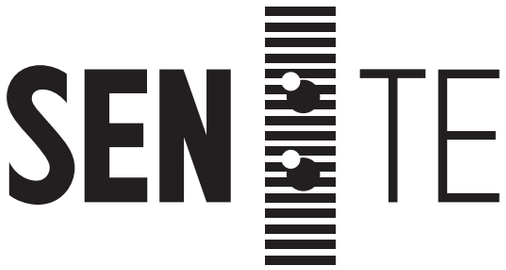
Performance, optimization and why it matters

Marco Scheurer

Lausanne, Switzerland / Shanghai, China



marco@sente.ch

WeChat: phink0

Performance

- Why?
- What?
- How?
- When?

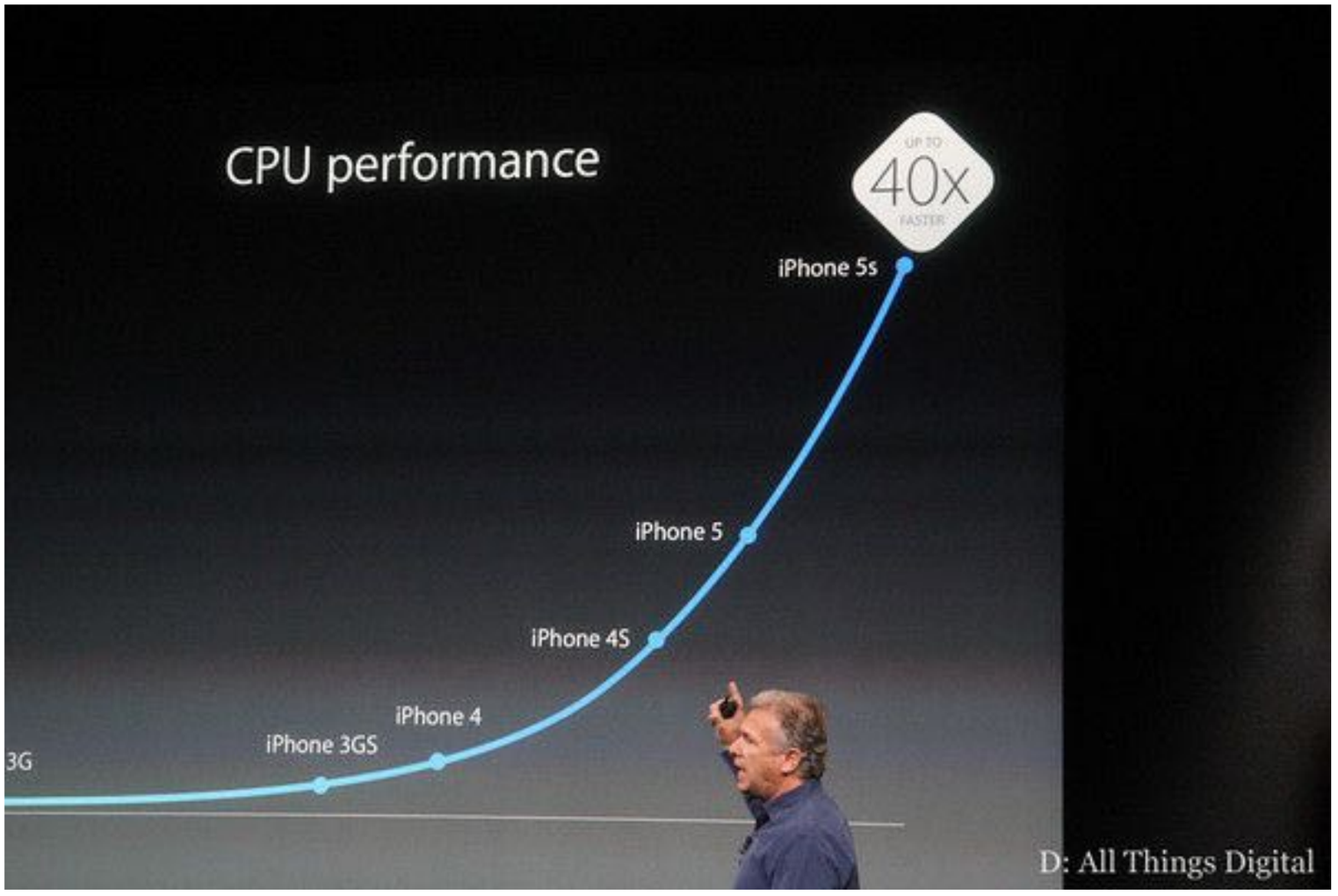
Why optimizing performance?

- Better user experience

Optimizing performance is…

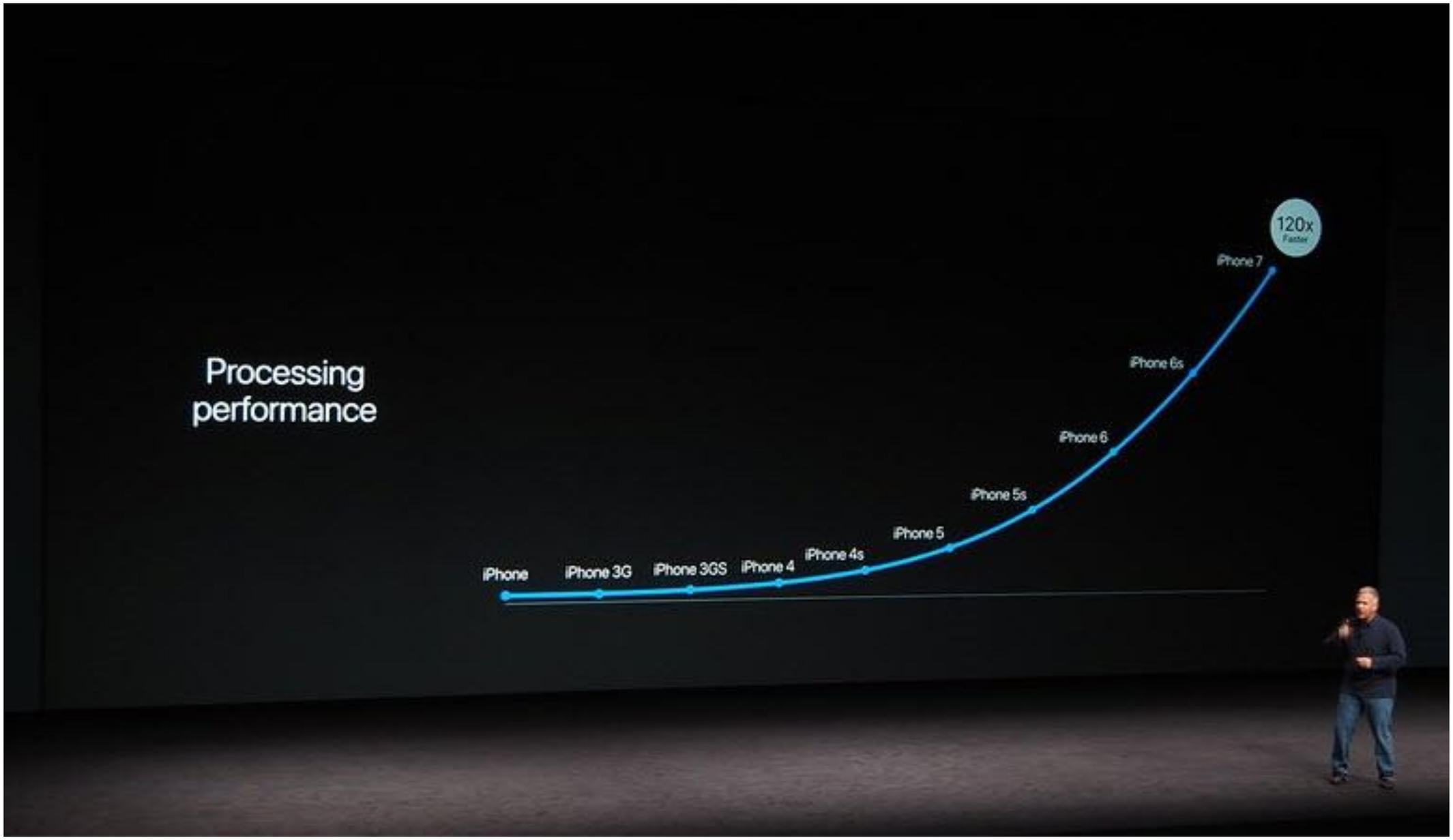
- Risky
  “If it ain’t broke, don’t fix it”
- Time consuming
  “I don’t have time for performance work”
- Complicated
  “I don’t know where to start or what to do”
- Better solved with faster hardware
  “Hardware is cheap, programmers are expensive”

Hardware



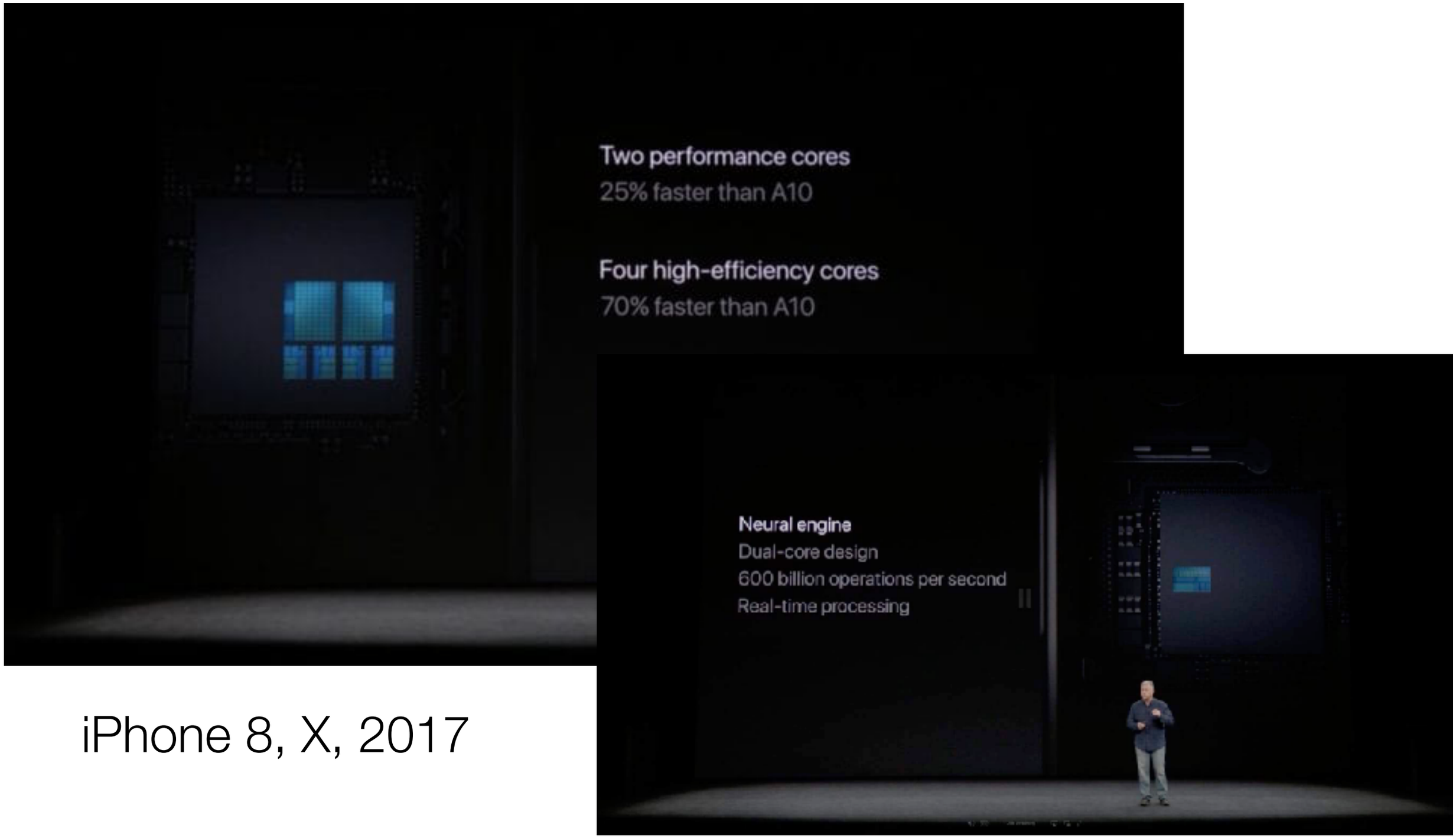
iPhone 5S, 2013

Hardware



iPhone 7, 2016

Hardware



iPhone 8, X, 2017

Hardware

- iPhone in 2017 ~200x faster than in 2007 but:
  - Screen with 25x more pixels
  - Augmented reality
  - 240 fps 1080p video capture
  - Animated emojis
  - etc.

Not everybody has an A11

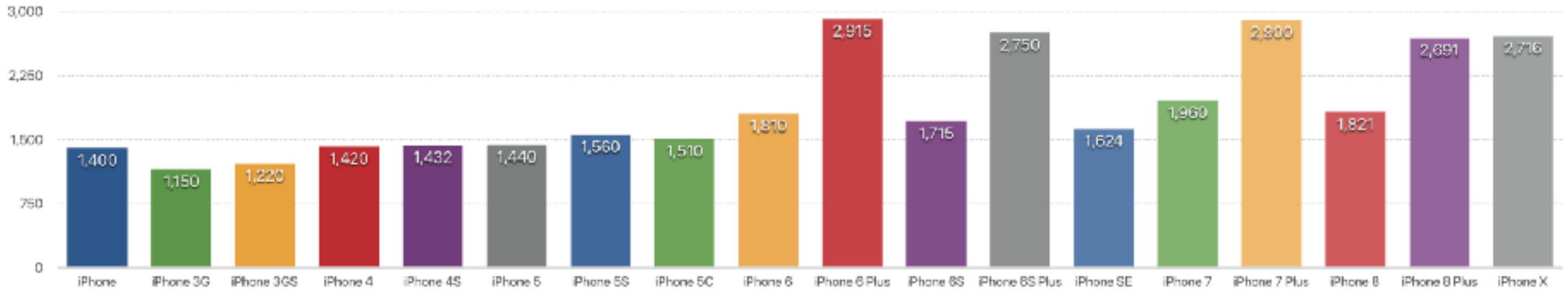
Rank	Device	Usage Share [?]
1	iPhone 7	14.57%
2	iPhone 6S	11.34%
3	iPhone 7 Plus	9.14%
4	iPhone 6	8.26%
5	iPhone X	6.09%
6	iPhone 8	5.02%
7	iPhone 8 Plus	4.88%
8	iPhone SE	4.50%
9	iPhone 6S Plus	4.18%
10	iPad Air 2	4.06%
11	iPhone 5S	3.65%
12	iPad 5	3.54%
13	iPad Mini 2	2.98%
14	iPad Air	2.77%

https://data.apteligent.com/ios/devices/

Device	Device Usage
iPhone 6	13.6%
iPhone 6s	11.0%
iPhone 7	8.9%
iPhone 5s	6.1%
iPhone 8	5.0%
iPhone 7+	4.6%
iPhone se	4.2%
iPhone X	4.2%
iPad Air 2	4.1%
iPhone 8+	4.0%
iPhone 6+	3.6%
iPad Air	3.4%
iPhone 6s+	3.2%
iPad 4G	2.9%
iPod touch 4G	2.7%
iPad 2G	1.9%
iPad Mini 2	1.8%
iPhone 5	1.8%

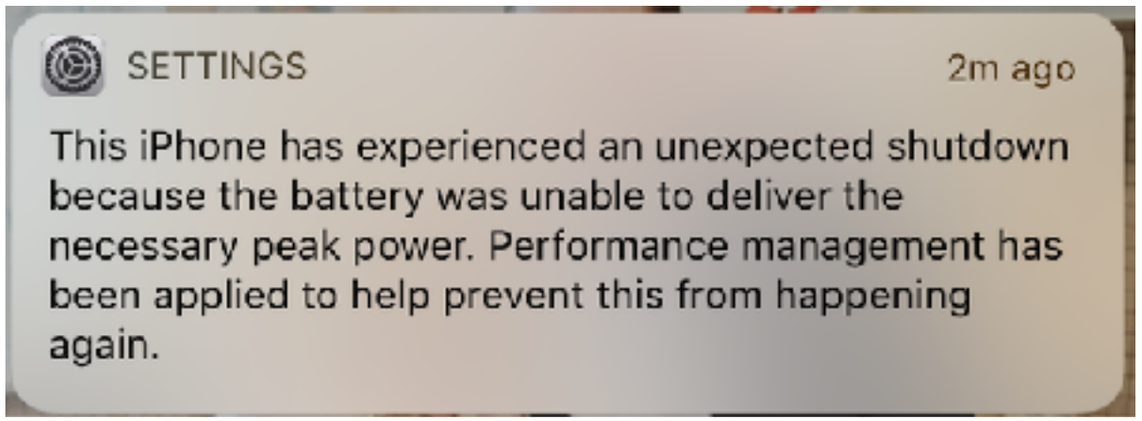
https://david-smith.org/iosversionstats/

Battery



- 1400 mAh to 2700 mAh
- Almost no progress, related to device size
- (some Android phones have 5000 mAh)
- iPhone X: 2716 mAh = 10.35 Wh (3.8 V)

Battery



~ Power consumption (mW)

CPU	500 - 2000
Display	400 - 1000
Cell radio	800
WiFi	200
Bluetooth	40
GPS	200
Microphone	100
Gyroscope	100
Accelerometer	20

https://qnovo.com/understanding-power-usage-in-a-smartphone/
https://www.usenix.org/legacy/event/atc10/tech/full_papers/Carroll.pdf

1000 mW on iPhone X drains the battery in 10 hours

Better user experience

1. No crash (including killed by the watchdog)
2. Responsive application
3. Good battery life

Activity Watchdog

- The amount of time the watchdog gives you is not formally documented
- Was more an issue in 2007 than today
- Still happens…

Launch	20 s
Resume	10 s
Suspend	10 s
Quit	5 s
Operation	10 min

Incident Identifier: AD5E7187-C852-4873-8717-89150CCCCF1E
CrashReporter Key: aad64d5368f33ab8bd00f10c7e7414b7067d35ec
Hardware Model: iPhone6,2
Process: **WeChat** [4240]
Path: /private/var/containers/Bundle/Application/B2CB8F37-61B9-43D3-8F29-C0FBD485E2D3/WeChat.app/WeChat
Identifier: com.tencent.xin
Version: 6.6.2.34 (6.6.2)
Code Type: ARM-64 (Native)
Role: Foreground
Parent Process: launchd [1]
Coalition: com.tencent.xin [1154]
Date/Time: 2018-04-02 12:23:36.3672 +0800
Launch Time: 2018-04-02 12:20:10.5635 +0800
OS Version: iPhone OS 11.2.6 (15D100)
Baseband Version: 8.30.01
Report Version: 104
Exception Type: EXC_CRASH (SIGKILL)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Exception Note: EXC_CORPSE_NOTIFY
Termination Reason: Namespace SPRINGBOARD, Code 0x8badf00d
Termination Description: SPRINGBOARD, process-exit watchdog transgression: com.tencent.xin exhausted real (wall clock) **time allowance of 5.00 seconds** | | ProcessVisibility: Background | ProcessState: Running | WatchdogEvent: process-exit | WatchdogVisibility: Background | WatchdogCPUStatistics: (| "Elapsed total CPU time (seconds): 2.770 (user 2.770, system 0.000), 28% CPU", | "Elapsed application CPU time (seconds): 0.176, 2% CPU" |)

Code 0xc00010ff

- “Killed by the operating system in response to a thermal event”
- Another reason to make apps run efficiently

Where to start? Measuring Performance

- Guessing often does not work
- Measure - Change - Measure
- Use measuring tools

Measuring Tools

- NSLog / print
- Instruments
- Simulator
- measure()

NSLog/print

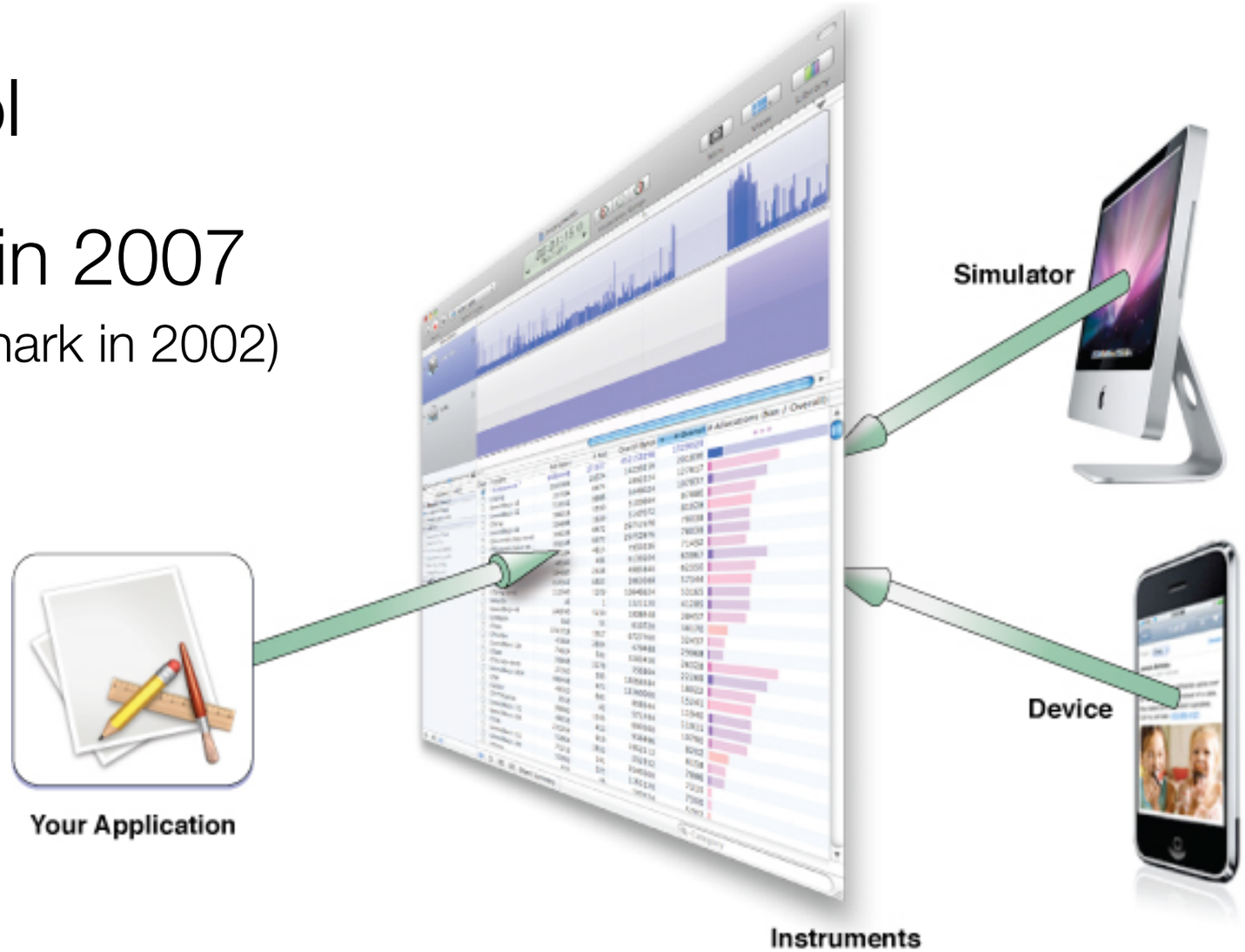
```
NSDate *startTime = [NSDate date];
...
NSLog (@"Elapsed time: %.3f", -[startTime timeIntervalSinceNow]);
```

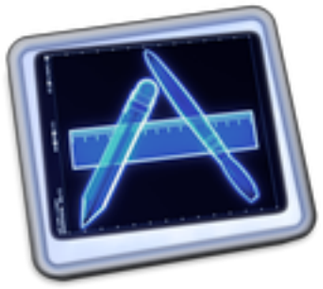
```
double start = CFAbsoluteTimeGetCurrent();
...
double elapsedTime = CFAbsoluteTimeGetCurrent() - start;

NSLog (@"Elapsed time: %.3f", elapsedTime);
```

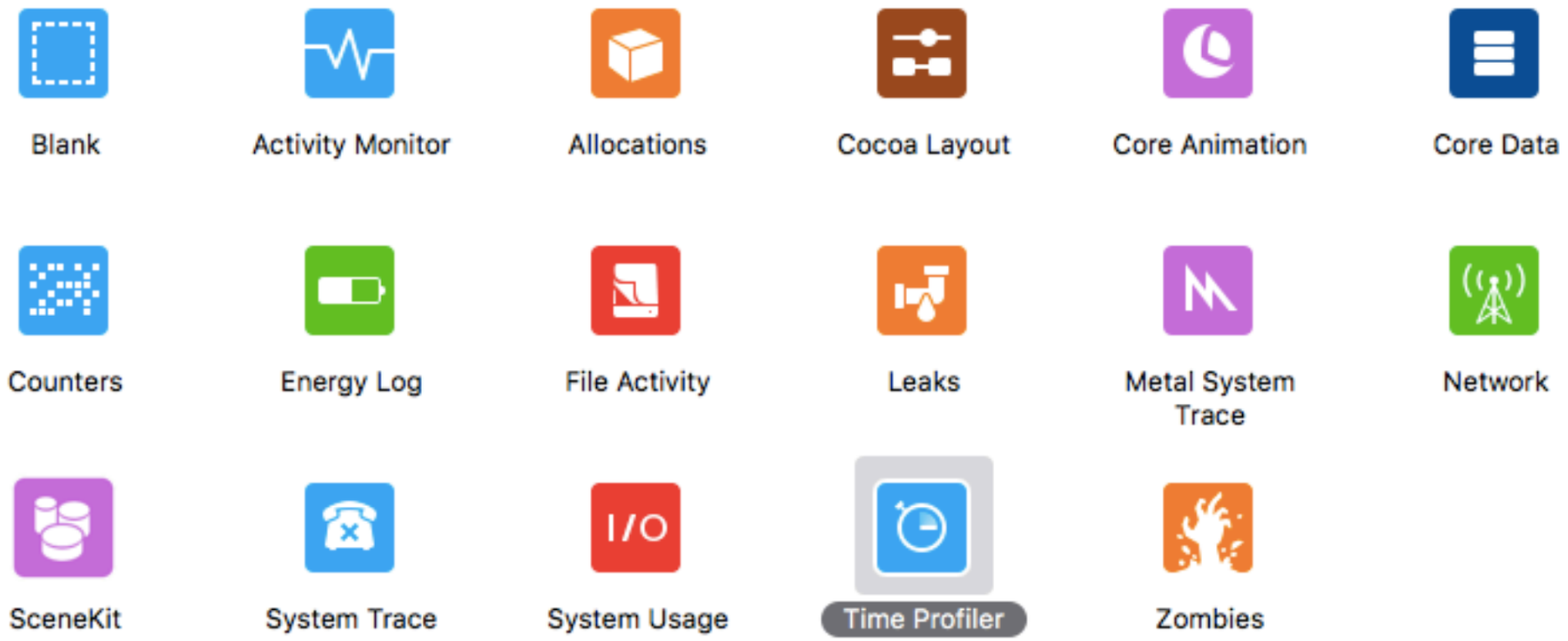
Instruments

- Profiling tool
- Introduced in 2007
  (as Shikari and Shark in 2002)





Instruments



Demo

Simulator

- Software simulator ≠ hardware emulator

Simulator	iPhone	5S	X
i386	ARM	ARM	ARM
2.5 GHz	400 MHz	1.3 GHz	2.4 GHz
Unlimited	128 MB	1GB	3GB

- Measure and test on oldest supported device (5S)
- Simulator OK to check memory usage

XCTestCase: measure()

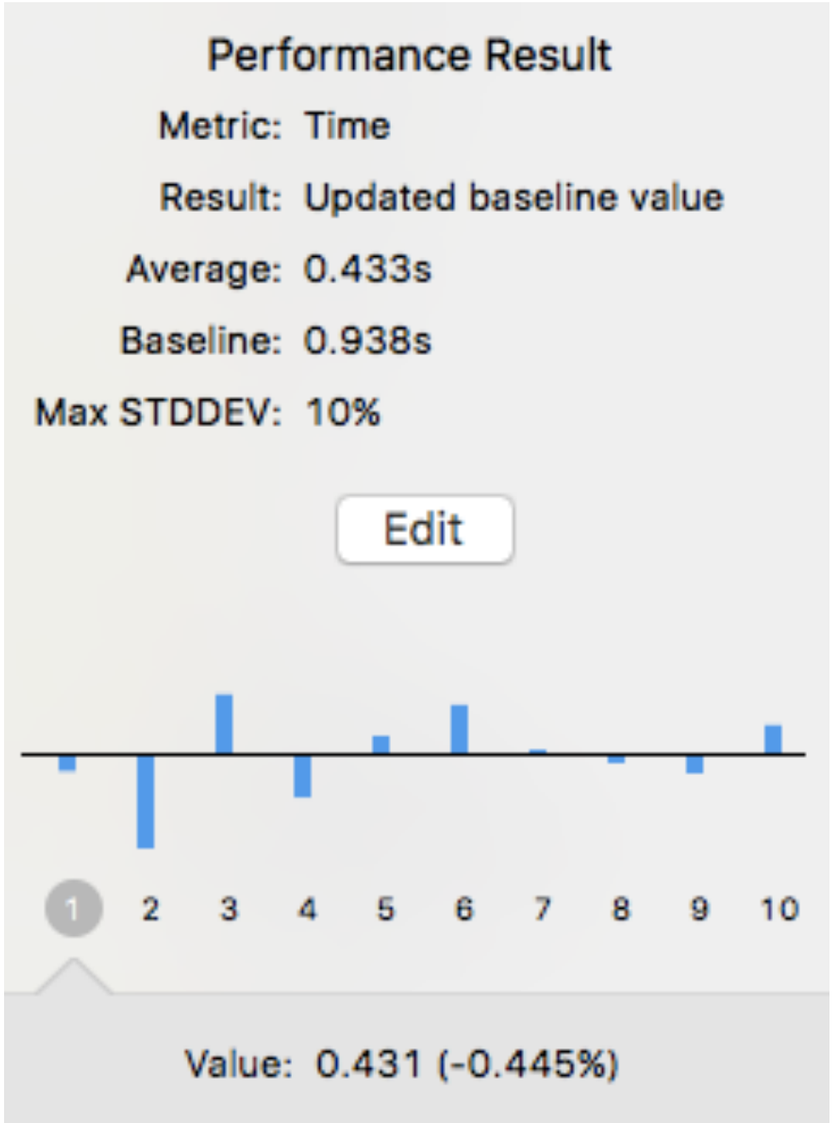
- Objective C

```
70
- (void)testPerformanceExample {
    [self measureBlock:^{
        [self loopContiguous];
    }];
75 }
```

Time: 0.433 sec (54% better)

- Swift

```
func testPerformanceExample() {
    self.measure {     }
}
```



measure ()

- Set or change the baseline after each run
- Also startMeasuring() / stopMeasuring()
- Match the baseline using device types
- Many tests will slow the suite: run separately
- Demo

1. No crash (including kill by the watchdog)
2. Responsive application
3. Good battery life

1. No Crash

Memory Watchdog

- Watches memory pressure
- Issues low memory warning
- Instant termination of application

Memory

- Swap: no
- Virtual memory: yes
- Memory usage also impacts time

Memory Usage

- Avoid
  - spikes
  - leaks
  - abandoned memory
  - zombies

Memory Spikes

- Be smart with autorelease
- Process large quantities of data in batches
  - Nested autorelease pools
- Demo

Memory Leaks

- Allocated memory that is inaccessible
- Unbalanced retain/release
- Forget to release property's original value
- Leaks Instrument examines heap for leaked memory, identify allocation
- Xcode: Build and Analyze

Abandoned Memory

- Wasted memory
- Accessible but never used
- Memory should not grow without bounds
- Detect with Allocation Instrument

Zombies

- Messages send to deallocated object
- Detect with Zombies instrument
- Demo

1. No crash (including kill by the watchdog)
2. Responsive application
3. Good battery life

2. Responsive applications

Time

- Faster code
- makes applications more responsive
- uses less power

Time

- C, ARM code generation
- Objective C
- Frameworks
- File system
- Animation
- Swift

C and ARM Code Generation

- ARM v6 (2G, 3G)
- ARM v7 (3GS, 4)
- ARM 64 (5 … X)
- -Onone for Debug
- -Os vs -O3 vs -Ofast for Release
  (and Profiling and Performance testing)

Code Generation

```
#define MATRIX_SIZE 5000
static double array[MATRIX_SIZE][MATRIX_SIZE];

double sum = 0.0;
for (int j = 0; j < MATRIX_SIZE; j++) {
    for (int i = 0; i < MATRIX_SIZE; i++) {
        array[i][j] = i;
        sum += array[i][j];
    }
}
```

0.55 s

Code Generation

```
#define MATRIX_SIZE 5000
static double array[MATRIX_SIZE][MATRIX_SIZE];

double sum = 0.0;
for (int j = 0; j < MATRIX_SIZE; j++) {
    for (int i = 0; i < MATRIX_SIZE; i++) {
        array[i][j] = i;
        sum += array[i][j];
    }
}
```

0.55 s

```
double sum = 0.0;
 for (int i = 0; i < MATRIX_SIZE; i++) {
    for (int j = 0; j < MATRIX_SIZE; j++) {
        array[i][j] = i;
        sum += array[i][j];
    }
}
```

0.28 s

- Speedup: ~2x

Objective C

- MYTH: Dynamic dispatch in Objective C is slow
- objc_msgSend() is very, VERY FAST
- …and very, VERY USEFUL
- IMP caching to remove overhead:
  - Loop with known receiver
  - Rarely useful

objc_msgSend()

```objc
#define COUNT 100 * 1000 * 1000
- (uint64_t) withMessageSend
{
    NSNumber *n = [NSNumber numberWithInt:42];
    uint64_t r = 0;
    for (int i = 0; i < COUNT; i++) {
        r += [n intValue];
    }
    return r;
}
```

1.37 s

```objc
- (uint64_t) withIMPCaching
{
    NSNumber *n = [NSNumber numberWithInt:42];
    SEL s = @selector(intValue);
    int (*f) (id, SEL, ...) = (int (*) (id, SEL, ...))[n methodForSelector:s];
    uint64_t r = 0;
    for (int i = 0; i < COUNT; i++) {
        r += (int)f(n,s);
    }
    return r;
}
```

0.92 s

- Speedup 1.48x but…
- > 200 millions objc_msgSend() / s on iPhone 5S

Frameworks

- Foundation
- Grand Central Dispatch
- Accelerate

Foundation

- NSMutableArray, NSMutableString
  - insert O(N), O(1) at begin and end
- NSMutableDictionary, NSMutableSet
  - lookup O(1) with good hash function
- NSMutableIndexSet
  - Store integers without NSNumber
- NSMutableOrderedSet, NSCountedSet…

Foundation

```
[array indexesOfObjectsWithOptions:NSEnumerationConcurrent
     passingTest:^(id obj, NSUInteger idx, BOOL *stop) {
                    return [obj test:value];
     }
]
```

- Demo

Grand Central Dispatch

- Concurrent programming
- Multicore or single core
- Easier and lighter than threads
- Move long processes off the main thread
- dispatch_async()
- dispatch_apply() to parallelize loops

Grand Central Dispatch

```
[resultLabel setText:@""];
[progressView startAnimating];

dispatch_queue_t queue = dispatch_get_global_queue(QOS_CLASS_USER_INITIATED, 0);
dispatch_queue_t main_queue = dispatch_get_main_queue();
```

```
dispatch_async (queue,^{
    float result = [self longDotWithAccelerate];
    dispatch_async(main_queue, ^{
      [progressView stopAnimating];
      [resultLabel setText:[NSString stringWithFormat:@"%.1f", result]];
    });
});
```

Accelerate

- ARM SIMD architecture: NEON
  - 16 8-bit operations at the same time
  - NEON can decode MP3 on 10MHz CPU in real time
- A7 (iPhone 5S)
  - 2 cores
  - 31 general purpose 64-bit registers
  - 32 128-bit vector register
- A11 (iPhone 8, iPhone X)
  - 6 cores
  - Neural Engine, 600 billions ops/s

Accelerate

- 2600 APIs for hardware accelerated math
- vDSP (Digital Signal Processing)
  - Fourier transforms, …
- BLAS (Basic Linear Algebra Subprograms)
  - Matrix product, …
- LAPACK (Linear algebra)
  - Solving system of linear equations
- vImage, vForce, BNNS, Compression…

https://developer.apple.com/wwdc17/711

Accelerate

```
float dotProduct = 0.0;
for (int i = 0; i < VECTOR_SIZE; i++) {
    dotProduct += a[i] * b[i];
}
```

```
float dotProduct = 0.0;
vDSP_dotpr(a, 1, b, 1, &dotProduct, VECTOR_SIZE);
```

- Makes using NEON and parallel computing easy

Accelerate

```
float dotProduct = 0.0;
for (int i = 0; i < VECTOR_SIZE; i++) {
    dotProduct += a[i] * b[i];
}
```

0.958 s

```
float dotProduct = 0.0;
vDSP_dotpr(a, 1, b, 1, &dotProduct, VECTOR_SIZE);
```

0.023 s

- Vector size 1024
- 100 000 times on iPhone 5S
- Speedup: 40x

Frameworks

- Demo

File system

- Access to Flash memory is relatively slow
- Speed vary a lot from device to device
- For large files use memory mapped files

```
NSDataReadingOptions.DataReadingMappedAlways
```

- Long I/O off the main thread

File System

- I/O of small amount of data:
  - NSUserDefaults
  - Property Lists
  - JSON
  - NSKeyedArchiver
- I/O of large amount of data:
  - Incremental formats
  - Databases (CoreData)

Property Lists

- Binary Format 2-3x faster than XML
- NSPropertyListSerialization
- Slow:

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

- Fast:

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

NSJSONSerialization

- Fast
- Portable
- Convenient

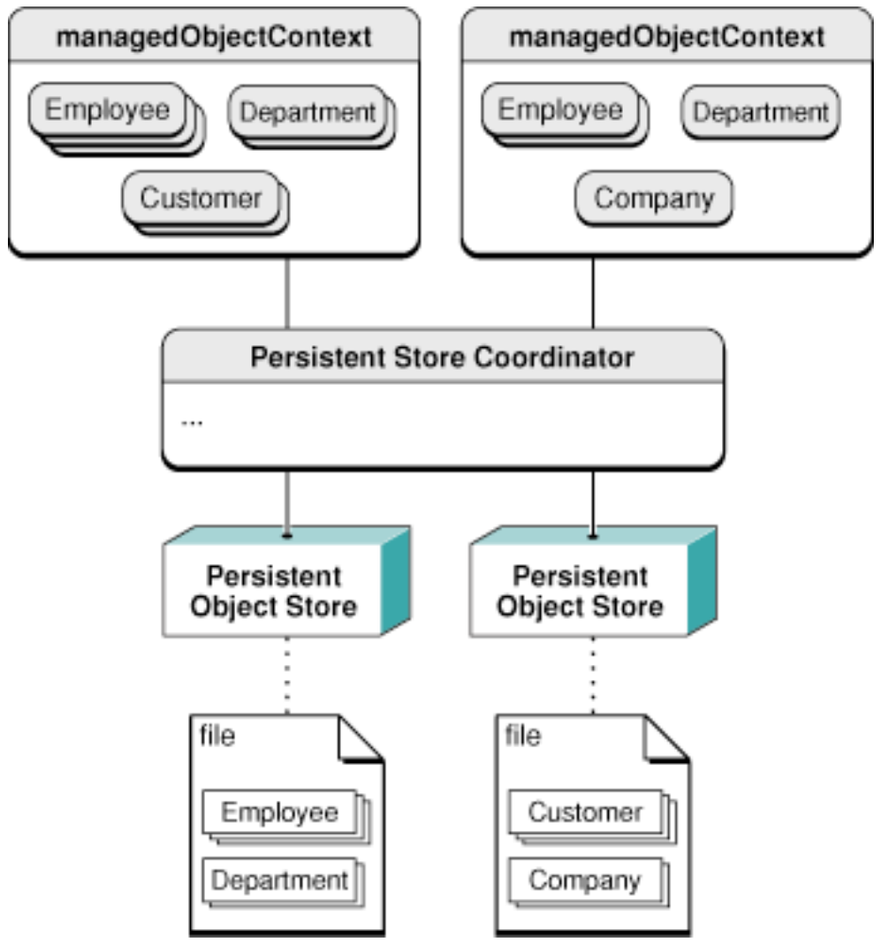
```
NSError *error = nil;
NSArray *json = [NSJSONSerialization JSONObjectWithData:data
                                                options: NSJSONReadingMutableContainers
                                                  error: &error];
```

NSKeyedArchiver

- For all objects supporting NSCoding
- Supports versioning
- Not an incremental file format
- Heavy, slow
- Use NSJSONSerialization for “property list”-like values

Database

- Use CoreData… or not
  - fmdb
  - SQLIte.swift
  - … SQL wrappers sometime too much overhead
- Do not import large quantity of data...
- Add object stores instead



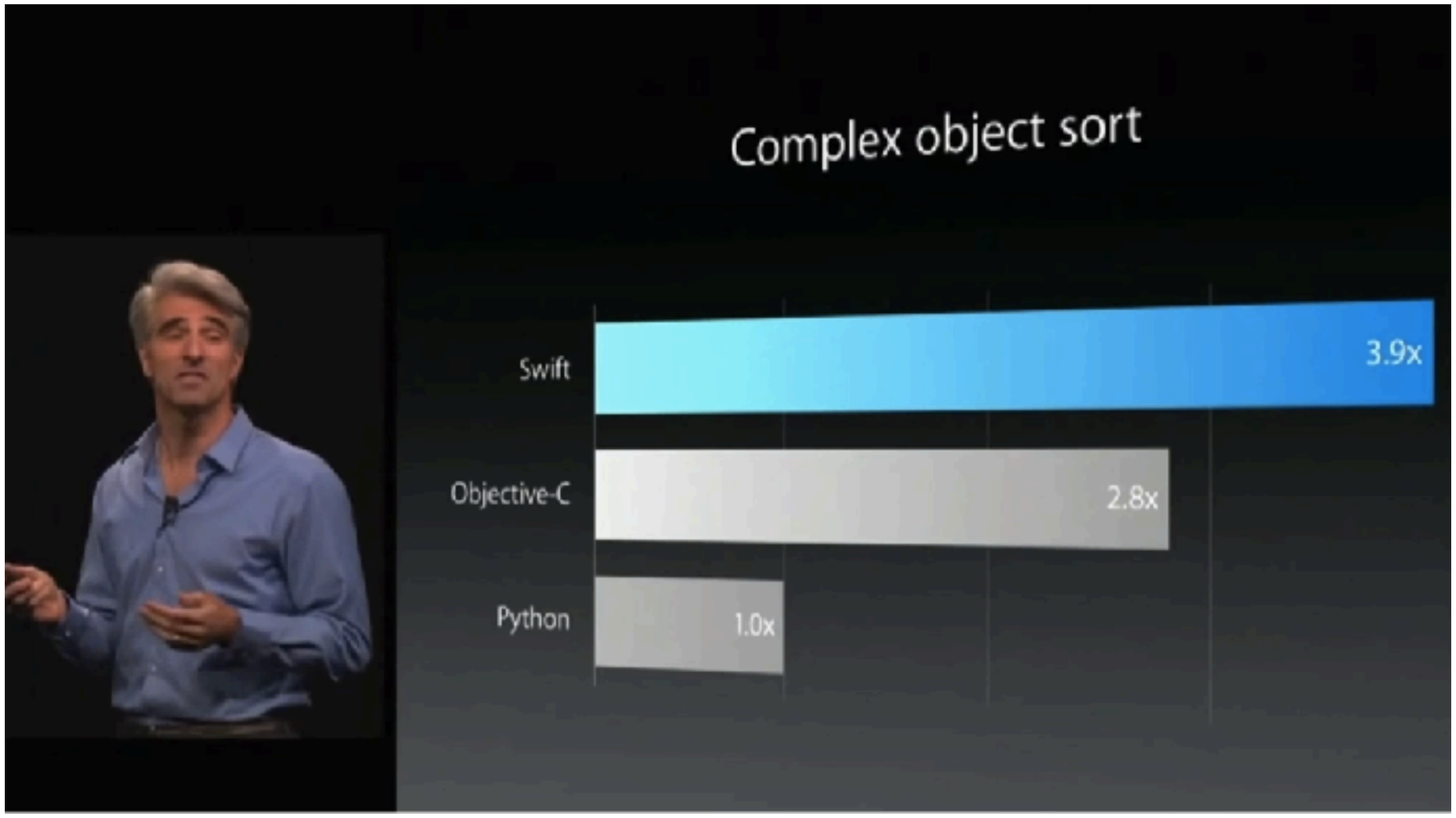
Database Search

- Unicode string comparisons are expensive
- Use derived attribute (without accents)
- Prefer prefix searching
- Use <= and < instead of BEGINSWITH

Animation

- Scrolling at 60 fps
  - Reuse identifier
  - Reuse formatters
  - Opaque views
  - UINib

Swift

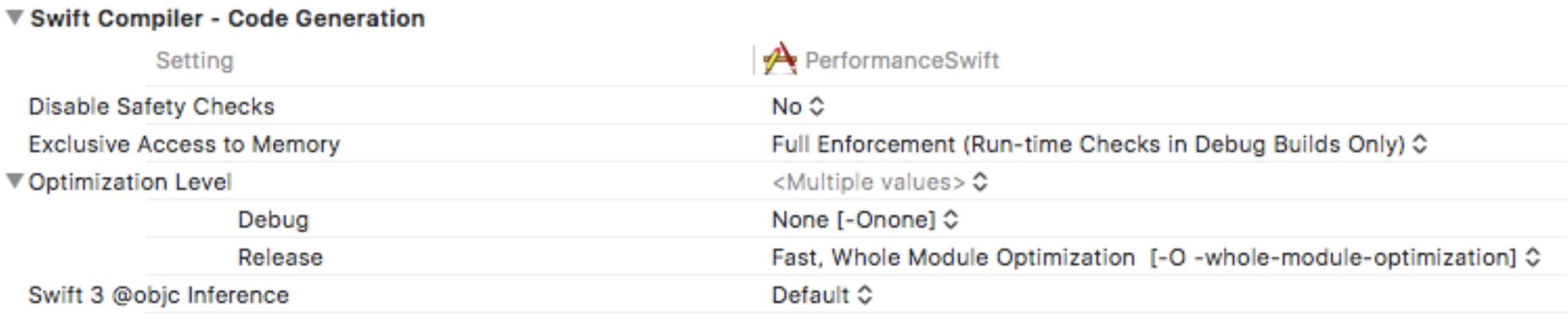


- Who in the world compares speed with Python?

Swift

- 1.4 times faster would be nice
- May be true on “complex object sort"
- Not so sure for the real world

Swift Compilation



- When optimizing:
  - Release, not Debug
  - -O -whole-module-optimization
  - Disable runtime safety checks

Inlining

- Automatic inlining by the compiler
- Inline all the things with @inline(__always)
- Force inlining with @_transparent
- Public interface @_inlineable

Source Optimizations

- struct vs class
- final class
- private
- enum vs string

struct vs class

- stack vs heap
- value vs reference
- copy on write
- dynamic dispatch
- reference counting

More Source Optimizations

- map/reduce vs for loop
- Generics
- Protocols
- inout
- Memory allocation
- Unchecked operators (&+)

map vs for loop

```
var array = [Float](repeating: 0, count: 10_000_000)

func useMap () {
    var output = array.map({ (element) -> Float in
        return element * 5
    })
}

 func useForLoop () {
     var output = [Float]()
     output.reserveCapacity(array.count)
     for element in array {
         output.append(element * 5)
     }
}
```

map vs for loop

```
var array = [Float](repeating: 0, count: 10_000_000)

func useMap () {
    var output = array.map({ (element) -> Float in
        return element * 5
    })
}
```

0.10 s

```
 func useForLoop () {
     var output = [Float]()
     output.reserveCapacity(array.count)
     for element in array {
         output.append(element * 5)
     }
}
```

0.08 s

- Speedup: 1.25x
- Not much but still surprising

Generics

- Optimizer will try to specialize the code if the generic definition is visible
- Put generics declaration in module where used
- Or use -whole-module-optimization

Protocols

- struct can conforms to protocols
- Benefits over class get lost
- references to Protocol Witness Table, Value Witness Table
- 3-words or less inline otherwise on heap

Apple, *Understanding Swift Performance, WWDC 2016*
https://developer.apple.com/videos/play/wwdc2016/416/

inout

- Parameters passed to functions:
  1. Function is called: value of argument is copied.
  2. Body of the function: copy is modified.
  3. Function returns: copy's value assigned to original argument.
- *Can* be optimized as a call by reference

Allocation

```
func doSomething () {
    var s = MyStruct ()
    DispatchQueue (label:"q").async {
        s.perform ()
    }
}
```

- Heap or Stack ?

Allocation

```
func doSomething () {
    var s = MyStruct ()
    DispatchQueue (label:"q").async {
        s.perform ()
    }
}
```

- Heap !

Swift

- Language, compiler and optimizer keep changing
- struct, class, protocol, generics…
  - complicated?
  - low level premature optimization?
- Compiler is slow

1. No crash (including kill by the watchdog)
2. Responsive application
3. Good battery life

3. Battery Life

Energy

- 4G, WiFi, Bluetooth, GPS: 2000 mW
- CPU + GPU: 1000 mW
- Screen: 400 mW

Radios and sensors

- Network
- CoreLocation
- CoreMotion

Radios

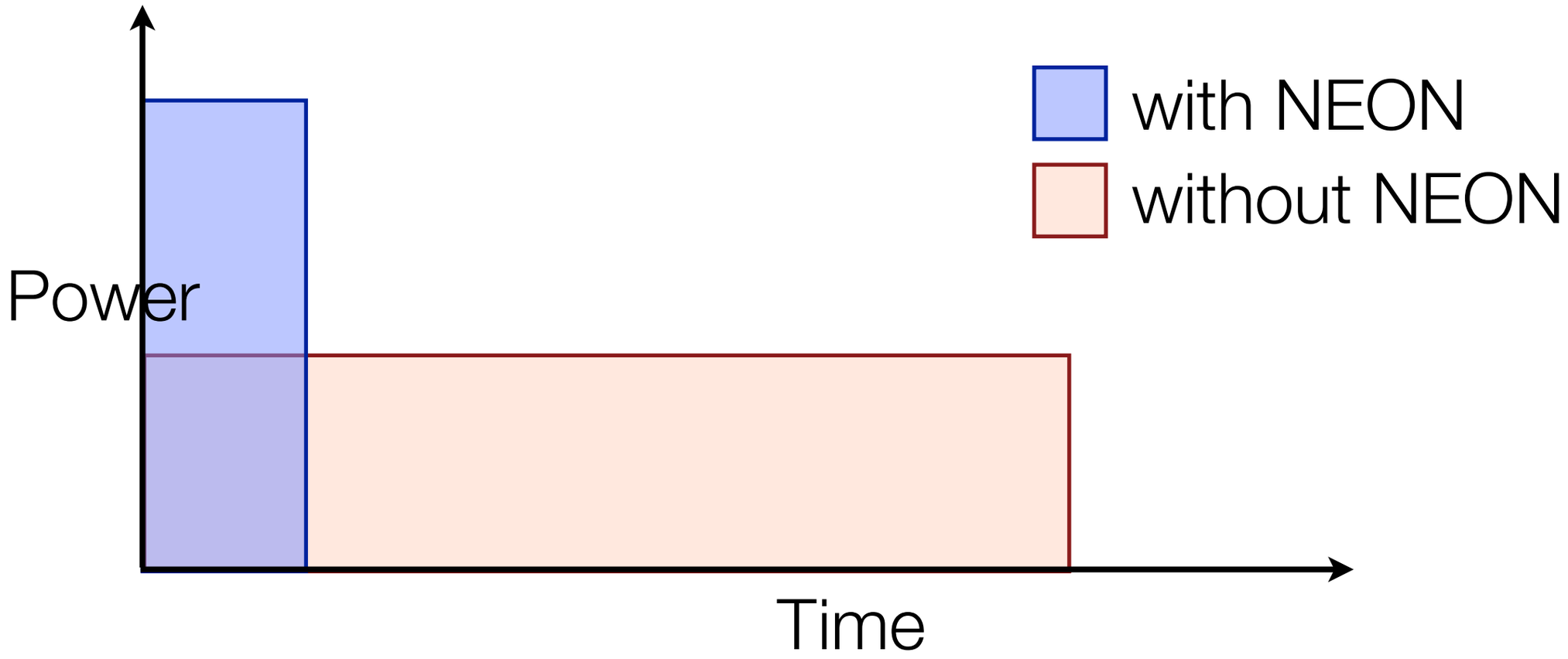
- 4G > 3G > 2G > WiFi (LTE = ~5x WiFi)
- 4G / 3G requires radio in high/moderate power state after last packet is sent or received
- LTE: 11 seconds, WiFi: 210 ms
- Low signal uses more power
- Jumping between 4G and 3G/2G drains power
- 2G less power but for a longer time

Network

- Bandwidth usage impacts time and power
- Limit number of connections
- Do not poll
- Use compact data formats
  - (CSV vs JSON vs XML)
  - Compress

CPU

- Do not poll, use events
- Accelerate: more power, less energy



CoreLocation

- Use minimum required accuracy

GPS	kCLLocationAccuracyBest
GPS	kCLLocationAccuracyNearestTenMeters
WiFi	kCLLocationAccuracyHundredMeters
Cell / WiFi	kCLLocationAccuracyKilometer

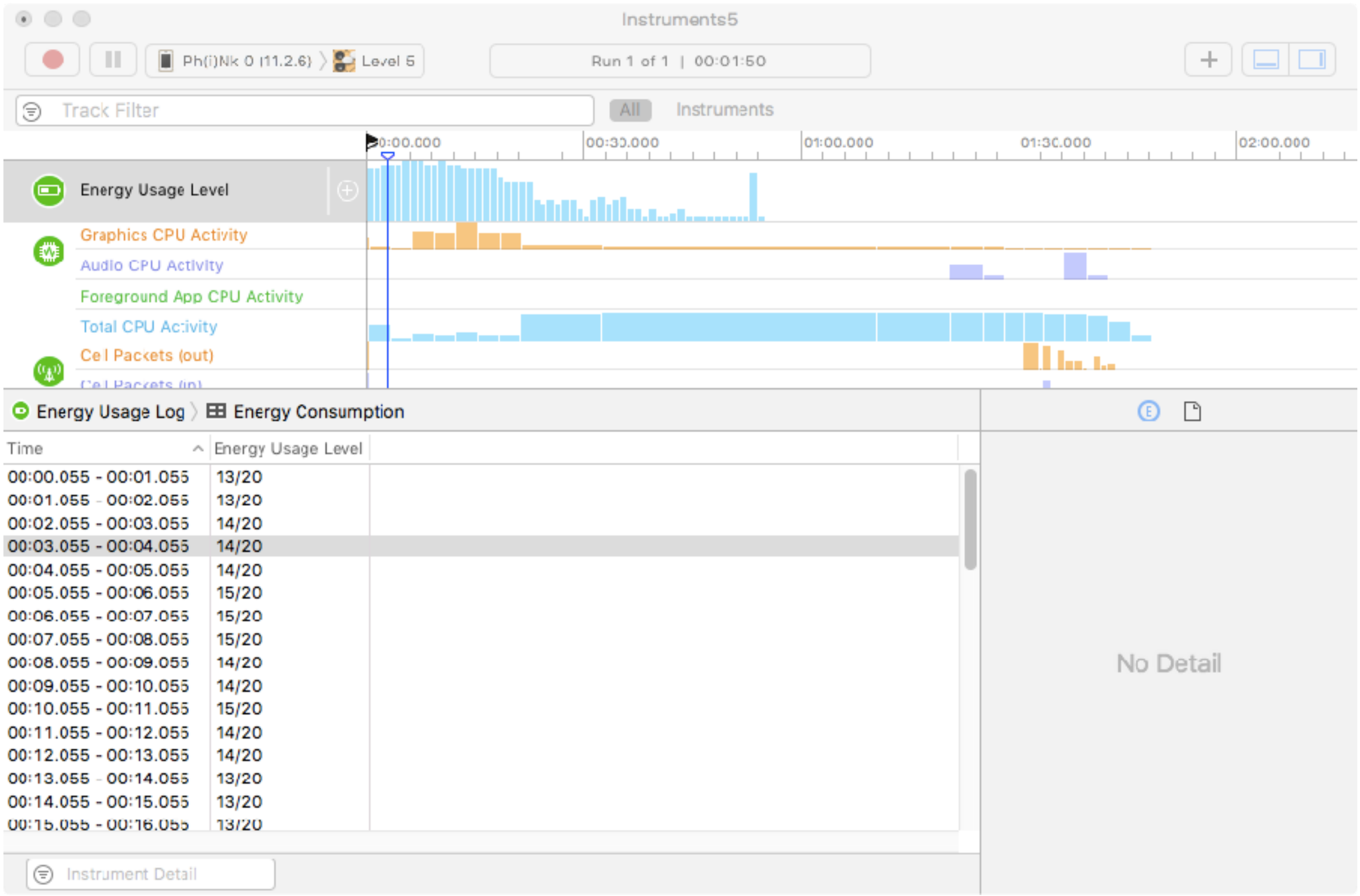
CoreLocation

- distanceFilter
  - how often you receive location changed notifications
    - default: all changes, many events, high CPU usage
- stopUpdatingLocation
  - when good enough accuracy, switch GPS off.

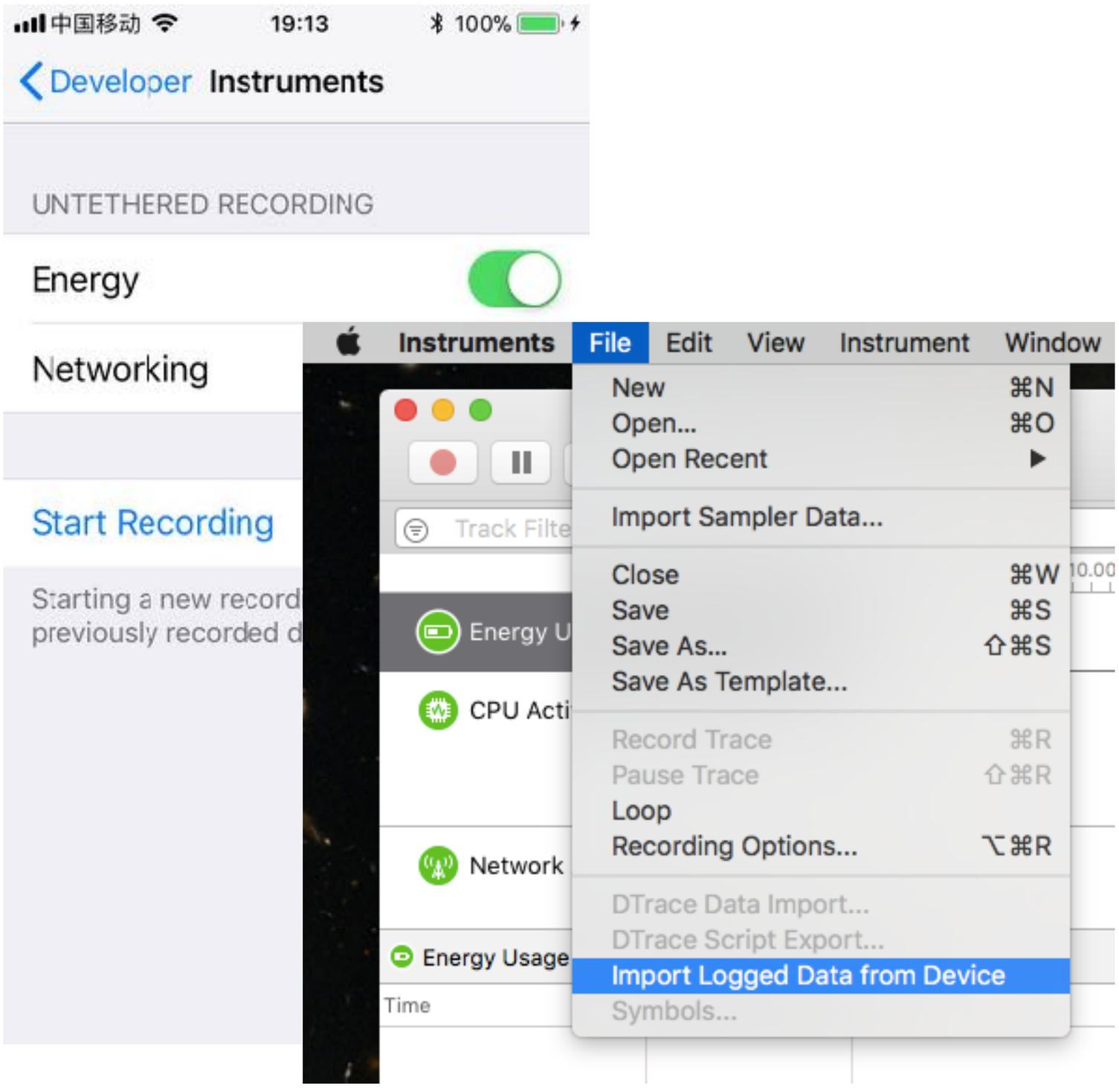
CoreMotion

- Same thing
- Turn sensors off when in background

Energy diagnostics



Measure on the go



- Enable device to collect data
- Start/Stop recording
- Import in Instruments

Summary

- Performance optimization is important
- It’s also fun
- Remember your data structures, algorithms, compiler classes…
- When? Always!

Resources

Marcel Weiher, *iOS and macOS Performance Tuning: Cocoa, Cocoa Touch, Objective-C*, and Swift. ISBN-13: 978-0321842848

Apple, *Performance Tips*,
https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/PerformanceTips/PerformanceTips.html

Apple, *Instruments User Guide*,
https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/index.html#//apple_ref/doc/uid/TP40004652

Apple, *Understanding Swift Performance, WWDC 2016*
https://developer.apple.com/videos/play/wwdc2016/416/

Writing High-Performance Swift Code,
https://github.com/apple/swift/blob/master/docs/OptimizationTips.rst

Performance, optimization and why it matters

Marco Scheurer

Lausanne, Switzerland / Shanghai, China

SENTE

marco@sente.ch

phink0