

Classification of Thread Based Async Models

Threading Model

The Registered Callback or Delegate Model

The Threads with Data Sharing Model

The Threaded WorkQueue Model

The Async Expression Model

Model Examples

NSNotification based services such as Location/Orientation

Delegates

Separate threads with shared data employing locking and signal events for notification.

GCD Dispatch Queues

NSOperation

Reactive Extensions

RxPatterns

Comparison Of Async Models (out of 10, higher is better)

Capability	Ease of Use	Registered Callback	Threads with Data Sharing	Threaded WorkQueue	Async Expression
Multiple Consumers		9/10	2/10	2/10	10/10
Operators		0/10	0/10	0/10	10/10
Error Handling		2/10	1/10	1/10	10/10
Completion Handling		2/10	1/10	1/10	10/10
Synchronisation		2/10	2/10	10/10	10/10
Custom Producers		2/10	2/10	2/10	10/10
Custom Consumers		10/10	9/10	10/10	10/10
	Operational Simplicity	10/10	4/10	5/10	2/10
	Ease of Tool-up	10/10	8/10	9/10	1/10

Process/Distributed Async Models

Process Async Model

The CoProcess with Messaging Model

The Remote Procedure Call Model

Map Reduce Model

Model Examples

Darwin processes with Mach ports

Soap and various RCP mechanisms

Hadoop

A

Process/System Async Models are Different to Thread Models

Thread models share the same memory space, others do not.

Process/System models require data to be serialised across boundaries.

Variable capture is available to threaded async models only.

Terminology For Classifying Async Models

Concept Type	Concept	Sub Feature
Transport Entities	Notification	Data Item Notification Error Notification Completion Notification
Control	Consumer Producer Operator Notifier	
Execution and Synchronisation	EvalQueue Subscription	

The Notification Concept

The Notification Concept

Represents a notification of an event

There are different types of events:

Data Associated with the Event

The event may be a notification of an error

The event may be a notification of the completion of events.

Events are produced and consumed

Events may be operated on.

Notification Examples

NSNotification

A data parameter of an event callback handler

Touch Events

The Notification Operator Concept

The Operator Concept

Basically receives Notifications and emits Notifications.

Operators are composable with other operators.

Operators participate in subscription,

Depending on behaviour may do something with received notifications.

Operators may operate on notifications by emitting a different notification to the one received.

A

Operator Examples

Lambda operators such as zip.

take(count: UInt) - emit count item notifications.

skip(count: UInt) - skip the first count item notifications and emit the remaining item notifications.

map(mapFunc: (ItemType) -> ItemType?) - map item notifications.

The Notifier Concept

The Notifier Concept

Consume Notifications.

They are a light-weight means of doing something with a notification.

They typically forward notifications to some target(s) (i.e. perform the act of notifying)

Notifiers are not formal operators as they are not composable.

Not directly concerned with subscription.

They are the work horse of the action of notifying.

Notifier Roles Examples

A Notifier that notifies targets by means of delegate functions. that can be set.

A Notifier that notices a collection of consumers.

A Notifier that queues notifications and then forwards them to targets when the targets are ready to receive (Performs a buffering action).

The Evaluation Queue (EvalQueue) Concept

The EvalQueue Concept

The concept is akin to a dispatch queue in that it has its own work thread(s)

The role is to perform Sync and Async tasks that are submitted to the EvalQueue for execution.

Provides synchronisation for operations running in the same evalqueue.

Obviates the need for locking (simplifies code and performs faster)

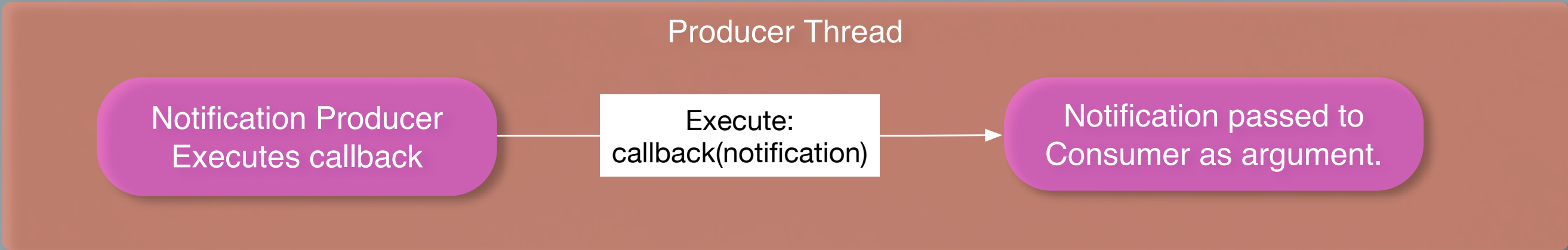
Provides a predictable execution environment that can be optimised and safeguarded against thread dead-lock.

EvalQueue Examples

The OSX/iOS dispatch queues.

Registered Callback Delegate Design

Model



Component Mapping

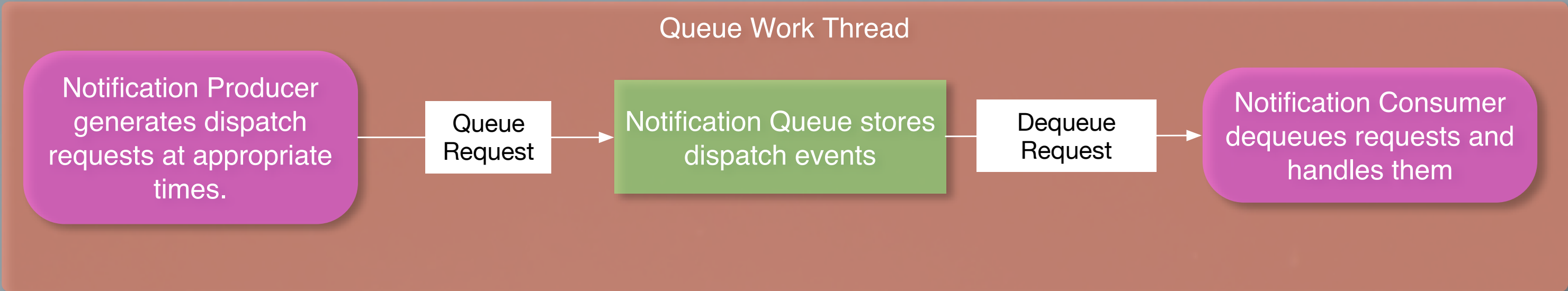


Notes

- Push Notification Design.
- Typically Callback handlers are registered with Producers
- Typically Callback handlers are unregistered
- Notifications may pass error conditions or termination as well as data
- There may be a separate completion callback.

Threaded Work Queue Design

Model



Component Mapping

Generic Mapping

Producer = queues dispatch requests

Consumer = dequeues requests and handles them

Notification = dispatch request

Subscription = set request handler

Eval Queue = dispatch queue

A

GCD Dispatch Queue Mapping

Producer = dispatch source

Consumer = dispatch handler

Notification = dispatch context or none

Subscription = set dispatch handler

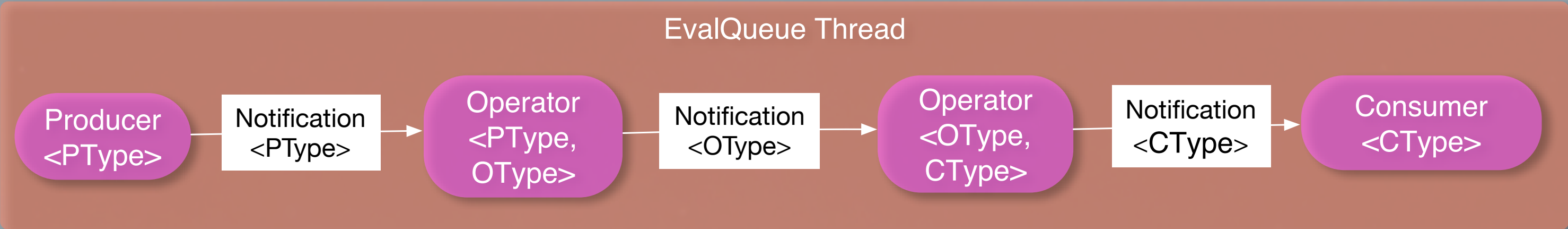
Eval Queue = dispatch queue

Notes

- No operators and decomposition of processing
- Producers and hidden or are not simple to implement
- Notification data is harder to access (not really designed into the model)

The Async Expression Design

Model



Component Mapping

Reactive Extensions Mapping

Producer = Source

Consumer = Observer or delegates

Notification = Data, Error or Completed Events

Eval Queue = None, evaluated in subscription thread or scheduled thread.

RxPatterns Mapping

Producer = RxSource

Consumer = RxObserver, RxNotifier or delegates

Notification = Data or Completed (with optional error) Events

Eval Queue = Every expression runs in it own serial dispatch queue thread.

Operator = RxNotifier<InType> -> RxNotifier<OutType>

Notes

The generic type above represents the notification item data type.