Laboratory of Nomadic Communication

Francesco Gringoli

University of Trento

Course Overview

- Introduction to Linux Networking Stack

A glimpse into the Linux Wireless Core: From kernel to firmware

Francesco Gringoli

Laboratory of Nomadic Communication

University of Trento

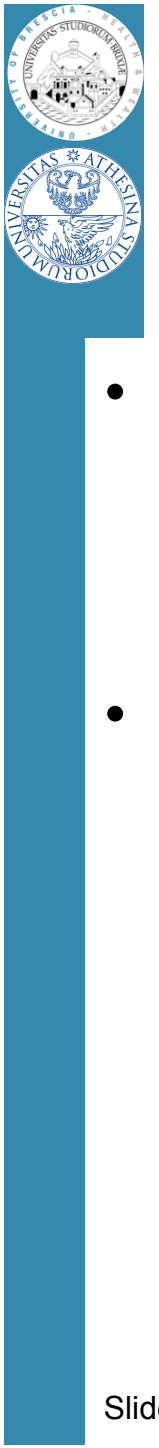
Outline

- Linux Kernel Network Code
  - Modular architecture: follows layering
- Descent to (hell?) layer 2 and below
  - Why hacking layer 2
  - OpenFirmWare for WiFi networks
- OpenFWWF: RX & TX data paths
  - Hands on: examples
- OpenFWWF exploitations

Linux Kernel Network Code

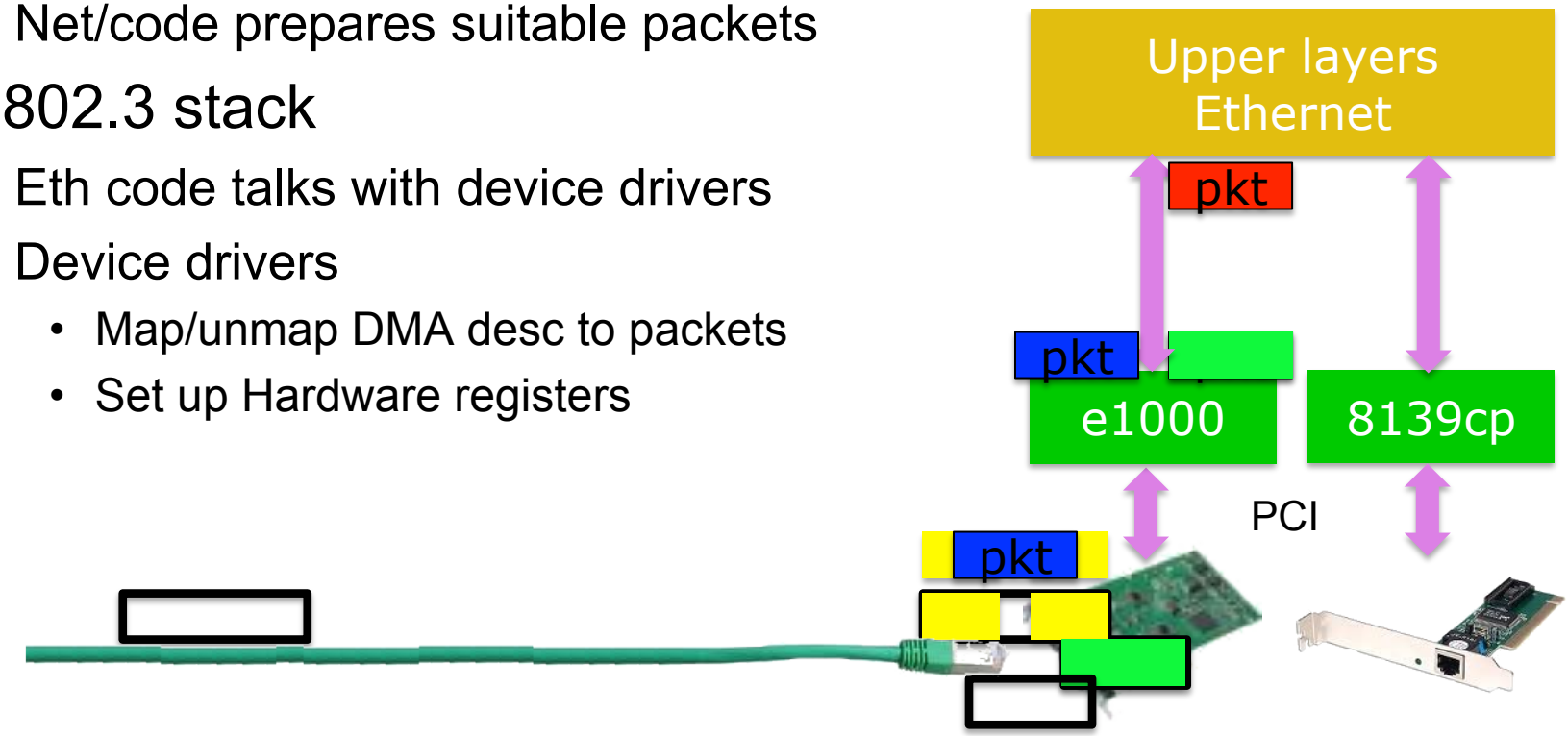
A glimpse into the Linux Kernel Wireless Code

Part 1



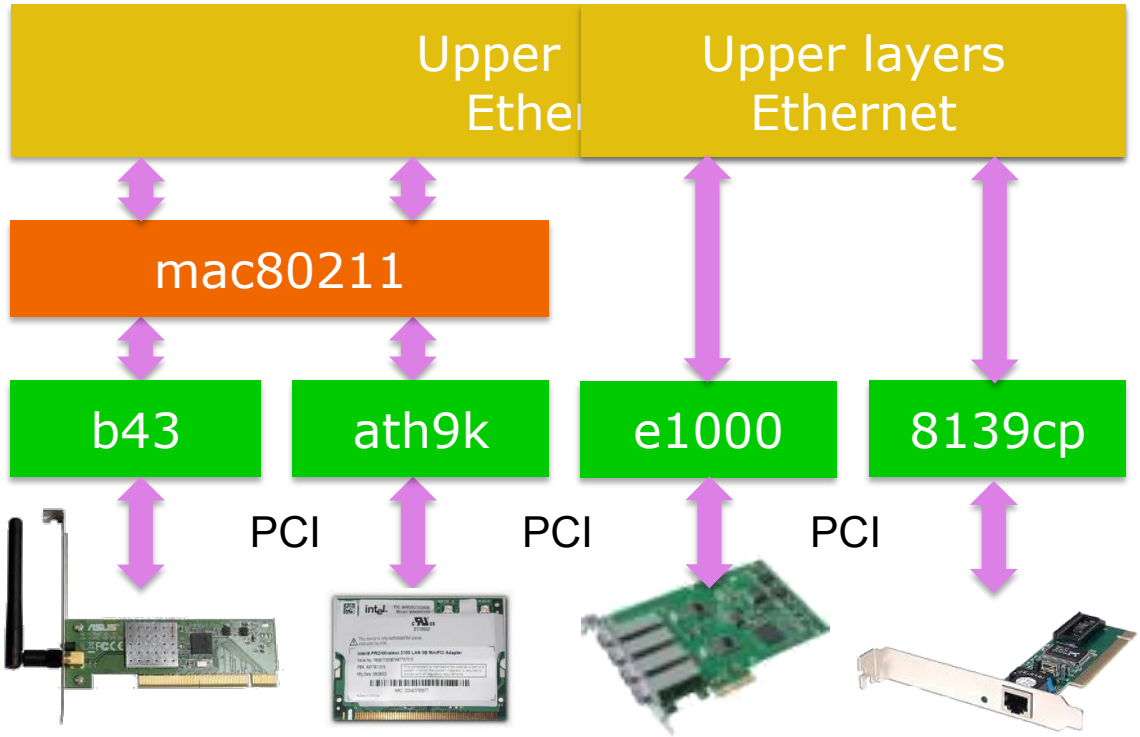
Linux Networking Stack Modular architecture

- Layers down to MAC (included)
  - All operations above/including layer 2 done by kernel code
  - Network code device agnostic
  - Net/code prepares suitable packets
- In 802.3 stack
  - Eth code talks with device drivers
  - Device drivers
    - Map/unmap DMA desc to packets
    - Set up Hardware registers



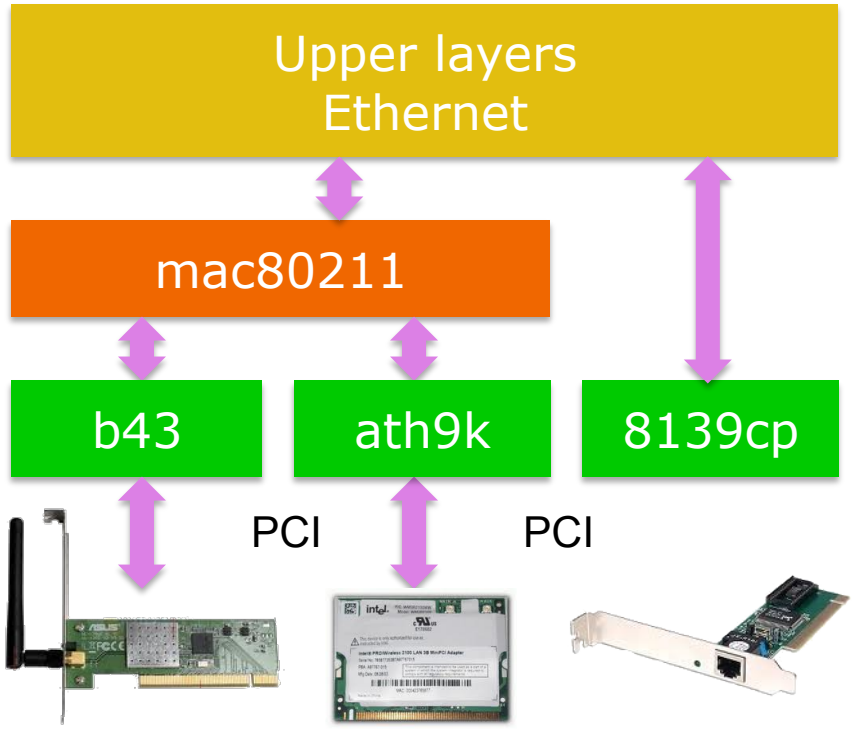
Linux Networking Stack Modular architecture

- What happens with 802.11?
  - New drivers to handle WiFi HW: how to link to net code?
  - A wrapper “mac80211” module is added



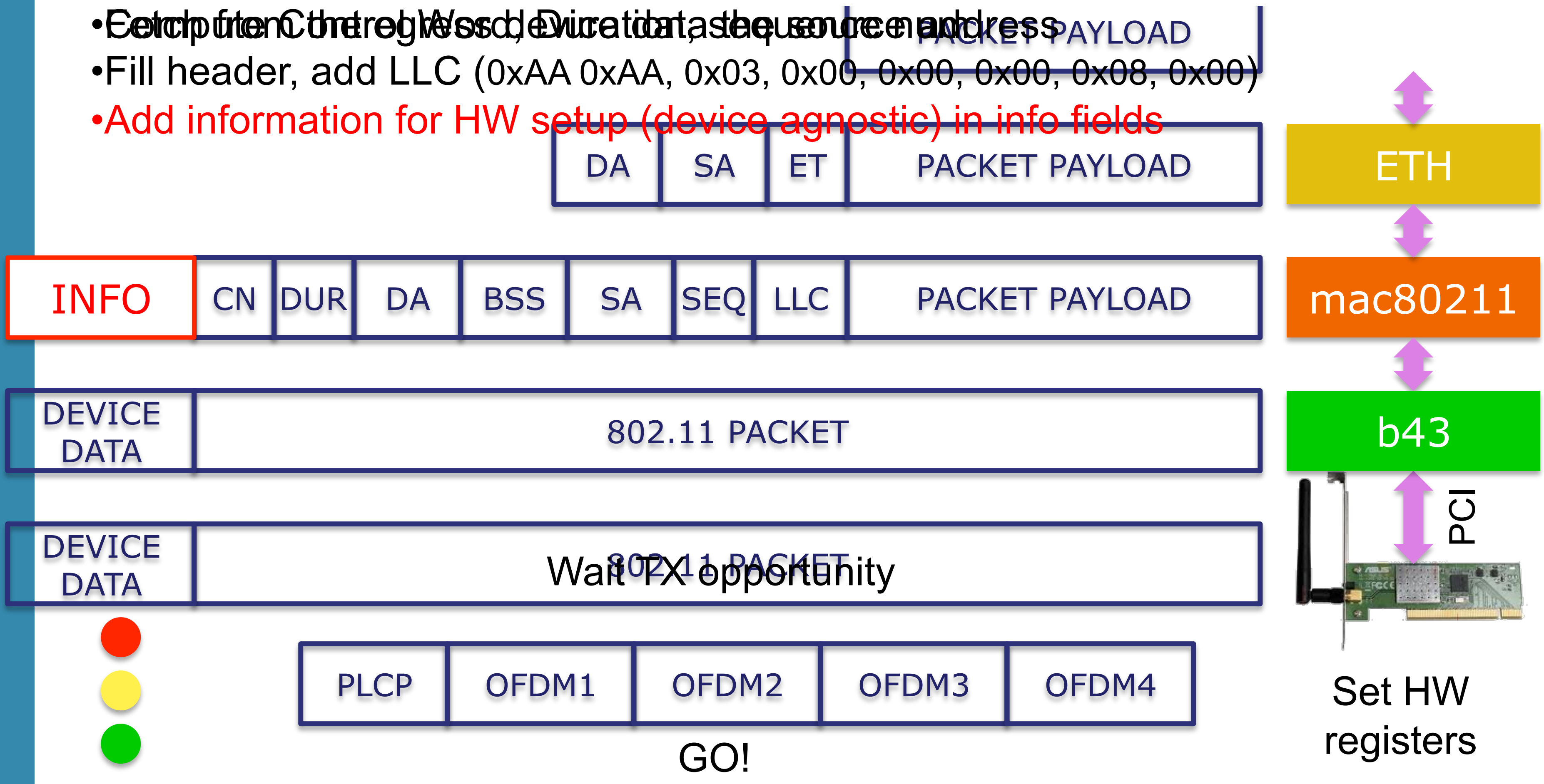
Linux & 802.11 Modular architecture

- Layers down to LLC (~mac) common with 802.3
  - All operations above/including layer 2 done by ETH/UP code
- Packets converted to 802.11 format for rx/tx
  - By wrapper “mac80211”
    - Manage packet conversion
    - Handle AAA operations
- Drivers: packets to devices
  - One dev type/one driver
    - Add data to “drive” the device



Linux & 802.11
Modular architecture/1

- Convert agnostic info into device dependent data
- Fetch from the registered device data the source address
- Compute Control Word, Duration, sequence number
- Fill header, add LLC (0xAA 0xAA, 0x03, 0x00, 0x00, 0x00, 0x08, 0x00)
- Add information for HW setup (device agnostic) in info fields

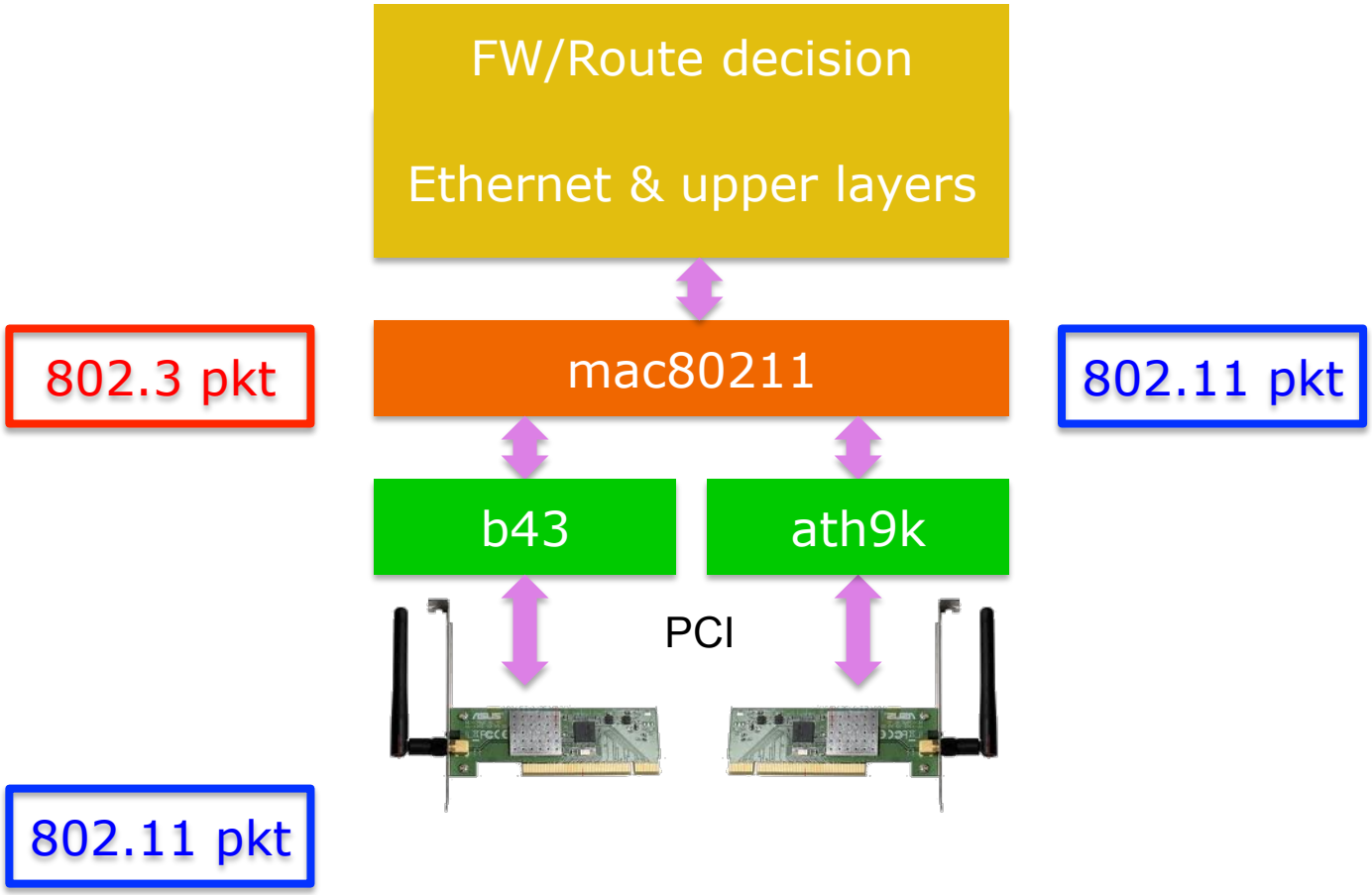


Linux & 802.11

- Opposite path: conversions reversed
- ☹ Several operations involved for each packet
- ☺ Multiple buffer copies (should be) avoided
  - E.g., original packet at layer 4 correctly allocated
    - Before L3 encapsulation output device already known
- ☹ Packets are queued twice/(3 times ☺)
  - Qdisc: before wrapper
  - Device queues: between wrapper and driver/(+DMA)
- Bottom line:
  - Clean design but can be resource exhausting

Linux & 802.11
Modular architecture

- Forwarding/routing packet on a double interface box



Linux & 802.11

- On CPU limited platform, fw performance too low
  - Need to accelerate/offload some operations
- Ralink was first to introduce SoC WiFi devices
  - A mini-pci card hosts an ARM CPU
  - Main host attaches a standard ethernet iface
  - The ARM CPU converts ETH packet to 802.11
  - Main host focuses on data forwarding
- Question: where can be profitably used?
  - Take a look to Andriod phones
  - 2016: new 11ac cards are switching to such approach!!

Linux & 802.11: setup

- A simple BSS with Linux only nodes
  - One station runs hostapd (AP)
  - Others (STAs) join:
    - Once, with iw/iwconfig
    - Use a supplicant to join, e.g., use wpa_supplicant
  - Why using a supplicant?
    - management frame losses➔STA disconnection
    - Why? Kernel (STA) periodically checks if AP is alive
    - If management frames lost, kernel (STA) does not retransmit!
    - A supplicant (wpa_supplicant) is needed to re-join the BSS transparently

Linux & 802.11: kernel setup

- Check the device type with

```
$: lspci | grep -i net
```

- Load the driver for Broadcom devices and check is loaded

```
$: modprobe b43 qos=0
$: lsmod | grep b43
```

- Check kernel ring buffer with

```
$: dmesg | tail -30
```

- Bring net up and configure an IP address

```
$AP: ifconfig wlan0 172.16.0.1 up
$STA: ifconfig wlan0 172.16.0.10 up
```

- In following experiments we fix arp associations

```
$: ip neigh replace to PEERIP lladdr PEERMAC dev wlan0
```

  - Traffic not encrypted
  - QoS disabled

Linux & 802.11: hostapd setup

- Configuration of the AP in “hostapd.conf”

```
interface=wlan0
driver=nl80211
dump_file=/tmp/hostapd.dump
ctrl_interface=/tmp/hostapd
ssid=TESTTODAY
hw_mode=g
channel=14
beacon_int=100
auth_algs=3
wpa=0
```

Try to send SIGUSR1

PIPE used by

BSS properties

No encryption/ authentication

- Runs with

```
$: hostapd -B hostapd.conf    # -B: run in background
```

- Check dmesg!

Linux & 802.11: station setup

- Scan for networks

```
$: iwlist wlan0 scan
```

- Configuration of STAs in wpasupp.conf

```
ctrl_interface=/tmp/wpa_supplicant
network={
        ssid="TESTODAY"
        scan_ssid=1
        key_mgmt=NONE
}
```

PIPE used by wpa_cli

BSS to join

- Runs with

```
$: wpa_supplicant -B -i wlan0 -c wpasupp.conf
```

- Check dmesg!
- **Simple experiment: ping the AP**

```
$: ping 172.16.0.1
```

Linux & 802.11: run some traffic

- We use iperf in UDP mode
- On AP, server mode

```
$: iperf -s -u -p3000 -i1
```

- On STA, client mode

```
$: iperf -c172.16.0.1 -u -p3000 -i1 -t100 -b54M
```

- Channel 14 is usually free (by law)
  - Try another channel, e.g., 1 or 6 or 11
  - How to do it?
  - Reconfigure hostapd and reconnect, let's see how…

Linux & 802.11: check status

- There are some “debug” helpers, on AP:
  - Browse this folder

    `/sys/kernel/debug/ieee80211`
  - Learn what is `phy0`
  - Cd to `phy0/stations`
  - Cd to the MAC address of the STA!!
    - Explore all the stats
    - Why rc_stats is almost empty?
- What on the STA?

Linux & 802.11: capturing packets

- On both AP and STA run “tcpdump”

```
$: tcpdump -i wlan0 -nn
```

- Is exactly what we expect?
  - What is missing?
  - Layer 2 acknowledgment?
- Display captured data

```
$: tcpdump -i wlan0 -nn -XXX
```

- What kind of layer 2 header?
- What have we captured?

Linux & 802.11: capturing packets

- Run “tcpdump” on another station set in monitor mode

```
$: ifconfig wlan0 down
$: iwconfig wlan0 mode monitor chan 4(?)
$: ifconfig wlan0 up
$: tcpdump -i wlan0 -nn
```

- What’s going on? What is that traffic?
  - Beacons (try to analyze the reported channel, what’s wrong?)
  - Probe requests/replies
  - Data frames
- Try to dump some packet’s payload
  - What kind of header?
  - Collect a trace with tcpdump and display with Wireshark

Linux & 802.11: capturing packets

- Exercise: try to capture only selected packets
- Play with matching expression in tcpdump

```
$: [cut] ether[N] ==|!= 0xAB
```

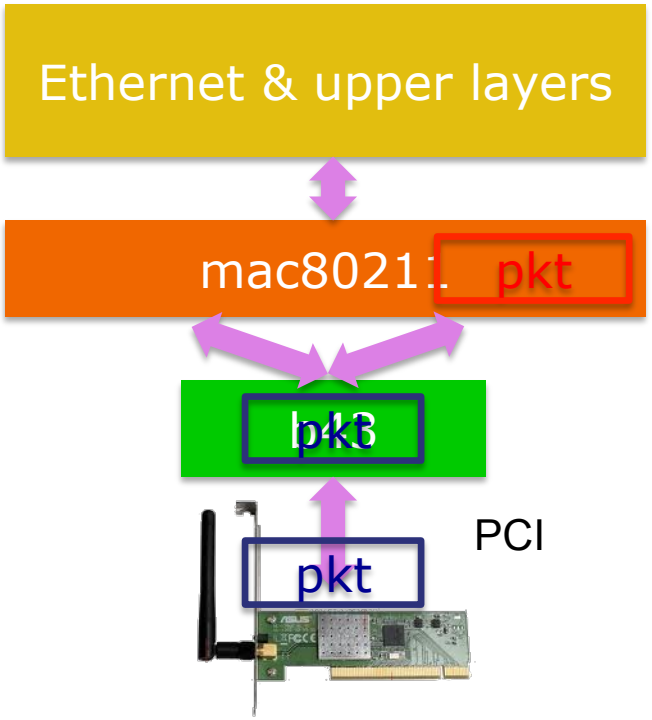
- Discard beacons and probes
- Display acknowledgments
- Display only AP and STA acknowledgments
- Question: is a third host needed?

Virtual Interfaces

- Wrapper/driver “may agree” on virtual packet path
  - Each received packet duplicated by the driver
  - mac80211 creates many interfaces “bound” to same HW
  - In this example
    - Monitor interface attached
    - Blue stream follow upper stack
    - Red stream hooked to pcap

```
$: iw dev wlan0 interface add \
          fish0 type monitor
```

  - Try capturing packets on the AP
    - What’s missing?



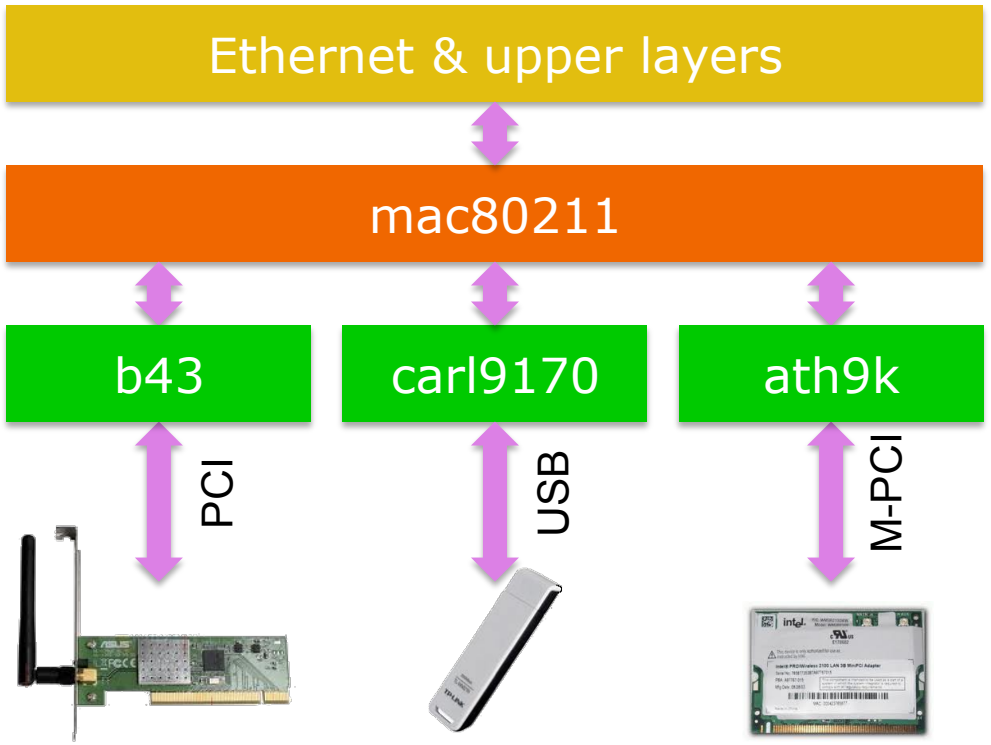
Descent to layer 2 and below
An open firmware

A glimpse into the Linux Kernel Wireless Code

Part 2

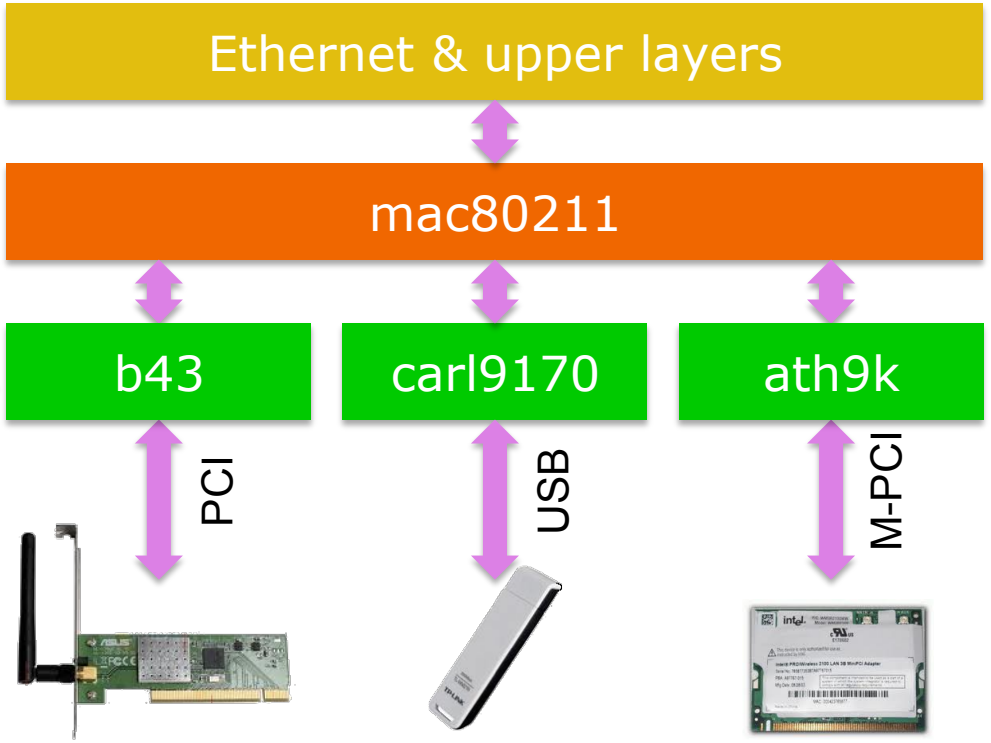
Linux & 802.11
Modular architecture

Wrapper for all hw
Find interface;
remove eth head;
add LLC&dot11 head;
fill (sa;da;ra;seq);
fill(control;duration);
set rate (from RC);
fill (rate;fallback);



Linux & 802.11 Modular architecture/2

Set up hw regs;
Fill hw private fields;
Send frame on DMA;
Get stats;
Reports to mac80211
Several MAC primitives missing!
Who takes care of ack?



Linux & 802.11 Modular architecture/3

Ethernet & upper layers

For sure

We will see the firmware in this course
but first…
Let’s check why we should do that ☺

Firmware does

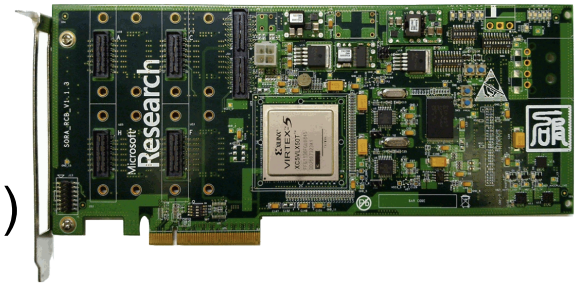


Why/how playing with 802.11

- Radio access protocols: issues
  - Some are unpredictable: noise & intf, competing stations
- Experimenting with simulators (e.g., ns-3)
  - Captures all “known” problems
    - Testing changes to back-off strategy is possible ☺
  - Unknown (not expected)?
    - Testing how noise affects packets not possible ☹
- **In the field testing is mandatory**
  - Problem: one station is not enough!

Programmable Boards

- Complete platforms like
  - RICE-WARP: Wireless open-Access Research Platform
  - NI-RIO2940
  - Microsoft SORA
  - Based on FPGA
  - Everything can be changed
    - MAC + PHY (access to OFDM symbols!)
  - Two major drawbacks
    - More than very expensive
    - Complex deployment
  - **If PHY untouched: look for other solutions!**

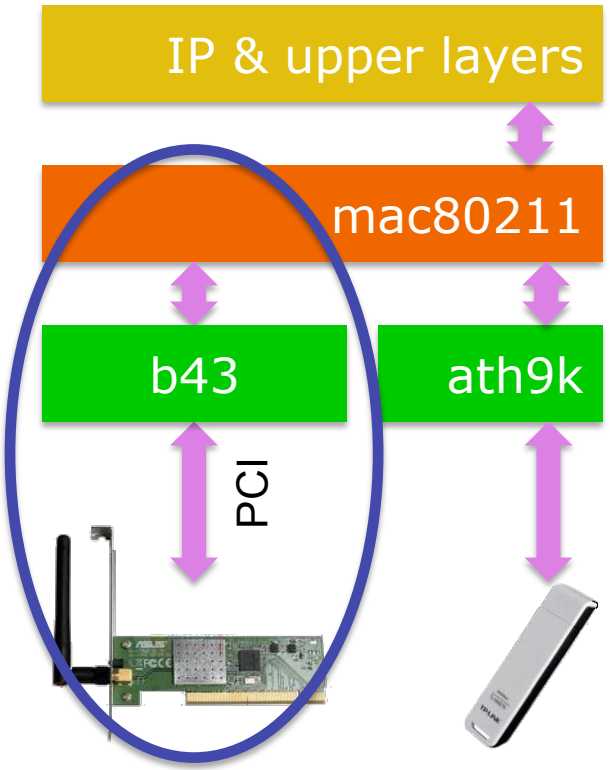


Off-the-shelf hardware

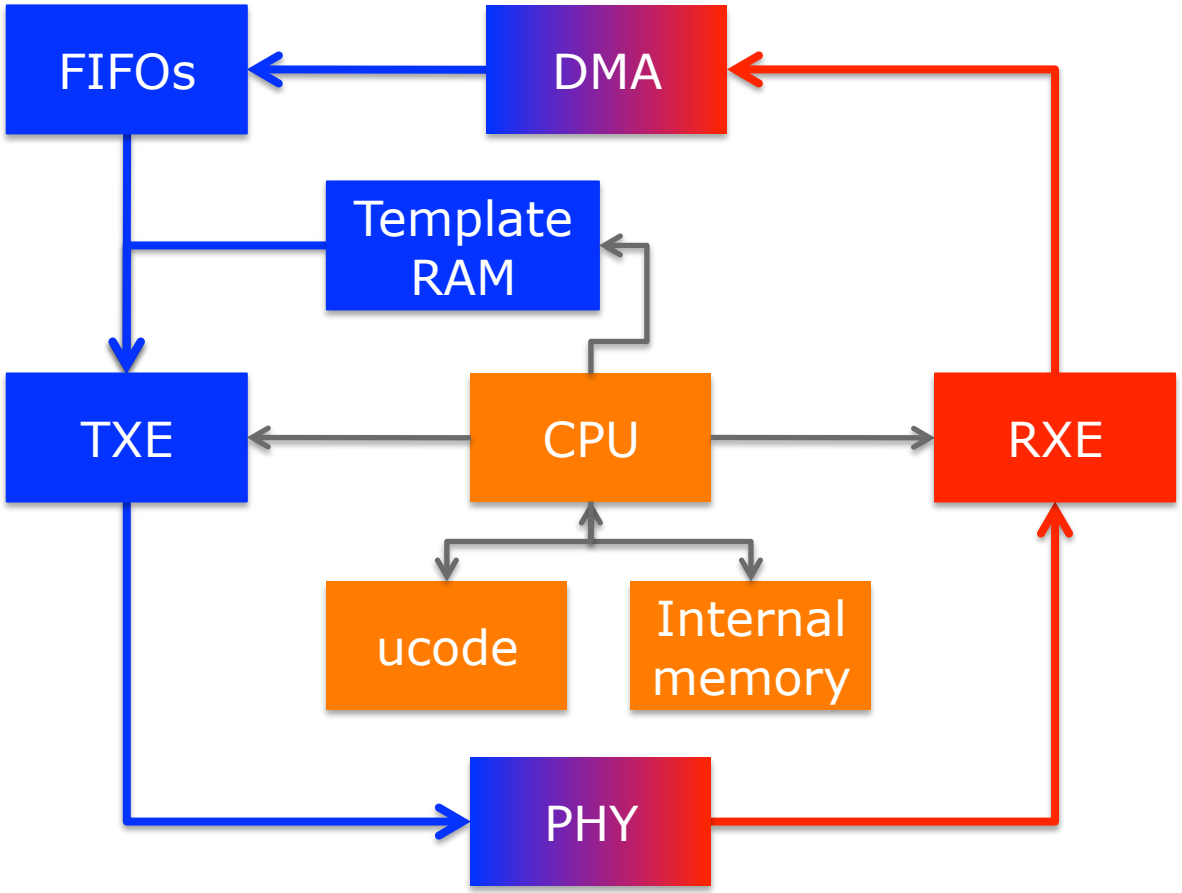
- Five/Six vendors develop cheap WiFi hw
  - Hundreds different boards
  - Almost all boards load a binary firmware
    - MAC primitives driven by a programmable CPU
  - Changing the firmware ➔ Changing the MAC!
- Target platform:
  - Linux & 802.11: modular architecture
  - Official support prefers closed-source drivers ☹
  - Open source drivers && Good documentation
    - Thanks to community! ☺

Linux & 802.11 Broadcom AirForce54g

- Architecture chosen because
  - Existing asm/dasm tools
    - A new firmware can be written!
  - Some info about hw regs
- We analyzed hw behavior
  - Internal state machine decoded
  - Got more details about hw regs
  - Found timers, tx&rx commands
  - Open source firmware for DCF possible
- We released OpenFWWF!
  - OpenFirmWare for WiFi networks



Broadcom AirForce54g Basic HW blocks



Description of the HW

- CPU/MAC processor capabilities
  - 88MHz CPU, 64 general purpose registers
- Data memory is 4KB, direct and indirect access
  - From here on it's called Shared Memory (SHM)
- Separate template memory (arrangeable > 2KB)
  - Where packets can be composed, e.g., ACKs & beacons
- Separate code memory is 32KB (4096 lines of code)
- Access to HW registers, e.g.:
  - Channel frequency and tx power
  - Access to channel transmission within N slots, etc…

TX side

- Interface from host/kernel
  - Six independent TX FIFOs
  - DMA transfers @ 32 or 64 bits
  - HOL packet from each FIFO
    - can be copied in data memory
      - Analysis of packet data before transmission
      - Kernel appends a header at head with rate, power etc
    - can be transmitted “as is”
    - can be modified and txed
      - Direct access to first 64 bytes

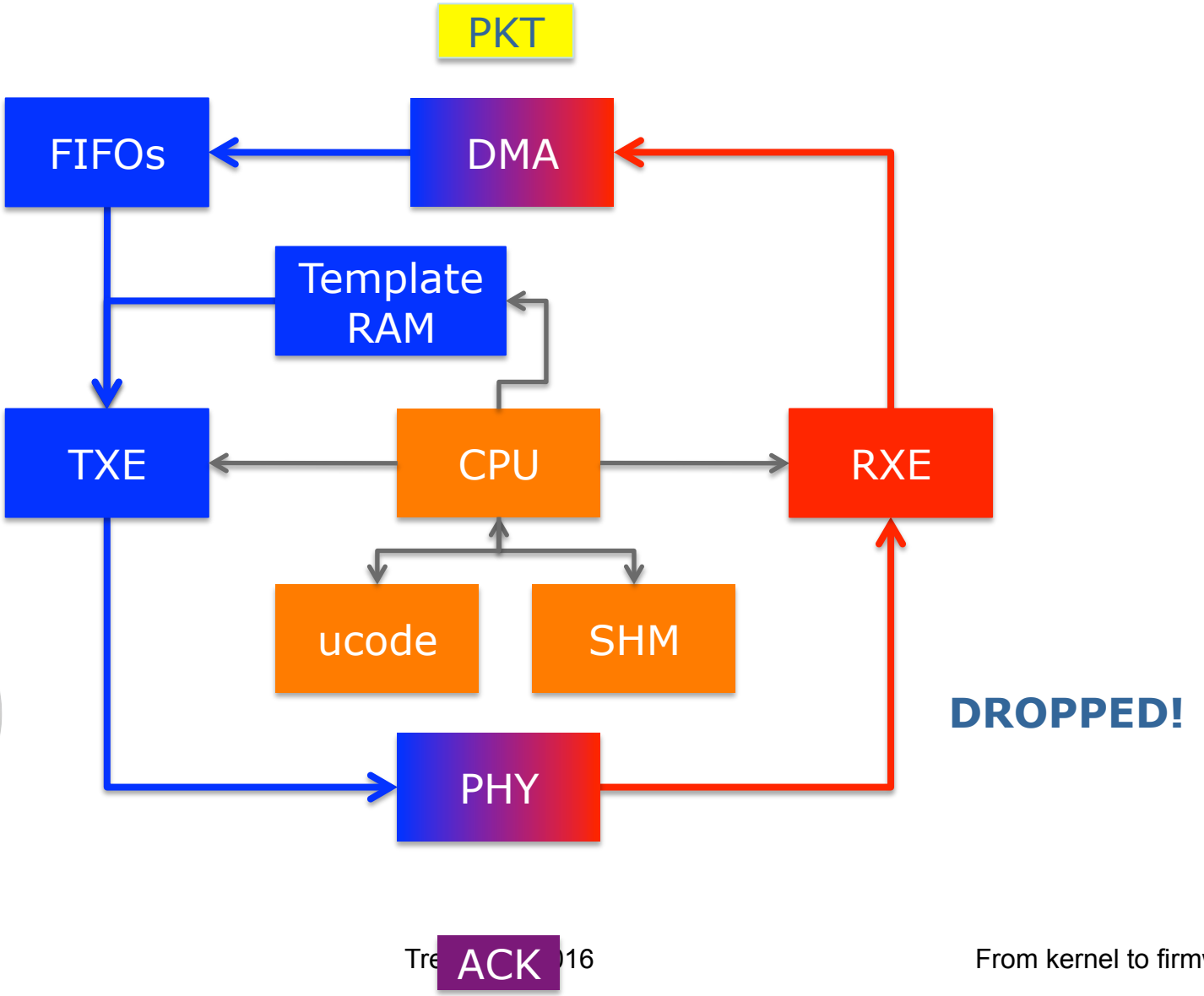
TX side/2

- Interface to air
  - Only 802.11 b/g supported, soon n
  - Full MTU packets can be transmitted (~2300bytes)
    - If full packet analysis is needed, analyze block-by-block
  - All 802.11 timings supported
    - Minimum distance between Txed frames is 0us
      - Note: channel can be completely taken by such firmware!!
  - Backoff implemented in software (fw)
    - Simply count slots and ask the HW to transmit

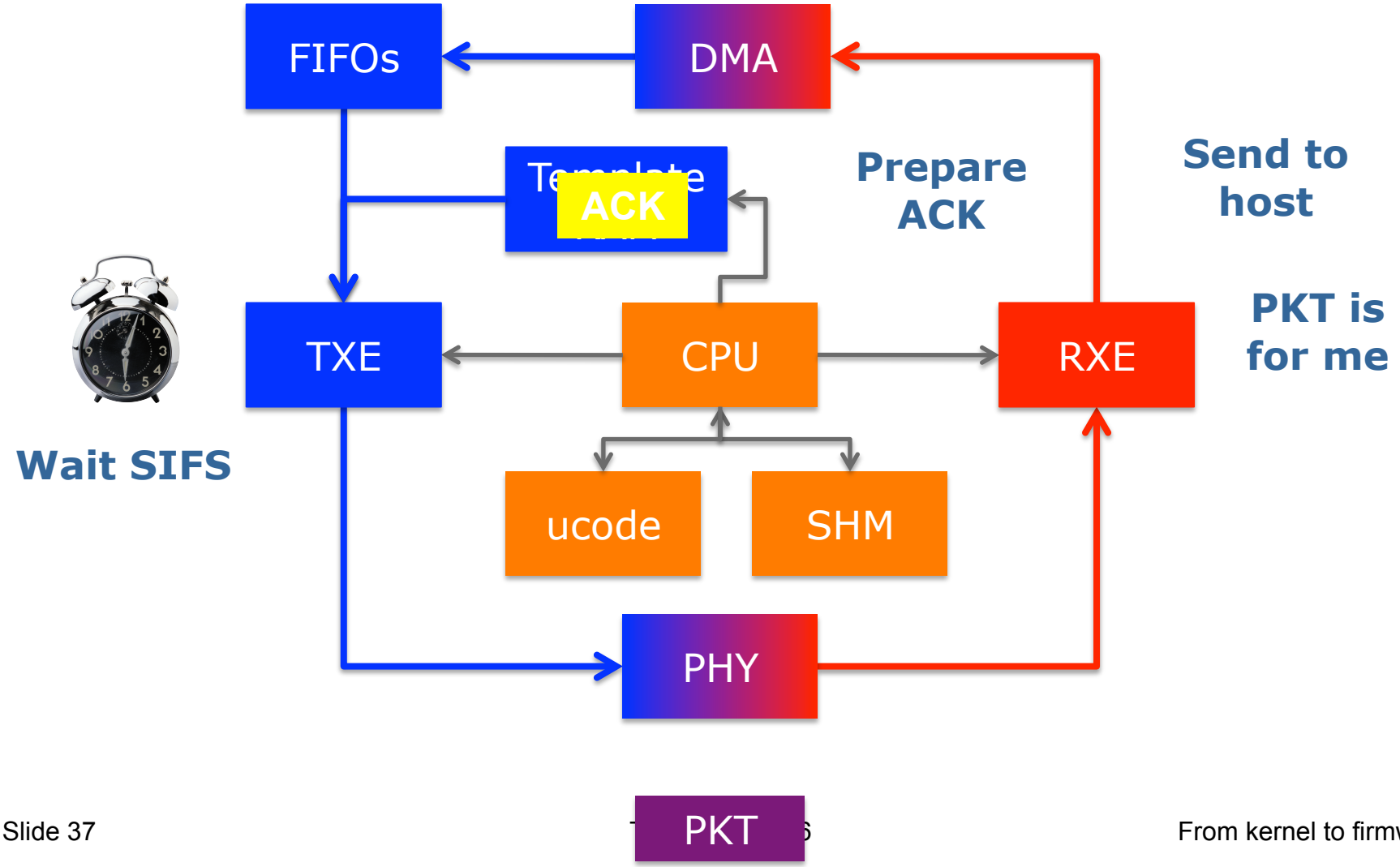
RX side

- Interface from AIR
  - HW acceleration for
    - PLCP and global packet FCS - Destination address matching
  - Packet can be copied to internal memory for analysis
    - Bytes buffered as soon as symbols is decoded
  - During reception and copying CPU is idle!
    - Can be used to offload other operations
    - Try to suggest something
  - Packets are pushed to host/kernel
    - If FW decides to go and through one FIFO ONLY
    - May drop! (e.g., corrupt packets, control…)

Example:
TX a packet, wait for the ACK



Example:
RX a packet, transmit an ACK



What lesson we learned

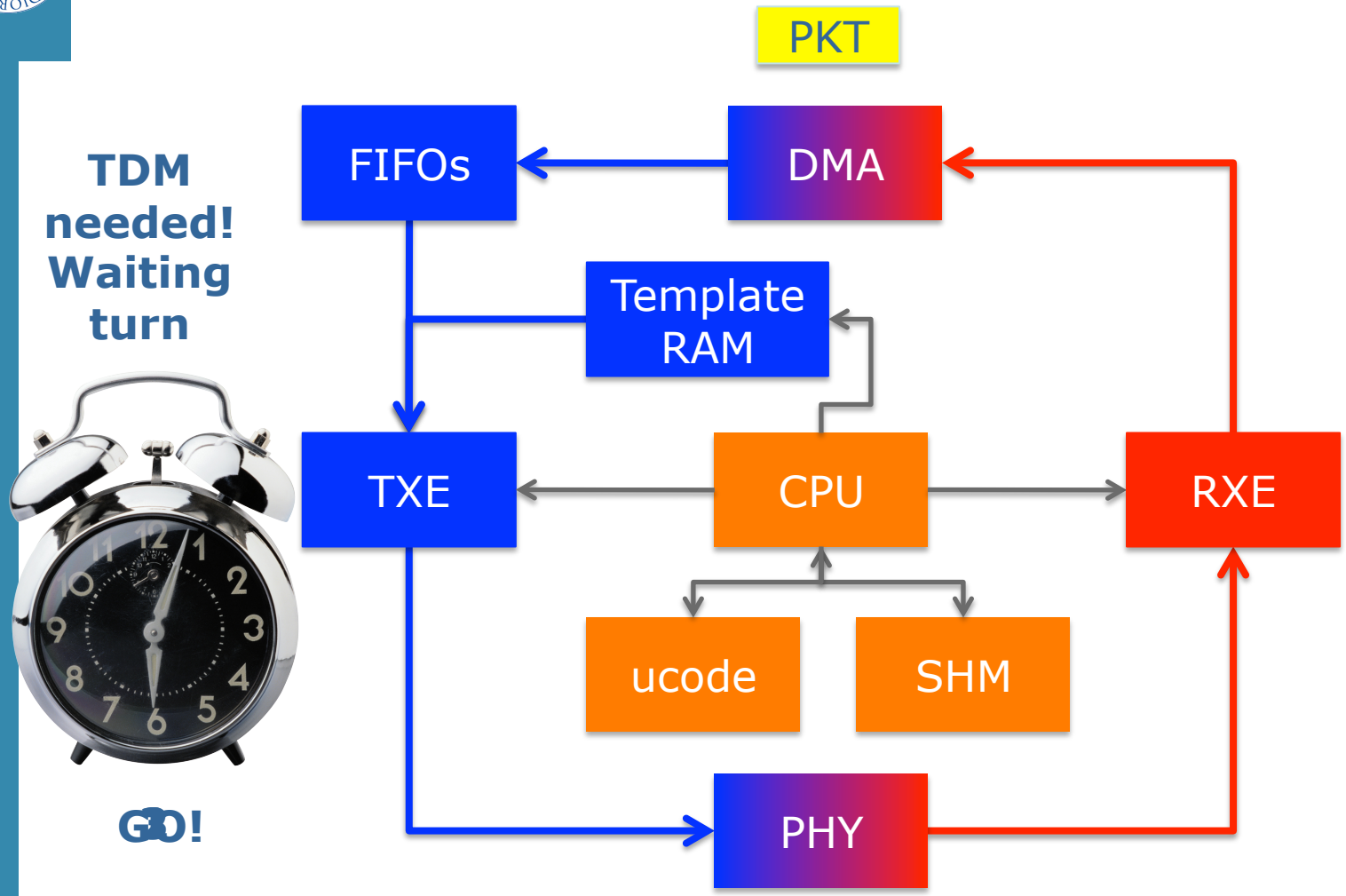
- From the previous slides
  - Time to wait ack (success/no success)
  - Dropping ack (rcvd data not dropped, goes up)
  - And much more
    - When to send beacon
    - Backoff exponential procedure and rate choice
  - Decided by MAC processor (by the firmware)
- Bottom line:

  Hardware is (almost) general purpose

From lesson to OpenFWWF Description of the FW

- OpenFWWF
  - It's not a production firmware
  - It supports basic DCF
    - No RTS/CTS yet, No QoS, only one queue from Kernel
  - Full support for capturing broken frames
  - It takes 9KB for code, it uses < 200byte for data
    - **We have lot of space to add several features**
- Works with 4306, 4311, 4318 hw
  - Linksys Routers supported (e.g., WRT54GL)

Broadcom AirForce54g Simple TDM



Broadcom AirForce54g Simple TDM/2