ACCESS SOFTEK, INC
WWW.SOFTEK-TOOLCHAINS.COM

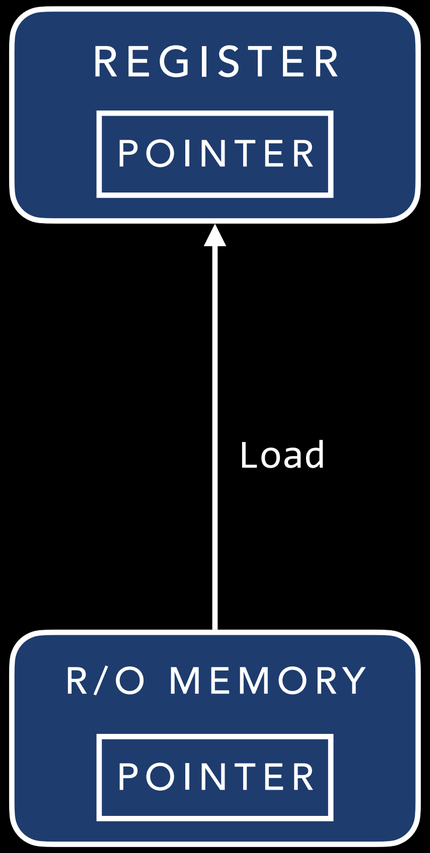
POINTER AUTHENTICATION ABI + ELF-BASED PLATFORMS

ANTON KOROBEYNIKOV

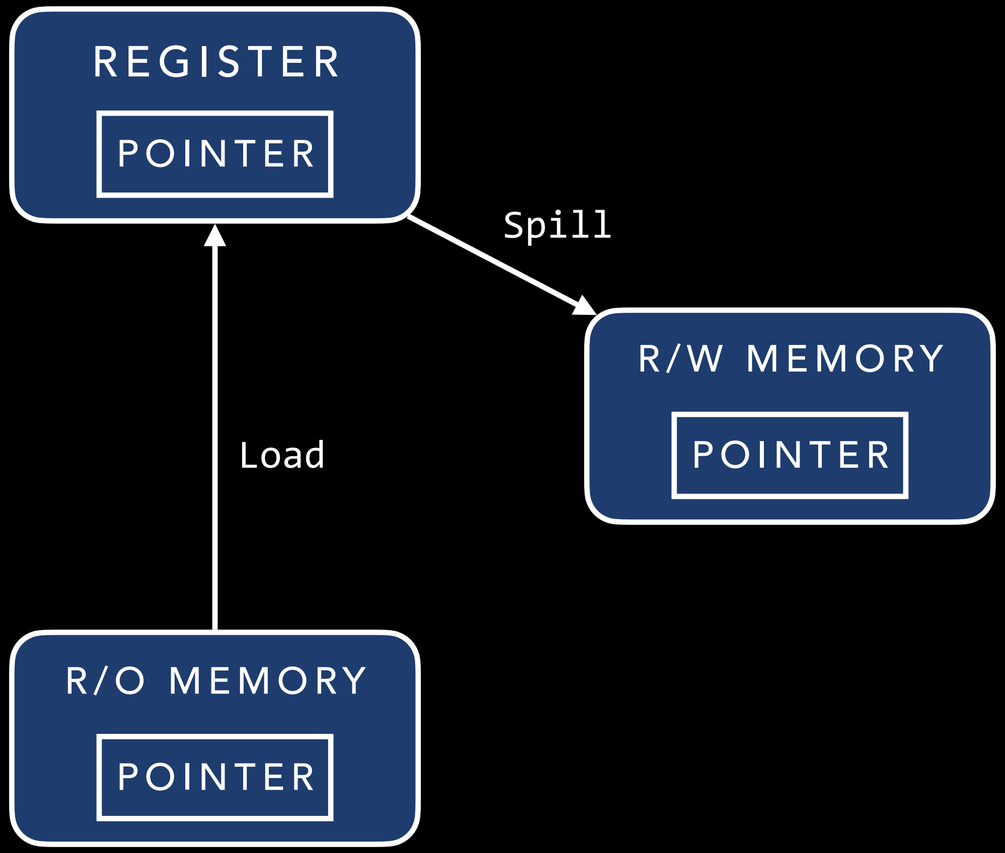
CODE POINTERS AND THEIR LIFECYCLES

CODE POINTERS AND THEIR LIFECYCLES

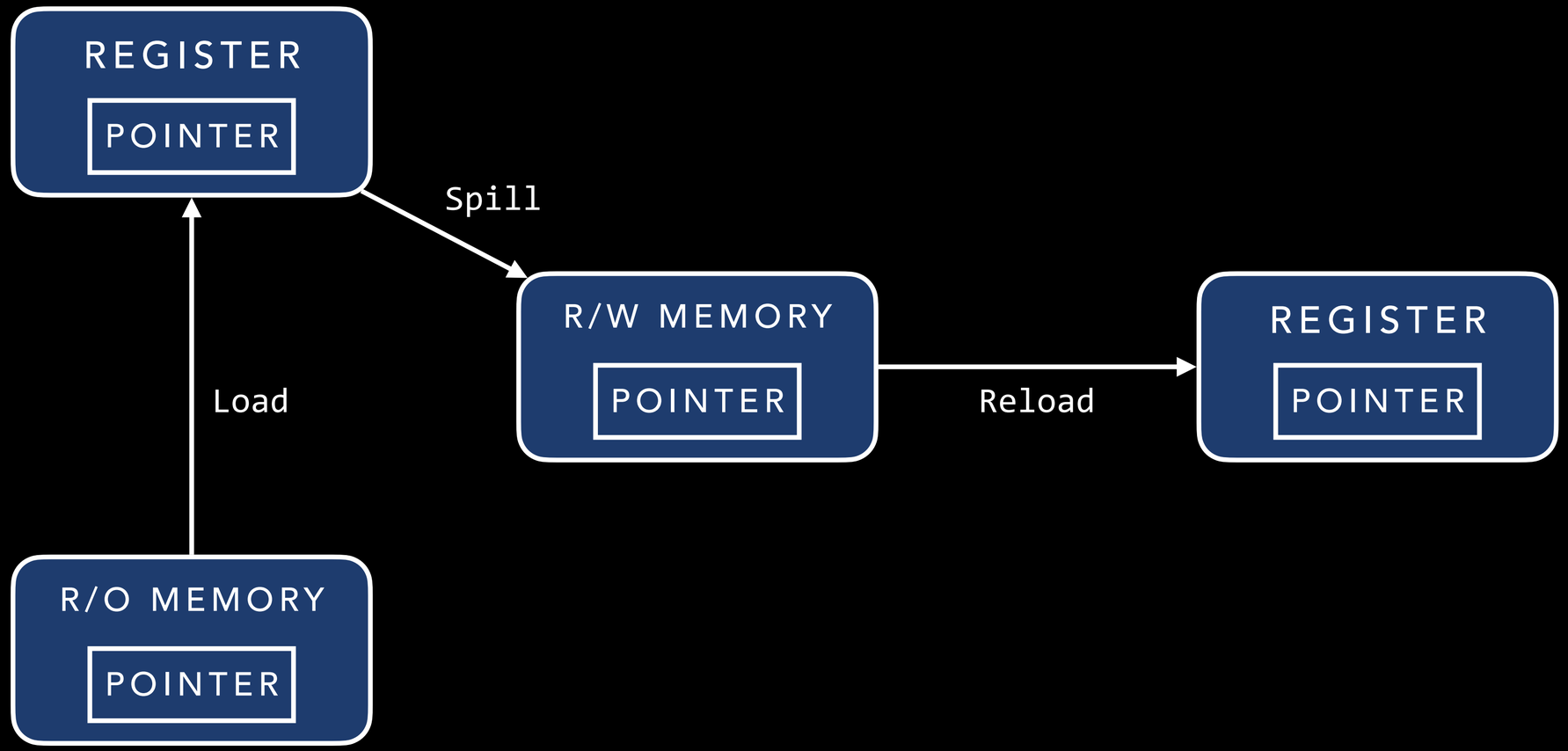
CODE POINTERS AND THEIR LIFECYCLES



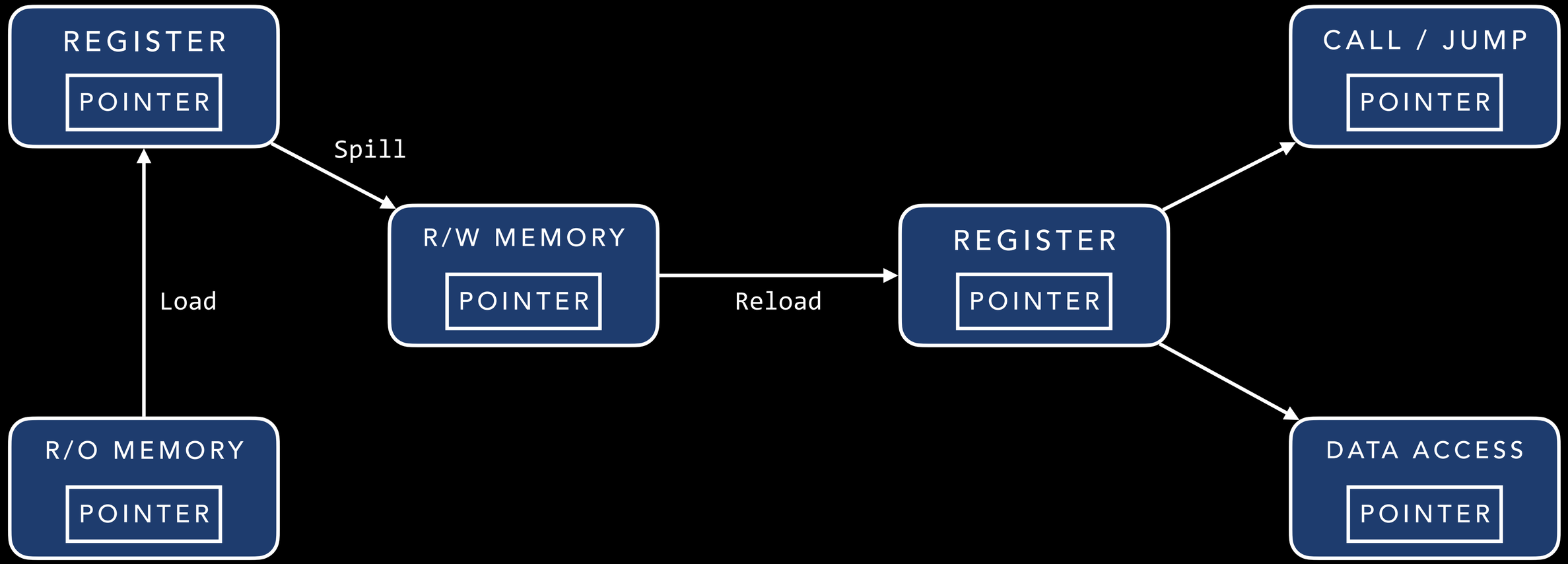
CODE POINTERS AND THEIR LIFECYCLES



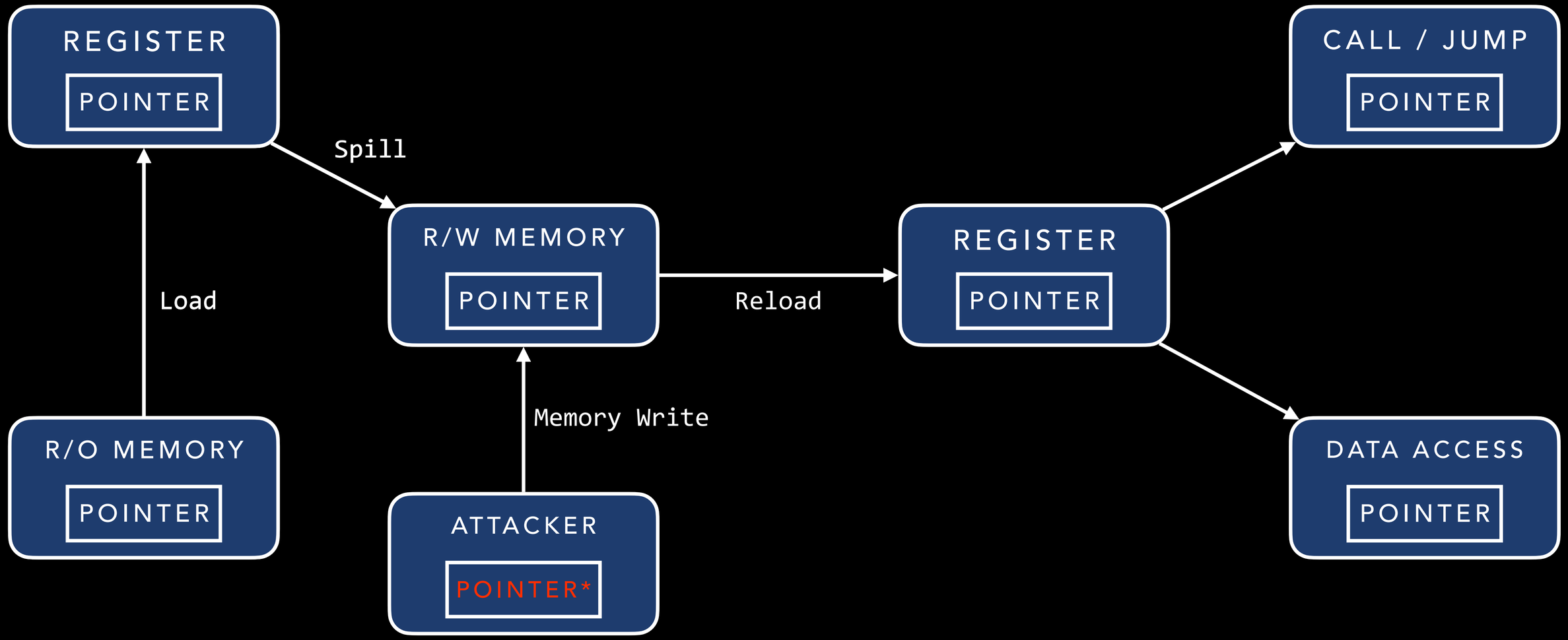
CODE POINTERS AND THEIR LIFECYCLES



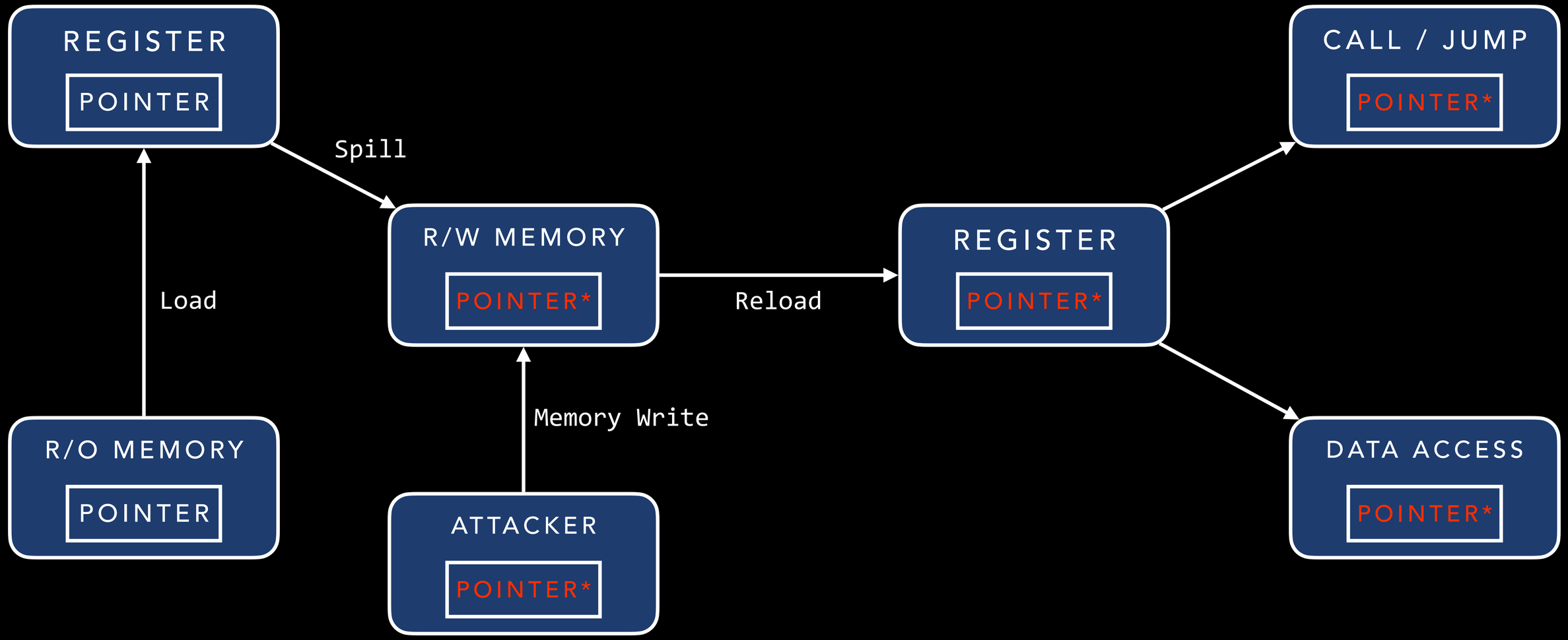
CODE POINTERS AND THEIR LIFECYCLES



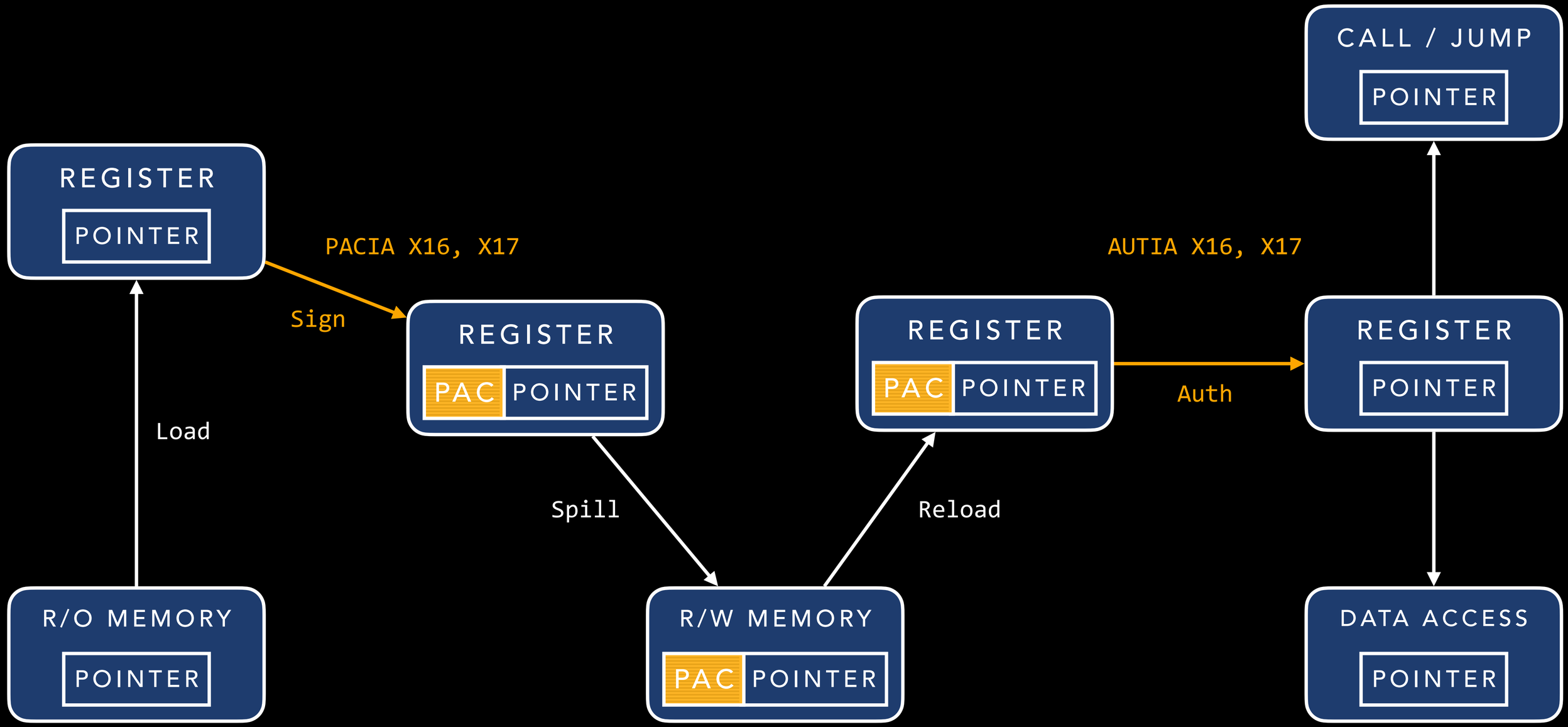
SUBSTITUTION ATTACKS



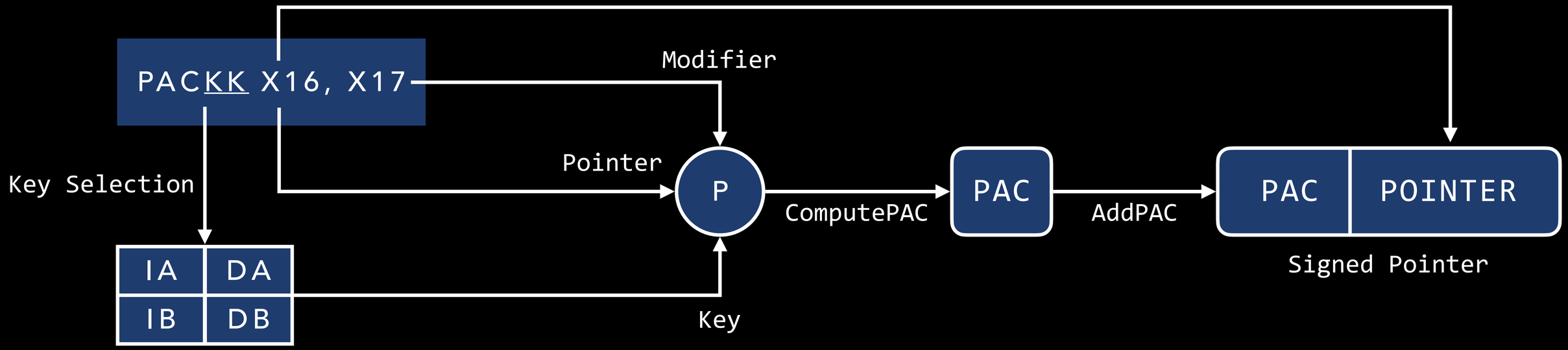
SUBSTITUTION ATTACKS



POINTER AUTHENTICATION



HOW DOES IT WORK?



POINTER AUTHENTICATION: ISA

- Armv8.3: signed pointers
  - Additional changes in Armv8.5 and Armv8.6
- Overall: 48 instructions
  - Some of them in HINT space: nops are on older CPU cores

ISA VS HIGH-LEVEL-LANGUAGES

PACxx

AUTxx

XPACxx

ISA VS HIGH-LEVEL-LANGUAGES

PACxx

AUTxx

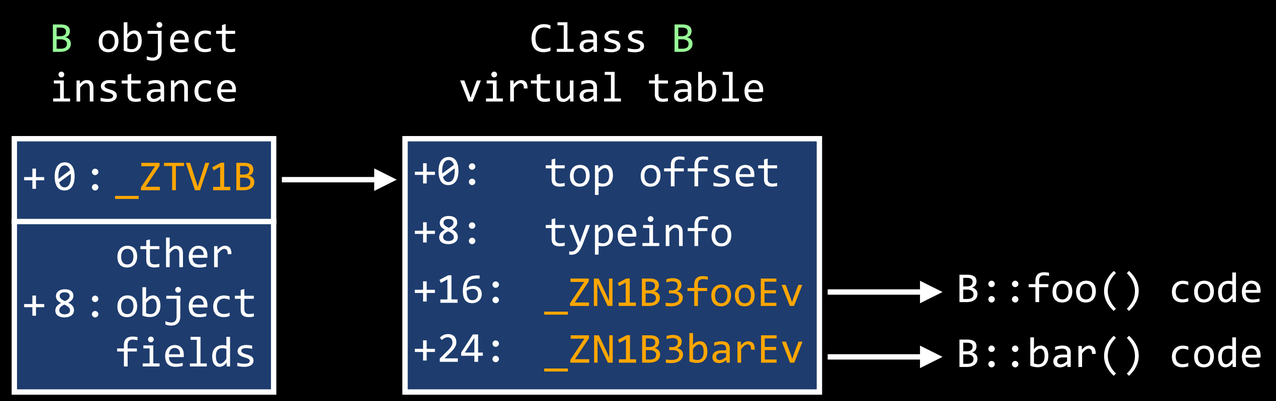
XPACxx

vs

```
class B : public A {
public:
    void foo() const override;
    void bar() const override;
};

int test(const A &obj) {
    obj.bar();
    return 0;
}
```

ISA VS HIGH-LEVEL-LANGUAGES



POINTER AUTHENTICATION C++ ABI

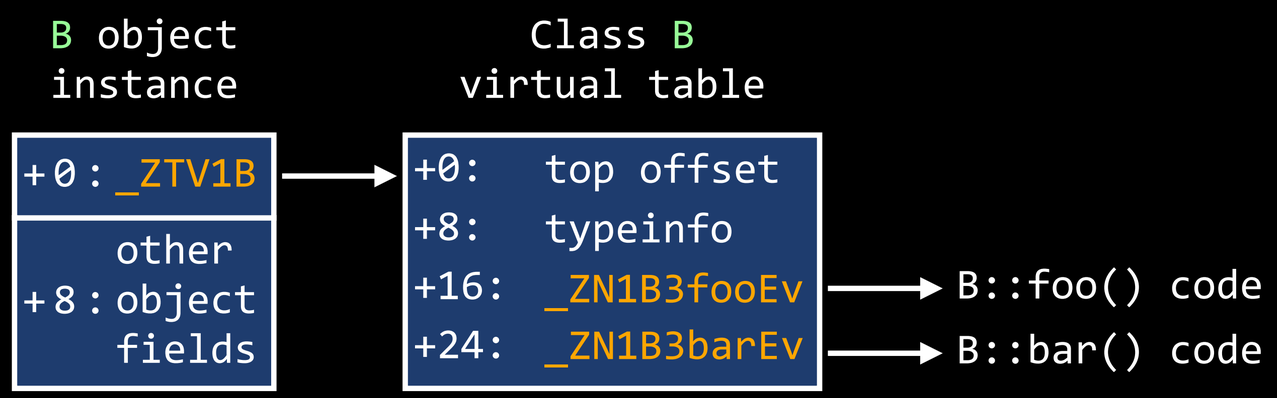
- Developed by Apple to use on Mac hardware
- Presented on LLVM US Developers Meeting 2019
  - arm64e: An ABI for Pointer Authentication – *Ahmed Bougacha, John McCall*
- Available as arm64e architecture since Apple A12 (~2018)
- Until recently was only available in Apple downstream clang fork.

https://github.com/swiftlang/llvm-project/blob/next/clang/docs/PointerAuthentication.rst

ELEMENTS OF C++ PAUTH ABI

Sources of indirect branches:

- returns
- switches
- symbol imports (GOT)
- C function pointers
- C++ virtual functions
- C++ member pointers
- static object constructors
- computed gotos
- …



ABI rule (signing scheme) specifies key and how to compute the modifier

DIVERSITY

- Armv8.3 allows arbitrary 64-bit modifiers to be used
- Address diversity: can use storage address of a pointer as modifier
  - Copying of pointer requires resigning
- Semantic diversity: can use semantics of the pointed code/data as a modifier
  - Derived from declaration, type, or particular usage …
- Top 16 bits of address is usually reserved
  - Can combine address and semantic diversity (small *constant discriminator*)

PAUTH + ELF

- PAuth ABI Extension to ELF for the Arm® 64-bit Architecture (*pauthabielf64*)
  - Currently in Alpha state
  - https://github.com/ARM-software/abi-aa/blob/main/pauthabielf64/pauthabielf64.rst
- Set of ELF-specific relocations & relocation operations
- Marking schema
  - Entry in `.note.gnu.property` section
- Platform decisions
  - RELRO GOT
  - PLT GOT signing

LLVM 19: STATUS

- The majority of required frontend patches were ported from Apple clang
  - Thanks to Ahmed, John, Akira, Oliver et al!
- Generic pauth codegen support ported from Apple clang
- Implemented ELF-specific pauth codegen
- Implemented object file & linker support for pauth relocations
- Added experimental `pauthtest` ABI for AArch64/Linux

LLVM testsuite passes on AArch64/Linux with `pauthtest` ABI

PAUTHTEST

- ABI to test pointer authentication on AArch64 Linux
- Mostly follows Apple arm64e for signing schema, etc.
- Enable via `-mabi=pauthtest` or `pauthest` environment in target triple (e.g. `aarch64-gnu-linux-pauthtest`)
  - `-mabi` is normalized to environment in triple, similar to `eabihf` on 32-bit ARM.
- ELF marking:
  - “LLVM Linux” test vendor (`0x10000002`)
  - Encodes entire signing schema in place of version making catching mismatches easier
- Requires pauth-enabled standard library

KNOWN ISSUES

- Few corner cases with no-op casts (mostly around `noexcept`)
- No diagnostics for some unsupported computed gotos (code will crash though)
- libunwind might materialize pointers in not very secure way across exceptions being thrown
- Enabling pauth for platform is manual: no hooks for signing schema, etc.

TBD

- PRs under review
  - Signed GOT support
  - TLS support
- `__ptrauth` qualifier
  - See RFC: https://discourse.llvm.org/t/rfc-ptrauth-qualifier/
- Optimizations & relaxations
  - Some already available in Apple downstream tree
- Some code unification & cleanup
- Documentation!

NEW PLATFORM

- Pointer Authentication ABI is an ABI:
  - Cannot safely mix code that uses different ABIs
- Could be deployed on bare-metal platform as default ABI
- Could be deployed on commodity platform as an isolated part
  - OS kernel
  - System-critical process

COMPONENTS OF PAUTH SUPPORT

- Kernel support
- Signing schema
- Marking and versioning
- Runtime libraries

KERNEL SUPPORT

- Key allocation & management
  - Set of 5 128-bit keys
  - Some of them are expected to be process-dependent
- Key preservation during context switches
- Protect values of X16 / X17
- Do you have fork()?

SIGNING SCHEMA

- Lots of variations and customizations in signing scheme:
  - Discriminator usage
  - Objects to sign, etc.
- Every variation effectively defines a new ABI
- Cannot safely mix code that uses different ABIs
- Default set (aka `arm64e` or `pauthtest`) is good enough to start
  - Decide about function pointer type discrimination

Do you want to expose all options and knobs that may change ABI?

VERSION & MARKING

https://github.com/ARM-software/abi-aa/blob/main/pauthabielf64/pauthabielf64.rst#elf-marking

- Uses `.note.gnu.property` section
  - Need to define platform
  - Need to define ABI version
- Might want to encode some additional metadata in the version
- See if base compatibility mode is enough for platform's purposes

Both static linker and dynamic loader are expected to check ABI compatibility

LIBRARIES

- Compiler runtime:
  - libc++, libunwind and compiler-rt should work out of the box
- Dynamic loader:
  - Support for pauth relocations
  - Shared libraries, lazy symbol resolution and interop with unsigned code
  - GOT and PLT GOT choices
- C standard library:
  - Static constructors / destructors (including e.g. `atexit`)
  - `setjmp / longjmp`
  - signals

Can use musl+pauth PoC implementation from https://github.com/access-softek/musl/

TESTING

- Testing security-related code is non-trivial
  - Matching for expected code sequences
  - Checking final binaries
- Crashing testsuite
  - Try to perform a pointer substitution in different contexts
  - PAuth works: substitution fails => test crashes
  - PAuth does not work: substitution success => no crash
  - WIP, to be released soon
- Maybe a way to extend BOLT-based analysis tools?
  - EuroLLVM 2024: Does LLVM implement security hardenings correctly? A BOLT-based static analyzer to the rescue? – *Kristof Beyls*

Q & A