St. Louis CocoaHeads

Brian Coyner
Principal Software Engineer @ NISC

Techniques for building asynchronous workflows

August 2018

About Session

- Look at techniques for using NSOperation and NSOperationQueue to:
  - reduce app complexity
  - speedup development time
  - reduce bugs
  - easy to understand Cancellation Policy
  - ensure a consistent developer experience

Nothing New Here

- NSOperation and NSOperationQueue have been around a long time
- Added some additional APIs to help make it easier for my team
- Your desires and needs may vary

Where's The CocoaPod?

- Yes, it is a Framework in my projects
- Use the sample code as inspiration
- If you like it, then be sure to give me some credit

What About That WWDC Talk?

- Dave DeLong did a great talk called "Advanced NSOperations" in 2015
  - https://developer.apple.com/videos/play/wwdc2015/226/

- Review all solutions and determine what makes sense for you, your team, and your customers
- Validated what I built (which is nice)

What About...?

- Futures and Promises?
- Grand Central Dispatch?
- Locks and Threads?

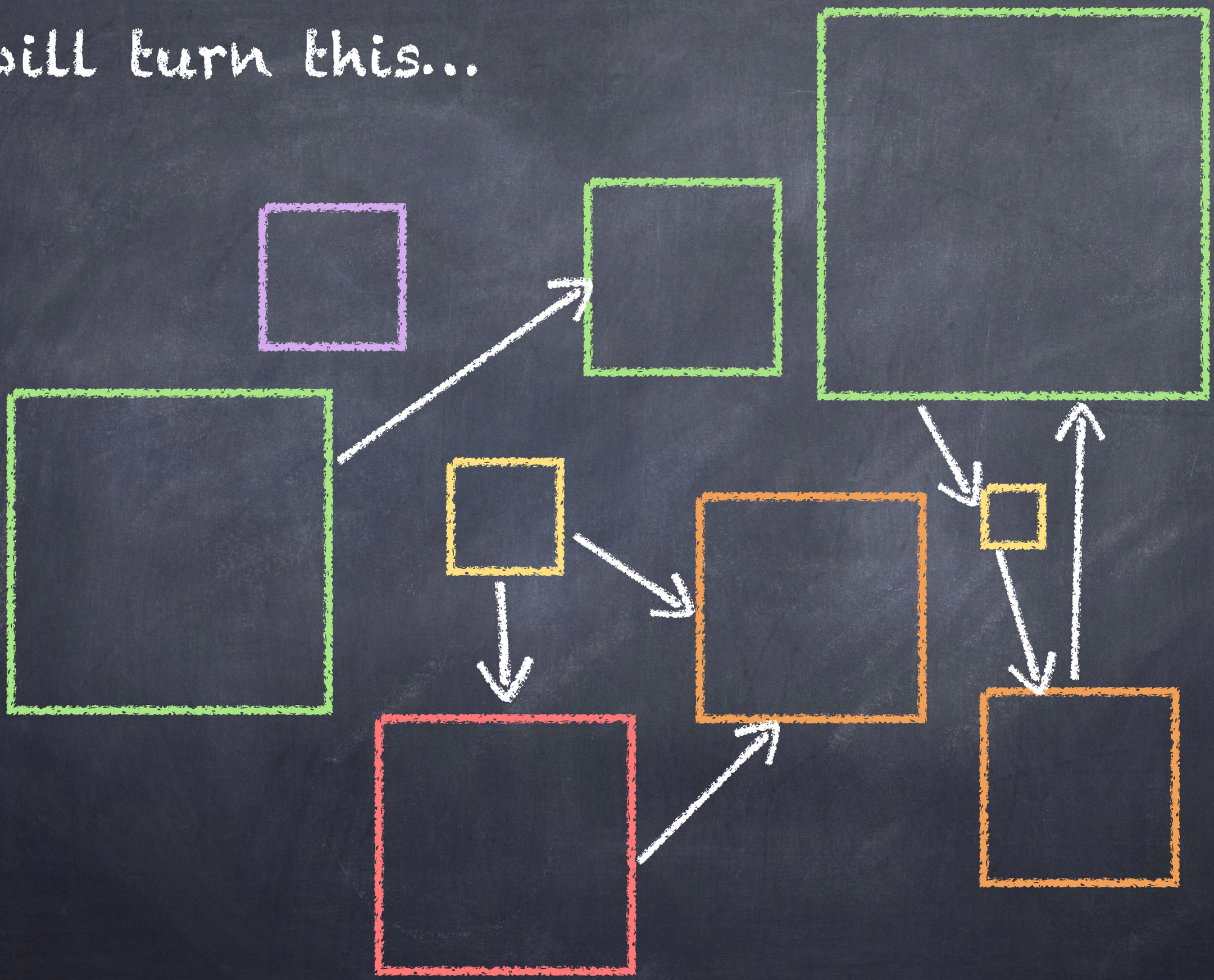
Next Steps

- Review some GCD APIs
- Review NSOperation and NSOperationQueue
- Look at some cool ways to build async workflows

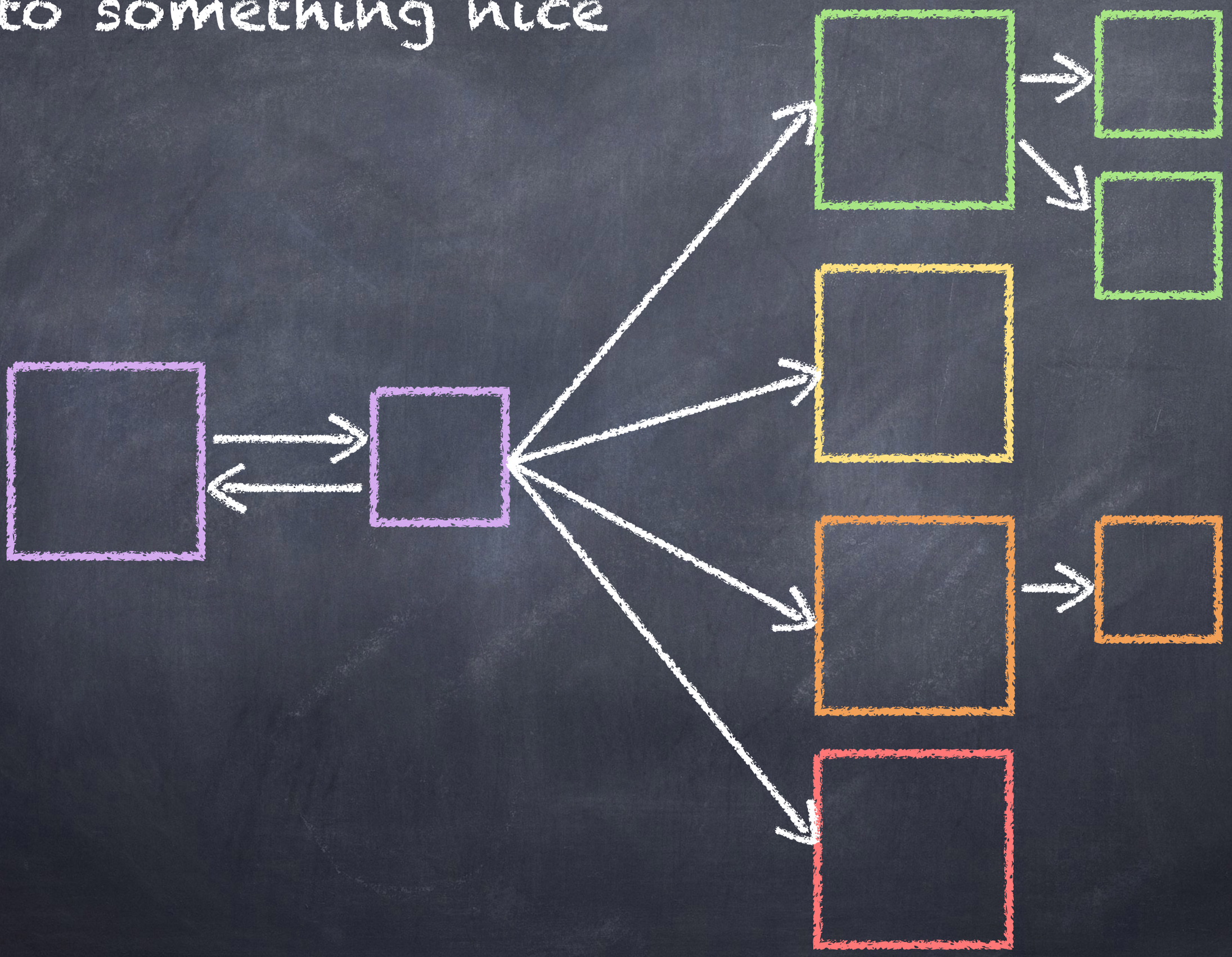
Asynchronous workflows

Apps have
workflows

will turn this...



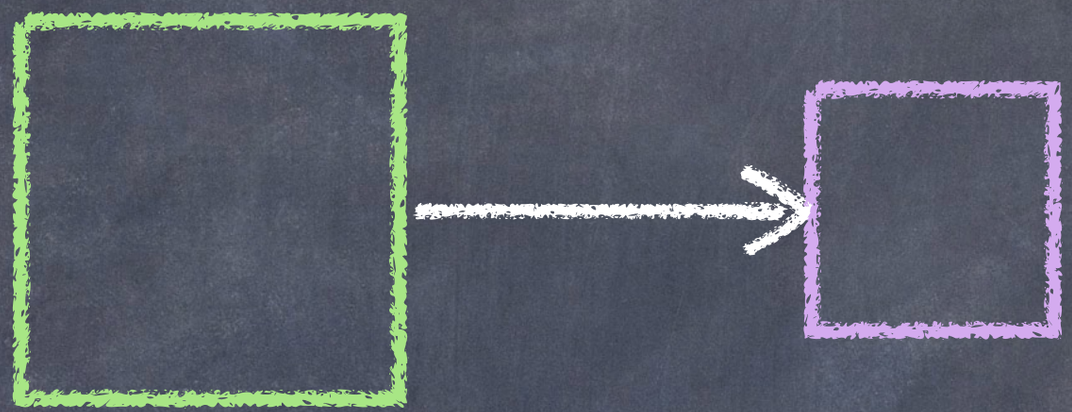
into something nice



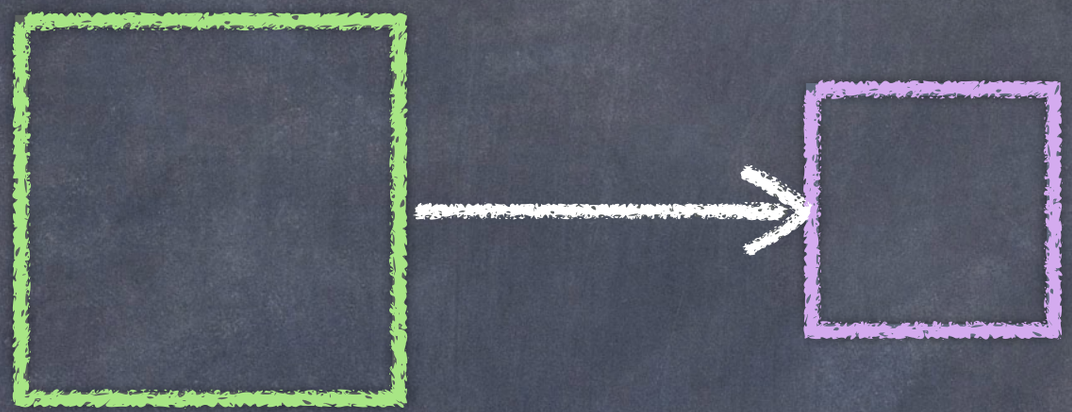
Example Workflows

- Sign in users with dynamic on-boarding steps
- Connect to external IP hardware
- Download large files
- Stream data into data stores
- Execute SQLite FTS searches
- Dynamic "action" discovery
- Sign out users

Let's Look At Various Ideas

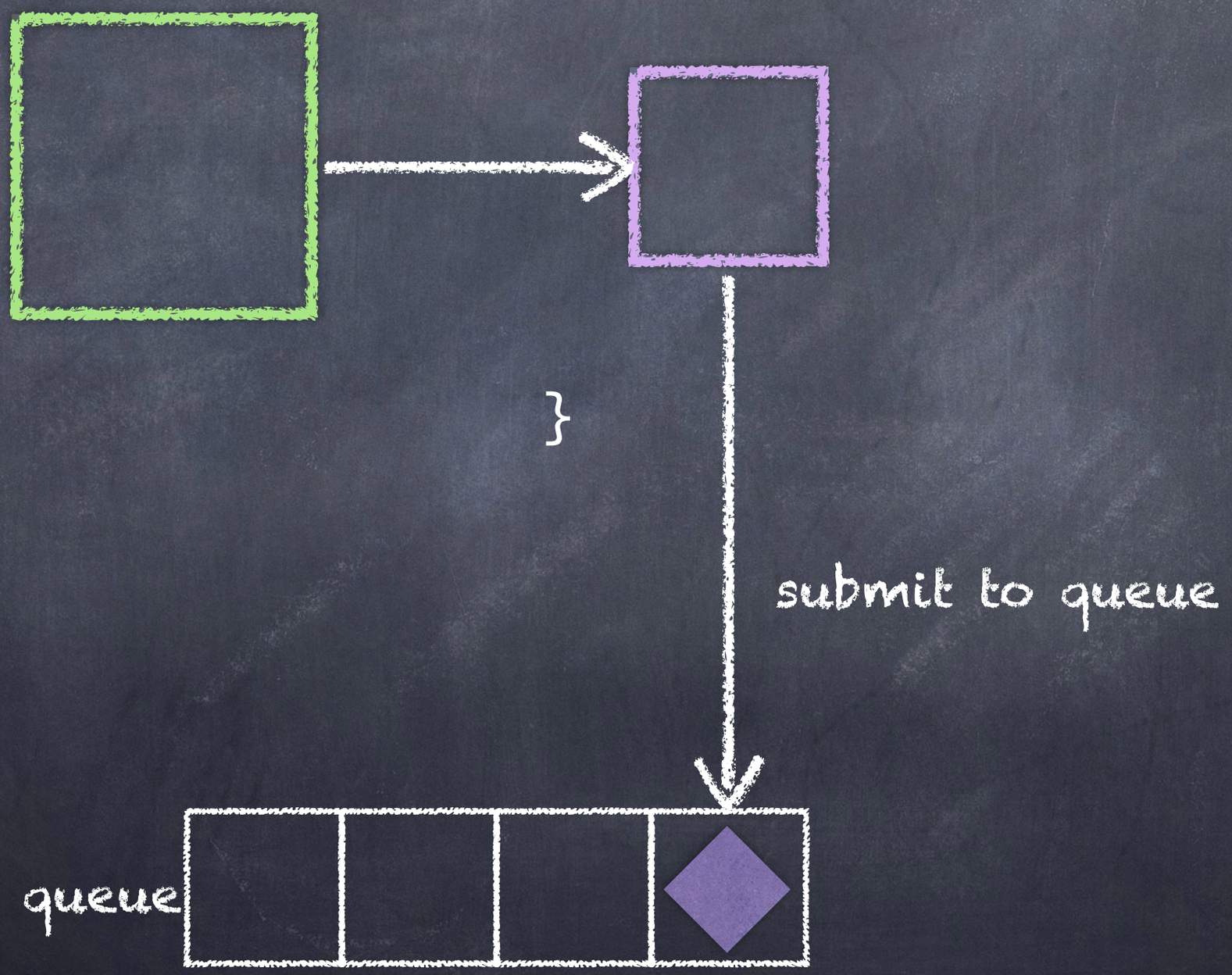


synchronous call to a body of code

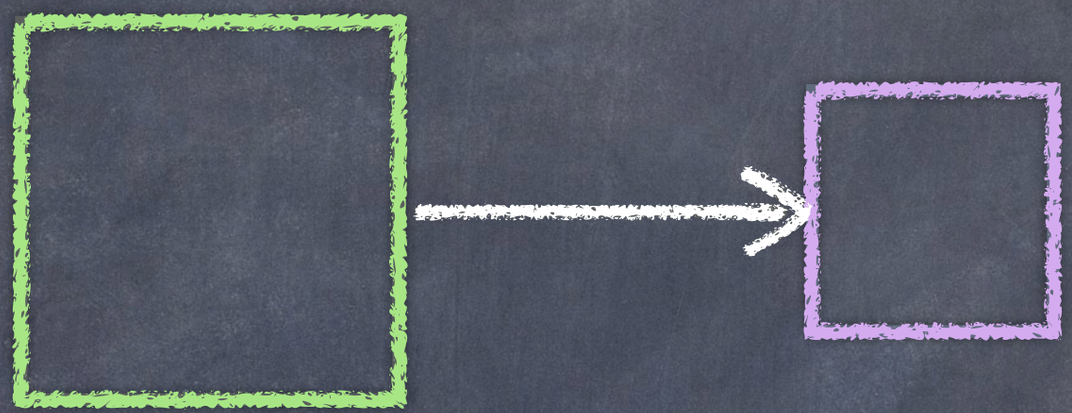


you say, "let's make this asynchronous"

```
dispatchQueue.async {

}
```

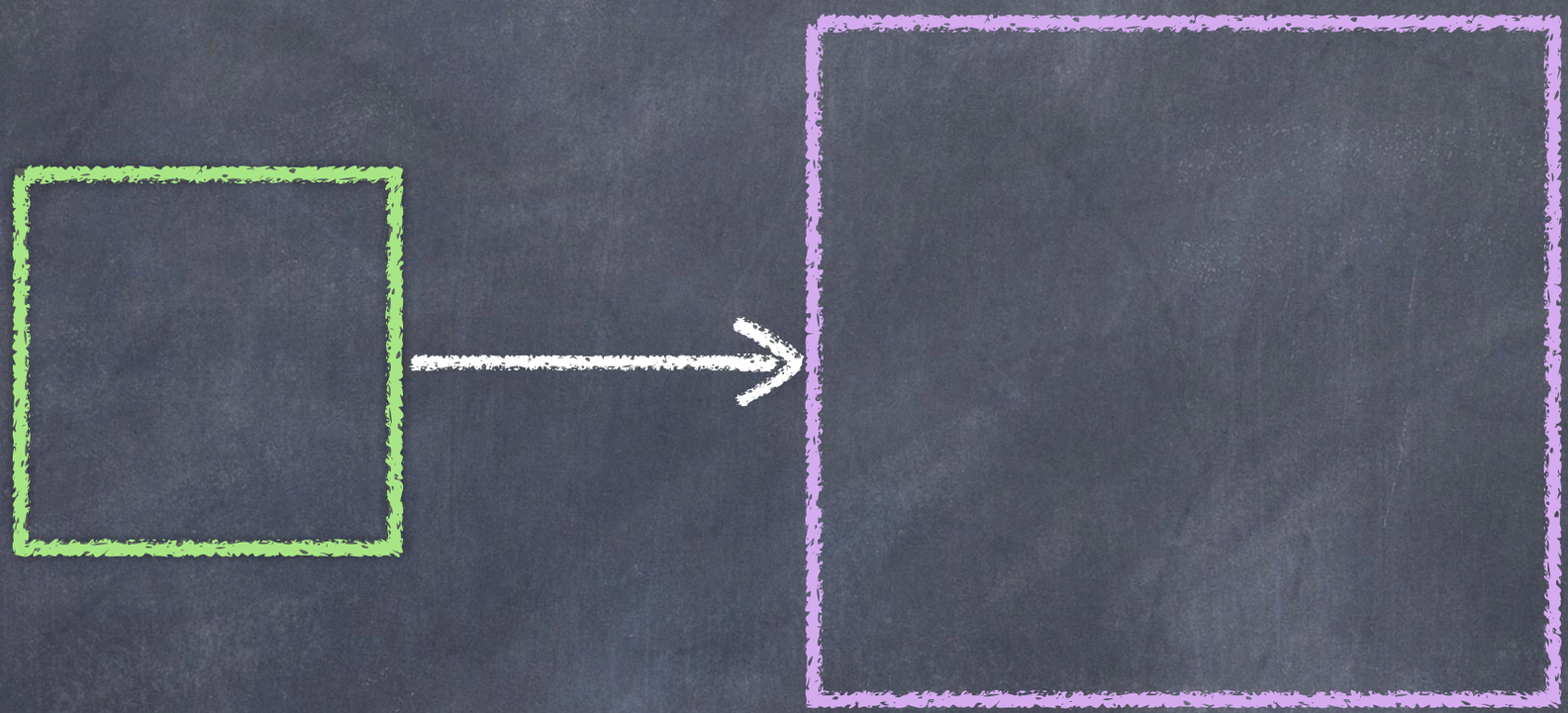


```
dispatchQueue.async {
```

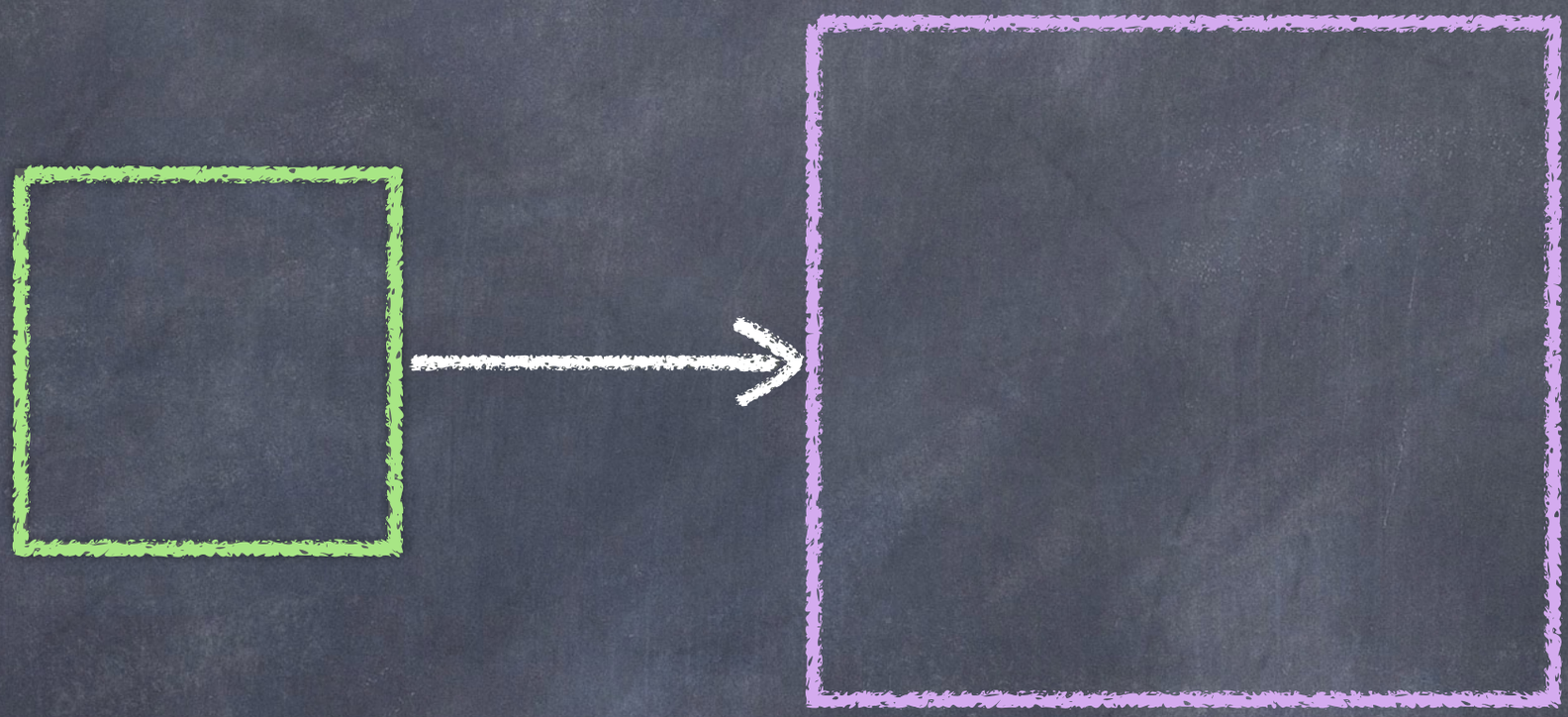


```
}
```

works well if work is very short lived

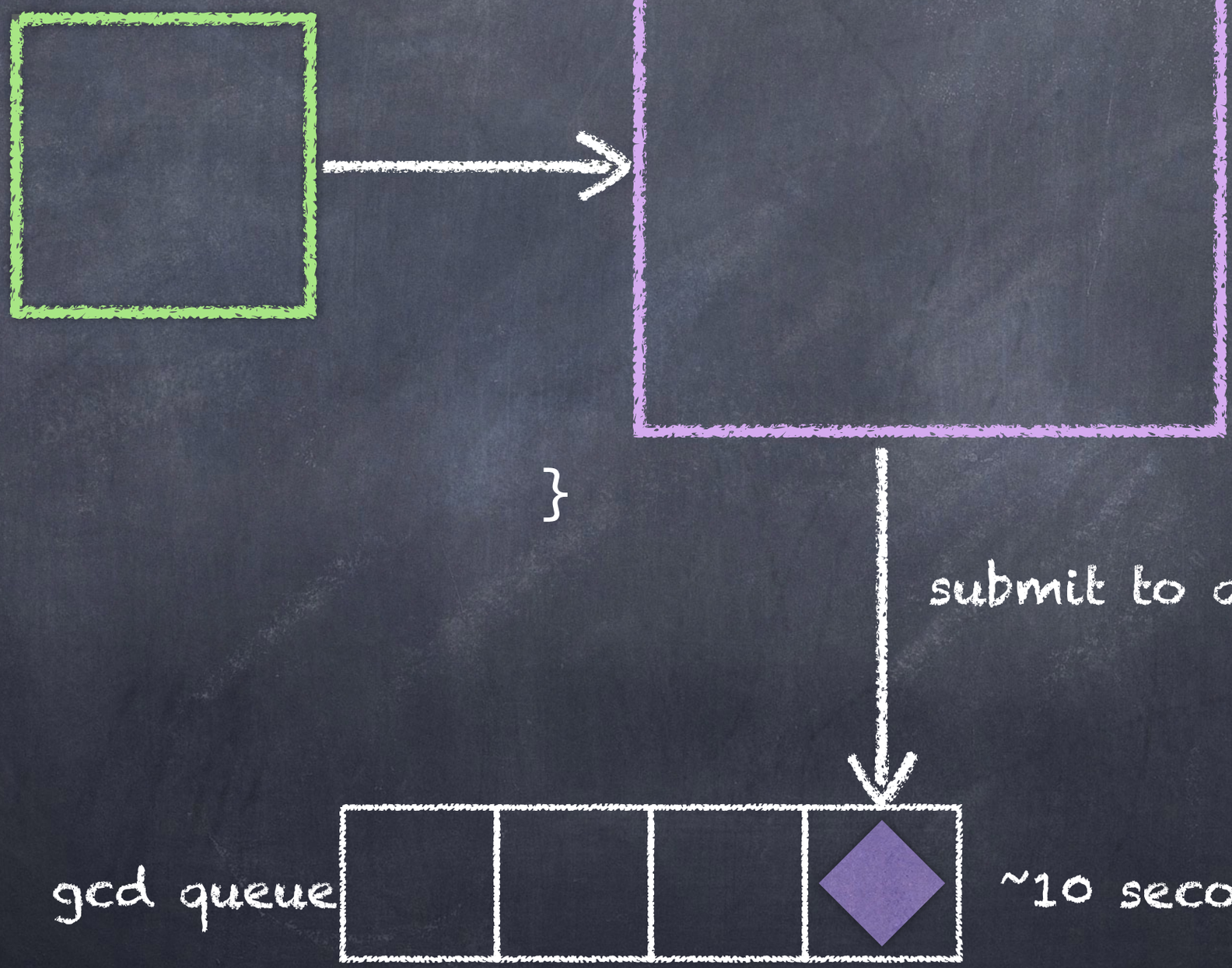


synchronous call to a longer running body of work



you say, "let's make this asynchronous"

dispatchQueue.async {



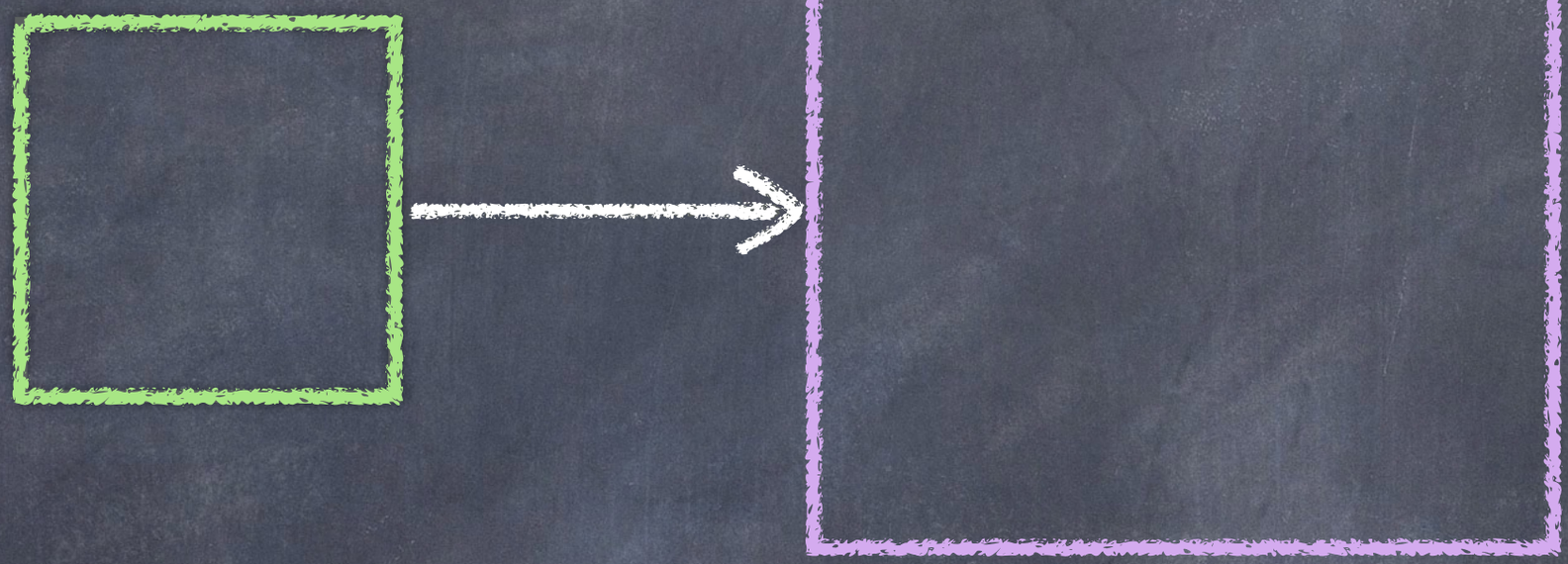
}

submit to queue

gcd queue

~10 seconds

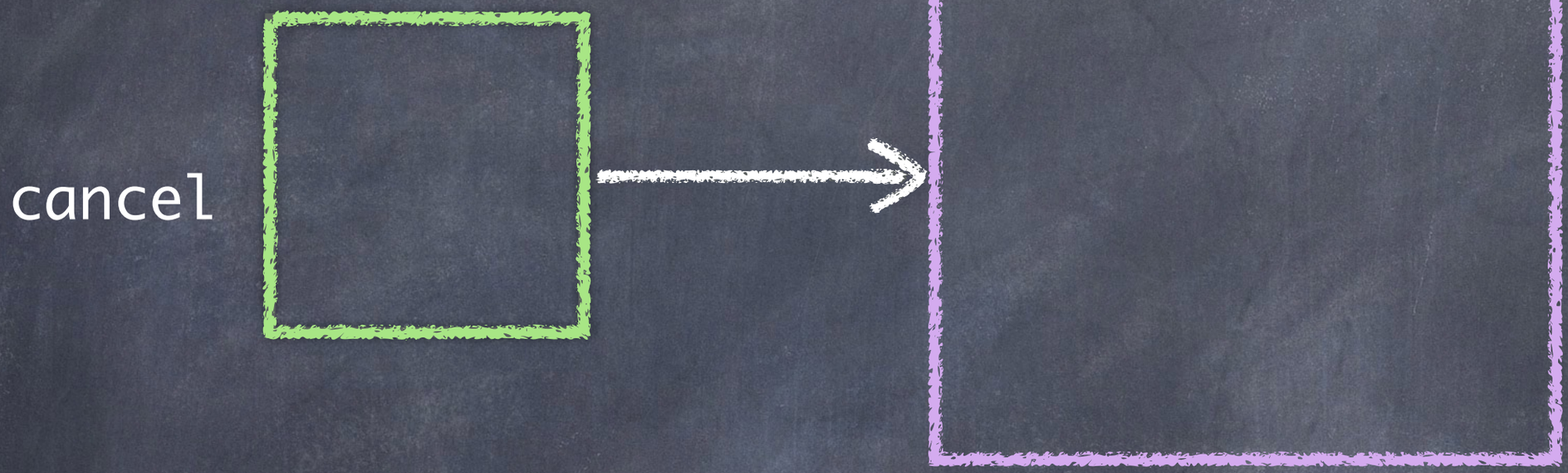
```
dispatchQueue.async {
```



```
}
```

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

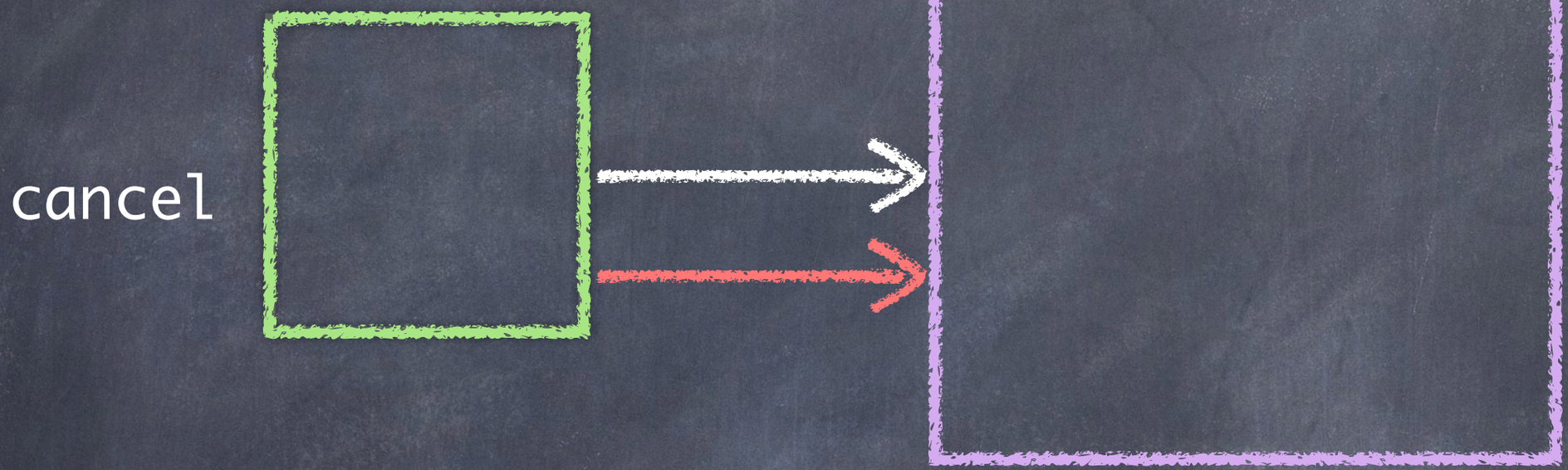
dispatchQueue.async {



}

you start adding cancel API

dispatchQueue.async {



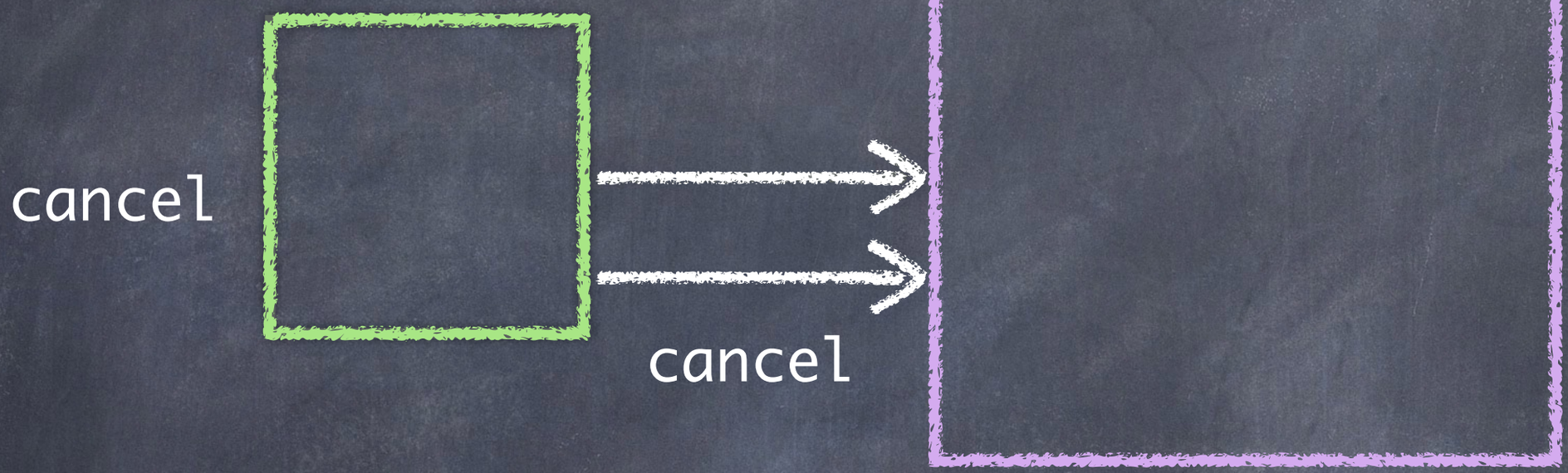
}

you say, "huh? i cant stop it!"

```
let work = DispatchWorkItem {

}

dispatchQueue.async(execute: work)
```

ok... let's use a DispatchWorkItem

```
let work = DispatchWorkItem {

  }

dispatchQueue.async(execute: work)
```



Seems ok...

GCD is awesome, but we can do better with NSOperation

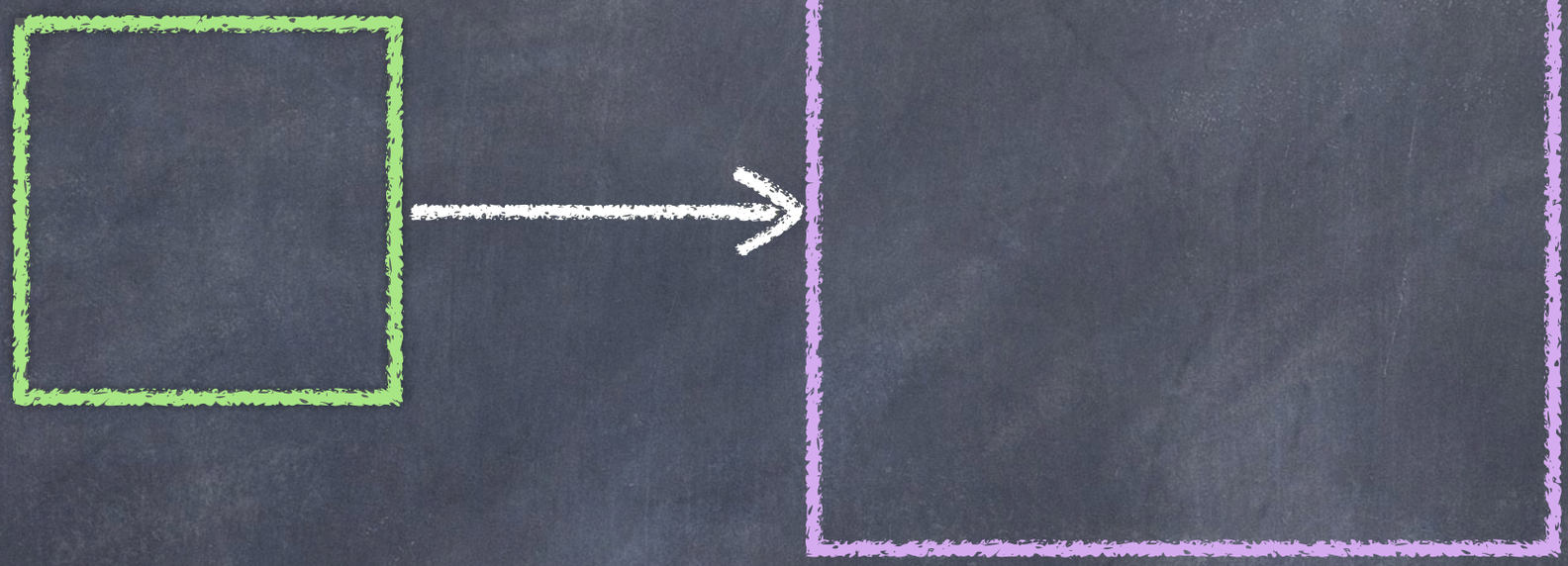
- higher level
- dependent operations
- reusable
- cancellable
- progress reporting
- unit test-able

The Async Workflow implementation will still use GCD...

- multiple readers
- single writer

NSOperation
&
NSOperationQueue

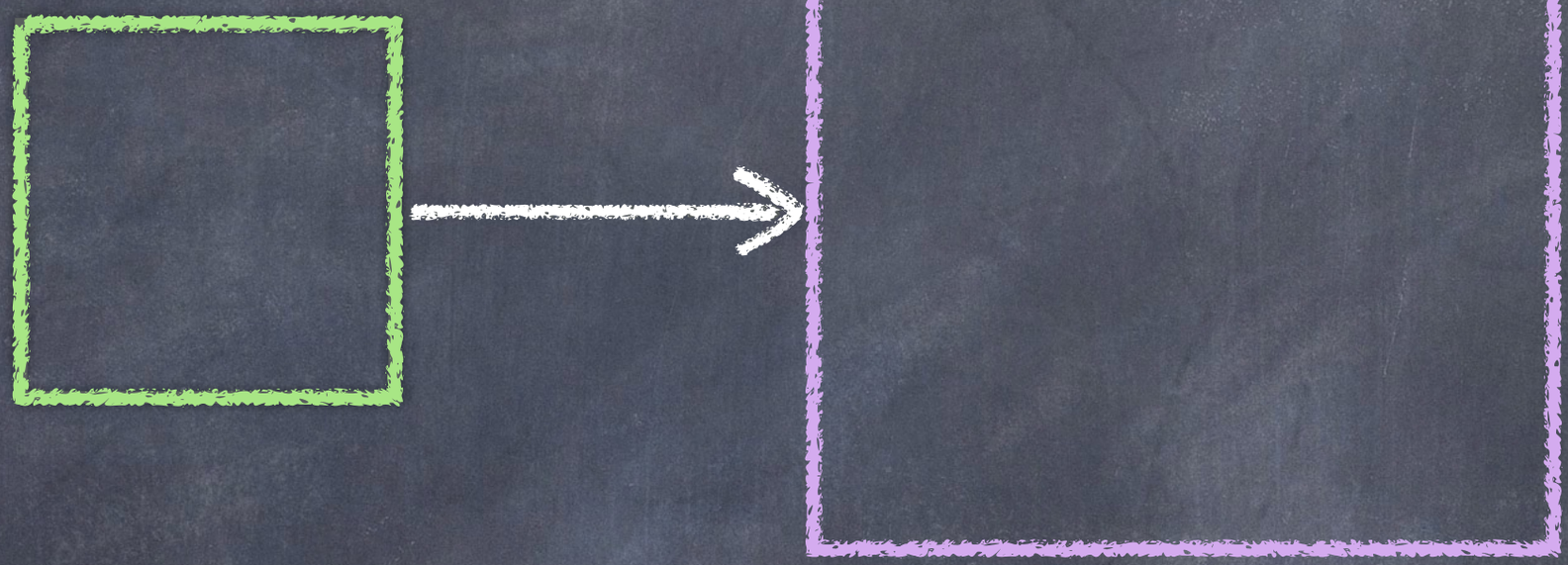
```
dispatchQueue.async {
```



```
}
```

transition to operations

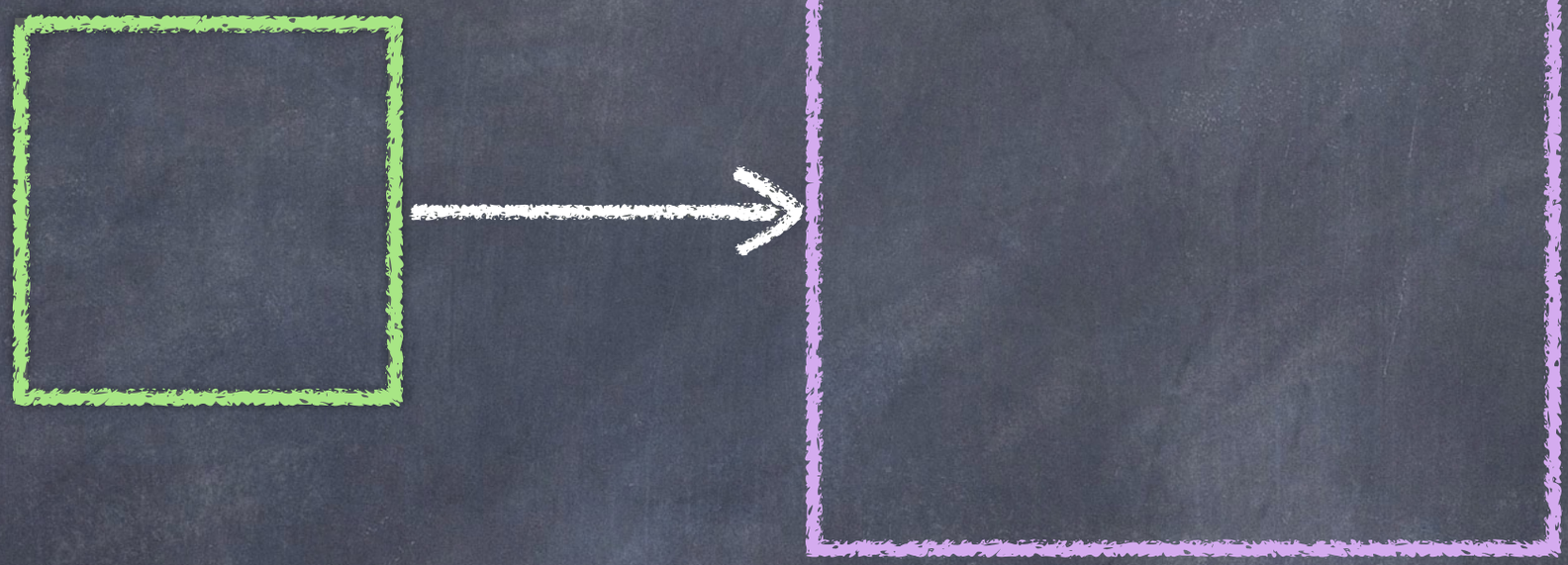
```
func void main() {
```



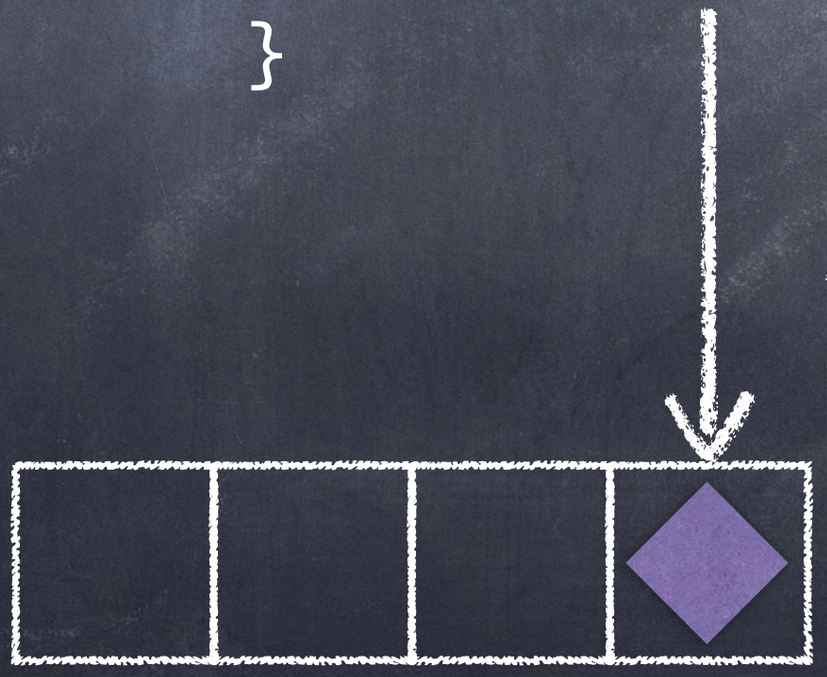
```
}
```

```
final class MyOperation: Operation
```

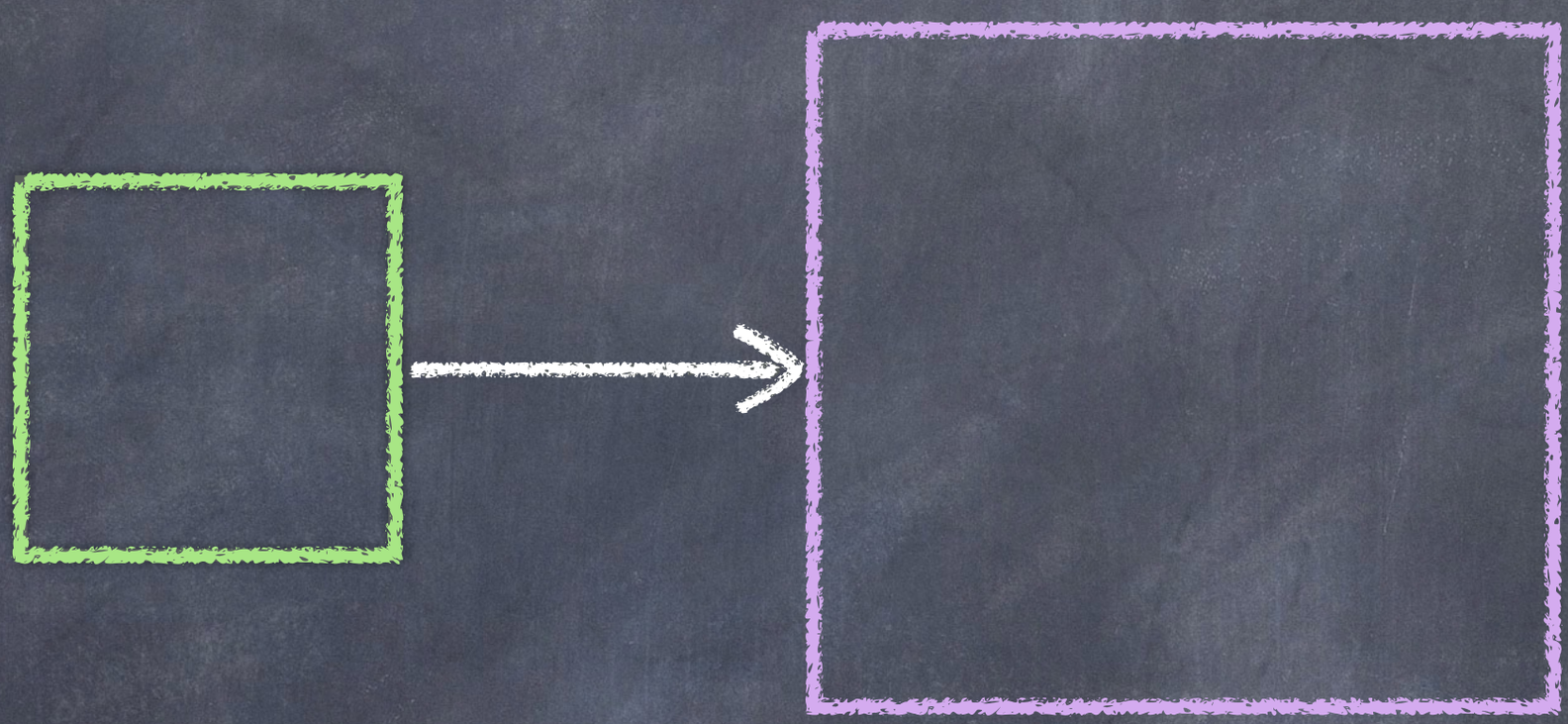
```
func void main() {
```



```
}
```



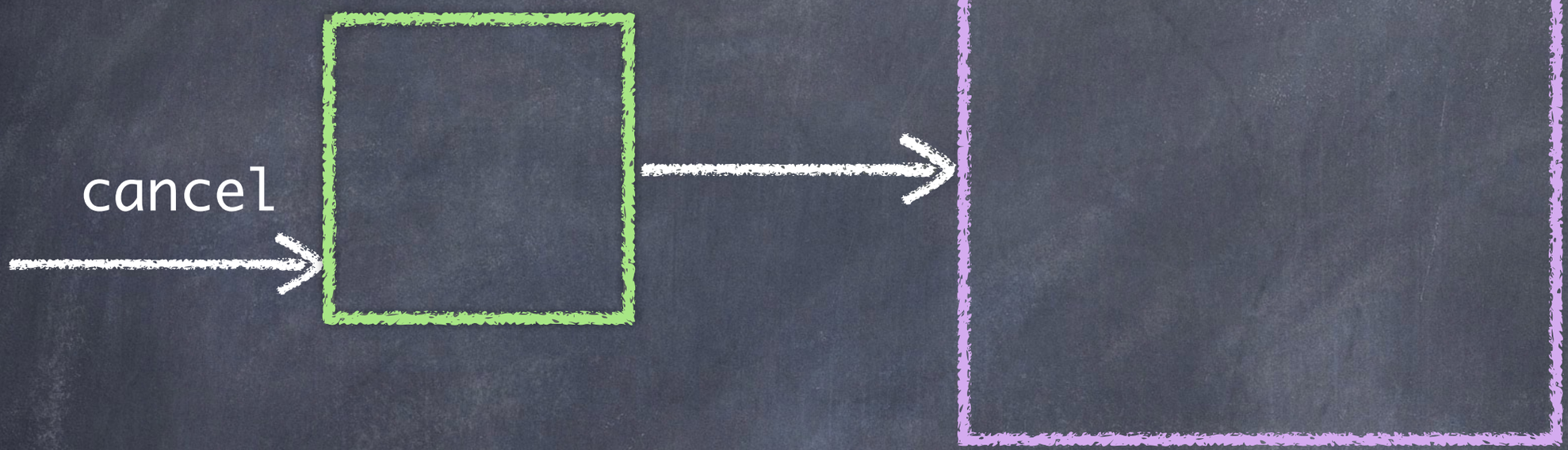
```
func void main() {
```



```
}
```

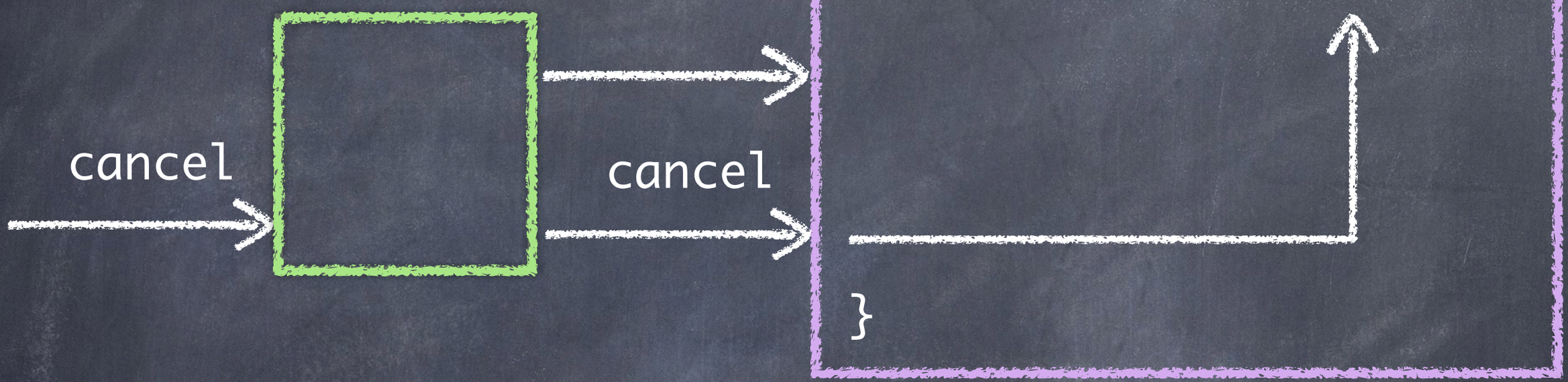
you say, "let's cancel"

```
func void main() {
```



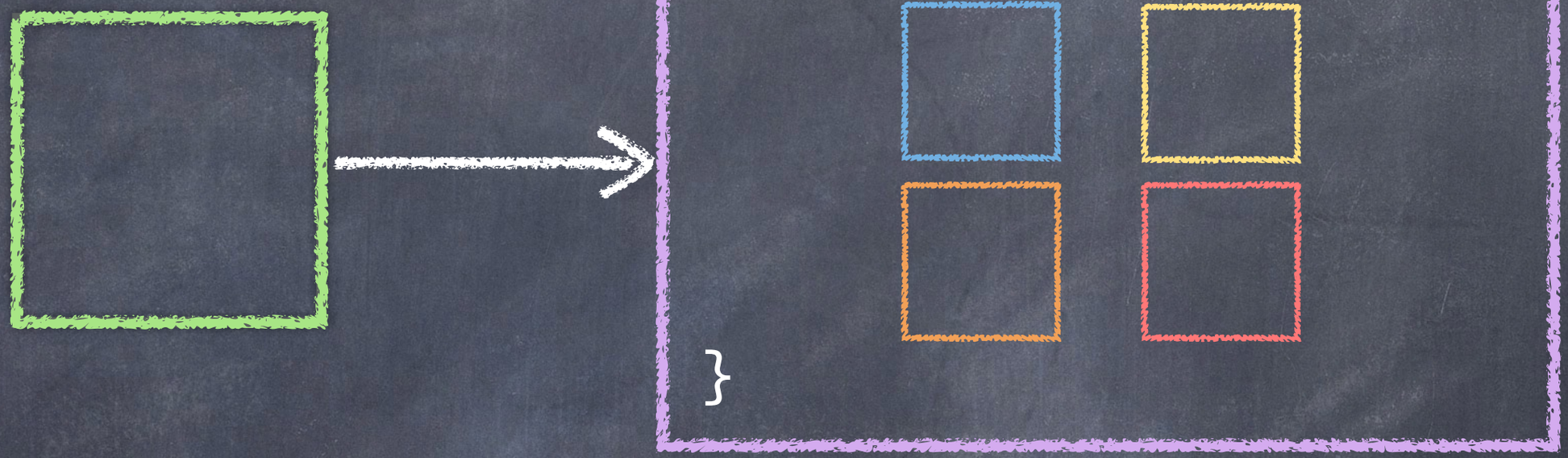

```
}
```

```
func void main() {
```




```
}
```

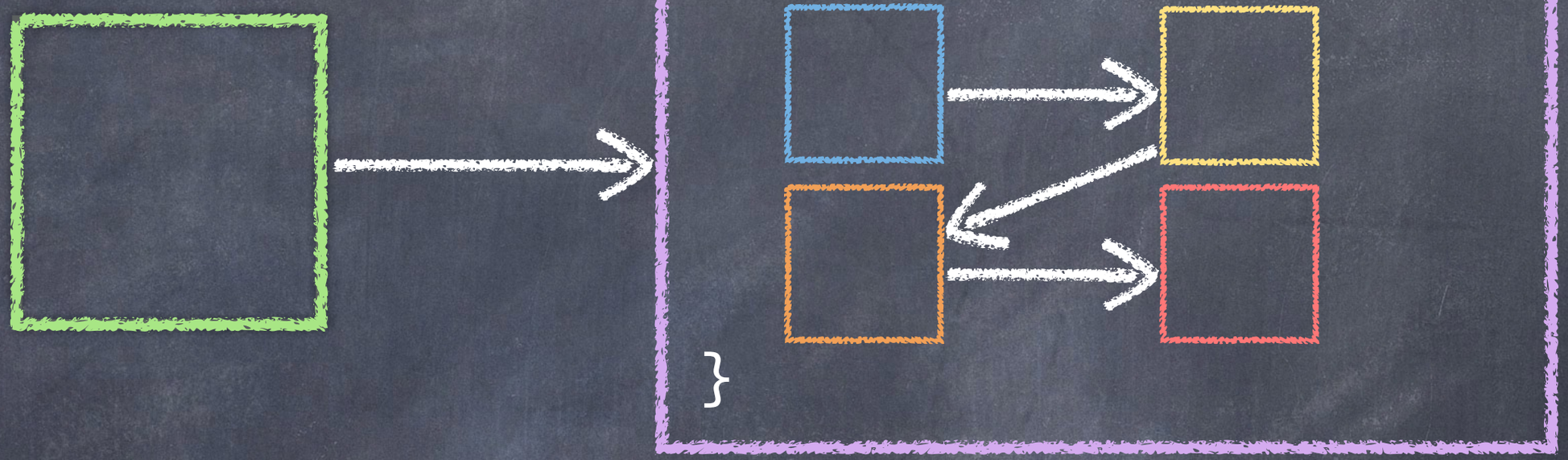
```
func void main() {
```




```
}
```

there are lots of small "units of work"
wrapped up in methods

```
func void main() {
```

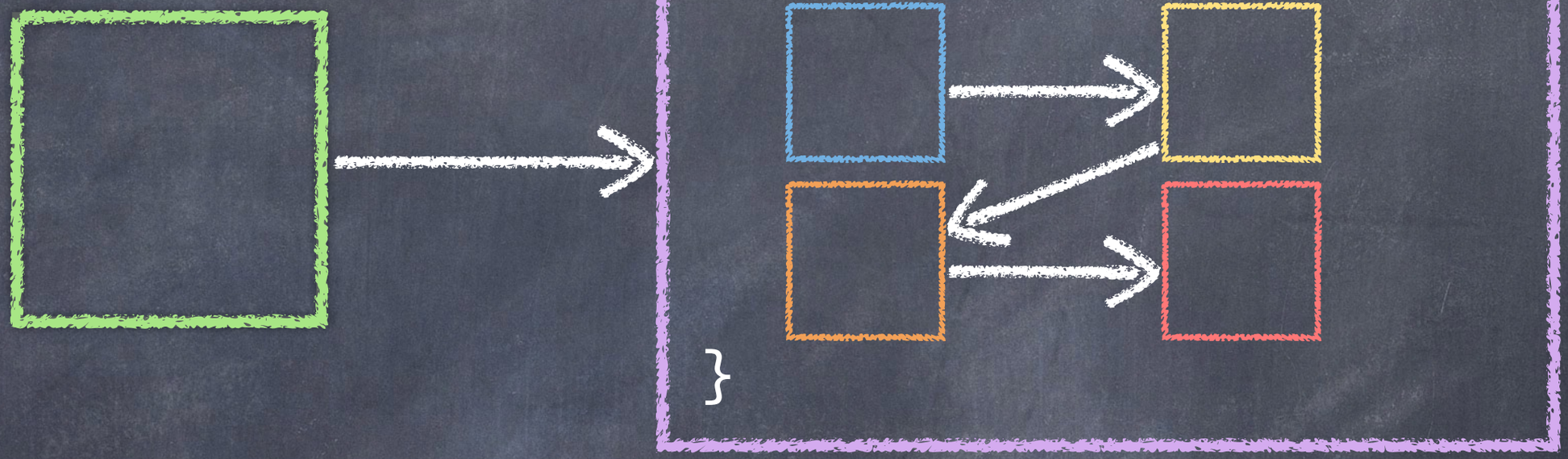



```
}
```

each "unit of work" consumes and/ or produces data

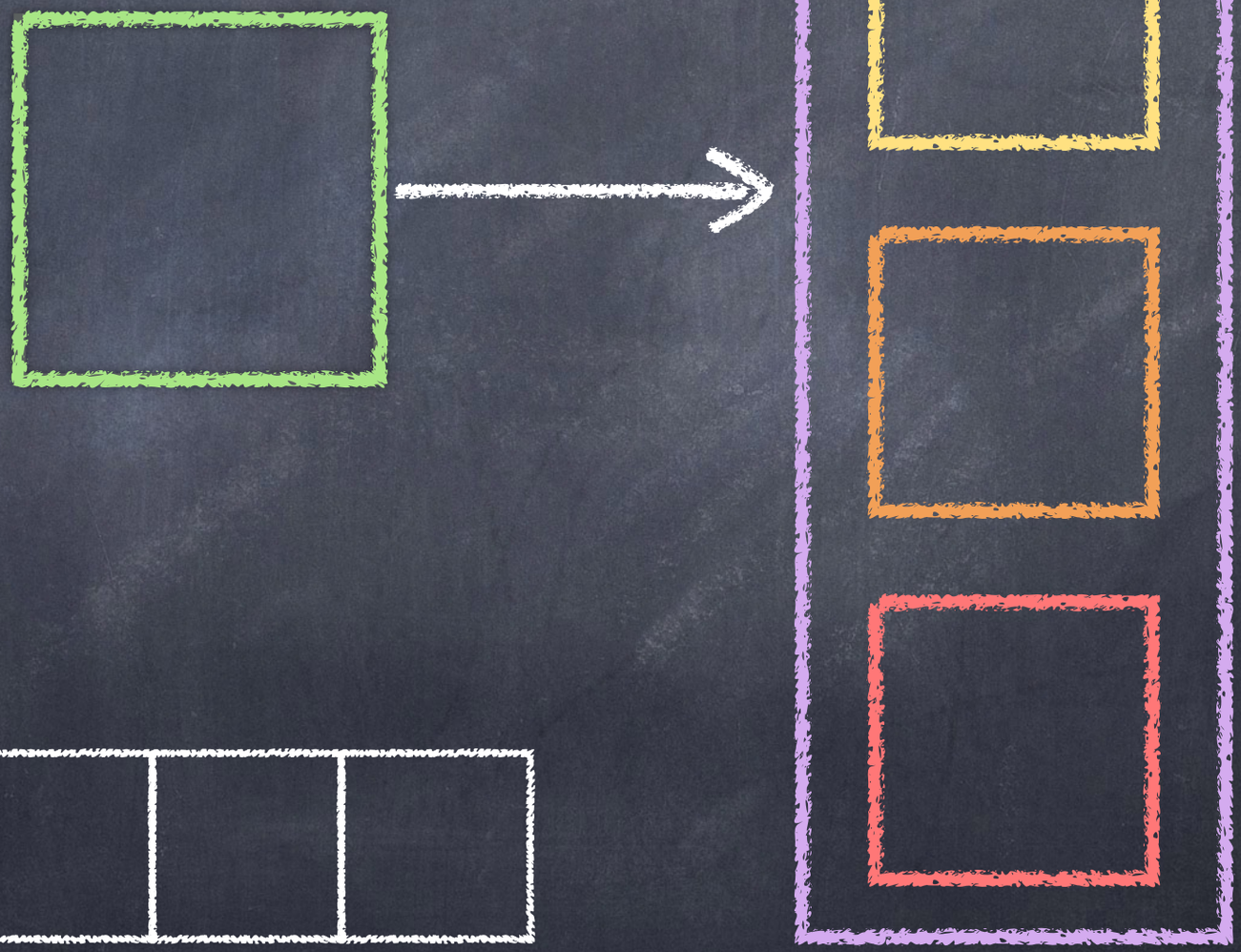
Operation Composition

```
func void main() {
```




```
}
```

Let's break each "unit of work" into separate operations



operation queue

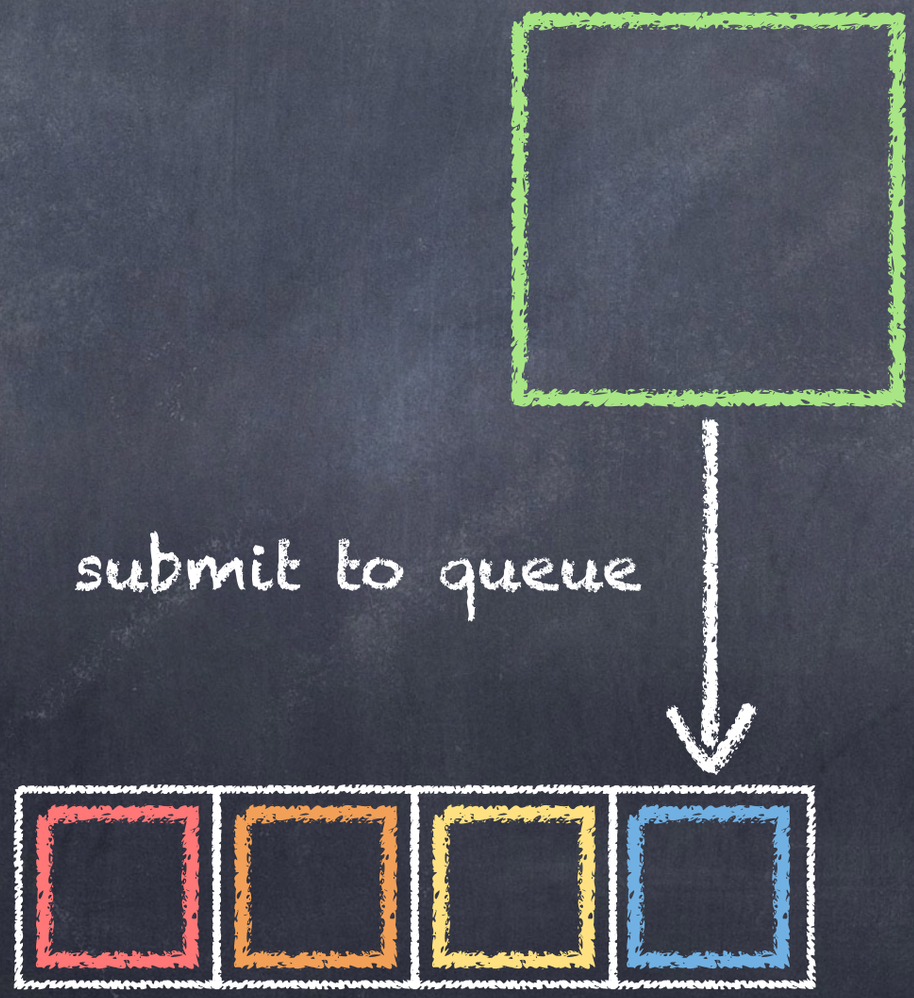
the type of operation queue depends on the "workflow"

submit to queue

operation queue

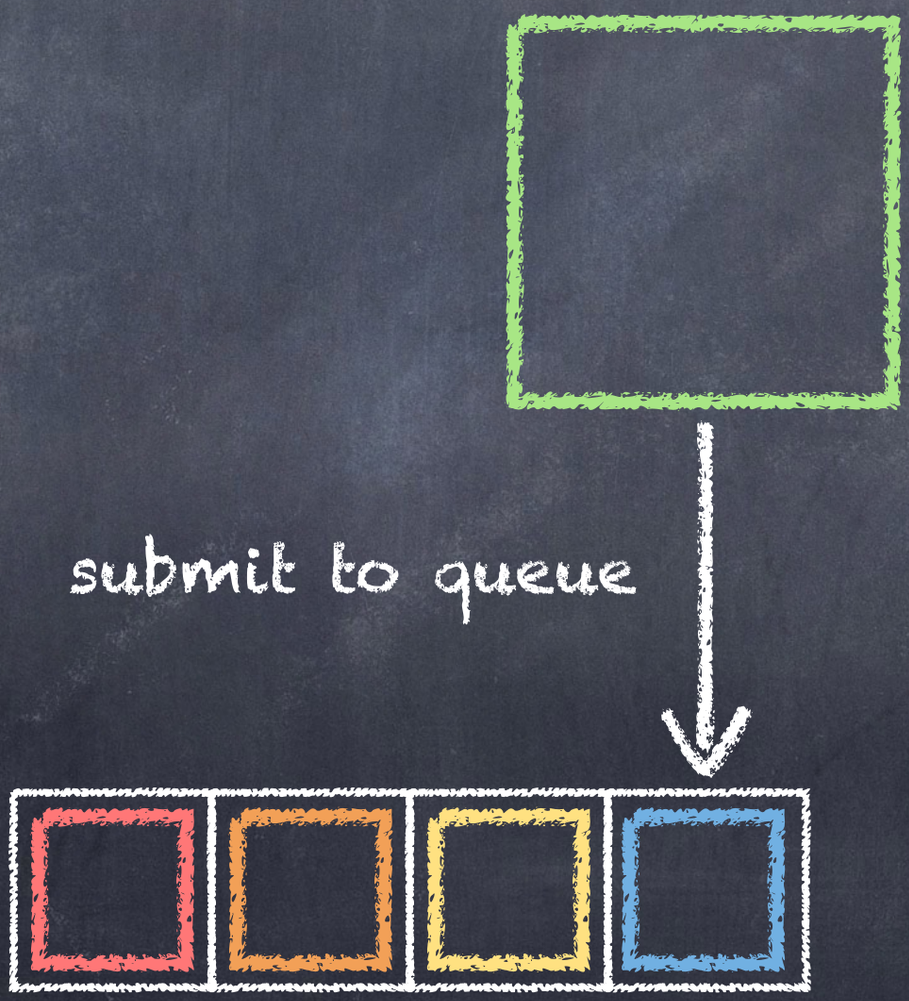
use a serial queue for a serial workflow

```
operationQueue.maxConcurrentOperationCount = 1
```

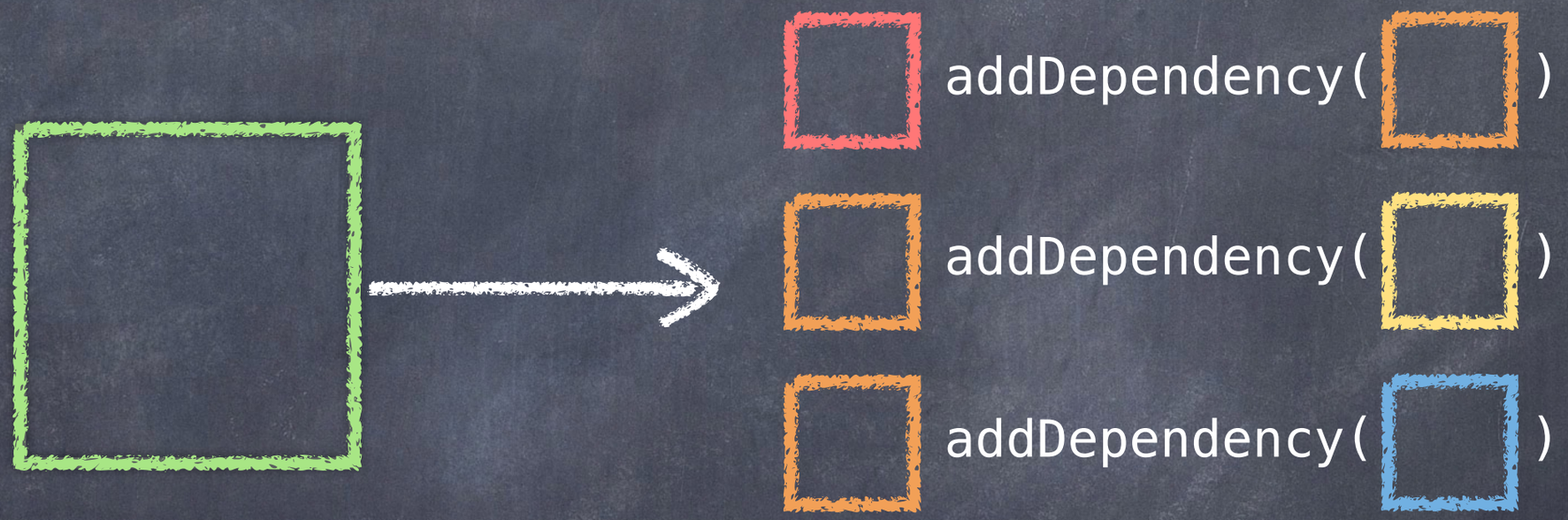


use a concurrent queue with dependent operations for a semi-concurrent "workflow"

```
operationQueue.maxConcurrentOperationCount = 10
```



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

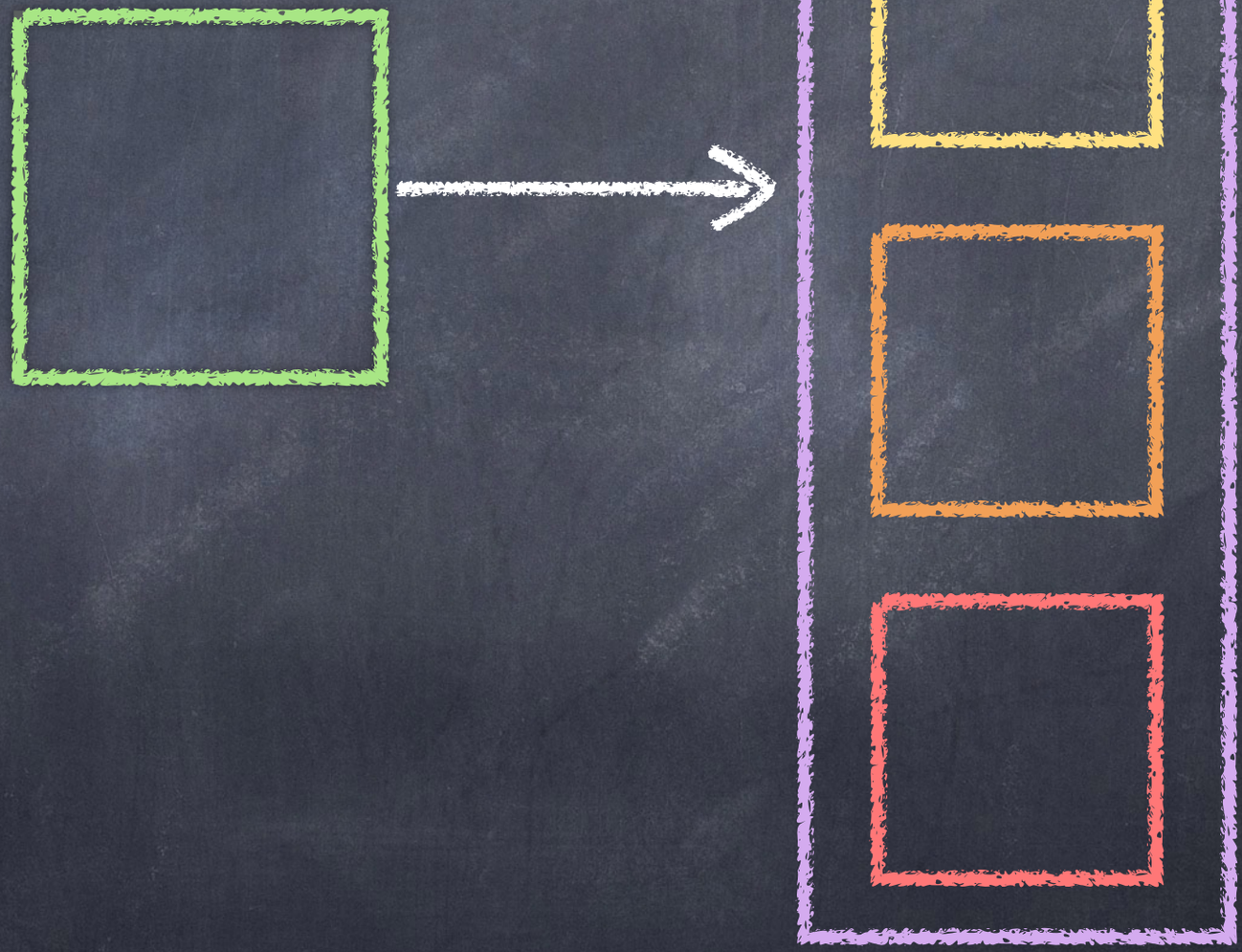


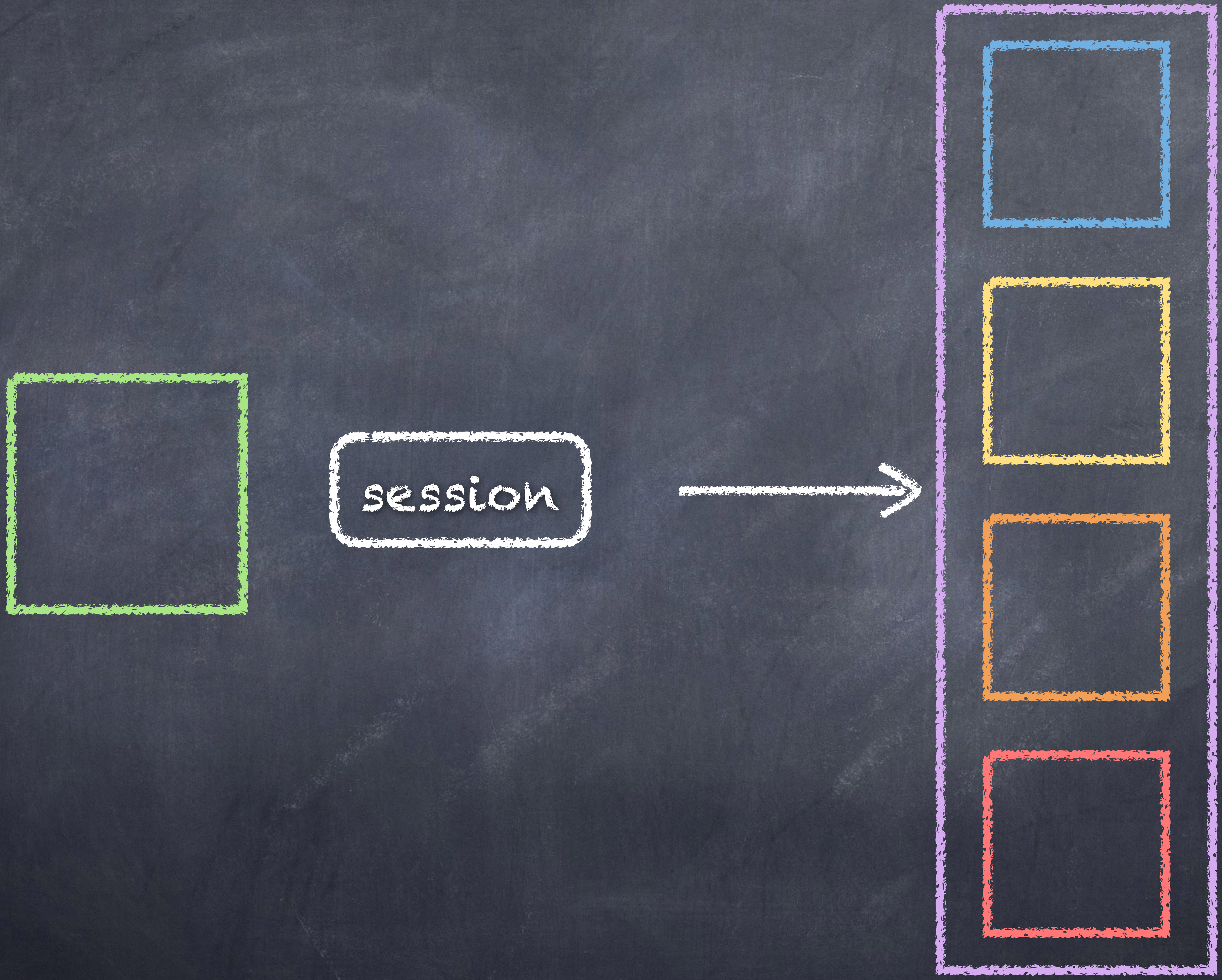
operation queue

That was the standard API

Let's add...

... a way to pass
data between
operations




session

session

- thread-safe dictionary
- multiple readers
- single writer
- pass data between operations
- workflow cancellation API

session

init(session: Session)

init(session: Session)

init(session: Session)

init(session: Session)

Live Code Demo

https://github.com/briancoyner/AsyncWorkflow

- Session API
- Cancellation Policies
- Progress Reporting
- Background Execution

Nothing New Here

- can do the same thing on other platforms

Async Workflow Recap

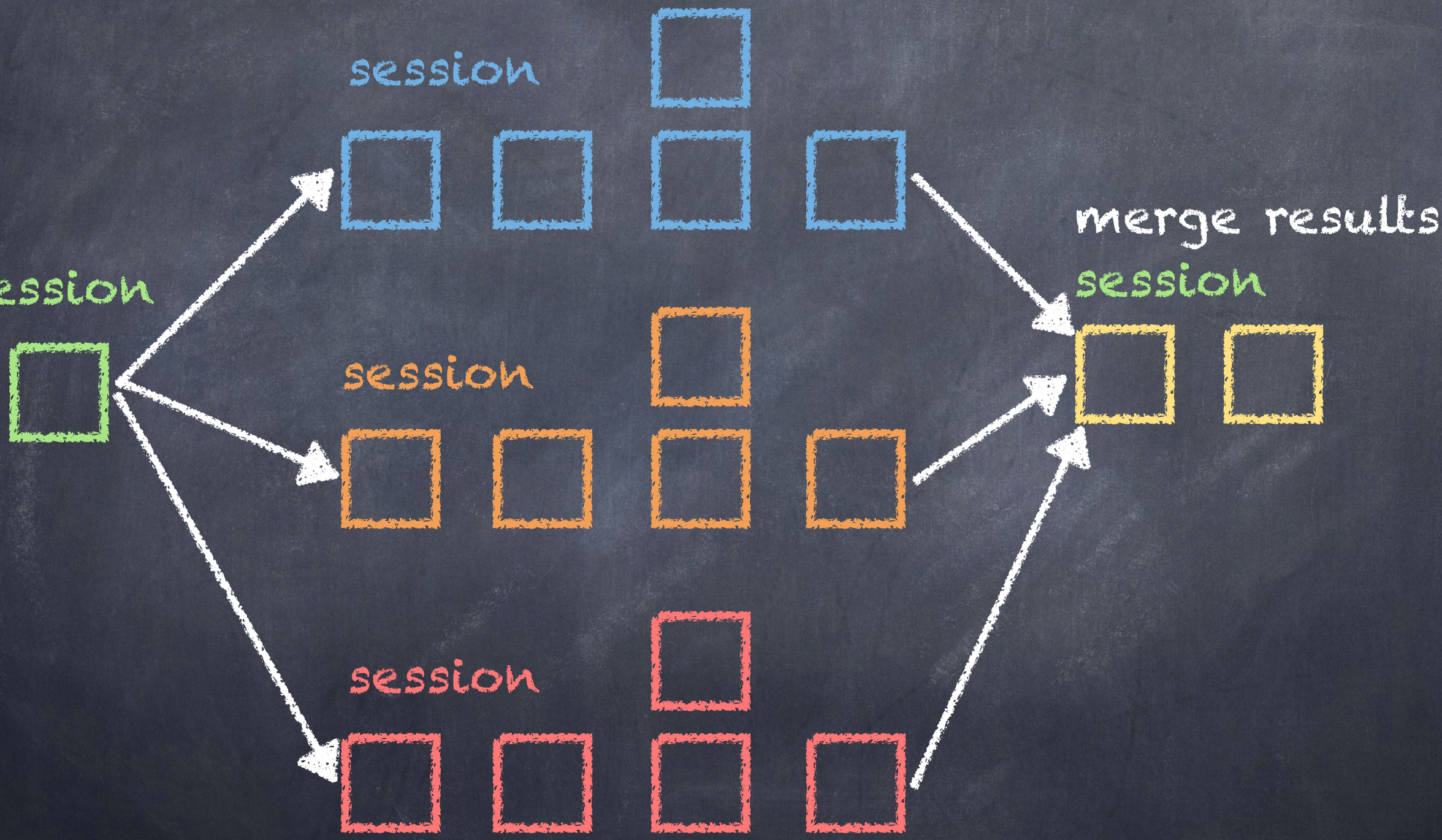
- progress reporting
- simplified cancellation handling
- simplified error handling
- easy to document and understand
- unit test operations in isolation

"Grouped" Workflows

- great for breaking up large workflows into sub-workflows
- merge the results of each group at the end

- not implemented in this demo code (but not hard to implement)

Grouped



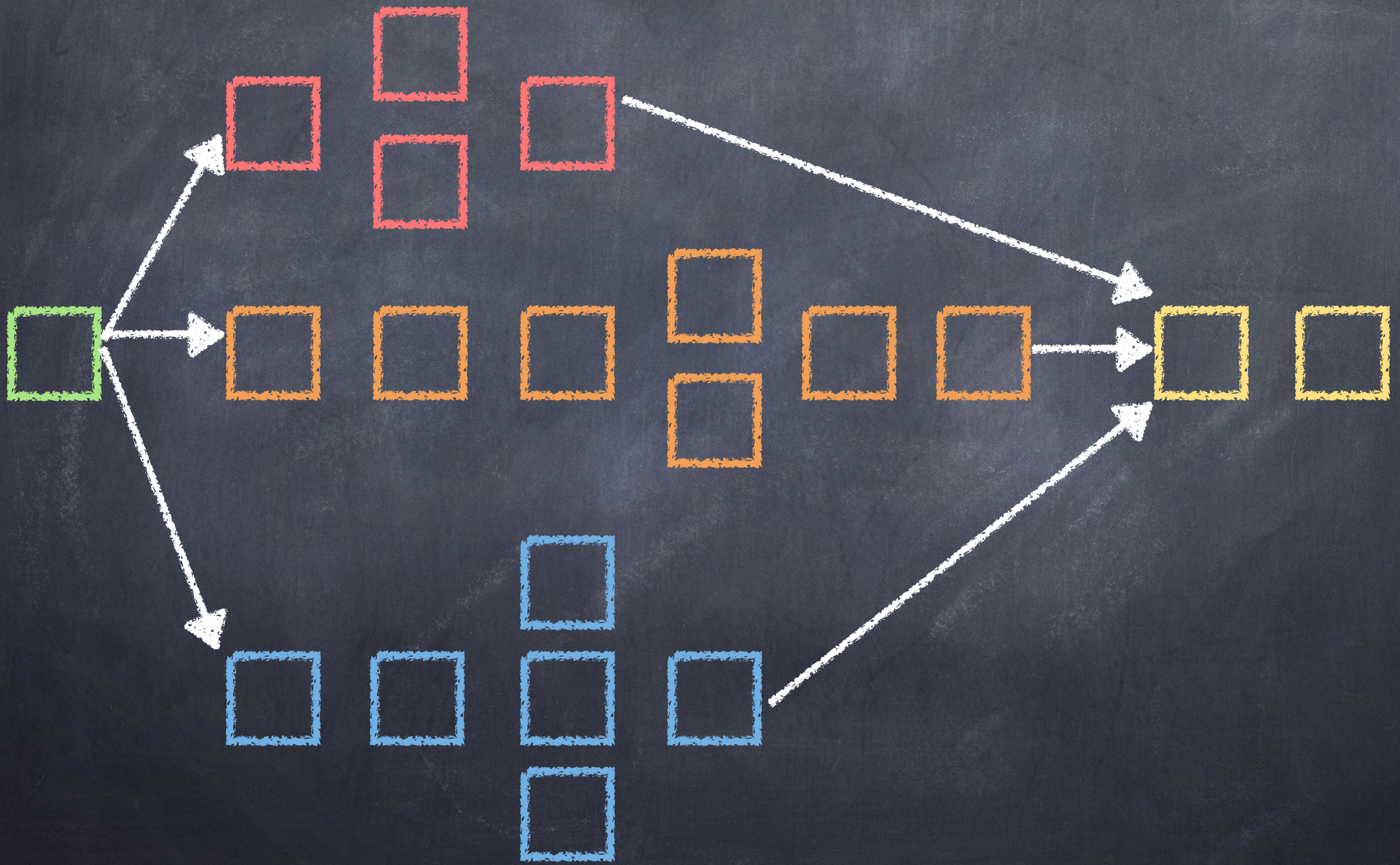
Same operations, different data

Arbitrarily Complex Workflows

- think about what you need to solve
- break into small operations
- set up dependencies
- execute

- don't forget to support cancellation

Arbitrarily Complex



Things Not Covered

- Capturing "warnings"
- Type-safe workflow "controller"
  - deserves its own CocoaHeads talk
  - makes it super-simple to build consistent, async user experiences

Things I Am Currently Investigating

- Using os.log
- Type-safe workflow data "injector"
  - instead of the non-type safe session
  - working on this idea now

Thanks

https://github.com/briancoyner/AsyncWorkflow