

Asynchronous Design Patterns with Blocks, GCD, and XPC

Session 712

Kevin Van Vechten

Core OS

These are confidential sessions—please refrain from streaming, blogging, or taking pictures

Blocks, GCD, and XPC

Overview

- Introduction to Blocks, GCD, and XPC
- New support for Objective-C and ARC in GCD and XPC
- Asynchronous design patterns

Introduction to Blocks

Function Pointer Type

```
typedef void (*callback_function)(char *arg);
```

Block Type

```
typedef void (^callback_block)(char *arg);
```

Using Blocks

```
callback_block b = ^(char *str) {  
    printf("%s\n", str);  
};  
  
b("Hello World");
```

Using Blocks

```
callback_block b = ^(char *str) {  
    printf("%s\n", str);  
};  
  
b("Hello World");
```

Using Blocks

```
callback_block b = ^(char *str) {  
    printf("%s\n", str);  
};  
  
b("Hello World");
```


Using Blocks

```
callback_block b = ^(char *str) {  
    printf("%s\n", str);  
};  
  
b("Hello World");
```

Using Blocks

```
callback_block b = ^(char *str) {  
    printf("%s\n", str);  
};  
  
b("Hello World");
```

Hello World

Benefits of Blocks

- Simplify callback syntax
 - Declare in place
 - Use variables from the enclosing scope
 - Modify variable in the enclosing scope

Three Common Examples

- Completion
- Comparison
- Enumeration

Completion

```
void (*MyCompletionFunction)(NSData *data);
```

Completion

```
void (^MyCompletionBlock)(NSData *data);
```

Completion

```
void (^MyCompletionBlock)(NSData *data);
```

```
extern void MyDownloadAsync(NSURL *url, MyCompletionBlock completion);
```

Completion

```
void (^MyCompletionBlock)(NSData *data);  
  
extern void MyDownloadAsync(NSURL *url, MyCompletionBlock completion);  
  
void MyUpdateImageWithURL(NSImageView *view, NSURL *url)  
{  
    MyDownloadAsync(url, ^(NSData *data) {  
        UIImage *image = [[UIImage alloc] initWithData:data];  
        [view setImage:image];  
        [image release];  
    });  
};
```


Comparison

```
NSComparisonResult (*MyComparatorFunction)(NSString *val1, NSString *val2);
```

Comparison

```
NSComparisonResult (^MyComparatorBlock)(NSString *val1, NSString *val2);
```

Comparison

```
NSComparisonResult (^MyComparatorBlock)(NSString *val1, NSString *val2);  
extern void MySort(NSMutableArray *array, MyComparatorBlock comparator);
```

Comparison

```
NSComparisonResult (^MyComparatorBlock)(NSString *val1, NSString *val2);  
  
extern void MySort(NSMutableArray *array, MyComparatorBlock comparator);  
  
void MySortAlphabetically(NSMutableArray *array)  
{  
    MySort(array, ^(NSString *val1, NSString *val2) {  
        return [val1 compare:val2 options:0];  
    });  
}
```

Comparison

Use variables from the enclosing scope

```
NSComparisonResult (^MyComparatorBlock)(NSString *val1, NSString *val2);  
  
extern void MySort(NSMutableArray *array, MyComparatorBlock comparator);  
  
void MySortAlphabetically(NSMutableArray *array, Boolean ignoreCase)  
{  
    NSStringCompareOptions options = 0;  
  
    if (ignoreCase) options = NSCaseInsensitiveSearch;  
  
    MySort(array, ^(NSString *val1, NSString *val2) {  
        return [val1 compare:val2 options:options];  
    });  
}
```

Iteration

```
void (*MyApplierFunction)(NSNumber *value);
```

Iteration

```
void (^MyApplierBlock)(NSNumber *value);
```

Iteration

```
void (^MyApplierBlock)(NSNumber *value);
```

```
extern void MyApply(NSSet *set, MyApplierBlock applier);
```


Iteration

Modify variables in the enclosing scope

```
void (^MyApplierBlock)(NSNumber *value);

extern void MyApply(NSSet *set, MyApplierBlock applier);

NSNumber *getMaximum(NSSet *set)
{
    __block NSNumber *result = @INT_MIN;

    MyApplySet(set, ^(NSNumber *value) {
        if ([value compare:result] > 0) {
            result = value;
        }
    });

    return result;
}
```

GCD

Grand Central Dispatch

- Enqueue blocks for invocation
 - Thread-safe enqueue
 - Asynchronous execution



Dispatch Blocks

- No arguments
- No return value
- Rely on capturing variables

`^ { ... }`

Dispatch Queues

```
dispatch_queue_t queue;  
  
queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);  
  
printf("Before async\n");  
  
dispatch_async(queue, ^{  
    printf("Hello World\n");  
});  
  
printf("After async\n");
```

Dispatch Queues

```
dispatch_queue_t queue;  
  
queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);  
  
printf("Before async\n");  
  
dispatch_async(queue, ^{  
    printf("Hello World\n");  
});  
  
printf("After async\n");
```

Dispatch Queues

```
dispatch_queue_t queue;  
  
queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);  
  
printf("Before async\n");  
  
dispatch_async(queue, ^{  
    printf("Hello World\n");  
});  
  
printf("After async\n");
```

Dispatch Queues

```
dispatch_queue_t queue;  
  
queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);  
  
printf("Before async\n");  
  
dispatch_async(queue, ^{  
    printf("Hello World\n");  
});  
  
printf("After async\n");
```


Dispatch Queues

```
dispatch_queue_t queue;  
  
queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);  
  
printf("Before async\n");  
  
dispatch_async(queue, ^{  
    printf("Hello World\n");  
});  
  
printf("After async\n");
```

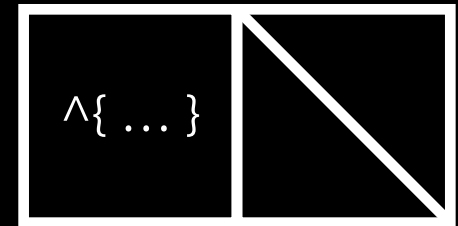
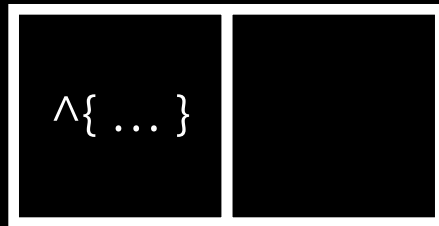
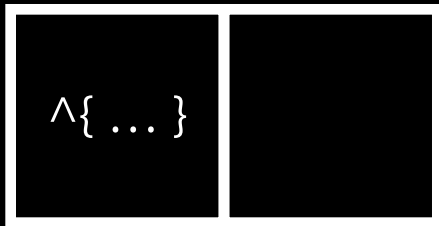
Dispatch Queues

```
dispatch_queue_t queue;  
  
queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);  
  
printf("Before async\n");  
  
dispatch_async(queue, ^{  
    printf("Hello World\n");  
});  
  
printf("After async\n");
```

Before Async
After Async
Hello World

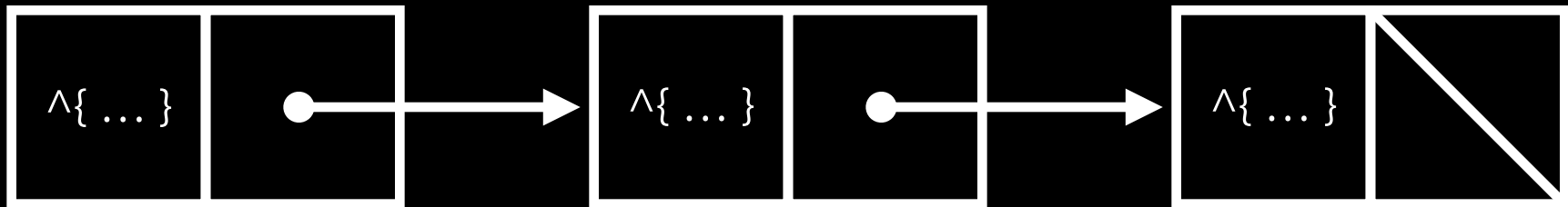
Dispatch Queues

- FIFO
- Atomic Enqueue
- Automatic Dequeue



Dispatch Queues

- FIFO
- Atomic Enqueue
- Automatic Dequeue



Automatic Dequeue



Time 

Automatic Dequeue



Time 

Asynchronous Blocks

- Execute work asynchronously from the main thread
- Keep the main thread responsive to UI events

Call-Callback Pattern

Call-Callback Pattern

```
void MyUpdateImageWithURL(NSImageView *view, NSURL *url)
{
    MyDownloadAsync(url, ^(NSData *data) {
        UIImage *image = [[UIImage alloc] initWithData:data];
        [view setImage:image];
        [image release];
    });
};
```

Call-Callback Pattern

```
void MyUpdateImageWithURL(NSImageView *view, NSURL *url)
{
    MyDownloadAsync(url, ^(NSData *data) {
        UIImage *image = [[UIImage alloc] initWithData:data];
        dispatch_async(dispatch_get_main_queue(), ^{
            [view setImage:image];
        });
        [image release];
    });
};
```

XPC

Communicate Between Processes



- Simple interface to look up services by name
- Send and receive asynchronous messages
- Deliver replies as blocks submitted to queues

Services



Bundles contained inside main app bundle

- Services launched on demand
- Fault isolation
 - Handle crashes gracefully
- Privilege separation
 - Run with different sandbox entitlements

Call-Callback Pattern

```
void MyDecodeImageRemote(NSData *data, MyCallbackBlock callback)
{
    dispatch_queue_t queue;
    xpc_connection_t connection;
    queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);
    connection = xpc_connection_create("com.example.render", queue);

    xpc_connection_set_event_handler(connection, ^(xpc_object_t reply) {
        if (xpc_get_type(reply) != XPC_TYPE_ERROR) {
            size_t len;
            void *bytes = xpc_dictionary_get_data(reply, "decoded", &len);
            callback([NSData dataWithBytes:bytes length:len]);
        }
    });
    xpc_resume(c);
    ...
}
```

Call-Callback Pattern

```
void MyDecodeImageRemote(NSData *data, MyCallbackBlock callback)
{
    dispatch_queue_t queue;
    xpc_connection_t connection;
    queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);
    connection = xpc_connection_create("com.example.render", queue);

    xpc_connection_set_event_handler(connection, ^(xpc_object_t reply) {
        if (xpc_get_type(reply) != XPC_TYPE_ERROR) {
            size_t len;
            void *bytes = xpc_dictionary_get_data(reply, "decoded", &len);
            callback([NSData dataWithBytes:bytes length:len]);
        }
    });
    xpc_resume(c);
    ...
}
```

Call-Callback Pattern

```
void MyDecodeImageRemote(NSData *data, MyCallbackBlock callback)
{
    dispatch_queue_t queue;
    xpc_connection_t connection;
    queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);
    connection = xpc_connection_create("com.example.render", queue);

    xpc_connection_set_event_handler(connection, ^(xpc_object_t reply) {
        if (xpc_get_type(reply) != XPC_TYPE_ERROR) {
            size_t len;
            void *bytes = xpc_dictionary_get_data(reply, "decoded", &len);
            callback([NSData dataWithBytes:bytes length:len]);
        }
    });
    xpc_resume(c);
    ...
}
```


Call-Callback Pattern

```
void MyDecodeImageRemote(NSData *data, MyCallbackBlock callback)
{
    dispatch_queue_t queue;
    xpc_connection_t connection;
    queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);
    connection = xpc_connection_create("com.example.render", queue);

    xpc_connection_set_event_handler(connection, ^(xpc_object_t reply) {
        if (xpc_get_type(reply) != XPC_TYPE_ERROR) {
            size_t len;
            void *bytes = xpc_dictionary_get_data(reply, "decoded", &len);
            callback([NSData dataWithBytes:bytes length:len]);
        }
    });
    xpc_resume(c);
    ...
}
```

Call-Callback Pattern

```
void MyDecodeImageRemote(NSData *data, MyCallbackBlock callback)
{
    dispatch_queue_t queue;
    xpc_connection_t connection;
    queue = dispatch_queue_create("com.example.queue", DISPATCH_QUEUE_SERIAL);
    connection = xpc_connection_create("com.example.render", queue);

    xpc_connection_set_event_handler(connection, ^(xpc_object_t reply) {
        if (xpc_get_type(reply) != XPC_TYPE_ERROR) {
            size_t len;
            void *bytes = xpc_dictionary_get_data(reply, "decoded", &len);
            callback([NSData dataWithBytes:bytes length:len]);
        }
    });
    xpc_resume(c);

```

...

Call-Callback Pattern

```
...  
    xpc_dictionary_t message;  
  
    message = xpc_dictionary_create(NULL, NULL, 0);  
    xpc_dictionary_set_data(message, "encoded", [data bytes], [data length]);  
    xpc_connection_send_message(connection, message);  
  
    xpc_release(message);  
}
```

Call-Callback Pattern

```
...  
    xpc_dictionary_t message;  
  
    message = xpc_dictionary_create(NULL, NULL, 0);  
    xpc_dictionary_set_data(message, "encoded", [data bytes], [data length]);  
    xpc_connection_send_message(connection, message);  
  
    xpc_release(message);  
}
```

Call-Callback Pattern

```
...  
    xpc_dictionary_t message;  
  
    message = xpc_dictionary_create(NULL, NULL, 0);  
    xpc_dictionary_set_data(message, "encoded", [data bytes], [data length]);  
    xpc_connection_send_message(connection, message);  
  
    xpc_release(message);  
}
```

Call-Callback Pattern

```
...  
    xpc_dictionary_t message;  
  
    message = xpc_dictionary_create(NULL, NULL, 0);  
    xpc_dictionary_set_data(message, "encoded", [data bytes], [data length]);  
    xpc_connection_send_message(connection, message);  
  
    xpc_release(message);  
}
```

Blocks and Objective-C

Daniel Steffen
Core OS

Blocks and Objective-C

- Blocks are created on the stack
- May be copied to the heap
 - const-copy scalar values
 - Objective-C objects are retained
 - Other pointers copied as values, NOT their storage

Object Lifetime

```
- (void)performAsyncWorkWithCallback:(id)obj onQueue:(dispatch_queue_t)q {
```

```
}
```

Object Lifetime

```
- (void)performAsyncWorkWithCallback:(id)obj onQueue:(dispatch_queue_t)q {  
    dispatch_async(self.queue, ^{  
        [self performWork];  
    });  
}
```

Object Lifetime



```
- (void)performAsyncWorkWithCallback:(id)obj onQueue:(dispatch_queue_t)q {  
    dispatch_async(self.queue, ^{  
        [self performWork];  
  
        dispatch_async(q, ^{  
            [obj callback];  
        });  
    });  
}
```

Object Lifetime

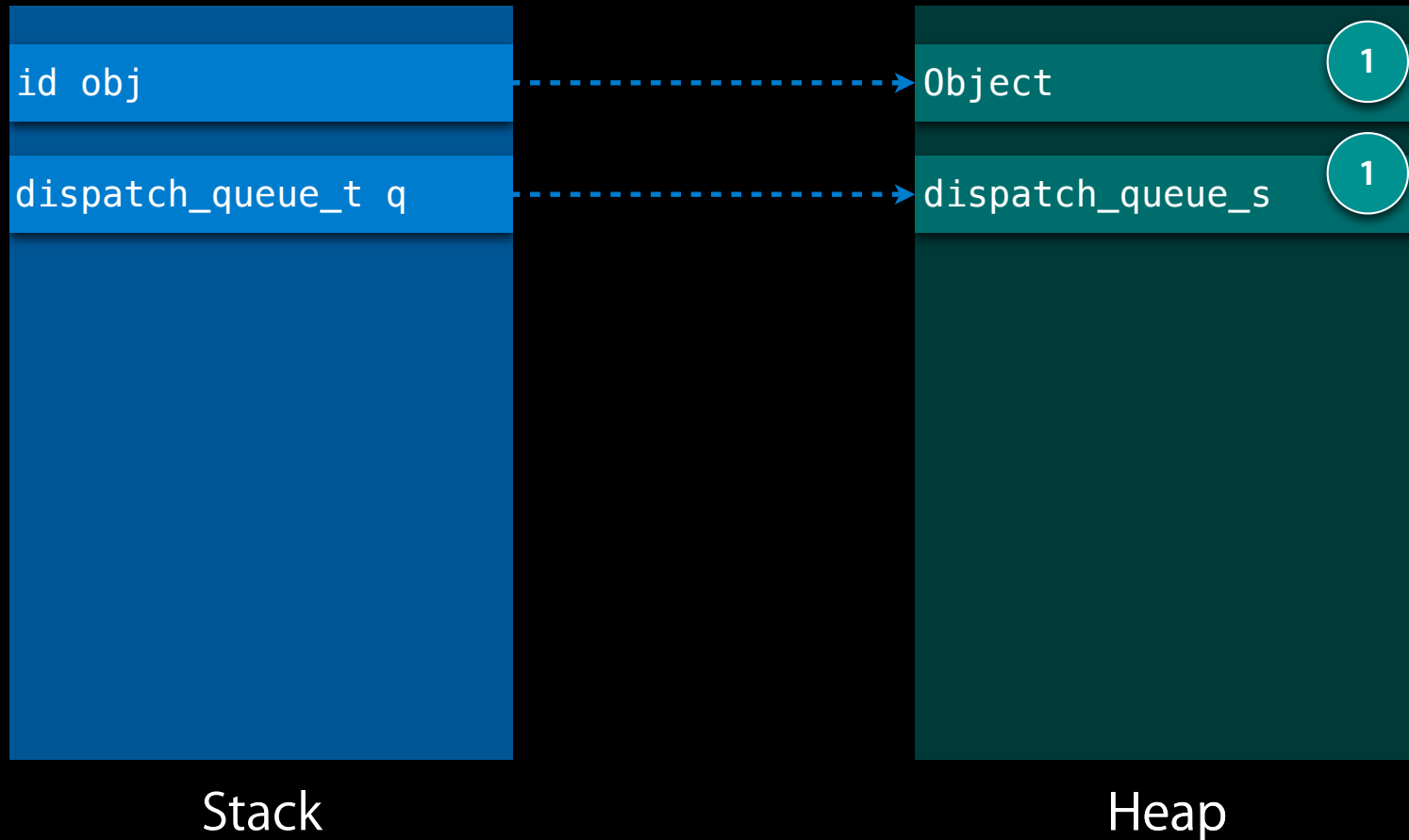


Stack

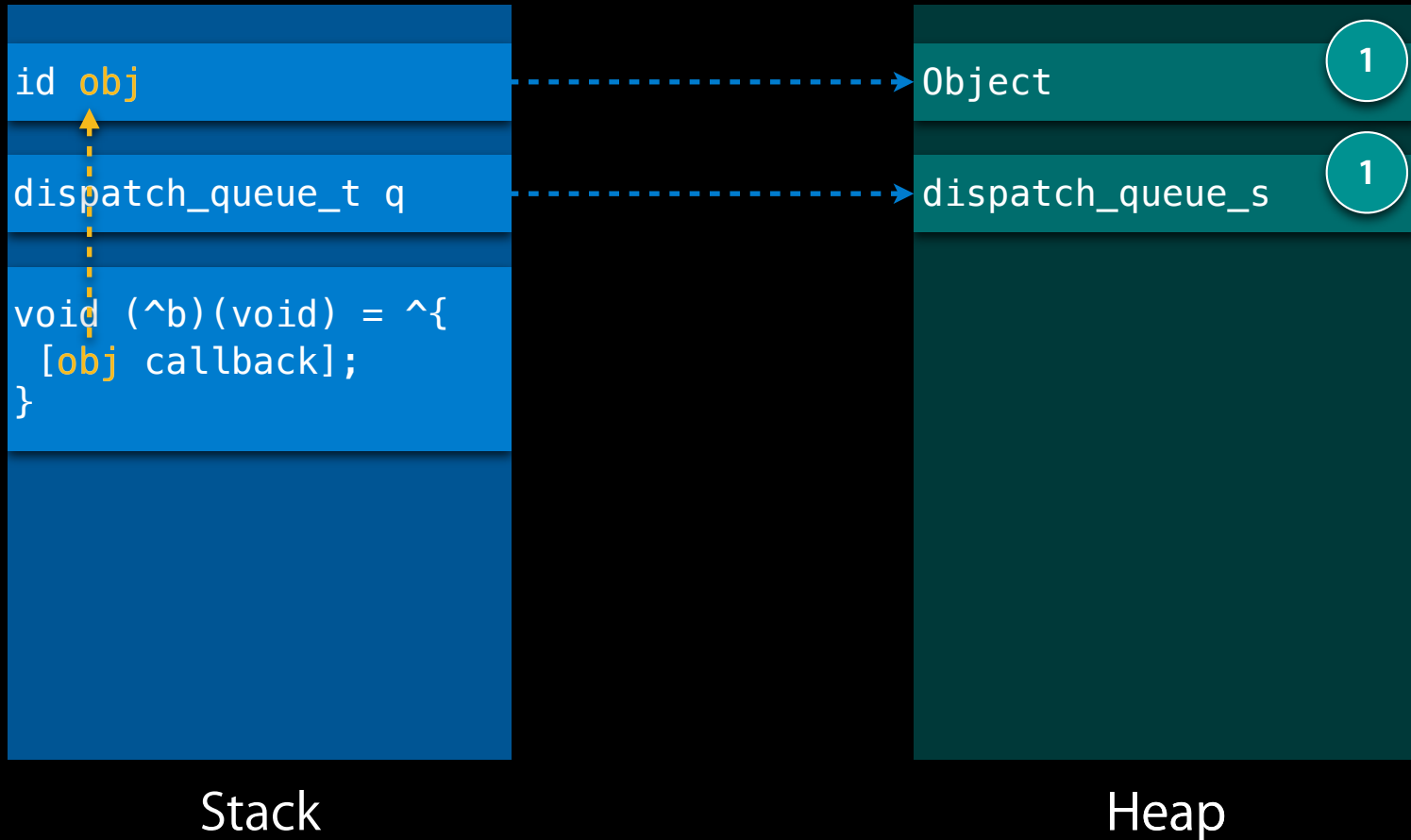


Heap

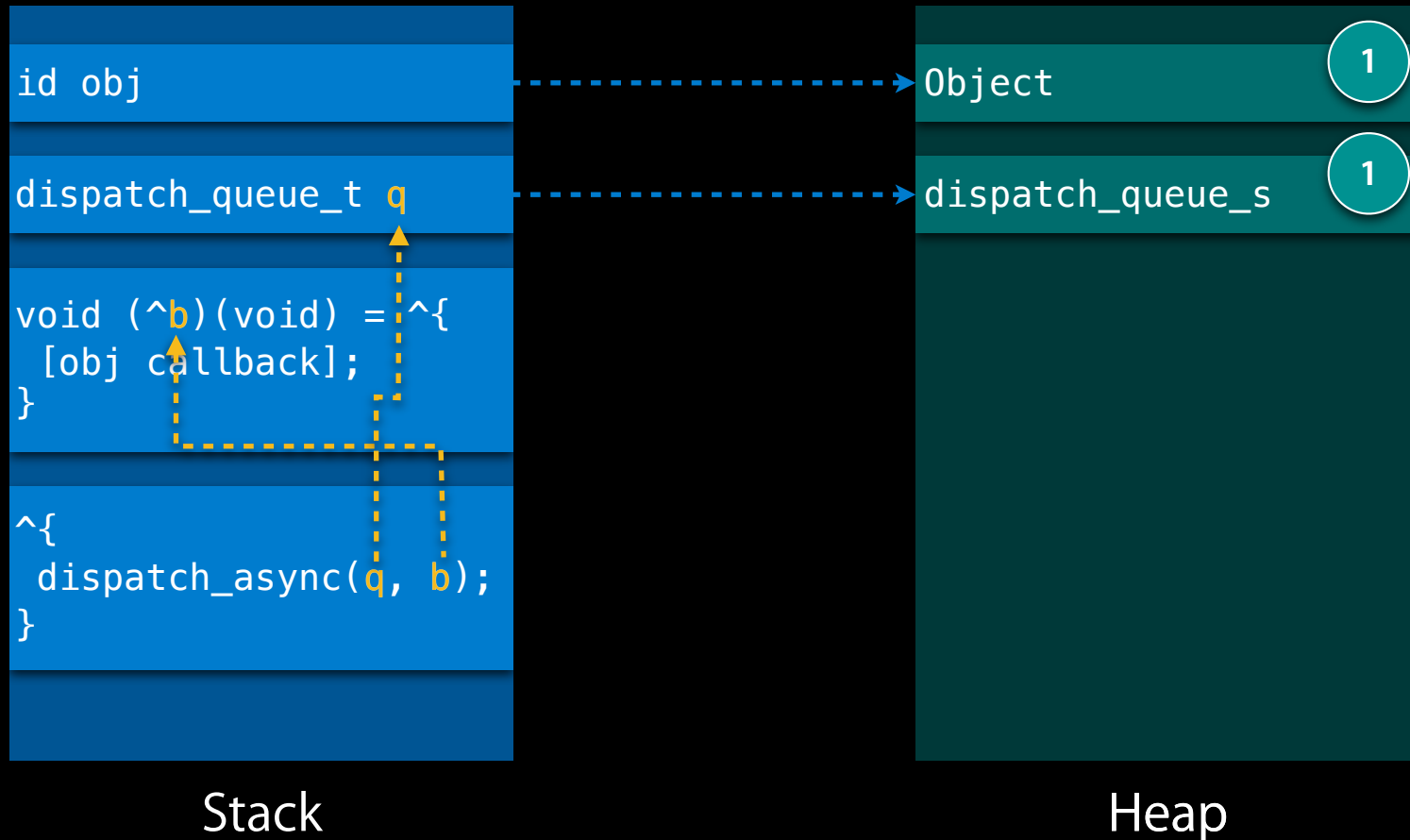
Object Lifetime



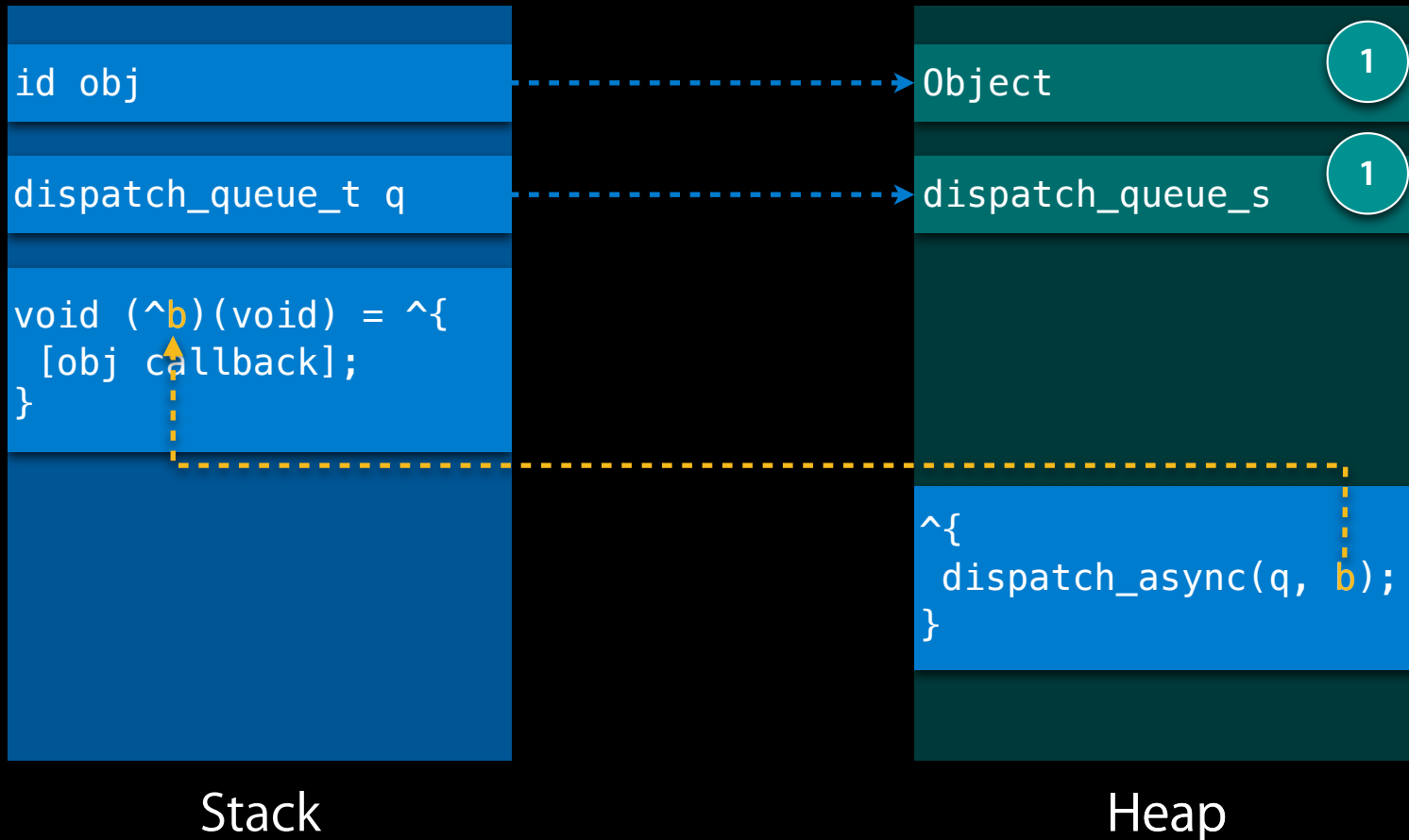
Object Lifetime



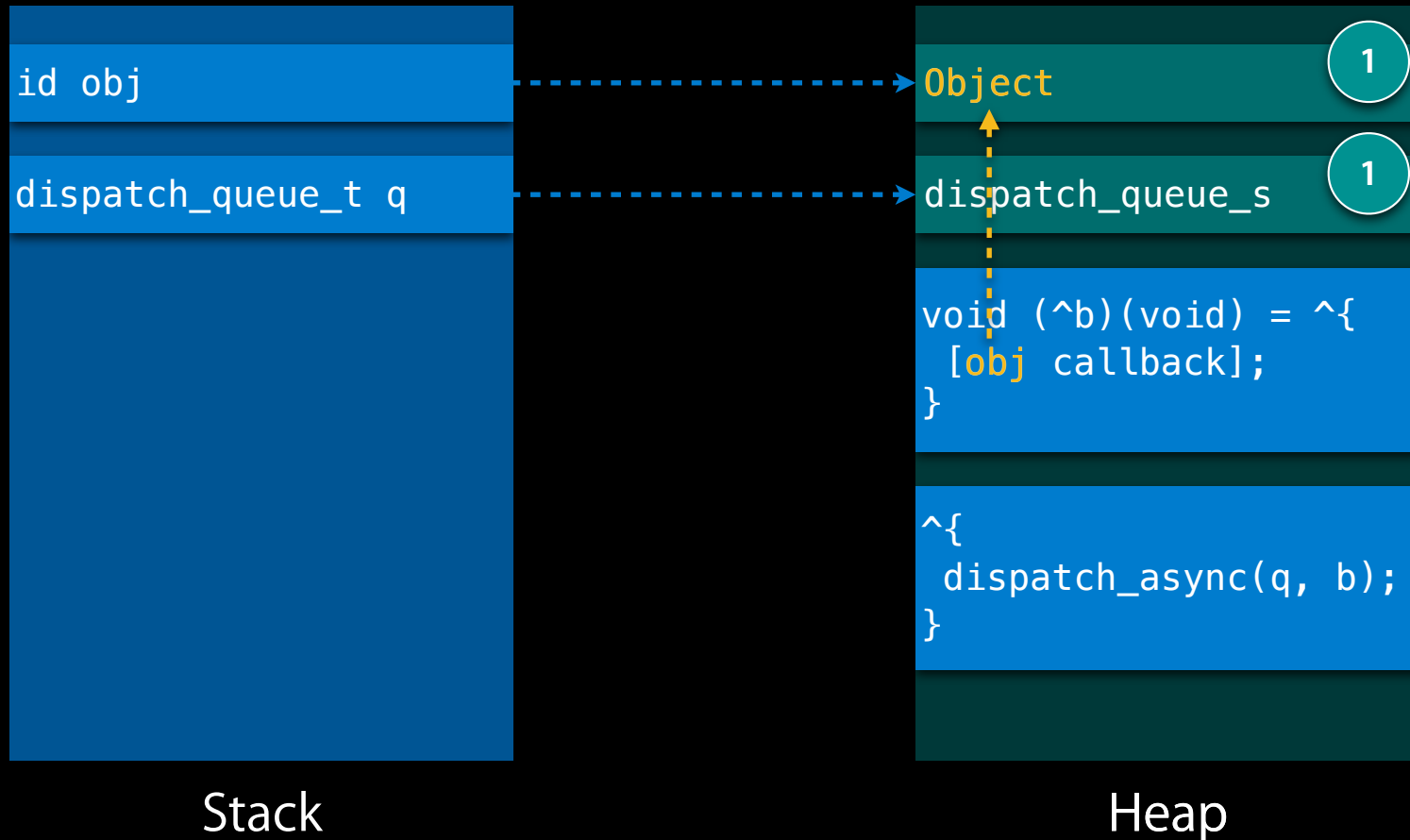
Object Lifetime



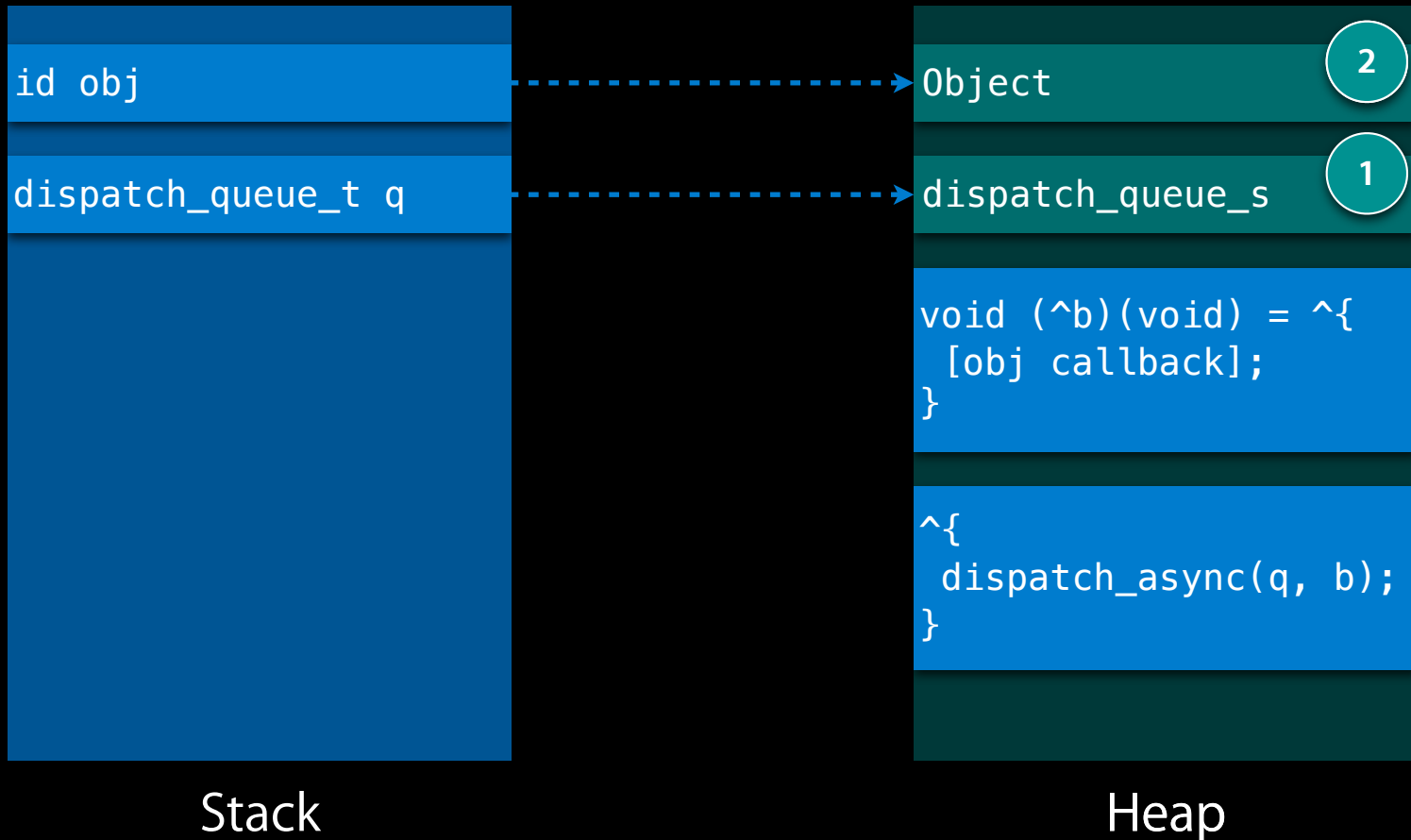
Object Lifetime



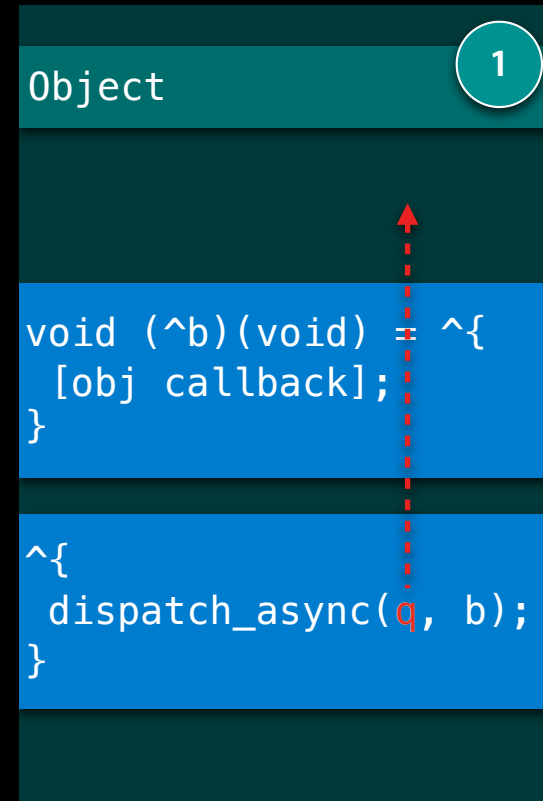
Object Lifetime



Object Lifetime

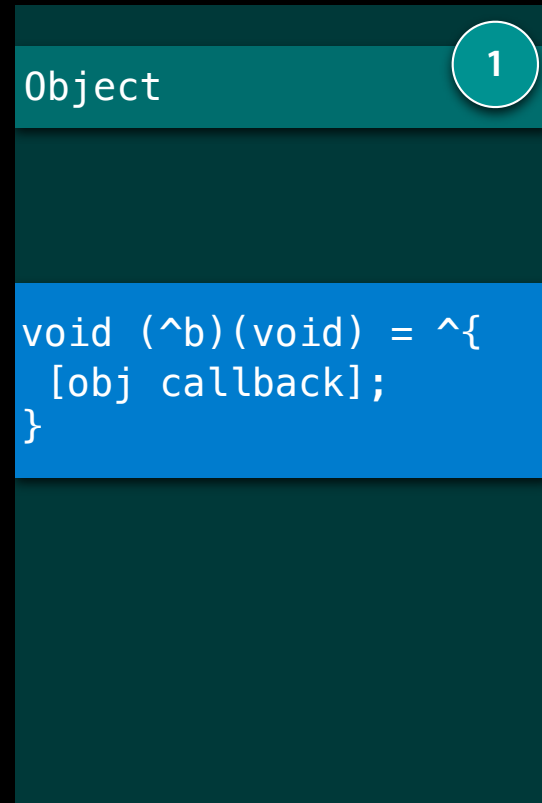


Object Lifetime



Heap

Object Lifetime



Heap

Object Lifetime



```
- (void)performAsyncWorkWithCallback:(id)obj onQueue:(dispatch_queue_t)q {  
    dispatch_async(self.queue, ^{  
        [self performWork];  
  
        dispatch_async(q, ^{  
            [obj callback];  
        });  
    });  
}
```

Object Lifetime



```
- (void)performAsyncWorkWithCallback:(id)obj onQueue:(dispatch_queue_t)q {
    dispatch_retain(q);
    dispatch_async(self.queue, ^{
        [self performWork];

        dispatch_async(q, ^{
            [obj callback];
        });
        dispatch_release(q);
    });
}
```

Objective-C

GCD and XPC objects are now Objective-C objects!

Objective-C !

```
- (void)performAsyncWorkWithCallback:(id)obj onQueue:(dispatch_queue_t)q {
    dispatch_retain(q);
    dispatch_async(self.queue, ^{
        [self performWork];

        dispatch_async(q, ^{
            [obj callback];
        });
        dispatch_release(q);
    });
}
```


Objective-C !



```
- (void)performAsyncWorkWithCallback:(id)obj onQueue:(dispatch_queue_t)q {
    dispatch_retain(q);
    dispatch_async(self.queue, ^{
        [self performWork];

        dispatch_async(q, ^{
            [obj callback];
        });
        dispatch_release(q);
    });
}
```

Objective-C !



```
- (void)performAsyncWorkWithCallback:(id)obj onQueue:(dispatch_queue_t)q {  
    dispatch_async(self.queue, ^{  
        [self performWork];  
  
        dispatch_async(q, ^{  
            [obj callback];  
        });  
    });  
}
```

Benefits



- Automatically retained/released by blocks
- @property(retain)
- Membership in Foundation collections
- Static analyzer
- Instruments and debugger support

Manual Reference Counting

```
- (void)sendData:(NSData *)data toConnection:(xpc_connection_t)connection {  
  
    dispatch_data_t d = dispatch_data_create([data bytes], [data length],  
                                             NULL, ^{ (void)data; });  
    xpc_object_t obj = xpc_data_create_with_dispatch_data(d);  
    xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);  
    xpc_dictionary_set_value(message, "data", obj);  
  
    xpc_connection_send_message(connection, message);  
  
    xpc_release(message);  
    xpc_release(obj);  
    dispatch_release(d);  
}
```

Automatic Reference Counting

```
- (void)sendData:(NSData *)data toConnection:(xpc_connection_t)connection {  
  
    dispatch_data_t d = dispatch_data_create([data bytes], [data length],  
                                             NULL, ^{ (void)data; });  
    xpc_object_t obj = xpc_data_create_with_dispatch_data(d);  
    xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);  
    xpc_dictionary_set_value(message, "data", obj);  
  
    xpc_connection_send_message(connection, message);  
  
    xpc_release(message);  
    xpc_release(obj);  
    dispatch_release(d);  
}
```

Automatic Reference Counting



```
- (void)sendData:(NSData *)data toConnection:(xpc_connection_t)connection {  
  
    dispatch_data_t d = dispatch_data_create([data bytes], [data length],  
                                             NULL, ^{ (void)data; });  
    xpc_object_t obj = xpc_data_create_with_dispatch_data(d);  
    xpc_object_t message = xpc_dictionary_create(NULL, NULL, 0);  
    xpc_dictionary_set_value(message, "data", obj);  
  
    xpc_connection_send_message(connection, message);  
  
}
```

Migrating to ARC



- GCD and XPC objects automatically managed
- Use “Convert to Objective-C ARC” in Xcode
- Or remove GCD and XPC retain/release calls

```
dispatch_retain/dispatch_release  
xpc_retain/xpc_release
```

Requirements



- Objective-C/Objective-C++
- Minimum Deployment Target
 - iOS 6 or Mac OS 10.8
- Opt-out
 - `-DOS_OBJECT_USE_OBJC=0`

Objective-C and ARC

Special considerations

Special Considerations

Things migration to ARC doesn't handle

- Blocks and retain cycles
- Interior pointers

Blocks and Retain Cycles

```
@interface MyClass ()  
@property(readonly) int val;  
@property(strong) dispatch_block_t work;  
@end
```

Blocks and Retain Cycles

```
@interface MyClass ()  
@property(readonly) int val;  
@property(strong) dispatch_block_t work;  
@end
```

MyClass *self

int val

1

Blocks and Retain Cycles

```
@interface MyClass ()
@property(readonly) int val;
@property(strong) dispatch_block_t work;
@end

- (void)setup {
    self.work = ^{
        NSLog(@"%d", val);
    };
}
```

1
MyClass *self

int val

Blocks and Retain Cycles

```
@interface MyClass ()
@property(readonly) int val;
@property(strong) dispatch_block_t work;
@end

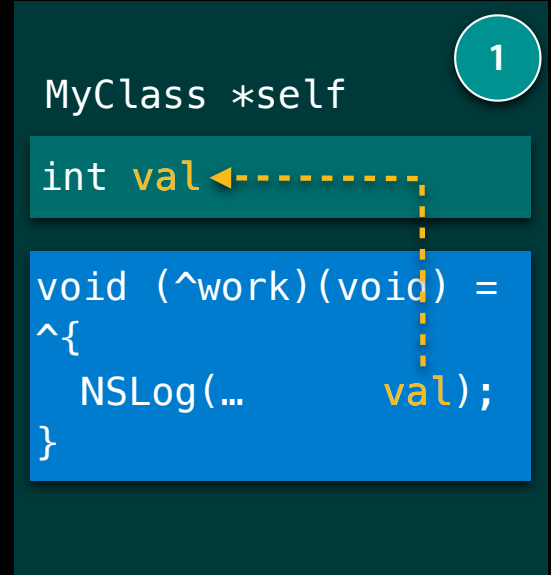
- (void)setup {
    self.work = ^{
        NSLog(@"%d", val);
    };
}
```

MyClass *self

int val

```
void (^work)(void) =
^{
    NSLog(... val);
}
```

1



Blocks and Retain Cycles

self capture

```
@interface MyClass ()
@property(readonly) int val;
@property(strong) dispatch_block_t work;
@end

- (void)setup {
    self.work = ^{
        NSLog(@"%d", self->val);
    };
}
```



1
MyClass *self

int val

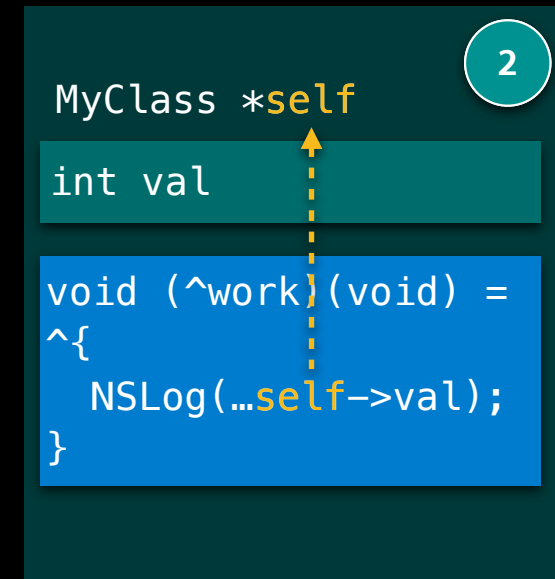
```
void (^work)(void) =
^{
    NSLog(...self->val);
}
```

Blocks and Retain Cycles

self capture

```
@interface MyClass ()
@property(readonly) int val;
@property(strong) dispatch_block_t work;
@end

- (void)setup {
    self.work = ^{
        NSLog(@"%d", self->val);
    };
}
```



Blocks and Retain Cycles

Ways to break them

- Scoping
- Programmatically
- Attributes

Scoping

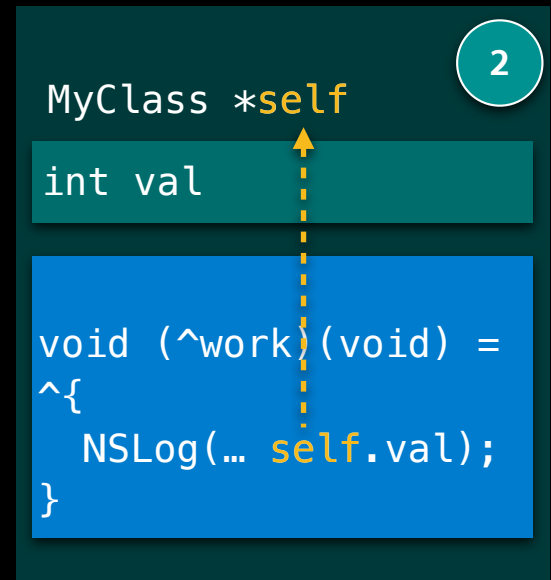
```
@interface MyClass ()
@property(readonly) int val;
@property(strong) dispatch_block_t work;
@end

- (void)setup {
    self.work = ^{
        NSLog(@"%d", self.val);
    };
}
```

Scoping

```
@interface MyClass ()
@property(readonly) int val;
@property(strong) dispatch_block_t work;
@end

- (void)setup {
    self.work = ^{
        NSLog(@"%d", self.val);
    };
}
```



Scoping

Avoid capturing self

```
@interface MyClass ()  
@property(readonly) int val;  
@property(strong) dispatch_block_t work;  
@end
```

```
- (void)setup {  
    int local = self.val;  
    self.work = ^{  
        NSLog(@"%d", local);  
    };  
}
```

1

```
MyClass *self
```

```
int val
```

```
int local;  
void (^work)(void) =  
^{  
    NSLog(... local);  
}
```

Scoping

Avoid capturing self

```
@interface MyClass ()  
@property(readonly) int val;  
@property(strong) dispatch_block_t work;  
@end
```

```
- (void)setup {  
    int local = self.val;  
    self.work = ^{  
        NSLog(@"%d", local);  
    };  
}
```

1
MyClass *self

int val

```
int local; ←-----  
void (^work)(void) =  
^{  
    NSLog(... local);  
}
```

Programmatically

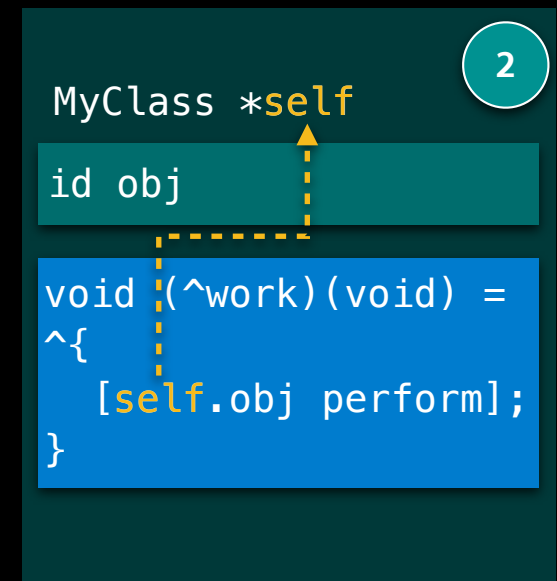
```
@interface MyClass ()  
@property(strong) id obj;  
@property(strong) dispatch_block_t work;  
@end
```

```
- (void)setup {  
    self.work = ^{  
        [self.obj perform];  
    };  
}
```

Programmatically

```
@interface MyClass ()
@property(strong) id obj;
@property(strong) dispatch_block_t work;
@end

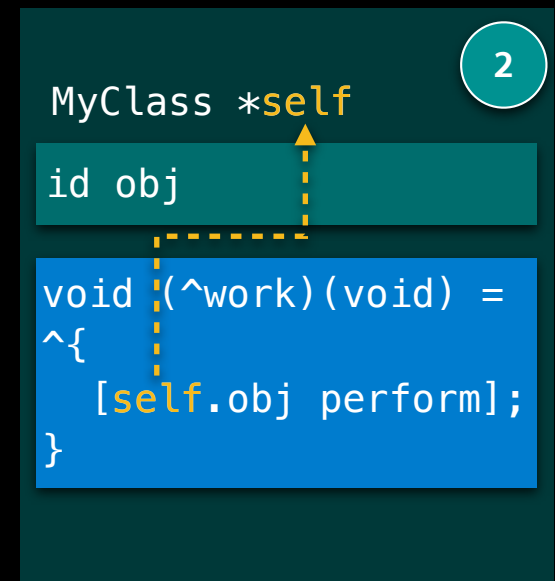
- (void)setup {
    self.work = ^{
        [self.obj perform];
    };
}
```



Programmatically nil the block property

```
@interface MyClass ()  
@property(strong) id obj;  
@property(strong) dispatch_block_t work;  
@end
```

```
- (void)setup {  
    self.work = ^{  
        [self.obj perform];  
    };  
}  
- (void)cancel {  
    self.work = nil;  
}
```



Programmatically nil the block property

```
@interface MyClass ()  
@property(strong) id obj;  
@property(strong) dispatch_block_t work;  
@end
```

```
- (void)setup {  
    self.work = ^{  
        [self.obj perform];  
    };  
}  
- (void)cancel {  
    self.work = nil;  
}
```

1
MyClass *self

id obj

void (^work)(void) =
 nil;

Programmatically

API patterns

```
dispatch_source_t source = dispatch_source_create(...);
dispatch_source_set_event_handler(source, ^{
    NSLog(@"%d", dispatch_source_get_data(source));
});
```

...

```
dispatch_release(source);
```

```
xpc_connection_t connection = xpc_connection_create(...);
xpc_connection_set_event_handler(connection, ^(xpc_object_t event){
    NSLog(@"%p", xpc_connection_get_context(connection));
});
```

...

```
xpc_release(connection);
```

Programmatically

API patterns



```
dispatch_source_t source = dispatch_source_create(...);
dispatch_source_set_event_handler(source, ^{
    NSLog(@"%d", dispatch_source_get_data(source));
});
...
dispatch_source_cancel(source);
dispatch_release(source);
```

```
xpc_connection_t connection = xpc_connection_create(...);
xpc_connection_set_event_handler(connection, ^(xpc_object_t event){
    NSLog(@"%p", xpc_connection_get_context(connection));
});
...
xpc_connection_cancel(connection);
xpc_release(connection);
```

Programmatically

API patterns



```
dispatch_source_t source = dispatch_source_create(...);
dispatch_source_set_event_handler(source, ^{
    NSLog(@"%d", dispatch_source_get_data(source));
});
...
dispatch_source_cancel(source);
```

```
xpc_connection_t connection = xpc_connection_create(...);
xpc_connection_set_event_handler(connection, ^(xpc_object_t event){
    NSLog(@"%p", xpc_connection_get_context(connection));
});
...
xpc_connection_cancel(connection);
```

Attributes

```
@interface MyClass ()  
@property(strong) id obj;  
@property(strong) dispatch_block_t work;  
@end
```

```
- (void)setup {  
    self.work = ^{  
        [ self.obj perform];  
    };  
}
```

Attributes

Use `__weak`

```
@interface MyClass ()  
@property(strong) id obj;  
@property(strong) dispatch_block_t work;  
@end
```

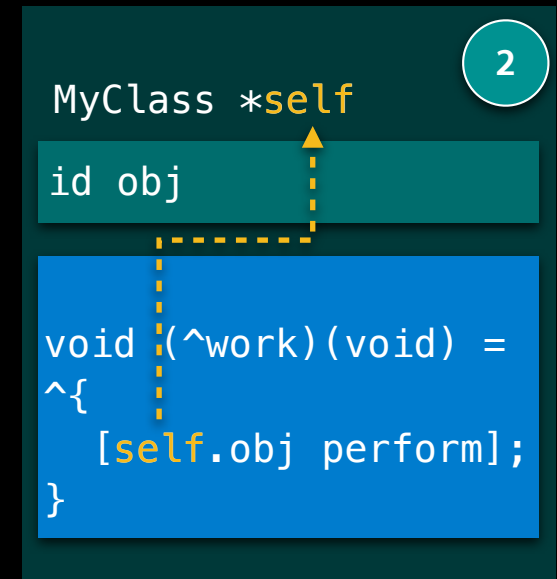
```
- (void)setup {  
    __weak MyClass *weakSelf = self;  
    self.work = ^{  
        __strong MyClass *strongSelf = weakSelf;  
        if (strongSelf) {  
            [strongSelf.obj perform];  
        }  
    };  
}
```

Attributes

Use `__weak`

```
@interface MyClass ()
@property(strong) id obj;
@property(strong) dispatch_block_t work;
@end

- (void)setup {
    __weak MyClass *weakSelf = self;
    self.work = ^{
        __strong MyClass *strongSelf = weakSelf;
        if (strongSelf) {
            [strongSelf.obj perform];
        }
    };
}
```



Attributes

Use `__weak`

```
@interface MyClass ()  
@property(strong) id obj;  
@property(strong) dispatch_block_t work;  
@end
```

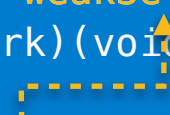
```
- (void)setup {  
    __weak MyClass *weakSelf = self;  
    self.work = ^{  
        __strong MyClass *strongSelf = weakSelf;  
        if (strongSelf) {  
            [strongSelf.obj perform];  
        }  
    };  
}
```

1

```
MyClass *self
```

```
id obj
```

```
__weak id weakSelf;  
void (^work)(void) =  
^{  
    ... weakSelf ...  
}
```



Attributes

Use `__weak`

```
@interface MyClass ()  
@property(strong) id obj;  
@property(strong) dispatch_block_t work;  
@end
```

```
- (void)setup {  
    __weak MyClass *weakSelf = self;  
    self.work = ^{  
        __strong MyClass *strongSelf = weakSelf;  
        if (strongSelf) {  
            [strongSelf.obj perform];  
        }  
    };  
}
```

1
MyClass *self

id obj

```
__weak id weakSelf;  
void (^work)(void) =  
^{  
    ... weakSelf ...  
}
```

Interior Pointers

Pointer lifetime tied to container object

Interior Pointers

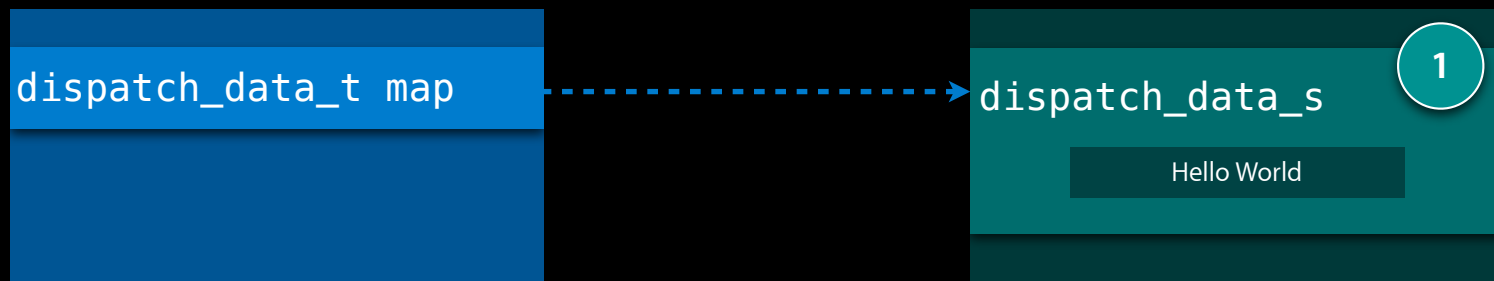
Pointer lifetime tied to container object

```
- (void)logWithData:(dispatch_data_t)data {  
    void *buf;  
    dispatch_data_t map;  
  
    map = dispatch_data_create_map(data, &buf, NULL);  
  
    NSLog(@"%@", [NSString stringWithUTF8String:buf]);  
  
    dispatch_release(map);  
}
```

Interior Pointers

Pointer lifetime tied to container object

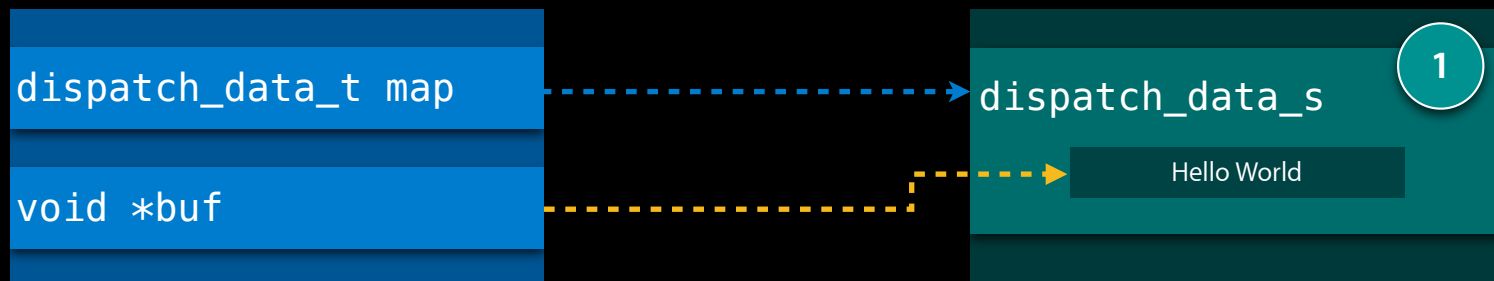
```
- (void)logWithData:(dispatch_data_t)data {  
    void *buf;  
    dispatch_data_t map;  
  
    map = dispatch_data_create_map(data, &buf, NULL);  
  
    NSLog(@"%@", [NSString stringWithUTF8String:buf]);  
  
    dispatch_release(map);  
}
```



Interior Pointers

Pointer lifetime tied to container object

```
- (void)logWithData:(dispatch_data_t)data {  
    void *buf;  
    dispatch_data_t map;  
  
    map = dispatch_data_create_map(data, &buf, NULL);  
  
    NSLog(@"%@", [NSString stringWithUTF8String:buf]);  
  
    dispatch_release(map);  
}
```



Interior Pointers

Pointer lifetime tied to container object

```
- (void)logWithData:(dispatch_data_t)data {
    void *buf;
    dispatch_data_t map;

    map = dispatch_data_create_map(data, &buf, NULL);

    NSLog(@"%@", [NSString stringWithUTF8String:buf]);

    dispatch_release(map);
}
```

Interior Pointers and ARC

Pointer lifetime tied to container object

```
- (void)logWithData:(dispatch_data_t)data {  
    void *buf;  
    dispatch_data_t map;  
  
    map = dispatch_data_create_map(data, &buf, NULL);  
  
    NSLog(@"%@", [NSString stringWithUTF8String:buf]);  
  
    dispatch_release(map);  
}
```

Interior Pointers and ARC

Pointer lifetime tied to container object



```
- (void)logWithData:(dispatch_data_t)data {
    void *buf;
    dispatch_data_t map;

    map = dispatch_data_create_map(data, &buf, NULL);

    NSLog(@"%@", [NSString stringWithUTF8String:buf]);
}
```


Interior Pointers and ARC

Pointer lifetime tied to container object



```
- (void)logWithData:(dispatch_data_t)data {
    void *buf;
    dispatch_data_t map;

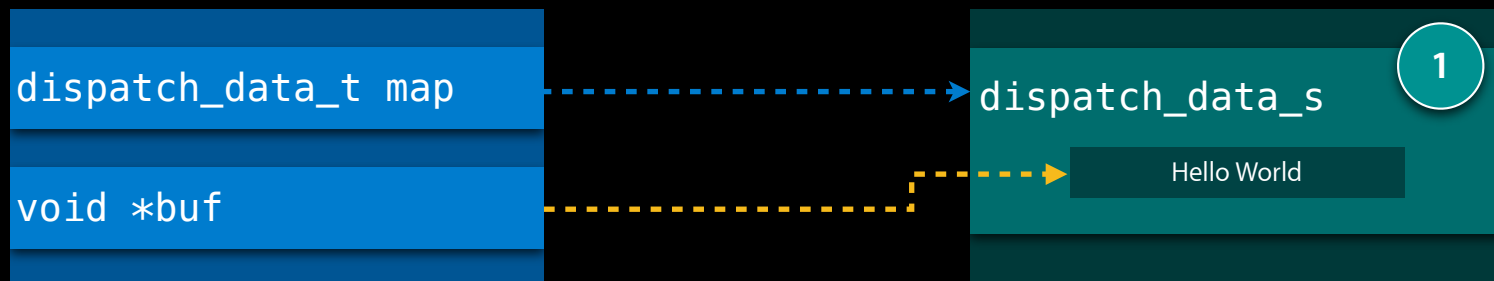
    map = dispatch_data_create_map(data, &buf, NULL);
    objc_release(map);
    NSLog(@"%@", [NSString stringWithUTF8String:buf]);
}
```

Interior Pointers and ARC

Pointer lifetime tied to container object



```
- (void)logWithData:(dispatch_data_t)data {  
    void *buf;  
    dispatch_data_t map;  
  
    map = dispatch_data_create_map(data, &buf, NULL);  
    objc_release(map);  
    NSLog(@"%@", [NSString stringWithUTF8String:buf]);  
  
}
```

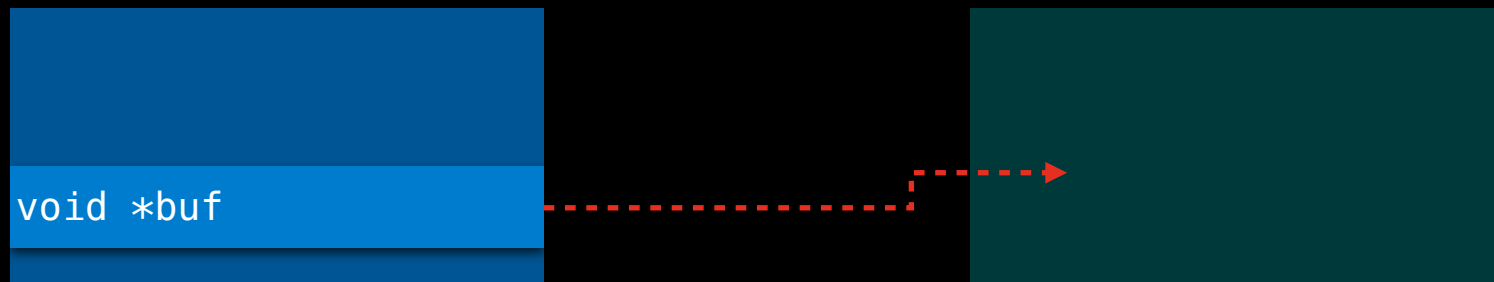


Interior Pointers and ARC

Pointer lifetime tied to container object



```
- (void)logWithData:(dispatch_data_t)data {  
    void *buf;  
    dispatch_data_t map;  
  
    map = dispatch_data_create_map(data, &buf, NULL);  
    objc_release(map);  
    NSLog(@"%@", [NSString stringWithUTF8String:buf]);  
  
}
```



Interior Pointers and ARC

Pointer lifetime tied to container object



```
- (void)logWithData:(dispatch_data_t)data {
    void *buf;
    dispatch_data_t map;

    map = dispatch_data_create_map(data, &buf, NULL);

    NSLog(@"%@", [NSString stringWithUTF8String:buf]);
}
```

Interior Pointers and ARC

Pointer lifetime tied to container object



```
- (void)logWithData:(dispatch_data_t)data {
    void *buf;
    dispatch_data_t map __attribute__((objc_precise_lifetime));

    map = dispatch_data_create_map(data, &buf, NULL);

    NSLog(@"%@ ", [NSString stringWithUTF8String:buf]);
}
```

Interior Pointers and ARC

Pointer lifetime tied to container object



```
- (void)logWithData:(dispatch_data_t)data {
    void *buf;
    dispatch_data_t map __attribute__((objc_precise_lifetime));

    map = dispatch_data_create_map(data, &buf, NULL);

    NSLog(@"%@ ", [NSString stringWithUTF8String:buf]);

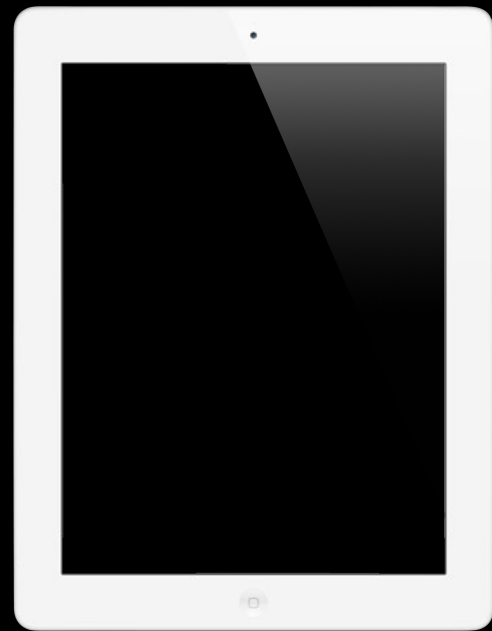
    objc_release(map);
}
```

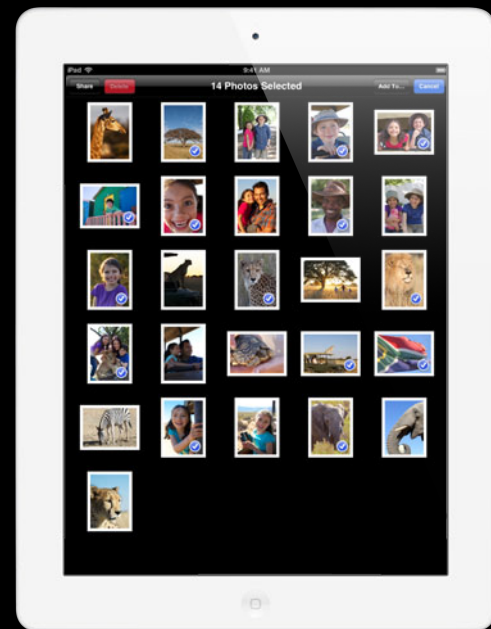
Asynchronous Design Patterns

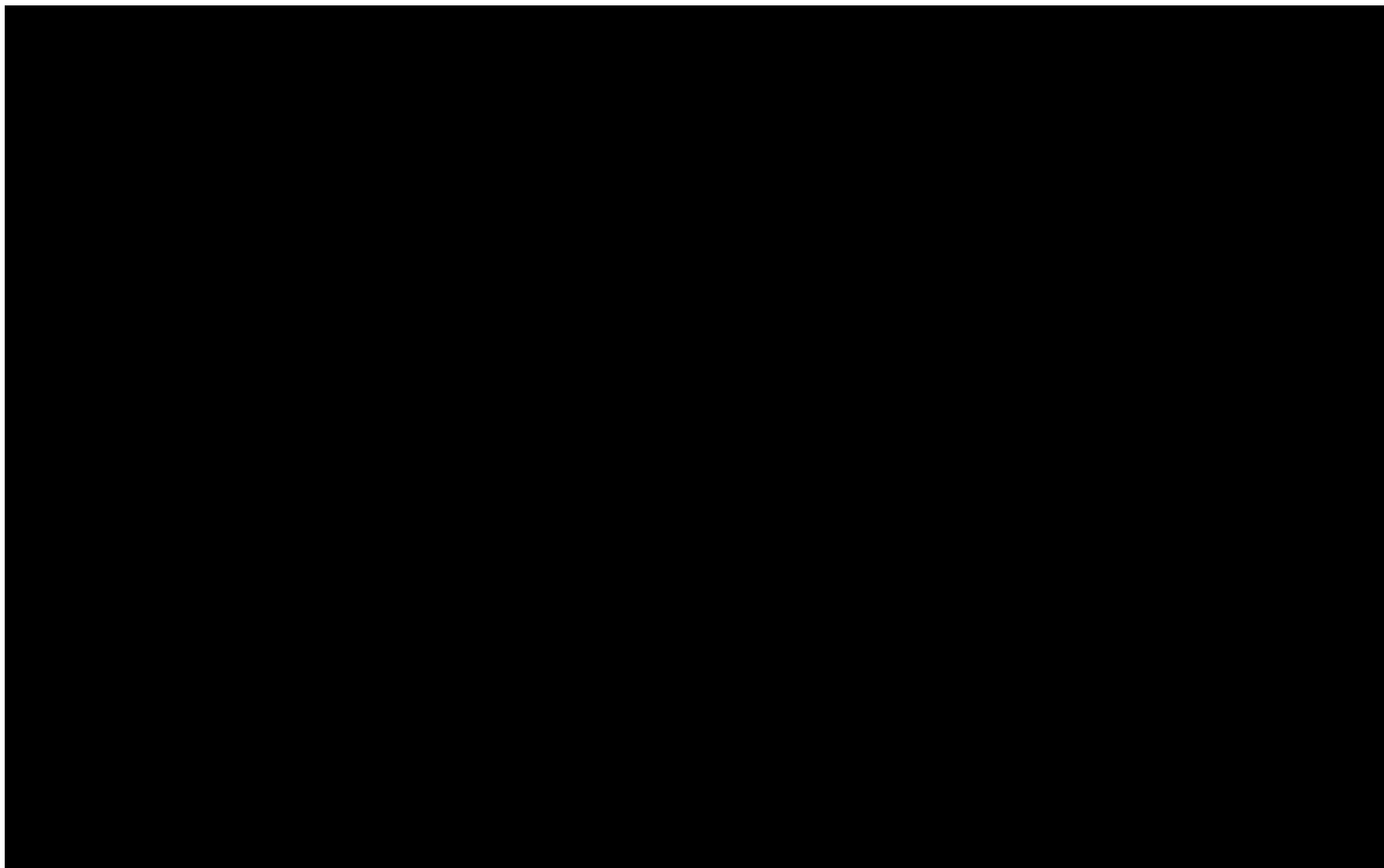
Daniel Steffen
Core OS

Overview

- Make code more asynchronous
- Avoid common trouble spots
- Apply patterns to Apple APIs or your own





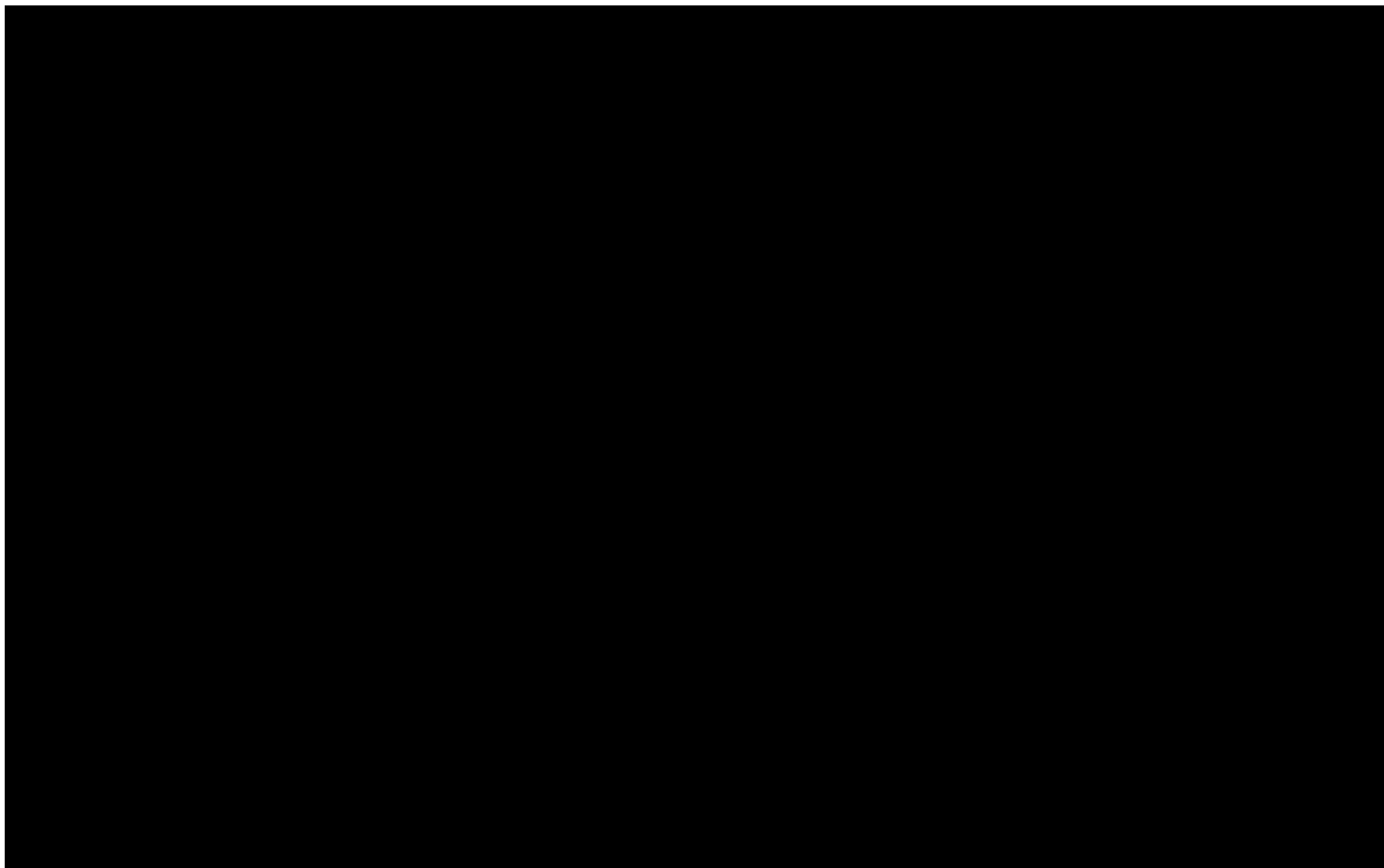


1

Don't Block the Main Thread

Don't Block the Main Thread

- Main thread should only handle user interaction and UI
- Keep UI responsive at all times
- Run CPU intensive or blocking code elsewhere



2

Run in the Background

Run in the Background

```
// Main Thread
```

```
[self renderThumbnails];
```

```
[self.thumbnailView setNeedsDisplay:YES];
```


Run in the Background

```
// Main Thread
dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

[self renderThumbnails];

[self.thumbnailView setNeedsDisplay:YES];
```

Run in the Background

```
// Main Thread
dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

    ^{

        [self renderThumbnails];

        [self.thumbnailView setNeedsDisplay:YES];

    }
```

Run in the Background

```
// Main Thread
dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{

    [self renderThumbnails];

    [self.thumbnailView setNeedsDisplay:YES];

});
```

Run in the Background

```
// Main Thread
dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{

    [self renderThumbnails];

    ^{
        [self.thumbnailView setNeedsDisplay:YES];
    }

});
```

Run in the Background

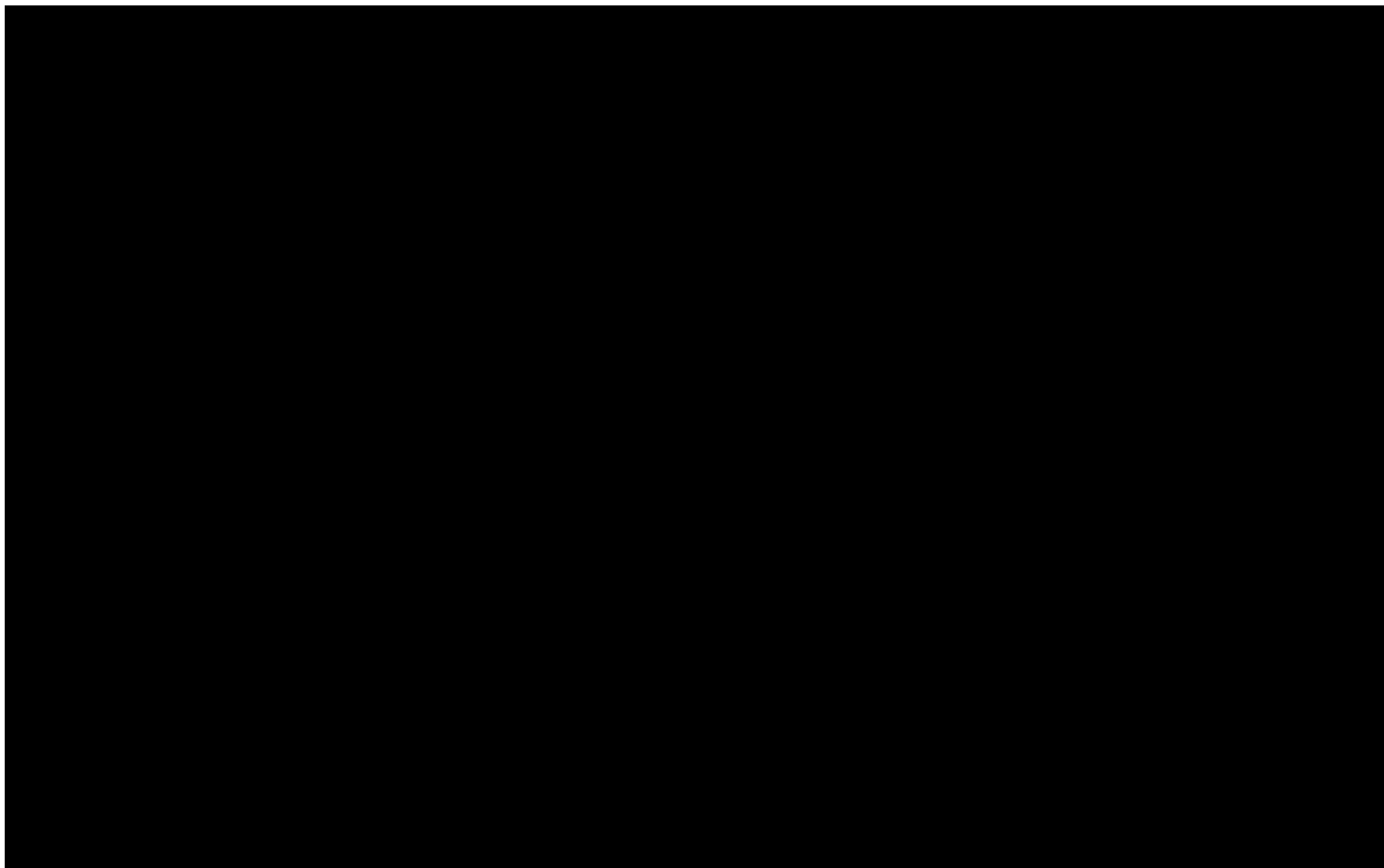
```
// Main Thread
dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{

    [self renderThumbnails];

    dispatch_async(dispatch_get_main_queue(), ^{
        [self.thumbnailView setNeedsDisplay:YES];
    });

});
```



3

Don't Block Many Background Threads

Don't Block Many Background Threads

```
// Main Thread
dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{
    NSData *data = [NSData dataWithContentsOfURL:url];

    dispatch_async(dispatch_get_main_queue(), ^{
        [self.imageStore setImageData:data forURL:url];
    });
});
```

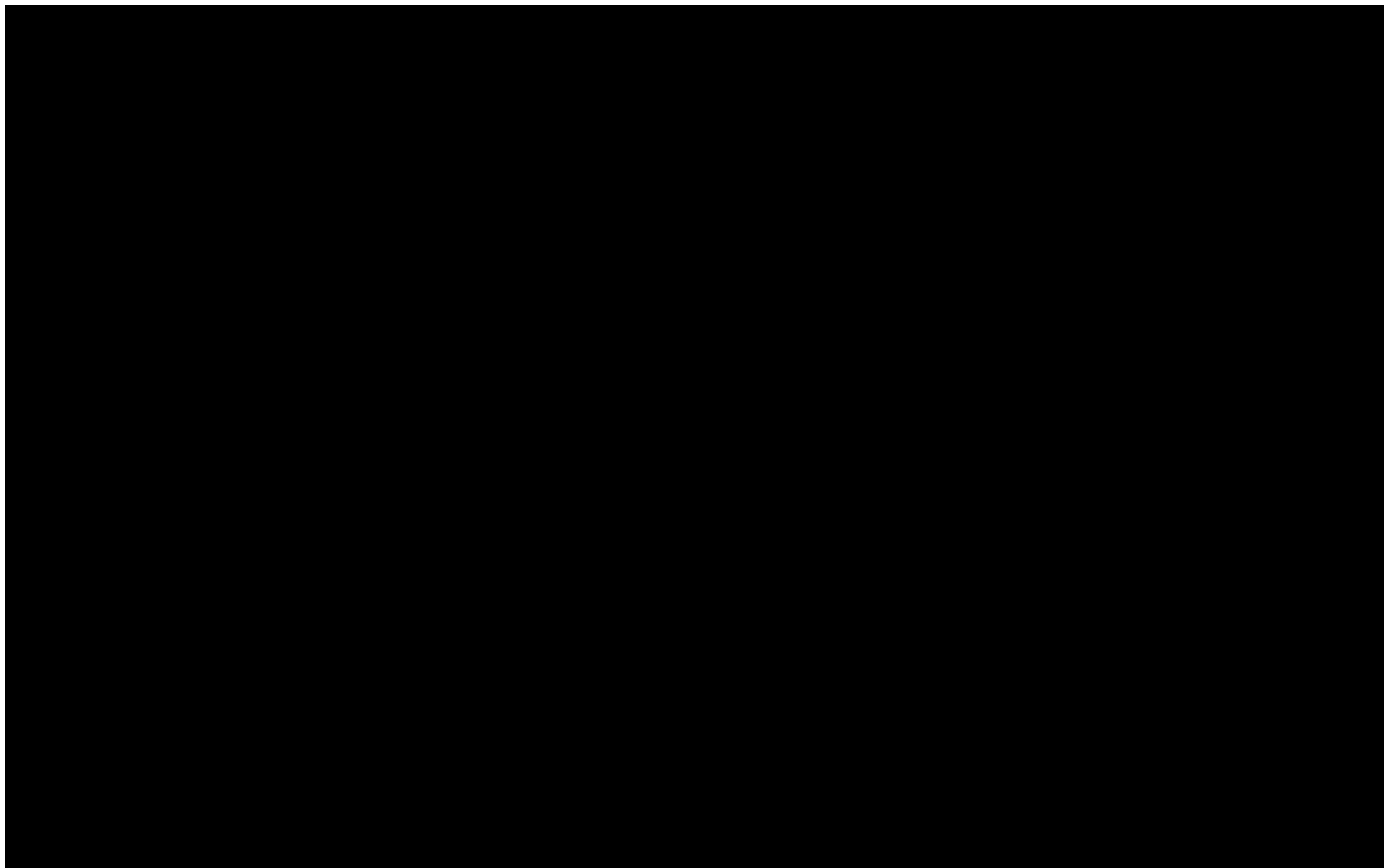

Don't Block Many Background Threads




```
// Main Thread
dispatch_queue_t queue;
queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

for (NSURL *url in [self.imageStore URLs]) {
    dispatch_async(queue, ^{
        NSData *data = [NSData dataWithContentsOfURL:url];

        dispatch_async(dispatch_get_main_queue(), ^{
            [self.imageStore setImageData:data forURL:url];
        });
    });
}
```

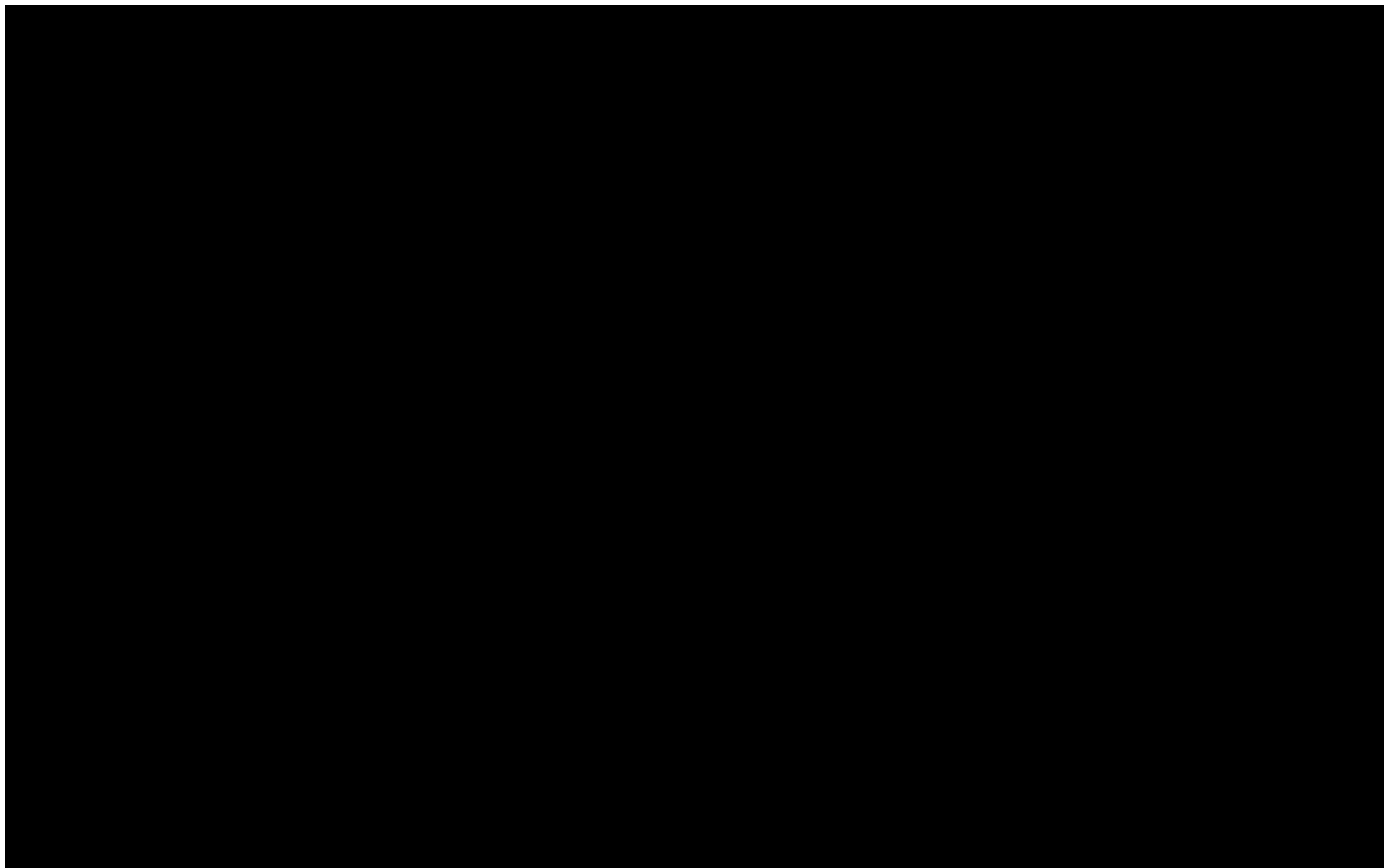


Main Thread

A square icon with rounded corners, featuring a grid pattern and a large white letter 'Q' in the center.

`^{ [NSData ...] }`

Automatic Thread





Don't Block Many Background Threads

Dispatch I/O



```
// Main Thread
for (NSURL *url in [self.imageStore URLs]) {

}
}
```

Don't Block Many Background Threads

Dispatch I/O



```
// Main Thread
for (NSURL *url in [self.imageStore URLs]) {
    dispatch_io_t io = dispatch_io_create_with_path(DISPATCH_IO_RANDOM,
        [[url path] fileSystemRepresentation], 0_RDONLY, 0, NULL, NULL);
    dispatch_io_set_low_water(io, SIZE_MAX);

    dispatch_io_read(io, 0, SIZE_MAX, dispatch_get_main_queue(),
        ^(bool done, dispatch_data_t data, int error){
            [self.imageStore setImageData:data forURL:url];
        });
}
```

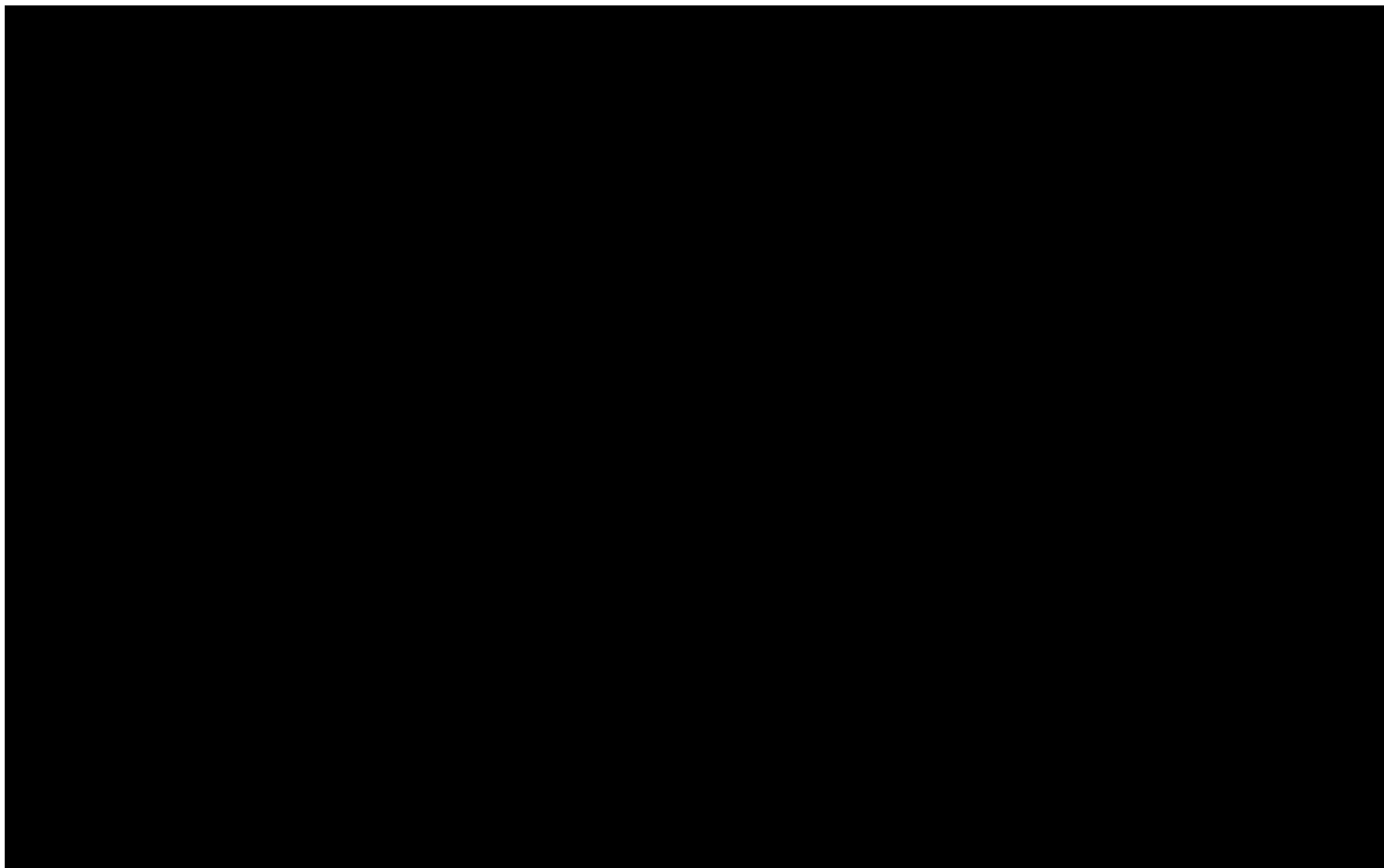
Don't Block Many Background Threads



Dispatch I/O

```
// Main Thread
for (NSURL *url in [self.imageStore URLs]) {
    dispatch_io_t io = dispatch_io_create_with_path(DISPATCH_IO_RANDOM,
        [[url path] fileSystemRepresentation], 0_RDONLY, 0, NULL, NULL);
    dispatch_io_set_low_water(io, SIZE_MAX);

    dispatch_io_read(io, 0, SIZE_MAX, dispatch_get_main_queue(),
        ^(bool done, dispatch_data_t data, int error){
            [self.imageStore setImageData:data forURL:url];
        });
}
```

4

Integrate with the *Main Runloop*

Integrate with the Main Runloop

- `dispatch_get_main_queue()`

Integrate with the Main Runloop

- `dispatch_get_main_queue()`
- API with runloop-based callbacks

Integrate with the Main Runloop

- `dispatch_get_main_queue()`
- API with runloop-based callbacks
 - Don't call on automatic worker threads

Integrate with the Main Runloop

- `dispatch_get_main_queue()`
- API with runloop-based callbacks
 - Don't call on automatic worker threads
 - Don't block in main runloop callbacks

Integrate with the Main Runloop

```
- (void)downloadFromRemotePictureViewer:(NSString *)name {
    // Main Thread

    NSNetService *service = [[NSNetService alloc] initWithDomain:@""
        type:@"_pictureviewer._tcp" name:name];
    [service setDelegate:self];
    [service resolveWithTimeout:5.0];

});

- (void)netServiceDidResolveAddress:(NSNetService *)service {

    [self downloadFromRemoteService:service];

}
```

Integrate with the Main Runloop

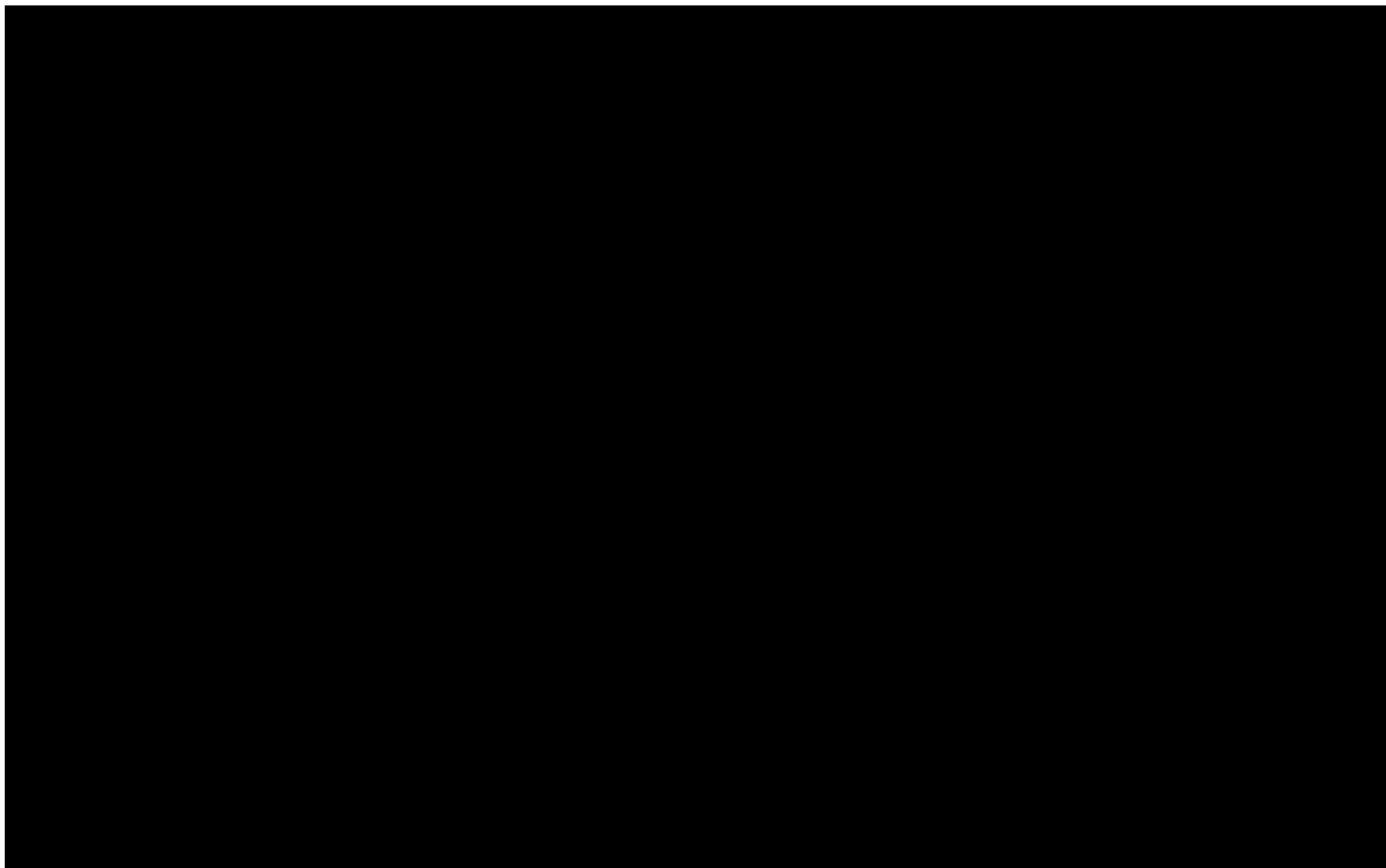


```
- (void)downloadFromRemotePictureViewer:(NSString *)name {
    // Main Thread
    dispatch_async(self.downloadQueue, ^{
        NSNetService *service = [[NSNetService alloc] initWithDomain:@""
            type:@"_pictureviewer._tcp" name:name];
        [service setDelegate:self];
        [service resolveWithTimeout:5.0];
    });
};

- (void)netServiceDidResolveAddress:(NSNetService *)service {

    [self downloadFromRemoteService:service];

}
```



Main Thread

`^ {[resolve...] }`

Main Thread



`^ {[resolve...]}`

Automatic Thread





Main Thread



Integrate with the Main Runloop



```
- (void)downloadFromRemotePictureViewer:(NSString *)name {
    // Main Thread

    NSNetService *service = [[NSNetService alloc] initWithDomain:@""
        type:@"_pictureviewer._tcp" name:name];
    [service setDelegate:self];
    [service resolveWithTimeout:5.0];

});

- (void)netServiceDidResolveAddress:(NSNetService *)service {

    [self downloadFromRemoteService:service];

}
```

Integrate with the Main Runloop

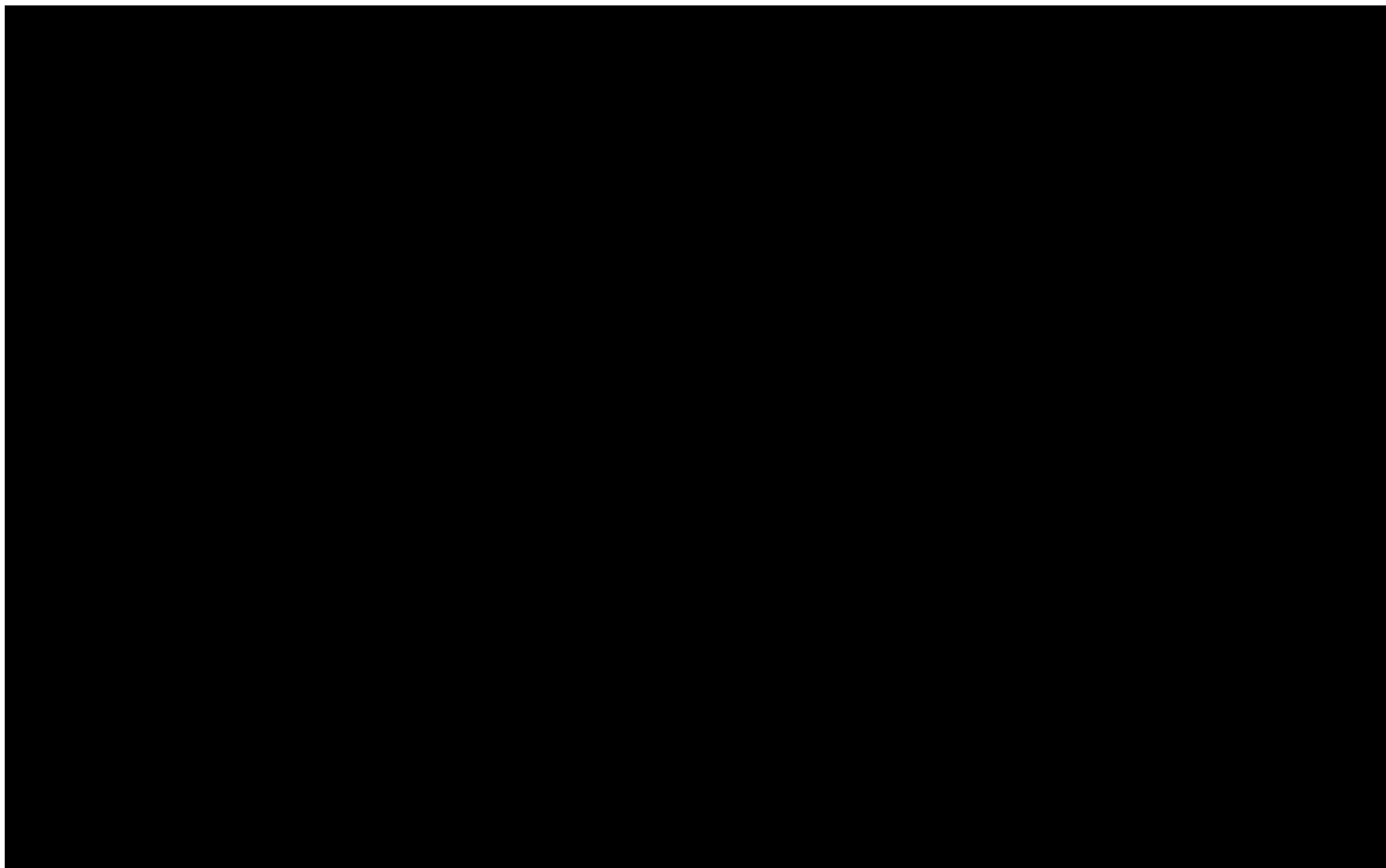


```
- (void)downloadFromRemotePictureViewer:(NSString *)name {
    // Main Thread

    NSNetService *service = [[NSNetService alloc] initWithDomain:@""
        type:@"_pictureviewer._tcp" name:name];
    [service setDelegate:self];
    [service resolveWithTimeout:5.0];

});

- (void)netServiceDidResolveAddress:(NSNetService *)service {
    dispatch_async(self.downloadQueue, ^{
        [self downloadFromRemoteService:service];
    });
}
```



5

One Queue per Subsystem

One Queue per Subsystem

- Subdivide app into independent subsystems
- Control access to subsystems with serial dispatch queues
- Main queue is access queue for UI subsystem





One Queue per Subsystem

```
- (void)netServiceDidResolveAddress:(NSNetService *)service {
```

```
}
```

One Queue per Subsystem

```
- (void)netServiceDidResolveAddress:(NSNetService *)service {  
    dispatch_async(self.downloadQueue, ^{  
        NSData data = [self downloadFromRemoteService:service];  
  
    });  
}
```

One Queue per Subsystem

```
- (void)netServiceDidResolveAddress:(NSNetService *)service {
    dispatch_async(self.downloadQueue, ^{
        NSData data = [self downloadFromRemoteService:service];

        dispatch_async(self.storeQueue, ^{
            int img = [self.imageStore addImage:data];
        });
    });
}
```

One Queue per Subsystem

```
- (void)netServiceDidResolveAddress:(NSNetService *)service {
    dispatch_async(self.downloadQueue, ^{
        NSData data = [self downloadFromRemoteService:service];

        dispatch_async(self.storeQueue, ^{
            int img = [self.imageStore addImage:data];

            dispatch_async(self.renderQueue, ^{
                [self renderThumbnail:img];
            });
        });
    });
}
```

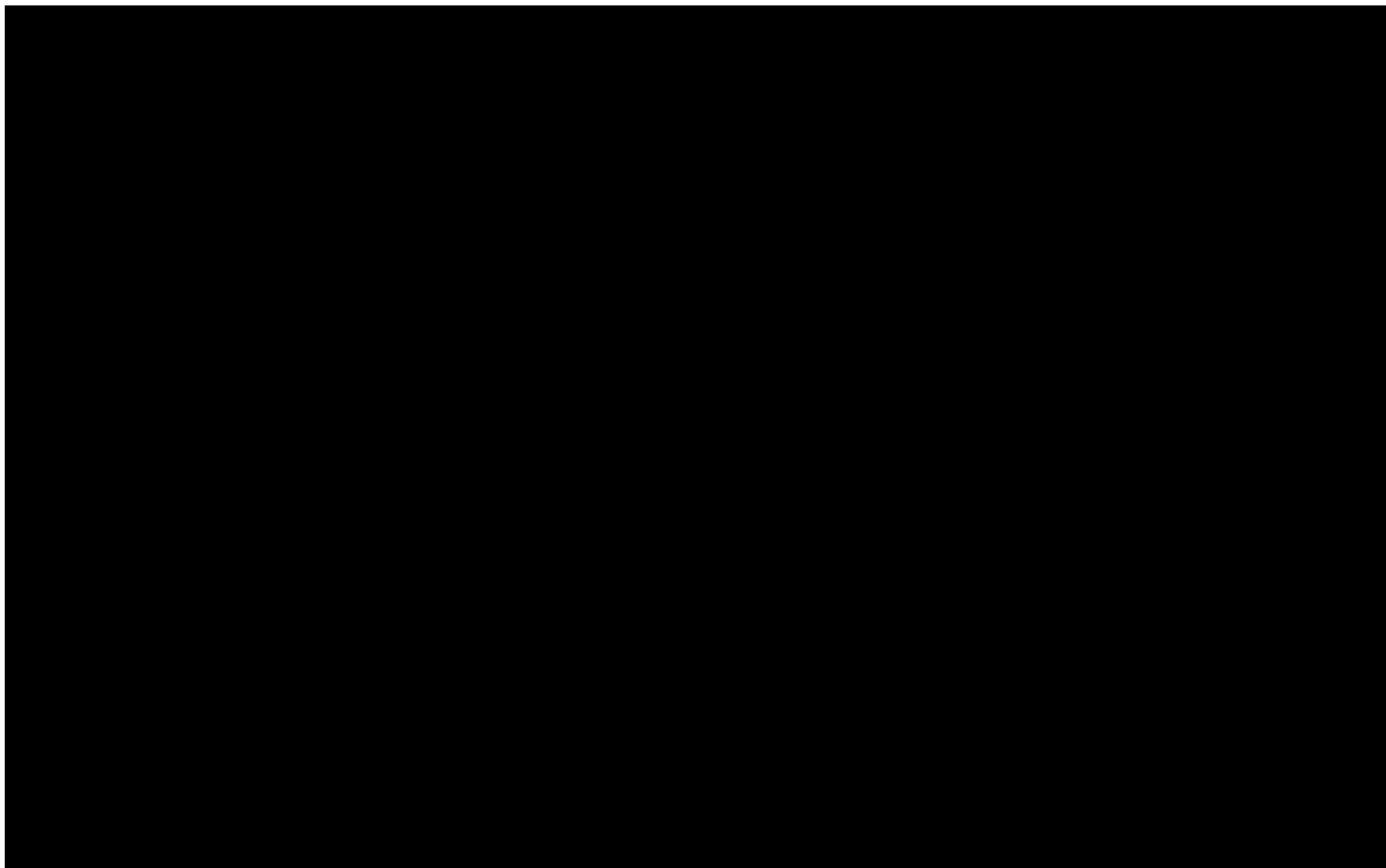
One Queue per Subsystem

```
- (void)netServiceDidResolveAddress:(NSNetService *)service {
    dispatch_async(self.downloadQueue, ^{
        NSData data = [self downloadFromRemoteService:service];

        dispatch_async(self.storeQueue, ^{
            int img = [self.imageStore addImage:data];

            dispatch_async(self.renderQueue, ^{
                [self renderThumbnail:img];

                dispatch_async(dispatch_get_main_queue(), ^{
                    [[self thumbnailViewForId:img] setNeedsDisplay:YES];
                });
            });
        });
    });
}
```

6

Improve Performance with
Reader-Writer Access

Reader-Writer Access

- Concurrent subsystem queue
DISPATCH_QUEUE_CONCURRENT

Reader-Writer Access

- Concurrent subsystem queue
DISPATCH_QUEUE_CONCURRENT
- synchronous concurrent “reads”
dispatch_sync()

Reader-Writer Access

- Concurrent subsystem queue
`DISPATCH_QUEUE_CONCURRENT`
- synchronous concurrent “reads”
`dispatch_sync()`
- asynchronous serialized “writes”
`dispatch_barrier_async()`

Reader-Writer Access

- Concurrent subsystem queue
`DISPATCH_QUEUE_CONCURRENT`
- synchronous concurrent “reads”
`dispatch_sync()`
- asynchronous serialized “writes”
`dispatch_barrier_async()`

Reader-Writer Access

```
self.storeQueue = dispatch_queue_create("com.example.imageviewer.store",
                                         DISPATCH_QUEUE_CONCURRENT);

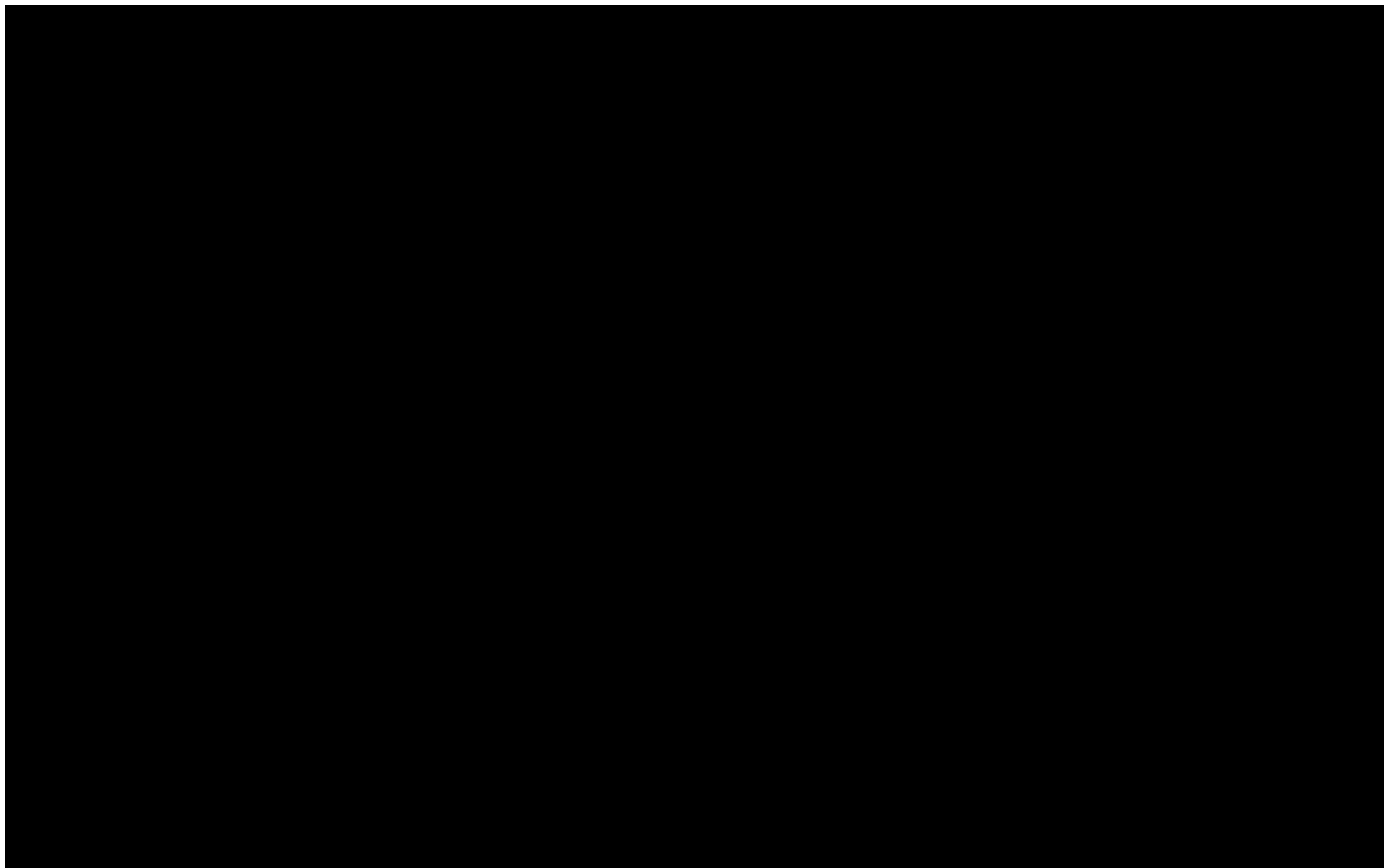
dispatch_barrier_async(self.storeQueue, ^{
    int img = [self.imageStore addImage:data];
    dispatch_async(self.renderQueue, ^{
        [self renderThumbnail:img];
    });
});
```


Reader-Writer Access

```
self.storeQueue = dispatch_queue_create("com.example.imageviewer.store",
                                         DISPATCH_QUEUE_CONCURRENT);

dispatch_barrier_async(self.storeQueue, ^{
    int img = [self.imageStore addImage:data];
    dispatch_async(self.renderQueue, ^{
        [self renderThumbnail:img];
    });
});

- (void)renderThumbnail:(int)img {
    __block NSData *data;
    dispatch_sync(self.storeQueue, ^{
        data = [self.imageStore copyImageData:img];
    });
    // ...
}
```



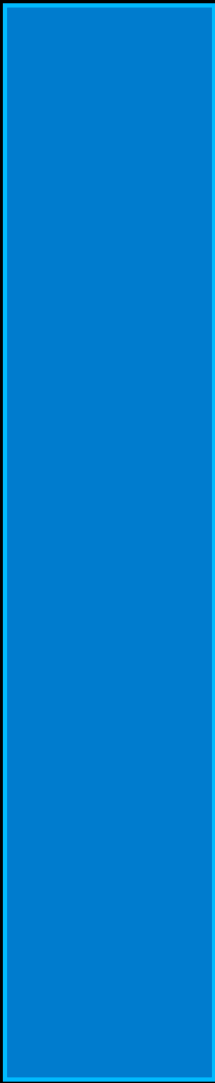
7

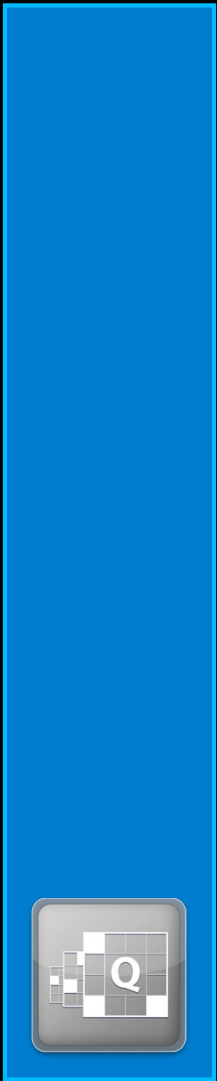
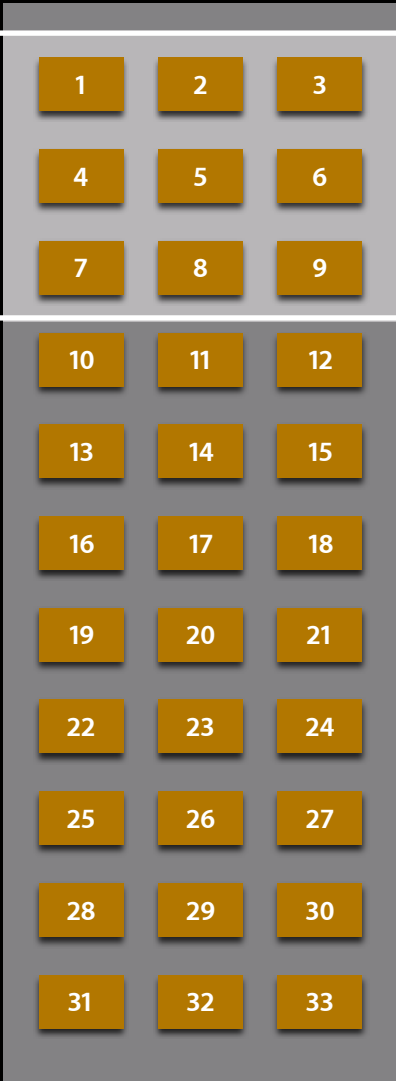
Separate Control and Data Flow

Separate Control and Data Flow

- Dispatch queues not designed for general-purpose data storage
- No cancellation, no random access
- Use data structures appropriate for problem


1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24
25	26	27
28	29	30
31	32	33



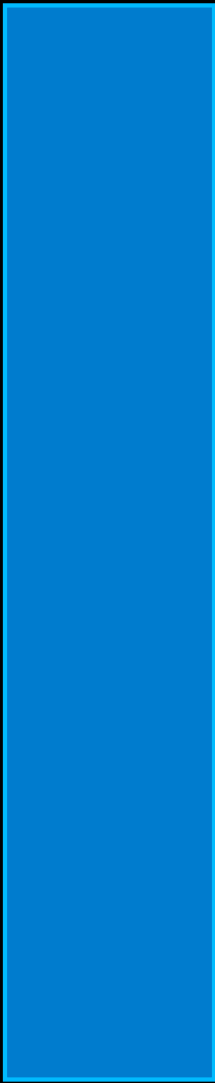


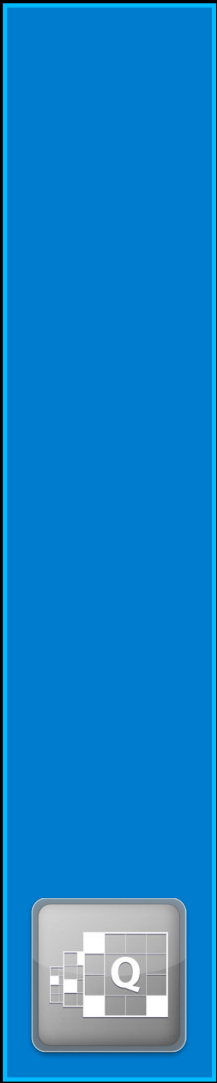
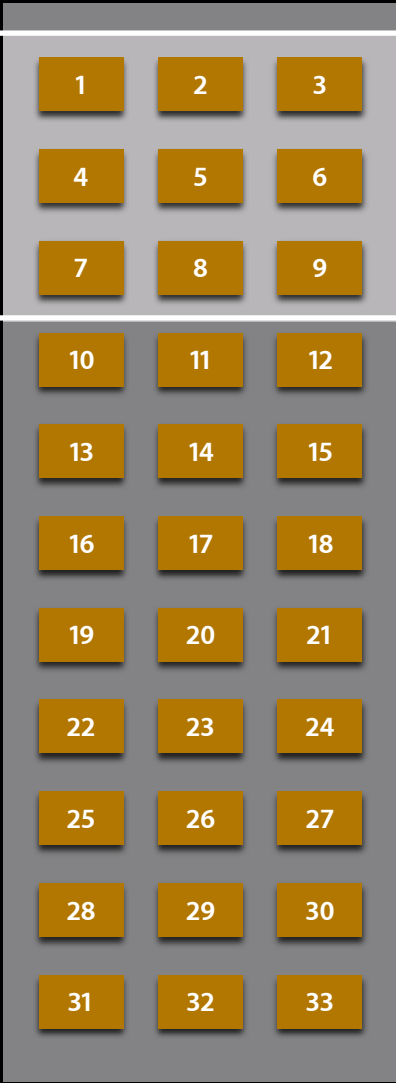
1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24
25	26	27
28	29	30
31	32	33

33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10




1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24
25	26	27
28	29	30
31	32	33

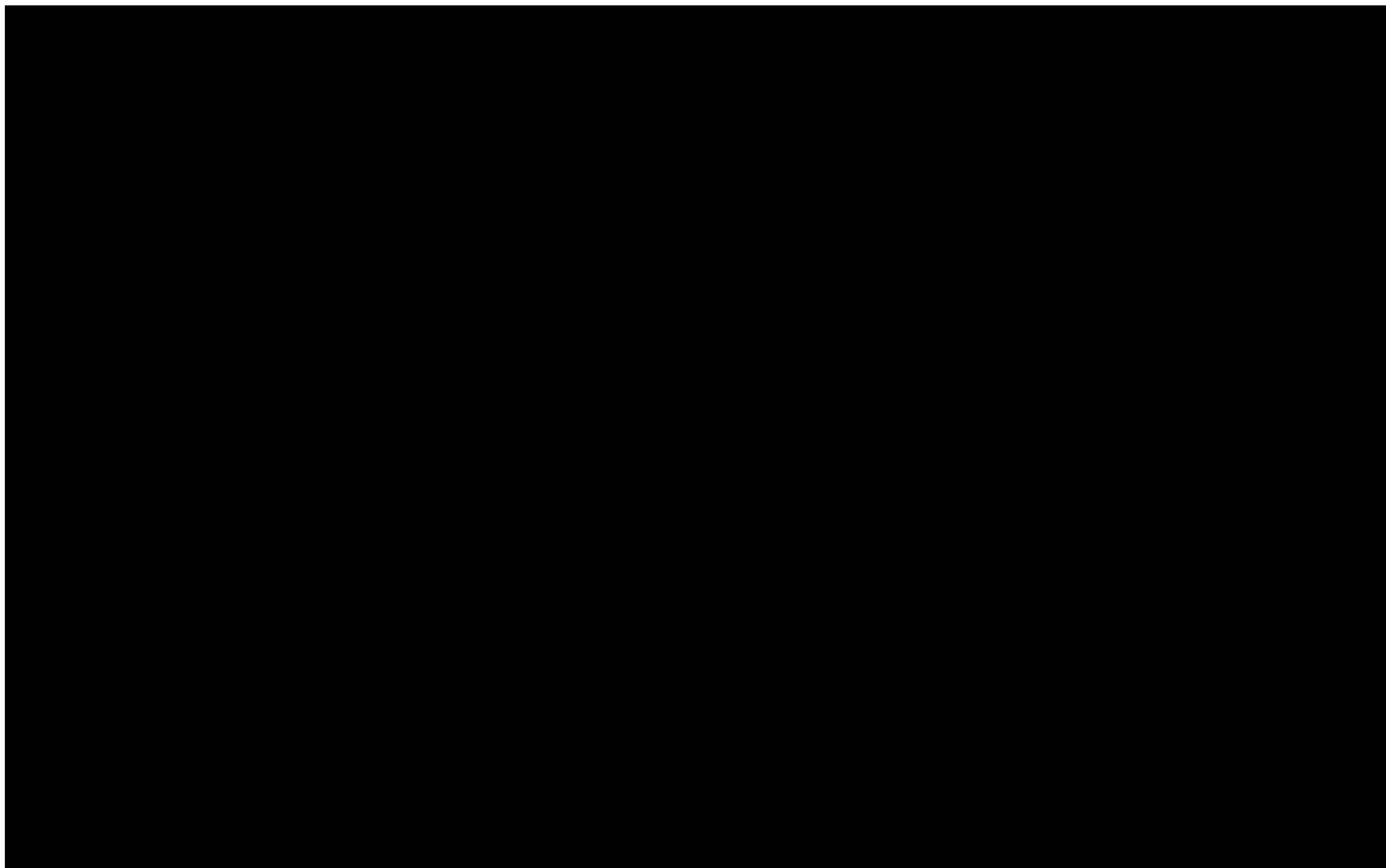




1	2	3
4	5	6
7	8	9
10	11	12
13	14	15
16	17	18
19	20	21
22	23	24
25	26	27
28	29	30
31	32	33

25-33





8

Update State Asynchronously

Update State Asynchronously

```
self.source = dispatch_source_create(DISPATCH_SOURCE_TYPE_DATA_ADD, 0, 0,  
                                     dispatch_get_main_queue());  
  
dispatch_source_set_event_handler(self.source, ^{  
    self.progress += dispatch_source_get_data(self.source);  
    [self.progressBar setProgress:(self.progress/self.total) animated:YES];  
});  
dispatch_resume(self.source);
```

Update State Asynchronously

```
self.source = dispatch_source_create(DISPATCH_SOURCE_TYPE_DATA_ADD, 0, 0,
                                     dispatch_get_main_queue());

dispatch_source_set_event_handler(self.source, ^{
    self.progress += dispatch_source_get_data(self.source);
    [self.progressBar setProgress:(self.progress/self.total) animated:YES];
});
dispatch_resume(self.source);

dispatch_async(self.renderQueue, ^{
    // ...
    dispatch_source_merge_data(self.source, 1);
});
```

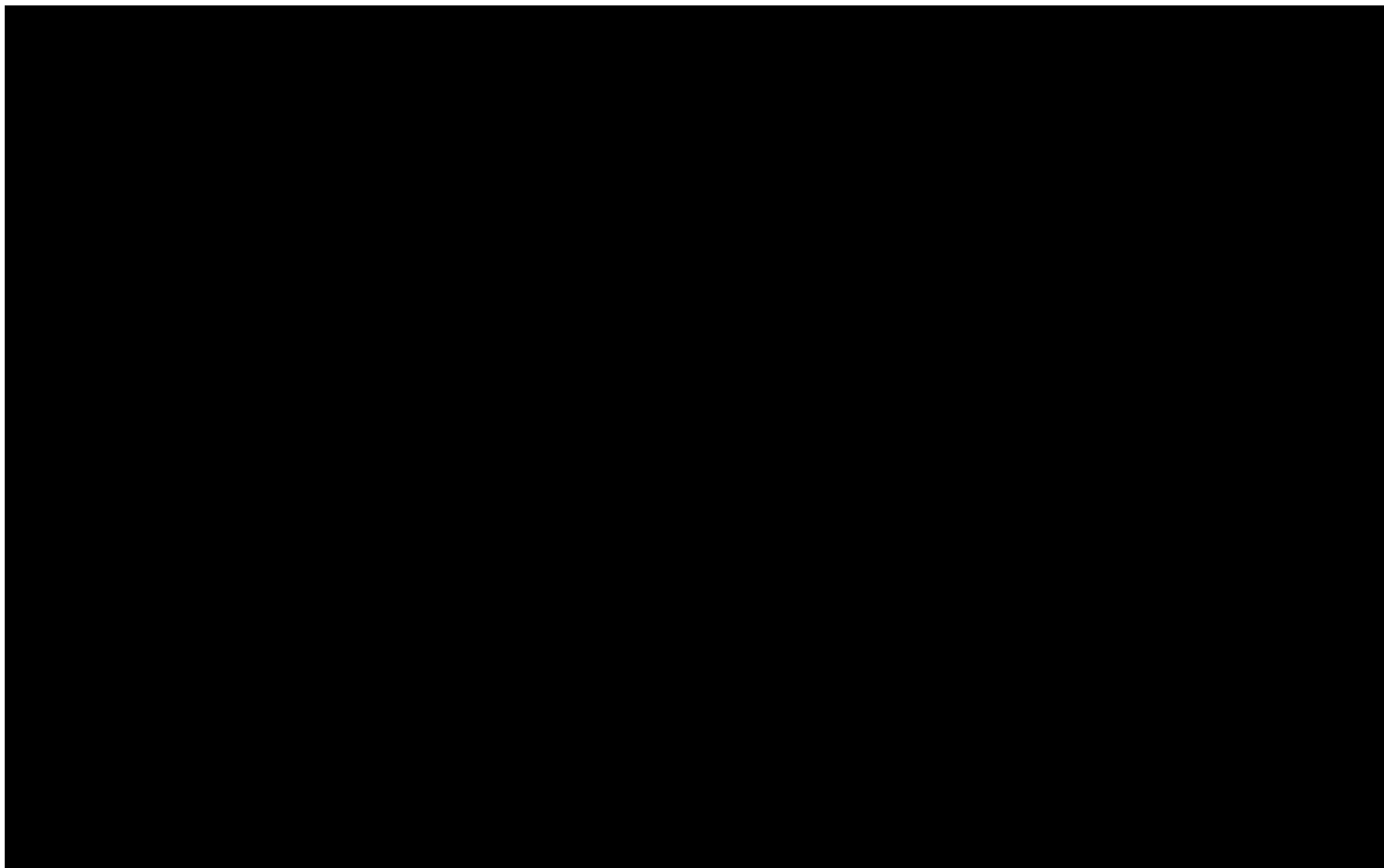
Update State Asynchronously

```
self.source = dispatch_source_create(DISPATCH_SOURCE_TYPE_DATA_ADD, 0, 0,
                                     dispatch_get_main_queue());

dispatch_source_set_event_handler(self.source, ^{
    self.progress += dispatch_source_get_data(self.source);
    [self.progressBar setProgress:(self.progress/self.total) animated:YES];
});
dispatch_resume(self.source);

dispatch_async(self.renderQueue, ^{
    // ...
    dispatch_source_merge_data(self.source, 1);
});

dispatch_source_cancel(self.source);
```

9

Move Out of Process with XPC

Move Out of Process with XPC



- Reliability and security
- Fault isolation and privilege separation
- XPC connection as a “remote queue”

Move Out of Process with XPC



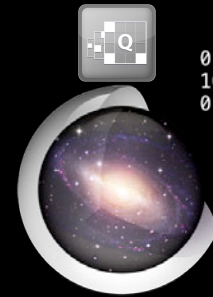
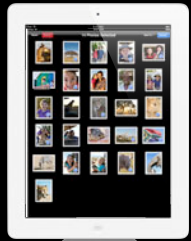
- Reliability and security
- Fault isolation and privilege separation
- XPC connection as a “remote queue”

Cocoa Interprocess Communication with XPC

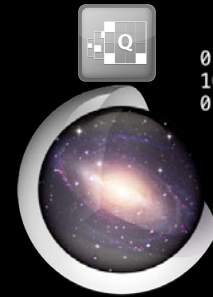
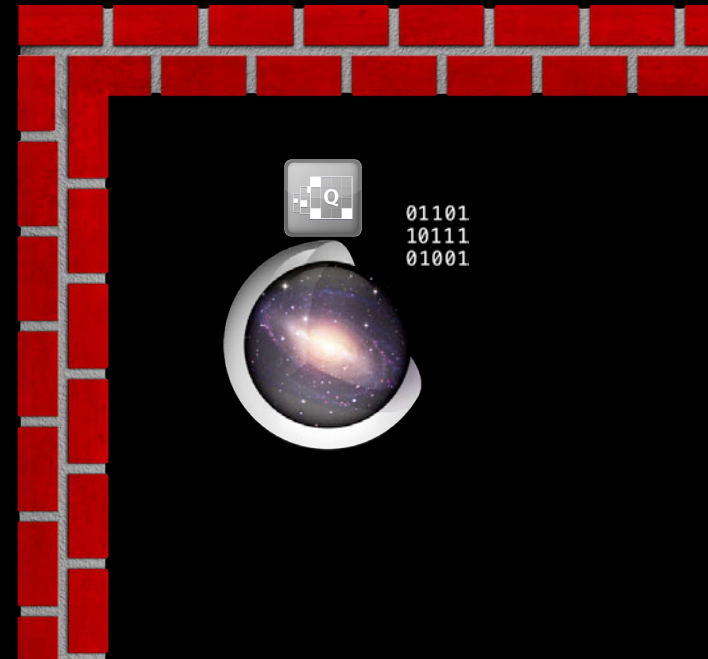
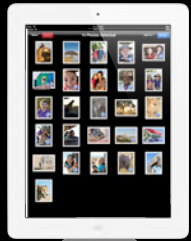
Russian Hill
Thursday 4:30PM

Introducing XPC

WWDC 2011
Session 206



01101
10111
01001



01101
10111
01001

Summary

Summary

1. Don't block the main thread

Summary

1. Don't block the main thread
2. Run in the background with GCD and Blocks

Summary

1. Don't block the main thread
2. Run in the background with GCD and Blocks
3. Don't block many background threads

Summary

1. Don't block the main thread
2. Run in the background with GCD and Blocks
3. Don't block many background threads
4. Integrate with the main runloop

Summary

Summary

5. Use one queue per subsystem

Summary

5. Use one queue per subsystem
6. Improve performance with reader-writer access

Summary

5. Use one queue per subsystem
6. Improve performance with reader-writer access
7. Separate control and data flow

Summary

5. Use one queue per subsystem
6. Improve performance with reader-writer access
7. Separate control and data flow
8. Update state asynchronously with dispatch sources

Summary

5. Use one queue per subsystem
6. Improve performance with reader-writer access
7. Separate control and data flow
8. Update state asynchronously with dispatch sources
9. Move operations out of process with XPC

More Information

Michael Jurewitz

Developer Tools and Performance Evangelist
jurewitz@apple.com

Documentation

Concurrency Programming Guide
Daemons and Services Programming Guide
Transitioning to ARC Release Notes
<http://developer.apple.com>

Apple Developer Forums

<http://devforums.apple.com>

Related Sessions

Adopting Automatic Reference Counting

Nob Hill
Friday 11:30AM

Cocoa Interprocess Communication with XPC

Russian Hill
Thursday 4:30PM

Mastering Grand Central Dispatch

WWDC 2011
Session 210

Introducing XPC

WWDC 2011
Session 206

Labs

Cocoa and XPC Lab

Essentials Lab A
Friday 10:15AM

Open Lab

Core OS Labs A & B
Friday 2:00PM

 **WWDC2012**

