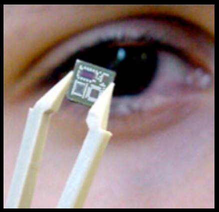
Experiences from a Decade of TinyOS Development

Philip Levis
Stanford University

@OSDI 2012


PRINCE
1999

Back to 1999...



"Information technology (IT) is on the verge of another revolution… The use of EmNets [embedded networks] throughout society could well dwarf previous milestones." [1]

"The motes [EmNet nodes] preview a future pervaded by networks of wireless battery-powered sensors that monitor our environment, our machines, and even us." [2]

[1] National Research Council. Embedded, Everywhere, 2001.
[2] MIT Technology Review. 10 Technologies That Will Change the World, 2003.

- Idea: operating system for “sensor networks”
  - Microcontrollers (bah, virtual memory and 32-bit words)
  - Low-power (2µA - 4mA)
  - Wireless communication (good luck with that)
  - Started as Perl scripts used by a handful of academics

- 13 years later...
  - ~25,000 downloads a year, hundreds of thousands of nodes!
  - Worldwide community of hundreds of contributors!
  - Hundreds of research papers!
  - The Internet of Things!

This Talk

- Two design principles for embedded software
  - Minimize resource use
  - Structure interfaces and code to prevent bugs
- A technical result: static virtualization
- A lesson: avoid the island syndrome

Disclaimer

TinyOS is the work of hundreds of contributors over a decade.

(of which I am only one, the core WG chair, who joined 18 months in)

This paper and talk are my personal opinions and observations.

This Talk

- Two design principles for embedded software
  - Minimize resource use
  - Structure interfaces and code to prevent bugs
- A technical result: static virtualization
- A lesson: avoid the island syndrome

Minimize Resource Use

Model	ROM	RAM	Sleep	Price
F2002	1kB	128B	1.3µA	$0.94
F1232	8kB	256B	1.6µA	$2.73
F155	16kB	512B	2.0µA	$6.54
F168	48kB	2048B	2.0µA	$9.11
F1611	48kB	10240B	2.0µA	$12.86

TI MSP430 Microcontrollers

Minimize Resource Use

Model	ROM	RAM	Sleep	Price
F2002	1kB	128B	1.3μA	$0.94
F1232	8kB	256B	1.6μA	$2.73
F155	16kB	512B	2.0μA	$6.54
F168	48kB	2048B	2.0μA	$9.11
F1611	48kB	10240B	2.0μA	$12.86

TI MSP430 Microcontrollers

Minimize Resource Use

Model	ROM	RAM	Sleep	Price
F2002	1kB	128B	1.3µA	$0.94
F1232	8kB	256B	1.6µA	$2.73
F155	16kB	512B	2.0µA	$6.54
F168	48kB	2048B	2.0µA	$9.11
F1611	48kB	10240B	2.0µA	$12.86

TI MSP430 Microcontrollers

Model	ROM	RAM	Sleep	Price
LM2S600	32kB	8kB	950µA	$2.73
LM3S1608	128kB	32kB	950µA	$4.59
LM3S1968	256kB	64kB	950µA	$6.27

TI ARM CortexM3 Processors

Minimize Resource Use

Model	ROM	RAM	Sleep	Price
F2002	1kB	128B	1.3µA	$0.94
F1232	8kB	256B	1.6µA	$2.73
F155	16kB	512B	2.0µA	$6.54
F168	48kB	2048B	2.0µA	$9.11
F1611	48kB	10240B	2.0µA	$12.86

TI MSP430 Microcontrollers

Model	ROM	RAM	Sleep	Price
LM2S600	32kB	8kB	950µA	$2.73
LM3S1608	128kB	32kB	950µA	$4.59
LM3S1968	256kB	64kB	950µA	$6.27

TI ARM CortexM3 Processors

Minimize Resource Use

Model	ROM	RAM	Sleep	Price
F2002	1kB	128B	1.3μA	$0.94
F1232	8kB	256B	1.6μA	$2.73
F155	16kB	512B	2.0μA	$6.54
F168	48kB	2048B	2.0μA	$9.11
F1611	48kB	10240B	2.0μA	$12.86

TI MSP430 Microcontrollers

Model	ROM	RAM	Sleep	Price
LM2S600	32kB	8kB	950μA	$2.73
LM3S1608	128kB	32kB	950μA	$4.59
LM3S1968	256kB	64kB	950μA	$6.27

TI ARM CortexM3 Processors

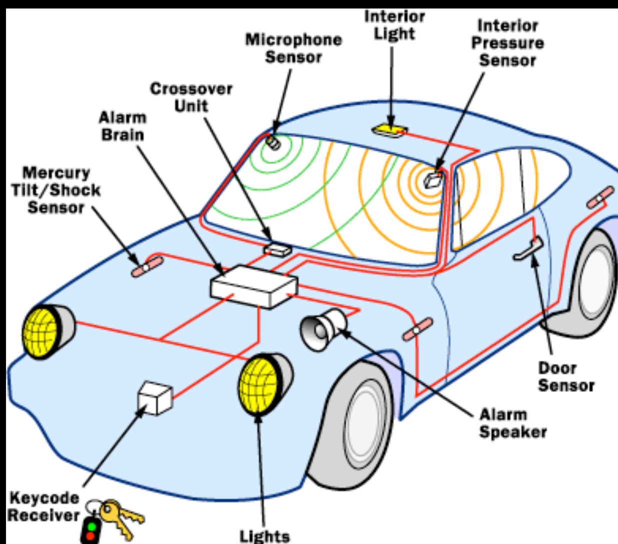
Sleep current necessitates microcontrollers.
Advanced applications run into ROM/RAM limits.

Two Principles

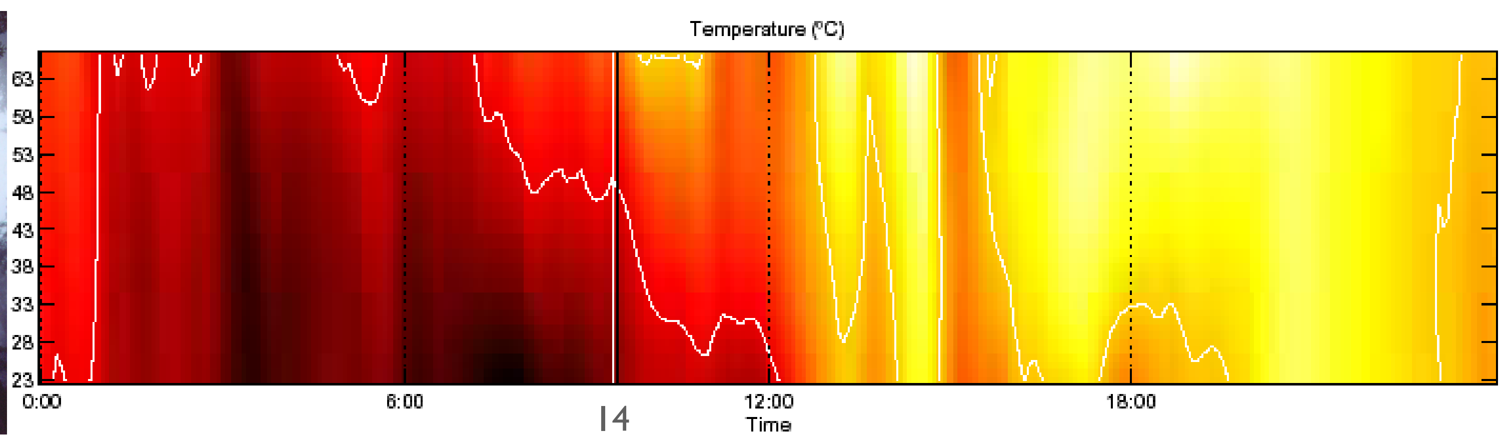
1. Minimize resource use

2. Structure code to prevent bugs

Vision

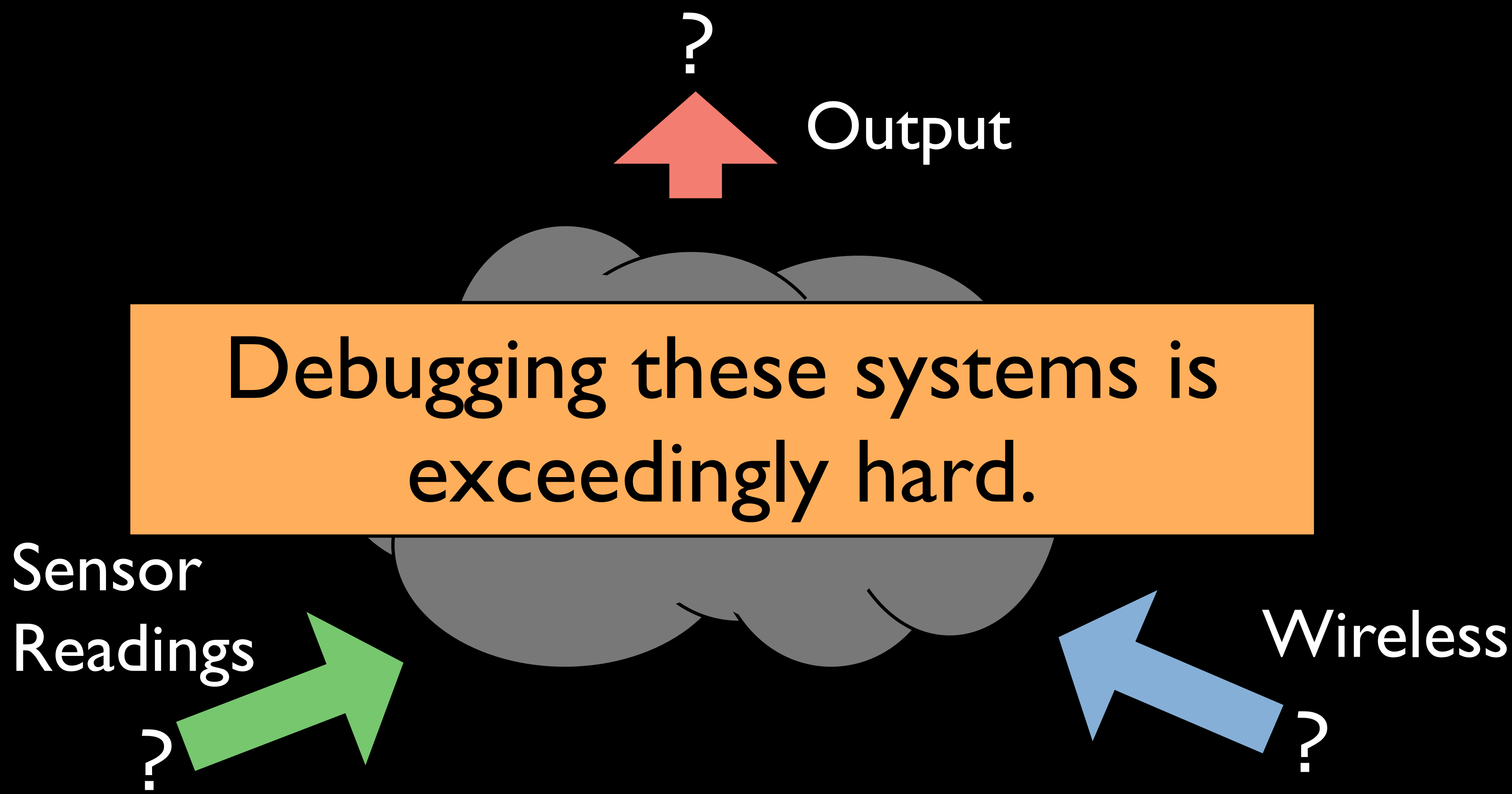


Interior Light
Microphone Sensor
Interior Pressure Sensor
Crossover Unit
Alarm Brain
Mercury Tilt/Shock Sensor
Door Sensor
Alarm Speaker
Keycode Receiver
Lights



Temperature (°C)
63
58
53
48
43
38
33
28
23
0:00
6:00
12:00
18:00
Time

Black Box


?
Output
Debugging these systems is exceedingly hard.
Sensor Readings
?
Wireless
?

This Talk

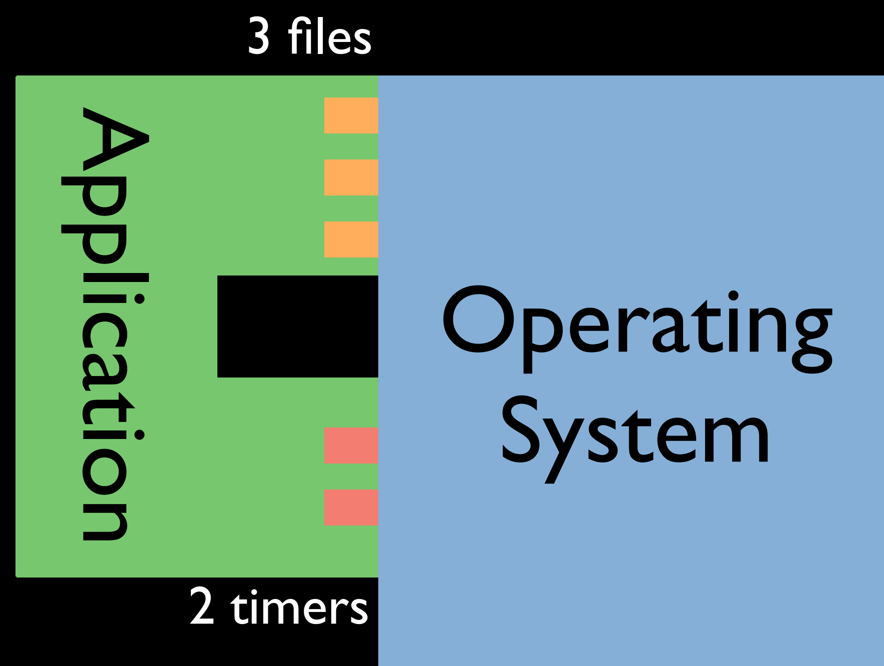
- Two design principles for embedded software
  - Minimize resource use
  - Structure interfaces and code to prevent bugs
- A technical result: static virtualization
- A lesson: avoid the island syndrome

Static Virtualization

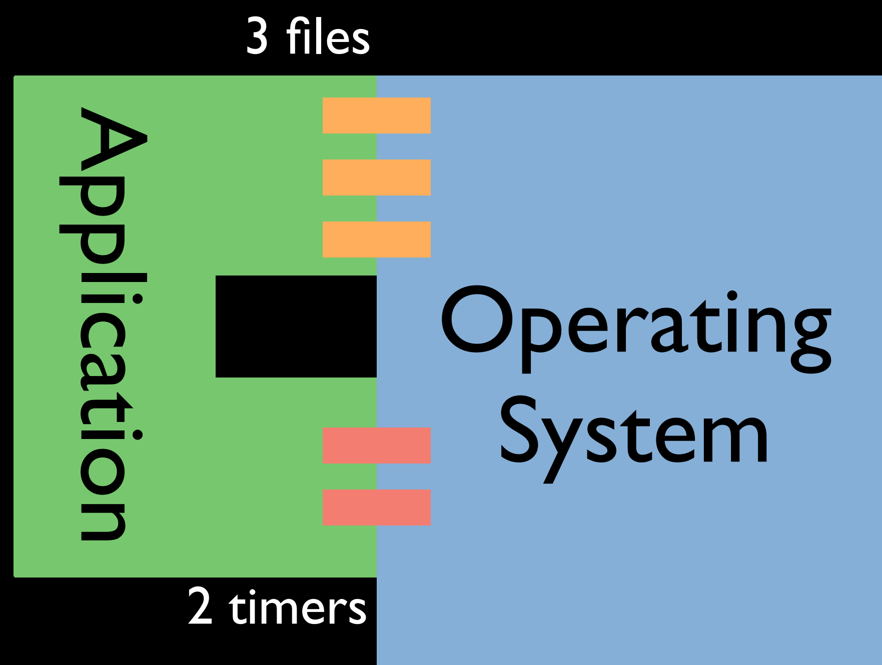
Operating System

Static Virtualization

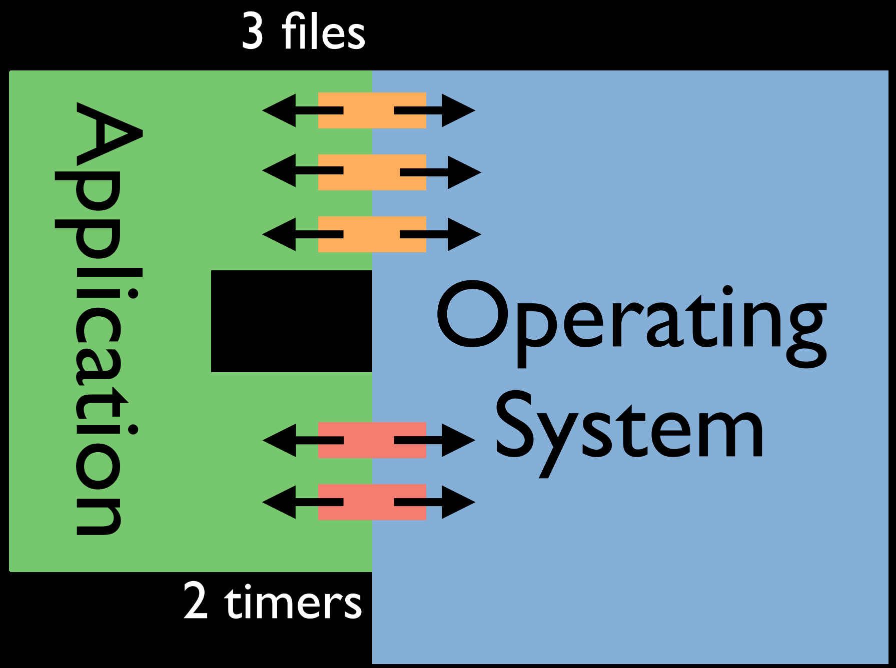
Static Virtualization



Static Virtualization

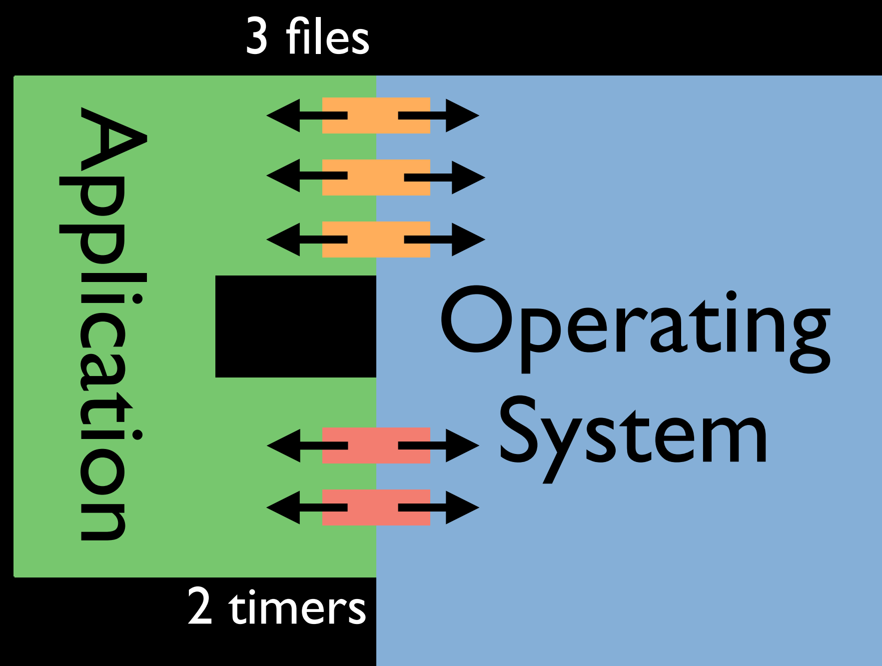


Static Virtualization



Static Virtualization

- Allocates exact RAM
- No pointers
- Cross-call optimization
- Dead code elimination
- Compile-time certainty



Result

Year	Version	Multihop yield
2003[a]	TinyOS 0.6	58%
2005[b]	TinyOS 1.1	68.5%
2009[c]	TinyOS 2.0	99.58%

[a]Szewczyk et al. "An Analysis of a Large Scale Habitat Monitoring Application." SenSys 2004.
"The multi-hop burrow motes perform worse (with a median yield of 58%) but within tolerance"

[b]Werner-Allen et al. "Fidelity and Yield in a Volcano Monitoring Sensor Network." OSDI 2006.
"the median event yield was 68.5%" (events, not packets)

[c]Chipara et al. "Reliable Clinical Monitoring using Wireless Sensor Networks: Experiences in a Step-down Hospital Unit." Sensys 2010.
"the system achieved a median network reliability of 99.68% (range 95.2% – 100%). In contrast, the sensing reliability was significantly lower."

Result

Year	Version	Multihop yield
2003[a]	TinyOS 0.6	58%
2005[b]	TinyOS 1.1	68.5%
2009[c]	TinyOS 2.0	99.58%

[a]Sz

Static virtualization enabled applications to be highly robust, dependable, and efficient.

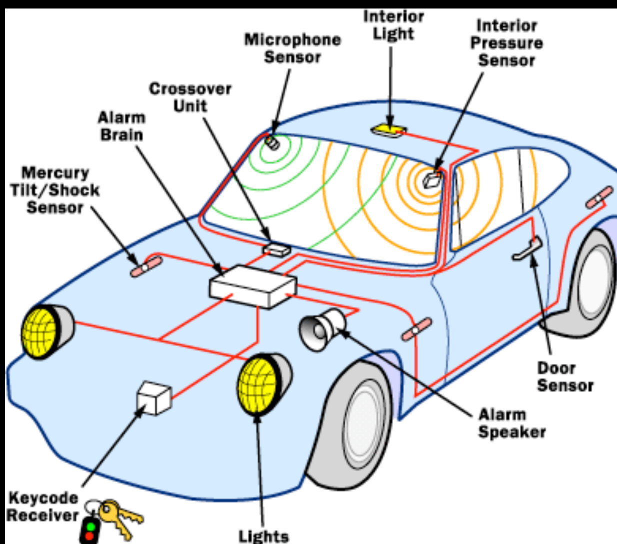
[b]W

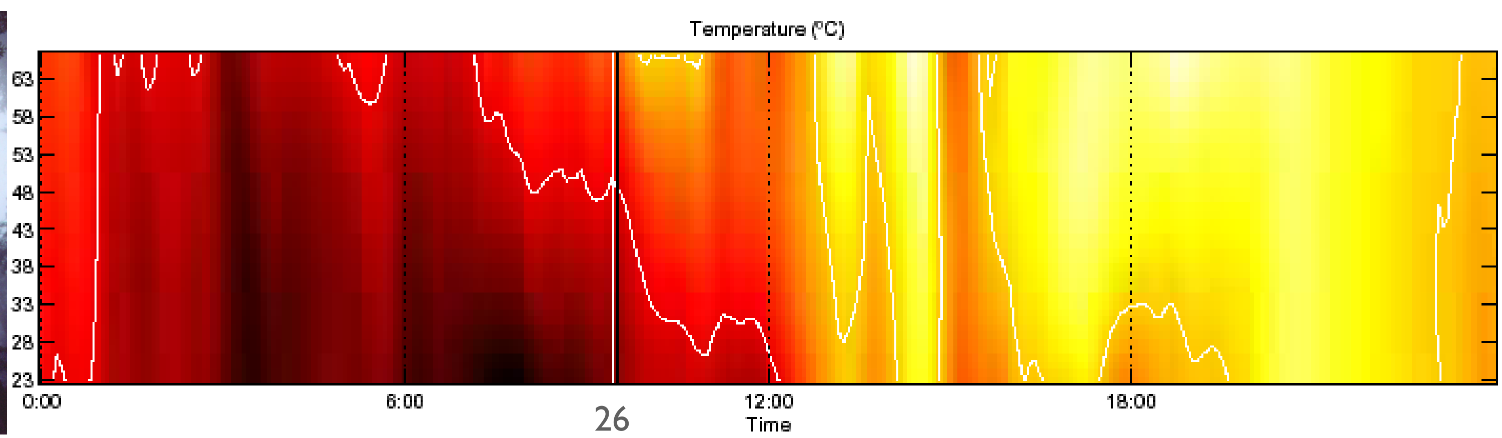
[c]Chipara et al. "Reliable Clinical Monitoring using Wireless Sensor Networks: Experiences in a Step-down Hospital Unit." Sensys 2010. "the system achieved a median network reliability of 99.68% (range 95.2% – 100%). In contrast, the sensing reliability was significantly lower."

This Talk

- Two design principles for embedded software
  - Minimize resource use
  - Structure interfaces and code to prevent bugs
- A technical result: static virtualization
- A lesson: avoid the island syndrome

Applications



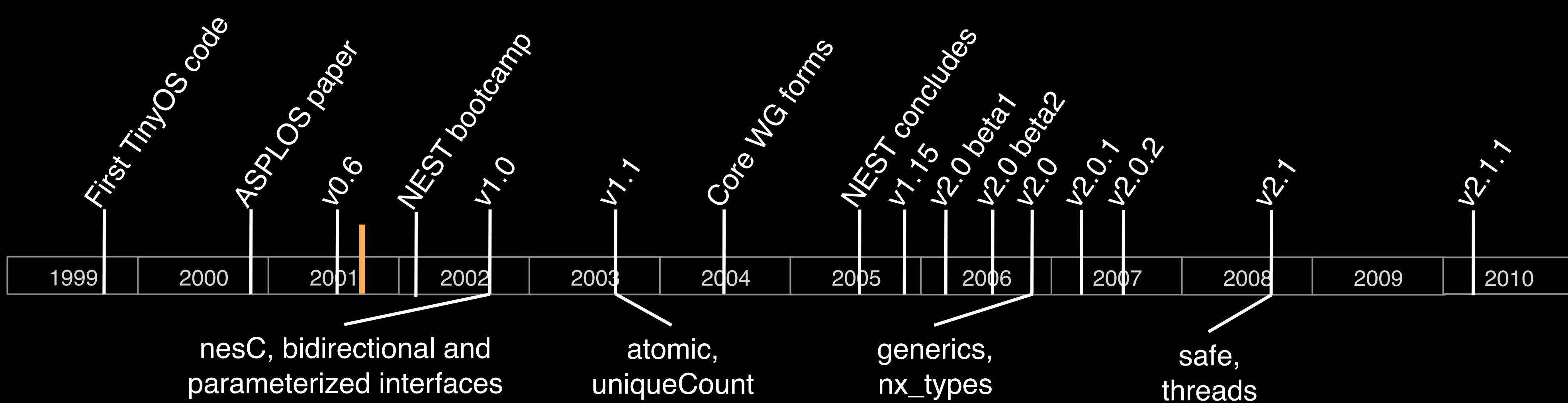


Research vs. Practice

- TinyOS technically focused on enabling users to build larger, more complex applications

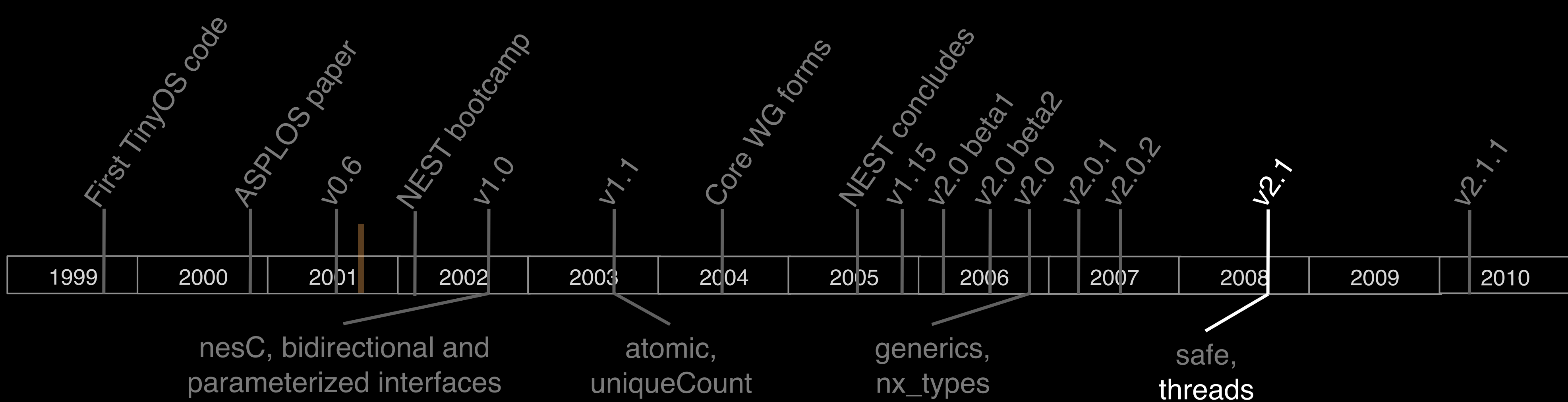
- Doing so increased the learning curve to building simple ones

Timeline



First TinyOS code
ASPLOS paper
v0.6
NEST bootcamp
v1.0
v1.1
Core WG forms
NEST concludes
v1.15
v2.0 beta1
v2.0 beta2
v2.0
v2.0.1
v2.0.2
v2.1
v2.1.1
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
nesC, bidirectional and parameterized interfaces
atomic, uniqueCount
generics, nx_types
safe, threads

Timeline



First TinyOS code
ASPLOS paper
v0.6
NEST bootcamp
v1.0
v1.1
Core WG forms
NEST concludes
v1.15
v2.0 beta1
v2.0 beta2
v2.0
v2.0.1
v2.0.2
v2.1
v2.1.1
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
nesC, bidirectional and parameterized interfaces
atomic, uniqueCount
generics, nx_types
safe, threads

Statically Virtualized Timer

(TinyOS 2.x)

AppP.nc

```
Control.start() {
    Timer.start(..)
}

Timer.fired() {
    send_packet();
}
```

AppC.nc

```
T = new TimerC()
AppP.Timer -> TimerC.Timer
```

Implementation

(TinyOS 2.x)

AppP.nc

```
Control.start() {
    Timer.start(..)
}

Timer.fired() {
    send_packet();
}
```

AppC.nc

```
T = new TimerC()
AppP.Timer -> TimerC.Timer
```

TimerC.nc

```
#define TS unique("T")
TimerC.Timer =
  TimerP.Timer[TS]
```

TimerP.nc

```
#define NT uCount("T")
timer_t ts[NT];

clock_interrupt {
  update_ts()
  for i = 0 to NT-1
    if (ts[i].fire)
      Timer[i].fired();
}

Timer[i].start(...) {
    startTimer(i, ..)
}
```

Implementation

(TinyOS 2.x)

AppP.nc

```
Control.start() {
    Timer.start(..)
}

Timer.fired() {
    send_packet();
}
```

AppC.nc

```
T = new TimerC()
AppP.Timer -> TimerC.Timer
```

TimerC.nc

```
#define TS unique("T")
TimerC.Timer =
  TimerP.Timer[TS]
```

TimerP.nc

```
#define NT uCount("T")
timer_t ts[NT];

clock_interrupt {
  update_ts()
  for i = 0 to NT-1
    if (ts[i].fire)
      Timer[i].fired();
}

Timer[i].start(...) {
    startTimer(i, ..)
}
```

TinyOS 0.6

APP.c

```
APP_START() {
  APP_TIMER_INIT(..);
}

APP_TIMER() {
  send_packet();
}
```

APP.desc

```
APP_TIMER_INIT TIMER_START
TIMER_FIRE APP_TIMER
```

TIMER.c

```
timer_t ts;

TIMER_START(...) {
  set_ts();
  init_interrupt();
}

clock_interrupt {
    update_ts()
    TIMER_FIRE();
}
```

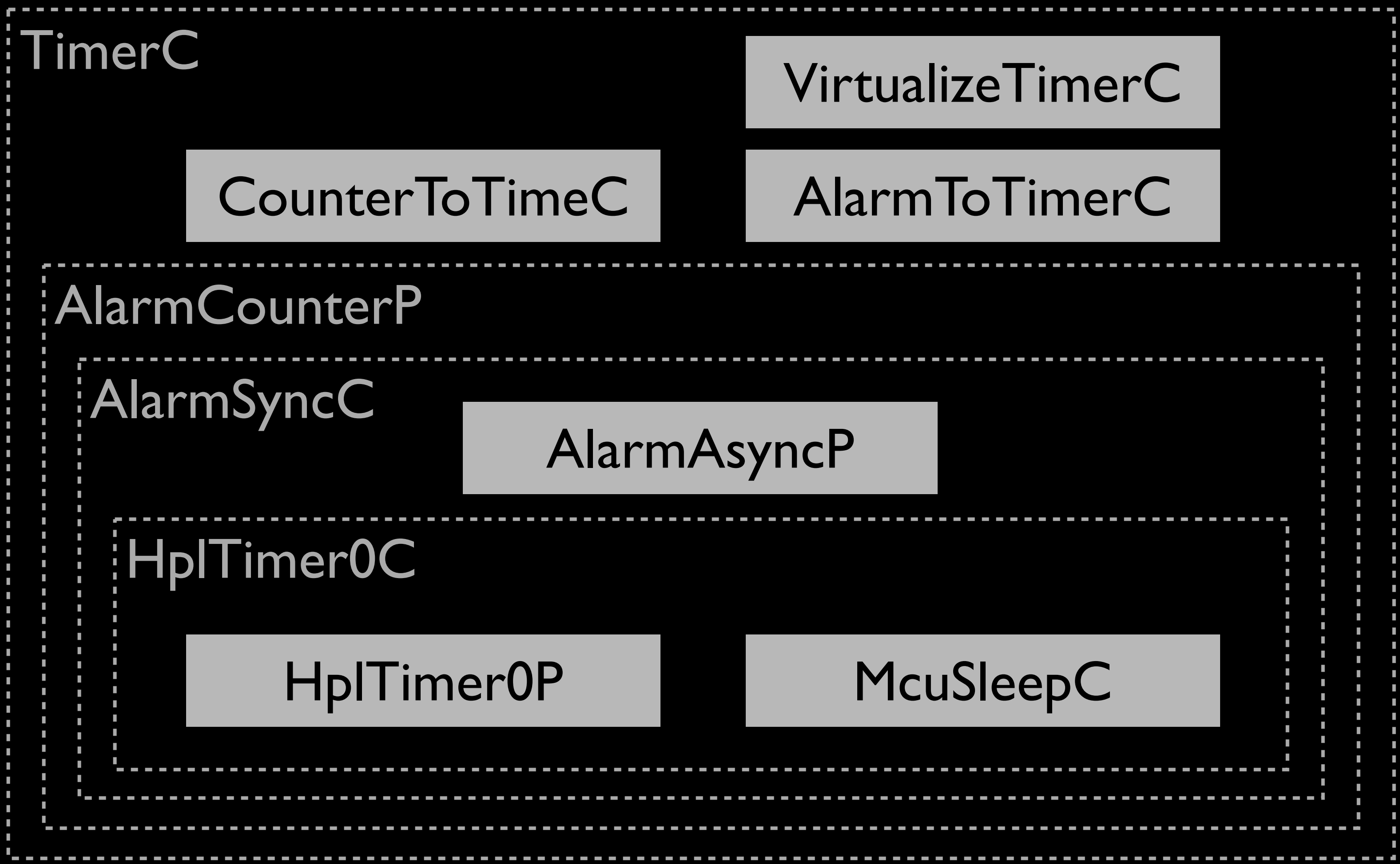
Code Evolution

- Code evolved to use nesC features in more complex and intricate ways
  - Improved software dependability
  - Allowed more complex applications
  - Served the needs of the community
- Increased barrier to entry: island syndrome

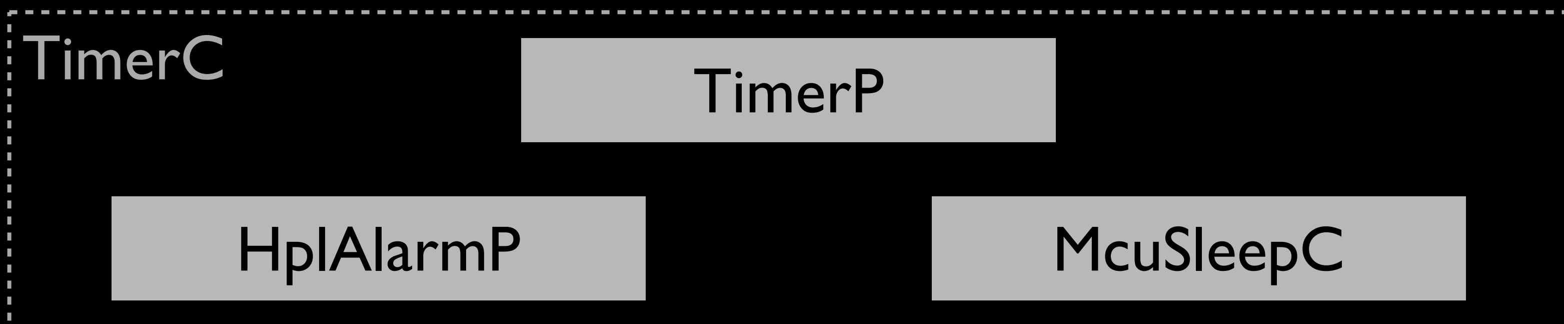
Death by Components

- Fine-grained component toolkits are great for building and evolving a system
- The end result is difficult for a new user to understand: increases the learning curve
- Need to transition to structurally simpler implementations

Death by Components



Another Approach




Main Site Blog Playground Forum Labs Store Help | Sign in or Register

ARDUINO

search

Buy Download Getting Started Learning Reference Hardware FAQ

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software running on a computer (e.g. Flash, Processing, MaxMSP).

- Tremendously successful academic project
    - Started as Perl scripts used by a handful of academics
    - Now ~100 downloads a day, hundreds of thousands of nodes
    - Has a worldwide community of hundreds of contributors

- But it could have been more so
    - Missed being a platform for simple sensing apps (Arduino)
    - Missed being a platform for the Internet of Things (Contiki)
    - “Applications” became “hard applications”
    - Should have focused on the simple as much as the complex (the island syndrome)

Disclaimer

TinyOS is the work of hundreds of contributors over a decade.

(of which I am only one, the core WG chair, who joined 18 months in)

This paper and talk are my personal opinions and observations.

TinyOS

TinyOS is an open source, BSD-licensed operating system designed for low-power wireless devices, such as those used in sensor networks, ubiquitous computing, personal area networks, smart buildings, and smart meters. A worldwide community from academia and industry use, develop, and support the operating system as well as its associated tools, averaging 35,000 downloads a year.

Latest News

August 20, 2012: TinyOS 2.1.2 is now officially released; you can download it from the debian packages on tinyos.stanford.edu. Manual installation with RPMs with **the instructions on docs.tinyos.net** will be forthcoming. TinyOS 2.1.2 includes:

- Support for updated msp430-gcc (4.6.3) and avr-gcc (4.1.2).
- A complete 6lowpan/RPL IPv6 stack.
- Support for the ucmini platform and ATmega128RFA1 chip.
- Numerous bug fixes and improvements.

July 21, 2010: The transition from hosting TinyOS at Sourceforge to **Google code** is now complete. Part of this transition included placing all of TinyOS under a **New BSD license** (in Sourceforge several compatible licenses were used). Thanks to all of the developers for agreeing to move to a uniform license!

FAQ

Frequently asked questions about TinyOS

Learn

Download TinyOS and learn how to use it

Community

TinyOS Working Groups, mailing lists, and TEPs

I'd like to especially acknowledge Jason Hill, David Culler, David Gay, Cory Sharp, Eric Brewer, Shankar Sastry, Joe Polastre, Vlado Handziski, Jan Heinrich-Hauer, Kevin Klues, David Moss, Omprakash Gnawali, Jonathan Hui, John Regehr, Matt Welsh, Alec Woo, Robert Szewczyk, Kamin Whitehouse, Philip Buonadonna, Ben Greenstein, Miklos Maroti, Andreas Koepke, and Janos Sallai, as well as Razvan Musaloiu-E., JeongGil Ko, Philipp Huppertz, Antonio Linan, Steve Ayers, Kristin Wright, Steven Dawson-Haggerty, Jan Beutel, Branislav Kusy, Prabal Dutta, Gilman Tolle, Thomas Schmid, Chad Metcalf, Henri Dubois-Ferriere, Deepak Ganesan, Laurynas Riliskis, Eric Decker, Martin Turon, and Peter Bigot.

TinyOS is also deeply indebted to its users, their bug reports, feature requests, and hard work.