Open Minded Storms

The NXT meets OpenSource

Agenda

- Introductions
- Naive programming
- Event-driven programming
- The bright side of the Moon
- Future and closing notes

Intro: myself & the presentation

- A few words about your humble servant
- Brief review of my academic work in robotics
  - Don’t worry, it’s remarkably brief
- The Jaw Dropper
  - I don’t have it.
  - Wish I had.
  - Sorry folks.

Intro: The Platform (brief)

- Programmable robotic LEGO brick
- Released July 2006
- Attaches to usual Technic bricks
- 3 motors / 4 sensors per brick
- Able to store programs on FLASH
- Can be controlled / programmed by USB / bluetooth
- 100x64 LCD screen
- PC Speaker-like audio
- 6xAA batteries or a battery pack

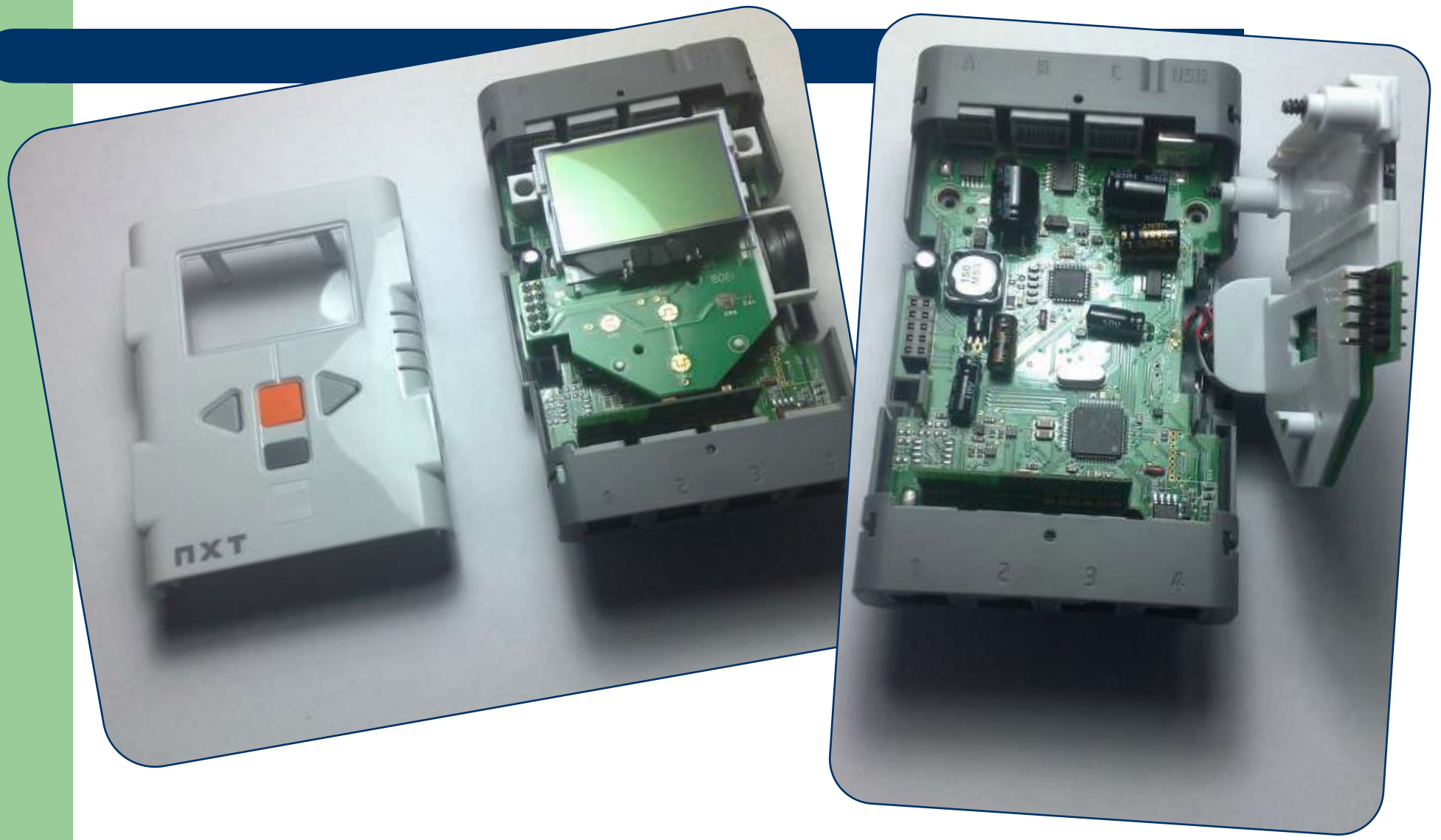
Intro: The Platform (specs.)

- 32-bit AT91SAM7S256 main microprocessor @ 48 MHz
  256KB flash, 64KB RAM
- 8-bit ATmega48 microcontroller @ 4 MHz
  4KB flash, 512bytes RAM
- CSR BlueCore 4 Bluetooth controller @ 26 MHz
  8mbit external flash, 47KB RAM
- 100×64 pixel LCD matrix display
- A single USB 2.0 port full speed (12Mbit/s)
- 4 input ports, 6-wire cable digital platform
  - One port complies to an IEC 61158 Fieldbus protocol
- 3 output ports, 6-wire cable digital platform
- Digital Wire Interface for third-party development of external devices

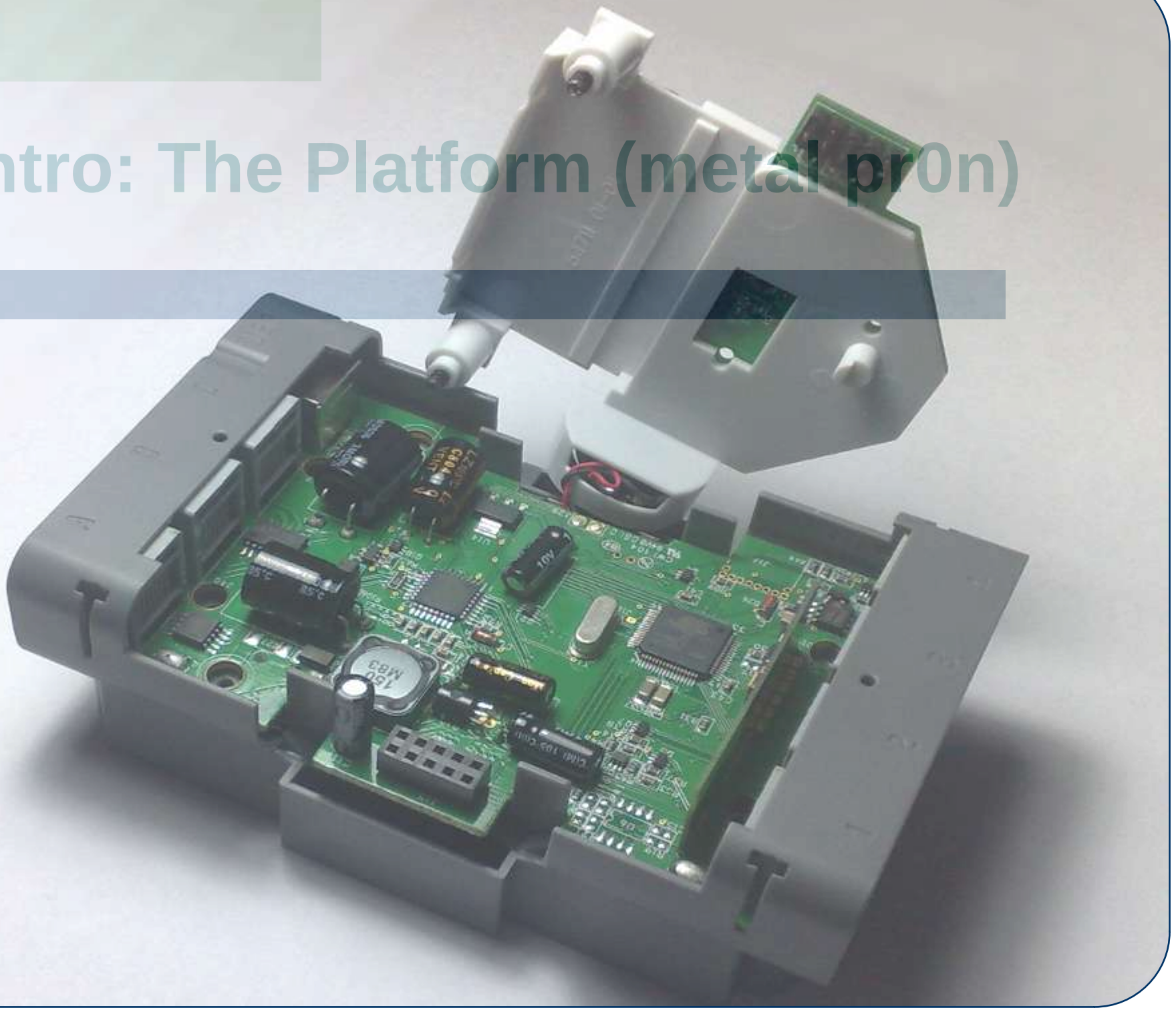
Intro: The Platform (metal pr0n)

Intro: The Platform (metal pr0n)

Intro: The Platform (metal pr0n)



Intro: The Platform (metal pr0n)



Naïve Programming

- Native firmware
  - NXT-G, NBC/NXC
  - RobotC / URBI
  - Simulink's autogenerated C
  - USB/Bluetooth bindings for everything
- leJOS NXJ (Java)
- LEJOS OSK (C/C++)
- pbLua (Lua)

Naïve: Got Python?

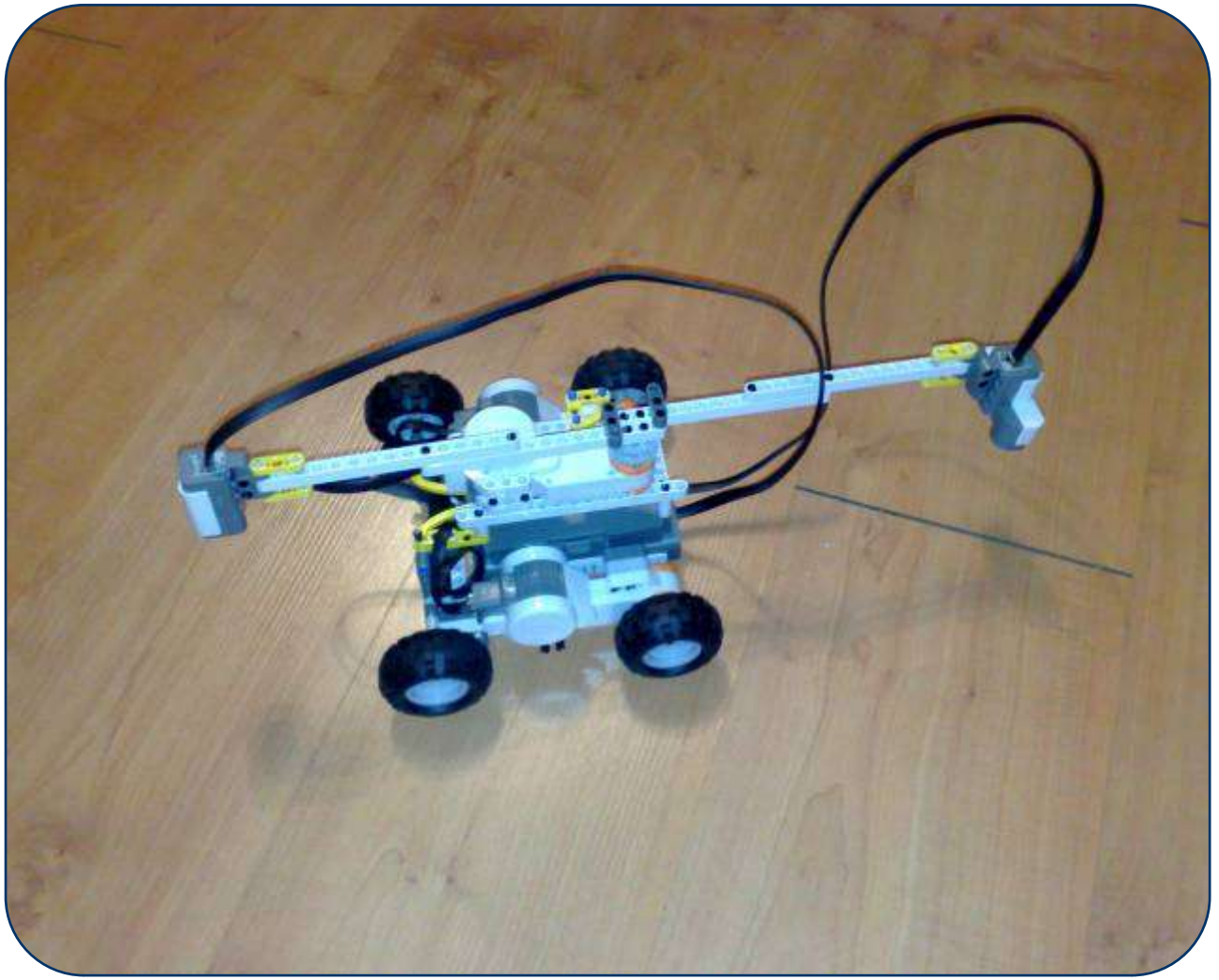
- Douglas P Lau wrote NXT_Python
- Works over USB/Bluetooth
  - Doesn't run on the NXT
- Gives you all the sweet Python lovin':

```
import nxt.locator
try:
    sock = nxt.locator.find_one_brick()
except:
    sock = None
if sock:
    brick = sock.connect()
    name, host, signal_strength, user_flash = brick.get_device_info()
    print ("Found brick: %s" % name)
    sock.close()
```

Naïve: Read sensors, do stuff

Demo: Turn wheel while pressing button

Naïve: Let’s build something!!!



Goal

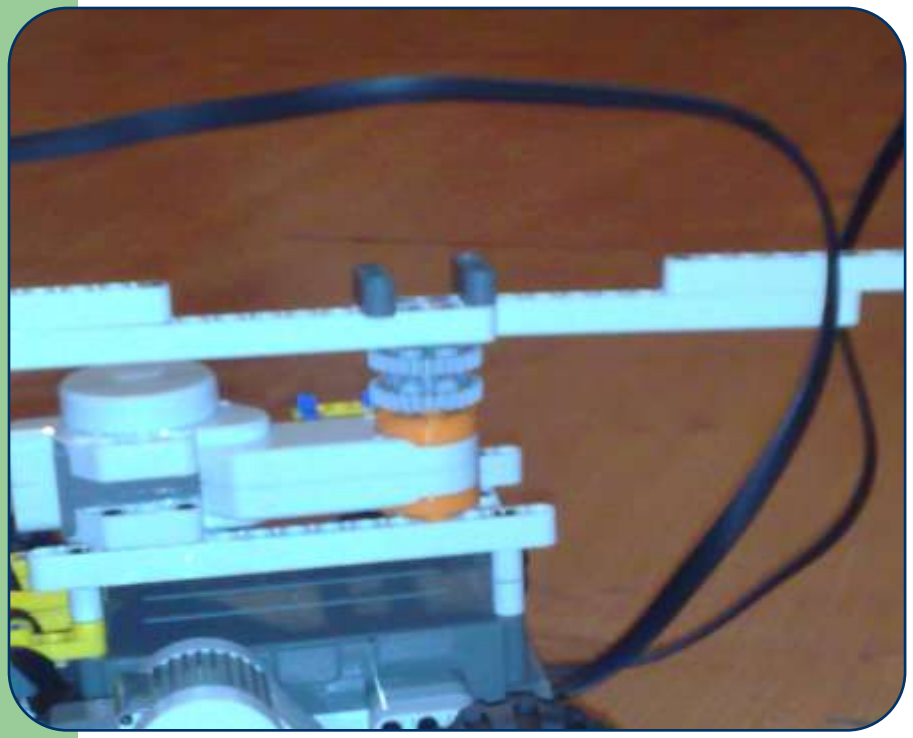
- A four-wheeled platform…
  - Two independently controlled
- Carrying a rotating sensor array…
  - Distance sensor and color sensor
  - Sensors point to ground
- Which will roam a desk…
  - Detecting its shape and not falling off…
- Finding a plastic ball located on it!
  - Attached to it, really
  - Distinguishing it from another ball with a different color

Pubic Apology to Civil Engineers

- I used to think your job is easy
- Repent me of my sin, friends
  - You guys don't even have source control! Gosh!
- Your construction will affect your code
  - More honestly, your coding cycle
- Yes, construction is similar to software design!

Pubic Apology to Civil Engineers

Pubic Apology to Civil Engineers



Naïve: It sucks.

- Robots are about concurrent real time
- Almost immediately code becomes a mess
- *Maybe* I could have made it
  - But then I would be left with just a robot
  - And I already said I like infrastructure
- Two approaches for solution:
  - Threads and locks
  - Event driven
- Actually, make that one just one approach.

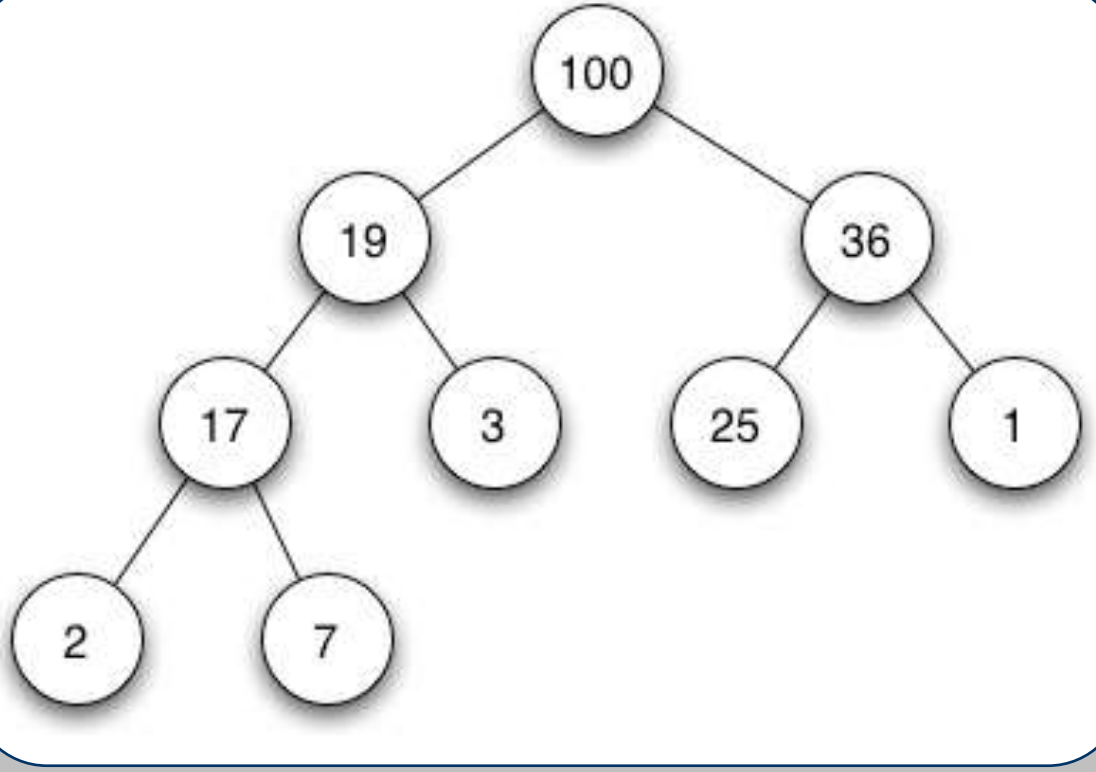
Event Driven Programming

- Python… Event driven… Twisted!
  - Actually, scratch that
- Writing a reactor in Python is dead easy
- An event has four properties:
  - interval
  - data source
  - predicate
  - callback
- We'll use an event heap
  - Sorted by next invocation time

Event Driven Programming

- Python… Event driven… Twisted!
  - Actually, scratch that
- Writing a reactor in
- An even
  - interva
  - data s
  - predic
  - callbac
- We'll us
  - Sorted by next invoc

Operation	Binary
createHeap	Θ(1)
findMin	Θ(*1*)
deleteMin	Θ(lg *n*)
insert	Θ(lg *n*)
decreaseKey	Θ(lg *n*)
merge	Θ(*n*)



Reminder: a heap

Event Driven Programming

User Code

```
reactor.add_sources(InvocationSource(name='sensor array motor source',
                                     interval=self.INTERVAL,
                                     data_source=self.motor.get_position,
                                     predicate=self.motor_predicate,
                                     callback=self.motor_callback,))
```

Reactor Core

```
def invoke_sources(self):
        evaluated_event_sources = []
        while self.event_source_heap:
            if self.event_source_heap[0].timestamp > time.time():
                break
            event_source = heapq.heappop(self.event_source_heap)
            self.logger.log(5, 'invoking source %s' % event_source)
            if event_source.invoke():
                evaluated_event_sources.append(event_source)
        for event_source in evaluated_event_sources:
            event_source.reset()
            self.add_sources(event_source)
```

E... v... e... n... t... ... D... r... i... v... e

- It was *easy* to write
  - …and as slow as an underclocked PDP11
- Potential suspects:
  - **CSR BlueCore 4 Bluetooth chip**
  - Some kind of BT related software stack suckiness
  - The NXT is simply slow? (?!)

Depeche Mode

- "It means hurried fashion or fashion dispatch. I like the sound of that.“
  - So do I
- We have clean reactor/logic separation
- We have the reactor in Python
  - s/Python/pseudo-code/
- Let’s just move the reactor inside

Naïve Programming

- Native firmware
  - NXT-G, NBC/NXC
  - RobotC / URBI
  - Simulink's autogenerated C
  - ~~USB/Bluetooth bindings for everything~~
- leJOS NXJ (Java)
- LEJOS OSK (C/C++)
- pbLua (Lua)

Lua? What's that?!

- Fast, portable, embeddable, powerful, small, free.
- An ultra-light dynamic language
  - Minimalism is king
  - Not features, mechanisms
- Dynamic, simple, multi-paradigm, extensible, first-class-everything,…
  - Kinda reminds me of Python, no?
  - Arguably yes - but with the batteries totally excluded

Lua? What's that?!

- Tables (associative arrays) are king
  - The only data structure
  - Have a few well defined magical properties (meta-table)
  - Many features implemented through them
- Lexical closures
- First-class-everything

```
function foo()
  local bar = 5
  function baz()
    print(bar)
  end
  return baz
end
foo()()
```

pbLua

- Free (as in beer) project by Ralph Hempel
  - free-as-in-for-real-soon-now™ :)
- Ports Lua 5.1 to the NXT
  - firmware replacement
- True to the spirit of Lua
  - “Whatever you want, I ain’t got it and you can make it yourself real easy with X, Y and Z”
- Covers all NXT functionality I needed so far

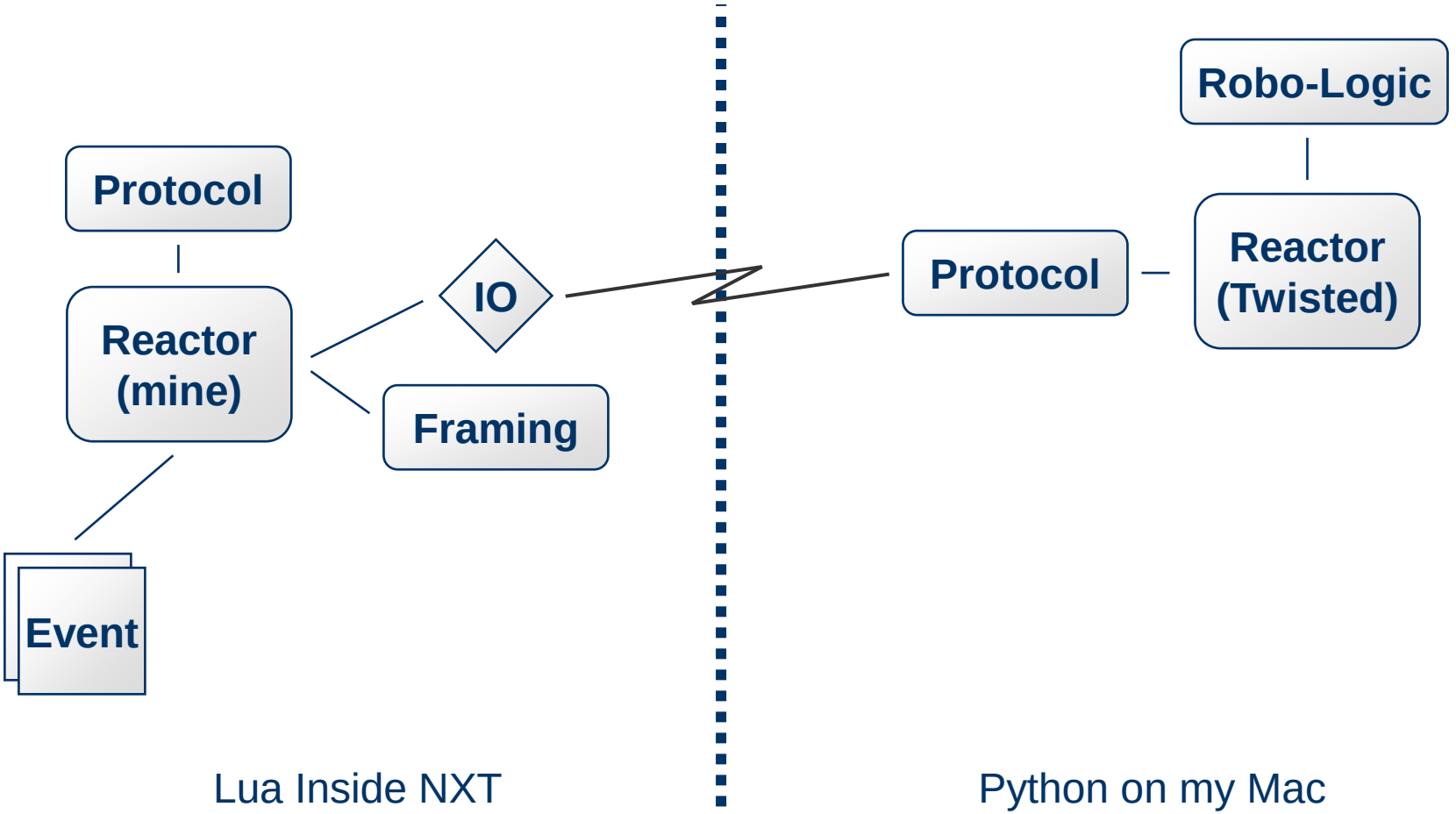
pbLua features

- Interactive interpreter over USB and BT
- Low and mid-level access to NXT features
- Very primitive file system
- XMODEM support for serial data transfer
- Pretty fast

How fast is pretty fast?

Demo: Speed measurement: NXT_Python vs. pbLua

So now what do we do?



Development Cycle

- No easy way to deploy stuff
- Copy-Paste to console is prime method
  - Thank God for pbcopy(1) and pbpaste(1) on OSX
- Saving files is a delicate process
- Debugging is a bitch
- Solutions?
  - Simulator
  - Over-The-Air Code Loading

Over The Air Code Loading

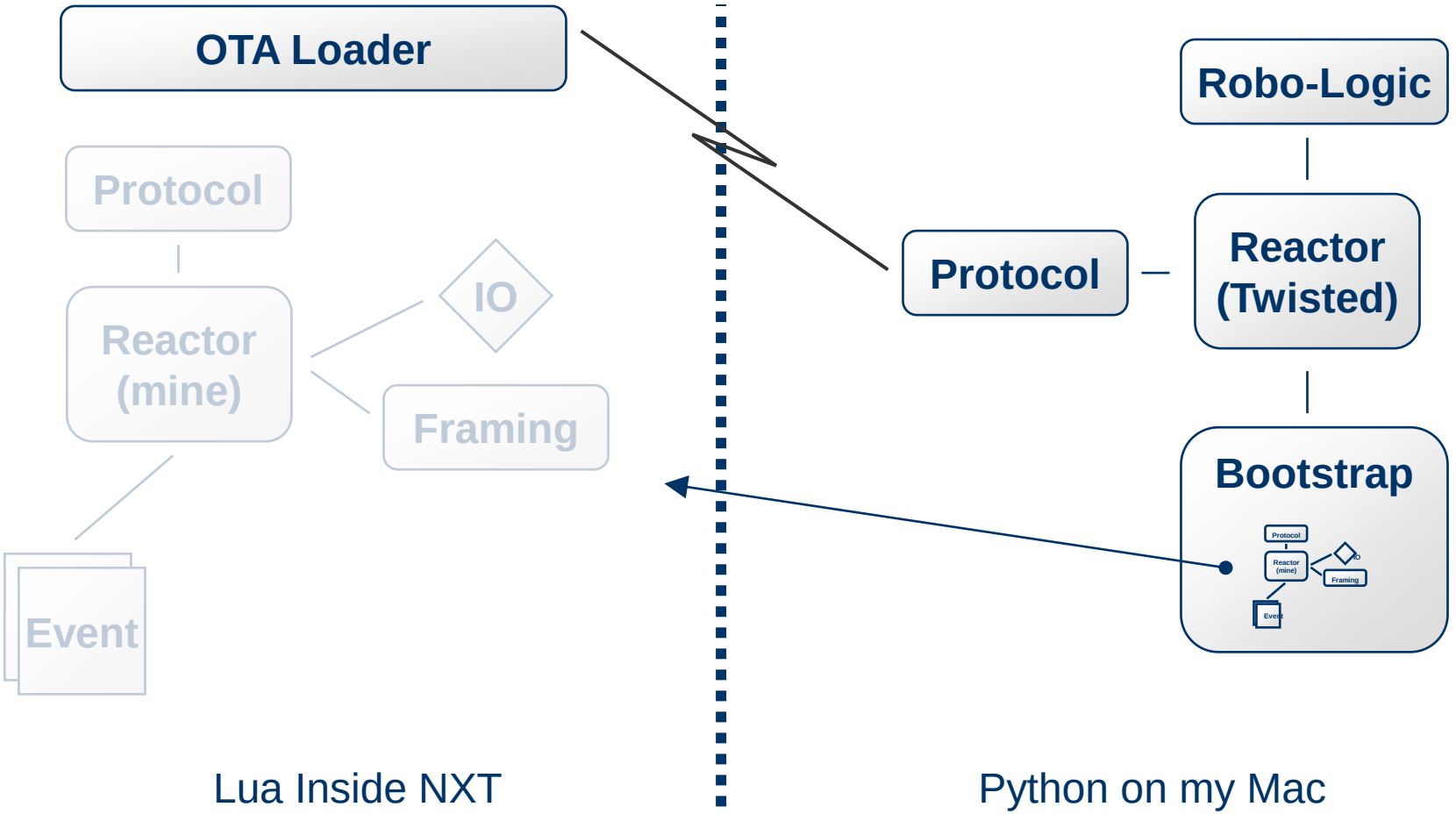
- A simple program
  - Needs no maintenance
  - Should be developed quickly
  - “Only” 250 lines
- Develop and store on Flash filesystem
- Dynamically load bigger programs at will

Key loader components


```
function LoadPieces()
    leftover_bytes = ''
    while (true) do
        local piece, leftover_bytes = BtBlockUntilOnePieceReceived(true, '')
        if piece == 'LoaderExecute' then
            break
        end
        runnable_piece = loadstring(piece)
        if (runnable_piece ~= nil) then
            DisplayTextScroll("running piece")
            runnable_piece()
        else
            error("bad piece :-(")
        end
    end
end
```

```
function OTAInitialize()
    StartupAndGetConnection()
    DisplayTextScroll("wait for chunk")
    LoadPieces()
    DisplayTextScroll("running program")
    LoaderExecute(leftover_bytes)
    nxt.BtDisconnectAll()
    WaitForUser('end: hit [orange]')
end
```

So now what do we do?



Lua Reactor

User

```
reactor:AddEvent(2500, nil, nil, function cb() nxt.SoundTone() end)
```

Reactor Core

```
function Reactor:Run()
    local data
    local current_event
    self.running = true
    while (self.running) do
        self:TestAbortButton()
        data = IO.read()
        if ((data ~= nil) and (#data > 0)) then
            self.framer:DataReceived(data)
        end
        repeat
            current_event = self:ConditionalPopEvent()
            if (current_event ~= nil) then
                self:HandleEvent(current_event)
            end
        until (current_event == nil)
    end
end
```

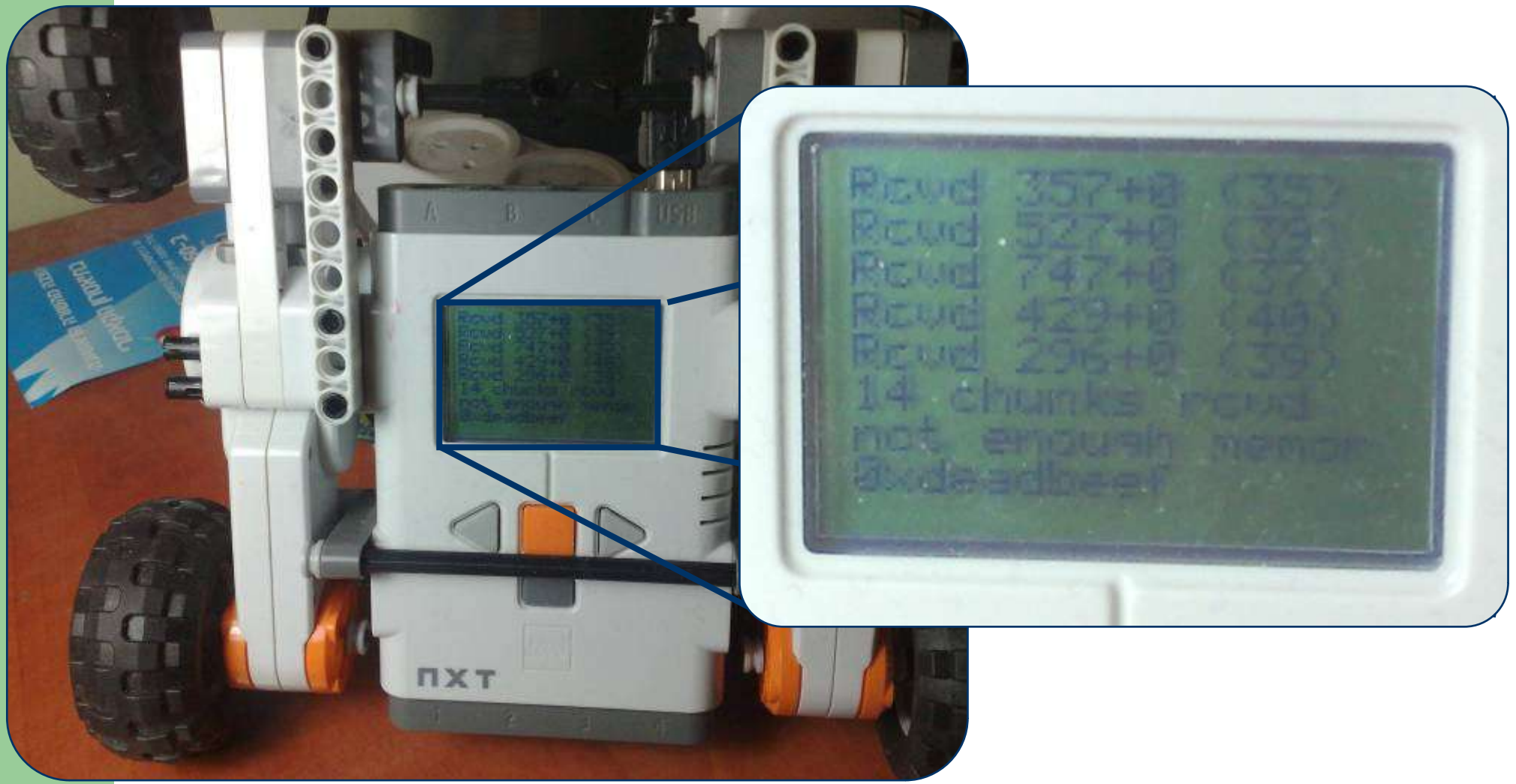
Intro: The Platform (specs.)

- 32-bit AT91SAM7S256 main microprocessor @ 48 MHz
  256KB flash, **64KB RAM**
- 8-bit ATmega48 microcontroller @ 4 MHz
  4KB flash, 512bytes RAM
- CSR BlueC… …controller @ 26 MHz
  8mbit external…
- 100×64 pix…
- A single U…
- 4 input por…
  - One po… …protocol
- 3 output p…
- Digital Wir… …t of external devices

You guys remember how DOS dudes were bitchin' about 640KB RAM?

'nuff said.

All these buzzwords won't fit in 64KB



All these buzzwords won't fit in 64KB

- I started catching MemoryErrors
  - "Remember, it's just a toy!"
    Ralph Hempel, after seeing my loader program
- Memory fragmentation is the real culprit
  - No MMU, eh?
- Fight for every byte
  - Use preprocessor to split sources into tiny chunks
  - Upload no-whitespace code (looks a bit obfuscated)
  - Three-stage loader: set `nil` to anything not in use
  - Obsessive use of `garbagecollect('collect')`

All these buzzwords won't fit in 64KB

```
function ReclaimStage1RAM()
    ReclaimStage0RAM = nil
    StartupAndGetConnection = nil
    BtWaitForConnection = nil
    BtMessageSend = nil
    BtEncodeBuffer = nil
    VerboseCollection()
end


function VerboseCollection()
    old_usage = collectgarbage("count")
    collectgarbage("collect")
    new_usage = collectgarbage("count")
    DisplayTextScroll('reduced ' .. old_usage .. ' to ' .. new_usage)
end
```

Potential solutions for RAM issues

- Just crammin’ it in
- Better heap management tools for pbLua
- eLua ROM-able tables support
- Replacing AT91SAM7S256 with something bigger
  - Not my cup of tea
  - And doesn’t look too easy
  - But I’ll do anything

Framing woes

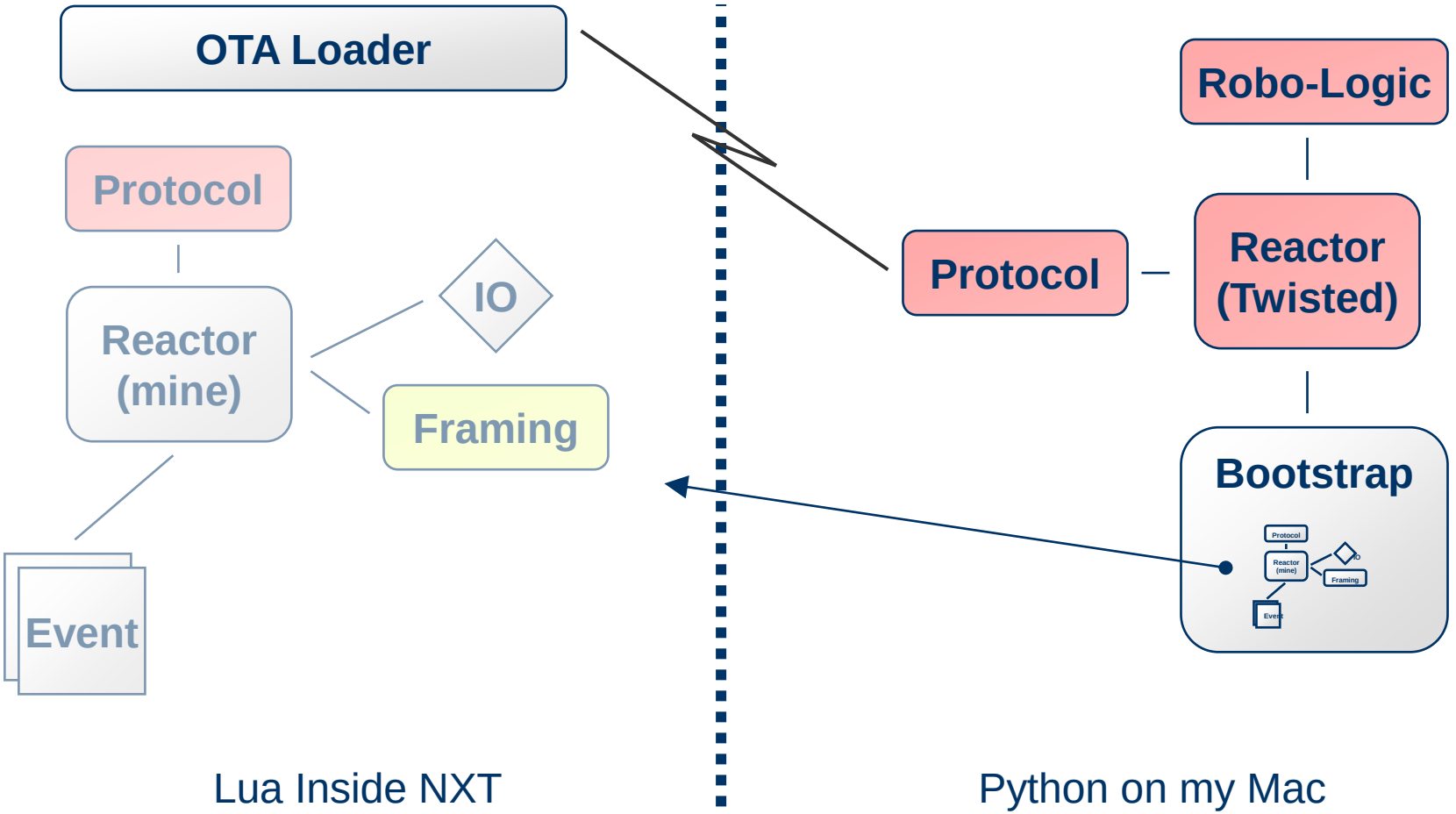
- “Ghost” data appears in new connections
  - After reboots, after disconnects, after a while…
- Strengthen framing mechanism to overcome:
  - Stuff silence with NULLs
  - Add “instance key” made of random 4 digits
    - Messages with wrong key discarded

Framing woes

- "Ghost" data appears in new connections
  - After reboots, after disconnects, after a while…
- Strengthen framing mechanism to overcome

```
function BtBlockUntilChunkHeaderReceived()
    DisplayTextScroll("wait for chunk")
    local buffer = ''
    local chunk_header = nil
    repeat
        buffer = buffer .. BtBlockUntilDataReceived()
        if string.find(buffer, '%z') == nil then
            buffer = ''
        end
        first_match_offset, last_match_offset = string.find(buffer,
                                                '%z%z%z%z%x%x%x%x')
        if first_match_offset ~= nil then
            chunk_header = string.sub(buffer, first_match_offset + 4)
        end
    until chunk_header ~= nil
    -- the next line means: return expected_bytes, data
    return nxt.abs(string.format('0x%s', string.sub(chunk_header, 1, 4))),
                   string.sub(chunk_header, 5)
end
```

So what will I do?



Some closing thoughts

- Overall, maybe pbLua was the wrong choice
- To be honest, maybe NXT wasn't best, either
- It has been incredibly fun
  - I plan on finishing it, time and feasibility allowing
- Far from being an extreme platform
  - Not for hardware junkies nor for algo-heads
  - Probably a good overall first choice
- Excellent open-source support

Closing: Alternatives

- You can get a far stronger box (for more $)
- Imagine the same lecture:
  - With 256MB RAM
  - SSH, SCP, local Python debugger
  - With wi-fi, bluetooth, USB host, GPS, …
- Consider gumstix or beagleboard
  - 150$ – 250$ for computer alone
- My dream project

Closing: Use my source, Luke

- Everything I contributed to this lecture is in the public's domain
- I plan on releasing publicly when (if) things work better and after some cleanups
- Feel free to contact me with questions or requests for source
- Copies of this presentation will be made available to Haifux after the lecure

Thanks!

I’m available at:
yaniv at aknin dot name