Operating Systems
Introduction

MSc CO502
Autumn Term Weeks 7-11

Morris Sloman & Roman Kolcun

m.sloman@imperial.ac.uk
Room 572

Course Objectives

What is an operating system, and how it supports the implementation of software on a computer.

Understand the features and mechanisms that underlie operating systems, including:

- process and thread management and synchronization
- memory management
- security
- input-output
- file systems

Linux characteristics as a case study

Outline

Morris Sloman (12 lectures)

- Overview: function and structure
- Processes and Threads: concepts and scheduling
- Process Coordination: synchronization & deadlocks
- Memory Management: allocation and virtual memory
- Security: authentication and access control

Roman Kolcun (6 lectures)

- Input/Output: device drivers, disk management & scheduling
- File Systems: files and directory structures
- Virtual Machines Systems (if time)

Course Structure

Four lectures + 2 tutorials per week (Weeks 7 – 11)

Lectures: Mondays 9am LT145 , Tuesdays 2pm LT145,
Fridays 2-4pm LT 311

Tutorials: Mondays 10am LT 145, Fridays 4pm R341/2

Course slides are on Cate

Acknowledgements:

Slides based on material by Peter Pietzuch, Cristian Cadar and Julie McCann

Recommended Books

1. **Modern Operating Systems: Global Edition**, A. Tanenbaum, H. Bos, 4th edition, Pearson, 2015
2. **Operating Systems – Internals and Design Principles**, W. Stallings, 8th Edition, Pearson, 2014
3. **Operating System Concepts**, A. Silberschatz, P. Galvin, G. Gagne, 8th Edition, John Wiley & Sons, 2014

Note: Earlier editions of these are OK and may be more readily available

☛ Important: Do not just rely on these slides!

OS Overview

Computer Architecture Overview

Processor

– Controls computer hardware
  Executes instructions and programs

Memory

– Stores data and programs

I/O modules

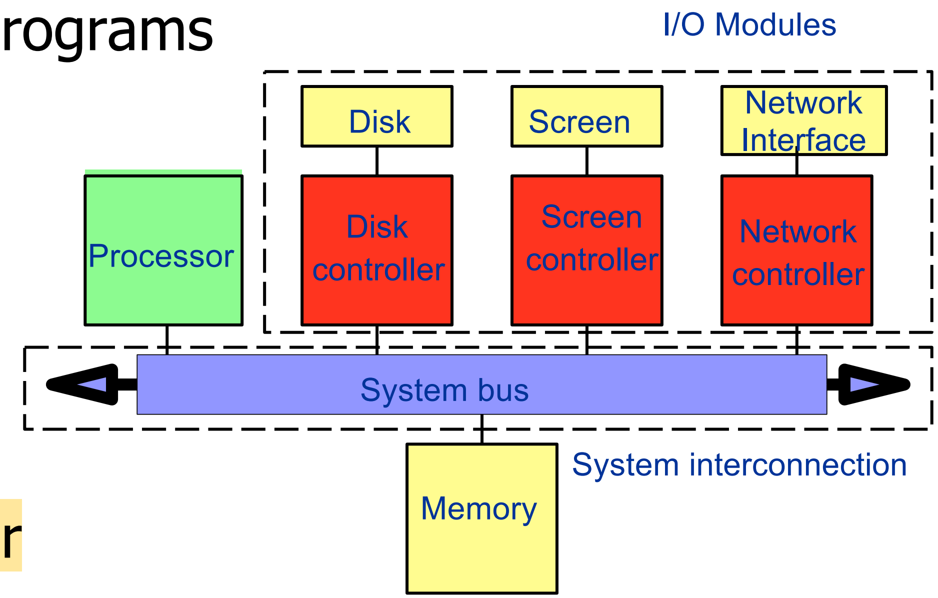
– Read and write from
  I/O devices

– Intelligence in I/O controller

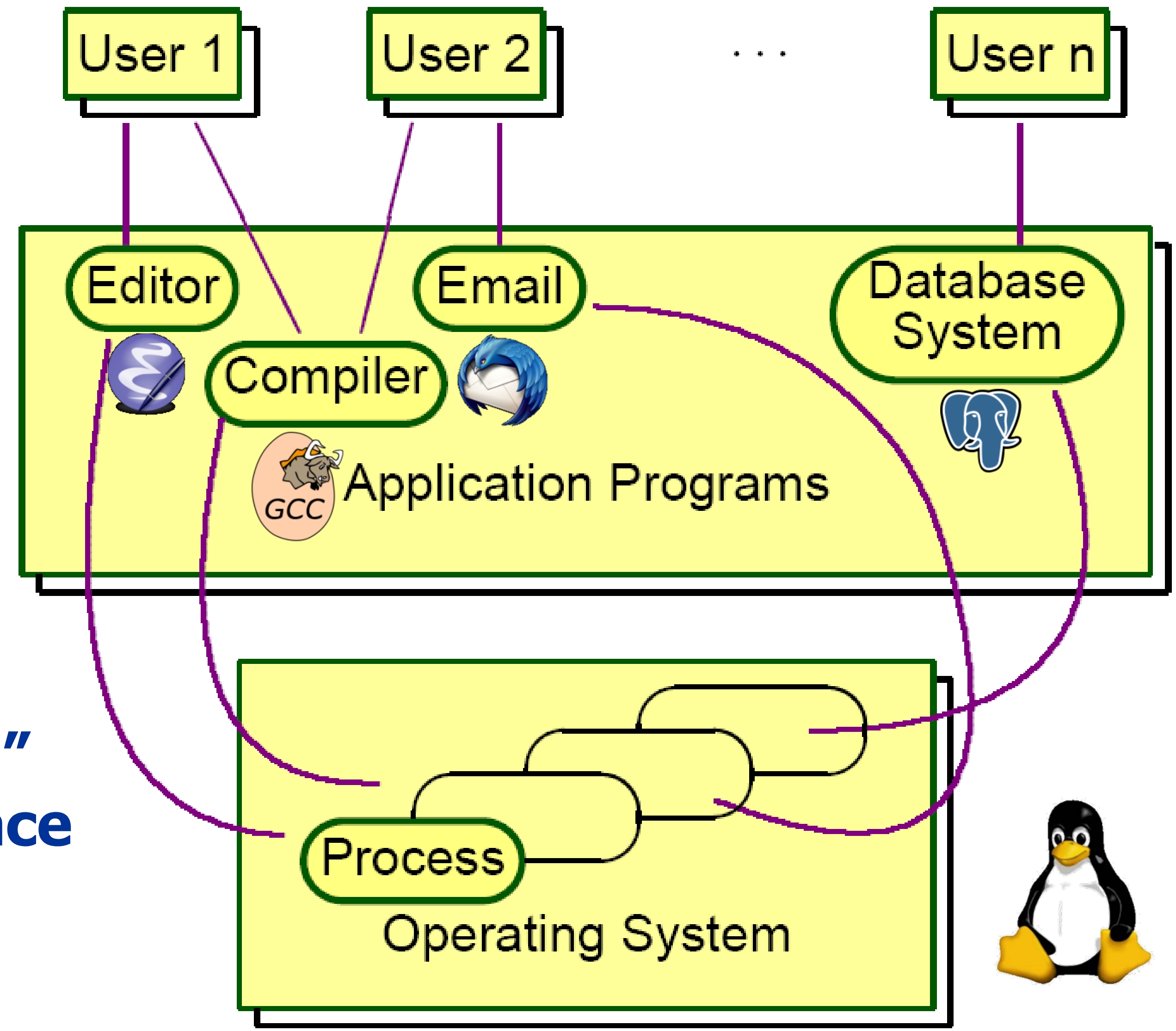
System interconnection

– Connects different hardware components via bus

– Provides communication between hardware components

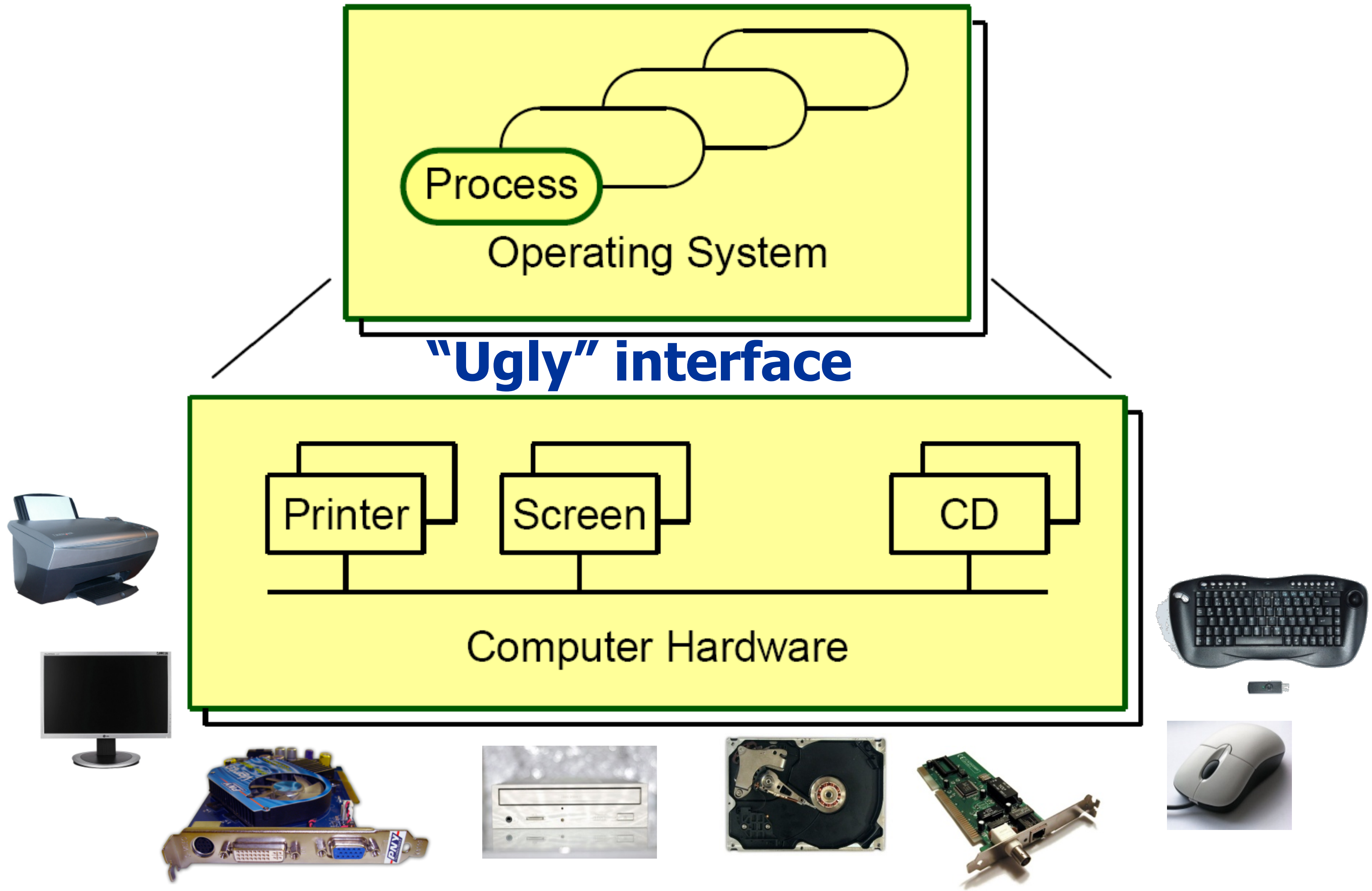


Operating Systems – Top Level View



"Clean" interface

Operating Systems – Bottom Level View



1. Resource Management

Making efficient use of (limited) available resources

- Optimise utilisation of processor, memory, disks, network etc....

Sharing resources among multiple users

- Schedule access, fair allocation
- Prevent interference

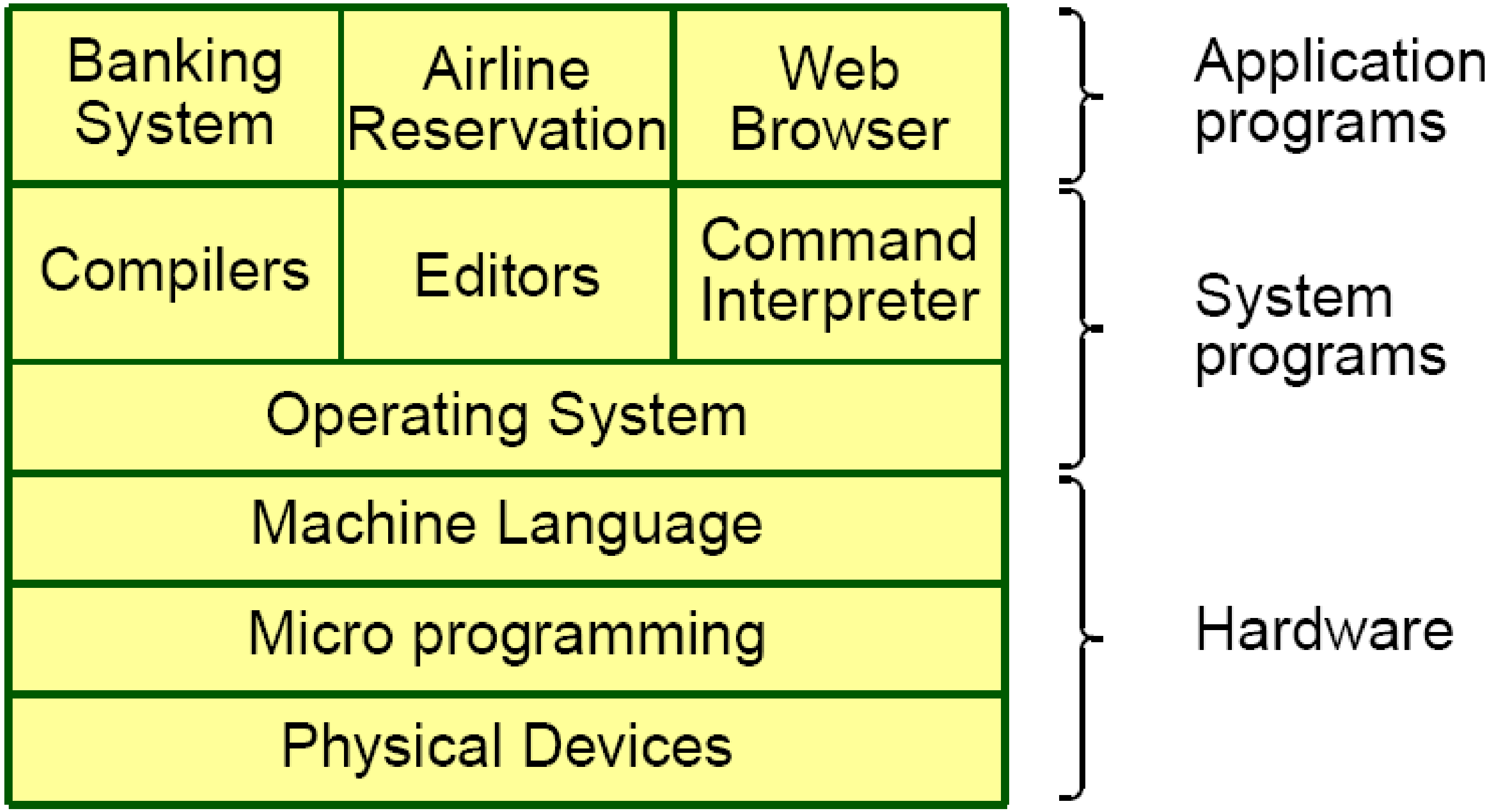
Resources

- Processors
  - Divide number and/or time
- Memory
  - RAM, cache, disks, ...
- Input/Output devices
  - Screens, printers, network interface, ...
- Internal devices
  - Clocks, timers, accelerometers ...
- Long-term storage (files)
  - Disks, storage cards, DVD, tapes, ...
- Software
  - Browsers, editors, e-mail clients, databases, …….

2. Providing Clean Interfaces

OS converts raw hardware into usable computer system

– Hides complexity of lower levels from higher levels



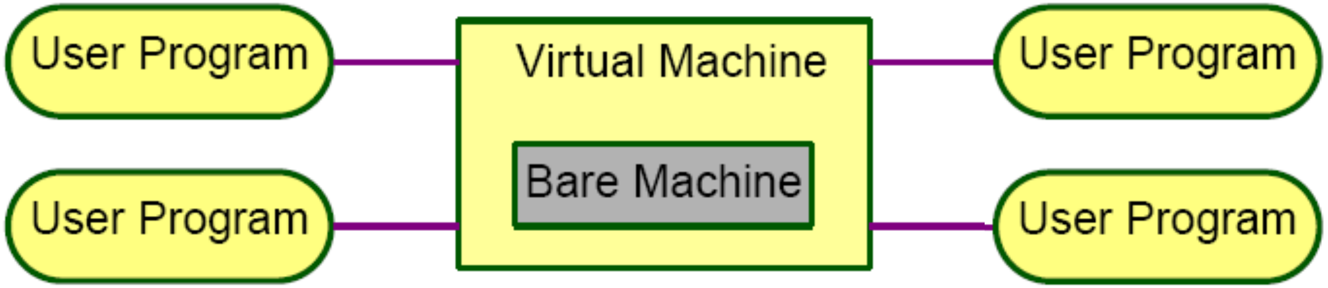
Virtual Machine Abstraction

Details of hardware kept hidden from programs

Only OS can allow access to hardware resources

User request should be abstract

- e.g. no need to know how files stored on disk



Virtual Machine Facilities

Simplified I/O: Device independence; open a file on disk, CD, screen is one operation.

Virtual Memory: Larger than real or partitioned.

Filing System: Long term storage, on disk or tape, accessed by symbolic names.

Program Interaction and communication: Pipes, semaphores, locks, monitors.

Network communication: Message passing

Protection: Prevent programs accessing resources not allocated to them.

Program Control: User interaction with programs, command language, shells.

Accounting & Management Information: Usage of processors, memory, file storage etc.

OS Characteristics: Sharing

Sharing of data, programs and hardware

- Time multiplexing and space multiplexing

Resource allocation

- Efficient and fair use of memory, CPU time, disk space, ...
- Simultaneous access to resources
  - Processor, Disks, RAM, code, network, ...
- Mutual exclusion
  - Protect multiple programs from uncontrolled access to shared resources.
  - Prevent multiple writes to same data structure or file.
- Protection against corruption
  - Accidental or malicious

OS Characteristics: Concurrency I

Several simultaneous parallel activities

- Overlapped I/O & computation
- Multiple users and programs run in parallel

Switch activities at arbitrary times

- Guarantee fairness and prompt response
- Differential responsiveness e.g. interactive vs. batch

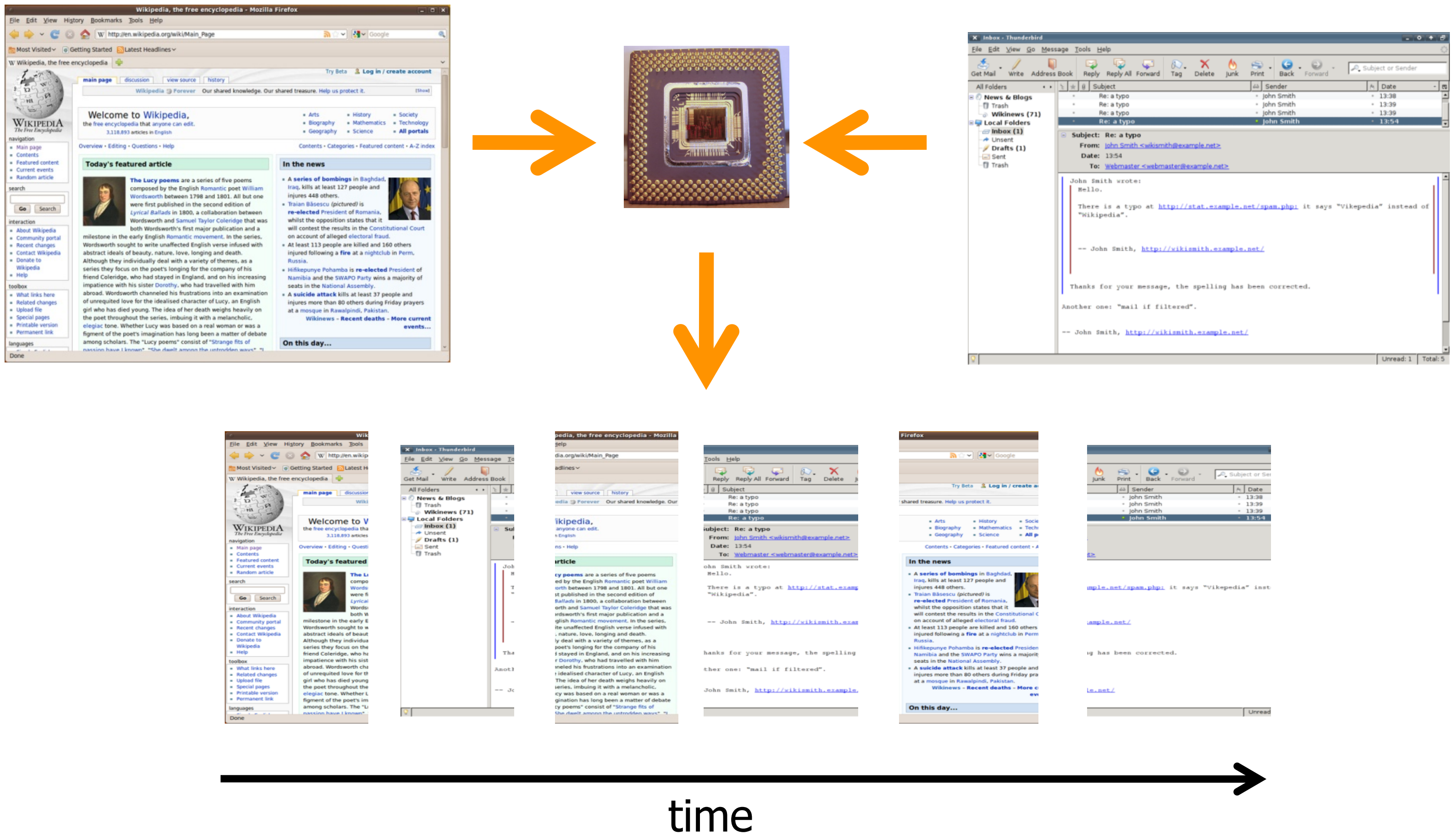
Safe concurrency

- Synchronisation of actions
  - Avoids long waiting cycles; gives accurate error handling
- Protection from interference
  - Each process has its own space

OS Characteristics: Concurrency II

Time-slicing

- Switch application running on physical CPU every 50ms



What other switching mechanisms are feasible?

OS Characteristics: Non-determinism

Non-determinism

- Results from events occurring in unpredictable order
  - e.g. timer interrupts, user input, program error, network packet loss, disk errors, . . .
- Makes programming OS hard!

OS Characteristics: Storing Data

Long term storage: File systems for disks, DVDs, memory cards …..

- Easy access to files through user-defined names
  - Directory structure, links, shared disks
- Access controls
  - Read, write, delete, execute or copy permissions
- Protection against failure (backups)
  - Daily/weekly/monthly, partial/complete
- Storage management for easy expansion
  - Add disks without need for re-compilation of OS

Operating System Zoo

Desktop/Laptop (e.g. Windows, Mac OS X, Linux)

- Typically 2-8 cores + high resolution screen

Server OS (e.g. Linux, Windows Server 20XX, Solaris, FreeBSD,)

- Share hardware/software resources e.g. internet servers
- Typically many multicore processors + large disks

Smartphones (e.g. iOS, Android)

- Simpler CPUs, starting to be sophisticated

Real-time OS

- Guaranteed time constraints

Embedded OS (e.g. QNX, VXWorks)

- Transport, communications, banking, homes etc.
- Only trusted software

Smart card OS

- Usually single function
- Many have JVM
- OS is primitive

Sensor Network OS (e.g. TinyOS)

- Resource/energy conscious

OS Structure

Monolithic OS kernels (e.g. Linux, BSD, Solaris, ...)

- Single black box

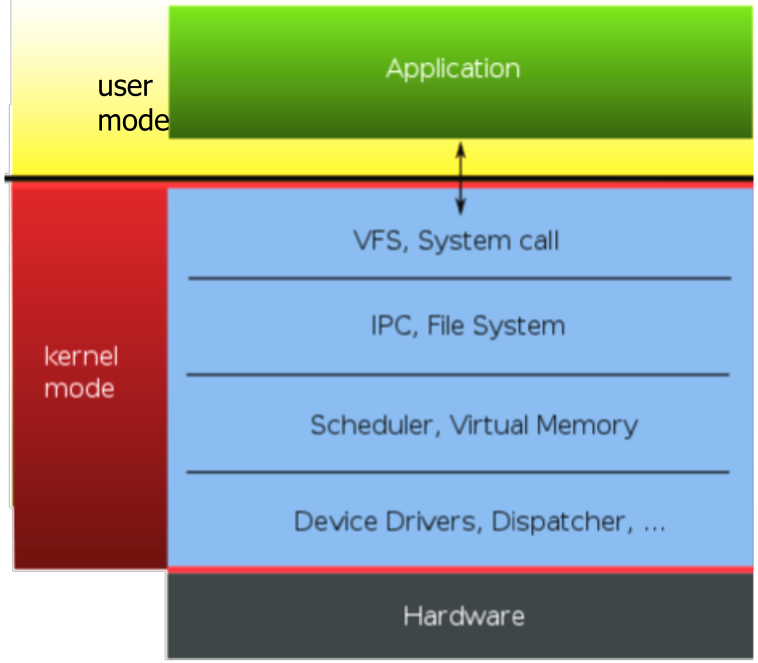
Microkernels (e.g. Symbian, L4, Mach, ...)

- Little as possible in kernel (fewer bugs)

Hybrid kernels (e.g. Windows NT, Mac OS X, ...)

- Take a guess... ☺

Monolithic Kernels



Kernel is single executable with own address space

- Structure implied through pushing parameters to stack and trap (systems calls)
- Most popular kernel style

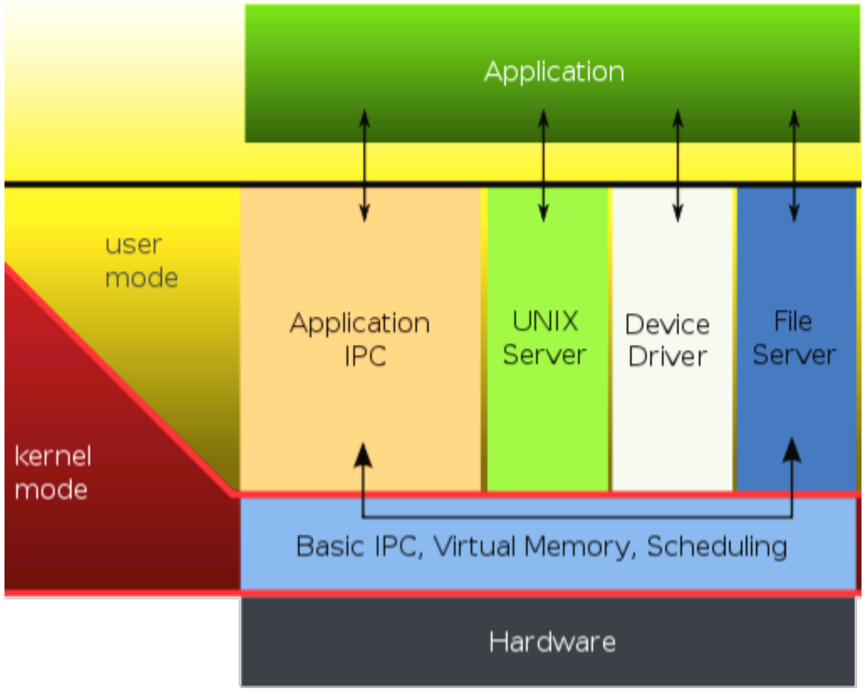
Advantages

- Efficient calls within kernel
- Easier to write kernel components due to shared memory

Disadvantages

- Complex design with lots of interactions
- No protection between kernel components

Microkernels



Minimal "kernel" with functionality in user-level servers

- Kernel does IPC (message-passing) between servers
- Servers for device I/O, file access, process scheduling, ...

Advantages

- Kernel itself not complex ➔ less error-prone
- Servers have clean interfaces
- Servers can crash and restart without bringing kernel down

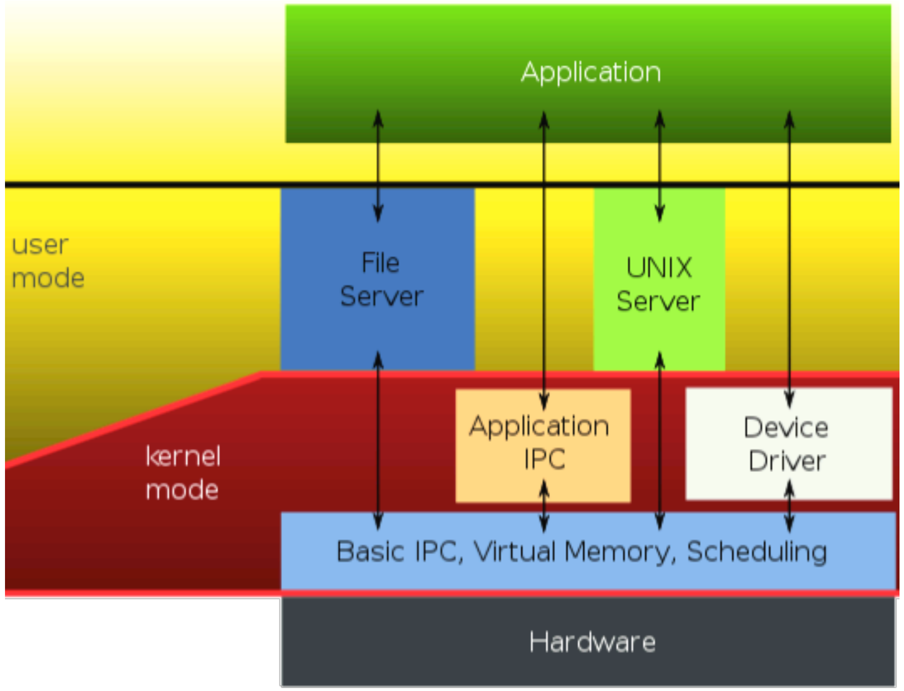
Disadvantages

- Overhead of IPC within kernel high

Hybrid Kernels

Combines features of both monolithic and microkernels

– Often a design philosophy



Advantages

– More structured design

Disadvantages

– Performance penalty for user-level servers

Introduction to Linux

Linux History and Motivation

Variant of Unix like FreeBSD, System V, Solaris etc.

- Ken Thomson left Multics (Bell Labs)
  – *Uniplexed* information and computing service
- Dennis Ritchie got interested

Late 80's: 4.3 BSD and System V r3 dominant

- Systems call libraries reconciliation POSIX

1987 Tanenbaum released MINIX microkernel

- Tractable by single person (student)

Linus Torvalds, frustrated, built fully-featured yet monolithic version ➔ Linux

- Major goal was interactivity, multiple processes and users
- Code contributed by world-wide community

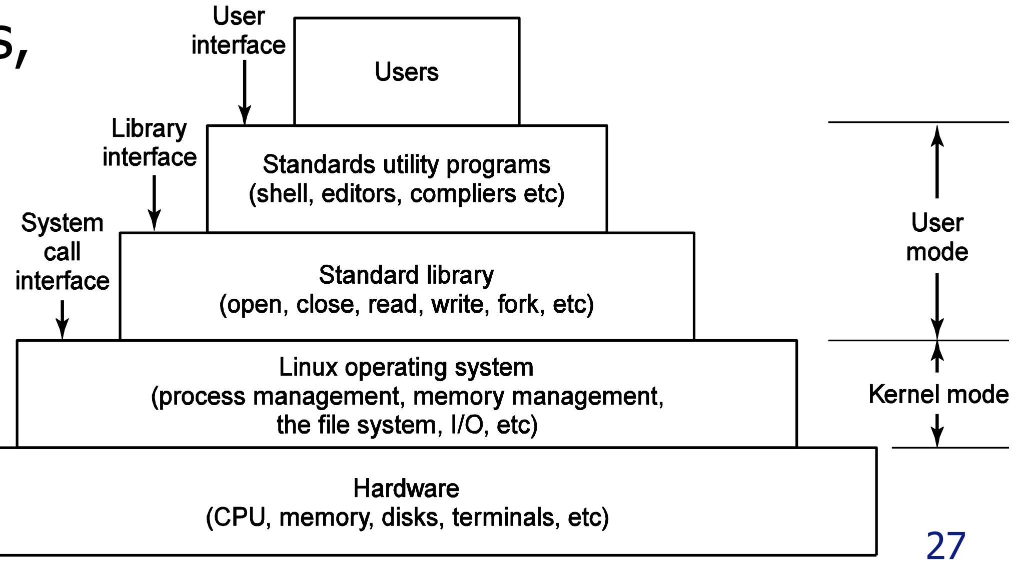
Structure and Interfaces

System calls

- Implemented by putting arguments in registers (or stack)
- Issue trap to switch from user to kernel

Rich set of programs (through GNU project)

- e.g. shells (bash, ksh, ...), compilers, editors, ...
- Desktop environments: GNOME, KDE, ...
- Utility programs: file, filters, editors, compilers, text processing, sys admin, etc



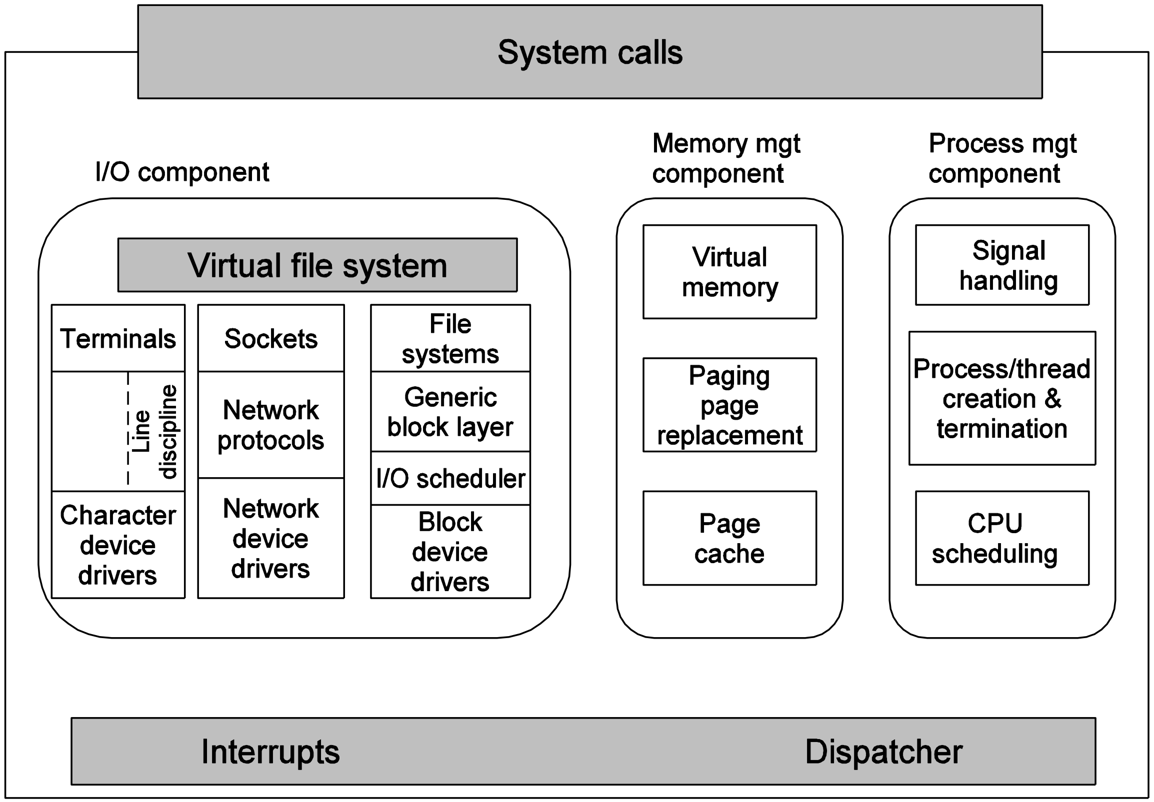
Kernel Structure

Interrupt handlers primary means to interact with devices

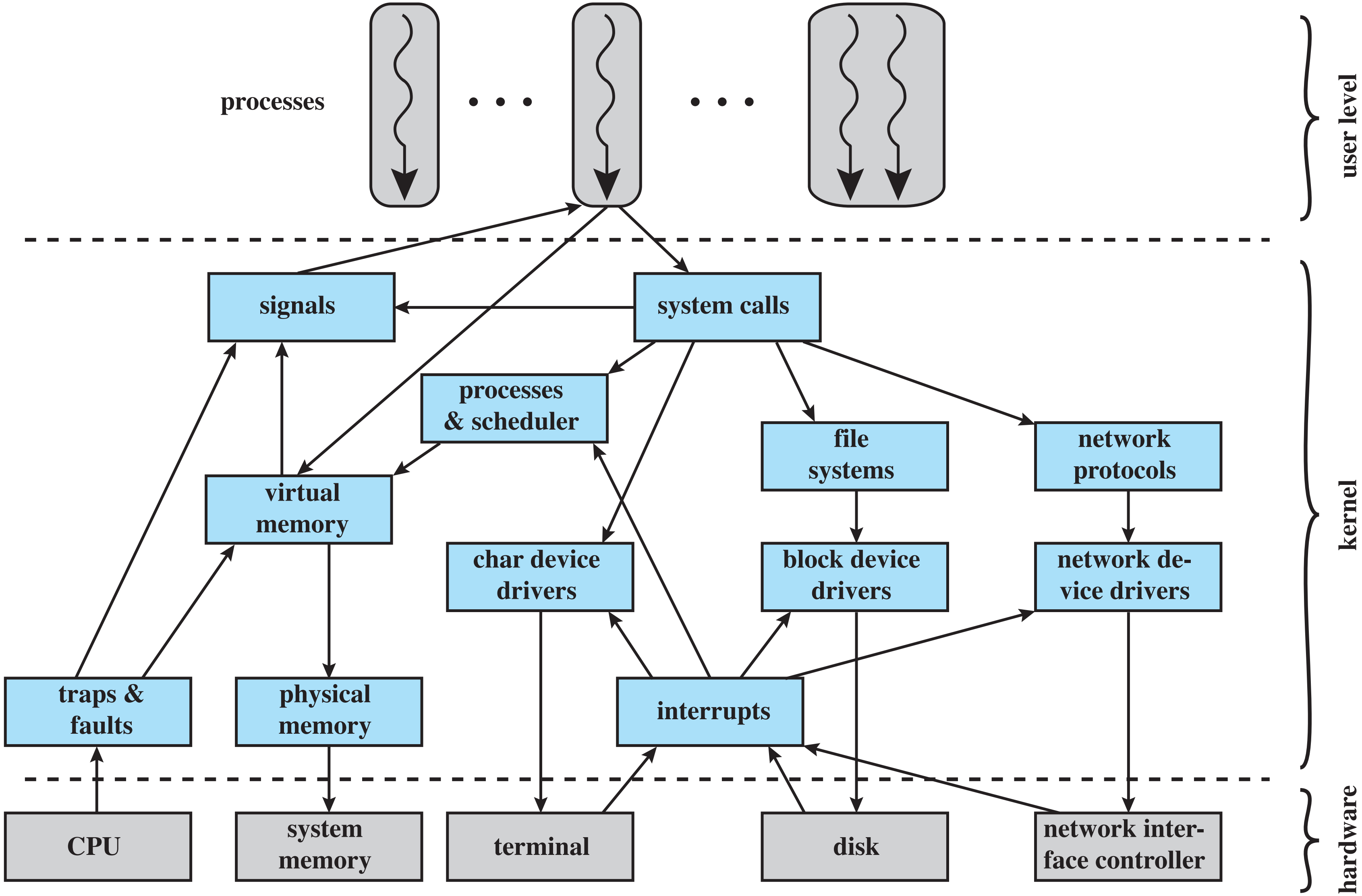
- Kicks off dispatching
  - Stop process, save state and start driver and return
- Dispatcher written in assembler

IO scheduler orders disk operation

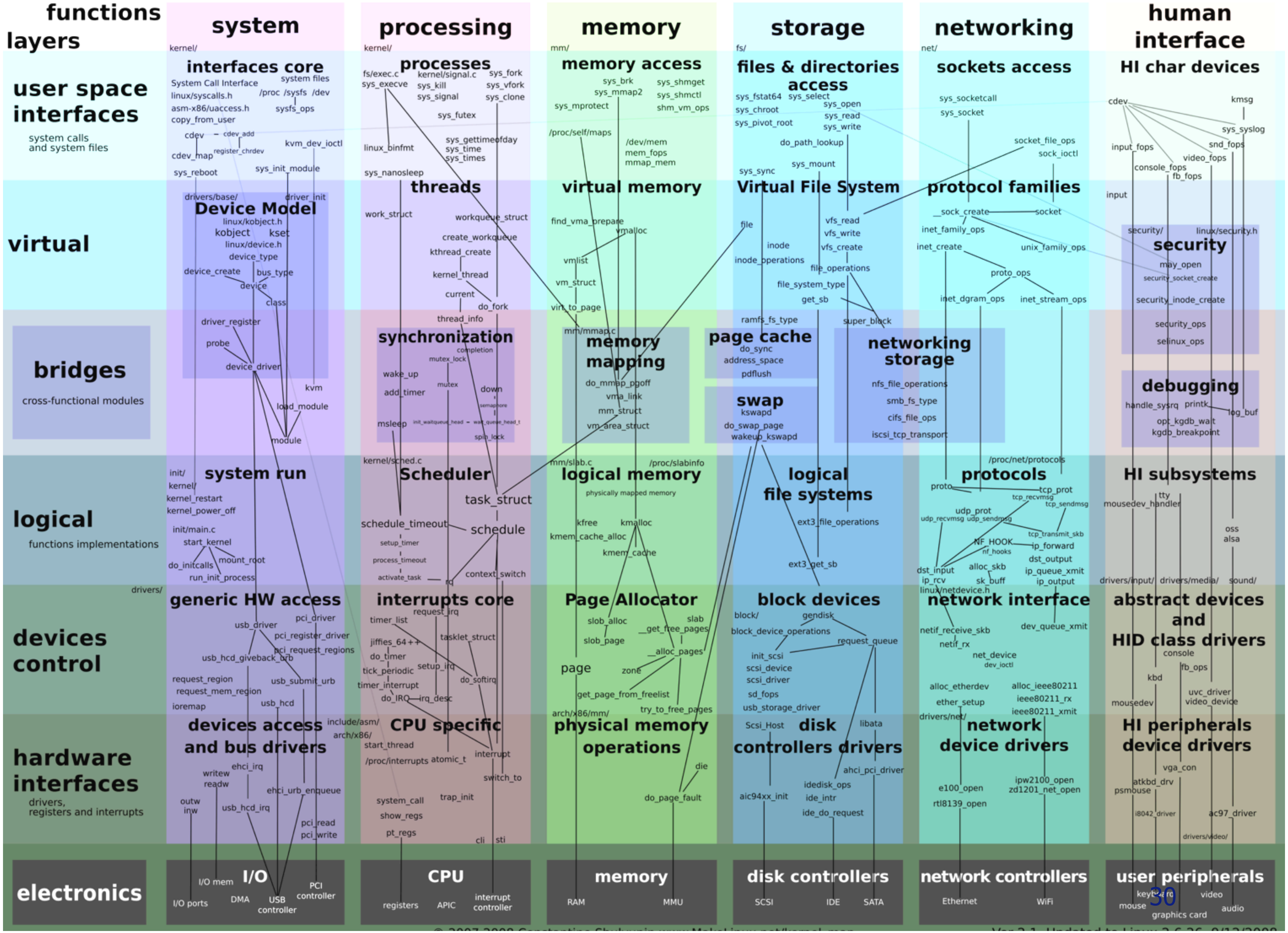
Monolithic: Static in-kernel components and dynamically loadable modules with shared internal data structures



Linux Kernel Components



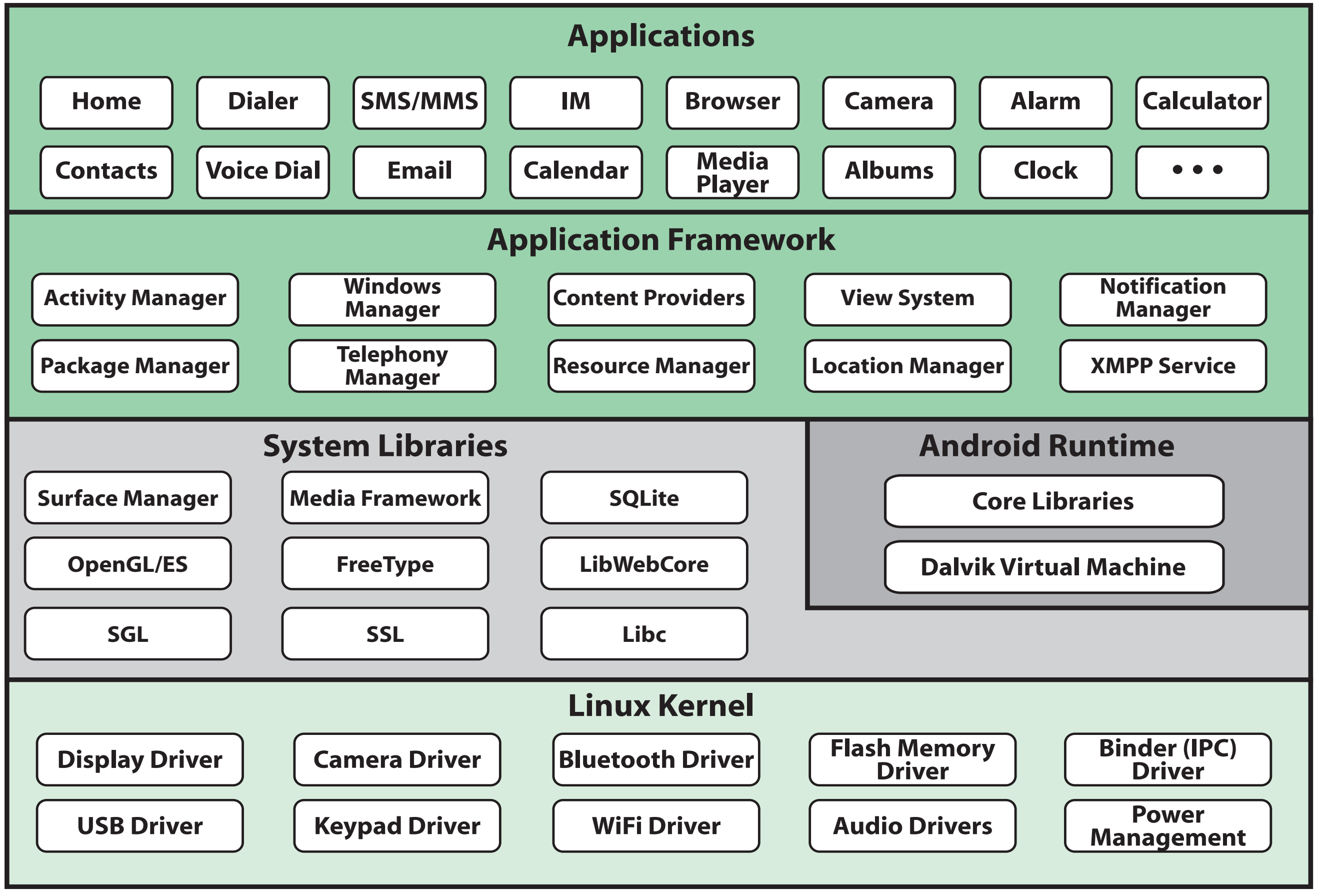
Linux kernel map



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

Android Software Architecture



Implementation:

Applications, Application Framework: Java

System Libraries, Android Runtime: C and C++

Linux Kernel: C

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 components
- Key components:

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

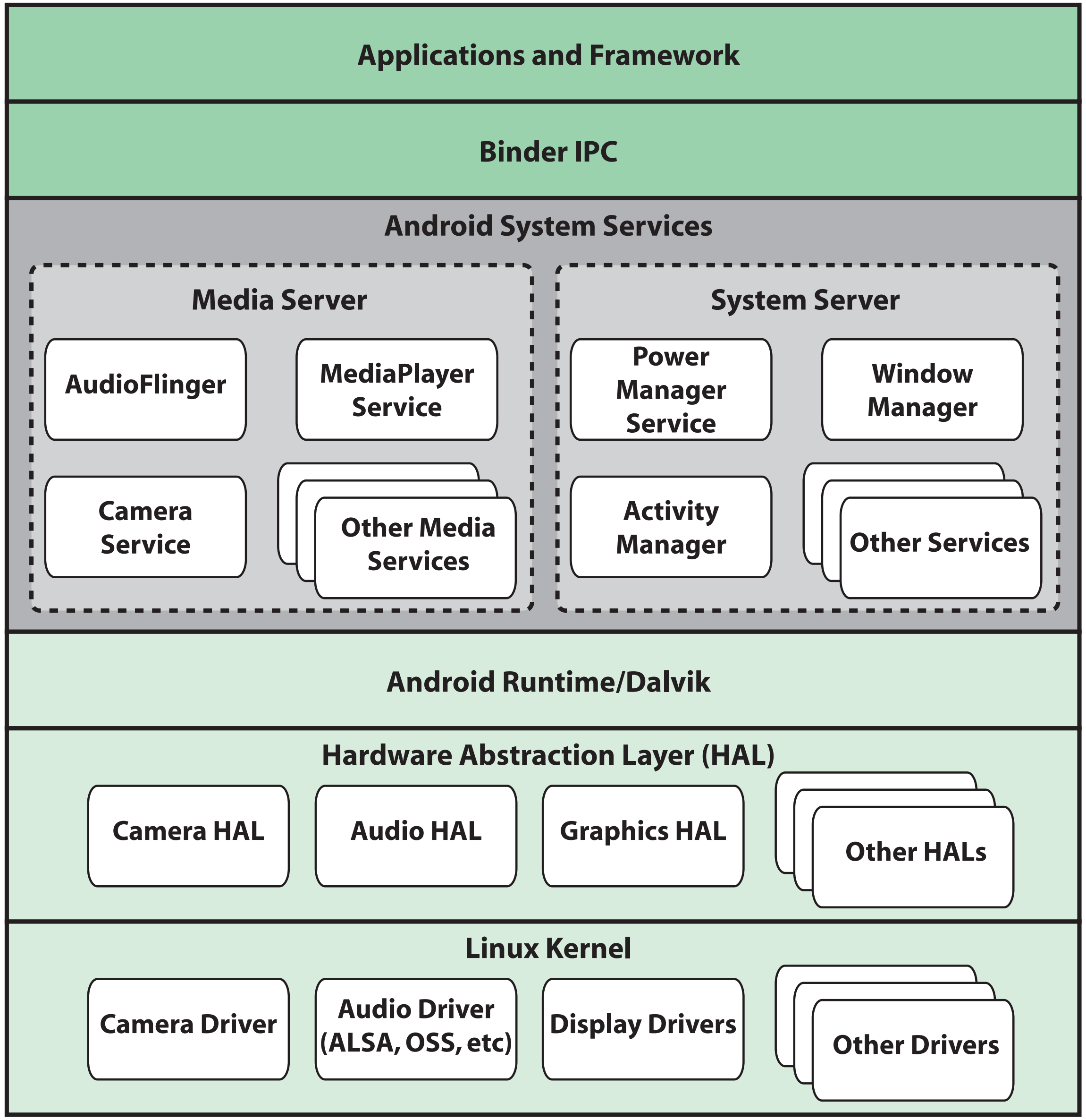
Some of the key system libraries include:

- 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)

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

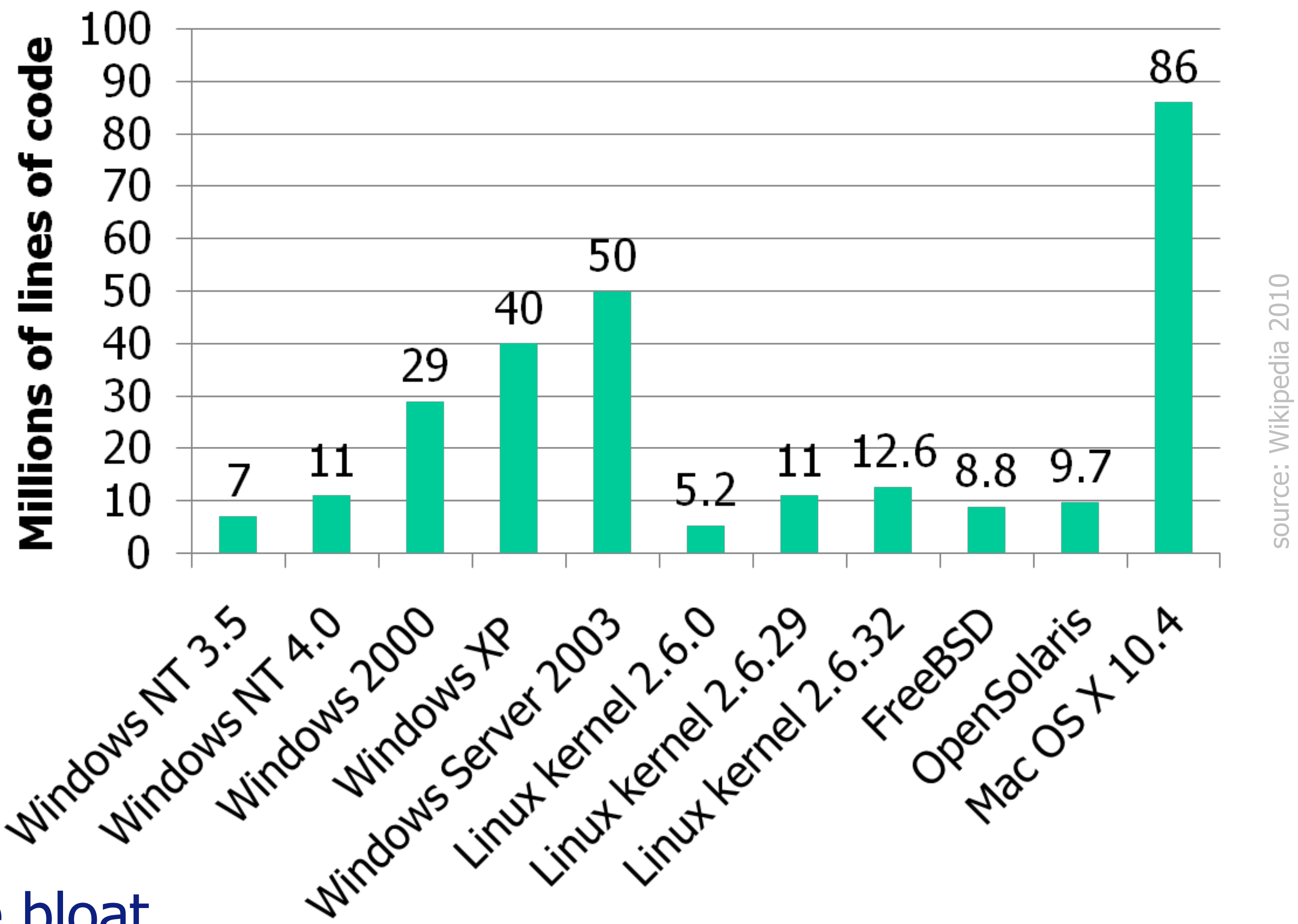
Alarms

- Implemented in the Linux kernel and is visible to the app developer through the AlarmManager in the RunTime core libraries
- 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

Evolution of OS Code Sizes



Code bloat

- Is lines of code useful comparison for complexity?
  - e.g. Linux scheduler (50K LoC); Vista scheduler (75K LoC)

Summary

OS Functions

- Simplify programming: device abstraction; virtual machine; memory management, file systems.
- Support concurrency, resource sharing & synchronisation

Kernel Structure

- Monolithic, Micro & Hybrid.

Operating System complexity