Fault-tolerant Sensor Nodes with Erlang/OTP and Arduino

Kenji Rikitake

10-MAR-2016
Erlang Factory SF Bay 2016
San Francisco, CA, USA
@jj1bdx

Erlang Factory SF Bay 2010-2016
speaker (7th year!)

Program Committee Member of ACM
Erlang Workshop (2011, 2013, 2016)
and CUFP 2016

Basic principles

Stabilize
Simplify
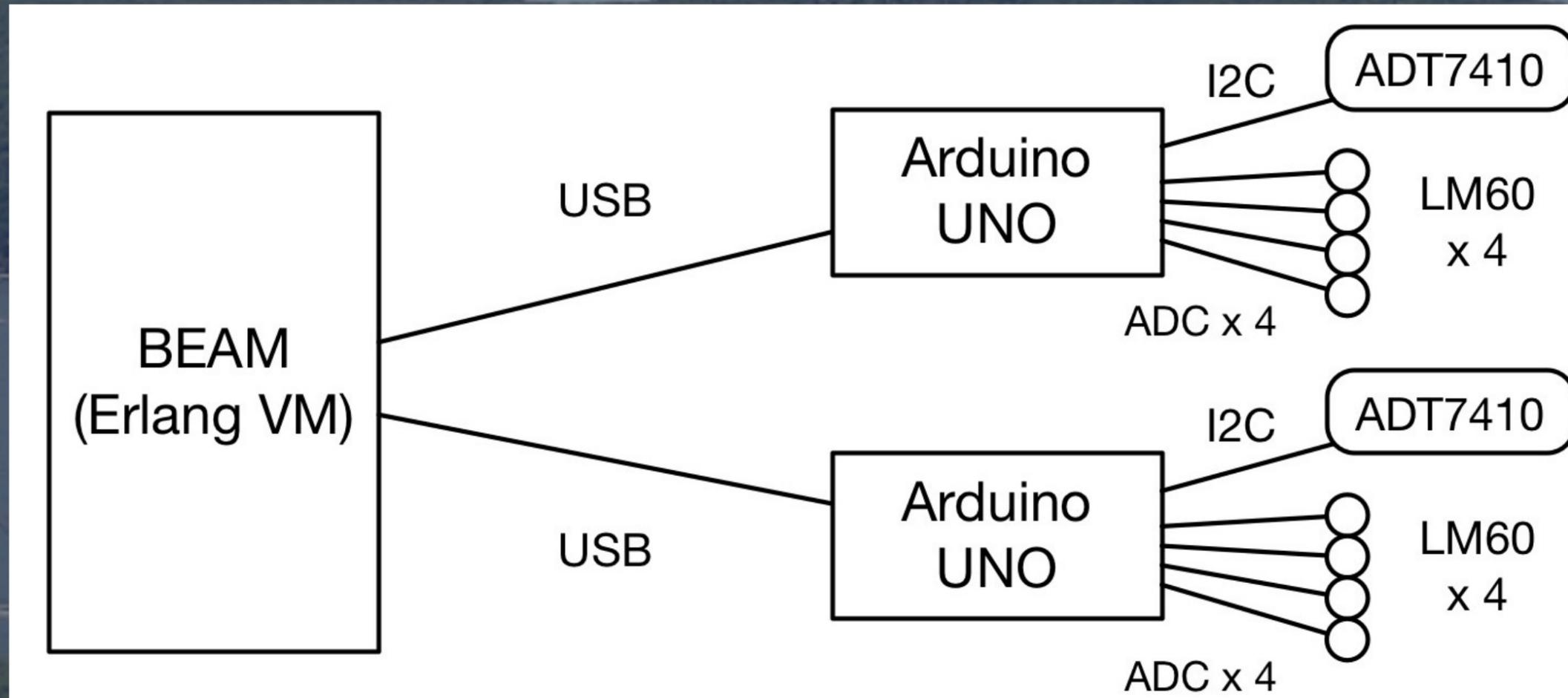## "Let It Crash"
Update dynamically

In this talk

**Bearfort sensor shield
8-bit Arduino basics
Wire protocols
How Erlang talks with Arduino**

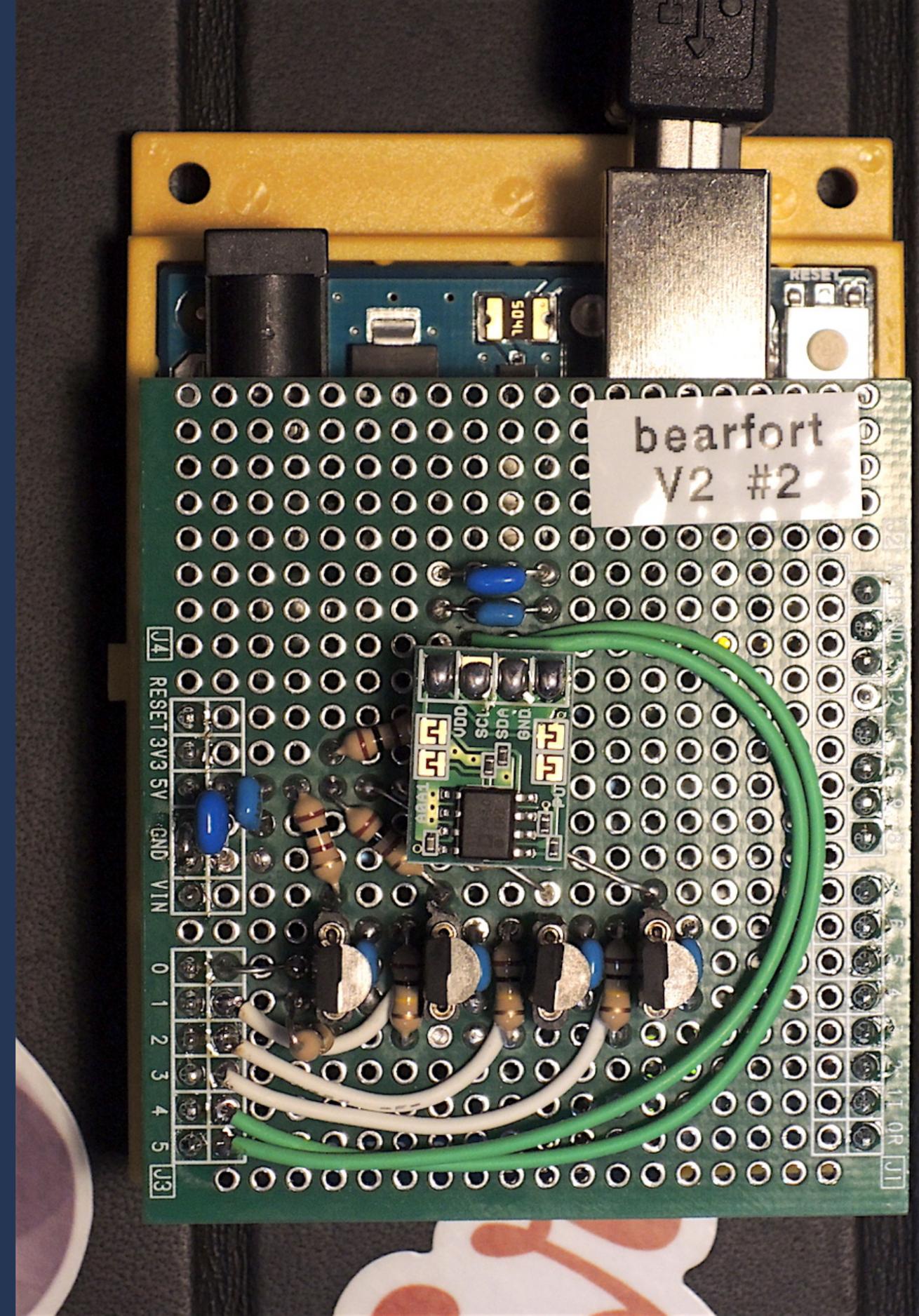# Bearfort[1] system diagram

Bearfort sensor shield

Temperature sensors

Analog Devices ADT7410 on TWI/I2C

4 x Texas Instruments LM60s on ADC0-3

All sensors are powered by +5V
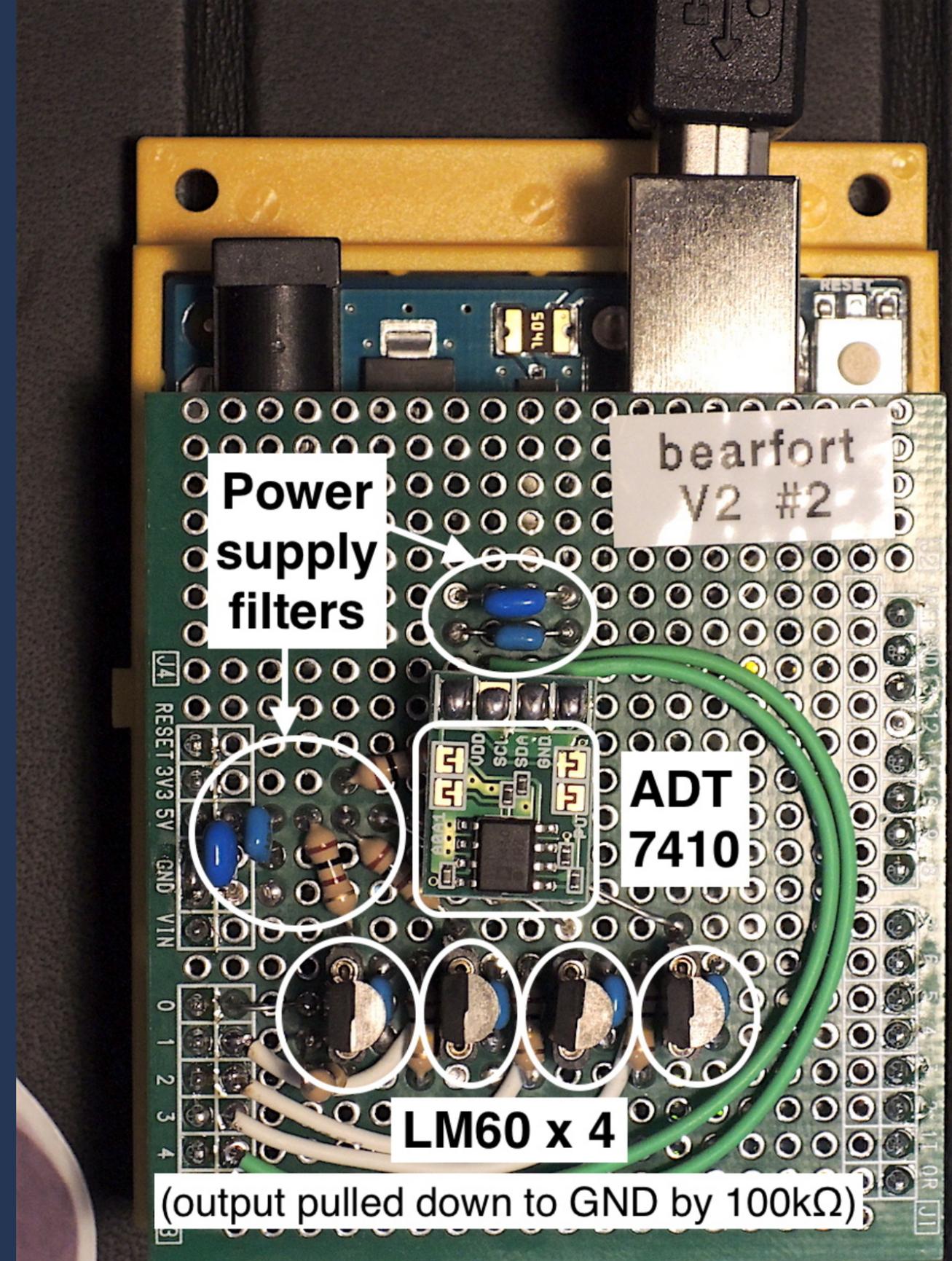All sensors are replaceable

What Bearfort shield can do?

Fault tolerant temperature sensing
5 sensors
Robust against sensor failures

Power supply filters

bearfort V2 #2

ADT 7410

LM60 x 4
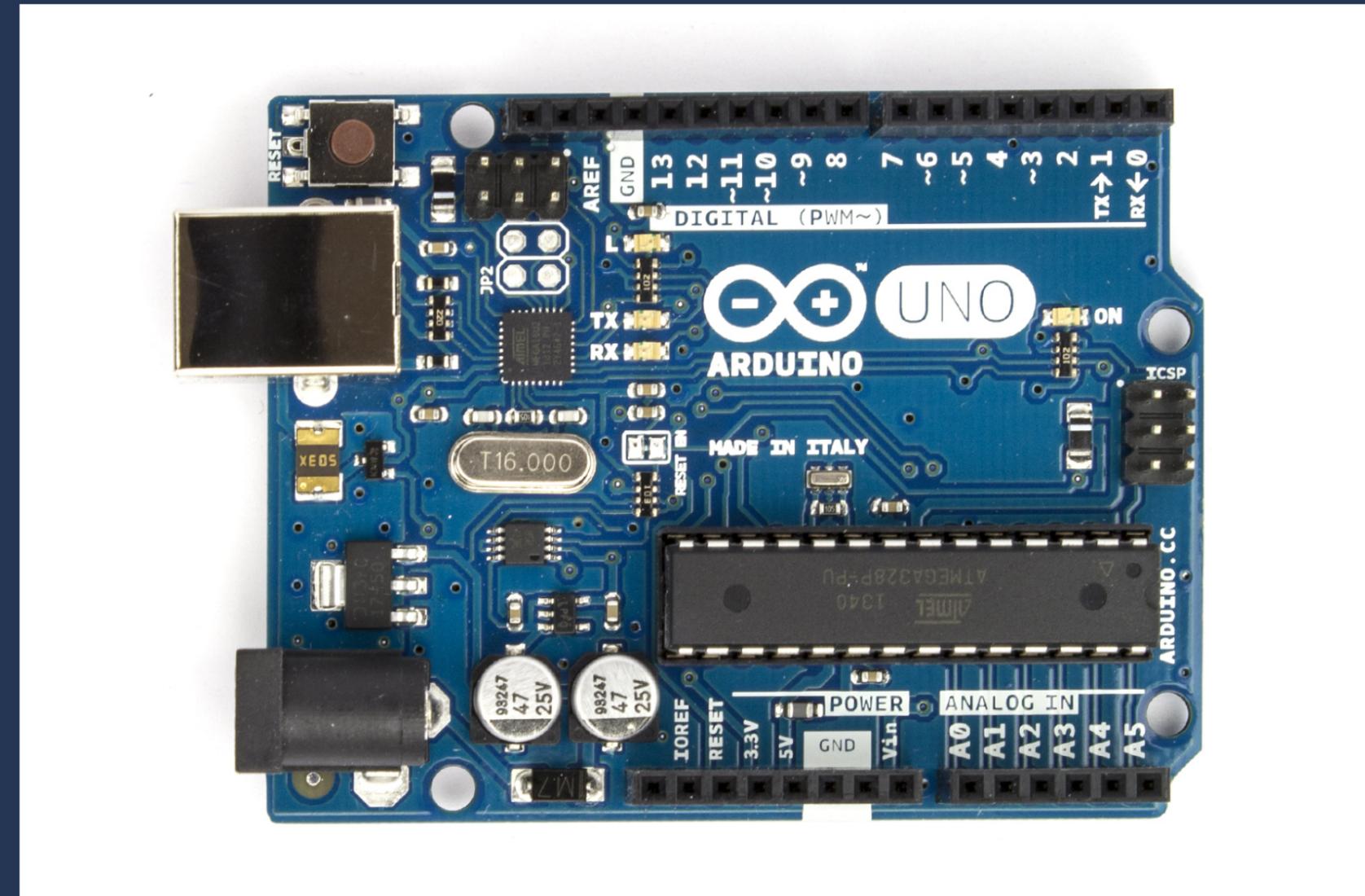
(output pulled down to GND by 100kΩ)

Arduino Uno R3

Atmel AVR ATmega328P

Powered by USB (5V) or external power supply (7~12V)

4 Analog Input + I2C + SPI

Price: USD24.95[2] as of March 2016 at SparkFun Electronics

[2] Photo: Wikimedia Commons, By oomlout - ARDU-UNO-03-Front, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=40551883

AVR development

Write C event loop

Try and error

Interrupts are for timer only

Chip programmer

Essential for writing boot loader firmware

Hardware diagnostics

Hardware protection bit configuration

Replicating chips

Photo: AVR Dragon, circa 2008

AVR Dragon

Hardware principles

Stabilize
Simplify
"Let It Crash"

Stabilize hardware

Solder (no breadboard)
Less contact points
Prepare for contact failures

Contact failures

Open circuit
Short circuit

Open circuit

Stabilize output for open contact failure

disconnected

input ✗ output:
unstable

disconnected

input ✗ output:
grounded
~100kΩ when
disconnected
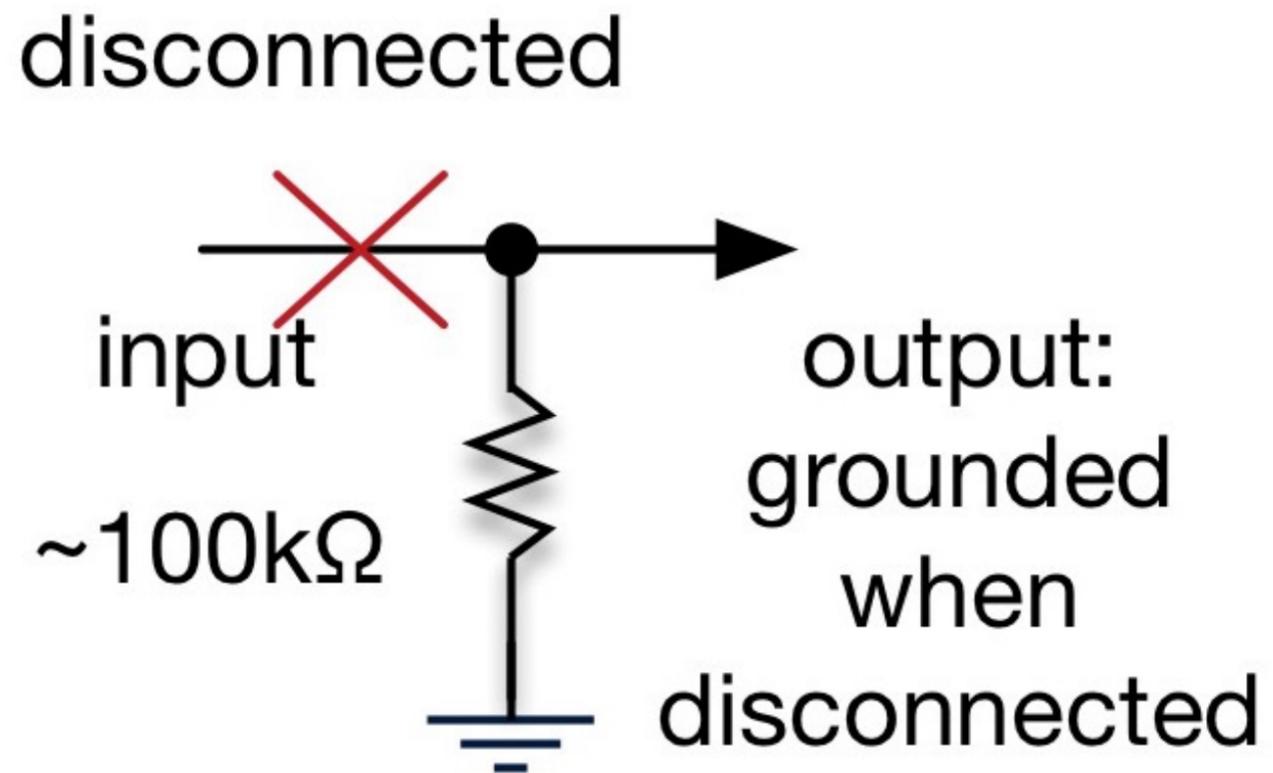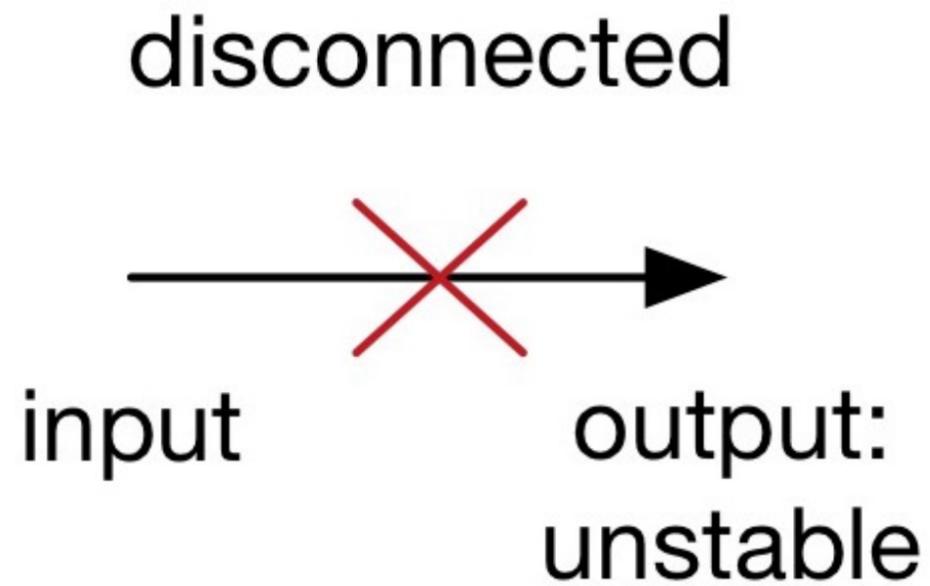
LM60 open circuit failure

Output
Voltage

Expected failure mode

Actual failure mode example

0V
(Ground)

Pulled-down to the ground

Open contact failure occurs
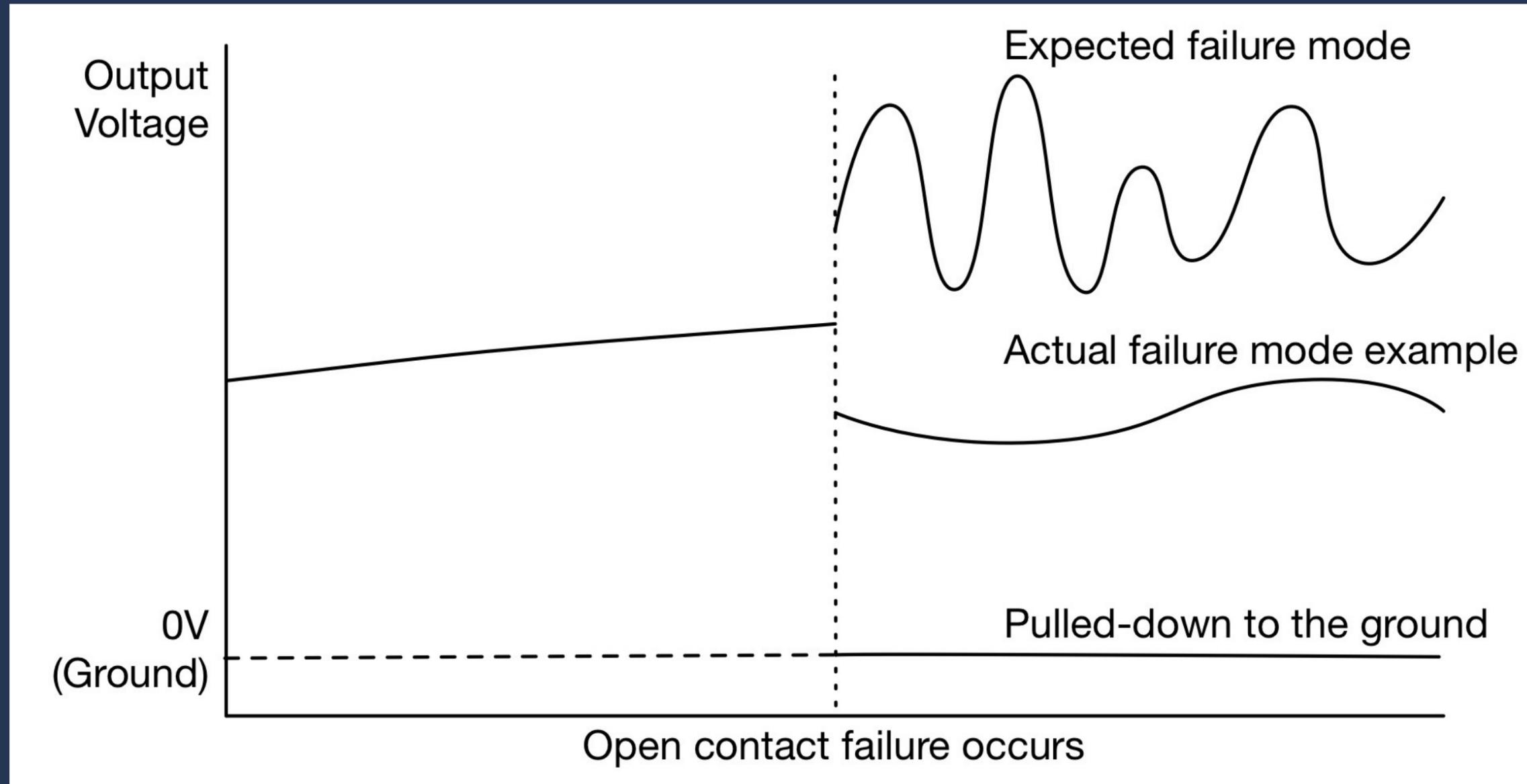
Short circuit
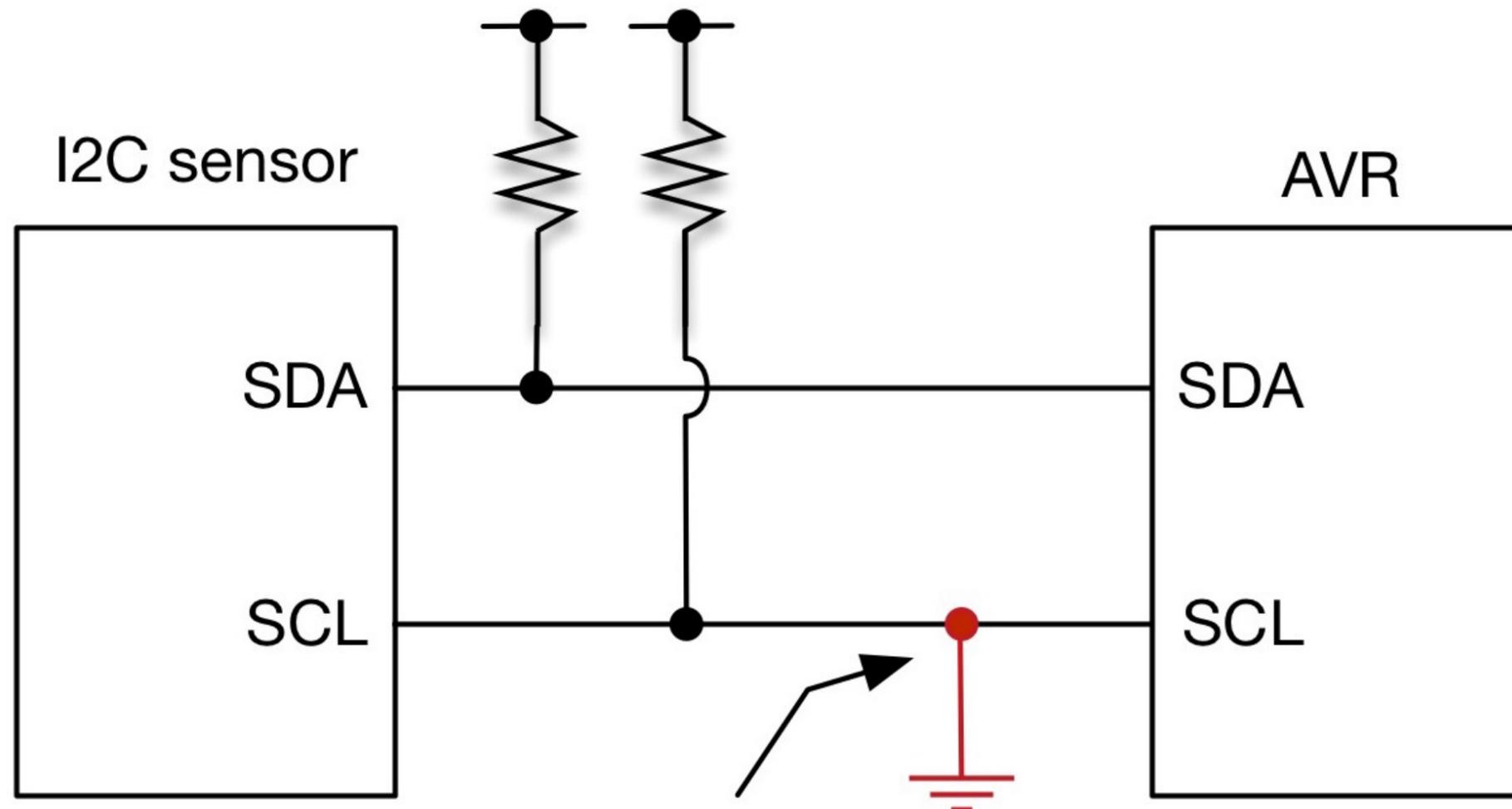
Forcefully grounded: disrupting clock - system halts!

Simplify hardware

Keep firmware small
Return raw sensor values
Leave no tunable parts

"Let It Crash"

Reset when hardware fails
Allow external reset
Use watchdog timer if needed

Resetting Arduino from USB
Turn off DTR/RTS for 50msec and turn back on

```
%%% Using Michael Santos'
%%% stk500 and srly repository code
{ok,FD} = serctl:open("/dev/cu.usbmodem1D11311"),
[begin dtrrts(FD, Status),
    timer:sleep(50) end || Status <- [off, on] ].
```

Yes, that's it!

Software principles

Simplify
"Let It Crash"
Dynamic update

Simplify wire protocol
Polling from host
Fixed-length output
No tunable parts

Serial line protocol

No frame: synchronization needed
Fixed length = pattern matching
No tuning = idempotent

Serial line control from Erlang/OTP

Michael Santos' <u>srly</u> package
TTY control (ioctl())
Fixed-length reading function is *extremely* useful

Wire protocol message format
16-byte fixed length

STX	magic number	Device ID	ADT7410 value
LM60 #1 value	LM60 #2 value	LM60 #3 value	LM60 #4 value

ETX

16 bytes in total
Note: 2-byte values are all little endian (LSB, MSB)

Wire protocol in Erlang

```erlang
{ok, <<2, 16#51, 16#82,  % 2 == STX
 DevId:2/little-unsigned-integer-unit:8,
 ADT0:2/little-signed-integer-unit:8,
 A0:2/little-unsigned-integer-unit:8,
 A1:2/little-unsigned-integer-unit:8,
 A2:2/little-unsigned-integer-unit:8,
 A3:2/little-unsigned-integer-unit:8,
 3>>} = read_serial(FD). % 3 == ETX
```

"Let It Crash"

Erlang does it *very well*
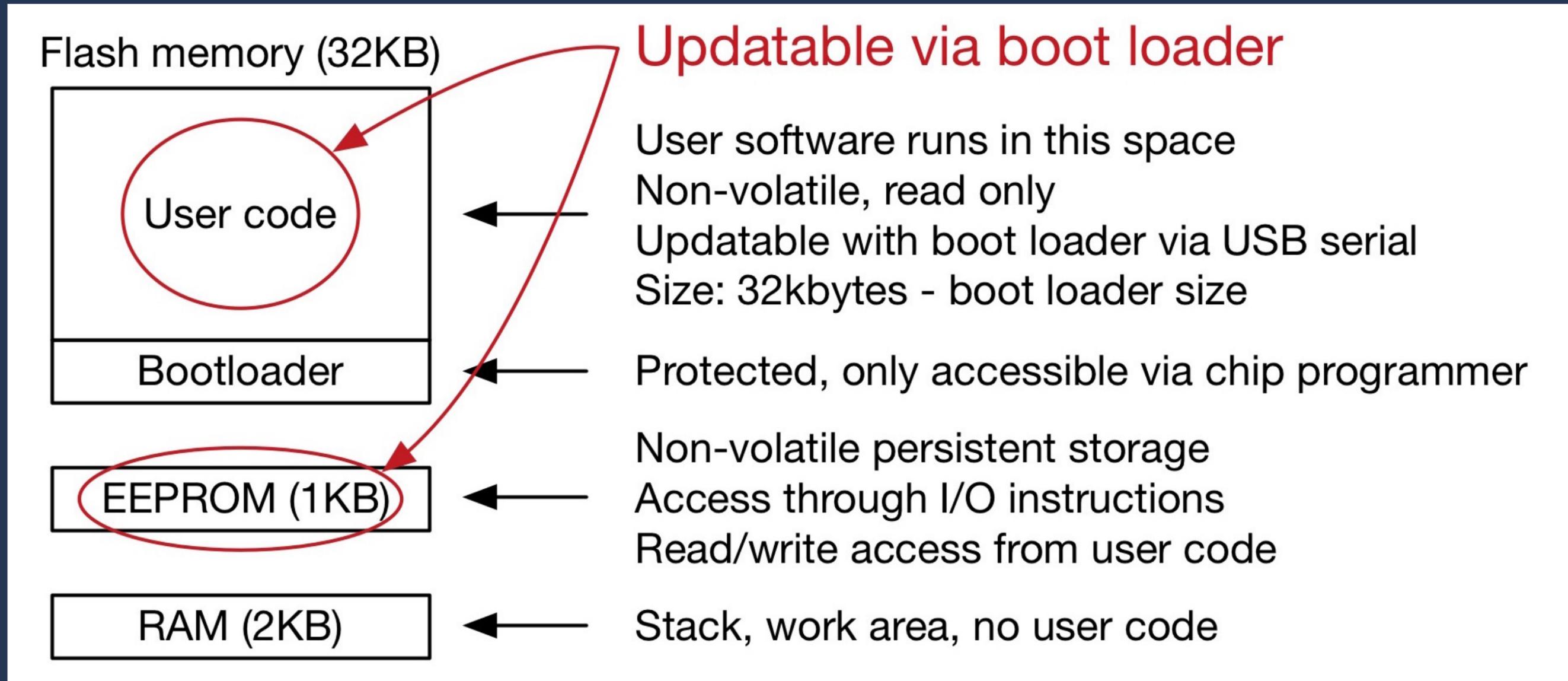**Hardware reset control**
Serial ioctl-capable API

Update dynamically

**Updating Arduino from Erlang
Use boot loader for code loading
Slow (5~10 seconds) but feasible**

ATmega328p memory model

Flash memory (32KB)

User code

Bootloader

EEPROM (1KB)

RAM (2KB)

Updatable via boot loader

User software runs in this space
Non-volatile, read only
Updatable with boot loader via USB serial
Size: 32kbytes - boot loader size

Protected, only accessible via chip programmer

Non-volatile persistent storage
Access through I/O instructions
Read/write access from user code

Stack, work area, no user code

Update example

```
update() ->
    Hex = stk500:hex_file(
        "./arduino-uno/bearfort-arduino.hex"),
    Bytes = stk500:chunk(Hex, 128),
    {ok,FD} = stk500:open(
                "/dev/cu.usbmodem1D11311",
                [{speed, b115200}]),
    ok = stk500:load(FD, Bytes).
```

Issues

**Slow USB connection transition
Automated sensor calibration
Modeling fault tolerant operation**

Future directions

Indoor/outdoor field testing
Embedded Erlang node
Multiple sensors/nodes

Excluded from this talk

- TCP/IP: MQTT, CoAP, etc.

- Cryptographic security

- Host OS device drivers

- non-8bit Arduino boards

- Erlang/ALE = for Raspeberry Pi

Related work (1/2)

- Code and slides of this presentation

- srly: Erlang NIF serial API

- stk500: Erlang AVR boot loader API

- avr-libc, avr-binutils, avr-gcc

- avrdude: AVR program loader

- optiboot: AVR boot loader firmware

Related work (2/2)

- Omer Kiric's talk on EF SF Bay 2013

  - Erlang/ALE on GitHub

- Frank Hunleth's talk on EF SF Bay 2015

  - Robot running in Elixir

  - elixirbot on GitHub

Thanks to:

Michael Santos
Erlang Solutions
...and you all!

Thank you Questions?