londonknowledgelab

Mobile and Ubiquitous Computing

Wireless Sensor Networks

George Roussos

g.roussos@dcs.bbk.ac.uk

Birkbeck UNIVERSITY OF LONDON

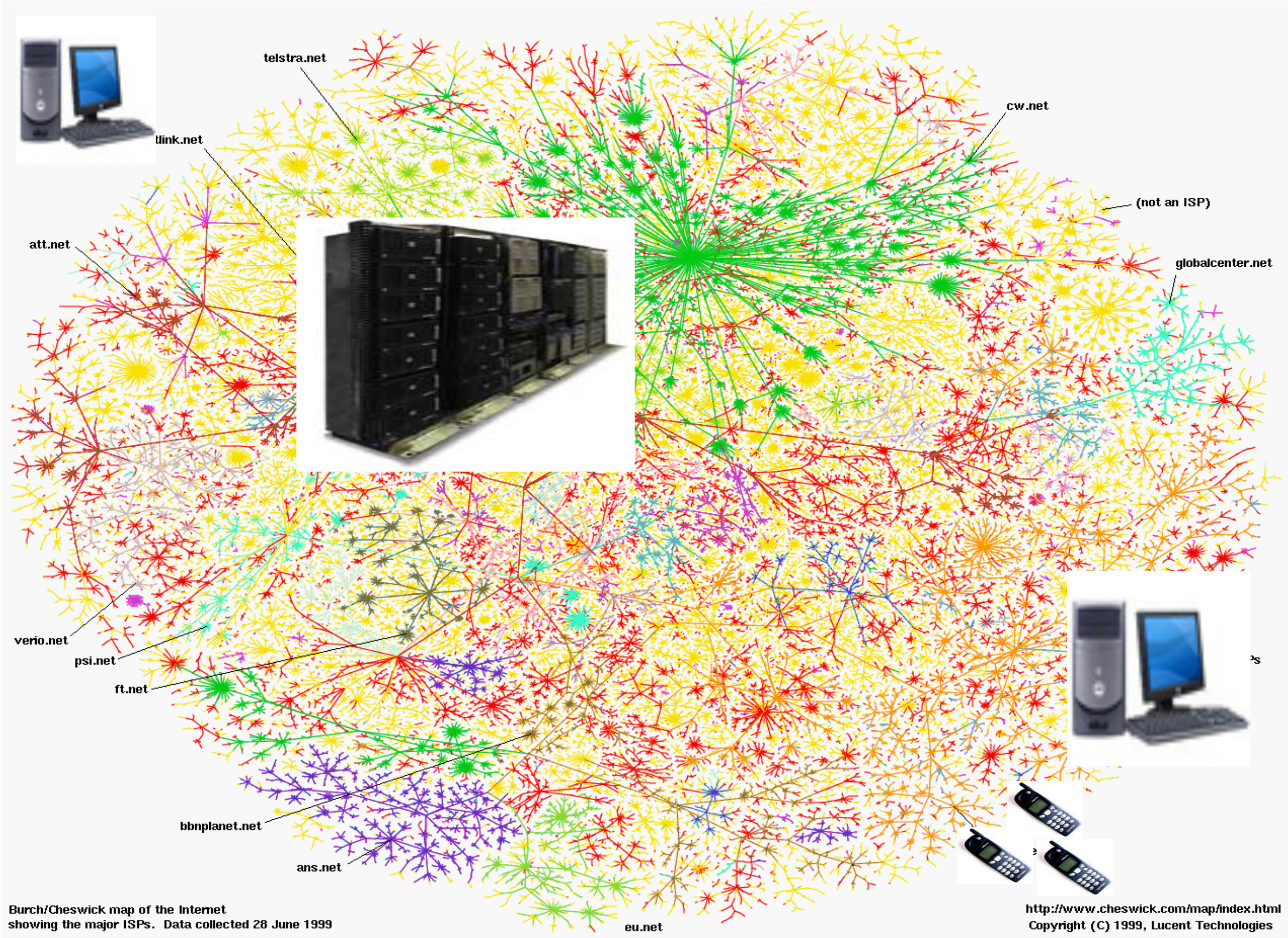
uID Center

Session Overview

- Resource constrained devices
  - evolution, architecture, components
  - a detailed example
- Energy efficiency
- Programming primitives in Tiny OS
- Concurrency

Birkbeck
UNIVERSITY OF LONDON

The Internet today

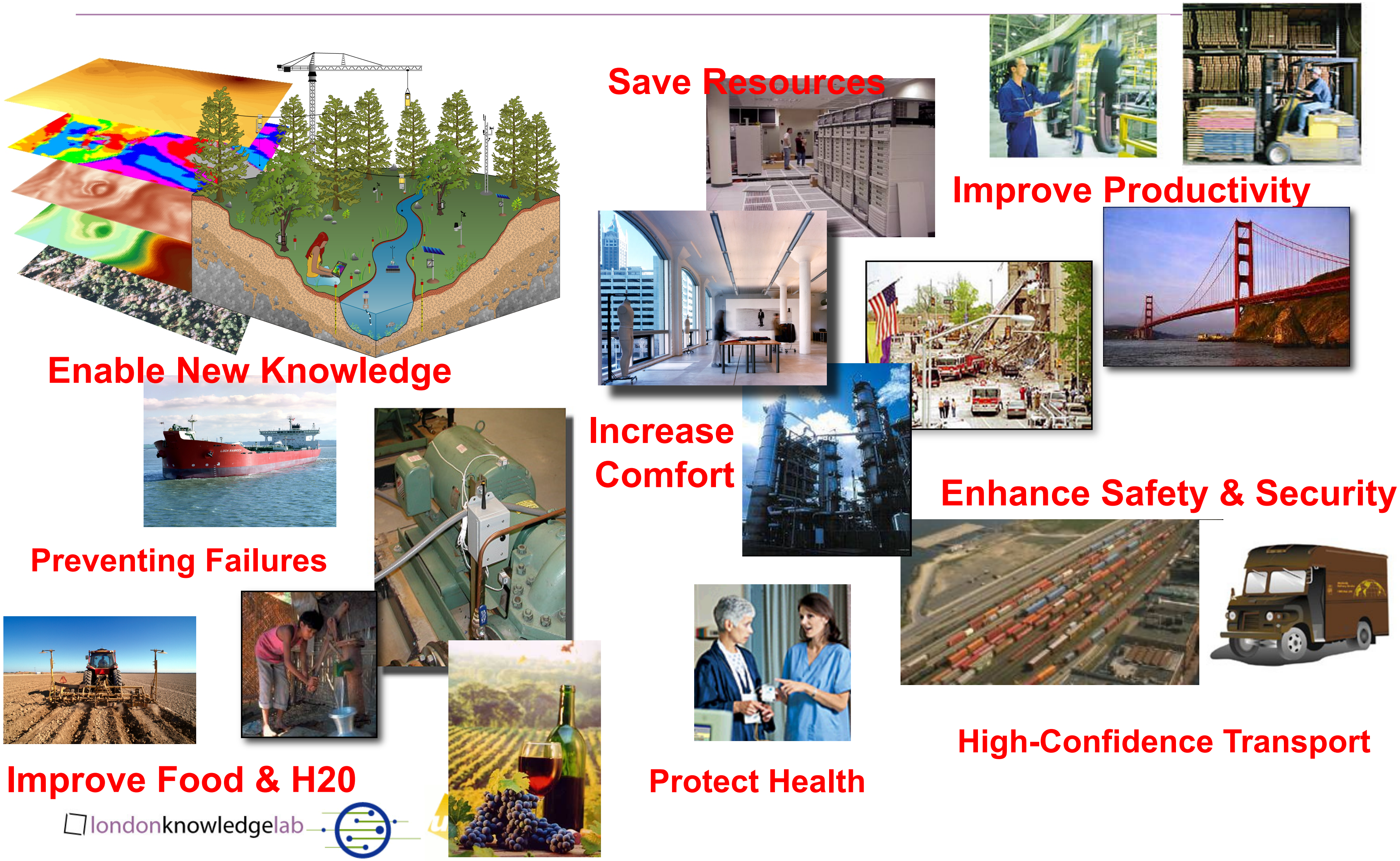


londonknowledgelab uID Center

The Internet ahead

Birkbeck UNIVERSITY OF LONDON

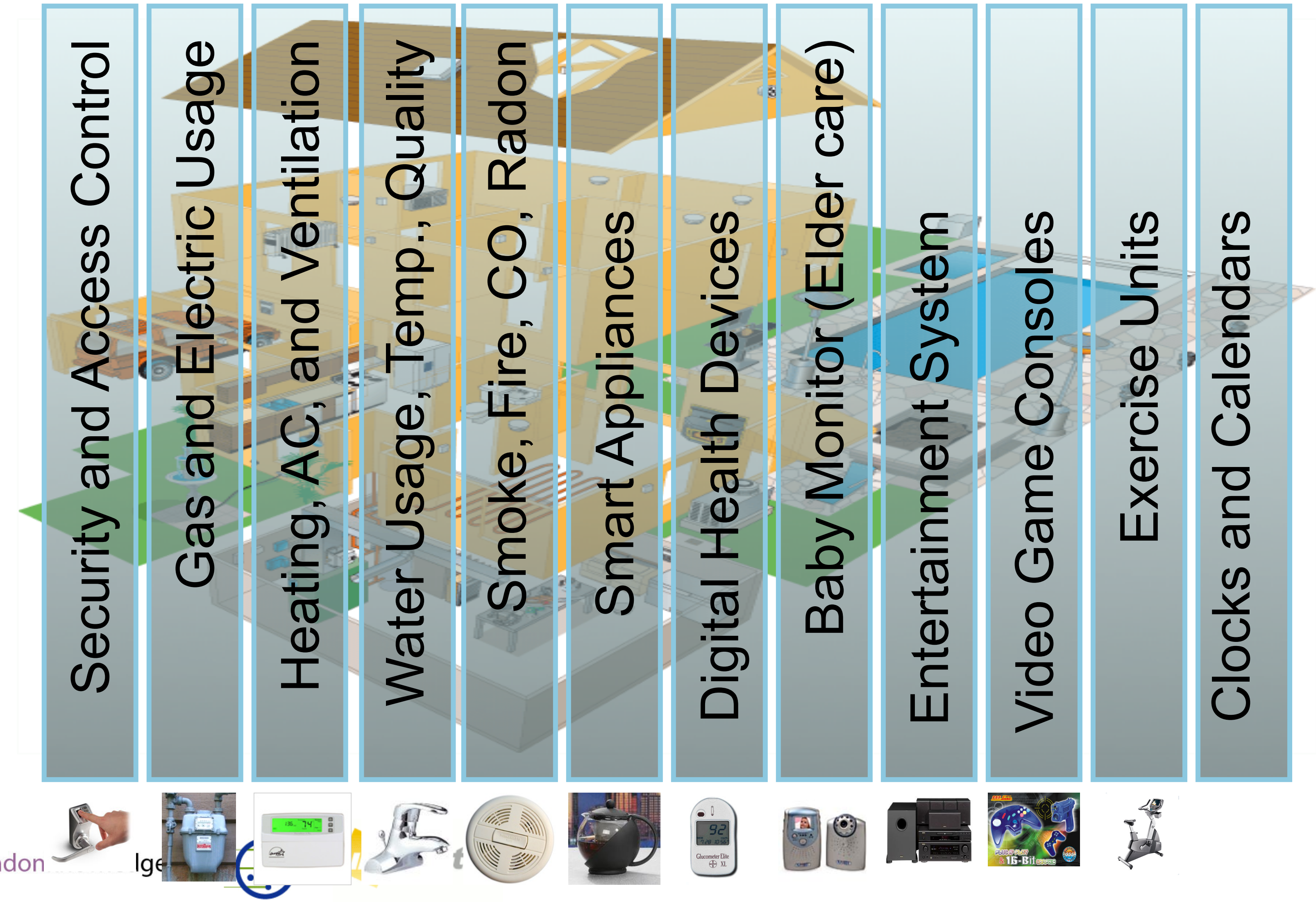
Why "Real" Information is so Important?



londonknowledgelab

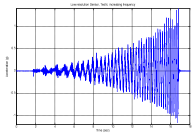
The Web: *Human Generated* Information

Most Real World Information is lost



Physical Information Streams

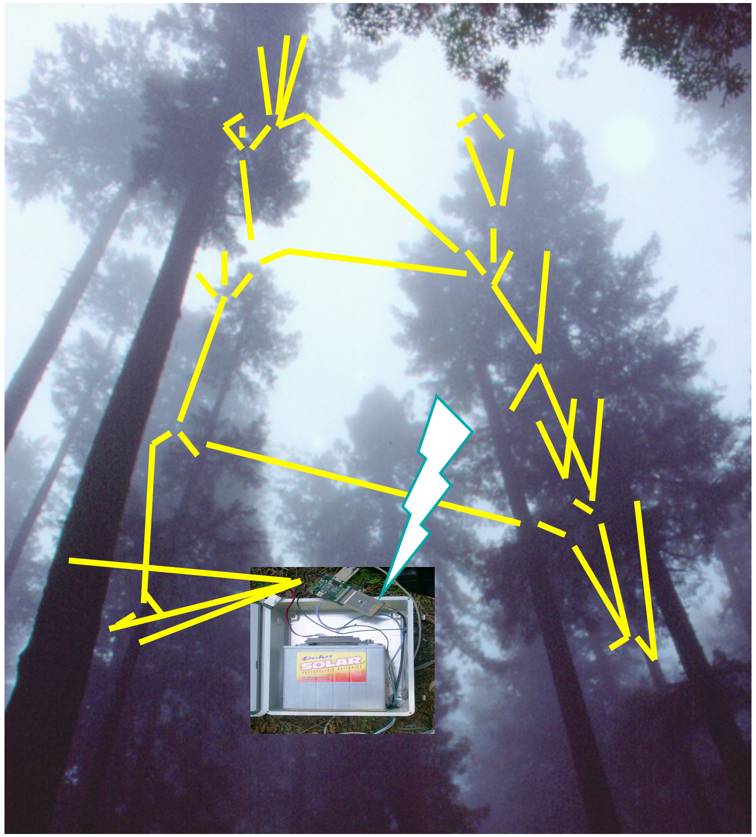
<value>
temp=35
<\value>



- Sensors are everywhere
  - But the data is mostly dropped on the floor
- Physical => Digital => *Information*
- Each sensor becomes a network citizen

Wireless Sensor Networks

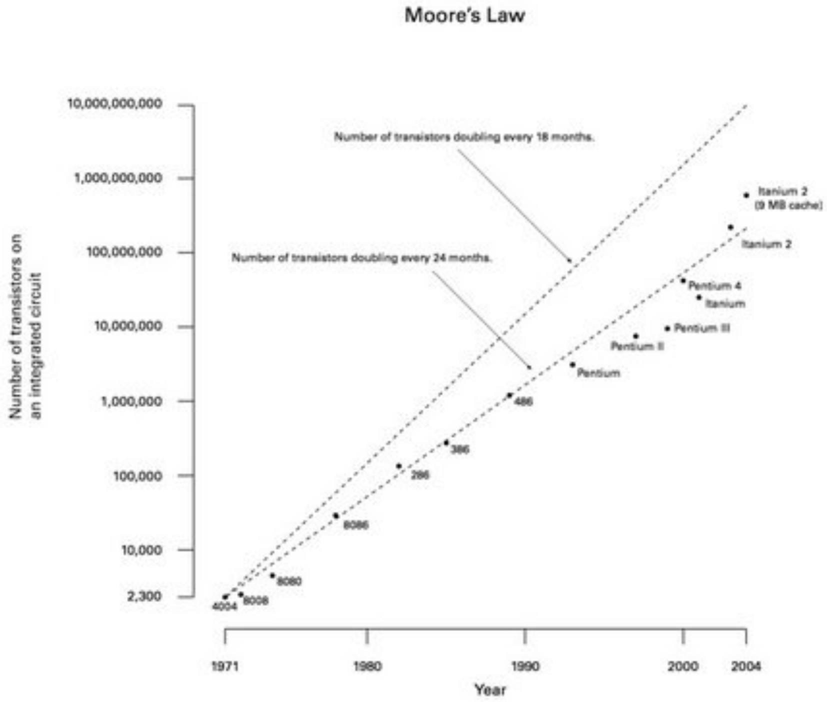
- Network of tiny footprint computers
- Optimized for long life on low power
- Equipped to sense physical data
- Networked using low-power radio
- Function:
  - Sense any measurable parameter
    - Light, motion, chemicals, proximity, biometrics
  - Form network and communicate
    - Automatic meshing and routing over the air
  - Apply user-defined business logic
    - Sampling, summarizing, reporting events
- Form:
  - Mote (Processor, Radio, Storage) + Sensors
  - Embedded Operating System and Networking
  - Router & Gateways towards Enterprise IT systems



Drivers

Moore's Law:

"the complexity of an integrated circuit, with respect to minimum component cost, will double in about 18 months



"Cramming more components onto integrated circuits", *Electronics Magazine*, April 1965.

Recent Developments

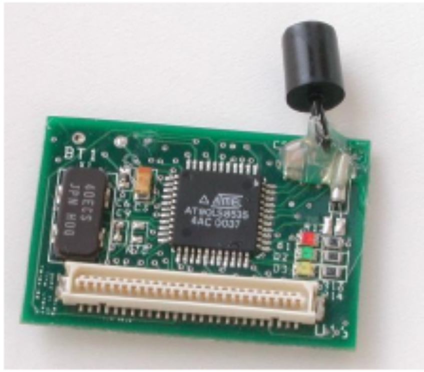
- Cheap and reliable communications:
  - short-range RF, infrared, optical
  - low power
- New interesting sensors
  - light, heat, humidity
  - position, movement, acceleration, vibration
  - chemical presence, biosensor
  - magnetic field, electrical inc. bio-signals (ECG and EEG)
  - RFID
  - acoustic (microphone)

Long-term objective

- Completely integrated
  - one package includes: computation, communication, sensing, actuation, (renewable) power source
  - modular
- Less than a cubic millimeter in volume
- Cheap
- Diverse in design and usage
- Robust
- Main challenge: energy efficiency!

Device evolution

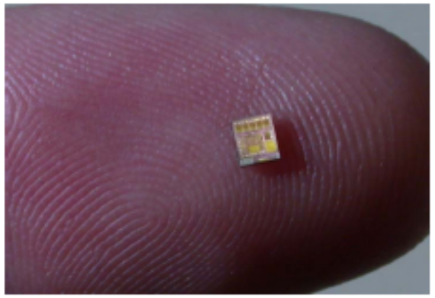
WeC (1999)



René (2000)

DOT (2001)

MICA (2002)

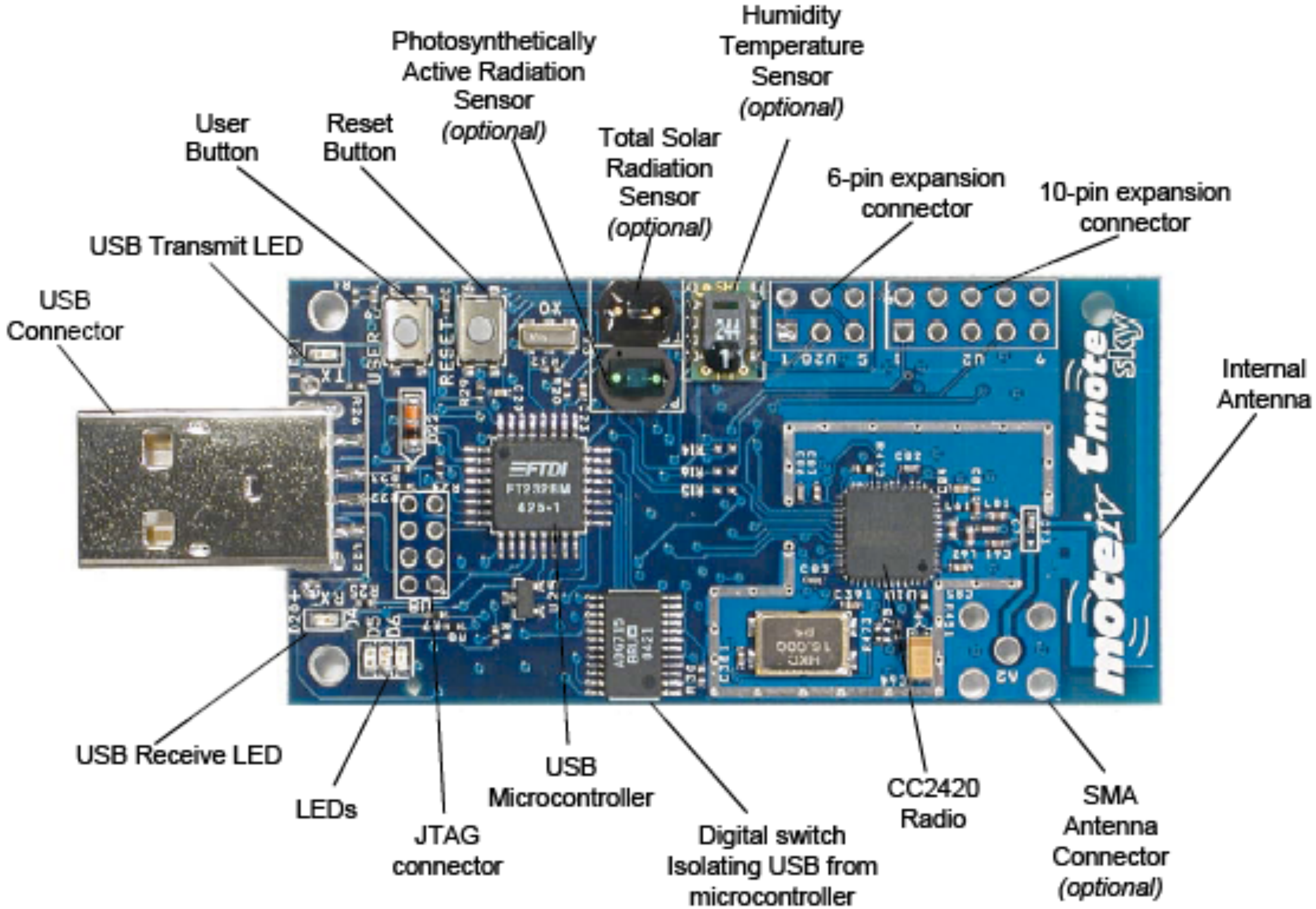


Speck (2003)

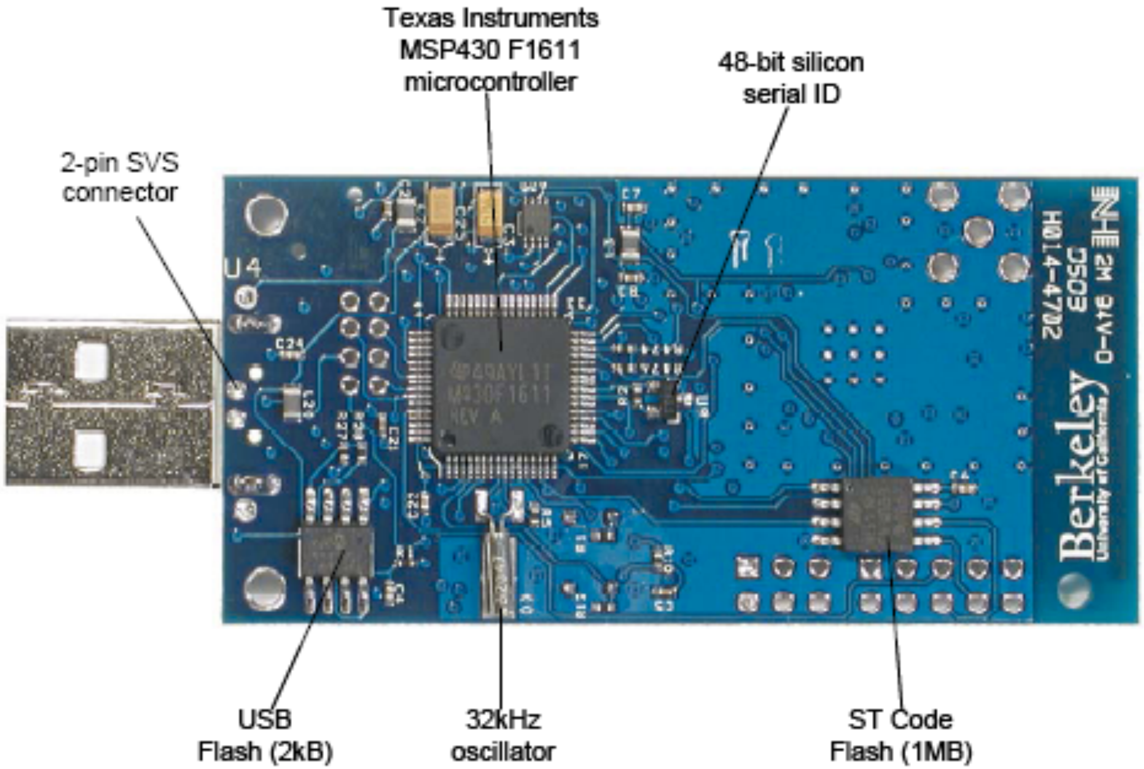
Tmote Sky

- Texas Instruments MSP430
  - 16-bit RISC, 8MHz, 10k RAM, 48k Flash, 128b storage
  - Integrated analog-to-digital converter (12 bit ADC)
- Chipcon wireless transceiver
  - IEEE 802.15.4 (Zigbee) compatible
  - 250kbps at 2.4GHz
- Sensirion SHT11/SHT15 sensor module
  - humidity and temperature
- Hamamatsu light sensors
  - S1087 (photosynthetic)
  - S1087-01 (full visible spectrum)

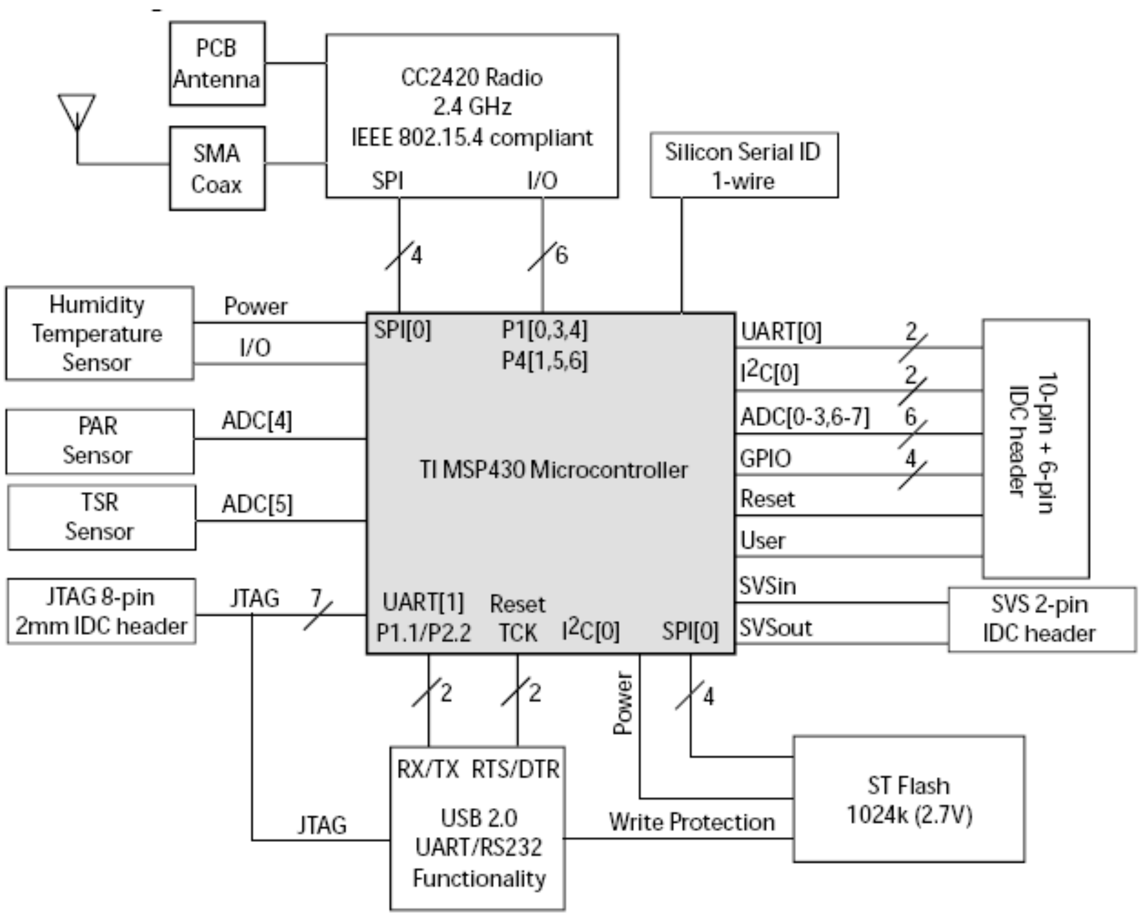
Module layout (top)



Module layout (bottom)

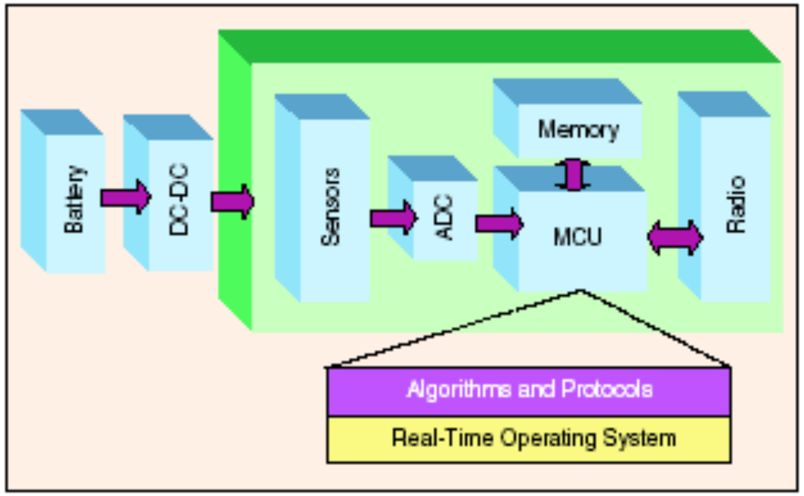


Block diagram



Where does the power go?

- Processing
  - excluding low-level processing for radio, sensors, actuators
- Radio
- Sensors
- Actuators
- Power supply



discussion follows Srivastana tutorial
(check module website)

Sky module characteristics

Current Consumption: MCU on, Radio RX		21.8	23	mA
Current Consumption: MCU on, Radio TX		19.5	21	mA
Current Consumption: MCU on, Radio off		1800	2400	µA
Current Consumption: MCU idle, Radio off		54.5	1200	µA
Current Consumption: MCU standby		5.1	21.0	µA

Need power management to actually exploit energy efficiency:

- idle and sleep modes
- variable voltage
- variable frequency
- in-network storage and processing

Chipcon radio is only a transceiver, and a lot of low-level processing takes place in the main CPU. Contrast this with Wi-Fi radio which will do everything up to MAC and link level encryption in the "radio."

Sensors and power consumption

- Several energy consumption sources
  - transducer
  - front-end processing and signal conditioning
    - analog, digital
  - ADC conversion
- Diversity of sensors: no general conclusions can be drawn
  - Low-power modalities
    - Temperature, light, accelerometer
  - Medium-power modalities
    - Acoustic, magnetic
  - High-power modalities
    - Image, video, chemical

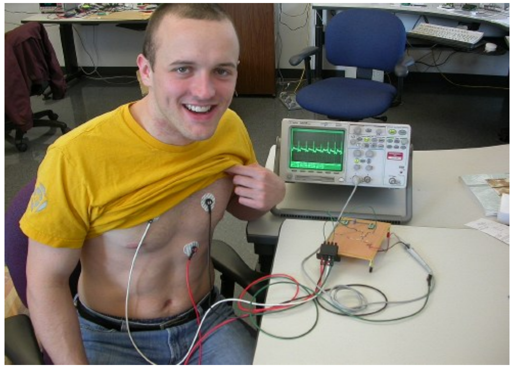
Observations

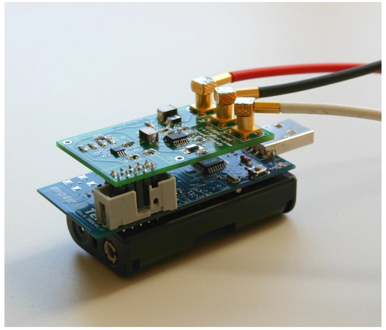
- Radio benefits less from technology improvements than processors
- The relative impact of the communication subsystem on the system energy consumption will grow
- Using low-power components and trading-off unnecessary performance for power savings can have orders of magnitude impact
- Node power consumption is strongly dependent on the operating mode
- At short ranges, the Rx power consumption > T power consumption
- Idle radio consumes almost as much power as radio in Rx mode
- Processor power fairly significant (30-50%) share of overall power
- In many cases, the sensor overhead is negligible

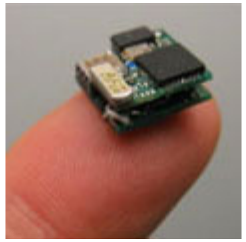
Programming challenges

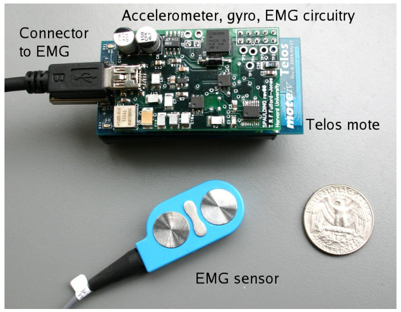
- Driven by interaction with environment
  - Data collection and control, not general purpose computation
  - Reactive, event-driven programming model
- Extremely limited resources
  - Very low cost, size, and power consumption
  - Typical embedded OSs consume hundreds of KB of memory
- Reliability for long-lived applications
  - Apps run for months/years without human intervention
  - Reduce run time errors and complexity
- Soft real-time requirements
  - Few time-critical tasks (sensor acquisition and radio timing)
  - Timing constraints through complete control over app and OS

Medical Monitoring

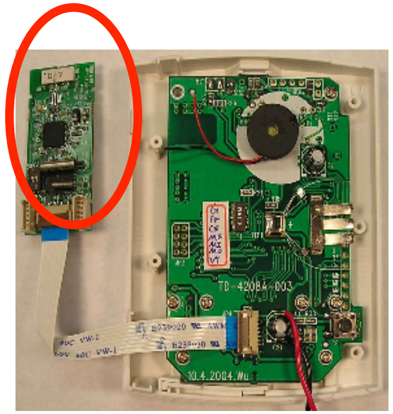








Wireless module



Millennial Net
Wireless Sensor Networking...Anywhere

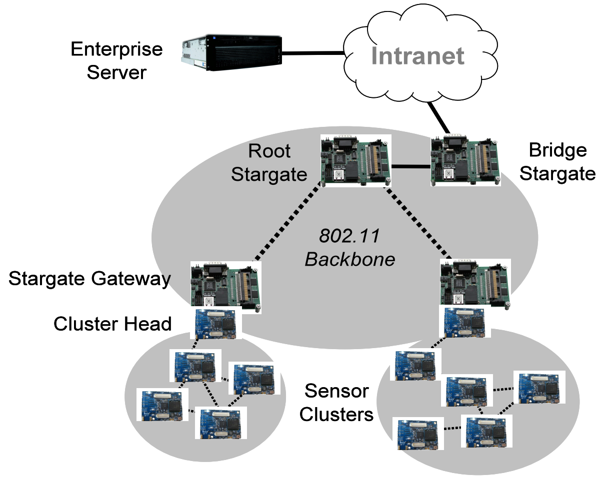
UCIrvine | UNIVERSITY OF CALIFORNIA

londonknowledgelab

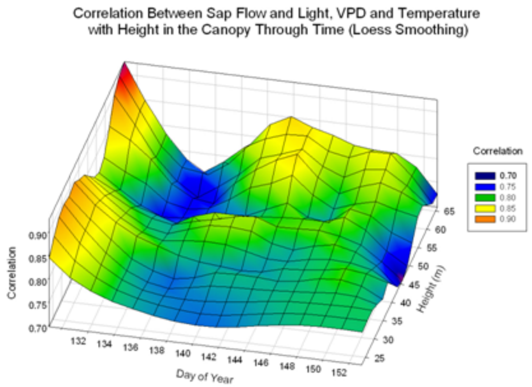
uID Center

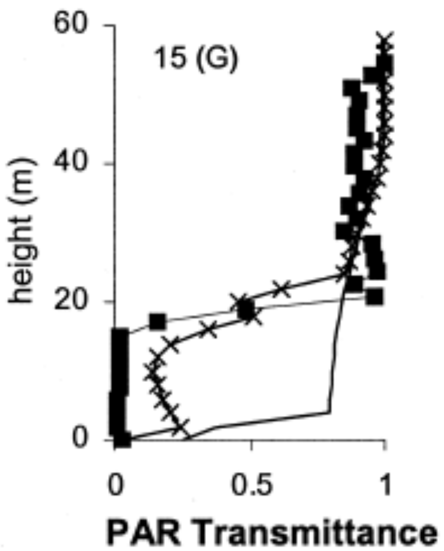
Asset Monitoring

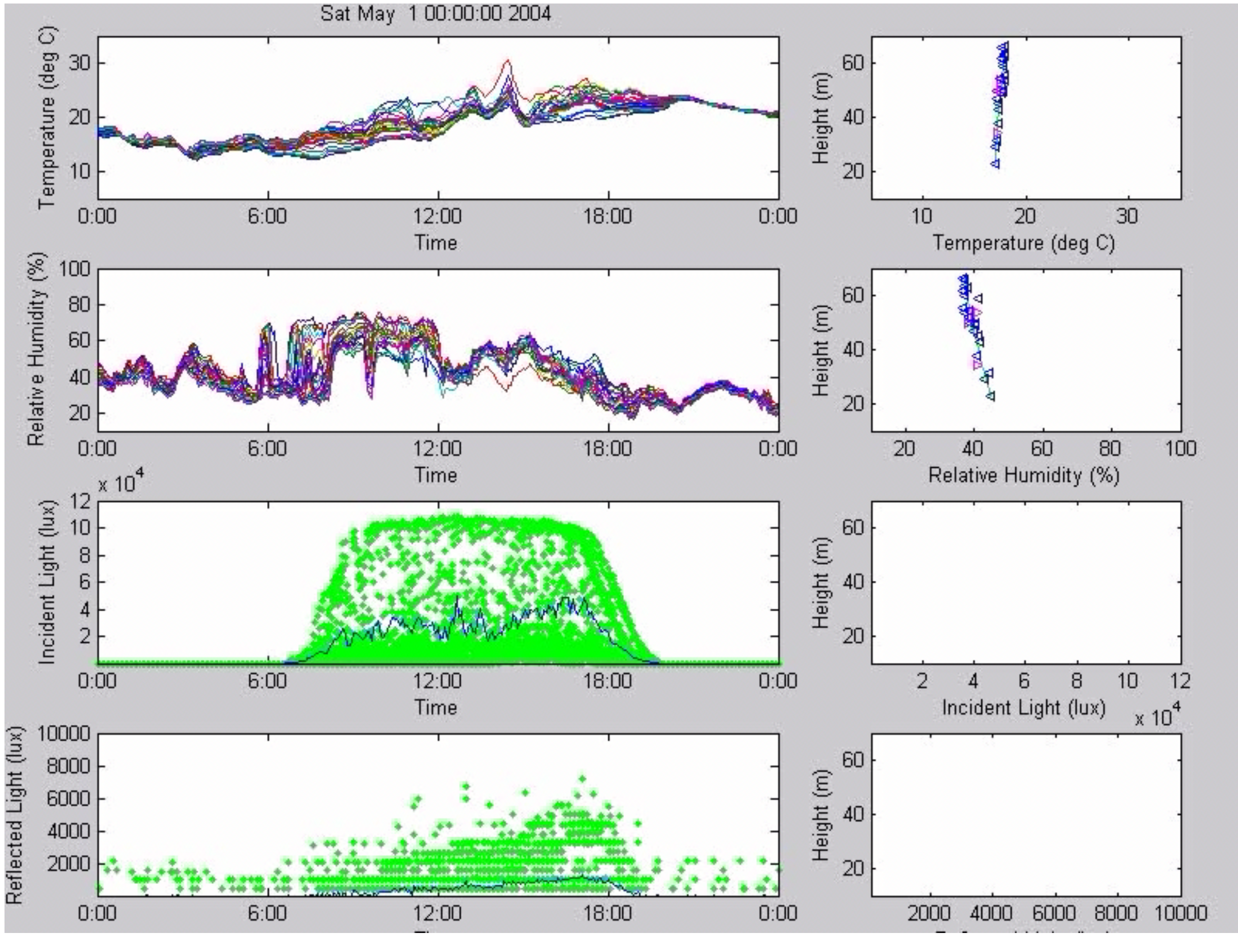
- Goal: Pre-empt equipment failures through non-destructive analysis
- Media Gap: Majority of data is collected by hand
  - Thousands of sense points
- Intel Fab and an Oil Tanker engine room
- Wireless vibration data collection
  - High-speed sampling, reliable bulk transfer
  - Sensor-to-Analysis App flow
  - Overcome interference
  - Support disconnected operation
- Loch Rannoch Network
  - 150 accelerometers
  - 26 motes
  - 4 stargates
  - 1 PC
- Efficient installation and management
  - 36hr install period on tanker
  - No crew intervention



Environmental Monitoring





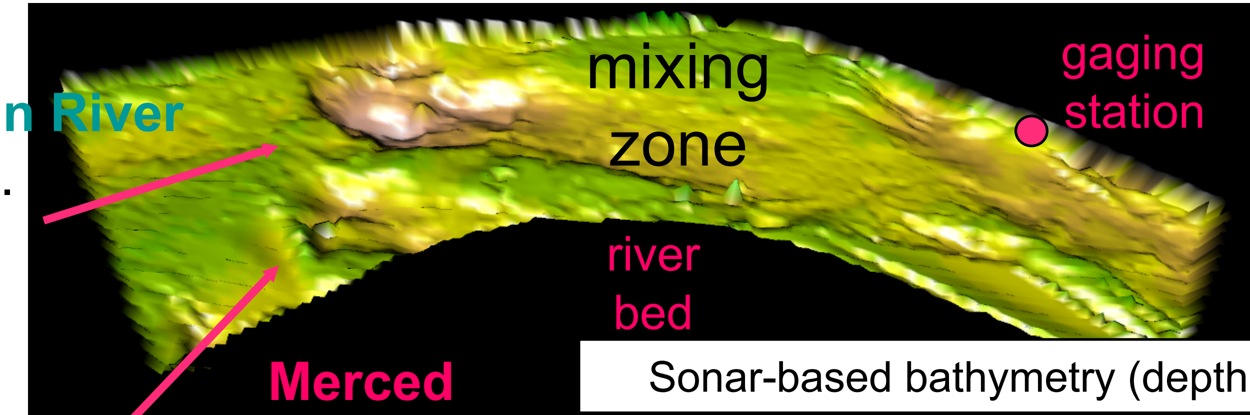


Macroscope in the Redwoods, Tolle et all,
ACM SENSYS 2005

More Environmental Monitoring

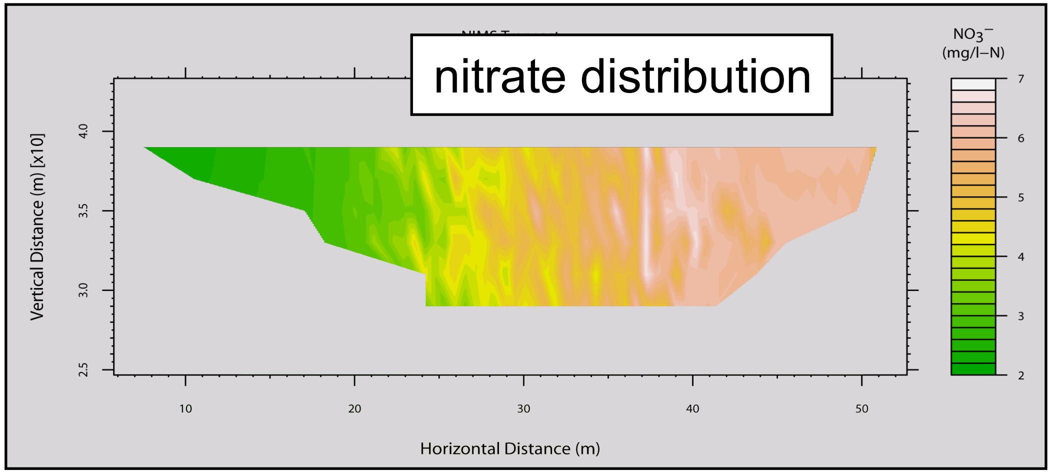
San Joaquin River confluence.

NIMS RD Merced and San Joaquin River Confluence (Harmon, Kaiser, et al)



UCLA USC UCR CALTECH CSU UC MERCED

CENTER FOR EMBEDDED NETWORKED SENSING



londonknowledgelab uID Center

Principle platform

- **NesC**: a C dialect for embedded programming
  - Components, “wired together”
  - Quick commands and asynch events

- **TinyOS**: a set of NesC components
  - hardware components
  - ad-hoc network formation & maintenance
  - time synchronization

Tiny OS facts

- Very small "operating system" for sensor networks
  - Core OS requires 396 bytes of memory
- Component-oriented architecture
  - Set of reusable system components: sensing, communication, timers, etc.
  - No binary kernel - build *app specific* OS from components
- Concurrency based on **tasks** and **events**
  - **Task:** deferred computation, runs to completion, no preemption
  - **Event:** Invoked by module (upcall) or interrupt, may preempt tasks or other events
  - Very low overhead, no threads
- Split-phase operations
  - No blocking operations
  - Long-latency ops (sensing, comm, etc.) are **split phase**
  - Request to execute an operation returns immediately
  - Event signals completion of operation

discussion follows Welsh
check module website

nesC facts

- Dialect of C with support for *components*
  - Components **provide** and **require** interfaces
  - Create application by wiring together components using **configurations**
- Whole-program compilation and analysis
  - nesC compiles entire application into a single C file
  - Compiled to mote binary by back-end C compiler (e.g., gcc)
  - Allows aggressive cross-component inlining
  - Static data-race detection
- Important restrictions
  - No function pointers (makes whole-program analysis difficult)
  - No dynamic memory allocation
  - No dynamic component instantiation/destruction
    - *These static requirements enable analysis and optimization*

nesC interfaces

nesC interfaces are bidirectional

- **Command:** Function call from one component requesting service from another
- **Event:** Function call indicating completion of service by a component
- Grouping commands/events together makes inter-component protocols clear

```
interface Timer {
  command result_t start(char type, uint32_t interval);
  command result_t stop();
  event result_t fired();
}

interface SendMsg {
  command result_t send(TOS_Msg *msg, uint16_t length);
  event result_t sendDone(TOS_Msg *msg, result_t success);
}
```

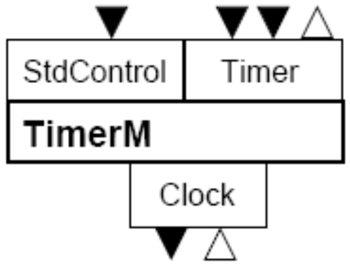
nesC components

- Two types of components
  - **Modules** contain implementation code
  - **Configurations** wire other components together
  - An application is defined with a single top-level configuration

```
module TimerM {
  provides {
    interface StdControl;
    interface Timer;
  }
  uses interface Clock;

} implementation {

  command result_t Timer.start(char type, uint32_t interval) { ... }
  command result_t Timer.stop() { ... }
  event void Clock.tick() { ... }
}
```



nesC configurations

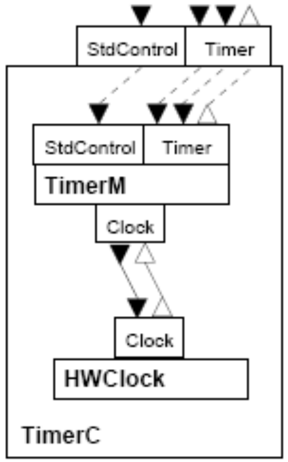
```
configuration TimerC {
  provides {
    interface StdControl;
    interface Timer;
  }

} implementation {

   components TimerM, HWClock;

   // Pass-through: Connect our "provides" to TimerM "provides"
   StdControl = TimerM.StdControl;
   Timer = TimerM.Timer;

   // Normal wiring: Connect "requires" to "provides"
   TimerM.Clock -> HWClock.Clock;
}
```



Concurrency in nesC

- **Tasks** used as deferred computation mechanism
  - Commands and events cannot block
  - Tasks run to completion, scheduled non-preemptively
  - Scheduler may be FIFO, EDF, etc.

```
// Signaled by interrupt handler
event void Receive.receiveMsg(TOS_Msg *msg) {
  if (recv_task_busy) {
    return; // Drop!
  }
  recv_task_busy = TRUE;
  curmsg = msg;
  post recv_task();
}

task void recv_task() {
  // Process curmsg ...
  recv_task_busy = FALSE;
}
```

More on concurrency

- All code is classified as one of two types:
  - **Asynchronous code (AC):** Code reachable from at least one interrupt handler
  - **Synchronous code (SC):** Code reachable only from tasks
- Any update to shared state from AC is a potential data race
  - SC is atomic with respect to other SC (no preemption)
  - Race conditions are shared variables between SC and AC, and AC and AC
  - Compiler detects data races by walking call graph from interrupt handlers

Avoiding a data race

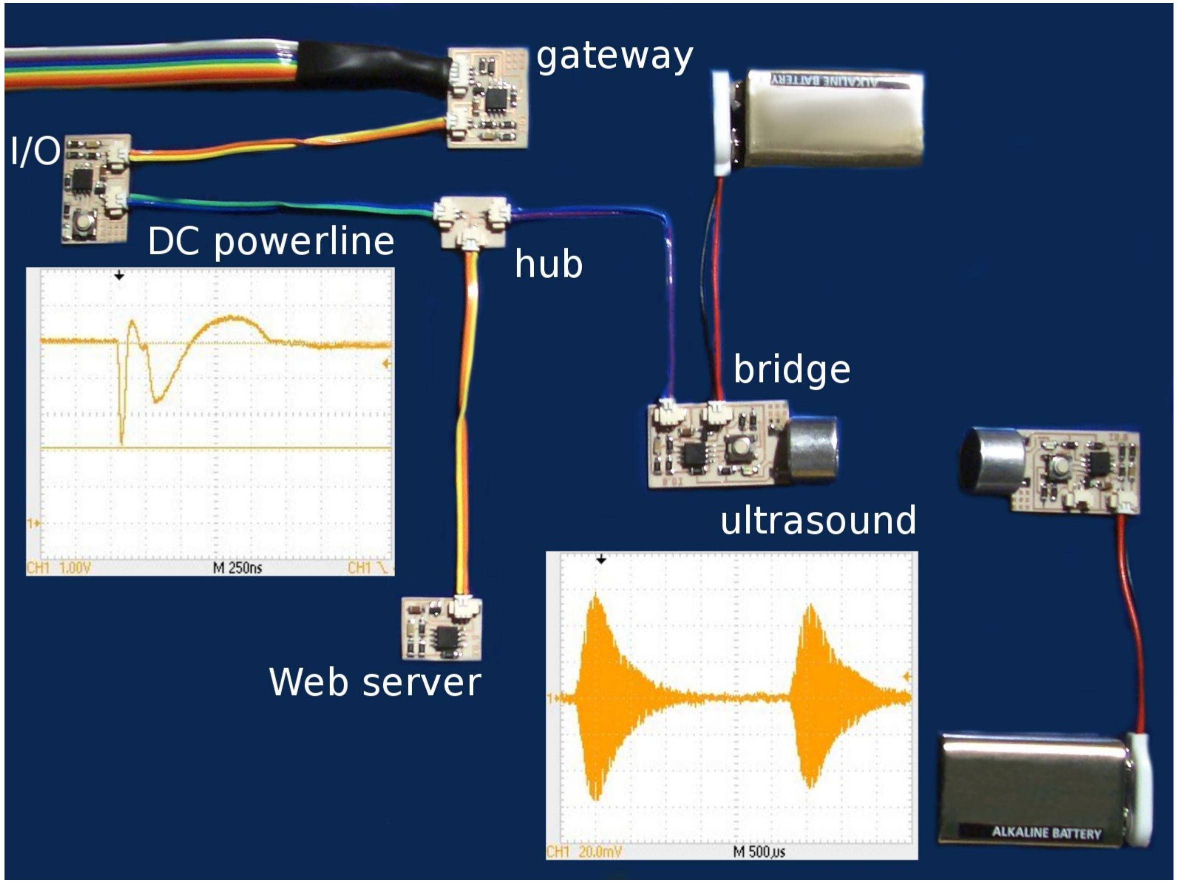
- Two ways to fix a data race
  - Move shared variable access into tasks
  - Use an *atomic section*

  or

  - Short, run-to-completion atomic blocks
  - Currently implemented by disabling interrupts

```
atomic {
  sharedvar = sharedvar+1;
}
```

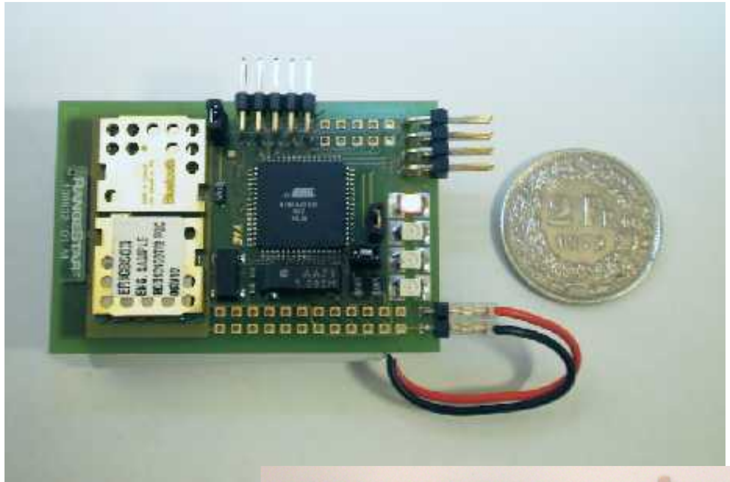
What else is out there?

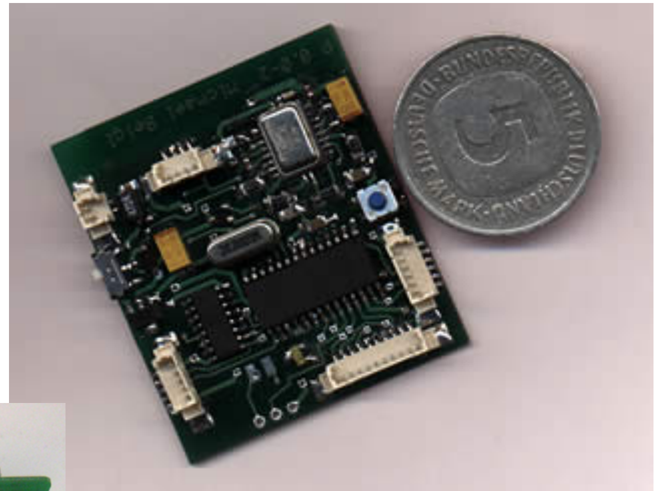


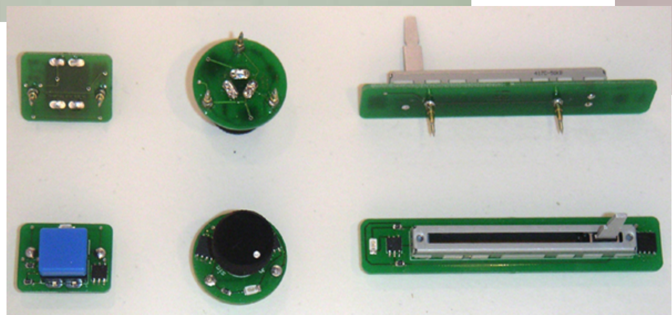
Internet 0 at MIT Centre of Atoms and Bits

http://cba.mit.edu/~neilg

What else is out there?

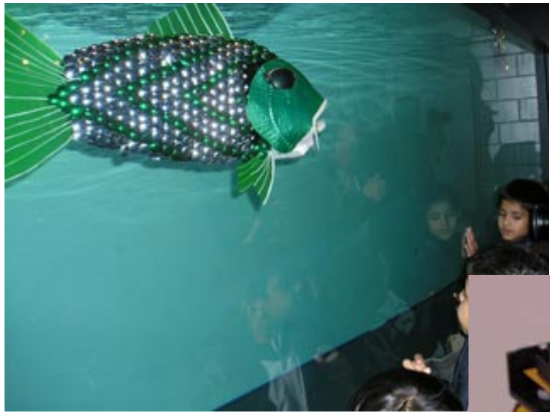


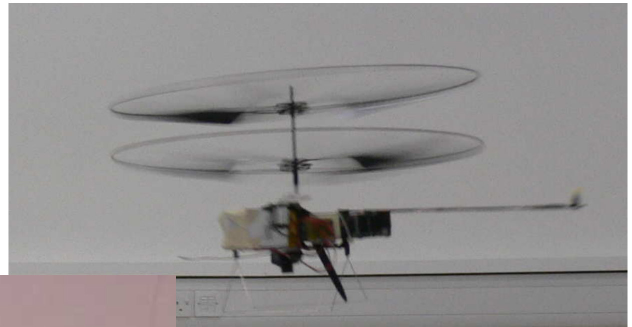


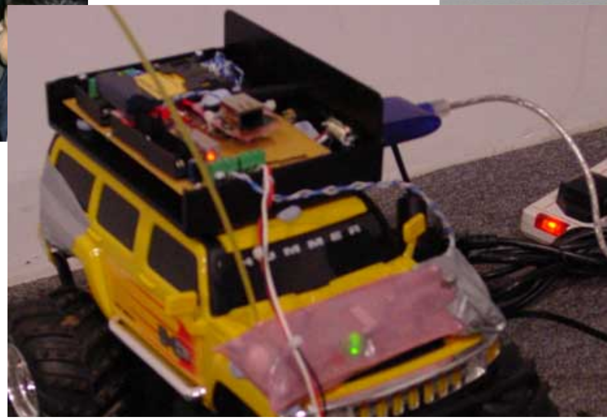


Smart-its http://www.smart-its.org/

What else is out there?



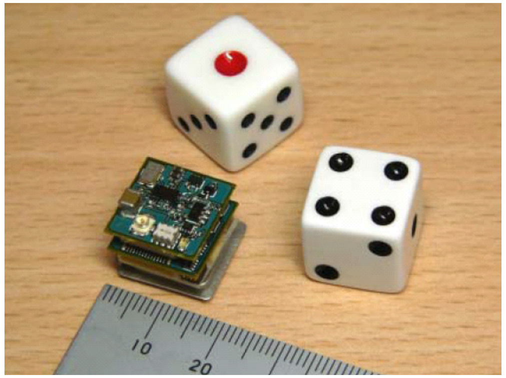




Embedded Linux

gumstix http://www.gumstix.org/

What else is out there?

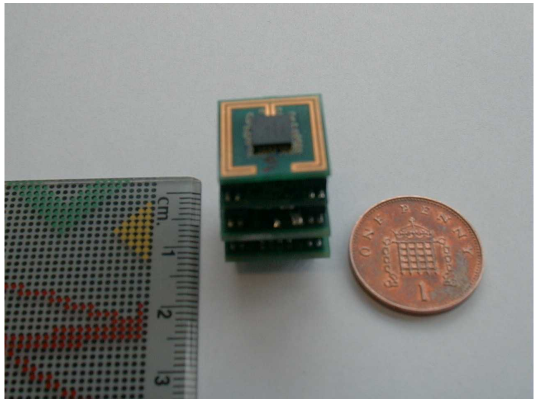


pico-TRON

Hardware-software platform from Japan

Derived from TRON

http://www.t-engine.org/



IMEC Sensor Cube

Very low power, modular design for body area applications

Tiny OS and embedded C

What else is out there?

- Intel Research Wireless Identification and Sensing Platform
- Uses UHF passive RFID techniques to harvest power and communicate (compatible with EPC Gen2)
- Includes most of the components on the Tmore Sky
- Programming must cater for frequent power failure (several times per second possibly
- Native MSP430 C programming

Summary

- Resource constrained devices
  - evolution, architecture, components
  - a detailed example
- Energy efficiency
- Programming primitives in Tiny OS
- Concurrency