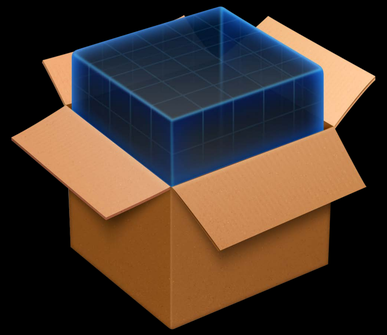
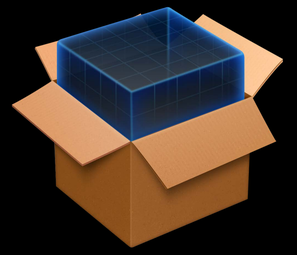
#WWDC19

Developing a Great Profiling Experience

Telling your story with Custom Instruments

Daniel Delwood, Software Radiologist
Kacper Harasim, Software Engineer





```
/**
@protocol MTLCommandBuffer
 @abstract A serial list of commands for the device to execute.

public protocol MTLCommandBuffer : NSObjectProtocol {
    var device: MTLDevice { get }
    var commandQueue: MTLCommandQueue { get }
    var retainedReferences: Bool { get }
    var label: String? { get set }

    func enqueue()
    func commit()

    func addScheduledHandler(_ block: @escaping
            MTLCommandBufferHandler)
    func present(_ drawable: MTLDrawable)
    func present(_ drawable: MTLDrawable, atTime
            presentationTime: CFTimeInterval)
    func waitUntilScheduled()
    func addCompletedHandler(_ block: @escaping
            MTLCommandBufferHandler)
...
```

Documentation › Metal

Language: Swift API Char

Framework

Metal

Render advanced 3D graphics and perform data-parallel computations using graphics processors.

SDKs

iOS 8.0+

macOS 10.11+

tvOS 9.0+

On This Page

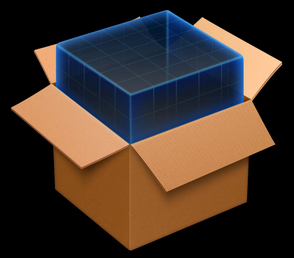
Overview

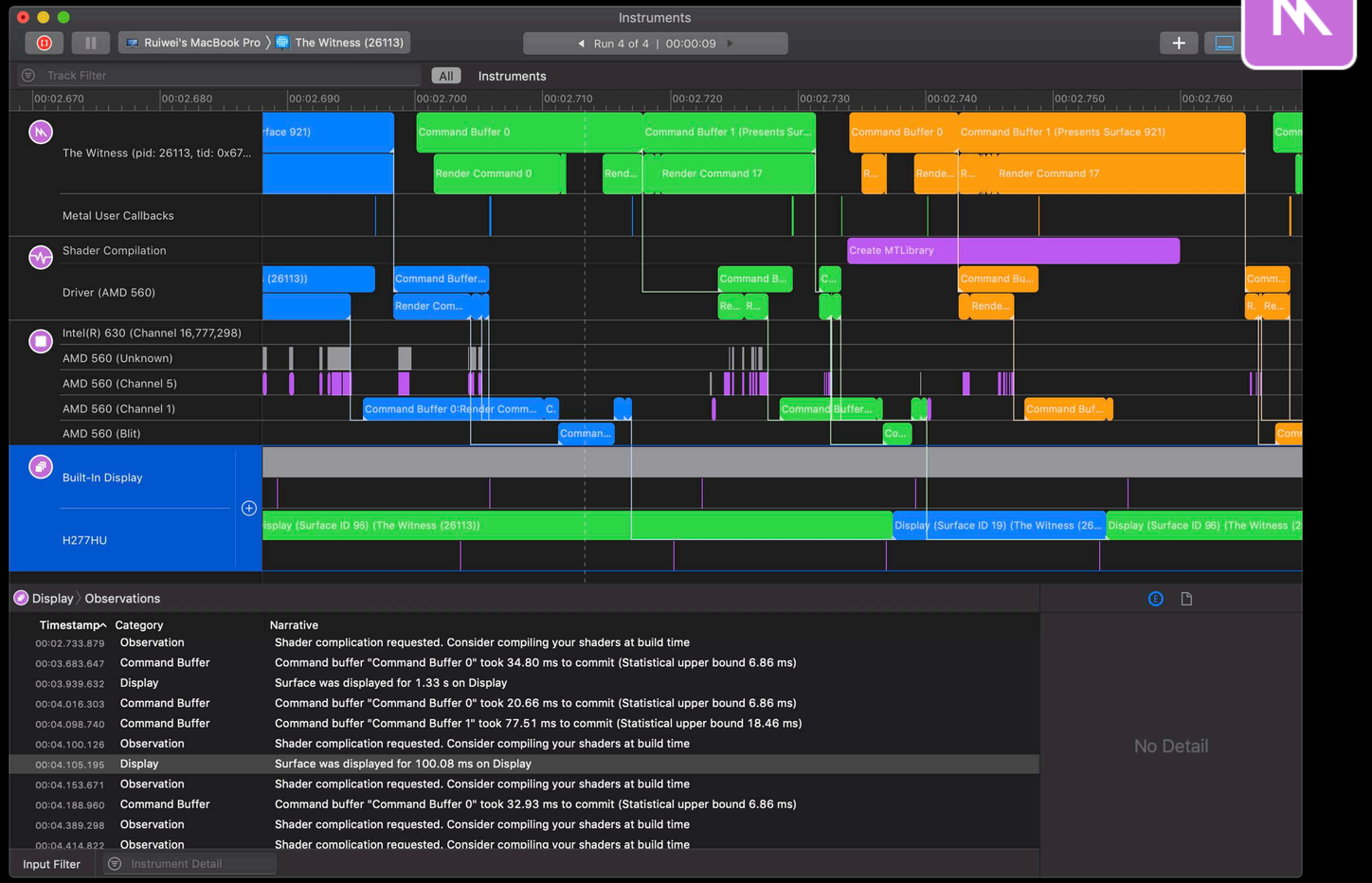
Graphics processors (GPUs) are designed to quickly render graphics and perform data-parallel calculations. Use the Metal framework when you need to communicate directly with the GPUs available on a device. Apps that render complex scenes or that perform advanced scientific calculations can use this power to achieve maximum performance. Such apps include:

- Games that render sophisticated 3D environments
- Video processing apps, like Final Cut Pro
- Data-crunching apps, such as those used to perform scientific research

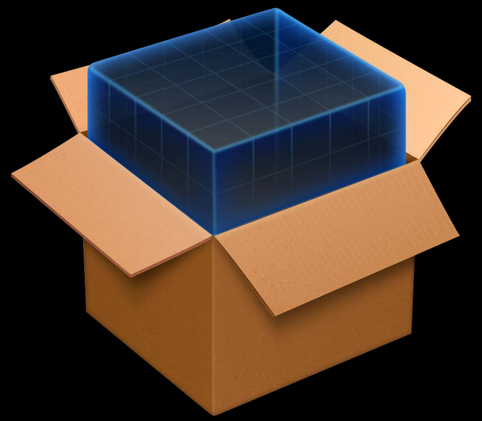
Metal works hand-in-hand with other frameworks that supplement its capability. Use MetalKit to simplify the task of getting your Metal content onscreen. Use Metal Performance Shaders to implement custom rendering functions or to take advantage of a large library of existing functions.

Many high level Apple frameworks are built on top of Metal to take advantage of its performance, including Core Image, SpriteKit, and SceneKit. Using one of these high-level frameworks shields you from the details of GPU programming, but writing custom Metal code enables you to achieve the highest level of performance.



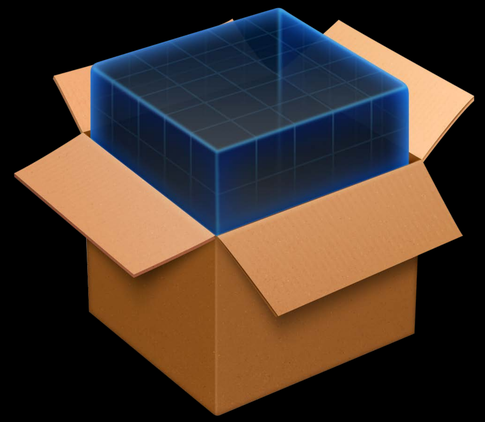

Instruments
Ruiwei's MacBook Pro 〉 The Witness (26113)
Run 4 of 4 | 00:00:09
Track Filter
All
Instruments
00:02.670
00:02.680
00:02.690
00:02.700
00:02.710
00:02.720
00:02.730
00:02.740
00:02.750
00:02.760
The Witness (pid: 26113, tid: 0x67...
Command Buffer 0
Command Buffer 1 (Presents Sur...
Command Buffer 0
Command Buffer 1 (Presents Surface 921)
Render Command 0
Render Command 17
Render Command 17
Metal User Callbacks
Shader Compilation
Create MTLibrary
Driver (AMD 560)
Command Buffer...
Render Com...
Intel(R) 630 (Channel 16,777,298)
AMD 560 (Unknown)
AMD 560 (Channel 5)
AMD 560 (Channel 1)
Command Buffer 0:Render Comm...
AMD 560 (Blit)
Built-In Display
H277HU
Display (Surface ID 96) (The Witness (26113))
Display (Surface ID 19) (The Witness (26...
Display 〉 Observations
Timestamp
Category
Narrative
00:02.733.879 Observation Shader complication requested. Consider compiling your shaders at build time
00:03.683.647 Command Buffer Command buffer "Command Buffer 0" took 34.80 ms to commit (Statistical upper bound 6.86 ms)
00:03.939.632 Display Surface was displayed for 1.33 s on Display
00:04.016.303 Command Buffer Command buffer "Command Buffer 0" took 20.66 ms to commit (Statistical upper bound 6.86 ms)
00:04.098.740 Command Buffer Command buffer "Command Buffer 1" took 77.51 ms to commit (Statistical upper bound 18.46 ms)
00:04.100.126 Observation Shader complication requested. Consider compiling your shaders at build time
00:04.105.195 Display Surface was displayed for 100.08 ms on Display
00:04.153.671 Observation Shader complication requested. Consider compiling your shaders at build time
00:04.188.960 Command Buffer Command buffer "Command Buffer 0" took 32.93 ms to commit (Statistical upper bound 6.86 ms)
00:04.389.298 Observation Shader complication requested. Consider compiling your shaders at build time
Input Filter
Instrument Detail
No Detail

Goals



Goals

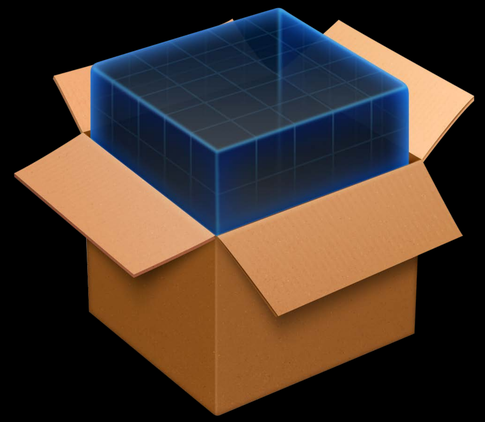
Transparency → Trust



Goals

Transparency → Trust

Explain cost, determine responsibility

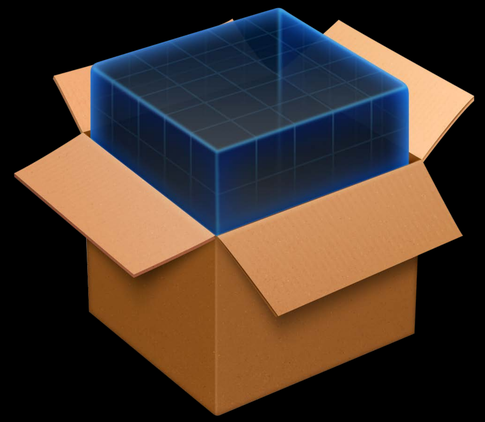


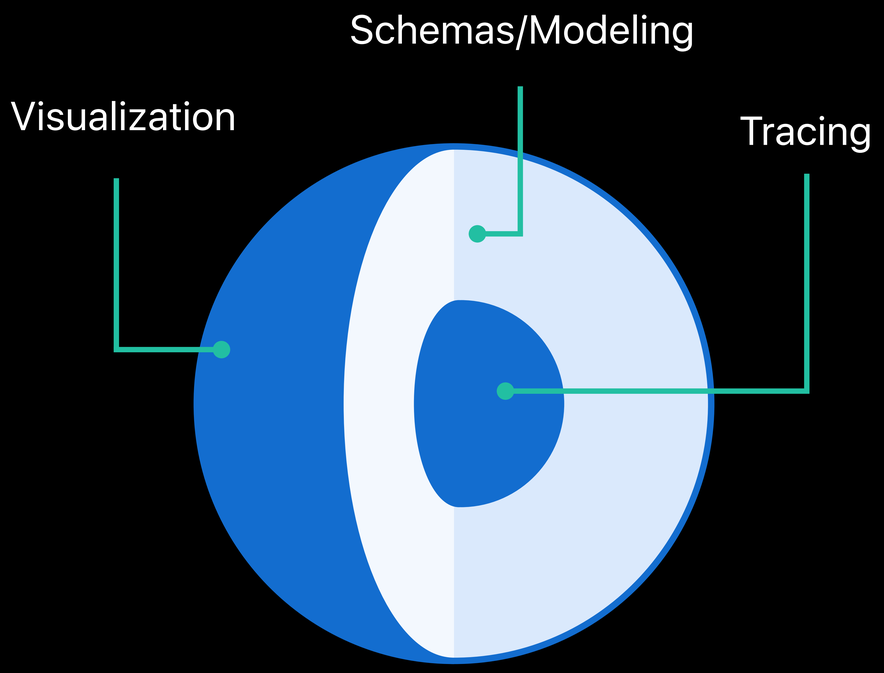
Goals

Transparency → Trust

Explain cost, determine responsibility

Opportunity to tell your story




Schemas/Modeling
Visualization
Tracing

Tracing: Adding os_signpost to your code

Modeling: Adding structure to temporal data

Visualization: Telling your story with an instrument

Tracing: Adding os_signpost to your code

Modeling: Adding structure to temporal data

Visualization: Telling your story with an instrument

os_signpost

Low-cost tracing primitive

Points (.event) or intervals (.begin/.end)

```
os_signpost(type, log: logHandle, name: name, signpostID: spid, format, args...)
```

```
os_signpost_interval_begin(logHandle, spid, name, ...)
os_signpost_interval_end(logHandle, spid, name, ...)
os_signpost_event_emit(logHandle, spid, name, ...)
```

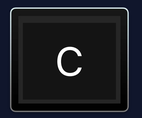
os_signpost

Low-cost tracing primitive

Points (.event) or intervals (.begin/.end)

```
os_signpost(type, log: logHandle, name: name, signpostID: spid, format, args...)
```

```
os_signpost_interval_begin(logHandle, spid, name, ...)
os_signpost_interval_end(logHandle, spid, name, ...)
os_signpost_event_emit(logHandle, spid, name, ...)
```



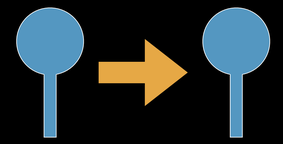
os_signpost

OSLog-based

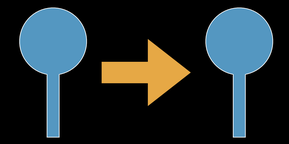
- Subsystem
- Category

Structure: Subsystem → Category → Name

Tracing Merits



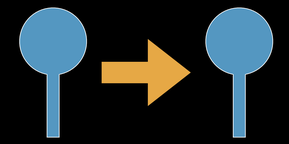
Tracing Merits



Temporal

- Implicitly records `mach_continuous_time`
- Inherently support overlapping intervals with signpostID

Tracing Merits



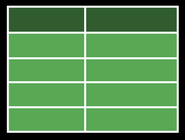
Temporal

- Implicitly records `mach_continuous_time`
- Inherently support overlapping intervals with signpostID

Low cost

- Format strings are `__TEXT` constants, cheap
- Not #DEBUG-only — leave them in

Signpost Schema



```
os_signpost(type, log: logHandle, name: name, signpostID: spid, format, args...)
```

	Available Columns
Explicit	event-type, subsystem, category, name, identifier, format-string, message
Implicit	time, thread, process, scope, backtrace*

* .dynamicStackTracing only

Modes of os_signpost

Modes of os_signpost

Enabled (default)

- Low-cost, logged to ring buffer

Modes of os_signpost

Enabled (default)

- Low-cost, logged to ring buffer

Streaming

- Higher cost when Instruments, `log stream --signpost` displaying live

Modes of os_signpost

Enabled (default)

- Low-cost, logged to ring buffer

Streaming

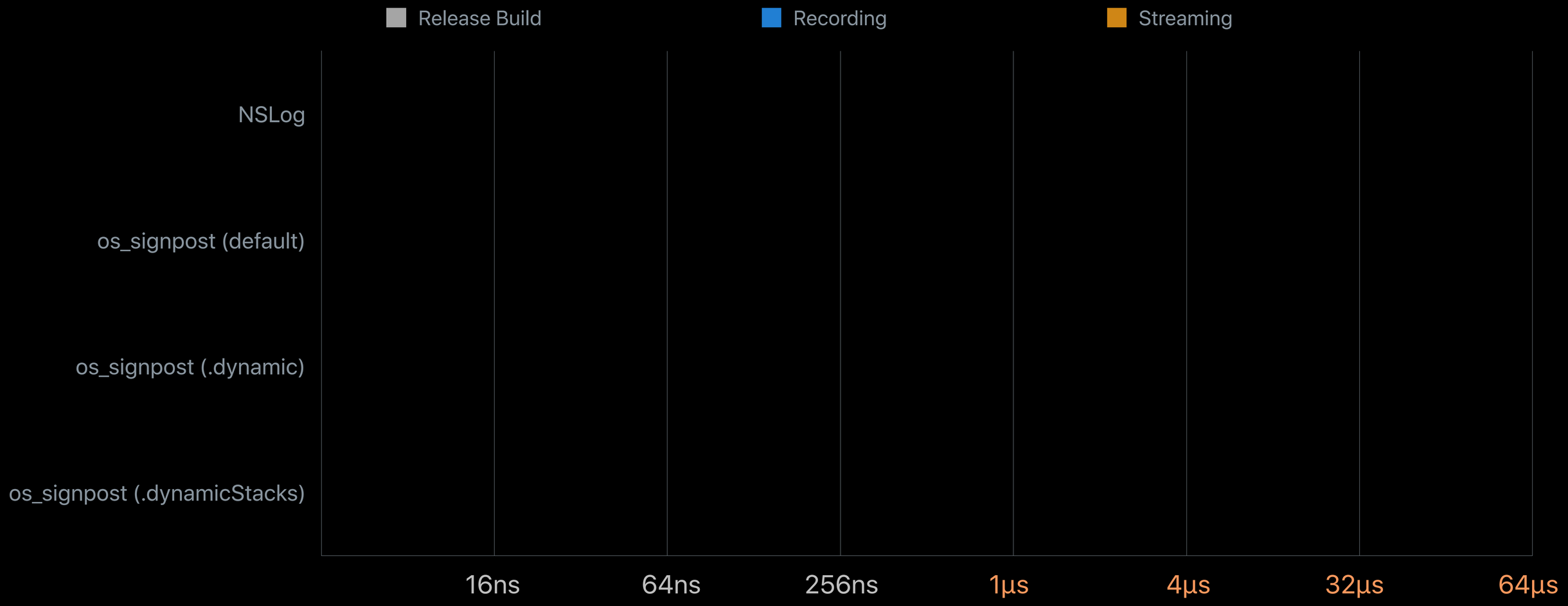
- Higher cost when Instruments, `log stream --signpost` displaying live

Dynamic

- `OSLog.Category.dynamicTracing, .dynamicStackTracing`
- Instruments enables only when recording

Tracing Cost

Order of magnitude approximations

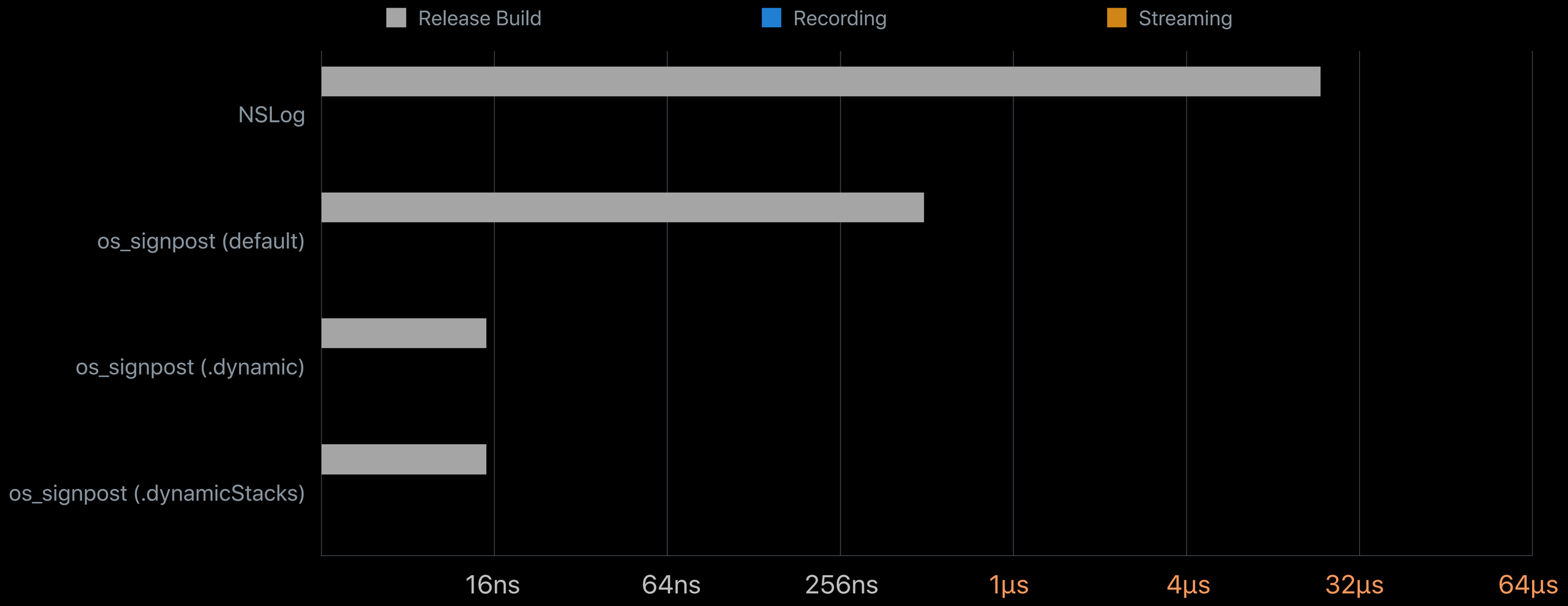


Smaller is better

(logarithmic scale)

Tracing Cost

Order of magnitude approximations

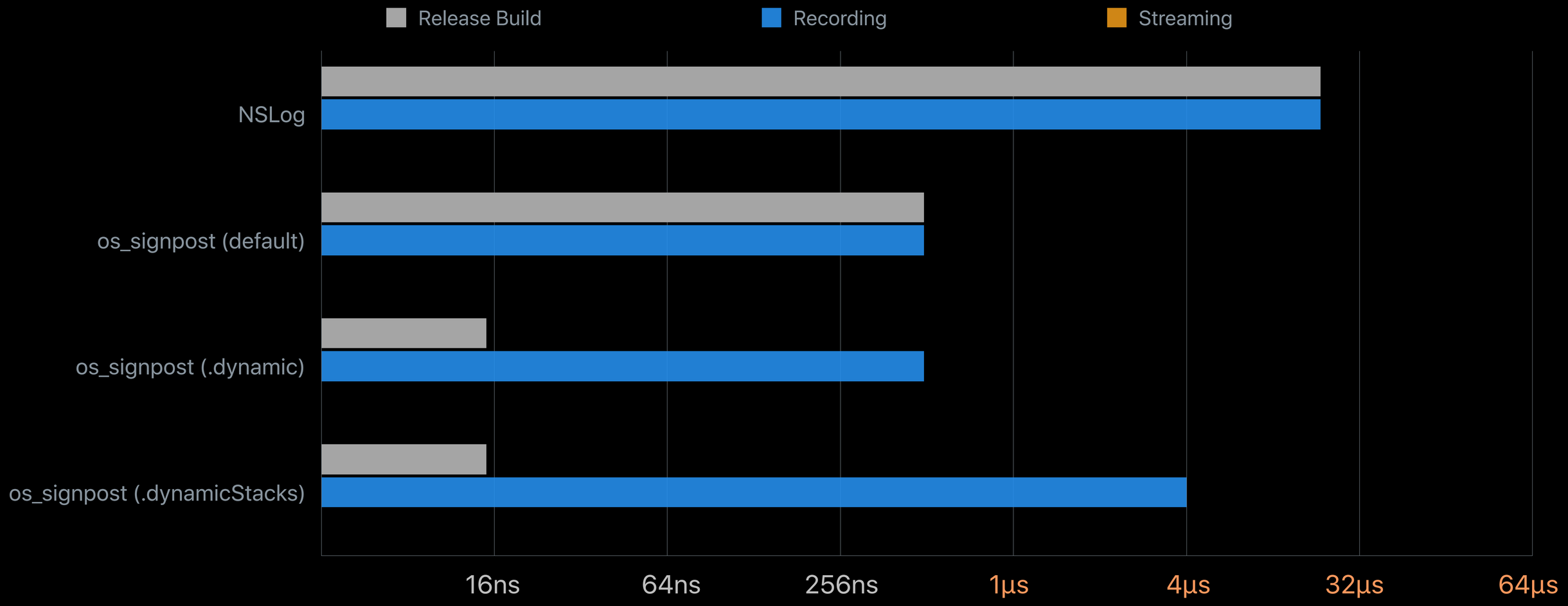


Smaller is better

(logarithmic scale)

Tracing Cost

Order of magnitude approximations

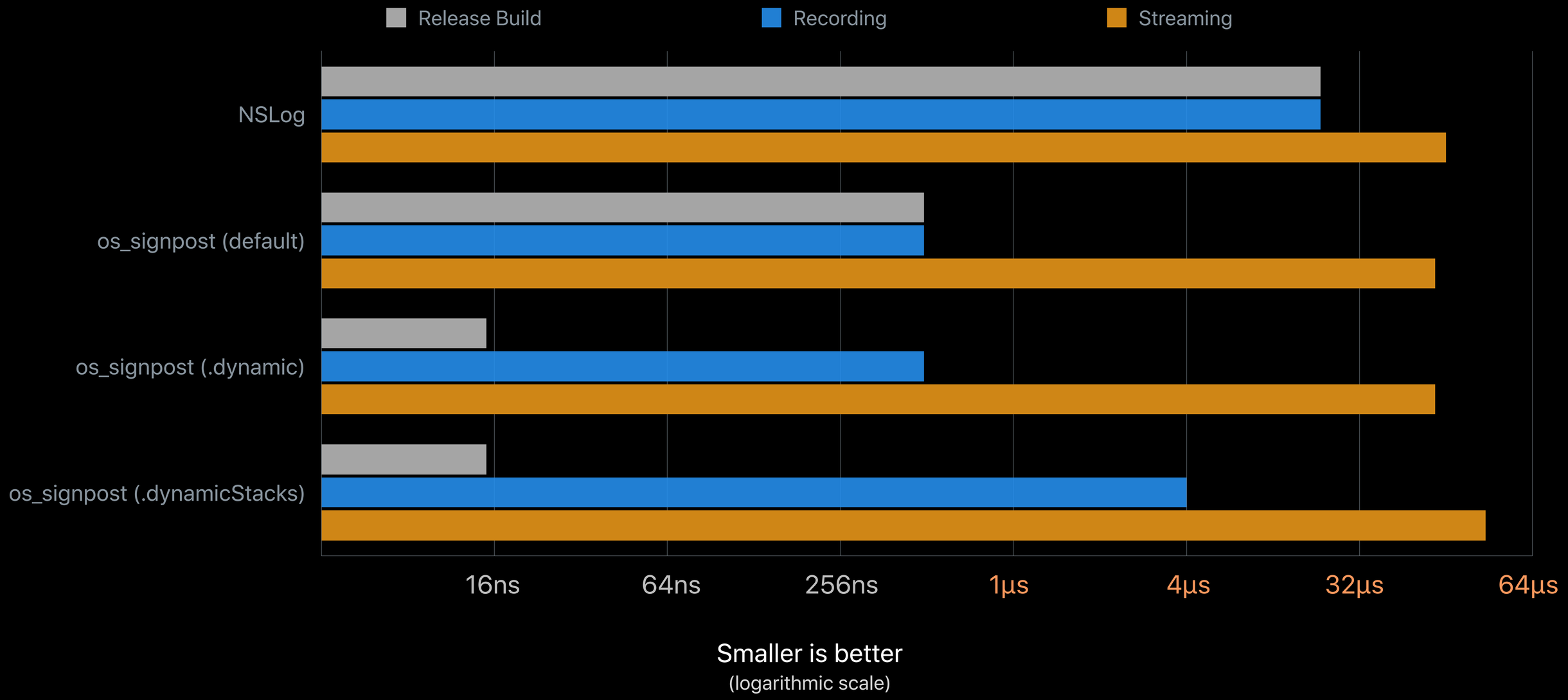


Smaller is better

(logarithmic scale)

Tracing Cost

Order of magnitude approximations



Minimizing Overhead

Minimizing Overhead

Record in deferred/last few seconds mode

Minimizing Overhead

Record in deferred/last few seconds mode

Use .dynamicTracing categories

```
let normalLog = OSLog(subsystem: "com.org.MyFramework", category: "CompressorPhases")
let highRateLog = OSLog(subsystem: "com.org.MyFramework", category: .dynamicTracing)
let needStacksLog = OSLog(subsystem: "com.org.MyFramework", category: .dynamicStackTracing)
```

Staying Under the Radar

Back-of-the-envelope math

Staying Under the Radar

Back-of-the-envelope math

< 1% CPU

(enabled, not streaming)

Staying Under the Radar

Back-of-the-envelope math

$$x * \frac{\sim 500ns}{signpost} < 1\% \text{ CPU}$$

(enabled, not streaming)

Staying Under the Radar

Back-of-the-envelope math

$$x * \frac{\sim 500ns}{signpost} < 1\% \text{ CPU}$$

(enabled, not streaming)

$$\rightarrow x < \frac{\sim 20{,}000 \text{ signposts}}{second}$$

Staying Under the Radar

Back-of-the-envelope math

$$x * \frac{\sim 500ns}{signpost} \quad < 1\% \text{ CPU}$$

(enabled, not streaming)

$$\rightarrow x < \frac{\sim 20{,}000 \text{ signposts}}{second} = \frac{\sim 83 \text{ intervals}}{frame} \text{ @120 fps}$$

Tailored to Your Audience

Tailored to Your Audience

Signposts use shared system resources

- Cost will vary by device hardware, software load
- As always — use what you need

Tailored to Your Audience

Signposts use shared system resources

- Cost will vary by device hardware, software load
- As always — use what you need

One or more OSLog handles per audience

- Tools for clients
- Tools for contributors

Tracing Best Practices

Tracing Best Practices

1. Always end intervals! `defer { }` when possible to avoid missing returns

```
do {
    let uniqueID = OSSignpostID(myHandle)
    os_signpost(.begin, log: myHandle, name: "Calculations", signpostID: uniqueID)
    let result = try expensiveProcessing(payloadData)
    /* additional result processing */
    os_signpost(.end, log: myHandle, name: "Calculations", signpostID: uniqueID)
} catch {
    /* handle error */
}
```

Tracing Best Practices

1. Always end intervals! `defer { }` when possible to avoid missing returns

```
do {
    let uniqueID = OSSignpostID(myHandle)
    os_signpost(.begin, log: myHandle, name: "Calculations", signpostID: uniqueID)
    let result = try expensiveProcessing(payloadData)
    /* additional result processing */
    os_signpost(.end, log: myHandle, name: "Calculations", signpostID: uniqueID)
} catch {
    /* handle error */
}
```

Tracing Best Practices

1. Always end intervals! `defer { }` when possible to avoid missing returns

```
do {
    let uniqueID = OSSignpostID(myHandle)
    os_signpost(.begin, log: myHandle, name: "Calculations", signpostID: uniqueID)
    defer { os_signpost(.end, log: myHandle, name: "Calculations", signpostID: uniqueID) }
    let result = try expensiveProcessing(payloadData)
    /* additional result processing */
} catch {
    /* handle error */
}
```

Tracing Best Practices

2. Log data once in earliest trace point available

```
os_signpost(.begin, log: myHandle, name: "DecodingPhase",
    "Request %u: decoding %lu bytes", requestNumber, raw.size)
```

```
// deserialization work
```

```
os_signpost(.end, log: myHandle, name: "DecodingPhase",
    "Request %u: %lu -> %lu bytes", requestNumber, raw.size, decoded.size)
```

```
os_signpost(.end, log: myHandle, name: "DecodingPhase",
    "Decoded size: %lu bytes", decoded.size)
```

Tracing Best Practices

2. Log data once in earliest trace point available

```
let spid = OSSignpostID(myHandle)
os_signpost(.begin, log: myHandle, name: "DecodingPhase", signpostID: spid
    "Request %u: decoding %lu bytes", requestNumber, raw.size)
```

```
os_signpost(.end, log: myHandle, name: "DecodingPhase", signpostID: spid
    "Decoded size: %lu bytes", decoded.size)
```

Tracing Best Practices

3. Avoid expensive argument evaluation

```
os_signpost(.event, log: myLoggingHandle, name: "ReceivedPayload",
                "Details: %@", myObject.expensiveJSONDecodedProperty.description)
```

Tracing Best Practices

3. Avoid expensive argument evaluation

```
os_signpost(.event, log: myLoggingHandle, name: "ReceivedPayload",
            "Details: %@", myObject.expensiveJSONDecodedProperty.description)
```

```
if myLoggingHandle.signpostsEnabled {
    let decoded = myObject.expensiveJSONDecodedProperty
    os_signpost(.event, log: myLoggingHandle, name: "ReceivedPayload",
                "Details: %@", decoded.description)
}
```

Tracing Best Practices

4. Only trace what you need, consider guard preconditions

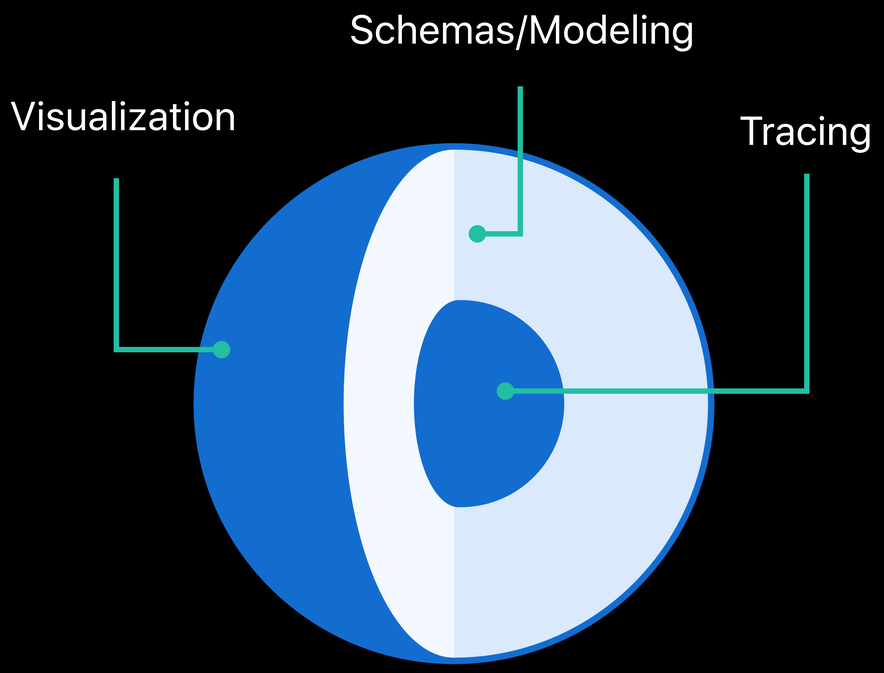
```
func needsDecompression(data: Data) -> Bool {
    os_signpost(.begin, log: myHandle, name: "Validation")
    defer { os_signpost(.end, log: myHandle, name: "Validation") }
    guard data.count > Int(vm_page_size) else {
        return false // not worth the trouble for such a small payload
    }
    // more expensive validation logic
}
```

Tracing Best Practices

4. Only trace what you need, consider guard preconditions

```
func needsDecompression(data: Data) -> Bool {
    guard data.count > Int(vm_page_size) else {
        return false // not worth the trouble for such a small payload
    }
    os_signpost(.begin, log: myHandle, name: "Validation")
    defer { os_signpost(.end, log: myHandle, name: "Validation") }
    // more expensive validation logic
}
```

Tracing Is Basis for Tools



Keeping Trace Points Stable

Avoid tracing frequently changing implementation details

No need to worry about inlining

Tools depend on stability of handles, signpost names, and format strings

Keeping Trace Points Stable

Avoid tracing frequently changing implementation details

No need to worry about inlining

Tools depend on stability of handles, signpost names, and format strings

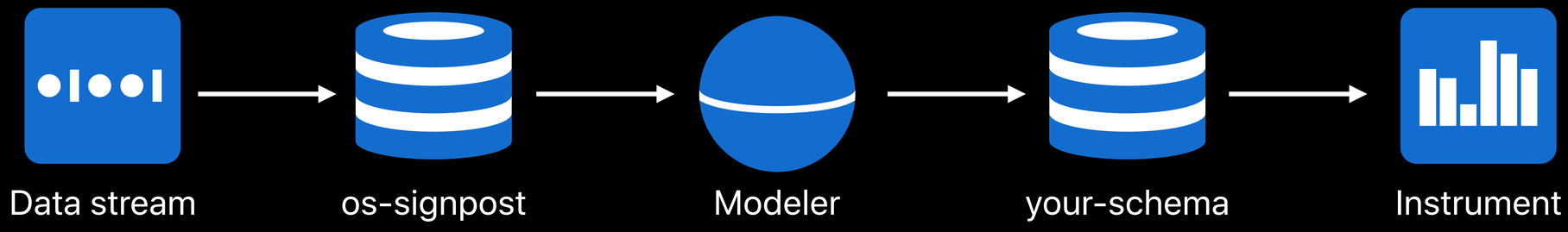
```
let logHandle = OSLog(subsystem: "com.org.MyFramework",
                      category: "Instrumented Activity")
os_signpost(.begin, log: logHandle, name: "ManagerLifecycle",
            signpostID: .exclusive, "[%@] Activated with delegate: %p",
            self.managerUUID, self.delegate)
```

Tracing: Adding os_signpost to your code

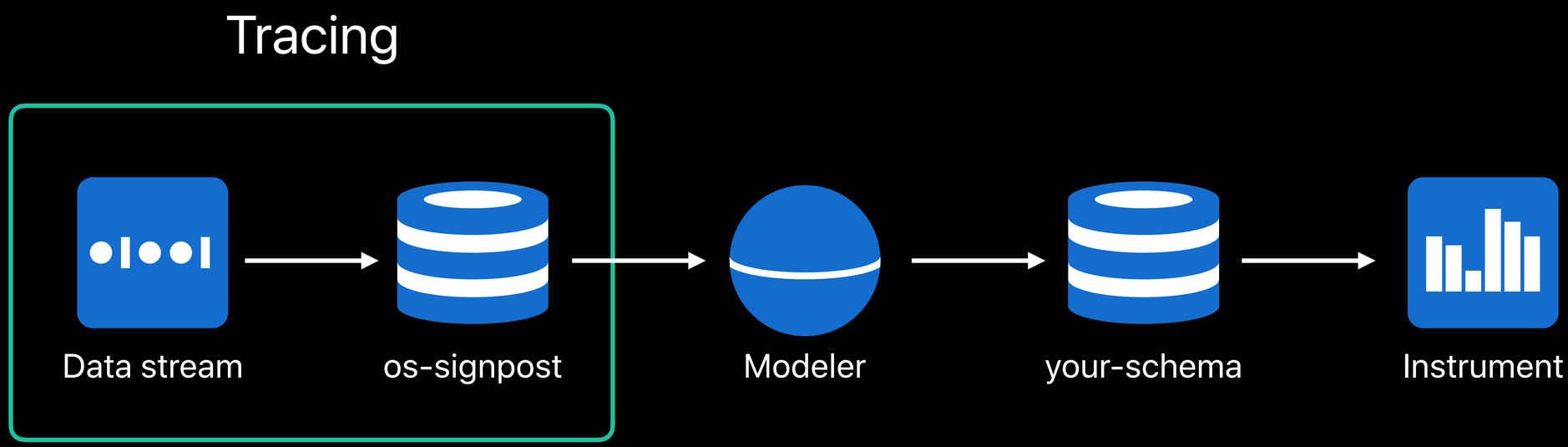
Modeling: Adding structure to temporal data

Visualization: Telling your story with an instrument

Instruments Architecture

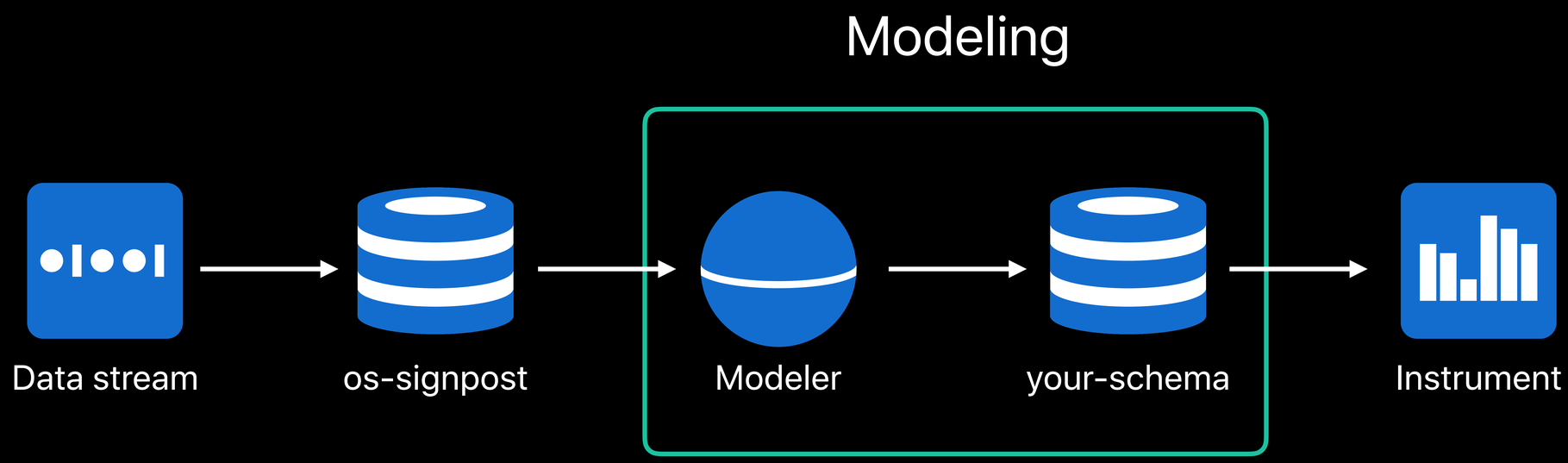


Instruments Architecture



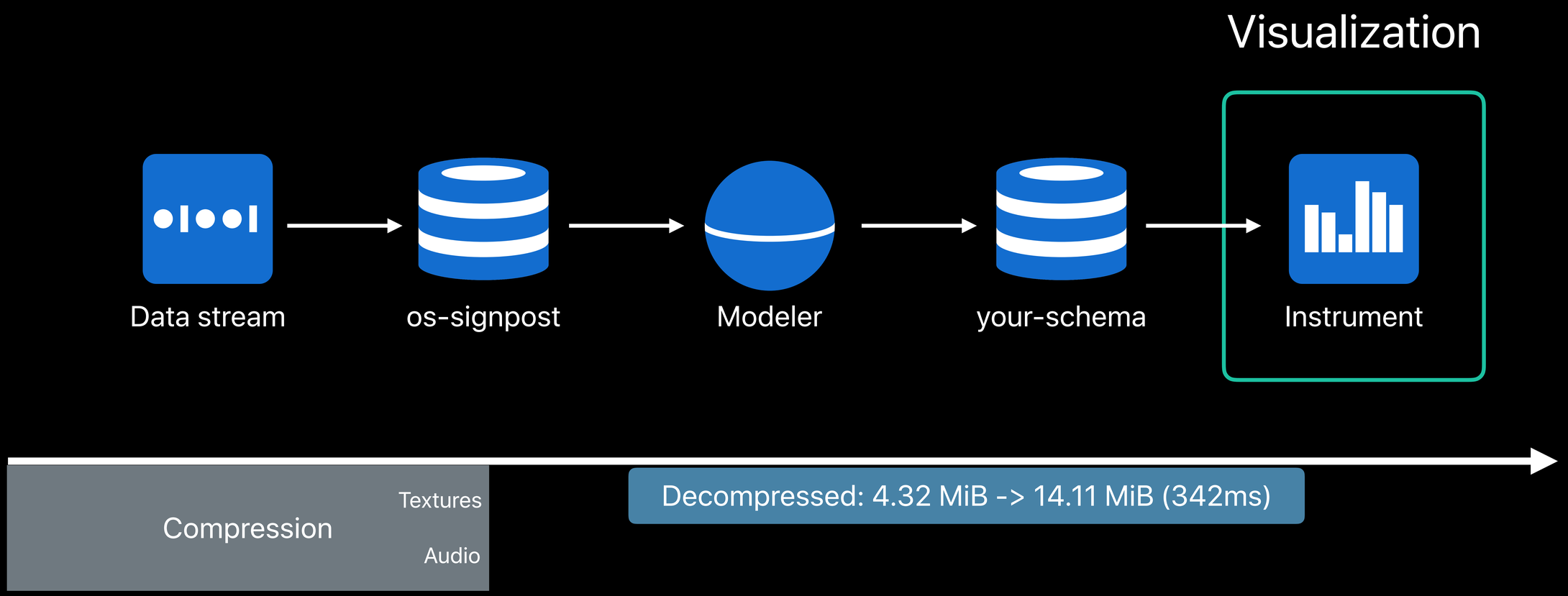

```
os_signpost(.begin, log: log, name: "Inflate", "(%@) Compressed: %d", tag, size)
[…]
os_signpost(.end, log: log, name: "Inflate", "Expanded: %d", final)
```

Instruments Architecture



start	duration	thread	client-tag	expanded	compressed	ratio
0:00:001.324	342ms	Thread 0x4016	Textures	14.11 MiB	4.32 MiB	3.27:1

Instruments Architecture



Getting Data into Instruments

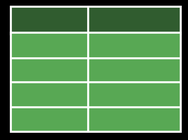
All data is time-ordered

- Point or Interval schemas

CLIPS Modelers

- Generated and custom

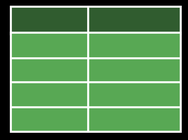
Getting Started



1. Built-in os_signpost instrument

- Use instrument inspector to see raw data

Getting Started



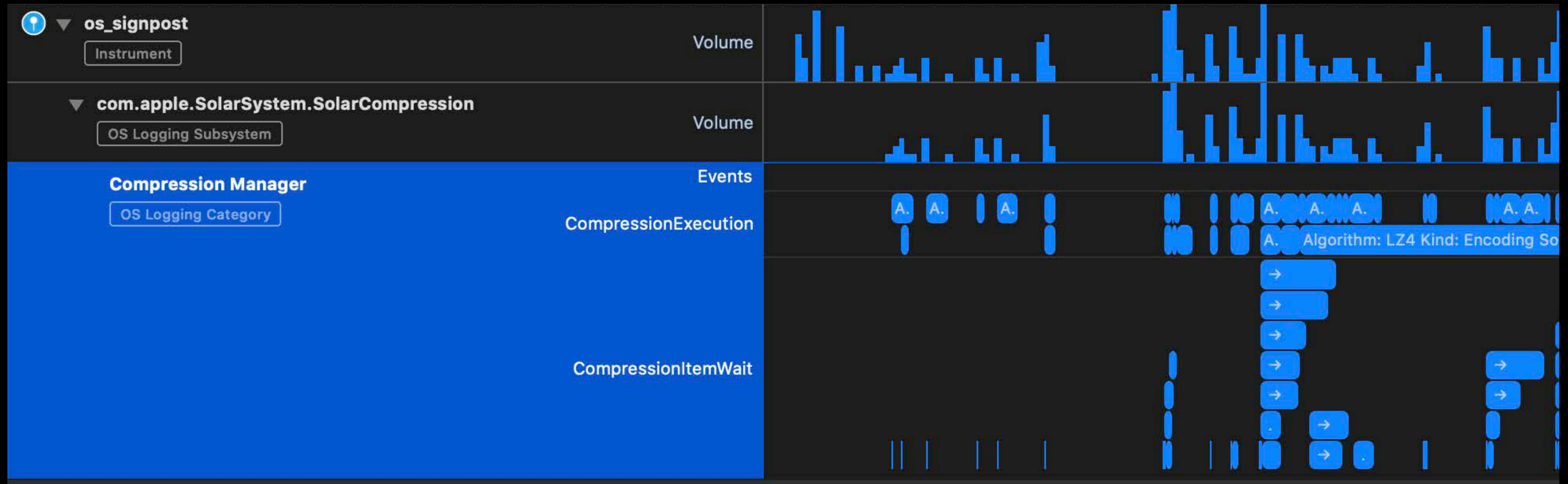
1. Built-in os_signpost instrument
- Use instrument inspector to see raw data

2. New Target → Instruments Package
- Define schema
  - Automatic modeler: <os-signpost-{point|interval}-schema>
- Define an Instrument to require/display
  - <instrument> must <create-table>

Demo

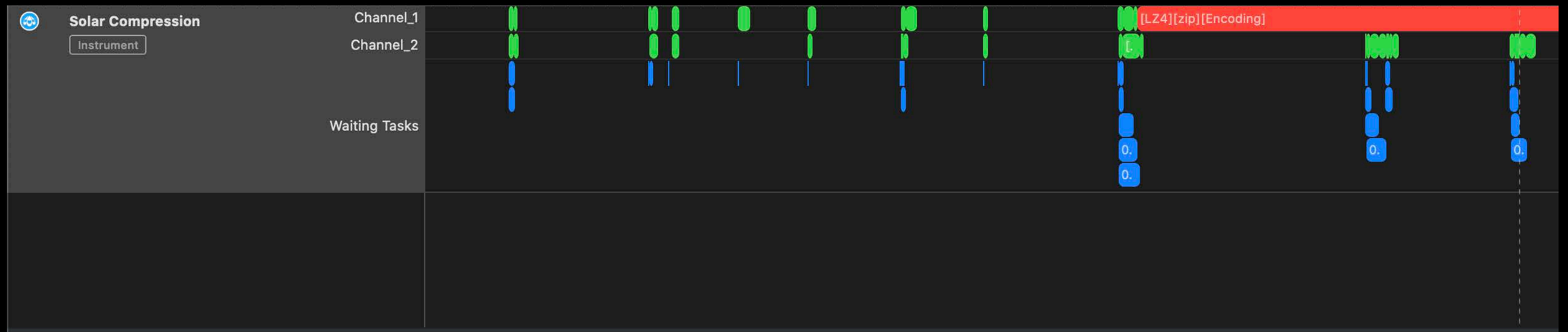
From os_signpost to Instruments

Kacper Harasim, Software Engineer



Compression Manager 〉 Summary: Intervals

Name / Start Message / End Message	Count	Duration	Min Duration	Avg Duration	Std Dev Duration	Max Duration
* All *	94	4.06 s	83.26 µs	43.16 ms	214.55 ms	2.09 s
CompressionExecution	47	2.92 s	2.08 ms	62.06 ms	302.19 ms	2.09 s
CompressionItemWait	47	1.14 s	83.26 µs	24.25 ms	31.90 ms	124.31 ms



Solar Compression 〉 Task Summary

Compression Kind / Source Extension / Algorithm	Count	Avg Compres...	Avg Duration	Total Durat...	Min Duration	Max Durati...	Std Dev Du...	Min Compres...	Max Compre...
▼ * All *	45	1.1209	11.26 ms	506.69 ms	729.13 µs	332.55 ms	49.05 ms	1.0174	1.4996
▼ Encoding	45	1.1209	11.26 ms	506.69 ms	729.13 µs	332.55 ms	49.05 ms	1.0174	1.4996
▶ png	25	1.0972	2.91 ms	72.77 ms	2.05 ms	5.08 ms	836.19 µs	1.0971	1.0984
▶ mov	14	1.0906	6.81 ms	95.39 ms	4.34 ms	13.14 ms	2.65 ms	1.0881	1.0965
▼ jpg	5	1.3451	1.20 ms	5.99 ms	729.13 µs	1.44 ms	271.70 µs	1.2641	1.4996
LZ4	5	1.3451	1.20 ms	5.99 ms	729.13 µs	1.44 ms	271.70 µs	1.2641	1.4996
▶ zip	1	1.0173	332.55 ms	332.55 ms	332.55 ms	332.55 ms	n/a	1.0174	1.0174

Custom schema, custom modeler

When you need more: data fusion, inference

Allows you to embed more complex logic

Draw more customized graphs

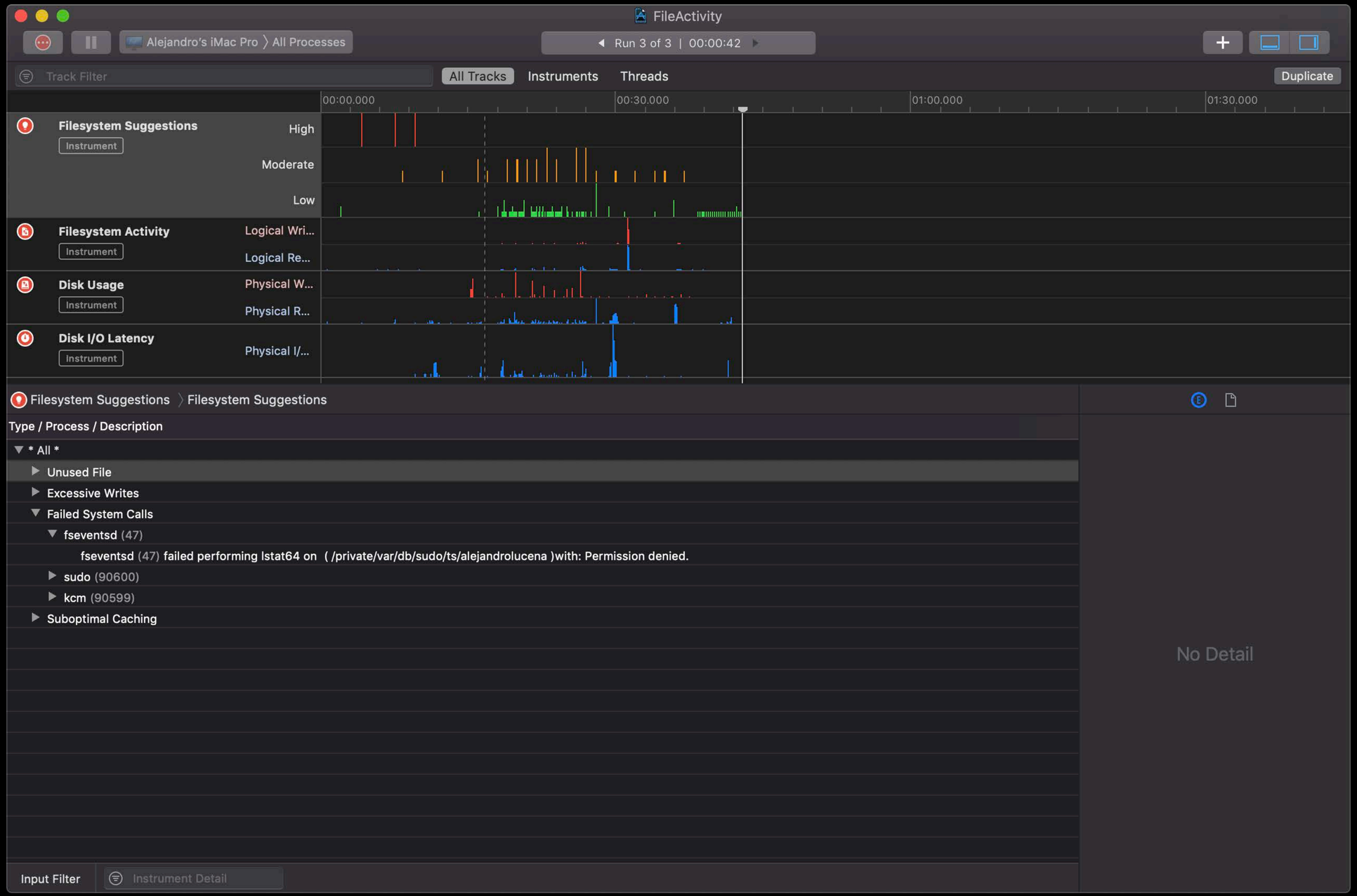
<point-schema>, <interval-schema>, <modeler>

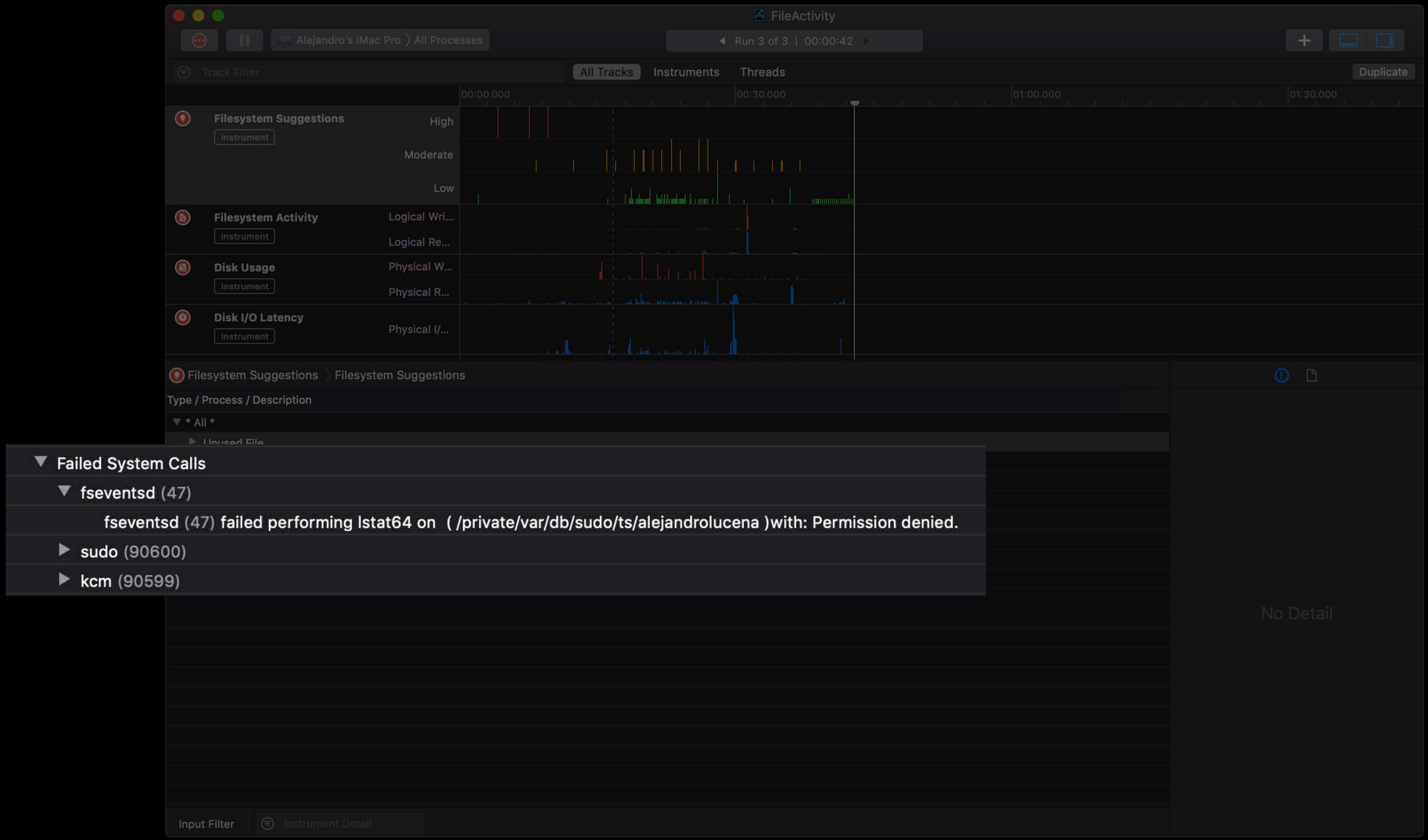
Tracing: Adding os_signpost to your code

Modeling: Adding structure to temporal data

Visualization: Telling your story with an instrument

Data ≠ Story


FileActivity
Alejandro's iMac Pro 〉All Processes
Run 3 of 3 | 00:00:42
Track Filter
All Tracks
Instruments
Threads
Duplicate
00:00.000
00:30.000
01:00.000
01:30.000
Filesystem Suggestions
Instrument
High
Moderate
Low
Filesystem Activity
Instrument
Logical Wri...
Logical Re...
Disk Usage
Instrument
Physical W...
Physical R...
Disk I/O Latency
Instrument
Physical I/...
Filesystem Suggestions 〉Filesystem Suggestions
Type / Process / Description
* All *
Unused File
Excessive Writes
Failed System Calls
fseventsd (47)
fseventsd (47) failed performing lstat64 on (/private/var/db/sudo/ts/alejandrolucena)with: Permission denied.
sudo (90600)
kcm (90599)
Suboptimal Caching
No Detail
Input Filter
Instrument Detail

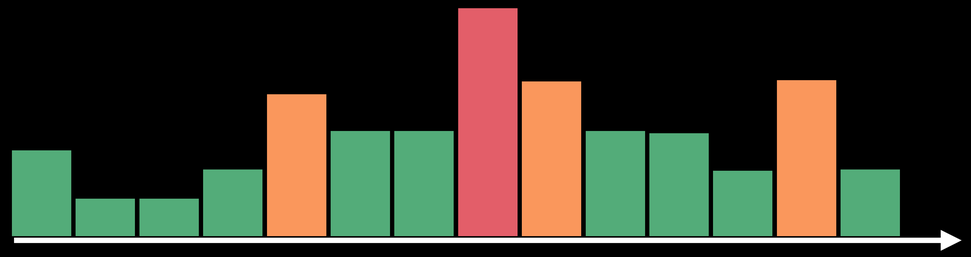

FileActivity
Alejandro's iMac Pro 〉All Processes
Run 3 of 3 | 00:00:42
Track Filter
All Tracks
Instruments
Threads
Duplicate
00:00.000
00:30.000
01:00.000
01:30.000
Filesystem Suggestions
Instrument
High
Moderate
Low
Filesystem Activity
Instrument
Logical Wri...
Logical Re...
Disk Usage
Instrument
Physical W...
Physical R...
Disk I/O Latency
Instrument
Physical I/...
Filesystem Suggestions 〉Filesystem Suggestions
Type / Process / Description
* All *
Failed System Calls
fseventsd (47)
fseventsd (47) failed performing lstat64 on (/private/var/db/sudo/ts/alejandrolucena)with: Permission denied.
sudo (90600)
kcm (90599)
No Detail
Input Filter
Instrument Detail

Compelling Stories Have Plots

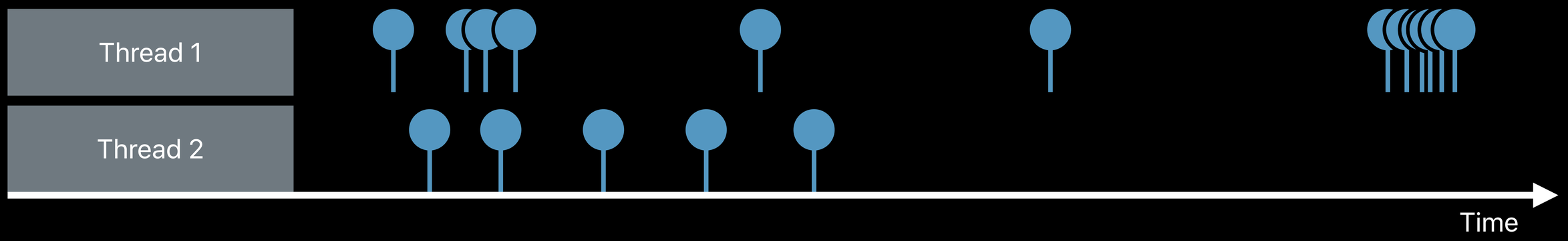
Make problems apparent

Graphs are the first summary

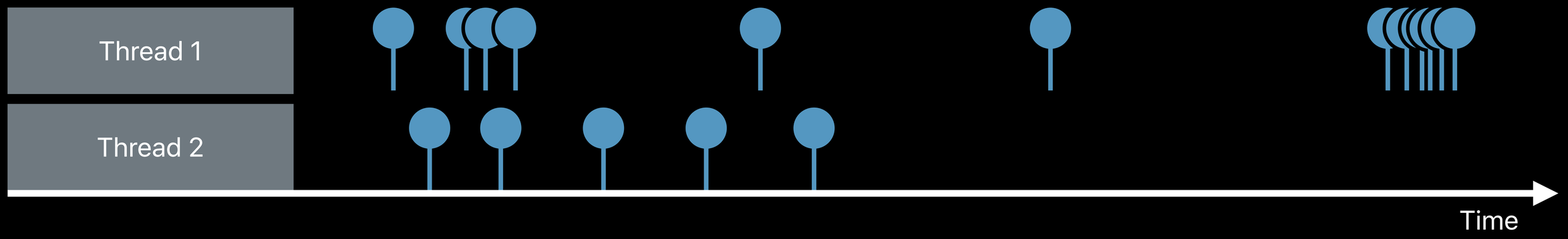
Display in terms of familiar concepts, metrics



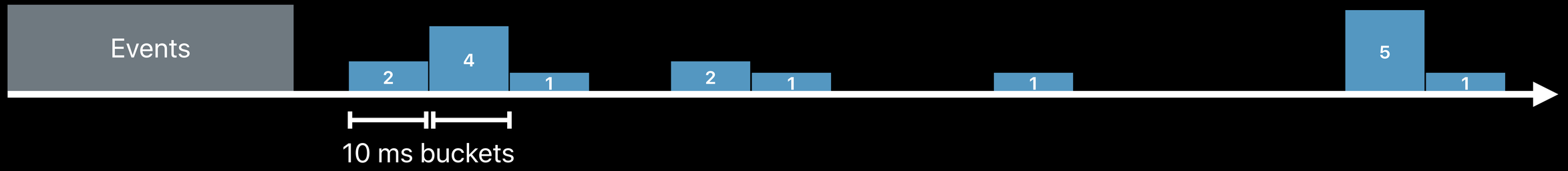
Summarizing Points



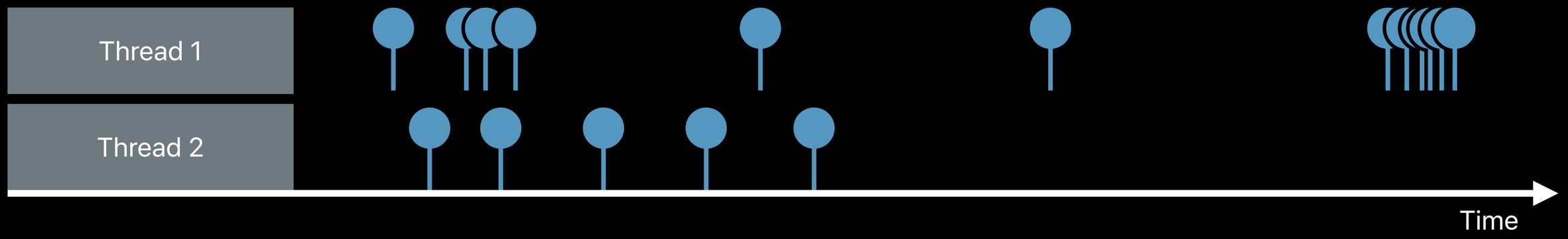
Summarizing Points

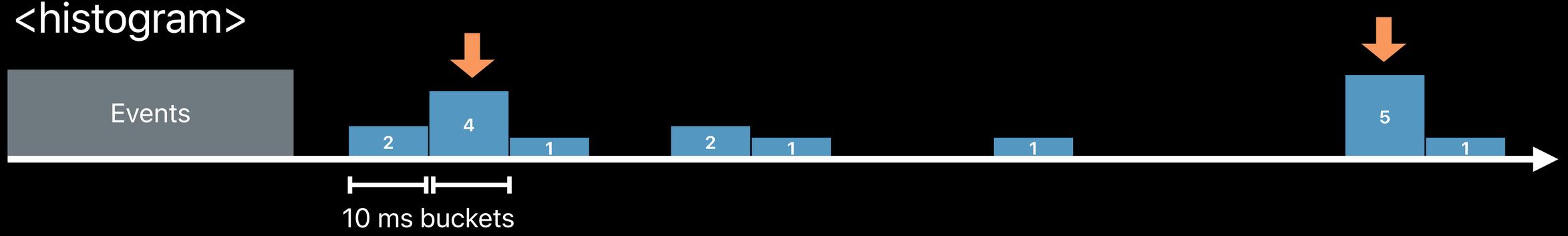


<histogram>

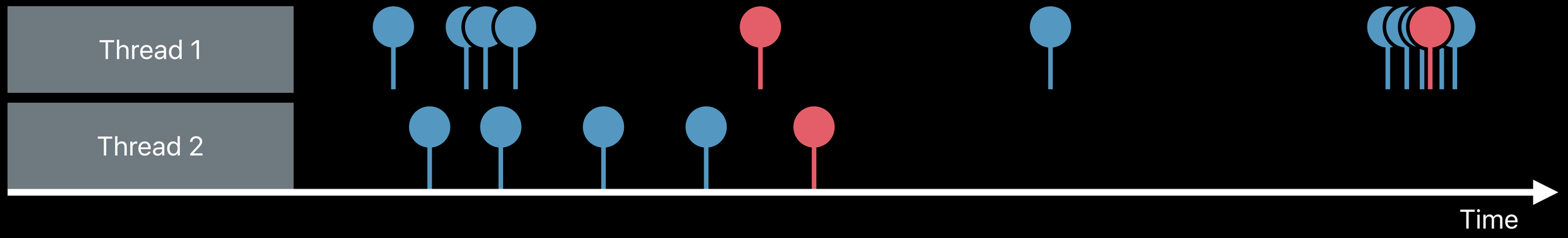


Summarizing Points

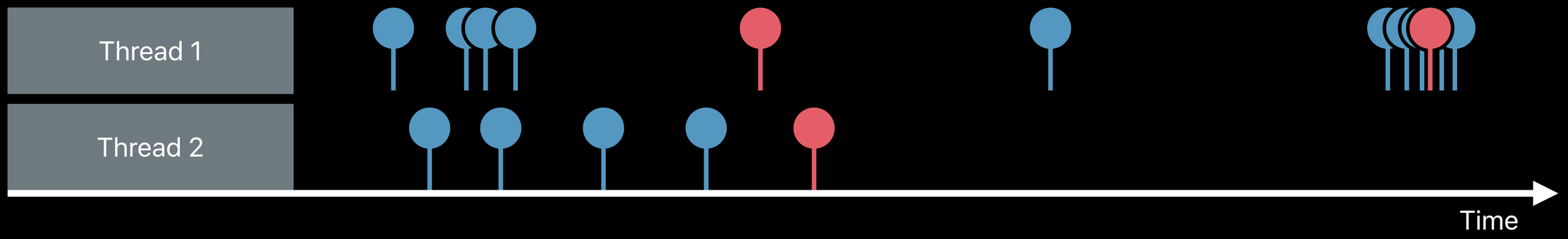




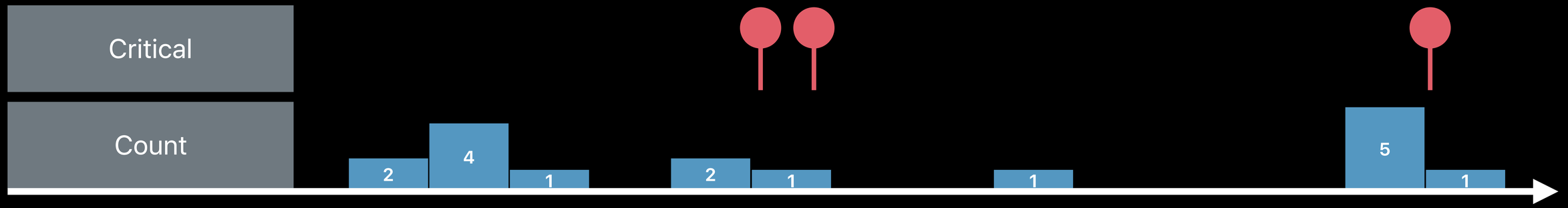
Summarizing Points



Summarizing Points



<plot> + <histogram>



Tabular Summaries

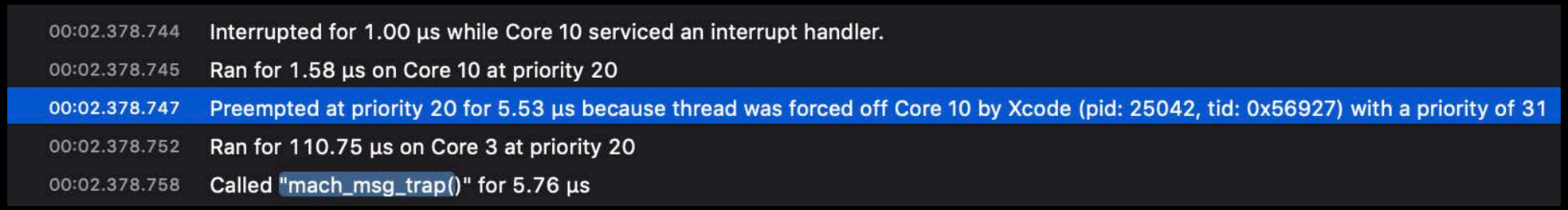
Define which metrics are important

- <aggregation>: <count>, <min>, <max>, <average>, <std-dev>
- Titles, order convey importance

Operation / Process / Path	Total Latency	Count	Bytes
▼ * All *	15.34 ms	28	480.00 KiB
▶ Data Read	10.01 ms	12	80.00 KiB
▶ Page In	4.77 ms	14	392.00 KiB
▶ Data Write	558.90 µs	2	8.00 KiB

Even More Detail

<narrative>: natural language, expressive types

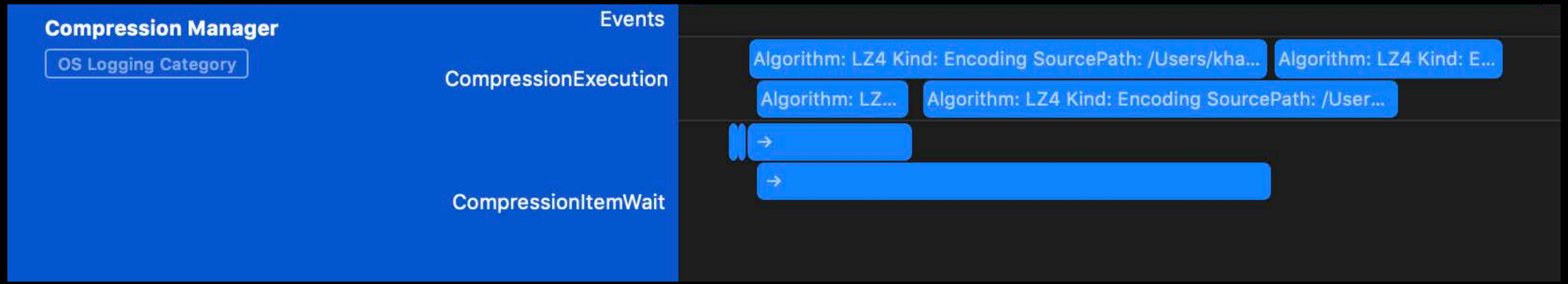



```
(set-column-narrative narrative
    "Interrupted for %duration% while %core% serviced an interrupt handler."
    ?duration ?core)
```

Displaying Intervals

Qualified, multiple plots

- <plot>, <plot-template>
- <qualified-by>, <instance-by>

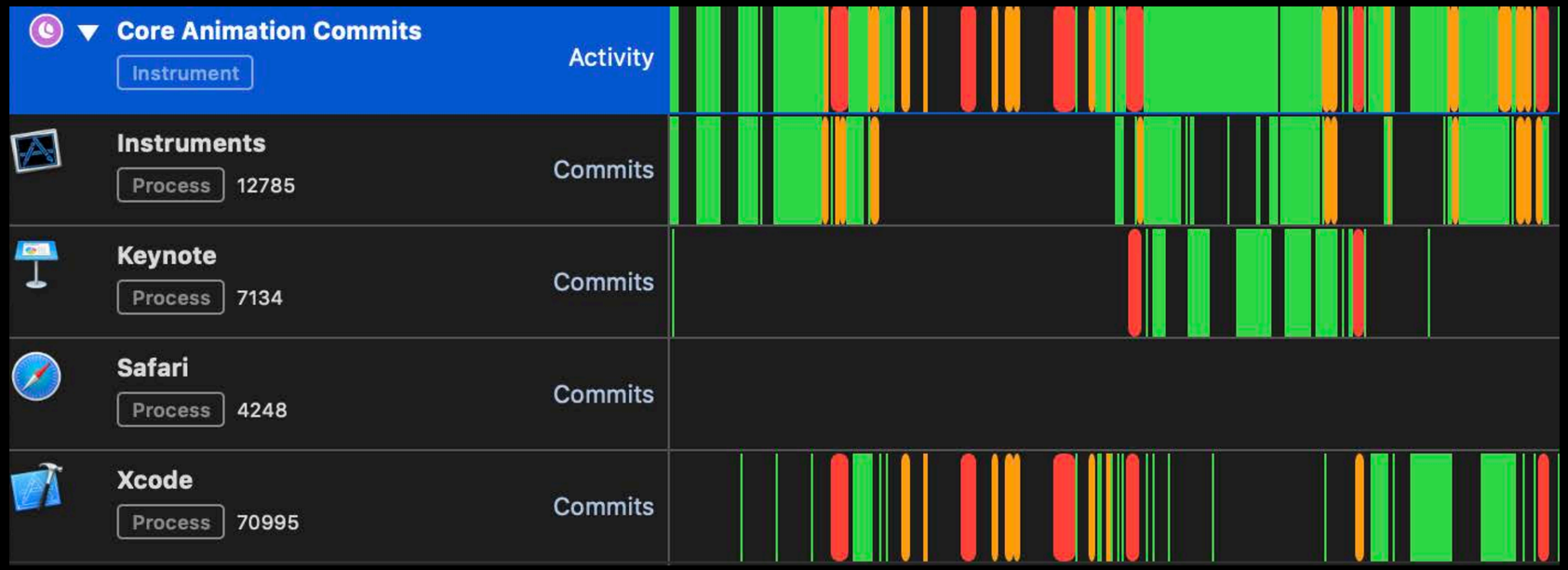


Displaying Intervals

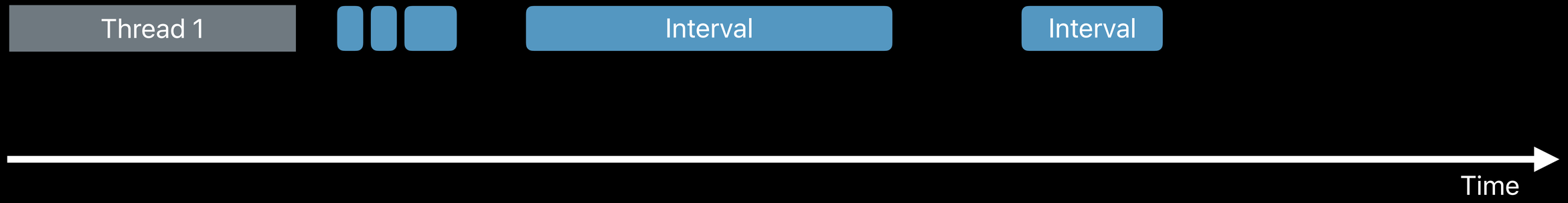
NEW

Hierarchies

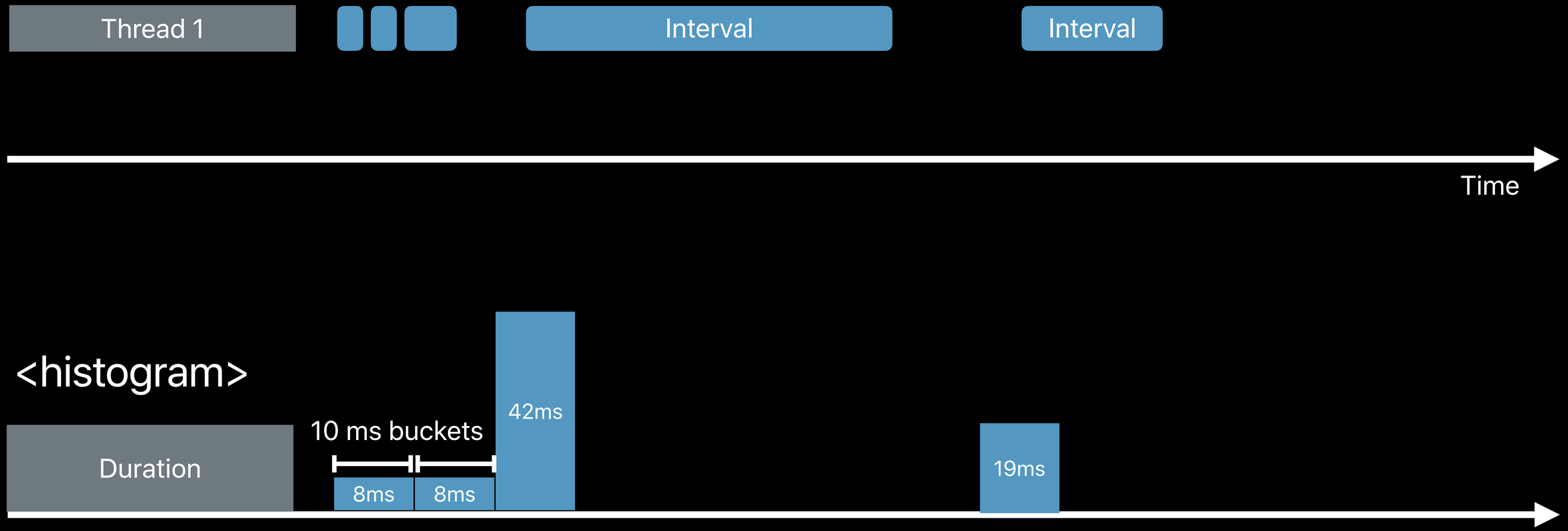
- <engineering-type-track>, <augmentation>
- <hierarchy>, <add-graph>



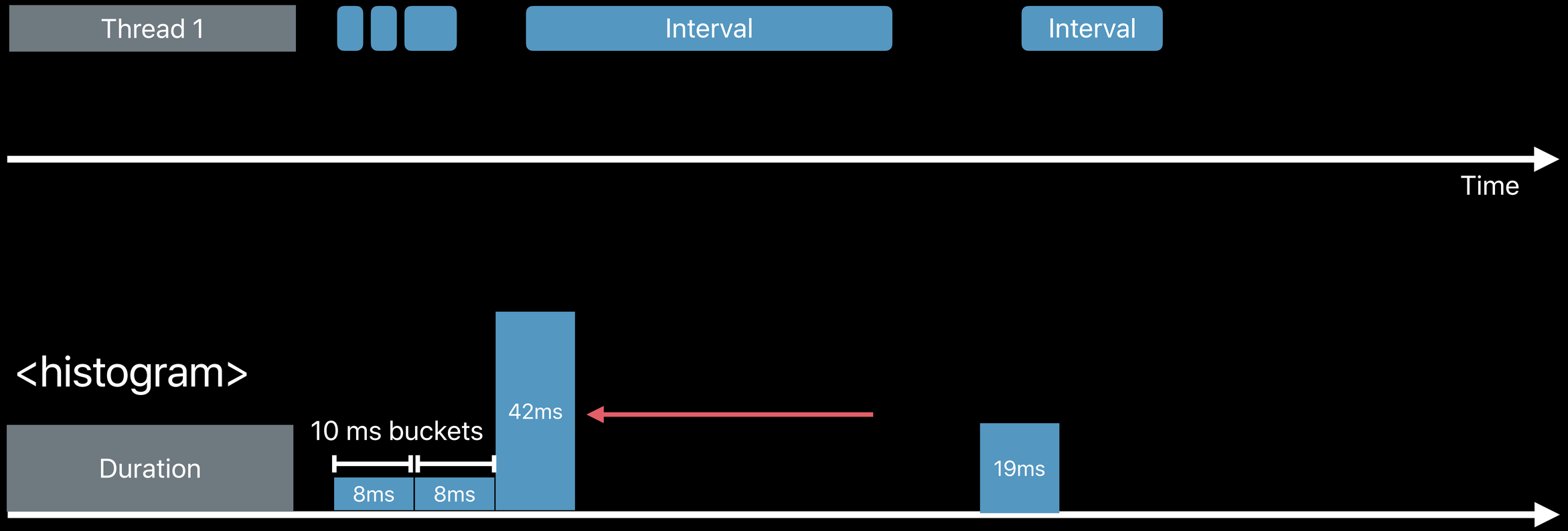
Summarizing Intervals



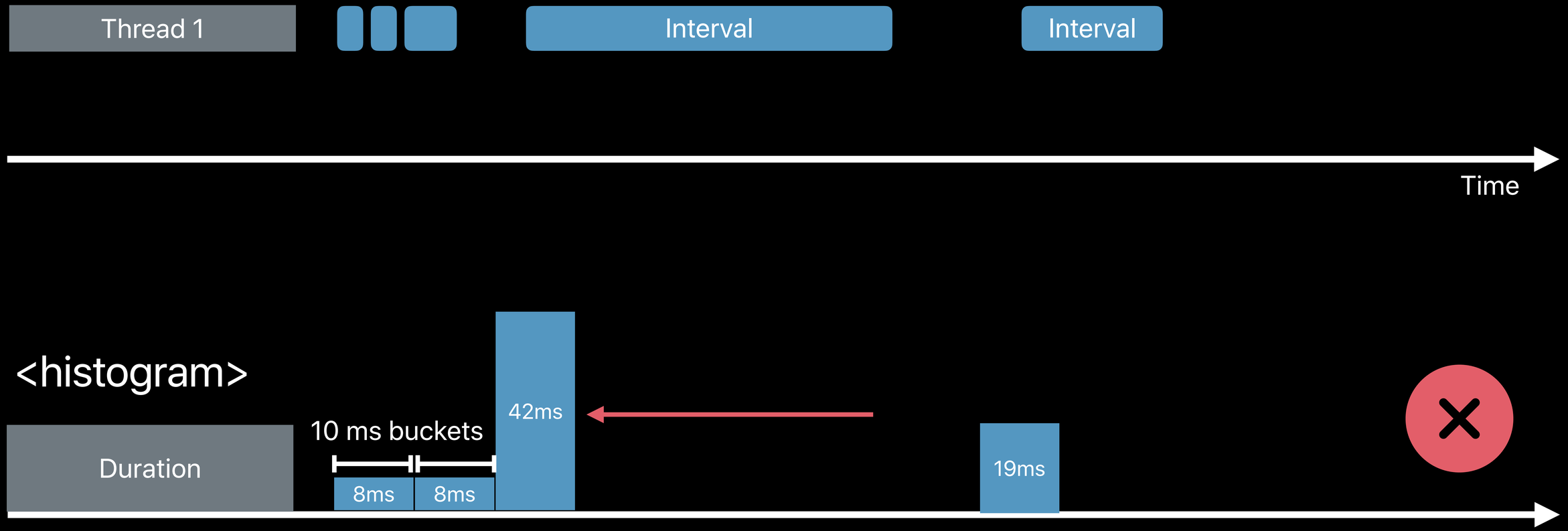
Summarizing Intervals



Summarizing Intervals

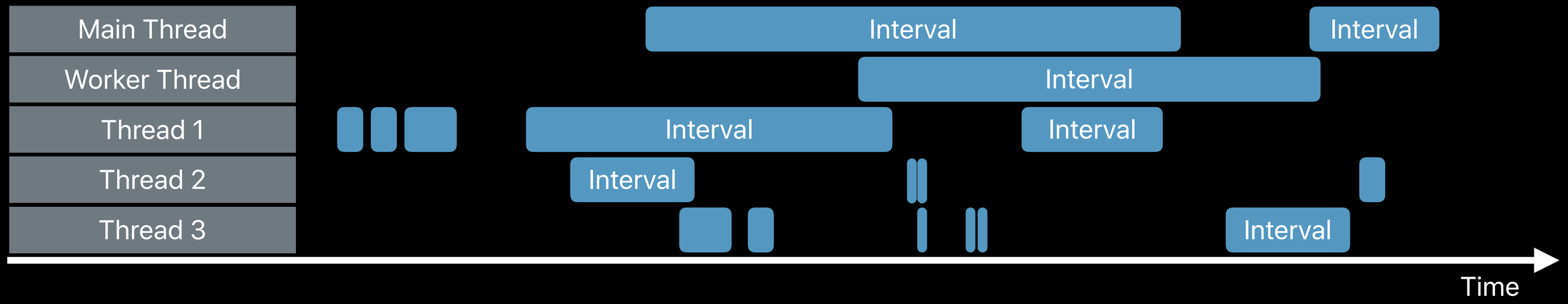


Summarizing Intervals

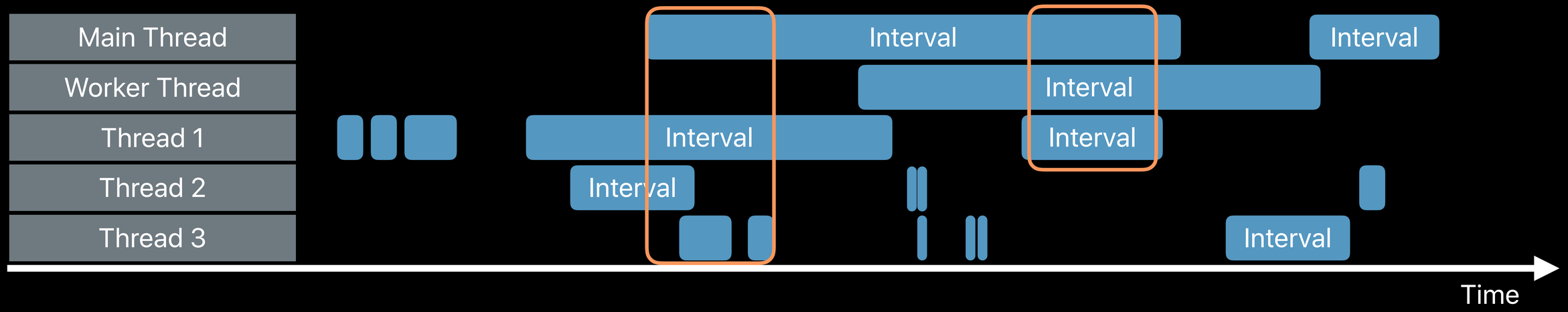


Don't graph time/duration on the Y-axis

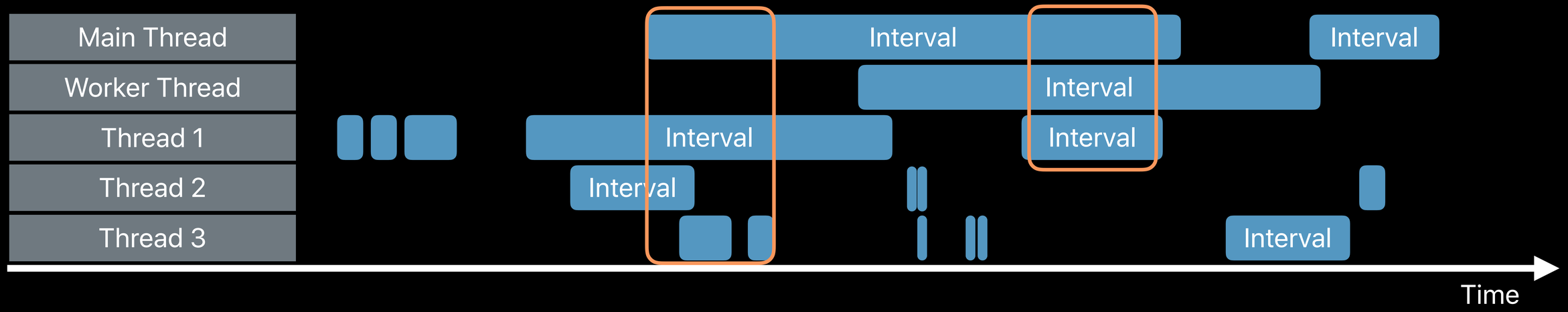
Summarizing Intervals



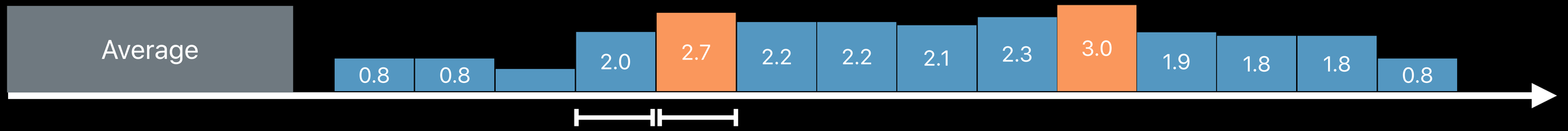
Summarizing Intervals



Summarizing Intervals

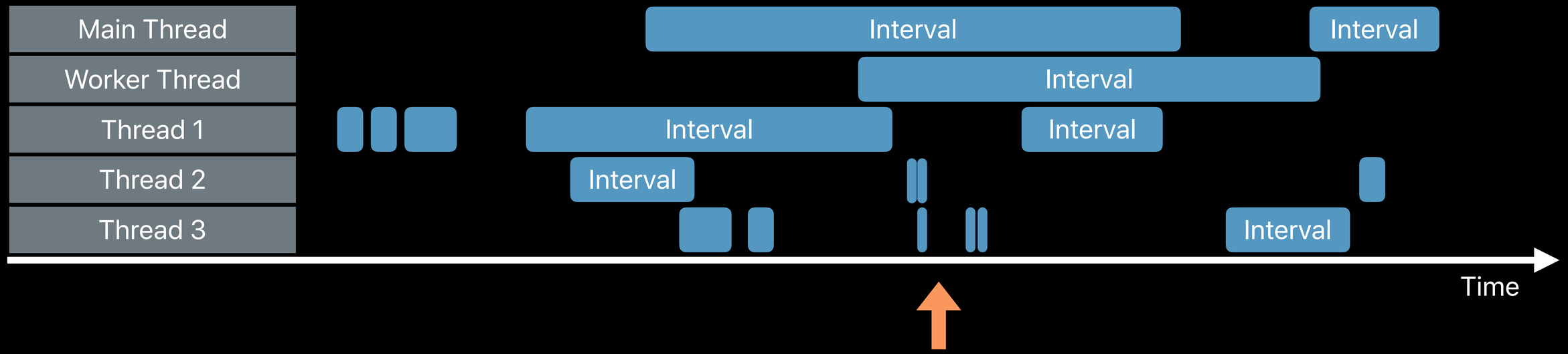


Quantized Utilization

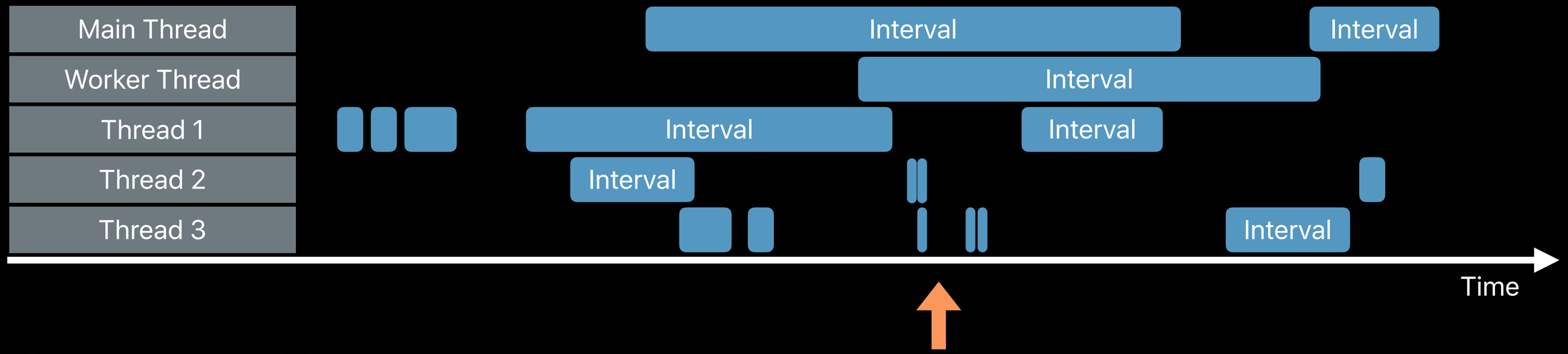


start	duration	utilization	severity
0:00:030.000	10ms	2.0	Low
0:00:040.000	10ms	2.7	Moderate

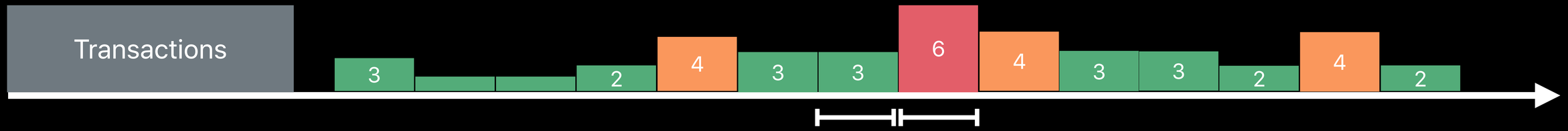
Summarizing Intervals



Summarizing Intervals

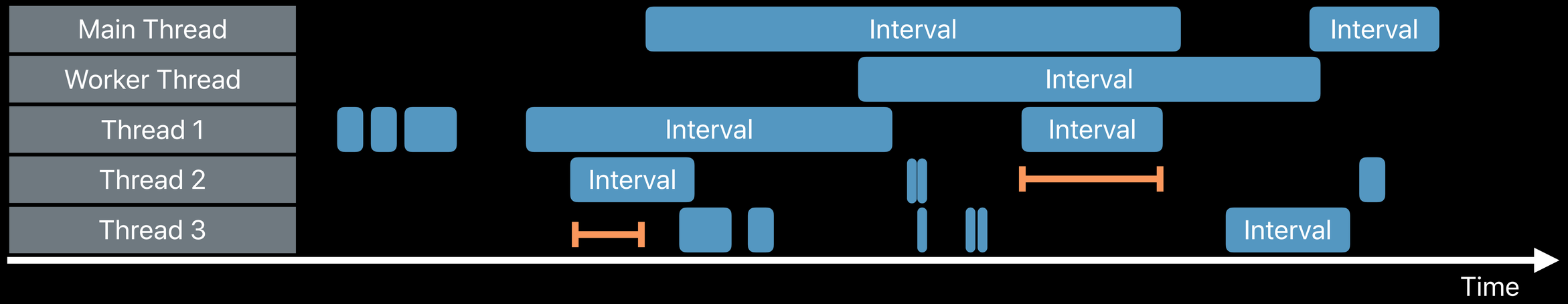


Quantized Activity

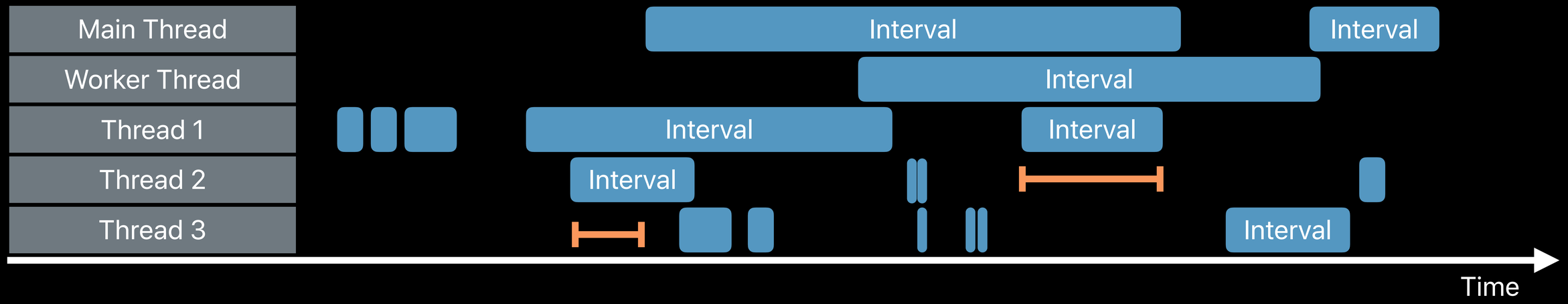


start	duration	unique-intervals	severity
0:00:060.000	10ms	3	Low
0:00:070.000	10ms	6	High

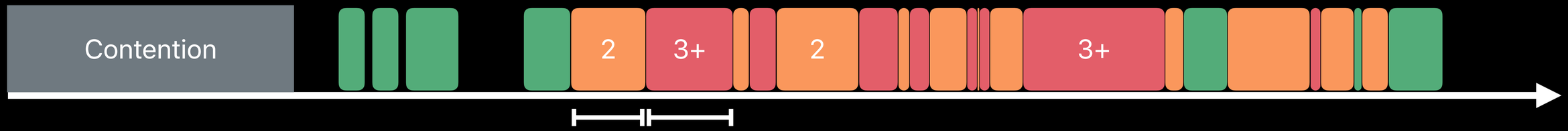
Summarizing Intervals



Summarizing Intervals

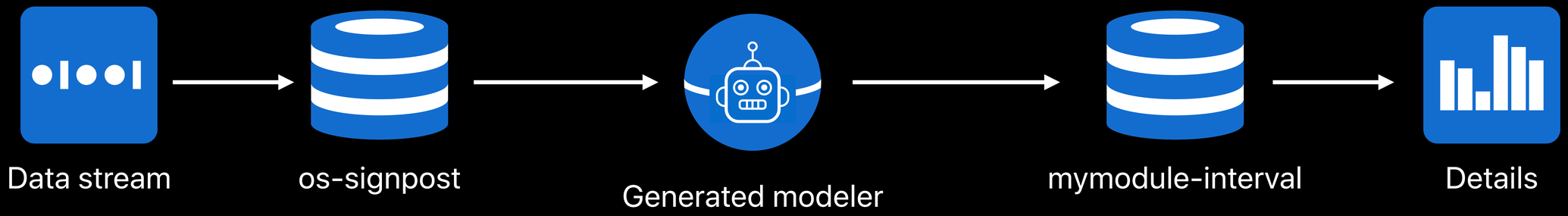


Overlap Plot

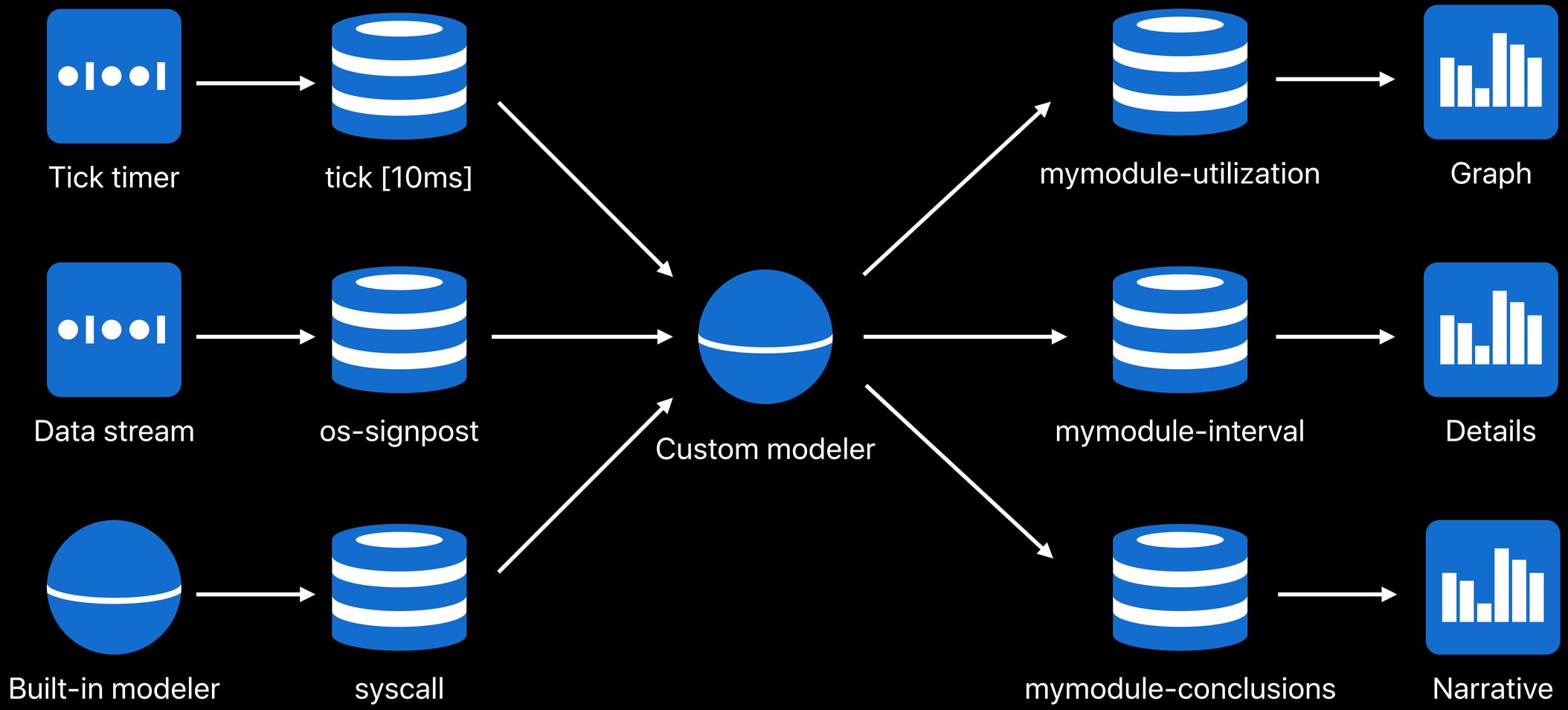


start	duration	description	severity
0:00:031.401	9.32ms	2	Moderate
0:00:040.721	11.41ms	3+	High

Modeler Outputs



Modeler Outputs



Demo

Crafting your story

Kacper Harasim, Software Engineer

Complete Profiling Experience

Users start with a template

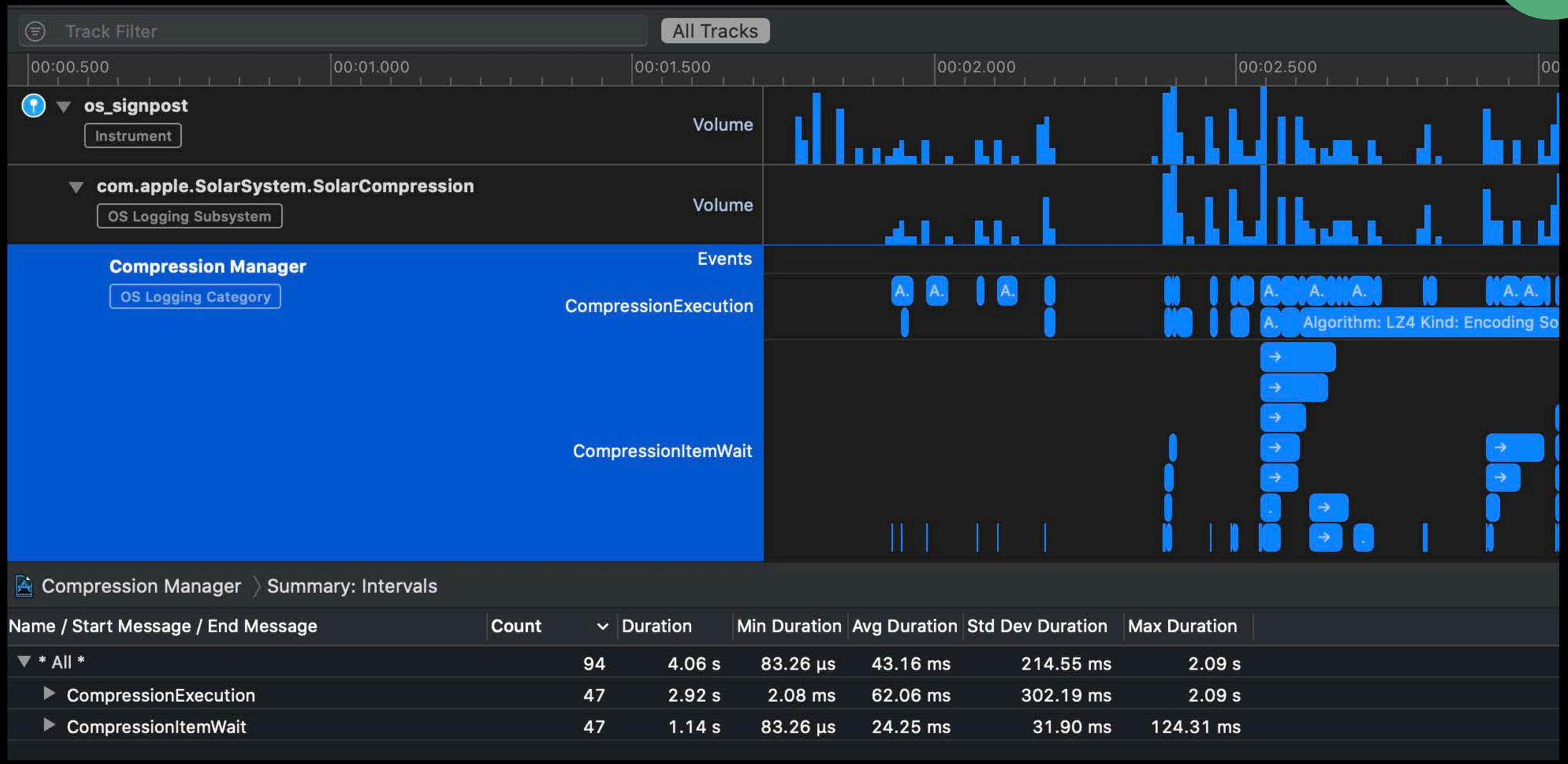
Utilize tools bundled with Instruments

Presented graphs should quickly draw users to problems

Detail views should lead user to root cause

Instruments 11 — Hierarchies

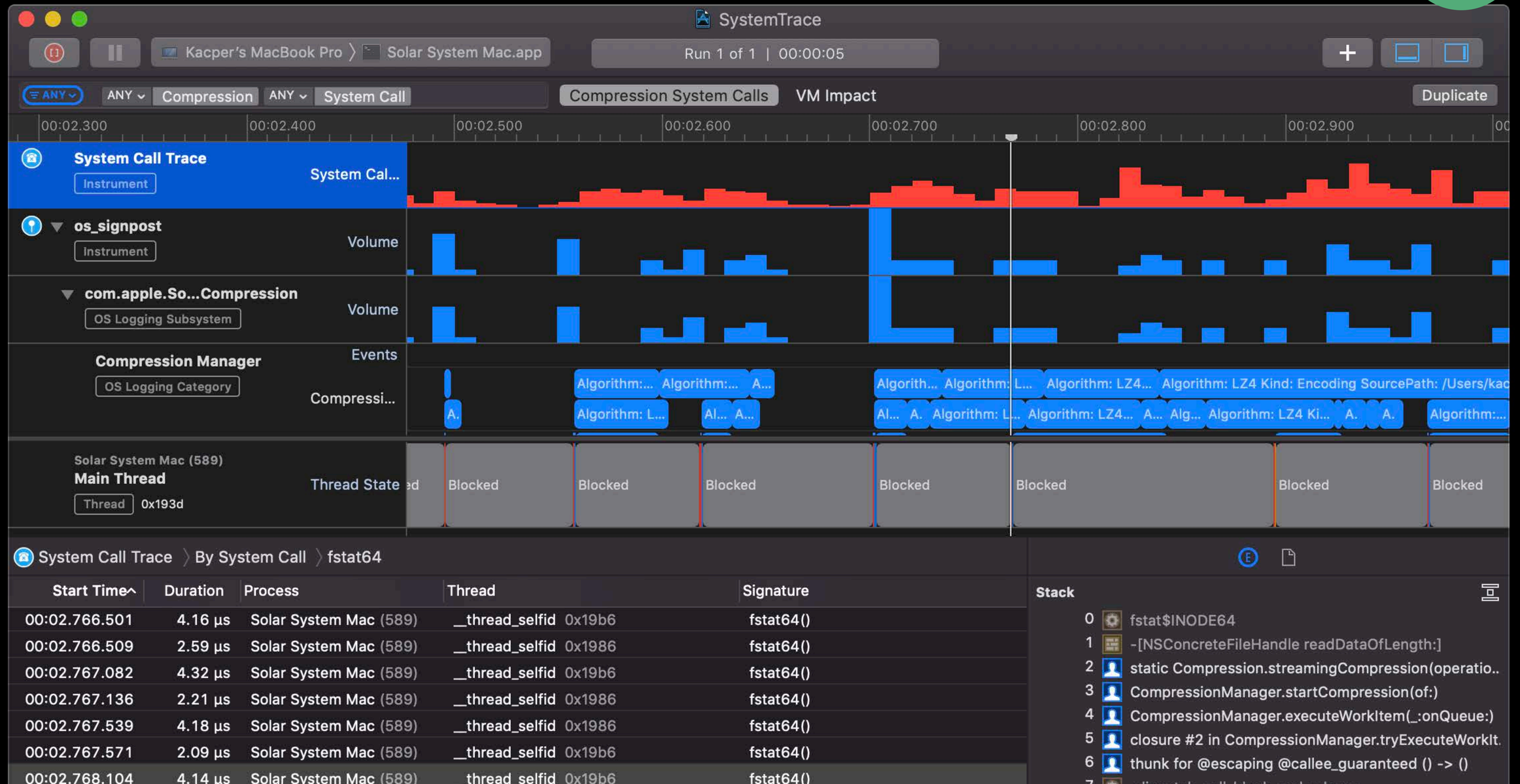
NEW



Name / Start Message / End Message	Count	Duration	Min Duration	Avg Duration	Std Dev Duration	Max Duration
* All *	94	4.06 s	83.26 µs	43.16 ms	214.55 ms	2.09 s
CompressionExecution	47	2.92 s	2.08 ms	62.06 ms	302.19 ms	2.09 s
CompressionItemWait	47	1.14 s	83.26 µs	24.25 ms	31.90 ms	124.31 ms

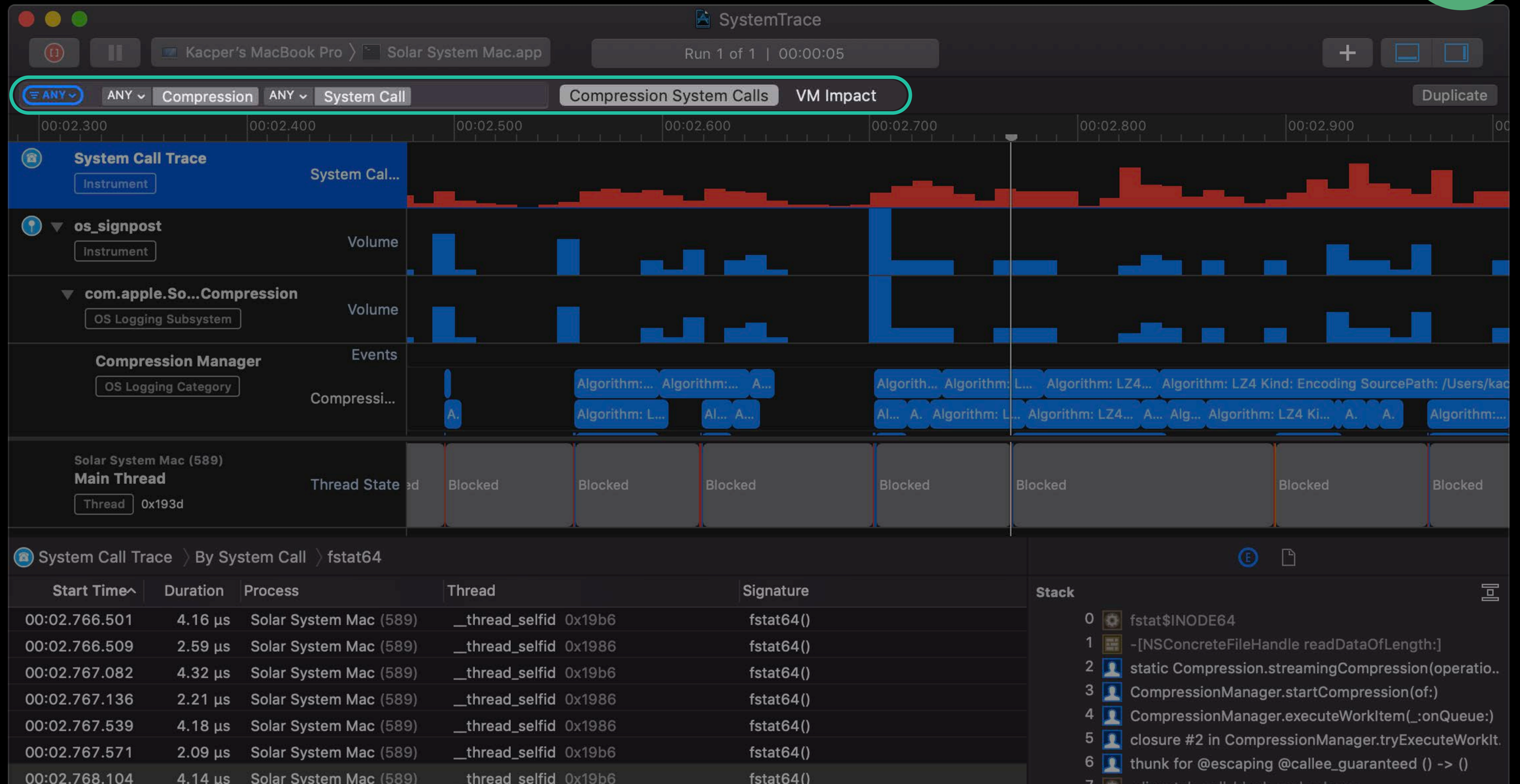
Instruments 11 — Custom Track Scopes

NEW



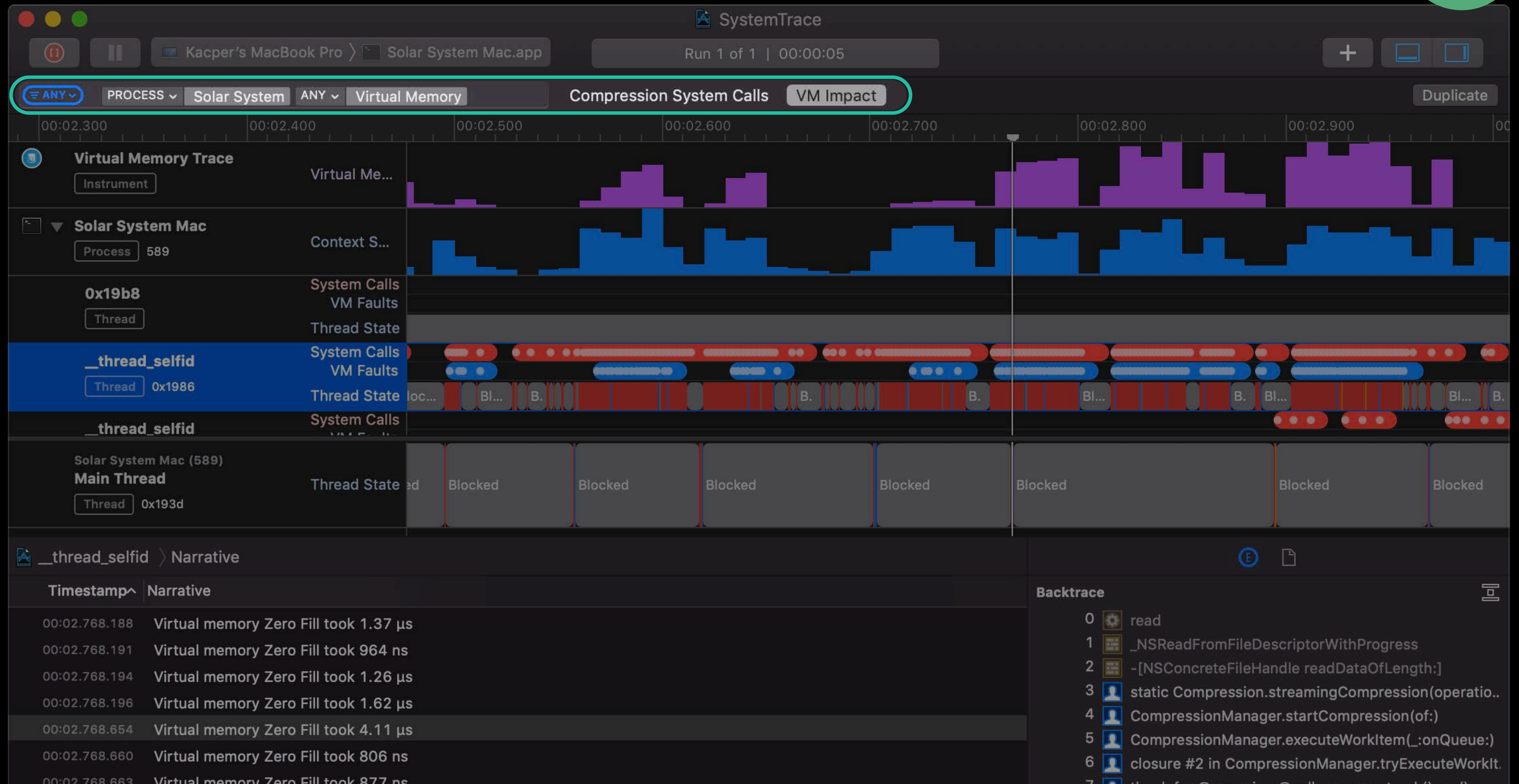
Instruments 11 — Custom Track Scopes

NEW



Instruments 11 — Custom Track Scopes

NEW



Summary

Opportunity to tell your story, educate, and catch easy mistakes

When something goes wrong, tools provide an answer

Increase confidence and trust in your library

More Information

developer.apple.com/wwdc19/414

Modeling in Custom Instruments	Friday, 3:20
Creating Custom Instruments	WWDC18
Measuring Performance Using Logging	WWDC18


WWDC19