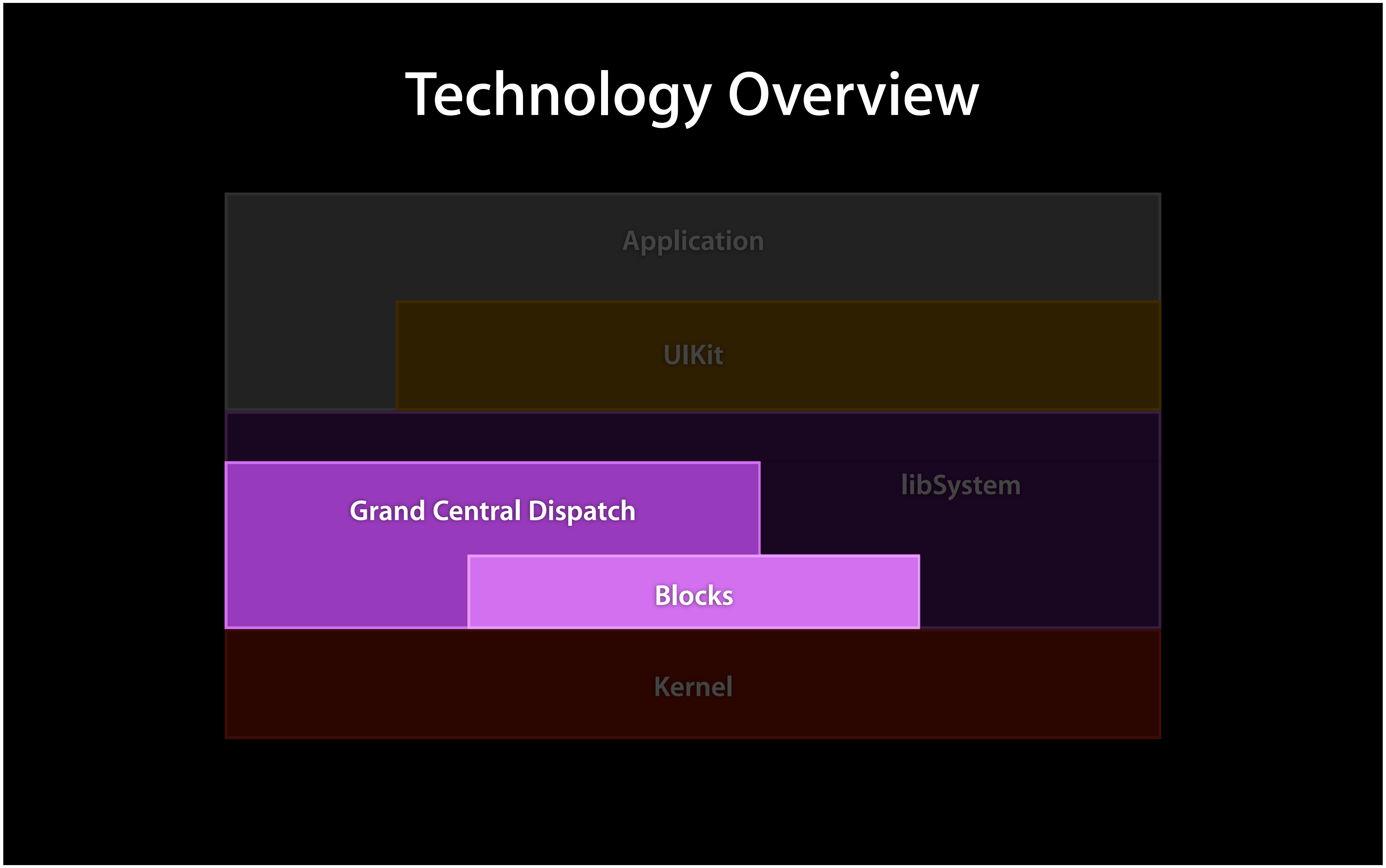
Simplifying iPhone App Development with Grand Central Dispatch

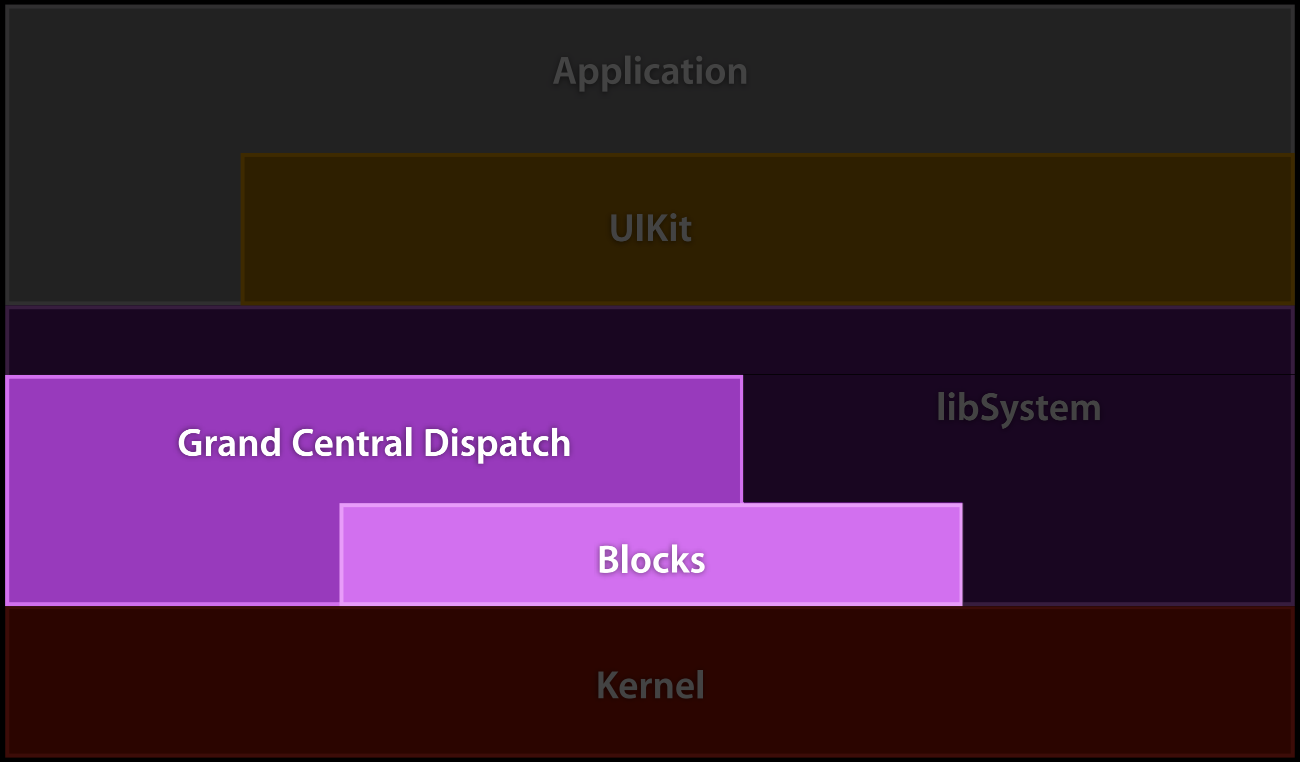
Daniel Steffen
Core OS

What You'll Learn

- Technology overview
- Simplifying multithreaded code
- Design patterns
- GCD objects in depth



Technology Overview



Technology Overview

- GCD is part of libSystem.dylib
- Available to all Apps

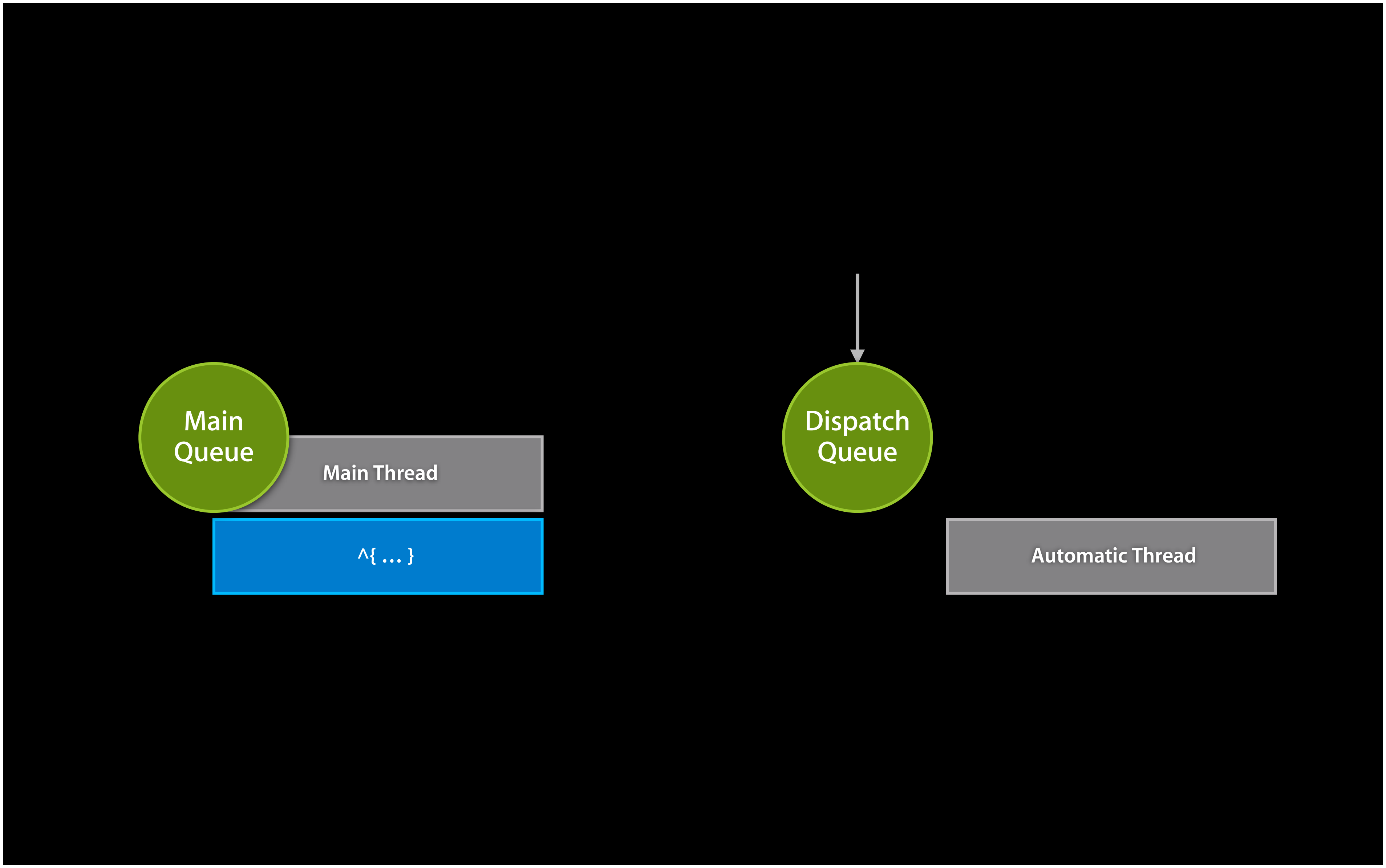
```
#include <dispatch/dispatch.h>
```

- GCD API has block-based and function-based variants
  - Focus today on block-based API

Grand Central Dispatch

Introduction to GCD recap

- Blocks
  - dispatch_async()
- Queues
  - Lightweight list of blocks
  - Enqueue/dequeue is FIFO
- dispatch_get_main_queue()
  - Main thread/main runloop
- dispatch_queue_create()
  - Automatic helper thread


Main
Queue
Main Thread
^{ … }
Dispatch
Queue
Automatic Thread

Simplifying Your Code with GCD

Simplifying Your Code with GCD

GCD advantages

- Efficiency
  - More CPU cycles available for your code
- Better metaphors
  - Blocks are easy to use
  - Queues are inherently producer/consumer
- Systemwide perspective
  - Only the OS can balance unrelated subsystems

Simplifying Your Code with GCD

Compatibility

- Existing threading and synchronization primitives are 100% compatible
- GCD threads are wrapped POSIX threads
  - Do not cancel, exit, kill, join, or detach GCD threads
- GCD reuses threads
  - Restore any per-thread state changed within a block

Threads

Threads

- Why use threads on iPhone?
- App responsiveness
    - Free up main thread
- NSThread, pthread_create()
- Non-trivial cost

Threads

```
- (void)doTimeConsumingOperation:(id)operation {
    id t = [[NSThread alloc] initWithTarget:self
                selector:@selector(runHelperThread:)
                object:operation];
    [t run];
    [t autorelease];
}
- (void)runHelperThread:(id)operation {
    NSAutoreleasePool *p = [NSAutoreleasePool new];
    [operation doOperation];
    [p release];
}
```

Threads

```
- (void)doTimeConsumingOperation:(id)operation {
    dispatch_queue_t queue;
    queue = dispatch_queue_create("com.example.operation", NULL);
    dispatch_async(queue, ^{
        [operation doOperation];
    });
    dispatch_release(queue);
}
```

GCD Advantages

Convenient

- Less boilerplate
- No explicit thread management

GCD Advantages

Efficient

- Thread recycling
- Deferred based on availability

Locking

Locking

- Enforce mutually exclusive access to critical sections
- Serialize access to shared state between threads
- Ensure data integrity

Locking

```
- (void)updateImageCacheWithImg:(UIImage*)img {
    NSLock *l = self.imageCacheLock;
    [l lock];
    // Critical section
    if ([self.imageCache containsObj:img]) {
        [l unlock]; // Don't forget to unlock
        return;
    }
    [self.imageCache addObj:img];
    [l unlock];
}
```

Locking

```
- (void)updateImageCacheWithImg:(NSImage*)img {
    dispatch_queue_t queue = self.imageCacheQueue;
    dispatch_sync(queue, ^{
        // Critical section
        if ([self.imageCache containsObj:img]) {
            return;
        }
        [self.imageCache addObj:img];
    });
}
```

Locking

But wait, there’s even more…

Locking

Deferred critical section

```
- (void)updateImageCacheWithImg:(NSImage*)img {
    dispatch_queue_t queue = self.imageCacheQueue;
    dispatch_async(queue, ^{
        // Critical section
        if ([self.imageCache containsObj:img]) {
            return;
        }
        [self.imageCache addObj:img];
    });
}
```

GCD Advantages

Safe

Cannot return without unlocking

GCD Advantages

More expressive

Deferrable critical sections

GCD Advantages

Efficient

Wait-free synchronization

Inter-Thread Communication

Inter-Thread Communication

- Send messages between threads
- Wake up background threads
- Transfer data between threads

Inter-Thread Communication

Performing selectors

```
– performSelectorOnMainThread:withObject:waitUntilDone:

– performSelector:onThread:withObject:waitUntilDone:

– performSelector:withObject:afterDelay:

– performSelectorInBackground:withObject:
```

Inter-Thread Communication

performSelector:onThread:withObject:waitUntilDone:

```
// waitUntilDone: NO
dispatch_async(queue, ^{
    [myObject doSomething:foo withData:bar];
});


// waitUntilDone: YES
dispatch_sync(queue, ^{
    [myObject doSomething:foo withData:bar];
});
```

Inter-Thread Communication

performSelector:withObject:afterDelay:

```
dispatch_time_t delay;
delay = dispatch_time(DISPATCH_TIME_NOW, 50000 /* 50µs */);

dispatch_after(delay, queue, ^{
    [myObject doSomething:foo withData:bar];
});
```

Inter-Thread Communication

performSelectorInBackground:withObject:

```
dispatch_queue_t queue = dispatch_get_global_queue(0, 0);

dispatch_async(queue, ^{
    [myObject doSomething:foo withData:bar];
});
```

GCD Advantages

Flexible

- Blocks
  - Can call any selector and multiple selectors
  - No need to pack and unpack arguments

GCD Advantages

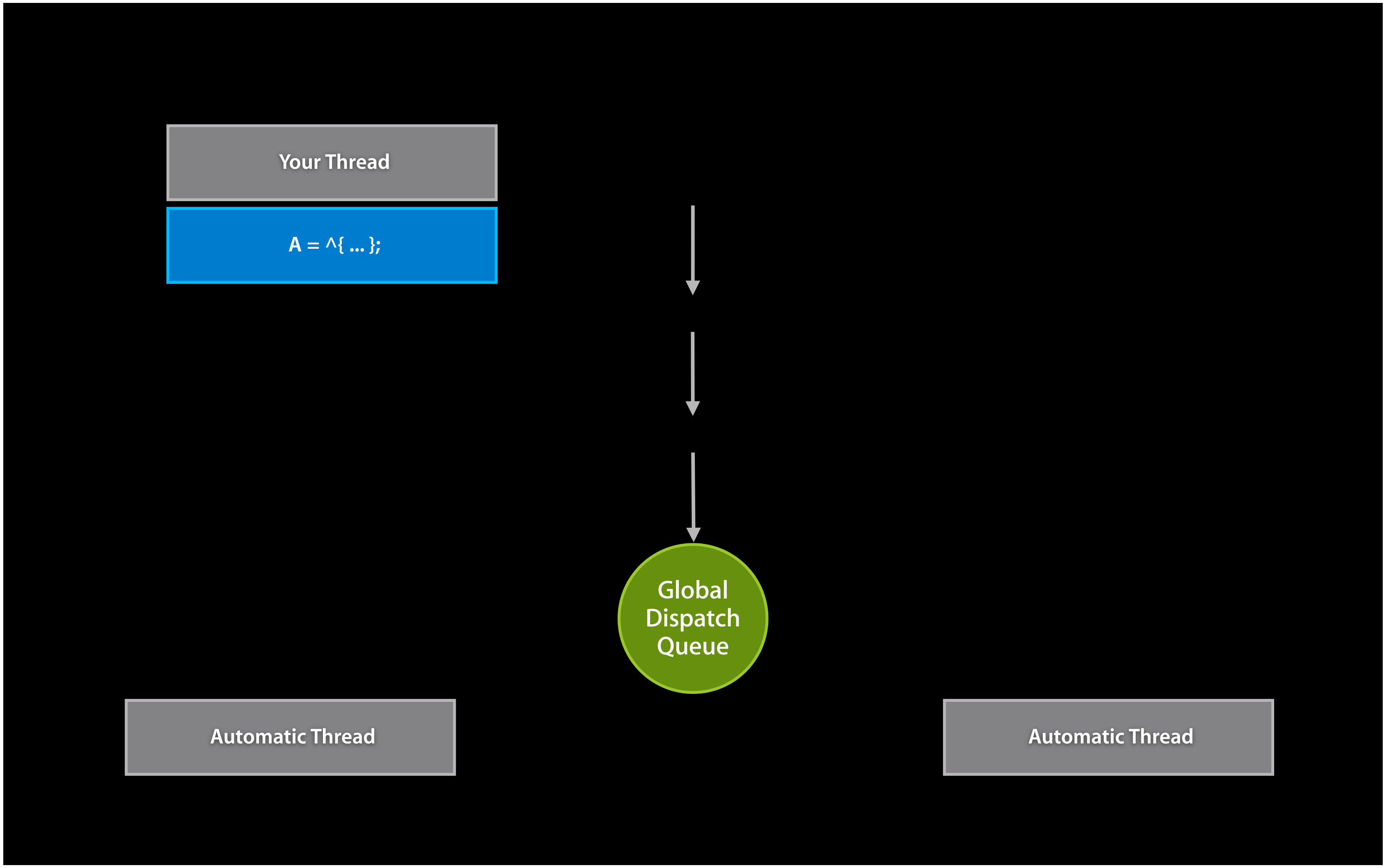
Efficient

- Queues
  - Helper threads created/woken up as needed

Global Queues

Global Queues

- Enqueue/dequeue is FIFO
- Concurrent execution
  - Non-FIFO completion order
- `dispatch_get_global_queue(priority, 0)`


Your Thread
A = ^{ ... };
Global Dispatch Queue
Automatic Thread
Automatic Thread

Global Queues

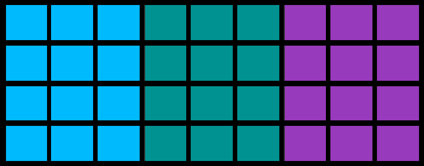
- Global queues map GCD activity to real threads
- Priority bands
  - DISPATCH_QUEUE_PRIORITY_HIGH
  - DISPATCH_QUEUE_PRIORITY_DEFAULT
  - DISPATCH_QUEUE_PRIORITY_LOW

GCD Design Patterns

Shiva Bhattacharjee
Core OS

GCD Design Patterns

One queue per task or subsystem



- Easy communication
  - dispatch_async()
- Queues are inherently producer/consumer
  - Blocks carry data between tasks
- Queues are lightweight and efficient
  - Automatic thread creation and recycling

GCD Design Patterns

Low-level event notifications

- Similar approach to UI event-driven programming
- Don't poll or block a thread waiting for external events
  - Waiting on a socket
  - Polling for directory changes
- Dispatch sources
  - Monitor external OS events
  - Respond on-demand

Dispatch Sources

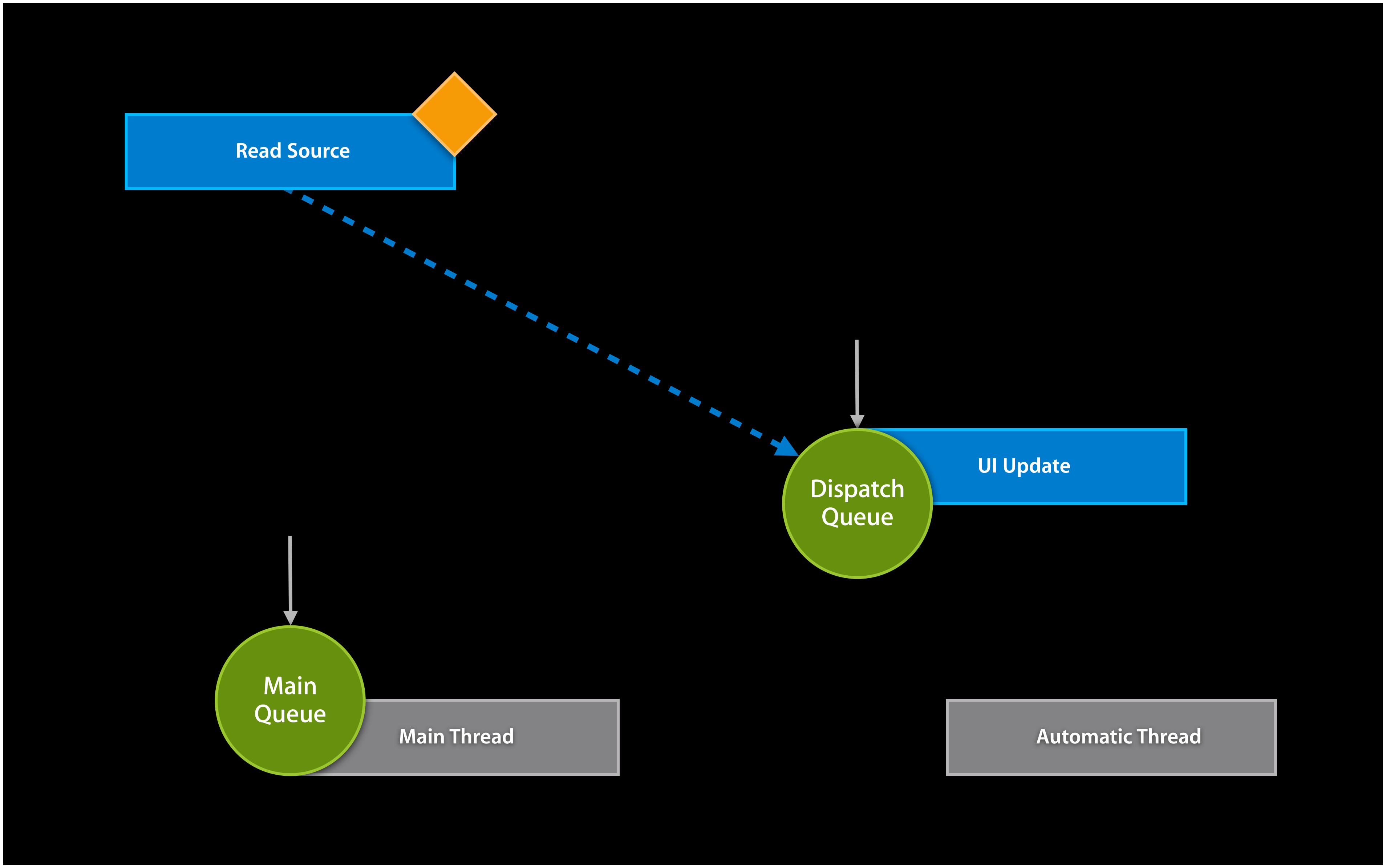
Dispatch Sources

- Simple unified way to monitor low-level events
  - dispatch_source_create()
- Event handlers delivered to any queue
  - Monitoring and event handling is decoupled
- Event handler is not re-entrant
- Suspend and resume at will
  - Sources are created suspended, initial resume is required

Dispatch Sources

Creating a read source

```
int socket; // file-descriptor, set to be non-blocking

dispatch_source_t source = dispatch_source_create(
        DISPATCH_SOURCE_TYPE_READ, socket, 0, queue);

dispatch_source_set_event_handler(source, ^{
    size = read(socket, buffer, sizeof(buffer));
    if (size == -1 && errno == EAGAIN) {
        // non-blocking I/O returned no data
        // will get called again when more data available
    }
});

dispatch_resume(source);
```


Read Source
Dispatch Queue
UI Update
Main Queue
Main Thread
Automatic Thread

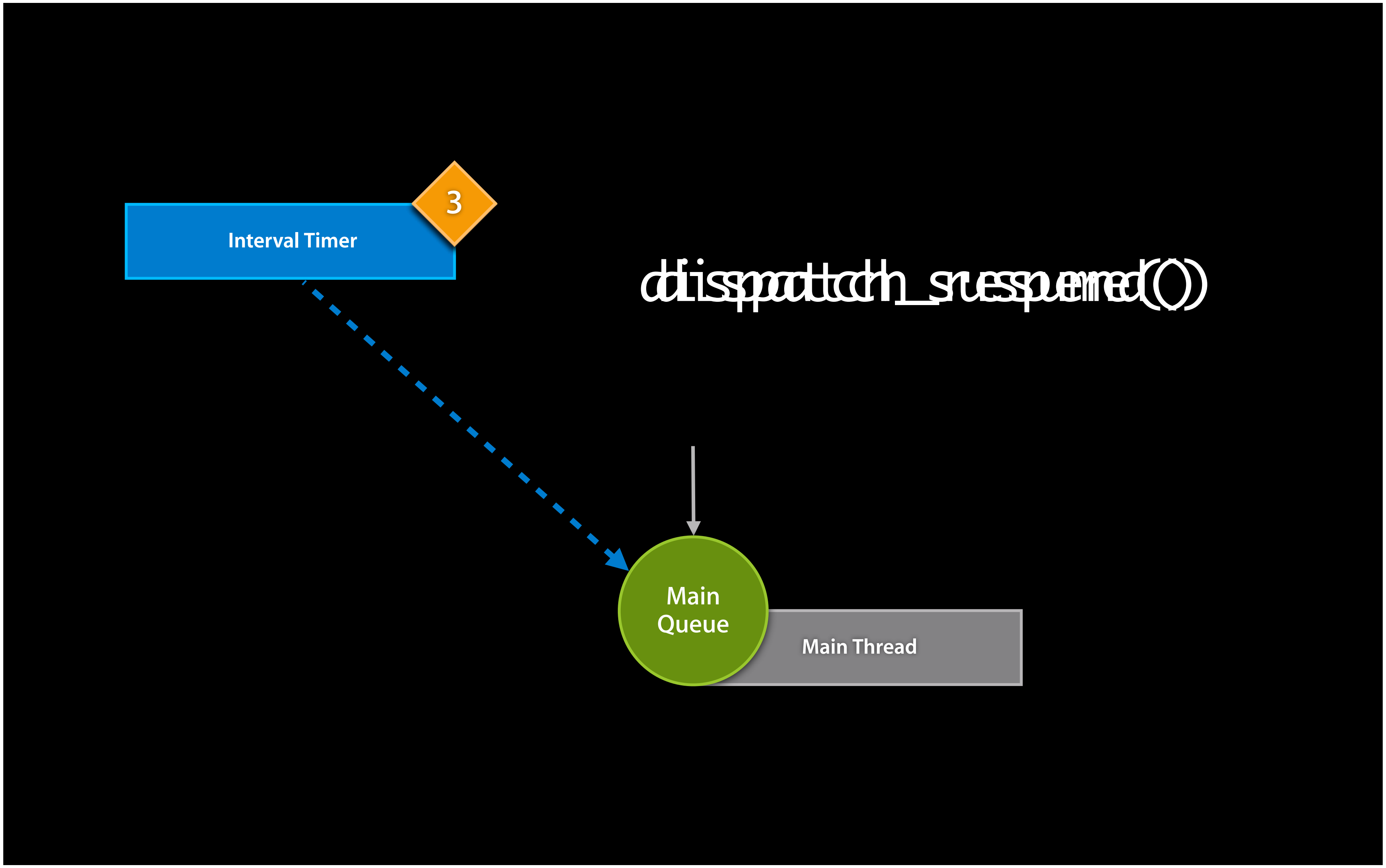
Dispatch Sources

- Coalesce event data in background
  - While handling events or when source suspended
  - dispatch_source_get_data()
- High performance
  - Data coalesced with atomic operations
  - No ephemeral heap allocations
- Monitor all event types supported by BSD kqueue

Dispatch Sources

Source types and event data

Type	Data	Handle
`DISPATCH_SOURCE_TYPE_READ`	`count`	`int (fd)`
`DISPATCH_SOURCE_TYPE_WRITE`	`count`	`int (fd)`
`DISPATCH_SOURCE_TYPE_VNODE`	`bitmask`	`int (fd)`
`DISPATCH_SOURCE_TYPE_TIMER`	`count`	
`DISPATCH_SOURCE_TYPE_DATA_ADD`	`count`	
`DISPATCH_SOURCE_TYPE_DATA_OR`	`bitmask`	



dispatch_suspend()
dispatch_resume()

Source Cancellation

Source Cancellation

- Stops event delivery asynchronously
  - Does not interrupt event handler
- Optional cancellation handler
  - Required for filedescriptor-based sources
  - Opportunity to deallocate resources
  - Delivered only once
- Suspension defers cancellation handler

Source Cancellation

Canceling a read source

```
dispatch_source_t source = dispatch_source_create(
        DISPATCH_SOURCE_TYPE_READ, socket, 0, queue);
dispatch_source_set_event_handler(source, ^{
    if (dispatch_source_get_data(source) == 0 /* EOF */ ) {
        dispatch_source_cancel(source);
        return;
    }
    size = read(socket, buffer, sizeof(buffer));
});
dispatch_source_set_cancel_handler(source, ^{
    close(socket);
});
dispatch_resume(source);
```

Target Queues

Target Queues

Sources

Advanced

- Target queue passed at creation time
- Changeable
  - dispatch_set_target_queue()

Target Queues

Queues

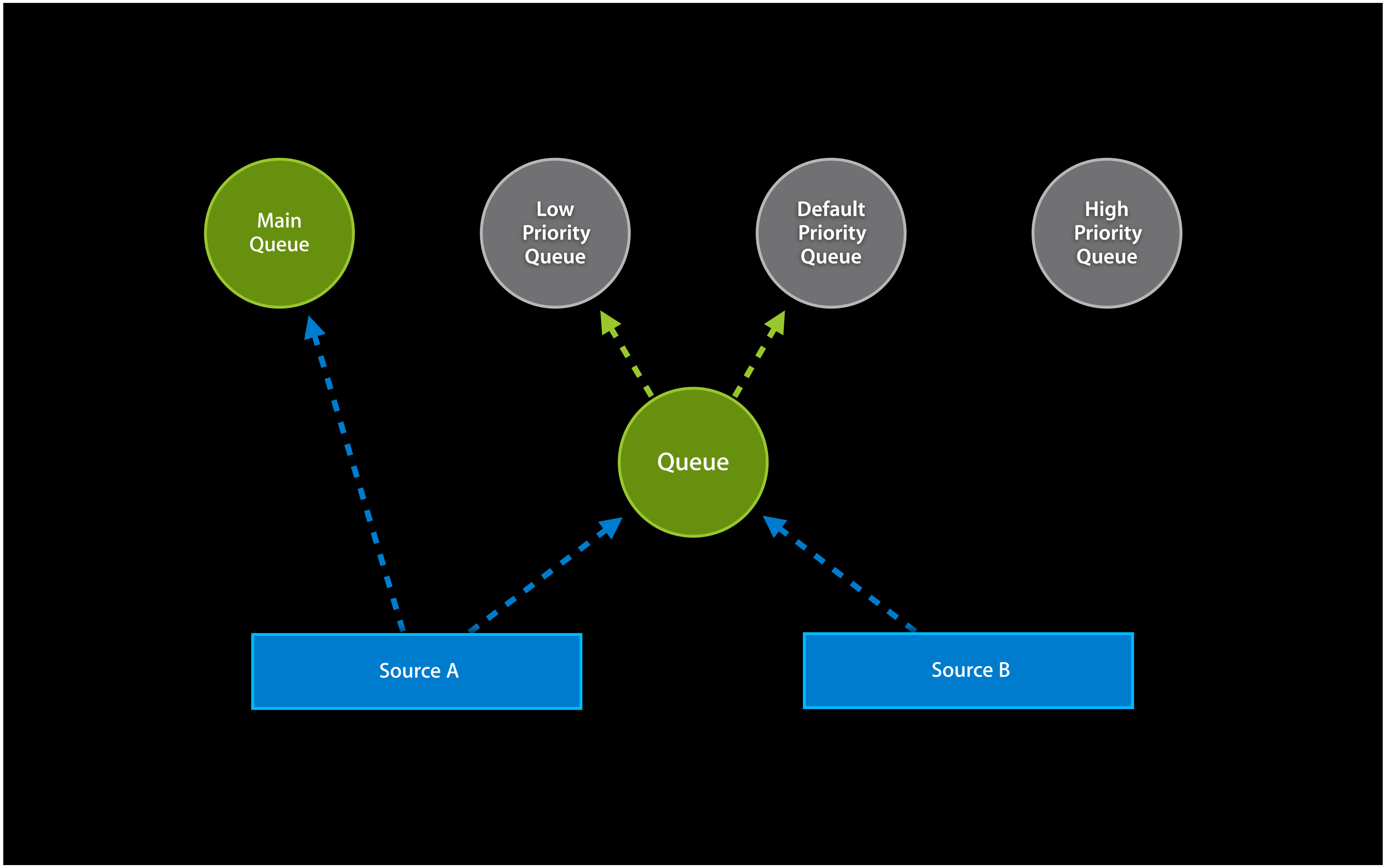
Advanced

- Global queues map GCD activity to real threads
  - Ultimate location of block execution
- Can change target queue of queues you create
  - Specifies where blocks execute
- Default target queue
  - Global queue with DISPATCH_QUEUE_PRIORITY_DEFAULT

Target Queues

Advanced

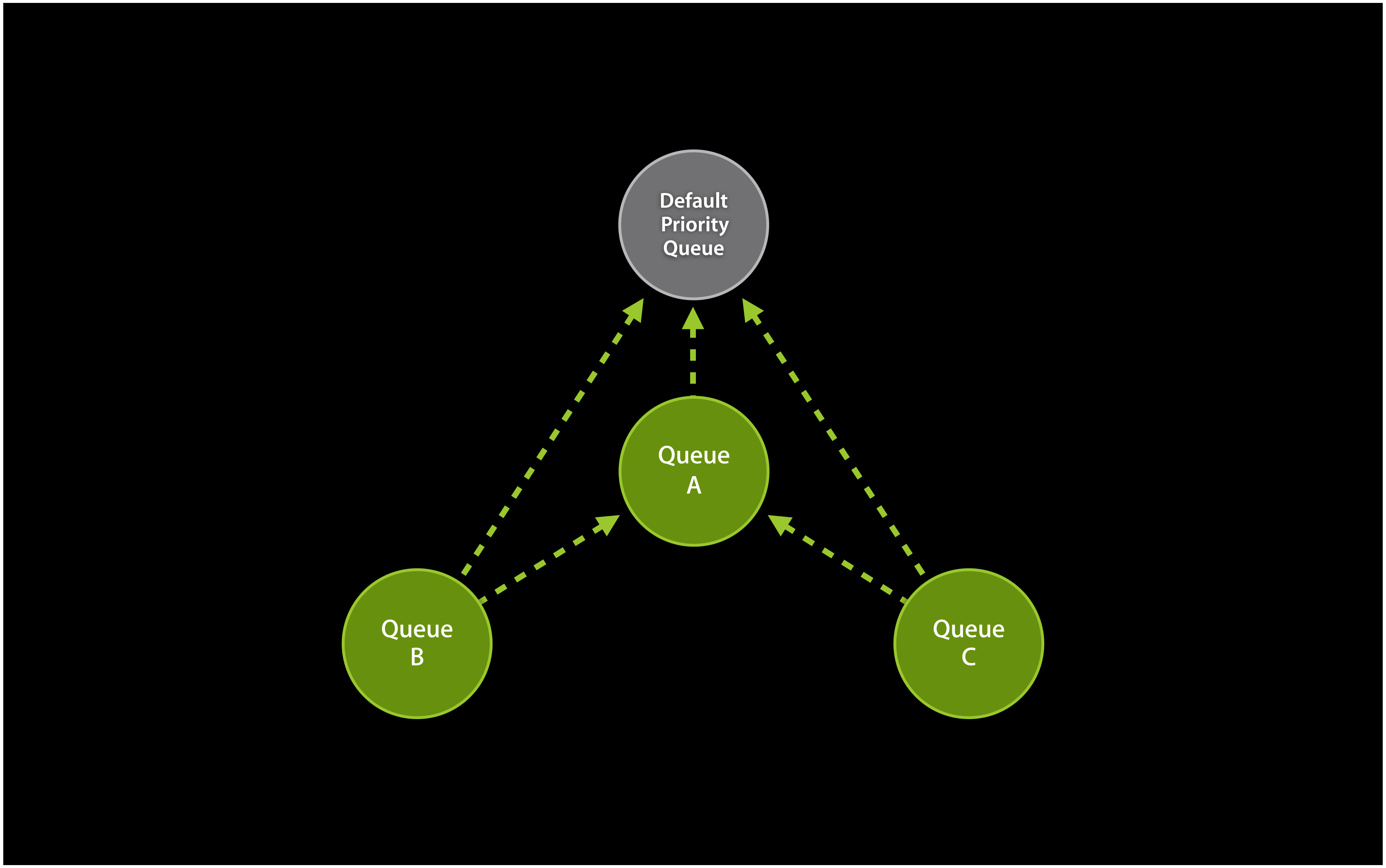
```
dispatch_queue_t queue, target;

queue = dispatch_queue_create(“com.example.test”, NULL);
target = dispatch_get_global_queue(
            DISPATCH_QUEUE_PRIORITY_LOW, 0);

dispatch_set_target_queue(queue, target);
```


Main Queue
Low Priority Queue
Default Priority Queue
High Priority Queue
Queue
Source A
Source B

Target Queues

Advanced

- Arbitrary hierarchies are supported
  - Creating loops is undefined
- Block ordering between different subqueues
  - Many blocks on subqueue ⇔ one block on target queue


Default Priority Queue
Queue A
Queue B
Queue C

Target Queues

Advanced

- Why stack your queues?
- For example
  - One subqueue per access type to global data structure
  - Can independently control each access type

GCD Objects

Queues

`dispatch_queue_t`

```
dispatch_queue_create
dispatch_queue_get_label
dispatch_get_main_queue
dispatch_get_global_queue
dispatch_get_current_queue
dispatch_main
dispatch_async
dispatch_async_f
dispatch_sync
dispatch_sync_f
dispatch_after
dispatch_after_f
dispatch_apply
dispatch_apply_f
```

Semaphores

`dispatch_semaphore_t`

```
dispatch_semaphore_create
dispatch_semaphore_signal
dispatch_semaphore_wait
```

Objects

`dispatch_object_t`

```
dispatch_retain
dispatch_release
dispatch_suspend
dispatch_resume
dispatch_debug
dispatch_get_context
dispatch_set_context
dispatch_set_finalizer_f
dispatch_set_target_queue
```

Groups

`dispatch_group_t`

```
dispatch_group_create
dispatch_group_enter
dispatch_group_leave
dispatch_group_wait
dispatch_group_notify
dispatch_group_notify_f
dispatch_group_async
dispatch_group_async_f
```

Sources

`dispatch_source_t`

```
dispatch_source_create
dispatch_source_cancel
dispatch_source_testcancel
dispatch_source_merge_data
dispatch_source_get_handle
dispatch_source_get_mask
dispatch_source_get_data
dispatch_source_set_timer
dispatch_source_set_event_handler
dispatch_source_set_event_handler_f
dispatch_source_set_cancel_handler
dispatch_source_set_cancel_handler_f
```

Not Objects

Time

`dispatch_time_t`

```
dispatch_time
dispatch_walltime
```

Once

`dispatch_once_t`

```
dispatch_once
dispatch_once_f
```

GCD Objects

- Dispatch objects are reference counted
  - `dispatch_retain(object);`
  - `dispatch_release(object);`
- GCD retains parameters to dispatch API as needed

GCD Objects

Managing object lifetime

- Ensure objects captured by blocks are valid when blocks are executed
  - Objects must be retained and released around asynchronous operations
- Objective-C objects captured by blocks are auto-retained and auto-released
- Other objects captured by blocks must be retained by your code
  - CFRetain()/CFRelease()
  - dispatch_retain()/dispatch_release()

GCD Objects

Suspend and resume

- Suspend and resume only affects queues and sources you create
  - Sources are created suspended
- Suspension is asynchronous
  - Takes effect between blocks
- Your queues can predictably suspend objects that target them

GCD Objects

Application contexts

- Applications can attach custom data to GCD objects
  - dispatch_set_context()/dispatch_get_context()
- Optional finalizer callback
  - dispatch_set_finalizer_f()
  - Allows attached context to be freed with object
  - Called on the target queue of the object

More Information

Michael Jurewitz
Developer Tools and Performance Evangelist
jurewitz@apple.com

Documentation
Concurrency Programming Guide
http://developer.apple.com

Open Source
Mac OS Forge > libdispatch
http://libdispatch.macosforge.org

Apple Developer Forums
http://devforums.apple.com

Related Session

Introducing Blocks and Grand Central Dispatch on iPhone	Russian Hill Wednesday 11:30AM
Introducing Blocks and Grand Central Dispatch on iPhone (R)	Pacific Heights Friday 2:00PM

Lab

Grand Central Dispatch Lab	Core OS Lab A Friday 11:30AM

WWDC10