L4: Smart Objects Software

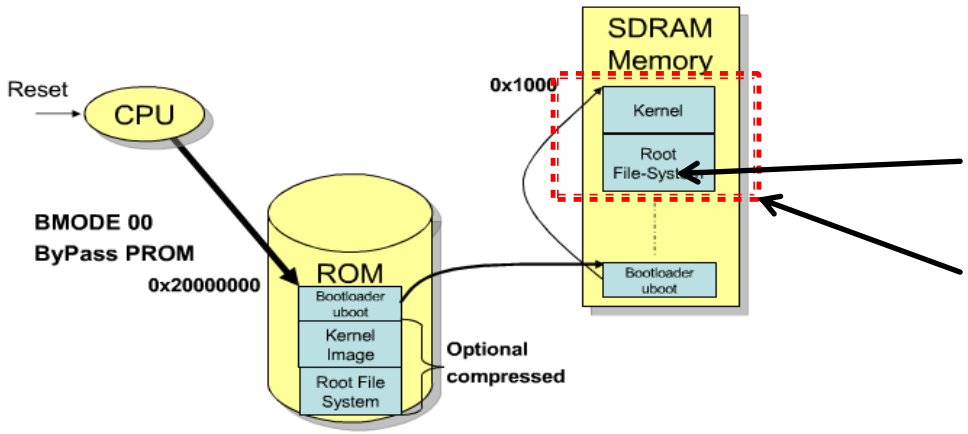
IT3779 Smart Object Technologies

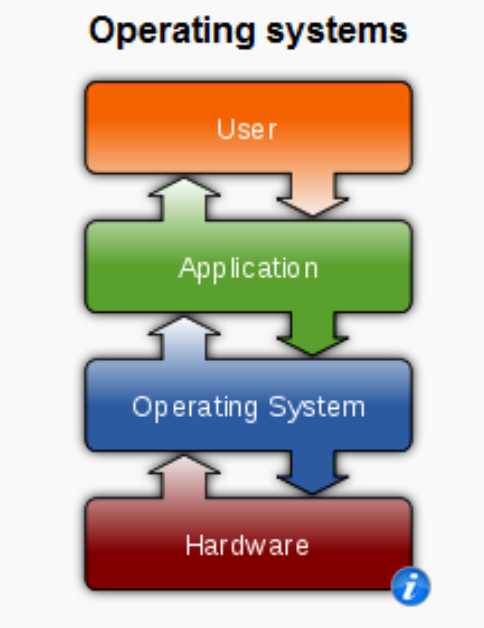
Outline

- Introduction to Operating System
- Review of Operating System Concepts
- Design Considerations for Smart Object Operating System
- Case Study of a Smart Object Operating System: TinyOS

Introduction to Operating System

- What is an Operating System?
  - A set of programs that manage computer hardware resources and **provide common services** for application software





Types of Operating Systems

Server
running UNIX

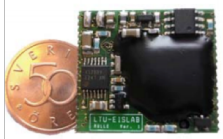
PC running
Linux,
Windows OS

Mobile devices
running Andorid, iOS

IBM mainframe
running Z/OS

Embedded OS

Embedded Operating System



Heavy-weight processes with memory protection

Linux

eCos, OSE, Mantis

Lightweight multi-threading, preemption

Contiki

Event-driven system

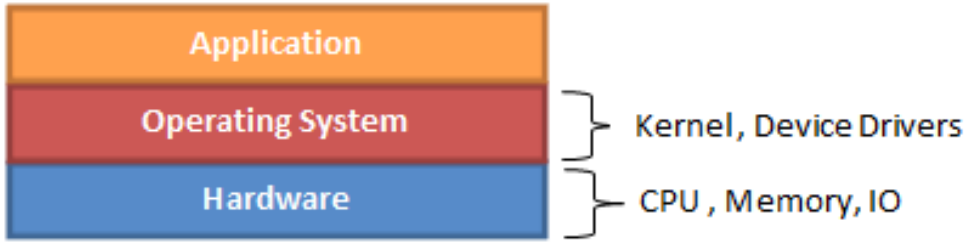
TinyOS

Simple control loop

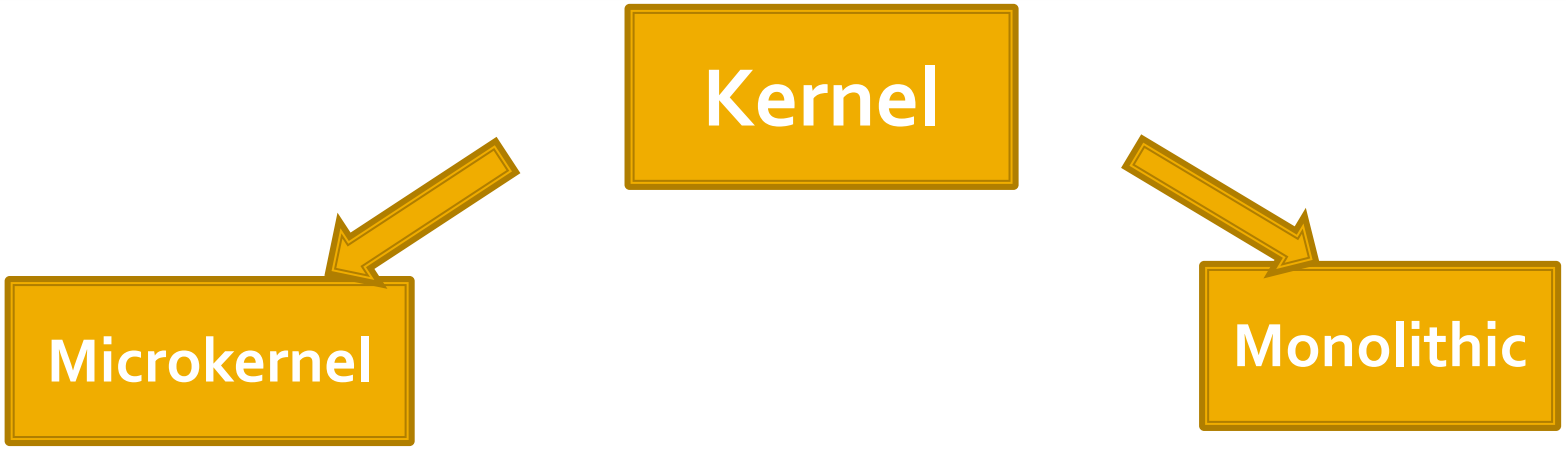
.NET MF based

Review of Operating System Concepts

- What is a Kernel?
  - Main component of an operating system
  - Bridge between applications and actual data processing done at the hardware level
- Device Drivers
  - Software written to facilitate access to hardware resource through the kernel



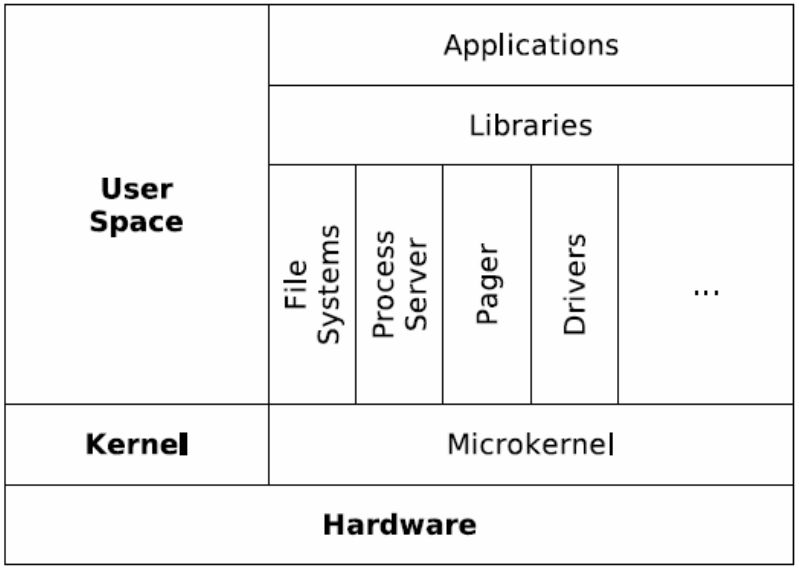
Types of Kernels



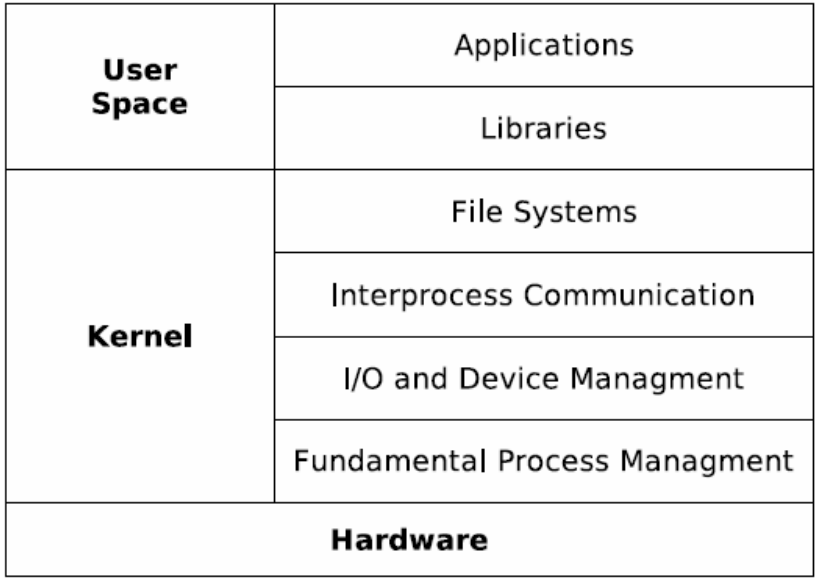
- Some system functionality moved out of the traditional "kernel".
- Contain minimal OS services such as memory management, multitasking, and inter-process communication.
- Other services, such as networking, are implemented in user-space programs

- All OS services run along with the main kernel thread
- All OS services reside in the same memory area
- Contain all the operating system core functions and the device drivers

Types of Kernels



Microkernel based operating system



Monolithic based operating system

Multi-tasking versus Single-tasking

Multi-tasking

- Perform several operations concurrently at the same time
- Share resource between operations
- More overheads for each operation
- Co-operative or Pre-emptive multi-tasking

Single-tasking

- Perform only one operation at one time
- Task has many operations.
- Each operation should have short duration
- Suitable for motes which typically perform specific function.
- May use multi-threading.

Multithreading programming versus Event driven programming

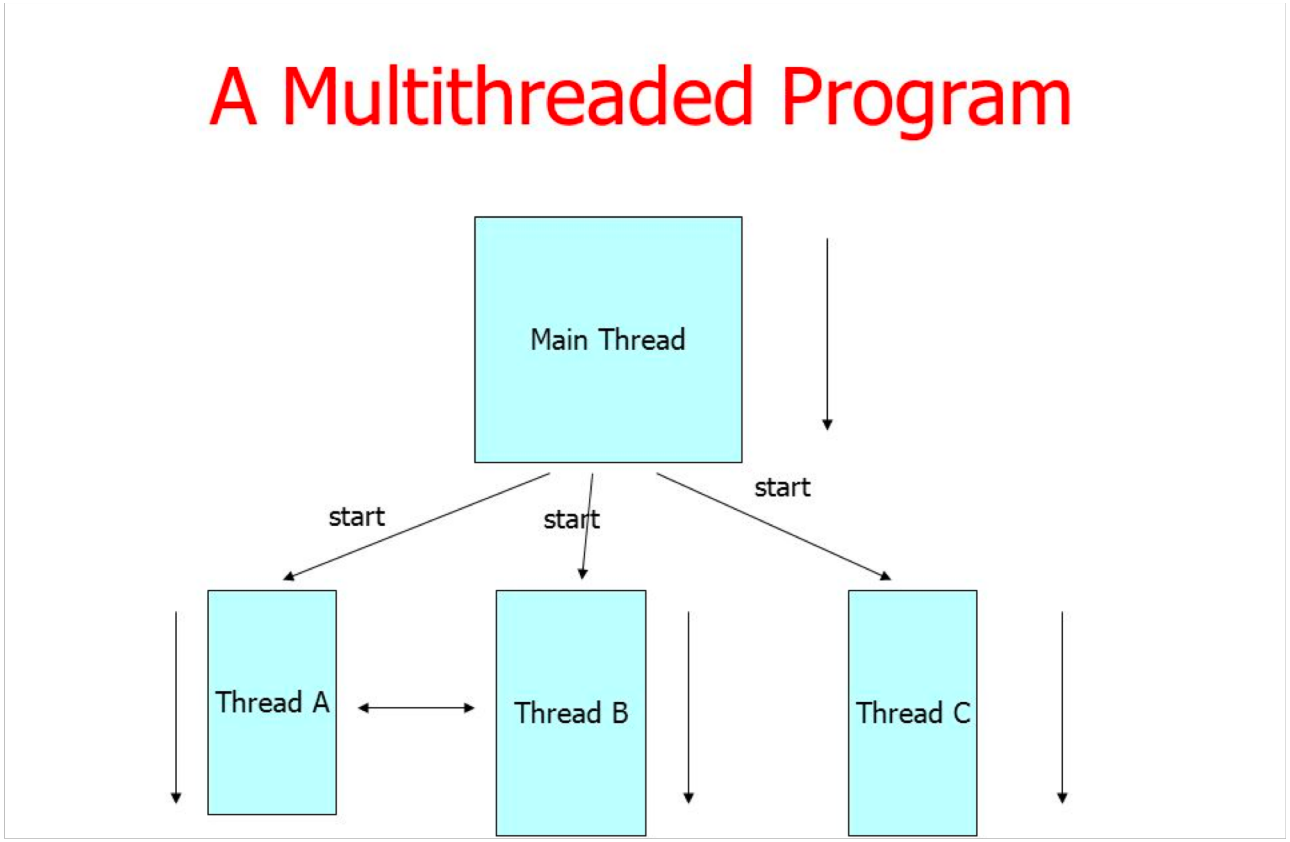
Multi-threading

- A big task is broken into smaller tasks
- Multiple threads (Smallest unit of processing) are created and assigned to run the different tasks
- Threads run **independently** from each other
- Threads can run **concurrently**

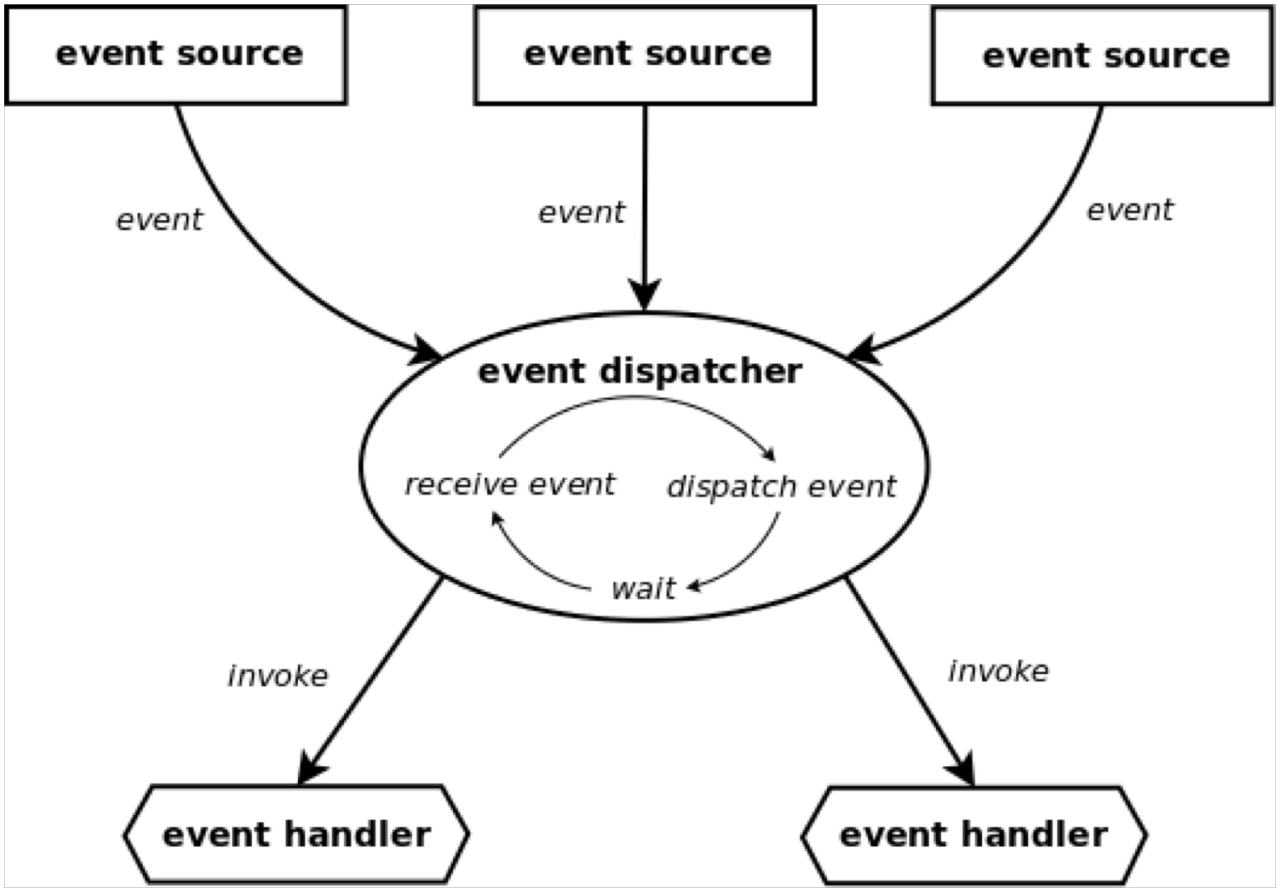
Event driven

- Program flow driven by events
- Event-handlers, which are codes written to handle specific events are registered with OS
- When an event occurs, its registered **event-handler** is called to handle it

Multi-threading Programming Model



Event-driven Programming Model



Why is 'Traditional' OS is not suitable for Smart Objects?

- Design for general purpose :Not suited for smart objects which have specific requirements e.g. low power, limited memory etc
- Many functionalities & programming APIs : large code base
- Protection between "untrusted" applications and kernel: high overhead for crossing kernel/user boundary & interrupt handling
- Multi-threaded architecture: Large number of threads results in large memory used

Windows, Linux

IBM mainframe running Z/OS

SUN Server running UNIX

Challenges & Issues for Smart Object Operating Systems

- Restricted Resources
  - Battery has to last for long duration
  - Processing Power is limited; Computation intensive tasks must be scheduled so as not to delay higher priority task
  - OS must be able to fit into a small memory footprint
- Multi-tasking
  - At any one time, Smart Object may be doing several task e.g. sending data, reading temperature etc
- Customizability
  - OS must be customizable to support specific industry applications

Operating Systems for Wireless Sensor Networks: A Survey
Technical Report
Adi Mallikarjuna Reddy V AVU Phani Kumar, D Janakiram, and G Ashok Kumar

Design Considerations for Smart Object Operating System

- Architecture
- Programming Model
- Scheduling
- Memory Management and Protection
- Communication Protocol Support
- Resource Sharing
- Support for Real-Time Applications

Design Considerations for Smart Object Operating System

- Operating System Architecture Requirements
  - Have small memory footprint
  - Allow extensions to the kernel if required
  - Be flexible i.e., only application-required services get loaded onto the system.

Design Considerations for Smart Object Operating System

- Programming Model : Multi threading or Event Driven Programming?
  - Multi threaded Model
    - Familiar to programmers
    - Rather resource intensive, hence not suited for SO
  - Event Driven Model
    - More suited for scarce resource application
    - Not familiar with traditional application developers
  - Requirement **: Both *light weight* Multi threaded & event programming model should be supported**

Design Requirements for Smart Object Operating System

- Scheduling
  - Central Processing Unit (CPU) scheduling determines the order in which tasks are executed
  - In traditional OS, objective of scheduling is to maximize throughput and resource utilization
  - Requirement : **Scheduling algorithm must be memory and power efficient**

Design Requirements for Smart Object Operating System

- Memory Management
  - Two commonly used techniques : static memory management and dynamic memory management
  - Static memory management
    - Simple when dealing with scarce memory resources.
    - But results in inflexible systems because run-time memory allocation cannot occur.
  - Dynamic memory management
    - More flexible system because memory can be allocated and de-allocated at run-time

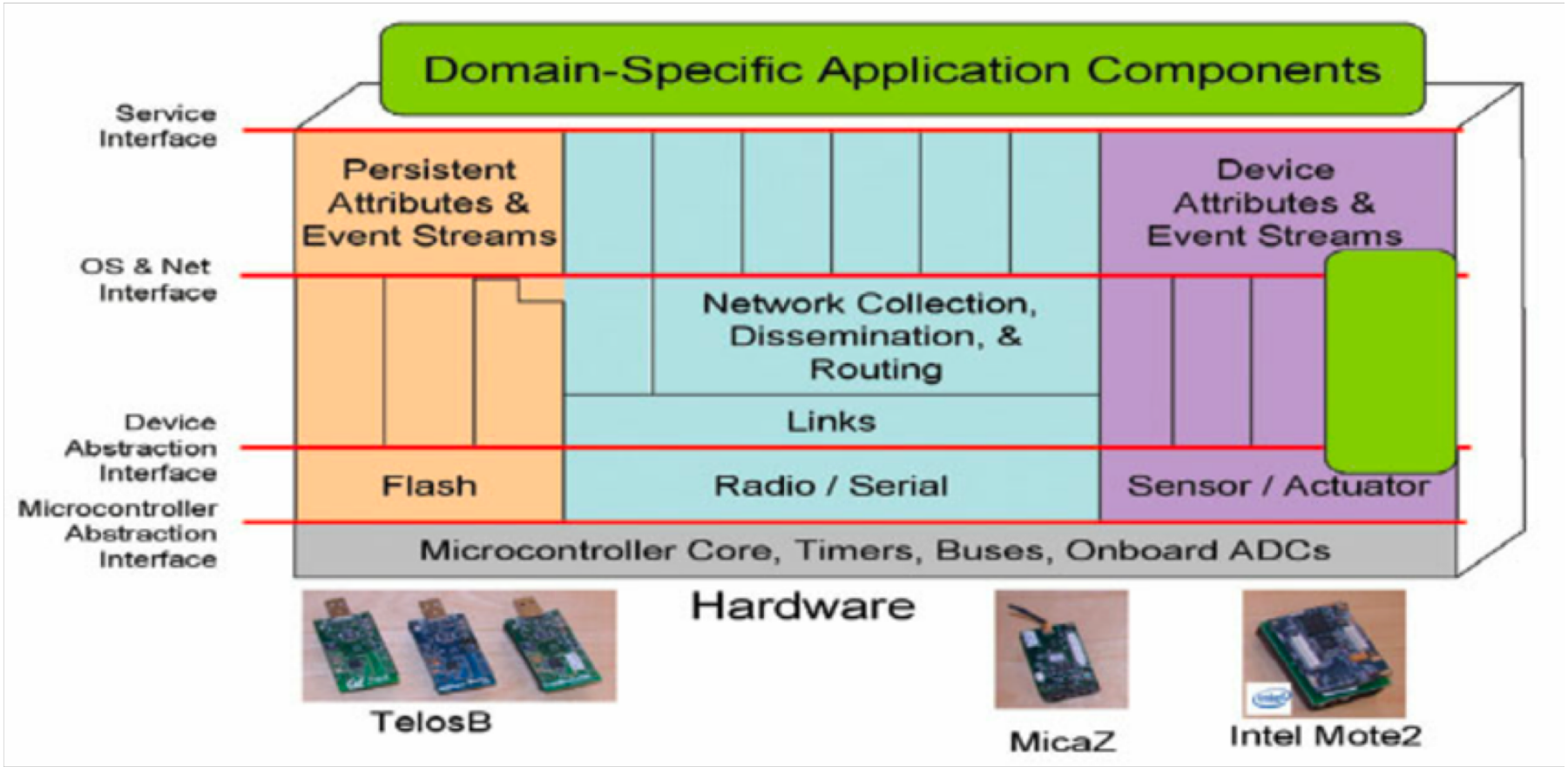
Design Considerations for Smart Object Operating System

- Resource Sharing
  - Resources sharing is important when multiple programs are concurrently executing
  - Requirement : **Resource sharing mechanism should be available**
- Support for Real-Time Applications
  - Smart Object may be used to monitor a mission critical system.
  - Requirement : **Real-time scheduling algorithms should be supported to meet the deadlines of hard real-time tasks**

Case Study of a Smart Object Operating System : TinyOS

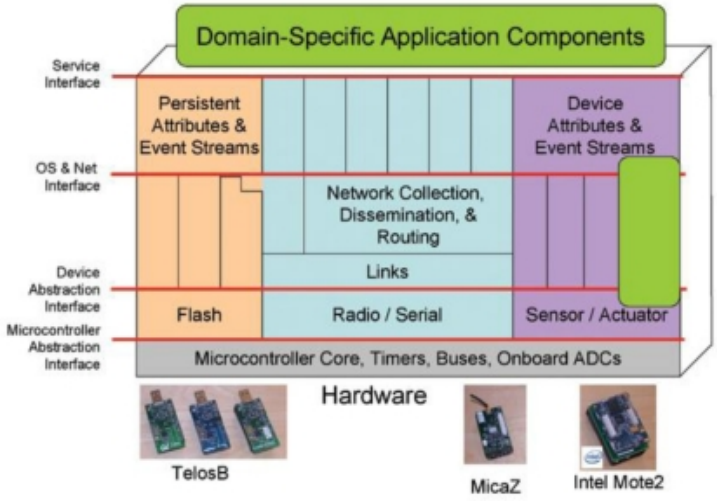
- Uses unique software architecture that was designed specifically for resource-constrained applications
- An open source, flexible, component based, and application-specific operating system
- Very small memory requirements : has a footprint that fits in 400 bytes
- Component library includes
  - Network protocols
  - Sensor drivers
  - Data acquisition tools

TinyOS Architecture



TinyOS

- Architecture
  - Has a monolithic architecture
  - Uses the component model: according to the requirements of an application, different components are glued together with the scheduler to compose a static image that runs on Smart Object
  - Thus, it has a very small memory footprint



TinyOS Architecture

TinyOS

- Programming Model
  - Supports both event driven programming and multi-threading
  - An event-based programming approach permits greater concurrency
  - Multi-threading approach offers an intuitive programming model
  - Thus, TinyOS provides the ease of a threading programming model coupled with the efficiency of an event driven kernel

TinyOS

- Two-level scheduling (Event and Task)
  - Supports 2 levels of scheduling – Events & Task
  - Scheduling of tasks is based on First In First Out algorithm
  - An Event has high priority than Task and can preempt task
  - Events can also preempt each other
  - To conserve resources, when idle, scheduler shutdowns Smart Object except for clock

TinyOS

- Memory Management
  - Incorporates memory safety i.e. all array errors and pointers are captured to provide useful diagnostics, and provide recovery strategies
  - Uses a static memory management approach

TinyOS

- Resource Sharing
  - Uses two mechanisms for managing shared resources: Virtualization and Completion Events.
  - A virtualized resource appears as an independent instance. i.e., the application uses it independent of other applications
  - Resources that cannot be virtualized are handled through completion events

TinyOS

- Support for real time applications
  - Does not provide support for real-time applications.
  - As mentioned, scheduling is based on FIFO manner
  - TinyOS is not a good choice for mission critical, real time applications

Summary : Suitability of TinyOS for Smart Objects

- Uses Component Model
  - Different components are glued together with the scheduler to compose a static image
  - Results in small memory footprint
- Programming Model
  - Provides the ease of a threading programming model coupled with the efficiency of an event driven kernel
- Efficient Scheduling
  - Supports Two-level scheduling (Event and Task)
- Memory Management
  - Uses static memory approach resulting in small footprint
- Resource sharing
  - Supports Virtualization and Completion Events

Comparison with Contiki

- Table below compares TinyOS with another popular OS Contiki

Feature	TinyOS	Contiki
Architecture	Monolithic architecture; Components are glued together to form a static image	Modular architecture
Programming Model	Supports multithreading & event based programming	Supports multithreading & event driven programming
Scheduling	Supports FIFO scheduling	Event driven OS; does not have sophistated scheduing algorithm
Memory Mgt	Incorporates memory safety; Uses static memory management	Support dynamic memory management
Resource sharing	Use Virtualisation & Completion Events	Provides serialised access to all resources
Real time support	Does not support real time applications	Does not support real time applications

Other Operating Systems for Smart Objects

- Some Smart Objects do not have an OS
- They have start-up code which is run once
- They have loop code which run infinitely.
- Hardware that has no OS
  - Arduino – Open Source hardware
  - Libelium Waspmote
- Advantages
  - Fast
  - Memory efficient

End