# What's New in LLVM

Session 405

Alex Rosenberg Final Boss Level, Compilers and Stuff
Duncan Exon Smith Manager, Clang Frontend
Gerolf Hoflehner Manager, LLVM Backend

## Agenda

LLVM Open Source

Language Support

Compiler Optimizations

## LLVM is Everywhere

## LLVM is Everywhere

## LLVM is Everywhere

## LLVM is Everywhere

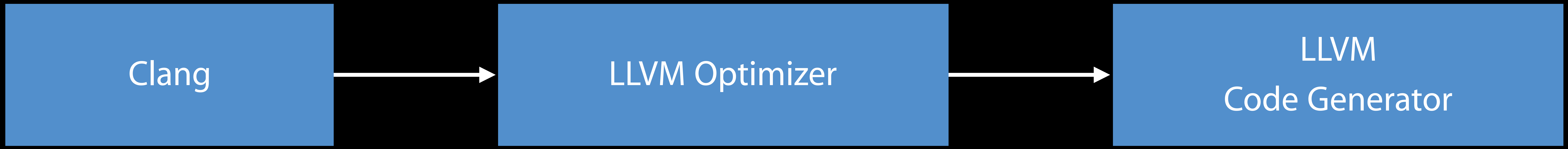
XCODE

OpenGL

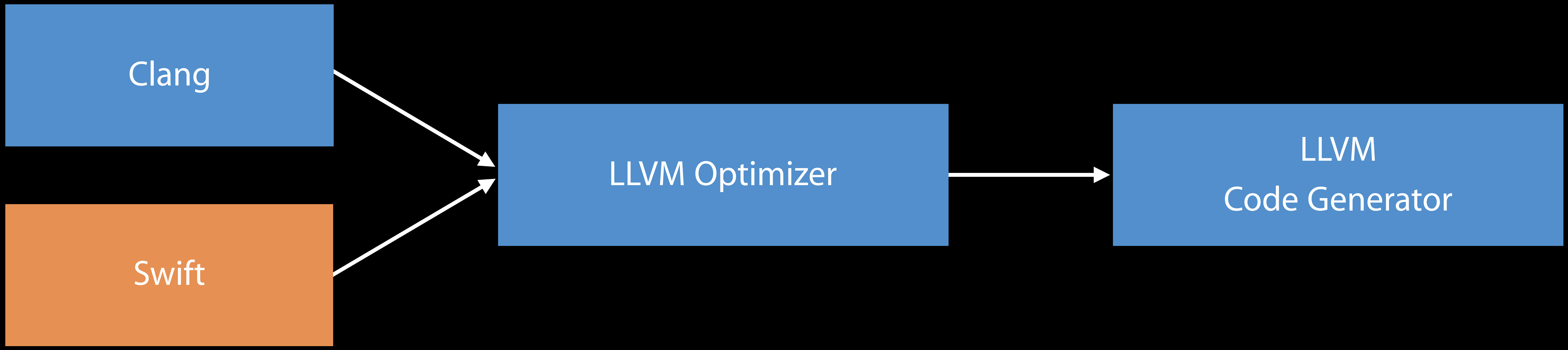
OpenCL

## Open Source

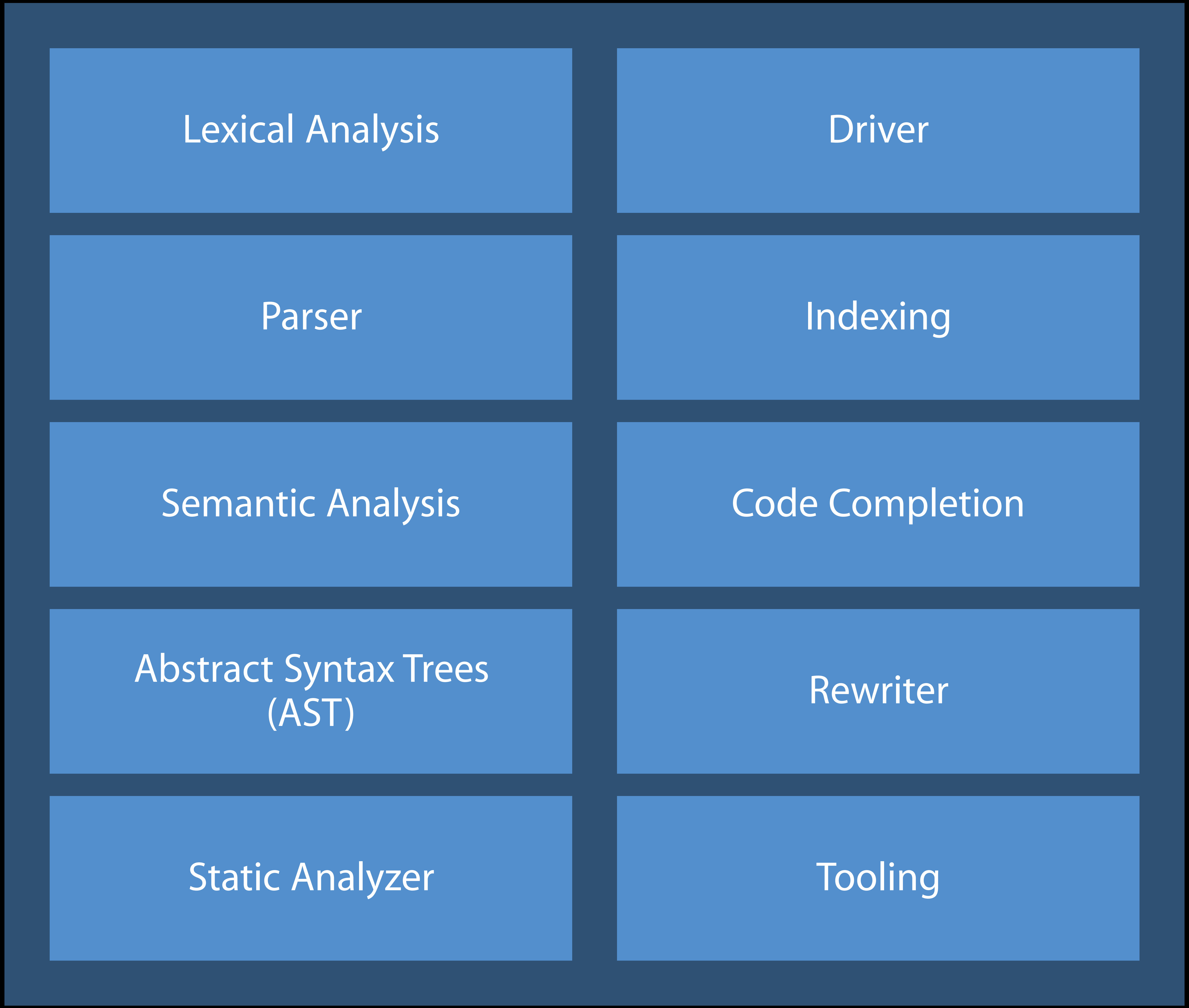
## LLVM Overview



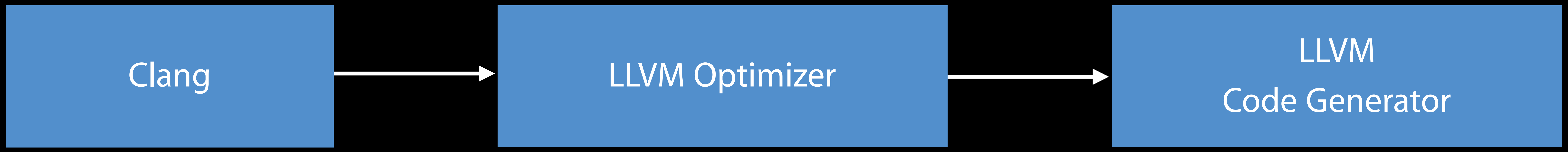
## LLVM Overview



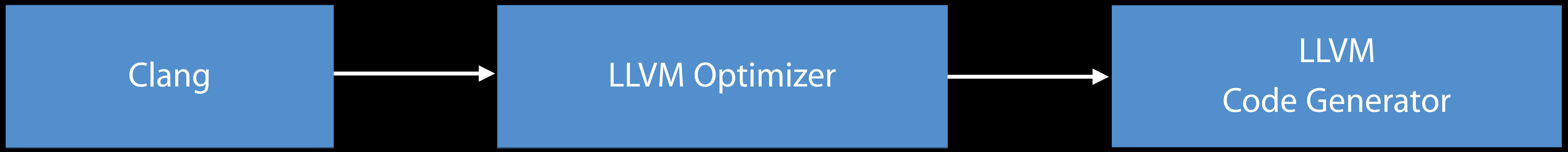
# Clang Overview



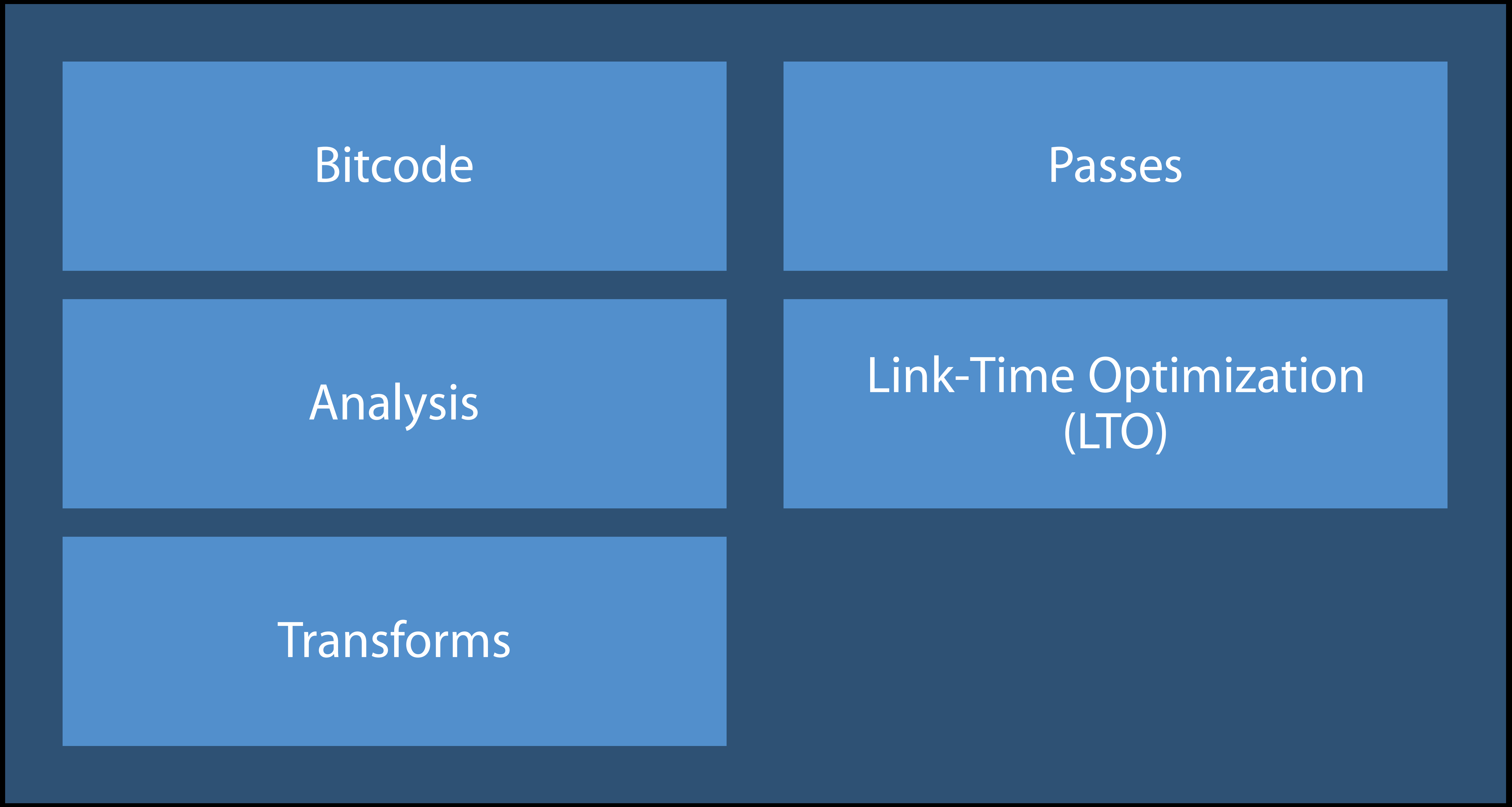
## LLVM Overview



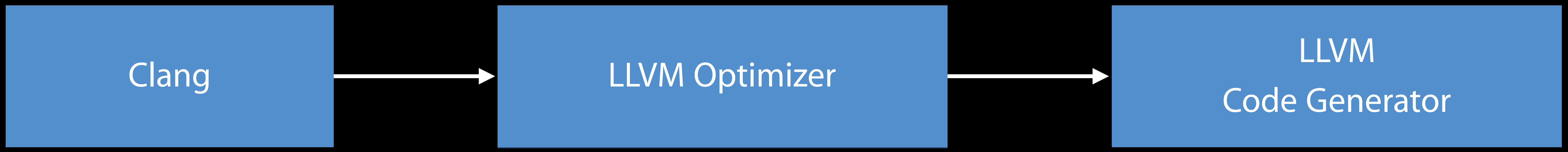
## LLVM Overview



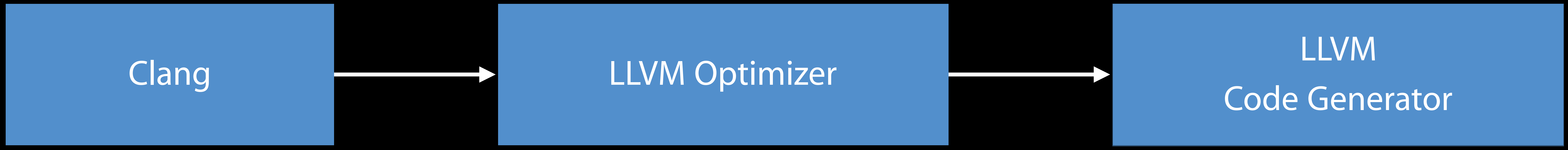
# LLVM Optimizer



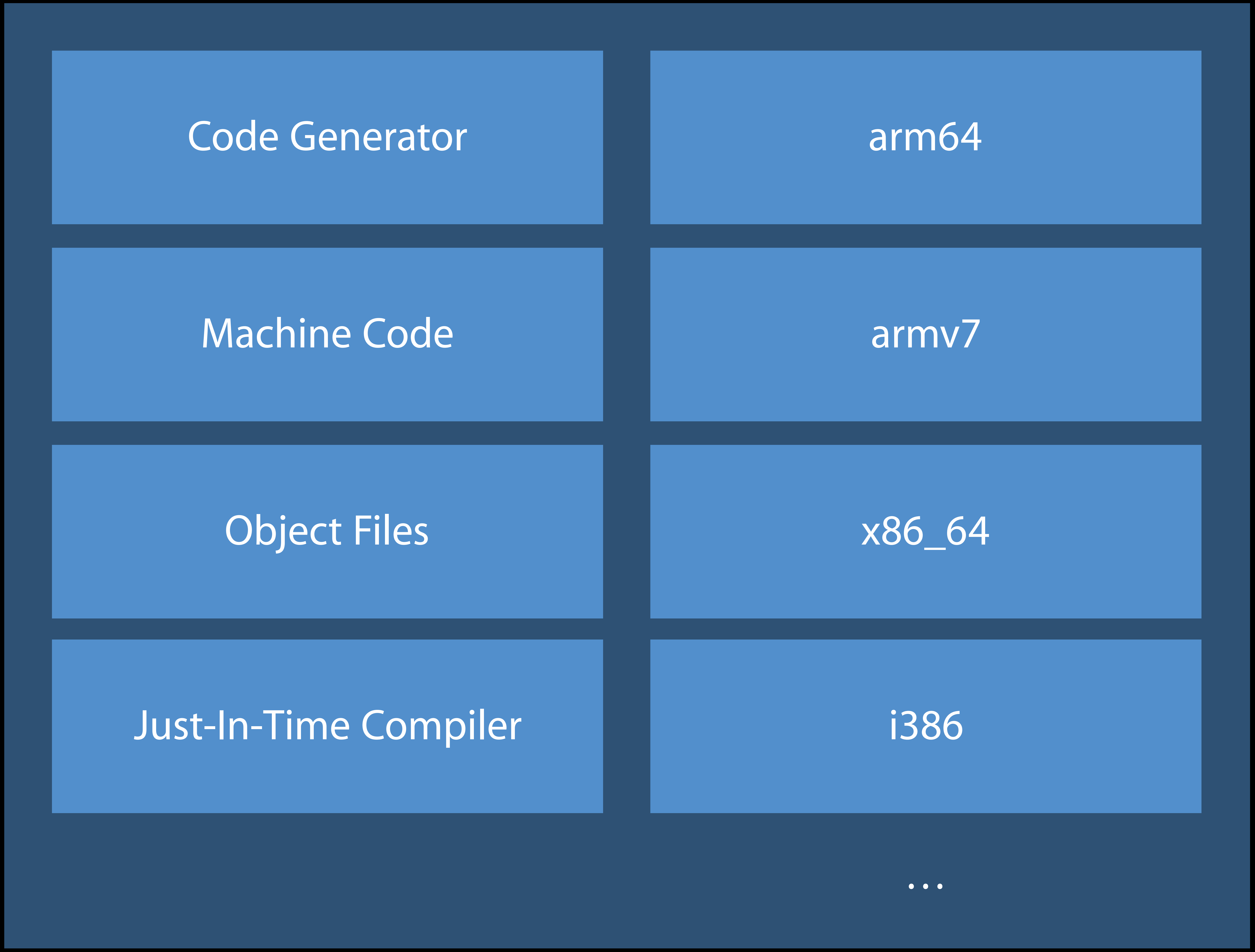
# LLVM Overview



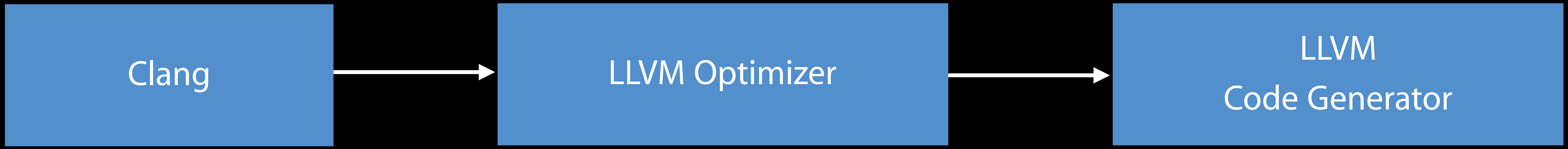
# LLVM Overview



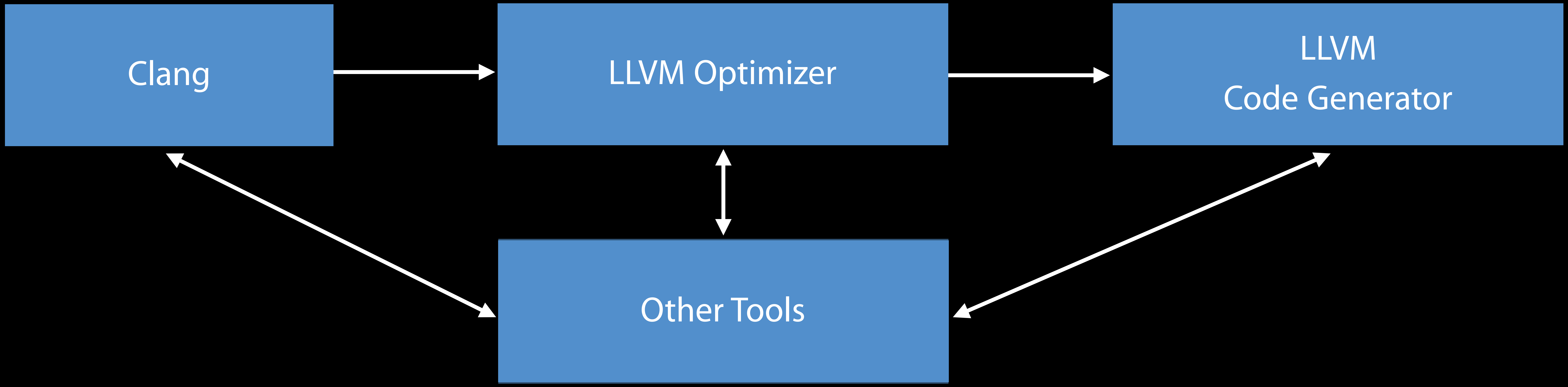
# LLVM Code Generator Overview



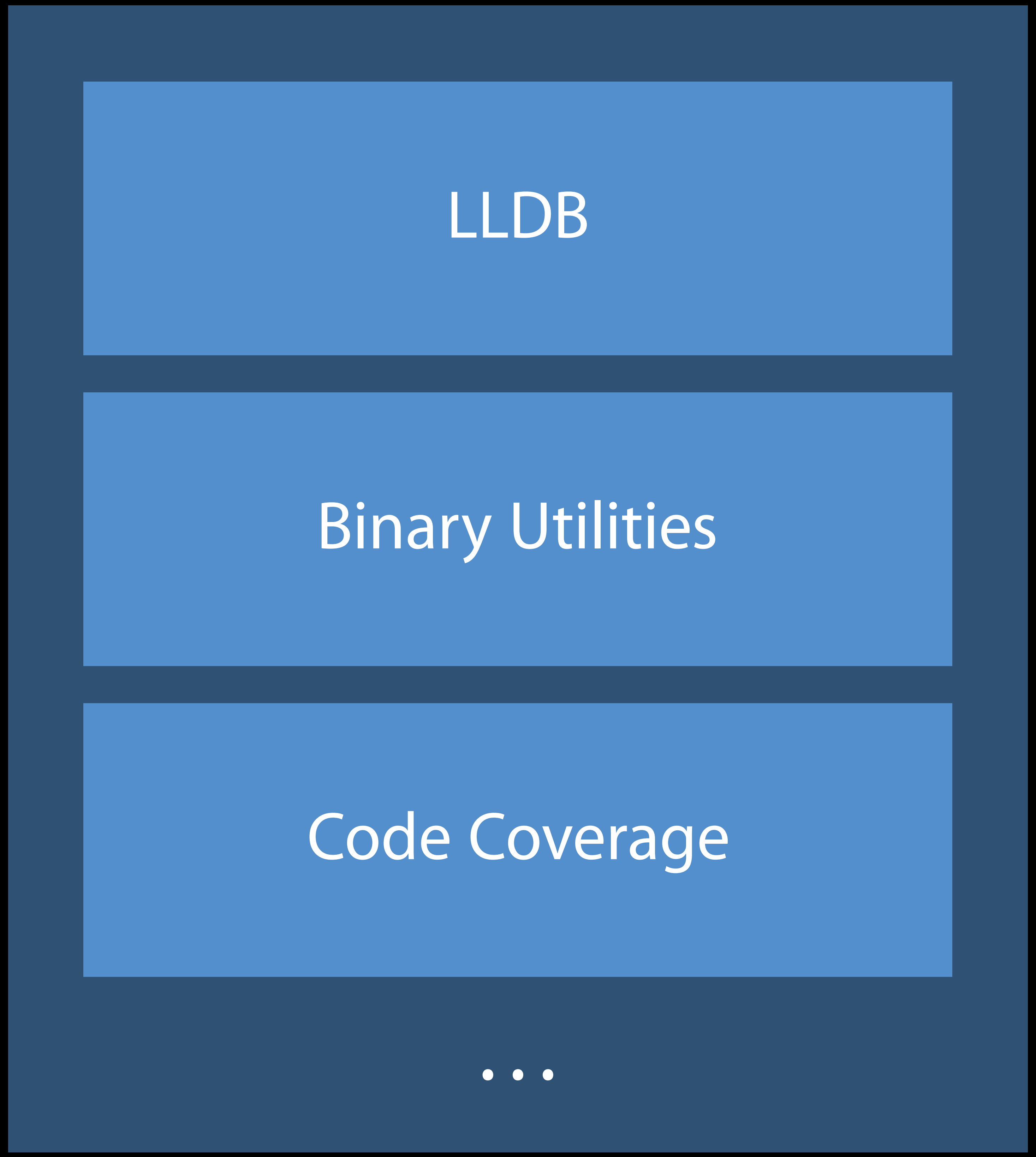
## LLVM Overview



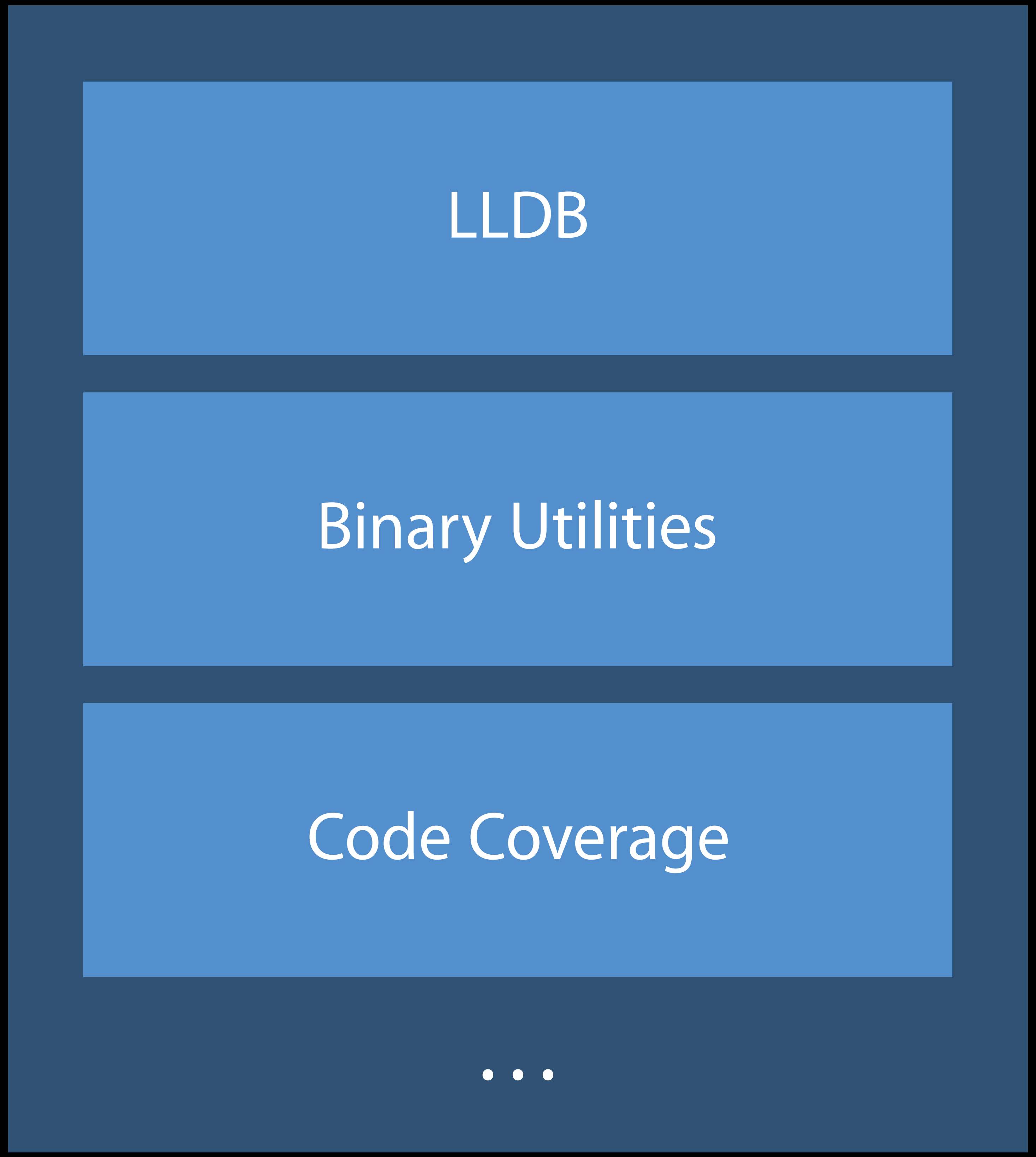
# LLVM Overview



## Other Tools Overview

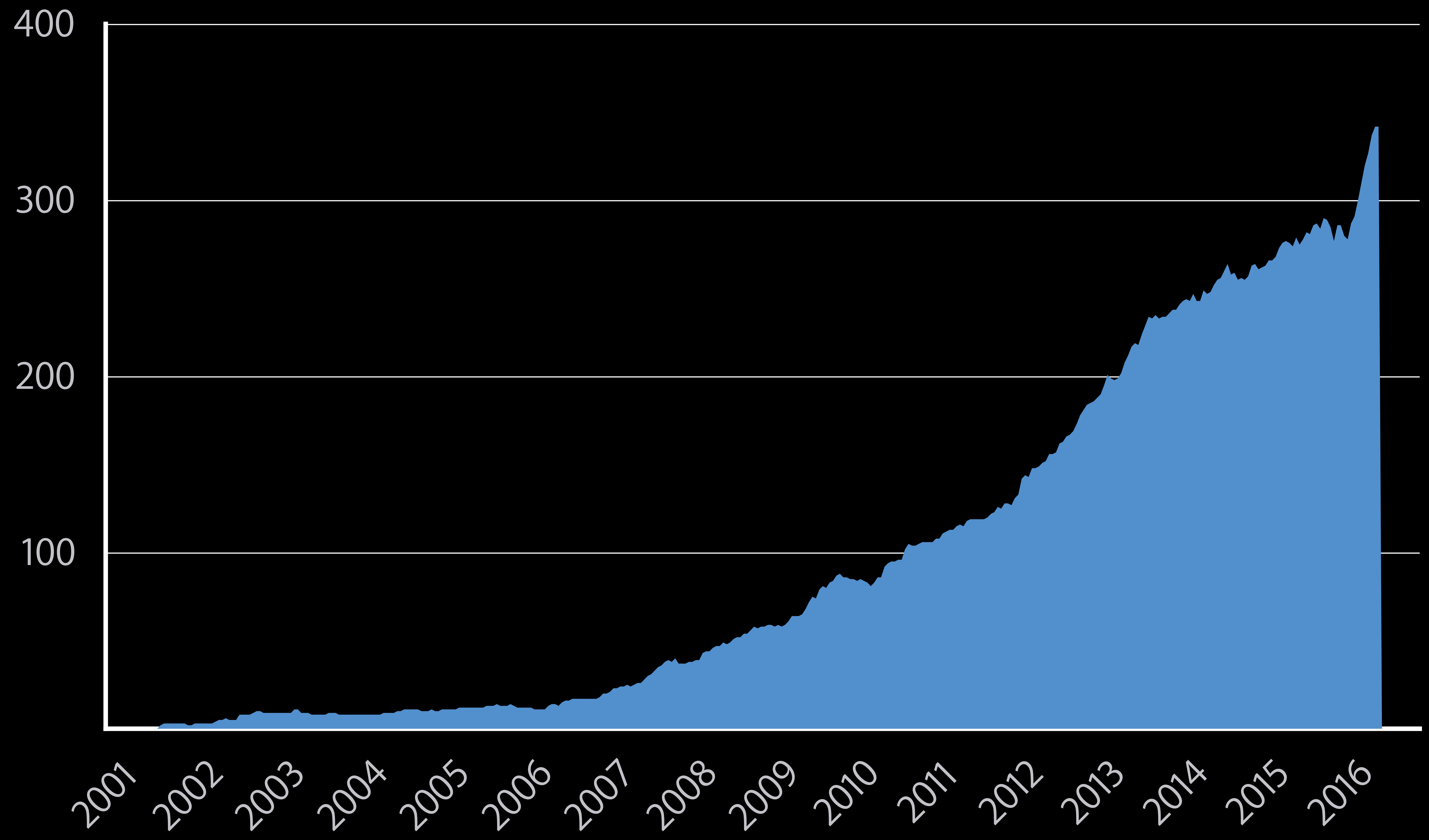


## Other Tools Overview

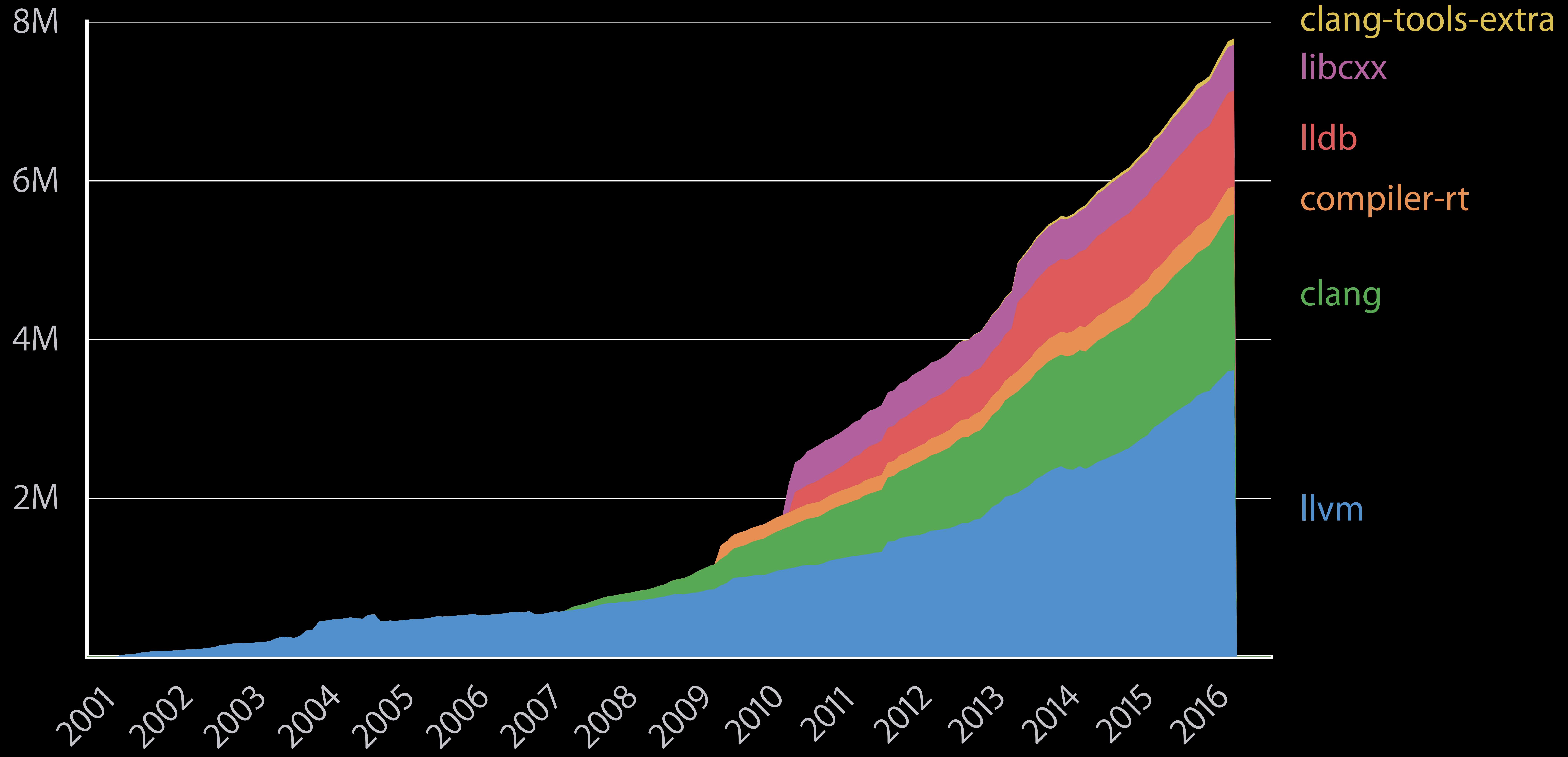


Debugging Tips and Tricks | Pacific Heights | Friday 1:40PM

# Active Committers



## Growth of LLVM



## Patches Welcome

llvm.org

## Language Support

Duncan Exon Smith Manager, Clang Frontend

## Language Support

New Language Features

C++ Library Updates

New Diagnostics

## New Language Features

## Objective-C Class Properties

NEW

Interoperate with Swift type properties

```
@interface MyType : NSObject
@property (class) NSString *someString;
@end

NSLog(@"format string: %@", MyType.someString);
```

# Objective-C Class Properties

NEW

## Interoperate with Swift type properties

```
@interface MyType : NSObject
@property (class) NSString *someString;
@end

NSLog(@"format string: %@", MyType.someString);
```

Declared with `class` flag

## Objective-C Class Properties

NEW

Interoperate with Swift type properties

```
@interface MyType : NSObject
@property (class) NSString *someString;
@end

NSLog(@"format string: %@", MyType.someString);
```

Declared with `class` flag

Accessed with dot syntax

# Objective-C Class Properties

NEW

## Interoperate with Swift type properties

```
@implementation MyType
static NSString *_someString = nil;
+ (NSString *)someString { return _someString; }
+ (void)setSomeString:(NSString *)newString { _someString = newString; }
@end
```

Declared with `class` flag

Accessed with dot syntax

Never synthesized

# Objective-C Class Properties

NEW

## Interoperate with Swift type properties

```
@implementation MyType
static NSString *_someString = nil;
+ (NSString *)someString { return _someString; }
+ (void)setSomeString:(NSString *)newString { _someString = newString; }
@end
```

Declared with `class` flag

Accessed with dot syntax

Never synthesized

# Objective-C Class Properties

NEW

Interoperate with Swift type properties

```
@implementation MyType
static NSString *_someString = nil;
+ (NSString *)someString { return _someString; }
+ (void)setSomeString:(NSString *)newString { _someString = newString; }
@end
```

Declared with `class` flag

Accessed with dot syntax

Never synthesized

# Objective-C Class Properties

NEW

## Interoperate with Swift type properties

```
@implementation MyType
static NSString *_someString = nil;
+ (NSString *)someString { return _someString; }
+ (void)setSomeString:(NSString *)newString { _someString = newString; }
@end
```

Declared with `class` flag

Accessed with dot syntax

Never synthesized

# Objective-C Class Properties

NEW

## Interoperate with Swift type properties

```
@implementation MyType
@dynamic (class) someString;
+ (BOOL)resolveClassMethod:(SEL) name {
  ...
}
@end
```

Declared with `class` flag

Accessed with dot syntax

Never synthesized

Use `@dynamic` to defer to runtime

## C++ Thread-Local Storage (TLS)

NEW

Separate variable per thread

```
// C++11 TLS
thread_local int intPerThread = initializeAnInt();
thread_local SomeClass someClassPerThread(5, getSomeArgument());
```

## C++ Thread-Local Storage (TLS)

NEW

Separate variable per thread

```
// C++11 TLS
thread_local int intPerThread = initializeAnInt();
thread_local SomeClass someClassPerThread(5, getSomeArgument());
```

## C++ Thread-Local Storage (TLS)

NEW

Separate variable per thread

```
// C++11 TLS
thread_local int intPerThread = initializeAnInt();
thread_local SomeClass someClassPerThread(5, getSomeArgument());
```

Dynamic initialization and destruction

## C++ Thread-Local Storage (TLS)

NEW

### Separate variable per thread

```
// C++11 TLS
thread_local int intPerThread = initializeAnInt();
thread_local SomeClass someClassPerThread(5, getSomeArgument());
```

Dynamic initialization and destruction

Arbitrary types

## C++ Thread-Local Storage (TLS)

NEW

Separate variable per thread

```
// C++11 TLS
thread_local int intPerThread = initializeAnInt();
thread_local SomeClass someClassPerThread(5, getSomeArgument());
```

Dynamic initialization and destruction

Arbitrary types

Portable syntax with other C++ compilers

# C Thread-Local Storage (TLS)

## Available even in C++

```
// GCC-style TLS
__thread int intPerThread = 5;
__thread SomeClass *someClassPerThread;

// C11 TLS
_Thread_local int intPerThread = 5;
_Thread_local SomeClass *someClassPerThread;
```

# C Thread-Local Storage (TLS)

Available even in C++

```
// GCC-style TLS
__thread int intPerThread = 5;
__thread SomeClass *someClassPerThread;

// C11 TLS
_Thread_local int intPerThread = 5;
_Thread_local SomeClass *someClassPerThread;
```

# C Thread-Local Storage (TLS)

Available even in C++

```
// GCC-style TLS
__thread int intPerThread = 5;
__thread SomeClass *someClassPerThread;

// C11 TLS
_Thread_local int intPerThread = 5;
_Thread_local SomeClass *someClassPerThread;
```

# C Thread-Local Storage (TLS)

## Available even in C++

```
// GCC-style TLS
__thread int intPerThread = 5;
__thread SomeClass *someClassPerThread;

// C11 TLS
_Thread_local int intPerThread = 5;
_Thread_local SomeClass *someClassPerThread;
```

Lower overhead than C++ `thread_local`

Initializers must be constant

Only "plain old data" (POD) types

## What Kind of TLS is Right for Me?

C thread-local storage

- Constant initializers
- POD types
- Lower overhead

C++ thread-local storage

- Complicated initializers
- Non-POD types
- Portability with other C++ compilers

## What Kind of TLS is Right for Me?

C thread-local storage

- Constant initializers
- POD types
- Lower overhead

C++ thread-local storage

- Complicated initializers
- Non-POD types
- Portability with other C++ compilers

## C++ Library Updates

## Libstdc++ is Deprecated

Upgrade projects to use libc++

- macOS 10.9 or later
- iOS 7 or later

Deprecated

Use libc++ instead

# Libstdc++ is Deprecated

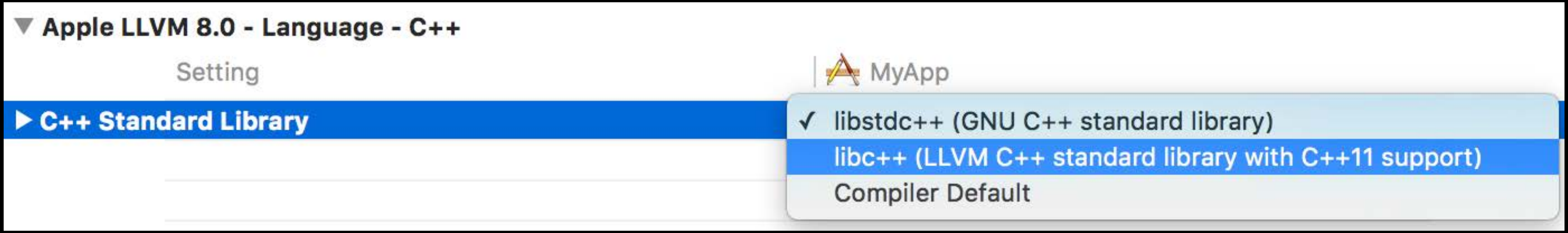
Upgrade projects to use libc++

- macOS 10.9 or later
- iOS 7 or later

Deprecated

Use libc++ instead

```
warning: libstdc++ is deprecated; move to libc++
```



▼ Apple LLVM 8.0 - Language - C++

Setting | MyApp

▶ C++ Standard Library

✓ libstdc++ (GNU C++ standard library)
libc++ (LLVM C++ standard library with C++11 support)
Compiler Default

## Complete C++14 Support

NEW

Updated libc++.dylib

Complete library support for C++14

Over 50 performance improvements and bug fixes on iOS, watchOS, and tvOS

Over 100 performance improvements and bug fixes on macOS

## Libc++ Availability Attributes

NEW

Compile-time availability attributes for features in libc++.dylib

- Compile error if targeting an OS without library support
- Newest API requires macOS 10.12 or iOS 10

## Libc++ Availability Attributes

NEW

Compile-time availability attributes for features in libc++.dylib

- Compile error if targeting an OS without library support
- Newest API requires macOS 10.12 or iOS 10

```
#include <shared_mutex>

int foo(std::shared_timed_mutex &);
```

```
error: 'shared_timed_mutex' is unavailable: introduced in macOS 10.12
```

## New Diagnostics

# Method Outside __kindof Hierarchy

## Type checking of methods on __kindof types

```
@interface MyCustomType : NSObject
- (int)getAwesomeNumber;
@end

__kindof UIView *view = ...;
int i = [view getAwesomeNumber];
```

## Method Outside `__kindof` Hierarchy

### Type checking of methods on `__kindof` types

```
@interface MyCustomType : NSObject
- (int)getAwesomeNumber;
@end


__kindof UIView *view = ...;
int i = [view getAwesomeNumber];
```

# Method Outside __kindof Hierarchy

## Type checking of methods on __kindof types

```
@interface MyCustomType : NSObject
- (int)getAwesomeNumber;
@end

__kindof UIView *view = ...;
int i = [view getAwesomeNumber];
```

# Method Outside __kindof Hierarchy

## Type checking of methods on __kindof types

```
@interface MyCustomType : NSObject
- (int)getAwesomeNumber;
@end

__kindof UIView *view = ...;
int i = [view getAwesomeNumber];
```

```
error: method 'getAwesomeNumber' on kindof returns a method that is outside of the class hierarchy
int i = [view getAwesomeNumber];
        ~~~~~~^~~~~~~~~~~~~~~~
note: receiver is instance of class declared here
@interface UIView : UIResponder <NSCoding, UIAppearance, UIAppearanceContainer, ...
           ^
```

# Circular Dependencies in Containers

## Strong reference cycles and undefined behavior

```
NSMutableSet *s = [NSMutableSet new];
[s addObject:s];
```

# Circular Dependencies in Containers

## Strong reference cycles and undefined behavior

```
NSMutableSet *s = [NSMutableSet new];
[s addObject:s];
```

# Circular Dependencies in Containers

## Strong reference cycles and undefined behavior

```
NSMutableSet *s = [NSMutableSet new];
[s addObject:s];
```

```
warning: adding 's' to 's' might cause circular dependency in container [-Wobjc-circular-container]
  [s addObject:s];
  ^
```

## Infinite Recursion

All paths through a function call itself

```
unsigned factorial(unsigned n) {
  return n ? factorial(n - 1) * n
           : factorial(1);
}
```

## Infinite Recursion

All paths through a function call itself

```
unsigned factorial(unsigned n) {
  return n ? factorial(n - 1) * n
           : factorial(1);
}
```

## Infinite Recursion

All paths through a function call itself

```
unsigned factorial(unsigned n) {
  return n ? factorial(n - 1) * n
           : factorial(1);
}
```

# Infinite Recursion

## All paths through a function call itself

```
unsigned factorial(unsigned n) {
  return n ? factorial(n - 1) * n
           : factorial(1);
}
```

```
warning: all paths through this function will call itself [-Winfinite-recursion]
unsigned factorial(unsigned n) {
                               ^
```

## Infinite Recursion

All paths through a function call itself

```
unsigned factorial(unsigned n) {
  return n ? factorial(n - 1) * n
           : 1;
}
```

```
warning: all paths through this function will call itself [-Winfinite-recursion]
unsigned factorial(unsigned n) {
                               ^
```

# Infinite Recursion

## All paths through a function call itself

```
unsigned factorial(unsigned n) {
  return n ? factorial(n - 1) * n
           : 1;
}
```

## Pessimizing Moves

### Blocking Return Value Optimization (RVO)

```
std::vector<int> generateBars() {
  std::vector<int> bars = loadBullion();
  return std::move(bars);
}
```

## Pessimizing Moves

### Blocking Return Value Optimization (RVO)

```
std::vector<int> generateBars() {
  std::vector<int> bars = loadBullion();
  return std::move(bars);
}
```

# Pessimizing Moves

## Blocking Return Value Optimization (RVO)

```
std::vector<int> generateBars() {
  std::vector<int> bars = loadBullion();
  return std::move(bars);
}
```

```
warning: moving a local object in a return statement prevents copy elision [-Wpessimizing-move]
  return std::move(bars);
         ^
```

# Pessimizing Moves

## Blocking Return Value Optimization (RVO)

```
std::vector<int> generateBars() {
  std::vector<int> bars = loadBullion();
  return bars;
}
```

```
warning: moving a local object in a return statement prevents copy elision [-Wpessimizing-move]
  return std::move(bars);
         ^
```

## Pessimizing Moves

### Blocking Return Value Optimization (RVO)

```
std::vector<int> generateBars() {
  std::vector<int> bars = loadBullion();
  return bars;
}
```

## Redundant Moves

### Distracting boilerplate

```
std::string rewriteText(std::string text) {
  rewriteTextInline(text);
  return std::move(text);
}
```

## Redundant Moves

### Distracting boilerplate

```
std::string rewriteText(std::string text) {
  rewriteTextInline(text);
  return std::move(text);
}
```

```
warning: redundant move in return statement [-Wredundant-move]
  return std::move(text);
         ^
```

## Redundant Moves

### Distracting boilerplate

```
std::string rewriteText(std::string text) {
  rewriteTextInline(text);
  return text;
}
```

```
warning: redundant move in return statement [-Wredundant-move]
  return std::move(text);
         ^
```

## Redundant Moves

### Distracting boilerplate

```
std::string rewriteText(std::string text) {
  rewriteTextInline(text);
  return text;
}
```

# Reference to an Implicit Conversion

## C++ range-based for loops

```
std::vector<short> shorts = makeShorts();
for (const int &i : shorts) {
  printNumber(i);
}
```

# Reference to an Implicit Conversion

## C++ range-based for loops

```
std::vector<short> shorts = makeShorts();
for (const int &i : shorts) {
  printNumber(i);
}
```

# Reference to an Implicit Conversion

## C++ range-based for loops

```
std::vector<short> shorts = makeShorts();
for (const int &i : shorts) {
  printNumber(i);
}
```

```
warning: loop variable 'i' has type 'const int &' but is initialized with type 'short'
         resulting in a copy [-Wrange-loop-analysis]
for (const int &i : shorts)
                ^

note: use non-reference type 'int' to keep the copy or type 'const short &' to prevent copying
for (const int &i : shorts)
     ^~~~~~~~~~~~~~
```

# Reference to an Implicit Conversion

## C++ range-based for loops

```
std::vector<short> shorts = makeShorts();
for (int i : shorts) {
  printNumber(i);
}
```

```
warning: loop variable 'i' has type 'const int &' but is initialized with type 'short'
         resulting in a copy [-Wrange-loop-analysis]
for (const int &i : shorts)
                ^

note: use non-reference type 'int' to keep the copy or type 'const short &' to prevent copying
for (const int &i : shorts)
     ^~~~~~~~~~~~~~
```

# Reference to an Implicit Conversion

## C++ range-based for loops

```
std::vector<short> shorts = makeShorts();
for (int i : shorts) {
  printNumber(i);
}
```

# Reference to a Copy

## C++ range-based for loops

```
std::vector<bool> bools = makeBools();
for (const bool &b : bools) {
  useABool(b);
}
```

# Reference to a Copy

## C++ range-based for loops

```
std::vector<bool> bools = makeBools();
for (const bool &b : bools) {
  useABool(b);
}
```

# Reference to a Copy

## C++ range-based for loops

```
std::vector<bool> bools = makeBools();
for (const bool &b : bools) {
  useABool(b);
}
```

```
warning: loop variable 'b' is always a copy because the range of type 'std::vector<bool>'
         does not return a reference [-Wrange-loop-analysis]
for (const bool &b : bools)
                 ^

note: use non-reference type 'bool'
for (const bool &b : bools)
     ^~~~~~~~~~~~~~~
```

# Reference to a Copy

## C++ range-based for loops

```
std::vector<bool> bools = makeBools();
for (bool b : bools) {
  useABool(b);
}
```

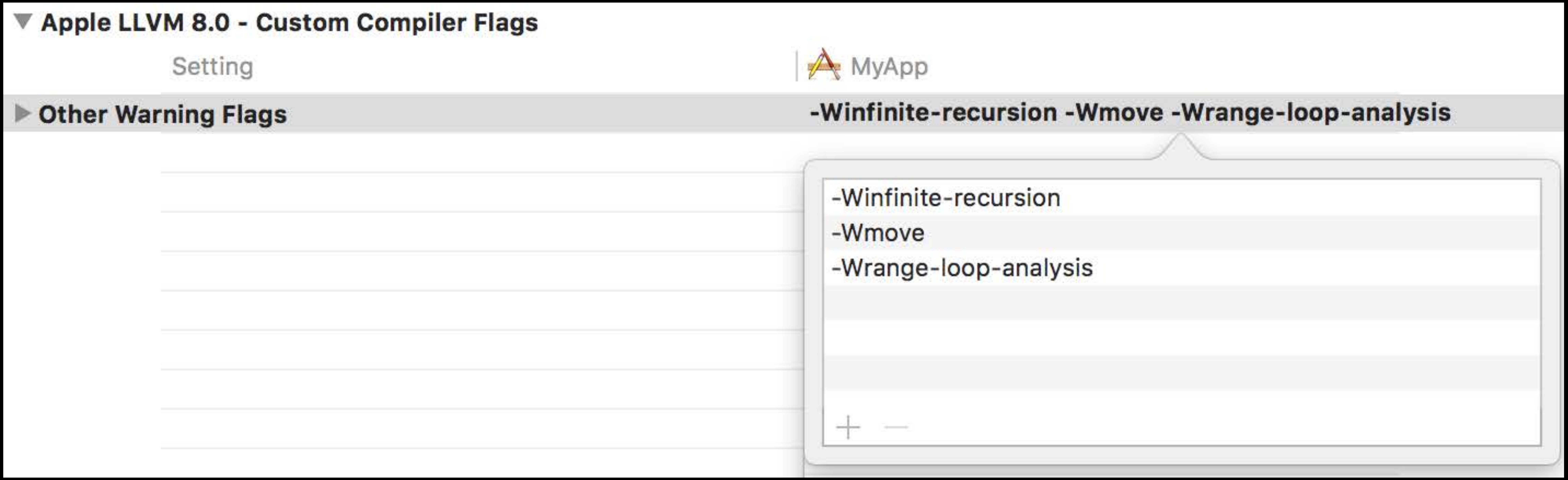
```
warning: loop variable 'b' is always a copy because the range of type 'std::vector<bool>'
         does not return a reference [-Wrange-loop-analysis]
for (const bool &b : bools)
                  ^

note: use non-reference type 'bool'
for (const bool &b : bools)
     ^~~~~~~~~~~~~~~
```

# Reference to a Copy

## C++ range-based for loops

```
std::vector<bool> bools = makeBools();
for (bool b : bools) {
  useABool(b);
}
```

# Enabling Warnings in Xcode


▼ Apple LLVM 8.0 - Custom Compiler Flags

| Setting | MyApp |
|---|---|
| ▶ Other Warning Flags | -Winfinite-recursion -Wmove -Wrange-loop-analysis |

```
-Winfinite-recursion
-Wmove
-Wrange-loop-analysis
```

## Compiler Optimizations

## Compiler Optimizations

Loop Distribution

Non-Temporal Store

Selective Fused Multiply-Adds

Shrink-Wrapping

Load Scheduling

Advanced Loop Unrolling

Vectorization Enhancements

Stack Packing

Software Prefetching

Link-Time Optimization

## Compiler Optimizations

Link-Time Optimization

Code Generation

arm64 Cache Tuning

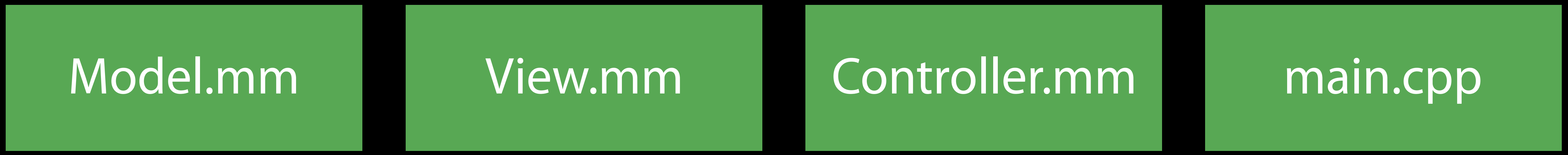
## Link-Time Optimization

## What is Link-Time Optimization (LTO)?

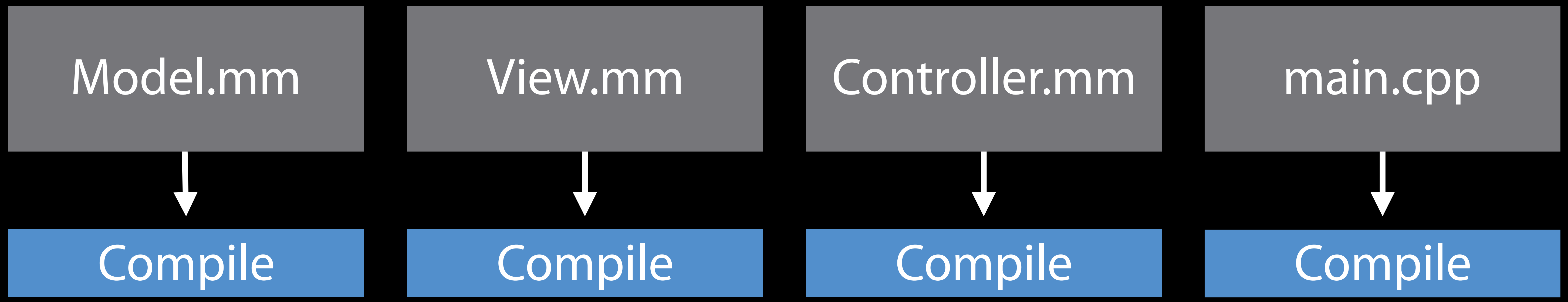
Maximize runtime performance by optimizing at link-time

- Inline functions across source files
- Remove dead code
- Enable powerful whole program optimizations

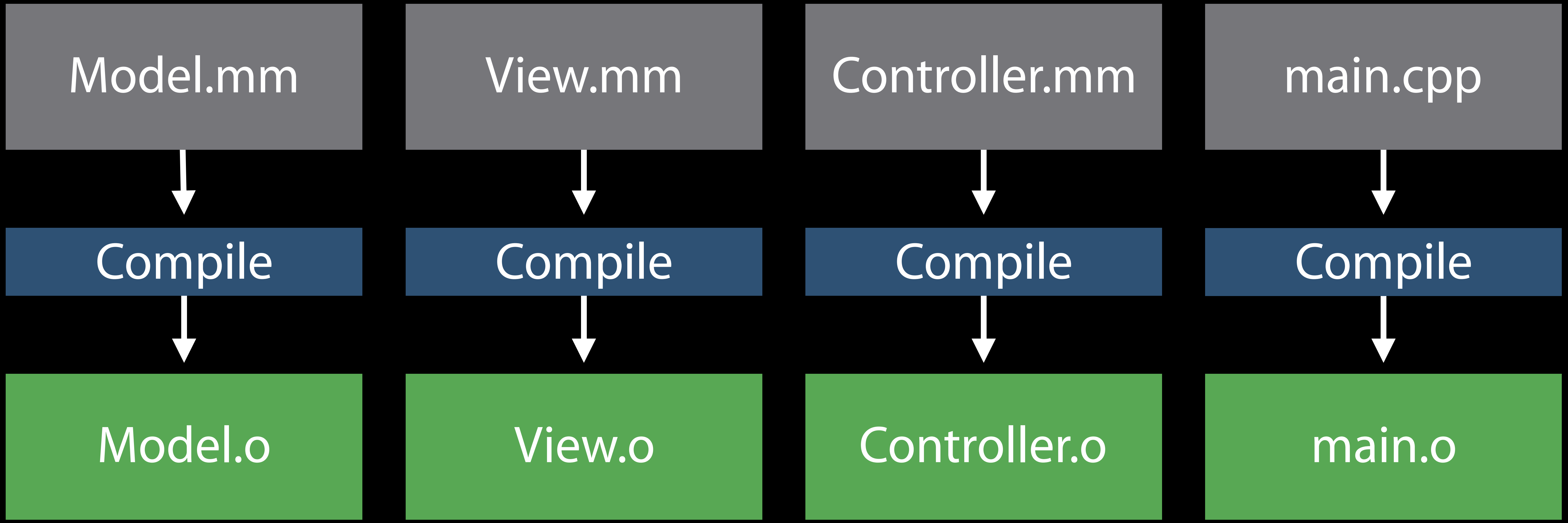
# Traditional Compilation Model



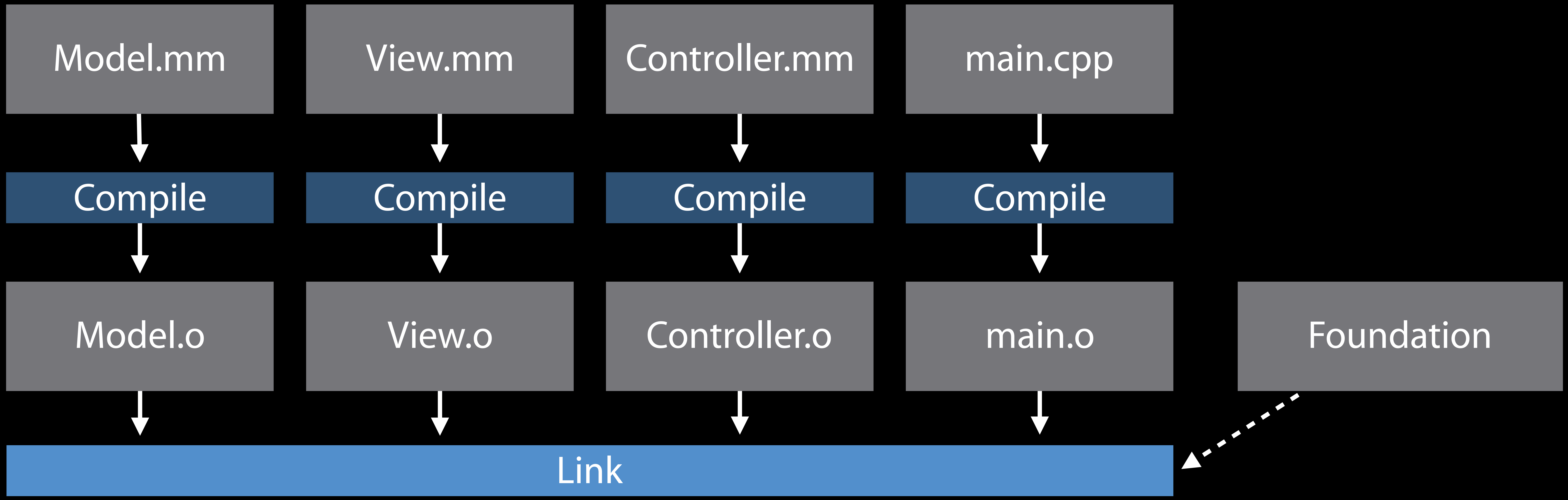
# Traditional Compilation Model



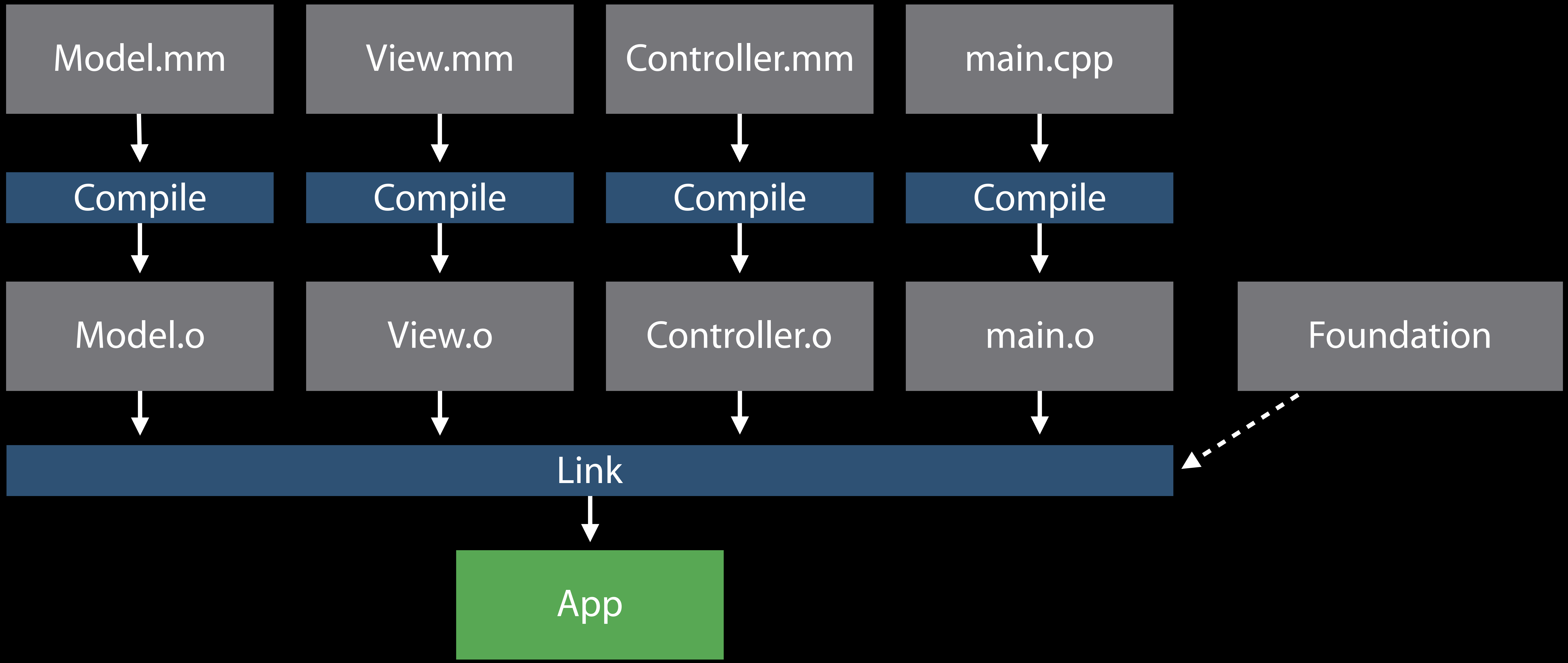
# Traditional Compilation Model



# Traditional Compilation Model



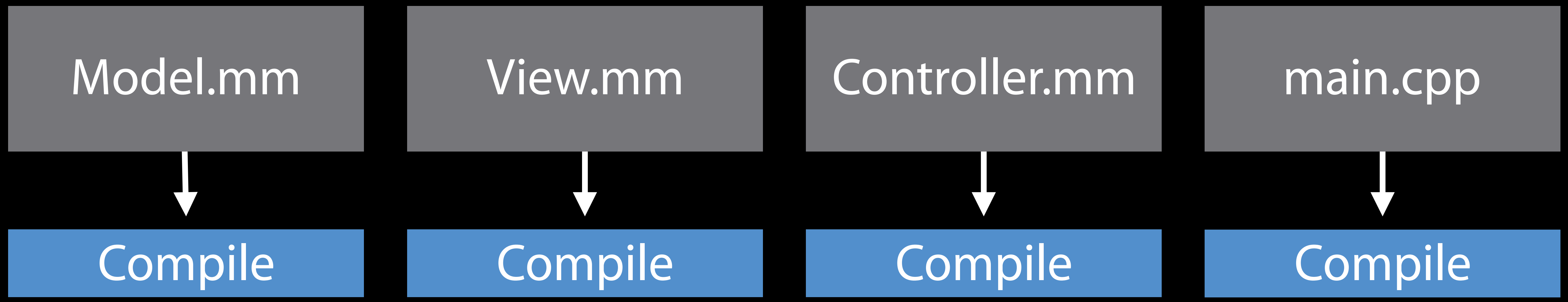
# Traditional Compilation Model



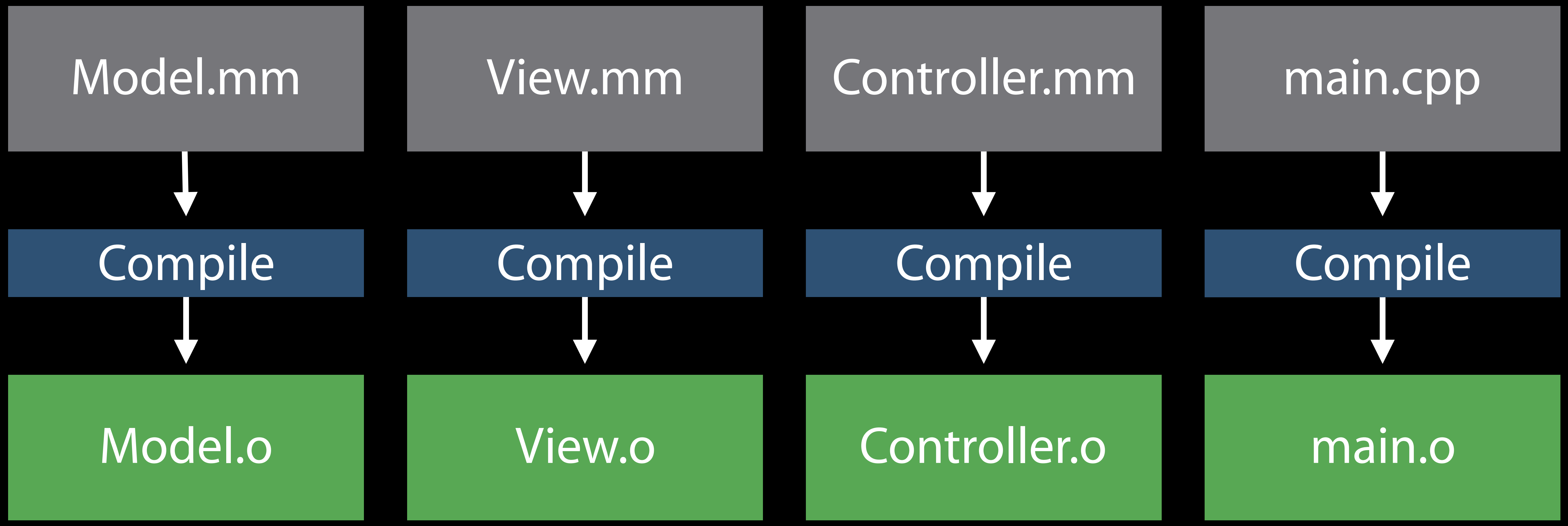
# LTO Compilation Model

Model.mm
View.mm
Controller.mm
main.cpp

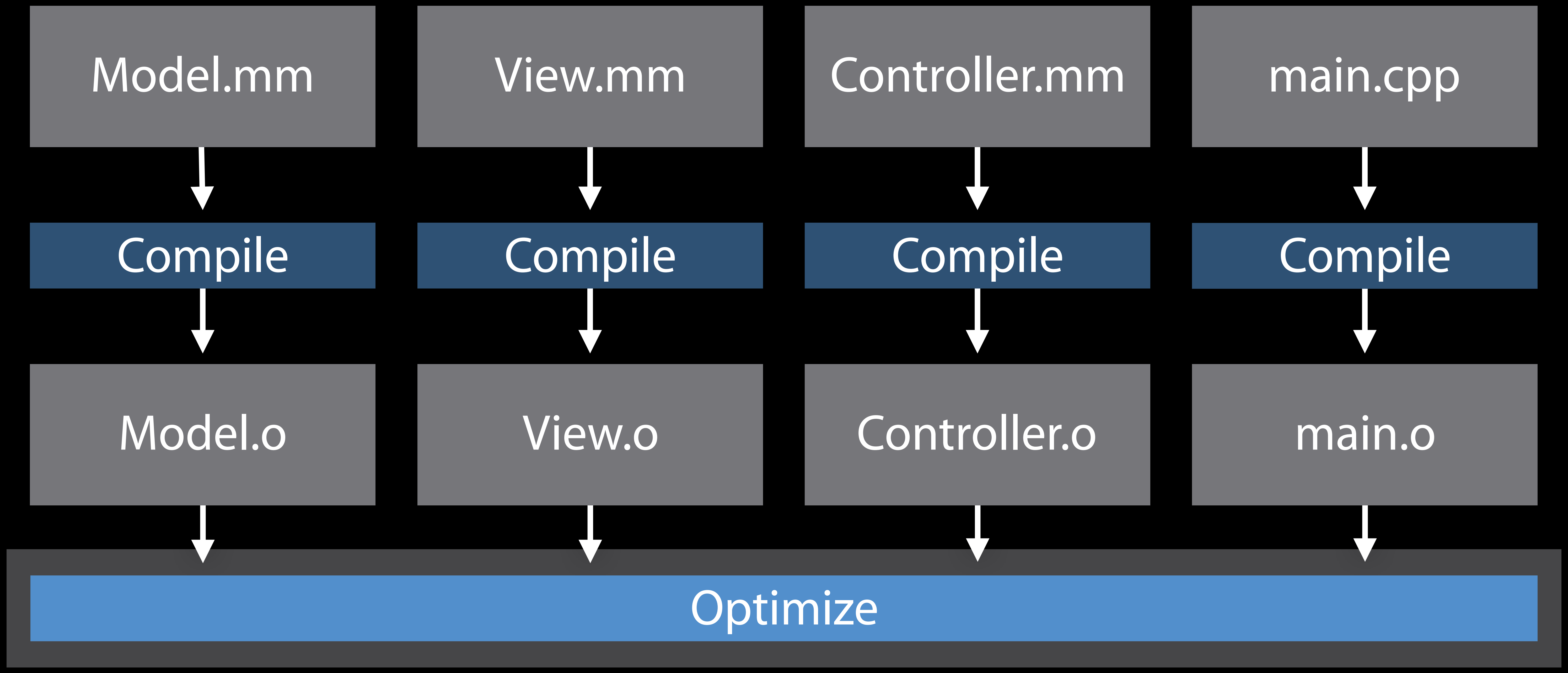
# LTO Compilation Model



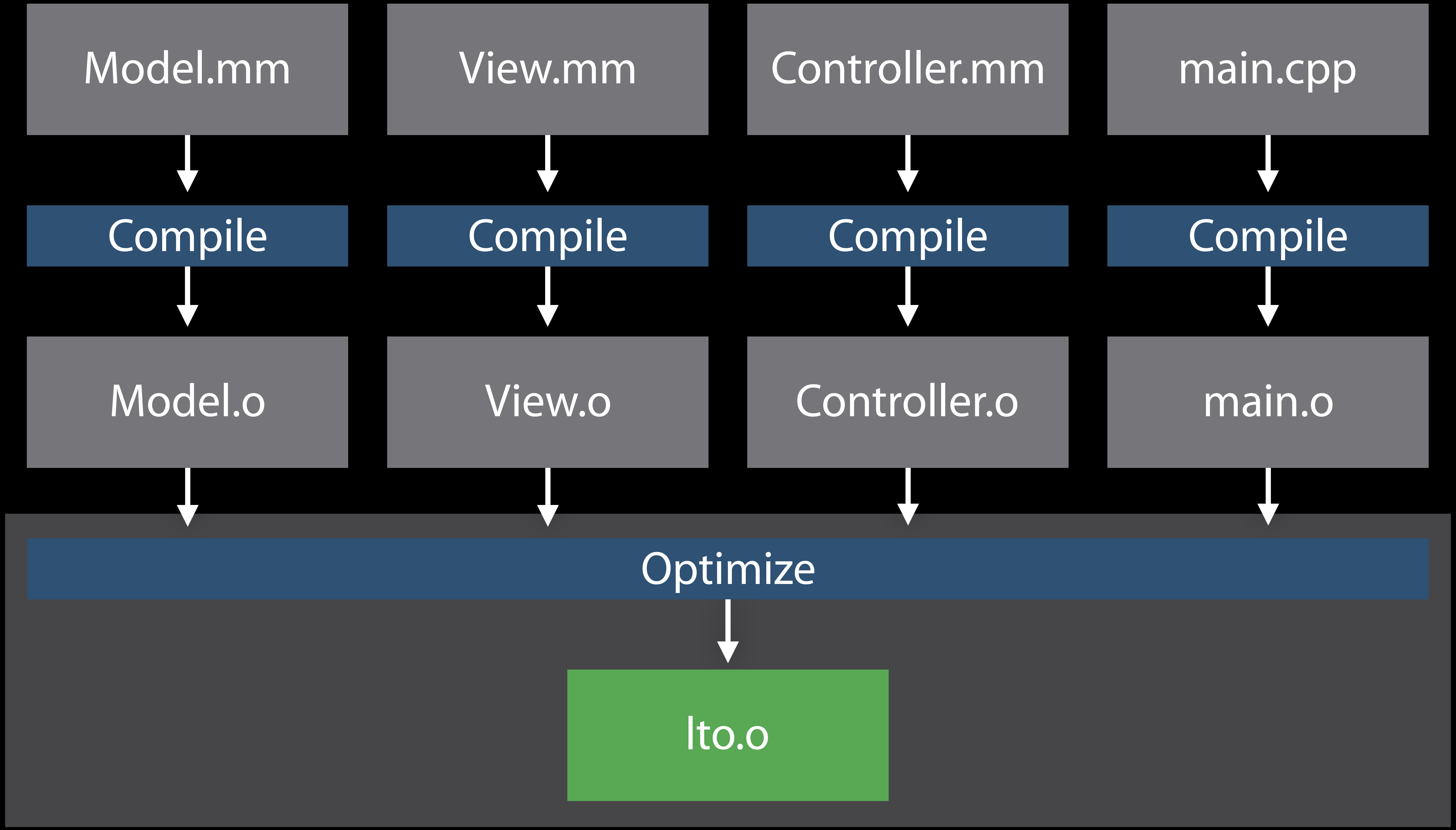
# LTO Compilation Model



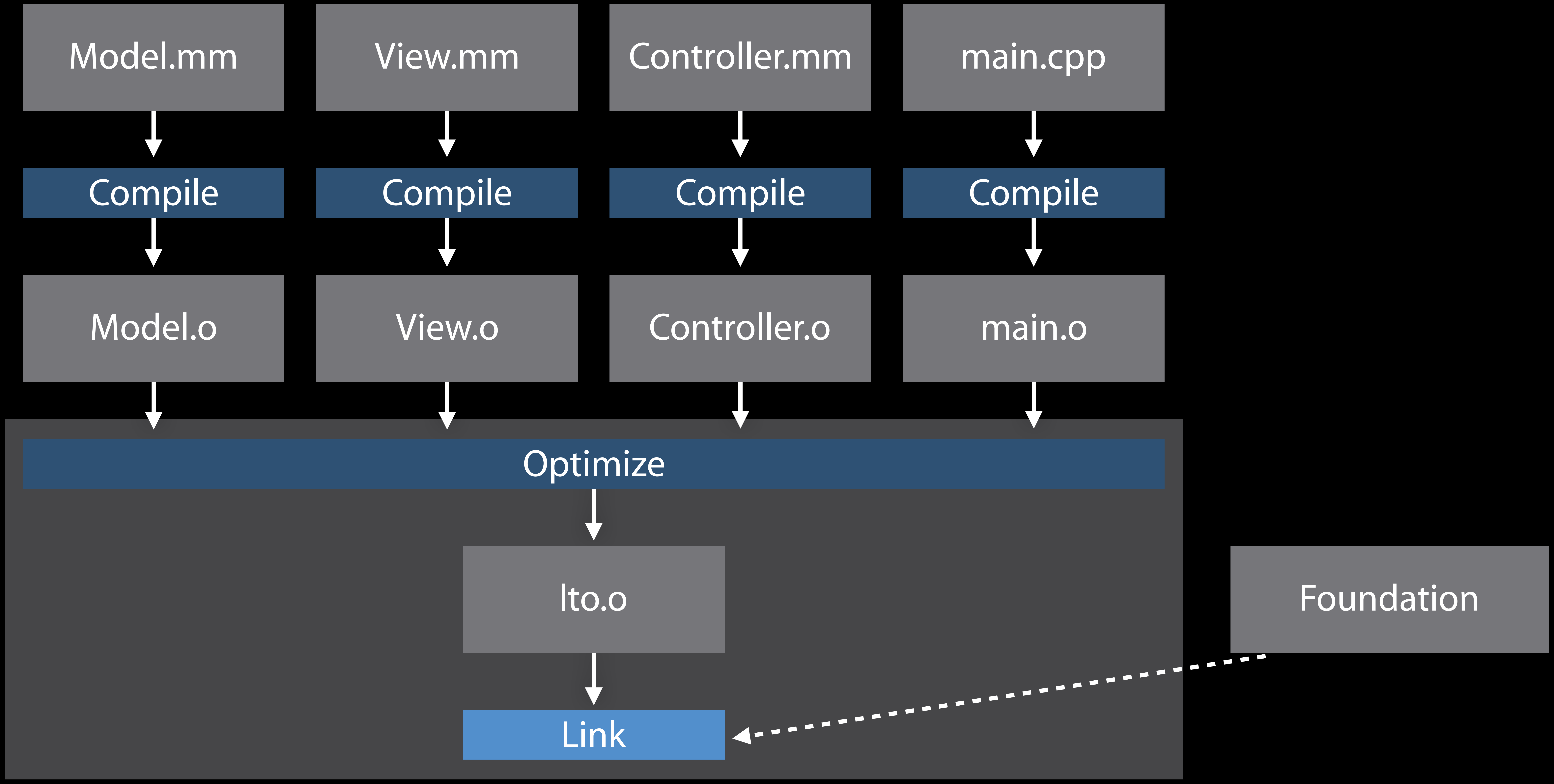
# LTO Compilation Model



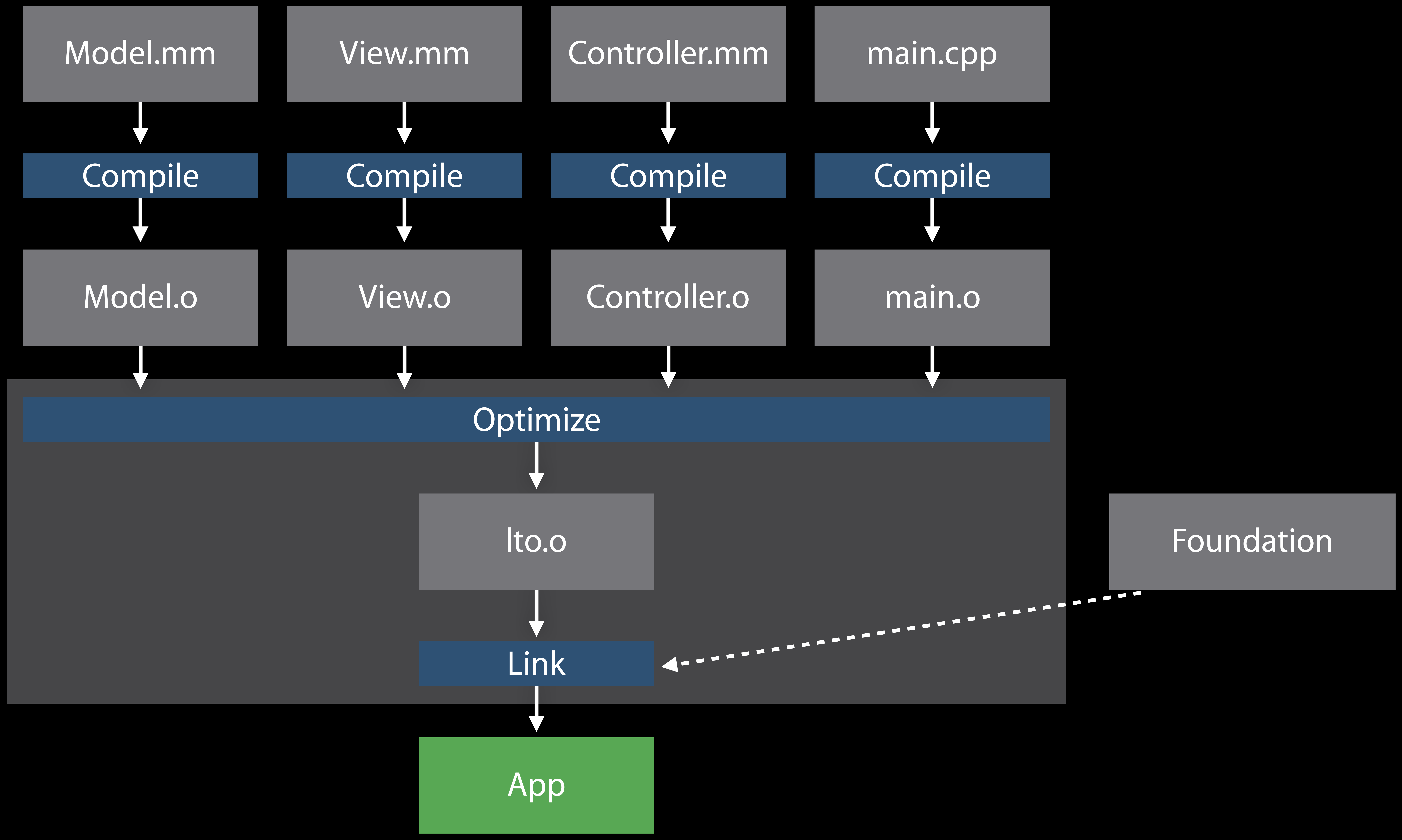
# LTO Compilation Model



# LTO Compilation Model



# LTO Compilation Model



# LTO Runtime Performance

## Maximize performance with LTO

Apple uses LTO extensively internally

## LTO Runtime Performance

### Maximize performance with LTO

Apple uses LTO extensively internally

- Typically 10% faster than executables from regular Release builds

## LTO Runtime Performance

### Maximize performance with LTO

Apple uses LTO extensively internally

- Typically 10% faster than executables from regular Release builds
- Multiplies with Profile Guided Optimization (PGO)

# LTO Runtime Performance

## Maximize performance with LTO

Apple uses LTO extensively internally

- Typically 10% faster than executables from regular Release builds
- Multiplies with Profile Guided Optimization (PGO)
- Reduces code size when optimizing for size

## LTO Compile Time Tradeoff

LTO trades compile time for runtime performance

## LTO Compile Time Tradeoff

LTO trades compile time for runtime performance

- Large memory requirements

# LTO Compile Time Tradeoff

LTO trades compile time for runtime performance

- Large memory requirements
- Optimizations are not done in parallel

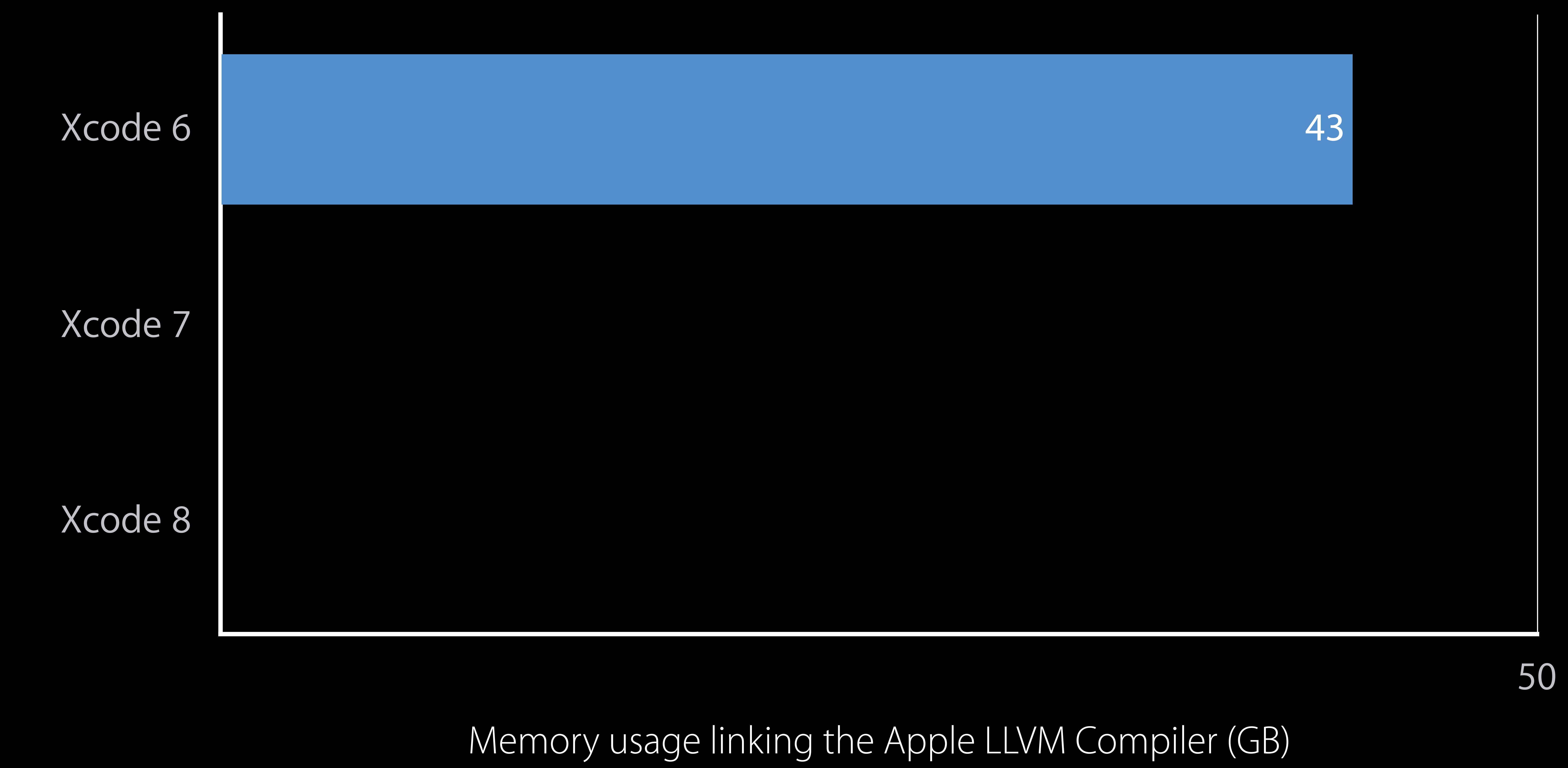
# LTO Compile Time Tradeoff

LTO trades compile time for runtime performance

- Large memory requirements
- Optimizations are not done in parallel
- Incremental builds repeat all the work

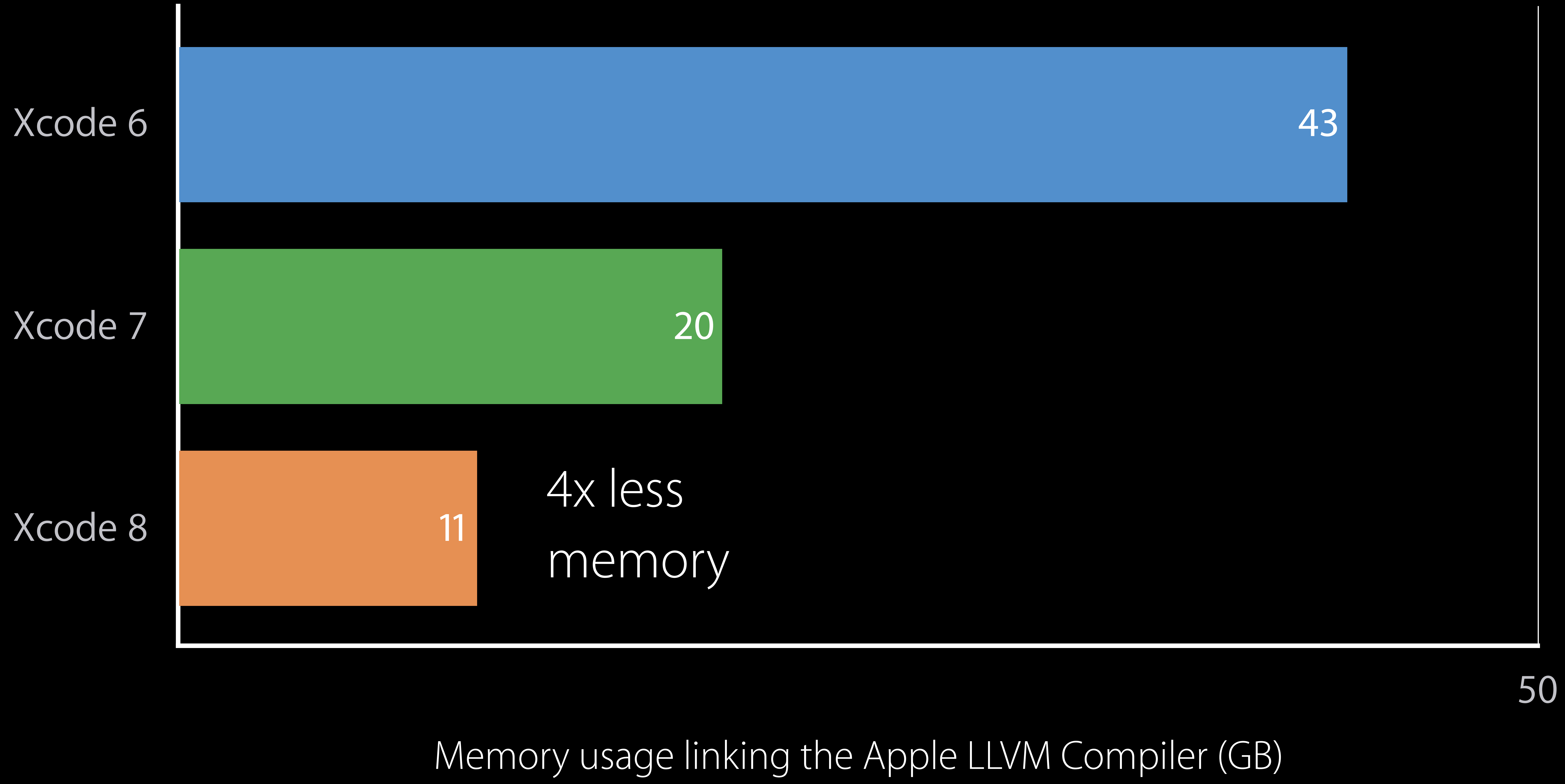
## LTO Memory Usage—Full Debug Info

Large C++ project with **–g**



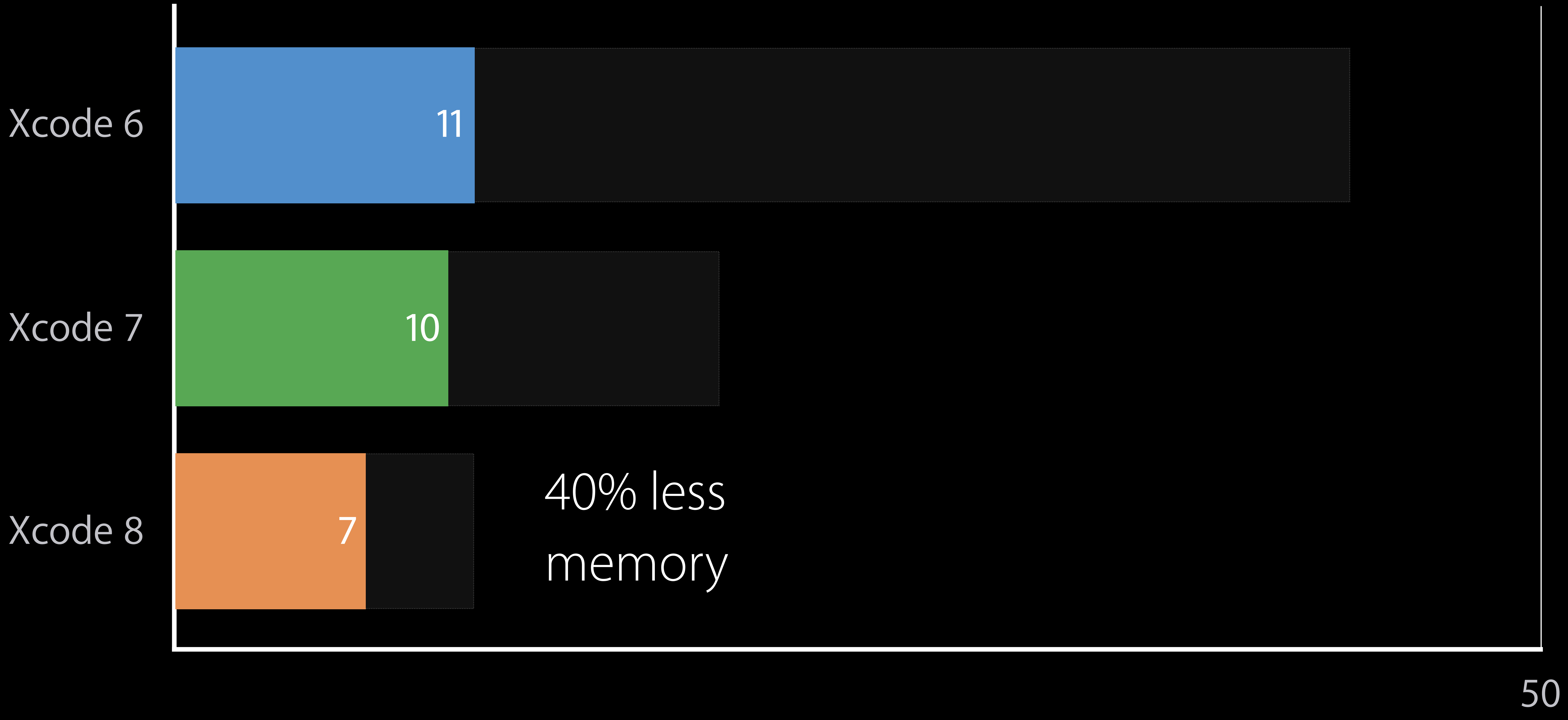
## LTO Memory Usage—Full Debug Info

Large C++ project with **–g**



# LTO Memory Usage—Line Tables Only

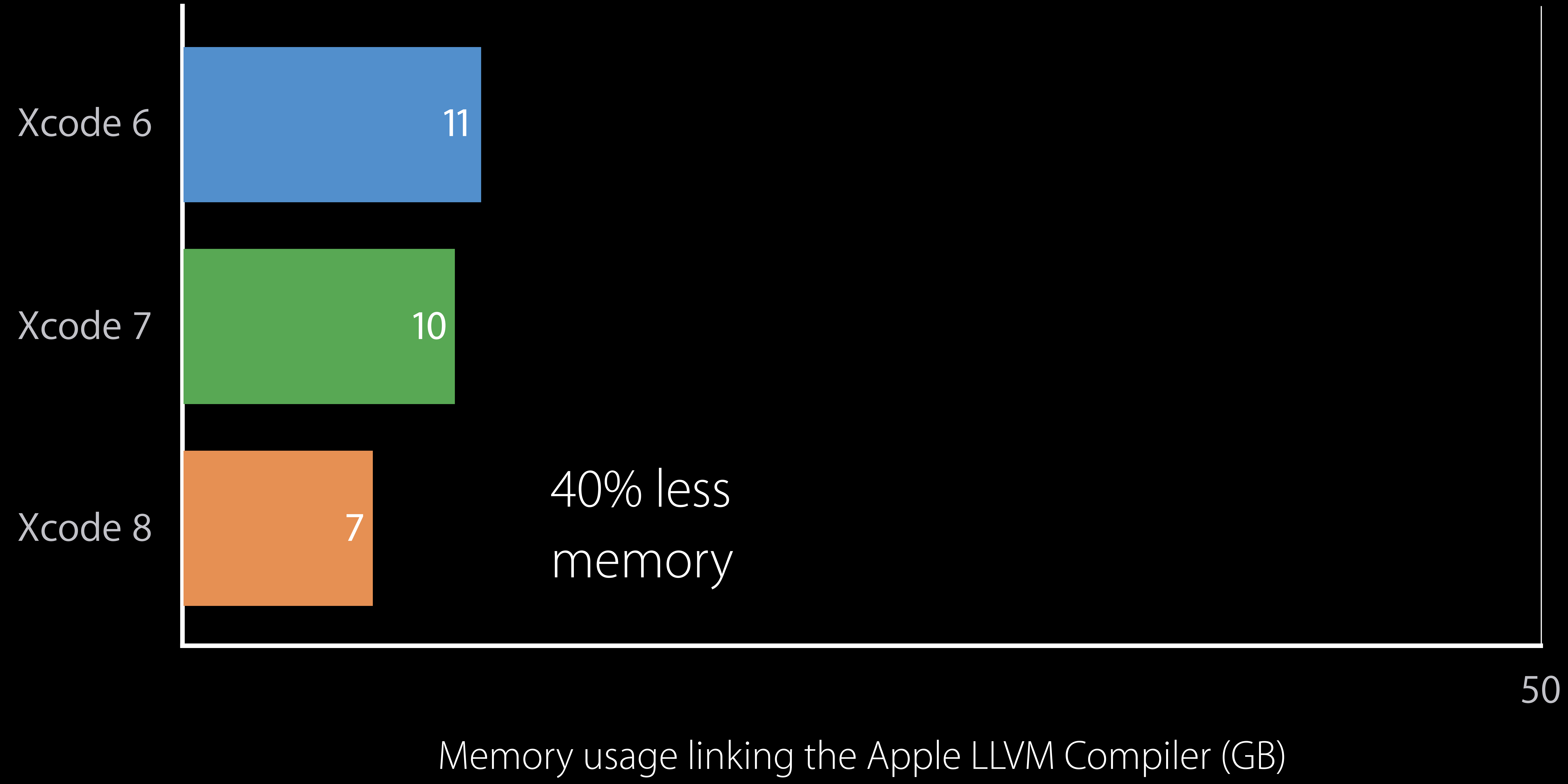
## Large C++ project with `-gline-tables-only`



Memory usage linking the Apple LLVM Compiler (GB)

## LTO Memory Usage—Line Tables Only

Large C++ project with `-gline-tables-only`



## Incremental LTO

NEW

New model for link-time optimization that scales with your system

# Incremental LTO

NEW

New model for link-time optimization that scales with your system

- Analysis and inlining without merging object files

# Incremental LTO

NEW

New model for link-time optimization that scales with your system

- Analysis and inlining without merging object files
- Optimizations run in parallel

## Incremental LTO

NEW

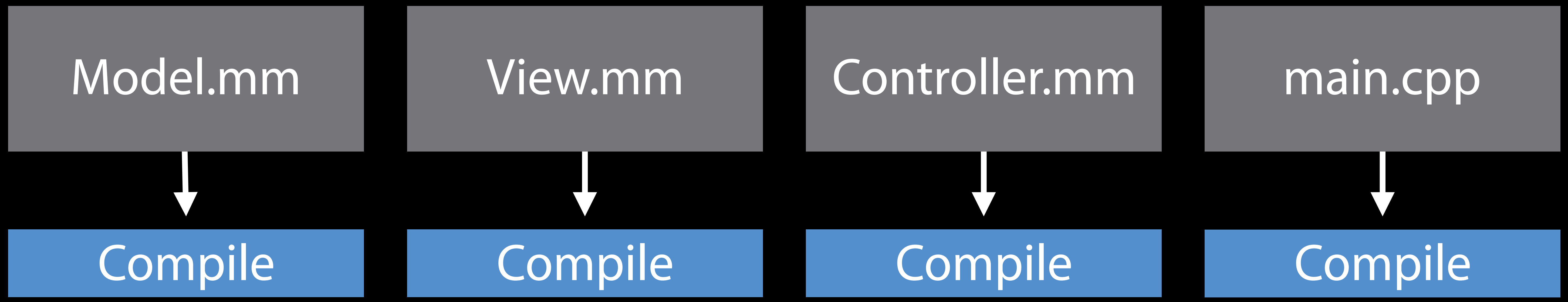
New model for link-time optimization that scales with your system

- Analysis and inlining without merging object files
- Optimizations run in parallel
- Linker cache for fast incremental builds

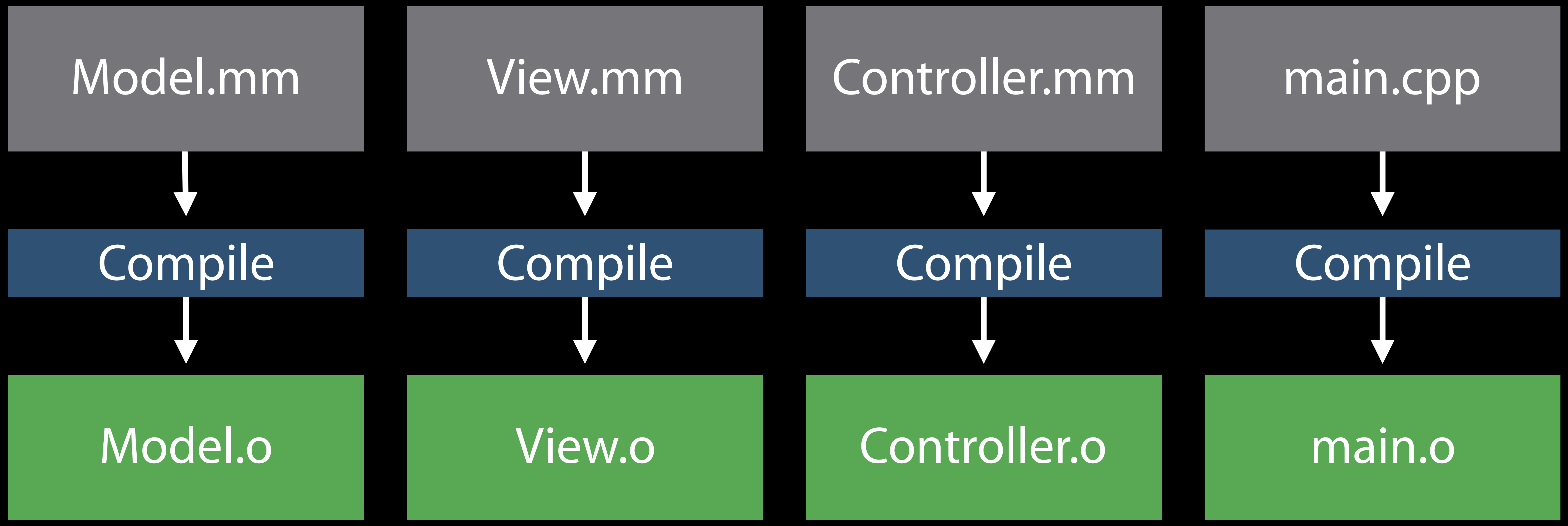
# Incremental LTO Compilation Model

Model.mm
View.mm
Controller.mm
main.cpp

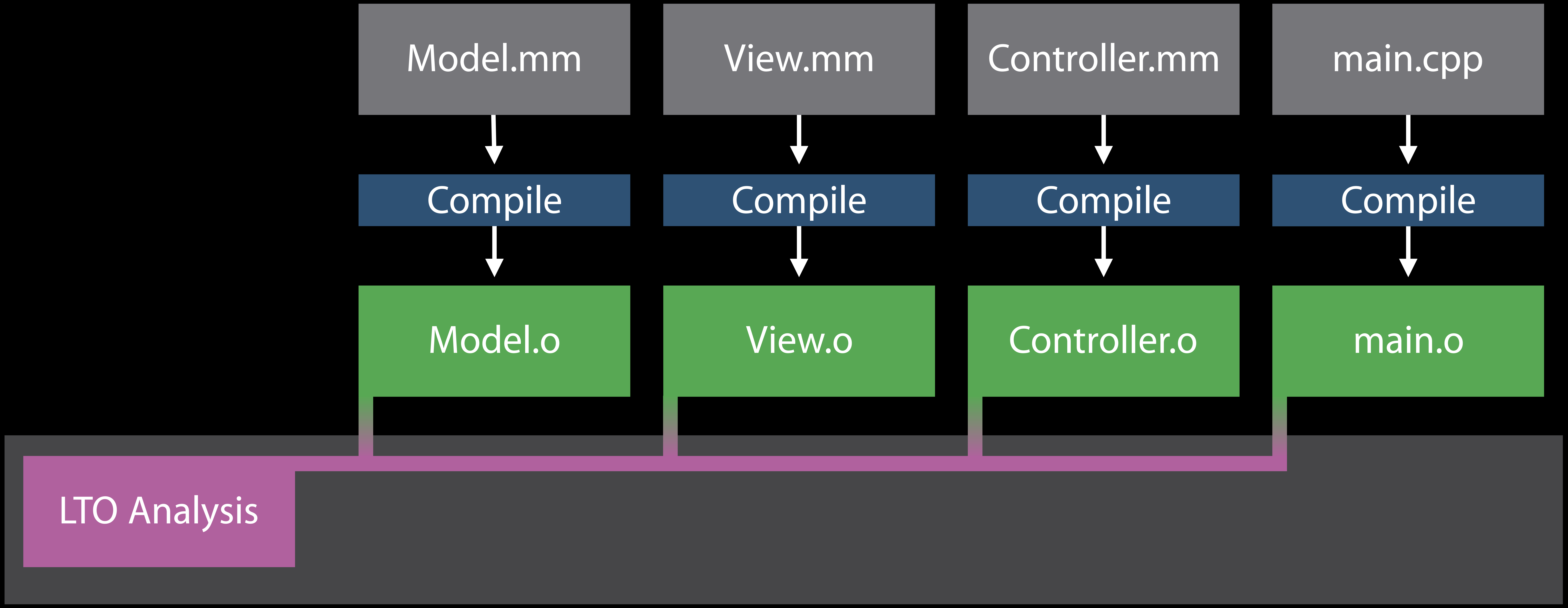
# Incremental LTO Compilation Model



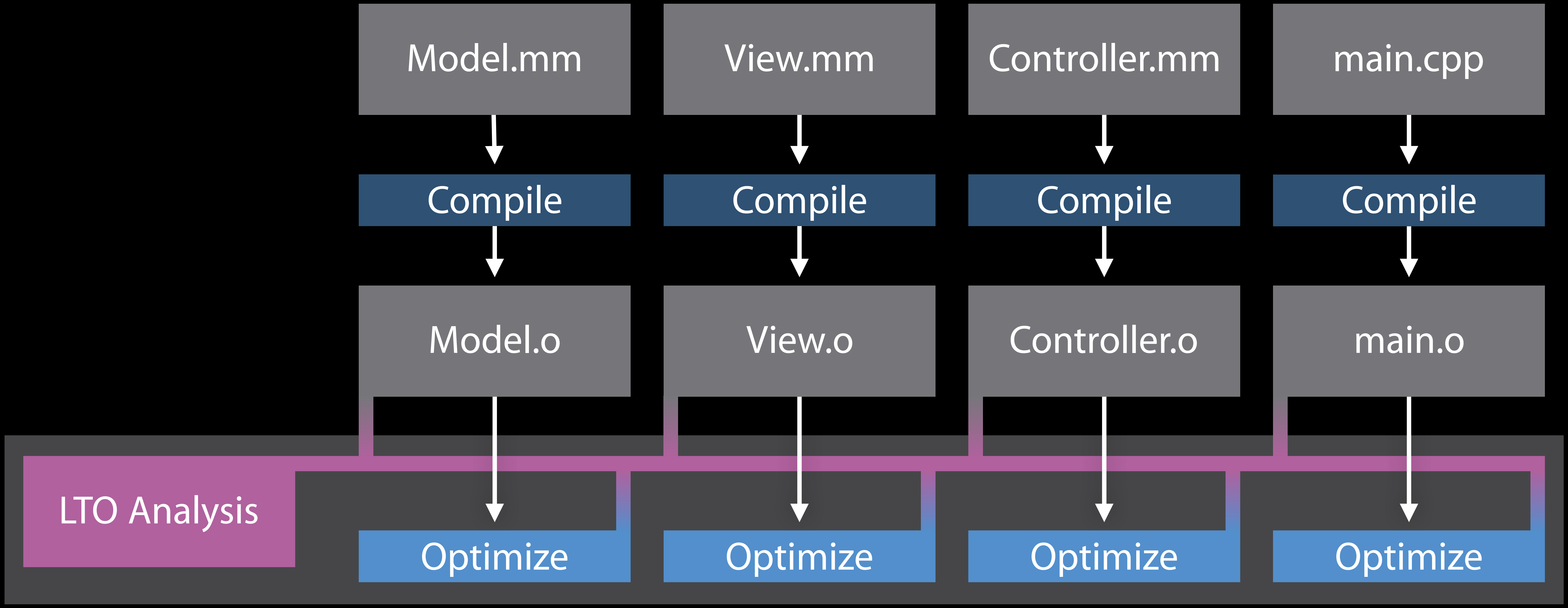
# Incremental LTO Compilation Model



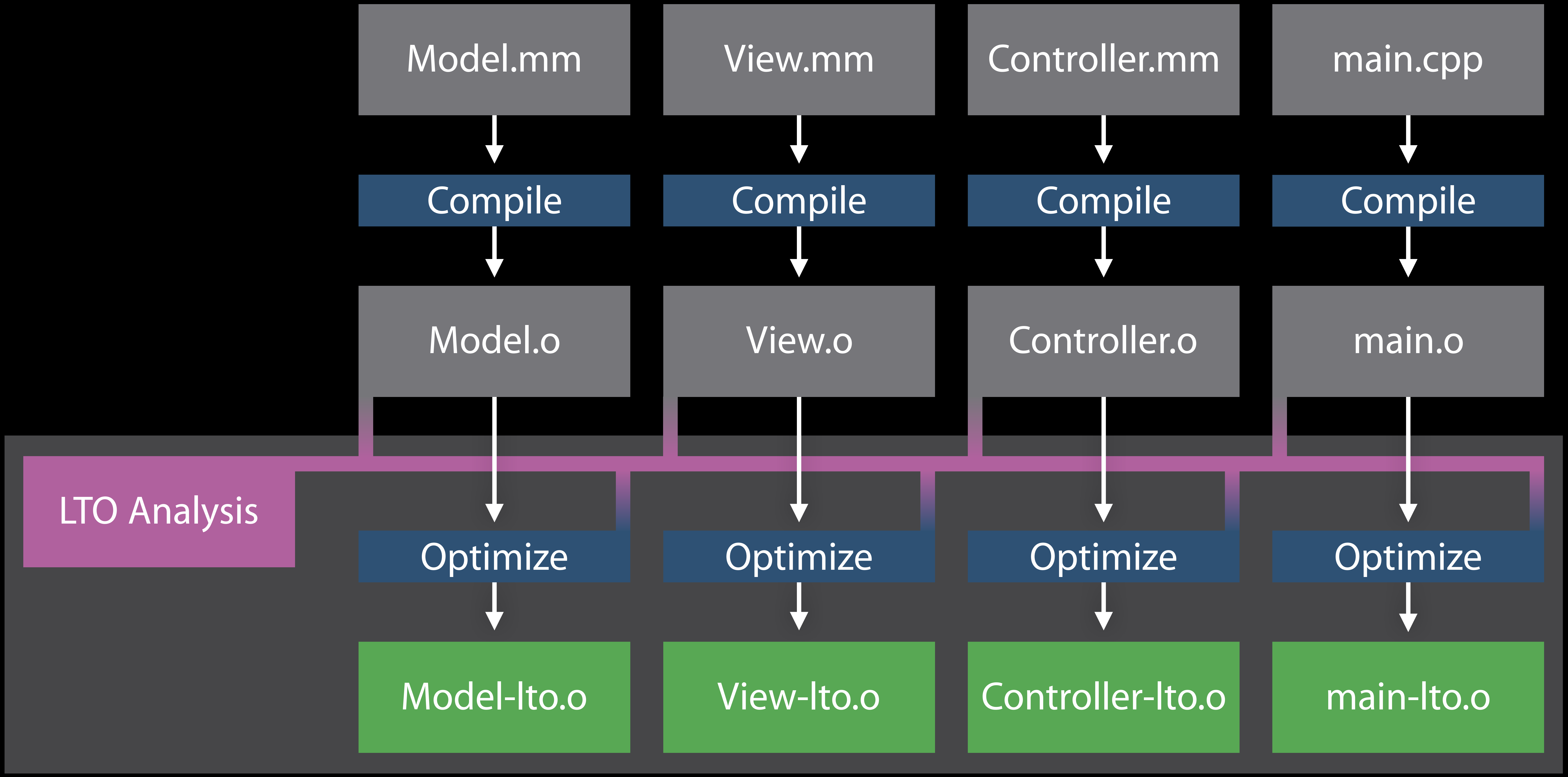
# Incremental LTO Compilation Model



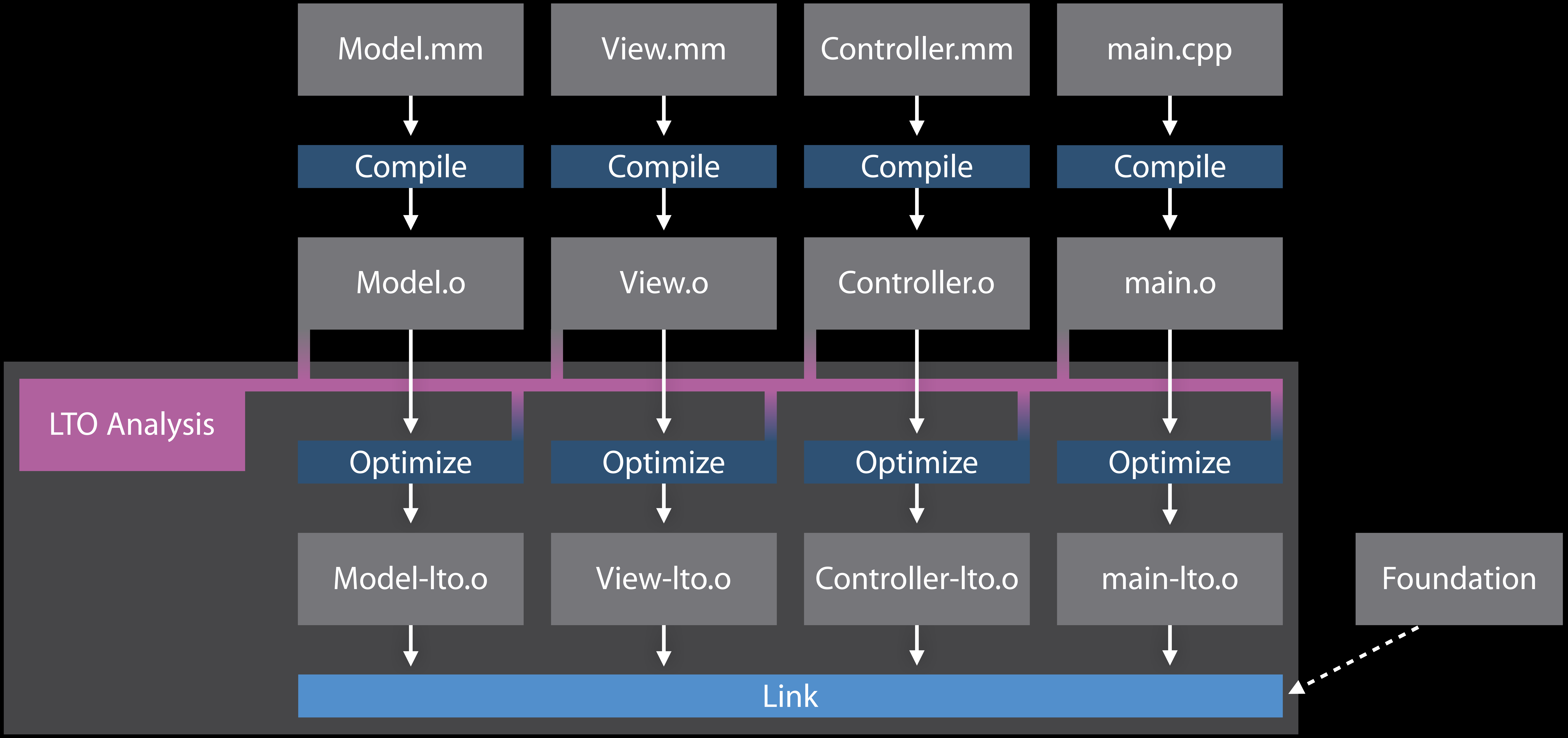
# Incremental LTO Compilation Model



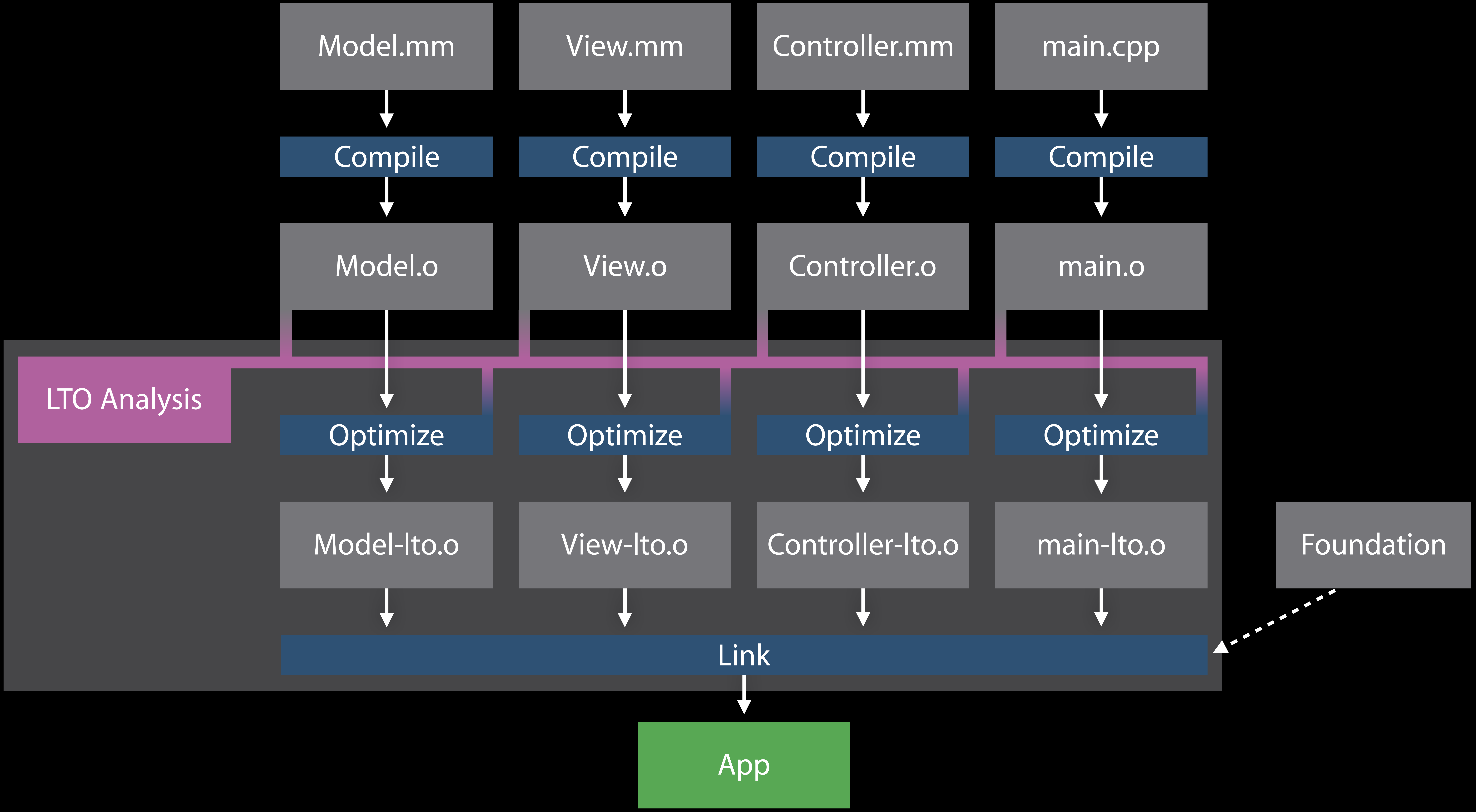
# Incremental LTO Compilation Model



# Incremental LTO Compilation Model

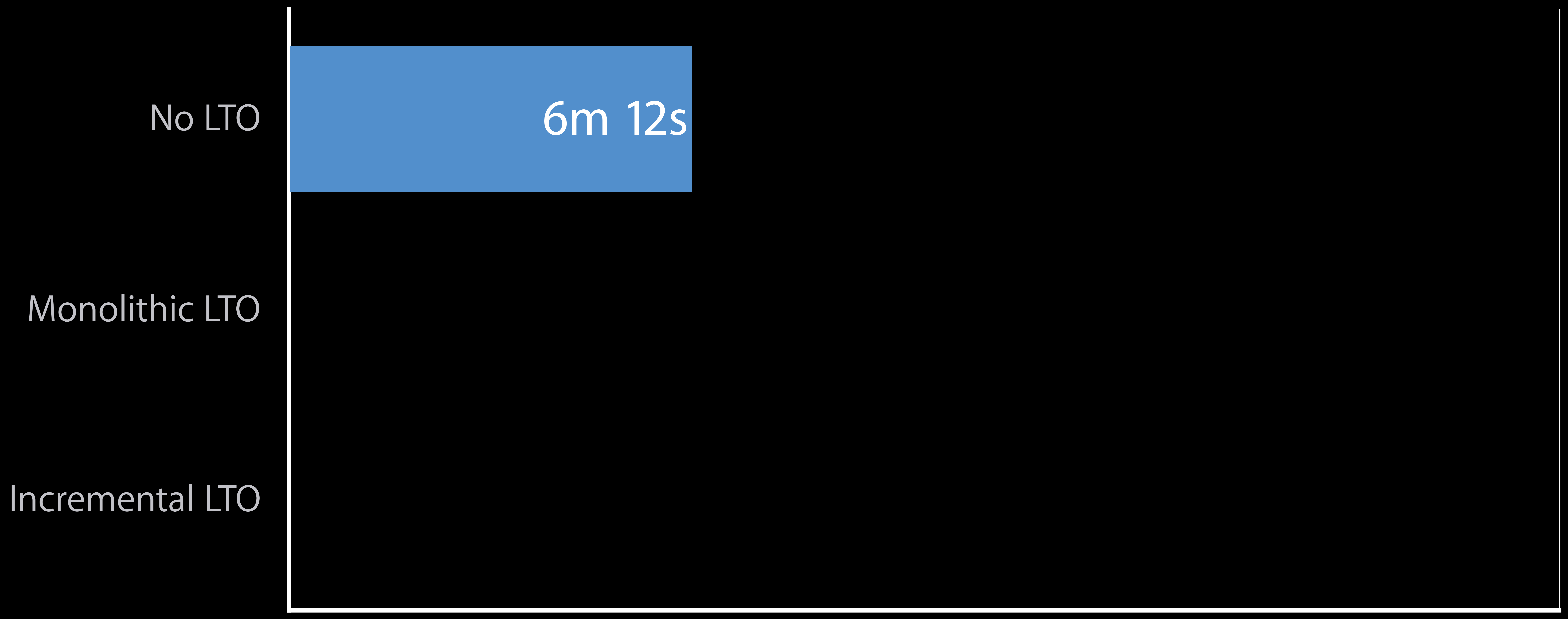


# Incremental LTO Compilation Model



## Time to Build a Large C++ Project

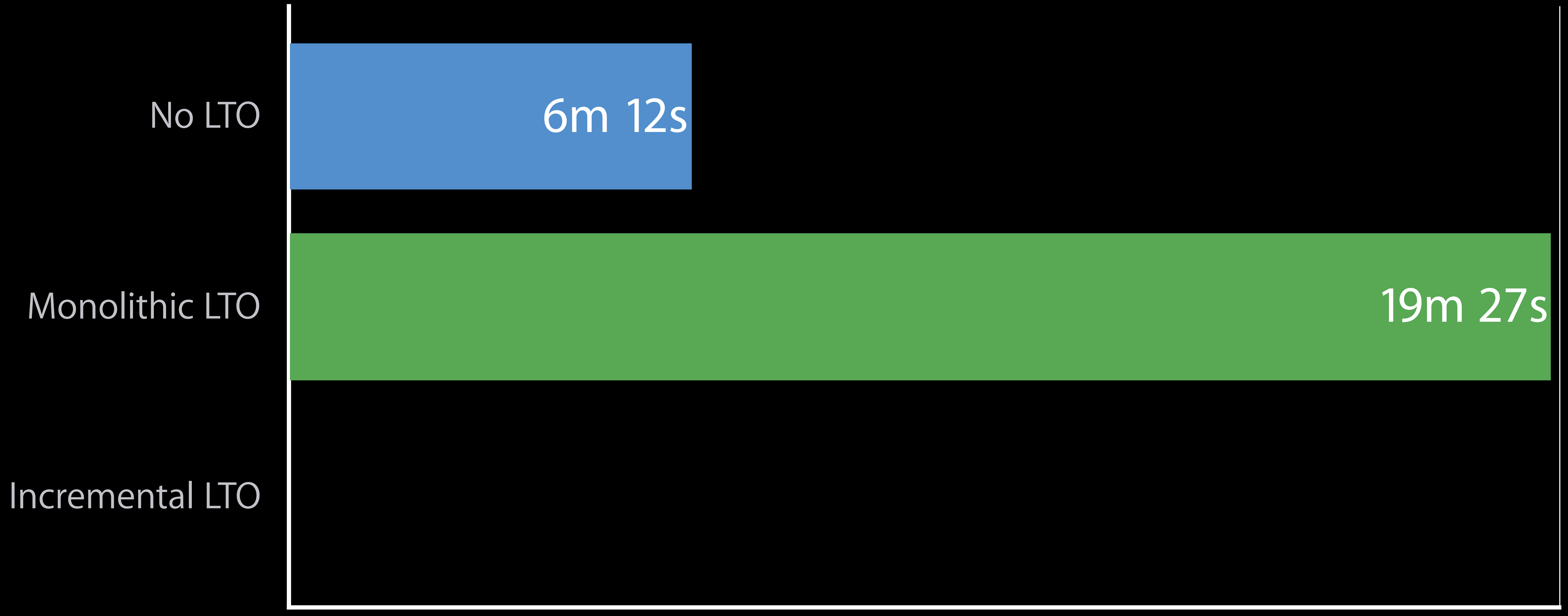
Smaller is better



Time for full build of Apple LLVM Compiler

# Time to Build a Large C++ Project

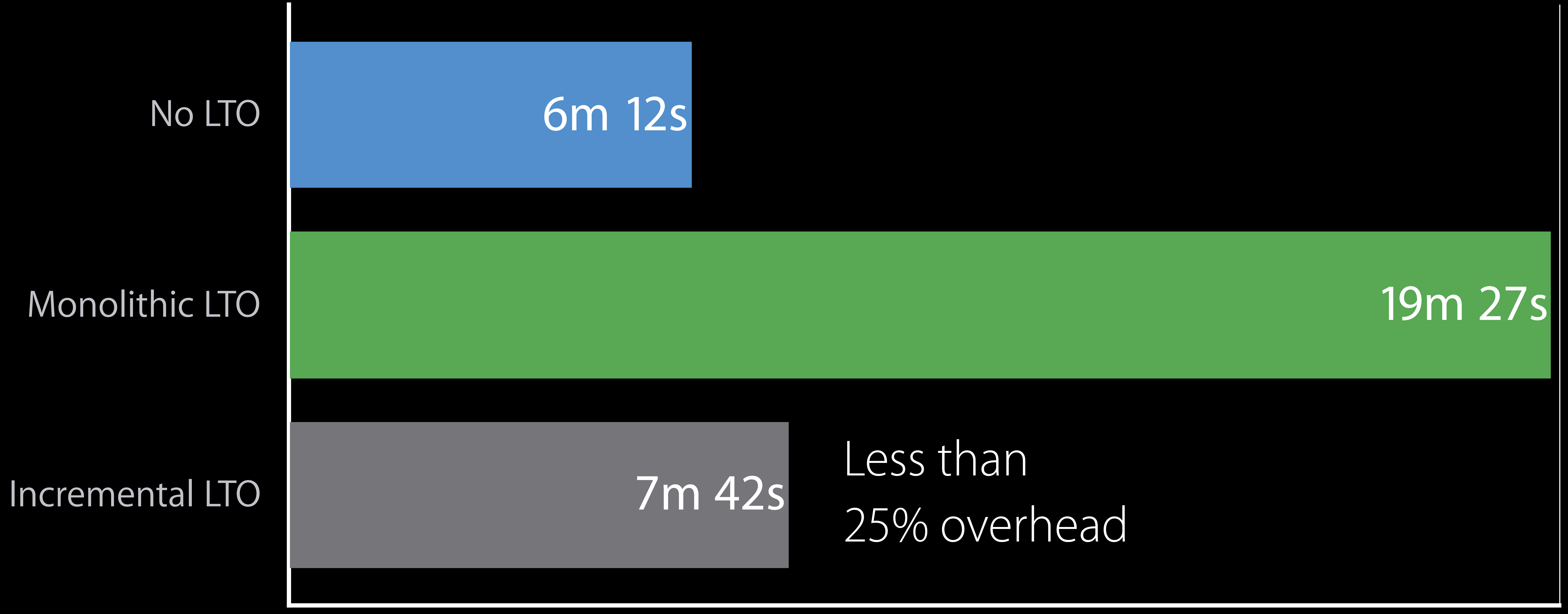
Smaller is better



Time for full build of Apple LLVM Compiler

# Time to Build a Large C++ Project

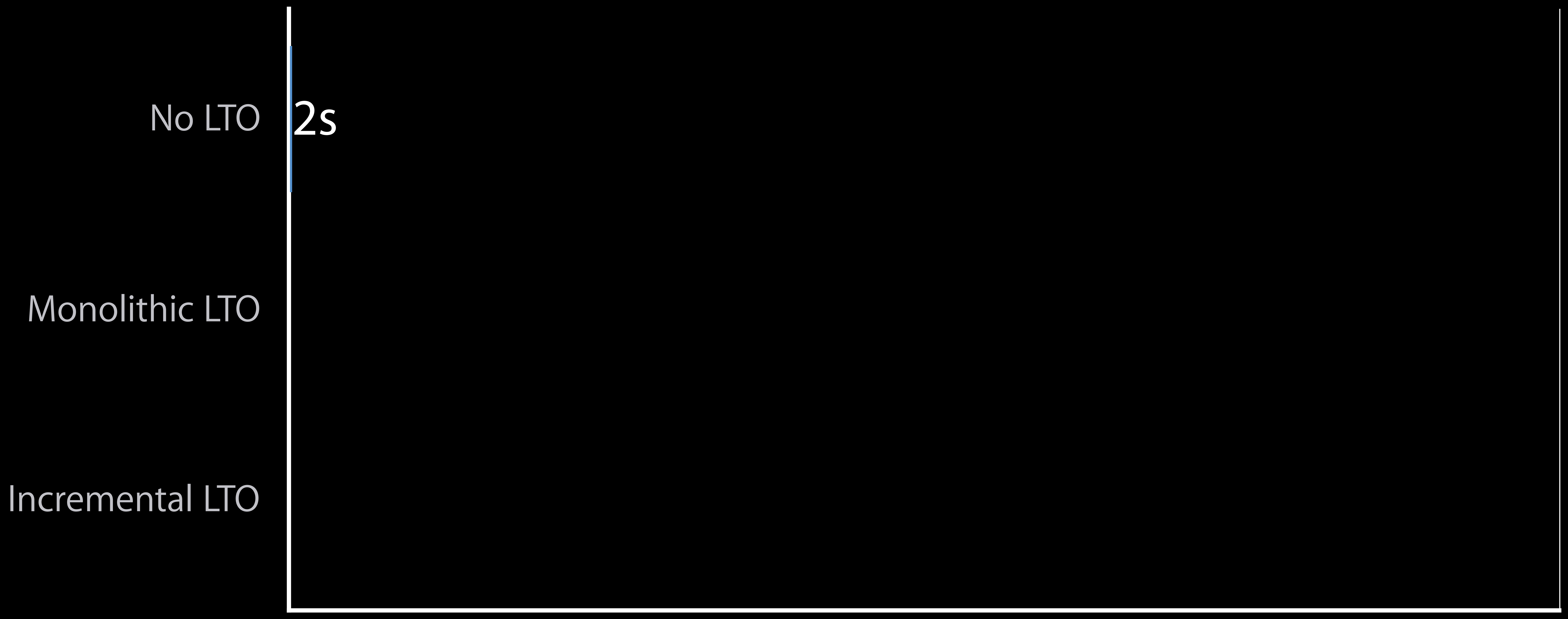
Smaller is better



Time for full build of Apple LLVM Compiler

# Time to Link a Large C++ Project

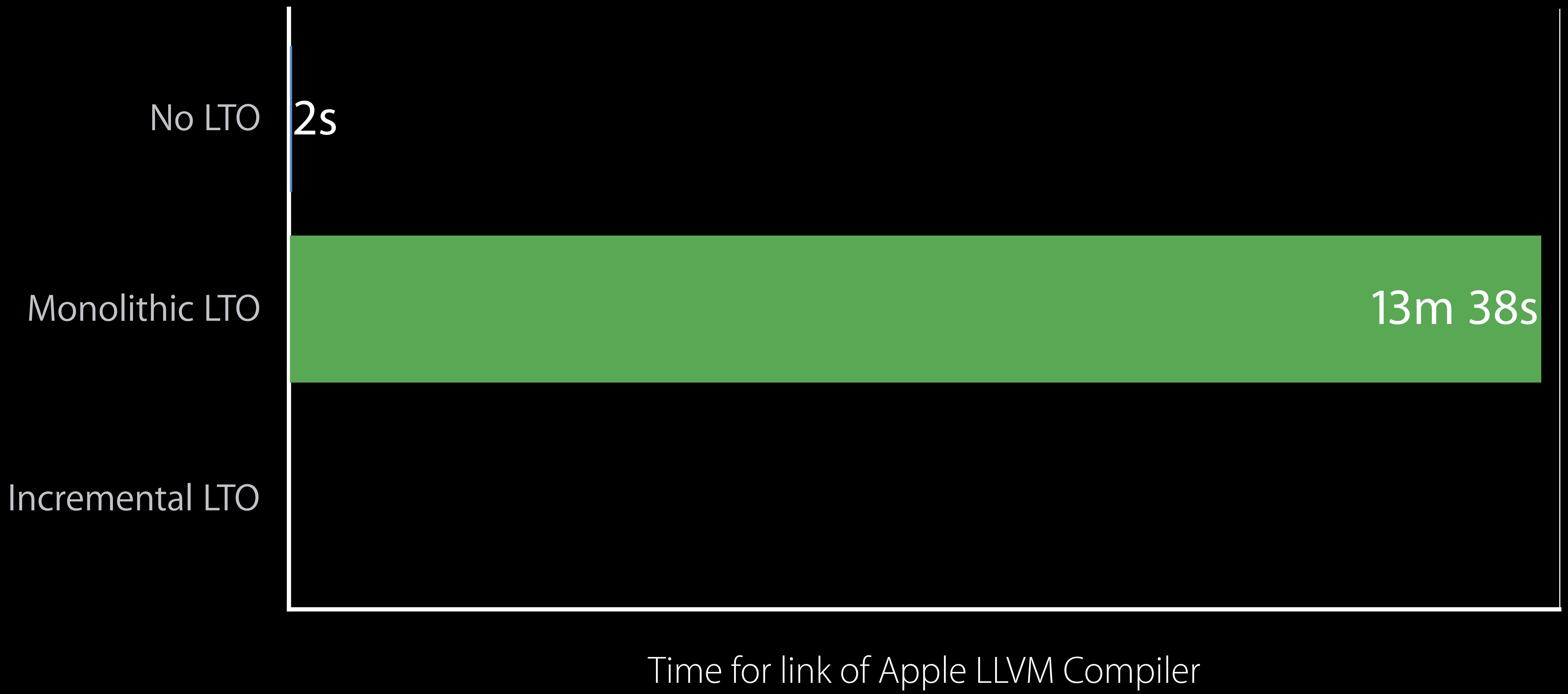
Smaller is better



Time for link of Apple LLVM Compiler

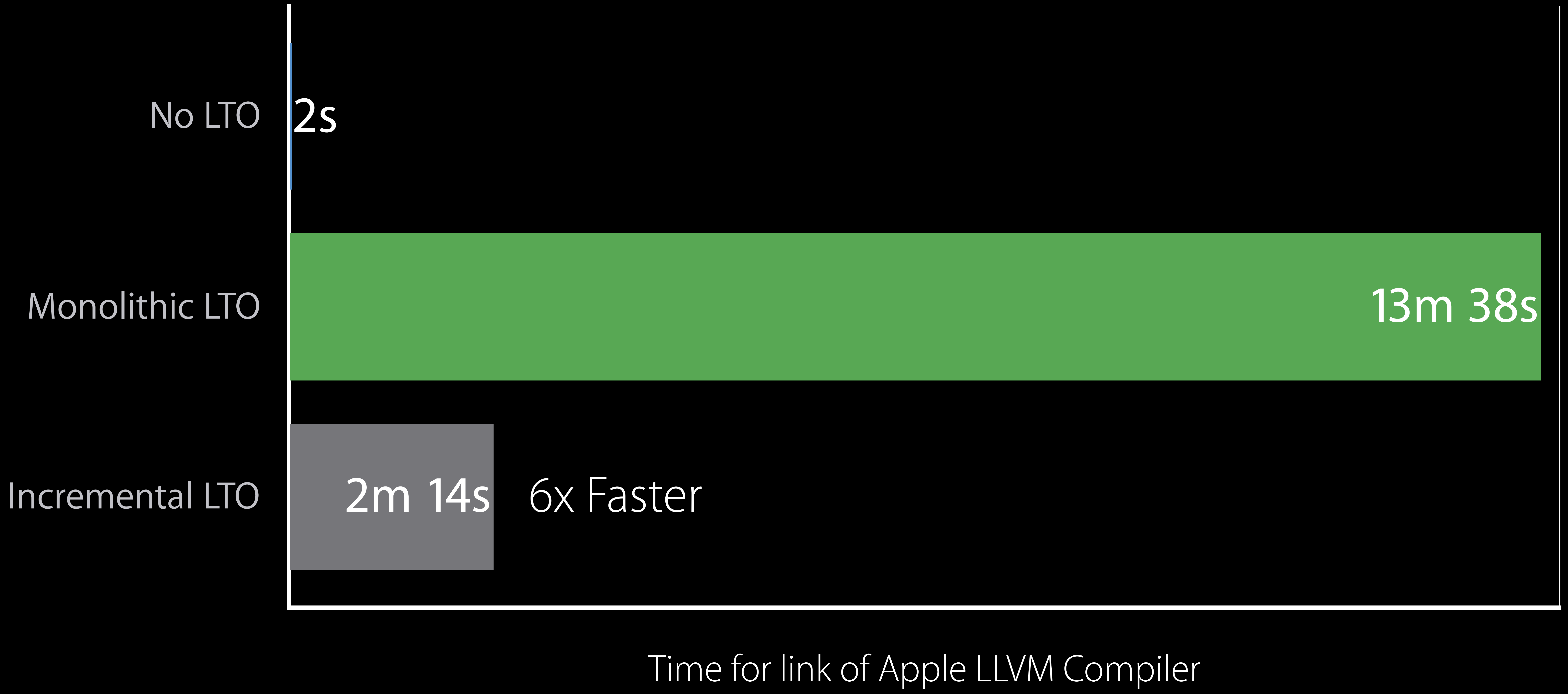
# Time to Link a Large C++ Project

## Smaller is better



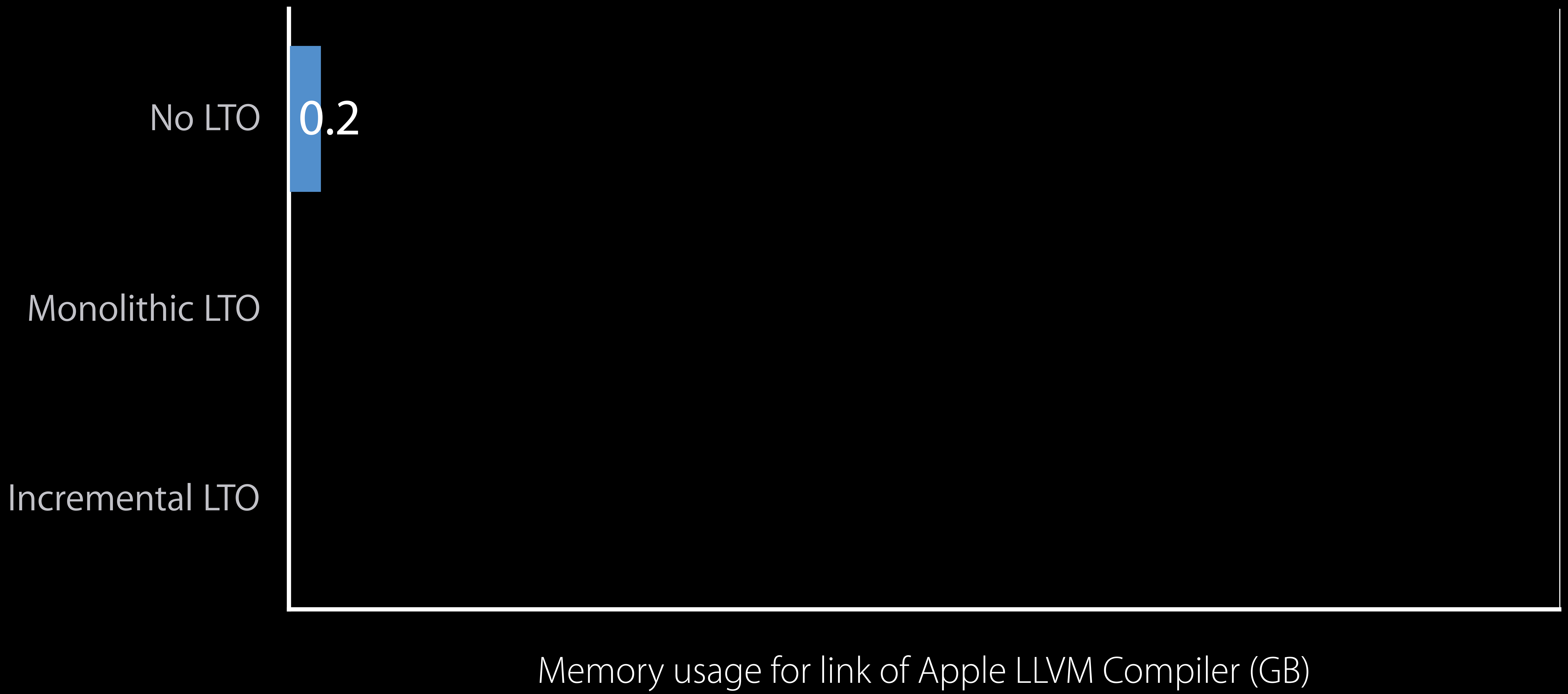
## Time to Link a Large C++ Project

Smaller is better



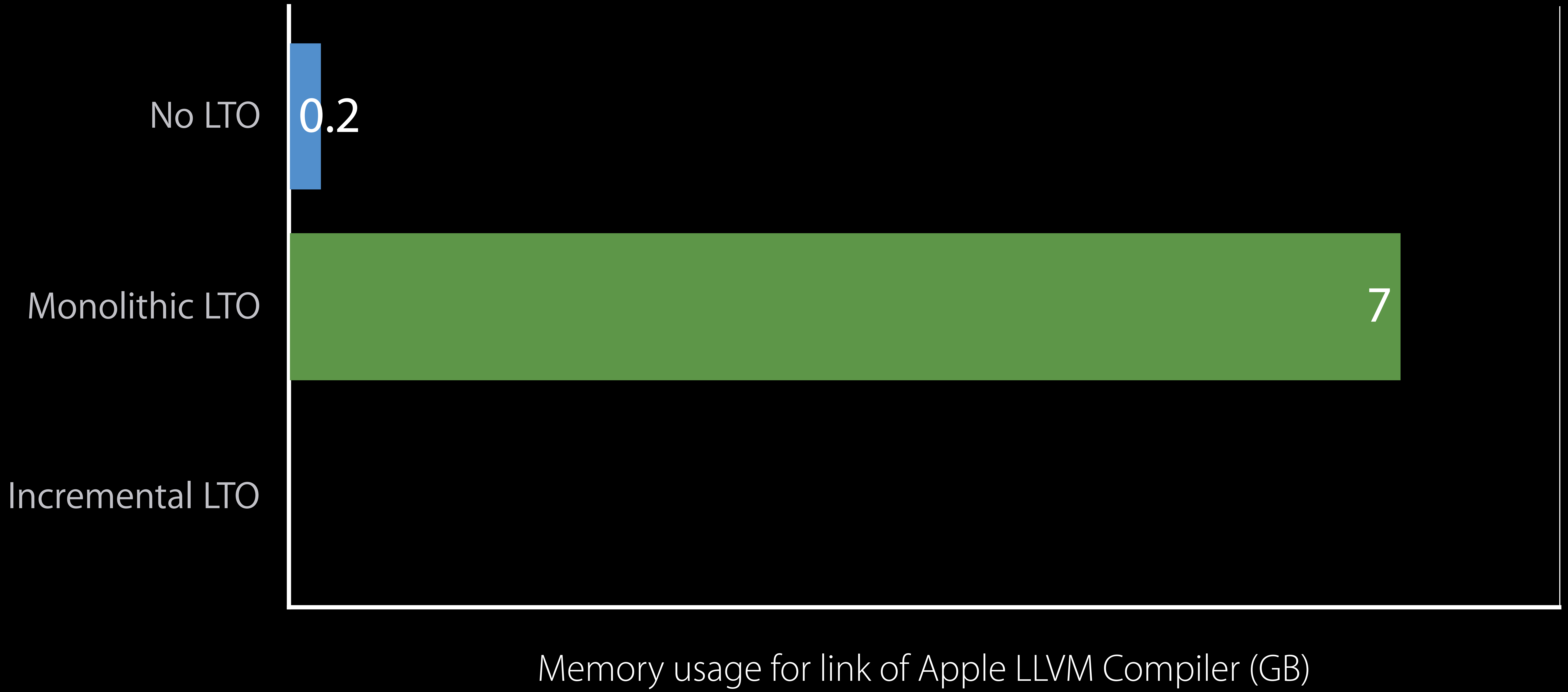
## Memory to Link a Large C++ Project

Smaller is better



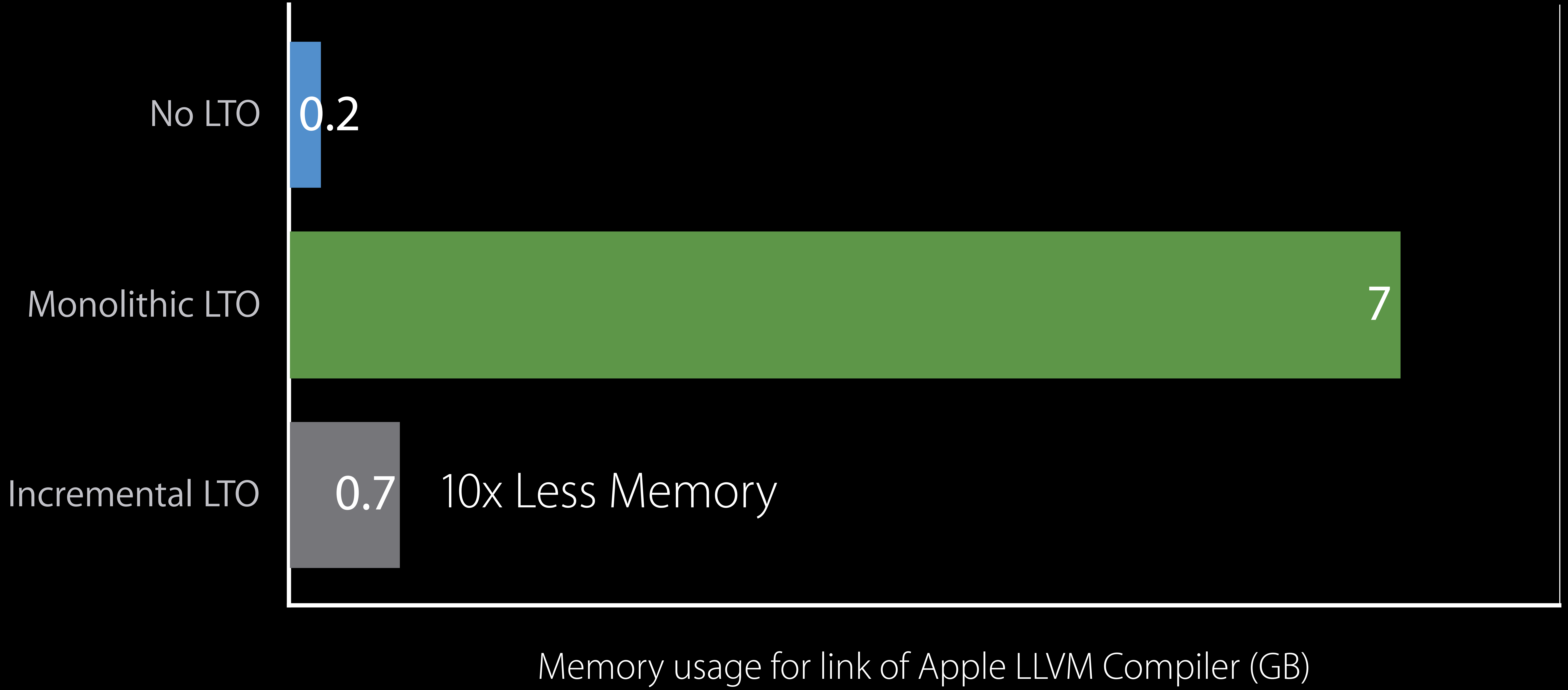
# Memory to Link a Large C++ Project

Smaller is better

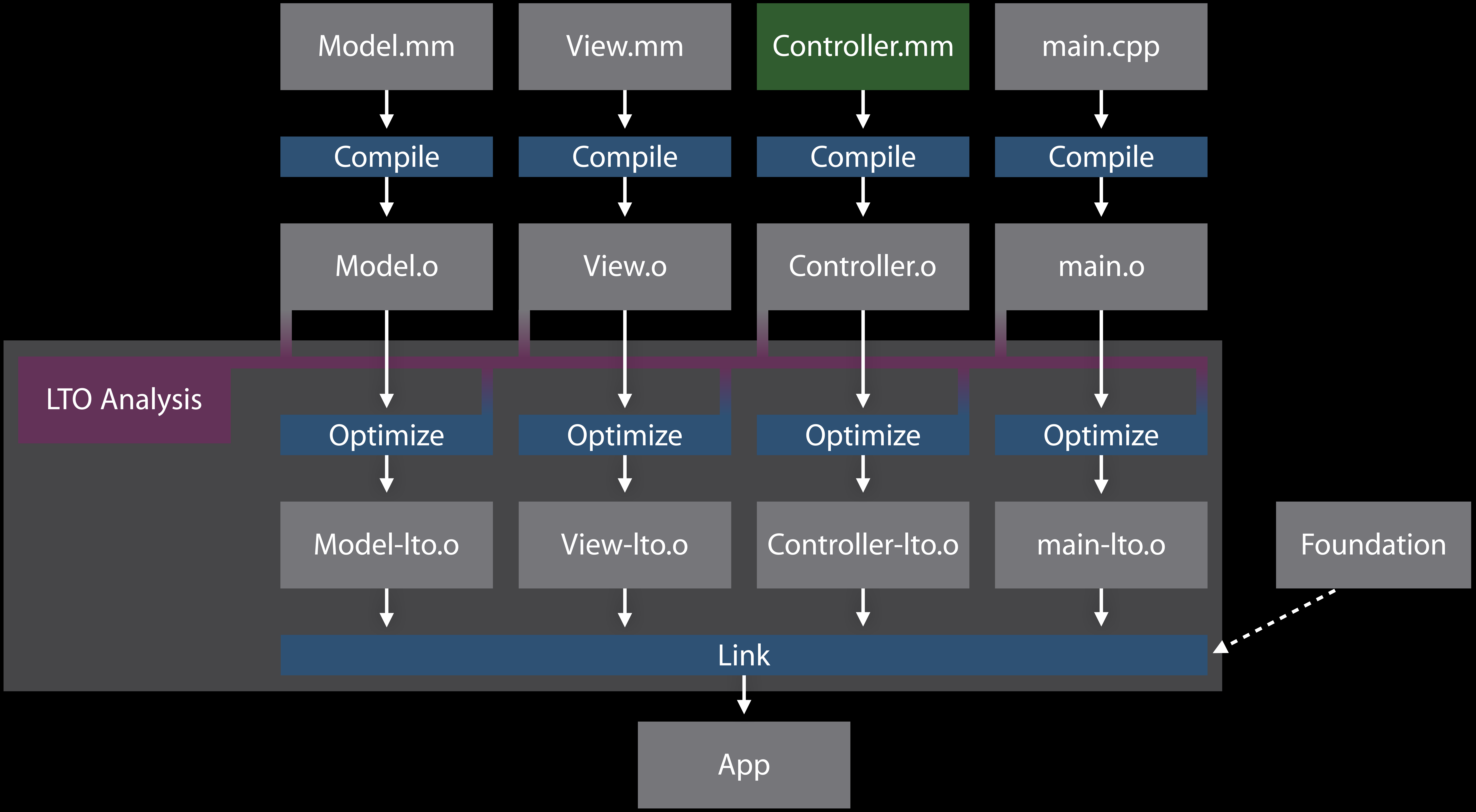


## Memory to Link a Large C++ Project

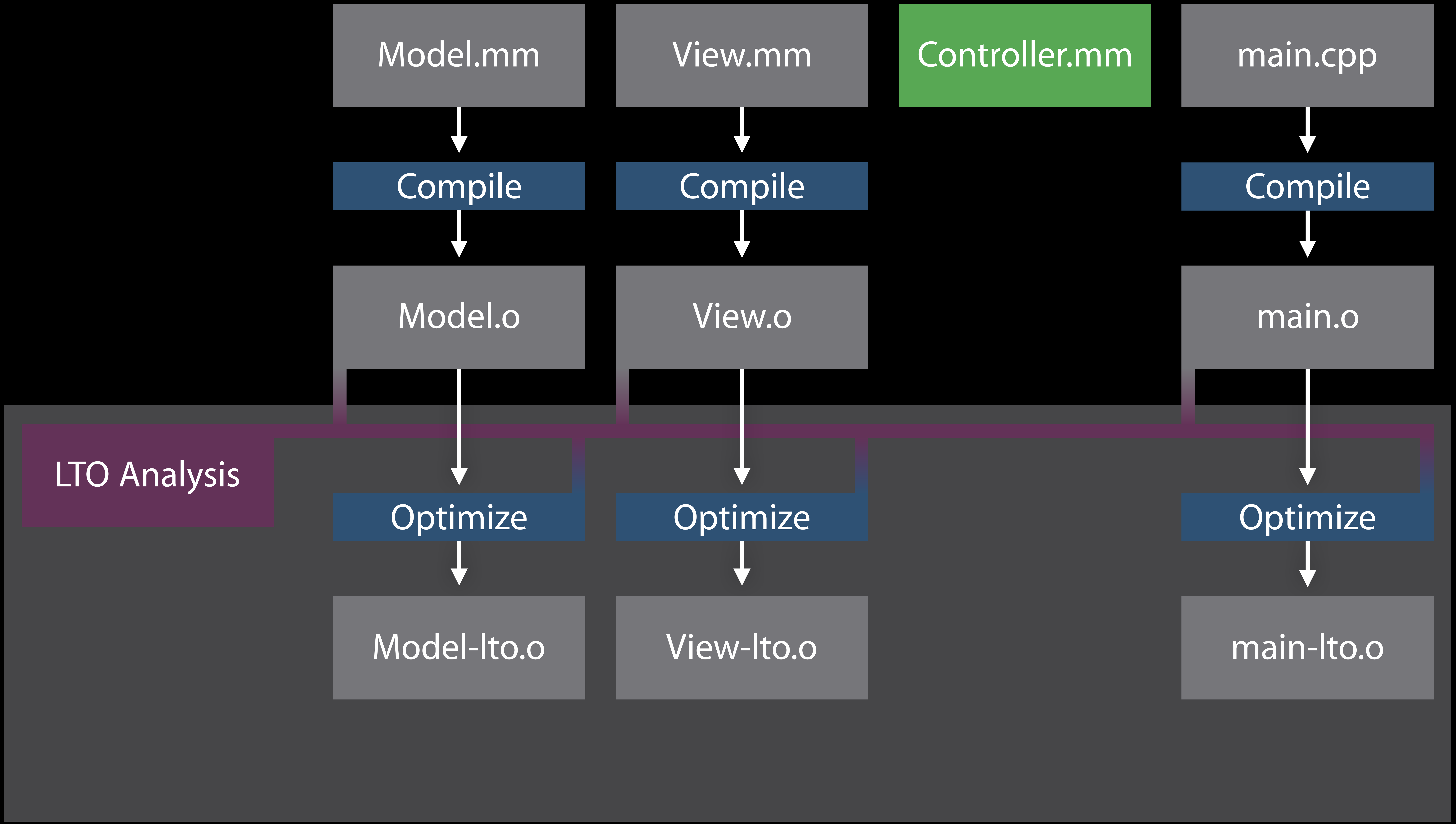
Smaller is better



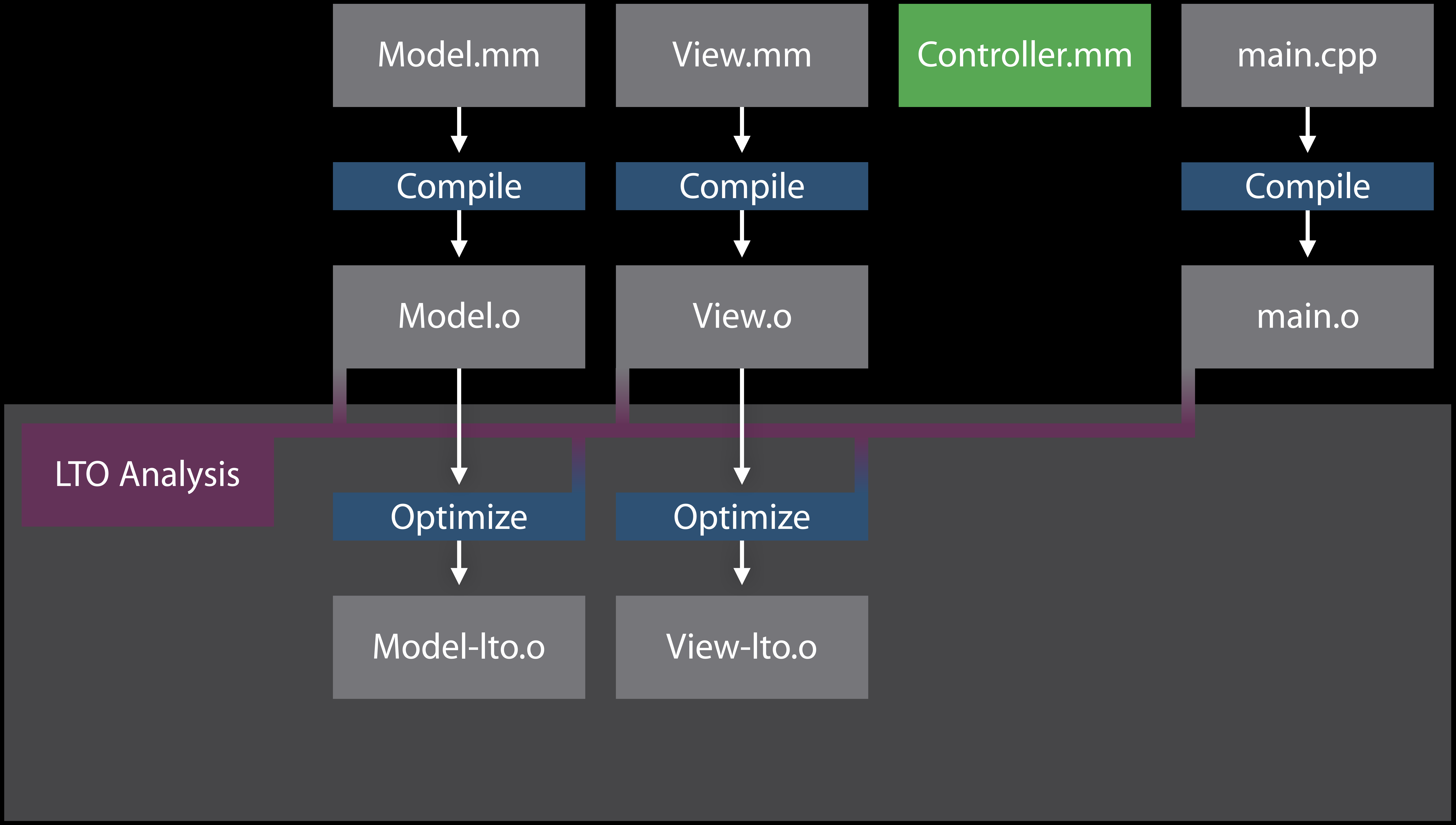
# Example of Incremental Build



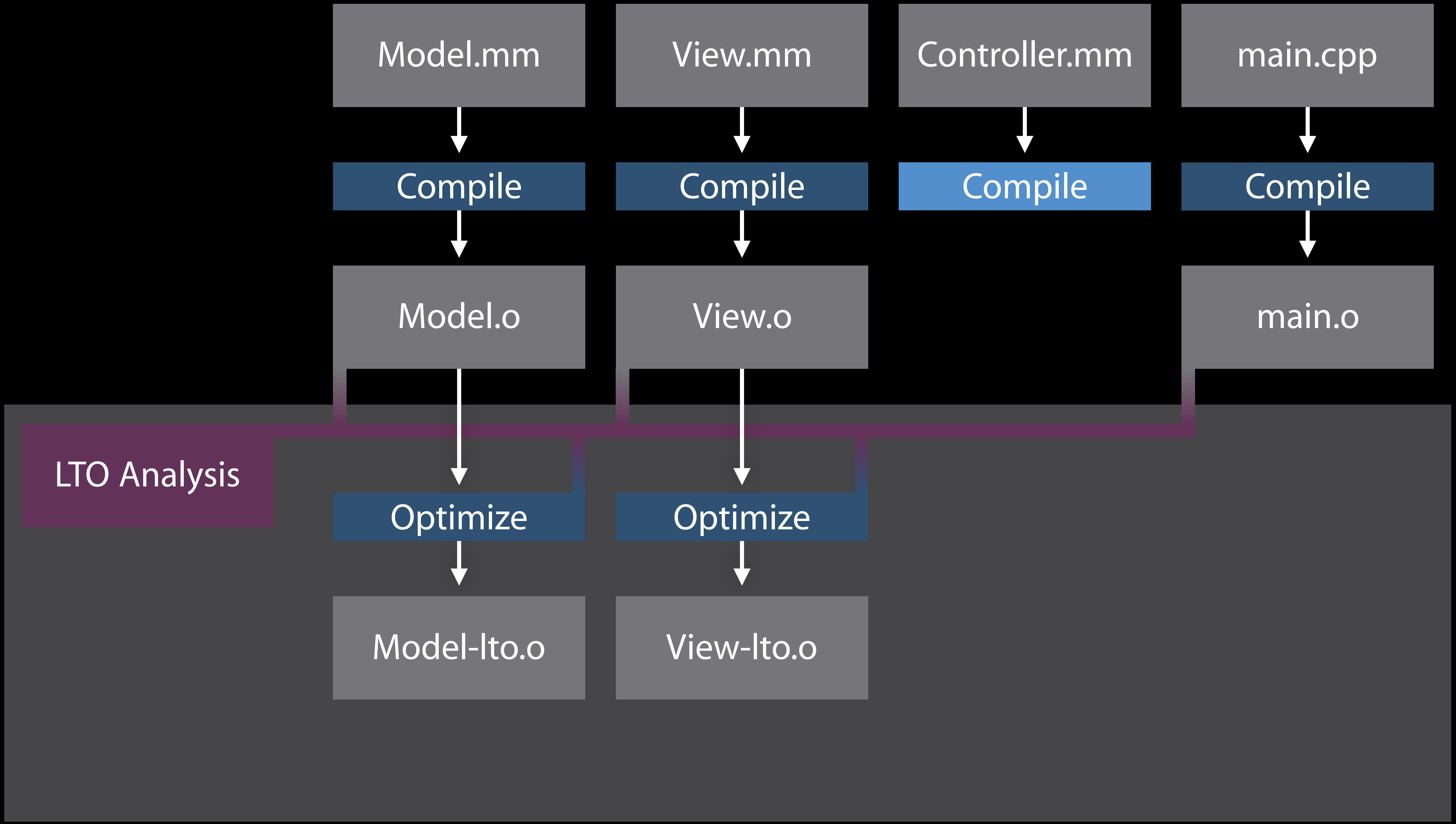
# Example of Incremental Build



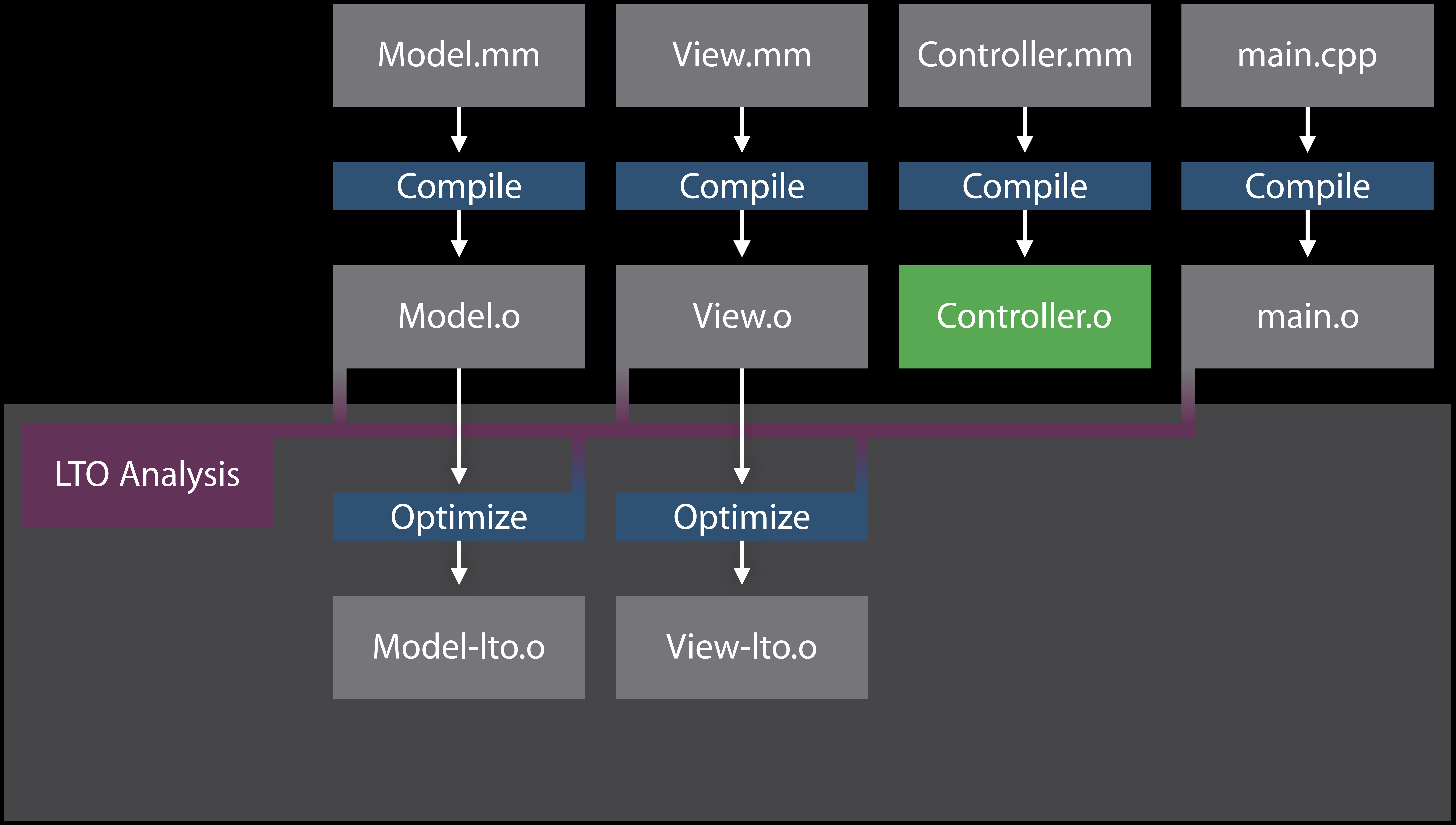
# Example of Incremental Build



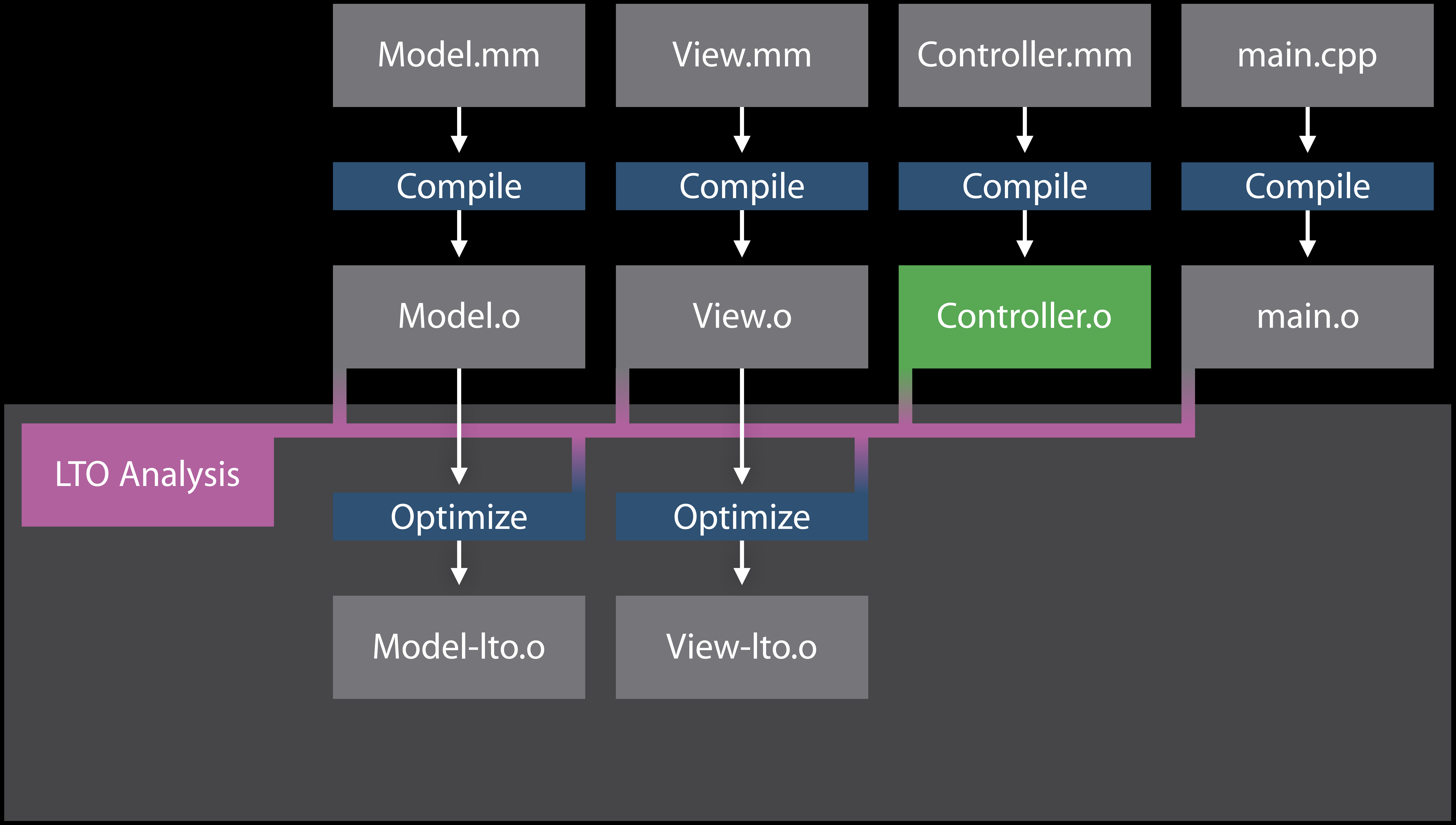
## Example of Incremental Build



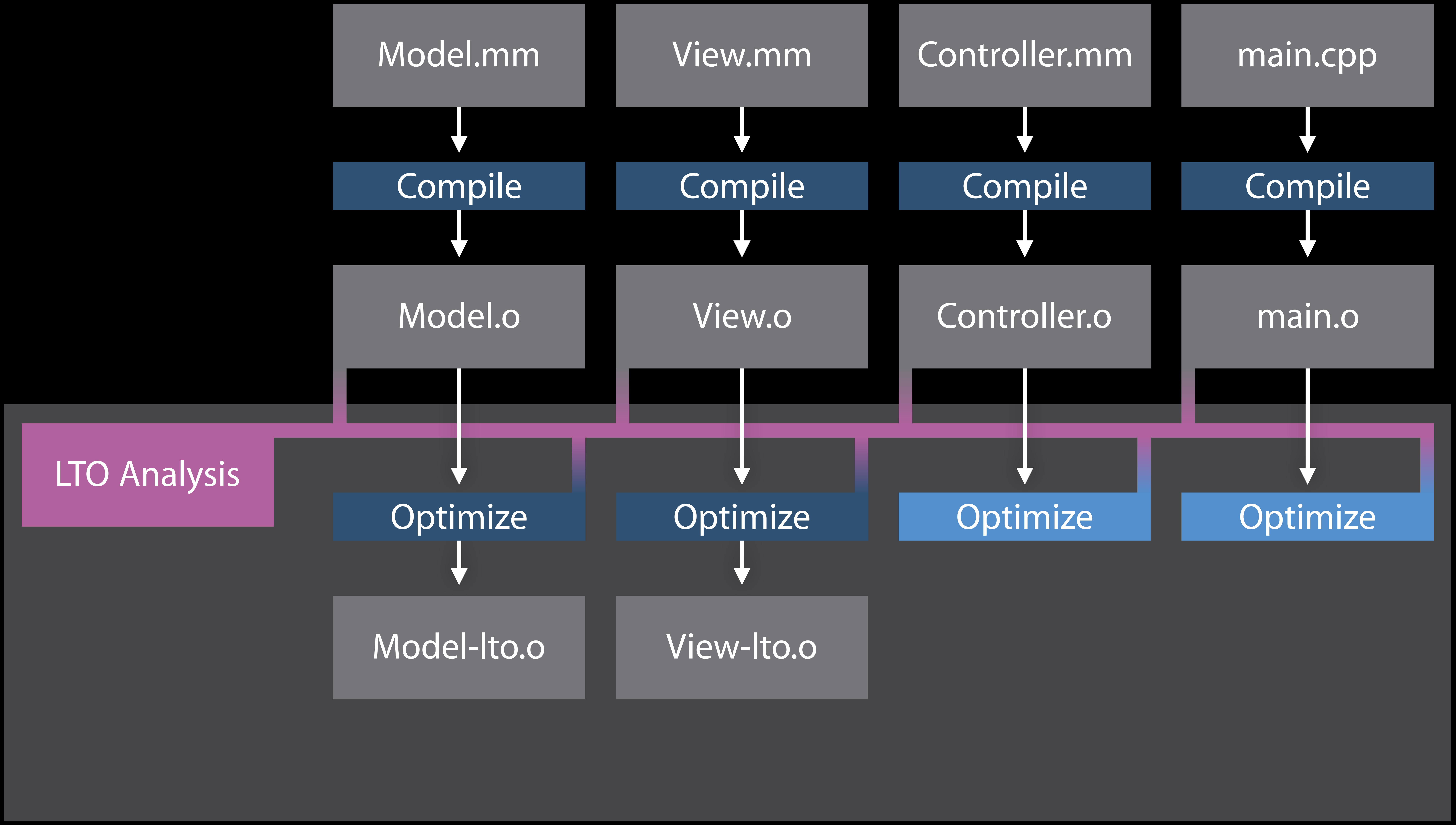
# Example of Incremental Build



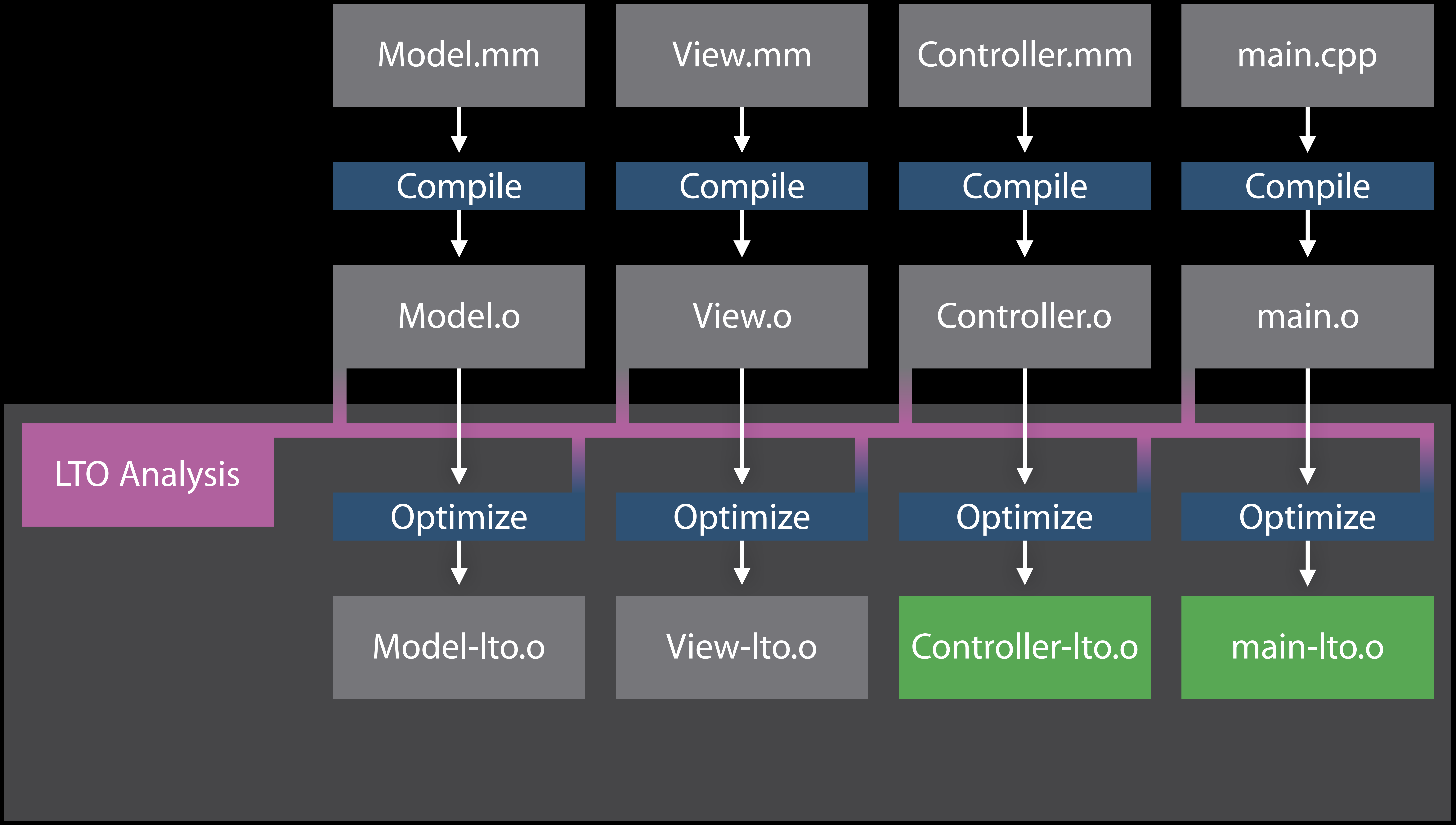
# Example of Incremental Build



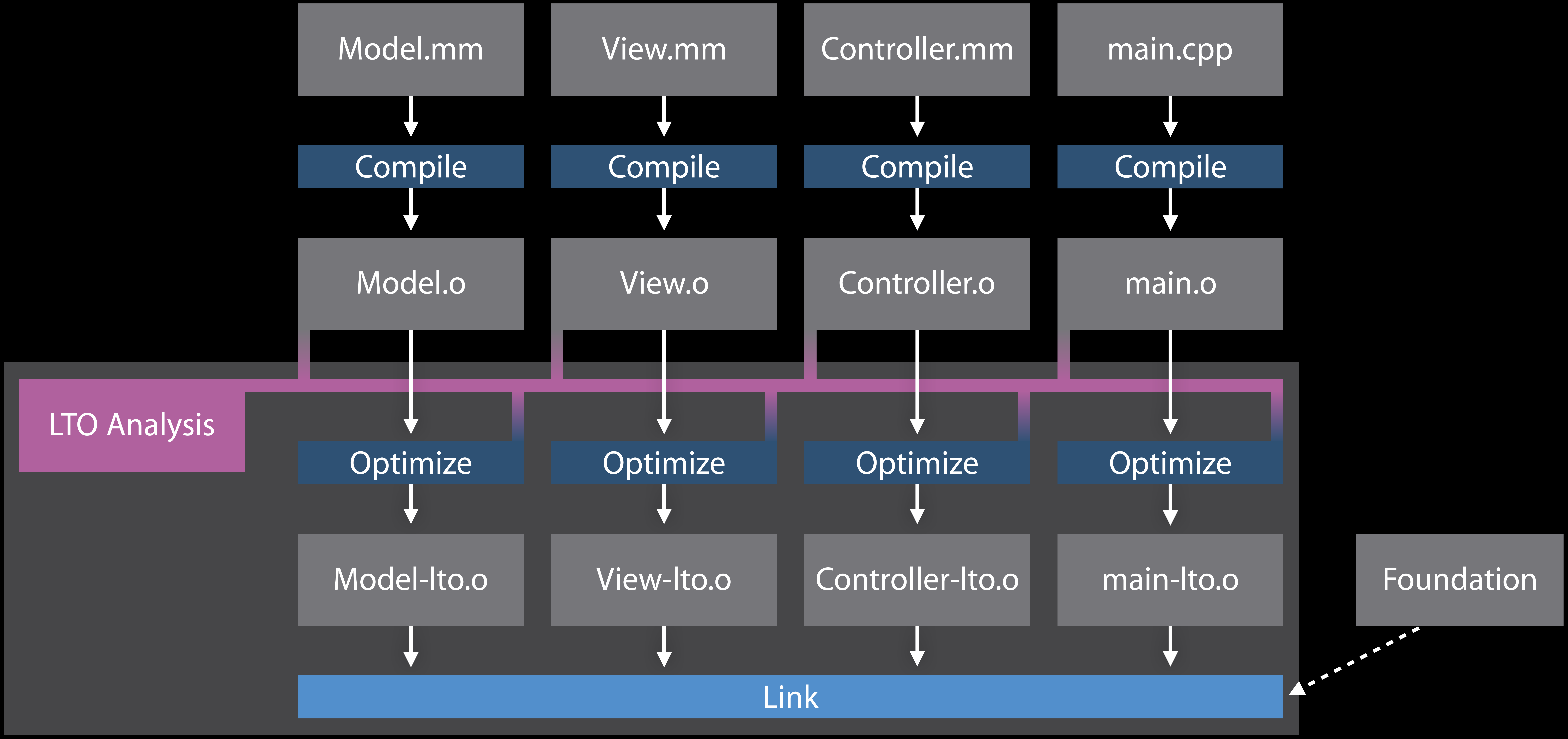
# Example of Incremental Build



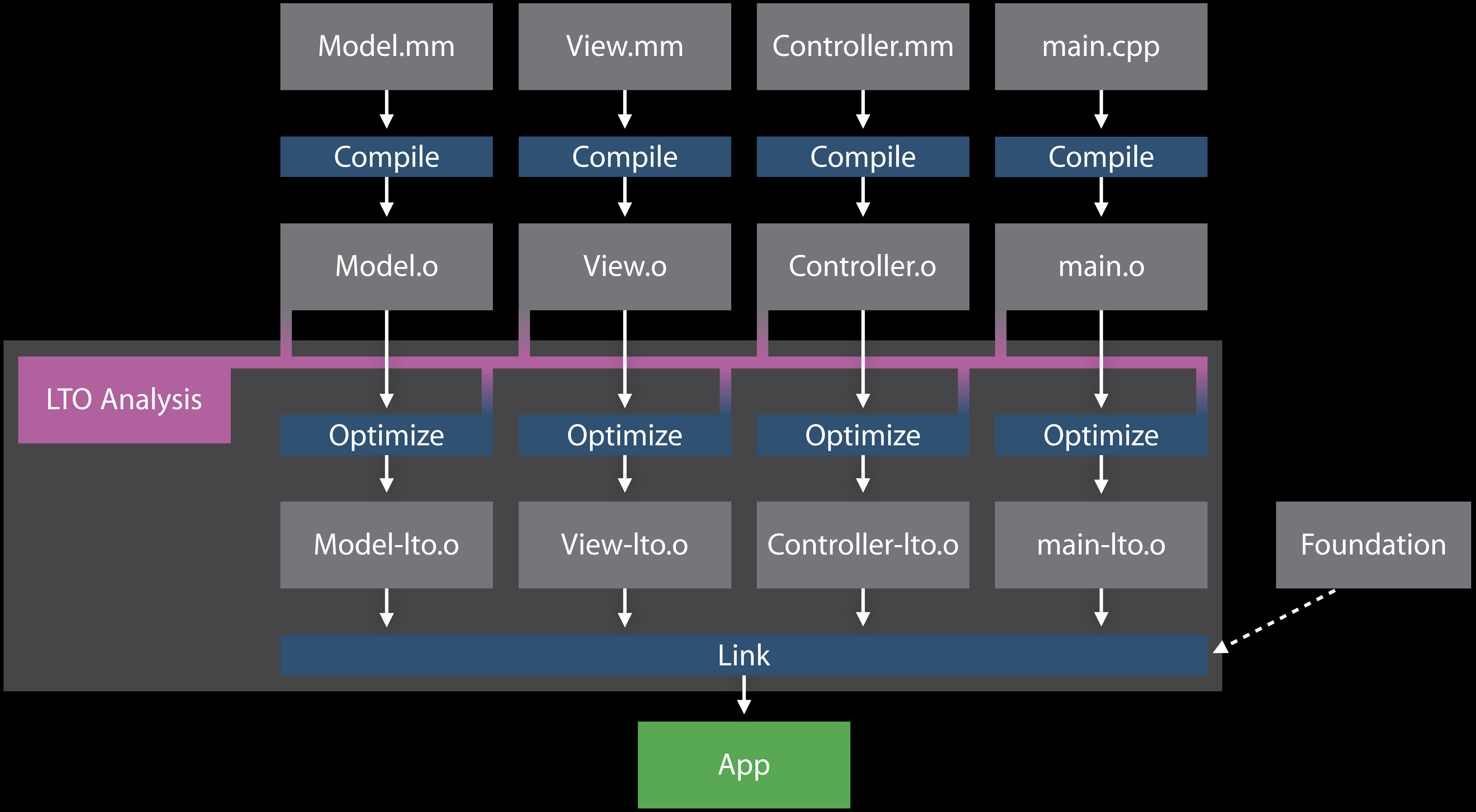
# Example of Incremental Build



# Example of Incremental Build

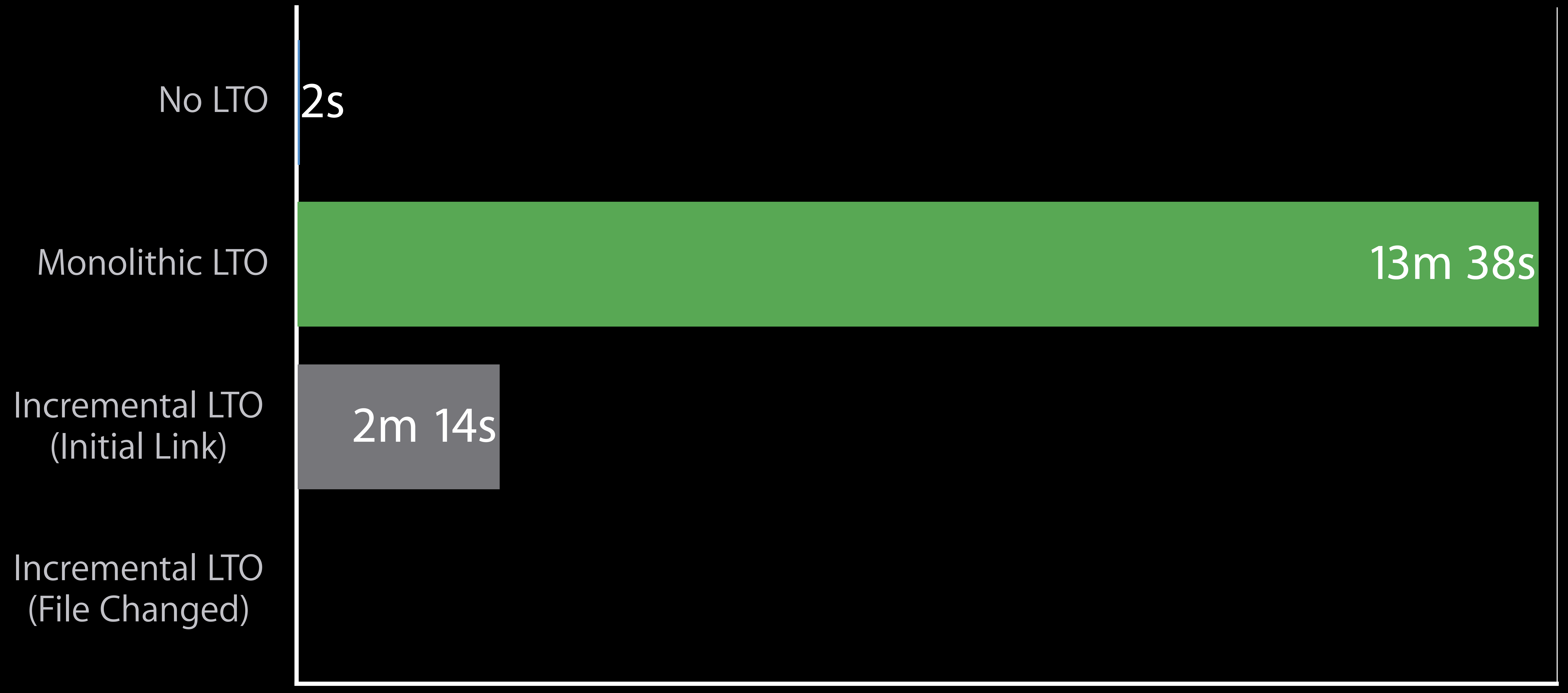


# Example of Incremental Build



# Incremental Link of Large C++ Project

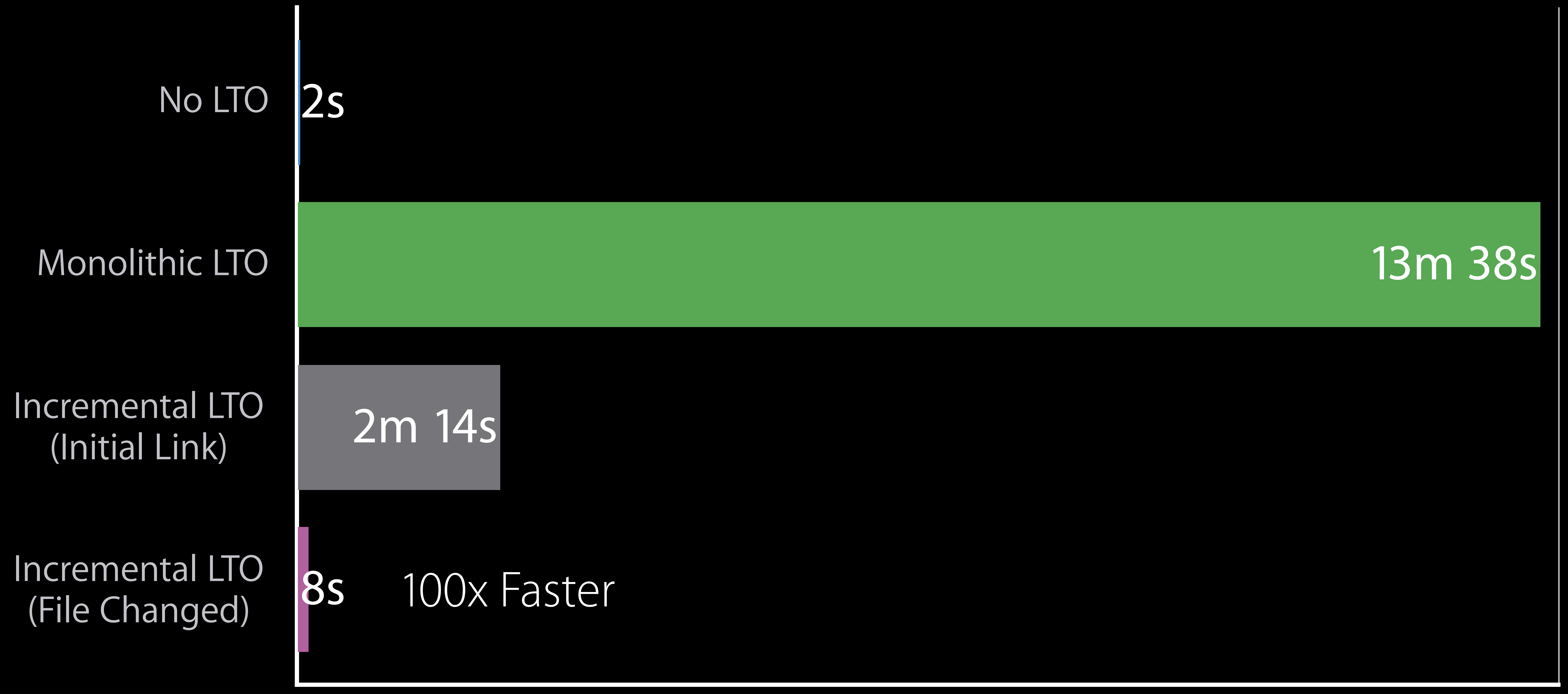
## Smaller is better



Time for link of Apple LLVM Compiler

# Incremental Link of Large C++ Project

Smaller is better



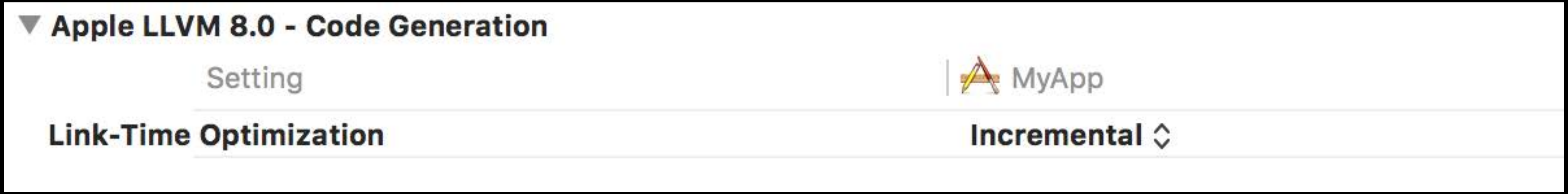
Time for link of Apple LLVM Compiler

## Enable Incremental LTO

Runtime performance similar to Monolithic LTO

Memory usage 10x smaller than Monolithic LTO

Incremental link almost as fast as No LTO



▼ Apple LLVM 8.0 - Code Generation

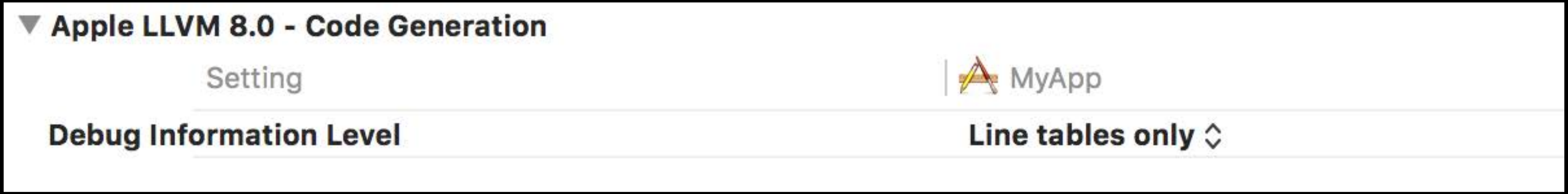
| Setting | MyApp |
|---|---|
| Link-Time Optimization | Incremental ◇ |

# LTO and Debug Info

## Recommendation

Use `-gline-tables-only` with large C++ projects

- Shorter compile time
- Smaller memory footprint
- Same rich backtraces at runtime



▼ Apple LLVM 8.0 - Code Generation

| Setting | MyApp |
|---|---|
| Debug Information Level | Line tables only ⌃⌄ |

## Compiler Optimizations

Link-Time Optimization

Code Generation

arm64 Cache Tuning

## Code Generation

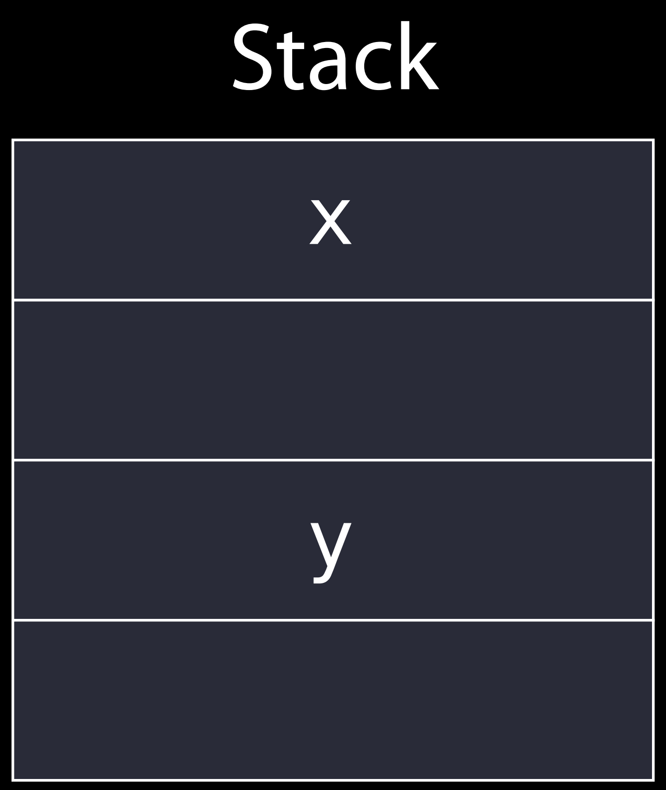
Gerolf Hoflehner Manager, LLVM Backend

## Stack Packing

```c
int *ptr = NULL;
if (cond) {
  int x = 71;
  // ...
}
int y = 79;
// ...
```

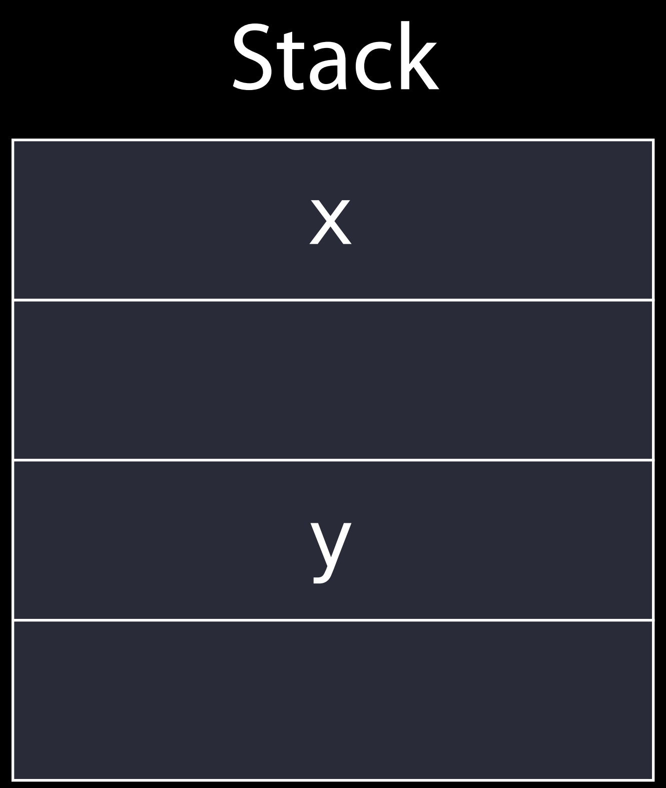
# Stack Packing

```
int *ptr = NULL;
if (cond) {
  int x = 71;
  // ...
}
int y = 79;
// ...
```



# Stack Packing

```
int *ptr = NULL;
if (cond) {
  int x = 71;
  // ...
}
int y = 79;
// ...
```

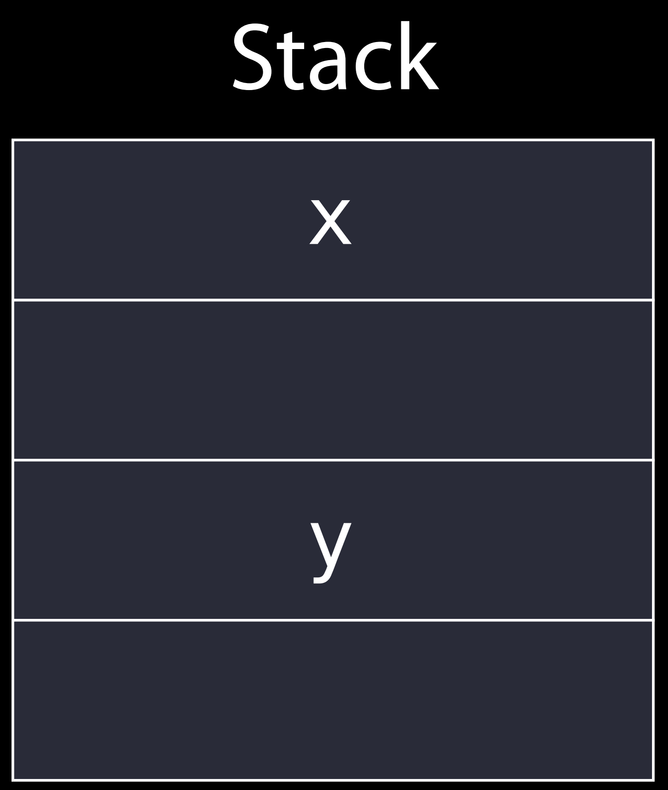


Local variables live until the end of scope

- Stack slots can be reused when variable lifetime ends

# Stack Packing

```
int *ptr = NULL;
if (cond) {
  int x = 71;
  // ...
}
int y = 79;
// ...
```

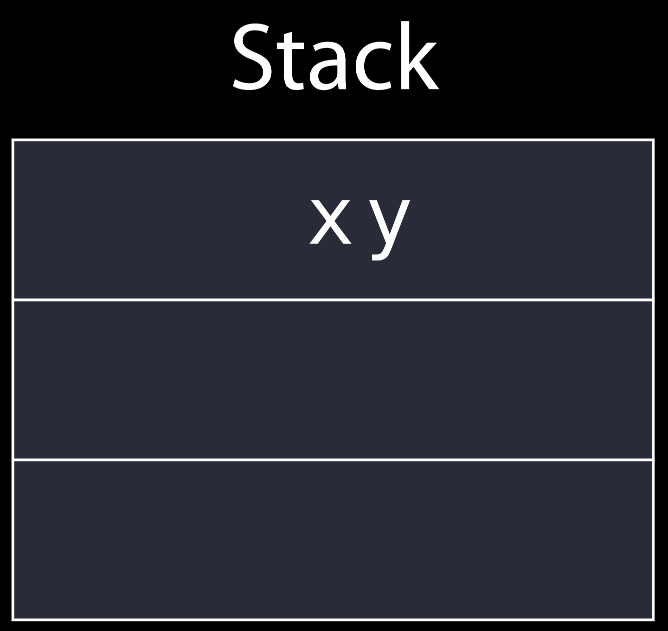


Local variables live until the end of scope

- Stack slots can be reused when variable lifetime ends

# Stack Packing

```
int *ptr = NULL;
if (cond) {
  int x = 71;
  // ...
}
int y = 79;
// ...
```



Local variables live until the end of scope

- Stack slots can be reused when variable lifetime ends
- Another local variable may have the same address
- Reduces stack usage in Release builds

# Escaping Local Addresses

```
int *ptr = NULL;
if (cond) {
  int x = 71;
  ptr = &x;
}
int y = 79;
if (ptr) printf("ptr = %d\n", *ptr);
```

# Escaping Local Addresses

```
int *ptr = NULL;
if (cond) {
  int x = 71;
  ptr = &x;
}
int y = 79;
if (ptr) printf("ptr = %d\n", *ptr);
```

Undefined Behavior

Use of local variable after its lifetime ends

Using an out-of-scope address is undefined behavior

- Different results in Debug and Release builds
- May crash

# Escaping Local Addresses

```c
int *ptr = NULL;
int x = 71;
if (cond) {
  ptr = &x;
}
int y = 79;
if (ptr) printf("ptr = %d\n", *ptr);
```

Using an out-of-scope address is undefined behavior

- Different results in Debug and Release builds
- May crash
- Expand local variable lifetimes

## Shrink-Wrapping

NEW

Reduce code at function boundaries

Code at function entries and exit might not be needed on all paths

- Stack operations
- Register saves/restores

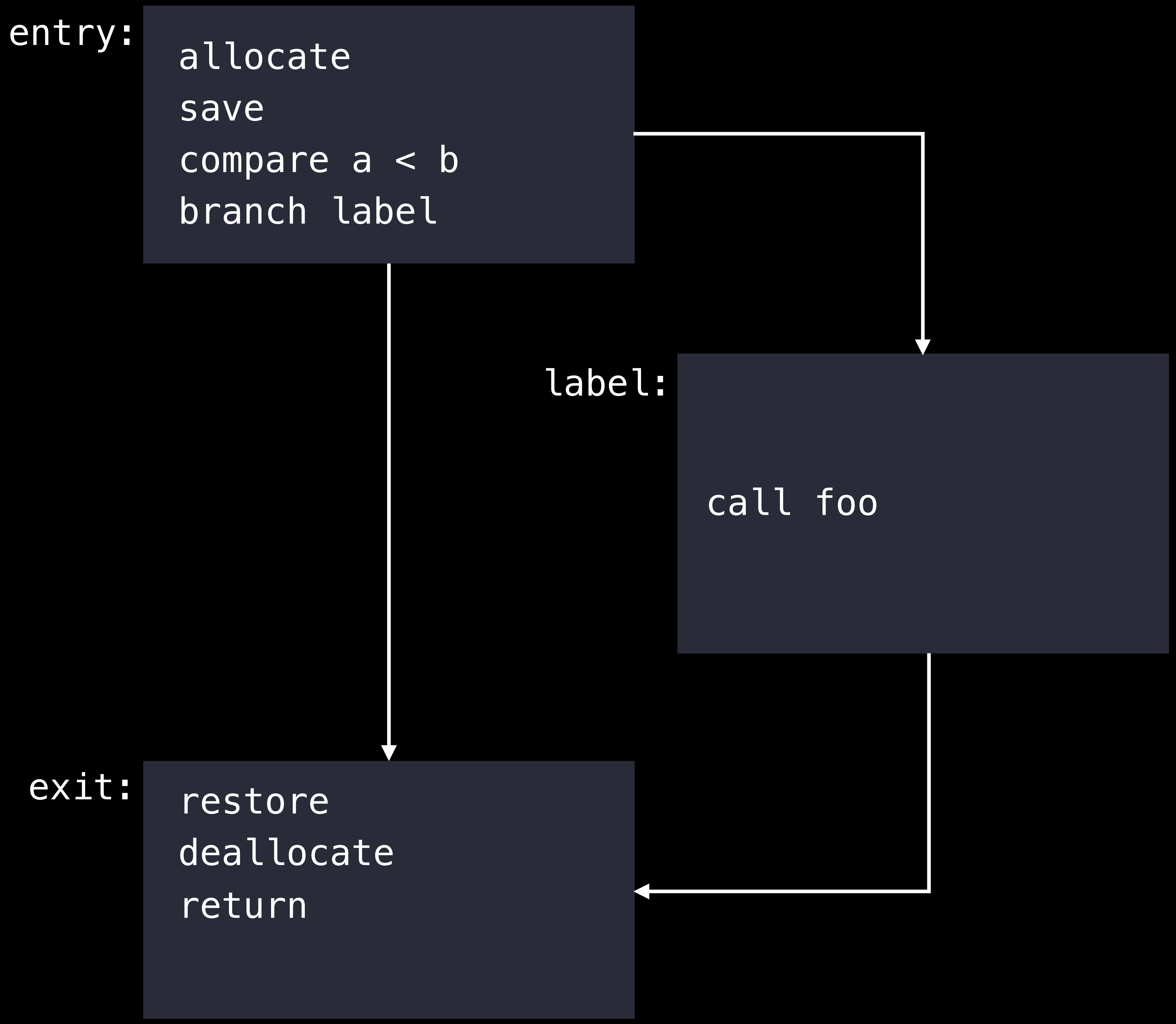
## Shrink-Wrapping Illustration

## Shrink-Wrapping Illustration

```
int doSomething(int a, int b) {
  int r = 0;
  if (a < b) {
    int v1;
    r = foo(&v1);
    …
  }
  return r;
}
```

# Shrink-Wrapping Illustration

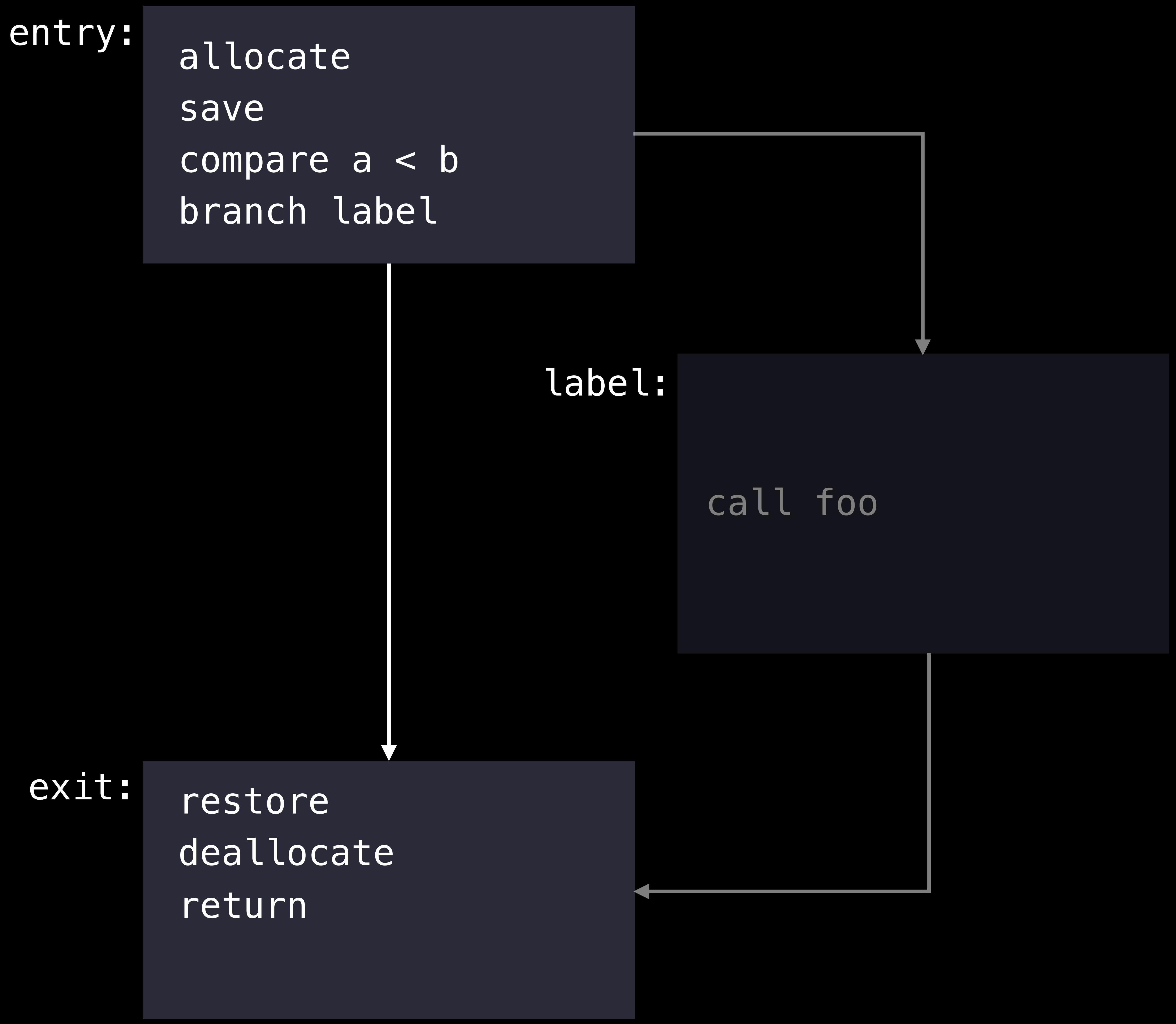
```c
int doSomething(int a, int b) {
  int r = 0;
  if (a < b) {
    int v1;
    r = foo(&v1);
    …
  }
  return r;
}
```



```
entry:
  allocate
  save
  compare a < b
  branch label

label:
  call foo

exit:
  restore
  deallocate
  return
```

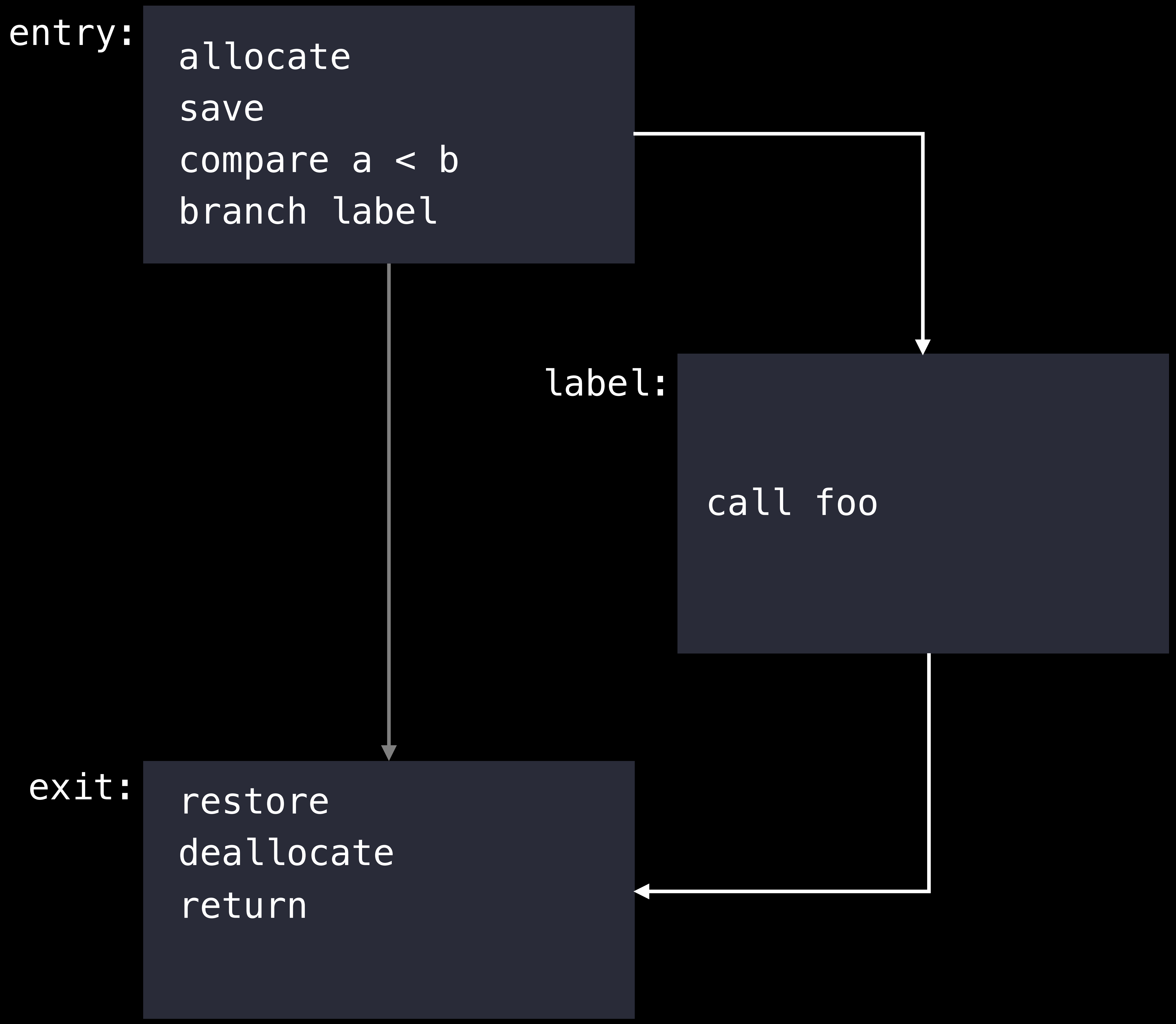
# Shrink-Wrapping Illustration

```c
int doSomething(int a, int b) {
  int r = 0;
  if (a < b) {
    int v1;
    r = foo(&v1);
    …
  }
  return r;
}
```



# Shrink-Wrapping Illustration

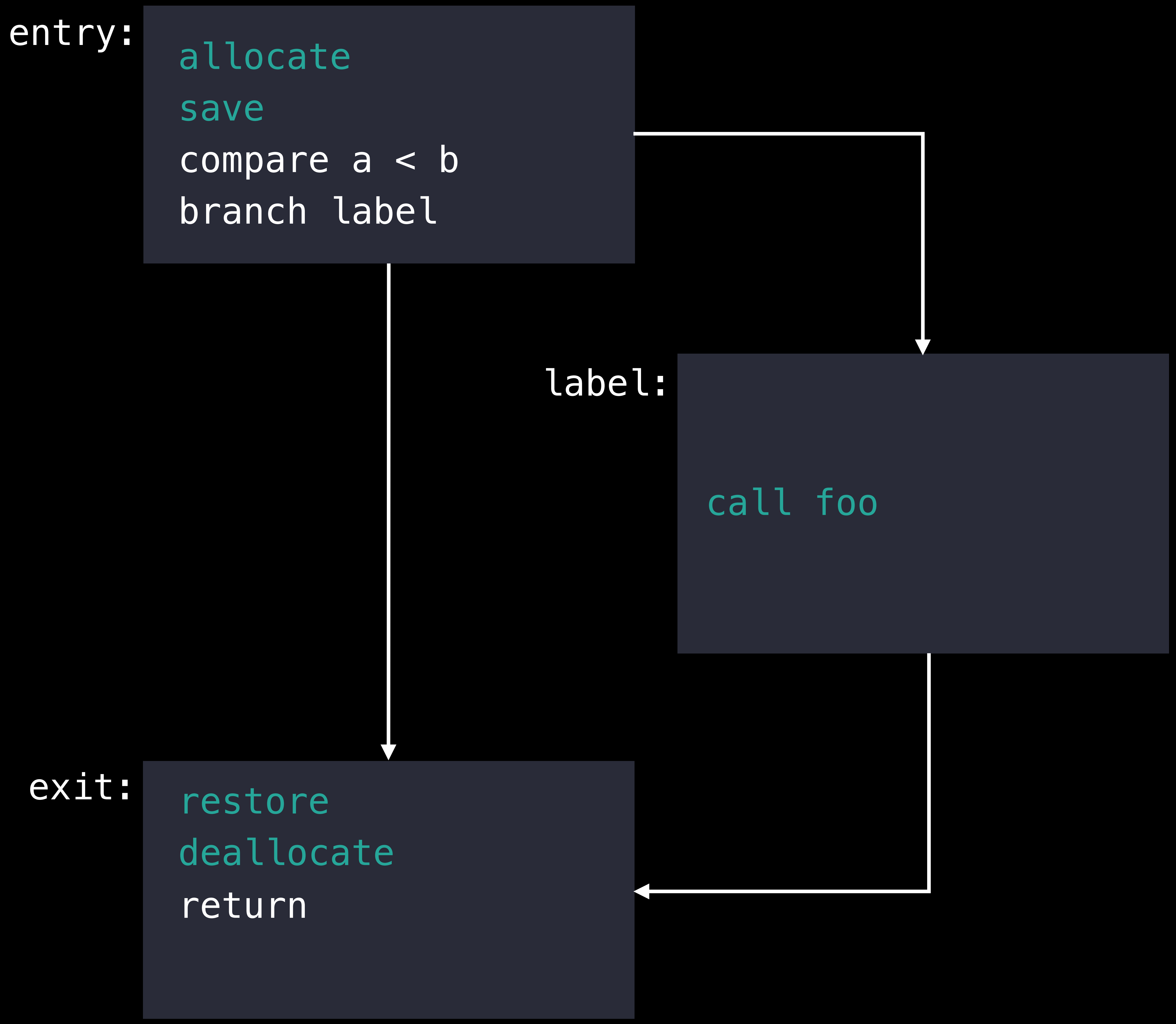
```
int doSomething(int a, int b) {
  int r = 0;
  if (a < b) {
    int v1;
    r = foo(&v1);
    …
  }
  return r;
}
```



```
entry:
allocate
save
compare a < b
branch label

label:
call foo

exit:
restore
deallocate
return
```

# Shrink-Wrapping Illustration

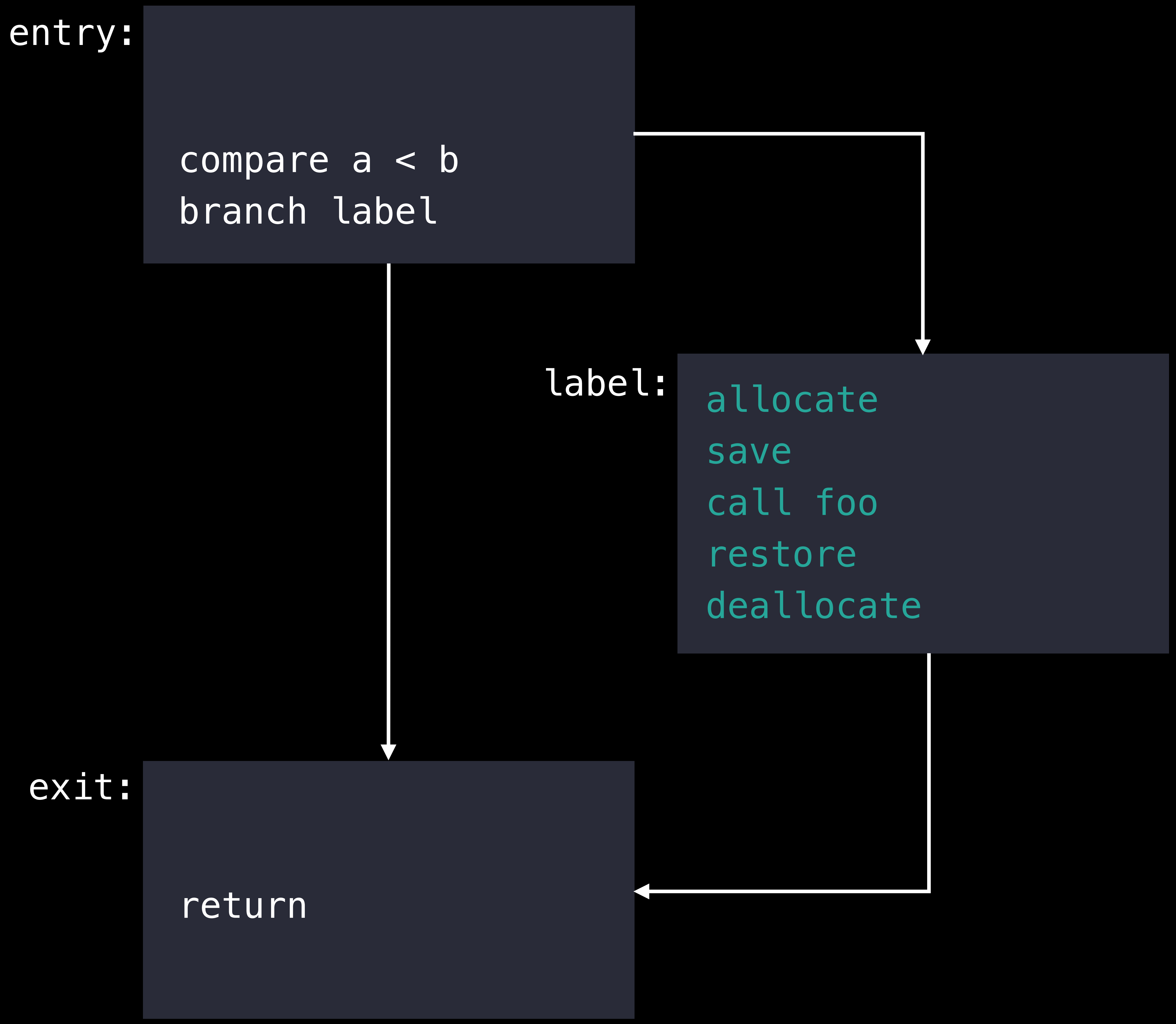
```
int doSomething(int a, int b) {
  int r = 0;
  if (a < b) {
    int v1;
    r = foo(&v1);
    …
  }
  return r;
}
```



```
entry:
allocate
save
compare a < b
branch label

label:
call foo

exit:
restore
deallocate
return
```

# Shrink-Wrapping Illustration

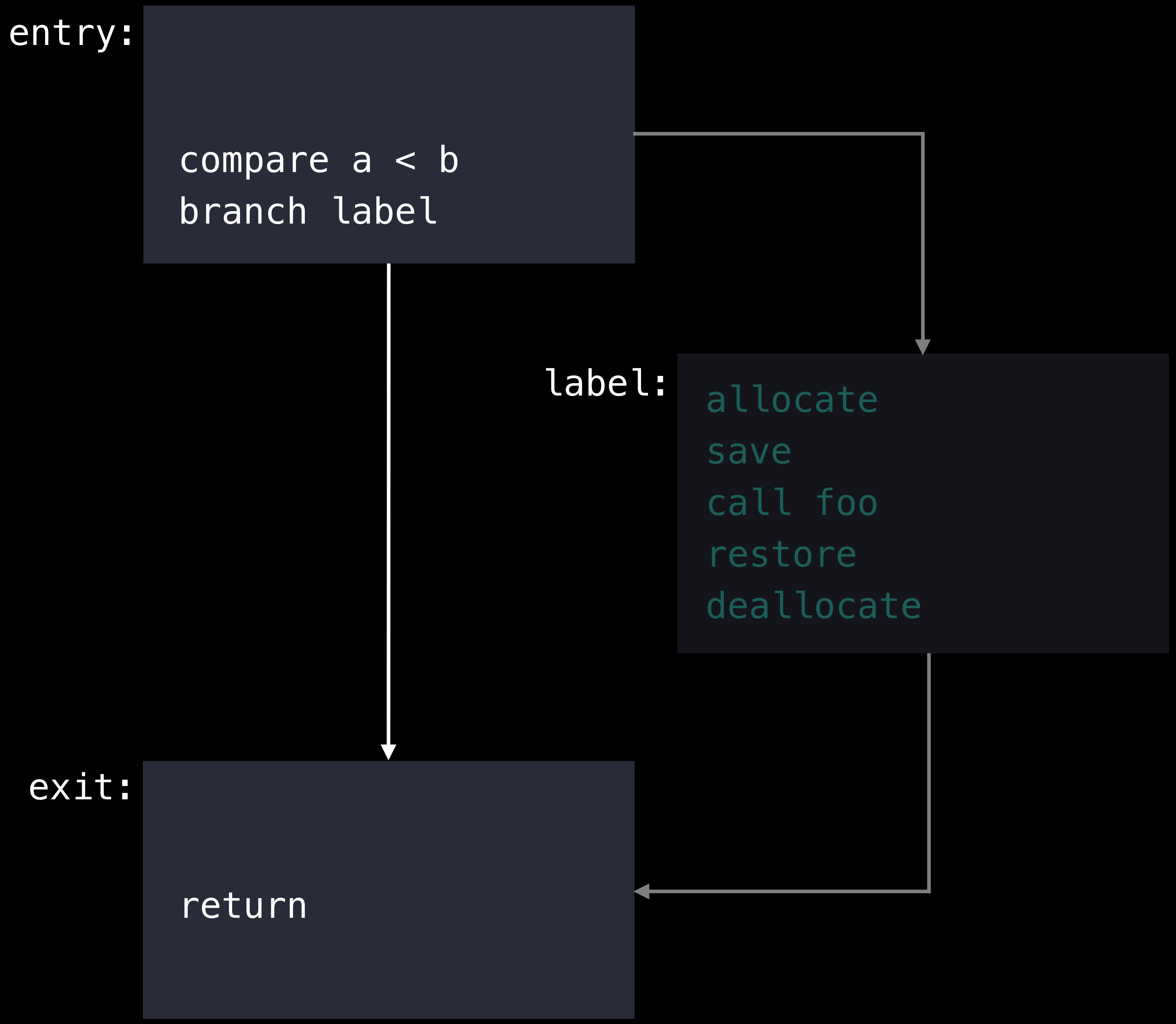
```
int doSomething(int a, int b) {
  int r = 0;
  if (a < b) {
    int v1;
    r = foo(&v1);
    …
  }
  return r;
}
```



```
entry:
  compare a < b
  branch label

label:
  allocate
  save
  call foo
  restore
  deallocate

exit:
  return
```

# Shrink-Wrapping Illustration

```
int doSomething(int a, int b) {
  int r = 0;
  if (a < b) {
    int v1;
    r = foo(&v1);
    …
  }
  return r;
}
```



## Selective Fused Multiply-Add

arm64

NEW

Usually generating a single multiply-add instruction is the better choice

- Fused multiply-add (`madd`) computes a+b*c
- Single instruction rather than two instructions: `mul` and `add`
- Generating `mul` and `add` may increase instruction-level parallelism

## Selective Fused Multiply-Add

NEW

arm64

Usually generating a single multiply-add instruction is the better choice

- Fused multiply-add (`madd`) computes a+b*c
- Single instruction rather than two instructions: `mul` and `add`
- Generating `mul` and `add` may increase instruction-level parallelism

```
int compute(int a, int b, int c, int d) {
  return a * b + c * d;
}
```

# Example with Fused Multiply-Add

```
a * b + c * d:
```

# Example with Fused Multiply-Add

```
a * b + c * d:

mul  w8, w1, w0       // t = a*b
```

# Example with Fused Multiply-Add

```
a * b + c * d:

mul  w8, w1, w0        // t = a*b
```

## Example with Fused Multiply-Add

```
a * b + c * d:

mul  w8, w1, w0         // t = a*b
madd w0, w3, w2, w8  // r = c*d + t
```

# Example with Fused Multiply-Add

```
a * b + c * d:

mul  w8, w1, w0       // t = a*b
madd w0, w3, w2, w8   // r = c*d + t
```

## Example with Fused Multiply-Add

```
     a * b + c * d:

4    mul  w8, w1, w0        // t = a*b
     madd w0, w3, w2, w8    // r = c*d + t
```

# Example with Fused Multiply-Add

```
    a * b + c * d:

4   mul  w8, w1, w0        // t = a*b
4   madd w0, w3, w2, w8   // r = c*d + t
```

# Example with Fused Multiply-Add

```
        a * b + c * d:

  4     mul  w8, w1, w0        // t = a*b
  4     madd w0, w3, w2, w8    // r = c*d + t
------
  8
```

## Faster Code with Two Multiplies

```
a * b + c * d:
```

## Faster Code with Two Multiplies

```
a * b + c * d:

mul  w8, w1, w0 //t1=a*b
```

## Faster Code with Two Multiplies

```
a * b + c * d:

mul  w8, w1, w0 //t1=a*b
mul  w9, w3, w2 //t2=c*d
```

## Faster Code with Two Multiplies

```
a * b + c * d:

mul  w8, w1, w0 //t1=a*b
mul  w9, w3, w2 //t2=c*d
add  w0, w9, w8 //t1+t2
```

# Faster Code with Two Multiplies

```
a * b + c * d:

mul  w8, w1, w0 //t1=a*b   mul  w9, w3, w2 //t2=c*d
add  w0, w9, w8 //t1+t2
```

# Faster Code with Two Multiplies

```
    a * b + c * d:

  4 mul  w8, w1, w0 //t1=a*b   mul  w9, w3, w2 //t2=c*d
    add  w0, w9, w8 //t1+t2
```

## Faster Code with Two Multiplies

```
  a * b + c * d:
4 mul  w8, w1, w0 //t1=a*b   mul  w9, w3, w2 //t2=c*d
1 add  w0, w9, w8 //t1+t2
```

## Faster Code with Two Multiplies

```
    a * b + c * d:

  4 mul  w8, w1, w0 //t1=a*b   mul  w9, w3, w2 //t2=c*d
  1 add  w0, w9, w8 //t1+t2
-----------
  5
```

## Faster Code with Two Multiplies

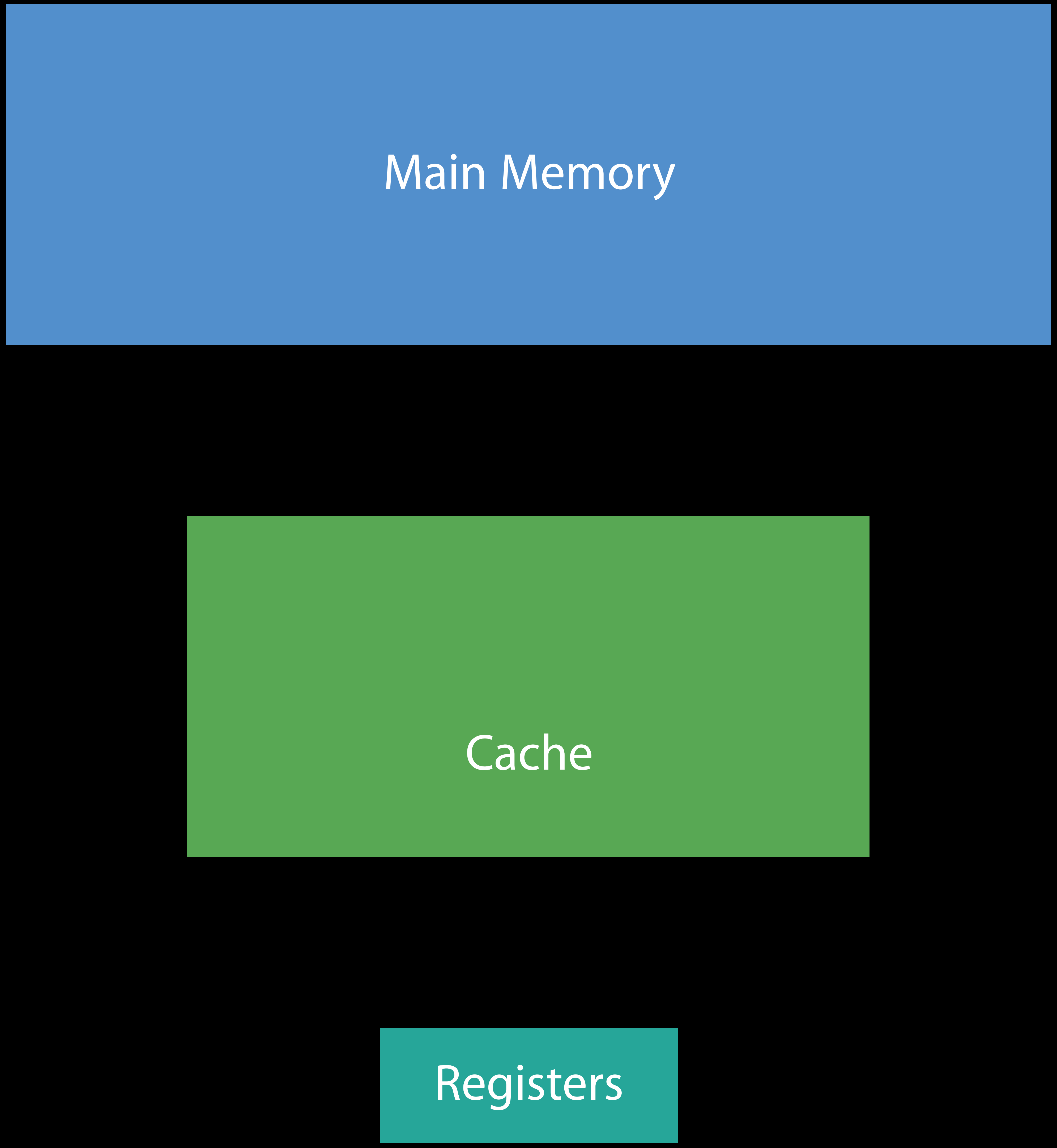
```
    a * b + c * d:

  4 mul  w8, w1, w0 //t1=a*b   mul  w9, w3, w2 //t2=c*d
  1 add  w0, w9, w8 //t1+t2
-----------
 5
```

5 vs 8

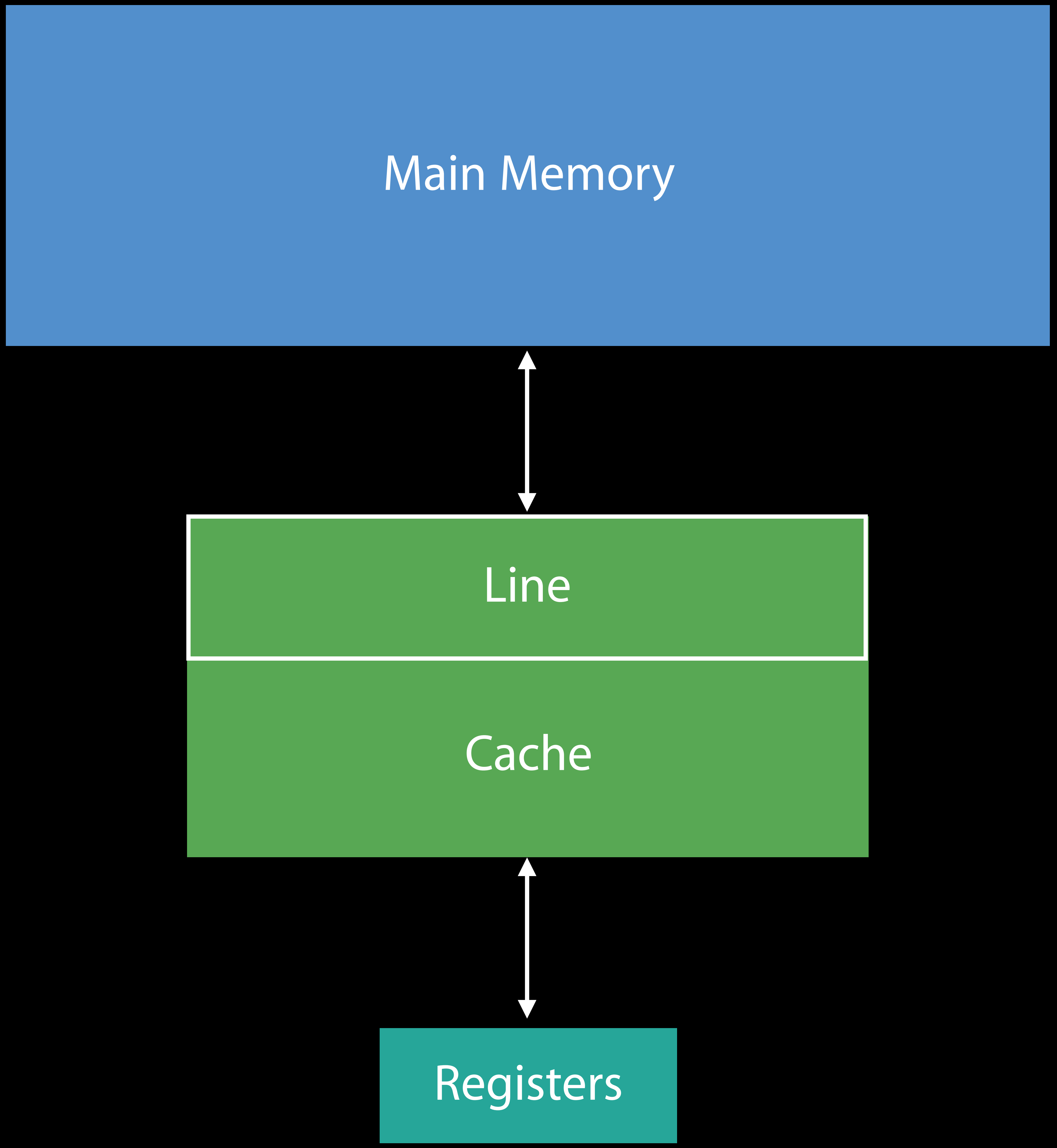
```
mul    w8, w1, w0
madd   w0, w3, w2, w8
```

## arm64 Cache Tuning

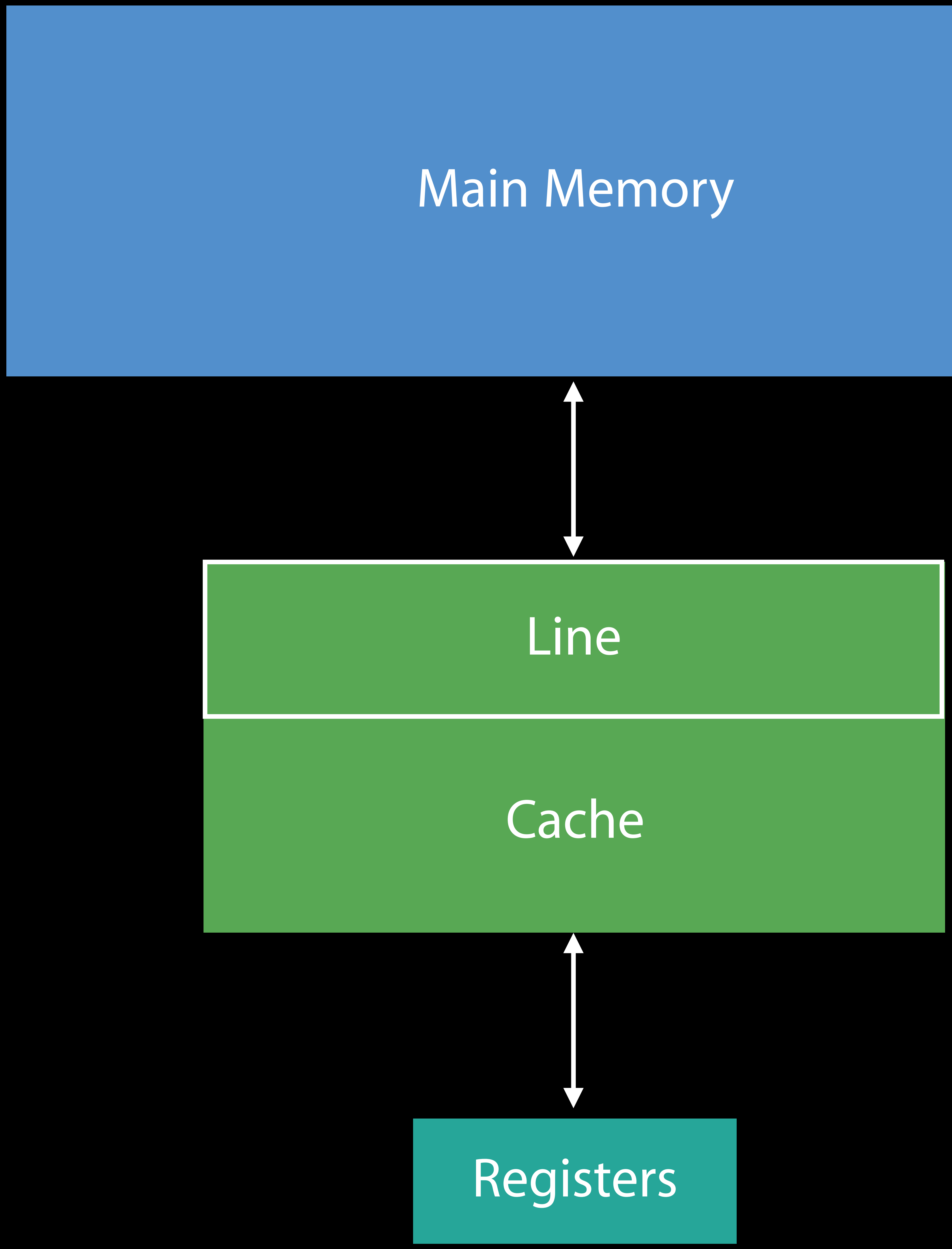
## Illustrated Memory Hierarchy



# Illustrated Memory Hierarchy



## Illustrated Memory Hierarchy



Temporal Locality:
Data will be reused soon

Spatial Locality:
Near-by data will be used soon

Future data accesses will be fast

# Software Prefetching

arm64

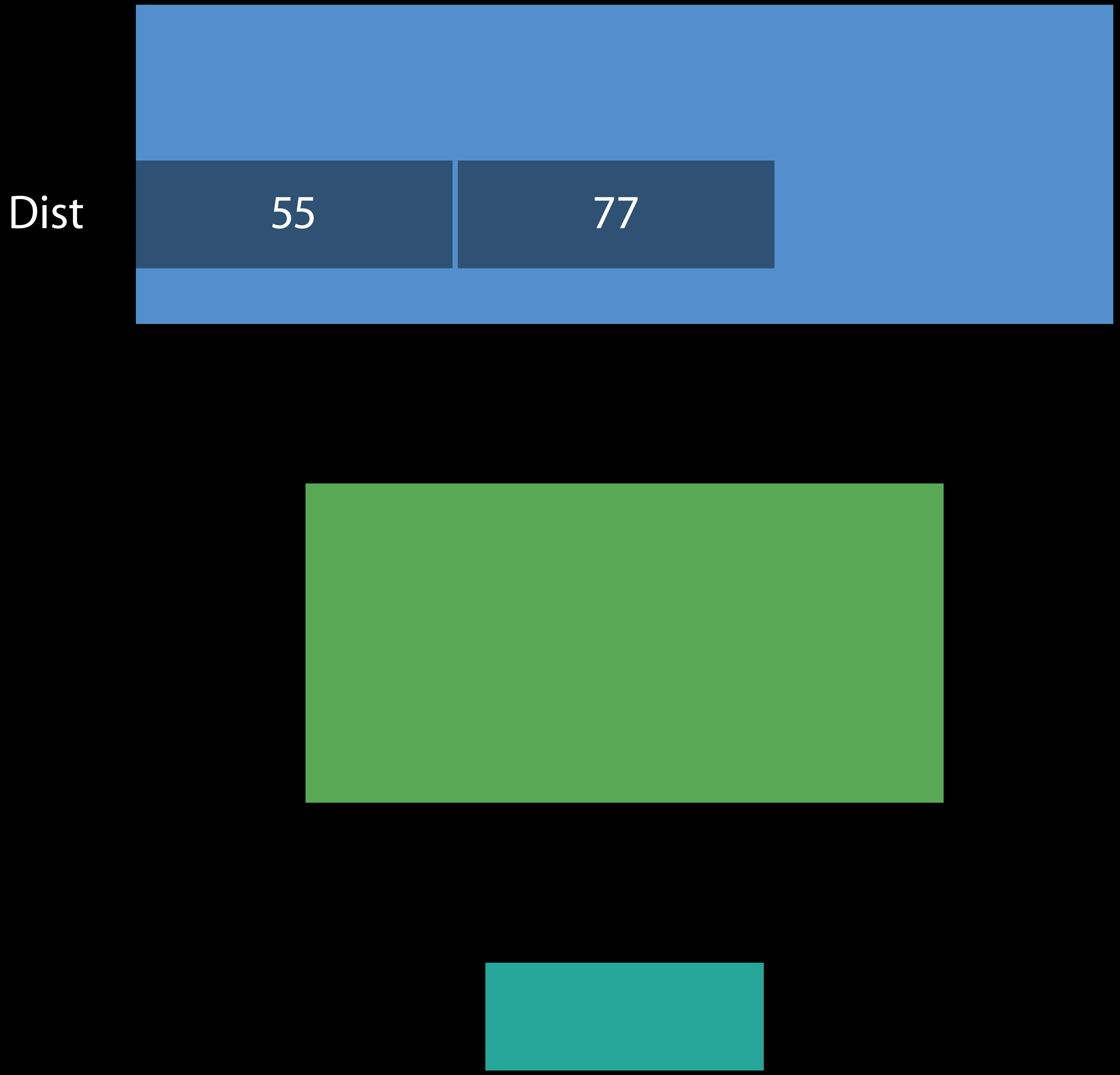
NEW

Moves data into cache ahead of time

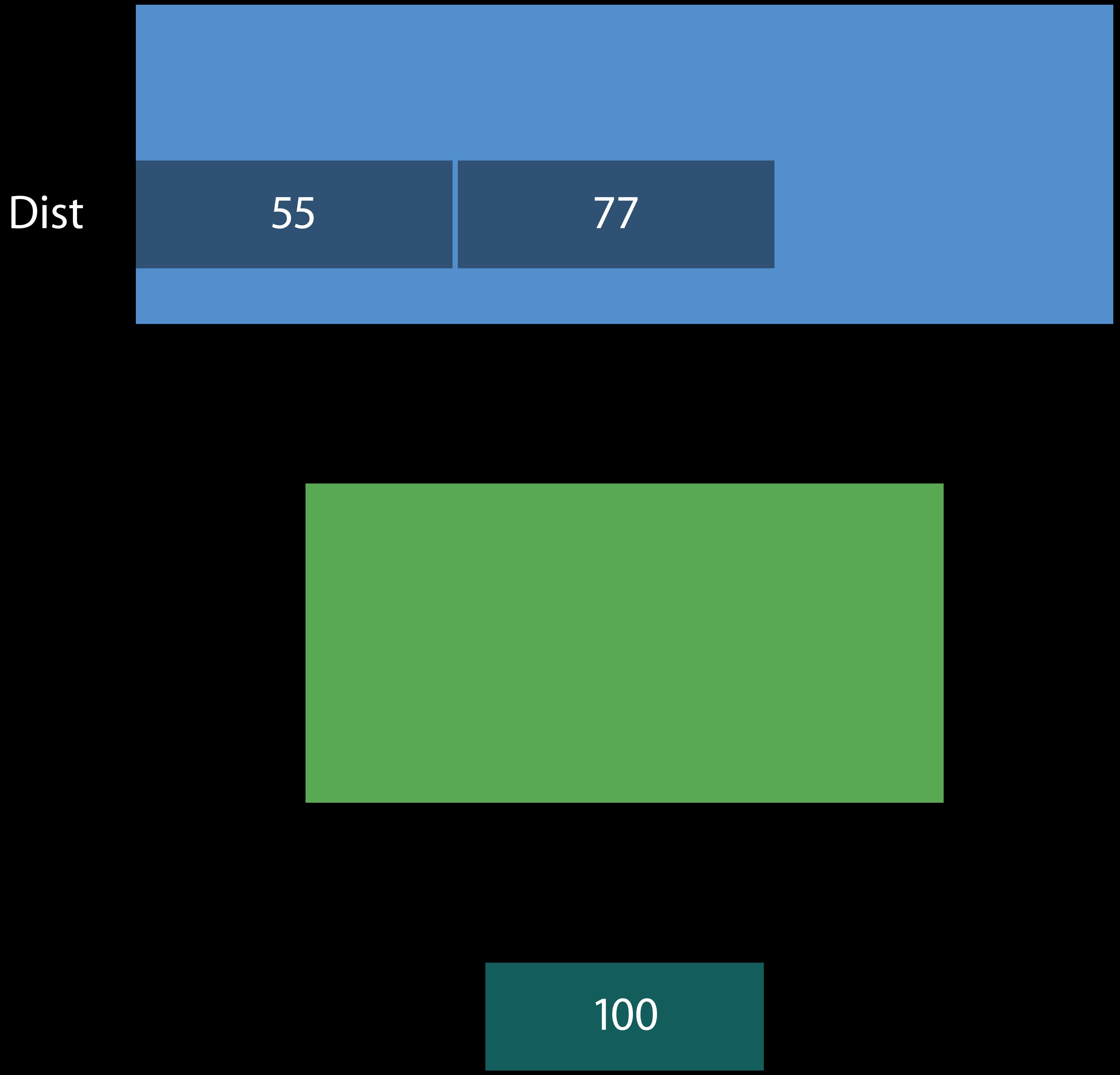
Compiler inserts prefetch instructions into loops

Complements hardware prefetching

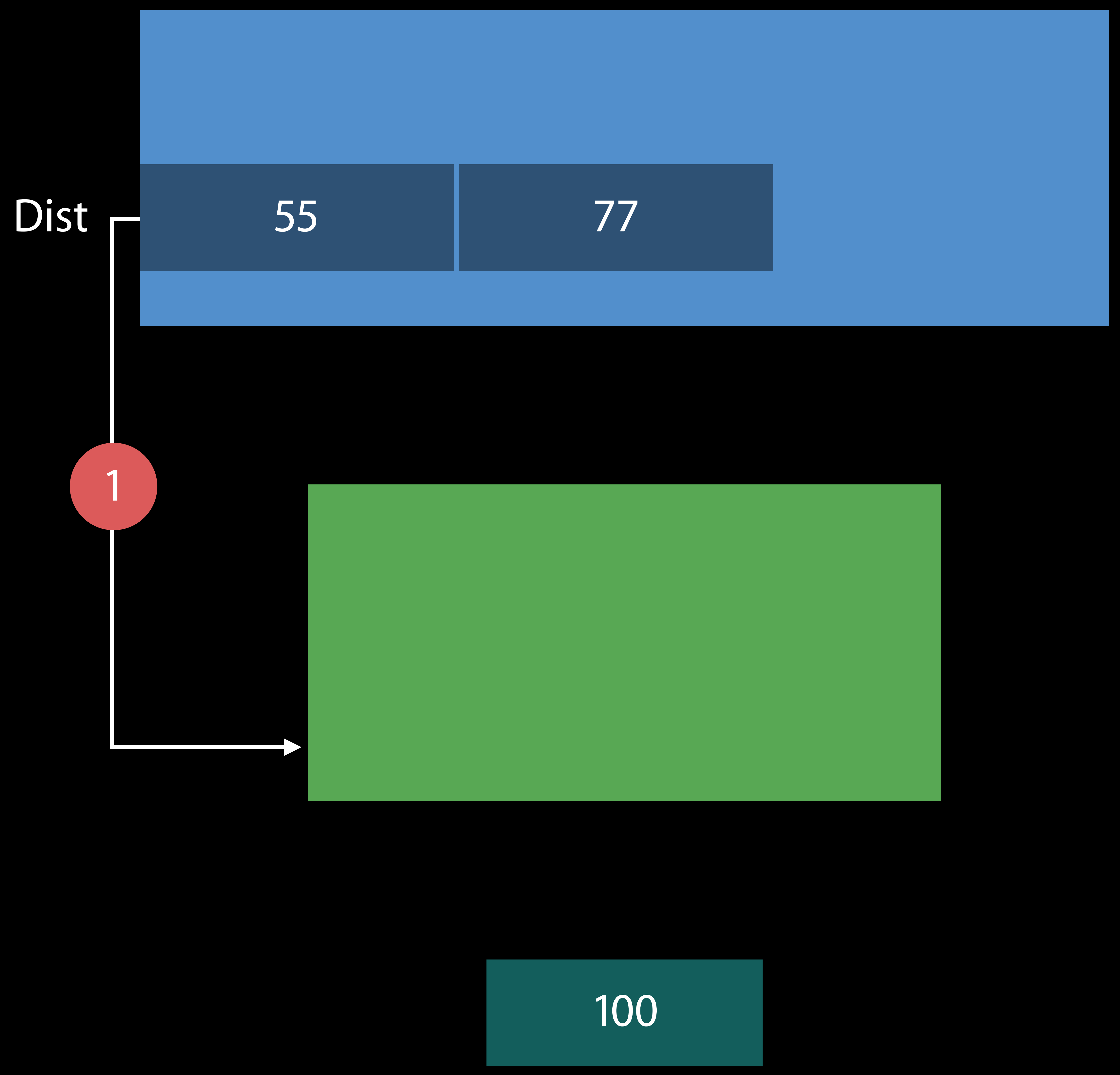
# Data Store 101



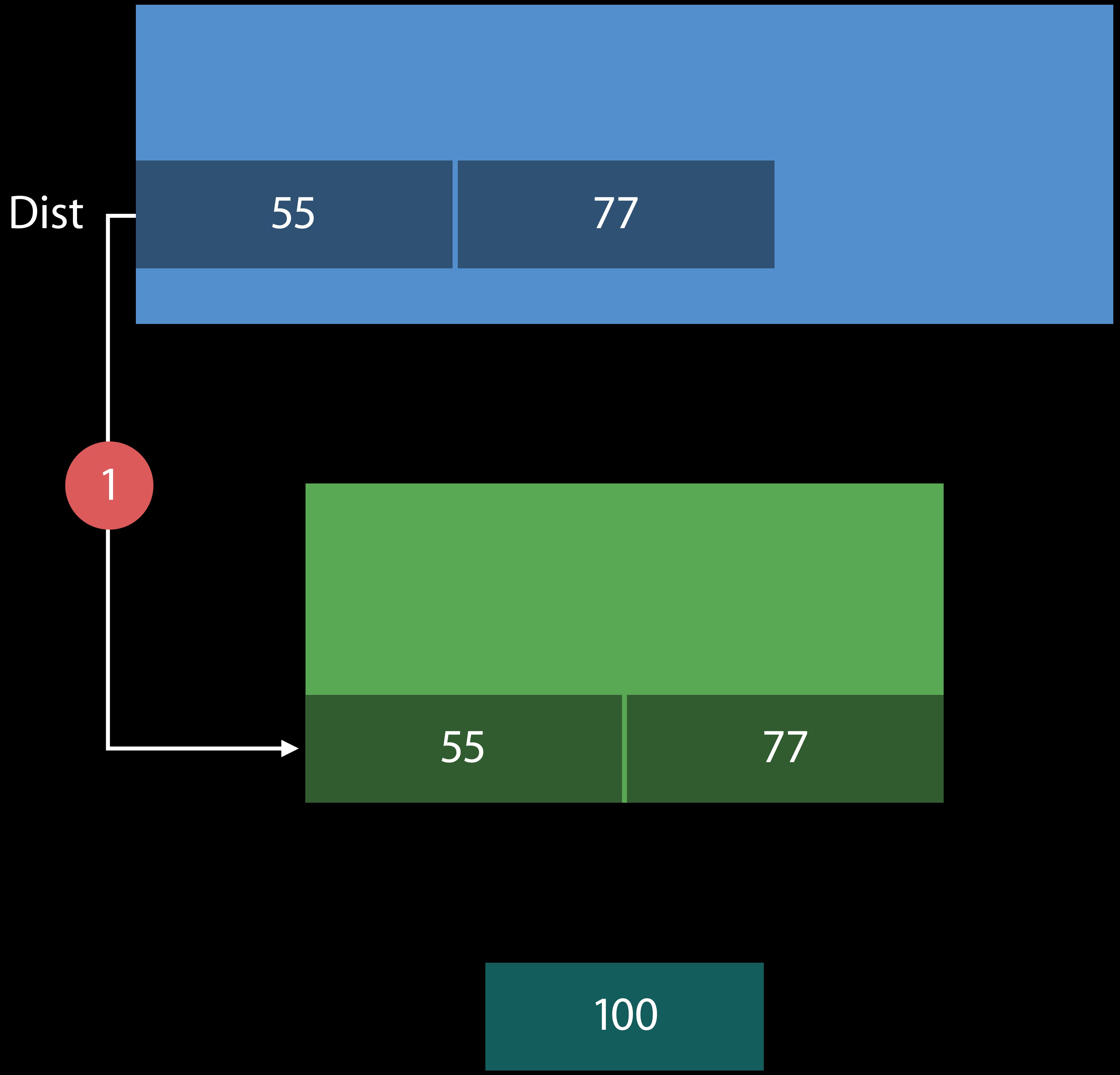
# Data Store 101



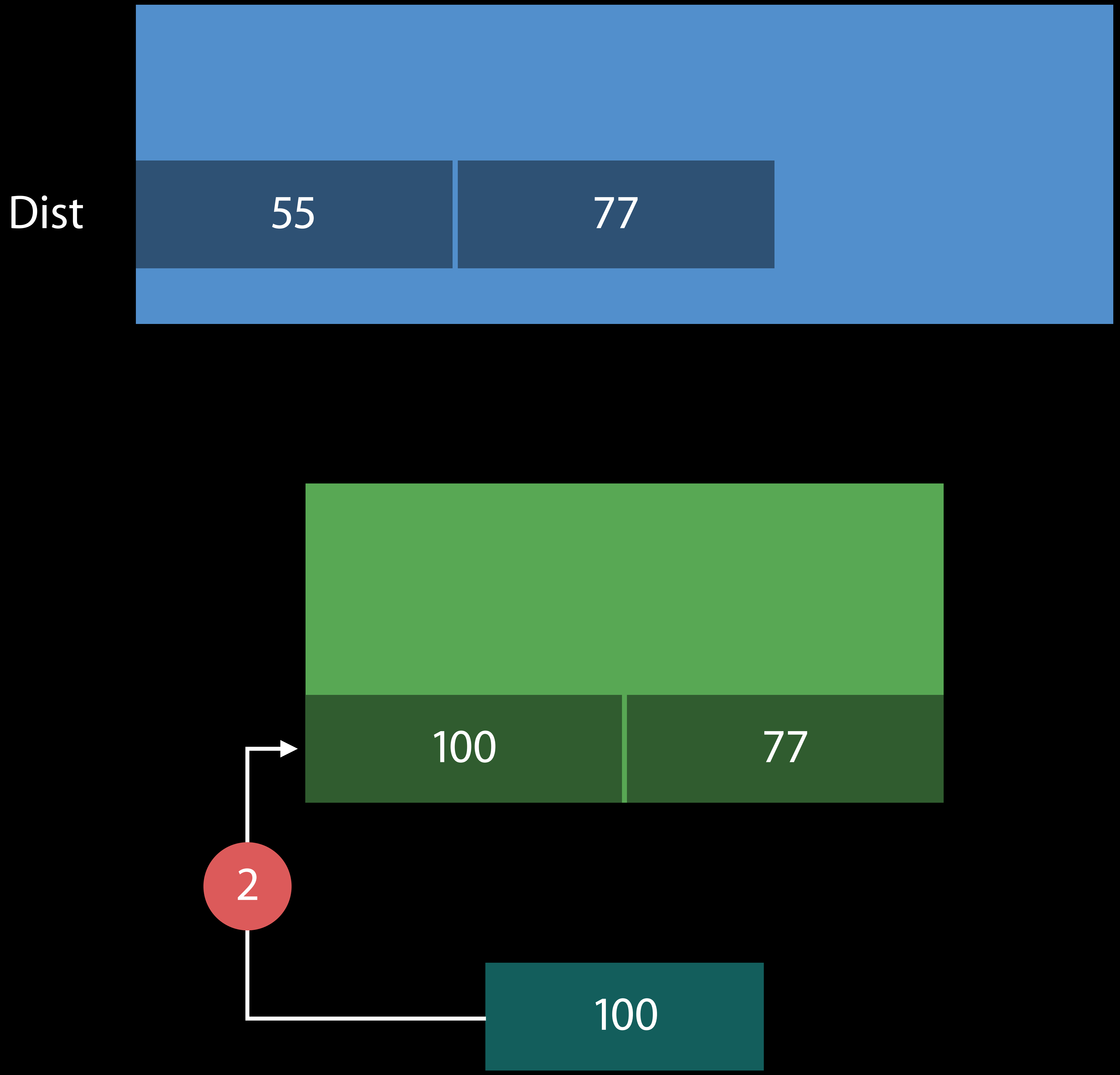
# Data Store 101



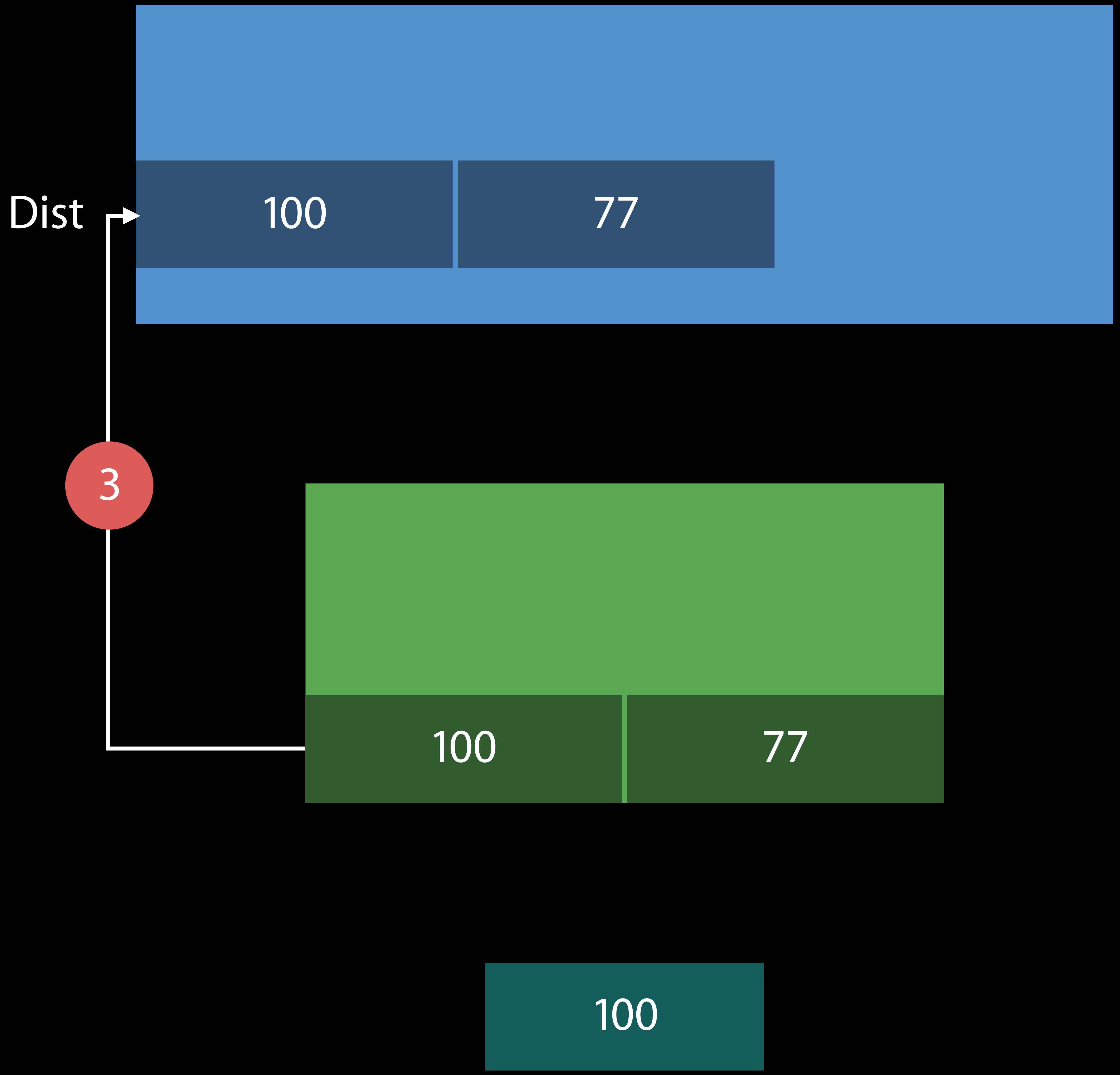
# Data Store 101



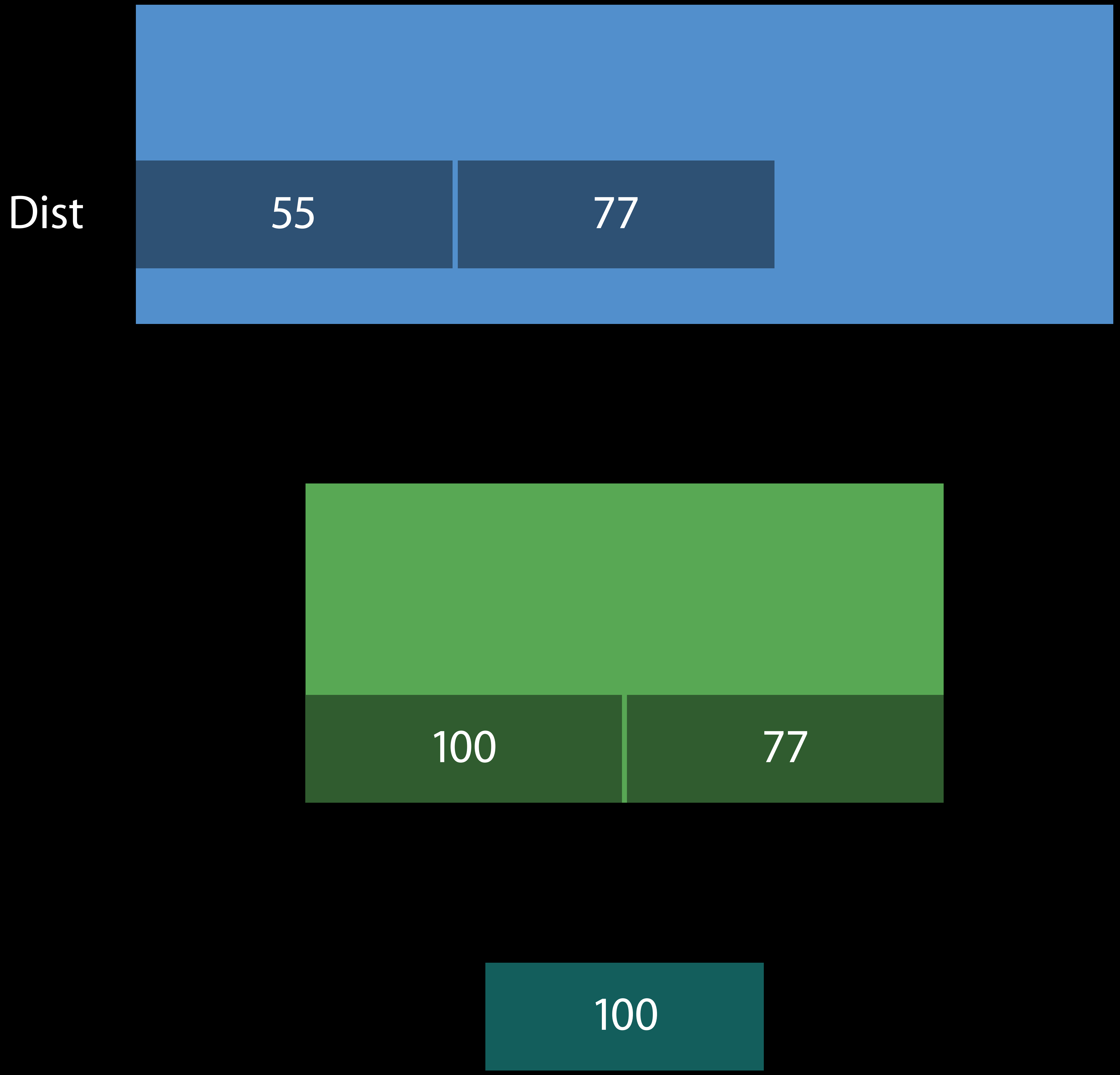
# Data Store 101



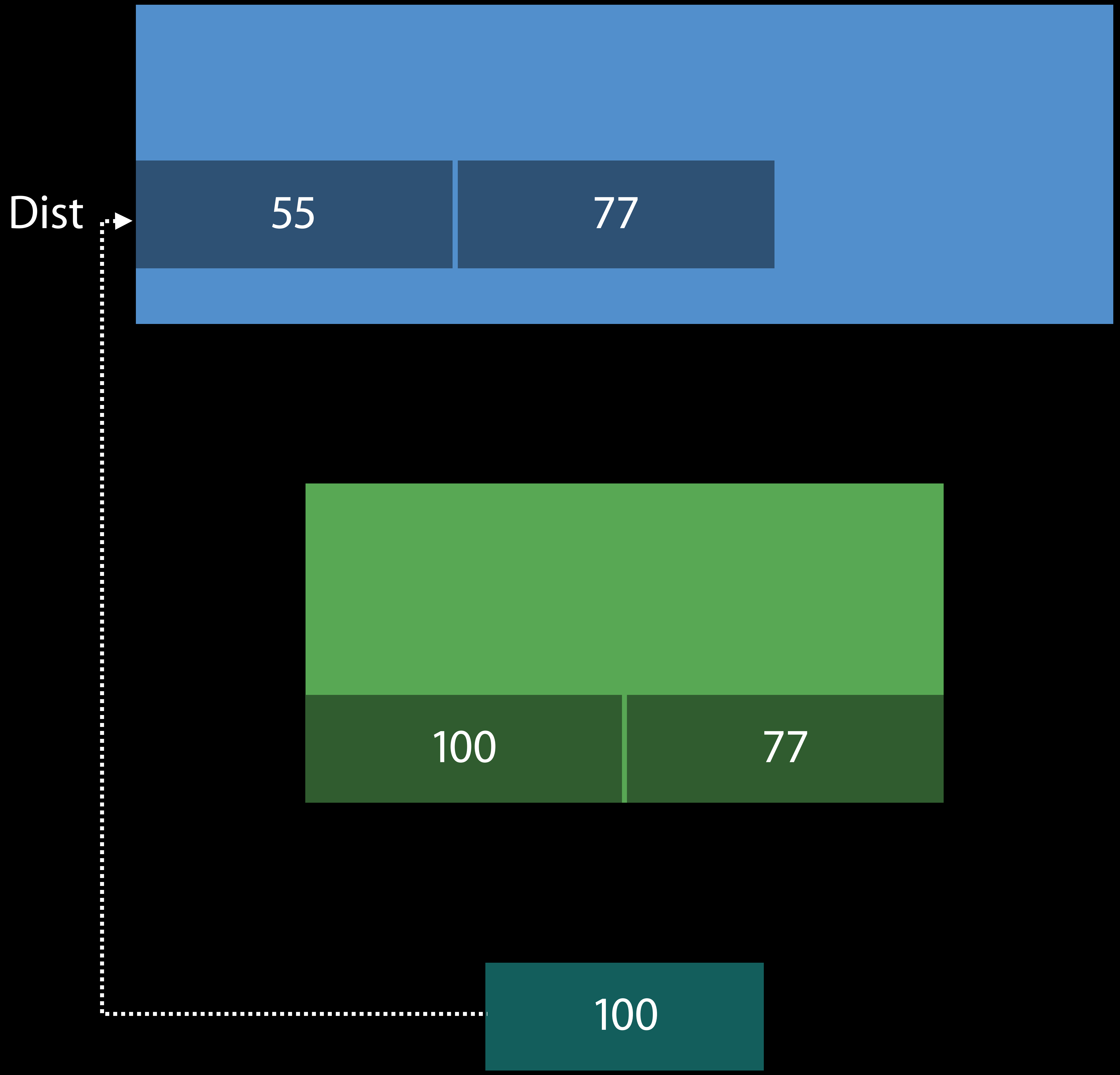
# Data Store 101



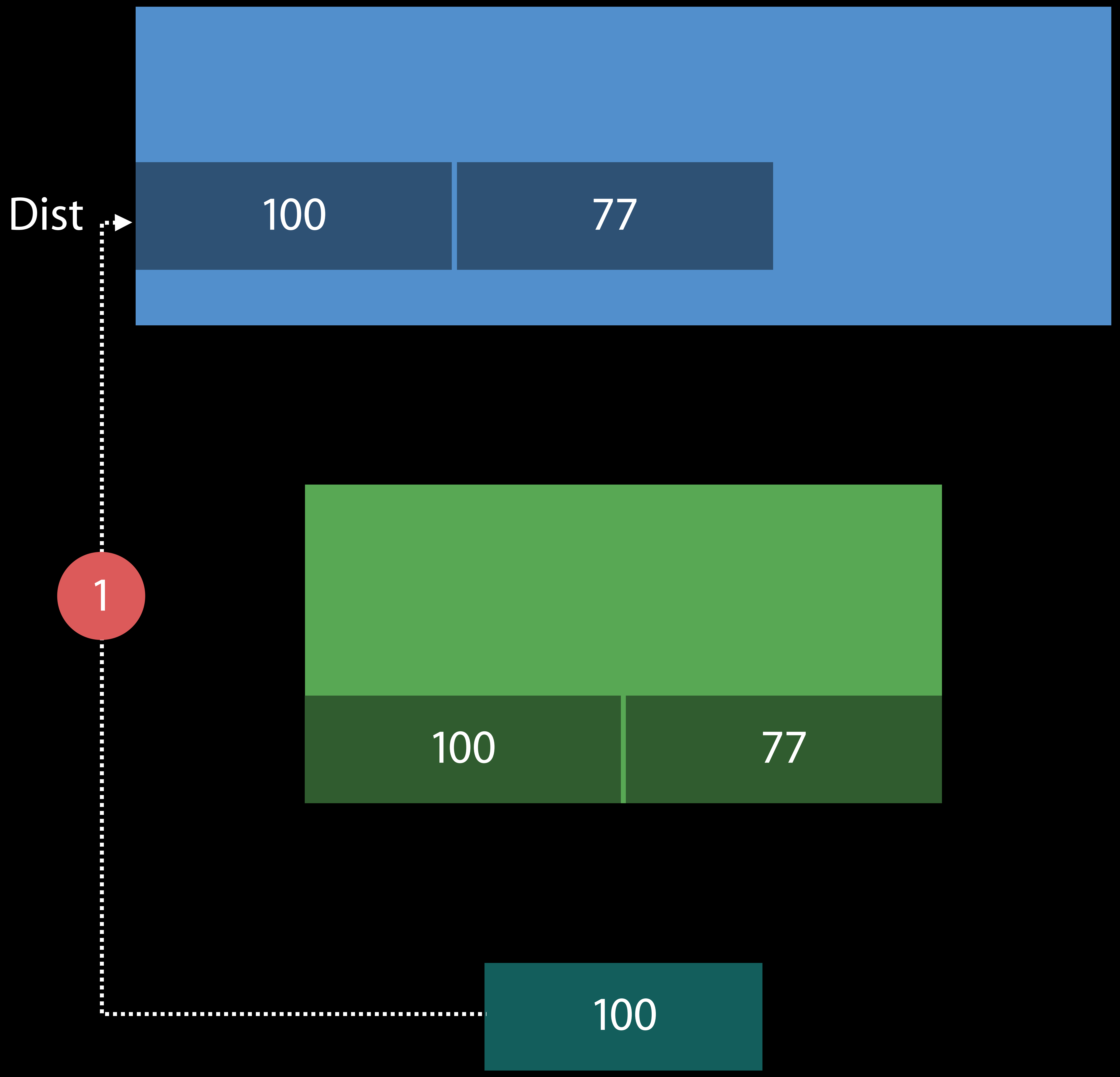
## Can We Store Data Faster?



# Can We Store Data Faster?



# Can We Store Data Faster?



## Non-Temporal Stores

builtin on arm64

NEW

Avoid extra load of a cache line

No data store into the cache

```
void scaledCopy(int *Dst, int *Src, int Scale, int N) {
  for (int i = 0; i < N; i++) {
     Dst[i] = Scale * Src[i];
  }
}
```

## Non-Temporal Stores

NEW

### builtin on arm64

Avoid extra load of a cache line

No data store into the cache

```
void scaledCopy(int *Dst, int *Src, int Scale, int N) {
  for (int i = 0; i < N; i++) {
#if defined(__arm64__)
     __builtin_nontemporal_store(Scale * Src[i], &Dst[i]);
#else
     Dst[i] = Scale * Src[i];
#endif
  }
}
```

## Non-Temporal Store Usage

No reuse

Large chunks of data

Hot loops

## Non-Temporal Store Usage

No reuse

Large chunks of data ✓

Hot loops

# Non-Temporal Store Usage

No reuse

Large chunks of data

Hot loops

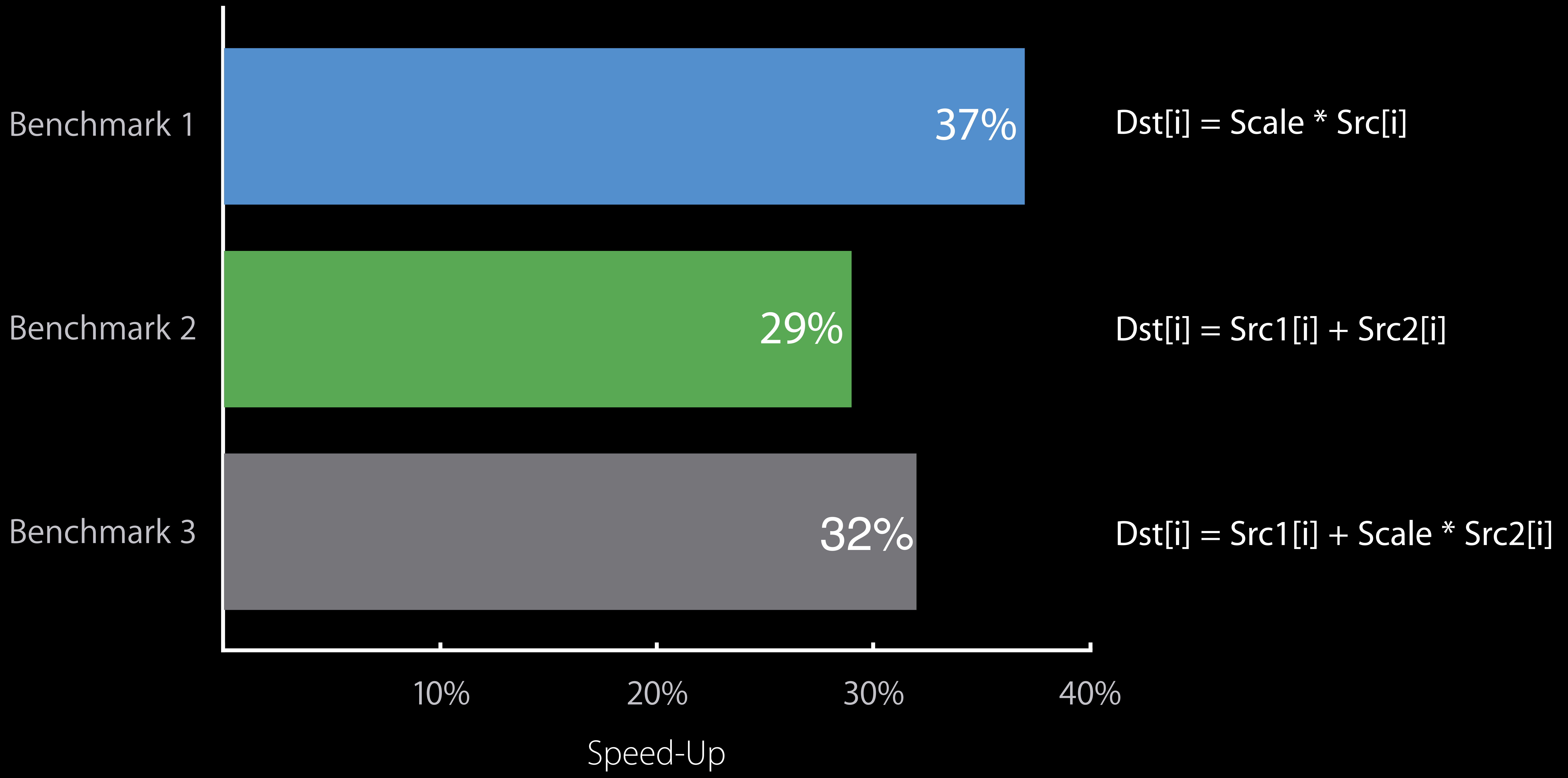
✓

For most loops

Data reuse

✗

# Non-Temporal Store Performance



## Summary

## Summary

LLVM open source

## Summary

LLVM open source

Objective-C class properties

## Summary

LLVM open source

Objective-C class properties

C++ Thread-Local storage

## Summary

LLVM open source

Objective-C class properties

C++ Thread-Local storage

Library support for C++14

## Summary

LLVM open source

Objective-C class properties

C++ Thread-Local storage

Library support for C++14

Over 100 new diagnostics

## Summary

LLVM open source

Objective-C class properties

C++ Thread-Local storage

Library support for C++14

Over 100 new diagnostics

Incremental LTO

## Summary

LLVM open source

Objective-C class properties

C++ Thread-Local storage

Library support for C++14

Over 100 new diagnostics

Incremental LTO

Code generation

## Summary

LLVM open source

Objective-C class properties

C++ Thread-Local storage

Library support for C++14

Over 100 new diagnostics

Incremental LTO

Code generation

arm64 cache tuning

## More Information

https://developer.apple.com/wwdc16/405

## Related Sessions

| | | |
|---|---|---|
| What's New in Swift | Presidio | Tuesday 9:00AM |
| Optimizing App Startup Time | Mission | Wednesday 10:00AM |
| Thread Sanitizer and Static Analysis | Nob Hill | Thursday 10:00AM |
| Debugging Tips and Tricks | Pacific Heights | Friday 1:40PM |

## Labs

| | | |
|---|---|---|
| LLVM Compiler, Objective-C, and C++ Lab | Developer Tools Lab B | Wednesday 12:00PM |
| LLVM Compiler, Objective-C, and C++ Lab | Developer Tools Lab C | Friday 4:30PM |

WWDC16