Instruments

IRON
HACK

Leo Font

Instruments

- Set of tools to analyze & fine tune our App.
- The most important thing: our App has to perform smoothly.

Instruments

If your App consumes too much memory or resources:

- Poor user experience (loading times…)
- System will get your App terminated.
- Bugs will get your App terminated too.

How to start Instruments?

- From Xcode Debug Navigator.
- Xcode Cmd + I == Profile.
- We can anchor Instruments to the Dock.

Xcode debug Gauges

- Show us memory, cpu, disk & network use in a simple way.
- Instruments can be launched from there.

Memory

- Instruments focused on allocation Heap
- Processes contain more memory than just memory:
  - Application code.
  - Images and other media.

How to get there? Depends in what you are measuring and how.

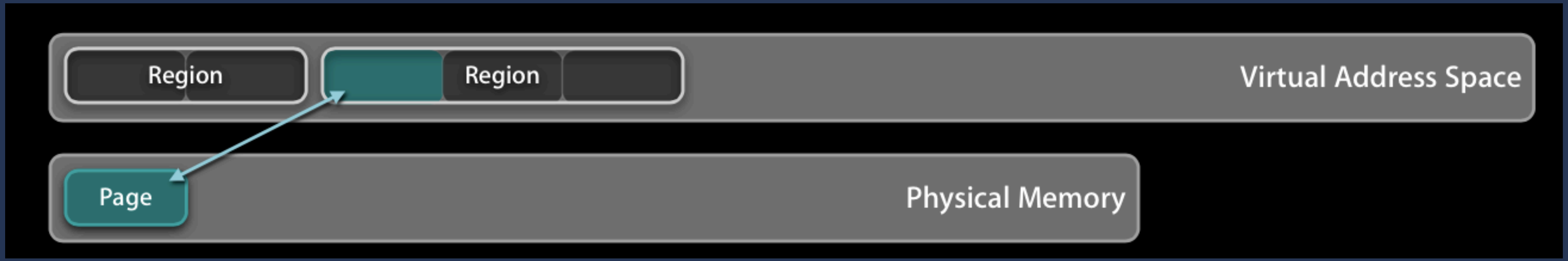
All Allocations

- Shows all allocations (not only the heap).
- We can filter.

KEEP
CALM
IT'S
YOUR
TURN NOW

Virtual memory vs Resident memory

- Virtual memory: all RAM in our device visible to our process reserved as regions.
- Memory aligned in 4K pages.
- When program starts, we map Pages to physical memory -> virtual memory->resident memory



Clean vs Dirty

- Clean memory: Pages that haven't been changed in memory
  - Can recreate them easily
  - Images, sounds, plists…
  - Can be discarded and reloaded.
  - If you change it, it’ll become dirty.

Clean vs Dirty

- Dirty memory: Pages changed by our code, or the user
  - Malloc heap, global variables, stack etc.
  - Can be swapped out / in (OSX)
  - In iOS, too much: process terminated

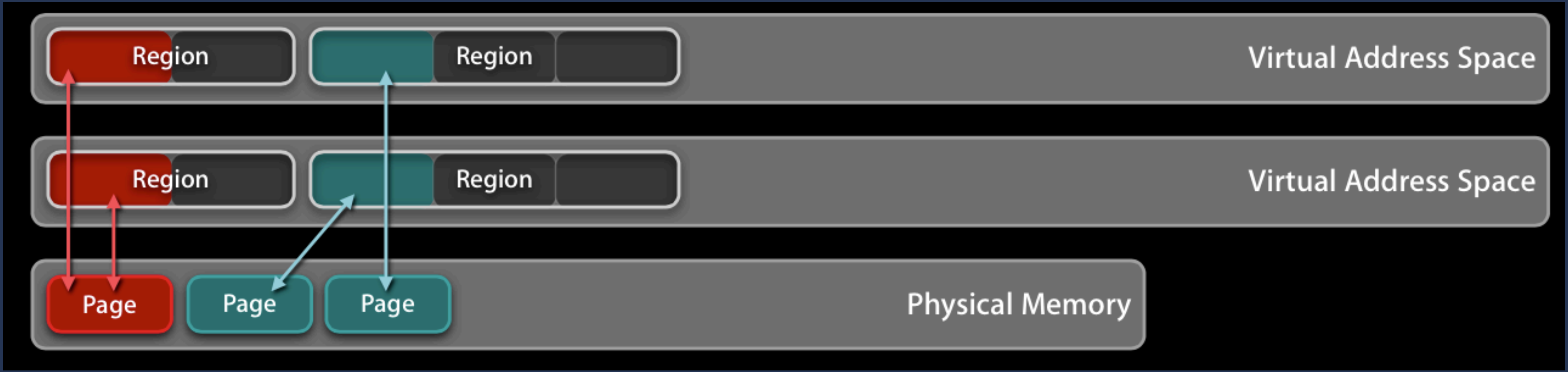
Private vs Shared

- Private:
    - Pages of memory only from my App
    - Where I create my objects

Private vs Shared

- Shared:
  - Pages of memory can be shared between processes
  - Frameworks loaded into memory

Private vs Shared



Private Dirty memory

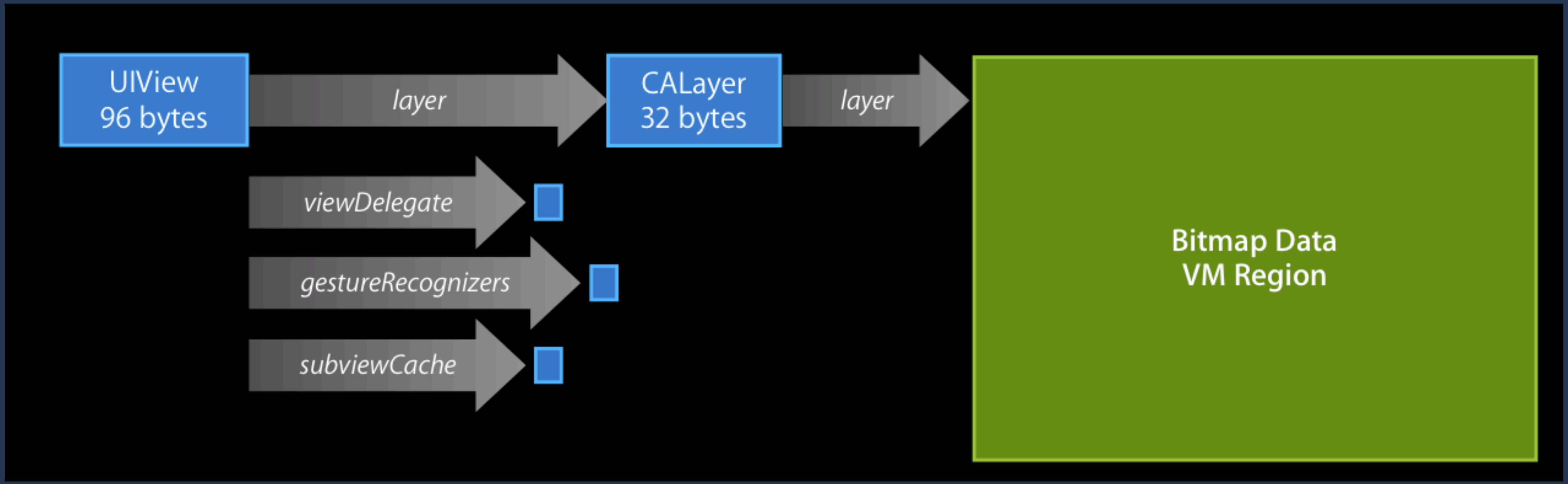
- What Xcode gauges show

Heap

- Storage for malloc() calls
    - malloc(), calloc(), realloc() in C
    - [NSObject alloc] in Objective C
    - new in C++
    - Backed by VM: MALLOC regions
    - Often reference managed

Expensive types

- VM is about bytes, heap is about counts
- Small object can have a large graph
- Obvious containers: NSSet, NSDictionary, NSArray…
- Less obvious: UIView, UIViewController, NSImage…



KEEP
CALM
IT'S
YOUR
TURN NOW

Leaks

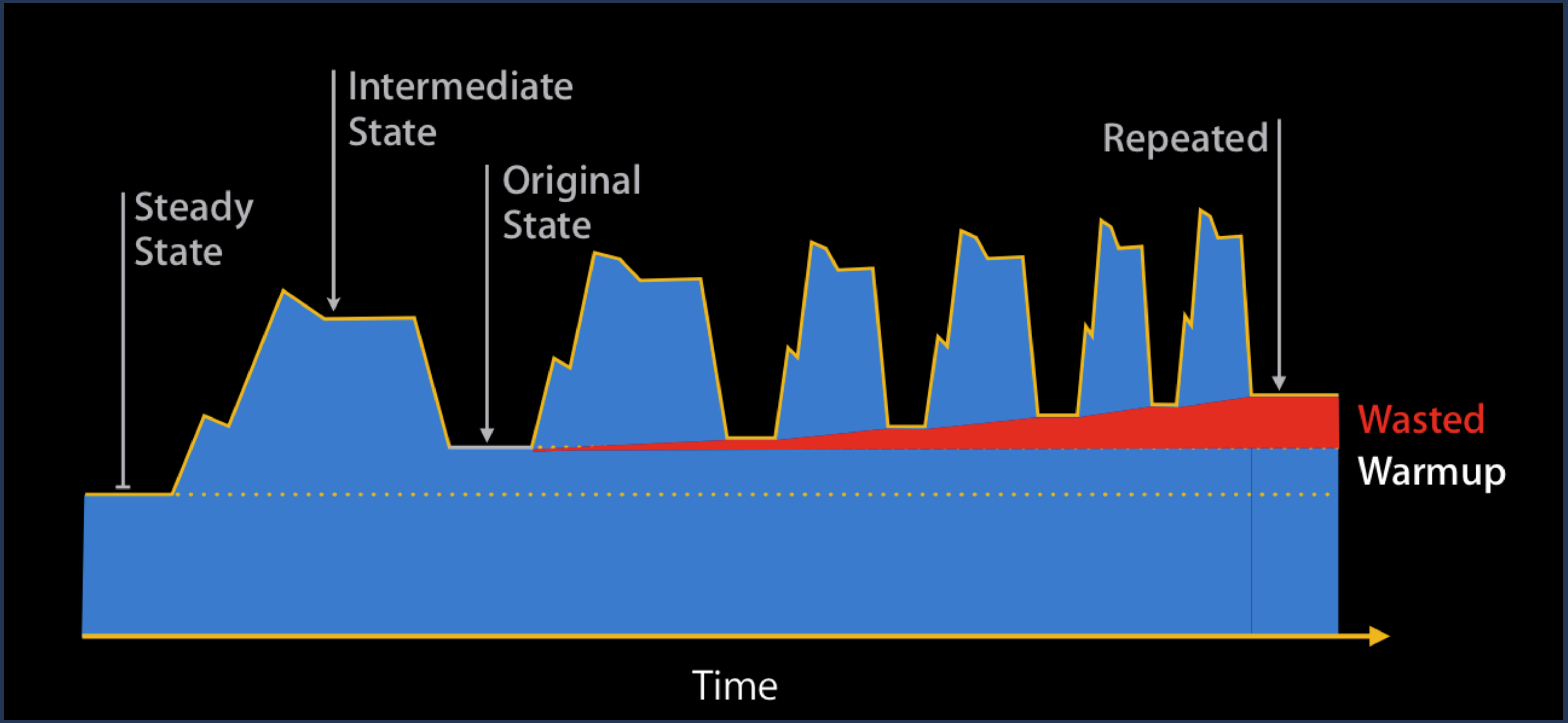
- No more pointers to access that memory
- Can't be used again

Abandoned memory

- Still referenced, but wasted
- Won’t ever be used again
- Be careful with cache (Maybe use NSCache)

Generational Analysis

- Technique for measuring memory growth



Document model

- Instruments use a document based mode: templates.
- We can choose one upon start or before pressing:
  - Alt+CMD+I
- Each template has a set of “Instruments”

Profile debug vs release builds

- Default: profile **release** builds
- Leaks: **debug** info helps
  - Time profiling: profile **release** builds
  - Memory: profile **debug** bulds
- Change in Scheme > Profile App

Memory in your app

- Heap memory: what you see
- Everything else: you don't notice it, but it's there
  - Managed object contexts internal mem
  - Backing Layers

Managing memory manually?

- Switch to ARC
- Run the Static Analyser
- Zombie template to the rescue!

Allocations Instrument

- Info for your **Heap** allocations
  - Class names different in ObjC code / Swift: Swift prefixes with module name
  - Only shows reference types (classes, not structs)
- Can search (filter) for class name
- Can select a class > Dive into, we can see the retain / release history
- Number of persistent objects vs transient objects

Leaks

- Will show memory unreferenced in system: a type of persistent memory growth (Leak)

Generations of objects:

- Can mark generations and see when they where created

Zombies

- NSZombieEnabled=1
- Zombie objects are not deallocated: stay in memory and always crashes when accessed
- Every Zombie Object leaks: don't look for leaks & zombies

Time profiler

- Select a part of the timeline and zoom-in
- Hide system libraries: shows just my code
- Invert call stacks: Usually last lines of code tend to be the problematic ones