Android Concurrency: The Half-Sync/Half-Async Pattern (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

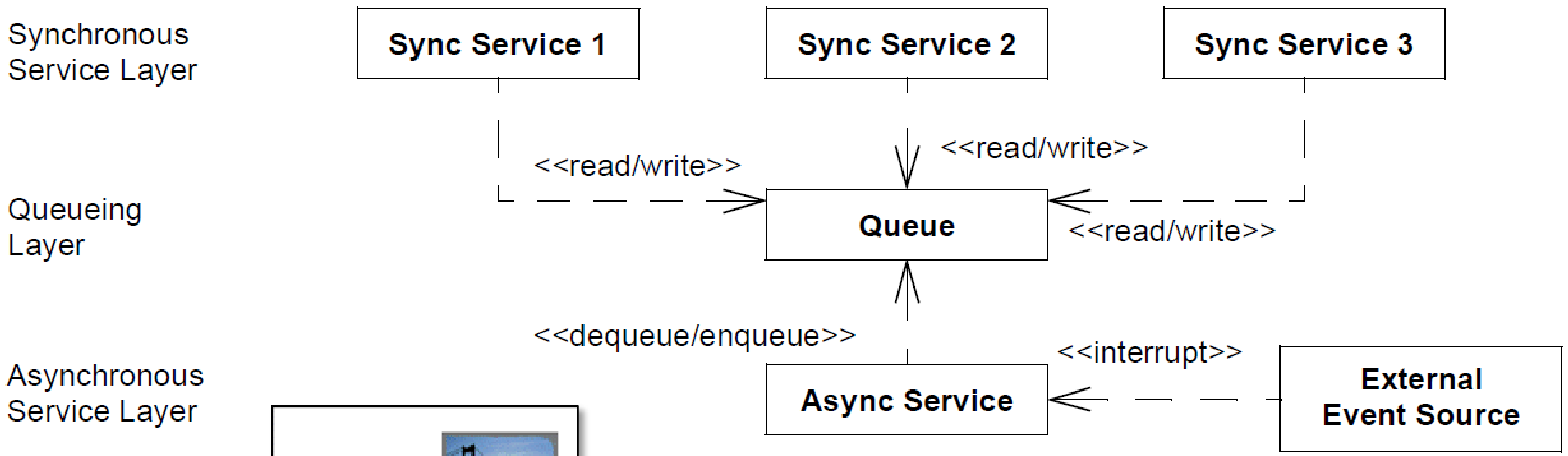
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

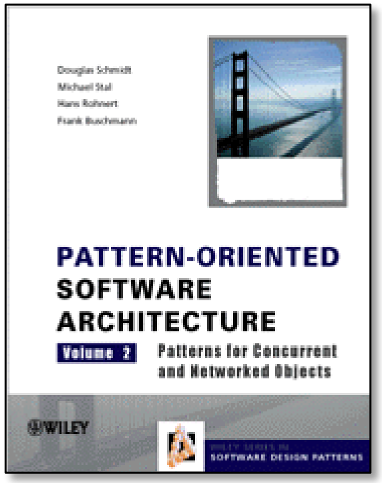
CS 282 Principles of Operating Systems II

Systems Programming for Android

Learning Objectives in this Part of the Module

- Understand the *Half-Sync/Half-Async* pattern




Douglas Schmidt
Michael Stal
Hans Rohnert
Frank Buschmann

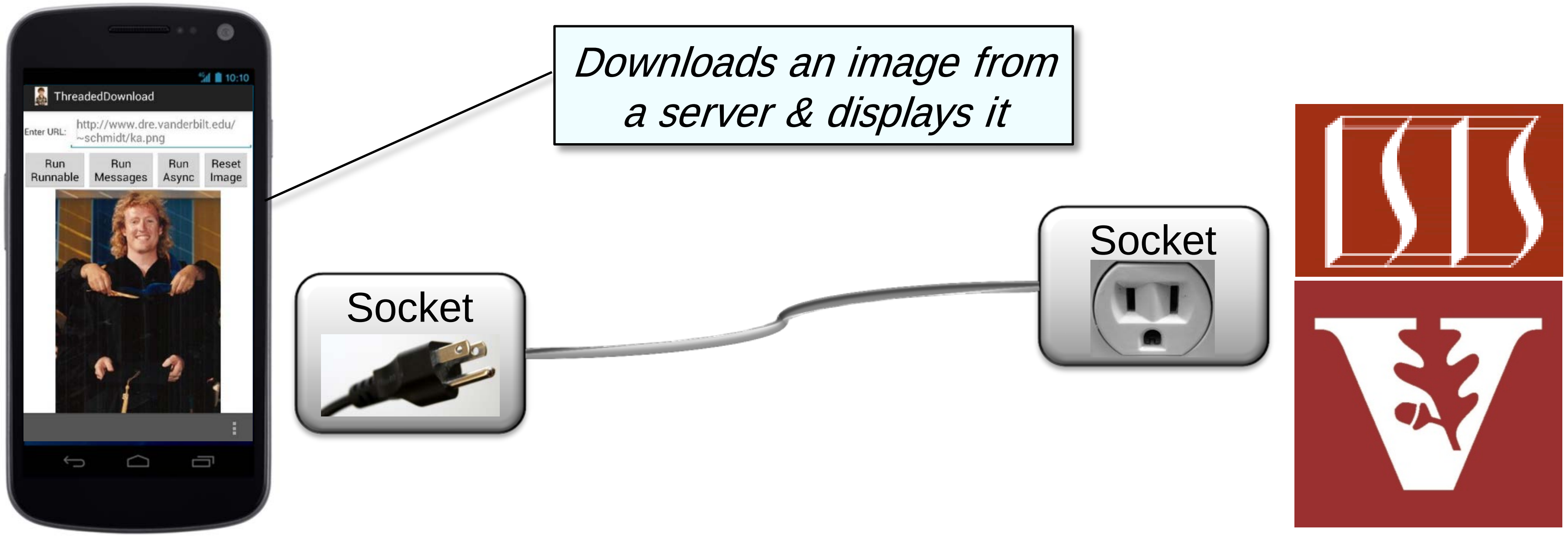
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
Volume 2 Patterns for Concurrent and Networked Objects

WILEY
WILEY SERIES IN SOFTWARE DESIGN PATTERNS

Challenge: Combining Sync & Async Processing

Context

- A concurrent system that performs both asynchronous & synchronous processing services that must communicate
  - The ThreadedDownload app a good example of this context


10:10
ThreadedDownload
Enter URL:
http://www.dre.vanderbilt.edu/~schmidt/ka.png
Run Runnable
Run Messages
Run Async
Reset Image
Downloads an image from a server & displays it
Socket
Socket

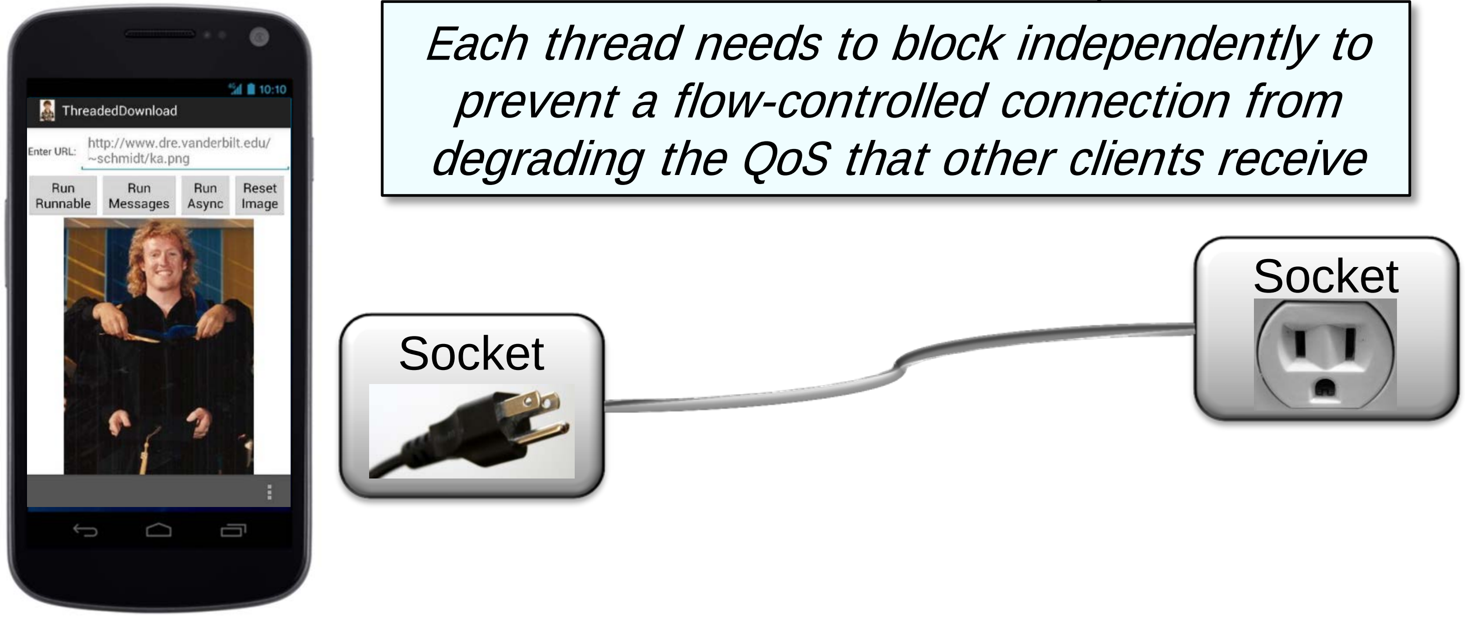
Challenge: Combining Sync & Async Processing

Problems

- Services that want the simplicity of synchronous processing shouldn't need to address the complexities of asynchrony

```
Bitmap downloadBitmap(String url) {
  InputStream is = (InputStream) new URL(url).getContent();
  return BitmapFactory.decodeStream(is);
}
```

Each thread needs to block independently to prevent a flow-controlled connection from degrading the QoS that other clients receive

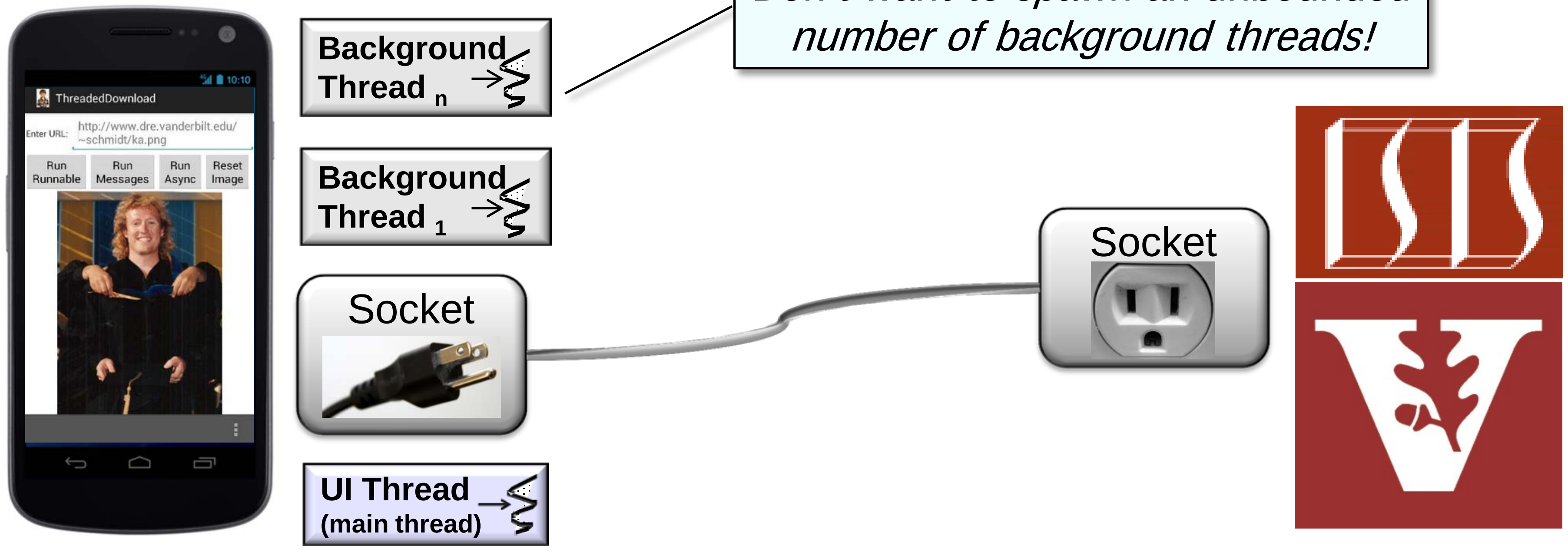


Challenge: Combining Sync & Async Processing

Problems

- Services that want the simplicity of synchronous processing shouldn't need to address the complexities of asynchrony
- Synchronous & asynchronous processing services should be able to communicate without complicating their programming model or unduly degrading their performance

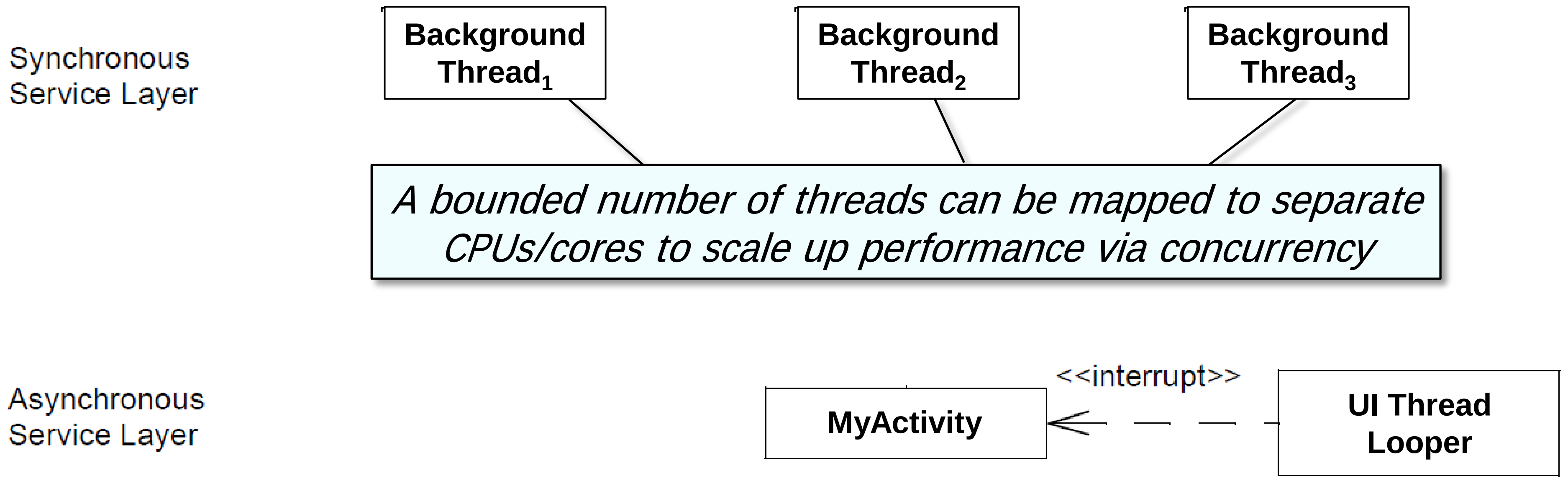
Don't want to spawn an unbounded number of background threads!



Challenge: Combining Sync & Async Processing

Solution

- Decompose the services in the system into two layers: *synchronous* & *asynchronous*

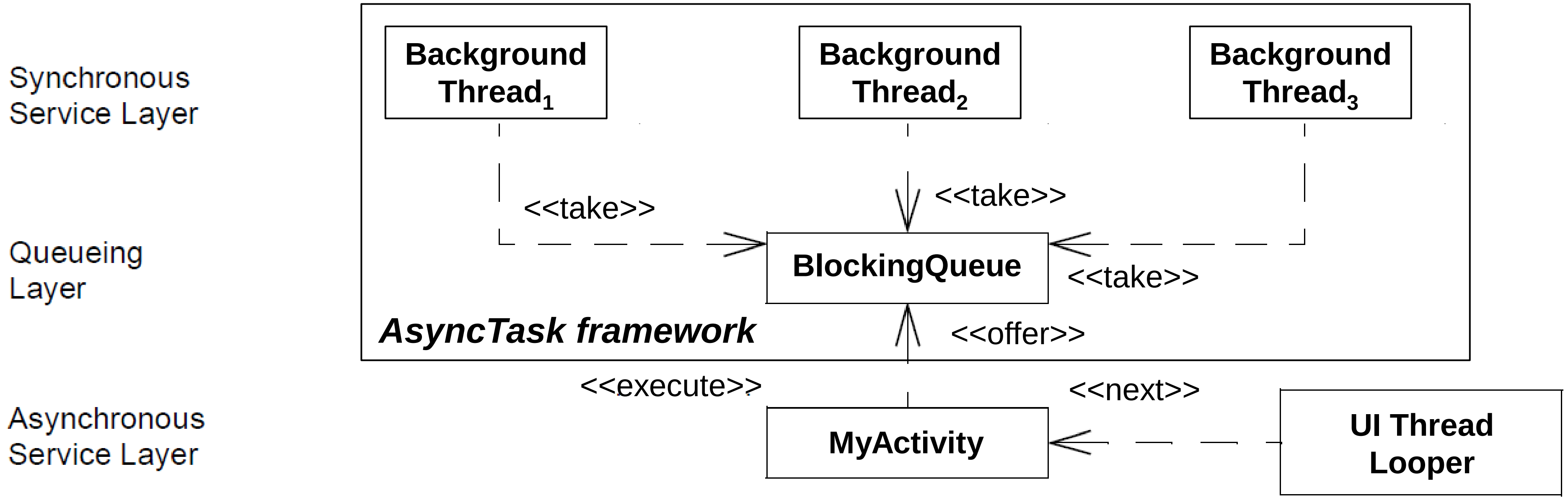


A bounded number of threads can be mapped to separate CPUs/cores to scale up performance via concurrency

Challenge: Combining Sync & Async Processing

Solution

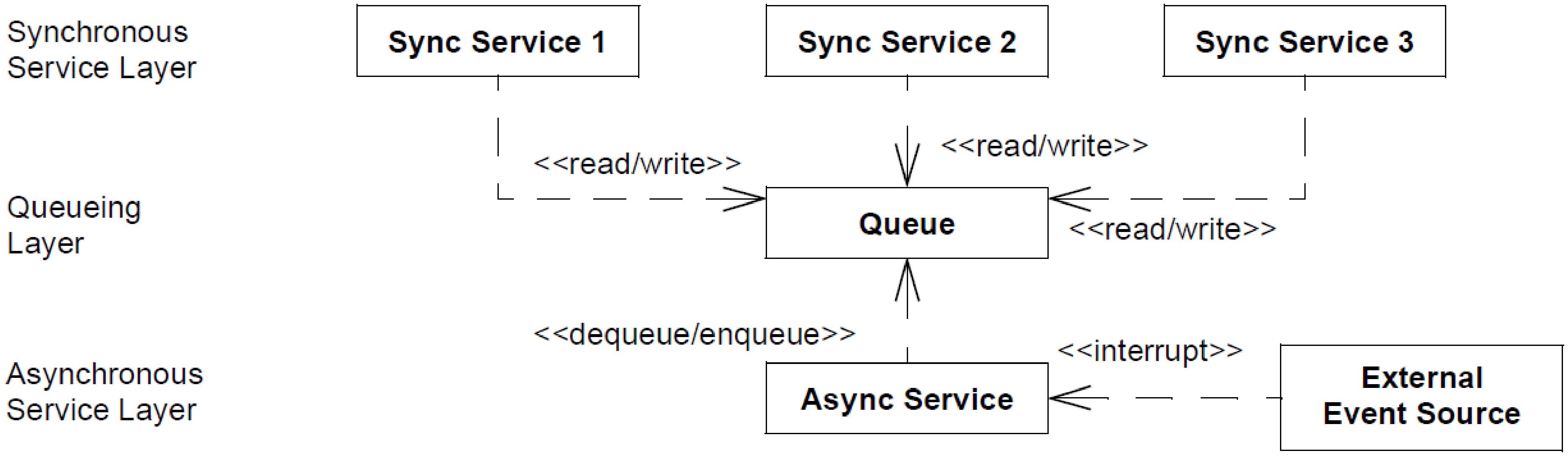
- Decompose the services in the system into two layers: *synchronous* & *asynchronous*
- Add a queueing layer between them to mediate the communication between services in the asynchronous & synchronous layers



Half-Sync/Half-Async POSA2 Concurrency

Intent

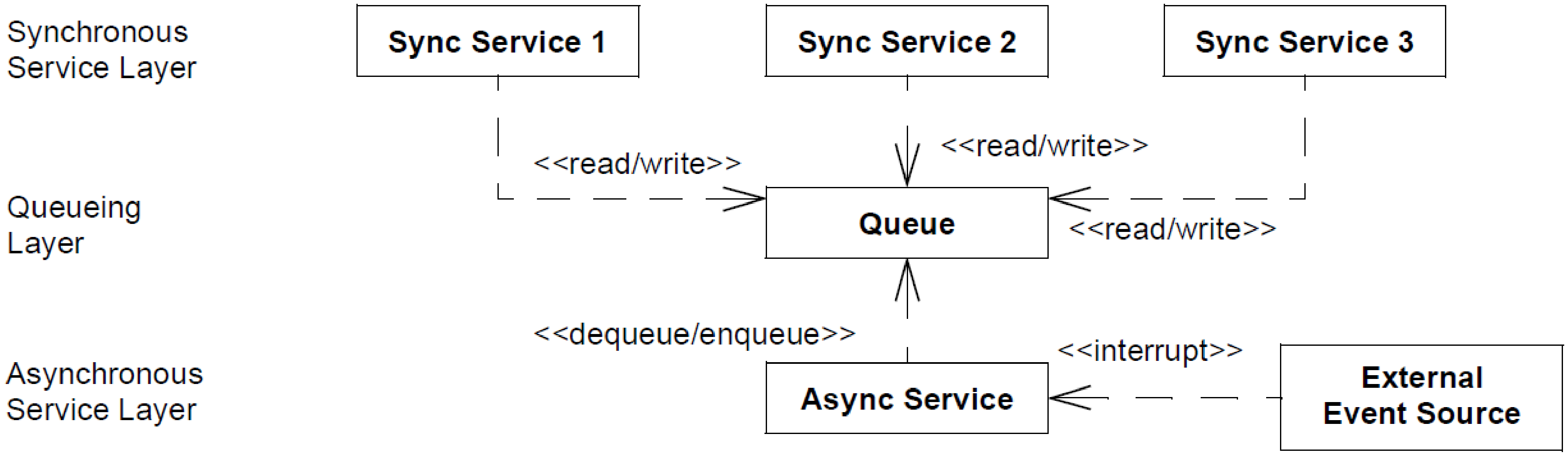
- Decouple asynchronous (async) & synchronous (sync) service processing in concurrent systems by introducing two intercommunicating layers—one for async & one for sync service processing—to simplify programming without unduly reducing performance



Half-Sync/Half-Async POSA2 Concurrency

Applicability

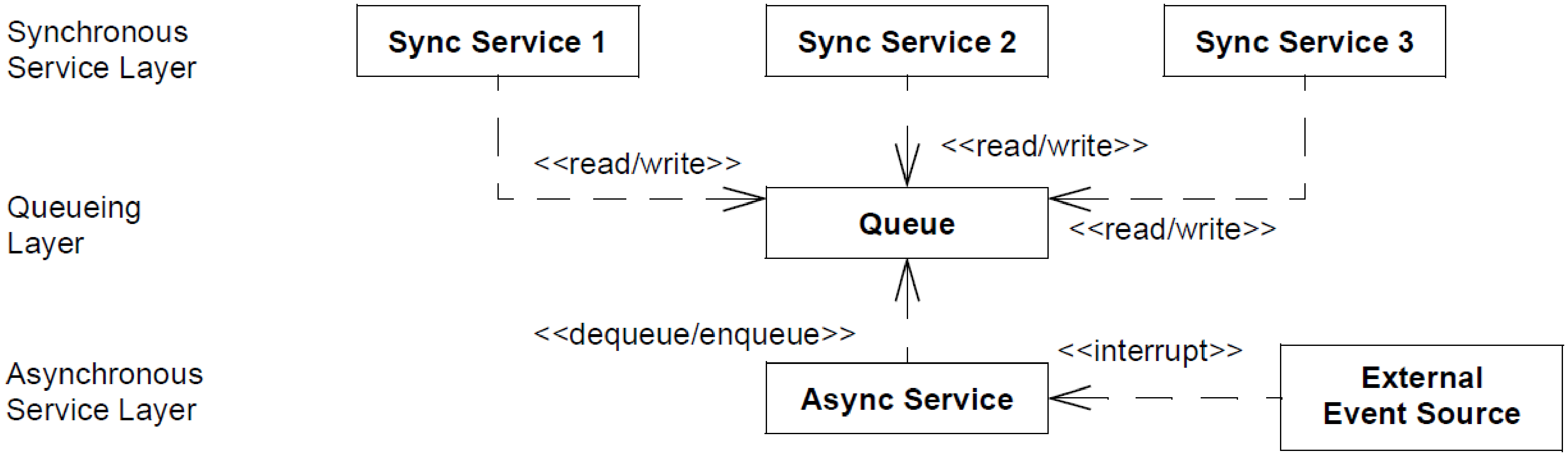
- When it’s necessary to make performance efficient & scalable, while also ensuring that the use of concurrency simplifies—rather than complicates—programming



Half-Sync/Half-Async POSA2 Concurrency

Applicability

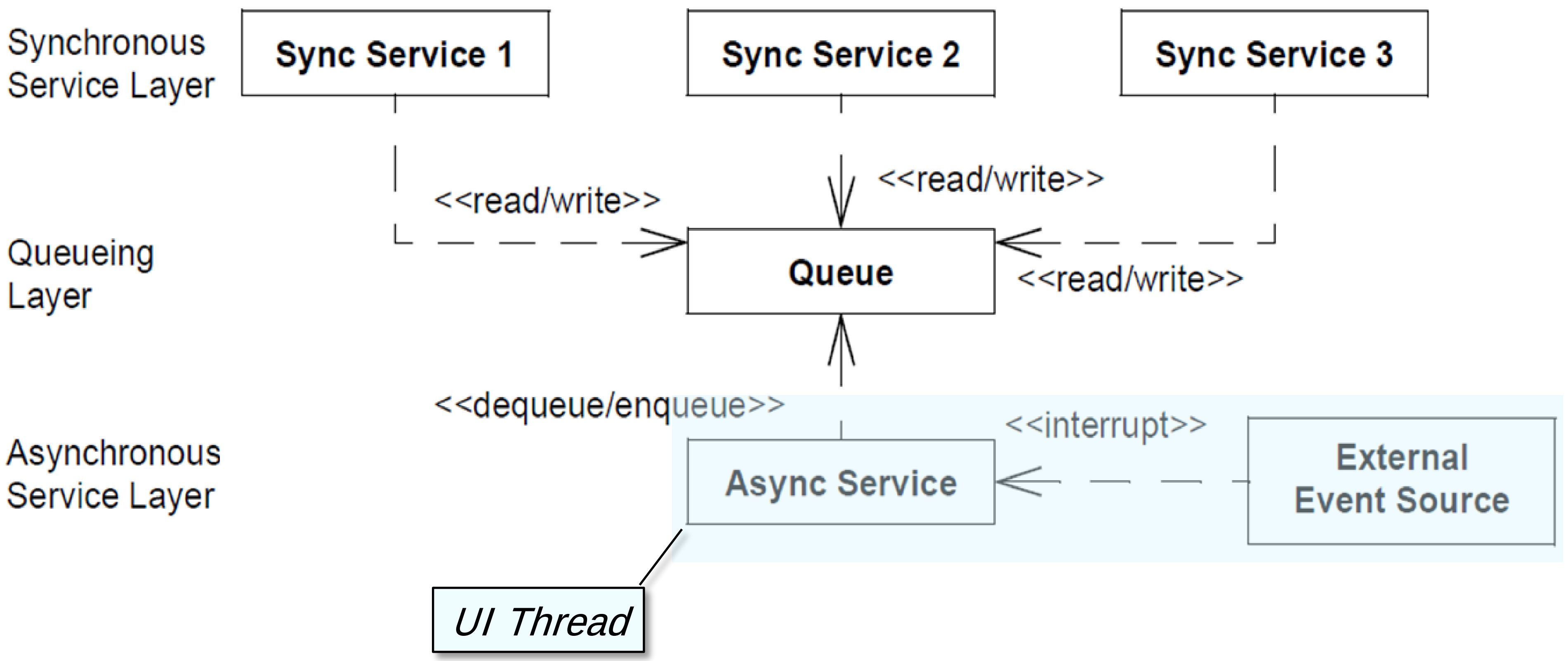
- When it’s necessary to make performance efficient & scalable, while also ensuring that the use of concurrency simplifies—rather than complicates—programming
- When there are constraints on certain types of operations in certain contexts
  - e.g., short-duration vs. long-duration, blocking vs. non-blocking, etc.



This pattern is widely applied in operating systems & modern GUI frameworks

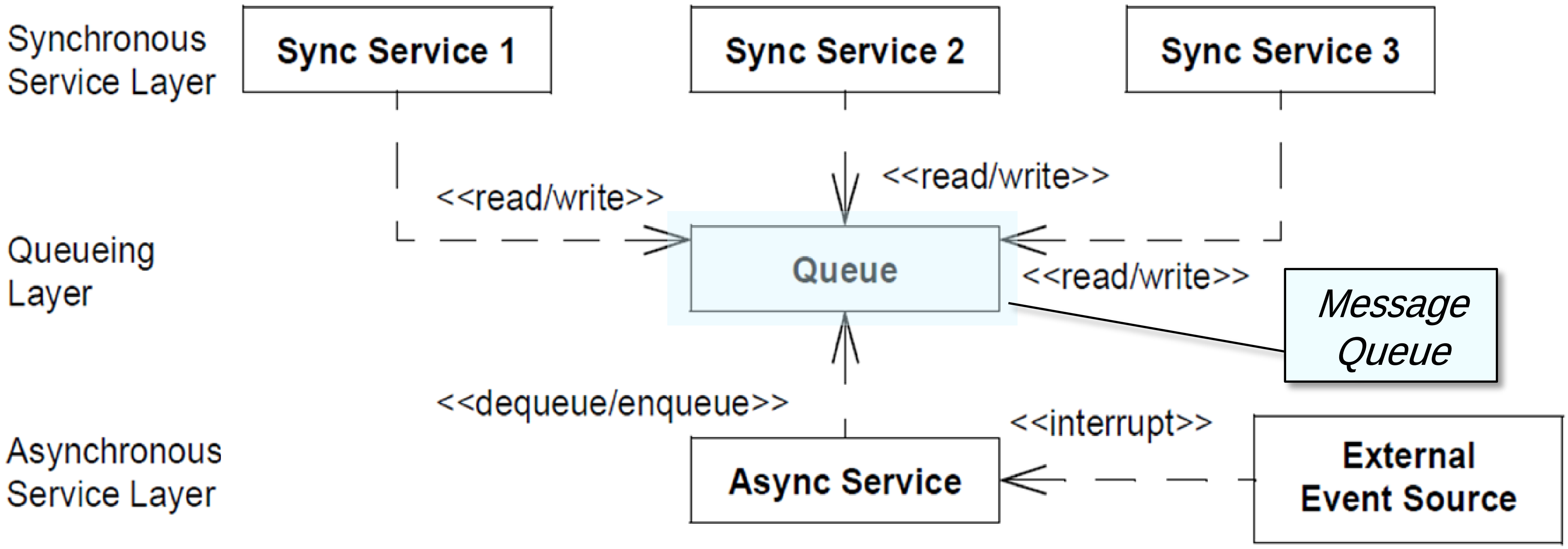
Half-Sync/Half-Async POSA2 Concurrency

Structure & Participants



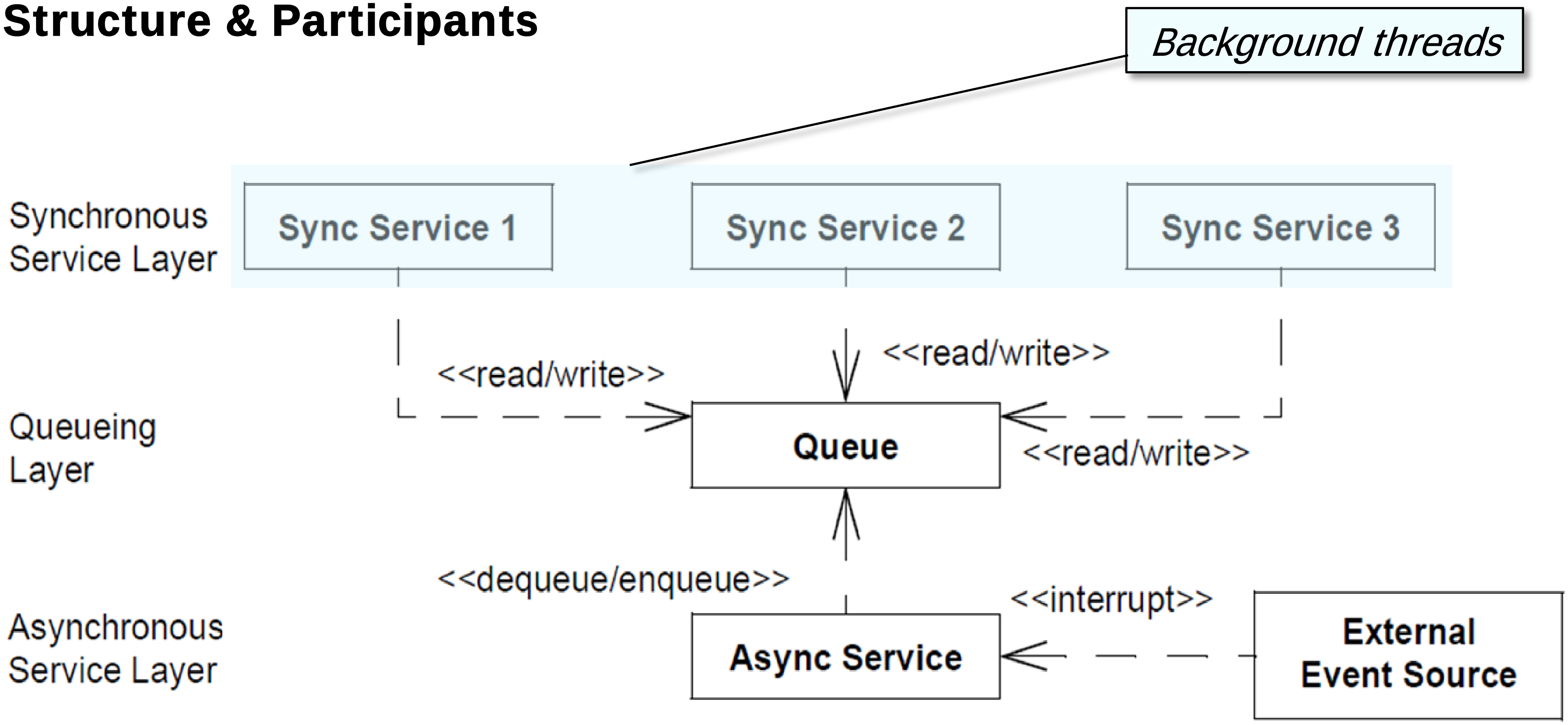
Half-Sync/Half-Async POSA2 Concurrency

Structure & Participants



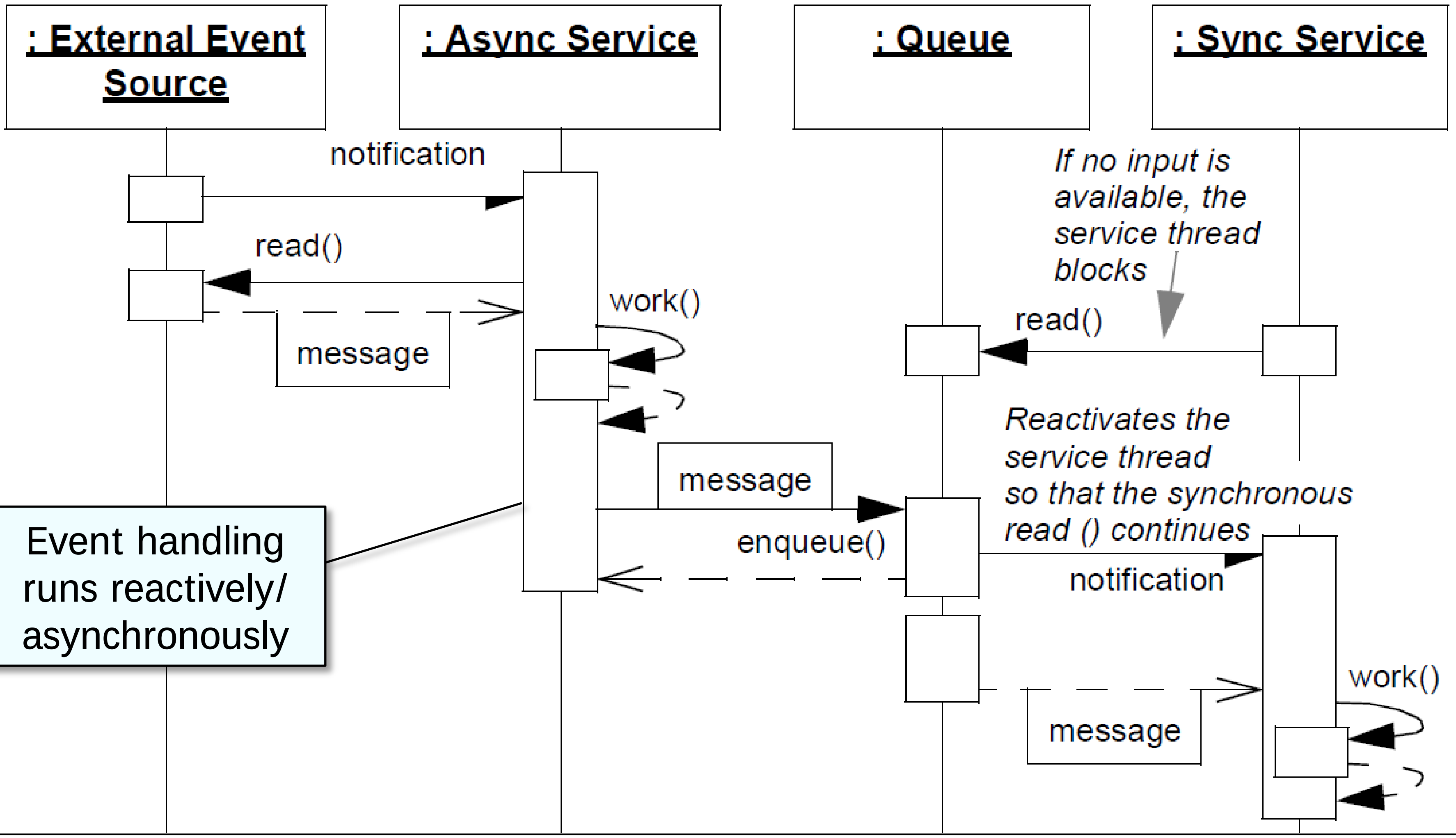
Half-Sync/Half-Async POSA2 Concurrency

Structure & Participants



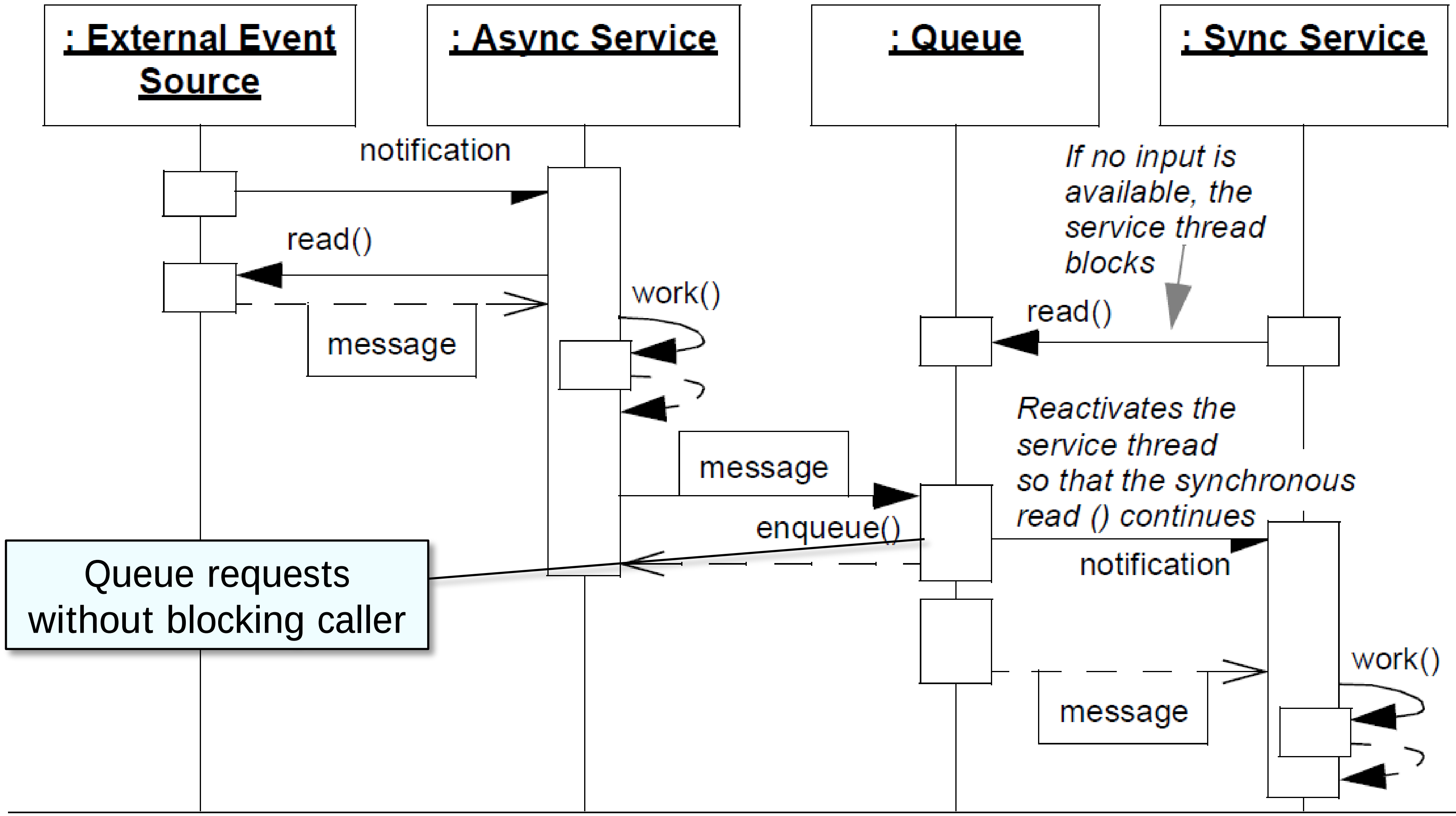
Half-Sync/Half-Async POSA2 Concurrency

Dynamics


: External Event Source
: Async Service
: Queue
: Sync Service
notification
read()
message
work()
If no input is available, the service thread blocks
read()
message
enqueue()
Reactivates the service thread so that the synchronous read () continues
notification
message
work()
Event handling runs reactively/ asynchronously

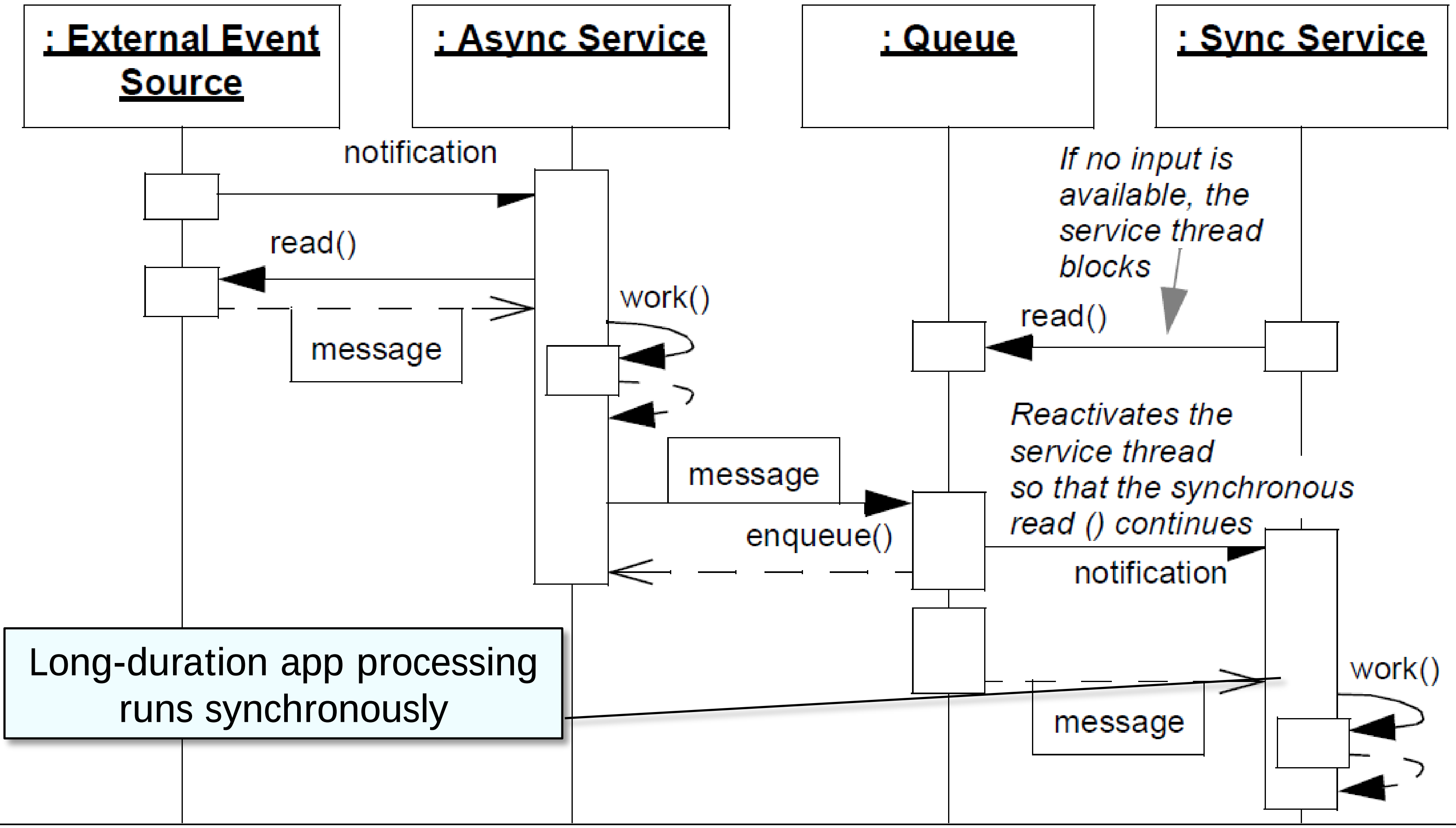
Half-Sync/Half-Async POSA2 Concurrency

Dynamics


: External Event Source
: Async Service
: Queue
: Sync Service
notification
read()
message
work()
If no input is available, the service thread blocks
read()
message
enqueue()
Reactivates the service thread so that the synchronous read () continues
notification
message
work()
Queue requests without blocking caller

Half-Sync/Half-Async POSA2 Concurrency

Dynamics

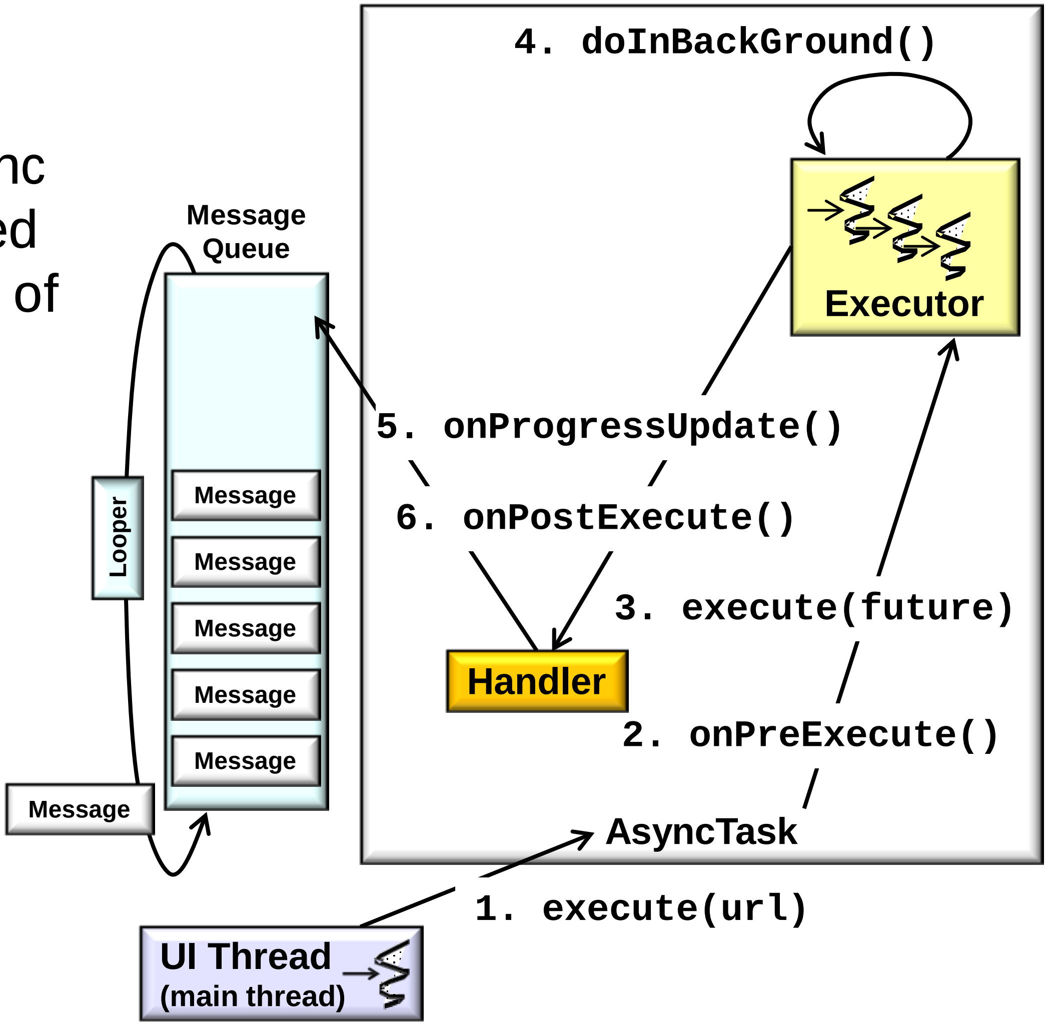

: External Event Source
: Async Service
: Queue
: Sync Service
notification
read()
message
work()
If no input is available, the service thread blocks
read()
message
enqueue()
Reactivates the service thread so that the synchronous read () continues
notification
work()
message
Long-duration app processing runs synchronously

Sync services run concurrently, relative both to each other & to async services

Half-Sync/Half-Async POSA2 Concurrency

Consequences

+ Simplification & performance
  - Programming of higher-level sync processing services are simplified without degrading performance of lower-level system services

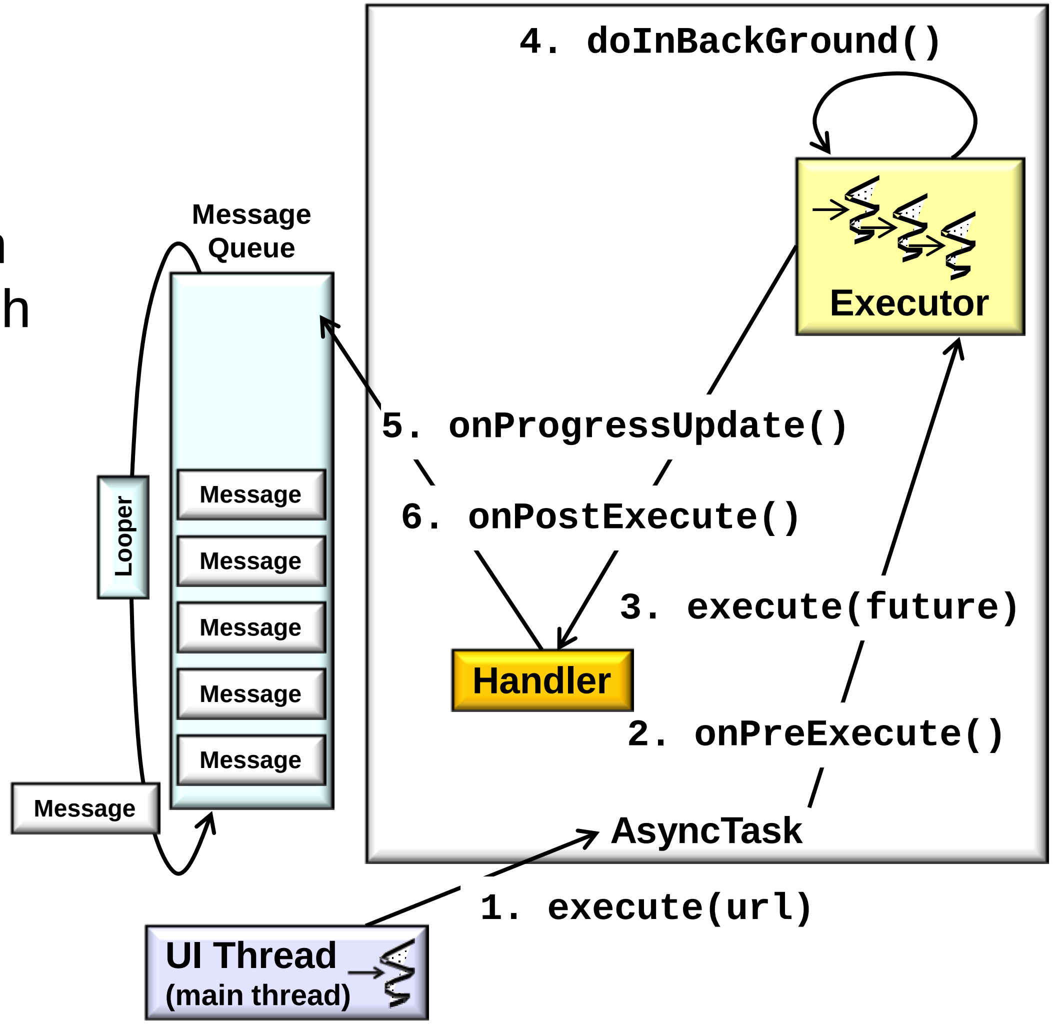


Half-Sync/Half-Async POSA2 Concurrency

Consequences

+ Simplification & performance

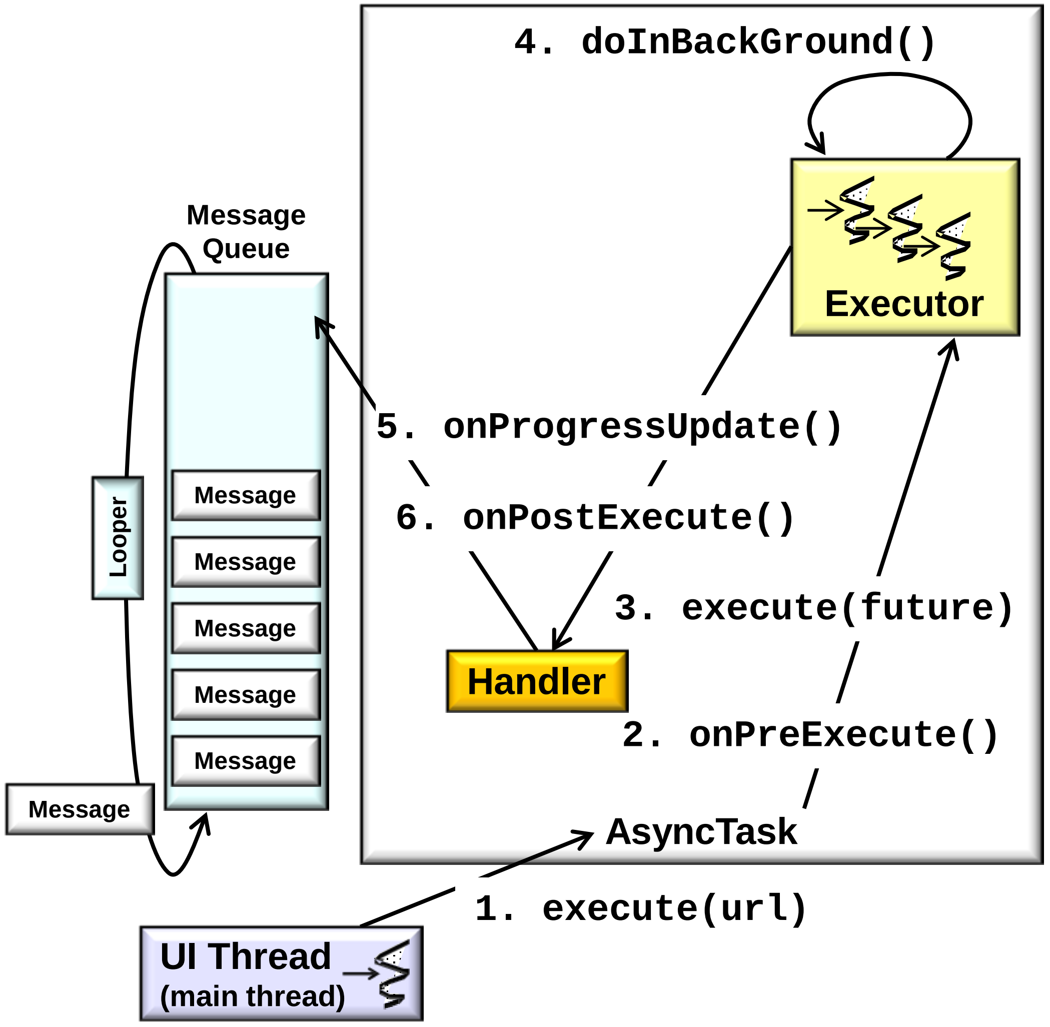
+ Separation of concerns
  - Synchronization policies in each layer are decoupled so that each layer need not use the same concurrency strategies



Half-Sync/Half-Async POSA2 Concurrency

Consequences

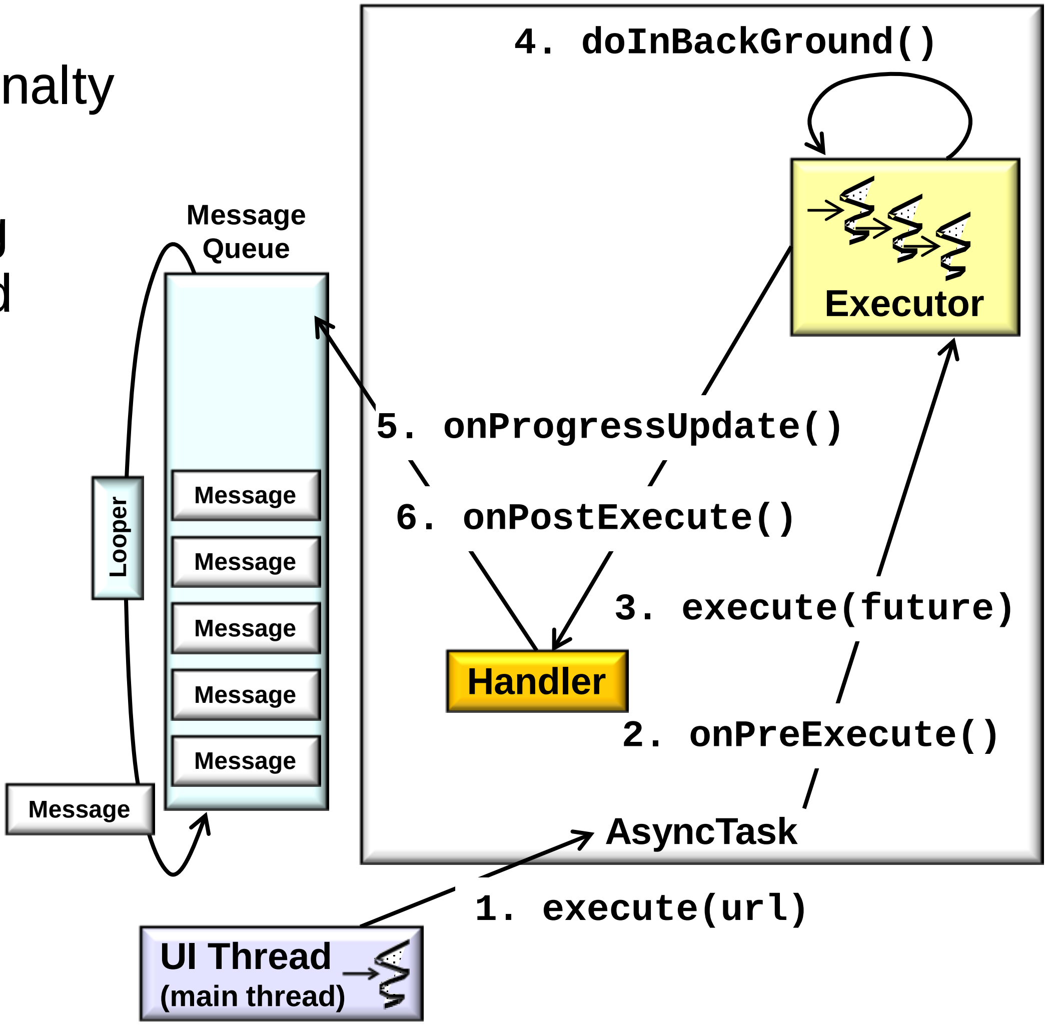
+ Simplification & performance
+ Separation of concerns
+ Centralization of inter-layer communication
  - Inter-layer communication is centralized because all interaction is mediated by the queueing layer



Half-Sync/Half-Async POSA2 Concurrency

Consequences

– May incur a boundary-crossing penalty
  - Arising from context switching, synchronization, & data copying overhead when data transferred between sync & async service layers via queueing layer



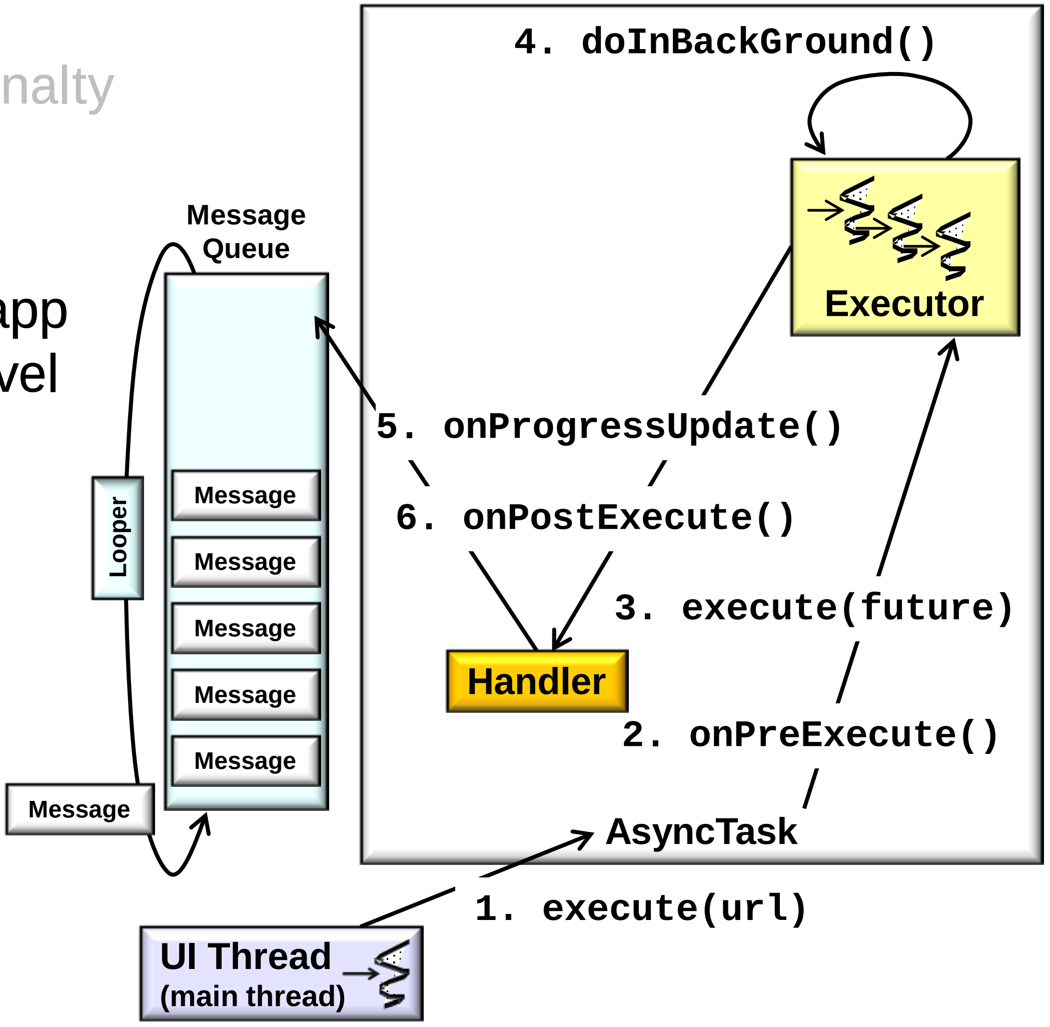
Half-Sync/Half-Async POSA2 Concurrency

Consequences

– May incur a boundary-crossing penalty

– Higher-level app services may not benefit from async I/O

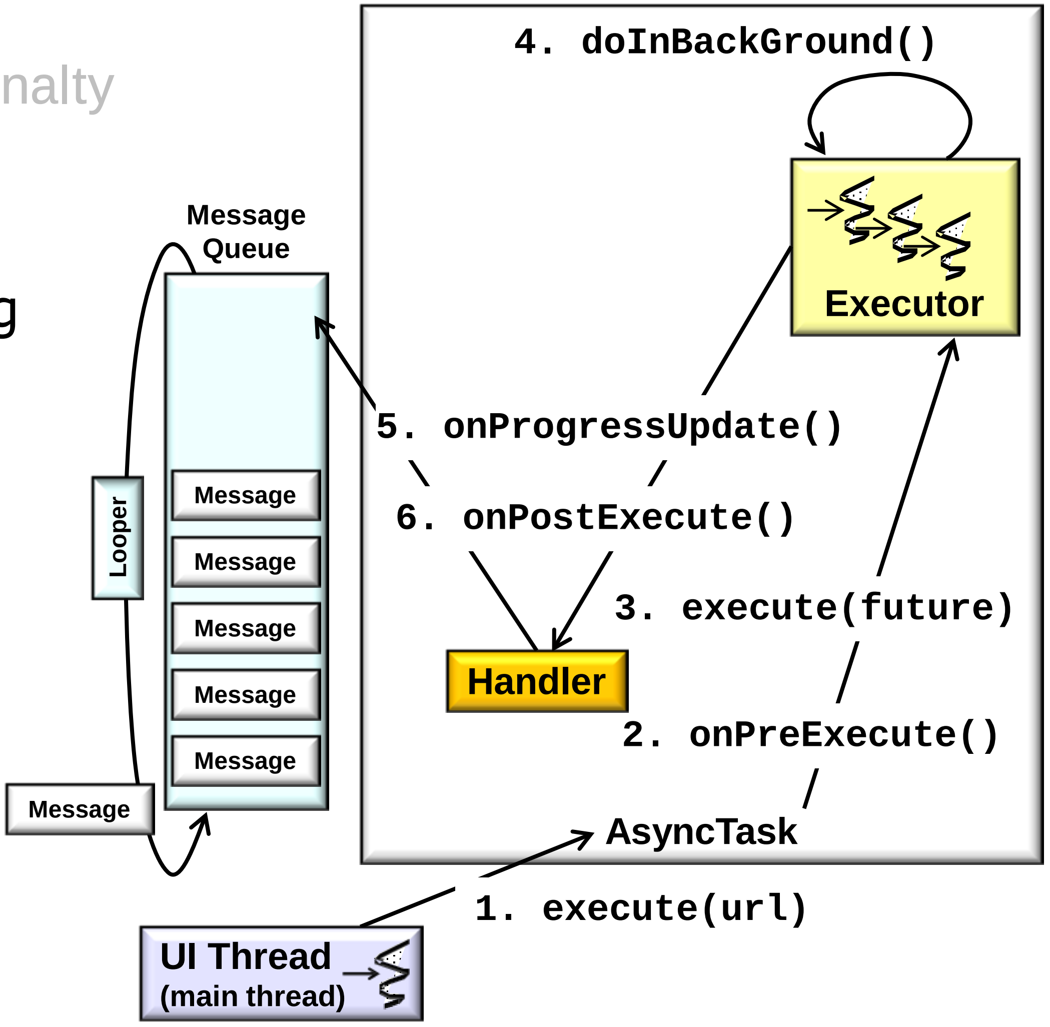
- Depending on design of OS or app framework interfaces, higher-level services may not use low-level async I/O devices effectively



Half-Sync/Half-Async POSA2 Concurrency

Consequences

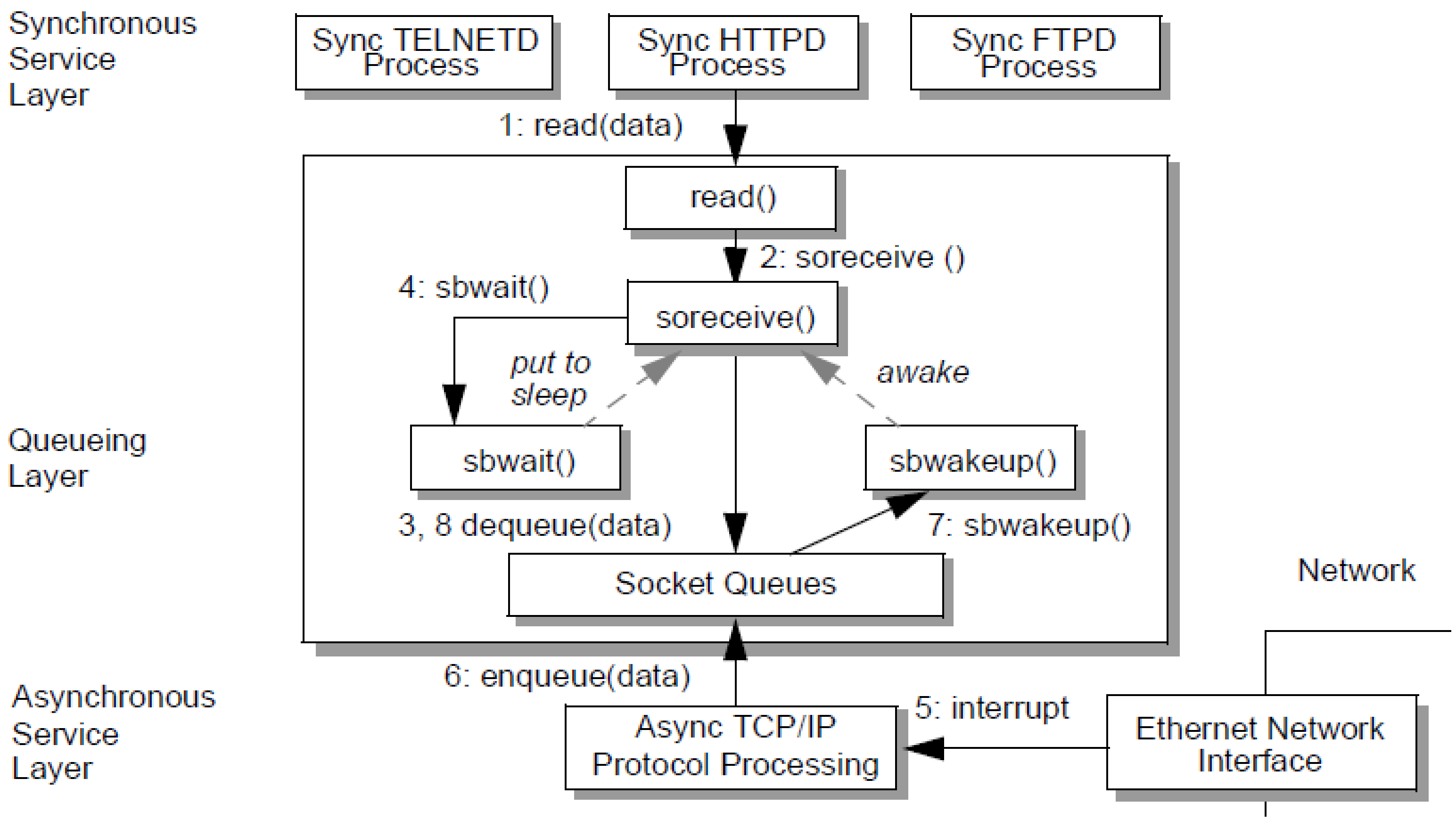
- May incur a boundary-crossing penalty
- Higher-level app services may not benefit from async I/O
- Complexity of debugging & testing
  - Apps can be hard to debug due to concurrent execution



Half-Sync/Half-Async POSA2 Concurrency

Known Uses

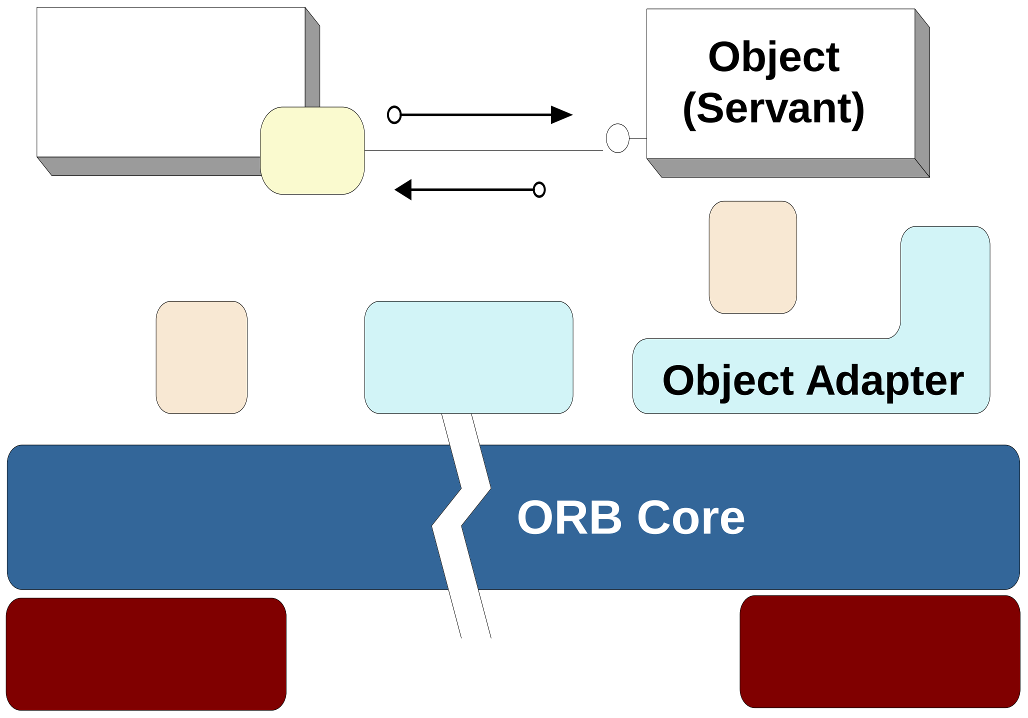
- UNIX Networking Subsystems

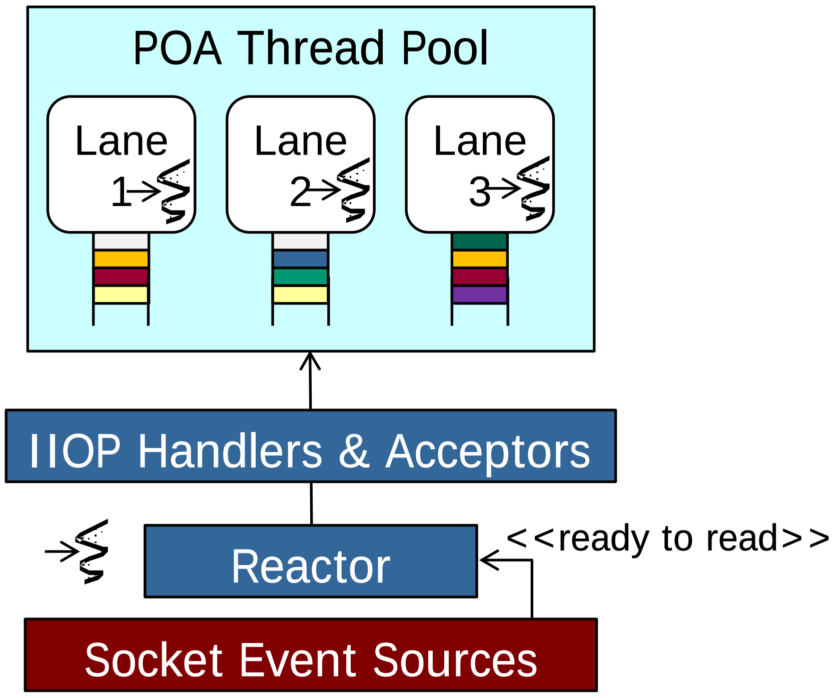


Half-Sync/Half-Async POSA2 Concurrency

Known Uses

- UNIX Networking Subsystems
- Object Request Brokers (ORBs)

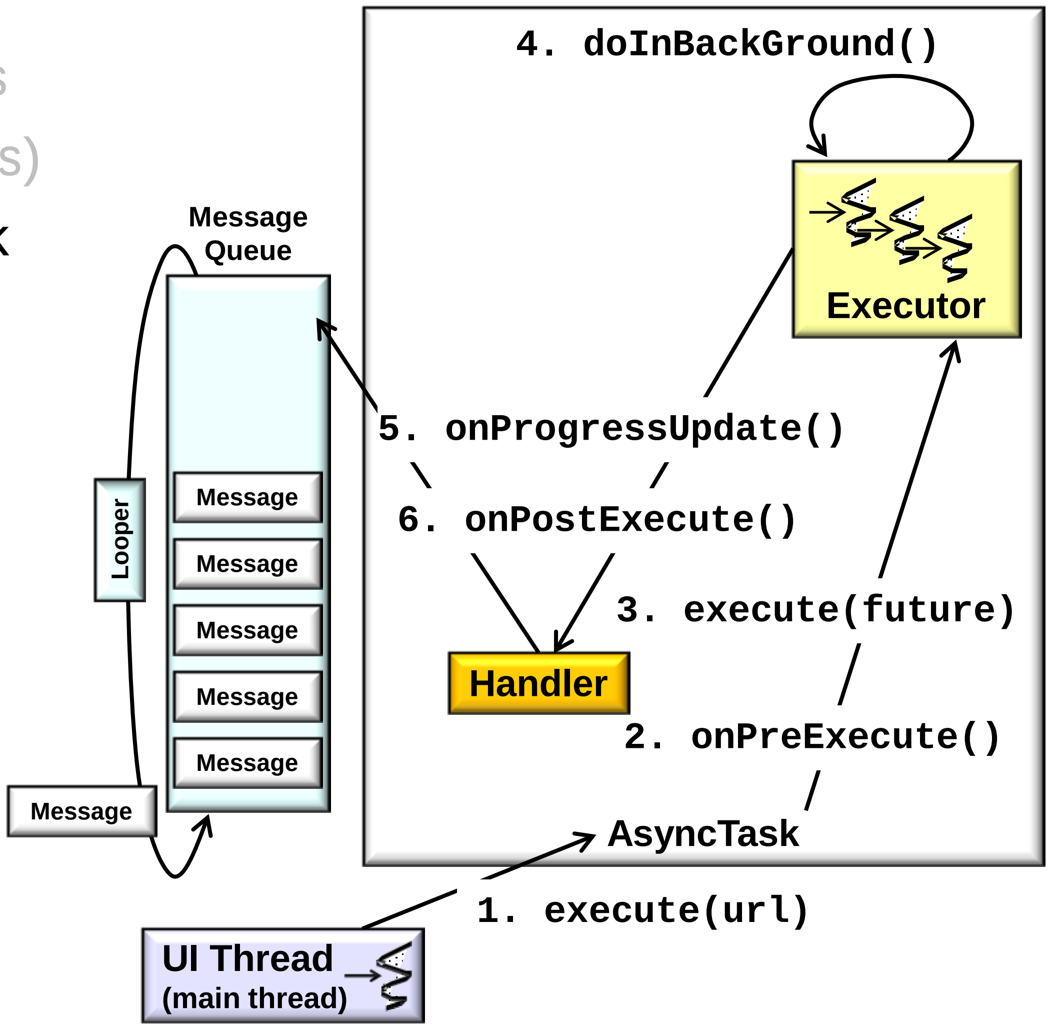




Half-Sync/Half-Async POSA2 Concurrency

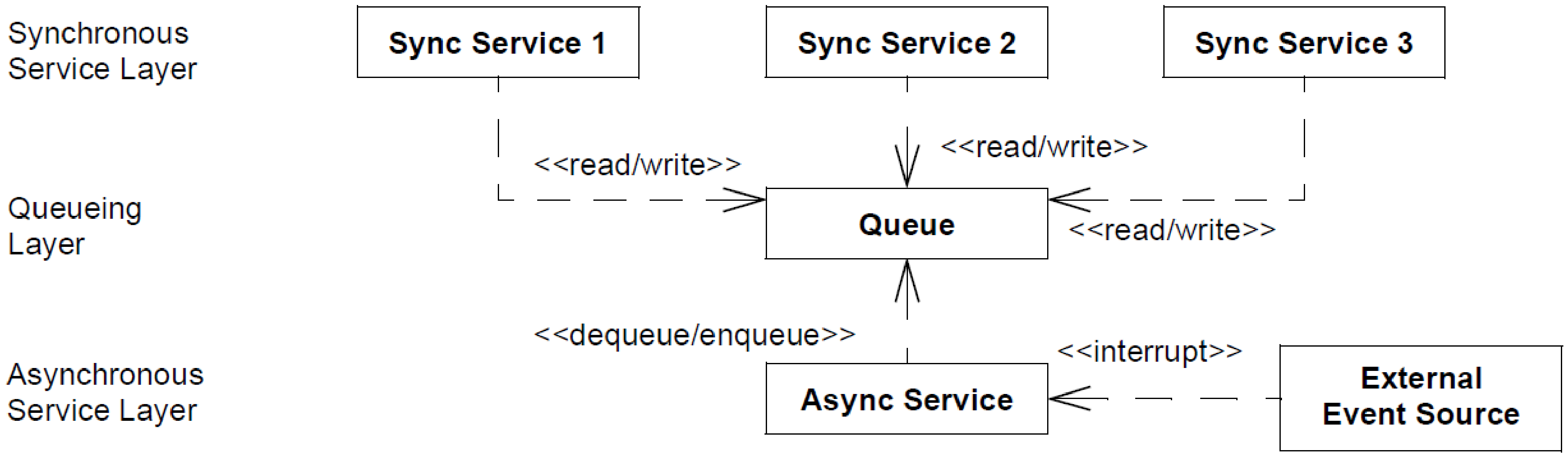
Known Uses

- UNIX Networking Subsystems
- Object Request Brokers (ORBs)
- Android AsyncTask framework



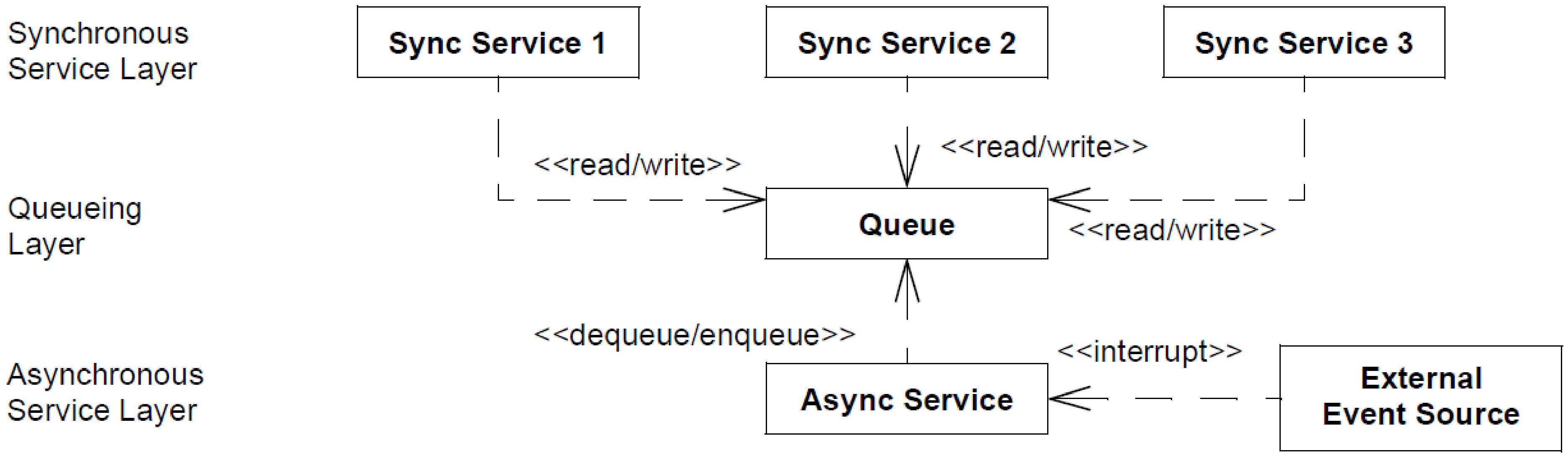
developer.android.com/training/multiple-threads/communicate-ui.html

Summary



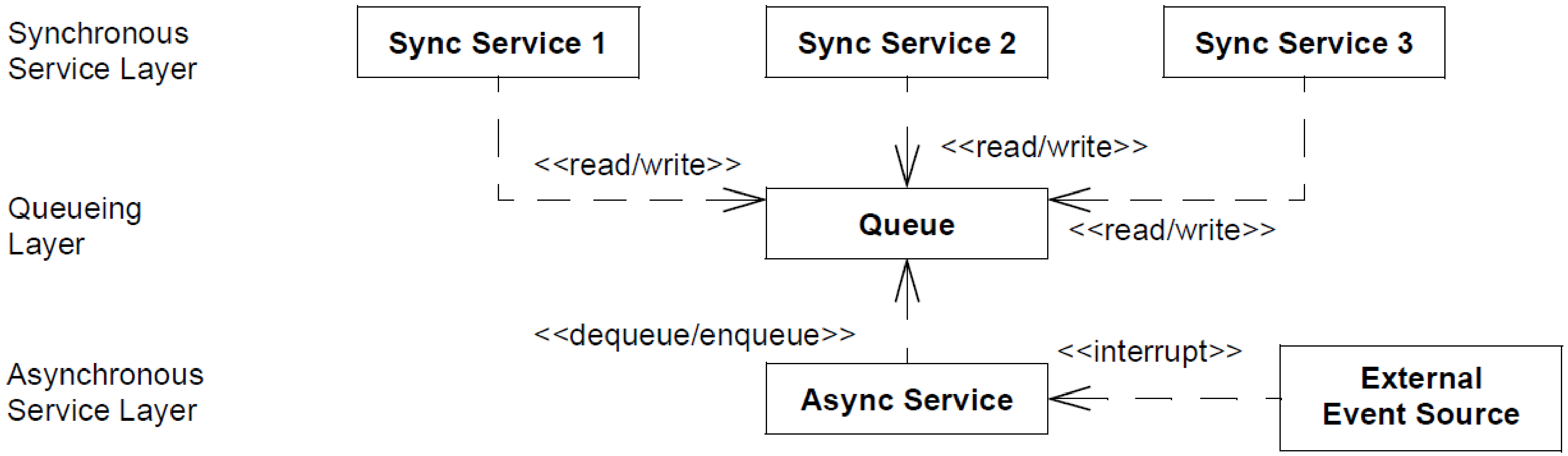
- This pattern separates concerns between the three layers, which makes concurrent software easier to understand, debug, & evolve

Summary



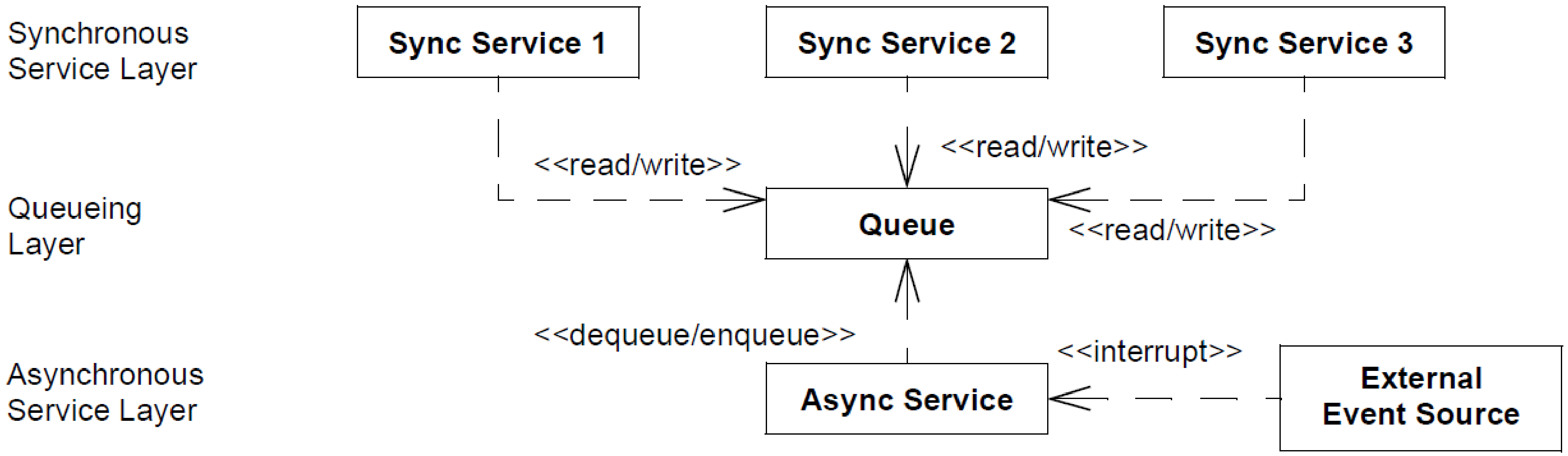
- This pattern separates concerns between the three layers, which makes concurrent software easier to understand, debug, & evolve
- In addition, async & sync services do not suffer from each other's liabilities
  - Async service performance does not degrade due to blocking sync services

Summary



- This pattern separates concerns between the three layers, which makes concurrent software easier to understand, debug, & evolve
- In addition, async & sync services do not suffer from each other's liabilities
  - Async service performance does not degrade due to blocking sync services
  - The simplicity of programming sync services is unaffected by async complexities, such as explicit state management

Summary



- This pattern separates concerns between the three layers, which makes concurrent software easier to understand, debug, & evolve
- In addition, async & sync services do not suffer from each other's liabilities
- The queueing layer avoids hard-coded dependencies between the async & sync service layers
  - It's also easy to reprioritize the order in which messages are processed

Android Concurrency: The Half-Sync/Half-Async Pattern (Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

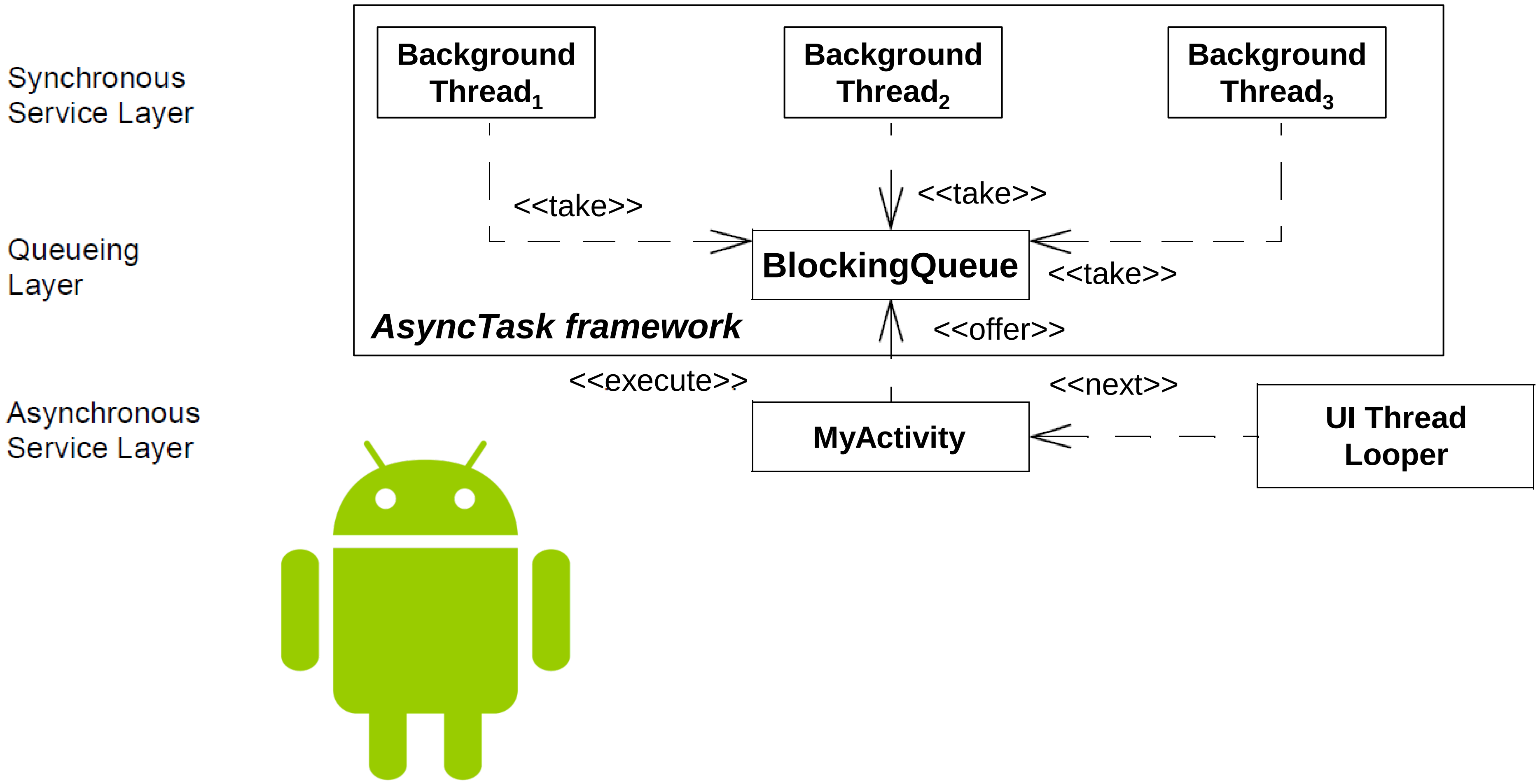
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

CS 282 Principles of Operating Systems II

Systems Programming for Android

Learning Objectives in this Part of the Module

- Understand how *Half-Sync/Half-Async* is implemented & applied in Android



Half-Sync/Half-Async POSA2 Concurrency

Implementation

- Decompose overall system into three layers: synchronous, asynchronous, & queueing

```
public abstract class AsyncTask<Params, Progress, Result> {
  public final AsyncTask<Params, Progress, Result>
    execute(Params... params) {
    return executeOnExecutor(sDefaultExecutor, params);
  }

  public final AsyncTask<Params, Progress, Result>
    executeOnExecutor(Executor exec, Params... params) {
    onPreExecute();
    mWorker.mParams = params;
    exec.execute(mFuture);
    return this;
  }
  ...
```

Identify short-duration services & implement them in the async layer

frameworks/base/core/java/android/os/AsyncTask.java has the source code

Half-Sync/Half-Async POSA2 Concurrency

Implementation

- Decompose overall system into three layers: synchronous, asynchronous, & queueing

```
public abstract class AsyncTask<Params, Progress, Result> {
  public AsyncTask() {
    mWorker = new WorkerRunnable<Params, Result>() {
      public Result call() throws Exception {
        ...
        return postResult(doInBackground(mParams));
      }
    };
    ...
```

Identify long-duration services & implement them in the sync layer

frameworks/base/core/java/android/os/AsyncTask.java has the source code

Half-Sync/Half-Async POSA2 Concurrency

Implementation

- Decompose overall system into three layers: synchronous, asynchronous, & queueing

```
public class ThreadPoolExecutor
       extends AbstractExecutorService {
  /**
   * The queue used for holding tasks and handing off to worker
   * threads. */
  private final BlockingQueue<Runnable> workQueue;
```

Identify inter-layer communication strategies & implement them in the queueing layer

frameworks/base/core/java/android/os/AsyncTask.java has the source code

Half-Sync/Half-Async POSA2 Concurrency

Implementation

- Decompose overall system into three layers: synchronous, asynchronous, & queueing
- Implement the services in the synchronous layer

```
class DownloadAsyncTask extends
      AsyncTask<String, Integer, Bitmap> {
  ...
  protected Bitmap
  doInBackground(String... url) {
    return downloadImage(url[0]);
  }
```

Download in background thread

Half-Sync/Half-Async POSA2 Concurrency

Implementation

- Decompose overall system into three layers: synchronous, asynchronous, & queueing
- Implement the services in the synchronous layer
- Implement the services in the asynchronous layer

```
class DownloadAsyncTask extends
      AsyncTask<String, Integer, Bitmap> {

  protected void onPreExecute() {
    dialog.display();
  }

  protected void onPostExecute
                    (Bitmap bitmap) {
    performPostDownloadOperations(bitmap);
    dialog.dismiss();
  }
}
```

Perform on UI thread

Perform on UI thread

Half-Sync/Half-Async POSA2 Concurrency

Implementation

- Decompose overall system into three layers: synchronous, asynchronous, & queueing
- Implement the services in the synchronous layer
- Implement the services in the asynchronous layer
- Implement (or reuse) the queueing layer

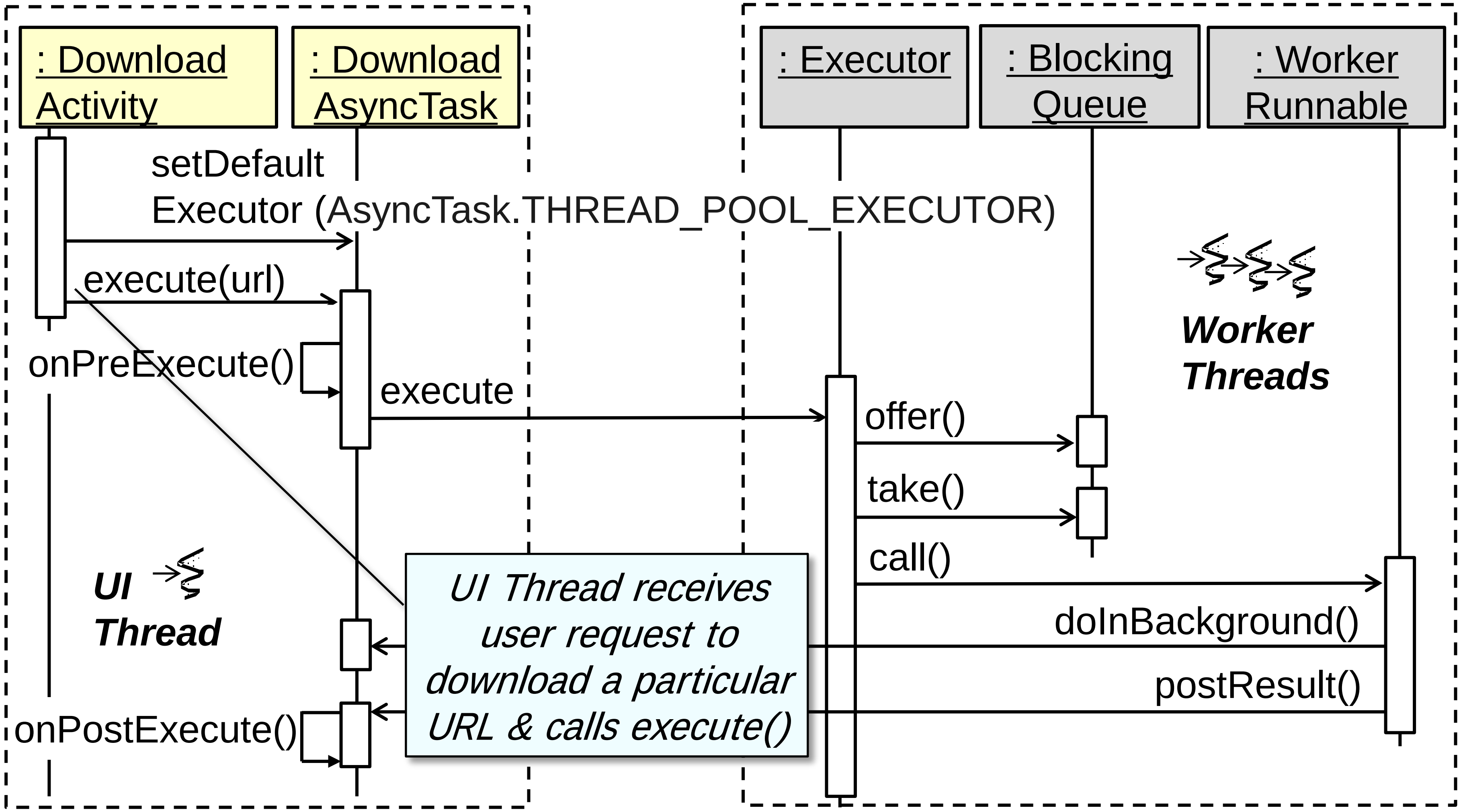
```
public class ThreadPoolExecutor
       extends AbstractExecutorService {
  ...
  private Runnable getTask() {
    ...
    Runnable r = workQueue.take();
    ...
    return r;
    ...

  public void execute(Runnable command) {
    ...
    workQueue.offer(command);
    ...
```

frameworks/base/core/java/android/os/AsyncTask.java has the source code

Half-Sync/Half-Async POSA2 Concurrency

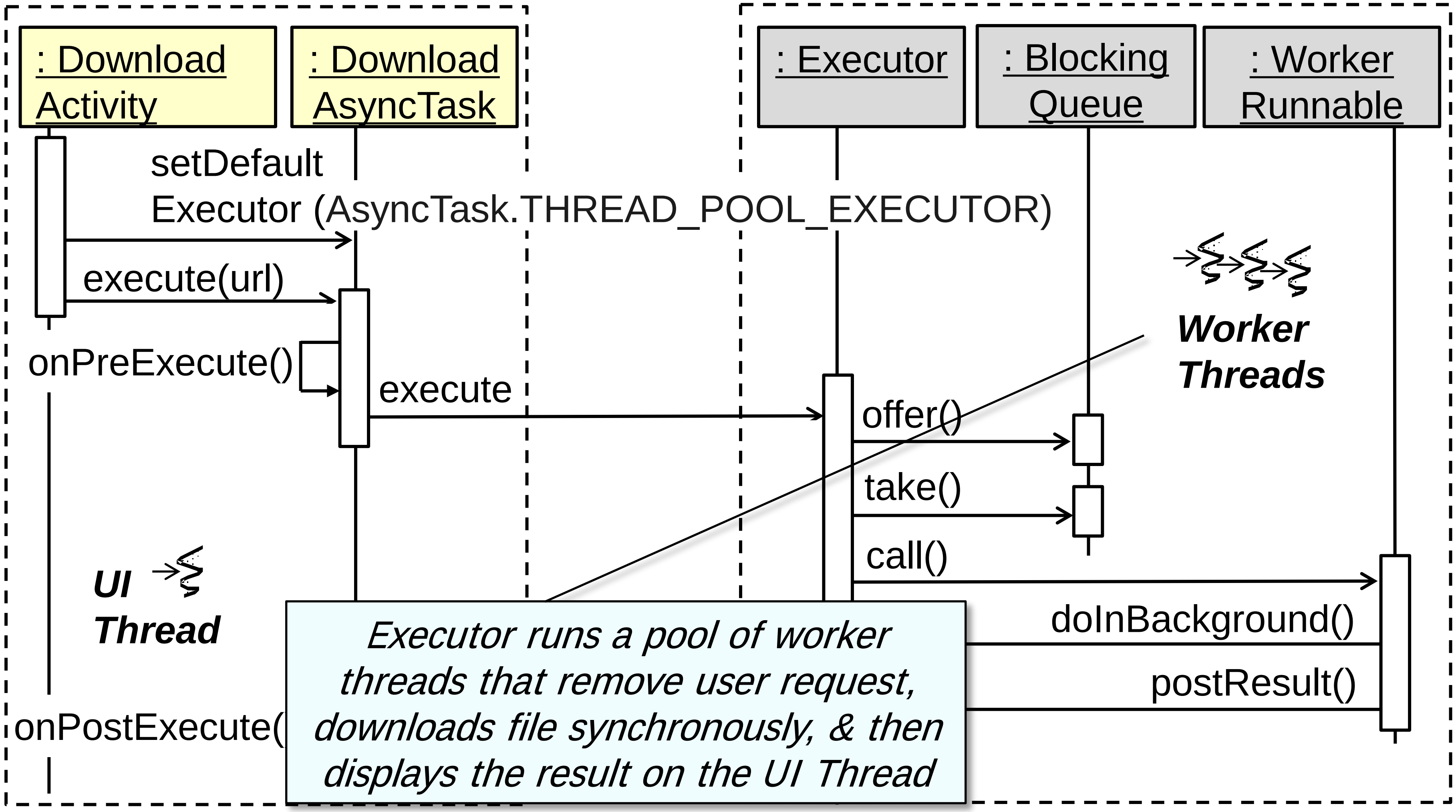
Applying Half-Sync/Half-Async in Android



UI Thread receives user request to download a particular URL & calls execute()

Half-Sync/Half-Async POSA2 Concurrency

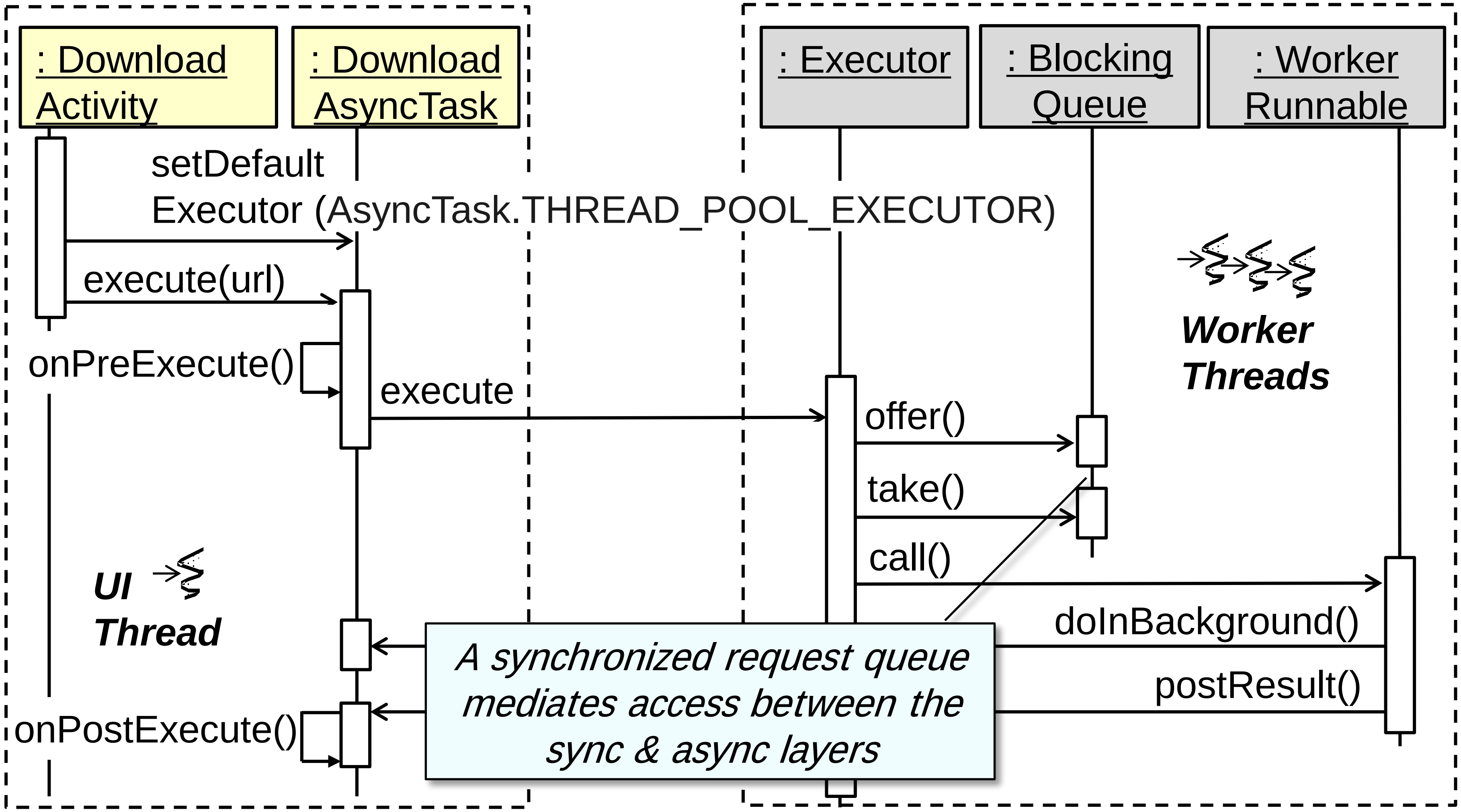
Applying Half-Sync/Half-Async in Android



Executor runs a pool of worker threads that remove user request, downloads file synchronously, & then displays the result on the UI Thread

Half-Sync/Half-Async POSA2 Concurrency

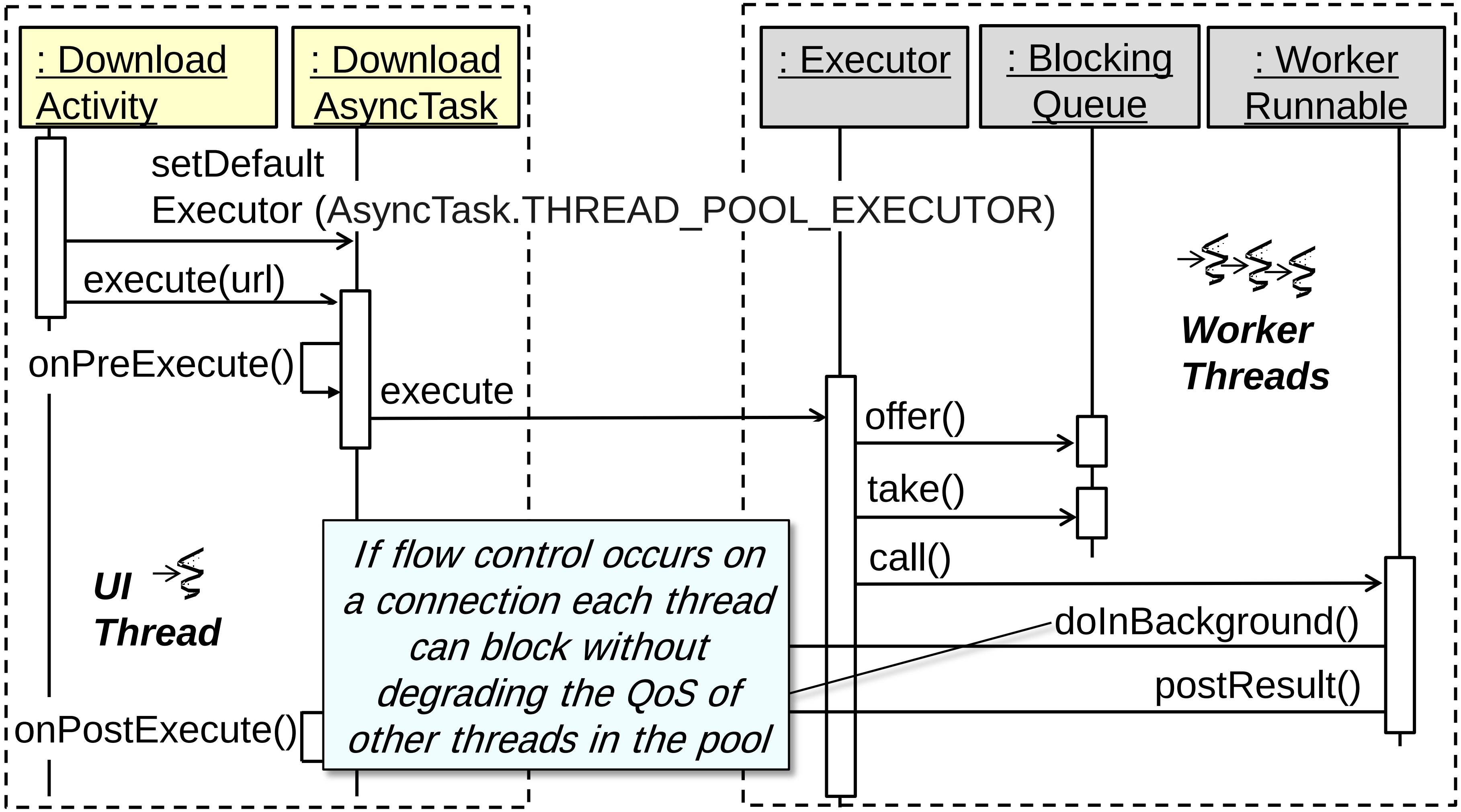
Applying Half-Sync/Half-Async in Android



A synchronized request queue mediates access between the sync & async layers

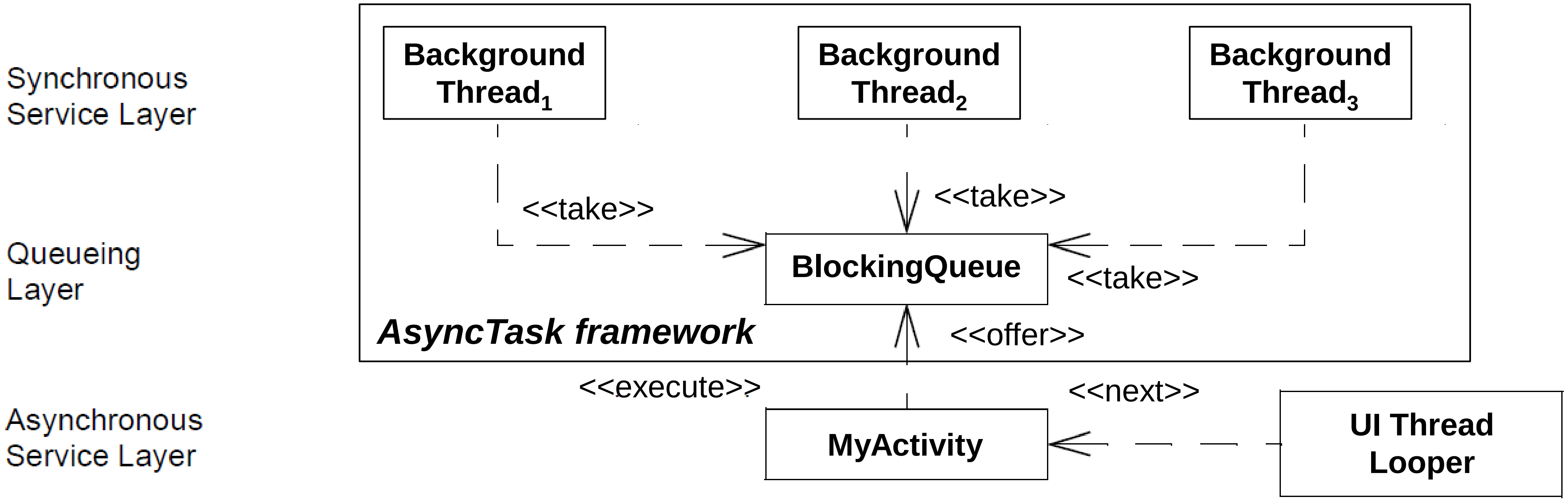
Half-Sync/Half-Async POSA2 Concurrency

Applying Half-Sync/Half-Async in Android



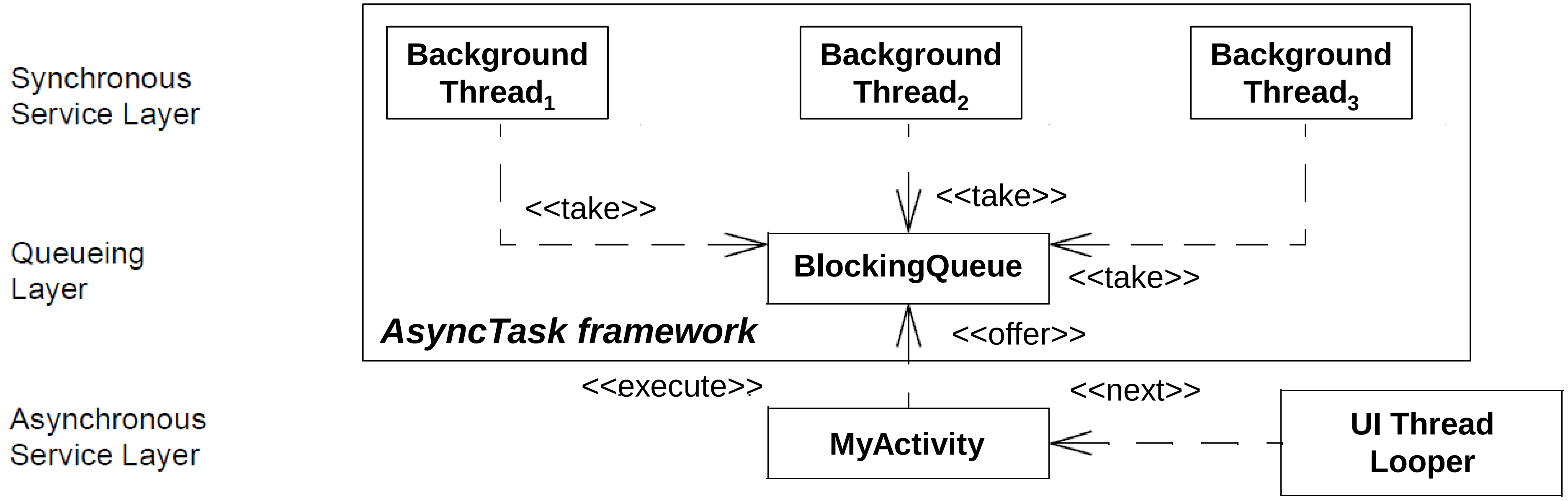
If flow control occurs on a connection each thread can block without degrading the QoS of other threads in the pool

Summary



- The Android AsyncTask framework implements *Half-Sync/Half-Async* pattern to encapsulate the creation of background thread processing & synchronization with the UI Thread
  - It also supports reporting progress of the running tasks

Summary



- The Android AsyncTask framework implements *Half-Sync/Half-Async* pattern to encapsulate the creation of background thread processing & synchronization with the UI Thread
- The AsyncTask framework is a sophisticated implementation of Half-Sync/Half-Async
  - e.g., there are multiple interactions between the sync & async portions via various queues