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

The Erlang Embedded Project

Omer Kilic || @OmerK

omer@erlang-solutions.com

Outline

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

Embedded Systems (I)

> *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

Embedded Systems (II)

??

- Specific and limited set of functions
- Designed for a particular application

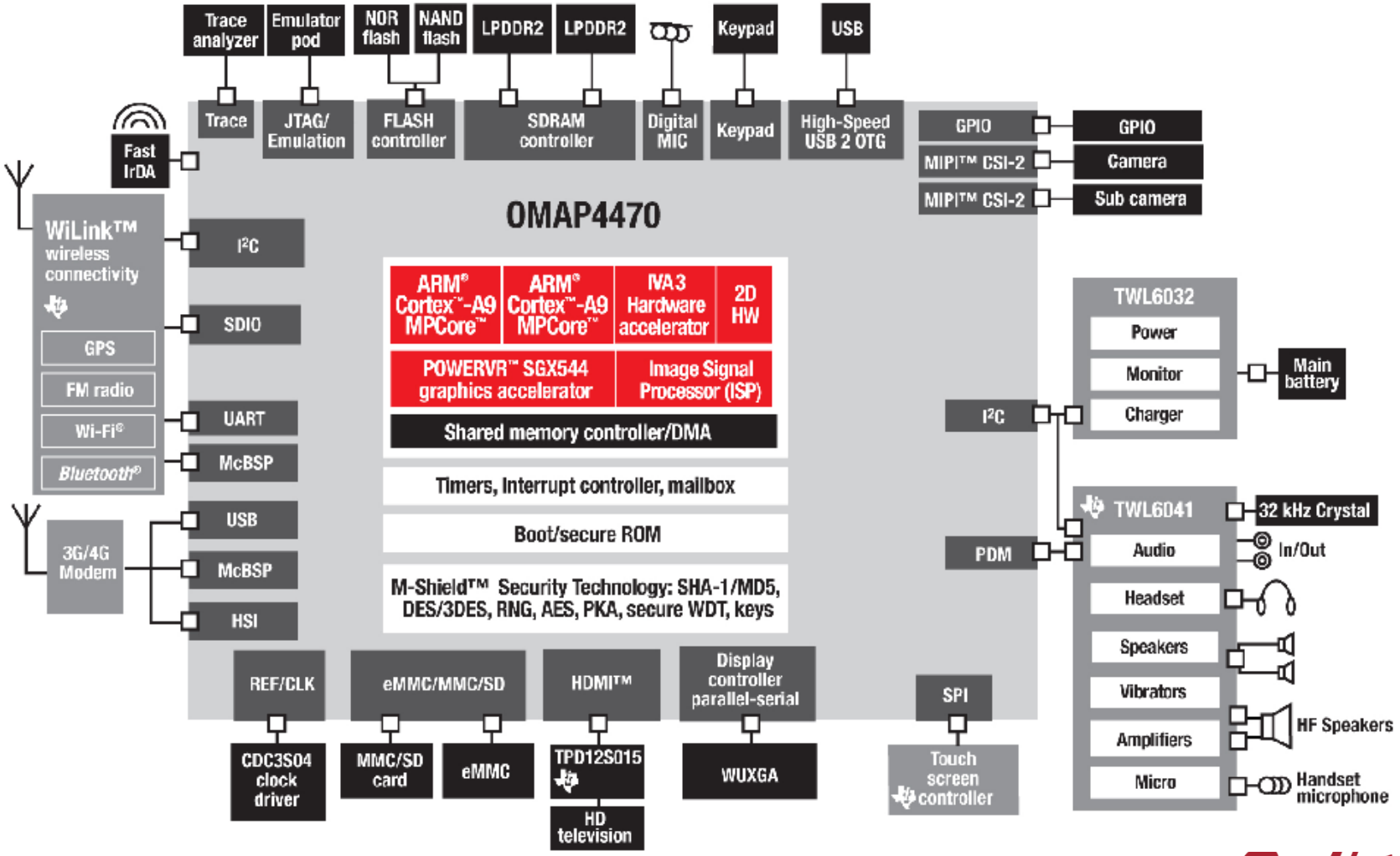
- General purpose
- Can be used for pretty much any computing needs

Erlang SOLUTIONS

Current Challenges

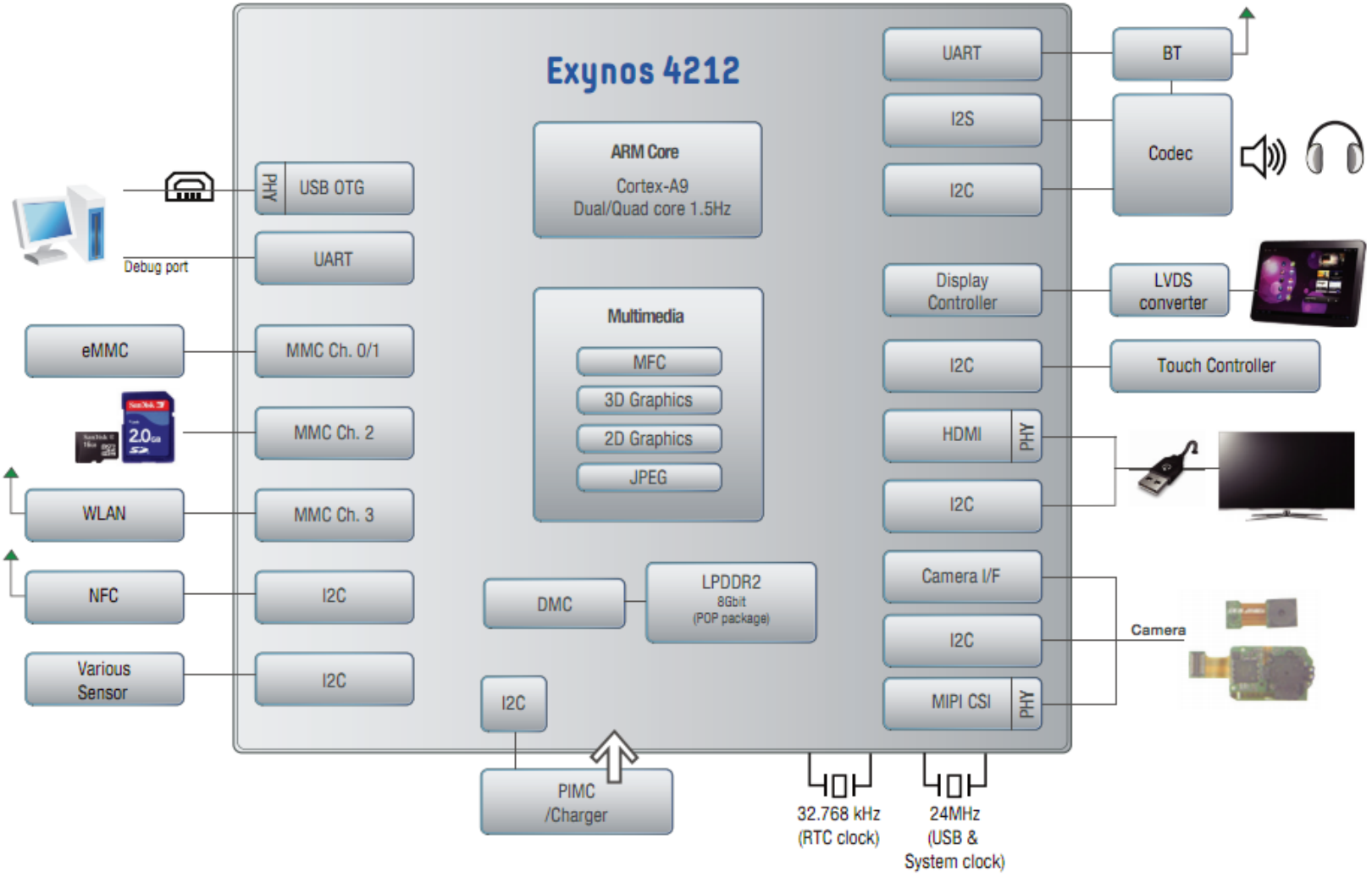
- Complex SoC platforms
- “Internet of Things”
  - Connected and distributed systems
- Multicore and/or heterogeneous devices
- Time to market constraints
  - The Kickstarter Era
  - Rapid prototyping
  - Maker Culture

TI OMAP Reference System

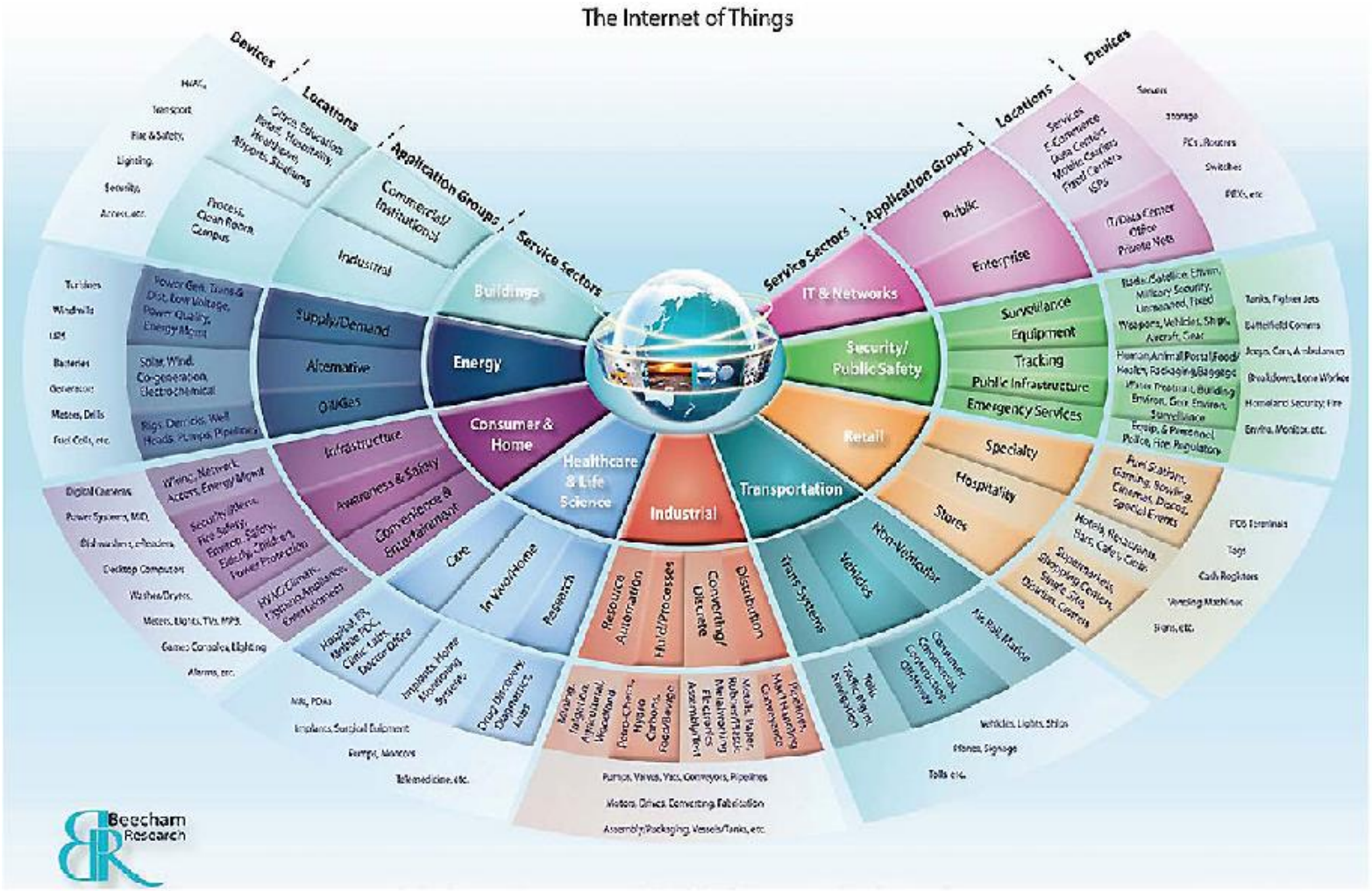


Erlang SOLUTIONS

Samsung Exynos Reference System



Internet of Things

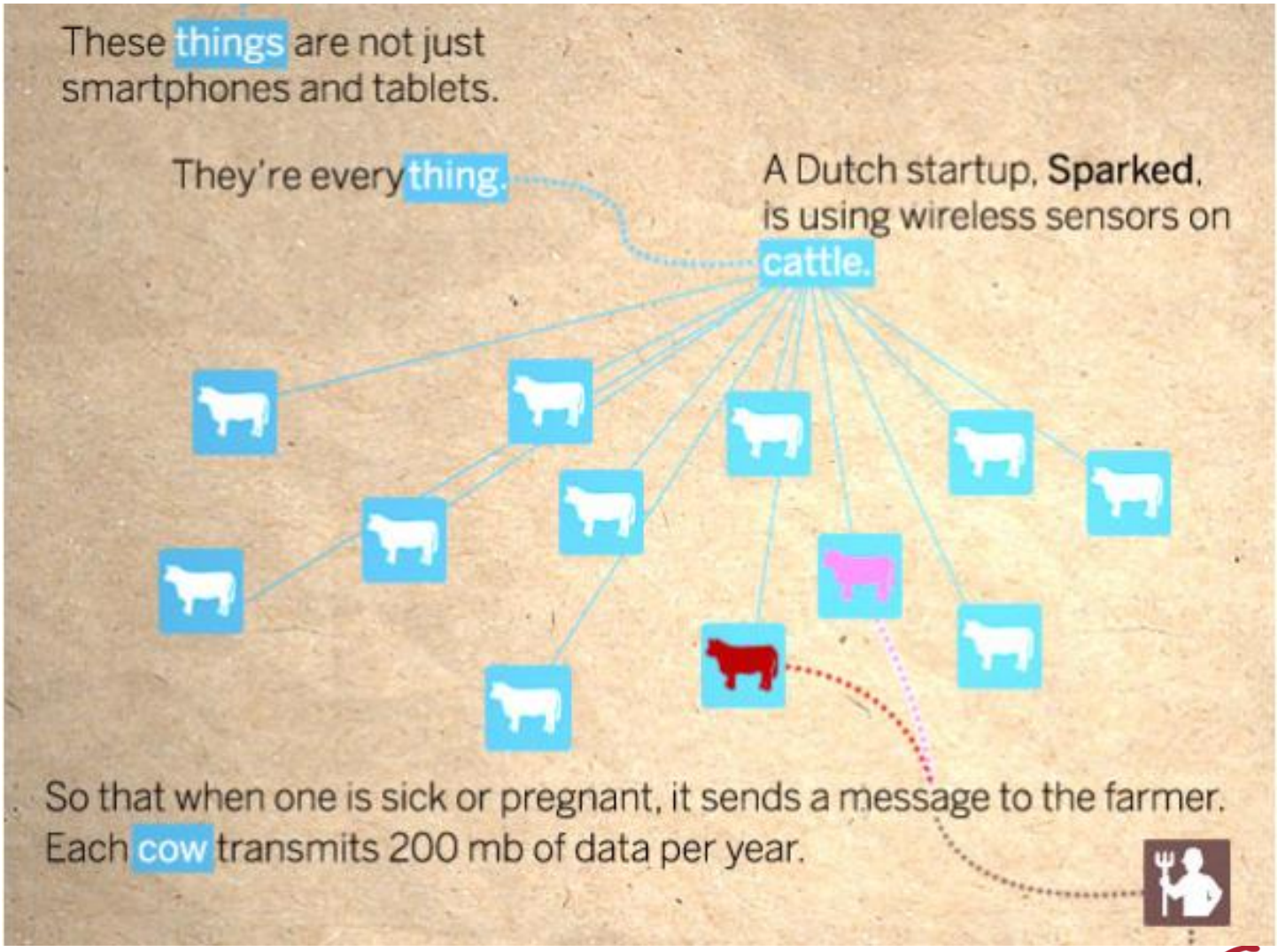


Erlang SOLUTIONS

Internet of Fridges?

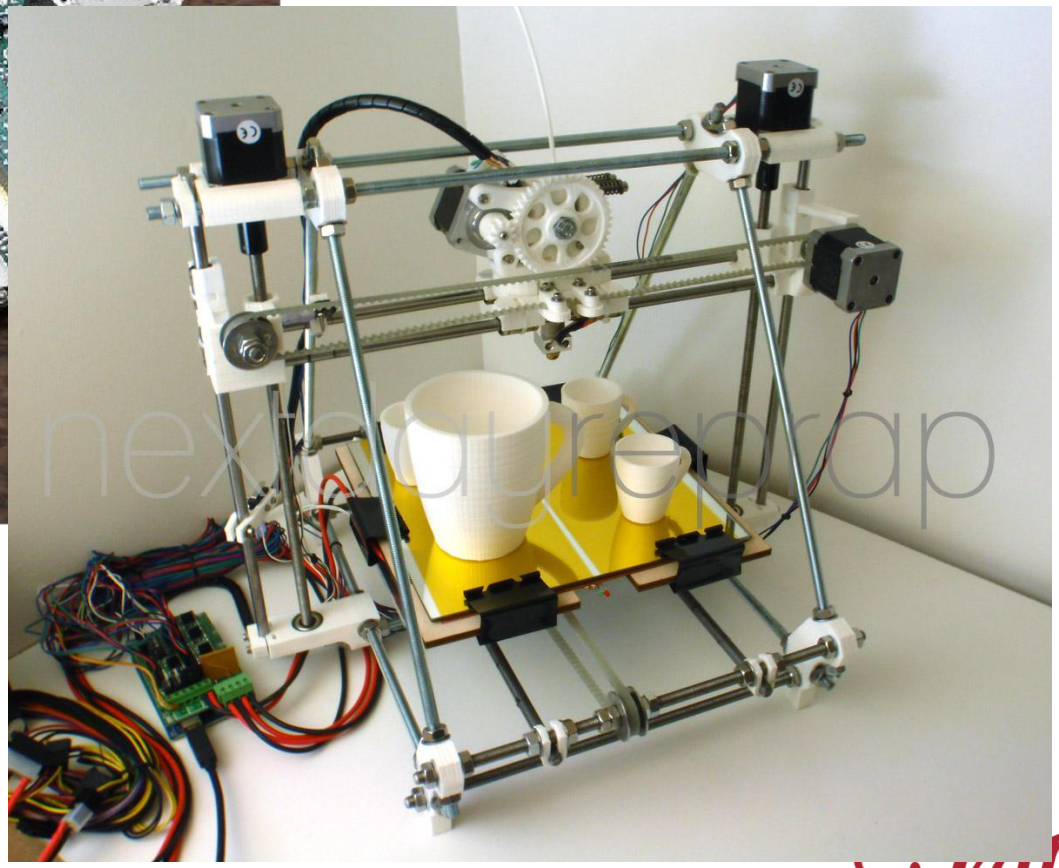
Erlang SOLUTIONS

Distributed Bovine Networks?

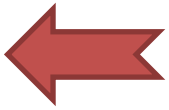
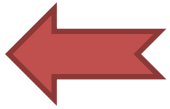


Erlang SOLUTIONS

Exciting times



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

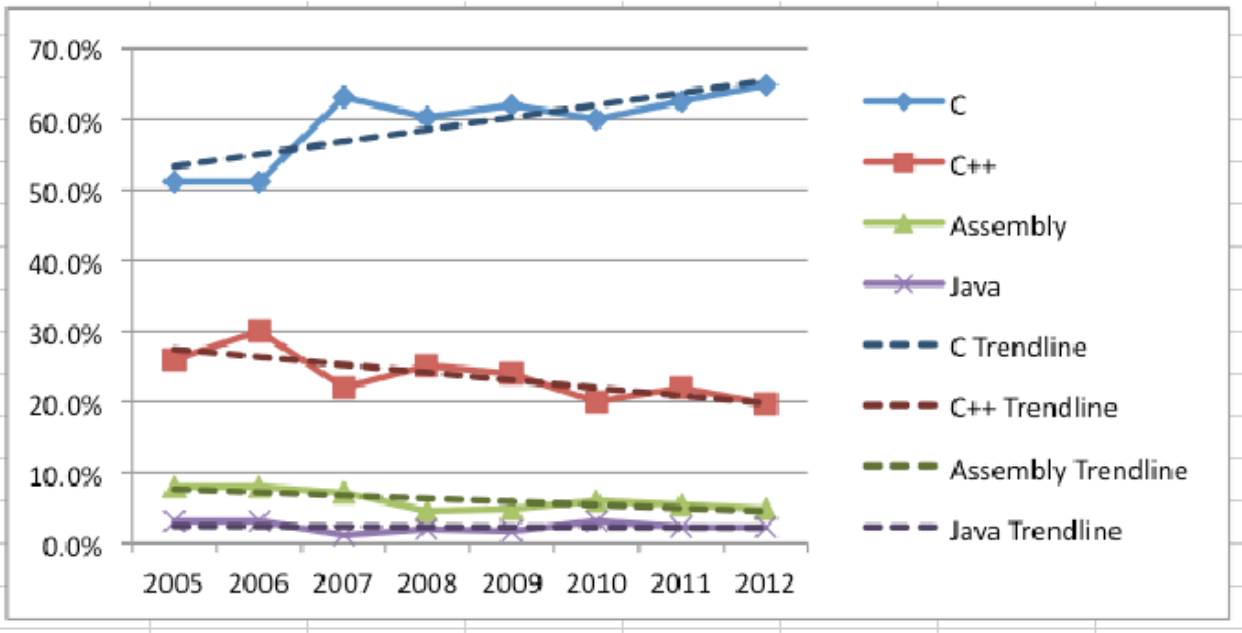
- Notion of “tasks”
- OS-supervised interprocess messaging
  - Shared memory
- Mutexes/Semaphores/Locks
- Scheduling
  - Pre-emptive: event driven
  - Round-robin: time multiplexed

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

#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

Erlang SOLUTIONS

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.

Erlang SOLUTIONS

Erlang? (I)

{functional, declarative,
concurrent, parallel,
garbage-collected,
soft real-time,
fault-tolerant, robust,
portable, distributed
hot code loading}

Erlang SOLUTIONS

Erlang? (II)

- First version developed in 1986
  - Open-sourced in 1998.
- Battle-tested at Ericsson and many other companies
  - Originally designed for Embedded Systems!
- Implements the Actor model
  - Support for concurrency and distributed systems out of the box
- Easy to create robust systems

Erlang SOLUTIONS

High Availability/Reliability

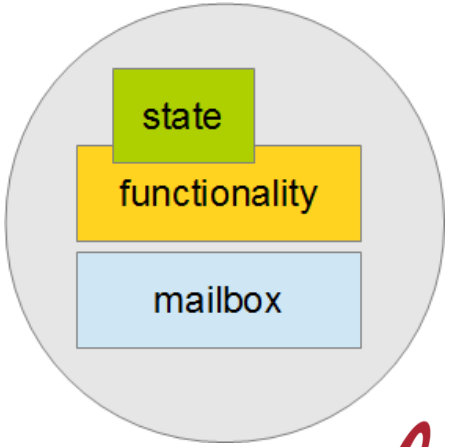
- Designed for systems with high availability constraints
  - Nine 9s availability
- Simple and consistent error recovery and supervision hierarchies
- Built in fault-tolerance
  - Isolation provided by Actors
- Support for dynamic reconfiguration
  - Hot code loading

Actor Model

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

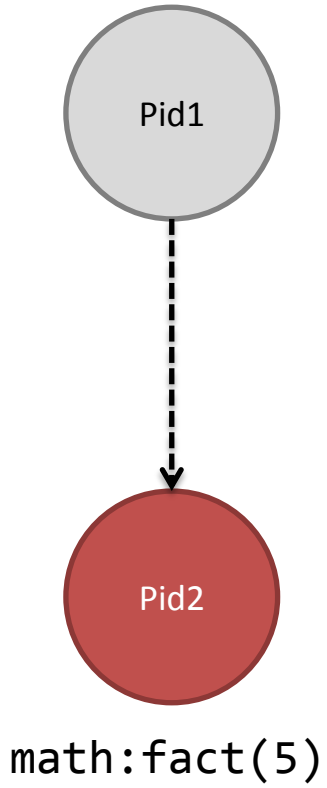


Limitations of the Actor Model

- 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

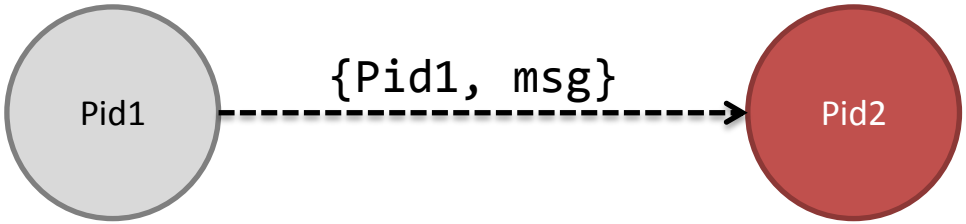
Creating an Actor

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



Erlang SOLUTIONS

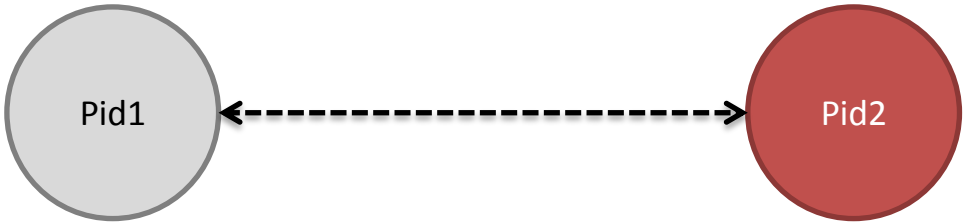
Communication




```
Pid2 ! {msg, data}
```

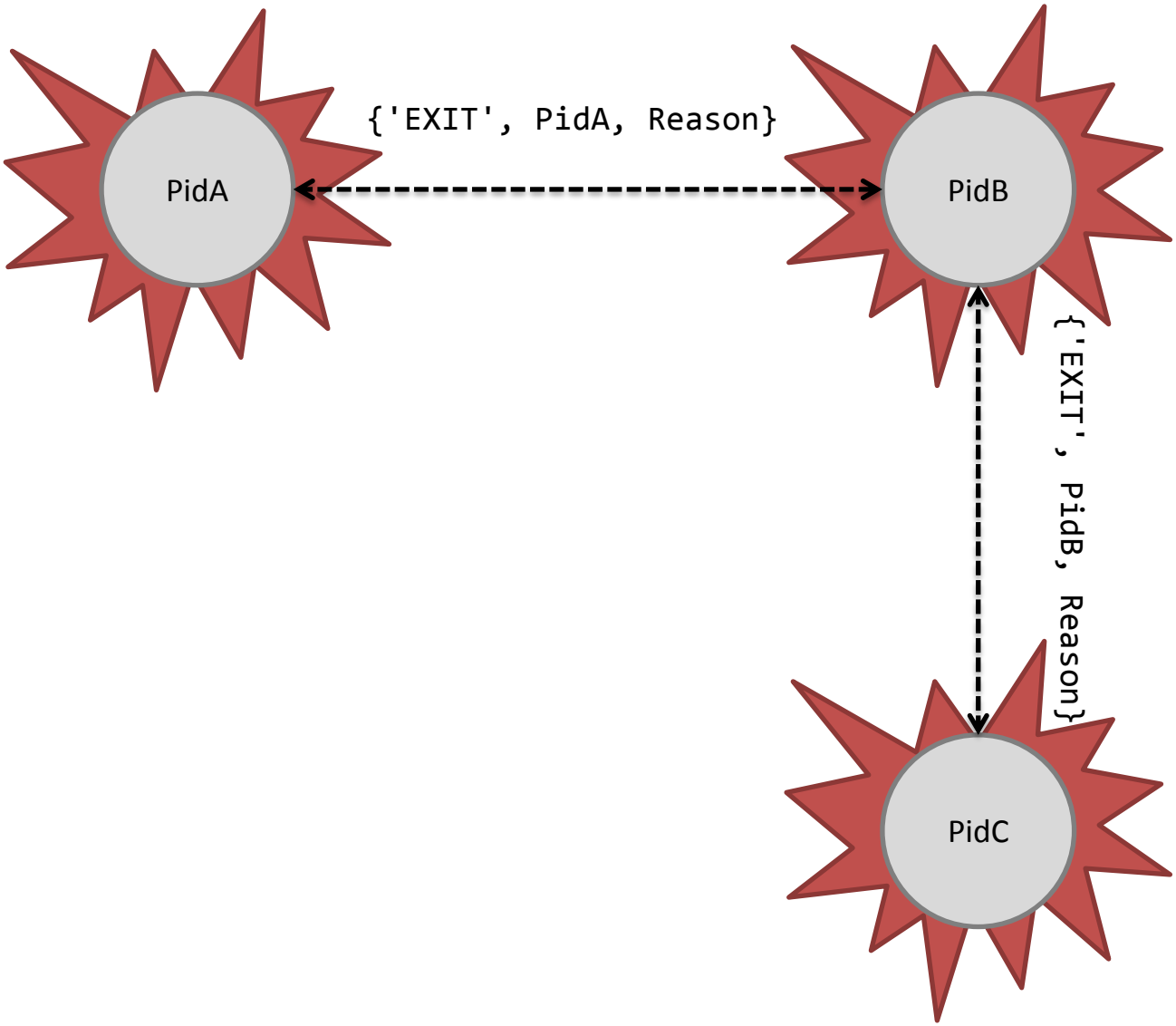

Erlang SOLUTIONS

Bidirectional Links



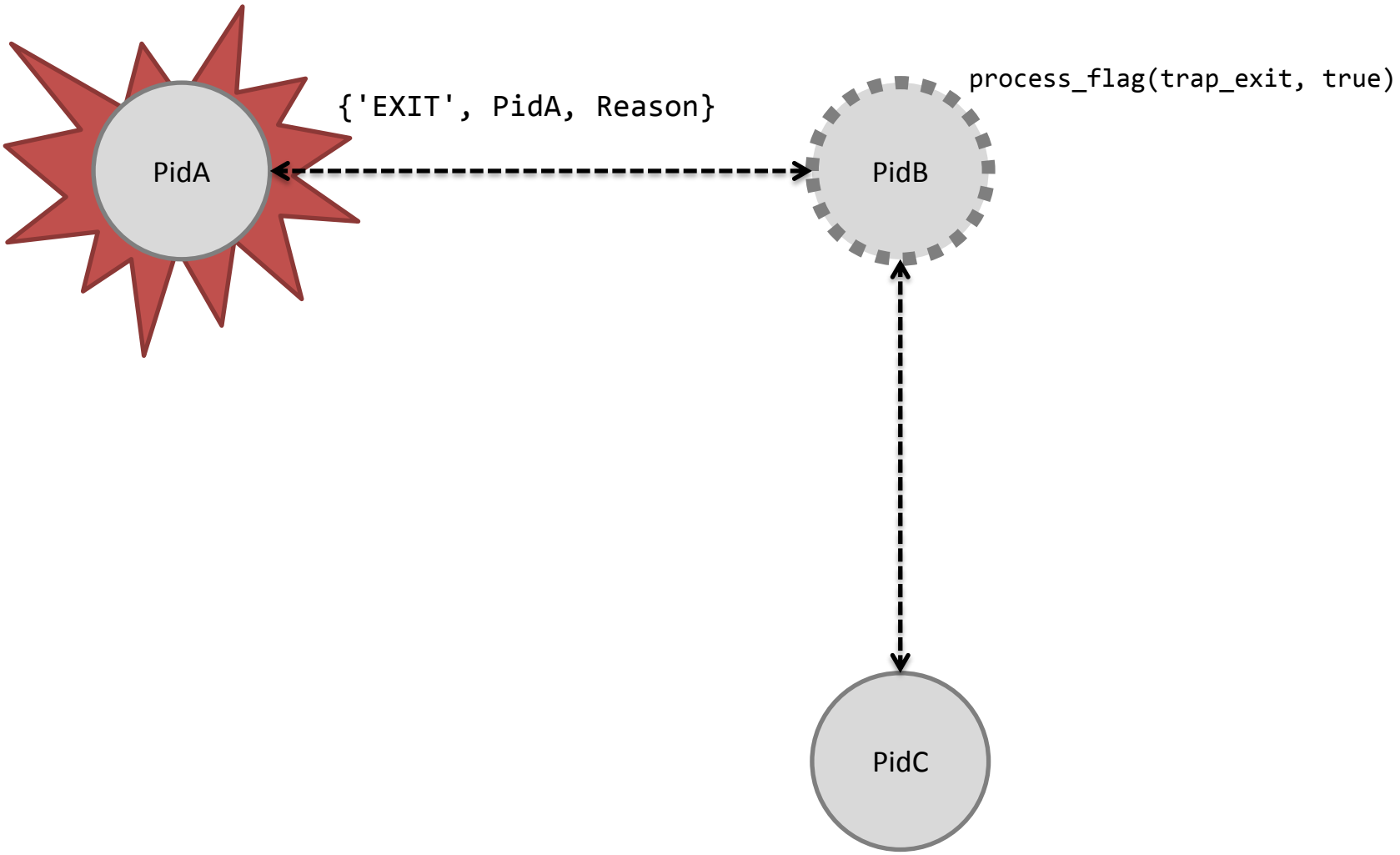

```
link(Pid2)
```

Propagating Exit Signals



Erlang SOLUTIONS

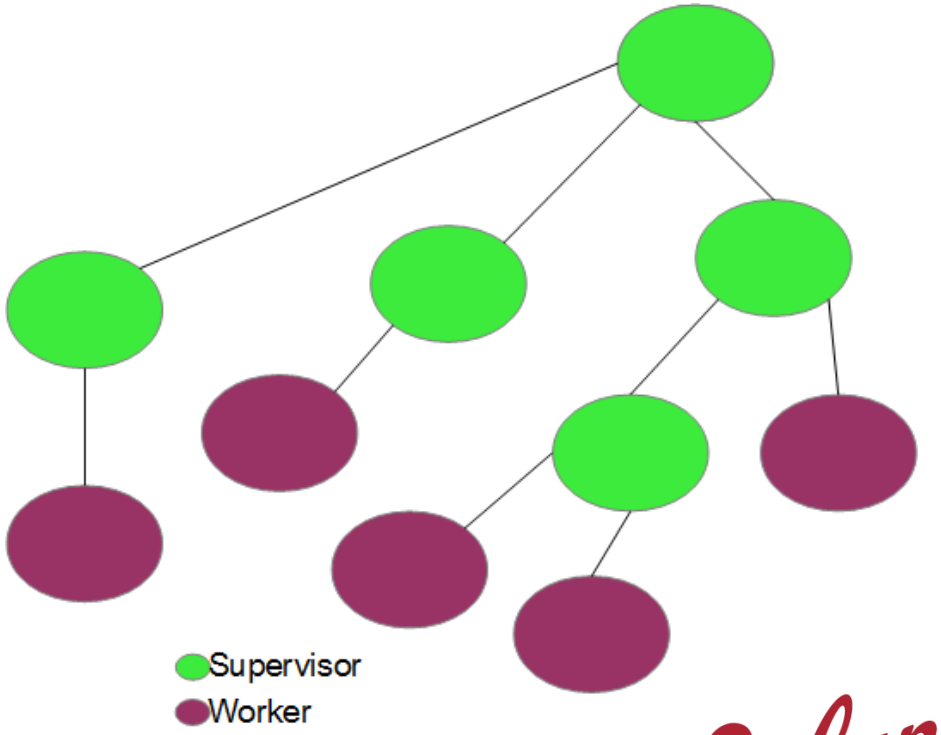
Trapping Exits



Erlang SOLUTIONS

Supervision Hierarchies

- Let it Fail!
  - Abstract error handling away from the modules

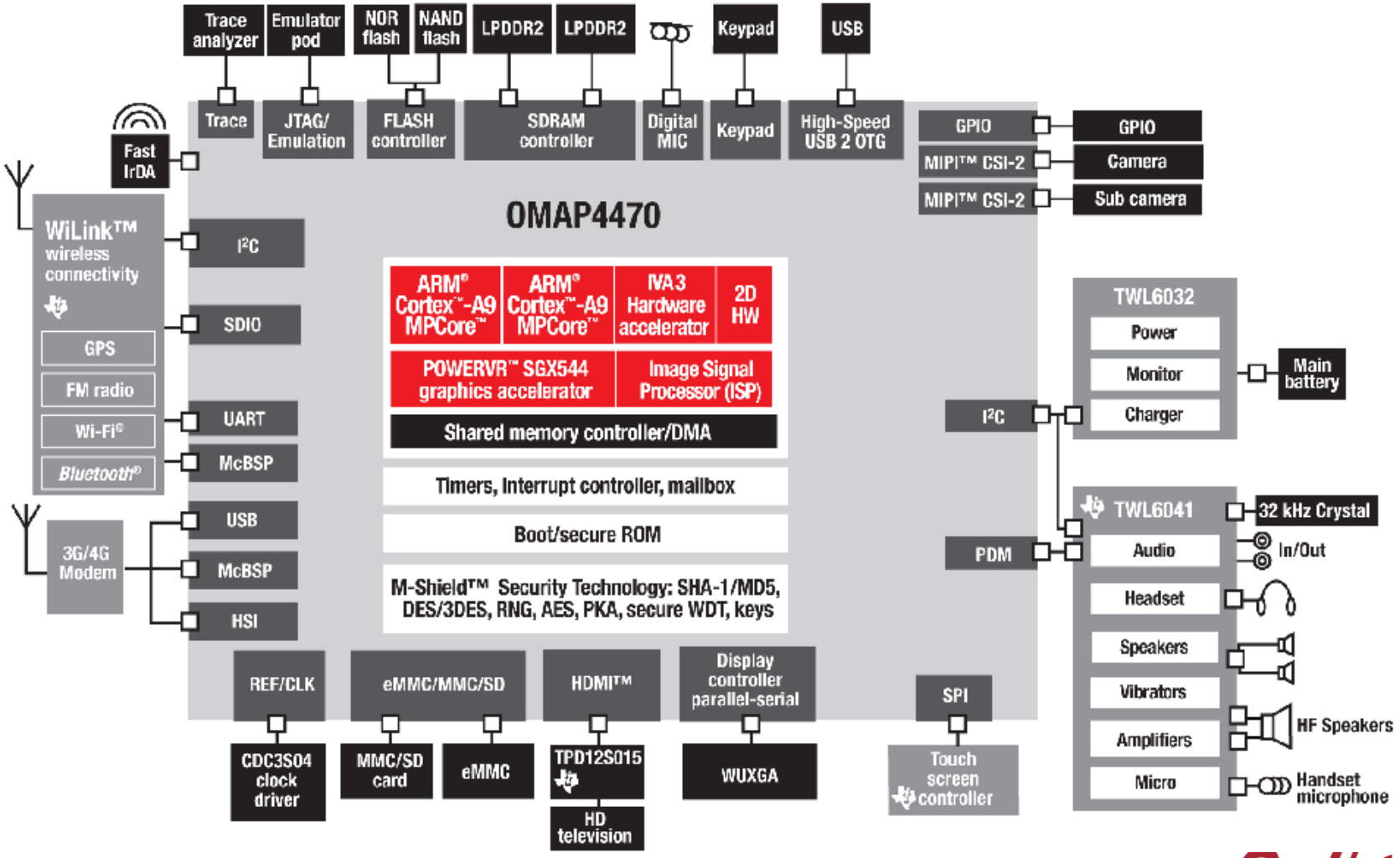


Erlang SOLUTIONS

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

TI OMAP Reference System



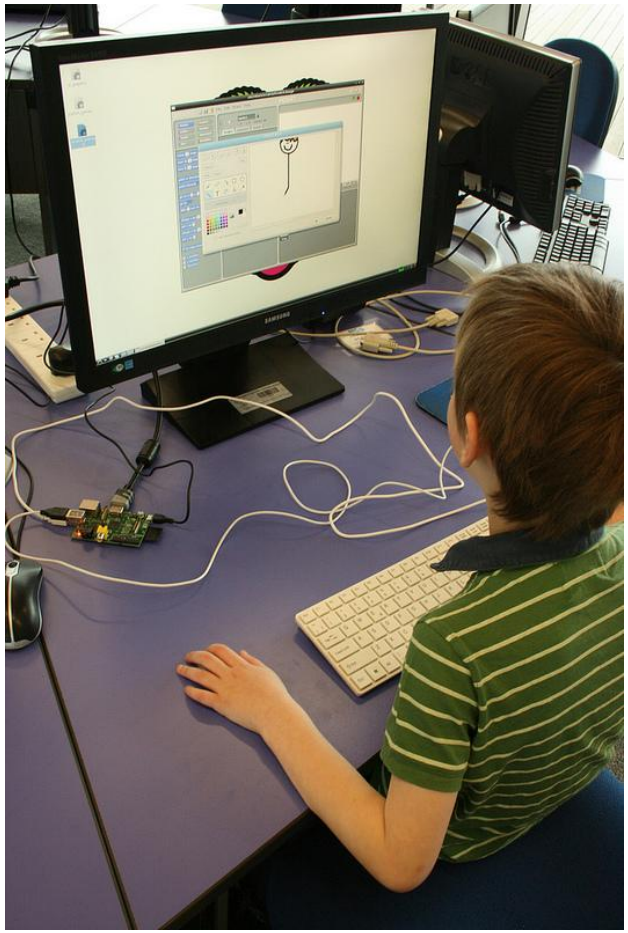
Erlang SOLUTIONS

Erlang, the Maestro

(flickr/dereckesanches)

Erlang SOLUTIONS

Why Raspberry Pi?



(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!

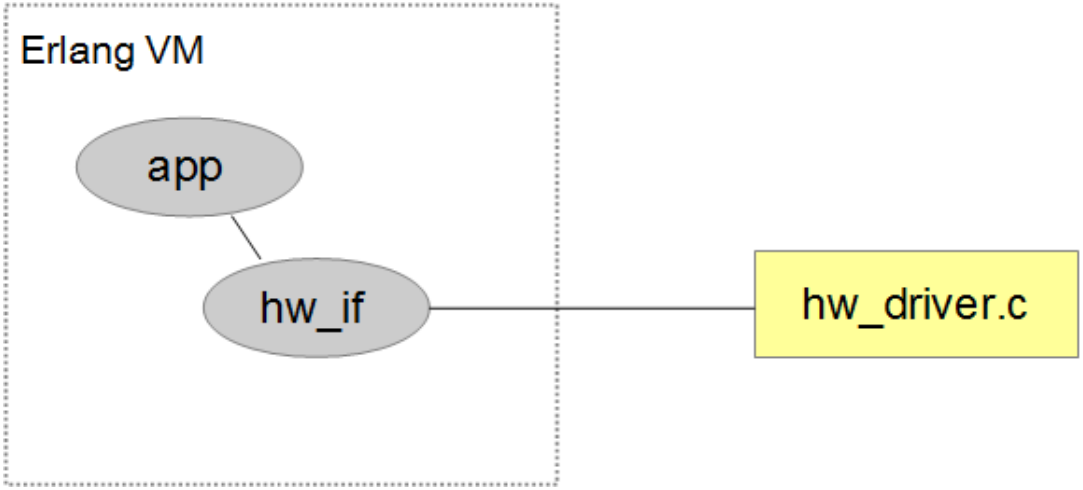
Raspberry Pi Peripherals

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

Erlang SOLUTIONS

External Interfaces in Erlang

- Facilities to interface the Erlang runtime to the outside world
- Used for device drivers and kernel abstractions in the embedded domain



Erlang SOLUTIONS

Accessing hardware

- 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

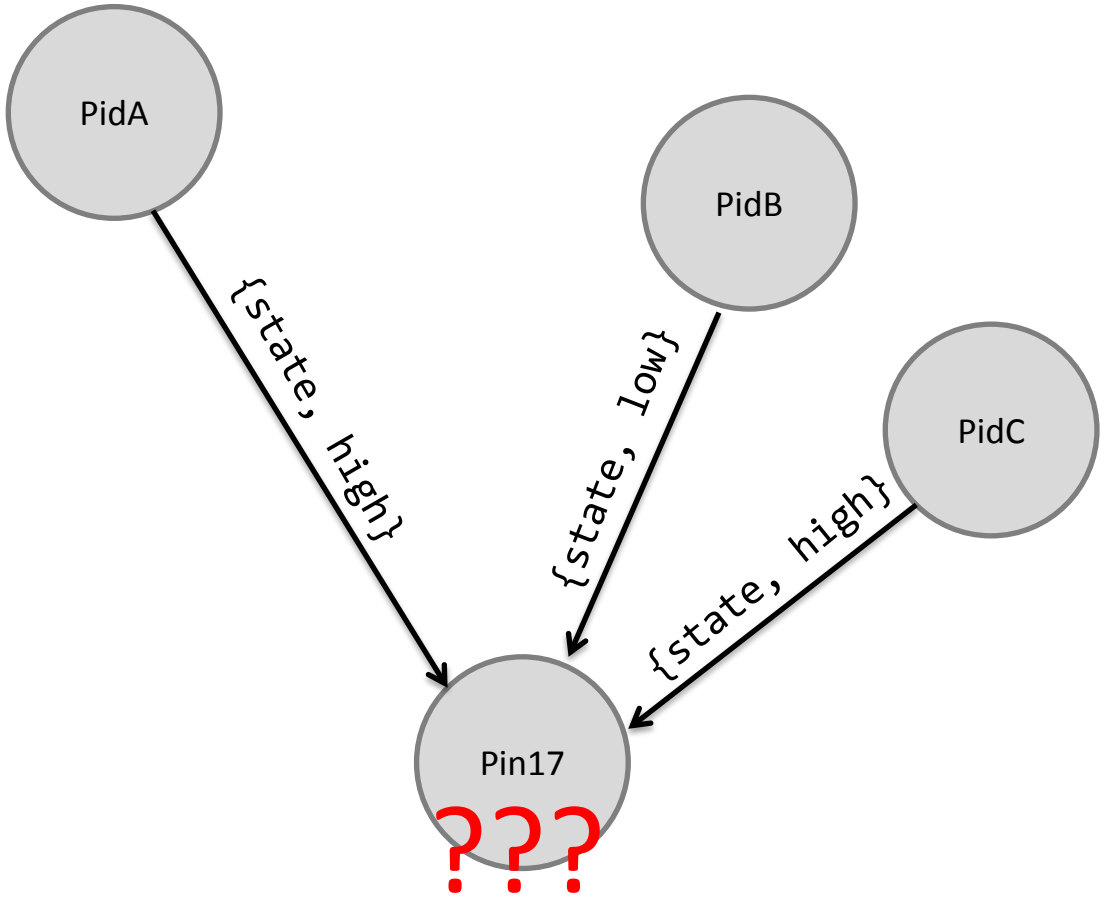
GPIO Interface (I)

```
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)

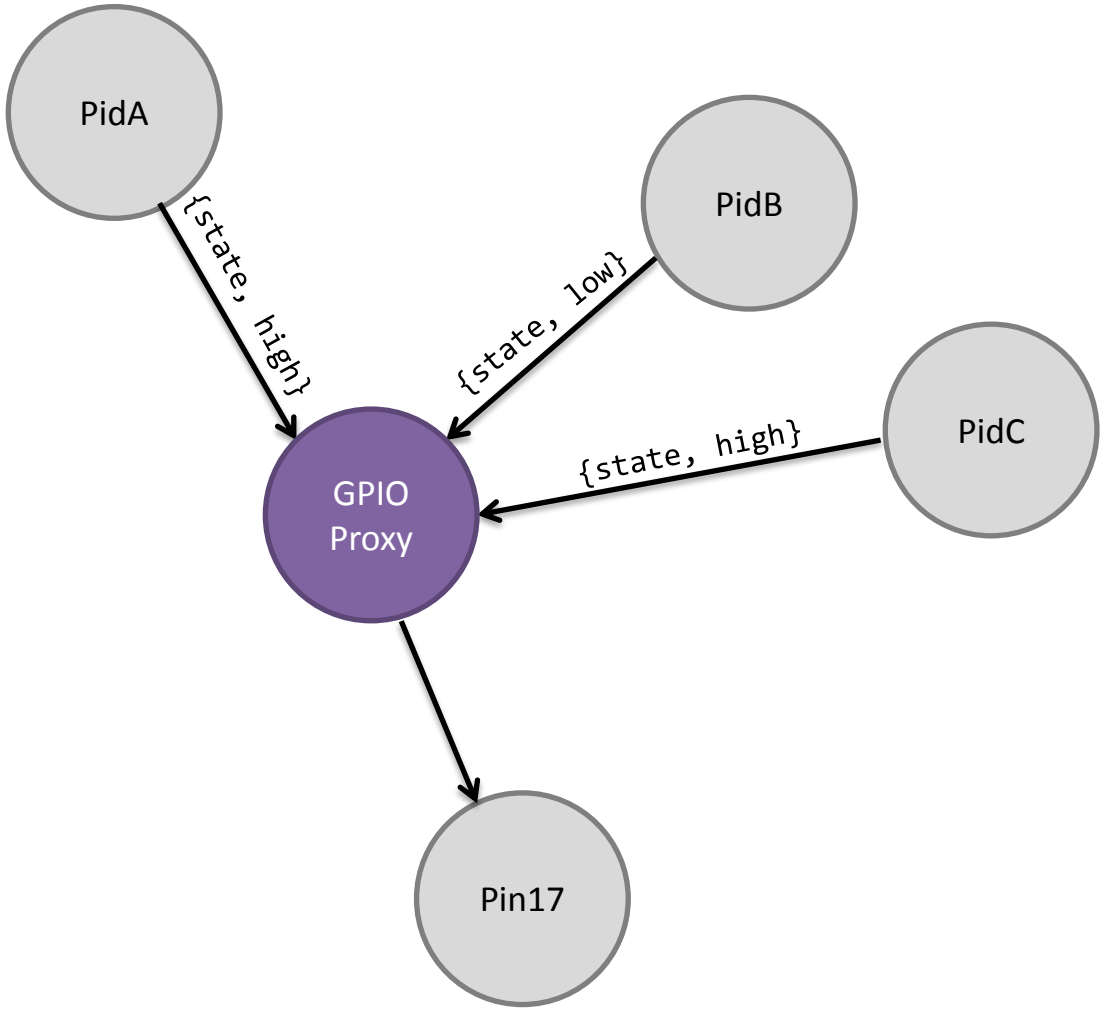
```
write(Fd, Val) ->
  file:position(Fd, 0),
  file:write(Fd, integer_to_list(Val)).


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).
```

Example: GPIO



Erlang SOLUTIONS

Example: GPIO



Erlang SOLUTIONS

GPIO Proxy

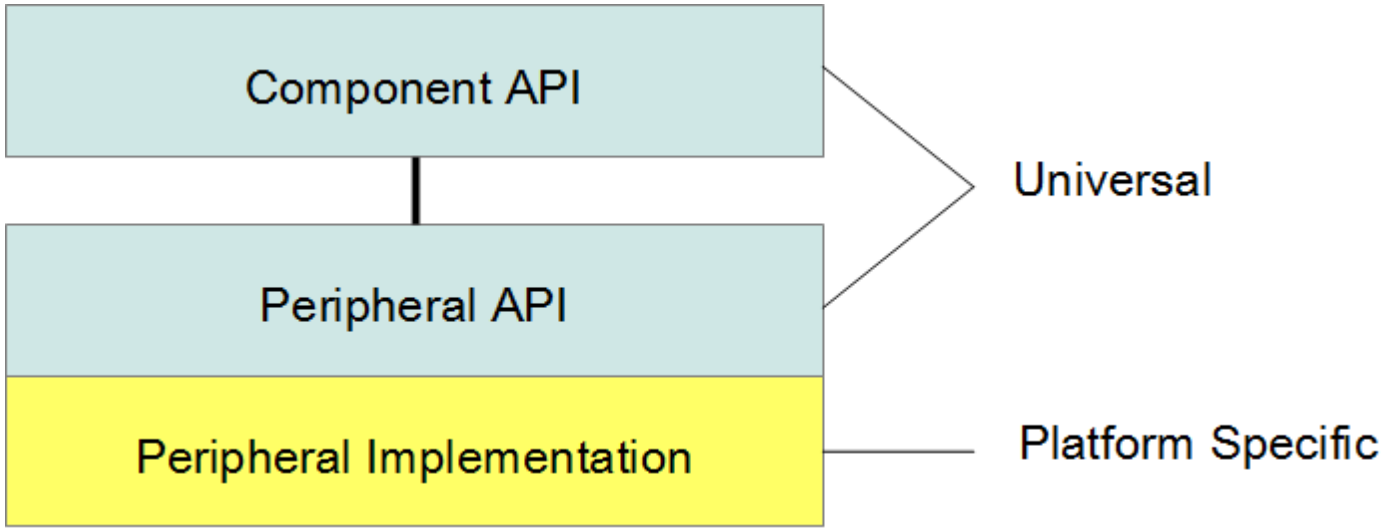
- Replaces ‘locks’ in traditional sense of embedded design
  - Access control/mutual exclusion
- Can be used to implement safety constraints
  - Toggling rate, sequence detection, direction control, etc.

Concurrency Demo

```
 1 -module(led).
 2 -export([start/1, stop/1, loop/2]).
 3
 4 start(Pin) ->
 5   Fd = gpio:init(Pin, out),
 6   Pid = spawn(?MODULE, loop, [Fd, Pin]),
 7   Pid.
 8
 9 stop(Pid) ->
10   Pid ! stop.
11
12 loop(Fd, Pin) ->
13   receive
14     on ->
15       file:write(Fd, "1"),
16       loop(Fd, Pin);
17     off ->
18       file:write(Fd, "0"),
19       loop(Fd, Pin);
20     {blink, Delay} ->
21       file:write(Fd, "1"),
22       timer:sleep(Delay),
23       file:write(Fd, "0"),
24       timer:sleep(Delay),
25       self() ! {blink, Delay},
26       loop(Fd, Pin);
27     stop ->
28       file:close(Fd),
29       gpio:release(Pin),
30       ok
31   end.
32
"led.erl" 32L, 571C                                          28,1          All
[0] 0:vim*                                    "raspberrypi" 17:17 20-Apr-12
```


Erlang SOLUTIONS

Universal Peripheral/Component Modules

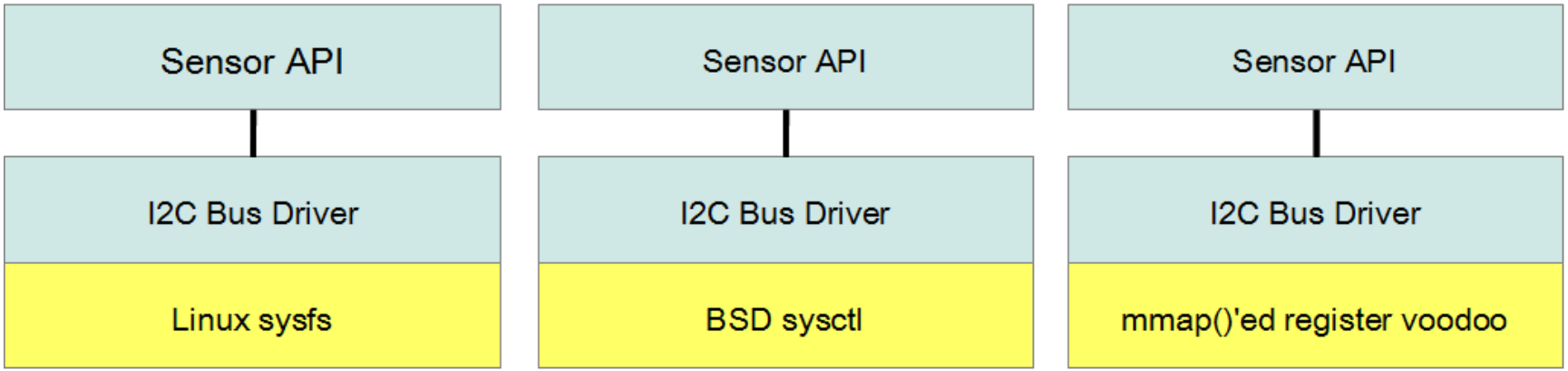


Erlang SOLUTIONS

Universal Peripheral/Component Modules

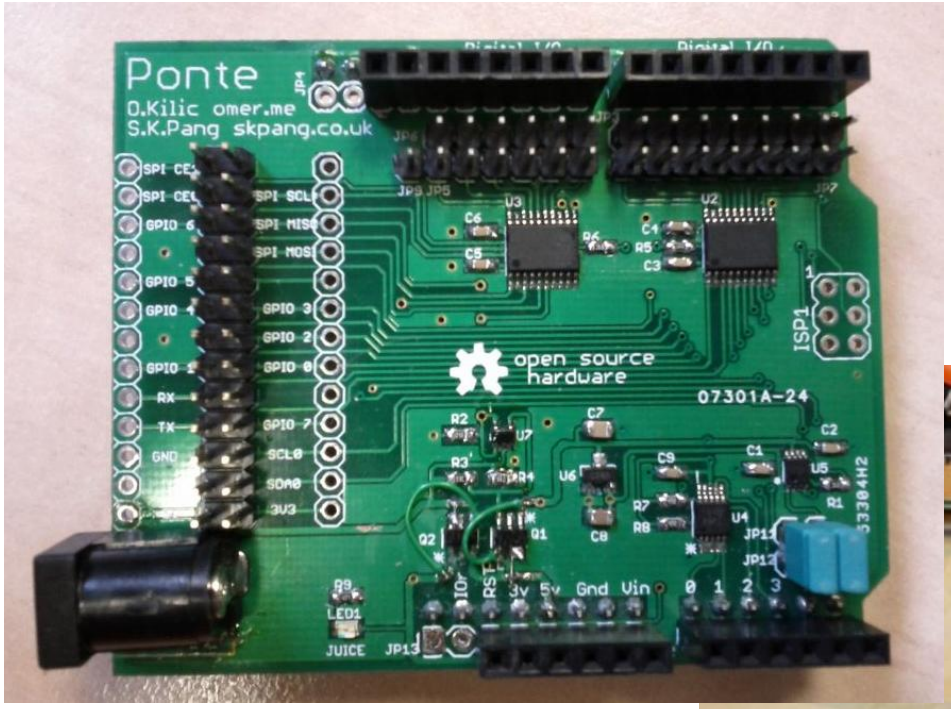
An Example:

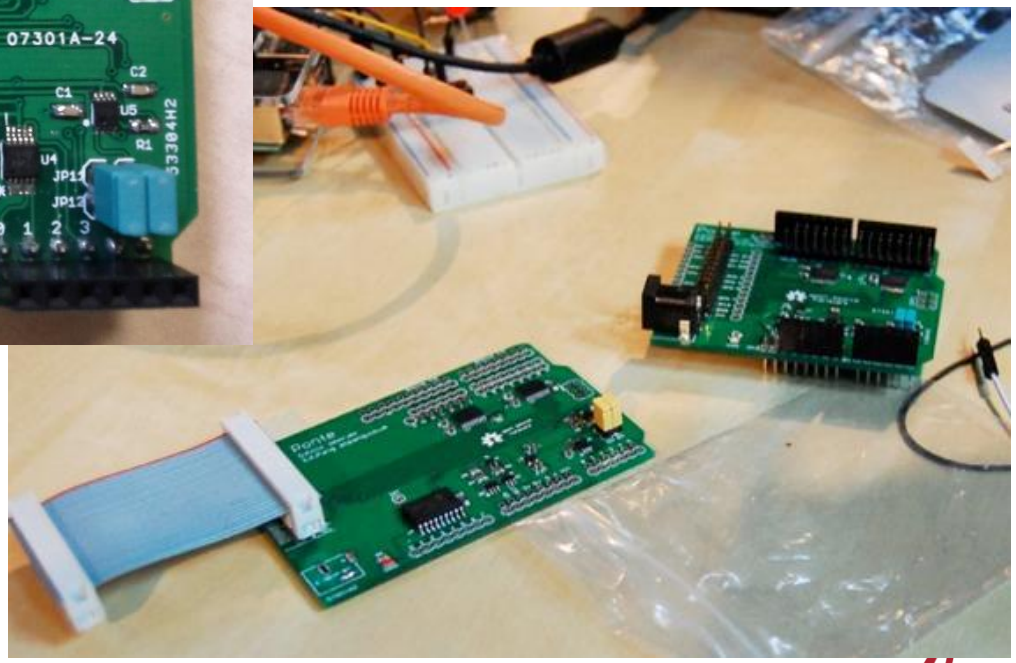
Temperature Sensor with I2C Interface



Erlang SOLUTIONS

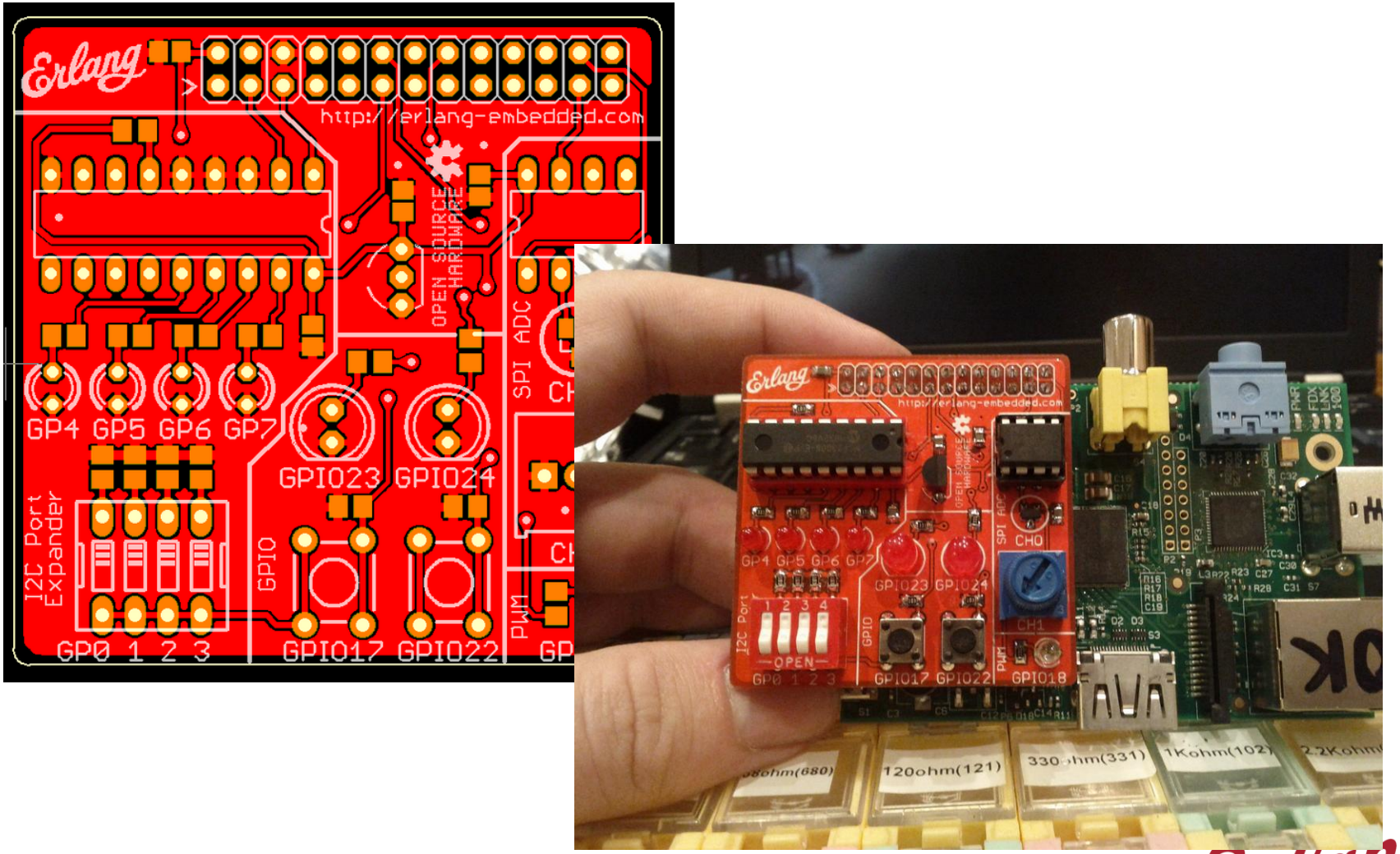
Hardware Projects – Ponte





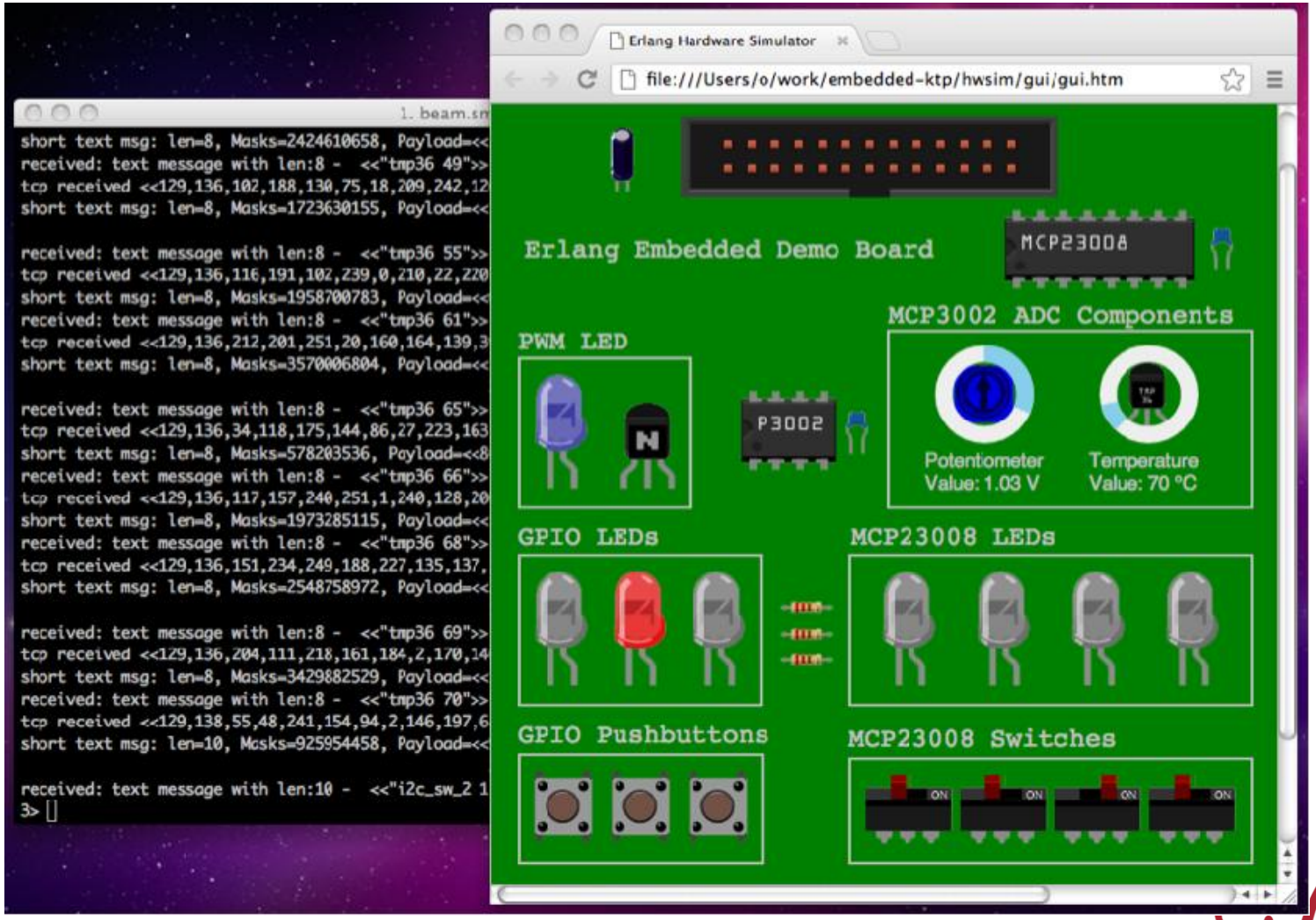
Erlang SOLUTIONS

Hardware Projects – Demo Board



Erlang SOLUTIONS

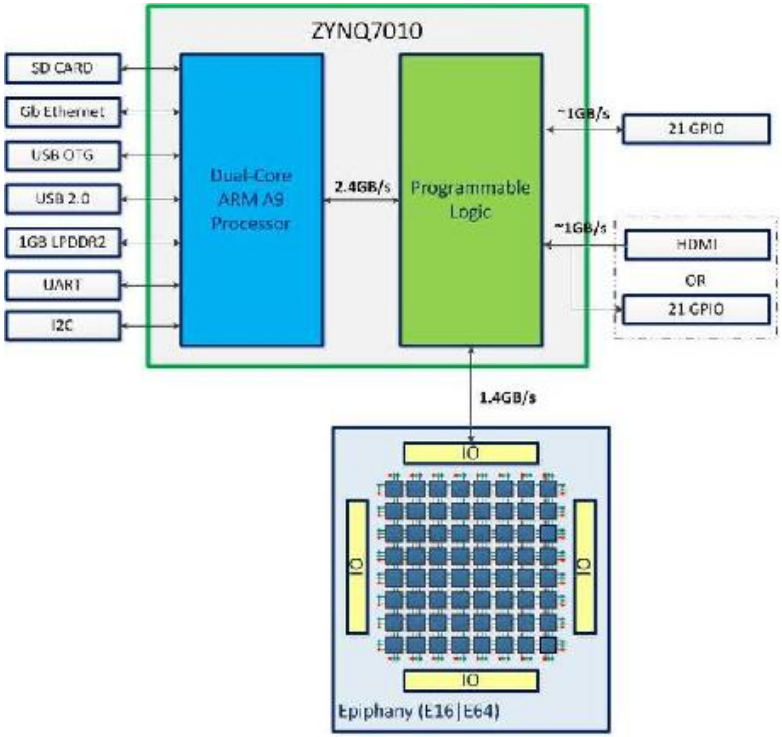
Hardware Simulator

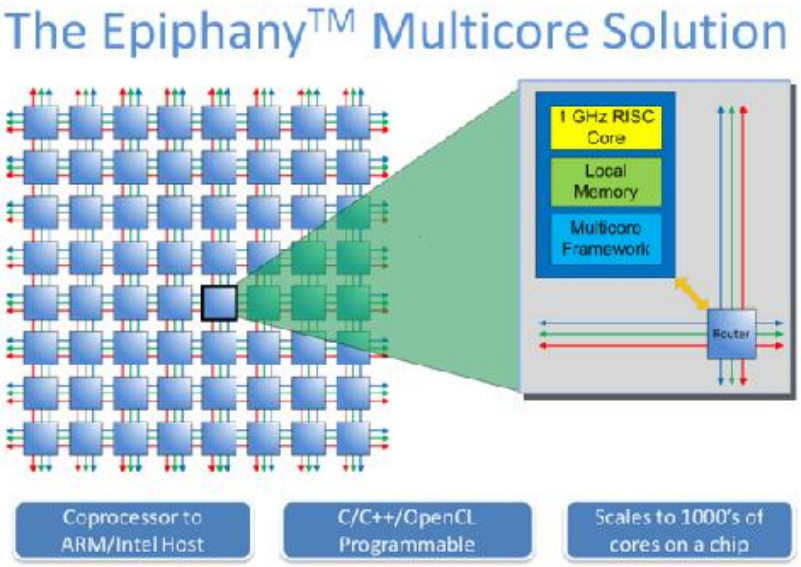


Erlang SOLUTIONS

Future Explorations

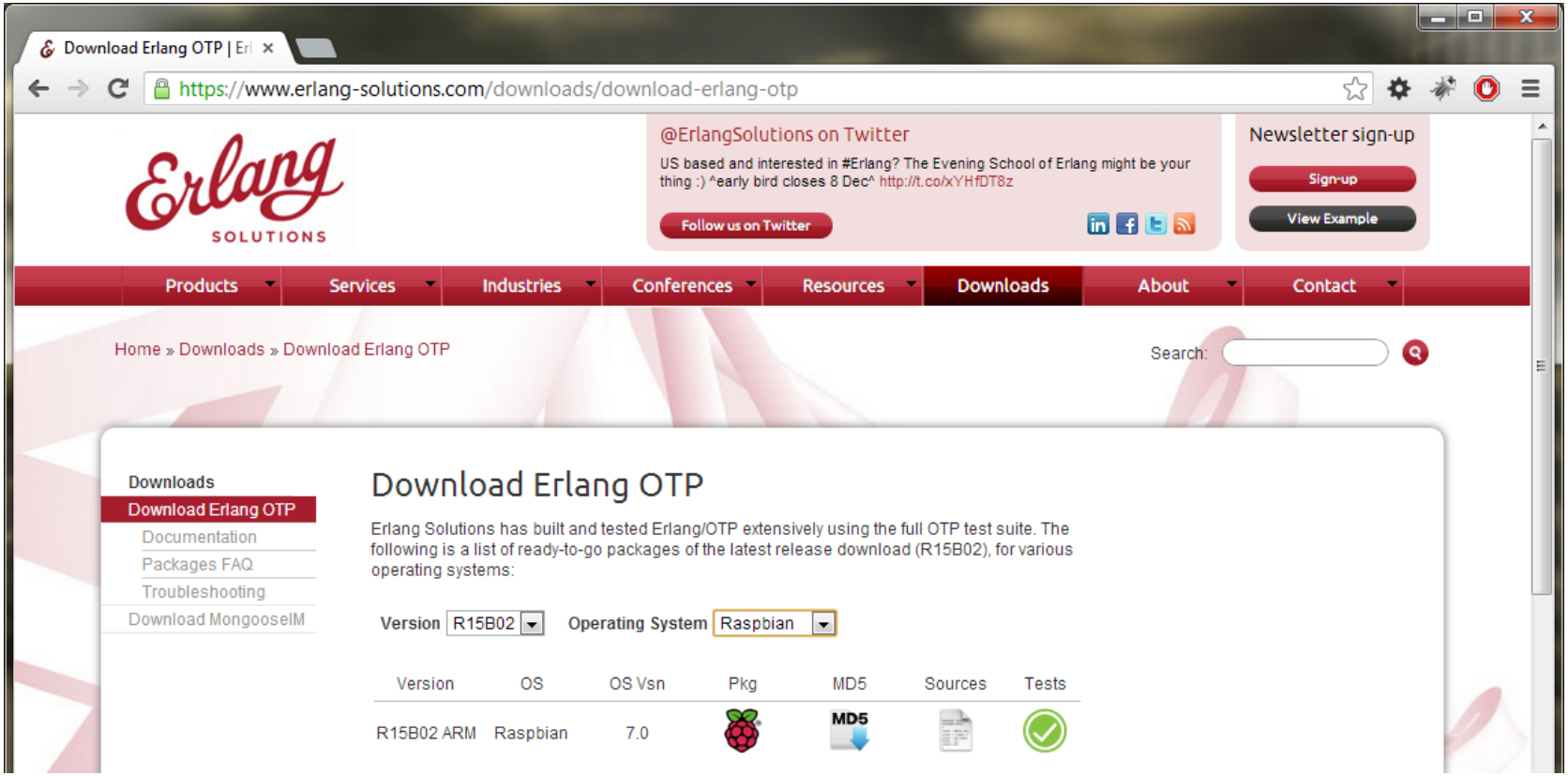
Parallella:





Erlang SOLUTIONS

Download Erlang for Raspbian



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

Erlang SOLUTIONS

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

Father of Erlang

Erlang SOLUTIONS