



# CS 425

## Distributed Systems

### Mobile and Ubiquitous Computing

Lecture 25

Klara Nahrstedt

# Reading List

- Reading: Section 16.1-2, 16.4.1
- <http://robotics.eecs.berkeley.edu/~pister/SmartDust/>
- <http://docs.tinyos.net/>

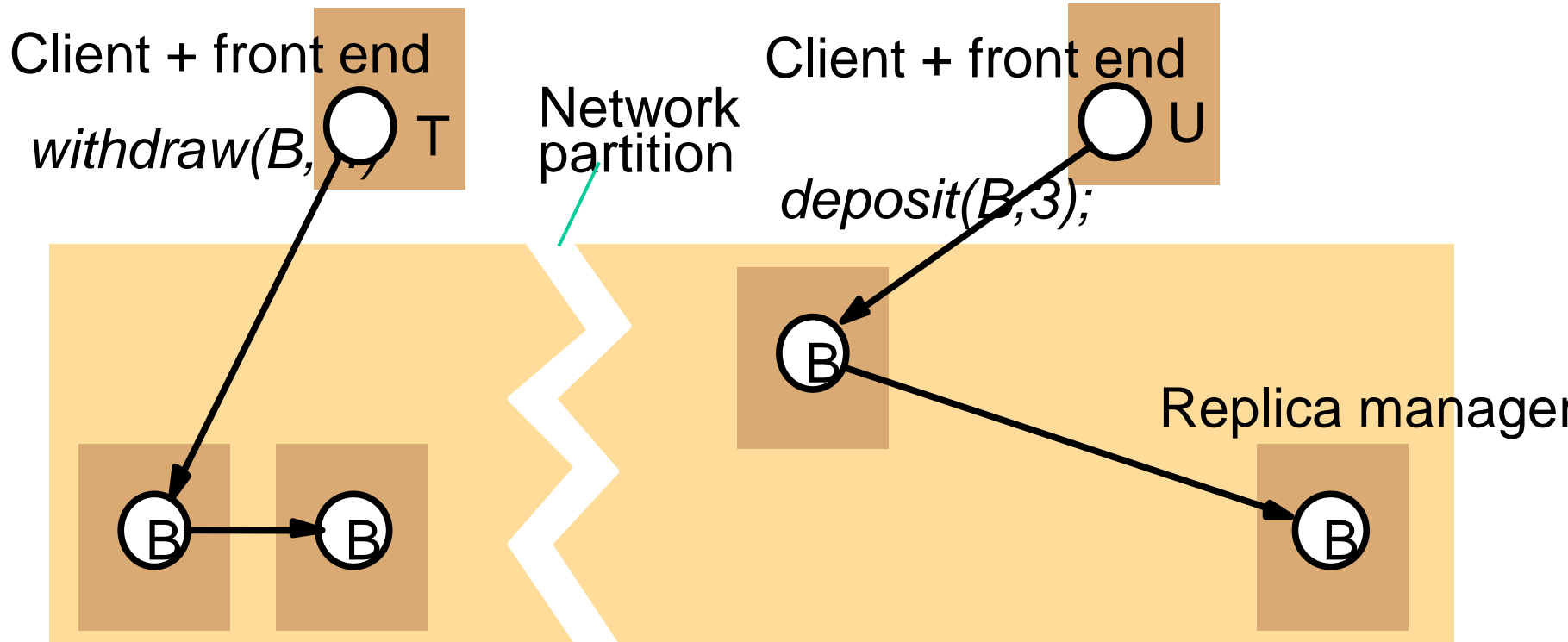
# Administrative

- **MP3 posted**
  - Deadline December 7 (Monday) – pre-competition
    - Top five groups will be selected for final demonstration on Tuesday, December 8
  - Demonstration Signup Sheets for Monday, 12/7, will be made available
  - Main Demonstration in front of the Qualcomm Representative will be on Tuesday, December 8 afternoon - details will be announced.
- **HW4 posted**
  - Deadline December 1, 2009 (Tuesday)

# Plan for Today

- Replications under failures
  - Network partition
    - Quorum-based approaches
- Discovery Service
- Sensing and Context-Awareness
- Sensors
  - Hardware, OS, Protocols, Applications

# Network Partition



# Dealing with Network Partitions

- During a partition, pairs of conflicting transactions may have been allowed to execute in different partitions. The only choice is to take corrective action after the network has recovered
  - Assumption: Partitions heal eventually
- Abort one of the transactions after the partition has healed
- **Basic idea:** allow operations to continue in partitions, but **finalize and commit trans. only after** partitions have healed
- But need to avoid executing operations that will eventually lead to aborts...

# Quorum Approaches for Network Partitions

- Quorum approaches used to decide whether reads and writes are allowed
- In the *pessimistic quorum philosophy*, updates are **allowed only in a partition** that has the majority of RMs
  - Updates are then propagated to the other RMs when the partition is repaired.

# Static Quorums

- ❖ The decision about how many RMs should be involved in an operation on replicated data is called *Quorum selection*
- ❖ Quorum rules state that:
  - ❖ At least  $\underline{r}$  replicas must be accessed for read
  - ❖ At least  $\underline{w}$  replicas must be accessed for write
  - ❖  $\underline{r} + \underline{w} > N$ , where  $N$  is the number of replicas
  - ❖  $\underline{w} > N/2$
  - ❖ Each object has a **version number** or a **consistent timestamp**
- ❖ Static Quorum predefines  $\underline{r}$  and  $\underline{w}$ , & is a **pessimistic approach**: if partition occurs, update will be possible in at most one partition

# Voting with Static Quorums

- ❖ A version of quorum selection where each **replica has a number of votes**. Quorum is reached by majority of votes (N is the total number of votes)  
e.g., a cache replica may be given a 0 vote

Replica	votes	access time	version chk	P(failure)
Cache	0	100ms	0ms	0%
Rep1	1	750ms	75ms	1%
Rep2	1	750ms	75ms	1%
Rep3	1	750ms	75ms	1%

- with  $\underline{r} = \underline{w} = 2$ , Access time for write is 750 ms (parallel writes). Access time for read without cache is 750 ms. Access time for read with cache is 175ms to 825ms. (chk – check time)

# Optimistic Quorum Approaches

- ❖ An Optimistic *Quorum selection* allows writes to proceed in any partition.
- ❖ This might lead to **write-write** conflicts. Such conflicts will be detected when the partition heals
  - ❖ Any writes that violate one-copy serializability will then result in the transaction (that contained the write) to abort
  - ❖ Still improves performance because partition repair not needed until commit time
- ❖ **Optimistic Quorum is practical** when:
  - ❖ Conflicting updates **are rare**
  - ❖ Conflicts are **always detectable**
  - ❖ Damage from conflicts can be **easily confined**
  - ❖ **Repair** of damaged data is **possible** or an update can be **discarded without consequences**

# View-based Quorum

- ❖ An optimistic approach
- ❖ Quorum is based on views at any time
- ❖ In a partition, **inaccessible nodes** are considered in the quorum as **ghost participants** that reply “Yes” to all requests.
  - ❖ Allows operations to proceed if the partition is large enough (need not be majority)
- ❖ Once the partition is repaired, participants in the smaller partition know whom to contact for updates.

# View-based Quorum - details

❖ We define **thresholds** for each of read and write :

❖  $A_w$ : minimum nodes in a view for write, e.g.,  $A_w > N/2$

❖  $A_r$ : minimum nodes in a view for read

❖ E.g.,  $A_w + A_r > N$

❖ If ordinary quorum cannot be reached for an operation, then we take a straw poll, i.e., we update views

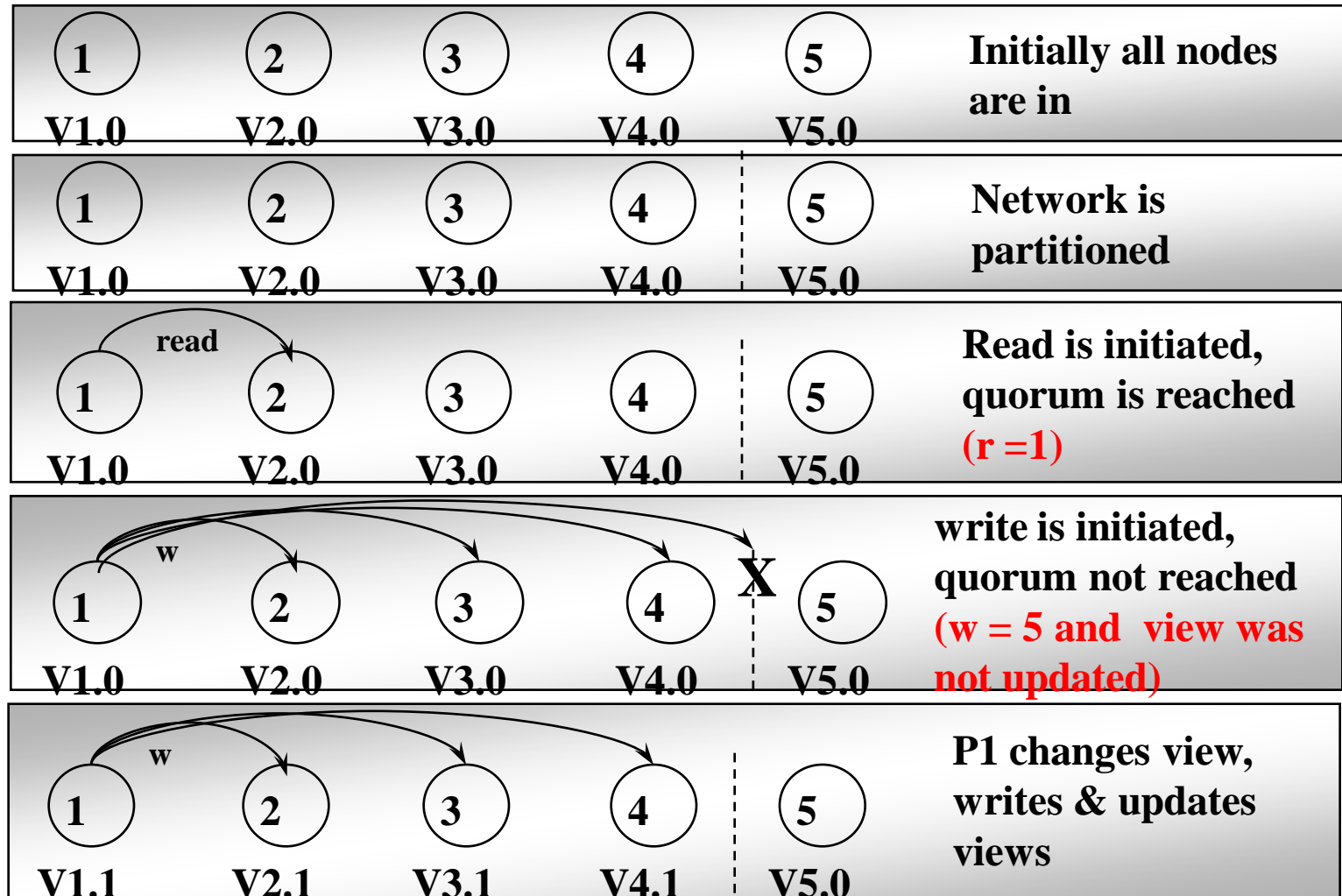
❖ In a large enough partition for read,  $View_{size} \geq A_r$  In a large enough partition for write,  $View_{size} \geq A_w$  (inaccessible nodes are considered as ghosts that reply Yes to all requests.)

❖ Views are per object, numbered sequentially and only updated if necessary

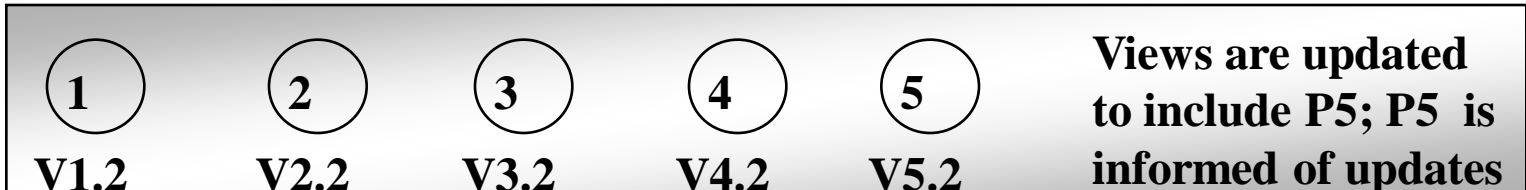
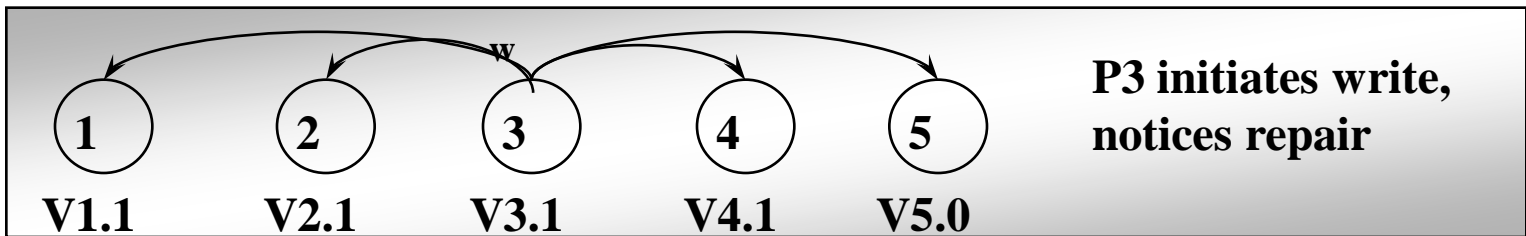
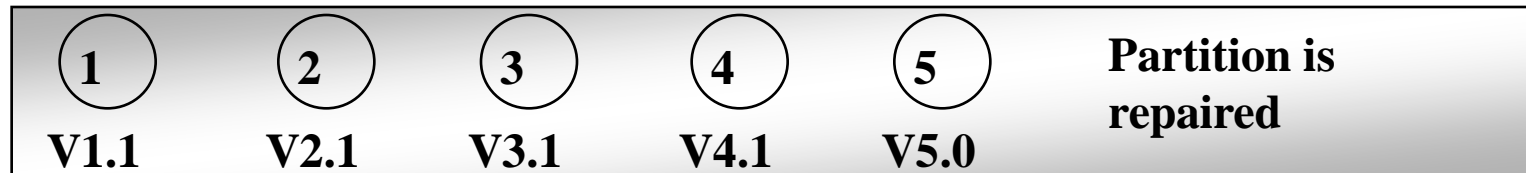
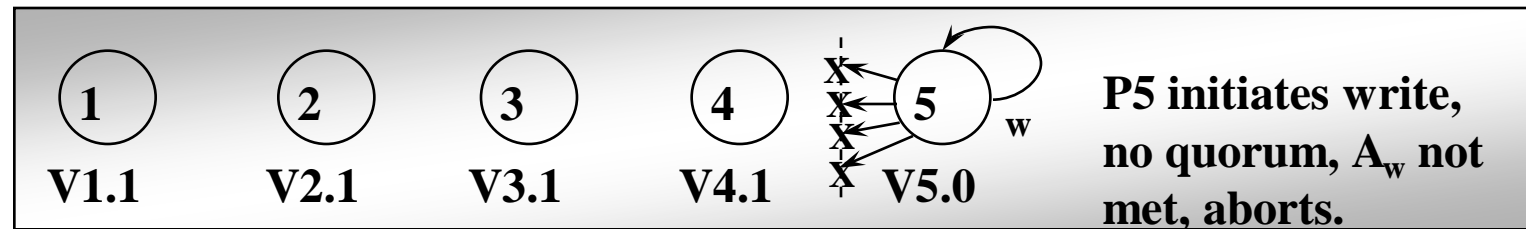
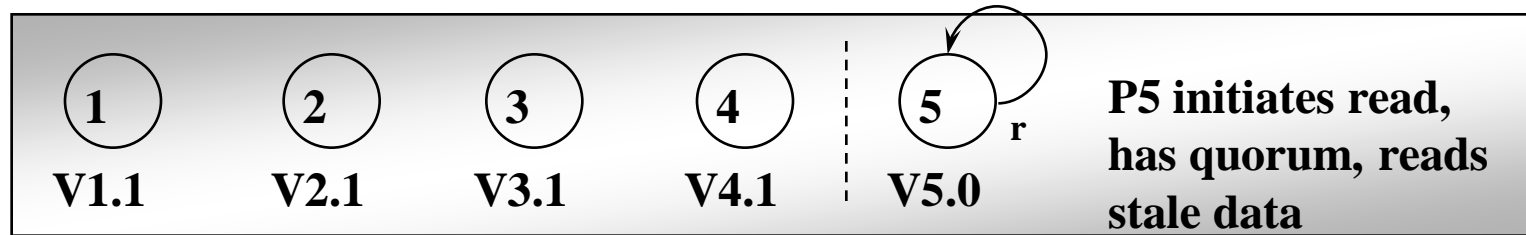
❖ The first update after partition repair forces restoration for nodes in the smaller partition

# Example: View-based Quorum

❖ Consider:  $N = 5$ ,  $w = 5$ ,  $r = 1$ ,  $A_w = 3$ ,  $A_r = 3$



# Example: View-based Quorum (cont'd)



# **MOBILE AND UBIQUITOUS COMPUTING**

# Introduction

- **Mobile Computing** – paradigm in which users carry their personal computers and retain connectivity to other machines
  - Handheld computing (use PDAs, iPhones)
- **Ubiquitous Computing** (pervasive computing) – paradigm where the prevalence of computing devices leads to revolutionary shift how we use computers
  - Mark Weiser – 1988
- **Wearable Computing** – paradigm where users carry wearable computing devices on their person (clothes, watches, rings, ..)

# Characteristics of Mobile and Ubiquitous Systems

- **Volatile Systems**

- Failure of devices and communication links WiFi, Bluetooth, ..
- Changes of characteristics of communication such as bandwidth and latency
- Creation and destruction of associations – logical communication relationships between software components
- Spontaneous interoperation
- Lowered trust and privacy

- **Device Model**

- Limited energy
- Resource constraint
- Sensors and Actuators - Motes - Camera phones

# Gaia Smart Space (Physical Space for Pervasive System at UIUC)



# Discovery Services

- Service for clients to **find out about the services** provided in smart space
- Need
  - **Directory service** in which services in smart space are **registered and looked up** by their attributes
    - Methods for service de/registration
    - Methods for invocation to look up a service (query)
  - Device discovery
  - Service discovery

# Physical Association

- **Human input to scope discovery**
  - Type in or select the smart space's identifier such as room number in the case of hotel guest
- **Sensing and physically constrained channels to scope discovery**
  - Employs sensor on user's device to discover service
    - Identifier-encoding symbol on surfaces in the space
    - Location-based service
- **Direct association**
  - Human directly associates two devices (no discovery)

# **SENSOR NETWORKS AND DISTRIBUTED SYSTEMS**

# Sensing and Context-Awareness

- **Sensors**
  - Sensing of location, temperature, ...
- Processing data collected from sensors
- **Context-aware systems** that respond to sensed physical circumstances
- **Context of entity**
  - Aspect of its physical circumstances of relevance to system behavior



# Sensing

- What is the smallest transistor out there today?
- How would you “**monitor**”:
  - a) a large battlefield (for enemy tanks)?
  - b) a large environmental area (e.g., movement of whales)?
  - c) your own backyard (for intruders)?

# Sensors!

- Coal mines have always had CO/CO<sub>2</sub> sensors
- Industry has used sensors for a long time

Today...

- Excessive Information
  - Environmentalists collecting data on an island
  - Army needs to know about enemy troop deployments
  - Humans in society face information overload
- Sensor Networking technology can help filter and process this information (And then perhaps respond automatically?)

# Sensors

Growth of a technology requires

## **I. Sensor Hardware**

- **Smart Dust, Motes**

## **II. Operating Systems and Protocols**

- **TinyOS**

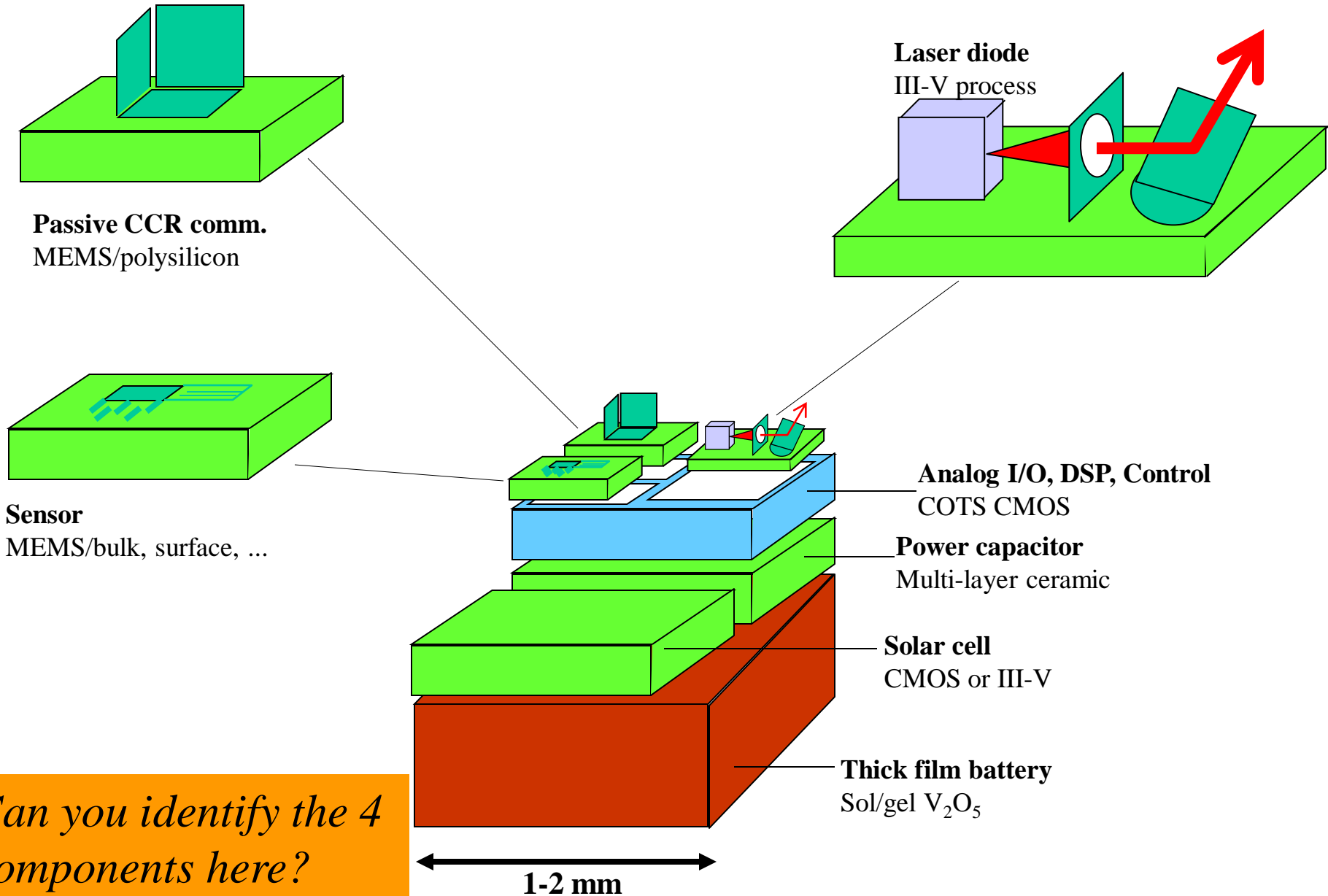
## **III. Killer applications**

- **Military and Civilian**

# Sensor Nodes

- Motivating factors for emergence: applications, Moore's Law, wireless comm., MEMS (“micro electro mechanical systems”)
- Canonical *Sensor Node* contains
  1. Sensor(s) to convert a different energy form to an electrical impulse e.g., to measure temperature
  2. Microprocessor
  3. Communications link e.g., wireless
  4. Power source e.g., battery

# Example: Berkeley “Motes” or “Smart Dust”



*Can you identify the 4 components here?*

# Example Hardware

- Size
  - **Golem Dust**: 11.7 cu. mm
  - **MICA motes**: Few inches
- Everything on one chip: micro-everything
  - processor, transceiver, battery, sensors, memory, bus
  - MICA: 4 MHz, 40 Kbps, 4 KB SRAM / 512 KB Serial Flash, lasts 7 days at full blast on 2 x AA batteries

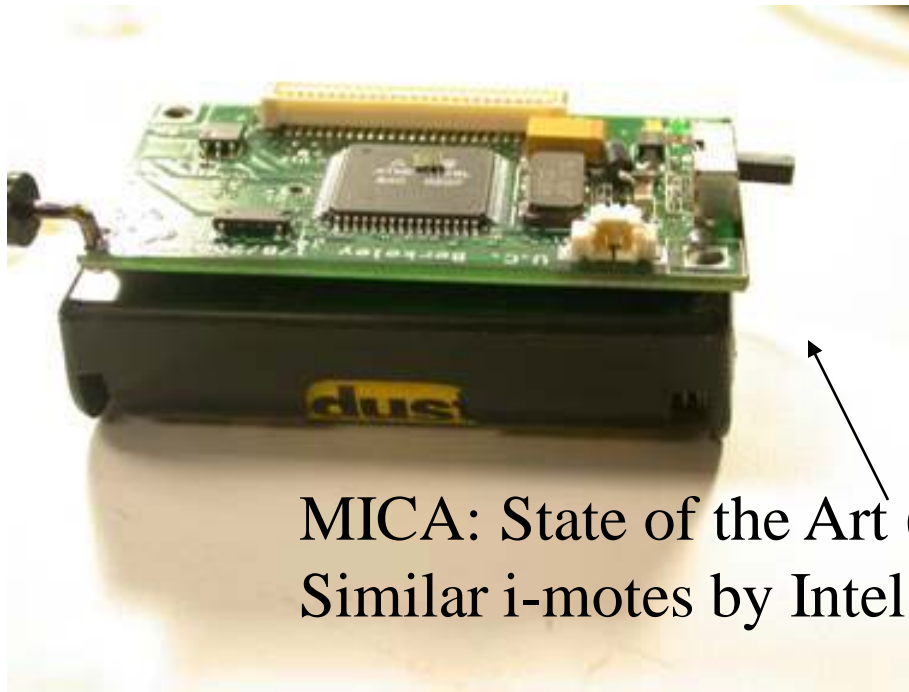


# Examples



## Spec, 3/03

- 4 KB RAM
- 4 MHz clock
- 19.2 Kbps, 40 feet
- Supposedly \$0.30



MICA: State of the Art (xbow)  
Similar i-motes by Intel

# Types of Sensors

- **Micro-sensors** (MEMS, Materials, Circuits)
  - acceleration, vibration, gyroscope, tilt, magnetic, heat, motion, pressure, temp, light, moisture, humidity, barometric, sound
- **Chemical**
  - CO, CO<sub>2</sub>, radon
- **Biological**
  - pathogen detectors
- [Actuators too (mirrors, motors, smart surfaces, micro-robots) ]

# Transmission Medium

- Spec, MICA: **Radio Frequency (RF)**
  - Broadcast medium, routing is “store and forward”, links are bidirectional
- Smart Dust : smaller size => RF needs high frequency => higher power consumption => RF not good
  - Instead, use ***Optical transmission***: simpler hardware, lower power
  - Directional antennas only, broadcast costly
  - Line of sight required
  - However, switching links costly : mechanical antenna movements
  - Passive transmission (reflectors) => wormhole routing
  - Unidirectional links

# Summary: Sensor Node

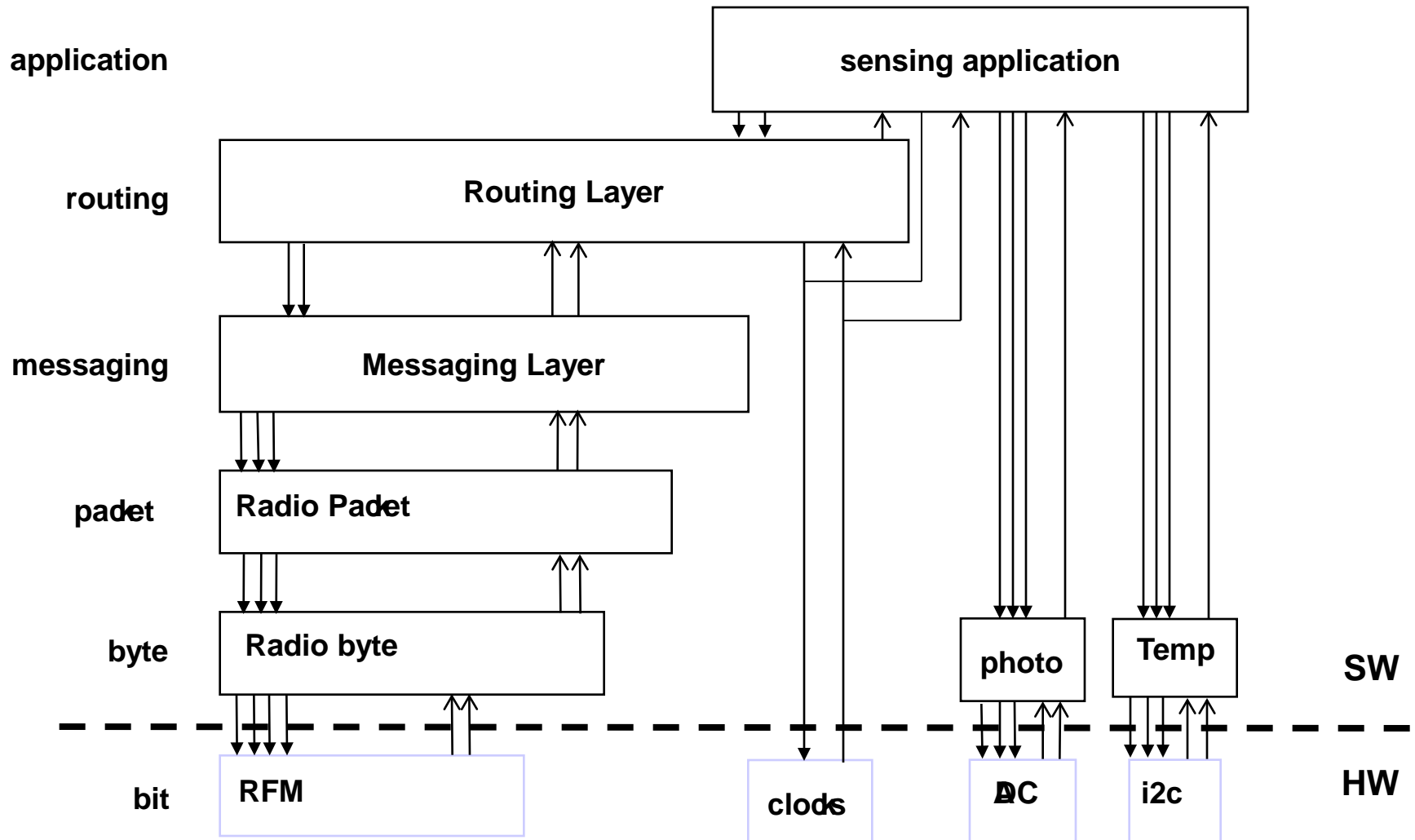
- Small Size : few mm to a few inches
- Limited processing and communication
  - MhZ clock, MB flash, KB RAM, 100's Kbps (wireless) bandwidth
- Limited power (MICA: 7-10 days at full blast)
- Failure prone nodes and links (due to deployment, fab, wireless medium, etc.)
- But easy to manufacture and deploy in large numbers
- Need to offset this with scalable and fault-tolerant OS's and protocols

# Sensor-Node Operating System

## Issues

- **Size of code and run-time memory footprint**
  - Embedded System OS's inapplicable: need hundreds of KB ROM
- **Workload characteristics**
  - Continuous ? Bursty ?
- **Application diversity**
  - Reuse sensor nodes
- **Tasks and processes**
  - Scheduling
  - Hard and soft real-time
- **Power consumption**
- **Communication**

# A Complete TinyOS Application





# TinyOS Facts

- Software Footprint 3.4 KB
- Power Consumption on Rene Platform  
Transmission Cost: 1  $\mu\text{J/bit}$   
Inactive State: 5  $\mu\text{A}$   
Peak Load: 20 mA
- Concurrency support: at peak load CPU is asleep 50% of time
- Events propagate through stack  $<40 \mu\text{S}$



# Energy – a critical resource

- **Power saving modes:**
  - MICA: active, idle, sleep
- **Tremendous variance** in energy supply and demand
  - Sources: batteries, solar, vibration, AC
  - Requirements: long term deployment vs. short term deployment, bandwidth intensiveness
  - 1 year on 2xAA batteries  $\Rightarrow$  200 uA average current

# System Robustness

- **Individual sensor-node OS level:**
  - Small, therefore fewer bugs in code
  - TinyOS: efficient network interfaces and power conservation
  - Importance? Failure of a few sensor nodes can be made up by the distributed protocol
- **Application-level ?**
  - **Need:** Designer to know that sensor-node system is flaky
- **Level of Protocols?**
  - Need for **fault-tolerant protocols**
    - Nodes can fail due to deployment/fab; communication medium lossy
  - e.g., ad-hoc routing to base station (for **data aggregation**):
    - TinyOS's *Spanning Tree Routing*: simple but will partition on failures
    - Better: denser graph (e.g., DAG) - more robust, but more expensive maintenance

# Scalability

- **OS level ?**

TinyOS:

- Modularized and generic interfaces admit a variety of applications
- Correct direction for future technology
  - Growth rates: data > storage > CPU > communication > batteries
- Move functionality from base station into sensor nodes
- In sensor nodes, move functionality from s/w to h/w

- **Application-level ?**

- **Need:** Applications written with scalability in mind
- **Need:** Application-generic scalability strategies/paradigms

- **Level of protocols?**

- **Need:** protocols that scale well with a thousand or a million nodes

# Etcetera

- **Option:** specialized versus generic-sensors
  - Performance vs. applicability vs. money
  - Systems for sets of applications with common characteristics
- **Need:** Self-sufficient sensor networks
  - In-network processing, monitoring, and healing
- **Need:** Scheduling
  - Across networked nodes
  - Mix of real-time tasks and normal tasks
- **Need:** Security, and Privacy
- **Need:** Protocols for **anonymous** sensor nodes
  - E.g., Directed Diffusion protocol for aggregation

# Civilian Mote Deployment Examples

- Environmental Observation and Forecasting (EOFS)

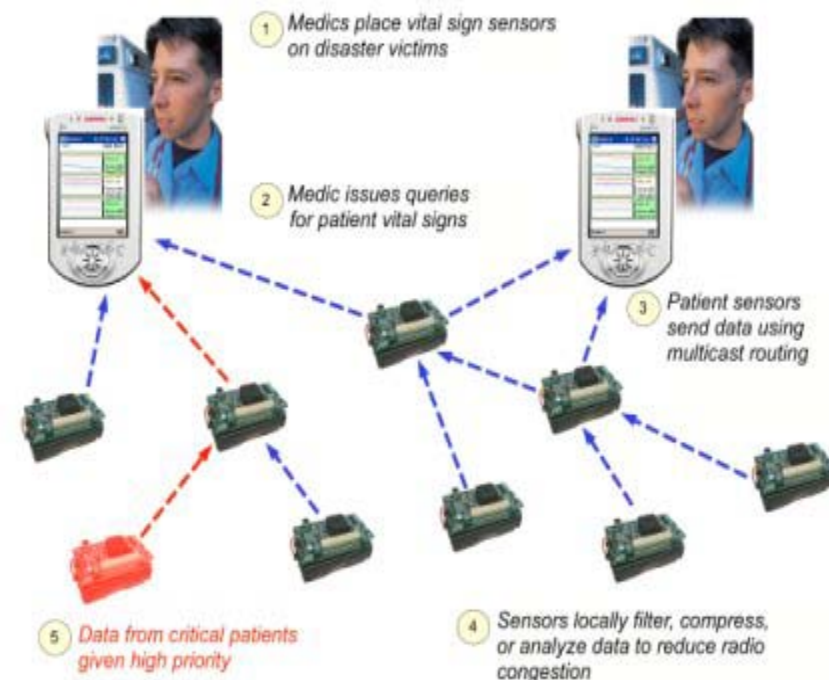
- <http://www.cc.gatech.edu/~calton/publications/mobicom-00.pdf>

- CASA Center for Collaborative Adaptive Sensing of the Atmosphere

- See <http://www.casa.umass.edu>

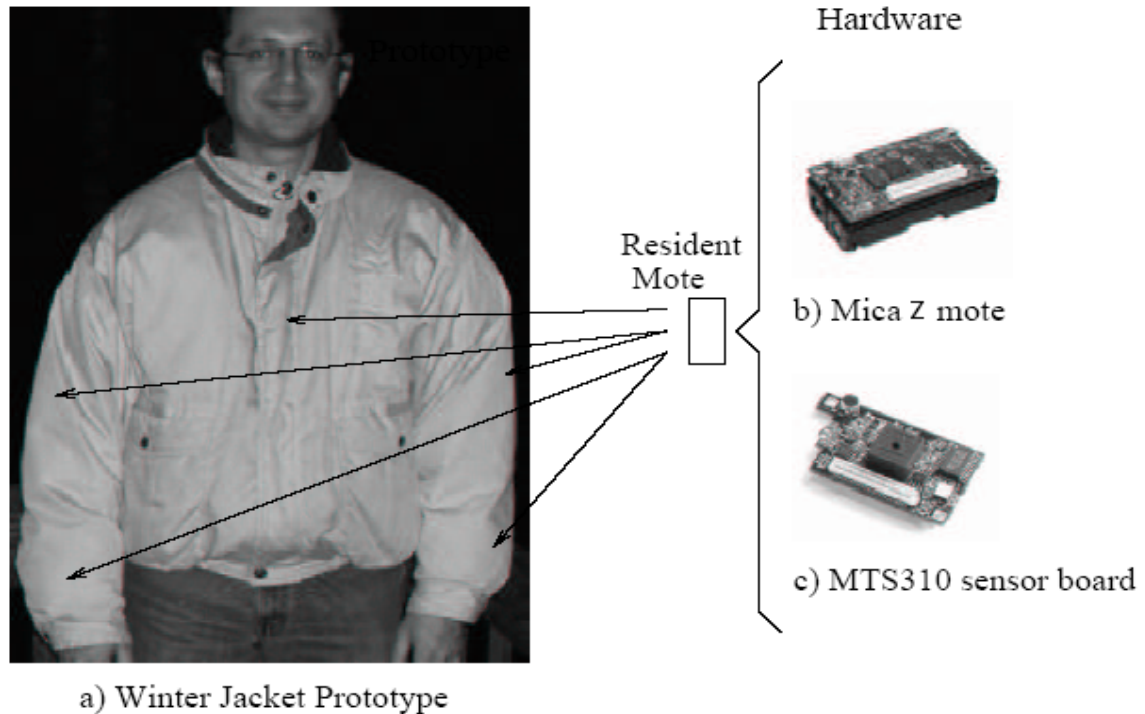
- CodeBlue Software Platform

- From Harvard
  - Routing, naming, discovery, security
  - For medical sensors, PDAs, PCs



# SATIRE – Implementation

- How? – Embed MicaZ motes in winter jacket
- Why MicaZ? - An example, reduction of size in future

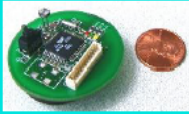






# Summary

- Smart Room spaces (Gaia, CoolTown from HP ,...) important pervasive/ubiquitous system infrastructures and need
  - Wireless sensor networks
  - Ad hoc networks
  - Disruption-tolerant networking
  - Delay tolerant networking
  - Location-sensing
  - Security and privacy

# **ADDITIONAL SLIDES**

# Berkeley Family of Motes

Mote Type	<i>WeC</i>	<i>René</i>	<i>René 2</i>	<i>Dot</i>	<i>Mica</i>	<i>MicaDot</i>
						

## Microcontroller

Type	AT90LS8535	ATmega163	ATmega128
Program memory (KB)	8	16	128
RAM (KB)	0.5	1	4

## Nonvolatile storage

Chip	24LC256	AT45DB041B
Connection type	I <sup>2</sup> C	SPI
Size (KB)	32	512

## Default power source

Type	Lithium	Alkaline	Alkaline	Lithium	Alkaline	Lithium
Size	CR2450	2 x AA	2 x AA	CR2032	2 x AA	3B45
Capacity (mAh)	575	2850	2850	225	2850	1000

## Communication

Radio	TR1000					CC1000
Radio speed (kbps)	10	10	10	10	40	38.4
Modulation type	OOK					ASK FSK

# TinyOS Design Point

(<http://www.tinyos.net>)

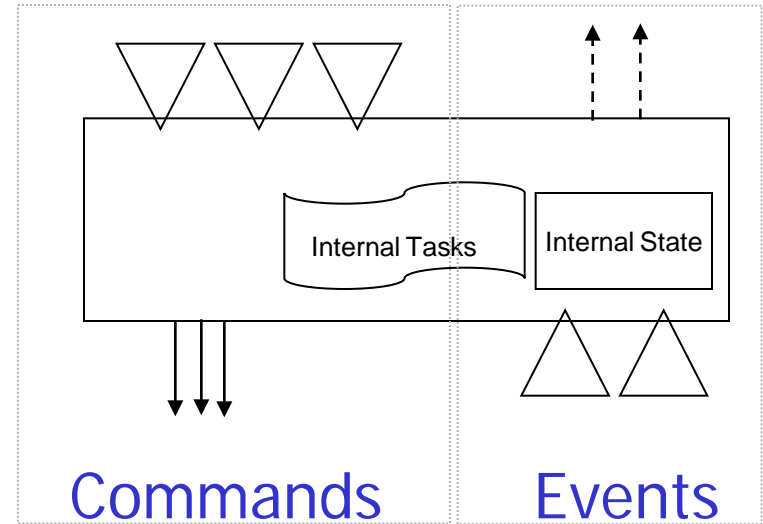
- **Bursty dataflow-driven** computations
- **Multiple data streams** => concurrency-intensive
- **Real-time computations** (hard and soft)
- Power conservation
- Size
- Accommodate diverse set of applications

## □ TinyOS:

- Event-driven execution (*reactive* mote)
- Modular structure (components) and clean interfaces

# TinyOS Component Model

- **Component** specifies:



- Component invocation is **event driven**, arising from hardware events
- *Static allocation* avoids run-time overhead
- **Scheduling**: dynamic, hard (or soft) real-time
- **Explicit interfaces** accommodate different applications



# Programming TinyOS

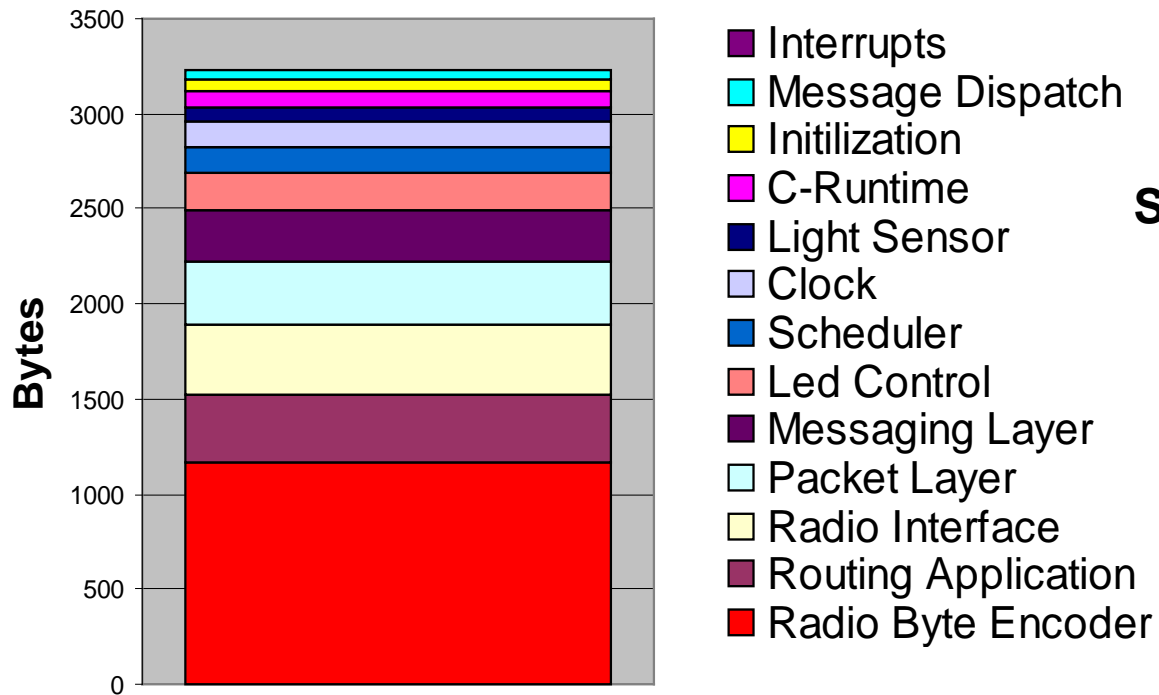
- Use a variant of C called **NesC**
- NesC defines *components*
- A component is either
  - A *module* specifying a set of methods and internal storage (~like a Java static class)

A module corresponds to either a hardware element on the chip (i.e., device driver for, e.g., the clock or the LED), or to a user-defined software module

- Modules implement and use
  - *interfaces*
  - Or a *configuration* , a set of other components *wired* (virtually) together by specifying the unimplemented methods invocation mappings
- A complete NesC application then consists of one top level configuration

# TinyOS: Size

Code size for ad hoc networking application



**Scheduler: 144 Bytes code**  
**Totals: 3430 Bytes code**  
**226 Bytes data**



# Energy – a critical resource

<i>Component</i>	<i>Rate</i>	<i>Startup time</i>	<i>Current consumption</i>
CPU Active	4 MHz	N/A	4.6 mA
CPU Idle	4 MHz	1 us	2.4 mA
CPU Suspend	32 kHz	4 ms	10 uA
Radio Transmit	40 kHz	30 ms	12 mA
Radio Receive	40 kHz	30 ms	3.6 mA
Photo	2000 Hz	10 ms	1.235 mA
I2C Temp	2 Hz	500 ms	0.150 mA
Pressure	10 Hz	500 ms	0.010 mA
Press Temp	10 Hz	500 ms	0.010 mA
Humidity	500 Hz	500 ms	0.775 mA
Thermopile	2000 Hz	200 ms	0.170 mA
Thermistor	2000 Hz	10 ms	0.126 mA

# Other Projects

- **Berkeley**
  - TOSSIM (+TinyViz)
    - TinyOS simulator (+ visualization GUI)
  - TinyDB
    - Querying a sensor net like a database
  - Maté, Trickle
    - Virtual machine for TinyOS motes, code propagation in sensor networks for automatic reprogramming, like an active network.
  - CITRIS
- Several projects in **other universities** too
  - UI, UCLA: networked vehicle testbed