+BrianCoyner

# ABOUT ME

# ABOUT ME

- software developer

# ABOUT ME

- software developer
- seek simple solutions to hard problems

# ABOUT ME

- software developer
- seek simple solutions to hard problems
- love product development

# ABOUT SESSION

# ABOUT SESSION

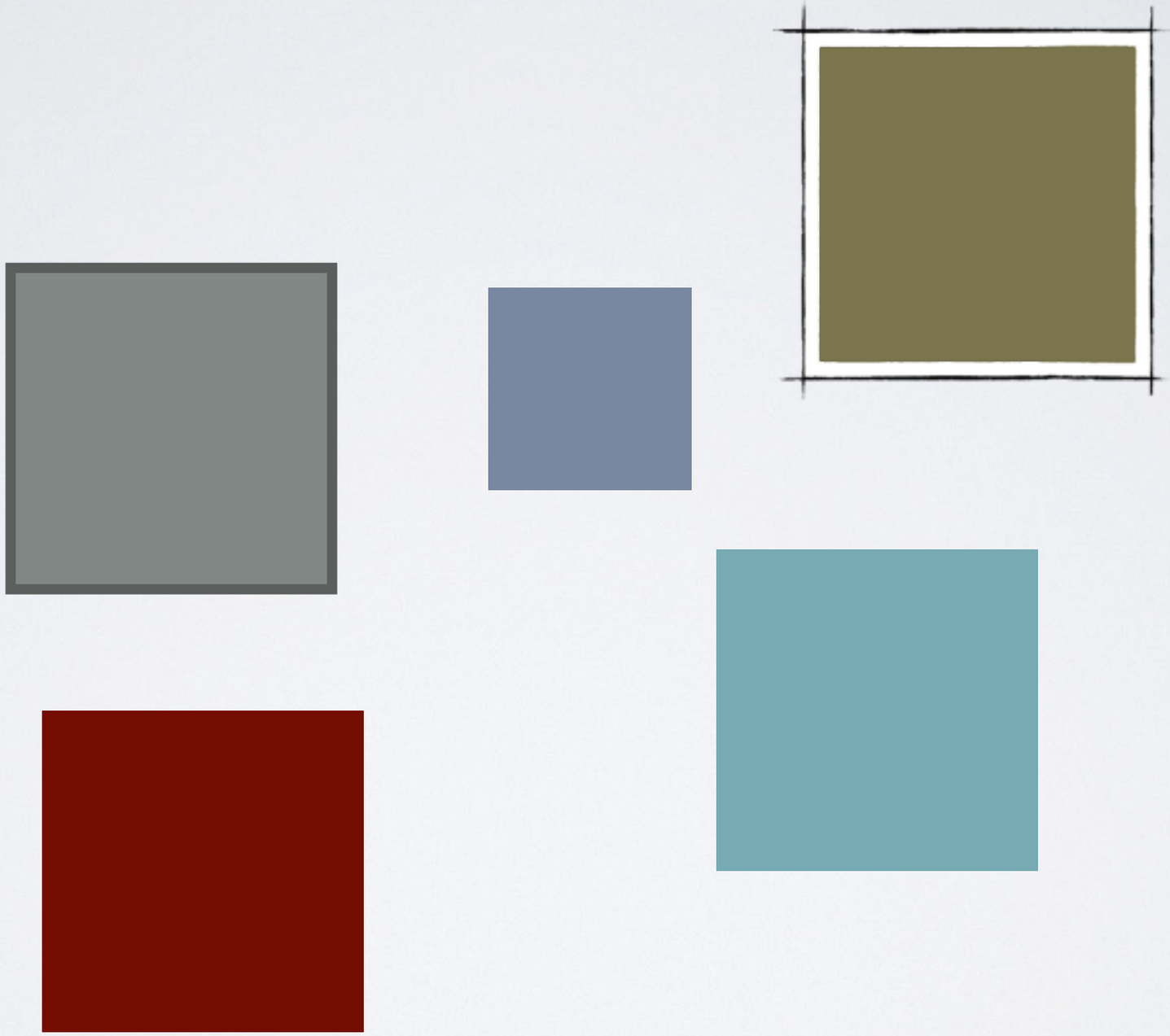
- applying “simple” concepts to “hard” solve problems

# ABOUT SESSION

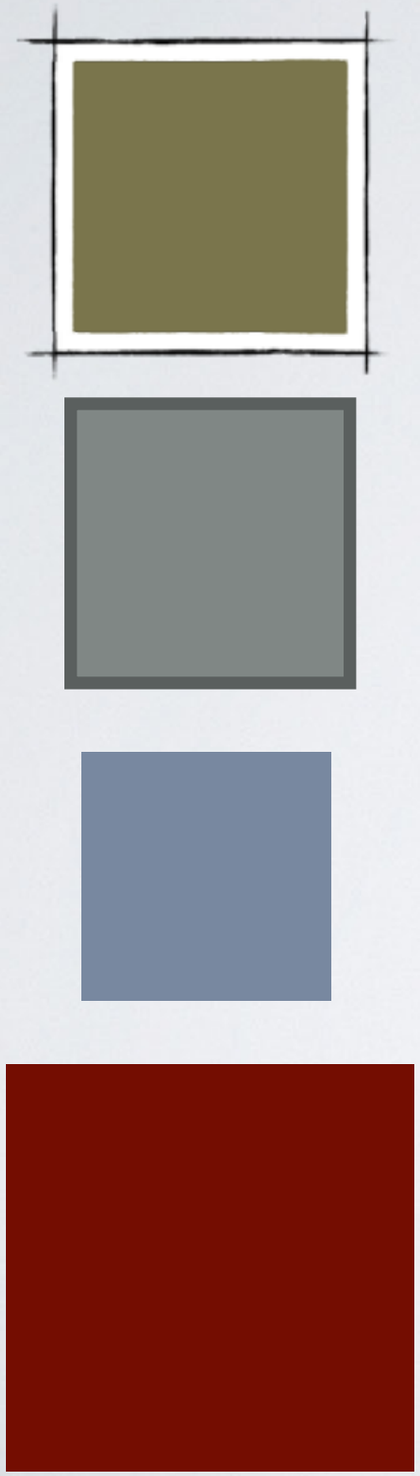
- applying “simple” concepts to “hard” solve problems

- don’t worry, it’s not about UML

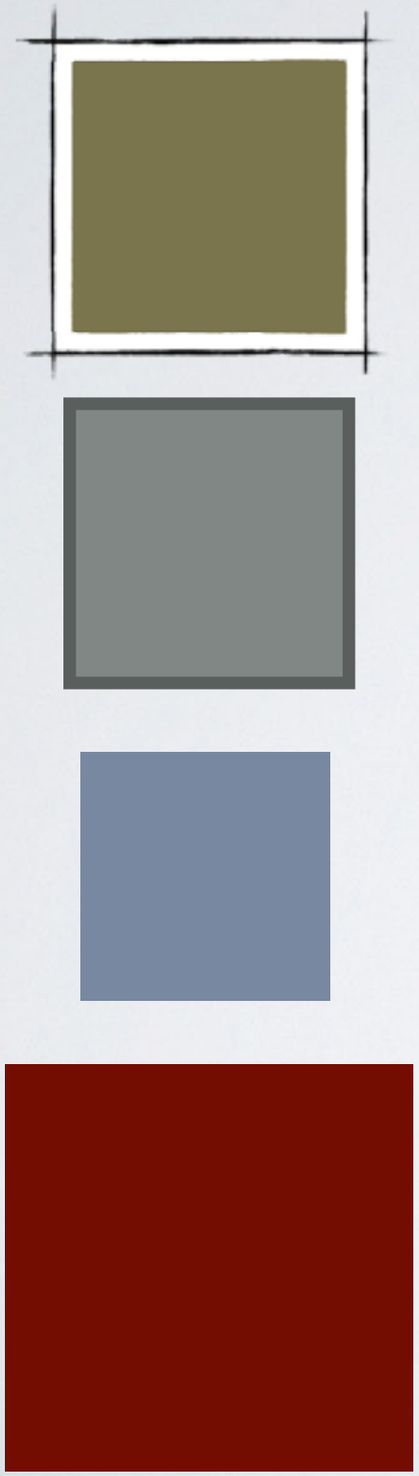
# WHAT'S A BOX?



# WHAT'S A BOX

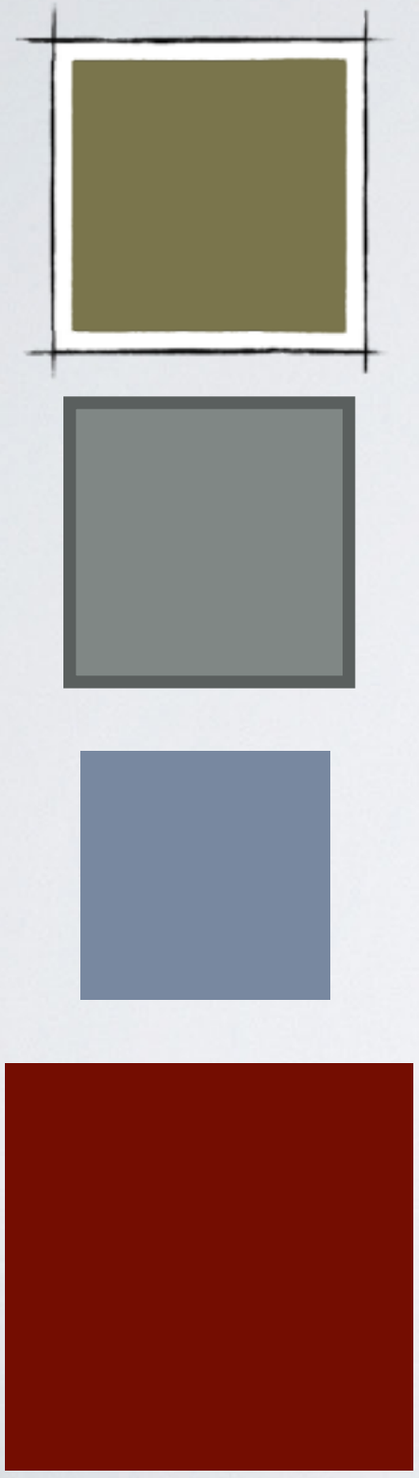


# WHAT'S A BOX



interface / protocol

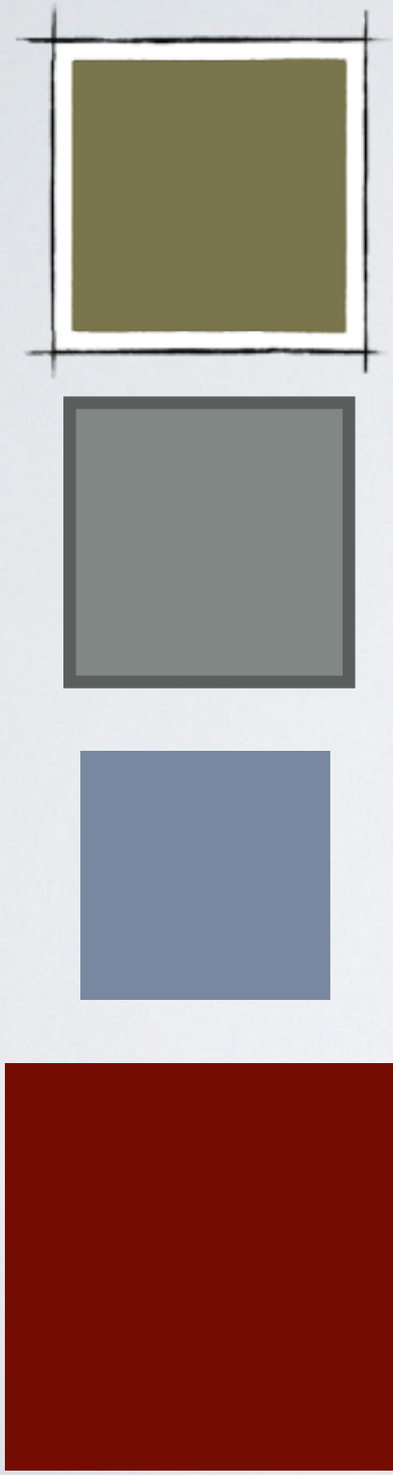
# WHAT'S A BOX



interface / protocol

concrete object

# WHAT'S A BOX

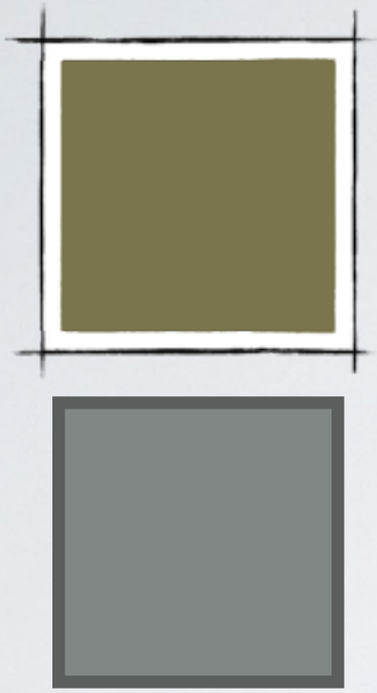


interface / protocol

concrete object

small block of code

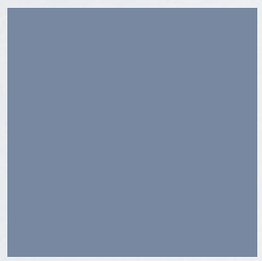
# WHAT'S A BOX



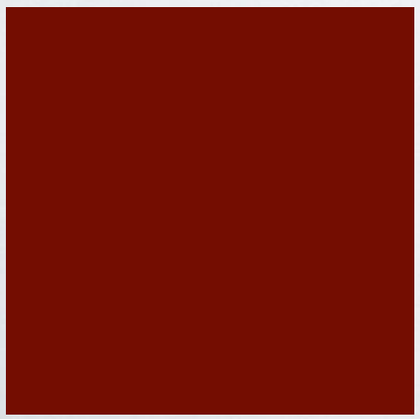
interface / protocol



concrete object

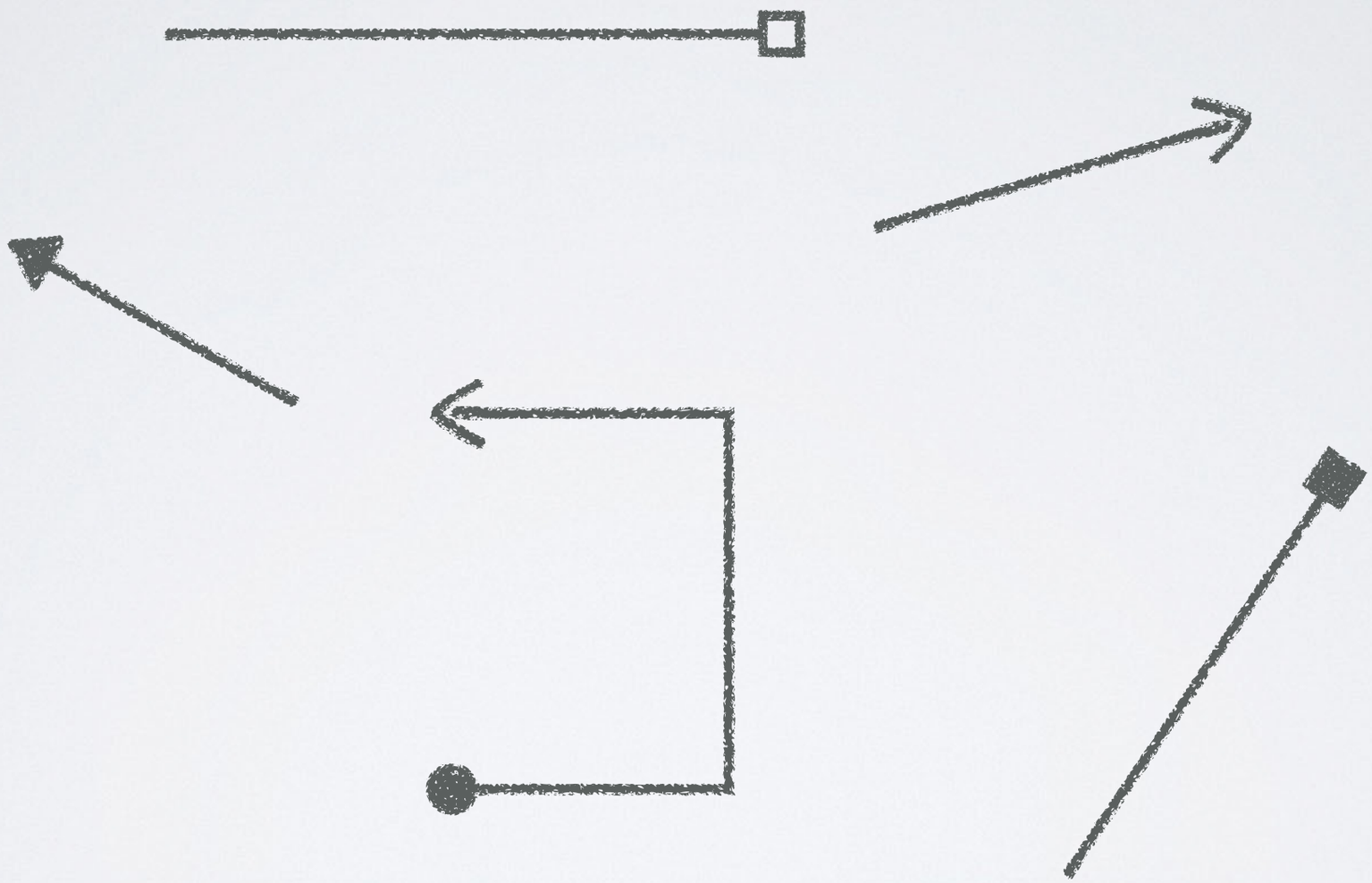


small block of code

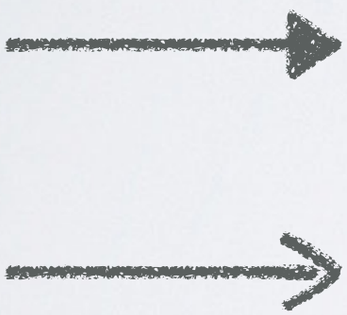


large block of code

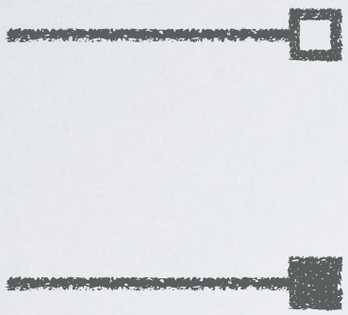
# WHAT'S AN ARROW?



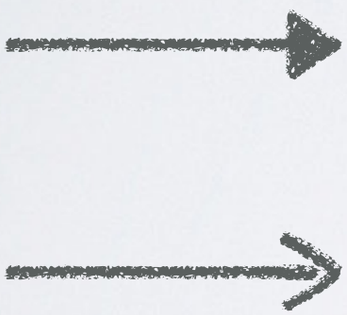
# WHAT'S AN ARROW?



# WHAT'S AN ARROW?



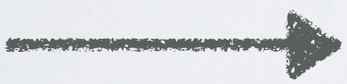
object receives via public API



# WHAT'S AN ARROW?

object receives via public API

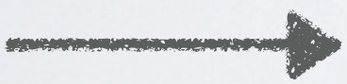
object looks up / creates



# WHAT'S AN ARROW?

object receives via public API

object looks up / creates

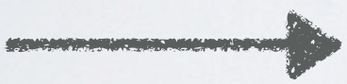


instance of / conforms to

# WHAT'S AN ARROW?

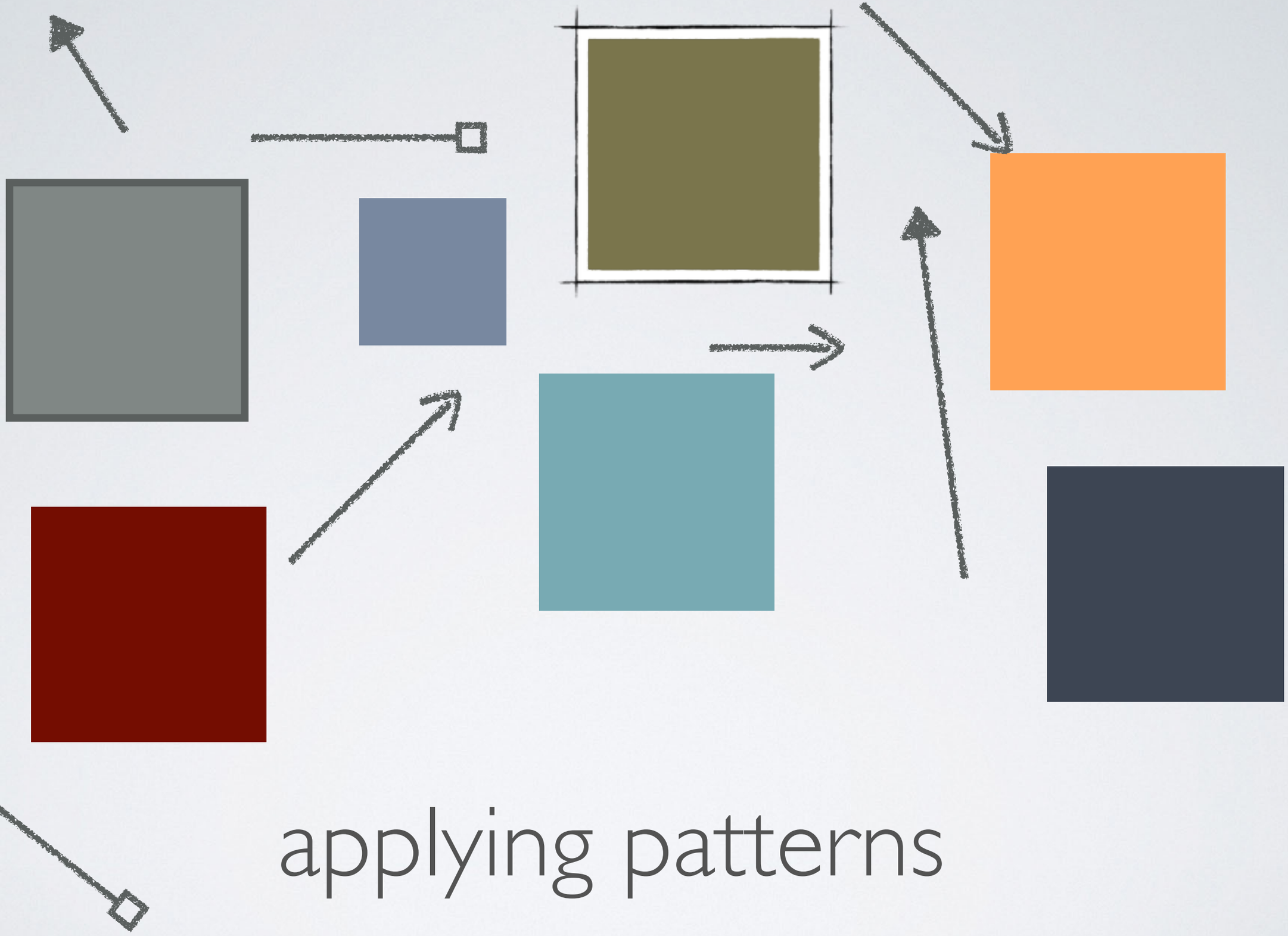
object receives via public API

object looks up / creates



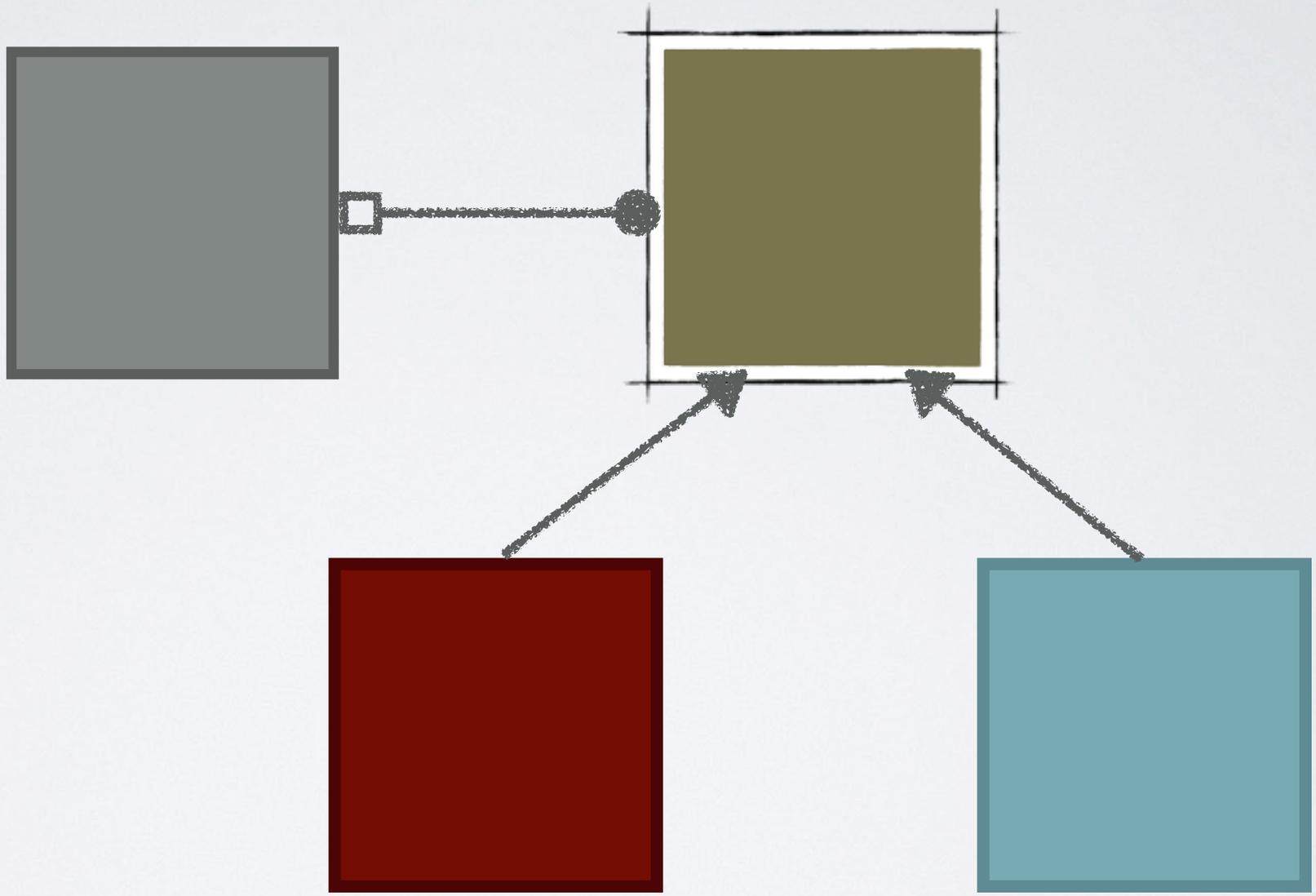
instance of / conforms to

code dependency / executes code

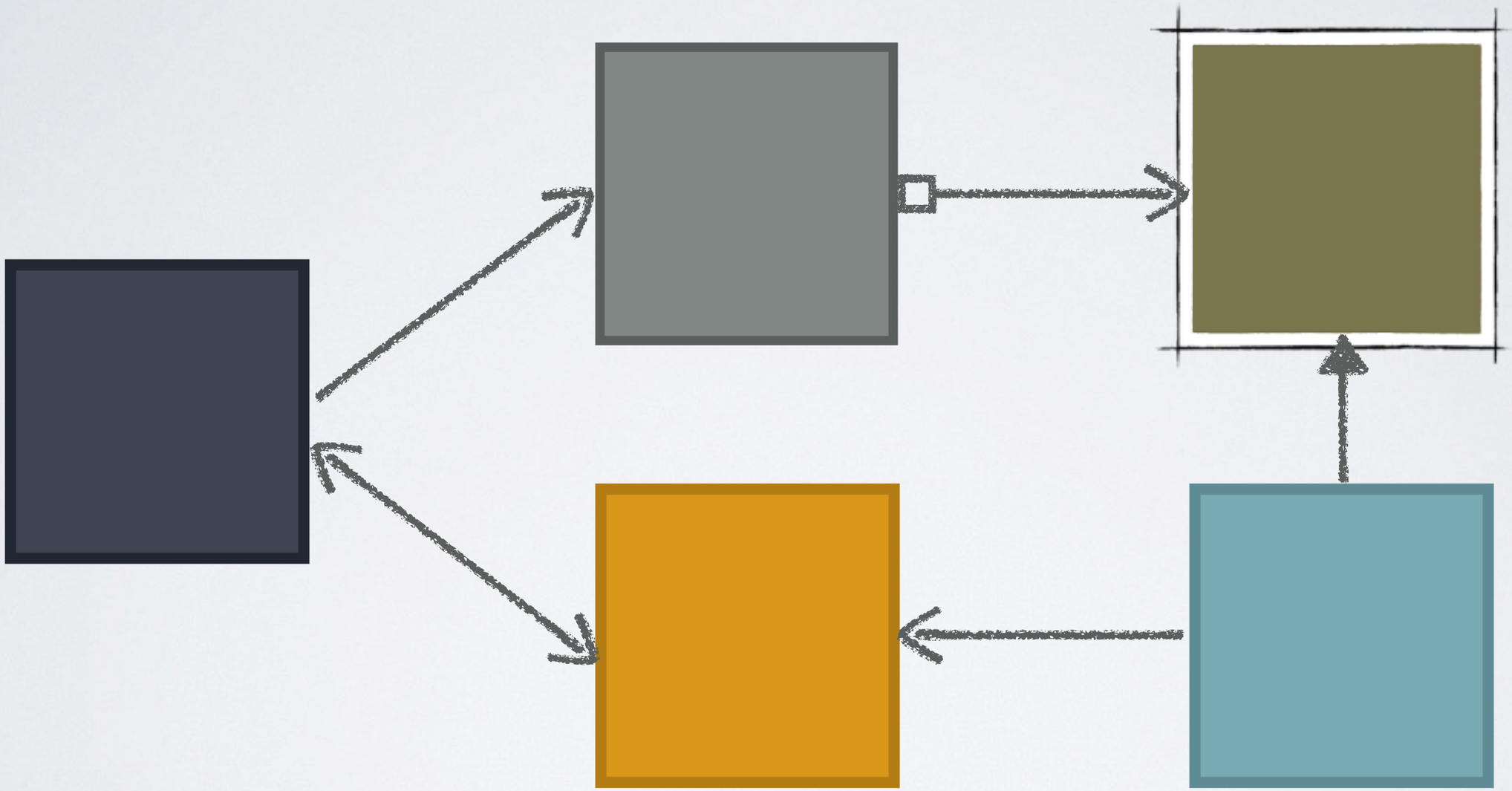


# applying patterns

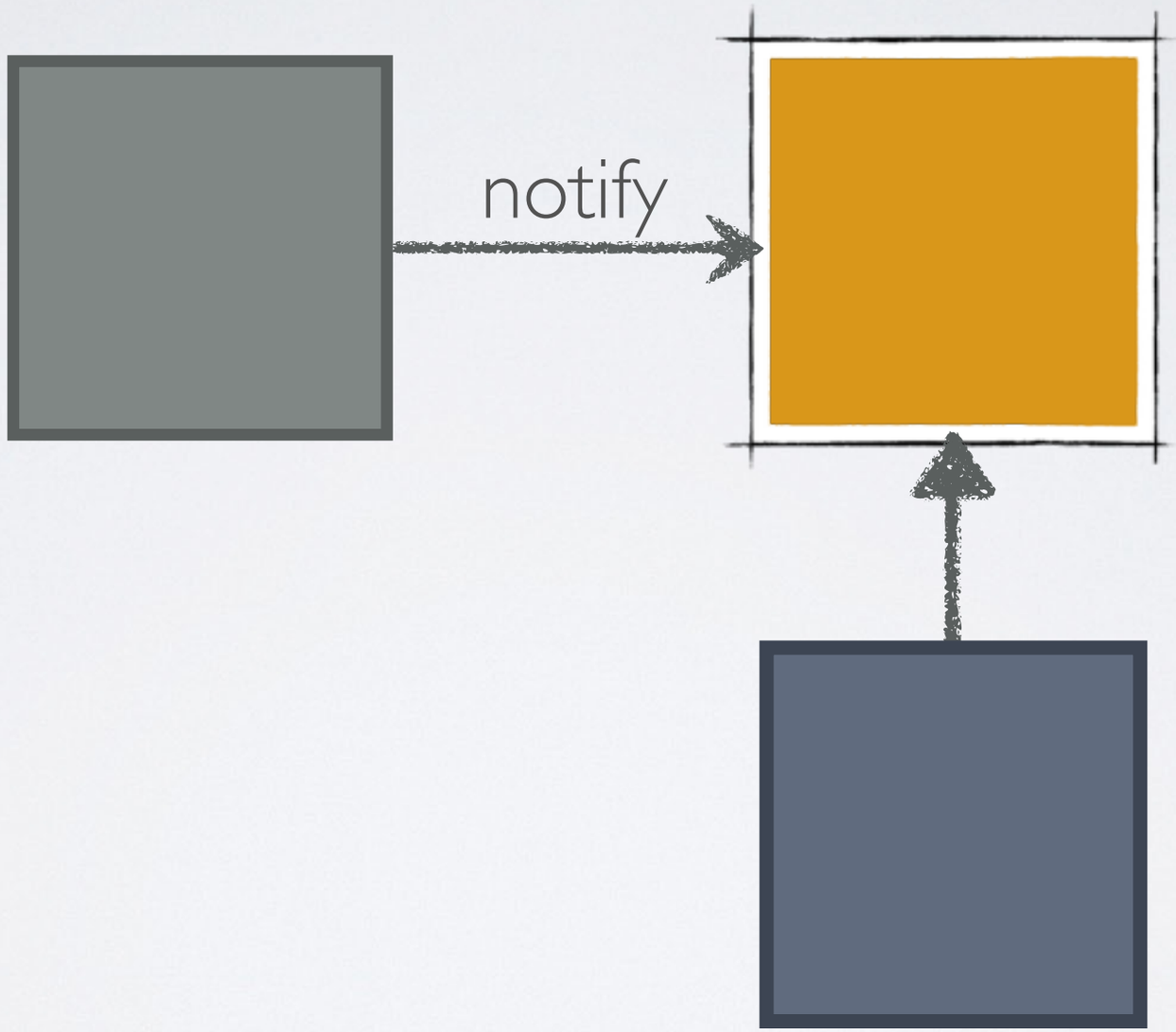
# STRATEGY



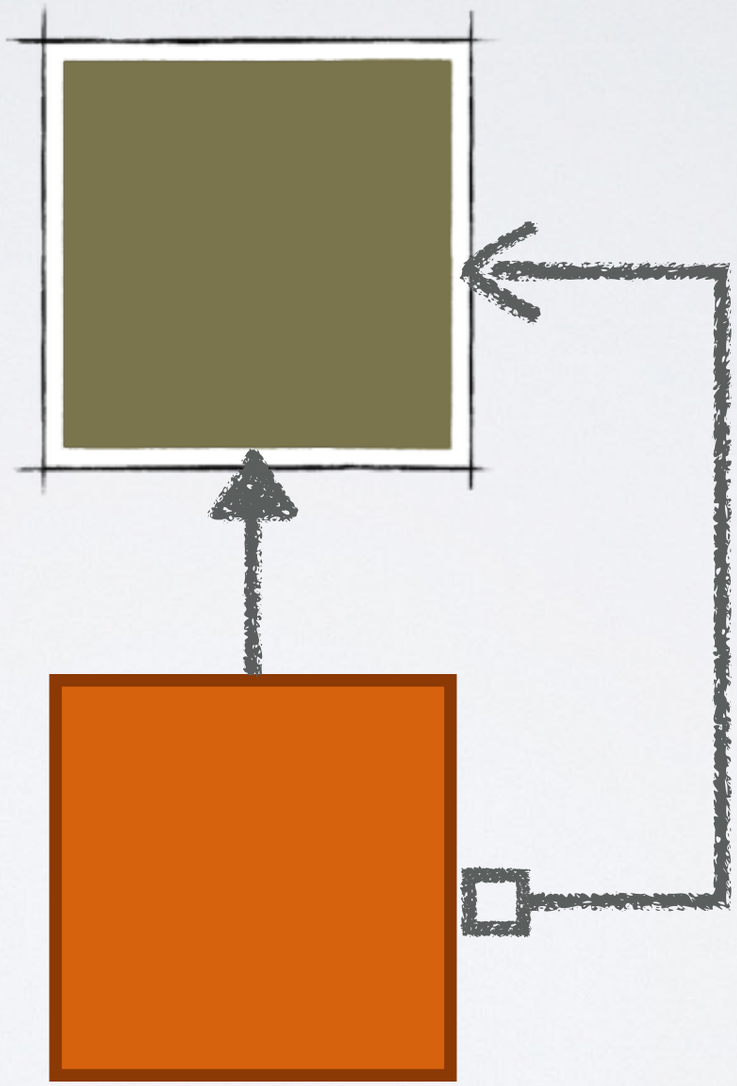
# COMMAND



# OBSERVER



# COMPOSITE



# WHAT'S NEXT?

# WHAT'S NEXT?

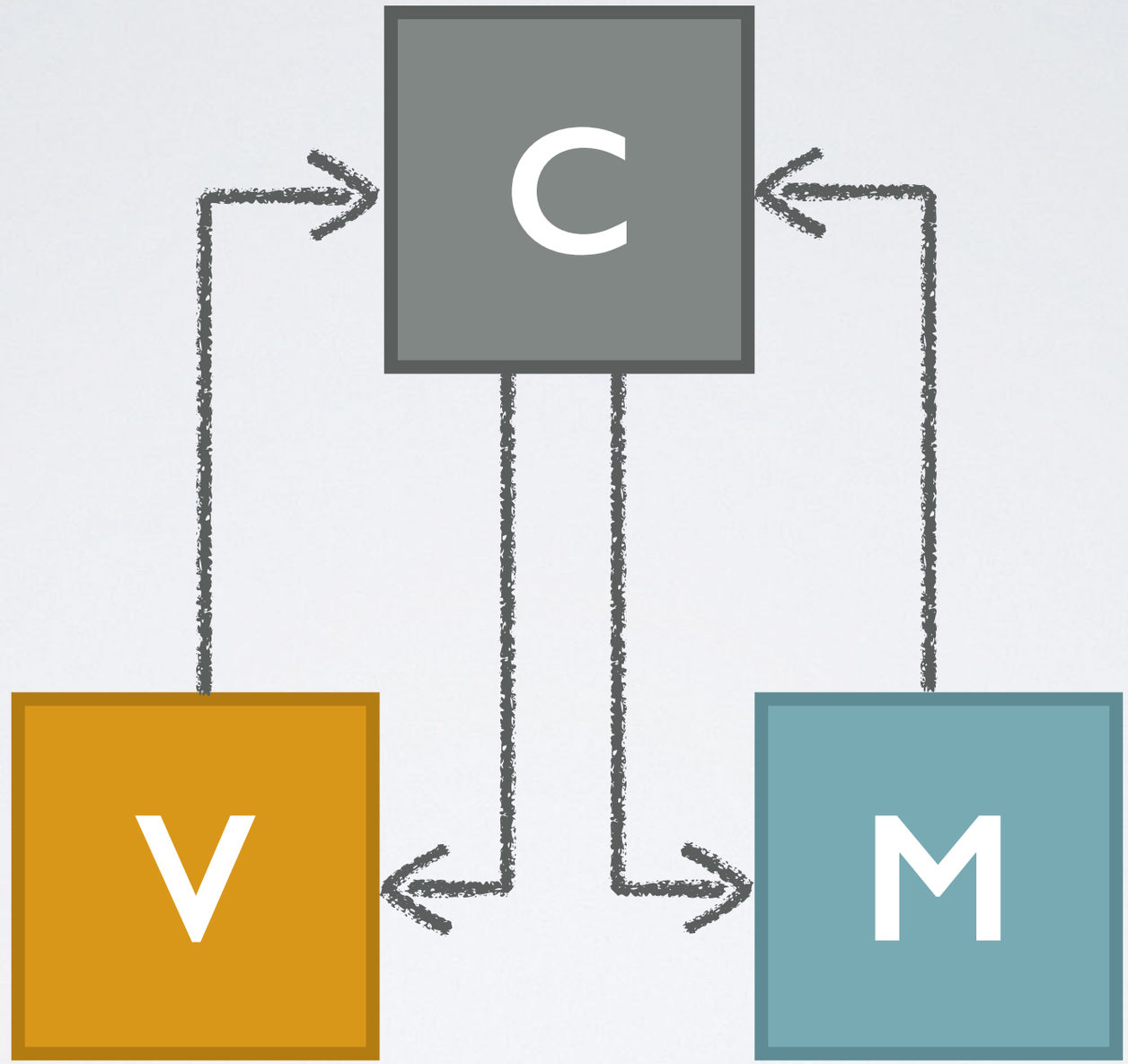
- well-behaved view controllers

# WHAT'S NEXT?

- well-behaved view controllers

- building an asynchronous workflow "engine"

# VIEW CONTROLLERS


C
V
M

# “STATIC”’VIEW CONTROLLERS

# “STATIC”VIEW CONTROLLERS

- view or edit “entity”

# “STATIC”VIEW CONTROLLERS

- view or edit “entity”
- e*ffectively* immutable

# “STATIC”VIEW CONTROLLERS

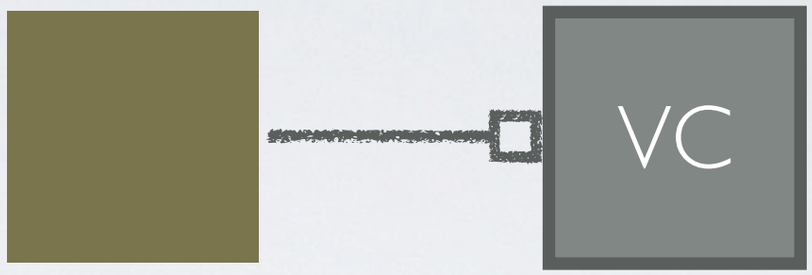
- view or edit “entity”
- e*ffectively* immutable
- *simple* public api

# “STATIC”VIEW CONTROLLERS

- view or edit “entity”
- *effectively* immutable
- *simple* public api
- *easy* to implement

# “STATIC”VIEW CONTROLLERS

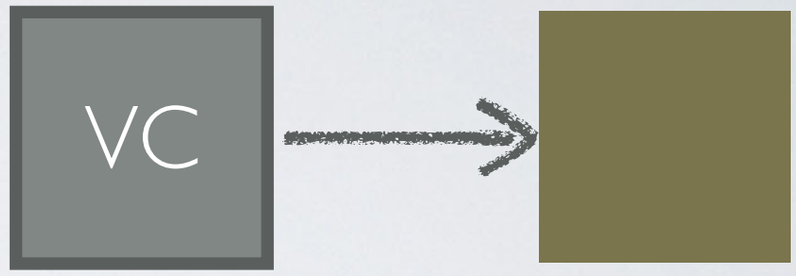
- view or edit “entity”
- *effectively* immutable
- *simple* public api
- *easy* to implement




```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readonly) BTSCargo *cargo;

- (instancetype)initWithCargo:(BTSCargo *)cargo;

@end
```

# “STATIC”VIEW CONTROLLERS

- view or edit “entity”
- *effectively* immutable
- *simple* public api
- *easy* to implement




```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readonly) BTSCargo *cargo;

- (instancetype)initWithCargo:(BTSCargo *)cargo;

@end
```

# “STATIC”VIEW CONTROLLERS

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readonly) BTSCargo *cargo;

@end
```

```
@implementation BTSCargoViewController

#pragma mark - Object Life Cycle

- (instancetype)init
{
    self = [super initWithStyle:UITableViewStyleGrouped];
    return self;
}
```

# “STATIC”VIEW CONTROLLERS

```
#pragma mark - View Life Cycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self configureNavigationItem:[self navigationItem]];
    [self registerTableViewCellsWithTableView:[self tableView] bundle:[self resourceBundle]];

    _tableModel = [self buildTableModelForCargo:_cargo];
}
```

# “STATIC”VIEW CONTROLLERS

```
#pragma mark - View Life Cycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self configureNavigationItem:[self navigationItem]];
    [self registerTableViewCellsWithTableView:[self tableView] bundle:[self resourceBundle]];

    _tableModel = [self buildTableModelForCargo:_cargo];
}
```

we can assume the “cargo” exists during view loading because the “cargo” was passed during construction

# “STATIC”VIEW CONTROLLERS

```
#pragma mark - View Life Cycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self configureNavigationItem:[self navigationItem]];
    [self registerTableViewCellsWithTableView:[self tableView] bundle:[self resourceBundle]];

    _tableModel = [self buildTableModelForCargo:_cargo];
}
```

# “STATIC”VIEW CONTROLLERS

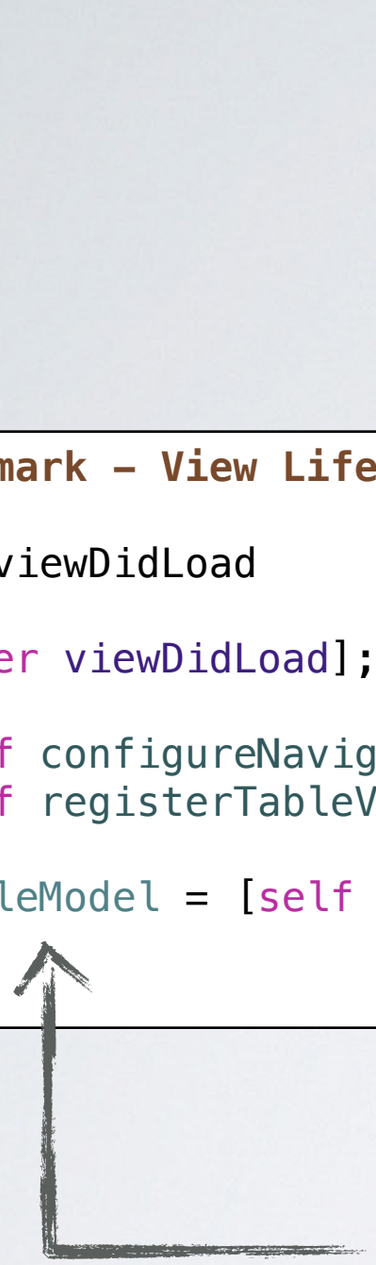
```
#pragma mark - View Life Cycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self configureNavigationItem:[self navigationItem]];
    [self registerTableViewCellsWithTableView:[self tableView] bundle:[self resourceBundle]];

    _tableModel = [self buildTableModelForCargo:_cargo];
}
```

data structure used to make it
*easier* to build the table few

```objc
@implementation BTSCargoViewController

#pragma mark - UITableView Data Source

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return [_tableModel count];
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    NSArray *rows = [self rowsInSection:section];
    return [rows count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
         cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSDictionary *rowInfo = [self rowInfoAtIndexPath:indexPath];

    NSString *cellIdentifier = [rowInfo objectForKey:@"cellIdentifier"];
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:cellIdentifier
                                                            forIndexPath:indexPath];

    [self configureCell:cell forRowInfo:rowInfo];

    return cell;
}

#pragma mark - Other Implementation Details

@end
```

# TABLE MODELS

```
#pragma mark - View Life Cycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self configureNavigationItem:[self navigationItem]];
    [self registerTableViewCellsWithTableView:[self tableView] bundle:[self resourceBundle]];

    _tableModel = [self buildTableModelForCargo:_cargo];
}
```



what is this?

# TABLE MODELS

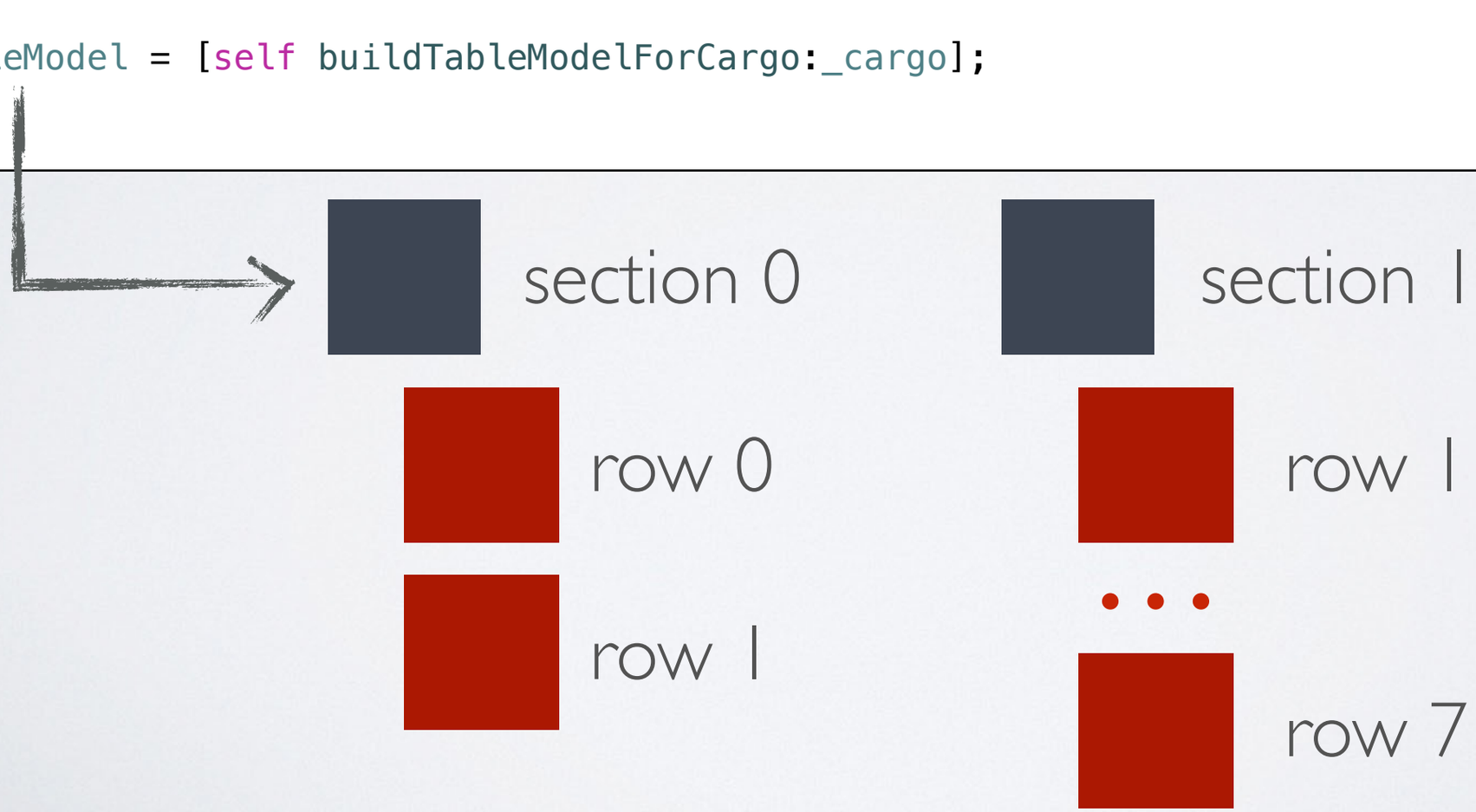
```
#pragma mark - View Life Cycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self configureNavigationItem:[self navigationItem]];
    [self registerTableViewCellsWithTableView:[self tableView] bundle:[self resourceBundle]];

    _tableModel = [self buildTableModelForCargo:_cargo];
}
```



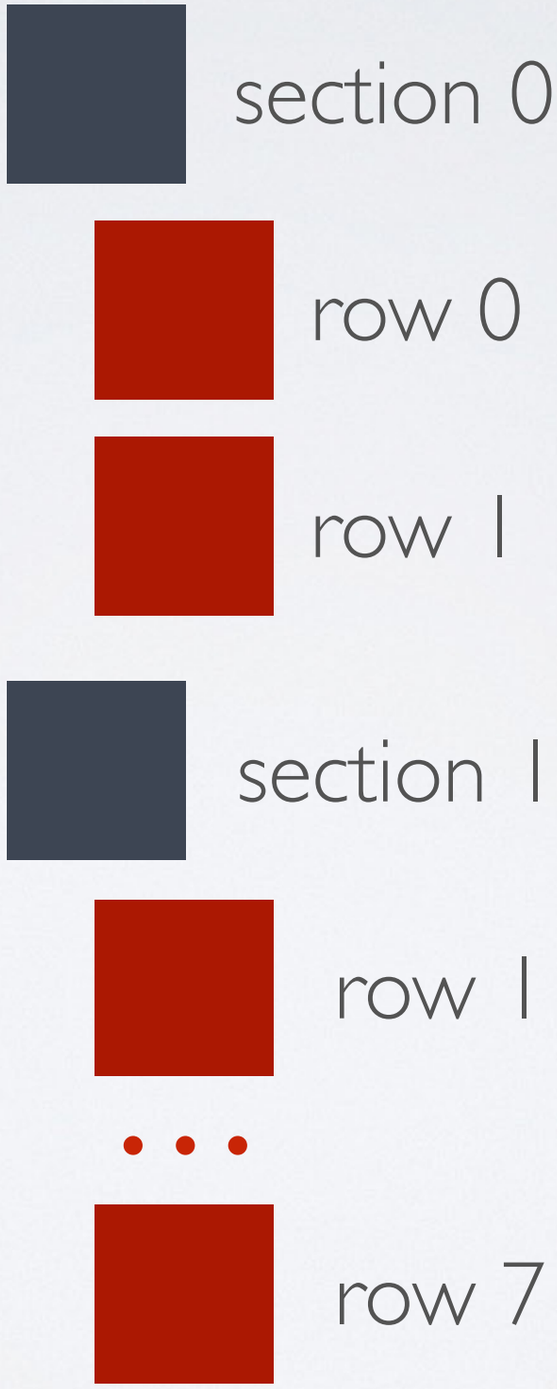
what is this? data binding technique

# TABLE MODELS

```
#pragma mark - View Life Cycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self configureNavigationItem:[self navigationItem]];
    [self registerTableViewCellsWithTableView:[self tableView] bundle:[self resourceBundle]];

    _tableModel = [self buildTableModelForCargo:_cargo];
}
```



# TABLE MODELS



# Example Structure

```
[
    {
        "headerTitle": "Cargo",
        "rows": [
            {
                "keyPath": "cargoID",
                "boundType": "BTSCargo",
                "cellIdentifier": "BTSFormFieldTableViewCell"
            },
            {
                "keyPath": "shipDate",
                "type": "BTSCargo",
                "cellIdentifier": "BTSDateTimeTableViewCell",
                "formatter": "BTSLocalizedDateTimeFormatter"
            }
        ]
    },
    {
        "headerTitle": "Destination",
        "rows": [
            {
                "keyPath": "cargo.destination.destinationID",
                "boundType": "BTSDestination",
                "cellIdentifier": "BTSFormFieldTableViewCell"
            },
            {
                "keyPath": "cargo.destination.cityCode",
                "type": "BTSDestination",
                "cellIdentifier": "BTSFormFieldTableViewCell",
                "transformer": "BTSCityCodeValueTransformer"
            }
        ]
    }
]
```

# TABLE MODELS

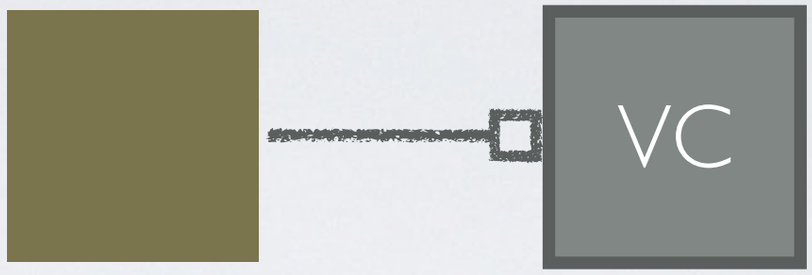
- "data binding" technique
- can use basic arrays and dictionaries
- should use custom, type safe data structures
- **extremely** powerful with KVO
- details are beyond the scope of this presentation

# "DYNAMIC" VIEW CONTROLLERS

- view or edit "entity"
- mutable
- simple public api
- little more work to implement
- separate "storage" from "behavior"

# “DYNAMIC”VIEW CONTROLLERS

- view or edit “entity”
- mutable
- simple public api
- little more work to implement
- separate “storage” from “behavior”




```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readwrite) BTSCargo *cargo;

@end
```

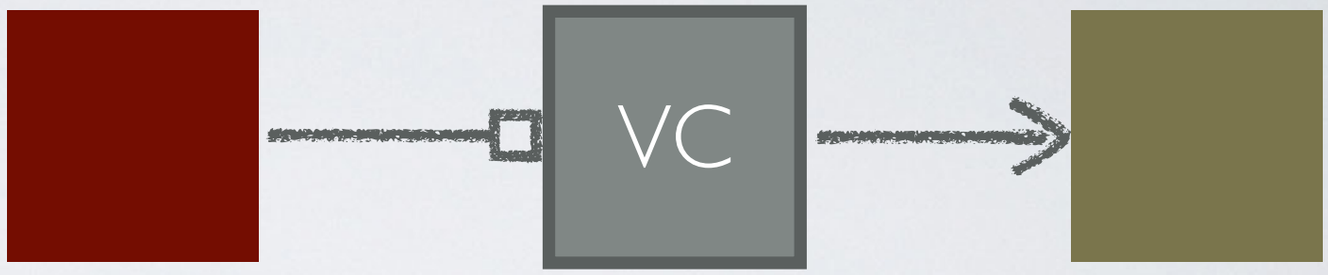
# "DYNAMIC" VIEW CONTROLLERS


- view or edit "entity"
- mutable
- simple public api
- little more work to implement
- separate "storage" from "behavior"

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readwrite) BTSCargo *cargo;

@end
```

# "DYNAMIC" VIEW CONTROLLERS

- view or edit "entity"
- mutable
- simple public api
- little more work to implement
- separate "storage" from "behavior"




```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readwrite) BTSCargo *cargo;

@end
```

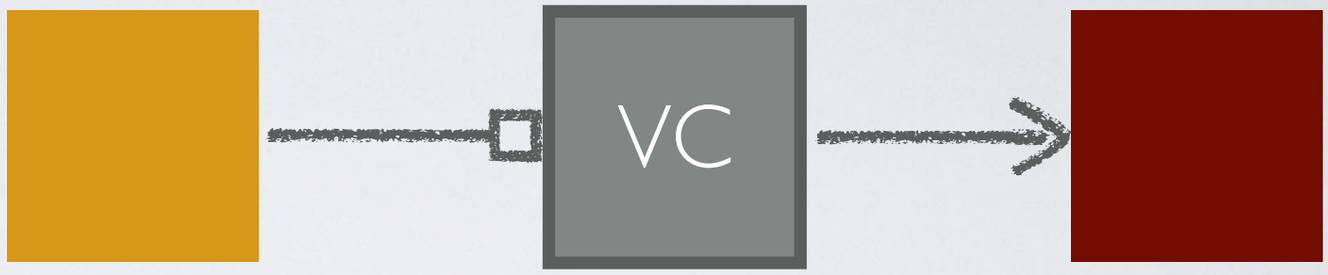
# "DYNAMIC" VIEW CONTROLLERS


- view or edit "entity"
- mutable
- simple public api
- little more work to implement
- separate "storage" from "behavior"

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readwrite) BTSCargo *cargo;

@end
```

# "DYNAMIC" VIEW CONTROLLERS

- view or edit "entity"
- mutable
- simple public api
- little more work to implement
- separate "storage" from "behavior"




```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readwrite) BTSCargo *cargo;

@end
```

# "DYNAMIC" VIEW CONTROLLERS


- view or edit "entity"
- mutable
- simple public api
- little more work to implement
- separate "storage" from "behavior"

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readwrite) BTSCargo *cargo;

@end
```

# STORAGE VS BEHAVIOR

```objc
@implementation BTSCargoViewController

static void *kCargoContext = &kCargoContext;

#pragma mark - Object Life Cycle

- (instancetype)init
{
    self = [super initWithStyle:UITableViewStyleGrouped];
    if (self) {
        NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;
        [self addObserver:self forKeyPath:@"cargo" options:options context:kCargoContext];
    }
    return self;
}
```

# STORAGE VS BEHAVIOR

```objc
@implementation BTSCargoViewController

static void *kCargoContext = &kCargoContext;

#pragma mark - Object Life Cycle

- (instancetype)init
{
    self = [super initWithStyle:UITableViewStyleGrouped];
    if (self) {
        NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;
        [self addObserver:self forKeyPath:@"cargo" options:options context:kCargoContext];
    }
    return self;
}
```

use KVO to react to changes to self

separate “behavior” from “storage”

```
#ifdef DEBUG
#define PROPERTY(propertyName) NSStringFromSelector(@selector(propertyName))
#else
#define PROPERTY(propertyName) @#propertyName
#endif
```

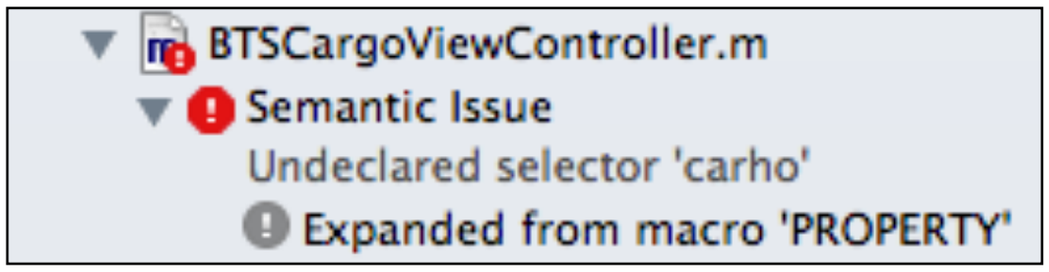
```
@implementation BTSCargoViewController

static void *kCargoContext = &kCargoContext;

#pragma mark - Object Life Cycle

- (instancetype)init
{
    self = [super initWithStyle:UITableViewStyleGrouped];
    if (self) {
        NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;
        [self addObserver:self forKeyPath:PROPERTY(cargo) options:options context:kCargoContext];
    }
    return self;
}
```

macro provides semi-type-safety

`-Wundeclared-selector`

```objc
#ifdef DEBUG
#define PROPERTY(propertyName) NSStringFromSelector(@selector(propertyName))
#else
#define PROPERTY(propertyName) @#propertyName
#endif
```

```objc
@implementation BTSCargoViewController

static void *kCargoContext = &kCargoContext;

#pragma mark - Object Life Cycle

- (instancetype)init
{
    self = [super initWithStyle:UITableViewStyleGrouped];
    if (self) {
        NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;
        [self addObserver:self forKeyPath:PROPERTY(carho) options:options context:kCargoContext];
    }
    return self;
}
```




macro provides semi-type-safety

`-Wundeclared-selector`

```objc
#ifdef DEBUG
#define PROPERTY(propertyName) NSStringFromSelector(@selector(propertyName))
#else
#define PROPERTY(propertyName) @#propertyName
#endif
```

```objc
@implementation BTSCargoViewController

static void *kCargoContext = &kCargoContext;

#pragma mark - Object Life Cycle

- (instancetype)init
{
    self = [super initWithStyle:UITableViewStyleGrouped];
    if (self) {
        NSKeyValueObservingOptions options = NSKeyValueObservingOptionNew;
        [self addObserver:self forKeyPath:PROPERTY(cargo) options:options context:kCargoContext];
    }
    return self;
}

- (void)dealloc
{
    [self removeObserver:self forKeyPath:PROPERTY(cargo) context:kCargoContext];
}
```

what is added must be removed

# CARGO “DID” CHANGE

```objc
#pragma mark - Cargo Observation

- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context
{
    if (context == kCargoContext) {
        [self handleChangeInCargo:_cargo];
    } else {
        [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
    }
}

- (void)handleChangeInCargo:(BTSCargo *)cargo
{
    if ([self isViewLoaded]) {
        // animate changes for new “cargo”
    }
}
```

no need to do anything if the user cannot see the cargo

will handle this case in `viewDidLoad`

# VIEW INITIAL CARGO

```
#pragma mark - View Life Cycle

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self configureNavigationItem:[self navigationItem]];
    [self registerTableViewCellsWithTableView:[self tableView] bundle:[self resourceBundle]];

    [self handleChangeInCargo:_cargo];
}
```

view displays “No Cargo” if cargo is nil

view controller simply waits for call to **`setCargo:`**

# IF YOU SEE…

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readonly) BTSCargo *cargo;

- (instancetype)initWithCargo:(BTSCargo *)cargo;

@end
```

# IF YOU SEE…

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readonly) BTSCargo *cargo;

- (instancetype)initWithCargo:(BTSCargo *)cargo;

@end
```

designed to display a single instance

# IF YOU SEE…

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readonly) BTSCargo *cargo;

- (instancetype)initWithCargo:(BTSCargo *)cargo;

@end
```

designed to display a single instance

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readwrite) BTSCargo *cargo;

@end
```

# IF YOU SEE…

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readonly) BTSCargo *cargo;

- (instancetype)initWithCargo:(BTSCargo *)cargo;

@end
```

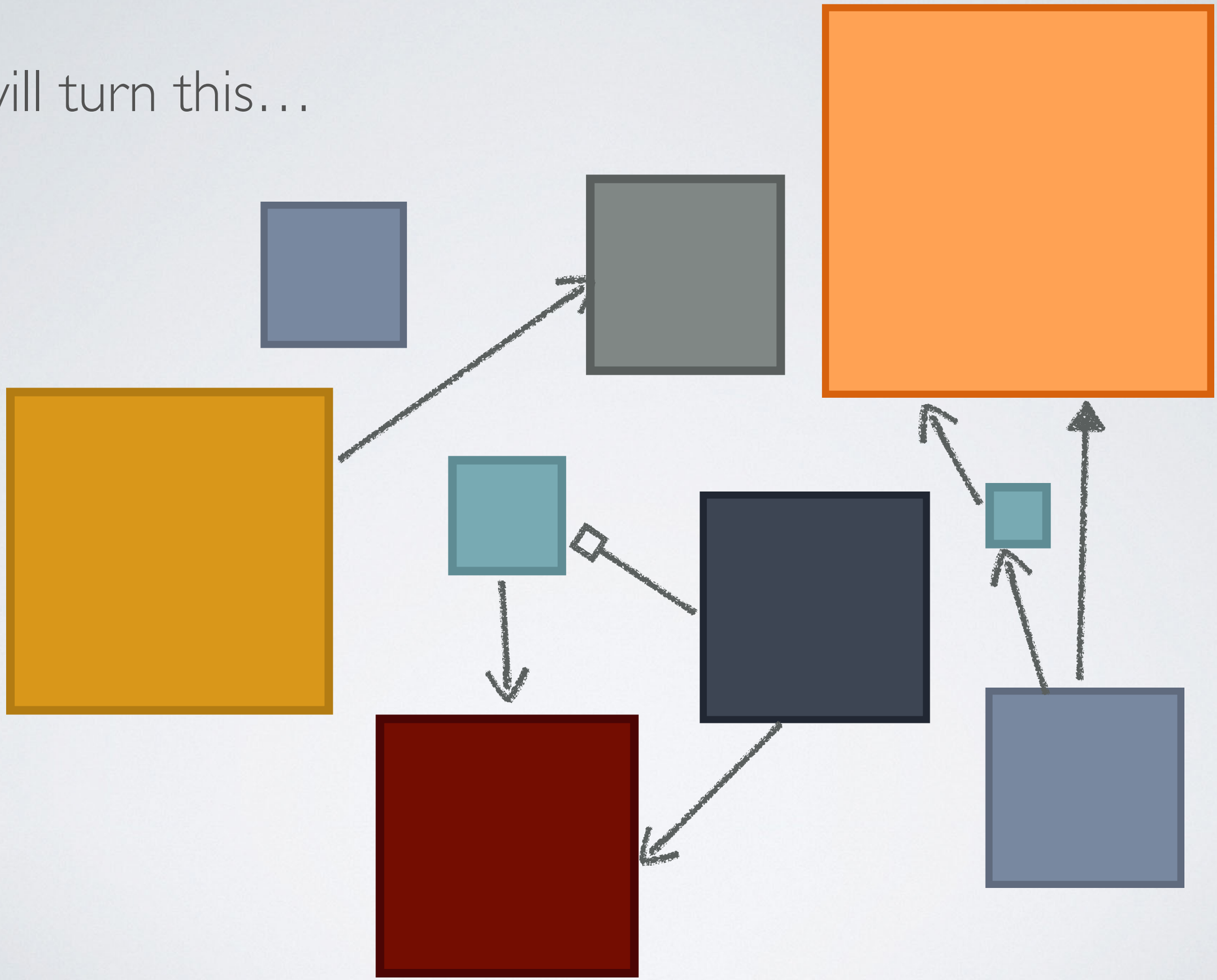
designed to display a single instance

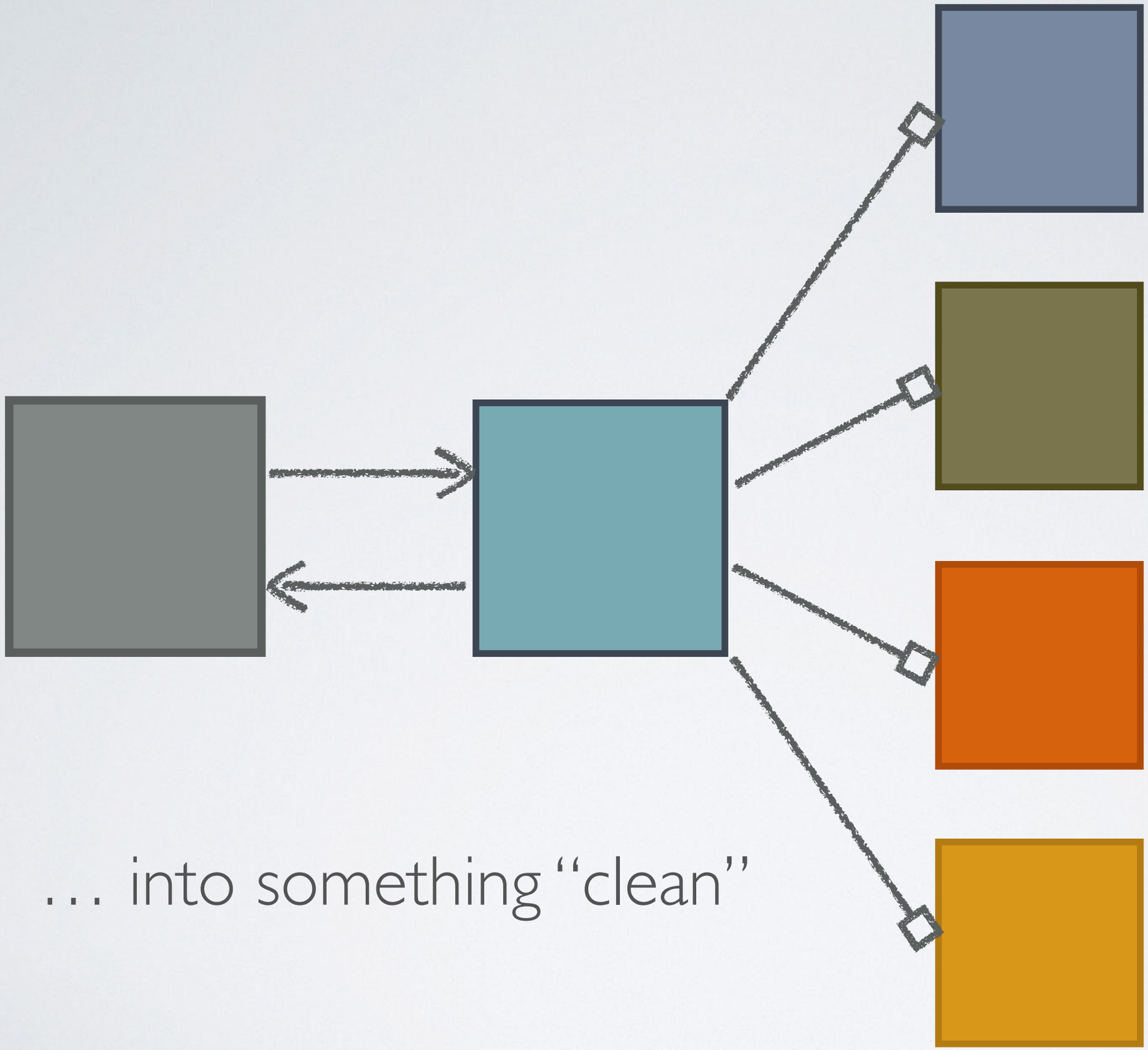
designed to change what the user sees

```
@class BTSCargo;

@interface BTSCargoViewController : UITableViewController

@property (nonatomic, strong, readwrite) BTSCargo *cargo;

@end
```

# ASYNCHRONOUS WORKFLOWS

will turn this…





… into something “clean”

# APPS HAVE WORKFLOWS

# WORKFLOWS

# WORKFLOWS

- sign in users

# WORKFLOWS

- sign in users
- download large files

# WORKFLOWS

- sign in users
- download large files
- stream data into data stores

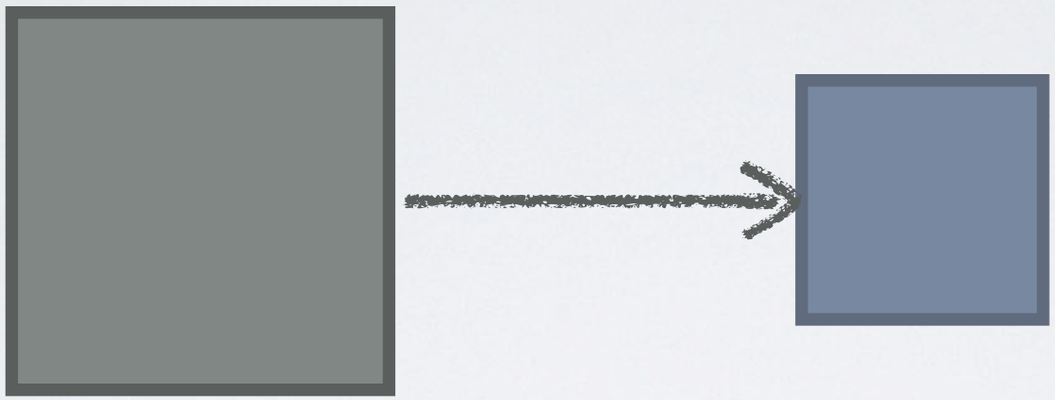
# WORKFLOWS

- sign in users
- download large files
- stream data into data stores
- execute searches

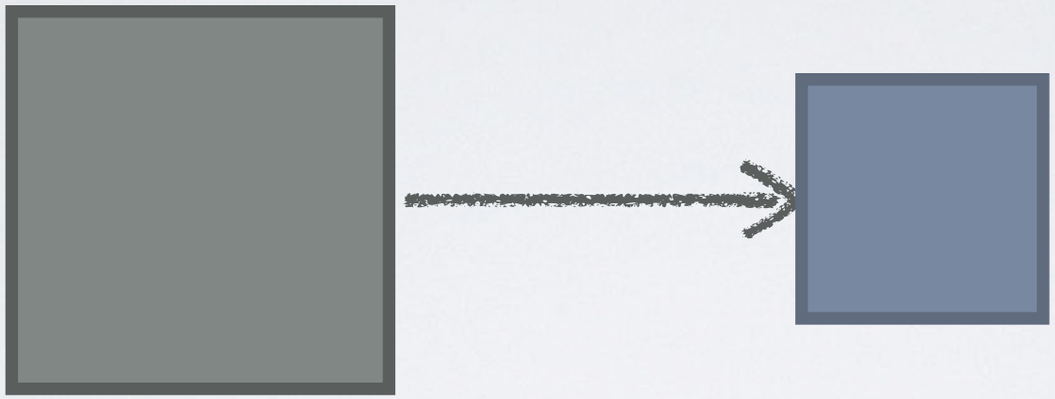
# WORKFLOWS

- sign in users
- download large files
- stream data into data stores
- execute searches
- sign out users

# WORKFLOWS ARE ASYNCHRONOUS

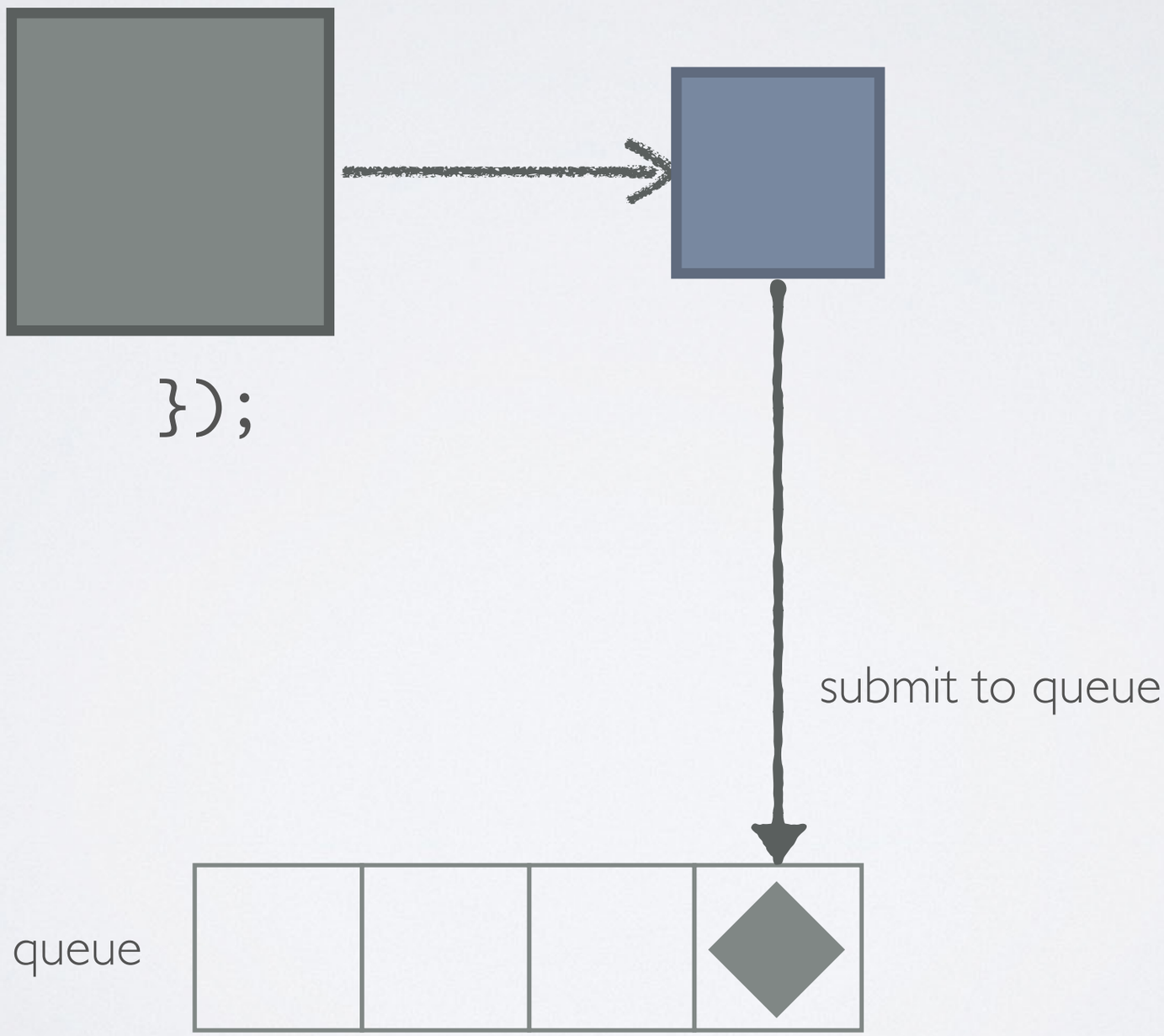


synchronous call to a body of code

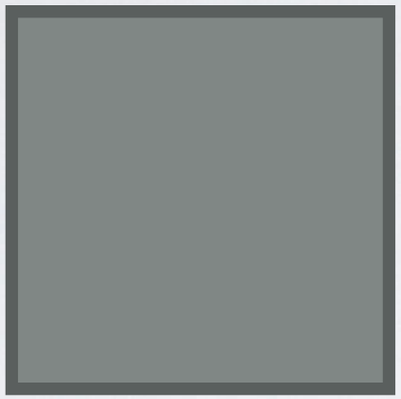


you say, “let’s make this asynchronous”

```
dispatch_async(_backgroundQueue, ^{
```

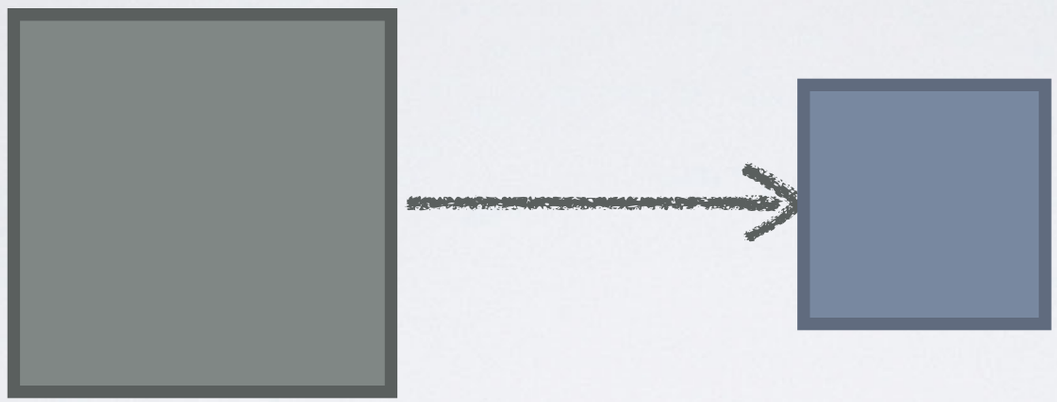



```
});
```



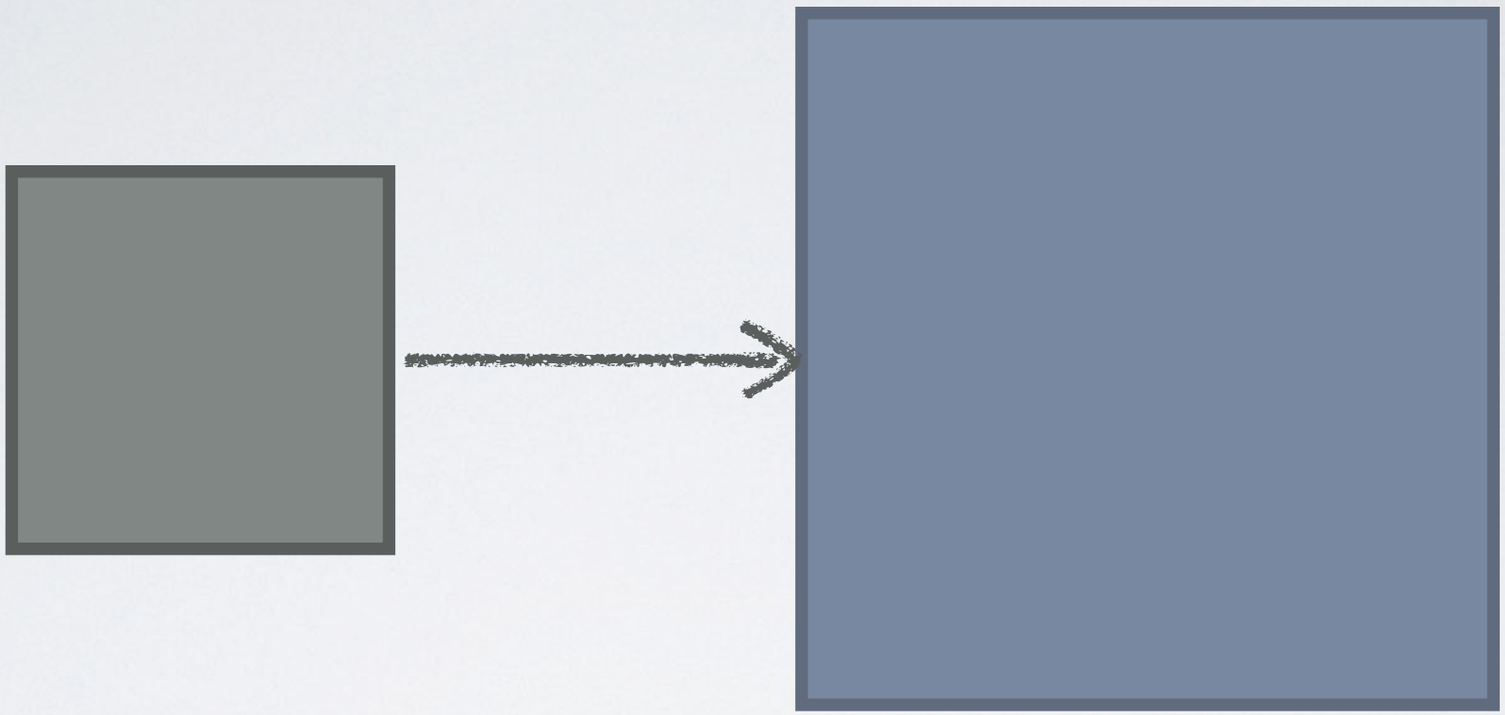
gcd queue

```
dispatch_async(_backgroundQueue, ^{
```

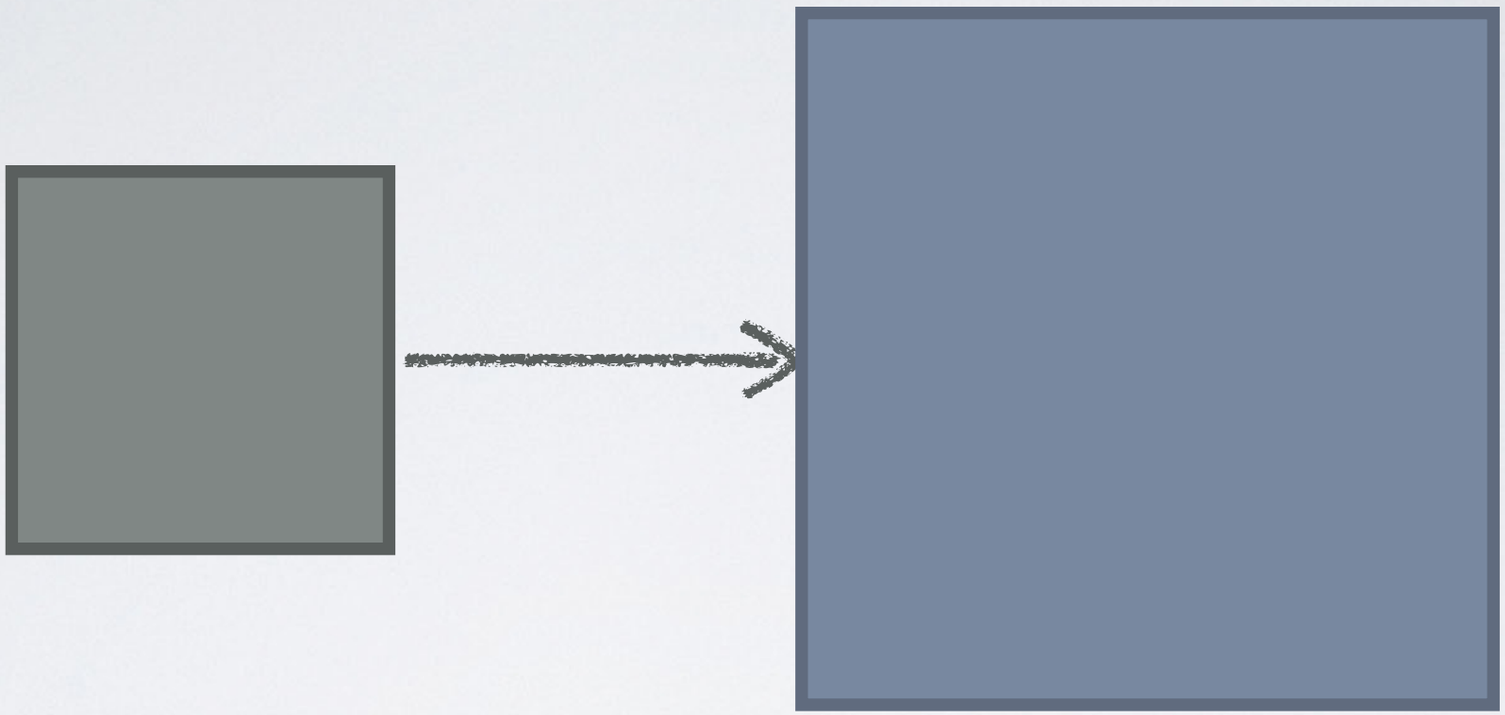


```
});
```

works well if operations are very short lived

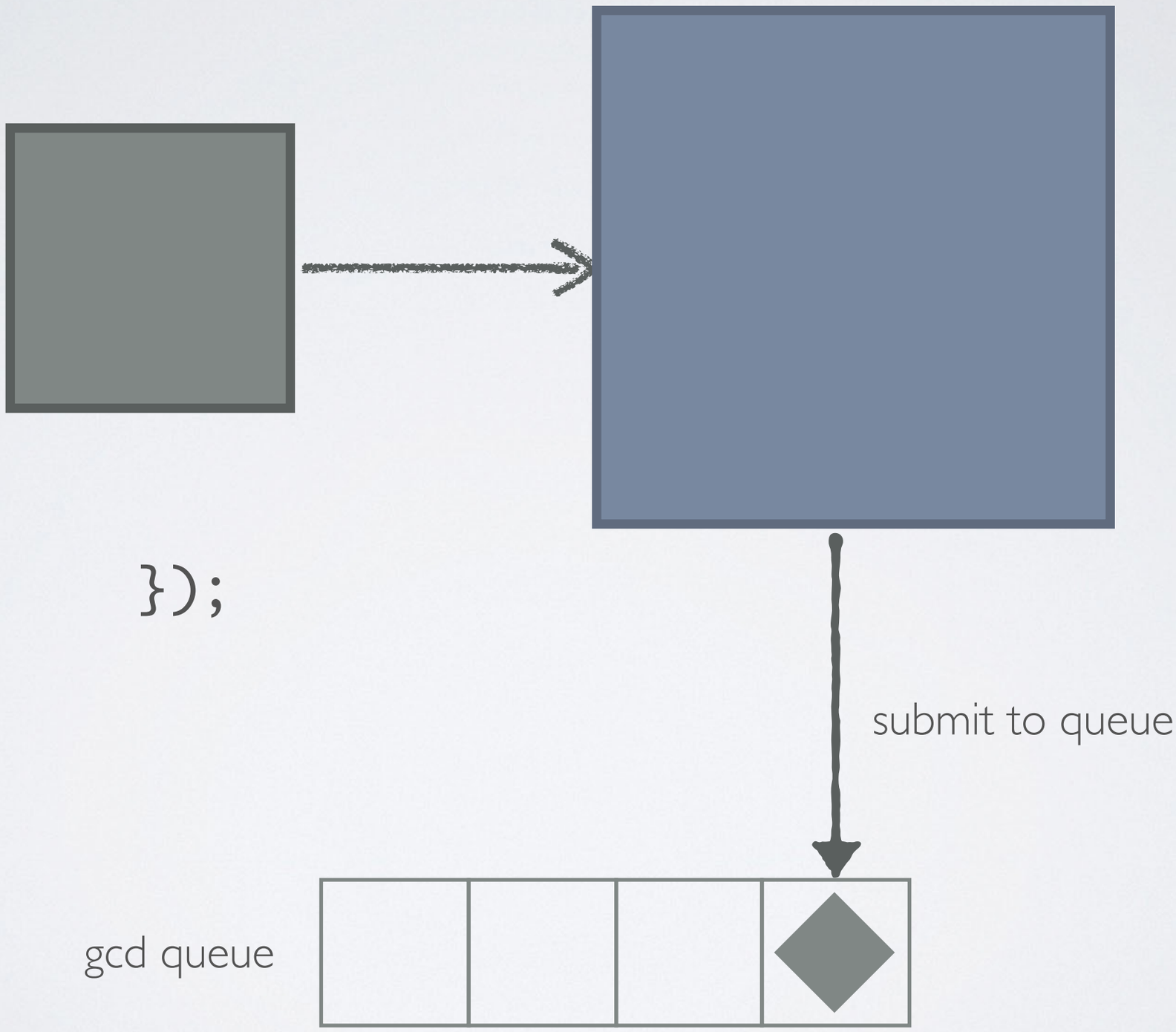


synchronous call to a larger body of code



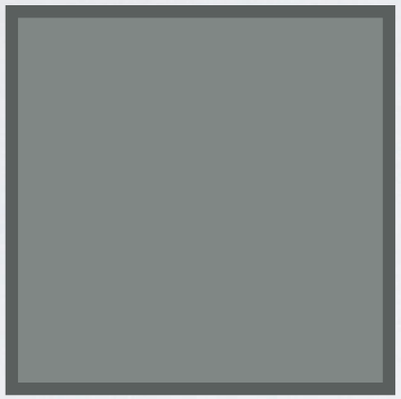
you say, “let’s make this asynchronous”

```
dispatch_async(_backgroundQueue, ^{
```



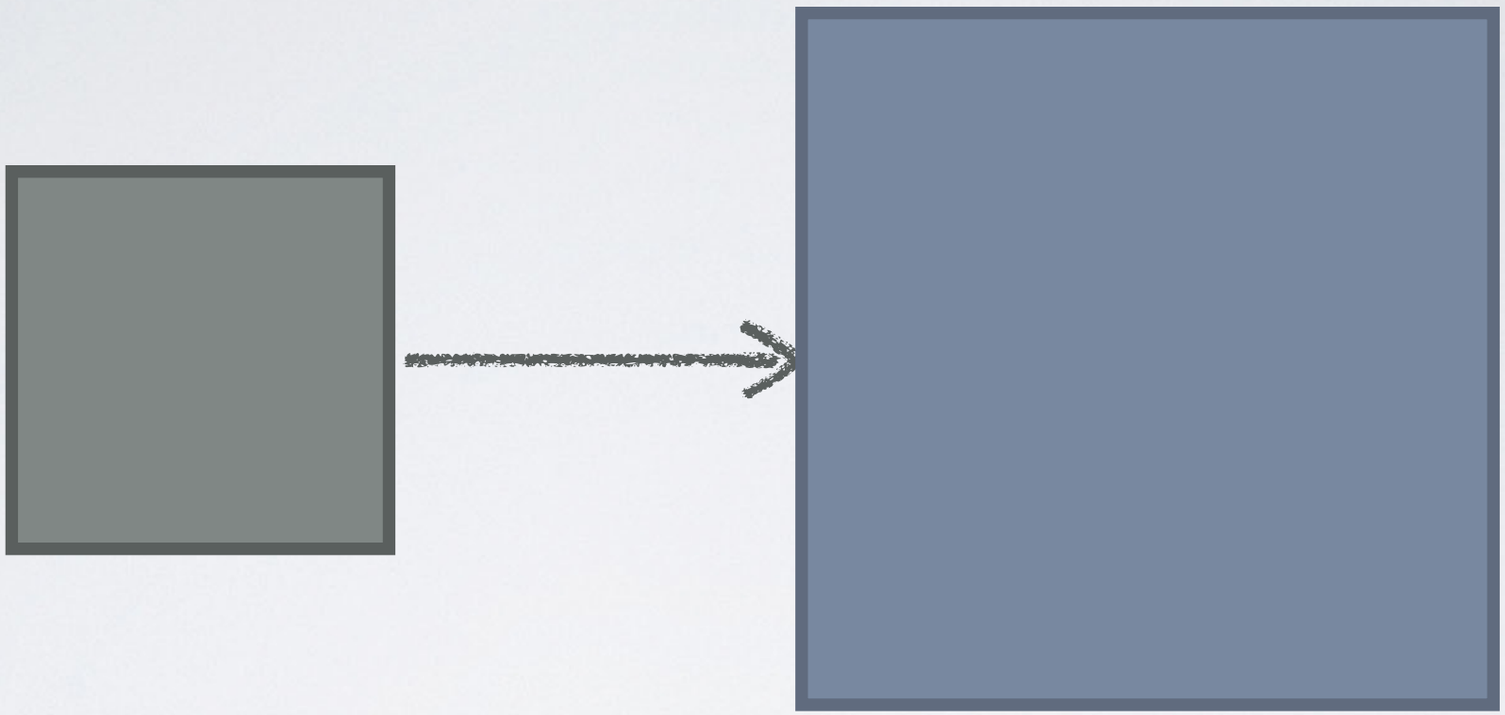

```
});
```

```
dispatch_async(_backgroundQueue, ^{
```



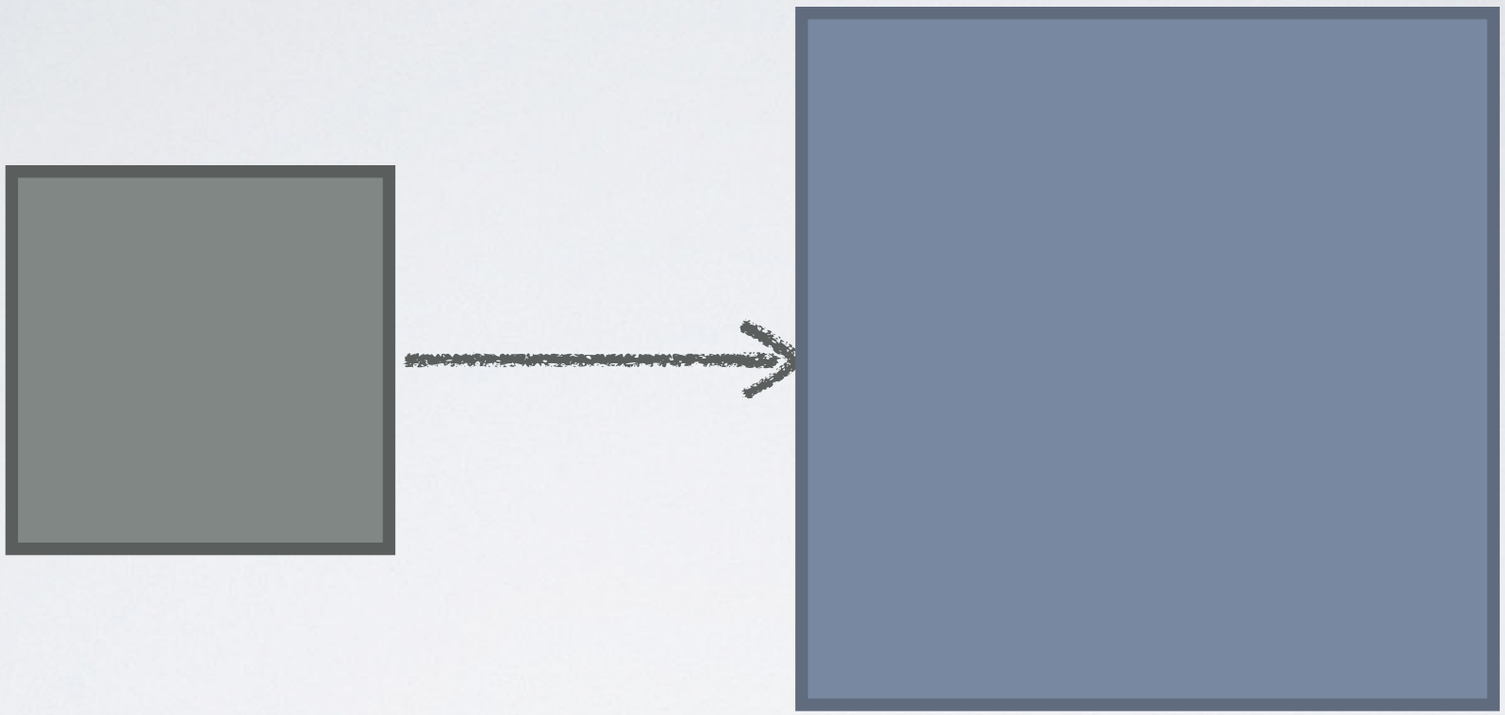
```
});
```

```
dispatch_async(_backgroundQueue, ^{
```



```
});
```

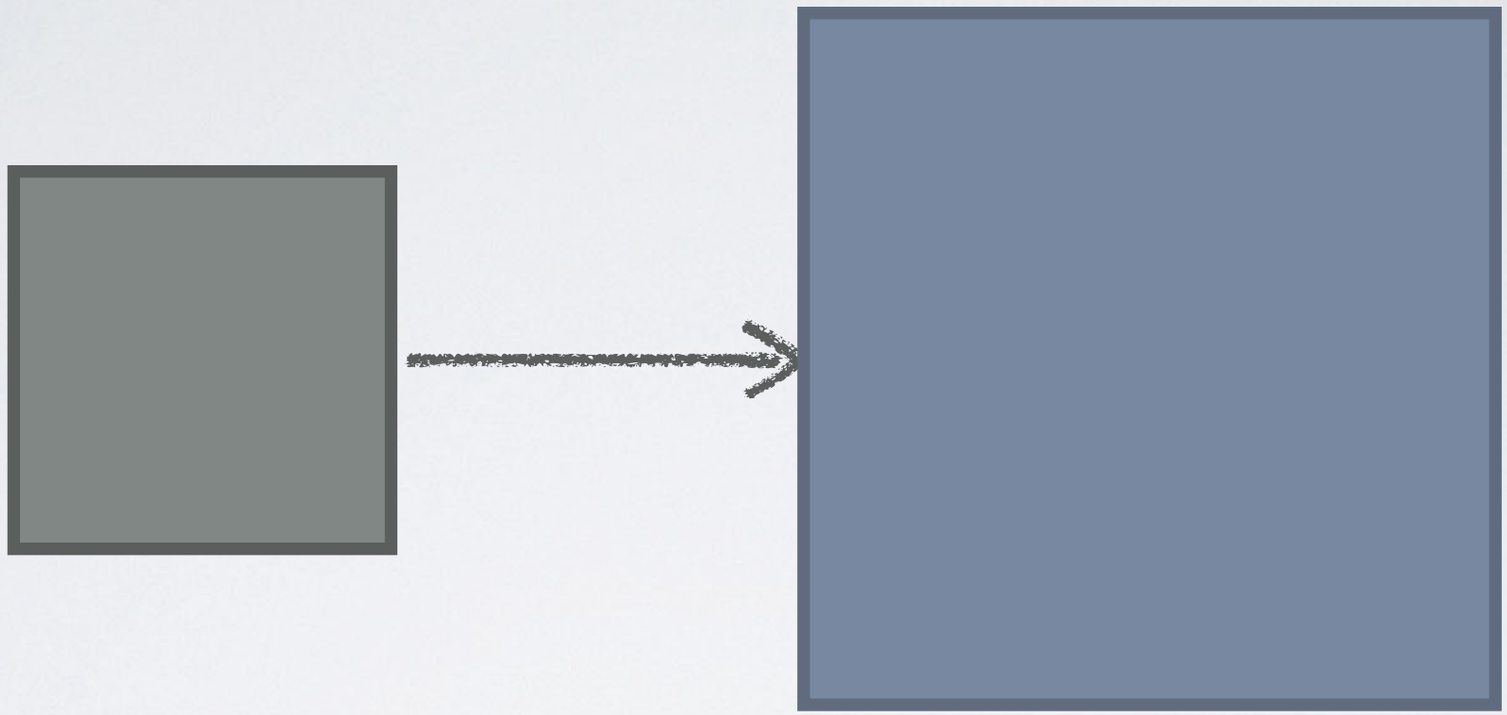
```
dispatch_async(_backgroundQueue, ^{
```



```
});
```

your boss says, “users want to cancel”

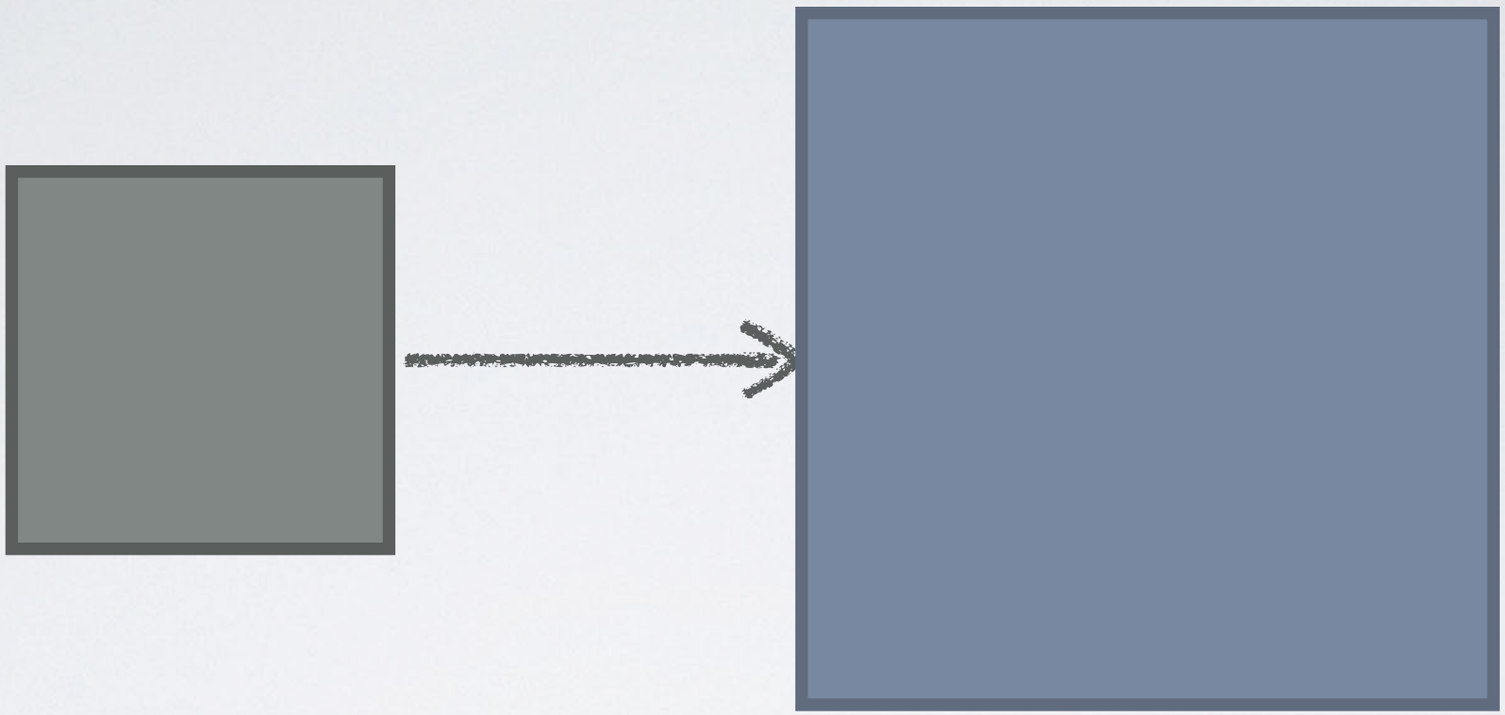
```
dispatch_async(_backgroundQueue, ^{
```



```
});
```

your boss says, “users want to cancel”
you say, “no problem!”

```
dispatch_async(_backgroundQueue, ^{
```

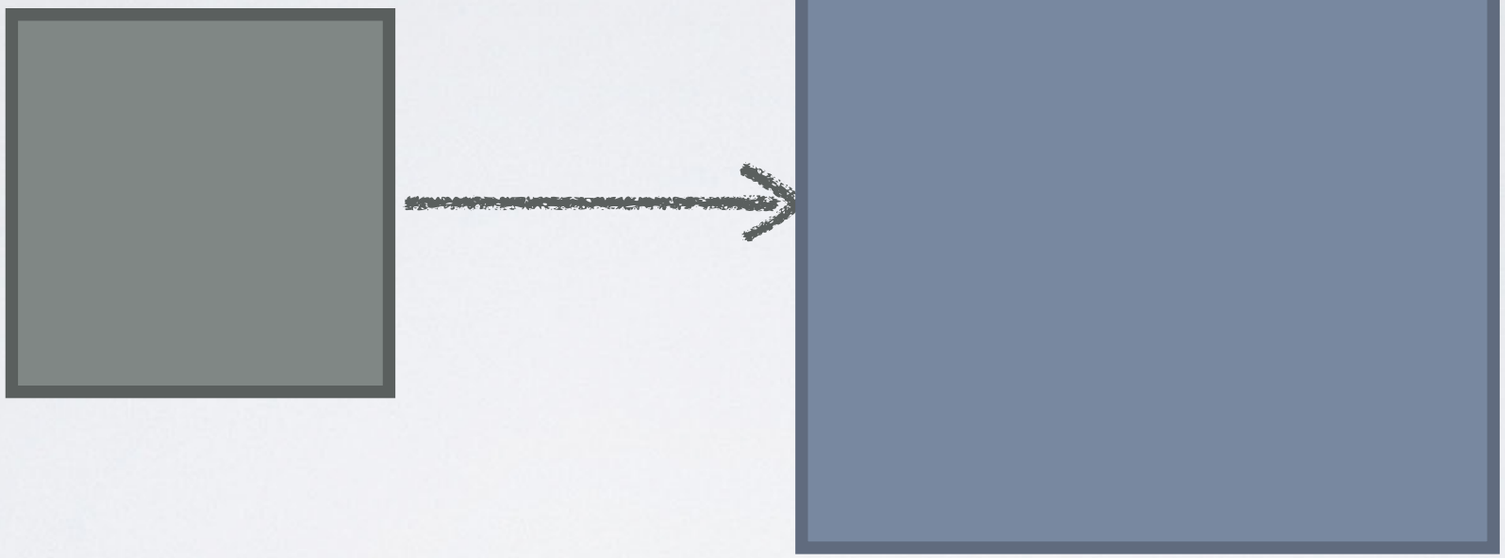


```
});
```

you start adding cancel API

```
dispatch_async(_backgroundQueue, ^{
```

cancel

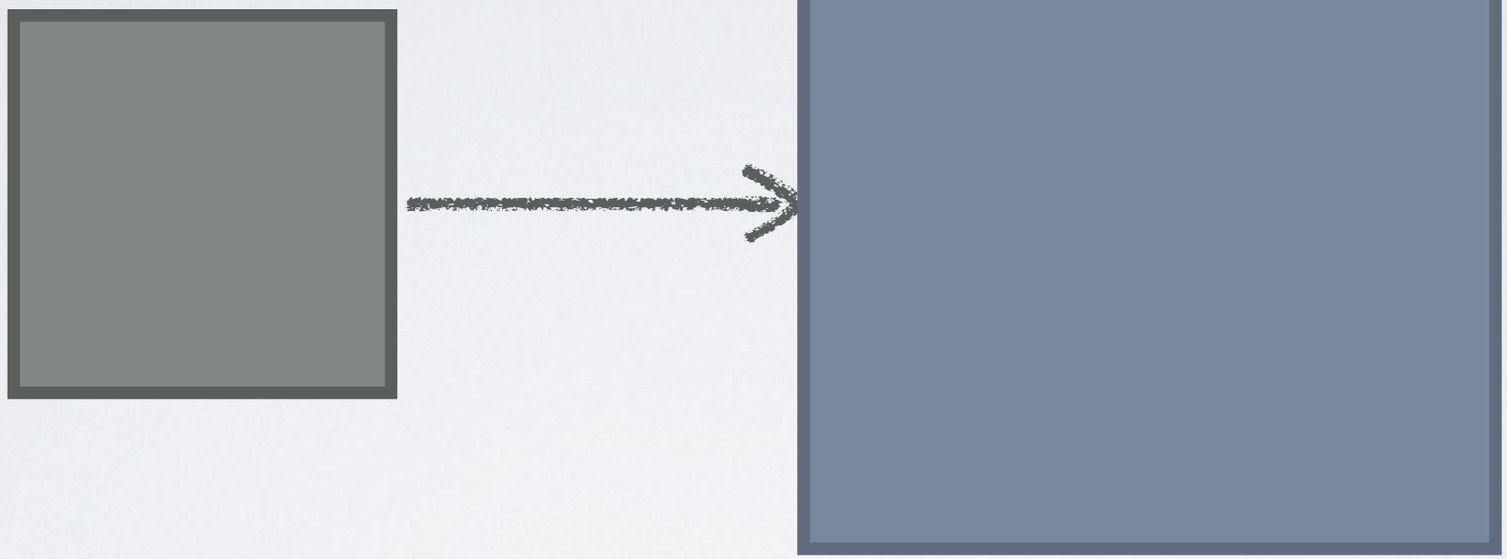


```
});
```

you start adding cancel API

```
dispatch_async(_backgroundQueue, ^{
```

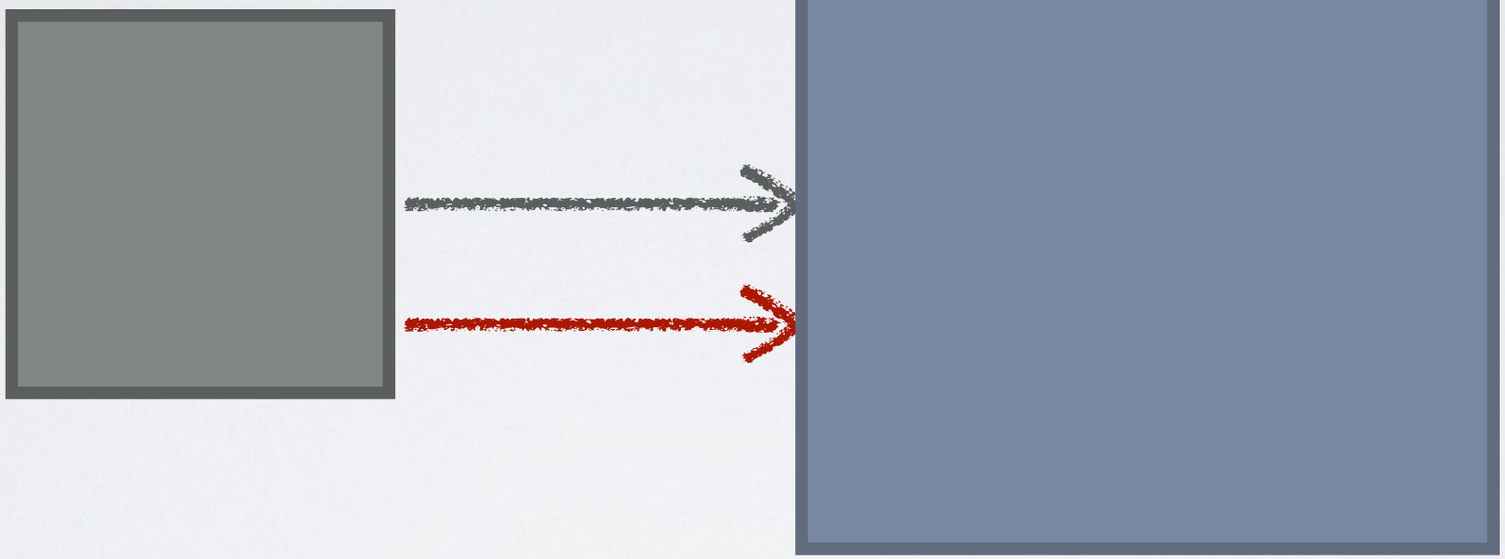
cancel



```
});
```

```
dispatch_async(_backgroundQueue, ^{
```

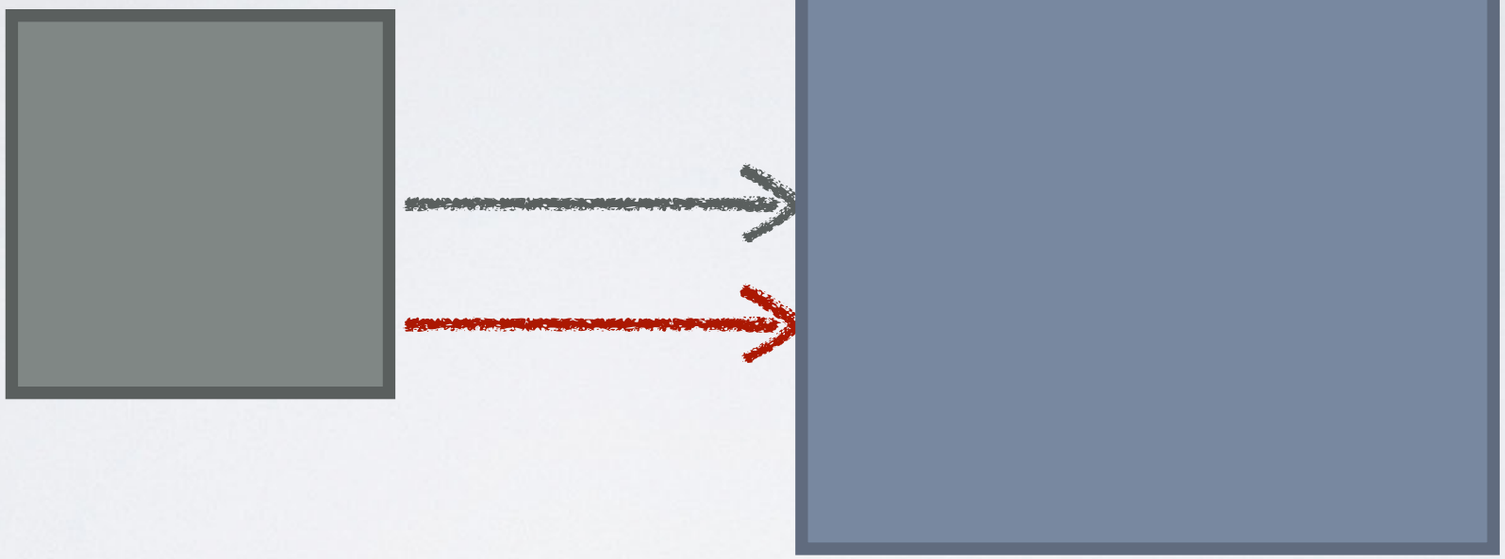
cancel



```
});
```

```
dispatch_async(_backgroundQueue, ^{
```

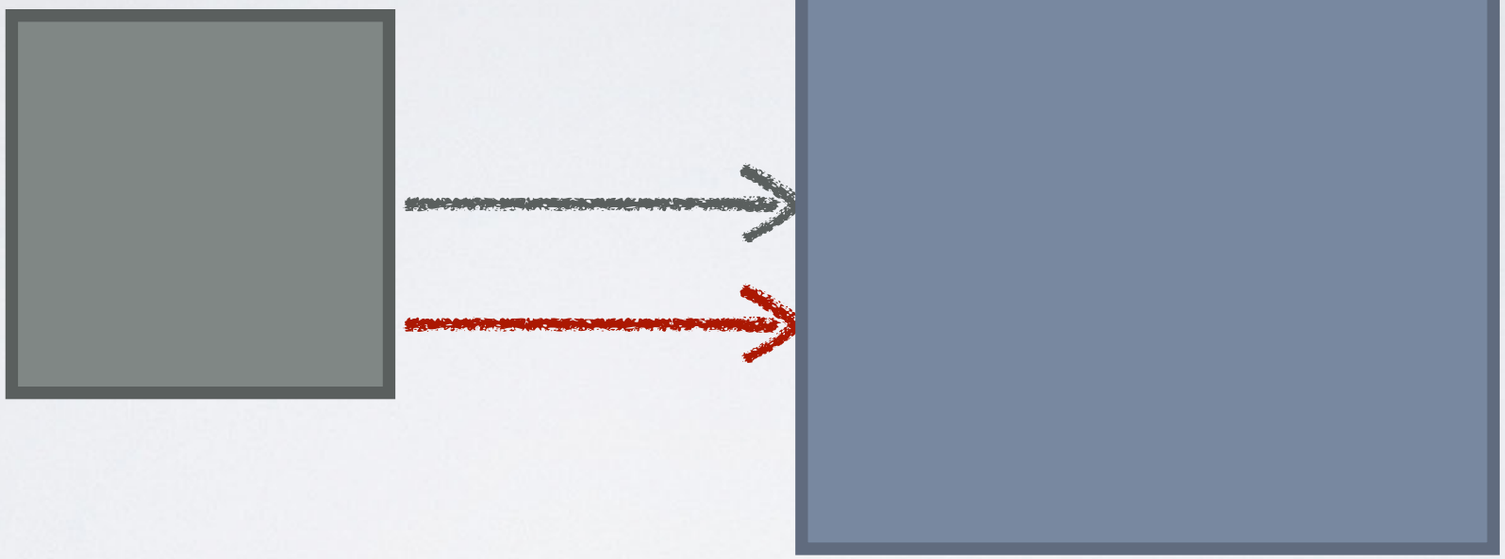
cancel



```
});
```

```
dispatch_async(_backgroundQueue, ^{
```

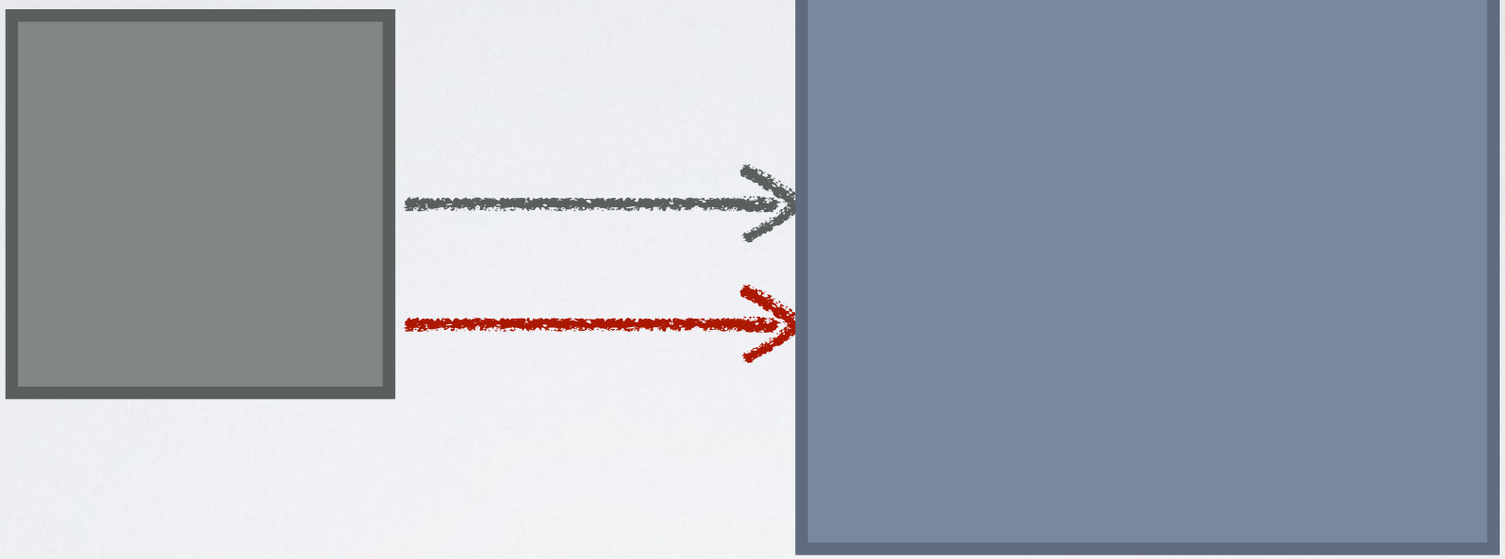
cancel



```
});
```

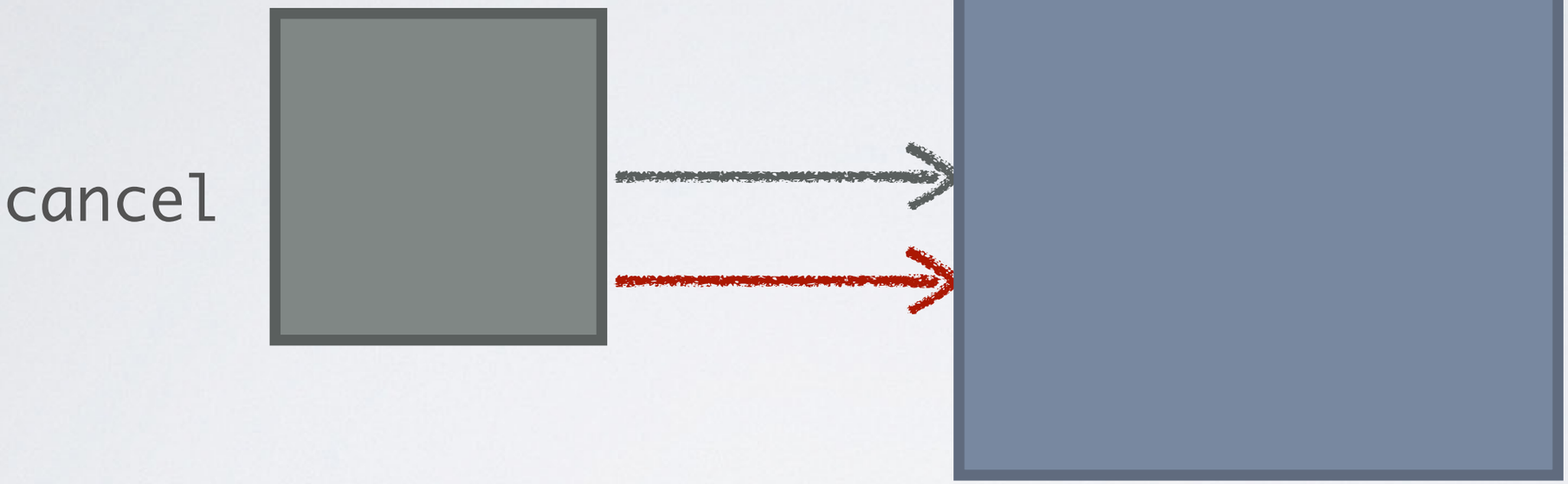
```
dispatch_async(_backgroundQueue, ^{
```

cancel



```
});
```

```
dispatch_async(_backgroundQueue, ^{
```

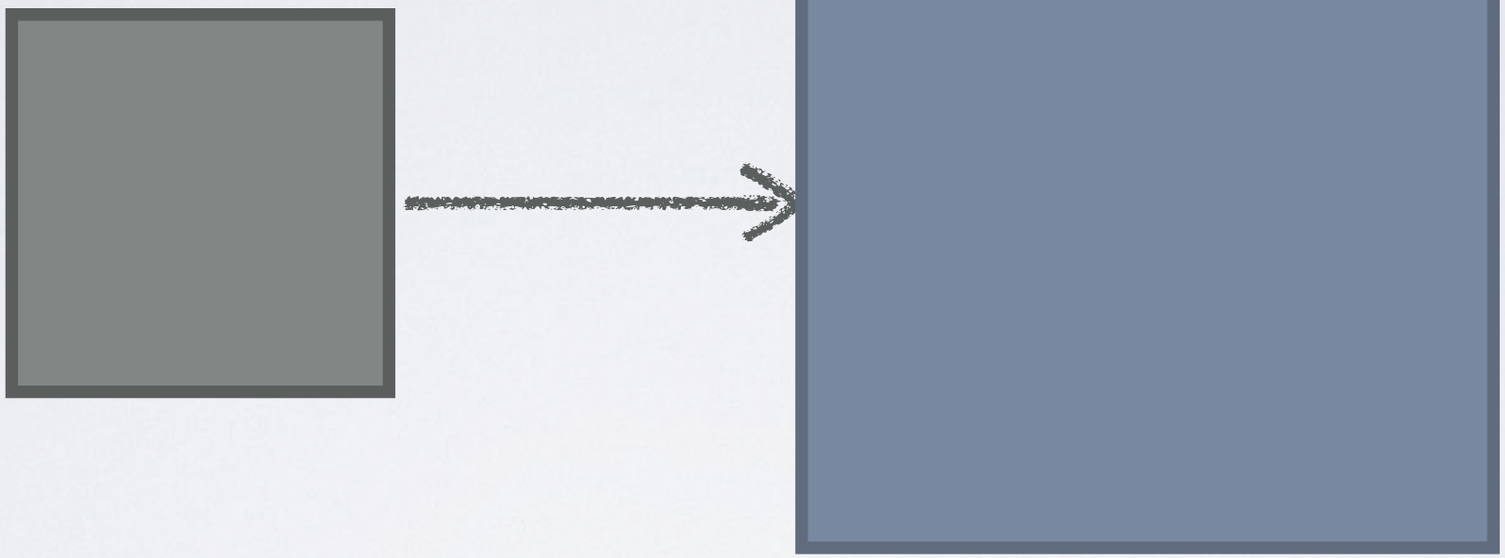



```
});
```

you say, “huh? i can’t stop it!”

```
dispatch_async(_backgroundQueue, ^{
```

cancel



```
});
```

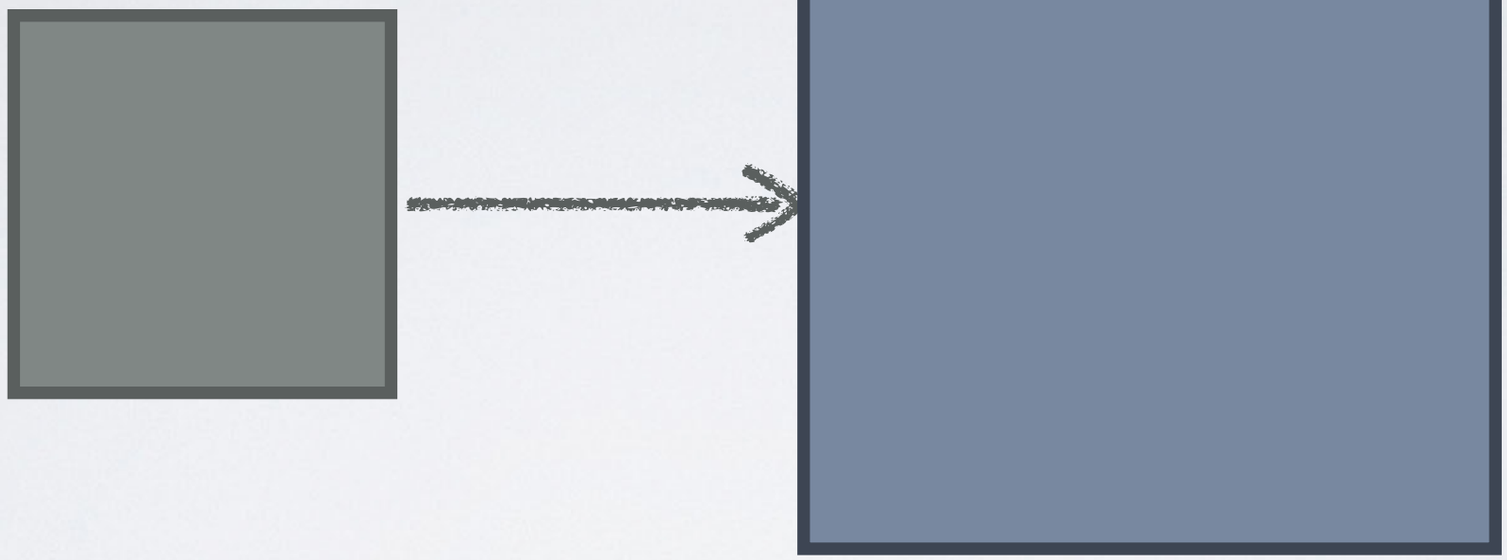
you say, “huh? i can’t stop it!”

# WORKFLOWS MUST BE CANCELABLE

# NSOperation
# &
# NSOperationQueue

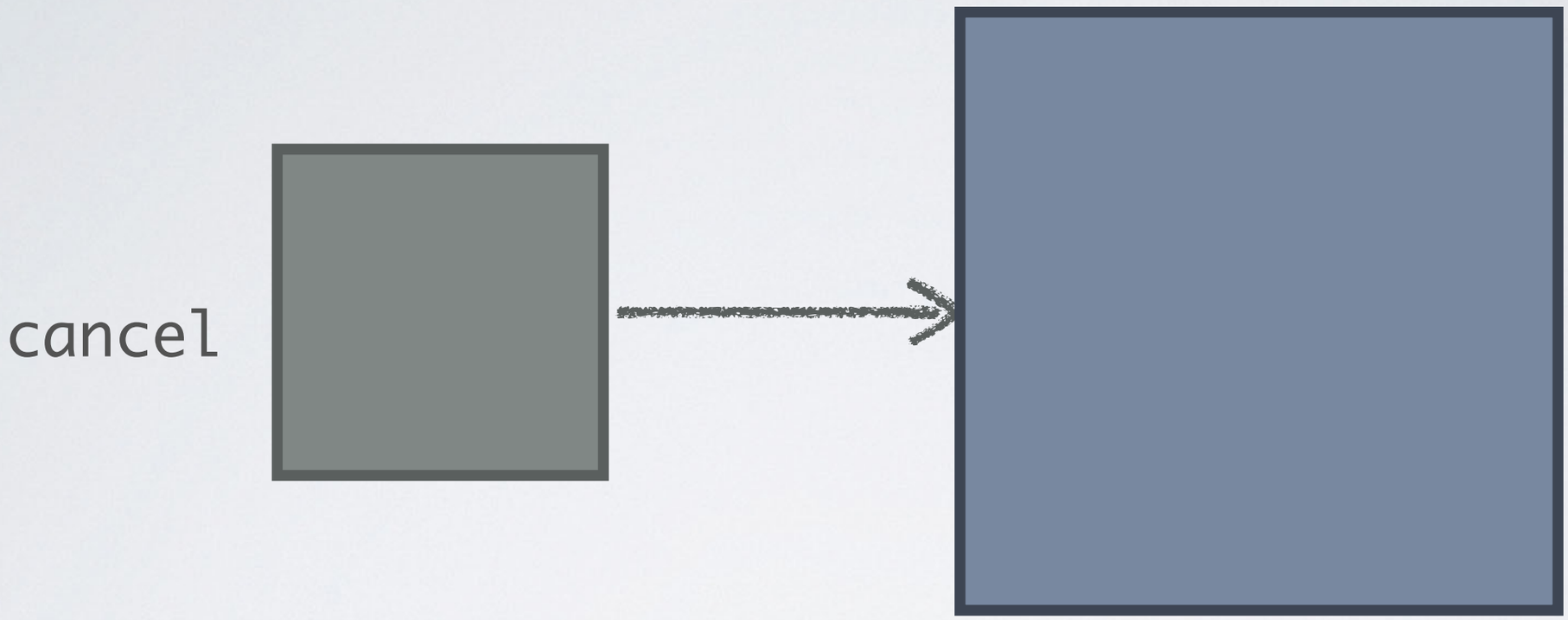
```
dispatch_async(_backgroundQueue, ^{
```

cancel



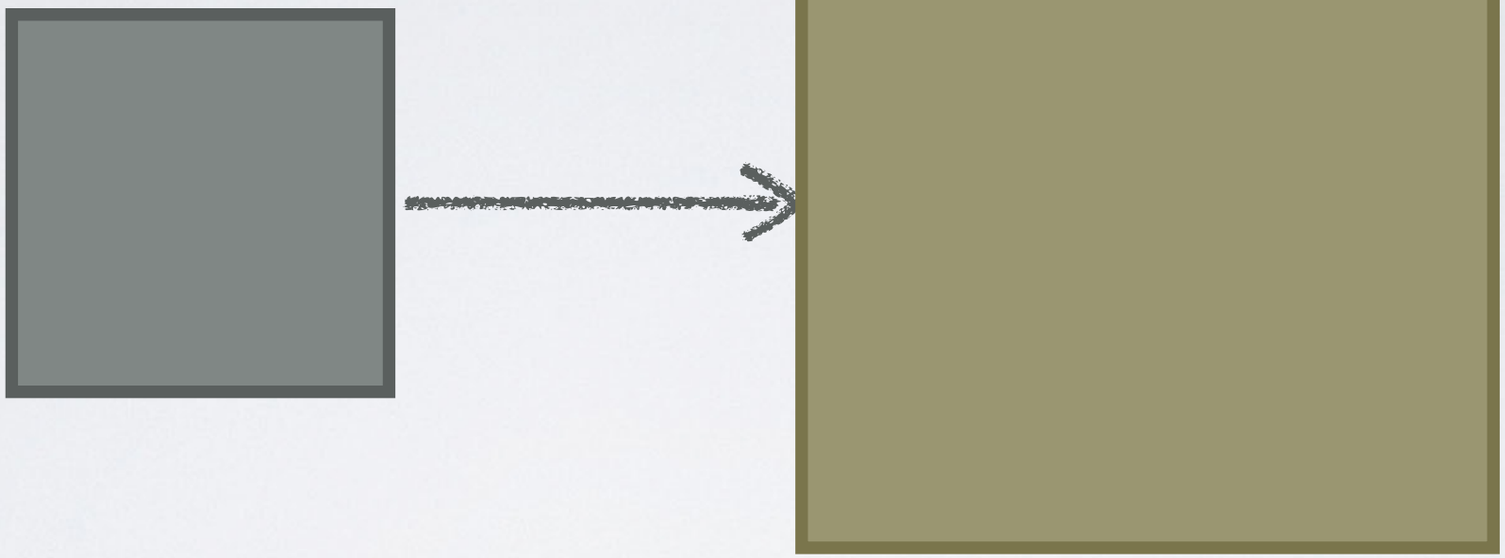
```
});
```

transition to operations



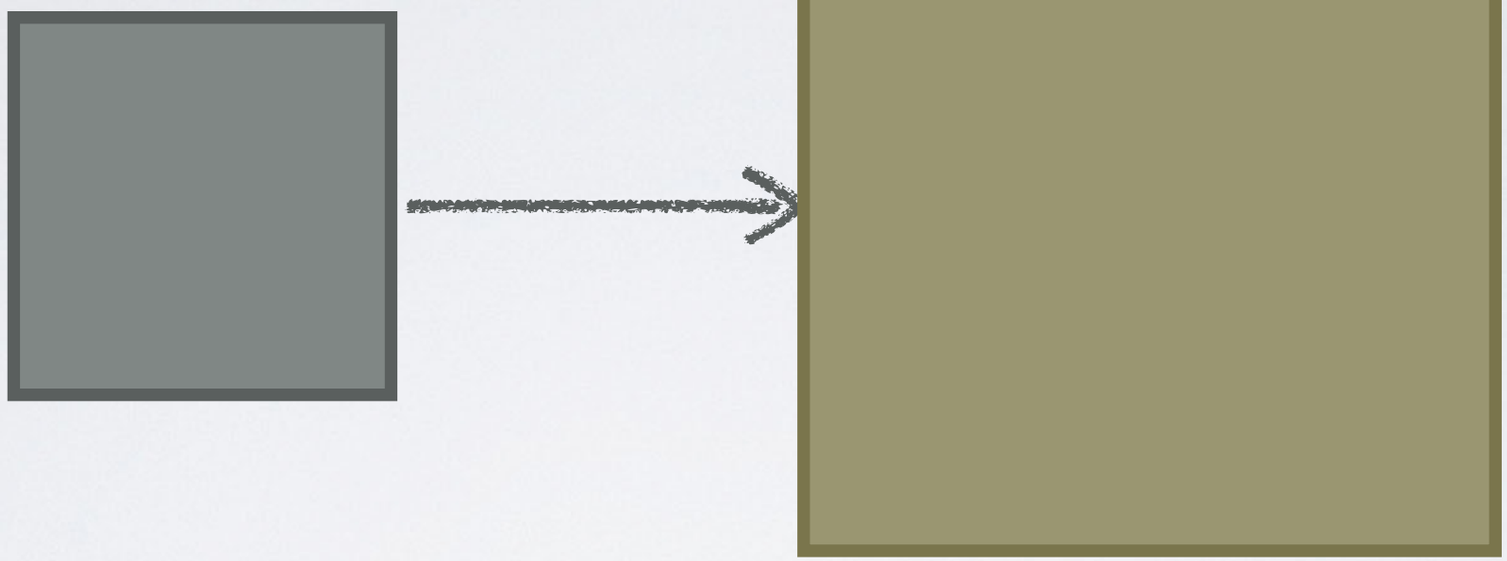

transition to operations

cancel



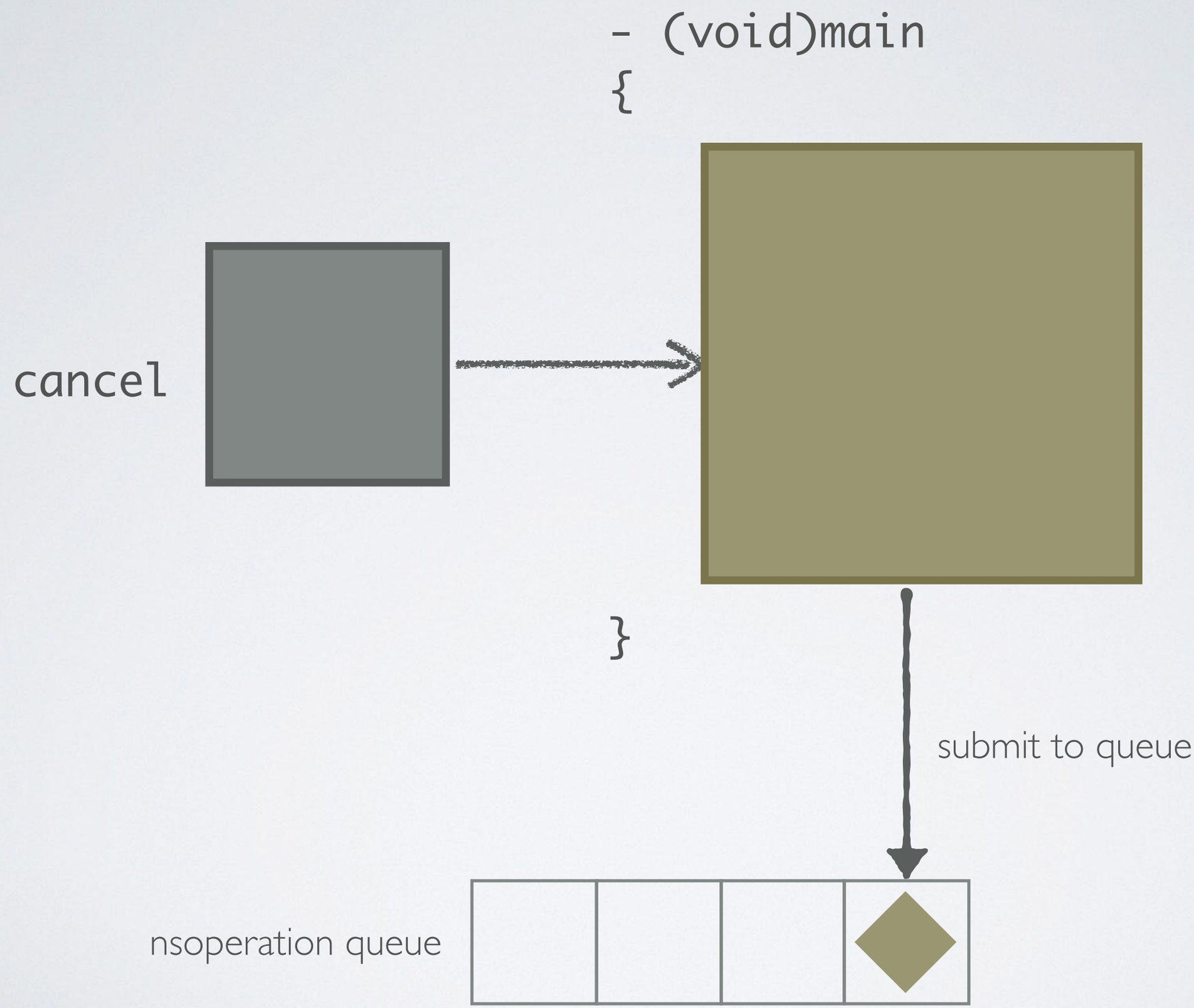
```
- (void)main
{
```

cancel

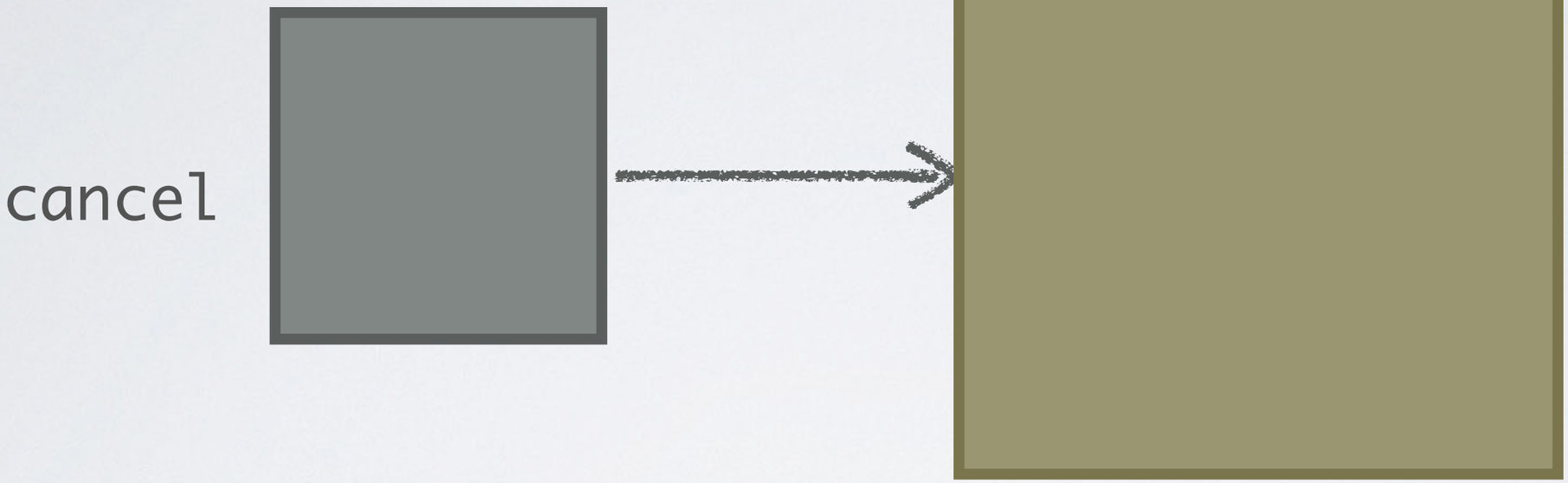


```
}
```

```
@interface BTSMyOperation : NSOperation
```

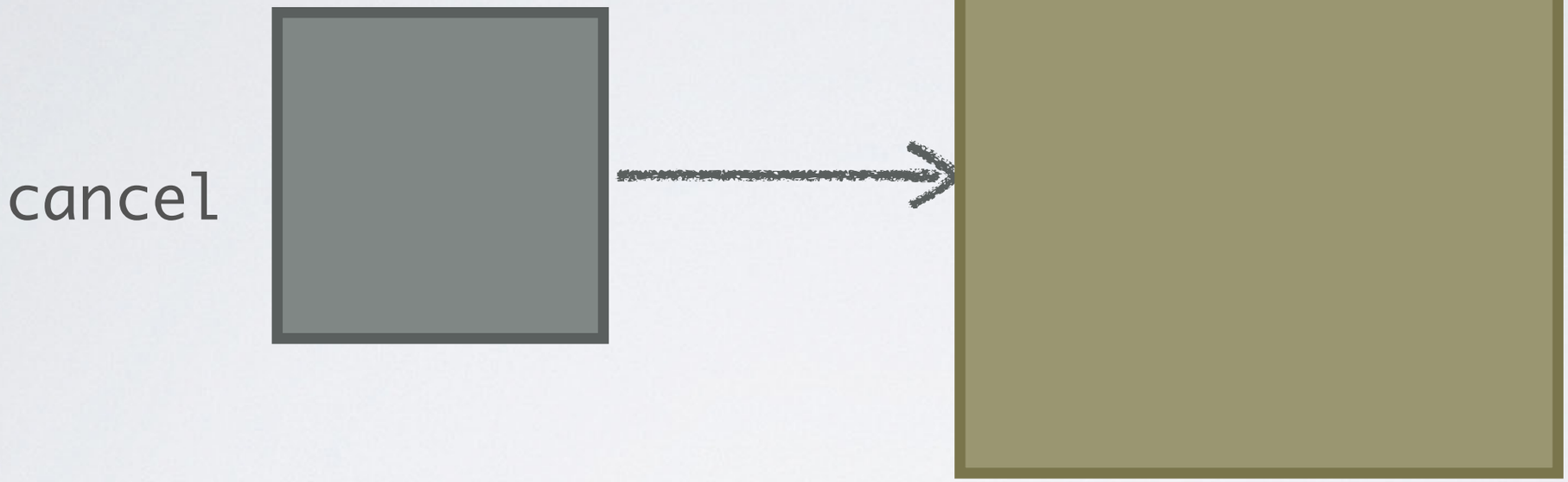

- (void)main
{
cancel
}
submit to queue
nsoperation queue

```
- (void)main
{
```




```
}
```

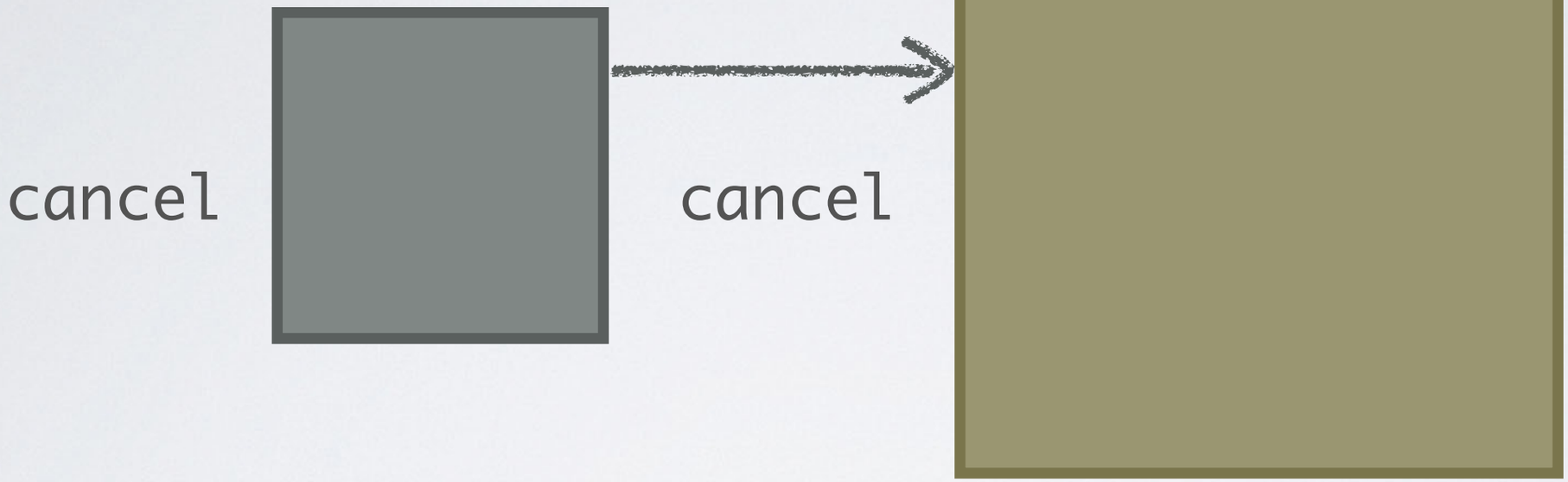
```
- (void)main
{
```




```
}
```

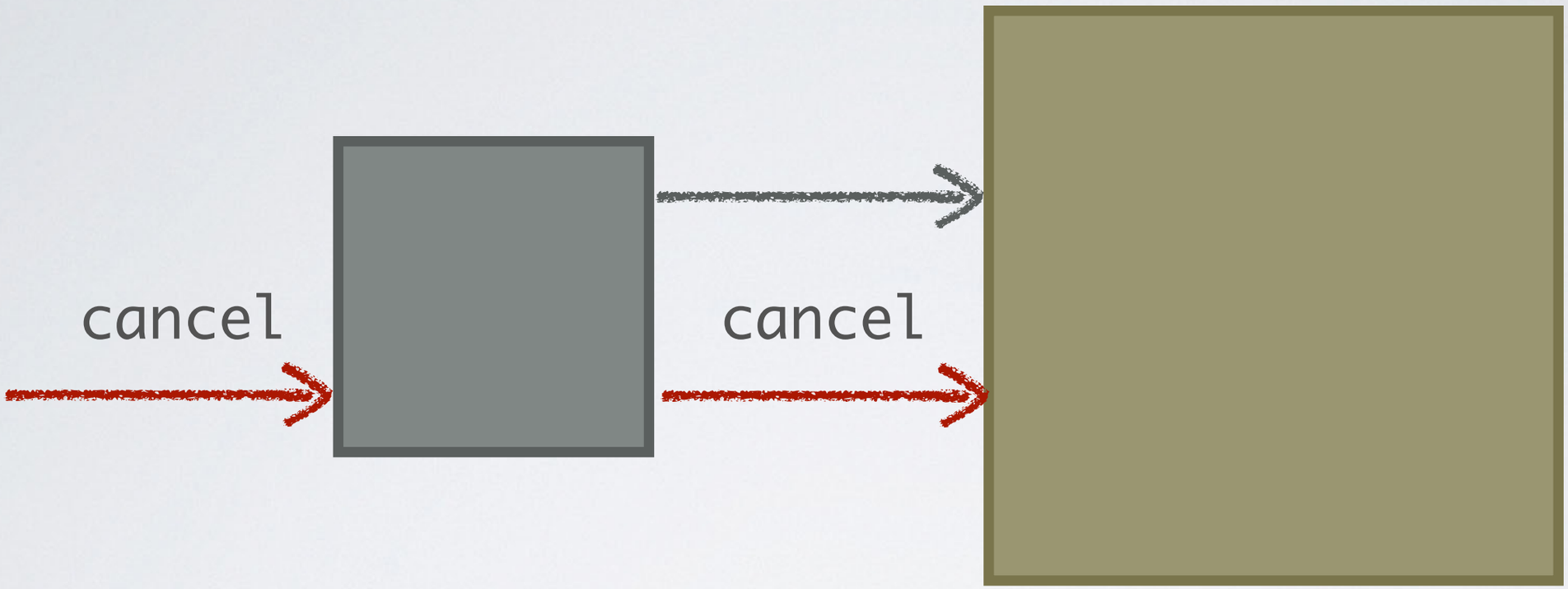
you say, “let’s cancel”

```
- (void)main
{
```




```
}
```

```
- (void)main
{
```



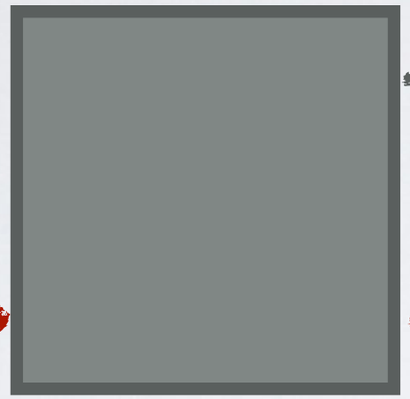

```
}
```

```
- (void)main
{
```

```
while (!done && ![self isCancelled]) {

}
```

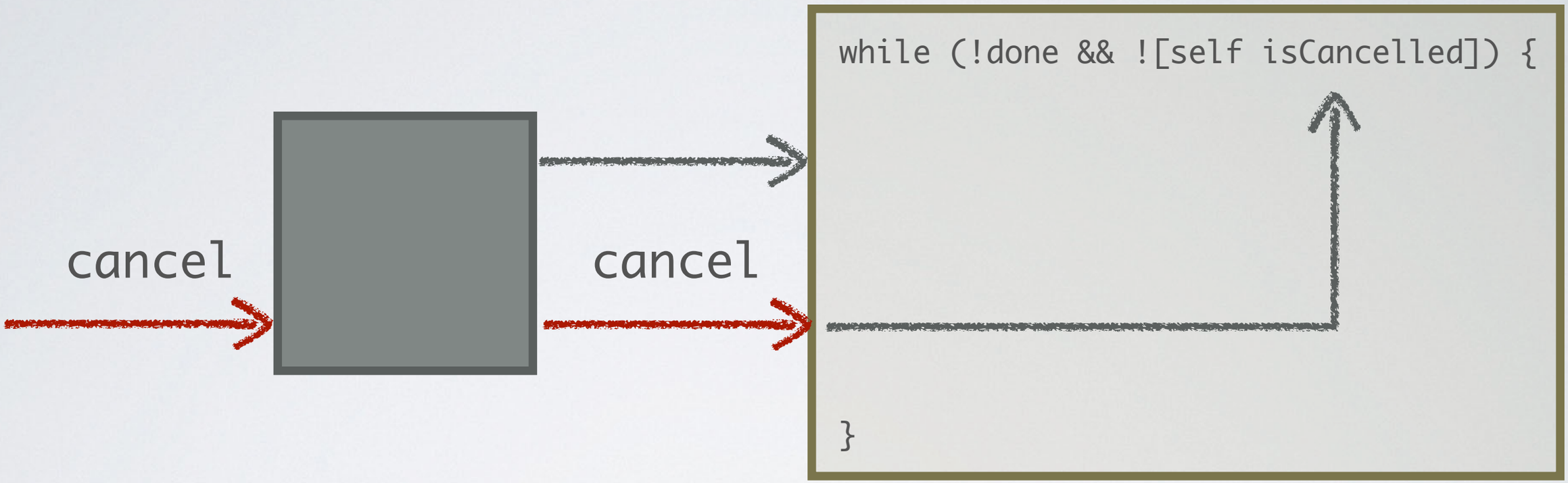
cancel

cancel



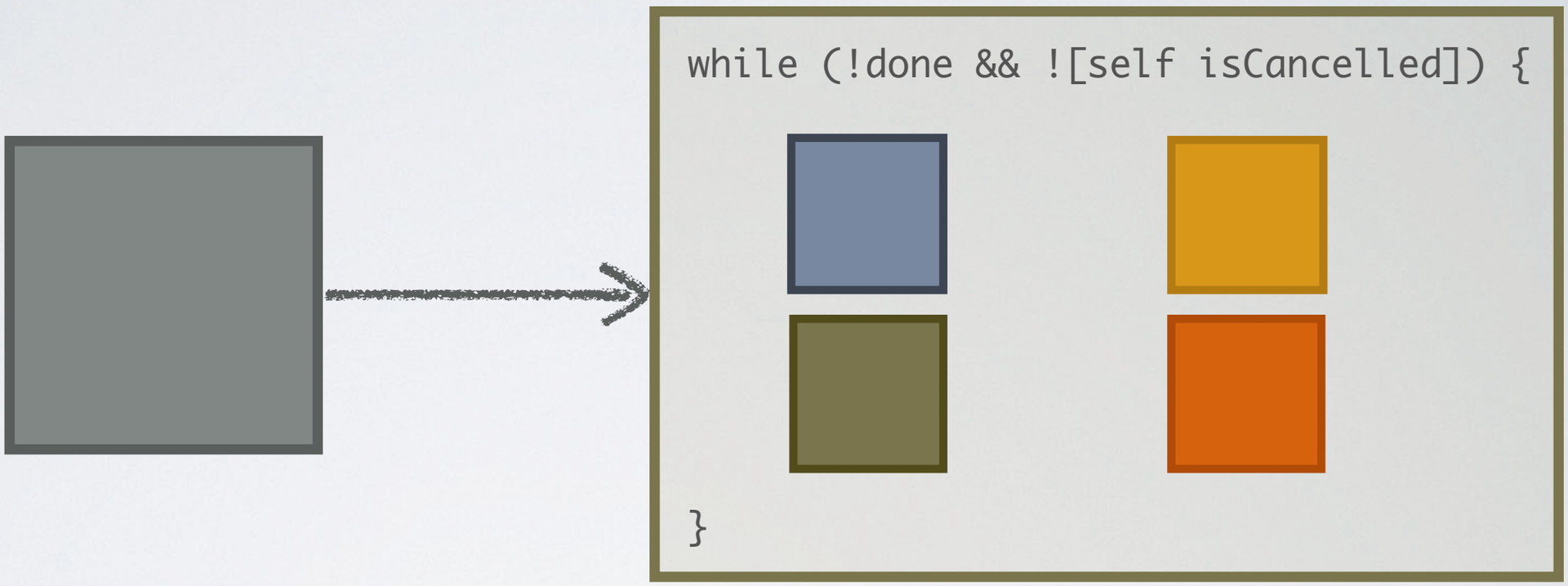
```
}
```

```
- (void)main
{
```




```
}
```

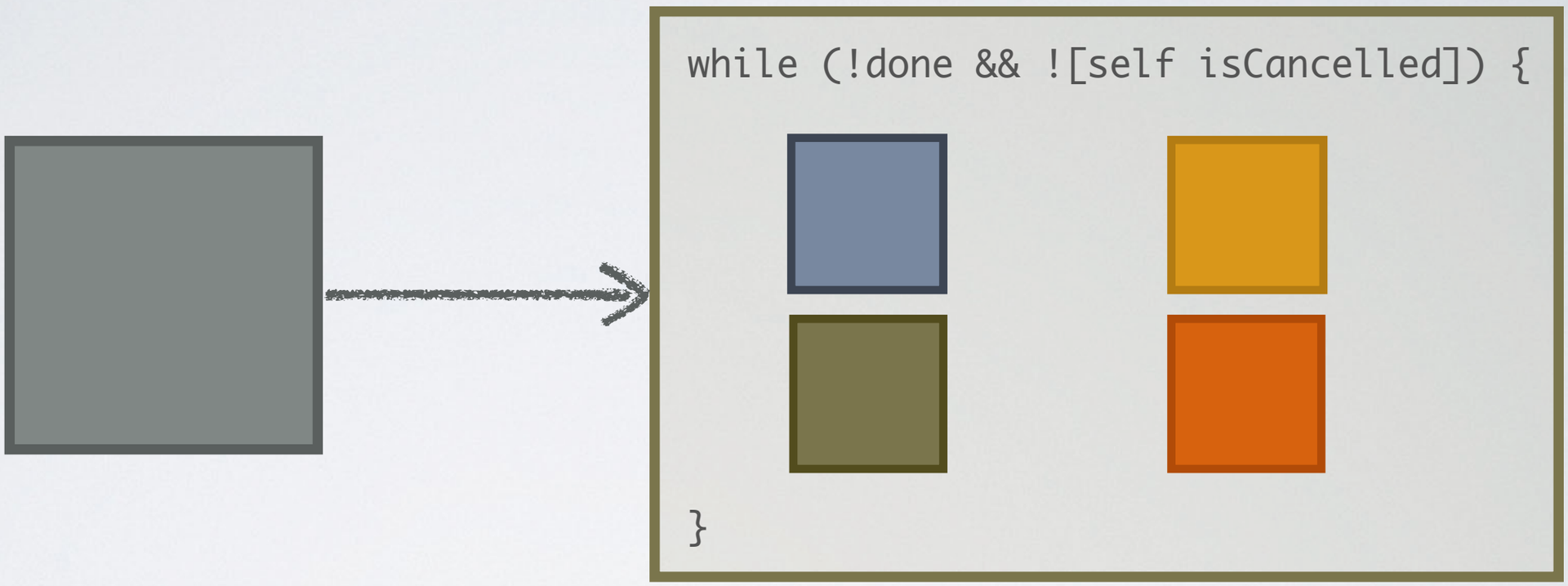
```
- (void)main
{
```




```
}
```

there are lots of small “units of work”

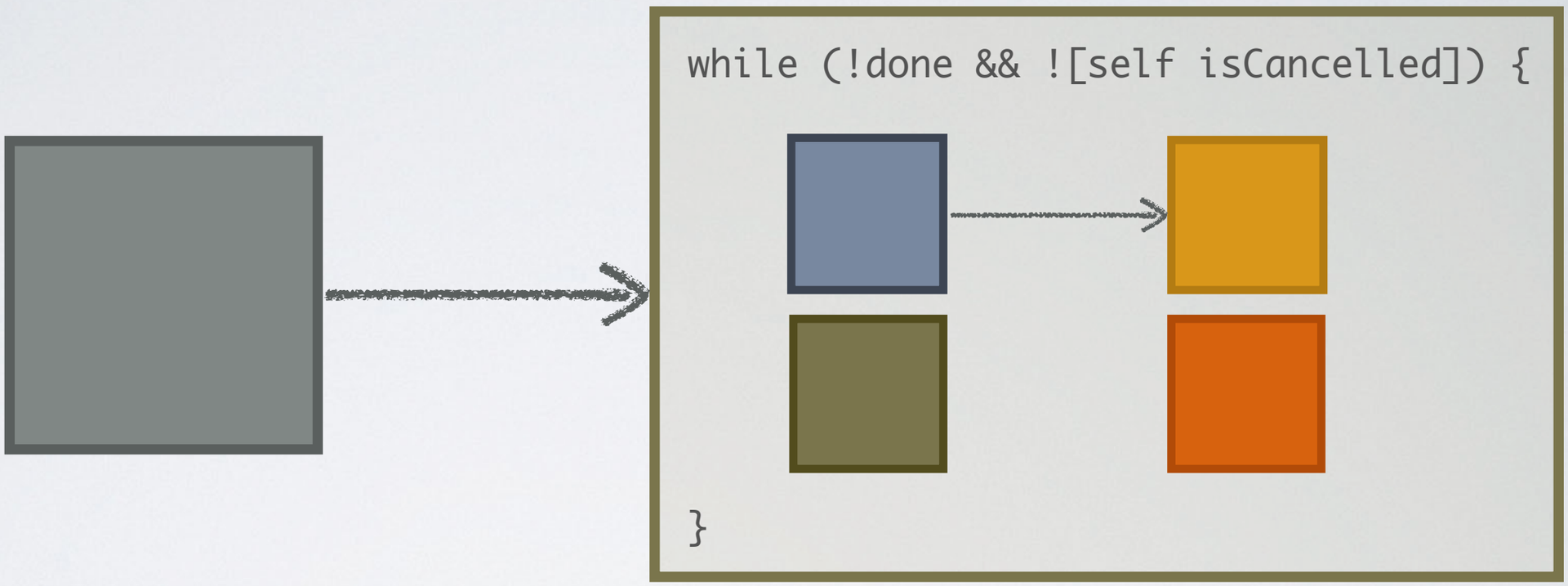
```
- (void)main
{
```




```
}
```

a “unit of work” consumes and/ or produces data

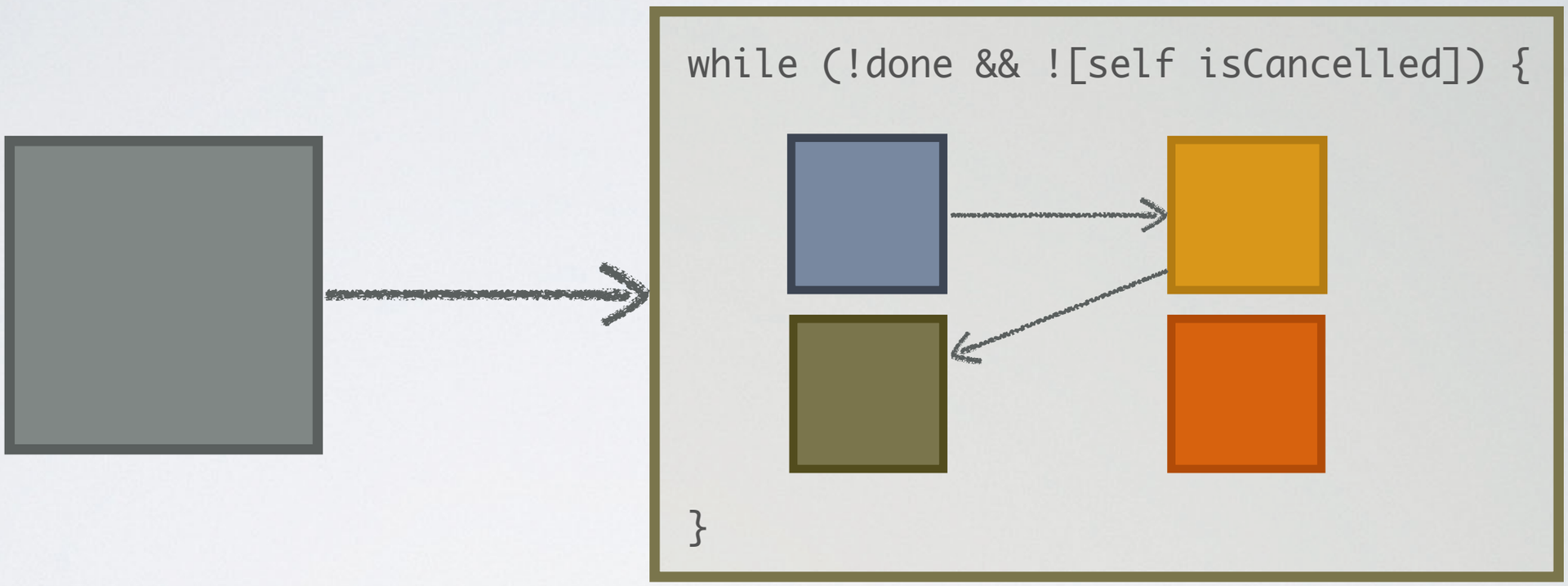
```
- (void)main
{
```




```
}
```

a “unit of work” consumes and/ or produces data

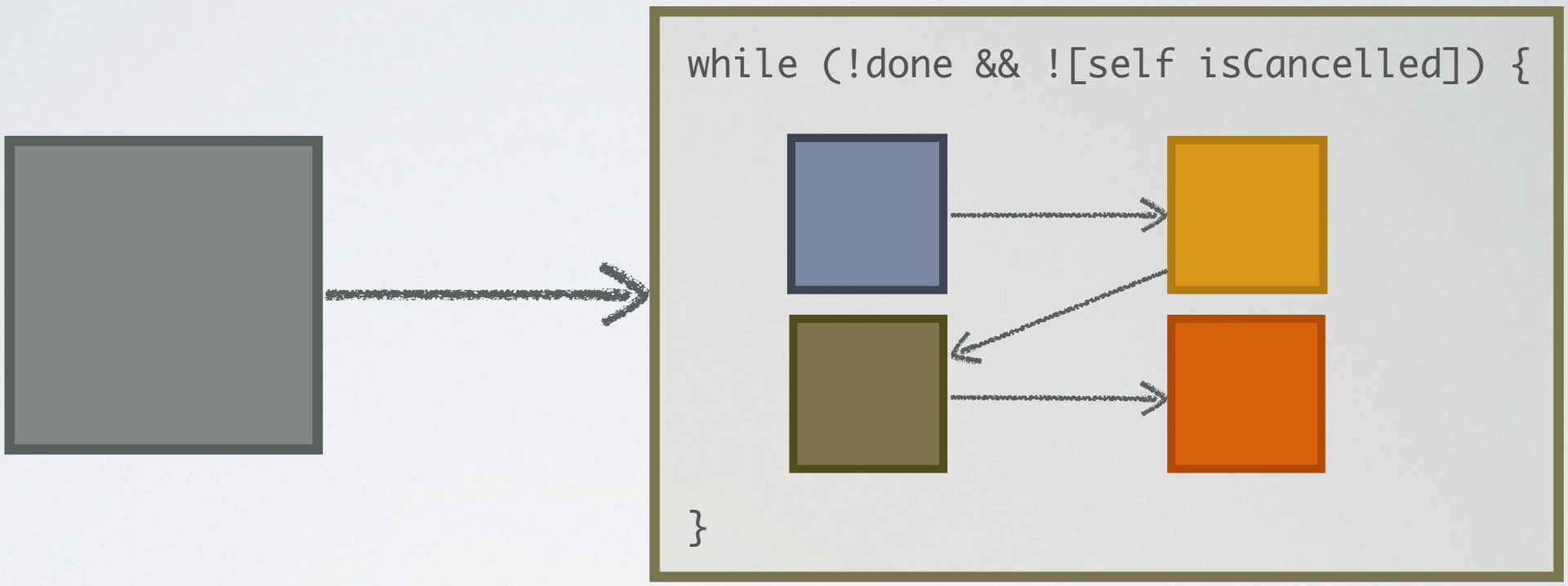
```
- (void)main
{
```




```
}
```

a “unit of work” consumes and/ or produces data

```
- (void)main
{
```

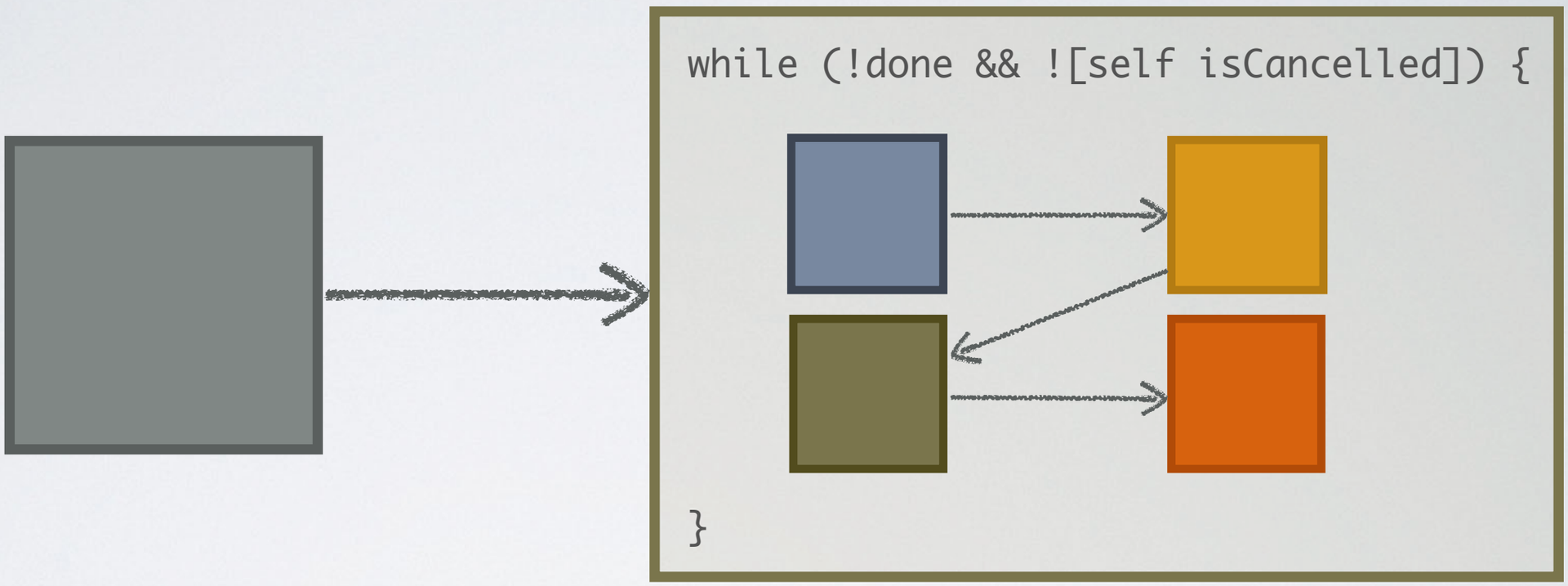



```
}
```

a “unit of work” consumes and/ or produces data

# Operation Composition

```
- (void)main
{
```

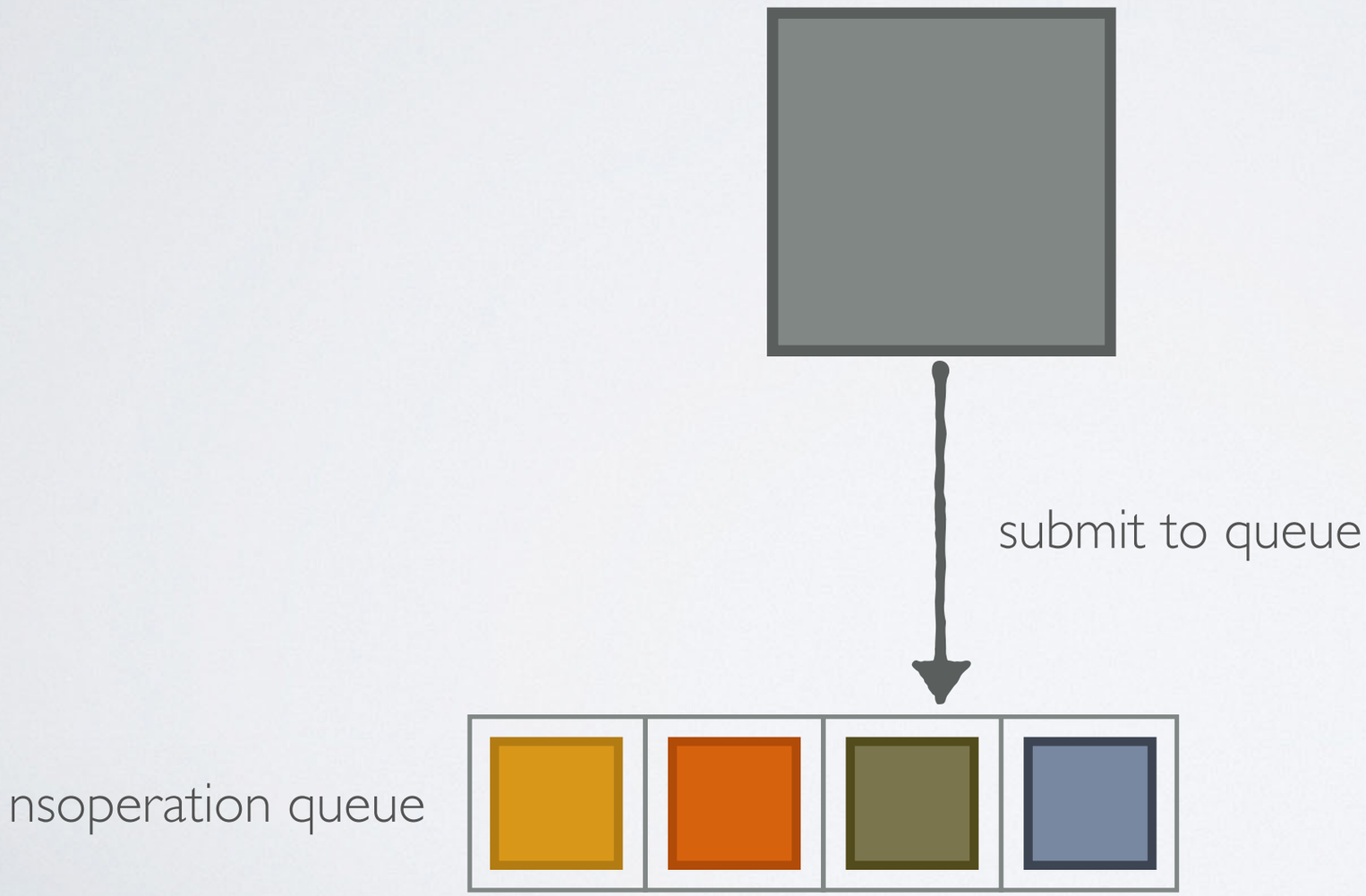



```
}
```

let's break each "unit of work"
into separate nsoperations

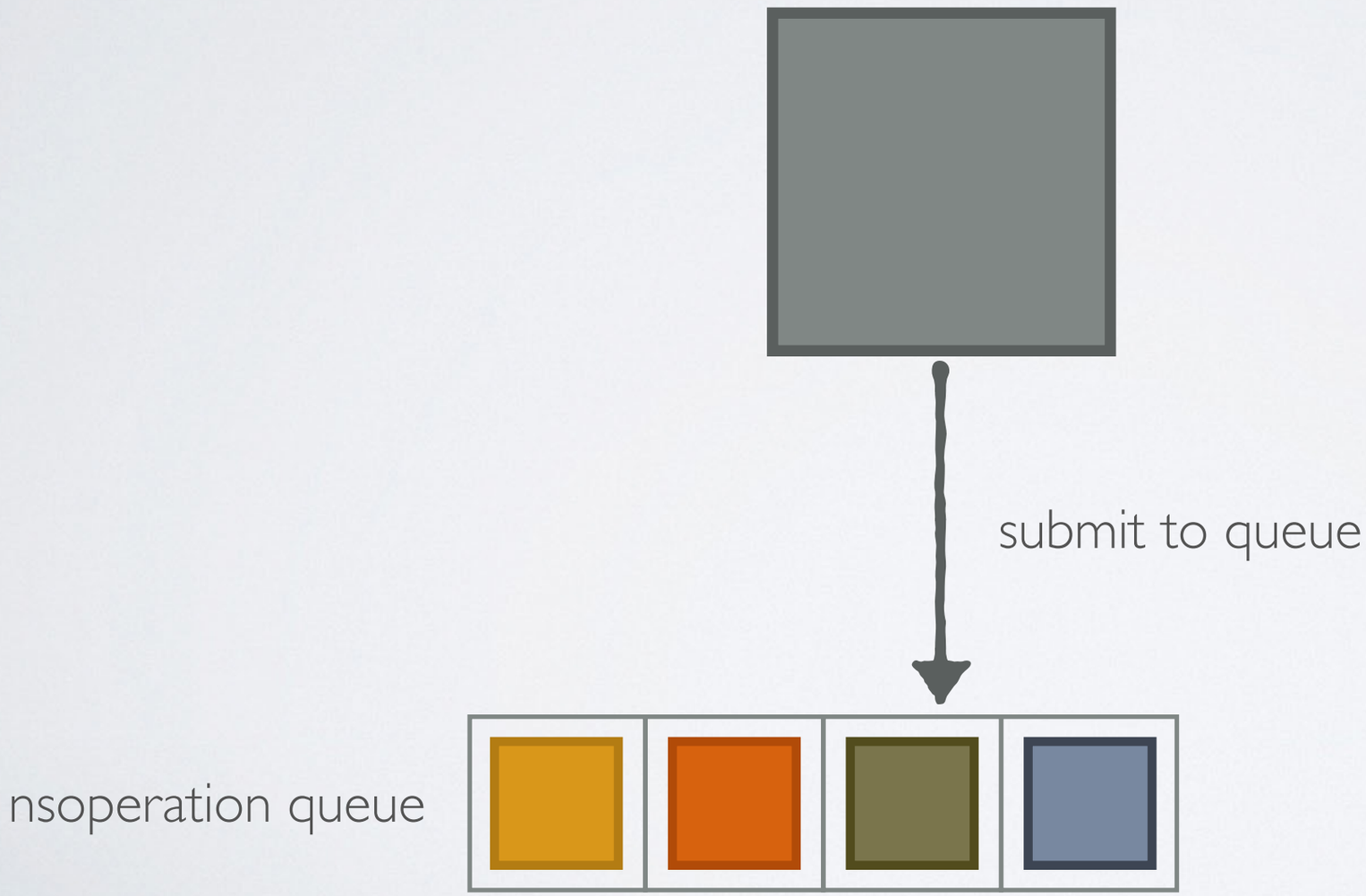
nsoperation queue

the type of operation queue
depends on the “workflow”



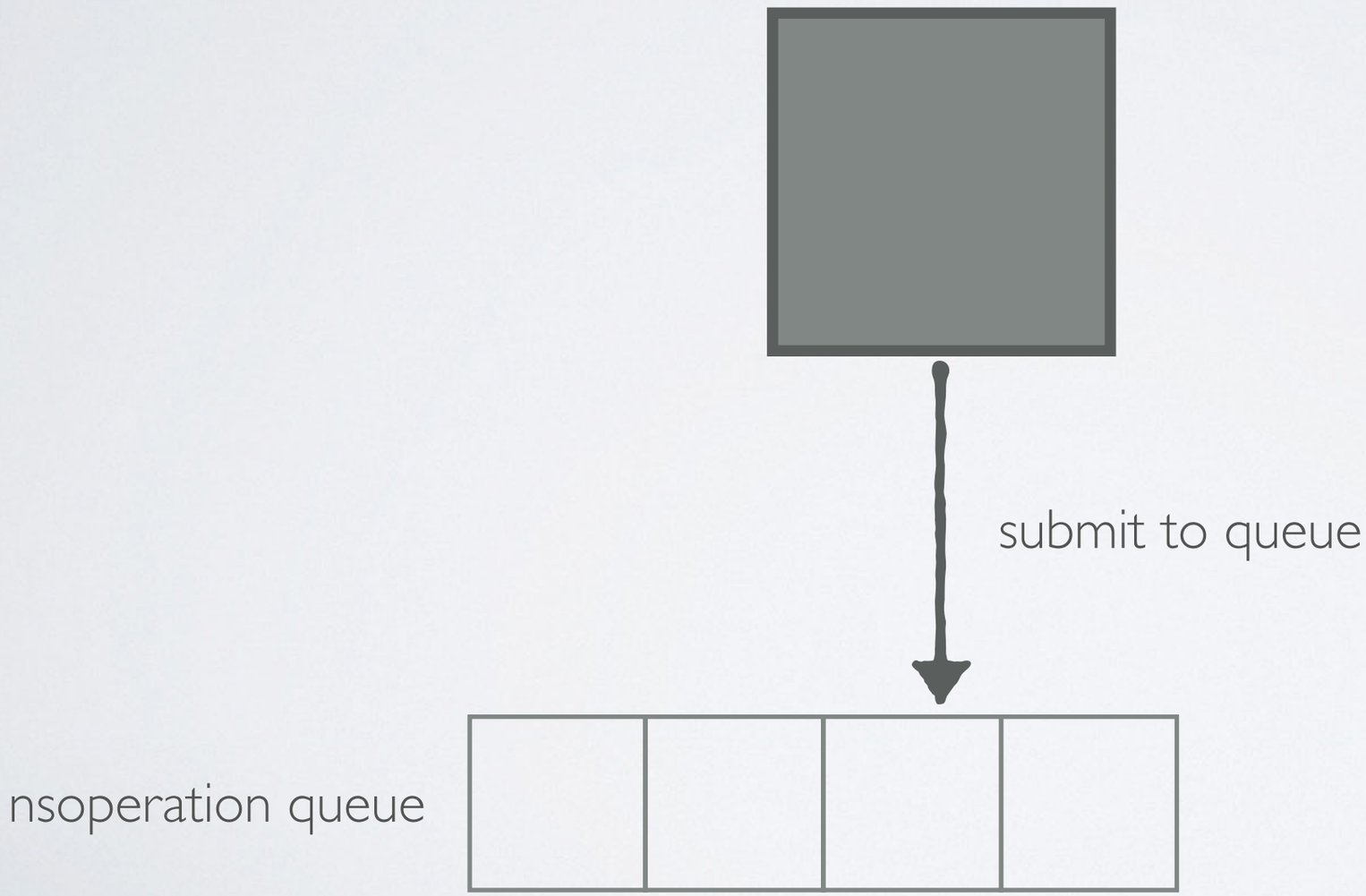
# use a serial queue for a serial workflow

```
[queue setMaxConcurrentOperationCount:1];
```

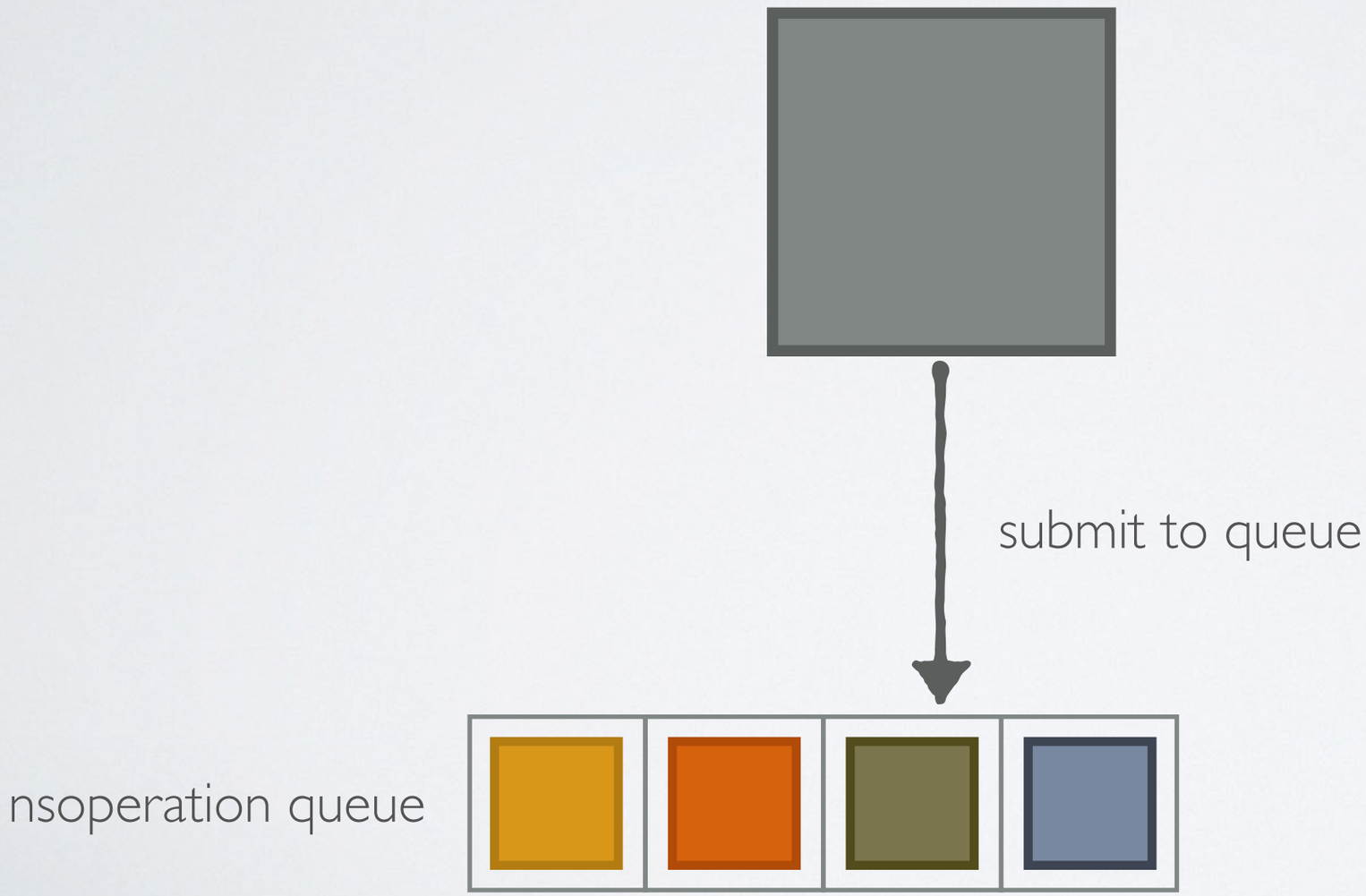


# use a serial queue for a serial workflow

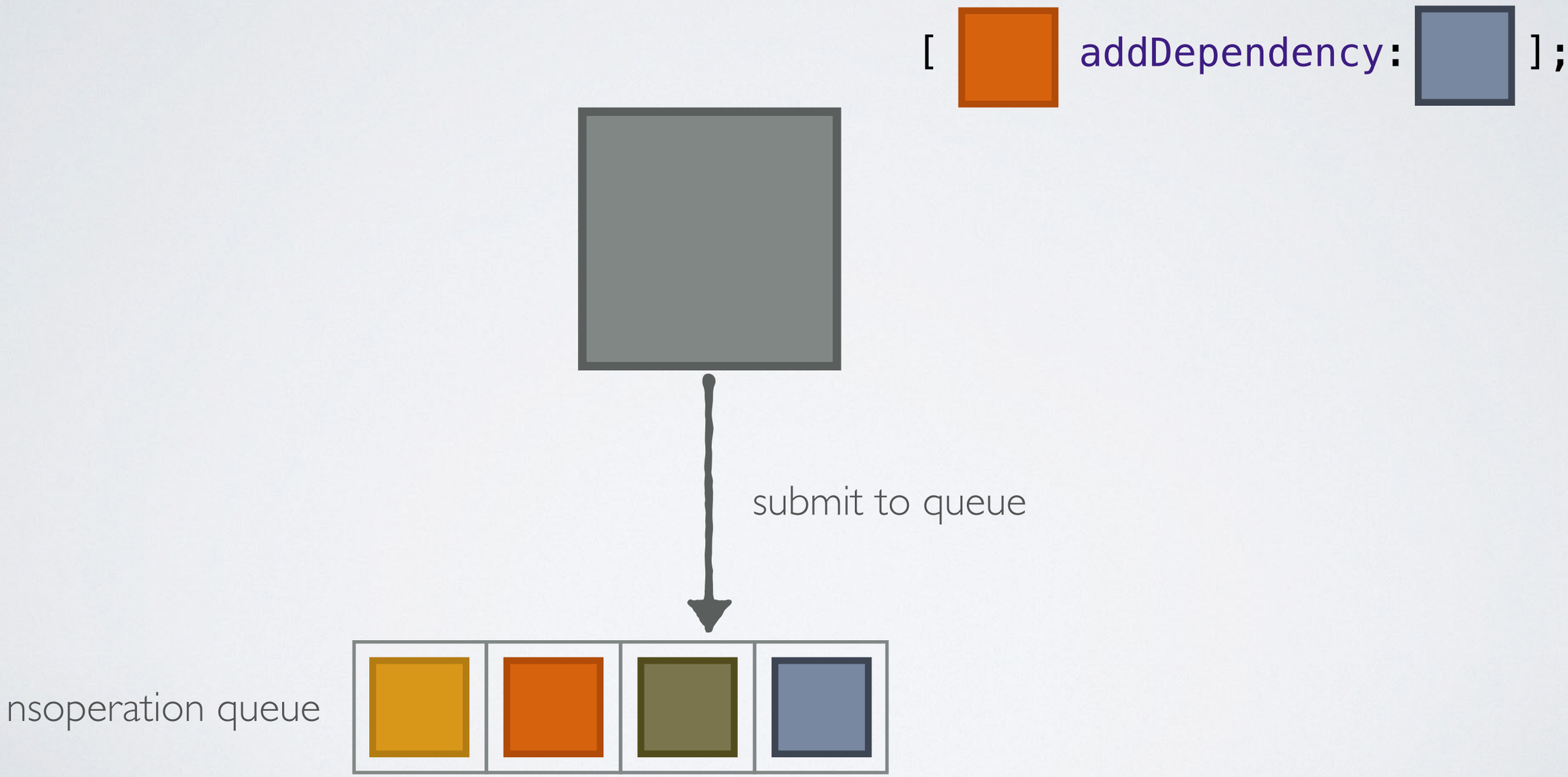
```
[queue setMaxConcurrentOperationCount:1];
```



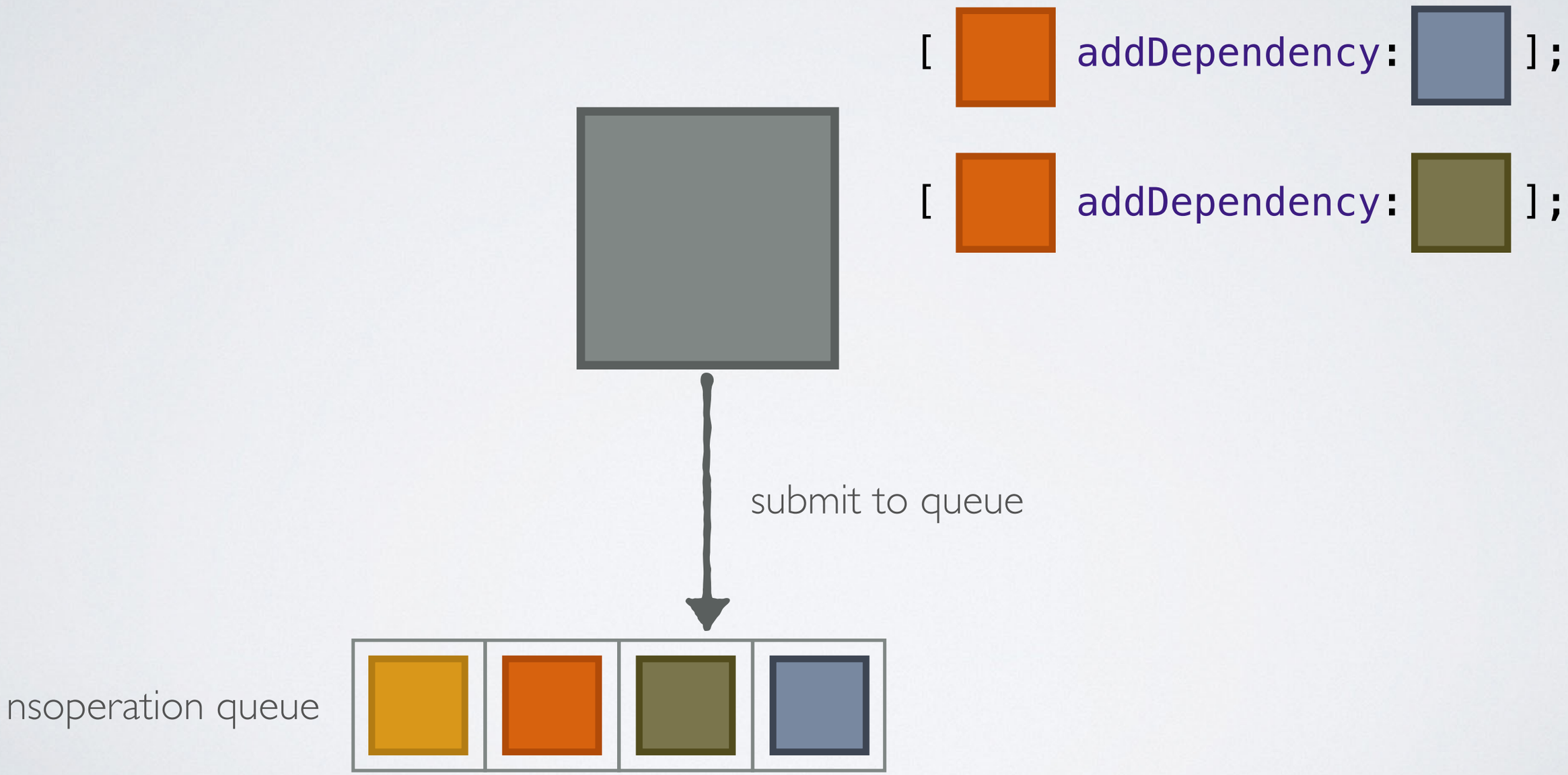
use a concurrent queue with dependent operations
for a semi-concurrent "workflow"



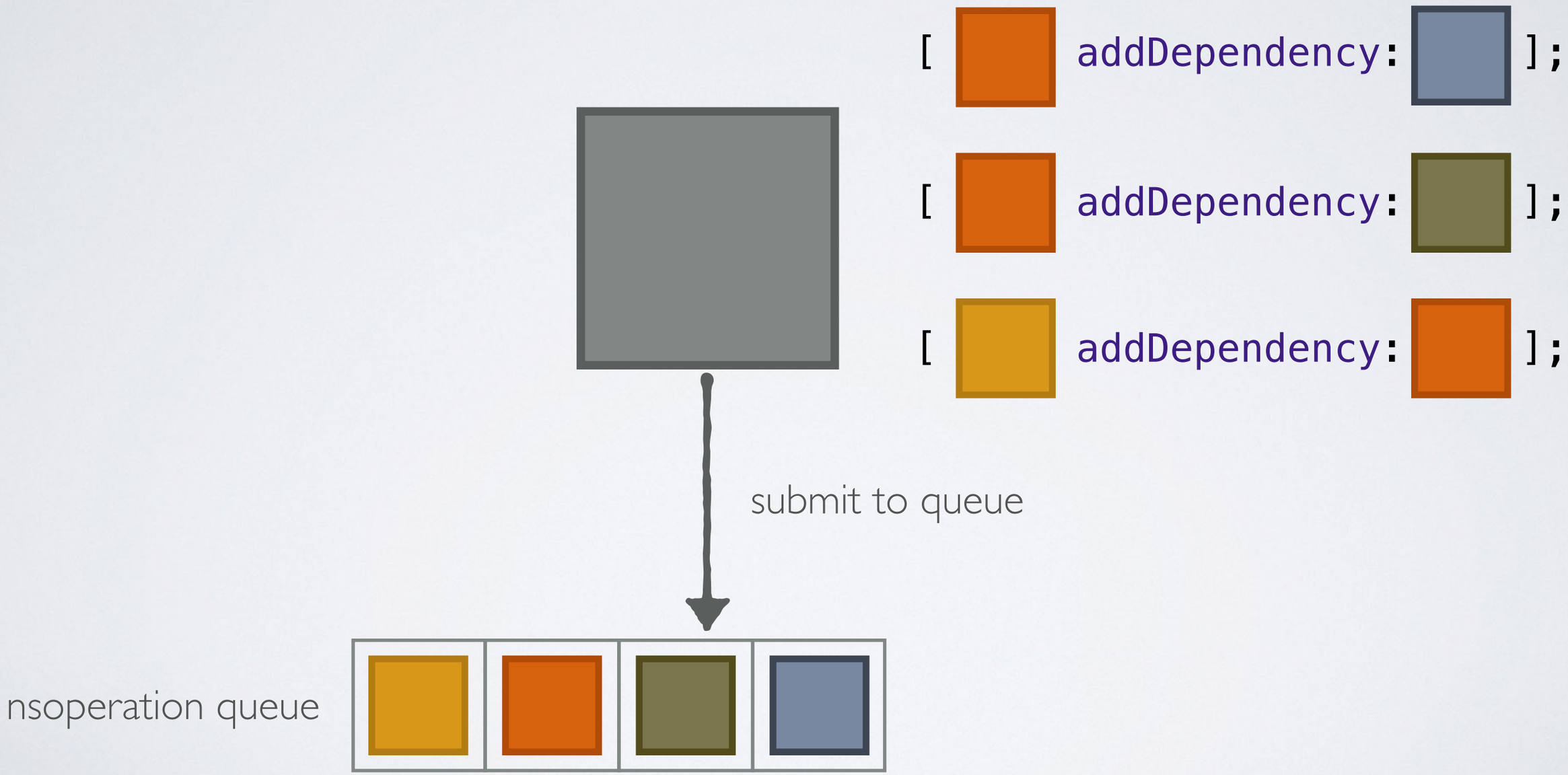
use a concurrent queue with dependent operations
for a semi-concurrent "workflow"



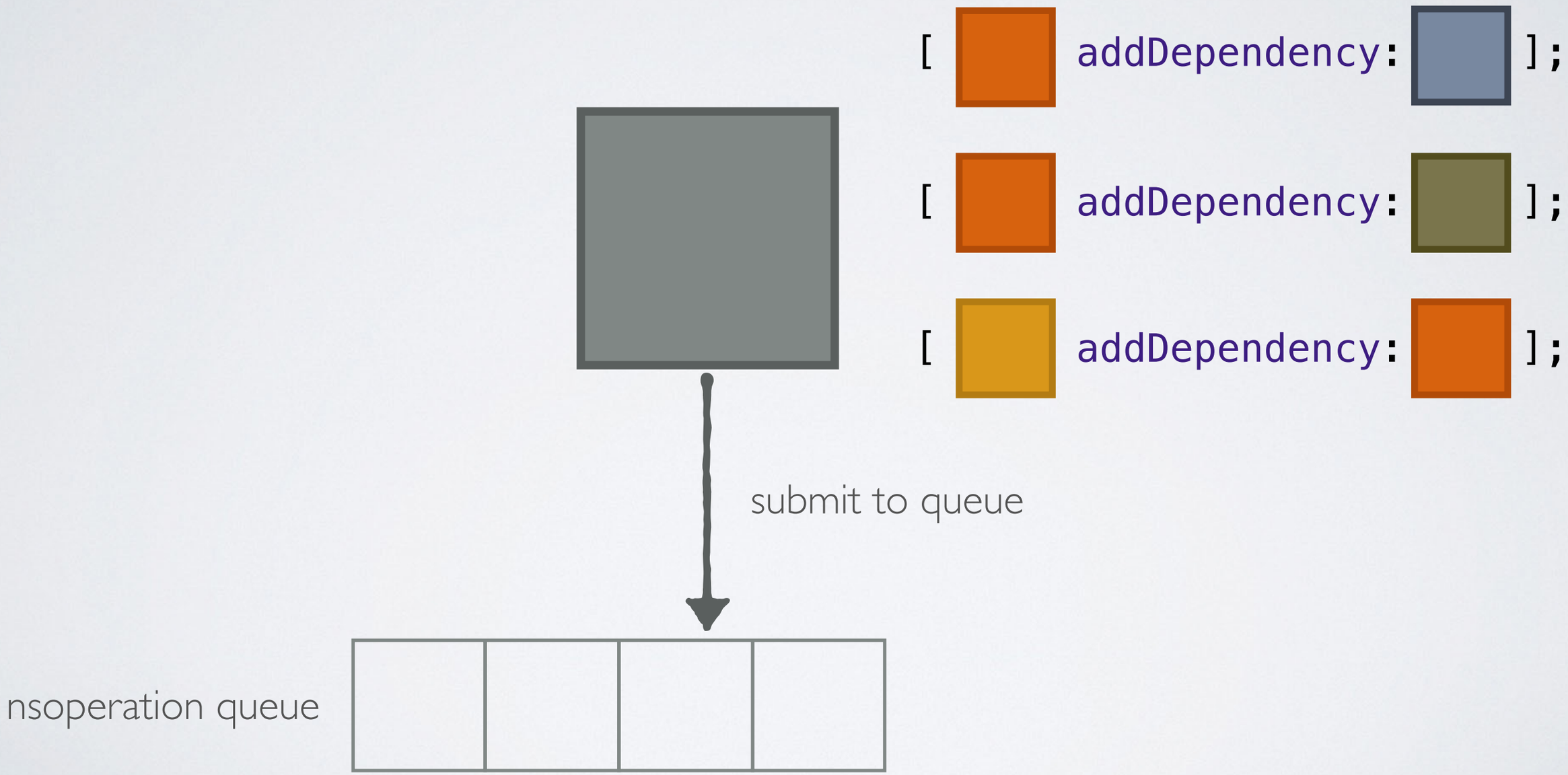
use a concurrent queue with dependent operations
for a semi-concurrent "workflow"



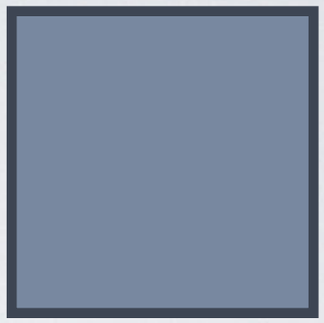
use a concurrent queue with dependent operations
for a semi-concurrent "workflow"

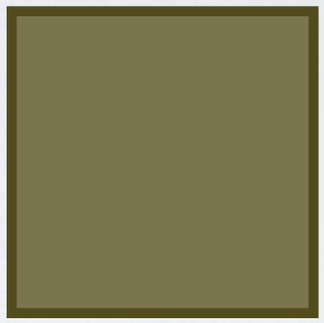


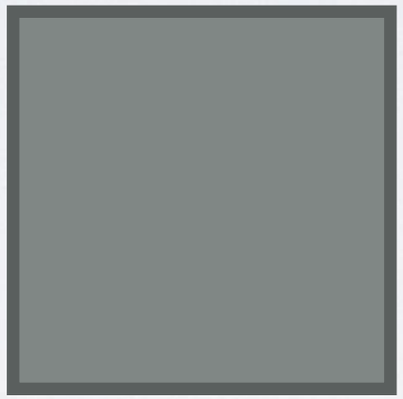
use a concurrent queue with dependent operations
for a semi-concurrent "workflow"

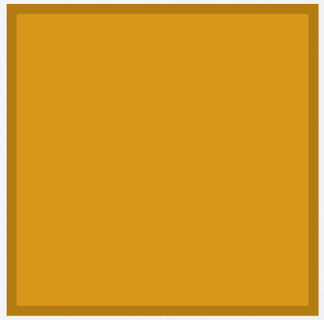


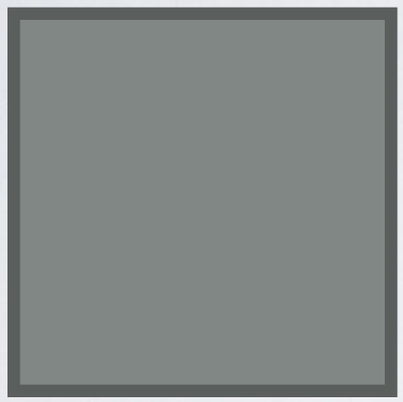
need a way to pass data
between operations



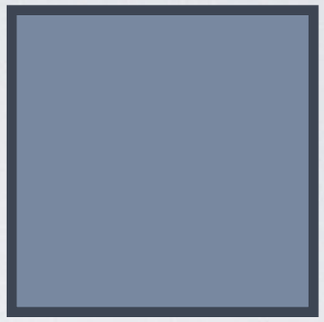


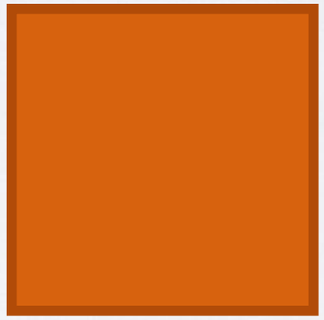


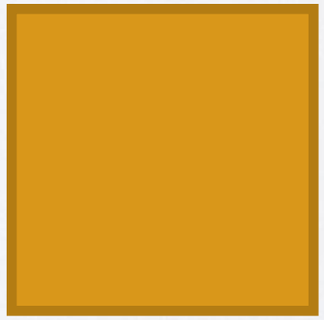





“session”







“session”

“session”

thread-safe map


thread-safe map

- locks

"session"

thread-safe map

- locks
- isolation queues

"session"

thread-safe map

- locks
- isolation queues


thread-safe map

- locks
- isolation queues

used to communicate between operations


thread-safe map

- locks
- isolation queues

used to communicate between operations

"session"

thread-safe map

- locks
- isolation queues

used to communicate between operations

use kvo to publish nserror


thread-safe map

- locks
- isolation queues

used to communicate between operations

use kvo to publish nserror

- used to cancel the workflow


thread-safe map

- locks
- isolation queues

used to communicate between operations

use kvo to publish nserror

- used to cancel the workflow


thread-safe map

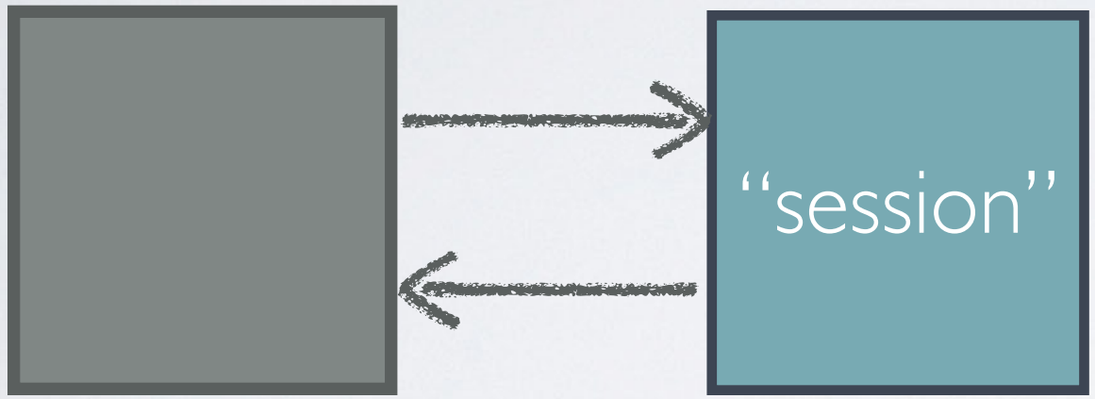
- locks
- isolation queues

used to communicate between operations

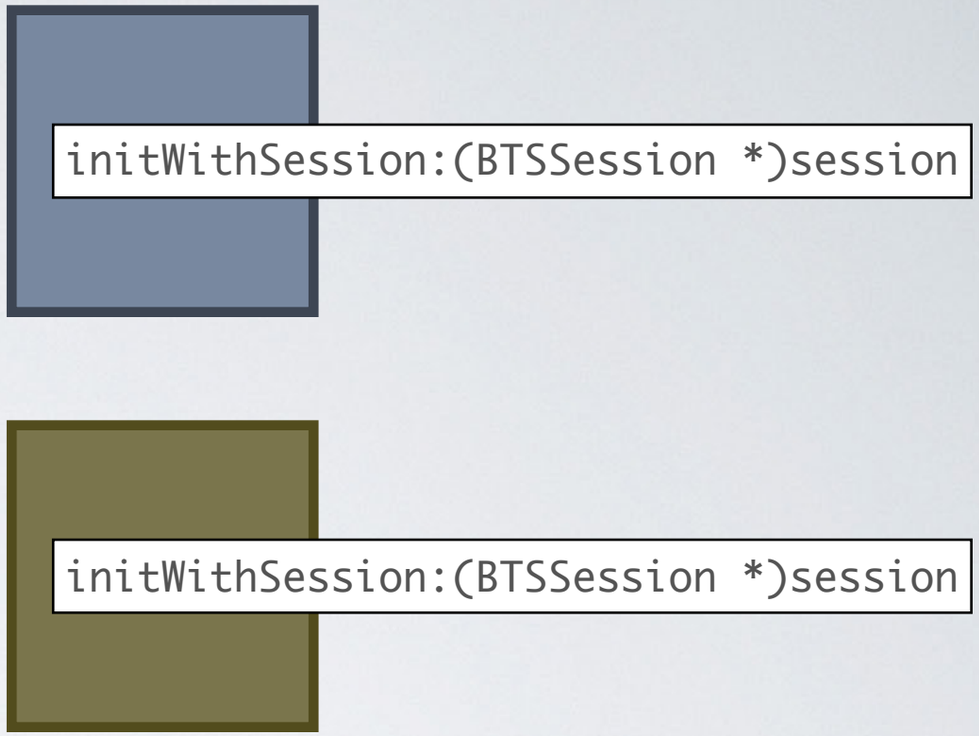
use kvo to publish nserror

- used to cancel the workflow

persist for “resumability”



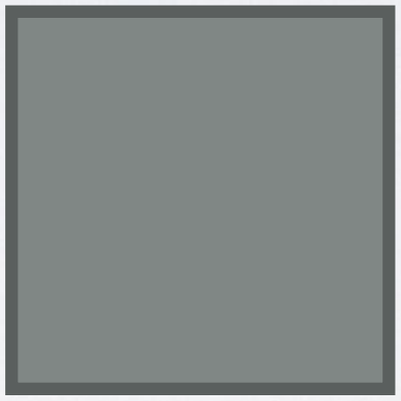

create operations
with “session” object



# let's build a workflow

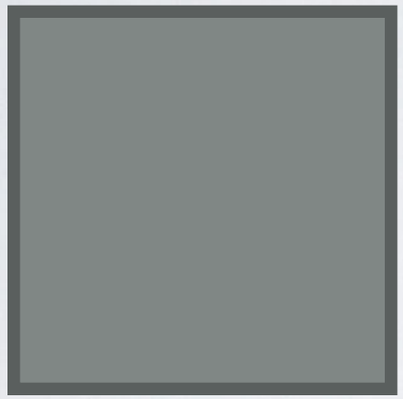
# download a large sqlite file

# download a large sqlite file

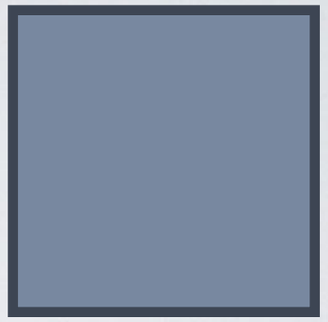


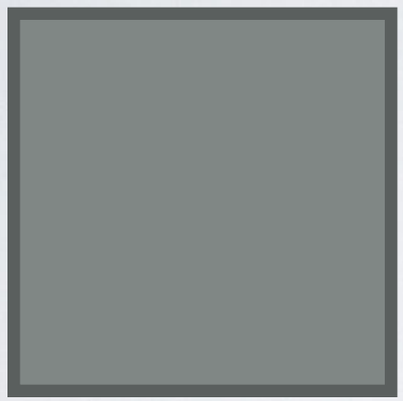
“download controller”

nsoperation serial queue

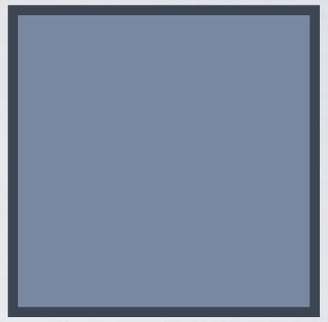


BTSFetchLatestMetaDataJSONOperation

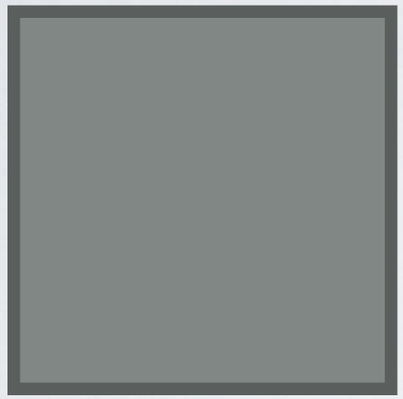




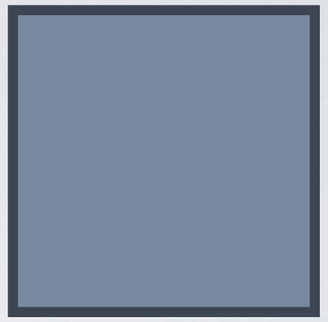
BTSFetchLatestMetaDataJSONOperation



BTSDownloadCompressedSQLiteFileOperation

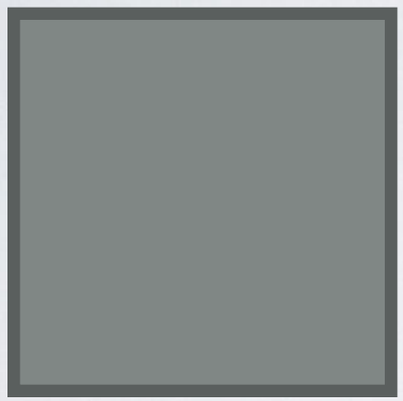


BTSFetchLatestMetaDataJSONOperation

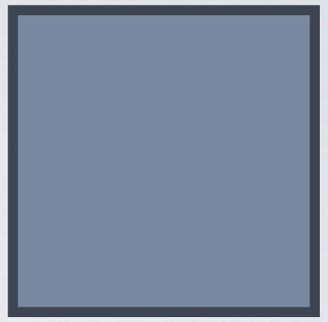


BTSDownloadCompressedSQLiteFileOperation

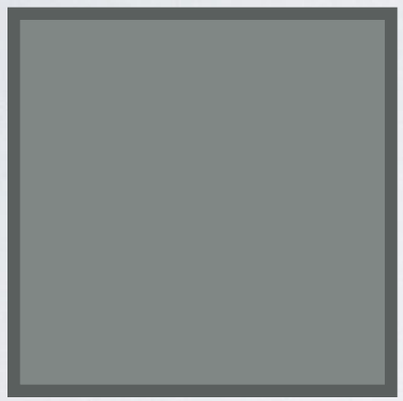
BTSDecompressSQLiteFileOperation



BTSFetchLatestMetaDataJSONOperation

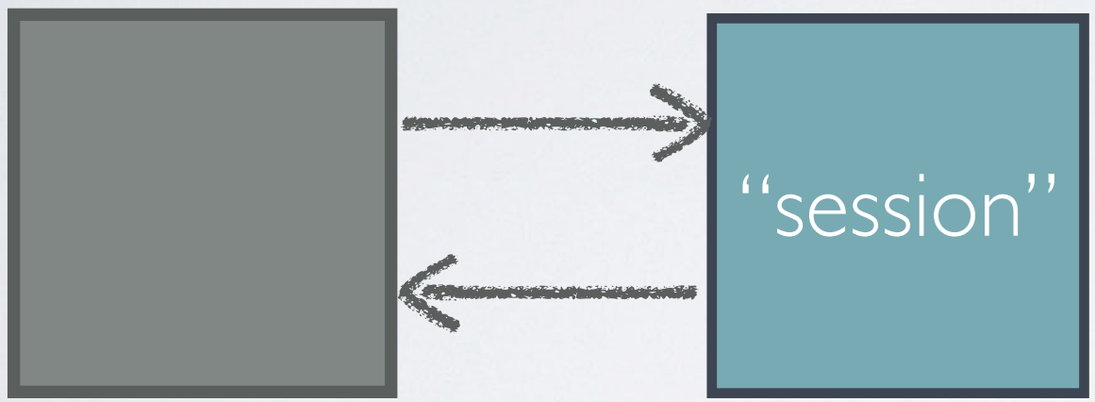


BTSDownloadCompressedSQLiteFileOperation

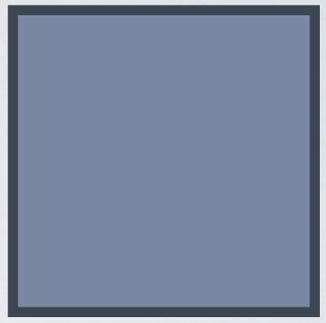


BTSDecompressSQLiteFileOperation

BTSMoveSQLiteFileOperation




BTSFetchLatestMetaDataJSONOperation

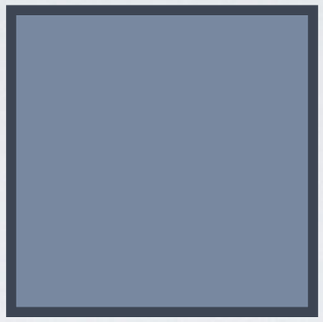


BTSDownloadCompressedSQLiteFileOperation

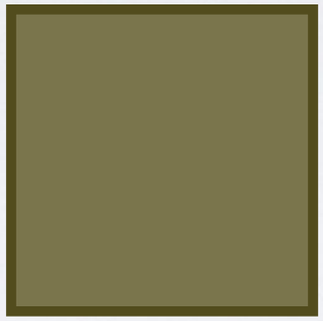
BTSDecompressSQLiteFileOperation

BTSMoveSQLiteFileOperation

BTSFetchLatestMetaDataJSONOperation



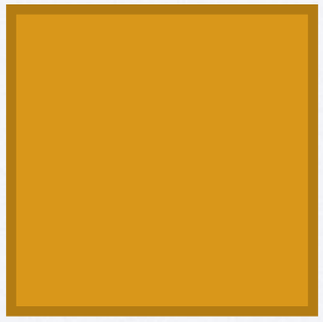
BTSDownloadCompressedSQLiteFileOperation



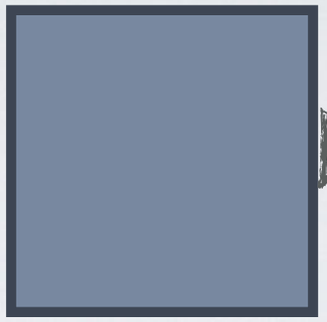
“session”

BTSDecompressSQLiteFileOperation

BTSMoveSQLiteFileOperation

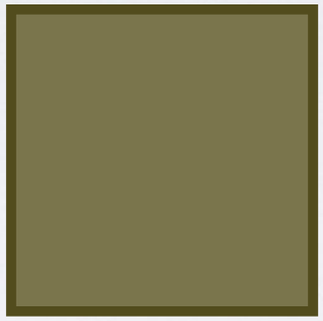


BTSFetchLatestMetaDataJSONOperation



publishes downloaded sqlite meta-data json

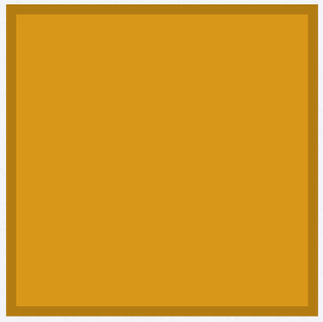
BTSDownloadCompressedSQLiteFileOperation



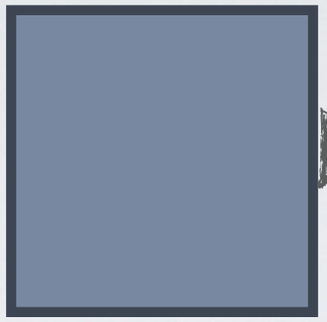
“session”

BTSDecompressSQLiteFileOperation

BTSMoveSQLiteFileOperation



BTSFetchLatestMetaDataJSONOperation



publishes downloaded sqlite meta-data json

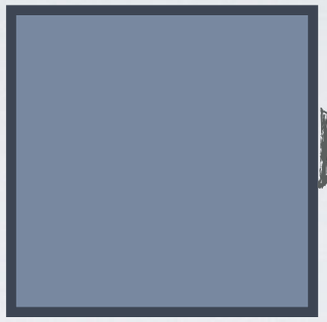
BTSDownloadCompressedSQLiteFileOperation

consumes json


BTSDecompressSQLiteFileOperation

BTSMoveSQLiteFileOperation

BTSFetchLatestMetaDataJSONOperation



publishes downloaded sqlite meta-data json

BTSDownloadCompressedSQLiteFileOperation

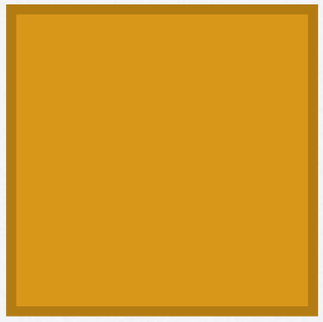
consumes json

publishes sqlite gzip cache file url

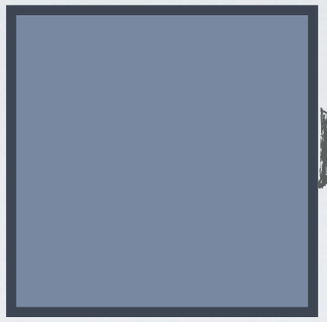
“session”

BTSDecompressSQLiteFileOperation

BTSMoveSQLiteFileOperation



BTSFetchLatestMetaDataJSONOperation



publishes downloaded sqlite meta-data json

BTSDownloadCompressedSQLiteFileOperation

consumes json

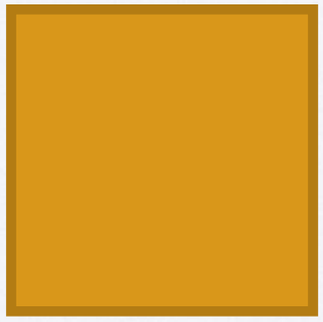
publishes sqlite gzip cache file url

"session"

BTSDecompressSQLiteFileOperation

consumes gzip cache file url

BTSMoveSQLiteFileOperation



BTSFetchLatestMetaDataJSONOperation

publishes downloaded sqlite meta-data json

BTSDownloadCompressedSQLiteFileOperation

consumes json

publishes sqlite gzip cache file url

“session”

BTSDecompressSQLiteFileOperation

consumes gzip cache file url

publishes sqlite cache file url

BTSMoveSQLiteFileOperation

BTSFetchLatestMetaDataJSONOperation

publishes downloaded sqlite meta-data json

BTSDownloadCompressedSQLiteFileOperation

consumes json

publishes sqlite gzip cache file url

“session”

BTSDecompressSQLiteFileOperation

consumes gzip cache file url

publishes sqlite cache file url

BTSMoveSQLiteFileOperation

consumes sqlite cache file url

BTSFetchLatestMetaDataJSONOperation

publishes downloaded sqlite meta-data json

BTSDownloadCompressedSQLiteFileOperation

consumes json

publishes sqlite gzip cache file url

"session"

BTSDecompressSQLiteFileOperation

consumes gzip cache file url

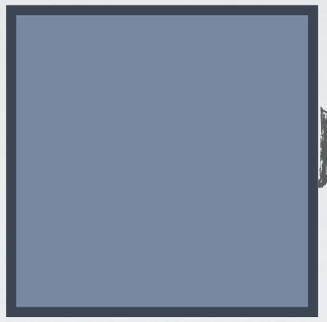
publishes sqlite cache file url

BTSMoveSQLiteFileOperation

consumes sqlite cache file url

publishes sqlite file url in app support directory

`BTSFetchLatestMetaDataJSONOperation`



publishes downloaded sqlite meta-data json

`BTSDownloadCompressedSQLiteFileOperation`

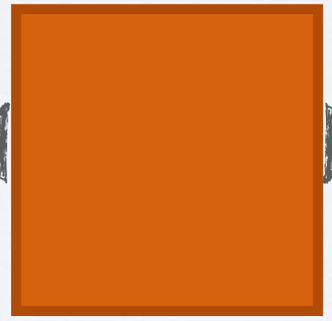
consumes json

publishes sqlite gzip cache file url

“session”

`BTSDecompressSQLiteFileOperation`

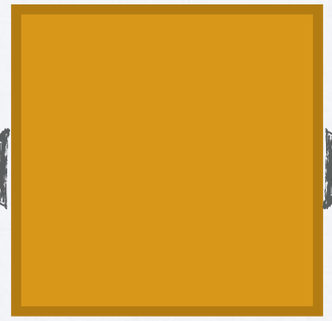
consumes gzip cache file url



publishes sqlite cache file url

`BTSMoveSQLiteFileOperation`

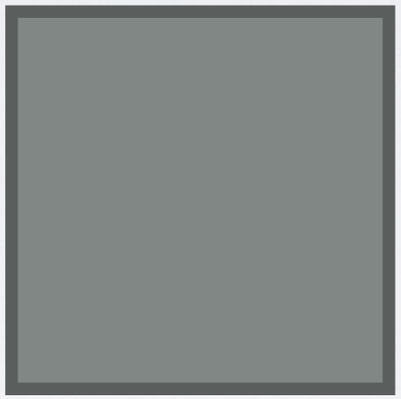
consumes sqlite cache file url



publishes sqlite file url in app support directory


“session”

# download complete



*/Library/Application Support/path/to/database.sqlite*

# adding features

your boss says, “the app is not displaying any data”

your boss says, “the app is not displaying any data”

you say, “looks like the sqlite database has issues”

your boss says, “the app is not displaying any data”

you say, “looks like the sqlite database has issues”

you suggest, “add verification to the download workflow”

your boss says, “the app is not displaying any data”

you say, “looks like the sqlite database has issues”

you suggest, “add verification to the download workflow”

BTSVerifySQLiteFileOperation

your boss says, “the app is not displaying any data”

you say, “looks like the sqlite database has issues”

you suggest, “add verification to the download workflow”

you suggest, “add verification to the download workflow”

`BTSVerifySQLiteFileOperation`

you suggest, “add verification to the download workflow”

BTSVerifySQLiteFileOperation

```
- (void)doMainWithSession:(BTSSession *)session
{
    BTSDatabase *database = [self openDatabaseForSession:session];

    NSError *error;
    BOOL success = [self verifyDatabase:database error:&error];
    if (!success) {
        [session setError:error];
    }

    [self closeDatabase:database];
}
```

you suggest, “add verification to the download workflow”

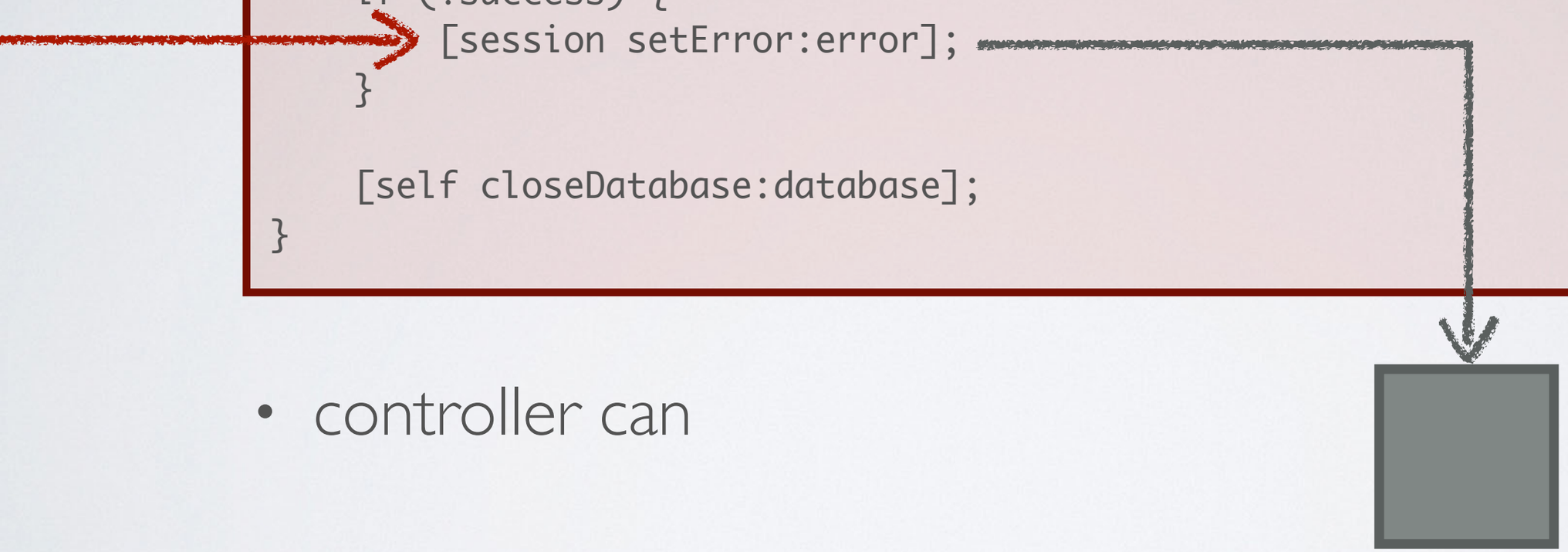
BTSVerifySQLiteFileOperation

```
- (void)doMainWithSession:(BTSSession *)session
{
    BTSDatabase *database = [self openDatabaseForSession:session];

    NSError *error;
    BOOL success = [self verifyDatabase:database error:&error];
    if (!success) {
        [session setError:error];
    }

    [self closeDatabase:database];
}
```

you suggest, "add verification to the download workflow"

BTSVerifySQLiteFileOperation

```
- (void)doMainWithSession:(BTSSession *)session
{
    BTSDatabase *database = [self openDatabaseForSession:session];

    NSError *error;
    BOOL success = [self verifyDatabase:database error:&error];
    if (!success) {
        [session setError:error];
    }

    [self closeDatabase:database];
}
```

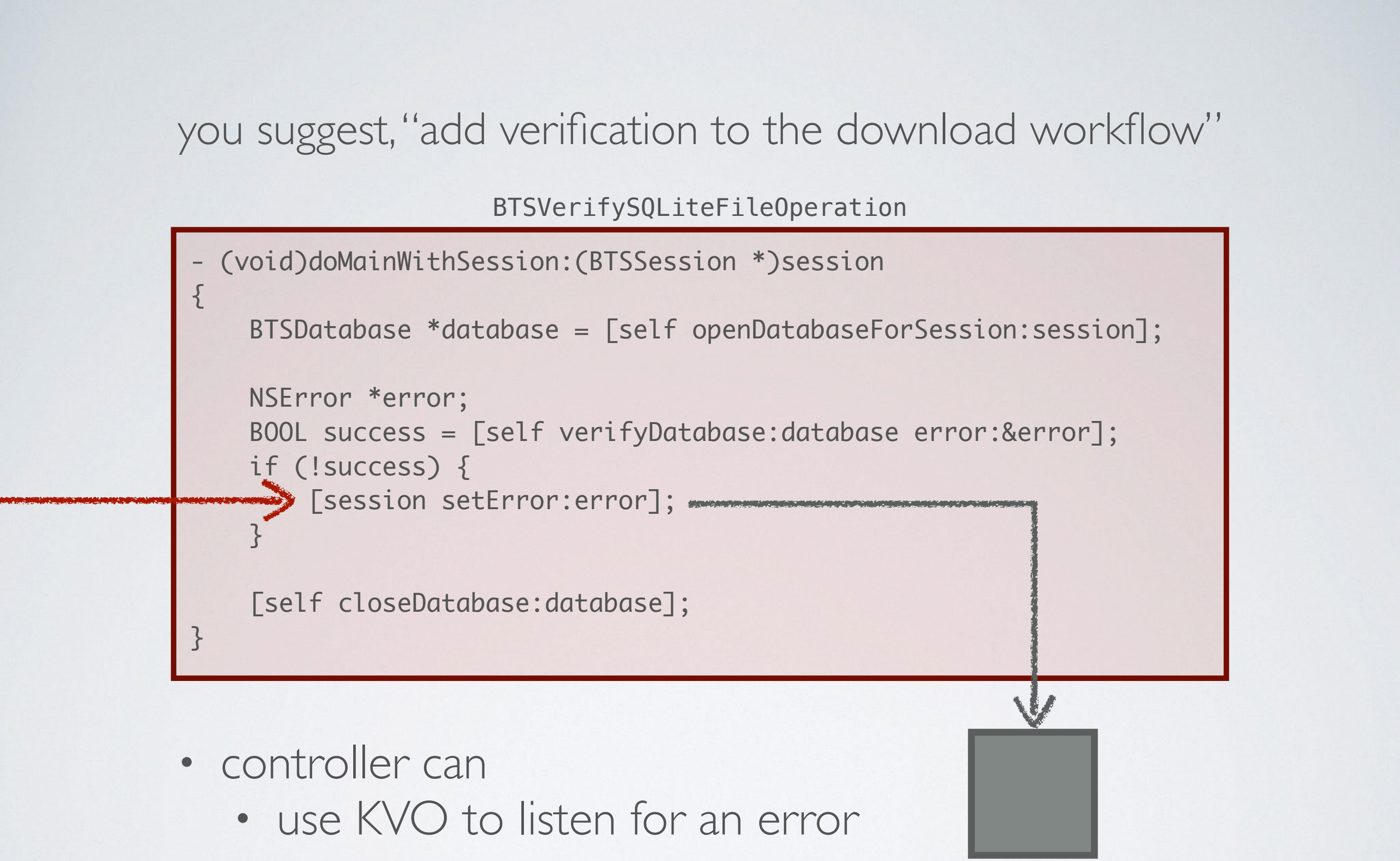


- controller can

you suggest, "add verification to the download workflow"

BTSVerifySQLiteFileOperation

```
- (void)doMainWithSession:(BTSSession *)session
{
    BTSDatabase *database = [self openDatabaseForSession:session];

    NSError *error;
    BOOL success = [self verifyDatabase:database error:&error];
    if (!success) {
        [session setError:error];
    }

    [self closeDatabase:database];
}
```

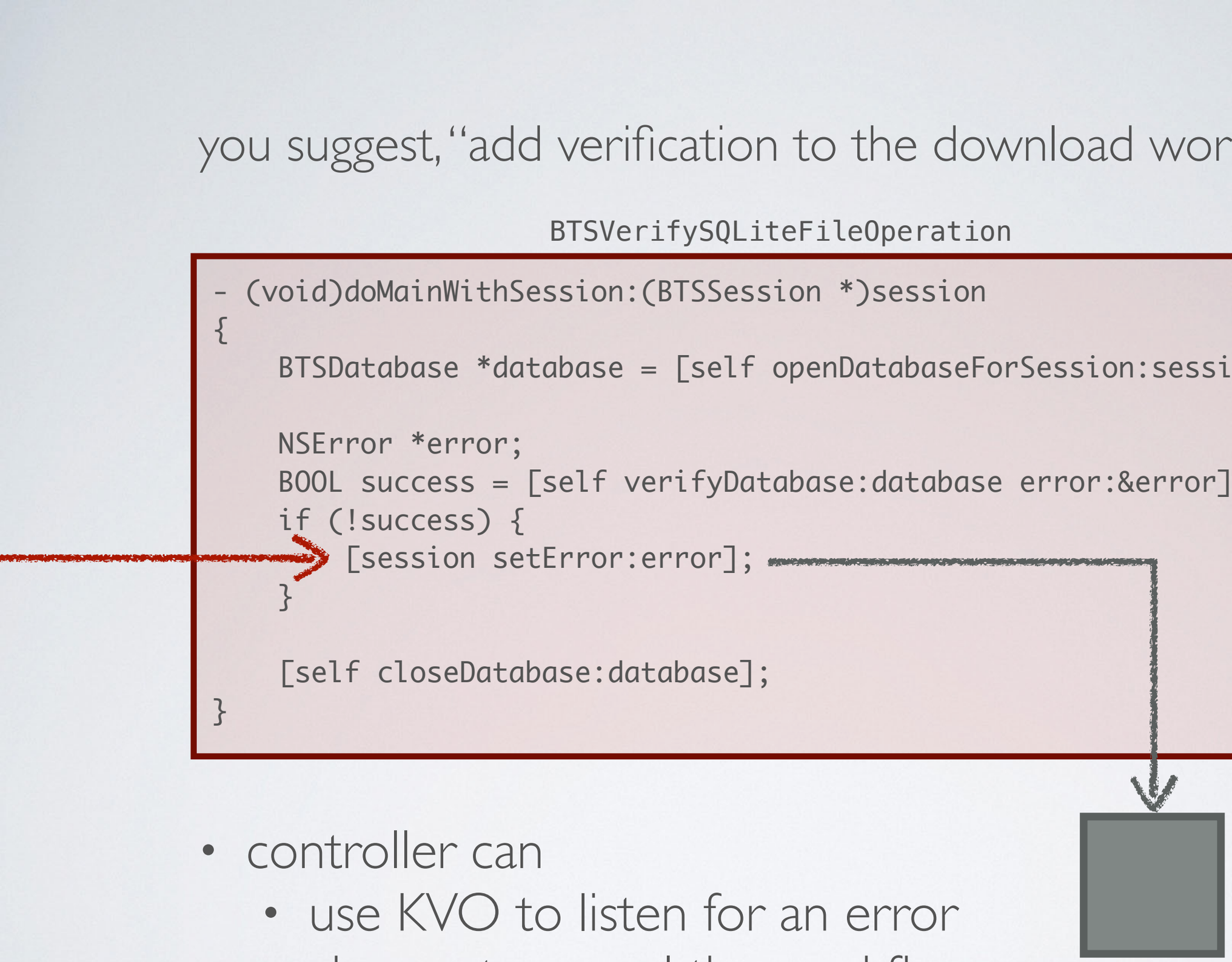


- controller can
  - use KVO to listen for an error

# you suggest, "add verification to the download workflow"

BTSVerifySQLiteFileOperation

```
- (void)doMainWithSession:(BTSSession *)session
{
    BTSDatabase *database = [self openDatabaseForSession:session];

    NSError *error;
    BOOL success = [self verifyDatabase:database error:&error];
    if (!success) {
        [session setError:error];
    }

    [self closeDatabase:database];
}
```



- controller can
  - use KVO to listen for an error
  - choose to cancel the workflow

you suggest, "add verification to the download workflow"

BTSVerifySQLiteFileOperation

```
- (void)doMainWithSession:(BTSSession *)session
{
    BTSDatabase *database = [self openDatabaseForSession:session];

    NSError *error;
    BOOL success = [self verifyDatabase:database error:&error];
    if (!success) {
        [session setError:error];
    }

    [self closeDatabase:database];
}
```







- controller can
  - use KVO to listen for an error
  - choose to cancel the workflow
  - choose to automatically restart, if appropriate

let's add the operation to the workflow

BTSFetchLatestMetaDataJSONOperation

publishes downloaded sqlite meta-data json

BTSDownloadSQLiteFileOperation

consumes json

publishes sqlite gzip cache file url

"session"

BTSDecompressSQLiteFileOperation

consumes gzip cache file url

publishes sqlite cache file url

BTSMoveSQLiteFileOperation

consumes sqlite cache file url

publishes sqlite file url to app support directory

`BTSDownloadSQLiteFileOperation`

consumes json

publishes sqlite gzip cache file url

`BTSDecompressSQLiteFileOperation`

consumes gzip cache file url

publishes sqlite cache file url

“session”

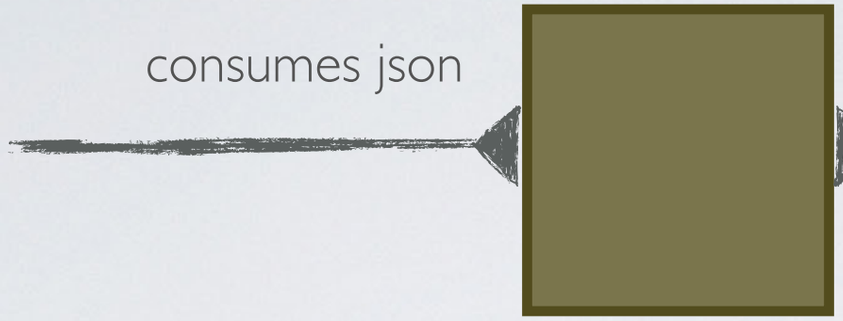
`BTSMoveSQLiteFileOperation`

consumes sqlite cache file url

publishes sqlite file url to app support directory

`BTSDownloadSQLiteFileOperation`

consumes json



publishes sqlite gzip cache file url

`BTSDecompressSQLiteFileOperation`

consumes gzip cache file url

publishes sqlite cache file url

“session”

`BTSVerifySQLiteFileOperation`

consumes sqlite cache file url

`BTSMoveSQLiteFileOperation`

consumes sqlite cache file url

publishes sqlite file url to app support directory

# WORKFLOW EXTRAS

- progress publishing
- simplified cancellation handling
- simplified error handling
- easy to document and understand
- unit test operations in isolation

# NOTHING NEW HERE

- nothing special about queues and operations

- can do the same thing on other platforms

# WHAT'S A BOX AND ARROW?

it's writing great software