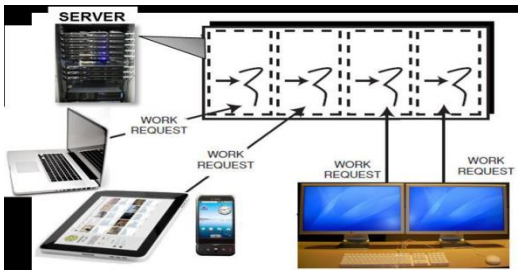
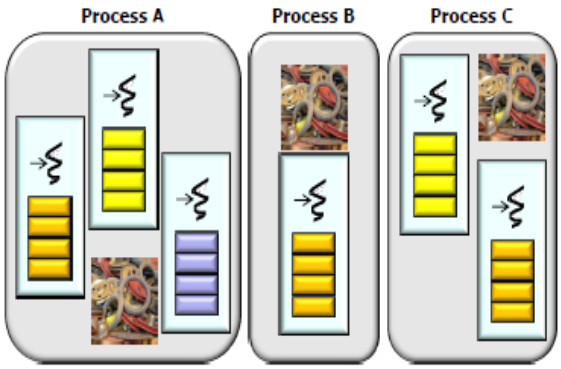
Android Concurrency & Synchronization

Introduction

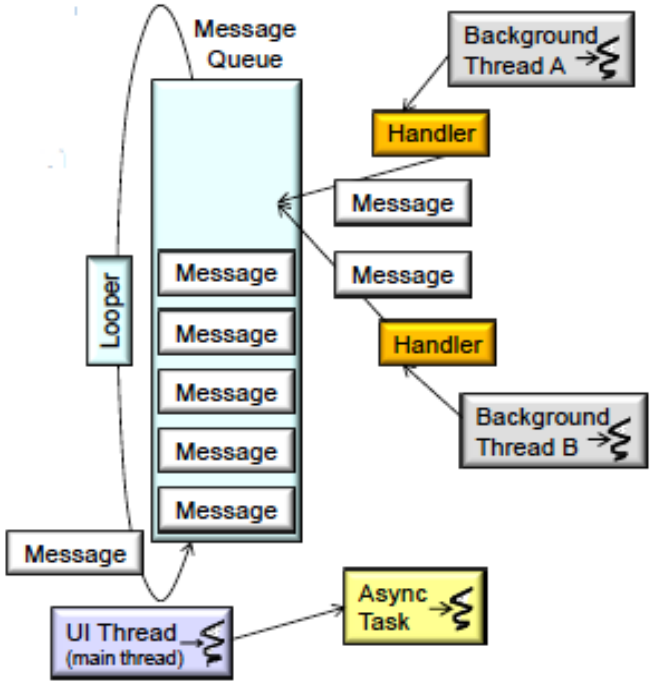
- Explore the **motivations** for & **challenges** of concurrent software
  - **Concurrent software** can simultaneously **run** multiple **computations** that potentially **interact with each other**.



- **Understand** the **mechanisms** that **Android** provides to **manage multiple threads** that **run concurrently** within a process

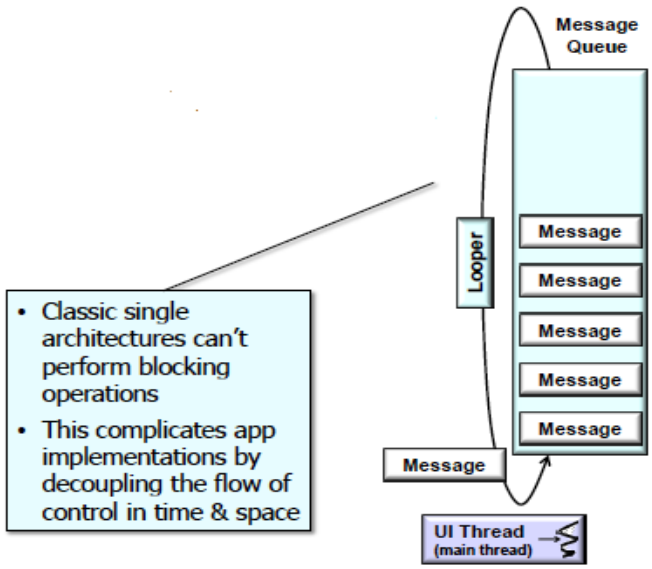


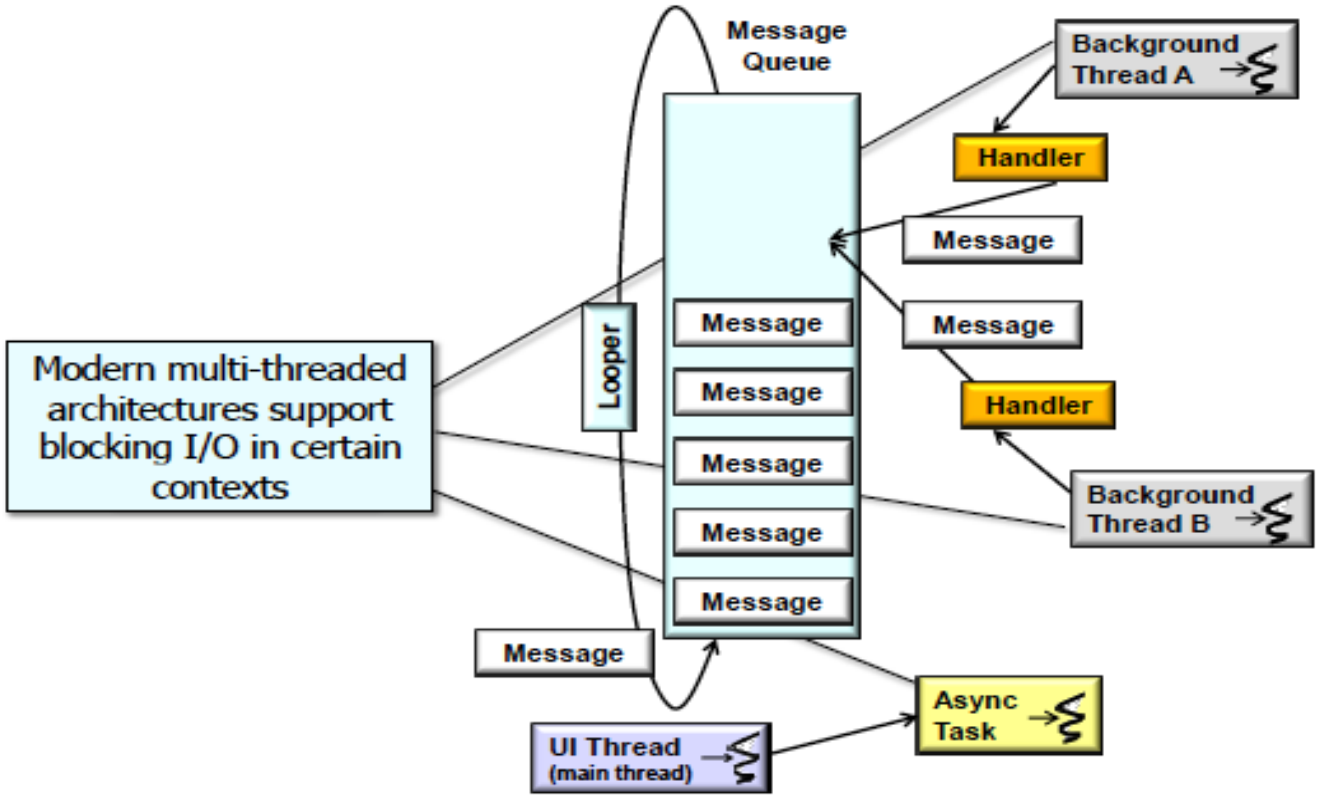
- Some **Android mechanisms** are based on **standard Java threading** & **locking** mechanisms.
- Other **mechanisms** are based on **Android concurrency idioms**



Motivations for Concurrent Software

- Leverage hardware/software advances
  - e.g., multi-core processors & multi-threaded operating systems, virtual machines, & middleware
- Increase performance
  - Parallelize computations & communications
- Improve response-time
  - e.g., don't starve the UI thread
- Simplify program structure
  - e.g., by allow blocking operations





Challenges for Concurrent Software

- **Accidental Complexities**
  - Low-level APIs
    - Tedious, error-prone, & non-portable
  - Limited debugging tools
- **Inherent Complexities**
  - **Synchronization** is the application of **mechanisms** to **ensure** that **two concurrently-executing threads do not** execute specific portions of a program **at the same time**
  - **Scheduling** is the **method** by which **threads**, **processes**, or **data flows** are given **access to system resources**
  - **A deadlock** is a **situation** in which **two or more competing actions** are each **waiting** for **the other to finish**, and thus neither ever does

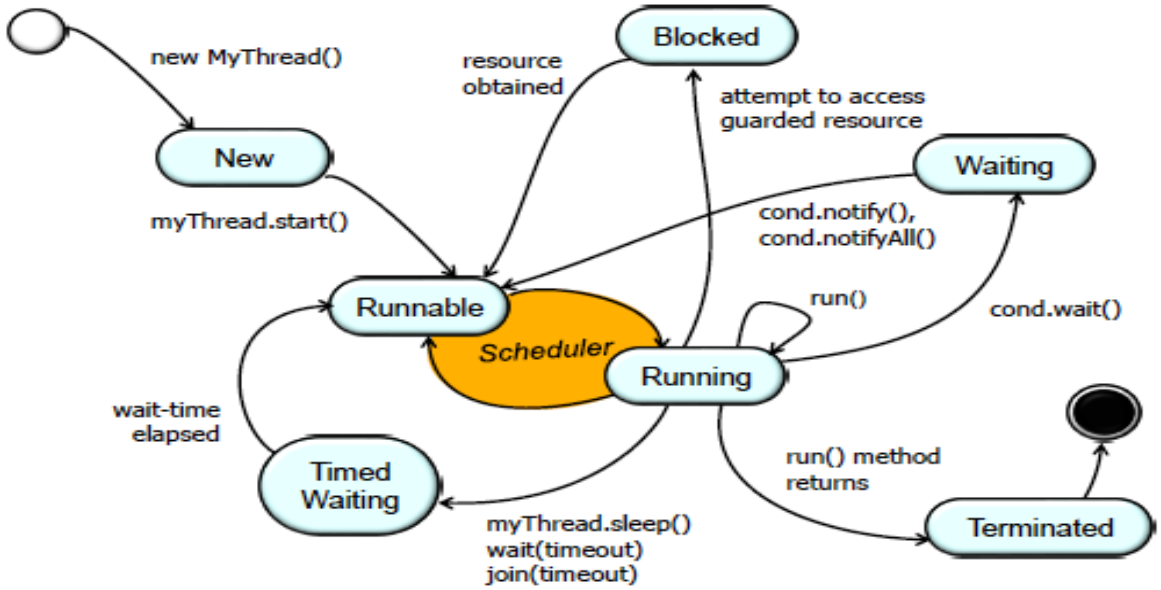
Overview of Java Threads in Android

- **Android** implements many standard Java concurrency & synchronization classes
- **Conceptual view**
  - Concurrent computations running in a (Linux) process that can communicate with each other via **shared memory** or **message passing**
- **Implementation view**
  - Each **Java thread** has a program **counter & a stack** (unique)
  - The **heap & static** areas are **shared** across threads (common)

Motivating Android's Concurrency Frameworks

- **Android's concurrency frameworks also address design constraints, e.g.**
  - "**ANR**" dialog is generated if the **UI thread blocks too long.**
  - **Network calls** are **disallowed** on the UI thread by default.
  - **Non UI threads** can't access **UI toolkit** components **directly**.

State Machine for Java Threads in Android



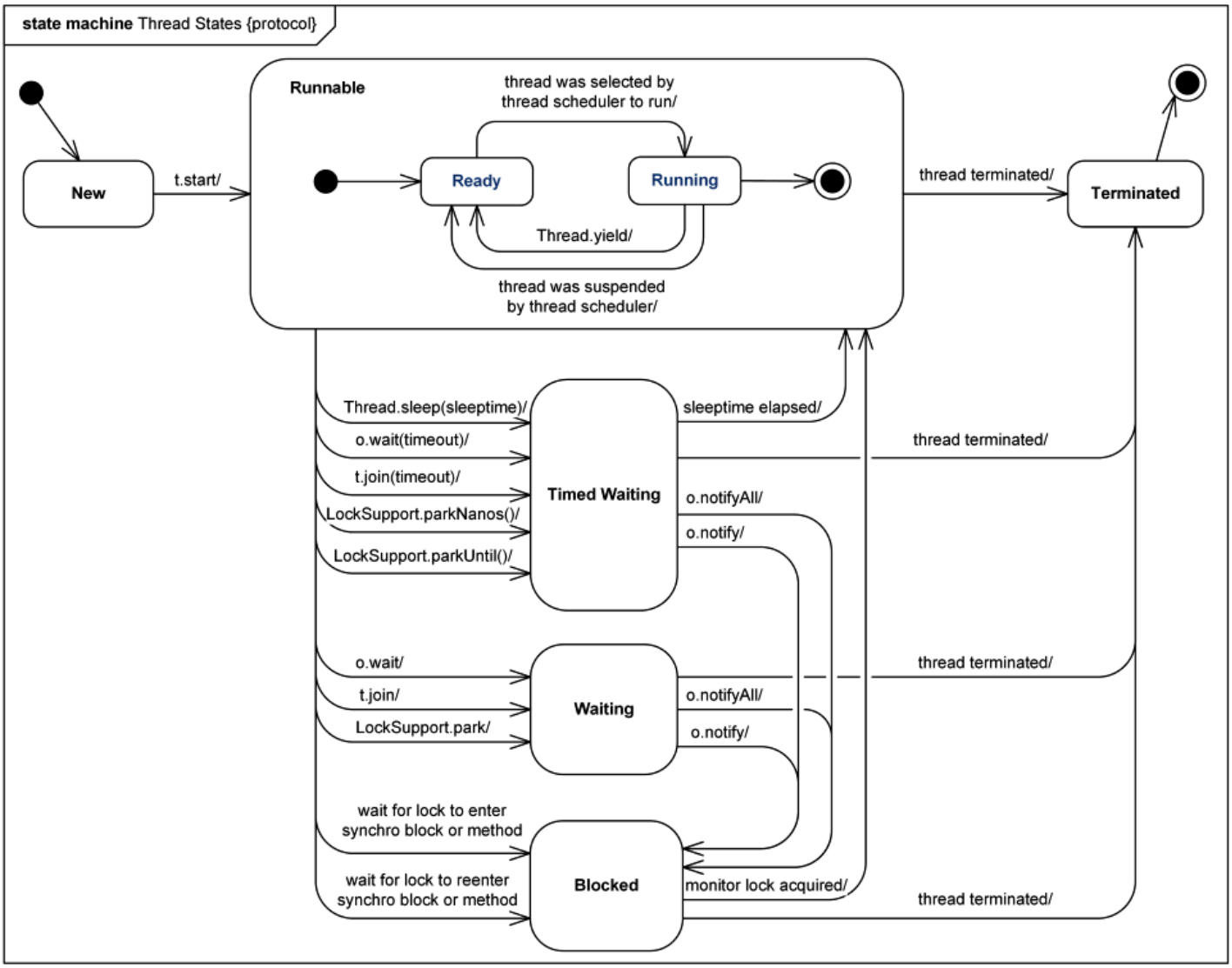
A thread state:

A thread can be in one of the following states:

- **NEW**
  A thread that has not yet started is in this state.
- **RUNNABLE**
  A thread executing in the Java virtual machine is in this state.
- **BLOCKED**
  A thread that is blocked waiting for a monitor lock is in this state.

- **WAITING**
  A thread that is waiting indefinitely for another thread to **perform** a particular action is in this state.
- **TIMED_WAITING**
  A thread that is waiting for another thread to **perform** an action for up to a specified waiting time is in this state.
- **TERMINATED**
  A thread that has **exited** is in this state.

A thread can be in only one state at a given point in time. These states are **virtual machine states** which do not reflect **any operating system thread** states.



Android's concurrency tools and mechanisms

1. **Threads**

   Java Threads: The basic unit of **concurrency in Java**. You can create a new thread by extending the **Thread class** or implementing the **Runnable interface**.

2. **AsyncTask**

   AsyncTask: Simplifies the use of **threads** and **handlers**. Designed to be a **quick** and **easy** way to perform **background operations** and **publish results on the UI thread.**

   Note: **AsyncTask** is deprecated in Android API level 30 and higher due to its limitations and potential for misuse**.**

3. **Handlers and HandlerThread**

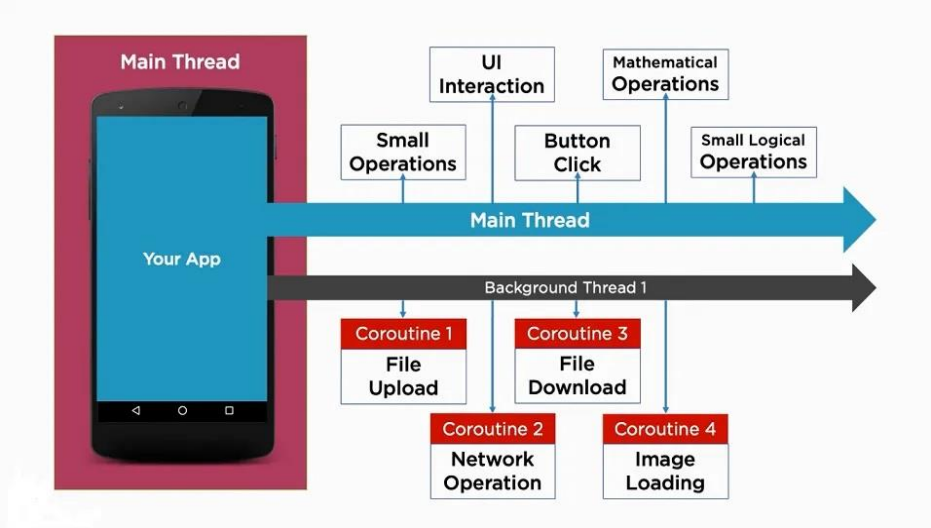
   - **Handler**: Allows you to **send** and process **Message** and **Runnable** objects associated with a **thread's MessageQueue**.
   - **HandlerThread**: A handy subclass of Thread that has its own **Looper**.

4. **Executors**

   Executor Framework: Provides a **higher-level replacement** for **working** with **threads directly**. The **Executors class** provides **factory methods** for creating different types of **thread pools**.

5. **Coroutines (Kotlin)**

   Coroutines: Provide a way to write **asynchronous code** in a **sequential manner**. **Kotlin coroutines** are a powerful tool for managing concurrency.



Summary

- **Concurrent software helps**
  - Leverage advances in hardware technology
  - Meet the quality & performance needs of apps & services
- **Successful concurrent software solutions must address key accidental & inherent complexities arising from**
  - Limitations with development tools/techniques
  - Fundamental domain challenges
- Some concurrency mechanisms provided by Android are based on standard Java threading classes
- Java Threads are implemented using various methods & functions defined by lower layers of the Android software stack
- A thread can be in only one state at a given point in time
- **Android's concurrency tools and mechanisms**