ReactiveCocoa入门到实战

第四周 冷热信号 & RAC并发编程

内容大纲

- 冷信号与热信号
- RAC并发编程

冷信号与热信号

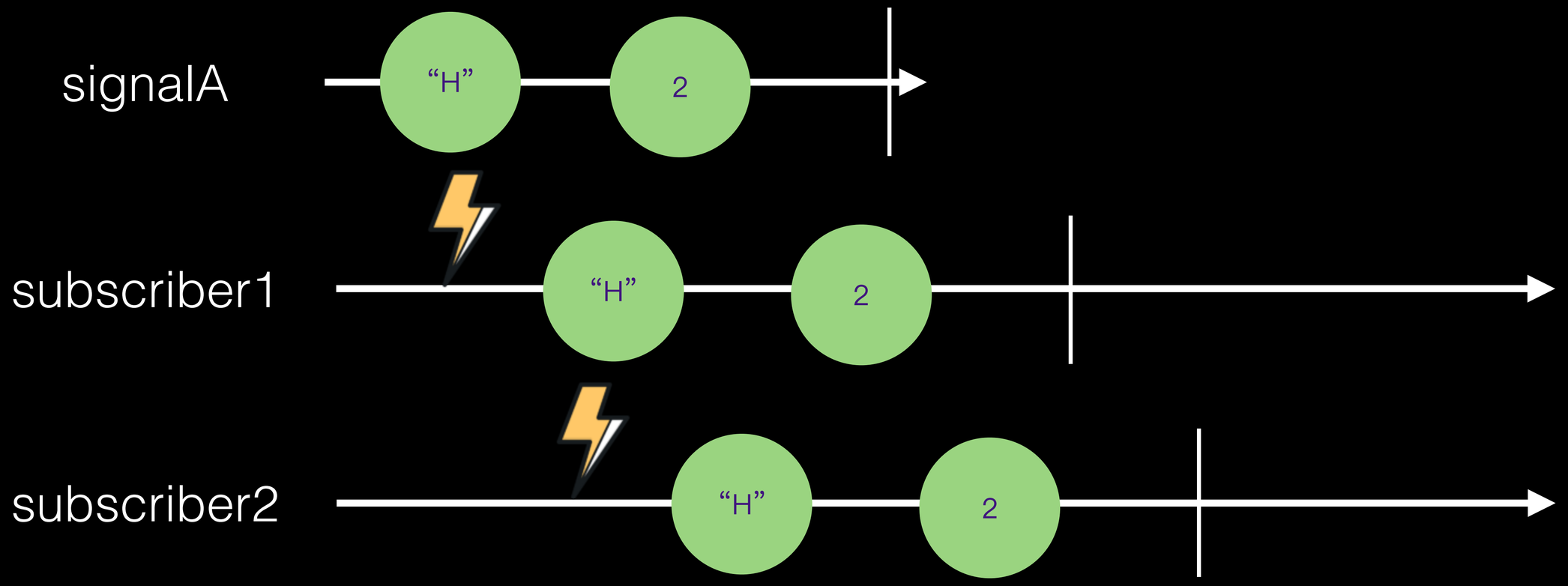
冷信号与热信号

- 什么是冷信号与热信号?
- Signal vs Subject
- 冷信号 -> 热信号

冷信号与热信号

什么是冷信号与热信号

- 当Signal有多个订阅者



冷信号与热信号

什么是冷信号与热信号

- 转换的本质

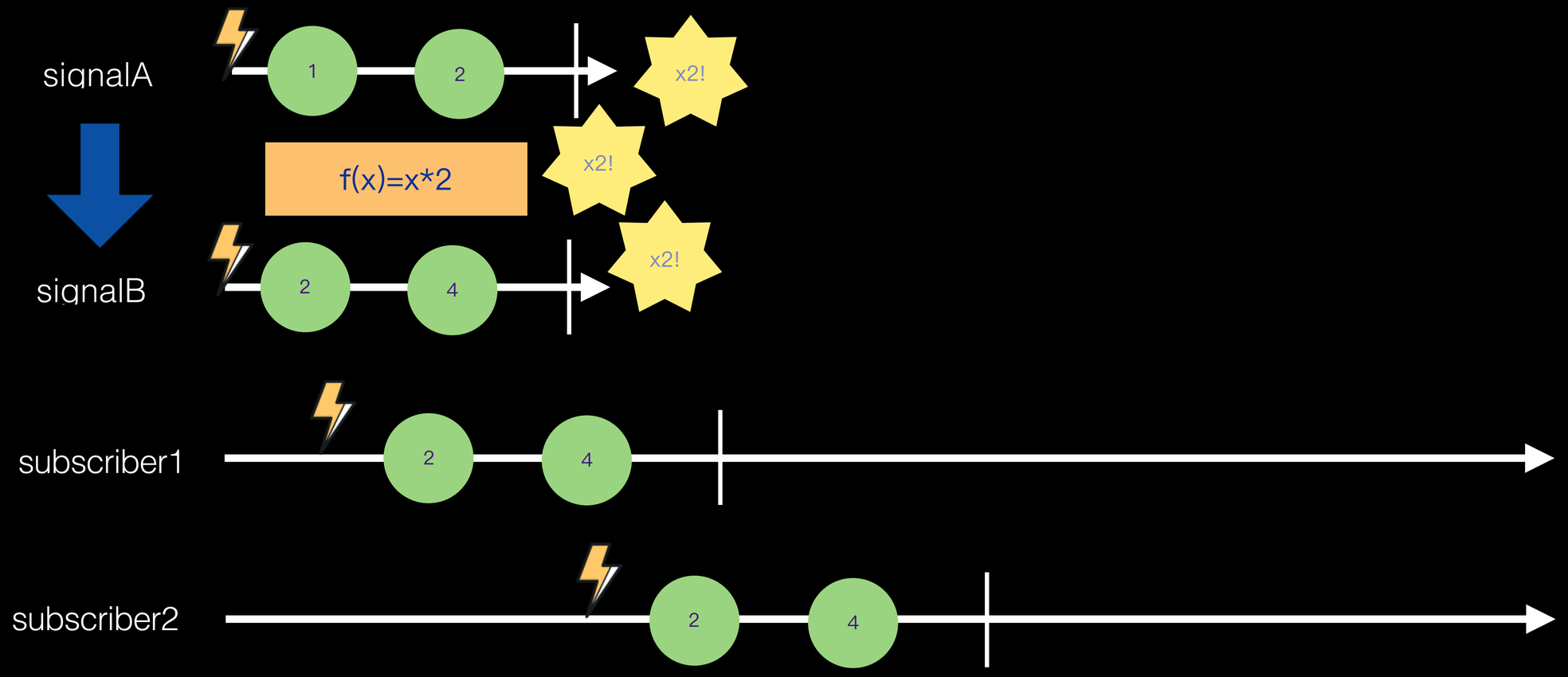
```
- (RACSignal *)bind:(RACStreamBindBlock (^)(void))block;
{
    return [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        RACStreamBindBlock bindBlock = block();

        [self subscribeNext:^(id x) {
            BOOL stop = NO;
            RACSignal *signal = (RACSignal *)bindBlock(x, &stop);
            if (signal == nil || stop) { [subscriber sendCompleted];
            } else {
                [signal subscribeNext:^(id x) { [subscriber sendNext:x];
                } error:^(NSError *error) { [subscriber sendError:error];
                } completed:^{ }];
            }
        } error:^(NSError *error) { [subscriber sendError:error];
        } completed:^{ [subscriber sendCompleted]; }];
        return nil;
    }];
}
```

冷信号与热信号

什么是冷信号与热信号

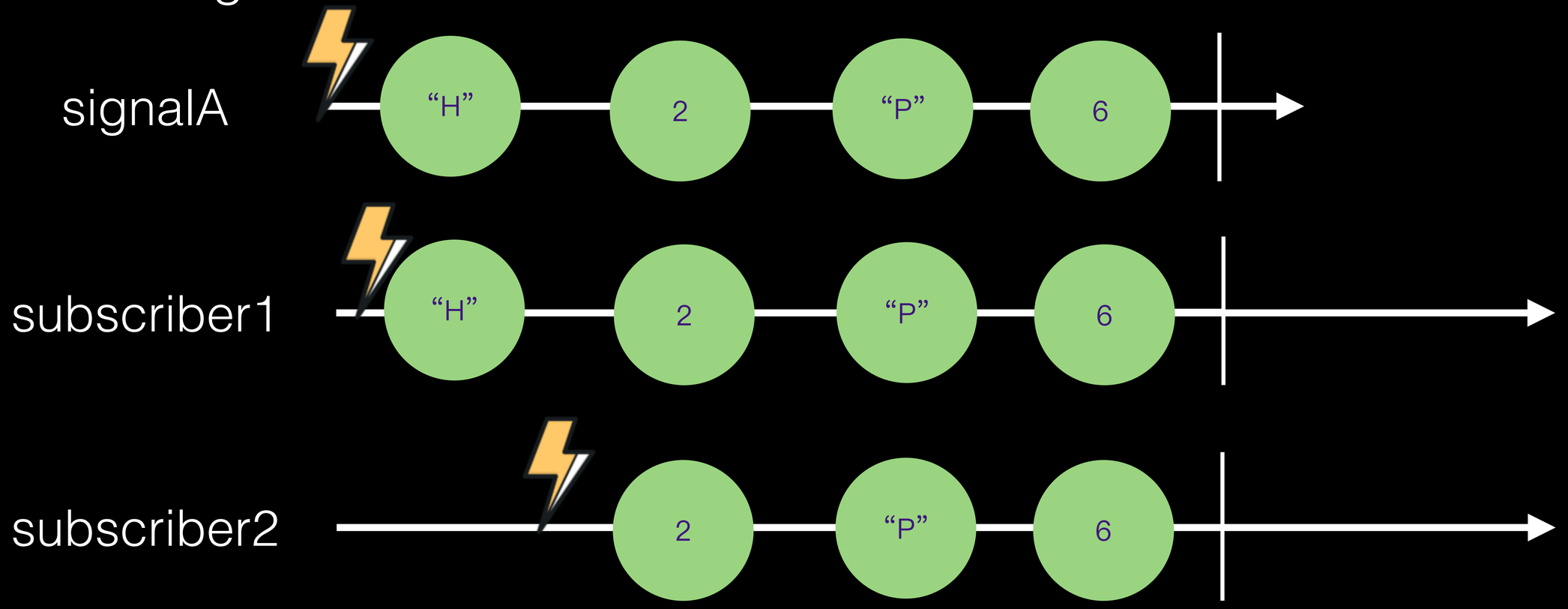
- Signal变换后多个订阅者



冷信号与热信号

什么是冷信号与热信号

- Signal订阅者共享



冷信号与热信号

什么是冷信号与热信号

- 看点播 vs 看直播
- 冷信号 剧本，订阅 舞台剧
- 热信号 舞台剧，订阅 观看舞台剧

冷信号与热信号

热信号在哪里?

- RACSubject

```
RACSubject *subject = [RACSubject subject];

[subject subscribeNext:^(id x) {
    // a
} error:^(NSError *error) {
    // b
} completed:^{
    // c
}];

[subject sendNext:@1];
[subject sendNext:@2];
[subject sendNext:@3];

[subject sendCompleted];
```

冷信号与热信号

RACSubject和它的子类

- RACReplaySubject
- 带快速回播的Subject
- 控制“历史值”的数量

```
RACReplaySubject *subject = [RACReplaySubject
                              replaySubjectWithCapacity:1];
[subject sendNext:@1];
[subject sendNext:@2];
[subject sendCompleted];

[subject subscribeNext:^(id x) {  /* a*/  }];
```

冷信号与热信号

RACSignal vs RACSubject

- 冷信号 vs 热信号
- 确定的未来 vs 不确定的未来
- 无视订阅者 vs 关心订阅者

冷信号与热信号

RACSubject的建议

- 等同于变量，有很强的使用吸引力
- 可用做全局通知
- 尽量少用，用在热信号的场景上

冷信号与热信号

冷信号 -> 热信号

- 一个人看视频 -> 叫上大家一起看
- 视频文件（剧本）-> 播放器 + 显示屏 + 音响 -> 一堆观众
- RACSignal -> RACSubject -> Subscribers

冷信号与热信号

冷信号 -> 热信号

```
RACSignal *signal = @[@1, @2, @3, @4].rac_sequence.signal;
RACSignal *signalB = [[signal map:^id(id value) {
    return [[RACSignal return:value] delay:1];
}] concat];

RACSubject *speaker = [RACSubject subject];
[signalB subscribe:speaker];

[speaker subscribeNext:^(id x) {  // a }];

[speaker subscribeNext:^(id x) {  // b }];

[speaker subscribeNext:^(id x) {  // c }];
```

冷信号与热信号

冷信号 -> 热信号 官方方案

```
- (RACMulticastConnection *)publish;
- (RACMulticastConnection *)multicast:(RACSubject *)subject;
- (RACSignal *)replay;
- (RACSignal *)replayLast;
- (RACSignal *)replayLazily;
```

直到现在，RAC信号操作还缺什么？

冷信号与热信号

冷信号+副作用方法

```
+ (RACSignal *)defer:(RACSignal * (^)(void))block;
- (RACSignal *)then:(RACSignal * (^)(void))block;
```

冷信号与热信号

不建议使用的同步方法

```
- (id)first;
- (id)firstOrDefault:(id)defaultValue;
- (id)firstOrDefault:(id)defaultValue
              success:(BOOL *)success
                error:(NSError **)error;
- (BOOL)waitUntilCompleted:(NSError **)error;
- (NSArray *)toArray;
@property (nonatomic, strong, readonly)
                RACSequence *sequence;
```

冷信号与热信号

留给大家自己研究的

```
- (RACSignal *)groupBy:(id<NSCopying> (^)(id object))keyBlock;
- (RACSignal *)groupBy:(id<NSCopying> (^)(id object))keyBlock
                transform:(id (^)(id object))transformBlock;

RACSignal *signal = @[@1, @2, @3, @4].rac_sequence.signal;

RACSignal *signalGroup = [signal groupBy:^NSString *(NSNumber *object) {
    return object.integerValue % 2 == 0 ? @"odd" : @"even";
}];

[[[signalGroup take:1] flatten] subscribeNext:^(id x) {
    NSLog(@"next: %@", x);
}];
```

RAC并发编程

RAC并发编程

- 同步和异步
- 并行和并发
- RACScheduler
- Signal遇上并发

同步和异步

RAC并发编程

同步

- 函数调用，不返回结果不进行下一步
- 书写顺序 == 执行顺序
- 阻塞IO

RAC并发编程

异步

- 函数调用，直接进行下一步
- 通过回调函数返回结果
- 书写顺序 != 执行顺序
- 非阻塞IO
- RAC整个是个异步库

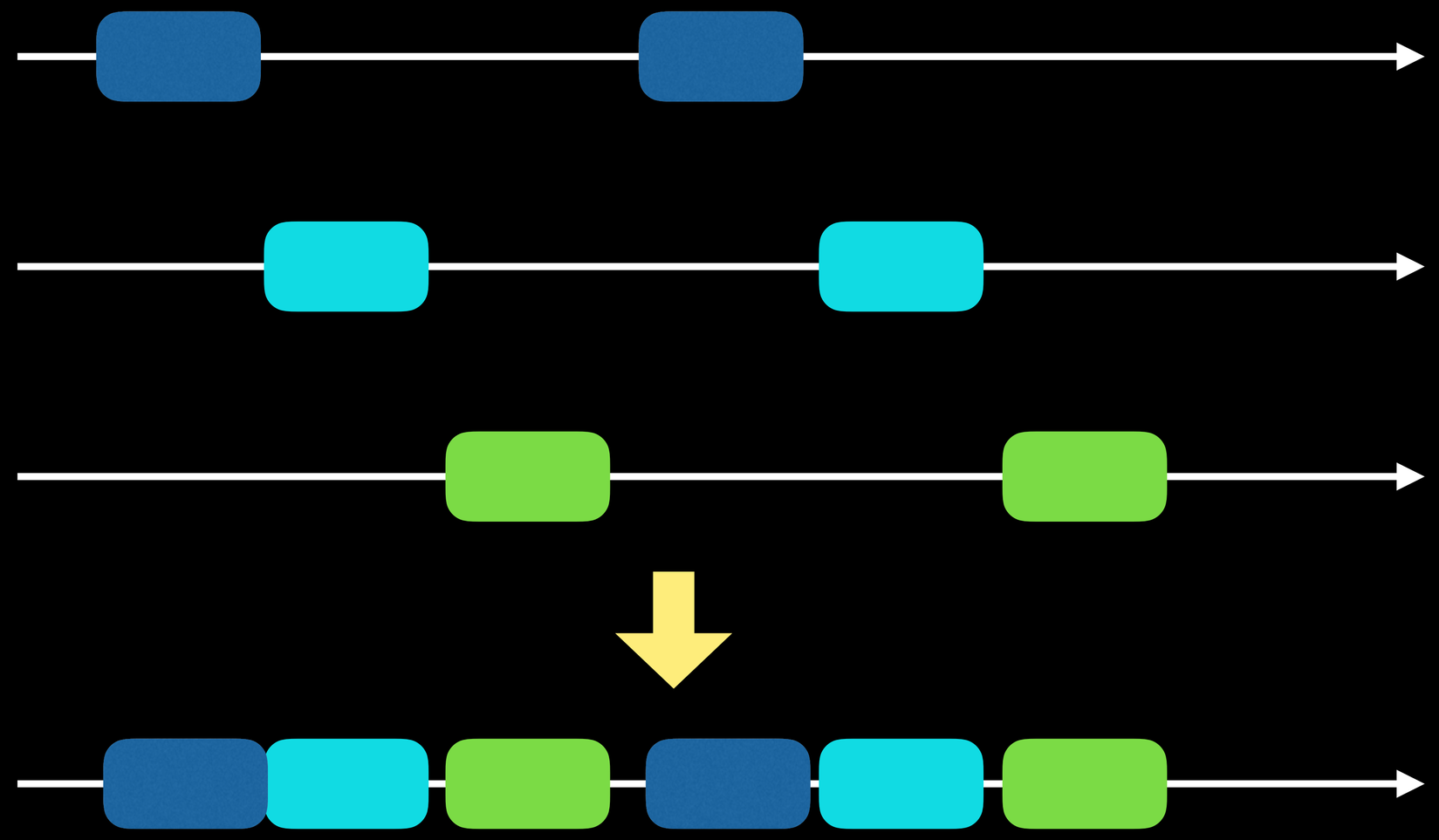
RAC并发编程

并发

- 在一个物理计算核心
- 通过调度手段兼顾多个任务
- 使任务看似一起执行
- “美女餐厅”游戏

RAC并发编程

并发



RAC并发编程

并行

- 在多个物理计算核心
- 通过分配手段处理多个任务
- 使任务一起执行
- 多个服务员的餐厅

RAC并发编程

并行

RAC并发编程

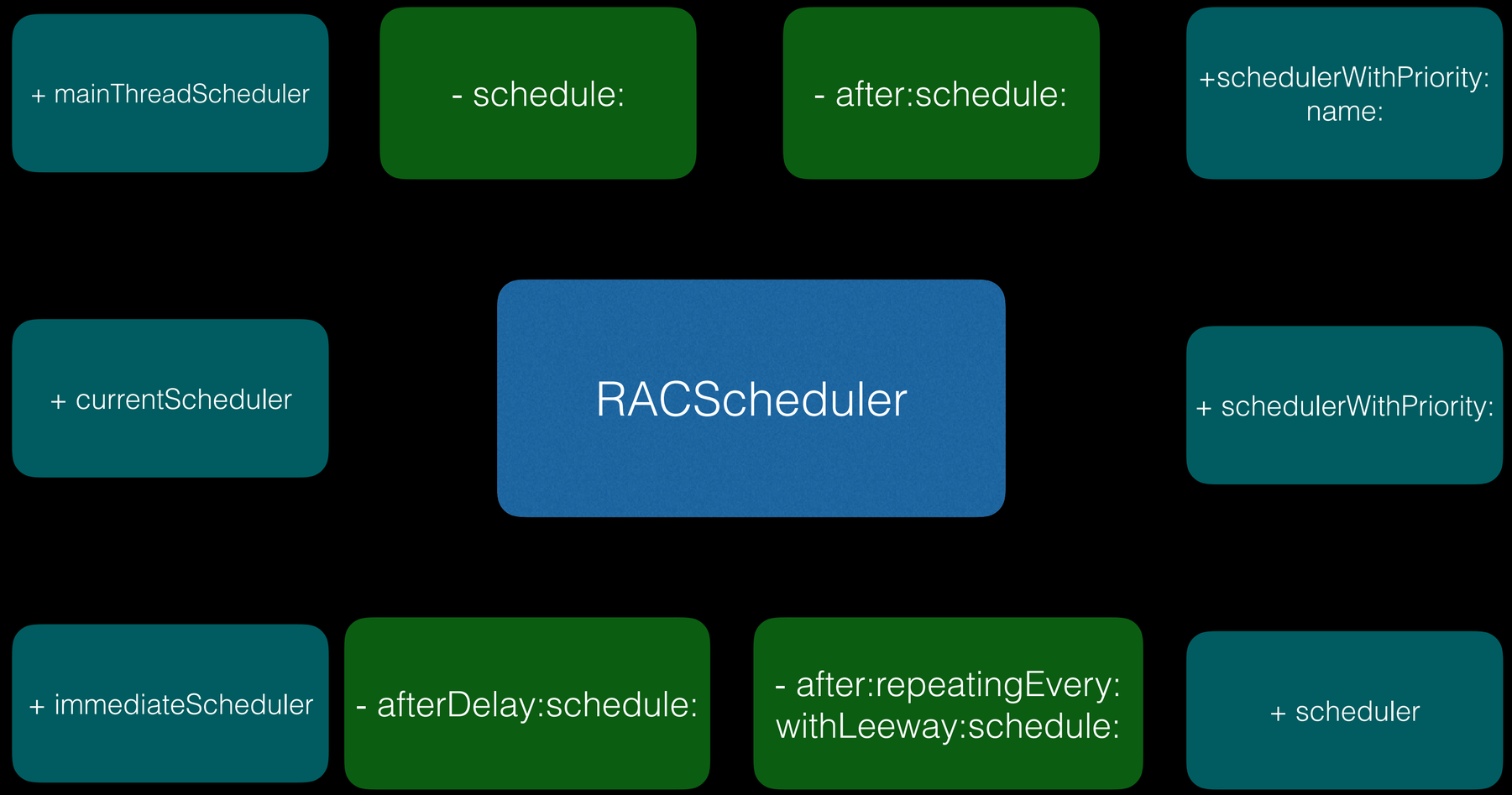
如何在RAC中并发编程?

RACScheduler

“Schedulers are used to control when and where work is performed…”

RAC并发编程

RACScheduler一览



RAC并发编程

RACScheduler示例

```
// 主线程的Scheduler
RACScheduler *mainScheduler = [RACScheduler mainThreadScheduler];

// 子线程的两个Scheduler，注意[RACScheduler scheduler]是返回一个新的
RACScheduler *scheduler1 = [RACScheduler scheduler];
RACScheduler *scheduler2 = [RACScheduler scheduler];

// 返回当前的Scheduler，自定义线程会返回nil
RACScheduler *scheduler3 = [RACScheduler currentScheduler];

// 创建某优先级Scheduler，不建议除非你知道你在干神马
RACScheduler *scheduler4 = [RACScheduler schedulerWithPriority:RACSchedulerPriorityHigh];
RACScheduler *scheduler5 = [RACScheduler schedulerWithPriority:RACSchedulerPriorityHigh
                                                        name:@"someName"];

// 创建立即Scheduler，不建议除非你知道你在干神马
RACScheduler *scheduler6 = [RACScheduler immediateScheduler];
```

RAC并发编程

RACScheduler示例

```
// 分派一个任务，[disposable dispose]用来取消
RACDisposable *disposable = [mainScheduler schedule:^{ /* 这里是个任务 */ }];
[disposable dispose];
// 定时任务
NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
formatter.dateFormat = @"yyyy-MM-dd HH:mm:ss";
NSDate *date = [formatter dateFromString:@"2016-07-20 21:00:00"];
[scheduler1 after:date schedule:^{ /* 将在2016-07-20 21:00:00执行 */ }];
// 延时任务
[scheduler2 afterDelay:30 schedule:^{ /* 将在30秒后执行 */ }];
// 循环任务
[scheduler3 after:[NSDate date] repeatingEvery:1 withLeeway:0.1 schedule:^{
    // 从现在开始，每1秒执行一次，最长不能操作1.1秒执行下一次
}];
```

RAC并发编程

RACScheduler vs GCD

- Scheduler使用GCD来实现
- 可以“取消”
- 与RAC其他组件高度整合
- 一个Scheduler保证串行执行
- 一个Scheduler的任务不保证线程是同一个

RAC并发编程

当Signal遇上并发

- 重温无并发的情况
- 异步订阅
- 异步发送
- 排列组合一下
- 解决之道

RAC并发编程

重温订阅顺序

```
NSLog(@"start test");
RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
    NSLog(@"sendNext:@1");
    [subscriber sendNext:@1];
    NSLog(@"sendNext:@2");
    [subscriber sendNext:@2];
    NSLog(@"sendCompleted");
    [subscriber sendCompleted];
    NSLog(@"return nil");
    return nil;
}];
NSLog(@"signal was created");
[signal subscribeNext:^(id x) {
    NSLog(@"receive next:%@", x);
} error:^(NSError *error) {
    NSLog(@"receive error:%@", error);
} completed:^{
    NSLog(@"receive complete");
}];
NSLog(@"subscribing finished");
```

```
start test
signal was created
sendNext:@1
receive next:1
sendNext:@2
receive next:2
sendCompleted
receive complete
return nil
subscribing finished
```

情形之一
异步订阅

RAC并发编程

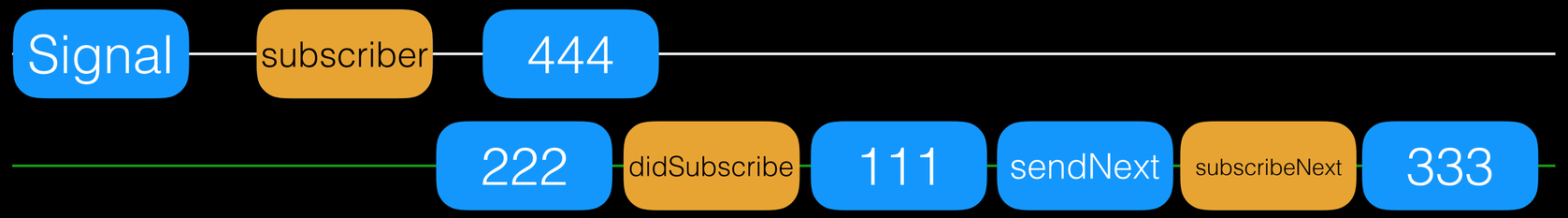
异步订阅

```
void subscribeAsync()
{
    RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        NSLog(@"111");
        [subscriber sendNext:@1];
        [subscriber sendCompleted];

        return nil;
    }];

    [[RACScheduler scheduler] schedule:^{
        NSLog(@"222");
        [signal subscribeNext:^(id x) {
            NSLog(@"333");
        }];
    }];

    NSLog(@"444");
}
```



情景之二
异步发送

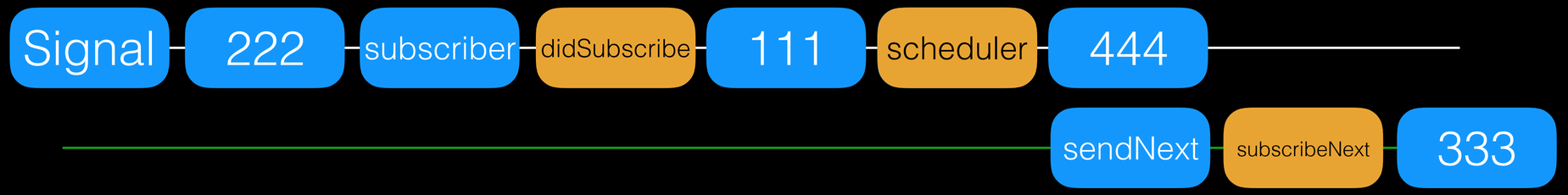
RAC并发编程

异步发送

```
void sendAsync()
{
    RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        NSLog(@"111");
        RACDisposable *disposable = [[RACScheduler scheduler] schedule:^{
            [subscriber sendNext:@1];
            [subscriber sendCompleted];
        }];
        return disposable;
    }];

    NSLog(@"222");
    [signal subscribeNext:^(id x) {
        NSLog(@"333");
    }];

    NSLog(@"444");
}
```

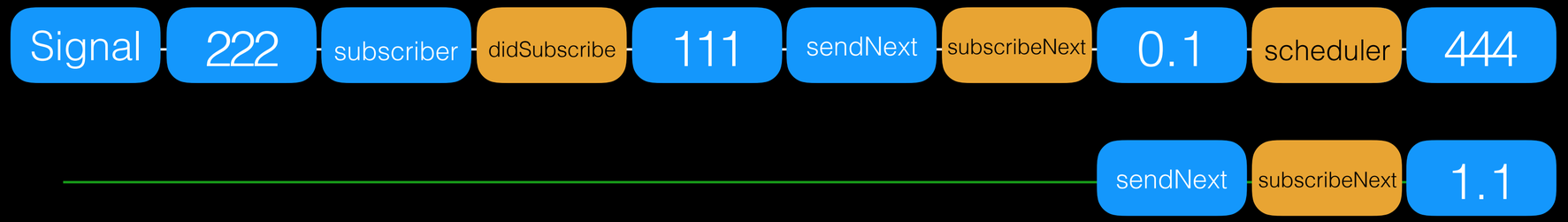


情景之三
同步+异步发送

RAC并发编程

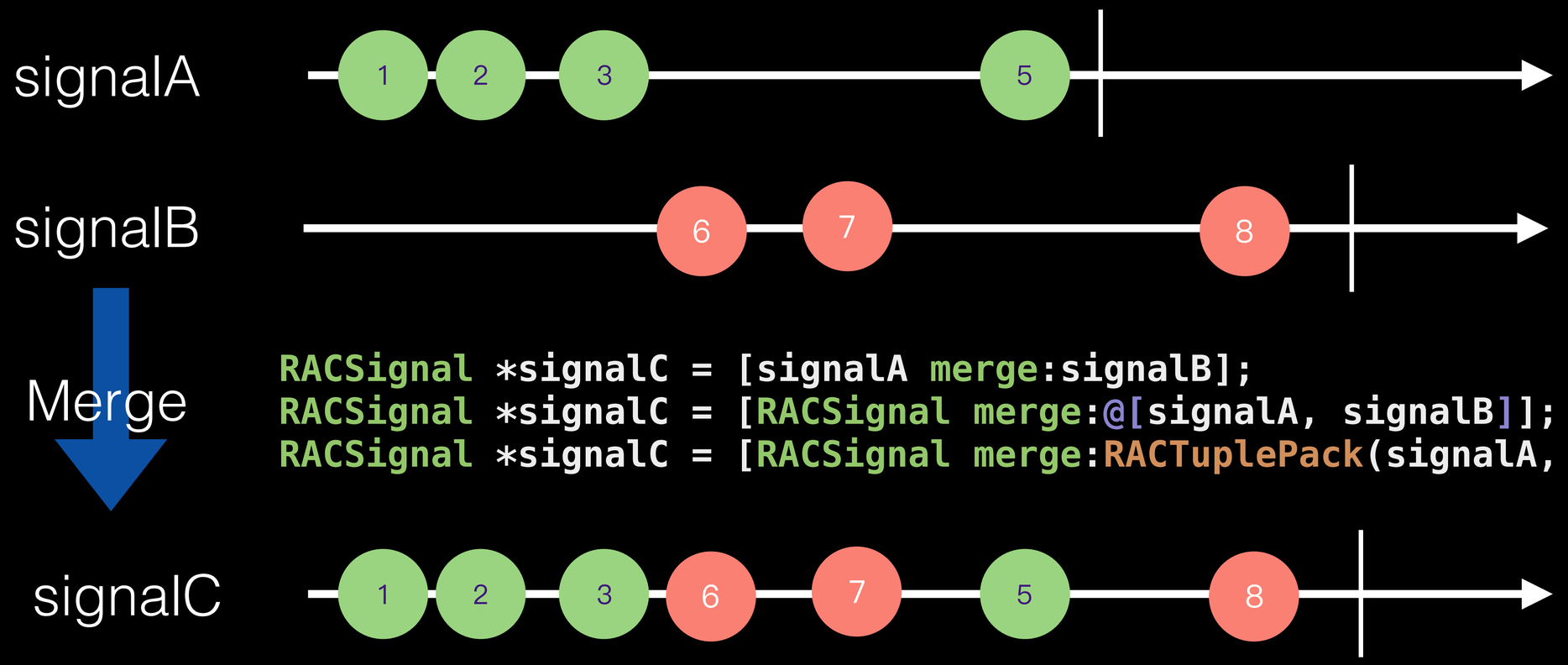
发送在不同的Scheduler

```
void sendEverywhere()
{
    RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        NSLog(@"111");
        [subscriber sendNext:@0.1];
        RACDisposable *disposable = [[RACScheduler scheduler] schedule:^{
            [subscriber sendNext:@1.1];
            [subscriber sendCompleted];
        }];
        return disposable;
    }];
    NSLog(@"222");
    [signal subscribeNext:^(id x) {
        NSLog(@"%@", x);
    }];
    NSLog(@"444");
}
```



RAC并发编程

回忆下Merge操作




```
RACSignal *signalC = [signalA merge:signalB];
RACSignal *signalC = [RACSignal merge:@[signalA, signalB]];
RACSignal *signalC = [RACSignal merge:RACTuplePack(signalA, signalB)];
```

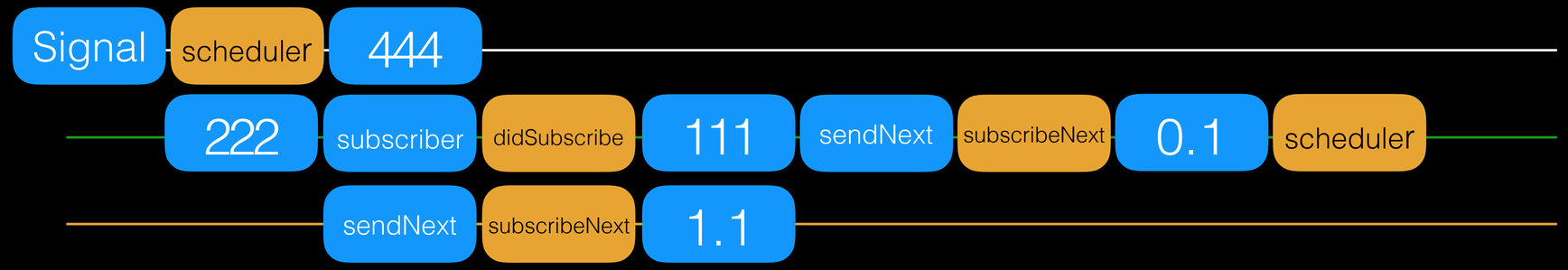
情景之四

异步订阅+异步发送

RAC并发编程

订阅和发送都在不同的Scheduler

```
void sendAndSubscribeEverywhere()
{
    RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        NSLog(@"111");
        [subscriber sendNext:@0.1];
        RACDisposable *disposable = [[RACScheduler scheduler] schedule:^{
            [subscriber sendNext:@1.1];
            [subscriber sendCompleted];
        }];
        return disposable;
    }];
    [[RACScheduler scheduler] schedule:^{
        NSLog(@"222");
        [signal subscribeNext:^(id x) {
            NSLog(@"%@", x);
        }];
    }];
    NSLog(@"444");
}
```



回忆一下之前我们的一些信号操作……

RAC并发编程

问题

- 订阅时机不确定
- 发送时机不确定

RAC并发编程

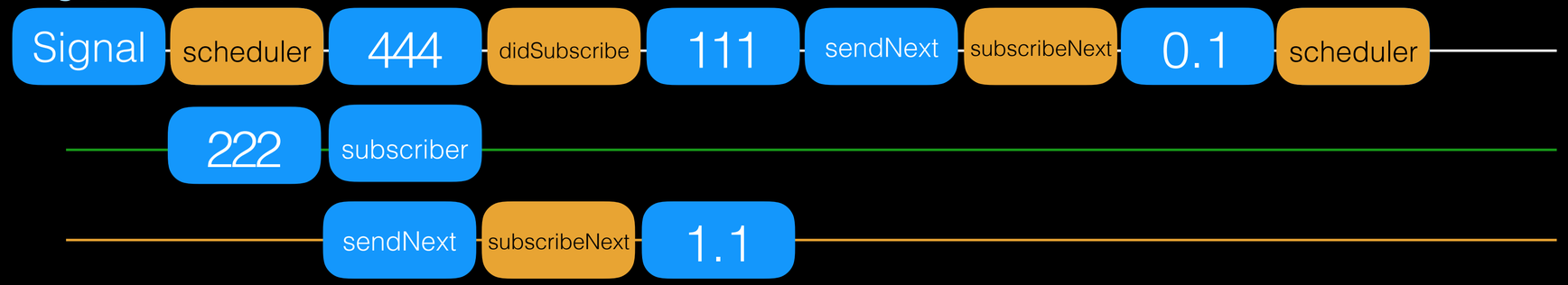
解决之道

- 订阅时机不确定 —> subscribeOn:
- 发送时机不确定 —> deliverOn:

RAC并发编程

使用subscribeOn:

```
void useSubscribeOn()
{
    RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        NSLog(@"111");
        [subscriber sendNext:@0.1];
        RACDisposable *disposable = [[RACScheduler scheduler] schedule:^{
            [subscriber sendNext:@1.1];
            [subscriber sendCompleted];
        }];
        return disposable;
    }];
    [[RACScheduler scheduler] schedule:^{
        NSLog(@"222");
        [[signal subscribeOn:[RACScheduler mainThreadScheduler]] subscribeNext:^(id x) {
            NSLog(@"%@", x);
        }];
    }];
    NSLog(@"444");
}
```



RAC并发编程

subscribeOn:总结

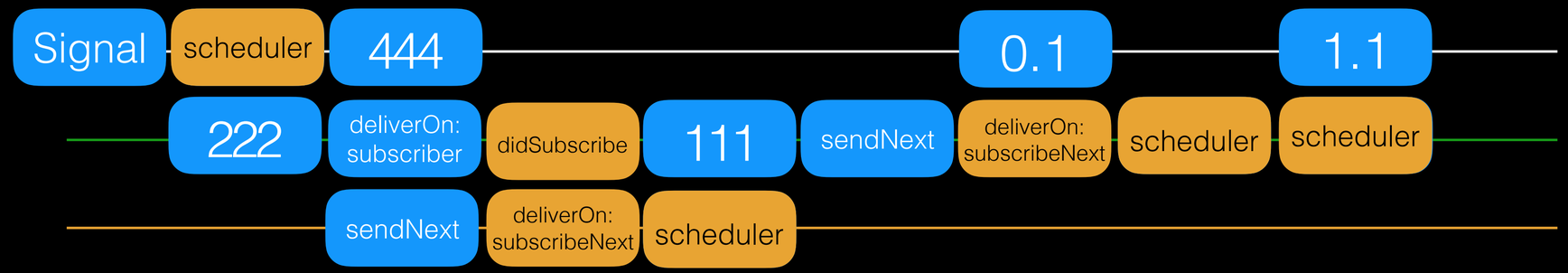
- 能够保证didSubscribe block在指定的scheduler
- 不能保证sendNext、error和complete在哪个scheduler
- 头文件描述

```
/// Use of this operator should be avoided whenever possible, because the
/// receiver's side effects may not be safe to run on another thread. If you just
/// want to receive the signal's events on `scheduler`, use -deliverOn: instead.
- (RACSignal *)subscribeOn:(RACScheduler *)scheduler;
```

RAC并发编程

使用deliverOn:

```
void useDeliverOn()
{
    RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        NSLog(@"111");
        [subscriber sendNext:@0.1];
        RACDisposable *disposable = [[RACScheduler scheduler] schedule:^{
            [subscriber sendNext:@1.1];
            [subscriber sendCompleted];
        }];
        return disposable;
    }];
    [[RACScheduler scheduler] schedule:^{
        NSLog(@"222");
        [[signal deliverOn:[RACScheduler mainThreadScheduler]] subscribeNext:^(id x) {
            NSLog(@"%@", x);
        }];
    }];
    NSLog(@"444");
}
```



RAC并发编程

subscribeOn:的用武之地

```
void whenShouldWeUseSubscribeOn()
{
    UIView *view = [[UIView alloc] init];
    RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {

        UILabel *label = [[UILabel alloc] init];
        label.text = @"Hello world";
        [view addSubview:label];

        [subscriber sendNext:@0.1];
        RACDisposable *disposable = [[RACScheduler scheduler] schedule:^{
            [subscriber sendNext:@1.1];
            [subscriber sendCompleted];
        }];
        return disposable;
    }];
    [[RACScheduler scheduler] schedule:^{
        [[signal subscribeOn:[RACScheduler mainThreadScheduler]] subscribeNext:^(id x) {
            NSLog(@"%@", x);
        }];
    }];
}
```

RAC操作总结

- 单个信号转换
- 多个信号组合
- 高阶信号操作
- 冷热信号操作
- 并发操作

RAC还漏掉什么？

RACDisposable

to be continue…