Modernizing GCD Usage

How to stay on core

Session 706

Daniel Chimene, Core Darwin
Daniel A. Steffen, Core Darwin
Pierre Habouzit, Core Darwin

dispatch_async

dispatch_queue_create

DispatchQueue.concurrentPerform

dispatch_after

DispatchQueue.async

dispatch_sync

DispatchQueue.sync

dispatch_activate

DispatchSource.activate

dispatch_source_create

DispatchSource.setEventHandler

dispatch_once

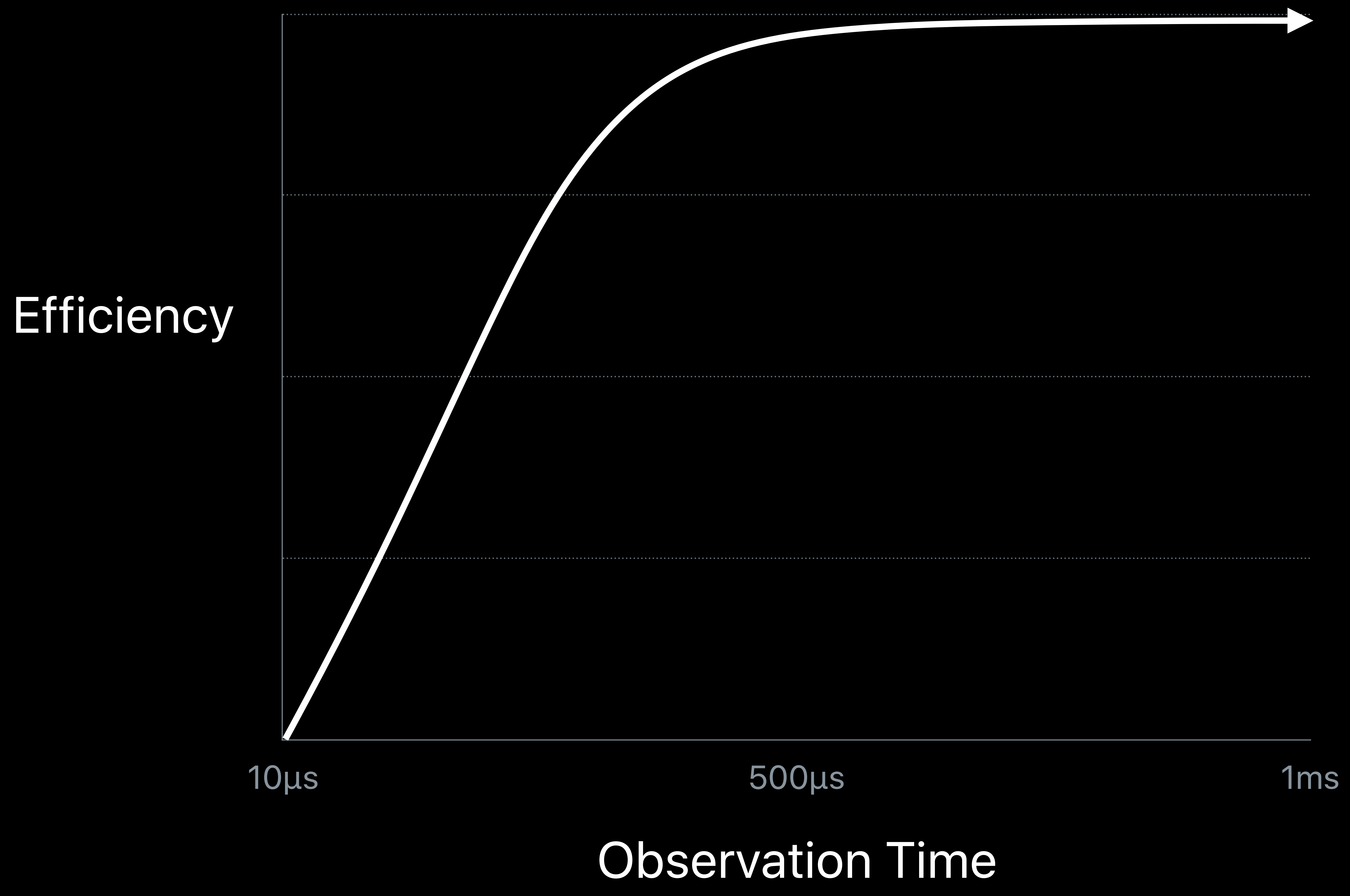
dispatch_apply

DispatchWorkItem.notify

A4 A5 A6 A7 A8 A9 A10

Efficiency Through Observation

Going off core during an operation reduces efficiency



1.3x

faster after combining queue hierarchies

Parallelism and concurrency

Parallelism and concurrency

Using GCD for concurrency

Parallelism and concurrency

Using GCD for concurrency

Unified Queue Identity

Parallelism and concurrency

Using GCD for concurrency

Unified Queue Identity

Finding problem spots

Parallelism

Simultaneous execution of closely related computations

Concurrency

Composition of independently executed tasks

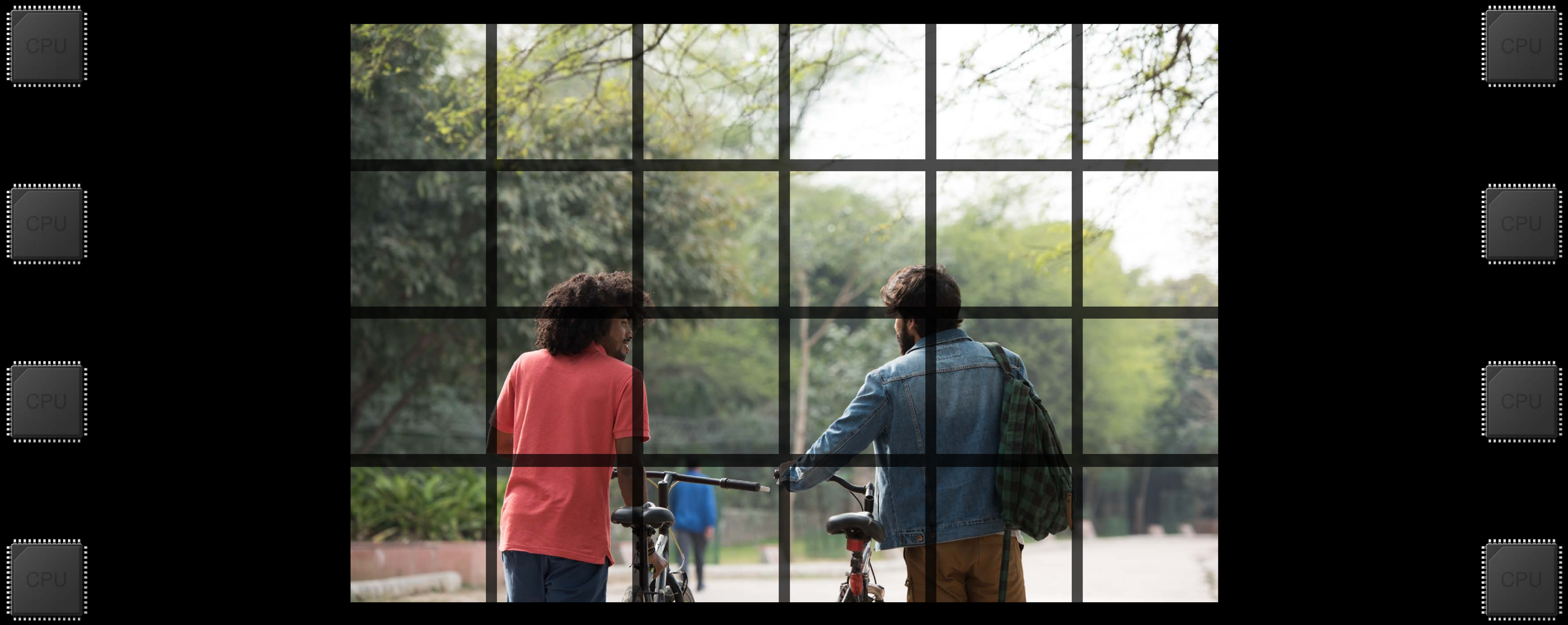
Parallelism

Parallelism

Simultaneous execution of closely related computations

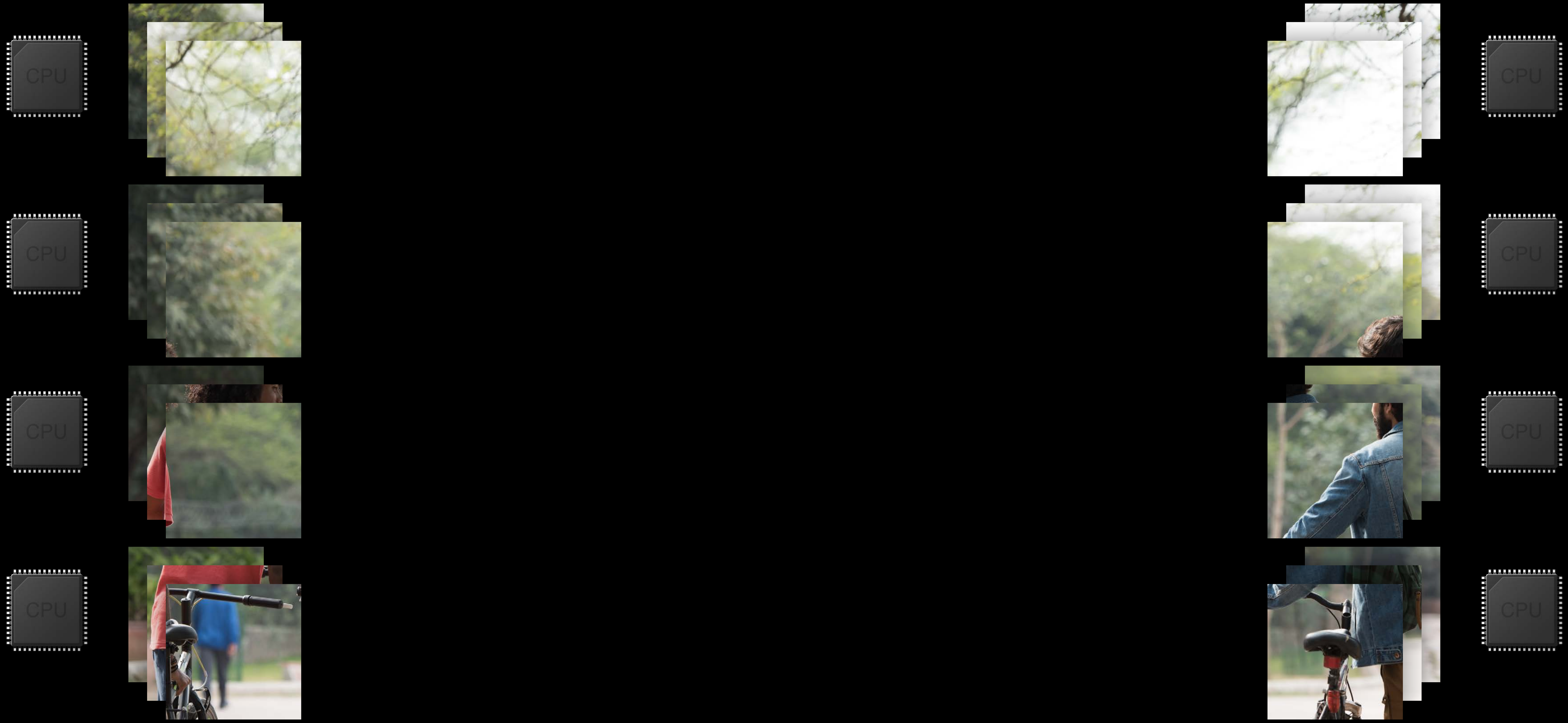
Parallelism

Simultaneous execution of closely related computations



Parallelism

Simultaneous execution of closely related computations



Take Advantage of System Frameworks

Accelerate

Metal 2

Core ML

Core Animation

Parallelism with GCD

Express explicit parallelism with `DispatchQueue.concurrentPerform`

Parallel for-loop—calling thread participates in the computation

More efficient than many asyncs to a concurrent queue

```swift
DispatchQueue.concurrentPerform(1000) { i in /* iteration i */ }
```

Parallelism with GCD

Express explicit parallelism with `DispatchQueue.concurrentPerform`

Parallel for-loop—calling thread participates in the computation

More efficient than many asyncs to a concurrent queue

```
DispatchQueue.concurrentPerform(1000) { i in /* iteration i */ }
```

Parallelism with GCD

NEW

Express explicit parallelism with `DispatchQueue.concurrentPerform`

Parallel for-loop—calling thread participates in the computation

More efficient than many asyncs to a concurrent queue

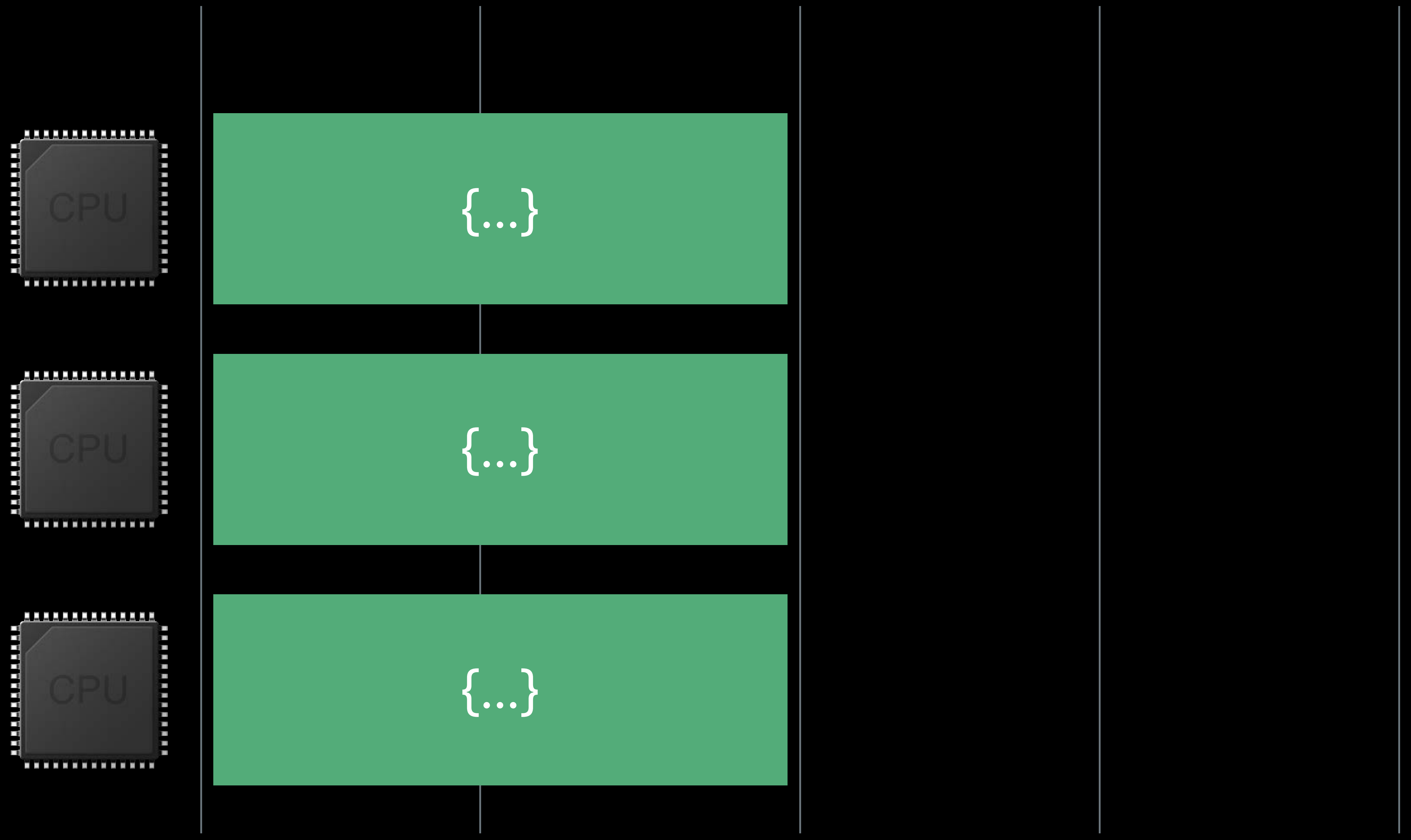
```
DispatchQueue.concurrentPerform(1000) { i in /* iteration i */ }

dispatch_apply(DISPATCH_APPLY_AUTO, 1000, ^(size_t i){ /* iteration i */ })
```

`DISPATCH_APPLY_AUTO` deploys back to macOS 10.9, iOS 7.0

Dynamic Resource Availability

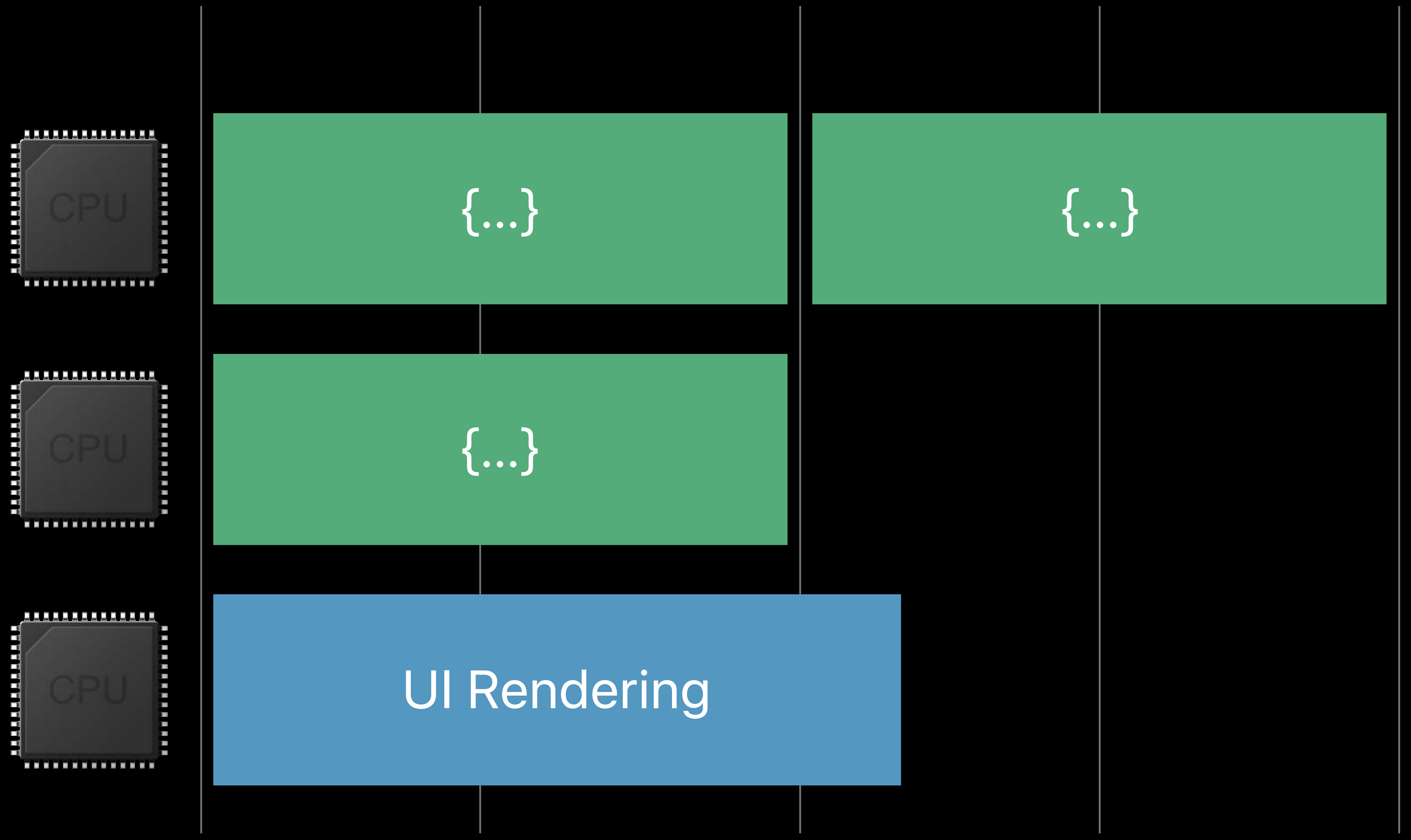
Choosing an iteration count



```
DispatchQueue.concurrentPerform(3) { i in /* iteration i */ }
```

Dynamic Resource Availability

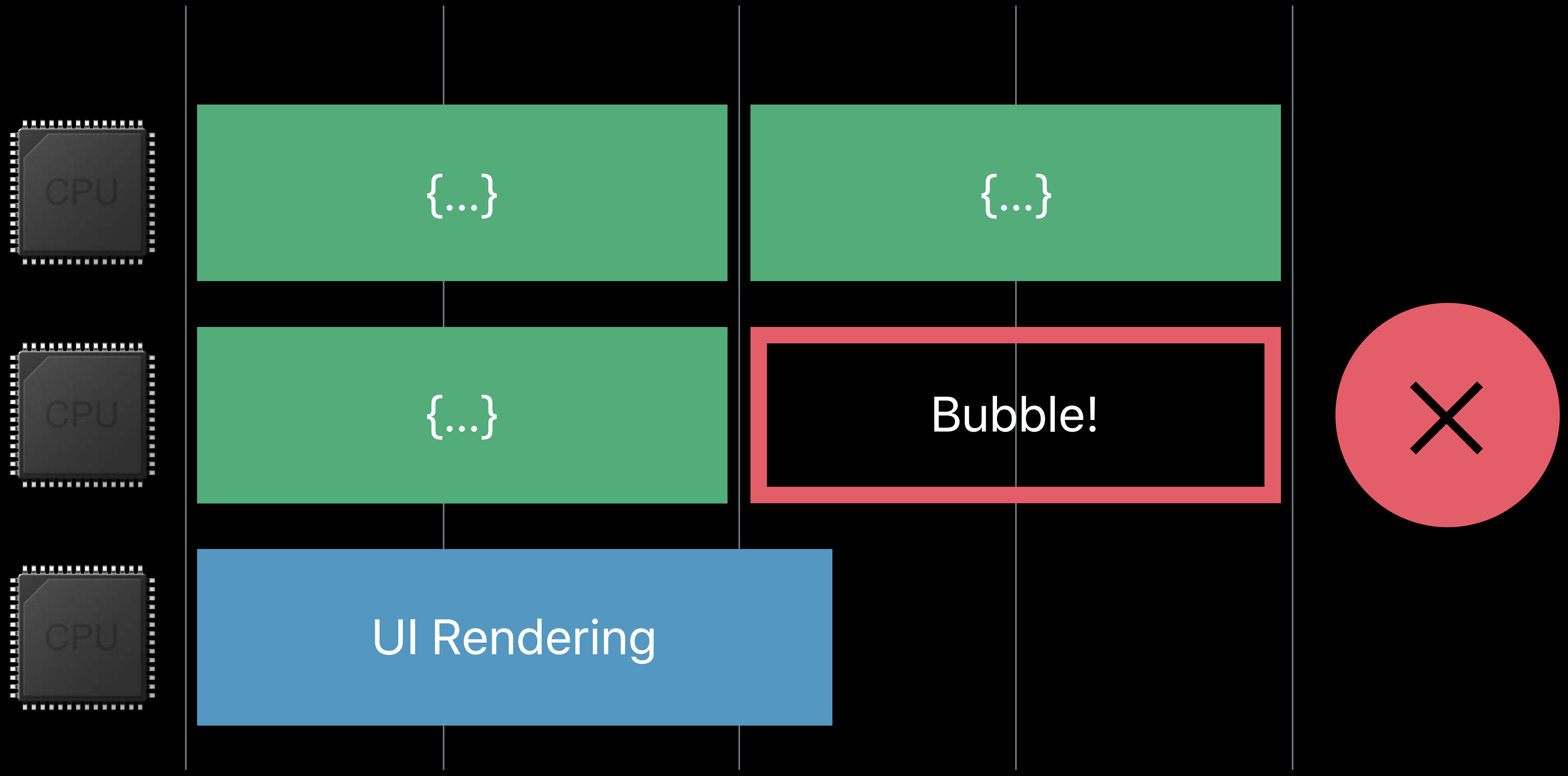
Choosing an iteration count



```
DispatchQueue.concurrentPerform(3) { i in /* iteration i */ }
```

Dynamic Resource Availability

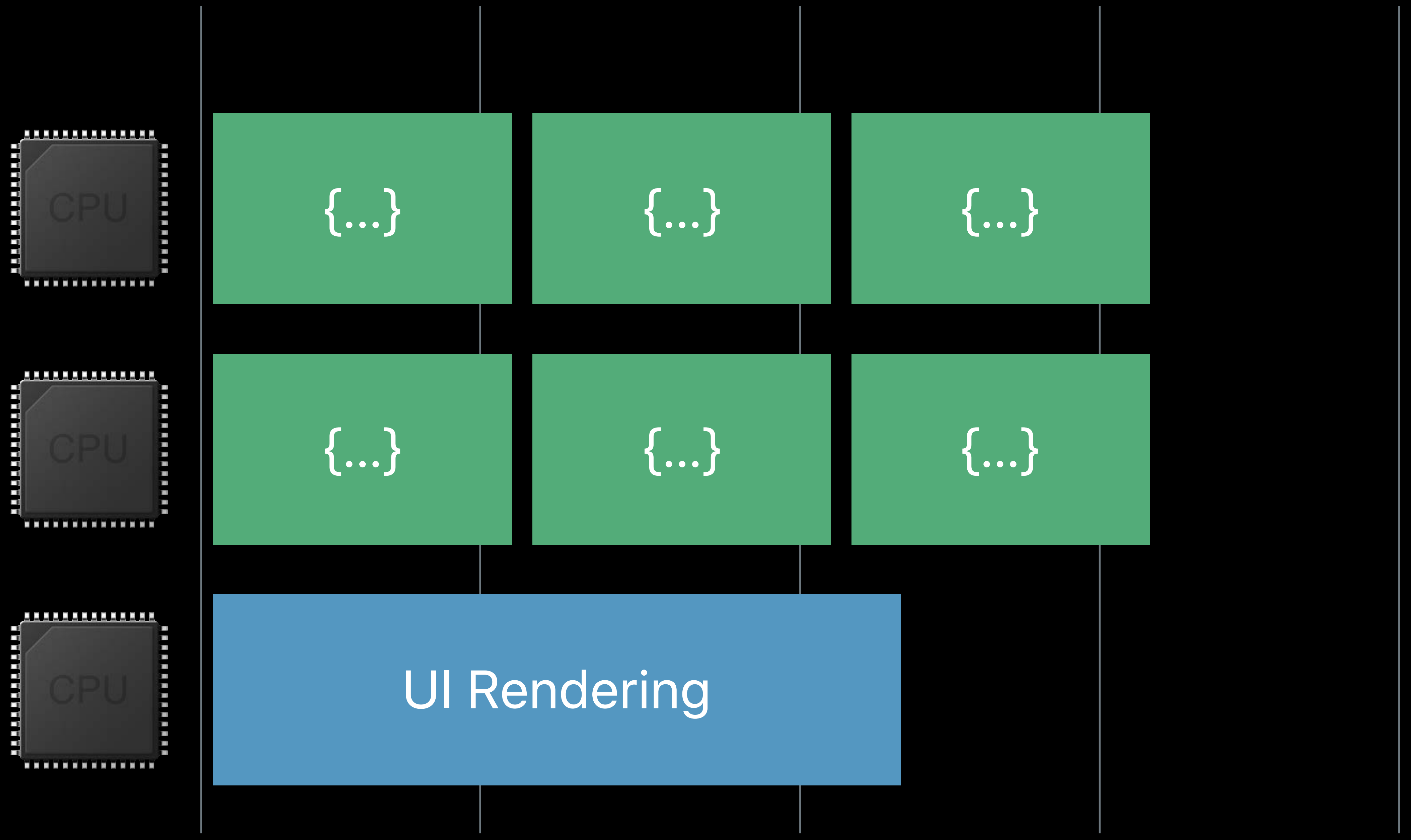
Choosing an iteration count



```
DispatchQueue.concurrentPerform(3) { i in /* iteration i */ }
```

Dynamic Resource Availability

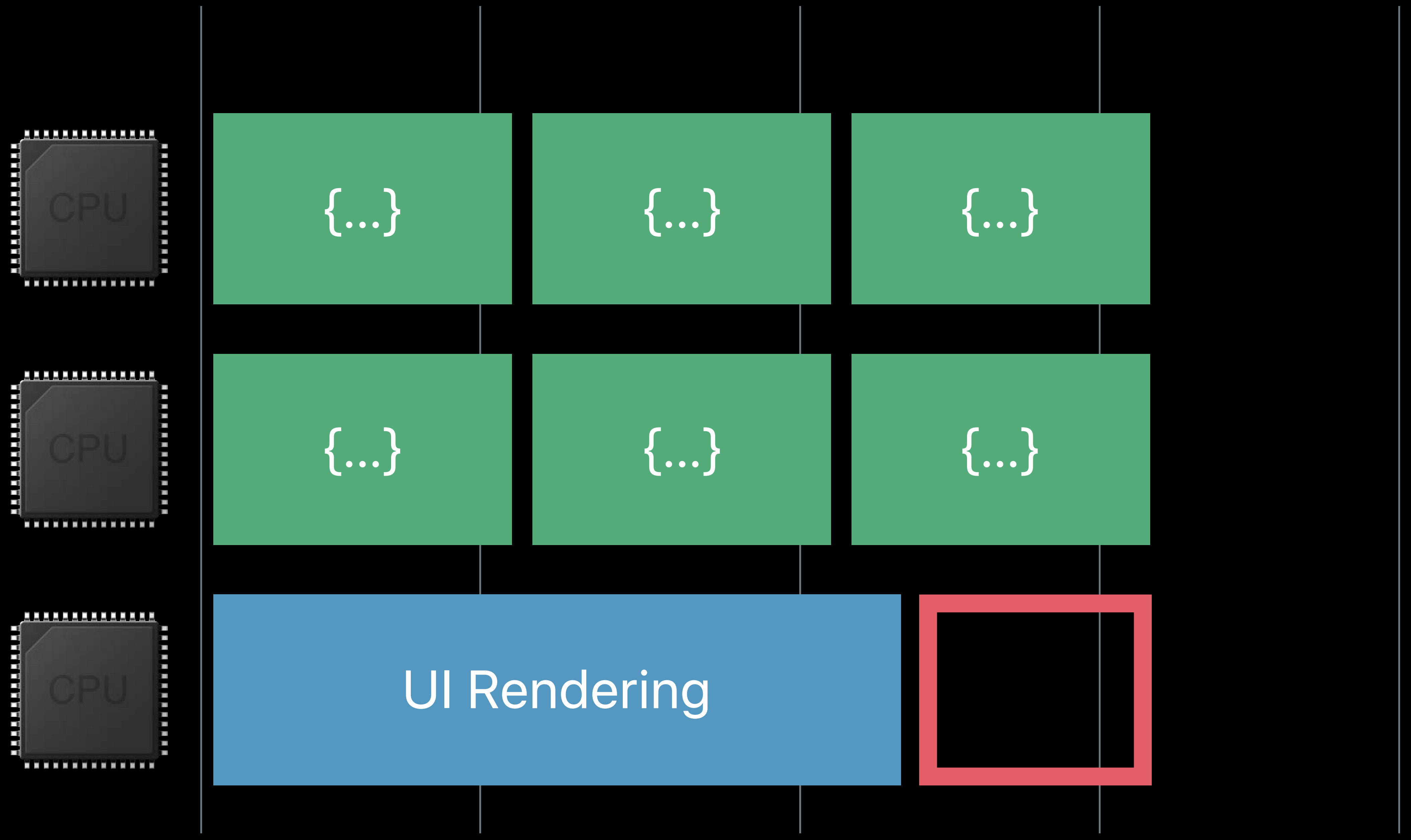
Choosing an iteration count



```
DispatchQueue.concurrentPerform(6) { i in /* iteration i */ }
```

Dynamic Resource Availability

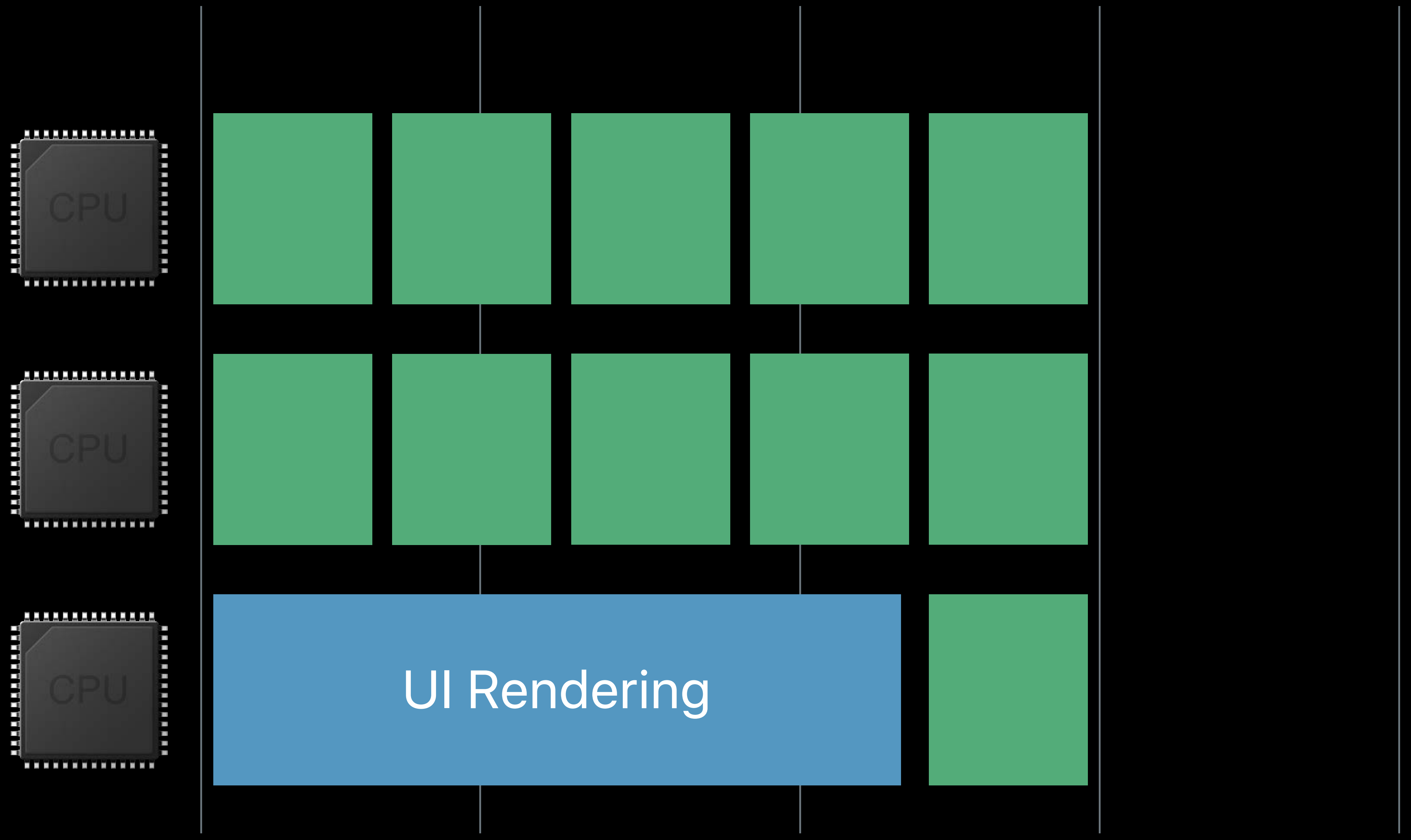
Choosing an iteration count



```
DispatchQueue.concurrentPerform(6) { i in /* iteration i */ }
```

Dynamic Resource Availability

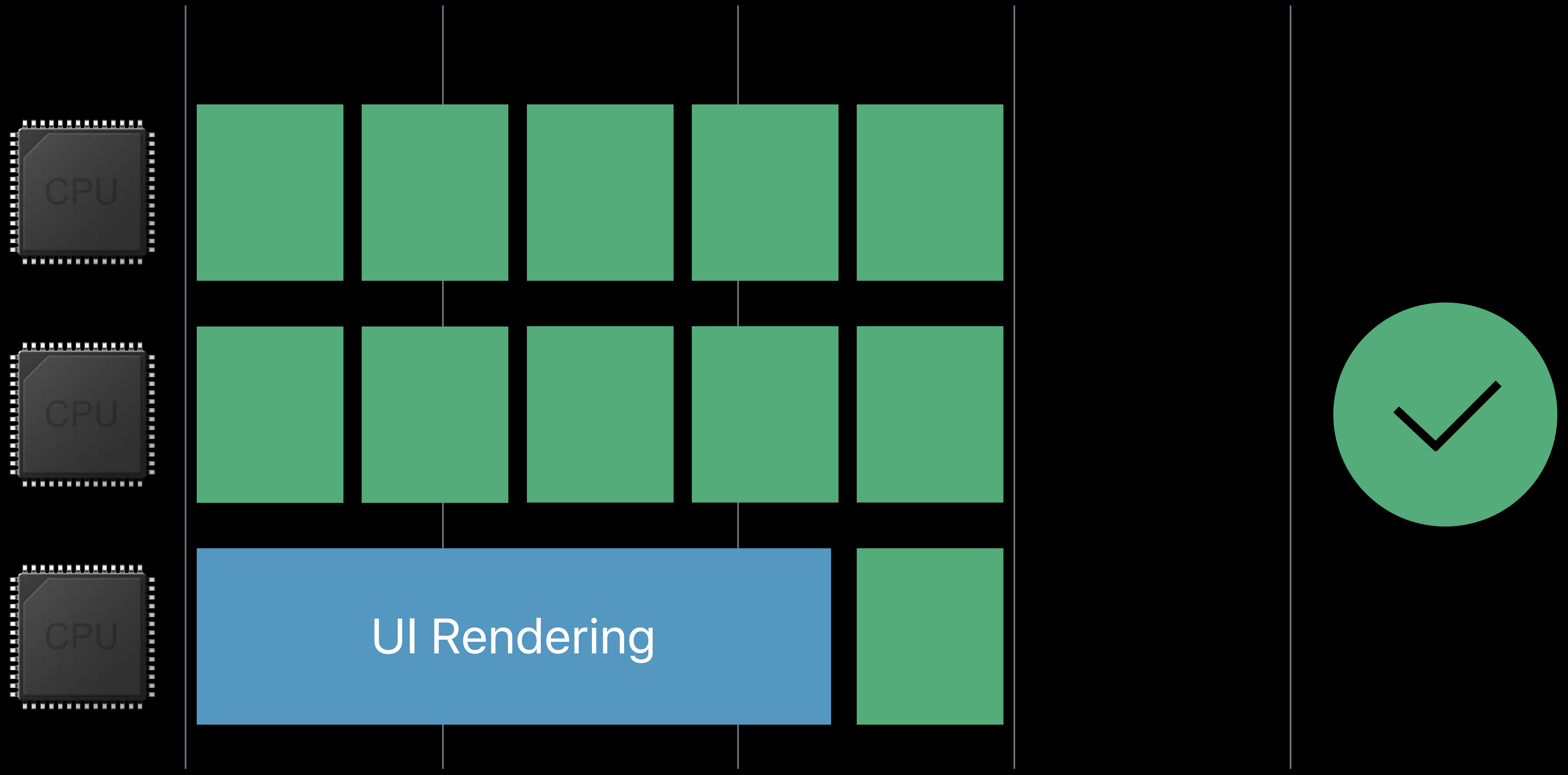
Choosing an iteration count



```
DispatchQueue.concurrentPerform(11) { i in /* iteration i */ }
```

Dynamic Resource Availability

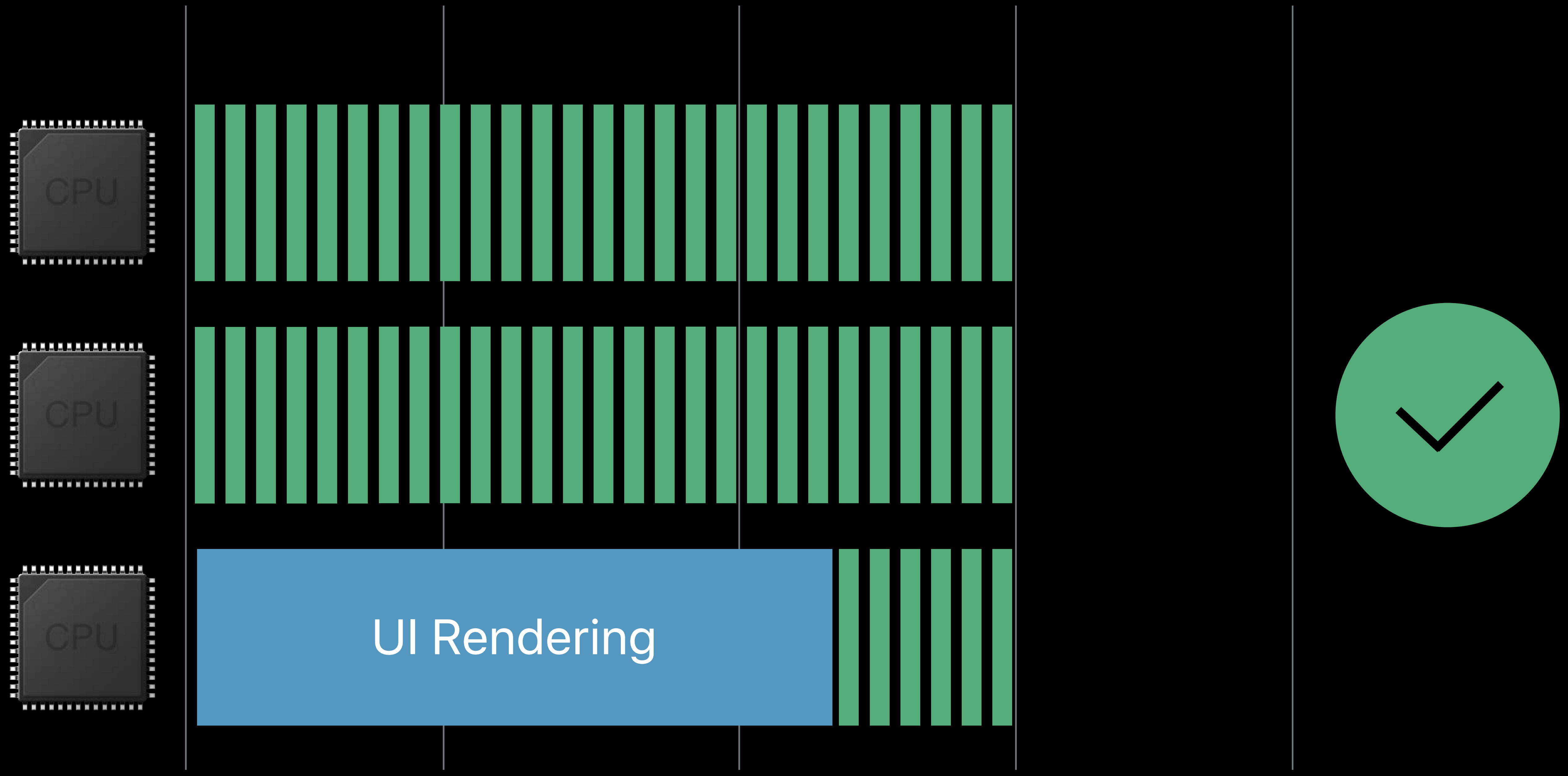
Choosing an iteration count



```
DispatchQueue.concurrentPerform(11) { i in /* iteration i */ }
```

Dynamic Resource Availability

Choosing an iteration count



```
DispatchQueue.concurrentPerform(1000) { i in /* iteration i */ }
```

Parallelism

Leverage system frameworks

Use `DispatchQueue.concurrentPerform`

Consider dynamic availability

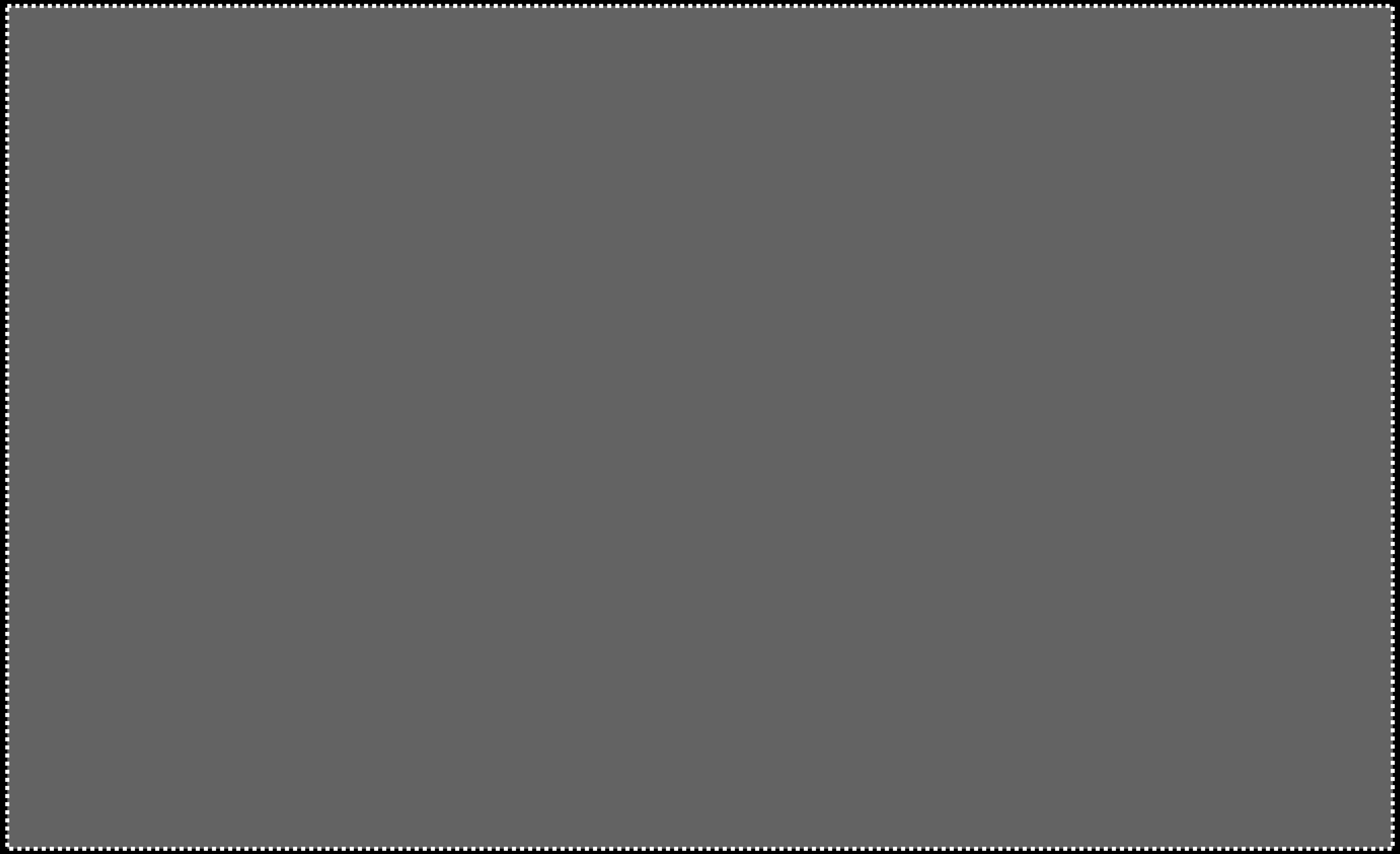
Concurrency

Concurrency

Composition of independently executed tasks

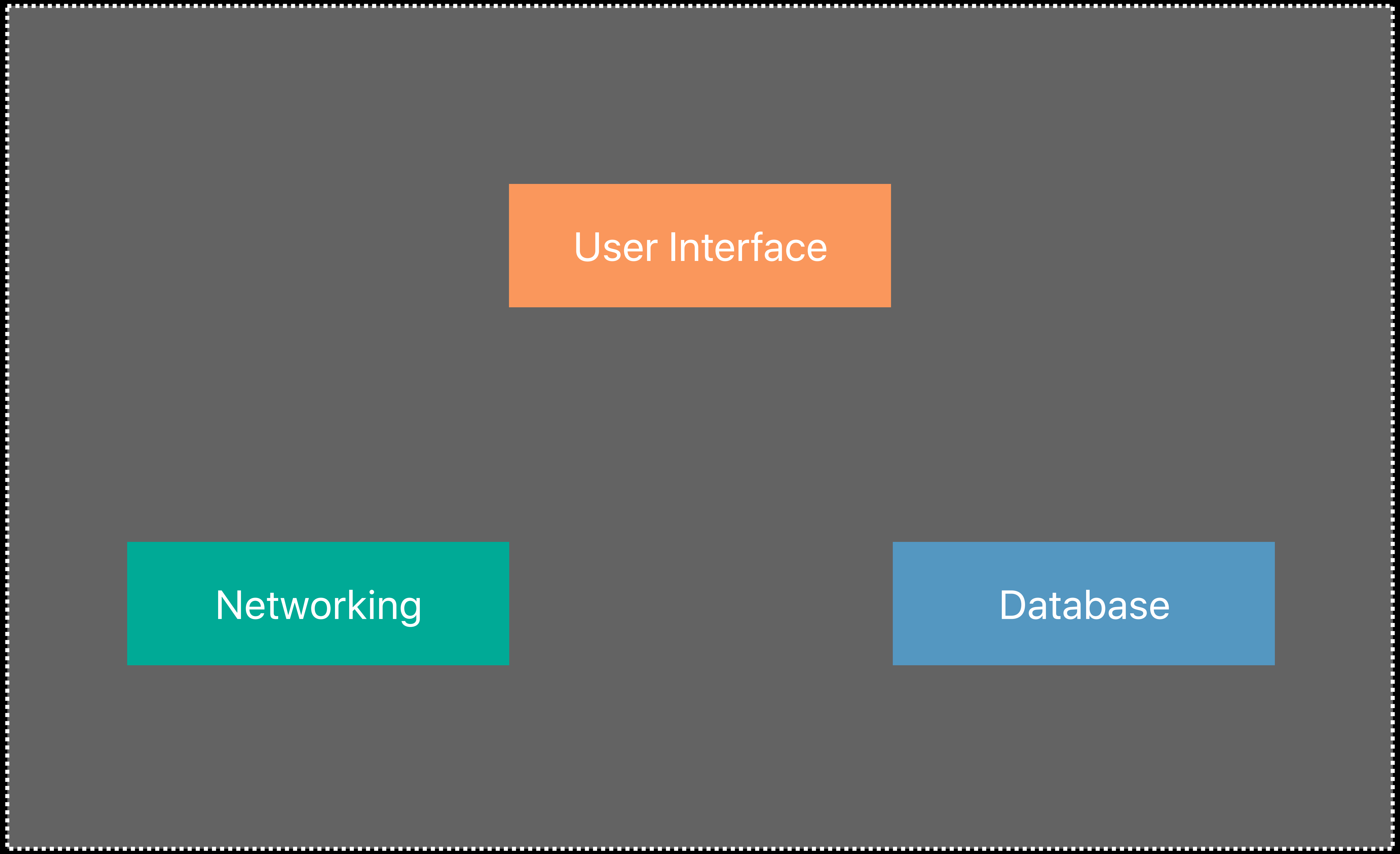
Concurrency

Composition of independently executed tasks



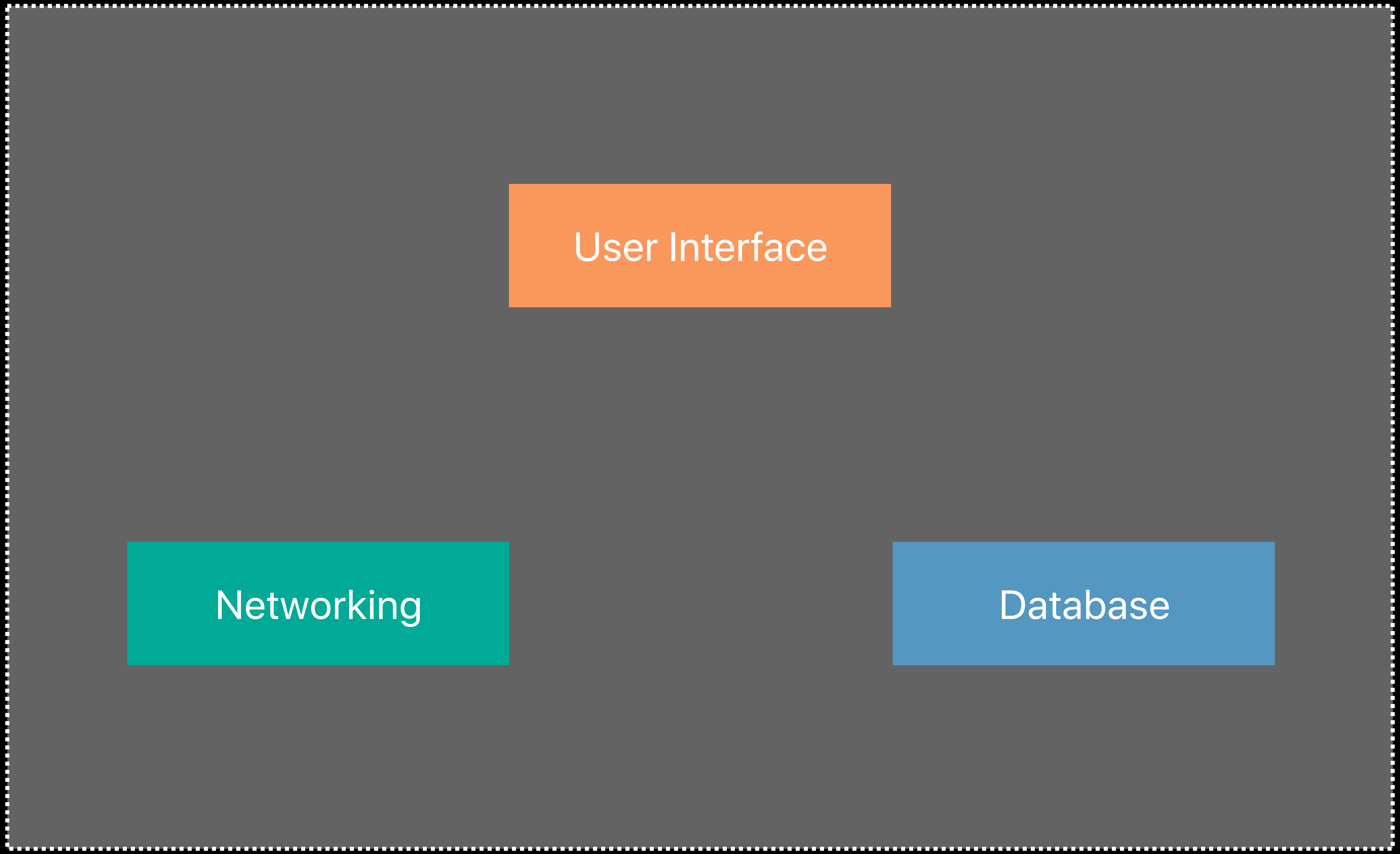
Concurrency

Composition of independently executed tasks



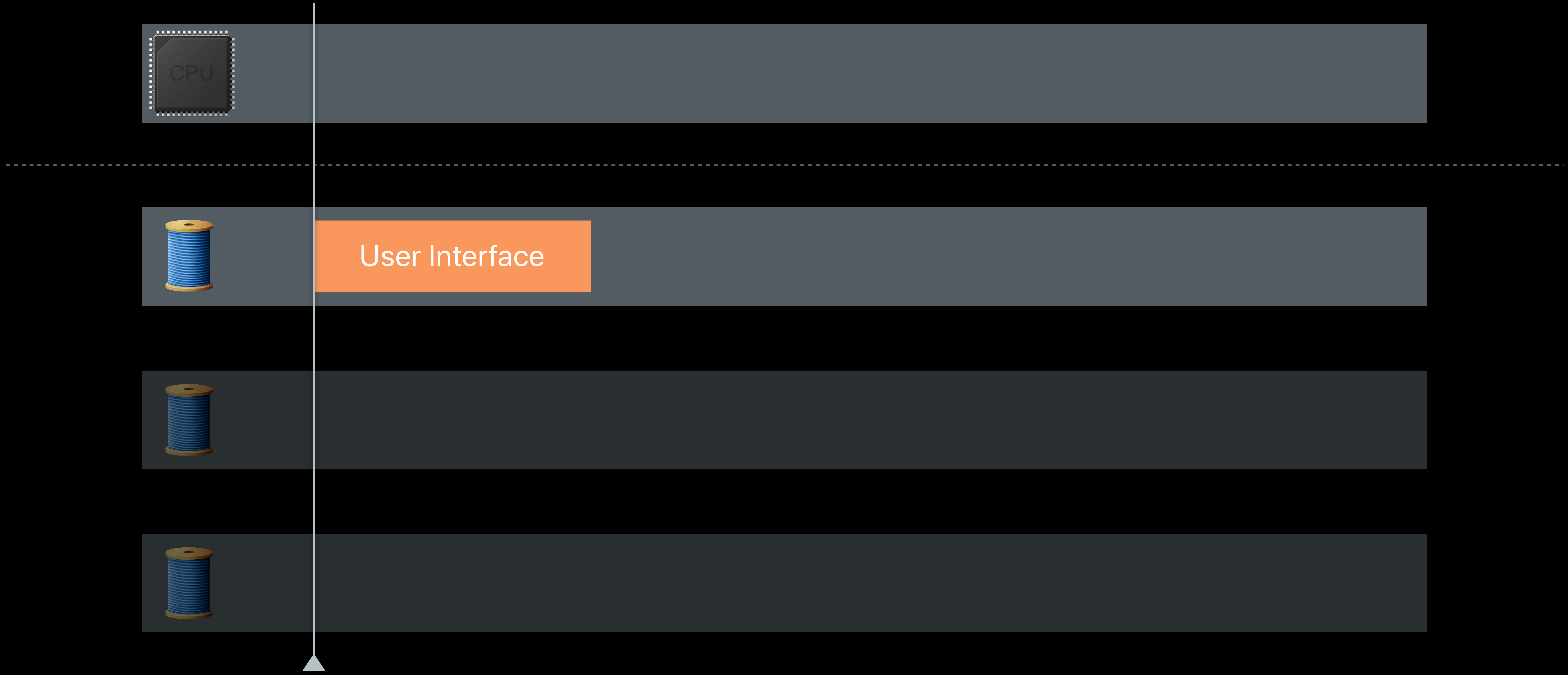
Concurrency

Composition of independently executed tasks



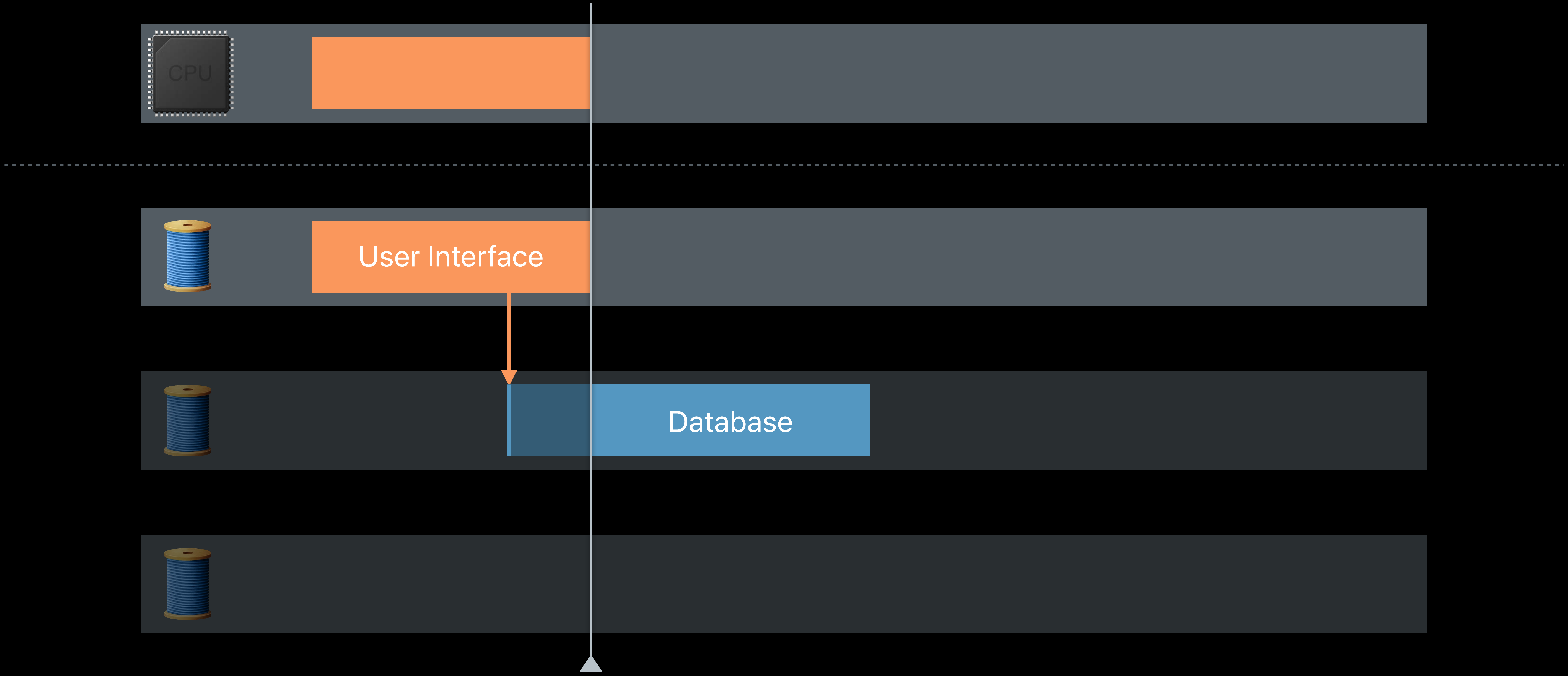
Concurrency

Composition of independently executed tasks



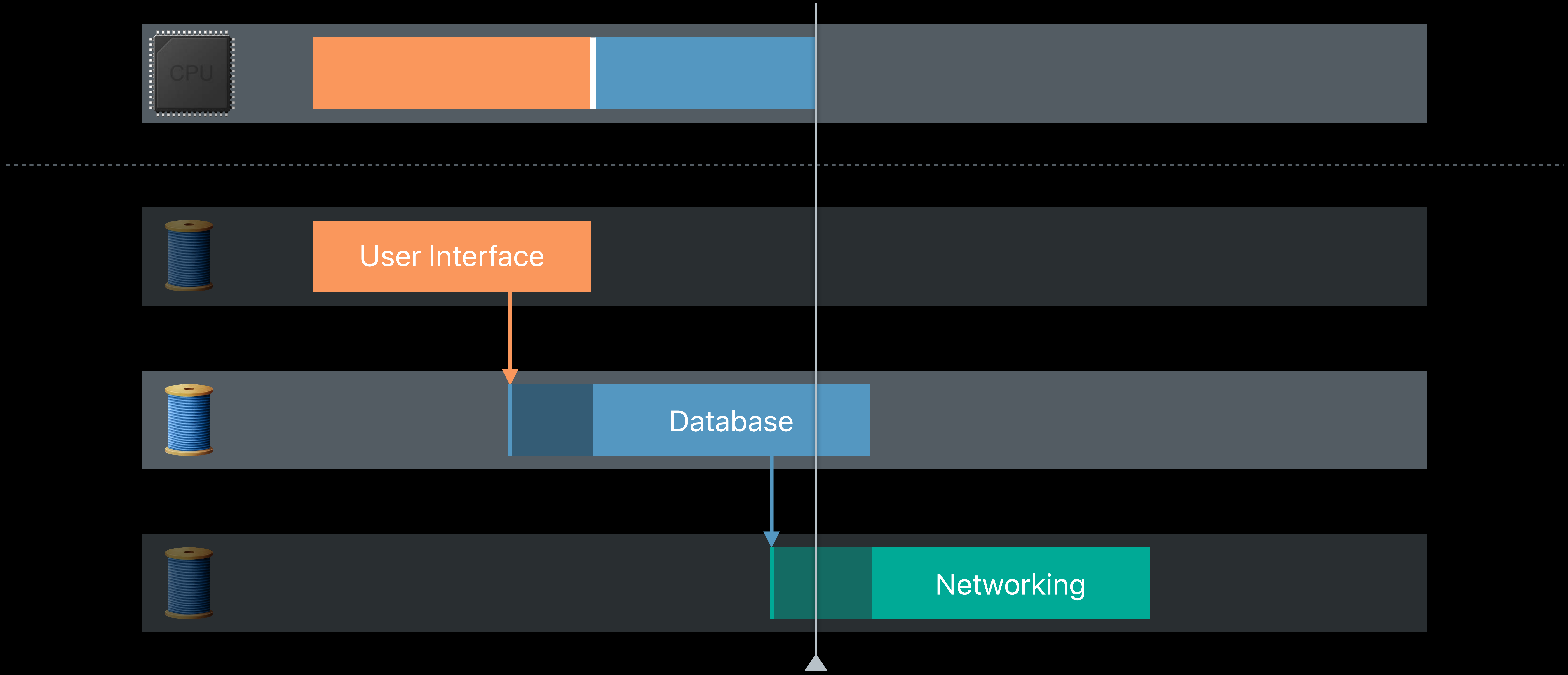
Concurrency

Composition of independently executed tasks



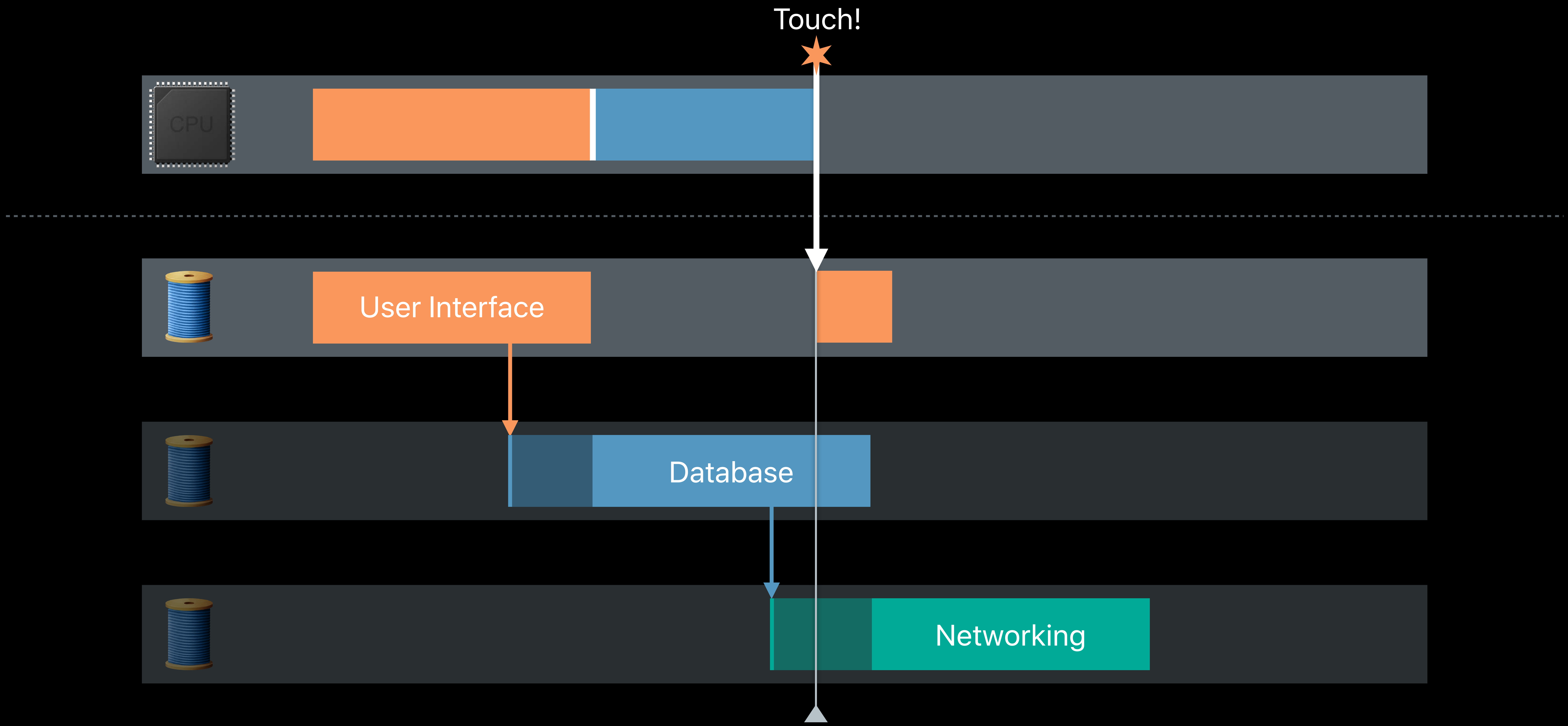
Concurrency

Composition of independently executed tasks



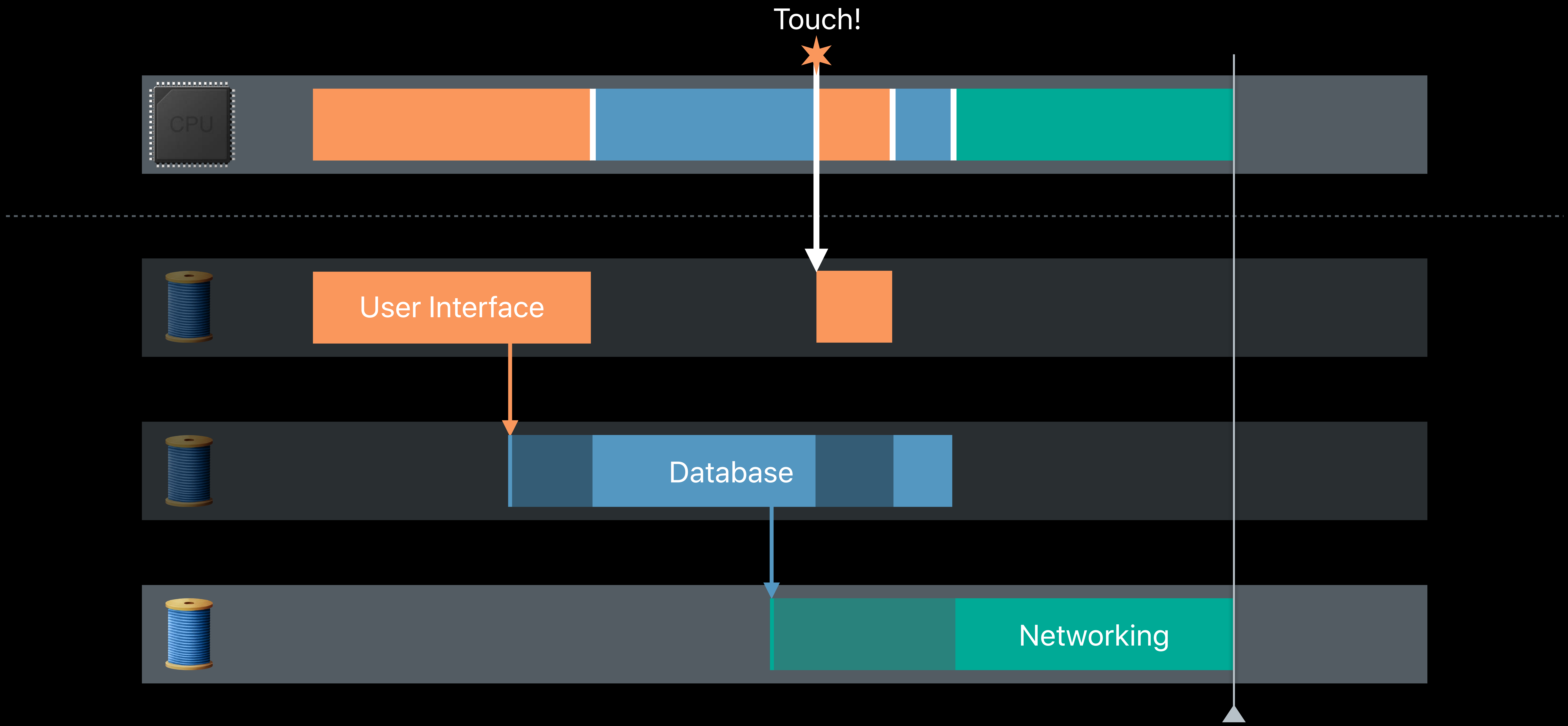
Concurrency

Composition of independently executed tasks



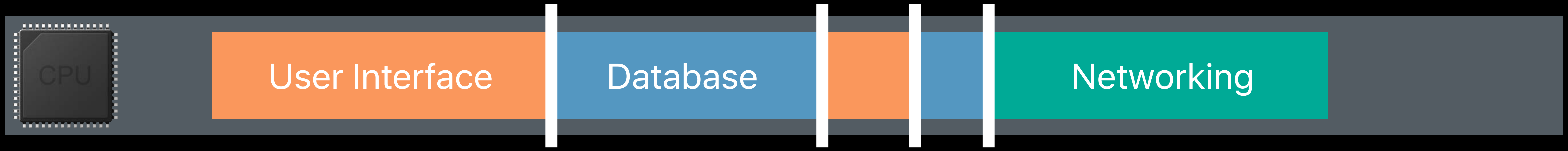
Concurrency

Composition of independently executed tasks

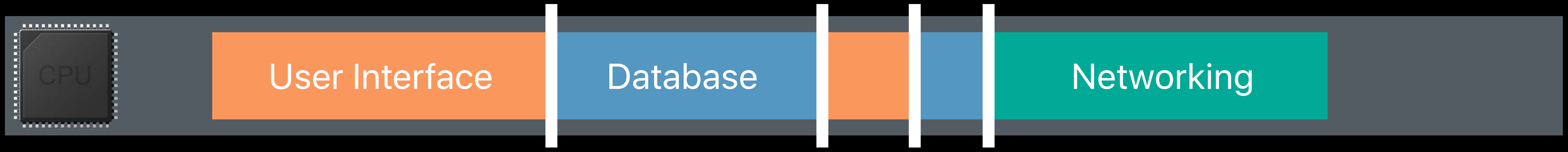


Concurrency

Context switching



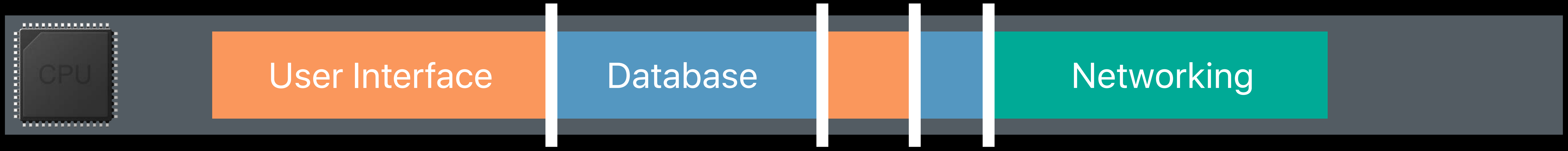
Concurrency
Context switching



Context Switching

The power of concurrency

The OS can choose a new thread at any time

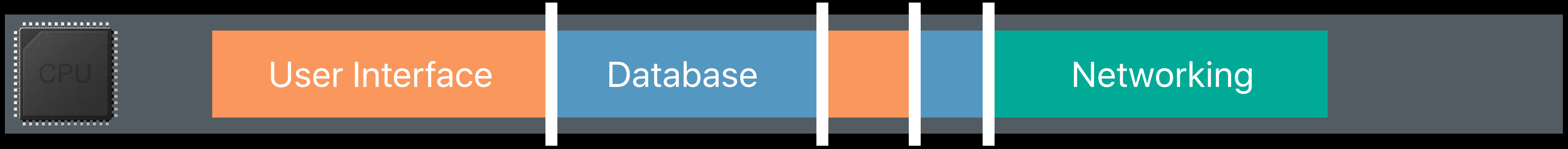


Context Switching

The power of concurrency

The OS can choose a new thread at any time

- A higher priority thread needs the CPU

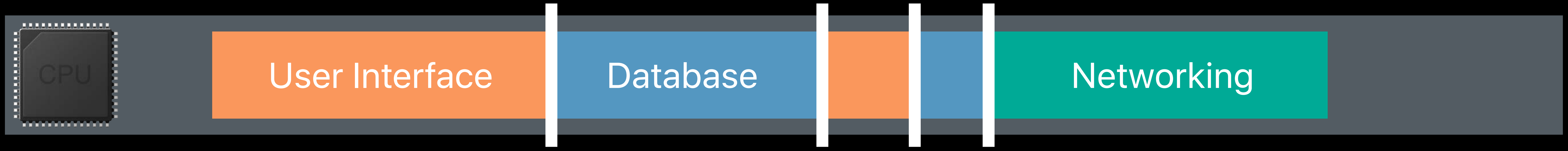


Context Switching

The power of concurrency

The OS can choose a new thread at any time

- A higher priority thread needs the CPU
- A thread finishes its current work

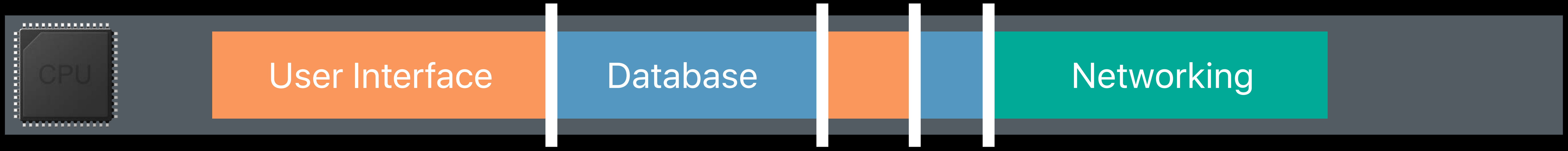


Context Switching

The power of concurrency

The OS can choose a new thread at any time

- A higher priority thread needs the CPU
- A thread finishes its current work
- Waiting to acquire a resource

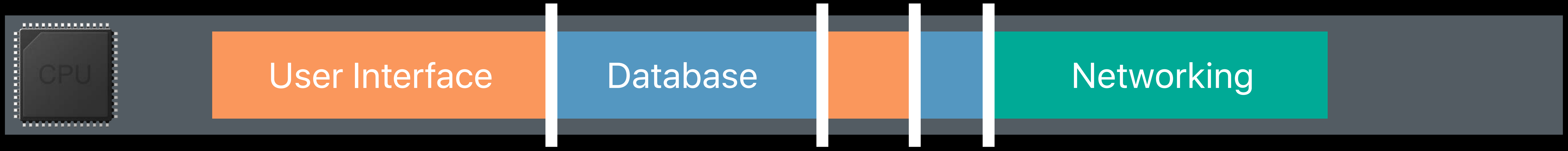


Context Switching

The power of concurrency

The OS can choose a new thread at any time

- A higher priority thread needs the CPU
- A thread finishes its current work
- Waiting to acquire a resource
- Waiting for an asynchronous request to complete



Excessive Context Switching

Too much of a good thing

Repeatedly bouncing between contexts can become expensive

Excessive Context Switching

Too much of a good thing

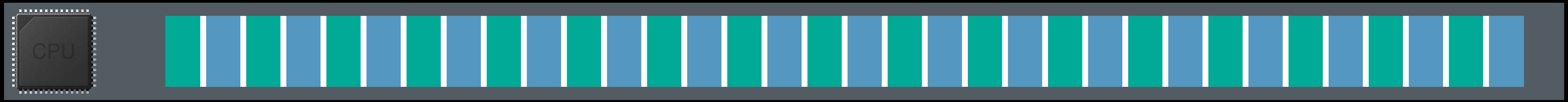
Repeatedly bouncing between contexts can become expensive

Excessive Context Switching

Too much of a good thing

Repeatedly bouncing between contexts can become expensive

- CPU runs less efficiently

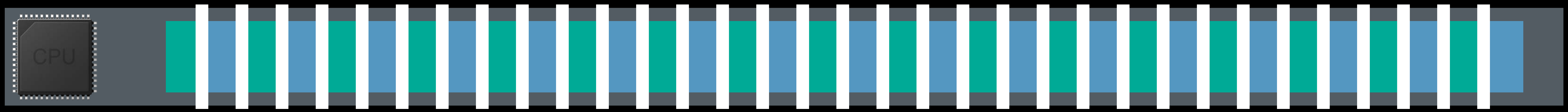


Excessive Context Switching

Too much of a good thing

Repeatedly bouncing between contexts can become expensive

- CPU runs less efficiently



Excessive Context Switching

Too much of a good thing

Repeatedly bouncing between contexts can become expensive

- CPU runs less efficiently
- There may be others ahead in line for CPU access

Excessive Context Switching

Too much of a good thing

Repeatedly bouncing between contexts can become expensive

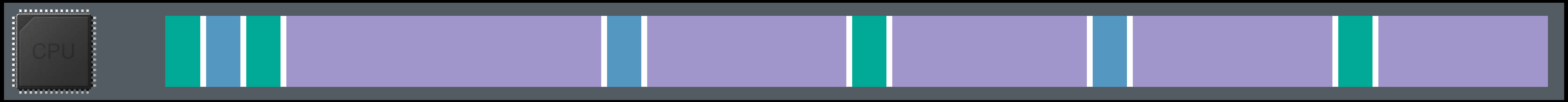
- CPU runs less efficiently
- There may be others ahead in line for CPU access

Excessive Context Switching

Too much of a good thing

Repeatedly bouncing between contexts can become expensive

- CPU runs less efficiently
- There may be others ahead in line for CPU access



Excessive Context Switching

Too much of a good thing

Repeatedly waiting for exclusive access to contended resources

Repeatedly switching between independent operations

Repeatedly bouncing an operation between threads

Excessive Context Switching

Too much of a good thing

Repeatedly waiting for exclusive access to contended resources

Repeatedly switching between independent operations

Repeatedly bouncing an operation between threads

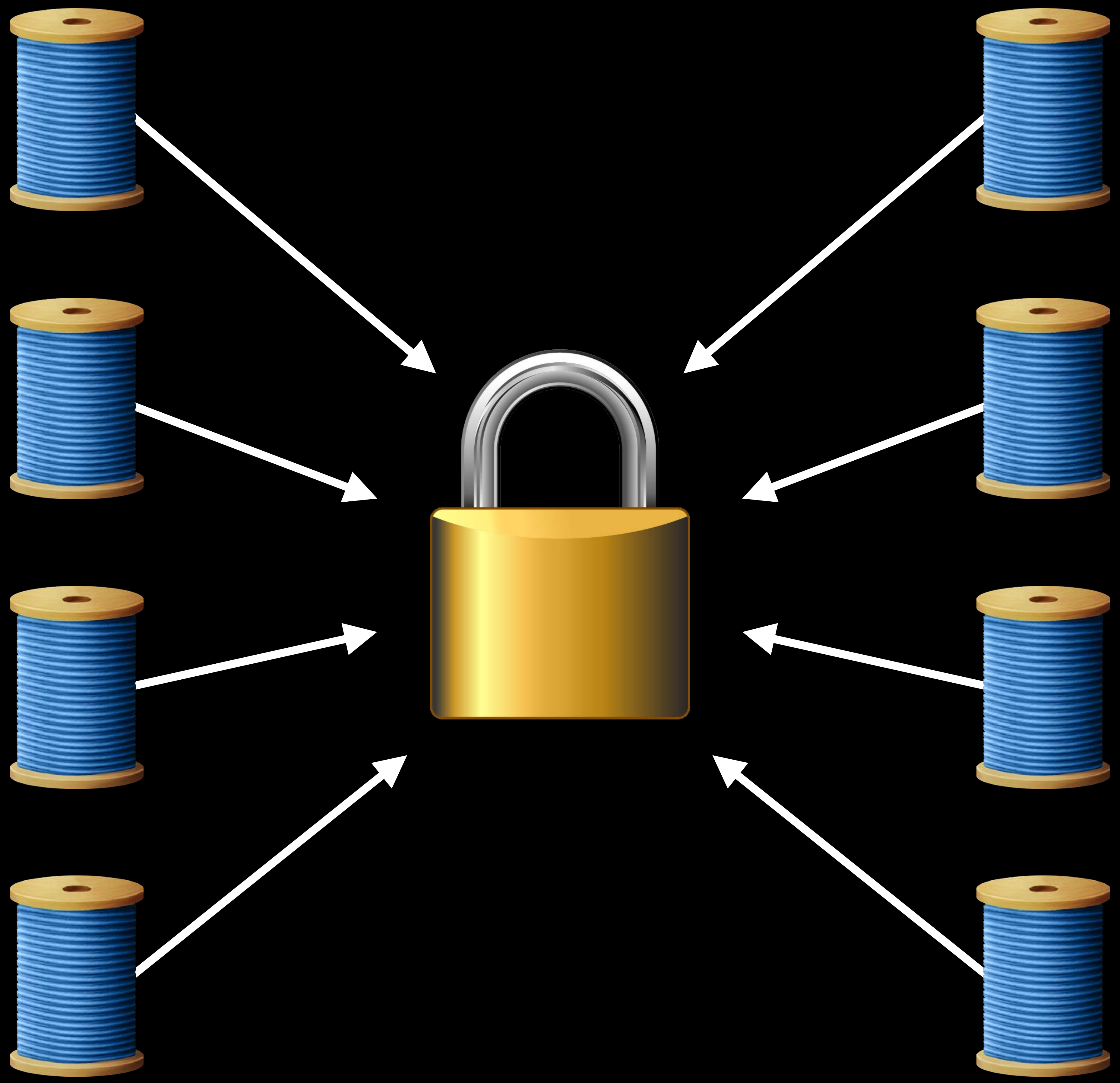
Too much of a good thing

Repeatedly waiting for exclusive access to contended resources

Repeatedly switching between independent operations

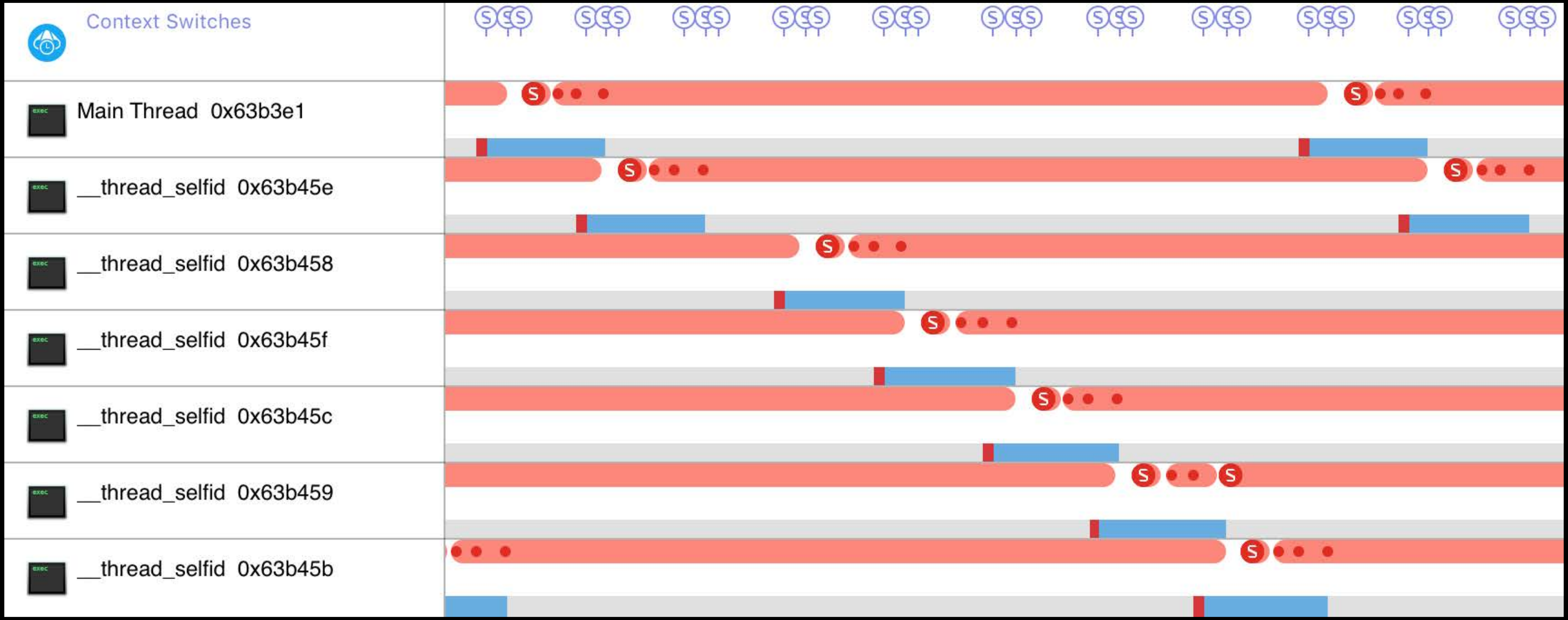
Repeatedly bouncing an operation between threads

Lock Contention



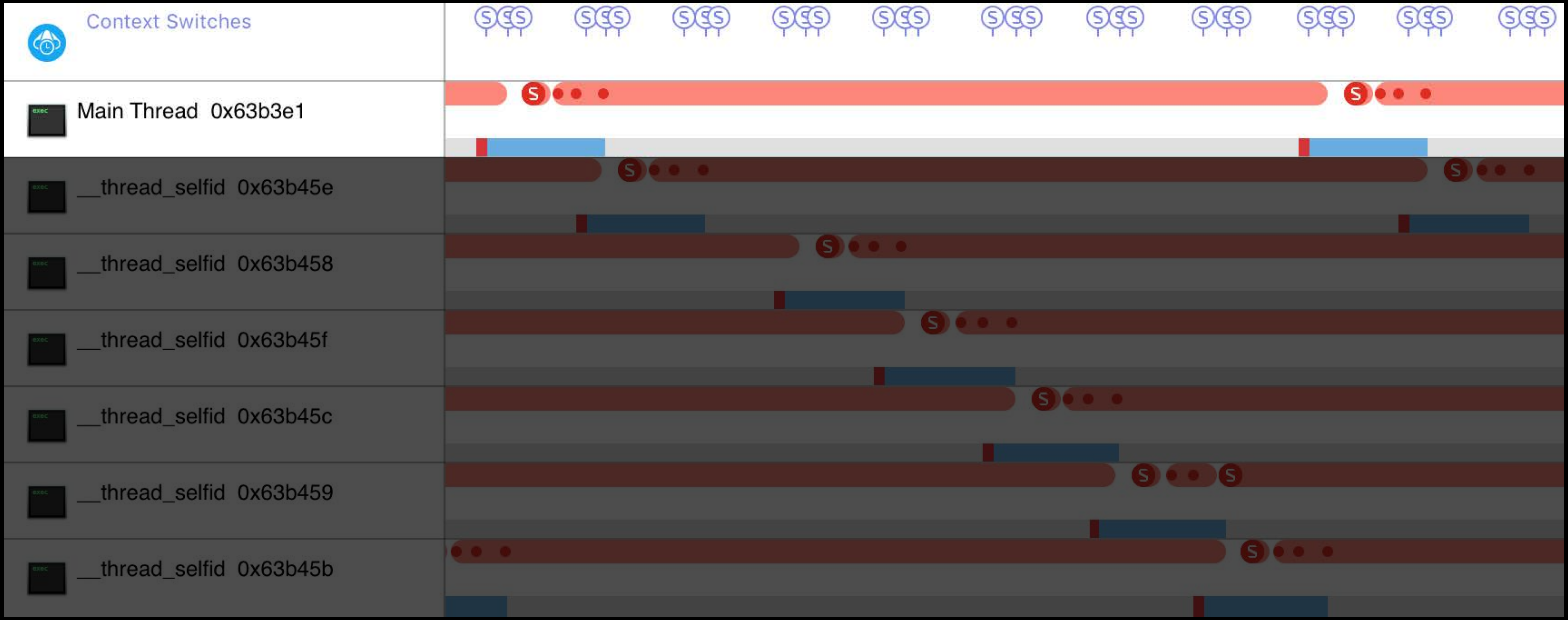
Lock Contention

Visualization in Instruments



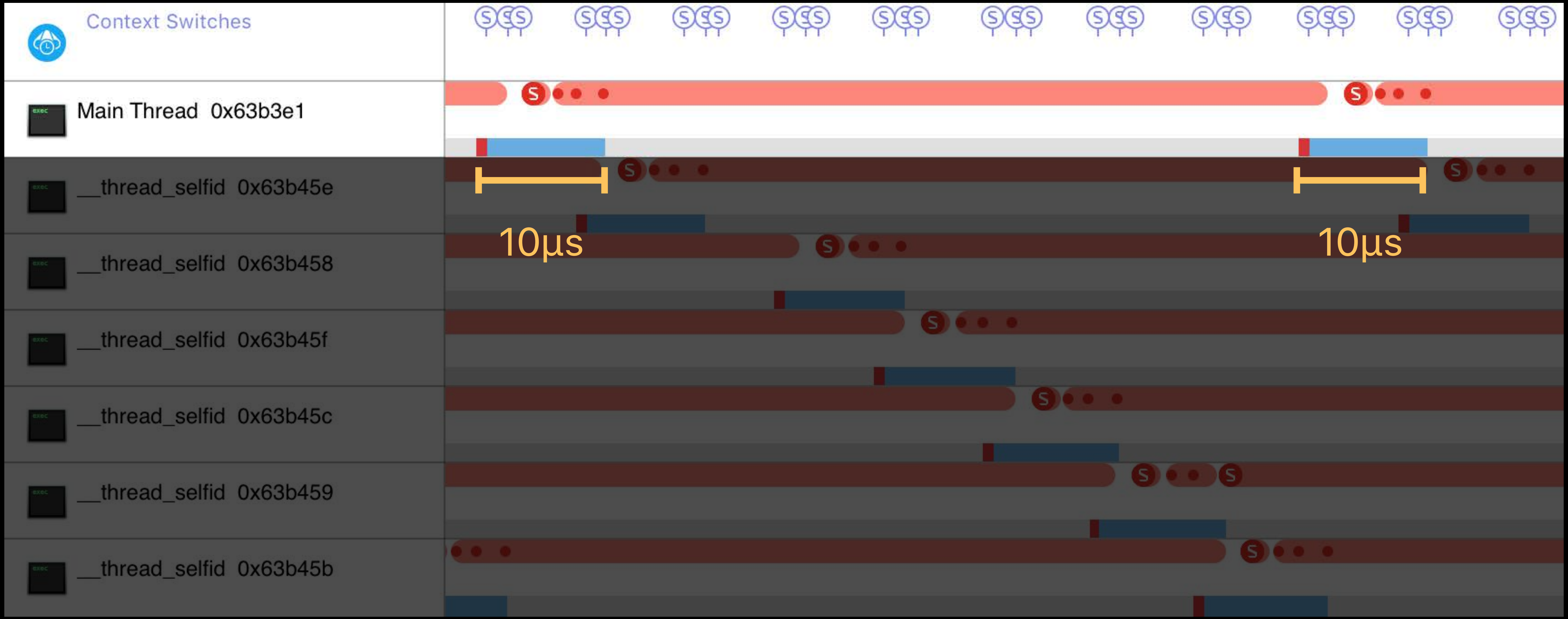
Lock Contention

Visualization in Instruments



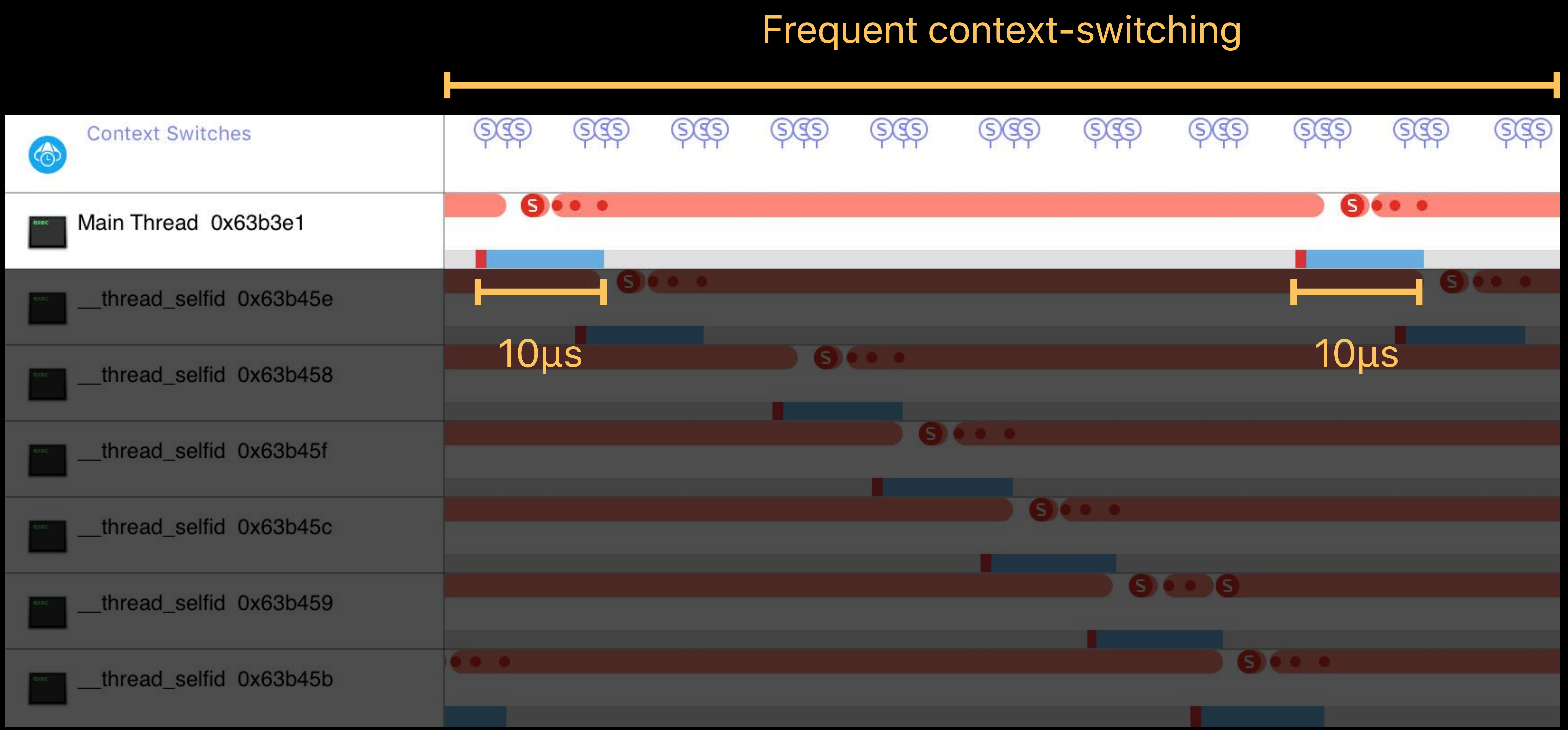
Lock Contention

Visualization in Instruments

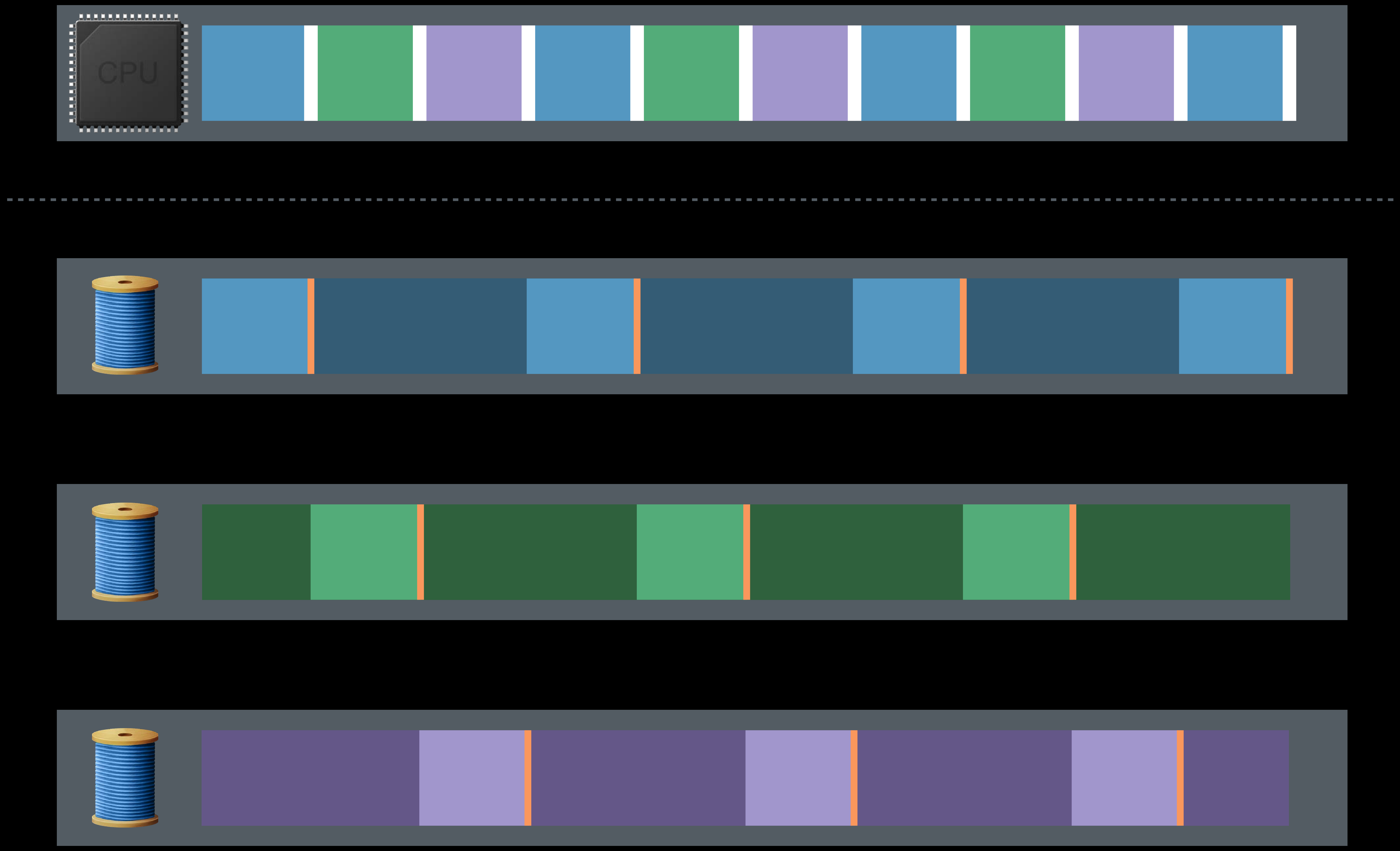


Lock Contention

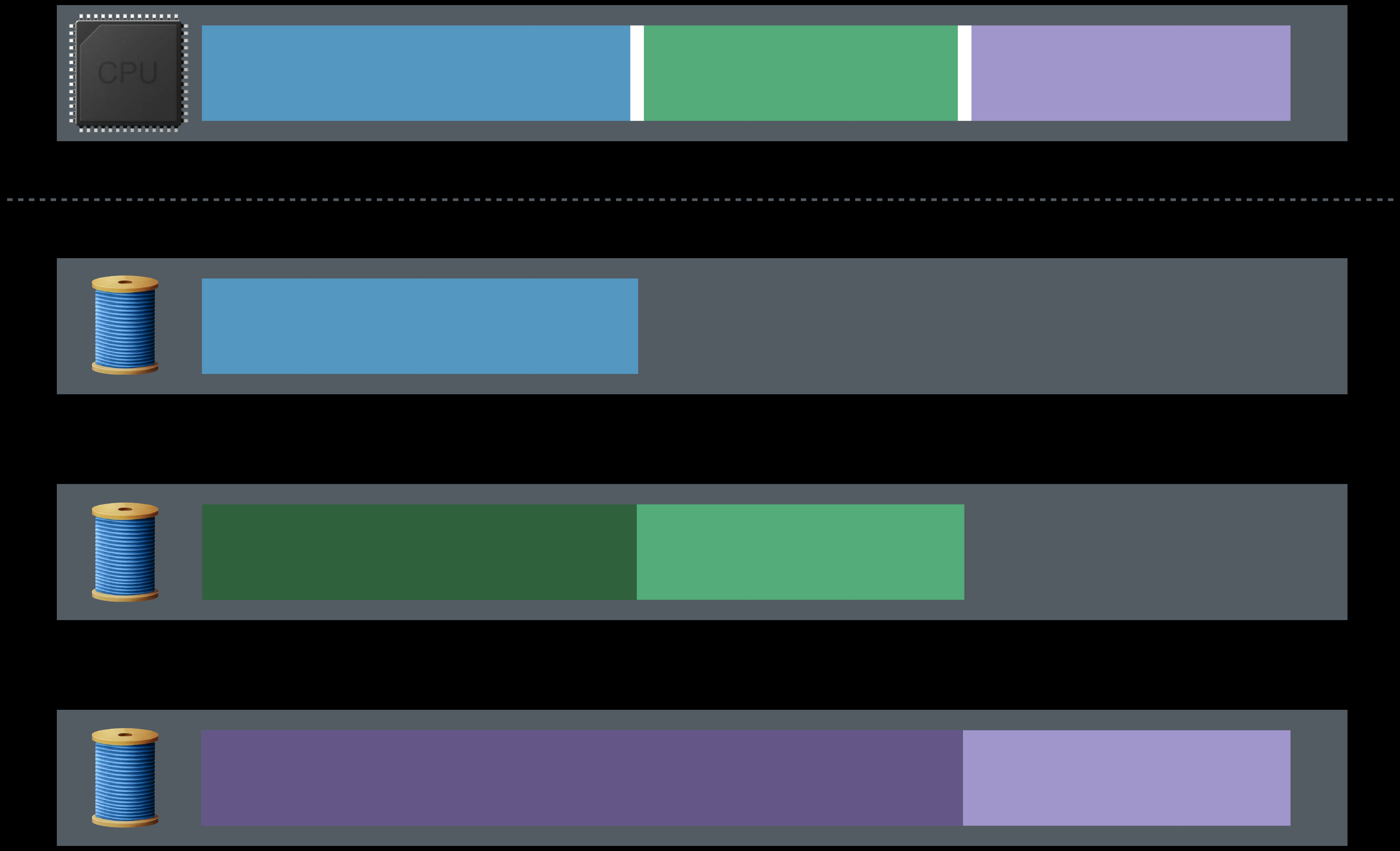
Visualization in Instruments



Lock Contention

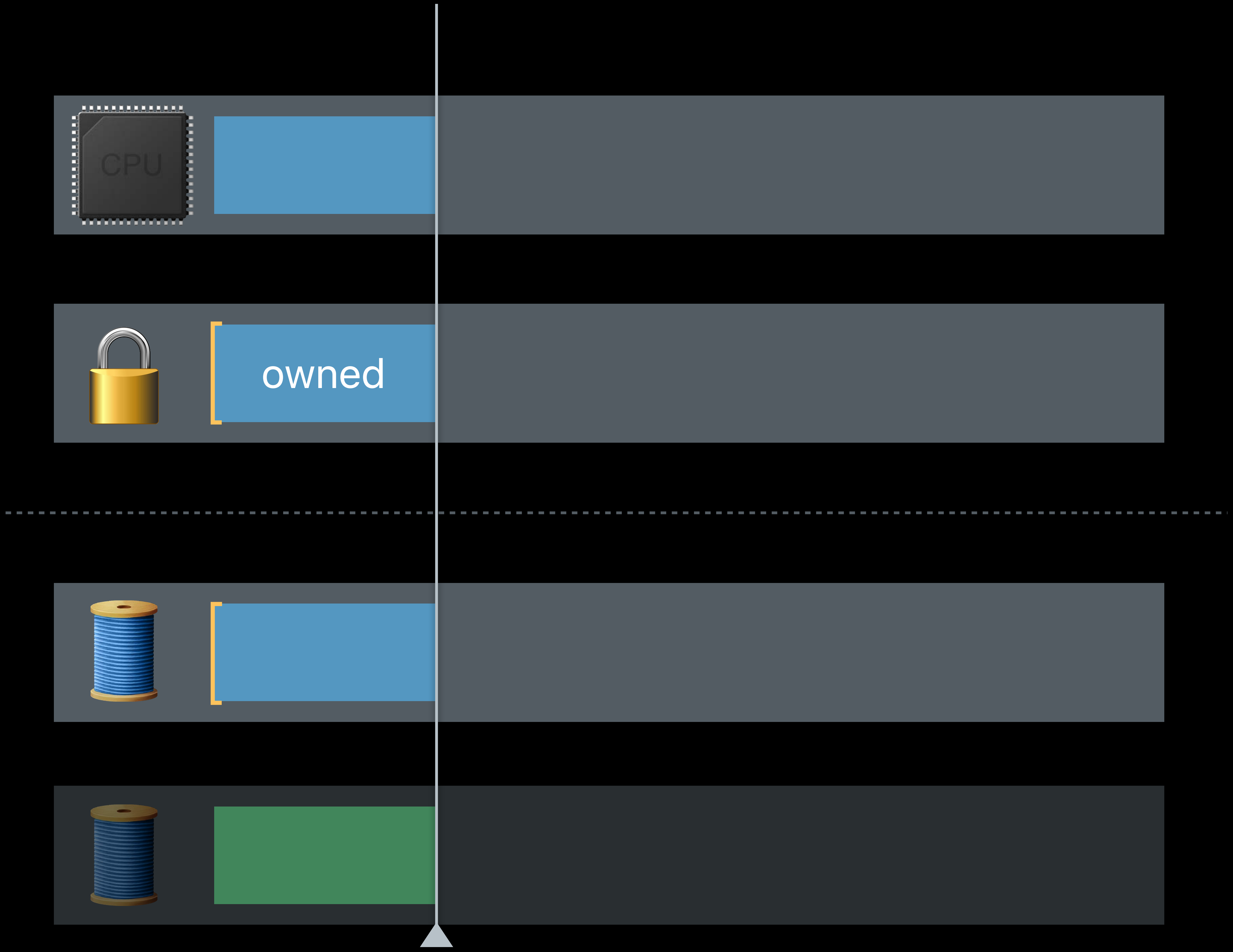


Lock Contention



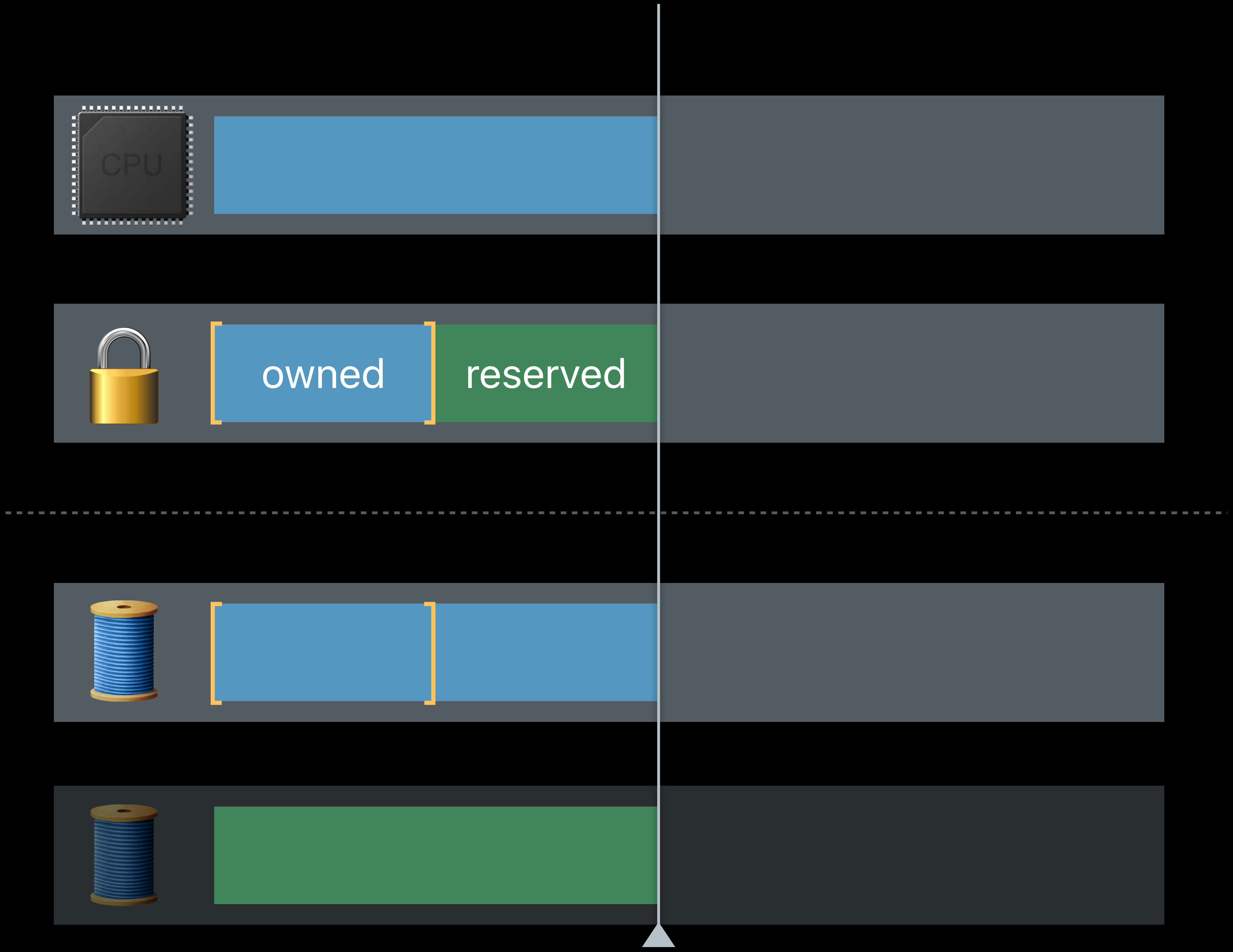
Lock Contention

Fair locks



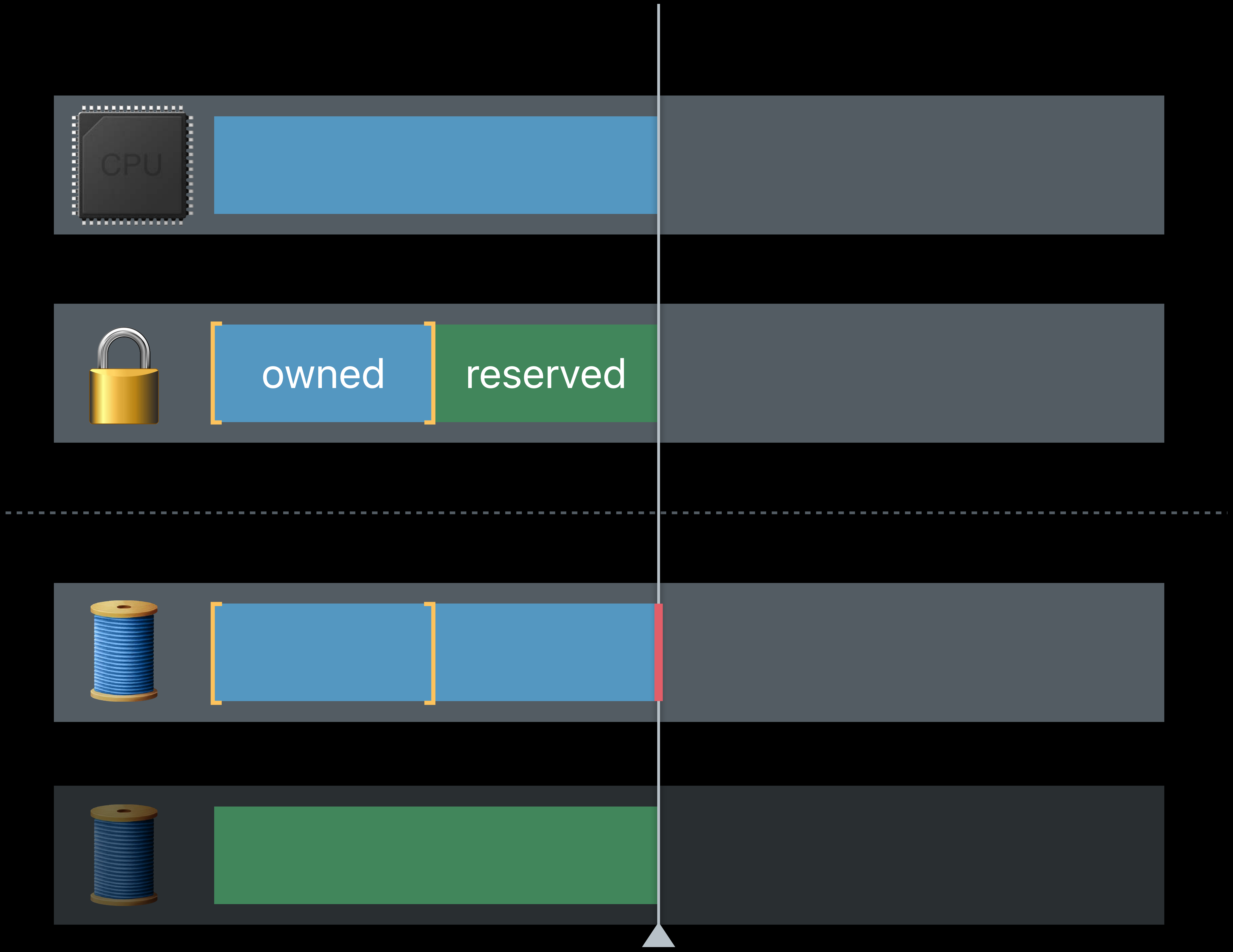
Lock Contention

Fair locks



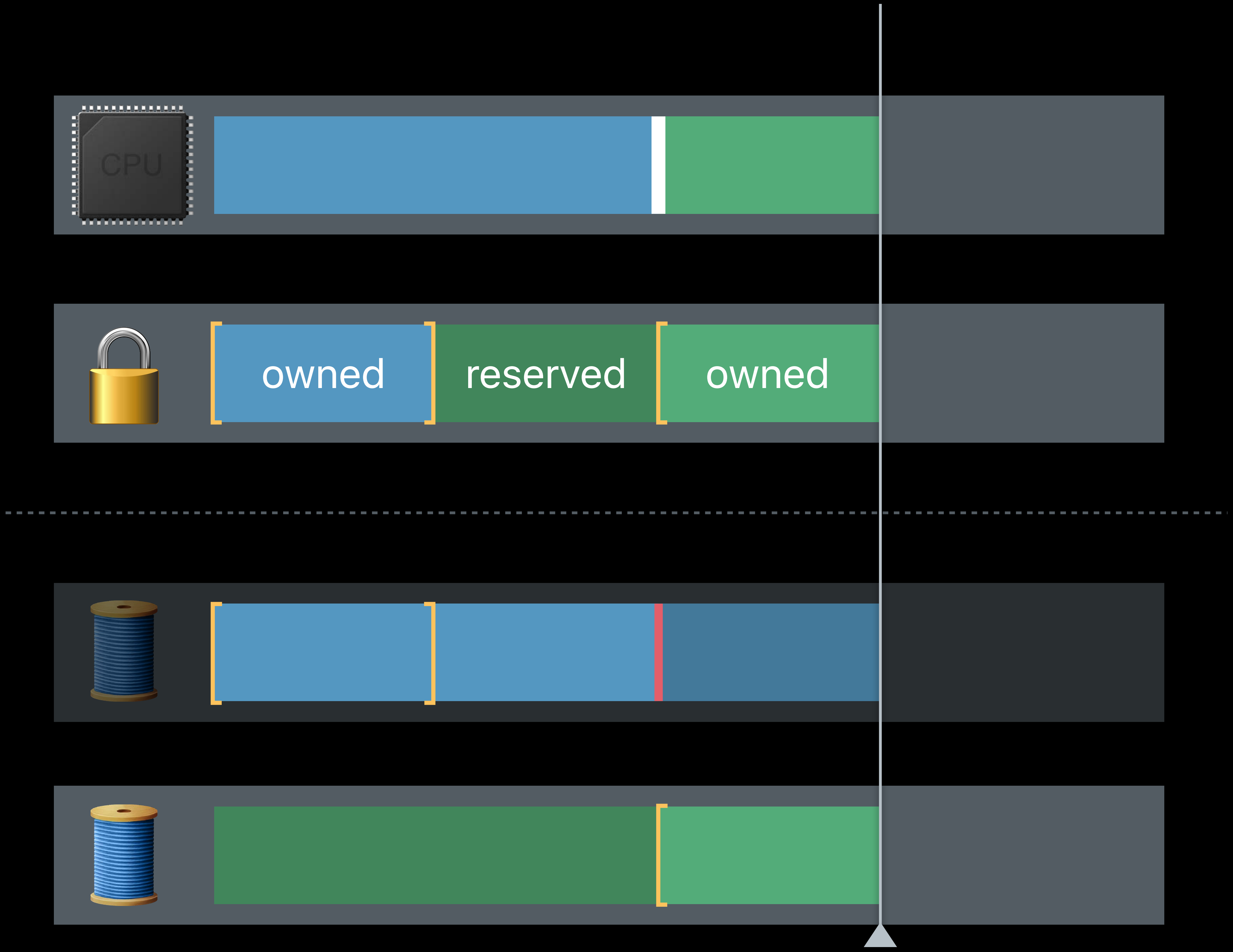
Lock Contention

Fair locks



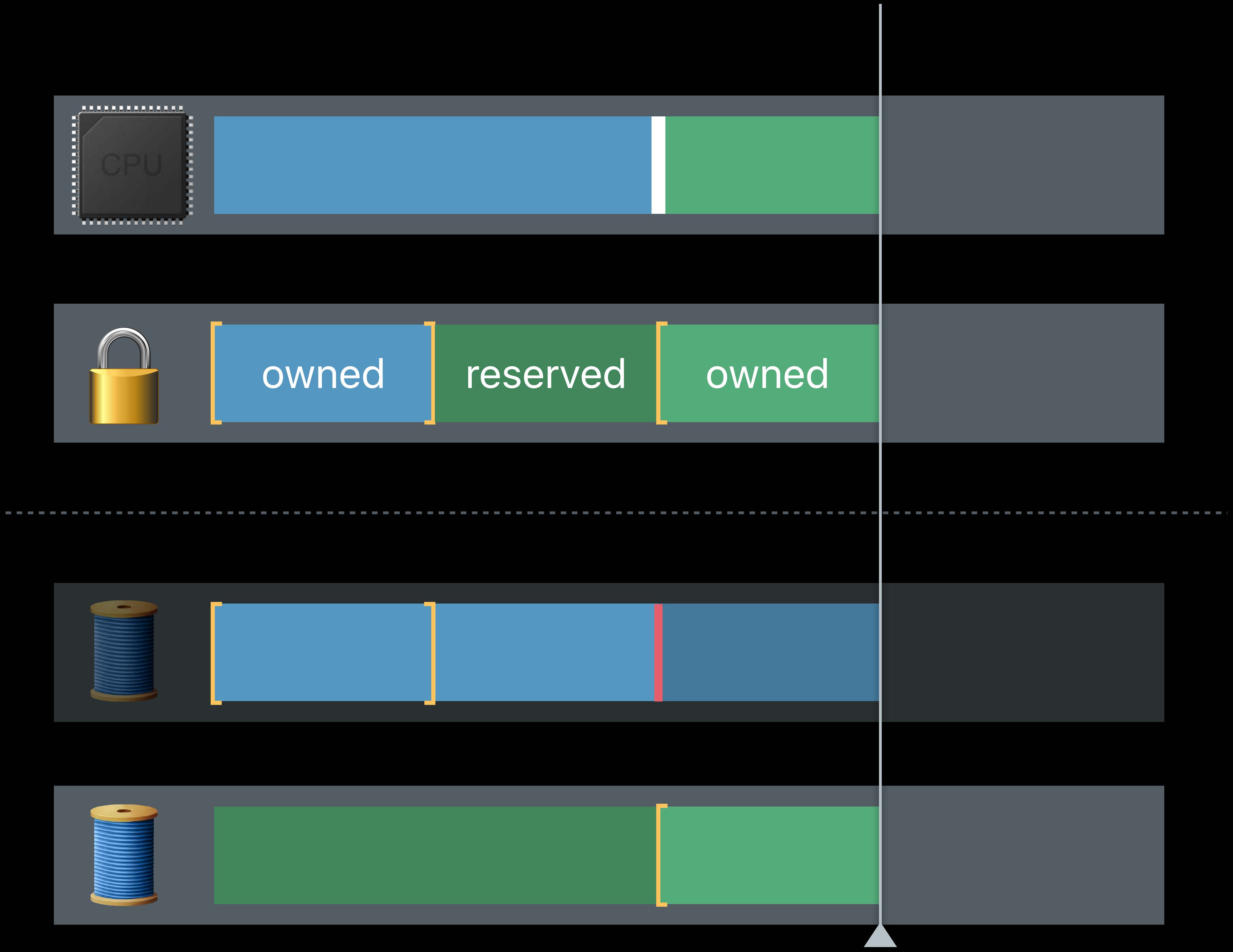
Lock Contention

Fair locks

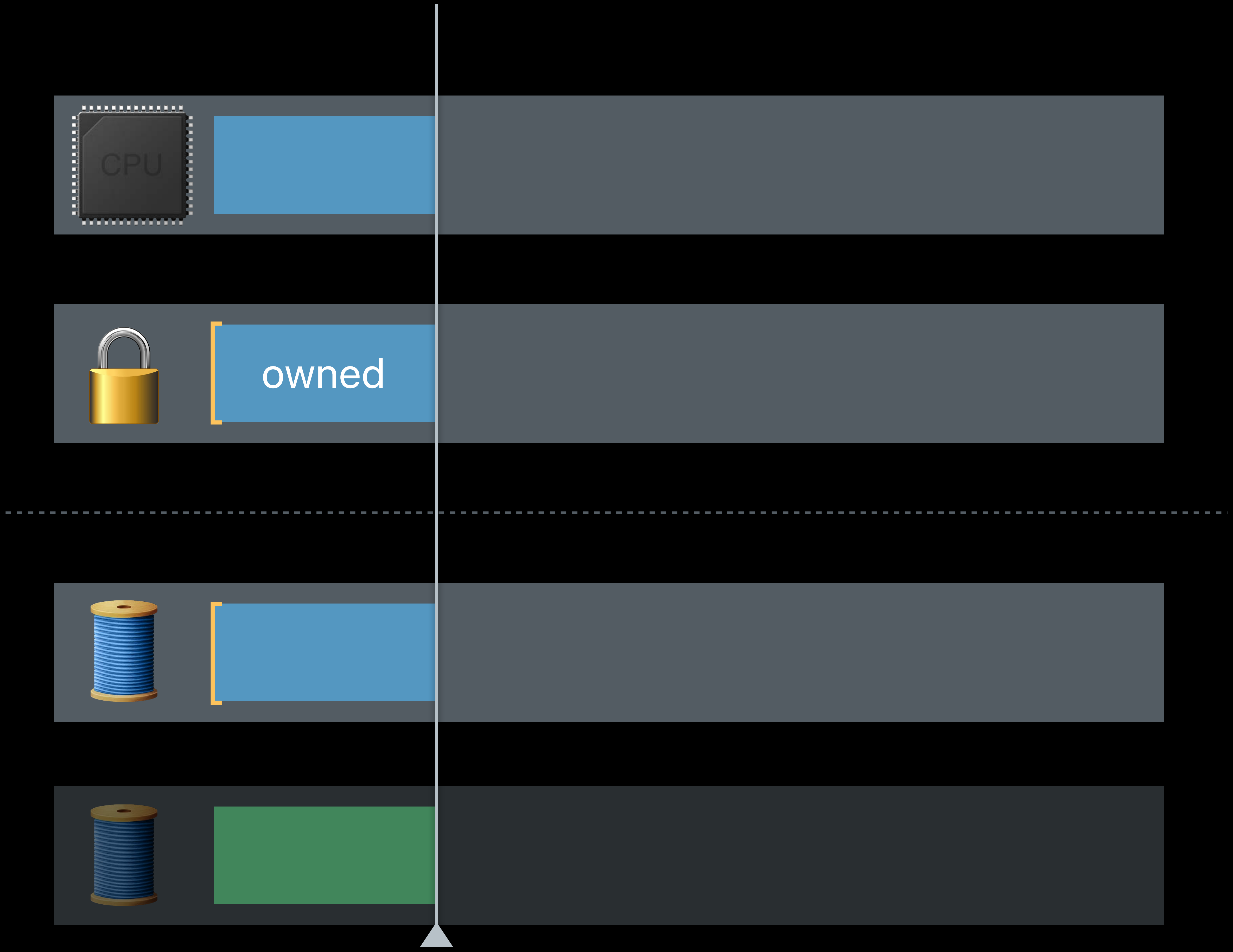


Lock Contention

Fair locks

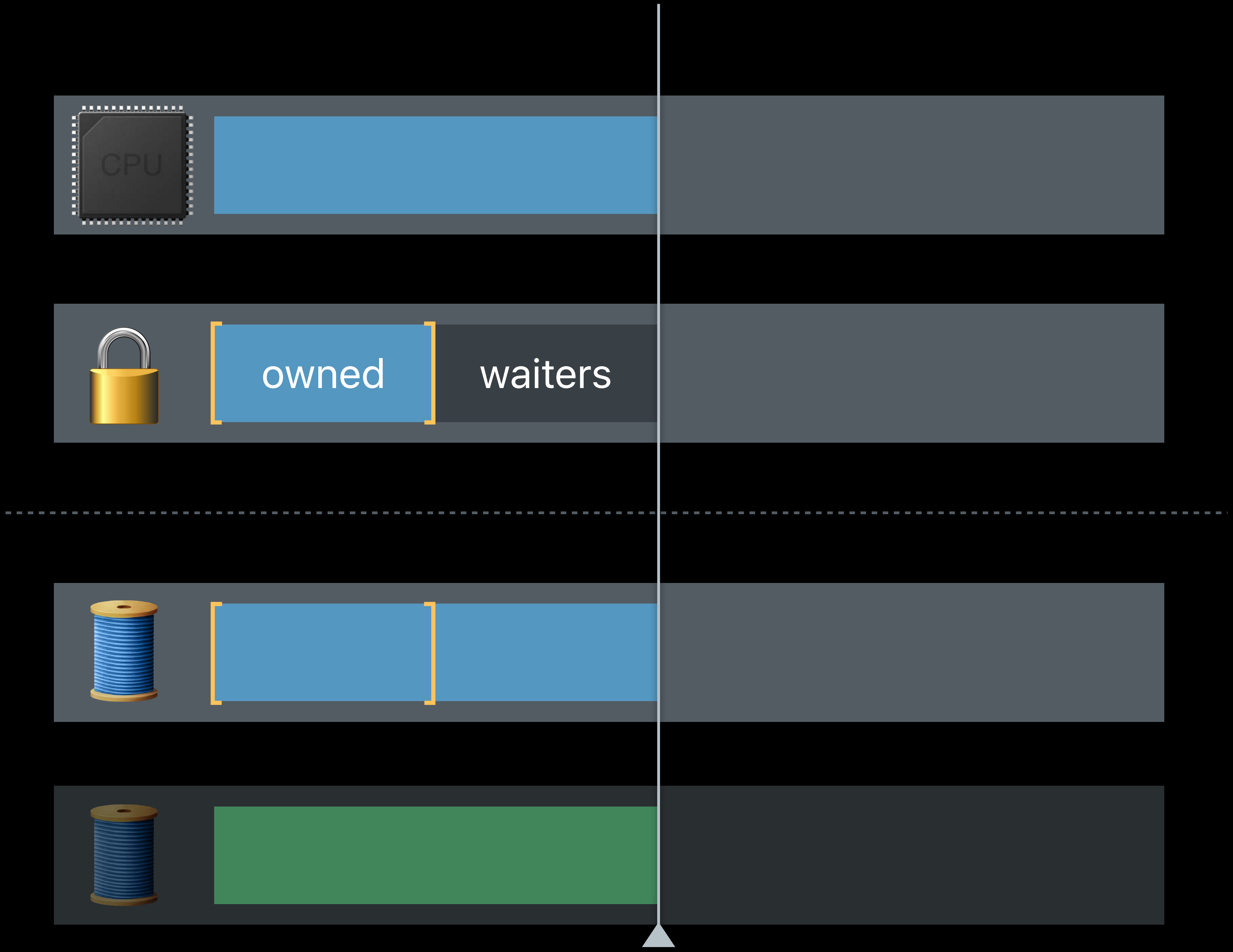


Lock Contention
Unfair locks



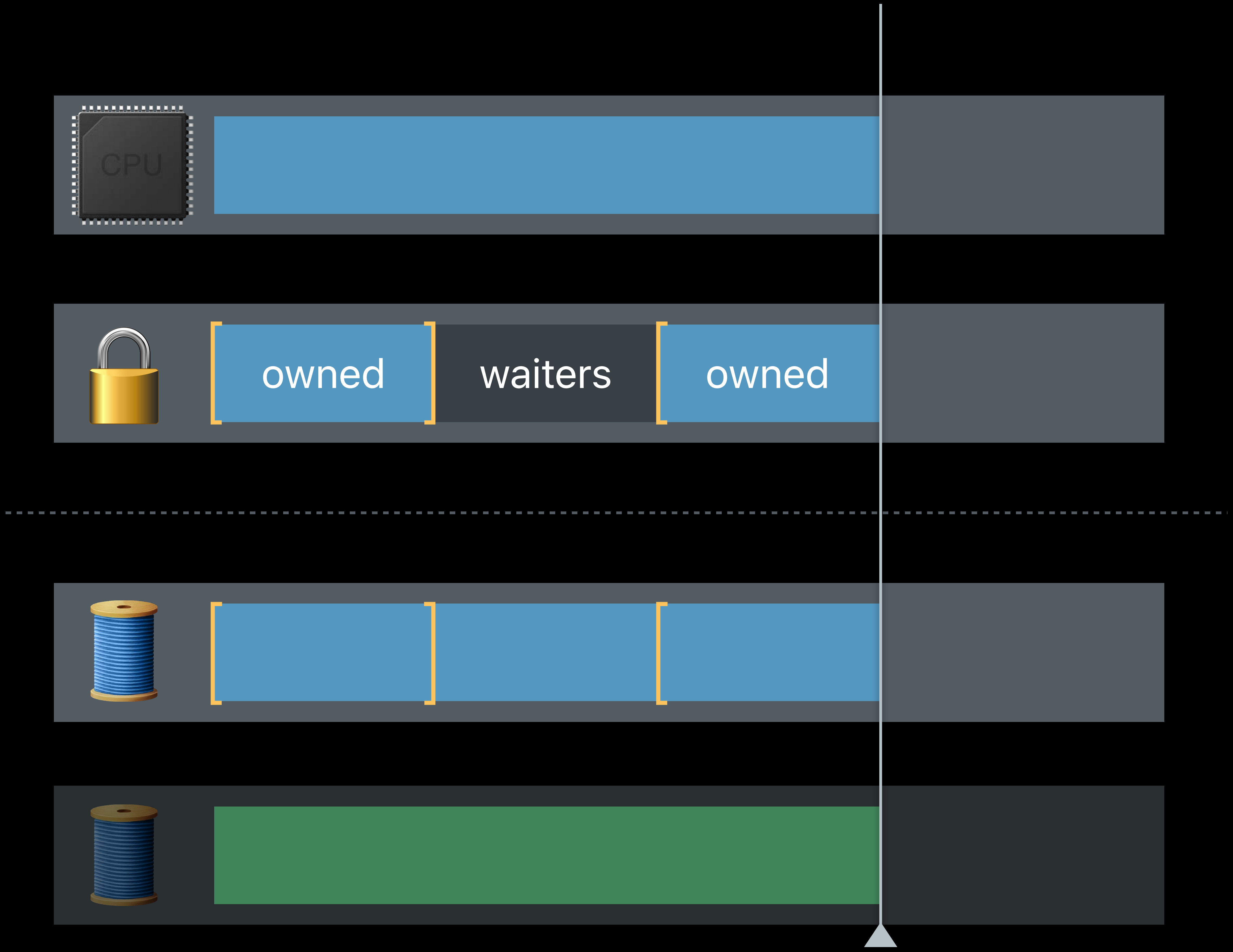
Lock Contention

Unfair locks



Lock Contention

Unfair locks



Lock Contention

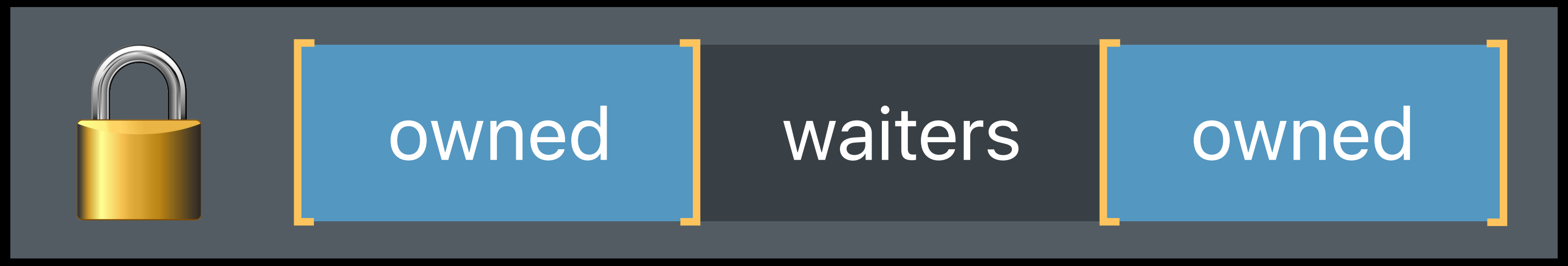
Use the right lock for the job

	Unfair	Fair

Available types	`os_unfair_lock`	`pthread_mutex_t`, `NSLock`
		`DispatchQueue.sync`

Contended lock re-acquisition	Can steal the lock	Context switches to next waiter
Subject to waiter starvation	Yes	No

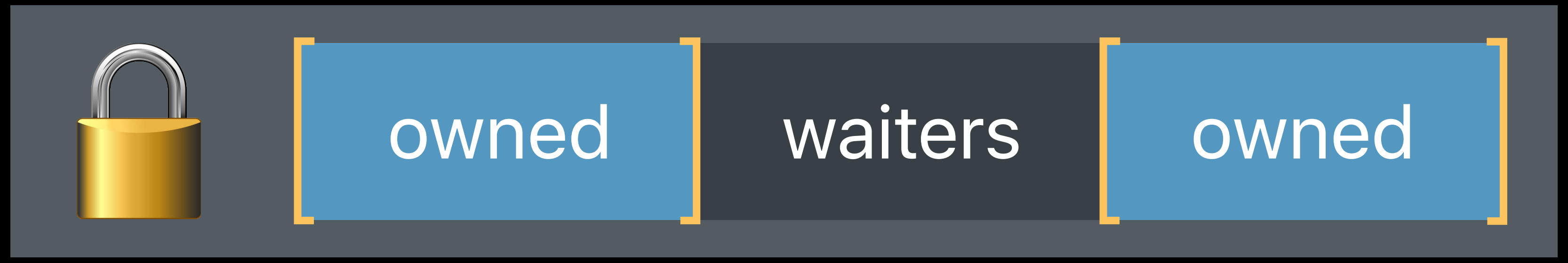
Lock Ownership



Lock Ownership

Ownership helps resolve priority inversion

- High priority waiter
- Low priority owner



Lock Ownership

Single Owner
Serial queues
`DispatchWorkItem.wait`
`os_unfair_lock`
`pthread_mutex`, `NSLock`

Lock Ownership

Single Owner	No Owner
Serial queues	`dispatch_semaphore`
`DispatchWorkItem.wait`	`dispatch_group`
`os_unfair_lock`	`pthread_cond`, `NSCondition`
`pthread_mutex`, `NSLock`	Queue suspension

Lock Ownership

Single Owner	No Owner	Multiple Owners
Serial queues	`dispatch_semaphore`	Private concurrent queues
`DispatchWorkItem.wait`	`dispatch_group`	`pthread_rwlock`
`os_unfair_lock`	`pthread_cond`, `NSCondition`	
`pthread_mutex`, `NSLock`	Queue suspension	

Optimizing Lock Contention

Inefficient behaviors are often emergent properties

Visualize your app’s behavior with Instruments

Use the right lock for the job

Too Much of a Good Thing

Repeatedly waiting for exclusive access to contended resources

Repeatedly switching between independent operations

Repeatedly bouncing an operation between threads

Using GCD for Concurrency

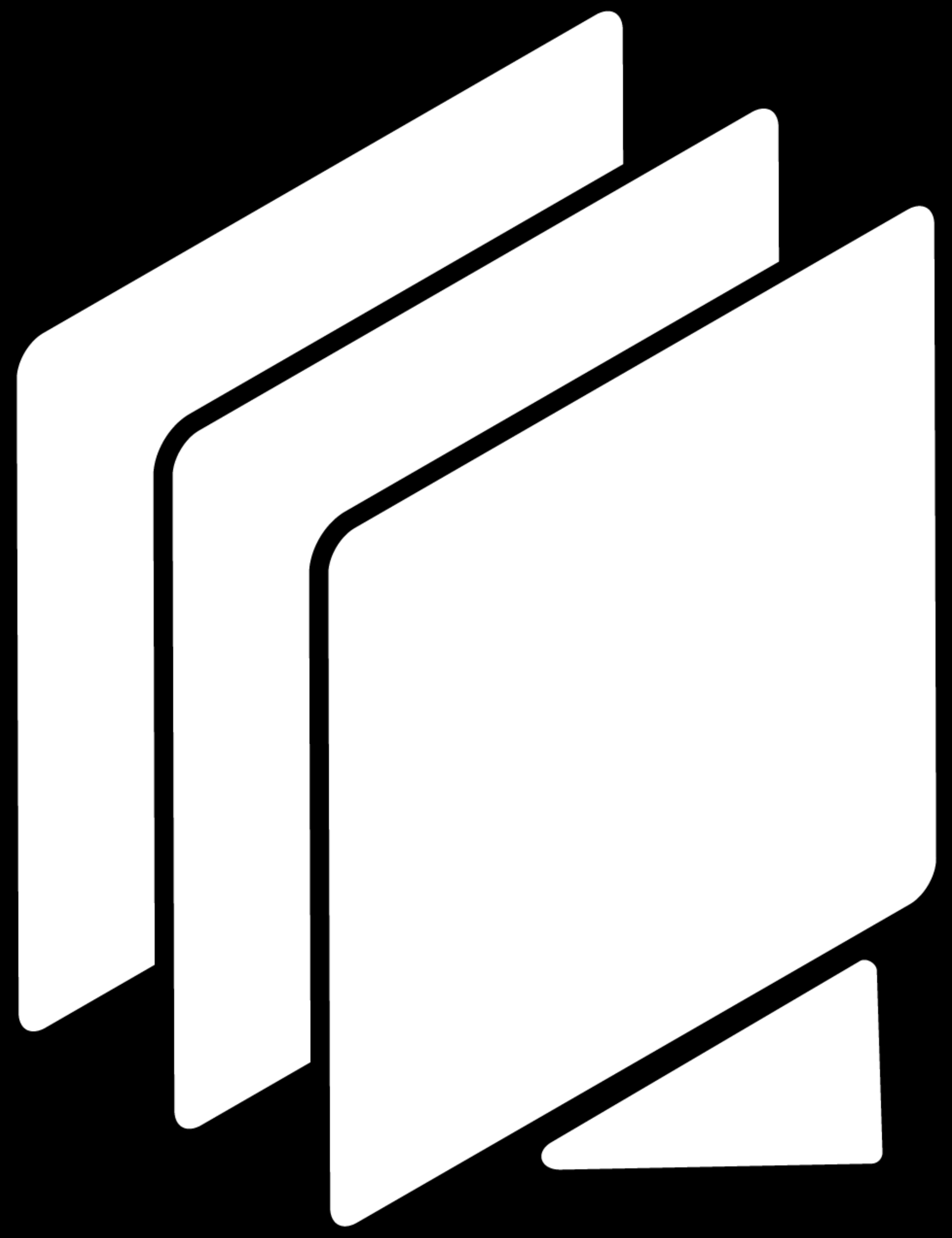
Daniel A. Steffen, Core Darwin

Grand Central Dispatch

Simplifying iPhone App Development with Grand Central Dispatch	WWDC 2010
Asynchronous Design Patterns with Blocks, GCD, and XPC	WWDC 2012
Power, Performance, and Diagnostics: What's new in GCD and XPC	WWDC 2014
Building Responsive and Efficient Apps with GCD	WWDC 2015
Concurrent Programming with GCD in Swift 3	WWDC 2016

Serial Dispatch Queue

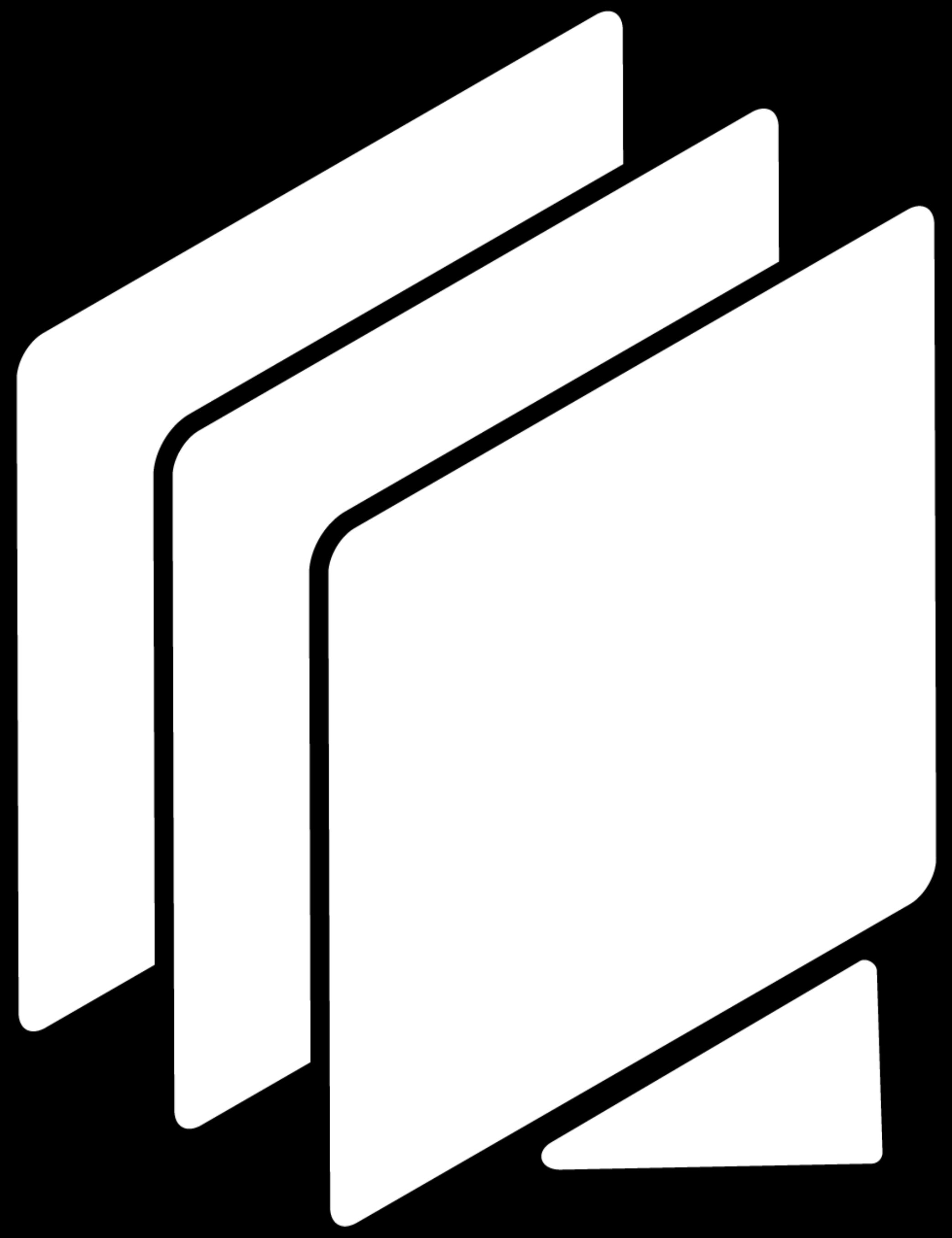
Fundamental GCD primitive



Serial Dispatch Queue

Fundamental GCD primitive

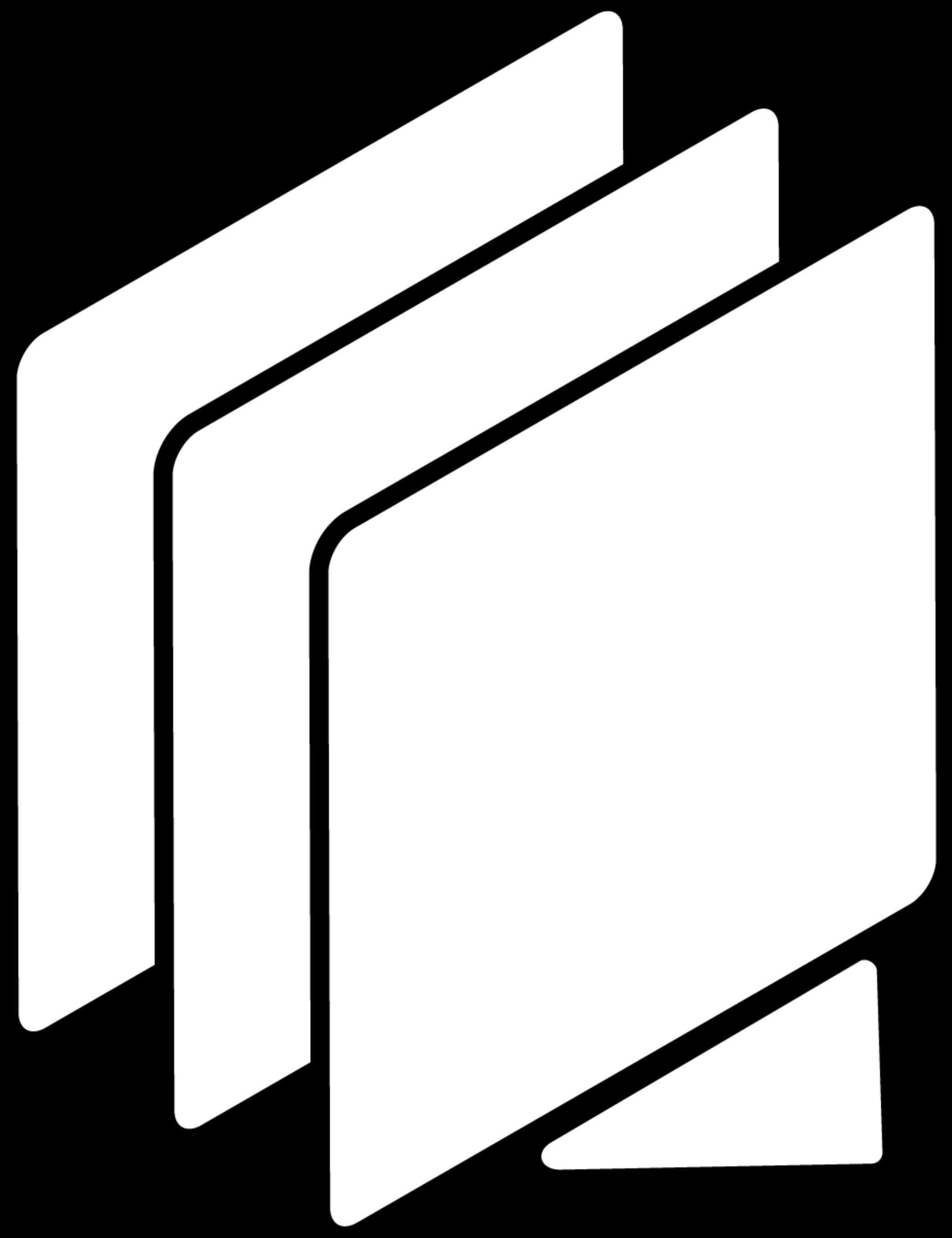
- Mutual exclusion
- FIFO ordered



Serial Dispatch Queue

Fundamental GCD primitive

- Mutual exclusion
- FIFO ordered
- Concurrent atomic enqueue
- Single dequeuer



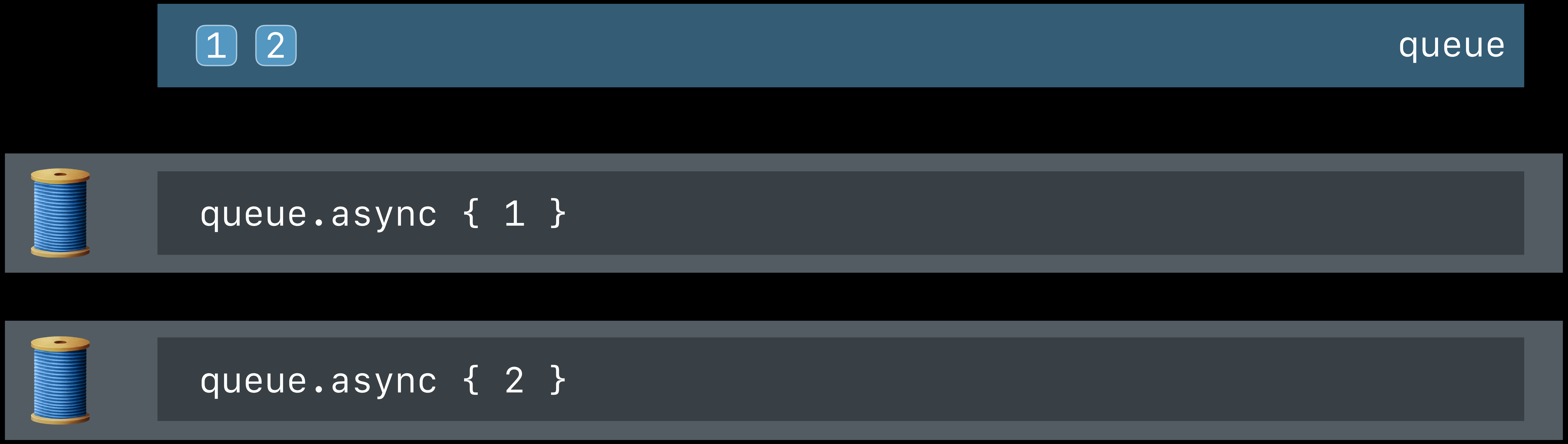
Serial Dispatch Queue

```
let queue = DispatchQueue(label: "com.example.queue")
queue.async { /*  1  */ }
queue.async { /*  2  */ }
queue.sync  { /*  3  */ }
```

Serial Dispatch Queue

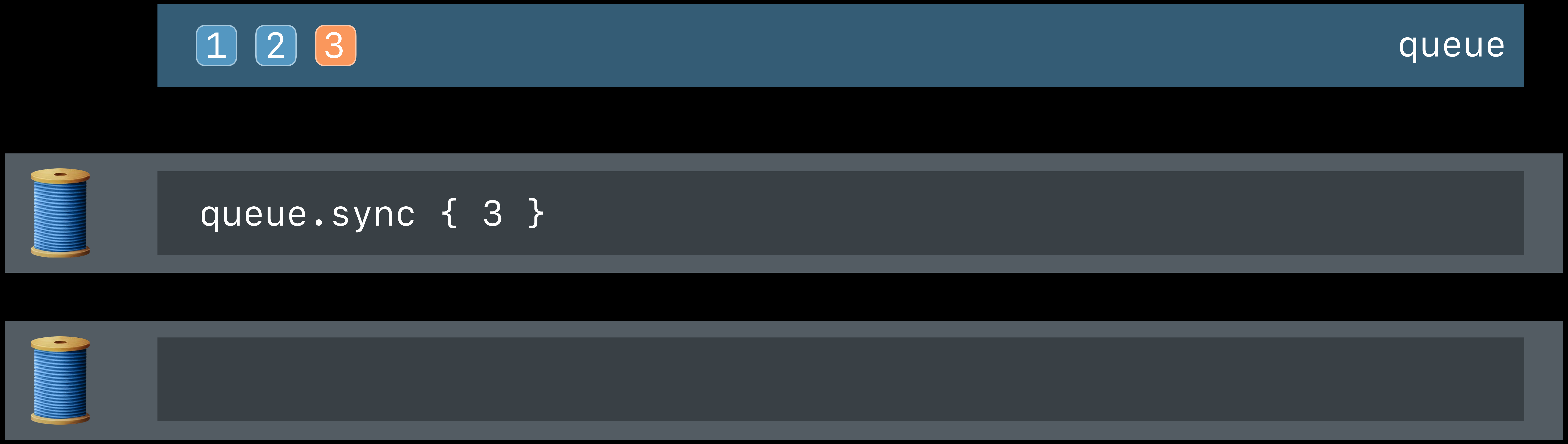
```
let queue = DispatchQueue(label: "com.example.queue")
queue.async { /*  1  */ }
queue.async { /*  2  */ }
queue.sync  { /*  3  */ }
```

Serial Dispatch Queue



```
let queue = DispatchQueue(label: "com.example.queue")
queue.async { /*  1  */ }
queue.async { /*  2  */ }
queue.sync  { /*  3  */ }
```

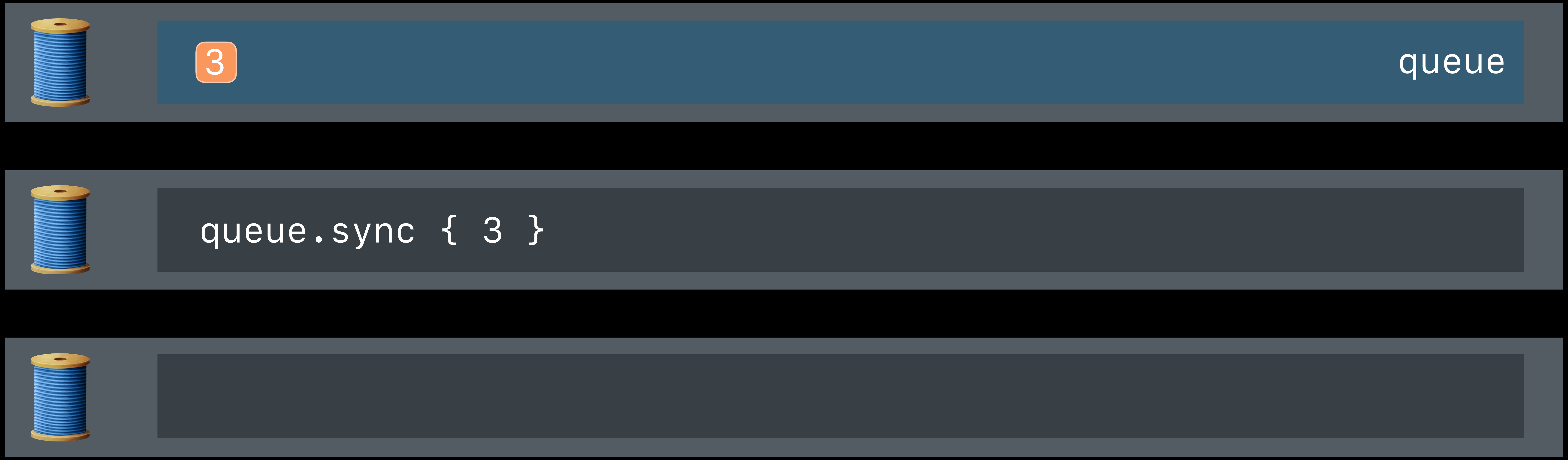
Serial Dispatch Queue



```
queue.sync { 3 }
```

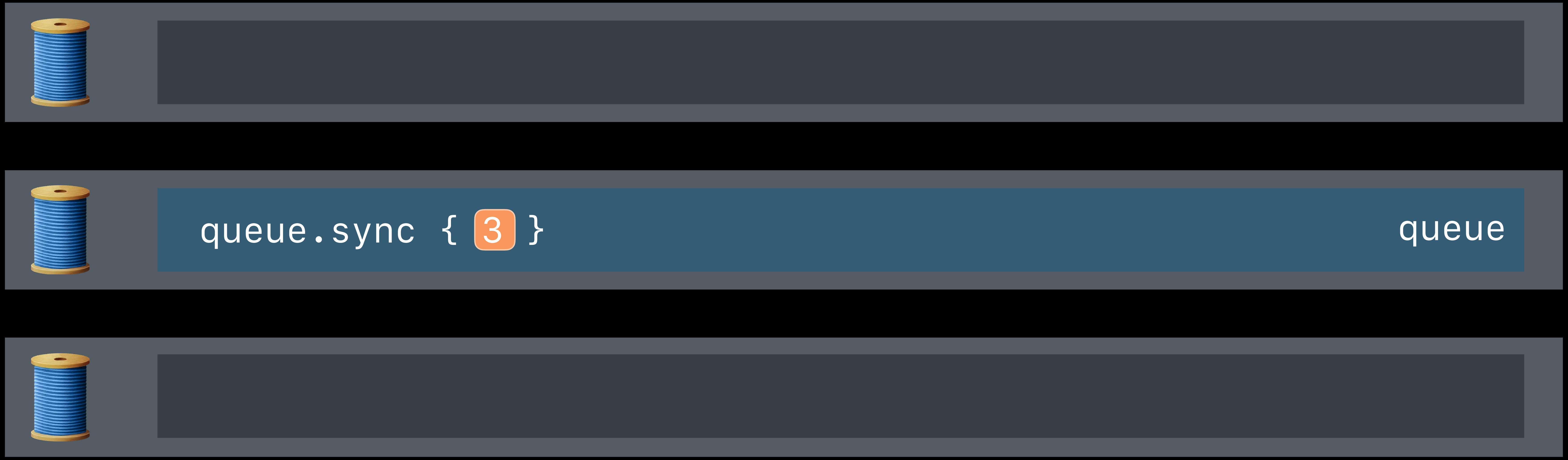
```
let queue = DispatchQueue(label: "com.example.queue")
queue.async { /*  1  */ }
queue.async { /*  2  */ }
queue.sync  { /*  3  */ }
```

Serial Dispatch Queue



```
let queue = DispatchQueue(label: "com.example.queue")
queue.async { /*  1  */ }
queue.async { /*  2  */ }
queue.sync  { /*  3  */ }
```

Serial Dispatch Queue



```
let queue = DispatchQueue(label: "com.example.queue")
queue.async { /*  1  */ }
queue.async { /*  2  */ }
queue.sync  { /*  3  */ }
```

Dispatch Source

Event monitoring primitive

```
let source = DispatchSource.makeReadSource(fileDescriptor: fd, queue: queue)
source.setEventHandler  { read(fd) }
source.setCancelHandler { close(fd) }
source.activate()
```

Dispatch Source

Event monitoring primitive

```
let source = DispatchSource.makeReadSource(fileDescriptor: fd, queue: queue)
source.setEventHandler  { read(fd) }
source.setCancelHandler { close(fd) }
source.activate()
```

Dispatch Source

Event monitoring primitive

- Event handler executes on target queue

```swift
let source = DispatchSource.makeReadSource(fileDescriptor: fd, queue: queue)
source.setEventHandler  { read(fd) }
source.setCancelHandler { close(fd) }
source.activate()
```

Dispatch Source

Event monitoring primitive

- Event handler executes on target queue
- Invalidation pattern with explicit cancellation

```
let source = DispatchSource.makeReadSource(fileDescriptor: fd, queue: queue)
source.setEventHandler  { read(fd) }
source.setCancelHandler { close(fd) }
source.activate()
```

Dispatch Source

Event monitoring primitive

- Event handler executes on target queue
- Invalidation pattern with explicit cancellation
- Initial setup followed by activate

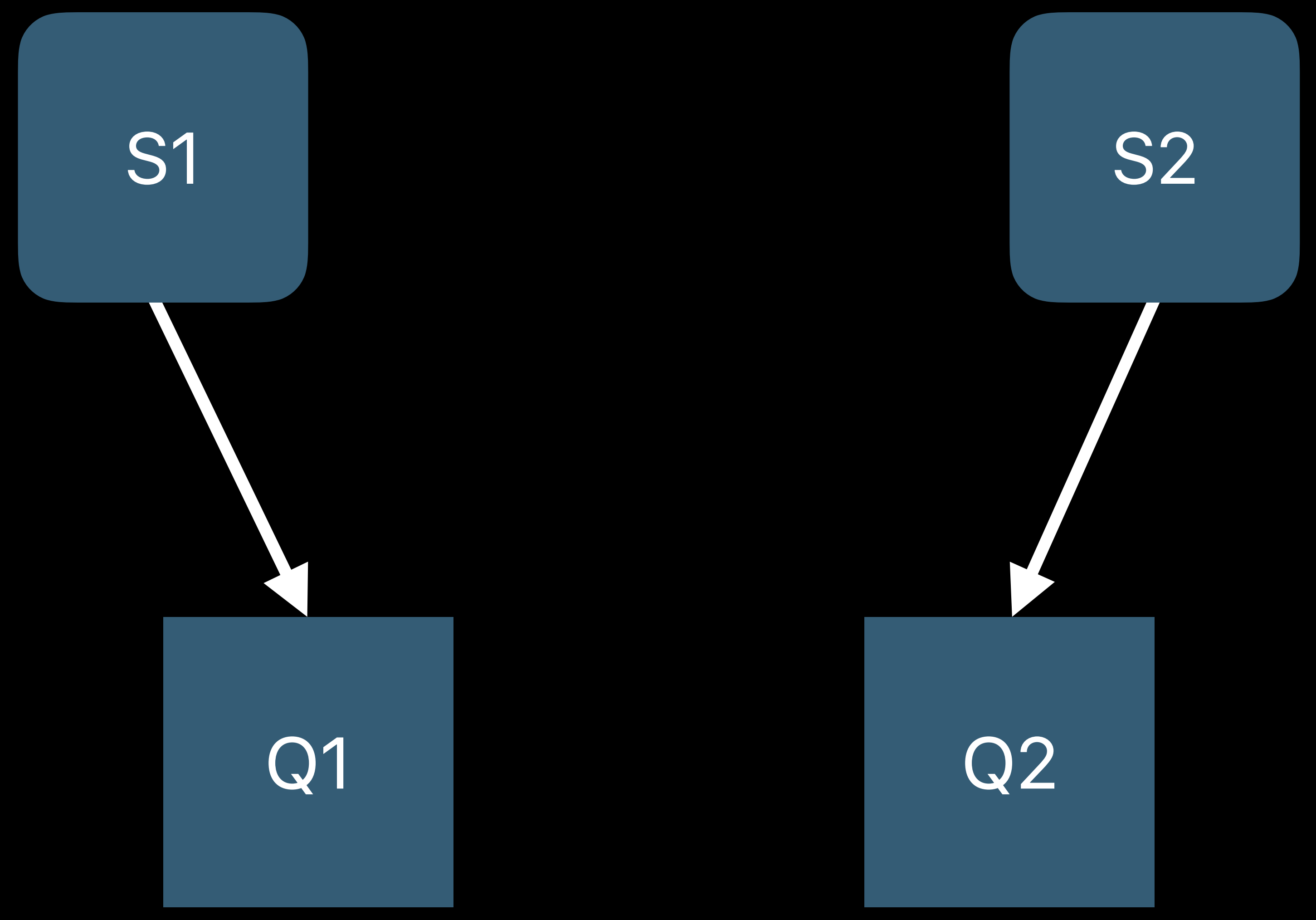
```
let source = DispatchSource.makeReadSource(fileDescriptor: fd, queue: queue)
source.setEventHandler  { read(fd) }
source.setCancelHandler { close(fd) }
source.activate()
```

Target Queue Hierarchy

Serial queues and sources can form a tree

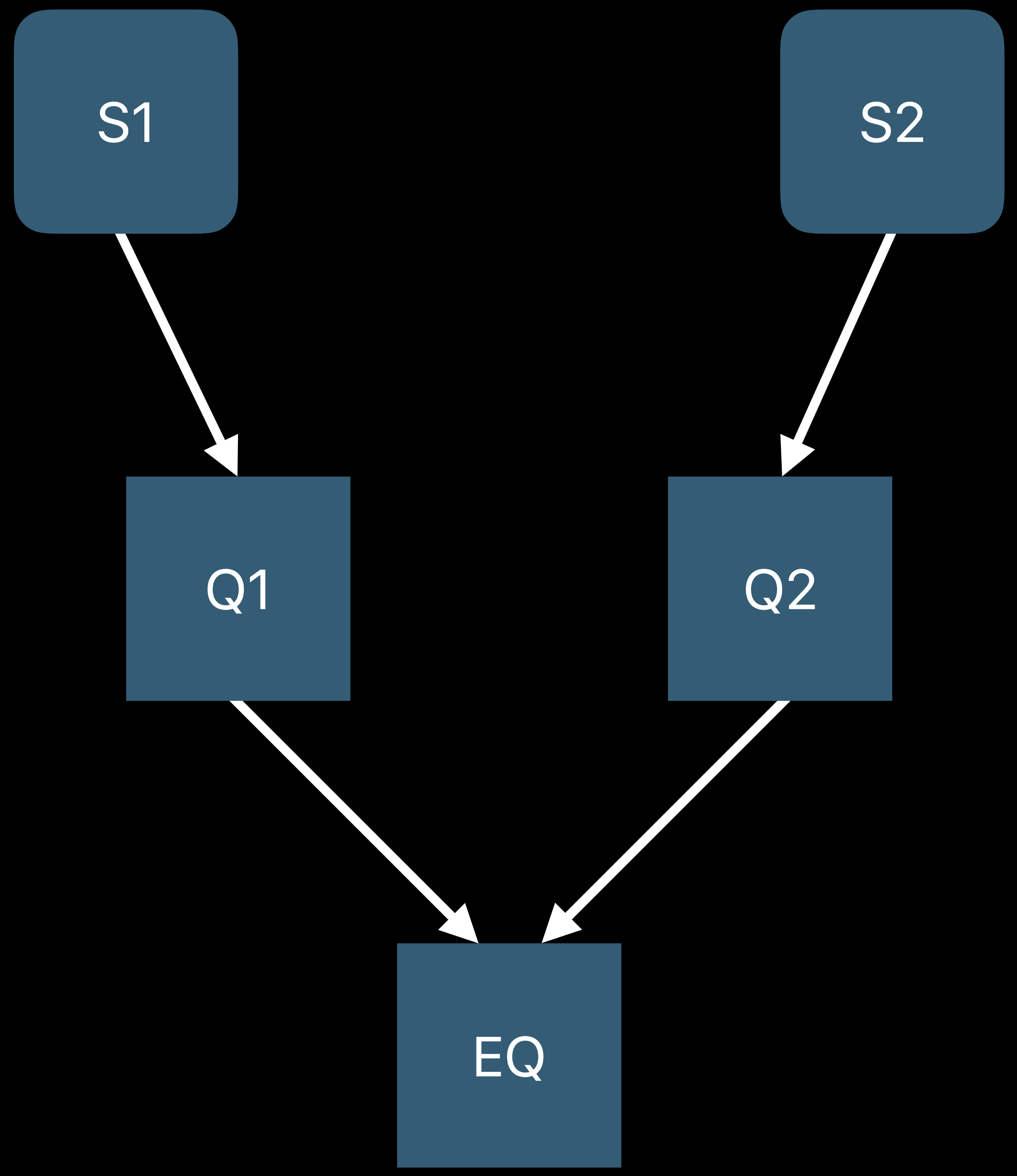
Target Queue Hierarchy

Serial queues and sources can form a tree



Target Queue Hierarchy

Serial queues and sources can form a tree



```
let Q1 = DispatchQueue(label: "Q1", target: EQ )
let Q2 = DispatchQueue(label: "Q2", target: EQ )
```

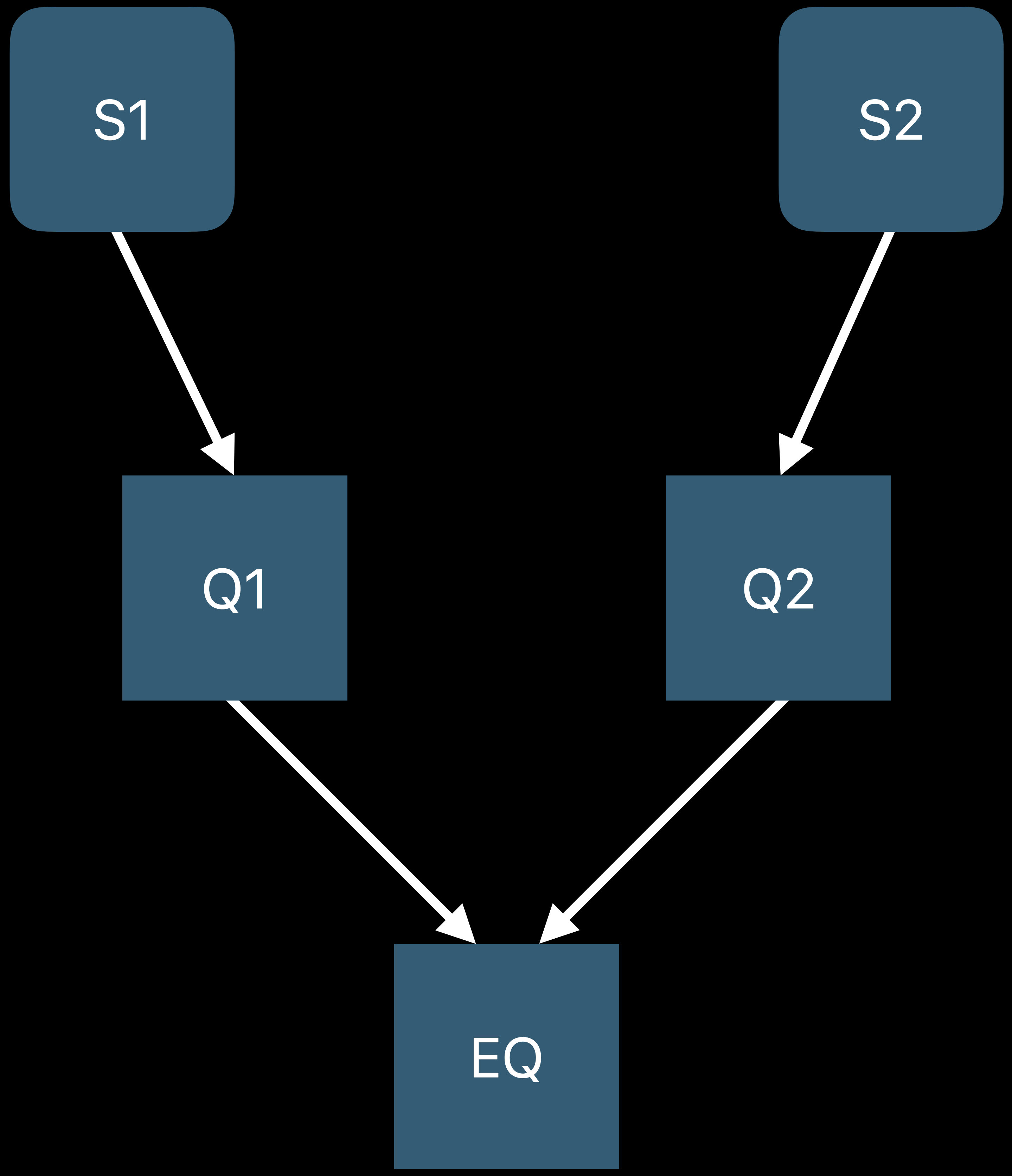
Target Queue Hierarchy

Serial queues and sources can form a tree

Shared single mutual exclusion context

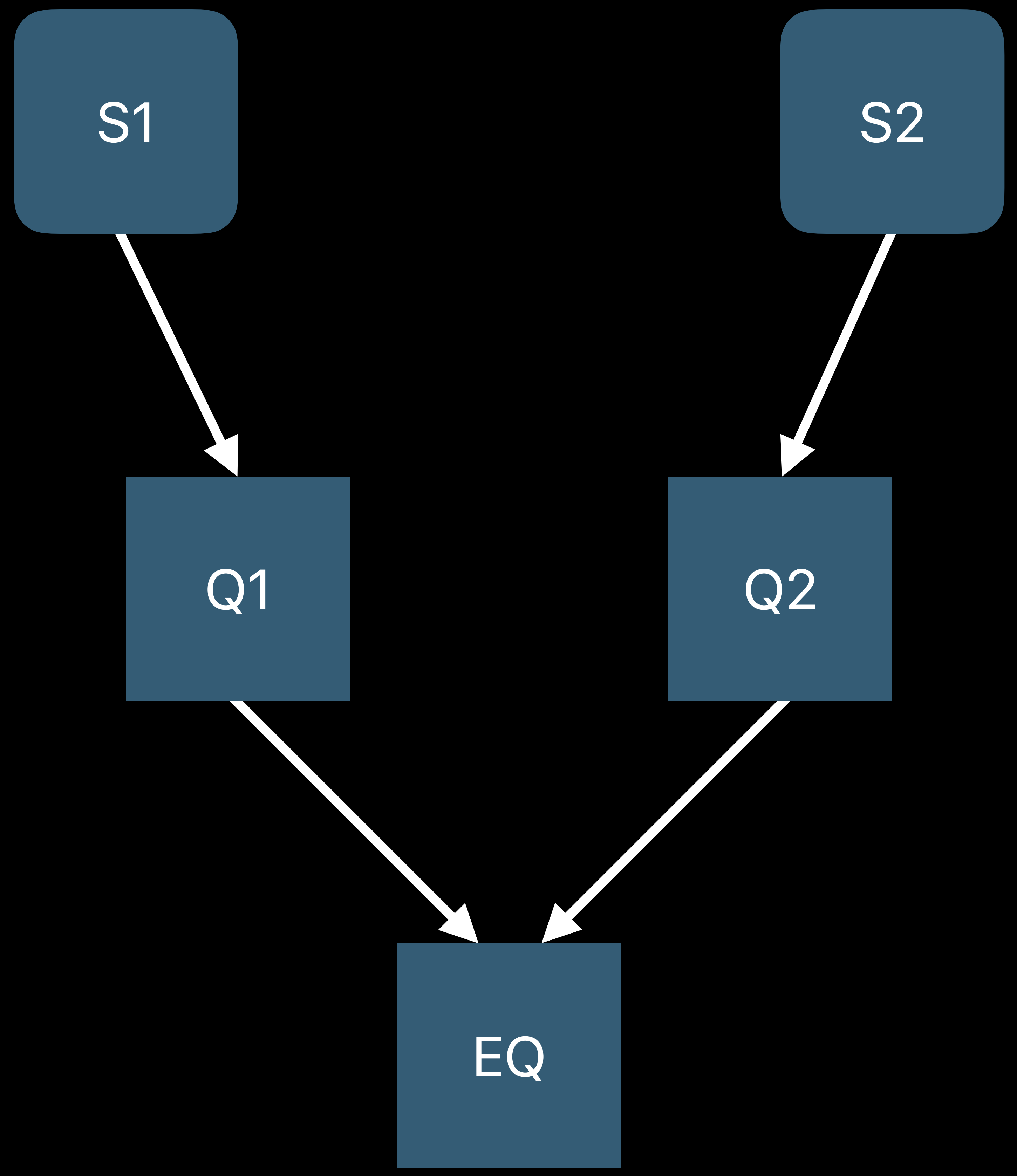
Independent individual queue order

```
let Q1 = DispatchQueue(label: "Q1", target: EQ )
let Q2 = DispatchQueue(label: "Q2", target: EQ )
```

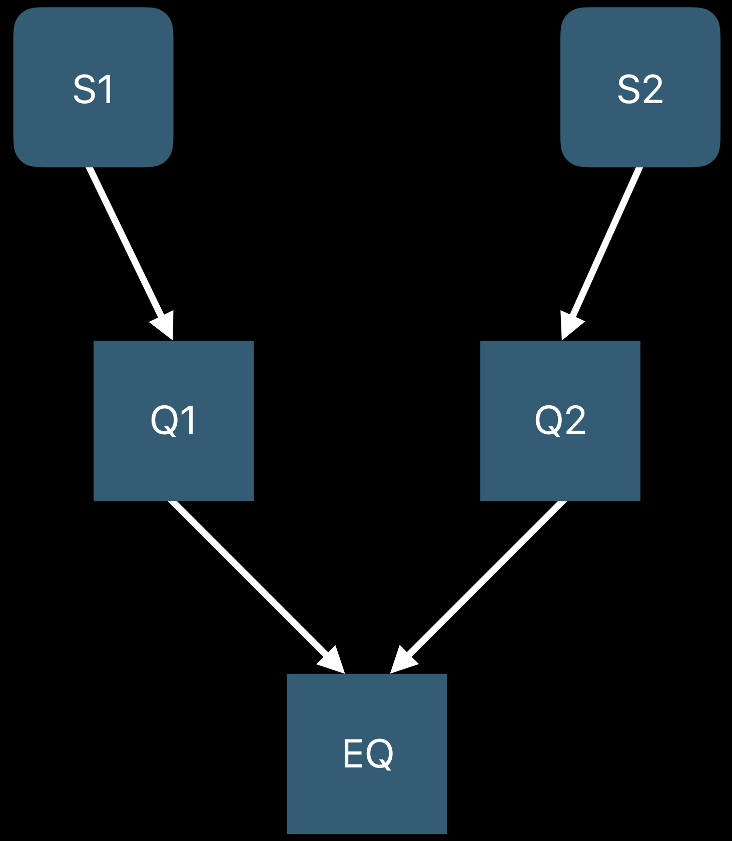


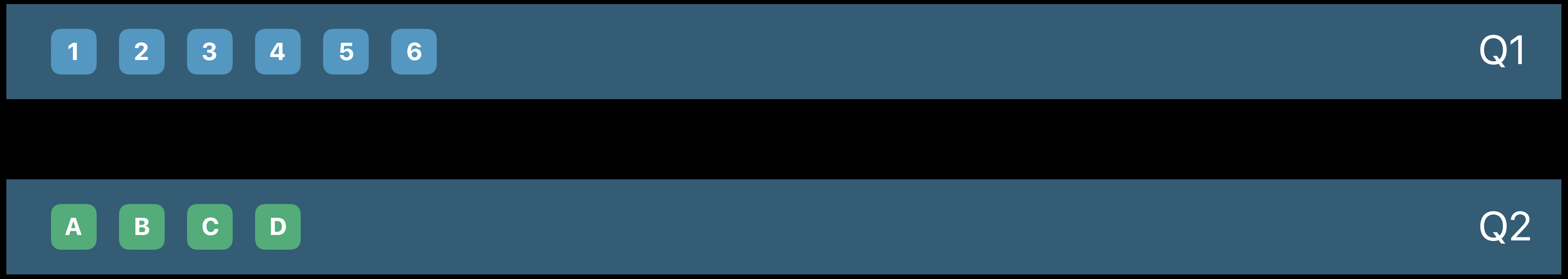
$$\begin{tikzcd}
S1 \arrow[d] & & S2 \arrow[d] \\
Q1 \arrow[dr] & & Q2 \arrow[dl] \\
& EQ &
\end{tikzcd}$$

Target Queue Hierarchy

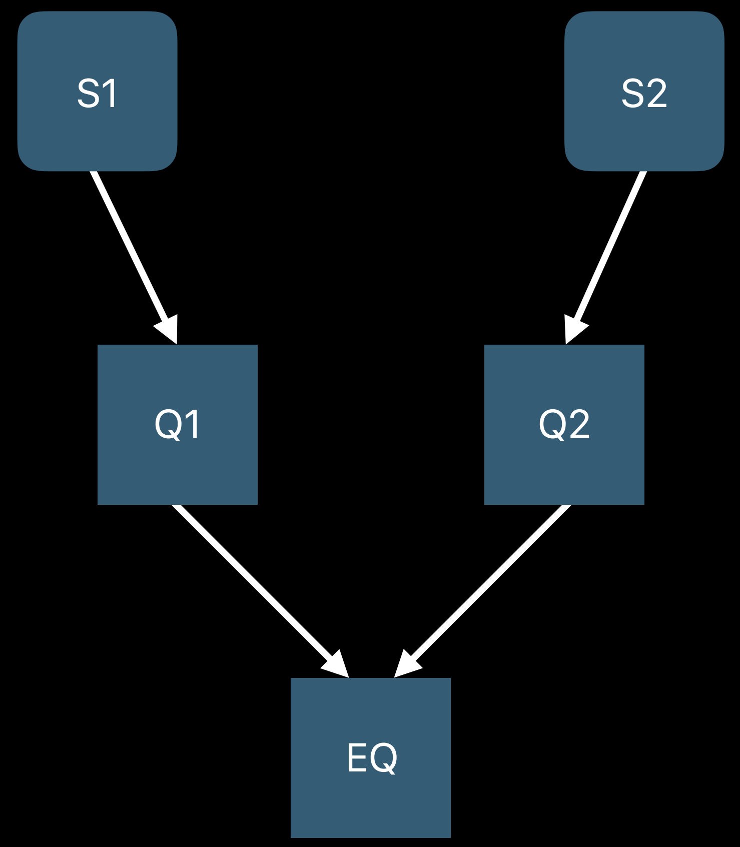

$$\begin{tikzcd}
S1 \arrow[d] & & S2 \arrow[d] \\
Q1 \arrow[dr] & & Q2 \arrow[dl] \\
& EQ &
\end{tikzcd}$$

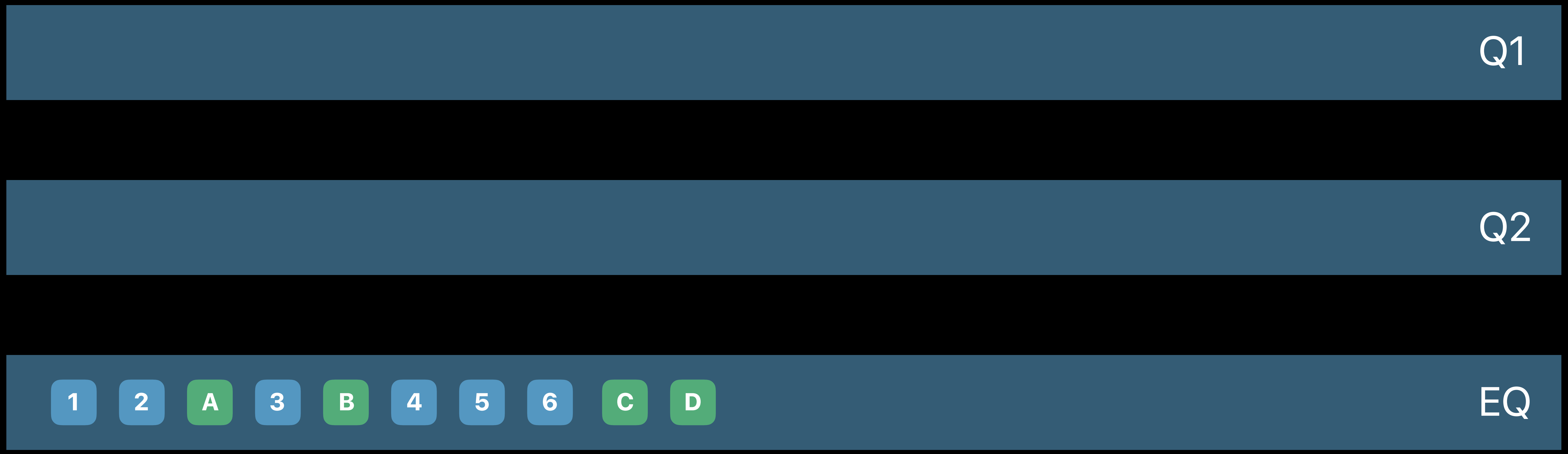
Target Queue Hierarchy



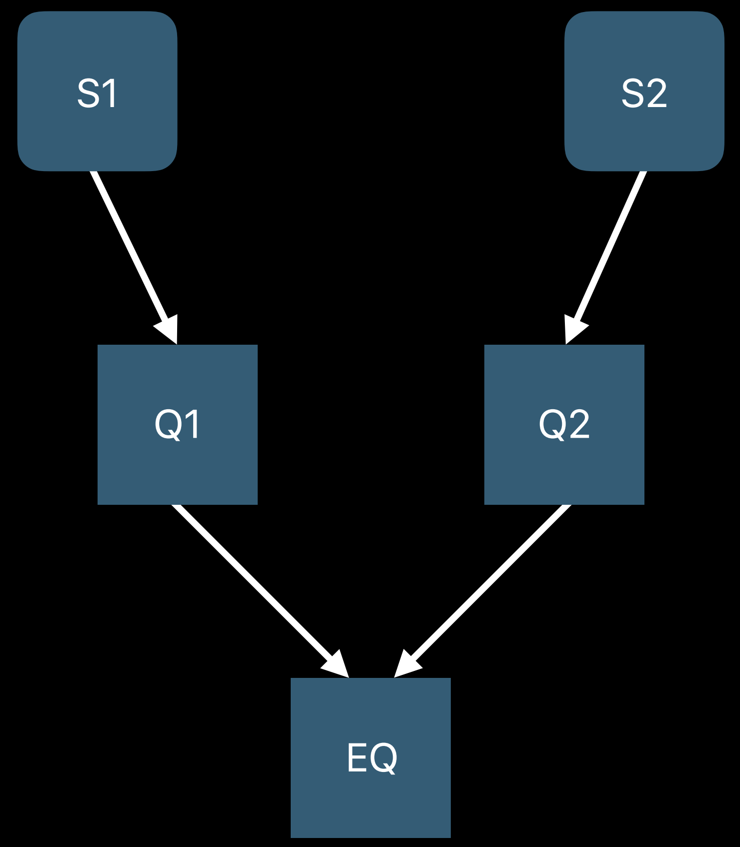


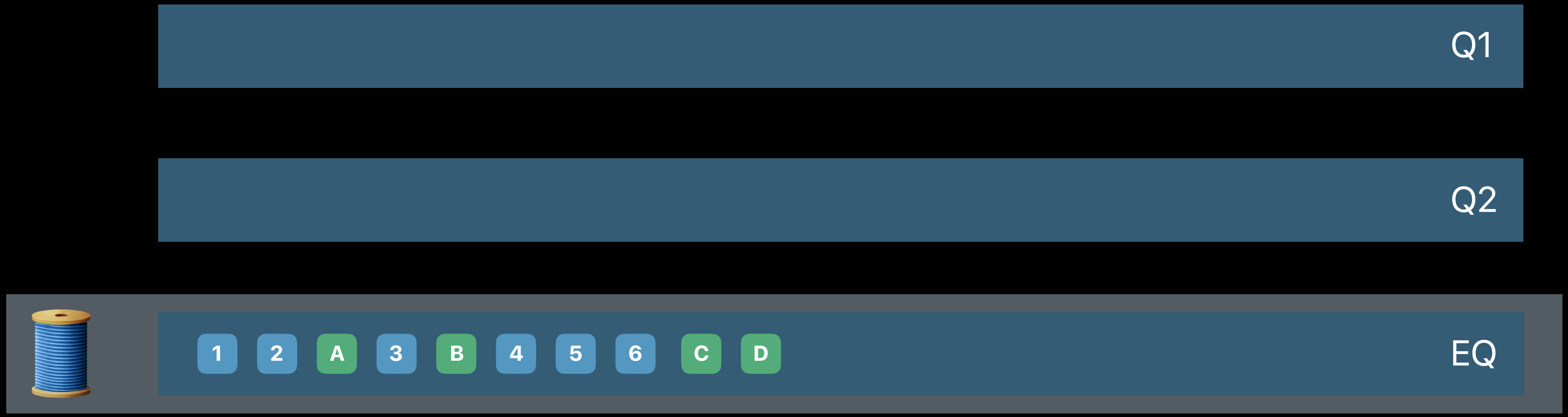
Target Queue Hierarchy





Target Queue Hierarchy





Quality of Service

Abstract notion of priority

Provides explicit classification of your work

Affects various execution properties

User Interactive

User Initiated

Utility

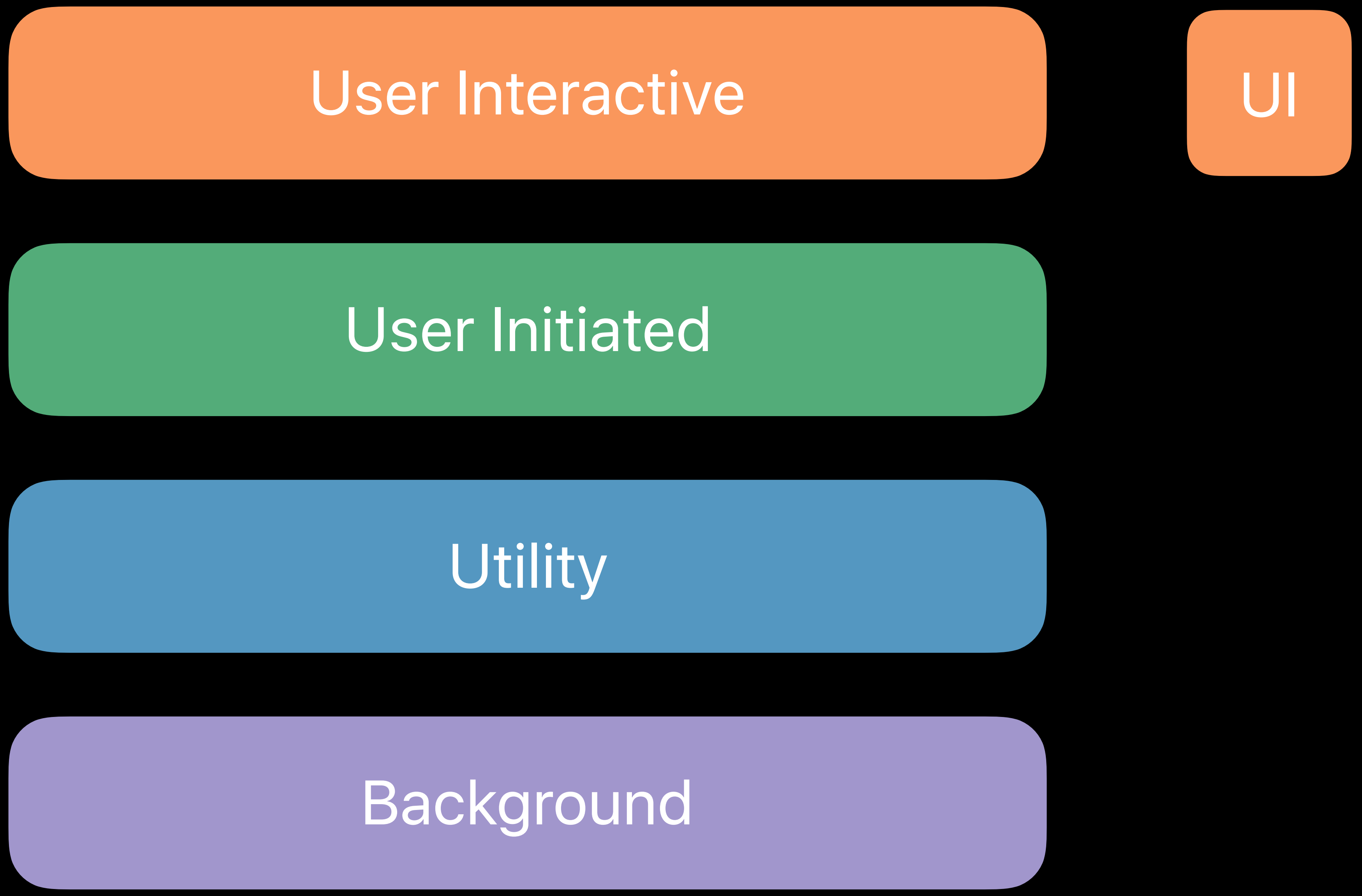
Background

Quality of Service

Abstract notion of priority

Provides explicit classification of your work

Affects various execution properties

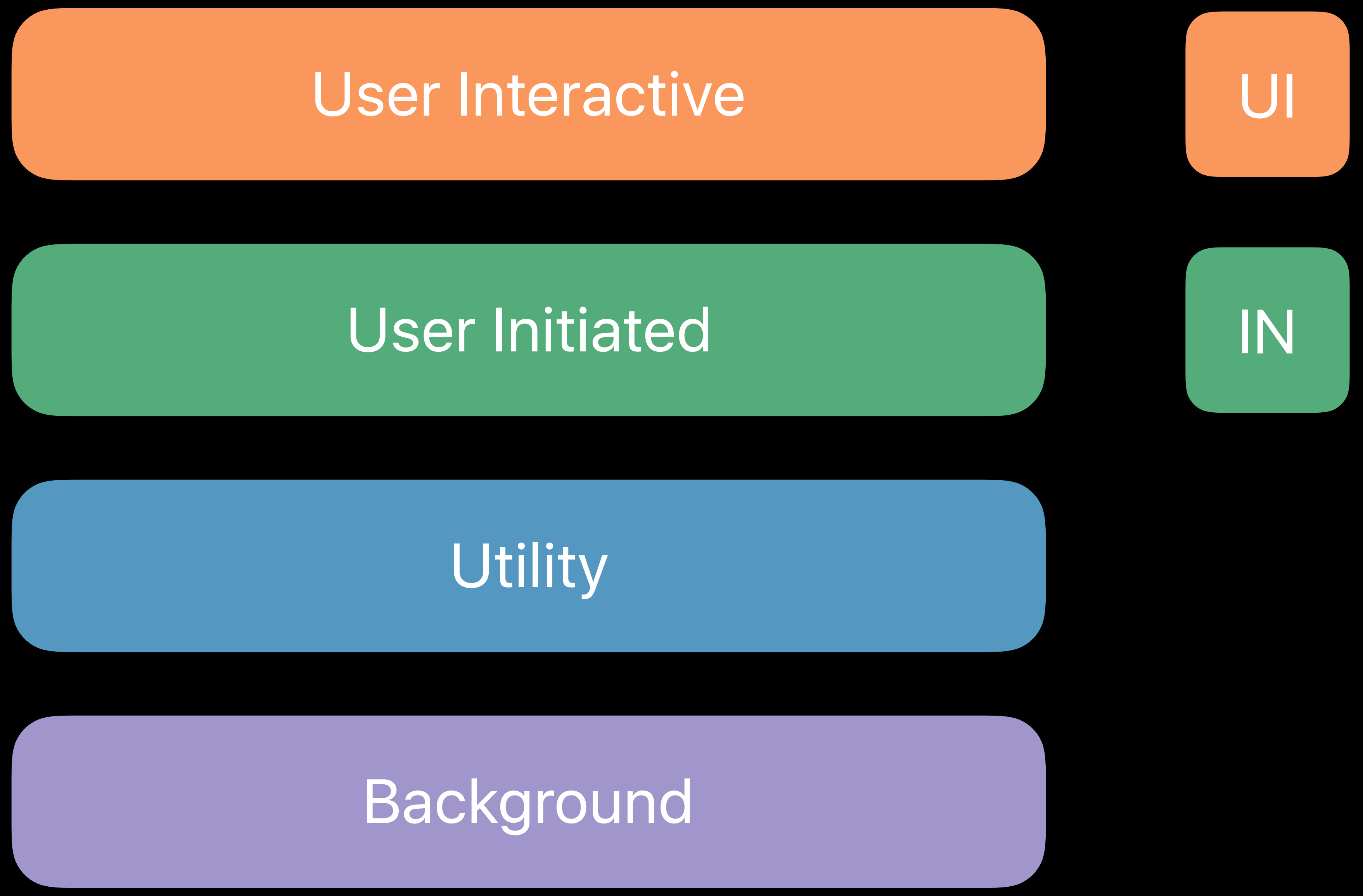


Quality of Service

Abstract notion of priority

Provides explicit classification of your work

Affects various execution properties

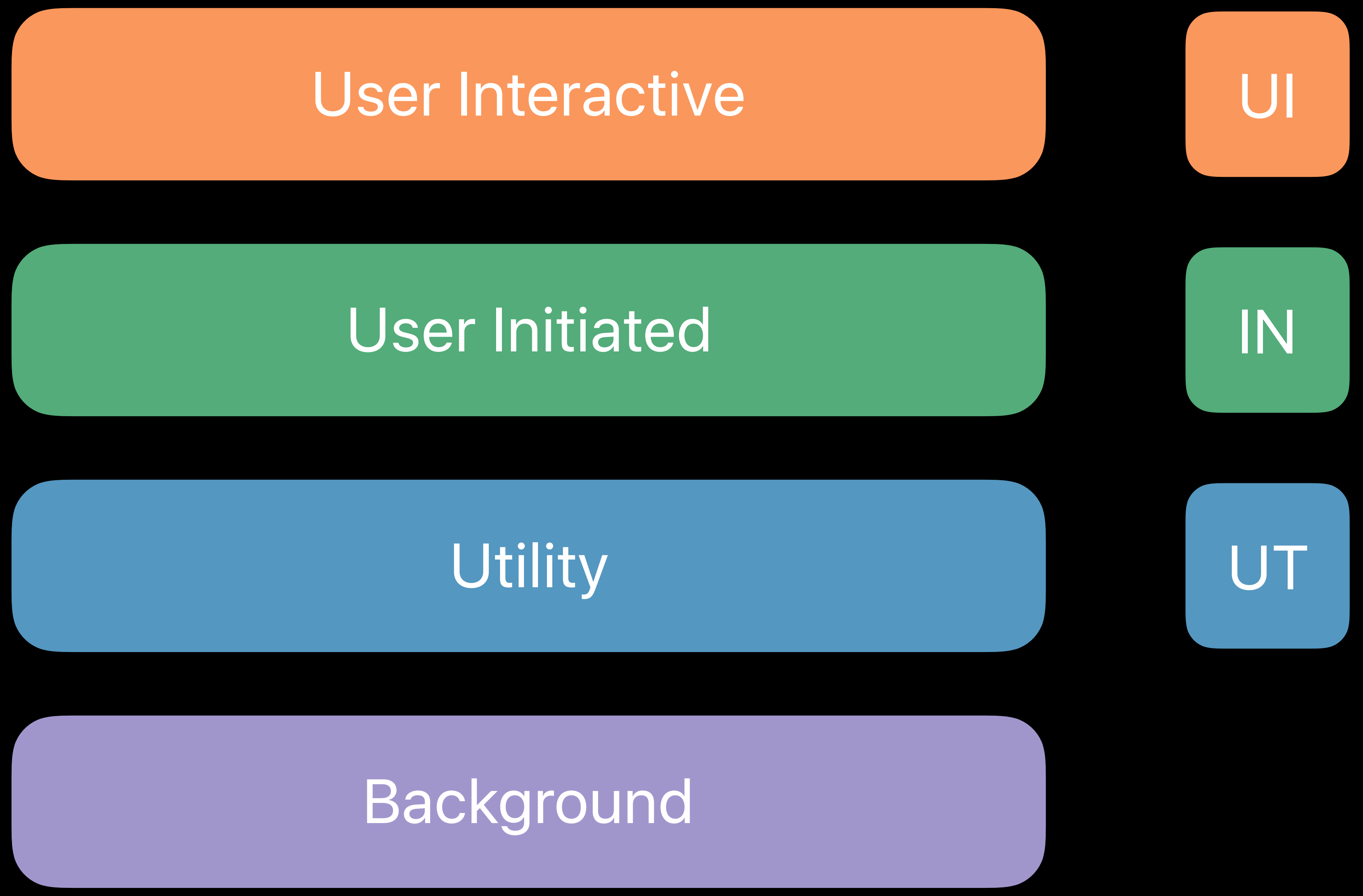


Quality of Service

Abstract notion of priority

Provides explicit classification of your work

Affects various execution properties

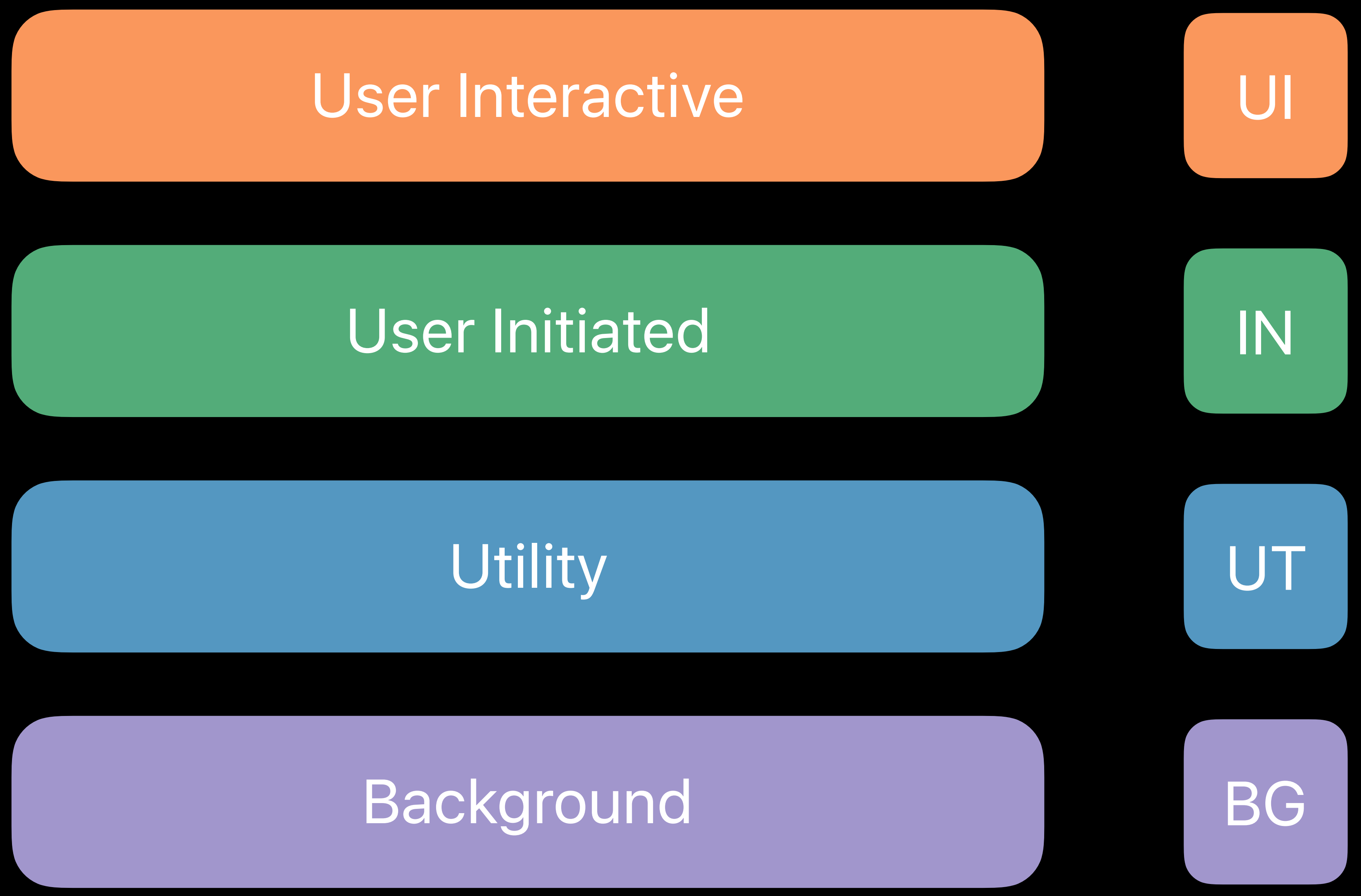


Quality of Service

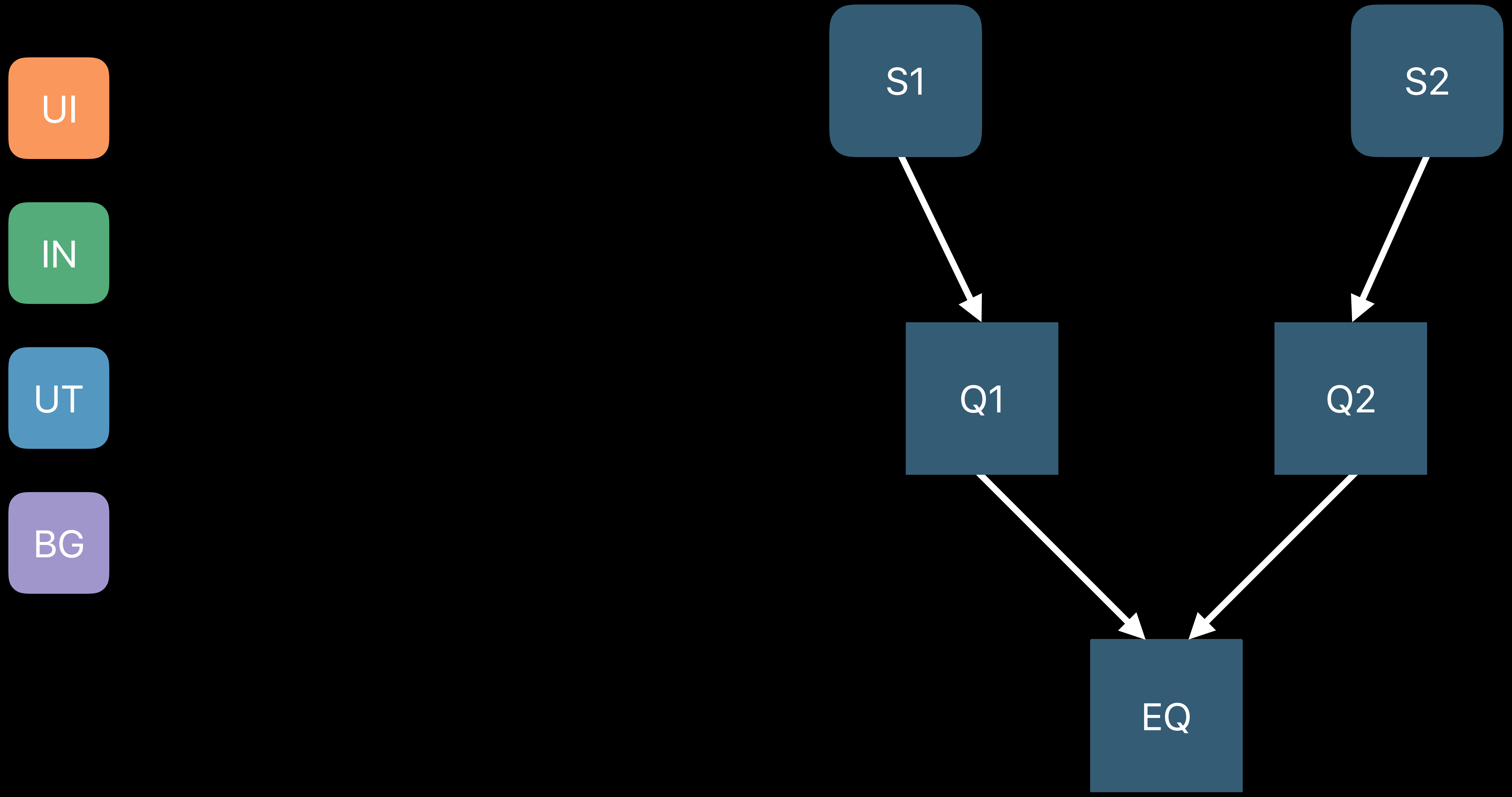
Abstract notion of priority

Provides explicit classification of your work

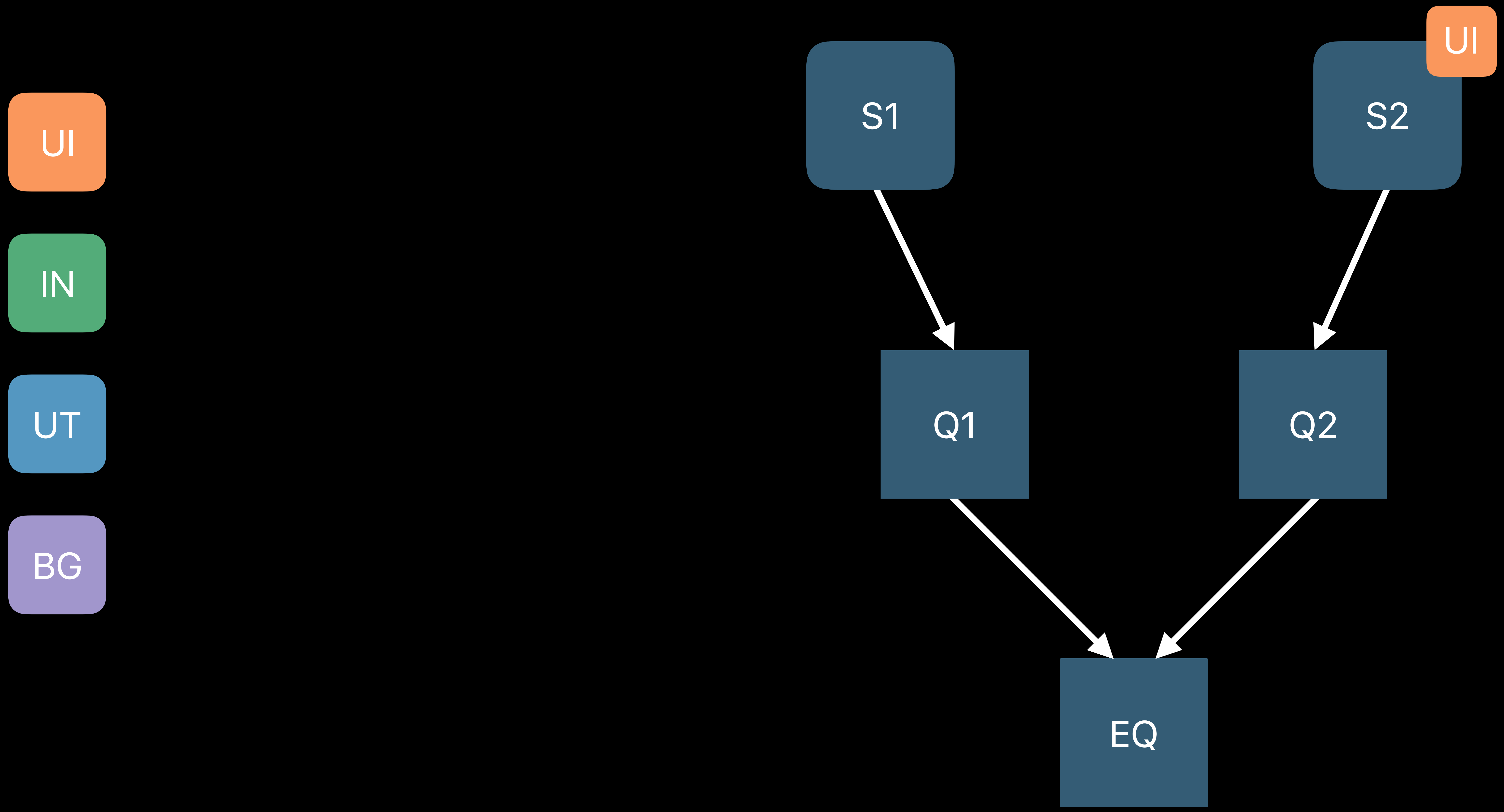
Affects various execution properties



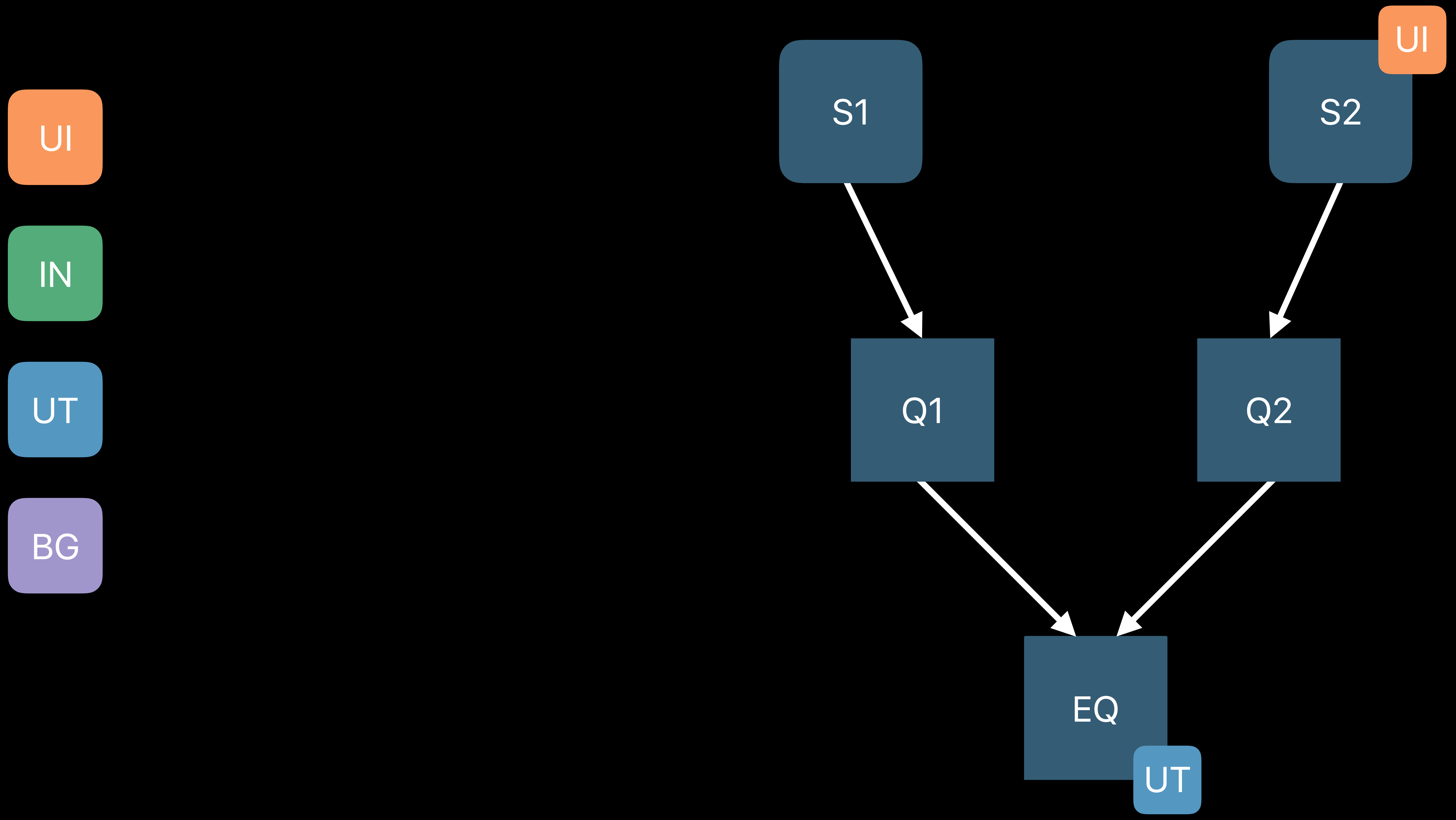
QoS and Target Queue Hierarchy



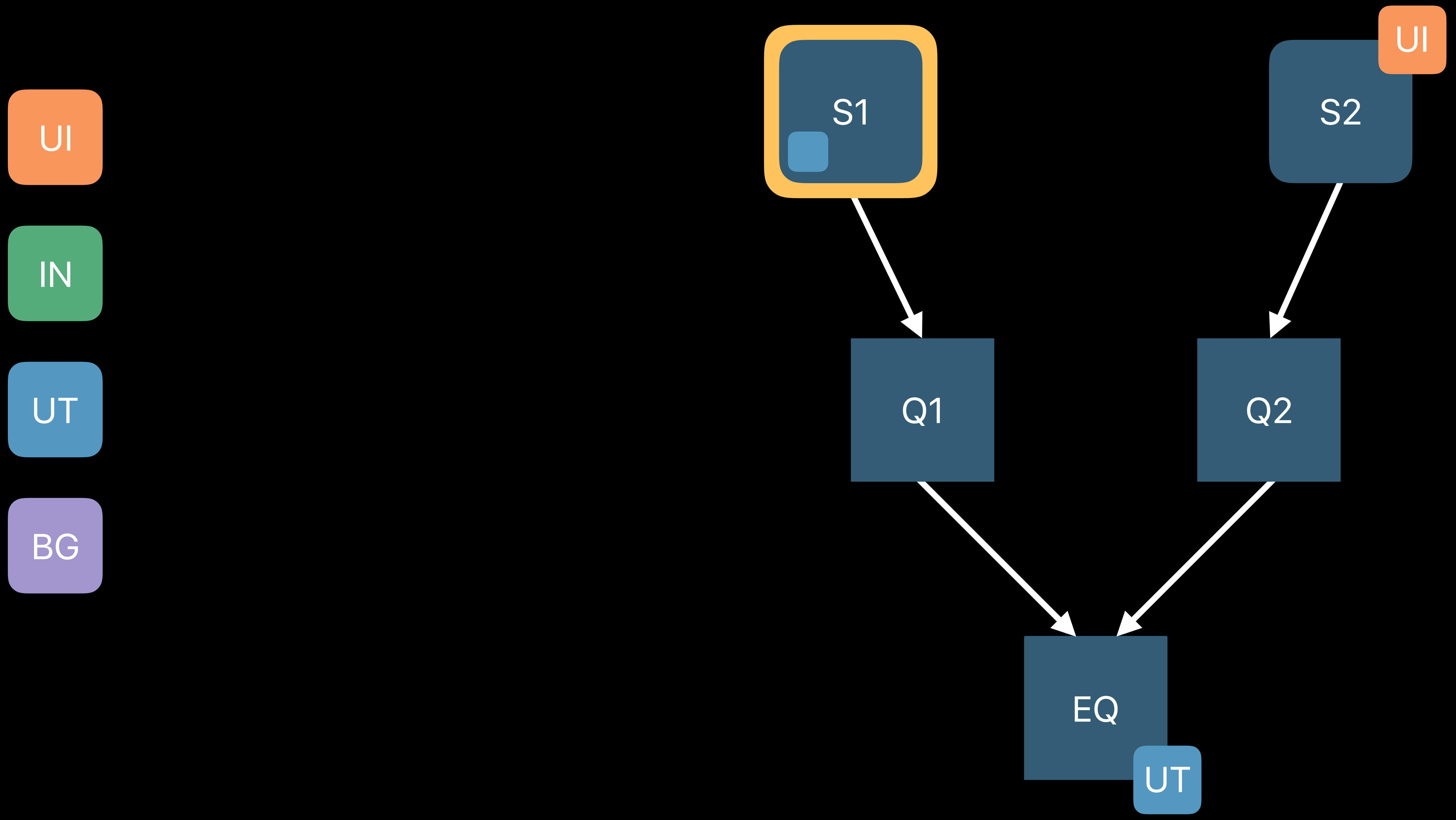
QoS and Target Queue Hierarchy



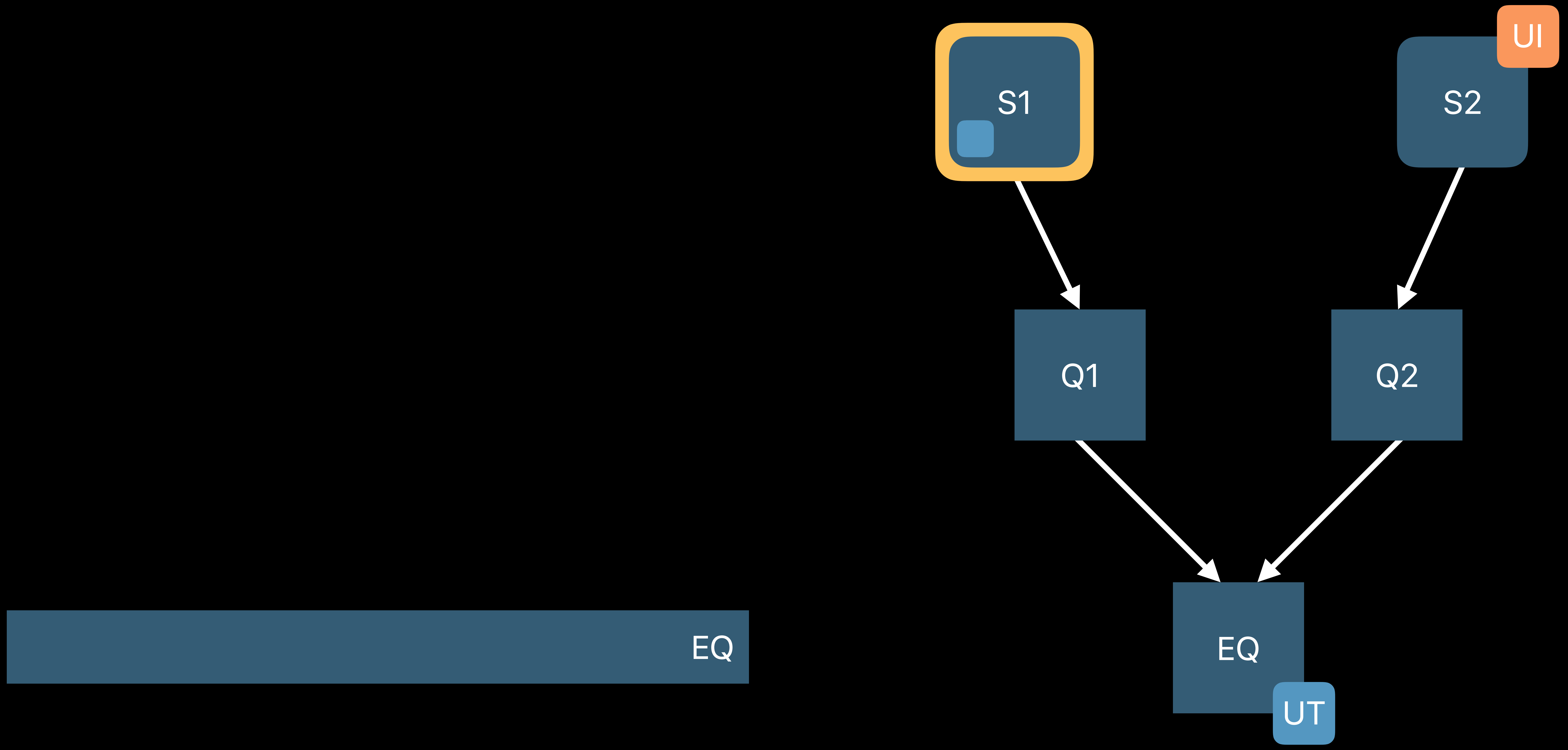
QoS and Target Queue Hierarchy



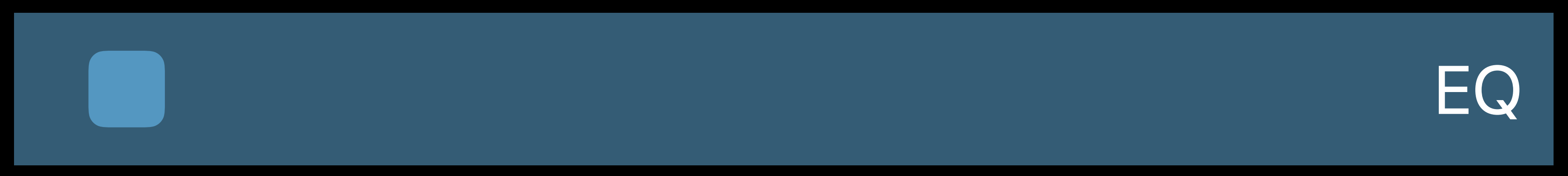
QoS and Target Queue Hierarchy

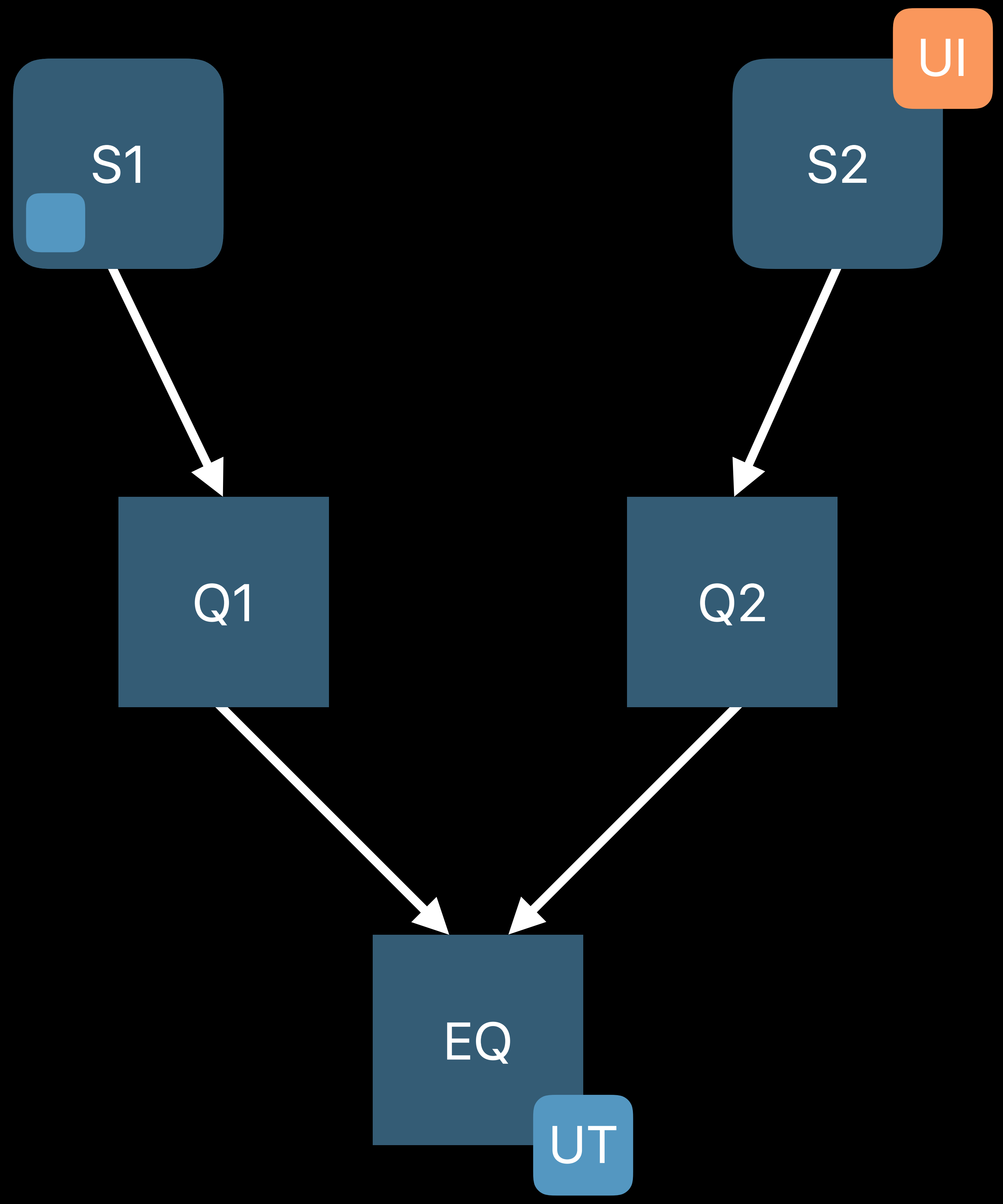


QoS and Target Queue Hierarchy

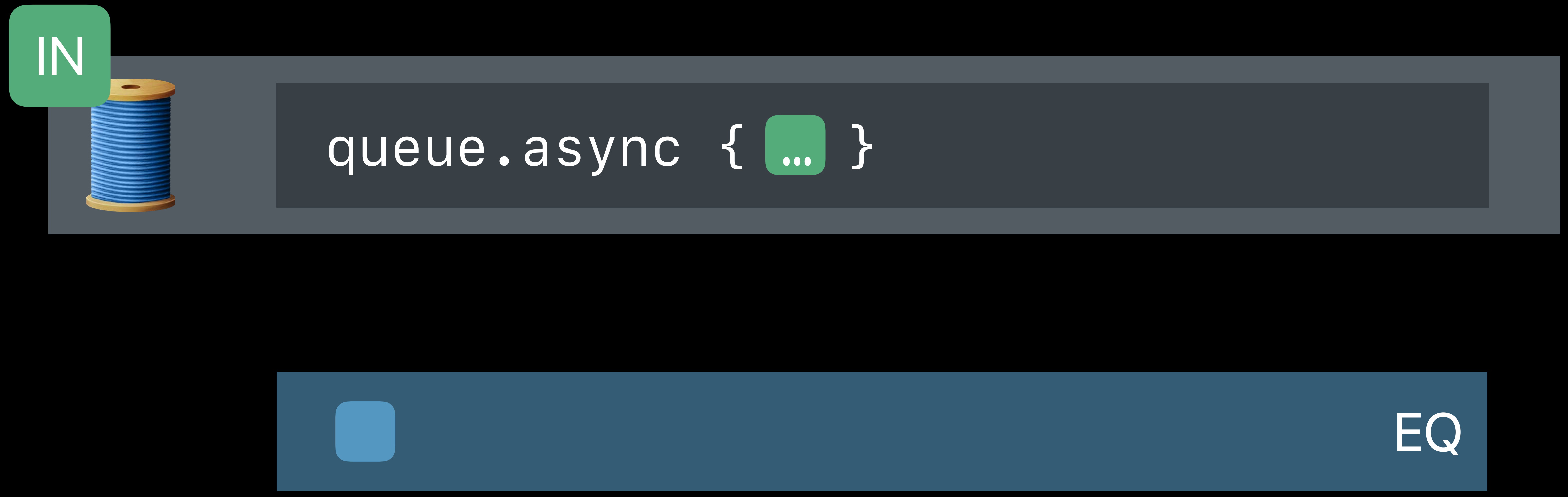


QoS and Target Queue Hierarchy

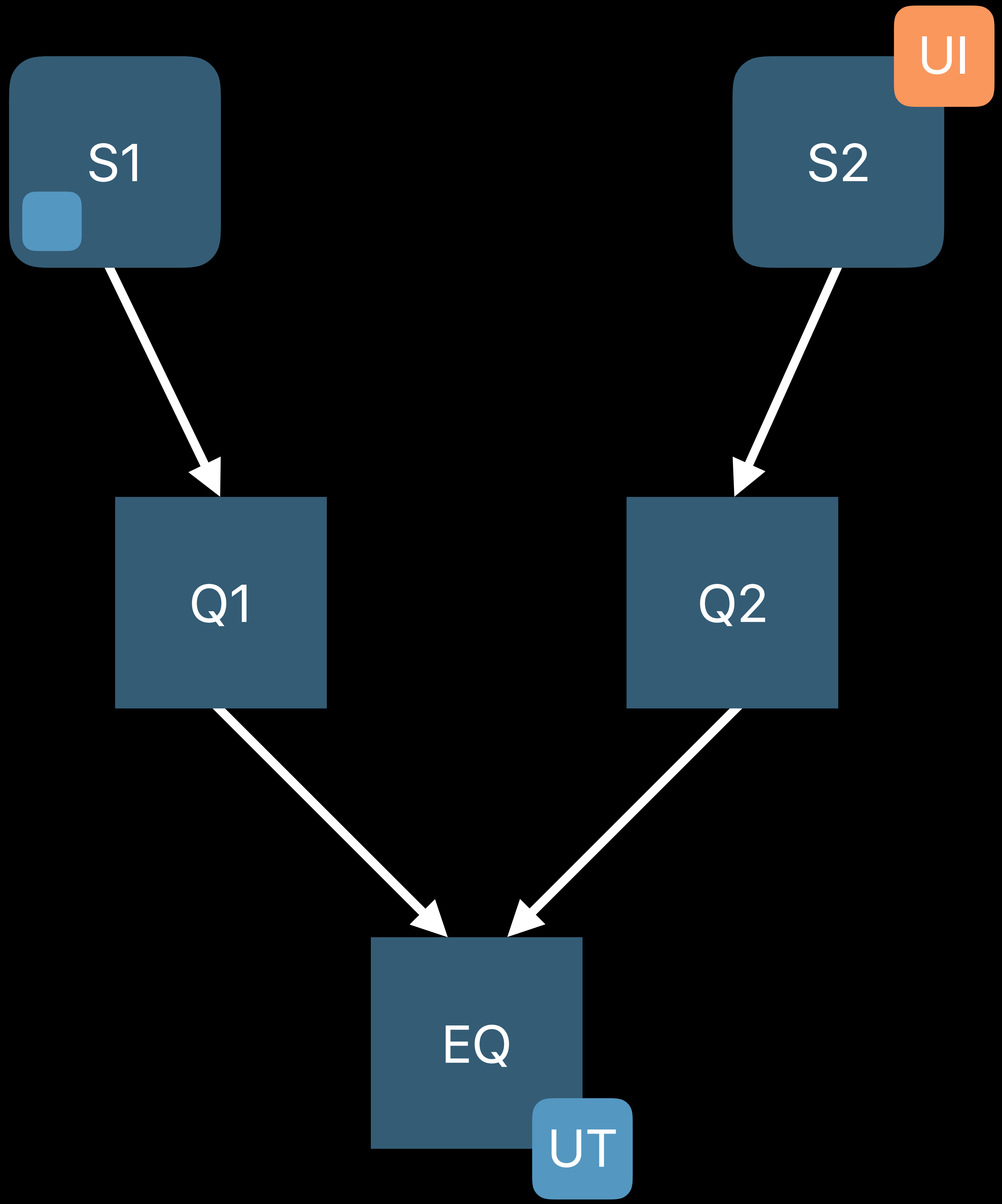




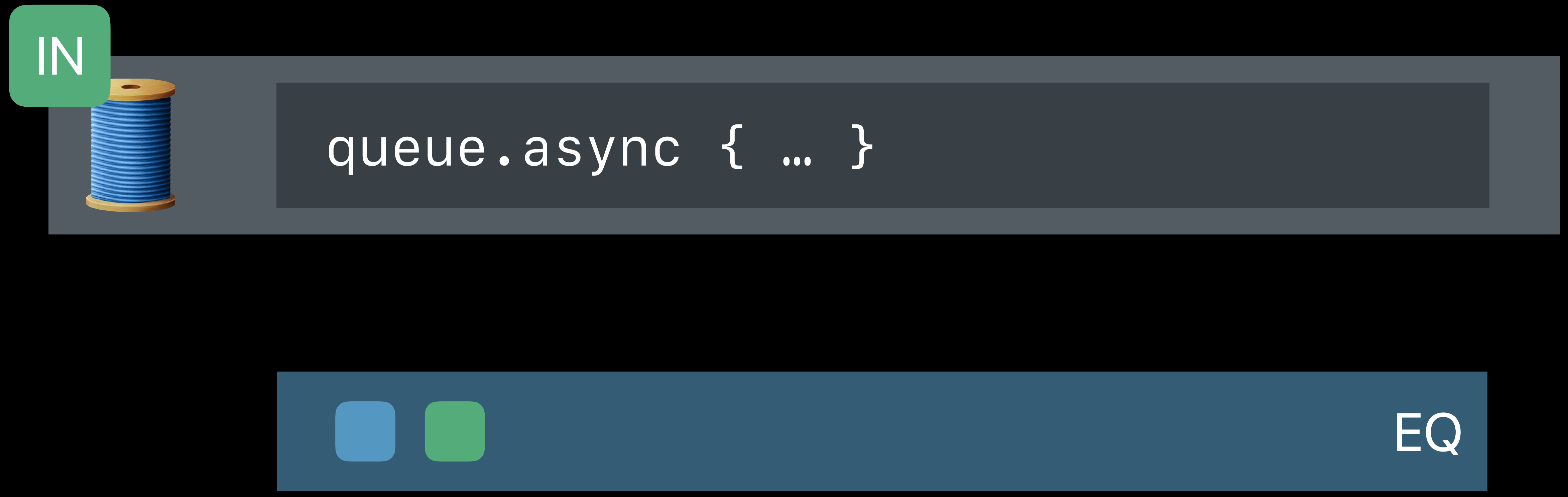
QoS and Target Queue Hierarchy



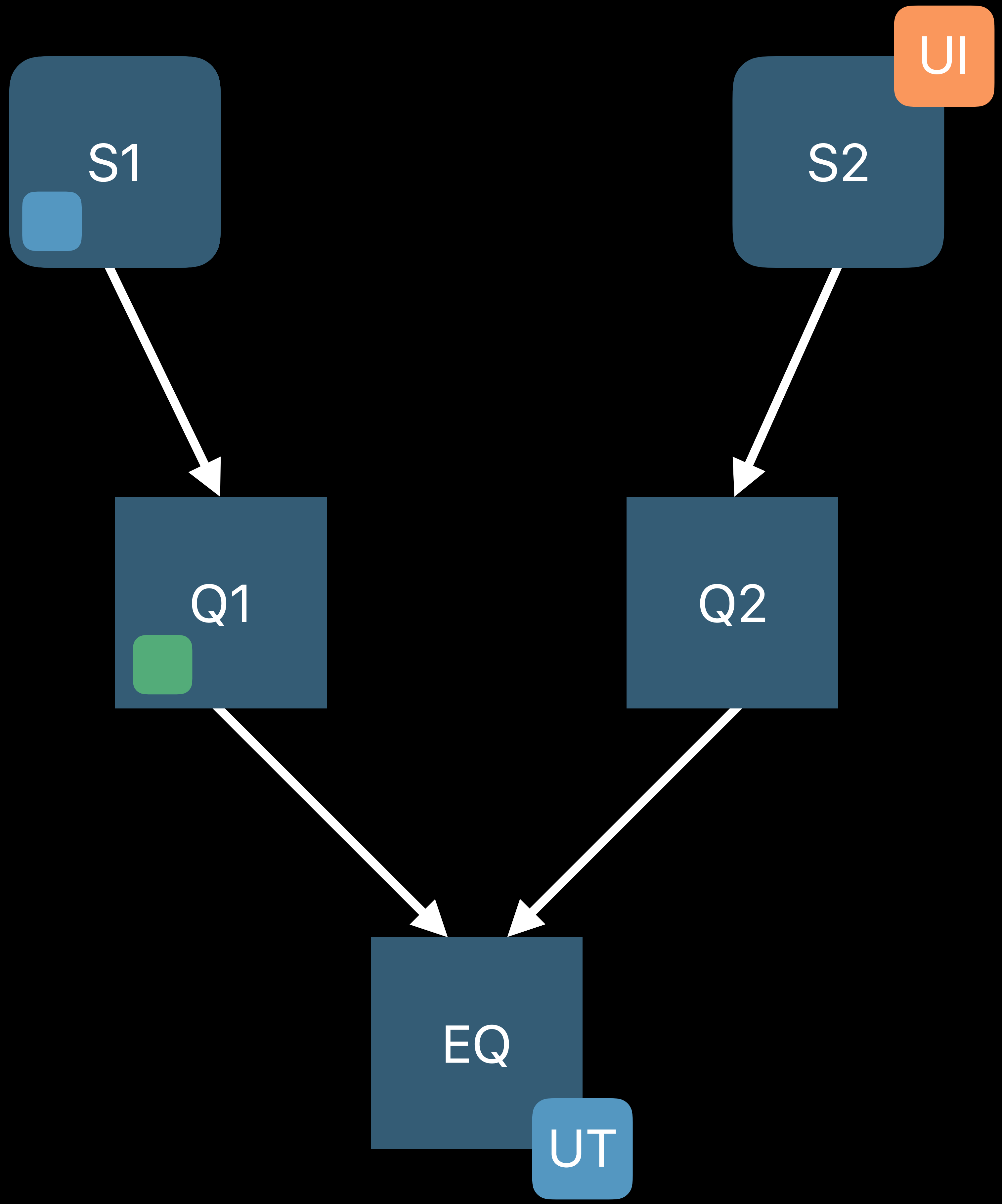
```
queue.async { … }
```



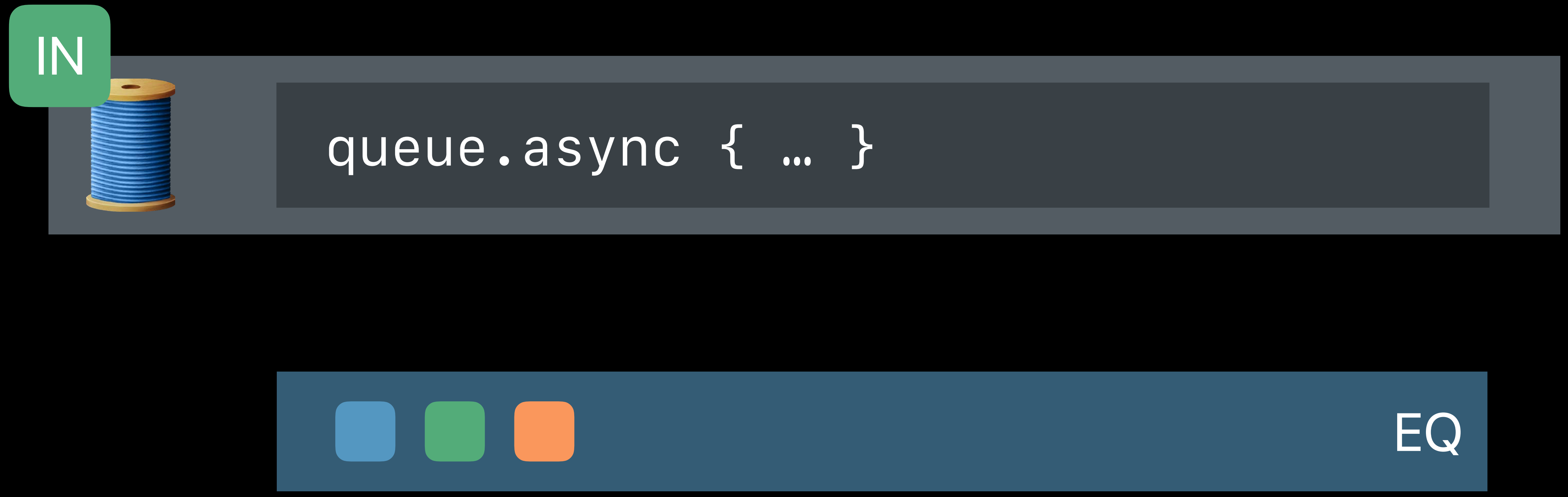
QoS and Target Queue Hierarchy



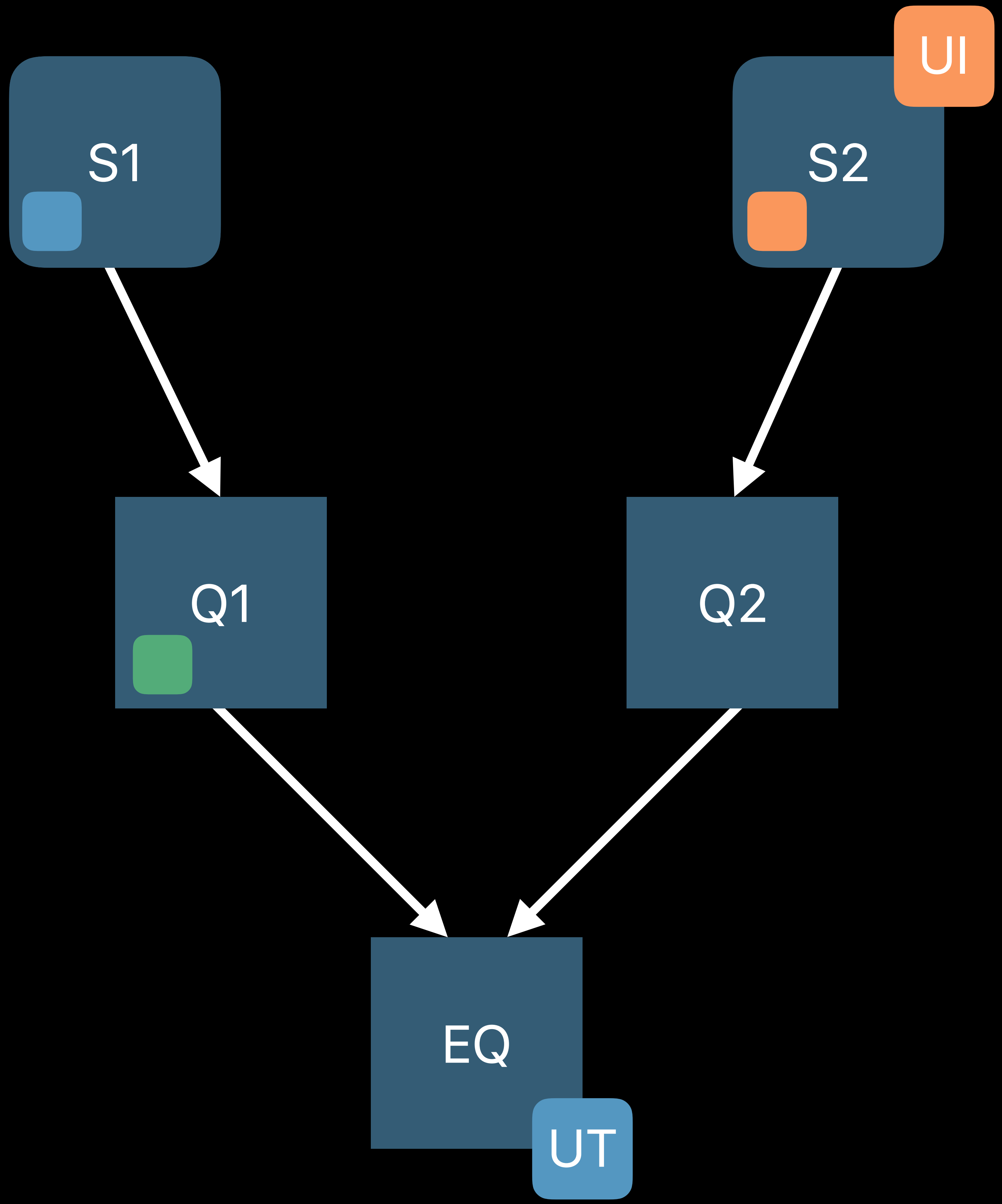
```
queue.async { … }
```



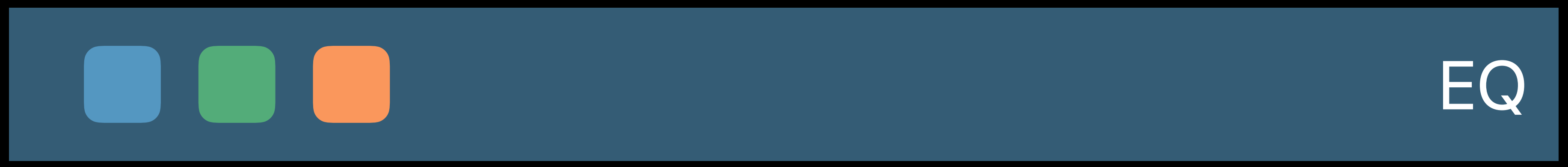
QoS and Target Queue Hierarchy

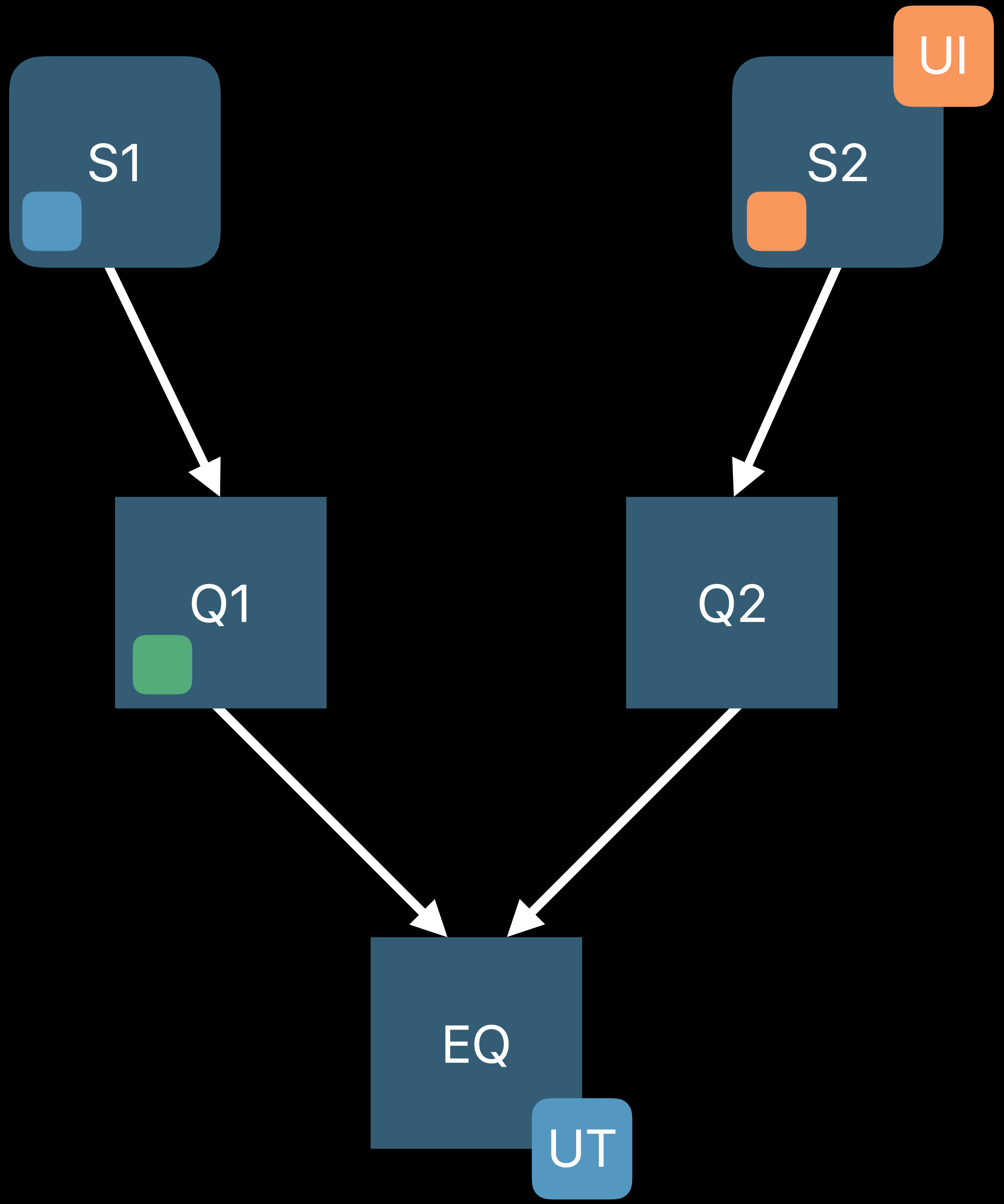


```
queue.async { … }
```

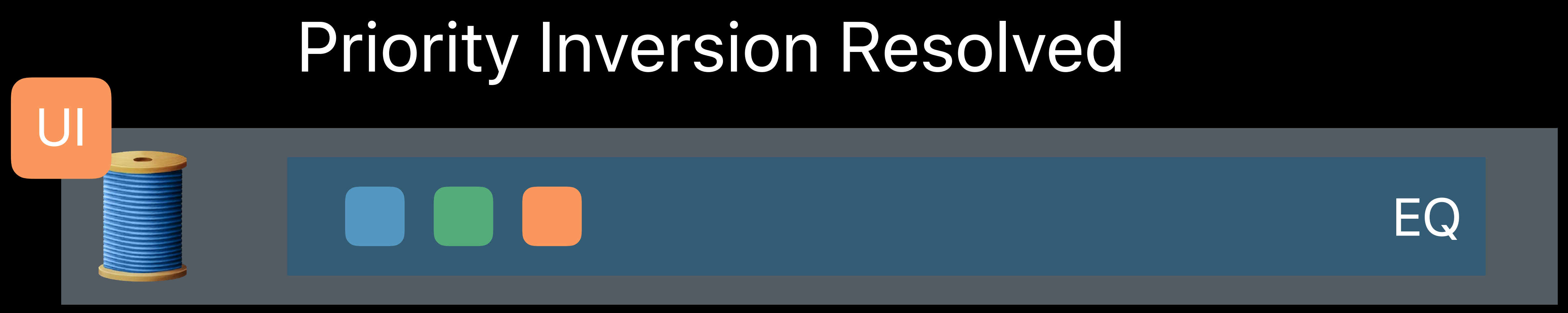


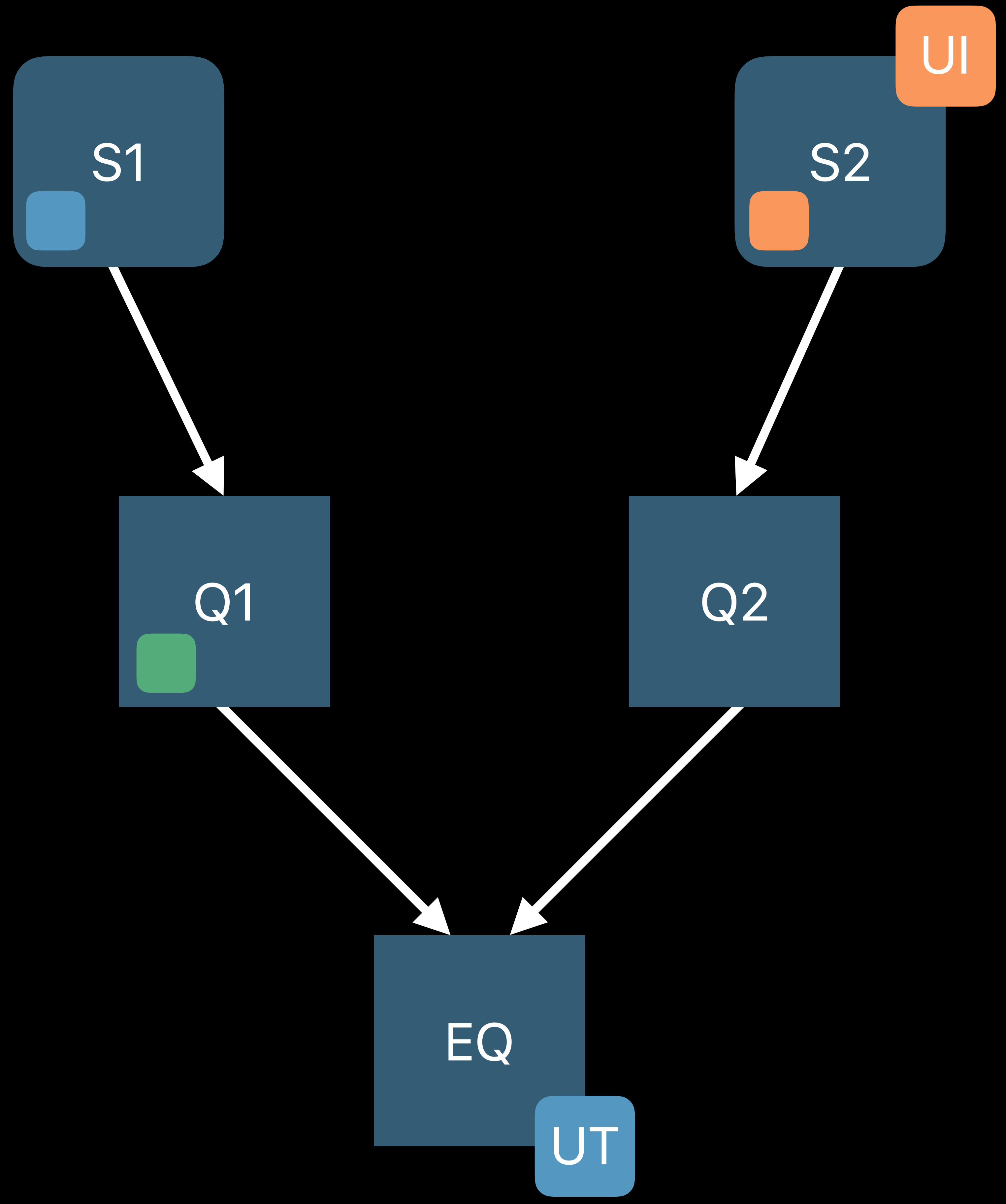
QoS and Target Queue Hierarchy


Priority Inversion

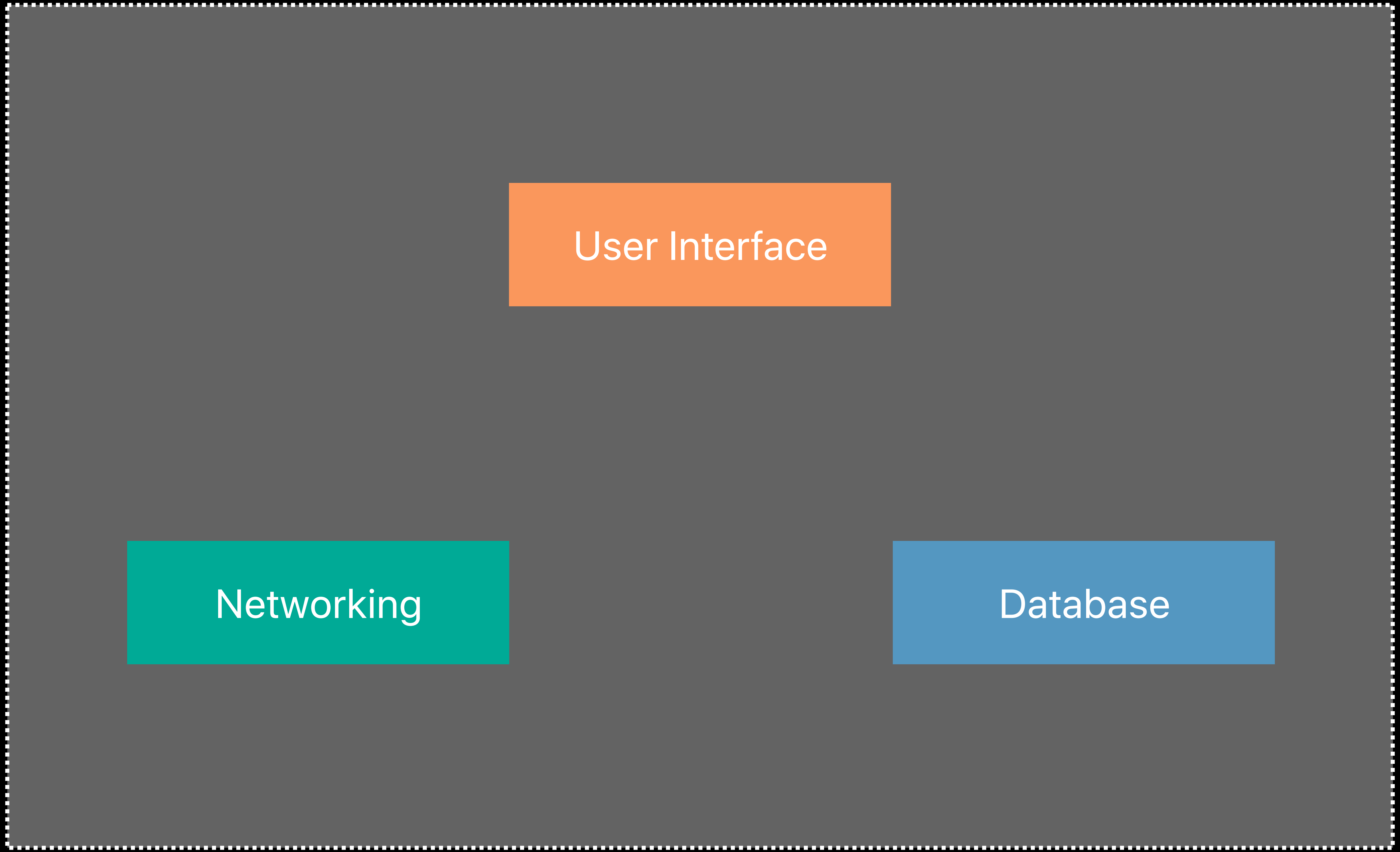


QoS and Target Queue Hierarchy





Granularity of Concurrency


User Interface
Networking
Database

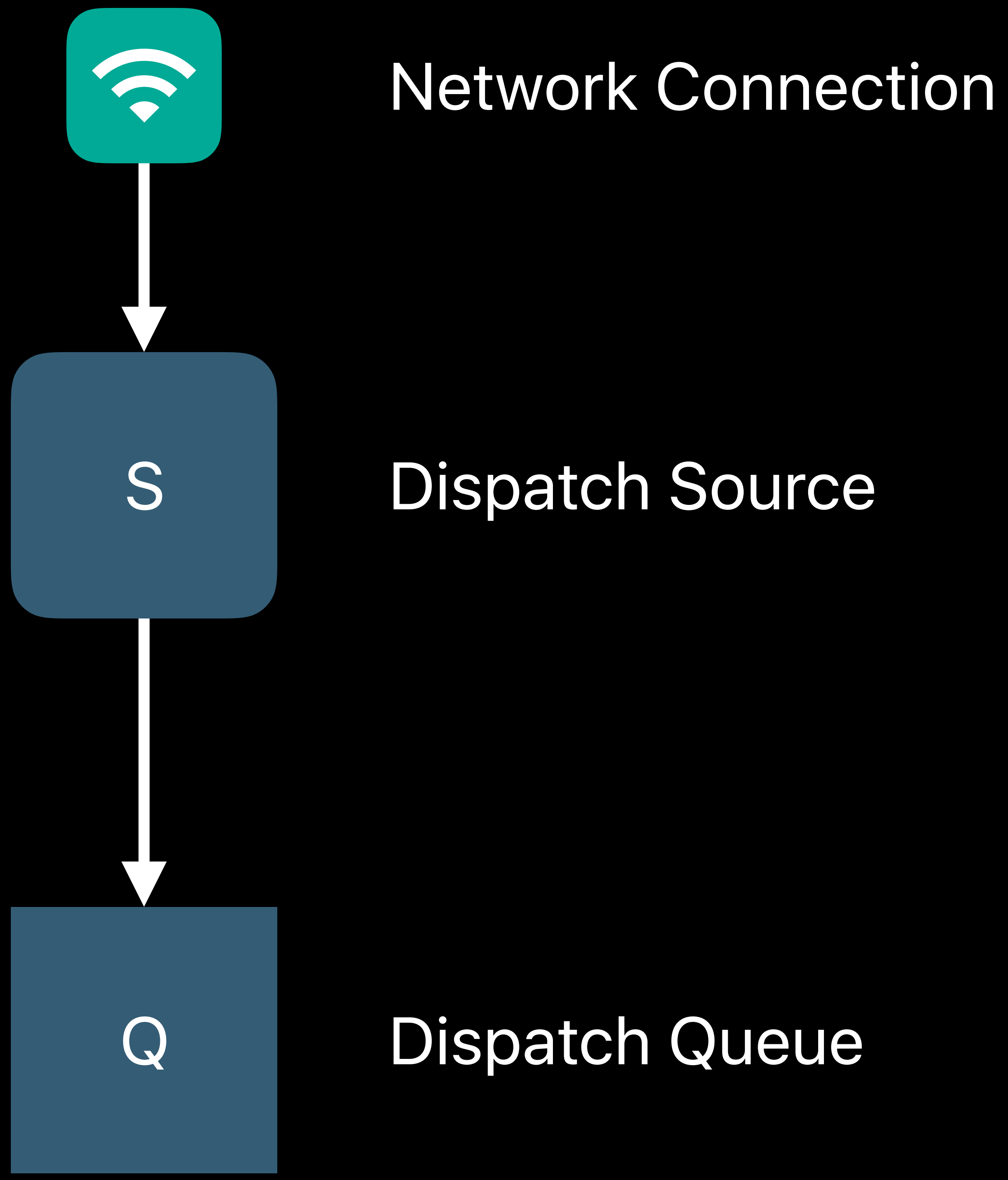
Networking

Event Monitoring Setup

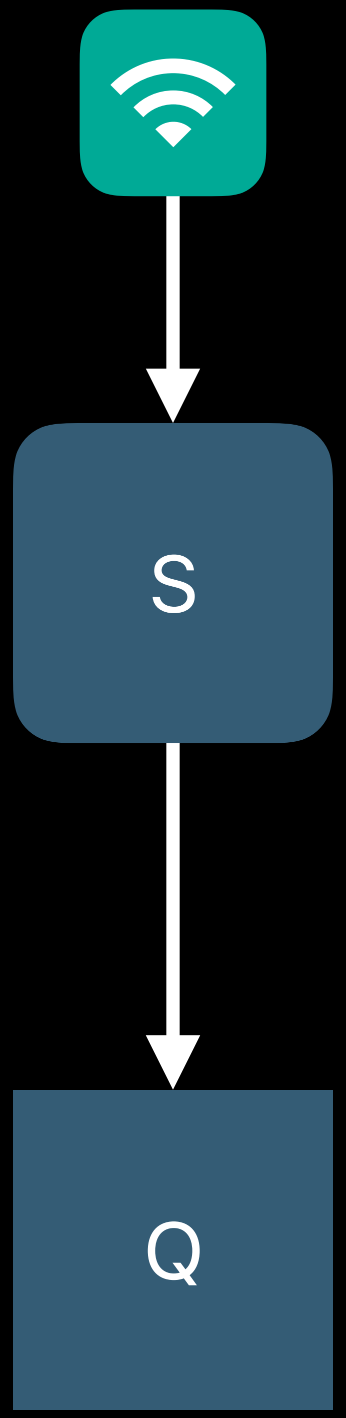
Event Monitoring Setup

Network Connection

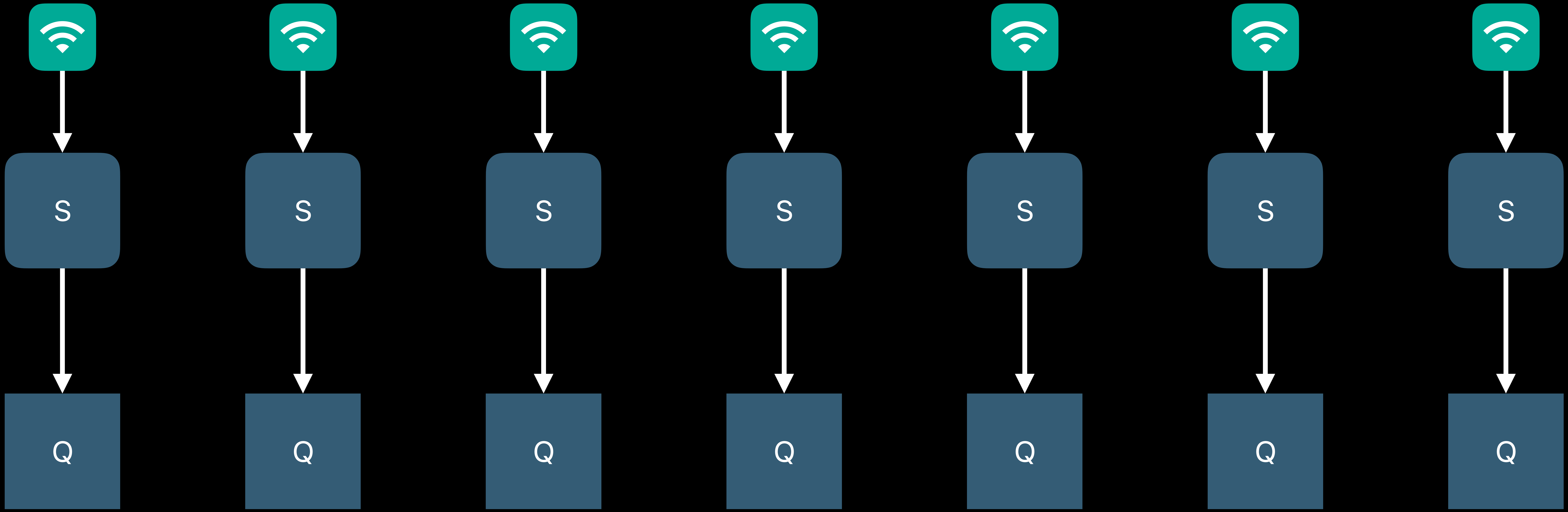
Event Monitoring Setup



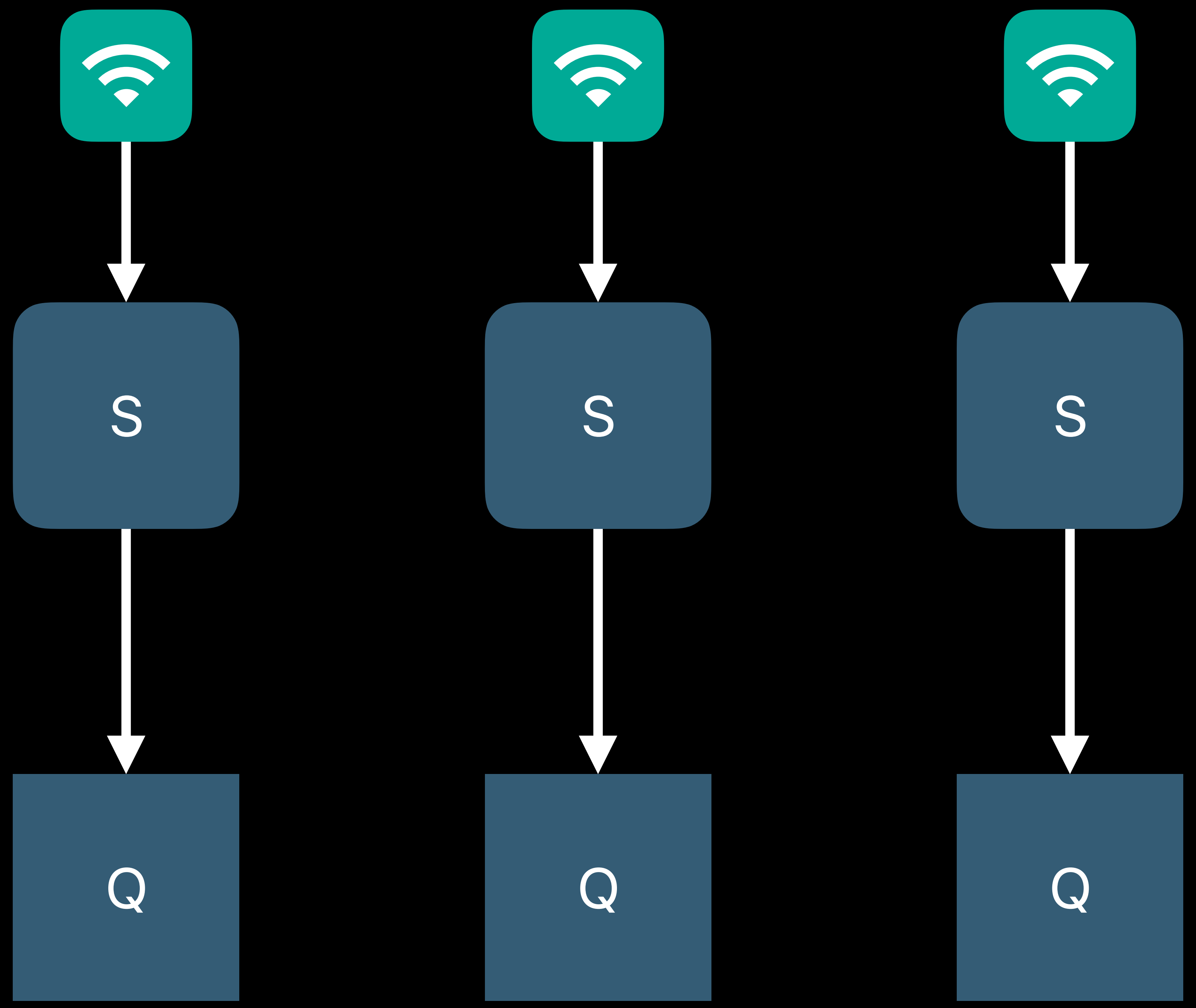
Event Monitoring Setup



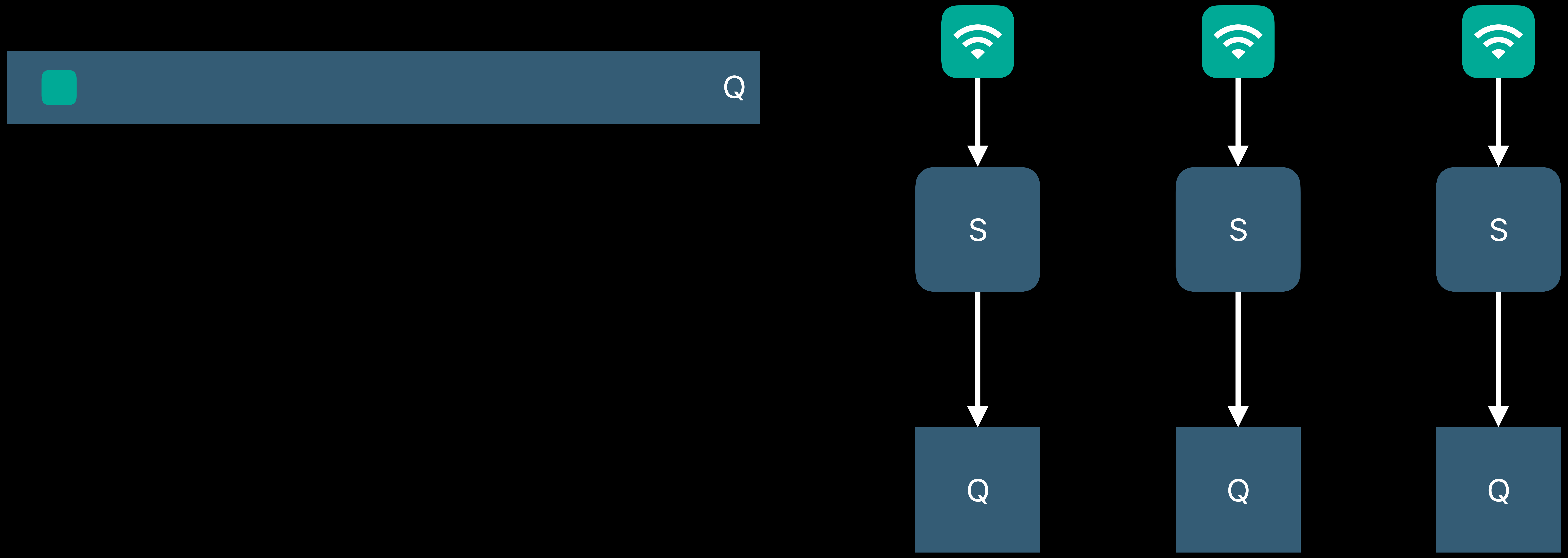
Event Monitoring Setup



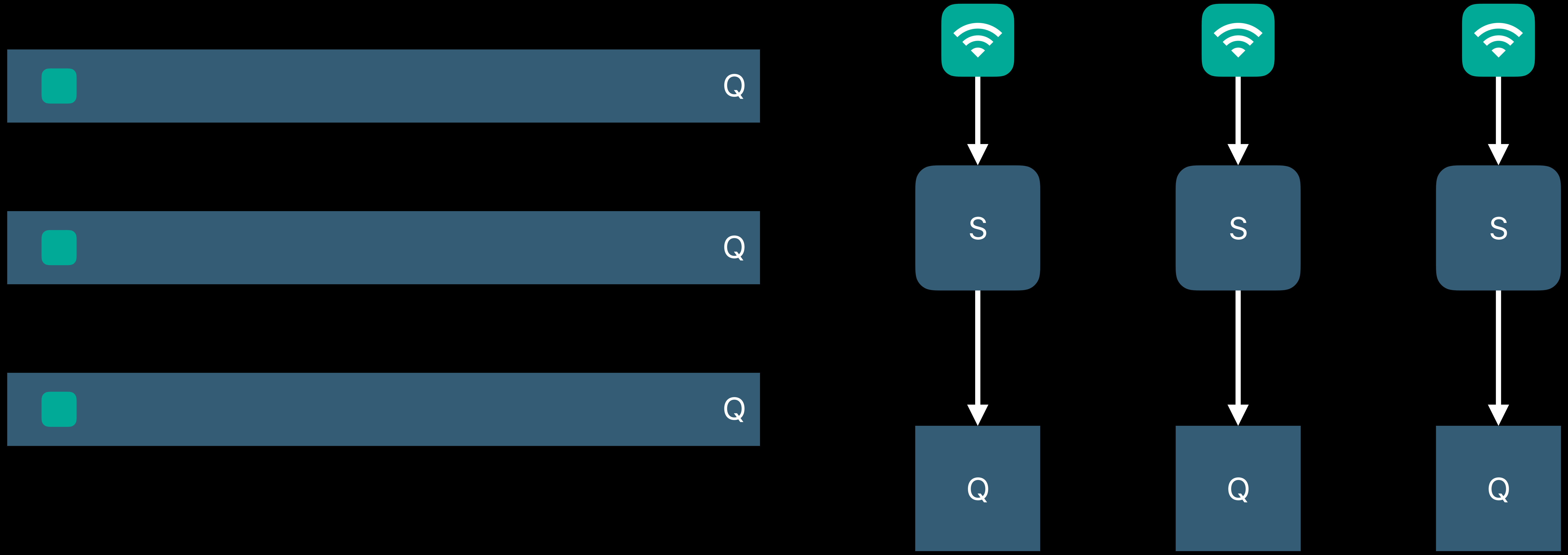
Event Handling on Many Independent Queues



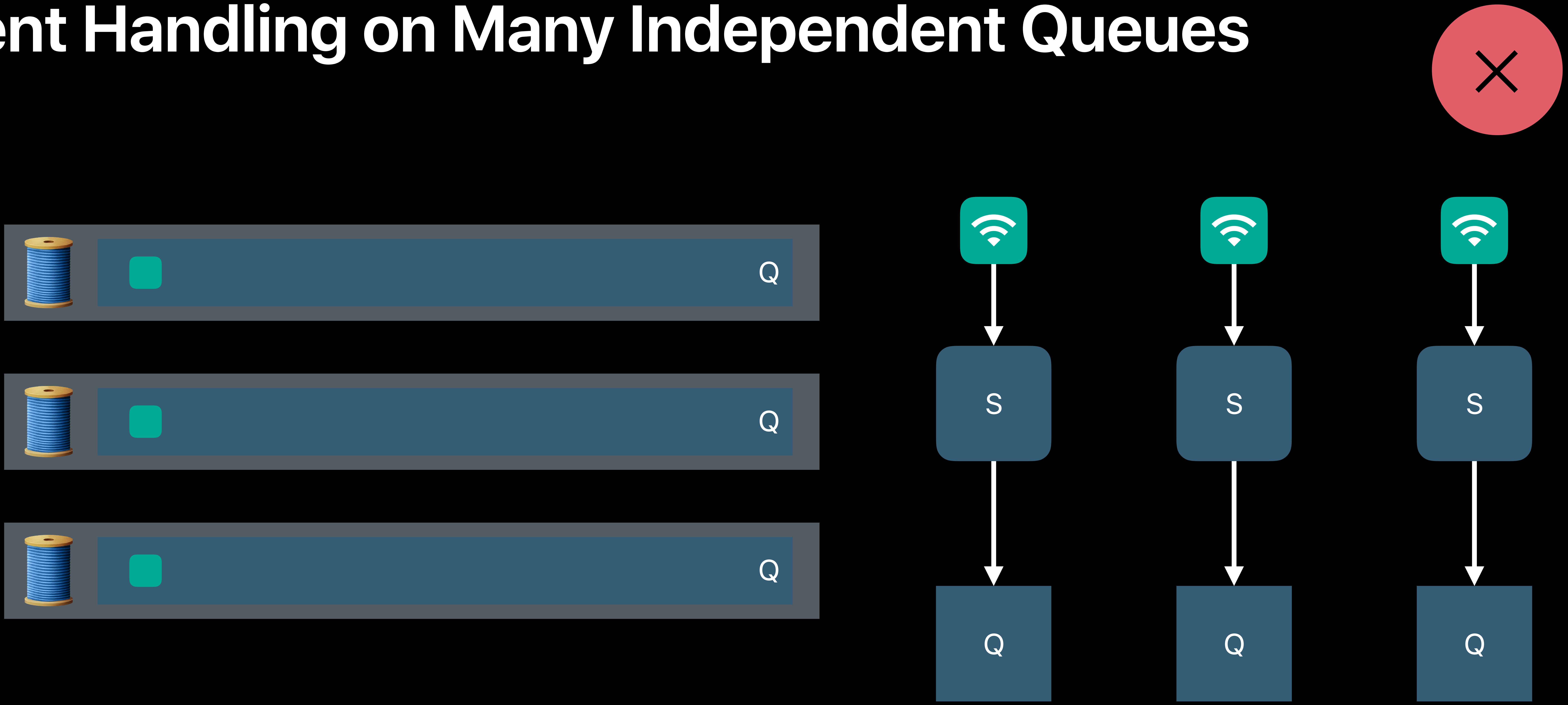
Event Handling on Many Independent Queues



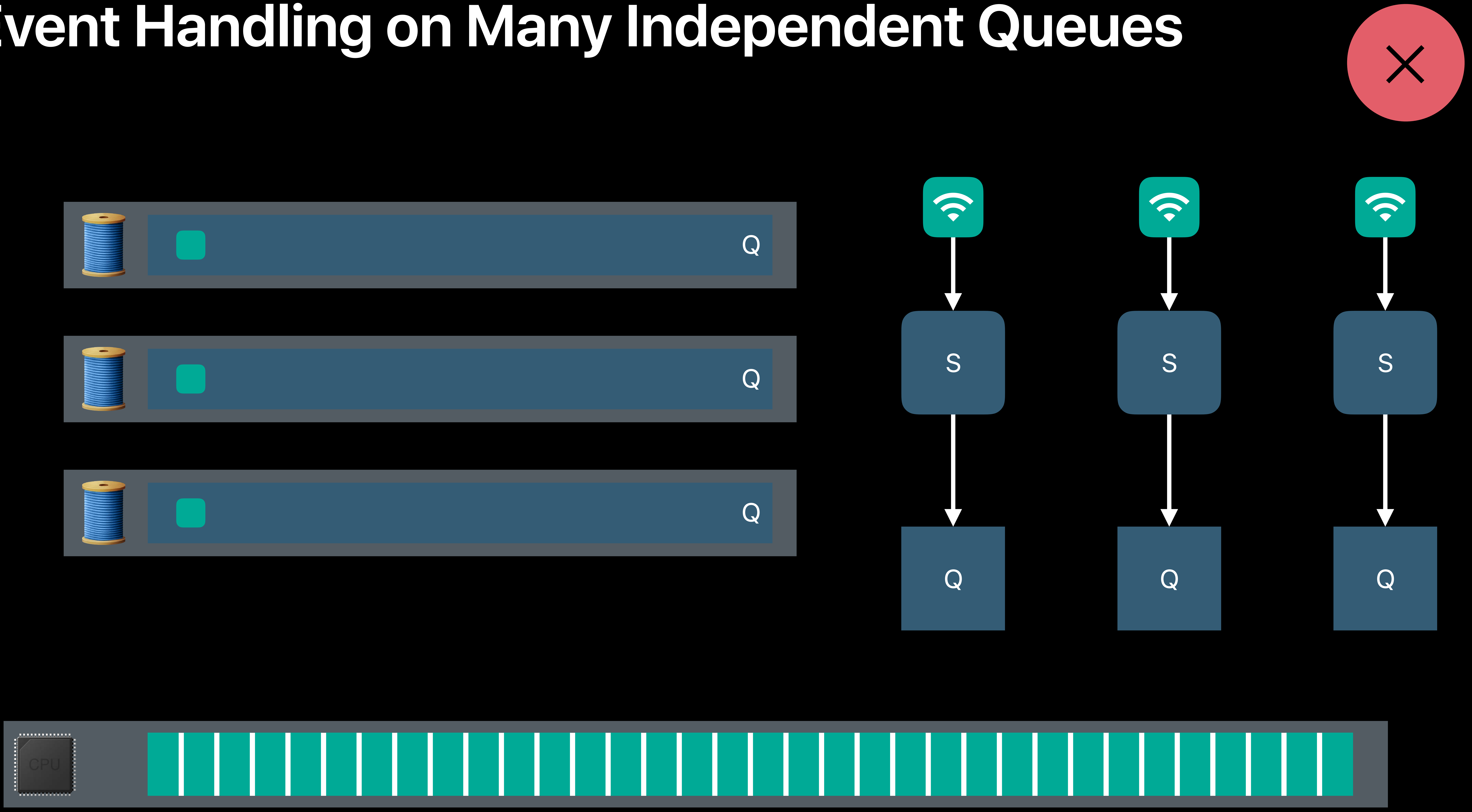
Event Handling on Many Independent Queues



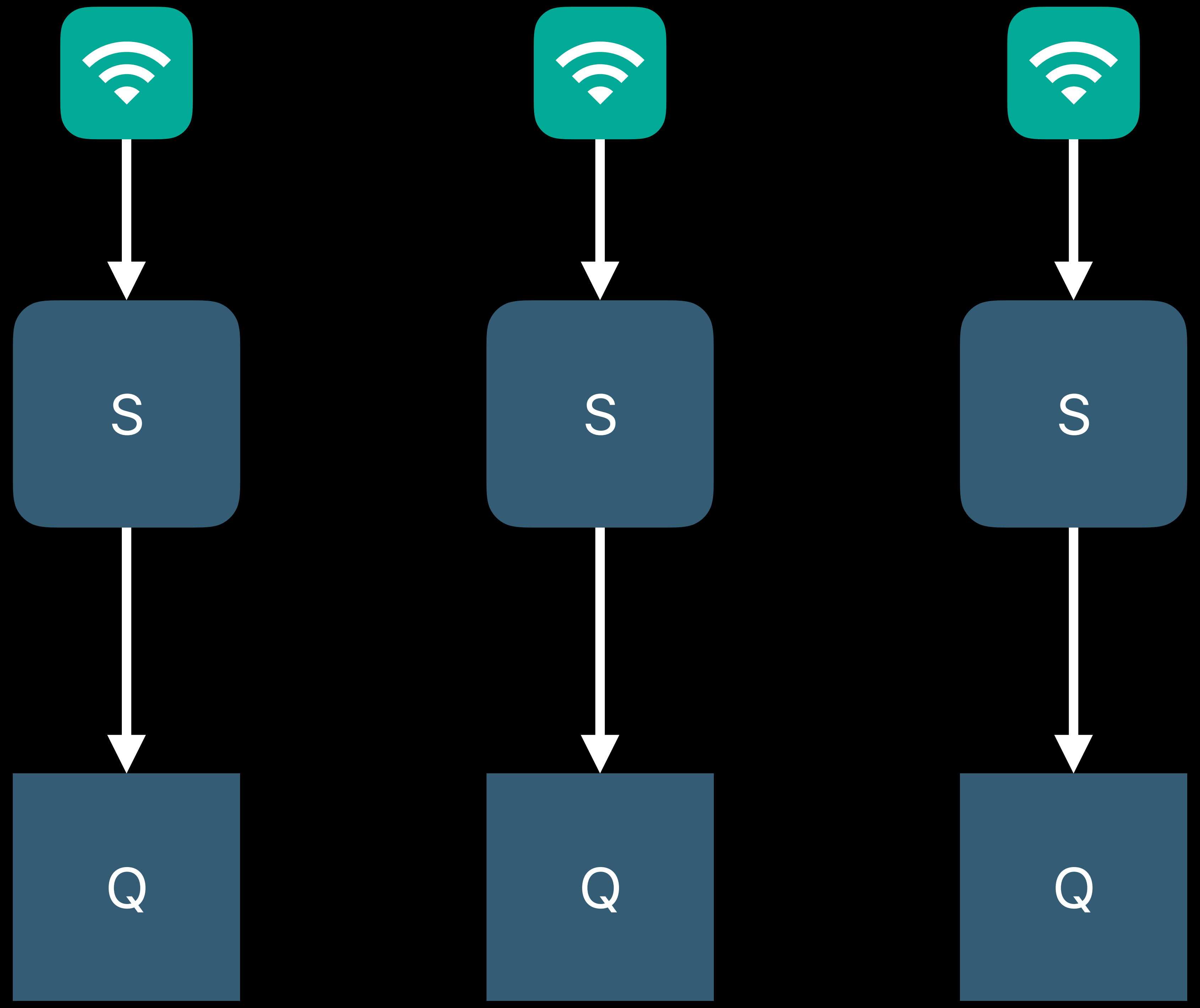
Event Handling on Many Independent Queues



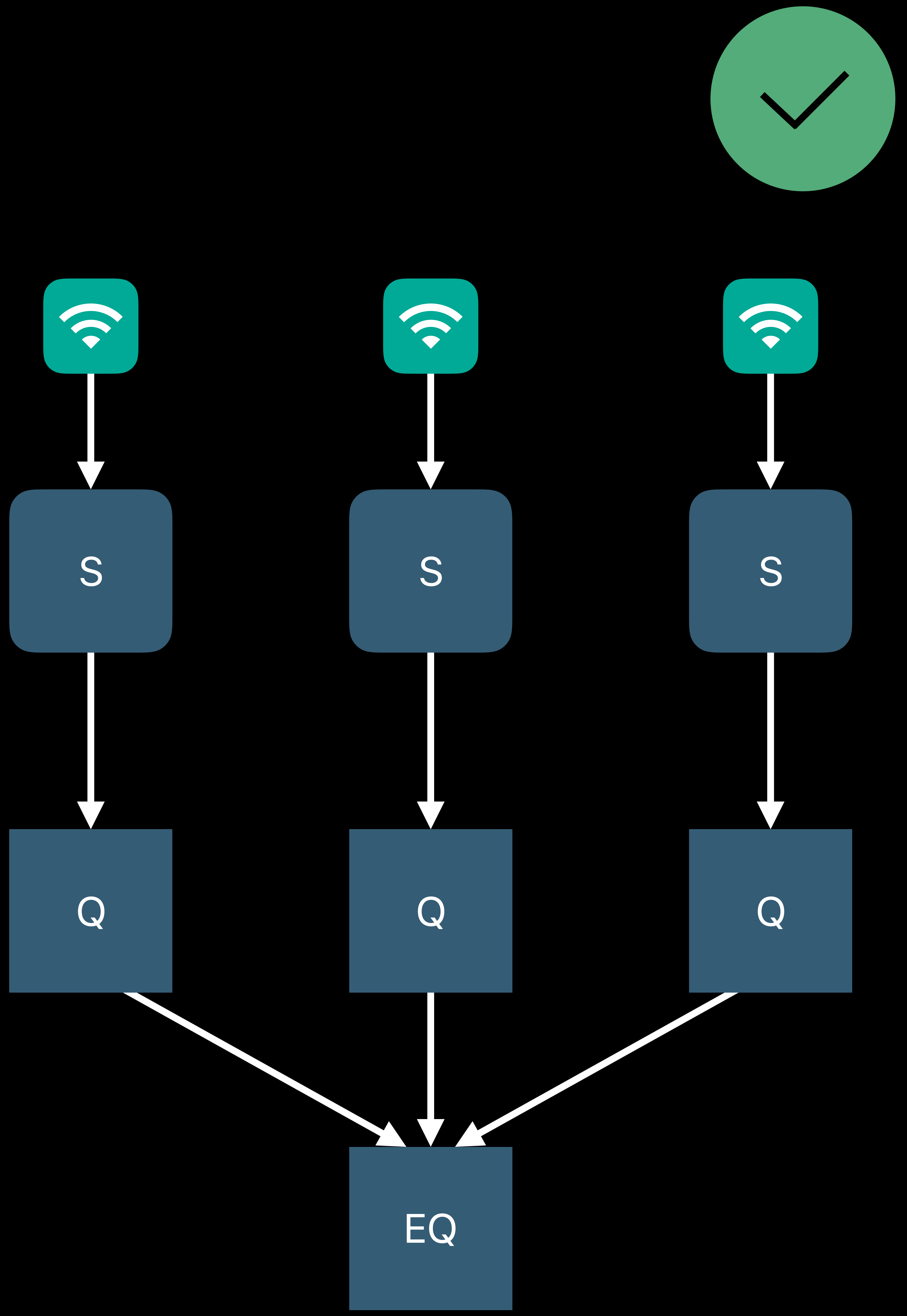
Event Handling on Many Independent Queues



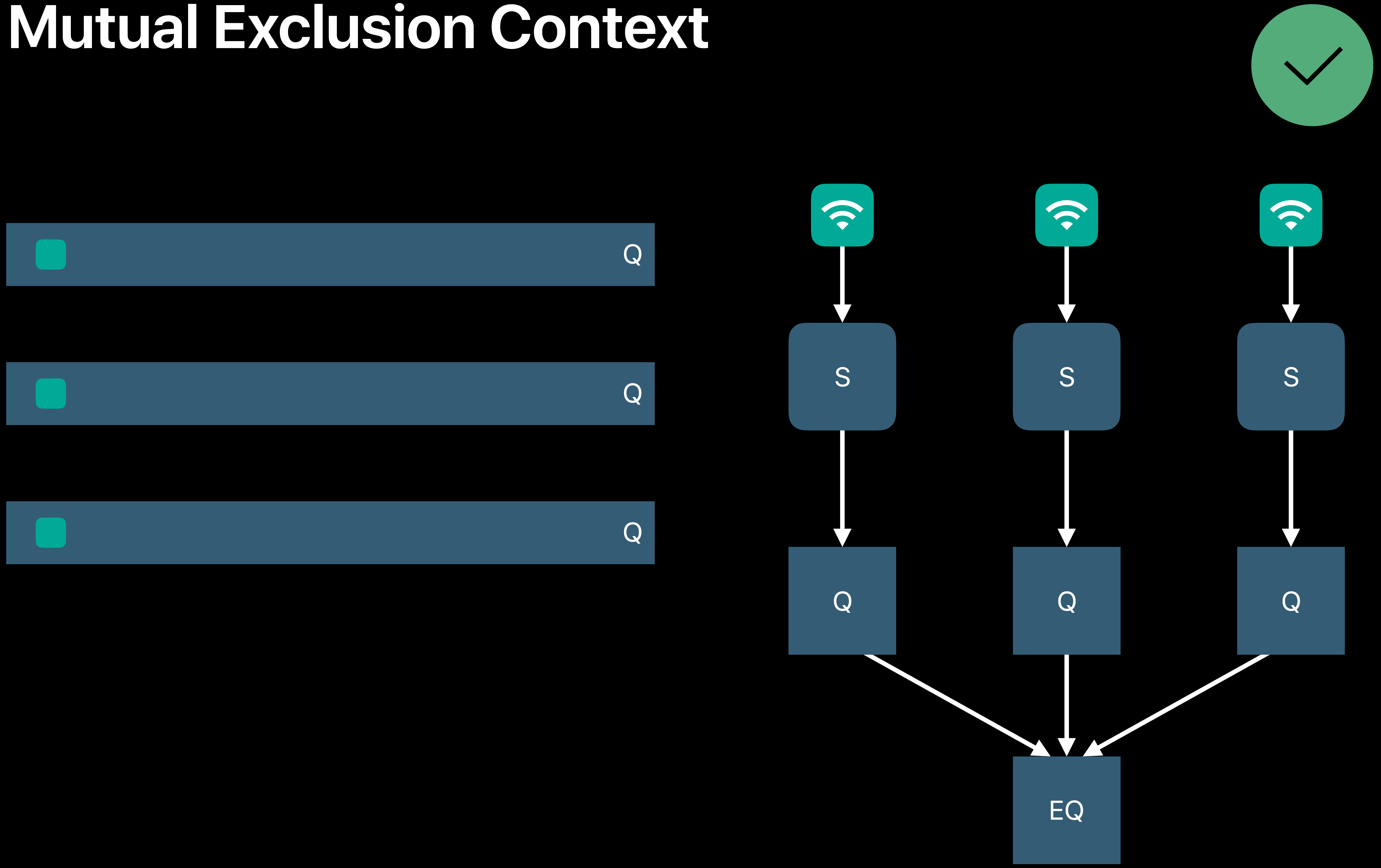
Single Mutual Exclusion Context



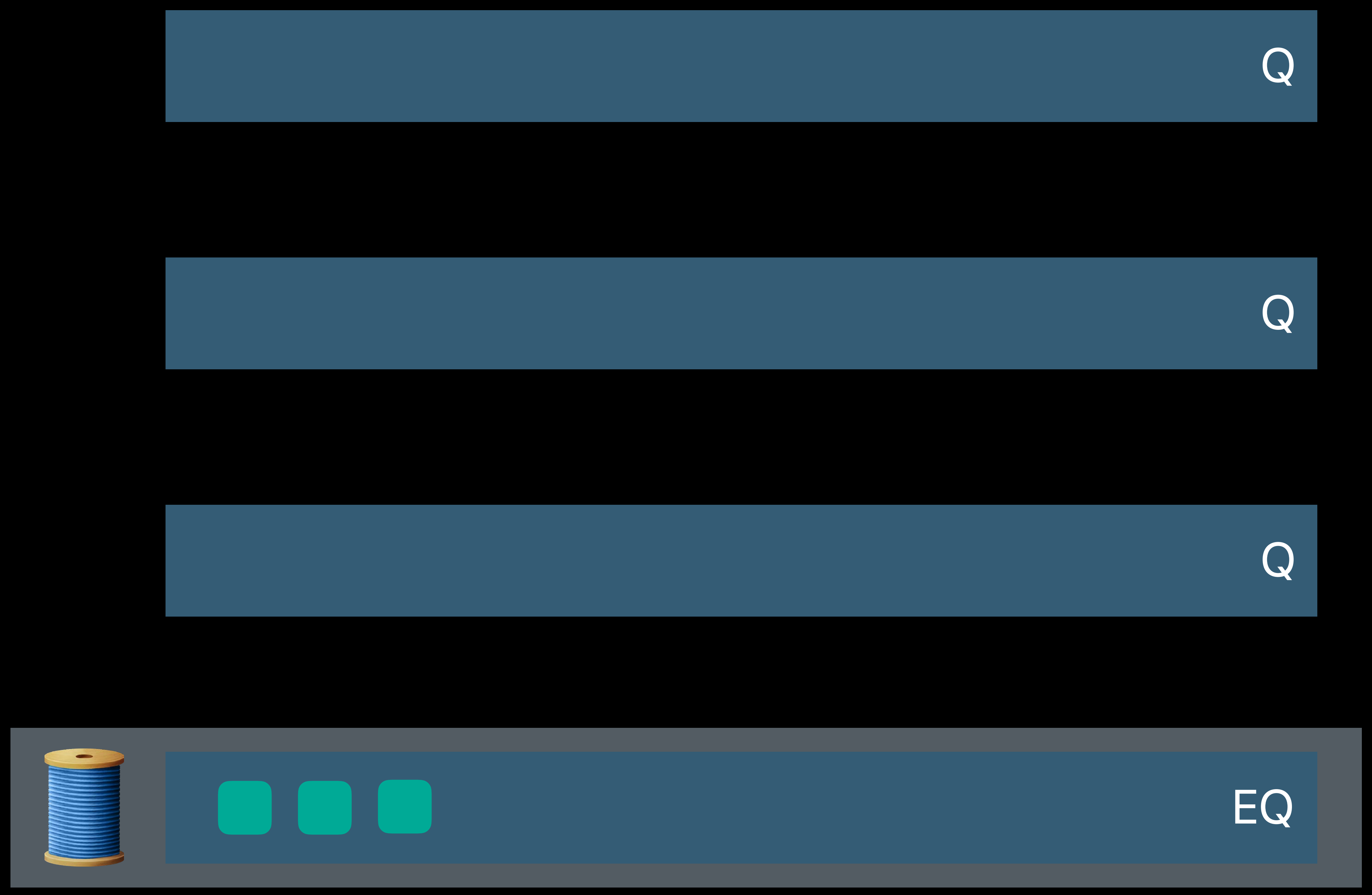
Single Mutual Exclusion Context

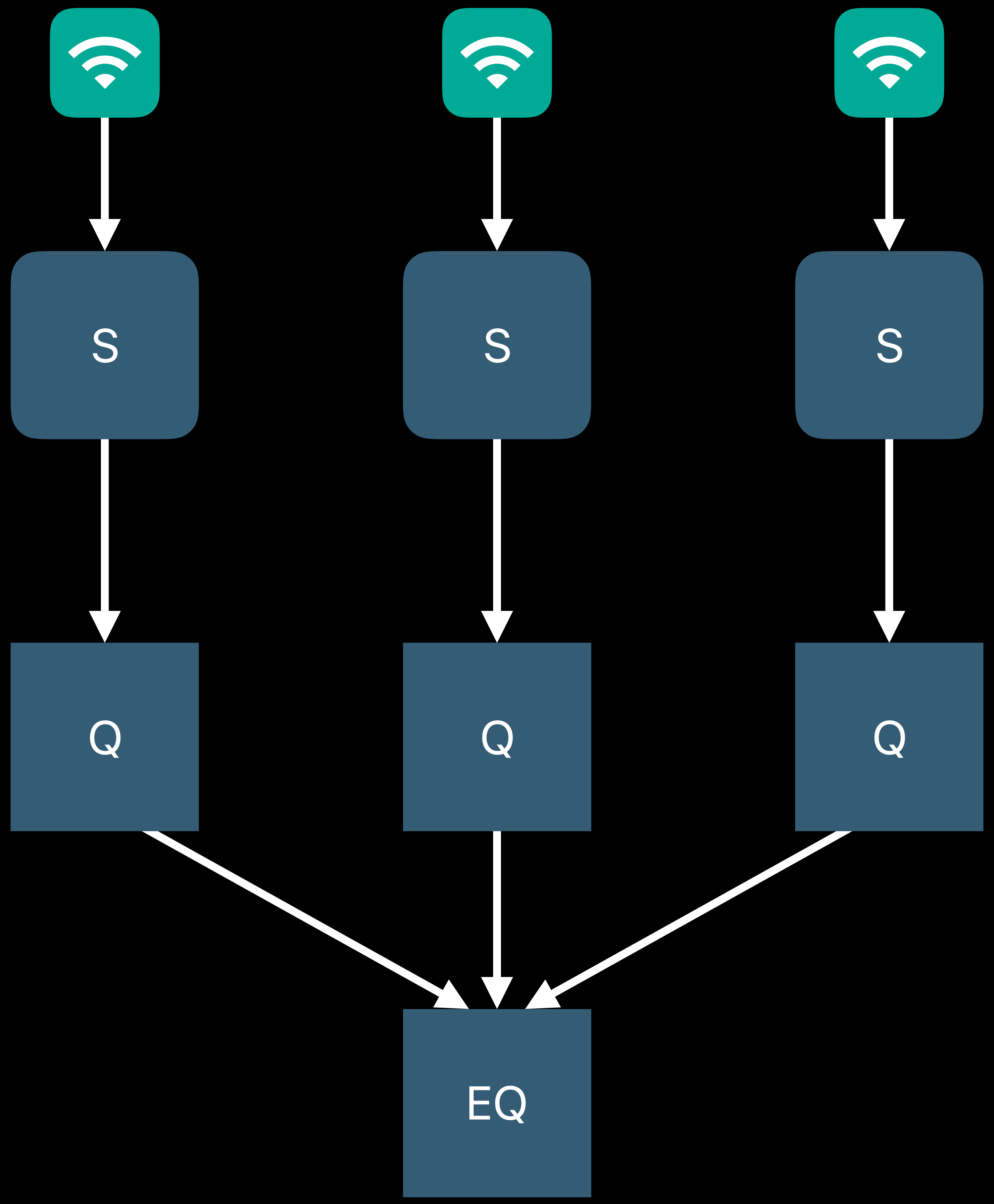


Single Mutual Exclusion Context



Single Mutual Exclusion Context





Too Much of a Good Thing

Repeatedly waiting for exclusive access to contended resources

Repeatedly switching between independent operations

Repeatedly bouncing an operation between threads

Avoid Unbounded Concurrency

Repeatedly switching between independent operations

Avoid Unbounded Concurrency

Repeatedly switching between independent operations

✗

Avoid Unbounded Concurrency

Repeatedly switching between independent operations

Many queues becoming active at once

- Independent per-client sources
- Independent per-object queues

Avoid Unbounded Concurrency

Repeatedly switching between independent operations

✗

Many workitems submitted to global concurrent queue

Avoid Unbounded Concurrency

Repeatedly switching between independent operations

Many workitems submitted to global concurrent queue

- If workitems block, more threads will be created
- May lead to thread explosion

Avoid Unbounded Concurrency

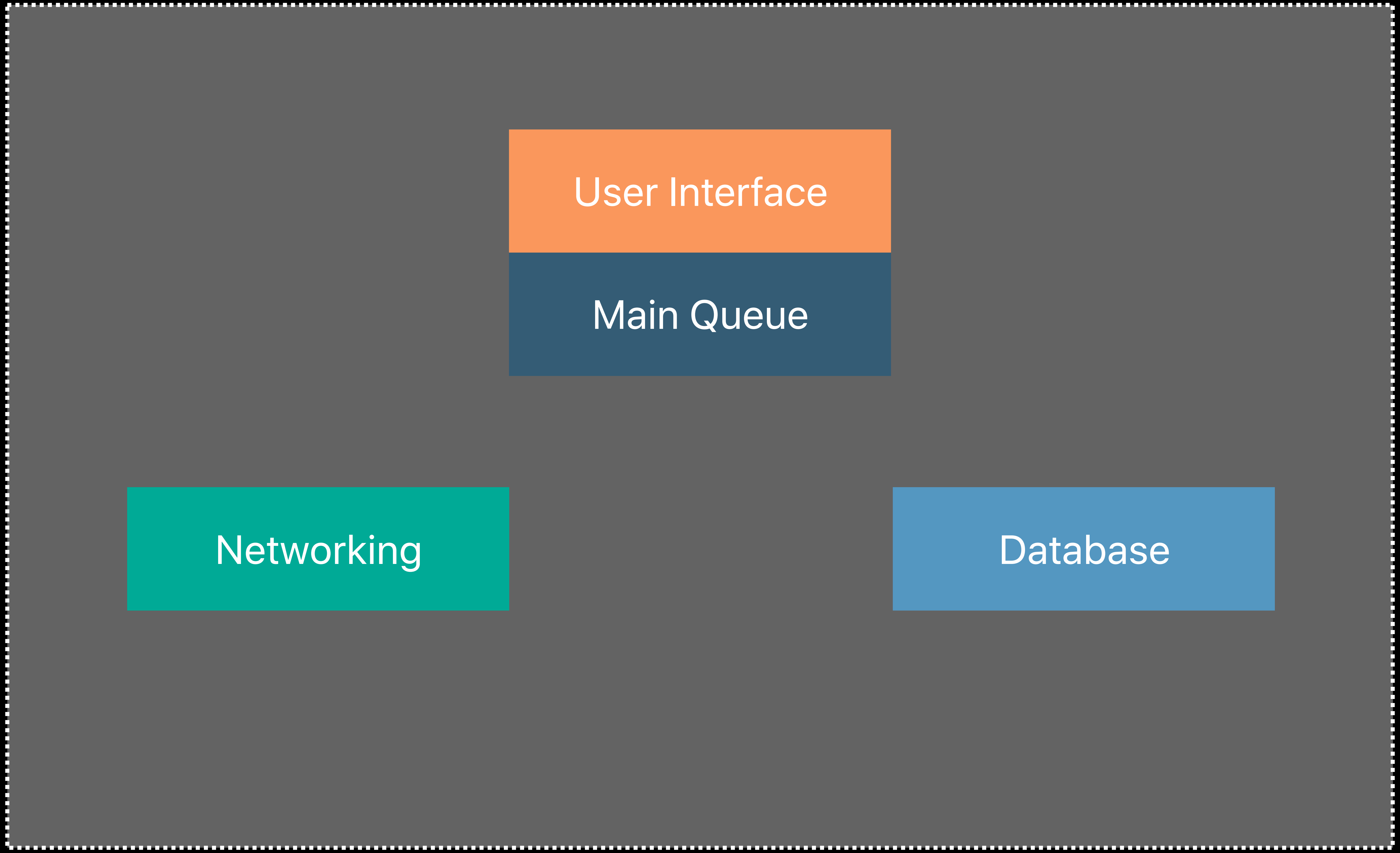
Repeatedly switching between independent operations

Many workitems submitted to global concurrent queue

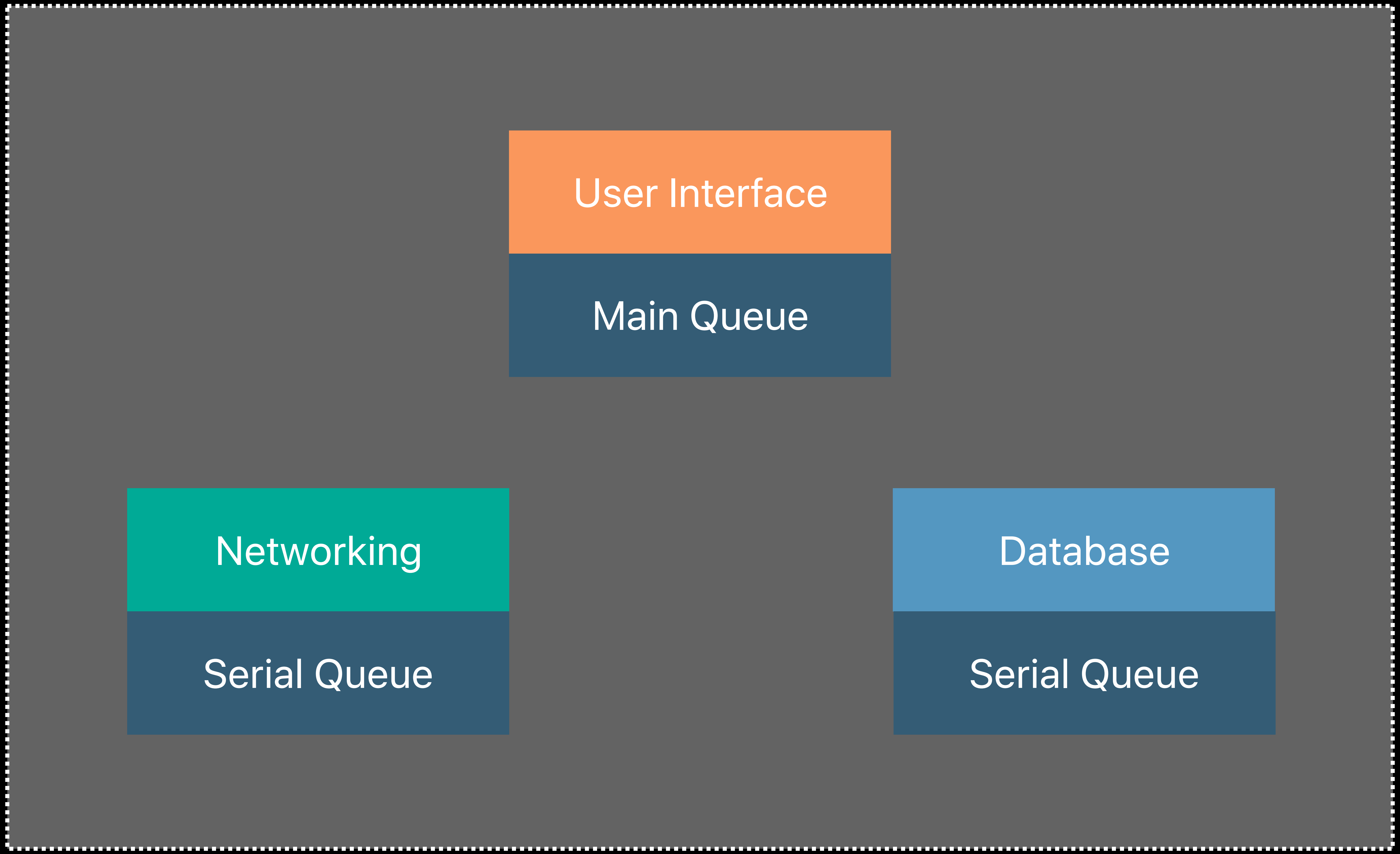
- If workitems block, more threads will be created
- May lead to thread explosion

One Queue per Subsystem

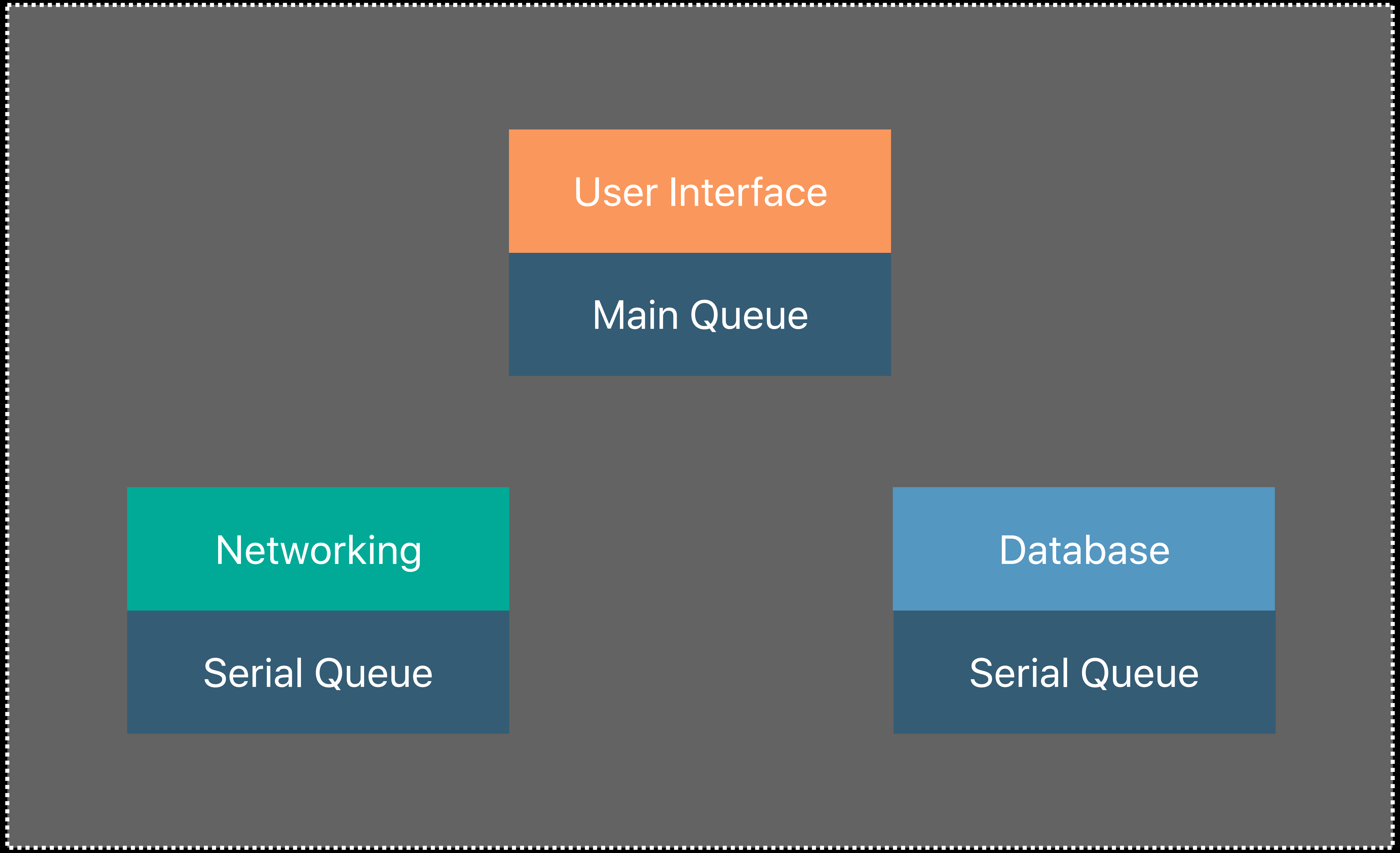
One Queue per Subsystem



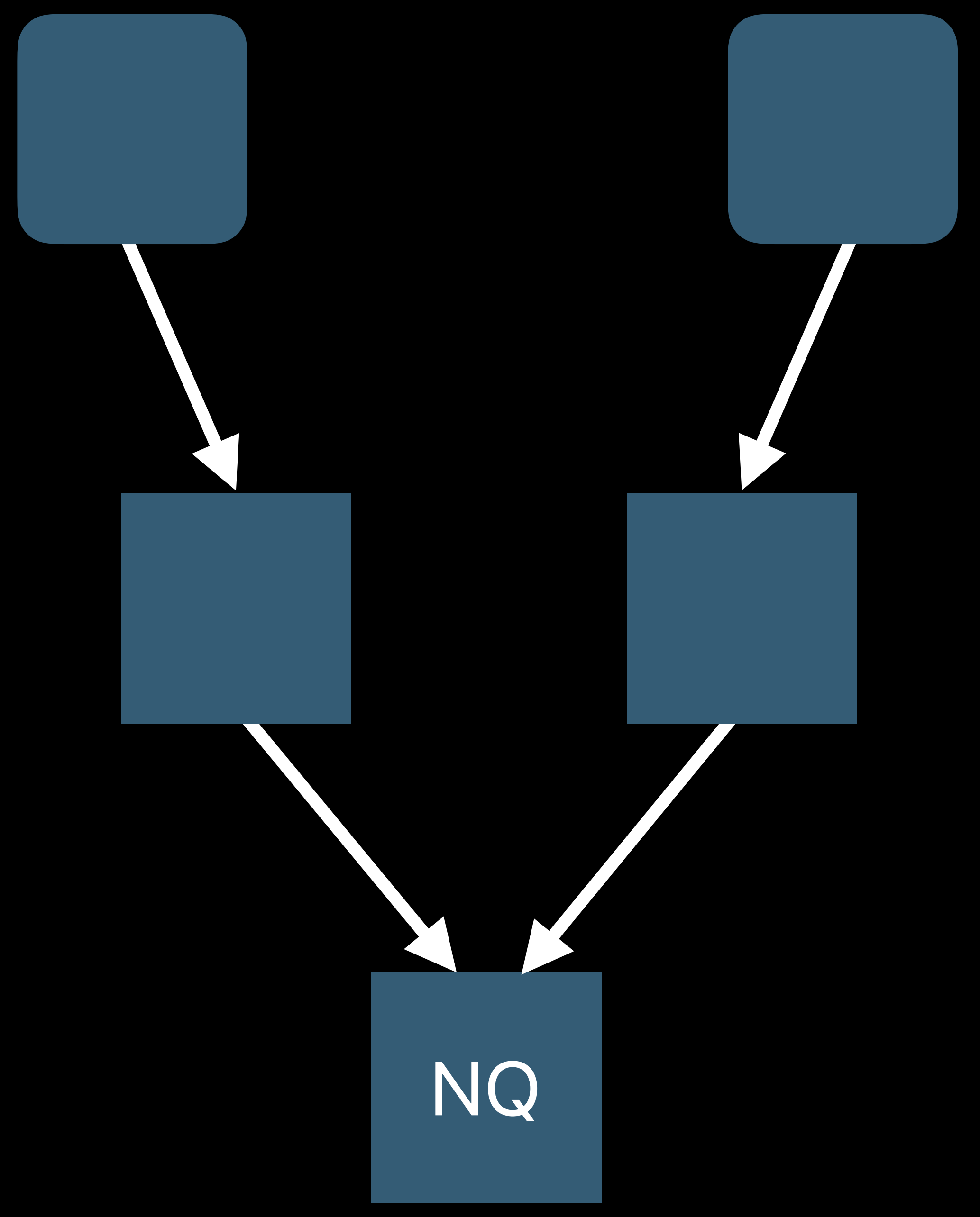
One Queue per Subsystem

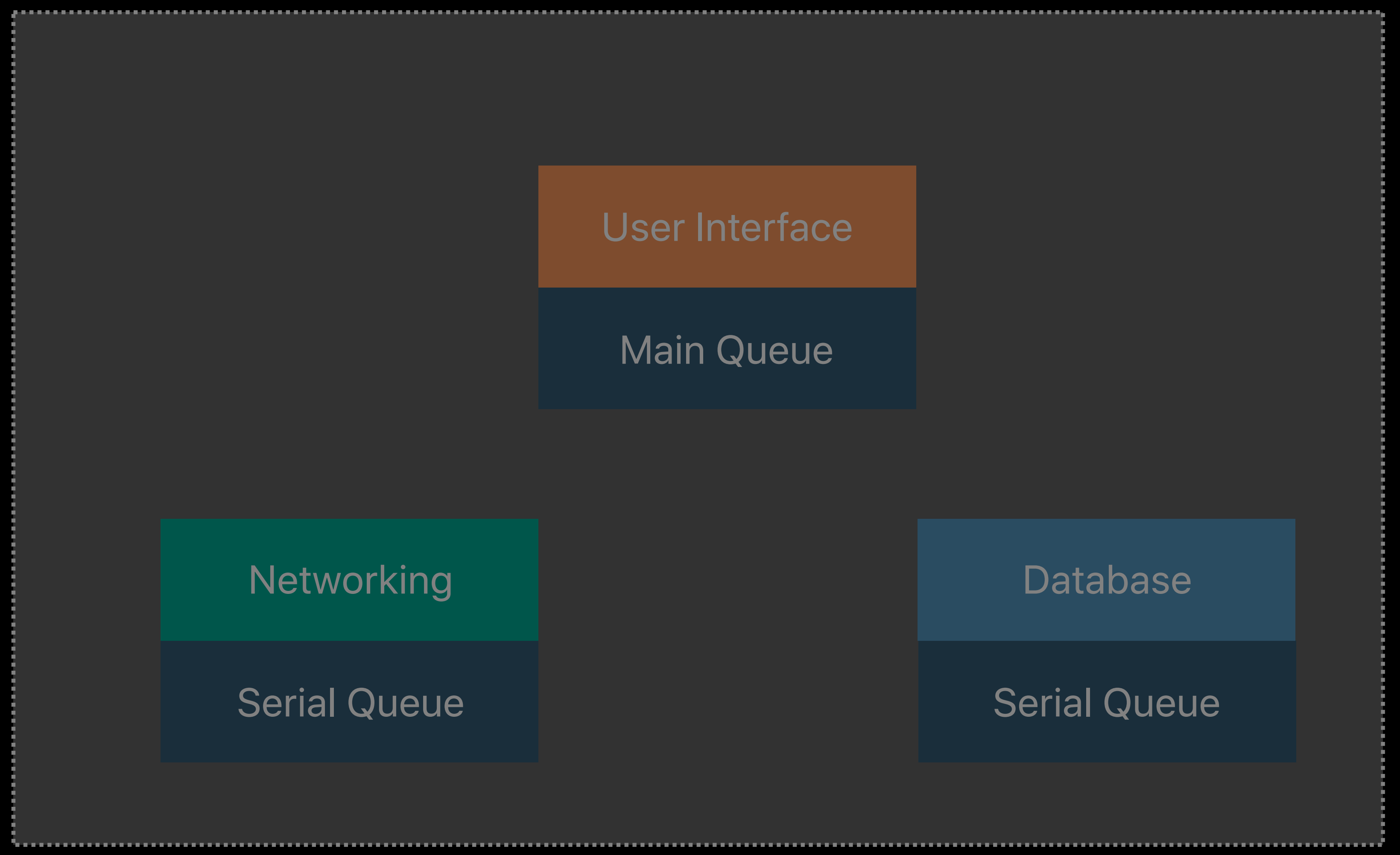


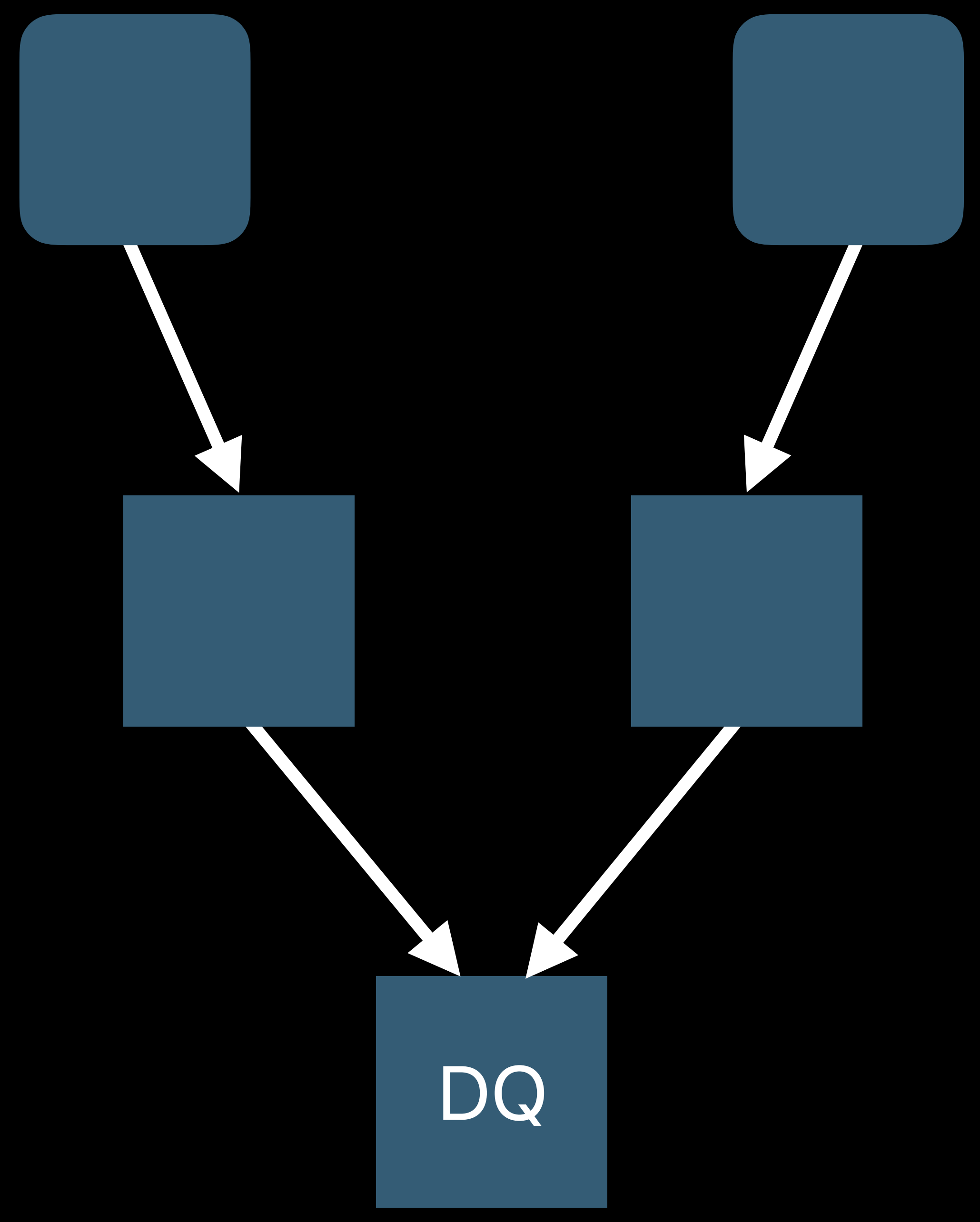
One Queue Hierarchy per Subsystem



One Queue Hierarchy per Subsystem







Good Granularity of Concurrency

Fixed number of serial queue hierarchies

Good Granularity of Concurrency ✓

Fixed number of serial queue hierarchies

Good Granularity of Concurrency

✓

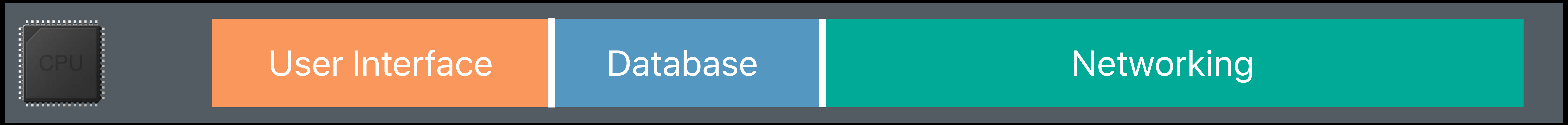
Fixed number of serial queue hierarchies

Coarse workitem granularity between hierarchies

Good Granularity of Concurrency

Fixed number of serial queue hierarchies

Coarse workitem granularity between hierarchies

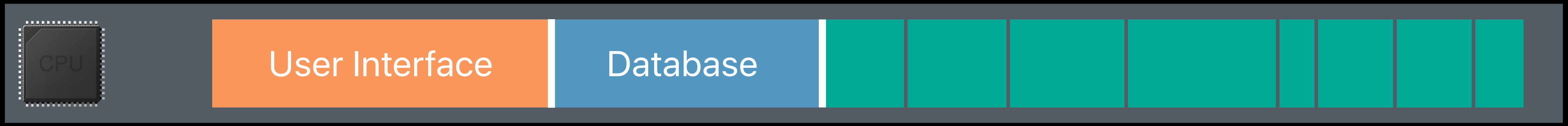


Good Granularity of Concurrency

Fixed number of serial queue hierarchies

Coarse workitem granularity between hierarchies

Finer workitem granularity inside a hierarchy



Using GCD for Concurrency

Organize queues and sources into serial queue hierarchies

Use a fixed number of serial queue hierarchies

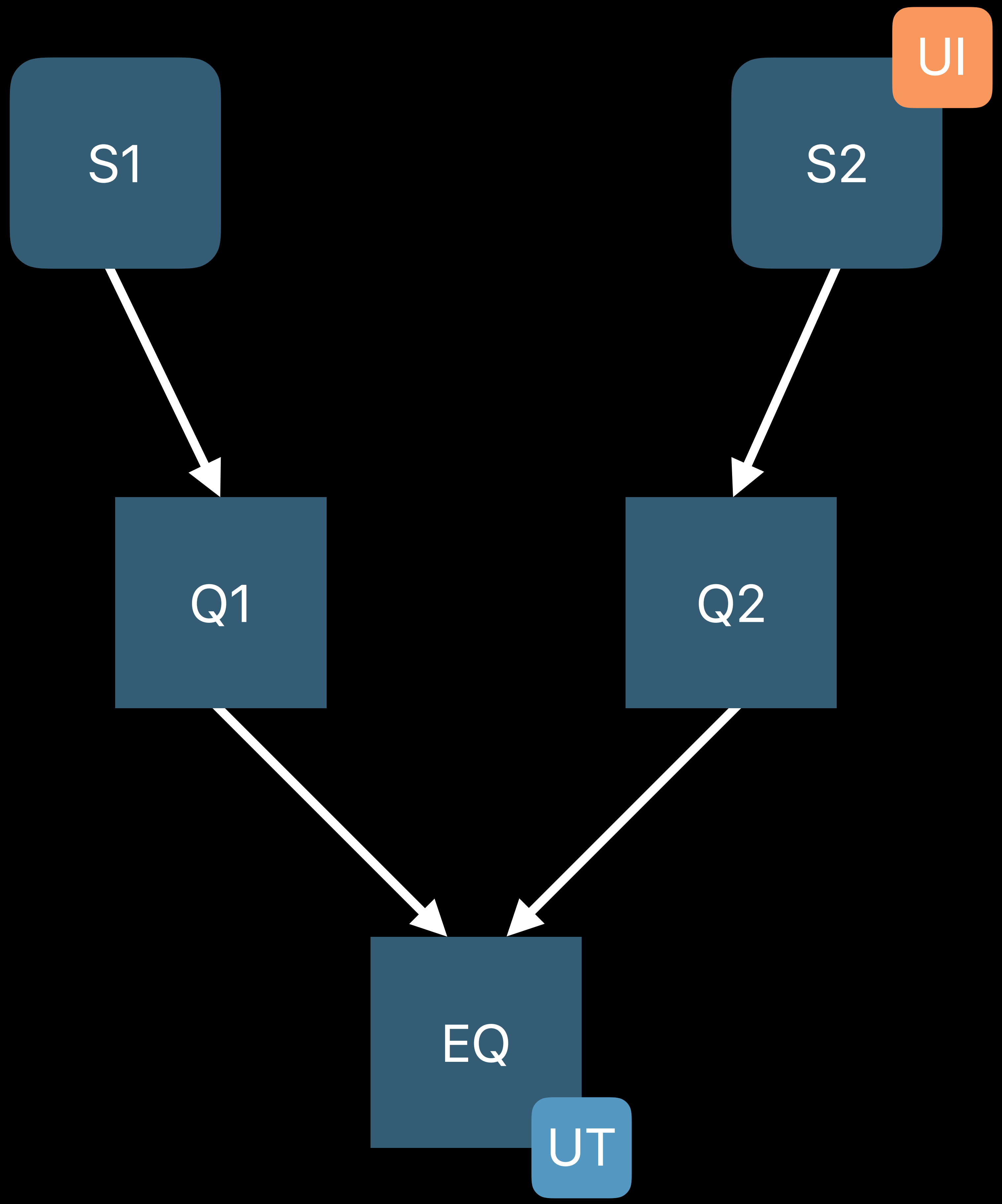
Size your workitems appropriately

Introducing Unified Queue Identity

Pierre Habouzit, Core Darwin

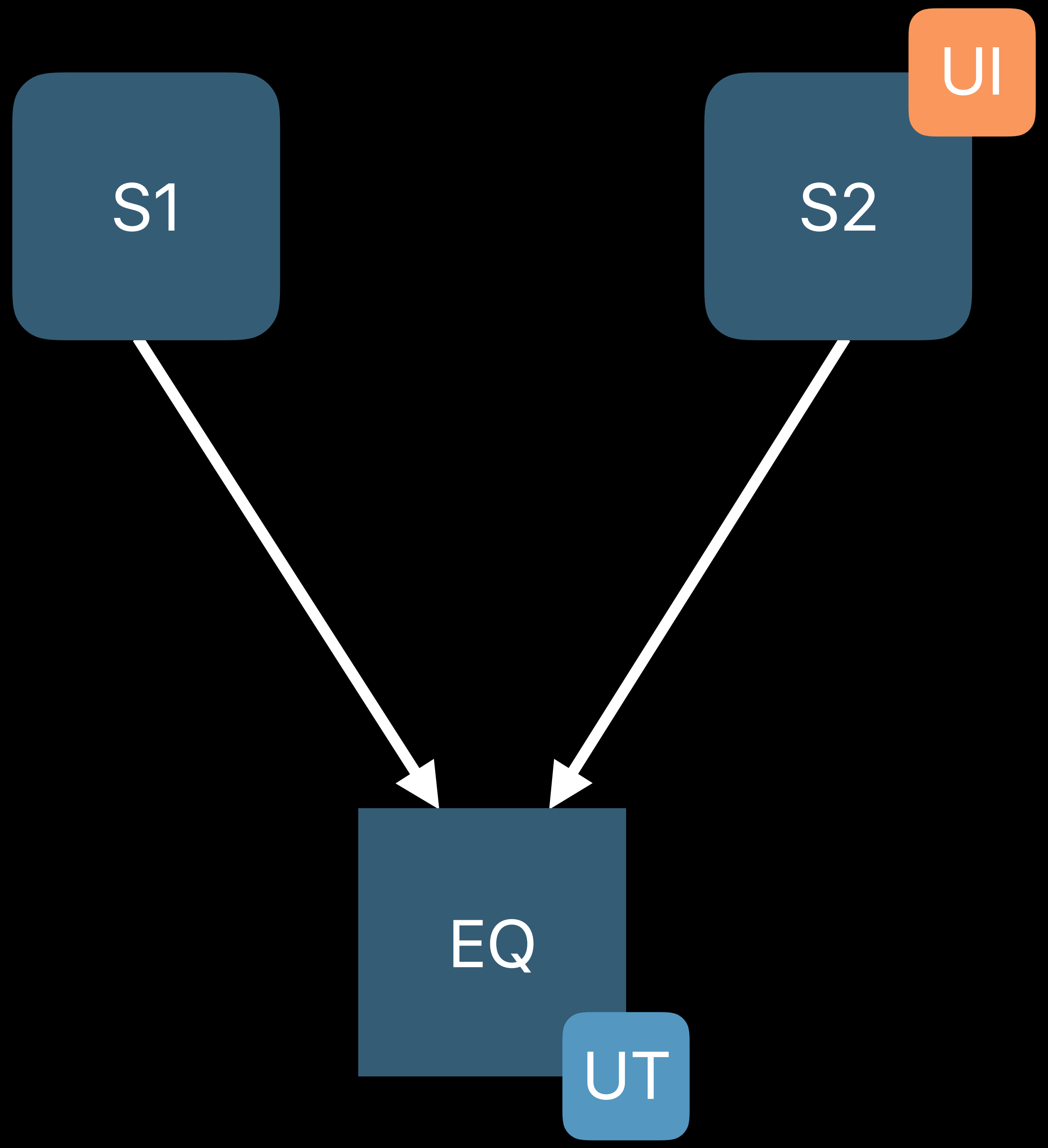
Mutual Exclusion Context

Deep dive

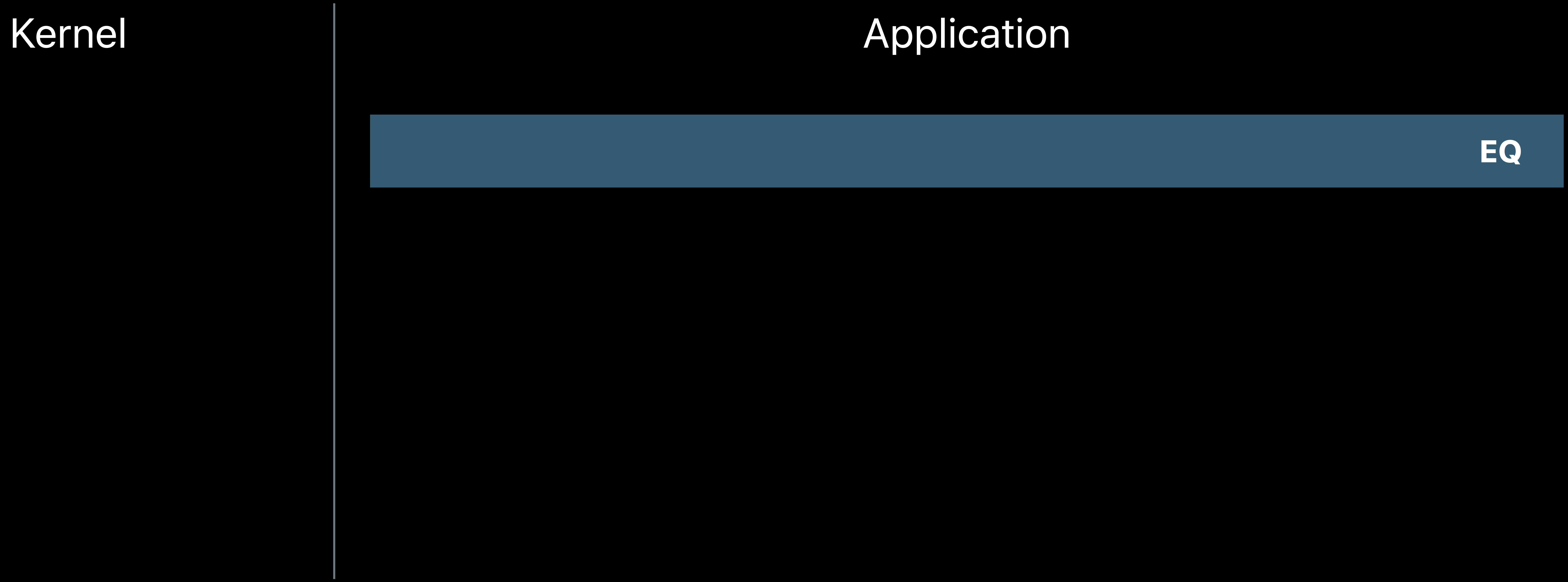


Mutual Exclusion Context

Deep dive



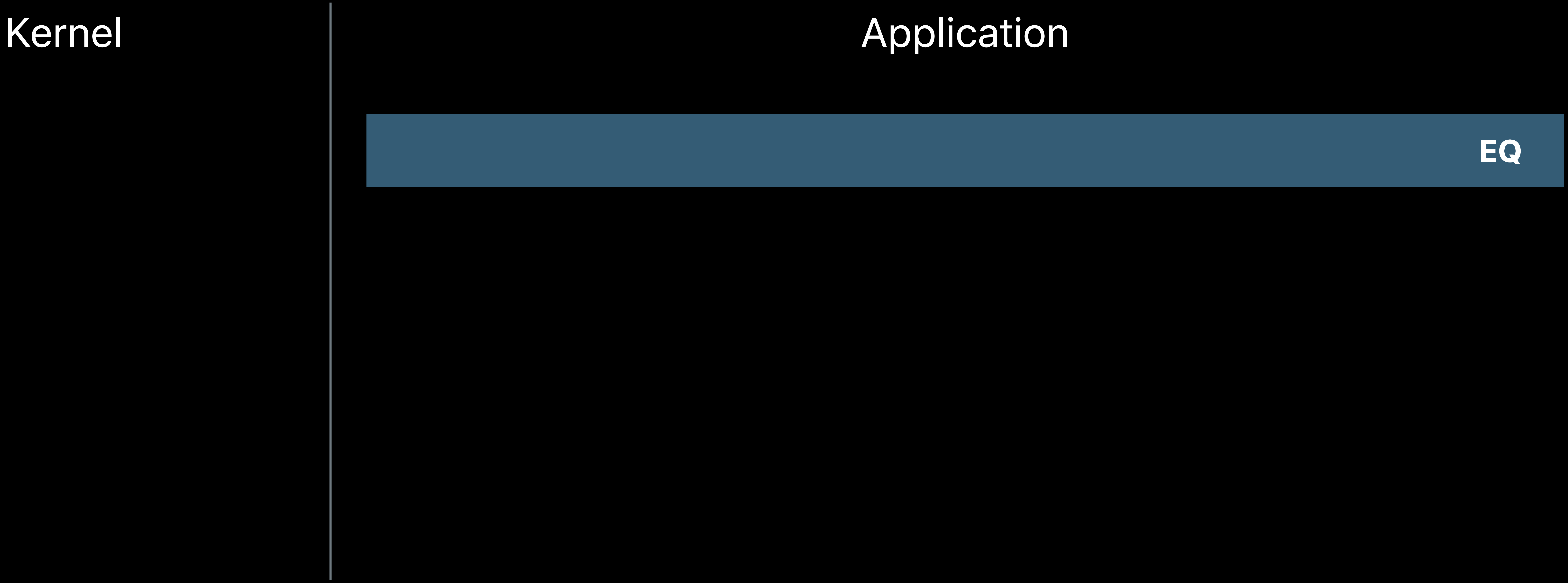
Unified Queue Identity



```
let EQ = DispatchQueue(label: "com.example.exclusion-context")
```

Unified Queue Identity

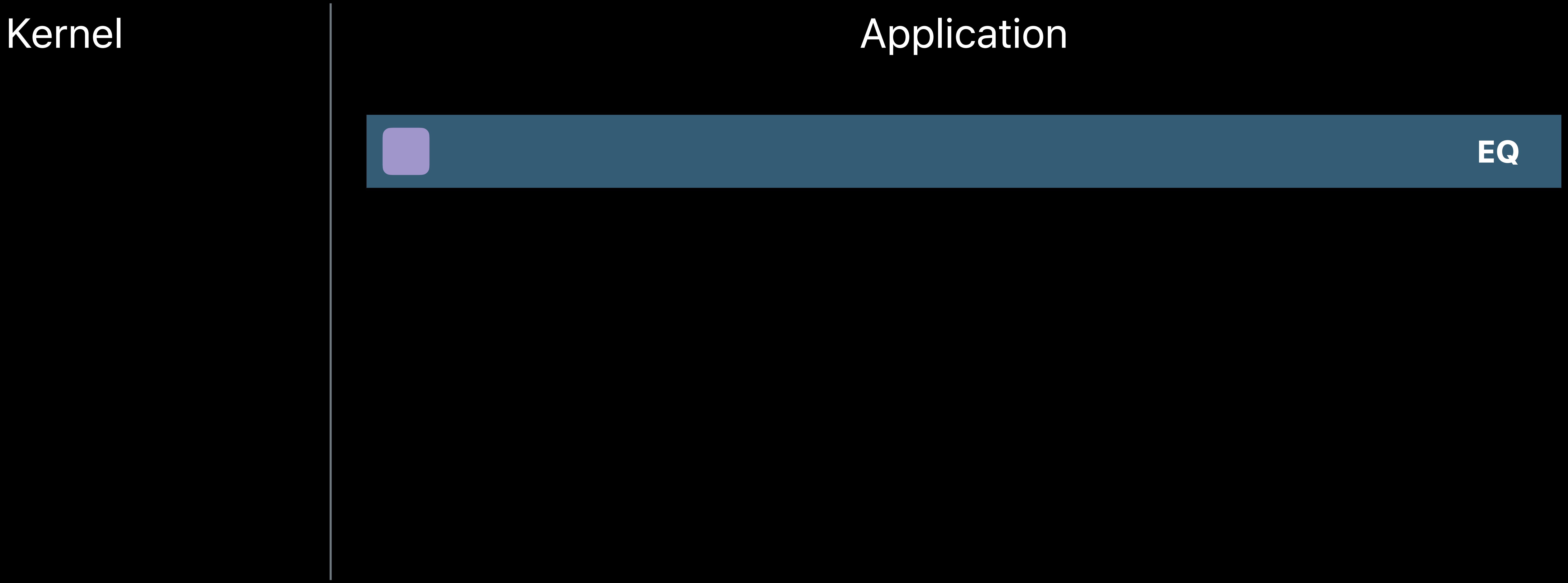
Asynchronous workitems



```
EQ.async { … }
```

Unified Queue Identity

Asynchronous workitems

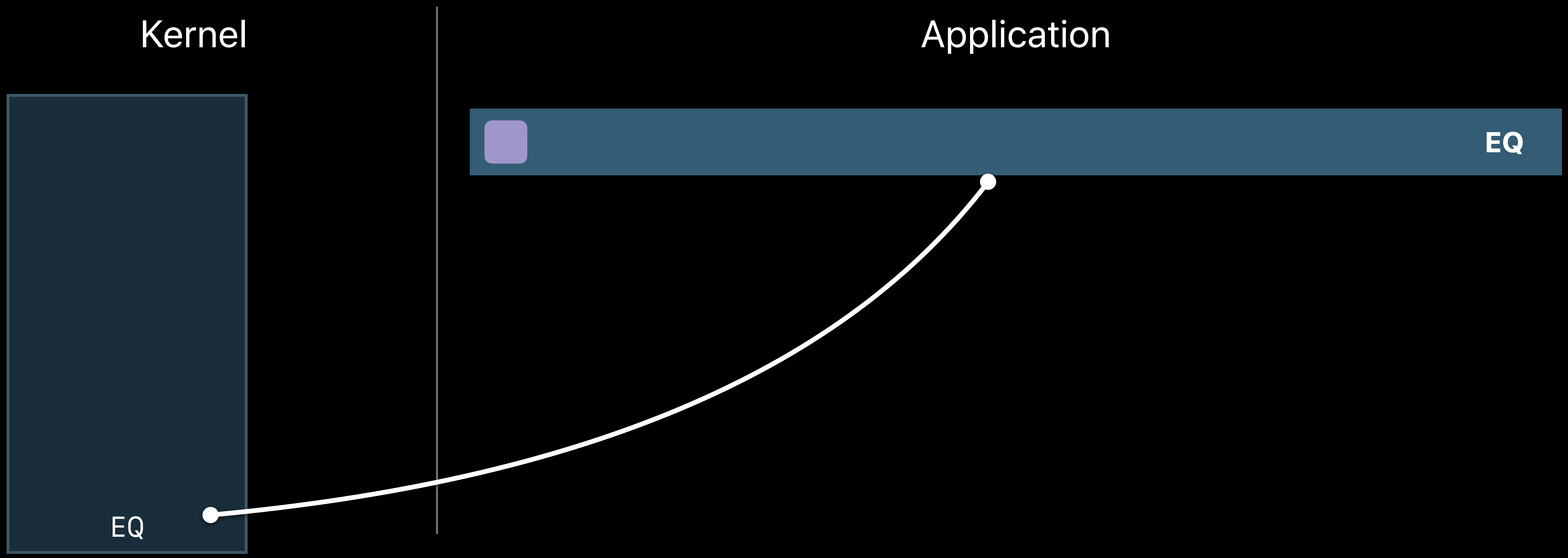


```
EQ.async { … }
```

Unified Queue Identity

Asynchronous workitems

NEW

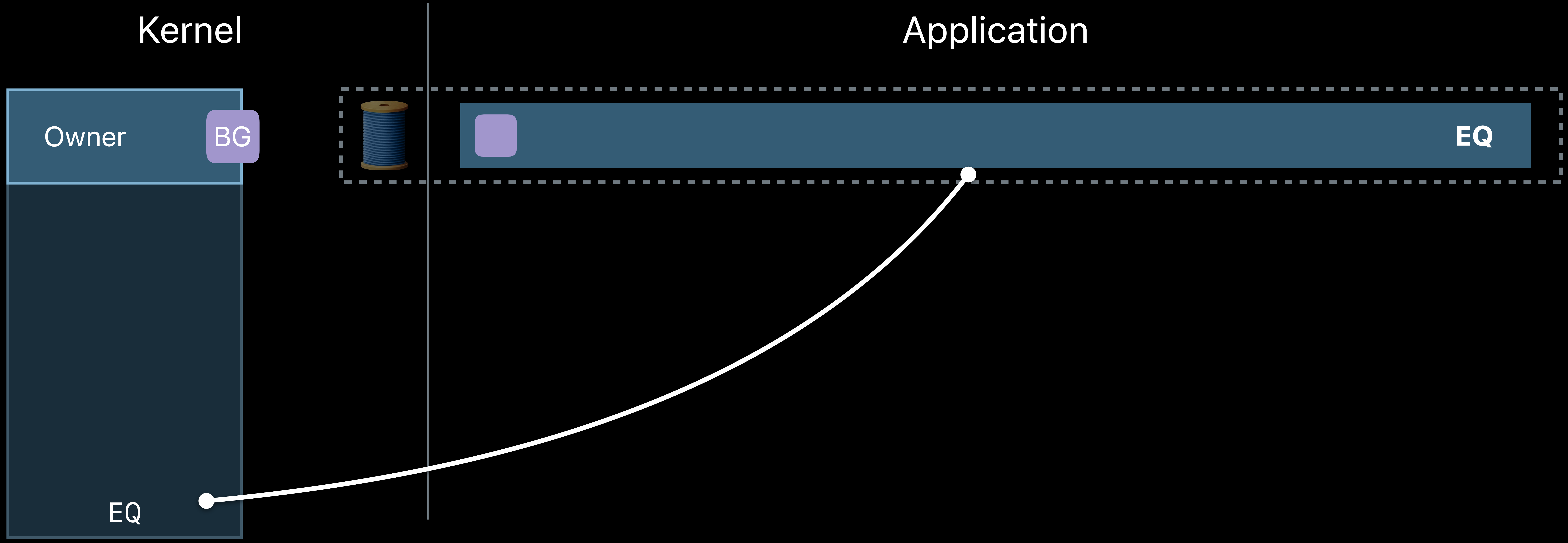


```
EQ.async { … }
```

Unified Queue Identity

Asynchronous workitems

NEW

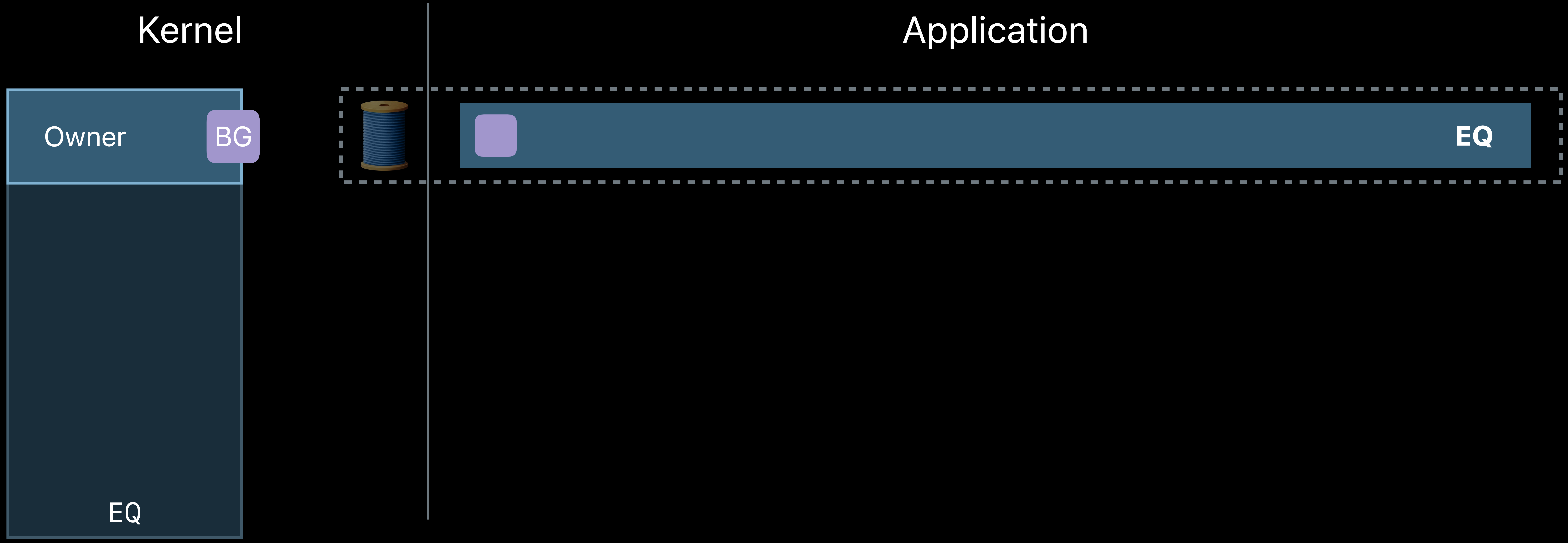


```
EQ.async { … }
```

Unified Queue Identity

Asynchronous workitems

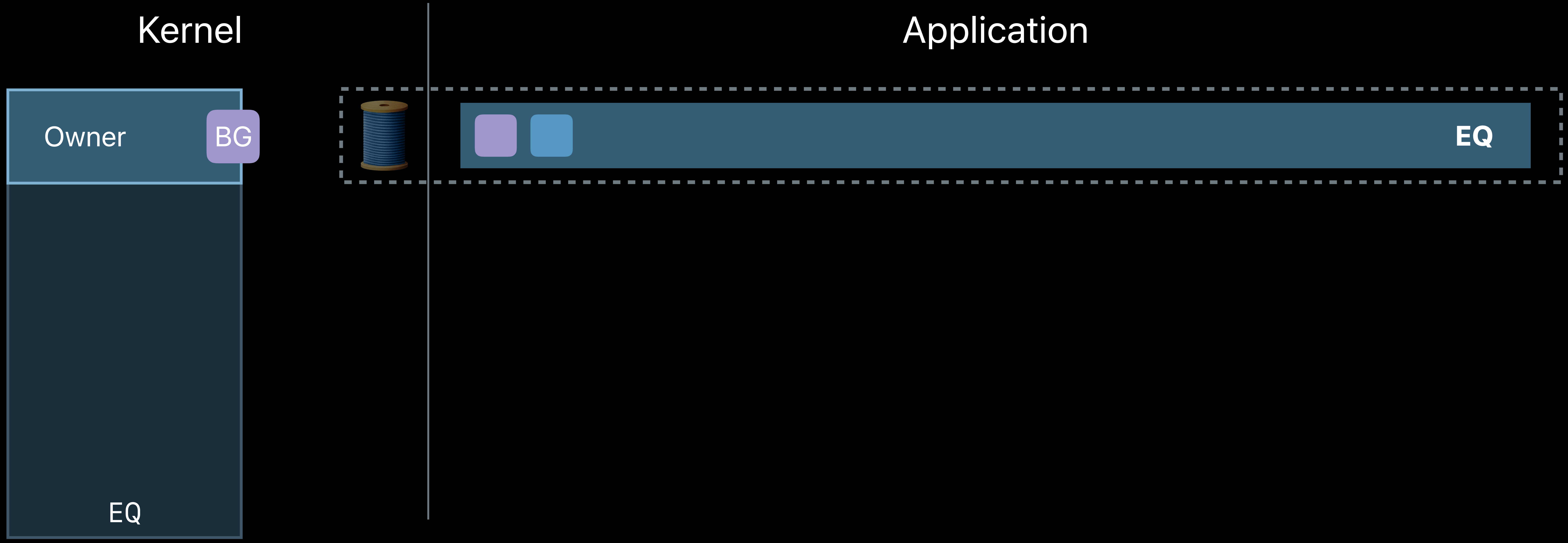
NEW



Unified Queue Identity

Asynchronous workitems

NEW

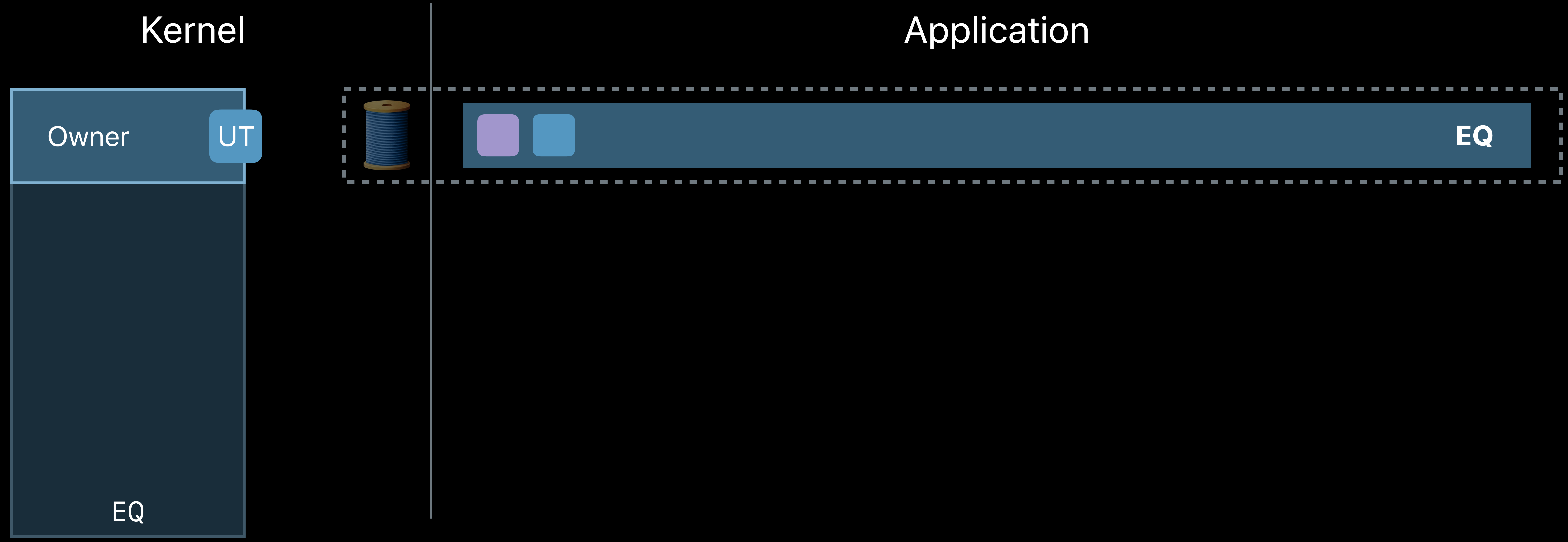


```
EQ.async { … }
```

Unified Queue Identity

Asynchronous workitems

NEW

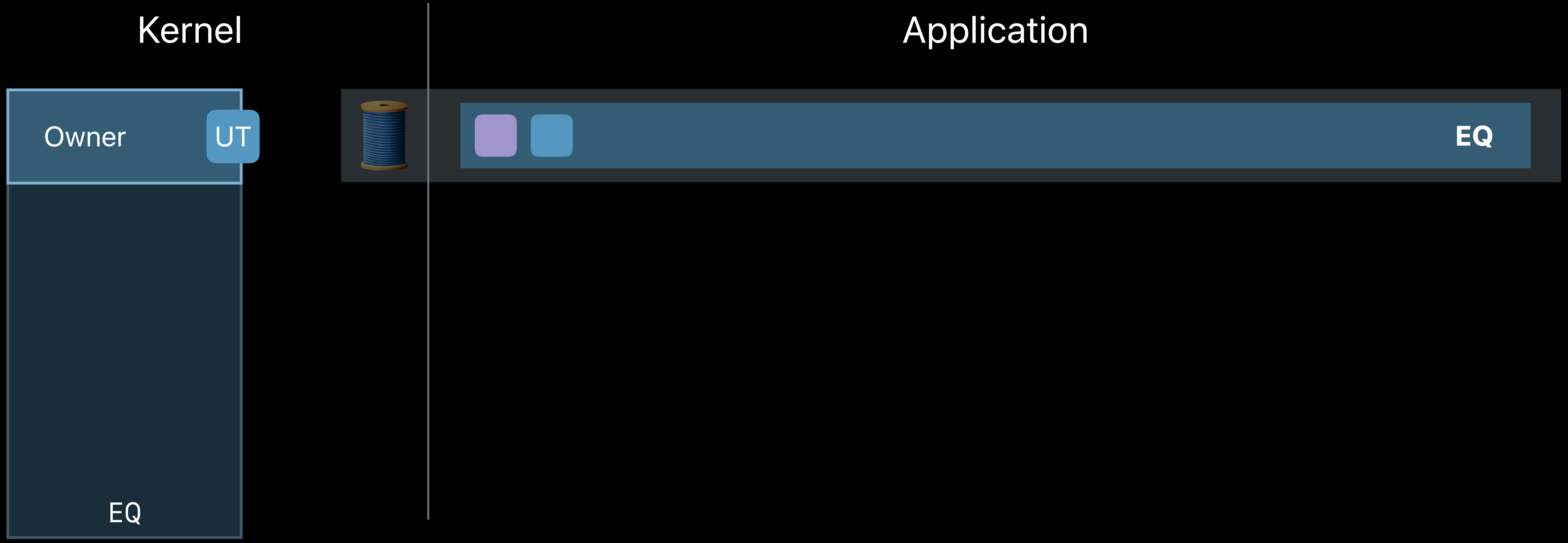


```
EQ.async { … }
```

Unified Queue Identity

Asynchronous workitems

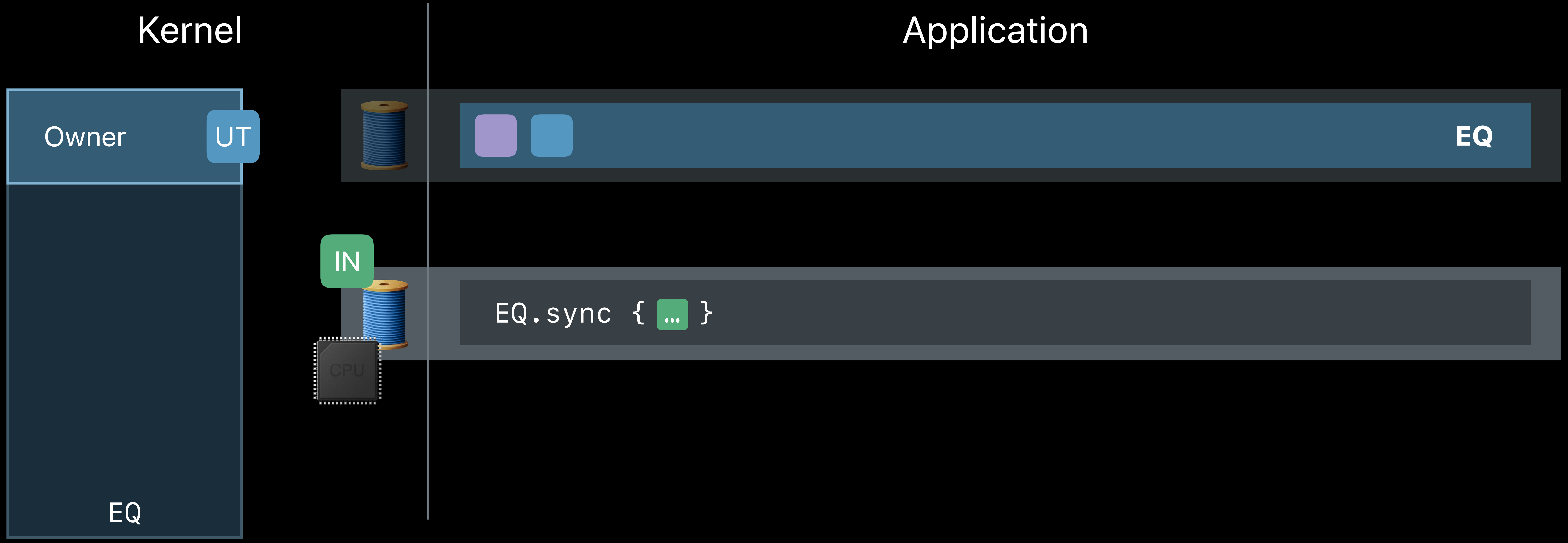
NEW



Unified Queue Identity

Synchronous workitems

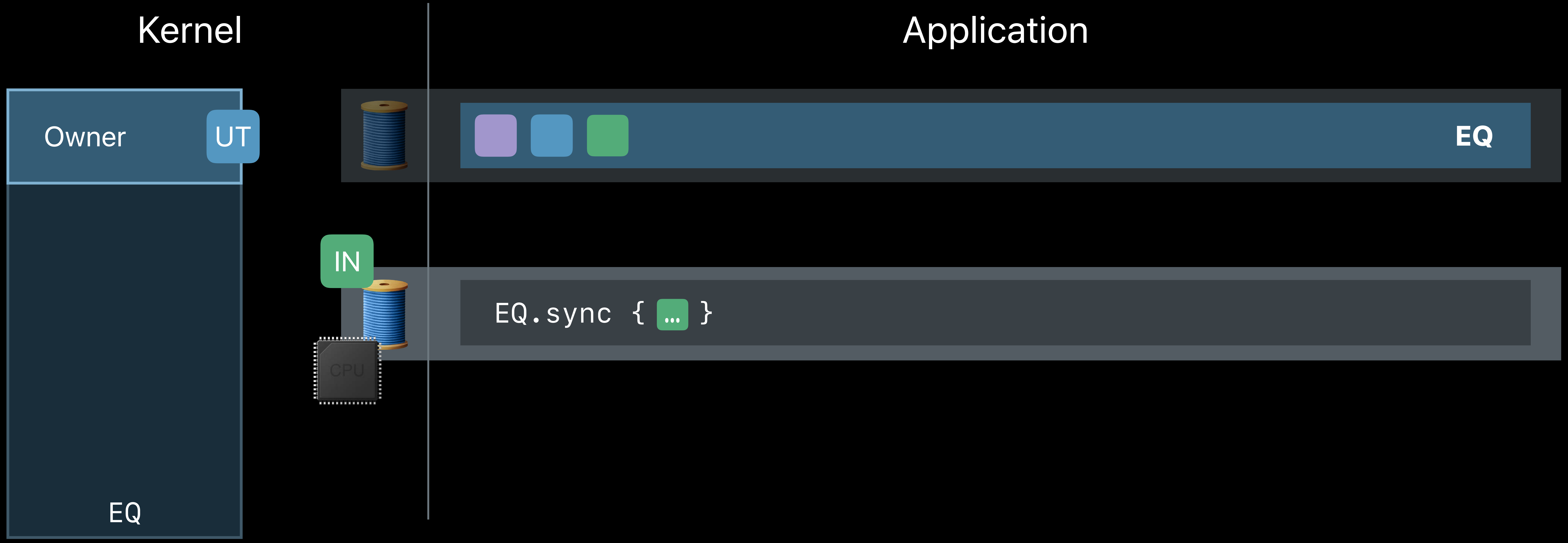
NEW



Unified Queue Identity

Synchronous workitems

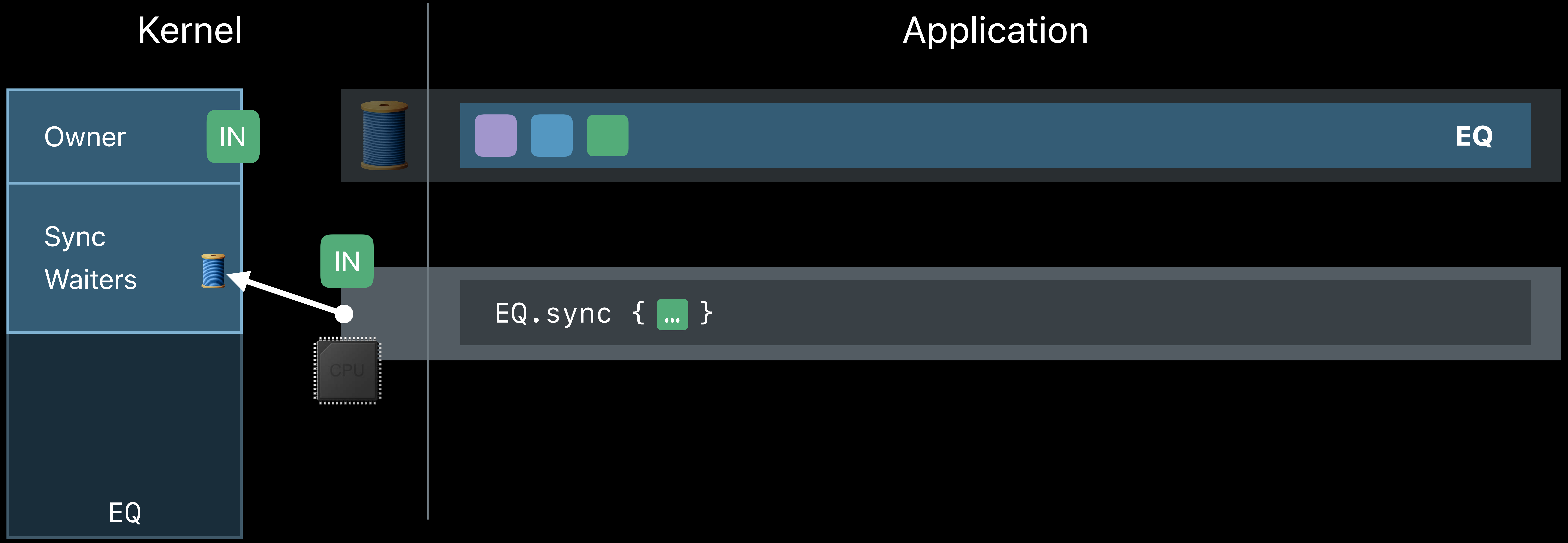
NEW



Unified Queue Identity

Synchronous workitems

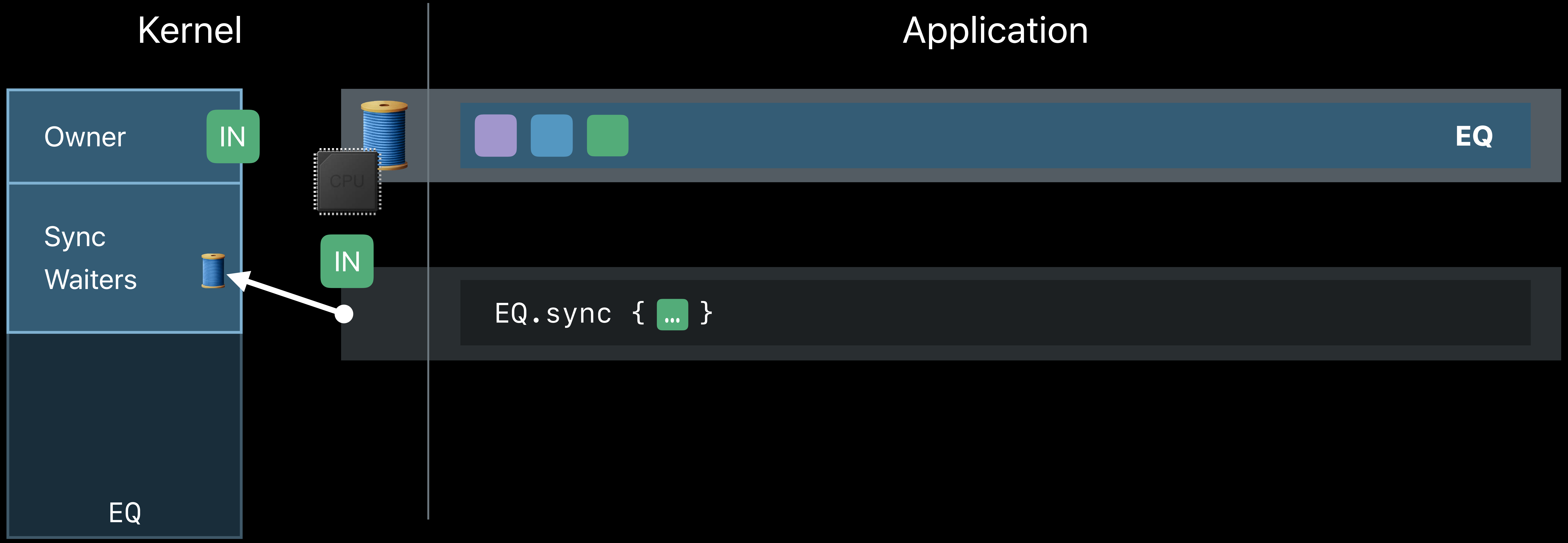
NEW



```
EQ.sync { … }
```

Unified Queue Identity

Synchronous workitems

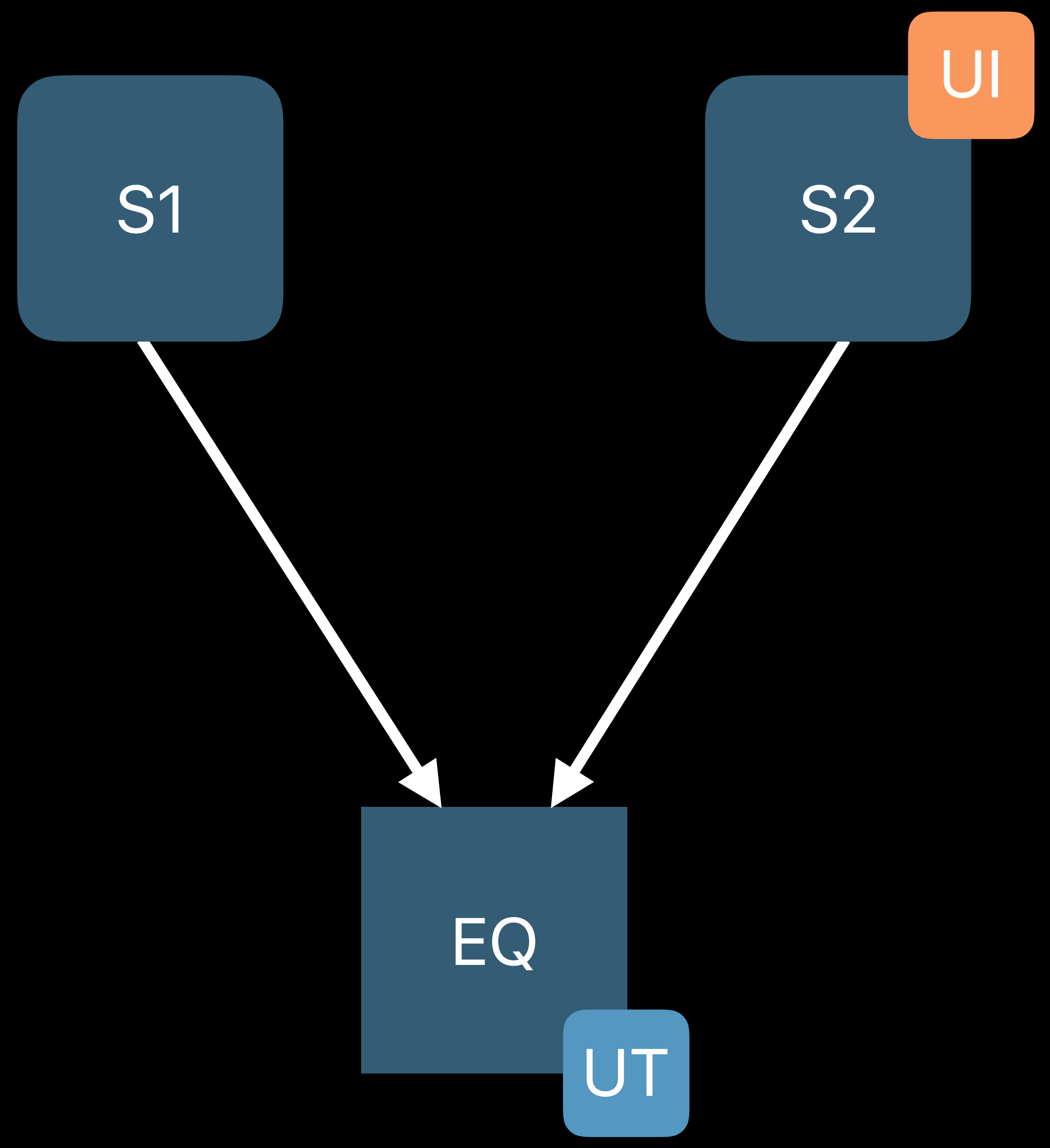


```
EQ.sync { … }
```

One Identity to Find Them All

... and in the kernel bind them

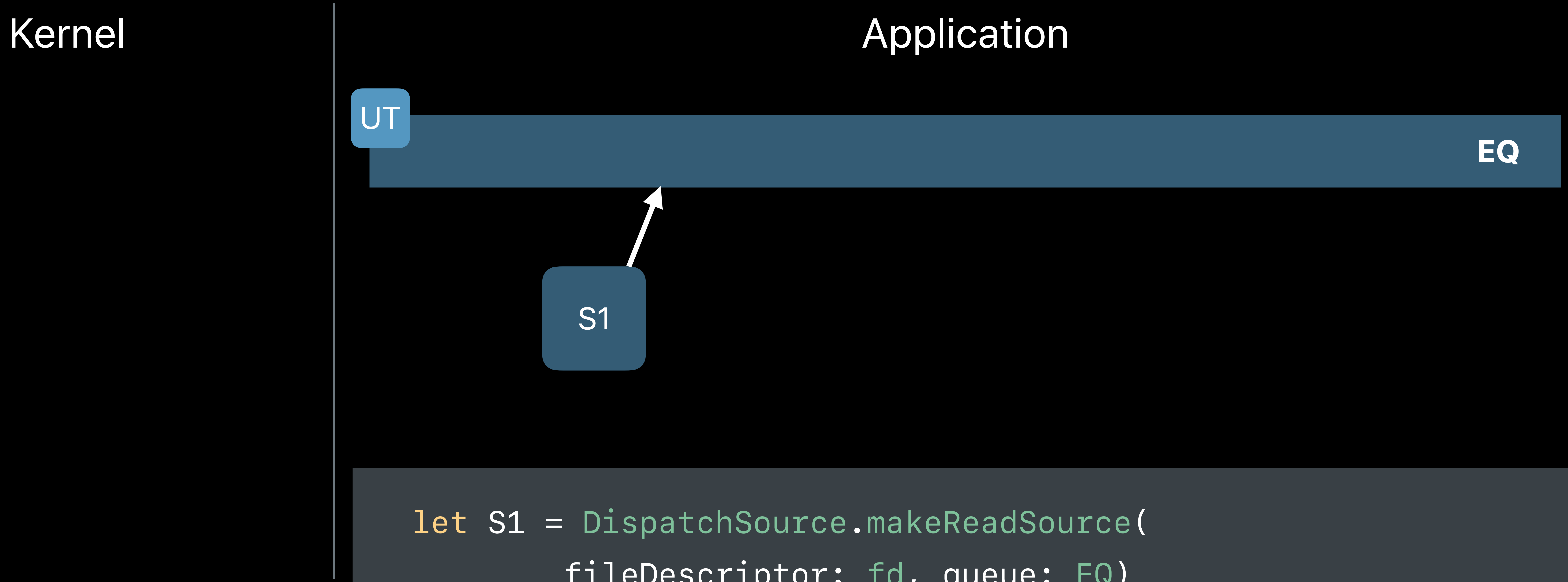
NEW



One Identity to Find Them All

... and in the kernel bind them

NEW

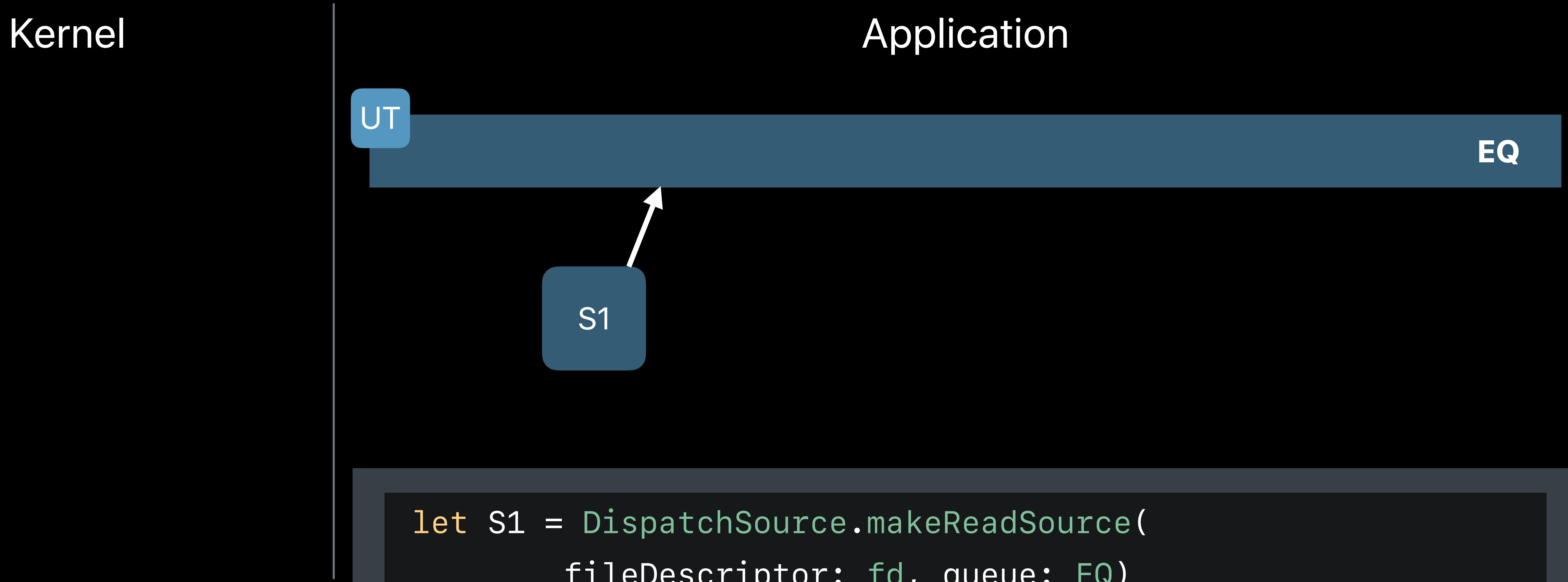


```
let S1 = DispatchSource.makeReadSource(
            fileDescriptor: fd, queue: EQ)
S1.setEventHandler { … }
S1.activate()
```

One Identity to Find Them All

... and in the kernel bind them

NEW

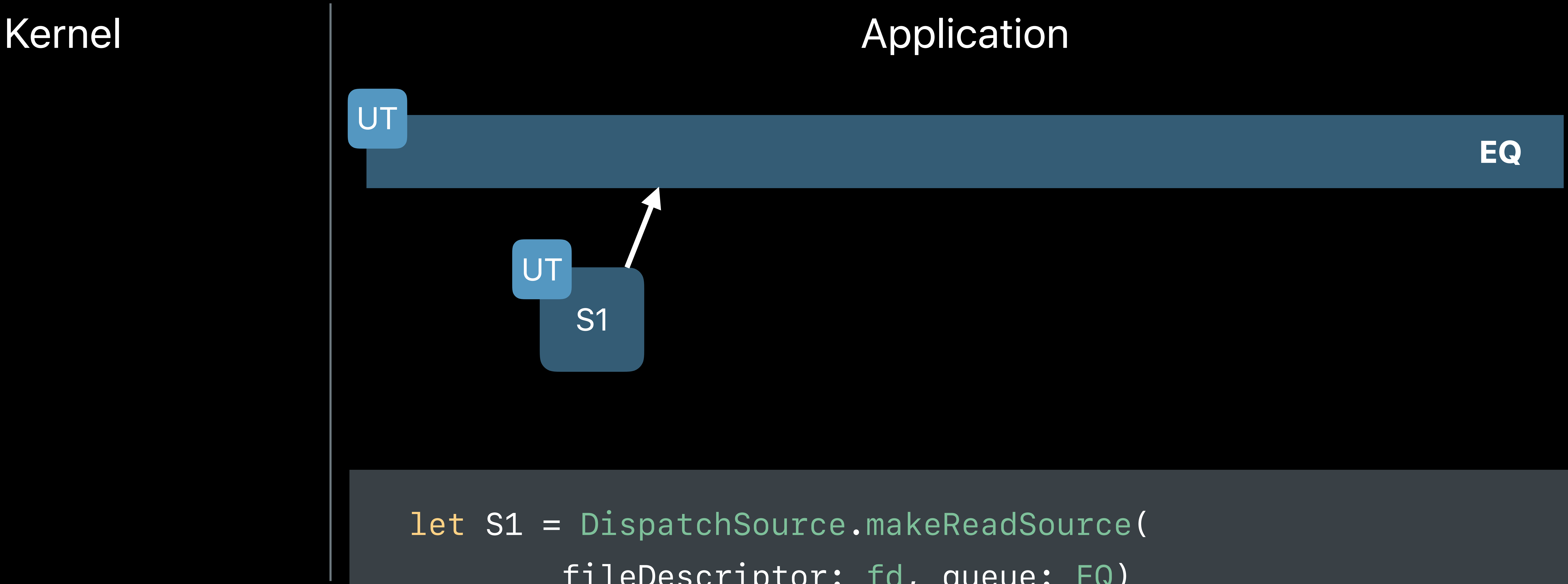


```
let S1 = DispatchSource.makeReadSource(
           fileDescriptor: fd, queue: EQ)
S1.setEventHandler { … }
S1.activate()
```

One Identity to Find Them All

... and in the kernel bind them

NEW

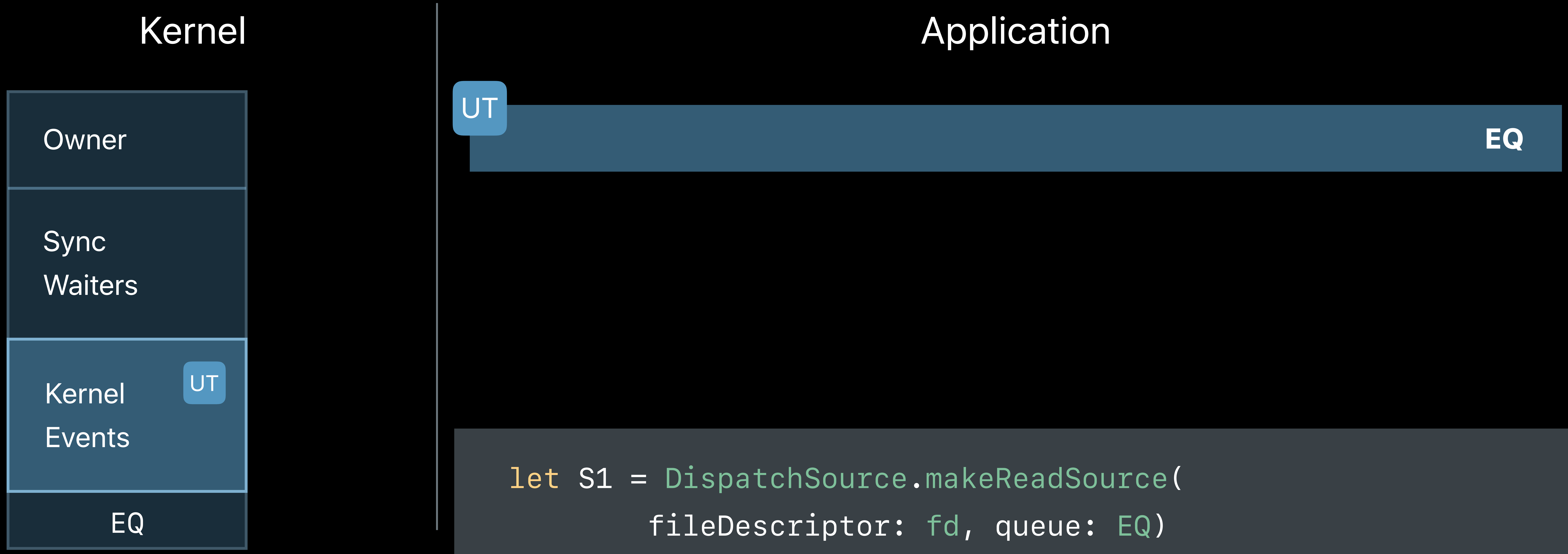


```
let S1 = DispatchSource.makeReadSource(
        fileDescriptor: fd, queue: EQ)
S1.setEventHandler { … }
S1.activate()
```

One Identity to Find Them All

... and in the kernel bind them

NEW

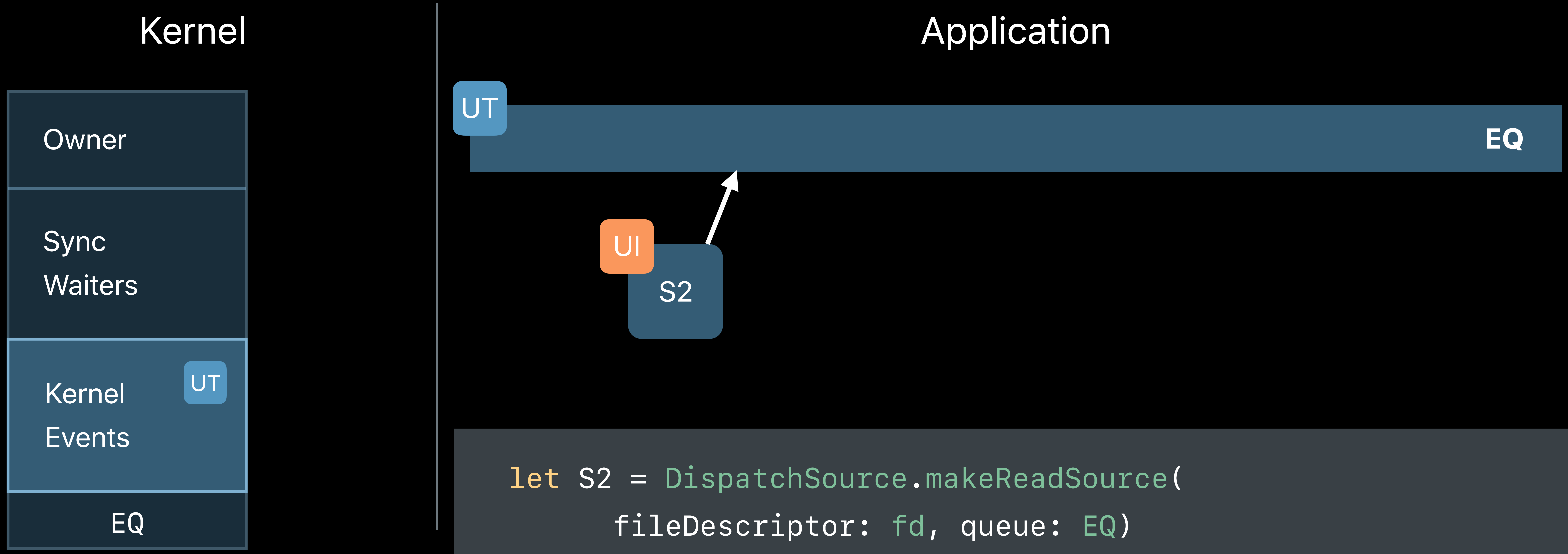


```
let S1 = DispatchSource.makeReadSource(
         fileDescriptor: fd, queue: EQ)
S1.setEventHandler { … }
S1.activate()
```

One Identity to Find Them All

... and in the kernel bind them

NEW

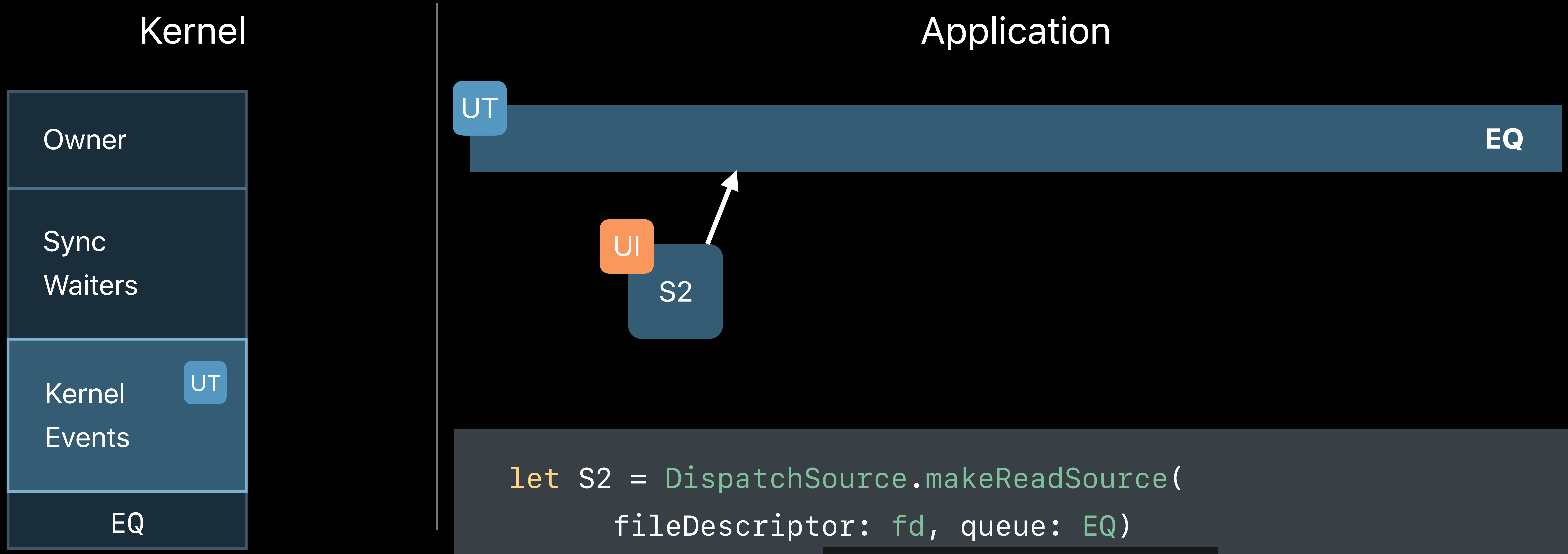


```
let S2 = DispatchSource.makeReadSource(
      fileDescriptor: fd, queue: EQ)
S2.setEventHandler(qos: .UserInteractive) { … }
S2.activate()
```

One Identity to Find Them All

... and in the kernel bind them

NEW

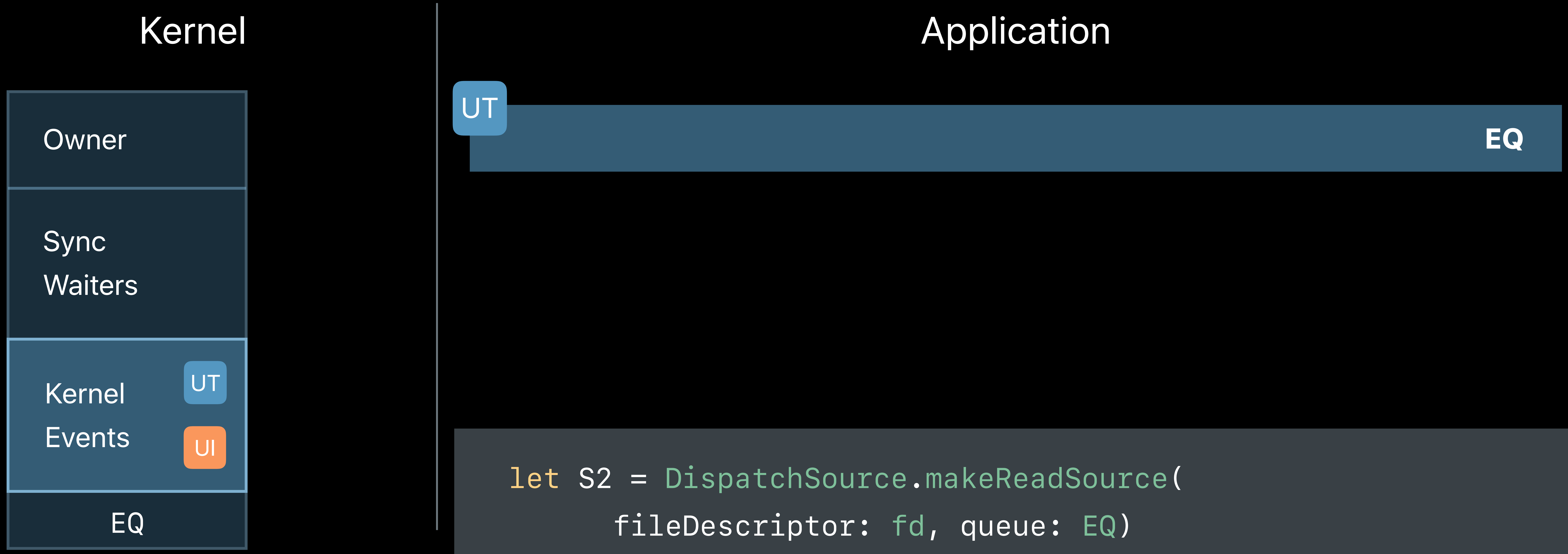


```
let S2 = DispatchSource.makeReadSource(
      fileDescriptor: fd, queue: EQ)
S2.setEventHandler(qos: .UserInteractive) { … }
S2.activate()
```

One Identity to Find Them All

... and in the kernel bind them

NEW



```
let S2 = DispatchSource.makeReadSource(
      fileDescriptor: fd, queue: EQ)
S2.setEventHandler(qos: .UserInteractive) { … }
S2.activate()
```

Too Much of a Good Thing

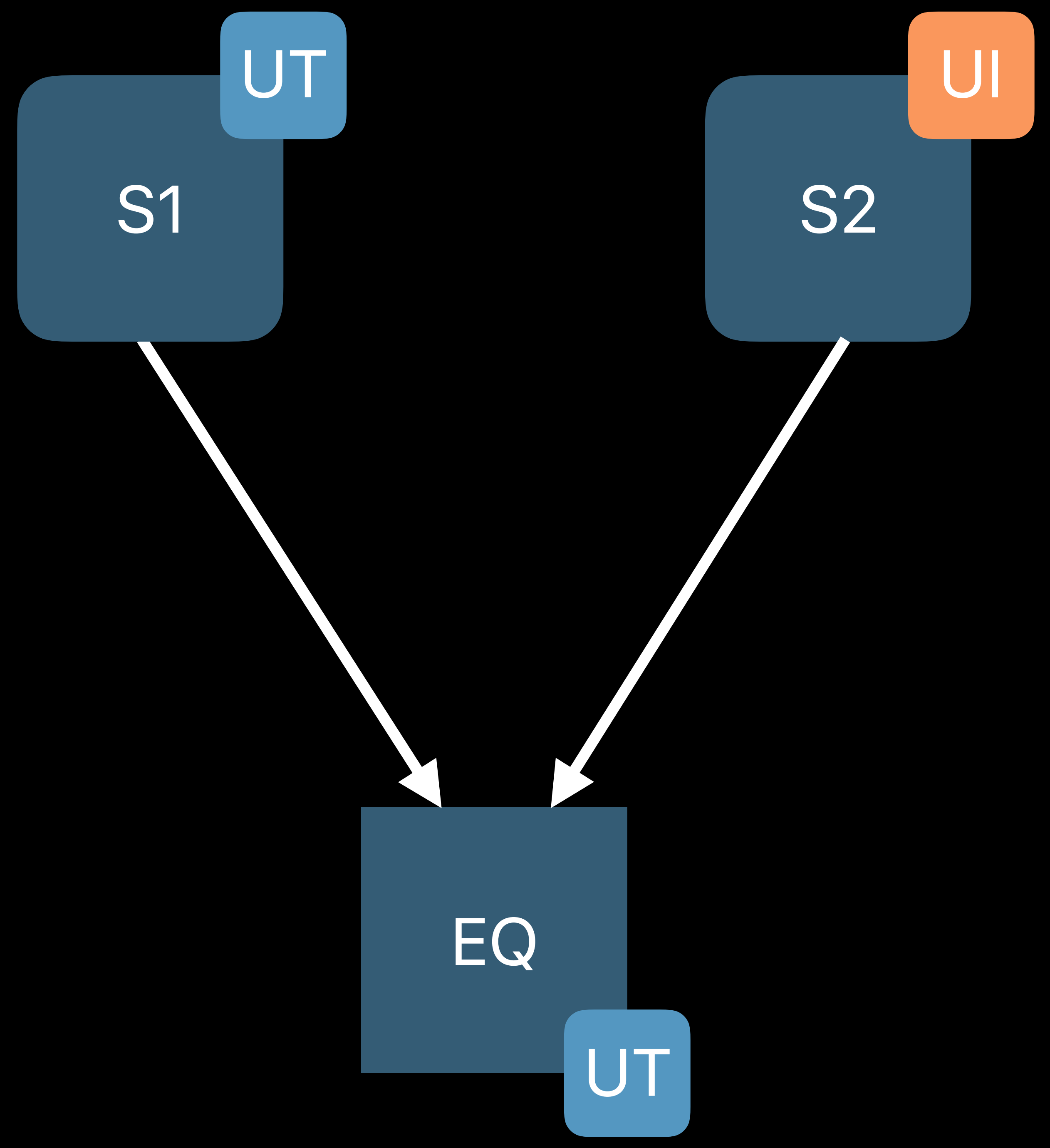
Repeatedly waiting for exclusive access to contended resources

Repeatedly switching between independent operations

Repeatedly bouncing an operation between threads

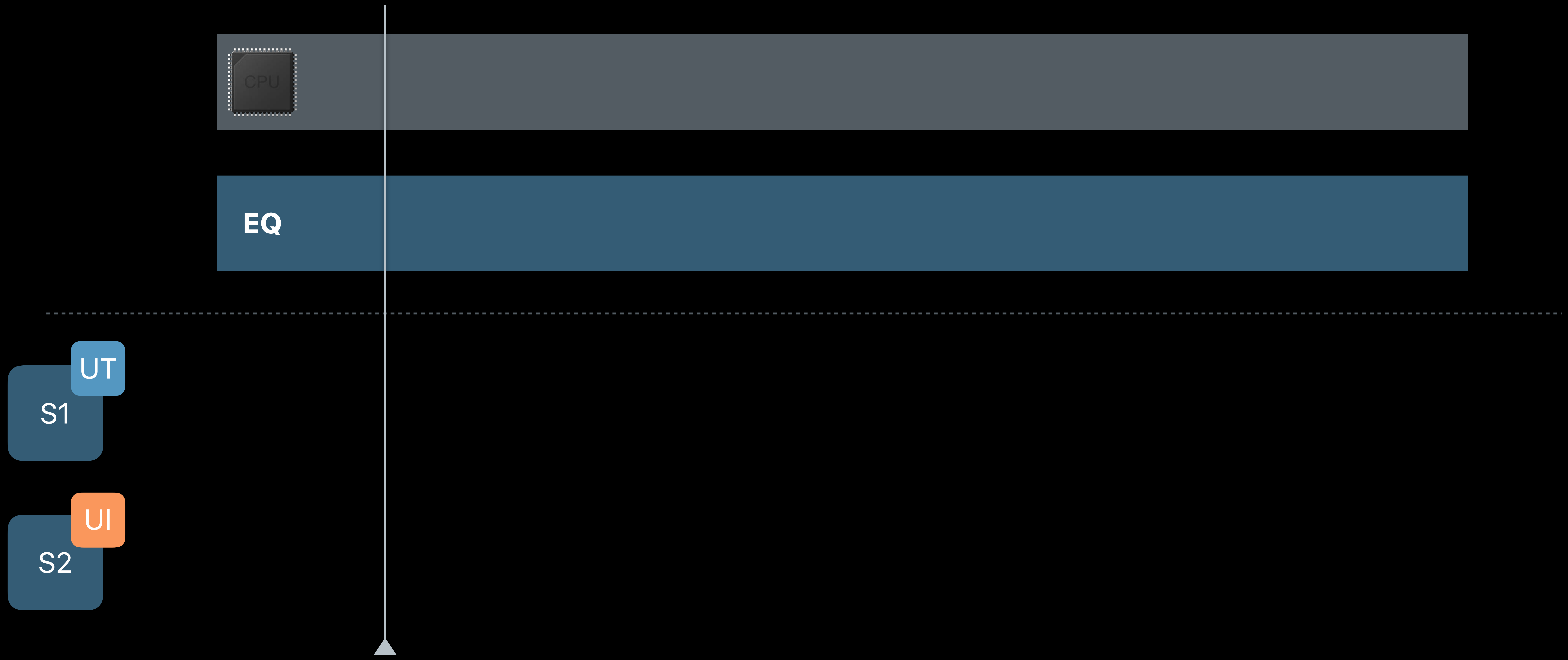
Without Unified Identity

In macOS Sierra and iOS 10



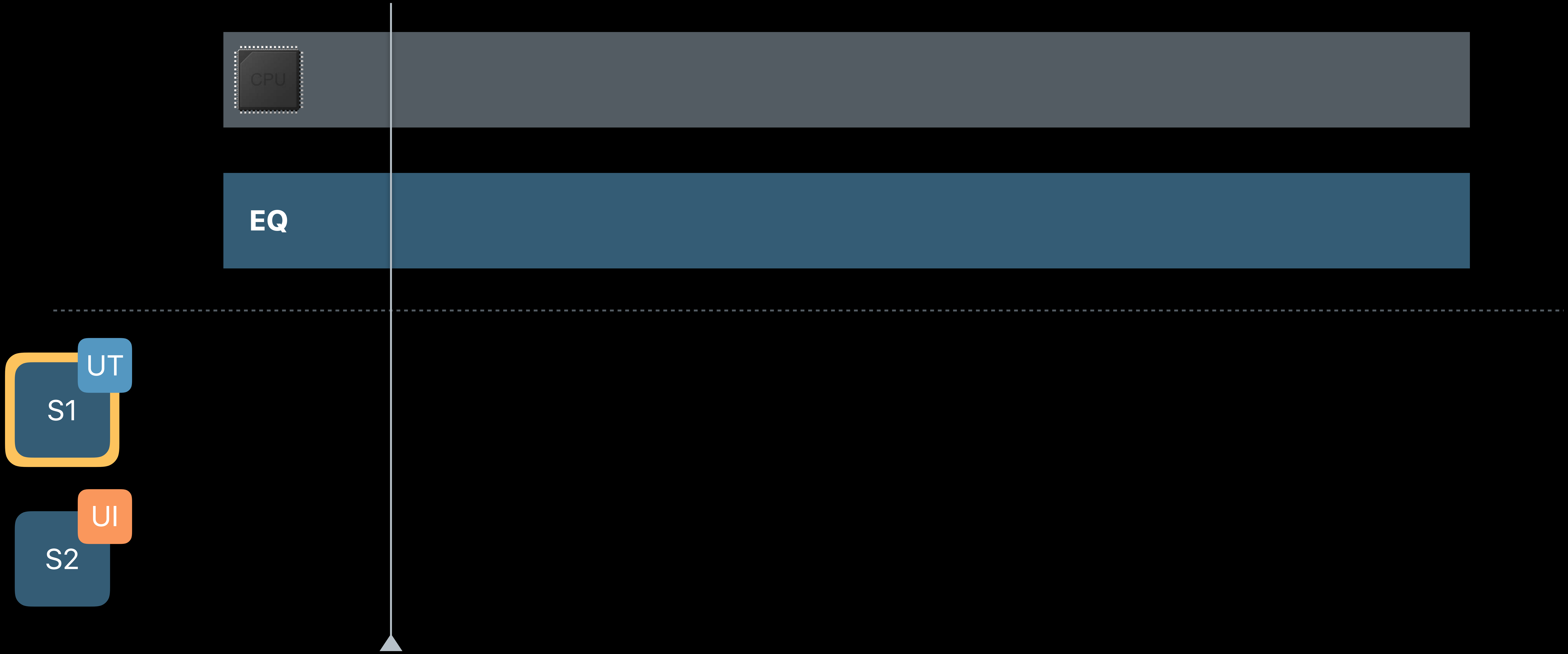
Without Unified Identity

In macOS Sierra and iOS 10



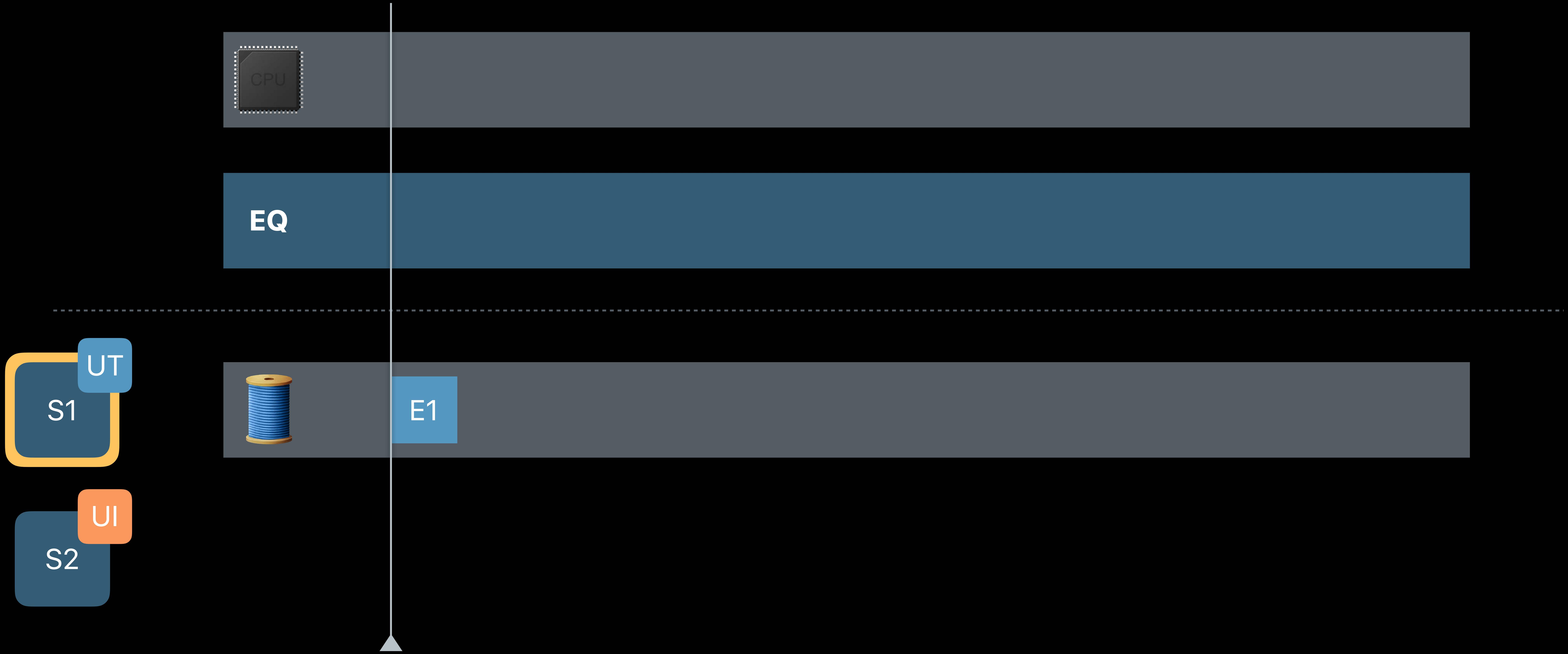
Without Unified Identity

In macOS Sierra and iOS 10



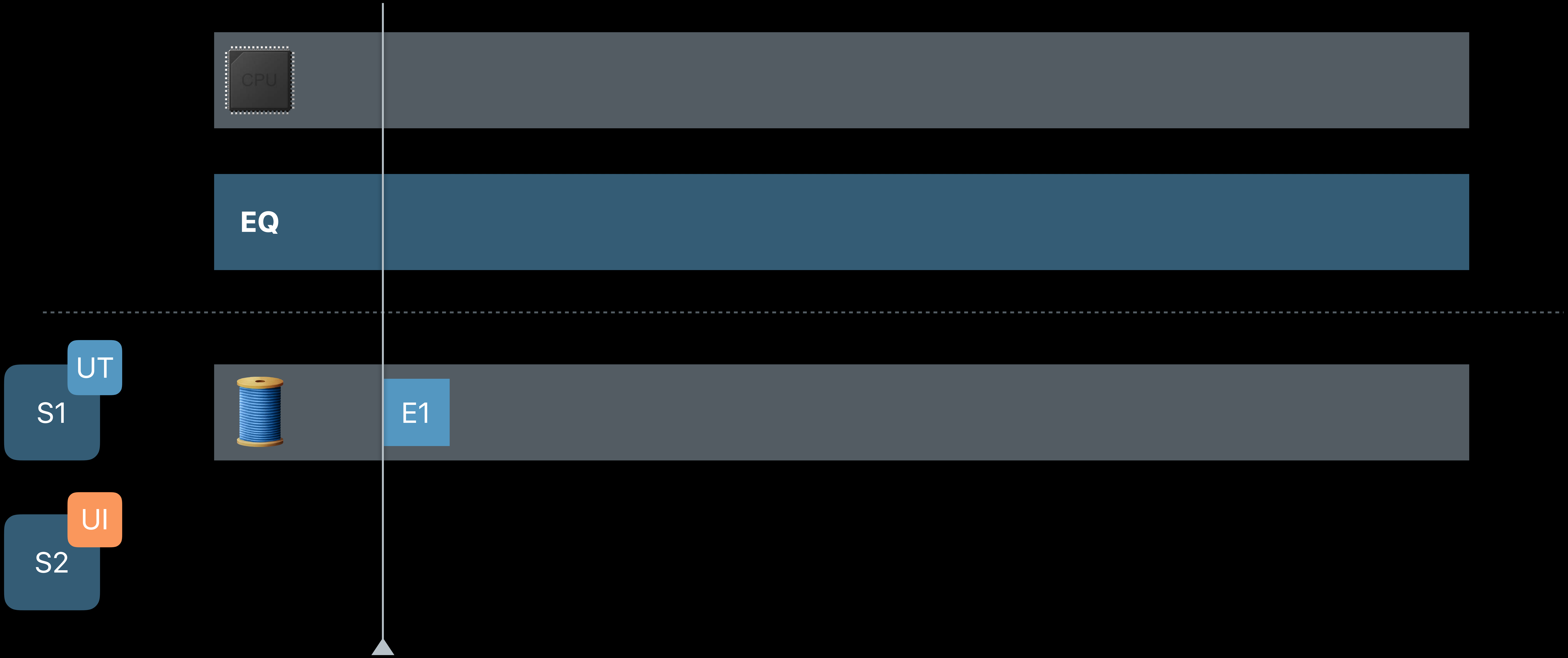
Without Unified Identity

In macOS Sierra and iOS 10



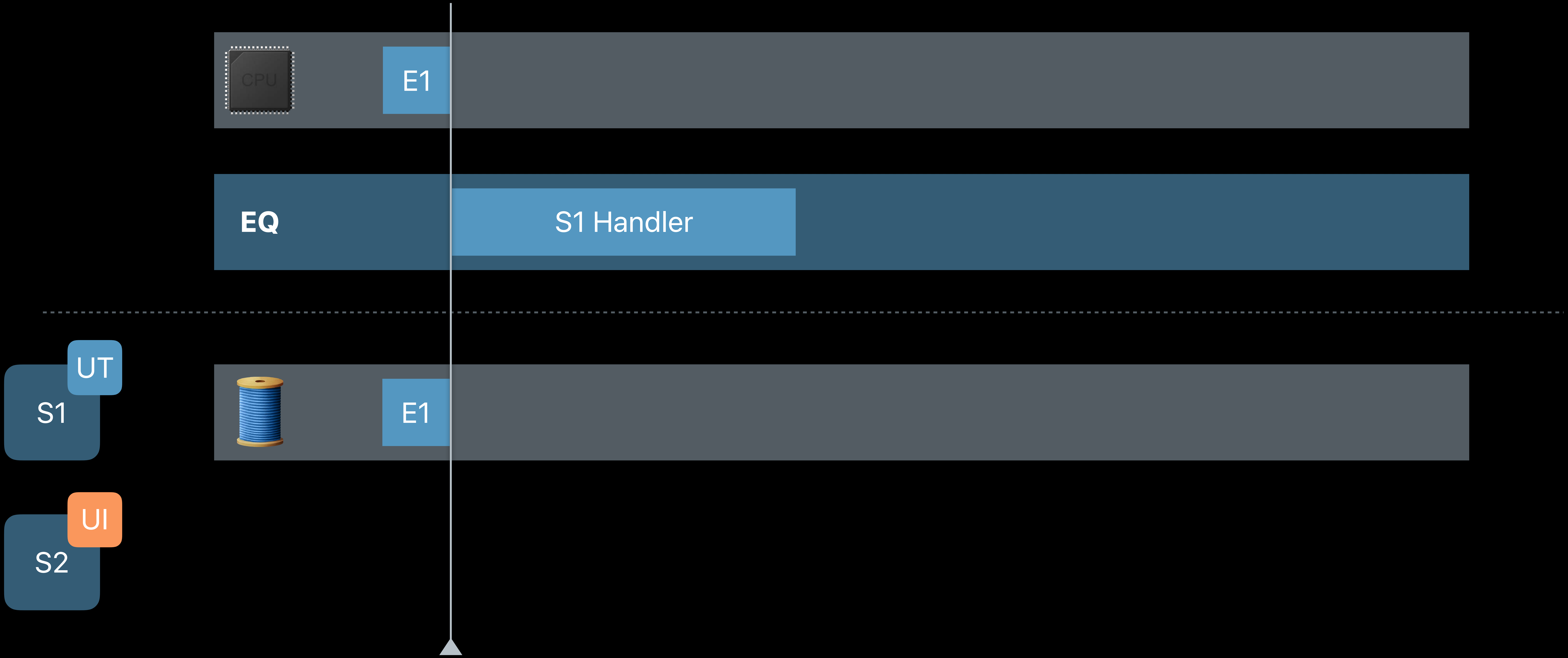
Without Unified Identity

In macOS Sierra and iOS 10



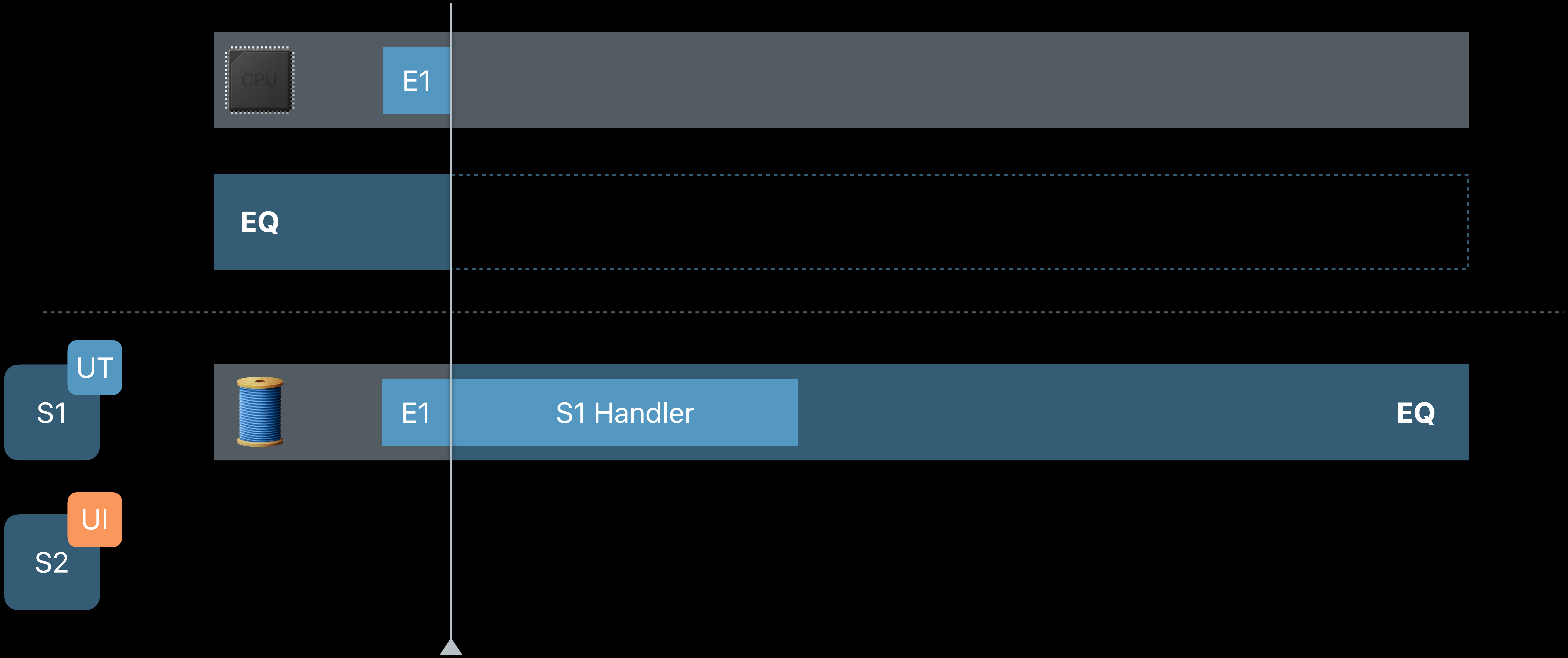
Without Unified Identity

In macOS Sierra and iOS 10



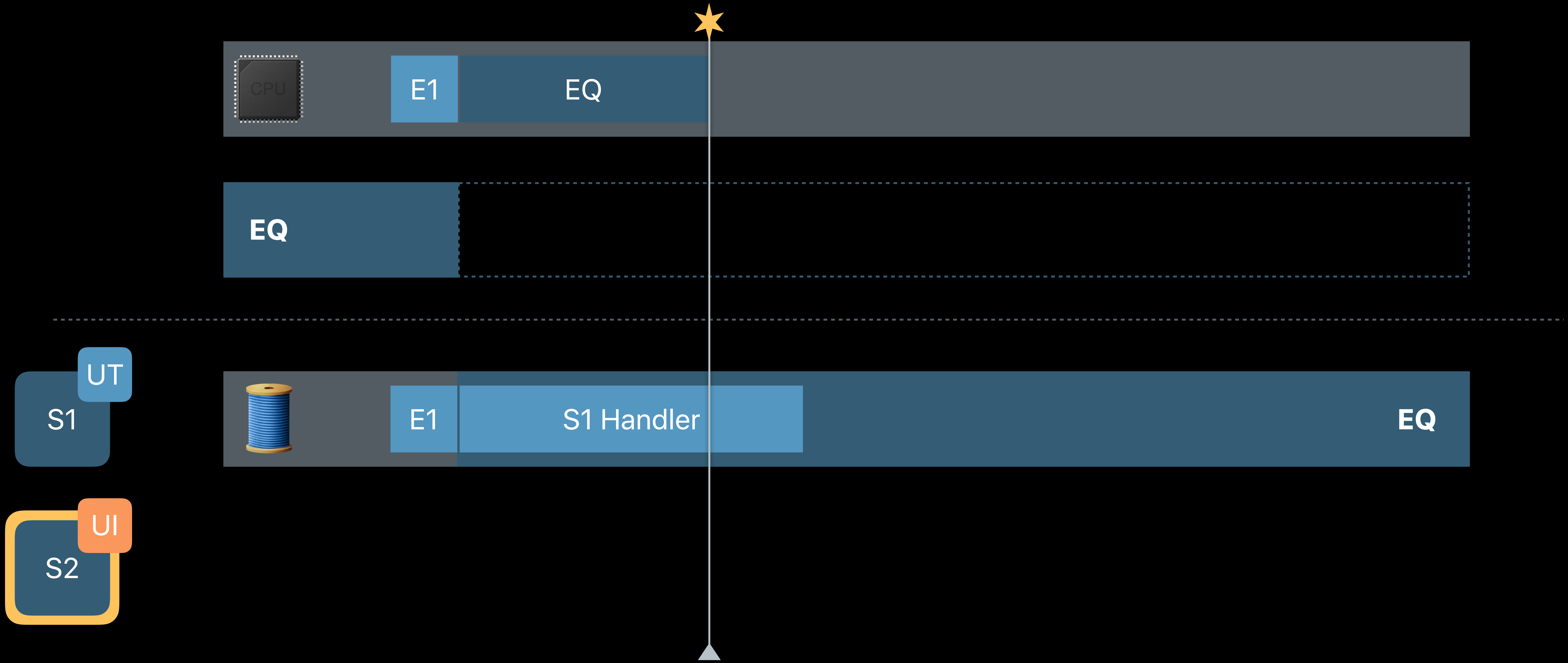
Without Unified Identity

In macOS Sierra and iOS 10



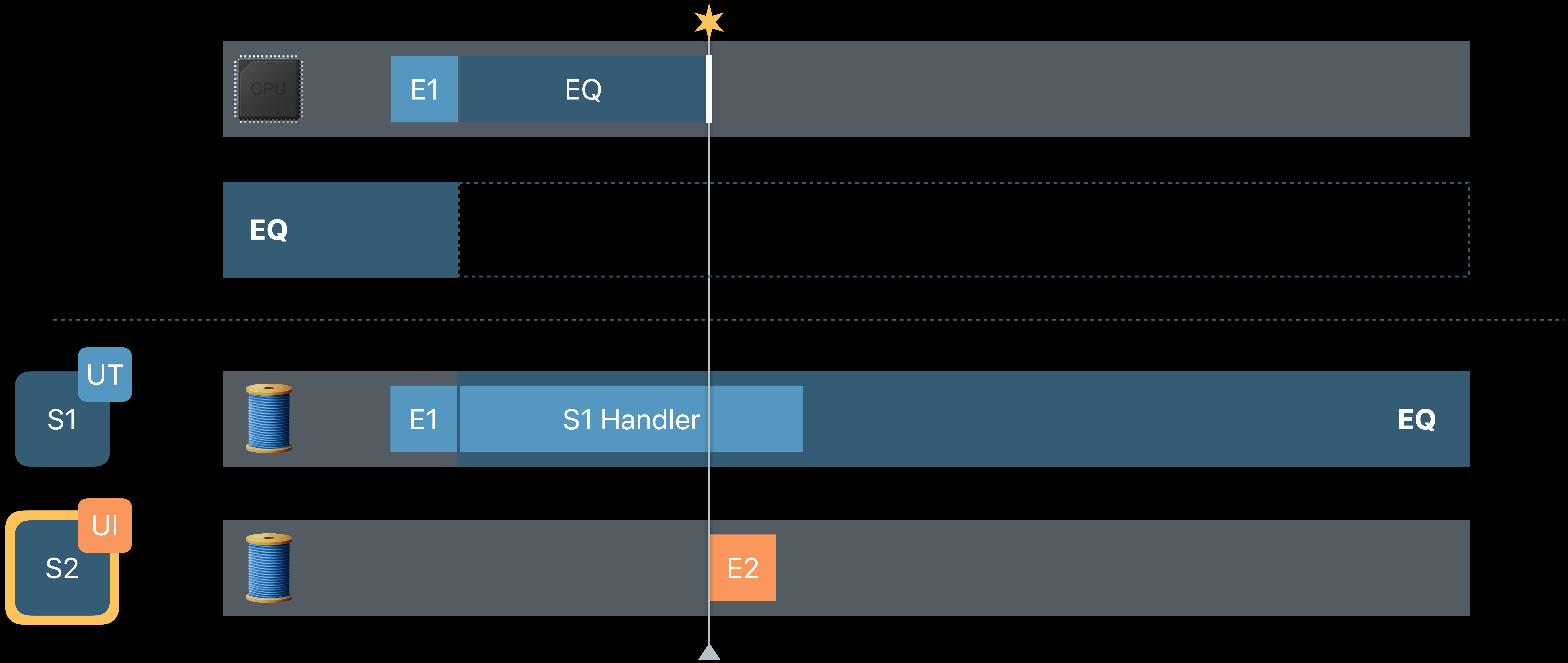
Without Unified Identity

In macOS Sierra and iOS 10



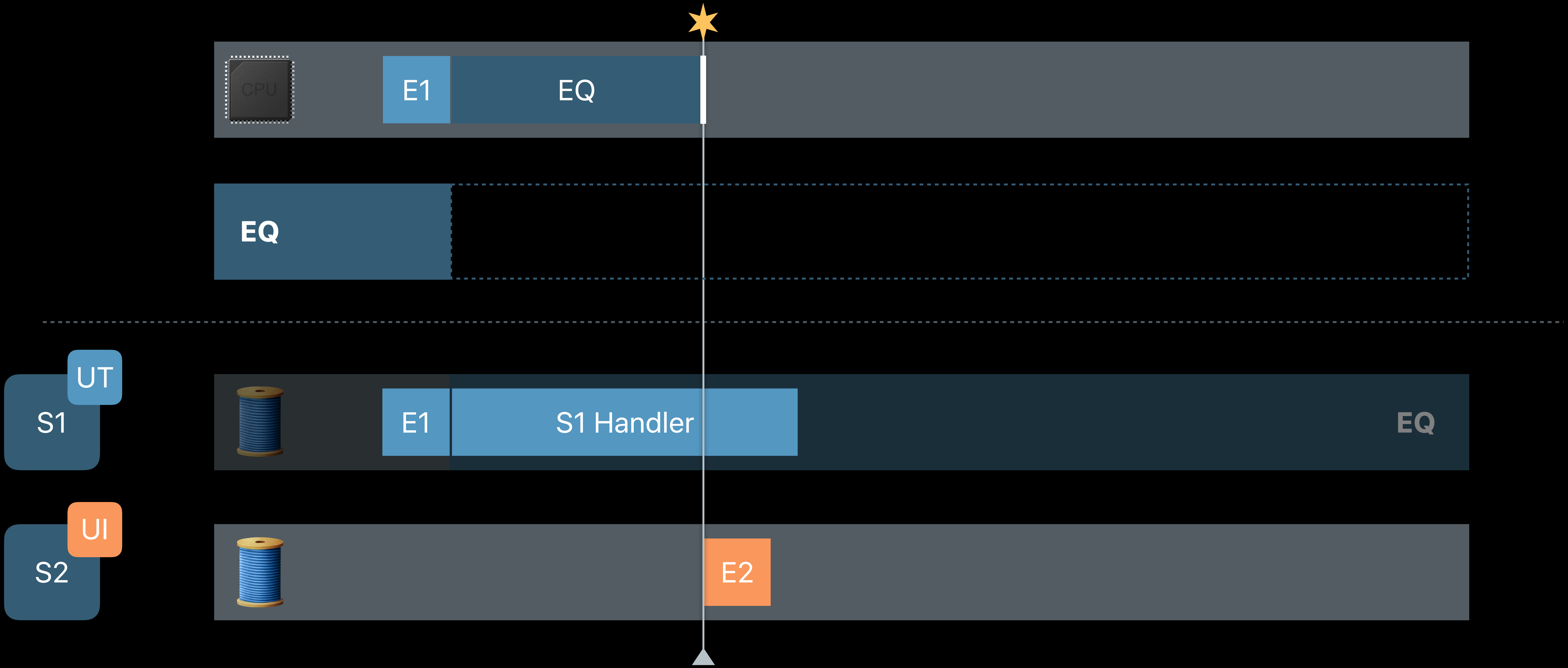
Without Unified Identity

In macOS Sierra and iOS 10



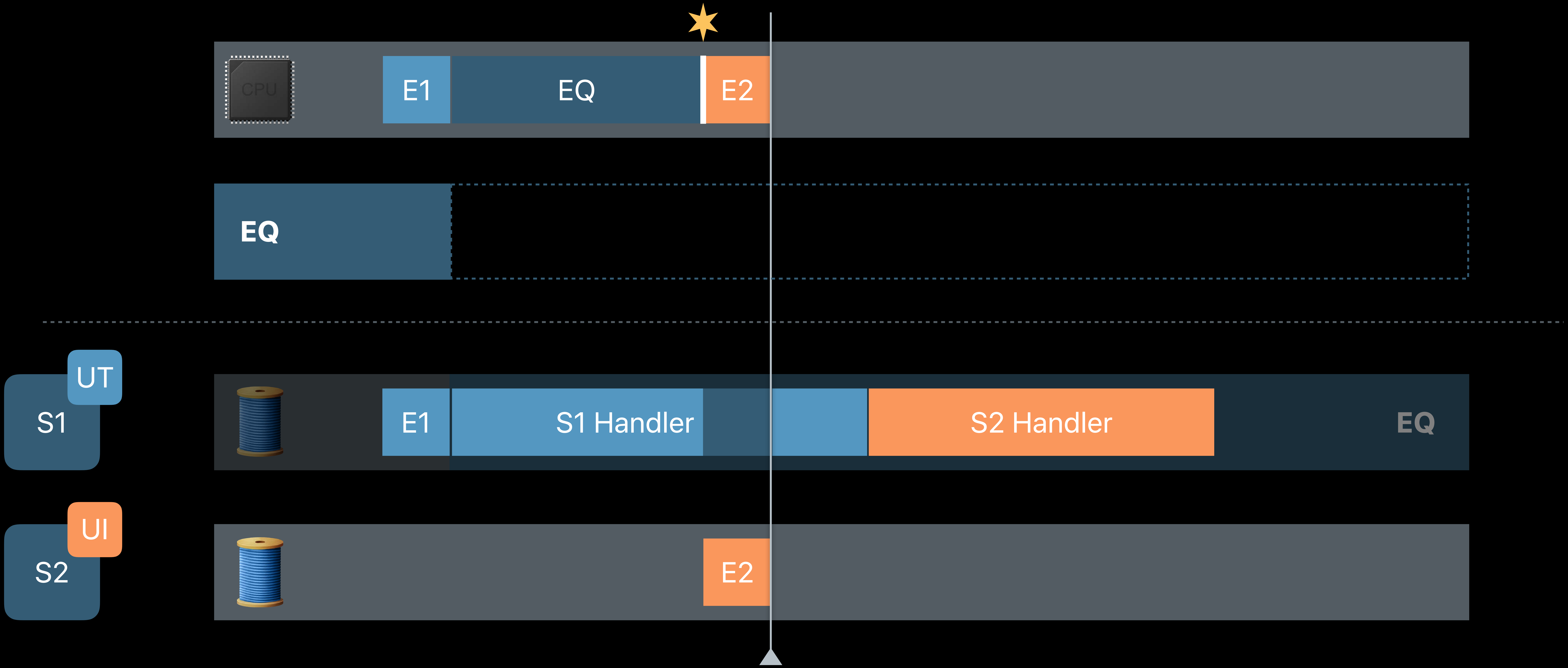
Without Unified Identity

In macOS Sierra and iOS 10



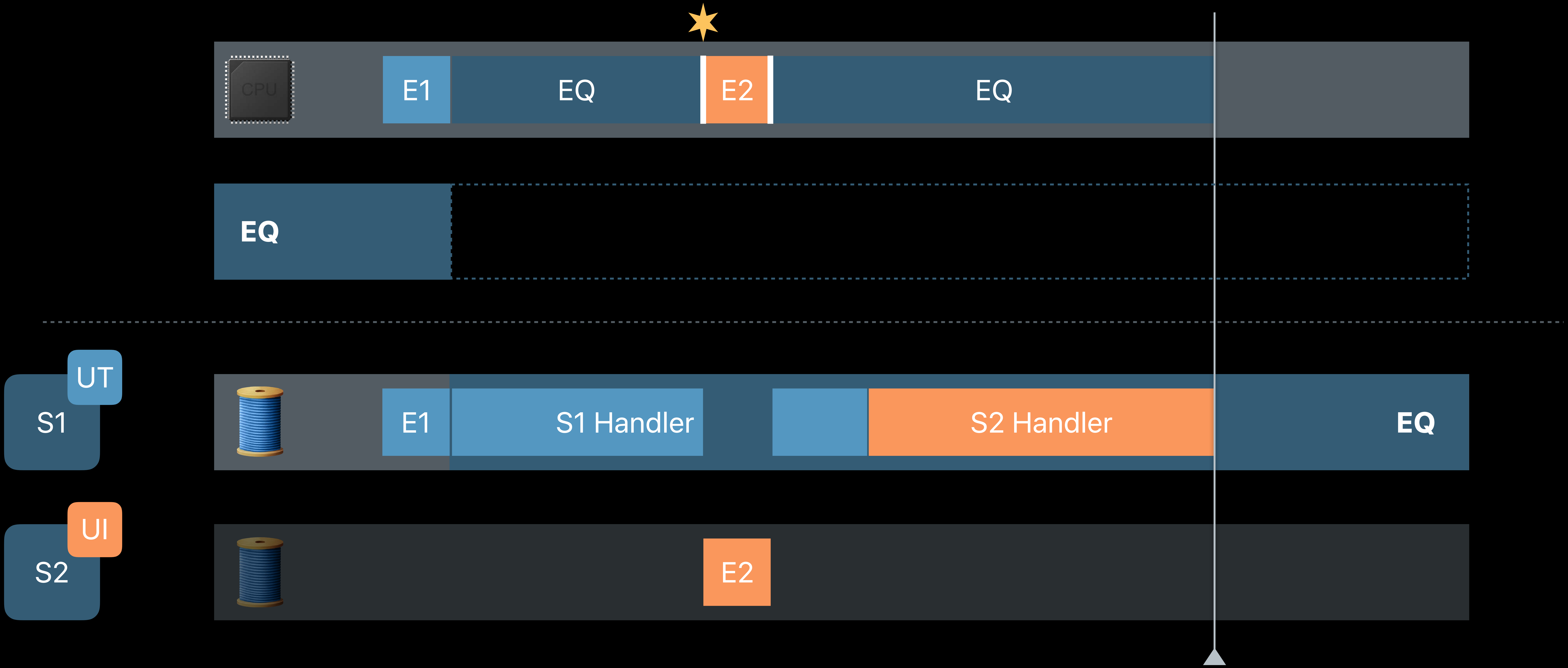
Without Unified Identity

In macOS Sierra and iOS 10

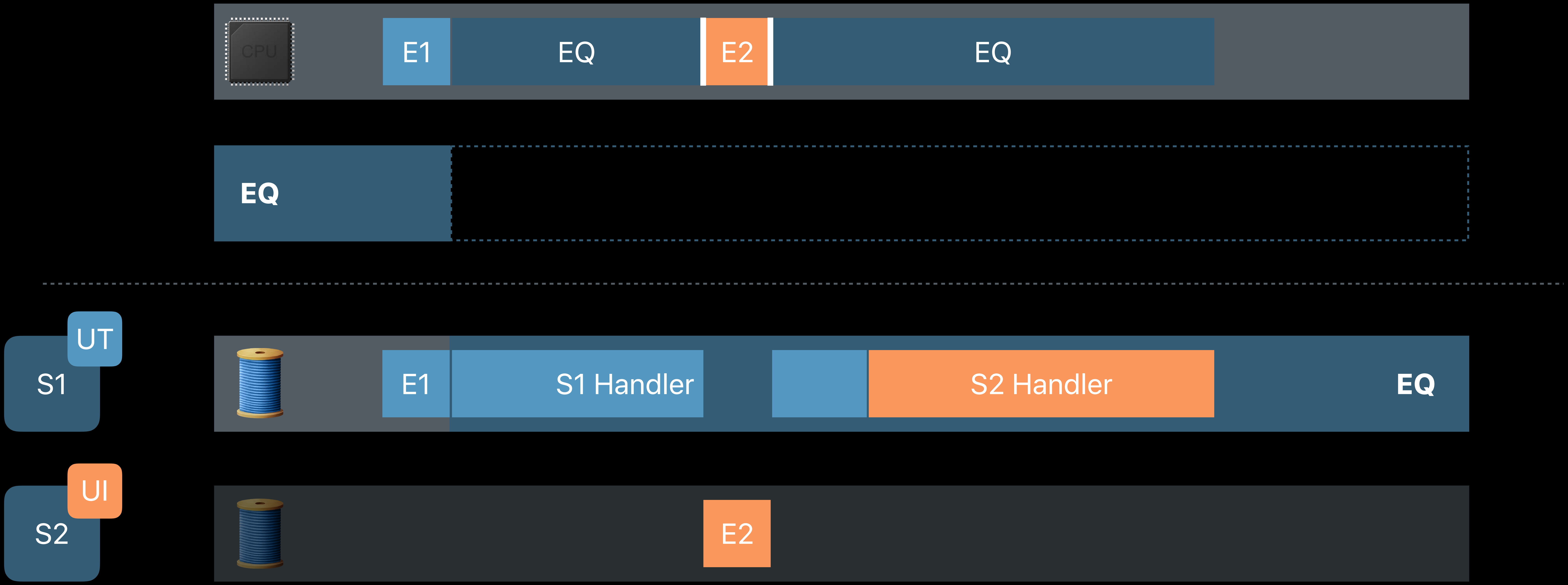


Without Unified Identity

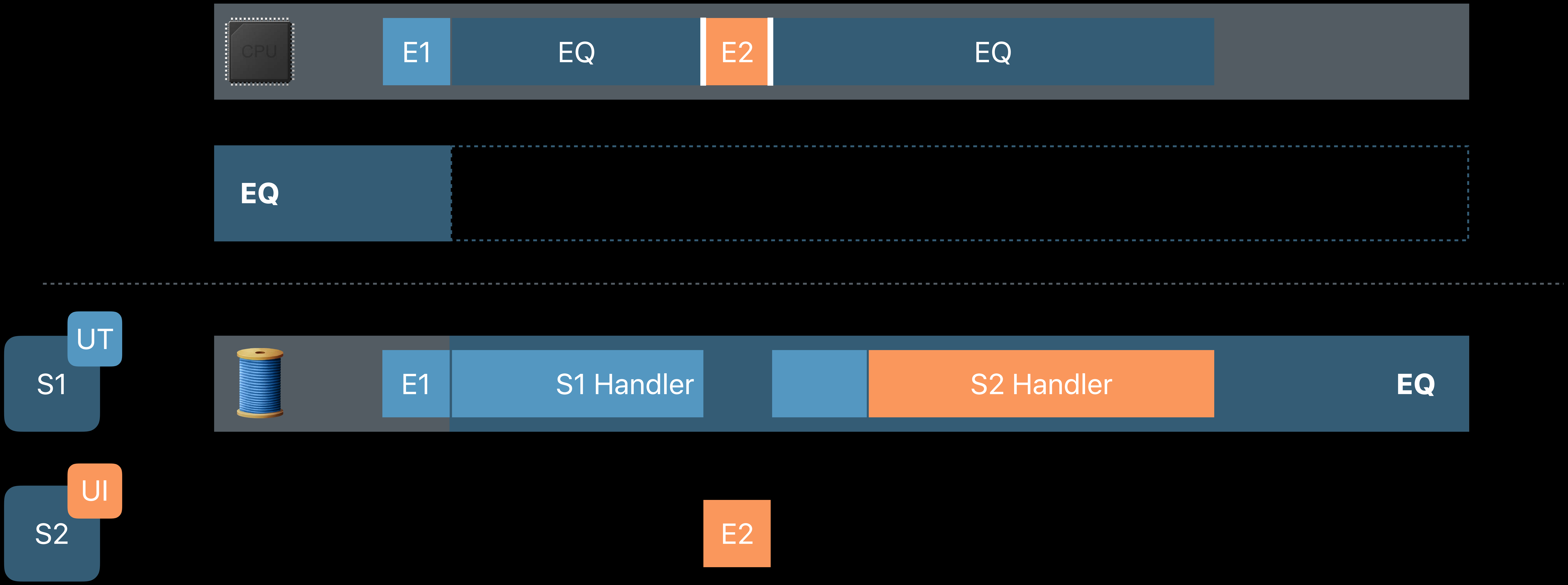
In macOS Sierra and iOS 10



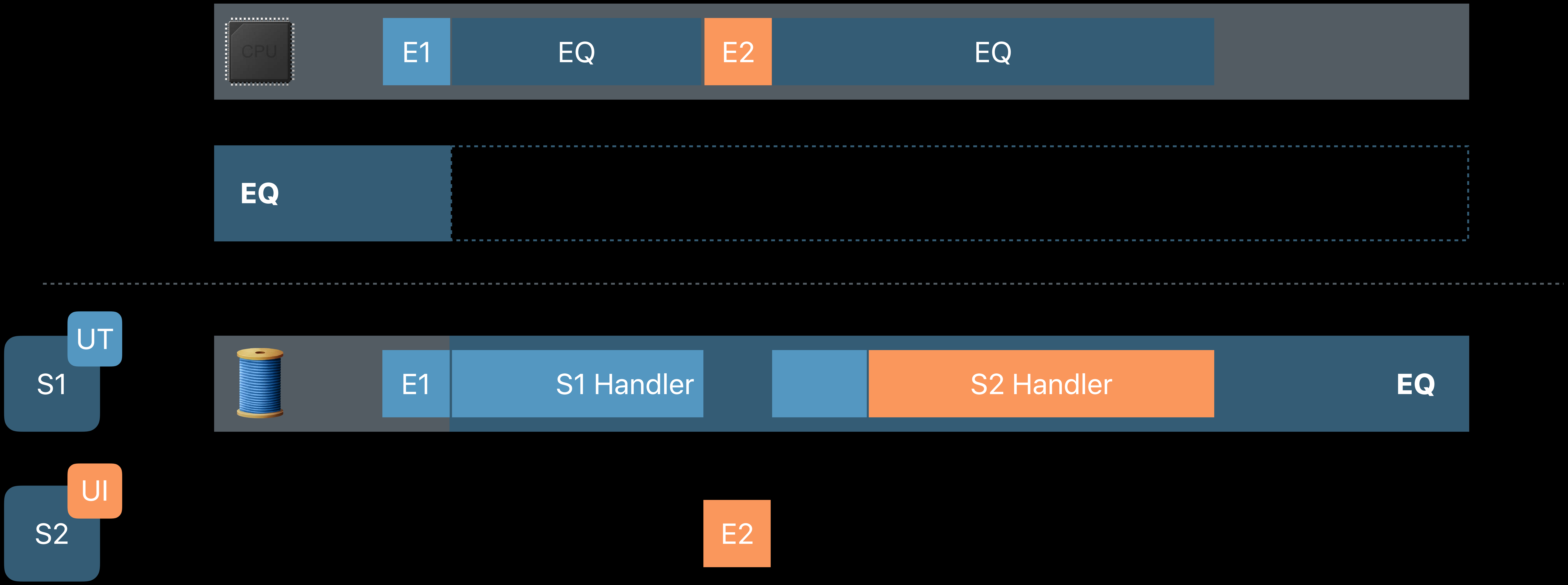
Leveraging Ownership and Unified Identity



Leveraging Ownership and Unified Identity

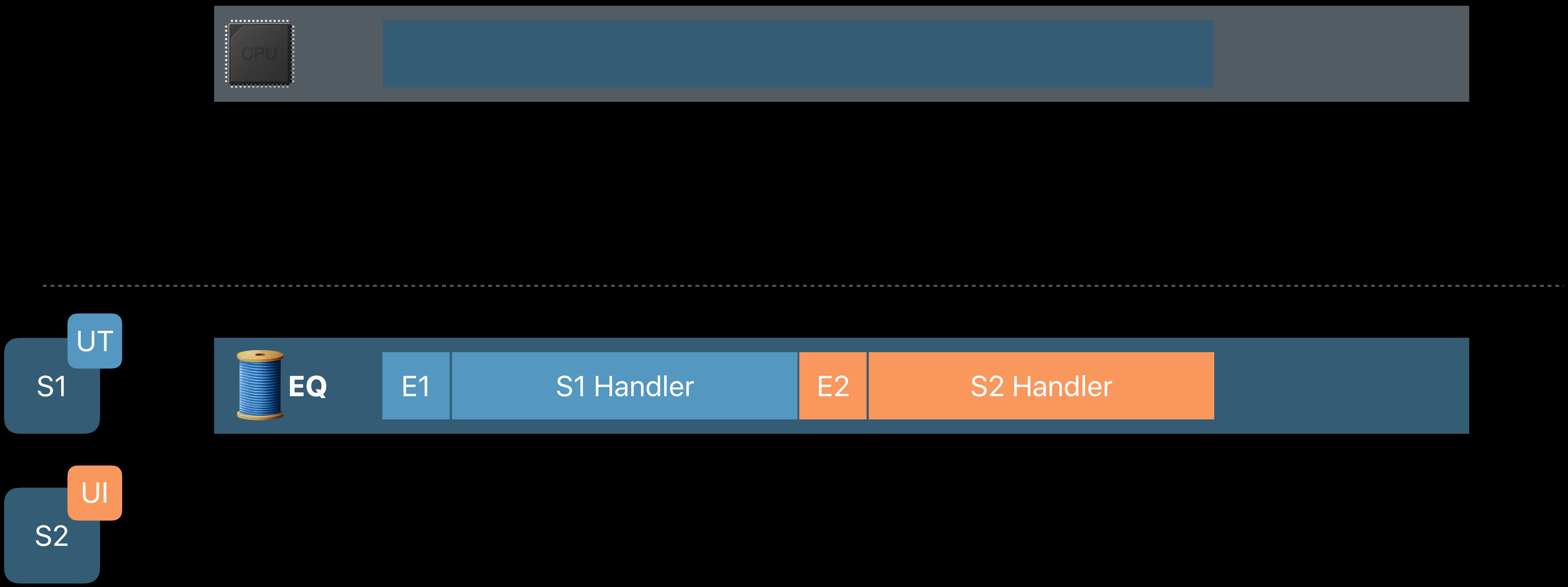


Leveraging Ownership and Unified Identity

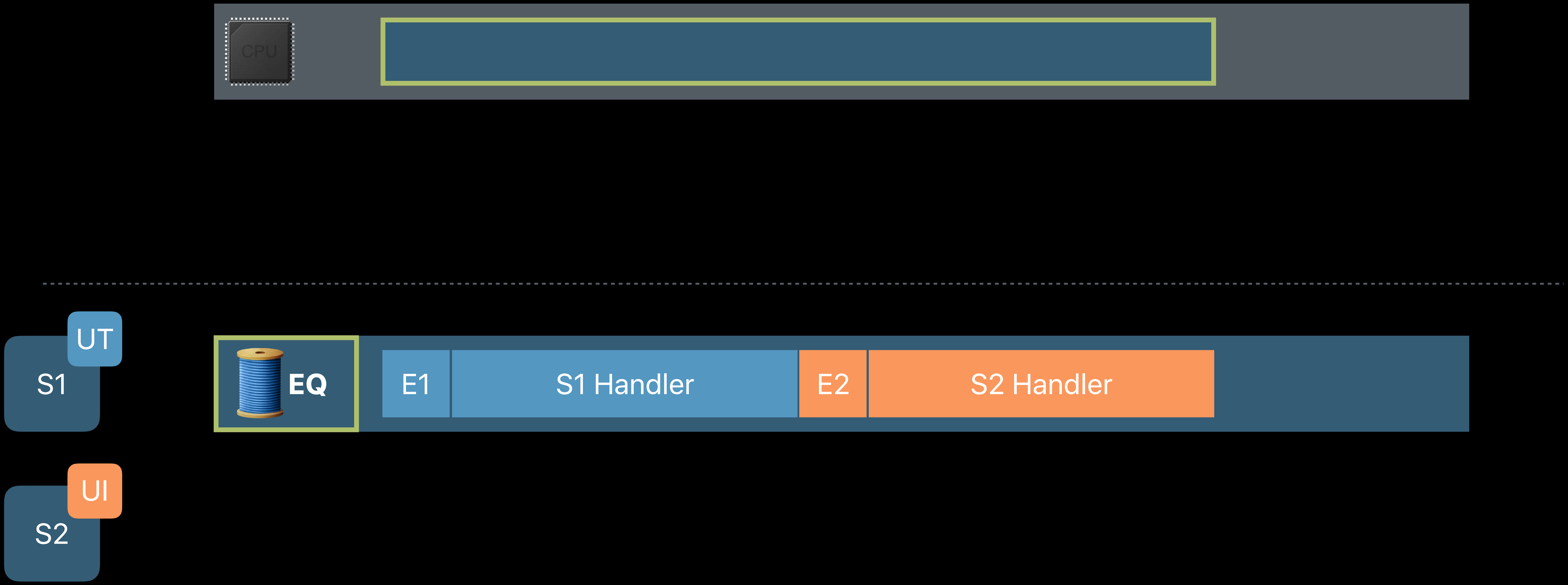


Leveraging Ownership and Unified Identity

NEW

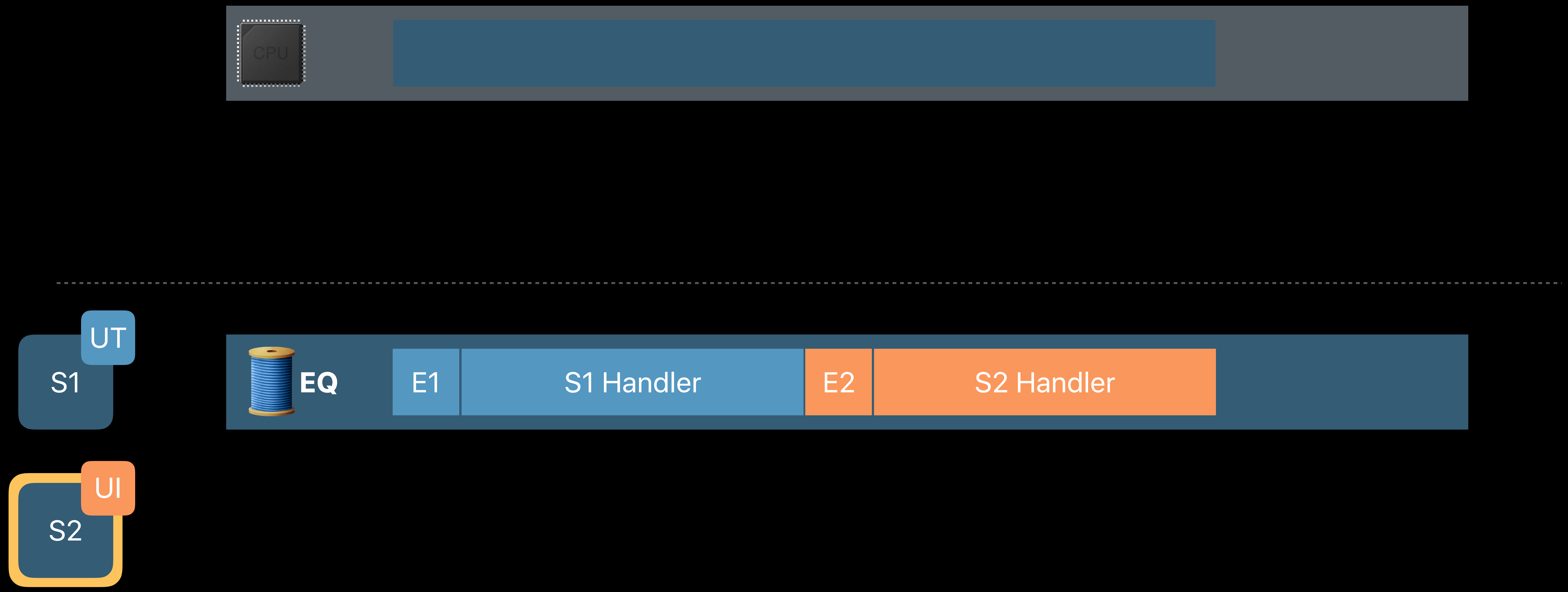


Leveraging Ownership and Unified Identity



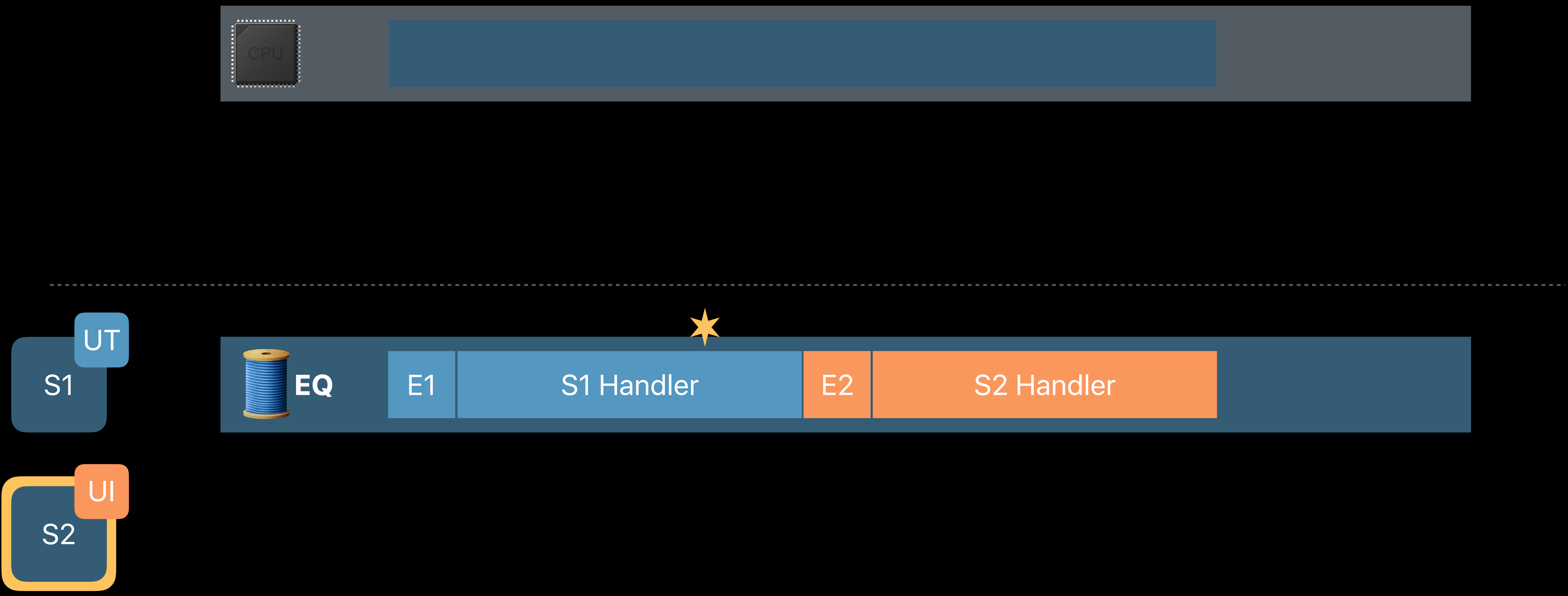
Leveraging Ownership and Unified Identity

NEW



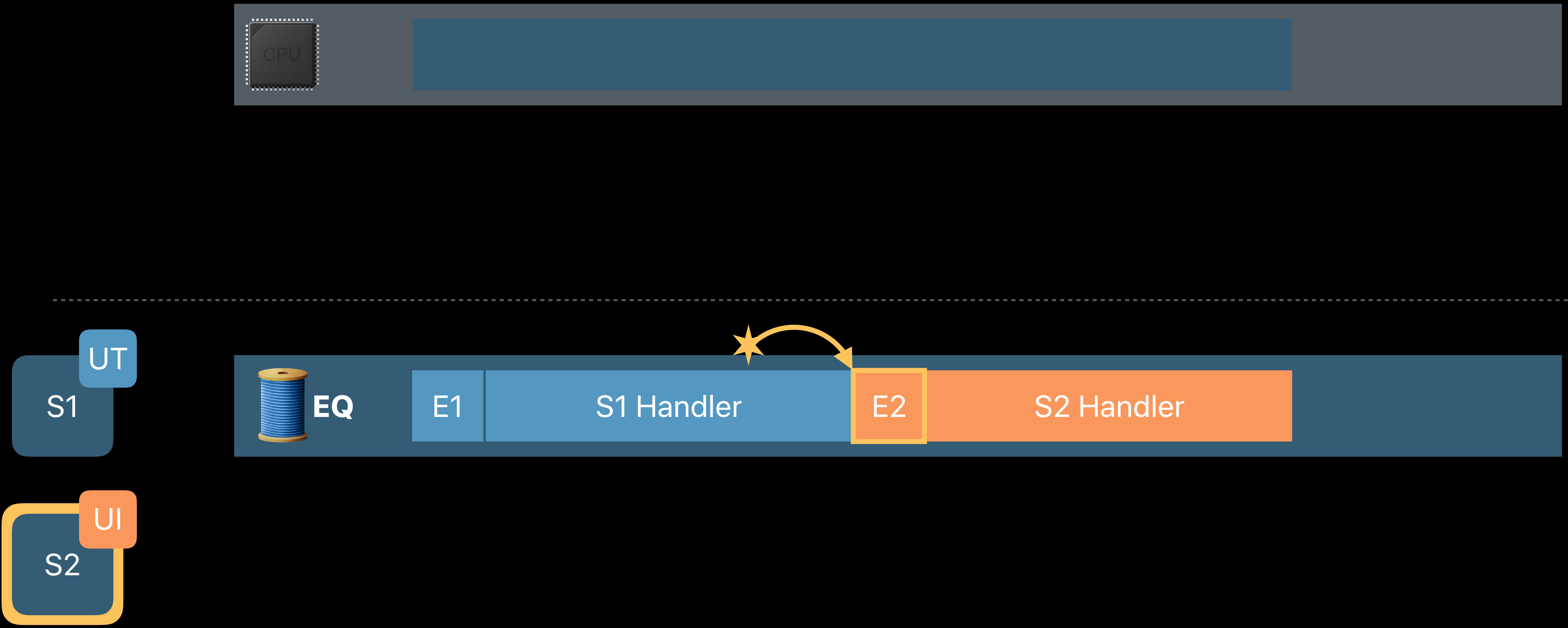
Leveraging Ownership and Unified Identity

NEW



Leveraging Ownership and Unified Identity

NEW



The runtime uses every possible hint to optimize behavior

Modernizing Existing Code

Modernizing Existing Code

No dispatch object mutation after activation

Protect your target queue hierarchy

No Mutation Past Activation

Set the properties of inactive objects before activation

- Source handlers
- Target queues

No Mutation Past Activation

Set the properties of inactive objects before activation

- Source handlers
- Target queues

```swift
let mySource = DispatchSource.makeReadSource(fileDescriptor: fd, queue: myQueue)
```

No Mutation Past Activation

Set the properties of inactive objects before activation

- Source handlers
- Target queues

```
let mySource = DispatchSource.makeReadSource(fileDescriptor: fd, queue: myQueue)

mySource.setEventHandler(qos: .userInteractive) { … }
mySource.setCancelHandler { close(fd) }
```

No Mutation Past Activation

✓

Set the properties of inactive objects before activation

- Source handlers
- Target queues

```
let mySource = DispatchSource.makeReadSource(fileDescriptor: fd, queue: myQueue)

mySource.setEventHandler(qos: .userInteractive) { … }
mySource.setCancelHandler { close(fd) }

mySource.activate()
```

No Mutation Past Activation

Set the properties of inactive objects before activation

- Source handlers
- Target queues

```
let mySource = DispatchSource.makeReadSource(fileDescriptor: fd, queue: myQueue)

mySource.setEventHandler(qos: .userInteractive) { … }
mySource.setCancelHandler { close(fd) }

mySource.activate()

mySource.setTarget(queue: otherQueue)
```

Effects of Queue Graph Mutation

Priority and ownership snapshots can become stale

- Defeats priority inversion avoidance
- Defeats direct handoff optimization
- Defeats event delivery optimization

Effects of Queue Graph Mutation

Priority and ownership snapshots can become stale

- Defeats priority inversion avoidance
- Defeats direct handoff optimization
- Defeats event delivery optimization

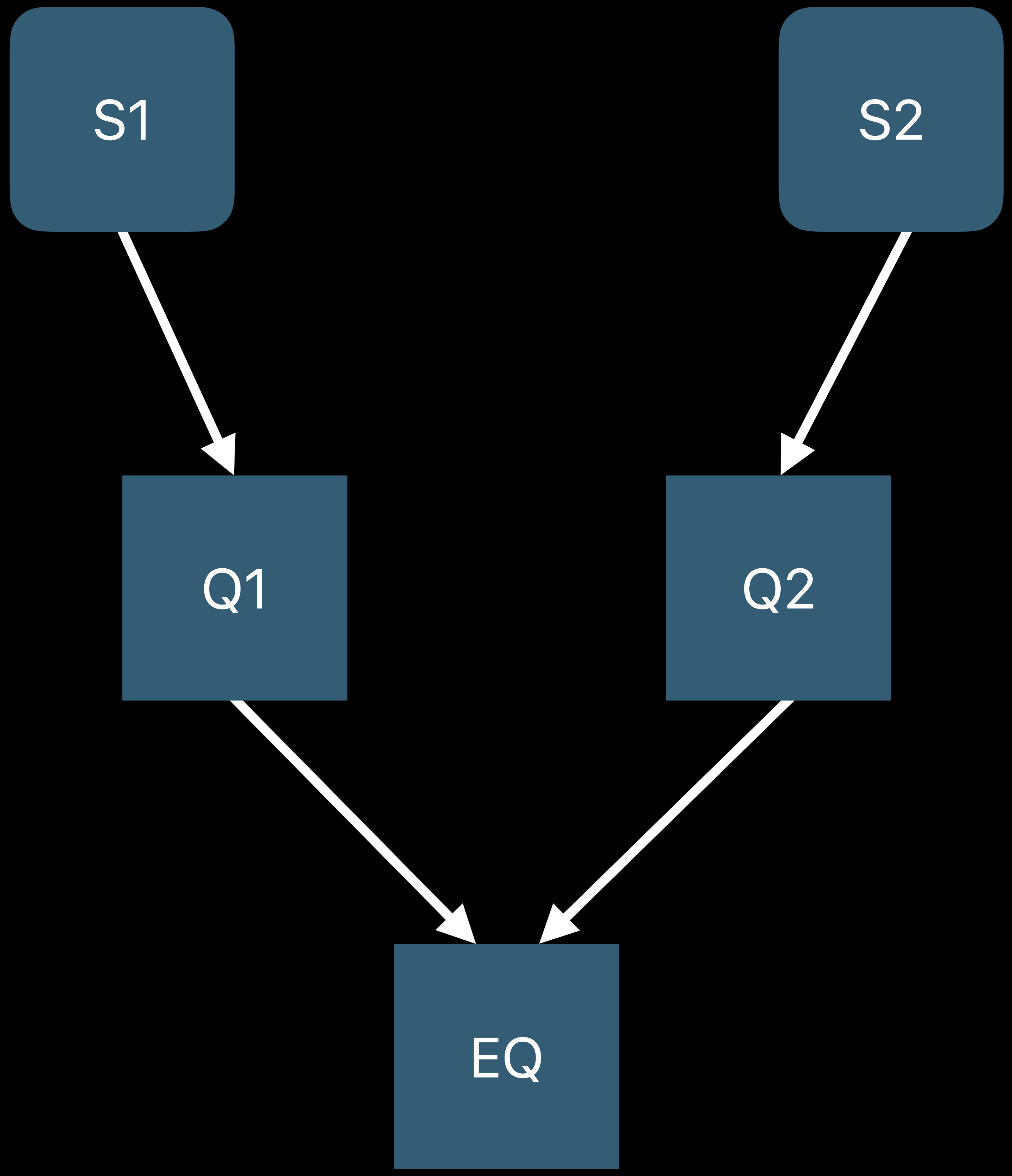
System frameworks may create sources on your behalf

- XPC connections are like sources

Protecting the Target Queue Hierarchy

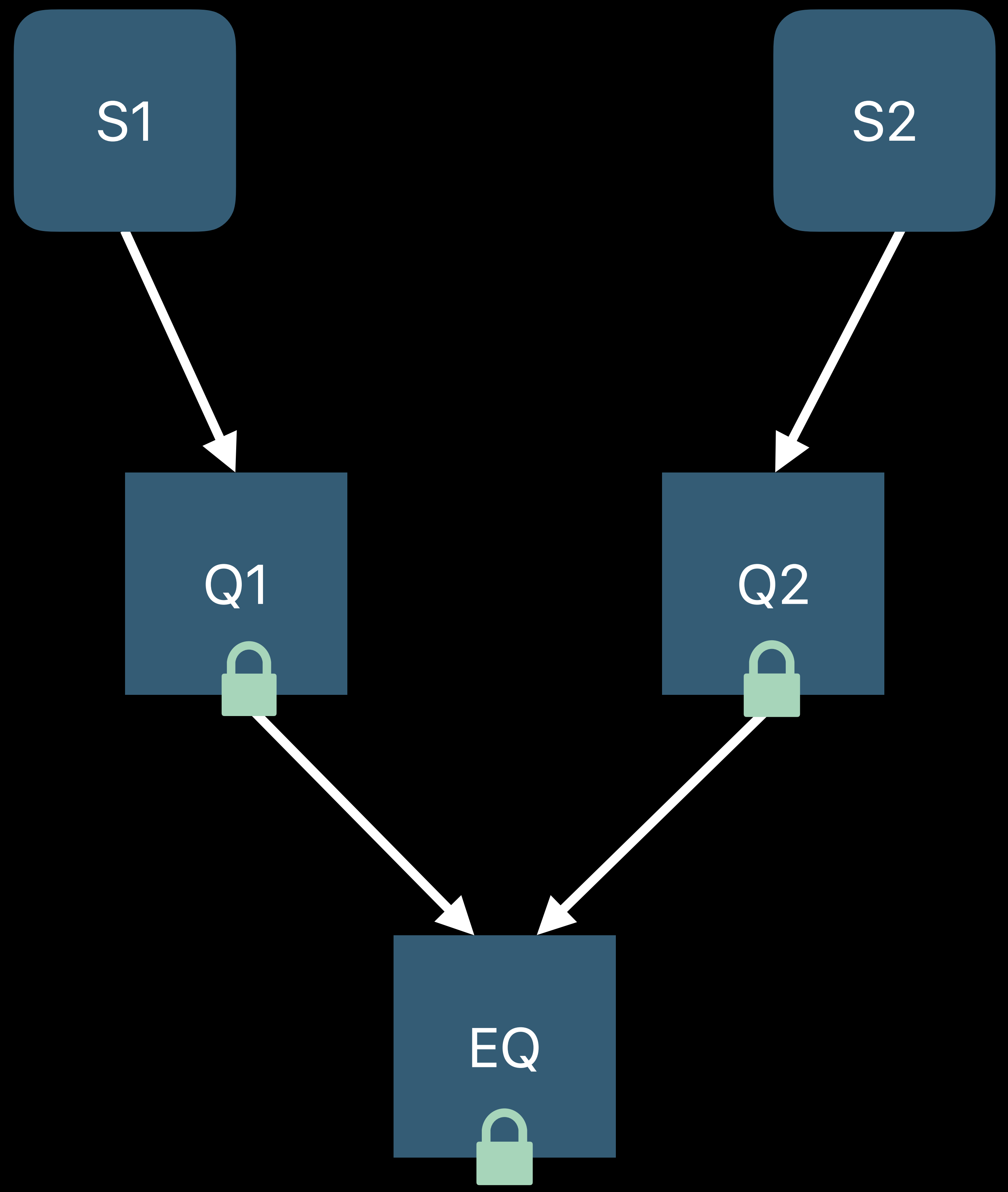
Protecting the Target Queue Hierarchy

Build your queue hierarchy bottom to top



Protecting the Target Queue Hierarchy

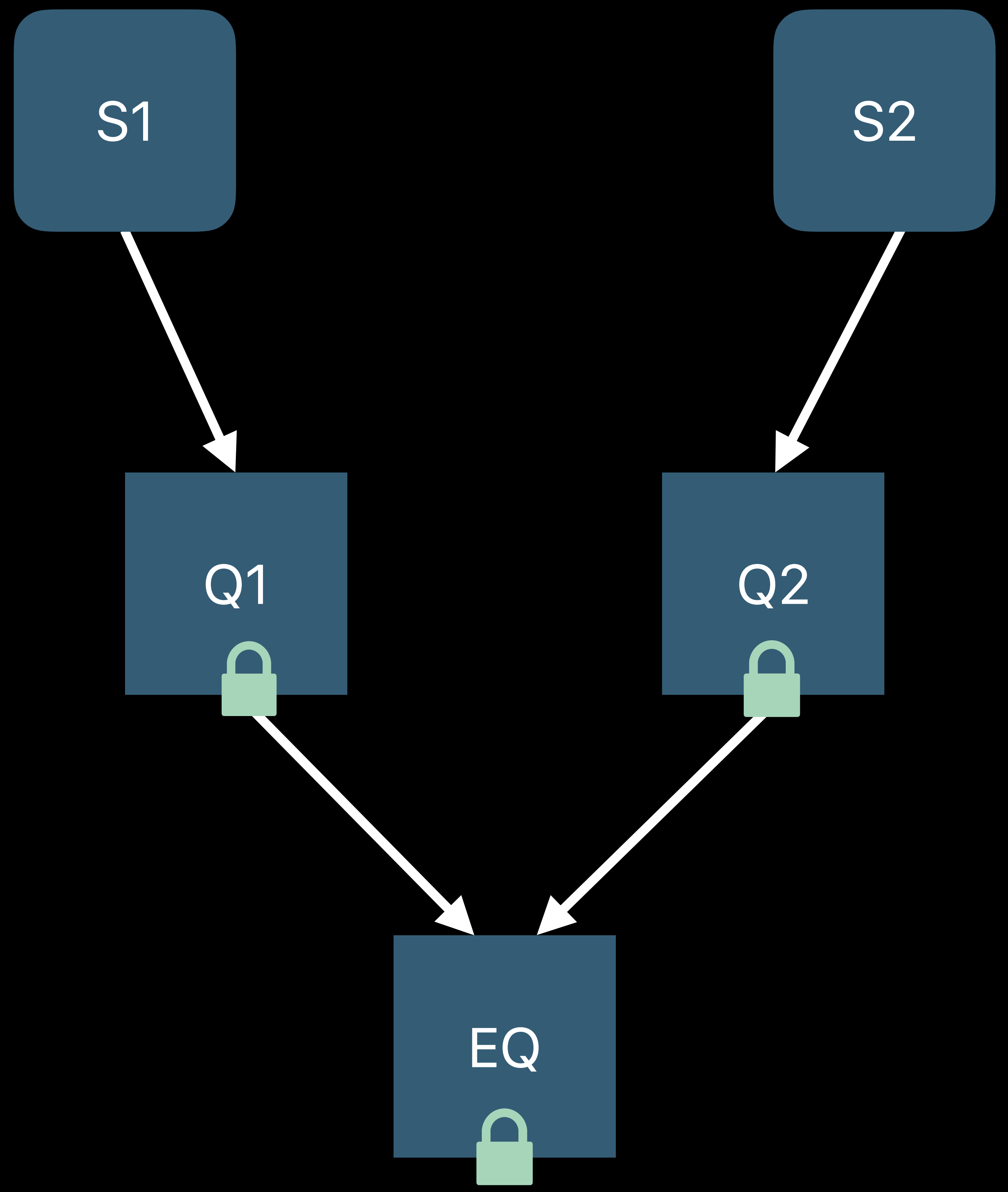
Build your queue hierarchy bottom to top



Protecting the Target Queue Hierarchy

Build your queue hierarchy bottom to top

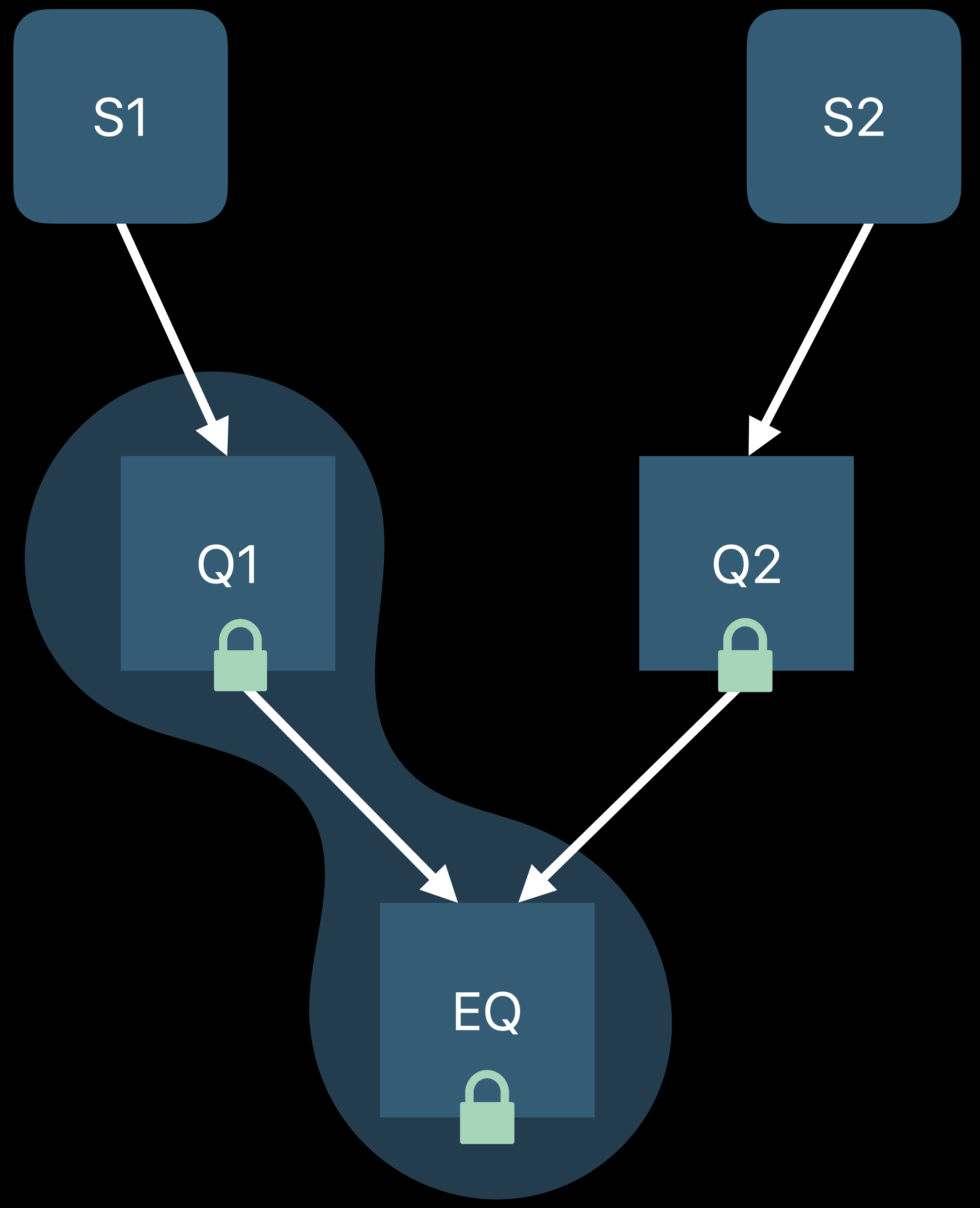
Opt into "static queue hierarchy"



Protecting the Target Queue Hierarchy

Build your queue hierarchy bottom to top

Opt into “static queue hierarchy”

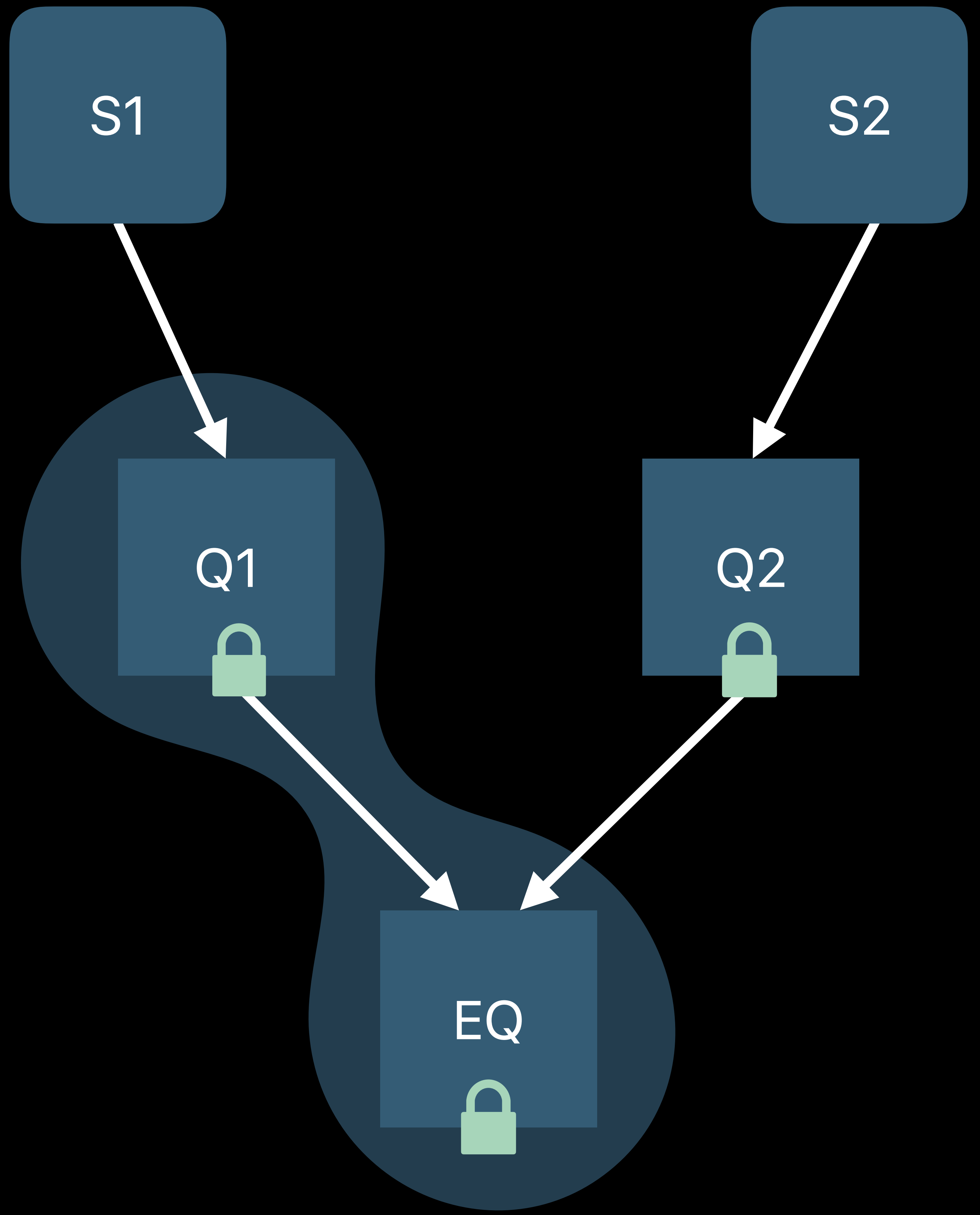


Protecting the Target Queue Hierarchy

Build your queue hierarchy bottom to top

Opt into “static queue hierarchy”

```
Q1 = dispatch_queue_create("Q1",
         DISPATCH_QUEUE_SERIAL)
dispatch_set_target_queue(Q1, EQ)
```



Protecting the Target Queue Hierarchy

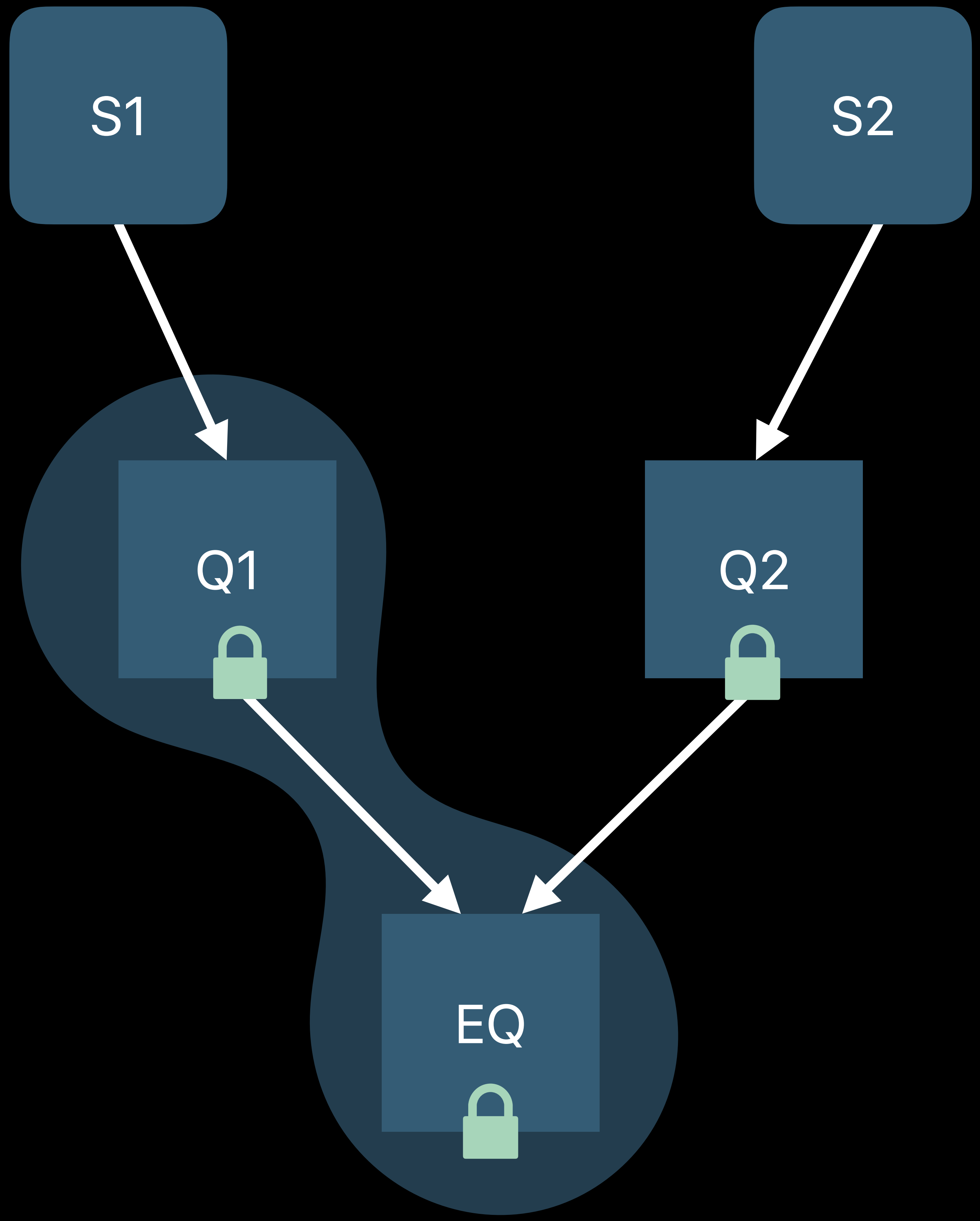
Build your queue hierarchy bottom to top

Opt into “static queue hierarchy”

```
Q1 = dispatch_queue_create("Q1",
         DISPATCH_QUEUE_SERIAL)
dispatch_set_target_queue(Q1, EQ)
```

```
Q1 = dispatch_queue_create_with_target("Q1",
         DISPATCH_QUEUE_SERIAL, EQ)
```

✓


$$\begin{tikzcd}
S1 \arrow[d] & & S2 \arrow[d] \\
Q1 \arrow[dr] & & Q2 \arrow[dl] \\
& EQ &
\end{tikzcd}$$

Demo

Finding problem spots

Daniel A. Steffen, Core Darwin

MyNews
MyNews
ViewController.m
-server

```
- (void)createConnections:(int) numberOfConnections serverPort:(int)port
{
    struct sockaddr_in serverAddr = [self server];
    conns = (struct client_connection *)malloc
            (numberOfConnections * sizeof(struct client_connection));

    for (int i = 0; i < numberOfConnections; i++) {
        int sock = socket(PF_INET, SOCK_STREAM, 0);
        int ret = connect(sock, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
        assert(ret >= 0);

        int flags = fcntl(sock, F_GETFL, 0);
        fcntl(sock, F_SETFL, flags | O_NONBLOCK);

        char queue_name[1024];
        snprintf(queue_name, 1024, "com.apple.client-queue-%d", i);
        dispatch_queue_t queue = dispatch_queue_create(queue_name, DISPATCH_QUEUE_SERIAL);

        dispatch_source_t source = dispatch_source_create(DISPATCH_SOURCE_TYPE_READ, sock, 0, NULL);

        dispatch_block_t block = dispatch_block_create(DISPATCH_BLOCK_ASSIGN_CURRENT, ^{
            /* Drop the data read block start signpost */
            kdebug_signpost_start(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);

            /* Re-initialize the buffer for the connection */
```

MyNews 〉 MyNews 〉 ViewController.m 〉 -server

```
- (void)createConnections:(int) numberOfConnections serverPort:(int)port
{
    struct sockaddr_in serverAddr = [self server];
    conns = (struct client_connection *)malloc
            (numberOfConnections * sizeof(struct client_connection));

    for (int i = 0; i < numberOfConnections; i++) {
        int sock = socket(PF_INET, SOCK_STREAM, 0);
        int ret = connect(sock, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
        assert(ret >= 0);

        int flags = fcntl(sock, F_GETFL, 0);
        fcntl(sock, F_SETFL, flags | O_NONBLOCK);

        char queue_name[1024];
        snprintf(queue_name, 1024, "com.apple.client-queue-%d", i);
        dispatch_queue_t queue = dispatch_queue_create(queue_name, DISPATCH_QUEUE_SERIAL);

        dispatch_source_t source = dispatch_source_create(DISPATCH_SOURCE_TYPE_READ, sock, 0, NULL);

        dispatch_block_t block = dispatch_block_create(DISPATCH_BLOCK_ASSIGN_CURRENT, ^{
            /* Drop the data read block start signpost */
            kdebug_signpost_start(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);

            /* Re-initialize the buffer for the connection */
```

MyNews
MyNews
ViewController.m
-createConnections:serverPort:

```
                if (err == 0 || (err < 0 && errno != EAGAIN && errno != EINTR)) {
                    dispatch_source_cancel(source);
                    break;
                }
                if (err < 0 && errno == EAGAIN) {
                    break;
                }
                conns[i].index += err;
            }

            /* Add URL to global Set */
            [self processURL:conns[i].buffer];

            /* Drop the data read block end signpost */
            kdebug_signpost_end(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);
        });

        dispatch_activate(source);

        dispatch_source_set_event_handler(source, block);
        dispatch_source_set_cancel_handler(source, ^{
            close(sock);
        });
        dispatch_set_target_queue(source, queue);
    }
}
```

MyNews 〉 MyNews 〉 ViewController.m 〉 -createConnections:serverPort:

```
                if (err == 0 || (err < 0 && errno != EAGAIN && errno != EINTR)) {
                    dispatch_source_cancel(source);
                    break;
                }
                if (err < 0 && errno == EAGAIN) {
                    break;
                }
                conns[i].index += err;
            }

            /* Add URL to global Set */
            [self processURL:conns[i].buffer];

            /* Drop the data read block end signpost */
            kdebug_signpost_end(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);
        });

        dispatch_activate(source);

        dispatch_source_set_event_handler(source, block);
        dispatch_source_set_cancel_handler(source, ^{
            close(sock);
        });
        dispatch_set_target_queue(source, queue);
    }
}
```

MyNews 〉 MyNews 〉 ViewController.m 〉 -server

```
- (void)createConnections:(int) numberOfConnections serverPort:(int)port
{
    struct sockaddr_in serverAddr = [self server];
    conns = (struct client_connection *)malloc
            (numberOfConnections * sizeof(struct client_connection));

    for (int i = 0; i < numberOfConnections; i++) {
        int sock = socket(PF_INET, SOCK_STREAM, 0);
        int ret = connect(sock, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
        assert(ret >= 0);

        int flags = fcntl(sock, F_GETFL, 0);
        fcntl(sock, F_SETFL, flags | O_NONBLOCK);

        char queue_name[1024];
        snprintf(queue_name, 1024, "com.apple.client-queue-%d", i);
        dispatch_queue_t queue = dispatch_queue_create(queue_name, DISPATCH_QUEUE_SERIAL);

        dispatch_source_t source = dispatch_source_create(DISPATCH_SOURCE_TYPE_READ, sock, 0, NULL);

        dispatch_block_t block = dispatch_block_create(DISPATCH_BLOCK_ASSIGN_CURRENT, ^{
            /* Drop the data read block start signpost */
            kdebug_signpost_start(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);

            /* Re-initialize the buffer for the connection */
```

MyNews 〉 MyNews 〉 ViewController.m 〉 -server

```
- (void)createConnections:(int) numberOfConnections serverPort:(int)port
{
    struct sockaddr_in serverAddr = [self server];
    conns = (struct client_connection *)malloc
            (numberOfConnections * sizeof(struct client_connection));

    for (int i = 0; i < numberOfConnections; i++) {
        int sock = socket(PF_INET, SOCK_STREAM, 0);
        int ret = connect(sock, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
        assert(ret >= 0);

        int flags = fcntl(sock, F_GETFL, 0);
        fcntl(sock, F_SETFL, flags | O_NONBLOCK);

        char queue_name[1024];
        snprintf(queue_name, 1024, "com.apple.client-queue-%d", i);
        dispatch_queue_t queue = dispatch_queue_create(queue_name, DISPATCH_QUEUE_SERIAL);

        dispatch_source_t source = dispatch_source_create(DISPATCH_SOURCE_TYPE_READ, sock, 0, NULL);

        dispatch_block_t block = dispatch_block_create(DISPATCH_BLOCK_ASSIGN_CURRENT, ^{
            /* Drop the data read block start signpost */
            kdebug_signpost_start(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);

            /* Re-initialize the buffer for the connection */
```

MyNews 〉 MyNews 〉 ViewController.m 〉 -createConnections:serverPort:

```
                if (err == 0 || (err < 0 && errno != EAGAIN && errno != EINTR)) {
                    dispatch_source_cancel(source);
                    break;
                }
                if (err < 0 && errno == EAGAIN) {
                    break;
                }
                conns[i].index += err;
            }

            /* Add URL to global Set */
            [self processURL:conns[i].buffer];

            /* Drop the data read block end signpost */
            kdebug_signpost_end(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);
        });

        dispatch_source_set_event_handler(source, block);
        dispatch_source_set_cancel_handler(source, ^{
            close(sock);
        });
        dispatch_set_target_queue(source, queue);

        dispatch_activate(source);
    }
}
```

MyNews
MyNews
ViewController.m
-createConnections:serverPort:

```
                if (err == 0 || (err < 0 && errno != EAGAIN && errno != EINTR)) {
                    dispatch_source_cancel(source);
                    break;
                }
                if (err < 0 && errno == EAGAIN) {
                    break;
                }
                conns[i].index += err;
            }

            /* Add URL to global Set */
            [self processURL:conns[i].buffer];

            /* Drop the data read block end signpost */
            kdebug_signpost_end(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);
        });

        dispatch_source_set_event_handler(source, block);
        dispatch_source_set_cancel_handler(source, ^{
            close(sock);
        });
        dispatch_set_target_queue(source, queue);

        dispatch_activate(source);
    }
}
```

MyNews 〉 MyNews 〉 ViewController.m 〉 -createConnections:serverPort:

```
- (void)createConnections:(int) numberOfConnections serverPort:(int)port
{
    struct sockaddr_in serverAddr = [self server];
    conns = (struct client_connection *)malloc
            (numberOfConnections * sizeof(struct client_connection));

    for (int i = 0; i < numberOfConnections; i++) {
        int sock = socket(PF_INET, SOCK_STREAM, 0);
        int ret = connect(sock, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
        assert(ret >= 0);

        int flags = fcntl(sock, F_GETFL, 0);
        fcntl(sock, F_SETFL, flags | O_NONBLOCK);

        char queue_name[1024];
        snprintf(queue_name, 1024, "com.apple.client-queue-%d", i);
        dispatch_queue_t queue = dispatch_queue_create(queue_name, DISPATCH_QUEUE_SERIAL);

        dispatch_source_t source = dispatch_source_create(DISPATCH_SOURCE_TYPE_READ, sock, 0, NULL);

        dispatch_block_t block = dispatch_block_create(DISPATCH_BLOCK_ASSIGN_CURRENT, ^{
            /* Drop the data read block start signpost */
            kdebug_signpost_start(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);

            /* Re-initialize the buffer for the connection */
            bzero(&conns[i].buffer, CONNECTION_BUFFER_SIZE);
```

MyNews
MyNews
ViewController.m
-createConnections:serverPort:

```
- (void)createConnections:(int) numberOfConnections serverPort:(int)port
{
    struct sockaddr_in serverAddr = [self server];
    conns = (struct client_connection *)malloc
            (numberOfConnections * sizeof(struct client_connection));

    for (int i = 0; i < numberOfConnections; i++) {
        int sock = socket(PF_INET, SOCK_STREAM, 0);
        int ret = connect(sock, (struct sockaddr *) &serverAddr, sizeof(serverAddr));
        assert(ret >= 0);

        int flags = fcntl(sock, F_GETFL, 0);
        fcntl(sock, F_SETFL, flags | O_NONBLOCK);

        char queue_name[1024];
        snprintf(queue_name, 1024, "com.apple.client-queue-%d", i);
        dispatch_queue_t queue = dispatch_queue_create(queue_name, DISPATCH_QUEUE_SERIAL);

        dispatch_source_t source = dispatch_source_create(DISPATCH_SOURCE_TYPE_READ, sock, 0, NULL);

        dispatch_block_t block = dispatch_block_create(DISPATCH_BLOCK_ASSIGN_CURRENT, ^{
            /* Drop the data read block start signpost */
            kdebug_signpost_start(MYNEWS_CONN_DATA_RECV, i, sock, 0, 0);

            /* Re-initialize the buffer for the connection */
            bzero(&conns[i].buffer, CONNECTION_BUFFER_SIZE);
```

Demo

Finding problem spots

Daniel A. Steffen, Core Darwin

Summary

Not going off-core is ever more important

Size your work appropriately

Choose good granularity of concurrency

Modernize your GCD usage

Use tools to find problem spots

More Information

https://developer.apple.com/wwdc17/706

Related Sessions

Introducing Core ML		WWDC 2017
Accelerate and Sparse Solvers	Grand Ballroom A	Thursday 10:00AM
Using Metal 2 for Compute	Grand Ballroom A	Thursday 4:10PM
Writing Energy Efficient Apps	Executive Ballroom	Friday 9:00AM
App Startup Time: Past, Present, and Future	Hall 2	Friday 10:00AM

Labs

Kernel & Runtime Lab	Technology Lab D	Wed 1:50PM–4:10PM
Kernel & Runtime Lab	Technology Lab J	Thu 10:00AM–12:00PM
Performance Profiling and Runtime Analysis Tools Lab	Technology Lab K	Thu 1:00PM–4:10PM
Optimizing App Startup Time Lab	Technology Lab E	Fri 11:00AM–12:30PM

WWDC17