ReactiveCocoa入门到实战

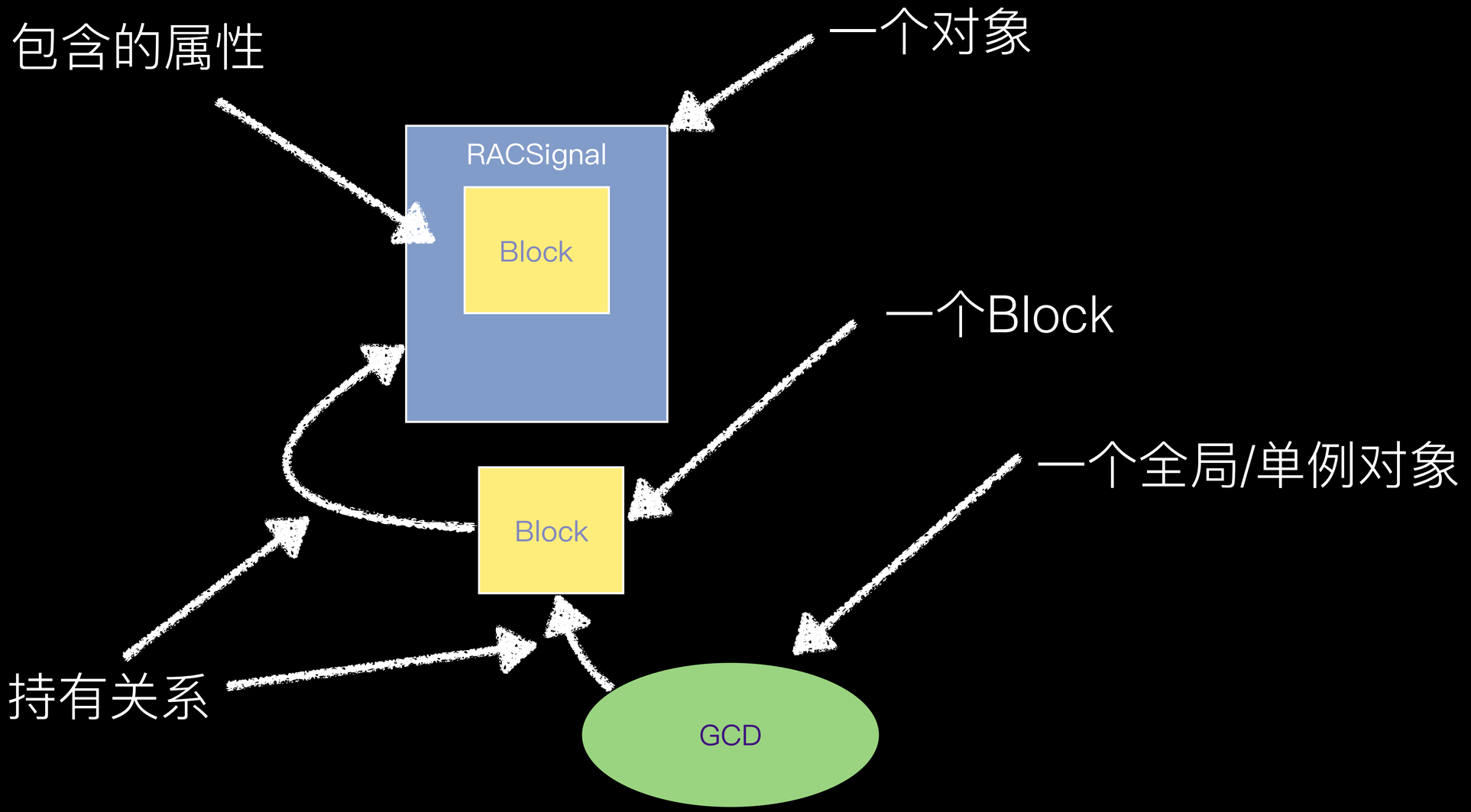
第五周 生命周期详解

内容大纲

- 信号的生命周期
- 延时情况下的生命周期
- 变换组合时的生命周期
- RACDisposable对象

信号的生命周期

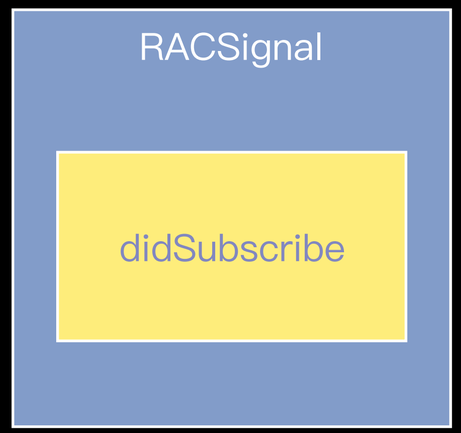
图例

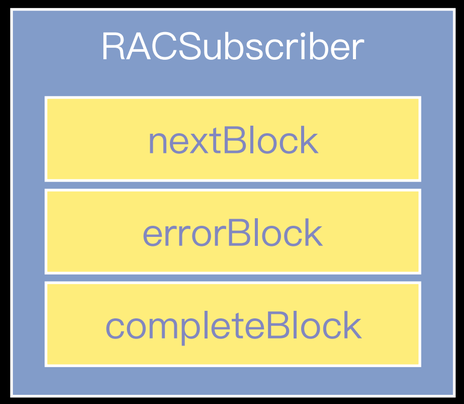

包含的属性
一个对象
RACSignal
Block
一个Block
Block
一个全局/单例对象
持有关系
GCD

信号的生命周期

冷信号的实例状态

```
RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
    [subscriber sendNext:@1];
    [subscriber sendNext:@2];
    [subscriber sendCompleted];
    return nil;
}];

[signal subscribeNext:^(id x) {
    NSLog(@"next: %@", x);
} error:^(NSError *error) {
    NSLog(@"error: %@", error);
} completed:^{
    NSLog(@"complete");
}];
```

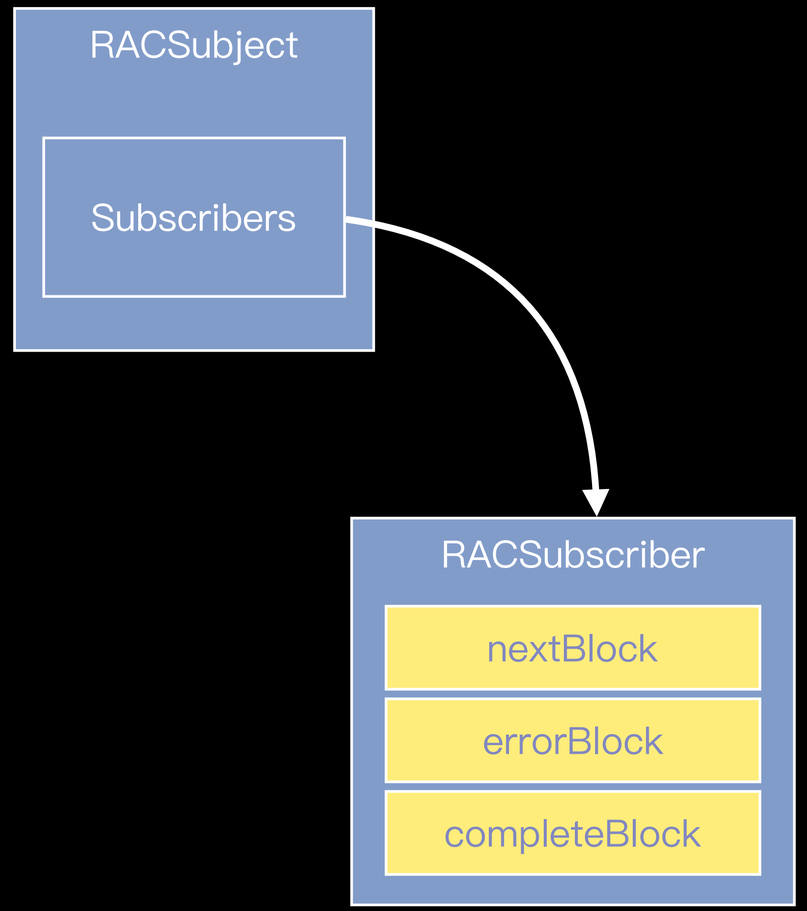




信号的生命周期

热信号的实例状态

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

[subject subscribeNext:^(id x) {
    NSLog(@"Subscriber 1 receive next: %@", x);
} error:^(NSError *error) {
    NSLog(@"Subscriber 1 receive error: %@", error);
} completed:^{
    NSLog(@"Subscriber 1 receive complete");
}];

[subject sendNext:@1];
[subject sendNext:@2];
[subject sendCompleted];
```

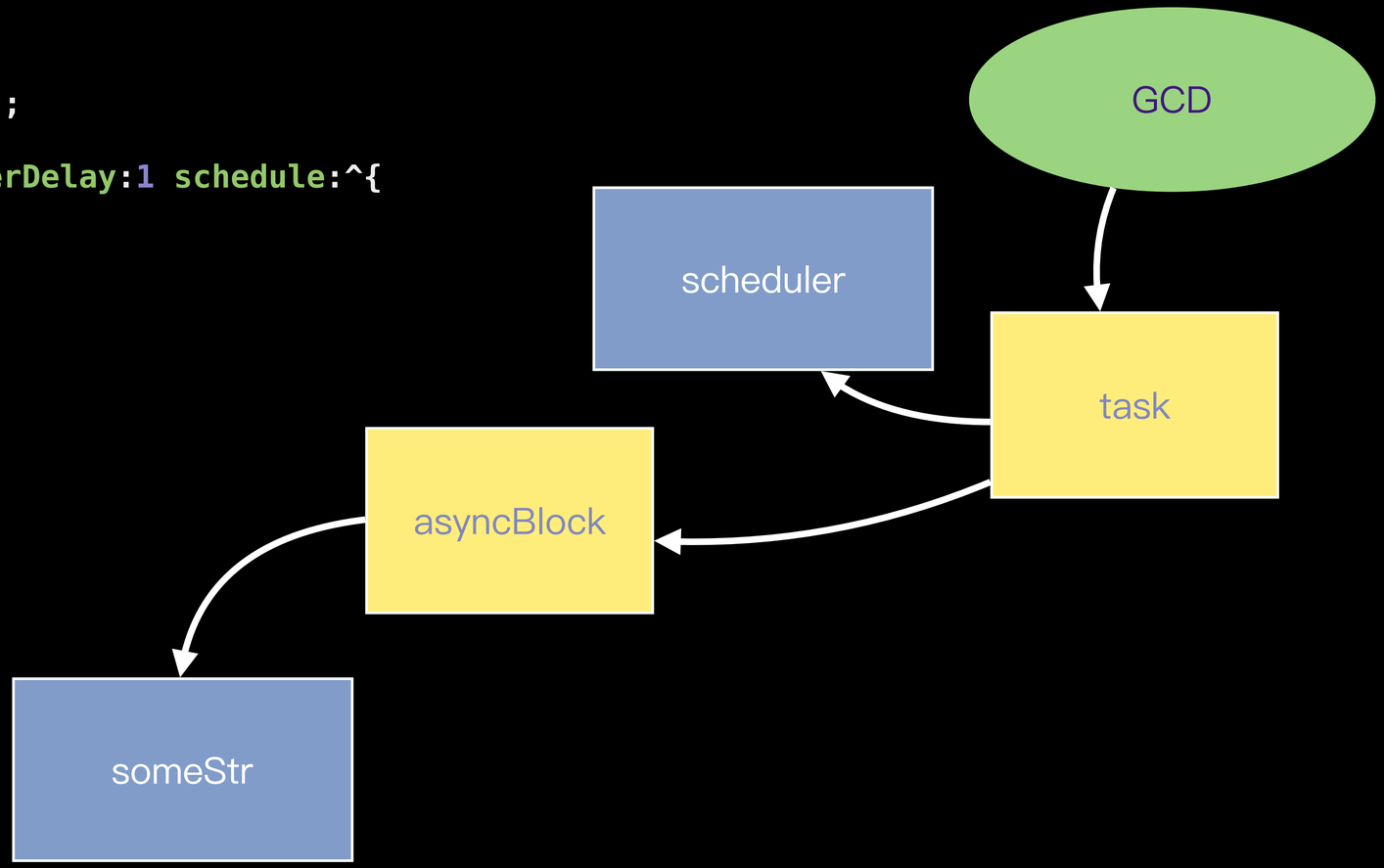


延时情况下的生命周期

延时情况下的生命周期

延时操作和Block生命周期

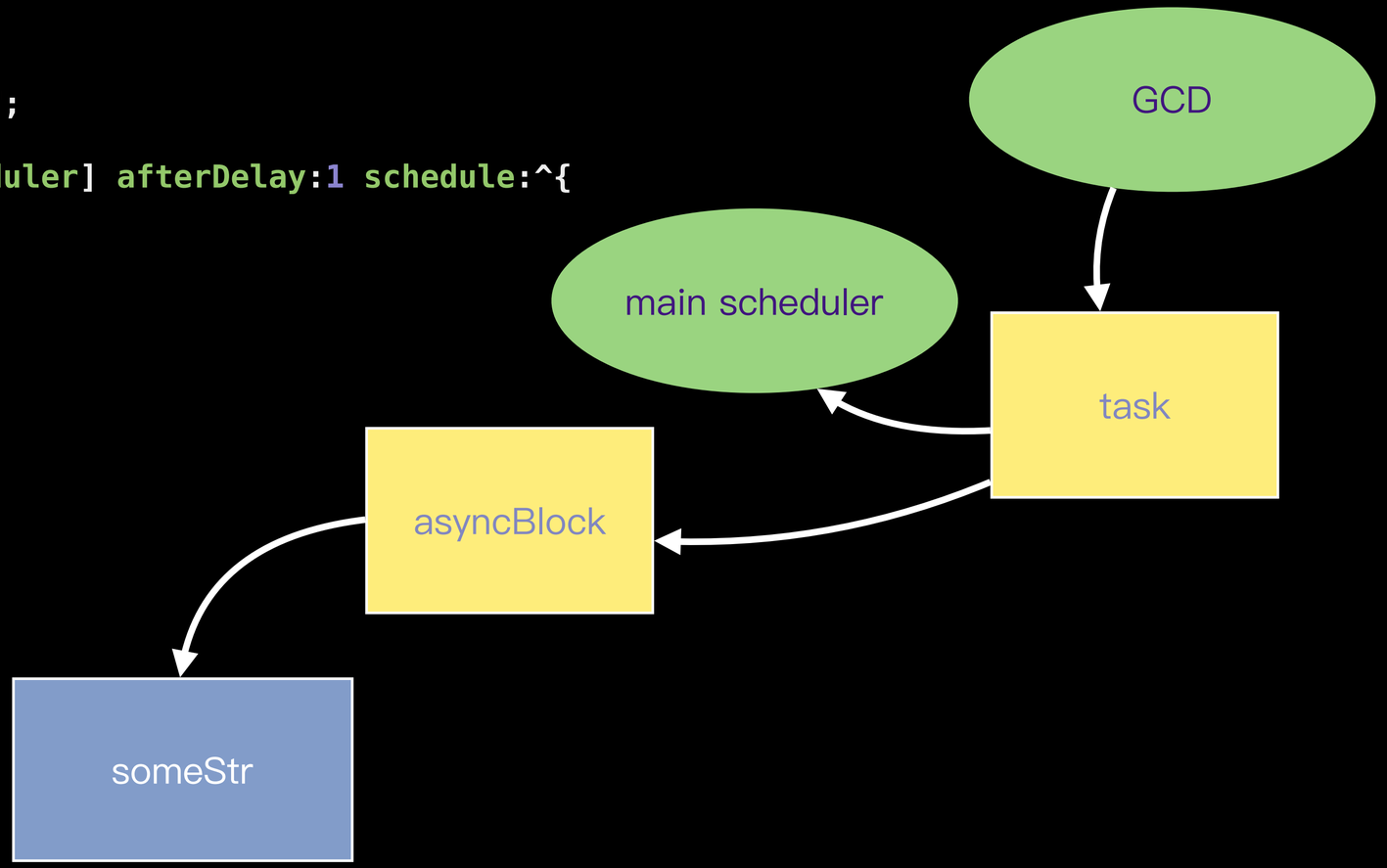
```objc
NSString *someStr = @"someStr";

[[RACScheduler scheduler] afterDelay:1 schedule:^{
    NSLog(@"%@", someStr);
}];
```



延时情况下的生命周期

延时操作和Block生命周期

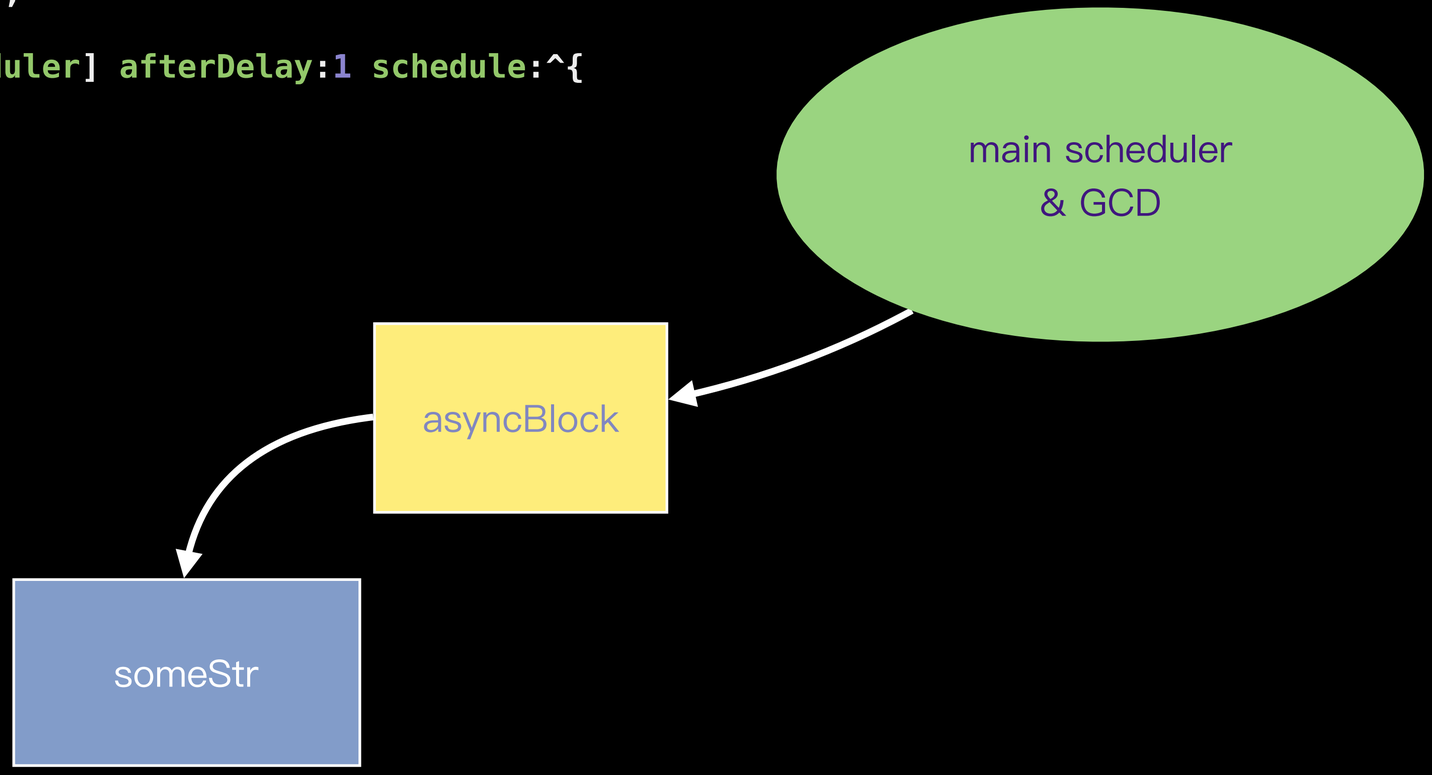
```
NSString *someStr = @"someStr";

[[RACScheduler mainThreadScheduler] afterDelay:1 schedule:^{
    NSLog(@"%@", someStr);
}];
```



延时情况下的生命周期

延时操作和Block生命周期-简化表示

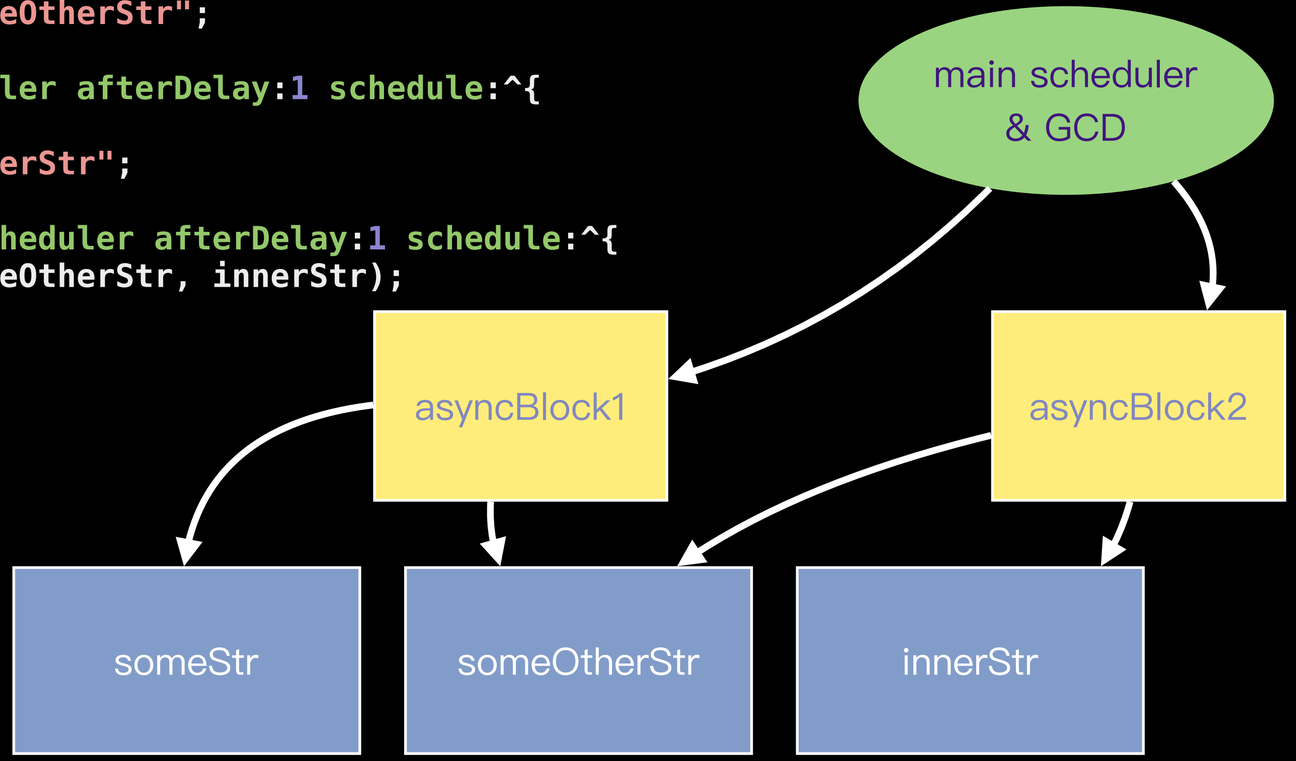
```
NSString *someStr = @"someStr";

[[RACScheduler mainThreadScheduler] afterDelay:1 schedule:^{
    NSLog(@"%@", someStr);
}];
```



延时情况下的生命周期

延时操作和Block生命周期

```
NSString *someStr = @"someStr";
NSString *someOtherStr = @"someOtherStr";

[RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
    NSLog(@"%@", someStr);
    NSString *innerStr = @"innerStr";

    [RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
        NSLog(@"%@ && %@", someOtherStr, innerStr);
    }];
}];
```



延时情况下的生命周期

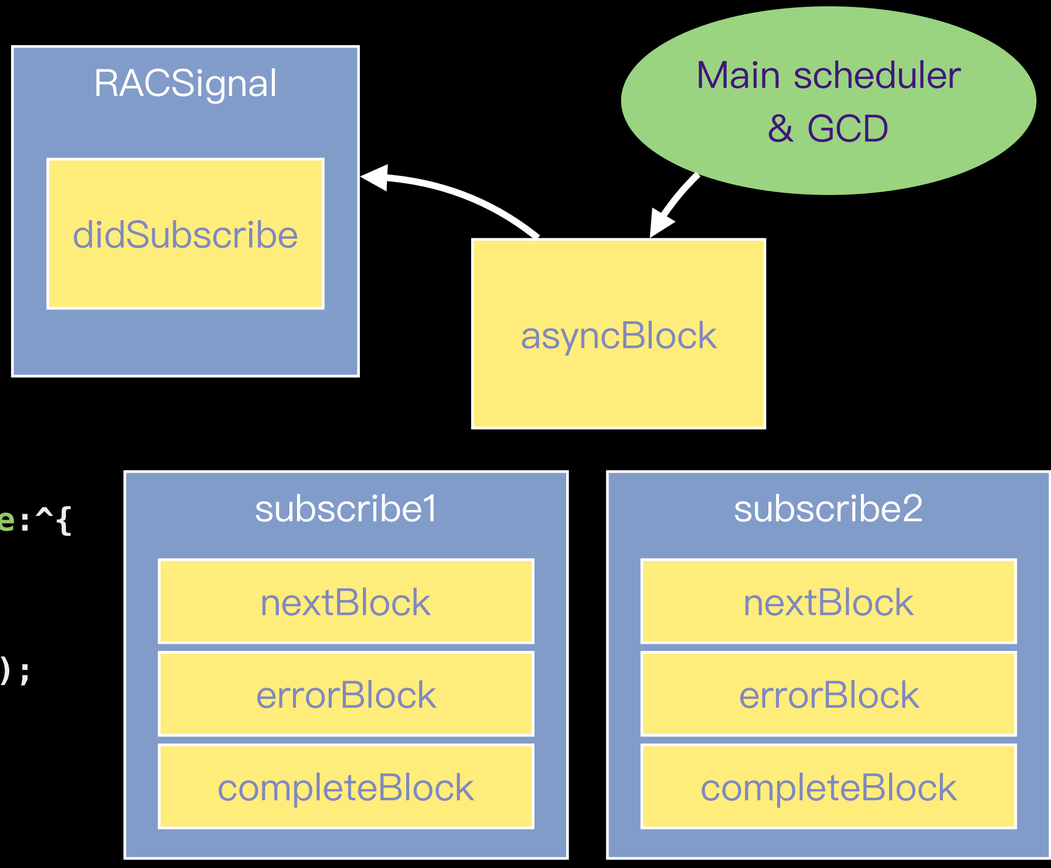
延时的情况分类

- 延时订阅
- 延时发送

延时情况下的生命周期

延时订阅冷信号的情况

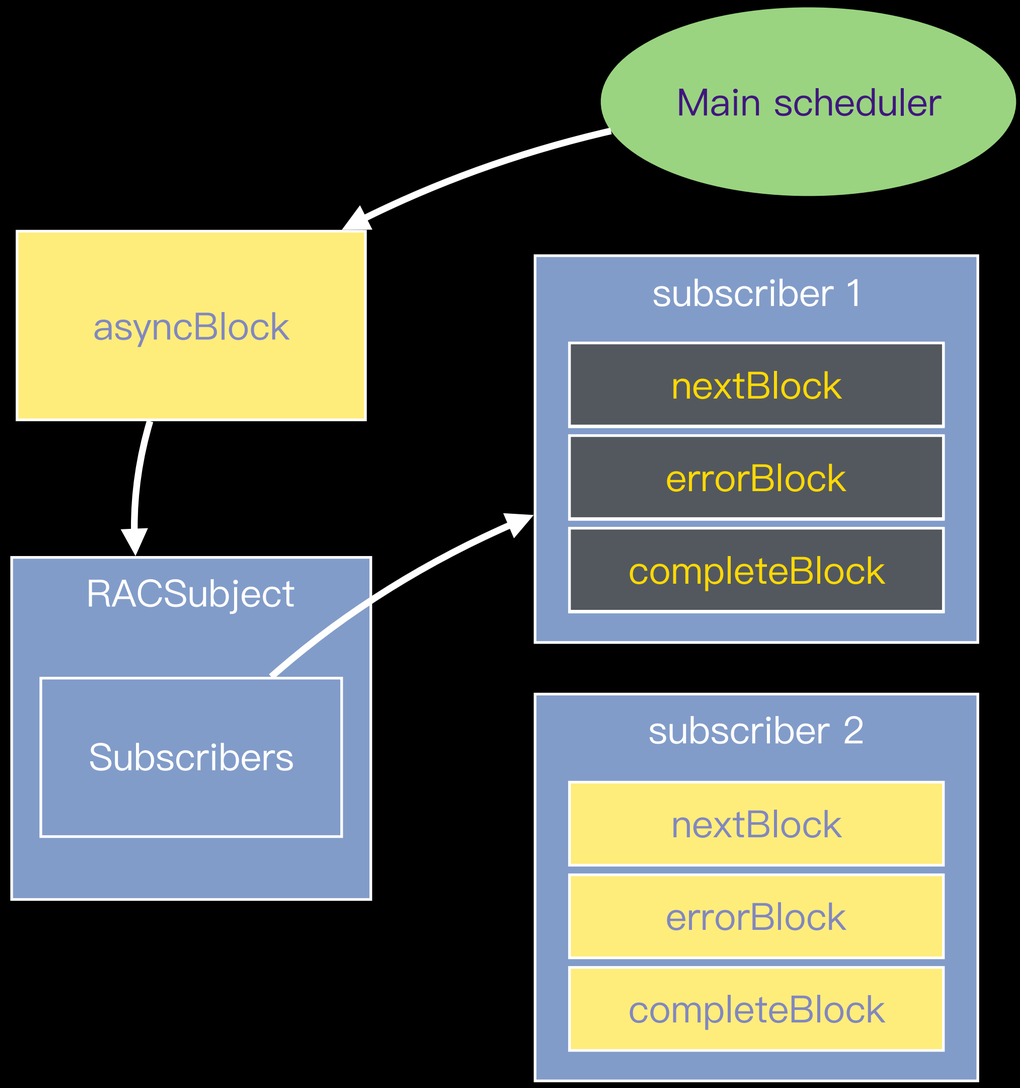
```
RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
    [subscriber sendNext:@1];
    [subscriber sendNext:@2];
    [subscriber sendCompleted];
    return nil;
}];

[signal subscribeNext:^(id x) {
    NSLog(@"Subscriber 1 receive next: %@", x);
} error:^(NSError *error) {
    NSLog(@"Subscriber 1 receive error: %@", error);
} completed:^{
    NSLog(@"Subscriber 1 receive complete");
}];

[RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
    [signal subscribeNext:^(id x) {
        NSLog(@"Subscriber 2 receive next: %@", x);
    } error:^(NSError *error) {
        NSLog(@"Subscriber 2 receive error: %@", error);
    } completed:^{
        NSLog(@"Subscriber 2 receive complete");
    }];
}];
```



延时情况下的生命周期

延时订阅热信号的情况

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

[subject subscribeNext:^(id x) {
    NSLog(@"Subscriber 1 receive next: %@", x);
} error:^(NSError *error) {
    NSLog(@"Subscriber 1 receive error: %@", error);
} completed:^{
    NSLog(@"Subscriber 1 receive complete");
}];

[RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
    [subject subscribeNext:^(id x) {
        NSLog(@"Subscriber 2 receive next: %@", x);
    } error:^(NSError *error) {
        NSLog(@"Subscriber 2 receive error: %@", error);
    } completed:^{
        NSLog(@"Subscriber 2 receive complete");
    }];
}];

[subject sendNext:@1];
[subject sendNext:@2];
[subject sendCompleted];
```



延时情况下的生命周期

延时订阅总结

- 需要其他对象来维持信号的引用
- 冷信号的订阅者仍然与自己没有引用关系
- 热信号的订阅者会随着维持信号的对象被引用

延时情况下的生命周期

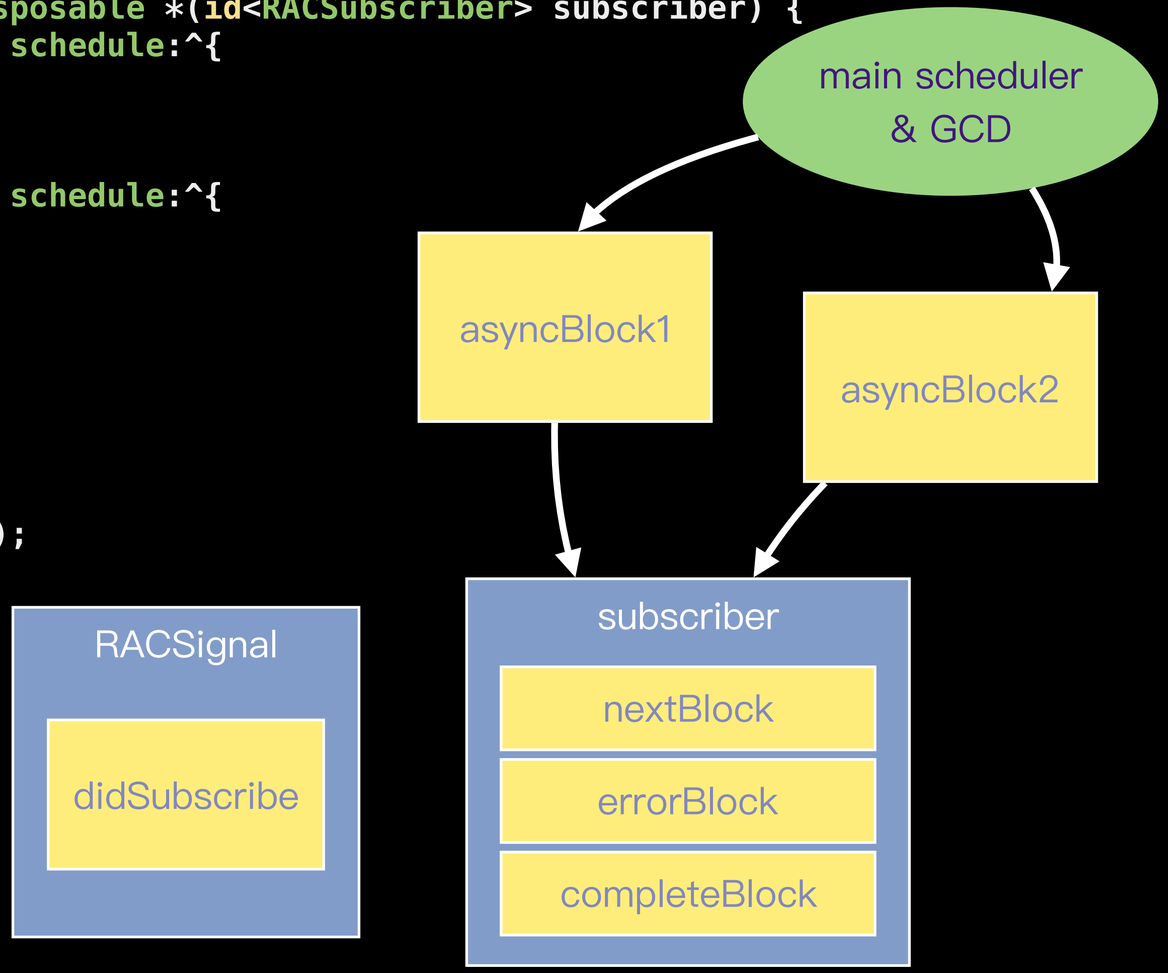
冷信号延时发送的情况

```
RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
    [RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
        [subscriber sendNext:@1];
        [subscriber sendNext:@2];
    }];
    [RACScheduler.mainThreadScheduler afterDelay:2 schedule:^{
        [subscriber sendCompleted];
    }];
    return nil;
}];

[signal subscribeNext:^(id x) {
    NSLog(@"Subscriber 1 receive next: %@", x);
} error:^(NSError *error) {
    NSLog(@"Subscriber 1 receive error: %@", error);
} completed:^{
    NSLog(@"Subscriber 1 receive complete");
}];
```

延时情况下的生命周期

冷信号延时发送的情况

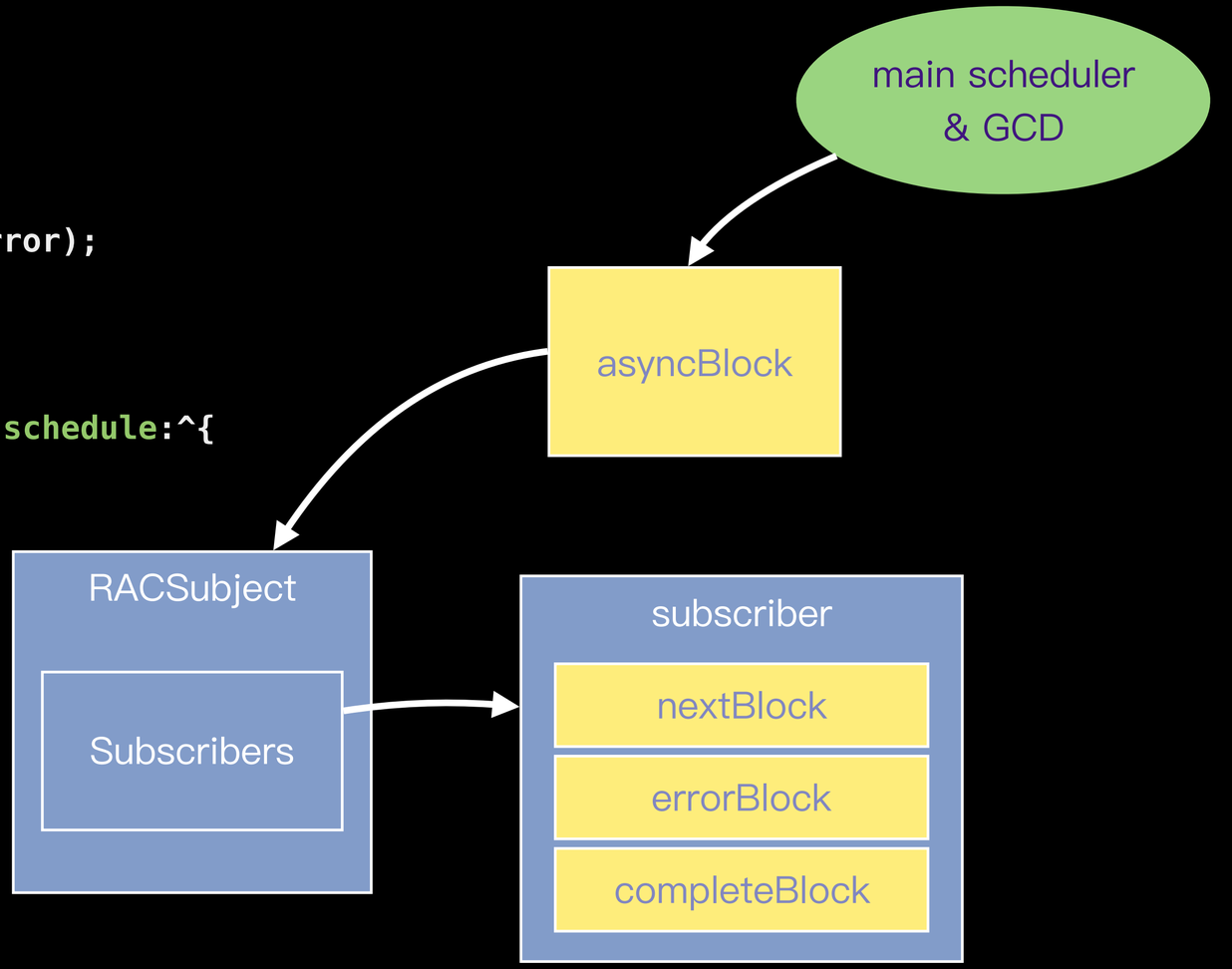
```
RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
    [RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
        [subscriber sendNext:@1];
        [subscriber sendNext:@2];
    }];
    [RACScheduler.mainThreadScheduler afterDelay:2 schedule:^{
        [subscriber sendCompleted];
    }];
    return nil;
}];

[signal subscribeNext:^(id x) {
    NSLog(@"Subscriber 1 receive next: %@", x);
} error:^(NSError *error) {
    NSLog(@"Subscriber 1 receive error: %@", error);
} completed:^{
    NSLog(@"Subscriber 1 receive complete");
}];
```



延时情况下的生命周期

热信号延时发送的情况

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

[subject subscribeNext:^(id x) {
    NSLog(@"Subscriber 1 receive next: %@", x);
} error:^(NSError *error) {
    NSLog(@"Subscriber 1 receive error: %@", error);
} completed:^{
    NSLog(@"Subscriber 1 receive complete");
}];

[RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
    [subject sendNext:@1];
    [subject sendNext:@2];
    [subject sendCompleted];
}];
```



延时情况下的生命周期

延时发送总结

- 冷信号在延迟发送的时刻早已销毁
- 冷信号通过维持订阅者的生存期达到延迟发送
- 热信号需要延迟对其发送所以会保留
- 热信号会持有订阅者，所以不需要特殊处理

变换组合时的生命周期

变换组合时的生命周期

情况分类

- 变换
- 组合

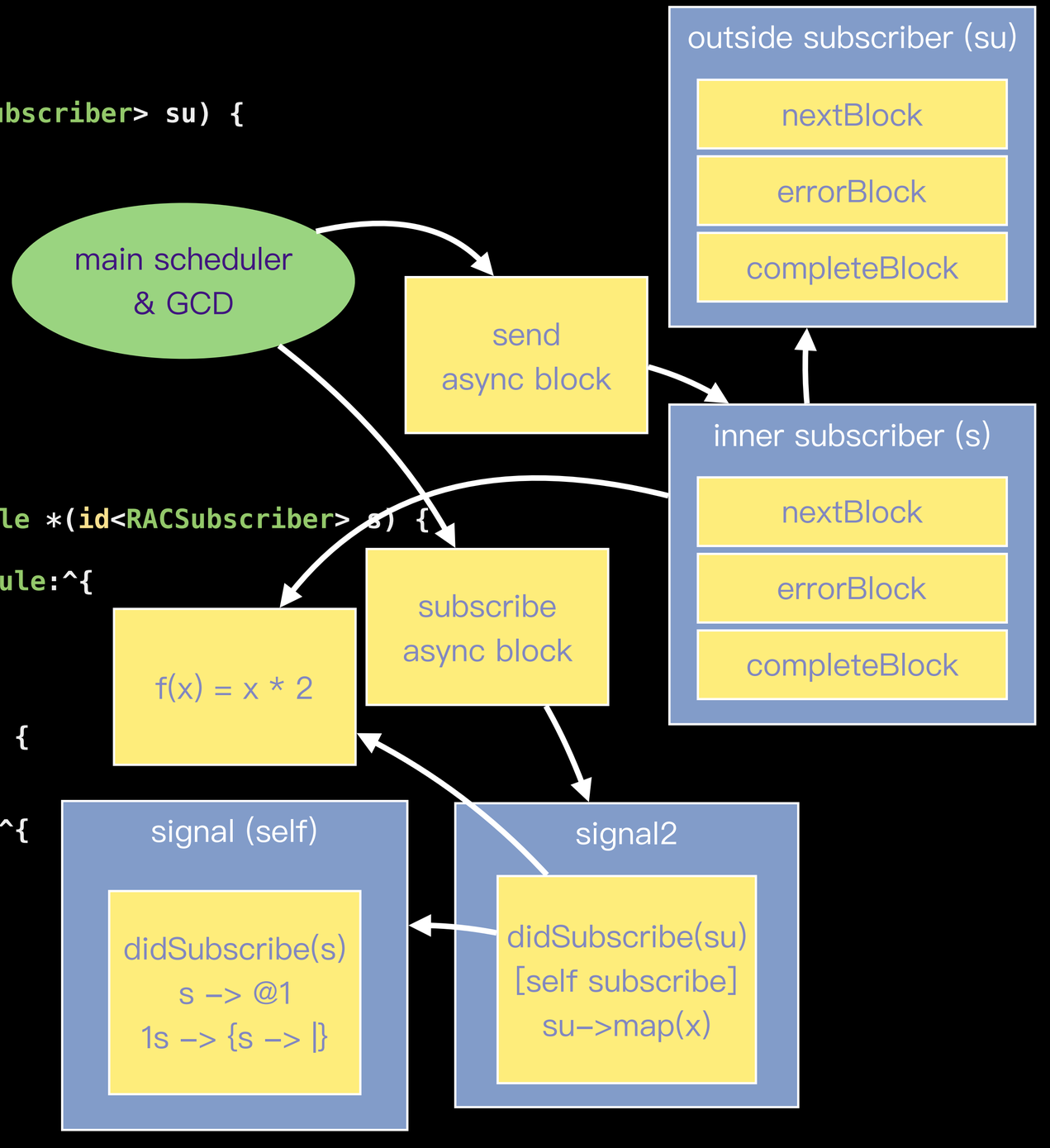
变换组合时的生命周期

信号变换的生命周期

```
- (RACSignal *)myMap_:(id (^)(id))map
{
    NSCParameterAssert(map != nil);
    return [RACSignal createSignal:
            ^RACDisposable *(id<RACSubscriber> su) {
        [self subscribeNext:^(id x) {
            [su sendNext:map(x)];
        } error:^(NSError *error) {
            [su sendError:error];
        } completed:^{
            [su sendCompleted];
        }];
        return nil;
    }];
}
```

```
- (RACSignal *)myMap2_:(id (^)(id))map
{
    @weakify(self)
    NSCParameterAssert(map != nil);
    return [RACSignal createSignal:
            ^RACDisposable *(id<RACSubscriber> su) {
        @strongify(self)
        [self subscribeNext:^(id x) {
            [su sendNext:map(x)];
        } error:^(NSError *error) {
            [su sendError:error];
        } completed:^{
            [su sendCompleted];
        }];
        return nil;
    }];
}
```

```objc
- (RACSignal *)myMap_:(id (^)(id))map
{
    NSCParameterAssert(map != nil);
    return [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> su) {
        [self subscribeNext:^(id x) {
            [su sendNext:map(x)];
        } error:^(NSError *error) {
            [su sendError:error];
        } completed:^{
            [su sendCompleted];
        }];
        return nil;
    }];
}
void signalTransform()
{
    RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> s) {
        [s sendNext:@1];
        [RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
            [s sendCompleted];
        }];
        return nil;
    }];
    RACSignal *signal2 = [signal myMap_:^id(NSNumber *value) {
        return @(value.integerValue * 2);
    }];
    [RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
        [signal2 subscribeNext:^(id x) {
            NSLog(@"Subscriber 1 receive next: %@", x);
        } error:^(NSError *error) {
            NSLog(@"Subscriber 1 receive error: %@", error);
        } completed:^{
            NSLog(@"Subscriber 1 receive complete");
        }];
    }];
}
```



变换组合时的生命周期

信号变换总结

- 变换后的信号持有源信号
- 变换后的信号一般持有变换用到的闭包
- 订阅时会创建内部订阅者，并持有外部订阅者
- 内部订阅者一般会持有变换用到的闭包

变换组合时的生命周期

信号组合时的生命周期

```
+ (RACSignal *)myMerge_:(id<NSFastEnumeration>)signals
{
    return [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        for (RACSignal *signal in signals) {
            [signal subscribeNext:^(id x) {
                [subscriber sendNext:x];
            } error:^(NSError *error) {
                [subscriber sendError:error];
            } completed:^{
                [subscriber sendCompleted];
            }];
        }
        return nil;
    }];
}
```

变换组合时的生命周期

信号组合总结

- 组合后的信号持有所有的源信号
- 订阅时会创建很多内部订阅者，并持有外部订阅者

RACDisposable

RACDisposable

是什么？

- 可销毁对象，“销毁按钮”
- 主要用于RACSignal，但也用于RACScheduler
- 用于RACSignal时，相当于断开订阅连接

RACDisposable

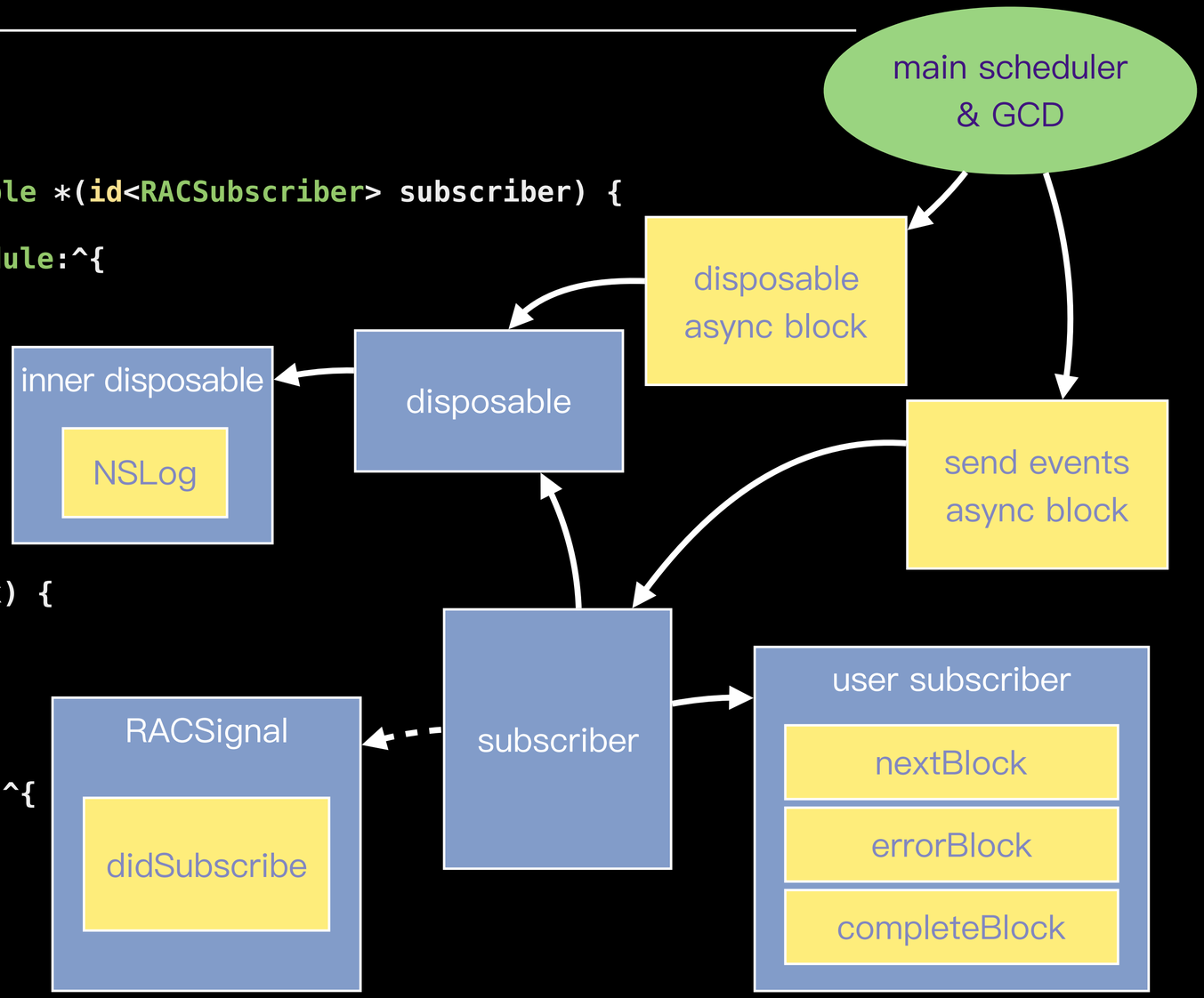
作用

```
@interface RACDisposable : NSObject

@property (atomic, assign, getter = isDisposed, readonly) BOOL disposed;

+ (instancetype)disposableWithBlock:(void (^)(void))block;

- (void)dispose;

@end
```

RACDisposable

真·生命周期

```
RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
    [subscriber sendNext:@1];
    [RACScheduler.mainThreadScheduler afterDelay:2 schedule:^{
        [subscriber sendNext:@2];
        [subscriber sendCompleted];
    }];

    return [RACDisposable disposableWithBlock:^{
        NSLog(@"dispose!!!");
    }];
}];

RACDisposable *disposable = [signal subscribeNext:^(id x) {
    NSLog(@"next: %@", x);
} completed:^{
    NSLog(@"complete");
}];

[RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
    [disposable dispose];
}];
```



RACDisposable

disposable中block作用

```
RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
    [subscriber sendNext:@1];
    RACDisposable *disposable1 = [RACScheduler.mainThreadScheduler afterDelay:2 schedule:^{
        [subscriber sendNext:@2];

    }];
    RACDisposable *disposable2 = [RACScheduler.mainThreadScheduler afterDelay:2 schedule:^{
        [subscriber sendCompleted];
    }];
    return [RACDisposable disposableWithBlock:^{
        [disposable1 dispose];
        [disposable2 dispose];
        NSLog(@"dispose!!!");
    }];
}];
RACDisposable *disposable = [signal subscribeNext:^(id x) {
    NSLog(@"next: %@", x);
} completed:^{
    NSLog(@"complete");
}];
[RACScheduler.mainThreadScheduler afterDelay:1 schedule:^{
    [disposable dispose];
}];
```

信号的生命周期

订阅总结

- 订阅时创建订阅者
- 创建disposable对象
- 创建“透明”订阅者
- 执行didSubscribe闭包，把“透明”订阅者传入
- 把didSubscribe闭包返回的disposable对象添加到一开始的disposable对象中
- 把一开始的disposable作为返回值返回给订阅函数

RACDisposable家族

- RACScopedDisposable
- RACSerialDisposable
- RACCompoundDisposable

RACDisposable家族

RACScopedDisposable——绑定对象生存期的disposable

- + (instancetype)scopedDisposableWithDisposable:
  (RACDisposable *)disposable;
- - (RACScopedDisposable *)asScopedDisposable;
  <RACDisposable>

RACDisposable家族

RACScopedDisposable例子

```
RACSignal *signal = [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
    [subscriber sendNext:@1];
    RACDisposable *disposable1 = [RACScheduler.mainThreadScheduler afterDelay:2 schedule:^{
        [subscriber sendNext:@2];

    }];
    RACDisposable *disposable2 = [RACScheduler.mainThreadScheduler afterDelay:2 schedule:^{
        [subscriber sendCompleted];
    }];
    return [RACDisposable disposableWithBlock:^{
        [disposable1 dispose];
        [disposable2 dispose];
        NSLog(@"dispose!!!");
    }];
}];
RACDisposable *disposable = [signal subscribeNext:^(id x) {
    NSLog(@"next: %@", x);
} completed:^{
    NSLog(@"complete");
}];
self.someProp = disposable.asScopedDisposable;
```

RACDisposable家族

RACSerialDisposable——可替换的disposable包裹

- + (instancetype)serialDisposableWithDisposable:
  (RACDisposable *)disposable;
- @property (atomic, strong) RACDisposable *disposable;

RACDisposable家族

RACSerialDisposable例子

```
- (RACSignal *)myConcat:(RACSignal *)signal
{
    return [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        RACSerialDisposable *mainDisposable = [[RACSerialDisposable alloc] init];
        mainDisposable.disposable = [self subscribeNext:^(id x) {
            [subscriber sendNext:x];
        } error:^(NSError *error) {
            [subscriber sendError:error];
        } completed:^{
            mainDisposable.disposable = [signal subscribe:subscriber];
        }];
        return mainDisposable;
    }];
}
```

RACDisposable家族

RACCompoundDisposable——多个disposable的包裹

- - (void)addDisposable:(RACDisposable *)disposable;
- - (void)removeDisposable:(RACDisposable *)disposable;

RACDisposable家族

RACCompoundDisposable例子

```
+ (RACSignal *)myMerge:(id<NSFastEnumeration>)signals
{
    return [RACSignal createSignal:^RACDisposable *(id<RACSubscriber> subscriber) {
        RACCompoundDisposable *mainDisposable = [[RACCompoundDisposable alloc] init];
        for (RACSignal *signal in signals) {
            [mainDisposable addDisposable:[signal subscribeNext:^(id x) {
                [subscriber sendNext:x];
            } error:^(NSError *error) {
                [subscriber sendError:error];
                [mainDisposable dispose];
            } completed:^{
                [subscriber sendCompleted];
                [mainDisposable dispose];
            }]];
        }
        return mainDisposable;
    }];
}
```

还缺什么？

- RACCommand
- RACChannel
- RACChannelTerminal
- continue…

Q&A

谢谢大家！