WSN PLATFORMS, HARDWARE & SOFTWARE

Murat Demirbas
SUNY Buffalo

Last lecture: Why use WSNs

Ease of deployment: Wireless communication means no need for a communication infrastructure setup

Low-cost of deployment: Nodes are built using off-the-shelf cheap components

Fine grain monitoring: Feasible to deploy nodes densely for fine grain monitoring

HARDWARE PLATFORMS

3 broad types of nodes

Grain sized: RFID, smart dust

Matchbox sized: Berkeley motes and derivatives

Brick sized: Stargates (potentially using wall power)

Node	CPU	Power	Memory	I/O and Sensors	Radio	Remarks
Special-purpose Sensor Nodes						
Spec 2003	4–8Mhz Custom 8-bit	3mW peak 3uW idle	3K RAM	I/O Pads on chip, ADC	50–100Kbps	Full custom silicon, traded RF range and accuracy for low-power operation.
Generic Sensor Nodes						
Rene 1999	ATMEL 8535	.036mW sleep 60mW active	512B RAM 8K Flash	Large expansion connector	10Kbps	Primary TinyOS development platform.
Mica-2 2001	ATMEGA 128	.036mW sleep 60mW active	4K RAM 128K Flash	Large expansion connector	76Kbps	Primary TinyOS development platform.
Telos 2004	Motorola HCS08	.001mW sleep 32mW active	4K RAM	USB and Ethernet	250Kbps	Supports IEEE 802.15.4 standard. Allows higher-layer Zigbee stardard. 1.8V operation
Mica-Z 2004	ATMEGA 128		4K RAM 128K Flash	Large expansion connector	250Kbps	Supports IEEE 802.15.4 standard. Allows higher-layer Zigbee stardard.
High-bandwidth Sensor Nodes						
BT Node 2001	ATMEL Mega 128L 7.328Mhz	50MW idle 285MW active	128KB Flash 4KB EEPROM 4KB SRAM	8-channel 10-bit A/D, 2 UARTS Expandable connectors	Bluetooth	Easy connectivity with cell phones. Supports TinyOS. Multihop using multiple radios/nodes.
Imote 1.0 2003	ARM 7TDMI 12-48MHz	1mW idle 120mW active	64KB SRAM 512KB Flash	UART, USB, GPIO, I^2C, SPI	Bluetooth 1.1	Multihop using scatternets, easy connections to PDAs, phones, TinyOS 1.0, 1.1.
Gateway Nodes						
Stargate 2003	Intel PXA255		64KNSRM	2 PCMICA/CF, com ports, Ethernet, USB	Serial connection to sensor network	Flexible I/O and small form factor power management.
Inrysnc Cerfcube 2003	Intel PXA255		32KB Flash 64KB SRAM	Single CF card, general-purpose I/O		Small form factor, robust industrial support, Linux and Windows CE support.
PC104 nodes	X86 processor		32KB Flash 64KB SRAM	PCI Bus		Embedded Linux or Windows support.

GRAIN-SIZED

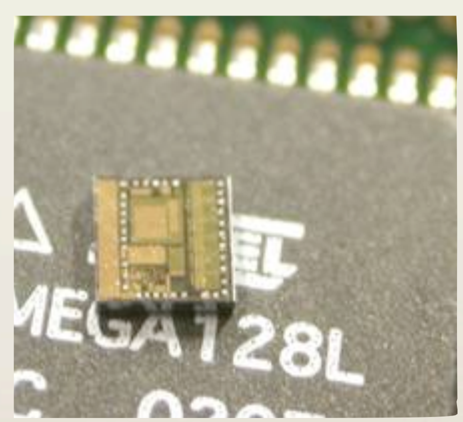
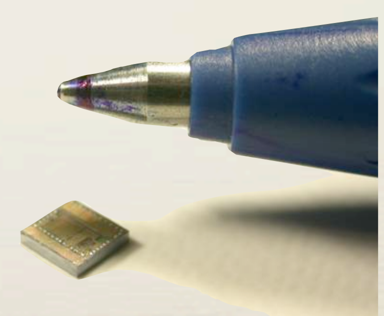
Grain-sized nodes

RFIDs are powered by inductive coupling to a transmission from a reader device to transmit a message back, and are available commercially at very low prices

Computation power is severely limited, usually they only transmit stored unique id and variable

Spec mote (2003)

size 2x2.5mm, AVR RISC core, 3KB memory, FSK radio (CC1000), encrypted communication hardware support, memory-mapped active messages

MATCHBOX-SIZED

Matchbox-sized nodes

Examples are, Mica, Mica2, Telos motes, XSM node

8-bit microprocessor, 4MHz CPU ATMEGA 128, ATMEL 8535, or Motorola HCS08

~8Kb RAM, holds run-time state (values of the variables) of the program

Flash memory in motes

~128Kb programmable Flash memory, holds the application program which is downloaded via a programmer-board or wirelessly

additional Flash memory storage space up to 512Kb for logging sensor data

Mica2 and MicaDot

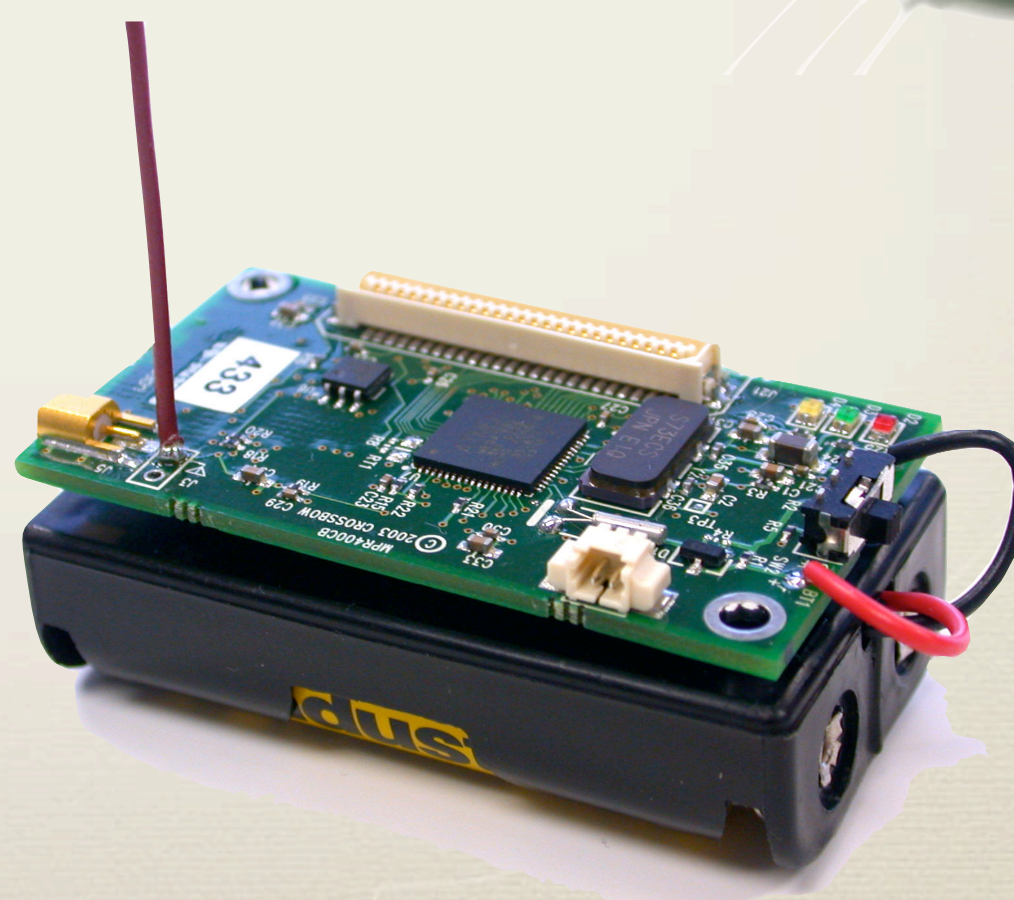
ATmega128 CPU

Self-programming

Chipcon CC1000

FSK, Tunable frequency

2 AA battery = 3V

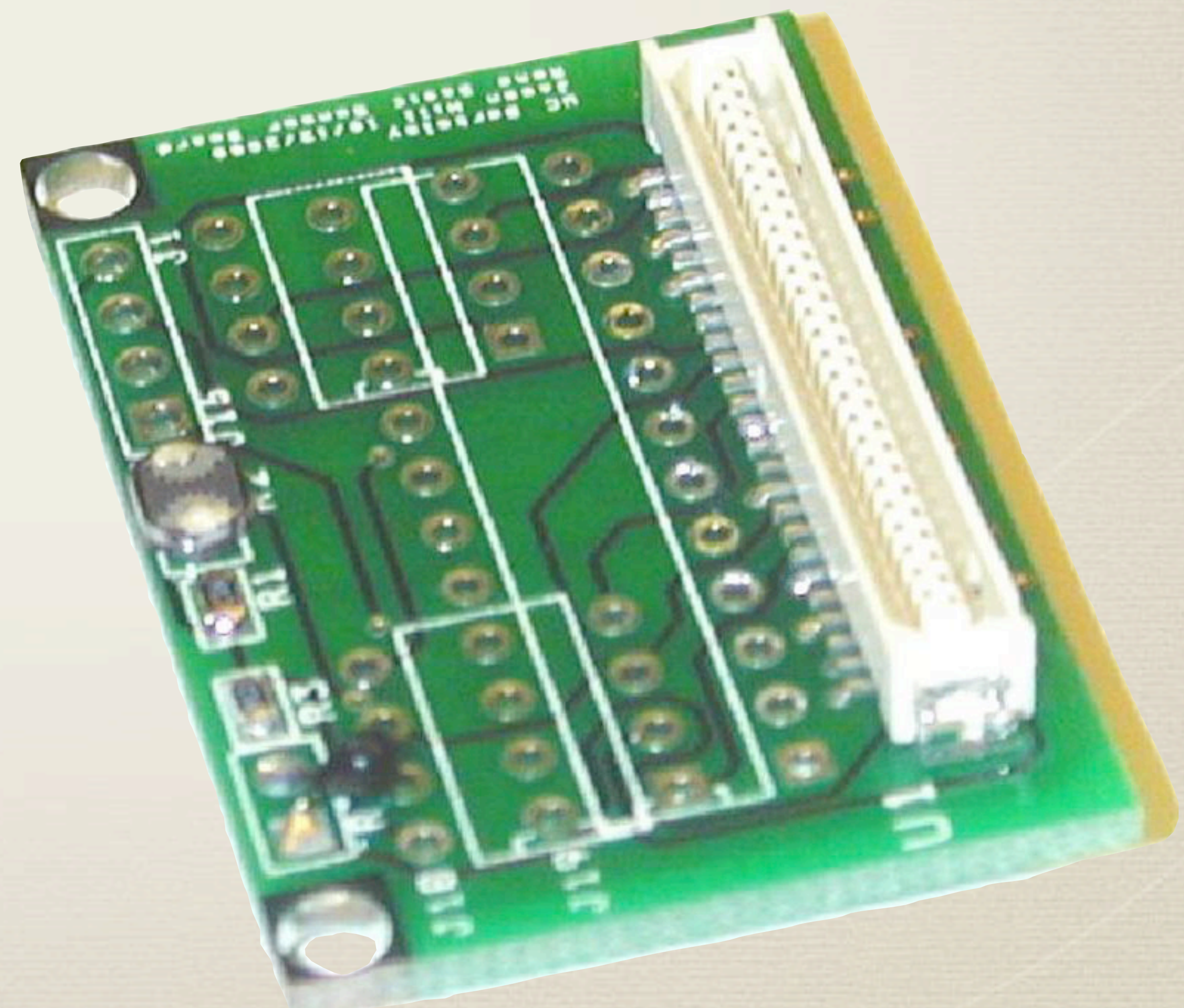


Basic sensor board

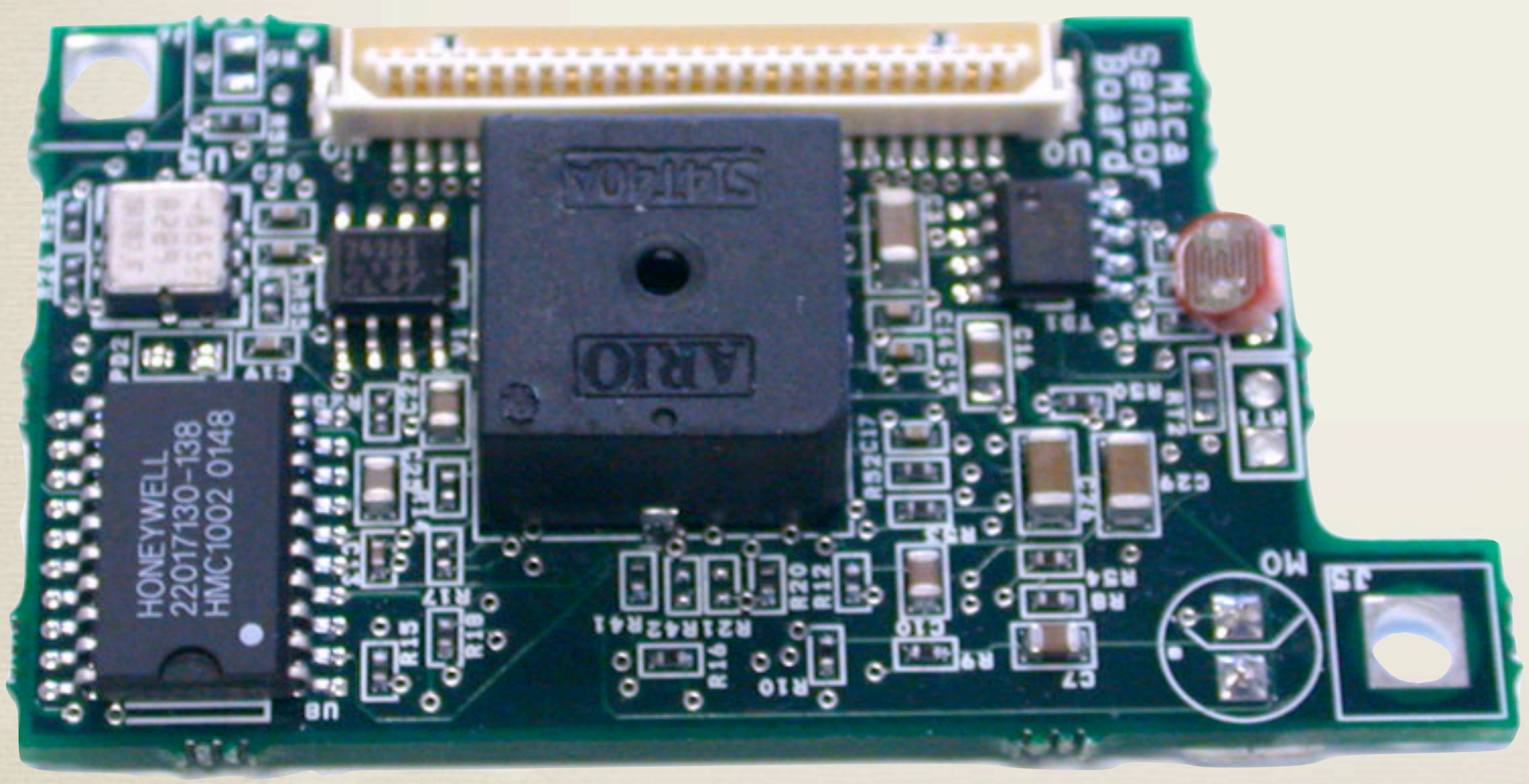
Light (Photo)

Temperature

Prototyping space for new hardware designs



Mica sensor board



Light

Temperature

Acceleration 2 axis
Resolution: ±2mg

Magnetometer
Resolution: 134µG

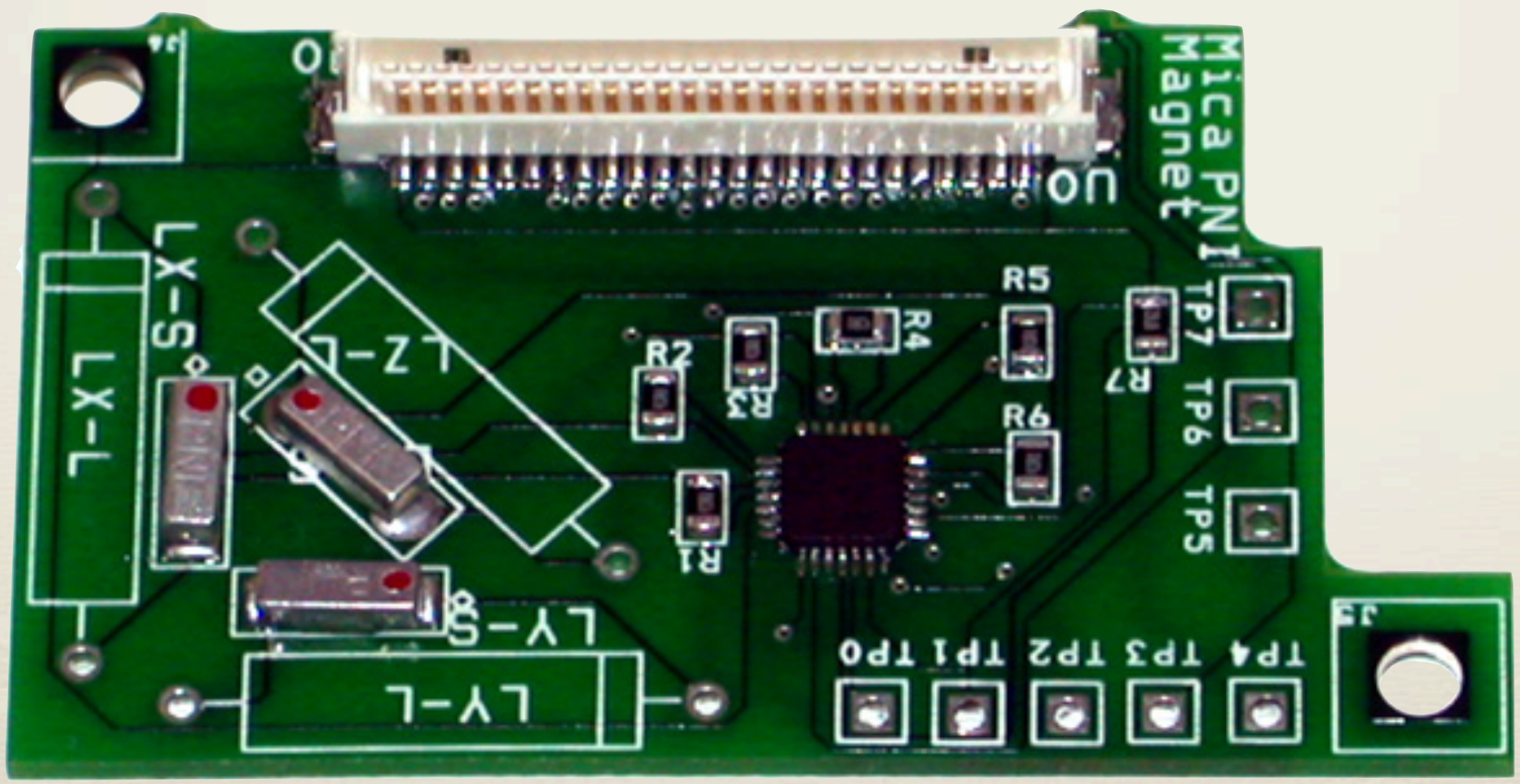
Microphone

Sounder 4.5kHz

Magnetometer/compass

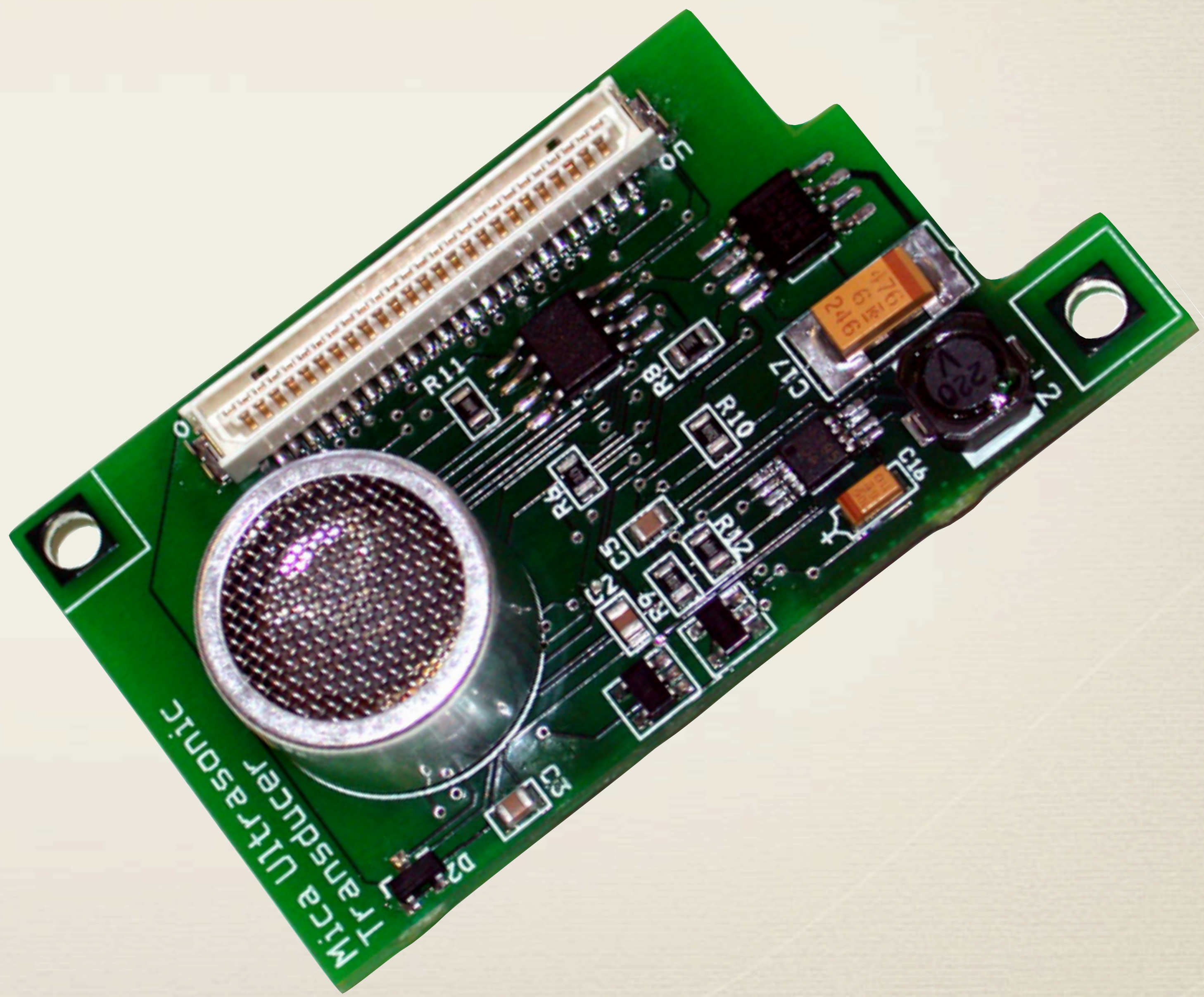
Resolution: 400 μ Gauss

Three axis, under $15 in large quantities



Ultrasonic transceiver

- Used for ranging
- Up to 2.5m range
- 6cm accuracy
- Dedicated microprocessor



Mica weather board

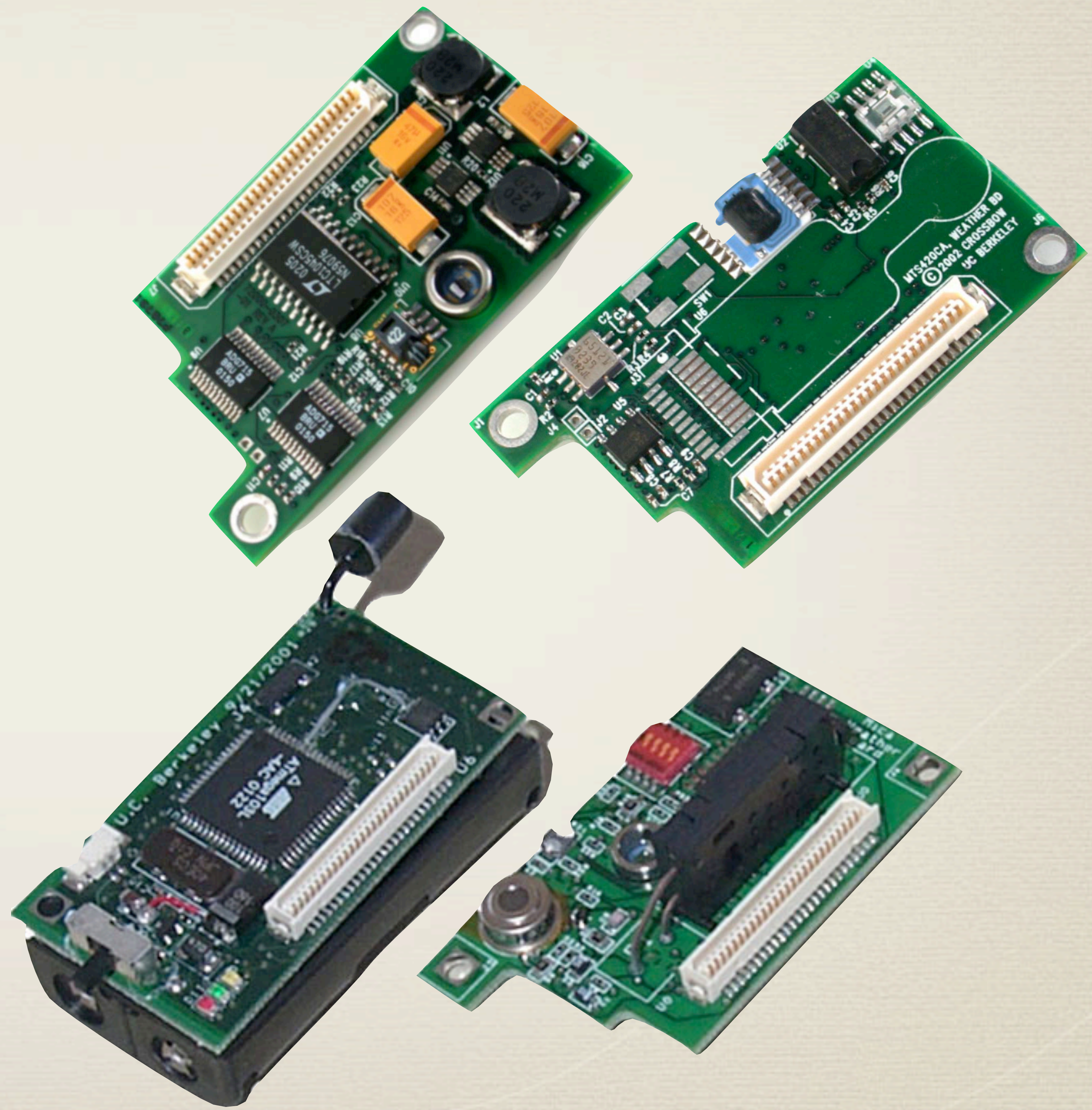
Photosynthetically Active Radiation

Humidity
Temperature

Barometric Pressure

Acceleration 2 axis

UCB, Crossbow, UCLA



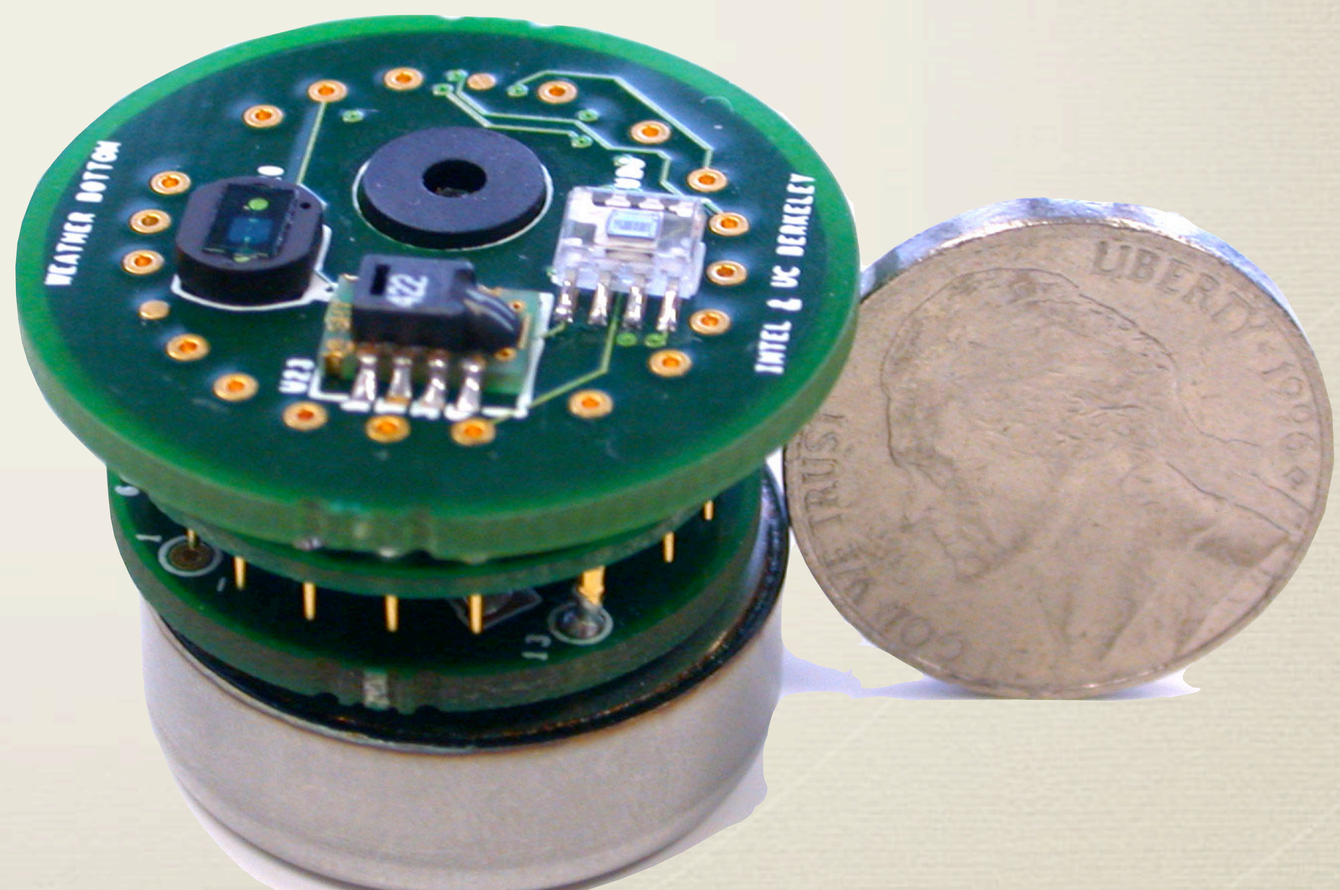
MicaDot sensorboards

"Dot" sensorboards (1"diameter)

HoneyDot: Magnetometer

Ultrasonic Transceiver

Weather Station



XSM node

Derived from Mica2

Better sensor range

4 Passive Infrared: ~ 25m for SUV

Sounder: ~10m

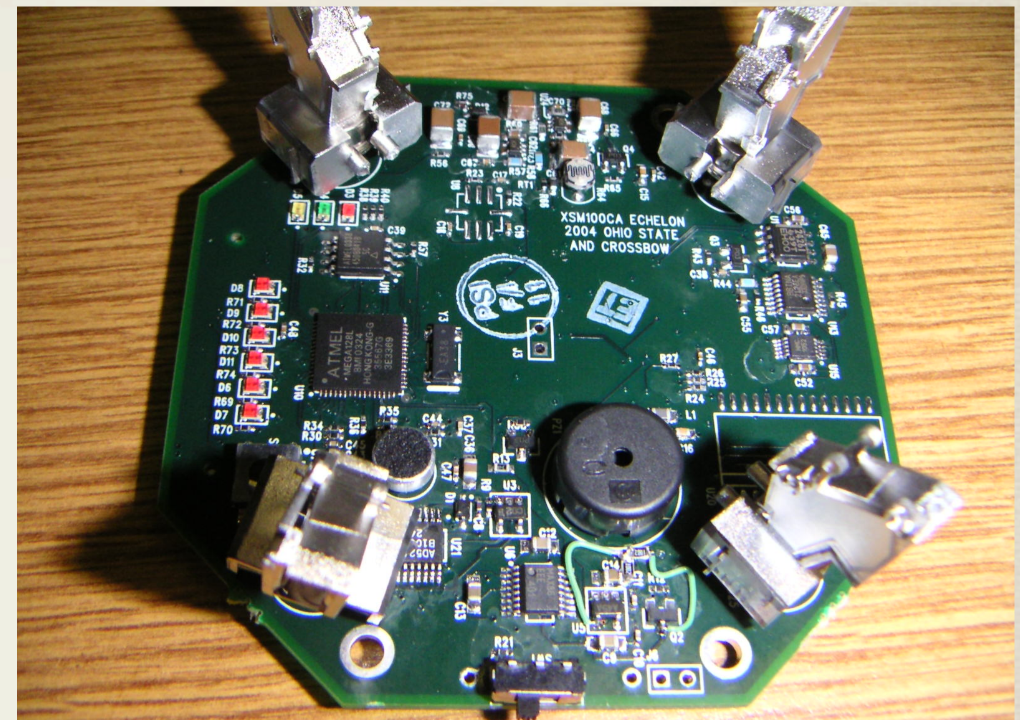
Microphone: ~ 50m for ATV

Magnetometer: ~ 7m for SUV

Better radio range ~30m

Grenade timer

Wakeup circuits (Mic, PIR)



Telos mote

Low Power

Integrated antenna
(50m-125m)

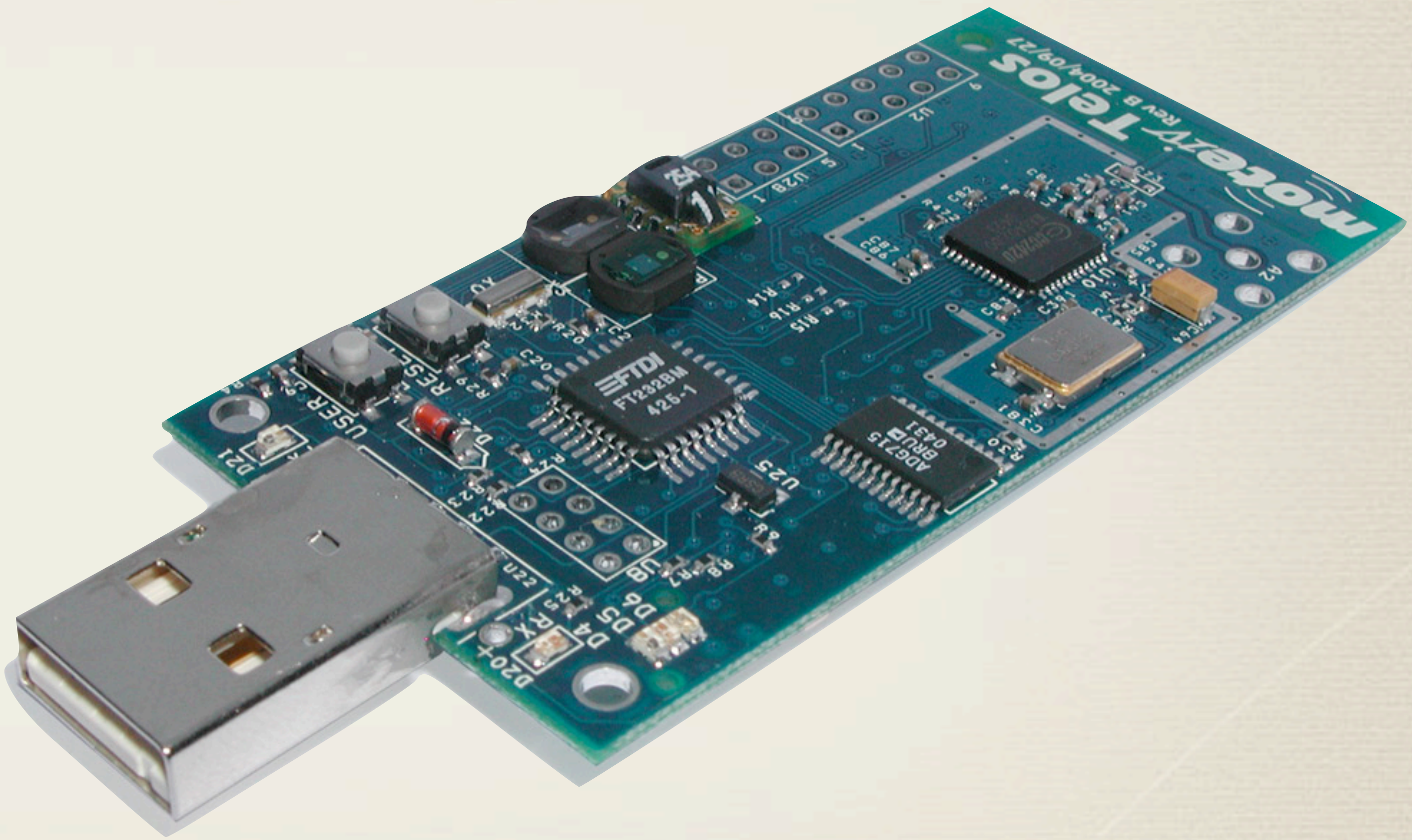
USB

IEEE 802.15.4

(CC2420 radio)

10kB RAM, 16-bit core

DMA transfers while CPU off



Telos is low power

Operation	Telos	Mica2	MicaZ
Minimum Voltage	1.8V	2.7V	2.7V
Mote Standby (RTC on)	5.1 μA	19.0 μA	27.0 μA
MCU Idle (DCO on)	54.5 μA	3.2 mA	3.2 mA
MCU Active	1.8 mA	8.0 mA	8.0 mA
MCU + Radio RX	21.8 mA	15.1 mA	23.3 mA
MCU + Radio TX (0dBm)	19.5 mA	25.4 mA	21.0 mA
MCU + Flash Read	4.1 mA	9.4 mA	9.4 mA
MCU + Flash Write	15.1 mA	21.6 mA	21.6 mA
MCU Wakeup	6 μs	180 μs	180 μs
Radio Wakeup	580 μs	1800 μs	860 μs

BRICK-SIZED

Stargate

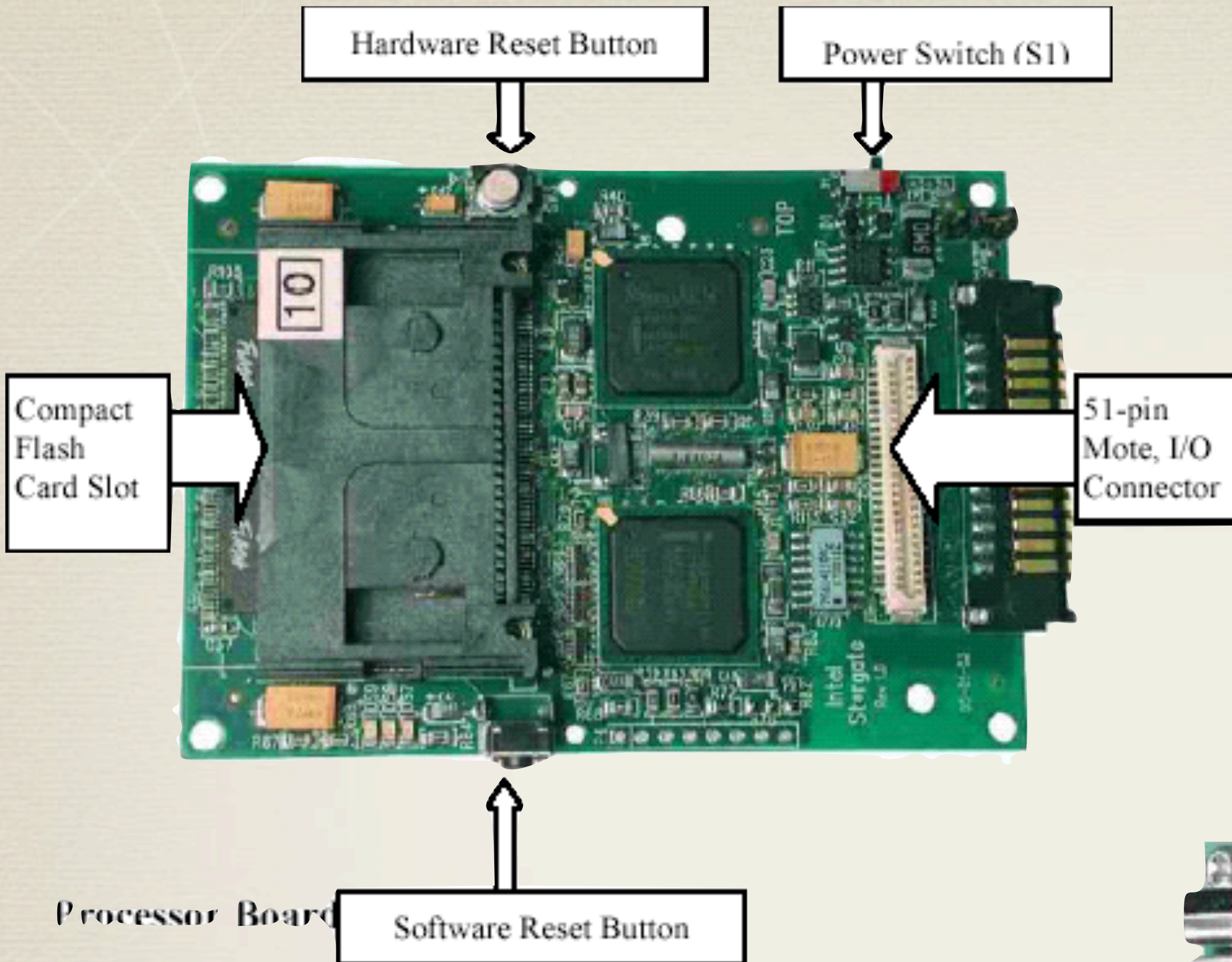
Mini Linux computers communicating via 802.11 radios

Computationally powerful

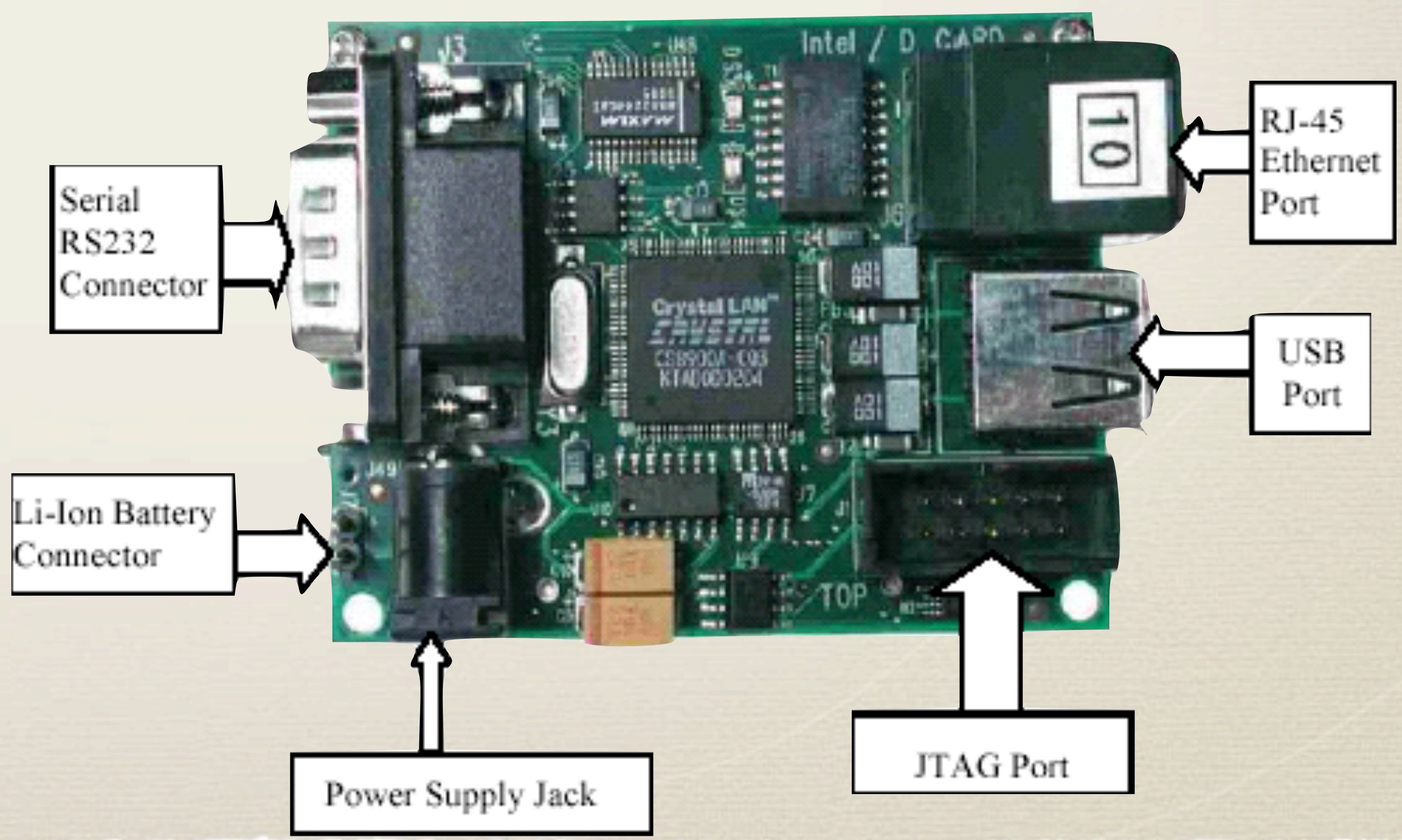
High bandwidth

Requires more energy (AA infeasible)

Used as a gateway between the Internet and WSN



Processor Board



Manifacturers

Crossbow (www.xbow.com) : Mica2, Dot, Micaz, Dot

Intel Research: Stargate, iMote, iMote2

Moteiv: Telos Mote

Dust Inc: Smart Dust

Sensoria Corporation (www.sensoria.com) : WINS NG

Millenial Net (www.millenial.com) : iBean sensor nodes

Ember (www.ember.com): IEEE 802.15.4 (zigbee) nodes

RECAP

Challenges in WSNs

Energy constraint :	battery powered
Unreliable commn. :	limited bursty bandwidth
Unreliable sensors :	false positives
Ad hoc deployment :	no pre-configuration
Large scale networks :	inscalable algorithms
Limited computation :	no centralized algorithms
Distributed execution :	difficult to debug & get it right

Opportunities in WSNs

Redundancy :	many nodes in same area
Precise clock at nodes :	synchronized clocks
Atomic broadcast primitive :	all recipients hear same message at same time
Geometry :	Dense nodes over 2D
New applications:	Tracking, querying, localization, network reprogramming, etc.

SOFTWARE PLATFORMS

TinyOS

- Most popular OS for WSN developed by UC Berkeley
- Features a component-based architecture
  - software is written in modular components
  - each component denotes the interfaces that it provides
  - an interface declares a set of functions called commands that the interface provider implements and another set of functions called events that the interface user should be ready to handle
- Easy to link components together by "wiring" their interfaces to form larger components similar to using Lego blocks

TinyOS ...

Provides a component library that includes network protocols, services, and sensor drivers

An application consists of

1) a component written by the application developer and

2) the library components that are used by the components in (1)

An application developer writes only the application component that describes the sensors used, and configures the middleware services with parameters

Benefits of using TinyOS

1) Separation of concerns

TinyOS provides a proper networking stack for wireless communication that abstracts away the underlying problems and complexity of message transfer from the application developer

E.g., MAC layer

Benefits of using TinyOS...

2) Concurrency control

TinyOS provides a scheduler that achieves efficient concurrency control (at the node level)

An interrupt-driven execution model is needed to achieve a quick response time for the events and capture the data

For example, a message transmission may take up to 100msec, and without an interrupt-driven approach the node would miss sensing and processing of interesting data in this period

TinyOS scheduler takes care of the intricacies of interrupt-driven execution and provides concurrency in a safe manner by scheduling the execution in small threads

Benefits of using TinyOS...

3) Modularity

TinyOS's component model facilitates reuse and reconfigurability since software is written in small functional modules. Several middleware services are available as well-documented components

Over 500 research groups and companies are using TinyOS and numerous groups are actively contributing code to the public domain

TinyOS concepts

Microthreaded OS (lightweight thread support) and efficient network interfaces

Two level scheduling structure

Long running tasks that can be interrupted by hardware events

Small, tightly integrated design allows crossover of software components into hardware

TinyOS concepts...

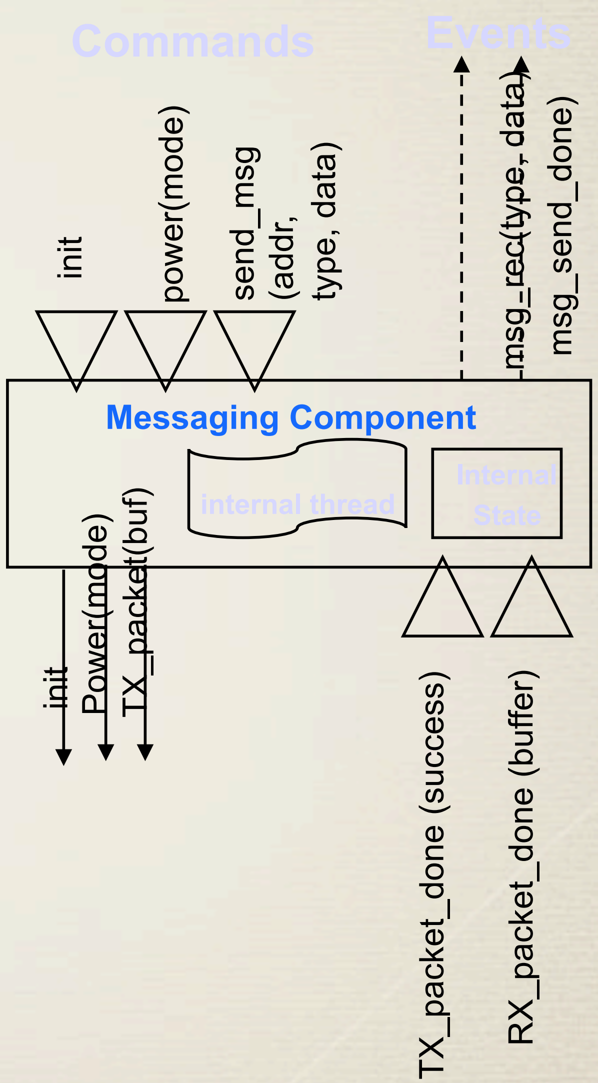
Scheduler + Graph of Components

Component includes :

Commands

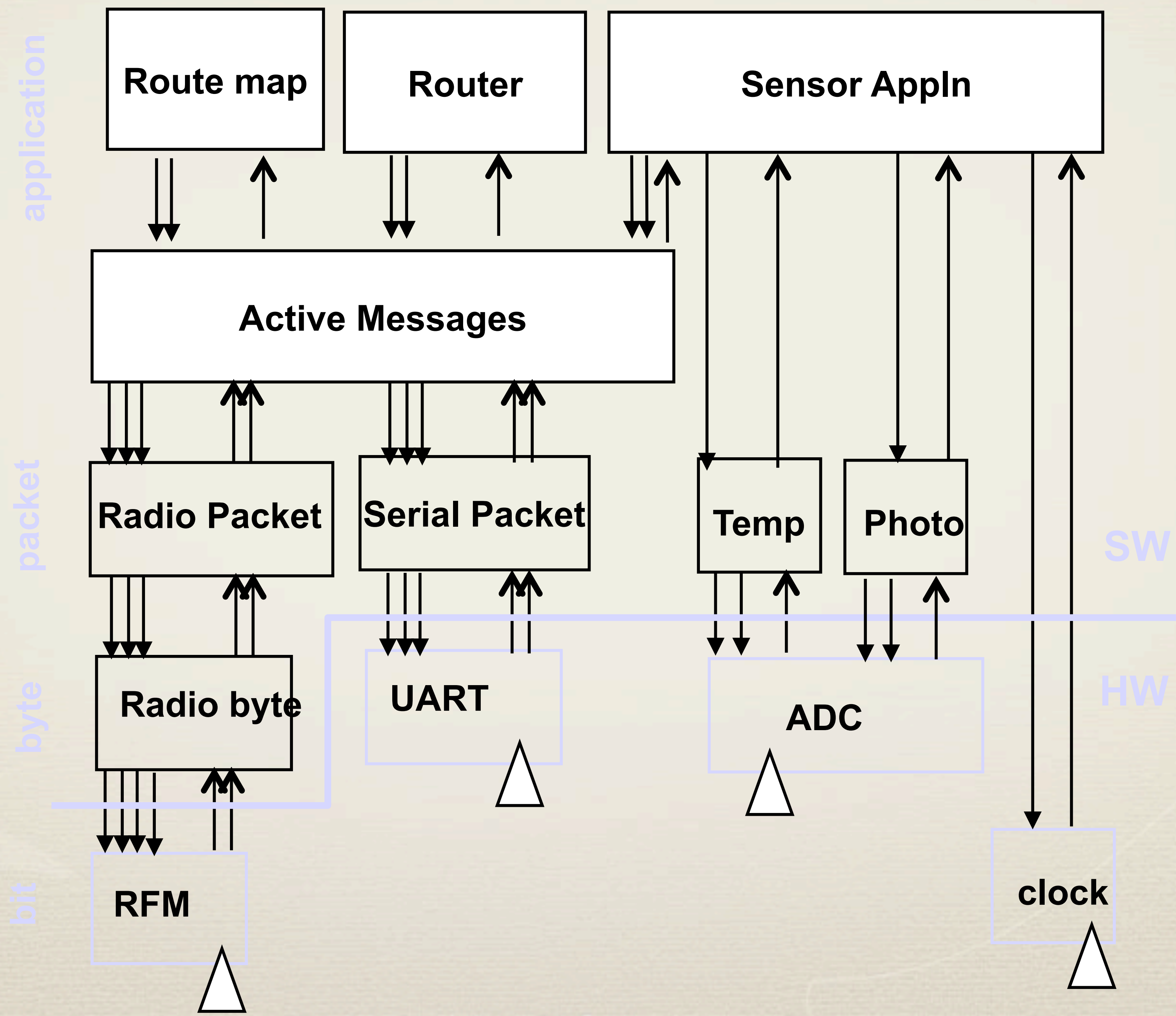
Event Handlers

Tasks (concurrency)



Frame (storage) per component, shared stack, no heap

Application is a graph of components



TinyOS execution model

Commands request action

ack/nack at every boundary

call command or post task

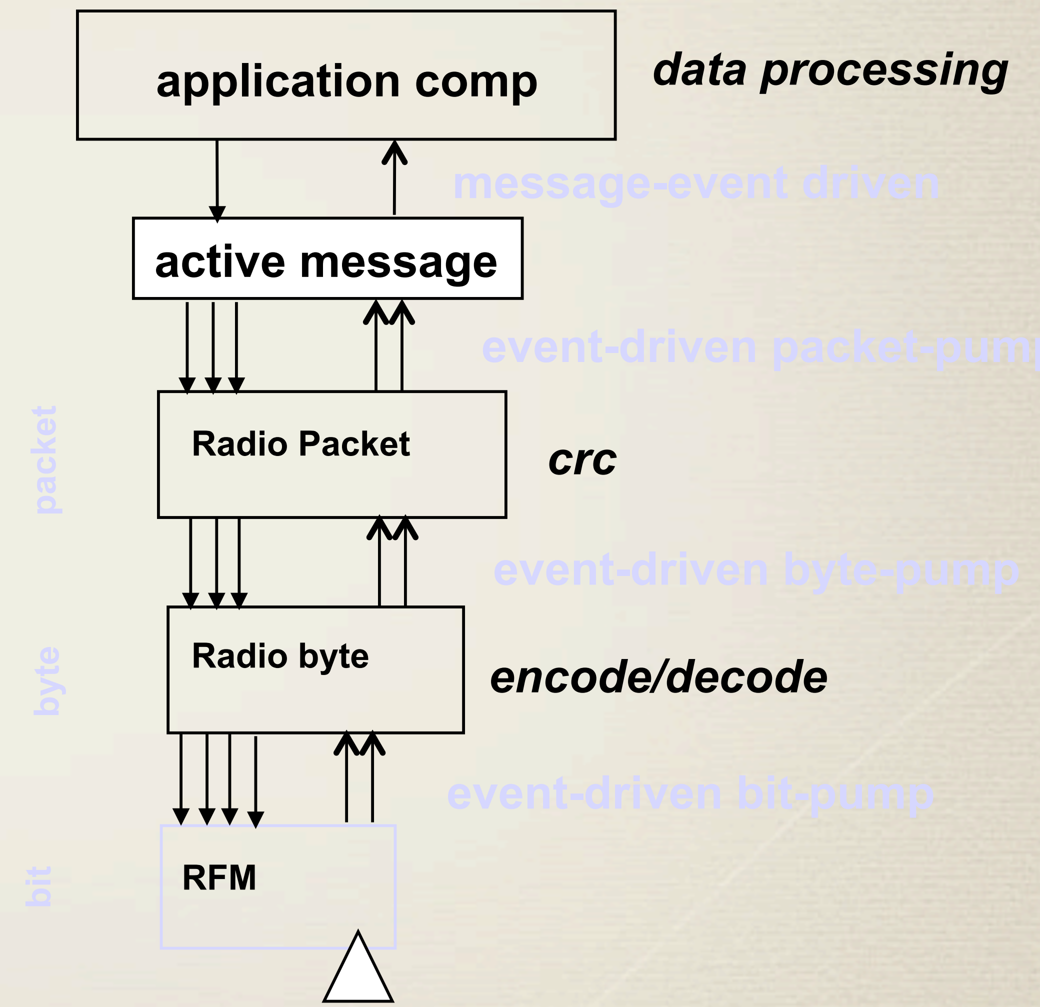
Events notify occurrence

hardware interrupt at lowest level

signal event, call command, or post task

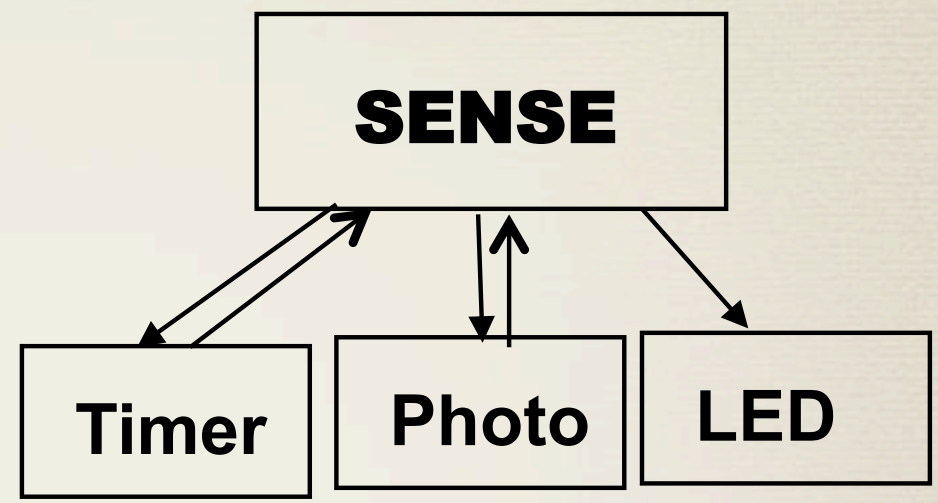
Split-phase operations

command-acked quickly, work done by task, event signals completion



Event-driven sensing app.

```
command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 200);
  }
event result_t Timer.fired() {
    return call sensor.getData();
  }
event result_t sensor.dataReady(uint16_t data) {
    display(data)
    return SUCCESS;
  }
```



clock event handler initiates data collection
sensor signals data ready event
data event handler calls output command
device sleeps or handles other activity while waiting
conservative send/ack at component boundary

TinyOS commands & events

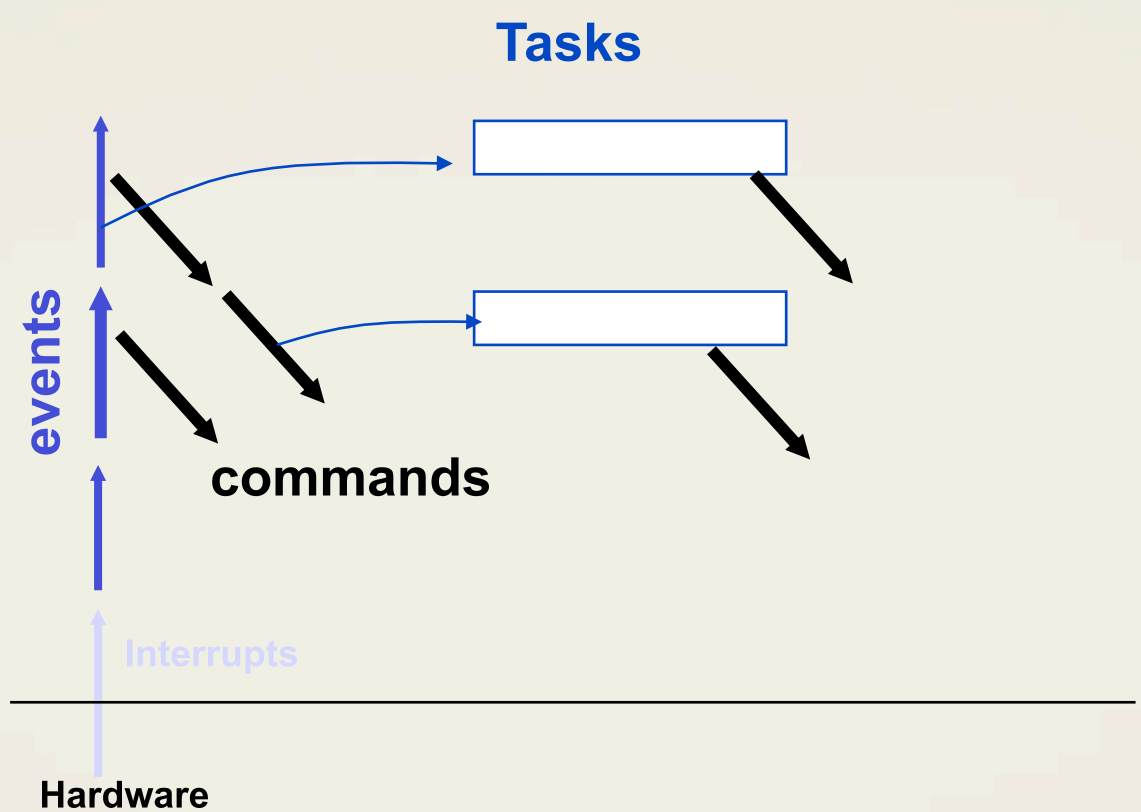
```
{
...
 status = call CmdName(args)
...
}
```

```
command CmdName(args) {
...
return status;
}
```

```
event EvtName(args) {
...
return status;
}
```

```
{
...
 status = signal EvtName(args)
...
}
```

TinyOS execution contexts



Events generated by interrupts preempt tasks

Tasks do not preempt tasks

Tasks

Provide concurrency internal to a component, and longer running operations

Tasks are preempted by events, able to perform operations beyond event context, may call commands, may signal events, not preempted by tasks

Typical use of tasks

event driven data acquisition

schedule task to do computational portion

```
event result_t sensor.dataReady(uint16_t data) {
    putdata(data);
    post processData();
    return SUCCESS;
 }
task void processData() {
    int16_t i, sum=0;
    for (i=0; i < maxdata; i++)
        sum += (rdata[i] >> 7);
    display(sum >> shiftdata);
 }
```

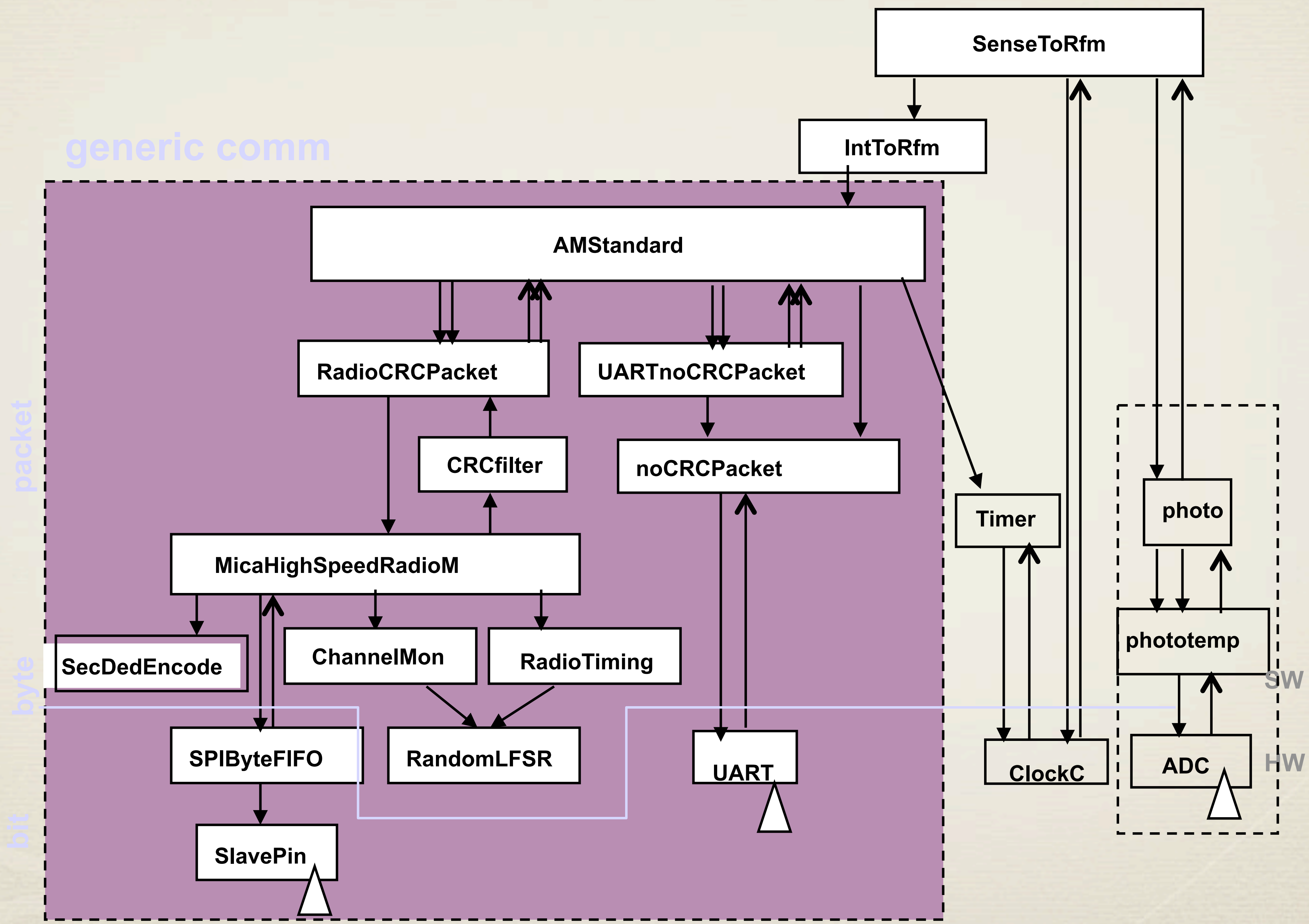
Task scheduling

- Currently simple fifo scheduler
- Bounded number of pending tasks
- When idle, shuts down node except clock
- Uses non-blocking task queue data structure
- Simple event-driven structure + control over complete application/system graph instead of complex task priorities

Maintaining schedule agility

- Need logical concurrency at many levels of the graph
- While meeting hard timing constraints, sample the radio in every bit window
- Retain event-driven structure throughout application
- Tasks extend processing outside event window
- All operations are non-blocking

The complete application



TINYOS SYNTAX

TinyOS

TinyOS 2.0 is written in an extension of C, called nesC, applications are also in nesC

NesC provides syntax for TinyOS concurrency and storage model: commands, events, tasks, local frame variable

Compositional support: separation of definition and linkage, robustness through narrow interfaces and reus

Whole system analysis and optimization

Components

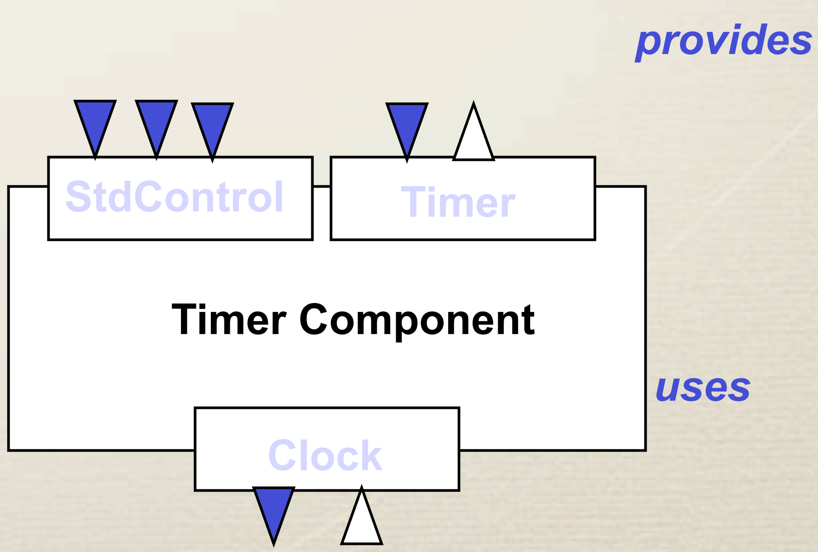
A component specifies a set of interfaces by which it is connected to other components:

provides a set of interfaces to others, and

uses a set of interfaces provided by others

Interfaces are bidirectional: includes commands and

```
provides
    interface StdControl;
    interface Timer:
uses
    interface Clock
```



Component Interface

logically related set of commands and events

StdControl.nc

```
interface StdControl {
      command result_t init();
      command result_t start();
      command result_t stop();
}
```

Clock.nc

```
interface Clock {
   command result_t setRate(char interval, char scale);
   event result_t fire();
}
```

Component types

Configurations:

link together components to compose new component

configurations can be nested

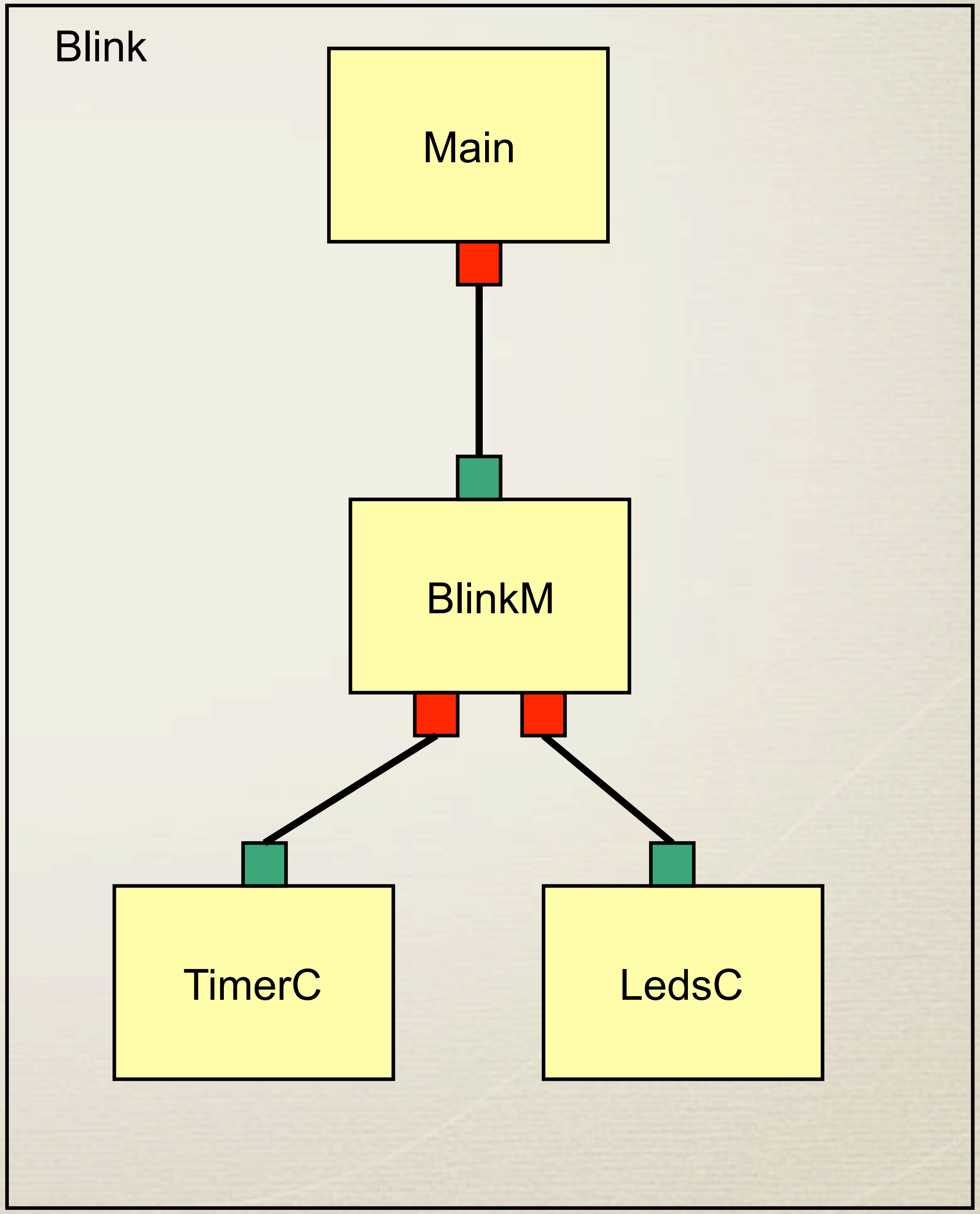
complete “main” application is always a configuration

Modules:

provides code that implements one or more interfaces and internal behavior

Blink example

```
configuration Blink {
}

implementation {
  components Main, BlinkM, TimerC, LedsC;

  Main.StdControl -> TimerC.StdControl;
  Main.StdControl -> BlinkM.StdControl;

  BlinkM.Timer -> TimerC.Timer[unique("Timer")];
  BlinkM.Leds -> LedsC;
}
```



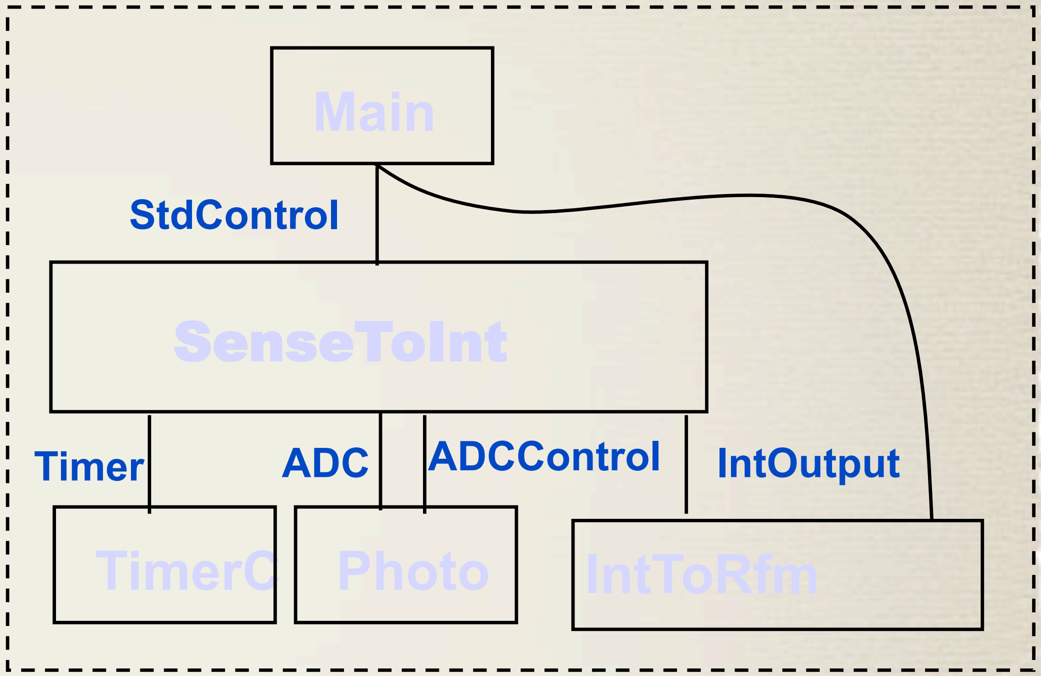
BlinkM module

```
module BlinkM {
   provides interface StdControl;
   uses interface Timer;
   uses interface Leds;
}

implementation {

   command result_t StdControl.init() {
      call Leds.init();
      return SUCCESS;
   }
```

```
   command result_t StdControl.start() {
      return call Timer.start(TIMER_REPEAT, 1000);
   }

   command result_t StdControl.stop() {
      return call Timer.stop();
   }

   event result_t Clock.fire() {
      call Leds.redToggle();
      return SUCCESS;
   }
}
```

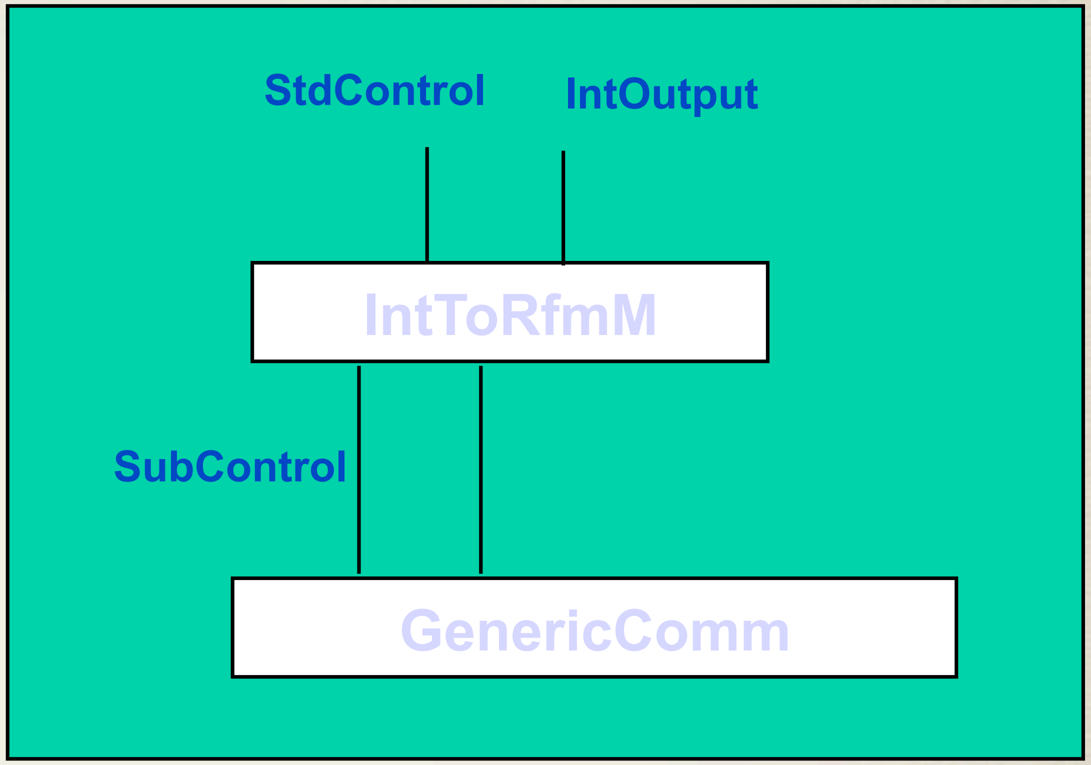
SenseToRFM example

```
configuration SenseToRfm {
}
implementation
{
  components Main, SenseToInt, IntToRfm,
TimerC, Photo as Sensor;

  Main.StdControl -> SenseToInt;
  Main.StdControl -> IntToRfm;

  SenseToInt.Timer ->
TimerC.Timer[unique"Timer"];
  SenseToInt.ADC -> Sensor;
  SenseToInt.ADCControl -> Sensor;
  SenseToInt.IntOutput -> IntToRfm;
}
```



Nested configuration

```
includes IntMsg;
configuration IntToRfm
{
  provides {
    interface IntOutput;
    interface StdControl;
  }
}
implementation
{
  components IntToRfmM, GenericComm as Comm;

  IntOutput = IntToRfmM;
  StdControl = IntToRfmM;

  IntToRfmM.Send -> Comm.SendMsg[AM_INTMSG];
  IntToRfmM.SubControl -> Comm;
}
```



IntToRFM module

```
includes IntMsg;

module IntToRfmM
{
  uses {
    interface StdControl as SubControl;
    interface SendMsg as Send;
  }
  provides {
    interface IntOutput;
    interface StdControl;
  }
}
implementation
{
  bool pending;
  struct TOS_Msg data;

  command result_t StdControl.init() {
    pending = FALSE;
    return call SubControl.init();
  }
```

```
command result_t StdControl.start()
  { return call SubControl.start(); }
command result_t StdControl.stop()
   { return call SubControl.stop(); }

command result_t IntOutput.output(uint16_t value)
  {
    ...
    if (call Send.send(TOS_BCAST_ADDR,sizeof(IntMsg), &data)
               return SUCCESS;
     ...
  }

event result_t Send.sendDone(TOS_MsgPtr msg, result_t success)
{
    ...
}
}
```

Atomicity support in nesC

Split phase operations require care to deal with pending operations

Race conditions may occur when shared state is accessed by premptible executions, e.g. when an event accesses a shared state, or when a task updates state (premptible by an event which then uses that state)

nesC supports atomic block

implemented by turning of interrupts

for efficiency, no calls are allowed in block

access to shared variable outside atomic block is not allowed

Supporting hw evolution

Component design so HW and SW look the same

example: temp component

may abstract particular channel of ADC on the microcontroller

may be a SW I2C protocol to a sensor board with digital sensor or ADC

HW/SW boundary can move up and down with minimal changes

Sending a message

```
bool pending;
struct TOS_Msg data;
command result_t IntOutput.output(uint16_t value) {
    IntMsg *message = (IntMsg *)data.data;
    if (!pending) {
         pending = TRUE;
         message->val = value;
         message->src = TOS_LOCAL_ADDRESS;
         if (call Send.send(TOS_BCAST_ADDR, sizeof(IntMsg), &data))
              return SUCCESS;
         pending = FALSE;
    }
    return FAIL;
}
```

Refuses to accept command if buffer is still full or network refuses to accept send command

Send done event

```
event result_t IntOutput.sendDone(TOS_MsgPtr msg, result_t success)
 {
   if (pending && msg == &data) {
             pending = FALSE;
             signal IntOutput.outputComplete(success);
     }
   return SUCCESS;
 }
}
```

TinyOS limitations

Static allocation allows for compile-time analysis, but can make programming harder

No support for heterogeneity

Limited visibility, Debugging, Intra-node ft-tolerance

TinyOS tools...

TOSSIM: a simulator for tinyos programs

ListenRaw, SerialForwarder: java tools to receive raw packets on PC from base node

Oscilloscope: java tool to visualize sense data real time

Memory usage: breaks down memory usage per component (in contrib)

TinyOS tools

- Peacekeeper: detect RAM corruption due to stack overflows (in lib)
- Stopwatch: tool to measure execution time of code block by timestamping at entry and exit (in osu CVS server)
- Makedoc and graphviz: generate and visualize component hierarchy
- Surge, Deluge, SNMS, TinyDB