Real-Time OS Frameworks

Sandeep D'souza

Lecture #7

Outline of Lecture

- Approaches limiting Real-time and Non-real-time Task Interactions
  - Compliant Kernel Approach
  - Dual/Thin Kernel Approach
- Approaches that integrate Real-time and Non-real-time tasks
  - Core Kernel Approach
  - Resource Kernel Approach

Approaches to Real-Time Linux

- Approaches limiting Real-time and Non-real-time Task Interactions
  - Compliant Kernel Approach
    - LynxOS/Blue Cat Linux
  - Dual Kernel Approach
    - RTLinux/RTAI
- Approaches that integrate Real-time and Non-real-time tasks
  - Core Kernel Approach
    - Monta Vista Linux, TimeSys Linux
  - Resource Kernel Approach
    - Linux/RK

Linux Internals: Scheduling

- Schedulable Entities
  - Processes
    - Real-Time Class: `SCHED_FIFO` or `SCHED_RR`
    - Time-Sharing Class: `SCHED_OTHER`
  - Real-time processes have
    - Application-defined priority
    - Higher priority than time-sharing processes
- Non-Schedulable Entities
  - Interrupt Handlers
    - Have priorities, and can be nested
  - "Bottom Halves" & Task Queues
    - Run on schedule, ret from system call, ret from interrupt

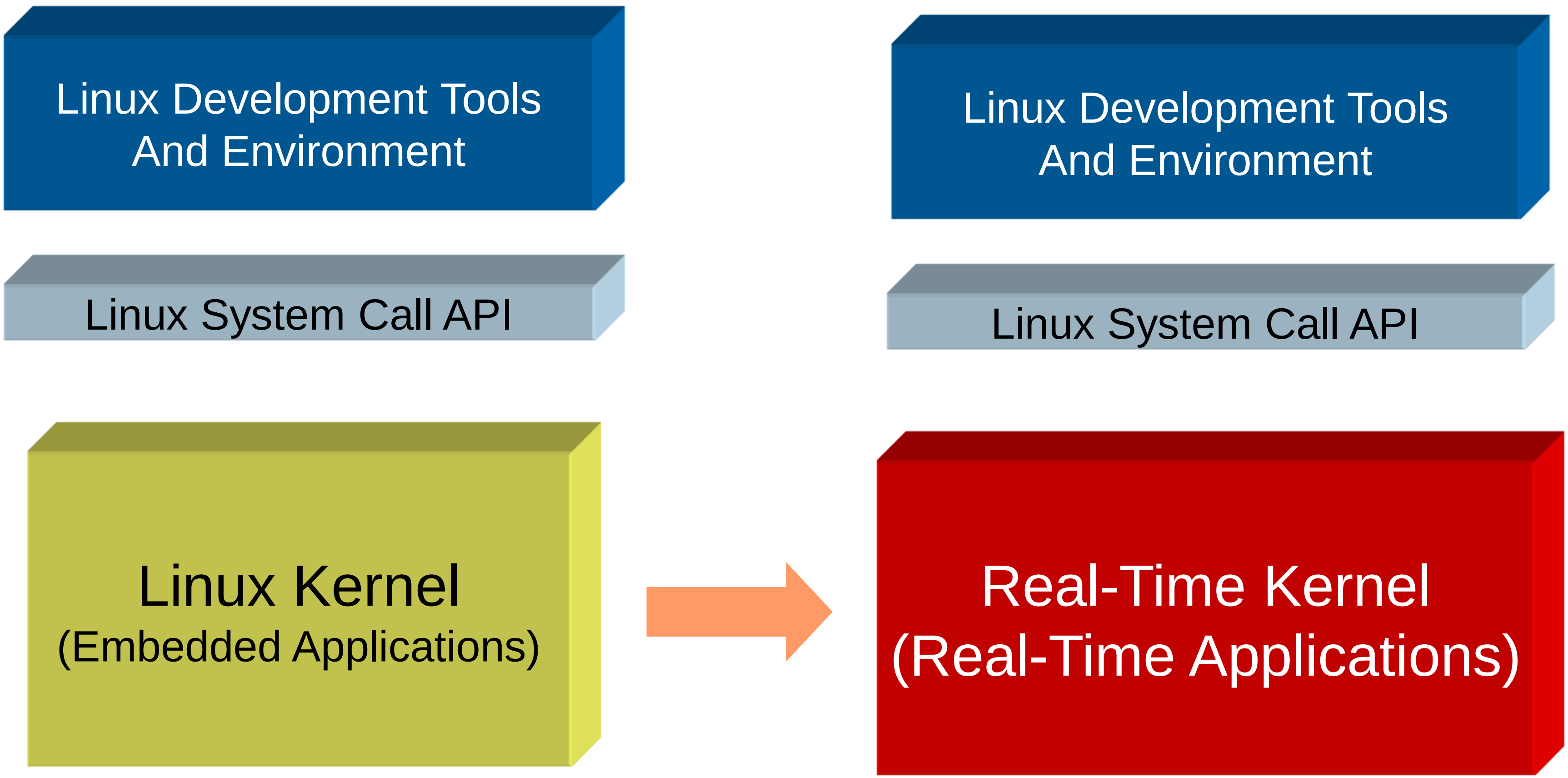
Linux and Real-Time: Traditional Problems

- Timer Granularity
  - Many real-time tasks are driven by timer interrupts
  - In Standard Linux, the timer was set to expire at 10 ms intervals
    - Beginning to change with usage of high-resolution timer and timestamp counters
- Scheduler Predictability
  - The Linux scheduler used to keep tasks in an unsorted list
  - Requires a scan of all tasks to make a scheduling decision
  - Scales poorly as number of tasks increases, and is especially poor for real-time performance
- Various subsystems NOT designed for real-time use
  - Network protocol stack
  - Filesystem
  - Windows manager

Approaches to Real-Time Linux

- Compliant Kernel Approach
- Dual Kernel Approach
- Core Kernel Approach
- Resource Kernel Approach

Compliant Kernel Approach



Compliant Kernel Approach

- Basic Claim
  - Linux is defined by its API and not by its internal implementation
  - The real-time kernel is a non-Linux kernel
- Implications
  - No benefits from the Linux kernel
  - Not possible to benefit from the Linux kernel evolution
  - Not possible to use Linux hardware support
  - Not (always) possible to use Linux device drivers

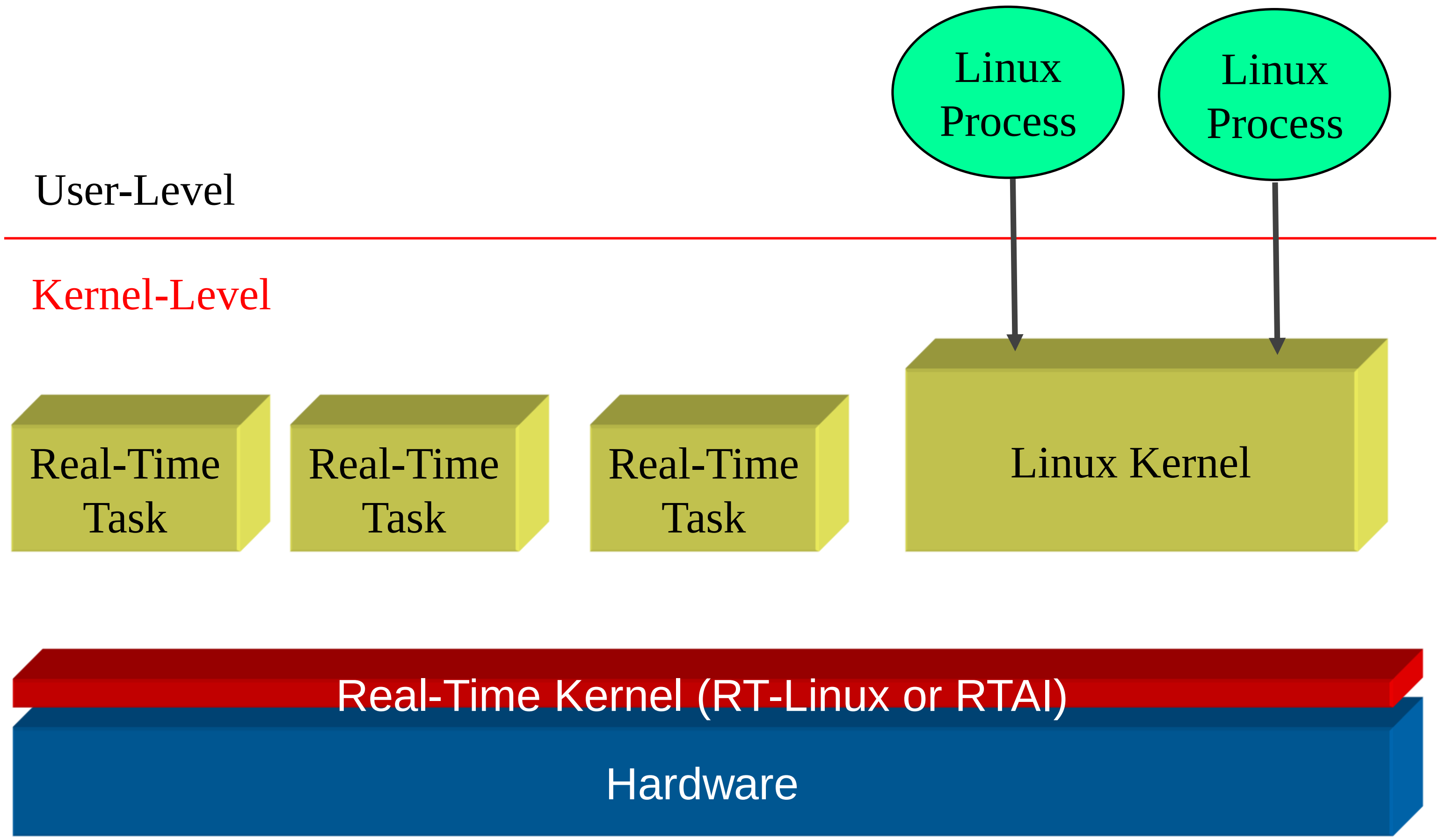
Compliance

- 100% Linux API
  - Support all of Linux kernel API
- Implications
  - + Any Linux application can run on real-time kernel
    - Development can be done on a Linux host, with a rich set of host tools for development
  - + All Linux libraries are trivially available to run on a real-time kernel
    - Third-party software
  - – Achieving 100% Linux API is non-trivial
    - Consider the amount of effort put into Linux kernel development

Approaches to Real-Time Linux

- Compliant Kernel Approach
- Dual Kernel Approach
- Core Kernel Approach
- Resource Kernel Approach

The Thin Kernel Approach



- Real-time tasks do NOT use the Linux API or Linux facilities
- Failure in any real-time task crashes the entire system

Approaches to Real-Time Linux

- Compliant Kernel Approach
- Dual Kernel Approach
- Core Kernel Approach
- Resource Kernel Approach

Core Kernel Approach

- Basic Ideas
  - Make the kernel more suitable for real-time
  - Ensure that the impact of changes is localized so that
    - Kernel upgrades can be easily incorporated
    - Kernel reliability and scalability is not compromised
- Mechanisms
  - Static Configuration
    - Can be configured at compile time
  - Dynamic Configuration
    - Using loadable kernel modules

Core Kernel Approach

- Allows the use of most, if not all, existing Linux primitives, applications, and tools.
  - Need to avoid primitives that can take extended time in the kernel
- Allows the use of most existing device drivers written to support Linux.
  - Need to avoid poorly written drivers that unfairly hog system resources
- Robustness and Reliability
  - Core kernel modifications can affect robustness, but source is available and extensive testing can be done.

Approaches to Real-Time Linux

- Compliant Kernel Approach
- Dual Kernel Approach
- Core Kernel Approach
- Resource Kernel Approach

Resource Kernel

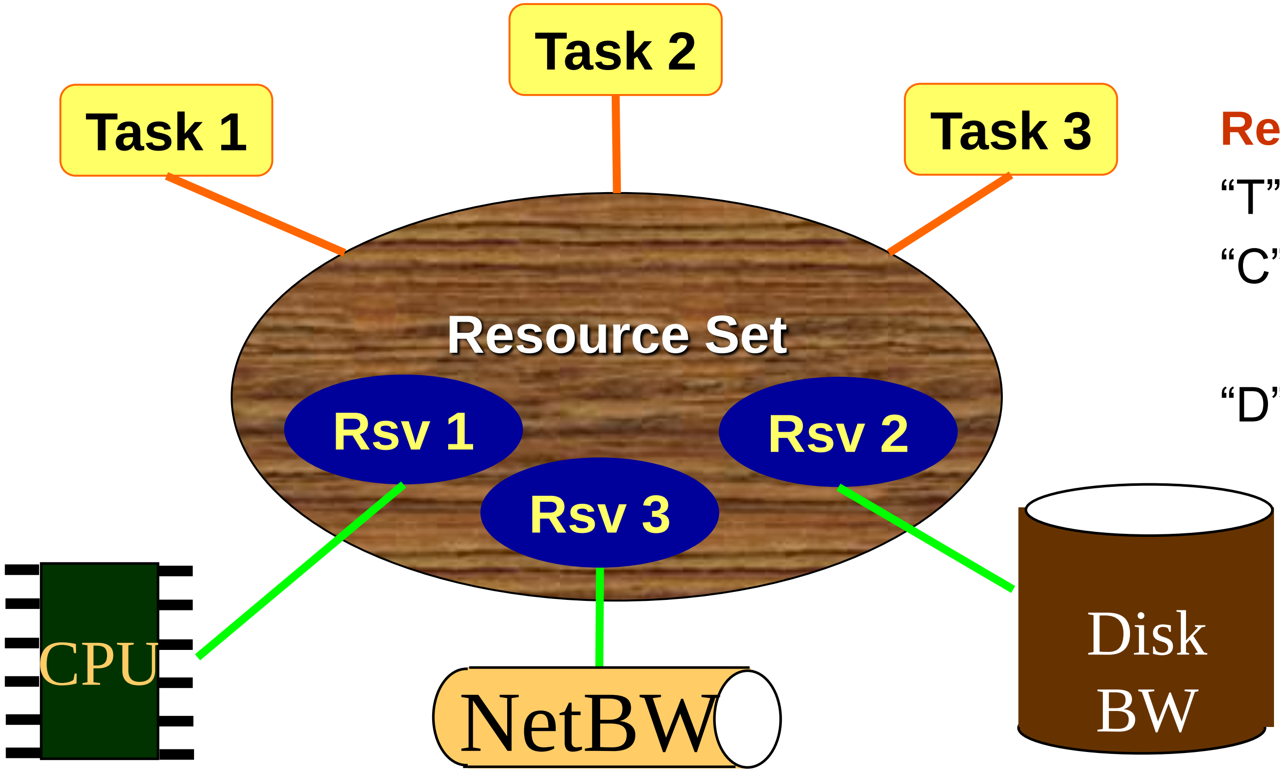
- A Kernel that provides to Applications Timely, Guaranteed, and Enforced access to System Resources
- Allows Applications to specify only their Resource Demands, leaving the Kernel to satisfy those Demands using hidden management schemes

Protection in Resource Kernels

- Each application (or a group of collaborating applications) operates in a "virtual machine":
  - a machine which consists of a well-defined and guaranteed portion of system resources
    - CPU capacity, disk bandwidth, network bandwidth, and memory resource
- Multiple virtual machines can run simultaneously on the same physical machine
  - guarantees available to each resource set is valid despite the presence of other (potentially mis-behaving) applications using other resource sets

Resource Kernel

- **A Kernel that provides to applications *Timely*, Guaranteed, and Enforced access to System Resources**
- Allows Applications to specify only their Resource Demands
  - leaving the Kernel to satisfy those Demands using hidden management schemes



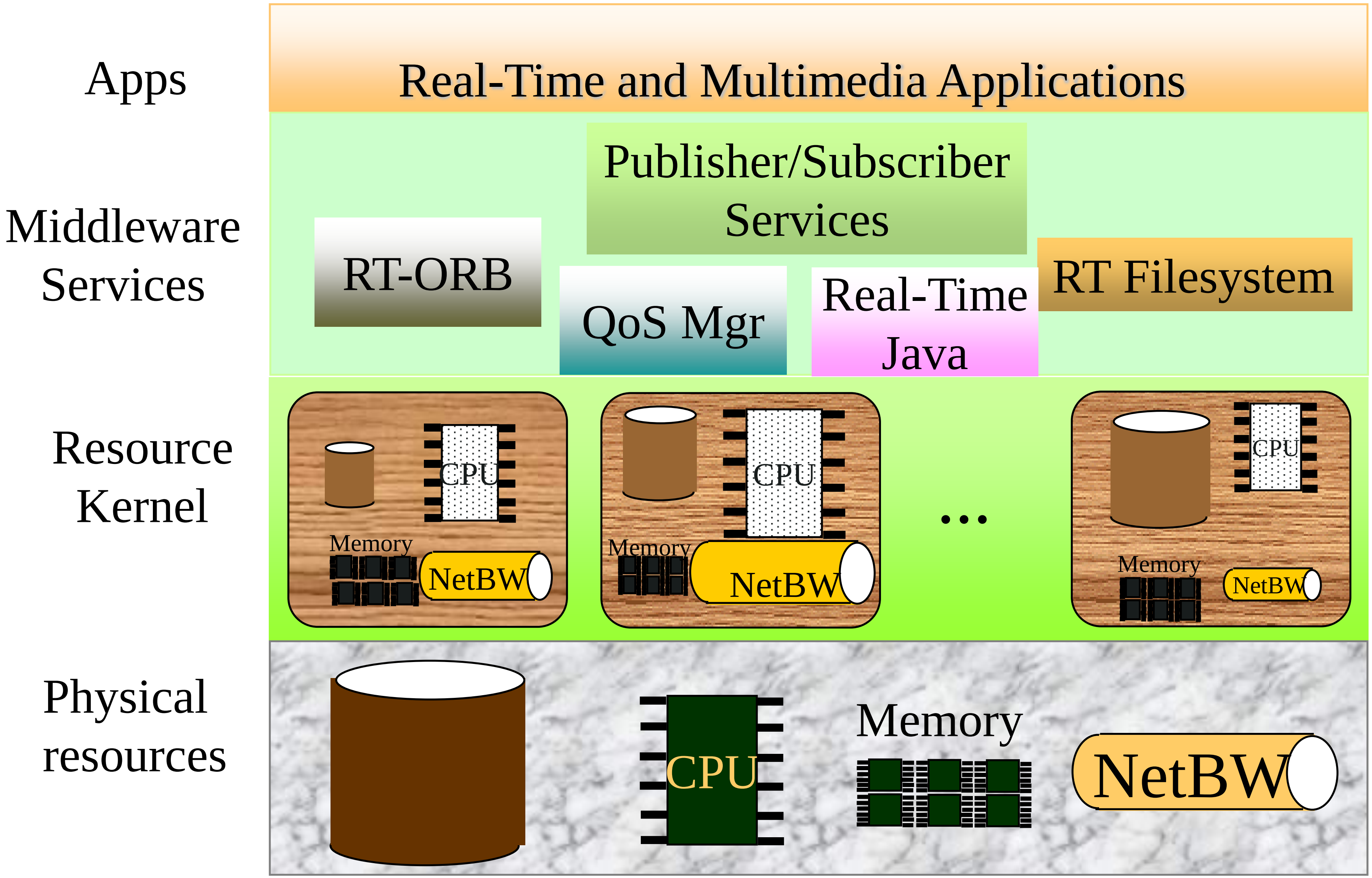
Reservation Parameters

"T": Period (1/*f*)

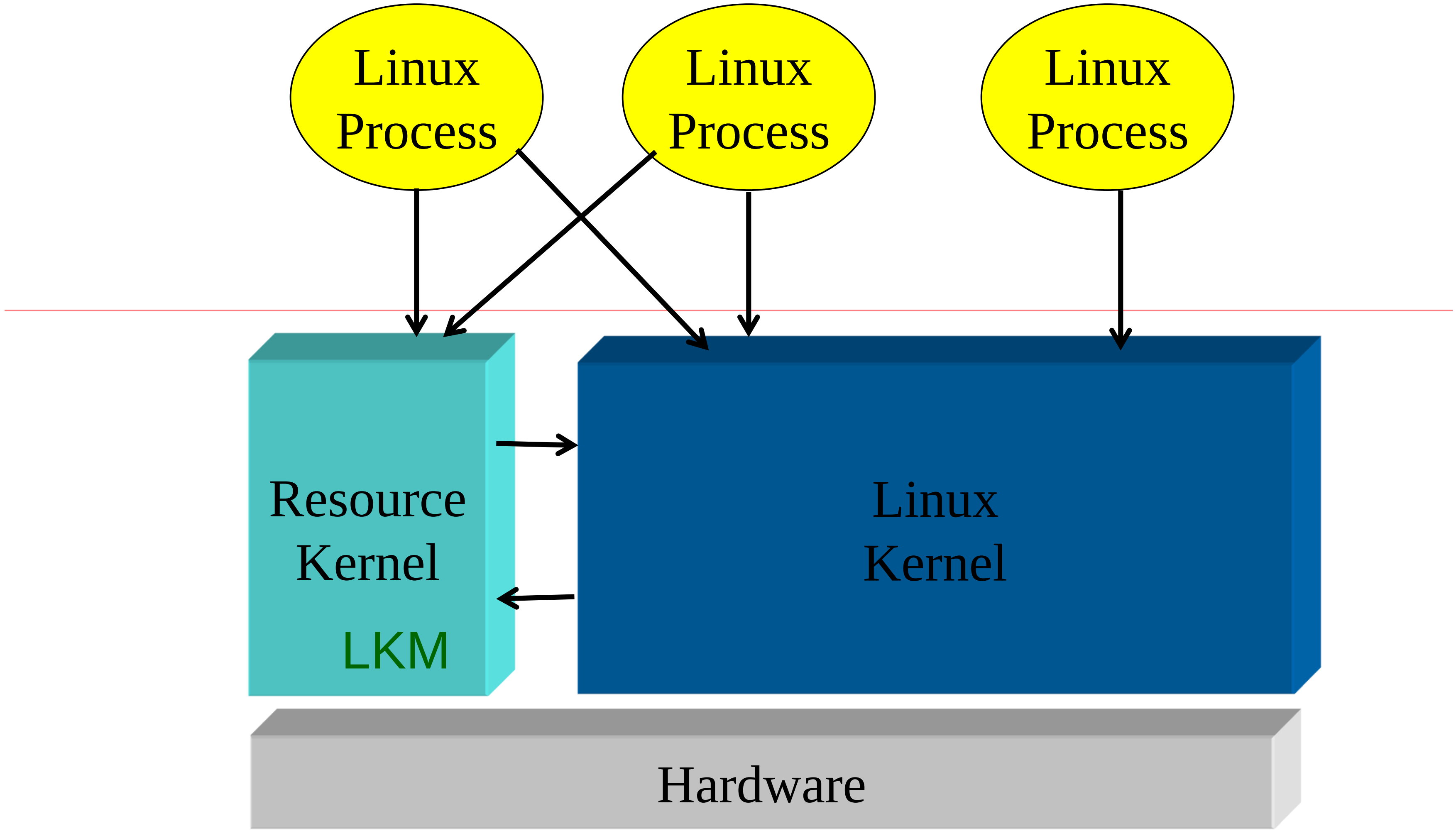
"C": Execution time within period

"D": Deadline within period

“Resource Kernel” Architecture



Linux Resource Kernel Architecture



Reserves and Resource Sets

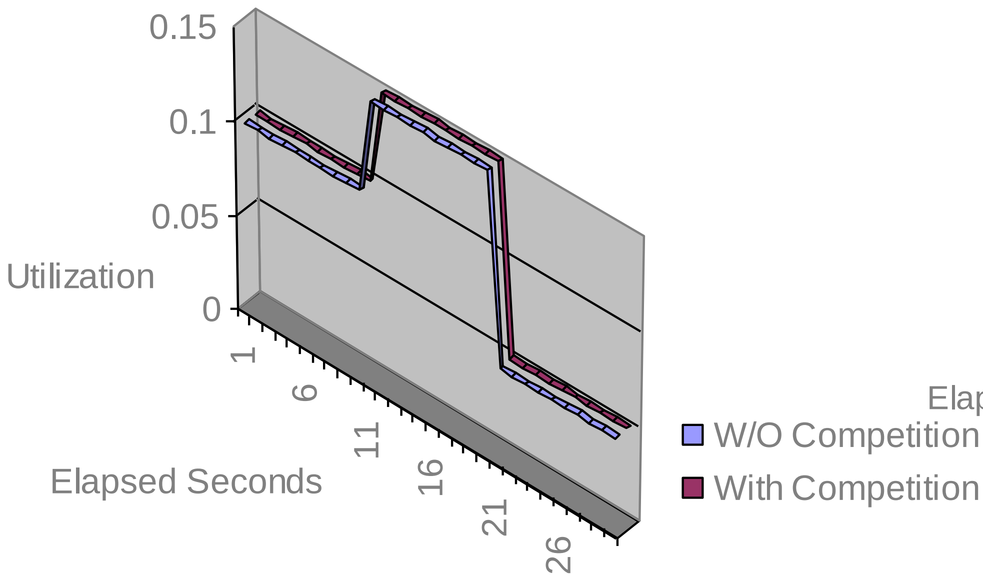
- Reserve
  - A Share of a Single Resource
  - Temporal Reserves
    - Parameters declare Portion and Timeframe of Resource Usage
      - E.g., CPU time, link bandwidth, disk bandwidth
  - Spatial Reserves
    - Amount of space
      - E.g., memory pages, network buffers
- Resource Set
  - A set of resource reserves
  - Zero, one or more processes can be bound to a resource set.

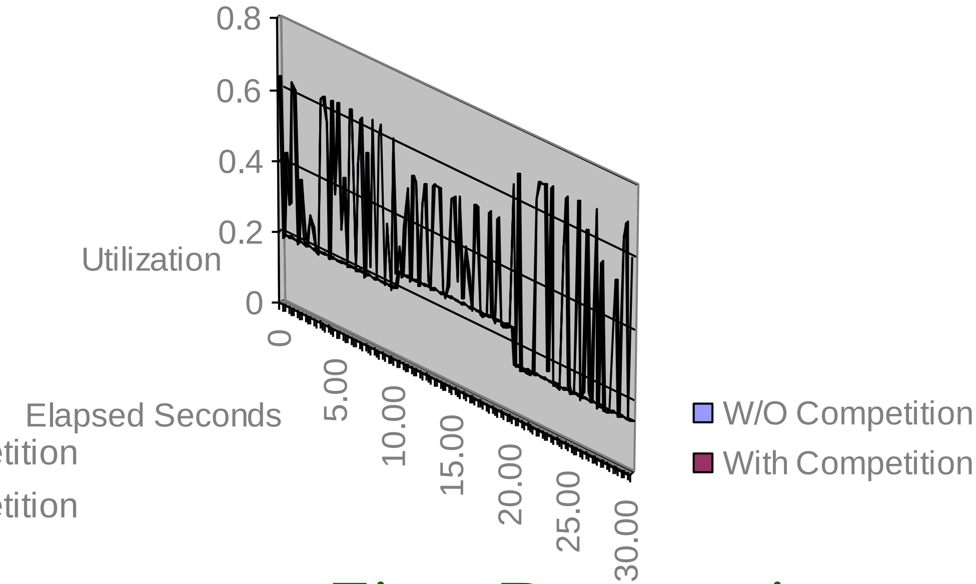
Linux/RK Abstractions

Linux/RK supports several abstractions and primitives for real-time scheduling of processes with real-time and QoS requirements:

- **Resource reservations with latency guarantees**
  - CPU cycles
  - Network bandwidth
  - Disk bandwidth
- Support for **periodic tasks**.
- Support for 256 real-time fixed-priority levels.
- **High-resolution timers and clocks**.
- **Bounding of priority inversion** during synchronization operations.
- Wiring down of memory pages.

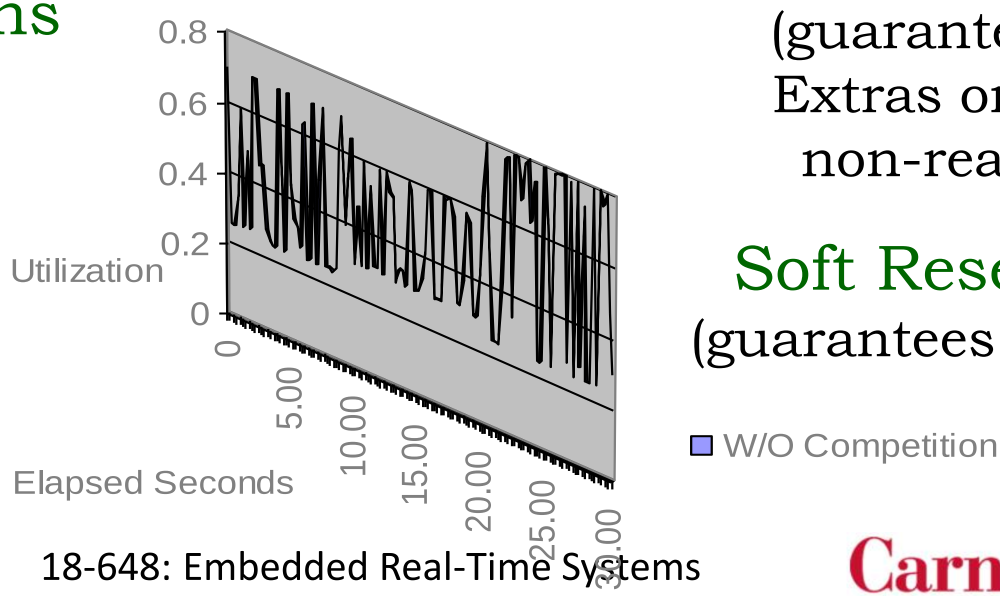
Reservation Types





Hard Reservations
(guarantees with **No** extras even if resource is idle)

Firm Reservations
(guarantees with Extras only if no non-real-time)



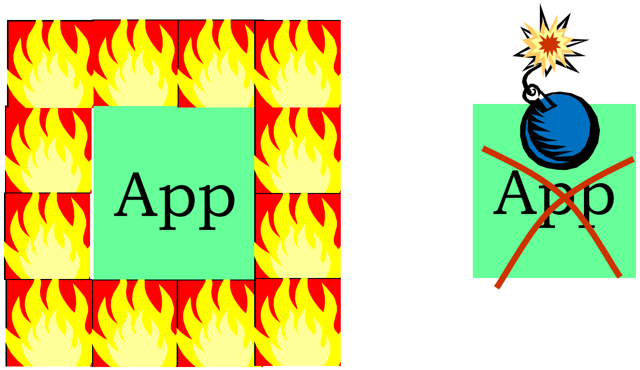
Soft Reservations
(guarantees with extras)

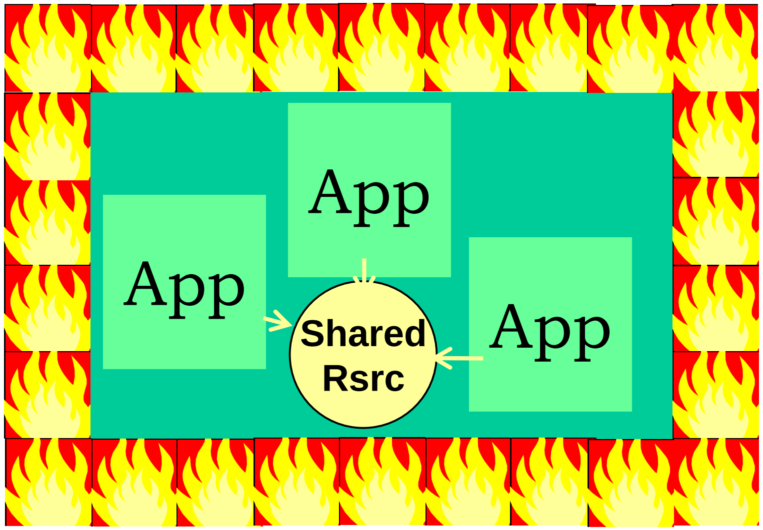
Performance Overhead of *Hierarchical* Reservations

- **Linear with the height of the hierarchy**
  - reservation replenishment and enforcement
- **Constant**
  - admission control (only *local* schedulability analysis)
  - scheduling (internal priority mapping and disabling/re-enabling of process eligibility to be scheduled.
- **Constraints**
  - reservation period must be greater than twice the parent's reservation period
- **Hidden overhead**
  - Higher degree of interrupts, because of replenishment, enforcement timers going off more frequently.

Degrees of Temporal Isolation

- Different degrees of temporal isolation in the presence of resource-sharing
  - **Strict Isolation**: the timing behavior of an application is not affected by the timing misbehavior of *any* other application
    - RK applications in the absence of logical sharing of resources
  - **Non-Strict Isolation**: traditional priority-driven systems
  - **Weak Isolation**: timing behavior is not affected by the timing mis-behavior of applications with which no logical resources are shared.





Resource-Sharing Protocols in RK

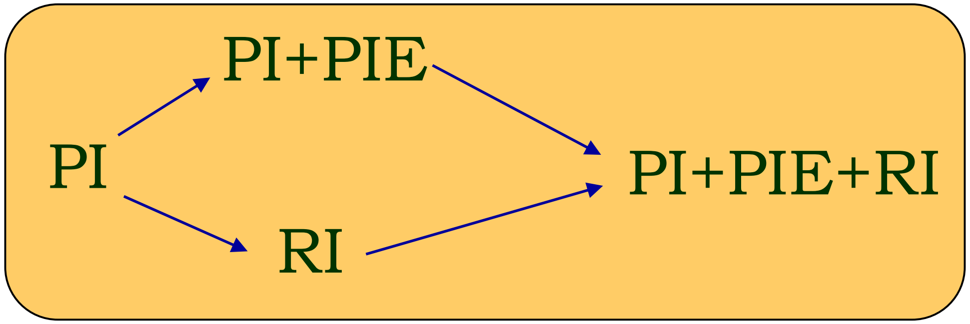
- **Analogues to Priority Inheritance and Priority Ceiling Protocols** in Resource kernels
- Temporal isolation **can** only be weak
  - under logical resource-sharing using mutexes and client-server architectures
  - timeout and restart schemes may need to be applied.

Priority Ceiling Protocol Equivalents

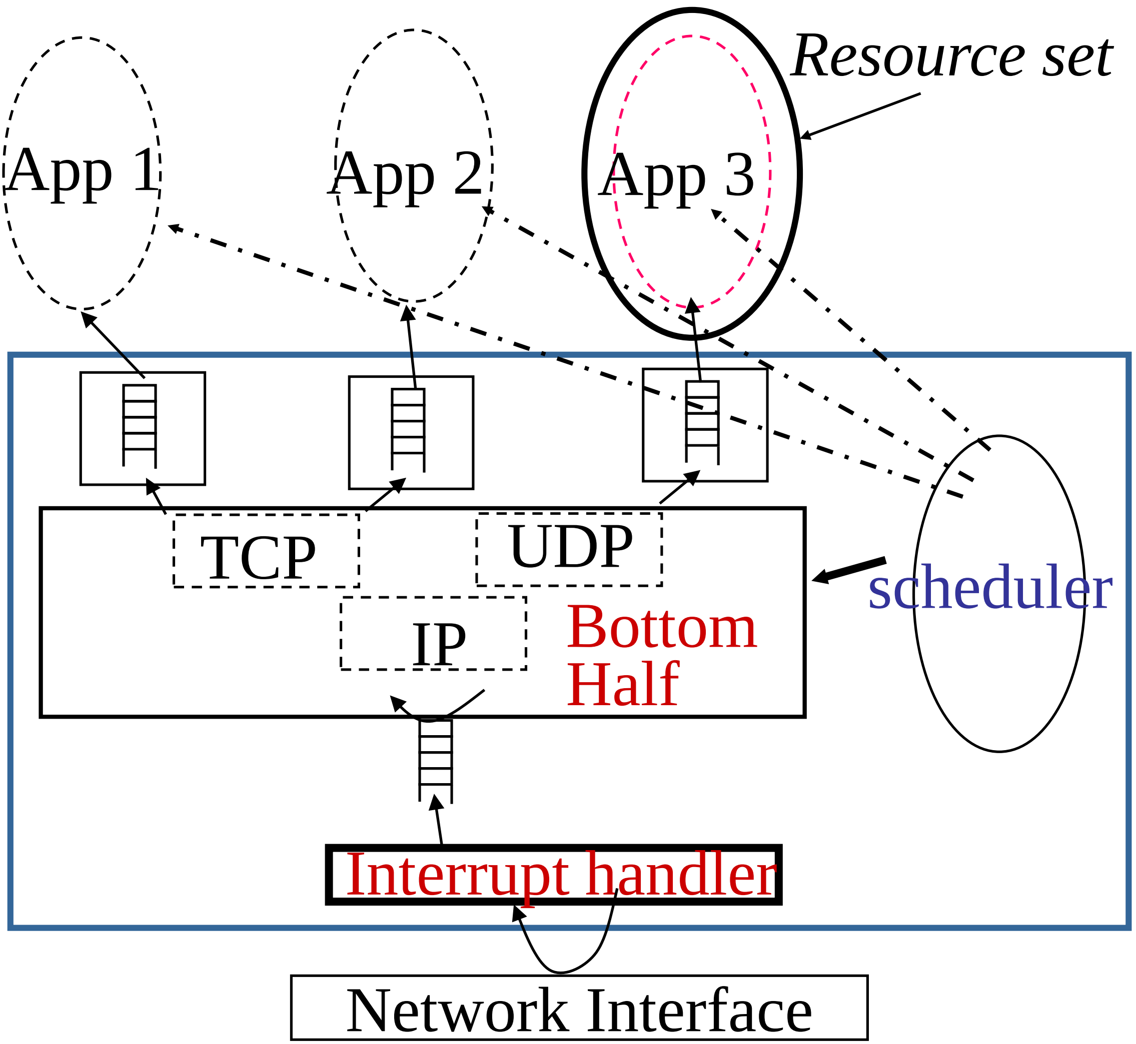
- **Single-Reserve PCP**
  - Assign one reservation to the logical resource execution
  - Very pessimistic allocation is required to maintain PCP semantics
- **Multi-Reserve PCP**
  - Has the same schedulability analysis as traditional PCP
  - Requires special support in RK
  - Can be applied to client-server models
    - Pass client's reserve to server along with request - charges

Priority Inheritance Protocol Equivalents

- **Priority Inheritance (PI)**
  - Server runs at the priority of the highest priority client waiting for server.
- **Priority Inheritance with Priority Inversion Enforcement (PIPIE)**
  - Enforce the duration of priority inversion encountered by any task
    - Can never exceed the amount specified at admission control
    - Need to track multiple tasks' priority inversions simultaneously
- **Reserve Inheritance (RI)**
  - Server usage is charged to client's reservation + inherit the highest priority of any client waiting for server
- **PIPIE + RI**



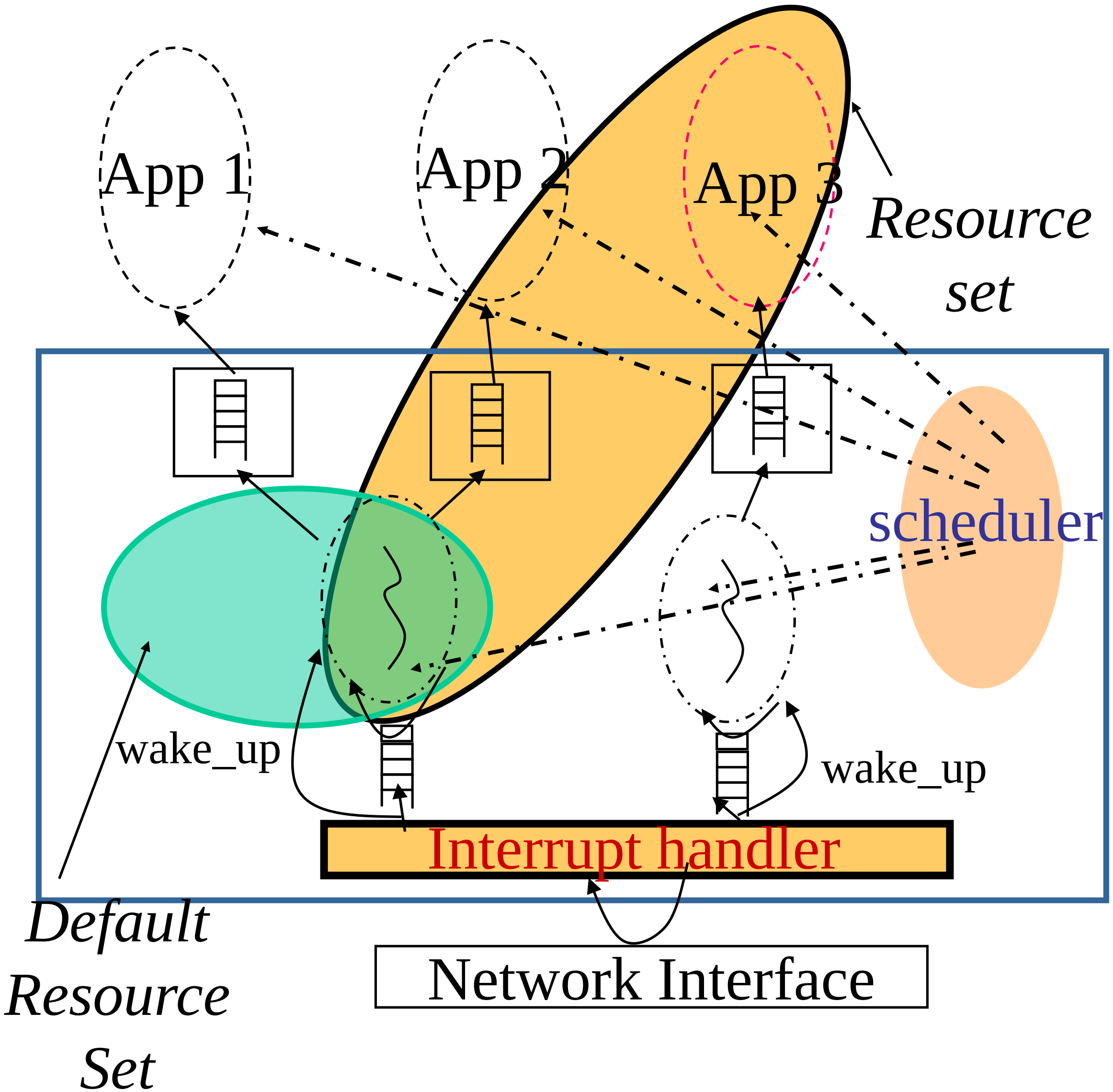
Network Processing Architecture

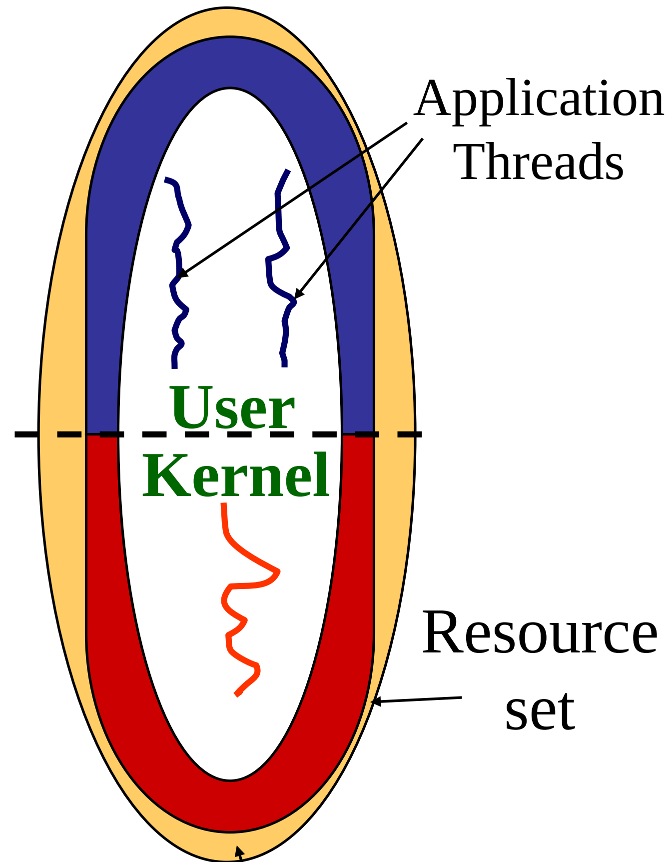


Problems

- Non-preemptive System calls
  - Priority Inversion
- "Eager" Receiver Processing
  - Interrupt-driven
    - Priority(capturing packet) > Priority(protocol processing) > Priority (application processing)
- Lack of effective load shedding
- Lack of traffic separation
- Inappropriate resource accounting

Threaded Network Processing





Application domain extends into the kernel, and its activity is controlled

Network *and* CPU Service Guarantees

- Reduction in non-preemptibility
- Control of receiver overload (*receive -livelock*)
- Prevention of scheduling disruption
- Separation of individual flows and proper resource accounting
- Packet scheduling for QoS (Quality of Service) guarantees

Summary

- OS Approaches limiting Real-time and Non-real-time Task Interactions
  - Compliant Kernel Approach
  - Thin Kernel Approach
- OS Approaches that integrate Real-time and Non-real-time tasks
  - Core Kernel Approach
  - Resource Kernel Approach