Python for less than $7

MicroPython done dirt cheap

Graeme Cross, Planet Innovation

PyCon AU 2015

or “How low can we go?”

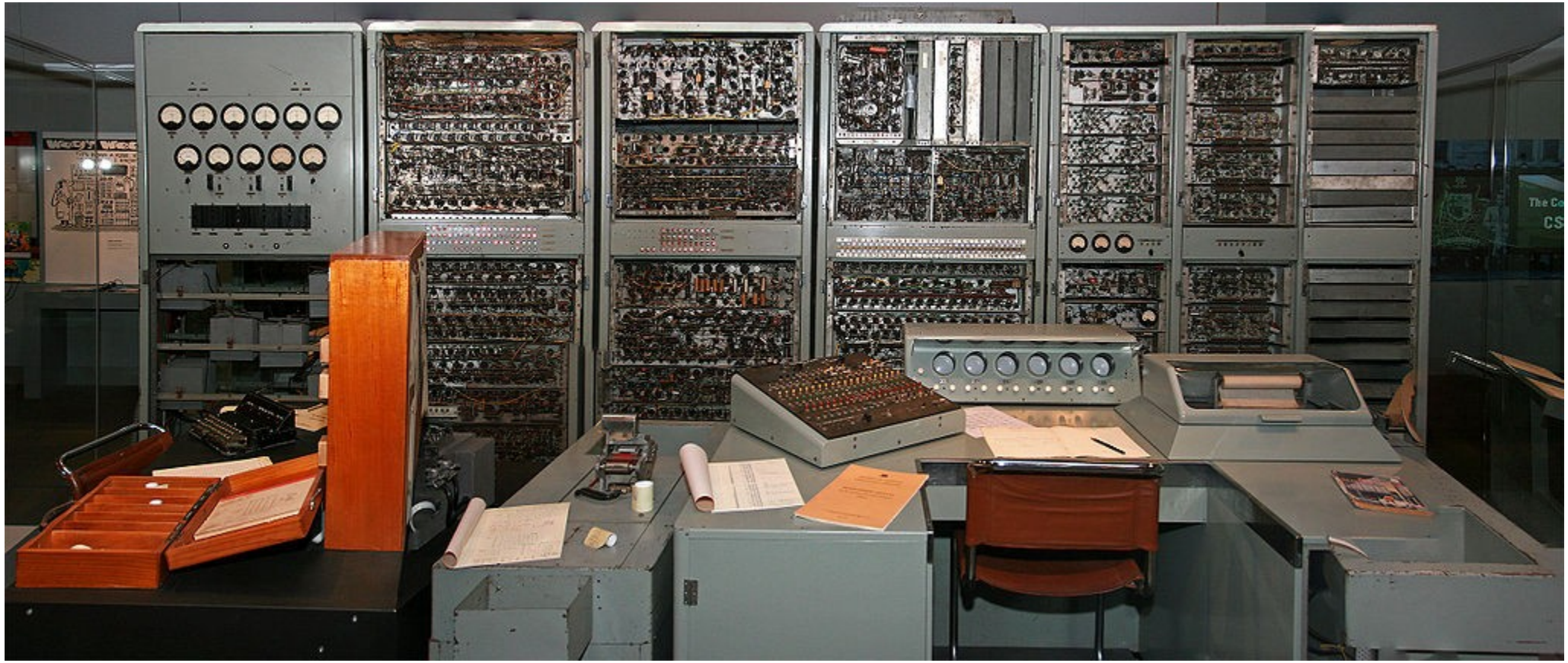
Firstly, a note

- This is a talk about limitations
- But not criticisms
- Lots of tough design decisions = limitations

Cramming Python into tiny places is amazing!!!

And a disclaimer

1949



1965

Electronics, Vol 38, 8, April 19, 1965

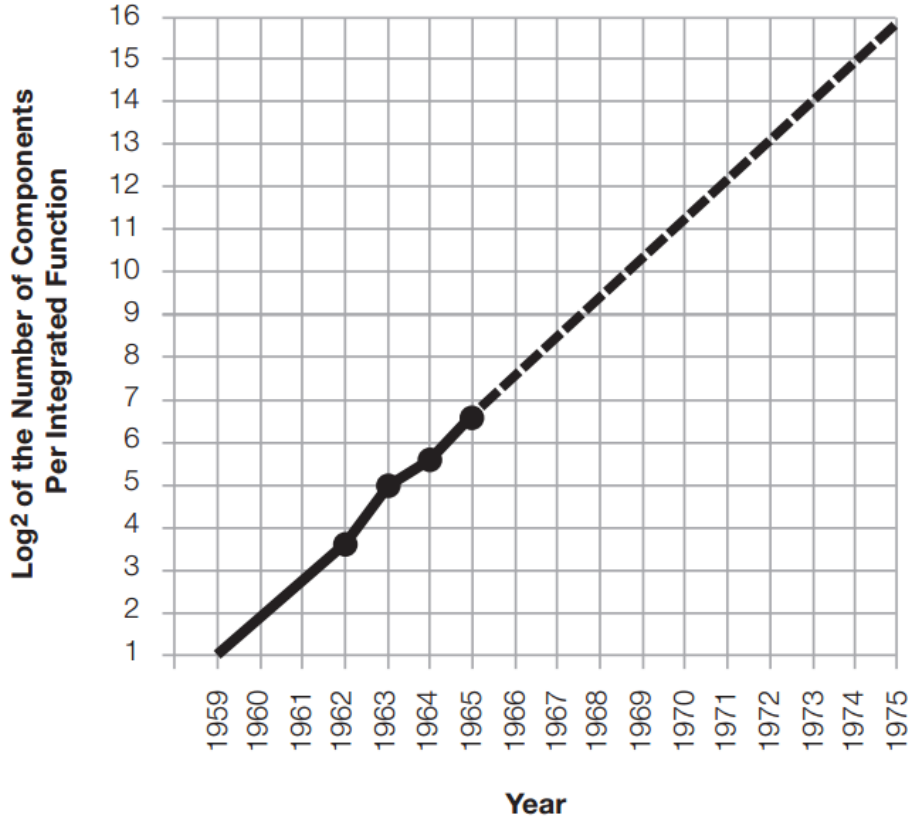
The experts look ahead

Cramming more components onto integrated circuits

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

By Gordon E. Moore

Director, Research and Development Laboratories, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp.

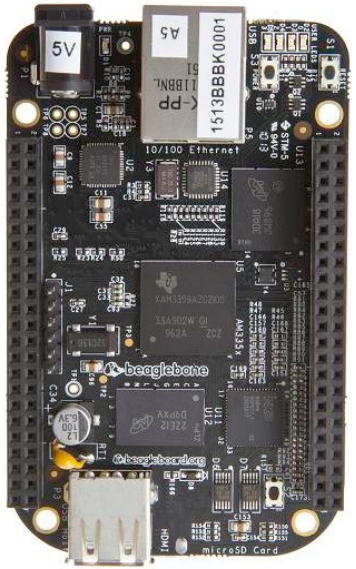


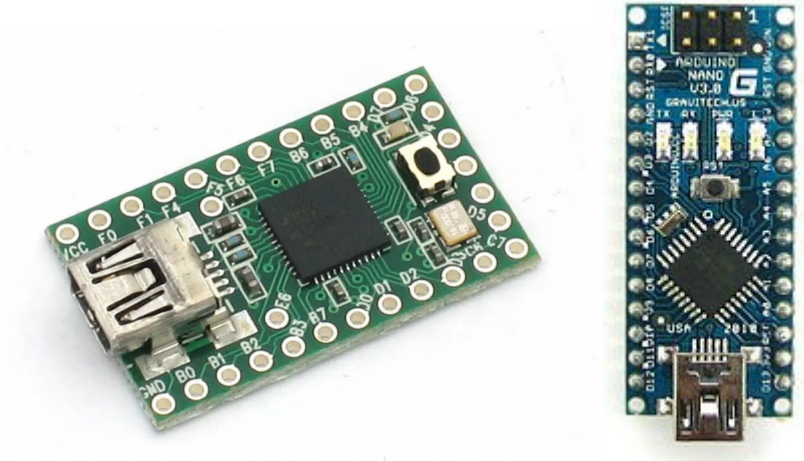
2015

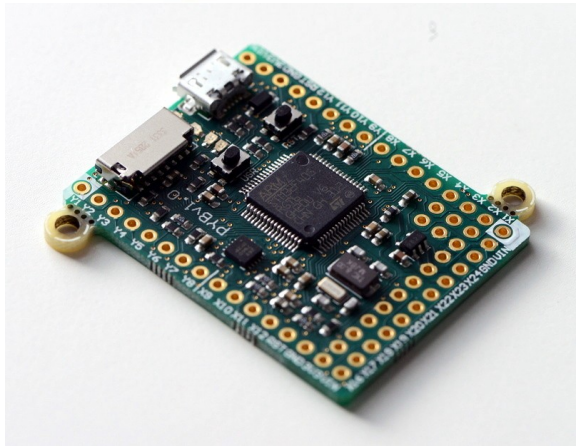
- Processing power
- Core count
- I/O options
- RAM
- Storage options & capacity

- Power consumption
- Price
- Footprint

Microcontrollers for hackers and makers



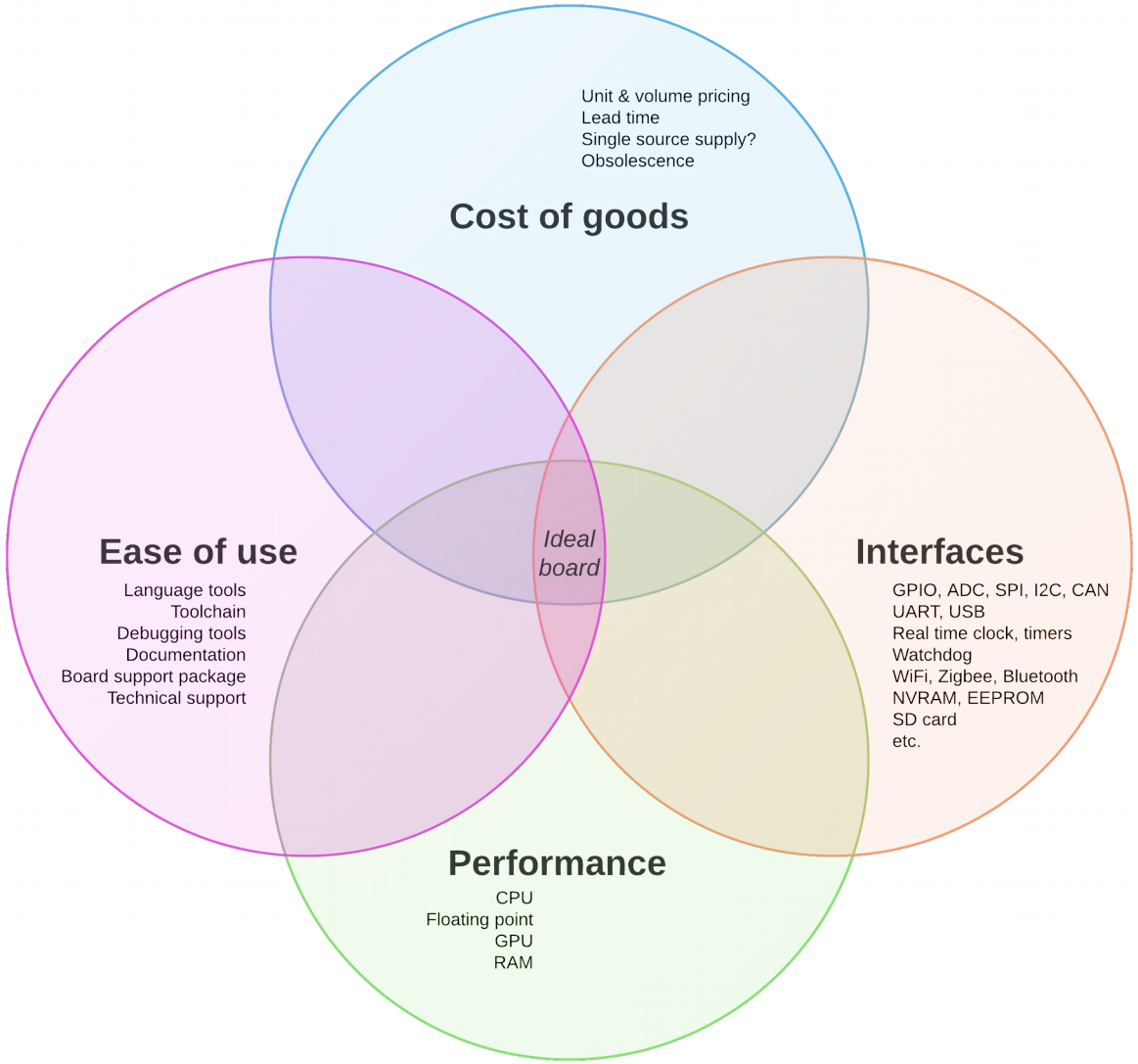




Lots of choices:

- Raspberry Pi
- BeagleBone Black
- Arduino
- Teensy
- pyboard
- Particle (Spark) Core & Photon
- C.H.I.P.
- And lots more...

The prototyping sweet spot



Why Python on a micro?

- Easy to learn
- REPL = rapid prototyping & feedback
- No compile cycle
- Good for engineers who already know Python on PCs
- Standard library (when available)
- Ecosystem
- Community

Why not Python on a micro?

- Performance
- Memory footprint
- Garbage collection versus real-time

MicroPython

- Python 3.4 for microcontrollers
- Bare metal = no O/S
- Supports multiple platforms
- Memory footprint: 75 – 260KB
- Can run in 8KB RAM
- Can optimise with “emitter” decorators
- Good documentation
- Active community

MicroPython & libraries

- Doesn't have a “normal” standard library
  - Micropython-lib (https://github.com/micropython/micropython-lib)
  - Non-monolithic standard library – install only what you need (+ dependencies)
- Is not bytecode compatible with CPython
- Has some board support modules (“pyb”, “esp”)
- upip: MicroPython specific lite version of pip

MicroPython limitations

- Not 100% compatible with CPython

  https://github.com/micropython/micropython/wiki/Differences
- Subset of the CPython object model
- Subset of introspection features
- print() is limited
- Unicode is a WIP

MicroPython ports

- pyboard
- Teensy
- PIC (16 bit)
- TI CC3200
- STM32F407
- “bare” ARM
- Unix
- ESP8266

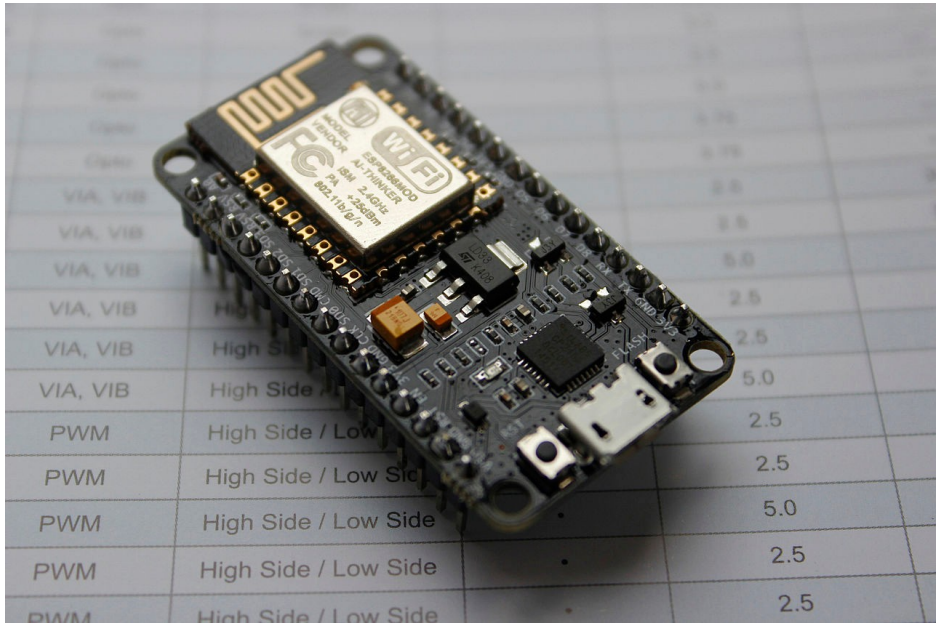
The ESP8266

- Very low cost WiFi SOC manufactured by Espressif
  - UART-WiFi bridge with user-programmable micro
  - Tensilica Xtensa LX106 32-bit RISC CPU running at 80 MHz
  - RAM: 64 KiB (instructions), 96 KiB (data)
  - 512 KiB to 4 MiB external QSPI flash
  - 802.11 b.g.n WiFi with WPA/WPA2 authentication
    - 300 metre wifi range with PCB antenna
  - 16 GPIO pins, 10-bit ADC, SPI, I^2C, I^2S (sharing pins with GPIO)
  - Dedicated UART + a transmit-only UART
  - ~ 200mA power consumption
  - (Partially?) open source SDK with GCC toolchain

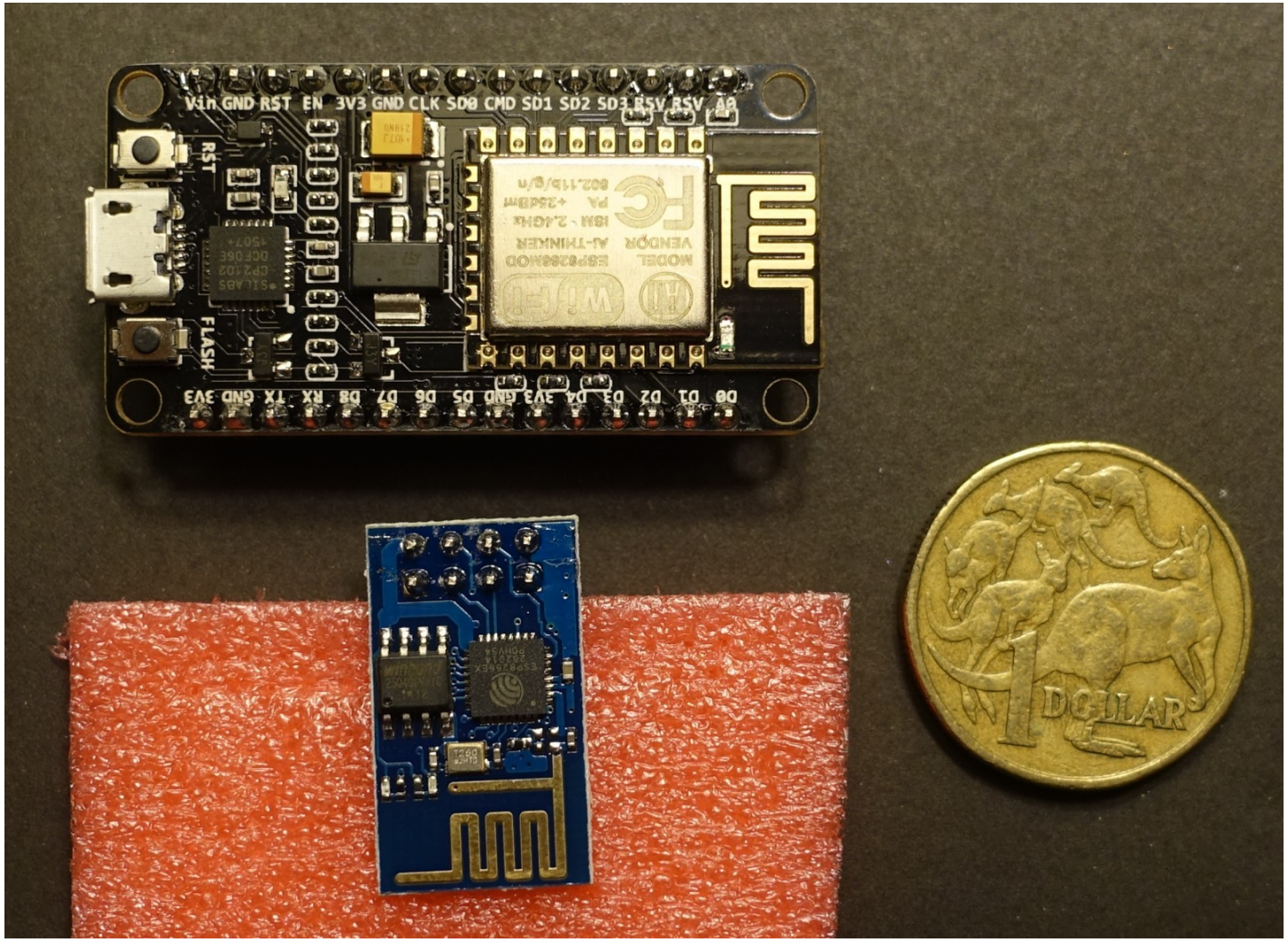
ESP8266 variants

The NodeMCU

- ESP8266 based
- GPIO, PWM, I2C, 1 Wire and ADC
- Open-sourced
- Lua scripting with event-driven networking library
- Supports MicroPython & Arduino
- USD 12.95



NodeMCU & ESP12 sizes



NodeMCU & MicroPython

- To do anything useful, you need:
  - The full toolchain
  - A firmware install tool
  - To read docs, source code, spec sheets & circuit diagrams

```
$ ./esptool.py --port /dev/ttyUSB0 write_flash --flash_mode dio 0x00 ./firmware-combined.bin
Connecting...
Erasing flash...
Writing at 0x0004d800... (100 %)
Leaving...
```

ESP8266/MicroPython limitations

- No mountable flash filesystem
- Barebones wifi implementation
- Partial standard library support
- Minimal documentation so far
- No floating point support (yet)
- You need the toolchain to do any meaningful work
- **We now know how low we can go!**

NodeMCU & MicroPython

Pros:

- Easy (GPIO) hardware prototyping
- Open source hardware and firmware
- Cheap

Cons:

- Different hardware configs
- Limited MicroPython functionality
- Minimal documentation

Demo

What we learnt...

- ESP8266/MicroPython is suitable for simple use cases
  - Lots of potential
  - Consider Lua or Arduino interfaces
- MicroPython on the STM32F407 Discovery?
  - More powerful than ESP8266, but no onboard wifi
  - USD 19
- The pyboard?
  - If need for Python outweighs cost

The embedded Python sweet-spot?

- What is your most important driver?
  - Cost?
  - Usability?
  - Speed, form factor, boot time, networking, I/O… ?
- A usable Python-specific embedded solution: the pyboard
- For a flexible solution: the Raspberry Pi
- For lowest cost solution: ESP8266
- For a hacking challenge: ESP8266

Useful resources

MicroPython: https://github.com/micropython/micropython

ESP8266 forum: http://www.esp8266.com/

ESP8266 quick start: http://benlo.com/esp8266/esp8266QuickStart.html

NodeMCU:

- http://nodemcu.com/index_en.html
- https://github.com/nodemcu

Building & running MicroPython on the ESP8266:
https://learn.adafruit.com/building-and-running-micropython-on-the-esp8266

This talk

http://www.curiousvenn.com/2015/07/pycon-au-2015-slides/

Python for less than $7

MicroPython done dirt cheap

Graeme Cross, Planet Innovation

PyCon AU 2015

Agenda

- Revisit 1949 and 1965
- Look at common micro prototyping / hacking options for makers
- Study what makes the ESP8266 chip interesting and the NodeMCU in particular
- Introduce MicroPython
- Do what I advise presenters to never do: a live demo
- Discuss some of the limitations of MicroPython on ESP8266
- What options should people use for low-cost embedded Python programming?
- Resources for more information

or “How low can we go?”

How simple, small, cheap can we go with a microcontroller that can run Python and still be usable for real-world interfacing, networking and straight-forward programming?

Source:

https://www.flickr.com/photos/reemul/7338644262/

Firstly, a note

- This is a talk about limitations
- But not criticisms
- Lots of tough design decisions = limitations

Cramming Python into tiny places is amazing!!!

- I am going to talk about exploration and limitations
- This is not a talk of criticism. At all.
- Cramming Python into tiny places is amazing
- Lots of tough design decisions = limitations
- We should be in **complete** awe of the engineering involved in developing MicroPython and porting it to such challenging embedded targets

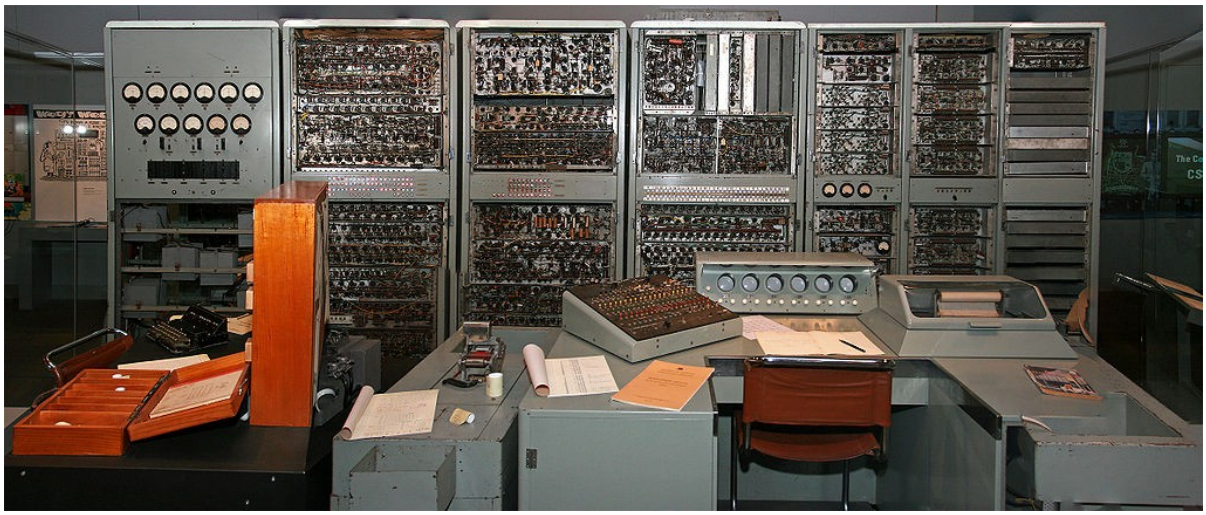
And a disclaimer

Prices quoted here are in USD, given the fluctuation in the AUD!

Sourced from:

https://au.finance.yahoo.com/q/bc?s=AUDUSD=X&t=1y&l=on&z=l&q=l&c=

1949



CSIRAC

- One of the first computers in the world, first in Australia
- The earliest remaining first generation computer
- Fastest computer of its time
- Operational from 1949 – 1964
- Clock speed: 500 – 1000 Hz
- RAM: 2000 bytes
- Weight: 2500 kg
- Power consumption: 30kW

http://museumvictoria.com.au/melbournemuseum/whatson/current-exhibitions/csirac/

1965

Electronics, Vol 38, 8, April 19, 1965

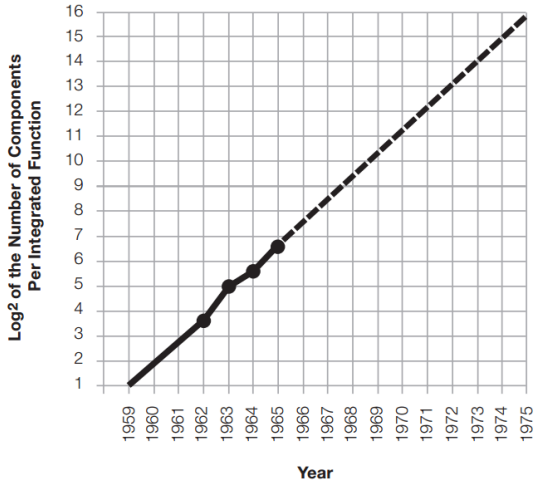
The experts look ahead

Cramming more components onto integrated circuits

With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65,000 components on a single silicon chip

By Gordon E. Moore

Director, Research and Development Laboratories, Fairchild Semiconductor division of Fairchild Camera and Instrument Corp.



Moore's Law

- Published in 1965 (doubling every year), revised in 1975 to every two years
  - Intel 4004: 1971 with 2,300 transistors
  - Pentium: mid 90's = 3.1 million transistors
  - Core i5 = 1.9 billion transistors in Broadwell-U, dual core, 14nm
- The cost of an integrated circuit transistor has fallen from:
  - 1965: USD 30 (in today's dollars)
  - 2015: ~ 1 nano-dollar
- Below 100nm fab, Dennard's scaling rule breaks down
- Below 20nm fab, Moore's Law breaks down
- 22nm is the current PPAC sweet spot (price, performance, area, cost)
- Lots of microcontrollers are manufactured at 90 - 45nm
- Enormous cost benefits at scale = plummeting prices for sophisticated micros

2015

- Processing power
- Core count
- I/O options
- RAM
- Storage options & capacity

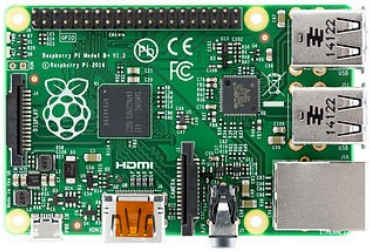
- Power consumption
- Price
- Footprint

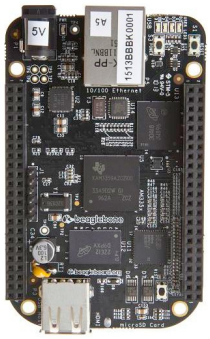
Lots of computer hardware dimensions have dramatically increased or decreased in value over the last five decades.

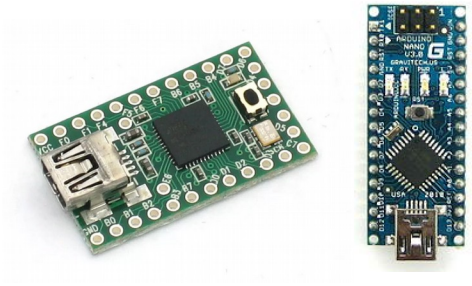
What is most interesting is the revolution in the embedded space:

- Low cost
- Powerful
- Low power
- Lots of interfacing options, for sensing, control and comms

Microcontrollers for hackers and makers







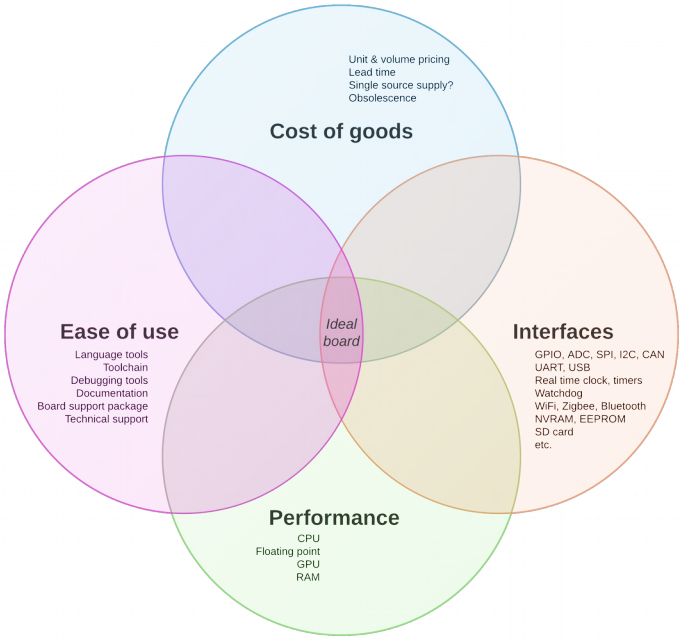
Lots of choices:

- Raspberry Pi
- BeagleBone Black
- Arduino
- Teensy
- pyboard
- Particle (Spark) Core & Photon
- C.H.I.P.
- And lots more…

Lots of options:

- "Bare metal" versus RTOS vs Linux (or other operating system)
- Onboard networking (ethernet and/or wifi)?
- Onboard storage?
- CPU family
- Graphics capabilities
- I/O capabilities: GPIO, ADC, DAC, SPI, I2C, CAN,
- Real-time clock
- Expansion options (e.g. Arduino "shields")
- Availability
- Price
- Size
- Firmware installation
- Startup ("boot") duration
- Development tools
- Programming languages
- Open source? Hardware, firmware, tools, etc.
- Community
- Documentation
- Single source supplier?
- Commercial volumes? Supply agreements?

The prototyping sweet spot



For prototyping and commercialisation, we need to make conscious tradeoffs.

At work, we are interested in all of these areas to varying degrees, depending on the project's requirements and the type of product.

In the “connected health” space, low-cost and low-power networked embedded devices are needed; but can we also leverage the power of Python on these devices for rapid prototyping and possible deployment?

Why Python on a micro?

- Easy to learn
- REPL = rapid prototyping & feedback
- No compile cycle
- Good for engineers who already know Python on PCs
- Standard library (when available)
- Ecosystem
- Community

- Easier to learn and teach than C/C++
- REPL = excellent for rapid prototyping & feedback
- No compile cycle = faster development
- Good for engineers who already know Python
- Good standard library (when available)
- Excellent ecosystem
- Fantastic community

Why not Python on a micro?

- Performance
- Memory footprint
- Garbage collection versus real-time

- Significantly slower than C/C++
- Large memory footprint
- Garbage collection can subvert real-time requirements

MicroPython

- Python 3.4 for microcontrollers
- Bare metal = no O/S
- Supports multiple platforms
- Memory footprint: 75 – 260KB
- Can run in 8KB RAM
- Can optimise with “emitter” decorators
- Good documentation
- Active community

- Python 3.4 for microcontrollers, 2013 Kickstarter campaign
- Bare metal = no operating system
- Initial target: the “pyboard”, now supports multiple platforms
- Memory footprint: 75 – 260KB
- Can run in 8KB RAM
- Can optimise with “emitter” decorators
- Good documentation, active community
- Lots of opportunities to contribute

MicroPython & libraries

- Doesn't have a “normal” standard library
  - Micropython-lib (https://github.com/micropython/micropython-lib)
  - Non-monolithic standard library – install only what you need (+ dependencies)
- Is not bytecode compatible with CPython
- Has some board support modules (“pyb”, “esp”)
- upip: MicroPython specific lite version of pip

If your hardware target doesn't support some form of user-accessible storage (e.g. USB mass storage or microSD card), then you will need to build the libraries into the firmware.

This is not ideal for rapid prototyping!

MicroPython limitations

- Not 100% compatible with CPython
  https://github.com/micropython/micropython/wiki/Differences
- Subset of the CPython object model
- Subset of introspection features
- print() is limited
- Unicode is a WIP

Squeezing MicroPython down into less memory means some tough design decisions.

Read the “Differences” page on the wiki to get the full details of implementation differences and limitations.

Note that different MicroPython targets may have other differences or limitations.

print() doesn't do recursive printing.

MicroPython ports

- pyboard
- Teensy
- PIC (16 bit)
- TI CC3200
- STM32F407
- “bare” ARM
- Unix
- ESP8266

- pyboard
  - STM32F405RG CPU (168MHz Cortex M4F) with HW FP, 192k RAM & 1M ROM
  - 29 GPIO pins, 3 ADC, 2 DAC, 4 LEDs, 2 switches, 3 axis accelerometer
  - Real time clock (RTC)
  - MicroSD card + MicroUSB (including DFU bootloader)
  - Expansion boards: “skins”
  - But no inbuilt wifi
  - Cost: USD 45 (AdaFruit)
- Teensy, PIC (16 bit), TI CC3200, STM32F407 Discovery,“bare” ARM & Unix
- ESP8266

The ESP8266

- Very low cost WiFi SOC manufactured by Espressif
  - UART-WiFi bridge with user-programmable micro
  - Tensilica Xtensa LX106 32-bit RISC CPU running at 80 MHz
  - RAM: 64 KiB (instructions), 96 KiB (data)
  - 512 KiB to 4 MiB external QSPI flash
  - 802.11 b.g.n WiFi with WPA/WPA2 authentication
    - 300 metre wifi range with PCB antenna
  - 16 GPIO pins, 10-bit ADC, SPI, I^2C, I^2S (sharing pins with GPIO)
  - Dedicated UART + a transmit-only UART
  - ~ 200mA power consumption
  - (Partially?) open source SDK with GCC toolchain

Can be used:

- As a WiFi module for a larger microcontroller (e.g. Arduino)
- On it's own with the Espressif SDK

ESP8266 variants

Range of modules, boards & dev kits.
Variations include:

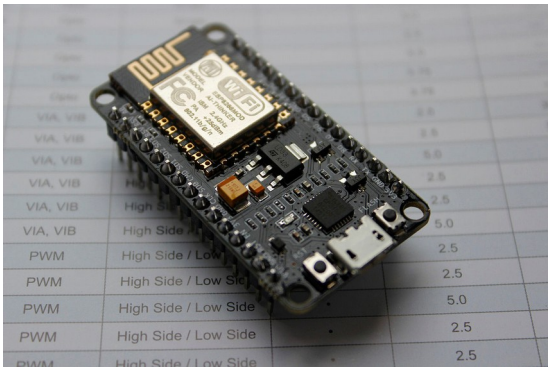
- Pin count & form factor
- Antenna design & FCC compliance

One-off board & module costs:

- USD 7 when I submitted this talk's abstract
- AI-Thinkers ESP-01: ~ USD 3
- NodeMCU: ~ USD 13

The NodeMCU

- ESP8266 based
- GPIO, PWM, I2C, 1 Wire and ADC
- Open-sourced
- Lua scripting with event-driven networking library
- Supports MicroPython & Arduino
- USD 12.95

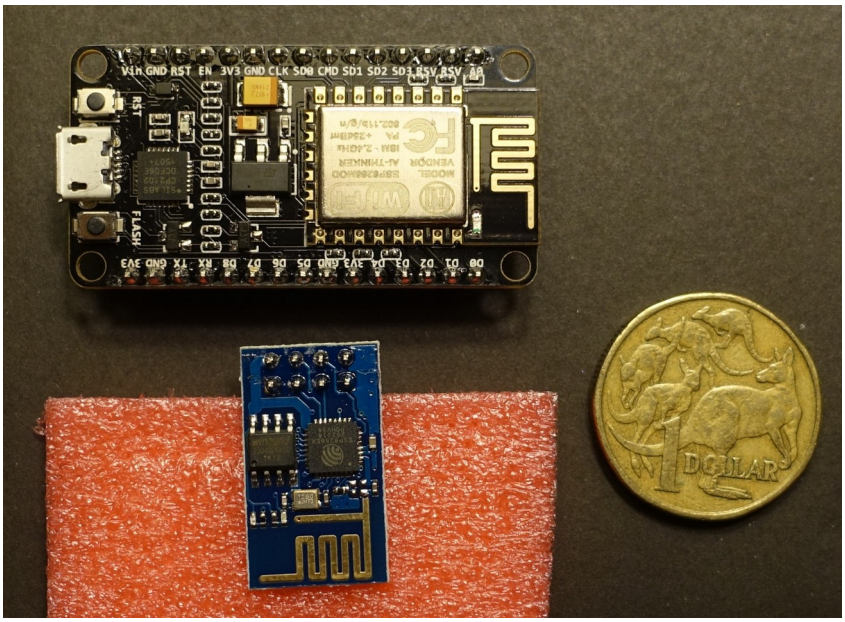


http://nodemcu.com/index_en.html

Photo sourced from:

"NodeMCU DEVKIT 1.0" by Vowstar - Own work. Licensed under CC BY-SA 4.0 via Wikimedia Commons - https://commons.wikimedia.org/wiki/File:NodeMCU_DEVKIT_1.0.jpg#/media/File:NodeMCU_DEVKIT_1.0.jpg

NodeMCU & ESP12 sizes



These boards are small; ideal for embedding in all sorts of “IoT” devices

NodeMCU & MicroPython

- To do anything useful, you need:
  - The full toolchain
  - A firmware install tool
  - To read docs, source code, spec sheets & circuit diagrams

```
$ ./esptool.py --port /dev/ttyUSB0 write_flash --flash_mode dio 0x00 ./firmware-combined.bin
Connecting...
Erasing flash...
Writing at 0x0004d800... (100 %)
Leaving...
```

You need the toolchain to build a version of firmware with your start up scripts, modules, etc

That “--flash_mode dio” command line flag is essential for some versions of the NodeMCU – if you can install firmware but can't get serial/USB comms working at any baud rate, try this

To build the toolchain:
https://github.com/pfalcon/esp-open-sdk.git

To install the firmware:
https://github.com/themadinventor/esptool.git

ESP8266/MicroPython limitations

- No mountable flash filesystem
- Barebones wifi implementation
- Partial standard library support
- Minimal documentation so far
- No floating point support (yet)
- You need the toolchain to do any meaningful work
- **We now know how low we can go!**

NodeMCU & MicroPython

Pros:

- Easy (GPIO) hardware prototyping
- Open source hardware and firmware
- Cheap

Cons:

- Different hardware configs
- Limited MicroPython functionality
- Minimal documentation

Pros:

- Easy to prototype hardware on
- Open source hardware and firmware
- Cheap

Cons:

- Variable hardware configs = set up and debug challenges
- MicroPython functionality is very limited (today)
  - Wifi functionality
  - Networking
  - Floating point
  - I2C, SPI, etc
- Minimal documentation (you need strong GoogleFu)
- This isn't a problem if you know this and go in prepared.

Demo

What we learnt...

- ESP8266/MicroPython is suitable for simple use cases
  - Lots of potential
  - Consider Lua or Arduino interfaces
- MicroPython on the STM32F407 Discovery?
  - More powerful than ESP8266, but no onboard wifi
  - USD 19
- The pyboard?
  - If need for Python outweighs cost

- ESP8266/MicroPython is only suitable for simple use cases
  - Lots of potential
  - Plenty of opportunities to contribute!
  - We are planning to contribute documentation & patches
  - Otherwise use the Lua or Arduino interfaces
- Consider MicroPython on the STM32F407 Discovery?
  - More powerful than ESP8266, but no onboard wifi
  - USD 19 cost
- Consider the pyboard if need for Python outweighs cost

If you want a working Python implementation today for embedded control, the Raspberry Pi is the most convenient low-cost solution IF these factors are acceptable:

- Boot time
- Operating system (updates, security issues, learning curve, complexity, etc)
- Power consumption
- Size

The embedded Python sweet-spot?

- What is your most important driver?
  - Cost?
  - Usability?
  - Speed, form factor, boot time, networking, I/O… ?
- A usable Python-specific embedded solution: the pyboard
- For a flexible solution: the Raspberry Pi
- For lowest cost solution: ESP8266
- For a hacking challenge: ESP8266

There are too many dimensions here to tell you what is THE solution for you.

The good news is that we have lots of options!

Useful resources

MicroPython: https://github.com/micropython/micropython

ESP8266 forum: http://www.esp8266.com/

ESP8266 quick start: http://benlo.com/esp8266/esp8266QuickStart.html

NodeMCU:

- http://nodemcu.com/index_en.html
- https://github.com/nodemcu

Building & running MicroPython on the ESP8266:
https://learn.adafruit.com/building-and-running-micropython-on-the-esp8266

A few resources to get you started.

The Adafruit tutorial has a lot of good advice and links to other resources.

This talk

http://www.curiousvenn.com/2015/07/pycon-au-2015-slides/

Thanks!
Questions?