# *Contiki – a Lightweight and Flexible Operating System for Tiny Networked Sensors*

***Adam Dunkels, Björn Grönvall, Thiemo Voigt***

***Swedish Institute of Computer Science***

***IEEE EmNetS-I, 16 November 2004***

SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

# *Sensor OS trade-offs:*

## *static* vs *dynamic*

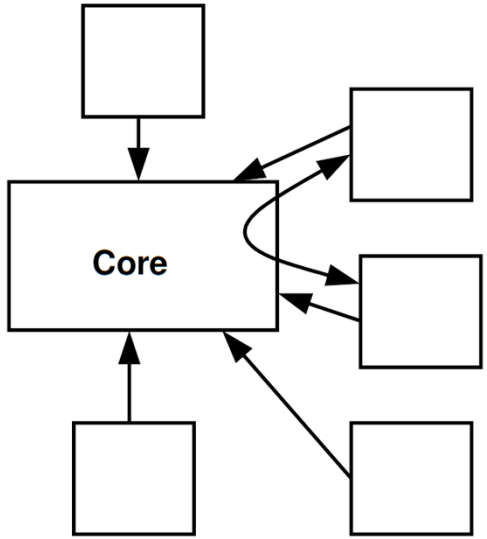
## *event-driven* vs *multi-threaded*

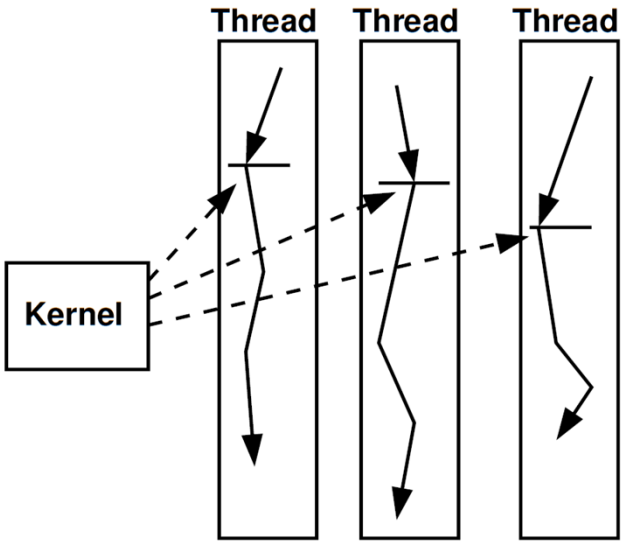
# *What we have done*

- **Contiki** – an OS for sensor network nodes
- Ported Contiki to a number of platforms
  - MSP430, AVR, HC12, Z80, 6502, x86, ...
  - Simulation environment for BSD/Linux/Windows
- Built a few applications for experimental network deployments

# *Contributions*

- **Dynamic loading** of programs
  - Selective reprogramming
  - Static vs dynamic linking
- **Concurrency** management mechanisms
  - Events vs threads
  - Trade-offs: preemption, size



# Contiki design target

- "Mote"-class device
  - 10-100 kilobytes of code ROM
  - 1-10 kilobytes of RAM
  - Communication (radio)
- ESB from FU Berlin
  - MSP430, 2k RAM, 60k ROM

# Contiki size (bytes)

| Module | Code MSP430 | Code AVR | RAM |
|---|---|---|---|
| Kernel | 810 | 1044 | $10 + e + p$ |
| Program loader | 658 | - | 8 |
| Multi-threading library | 582 | 678 | $8 + s$ |
| Timer library | 60 | 90 | 0 |
| Memory manager | 170 | 226 | 0 |
| | | | |
| Event log replicator | 1656 | 1934 | 200 |
| µIP TCP/IP stack | 4146 | 5218 | $18 + b$ |

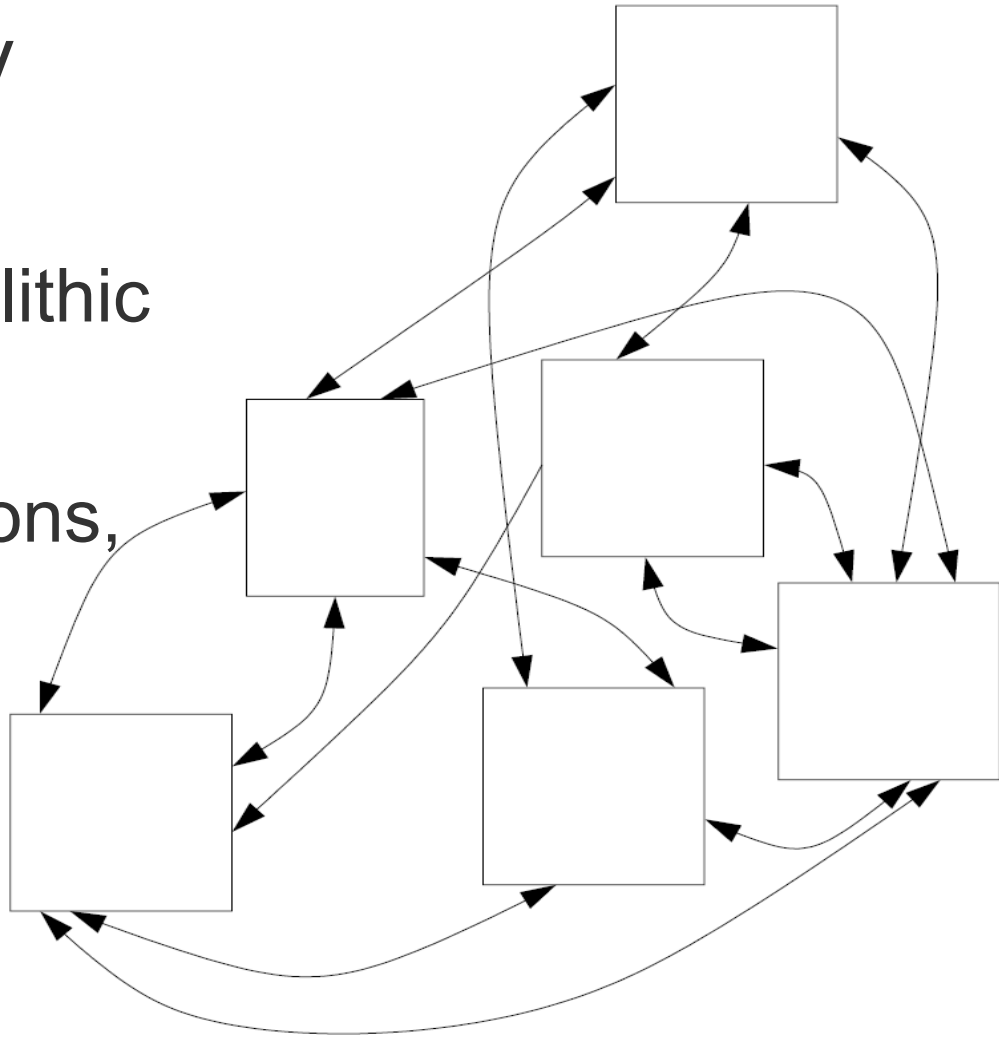
# *Run-time reprogramming and loadable programs*

# *Reprogramming sensor nodes*

- **Software development** for sensor nets
  - Need to reprogram many nodes quite often
- Utilize radio for reprogramming
  - Radio inherently **broadcast**
- Reprogram many nodes at once
  - **Much** faster than firmware download via cable or programming adapter
- Reprogram deployed networks

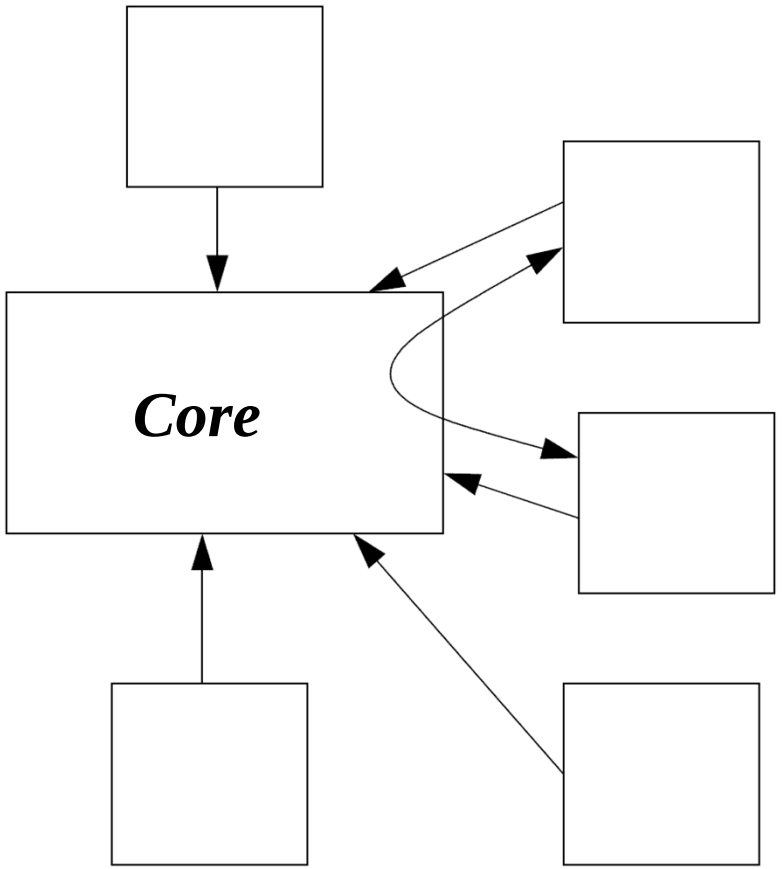
# *Traditional systems: entire system a monolithic binary*

- Most systems statically linked at compile-time
  - Entire system is a monolithic binary
  - Compile-time optimizations, analysis possible
  - Makes code smaller
- **But: hard to change**
  - **Must download entire system**



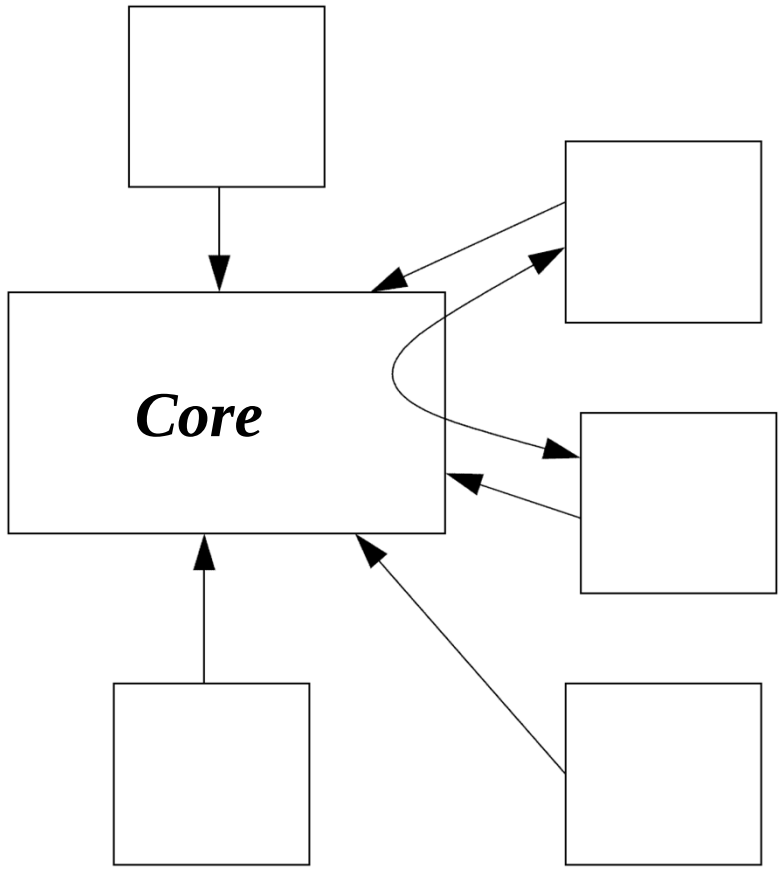
# Contiki: loadable programs

- Contiki: one-way dependencies
  - Core resident in memory
    - Language run-time, communication
  - Programs “know” the core
    - Statically linked against core
- **Individual programs can be loaded/unloaded**



# Loadable programs

- Programs can be loaded from anywhere
  - Radio (multi-hop, single-hop), EEPROM, etc
- During software development, usually change only one module



# *How well does it work?*

- Works well
  - Program typically much smaller than entire system image (1-10%)
    - Much quicker to transfer over the radio
  - Reprogramming takes seconds
- Static linking can be a problem
  - Small differences in core means module cannot be run
  - We are implementing a dynamic linker

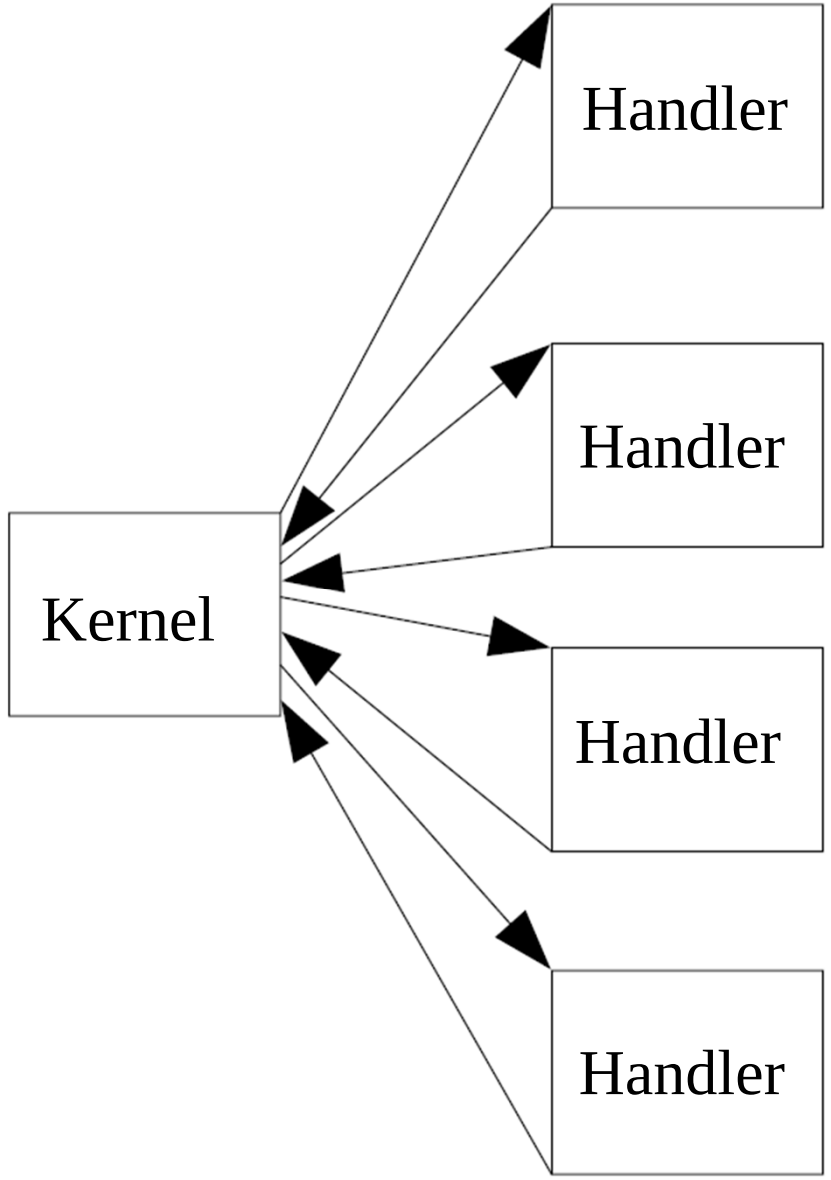
# *Concurrency in Contiki*

# *Concurrency is tricky!*

- Event-driven vs multi-threaded
- Event-driven (TinyOS)
  - Compact, low context switching overhead, fits well for reactive systems
  - Not suitable for e.g. long running computations
    - Public/private key cryptography
- Multi-threading
  - Suitable for long running computations
  - Requires more resources

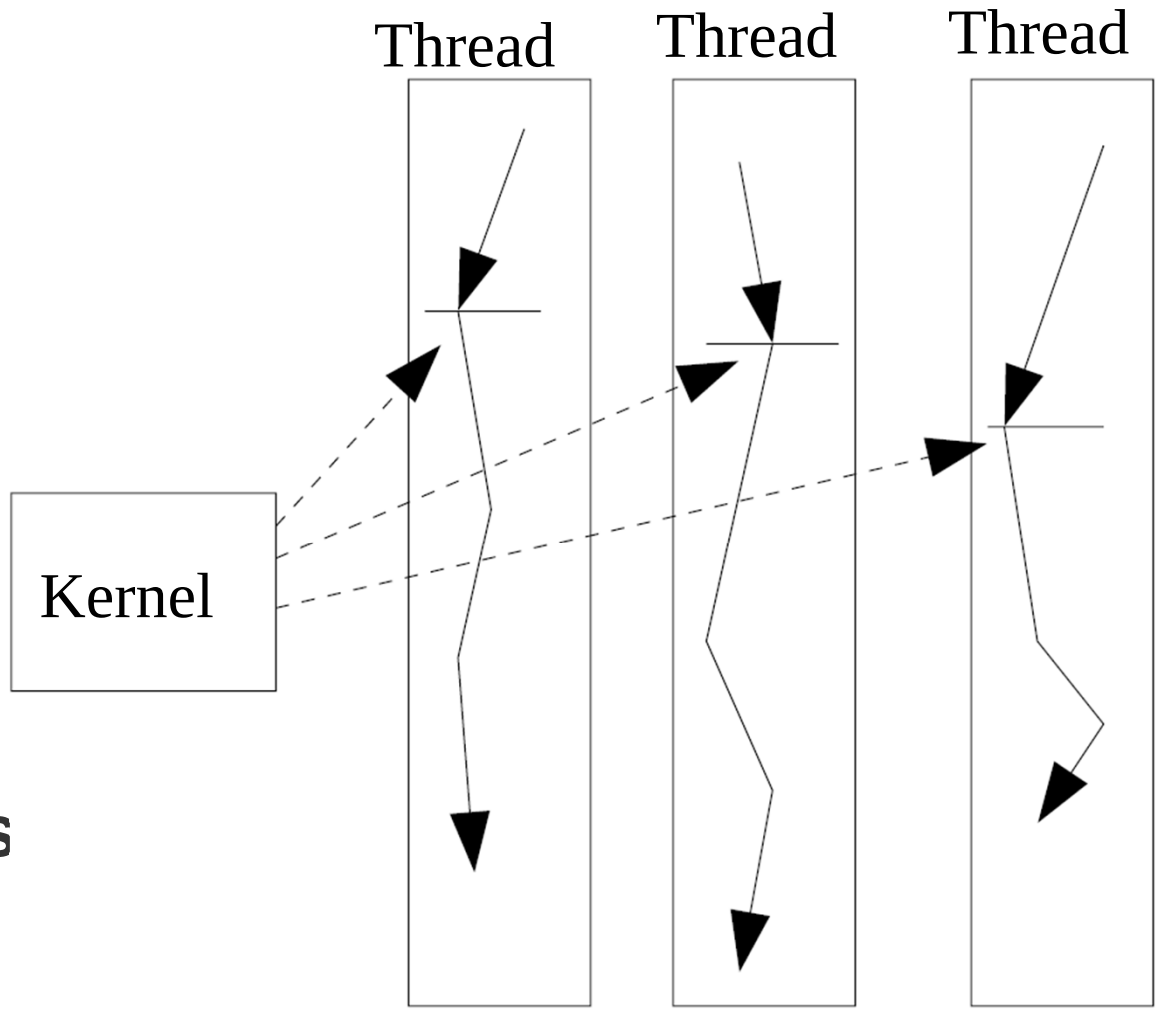
# Event-driven

- Event-driven (TinyOS)
  - Processes do not run without events
  - Event occurs: kernel invokes event handler
  - Event handler runs to completion (explicit `return;`)



# *Multi-threaded*

- Threads blocked, waiting for events
- Kernel unblocks threads when event occurs
- Thread runs until next blocking statement
- **Each thread requires its own stack**
  - **Larger memory usage**



# *Event-driven vs multi-threaded*

## Event-driven

- No `wait()` statements
- No preemption
- State machines
+ Compact code
+ Locking less of a problem
+ Memory efficient

## Multi-threaded

+ `wait()` statements
+ Preemption possible
+ Sequential code flow
- Larger code overhead
- Locking problematic
- Larger memory requirements

**Why don't we try to combine them?**

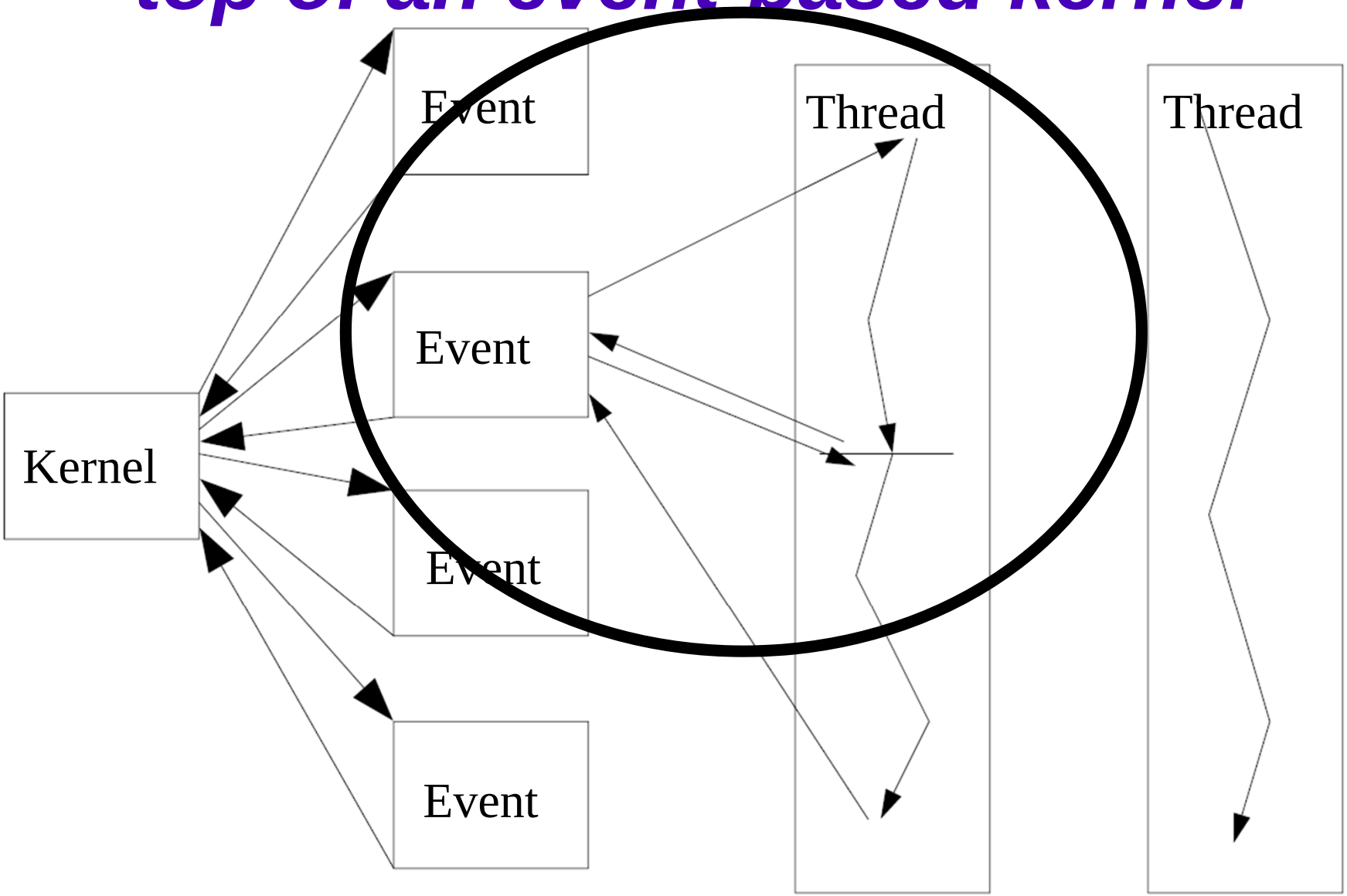
SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

# *Contiki: event-based kernel with threads*

- Contiki: **kernel** is **event-based**
  - Most programs run directly on top of the kernel
- **Multi-threading** implemented as a library
- Threads only used if **explicitly** needed
  - Long running computations, ...
- **Preemption** possible
  - Responsive system with running computations


SWEDISH
INSTITUTE OF
COMPUTER
SCIENCE
SICS

# Contiki: implementing threads on top of an event-based kernel




SWEDISH
INSTITUTE OF
COMPUTER
SCIENCE
SICS


SWEDISH
INSTITUTE OF
COMPUTER
SCIENCE
SICS

# *Conclusions*

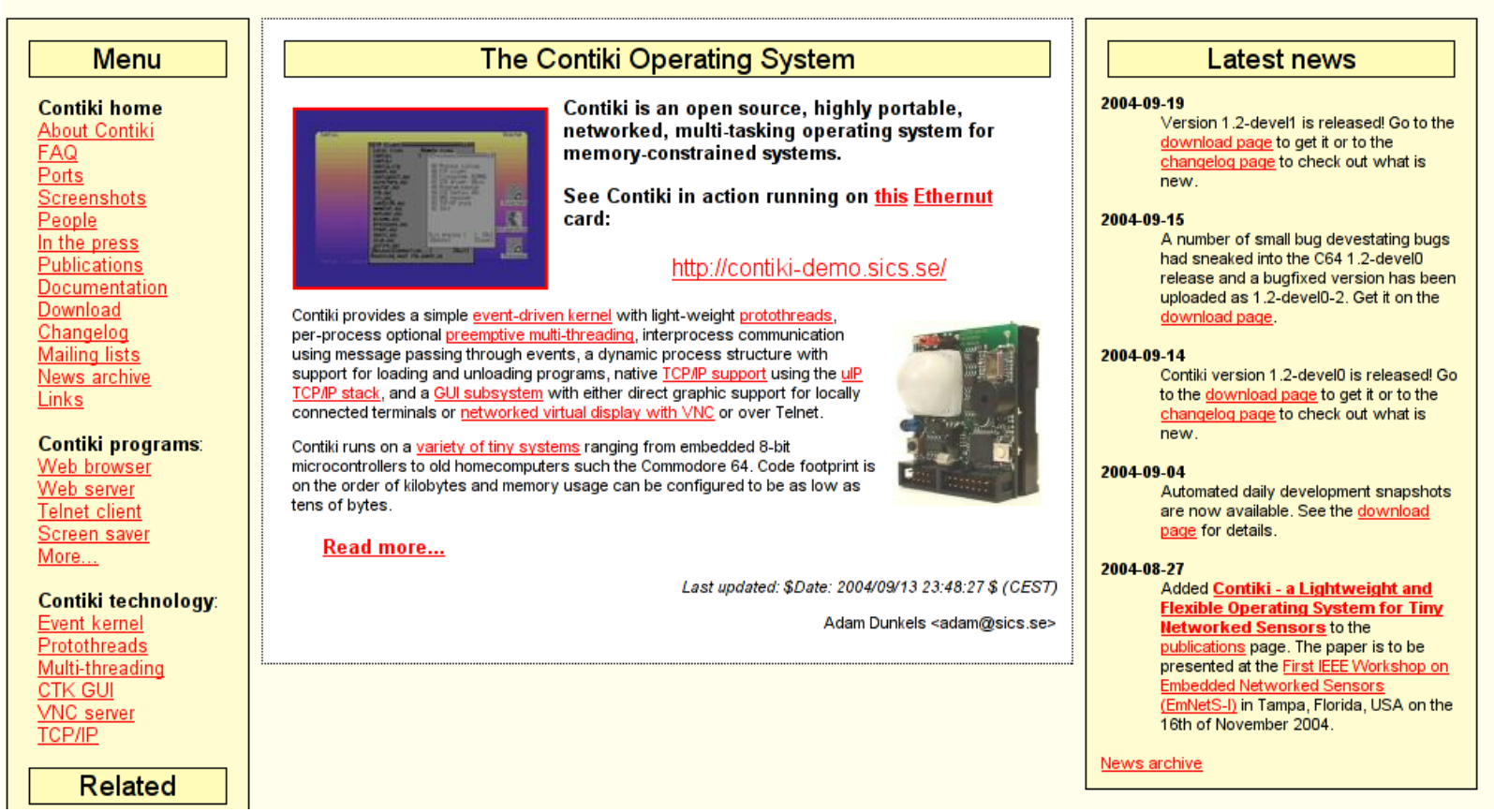
# *Conclusions*

- Contiki – OS for “mote”-class sensor nodes
- Contiki explores trade-offs in
  - **static vs dynamic**
  - **event-driven vs multi-threaded**
- **Loadable programs, works well**
  - **Static linking can be a problem**
- **Threads on an event-driven kernel**
  - **Multi-threading suitable for certain applications**

# *Thank you!*


## Adam Dunkels <adam@sics.se>

## http://www.sics.se/~adam/contiki/

Menu

**Contiki home**
About Contiki
FAQ
Ports
Screenshots
People
In the press
Publications
Documentation
Download
Changelog
Mailing lists
News archive
Links

**Contiki programs**:
Web browser
Web server
Telnet client
Screen saver
More...

**Contiki technology**:
Event kernel
Protothreads
Multi-threading
CTK GUI
VNC server
TCP/IP

Related

The Contiki Operating System

**Contiki is an open source, highly portable, networked, multi-tasking operating system for memory-constrained systems.**

**See Contiki in action running on this Ethernut card:**

http://contiki-demo.sics.se/

Contiki provides a simple event-driven kernel with light-weight protothreads, per-process optional preemptive multi-threading, interprocess communication using message passing through events, a dynamic process structure with support for loading and unloading programs, native TCP/IP support using the uIP TCP/IP stack, and a GUI subsystem with either direct graphic support for locally connected terminals or networked virtual display with VNC or over Telnet.

Contiki runs on a variety of tiny systems ranging from embedded 8-bit microcontrollers to old homecomputers such the Commodore 64. Code footprint is on the order of kilobytes and memory usage can be configured to be as low as tens of bytes.

**Read more...**

*Last updated: $Date: 2004/09/13 23:48:27 $ (CEST)*

Adam Dunkels <adam@sics.se>

Latest news

**2004-09-19**
Version 1.2-devel1 is released! Go to the download page to get it or to the changelog page to check out what is new.

**2004-09-15**
A number of small bug devestating bugs had sneaked into the C64 1.2-devel0 release and a bugfixed version has been uploaded as 1.2-devel0-2. Get it on the download page.

**2004-09-14**
Contiki version 1.2-devel0 is released! Go to the download page to get it or to the changelog page to check out what is new.

**2004-09-04**
Automated daily development snapshots are now available. See the download page for details.

**2004-08-27**
Added **Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors** to the publications page. The paper is to be presented at the First IEEE Workshop on Embedded Networked Sensors (EmNetS-I) in Tampa, Florida, USA on the 16th of November 2004.

News archive

SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

# Backup slides

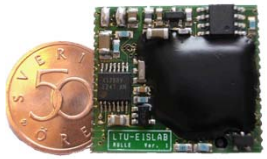
# *Memory management*

- Memory allocated when module is loaded
  - Both ROM and RAM
  - Fixed block memory allocator
- Code relocation made by module loader
  - Exercises flash ROM evenly

# *Protothreads: light-weight stackless threads*

- Protothreads: mixture between event-driven and threaded
  - A third concurrency mechanism
- Allows **blocked waiting**
- Requires per-thread **no stack**
- Each protothread runs inside a single C function
- 2 bytes of per-protothread state

# Embedded operating systems



Heavy-weight processes with memory protection

Lightweight multi-threading, preemption

Event-driven system

Simple control loop

**Linux**

**eCos, OSE, Mantis**

**Contiki**

**TinyOS**