**

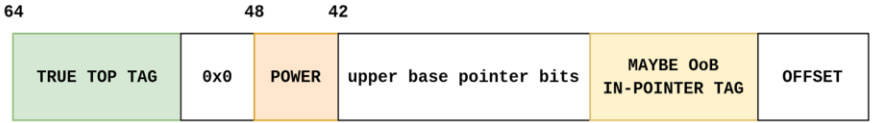


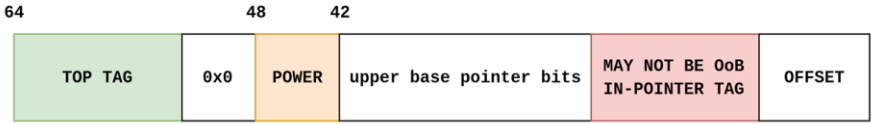
Twice the Checks, Twice the Overhead?

Table 4: Run-time overhead comparison of **N00B** versus the fastest current software-based bounds checkers.

Bounds Checker	SPEC CPU2006	SPECspeed 2017
CAMP [26]	76.1% [76]	48.6% [76]
ShadowBound [53]	20.1% [76]	5.72%
Low-Fat [24]	26–43%	27–42%
Delta Pointers [29]	35% [61]	–
CGuard [31]	–	42.1%
N00B	**29–54%**	**29–54%**



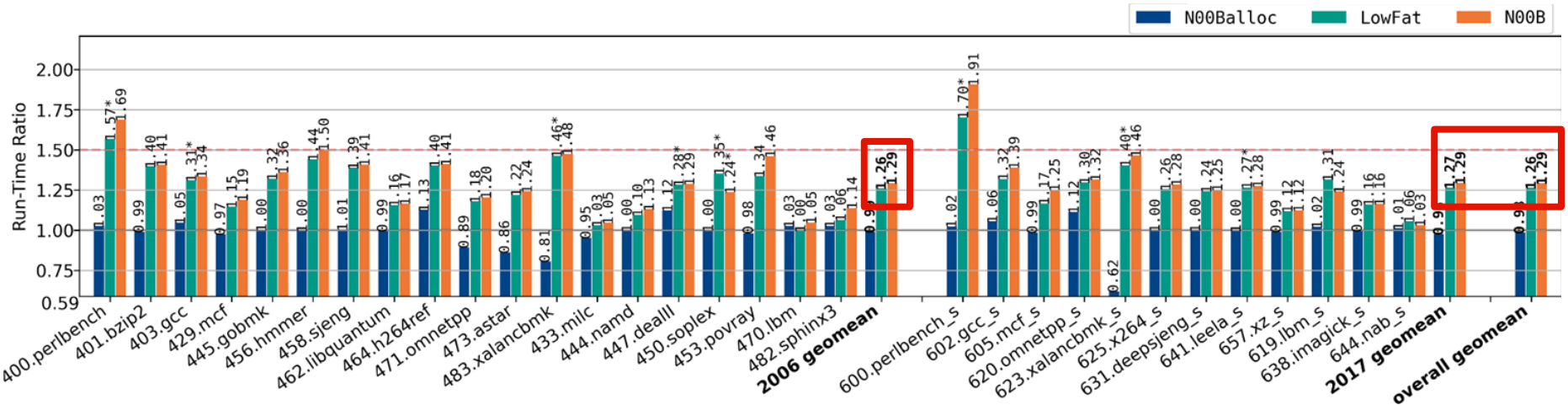


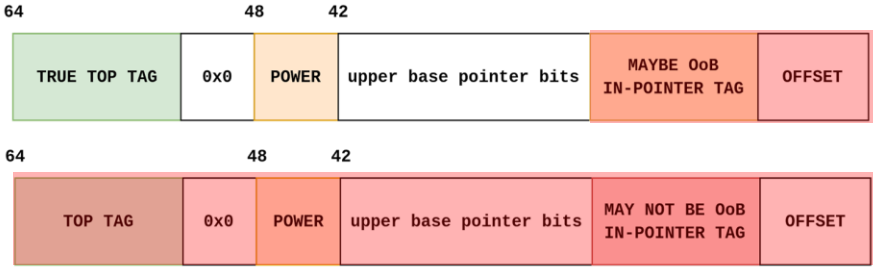


Twice the Checks, Twice the Overhead?

Table 4: Run-time overhead comparison of **N00B** versus the fastest current software-based bounds checkers.

Bounds Checker	SPEC CPU2006	SPECspeed 2017
CAMP [26]	76.1% [76]	48.6% [76]
ShadowBound [53]	20.1% [76]	5.72%
Low-Fat [24]	26–43%	27–42%
Delta Pointers [29]	35% [61]	–
CGuard [31]	–	42.1%
N00B	**29–54%**	**29–54%**

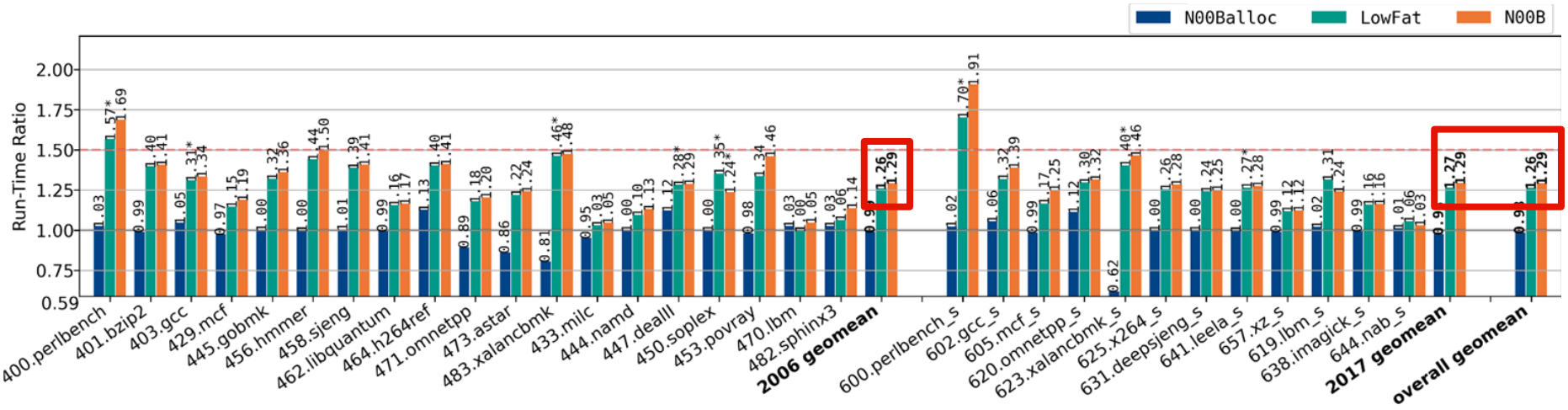


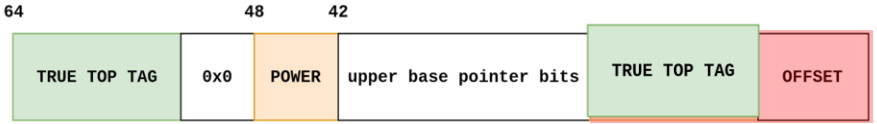


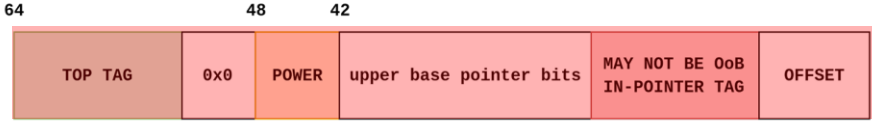
Twice the Checks, Twice the Overhead?

Table 4: Run-time overhead comparison of **N00B** versus the fastest current software-based bounds checkers.

Bounds Checker	SPEC CPU2006	SPECspeed 2017
CAMP [26]	76.1% [76]	48.6% [76]
ShadowBound [53]	20.1% [76]	5.72%
Low-Fat [24]	26–43%	27–42%
Delta Pointers [29]	35% [61]	–
CGuard [31]	–	42.1%
N00B	**29–54%**	**29–54%**



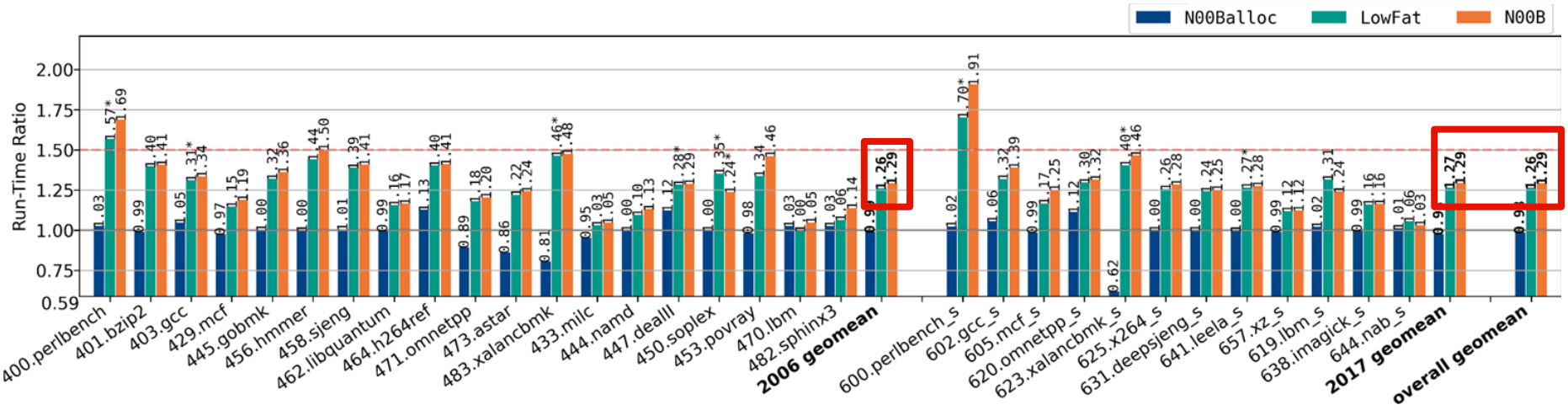


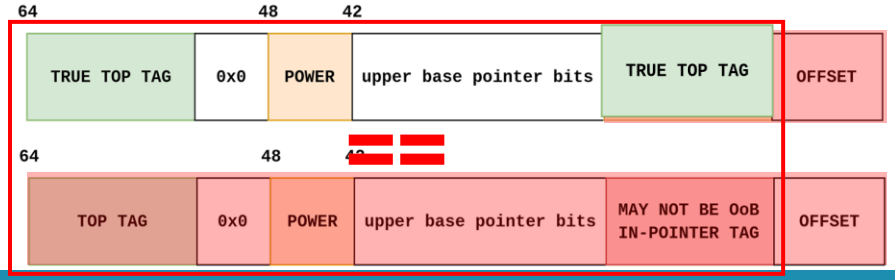


Twice the Checks, Twice the Overhead?

Table 4: Run-time overhead comparison of **N00B** versus the fastest current software-based bounds checkers.

Bounds Checker	SPEC CPU2006	SPECspeed 2017
CAMP [26]	76.1% [76]	48.6% [76]
ShadowBound [53]	20.1% [76]	5.72%
Low-Fat [24]	26–43%	27–42%
Delta Pointers [29]	35% [61]	–
CGuard [31]	–	42.1%
N00B	**29–54%**	**29–54%**

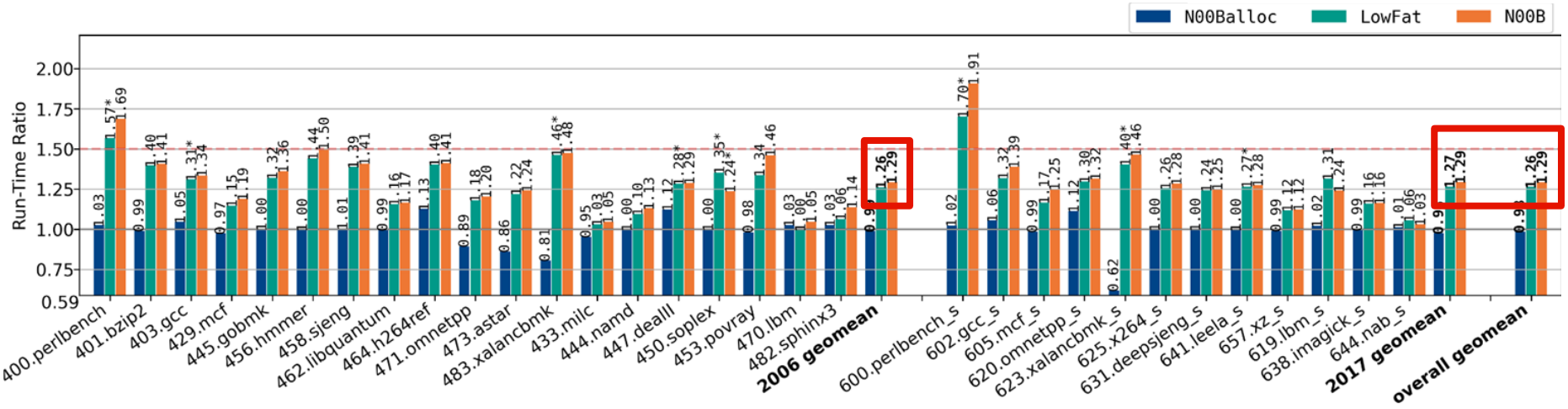


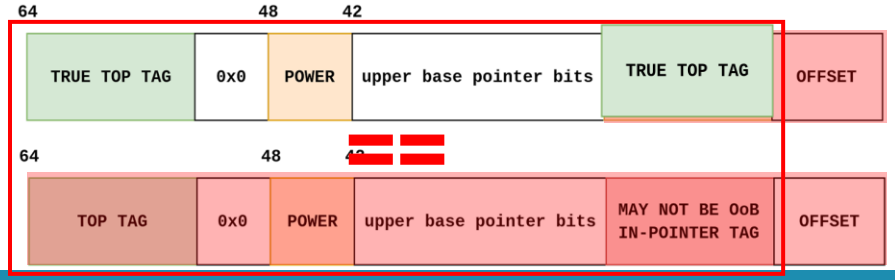


Twice the Checks, Twice the Overhead?

Table 4: Run-time overhead comparison of **N00B** versus the fastest current software-based bounds checkers.

Bounds Checker	SPEC CPU2006	SPECspeed 2017
CAMP [26]	76.1% [76]	48.6% [76]
ShadowBound [53]	20.1% [76]	5.72%
Low-Fat [24]	26–43%	27–42%
Delta Pointers [29]	35% [61]	–
CGuard [31]	–	42.1%
N00B	**29–54%**	**29–54%**





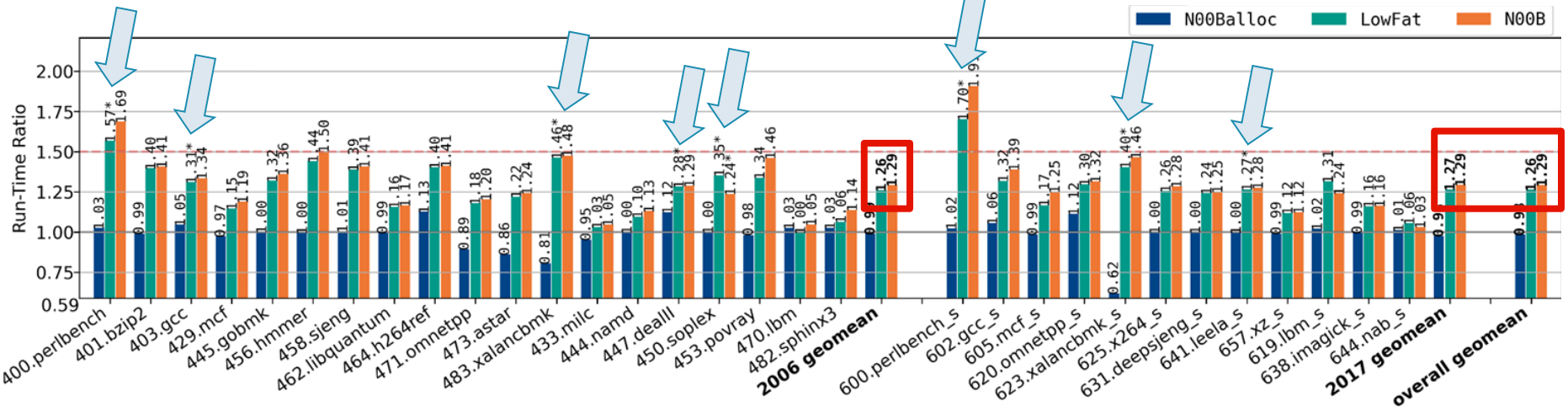

```
1 N = (base >> 42) & 0x7f;
2 referent = (base >> N) & ~TAG_MAX; /*arena base*/
3 toptag = base >> (64 - TW);        /*toptag*/
4 referent |= toptag;                /*orig obj*/
5 if (referent != (ptr >> N)) trap();
```

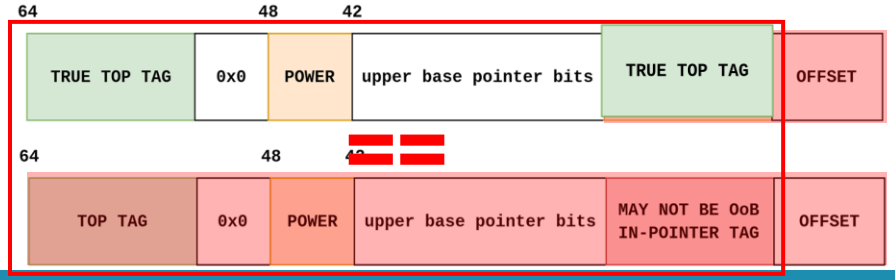

KU LEUVEN

Twice the Checks, Twice the Overhead?

Table 4: Run-time overhead comparison of **N00B** versus the fastest current software-based bounds checkers.

Bounds Checker	SPEC CPU2006	SPECspeed 2017
CAMP [26]	76.1% [76]	48.6% [76]
ShadowBound [53]	20.1% [76]	5.72%
Low-Fat [24]	26–43%	27–42%
Delta Pointers [29]	35% [61]	–
CGuard [31]	–	42.1%
N00B	**29–54%**	**29–54%**






```
1 N = (base >> 42) & 0x7f;
2 referent = (base >> N) & ~TAG_MAX; /*arena base*/
3 toptag = base >> (64 - TW);        /*toptag*/
4 referent |= toptag;                /*orig obj*/
5 if (referent != (ptr >> N)) trap();
```


KU LEUVEN

N00B Still Breaks Some Programs...

Table 2. Applied N00B compatibility patches.

Program	Revision	Year	Message
403.gcc	r62672	2003	"Don't use offset pointers."
403.gcc	r89543	2004	"Avoid undefined pointer arithmetic on qty_table"
403.gcc	r79945	2004	"Don't play queer offsetting games with reg_known_value"

KU LEUVEN

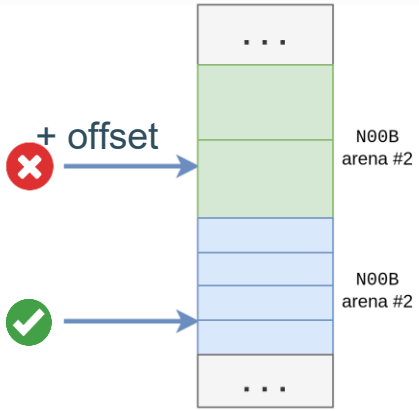
N00B Still Breaks Some Programs...

```
1 offset = realloc(array, grow) - array;
2 for (i = 0; i < size; i++)
3     array[i].ptr += offset;
```

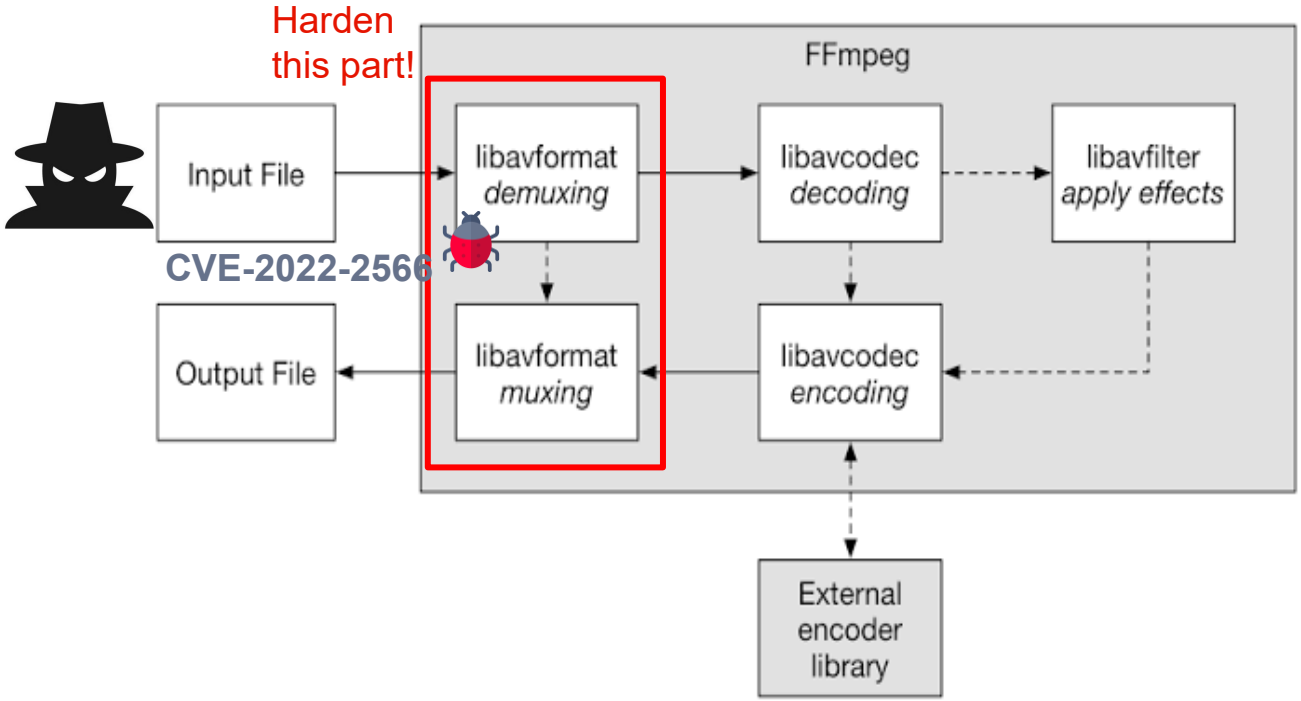

KU LEUVEN

N00B Still Breaks Some Programs...

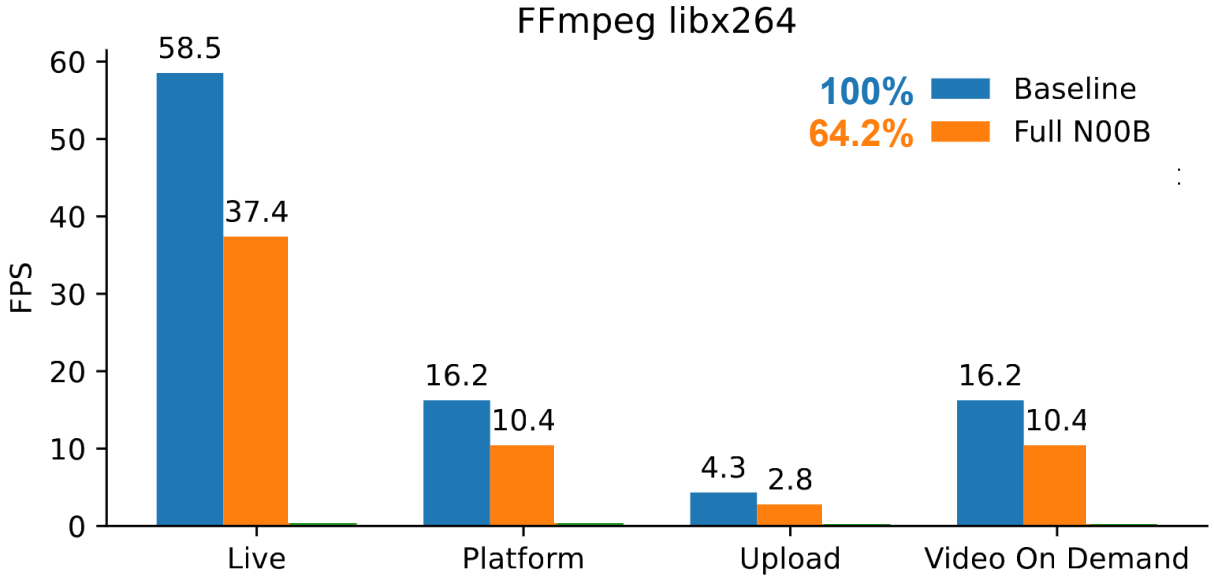
```
1 offset = realloc(array, grow) - array;
2 for (i = 0; i < size; i++)
3     array[i].ptr += offset;
```



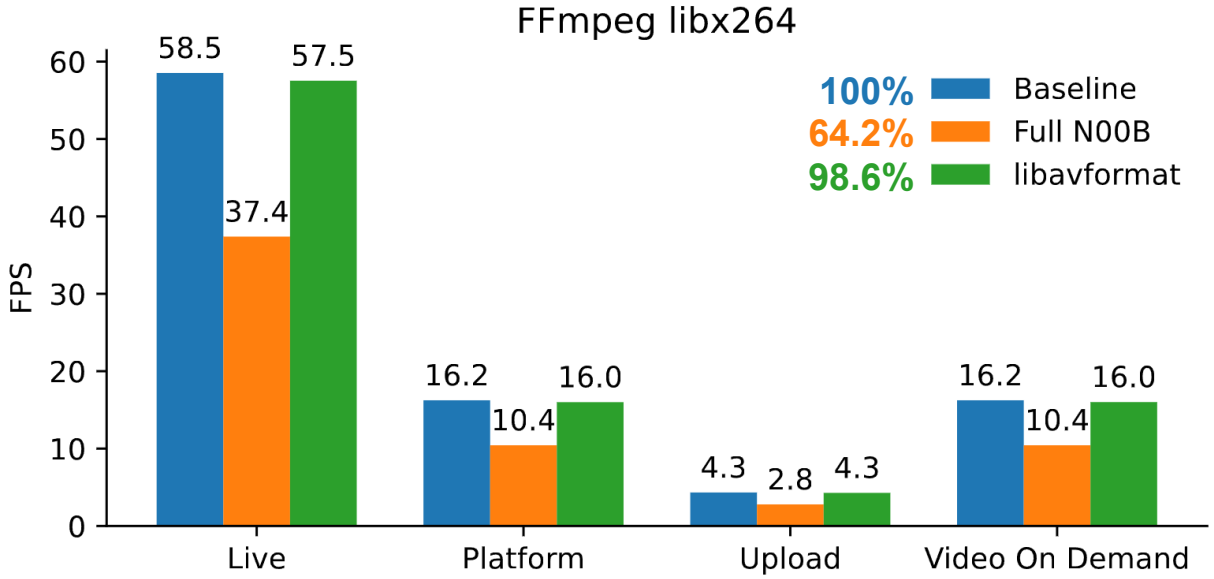
Selective Hardening Works!



Selective Hardening Works!



Selective Hardening Works!



N00B:
Bounds Checking for the Masses

Adriaan Jacobs
DistriNet, KU Leuven

Carlo Ramponi
University of Trento

Jonas Roels
DistriNet, KU Leuven

Bruno Crispo
University of Trento

Silviu Vlasceanu
Huawei Research

Mahmoud Ammar
Huawei Research

Stijn Volckaert
DistriNet, KU Leuven

Questions?