Taking Back Embedded

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

" *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

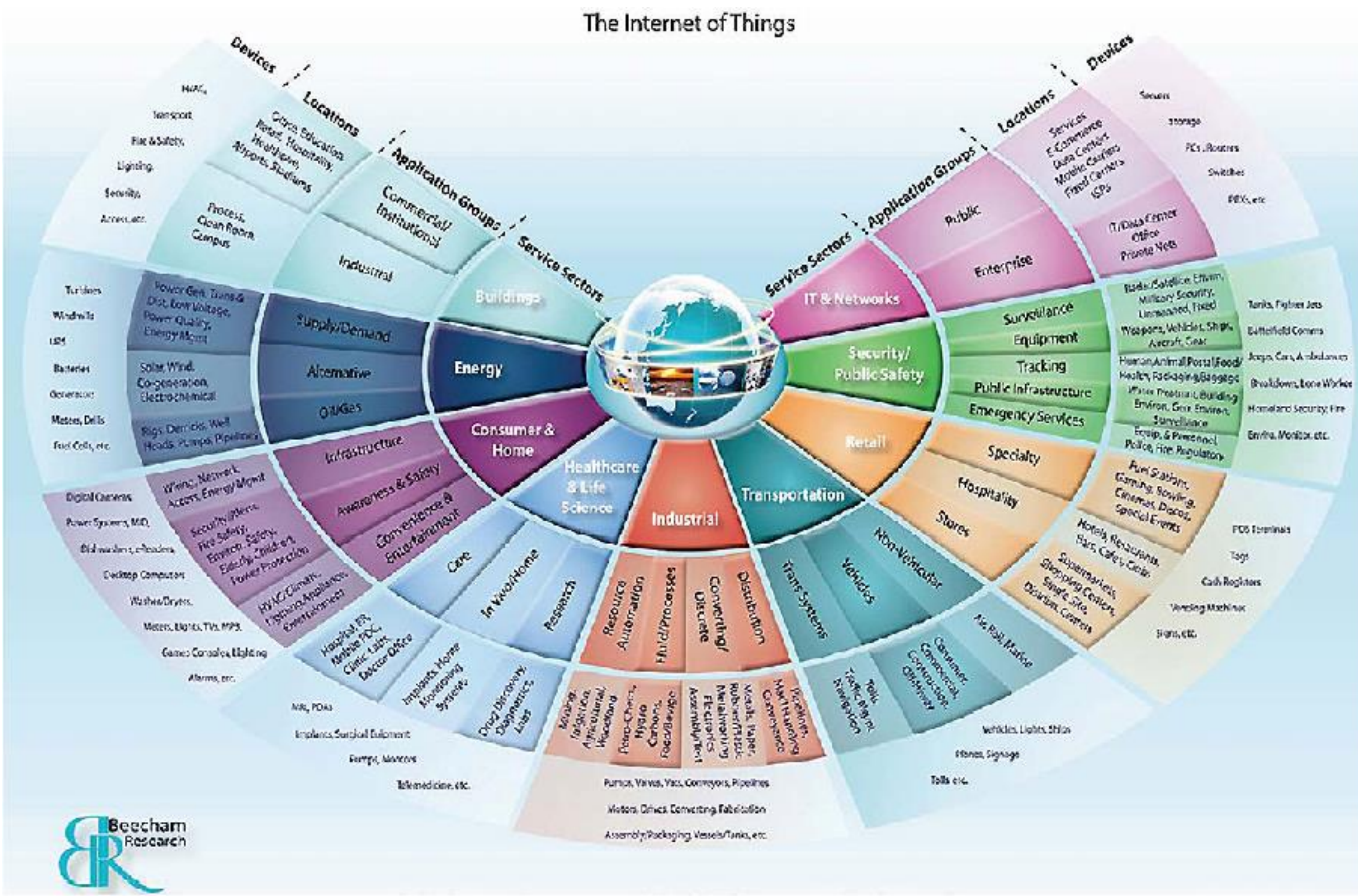
- Specific functions
- Designed for a particular application

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

Current Challenges

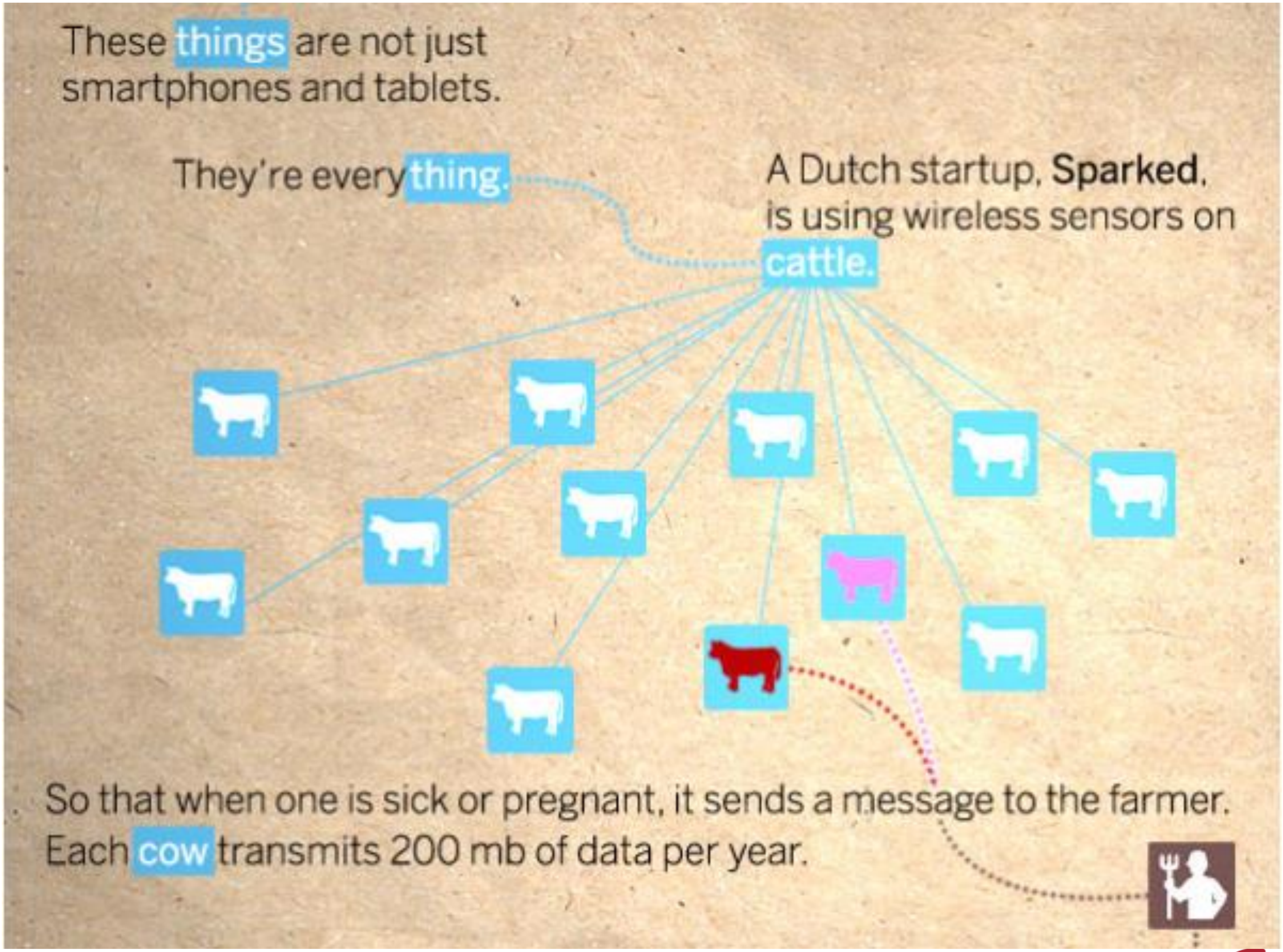
- Complex SoC platforms
- "Internet of Things"
  - Connected and distributed systems
- Multicore and/or heterogeneous devices
- Time to market constraints
  - The Kickstarter Era
  - Rapid prototyping
  - Maker Culture

Internet of Things

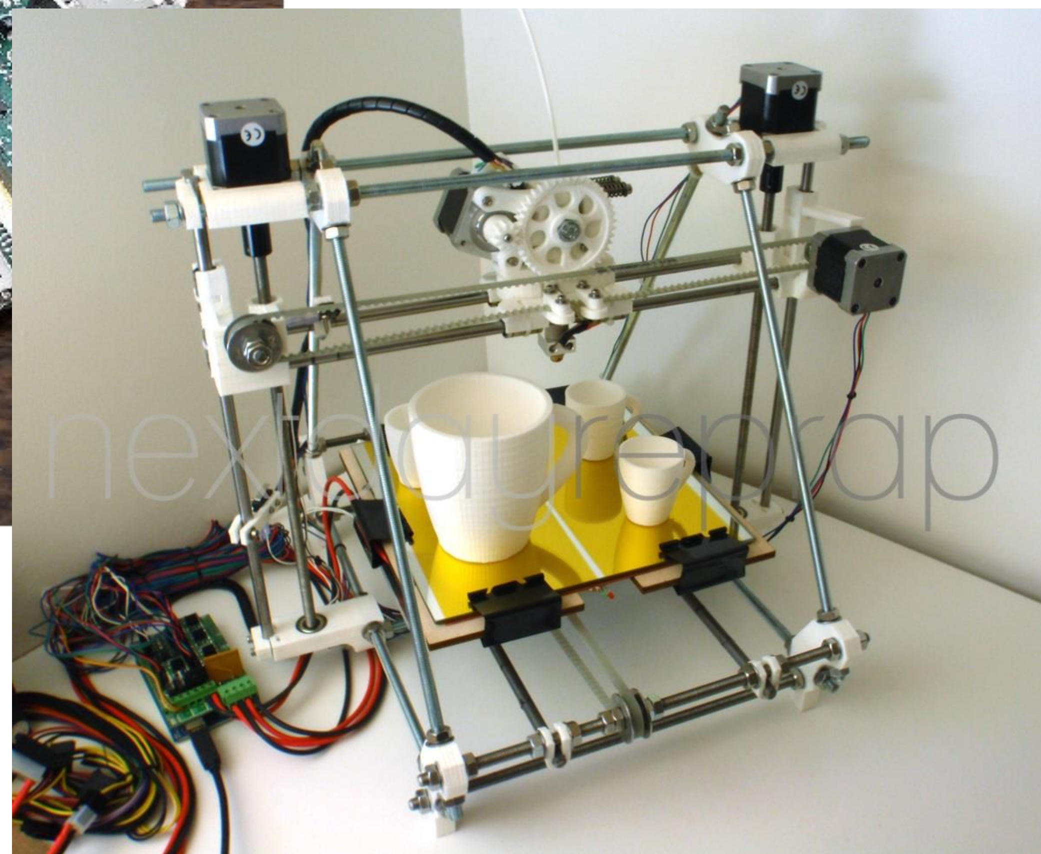


Internet of Fridges?

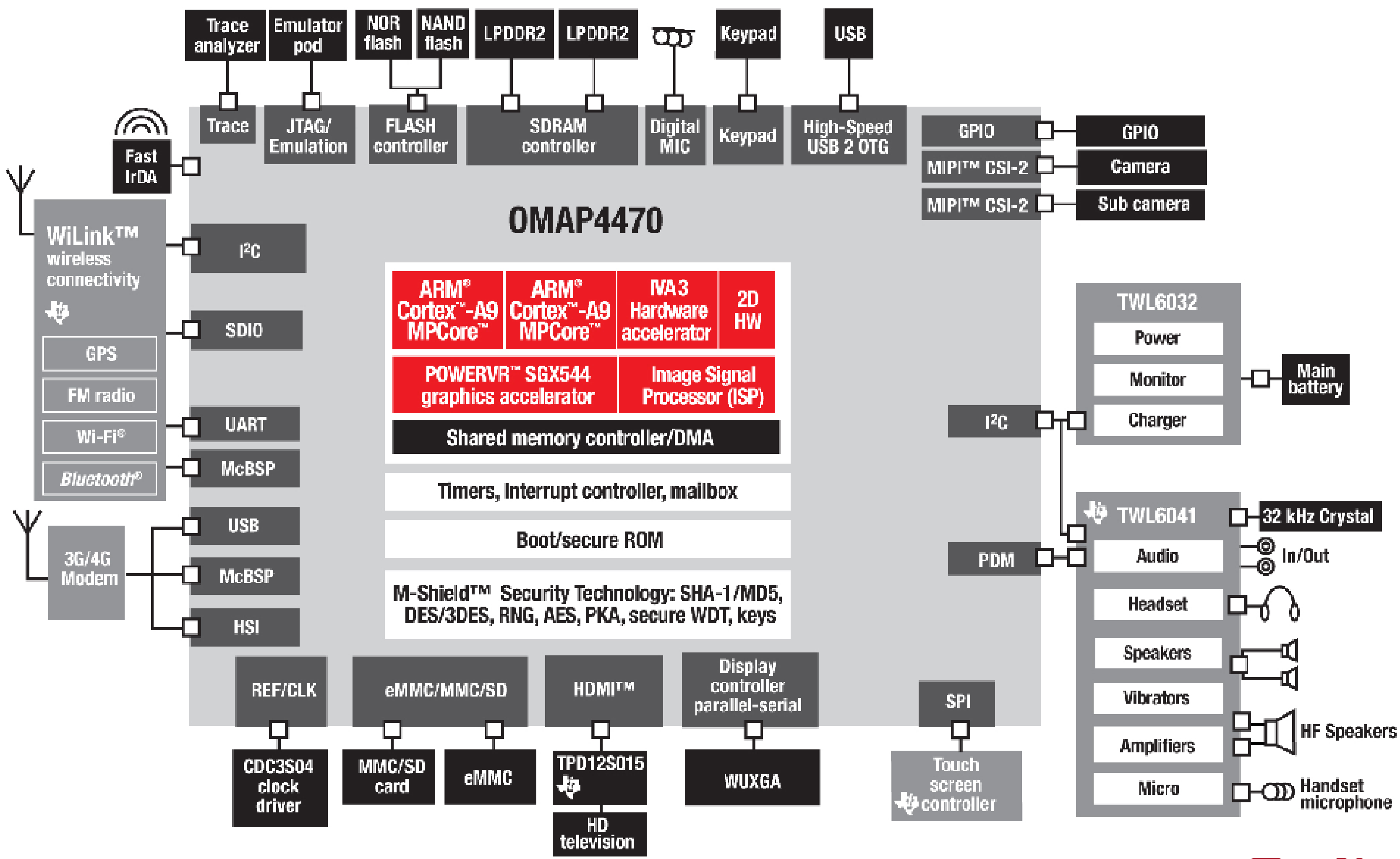
Distributed Bovine Networks?



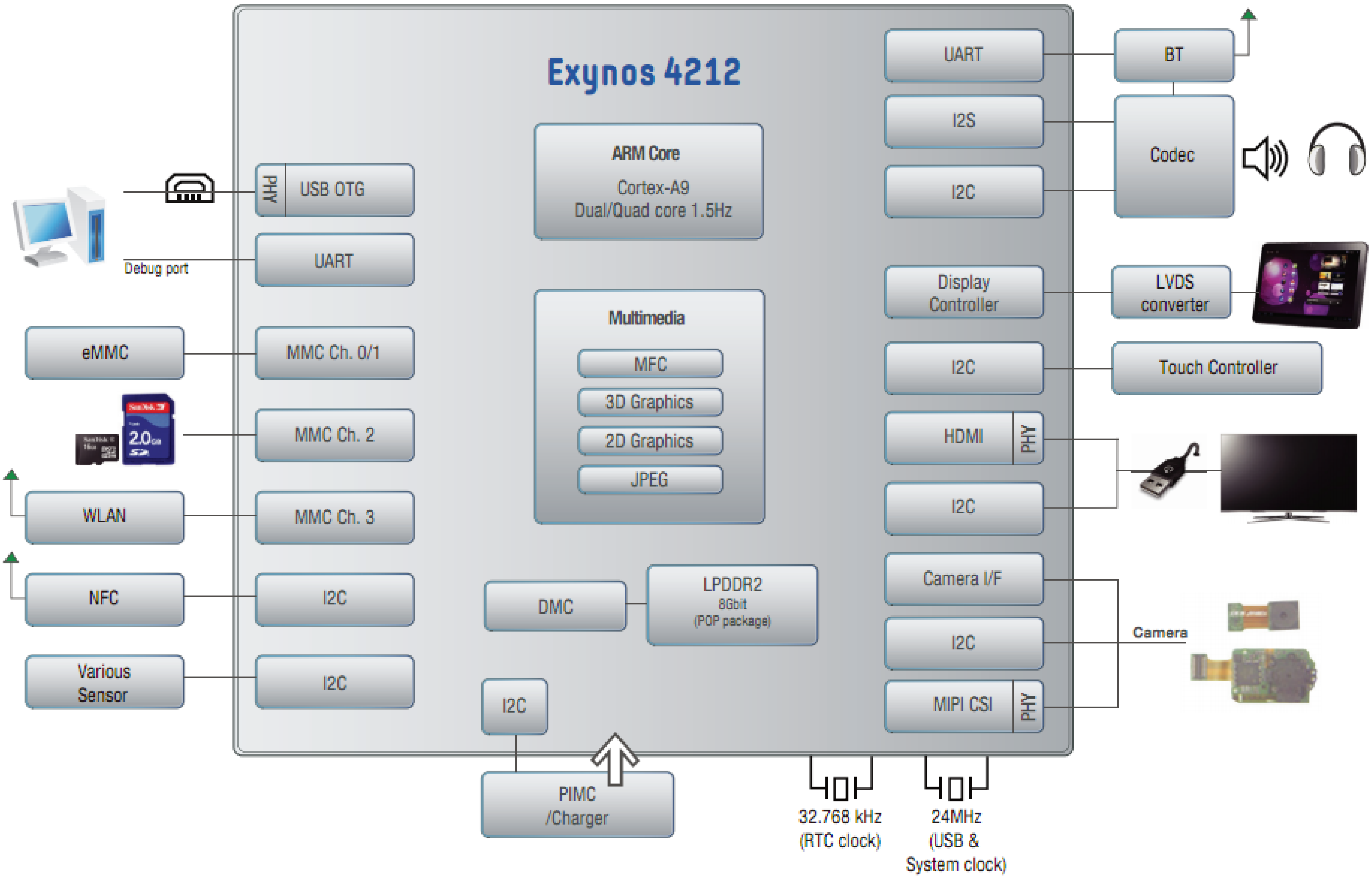
Exciting times



TI OMAP Reference System

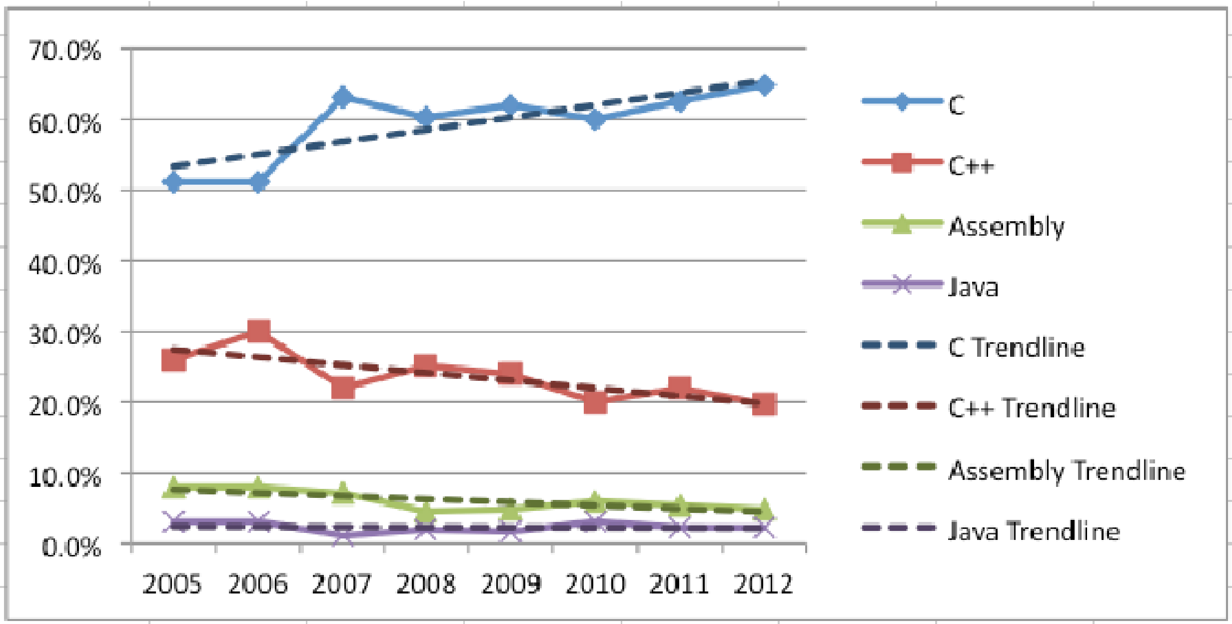


Samsung Exynos Reference System



#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

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 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

Erlang? (I)

{declarative, functional, concurrent, parallel, garbage-collected, soft real-time, fault-tolerant, robust,

portable, distributed

message-passing, hot code loading}

Erlang? (II)

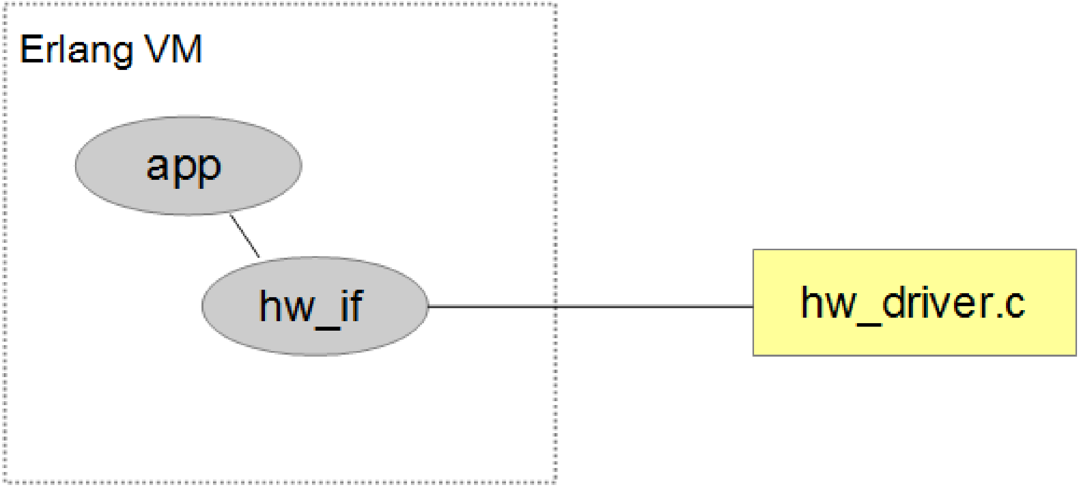
- First version developed in 1986
  - Open-sourced in 1998.
- Battle-tested at Ericsson and many other companies
  - Originally designed for Embedded Systems!
- Implements the Actor model
  - Support for concurrency and distributed systems out of the box
- Easy to create robust systems

High Availability/Reliability

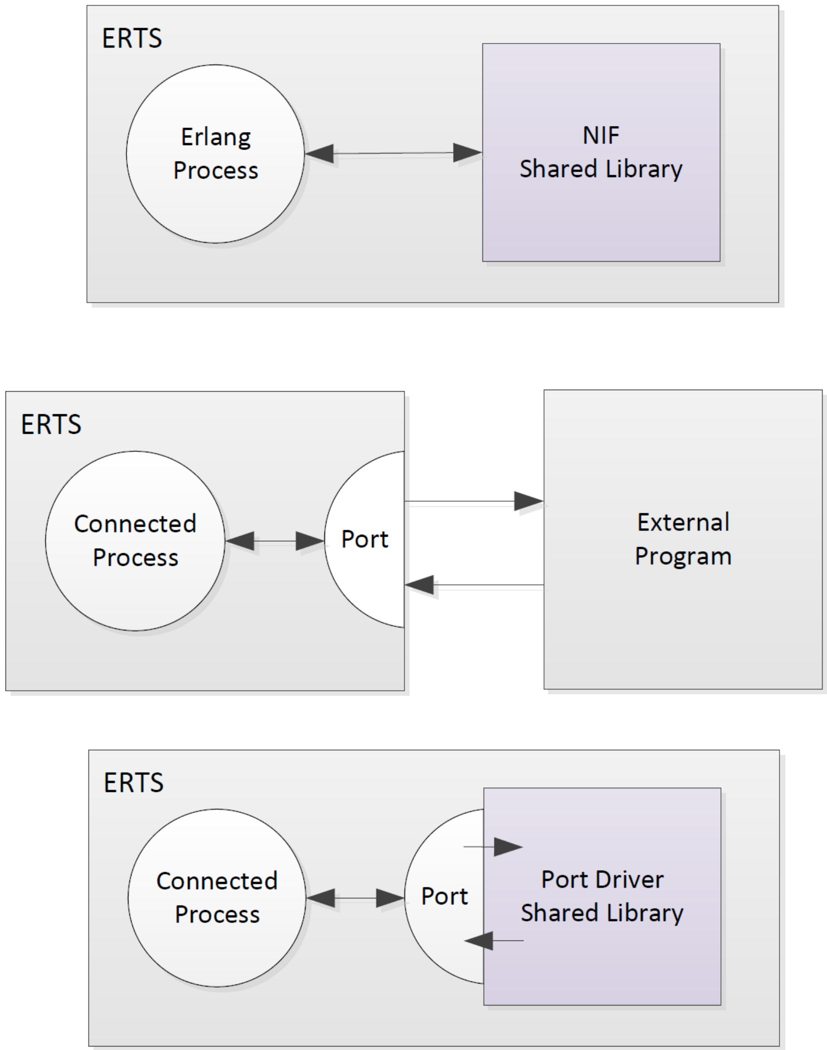
- Simple and consistent error recovery and supervision hierarchies
- Built in fault-tolerance
  - Isolation of Actors
- Support for dynamic reconfiguration
  - Hot code loading

External Interfaces

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



External Interfaces

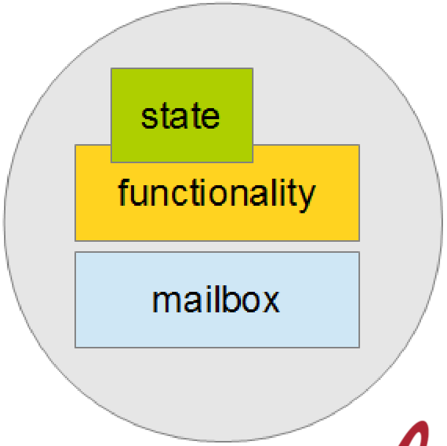


Actor Model

- 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

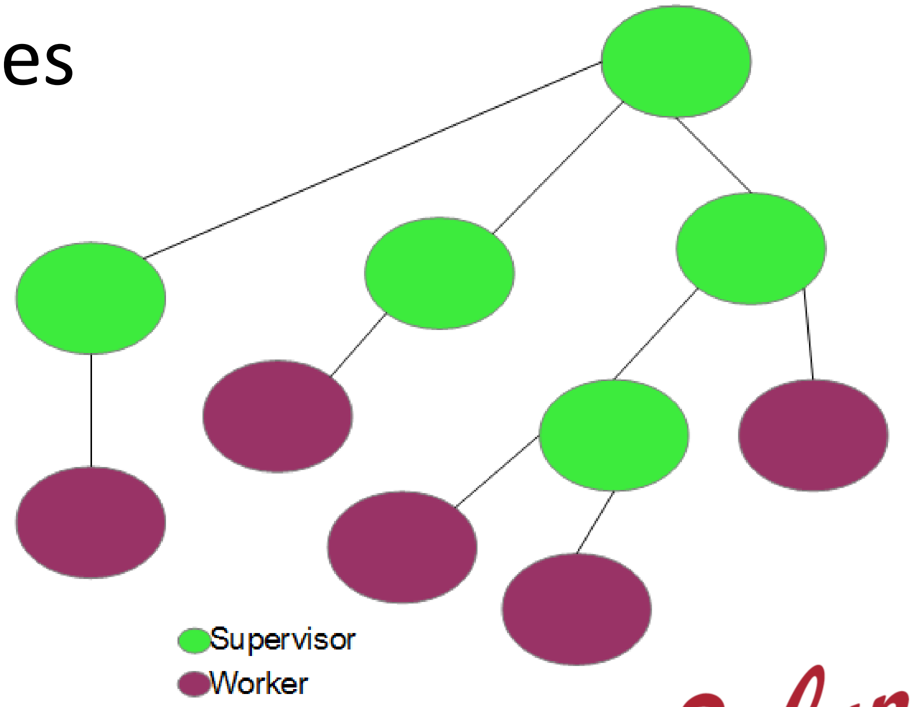
Actor Model

- 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

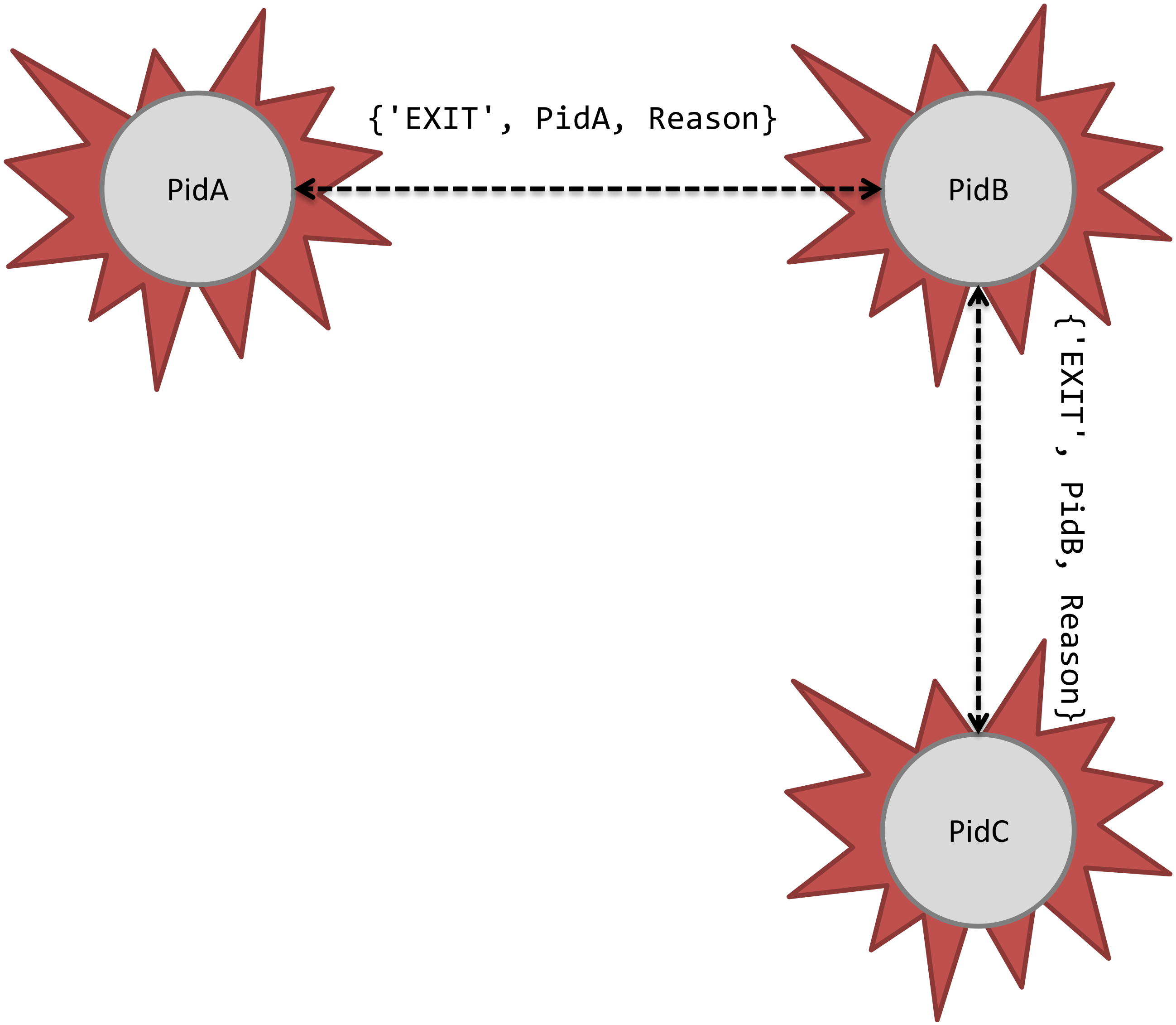


Process Error Handling

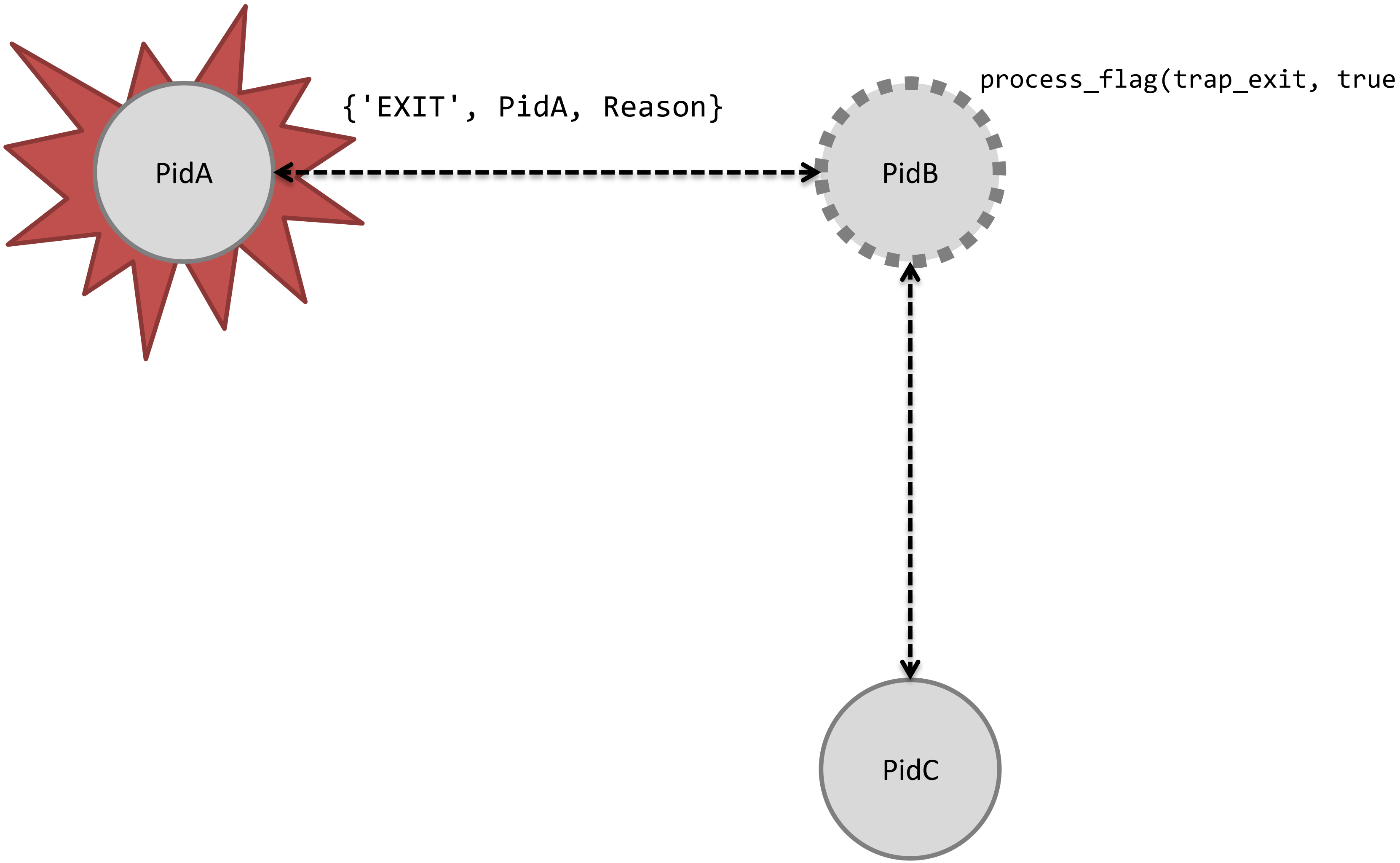
- Let it Fail!
  - Abstract error handling away from the modules
  - Results in leaner modules
- Supervision hierarchies



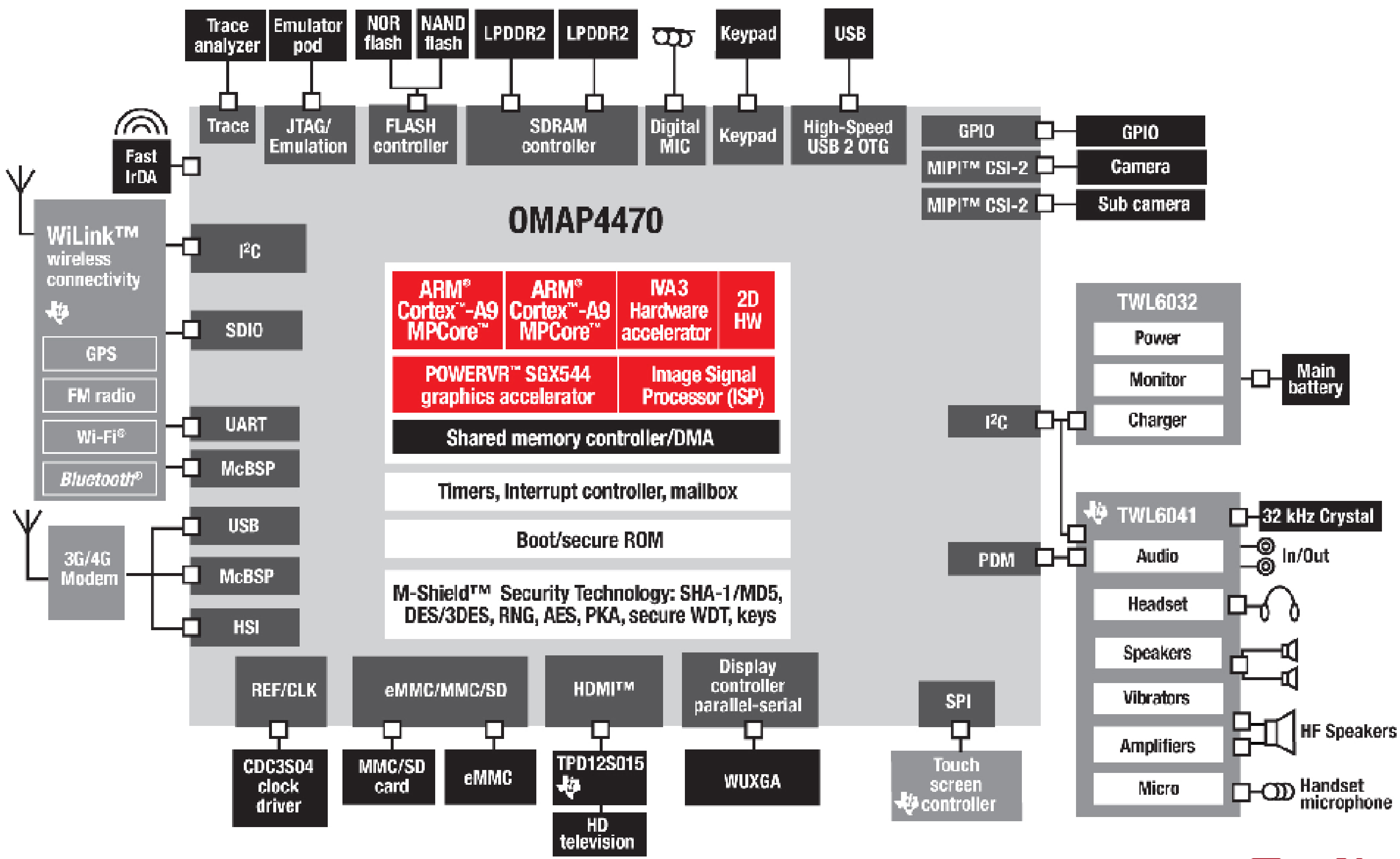
Propagating Exit Signals



Trapping Exits



TI OMAP Reference System



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

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, the Maestro

(flickr/dereckesanches)

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

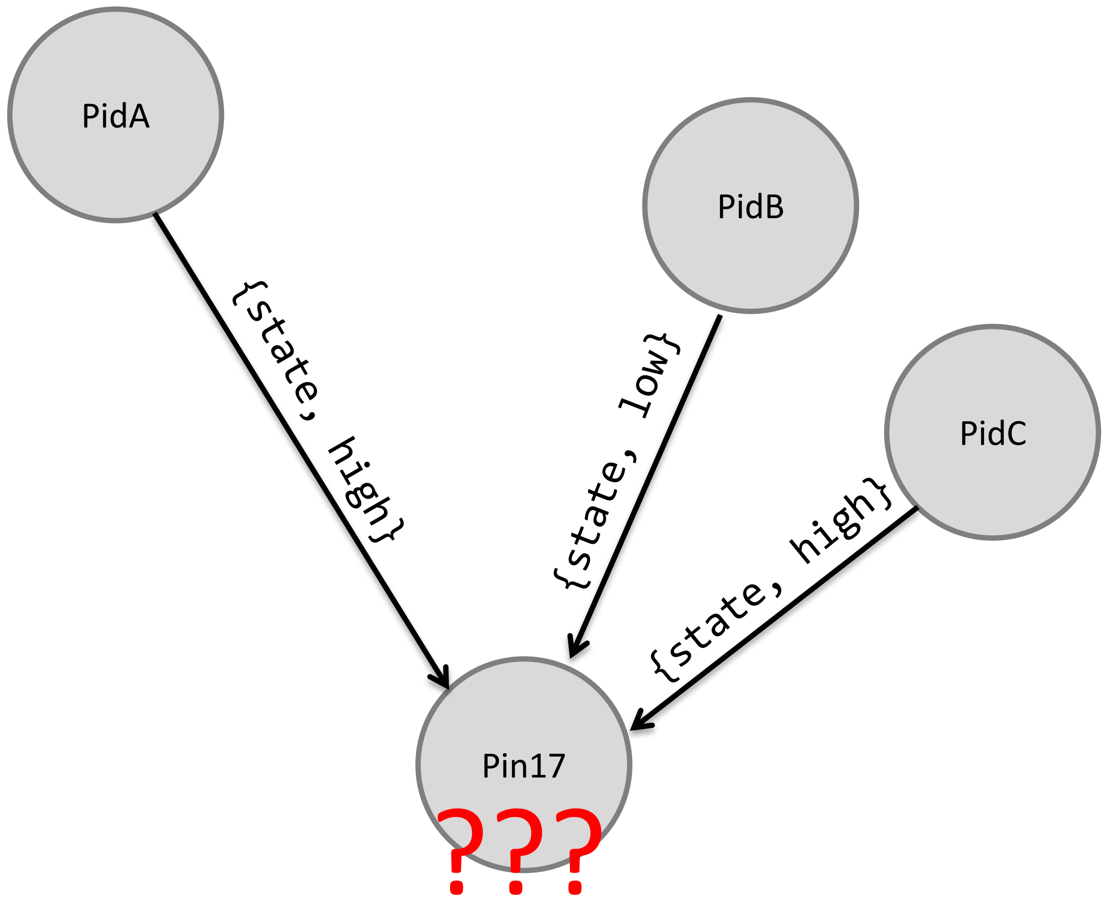
GPIO Interface (I)

```
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.
```

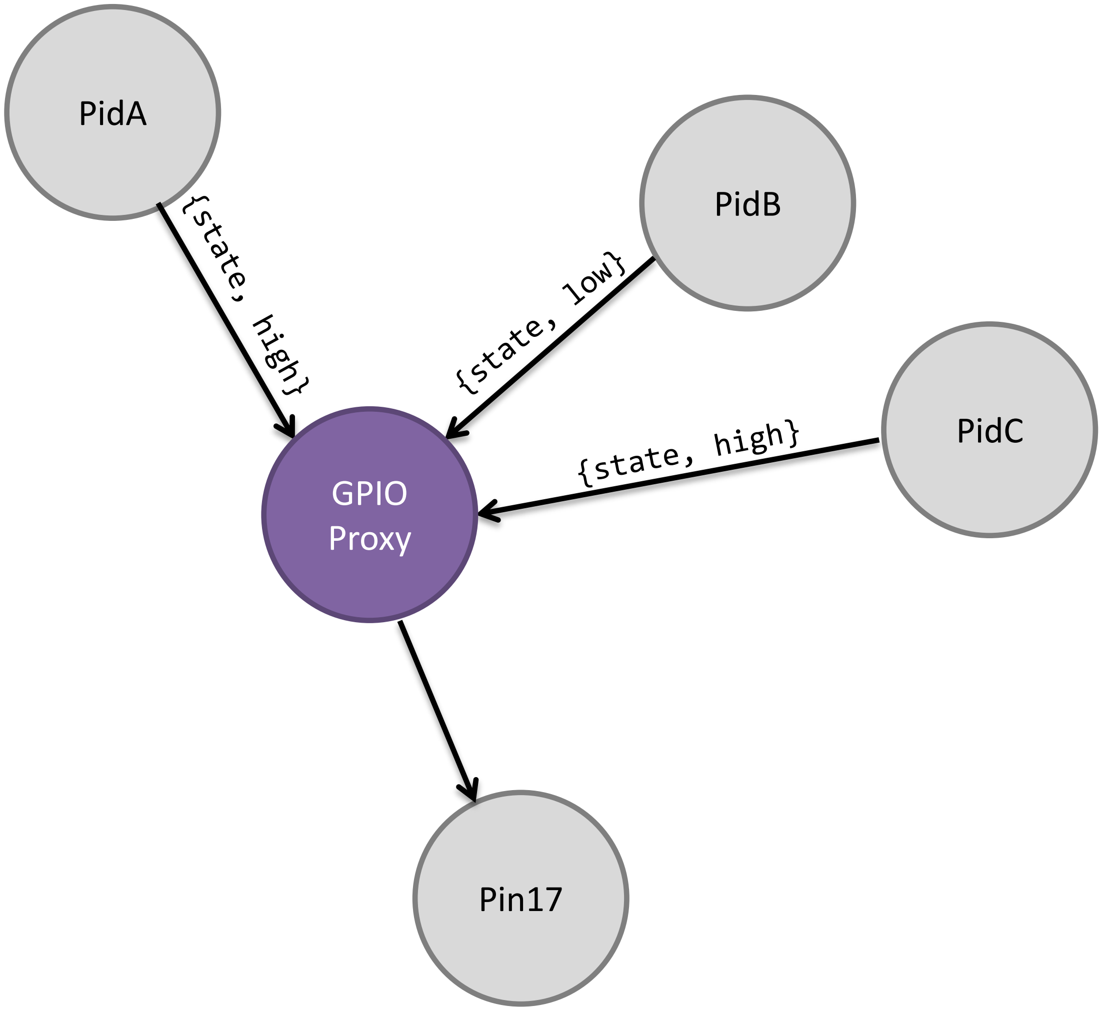
GPIO Interface (II)

```
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



Example: GPIO



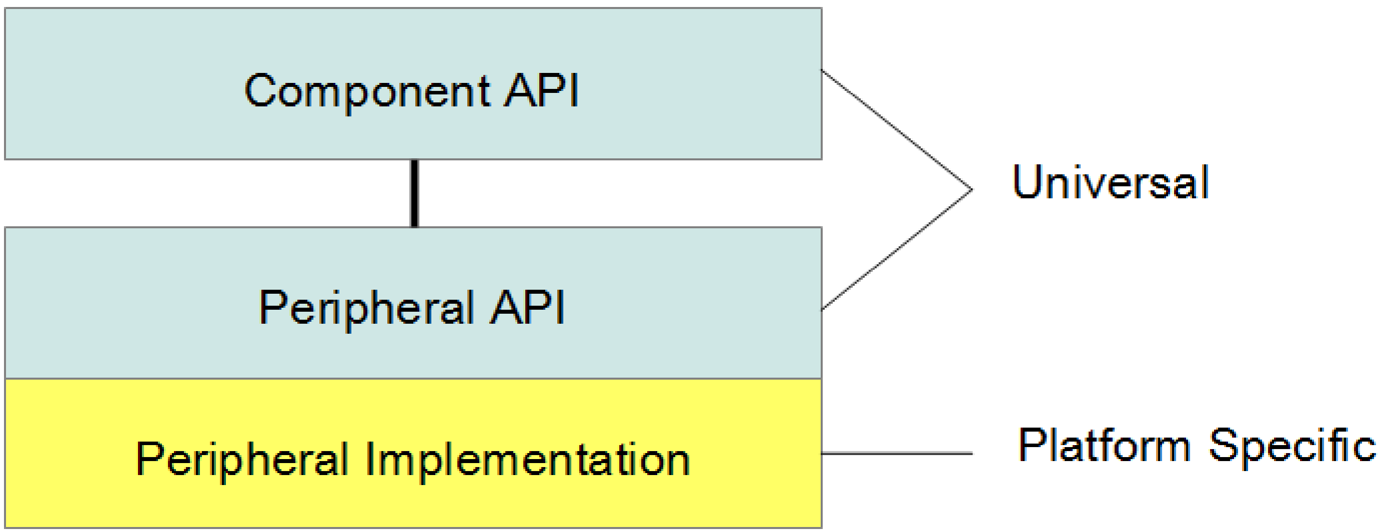
GPIO Proxy

- 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.

Concurrency Demo

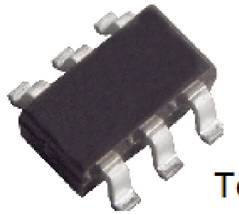
http://vimeo.com/40769788

Universal Peripheral/Component Modules

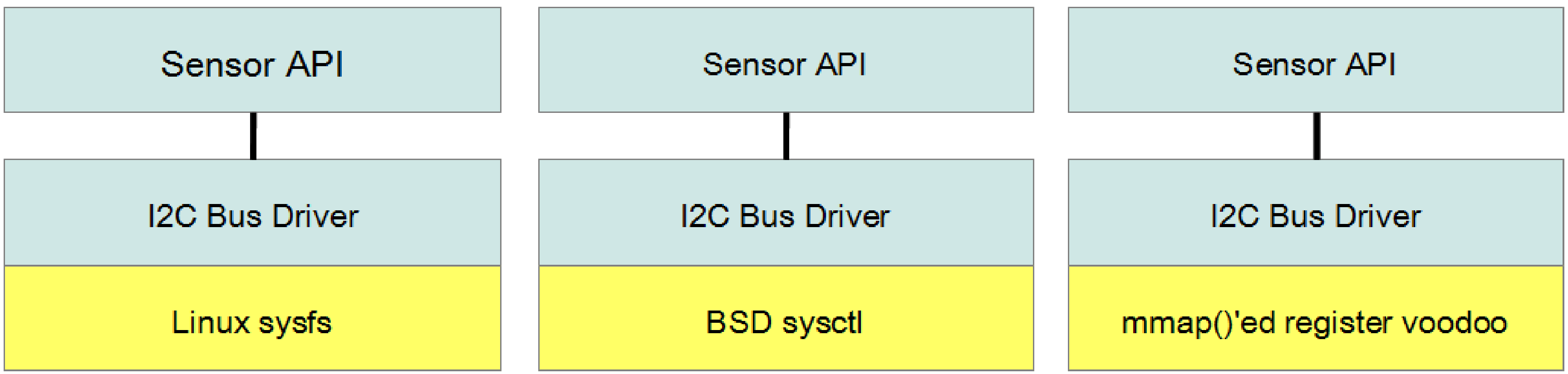


Universal Peripheral/Component Modules

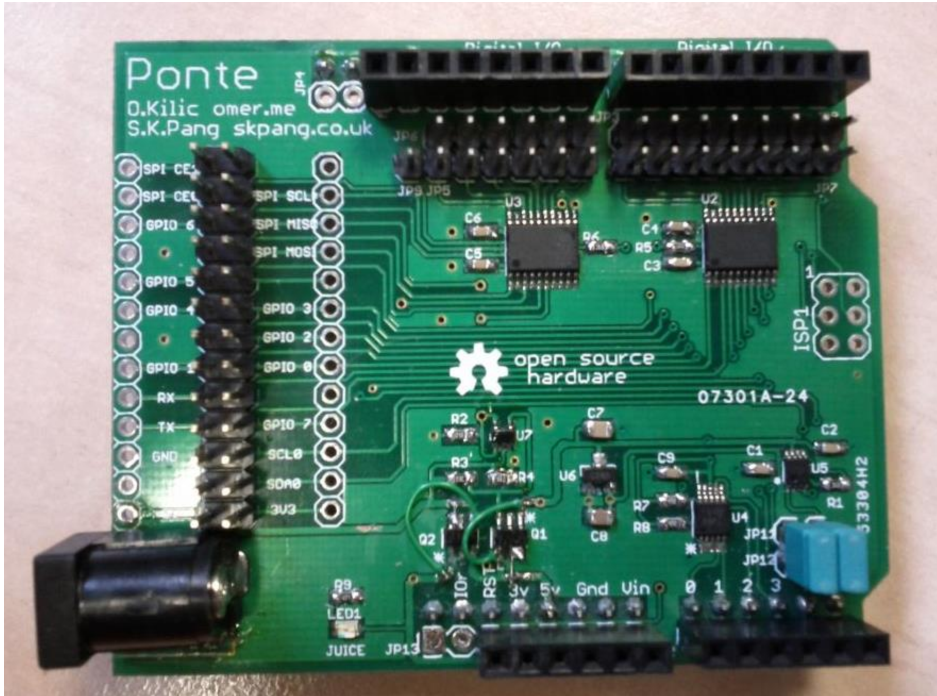
An Example:

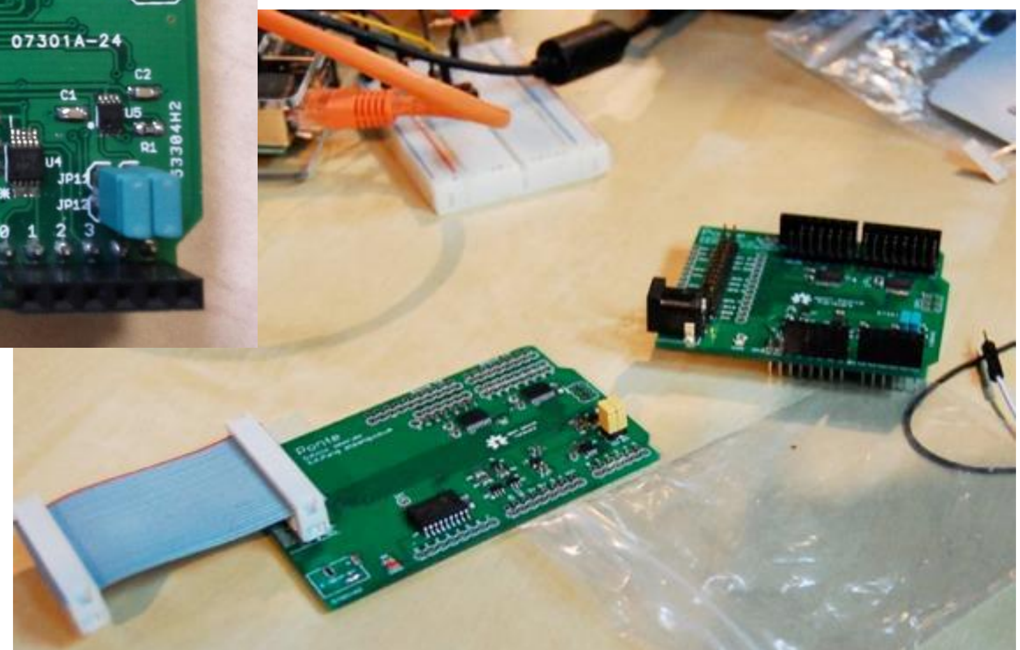


Temperature Sensor with I2C Interface

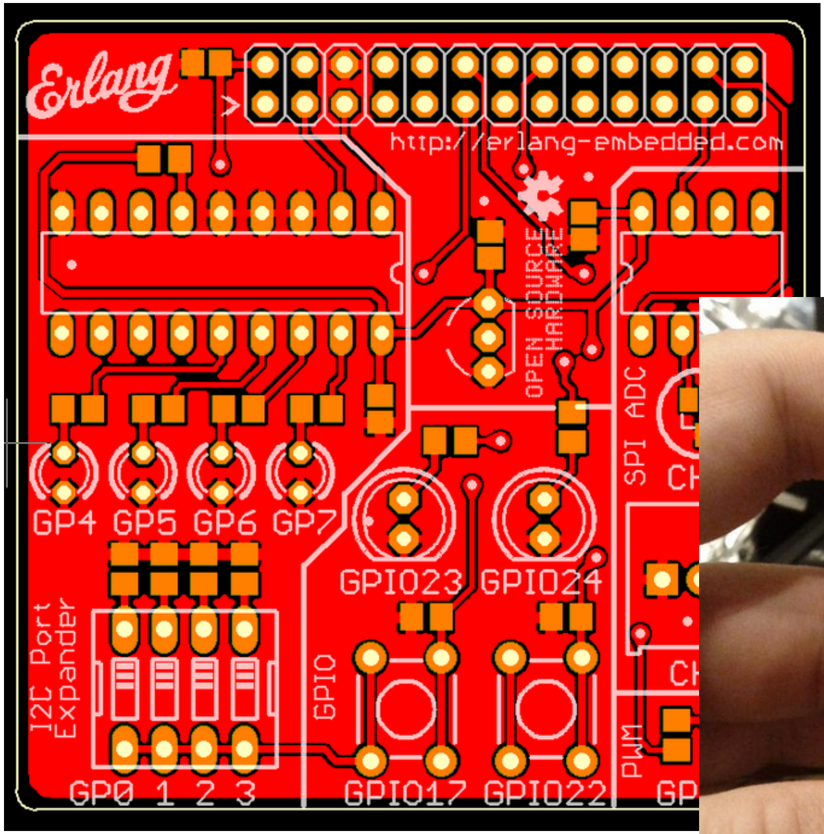


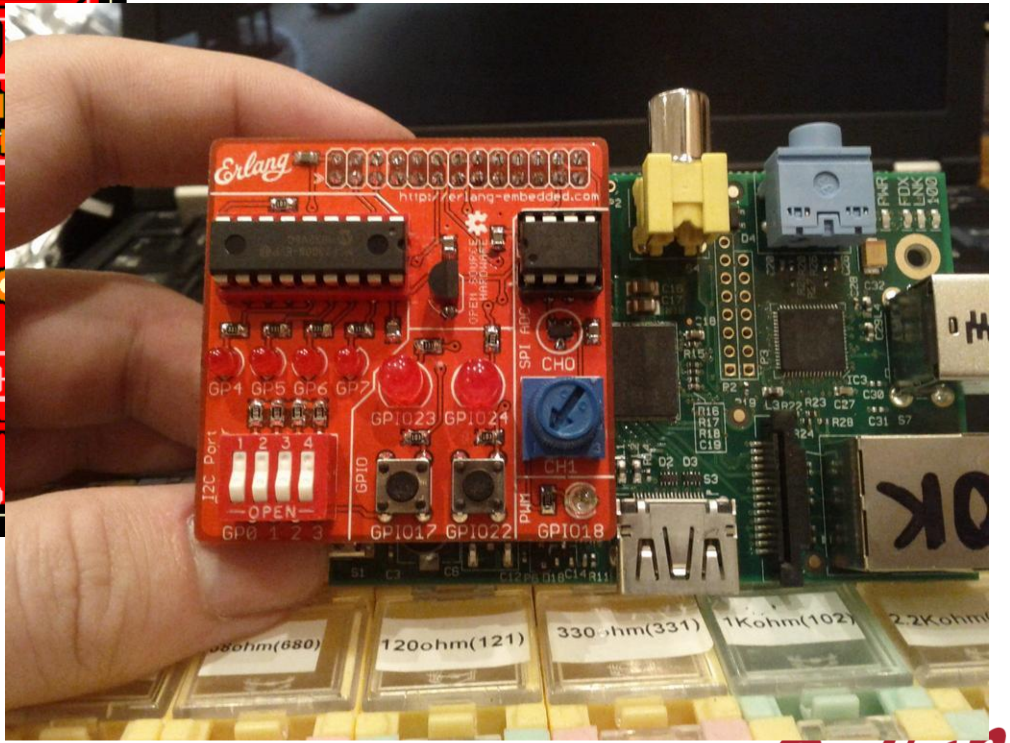
Hardware Projects – Ponte



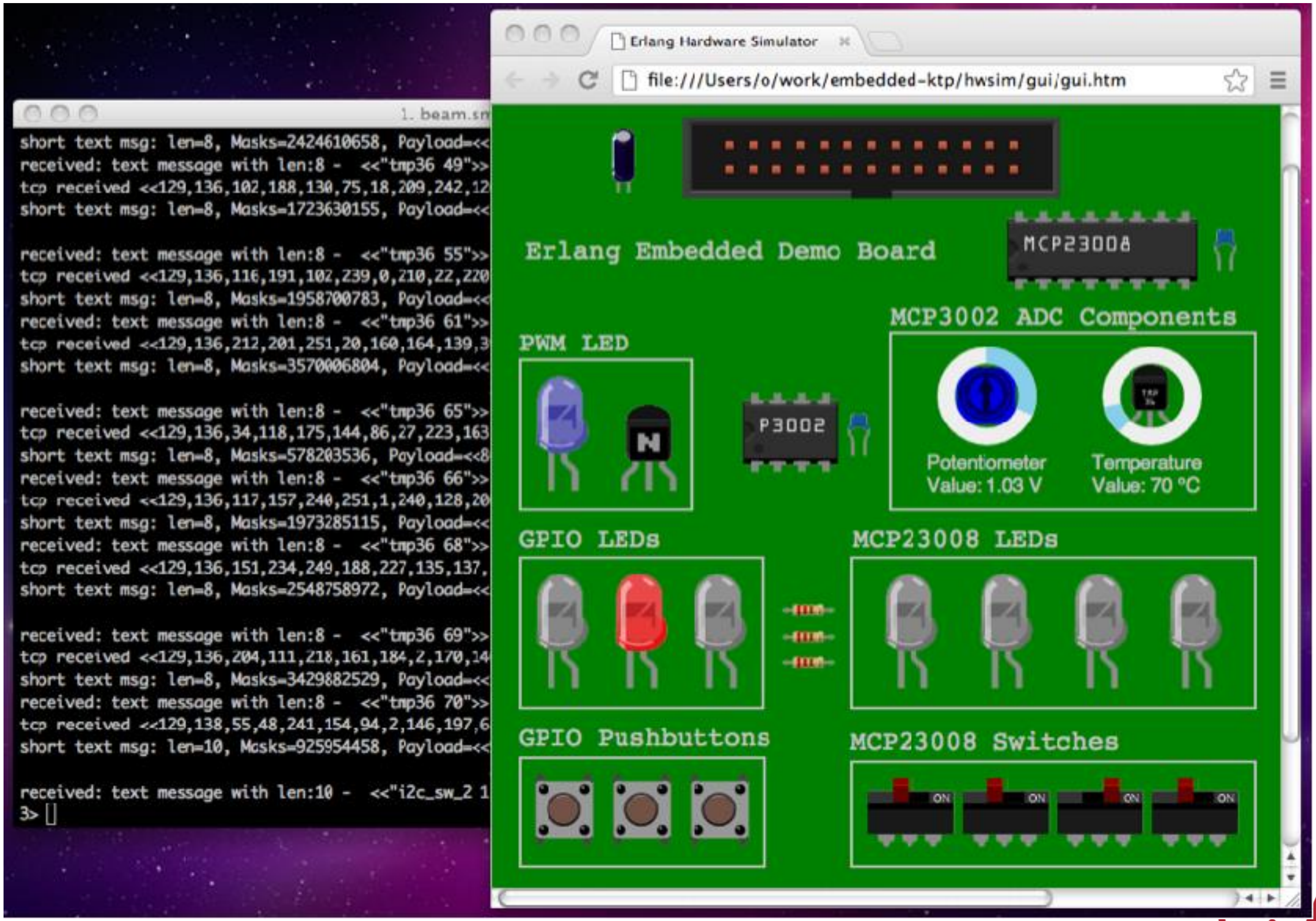


Hardware Projects – Demo Board



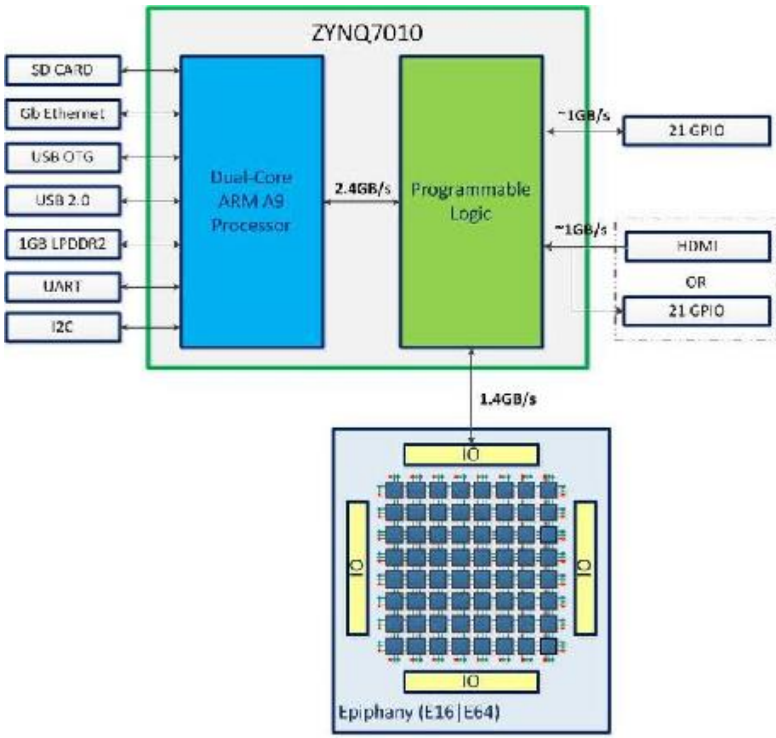


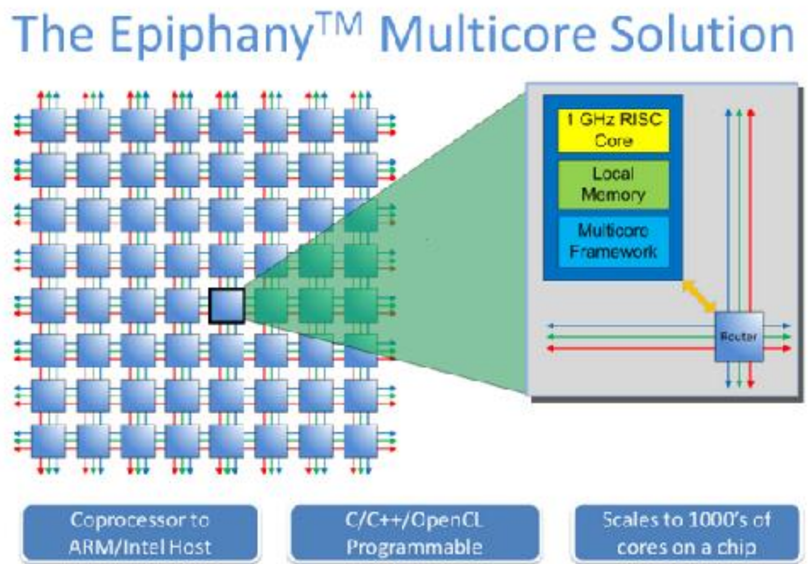
Hardware Simulator



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
>
> Father of Erlang