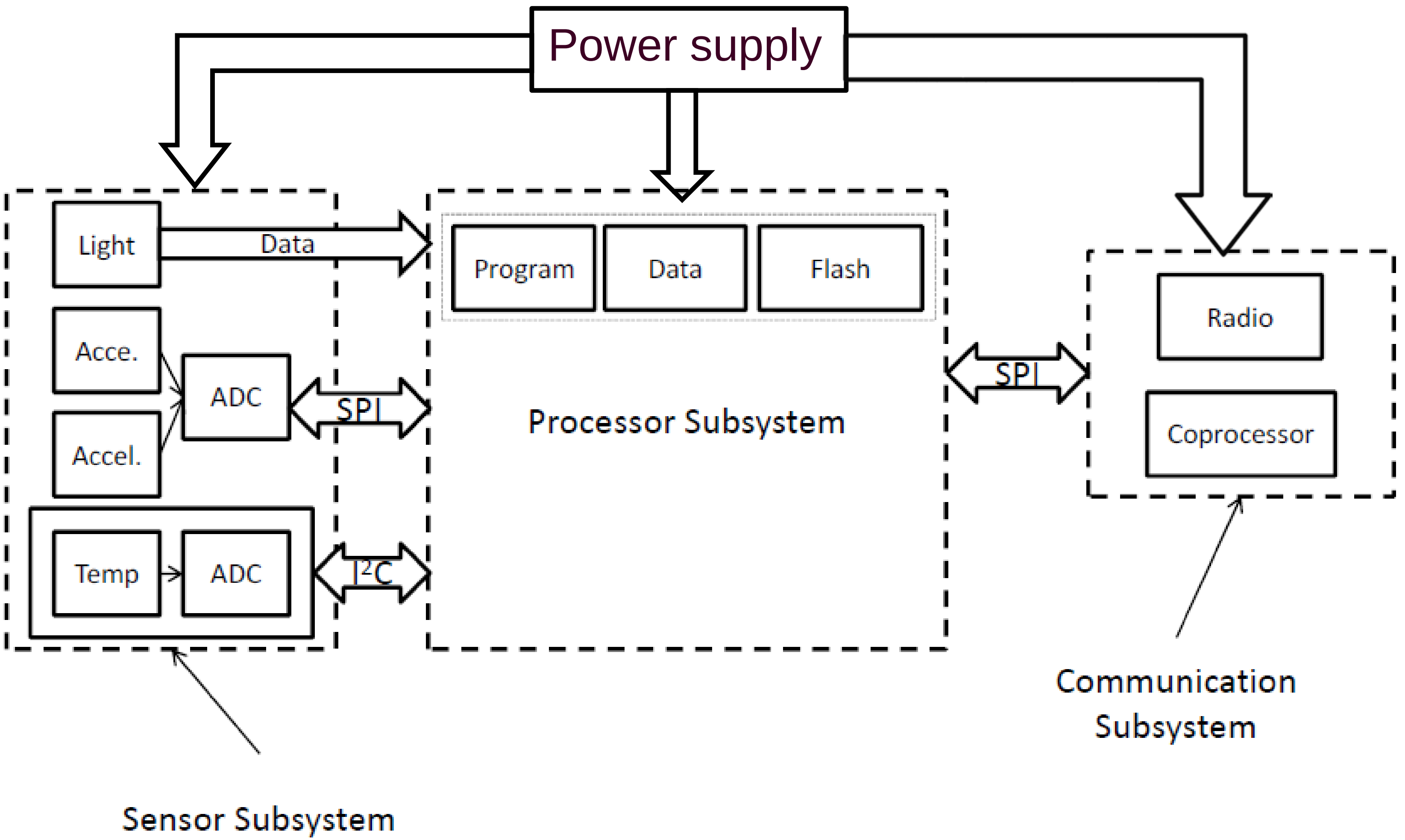
EE5726
Embedded Sensor Networks

Dr. Zhaohui Wang
Assistant Professor
Department of Electrical and Computer Engineering
Michigan Technological University
zhaohuiw@mtu.edu

Node Architecture



Today's Agenda

- Last class: Single-node Architecture
  - Hardware components
- Today: Single-node Architecture
  - Energy consumption
  - Operating system
  - Prototypes
- Next Time: Network Architecture (Chapter 3 of Karl's book)

Energy Consumption of Nodes

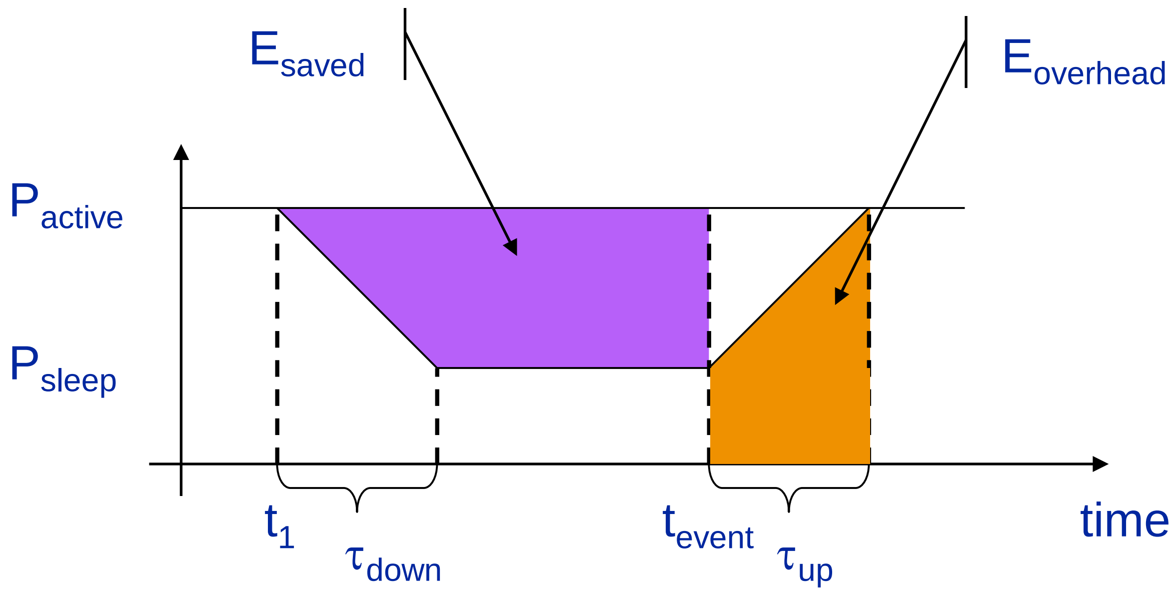
- Main energy consumers
  - *Processor*
  - *Transceiver (energy intensive)*
  - *Memory, to some degree*
  - *Possibly, sensors*
- Energy-efficient design
  - *Low-power chips*
  - *Energy-efficient operation*
    - *Rational: node has nothing to do most of the time*
    - *Adapt operational state to tasks at hand*
    - *Dynamic power management: multiple operational states*

Dynamic Power Management

- Applies to all components
  - *Processor:*
    - *Typical states: active, idle, and sleep*
  - *Transceiver : on or off*
  - *Memory: on or off*
  - *Sensor: on or off*
- When to switch operational states?
- Intuition: switch to lower mode whenever possible
- Fact:
  - *Transition between states takes time and energy!*

Switching Between Modes

- When to switch?
  - *Switching only pays off if $E_{saved} > E_{overhead}$*



Energy Consumption of Nodes

- Main energy consumption
  - *Processor*
  - *Transceiver (energy intensive)*
  - *Memory, to some degree*
  - *Possibly, sensors*

Processor Energy Consumption

- Some figures:

Microcontroller	Normal mode	Idle mode	Sleep mode
Inter StrongARM	400 mW	100 mW	50 μW
TI MSP 430	1.2 mW	---	(4 modes) 0.3 -6 μW
Atmel ATmega	15 mW	6 mW	75 μW

- Dynamic voltage scaling (DVS)
  - *Power consumption in CMOS chips: $P \sim f\, V_{DD}^2$*
  - *The supply voltage can be reduced at low clock rates*
  - *Adapt the controller operational speed to the deadline for power saving*
  - *Cost: execution takes longer*

Energy Consumption of Nodes

- Main consumers of energy consumption
  - *Processor*
  - *Transceiver (energy intensive)*
  - *Memory, to some degree*
  - *Possibly, sensors*

Radio Transceiver Energy Consumption

- Energy consumption during transmission
  - *Start-up power to turn on the transceiver: P_{start}*
  - *Start-up time to turn on the transceiver: T_{start}*
  - *Baseband signal processor and associated circuts: P_{txElec}*
  - *RF signal generation:*
    - P_{amp}: *amplifier power consumption*

      $P_{amp} = \alpha_{amp}+\beta_{amp}P_{tx}$

      P_{tx}: *transmission power*

      α_{amp} *and* β_{amp} *are constants*
    - *Efficiency of power amplifier*

      $\eta_{PA} = P_{tx}/P_{amp}$
    - *For* α_{amp}= 174 mW, β_{amp}= 5, P_{tx} = 1 mw, η_{PA} = 0.55%

Radio Transceiver Energy Consumption

- Consider
  - *a n-bits long packet*
  - *Coding rate R_{code}*
  - *Bit rate R*
- Energy consumption

$$E_{tx}(n, R_{code}, P_{amp}) = T_{start}P_{start} + \frac{n}{RR_{code}}\left(P_{txElec} + P_{amp}\right)$$

Radio Transceiver Energy Consumption

- Energy consumption during reception
  - *Start-up power to turn on the transceiver: P_{start}*
  - *Start-up time to turn on the transceiver: T_{start}*
  - *Receiver circuts: P_{rxElec}*
  - *Decoding overhead per bit: E_{decBit}*
- Consider
  - *a n-bits long packet*
  - *Coding rate R_{code}*
  - *Bit rate R*
- Energy consumption

$$E_{rcvd}(n, R_{code}) = T_{start} P_{start} + \frac{n}{R R_{code}} P_{rxElec} + n E_{decBit}$$

Radio Transceiver Energy Consumption

- Some figures of transceiver energy consumption

Symbol	Description	Example transceiver		
		μAMPS-1 [559]	WINS [670]	MEDUSA-II [670]
α_{amp}	Eq. (2.4)	174 mW	N/A	N/A
β_{amp}	Eq. (2.4)	5.0	8.9	7.43
P_{amp}	Amplifier pwr.	179 – 674 mW	N/A	N/A
P_{rxElec}	Reception pwr.	279 mW	368.3 mW	12.48 mW
P_{rxIdle}	Receive idle	N/A	344.2 mW	12.34 mW
P_{start}	Startup pwr.	58.7 mW	N/A	N/A
P_{txElec}	Transmit pwr.	151 mW	$\approx$ 386 mW	11.61 mW
R	Transmission rate	1 Mbps	100 kbps	OOK 30 kbps ASK 115.2 kbps
T_{start}	Startup time	466 μs	N/A	N/A

Relation between Computation and Communication

- Communication is more energy-expensive
  - *Energy ratio of "sending one bit" vs. "computing one instruction": Anything between 220 and 2900 in the literature*
  - *To communicate (send & receive) one kilobyte = computing three million instructions!*
- Perform computation whenever possible (guideline for WSN design)
  - *In-network processing*
  - *Aggregation*
  - *Advanced transceiver algorithm design, to decrease bit error rate*

Energy Consumption of Nodes

- Main consumers of energy consumption
  - *Processor*
  - *Transceiver (energy intensive)*
  - *Memory, to some degree*
  - *Possibly, sensors*

Memory Energy Consumption

- On-chip memory:
  - *Power consumption is included in the processor*
- FLASH memory:
  - *Energy for reading is similar for types of FLASH memories*
  - *Erasing/writing is expensive*
  - *Mica nodes:*
    - *Reading: 1.11 nAh per byte*
    - *Writing: 83.33 nAh per byte*
- Writing to FLASH memory should be avoided if possible!

Energy Consumption of Nodes

- Main consumers of energy consumption
  - *Processor*
  - *Transceiver (energy intensive)*
  - *Memory, to some degree*
  - *Possibly, sensors*

Energy Consumption of Sensors

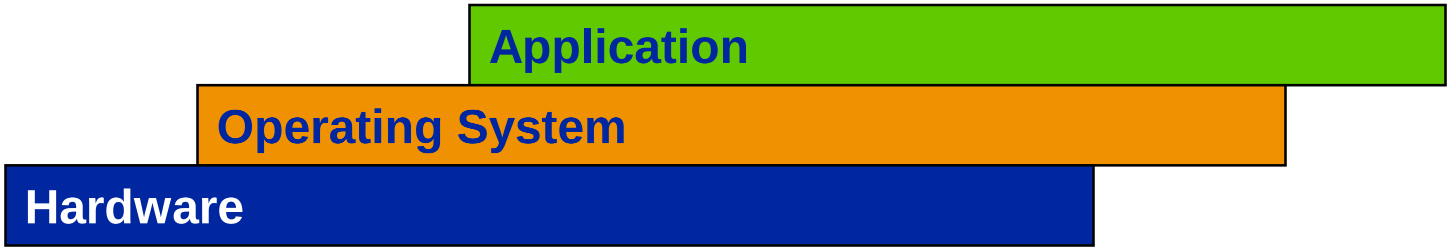
- Application dependent
  - *Passive sensors: negligible (e.g. light or temperature sensors)*
  - *Active sensors: could be considerable (e.g. sonar)*
- Sampling rate and resolution of AD converters affect the power consumption

Today’s Agenda

- Last Week: Single-node Architecture
  - Hardware components
- Today: Single-node Architecture
  - Energy consumption
  - Operating system
  - Prototypes
- Next Time: Chapter 3 of Karl’s book

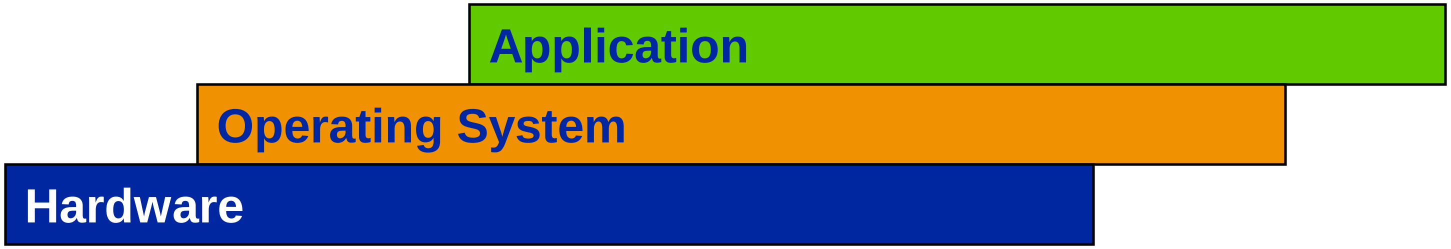
Operating Systems

- An operating system:
  - *A thin software layer*
  - *Resides between the hardware and the application layer*
  - *Provides basic programming abstractions to application developers*



Operating Systems

- Main tasks:
  - *Enable applications to interact with hardware resources*
  - *Schedule and prioritize tasks*
  - *Arbitrate between contending applications and services*



Operating Systems

- Additional features:
  - Memory management
  - Power management
  - File management
  - Networking
  - A set of programming environment and tools
  - Entry points to the operating system for resource access

Operating Systems in WSNs

- Functional aspects:
  - *Data types*
  - *Scheduling*
  - *Stacks*
  - *System calls*
  - *Handling interrupts*
  - *Memory allocation*
- Nonfunctional aspects:
  - *Separation of concern*
  - *System overhead*
  - *Portability*
  - *Dynamic reprogramming*
- Concurrent programming
- Structure of operating system and protocol stack

Data Types

- Interactions between different subsystems take place through:
  - *Well-formulated protocols*
  - *Data types*
- Simple data types are resource efficient but have limited expression capability --- C programming language
- Complex data types have strong expression power but consume resources --- struct and enum

Scheduling

- Task scheduling is one of the basic functions of an OS
- Two scheduling mechanisms:
  - *Queuing-based scheduling*
    - *FIFO ---- the simplest and minimum system overhead but treat tasks unfairly ---- long-duration tasks may block short-duration tasks*
    - *Sorted queue ---- e.g. the shortest job first (SJF) ---- incurs system overhead (to estimate execution duration)*
  - *Round-robin scheduling ---- time sharing scheduling technique ---- several tasks can be processed concurrently*
    - *Defines a time frame by dividing time into slots, and tasks will be given slots in a multiplexed manner*

Scheduling

- Regardless of how tasks are executed, a scheduler can be either
  - *A non-preemptive scheduler ---- a task is executed to the end, may not be interrupted by another task*
  - *or preemptive scheduler ---- a task of higher priority may interrupt a task of low priority*

System Calls & Stacks

- *System Calls*
  - *Decouple the concern of accessing hardware resources*
  - *Whenever users wish to access a hardware resource (e.g., sensor, watchdog timer), they invoke these operations without the need to concern themselves how the hardware is accessed*

- *Stacks*
  - *It is a data structure*
  - *Temporarily store data objects in a memory by piling one upon another*
  - *The objects are accessed on last-in-first-out (LIFO) basis*
  - *The processor core uses stacks to store system state information when it begins executing subroutines*

Handling Interrupts

- An interrupt is an asynchronous signal generated by
  - *A hardware device, e.g. a sensor, a watchdog timer, or the communication subsystem*
  - *OS itself, e.g., periodic interrupts to monitor the state of the hardware resources*
- An interrupt causes:
  - *The processor to interrupt executing the present instruction*
  - *To call for an appropriate interrupt handler*
- Interrupt signals can have different priority levels, a high priority interrupt can interrupt a low level interrupt

Memory Allocation

- The memory unit is a precious resource
  - *Where the OS resides*
  - *Stores the data and application's program code*
- How and for how long a memory is allocated for a piece of program determines the speed of task execution

Memory Allocation

- A memory can be allocated to a program:
  - *Statically ---- a frugal way but the requirement of memory must be known in advance*
    - *Memory is used efficiently*
    - *Runtime adaptation is not allowed*
  - *Dynamically ---- the requirement of memory is not known in advance (on a transient basis)*
    - *Enables flexibility in programming*
    - *But produces a considerable management overhead*

Operating Systems in WSNs

- Functional aspects:
  - *Data types*
  - *Scheduling*
  - *Stacks*
  - *System calls*
  - *Handling interrupts*
  - *Memory allocation*
- Nonfunctional aspects:
  - *Separation of concern*
  - *System overhead*
  - *Portability*
  - *Dynamic reprogramming*
- Concurrent programming
- Structure of operating system and protocol stack

Separation of Concern

- In general OS, there is a clear separation between the operating system and the applications layer
  - *Well-defined interfaces and system calls*
  - *Such a distinction is difficult to support in WSNs*
- The Operation System in WSNs consists of:
  - *A number of lightweight modules ---- "wired" together, or*
  - *An indivisible system kernel + a set of library components for building an application, or*
  - *A kernel + a set of reconfigurable low-level services which can be wired together to make up an application*
- Separation of concern:
  - *An update or an upgrade can be made as a whole or in part as required*
  - *Enables flexible and efficient reprogramming and reconfiguration*

System Overhead

- An operating system executes program code ---- requires its own share of resources
- The resources consumed by the OS are the system’s overhead, it depends on
  - *The size of the operating system*
  - *The type of services that OS provides to the higher-level services and applications*

System Overhead

- The resources of wireless sensor nodes have to be shared by programs that carry out:
  - *Sensing*
  - *Data aggregation*
  - *Self-organization*
  - *Network management*
  - *Network communication*

Portability

- The hardware architecture of WSNs is undergoing rapid evolution
- Different hardware architectures exist
- In order to accommodate unforeseen requirements, operating systems should be portable and extensible

Dynamic Reprogramming

- Once a wireless sensor network is deployed, it may be necessary to reprogram some part of the application or the operating system for the following reasons:
  - *The network may not perform optimally*
  - *Both the application requirements and the network's operating surrounding can change over time*
  - *Be necessary to detect and fix bugs*

Dynamic Reprogramming

- Manual replacement may not be feasible ---- develop an operating system that provides dynamic reprogramming support
  - *Needs to be supported with a clear separation between the application and the OS, in principle*
  - *In practice:*
    - *The OS can receive the software update and assemble and store it in memory*
    - *OS should make sure that this is indeed an updated version*
    - *OS can remove the piece of software that should be updated and install and configure the new version*
  - *All these consume resources and may cause their own bugs*
  - *Software reprogramming (update) requires robust code dissemination protocols, to ensure code consistency and version controlling*

Operating Systems in WSNs

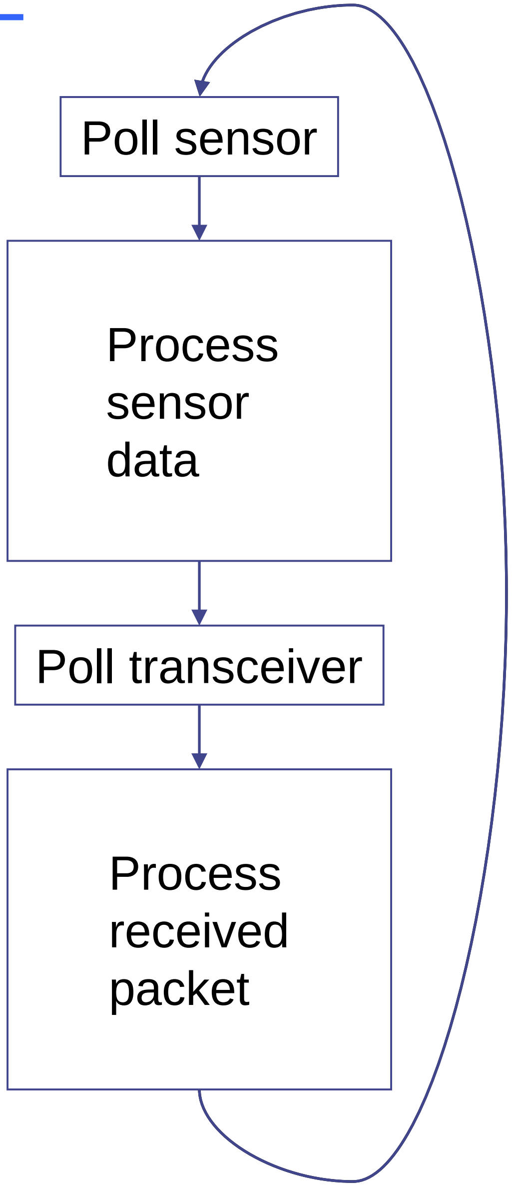
- Functional aspects:
  - *Data types*
  - *Scheduling*
  - *Stacks*
  - *System calls*
  - *Handling interrupts*
  - *Memory allocation*
- Nonfunctional aspects:
  - *Separation of concern*
  - *System overhead*
  - *Portability*
  - *Dynamic reprogramming*
- Concurrent programming
- Structure of operating system and protocol stack

Concurrent Programming

- Why concurrency is needed?
  - *Nodes have to handle data from arbitrary sources at arbitrary points in time, e.g.,*
    - *data from multiple sensors*
    - *data from transceivers*
  - *execute communication protocols*
  - *perform data computation*
  - *…*

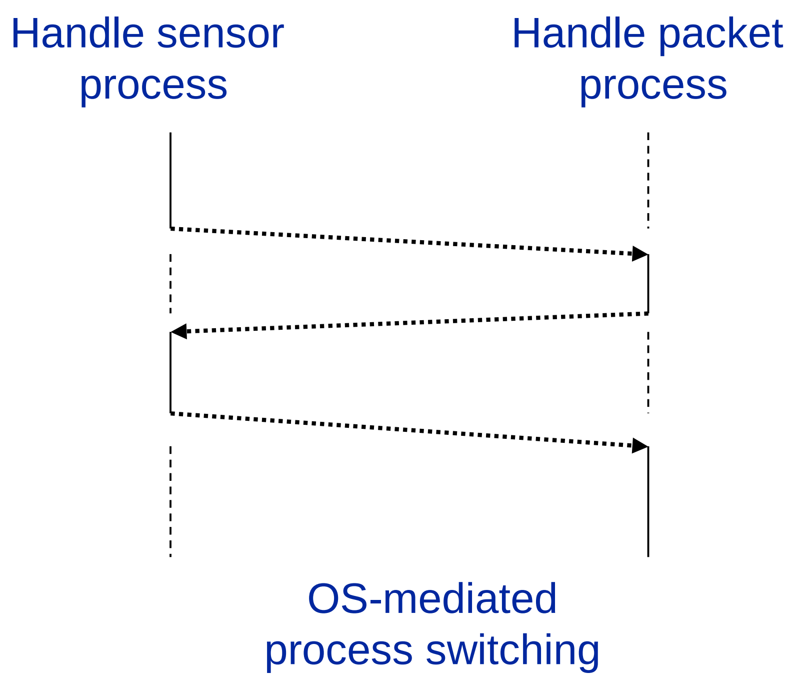
Concurrent Programming

- Simplest option: no concurrency, sequential processing of tasks
  - *Not satisfactory: Risk of missing packet (e.g., from transceiver) when processing data, etc.*



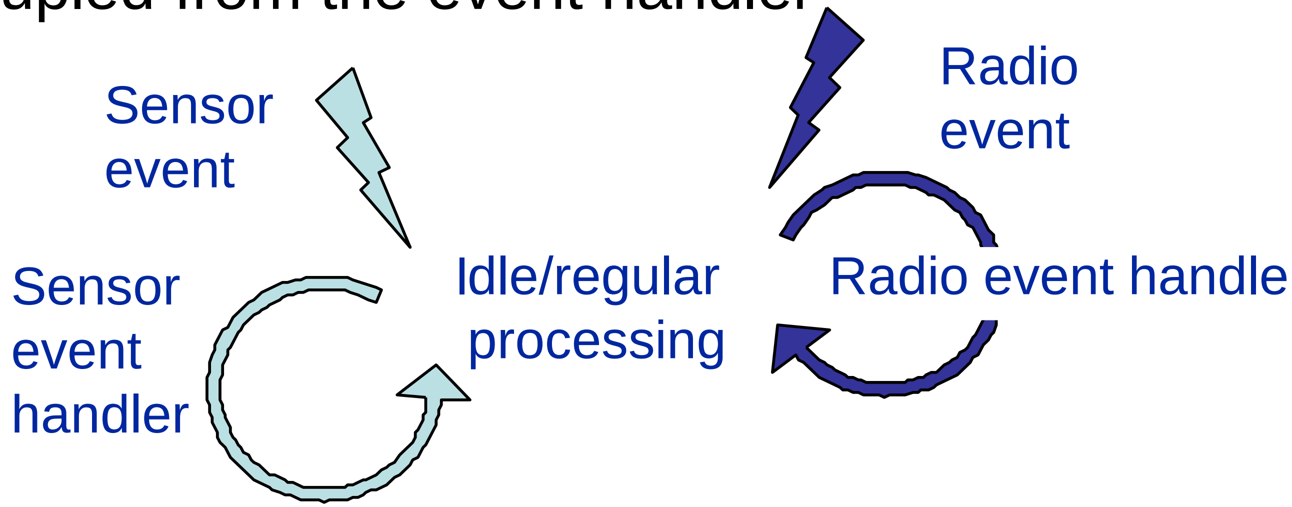
Process-based Concurrency

- Process-based concurrency?
  - *Concurrent execution of multiple processes on a single-CPU*
  - *Each process requires its own stack space in memory*
  - *Large overhead incurred for switching between tasks*



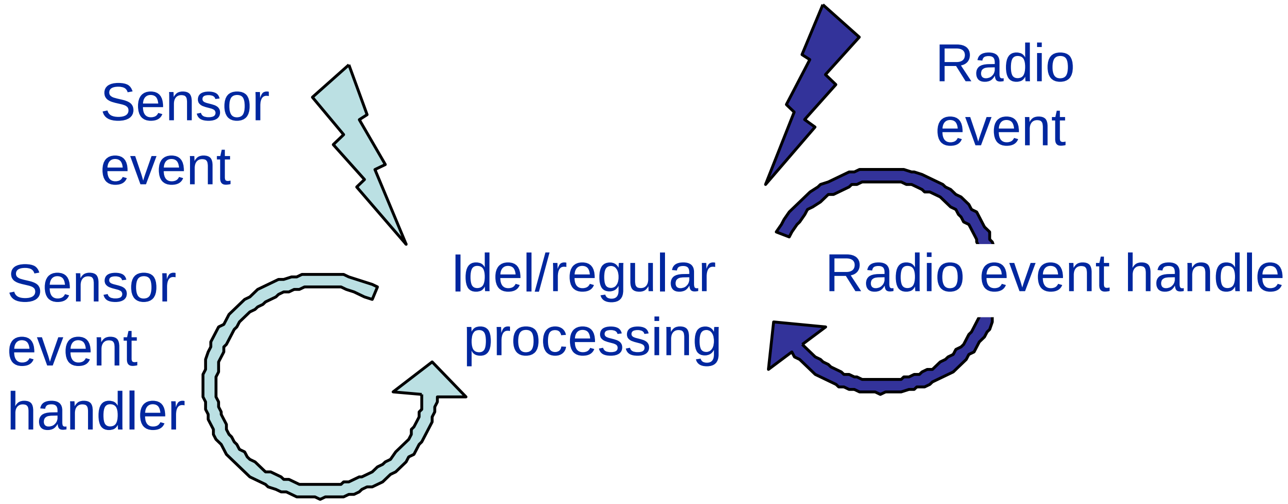
Event-based Programming

- Reactive nature of WSN nodes
  - *OS waits for an event, e.g.*
    - *The availability of data from a sensor*
    - *The arrival of a packet*
    - *The expiration of a timer*
- Event handler: a few instructions to store the event info.
- The information is processed by separate routines, decoupled from the event handler



Event-based Programming

- Event handler
  - *very simple and short*
  - *can interrupt the processing of normal codes*
  - *Run to completion without noticeably disturbing other code*
- Event handlers cannot interrupt each other, run sequentially



Event-based Programming

- Two types of “contexts”
  - *Event handler: time-critical*
  - *Processing of normal code: triggered by an event handler*

- Event-based programming model
  - *Different from traditional programming model*
  - *Comparable to the finite state machine, or VHDL*

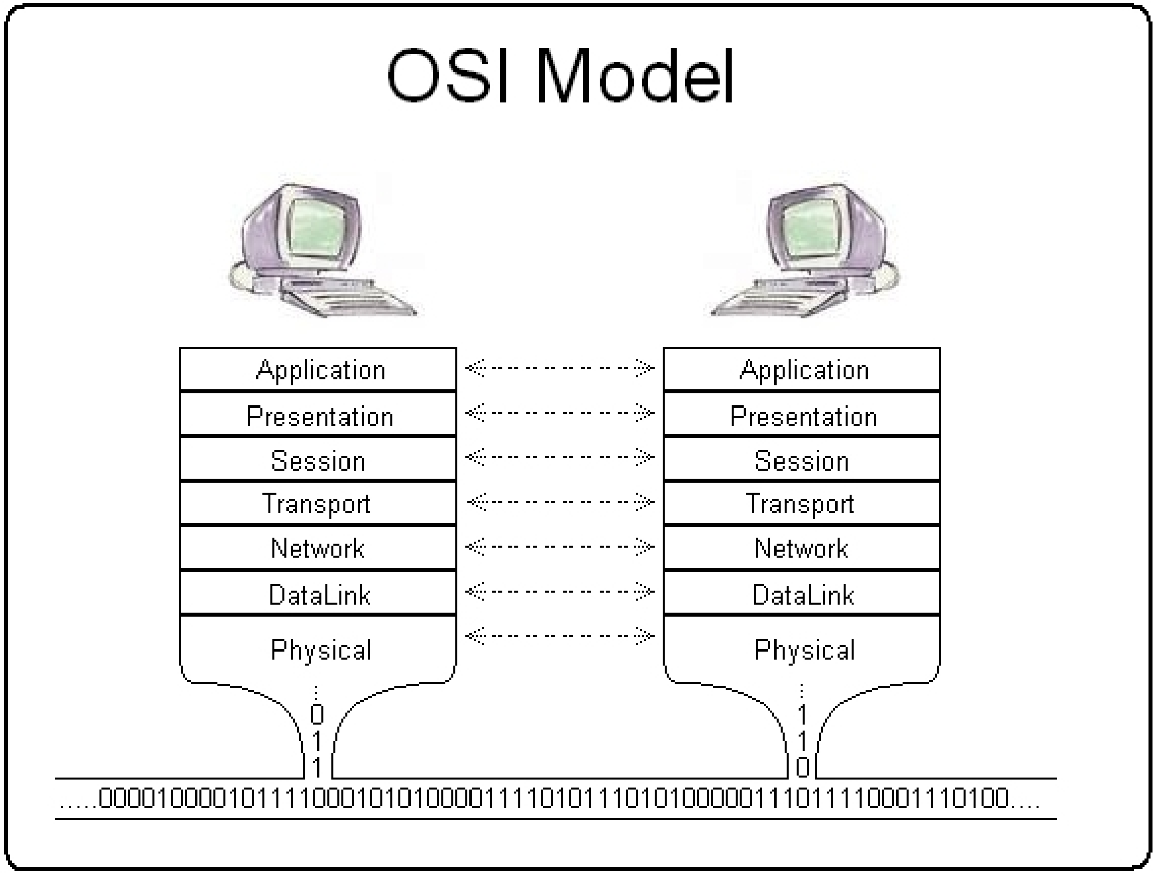
- Advantages over process-based programming model:
  - *Performance improved by a factor of 8*
  - *Instruction/data memory requirement reduced by factors of 2 and 30, respectively*
  - *Power consumption reduced by a factor of 12*

Operating Systems in WSNs

- Functional aspects:
  - *Data types*
  - *Scheduling*
  - *Stacks*
  - *System calls*
  - *Handling interrupts*
  - *Memory allocation*
- Nonfunctional aspects:
  - *Separation of concern*
  - *System overhead*
  - *Portability*
  - *Dynamic reprogramming*
- Concurrent programming
- Structure of operating system and protocol stack

Structure of Communication Protocols

- Computer networks: layered protocol stack
  - One layer only services its direct upper layer
  - Advantages: easy to manage, promoting modularity/reuse
  - Inflexibility in cross-layer information exchange
    - e.g., Signal strength information might be useful for routing protocols



Structure of OS and Protocol Stack

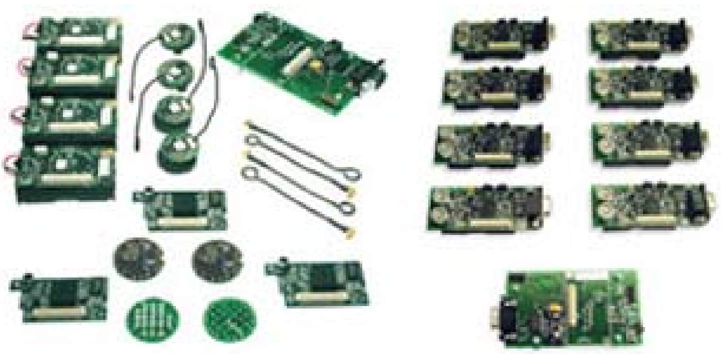
- Component-based model
  - The monolithic layers are broken up into small, self-contained "components"
  - Each component only fulfills one well-defined function
  - Wrapping of hardware, communication protocols, in-networking processing functionalities can all be designed as components

Component-based Protocol Stack

- Component-based protocol stack
  - *Components can be “wired” together*
  - *Components’ interactions are not confined to immediate neighbors, but can be with any other component - Cross-layer optimization*
  - *Fits well with the event-based programming*
- A collection of components:
  - Physical-layer protocols
  - MAC protocols
  - Link-layer protocols
  - Routing protocols
  - Transport layer protocols
  - Time synchronization
  - Localization
  - Topology control

TinyOS & NesC

- TinyOS developed by UC Berkely as runtime environment for their “Mica motes”
- NesC as adjunct “programming language” (event-based)
- Most important design aspects
  - *Component-based system*
  - *Components interact by exchanging asynchronous events*
  - *Components form a program by wiring them together (akin to VHDL – hardware description language)*



Comparison of Existing Operating Systems

OS	Programming Paradigm	Building Blocks	Scheduling	Memory Allocation	System Calls
TinyOS	Event-based (split-phase operation, active messages)	Components, interfaces and tasks	FIFO	Static	Not available
SOS	Event-based (Active messages)	Modules and messages	FIFO	Dynamic	Not available
Contiki	Predominantly event-based, but it provides an optional multi-threading support	Services, service interface stubs and service layer	FIFO, poll handlers with priority scheduling	Dynamic	Runtime libraries
LiteOS	Thread-based (based on thread pool)	Applications are independent entities	Priority-based scheduling with an optional Round-robin support	Dynamic	A host of system calls available to the user (file, process, environment, debugging and device commands)

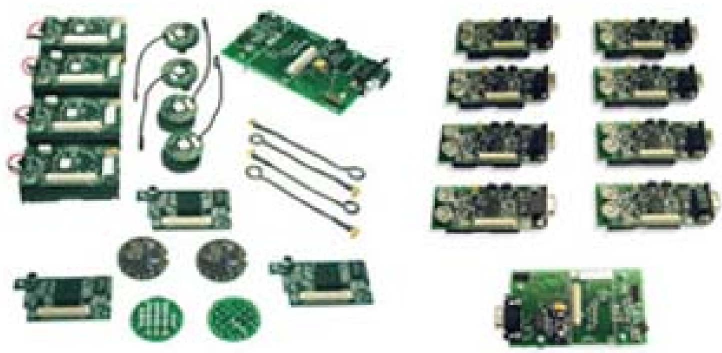
Comparison of Existing Operating Systems

OS	Minimum System overhead	Separation of Concern	Dynamic reprogramming	Portability
TinyOS	332 Bytes	There is no clean distinction between the OS and the application. At compilation time a particular configuration produces a monolithic, executable code.	Requires external software support	High
SOS	ca. 1163 Byte	Replaceable modules are compiled to produce an executable code. There is no clean distinction between the OS and the application.	Supported	Midium to low
Contiki	ca. 810 Byte	Modules are compiled to produce a reprogrammable and executable code, but there is no separation of concern between the application and the OS.	Supported	Medium
LiteOS	Not available	Application are separate entities; they are developed independent of the OS	Supported	Low

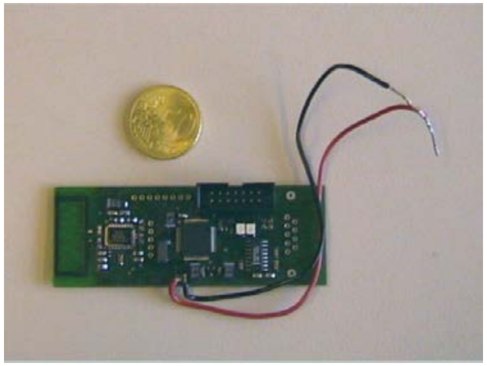
Examples of Sensor Nodes

- http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes

- Mica motes:
  - *Developer: UC Berkeley & Intel*
  - *Versions: Mica, Mica2, MicaDot*
  - *TinyOS*
  - *Microcontroller: Altmel*
  - *Interface to additional sensor boards*



- EYES nodes
  - *Developer: Infineon*
  - *Microcontroller: TI*
  - *USB interface*
  - *PeerOS*



Examples of Sensor Nodes

- BTnodes
  - *Developer: ETH Zürich*
  - *Microcontroller: Atmel Atmega 128L*
  - *Bluetooth with Chipcon CC1000*
  - *BTnut and TinyOS*

- Scatterweb
  - *Developer: Freie University Berlin*
  - *An family of nodes: from standard sensor nodes to embedded web server*
  - *With a wide range of interconnection possibilities*

Summary

- Last Week: Single-Node Architecture
  - Hardware components
- Today: Single-Node Architecture
  - Energy consumption
  - Operating system
  - Prototypes
- Next Time: Network Architecture (Chapter 3 in Karl's book)