# Develop Your Own Operating Systems using Cheap ARM Boards

Jim Huang ( 黃敬群 ) <**jserv**@0xlab.org>

Jan 8, 2014 / NCTU, Taiwan

# Rights to copy



creative commons
COMMONS DEED

Corrections, suggestions, contributions and translations are welcome!

Latest update: Jan 14, 2014



# Goals of This Presentation

- Know the reasons why various operating systems exist and how they are functioned for dedicated purposes
- Understand the basic concepts while building system software from scratch
- How can we benefit from cheap ARM boards and the related open source tools?
  – Raspberry Pi & STM32F4-Discovery

# Agenda

- Yet another Operating System?
- Embedded Operating System Designs
- Cheap ARM boards
- Development flow using open source technologies

# Yet Another Operating Systems?

# Reasons of Home-brew OS

- Escape from large-scale and uncertain code base
  - Linux is a good example: Mature but hard to fit everyone's need
- Exploit new systematic methods
  - Hypervisor for virtualization, runtime isolation
  - Heterogeneous computing for both performance and power efficiency
- Customization
  - Deeply embedded environments, security, education (xinu, minix, xv6), domain-specific language/runtime, mobile (OKL4; 1.5 billion shipment!)
- "Just for Fun" – Linus Torvalds

# Techniques inspired by OS

- Even Web/Application Framework learn the performance techniques from OS concepts.
- Virtualization
  - Hypervisor, Resource Kernel (KVM), …
  - Intel VT-d , ARM Cortex-A15/A7, ARMv8
- Security
  - Dynamic tracing, analysis, and instrumentation using VM

# Statistics about Large-scale Systems

- Drivers cause 85% of Windows XP crashes.
  - Michael M. Swift, Brian N. Bershad, Henry M. Levy: *"Improving the Reliability of Commodity Operating Systems"*, SOSP 2003
- Error rate in Linux drivers is 3x (maximum: 10x) higher than for the rest of the kernel
  - Life expectancy of a bug in the Linux kernel (~2.4): 1.8 years
  - Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, Dawson R. Engler: *"An Empirical Study of Operating System Errors"*, SOSP 2001

# Some statistics

- Causes for driver bugs
  - 23% programming error
  - 38% mismatch regarding device specification
  - 39% OS-driver-interface misconceptions
  - Leonid Ryzhyk, Peter Chubb, Ihor Kuz and Gernot Heiser: *“Dingo: Taming device drivers”*, EuroSys 2009

# Anecdote: Linux e1000 NVRAM bug

- **[Aug 8, 2008]** Bug report: e1000 PCI-X network cards rendered broken by Linux 2.6.27-rc
  - overwritten NVRAM on card
- **[Oct 1, 2008]** Intel releases quickfix
  - map NVRAM somewhere else
- **[Oct 15, 2008]** Reason found:
  - dynamic ftrace framework tries to patch __init code, but .init sections are unmapped after running init code
  - NVRAM got mapped to same location
  - scary cmpxchg() behavior on I/O memory
- **[Nov 2, 2008]** dynamic ftrace reworked for Linux 2.6.28-rc3

**FTrace** & **NIC driver**!
instrumentation vs. device driver

# Linux Device Driver bugs

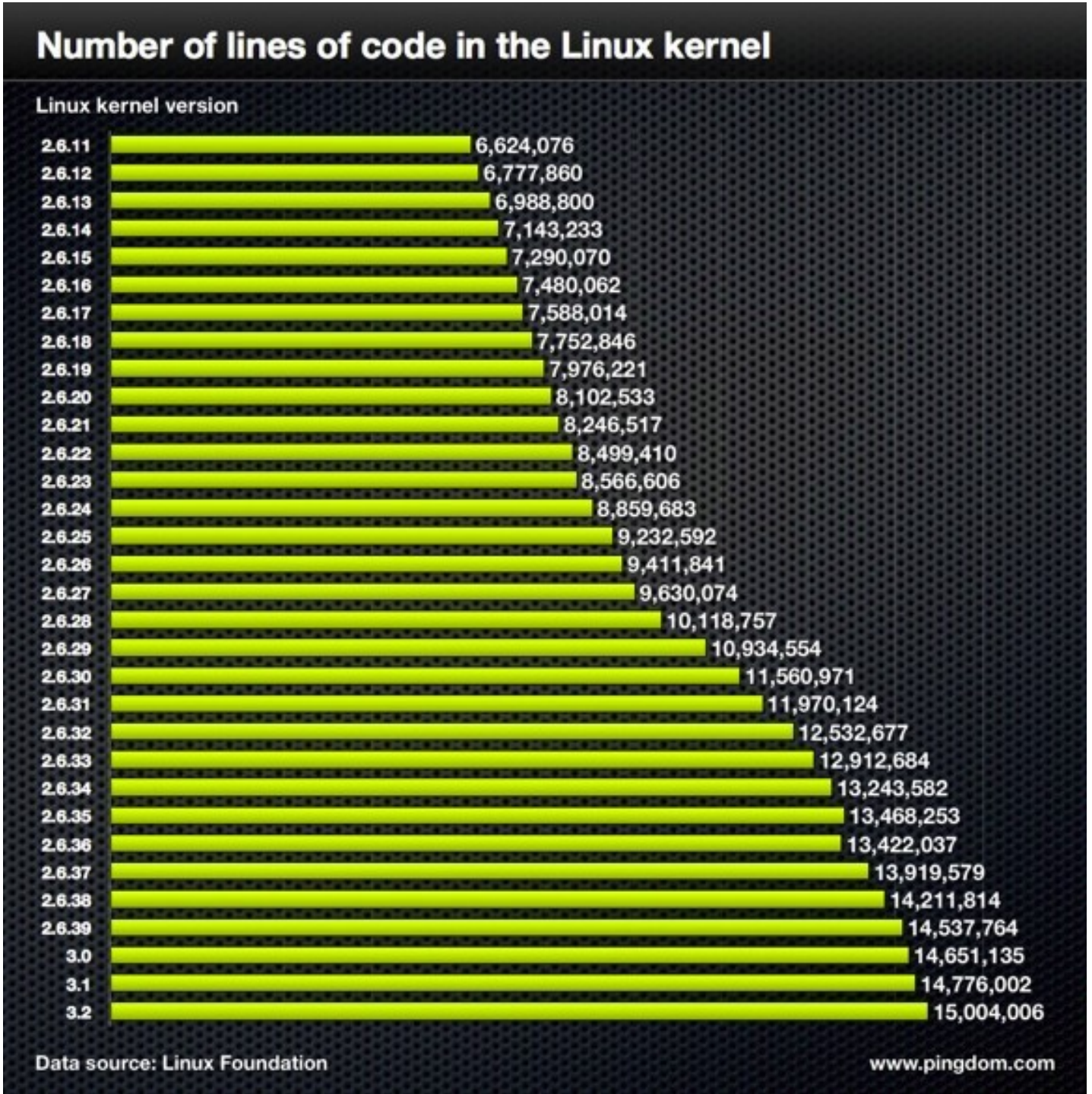
[Dingo: Taming device drivers, 2009]

| Driver | #loc | #bugs |
|---|---|---|
| **USB** | | |
| RTL8150 USB-to-Ethernet adapter | 827 | 16 |
| EL1210a USB-to-Ethernet adapter | 710 | 2 |
| KL5kusb101 USB-to-Ethernet apapter | 925 | 15 |
| Generic USB network driver | 1028 | 45 |
| USB hub | 2234 | 67 |
| USB-to-serial converter | 989 | 50 |
| USB mass storage | 803 | 23 |
| **Firewire** | | |
| IEEE1394 Ethernet controller | 1413 | 22 |
| SBP-2 transport protocol | 1713 | 46 |
| **PCI** | | |
| Mellanox InfiniHost InfiniBand adapter | 11718 | 123 |
| BNX2 Ethernet adapter | 5412 | 51 |
| i810 frame buffer | 2920 | 16 |
| CMI8338 audio | 2660 | 22 |
| | | **498** |

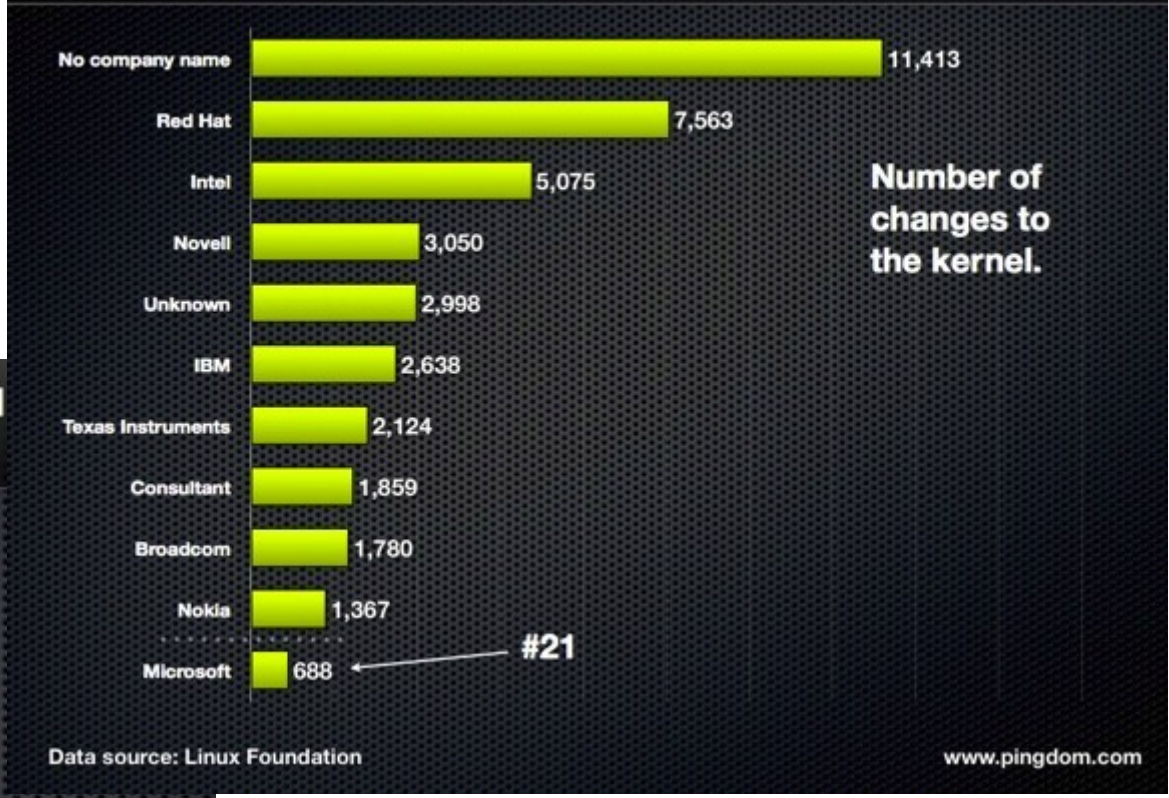
# Linux version 3.0

- consists of
  - 7702 features
  - 893 Kconfig files
  - 31281 source files
  - 88897 #ifdef blocks

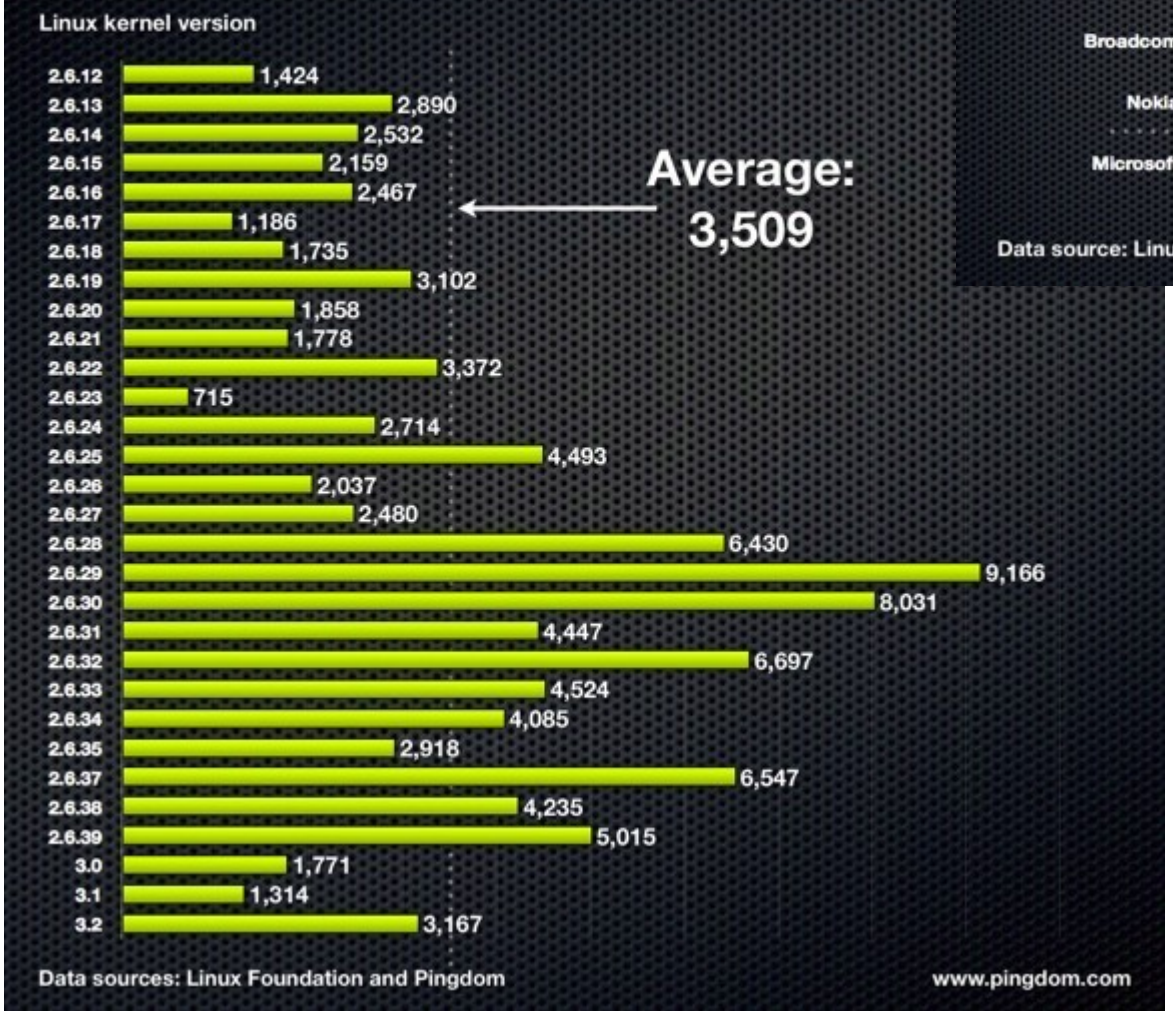
# Even worse...


Number of lines of code in the Linux kernel
Linux kernel version
2.6.11 6,624,076
2.6.12 6,777,860
2.6.13 6,988,800
2.6.14 7,143,233
2.6.15 7,290,070
2.6.16 7,480,062
2.6.17 7,588,014
2.6.18 7,752,846
2.6.19 7,976,221
2.6.20 8,102,533
2.6.21 8,246,517
2.6.22 8,499,410
2.6.23 8,566,606
2.6.24 8,859,683
2.6.25 9,232,592
2.6.26 9,411,841
2.6.27 9,630,074
2.6.28 10,118,757
2.6.29 10,934,554
2.6.30 11,560,971
2.6.31 11,970,124
2.6.32 12,532,677
2.6.33 12,912,684
2.6.34 13,243,582
2.6.35 13,468,253
2.6.36 13,422,037
2.6.37 13,919,579
2.6.38 14,211,814
2.6.39 14,537,764
3.0 14,651,135
3.1 14,776,002
3.2 15,004,006
Data source: Linux Foundation
www.pingdom.com

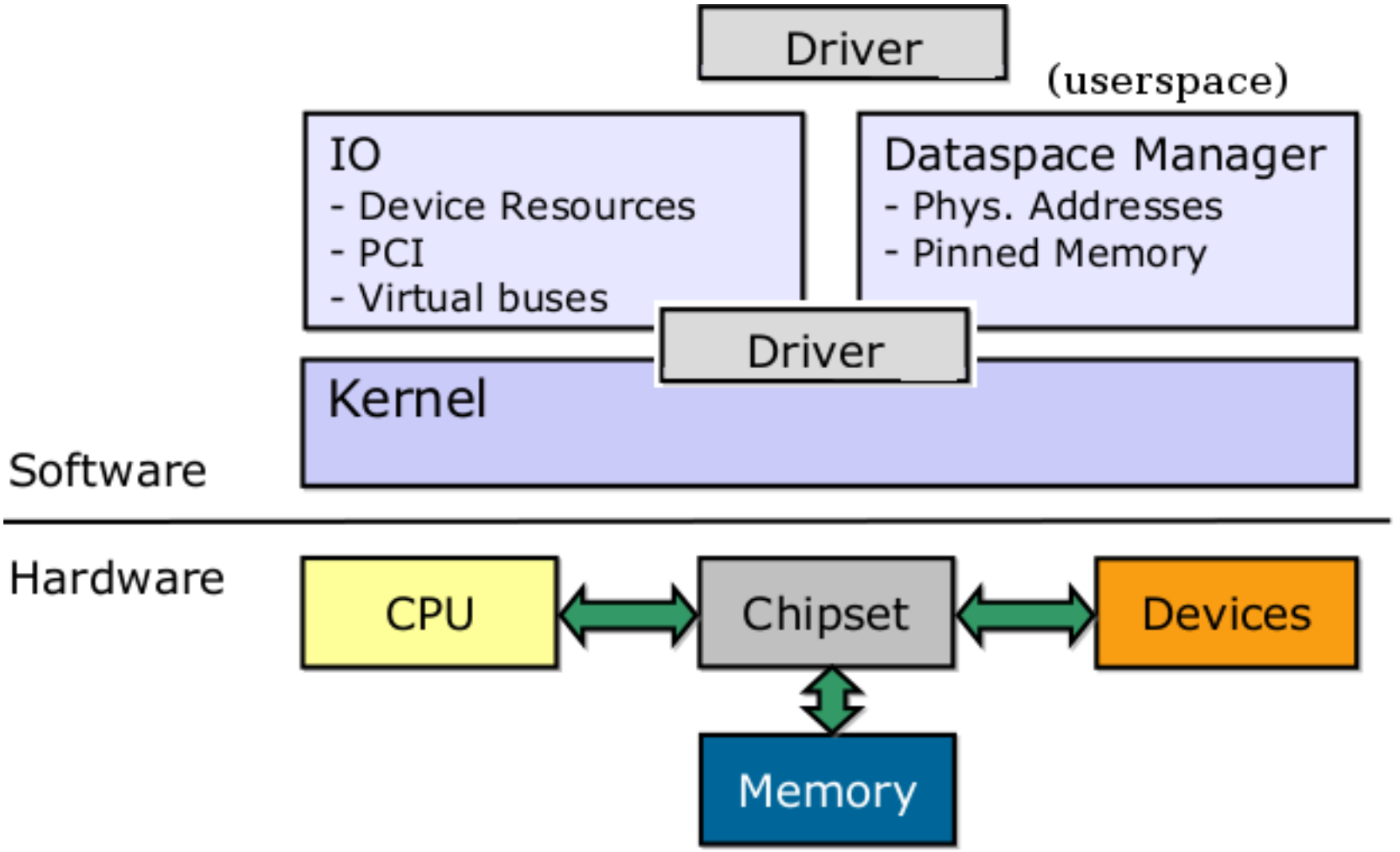
## Top 10 contributors to the Linux kernel since version 2.6.36



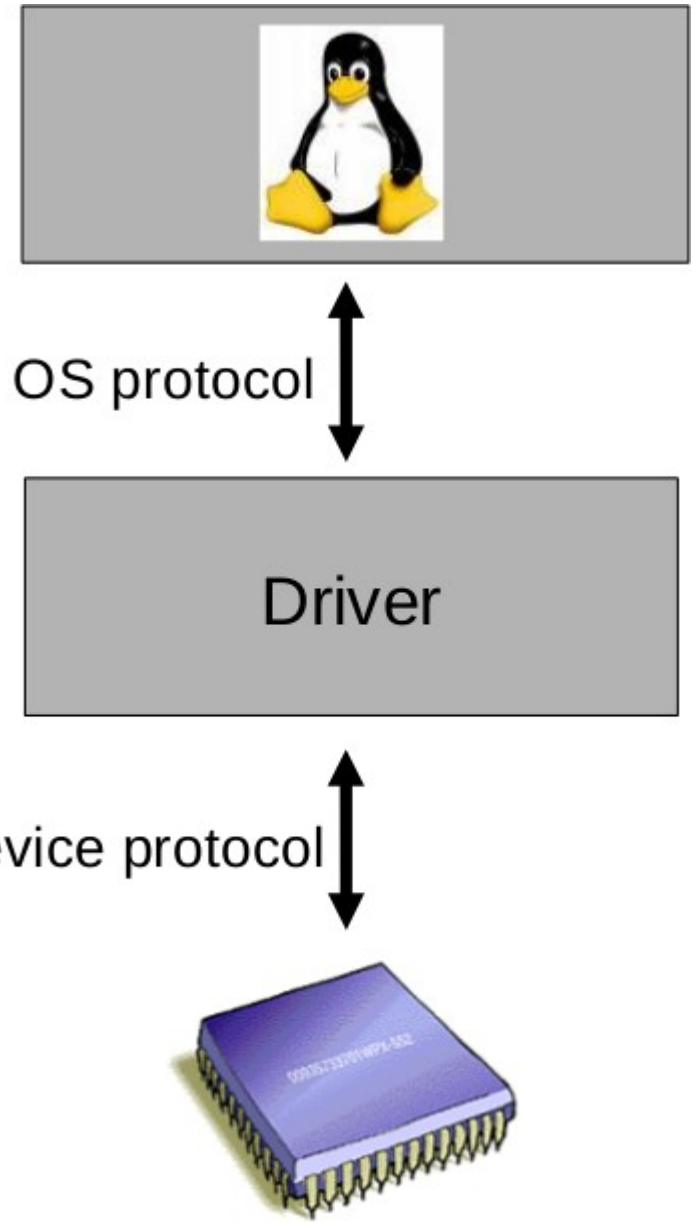

## Number of lines of code added to the Linux kernel per each day of development



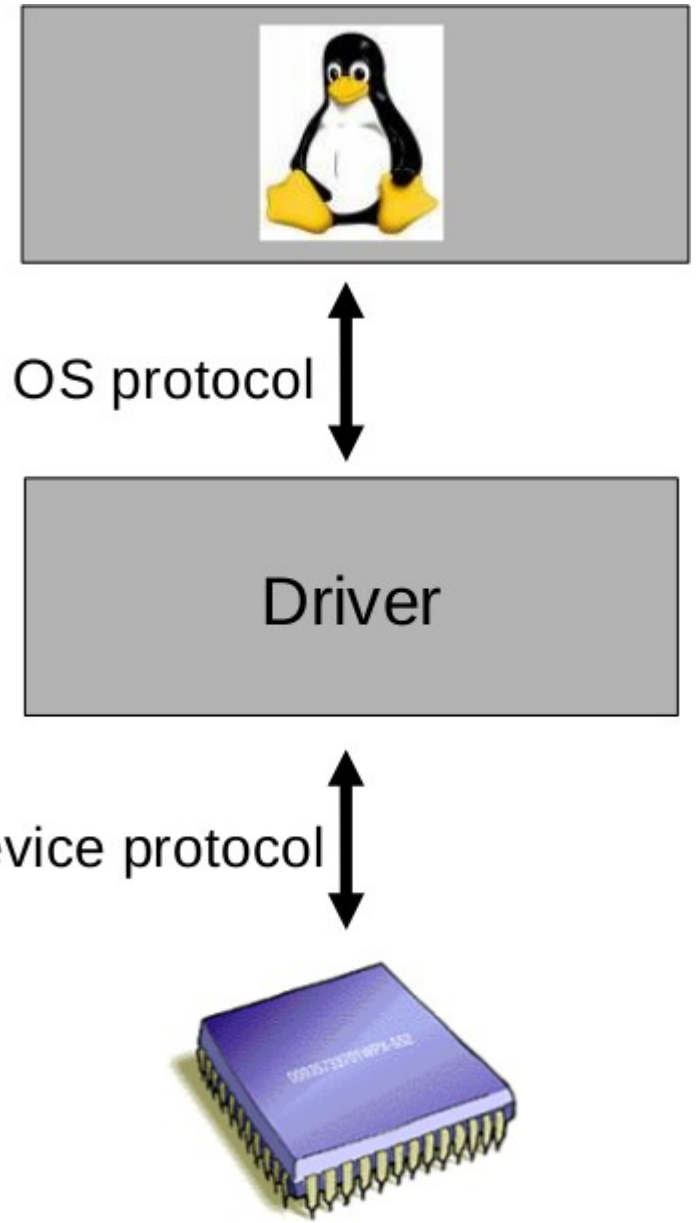

# Device Driver Model




# Bugs in Linux Device Driver



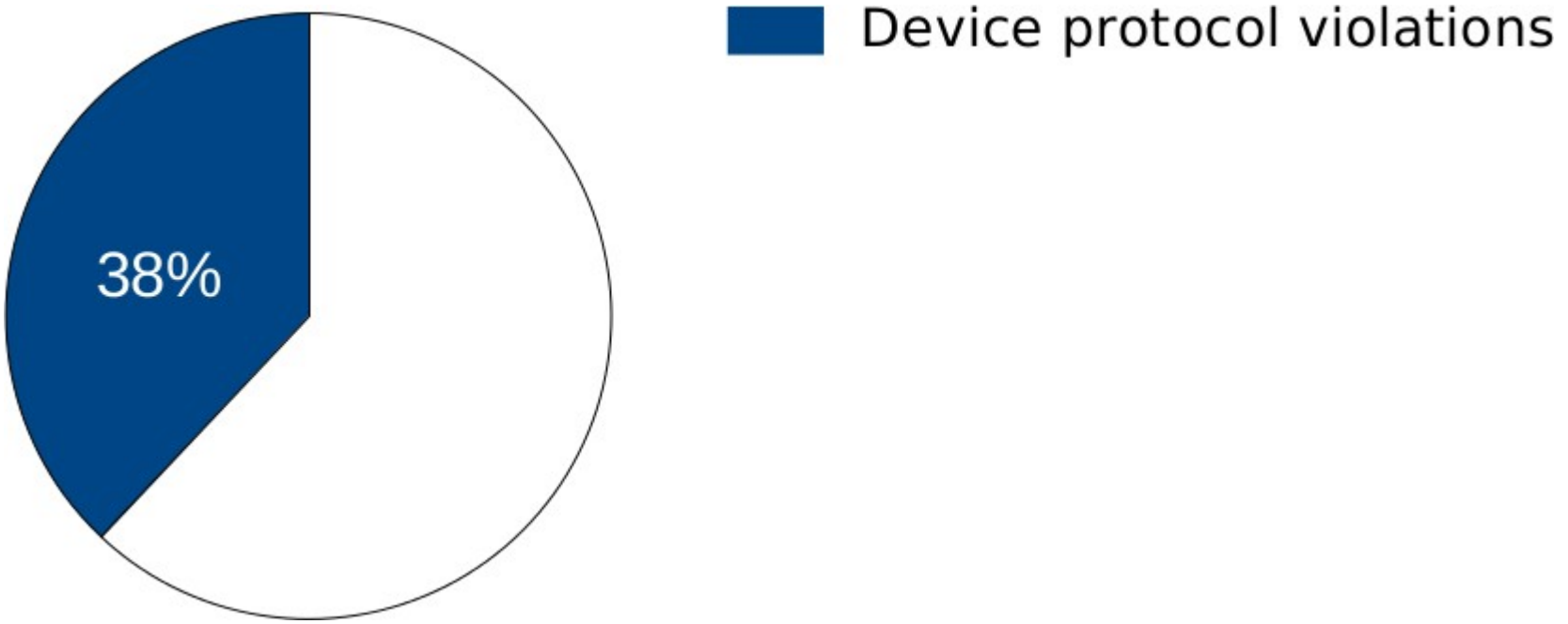

# Bugs in Linux Device Driver



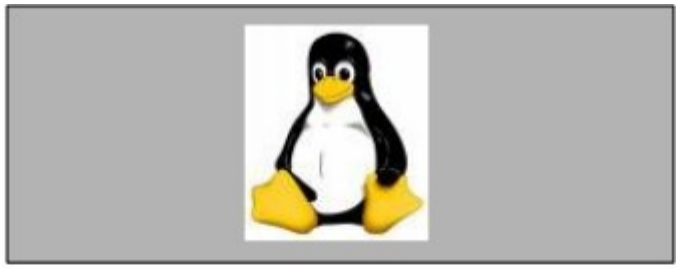

Device protocol violation examples:

- ✗Issuing a command to uninitialized device
- ✗Writing an invalid register value
- ✗Incorrectly managing DMA descriptors

# Linux Device Driver Bug Portion



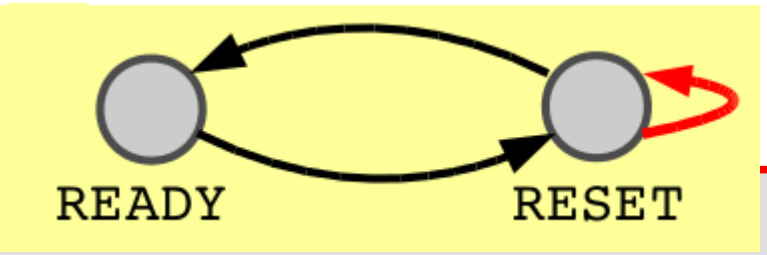

# Bugs in Linux Device Driver



OS protocol

Driver

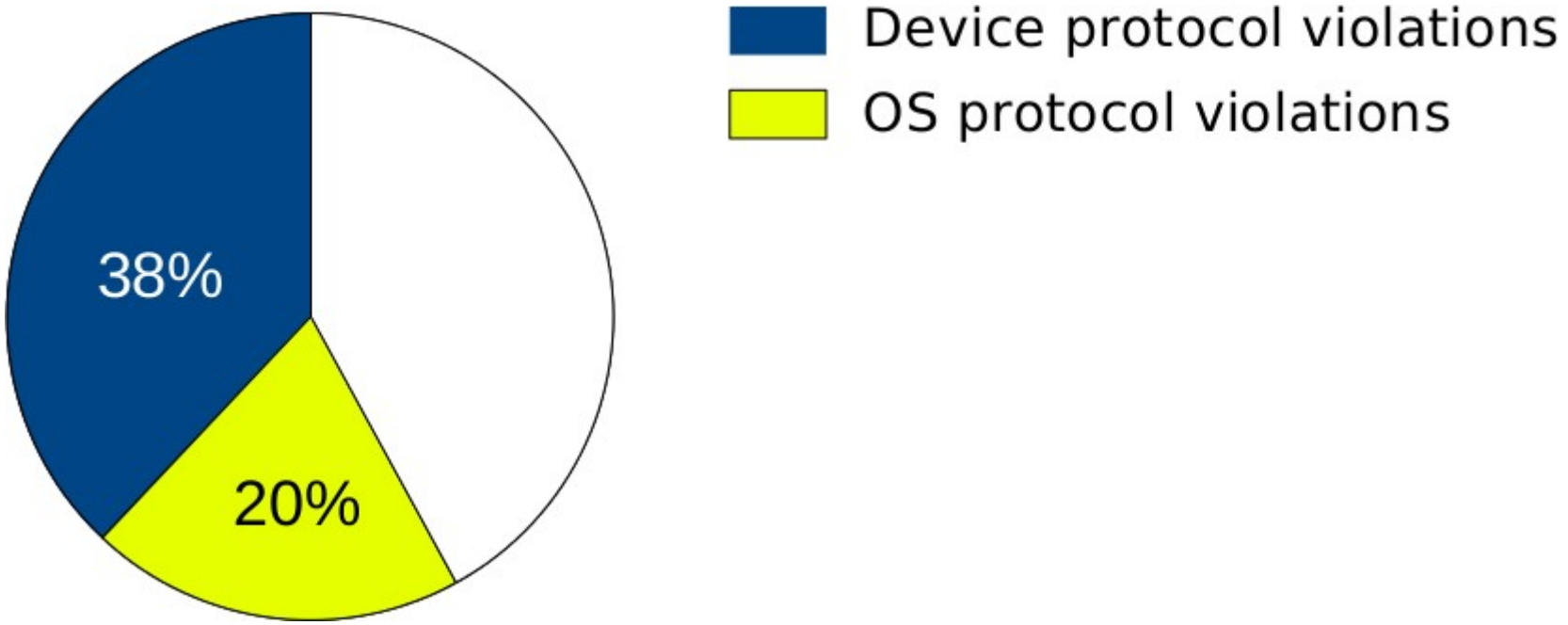
device protocol



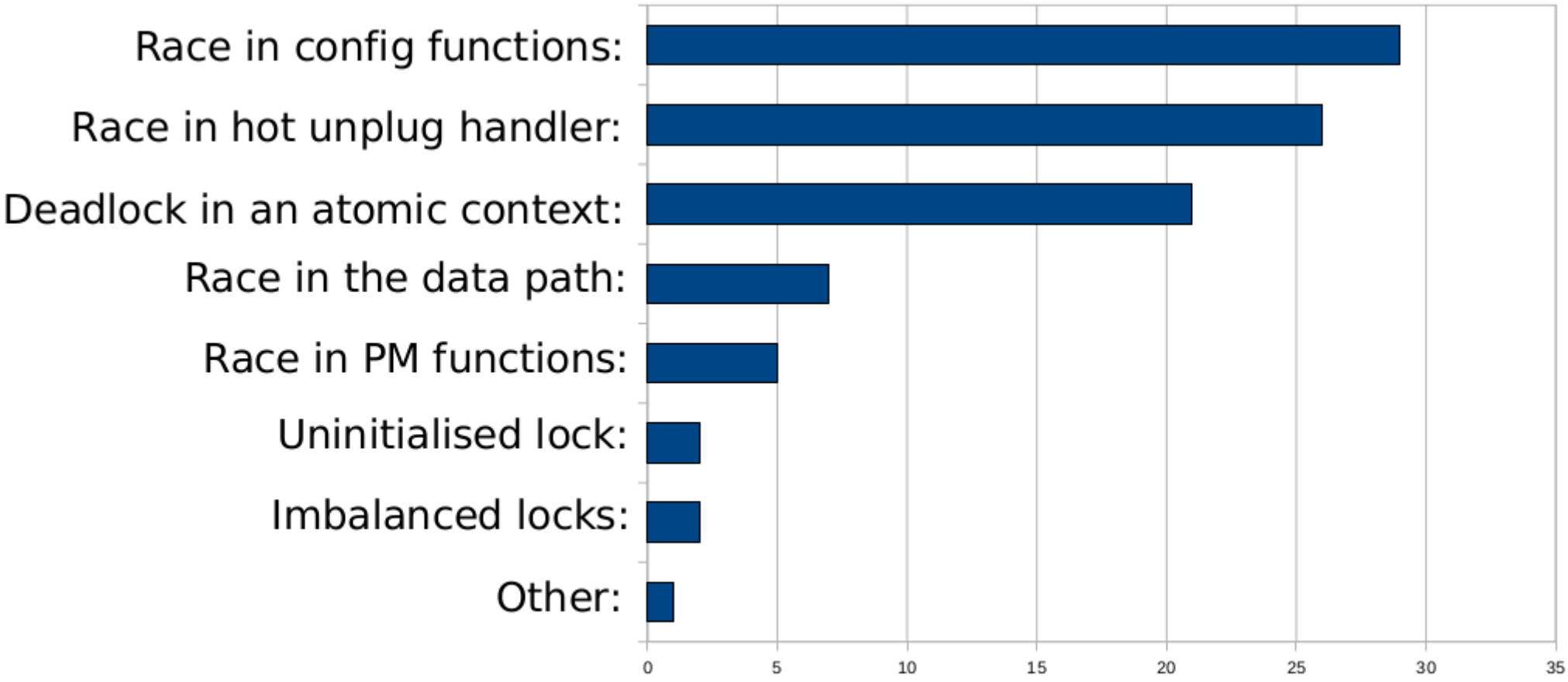

Mellanox Infinihost controller Driver

```
if(cur_state==IB_RESET &&
    new_state==IB_RESET){
   return 0;
}
```

# Linux Device Driver Bug Portion

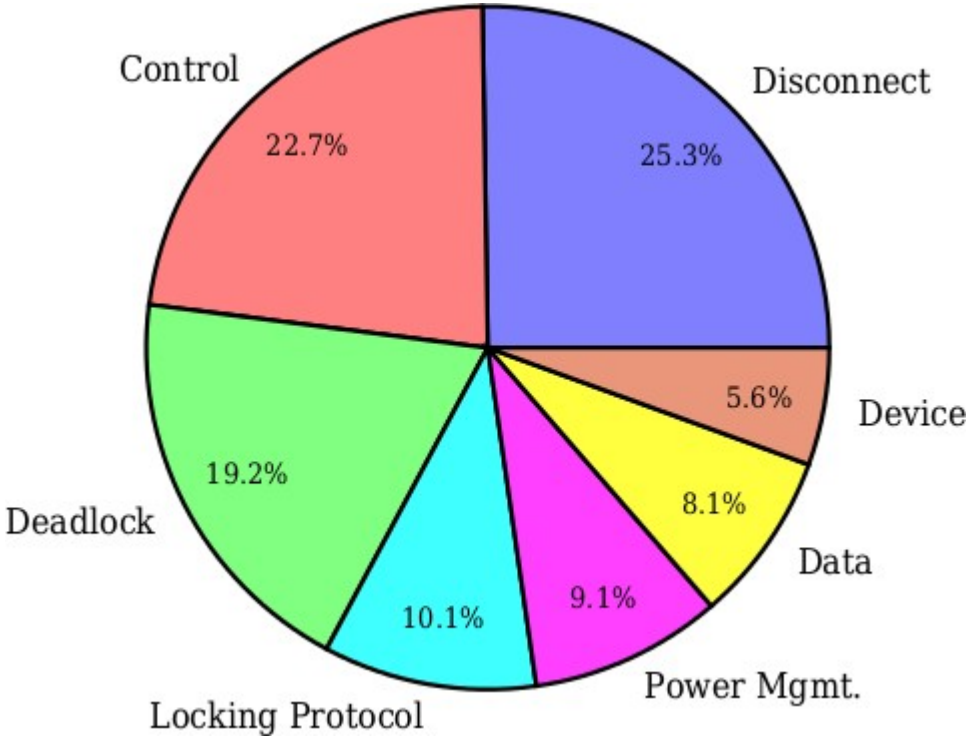



# Concurrency errors

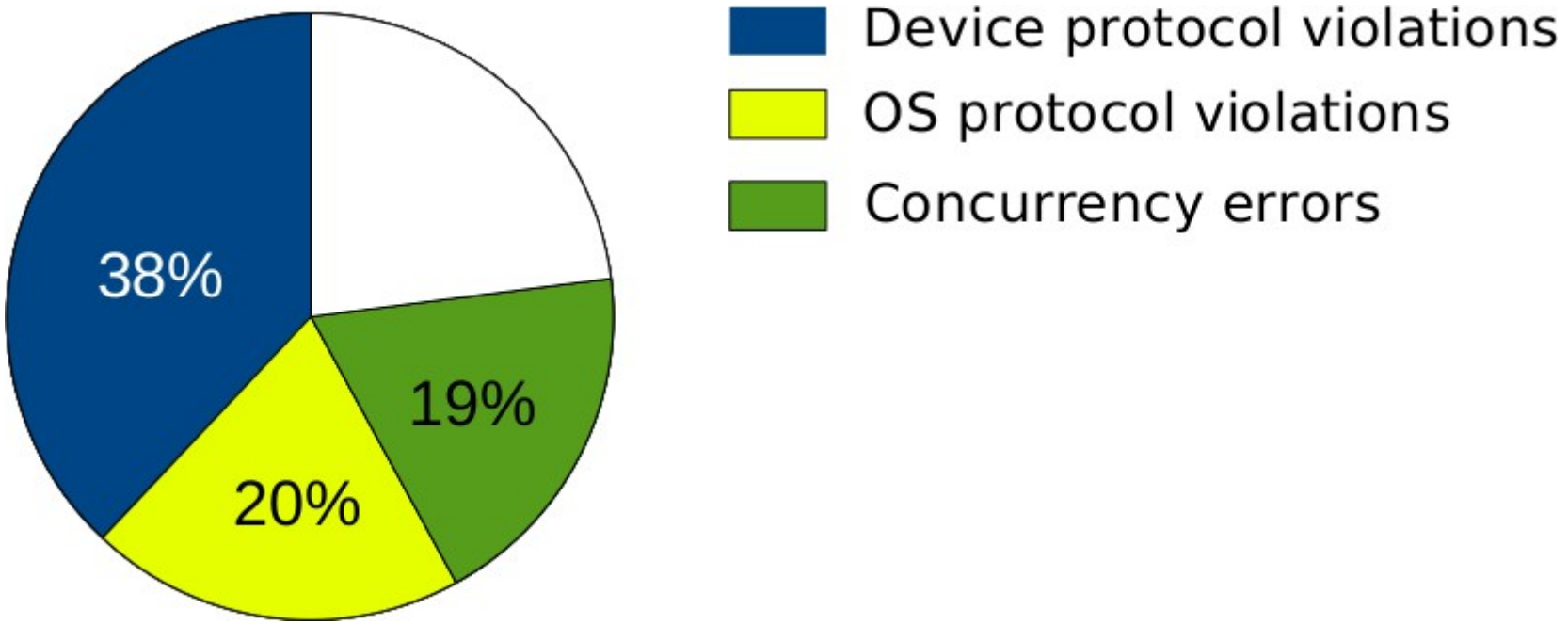



# Further study about concurrency bugs

- Markus Peloquin, Lena Olson, Andrew Coonce, University of Wisconsin–Madison, *"Simultaneity Safari: A Study of Concurrency Bugs in Device Drivers"* (2009)
- Types of Device Driver Bugs




# Linux Device Driver Bug Portion




# Embedded Operating System Designs

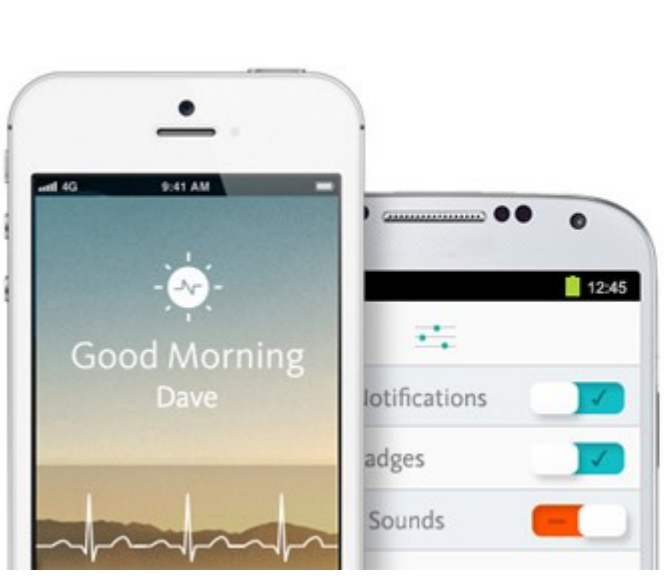
# IoT = I own Technologies, but...

- You can only use the devices, services, rights, etc. without really owning them even if you buy the “products" in higher price.
- Ecosystem is getting quite essential for applications.

# Deeply Embedded Devices

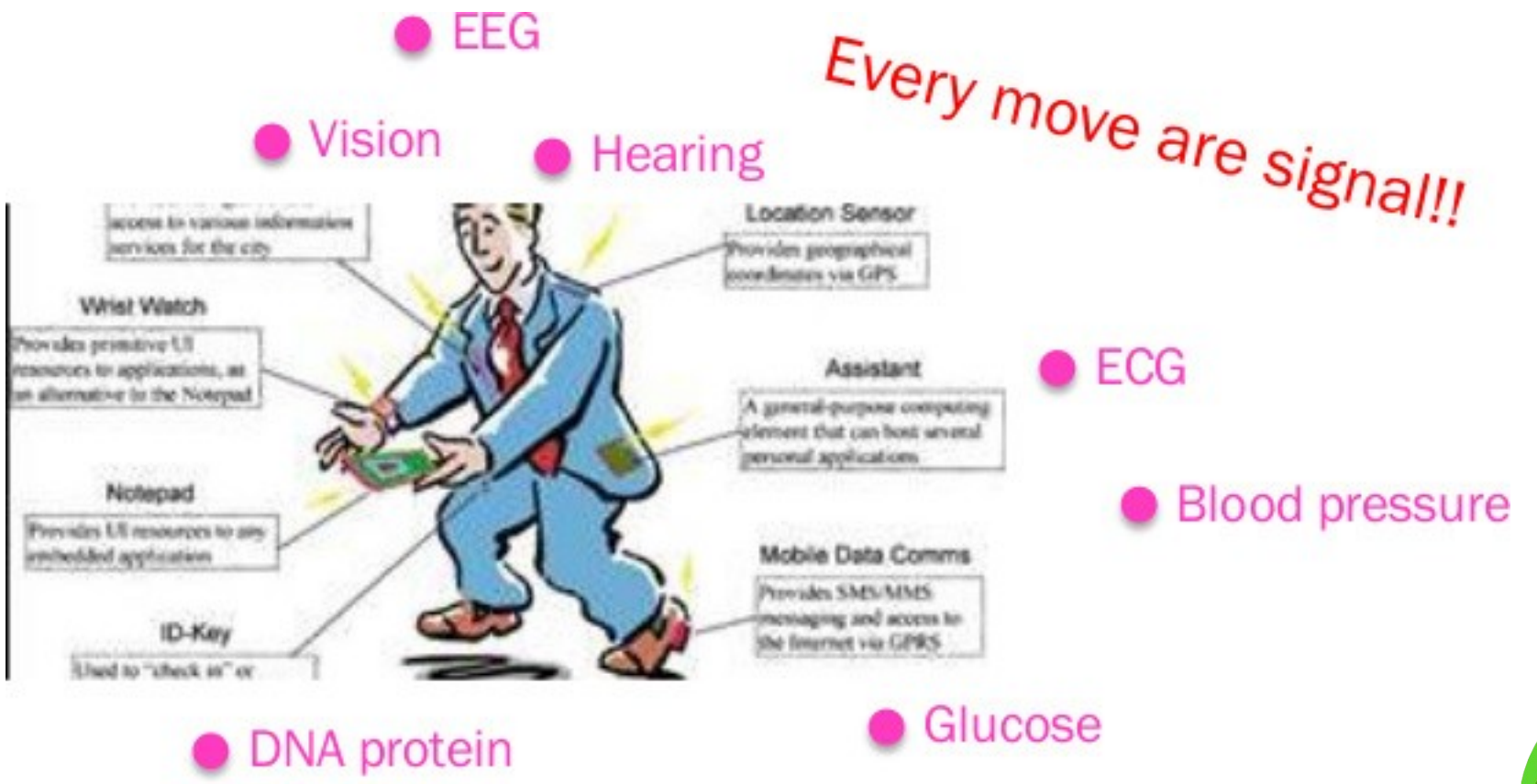
- Power awareness; solid and limited applications
- Multi-tasking or cooperative scheduling is still required
- IoT (Internet of Things) is the specialized derivative with networking facility
- Communication capability is built-in for some products
- Example: AIRO wristband (health tracker)

  `http://www.weweartech.com/amazing-new-uses-smart-watches/`

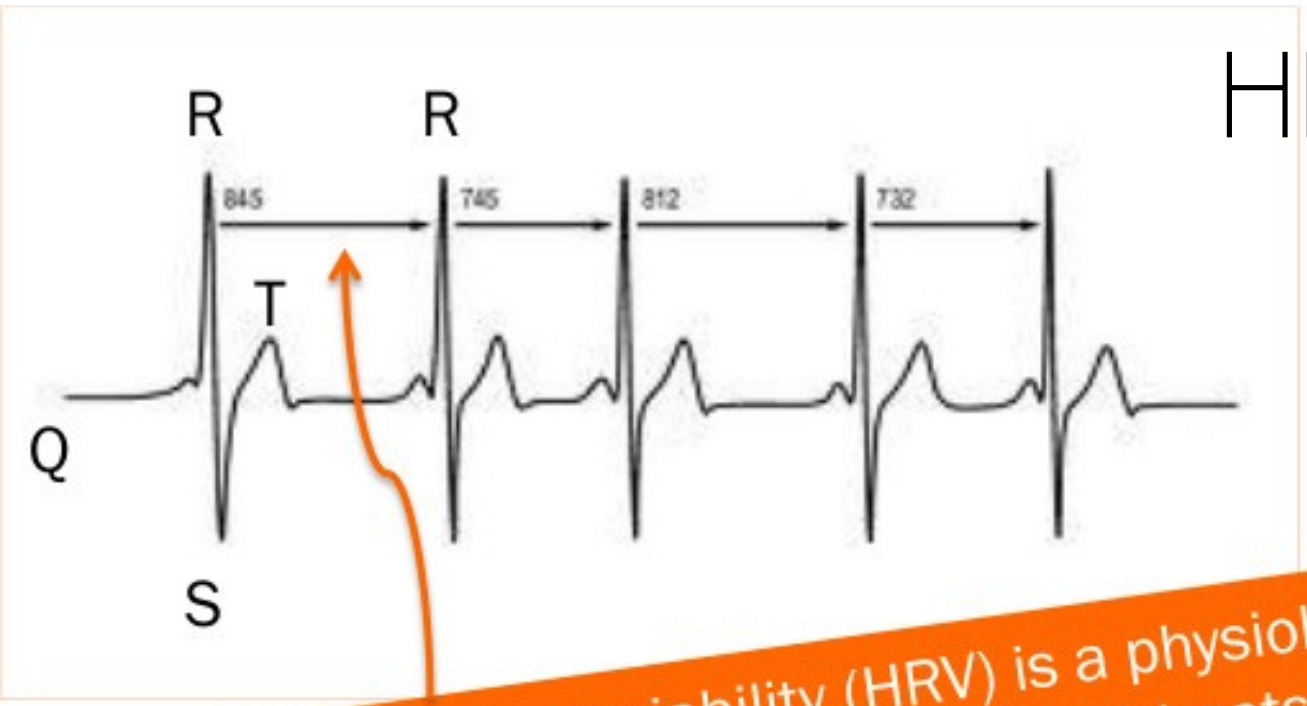



# Work at AcoMo: Physiological Inspection

- Analyze signals from various bio sensors and apply efficient algorithms to examine the healthy condition



# HRV Knows You

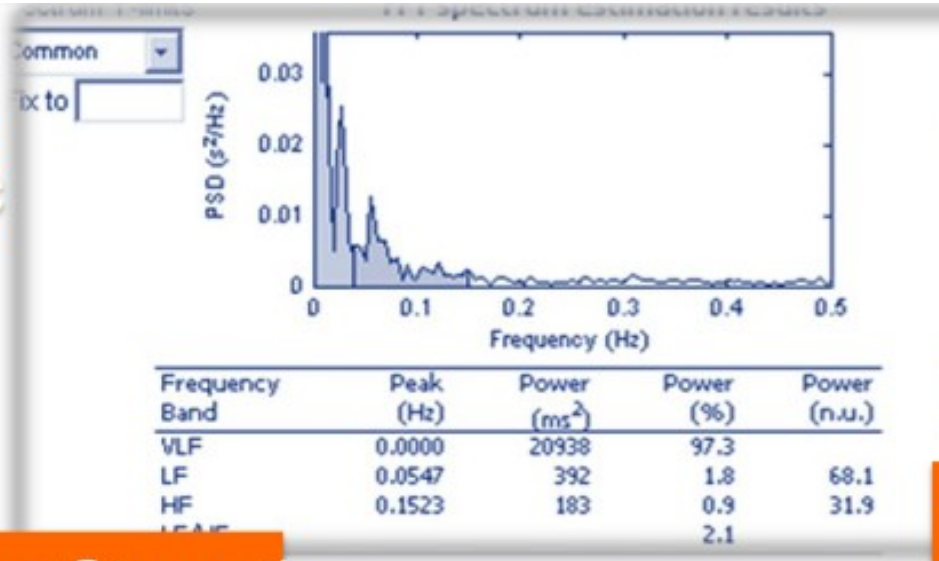



Heart rate variability (HRV) is a physiological phenomenon where the time interval between heart beats varies

Sympathetic

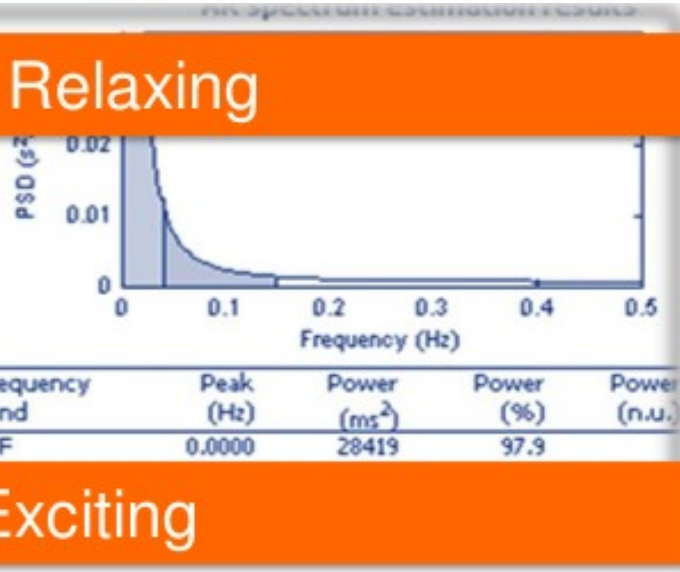
Stress

Exercise Strength




| Frequency Band | Peak (Hz) | Power ($ms^2$) | Power (%) | Power (n.u.) |
|---|---|---|---|---|
| VLF | 0.0000 | 20938 | 97.3 | |
| LF | 0.0547 | 392 | 1.8 | 68.1 |
| HF | 0.1523 | 183 | 0.9 | 31.9 |
| | | | 2.1 | |

Relaxing




| Frequency Band | Peak (Hz) | Power ($ms^2$) | Power (%) | Power (n.u.) |
|---|---|---|---|---|
| VLF | 0.0000 | 28419 | 97.9 | |

Exciting

# AcoMo built in-house OS for products and releases the basic part as an open source effort

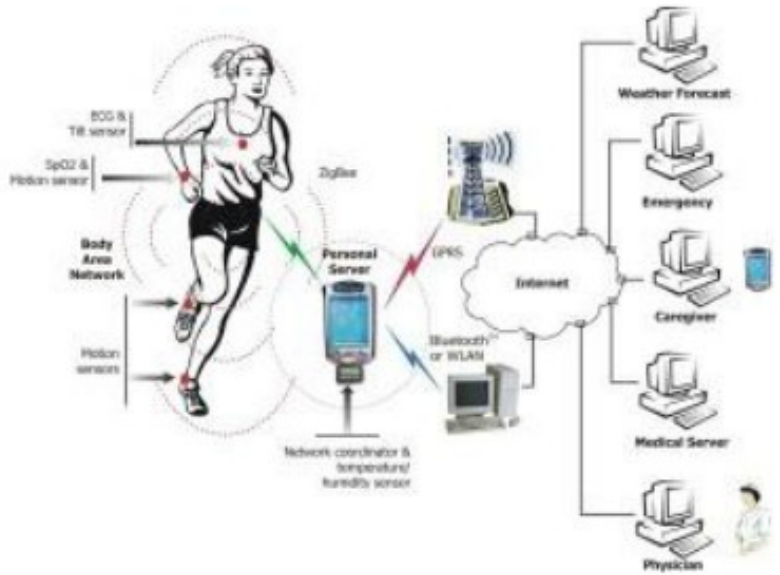
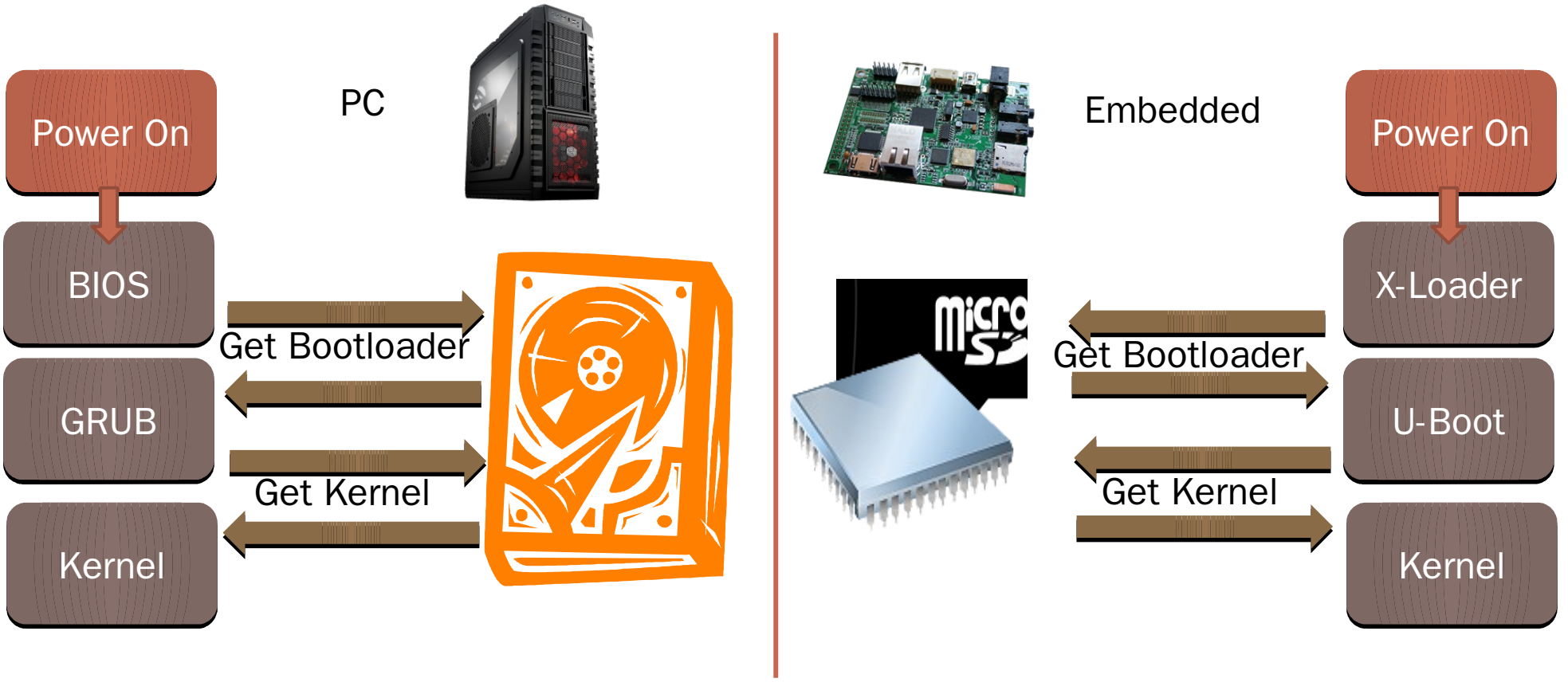



Photo: Philips

(invisible) Medical devices make sense in our life.
:: home-care :: advance warning :: security

# Booting Process

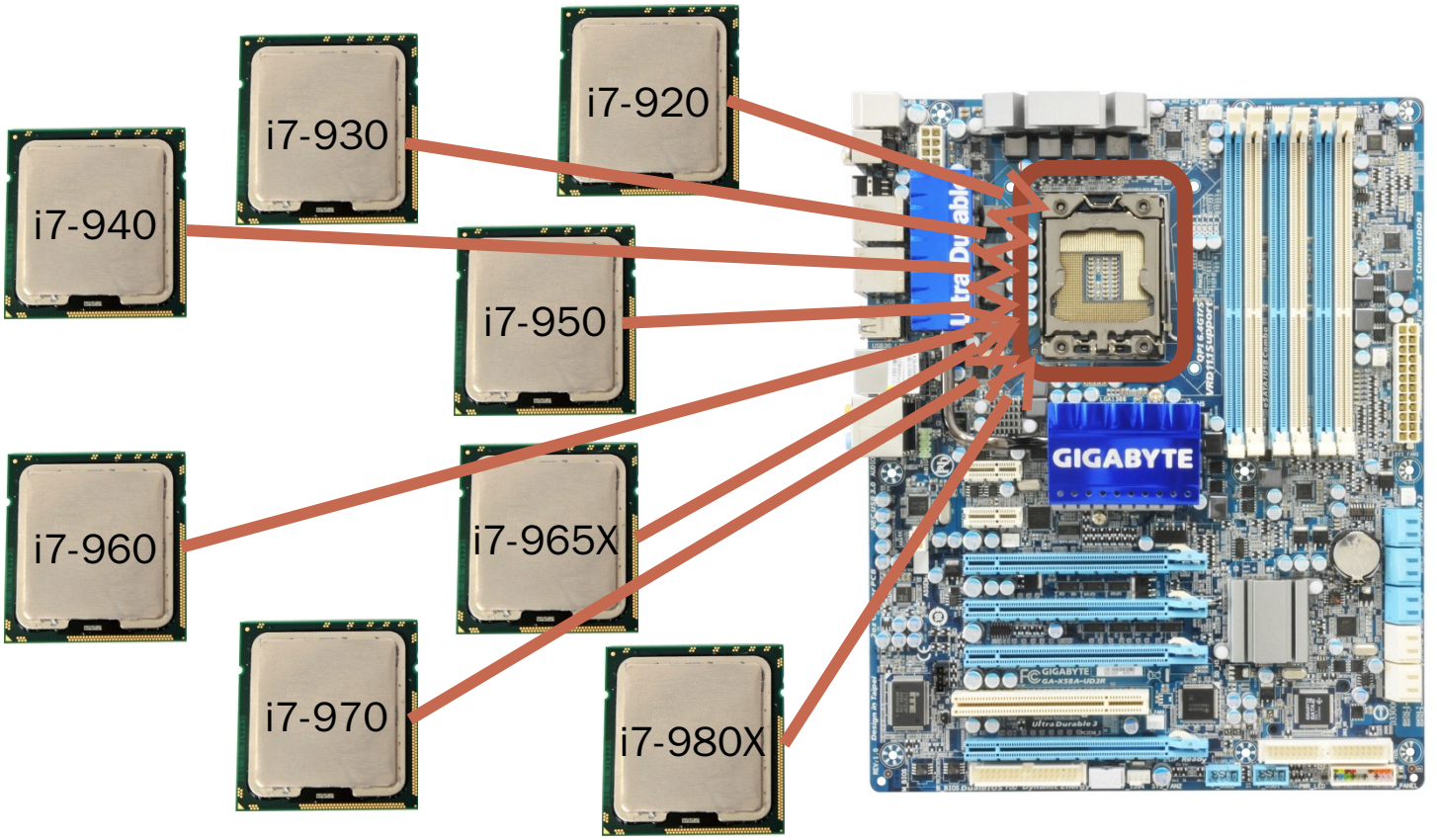



Embedded hardware: TI OMAP3730 SoC

# Embedded vs. PC

- Hardware specification is not the major difference!
  – The functionality is.
- PCs are a highly modular platform. Most components are in sockets or slots that permits easy replacement.
- Embedded Systems tend to solder their components directly to the PCB as they don't need to be replaced.

# Embedded vs. PC



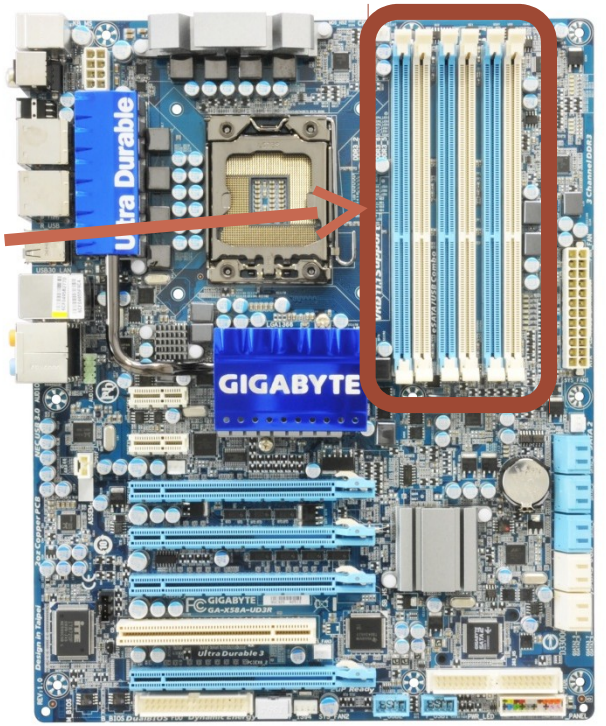

# Embedded vs. PC



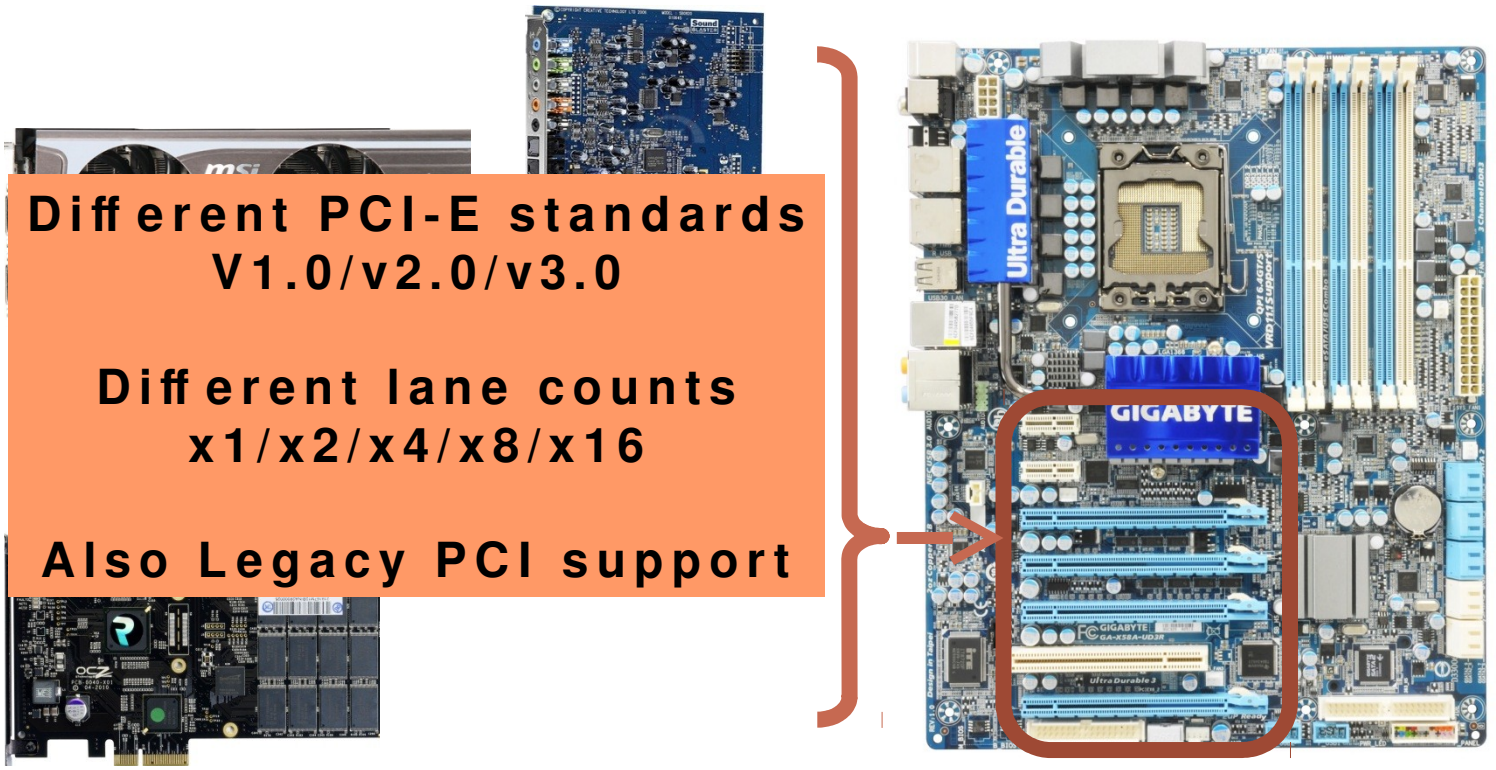
DDR3-2133
DDR3-1866
DDR3-1600
DDR3-1333
DDR3-1066
DDR3-800

**Voltage**
**1.2-1.5v**


**Legacy**
**CS-RAS-CAS-WE**




# Embedded vs. PC

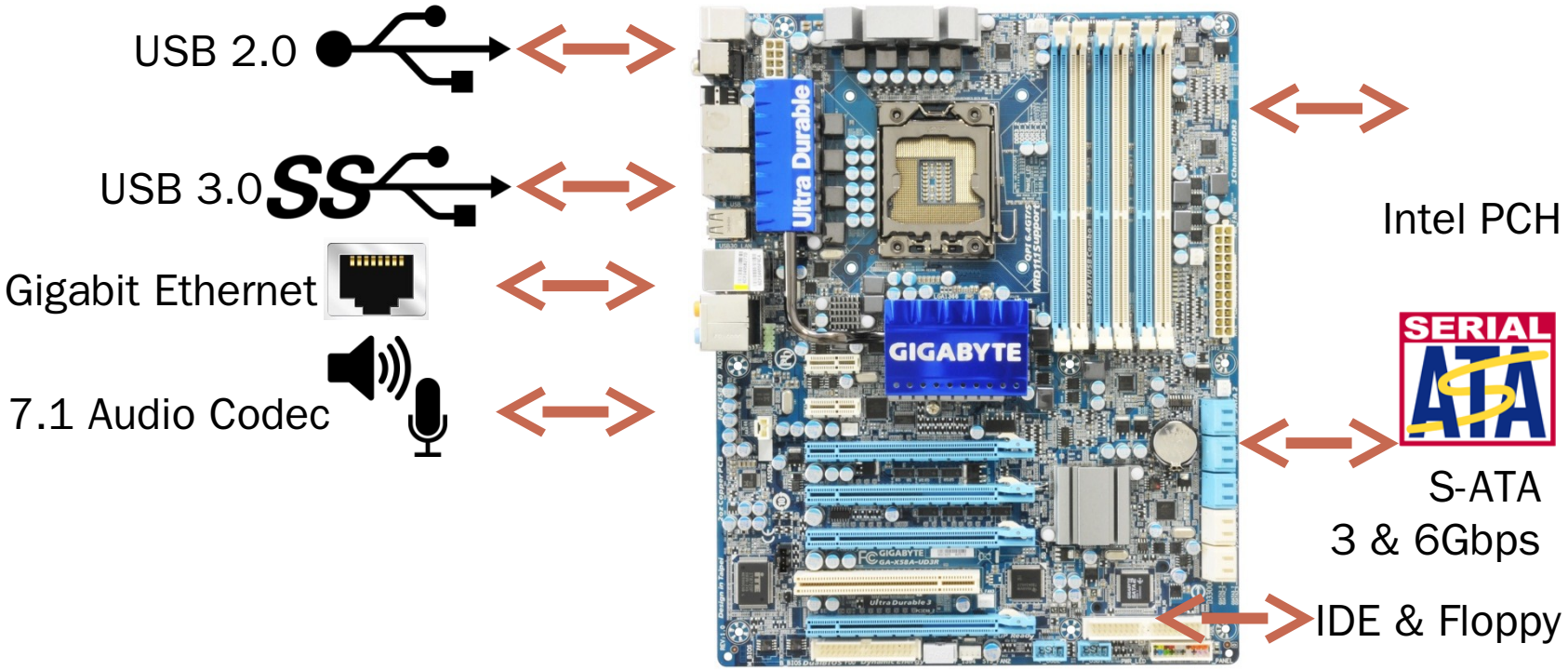



**Different PCI-E standards**
**V1.0/v2.0/v3.0**

**Different lane counts**
**x1/x2/x4/x8/x16**

**Also Legacy PCI support**

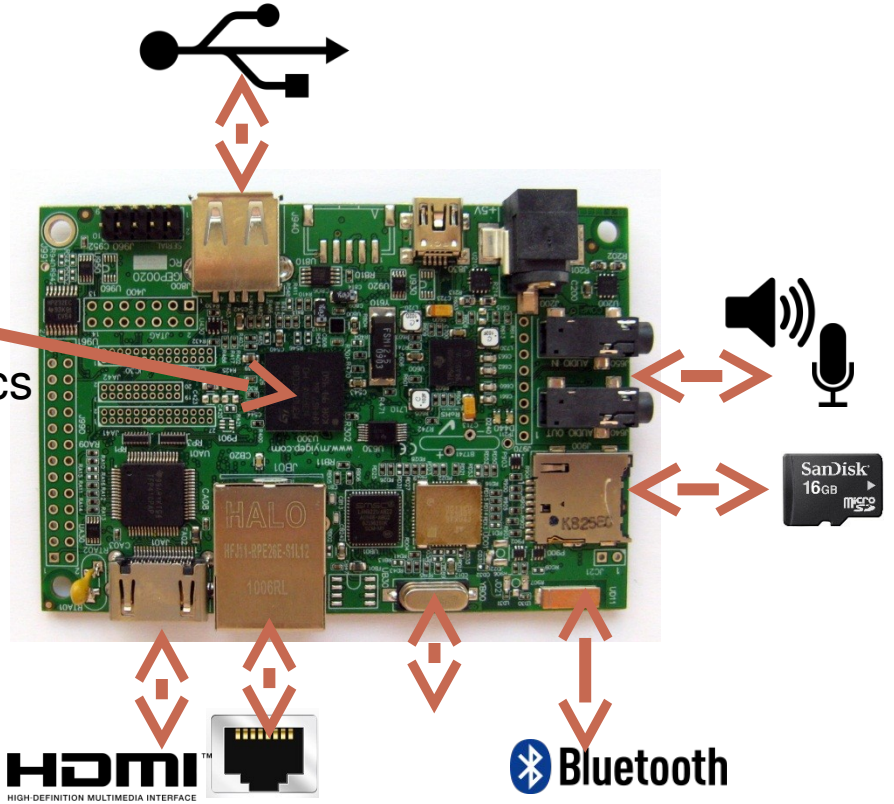
# Embedded vs. PC



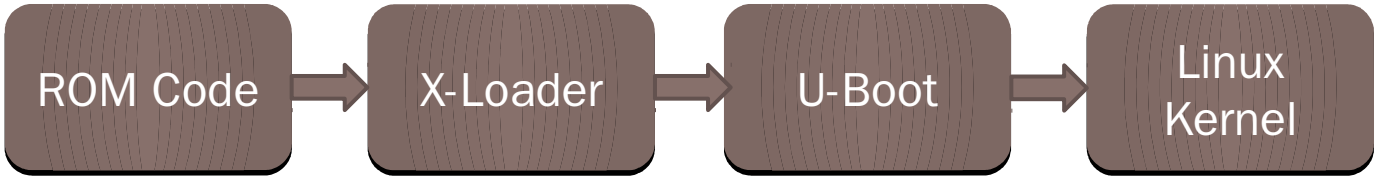

# Embedded vs. PC

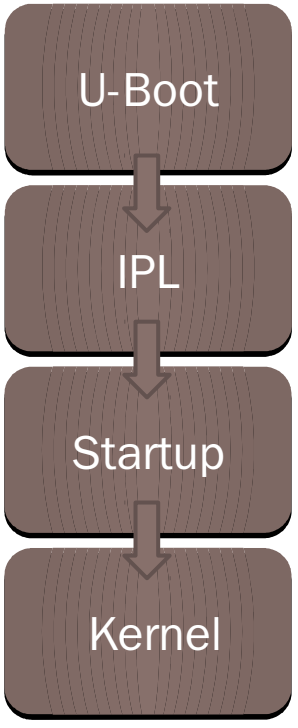
TI OMAP 3730 Processor
512MB DDR@200MHz
PowerVR SGX530 Graphics

*All soldered directly to the board. Not intended for replacements.*




# Booting Process




- On-Chip boot ROM code generally causes the CPU to perform minimal initialization of peripheral such as NAND Flash and instructs it to begin reading code from there into memory and executing it.
- This code can be a standard embedded bootloader such as U-Boot, or it can be an IPL.
- U-Boot loads the “IPL” image into memory, and begins executing it.
- The “IPL” is an “Initial Program Loader” which is responsible for initializing basic hardware and passing control to “Startup” code, and subsequently the Kernel.

Embedded hardware: TI OMAP3730 SoC

# IPL

- Begin in assembly, performs initialization for HLL
- Initialize CPU/(some) Peripheral Clocks
- Initialize basic I/O (serial)
- Minimal pin multiplexing for required peripherals (i.e. SDHC hardware)
- Read in and decompresse “IFS” image (ramdisk + kernel)
- Include basic (FAT) filesystem drivers for SDHC reading
- Passes control to “Startup”
- Can start “minidrivers” for device interaction before OS/Kernel even begins booting

# Startup

- Startup begin in C language, initialize most peripherals, and sets up important kernel structures
- Kernel expects a “syspage” structure to exist at a pre-defined location in memory. This structures provides important information about the host system.
- Enable CPU SMP operation (multiple-cores)
- Often re-do initialization done by IPL (such as serial I/O) to enable more advanced functionality
- Inform minidrivers of new environment before passing control to kernel.

# System Information

- Indicate CPU type (e.g. ARM) and vital information (e.g. number of cores), and other supported features such as NEON extensions.
- Provide access to hardware-specific function callouts made available to the system before the Kernel was running
- Provide information about the memory environment in which the kernel is running
- Information about bus devices, IRQs
- Information about connected peripherals and device trees for /**dev** population

# Cheap ARM Boards

check "Introduction to Raspberry Pi" by Computer Science Club, University of Cyprus Student Clubs

# Development flow using Open Source Technologies

# Baking Pi - Operating System Development

- Operating Systems Development! Course by Alex Chadwick. Version 1.0c (July 2013).
  `http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/`
- Divided into several basic “blocks".

# JTAG

- Initially devised for testing printed circuit boards with its boundary scanning functionality JTAG is now used extensively for debugging, programming CPLDS and initialising flash memory.
- Can be useful to recover bricked devices or write new firmware to NAND on restricted devices. But the IGEP is un-brickable
- Tools like OpenOCD and GDB ARM have successfully been used on OMAP530 devices like Beagleboard.

# Conclusion

- BYOD (Build Your Own Device) and BYOD (Build Your Own Operating System) are feasible and getting easier with open source technologies
- Domain specific applications always expect innovations from various sources including the customized kernel and userland
- System trial with very low cost

# Reference


- *“Dingo: Taming Device Drivers”*, Leonid Ryzhyk, Peter Chubb, Ihor Kuz, Gernot Heiser, UNSW/NICTA/Open Kernel Labs (2009)
- *"Hardware and Device Drivers"*, Björn Döbel, TU Dresden (2012)
- *"Configuration Coverage in the Analysis of Large-Scale System Software"*, Reinhard Tartler, Daniel Lohmann, Christian Dietrich, Christoph Egger, Julio Sincero, Friedrich-Alexander University (2011)
- *“AIRAC: A Static Analyzer for Detecting All Buffer Overrun Errors in C Programs"*, Kwangkeun Yi, Seoul National University (2005)
- *“CCured: Taming C Pointers”*, George Necula, Scott McPeak, Wes Weimer, Berkeley (2002)



http://0xlab.org