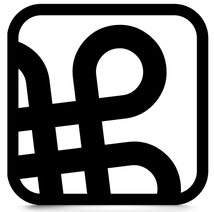


Macoun

ProjectWizards

Gutes Werkzeug, halbe Arbeit!

Kai Brüning & Frank Illenberger

Einleitung

Einleitung

- ProjectWizards
- Entwicklung einer Projektmanagement-Anwendung seit 2004
  - Merlin Project (macOS)
  - Merlin Project Go (iOS)
  - Merlin Server (macOS)
  - Objective-C!
  - auf Swift übertragbar

Einleitung

- Sammlung von bewährten Werkzeugen
  - Unit-Tests
  - Debugging
  - Produktion
- Das Basteln eigener Werkzeuge lohnt sich
  - …ist aber manchmal leider auch bitter nötig.

Wir verpacken
Grand Central Dispatch

Grand Central Dispatch

- 2009 mit Mac OS 10.6 veröffentlicht
- Tolles Konzept, aber
  - sperrige C-API
  - unterstützte keine Garbage Collection
- Entwicklung eines Objective-C-Wrappers
  - Volle Semantik von GCD
    - nicht eingeschränkt wie NSOperationQueue
    - mittlerweile natürlich auf ARC

GCD Wrapper

```
PWDispatchQueue* queue =
     [PWDispatchQueue serialDispatchQueueWithLabel:@"My Queue“];

[queue synchronouslyDispatchBlock:^{
    // do some stuff...
}];

PWDispatchSemaphore* semaphore =
        [[PWDispatchSemaphore alloc] initWithInitialValue:0];

[queue asynchronouslyDispatchBlock:^{
    // do more stuff...
    [semaphore signal];
}];
[semaphore waitWithTimeout:4.0 useWallTime:YES];
```

GCD Wrapper

PWDispatchQueue	PWDispatchSource
PWDispatchGroup	PWDispatchFileReader
PWDispatchSemaphore	PWDispatchFileWriter
PWDispatchIOChannel	PWDispatchPathObserver
PWDispatchIORandomChannel	PWDispatchSignalObserver
PWDispatchIOStreamChannel	PWDispatchProcessObserver
	PWDispatchTimer
	PWDispatchMemoryPressureObserver

GCD Wrapper

PWDispatchQueue	PWDispatchSource
PWDispatchGroup	PWDispatchFileReader
PWDispatchSemaphore	PWDispatchFileWriter
PWDispatchIOChannel	PWDispatchPathObserver
PWDispatchIORandomChannel	PWDispatchSignalObserver
PWDispatchIOStreamChannel	PWDispatchProcessObserver
	PWDispatchTimer
	PWDispatchMemoryPressureObserver

GCD-Wrapper

- Besonderheiten
  - Garbage Collection & ARC mittlerweile erledigt
  - Swift 3 hat swiftige GCD-APIs (7 Jahre später)
  - Reentranz

GCD-Reentranz

dispatch_sync unterstützt keine Reentranz!

```
dispatch_queue_t queue
    = dispatch_queue_create("My Queue", DISPATCH_QUEUE_SERIAL);

dispatch_sync(queue, ^{
    dispatch_sync(queue, ^{
        // Deadlocks
    });
});
```

GCD-Reentranz

- Ist aber nötig, wenn
  - man nicht auf synchrone calls verzichten kann/will
  - man Queues zum Schützen von Ressourcen verwendet
  - und Code Composition gebraucht
  - Traditionell: Recursive Lock

Reentranz bei Composition

```
- (int)calculateValueA {
    __block int result;
    dispatch_sync(_queue, ^{
        // perform calculation
    });
    return result;
}
```

```
- (int)calculateValueB {
    __block int result;
    dispatch_sync(_queue, ^{
        // perform other calculation
    });
    return result;
}
```

```
- (void)updateState {
    dispatch_sync(_queue, ^{
        self.state = [self calculateValueA] + [self calculateValueB];
    });
}
```

GCD-Reentranz

- Lösung
  - Man prüft, ob man bereits auf der Queue ist
    - wenn nein, dispatch_sync
    - wenn ja, Block sofort direkt aufrufen
    - Nachteil: Semantik von Dispatch Queues ändert sich
      - Reinhenfolgengarantie ist aufgehoben
      - Für async bleibt sie erhalten

GCD-Reentranz

- Naive Implementation: dispatch_get_current_queue()
  - deprecated seit iOS 6 / Mac OS 10.8
  - man kann auf mehreren Queues gleichzeitig sein!

GCD-Reentranz

Man kann auf mehreren Queues gleichzeitig sein.

```
PWDispatchQueue* queueA = [PWDispatchQueue serialDispatchQueueWithLabel:@„A"];
PWDispatchQueue* queueB = [PWDispatchQueue serialDispatchQueueWithLabel:@"B"];
PWDispatchQueue* queueC = [PWDispatchQueue serialDispatchQueueWithLabel:@"C"];
[queueA synchronouslyDispatchBlock:^{
    [queueB synchronouslyDispatchBlock:^{
        [queueC synchronouslyDispatchBlock:^{
                // Hier bin ich auf queue A, B & C gleichzeitig
            }];
        }];
    }];
```

Auf welcher Queue bin ich?

- Feststellen, ob man gerade auf einer gegeben Queue ist.
  - Sloppy Lock Pattern
  - man page zu dispatch_sync
  - nicht einfach, viele Randfälle
  - Implementiert in -[PWDispatchQueue isCurrentDispatchQueue]
  - Verwenden wir in -synchronouslyDispatchBlock:
  - Ähnlich wie -[NSManagedObjectContext performBlock:]

Dispatch-Queue-Kontrakte

- Queues dienen zum Schützen von Ressourcen
- Assertions mit isCurrentDispatchQueue helfen, Races zu vermeiden
  - Dokumentiert auch den Kontrakt
  - Auch negative Assertions helfen:
    - Auf welcher Queue darf ich gerade definitiv nicht sein?
  - Seit iOS 10 / macOS 10.12 auch als Bordmittel
    - dispatch_assert_queue / dispatch_assert_queue_not

Queue-Assertions

```
- (BOOL)processProject:(MEProject*)project
                 error:(NSError**)outError
{
    PWAssert(self.dispatchQueue.isCurrentDispatchQueue);
    PWAssert(project.meeManagedObjectContext.isCurrentDispatchQueue);
    PWAssert(!self.document.isCurrentDispatchQueue);

  // ...
```

Dispatch-Queue-Graph

- Synchrones Dispatchen kann zu Deadlocks führen.

```
PWDispatchQueue* queueA = [PWDispatchQueue serialDispatchQueueWithLabel:@"A"];
PWDispatchQueue* queueB = [PWDispatchQueue serialDispatchQueueWithLabel:@"B"];
[queueA asynchronouslyDispatchBlock:^{
     [queueB synchronouslyDispatchBlock:^{
           // …
     }];
   }];
[queueB asynchronouslyDispatchBlock:^{
    [queueA synchronouslyDispatchBlock:^{
           // ...
    }];
}];
```

Dispatch-Queue-Graph

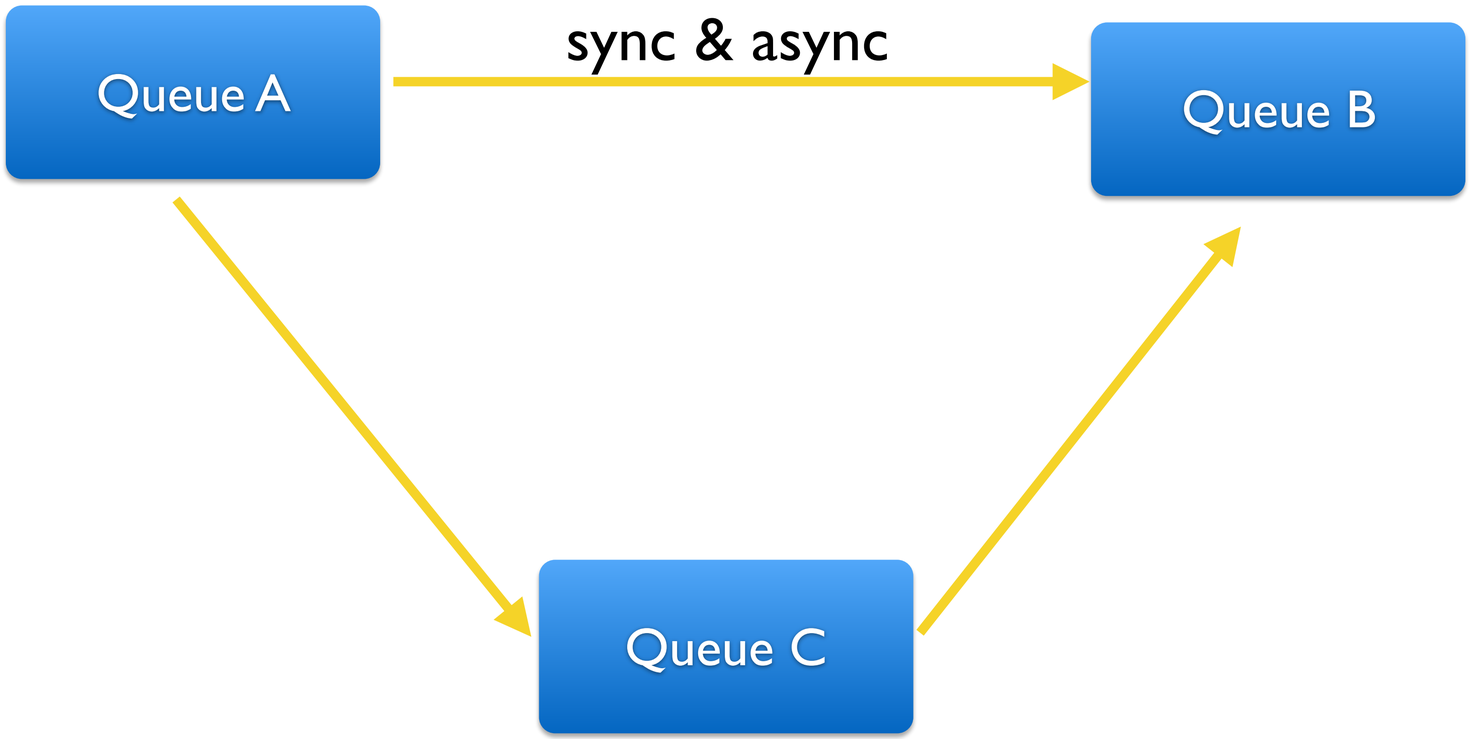
- Synchrones Dispatchen kann zu Deadlocks führen.
- Am besten wäre ein komplett asynchroner Aufbau.
  - Nicht immer praktikabel
    - Andocken an synchrone APIs
    - Performance: sync um Faktor 300 schneller als async (ohne Contention)
    - Sehr verbos
    - Schwer zu debuggen und zu denken

Dispatch-Queue-Graph

- Wie vermeidet man sicher Deadlocks mit synchrous dispatch?
  - Mit einem klaren Kontrakt, welche Queue welche synchron beschicken darf

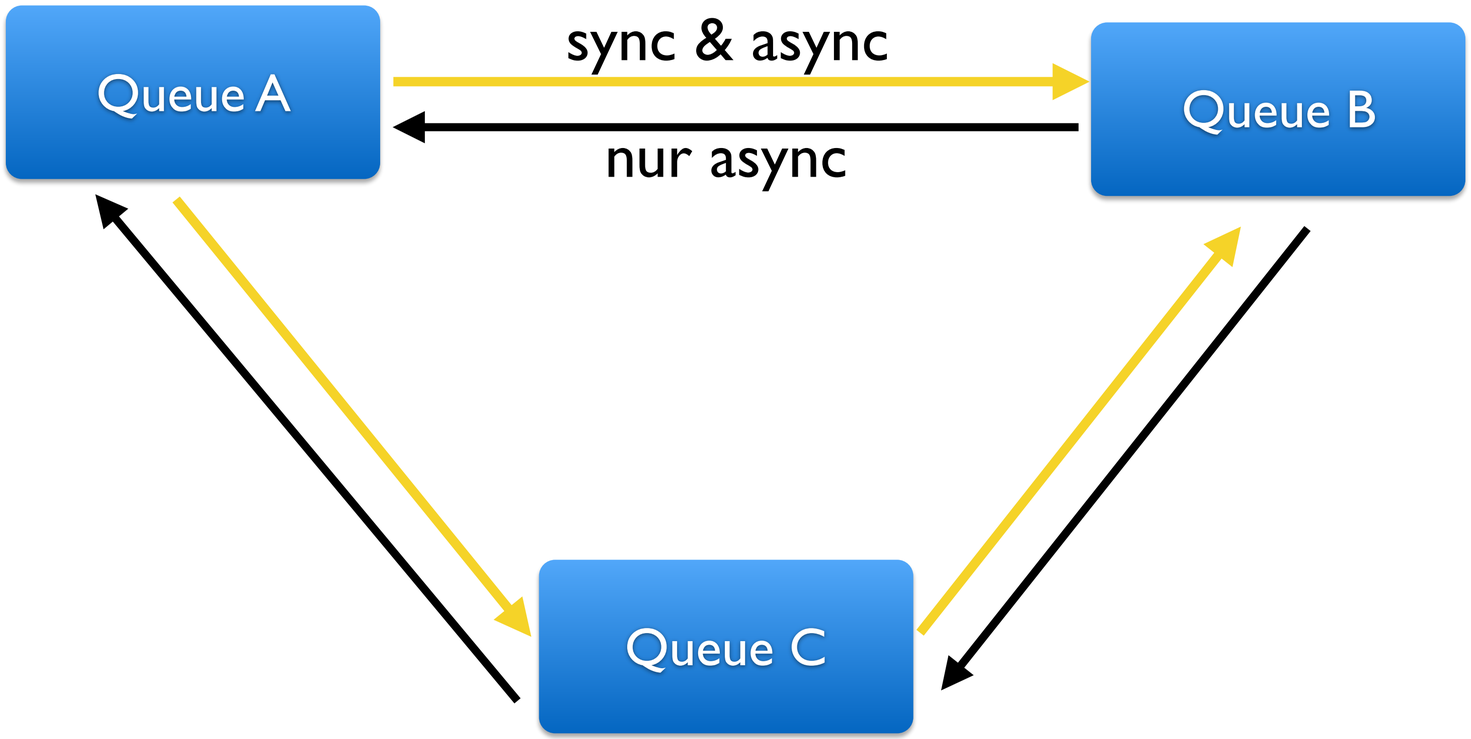
Dispatch-Queue-Graph

Beispielkontrakt



Dispatch-Queue-Graph

Beispielkontrakt



Dispatch-Queue-Graph

- Wie vermeidet man sicher Deadlocks mit synchrous dispatch?
  - Mit einem klaren Kontrakt, welche Queue welche synchron beschicken darf
  - Der Graph aus Sync Calls sollten azyklisch sein (Baum)
    - Ein Graph mit Zyklen kann trotzdem korrekt sein.
      - Es ist aber unheimlich schwer, sich sicher zu sein.
  - Man kann zur Laufzeit sicher stellen (sanitizing), dass der Graph azyklisch bleibt.

```
PWDispatchQueue* queueA = [PWDispatchQueue serialDispatchQueueWithLabel:@"A"];
PWDispatchQueue* queueB = [PWDispatchQueue serialDispatchQueueWithLabel:@"B"];
PWDispatchQueue* queueC = [PWDispatchQueue serialDispatchQueueWithLabel:@"C"];
PWDispatchGroup* group = [[PWDispatchGroup alloc] init];

[queueA asynchronouslyDispatchBlock:^{
    [queueB synchronouslyDispatchBlock:^{}];
} inGroup:group];

[queueA asynchronouslyDispatchBlock:^{
    [queueC synchronouslyDispatchBlock:^{}];
} inGroup:group];

[queueC asynchronouslyDispatchBlock:^{
    [queueB synchronouslyDispatchBlock:^{}];
} inGroup:group];

[group waitForCompletionWithTimeout:4.0 useWallTime:YES];

[PWDispatchQueueGraph.sharedGraph checkTreeStructure];
```

```
[queueA asynchronouslyDispatchBlock:^{
    [queueB synchronouslyDispatchBlock:^{}];
} inGroup:group];

[queueA asynchronouslyDispatchBlock:^{
    [queueC synchronouslyDispatchBlock:^{}];
} inGroup:group];

[queueC asynchronouslyDispatchBlock:^{
    [queueB synchronouslyDispatchBlock:^{}];
} inGroup:group];

[group waitForCompletionWithTimeout:4.0 useWallTime:YES];

[queueB asynchronouslyDispatchBlock:^{
     [queueA synchronouslyDispatchBlock:^{}];
} inGroup:group];

[group waitForCompletionWithTimeout:4.0 useWallTime:YES];

[PWDispatchQueueGraph.sharedGraph checkTreeStructure];
```

Synchronous dispatch cycle: A -> B -> A

A -> B:

```
    (
        "0   PWFoundation                        0x0000000010674b409 -[PWDispatchQueueGraph
addSynchronousDispatchFromQueue:toQueue:] + 1225",
        "1   PWFoundation                        0x0000000010674aef7 -[PWDispatchQueueGraph
addSynchronousDispatchToQueue:] + 423",
        "2   PWFoundation                        0x0000000010684ad14 doSyncDispatch + 1348",
        "3   libdispatch.dylib                   0x00007fff9e0af40b _dispatch_client_callout + 8",
        "4   libdispatch.dylib                   0x00007fff9e0b09f2 _dispatch_barrier_sync_f_invoke + 74",
        "5   PWFoundation                        0x0000000010684a7ba -[PWSerialDispatchQueue
synchronouslyDispatchBlock:] + 394",
        "6   PWFoundationTests                   0x0000000105af7750 __40-[PWDispatchTest
testDispatchSemaphore2]_block_invoke + 48",
        "7   PWFoundation                        0x0000000106849a8d doAsyncSerialDispatch + 1373",
        "8   libdispatch.dylib                   0x00007fff9e0af40b _dispatch_client_callout + 8",
        "9   libdispatch.dylib                   0x00007fff9e0b403b _dispatch_queue_drain + 754",
        "10  libdispatch.dylib                   0x00007fff9e0ba707 _dispatch_queue_invoke + 549",
        "11  libdispatch.dylib                   0x00007fff9e0af40b _dispatch_client_callout + 8",
        "12  libdispatch.dylib                   0x00007fff9e0b329b _dispatch_root_queue_drain + 1890",
        "13  libdispatch.dylib                   0x00007fff9e0b2b00 _dispatch_worker_thread3 + 91",
        "14  libsystem_pthread.dylib             0x00007fff905444de _pthread_wqthread + 1129",
        "15  libsystem_pthread.dylib             0x00007fff90542341 start_wqthread + 13"
    )
```

Synchronous dispatch cycle: A -> B -> A

B -> A:

```
   (
        "0   PWFoundation                        0x0000000106749409 -[PWDispatchQueueGraph
addSynchronousDispatchFromQueue:toQueue:] + 1225",
        "1   PWFoundation                        0x0000000106749ef7 -[PWDispatchQueueGraph
addSynchronousDispatchToQueue:] + 423",
        "2   PWFoundation                        0x0000000106781d14 doSyncDispatch + 1348",
        "3   libdispatch.dylib                    0x00007fff9e0af40b _dispatch_client_callout + 8",
        "4   libdispatch.dylib                    0x00007fff9e0b09f2 _dispatch_barrier_sync_f_invoke + 74",
        "5   PWFoundation                        0x0000000106781ba -[PWSerialDispatchQueue
synchronouslyDispatchBlock:] + 394",
        "6   PWFoundationTests                   0x0000000105af7990 __40-[PWDispatchTest
testDispatchSemaphore2]_block_invoke.644 + 48",
        "7   PWFoundation                        0x0000000106849a8d doAsyncSerialDispatch + 1373",
        "8   libdispatch.dylib                    0x00007fff9e0af40b _dispatch_client_callout + 8",
        "9   libdispatch.dylib                    0x00007fff9e0b403b _dispatch_queue_drain + 754",
        "10  libdispatch.dylib                    0x00007fff9e0ba707 _dispatch_queue_invoke + 549",
        "11  libdispatch.dylib                    0x00007fff9e0af40b _dispatch_client_callout + 8",
        "12  libdispatch.dylib                    0x00007fff9e0b329b _dispatch_root_queue_drain + 1890",
        "13  libdispatch.dylib                    0x00007fff9e0b2b00 _dispatch_worker_thread3 + 91",
        "14  libsystem_pthread.dylib              0x00007fff905444de _pthread_wqthread + 1129",
        "15  libsystem_pthread.dylib              0x00007fff90542341 start_wqthread + 13“
  )
```

Dispatch-Queue-Graph

- Standardmäßig an während aller Unit-Tests
  - -DebugOption PWDispatchQueueGraphStateOption 3
- Im tear down aller test cases:
  `[PWDispatchQueueGraph.sharedGraph checkTreeStructure]`
- Schlägt regelmäßig zu während der Entwicklung

Leak Checker

Leak Checker

„ARC is a giant leak-producing machinery.“

Apple-Entwickler bei der WWDC 2011

Leak Checker

- Zyklen sind seit Xcode 8 mit dem Memory Graph leichter zu finden.
  - Sie rutschen aber trotzdem sehr leicht wieder hinein
- Ziel: Wichtige Leaks automatisch in den Unit-Tests detektieren.

Leak Checker

- Unsere pragmatische Lösung: Zählen der Instanzen der wichtigsten Klassen:
  - Window controllers
  - View controllers
  - Views
  - Persistent Store Coordinators
  - Managed Object Contexts / Managed Objects

Leak Checker

- Am Ende eines Tests sollten alle Instanzen dieser Klassen üblicherweise wieder weg sein.
  - Ausnahmen sollten möglich sein.

```
@implementation PWViewController
#ifndef NDEBUG
- (nullable instancetype)initWithNibName:(nullable NSString*)nibNameOrNil
                                  bundle:(nullable NSBundle*)nibBundleOrNil {
    if(self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil])
        [PWLeakChecker.sharedLeakChecker addLivingInstance:self];
    return self;
}

- (instancetype)init {
    [PWLeakChecker.sharedLeakChecker addLivingInstance:self];
    return [super init];
}

- (nullable instancetype)initWithCoder:(NSCoder*)coder {
    [PWLeakChecker.sharedLeakChecker addLivingInstance:self];
    return [super initWithCoder:coder];
}

- (void) dealloc {
    [PWLeakChecker.sharedLeakChecker removeLivingInstance:self];
}
#endif
```

Leak Checker

```
@interface PWTestCase : XCTestCase
@end

@implementation PWTestCase

- (void) invokeTest
{
    @autoreleasepool {
        [super invokeTest];
    }
    XCTAssertTrue([PWLeakChecker.sharedLeakChecker
                                        checkLivingInstances]);
}

@end
```

Leak Checker

```
Test Case '-[PWOutlineViewControllerTest testView]' started.
Leaked <PWTestOutlineViewController: 0x1006447e0> (0x1006447e0) with 3 retains
Leaked 6 instances of PWView
Leaked 1 instances of PWManagedObjectContext
Leaked 1 instances of PWManagedObject
Leaked 1 instances of PWPersistentStoreCoordinator
PWOutlineViewControllerTest.m:1009: error: -[PWOutlineViewControllerTest testView]:
(([PWLeakChecker.sharedLeakChecker checkLivingInstances]) is true) failed
Test Case '-[PWOutlineViewControllerTest testView]' failed (10.404 seconds).
```

Leak Checker

- Besonderheiten
  - CoreData gibt Objekte unter Umständen auf einem anderen Thread frei.
  - Timer in AppKit/UIKit halten Objekte fest.

Leak Checker

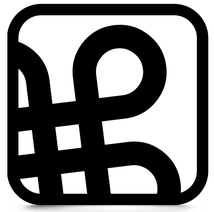
- Schlägt bei uns sehr häufig zu.
  - Ein Hälfte sind Testartefakte, keine echten Leaks.
  - Die andere Hälfte ist echt.

Hinweis von Euch ans Publikum:

Bitte die Hand zu heben, damit Mädels mit den Micros kommen können.

Fragen?

Vielen Dank



Macoun