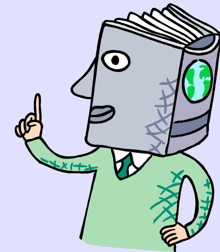
CSE 237A
Middleware and Operating Systems

Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.

Software components

- Standard software
  - e.g. MPEGx, databases
- Middleware
- Operating systems
  - schedulers



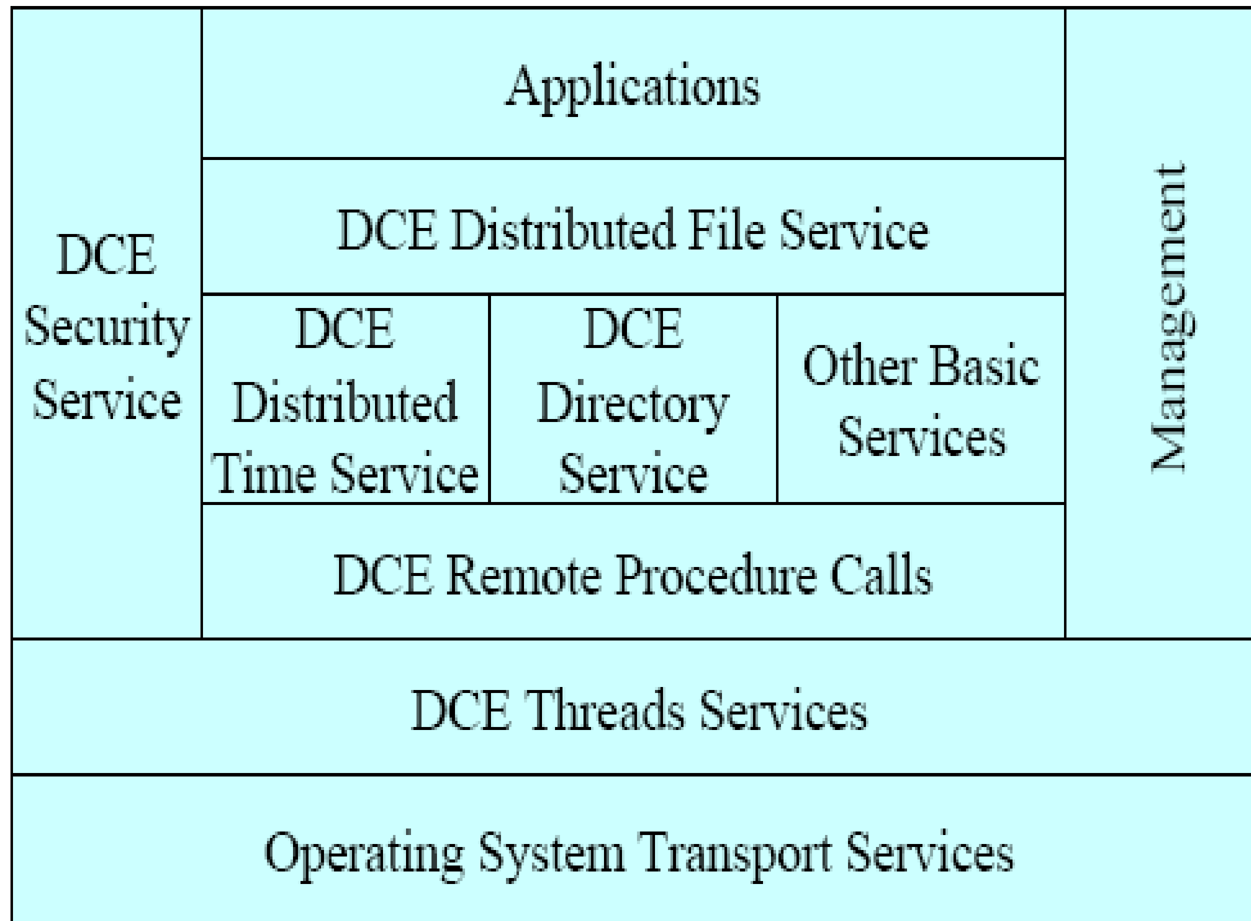
Middleware

- Between applications and OS
- Provides a set of higher-level capabilities and interfaces
- Customizable, composeable frameworks
- Types of services:
  - component – independent of other services
    - E.g. communication, information, computation
  - integrated sets
    - e.g. distributed computation environment
  - integration frameworks
    - Tailor to specific domain: e.g. transaction processing

Integrated sets

- A set of services that take significant advantage of each other
- Example: Distributed Computing Environment (DCE)
  - Provides key distributed technologies – RPC, DNS, distributed file system, time synch, network security and threads service
  - From Open SW Foundation, supported by multiple architectures and major SW vendors

DCE



Integration frameworks middleware

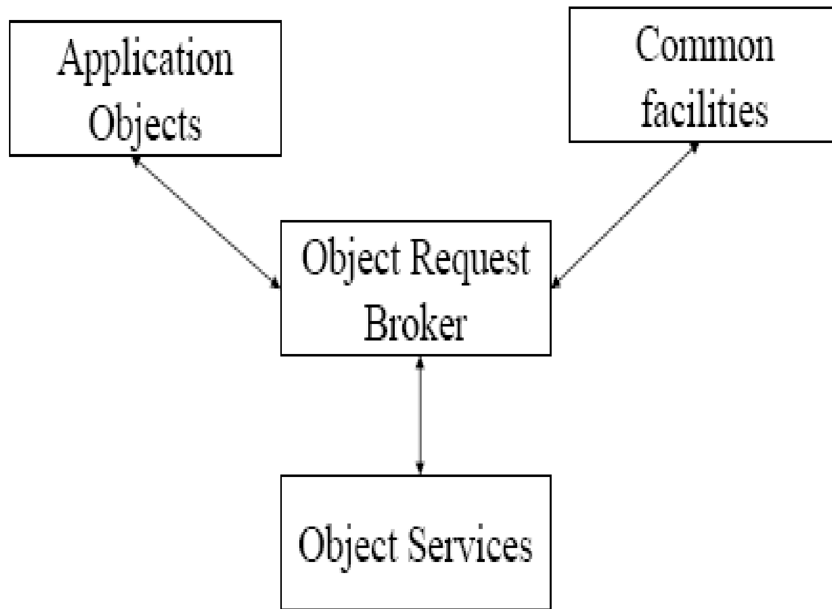
- Integration environments tailored to specific domain
- Examples:
  - Workgroup framework
  - Transaction processing framework
  - Network management framework
  - Distributed object computing (e.g. CORBA, E-SPEAK, JINI, message passing)

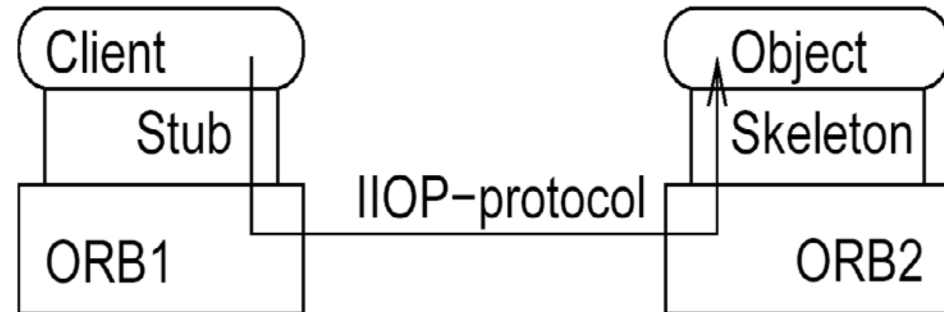
Distributed Object Computing

- Advantages:
  - SW reusability, more abstract programming, easier coordination among services
- Issues:
  - latency, partial failure, synchronization, complexity
- Techniques:
  - Message passing (object knows about network)
  - Argument/Return Passing – like RPC
    - network data = args + return result + names
  - Serialzing and sending
    - network data = obj code + obj state + synch info
  - Shared memory
    - network data = data touched + synch info

SW for access to remote objects

CORBA (Common Object Request Broker Architecture).
Information sent to Object Request Broker (ORB) via local stub.
ORB determines location to be accessed and sends information via the IIOP I/O protocol.





OBJ management architecture

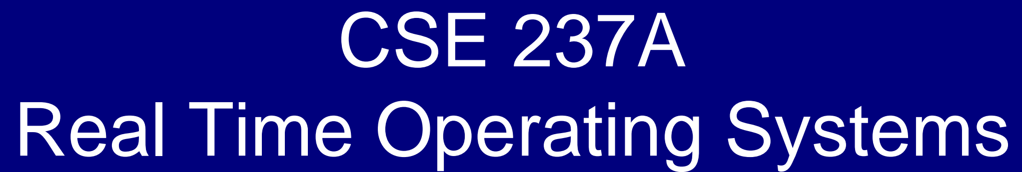
Access times not predictable.

Real-time CORBA

- End-to-end predictability of timeliness in a fixed priority system.
- respecting thread priorities between client and server for resolving resource contention,
- bounding the latencies of operation invocations.
- RT-CORBA includes provisions for bounding the time during which priority inversion due to competing resource access may occur.

Message passing interface

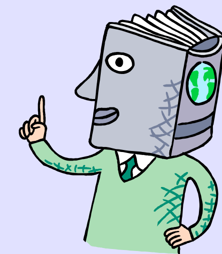
- Message passing interface (MPI): alternative to CORBA
- MPI/RT: a real-time version of MPI [MPI/RT forum, 2001].
- MPI-RT does not cover issues such as thread creation and termination.
- MPI/RT is conceived as a layer between the operating system and non real-time MPI.



Tajana Simunic Rosing
Department of Computer Science and Engineering
University of California, San Diego.

Software components

- Standard software
  - e.g. MPEGx, databases
- Middleware
- Operating systems
  - Focus on RTOS



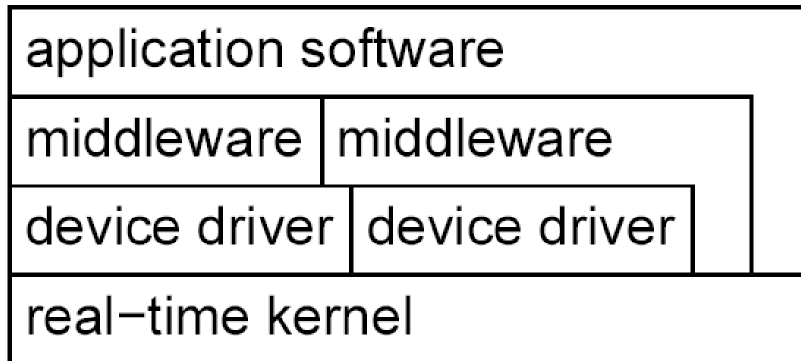
Real-time operating systems

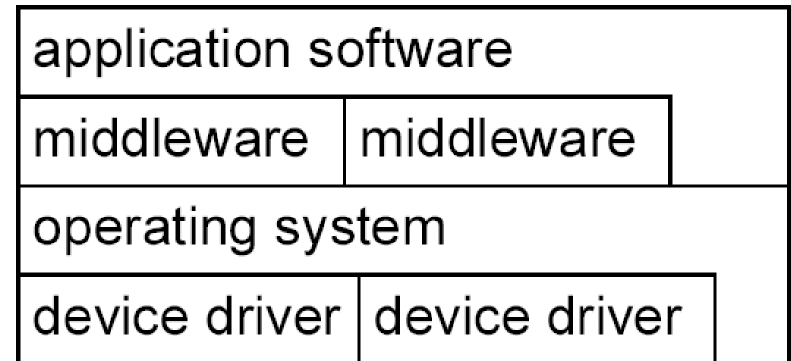
Three key requirements

1. **Predictable OS timing behavior**
   - upper bound on the execution time of OS services
   - short times during which interrupts are disabled,
   - contiguous files to avoid unpredictable head movements
2. **OS must be fast**
3. **OS must manage the timing and scheduling**
   - OS possibly has to be aware of task deadlines; (unless scheduling is done off-line).
   - OS must provide precise time services with high resolution.

RTOS-Kernels

- Distinction between real-time kernels and modified kernels of standard OSes.





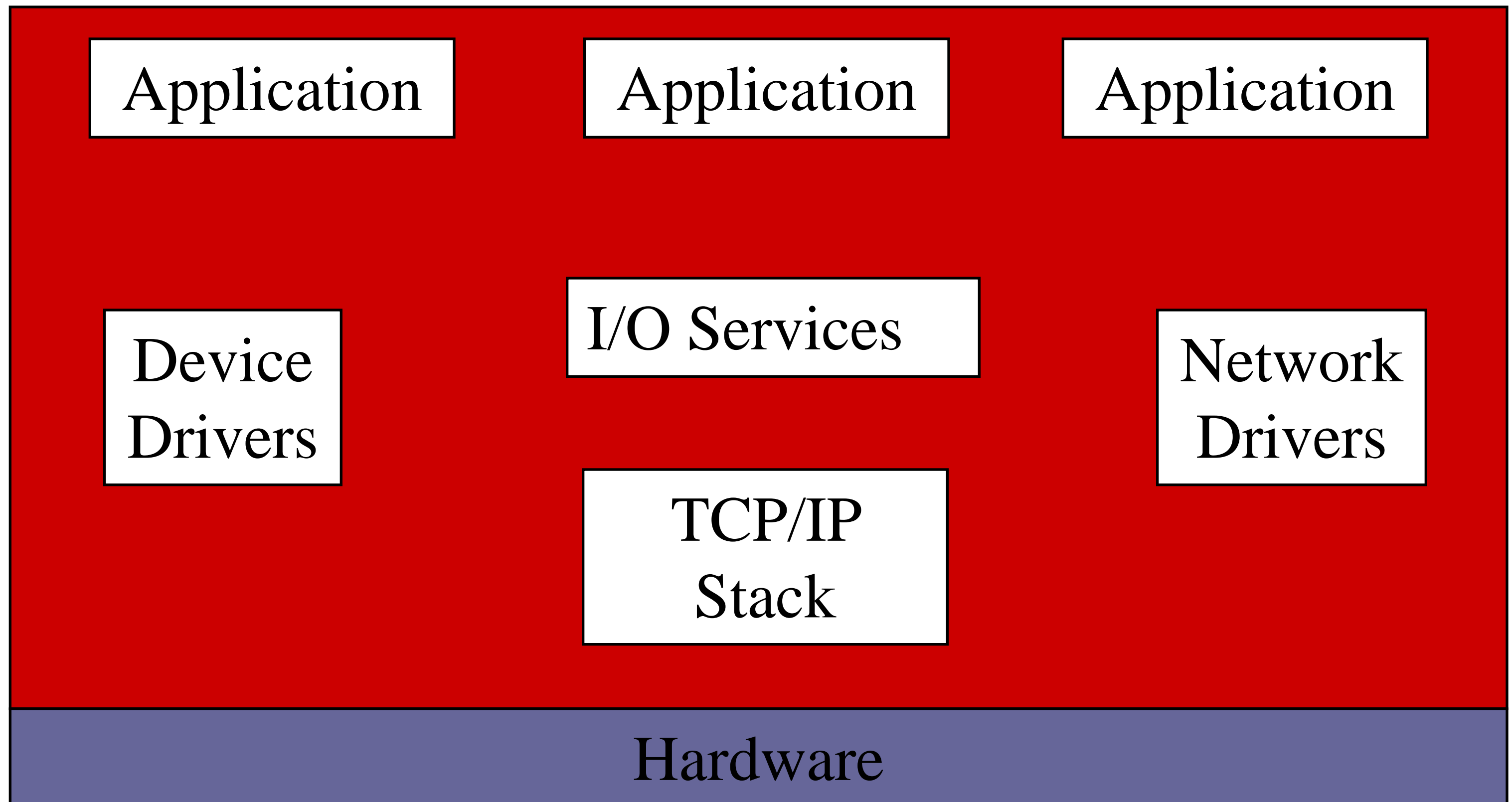
- **Distinction between**
- general and RTOSes for specific domains,
- standard APIs (e.g. POSIX RT-Extension of Unix) or proprietary APIs.

How to organize multiple tasks?

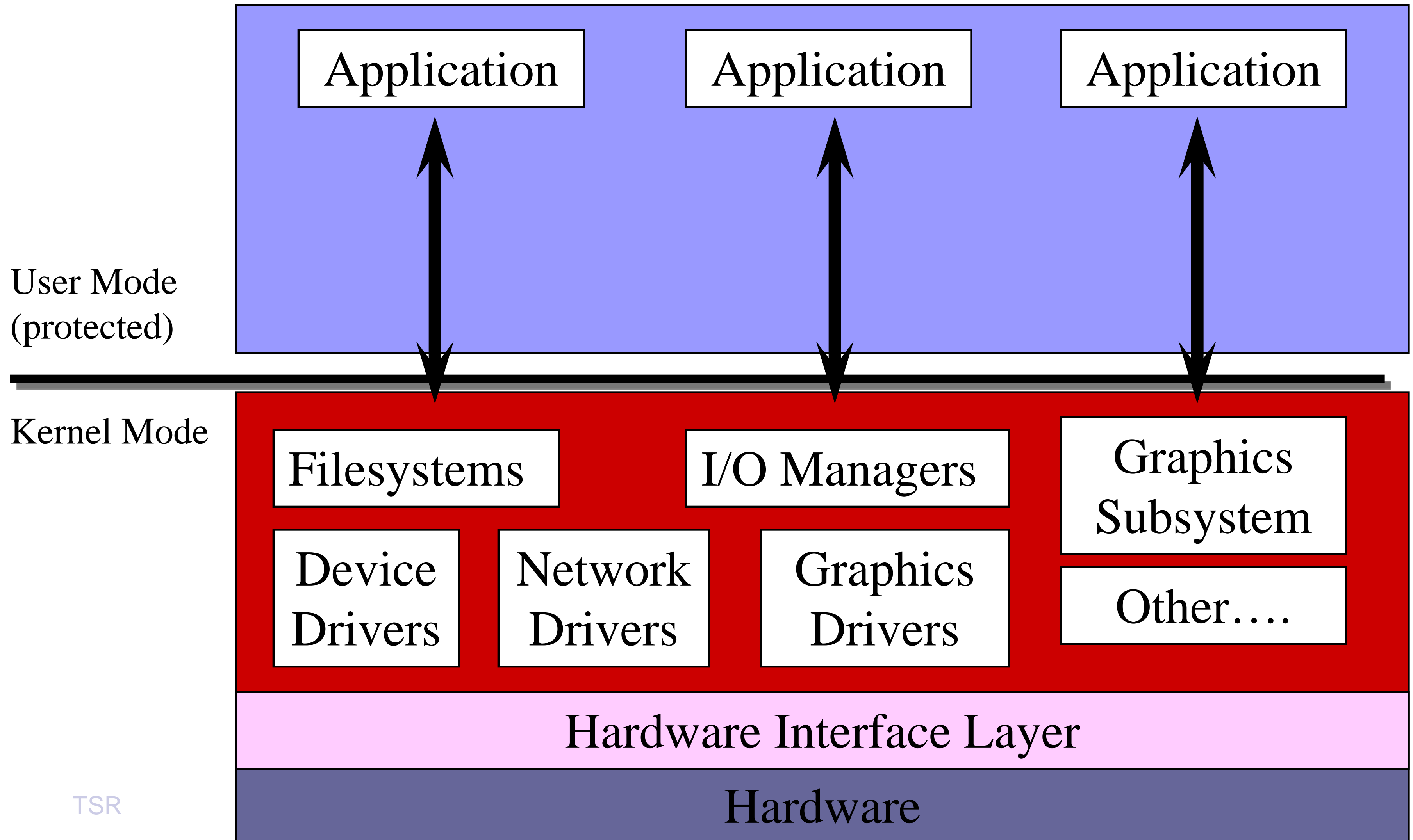
- Cyclic executive (Static table driven scheduling)
  - static schedulability analysis
  - resulting schedule or table used at run time
- Event-driven non-preemptive
  - tasks are represented by functions that are handlers for events
  - next event processed after function for previous event finishes
- Static and dynamic priority preemptive scheduling
  - static schedulability analysis
  - at run time tasks are executed “highest priority first”
  - Rate monotonic, deadline monotonic, earliest deadline first, least slack

RTOS Organization: Cyclic Executive

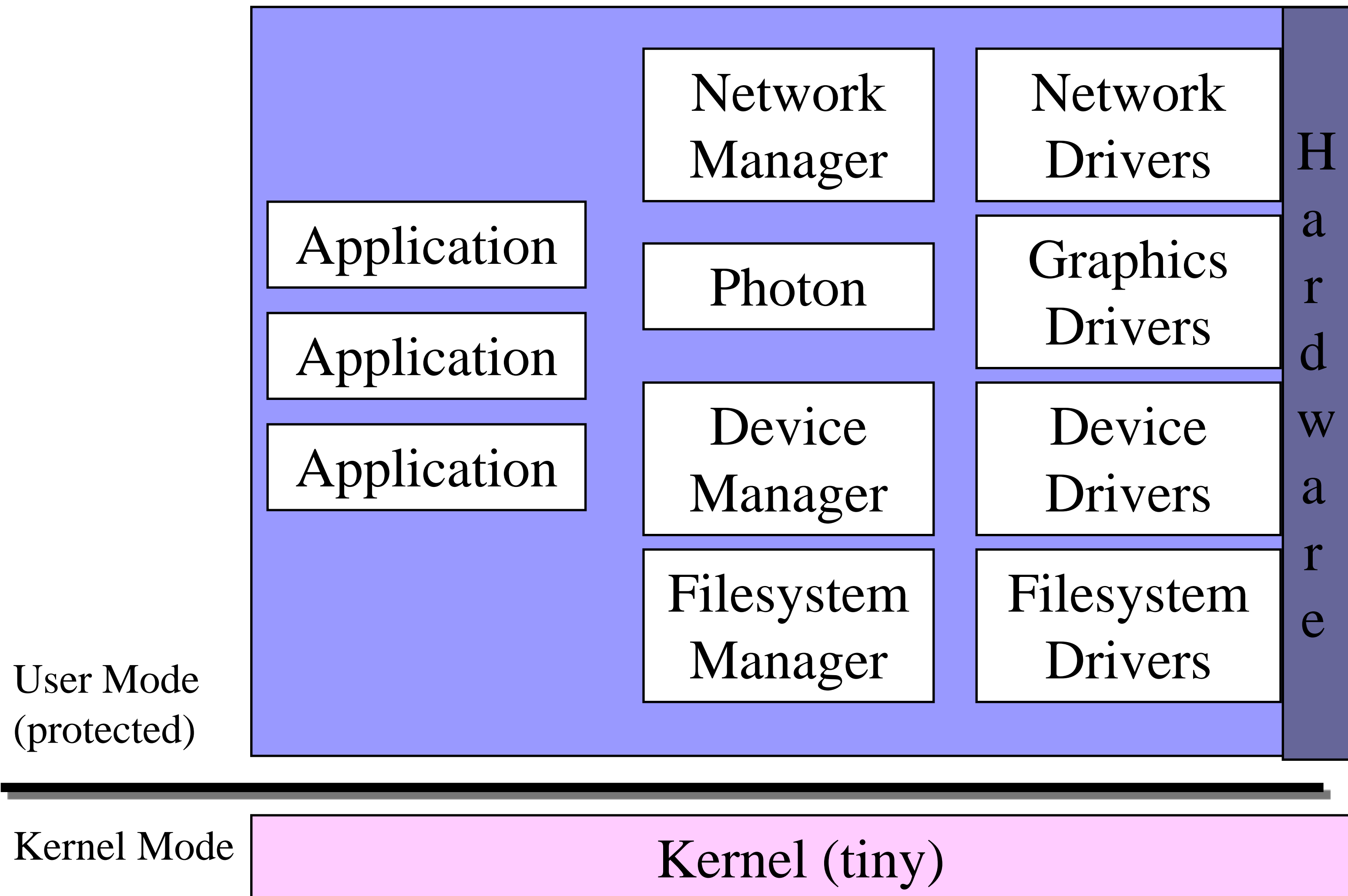
Kernel Mode



RTOS Organization: Monolithic Kernel



RTOS Organization: Microkernel

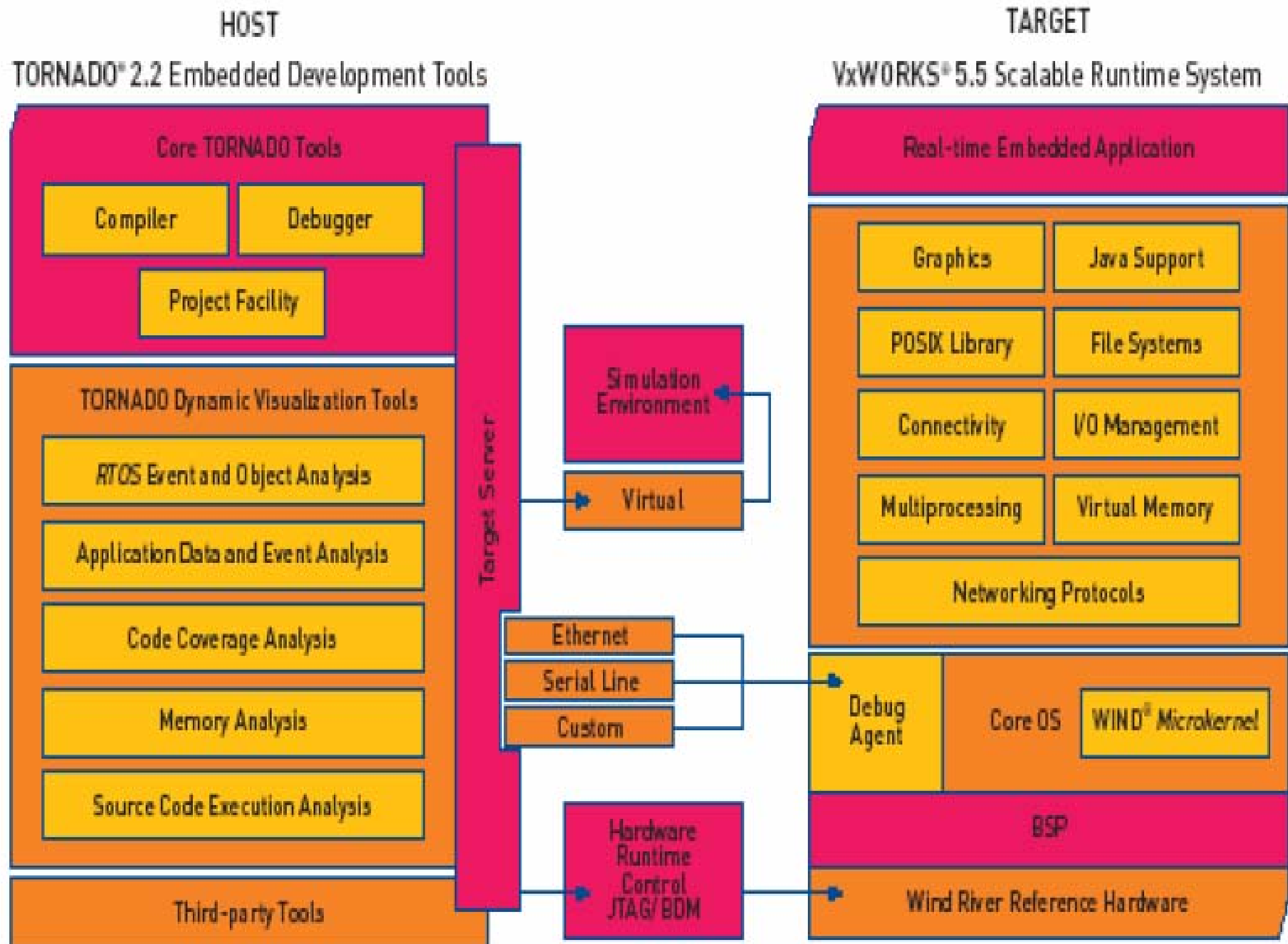


Types of RTOS Kernels
1. Fast proprietary kernels

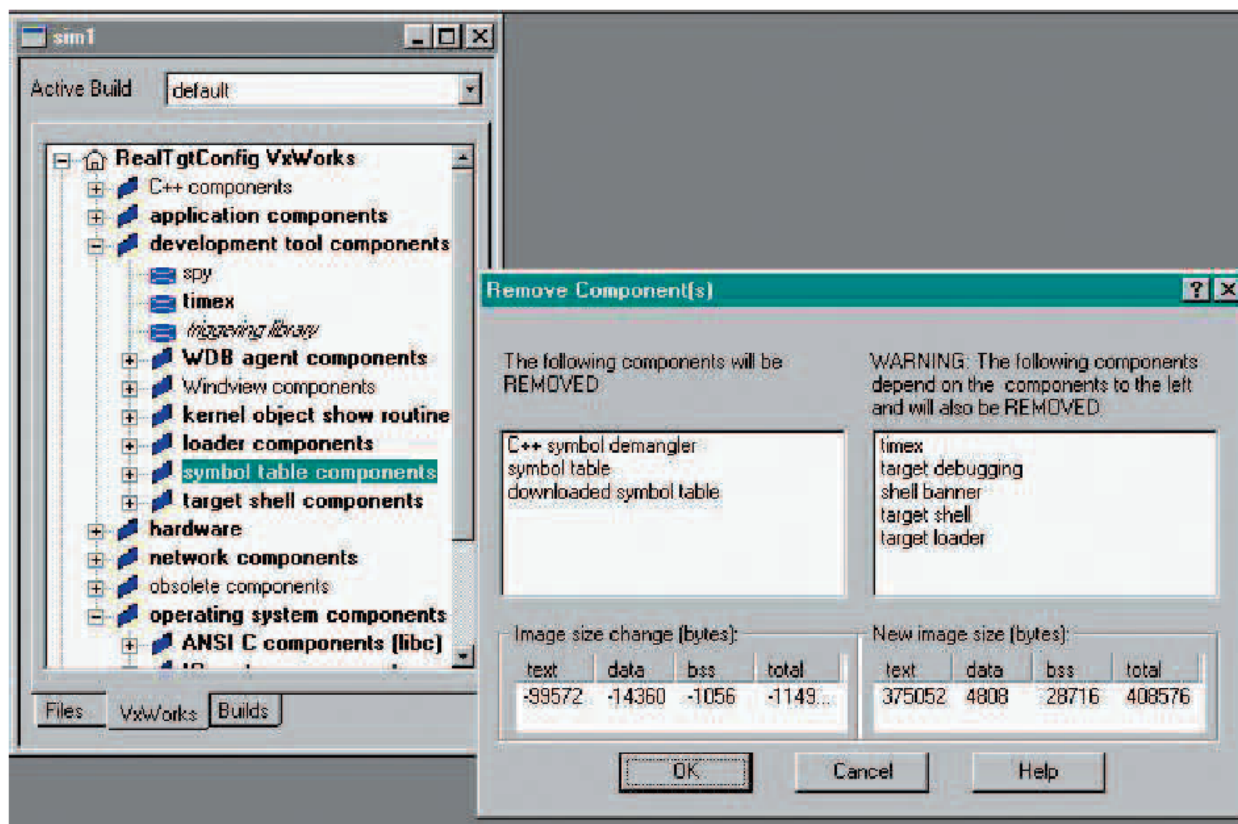
- designed to be fast, rather than predictable
- Inadequate for complex systems
- Examples include

  QNX, PDOS, VCOS, VTRX32, VxWORKS.

Example: VxWorks



VxWorks Configuration



Automatic dependency analysis and size calculations allow users to quickly custom-tailor the VxWORKS operating system.

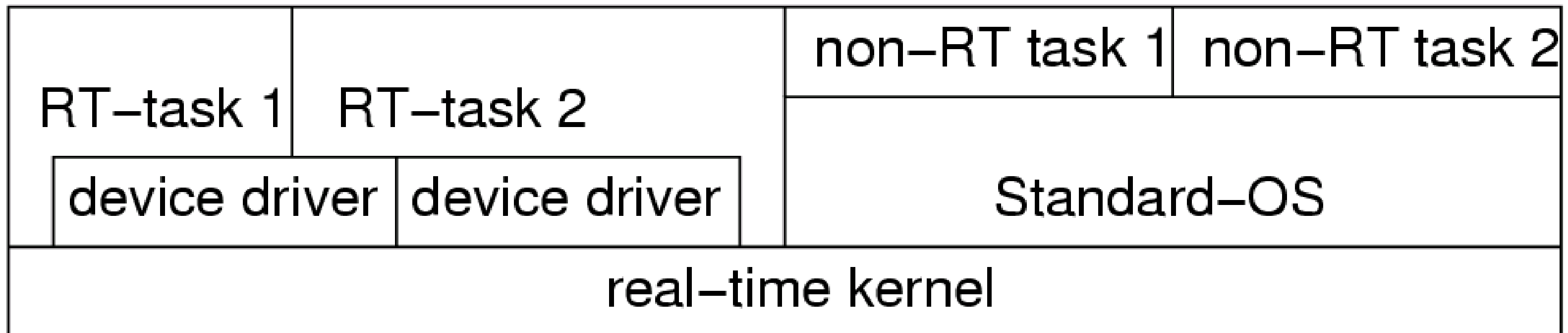
http://www.windriver.com/products/development_tools/ide/tornado2/tornado_2_ds.pdf

© Windriver

Types of RTOS Kernels

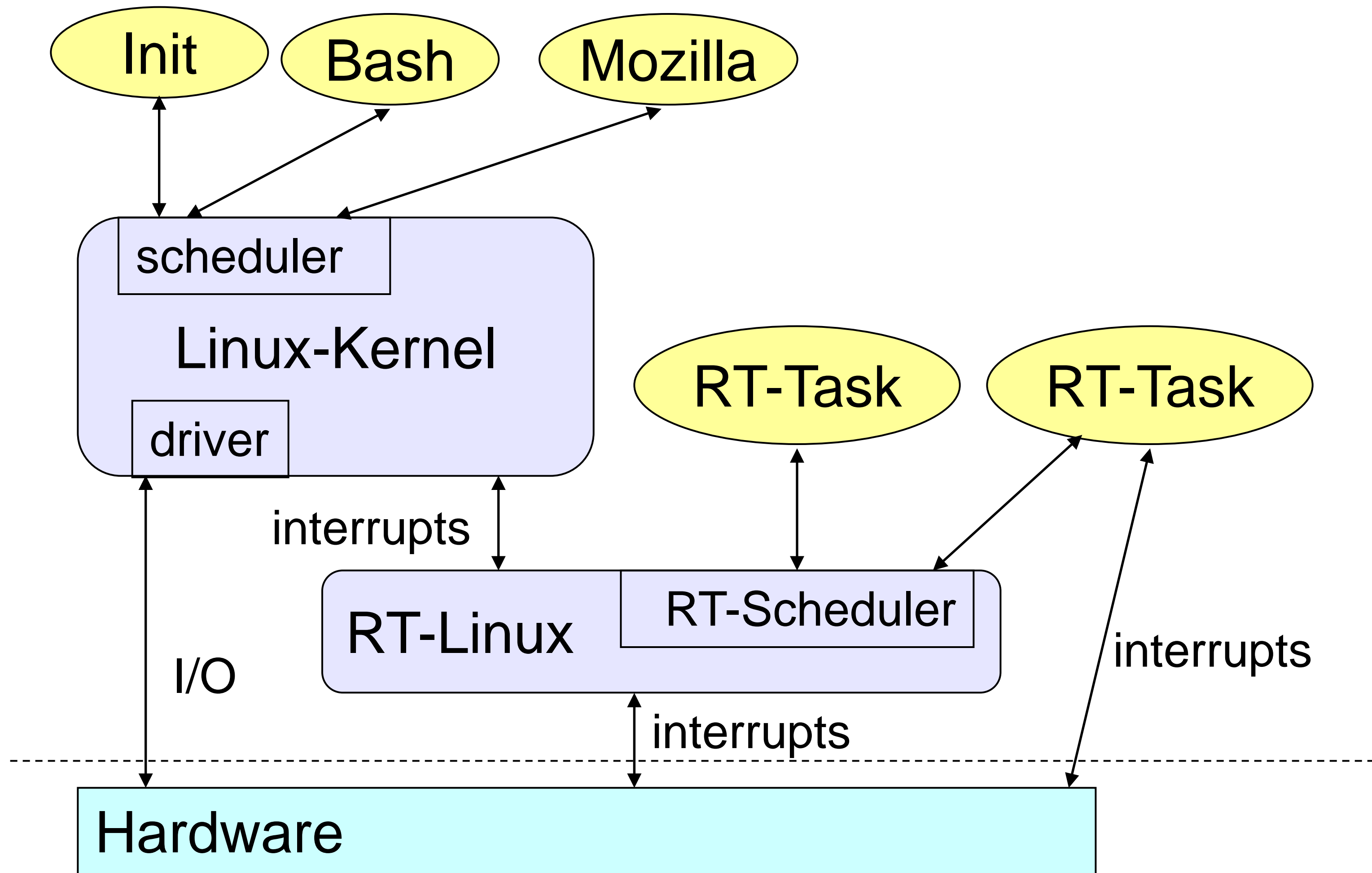
2. Standard OS with real-time extensions

- RT-kernel running all RT-tasks.
- Standard-OS executed as one task.



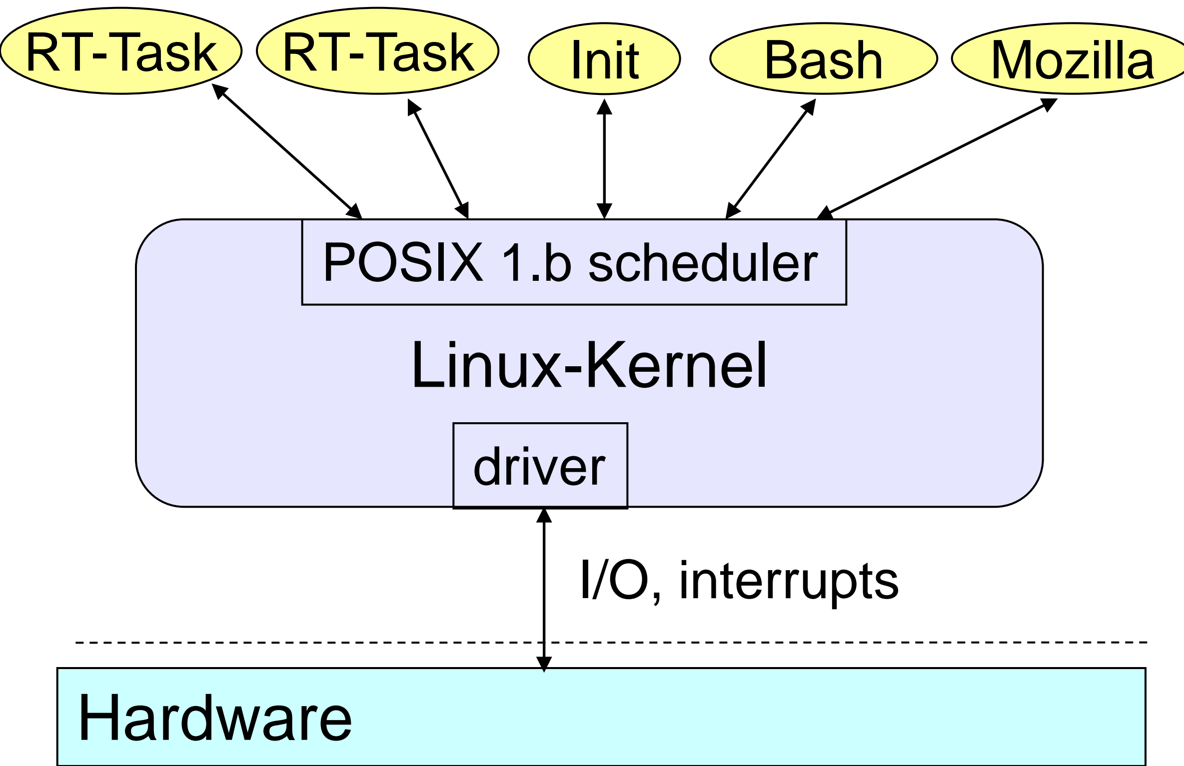
+ Crash of standard-OS does not affect RT-tasks;
- RT-tasks cannot use Standard-OS services;
  less comfortable than expected

Example: RT-Linux



Example: Posix 1.b RT

- Standard scheduler can be replaced by POSIX scheduler implementing priorities for RT tasks



- Special RT & standard OS calls available.
- Easy programming, no guarantee for meeting deadline

Types of RTOS Kernels 3. Research systems

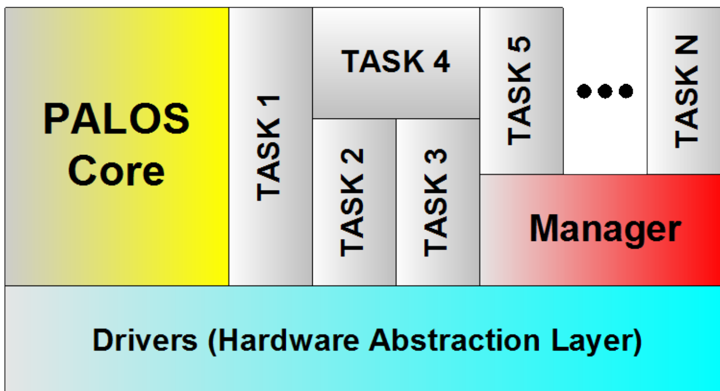
- **Research issues**
  - low overhead memory protection,
  - temporal protection of computing resources
  - RTOSes for on-chip multiprocessors
  - support for continuous media
  - quality of service (QoS) control.

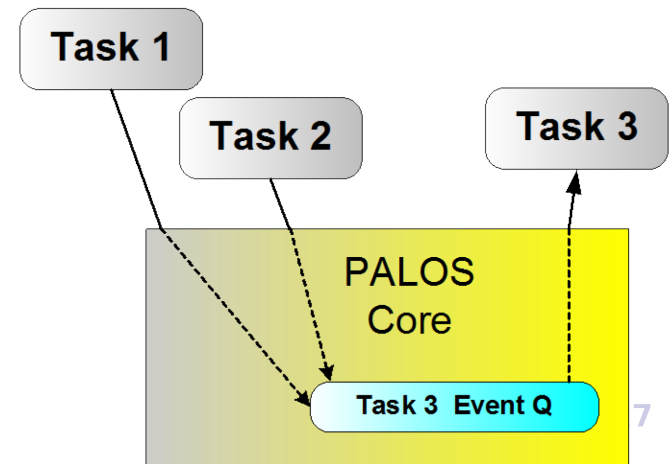
Kernel examples

- Small kernels
  - PALOS, TinyOS
- Medium size
  - uCos II, eCos
- Larger
  - RT Linux, WinCE

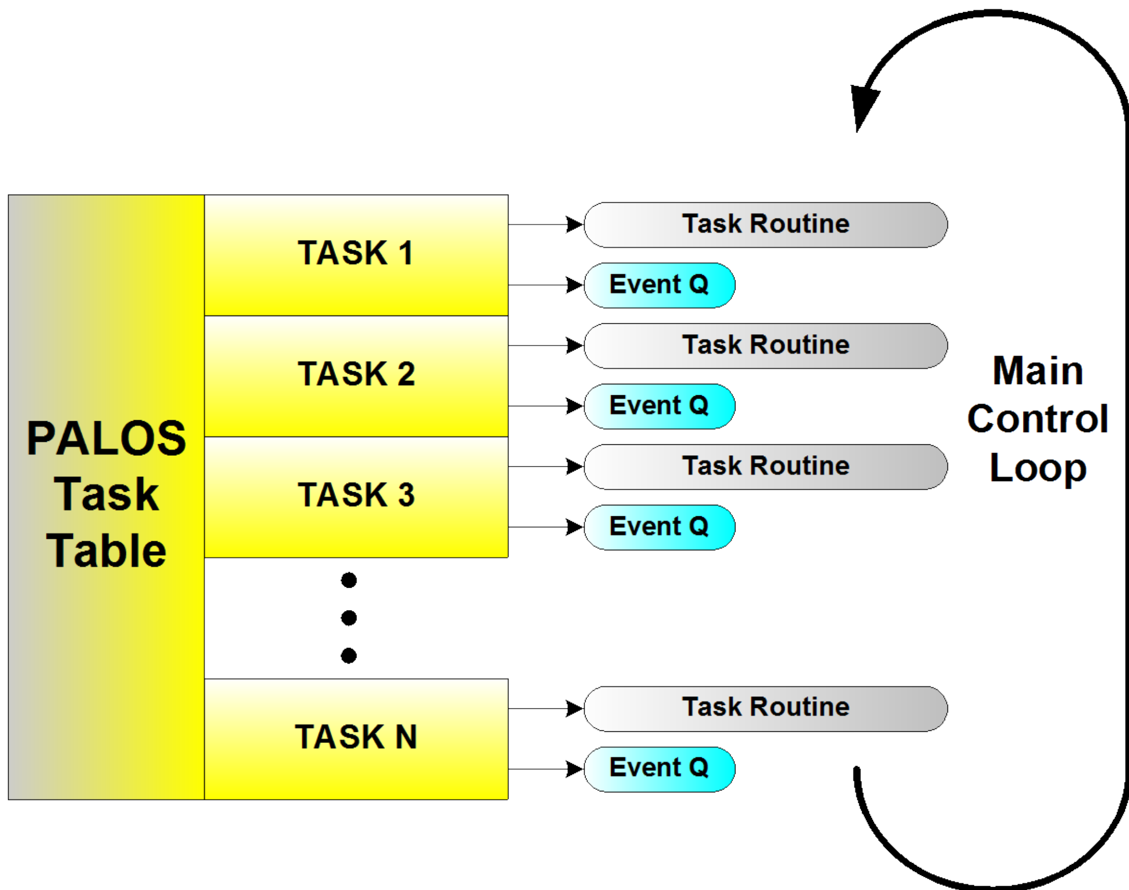
Example I: PALOS

- Structure – PALOS Core, Drivers, Managers, and user defined Tasks
- PALOS Core
  - Task control: slowing, stopping, resuming
  - Periodic and aperiodic handling and scheduling
  - Inter-task Communication via event queues
  - Event-driven tasks: task routine processes events stored in event queues
- Drivers
  - Processor-specific: UART, SPI, Timers..
  - Platform-specific: Radio, LEDs, Sensors
- Small Footprint
  - Core (compiled for ATMega128L) Code Size: 956 Bytes , Mem Size: 548 Bytes
  - Typical(3 drivers, 3 user tasks) Code Size: 8 Kbytes, Mem Size: 1.3 Kbytes



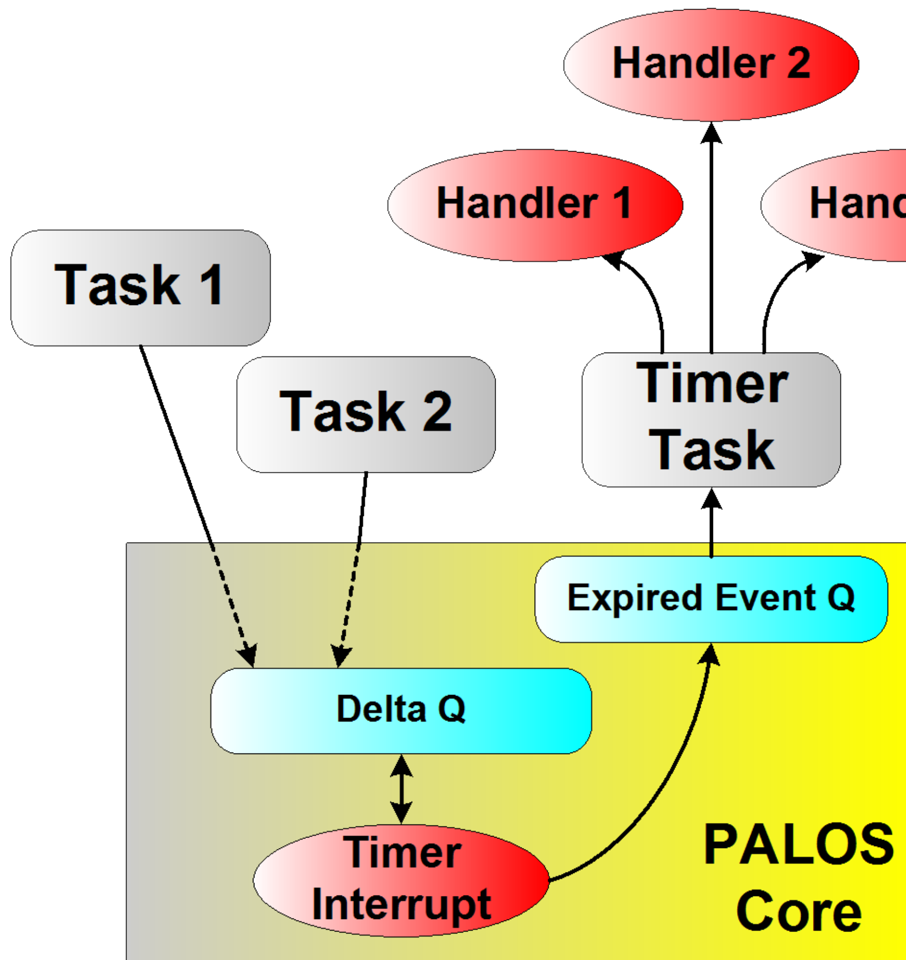


Execution control in PALOS



- Each task has a task counter
- Counters initialized to:
  - 0: normal
  - >>:0 slowdown
  - -1: stop
  - >=0: restart
- Decremented
  1) every iteration (relative timing)
  2) by timer interrupts (exact timing)
- If counter = 0, call taks; reset counter to initialization value

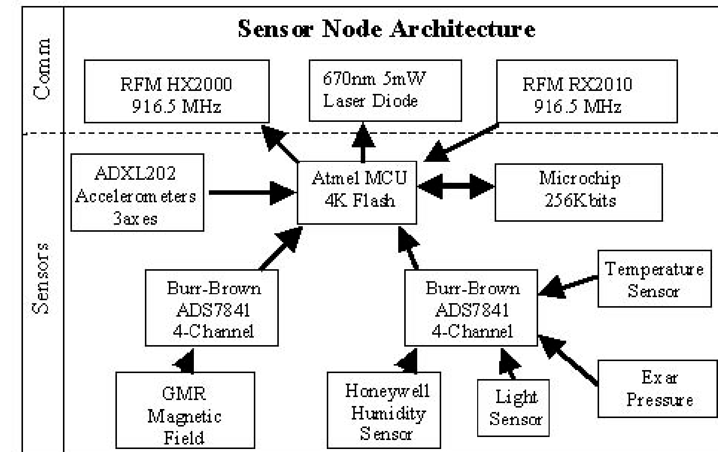
Event Handlers in PALOS

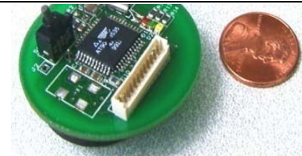


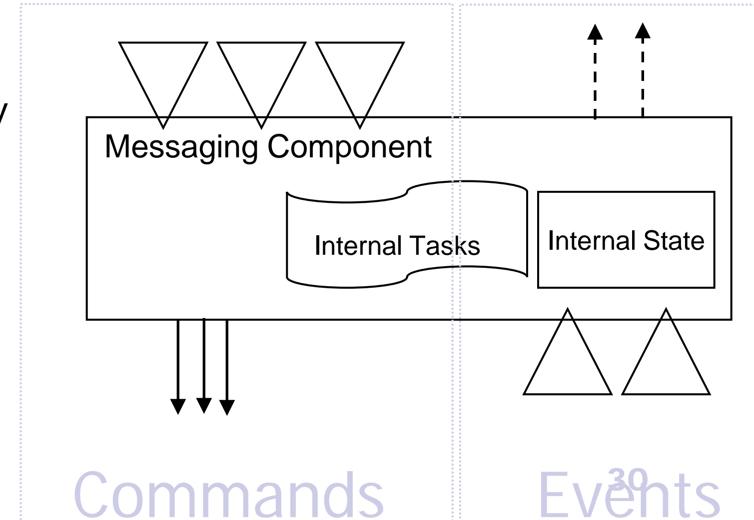
- Periodic or aperiodic events can be scheduled using Delta Q and Timer Interrupt
- When event expires appropriate event handler is called

Example II: TinyOS

- System composed of
  - scheduler, graph of components, execution context
- Component model
  - Basically FSMs
  - Four interrelated parts of implementation
    - Encapsulated fixed-size frame (storage)
    - A set of command & event handlers
    - A bundle of simple tasks (computation)
  - Modular interface
    - Commands it uses and accepts
    - Events it signals and handles
- Tasks, commands, and event handlers
  - Execute in context of the frame & operate on its state
  - Commands are non-blocking requests to lower level components
  - Event handlers deal with hardware events
  - Tasks perform primary work, but can be preempted by events
- Scheduling and storage model
  - Shared stack, static frames
  - Events prempt tasks, tasks do not
  - Events can signal events or call commands
  - Commands don't signal events
  - Either can post tasks



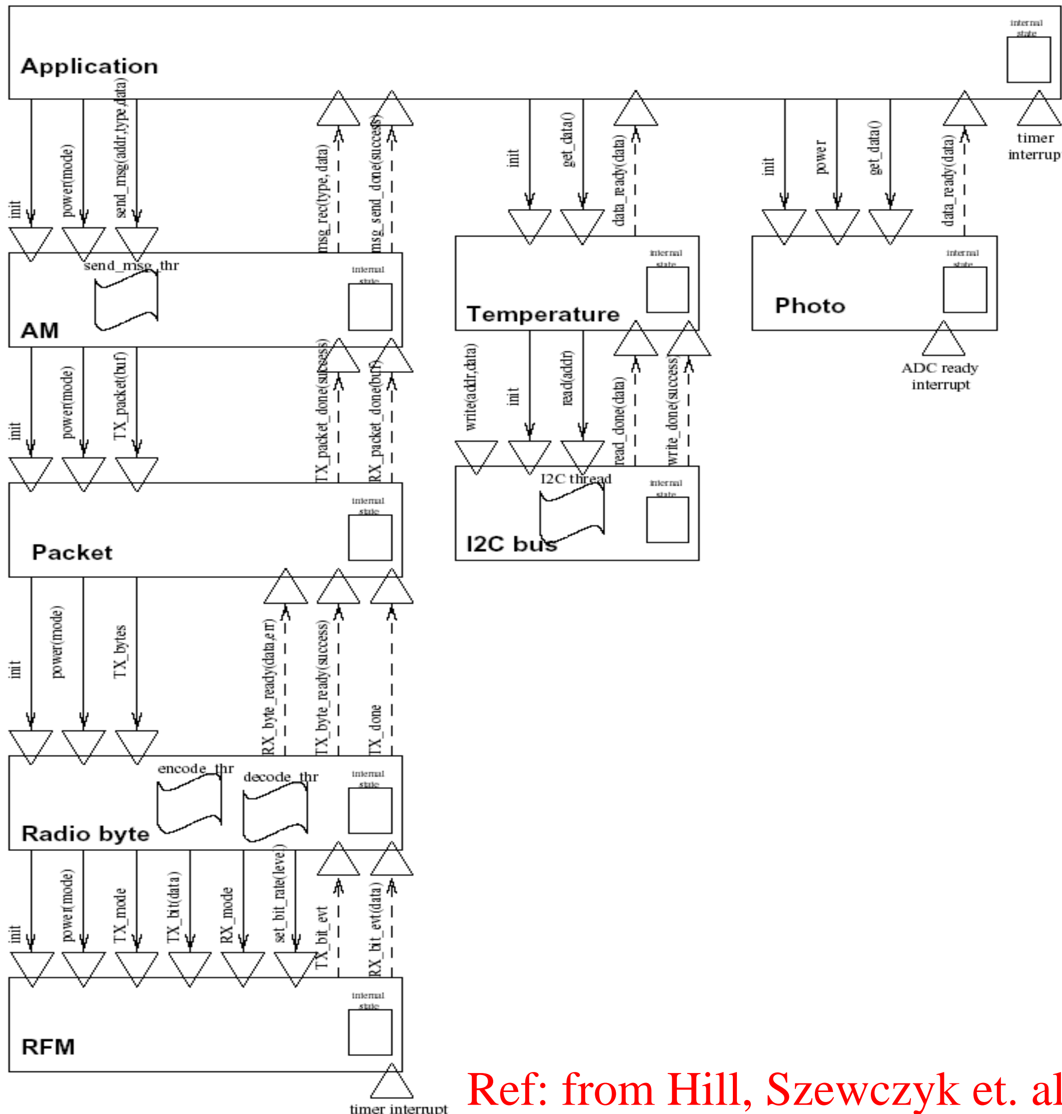




TinyOS Overview

- Stylized programming model with extensive static information
  - Compile time memory allocation
- Easy migration across h/w -s/w boundary
- Small Software Footprint - 3.4 KB
- Two level scheduling structure
  - Preemptive scheduling of event handlers
  - Non-preemptive FIFO scheduling of tasks
  - Bounded size scheduling data structure
- Rich and Efficient Concurrency Support
  - Events propagate across many components
  - Tasks provide internal concurrency
- Power Consumption on Rene Platform
  - Transmission Cost: 1 µJ/bit
  - Inactive State: 5 µA
  - Peak Load: 20 mA
- Efficient Modularity - events propagate through stack <40 µS

Complete TinyOS Application



Component Name	Code Size (bytes)	Data Size (bytes)
Multihop router	88	0
AM_dispatch	40	0
AM_temperature	78	32
AM_light	146	8
AM	356	40
Packet	334	40
RADIO_byte	810	8
RFM	310	1
Photo	84	1
Temperature	64	1
UART	196	1
UART_packet	314	40
I2C_bus	198	8
Procesor_init	172	30
TinyOS scheduler	178	16
C runtime	82	0
Total	3450	226

Ref: from Hill, Szewczyk et. al., ASPLOS 2000

Example III: µCOS-II

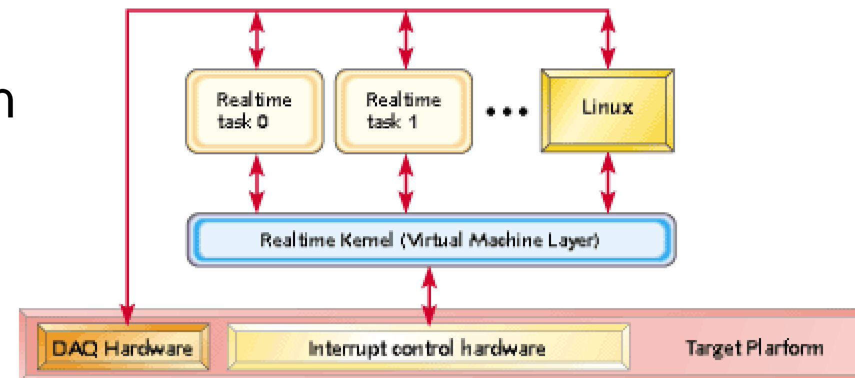
- Portable, ROMable, scalable, preemptive, multitasking RTOS
- Services
  - Semaphores, event flags, mailboxes, message queues, task management, fixed-size memory block management, time management
- Source freely available for academic non-commercial usage for many platforms
  - Value added products such as GUI, TCP/IP stack etc.

Example IV: eCos

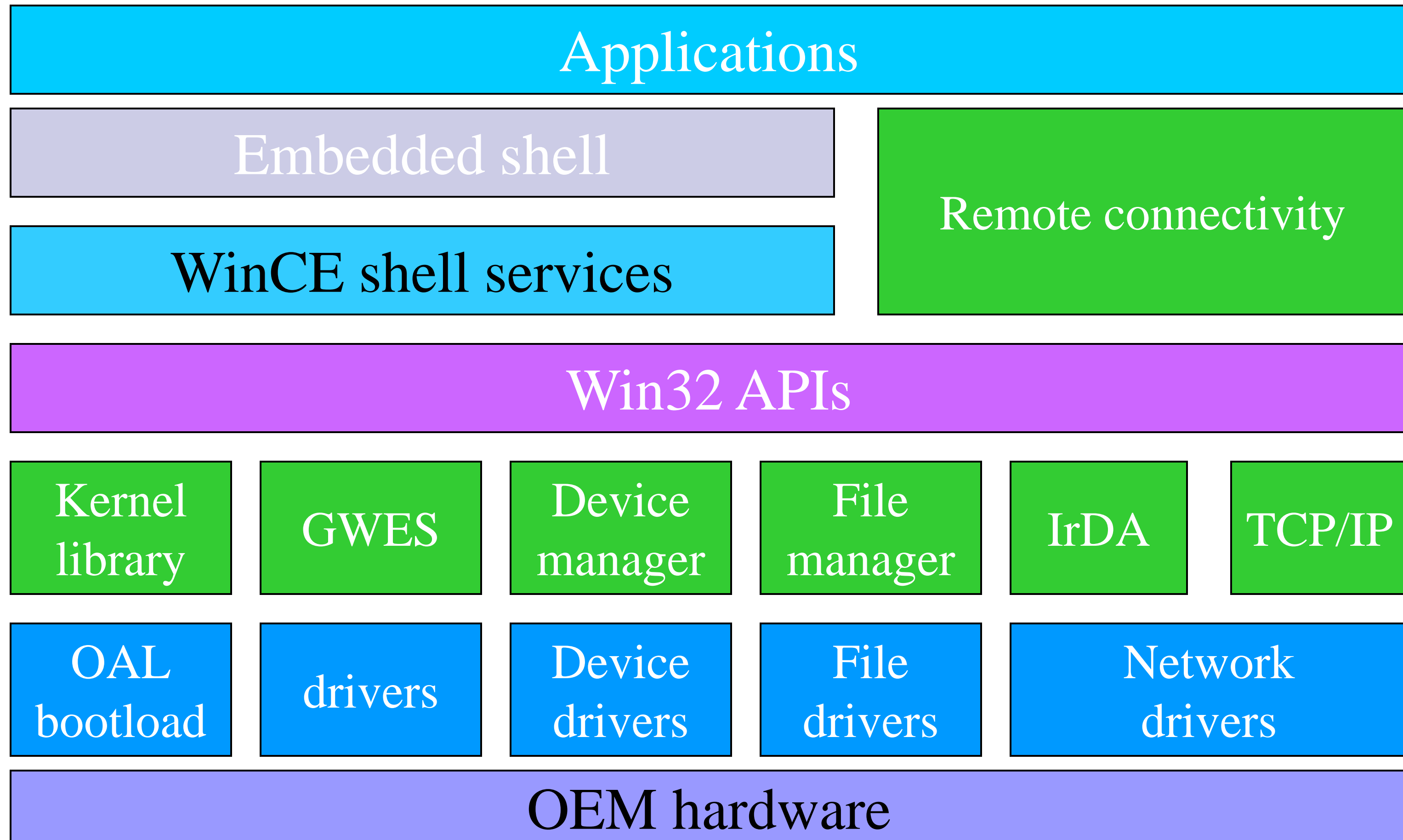
- Embedded, Configurable OS, Open-source
- Several scheduling options
  - bit-map scheduler, lottery scheduler, multi-level scheduler
- Three-level processing
  - Hardware interrupt (ISR), software interrupt (DSR), threads
- Inter-thread communication
  - Mutex, semaphores, condition variables, flags, message box
- Portable - Hardware Abstraction Layer (HAL)
- Based on configurable components
  - Package based configuration tool
  - Kernel size from 32 KB to 32 MB
  - Implements ITRON standard for embedded systems
  - OS-neutral POSIX compliant EL/IX API

Example V: Real-time Linux

- Microcontroller (no MMU) OSes:
  - uClinux - small-footprint Linux (< 512KB kernel) with full TCP/IP
- QoS extensions for desktop:
  - Linux-SRT and QLinux
    - soft real-time kernel extension
    - target: media applications
- Embedded PC
  - RTLinux, RTAI
    - hard real time OS
      - E.g. RTLinux has Linux kernel as the lowest priority task in a RTOS
    - fully compatible with GNU/Linux
  - HardHat Linux



Example VI: WinCE



Virtual memory

- WinCE uses virtual memory.
- Code can be paged from ROM, etc.
- WinCE suports a flat 32-bit virtual address space.
- Virtual address may be:
  - Statically mapped (kernel-mode code).
  - Dynamically mapped (user-mode and some kernel-mode code).
- Address space: bottom half user, top kernel

Driver structure

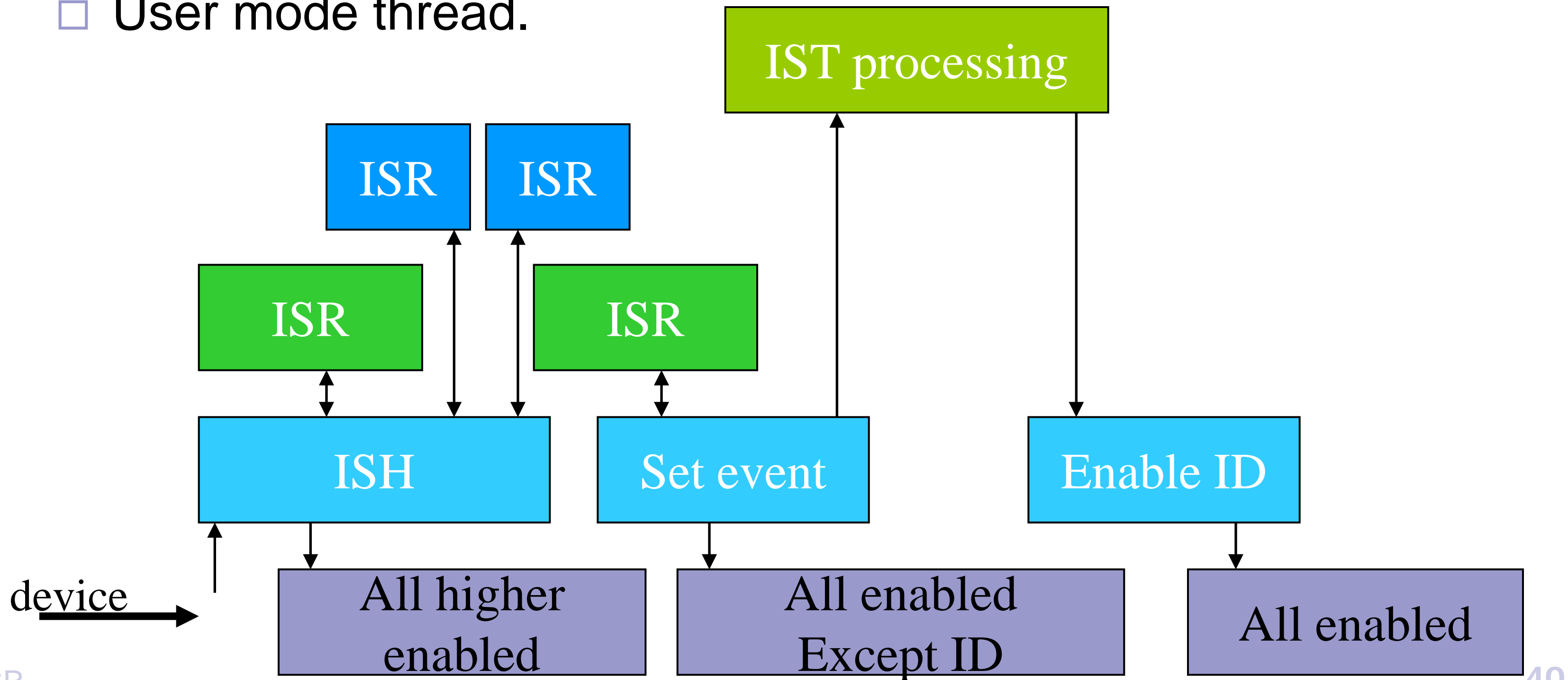
- Driver = DLL with particular interface points.
- Hosted by a device manager process space
- Handle interrupts by dedicated IST thread.
- Synchronize driver and application via critical sections and MUTEXes.

Device manager

- Always-running user-level process.
- Contains the I/O Resource Manager.
- Loads the registry enumerator DLL which in turn loads drivers.
  - RegEnum scans registry, loads bus drivers.
  - Bus driver scans bus, locates devices.
  - Searches registry for device information.
  - Loads appropriate drivers.
  - Sends notification that interface is available.
- Provides power notification callbacks.

Interrupt handling

- Interrupt service routine (ISR):
  - Kernel mode service.
  - May be static or installable.
- Interrupt service thread (IST):
  - User mode thread.



Kernel scheduler

- Two styles of preemptive multitasking.
  - Thread runs until end of quantum.
  - Thread runs until higher priority thread is ready to run.
- Round-robin within priority level.
- Quantum is defined by OEM and application.
- Priority inheritance to control priority inversion.
- 256 total priorities.
  - Top 248 can be protected by the OEM.

Summary

- SW
  - MPEG decode etc.
- Middleweare
  - E.g DCE, CORBA
- RTOS
  - E.g TinyOS, eCos, RT-Linux, WinCE

Sources and References

- Peter Marwedel, "Embedded Systems Design," 2004.
- Wayne Wolf, "Computers as Components," Morgan Kaufmann, 2001.
- Nikil Dutt @ UCI
- Mani Srivastava @ UCLA