The Actor Model applied to the Raspberry Pi and the Embedded Domain

Omer Kilic || @OmerK

omer@erlang-solutions.com

Agenda

- Current state of Embedded Systems
- Overview of the Actor Model
- Erlang Embedded Project
- Modelling and developing systems using Erlang
- Experiments with the Raspberry Pi
- Future Explorations
- Q & A

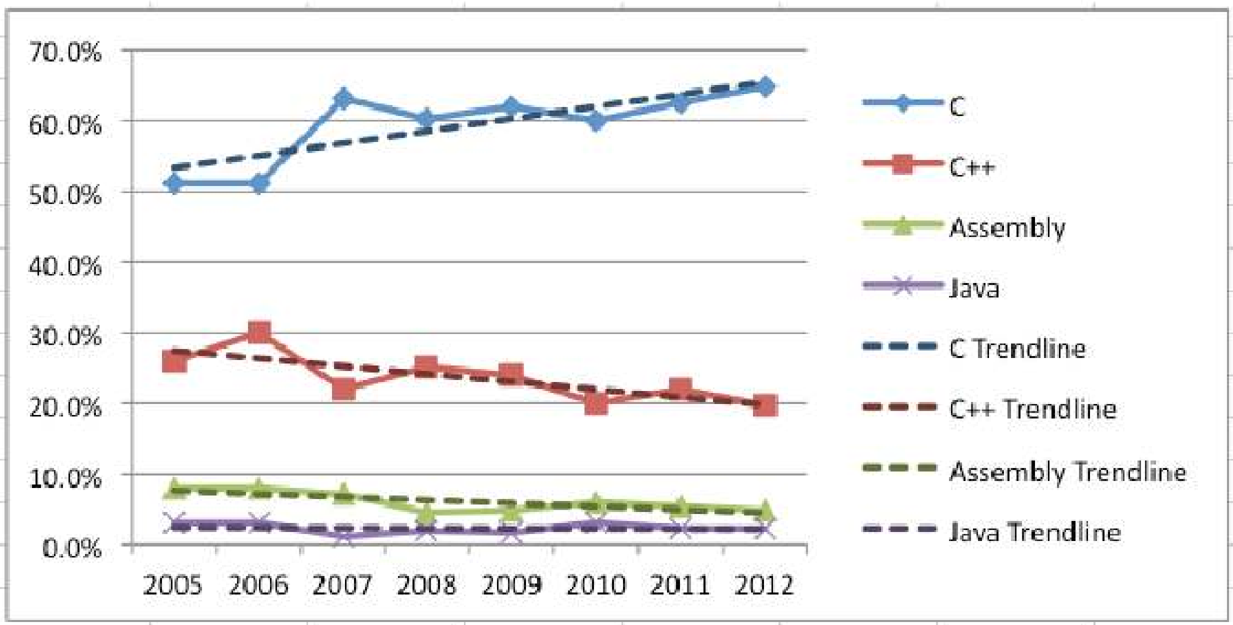
Embedded Systems

“An embedded system is a computer system designed for specific control functions within a larger system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs.

- Infinite Wisdom of Wikipedia

#include <stats.h>

The four languages most often reported as the primary language for embedded projects for the years 2005 to 2012, along with linear trendlines.



Source: http://embedded.com/electronics-blogs/programming-pointers/4372180/Unexpected-trends

Current Challenges

- Complex SoC platforms
- “Internet of Things”
  - Connected and distributed systems
- Multicore and/or heterogeneous devices
- Time to market constraints

Embedded Systems

- Bare Metal
  - No underlying OS or high level abstractions
- RTOS
  - Minimal interrupt and switching latency, scheduling guarantees, minimal jitter
- Embedded Linux
  - Slimmed down Linux with hardware interfaces

RTOS Concepts

- Notion of “tasks”
- OS-supervised interprocess messaging
  - Shared memory
- Mutexes/Semaphores/Locks
- Scheduling
  - Pre-emptive: event driven
  - Round-robin: time multiplexed

Embedded Linux

- Not a new concept, increased popularity due to abundant supply of cheap boards
  - Raspberry Pi, Beagleboard/Beaglebone, Gumstix et al.
- Familiar set of tools for software developers, new territory for embedded engineers
  - No direct mapping for RTOS concepts, especially tasks
- Complex device driver framework
  - Here be dragons

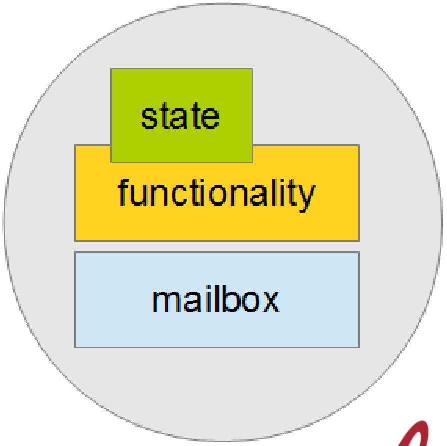
Erlang SOLUTIONS

Actor Model

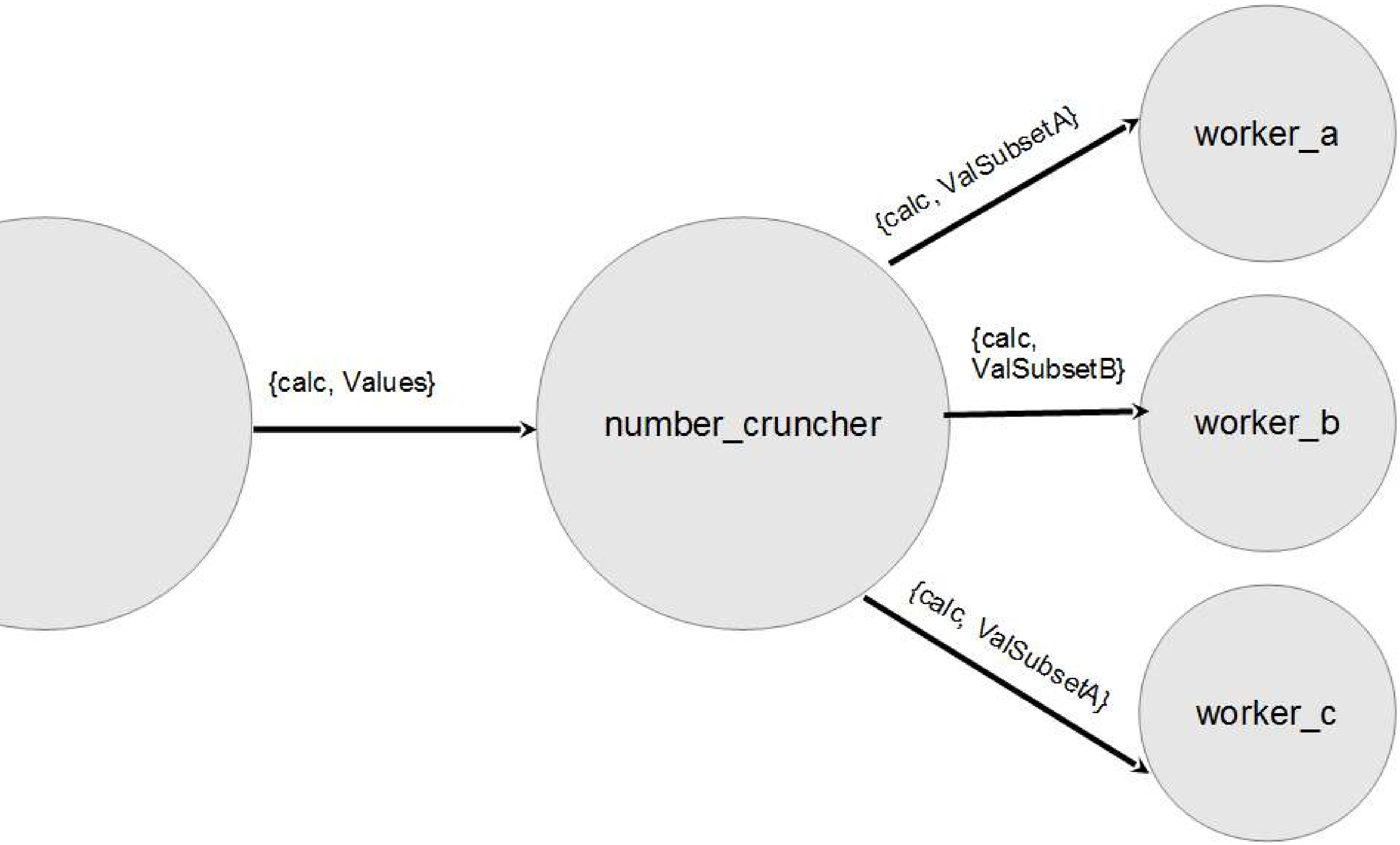
- Proposed in 1973 by Hewitt, Bishop and Steiger
  - "Universal primitives for concurrent computation"
- No shared-state, self-contained and atomic
- Building blocks for modular, distributed and concurrent systems
- Implemented in a variety of programming languages

Actor Model

- Asynchronous message passing
  - Messages kept in a mailbox and processed in the order they are received in
- Upon receiving messages, actors can:
  - Make local decisions and change internal state
  - Spawn new actors
  - Send messages to other actors

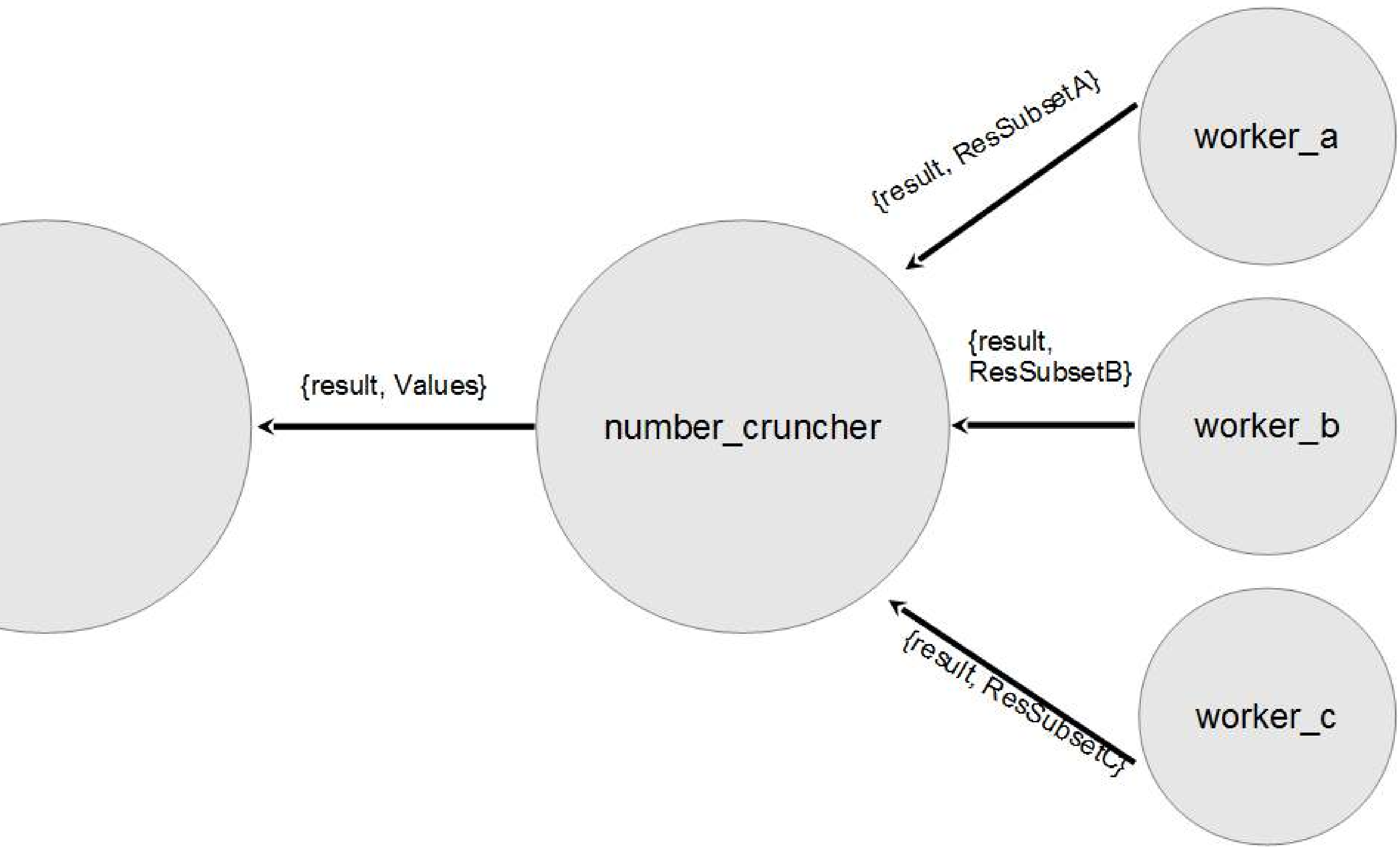


Actor Model



{calc, Values}
number_cruncher
{calc, ValSubsetA}
{calc,
ValSubsetB}
{calc, ValSubsetA}
worker_a
worker_b
worker_c

Actor Model



{result, ResSubsetA}
worker_a
{result, Values}
number_cruncher
{result,
ResSubsetB}
worker_b
{result, ResSubsetC}
worker_c

Limitations of the Actor Model

- No notion of inheritance or general hierarchy
  - Specific to language and library implementation
- Asynchronous message passing can be problematic for certain applications
  - Ordering of messages received from multiple processes
  - Abstract definition may lead to inconsistency in larger systems
    - Fine/Coarse Grain argument

Erlang Embedded

- Knowledge Transfer Partnership between Erlang Solutions and University of Kent
  - Aim of the project: Bring the benefits of concurrent systems development using Erlang to the field of embedded systems; through investigation, analysis, software development and evaluation.

http://erlang-embedded.com

Why Erlang?

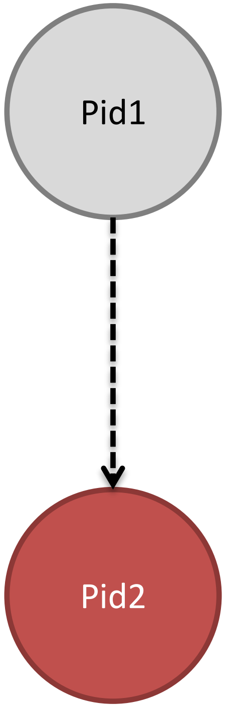
- Implements the Actor model
- Battle-tested at Ericsson and many other companies
  - Originally designed for embedded applications
- Support for concurrency and distributed systems out of the box
- Easy to create robust systems
- (...and more!)

High Availability/Reliability

- Simple and consistent error recovery and supervision hierarchies
- Built in fault-tolerance
  - Isolation of Actors
- Support for dynamic reconfiguration
  - Hot code loading

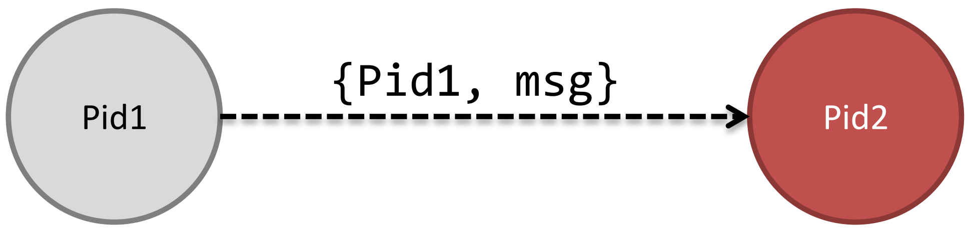
Creating an Actor

```
spawn(math, fact, [5])
```



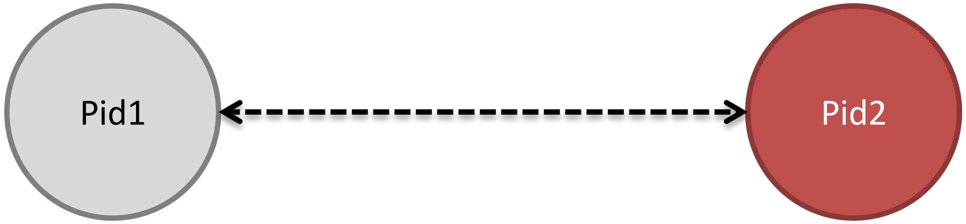

```
math:fact(5)
```

Communication




```
Pid2 ! {self(), msg}
```

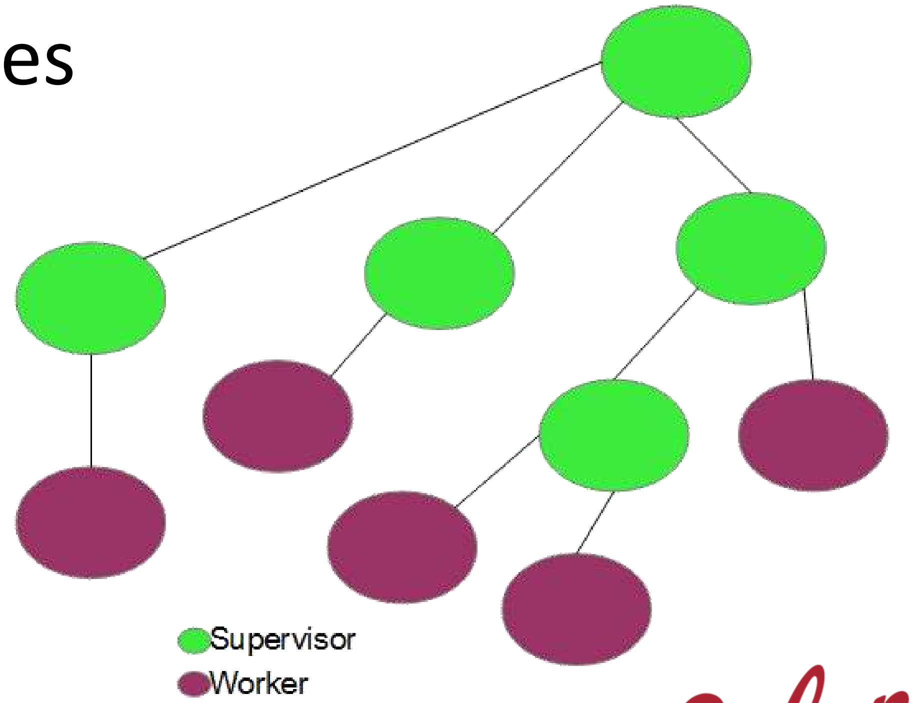
Bidirectional Links



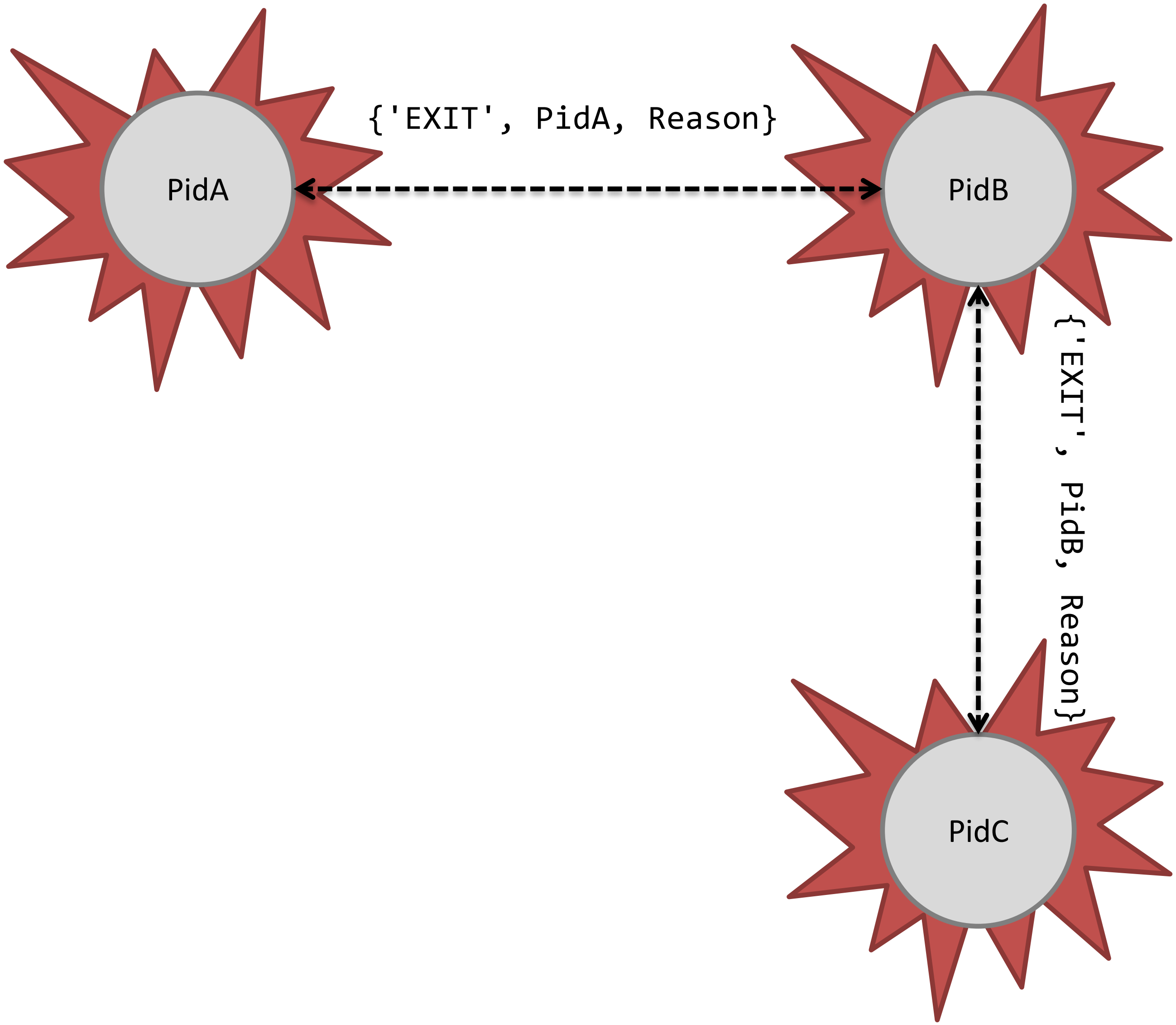

```
link(Pid2)
```

Process Error Handling

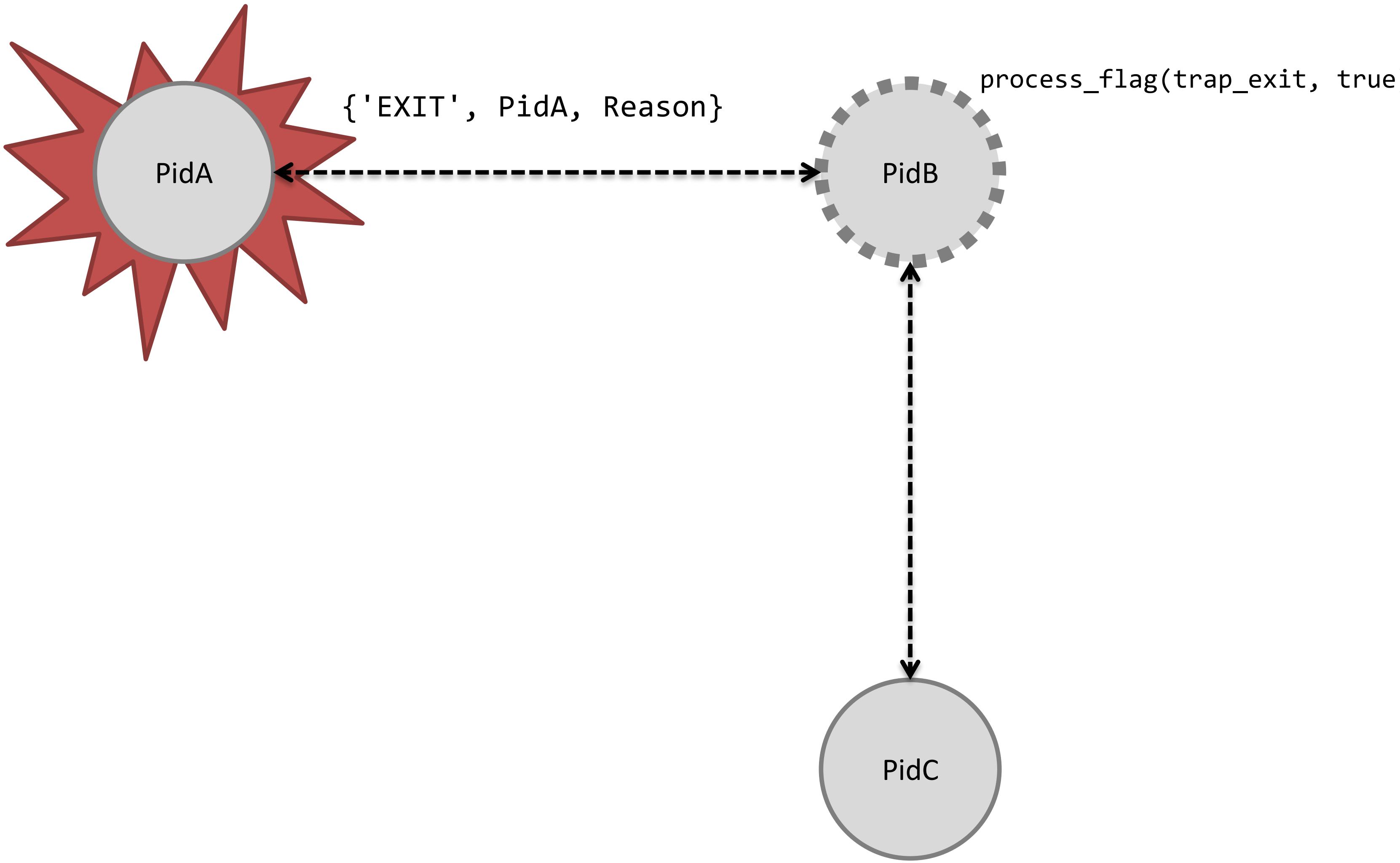
- Let it Fail!
  - Abstract error handling away from the modules
  - Results in leaner modules
- Supervision hierarchies



Propagating Exit Signals

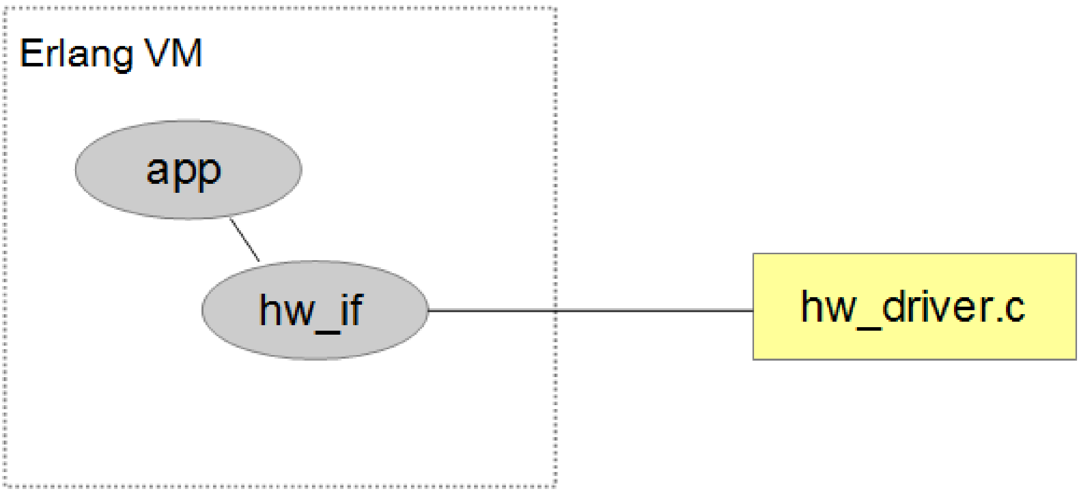


Trapping Exits



External Interfaces

- Native Implemented Functions (NIFs) and ports used to interface external world to the Erlang runtime.

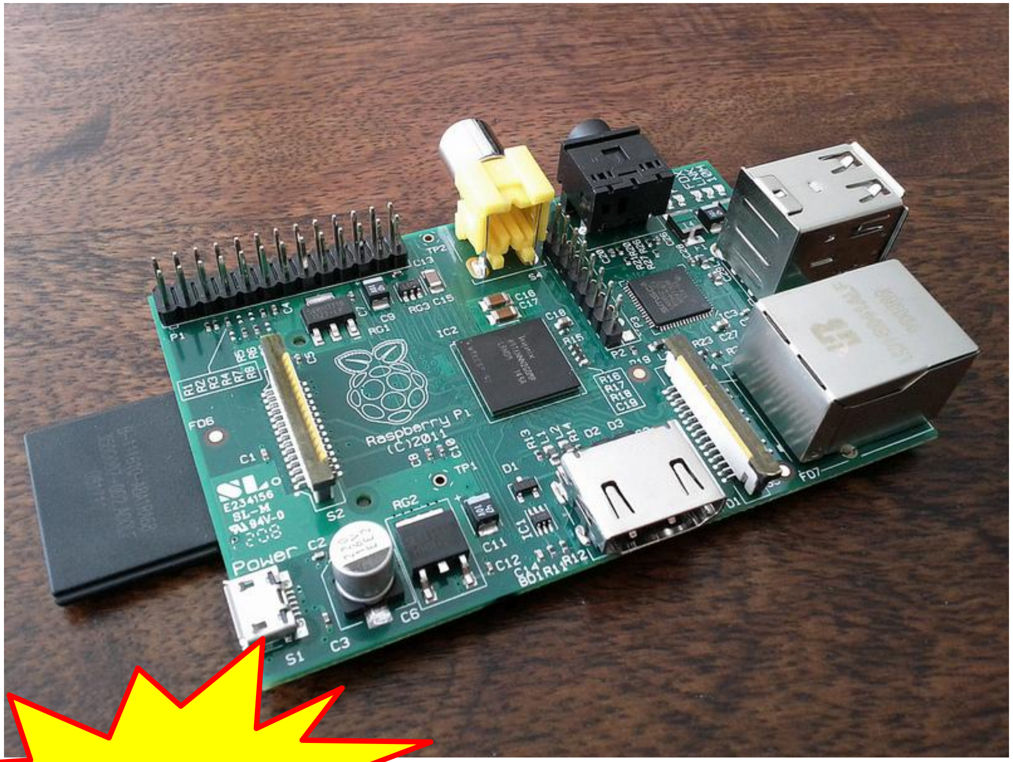


Erlang, the Maestro

(flickr/dereckesanches)

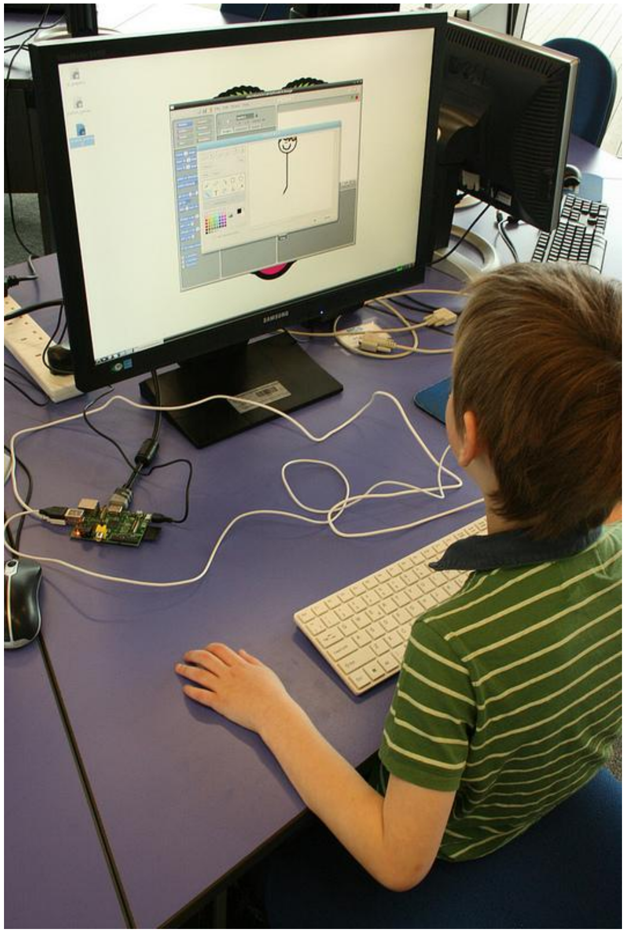
Raspberry Pi

- 700 MHz ARM11
- 256 MB DDR2 RAM
- 10/100Mb Ethernet
- 2x USB 2.0
- (HDMI, Composite Video, 3.5mm Stereo Jack, DSI, CSI-2)



$35

Raspberry Pi in Education



(flickr/lebeus)

- The Raspberry Pi Foundation is a UK registered charity.
- Mission statement: *"...to promote the study of computer science and related topics, especially at school level, and to put the fun back into learning computing.*"

Future Engineers/Programmers!

Raspberry Pi Peripherals

- GPIO
- UART
- I2C
- I2S
- SPI
- PWM
- DSI
- CSI-2

Accessing peripherals

- Peripherals are memory mapped
- Access via /dev/mem
  - Faster, needs root, potentially dangerous!
- Use kernel modules/sysfs
  - Slower, doesn’t need root, easier, relatively safer

GPIO Interface (I)

```
init(Pin, Direction) ->

  {ok, FdExport} = file:open("/sys/class/gpio/export", [write]),
  file:write(FdExport, integer_to_list(Pin)),
  file:close(FdExport),


  {ok, FdPinDir} = file:open("/sys/class/gpio/gpio" ++ integer_to_list(Pin)
++ "/direction", [write]),
  case Direction of
    in -> file:write(FdPinDir, "in");
    out -> file:write(FdPinDir, "out")
  end,
  file:close(FdPinDir),


  {ok, FdPinVal} = file:open("/sys/class/gpio/gpio" ++ integer_to_list(Pin)
++ "/value", [read, write]),

  FdPinVal.
```

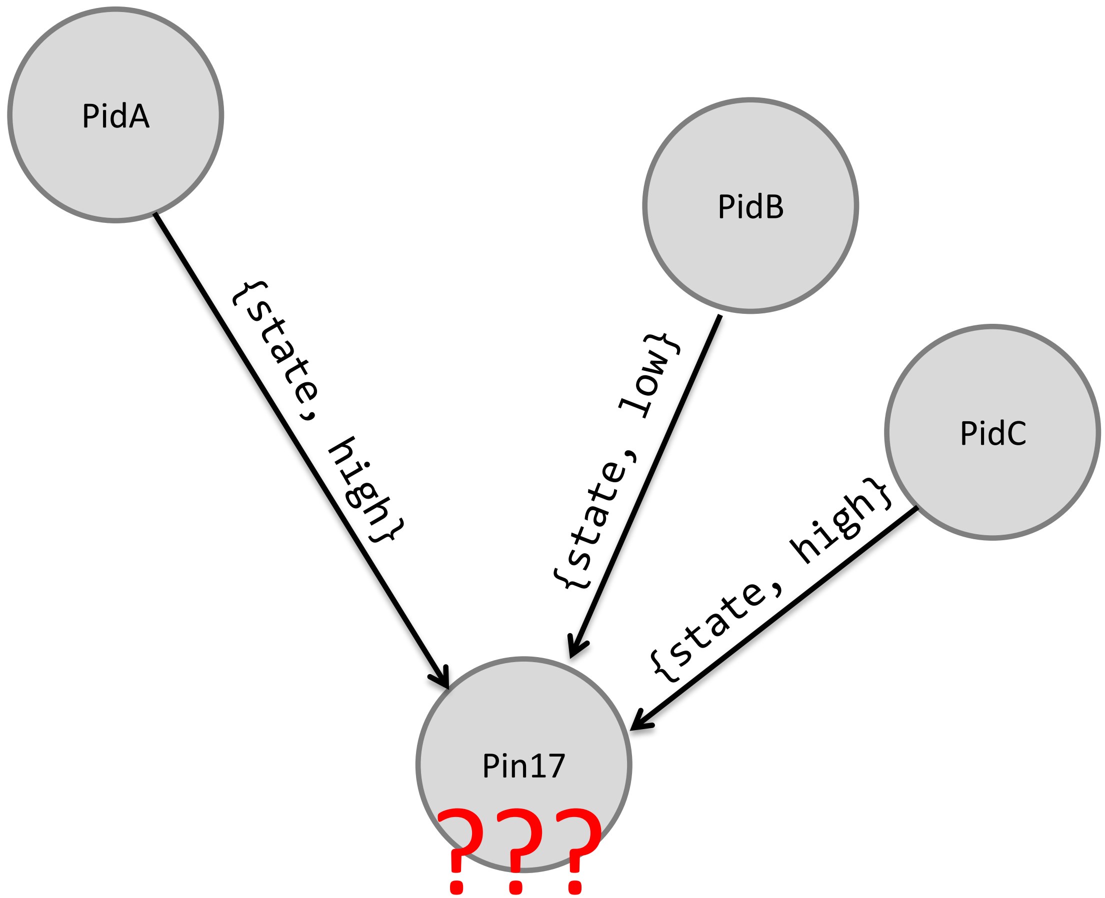
GPIO Interface (II)

```
write(Fd, Val) ->
  file:position(Fd, 0),
  file:write(Fd, integer_to_list(Val)).


read(Fd) ->
  file:position(Fd, 0),
  {ok, Val} = file:read(Fd, 1),
  Val.


release(Pin) ->
  {ok, FdUnexport} = file:open("/sys/class/gpio/unexport",
[write]),
  file:write(FdUnexport, integer_to_list(Pin)),
  file:close(FdUnexport).
```

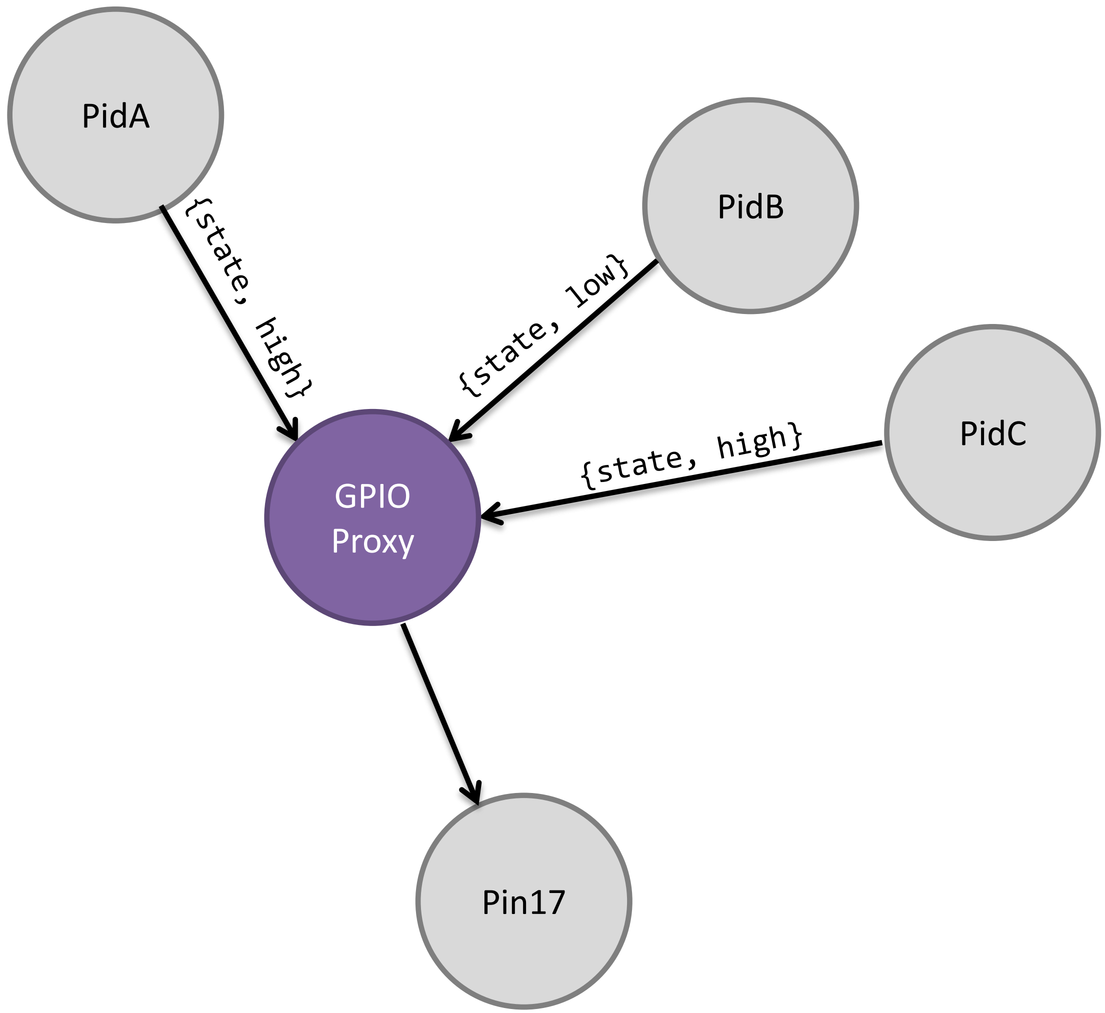
Concurrency Demo

http://vimeo.com/40769788

Example: GPIO



Example: GPIO



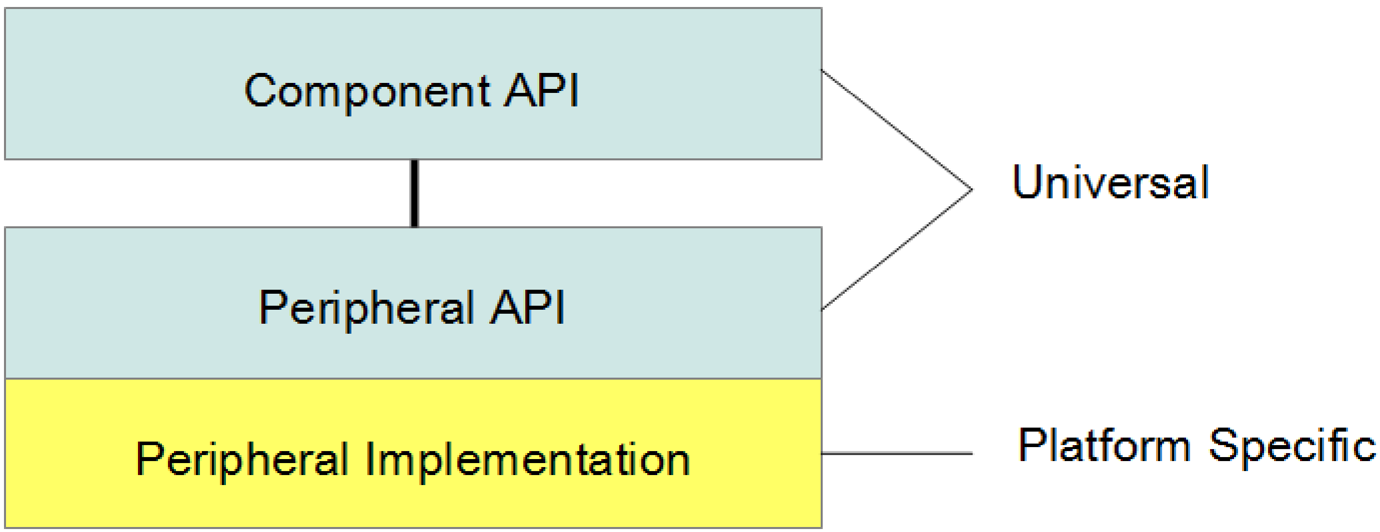
GPIO Proxy

- Replaces ‘locks’ in traditional sense of embedded design
  - Access control/mutual exclusion
- Can be used to implement safety constraints
  - Toggling rate, sequence detection, direction control, etc.

Fine Grain Abstraction

- Advantages
  - Application code becomes simpler
  - Concise and shorter modules
  - Testing becomes easier
  - Code re-use (potentially) increases
- Disadvantage
  - Architecting fine grain systems is difficult

Universal Peripheral/Component Modules



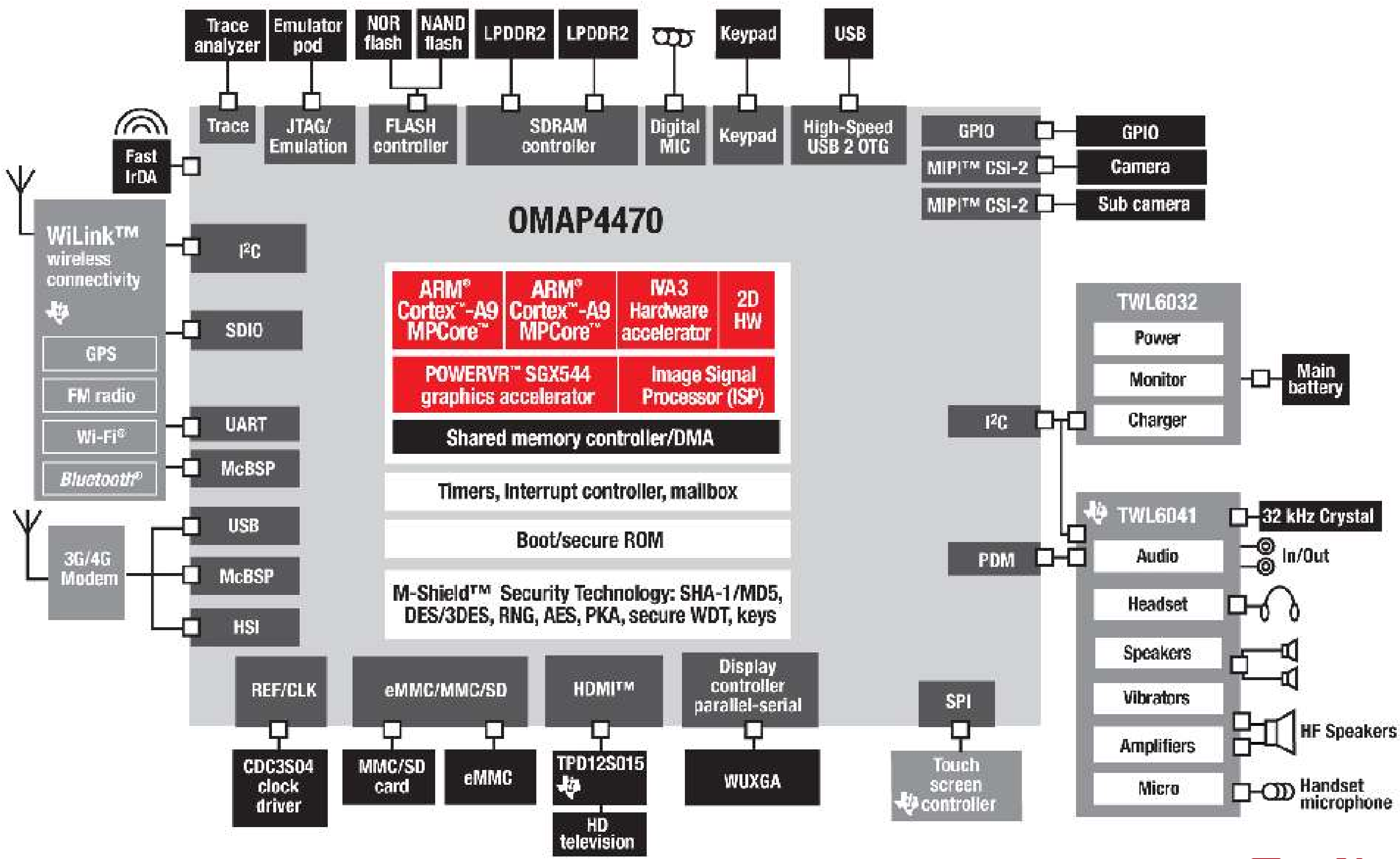
Universal Peripheral/Component Modules

An Example:

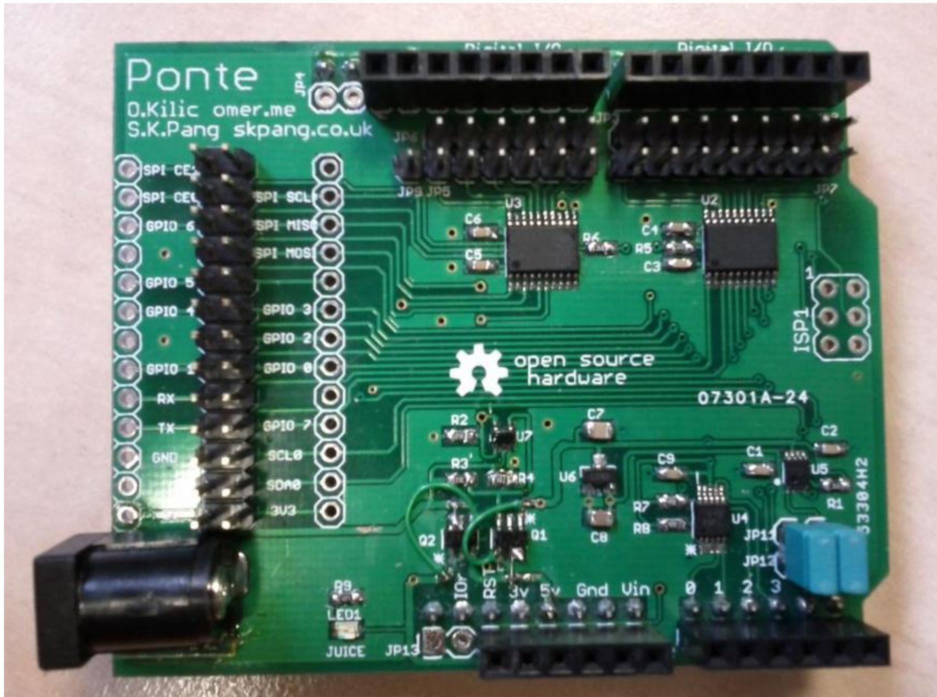
Temperature Sensor with I2C Interface

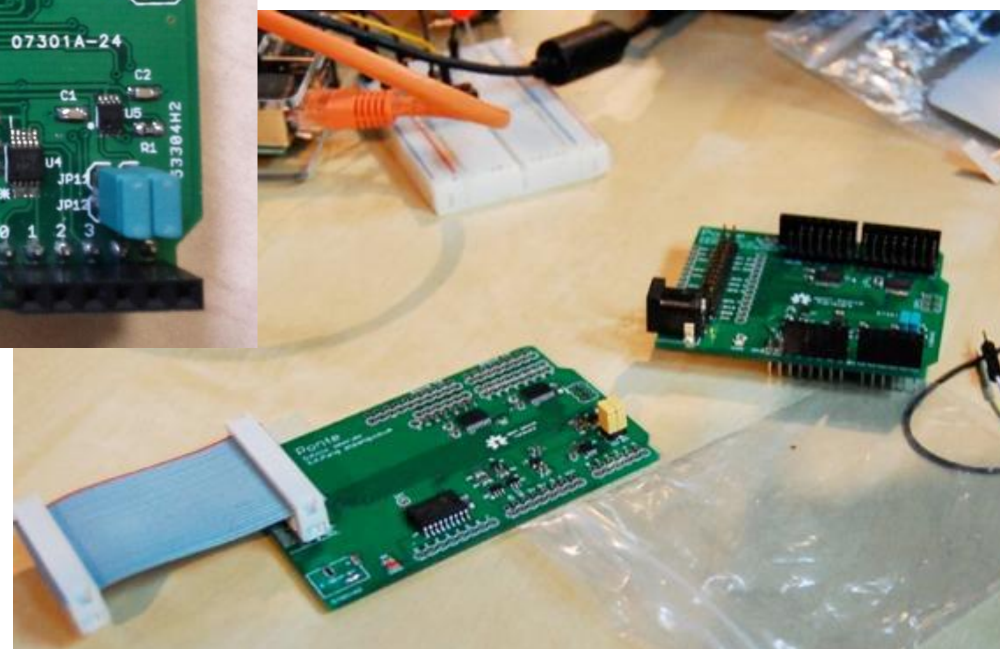
Sensor API	Sensor API	Sensor API
I2C Bus Driver	I2C Bus Driver	I2C Bus Driver
Linux sysfs	BSD sysctl	mmap()'ed register voodoo

TI OMAP Reference System

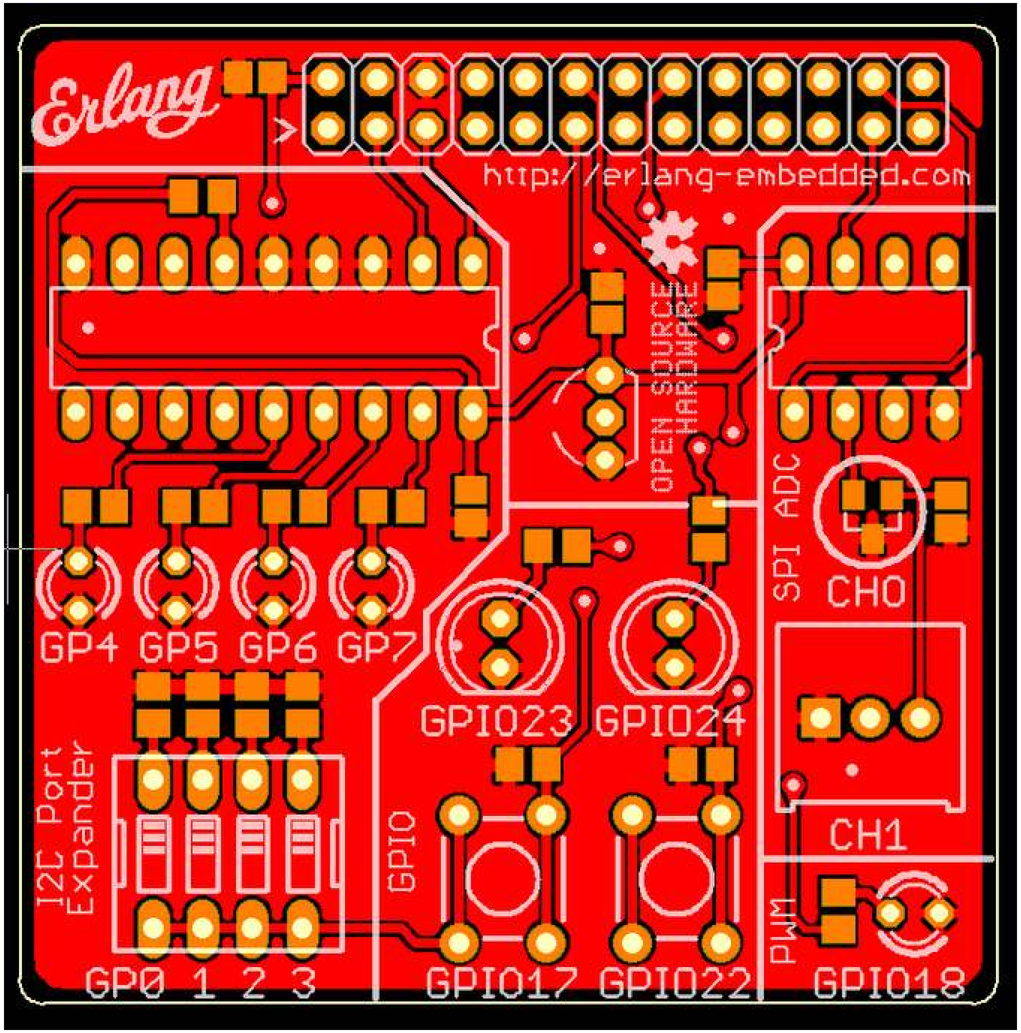


Hardware Projects – Ponte

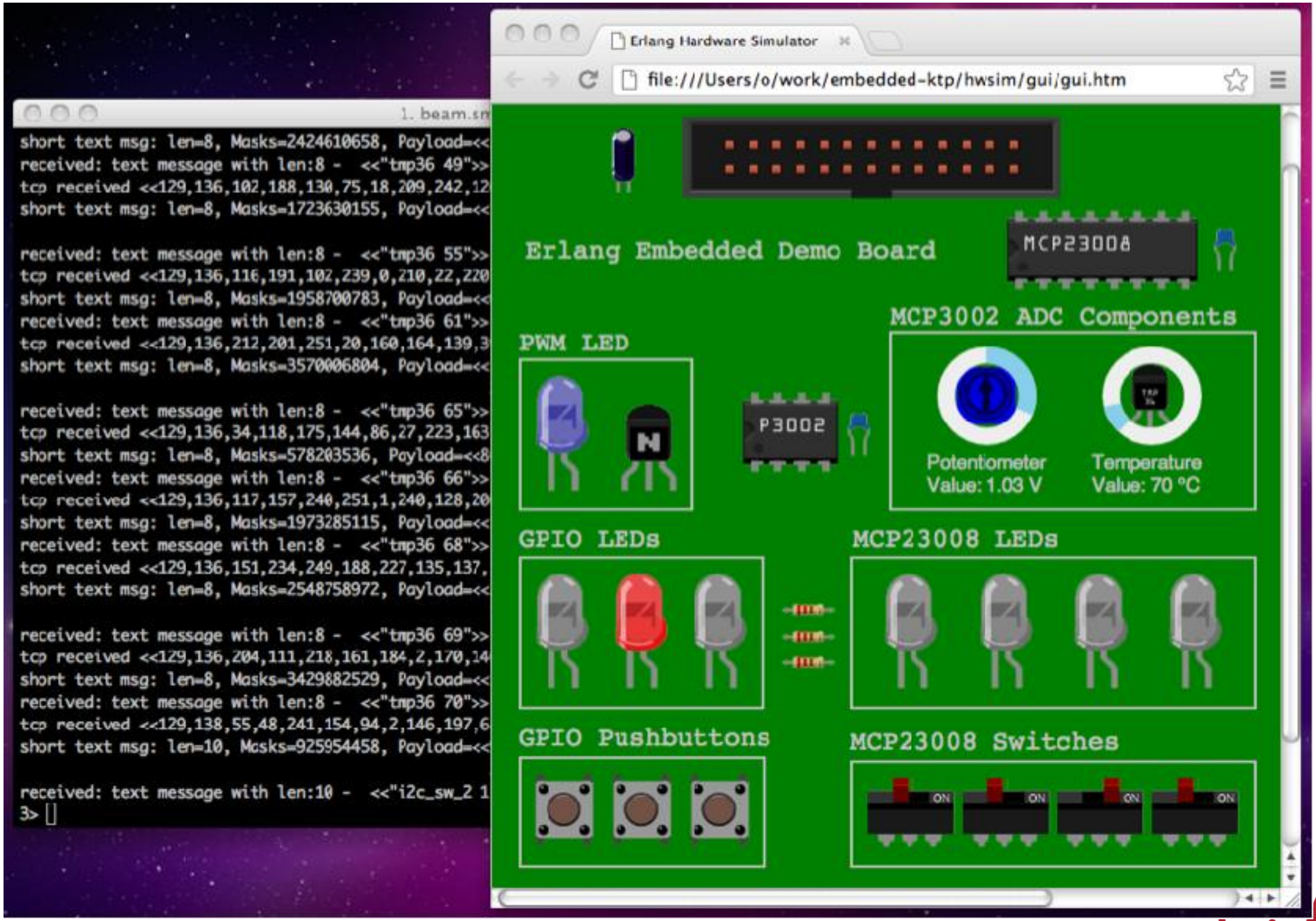




Hardware Projects – Demo Board

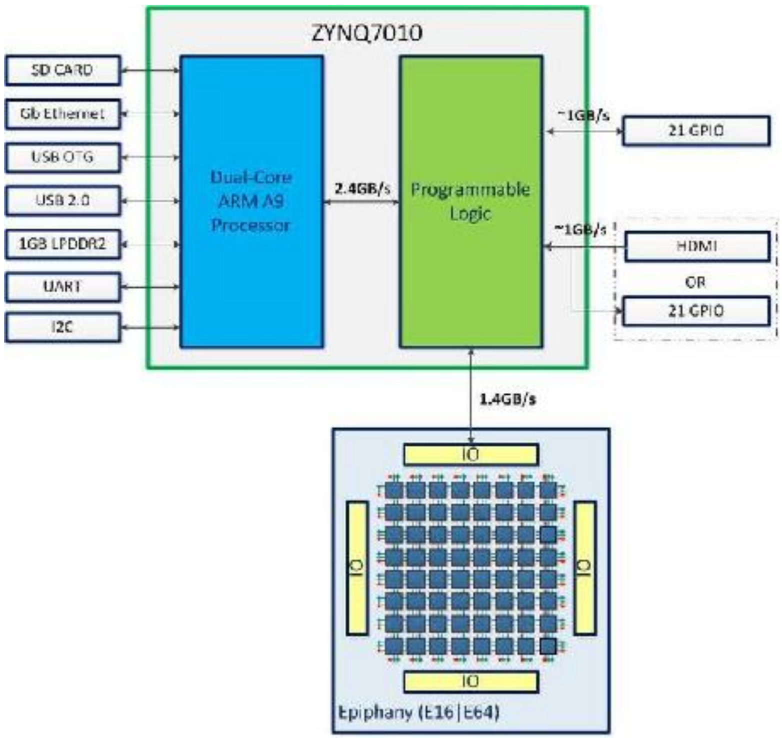


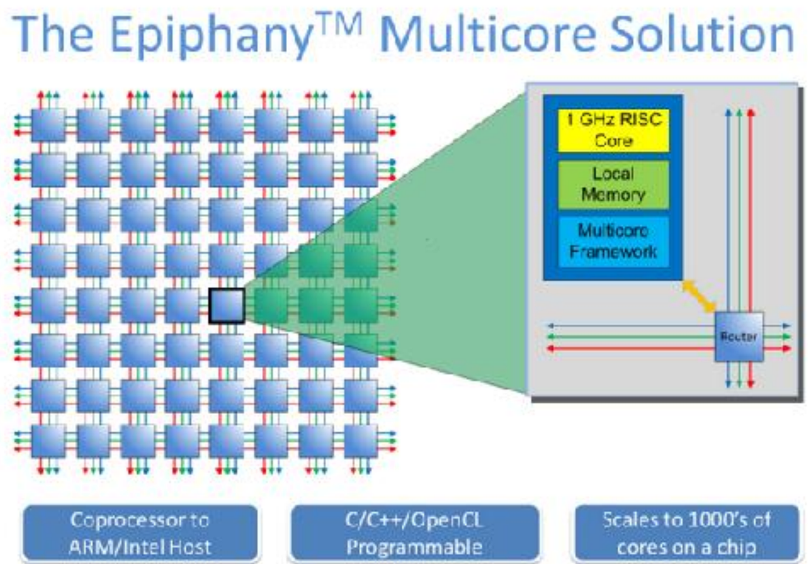
Hardware Simulator



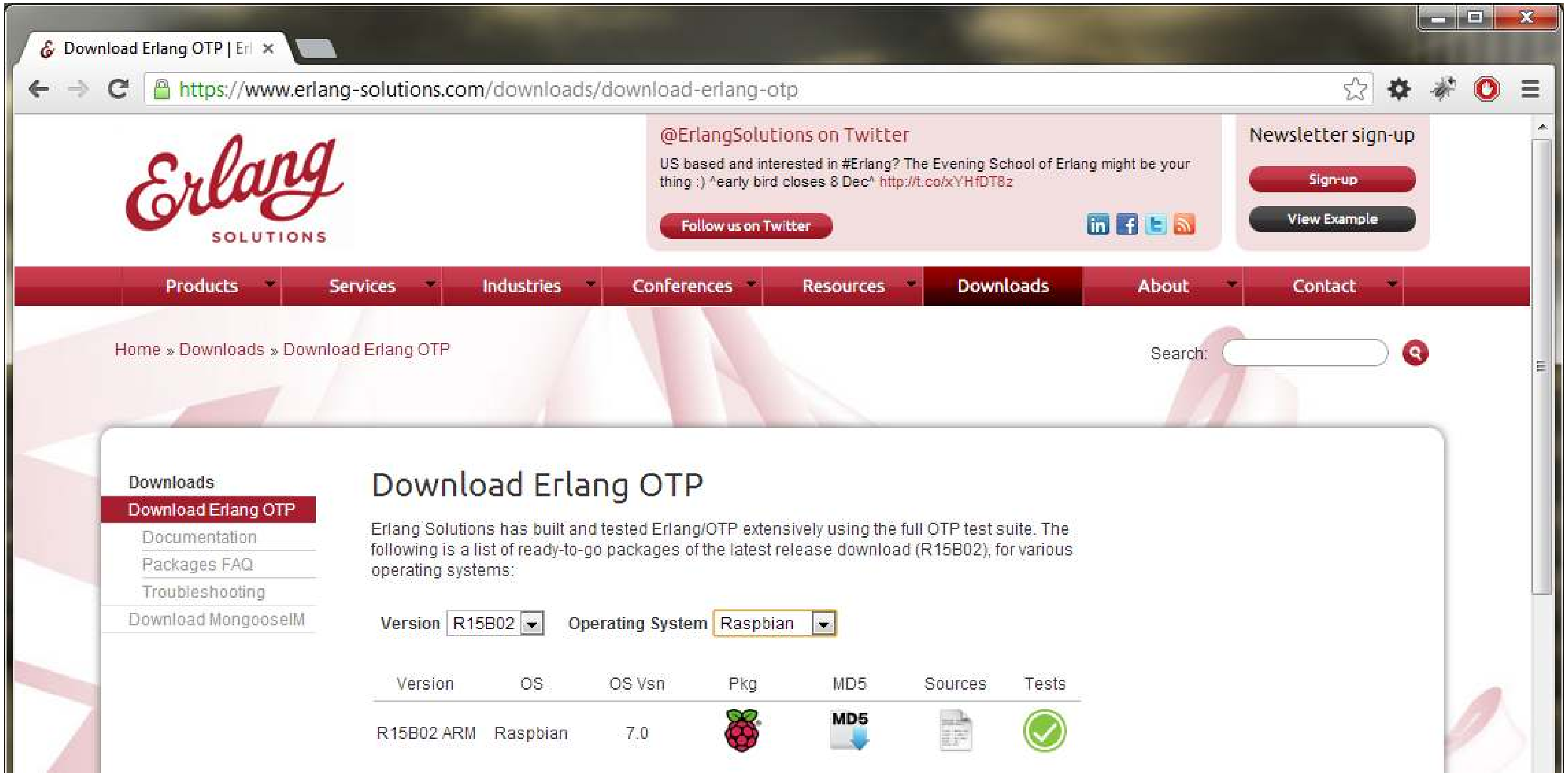
Future Explorations

Parallella:





Packages for Embedded Architectures



https://www.erlang-solutions.com/downloads/download-erlang-otp

Erlang Embedded Training Stack

- A complete package for people interested in developing the next generation of concurrent and distributed Embedded Systems
- Training Modules:
  - Embedded Linux Primer
  - Erlang/OTP 101
  - Erlang Embedded Framework

Get in touch if you're interested.

Thank you

- http://erlang-embedded.com
- embedded@erlang-solutions.com
- @ErlangEmbedded

> “ The world is concurrent.
> Things in the world don't share data.
> Things communicate with messages.
> Things fail.
>
> \- Joe Armstrong
>
> Co-inventor of Erlang