Swift / C++ Interoperability

Egor Zhdan

EuroLLVM l Apple, Inc. l Apr 2024

> 50%

of security vulnerabilities are caused by memory safety issues

Swift: Modern Memory-Safe Language

- Natively compiled to Darwin, Linux, Windows, etc.
- Leverages Clang/LLVM CodeGen and infrastructure

```
% find llvm-project/clang/lib -name "*.cpp" | xargs wc -l
```

```
1025676
```

Moving Towards Memory Safety

Goals of Swift / C++ Interoperability

Goals of Swift / C++ Interoperability

- Incremental integration of Swift code into C++ codebases
  - Implement minor chunks of new functionality in Swift
  - Build and test the codebase continuously

Goals of Swift / C++ Interoperability

- Incremental integration of Swift code into C++ codebases
  - Implement minor chunks of new functionality in Swift
  - Build and test the codebase continuously
- Using existing C++ libraries in Swift projects
  - Allow using well-designed C++ APIs from Swift
  - Avoid significant performance penalty

Non-Goals of Swift / C++ Interoperability

Non-Goals of Swift / C++ Interoperability

- ❌ Making every C++ API available in Swift

Non-Goals of Swift / C++ Interoperability

- ❌ Making every C++ API available in Swift
- ❌ Writing “C++ in Swift”

Non-Goals of Swift / C++ Interoperability

- ❌ Making every C++ API available in Swift
- ❌ Writing “C++ in Swift”
- ❌ Changing Swift’s core principles to accommodate C++

Non-Goals of Swift / C++ Interoperability

- ❌ Making every C++ API available in Swift
- ❌ Writing “C++ in Swift”
- ❌ Changing Swift’s core principles to accommodate C++
- ❌ Developing a dialect of Swift or C++

Example

```cpp
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    bool hasAttribute(std::string Kind) const;

}
}
```

```
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    bool hasAttribute(std::string Kind) const;

}
}
```

```
// Attributes.cpp

#include “llvm/IR/InstrTypes.h”

void checkHasAttribute(llvm::AttributeSet attrs) {
    attrs.hasAttribute(“builtin”);
}
```

```
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    bool hasAttribute(std::string Kind) const;

}
}
```

```
// Attributes.cpp

#include “llvm/IR/InstrTypes.h”

void checkHasAttribute(llvm::AttributeSet attrs) {
    attrs.hasAttribute(“builtin”);
}
```

```
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    bool hasAttribute(std::string Kind) const;

}
}
```

```
// Attributes.cpp

#include “llvm/IR/InstrTypes.h”

void checkHasAttribute(llvm::AttributeSet attrs) {
    attrs.hasAttribute(“builtin”);
}
```

```
// Attributes.swift

import LLVM_IR

func checkHasAttribute(attrs: llvm.AttributeSet) {
    attrs.hasAttribute(“builtin”)
}
```

```
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    bool hasAttribute(std::string Kind) const;

}
}
```

```
// Attributes.cpp

#include “llvm/IR/InstrTypes.h”

void checkHasAttribute(llvm::AttributeSet attrs) {
    attrs.hasAttribute(“builtin”);
}
```

```
// Attributes.swift

import LLVM_IR

func checkHasAttribute(attrs: llvm.AttributeSet) {
    attrs.hasAttribute(“builtin”)
}
```

Under The Hood

Clang

Swift

```
// main.swift
import Clang_AST
```

```
% swiftc main.swift
```

Clang

Swift

```
// main.swift
import Clang_AST
```

```
% swiftc main.swift
```

Swift loads Clang_AST module

```
swift::loadModule(“Clang_AST”)
```

Clang

Swift

```swift
// main.swift
import Clang_AST
```

```
% swiftc main.swift
```

Clang loads Clang_AST module

`clang::CompilerInstance::loadModule(“Clang_AST”)`

←

Swift loads Clang_AST module

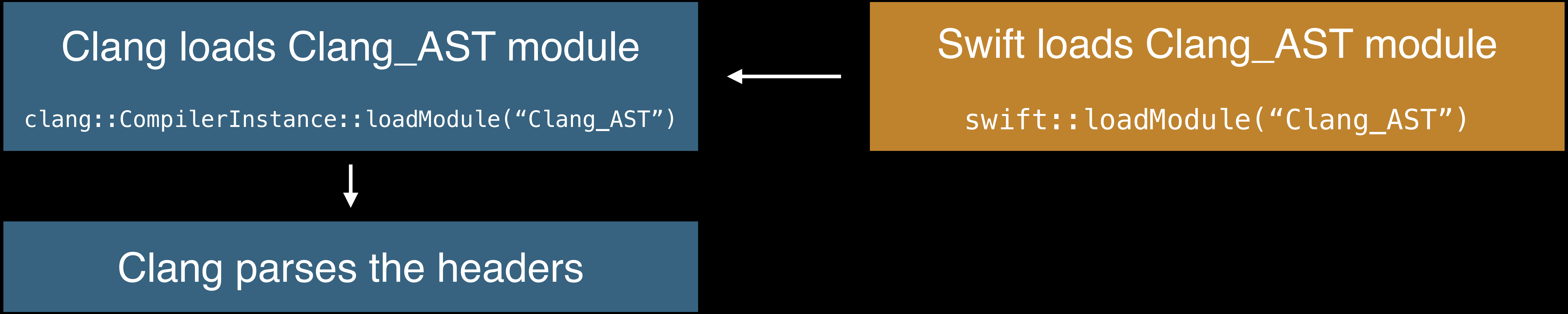
`swift::loadModule(“Clang_AST”)`

Clang

Swift

```
// main.swift
import Clang_AST
```

```
% swiftc main.swift
```



```
swift::loadModule("Clang_AST")
```

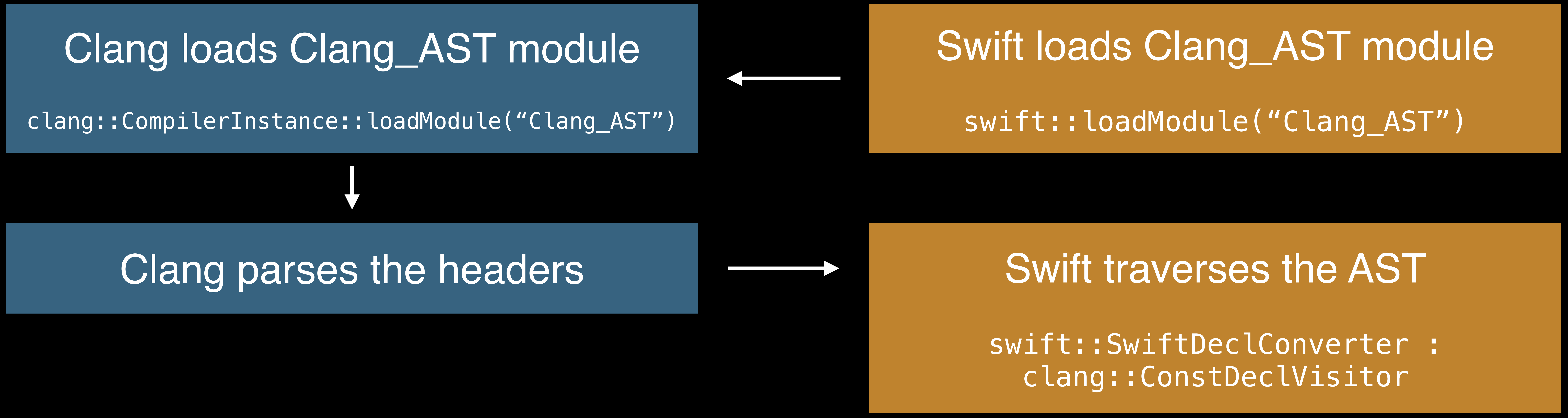
```
clang::CompilerInstance::loadModule("Clang_AST")
```

Clang

Swift

```
// main.swift
import Clang_AST
```

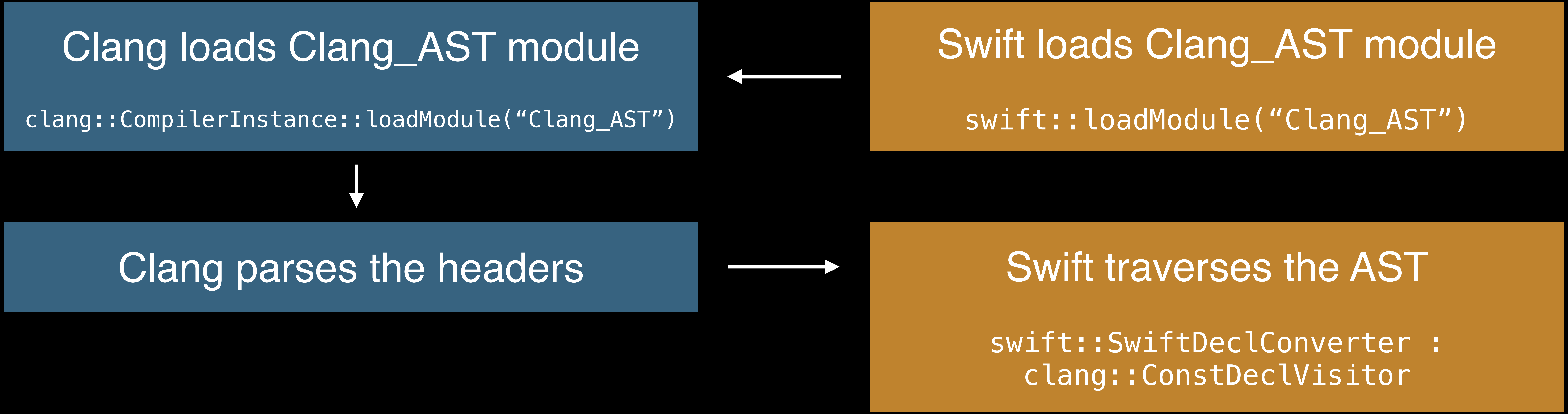
```
% swiftc main.swift
```



```
clang::CompilerInstance::loadModule("Clang_AST")
```

```
swift::loadModule("Clang_AST")
```

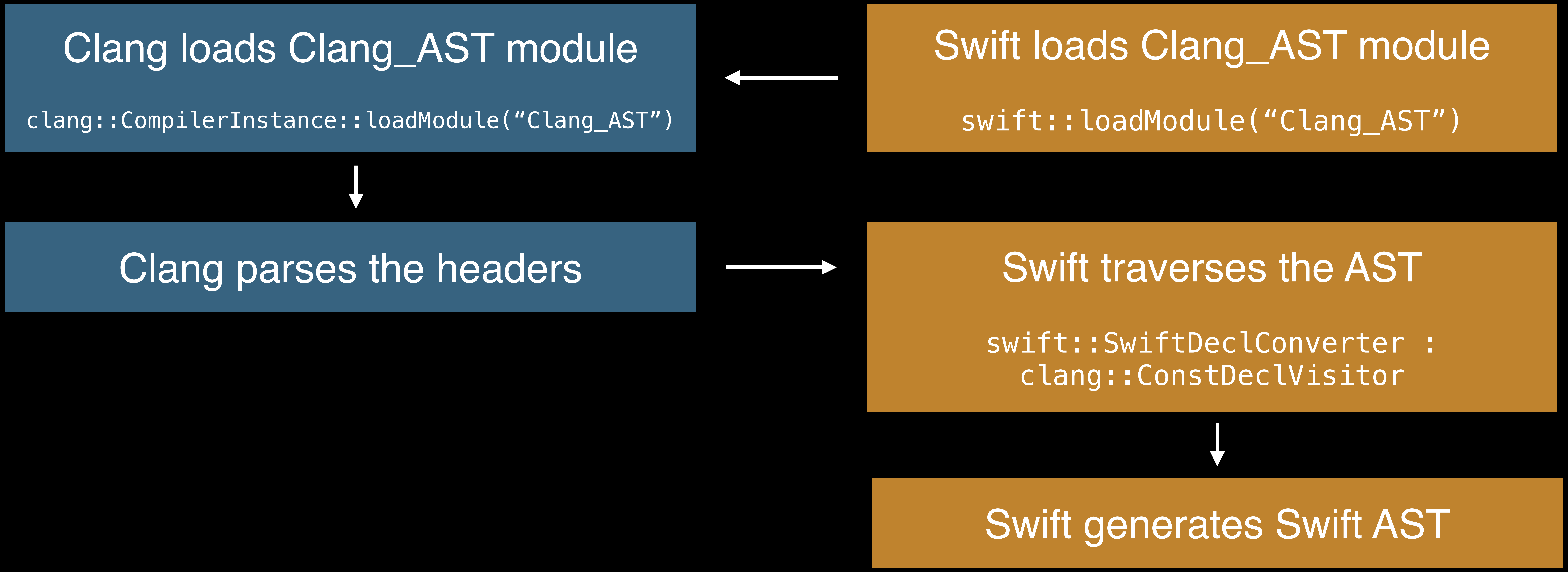
```
swift::SwiftDeclConverter :
  clang::ConstDeclVisitor
```



```
clang::CompilerInstance::loadModule(“Clang_AST”)
```

```
swift::loadModule(“Clang_AST”)
```

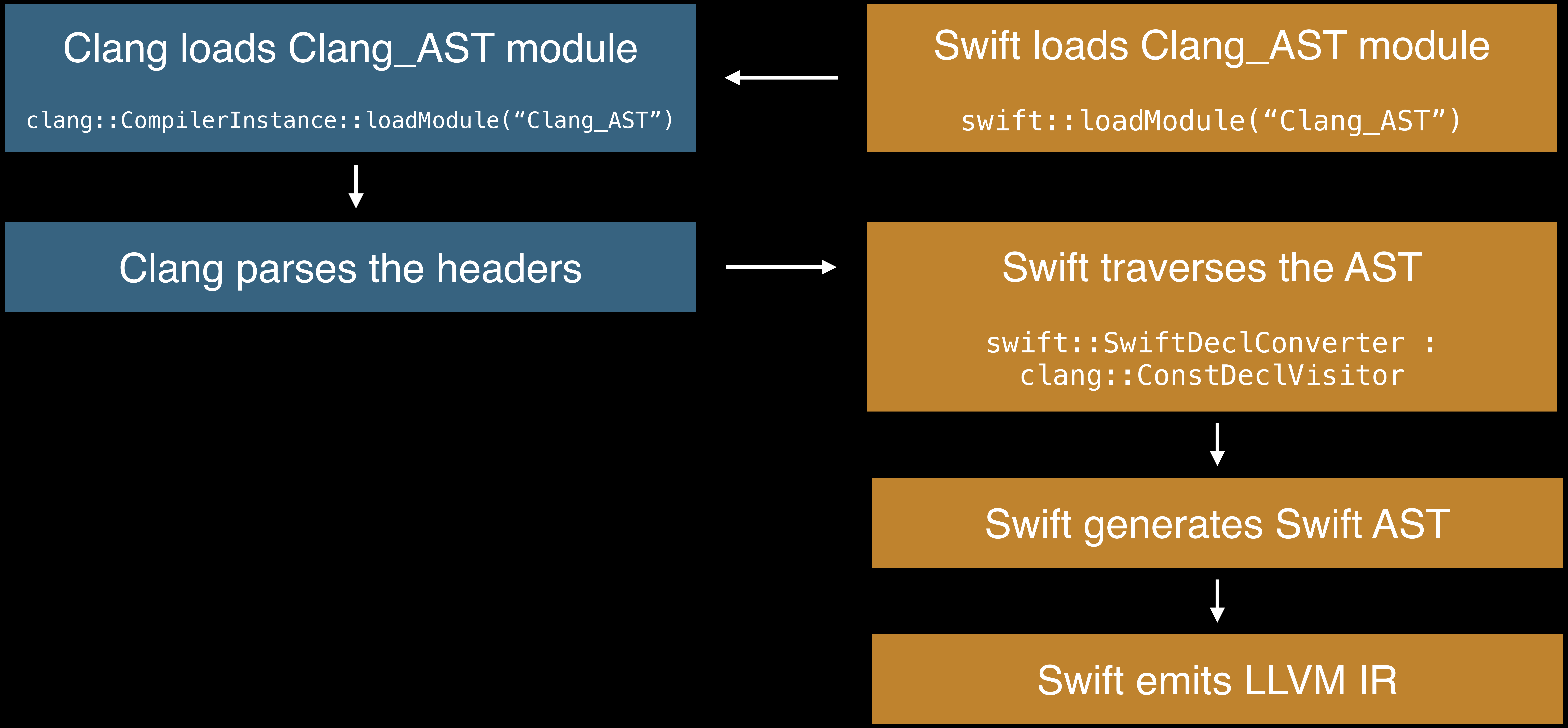
```
swift::SwiftDeclConverter :
  clang::ConstDeclVisitor
```


Clang loads Clang_AST module
Swift loads Clang_AST module
Clang parses the headers
Swift traverses the AST
Swift generates Swift AST

```
clang::CompilerInstance::loadModule(“Clang_AST”)
```

```
swift::loadModule(“Clang_AST”)
```

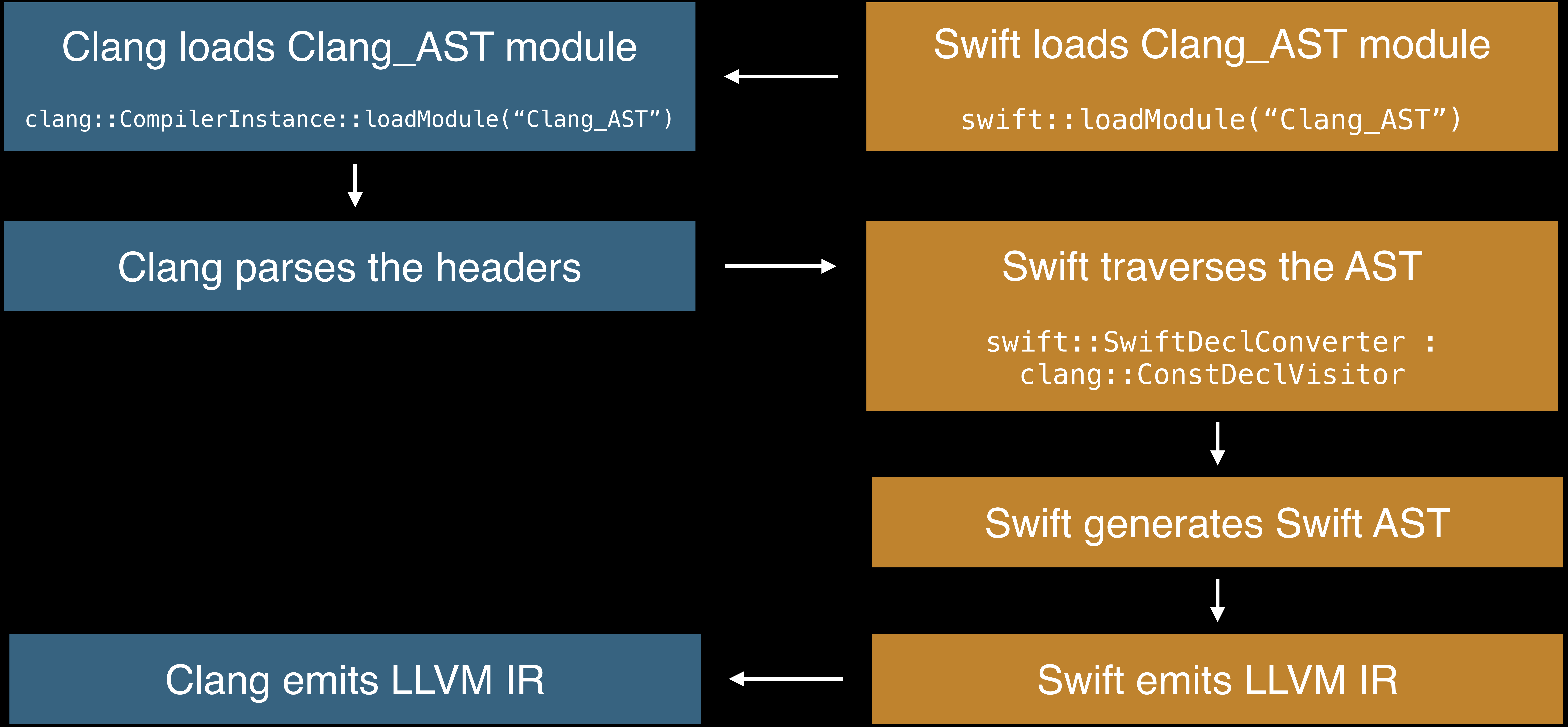
```
swift::SwiftDeclConverter :
  clang::ConstDeclVisitor
```

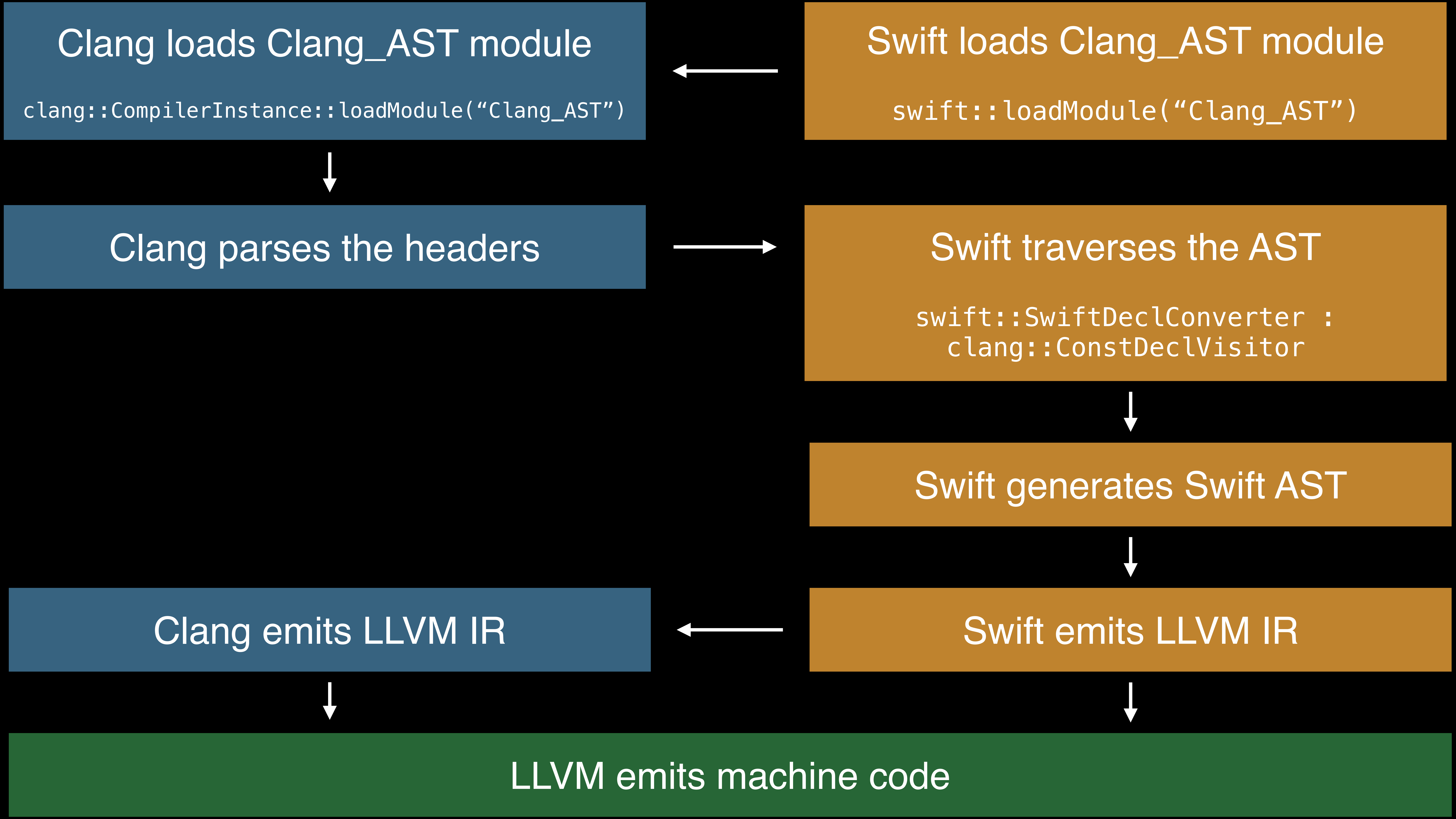

Clang loads Clang_AST module
Swift loads Clang_AST module
Clang parses the headers
Swift traverses the AST
Swift generates Swift AST
Swift emits LLVM IR

```
clang::CompilerInstance::loadModule("Clang_AST")
```

```
swift::loadModule("Clang_AST")
```

```
swift::SwiftDeclConverter :
  clang::ConstDeclVisitor
```


Clang loads Clang_AST module
clang::CompilerInstance::loadModule("Clang_AST")
Swift loads Clang_AST module
swift::loadModule("Clang_AST")
Clang parses the headers
Swift traverses the AST
swift::SwiftDeclConverter : clang::ConstDeclVisitor
Swift generates Swift AST
Clang emits LLVM IR
Swift emits LLVM IR



```
clang::CompilerInstance::loadModule("Clang_AST")
```

```
swift::loadModule("Clang_AST")
```

```
swift::SwiftDeclConverter :
 clang::ConstDeclVisitor
```

Clang

Swift

```
// MyLibrary.swift
public func myFunction()
```

```
% swiftc MyLibrary.swift
```

Clang

Swift

```
// MyLibrary.swift
public func myFunction()
```

```
% swiftc MyLibrary.swift
```

Swift emits a C++ header

↓

```
// MyLibrary-Swift.h
// This is a generated header!
inline void myFunction() { … }
```

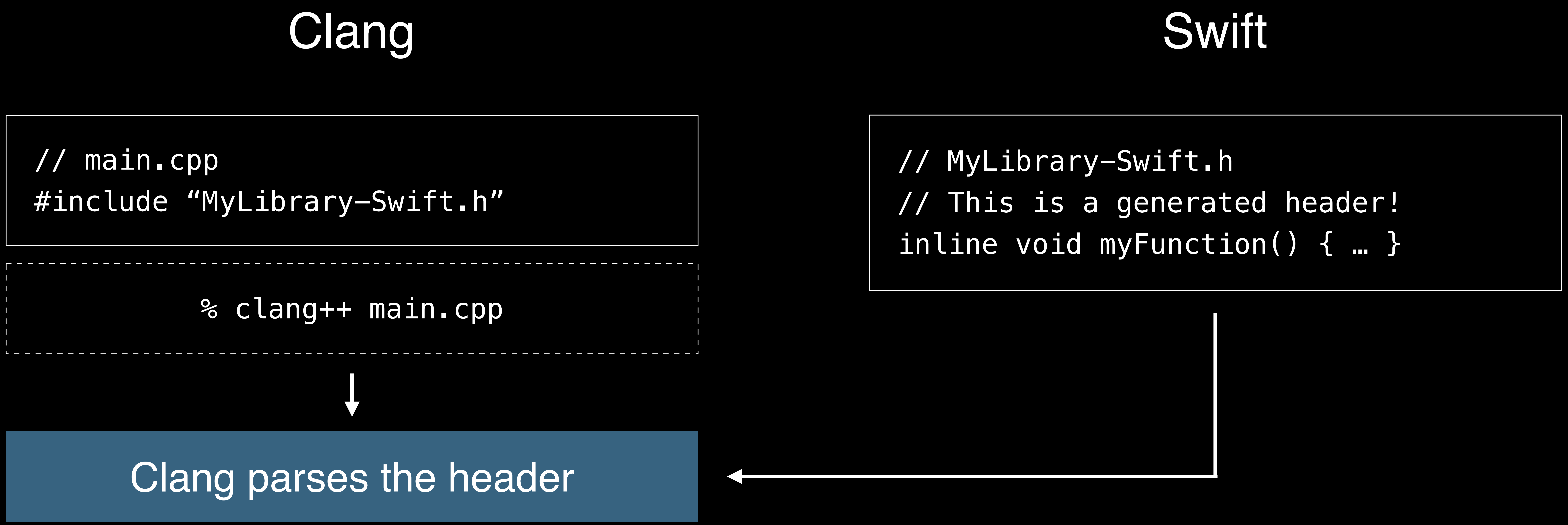
Clang

```
// main.cpp
#include “MyLibrary-Swift.h”
```

```
% clang++ main.cpp
```

Swift

```
// MyLibrary-Swift.h
// This is a generated header!
inline void myFunction() { … }
```



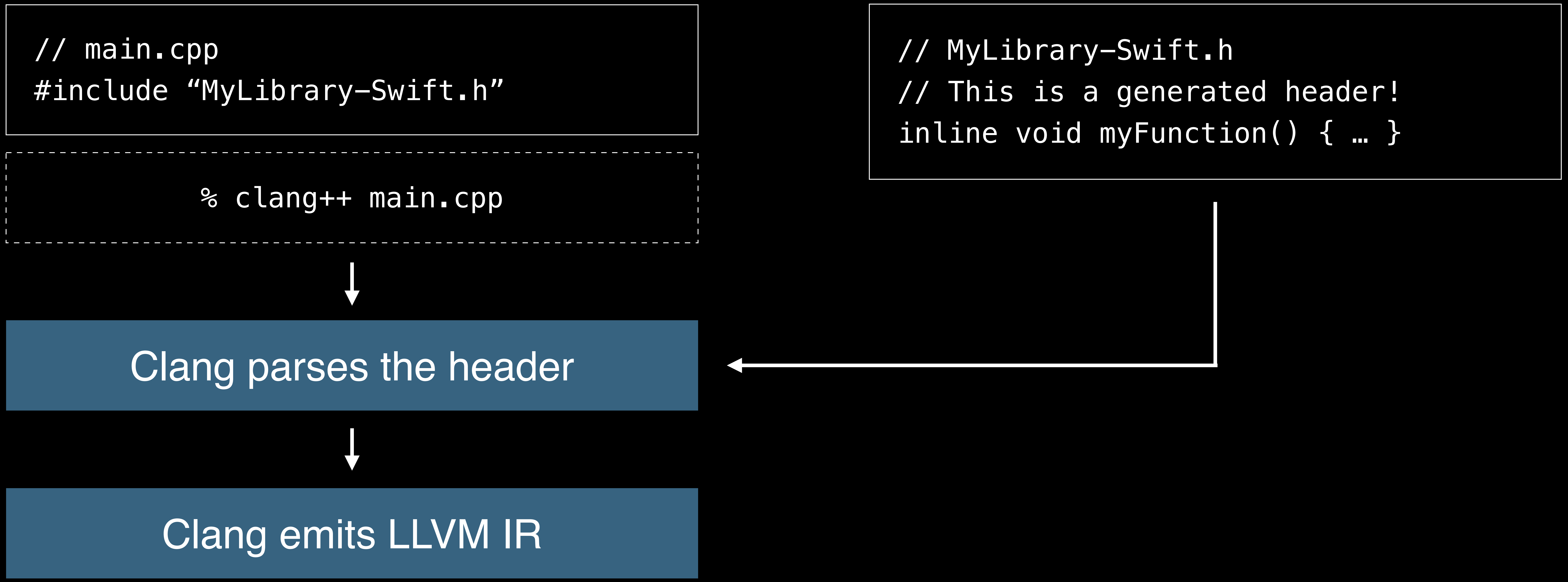
```
// main.cpp
#include “MyLibrary-Swift.h”
```

```
% clang++ main.cpp
```

```
// MyLibrary-Swift.h
// This is a generated header!
inline void myFunction() { … }
```

Clang

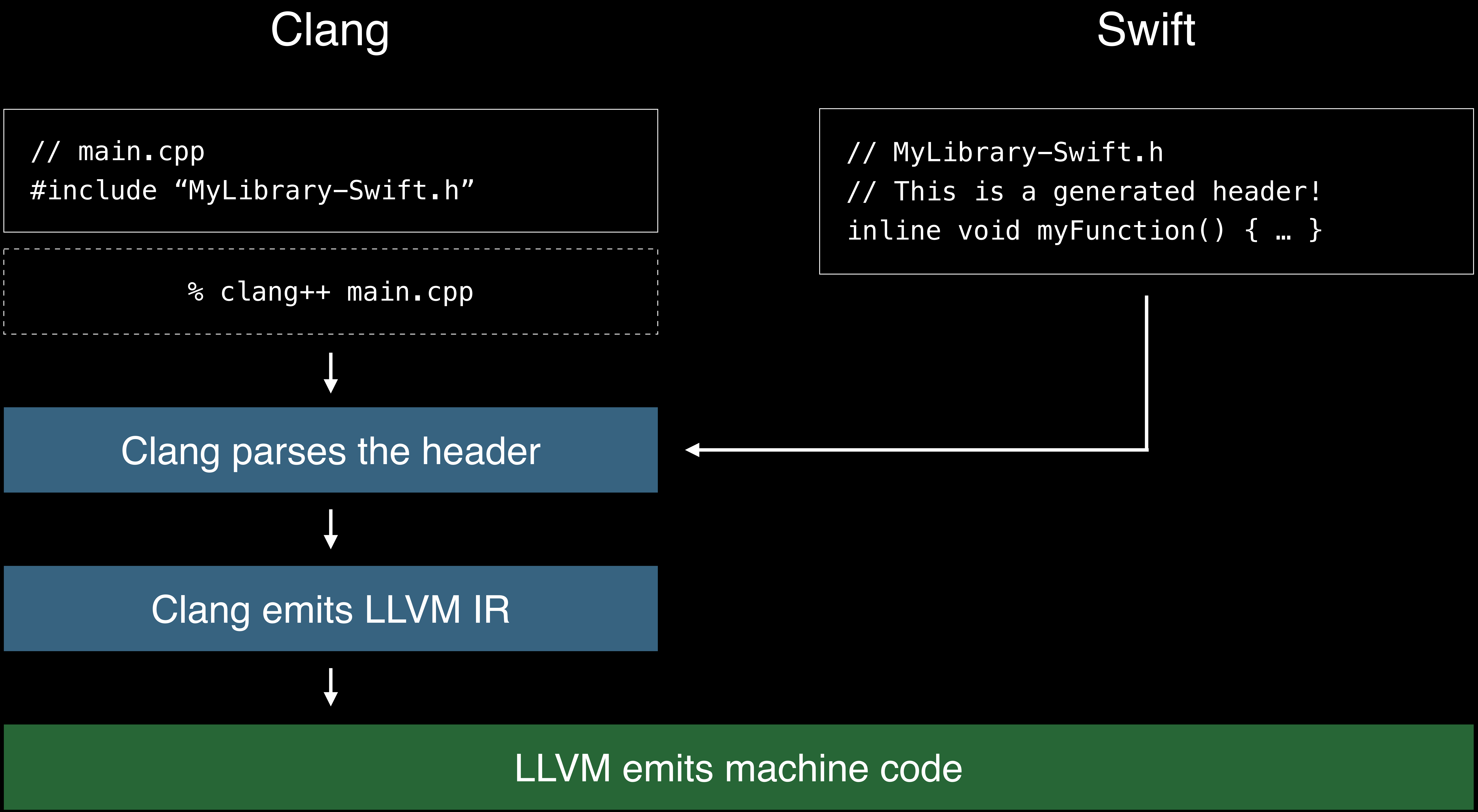
Swift



```
// main.cpp
#include “MyLibrary-Swift.h”
```

```
% clang++ main.cpp
```

```
// MyLibrary-Swift.h
// This is a generated header!
inline void myFunction() { … }
```



```
// main.cpp
#include "MyLibrary-Swift.h"
```

```
% clang++ main.cpp
```

```
// MyLibrary-Swift.h
// This is a generated header!
inline void myFunction() { … }
```

Importing C++ into Swift

Swift

Structs	Classes

Swift

Structs	Classes
• Value types • Do not have pointer identity	

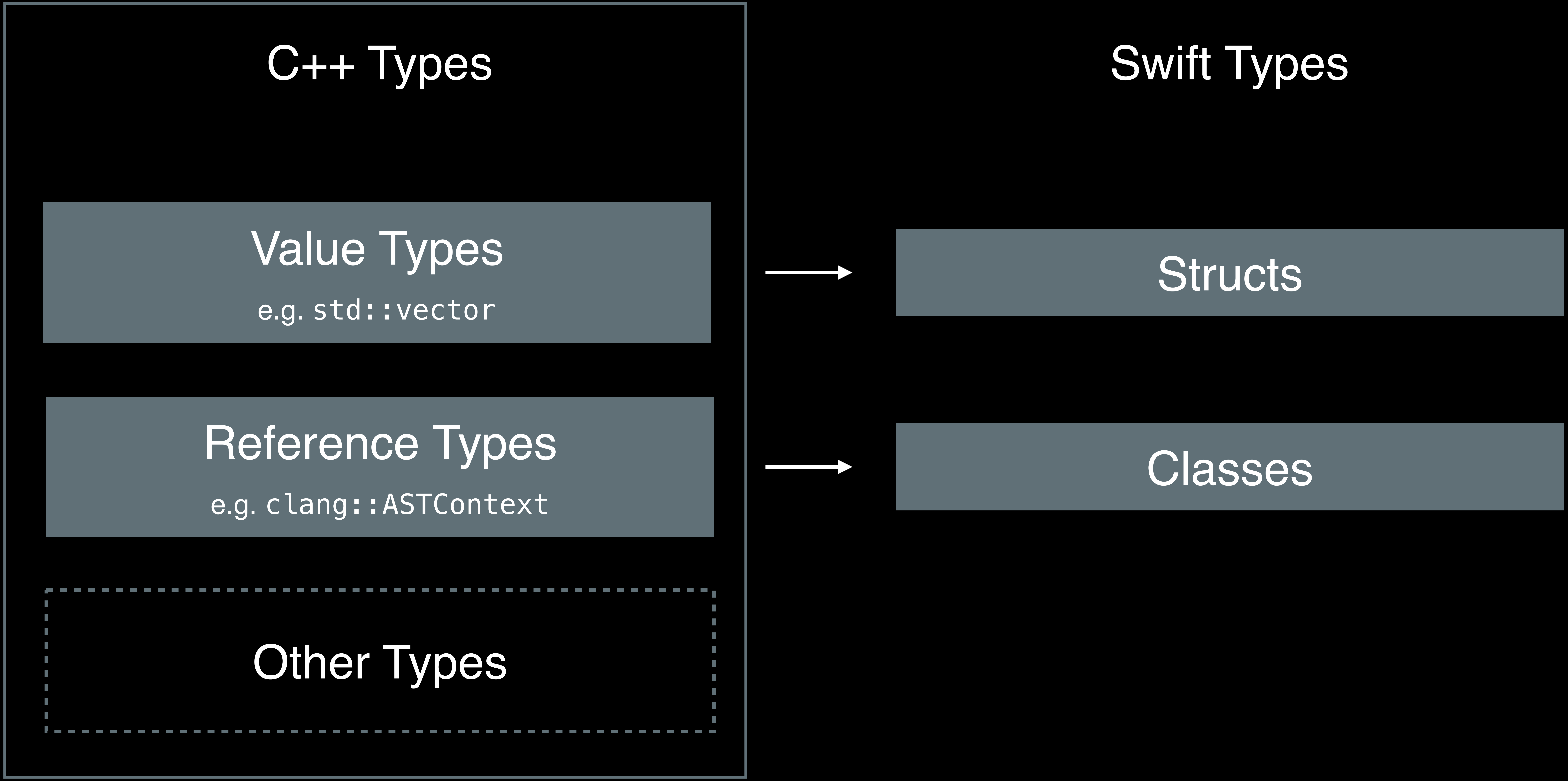
Swift

Structs

- Value types
- Do not have pointer identity

Classes

- Reference types
- Automatically reference-counted
- Can be inherited


C++ Types
Swift Types
Value Types
e.g. std::vector
Structs
Reference Types
e.g. clang::ASTContext
Classes
Other Types

C++ Pointers are Dangerous in Swift

C++ Pointers are Dangerous in Swift

- Difficult to infer lifetime dependencies statically
- Methods can return pointers to internal storage

C++ Pointers are Dangerous in Swift

- Difficult to infer lifetime dependencies statically
- Methods can return pointers to internal storage

```
// main.cpp

void iterate() {
    std::vector<int> v = { … };
    auto it = v.begin();
    auto end = v.end();

    while (it != end) {
        std::cout << *it;
        ++it;
    }
}
```

C++ Pointers are Dangerous in Swift

- Difficult to infer lifetime dependencies statically
- Methods can return pointers to internal storage

```
// main.cpp

void iterate() {
    std::vector<int> v = { … };
    auto it = v.begin();
    auto end = v.end();

    while (it != end) {
        std::cout << *it;
        ++it;
    }
}
```

v destroyed

C++ Pointers are Dangerous in Swift

- Difficult to infer lifetime dependencies statically
- Methods can return pointers to internal storage

```cpp
// main.cpp

void iterate() {
    std::vector<int> v = { … };
    auto it = v.begin();
    auto end = v.end();

    while (it != end) {
        std::cout << *it;
        ++it;
    }
}
```

v destroyed

```swift
// main.swift

func iterate() {
    let v: std.vector<CInt> = [ … ]
    var it = v.begin()
    let end = v.end()

    while it != end {
        print(it.pointee)
        it = it.successor()
    }
}
```

C++ Pointers are Dangerous in Swift

- Difficult to infer lifetime dependencies statically
- Methods can return pointers to internal storage

```cpp
// main.cpp

void iterate() {
    std::vector<int> v = { … };
    auto it = v.begin();
    auto end = v.end();

    while (it != end) {
        std::cout << *it;
        ++it;
    }
}
```

v destroyed

```swift
// main.swift

func iterate() {
    let v: std.vector<CInt> = [ … ]
    var it = v.begin()
    let end = v.end()

    while it != end {
        print(it.pointee)
        it = it.successor()
    }
}
```

v destroyed

C++ Pointers are Dangerous in Swift

- Difficult to infer lifetime dependencies statically
- Methods can return pointers to internal storage

```cpp
// main.cpp

void iterate() {
    std::vector<int> v = { … };
    auto it = v.begin();
    auto end = v.end();

    while (it != end) {
        std::cout << *it;
        ++it;
    }
}
```

v destroyed

```swift
// main.swift

func iterate() {
    let v: std.vector<CInt> = [ … ]
    var it = v.begin()
    let end = v.end()

    while it != end {
        print(it.pointee)
        it = it.successor()
    }
}
```

v destroyed

💥

C++		Swift
Iterators	⟶	🤔

C++ iterator types are generally not safe to use in Swift

- Iterators commonly store a pointer to the underlying container
- There is no lifetime dependency expressed statically

`begin()` and `end()` methods are not available in Swift

C++		Swift
Iterators	→	❌

C++ iterator types are generally not safe to use in Swift

- Iterators commonly store a pointer to the underlying container
- There is no lifetime dependency expressed statically

`begin()` and `end()` methods are not available in Swift

C++	Swift
Iterators	✗
Containers	Collections
…with InputIterators e.g. `std::set`	`Sequence` provides `map`, `filter`, etc.
…with RandomAccessIterators e.g. `std::vector`	`RandomAccessCollection` provides `contains`, `suffix`, etc.

```cpp
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    iterator begin() const;
    iterator end() const;

}
}
```

```
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    iterator begin() const;
    iterator end() const;

}
}



// Attributes.cpp

#include “llvm/IR/InstrTypes.h”

void attributesWithIndices(llvm::AttributeSet attrs) {

    std::vector<std::pair<llvm::Attribute, size_t> >
        result;

    size_t idx = 0;
    for (auto it = attrs.begin(); it != attrs.end();
         it++, idx++) {
        result.push_back(std::make_pair(*it, idx));
    }

}
```

```
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    iterator begin() const;
    iterator end() const;

}
}
```

```
// Attributes.cpp

#include “llvm/IR/InstrTypes.h”

void attributesWithIndices(llvm::AttributeSet attrs) {

    std::vector<std::pair<llvm::Attribute, size_t> >
        result;

    size_t idx = 0;
    for (auto it = attrs.begin(); it != attrs.end();
         it++, idx++) {
        result.push_back(std::make_pair(*it, idx));
    }

}
```

```
// Attributes.swift

import LLVM_IR

func attributesWithIndices(attrs: llvm.AttributeSet) {
    attrs.enumerated()
}
```

```cpp
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    iterator begin() const;
    iterator end() const;

}
}
```

```cpp
// Attributes.cpp

#include “llvm/IR/InstrTypes.h”

void attributesWithIndices(llvm::AttributeSet attrs) {

    std::vector<std::pair<llvm::Attribute, size_t> >
        result;

    size_t idx = 0;
    for (auto it = attrs.begin(); it != attrs.end();
         it++, idx++) {
        result.push_back(std::make_pair(*it, idx));
    }

}
```

```swift
// Attributes.swift

import LLVM_IR

func attributesWithIndices(attrs: llvm.AttributeSet) {
    attrs.enumerated()
}
```

```cpp
// llvm/IR/Attributes.h

namespace llvm {
class AttributeSet {

    iterator begin() const;
    iterator end() const;

}
}
```

```cpp
// Attributes.cpp

#include “llvm/IR/InstrTypes.h”

void attributesWithIndices(llvm::AttributeSet attrs) {

    std::vector<std::pair<llvm::Attribute, size_t> >
        result;

    size_t idx = 0;
    for (auto it = attrs.begin(); it != attrs.end();
         it++, idx++) {
        result.push_back(std::make_pair(*it, idx));
    }

}
```

```swift
// Attributes.swift

import LLVM_IR

func attributesWithIndices(attrs: llvm.AttributeSet) {
    attrs.enumerated()
}
```

Open Challenges

C++		Swift
Class Templates	⟶	🤔
Class Inheritance	⟶	🤔

Open Source Collaboration

Part of the Swift OSS project. We would appreciate your contributions!

github.com/apple/swift

Open Source Collaboration

Part of the Swift OSS project. We would appreciate your contributions!

swift.org/documentation/cxx-interop/

swift.org/cxx-interop-workgroup/

Open Source Collaboration

Part of the Swift OSS project. We would appreciate your contributions!

swift.org/documentation/cxx-interop/

swift.org/cxx-interop-workgroup/

Questions?