Pointer Authentication

What Compiler Implementers and Language Designers Need to Know

Oliver Hunt, LLVM Team, Apple

Agenda

Protecting control flow integrity

Pointer authentication basics

Pointer authentication in Clang and Swift

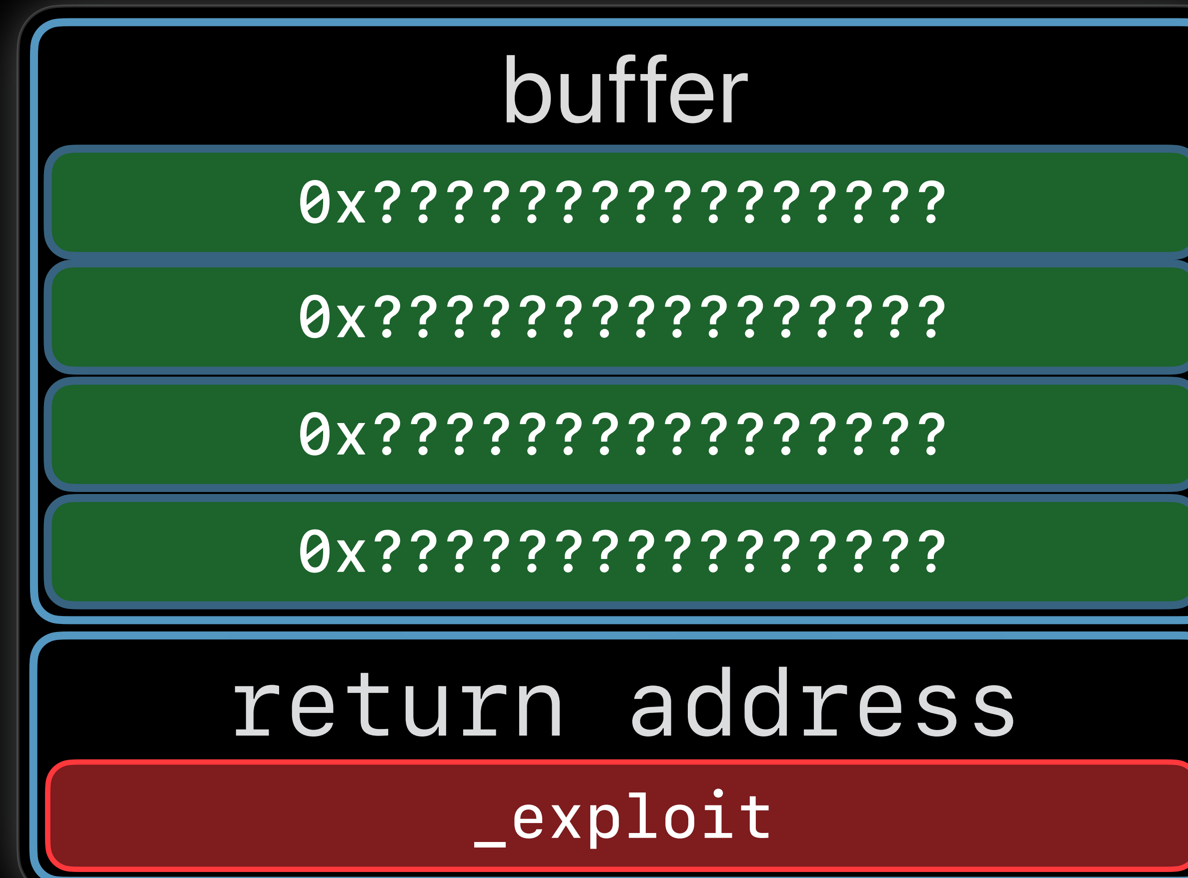
Supporting pointer authentication in your compiler

Designing features to support pointer authentication

Hijacking the return address

- Buffer overflow enables attack to overwrite return address
- This violates control flow integrity
- On return the attacker is in control

```c
void f(char *src, int sz) {
  char buffer[4];
  memcpy(buffer, src, sz);
}
```

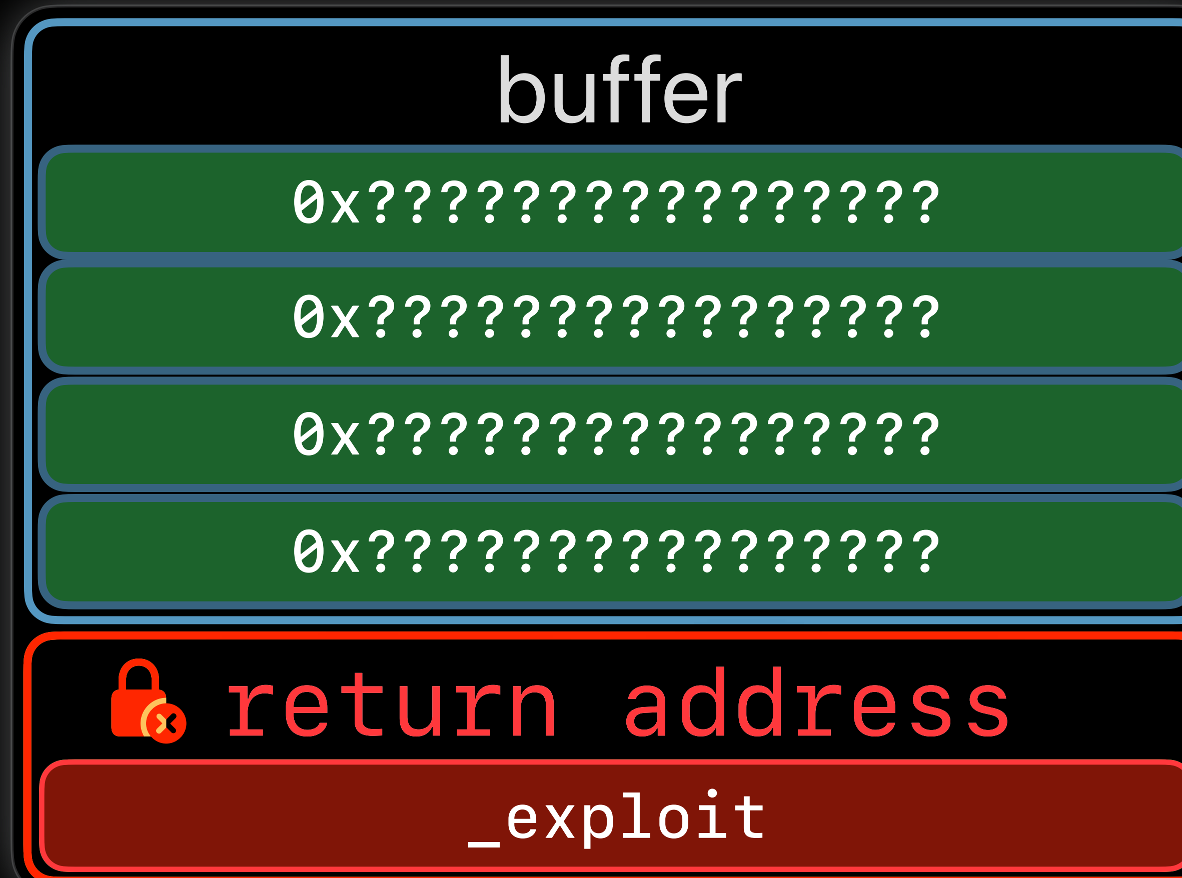


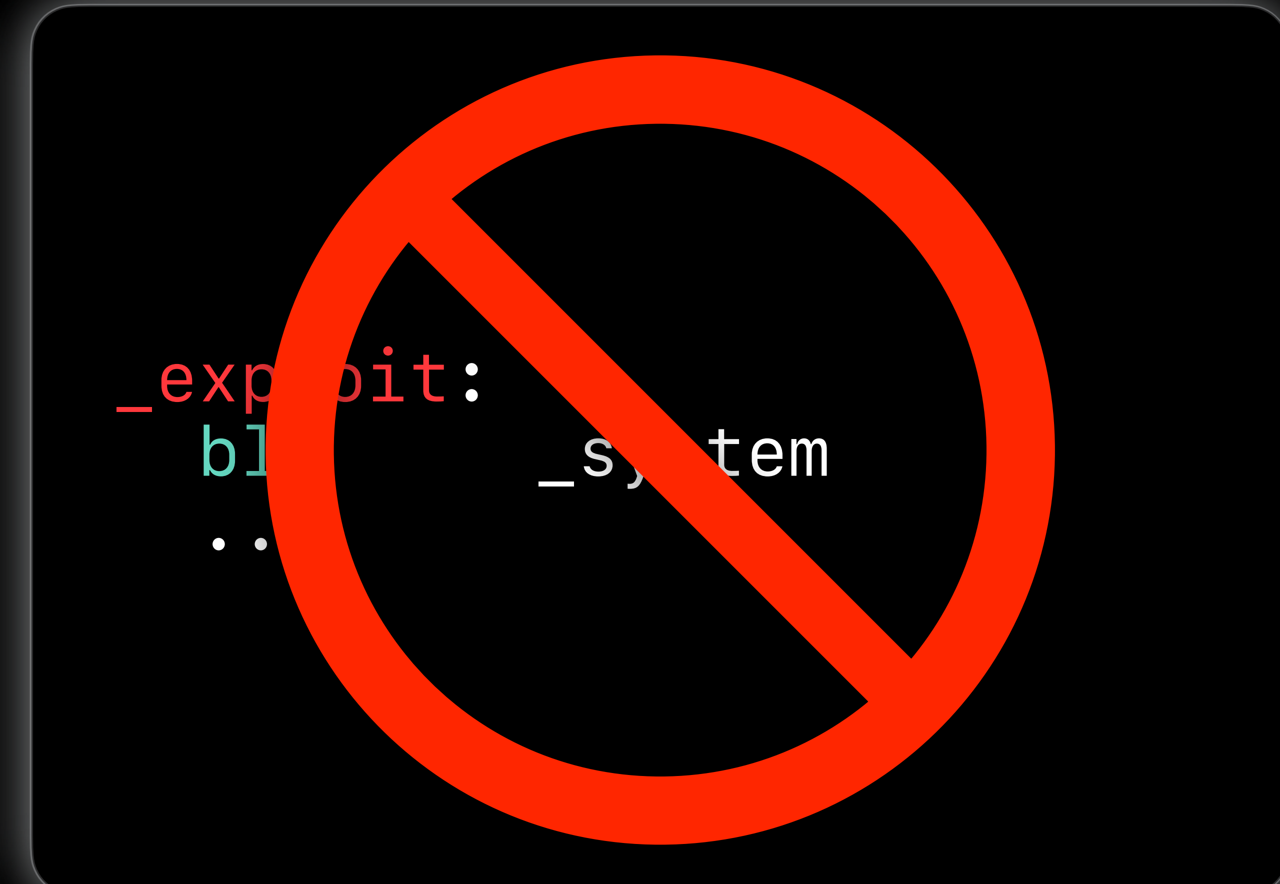

```
_exploit:
  bl        _system
  ...
```

Pointer authentication protects control flow

- Hardware supported control flow integrity
- Embeds a cryptographic signature into high bits of pointers
- Signature validated prior to any future use

```
void f(char *src, int sz) {
  char buffer[4];
  memcpy(buffer, src, sz);
}
```





Safe languages require pointer authentication too

- Programs written in safe languages still rely on unsafe library code
- Memory safety bugs in unsafe code can corrupt data structures in safe code
- Must protect control flow integrity of safe languages

Pointer authentication in Clang and Swift

- Apple implemented extensive language support in all system languages
  - C, C++, Objective-C, and Swift
- Supported pointer authentication since iPhone Xs and all Apple silicon Macs
- Deployed across kernel- and userspace
- Backend and language support has been upstreamed to llvm.org

Not just an Apple technology

- Introduced in ARMv8.3 and is supported by an increasing number of CPU vendors across the industry
- Linux and Windows ARM64 support pointer authentication to protect call stack
- Support for advanced protection schemas being implemented in Linux
- Pointer authentication is becoming a baseline security expectation across the ARM ecosystem

Goals for this talk

- Convince you to implement support for pointer authentication in your language and compiler
- Explain how to design language features with compatibility for pointer authentication

Why your language needs pointer authentication

- Pointer authentication is a process wide ABI, so applications can only use your language if it also supports that ABI
- Unsafe code can undermine safety guarantees in safe languages
- Match the protection level of other languages in the process

Adopt pointer authentication in your language

- Enable basic pointer authentication support in LLVM
- Protect runtime data structures
- Design language features with pointer authentication in mind

Enable basic pointer authentication support in LLVM

- LLVM already provides the basic pointer authentication ABI
- Make pointer authentication instructions and intrinsics available
  - Include `+pauth` feature in TargetMachine configuration
- Make LLVM use platform pointer authentication ABI
  - Attach `ptrauth-returns` and `ptrauth-calls` attributes to llvm::Function definitions

Protect runtime data structures

- Programming languages and their compilers are responsible for protecting their own languages features
- Identify what data needs to be protected
- Design authentication schemes to protect that data
- Implement those schemes

Identify what needs to be protected

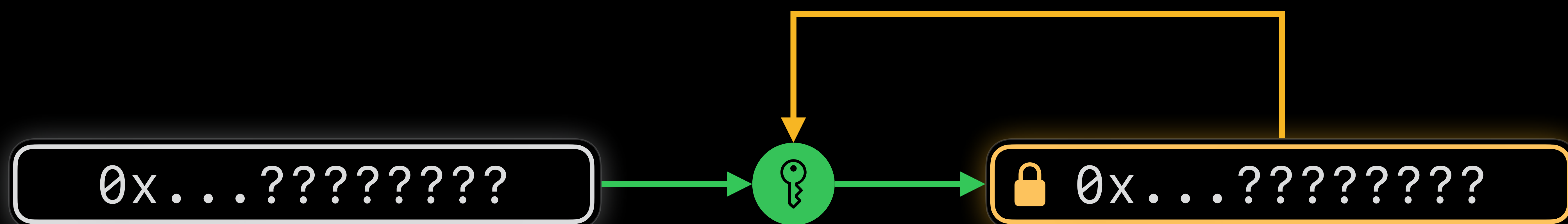
- Focus on features that impact control flow
- Critical to protect all forms of dynamic dispatch: function pointers, virtual functions, existential type tables, etc.
- Languages with rich runtime metadata need to protect information that can influence type and object behavior

Design authentication schemes to protect that data

- Any form of pointer authentication prevents an attacker from corrupting a pointer
- Attackers now have to copy and reuse validly signed pointers in invalid places
- Compilers can provide a custom discriminator for signing and authentication
- Compilers need to use different authentication schemes for different data to prevent pointer reuse attacks

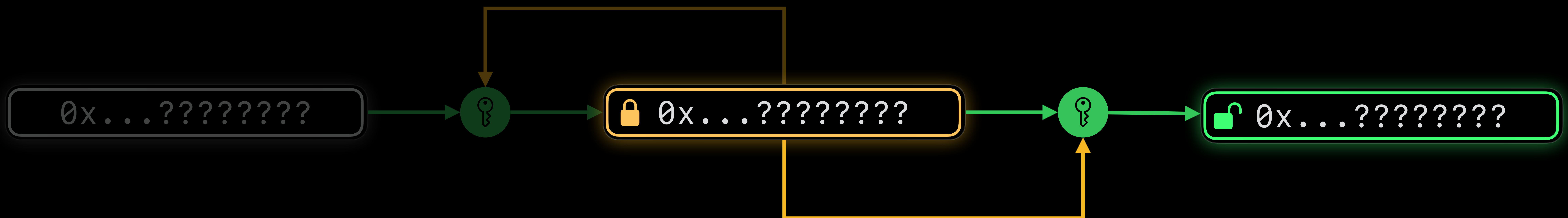
Address discrimination

- Special case is address discrimination
- Compiler uses the storage address of a pointer as discriminator



Address discrimination

- Special case is address discrimination
- Compiler uses the storage address of a pointer as discriminator
- Signature is only valid when the pointer is stored at that specific location in memory
- An attacker can no longer copy the pointer at all



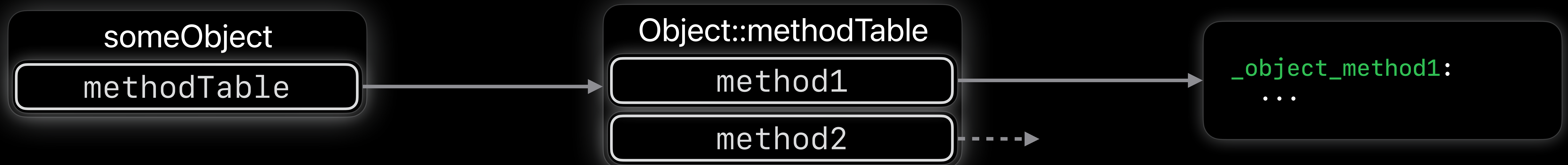
Trade offs in authentication schemes

- Choice of discriminator impacts the security properties of the signature
- Strongest options may not be compatible with existing language features or code
- Address discrimination provides tremendous security value but is incompatible with memcpy semantics

Protect dynamic dispatch

Protect class based polymorphism

- Object instances contain pointers to class metadata or method table
- Structures like this are powerful attack vectors and should be authenticated

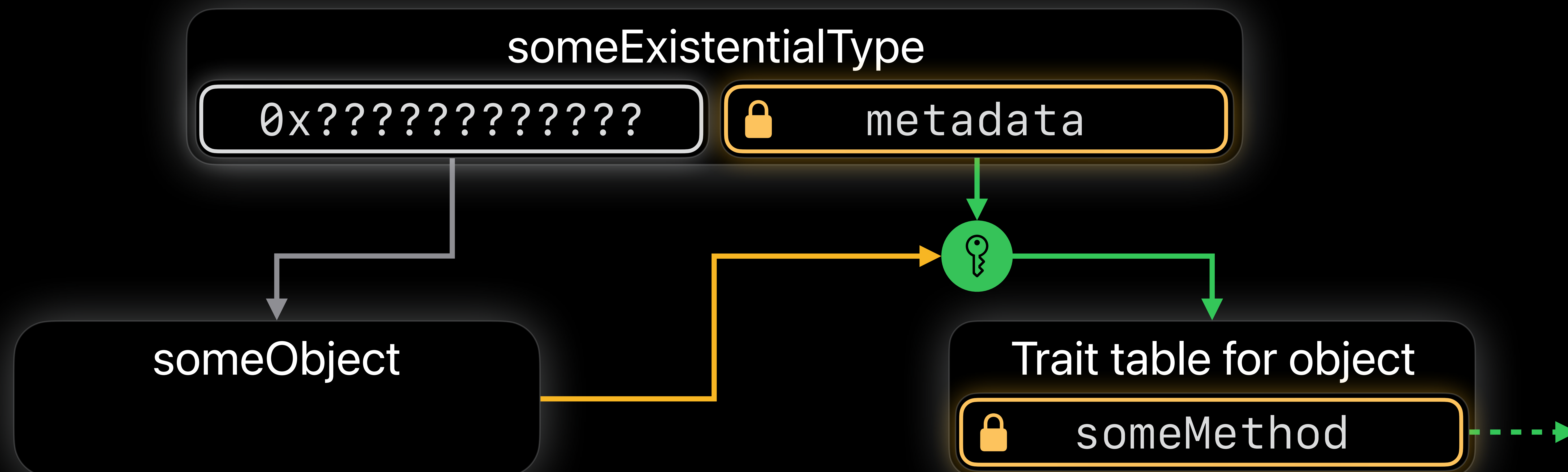


Protect class based polymorphism

- Object instances contain pointers to class metadata or method table
- Structures like this are powerful attack vectors and should be authenticated
- Use pointer authentication to protect each link in dispatch chain
  - Authentication should incorporate identity of owning object

Protect existential types

- Implemented as tuple of an object and metadata pointer
- Compromising the metadata is as powerful as class based polymorphism
- Even though separate values, authentication can still include object address



Implement your authentication schemes

- LLVM has all you need
- The `llvm.ptrauth.sign` and `llvm.ptrauth.auth` intrinsics do the heavy lifting
- The `llvm.ptrauth.blend` intrinsic lets a compiler combine constant and dynamic discriminators
- Compilers should not update a signature by authenticating and then signing the pointer. Use the `llvm.ptrauth.resign` to this safely and automatically

Do not design to require memcpy semantics

Design features with pointer authentication in mind

- Do not design to require memcpy semantics
- Avoid designs that require pointer tagging to be implemented efficiently
- Features can never require a compiler to generate code that could sign untrusted data
  - Untrusted does not mean from the internet, it means from writable memory

Summary

- Pointer authentication is proven, hardware-backed protection against control flow attacks with industry-wide adoption
- Safe languages also need pointer authentication
- Enable basic pointer authentication provided by LLVM
- Protect runtime data structures to close additional control-flow attack surfaces
- Design language features compatible with pointer authentication

TM and © 2026 Apple Inc. All rights reserved.