Aalto University

Hardware-assisted run-time Protection

N. Asokan
http://asokan.org/asokan/
@nasokan

Joint work: Hans Liljestrand, Thomas Nyman, Jan-Erik Ekberg, N. Asokan

Protect against runtime attacks without incurring a significant performance penalty

Full memory safety ideal solution

Ensures that a pointer is used only to access its intended object

- **Prevents malicious pointer manipulation, i.e., control-flow attacks**
- **Prevents data-only attacks**

Difficult to realize in practice

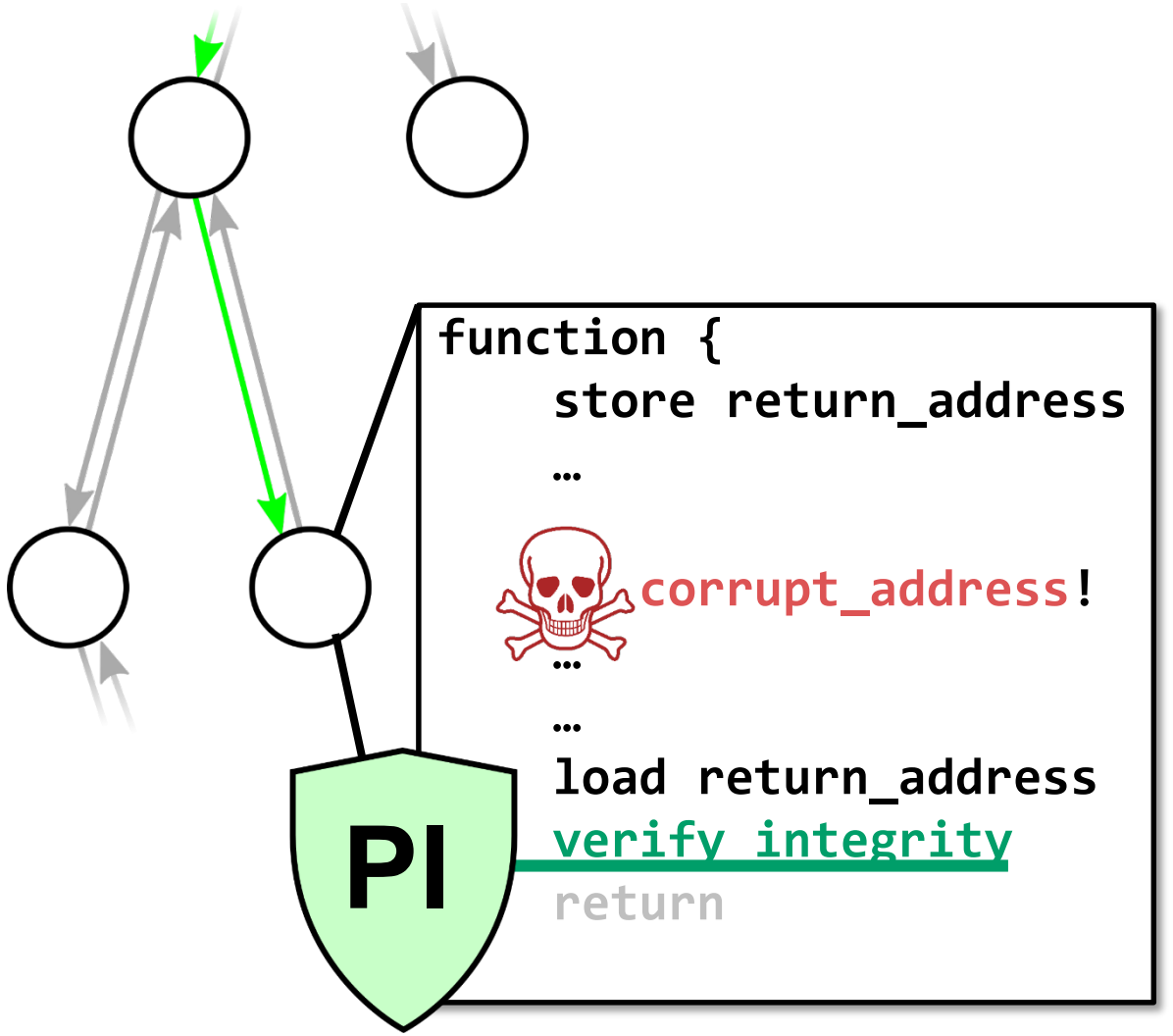
- **Performance overhead**
  - E.g., Softbound, Intel MPX
- **What is the "intended object"?**
  - E.g., bounds of an array iterator?

Szekeres et al. "Eternal War in Memory" IEEE S&P 2014
Nagarakatte et al. "SoftBound: Highly compatible and complete spatial memory safety for C", ACM PLDI 2009
Oleksenko et al. "Intel MPX Explained: A Cross-layer Analysis of the Intel MPX System Stack", ACM POMACS 2018

Pointer Integrity (PI): memory safety for pointers

Ensures that a pointer at use time is the same as at creation time

- **Code pointer integrity implies CFI**
  - CF attacks rely on pointer manipulation
- **Data pointer integrity**
  - reduces data-only attack surface
  - prevents all known Data-Oriented Programming (DOP) attacks



Kuznetsov et al. "Code-Pointer Integrity", USENIX OSDI 2014

Our goal: Realizing PI in practice

Use ARM v8.3-A Pointer Authentication (PA)

But, PA is vulnerable to pointer reuse!

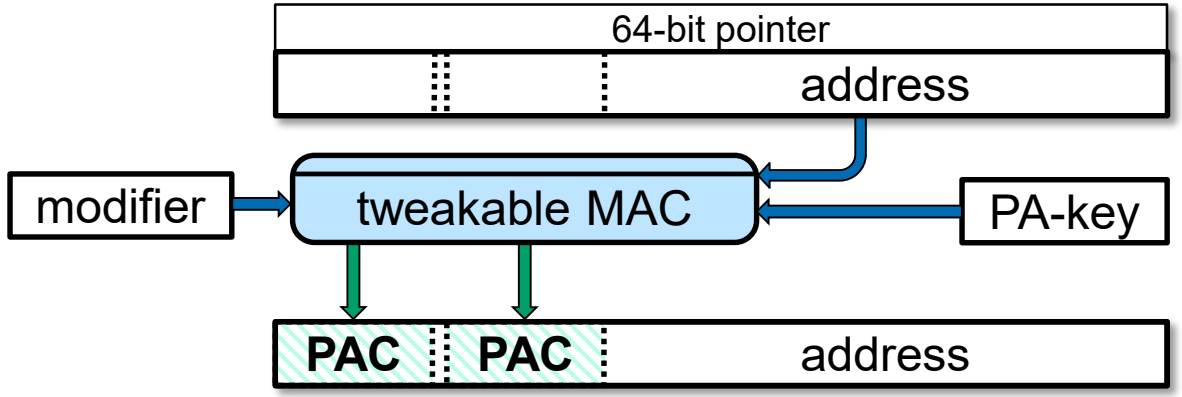
Design PA-assisted Run-time Safety (PARTS)

- Return address signing ≈ backward-edge CFI
- Code pointer integrity ≈ forward-edge CFI
- Data pointer integrity ≈ data-flow integrity for pointers
- Mitigates pointer reuse with run-time type safety

ARM 8.3-A Pointer Authentication

Pointer Authentication Codes (PAC)

- Tweakable MAC
- Set in unused bits of virtual address



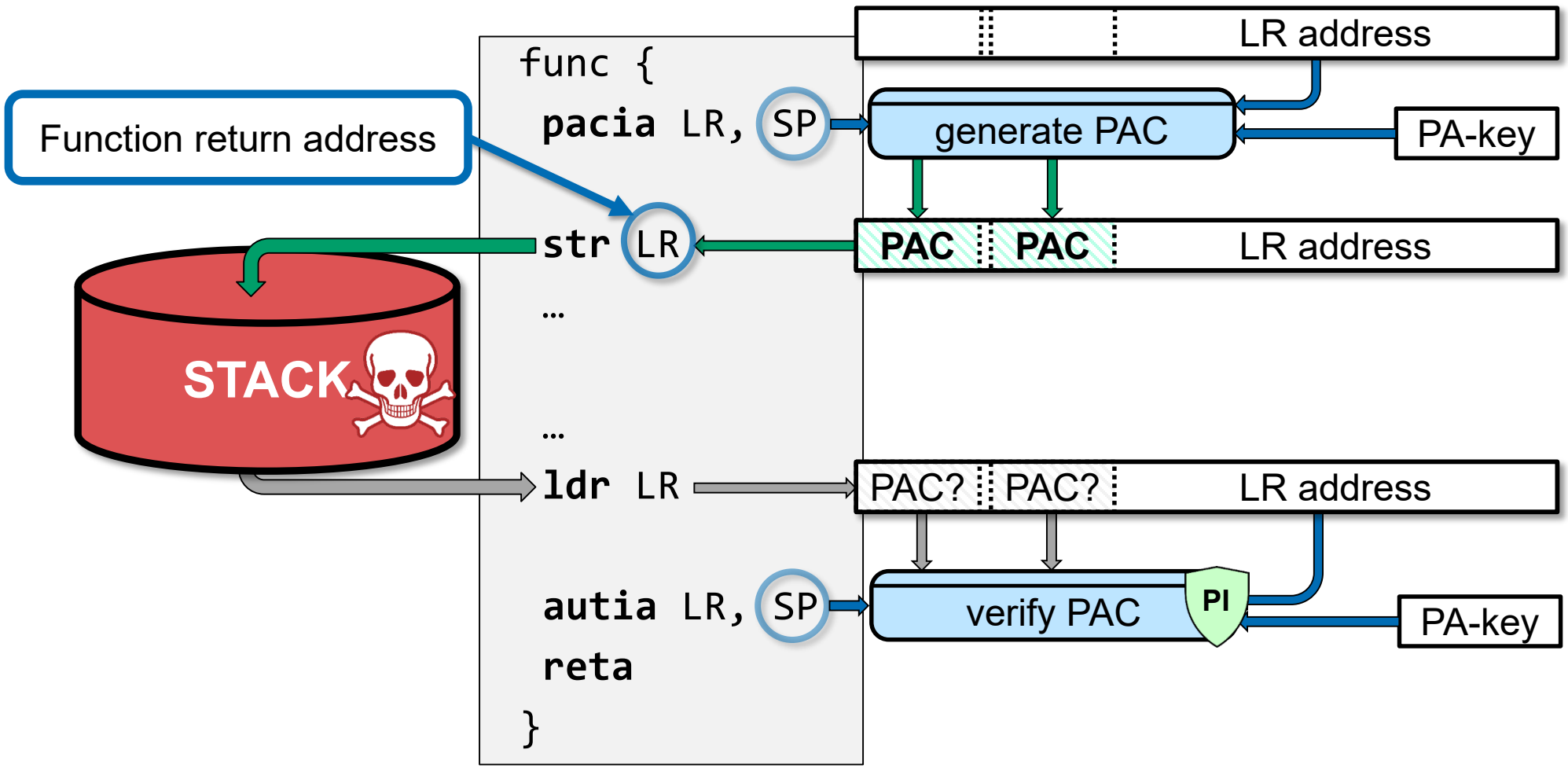
Key/configuration set at higher privilege level

Instrument with new PAC handling instructions

- Opcode determines used key
- Operands set PA modifier (tweak value)

instructions	Code-key		Data-key		Gen.-key
	A	B	A	B	
pacia	X				
pacib		X			
pacda			X		
pacdb				X	
pacga					X
autia	X				
autib		X			
autda			X		
autdb				X	

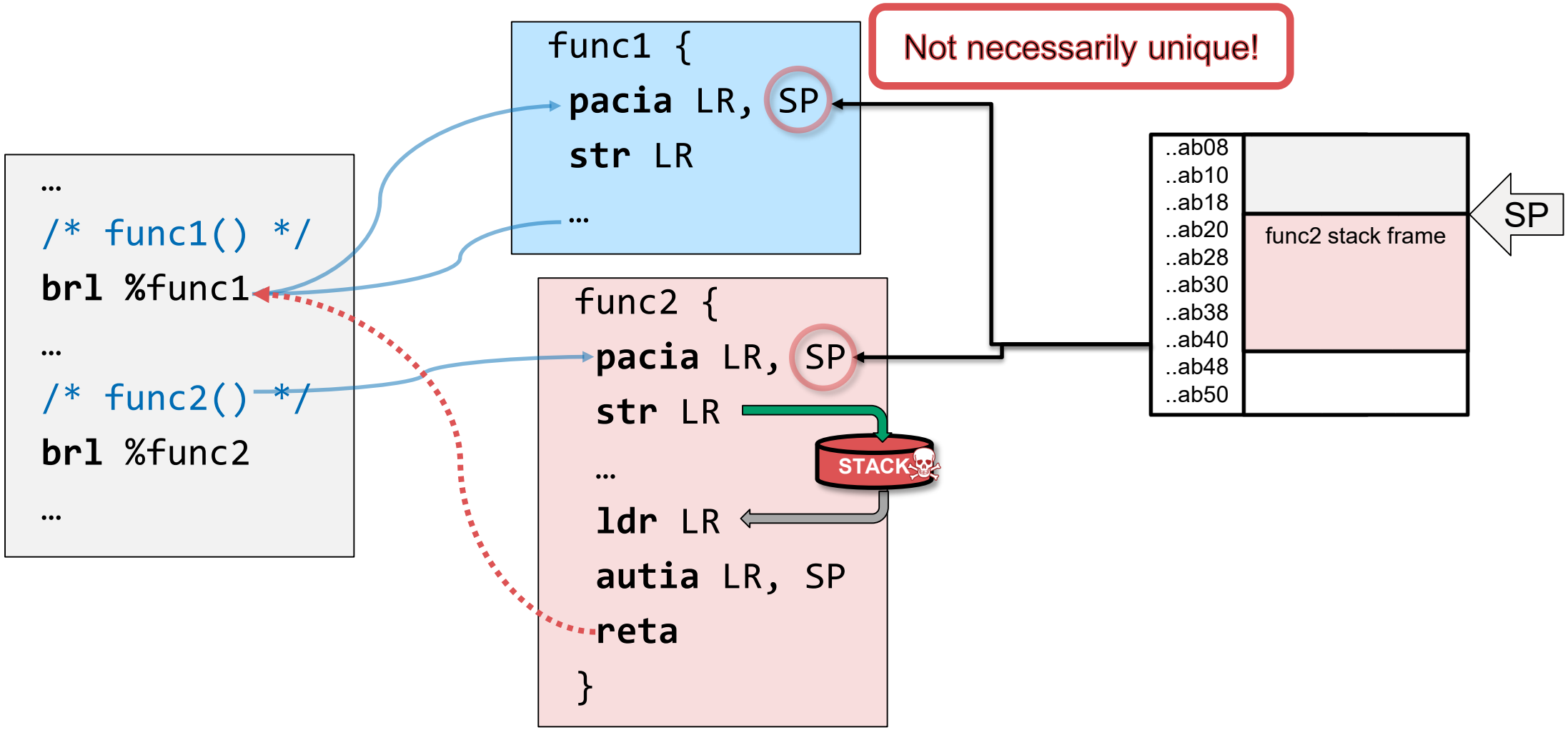
Example: PA-based return address signing



Qualcomm "Pointer Authentication on ARMv8.3", whitepaper 2017

PA only approximates fully-precise pointer integrity

Adversary may re-use PACs



Hardening return address signing

Modifier: function-id + SP value

- Function-id assigned at compile-time
- Prevent cross-function return address reuse

Future additions

- Combine with SP randomization

```
func {
 mov Xm, SP
 mov Xm, #f_id, #lsl_16
 pacia LR, Xm
 str LR
 …
 …
 ldr LR
 mov Xm, SP
 mov Xm, #f_id, #lsl_16
 autia LR, Xm
 reta
}
```

Mashtizadeh "Cryptographically Enforced Control Flow Integrity", ACM CCS 2015

Code pointer integrity

Modifier: type-id

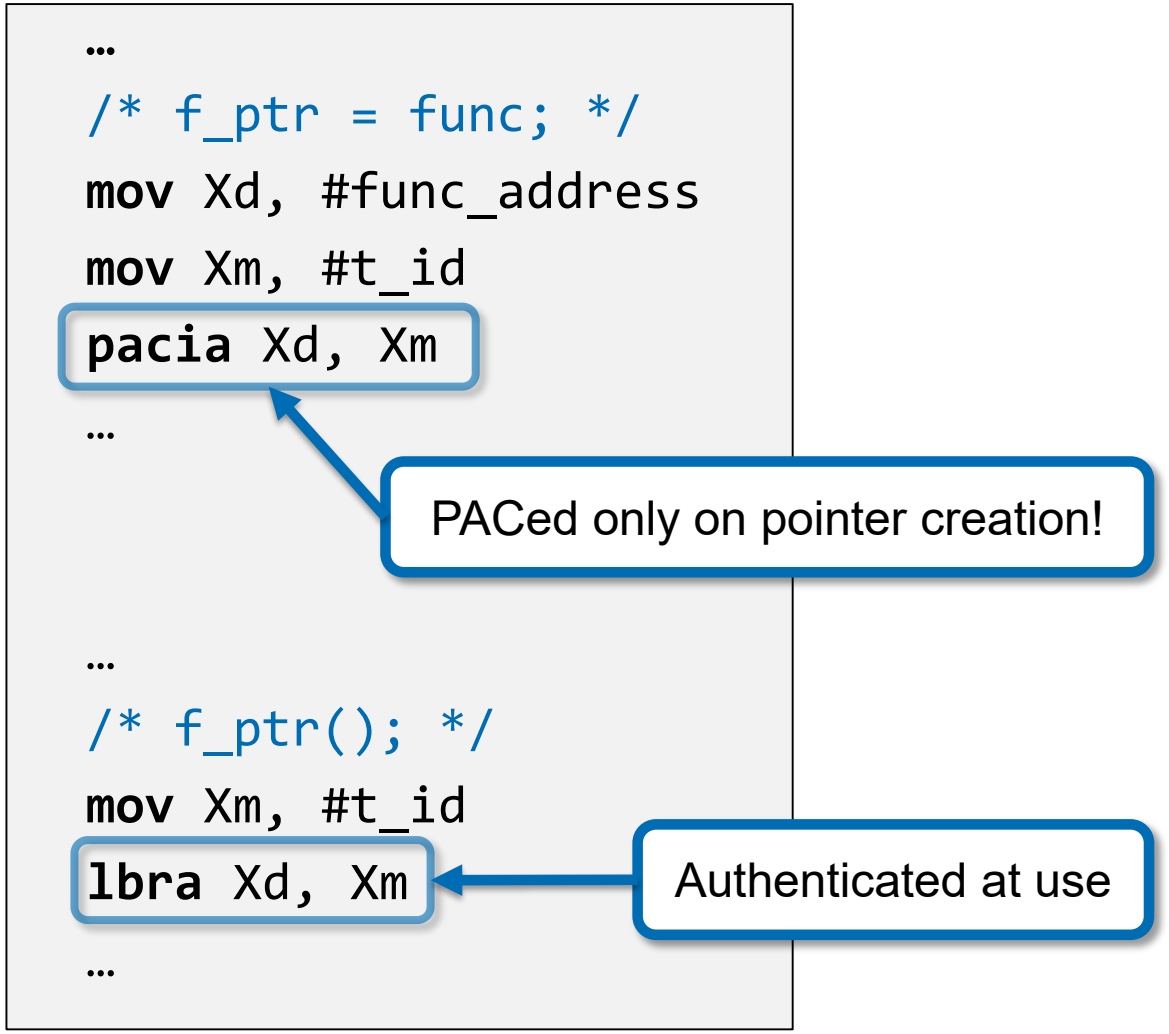
- Assigned at compile-time
- Based on LLVM ElementType
  - ≈ function signature

Uses on-use authentication

- With combined auth+branch instr.

Future additions

- Integrate with software fault isolation
- Storage-based modifiers



Data pointer integrity

Modifier: type-id

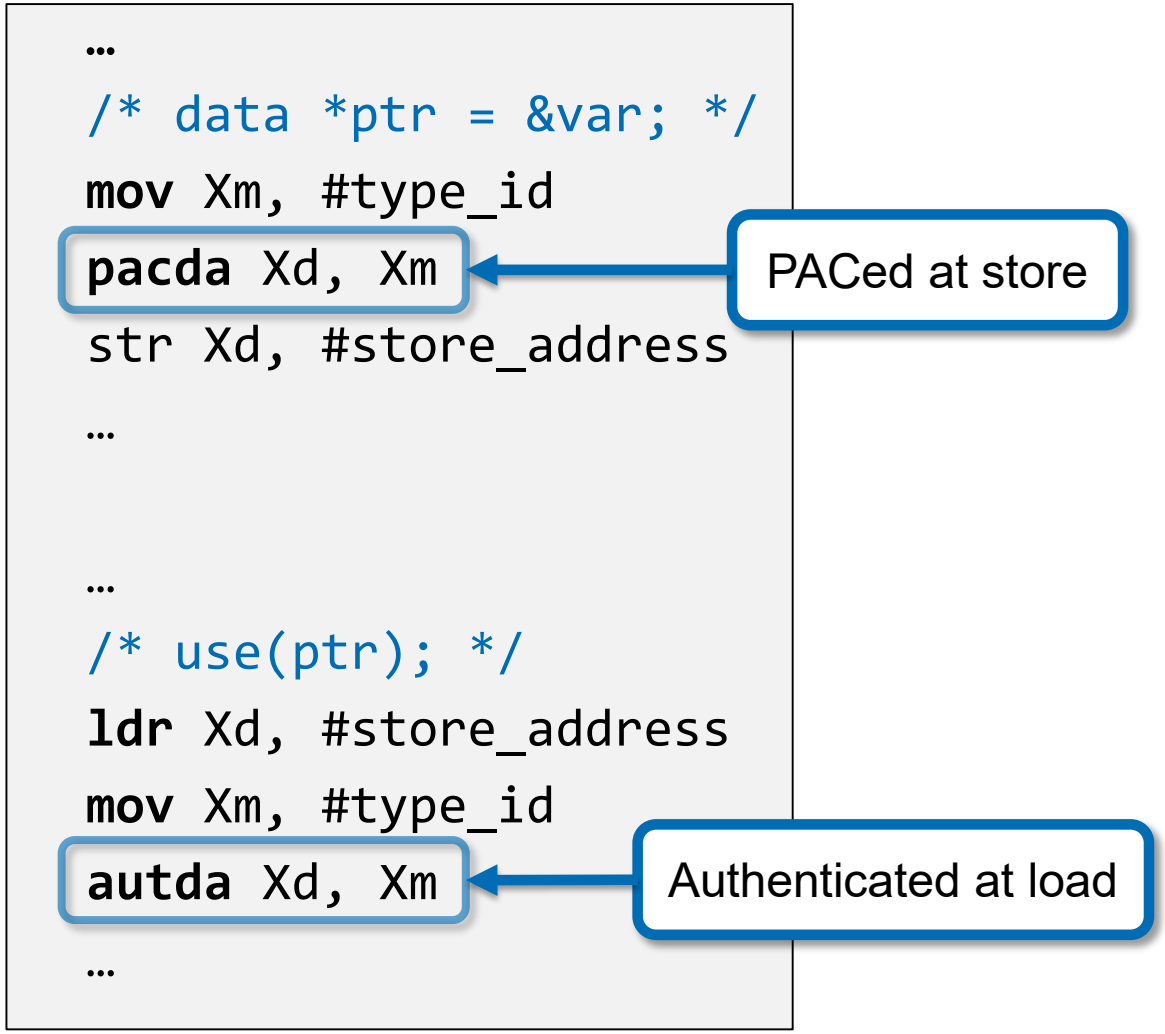
- Assigned at compile-time
- Based on IR ElementType
  - ≈ data type

Uses on-load authentication

- always auth on load

Future additions

- Integrate with software fault isolation
- Storage based modifiers



Summary

ARMv8.3-A PA used with appropriate modifiers provides effective runtime protection

Next steps:

- estimate prevalence of pointer reuse scenarios in real-world programs
- evaluate performance on real hardware
- extend to C++
- handle pre-forked/multithreaded programs
- (how) can we use PA towards achieving full memory safety?