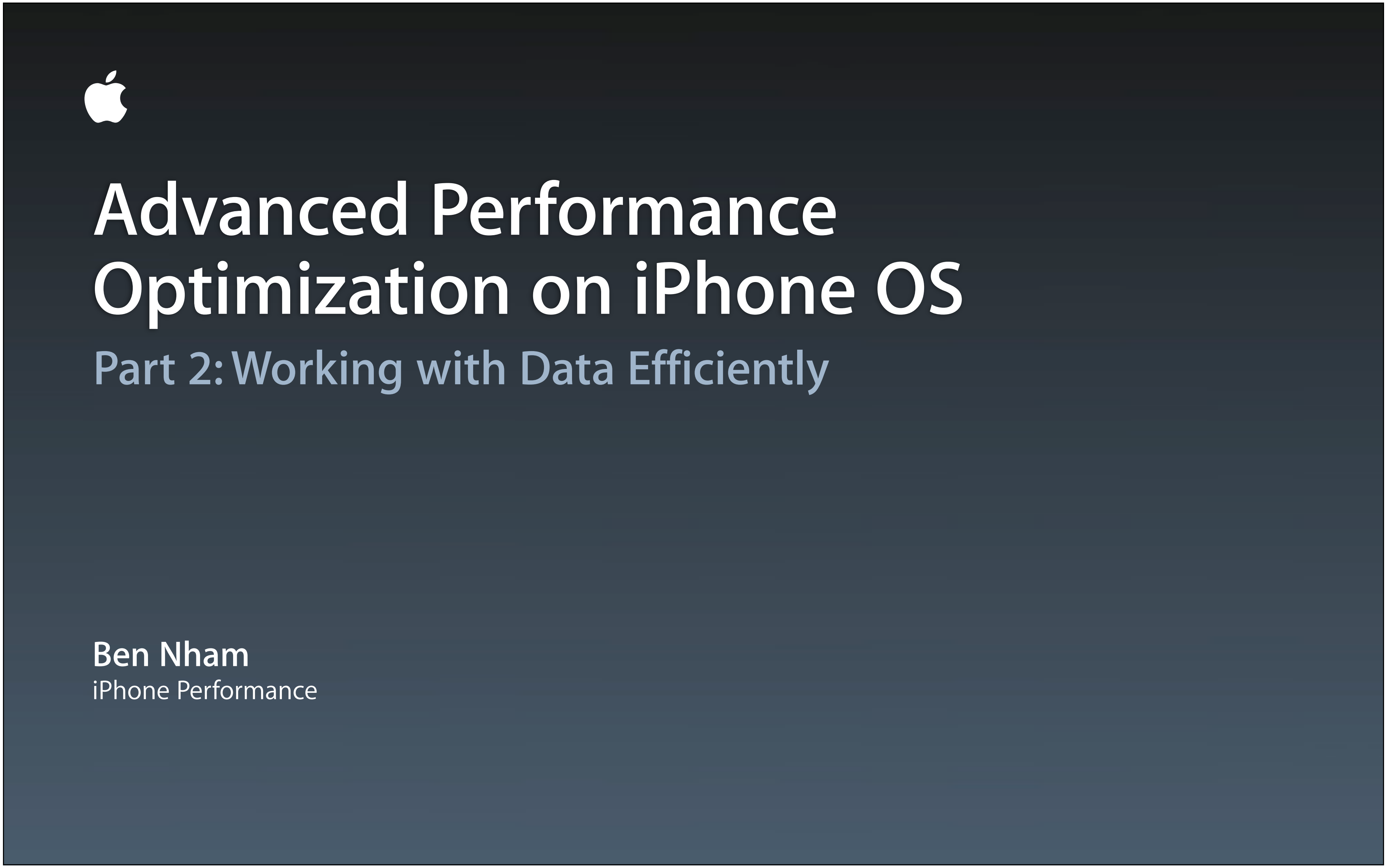


Advanced Performance Optimization on iPhone OS

Part 2: Working with Data Efficiently

Ben Nham
iPhone Performance

Introduction

- Focus on working with data efficiently
  - In-memory data structures
  - Serialization and deserialization
- Measurement tools
- Mental models
- Best practices

What You’ll Learn

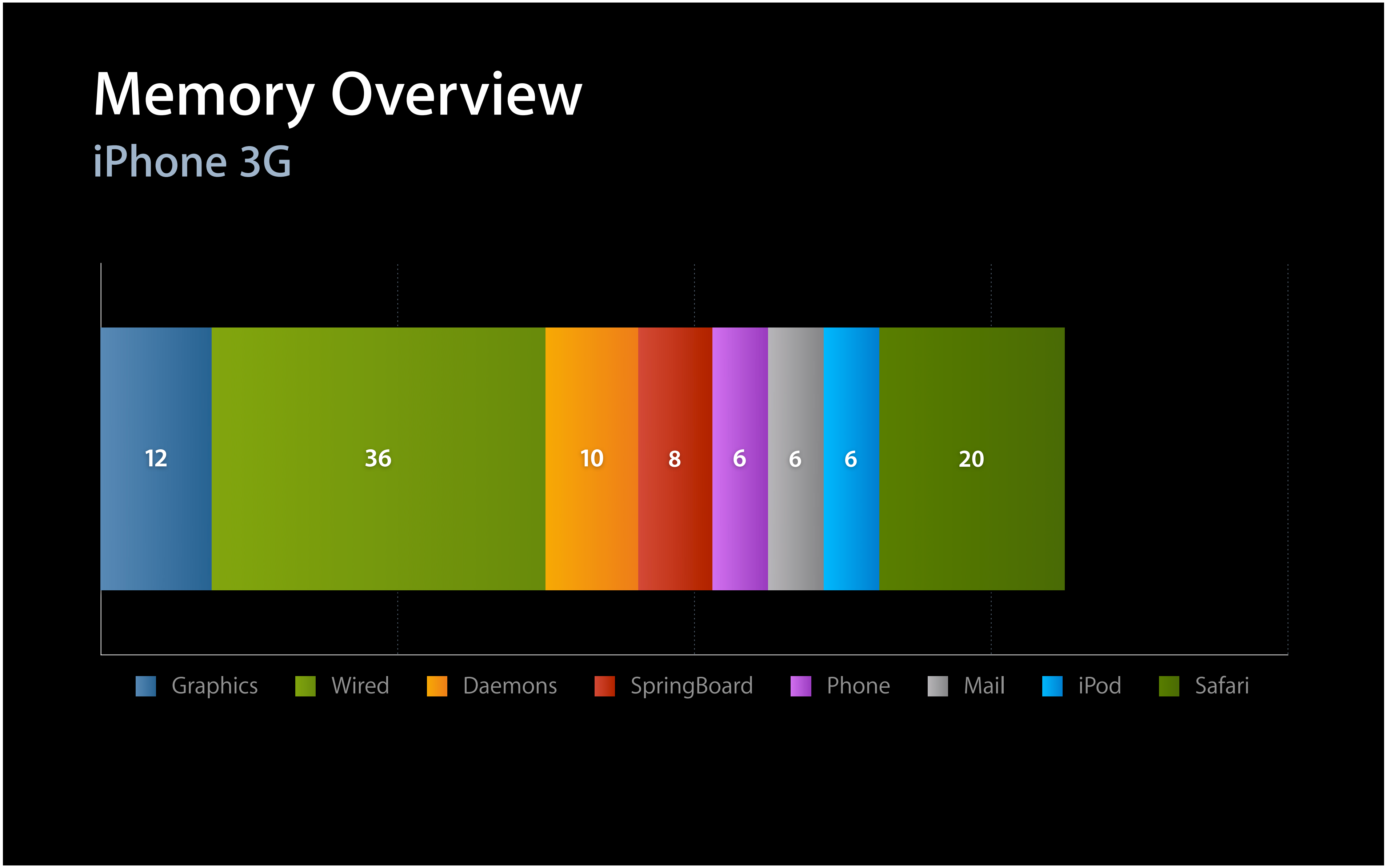
- Memory
- Foundation performance
- Filesystem
- Databases
- Scaling

Memory

Not a Desktop OS

- Limited memory
- Virtual memory, but no swap file
- Low memory notifications

	RAM
iPhone 4	512 MB
iPad	256 MB
iPhone 3GS	256 MB
3rd Generation iPod Touch	256 MB
2nd Generation iPod Touch	128 MB
iPhone 3G	128 MB
Mac Mini	2048 MB



Memory Overview

iPhone 3G

Virtual Memory

Paging

- The kernel deals with memory in 4KB chunks called pages
- Each application has a 32-bit address space broken into pages
- A page can be in several states
    - Nonresident
    - Resident and clean
    - Resident and dirty

Virtual Memory

Residency

- A page is resident if it is present in physical memory
- It is nonresident otherwise
    - If a nonresident page is accessed, a page fault occurs and the page becomes resident

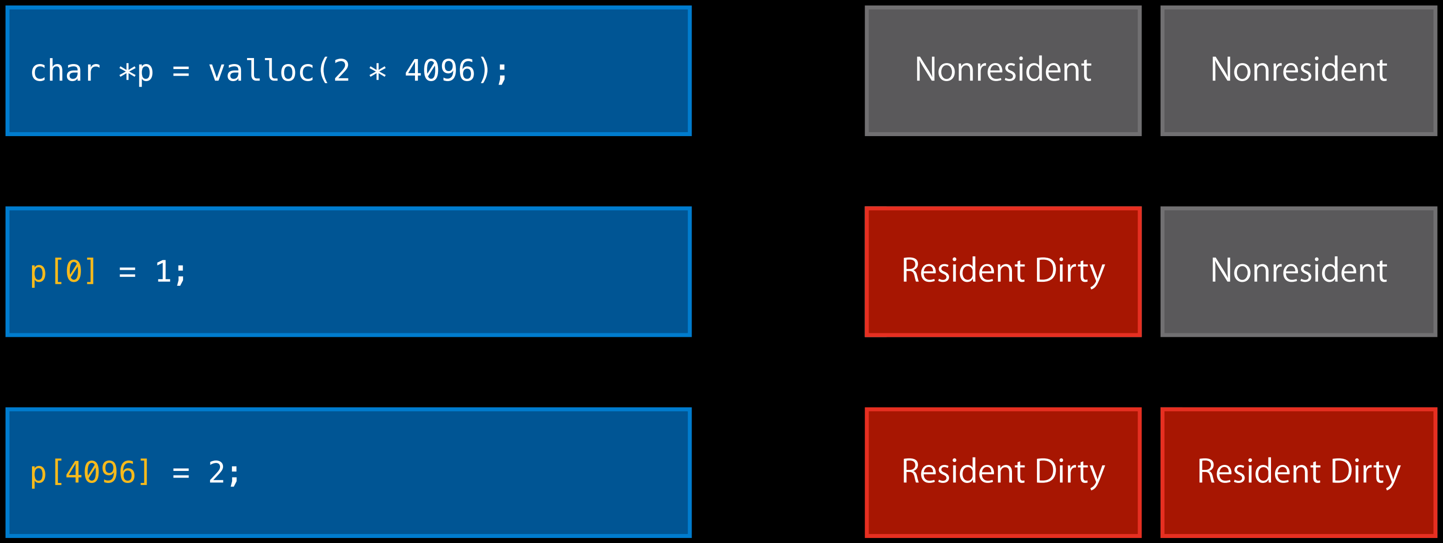
Virtual Memory

Dirty pages

- A resident page can be clean or dirty
  - Resident anonymous memory is always dirty (e.g., malloc)
  - Resident file-backed memory is usually clean
    - Becomes dirty if modified
- A clean page can be swapped out for “free”
  - But it still contributes to memory pressure in the system
- On iPhone OS, dirty pages cannot be swapped out!
  - Excessive amounts of dirty pages cause memory warnings and eventually the out-of-memory killer

Malloc Memory

- Malloc memory is anonymous (not backed by a file)
- When it is resident, it is dirty



`char *p = valloc(2 * 4096);`	Nonresident	Nonresident
`p[0] = 1;`	Resident Dirty	Nonresident
`p[4096] = 2;`	Resident Dirty	Resident Dirty

Example

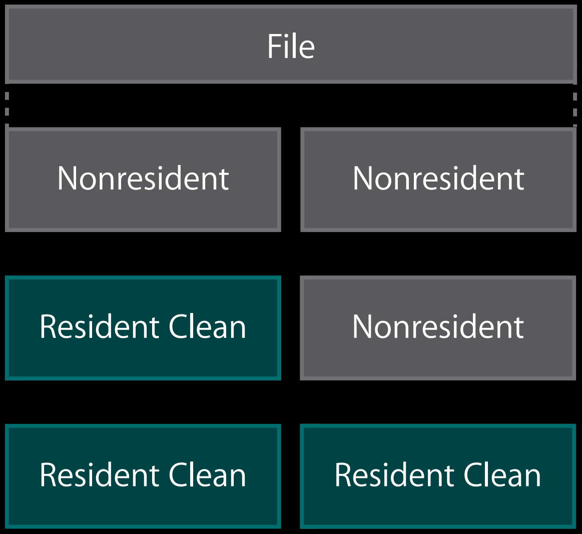
File-backed memory

- If mapped read-only, file backed memory will be clean when resident
- Code from app binary is mapped read-only

```
NSData *data = [NSData
    dataWithContentsOfMappedFile:file];
char *p = (char *)[data bytes];
```

```
printf("%c", p[0]);
```

```
printf("%c", p[4096]);
```



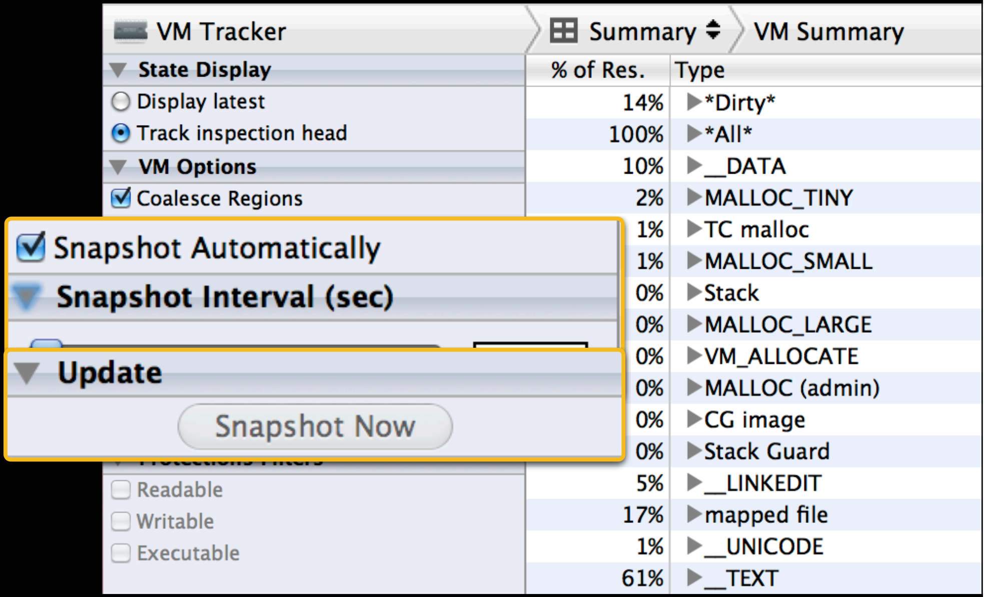
File

Nonresident	Nonresident
Resident Clean	Nonresident
Resident Clean	Resident Clean

VM Tracker

Taking snapshots

- A VM snapshot shows how memory usage is distributed across regions of memory usage
- To take a snapshot
    - Ask the instrument to periodically take snapshots automatically
    - Manually trigger a snapshot (default)
- Works best in simulator right now



State Display

○ Display latest

● Track inspection head

VM Options

☑ Coalesce Regions

☑ Snapshot Automatically

Snapshot Interval (sec)

Update

Protections Filters

□ Readable

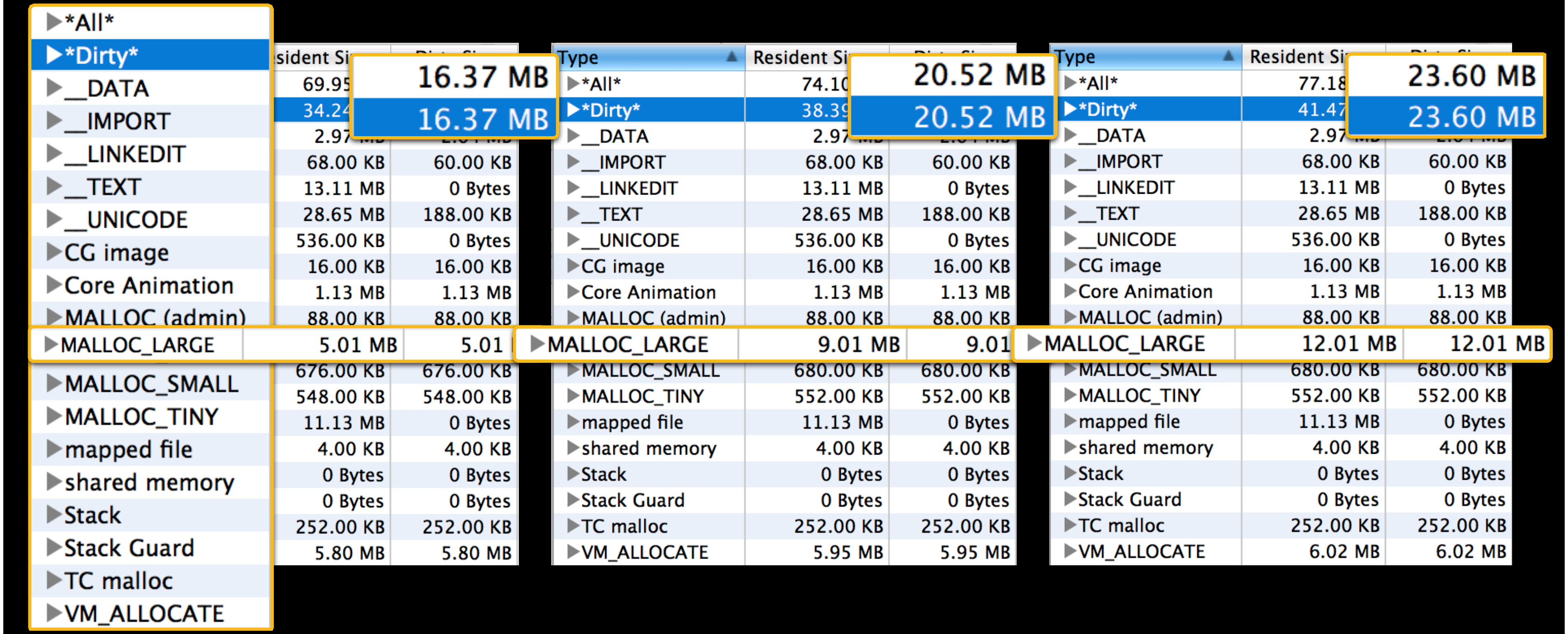
□ Writable

□ Executable

% of Res.	Type
14%	▶*Dirty*
100%	▶*All*
10%	▶__DATA
2%	▶MALLOC_TINY
1%	▶TC malloc
1%	▶MALLOC_SMALL
0%	▶Stack
0%	▶MALLOC_LARGE
0%	▶VM_ALLOCATE
0%	▶MALLOC (admin)
0%	▶CG image
0%	▶Stack Guard
5%	▶__LINKEDIT
17%	▶mapped file
1%	▶__UNICODE
61%	▶__TEXT

VM Tracker

Check samples over time



Type	Resident Si	Dirty Si
▶*All*	69.95	16.37 MB
▶*Dirty*	34.24	16.37 MB
▶__DATA	2.97 MB	[illegible]
▶__IMPORT	68.00 KB	60.00 KB
▶__LINKEDIT	13.11 MB	0 Bytes
▶__TEXT	28.65 MB	188.00 KB
▶__UNICODE	536.00 KB	0 Bytes
▶CG image	16.00 KB	16.00 KB
▶Core Animation	1.13 MB	1.13 MB
▶MALLOC (admin)	88.00 KB	88.00 KB
▶MALLOC_LARGE	5.01 MB	5.01
▶MALLOC_SMALL	676.00 KB	676.00 KB
▶MALLOC_TINY	548.00 KB	548.00 KB
▶mapped file	11.13 MB	0 Bytes
▶shared memory	4.00 KB	4.00 KB
▶Stack	0 Bytes	0 Bytes
▶Stack Guard	0 Bytes	0 Bytes
▶TC malloc	252.00 KB	252.00 KB
▶VM_ALLOCATE	5.80 MB	5.80 MB

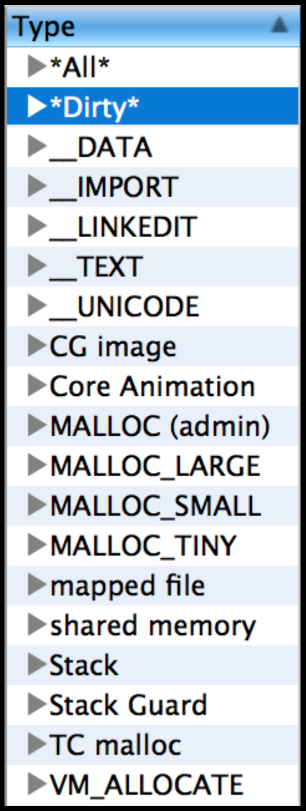
Type	Resident Si	Dirty Si
▶*All*	74.10	20.52 MB
▶*Dirty*	38.39	20.52 MB
▶__DATA	2.97 MB	[illegible]
▶__IMPORT	68.00 KB	60.00 KB
▶__LINKEDIT	13.11 MB	0 Bytes
▶__TEXT	28.65 MB	188.00 KB
▶__UNICODE	536.00 KB	0 Bytes
▶CG image	16.00 KB	16.00 KB
▶Core Animation	1.13 MB	1.13 MB
▶MALLOC (admin)	88.00 KB	88.00 KB
▶MALLOC_LARGE	9.01 MB	9.01
▶MALLOC_SMALL	680.00 KB	680.00 KB
▶MALLOC_TINY	552.00 KB	552.00 KB
▶mapped file	11.13 MB	0 Bytes
▶shared memory	4.00 KB	4.00 KB
▶Stack	0 Bytes	0 Bytes
▶Stack Guard	0 Bytes	0 Bytes
▶TC malloc	252.00 KB	252.00 KB
▶VM_ALLOCATE	5.95 MB	5.95 MB

Type	Resident Si	Dirty Si
▶*All*	77.18	23.60 MB
▶*Dirty*	41.47	23.60 MB
▶__DATA	2.97 MB	[illegible]
▶__IMPORT	68.00 KB	60.00 KB
▶__LINKEDIT	13.11 MB	0 Bytes
▶__TEXT	28.65 MB	188.00 KB
▶__UNICODE	536.00 KB	0 Bytes
▶CG image	16.00 KB	16.00 KB
▶Core Animation	1.13 MB	1.13 MB
▶MALLOC (admin)	88.00 KB	88.00 KB
▶MALLOC_LARGE	12.01 MB	12.01 MB
▶MALLOC_SMALL	680.00 KB	680.00 KB
▶MALLOC_TINY	552.00 KB	552.00 KB
▶mapped file	11.13 MB	0 Bytes
▶shared memory	4.00 KB	4.00 KB
▶Stack	0 Bytes	0 Bytes
▶Stack Guard	0 Bytes	0 Bytes
▶TC malloc	252.00 KB	252.00 KB
▶VM_ALLOCATE	6.02 MB	6.02 MB

VM Tracker

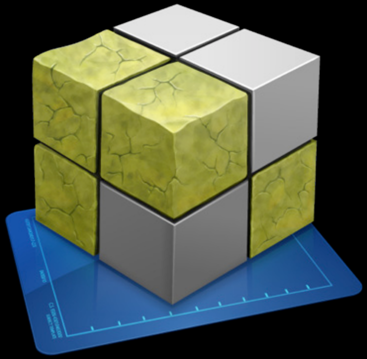
Check growth in dirty size of regions

- Growing dirty __DATA
  - Copy on write faults
  - Global variables that are modified
- Growing malloc
  - Check for leaks
  - Use Allocations tool to find backtraces
- Core Animation
  - Possible view leaks
- TC malloc
  - At least ~ 200KB used by WebKit



Type
▶ *All*
▶ *Dirty*
▶ __DATA
▶ __IMPORT
▶ __LINKEDIT
▶ __TEXT
▶ __UNICODE
▶ CG image
▶ Core Animation
▶ MALLOC (admin)
▶ MALLOC_LARGE
▶ MALLOC_SMALL
▶ MALLOC_TINY
▶ mapped file
▶ shared memory
▶ Stack
▶ Stack Guard
▶ TC malloc
▶ VM_ALLOCATE

Other Memory Measurement Tools

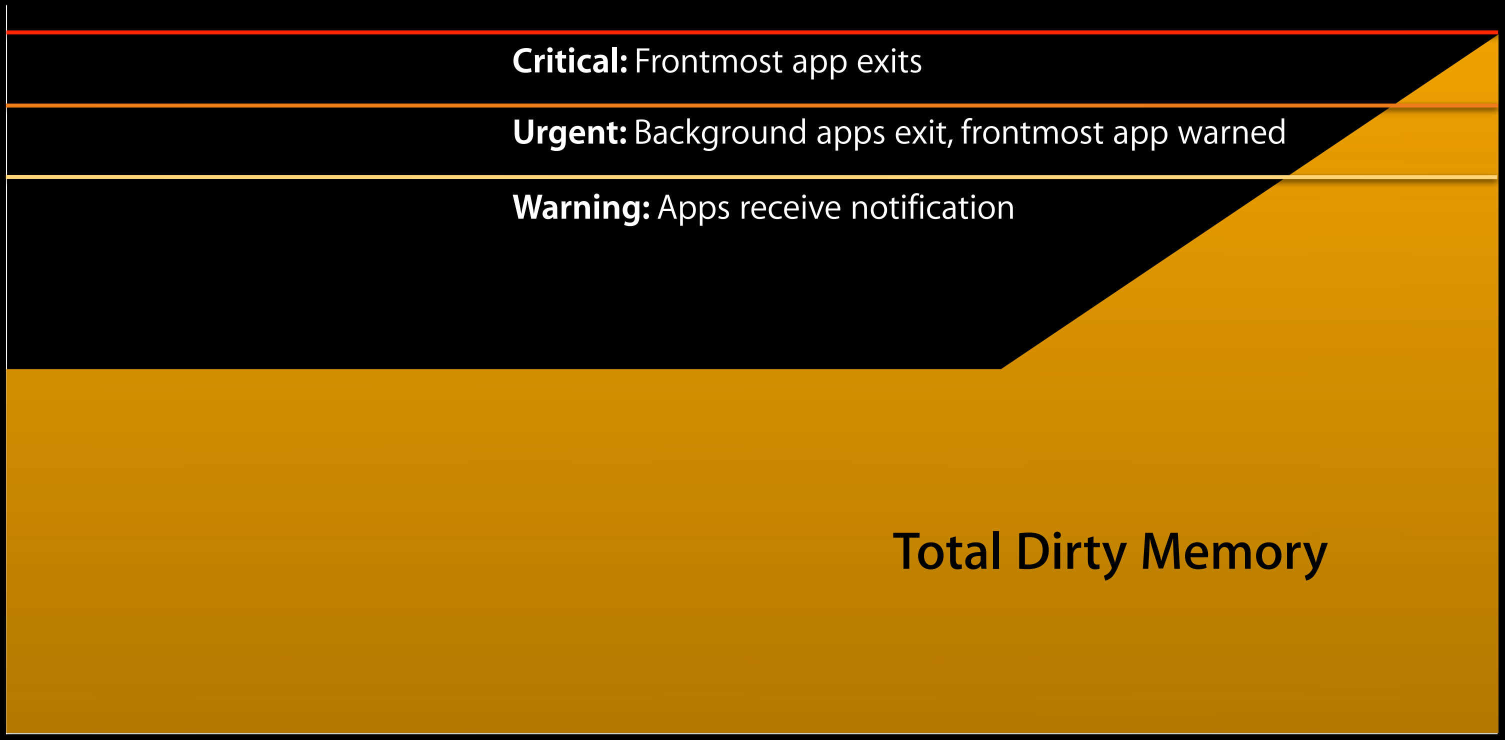


Low Memory Warnings

Don’t ignore the signs

- Any process may create low memory conditions
  - If the combined dirty memory usage of all processes becomes too high, a low memory notification is sent
- Expect memory warnings (normal part of the system)
- You must respond to low memory warnings!
  - Failure to respond can cause app termination

Low Memory Warnings



Critical: Frontmost app exits

Urgent: Background apps exit, frontmost app warned

Warning: Apps receive notification

Total Dirty Memory

Low Memory Warnings

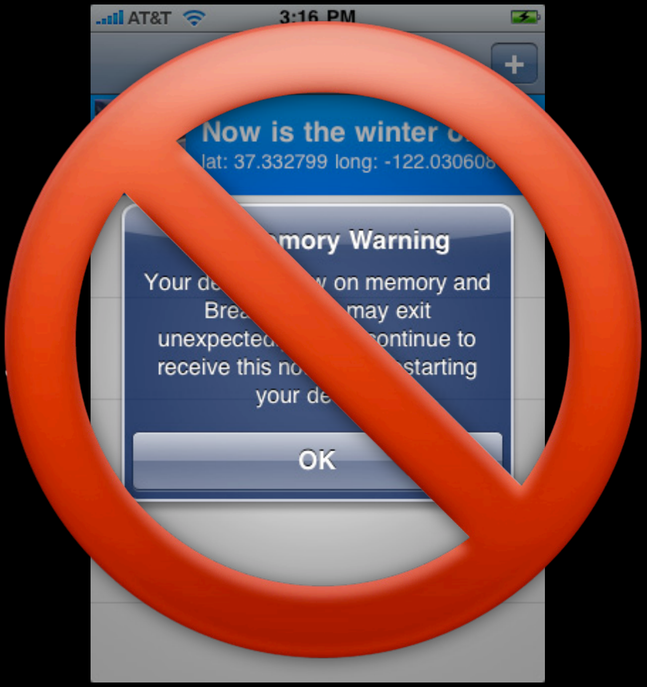
Taking action

- Release any objects that can be reconstructed
- Release cached objects
- Unload cached resource files

Low Memory Warnings

Taking action

- Don’t ask user to do anything (they can’t!)



Now is the winter o

lat: 37.332799 long: -122.030608

mory Warning

Your de w on memory and Brea may exit unexpected ontinue to receive this no starting your de

OK

Low Memory Warnings

Responding to low memory warnings

- In UIViewController subclasses
  - Override viewDidUnload
- In your app delegate
  - Implement applicationDidReceiveMemoryWarning: method
- Direct notifications
  - Register for UIApplicationDidReceiveMemoryWarningNotification

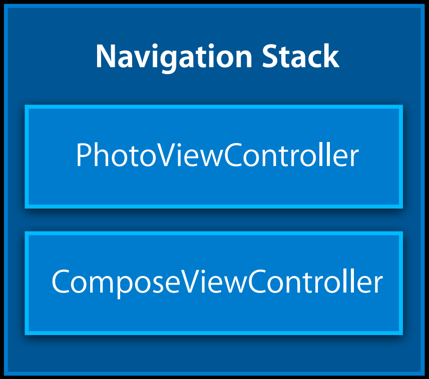
Low Memory Warnings

Unloading views

- -[UIViewController viewDidUnload] is called when the the controller's -view is unloaded
- But it needs help releasing views retained in instance variables

Low Memory Warnings

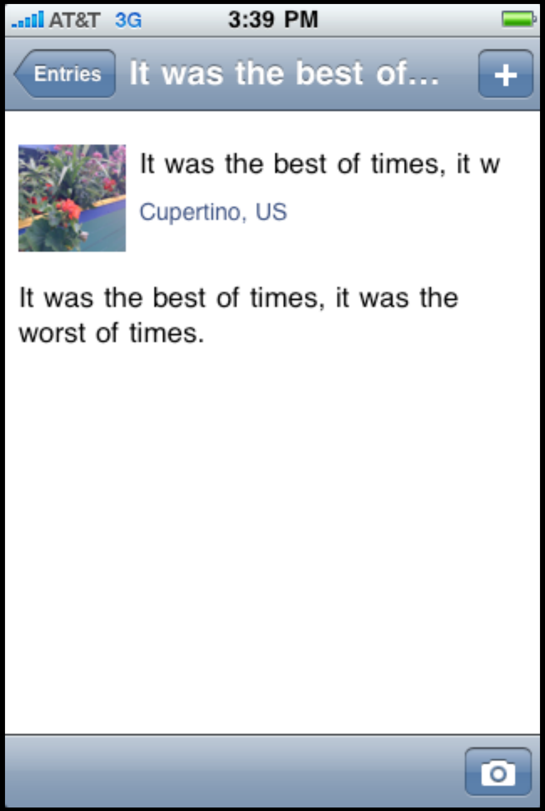
Unloading views



Navigation Stack

PhotoViewController

ComposeViewController

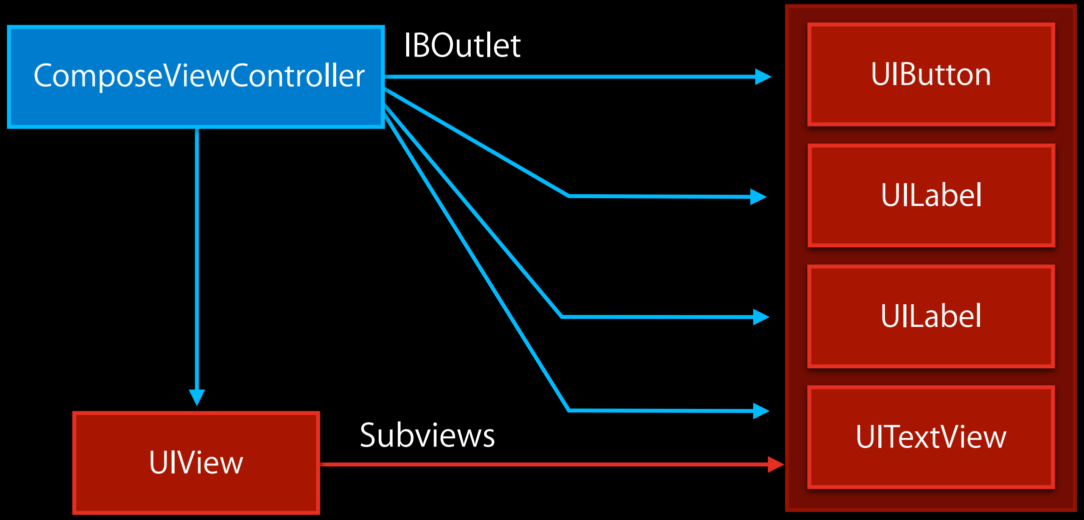


It was the best of times, it w

Cupertino, US

It was the best of times, it was the worst of times.

Low Memory Warnings



ComposeViewController

IBOutlet

UIButton

UILabel

UILabel

UITextView

UIView

Subviews

Low Memory Warnings

Unloading views

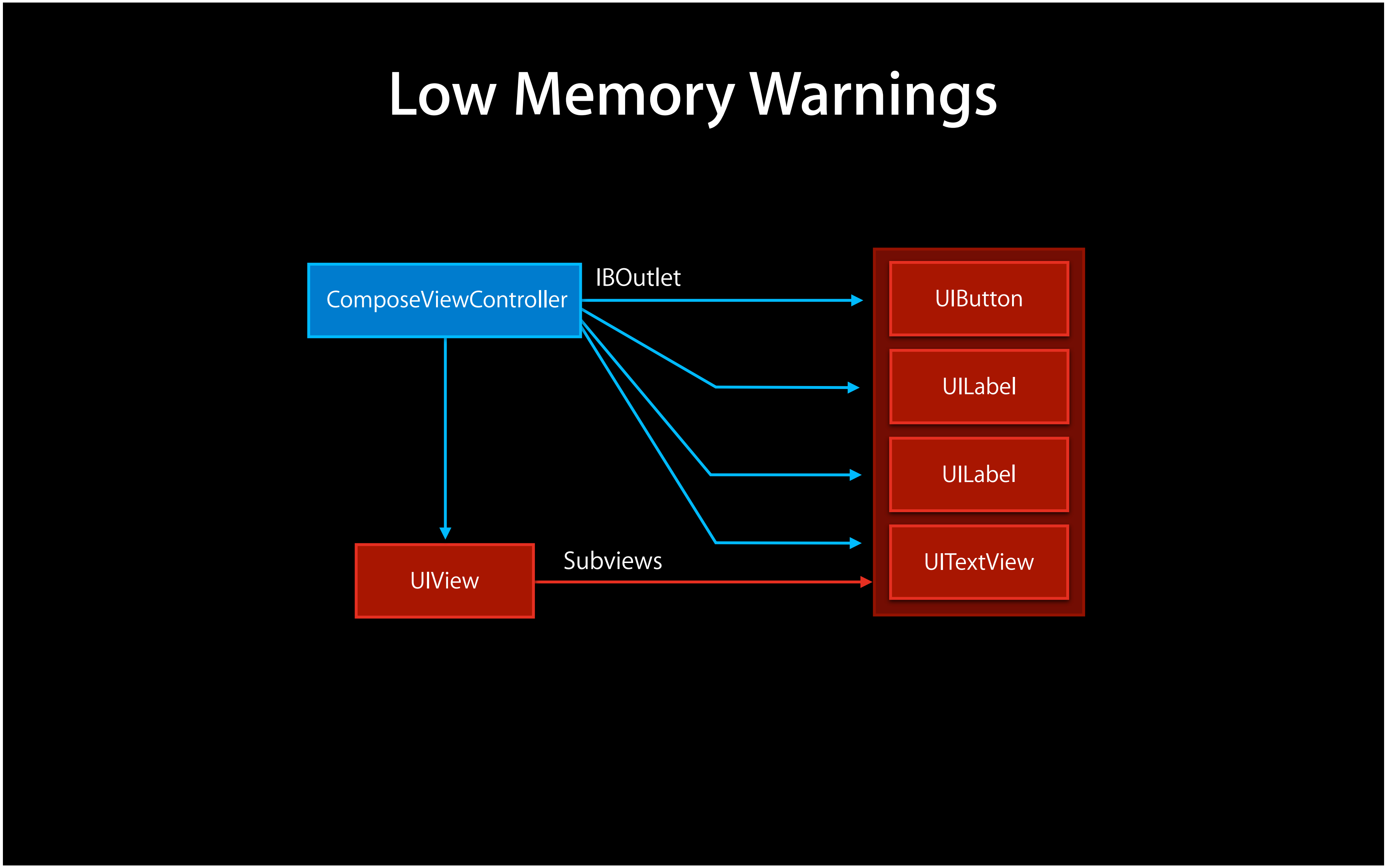
```
@interface ComposeViewController : UIViewController
{
    UILabel *titleLabel;
    UILabel *locationLabel;
    UITextView *textView;
    UIButton *imageButton;
}

@property (nonatomic, retain)
    IBOutlet UILabel *titleLabel;
@property (nonatomic, retain)
    IBOutlet UILabel *locationLabel;
@property (nonatomic, retain)
    IBOutlet UITextView *textView;
@property (nonatomic, retain)
    IBOutlet UIButton *imageButton;

@end
```

```
- (void)viewDidUnload {
    self.titleLabel = nil;
    self.locationLabel = nil;
    self.textView = nil;
    self.imageButton = nil;

    [super viewDidUnload];
}
```

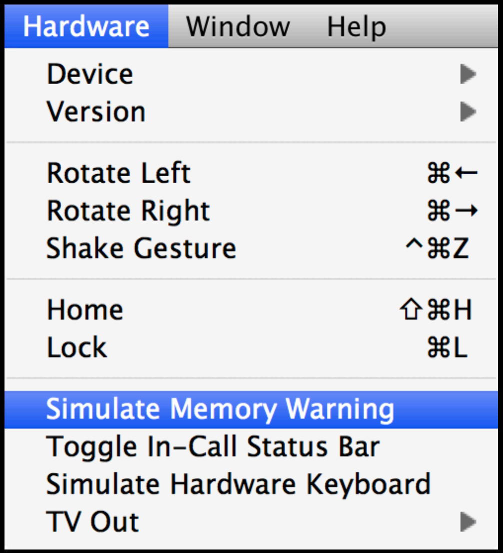


Low Memory Warnings

Low Memory Warnings

Simulating memory warnings

- Test memory warnings with the simulator



Interacting with Multitasking

- Low memory notifications are not sent when an app backgrounds
- Explicitly release resources in response to going into the background if not performing a background task
  - When delegate's applicationDidEnterBackground: is called
  - After receiving UIApplicationDidEnterBackgroundNotification
- Apps that use less memory have a lower chance of being terminated after suspension

Image Memory

Choosing the right method

- Use +[UIImage imageNamed:] with images that are used in UI elements
- Use +[UIImage imageWithContentsOfFile:] for everything else

Image Memory

Creating thumbnails with ImageIO

- CGImageSource can efficiently create thumbnails from data or file paths

```
// Assuming source is a CGImageSourceRef...

CFDictionaryRef options = (CFDictionaryRef)[NSDictionary dictionaryWithObjectsAndKeys:
  (id)kCFBooleanTrue, (id)kCGImageSourceCreateThumbnailWithTransform,
  (id)kCFBooleanTrue, (id)kCGImageSourceCreateThumbnailFromImageIfAbsent,
  (id)[NSNumber numberWithInt:size], (id)kCGImageSourceThumbnailMaxPixelSize];

CGImageRef imageRef = CGImageSourceCreateThumbnailAtIndex(source, 0, options);

if (imageRef) {
    image = [UIImage imageWithCGImage:imageRef];
    CGImageRelease(imageRef);
}
```

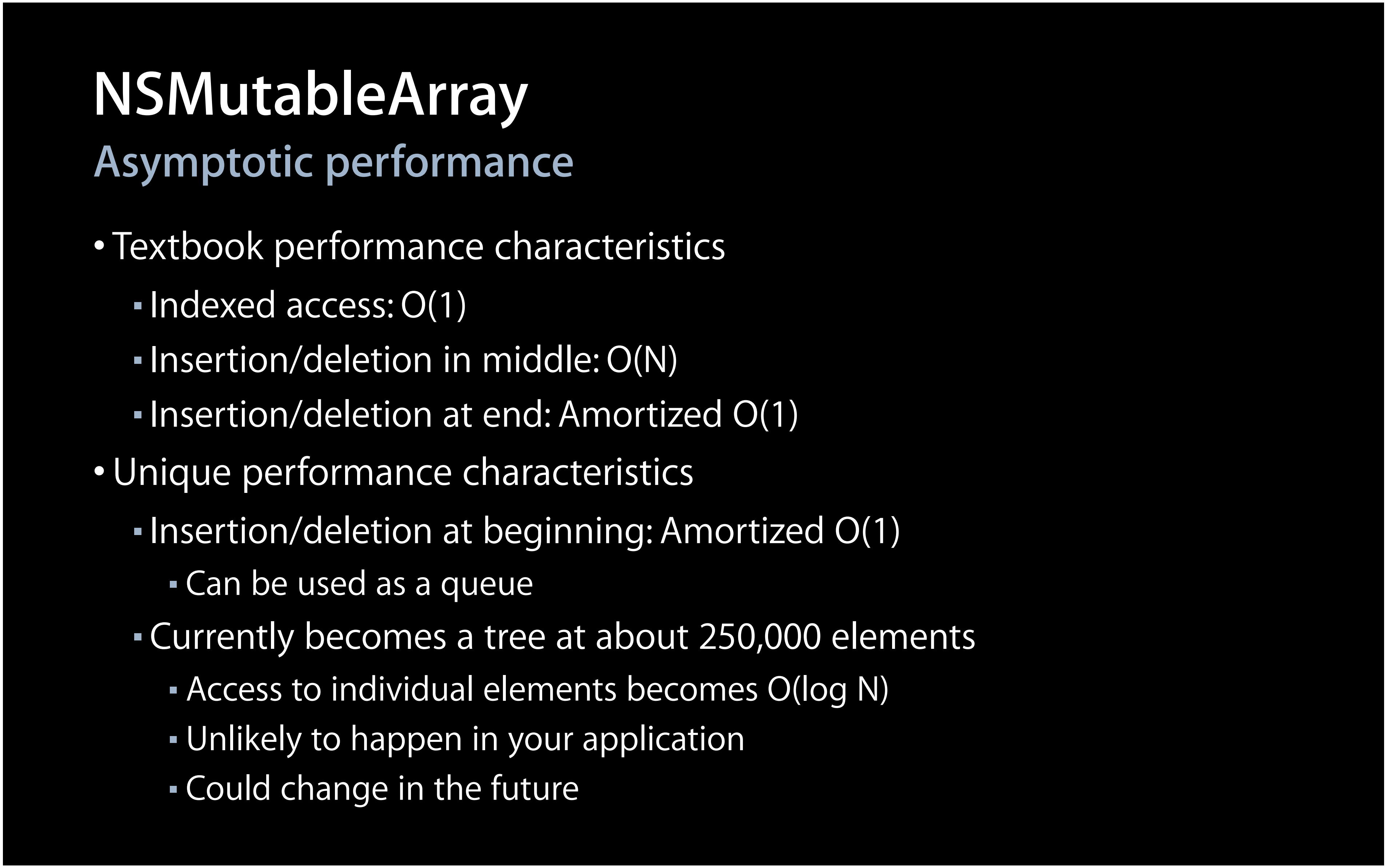
- Refer to Creating a Thumbnail Image in the Image I/O Programming Guide for more details

Memory

Summary

- Drive down the dirty memory usage of your app
- Respond to memory warnings
- Release resources when entering the background
- Additional resources
    - Introduction to Instruments User Guide
        - man vmmap to understand VM Tracker in detail
    - Memory Management Programming Guide

Foundation Performance



NSMutableArray

Asymptotic performance

- Textbook performance characteristics
    - Indexed access: O(1)
    - Insertion/deletion in middle: O(N)
    - Insertion/deletion at end: Amortized O(1)
- Unique performance characteristics
    - Insertion/deletion at beginning: Amortized O(1)
        - Can be used as a queue
    - Currently becomes a tree at about 250,000 elements
        - Access to individual elements becomes O(log N)
        - Unlikely to happen in your application
        - Could change in the future

NSMutableString

Asymptotic performance

- Indexed access: O(1)
- Insertion/deletion in middle: O(N)
- Insertion/deletion at end: Amortized O(1)

NSMutableDictionary

Asymptotic performance

- With a good hash function
  - Lookup, insertion, replacement, removal: O(1) on average
- With a bad hash function
  - Degenerates into an array or worse
  - Lookup: O(N)

NSMutableDictionary

Hash functions

- Bad hash functions

```
- (NSUInteger)hash {
    return 42;
}
```

✗

```
- (NSUInteger)hash {
    return random();
}
```

✗

- Return dispersed values
  - For objects that contain Foundation objects, XORing the -hash of each object is usually good enough

```
@interface ArrayDict : NSObject
    NSArray *_array;
    NSDictionary *_dict;
@end
```

```
- (NSUInteger)hash {
    return [_array hash] ^
      [_dict hash];
}
```

✓

NSMutableDictionary

Hash functions

- Make sure the hash function runs relatively quickly
  - When a dictionary grows, it has to rehash the existing values
  - Stick to relatively fast operations: add, shift, mask, XOR
- Remember the API contract
  - Keys are copied with NSCopying when calling -setObject:forKey:
  - Objects which are -isEqual: must return the same -hash

Avoiding Integer Boxing

- NSIndexSet can store ranges of indices efficiently without boxing
- CoreFoundation collections can store pointer-sized integers natively
    - Works for all collection types

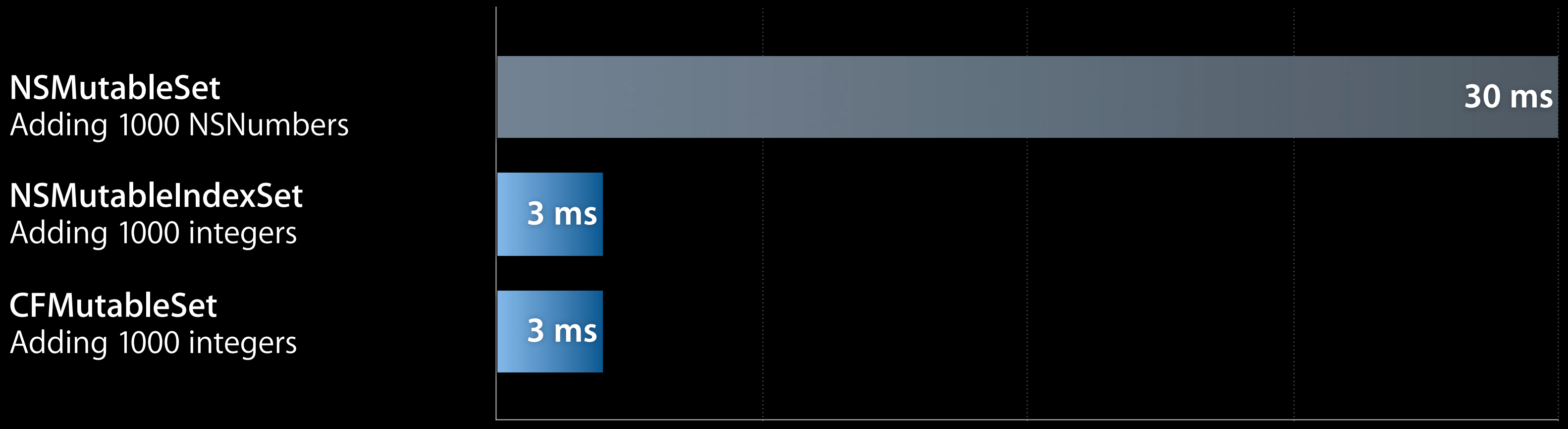
```
NSUInteger key = 0, value = 1;
```

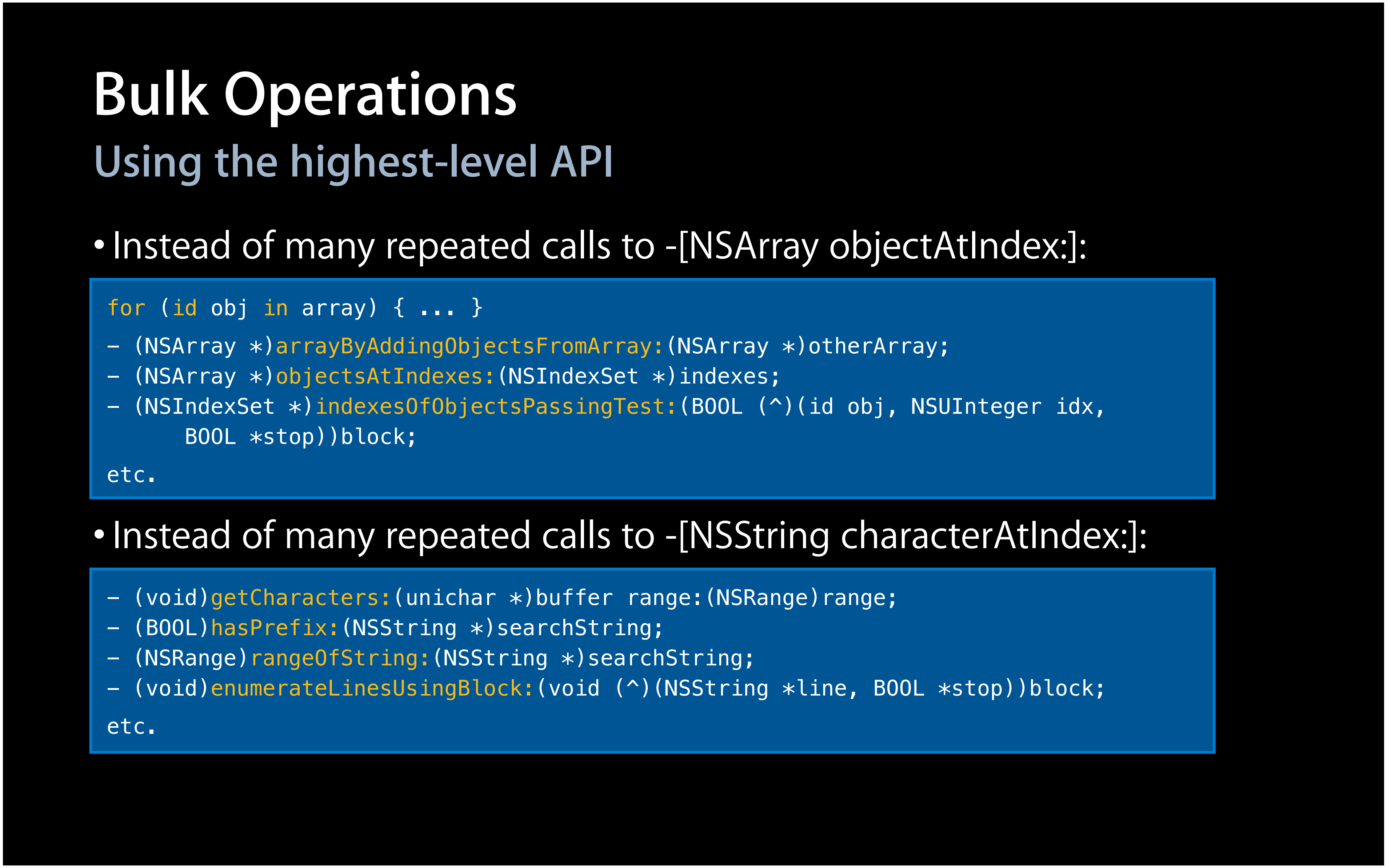
```
CFMutableArrayRef array = CFArrayCreateMutable(kCFAllocatorDefault, 0, NULL);
CFArrayAppendValue(array, (void *)key);
```

```
CFMutableSetRef set = CFSetCreateMutable(kCFAllocatorDefault, 0, NULL);
CFSetAddValue(set, (void *)key);
```

```
CFMutableDictionaryRef dict = CFDictionaryCreateMutable(kCFAllocatorDefault, 0,
    NULL, NULL);
CFDictionaryAddValue(dict, (void *)key, (void *)value);
```

Avoiding Integer Boxing





Bulk Operations

Using the highest-level API

- Instead of many repeated calls to -[NSArray objectAtIndex:]:

```
for (id obj in array) { ... }
- (NSArray *)arrayByAddingObjectsFromArray:(NSArray *)otherArray;
- (NSArray *)objectsAtIndexes:(NSIndexSet *)indexes;
- (NSIndexSet *)indexesOfObjectsPassingTest:(BOOL (^)(id obj, NSUInteger idx,
      BOOL *stop))block;
etc.
```

- Instead of many repeated calls to -[NSString characterAtIndex:]:

```
- (void)getCharacters:(unichar *)buffer range:(NSRange)range;
- (BOOL)hasPrefix:(NSString *)searchString;
- (NSRange)rangeOfString:(NSString *)searchString;
- (void)enumerateLinesUsingBlock:(void (^)(NSString *line, BOOL *stop))block;
etc.
```

NSRegularExpression

- Convenience methods in NSString are fine for one-off searches

```
[string rangeOfString:pattern options:NSRegularExpressionSearch];
```

- For repeated searches, create and reuse an NSRegularExpression object

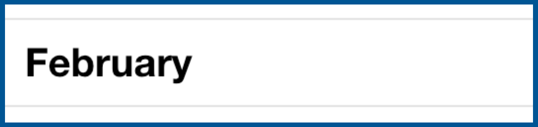
```
- (void)enumerateMatchesInString:(NSString *)string
                         options:(NSMatchingOptions)options
                           range:(NSRange)range
                      usingBlock:(void (^)(NSTextCheckingResult *result,
                                  NSMatchingFlags flags, BOOL *stop))block;
```

- By default, block is called back for every match
  - Use NSMatchingReportProgress to be called back periodically
  - Set the stop out parameter to YES to stop the search

Avoiding Expensive Initialization Costs

- Some classes are expensive to initialize and should not be initialized or mutated repeatedly if used multiple times
  - NSRegularExpression, NSDataDetector
  - NSDateFormatter, NSNumberFormatter

```
- (UITableViewCell *)tableView:(UITableView *)tableView
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // After creating or reusing a cell...
    NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
    [formatter setDateFormat:@"MMMM"];
    [[cell textLabel] setText:[formatter stringFromDate:date]];
    [formatter release];
    return cell;
}
```

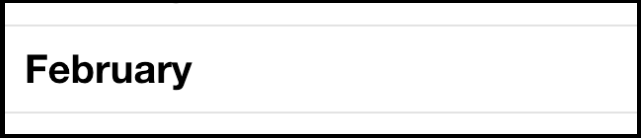


February

✗

Avoiding Expensive Initialization Costs

- Instead, lazily create a formatter for each style used, and keep using it



February

```
[[cell textLabel] setText:
    [MonthFormatter() stringFromDate:date]];
```

```
static NSDateFormatter *__monthFormatter = nil;

NSDateFormatter *MonthFormatter() {
    if (__monthFormatter == nil) {
        __monthFormatter = [[NSDateFormatter alloc] init];
        [__monthFormatter setDateStyle:@"MMMM"];
    }

    return __monthFormatter;
}
```

Avoiding Expensive Initialization Costs

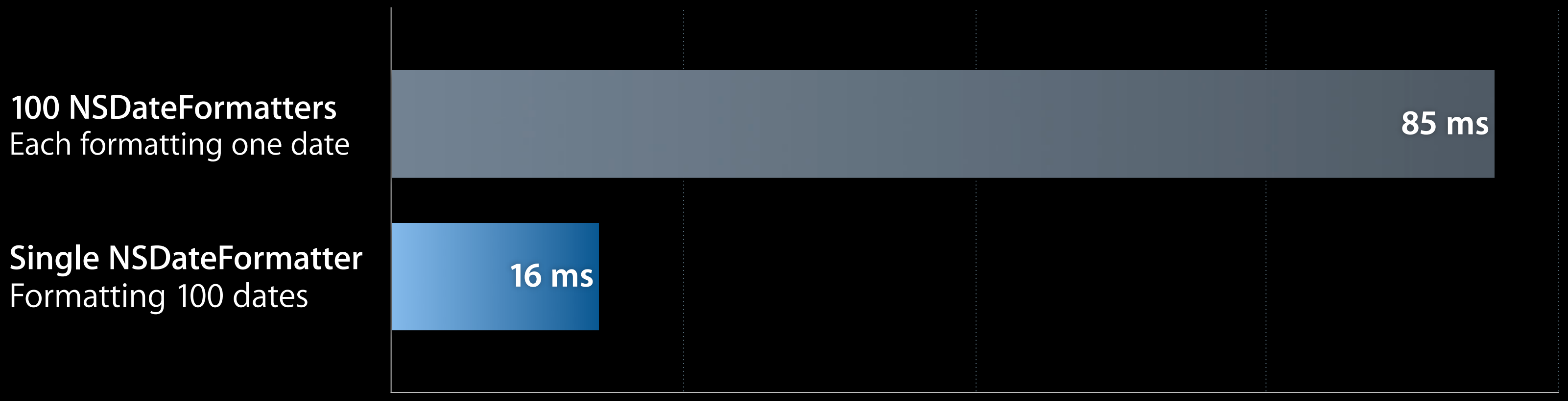
Some gotchas

- Date and number formatters do not automatically update when the locale changes, so this must be handled manually if they are cached

```
NSNotificationCenter *center = [NSNotificationCenter defaultCenter];
[center addObserverForName:NSCurrentLocaleDidChangeNotification
                    object:self
                     queue:[NSOperationQueue mainQueue]
                usingBlock:^(NSNotification *note) {
                               [__monthFormatter release];
                               __monthFormatter = nil;
                           }];
```

- Date and number formatters are not thread-safe
  - But NSRegularExpression, NSDataDetector are thread-safe

Avoiding Expensive Initialization Costs



Property Lists

Use the binary format

- Binary plists are 2–3x faster to decode than XML plists
- Plist resources in the app bundle are automatically converted to binary format at build time
- For plists created at run time, use NSPropertyListSerialization

```
–[NSArray writeToFile:atomically:]
–[NSDictionary writeToFile:atomically:]
–[NSString writeToFile:atomically:encoding:error:]
```

✗

```
NSData *data = [NSPropertyListSerialization
    dataWithPropertyList:dictionary
    format:NSPropertyListBinaryFormat_v1_0
    options:0
    error:NULL];
[data writeToFile:path atomically:YES];
```

✓

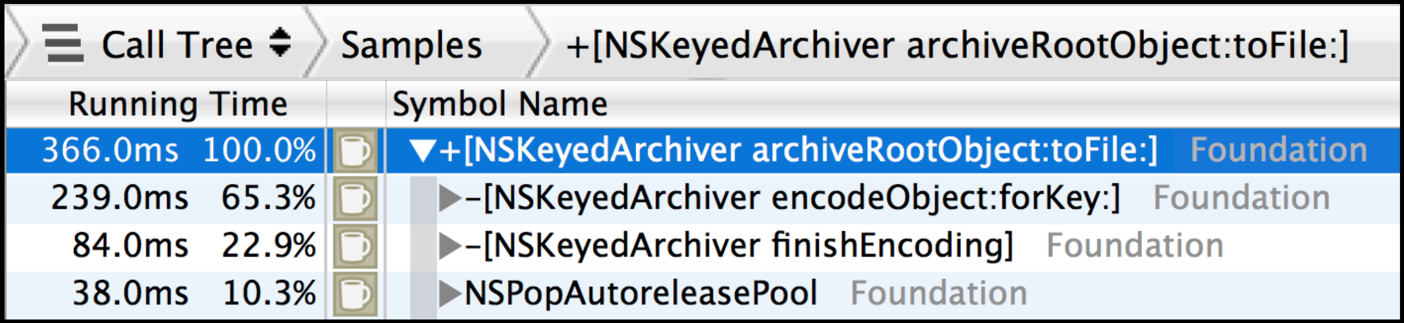
Property Lists

Proper usage

- Plists are great for storing small bits of data, like configuration files
    - Up to tens of kilobytes is generally fine
- Not an incremental format
    - Entire object graph in the plist must be recreated in memory at deserialization time
    - Entire object graph must be traversed and rewritten at serialization time
- Use another file format or a database to incrementally deserialize or serialize information

NSCoding

- Only use this for small object graphs
  - Large object graphs can take hundreds of milliseconds to read or write
  - Measure using Time Profiler

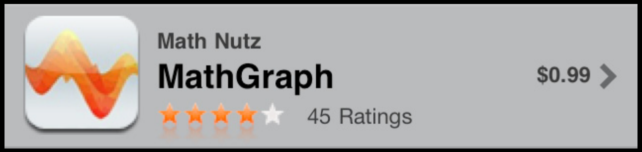


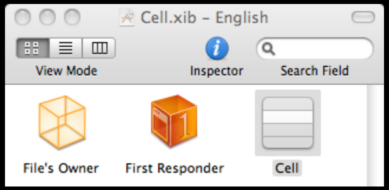
Running Time		Symbol Name	
366.0ms	100.0%	▼+[NSKeyedArchiver archiveRootObject:toFile:]	Foundation
239.0ms	65.3%	▶–[NSKeyedArchiver encodeObject:forKey:]	Foundation
84.0ms	22.9%	▶–[NSKeyedArchiver finishEncoding]	Foundation
38.0ms	10.3%	▶NSPopAutoreleasePool	Foundation

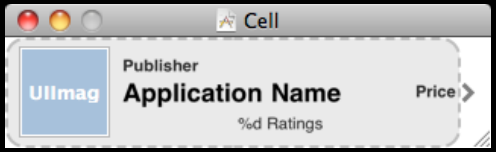
- NIBs use NSCoding
  - Avoid stuffing NIBs with unnecessary top-level objects

UINib

- Avoids deserializing NIBs from scratch for commonly accessed resources
- Useful for table view cell NIBs







```
- (UITableViewCell *)tableView:(UITableView *)view
        cellForRowAtIndexPath:(NSIndexPath *)idx
{
    AppCell *cell = (AppCell *)[tableView dequeueReusableCellWithIdentifier:@"AppCell"];
    if (cell == nil) {
        // load cell from NIB file
    }
}
```

UINib

- Old method of using NSBundle:

```
if (cell == nil) {
    NSArray *topLevelObjects = [[NSBundle mainBundle]
        loadNibNamed:@"Cell" owner:self options:nil];
    cell = [topLevelObjects objectAtIndex:0];
}
```

✗

- New method of using UINib:

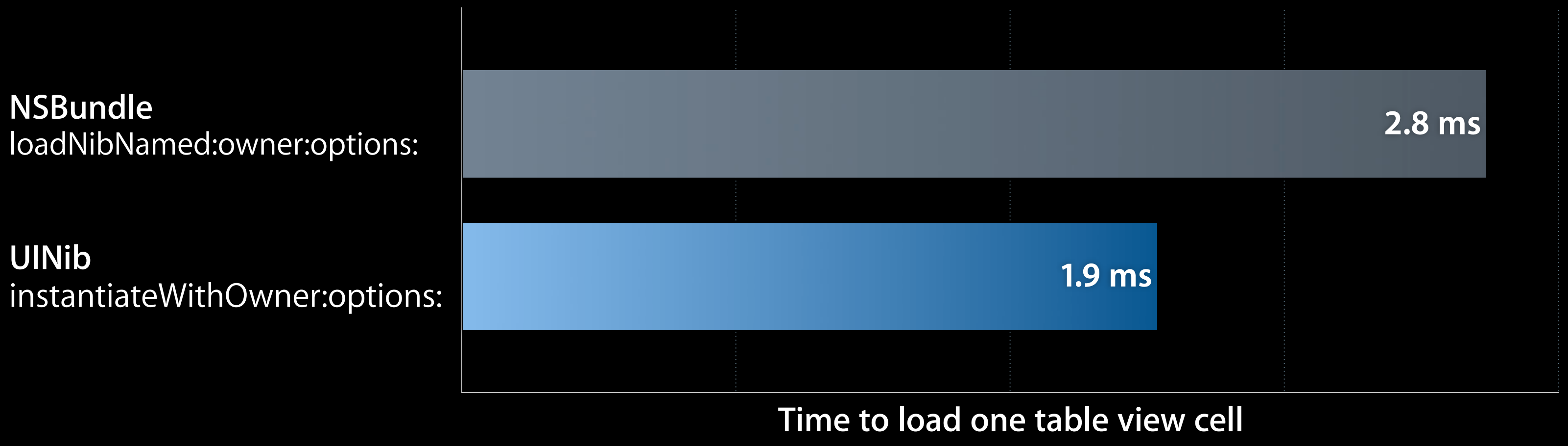
```
if (cell == nil) {
    if (!cellNib) // instance var
        cellNib = [UINib nibWithNibName:@"Cell" bundle:nil];

    NSArray *topLevelObjects = [cellNib instantiateWithOwner:self
                                                     options:nil];

    cell = [topLevelObjects objectAtIndex:0];
}
```

✓

UINib


NSBundle
loadNibNamed:owner:options:
2.8 ms
UINib
instantiateWithOwner:options:
1.9 ms
Time to load one table view cell

Foundation Performance

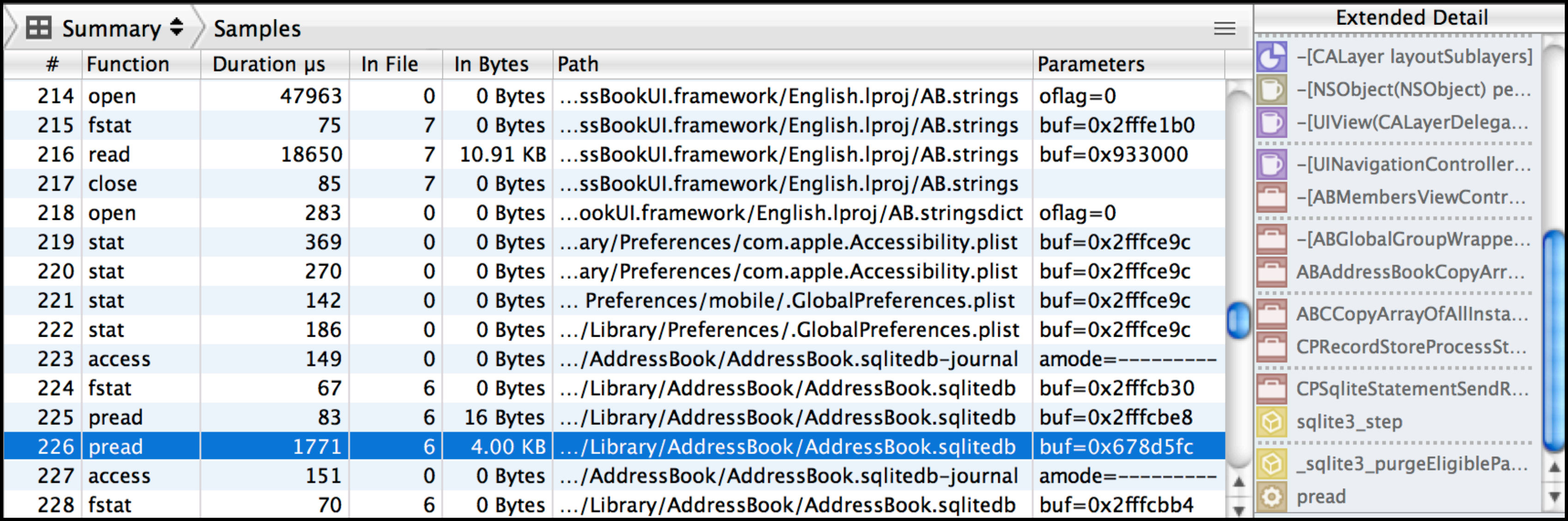
Summary

- Foundation types generally have good performance if used correctly
- Understand the API
  - Use a higher-level methods if possible
  - Avoid expensive re-initialization of certain classes
  - Use plists and NSCoding for small object graphs
- Additional resources
  - Collections Programming Topics
  - Property List Programming Guide
  - Archives and Serialization Programming Guide
  - Resource Programming Guide

Filesystem

Filesystem

Measuring performance with System Usage



#	Function	Duration µs	In File	In Bytes	Path	Parameters
214	open	47963	0	0 Bytes	...ssBookUI.framework/English.lproj/AB.strings	oflag=0
215	fstat	75	7	0 Bytes	...ssBookUI.framework/English.lproj/AB.strings	buf=0x2fffe1b0
216	read	18650	7	10.91 KB	...ssBookUI.framework/English.lproj/AB.strings	buf=0x933000
217	close	85	7	0 Bytes	...ssBookUI.framework/English.lproj/AB.strings	
218	open	283	0	0 Bytes	...ookUI.framework/English.lproj/AB.stringsdict	oflag=0
219	stat	369	0	0 Bytes	...ary/Preferences/com.apple.Accessibility.plist	buf=0x2fffce9c
220	stat	270	0	0 Bytes	...ary/Preferences/com.apple.Accessibility.plist	buf=0x2fffce9c
221	stat	142	0	0 Bytes	... Preferences/mobile/.GlobalPreferences.plist	buf=0x2fffce9c
222	stat	186	0	0 Bytes	.../Library/Preferences/.GlobalPreferences.plist	buf=0x2fffce9c
223	access	149	0	0 Bytes	.../AddressBook/AddressBook.sqlitedb-journal	amode=---------
224	fstat	67	6	0 Bytes	.../Library/AddressBook/AddressBook.sqlitedb	buf=0x2fffcb30
225	pread	83	6	16 Bytes	.../Library/AddressBook/AddressBook.sqlitedb	buf=0x2fffcbe8
226	pread	1771	6	4.00 KB	.../Library/AddressBook/AddressBook.sqlitedb	buf=0x678d5fc
227	access	151	0	0 Bytes	.../AddressBook/AddressBook.sqlitedb-journal	amode=---------
228	fstat	70	6	0 Bytes	.../Library/AddressBook/AddressBook.sqlitedb	buf=0x2fffcbb4

Extended Detail

-[CALayer layoutSublayers]
-[NSObject(NSObject) pe...
-[UIView(CALayerDelega...
-[UINavigationController...
-[ABMembersViewContr...
-[ABGlobalGroupWrappe...
ABAddressBookCopyArr...
ABCCopyArrayOfAllInsta...
CPRecordStoreProcessSt...
CPSqliteStatementSendR...
sqlite3_step
_sqlite3_purgeEligiblePa...
pread

- Use to ensure I/O activity seems sane
    - Extended detail shows backtrace that caused I/O
- Doesn’t yet measure bytes that are demand-paged from mapped files

Filesystem

Best practices

- Test apps on multiple kinds of devices
  - Significant differences in read/write performance
- Avoid doing long I/Os on main thread
- For extremely large files, avoid +[NSData dataWithContentsOfFile:]
  - Reads the entire file eagerly into a dirty memory buffer
  - Alternatives
    - Demand page data with +[NSData dataWithContentsOfMappedFile:]
    - Incrementally read data with -[NSFileHandle readDataOfLength:]
- Avoid repeatedly opening or checking attributes of a path
  - Incurs cost for path permissions check

Filesystem

Accessing paths

- Get read-only paths to application bundle with NSBundle
- Store preferences in application sandbox with NSUserDefaults
- Get writable paths in your application sandbox with NSSearchPathForDirectoriesInDomains or NSTemporaryDirectory

	Persists Across Launches	Persists Across Updates	Backed up by iTunes
NSDocumentDirectory	✓	✓	✓
NSUserDefaults	✓	✓	✓
NSCachesDirectory	✓		
NSTemporaryDirectory			

- Do not write outside of your application’s sandbox

Filesystem

Summary

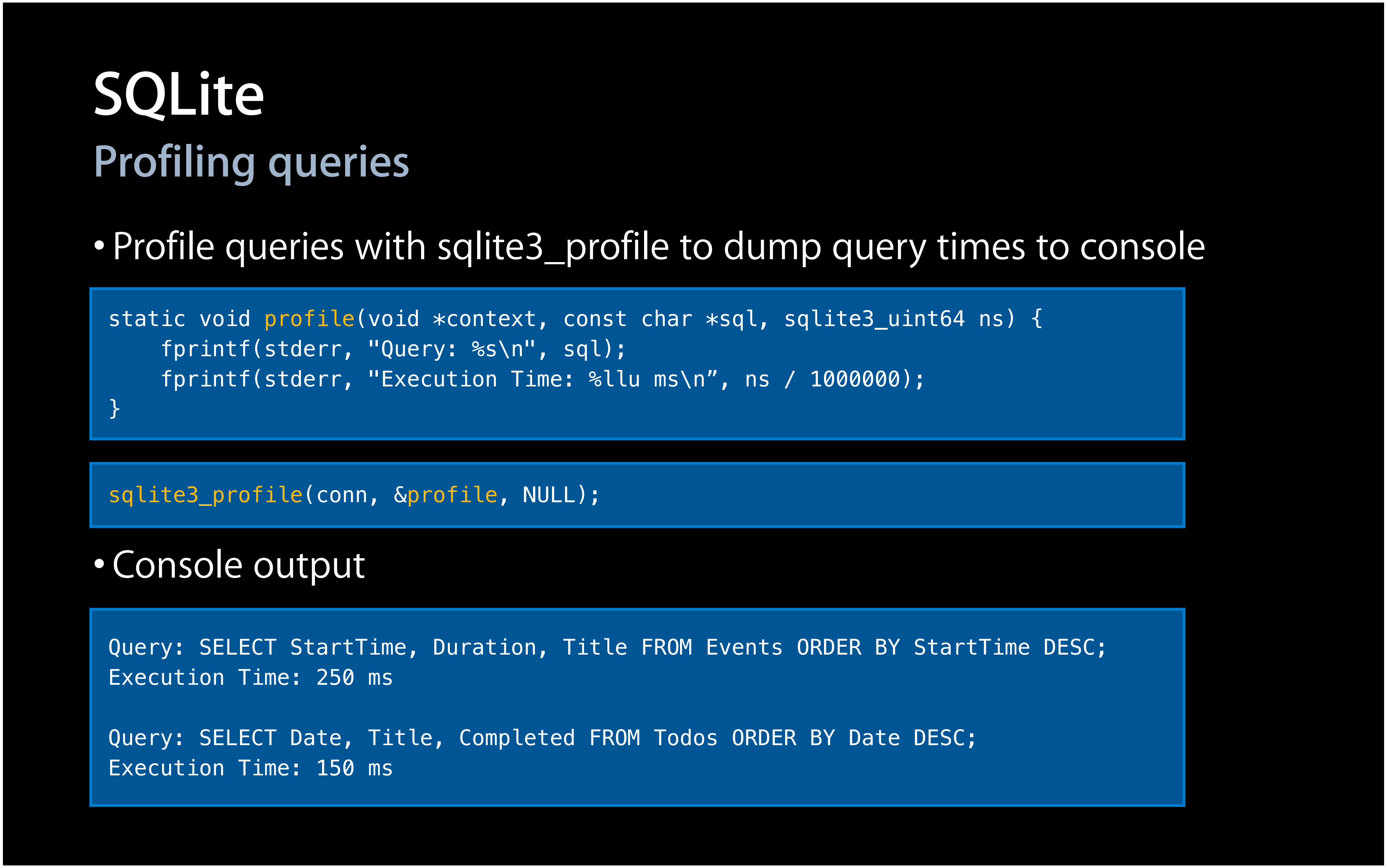
- Use System Usage to determine if there are filesystem bottlenecks in your app
- For large files, prefer interfaces and formats that read incrementally instead of all at once
- Perform long I/Os off the main thread
- Choose the correct path to avoid unnecessary backups

Databases

Databases

Overview

- Allow incremental reading and writing of data
- Great for transactional storage of structured information
- Use Core Data if possible
    - Provides automatic schema management
    - Has iPhone specific enhancements (e.g., table view section caching)
- Native SQLite library is available, but is much more low level
- Understand data modeling
    - “Object Modeling” in the Cocoa Fundamentals Guide



SQLite

Profiling queries

- Profile queries with sqlite3_profile to dump query times to console

```
static void profile(void *context, const char *sql, sqlite3_uint64 ns) {
    fprintf(stderr, "Query: %s\n", sql);
    fprintf(stderr, "Execution Time: %llu ms\n”, ns / 1000000);
}
```

```
sqlite3_profile(conn, &profile, NULL);
```

- Console output

```
Query: SELECT StartTime, Duration, Title FROM Events ORDER BY StartTime DESC;
Execution Time: 250 ms

Query: SELECT Date, Title, Completed FROM Todos ORDER BY Date DESC;
Execution Time: 150 ms
```

SQLite

Prepared statements

- Statement objects are backed by a program interpreted by SQLite
  - The EXPLAIN command shows the actual program
- Cache prepared statements that you plan to use over and over
  - Use bind parameters to change the statement’s behavior
- Don’t cache prepared statements you don’t plan to reuse

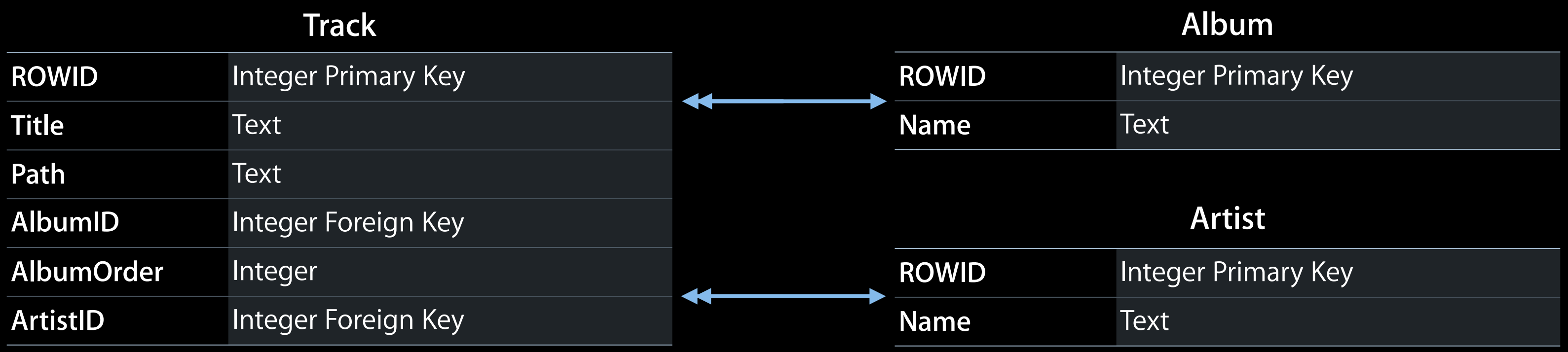
SQLite

Query plans

- Use EXPLAIN QUERY PLAN and EXPLAIN to understand the behavior of a statement
    - Execute the commands using the sqlite3 tool on your host
- Order of tables in a JOIN can affect query plan
- Watch out for transient tables
    - EXPLAIN will show an OpenEphemeral instruction
    - Common causes
        - Sorting without an index
        - Subselects
    - Can make the first sqlite3_step take a long time

SQLite

Sample schema



Track

ROWID	Integer Primary Key
Title	Text
Path	Text
AlbumID	Integer Foreign Key
AlbumOrder	Integer
ArtistID	Integer Foreign Key

Album

ROWID	Integer Primary Key
Name	Text

Artist

ROWID	Integer Primary Key
Name	Text

SQLite

Naive query plan

Track

ROWID	Title	Path	AlbumID	Album Order	ArtistID
1	A	T	3	1	1
2	Title	Path	AlbumID	Album Order	ArtistID
3	C	V	2	3	3
4	D	X	2	1	2
5	E	Y	2	2	3
6	F	Z	1	1	1

Transient Table

Title	Path	AlbumID	Album Order	ArtistID
D	X	2	1	2
E	Y	2	2	3
C	V	2	3	3

- Select all tracks in an album, ordered by track number

```
sqlite> EXPLAIN QUERY PLAN
   ...> SELECT * FROM Track WHERE AlbumID=2 ORDER BY AlbumOrder;
0|0|TABLE Track
```

- Table scan for WHERE, plus a sort of a transient table for ORDER BY

SQLite

Better query plan

Index

AlbumID	ROWID
1	6
2	3
2	4
2	5
3	1
3	2

Track

ROWID	Title	Path	AlbumID	Album Order	ArtistID
1	A	T	2	1	1
2	Title	Path	AlbumID	Album Order	ArtistID
3	C	V	2	3	3
4	D	X	2	1	2
5	E	Y	2	2	3
6	F	Z	1	1	1

Transient Table

Title	Path	AlbumID	Album Order	ArtistID
D	X	2	1	2
E	Y	2	2	3
C	V	2	3	3

- After adding an index to help finding all tracks in an album

```
sqlite> CREATE INDEX TrackAlbumIDIndex ON Track(AlbumID);
sqlite> EXPLAIN QUERY PLAN
   ...> SELECT * FROM Track WHERE AlbumID=2 ORDER BY AlbumOrder;
0|0|TABLE Track WITH INDEX TrackAlbumIDIndex
```

- Finds all tracks in an album using an index
- ORDER BY still handled by sorting a transient table

SQLite

Best query plan

Index

AlbumID	Album Order	ROWID
1	1	6
2	1	4
2	2	5
2	3	3
3	1	1
3	2	2

Track

ROWID	Title	Path	AlbumID	Album Order	ArtistID
1	A	T	3	1	1
2	B	U	3	2	2
3	C	V	2	3	3
4	D	X	2	1	2
5	E	Y	2	2	3
6	F	Z	1	1	1

- Select all tracks in an album, ordered by track number

```
sqlite> CREATE INDEX TrackAlbumIDOrderIndex ON Track(AlbumID, AlbumOrder);
sqlite> EXPLAIN QUERY PLAN
   ...> SELECT * FROM Track WHERE AlbumID=? ORDER BY AlbumOrder;
0|0|TABLE Track WITH INDEX TrackAlbumIDOrderIndex ORDER BY
```

- Finds all tracks in an album in logarithmic time using the index
- Uses second column in index to iterate over Track in sorted order

SQLite

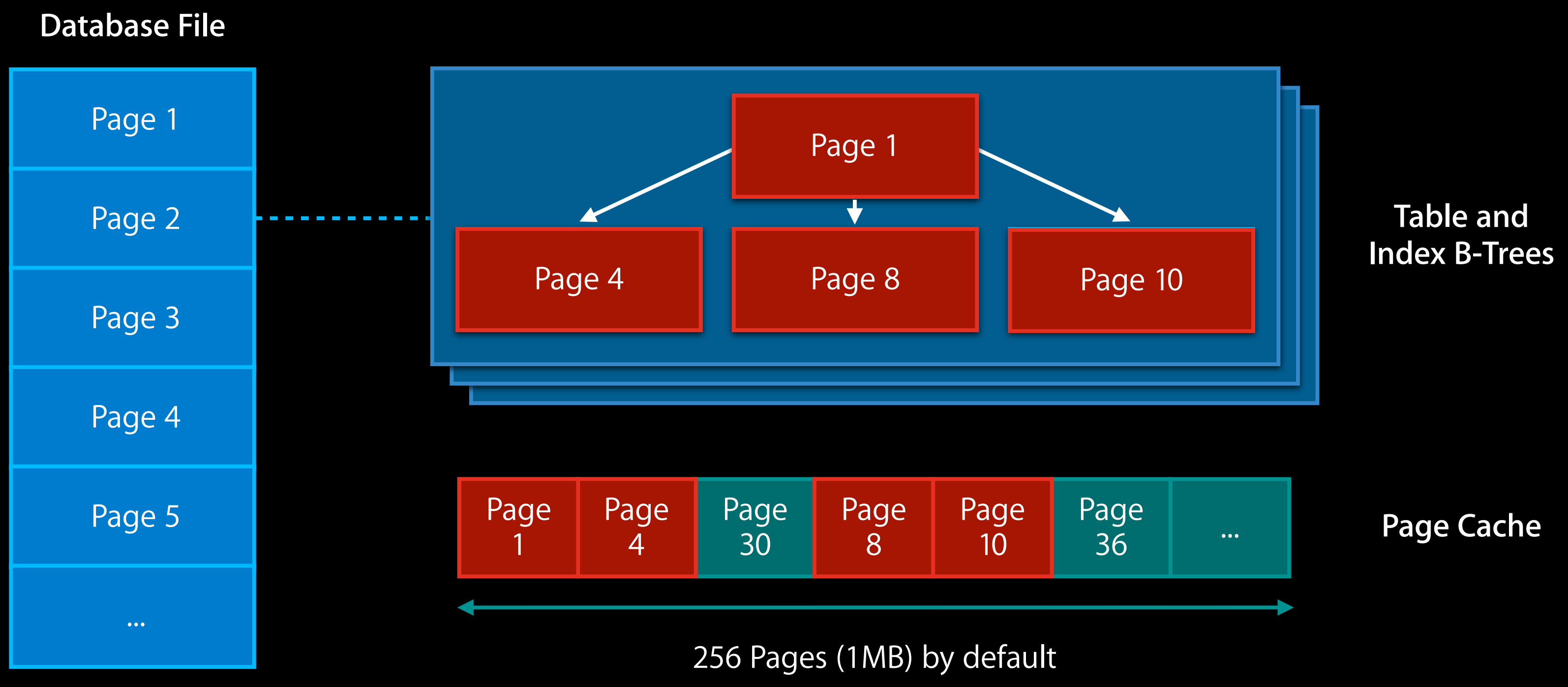
Query plan with joins

```
sqlite> CREATE INDEX TrackAlbumIDOrderIndex ON Track(AlbumID, AlbumOrder);
sqlite> EXPLAIN QUERY PLAN
   ...> SELECT * FROM Track JOIN Artist ON ArtistID=Artist.ROWID
   ...> WHERE AlbumID=? ORDER BY AlbumOrder;
0|0|TABLE Track WITH INDEX TrackAlbumIDOrderIndex ORDER BY
1|1|TABLE Artist USING PRIMARY KEY
```

- Same as previous queries, but join onto Artist by logarithmically looking up Artist’s primary key

SQLite

Understanding the page cache



Database File

Page 1

Page 2

Page 3

Page 4

Page 5

...

Page 1

Page 4

Page 8

Page 10

Table and Index B-Trees

Page 1

Page 4

Page 30

Page 8

Page 10

Page 36

...

Page Cache

256 Pages (1MB) by default

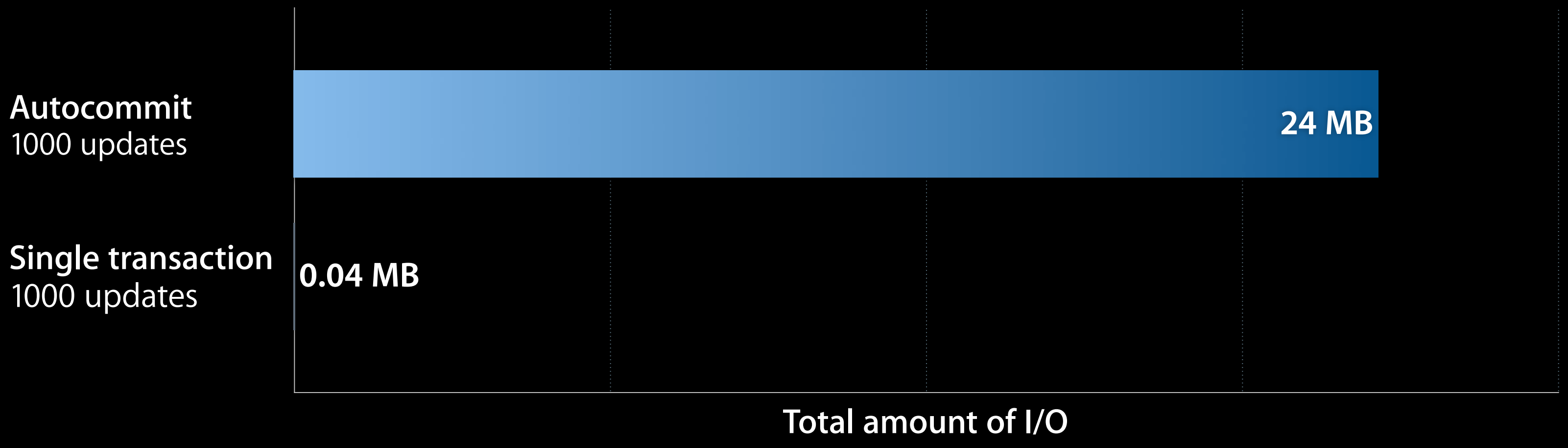
SQLite

Paged I/O guidelines

- I/O is done in page-sized increments
    - Surround batch INSERTs or UPDATEs with transactions
- Don’t store large arbitrarily sized binary objects in the database
    - Small (< 2k or so) BLOBs are fine
    - Large BLOBs work, but aren’t optimal
        - Crowd out other data from the page cache
    - Write traffic is doubled because of transactions
    - Consider storing pointers to the filesystem in the DB instead

SQLite

Using transactions



Databases

Summary

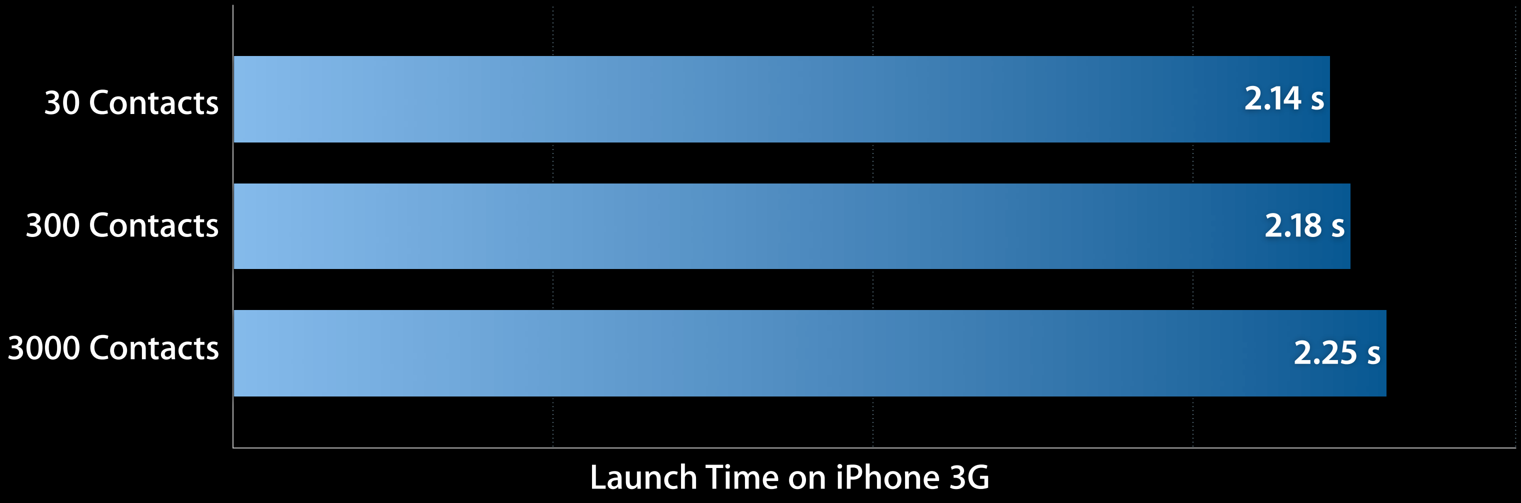
- Use CoreData if possible
- Find problematic queries using sqlite3_profile
- Understand problematic queries with EXPLAIN QUERY PLAN
- Use transactions where appropriate
- Scale gracefully with large data sets
- Additional resources
    - Core Data Programming Guide
    - Introduction to SQLite by D. Richard Hipp (on YouTube)
    - SQLite Documentation on SQLite.org

Scaling

Scaling

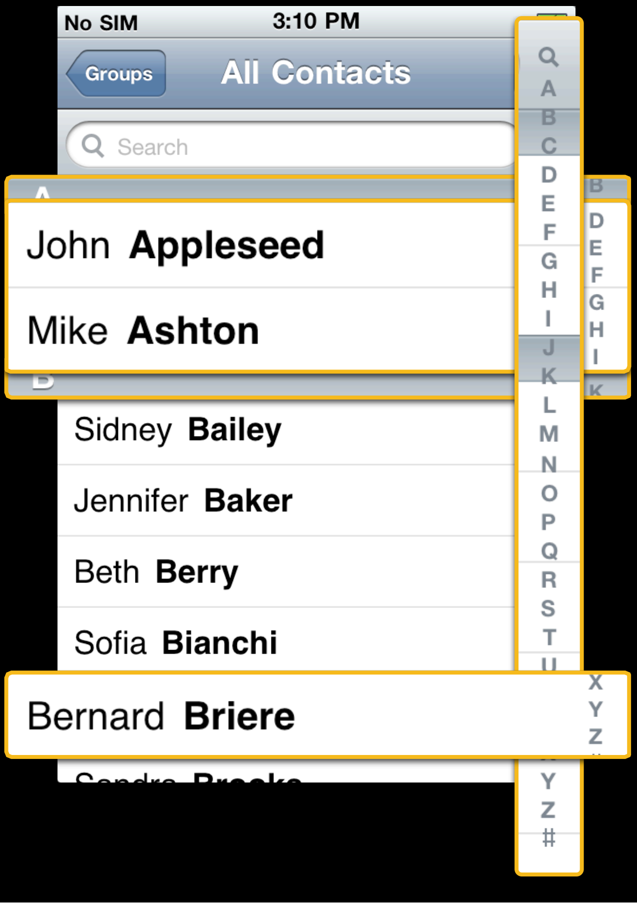
- Applications should scale gracefully in the face of large data sets
- Think about the minimum amount of work needed to make critical methods fast
- Case study: Contacts

Scaling in Contacts



Scaling in Contacts

Make critical methods fast



Loading sections

```
– numberOfSectionsInTableView:
```

```
– tableView:titleForHeaderInSection:
```

```
– tableView:numberOfRowsInSection:
```

Loading the index bar

```
– tableView:sectionIndexTitlesForTableView:
```

Loading visible cells

```
– tableView:cellForRowAtIndexPath:
```

Scaling in Contacts

Loading sections quickly

- Naive: load entire data set and group afterwards
- Better: cache the section counts
    - Tricky to do right for all localizations: see DerivedProperty example
    - Contacts uses a separate table for section counts, maintained by triggers
    - CoreData users get this for free

```
-[NSFetchedResultsController initWithFetchRequest:(NSFetchRequest *)fetchRequest
                             managedObjectContext:(NSManagedObjectContext *)context
                               sectionNameKeyPath:(NSString *)sectionNameKeyPath
                                        cacheName:(NSString *)name]
```

Scaling in Contacts

Loading the index bar quickly

- Approach 1: always loads the same index bar
  - Contacts does this: always loads A–Z and #
- Approach 2: change the index bar titles based on section count
  - Should be fast if section loading is fast

Scaling in Contacts

Loading visible cells quickly

- Do not table scan just to retrieve one cell's worth of information
    - Bring in data in small chunks
    - LIMIT/OFFSET is not particularly efficient in SQLite, but works if iterating over a small index
    - Can also use scrolling cursor method
- Make sure that proper indices are in place to avoid sorting a transient table

```
SELECT VisibleName
FROM People
ORDER BY LastName, FirstName
LIMIT 20
OFFSET 0;
```

```
CREATE INDEX
    PeopleLastFirstOrder
    (LastName, FirstName);
```

Scaling

Summary

- Test and profile apps with different data set sizes
- Only bring in the data necessary to display a view
    - Avoid bringing in the entire data set at view loading time

Summary

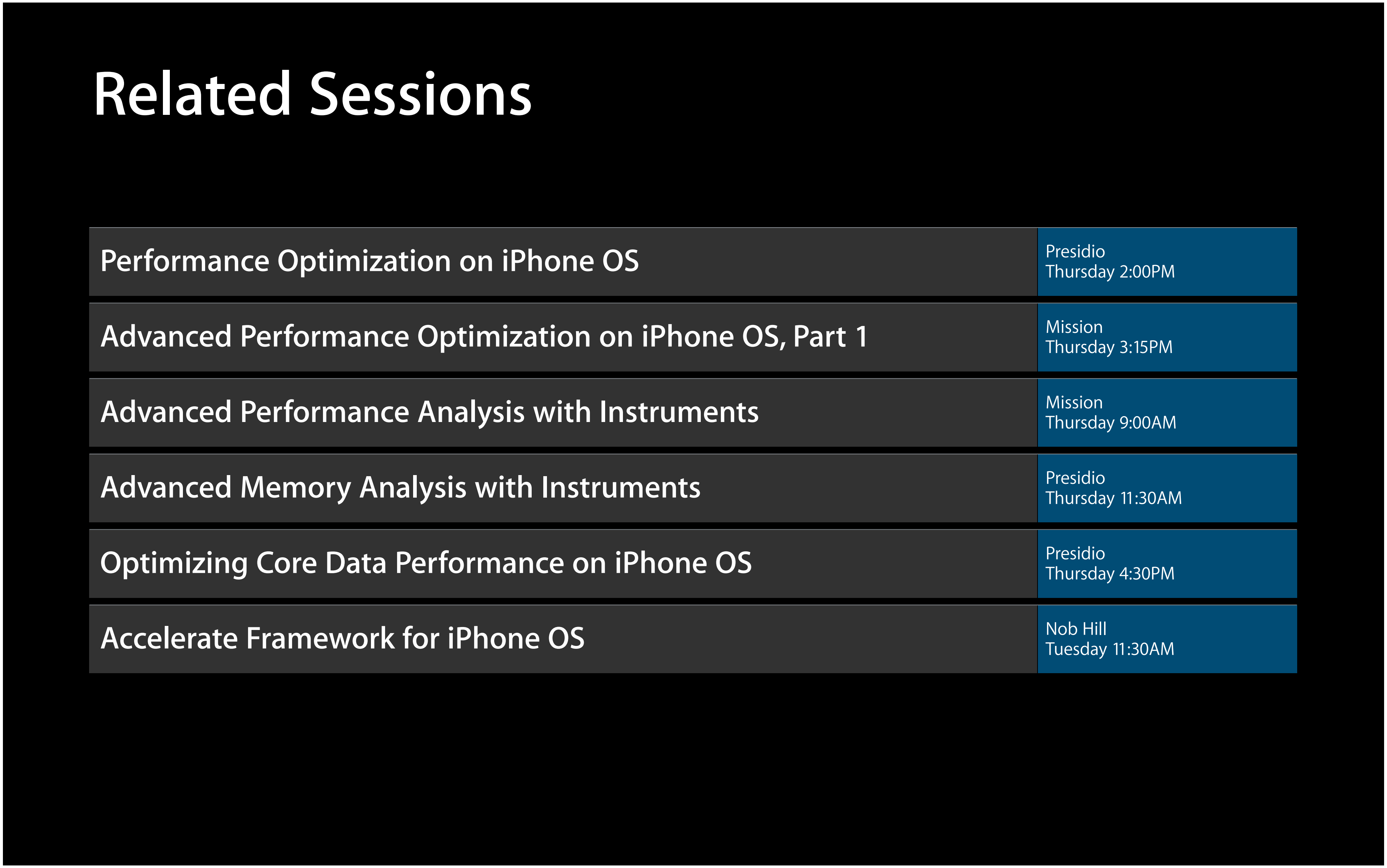
- Reduce dirty memory usage
- Adhere to Foundation API best practices
- Profile filesystem and database activity
- Test apps on different devices and varying sizes of data sets

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Documentation
iPhone OS Programming Guide
http://developer.apple.com/iphone

Apple Developer Forums
http://devforums.apple.com



Related Sessions

Performance Optimization on iPhone OS	Presidio Thursday 2:00PM
Advanced Performance Optimization on iPhone OS, Part 1	Mission Thursday 3:15PM
Advanced Performance Analysis with Instruments	Mission Thursday 9:00AM
Advanced Memory Analysis with Instruments	Presidio Thursday 11:30AM
Optimizing Core Data Performance on iPhone OS	Presidio Thursday 4:30PM
Accelerate Framework for iPhone OS	Nob Hill Tuesday 11:30AM

WWDC10

[redacted]