TinyOS & nesC

Chapter X

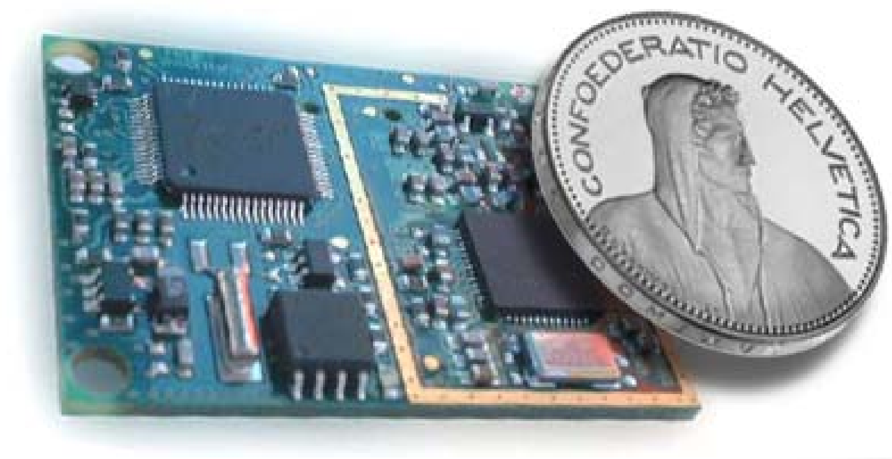
ETH

Eidgenössische Technische Hochschule Zürich

Swiss Federal Institute of Technology Zurich

Sensor Nodes

- System Constraints
  - Slow CPU
  - Little memory
  - Short-range radio
  - Battery powered

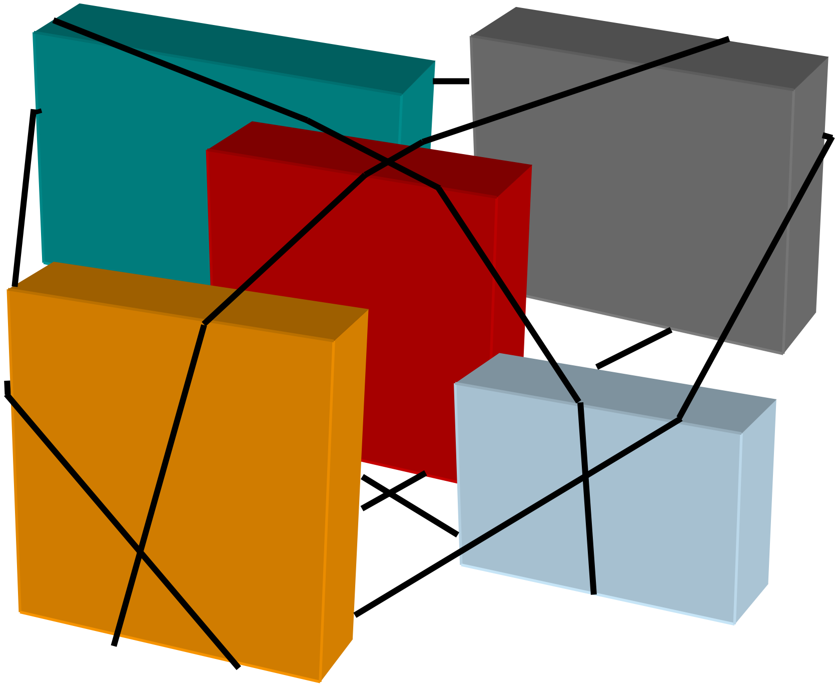


Operating System Requirements

- Measure real-world phenomena
  - Event-driven architecture
- Resource constraints
  - Hurry up and sleep!
- Adapt to changing technologies
  - Modularity & re-use
- Applications spread over many small nodes
  - Communication is fundamental
- Inaccessible location, critical operation
  - Robustness

TinyOS Platform

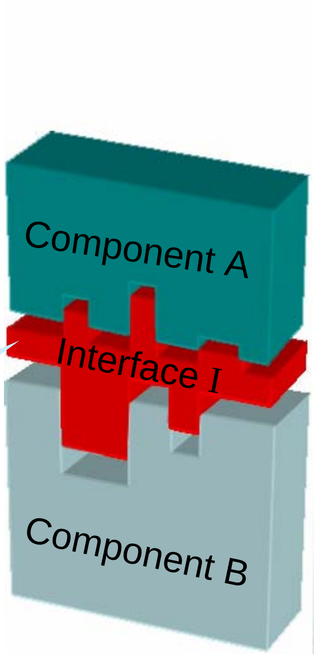
- TinyOS consists of a scheduler & graph of components



Programming Model

provides „hooks“ for component wiring

- Separate construction and composition
- Programs are built out of components specified by an interface
- Two types of components
  - Modules: Implement behavior
  - Configurations: Wire components together
- Components use and provide interfaces

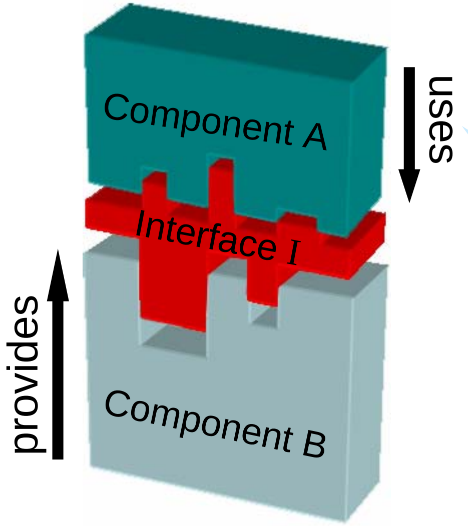


Interfaces are bidirectional

Programming Model

- Interfaces contain definitions of
  - Commands
  - Events

- Components implement the events they use and the commands they provide.



must implement commands, can signal events

can call commands, must implement events

Programming Model

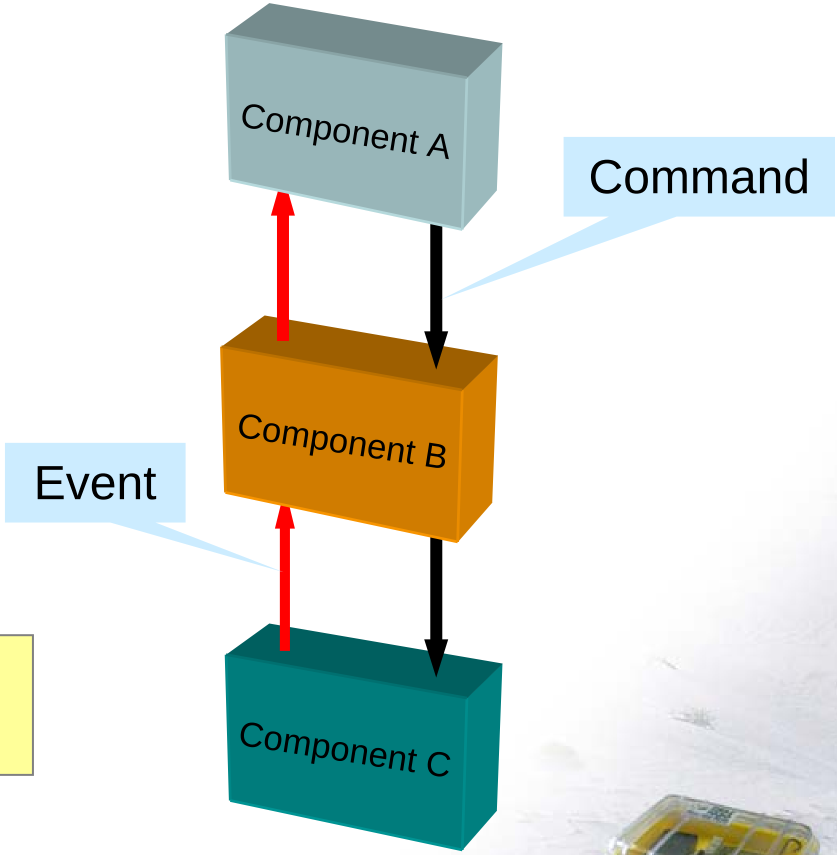
- Components are wired together by connecting interface users with interface providers.

- Commands flow downwards
  - Control returns to caller

- Events flow upwards
  - Control returns to signaler

- Commands are non-blocking requests.

Modular construction kit



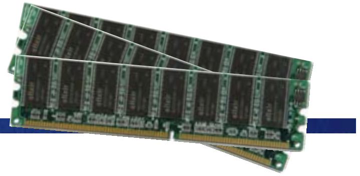
Concurrency Model

Actually single threaded!

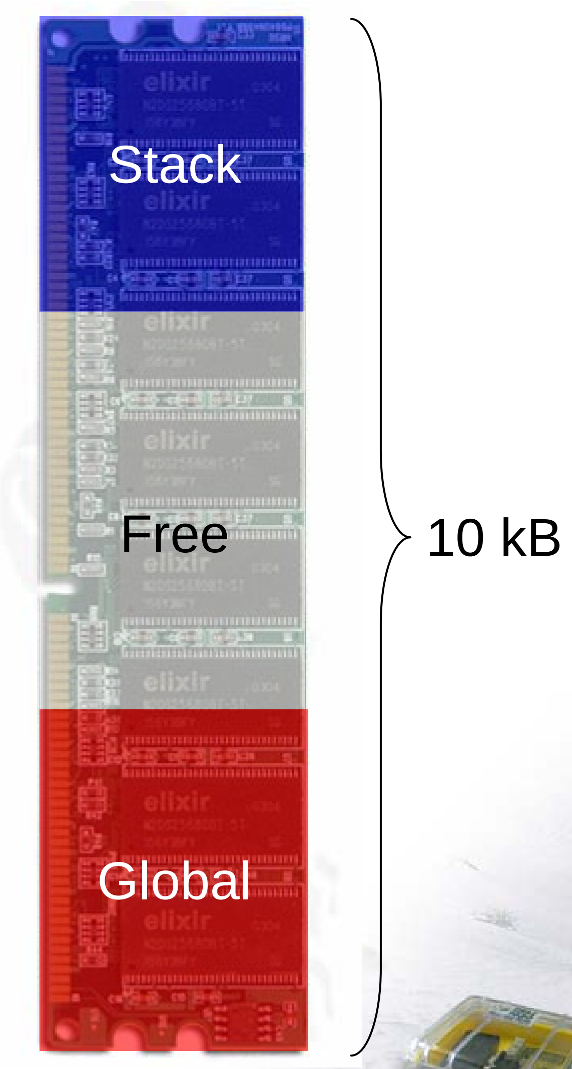
- Coarse-grained concurrency only
  - Implemented via tasks
- Tasks run sequentially by TinyOS scheduler
  - “Multi-threading” is done by the programmer
  - Atomic with respect to other tasks (single threaded)
  - Longer background processing jobs
- Events (interrupts)
  - Time critical
  - Preempt tasks
  - Short duration (hand off computation to tasks if needed)

Note that “event” is overloaded

Memory Model



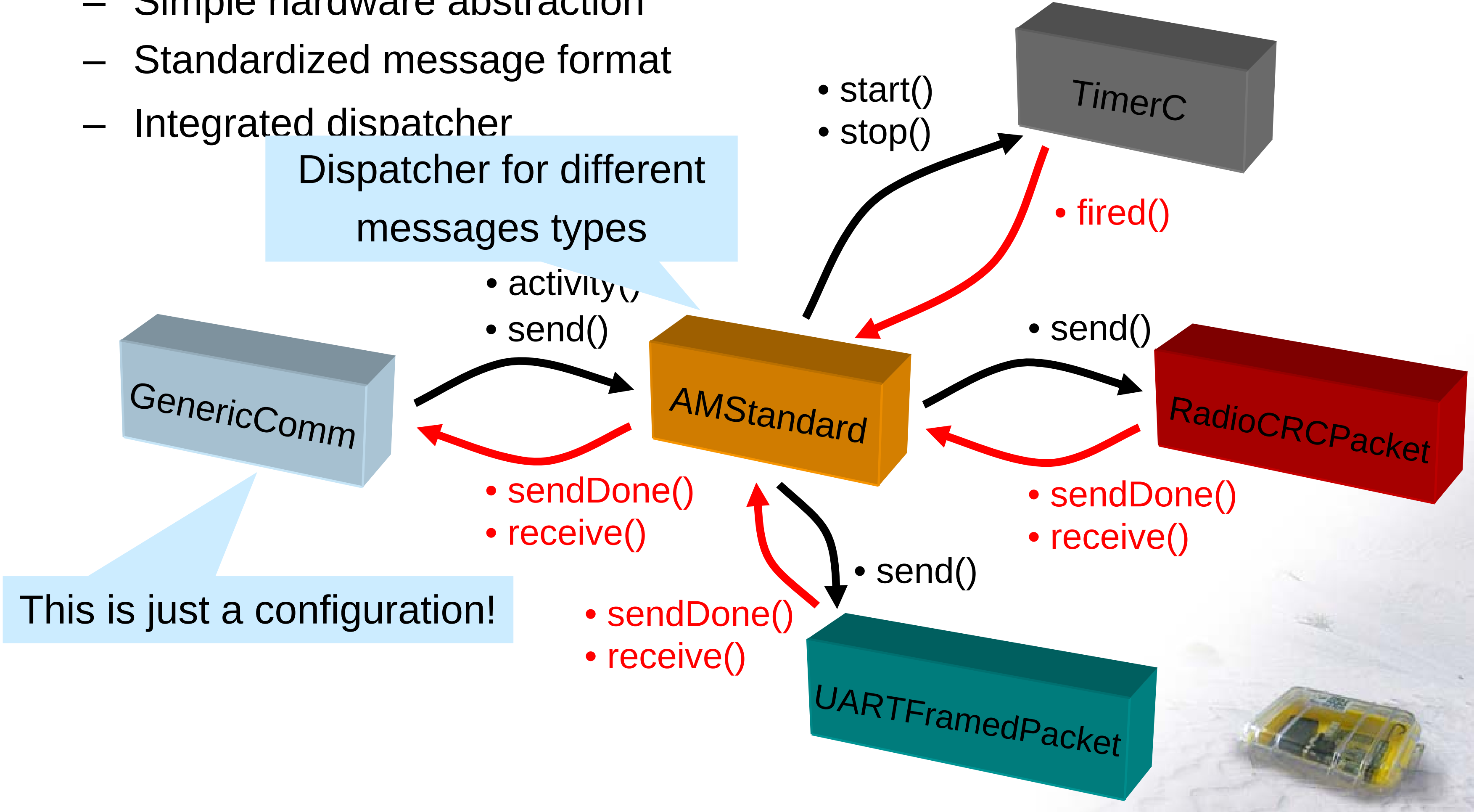
- Static memory allocation
  - No heap (malloc)
  - No function pointers
- Global variables
  - One frame per component
- Local variables
  - Declared within a method
  - Saved on the stack



- Conserve memory
- Use pointers, don't copy buffers

Network Stack

- Ready-to-use communication framework
  - Simple hardware abstraction
  - Standardized message format
  - Integrated dispatcher



Dispatcher for different messages types

This is just a configuration!

TinyOS Distribution

- TinyOS is distributed in source code
  - nesC as programming language
- nesC
  - Dialect of C
  - Embodies the structuring concepts and execution model of TinyOS
    - Module, configuration, interface
    - Tasks, calls, signals
  - Pre-processor producing C code
- nesC limitations
  - No dynamic memory allocation
  - No function pointers

nesC – Hello World

All involved components

```
configuration Blink {
}
implementation {
  components Main,BlinkM,TimerC,LedsC;

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

  BlinkM.Timer -> TimerC;
  BlinkM.Leds -> LedsC;
}
```

Wiring the components

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}
implementation {
  …
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000);
  }

  task void processing() {
    call Leds.redToggle();
  }

  event result_t Timer.fired() {
    post processing();
    return SUCCESS;
  }
}
```

Timer fires every second

Schedule the actual computation