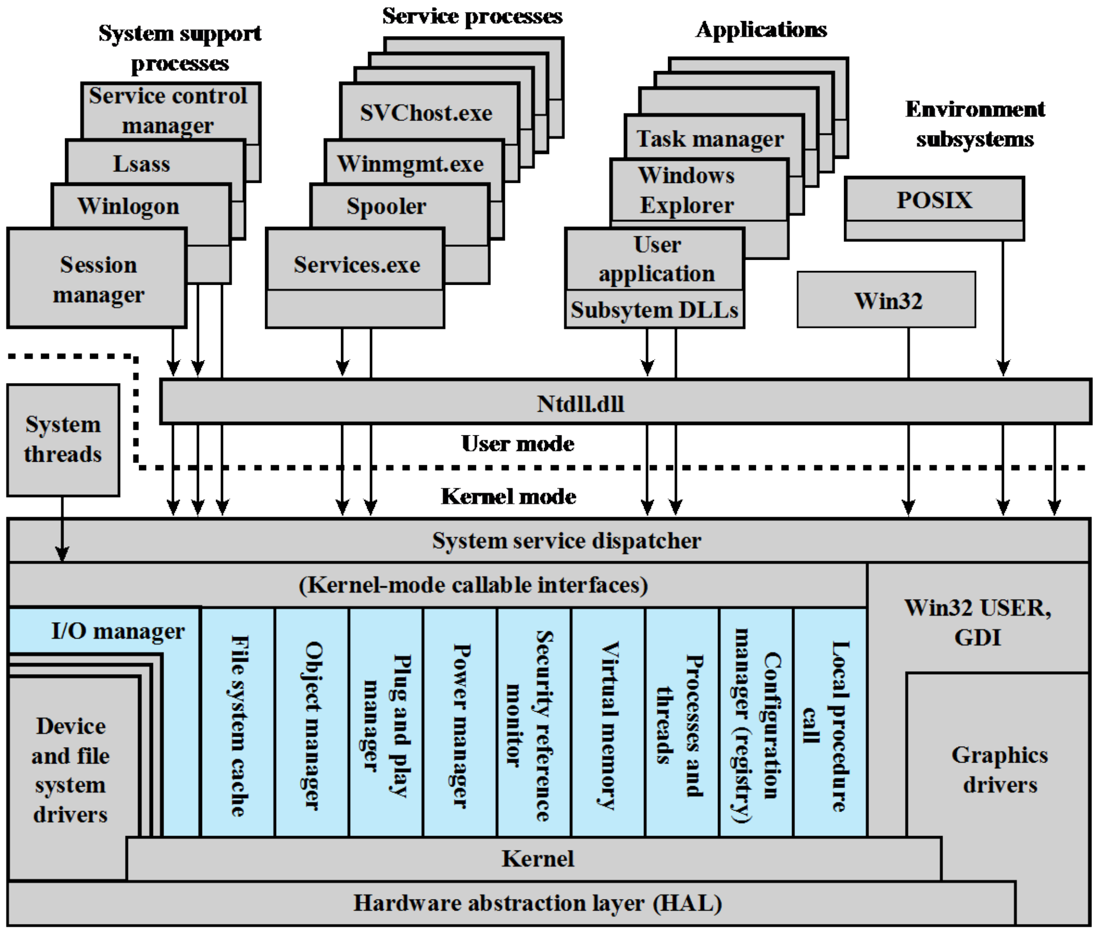
Week 3
Real Operating Systems



Lsass = local security authentication server
POSIX = portable operating system interface
GDI = graphics device interface
DLL = dynamic link libraries

Colored area indicates Executive

Figure 2.14 Windows Architecture

Kernel-Mode Components of Windows

- Executive
  - contains the core OS services
- Kernel
  - controls execution of the processors
- Hardware Abstraction Layer (HAL)
  - maps between generic hardware commands and responses and those unique to a specific platform
- Device Drivers
  - dynamic libraries that extend the functionality of the Executive
- Windowing and Graphics System
  - implements the GUI functions

User-Mode Processes

- Four basic types are supported by Windows:

Special System Processes	• user-mode services needed to manage the system
Service Processes	• the printer spooler, event logger, and user-mode components that cooperate with device drivers, and various network services
Environment Subsystems	• provide different OS personalities (environments)
User Applications	• executables (EXEs) and DLLs that provide the functionality users run to make use of the system

Client/Server Model

- Windows OS services, environmental subsystems, and applications are all structured using the client/server model
  - Common in distributed systems, but can be used internal to a single system
  - Processes communicate via RPC

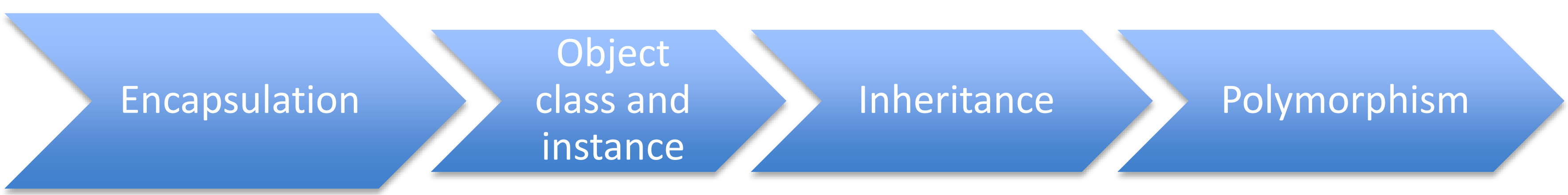
- Advantages:
  - it simplifies the Executive
  - it improves reliability
  - it provides a uniform means for applications to communicate with services via RPCs without restricting flexibility
  - it provides a suitable base for distributed computing

Threads and SMP

- Two important characteristics of Windows are its support for threads and for symmetric multiprocessing (SMP)
  - OS routines can run on any available processor, and different routines can execute simultaneously on different processors
  - Windows supports the use of multiple threads of execution within a single process. Multiple threads within the same process may execute on different processors simultaneously
  - server processes may use multiple threads to process requests from more than one client simultaneously
  - Windows provides mechanisms for sharing data and resources between processes and flexible interprocess communication capabilities

Windows Objects

- Windows draws heavily on the concepts of object-oriented design
- Key object-oriented concepts used by Windows are:



Asynchronous Procedure Call	Used to break into the execution of a specified thread and to cause a procedure to be called in a specified processor mode.
Deferred Procedure Call	Used to postpone interrupt processing to avoid delaying hardware interrupts. Also used to implement timers and inter-processor communication
Interrupt	Used to connect an interrupt source to an interrupt service routine by means of an entry in an Interrupt Dispatch Table (IDT). Each processor has an IDT that is used to dispatch interrupts that occur on that processor.
Process	Represents the virtual address space and control information necessary for the execution of a set of thread objects. A process contains a pointer to an address map, a list of ready threads containing thread objects, a list of threads belonging to the process, the total accumulated time for all threads executing within the process, and a base priority.
Thread	Represents thread objects, including scheduling priority and quantum, and which processors the thread may run on.
Profile	Used to measure the distribution of run time within a block of code. Both user and system code can be profiled.

Table 2.5 Windows Kernel Control Objects

Traditional UNIX Systems

- Were developed at Bell Labs and became operational on a PDP-7 in 1970
- Incorporated many ideas from Multics
- PDP-11was a milestone because it first showed that UNIX would be an OS for all computers
- Next milestone was rewriting UNIX in the programming language C
    - demonstrated the advantages of using a high-level language for system code

- Was described in a technical journal for the first time in 1974

- First widely available version outside Bell Labs was Version 6 in 1976

- Version 7, released in 1978 is the ancestor of most modern UNIX systems

- Most important of the non-AT&T systems was UNIX BSD (Berkeley Software Distribution)

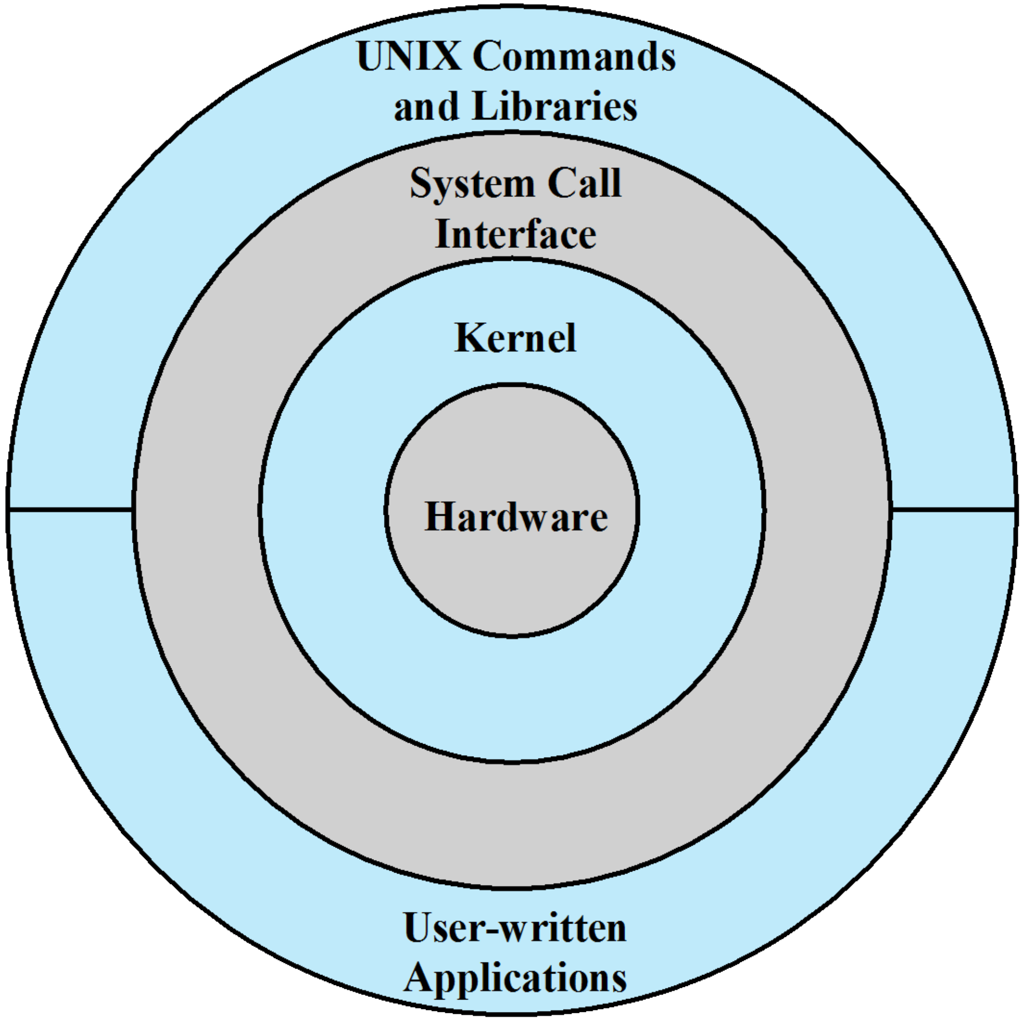


Figure 2.15 General UNIX Architecture

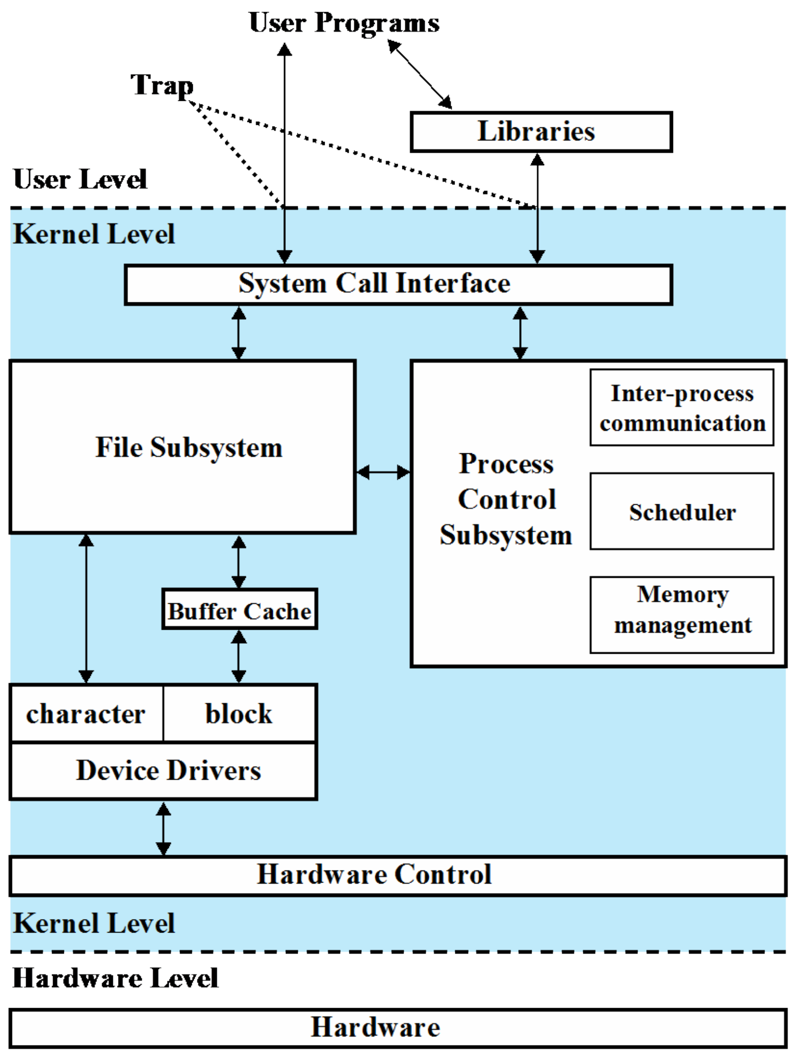


Figure 2.16 Traditional UNIX Kernel

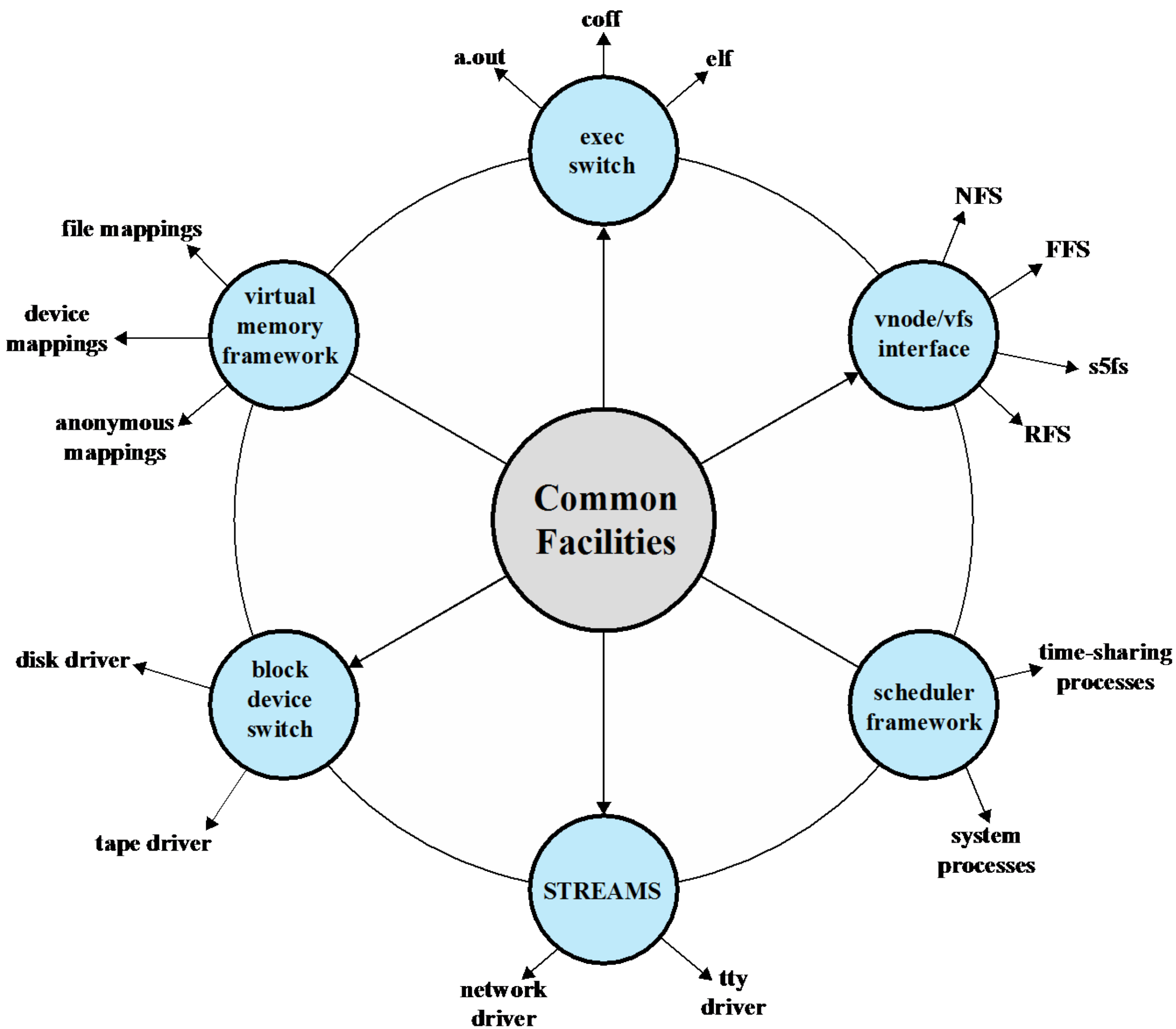


Figure 2.17 Modern UNIX Kernel [VAHA96]

System V Release 4 (SVR4)

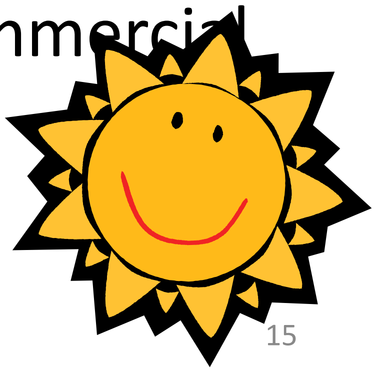
- Developed jointly by AT&T and Sun Microsystems
- Combines features from SVR3, 4.3BSD, Microsoft Xenix System V, and SunOS
- New features in the release include:
    - real-time processing support
    - process scheduling classes
    - dynamically allocated data structures
    - virtual memory management
    - virtual file system
    - preemptive kernel

BSD

- Berkeley Software Distribution
- 4.xBSD is widely used in academic installations and has served as the basis of a number of commercial UNIX products
- 4.4BSD was the final version of BSD to be released by Berkeley
    - major upgrade to 4.3BSD
    - includes
        - a new virtual memory system
        - changes in the kernel structure
        - several other feature enhancements
- FreeBSD
    - one of the most widely used and best documented versions
    - popular for Internet-based servers and firewalls
    - used in a number of embedded systems
    - Mac OS X is based on FreeBSD 5.0 and the Mach 3.0 microkernel

Solaris 10

- Sun's SVR4-based UNIX release
- Provides all of the features of SVR4 plus a number of more advanced features such as:
  - a fully preemptable, multithreaded kernel
  - full support for SMP
  - an object-oriented interface to file systems
- Most widely used and most successful commercial UNIX implementation



LINUX Overview

- Started out as a UNIX variant for the IBM PC
- Linus Torvalds, a Finnish student of computer science, wrote the initial version
- Linux was first posted on the Internet in 1991
- Today it is a full-featured UNIX system that runs on several platforms
- Is free and the source code is available
- Key to success has been the availability of free software packages
- Highly modular and easily configured

Modular Monolithic Kernel

- Includes virtually all of the OS functionality in one large block of code that runs as a single process with a single address space
- All the functional components of the kernel have access to all of its internal data structures and routines
- Linux is structured as a collection of modules

Loadable Modules

- Relatively independent blocks
- A module is an object file whose code can be linked to and unlinked from the kernel at runtime
- A module is executed in kernel mode on behalf of the current process
- Have two important characteristics:
    - dynamic linking
    - stackable modules

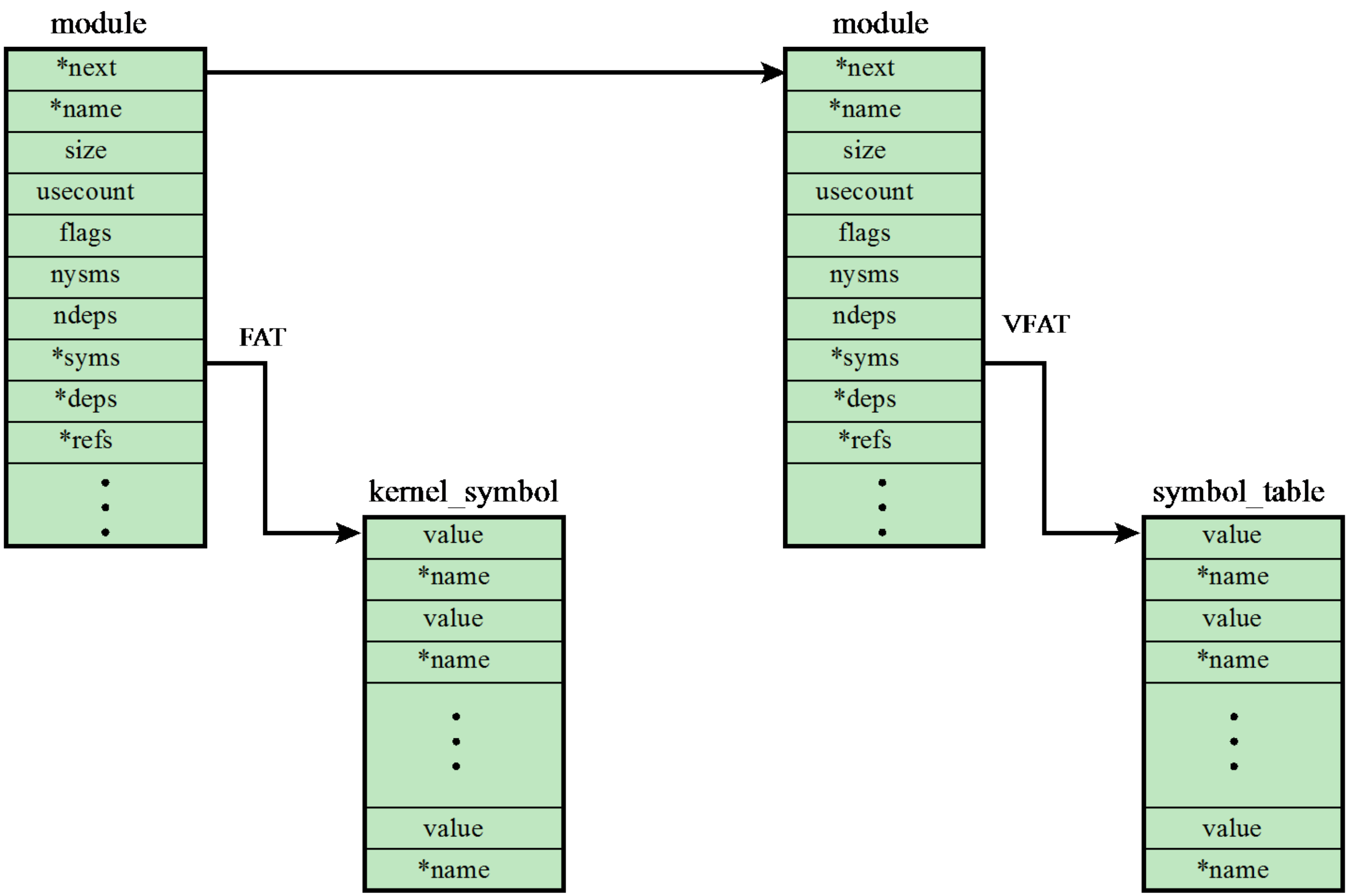


Figure 2.18 Example List of Linux Kernel Modules

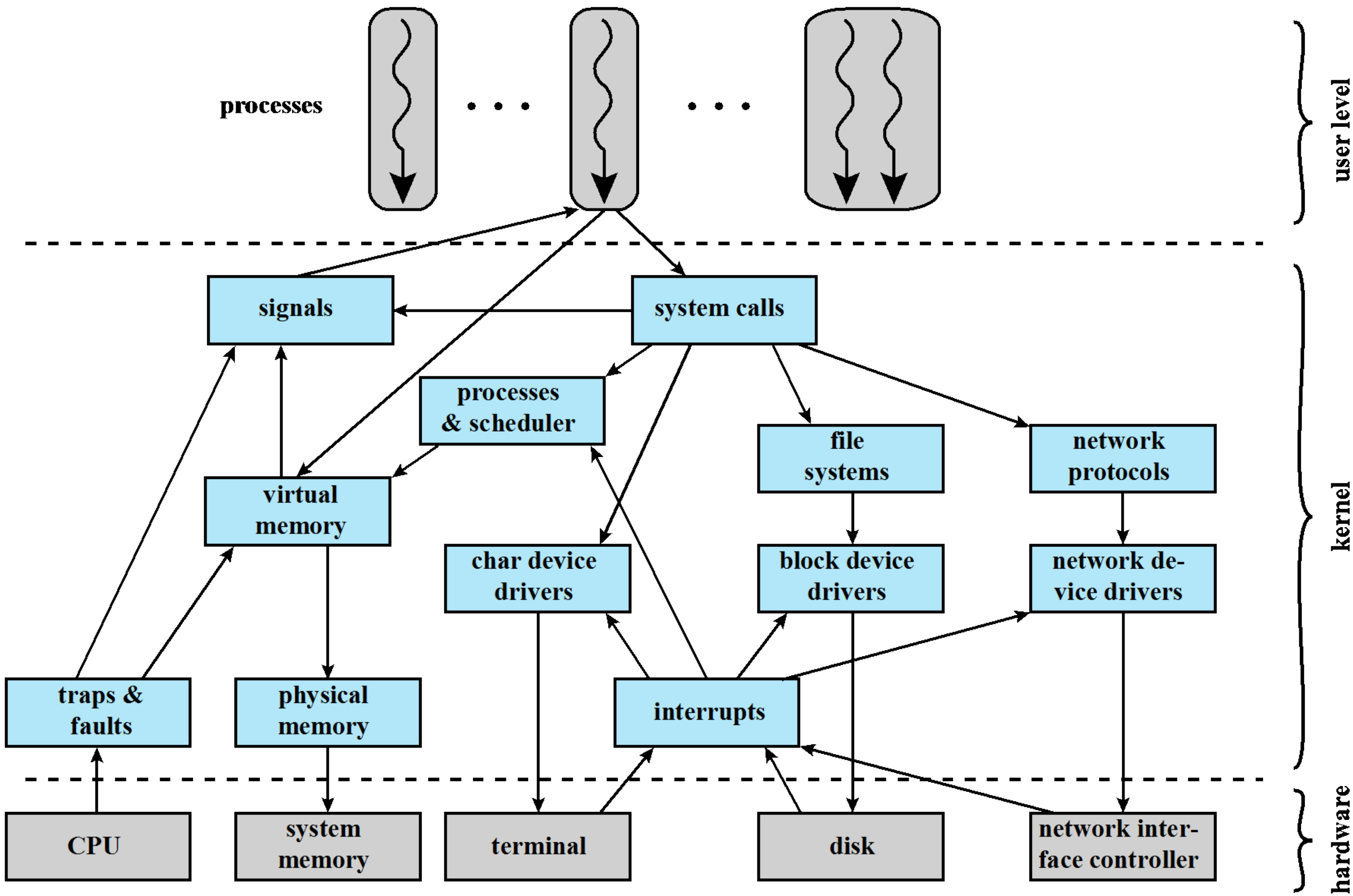


Figure 2.19 Linux Kernel Components

Linux Signals

SIGHUP	Terminal hangup	SIGCONT	Continue
SIGQUIT	Keyboard quit	SIGTSTP	Keyboard stop
SIGTRAP	Trace trap	SIGTTOU	Terminal write
SIGBUS	Bus error	SIGXCPU	CPU limit exceeded
SIGKILL	Kill signal	SIGVTALRM	Virtual alarm clock
SIGSEGV	Segmentation violation	SIGWINCH	Window size unchanged
SIGPIPT	Broken pipe	SIGPWR	Power failure
SIGTERM	Termination	SIGRTMIN	First real-time signal
SIGCHLD	Child status unchanged	SIGRTMAX	Last real-time signal

Table 2.6 Some Linux Signals

Filesystem related	
close	Close a file descriptor.
link	Make a new name for a file.
open	Open and possibly create a file or device.
read	Read from file descriptor.
write	Write to file descriptor
Process related	
execve	Execute program.
exit	Terminate the calling process.
getpid	Get process identification.
setuid	Set user identity of the current process.
ptrace	Provides a means by which a parent process my observe and control the execution of another process, and examine and change its core image and registers.
Scheduling related	
sched_getparam	Sets the scheduling parameters associated with the scheduling policy for the process identified by `pid`.
sched_get_priority_max	Returns the maximum priority value that can be used with the scheduling algorithm identified by `policy`.
sched_setscheduler	Sets both the scheduling policy (e.g., FIFO) and the associated parameters for the process `pid`.
sched_rr_get_interval	Writes into the timespec structure pointed to by the parameter `tp` the round robin time quantum for the process `pid`.
sched_yield	A process can relinquish the processor voluntarily without blocking via this system call. The process will then be moved to the end of the queue for its static priority and a new process gets to run.

Table 2.7 Some Linux System Calls (page 1 of 2)

Interprocess Communication (IPC) related	
msgrcv	A message buffer structure is allocated to receive a message. The system call then reads a message from the message queue specified by msqid into the newly created message buffer.
semctl	Performs the control operation specified by cmd on the semaphore set semid.
semop	Performs operations on selected members of the semaphore set semid.
shmat	Attaches the shared memory segment identified by shmid to the data segment of the calling process.
shmctl	Allows the user to receive information on a shared memory segment, set the owner, group, and permissions of a shared memory segment, or destroy a segment.
Socket (networking) related	
bind	Assigns the local IP address and port for a socket. Returns 0 for success and –1 for error.
connect	Establishes a connection between the given socket and the remote socket associated with sockaddr.
gethostname	Returns local host name.
send	Send the bytes contained in buffer pointed to by *msg over the given socket.
setsockopt	Sets the options on a socket
Miscellaneous	
fsync	Copies all in-core parts of a file to disk, and waits until the device reports that all parts are on stable storage.
time	Returns the time in seconds since January 1, 1970.
vhangup	Simulates a hangup on the current terminal. This call arranges for other users to have a "clean" tty at login time.

Table 2.7 Some Linux System Calls (page 2 of 2)

Android Operating System

- A Linux-based system originally designed for touchscreen mobile devices such as smartphones and tablet computers
- The most popular mobile OS
- Development was done by Android Inc., which was bought by Google in 2005
- 1st commercial version (Android 1.0) was released in 2008
- Most recent version is Android 4.3 (Jelly Bean)
- The Open Handset Alliance (OHA) was responsible for the Android OS releases as an open platform
- The open-source nature of Android has been the key to its success

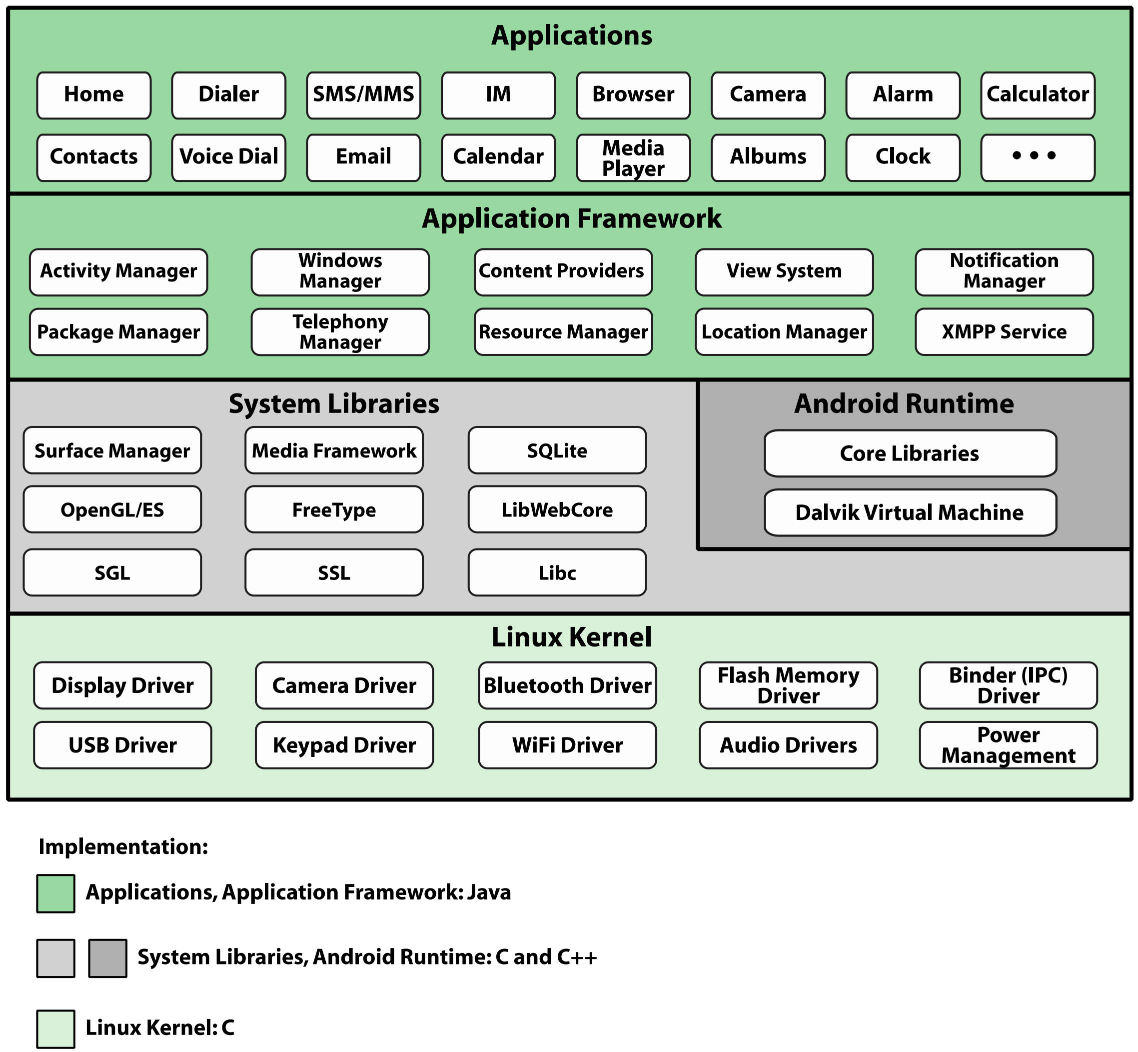


Figure 2.20 Android Software Architecture

Application Framework

- Provides high-level building blocks accessible through standardized API's that programmers use to create new apps
  - architecture is designed to simplify the reuse of

Activity Manager

- Manages lifecycle of applications
- Responsible for starting, stopping, and resuming the various applications

Window Manager

- Java abstraction of the underlying Surface Manager
- Allows applications to declare their client area and use features like the status bar

Package Manager

- Installs and removes applications

Telephony Manager

- Allows interaction with phone, SMS, and MMS services

Application Framework (cont.)

- Key components: (cont.)
  - Content Providers
    - these functions encapsulate application data that need to be shared between applications such as contacts
  - Resource Manager
    - manages application resources, such as localized strings and bitmaps
  - View System
    - provides the user interface (UI) primitives as well as UI Events
  - Location Manager
    - allows developers to tap into location-based services, whether by GPS, cell tower IDs, or local Wi-Fi databases
  - Notification Manager
    - manages events, such as arriving messages and appointments
  - XMPP
    - provides standardized messaging functions between applications

System Libraries

- Collection of useful system functions written in C or C++ and used by various components of the Android system
- Called from the application framework and applications through a Java interface
- Exposed to developers through the Android application framework
- Some of the key system libraries include:
    - Surface Manager
    - OpenGL
    - Media Framework
    - SQL Database
    - Browser Engine
    - Bionic LibC

Android Runtime

- Every Android application runs in its own process with its own instance of the Dalvik virtual machine (DVM)
- DVM executes files in the Dalvik Executable (.dex) format
- Component includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language
- To execute an operation the DVM calls on the corresponding C/C++ library using the Java Native Interface (JNI)

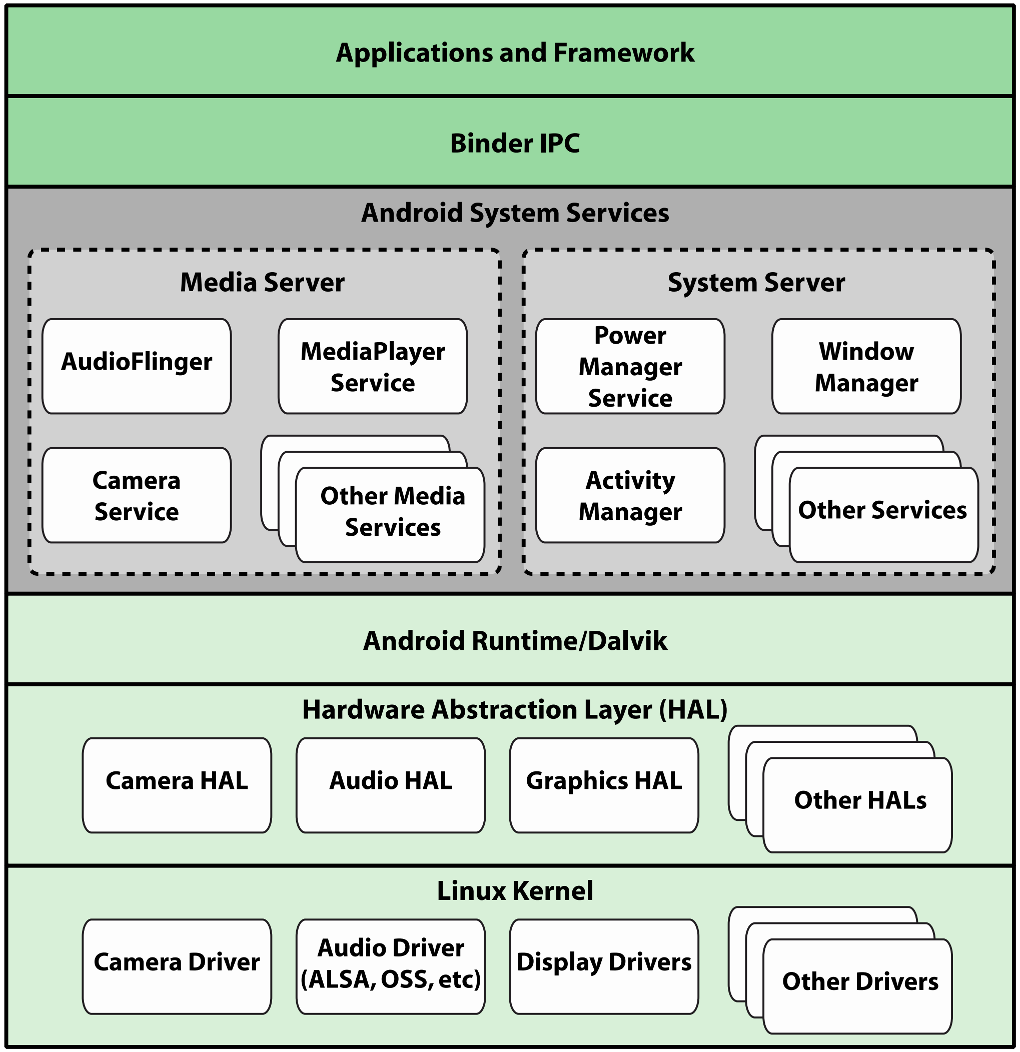


Figure 2.21 Android System Architecture

Activities

- An activity is a single visual user interface component, including things such as menu selections, icons, and checkboxes
- Every screen in an application is an extension of the Activity class
- Use Views to form graphical user interfaces that display information and respond to user actions

Power Management

Alarms

- Implemented in the Linux kernel and is visible to the app developer through the AlarmManager in the RunTime core libraries
- Is implemented in the kernel so that an alarm can trigger even if the system is in sleep mode
    - this allows the system to go into sleep mode, saving power, even though there is a process that requires a wake up

Wakelocks

- Prevents an Android system from entering into sleep mode
- These locks are requested through the API whenever an application requires one of the managed peripherals to remain powered on
- An application can hold one of the following wakelocks:
    - Full_Wake_Lock
    - Partial_Wake_Lock
    - Screen_Dim_Wake_Lock
    - Screen_Bright_Wake_Lock

Fundamental Concepts

- In window **processes** are **containers for programs.**
- Each process includes:
  – **Virtual address space**
  – **Handles to kernel-mode objects**
  – **Threads** and **resources** to threads execution
- Each process have user-mode system data called the **process environment block (PEB)**, includes:
  – List of loaded modules
  – The current working directory
  – Pointer to process’ heaps

Fundamental Concepts

- There is also a **threads environment block (TEB)** that keeps user-mode data, includes:
  - **Thread local storages**
  - Fields
- Another data structure that kernel-mode shared is **user shared data** which is **contains** various **form of time**, **version info**, amount of **physical memory**, number of shared **flags**.

Fundamental Concepts

- **Threads** are kernel's abstraction for scheduling the CPU
- Threads can also be **affinitized** to only run on **certain processors**
- Each thread **has two separate call stacks**, one **for** execution in **user-mode and** one for **kernel-mode**

What is a process?

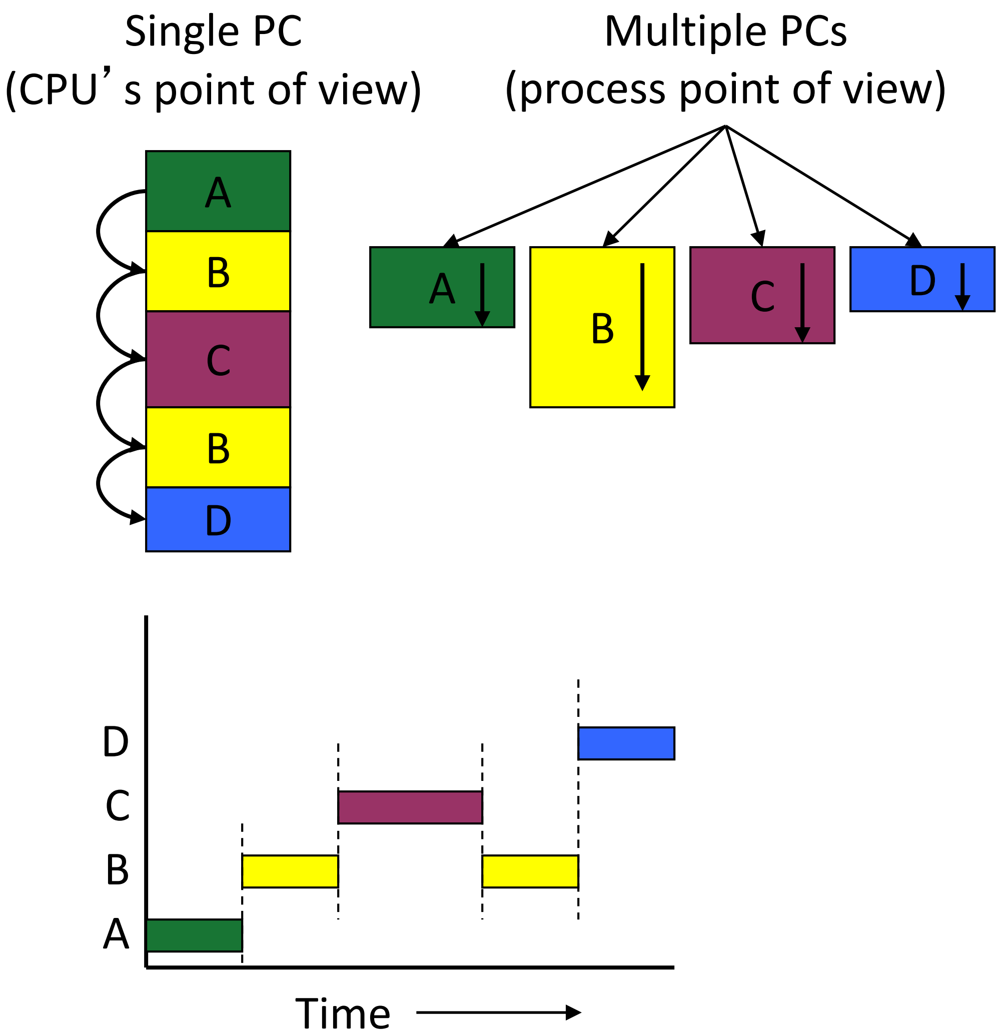
- Code, data, and stack
  - Usually (but not always) has its own address space
- Program state
  - CPU registers
  - Program counter (current location in the code)
  - Stack pointer
- Only one process can be running in the CPU at any given time!

Fundamental Concepts

Process

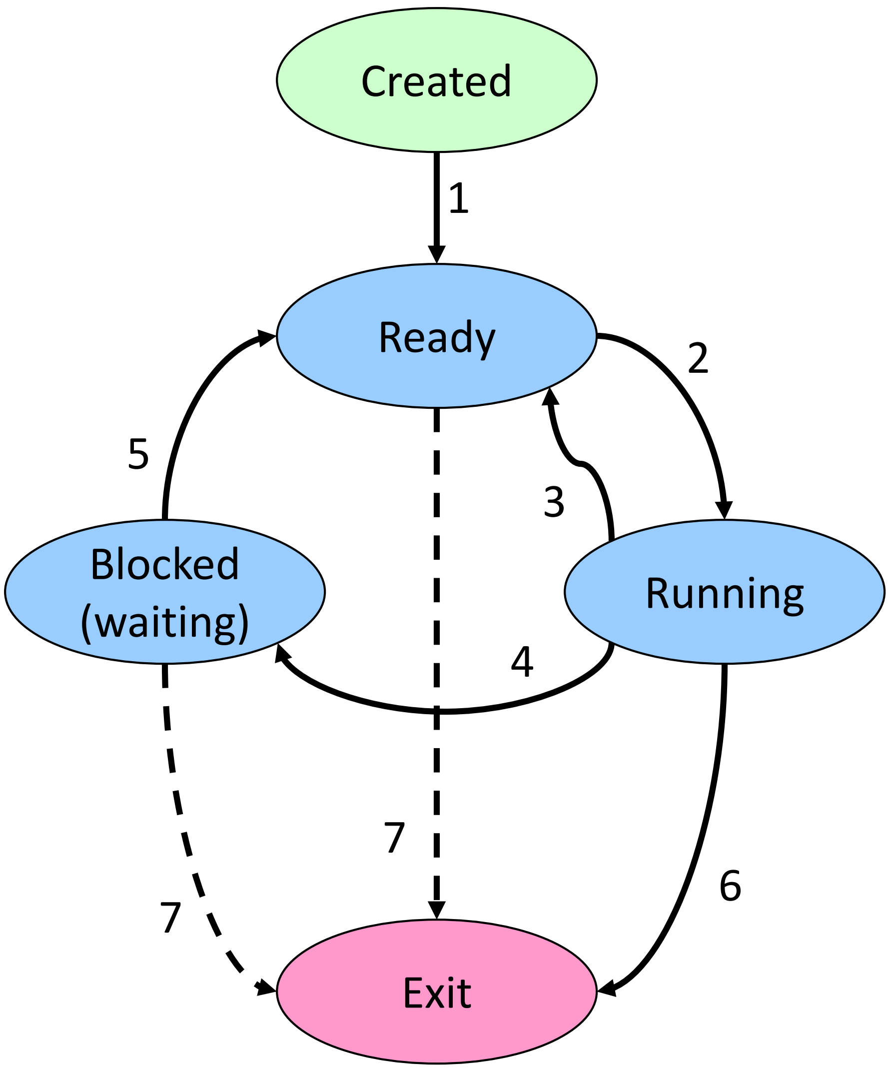
- Process are created from section objects, each of which describes a memory object backed by a file on disk.
- Create process:
  - Modify a new process by mapping section
  - Allocating virtual memory
  - Writing parameters and environmental data
  - Duplicating file descriptors
  - Creating threads.

The process model



- Multiprogramming of four programs
- Conceptual model
  - 4 independent processes
  - Processes run sequentially
- Only one program active at any instant!
  - That instant can be very short…

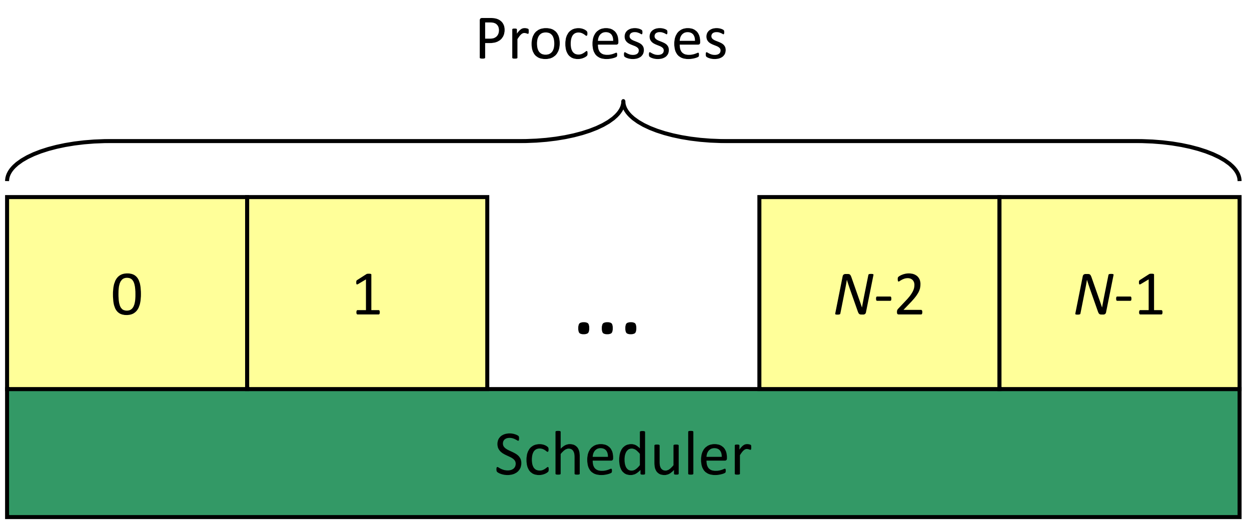
Process states



- Process in one of 5 states
  - Created
  - Ready
  - Running
  - Blocked
  - Exit
- Transitions between states
  1 - Process enters ready queue
  2 - Scheduler picks this process
  3 - Scheduler picks a different process
  4 - Process waits for event (such as I/O)
  5 - Event occurs
  6 - Process exits
  7 - Process ended by another process

Processes in the OS

- Two “layers” for processes
- Lowest layer of process-structured OS handles interrupts, scheduling
- Above that layer are sequential processes
  - Processes tracked in the *process table*
  - Each process has a *process table entry*



Fundamental Concepts

Jobs and Fibers

- **Definition**: **Jobs** is a group of processes.
- The main functions of a job is to constraints to the threads they contain such at:
  - Limiting resources
  - Prevents threads from accessing system objects by enforcing **restricted token**
- Once a process in a jobs, **all process threads** in those process create **will also be in the job**.
- Problems: **one process can be in one job**, there will be conflicts if many jobs attempt to manage the same process.

Fundamental Concepts

Fibers

- **Definition:** A fiber is a **unit** of **execution** that **must be** manually **scheduled** by the application
- **Fibers** are created by **allocating a stack** and **a user-mode fiber data** structure for storing registers and data can **also be created independently** of threads
- Fibers **will not run until another running fiber** in thread **make explicitly call** *SwithToFiber.*
- Advantage:
  - It easier and take fewer time to switch between fiber than threads

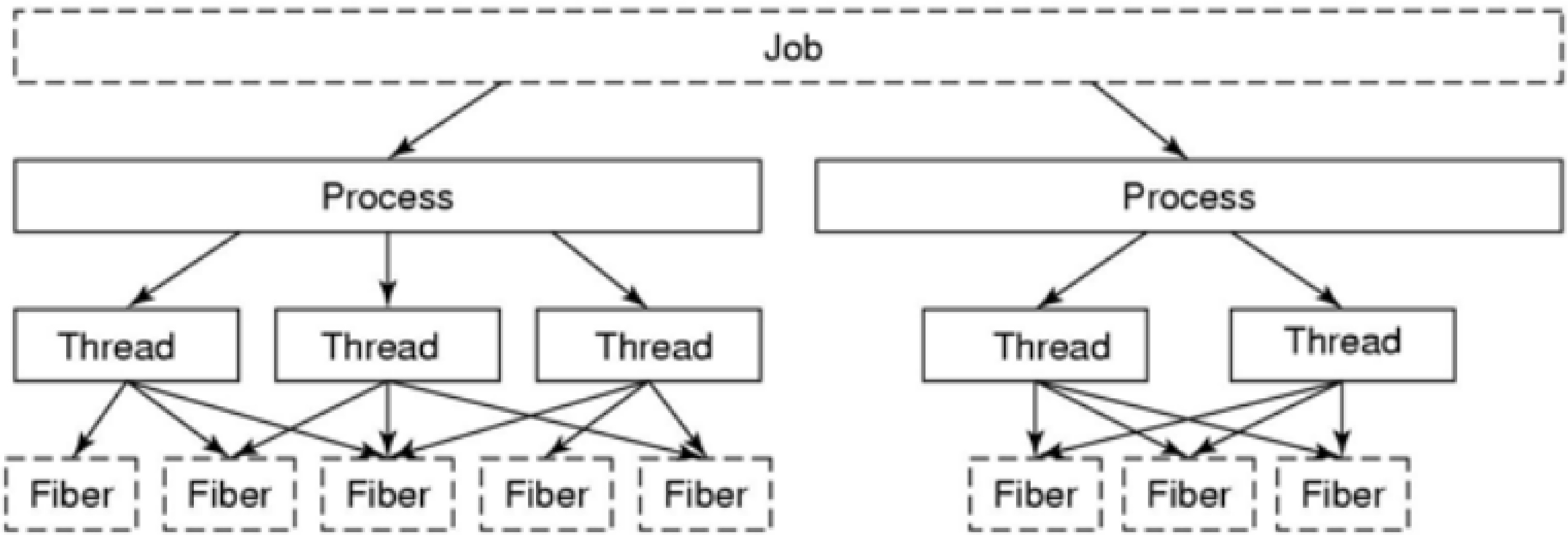
Fundamental Concepts

Jobs and Fibers

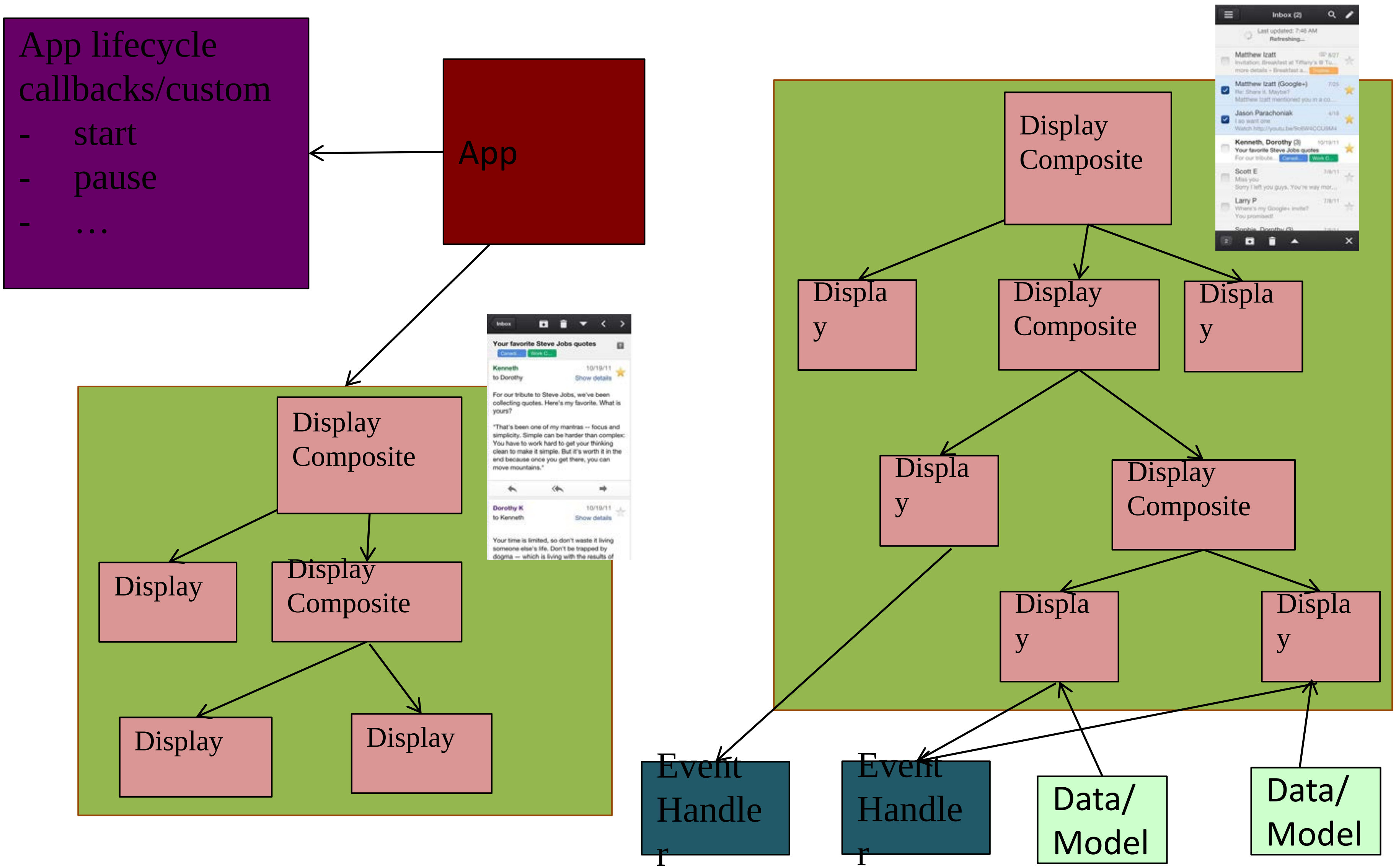
- Disadvantage: **need a lot of synchronization** to make sure fibers do not interface with each other.
- Solution: **create** only **as many threads as** there are processors to run them, and **affinitize** the **threads** to each **run only** on a **distinct set** of available **processors.**

Fundamental Concepts

Jobs and Fibers



Recap: Mobile GUI App Workflow

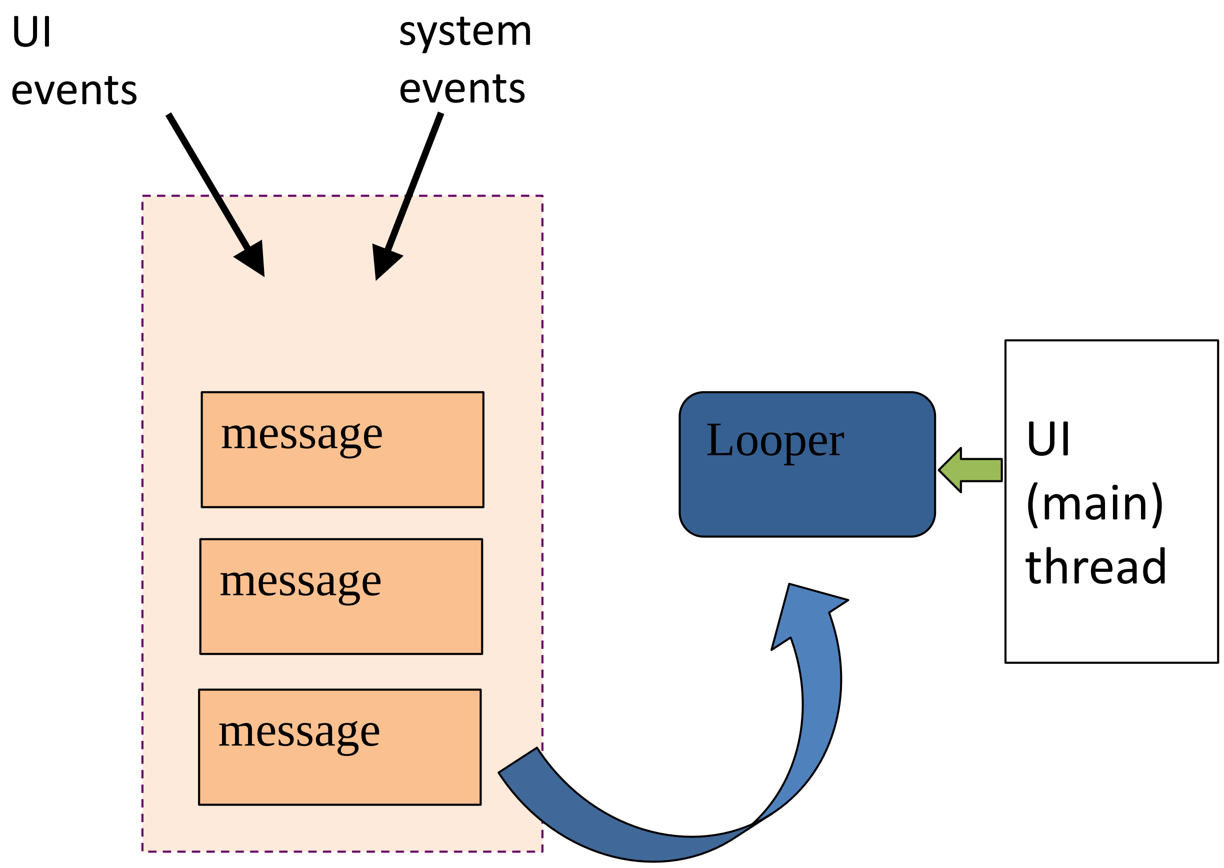


Recap: Android UI App Basic Concepts

- Activity
- View/ViewGroup
  - External definition of views in XML
  - findViewById() to reduce coupling
- Link view events to event handlers
  - set…Listener()

Event Handler Execution

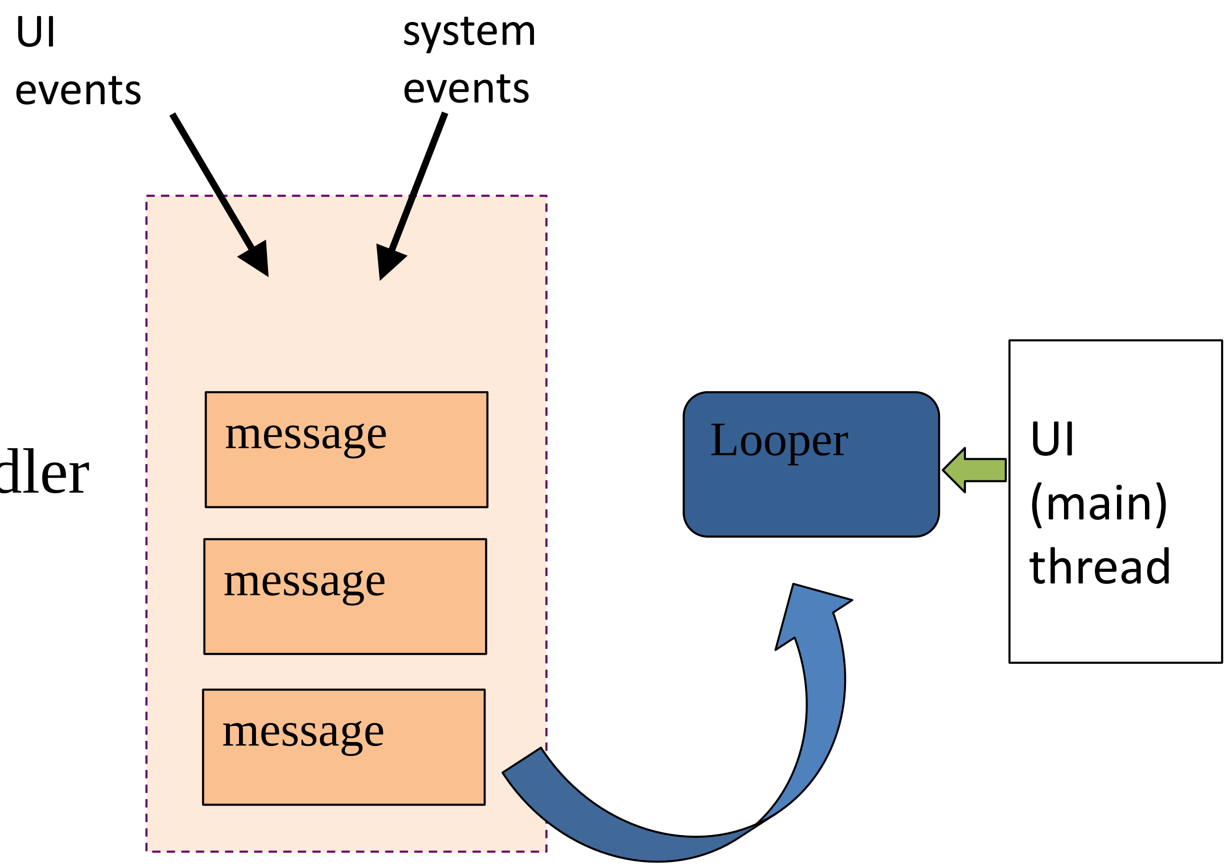
- Event handler executed by the main/UI thread



http://www.java2s.com/Open-Source/Android/android-core/platform-frameworks-base/android/os/Looper.java.htm

Event Handler and Responsiveness

- Event handler blo[cks]
  events in the msg
  queue from being
  processed
  =>
  slow running handler
  leads to no UI
  response



http://developer.android.com/guide/practices/responsiveness.html

Responsiveness: Numbers (Nexus One)

- ~5-25 ms – uncached flash reading a byte
- ~5-200+(!) ms – uncached flash *writing tiny amount*
- 100-200 ms – human perception of slow action
- 108/350/500/800 ms – ping over 3G. varies!
- ~1-6+ seconds – TCP setup + HTTP fetch of 6k over 3G

Process hierarchies

- Parent creates a child process
  - Child processes can create their own children
- Forms a hierarchy
  - UNIX calls this a “process group”
  - If a process exits, its children are “inherited” by the exiting process’s parent
- Windows has no concept of process hierarchy
  - All processes are created equal