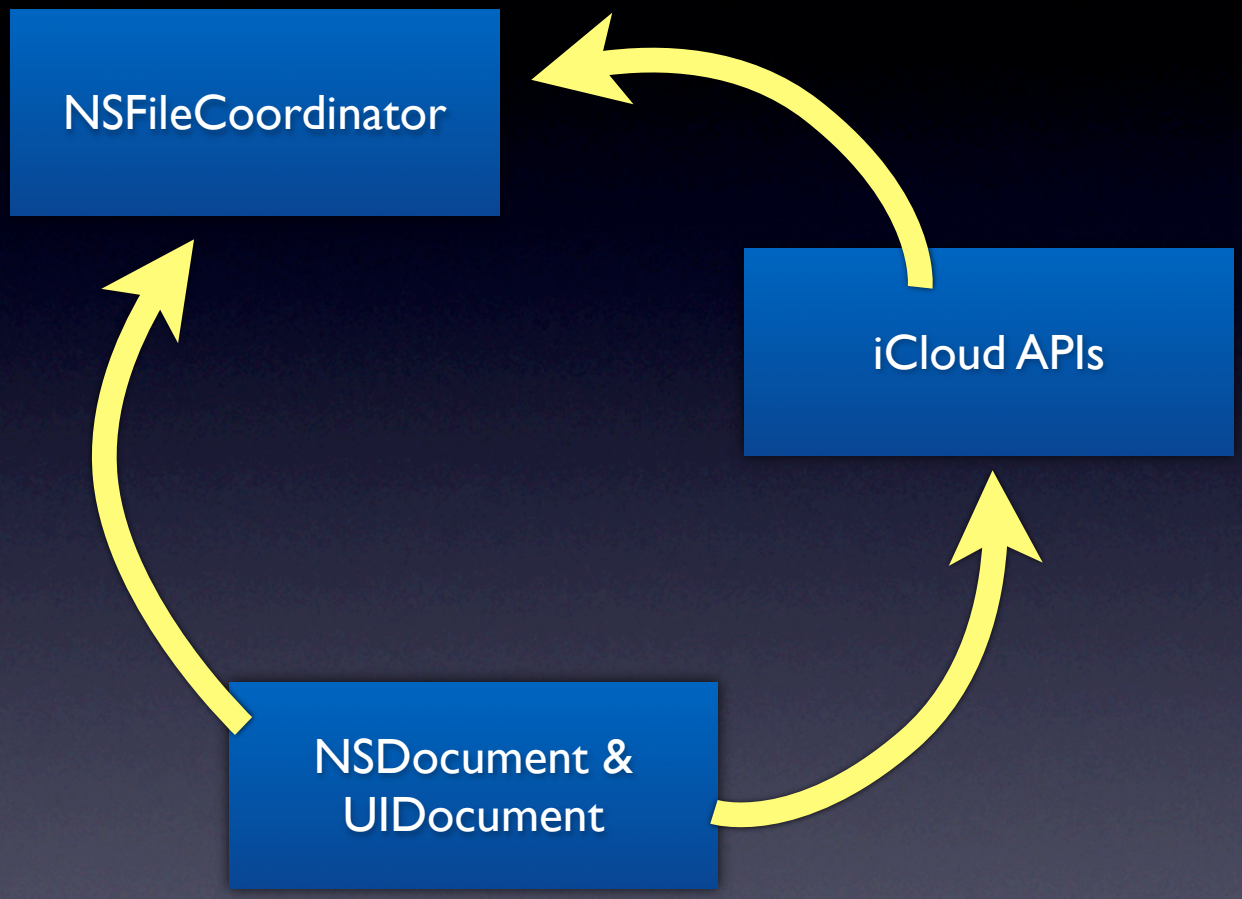
iCloud:
Lessons Learned

Kyle Sluder
The Omni Group


NSFileCoordinator
iCloud APIs
NSDocument &
UIDocument

NSFileCoordinator

Using NSFileCoordinator

- The file system changes constantly
- Not always under your control (esp. on Mac)
  - User can rename files using Finder, edit attachments in Preview…
- You can’t coordinate on entire directories

Using NSFileCoordinator

- Adopt an optimistic locking approach
  - Determine the operation you want to perform
  - Start coordination on relevant URLs
  - If preconditions are no longer met, fail gracefully
  - Otherwise, attempt the operation

File Coordination != Serialization

- File operations might happen in parallel
- Presenter notifications might be enqueued in any order
- Impose your own order on coordinated operations
- Interface with {NS,UI}Document synchronization
- Use a serial `presentedItemOperationQueue`
  - Don't return `[NSOperationQueue mainQueue]`

iCloud APIs

Finding Out About iCloud Docs

- Scan the Ubiquity Container
- NSMetadataQuery
- Treat the filesystem as authoritative
  - NSMetadataQuery might return items that you just deleted
  - NSMetadataQuery returns ancillary information
- Resolving conflicts can wedge NSMetadataQuery

Finding Out About iCloud Docs

- Renamed directories are synced as delete/add
  - You don't get `-presentedItemDidMoveToURL:`
  - Think hard about using document packages
- Be sure to standardize your paths before comparing
  - No other way to keep track of NSFilePresenters

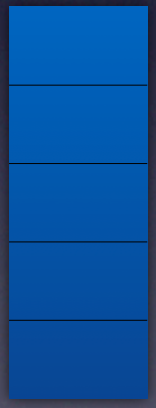
Moving Files To iCloud

- `-[NSFileManager setUbiquitous:itemAtURL:…]`
  - There's nothing special about it… yet
  - Performs a coordinated move to the ubiquity container
  - Must be called on a background thread
- `-[NSFileManager moveItemAtURL:…]`
  - iCloud will eventually notice the file
  - Not guaranteed
  - The only way to provide a file presenter to the NSFileCoordinator
- Use `-setUbiquitous…` unless absolutely necessary

NSDocument & UIDocument

NSDocument Synchronization

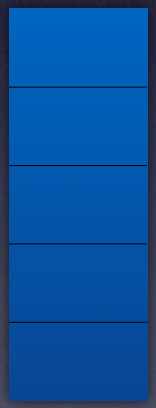
Activities



File Access

NSDocument Synchronization

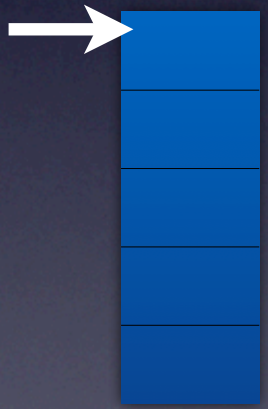
Activities



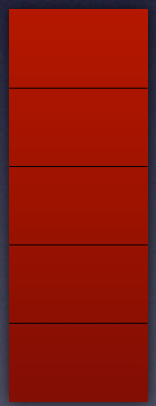
File Access

NSDocument Synchronization

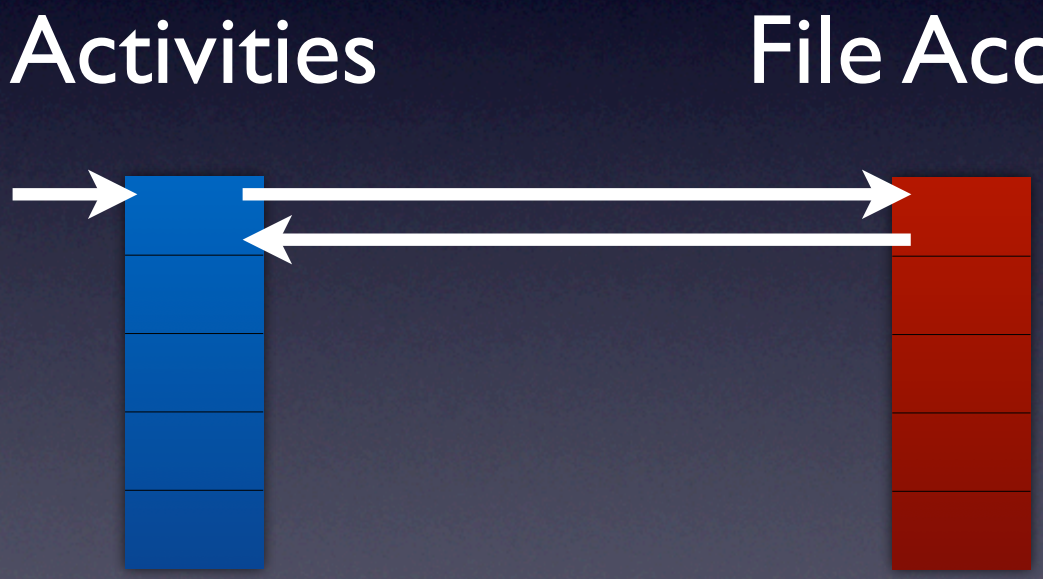
Activities



File Access



NSDocument Synchronization



NSDocument Synchronization

Activities

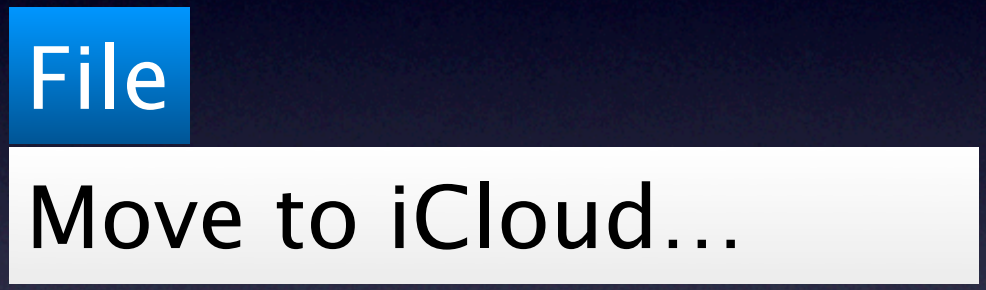
File Access

NSDocument Synchronization

- `-perform(A)SynchronousFileAccess`
- `-performActivityWithSynchronousWaiting:`
- Uses blocks—terms get confusing!
- You need to worry about this even if you don't support concurrent saving

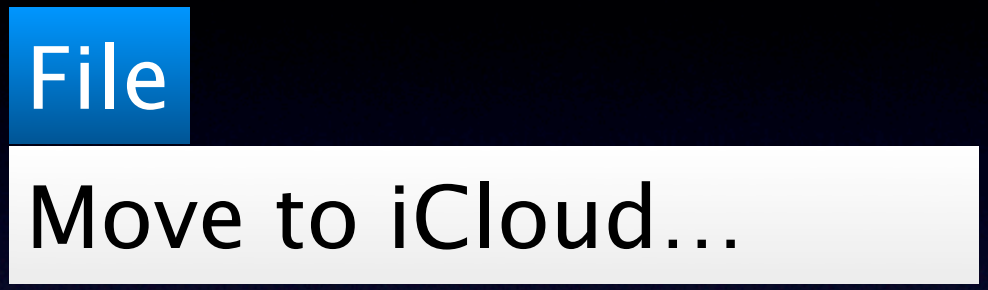
NSDocument Synchronization

- `-performActivity...` is main-thread only
- `-performFileAccess` can be called on any thread
- Block arguments are invoked on calling thread
- For async versions, you hold the "lock" until you invoke the completion handler
- You cannot transfer file access to another thread (r. 10365156)


File
Move to iCloud…

What do we need?

- Single menu item for in/out
  - Need to consult `fileURL`
  - Must be done in file access
  - Can’t block main thread
- Moving to iCloud uses NSFileCoordinator
  - Item must be disabled while move is in flight

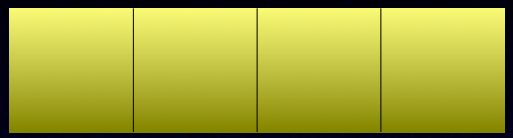


File

Main

File Access

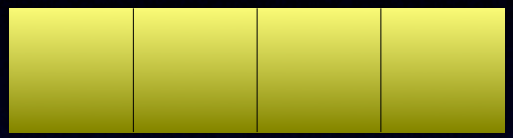
File



Main

File Access

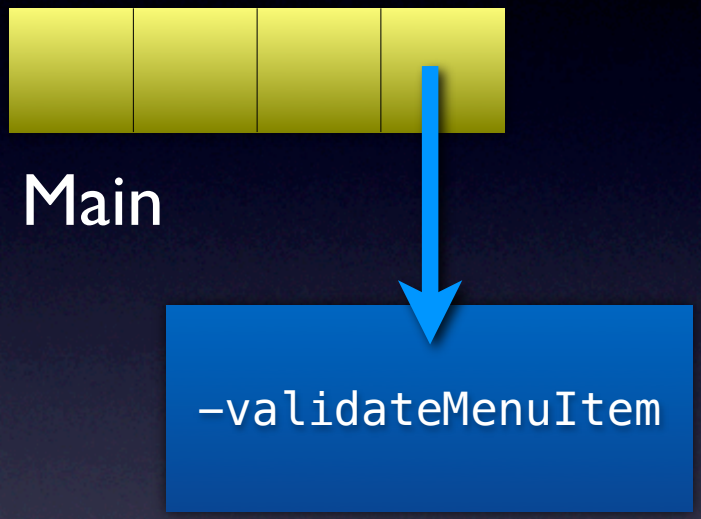
File



Main

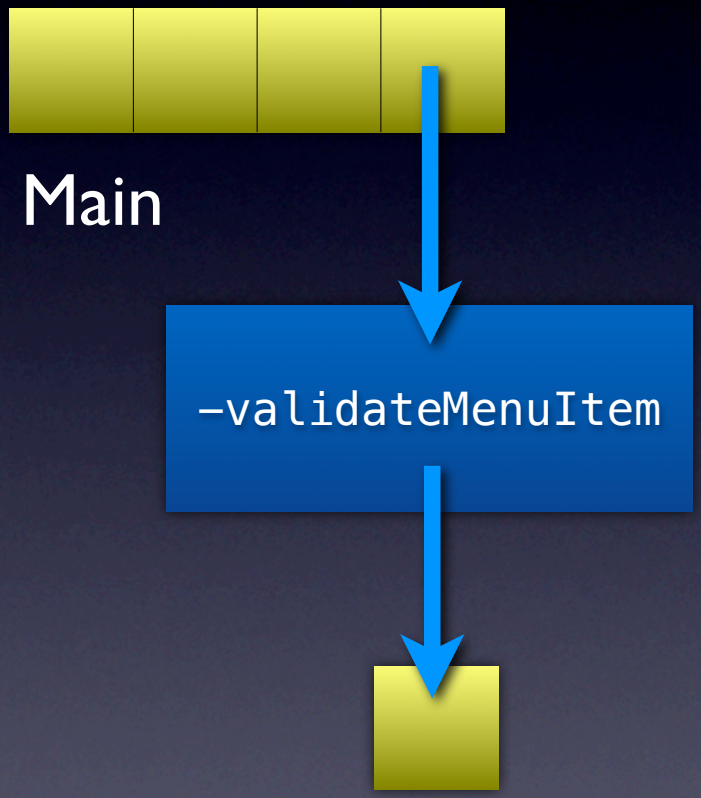
File Access

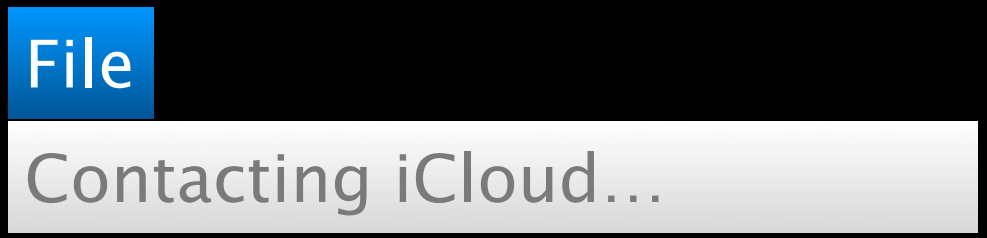
File



File Access

File





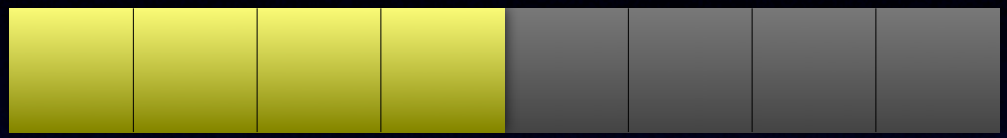
Main

`-validateMenuItem`

File Access

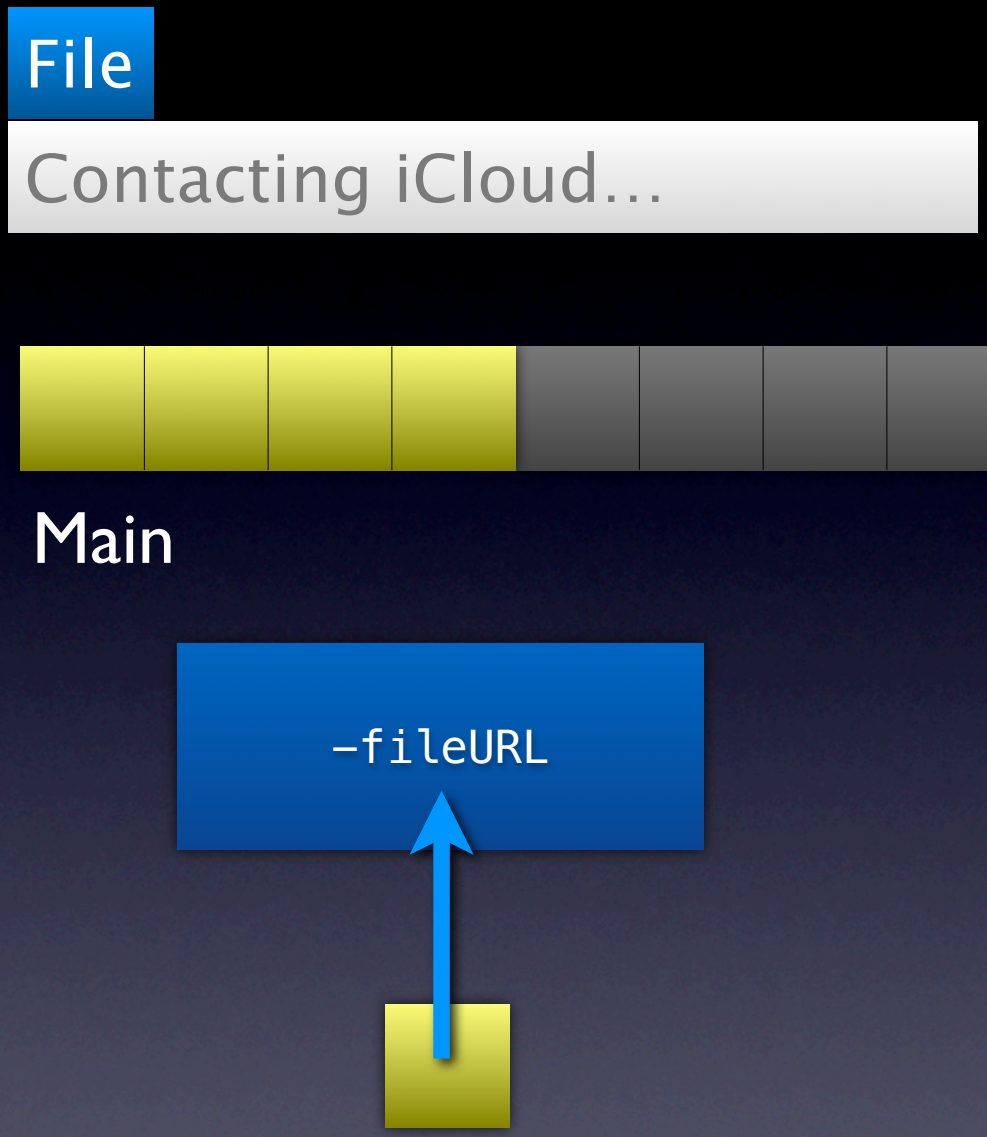
File

Contacting iCloud…



Main

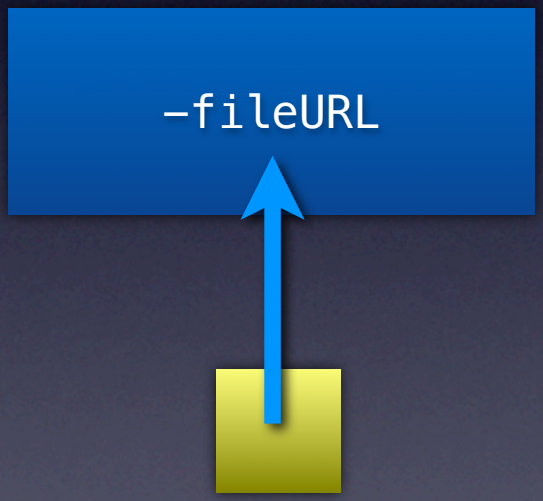
File Access


File
Contacting iCloud…
Main
-fileURL
File Access

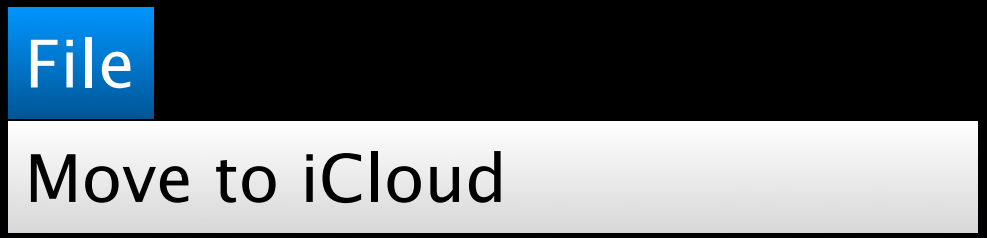
File

Move to iCloud

Main

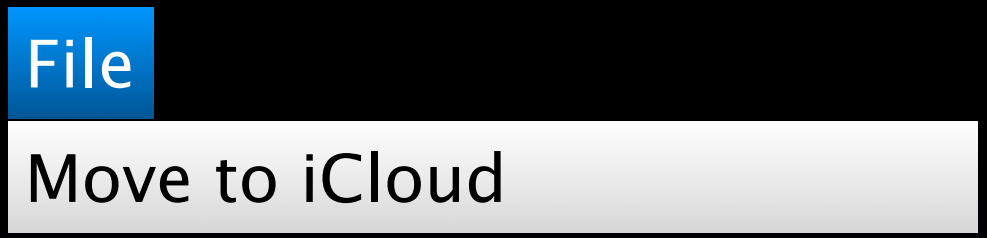


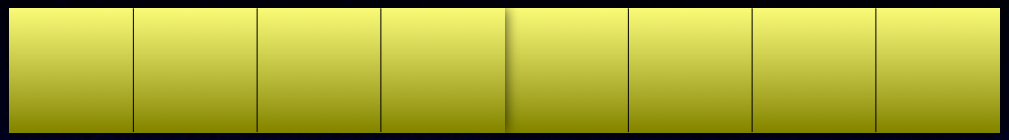
File Access



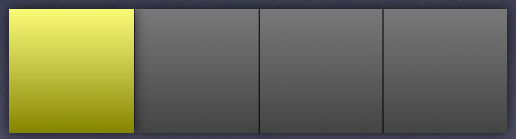
Main

File Access





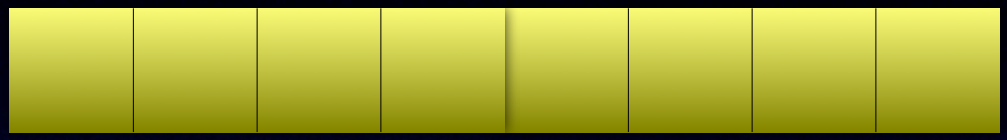
Main



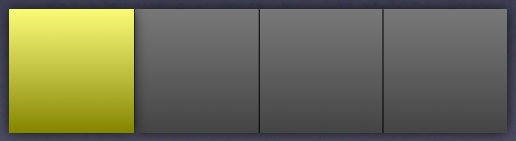
File Access

File

Move to iCloud



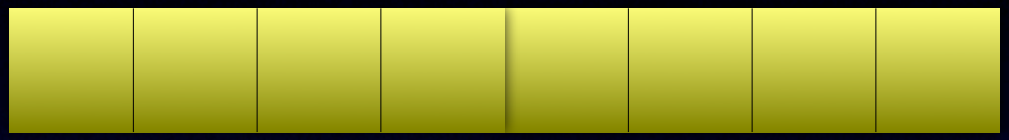
Main



File Access

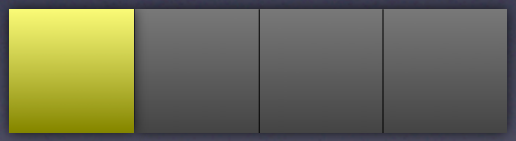
File

Move to iCloud



Main

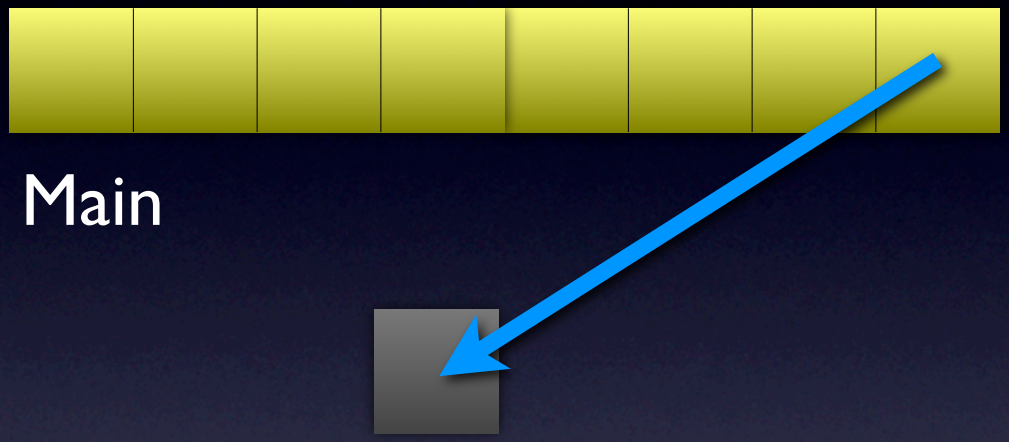
Activity

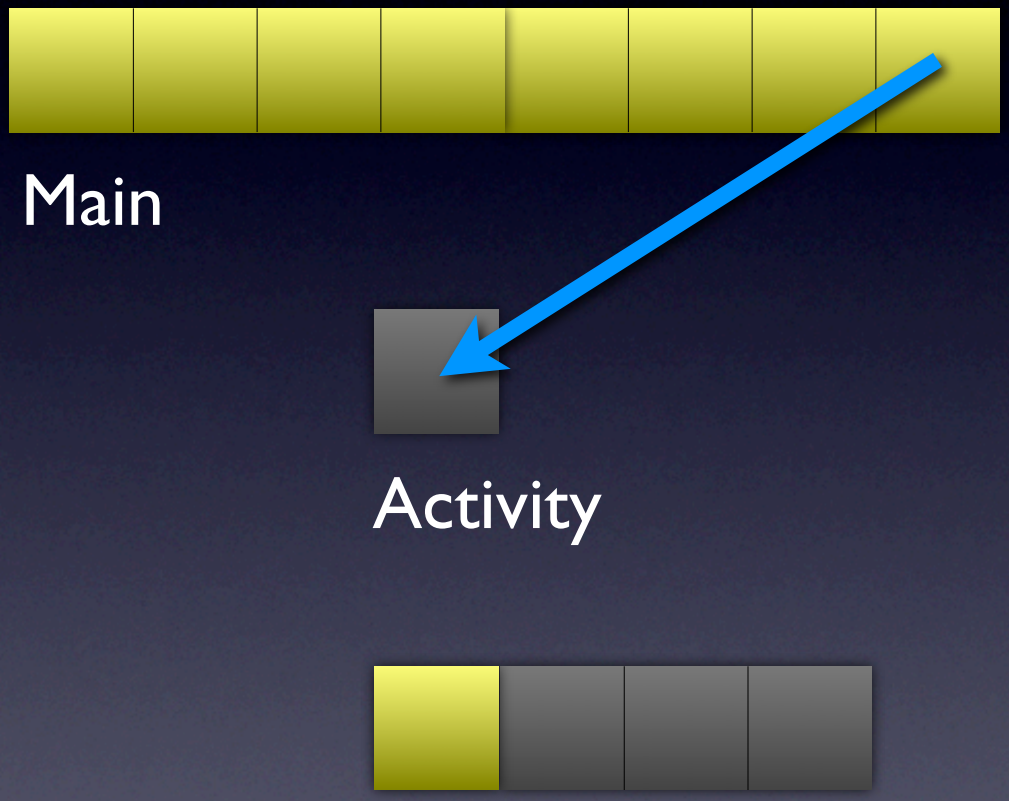


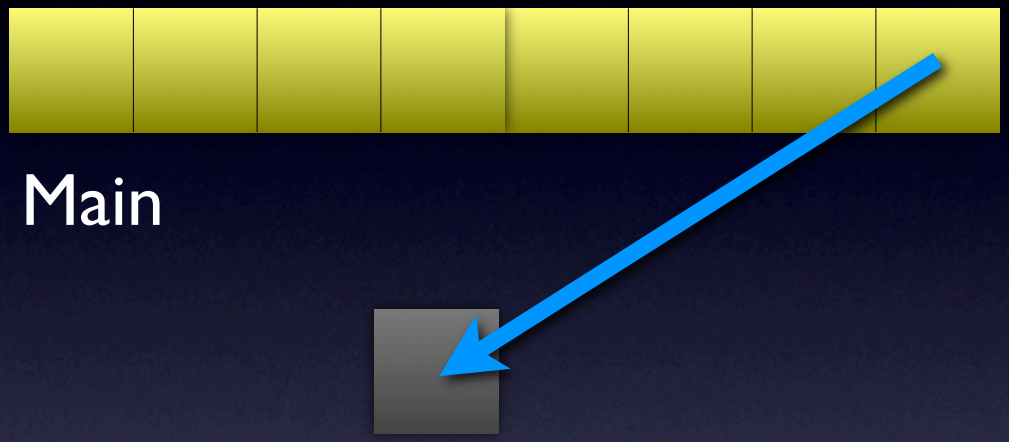
File Access

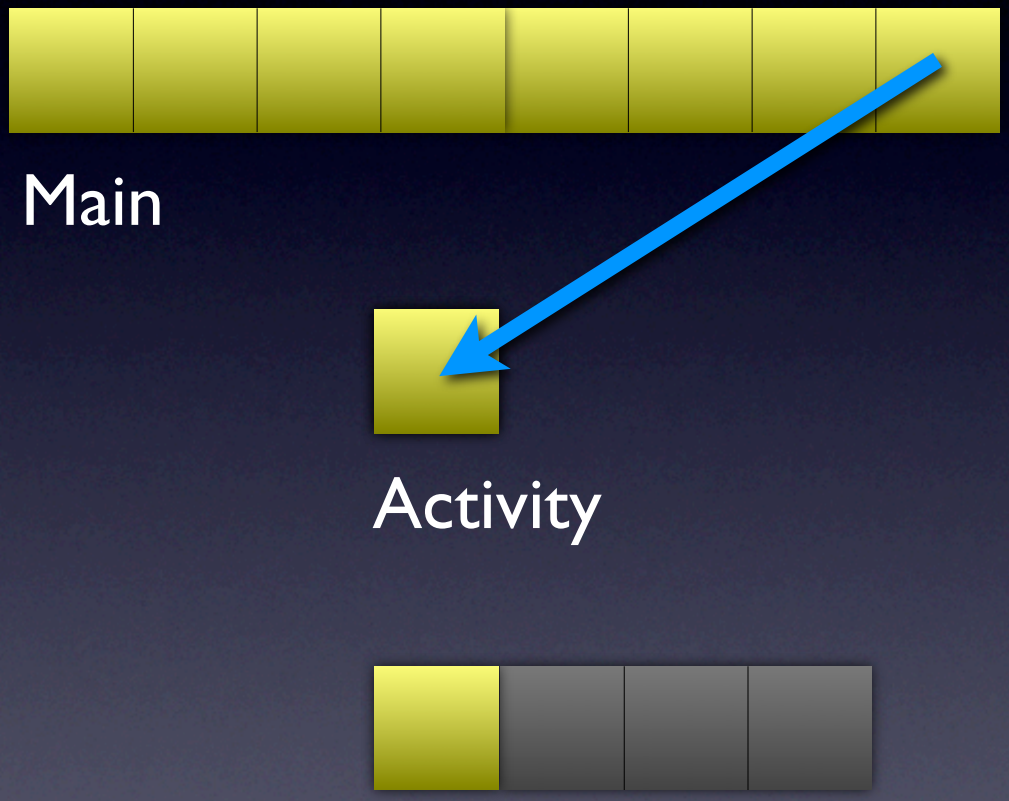
File

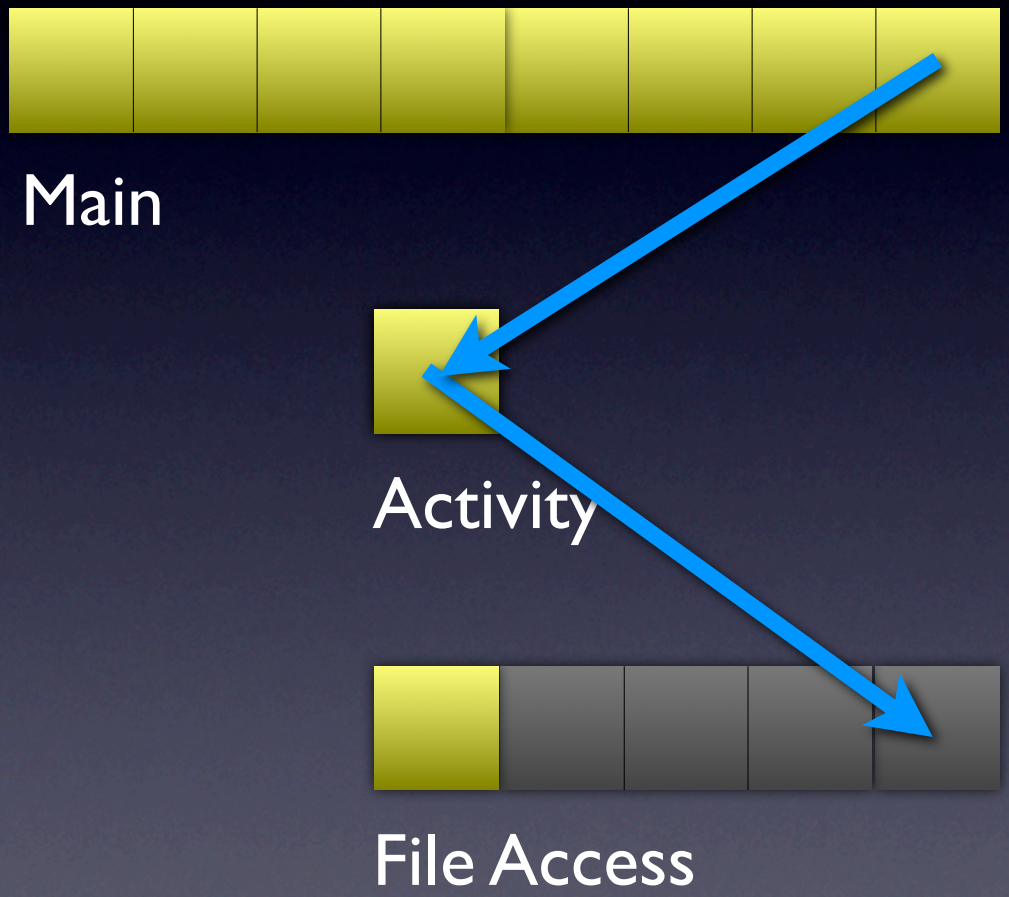
Move to iCloud

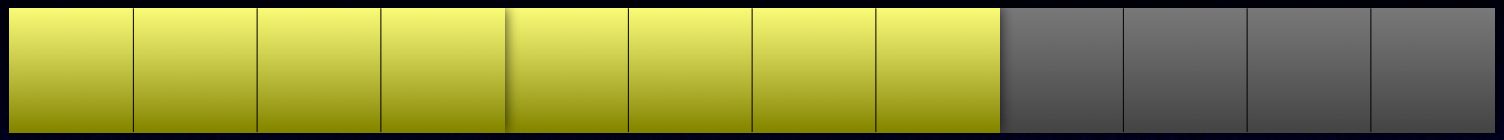



Main
Activity
File Access

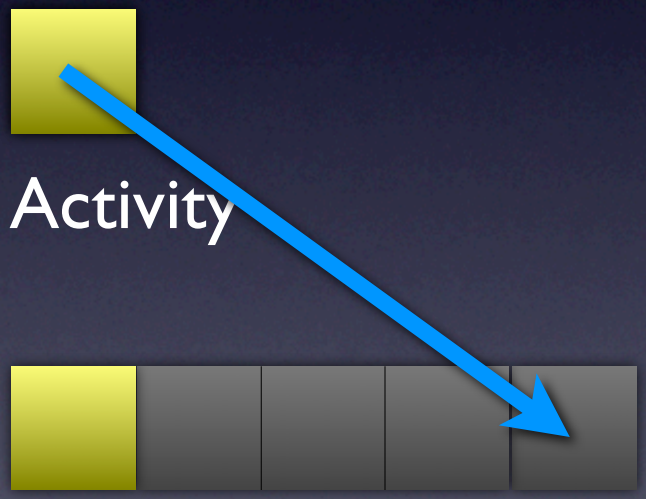

Main
Activity
File Access


Main
Activity
File Access


Main
Activity
File Access

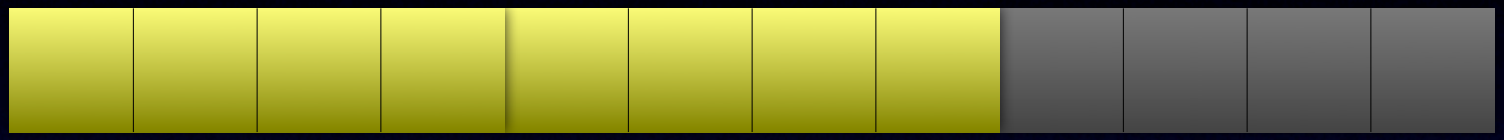


Main



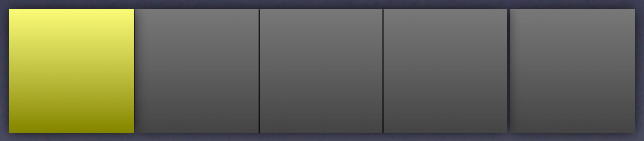
Activity

File Access

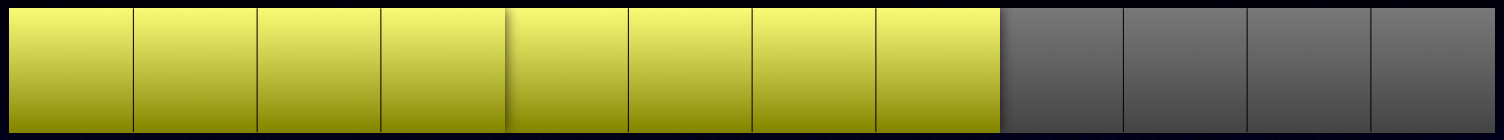


Main

Activity



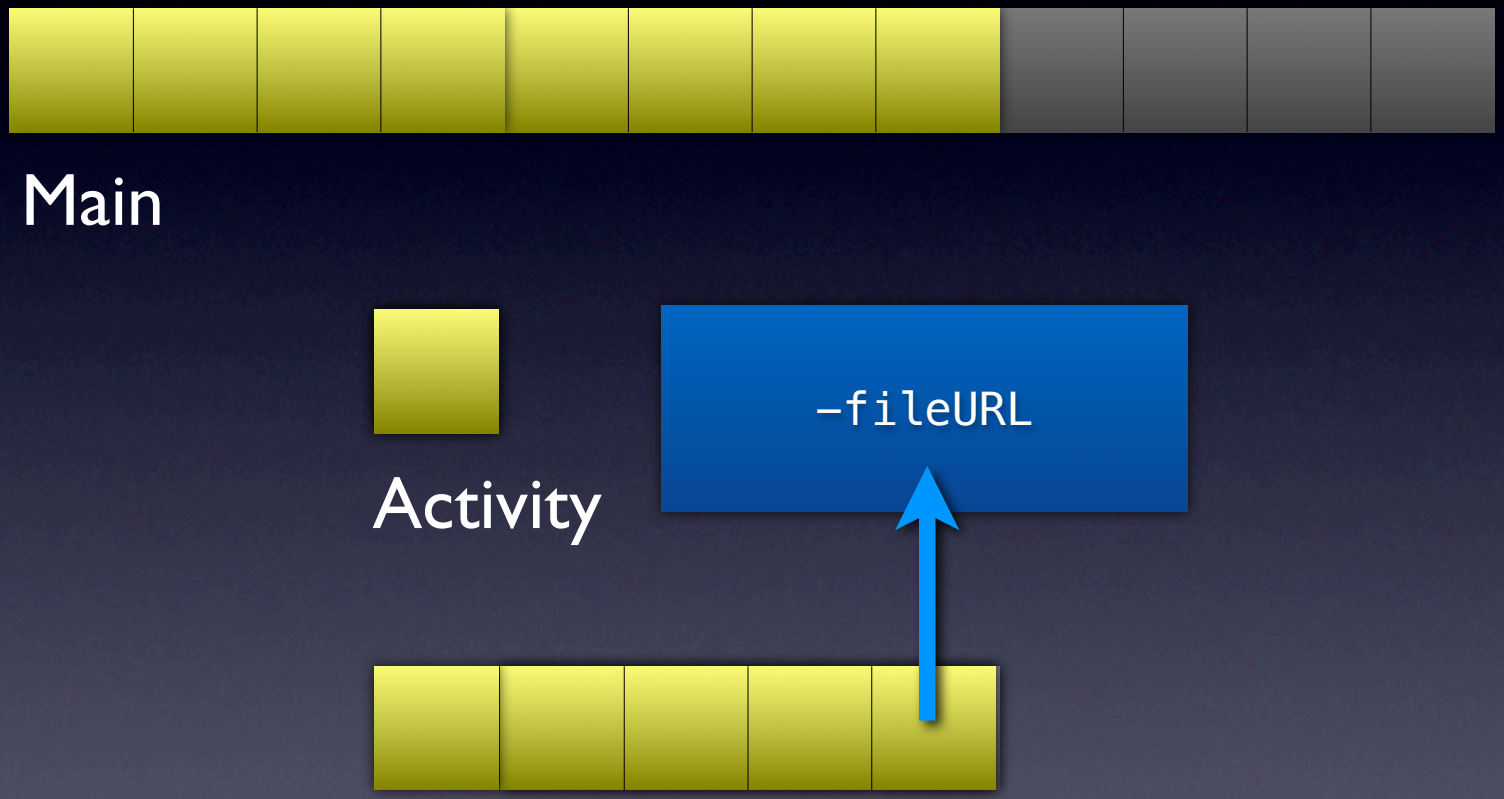
File Access

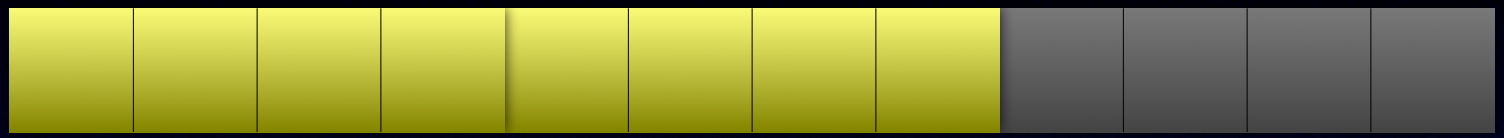


Main

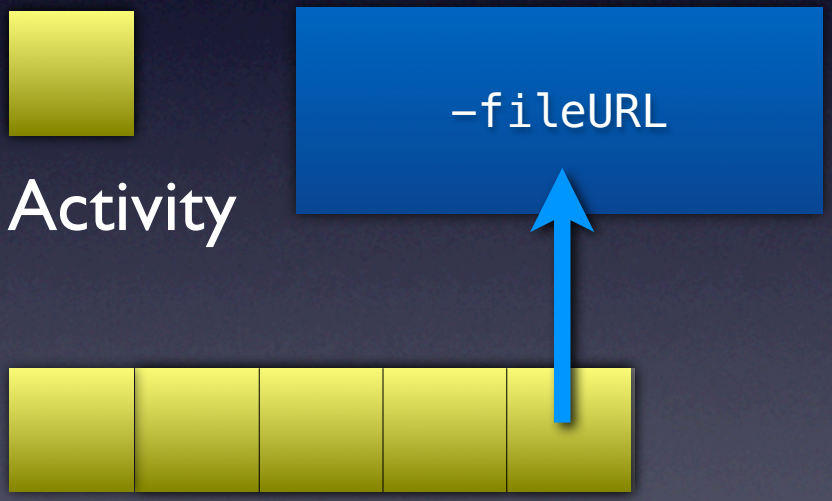
Activity

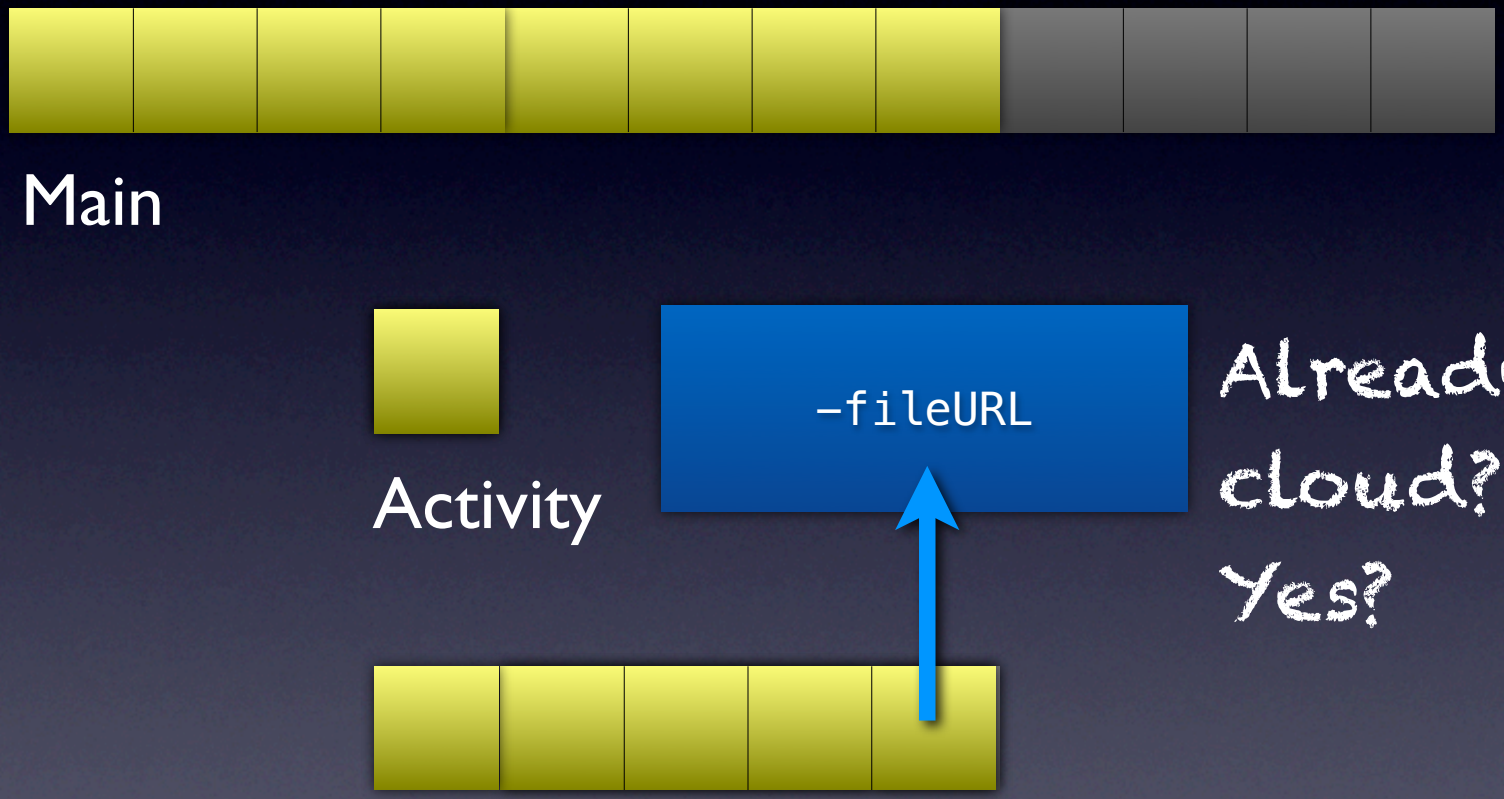
File Access

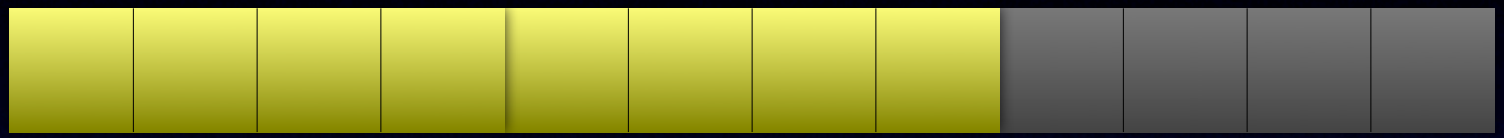

Main
Activity
-fileURL
File Access



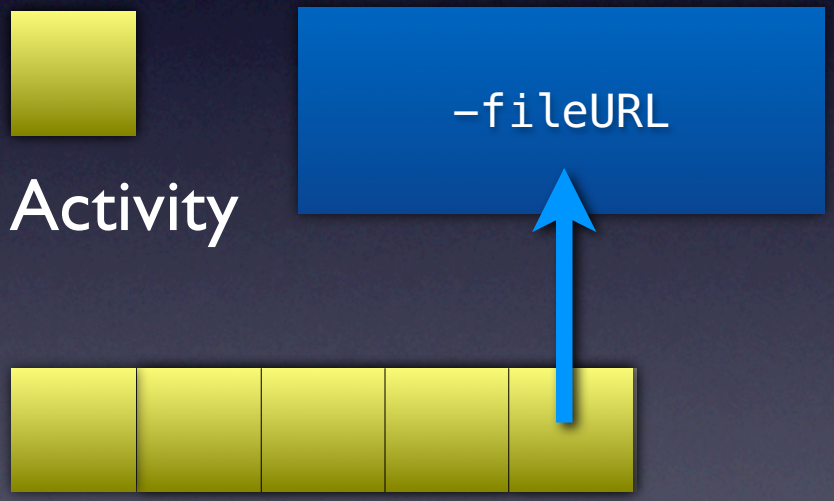
Main




Main
Activity
-fileURL
Already in cloud?
Yes?
File Access



Main



Activity

File Access

Already in cloud?

Yes?

Fail silently

A Race Condition Appears!

- Want to perform move to iCloud within file access
- NSDocument's file presenter methods use –`performFileAccess` and –`performActivity`
  - Which one(s) they use isn't documented

Main

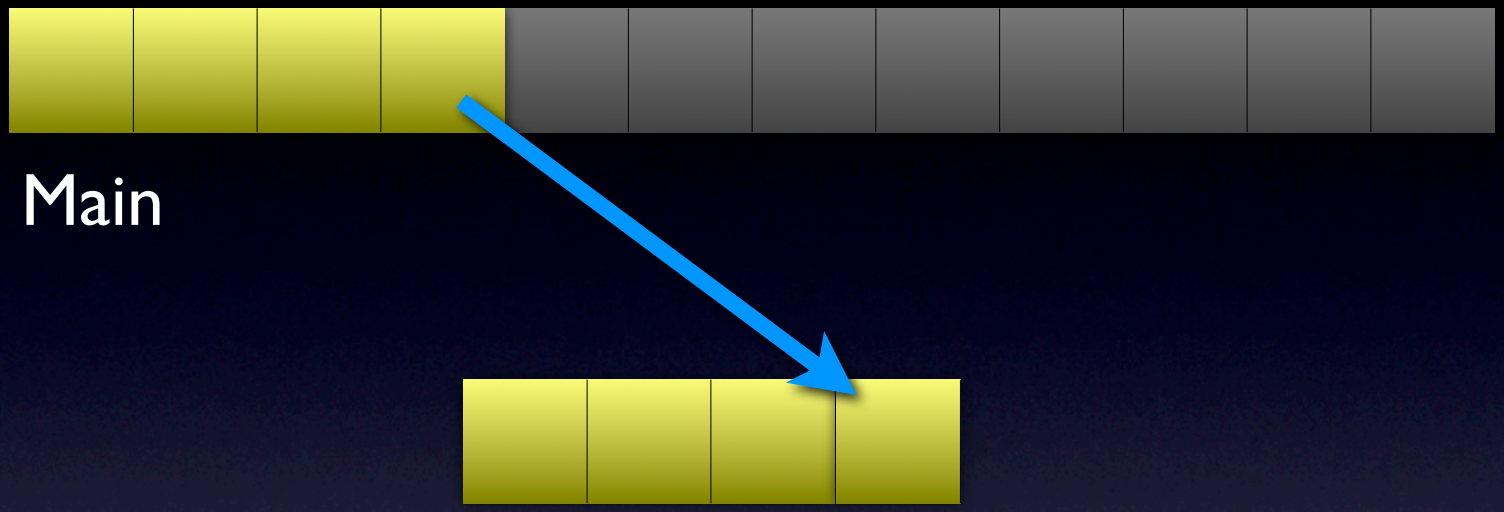
File Access

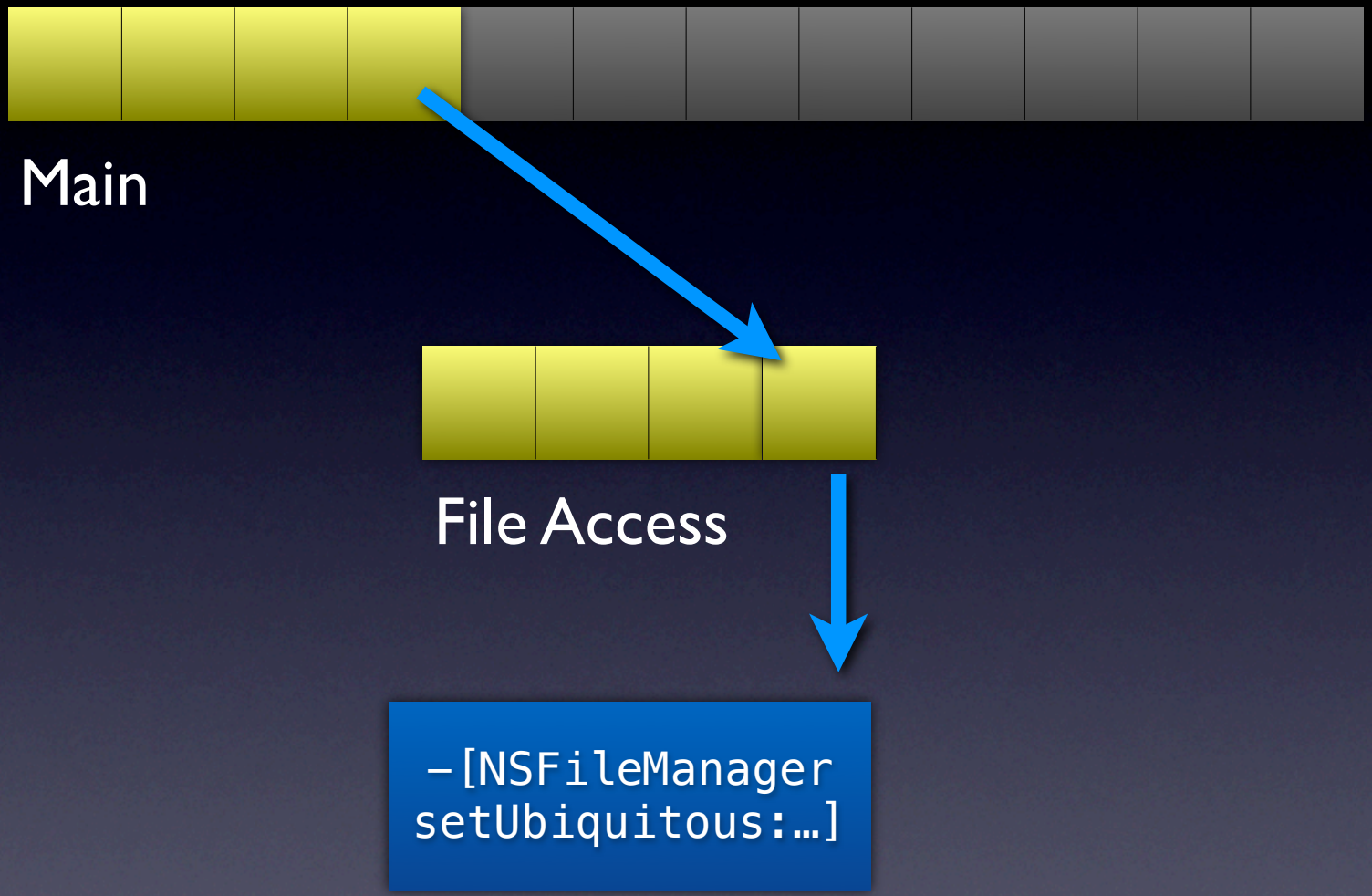
Main

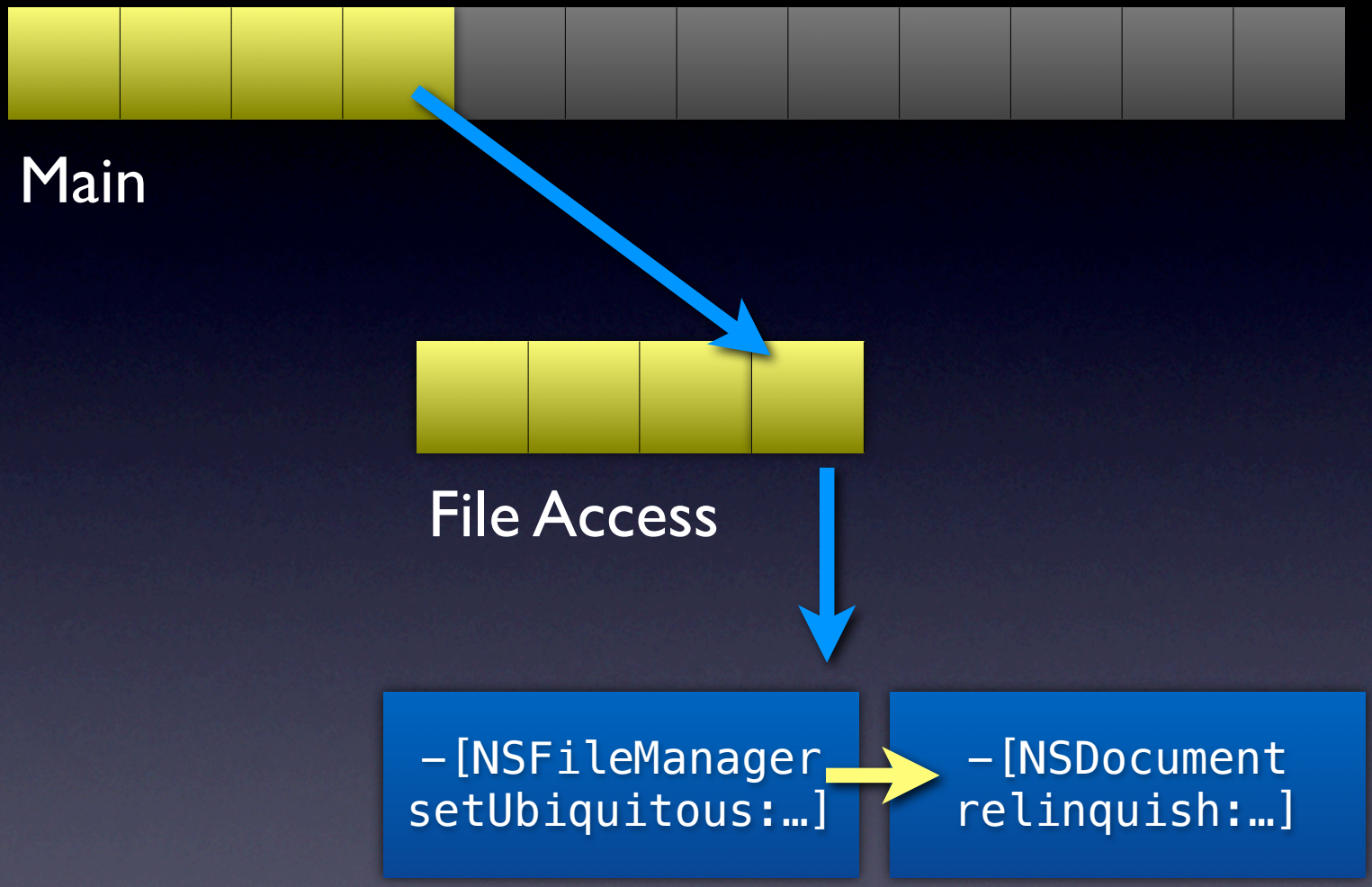
File Access

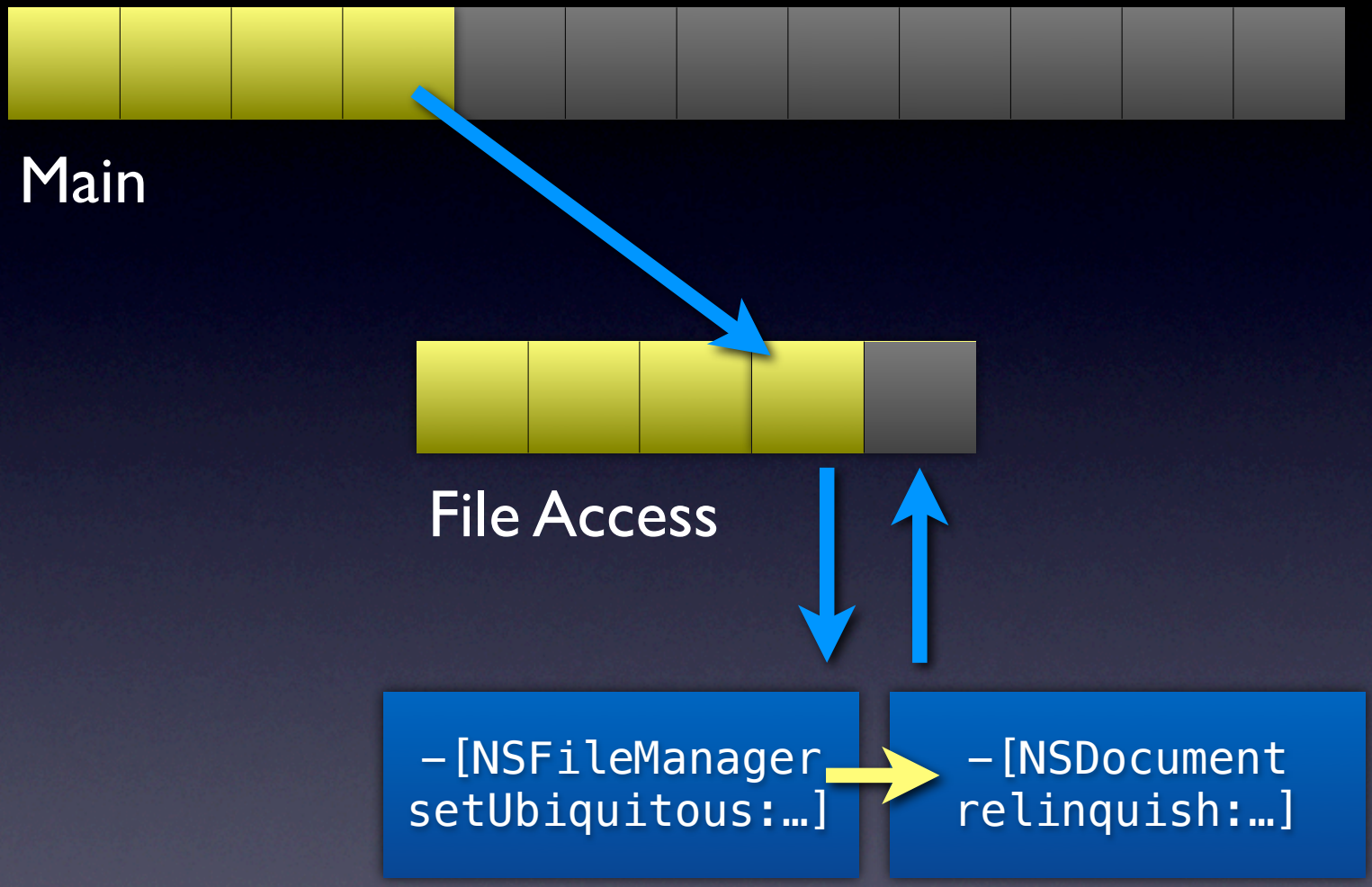
Main

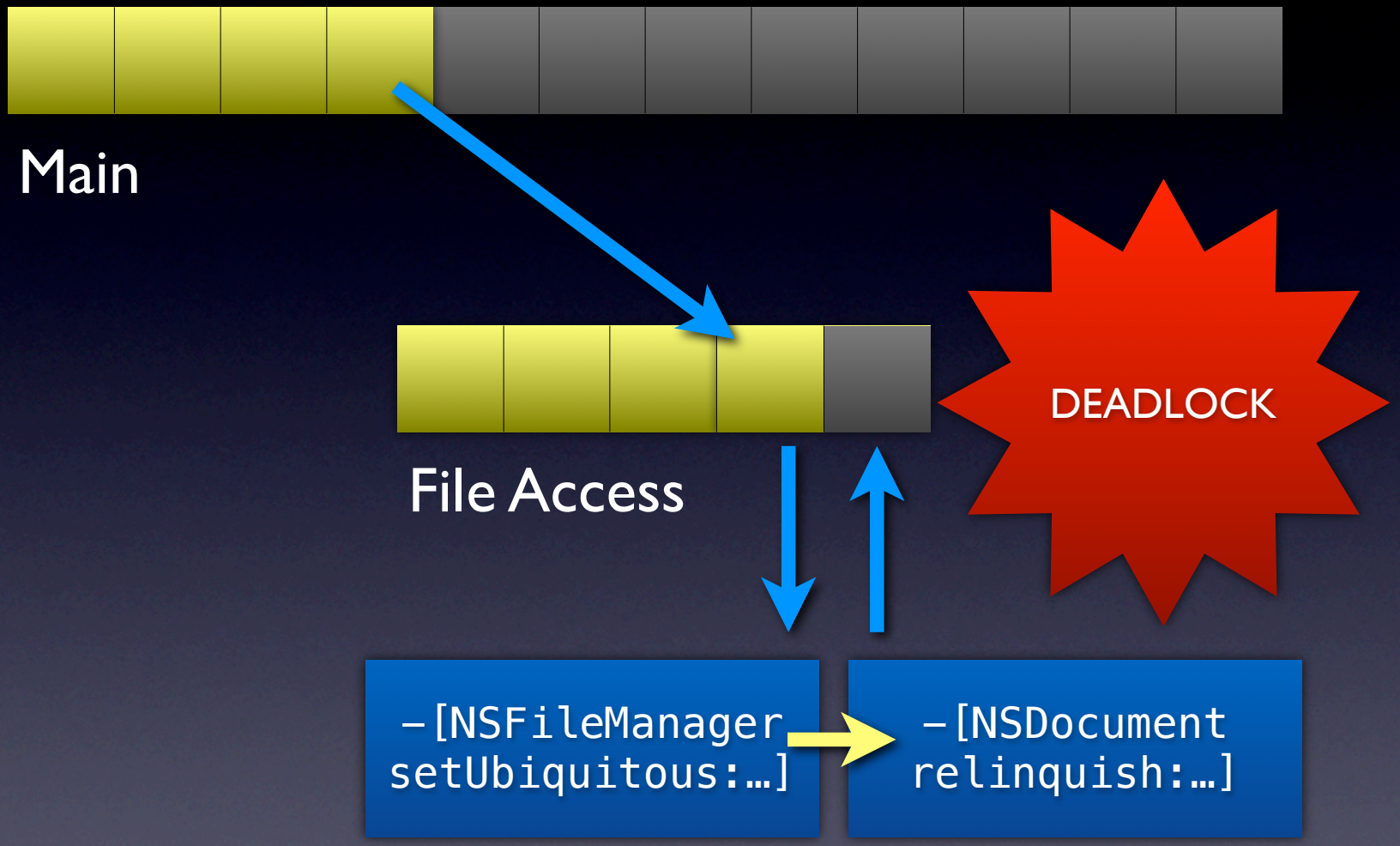
File Access


Main
File Access


Main
File Access
-[NSFileManager
setUbiquitous:…]


Main
File Access
-[NSFileManager setUbiquitous:…]
-[NSDocument relinquish:…]


Main
File Access
-[NSFileManager setUbiquitous:…]
-[NSDocument relinquish:…]


Main
File Access
DEADLOCK
-[NSFileManager setUbiquitous:…]
-[NSDocument relinquish:…]

Main

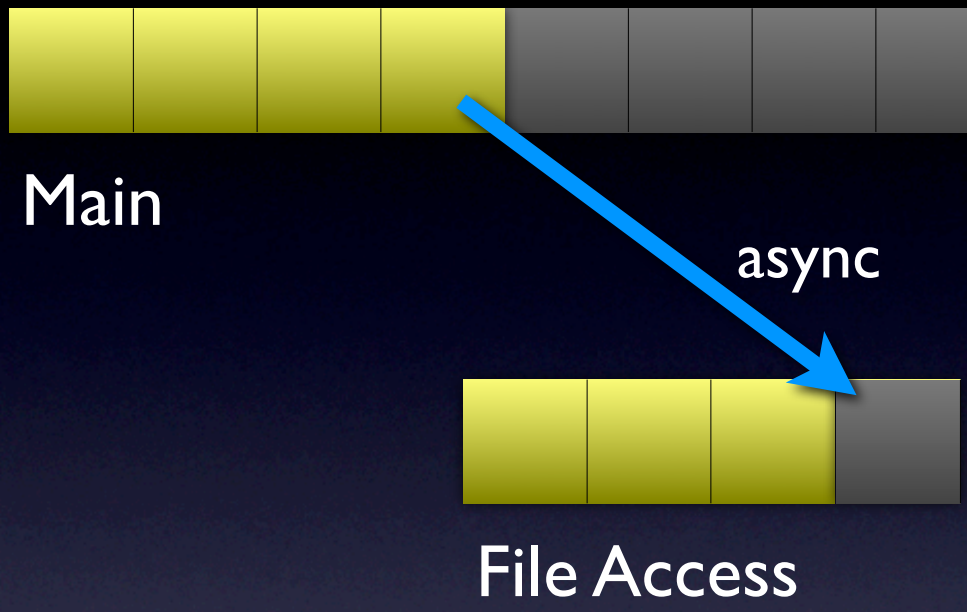
File Access

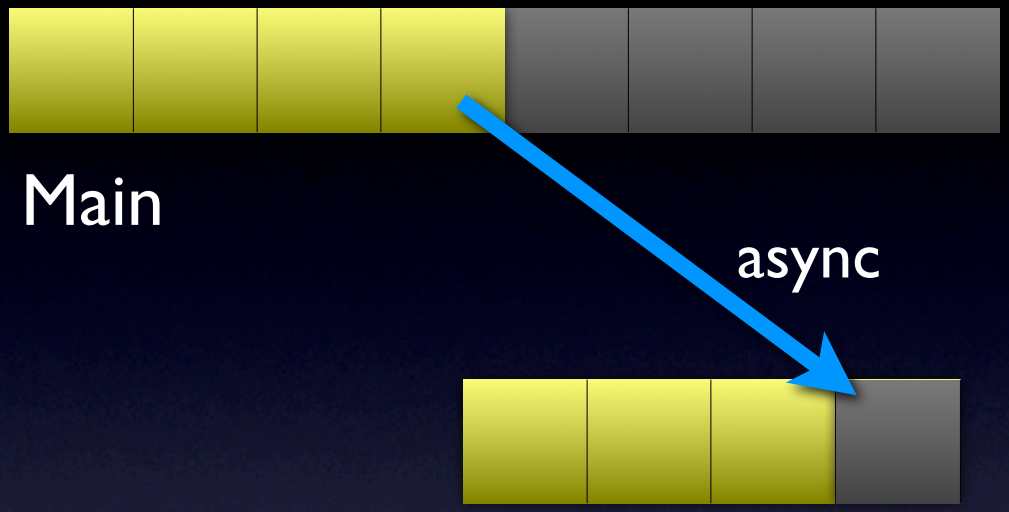
Main

File Access

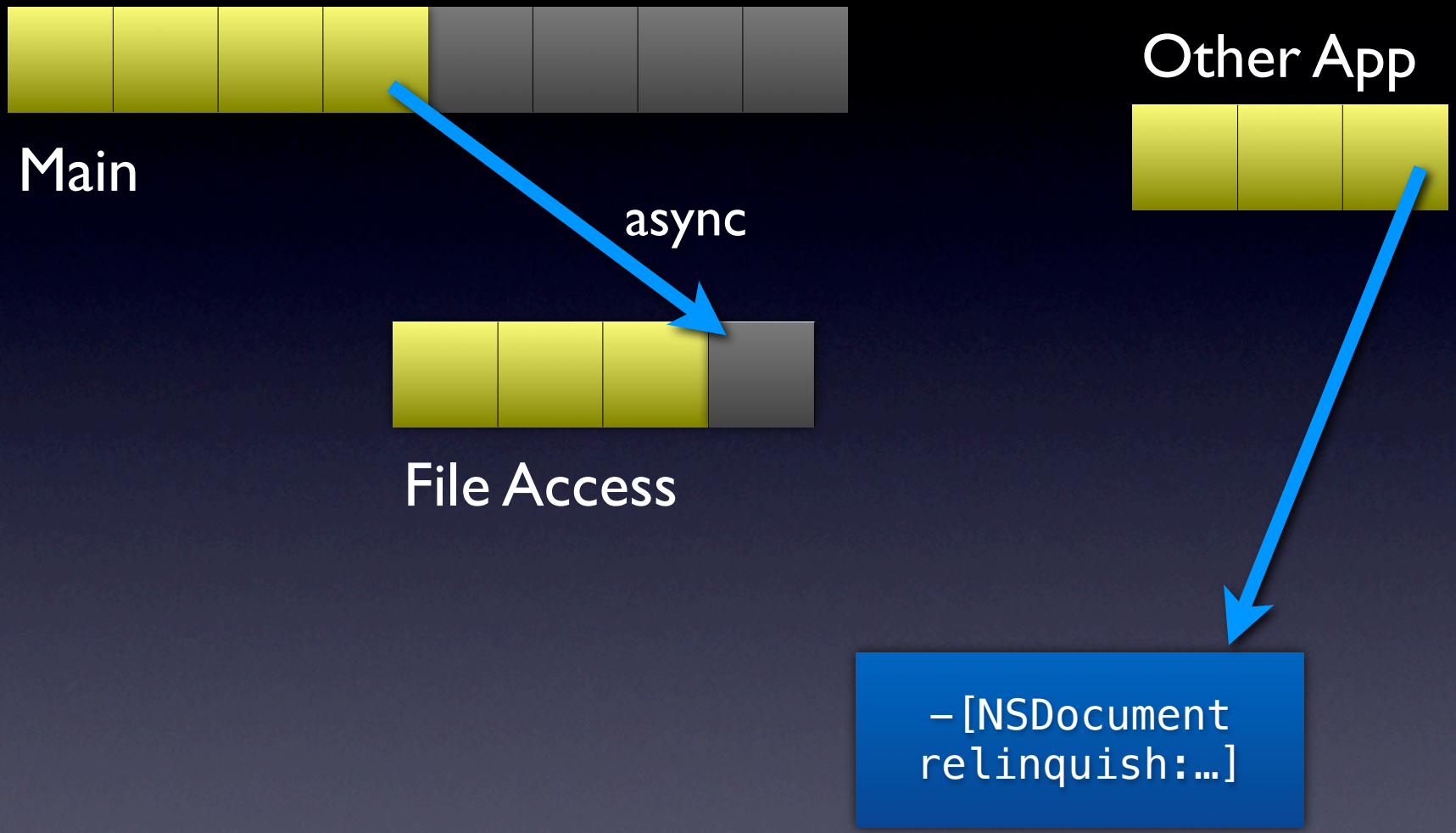
Main

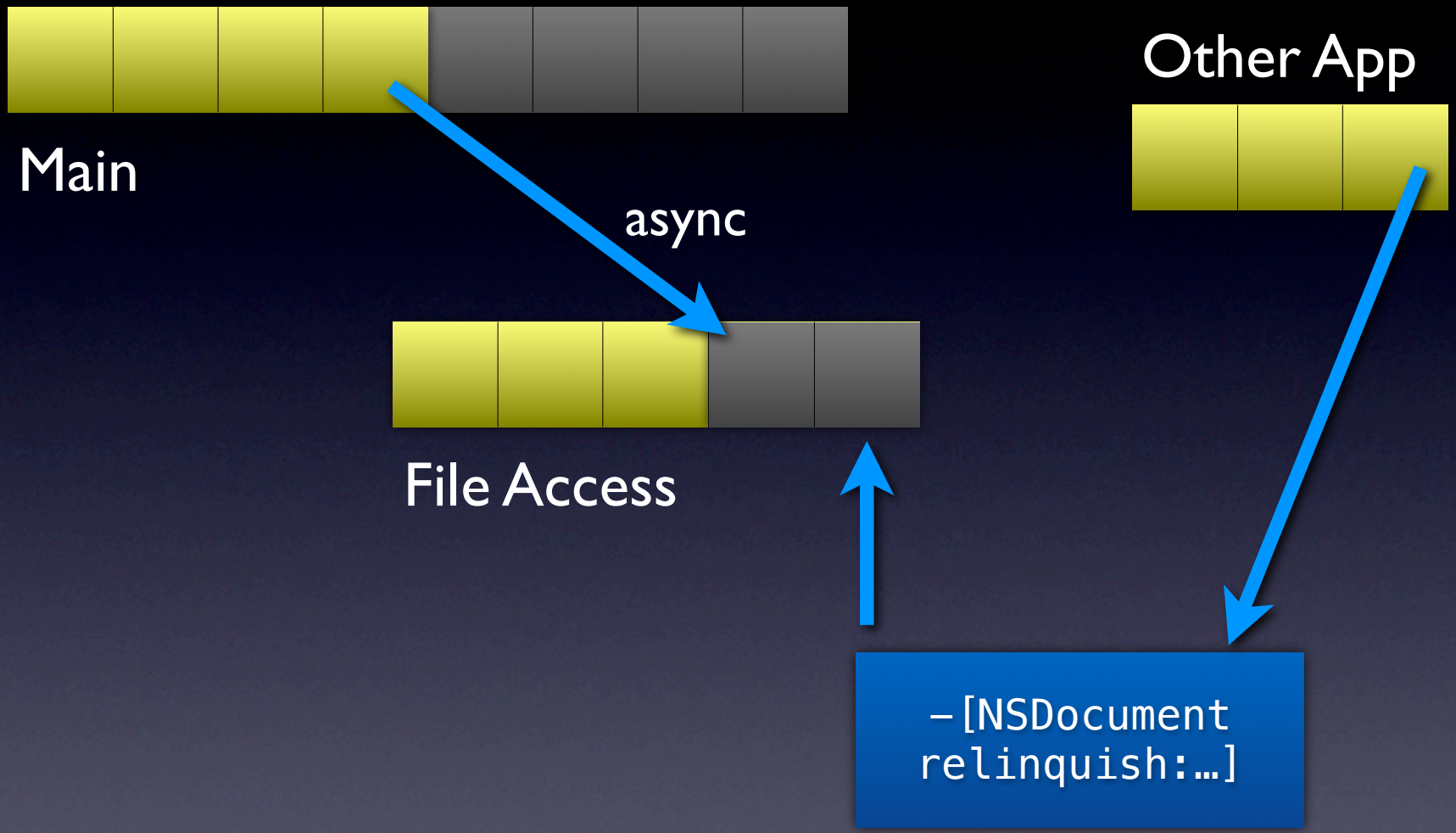
File Access

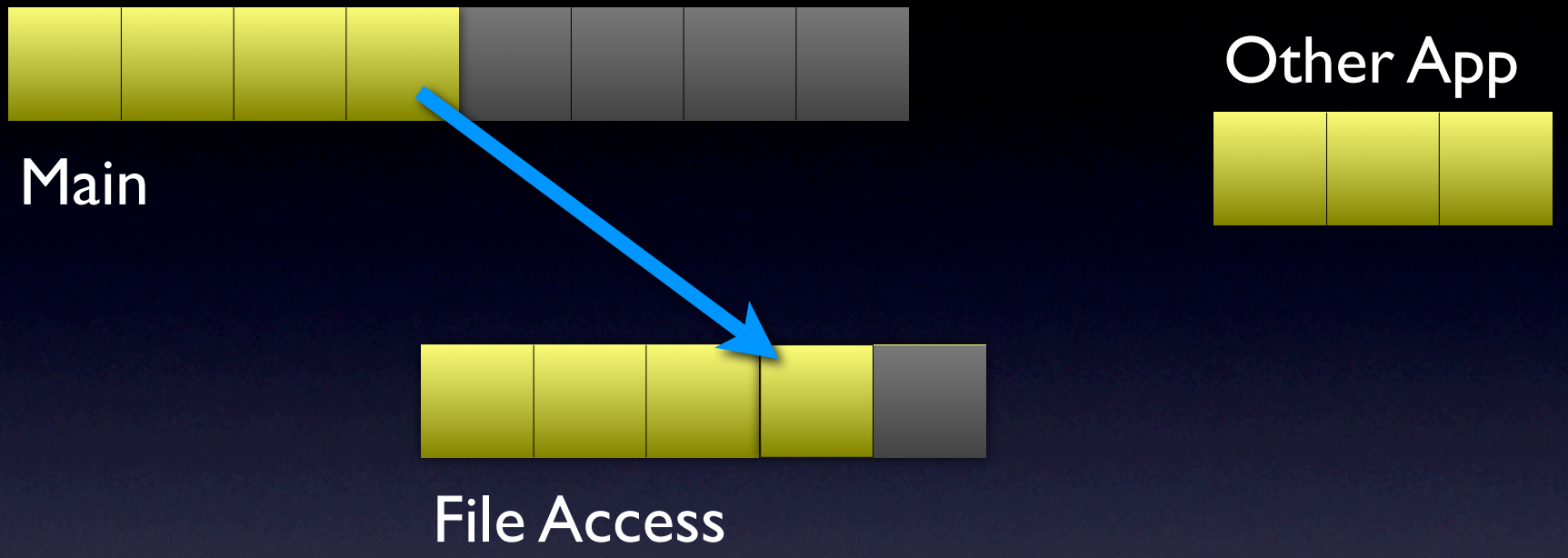

Main
async
File Access

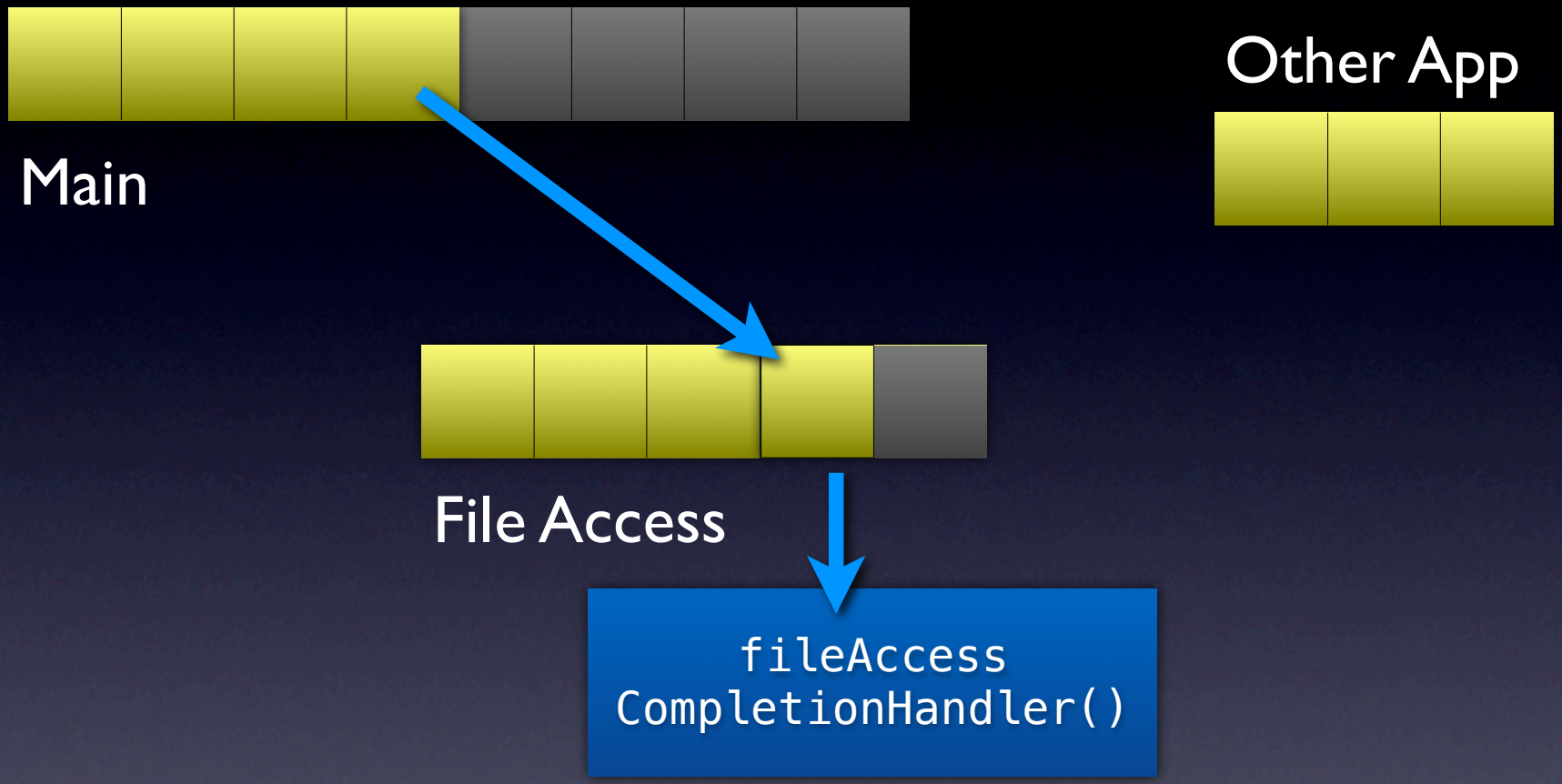


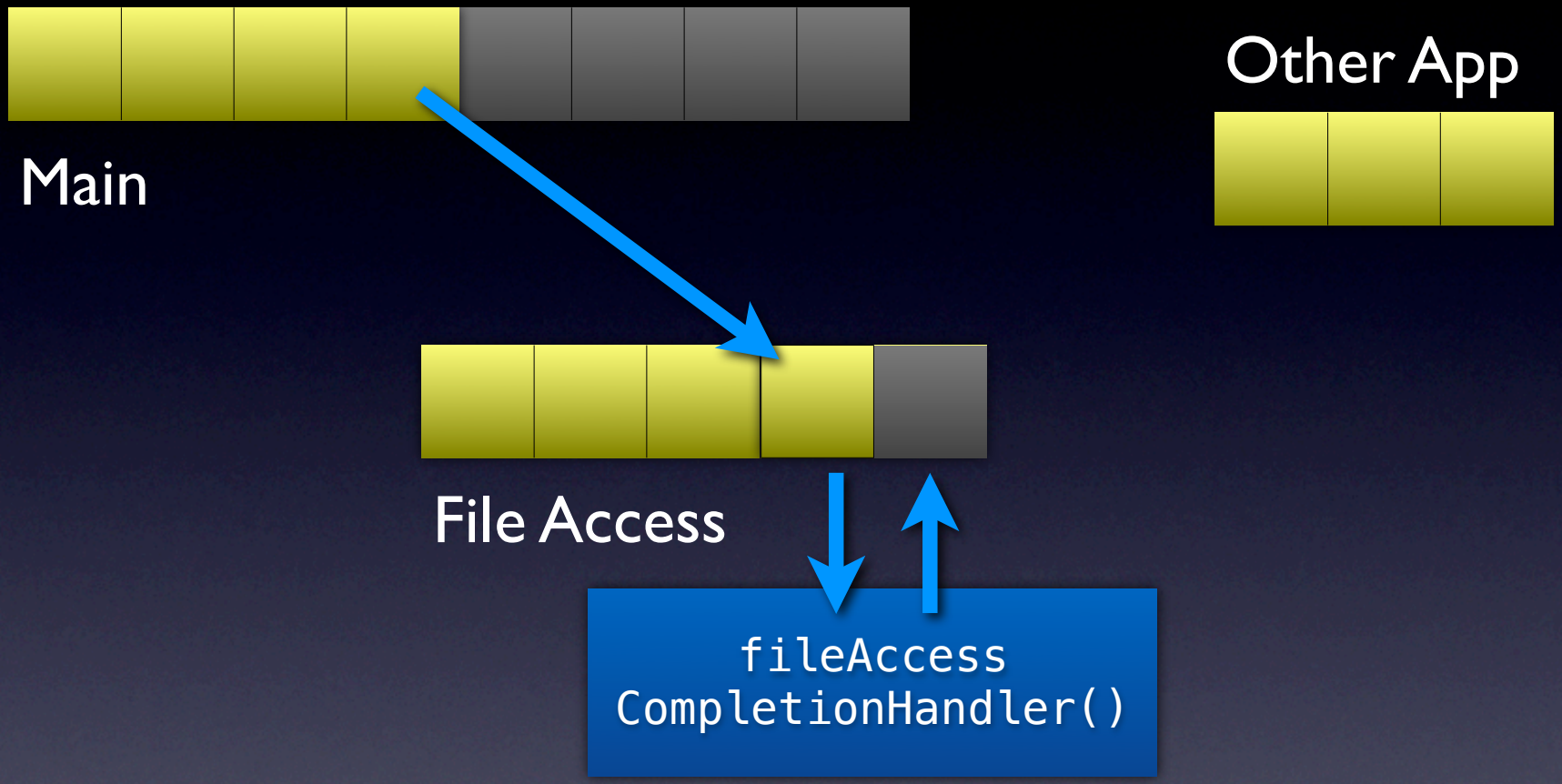

```
-[NSDocument
relinquish:…]
```

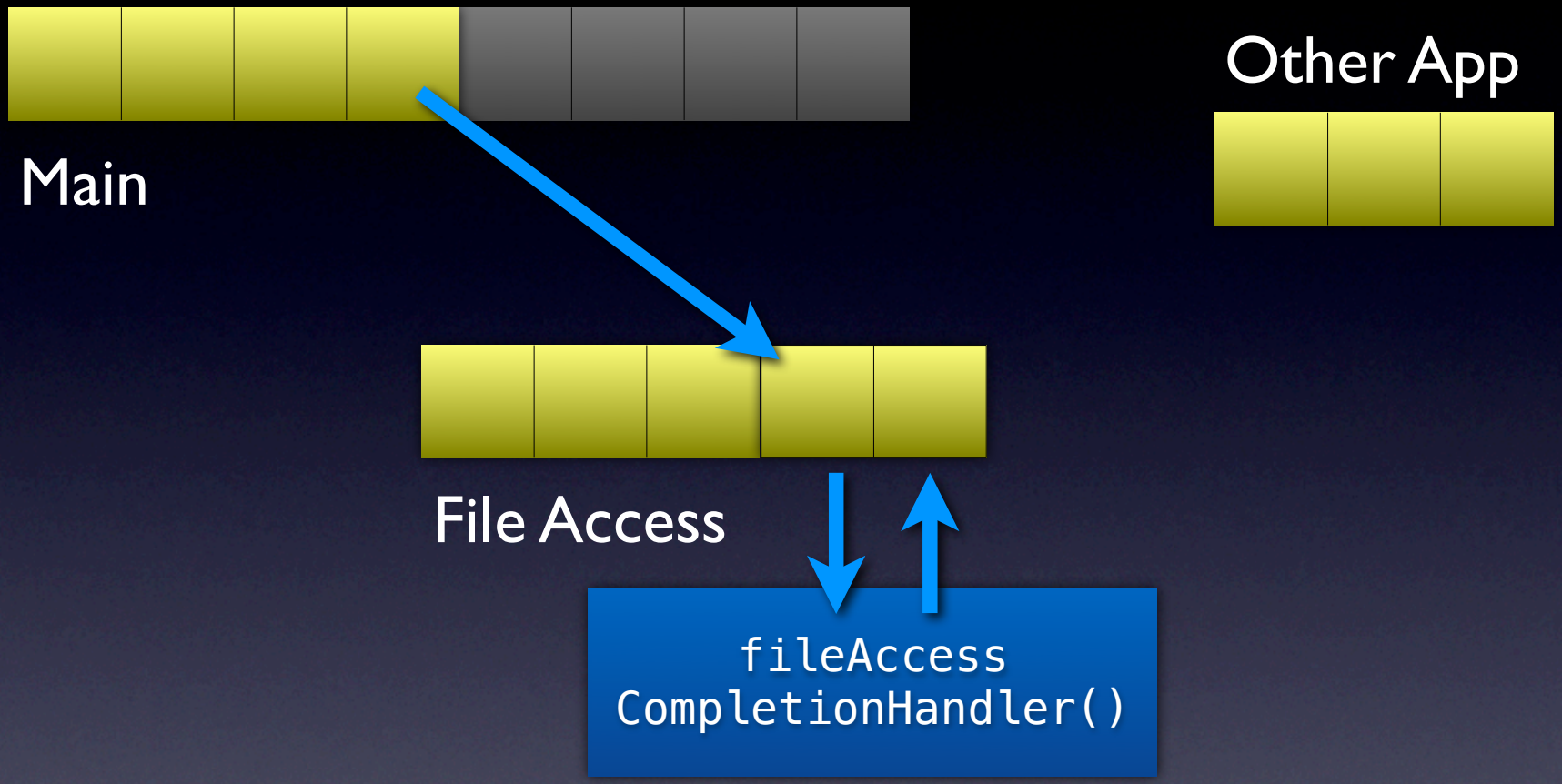

Main
async
File Access
Other App
-[NSDocument
relinquish:…]

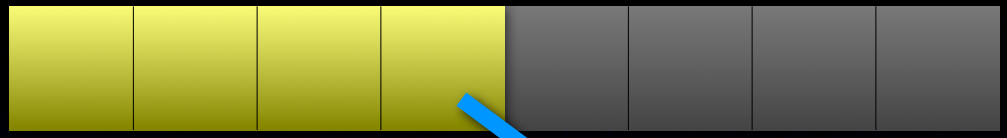

Main
async
File Access
Other App
-[NSDocument relinquish:…]


Other App
Main
File Access


Main
Other App
File Access
fileAccess
CompletionHandler()

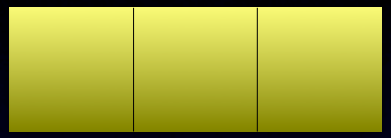

Other App
Main
File Access
fileAccess
CompletionHandler()


Other App
Main
File Access
fileAccess
CompletionHandler()



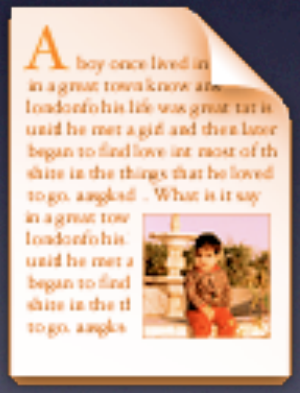
Main

Other App

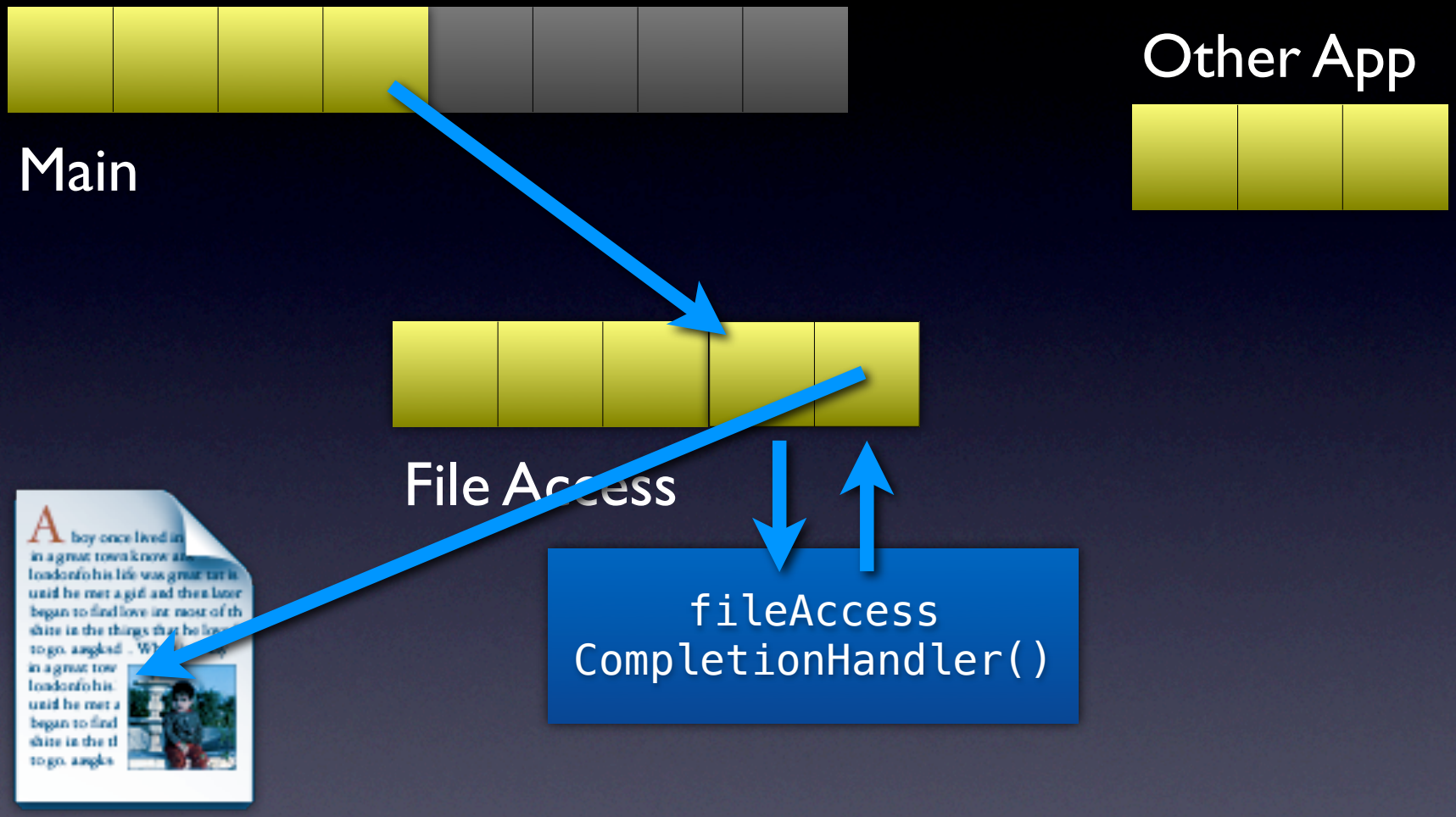


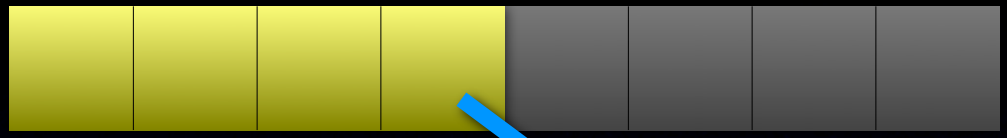
File Access

```
fileAccess
CompletionHandler()
```



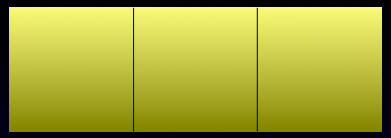
My Document


Other App
Main
File Access
fileAccess
CompletionHandler()
My Document



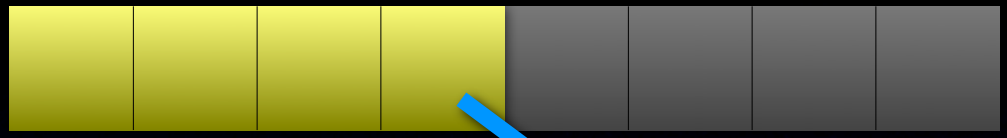
Main

Other App



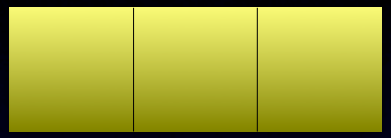
File Access

My Document



Main

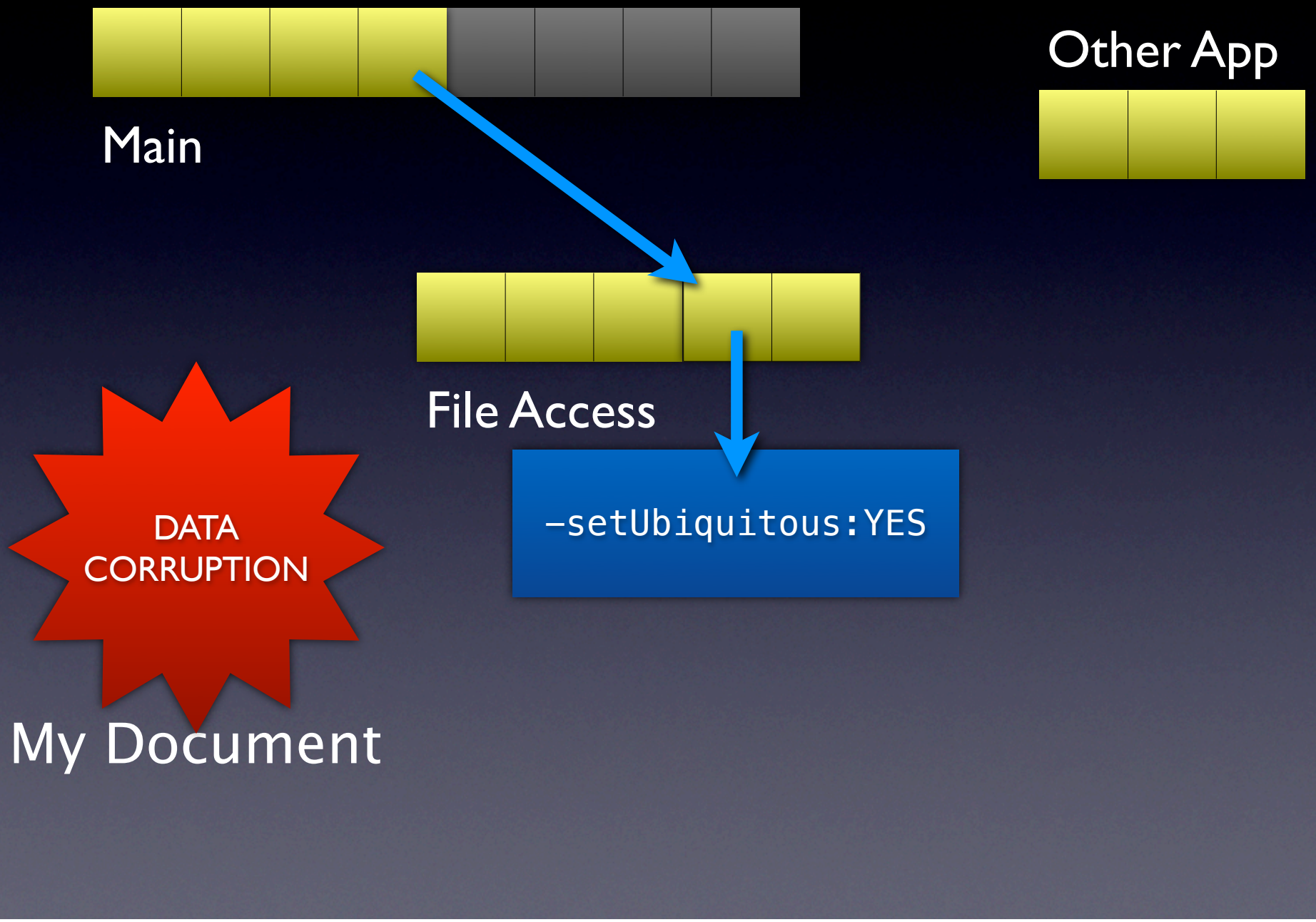
Other App



File Access

```
-setUbiquitous:YES
```

My Document


Main
Other App
File Access
-setUbiquitous:YES
DATA
CORRUPTION
My Document

UIDocument: It's Different!

- `-performAsynchronousFileAccess:` block is *always executed on a background thread*
- No concept of activities; use `-setUserInteractionEnabled:` instead
- No synchronous version of file access API

UIDocument: It's Broken!

- iTunes doesn't use NSFileCoordinator when accessing ~/Documents (r.10260542)
- `-accommodatePresentedItemDeletion` is broken

Handling UIDocument Deletion

- You get `-presentedItemDidMoveToURL:` with a URL in the (undocumented) "dead zone"
- Remember this URL, return it from `-presentedItemURL`, but *don't* call super
- You get `-accommodatePresentedItemDeletion`
- Call super with a wrapper completion block
  - Invoke the original completion handler
  - Attempt to salvage the file in case the user wants it
  - We haven't succeeded at that part yet

Speaking of Deletion…

- Put a file in a folder inside the ubiquity container
- Start editing the file on Client A
- Delete the folder on Client B
- Save your changes on Client A
- Save completes, then iCloud immediately deletes the folder (r.10247769)

iCloud: Lessons Learned

- Maintain a small set of serial queues
- Take care to interface with framework queues
- Be mindful of what thread you're executing on
- Be mindful of other processes touching your files
- Treat the filesystem as authoritative
- Directories suck
- File Radars!

Resources

- github.com/OmniGroup (OFSDocumentStore.m)
- cocoa-dev mailing list (lists.apple.com)
- Developer Forums (devforums.apple.com)
- @optshiftk
- optshiftk.com
- kyle.sluder@gmail.com

Thank You