

Understanding Undefined Behavior

Session 407

Fred Riss, Clang Team

Ryan Govostes, Security Engineering and Architecture Team

Anna Zaks, Program Analysis Team

What is undefined behavior?

The compiler and undefined behavior

Security implications

Tools can help

Swift is safer by default

What Is Undefined Behavior?

“undefined behavior:
behavior for which this International Standard
imposes no requirements.”

What Can the Compiler Do with Undefined Behavior?

What Can the Compiler Do with Undefined Behavior?

Diagnose using warnings or errors

What Can the Compiler Do with Undefined Behavior?

Diagnose using warnings or errors

Act in a documented manner

What Can the Compiler Do with Undefined Behavior?

Diagnose using warnings or errors

Act in a documented manner

Produce unpredictable results

Shift by negative value

Modification of a string literal

Type mismatch

Signed integer overflow

Invalid conversions

Invalid enum value

Out-of-bounds array subscript

Data races

NULL dereference

Use of uninitialized values

C++ dynamic type violation

Division by 0

Shift amounts bigger than type

Misaligned access

Access to an object past end of lifetime

Missing return statement

Undefined Behavior Is About Tradeoffs



Performance over safety

Some Undefined Behavior Examples

Use of an uninitialized variable

```
int uninitialized_variable(int arg) {  
    int value;  
  
    if (arg <= 0)  
        value = 42;  
  
    return arg + value;  
}
```

Some Undefined Behavior Examples

Use of an uninitialized variable

```
int uninitialized_variable(int arg) {  
    int value;  
  
    if (arg <= 0)  
        value = 42;  
  
    return arg + value;  
}
```

Some Undefined Behavior Examples

Use of an uninitialized variable

```
int uninitialized_variable(int arg) {  
    int value;  
  
    if (arg <= 0)  
        value = 42;  
  
    return arg + value;  
}
```

Some Undefined Behavior Examples

Use of an uninitialized variable

```
int uninitialized_variable(int arg) {  
    int value;  
  
    if (arg <= 0)  
        value = 42;  
  
    return arg + value;  
}
```

Some Undefined Behavior Examples

Use of an uninitialized variable

```
int uninitialized_variable(int arg) {  
    int value;  
  
    if (arg <= 0)  
        value = 42;  
  
    return arg + value;  
}
```

 Variable value is used uninitialized whenever the 'if' condition is false



Compiler warnings

Some Undefined Behavior Examples

Use of an uninitialized variable

```
int uninitialized_variable(int arg) {  
    int value;  
  
    if (arg <= 0)  
        value = 42;  
  
    return arg + value;  
}
```

 Variable value is used uninitialized whenever the 'if' condition is false



Compiler warnings



Static analyzer

Some Undefined Behavior Examples

Misaligned pointers

```
char *serialize_misaligned(char *buffer, int a, int b) {  
    *(int *)buffer = a;  
    buffer += sizeof(a);  
    *(int *)buffer = b;  
    buffer += sizeof(b);  
    return buffer;  
}
```

Some Undefined Behavior Examples

Misaligned pointers

```
char *serialize_misaligned(char *buffer, int a, int b) {  
    *(int *)buffer = a;  
    buffer += sizeof(a);  
    *(int *)buffer = b;  
    buffer += sizeof(b);  
    return buffer;  
}
```

Some Undefined Behavior Examples

Misaligned pointers

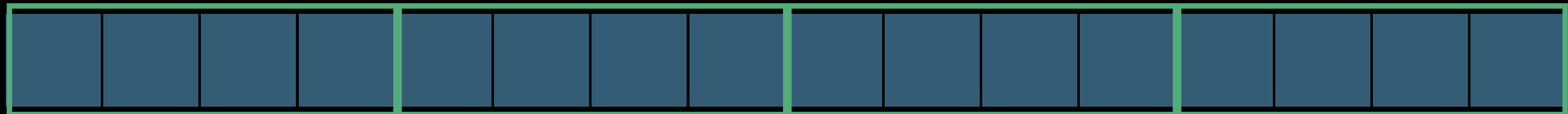
```
char *serialize_misaligned(char *buffer, int a, int b) {  
    *(int *)buffer = a;  
    buffer += sizeof(a);  
    *(int *)buffer = b;  
    buffer += sizeof(b);  
    return buffer;  
}
```

Some Undefined Behavior Examples

Misaligned pointers

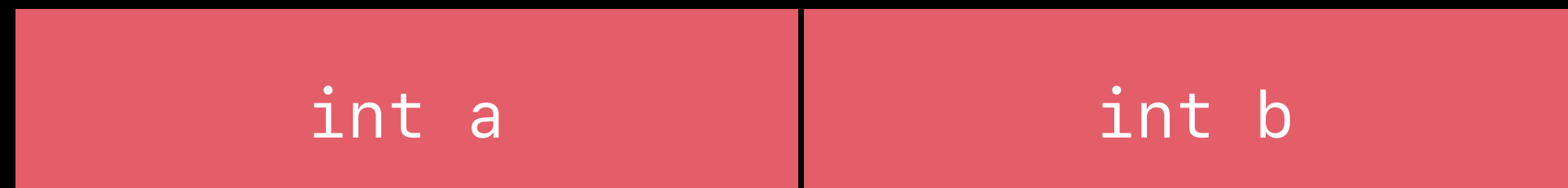
```
char *serialize_misaligned(char *buffer, int a, int b) {  
    *(int *)buffer = a;  
    buffer += sizeof(a);  
    *(int *)buffer = b;  
    buffer += sizeof(b);  
    return buffer;  
}
```

buffer



int a

int b

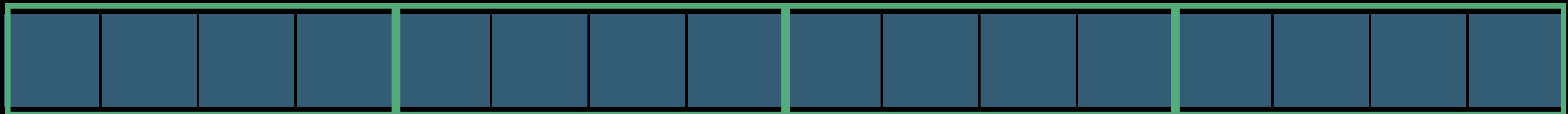


Some Undefined Behavior Examples

Misaligned pointers

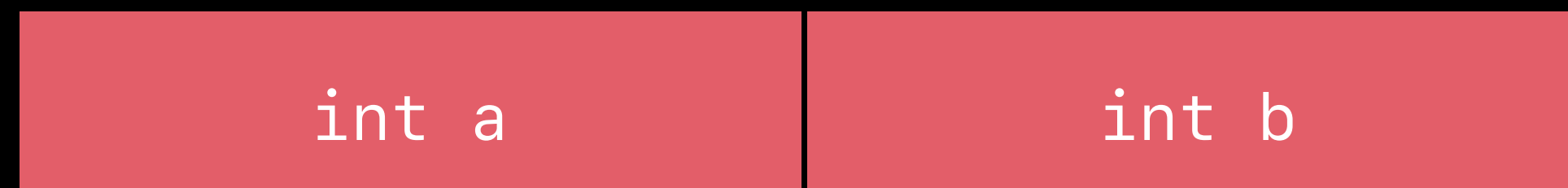
```
char *serialize_misaligned(char *buffer, int a, int b) {  
    *(int *)buffer = a;  
    buffer += sizeof(a);  
    *(int *)buffer = b;  
    buffer += sizeof(b);  
    return buffer;  
}
```

buffer



int a

int b

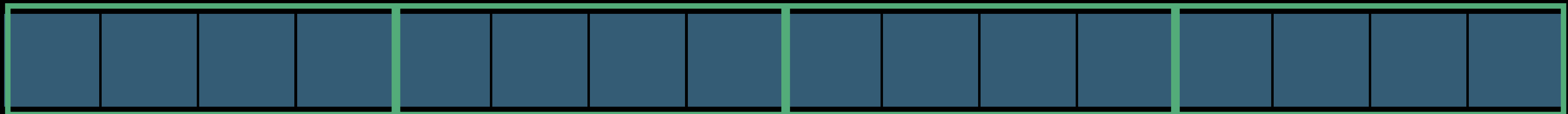


Some Undefined Behavior Examples

Misaligned pointers

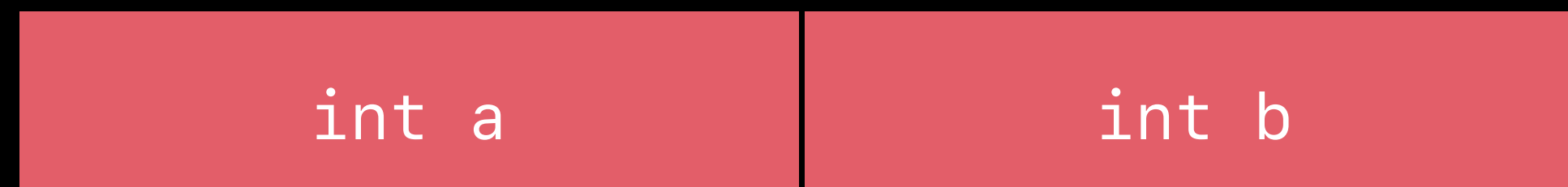
```
char *serialize_misaligned(char *buffer, int a, int b) {  
    *(int *)buffer = a;  
    buffer += sizeof(a);  
    *(int *)buffer = b;  
    buffer += sizeof(b);  
    return buffer;  
}
```

buffer



int a

int b

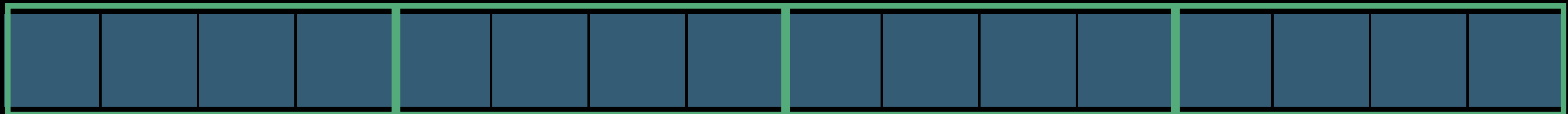


Some Undefined Behavior Examples

Misaligned pointers

```
char *serialize_misaligned(char *buffer, int a, int b) {  
    *(int *)buffer = a;  
    buffer += sizeof(a);  
    *(int *)buffer = b;  
    buffer += sizeof(b);  
    return buffer;  
}
```

buffer





int a

int b



Some Undefined Behavior Examples

Misaligned pointers

```
char *serialize_misaligned(char *buffer, int a, int b) {  
    *(int *)buffer = a;  Store of misaligned address 0x7fff5fbff642 for type 'int', which requires 4 byte alignment  
    buffer += sizeof(a);  
    *(int *)buffer = b;  Store of misaligned address 0x7fff5fbff646 for type 'int', which requires 4 byte alignment  
    buffer += sizeof(b);  
    return buffer;  
}
```



Undefined Behavior Sanitizer

Some Undefined Behavior Examples

Access to an object past end of lifetime

```
int lifetime_issue(int *value) {  
    if (value == NULL) {  
        int default_value = 42;  
        value = &default_value;  
    }  
    return *value;  
}
```

Some Undefined Behavior Examples

Access to an object past end of lifetime

```
int lifetime_issue(int *value) {  
    if (value == NULL) {  
        int default_value = 42;  
        value = &default_value;  
    }  
    return *value;  
}
```

Some Undefined Behavior Examples

Access to an object past end of lifetime

```
int lifetime_issue(int *value) {  
    if (value == NULL) {  
        int default_value = 42;  
        value = &default_value;  
    }  
    return *value;  
}
```

Some Undefined Behavior Examples

Access to an object past end of lifetime

```
int lifetime_issue(int *value) {  
    if (value == NULL) {  
        int default_value = 42;  
        value = &default_value;  
    }  
    return *value;  
}
```

Thread 1: Use of out of scope stack memory



Address Sanitizer

The Compiler and Undefined Behavior

Undefined Behavior Provides Information

Undefined Behavior Provides Information

Undefined Behavior

Information

Signed integers cannot overflow

$x < x+1$

Undefined Behavior Provides Information

Undefined Behavior

Information

Signed integers cannot overflow

$x < x+1$

Pointers are naturally aligned

Can use vector instructions

Undefined Behavior Provides Information

Undefined Behavior

Information

Signed integers cannot overflow

$x < x+1$

Pointers are naturally aligned

Can use vector instructions

NULL cannot be dereferenced

A dereferenced pointer cannot be NULL

Undefined Behavior Provides Information

Undefined Behavior

Information

Signed integers cannot overflow

$x < x+1$

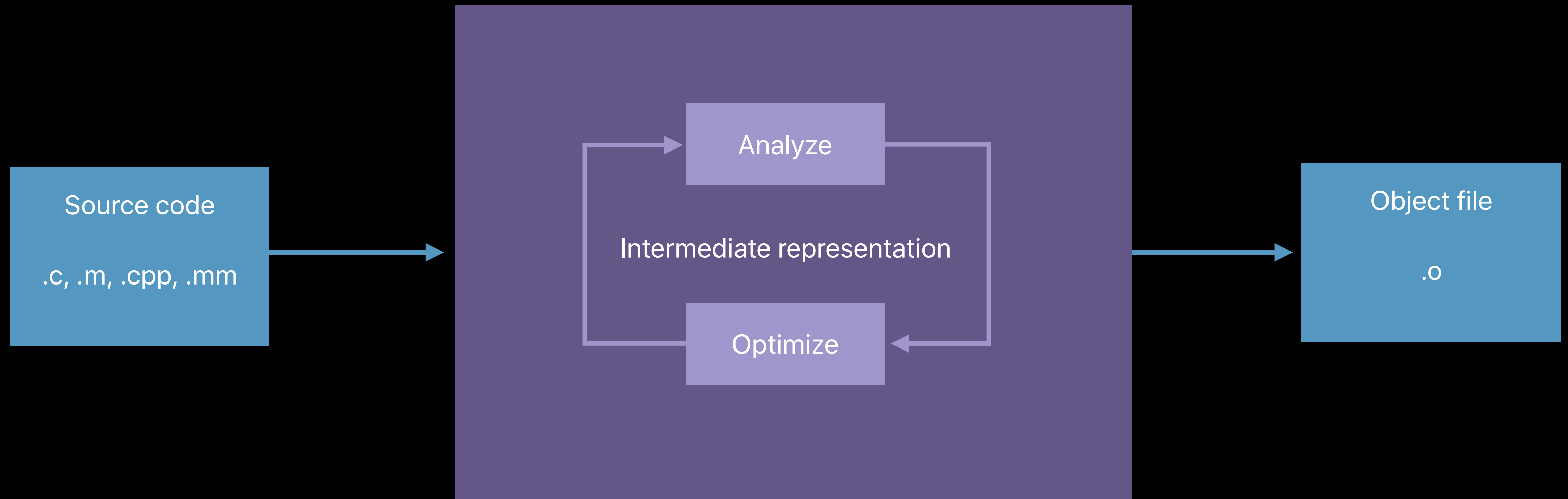
Pointers are naturally aligned

Can use vector instructions

NULL cannot be dereferenced

A dereferenced pointer cannot be NULL

The Compiler Executes an Optimization Pipeline



Dereferencing NULL Might Not Always Crash



Dereferencing NULL Might Not Always Crash



```
int foo(int *P) {  
    int var = *P;  
    return 42;  
}
```

Dereferencing NULL Might Not Always Crash



```
int foo(int *P) {  
    int var = *P;  
    return 42;  
}
```

Dead Code Elimination

```
int foo(int *P) {  
    int var = *P;  
    return 42;  
}
```

Dereferencing NULL Might Not Always Crash



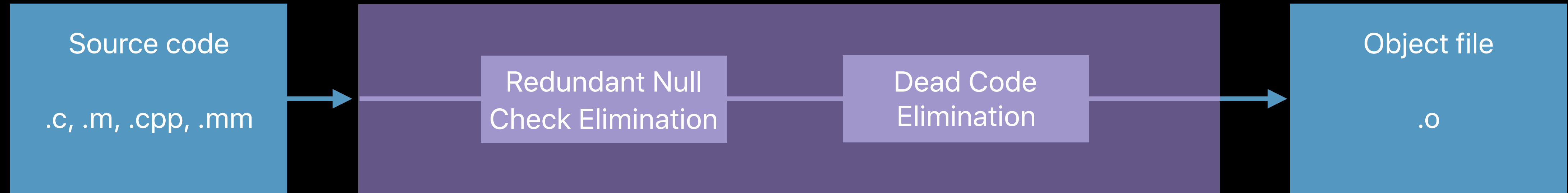
```
int foo(int *P) {  
    int var = *P;  
    return 42;  
}
```

Dead Code Elimination

```
int foo(int *P) {  
    return 42;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

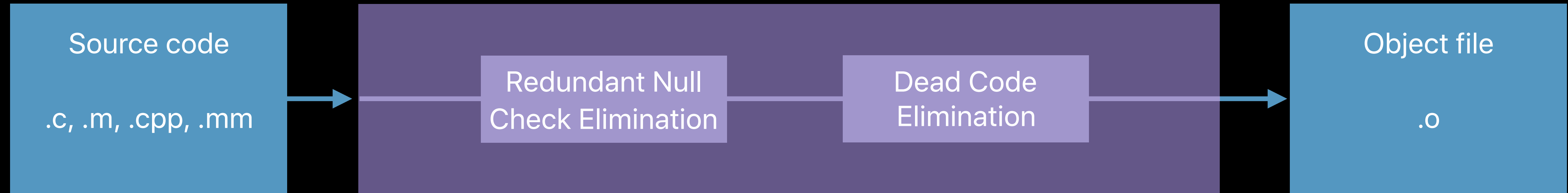
Compiler 1



```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```


Let's Experiment: A Very Simple Optimization Pipeline

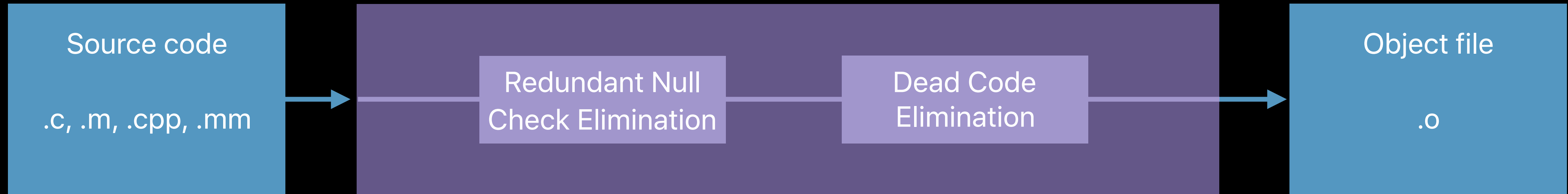
Compiler 1



```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 1



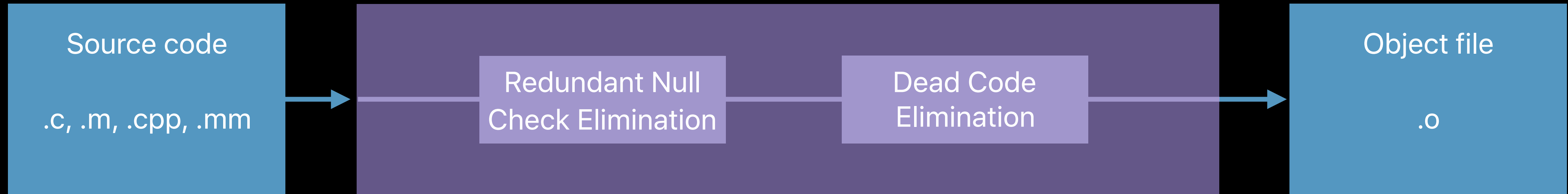
```
void contains_null_check(int *P) {  
    int unused = *P;  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Redundant Null
Check Elimination

```
void contains_null_check(int *P) {  
    int unused = *P;  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 1



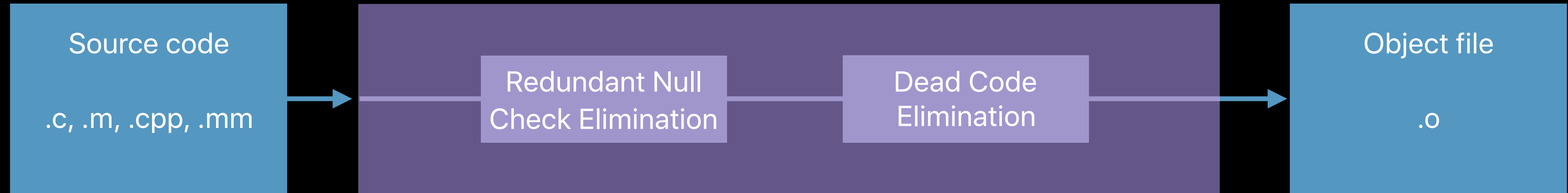
```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Redundant Null
Check Elimination

```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

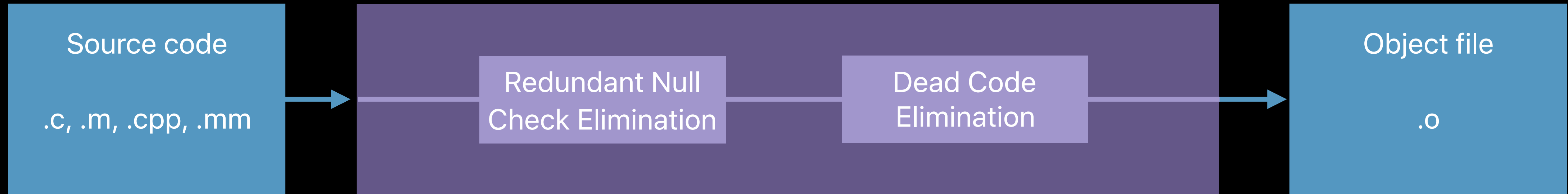
Compiler 1



```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 1



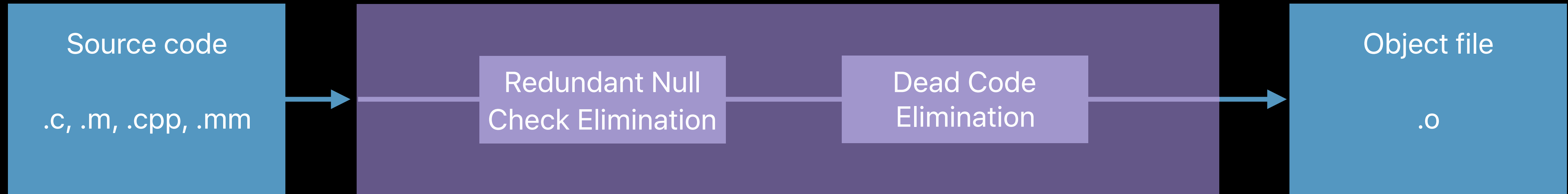
```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
  
    *P = 4;  
}
```

Dead Code
Elimination

```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 1



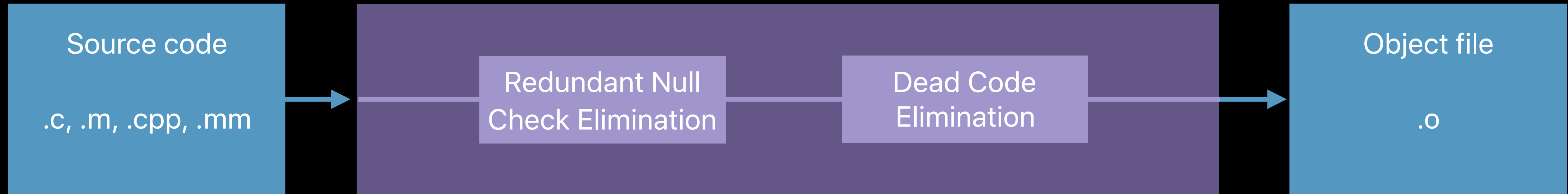
```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
  
    *P = 4;  
}
```

Dead Code
Elimination

```
void contains_null_check(int *P) {  
    ...  
  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 1



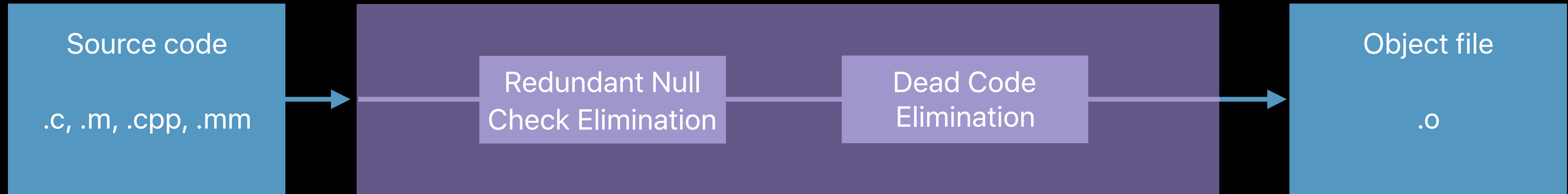
```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Compiler 1

```
void contains_null_check(int *P) {  
    ...  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 1



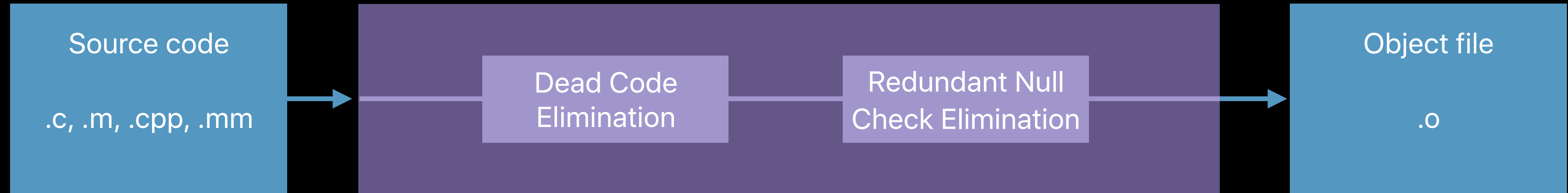
```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Compiler 1

```
void contains_null_check(int *P) {  
    ...  
    *P = 4;  
}
```


Let's Experiment: A Very Simple Optimization Pipeline

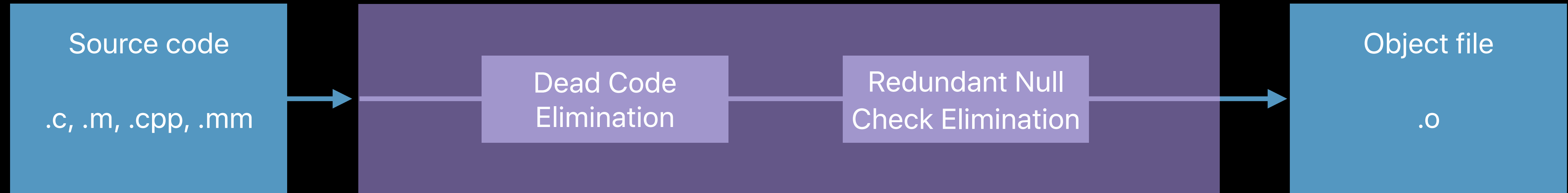
Compiler 2



```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

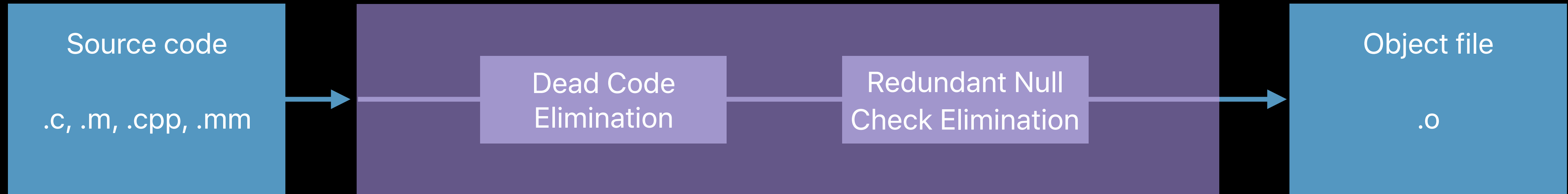
Compiler 2



```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 2



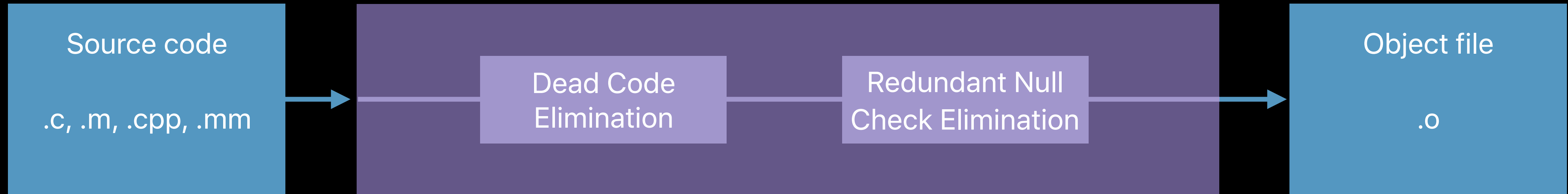
```
void contains_null_check(int *P) {  
    int unused = *P;  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Dead Code
Elimination

```
void contains_null_check(int *P) {  
    int unused = *P;  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 2



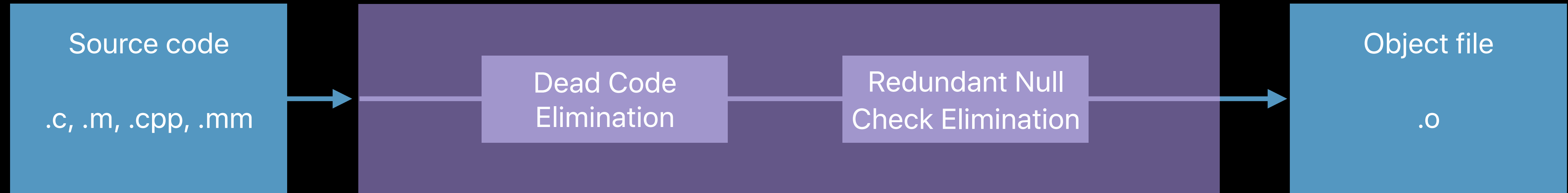
```
void contains_null_check(int *P) {  
    int unused = *P;  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Dead Code
Elimination

```
void contains_null_check(int *P) {  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

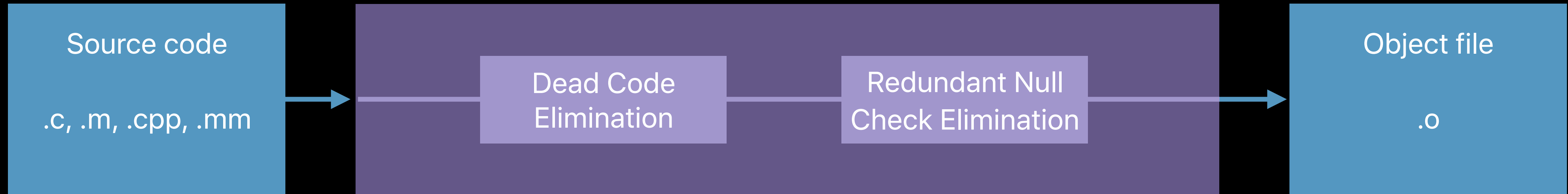
Compiler 2



```
void contains_null_check(int *P) {  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 2



```
void contains_null_check(int *P) {
```

```
...  
if (P == NULL)  
    return;  
*P = 4;
```

```
}
```

Redundant Null
Check Elimination

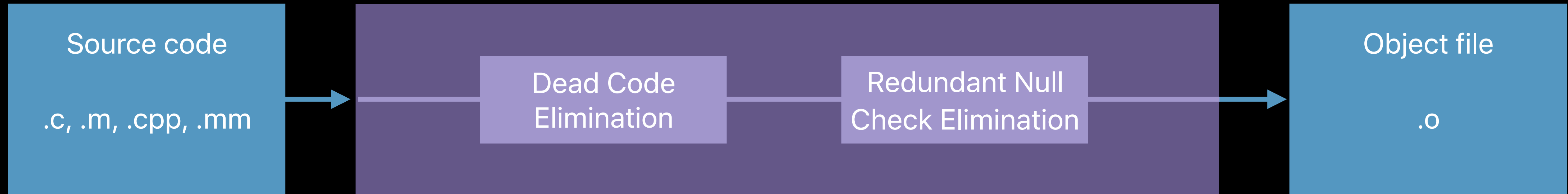
```
void contains_null_check(int *P) {
```

```
...  
if (P == NULL)  
    return;  
*P = 4;
```

```
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 2



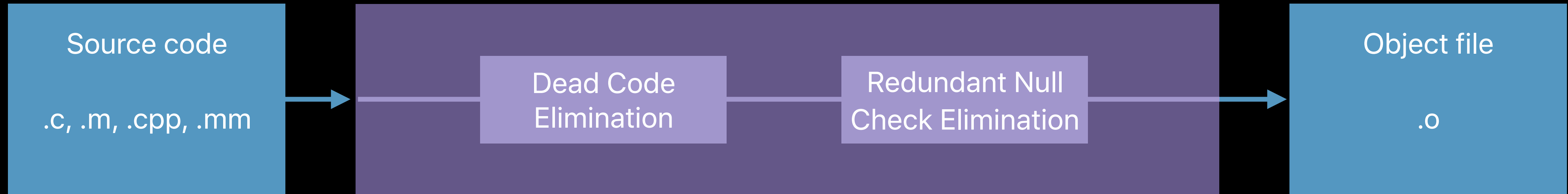
```
void contains_null_check(int *P) {  
    int unused = *P;  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Compiler 2

```
void contains_null_check(int *P) {  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

Compiler 2



```
void contains_null_check(int *P) {  
    int unused = *P;  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Compiler 2

```
void contains_null_check(int *P) {  
  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```


Let's Experiment: A Very Simple Optimization Pipeline

A surprising result

```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Compiler 2

```
void contains_null_check(int *P) {  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Let's Experiment: A Very Simple Optimization Pipeline

A surprising result

```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Compiler 1

```
void contains_null_check(int *P) {  
    ...  
    *P = 4;  
}
```

```
void contains_null_check(int *P) {  
    int unused = *P;  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

Compiler 2

```
void contains_null_check(int *P) {  
    ...  
    if (P == NULL)  
        return;  
    *P = 4;  
}
```

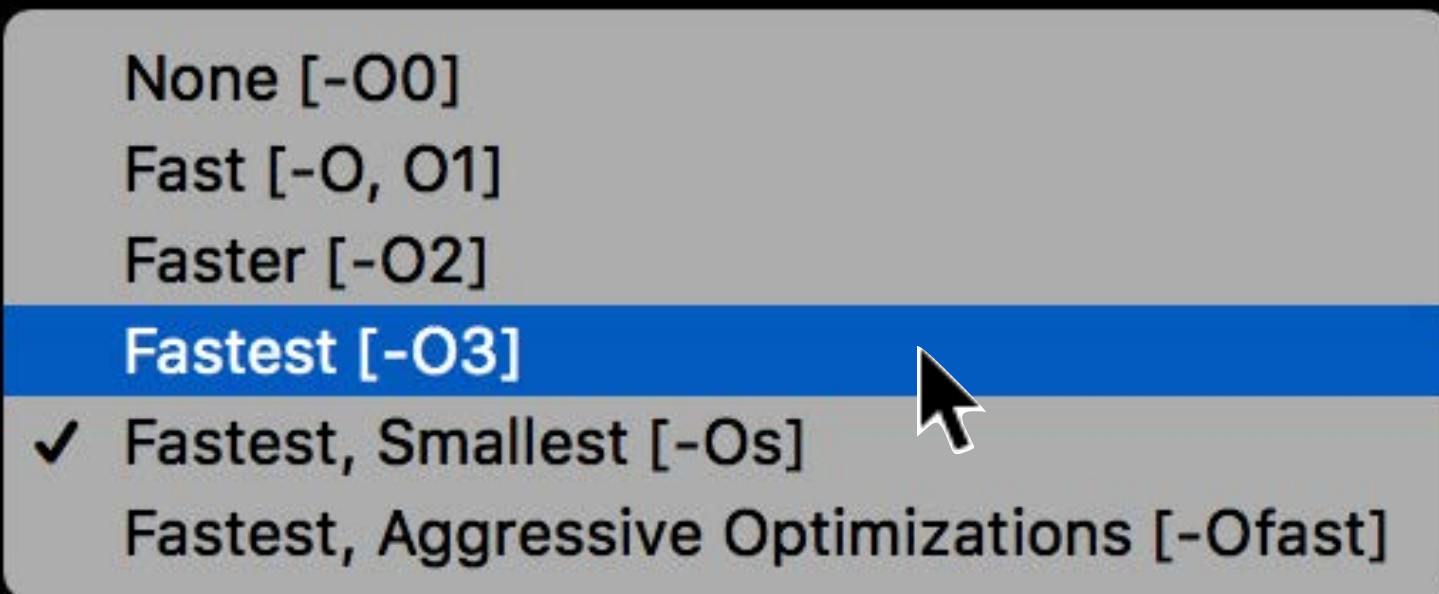
Compiler Behavior Changes More Often Than You Think

Compiler Behavior Changes More Often Than You Think

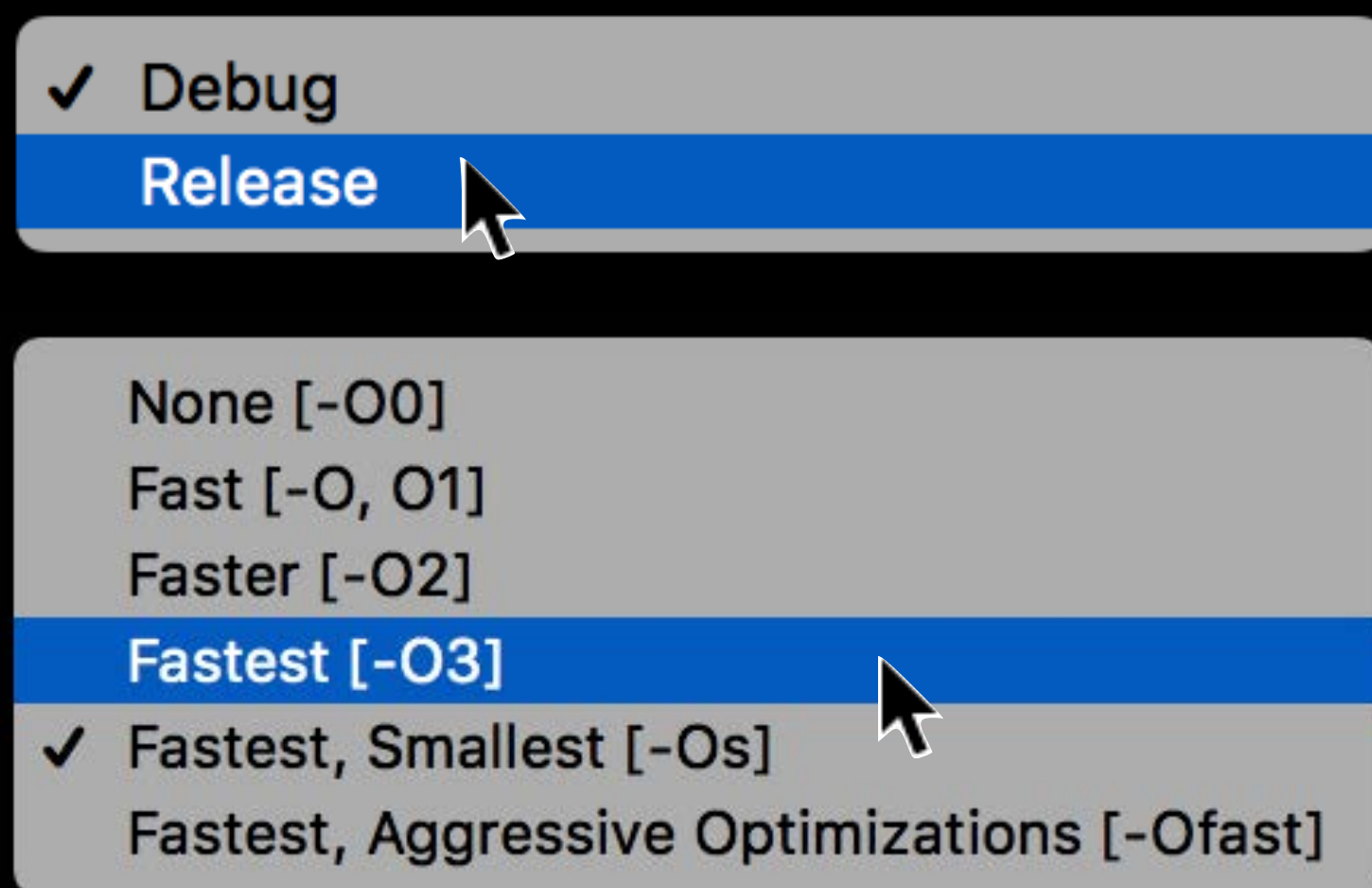
✓ Debug
Release

None [-O0]
Fast [-O, O1]
Faster [-O2]
Fastest [-O3]
✓ Fastest, Smallest [-Os]
Fastest, Aggressive Optimizations [-Ofast]

Compiler Behavior Changes More Often Than You Think



Compiler Behavior Changes More Often Than You Think



Issues with Undefined Behavior

Issues with Undefined Behavior

Undefined behavior is unpredictable

Issues with Undefined Behavior

Undefined behavior is unpredictable

Consequences can affect the whole program

Issues with Undefined Behavior

Undefined behavior is unpredictable

Consequences can affect the whole program

Bugs may be dormant

Security Implications of Undefined Behavior

Ryan Govostes, Security Engineering and Architecture Team



Private Keys

Passwords

Application State



E-mails

Business Documents

Photos

Undefined behavior is at the heart
of many security vulnerabilities

Examples of Security Vulnerabilities

Examples of Security Vulnerabilities

Buffer overflow

Examples of Security Vulnerabilities

Buffer overflow

Use of uninitialized variable

Examples of Security Vulnerabilities

Buffer overflow

Use of uninitialized variable

Use-after-free

Examples of Security Vulnerabilities

Buffer overflow

Use of uninitialized variable

Use-after-free

Double free

Examples of Security Vulnerabilities

Buffer overflow

Use of uninitialized variable

Use-after-free

Double free

Race condition

Defend Your Users

Build secure apps

Protect your reputation

Framework bugs are inherited

Tools Can Help

How Address Sanitizer "Saved" macOS Yosemite

CFString

CFString

/Users/tim

/Library/Caches

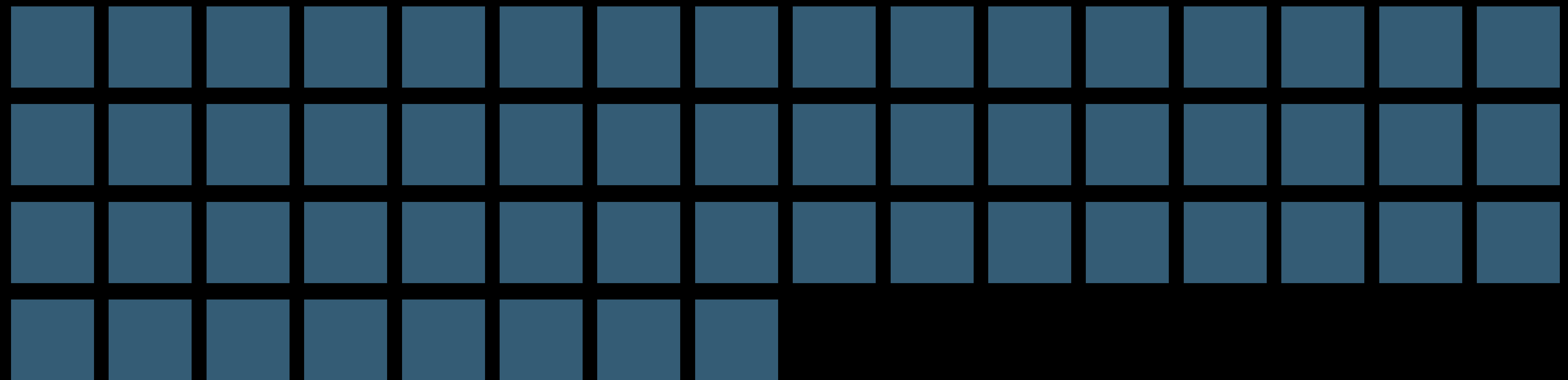
/com.apple.hypercard

/startup.db

CFString



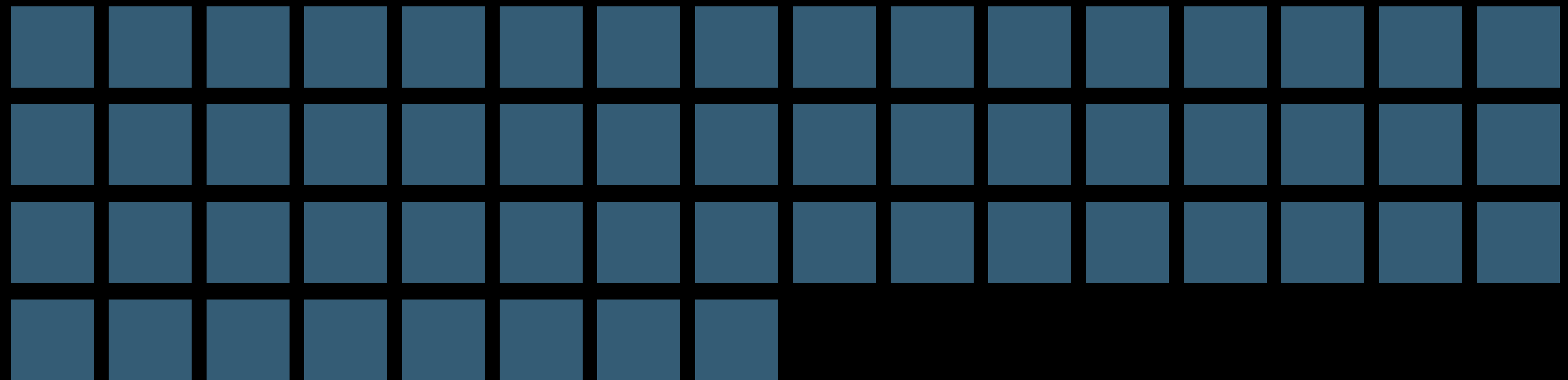
Character Buffer



CFString



Character Buffer



CFString

/Users/tim

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/

U

s

e

r

s

/

t

i

m

/

L

i

b

r

a

r

y

/

C

a

c

h

e

s

/

c

o

m

.

a

p

p

l

e

.

h

y

p

e

r

c

a

r

d

/

s

t

a

r

t

u

p

.

d

b

CFString

/Users/tim

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/	U	s	e	r	s	/	t	i	m	/	L	i	b	r	a
r	y	/	C	a	c	h	e	s	/	c	o	m	.	a	p
p	l	e	.	h	y	p	e	r	c	a	r	d	/	s	t
a	r	t	u	p	.	d	b	NUL							

CFString

/Users/tim

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/

U

s

e

r

s

/

t

i

m

/

L

i

b

r

a

r

y

/

C

a

c

h

e

s

/

c

o

m

.

a

p

p

l

e

.

h

y

p

e

r

c

a

r

d

/

s

t

a

r

t

u

p

.

d

b

CFString

/Users/tim

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/

U

s

e

r

s

/

t

i

m

/

L

i

b

r

a

r

y

/

C

a

c

h

e

s

/

c

o

m

.

a

p

p

l

e

.

h

y

p

e

r

c

a

r

d

/

s

t

a

r

t

u

p

.

d

b

NUL

CFString

/Users/tim

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/	U	s	e	r	s	/	t	i	m	/	L	i	b	r	a
r	y	/	C	a	c	h	e	s	/	c	o	m	.	a	p
p	l	e	.	h	y	p	e	r	c	a	r	d	/	s	t
a	r	t	u	p	.	d	b	NUL							

CFString

/Users/jappleseed4

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/	U	s	e	r	s	/	j	a	p	p	l	e	/	L	i
b	r	a	r	y	/	C	a	c	h	e	s	/	c	o	m
.	a	p	p	l	e	.	h	y	p	e	r	c	a	r	d
/	s	t	a	r	t	u	p	.	d	b	NUL				

CFString

/Users/jappleseed4

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/	U	s	e	r	s	/	j	a	p	p	l	e	s	e	e
d	/	L	i	b	r	a	r	y	/	C	a	c	h	e	s
/	c	o	m	.	a	p	p	l	e	.	h	y	p	e	r
c	a	r	d	/	s	t	a	r	t	u	p	.	d	b	NUL

CFString

/Users/jappleseed4

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/	U	s	e	r	s	/	j	a	p	p	l	e	s	e	e	
d	4	/	L	i	b	r	a	r	y	/	C	a	c	h	e	
s	/	c	o	m	.	a	p	p	l	e	.	h	y	p	e	
r	c	a	r	d	/	s	t	a	r	t	u	p	.	d	b	NUL

CFString

/Users/jappleseed4

/Library/Caches

/com.apple.hypercard

/startup.db

Character Buffer

/	U	s	e	r	s	/	j	a	p	p	l	e	s	e	e	
d	4	/	L	i	b	r	a	r	y	/	C	a	c	h	e	
s	/	c	o	m	.	a	p	p	l	e	.	h	y	p	e	
r	c	a	r	d	/	s	t	a	r	t	u	p	.	d	b	NUL

Tools for Addressing Undefined Behavior

Tools for Addressing Undefined Behavior

Compiler

Tools for Addressing Undefined Behavior

Compiler

Static Analyzer

Tools for Addressing Undefined Behavior

Compiler

Static Analyzer

Address Sanitizer

Tools for Addressing Undefined Behavior

Compiler

Static Analyzer

Address Sanitizer

Thread Sanitizer

Tools for Addressing Undefined Behavior

Compiler

Static Analyzer

Address Sanitizer

Thread Sanitizer

Undefined Behavior Sanitizer

Trust the Compiler



Pay attention to compiler warnings

Every release of Xcode has better warnings

Modernize your project (Editor → Validate Settings)

Trust the Compiler



Pay attention to compiler warnings

Every release of Xcode has better warnings

Modernize your project (Editor → Validate Settings)

Run the Static Analyzer



Explores your code

Analyze during every build

Analyze in Continuous Integration

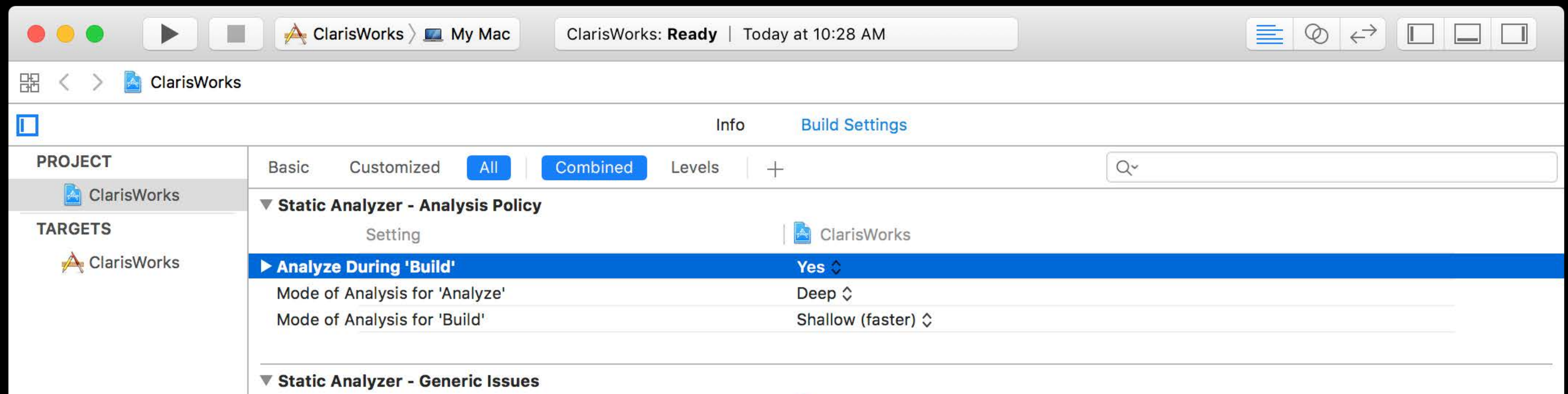
Run the Static Analyzer



Explores your code

Analyze during every build

Analyze in Continuous Integration



Use the Runtime Sanitizers



Use the Runtime Sanitizers



Tool

Undefined Behavior

Use the Runtime Sanitizers



Tool

Undefined Behavior

Address Sanitizer

buffer overflow, use-after-free, double free, use after end of scope

Use the Runtime Sanitizers



Tool

Undefined Behavior

Address Sanitizer

buffer overflow, use-after-free, double free, use after end of scope

Thread Sanitizer

data race

Use the Runtime Sanitizers



Tool	Undefined Behavior	
Address Sanitizer	buffer overflow, use-after-free, double free, use after end of scope	
Thread Sanitizer	data race	
Undefined Behavior Sanitizer	misaligned pointer, null pointer dereference, integer overflow, type mismatch, and more	NEW

ExampleGame > My Mac Running ExampleGame : ExampleGame 2

ExampleGame > ExampleGame > GameScene.m > -touchDownAtPoint:

Buildtime Runtime (2)

ExampleGame - 36620 2 issues

- Threading Issues
 - Data race in -[GameScene sceneDidLoad] at d
 - 'd' is a global variable (0x100008a70)
 - Write of size 8 by thread 1
 - 0 -[GameScene sceneDidLoad]
 - 1 -[SKScene initWithCoder:]
 - 3 start
 - Read of size 8 by thread 5
 - 0 __25-[GameScene sceneDidLoad]
 - 1 __tsan::invoke_and_releas...
 - 2_dispatch_client_callout
- Undefined Behavior
 - Signed integer overflow: 10240000 * 768000 cannot be represented in type 'int'
 - Signed integer overflow
 - 0 -[GameScene touchDown...]
 - 1 -[GameScene mouseDown:]
 - 2 -[SKView mouseDown:]
 - 8 NSApplicationMain
 - 9 main

```
67
68
69 - (void)touchDownAtPoint:(CGPoint)pos {
70     dispatch_queue_t q = dispatch_queue_create("com.example.queue",
71         DISPATCH_QUEUE_CONCURRENT);
72     // Thread 2
73     dispatch_async(q, ^{
74         [d setObject:@42 forKey:@"answer"];
75     });
76     SKShapeNode *n = [_spinnynode copy];
77     n.position = pos;
78     n.strokeColor = [SKColor greenColor];
79     int fastWidth = [self getFastWidth];
80     int fastHeight = [self getFastHeight];
81     if (_expandWidth == YES)
82         n.lineWidth = fastWidth * fastHeight / 10000;
83     [self addChild:n];
84 }
85
86
```

Signed integer overflow: 10240000 * 768000...

Buildtime

Runtime (2)

ExampleGame - 36620 2 issues

Threading Issues

Data race in `-[GameScene scene DidLoad]` at `d`

'd' is a global variable (0x100008a70)

Write of size 8 by thread 1

0 `-[GameScene sceneDidLoad]`

1 `-[SKScene initWithCoder:]`

3 start

Read of size 8 by thread 5

0 `_25-[GameScene sceneDidLoad]`

1 `__tsan::invoke_and_release`

2 `_dispatch_client_callout`

Undefined Behavior

Signed integer overflow:
`10240000 * 768000` cannot be represented in type 'int'

Signed integer overflow

0 `-[GameScene touchDown]`

1 `-[GameScene mouseDown:]`

2 `-[SKView mouseDown:]`

8 `NSApplicationMain`

9 main

Running ExampleGame : ExampleGame

2

ExampleGame > ExampleGame > GameScene.m > -touchDownAtPoint:

```
(-touchDownAtPoint:(CGPoint)pos {
    dispatch_queue_t q = dispatch_queue_create("com.example.queue",
        DISPATCH_QUEUE_CONCURRENT);
    dispatch_async(q, ^{
        [self setObject:@42 forKey:@"answer"];
    });

    CGPoint *n = [_spinnnyNode copy];
    n.position = pos;
    n.strokeColor = [SKColor greenColor];
    n.fastWidth = [self getFastWidth];
    n.fastHeight = [self getFastHeight];
    n.expandWidth == YES)
    [self addChild:n];
}
```

Signed integer overflow: 10240000 * 768000...

Turn on Sanitizers

Edit scheme – diagnostics tab

Runtime Sanitization

Requires recompilation

- Address Sanitizer
 - Detect use of stack after return
 - Thread Sanitizer
 - Pause on issues
- Undefined Behavior Sanitizer
 - Pause on issues

Turn on Sanitizers

Edit scheme – diagnostics tab

Runtime Sanitization
Requires recompilation

- Address Sanitizer
 - Detect use of stack after return
- Thread Sanitizer
 - Pause on issues
- Undefined Behavior Sanitizer
 - Pause on issues

Tools for Addressing Undefined Behavior

Compiler

Static Analyzer

Address Sanitizer

Thread Sanitizer

Undefined Behavior Sanitizer

Tools for Addressing Undefined Behavior

Compiler

Static Analyzer

Address Sanitizer

Thread Sanitizer

Undefined Behavior Sanitizer

Language

Use Safe Language Features

Prefer safe constructs

- Automatic Reference Counting
- C++ smart pointers (`std::shared_ptr`, `std::weak_ptr`)
- Bounds-checked containers (NSArray)

Use Safe Language Features

Prefer safe constructs

- Automatic Reference Counting
- C++ smart pointers (`std::shared_ptr`, `std::weak_ptr`)
- Bounds-checked containers (NSArray)

Consider using Swift

Swift Is Safer by Default

Anna Zaks, Program Analysis Team



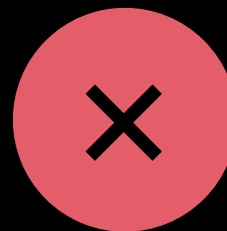
“Undefined behavior is the enemy of safety”

Safety Enforced on Many Levels

Safety Enforced on Many Levels

C Language Family

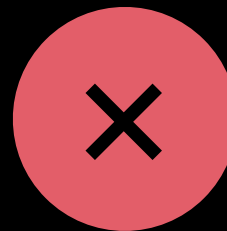
Swift



Null pointer dereferences



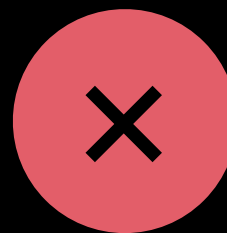
Stricter type system - Optionals



Use of uninitialized variables



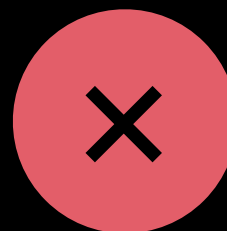
Definite initialization



Buffer and integer overflows



Runtime checks



Use-after-free



ARC (Automatic Reference Counting)

Optionals

Answer to NULL pointer dereferences

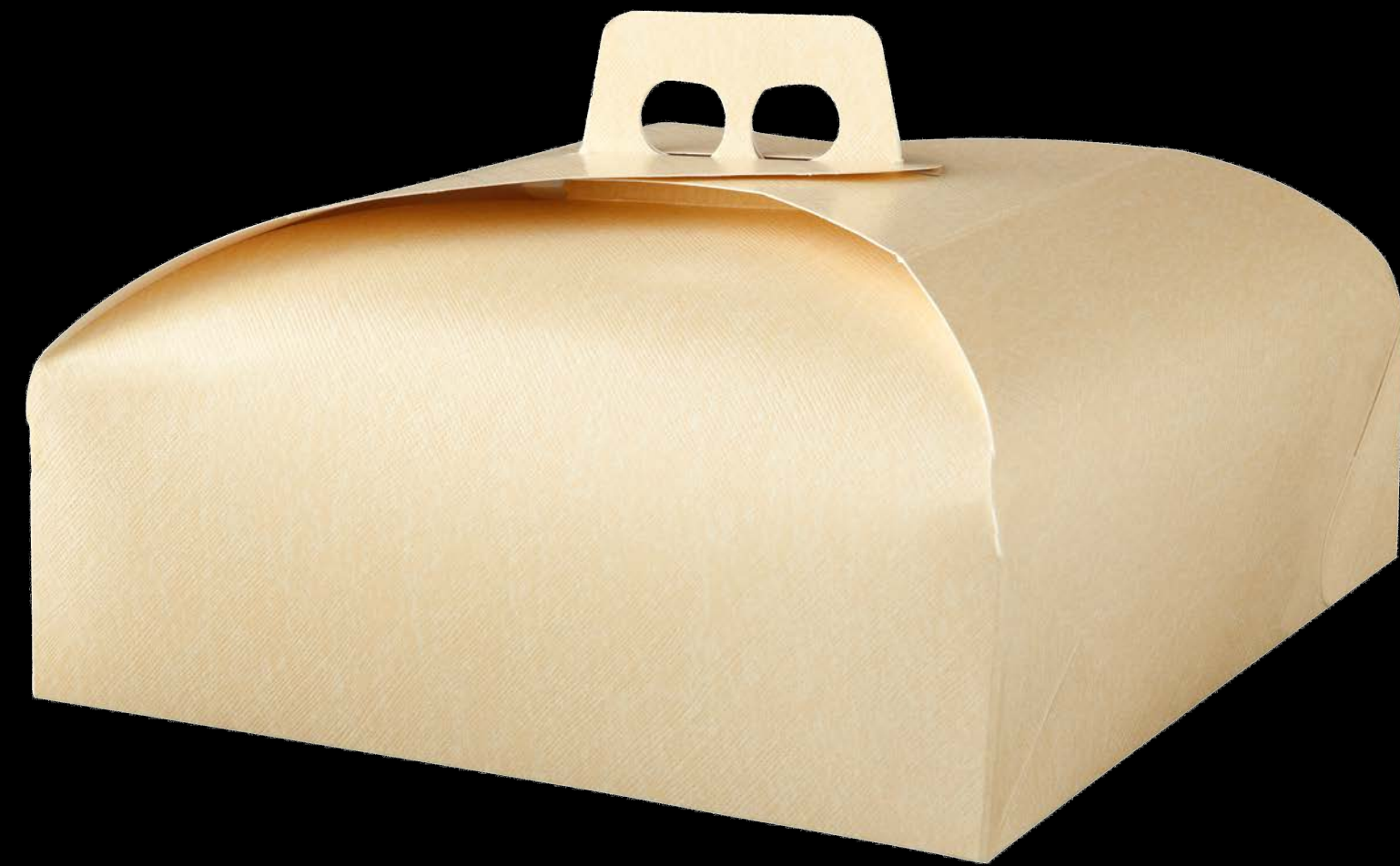
Optionals

Answer to NULL pointer dereferences

Non-optional and optional are different kinds of types



Cake



Cake?

Optionals

Answer to NULL pointer dereferences

Need to check before using

```
func receivePackage() -> Cake?  
...  
  
guard let cake = receivePackage() else {  
    // The cake is a lie.  
    return  
}  
print("Jump with joy! Eat \ \(cake.kind)!")
```



Optionals

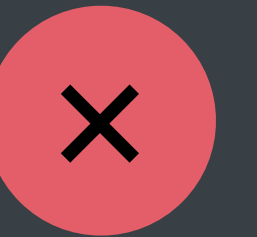
Do not abuse forced unwrap

Optionals

Do not abuse forced unwrap

Optional return type, means the API can return `nil`

```
cake = receivePackage()!
```

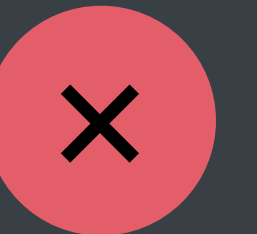


Optionals

Do not abuse forced unwrap

Optional return type, means the API can return `nil`

```
cake = receivePackage()!
```



Use forced unwrap only if:

- You can guarantee the value is never `nil`
- Cannot encode this in the type system
- For example: loading an image asset from the app bundle

Optionals

Implicitly-unwrapped optional (Cake!)

Optionals

Implicitly-unwrapped optional (Cake!)

Compiler does not enforce that value is checked before use

Optionals

Implicitly-unwrapped optional (`Cake!`)

Compiler does not enforce that value is checked before use

Safer than a pointer type in C

- Defined behavior
- Guaranteed to stop on `nil`

Optionals

Implicitly-unwrapped optional (`Cake!`)

Compiler does not enforce that value is checked before use

Safer than a pointer type in C

- Defined behavior
- Guaranteed to stop on `nil`

Useful for delayed initialization

Optionals

Implicitly-unwrapped optional (Cake!)

Compiler does not enforce that value is checked before use

Safer than a pointer type in C

- Defined behavior
- Guaranteed to stop on `nil`

Useful for delayed initialization

May come from Objective-C APIs

Optionals

Nullability annotations for safer ecosystem

Optionals

Nullability annotations for safer ecosystem

Nullability in C languages affects Swift interfaces

```
- (nullable NSView *)ancestorSharedWithView:(nonnull NSView *)aView; // Objective-C  
  
func ancestorShared(with view: NSView) -> NSView? // Swift
```

Optionals

Nullability annotations for safer ecosystem

Nullability in C languages affects Swift interfaces

```
- (nullable UIView *)ancestorSharedWithView:(nonnull UIView *)aView; // Objective-C
```

```
func ancestorShared(with view: UIView) -> UIView? // Swift
```

Optionals

Nullability annotations for safer ecosystem

Nullability in C languages affects Swift interfaces

```
- (nullable UIView *)ancestorSharedWithView:(nonnull UIView *)aView; // Objective-C  
  
func ancestorShared(with view: UIView) -> UIView? // Swift
```

Optionals

Nullability annotations for safer ecosystem

Nullability in C languages affects Swift interfaces

```
- (nullable NSView *)ancestorSharedWithView:(nonnull NSView *)aView; // Objective-C  
  
func ancestorShared(with view: NSView) -> NSView? // Swift
```

Optionals

Nullability annotations for safer ecosystem

Nullability in C languages affects Swift interfaces

```
- (nullable UIView *)ancestorSharedWithView:(nonnull UIView *)aView; // Objective-C  
  
func ancestorShared(with view: UIView) -> UIView? // Swift
```

Apple APIs are annotated for nullability

Optionals

Nullability annotations for safer ecosystem

Nullability in C languages affects Swift interfaces

```
- (nullable UIView *)ancestorSharedWithView:(nonnull UIView *)aView; // Objective-C  
  
func ancestorShared(with view: UIView) -> UIView? // Swift
```

Apple APIs are annotated for nullability

Use nullability on your Objective-C code!

Optionals

Nullability annotations for safer ecosystem

Nullability in C languages affects Swift interfaces

```
- (nullable UIView *)ancestorSharedWithView:(nonnull UIView *)aView; // Objective-C  
  
func ancestorShared(with view: UIView) -> UIView? // Swift
```

Apple APIs are annotated for nullability

Use nullability on your Objective-C code!

Find nullability inconsistencies with tools

- Static Analyzer, -Wnullability, Undefined Behavior Sanitizer

Definite Initialization

Answer to use of uninitialized variables

Definite Initialization

Answer to use of uninitialized variables

Checks that all values are initialized before use

Definite Initialization

Answer to use of uninitialized variables

Checks that all values are initialized before use

```
var myInstance: MyClass

if x > 42 {
    myInstance = MyClass(intValue: 13)
} else {
    myInstance = MyClass(floatValue: 92.3)
}

// myInstance has been initialized on all branches leading here!
myInstance.printIt()
```

Definite Initialization

Answer to use of uninitialized variables

Checks that all values are initialized before use

```
var myInstance: MyClass

if x > 42 {
    myInstance = MyClass(intValue: 13)
} else {
    myInstance = MyClass(floatValue: 92.3)
}

// myInstance has been initialized on all branches leading here!
myInstance.printIt()
```

Definite Initialization

Answer to use of uninitialized variables

Checks that all values are initialized before use

```
var myInstance: MyClass

if x > 42 {
    myInstance = MyClass(intValue: 13)
} else {
    myInstance = MyClass(floatValue: 92.3)
}

// myInstance has been initialized on all branches leading here!
myInstance.printIt()
```

Definite Initialization

Answer to use of uninitialized variables

Checks that all values are initialized before use

```
var myInstance: MyClass

if x > 42 {
    myInstance = MyClass(intValue: 13)
} else {
    myInstance = MyClass(floatValue: 92.3)
}

// myInstance has been initialized on all branches leading here!
myInstance.printIt()
```


Definite Initialization

Answer to use of uninitialized variables

Checks that all values are initialized before use

```
var myInstance: MyClass

if x > 42 {
    myInstance = MyClass(intValue: 13)
} else {
    myInstance = MyClass(floatValue: 92.3)
}

// myInstance has been initialized on all branches leading here!
myInstance.printIt()
```

Runtime Checks

Answer to buffer and integer overflows

Runtime Checks

Answer to buffer and integer overflows

Execution ends on `Array` and `Int` overflows

Runtime Checks

Answer to buffer and integer overflows

Execution ends on `Array` and `Int` overflows

Runtime checking is better than undefined behavior

- Predictable
- Provides security guarantees

Runtime Checks

Answer to buffer and integer overflows

Execution ends on `Array` and `Int` overflows

Runtime checking is better than undefined behavior

- Predictable
- Provides security guarantees

Integer wrapping behavior with `&+`, `&-`, `&*`

Does undefined behavior exist in Swift?

Unsafe Types

Unsafe Types

Need interoperability with C APIs

Unsafe Types

Need interoperability with C APIs

`UnsafePointer<T>`, `UnsafeMutableRawBufferPointer`

Unsafe Types

Need interoperability with C APIs

```
UnsafePointer<T>, UnsafeMutableRawBufferPointer
```

Unsafe Types

Need interoperability with C APIs

```
UnsafePointer<T>, UnsafeMutableRawBufferPointer
```

Use Address Sanitizer

Exclusive Memory Accesses

Enforcement in Swift 4

Exclusive Memory Accesses

Enforcement in Swift 4

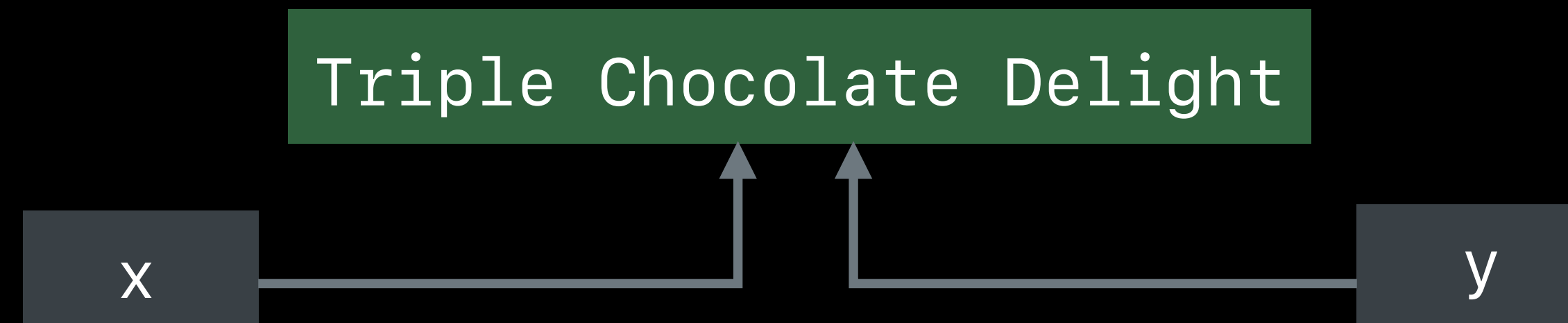
```
func have(_ x: inout Cake, andEat y: inout Cake)
```

Exclusive Memory Accesses

Enforcement in Swift 4

```
func have(_ x: inout Cake, andEat y: inout Cake)
```

```
have(&cake, andEat: &cake)
```

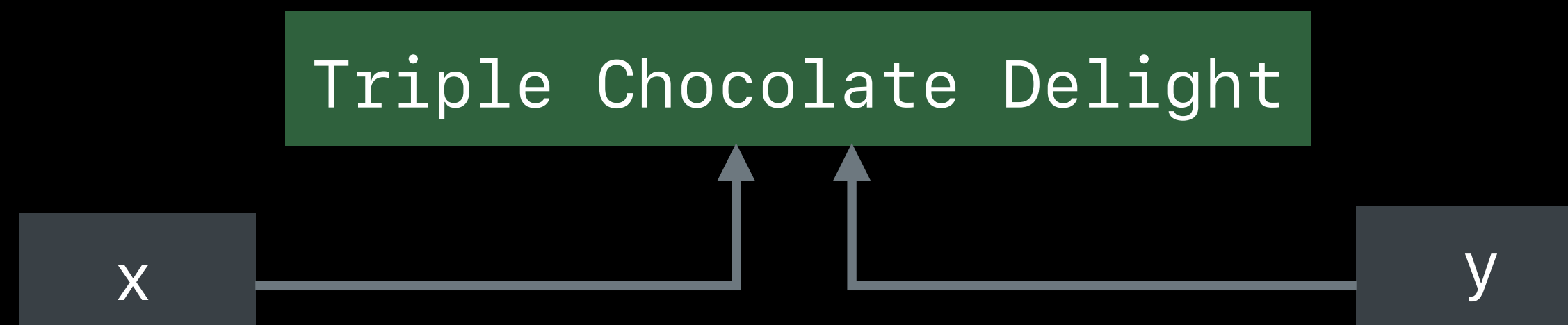


Exclusive Memory Accesses

Enforcement in Swift 4

```
func have(_ x: inout Cake, andEat y: inout Cake)
```

```
have(&cake, andEat: &cake)
```



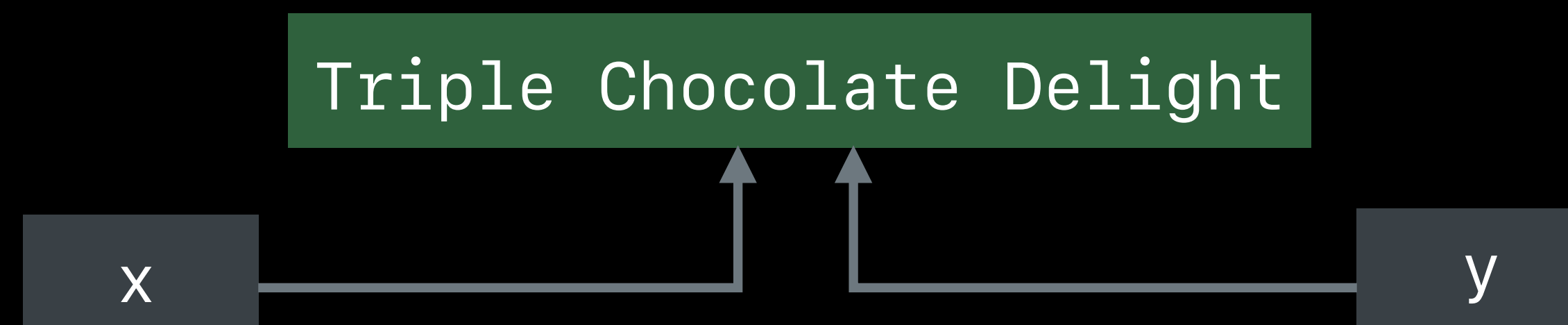
Similar to `restrict` in C but with different default behavior

Exclusive Memory Accesses

Enforcement in Swift 4

```
func have(_ x: inout Cake, andEat y: inout Cake)
```

```
have(&cake, andEat: &cake)
```



Similar to `restrict` in C but with different default behavior

Enforcement of Exclusive Memory Accesses

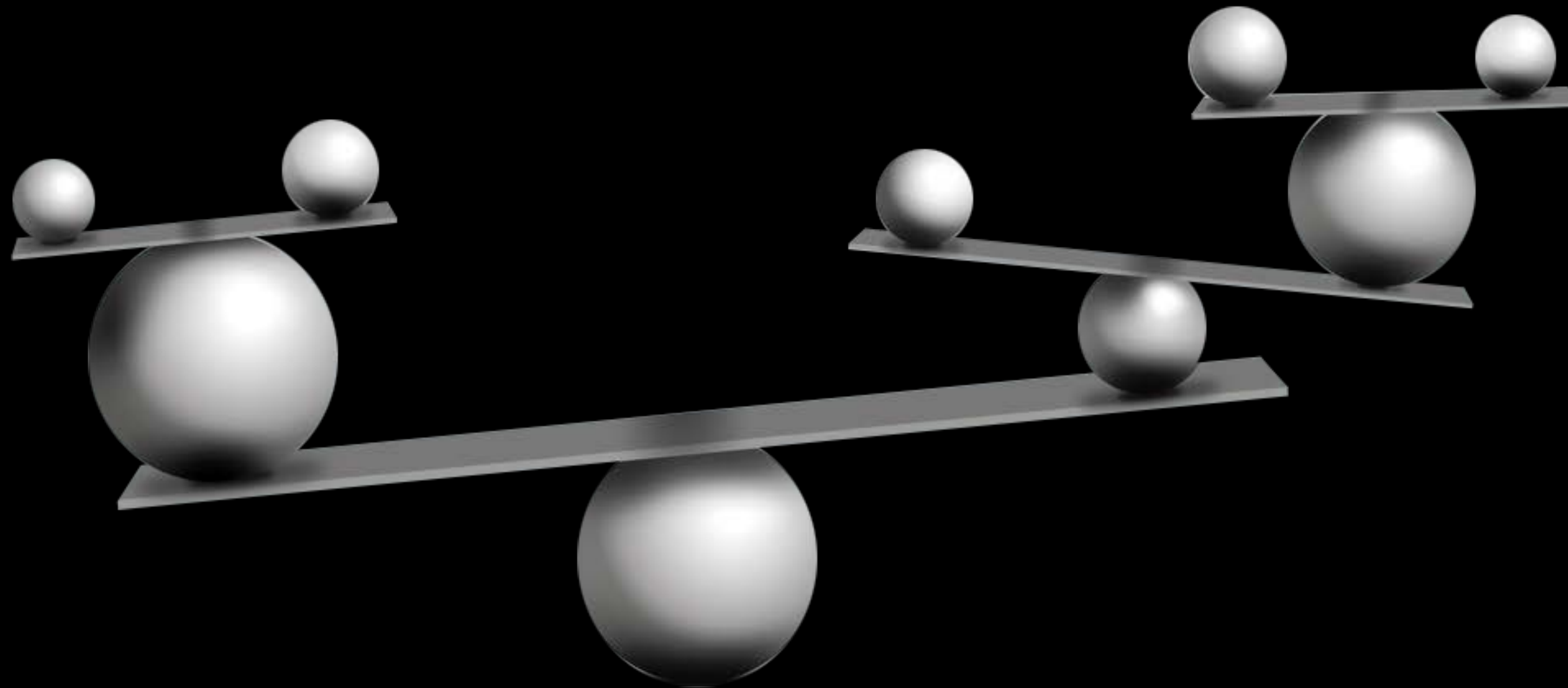
Options considered

Declare to be undefined behavior (like C)

Provide language guarantees

Enforcement of Exclusive Memory Accesses

Intricate balancing act



Enforcement of Exclusive Memory Accesses

Proposed solution

Enforce a slightly stricter language rule

Enforcement at compile time

Enforcement at run time

Guarantee exclusive access within a thread

Enforcement of Exclusive Access Across Threads

Too expensive to check by default

Access races can lead to memory corruption in Swift

Thread Sanitizer catches most violations

Enforcement of Exclusive Access Across Threads

Too expensive to check by default

Access races can lead to memory corruption in Swift

Thread Sanitizer catches most violations



Summary

C languages rely on undefined behavior

Leads to unpredictability and security issues

Swift is safer by default

Use tools to make your code safe and reliable

More Information

<https://developer.apple.com/wwdc17/407>

Related Sessions

Finding Bugs Using Xcode Runtime Tools

WWDC17

What's New in Swift

WWDC17

What's New in LLVM

Hall 2

Thursday 4:10PM

Labs

Performance Profiling and Runtime Analysis Tools Lab

Technology Lab K

Thur 1:00PM – 4:10PM

LLVM Compiler, Objective-C, and C++ Lab

Technology Lab E

Fri 9:00AM – 11:00AM

