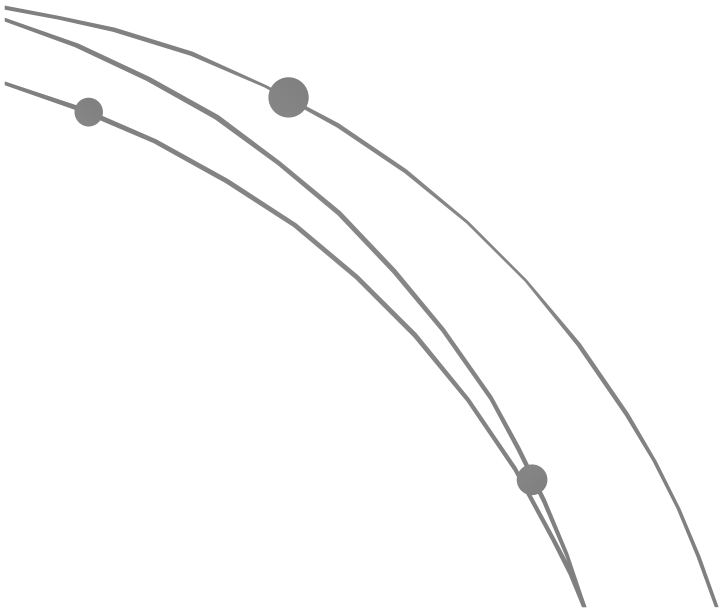
System Architecture Directions for Networked Sensors

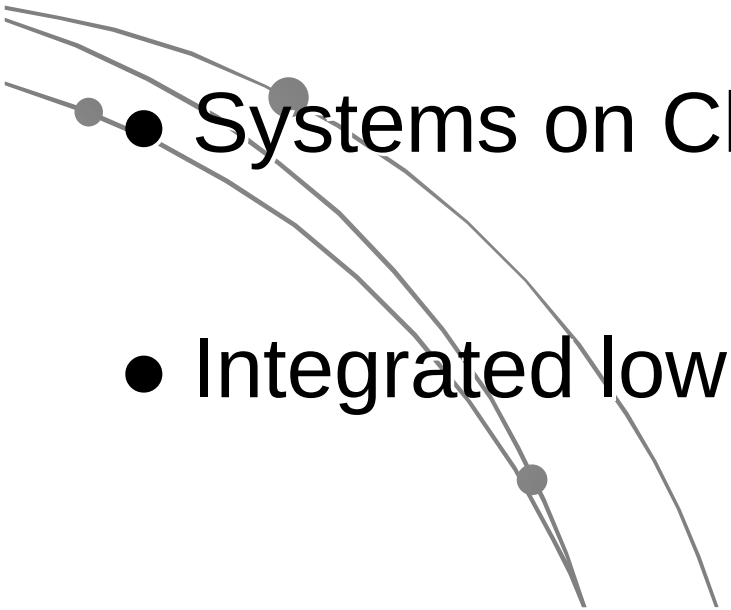
Jason Hill et. al.



A Presentation by
Dhyanesh Narayanan
MS, CS (Systems)

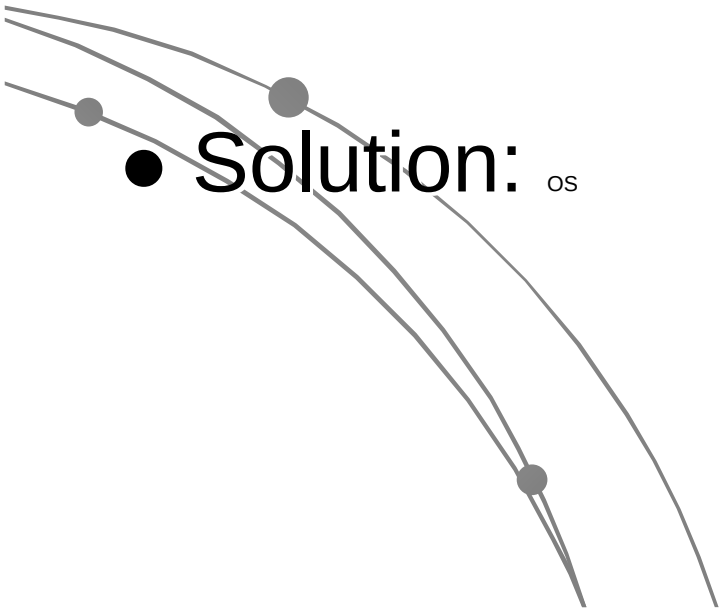
Sensor Networks – Key Enablers

- Moore’s Law:
  - More CPU
  - Less Size
  - Less Cost
- Systems on Chip (SoC)
- Integrated low-power communication



Sensor Networks – Need

- Overall System Architecture
- Software Platform to support & connect Sensors
- Solution: $_{OS}$



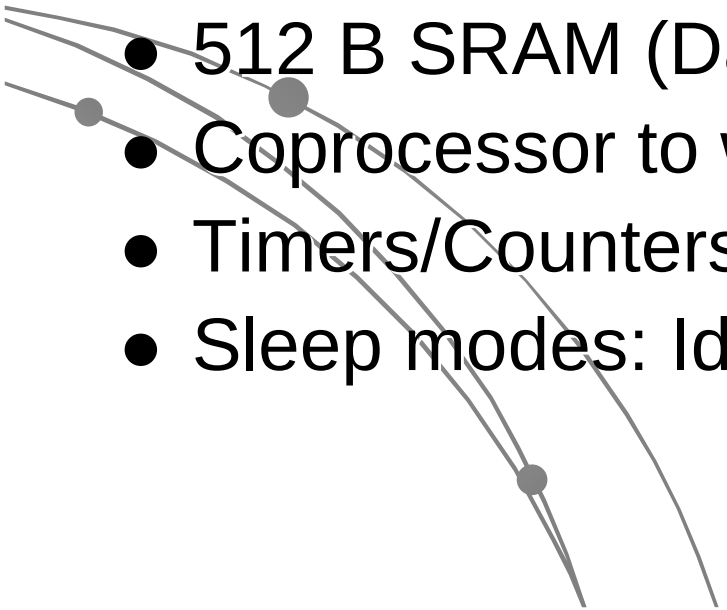
TinyOS ☺

Sensor Systems Software: Design Constraints

- Small physical size/Low power consumption
- Concurrency-intensive operation
- Limited Physical Parallelism & Controller Hierarchy
- Diversity in Design & Usage/Robust Operation

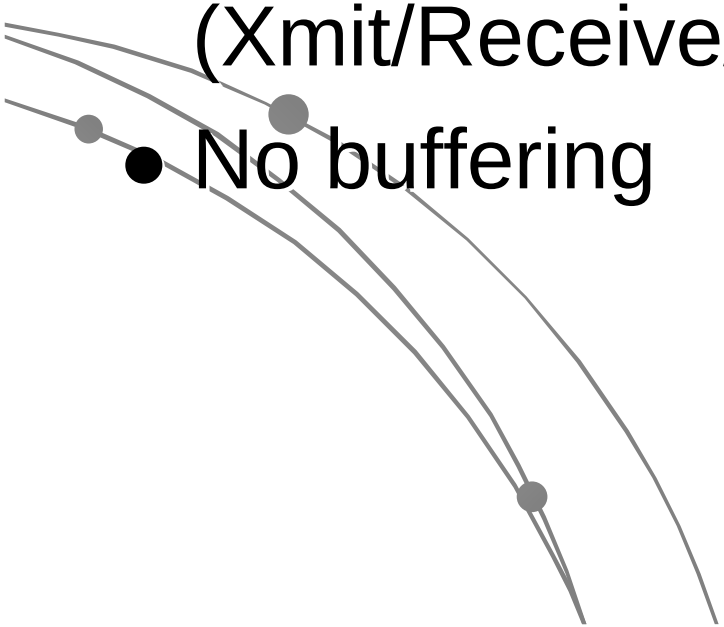
Sample Design Point: Processor

- 8-bit Harvard architecture
- 16-bit addresses
- 32X8-bit General Purpose Registers
- 4MHz / 3.0 V
- 8 KB Flash (Program Memory)
- 512 B SRAM (Data Memory)
- Coprocessor to write to instruction memory
- Timers/Counters to fire interrupts
- Sleep modes: Idle, Power Down, Power Save



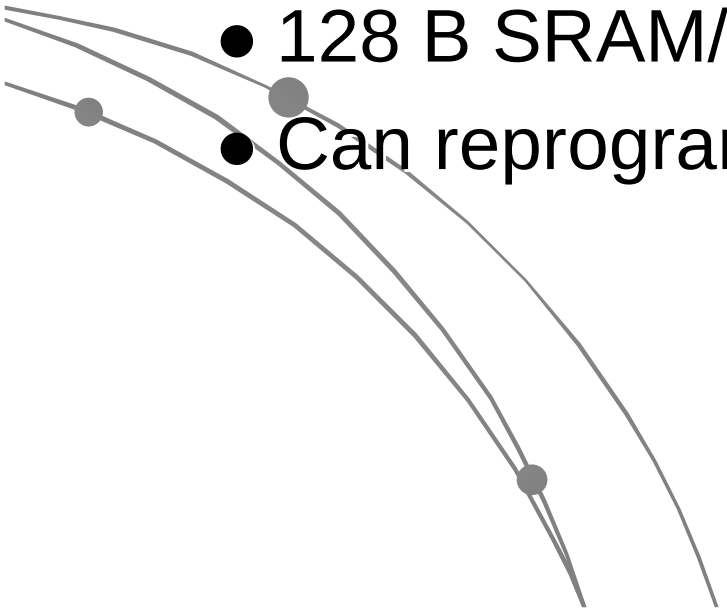
Sample Design Point: Radio

- Async IO device with hard real-time constraints
- Speeds up to 19.2 Kbps
- Control Signals configure mode (Xmit/Receive/Power-off)
- No buffering



Sample Design Point: Others

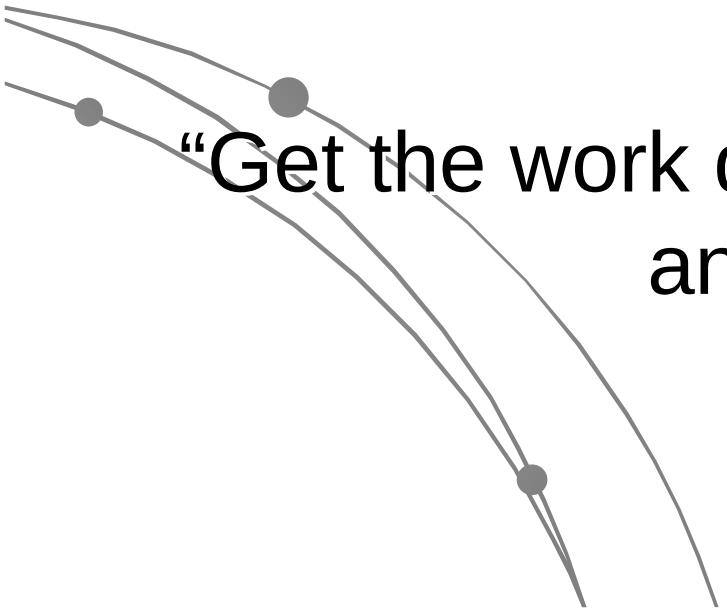
- Temperature Sensor
- Serial Port
- Coprocessor
  - 2 KB Flash (instruction memory)
  - 128 B SRAM/EEPROM
  - Can reprogram the main microcontroller



Power

- Key: Make unused components inactive whenever possible

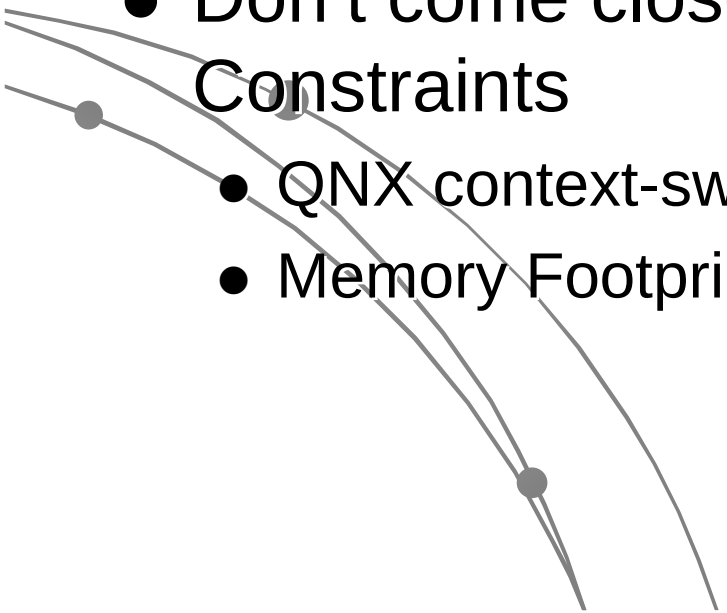
- Embrace the grad-student philosophy:



"Get the work done as quickly as possible and go to sleep" ☺

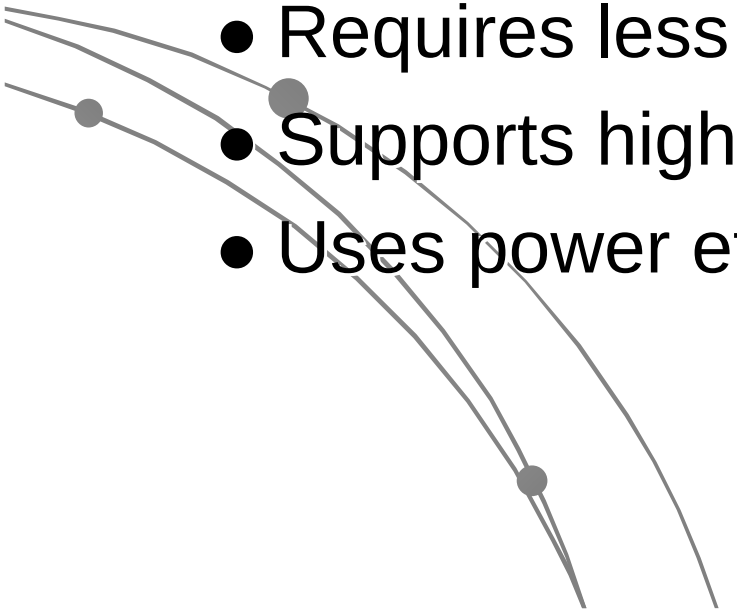
Why RTOS won’t work

- VxWorks, WindowsCE, PalmOS, QNX, …
- Based on µ-kernels
- Environment tries to mimic desktop systems
- POSIX compatible thread packages
- Don’t come close to meeting Sensor Design Constraints
  - QNX context-switch takes >2.4K cycles [33MHz CPU]
  - Memory Footprint of VxWorks ~ 100 KB



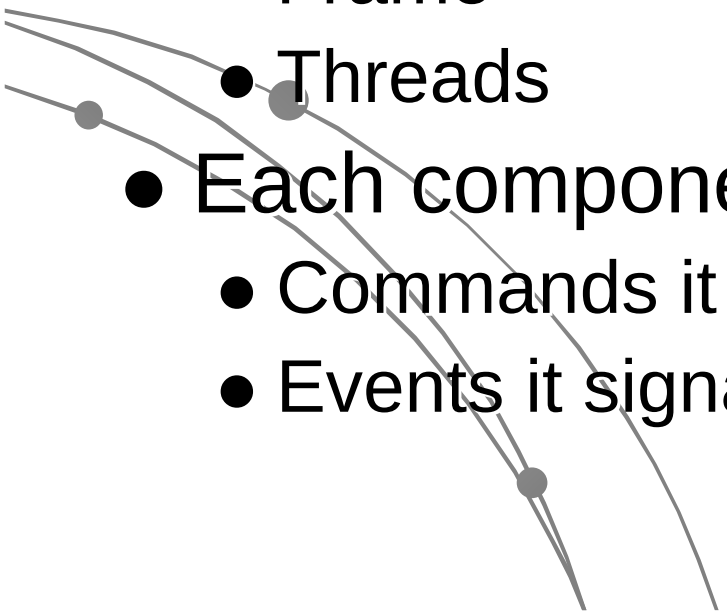
Solution: TinyOS

- Manages size/power constrained hardware effectively
- Supports concurrency-intensive operation
- Event-based model
  - Requires less space
  - Supports high rate of multi-tasking
  - Uses power efficiently



TinyOS Design

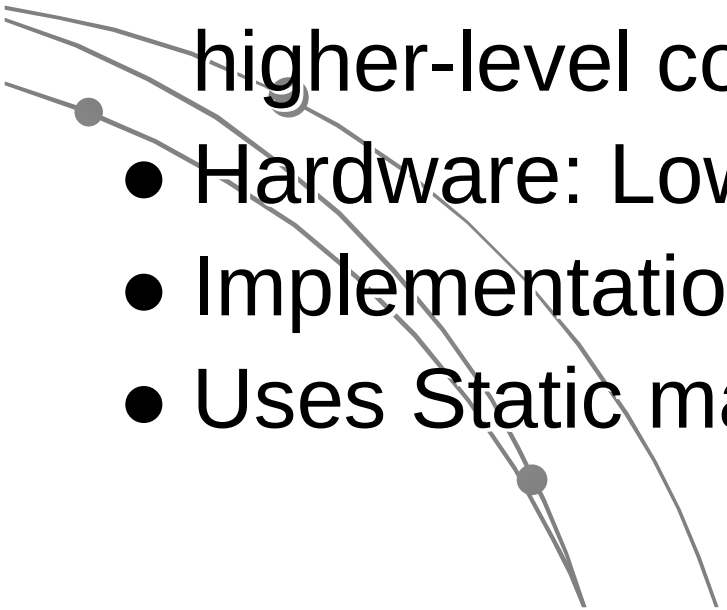
- TinyScheduler
- Components
  - Command Handlers
  - Event Handlers
  - Frame
  - Threads
- Each component declares:
  - Commands it uses
  - Events it signals



Used to compose components in a per-application config

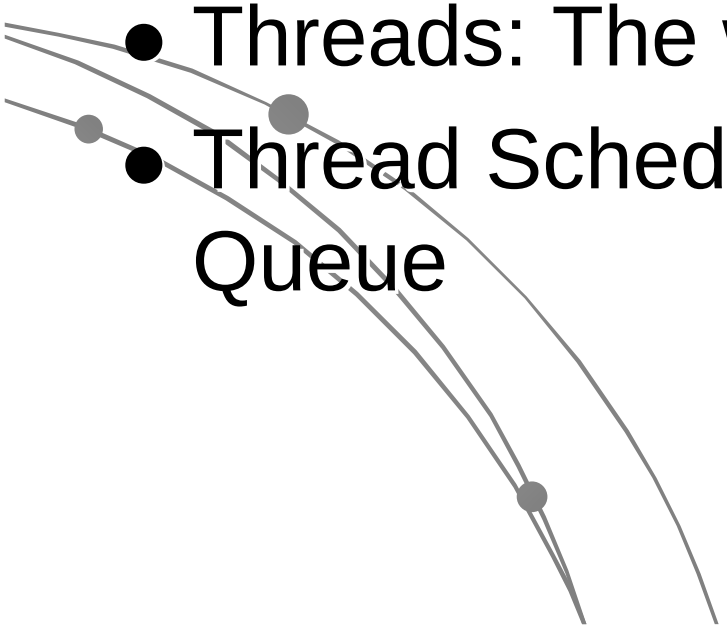
TinyOS Component Layering

- Module Composition creates layers of components
- Higher-level components issue *commands* to lower-level components
- Lower-level components signal *events* to higher-level components
- Hardware: Lowest level component
- Implementation: C
- Uses Static malloc



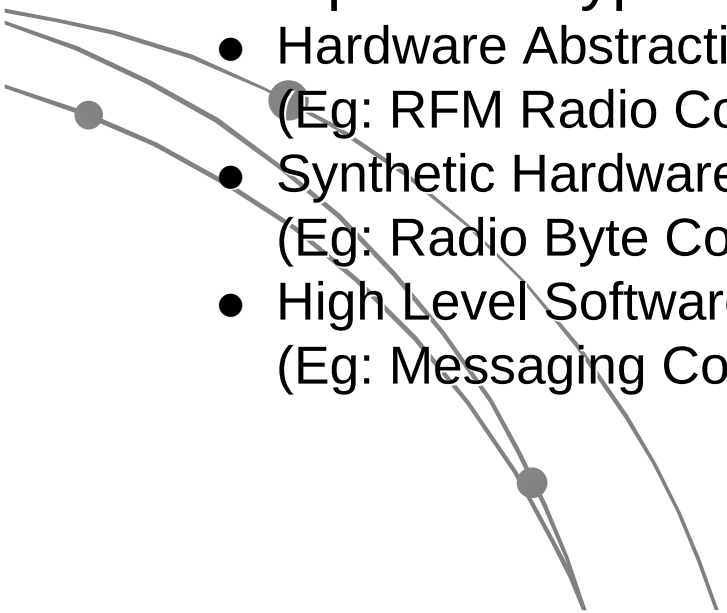
TinyOS Components

- Commands: Non-blocking requests to lower level components
- Event Handlers: Invoked to deal with direct/indirect hardware events
- Threads: The workhorses
- Thread Scheduler: Power-aware FIFO Queue



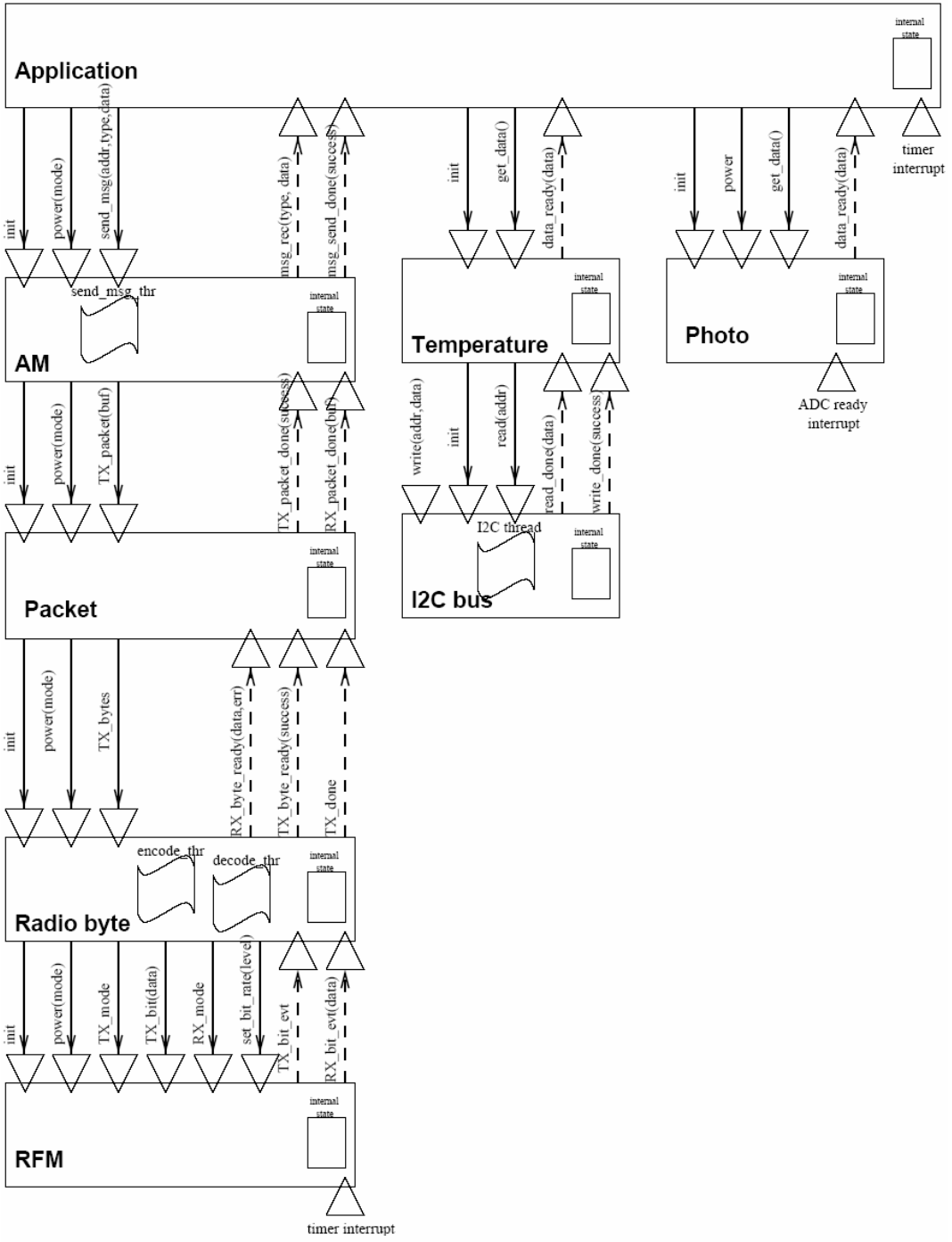
TinyOS Components

- Components describe what they offer and what they need: easy to lego-fit
- Inter-Component Communication: Low overhead function call
- Component Types
  - Hardware Abstraction Components (Eg: RFM Radio Component)
  - Synthetic Hardware Component (Eg: Radio Byte Component)
  - High Level Software Components (Eg: Messaging Component)



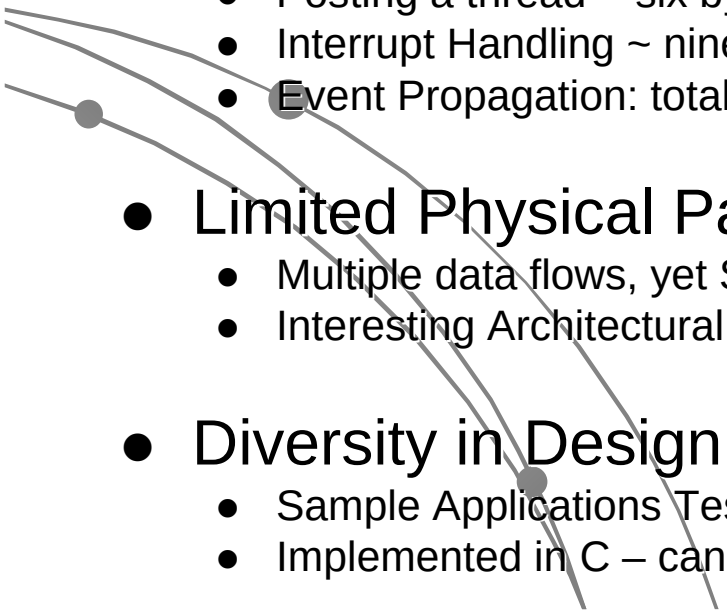
Example

- Three I/O Devices
  - Network
  - Light Sensor
  - Temperature Sensor
- Periodic timer-event triggers data collection
- Xmit: Each component is a data-drain and funnels the data out
- Recv: Each component is a data-pump and pulls the data in



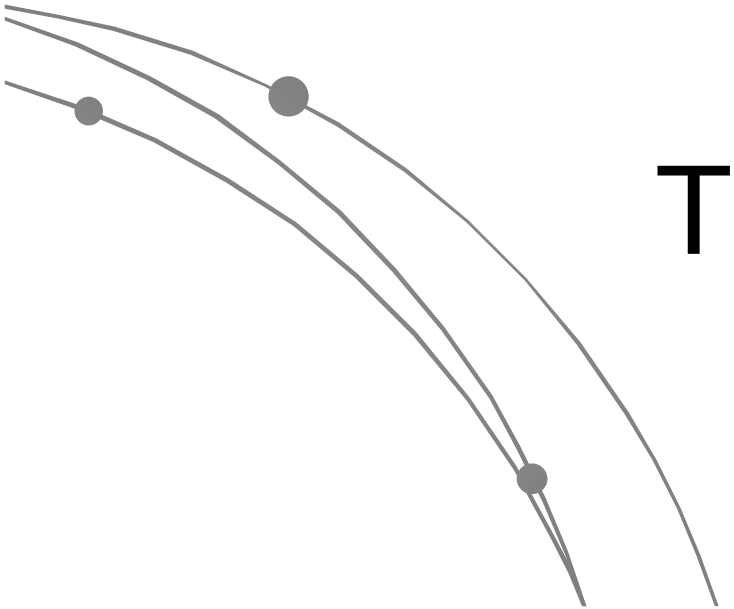
Evaluation
(in the light of Design Constraints)

- Small physical size/Low power consumption
  - Scheduler 178 Bytes
  - Network Sensor App 3 KB Instr Memory
  - Scheduler Data Size 16 Bytes (= 3% of available data memory)
- Concurrency-intensive operation
  - High throughput event-model
  - Cost of propagating an event ~ one byte-copy
  - Posting a thread ~ six byte-copies
  - Interrupt Handling ~ nine byte-copies
  - Event Propagation: total delay up a 5-layer stack 40µs (~ 80 instructions)
- Limited Physical Parallelism & Controller Hierarchy
  - Multiple data flows, yet Single Microcontroller
  - Interesting Architectural Implication
- Diversity in Design & Usage/Robust Operation
  - Sample Applications Tested
  - Implemented in C – can target multiple CPU Architectures



“May the (Tiny) OS be with you”

- Sensor Systems Jedi
Sensor Wars, Revenge of the Chip



Thank You

☺