Portable Platform-agnostic programming of Wireless MAC protocols

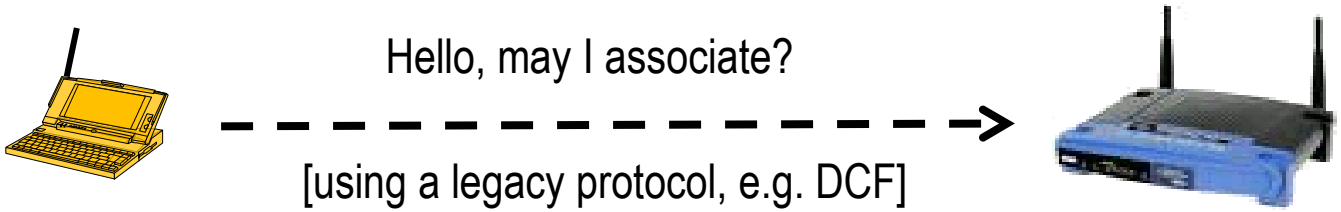
Giuseppe Bianchi, CNIT / University of Roma Tor Vergata

Joint work with: I. Tinnirello, N. Facchi, P. Gallo, D. Garlisi, F. Giuliano, F. Gringoli

Supported by:
EU FP7-FLAVIA STREP project
EU FP7-CREW FIRE project

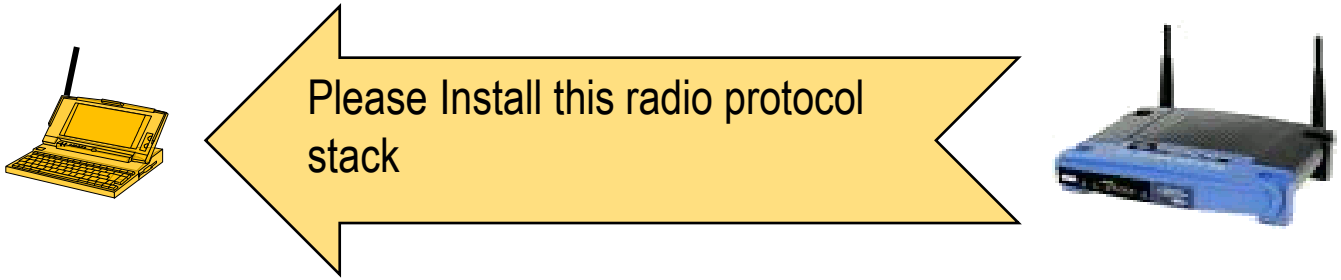
Our vision:
Context-specific MAC protocol

(dynamically installed at run-time)



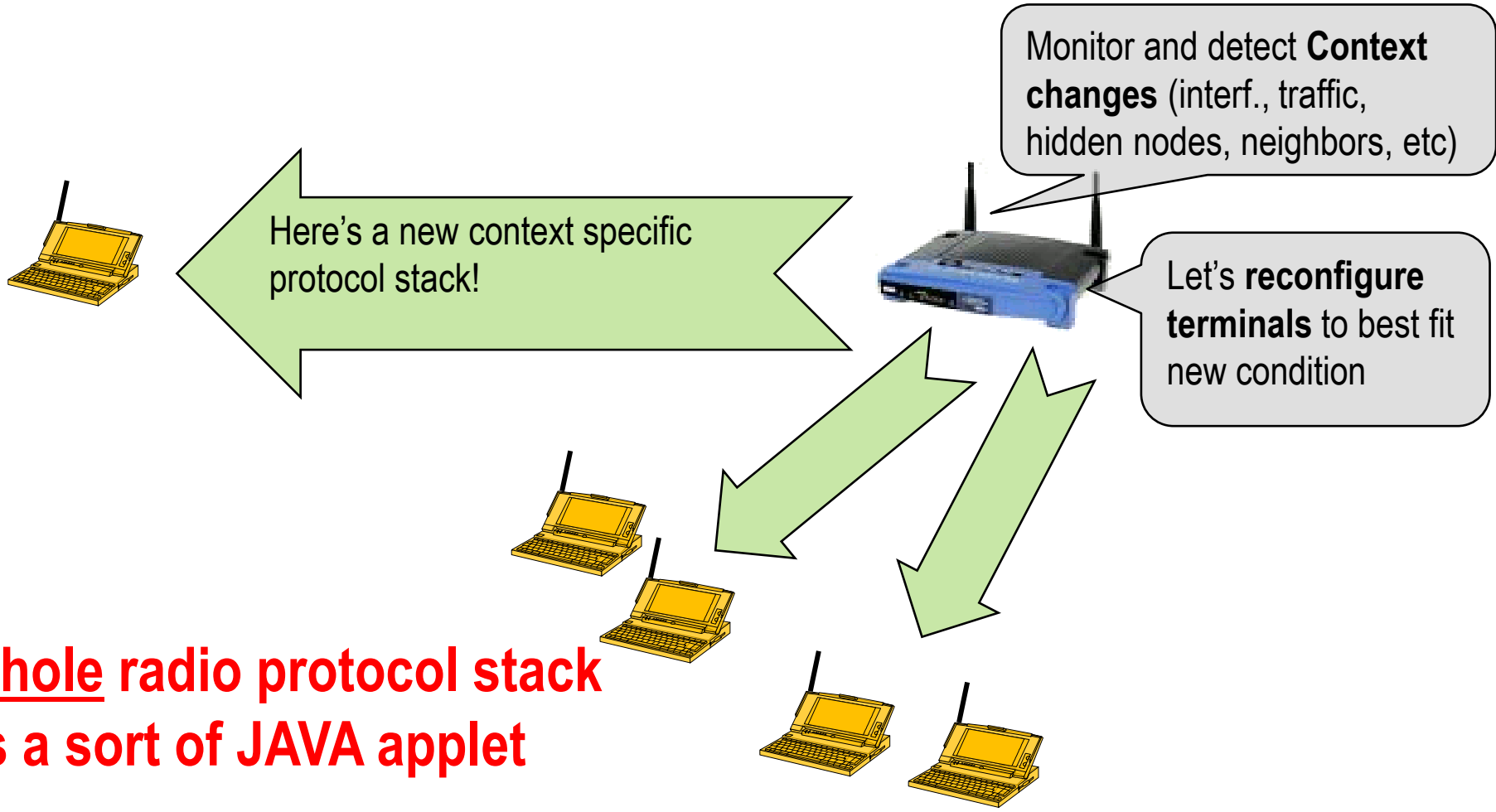
Our vision:
Context-specific MAC protocol

(dynamically installed at run-time)



Our vision:
Context-specific MAC protocol

(dynamically installed at run-time)



Whole radio protocol stack as a sort of JAVA applet

Motivation: one size does not fit all

different needs; many exploitation opportunities

- **→Dynamic spectrum access**
  - →Very diverse situations (e.g. countries)
  - →very diverse environmental conditions
    - » Dense/sparse, hidden terminals, legacy deployments, etc
  - →Different propagation characteristics
    - » Sub-GHz, THz
- **→Niche environments with specific needs**
  - →home, industrial, M2M, …
  - →Adaptation to specific context or applications
- **→Virtualization, access network sharing**
  - →Multi-tenancy
  - →Multiple coexisting applications
    - » M2M and H2H over same access network…
- **→And many more…**

Rediscovering the wheel? We do have SW defined radios!

➔**from radios with behavior fixed in hardware to radios with behavior determined by software**

➔**20+ years long research path; excellent technologies**

- ⇨E.g., restricting to research/academic platforms:
  - →AirBlue, CalRadio, GNURadio, RUNIC, SORA, USRP, WARP, …
- ⇨Permit to SW program «almost everything»
  - →Waveforms, PHY, MAC, etc

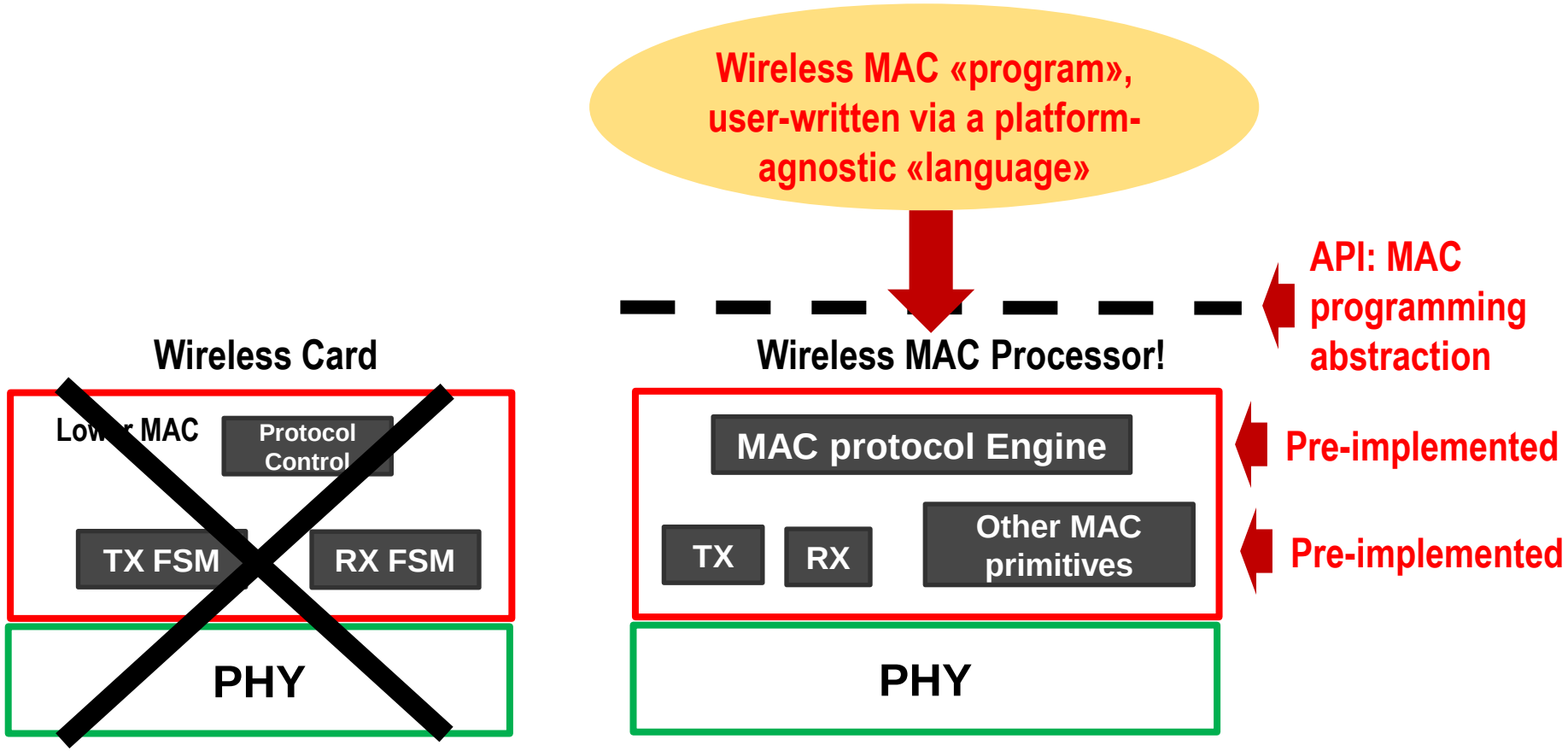
No: there is a serious concern!

Mostly neglected so far…

- **Different platforms/vendors = completely different programming models and languages!**
  - ⇨ Either force ALL to use the same platform
  - ⇨ Or port your protocol code for all possible platforms (a babel!)

- **Quoting C. Partridge, "realizing the future of Wireless Data Communication", Commun. ACM, 2011** *[talking about the transition from HW to SW radio]*:
  - ⇨ *… [we] are all **imperfectly prepared to take advantage**; research on vital questions has been extremely variable, with **wonderful work** (such as finding the right mix of programmable hardware to support high performance signal processing in radios) undermined by **almost complete neglect** - such as **how to describe radio behavior independent of platform** […].*

Our key idea: turn NIC into a «wireless MAC processor»



How to? mimic an ordinary computing system!

➔ **1: Instruction sets**
perform elementary tasks on the platform

→A-priori given by the platform

→Can be VERY rich in special purpose computing platforms

» Crypto accelerators, GPUs, DSPs, etc

➔ **2: Programming abstraction & languages**
sequence of such instructions + conditions

⇨ Convey desired platform's operation or algorithm

➔ **3: Central Processing Unit (CPU)**
execute program over the platform

⇨ Unaware of what the program specifically does

⇨ Fetch/invoke instructions, update registers, etc

Clear decoupling between:

- platform's vendor → implements (closed source!) instruction set & CPU

- programmer → produces SW code in given language

1: Which elementary MAC tasks?

("our" instruction set!)

➔ **ACTIONS**

⇨ **frame management, radio control, time scheduling**

➔TX frame, set PHY params, RX frame, set timer, freeze counter, build header, forge frame, switch channel, etc

➔ **EVENTS**

⇨ **available HW/SW signals/interrupts**

➔Busy channel signal, RX indication, inqueued frame, end timer, etc

➔ **REGISTRY CONDITIONS**

⇨ **boolean/arithmetic tests on available registers/info**

➔Frame address == X, queue length >0, ACK received, power level < P, etc

Set of supported Action, Events, and conditions = MAC programming API!

Current API (version 2.0)

events
CH_UP
CW_DOWN
RX_PLCP_END
RX_MAC_HEAD_END
RX_END
RX_ERROR
TX_START
TX_END
QUEUE_OUT_UP
IFS_EXPIRED
END_TIMER

actions
rx_header()
rx_msdu()
tx_start(prm)
extract_bk(reg, prm)
start_ifs(reg)
update_cw(reg, prm)
repor_to_host(prm)
start_ctrl_ifs(prm)
set_timer(reg, prm)
set_channel(reg)
write(queue, reg, prm)
read(queue, reg, prm)
set(reg, prm)
get(reg, prm)
incr(reg, prm)

conditions
channel
antenna
power
rate
AGC
queue
txrx_on
bk_slot
rx_chksum
busy_time
cw_prms
+ *protocol variables*

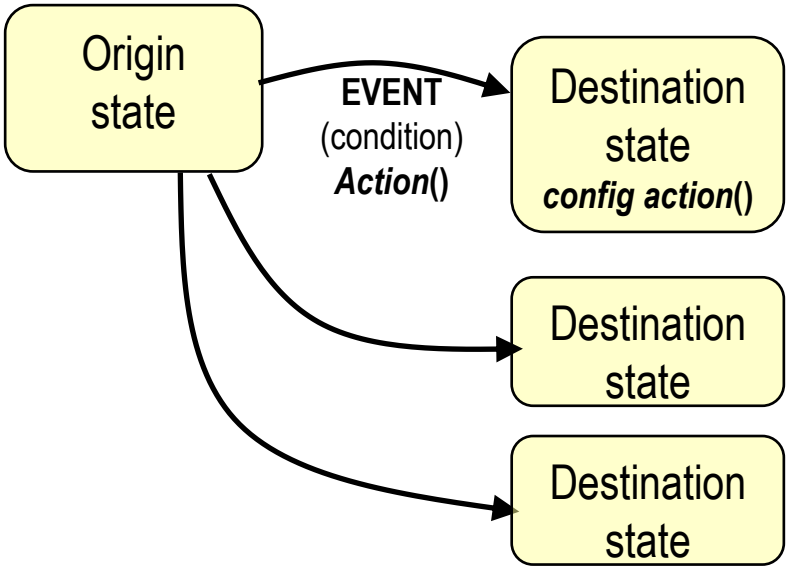
2: How to compose MAC tasks?

("our" programming language!)

➔Convenient abstraction: XFSM

eXtended Finite State Machines

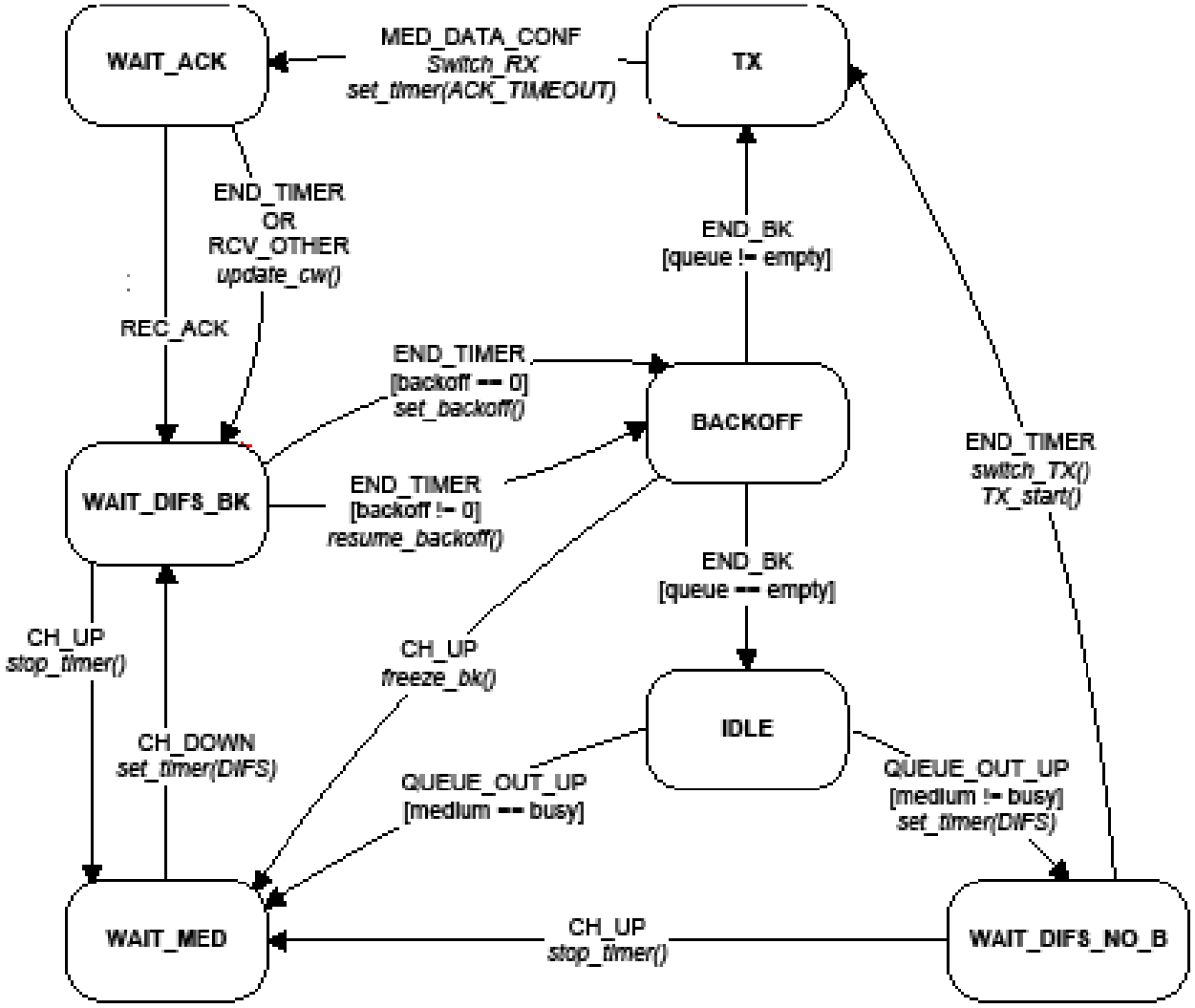
⇨Compact way for composing available actions, events and conditions, to form a custom MAC protocol logic



XFSM formal notation		meaning
S	symbolic states	MAC protocol states
I	input symbols	Events
O	output symbols	MAC actions
D	n-dimensional linear space $D_1 \times \cdots \times D_n$	all possible settings of n configuration registers
F	set of enabling functions $f_i : D \rightarrow \{0,1\}$	Conditions to be verified on the configuration registers
U	set of update functions $u_i : D \rightarrow D$	Configuration commands, update registers' content
T	transition relation $T : S \times F \times I \rightarrow S \times U \times O$	Target state, actions and configuration commands associated to each transition

XFSM example: legacy DCF

simplified for graphical convenience



Actions:
set_timer, stop_timer, set_backoff, resume_backoff, update_cw, switch_TX, TX_start

Events:
END_TIMER, QUEUE_OUT_UP, CH_DOWN, CH_UP, END_BK, MED_DATA_CONF

Conditions:
medium, backoff, queue

3: How to run a MAC program?

(MAC engine – XFSM onboard executor - our CPU!)

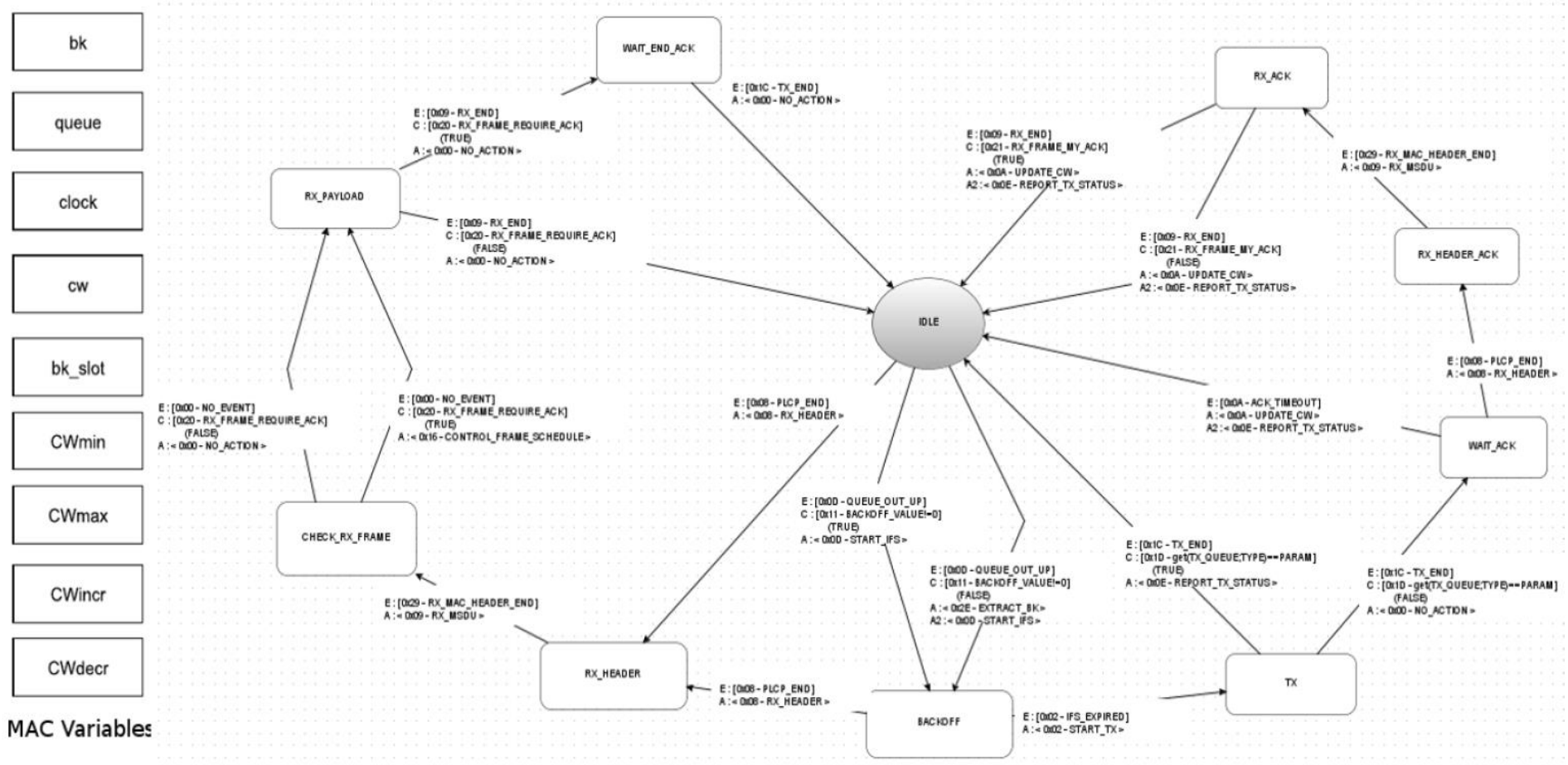
➔**MAC engine: specialized XFSM executor**
(unaware of MAC logic)

- ⇨Fetch state
- ⇨Receive events
- ⇨Verify conditions
- ⇨Perform actions and state transition

➔**Once-for-all implemented in NIC**
(no need for open source)

- ⇨"close" to radio resources = straightforward real-time handling
- ⇨Different MAC protocol = different "XFSM program"!

Programming your MAC: 1- graphical XFSM editor

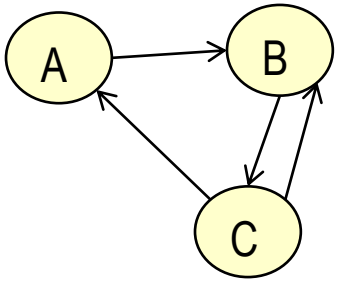


DCF example: define custom variables and state transitions with associated A/E/C chosen among those provided by the API

2- transform XFSM in bytecode

➔ **MAC description:**

⇨ XFSM



➔ **XFSM → tables**

	A	B	C
A		T(A,B)	
B			T(B,C)
C	T(C,A)	T(C,B)	

➔ **Transitions**

⇨ «byte»-code event, condition, action

→**Portable over different vendors' devices, as long as API is the same!!**

⇨ Pack & optimize in WMP «machine-language» bytecode

A	T(A,B)	
B	T(B,C)	
C	T(C,A)	T(C,B)

MAC protocol specification:
XFSM design
(e.g. Eclipse GMF)

Machine-readable code

Custom language compiler

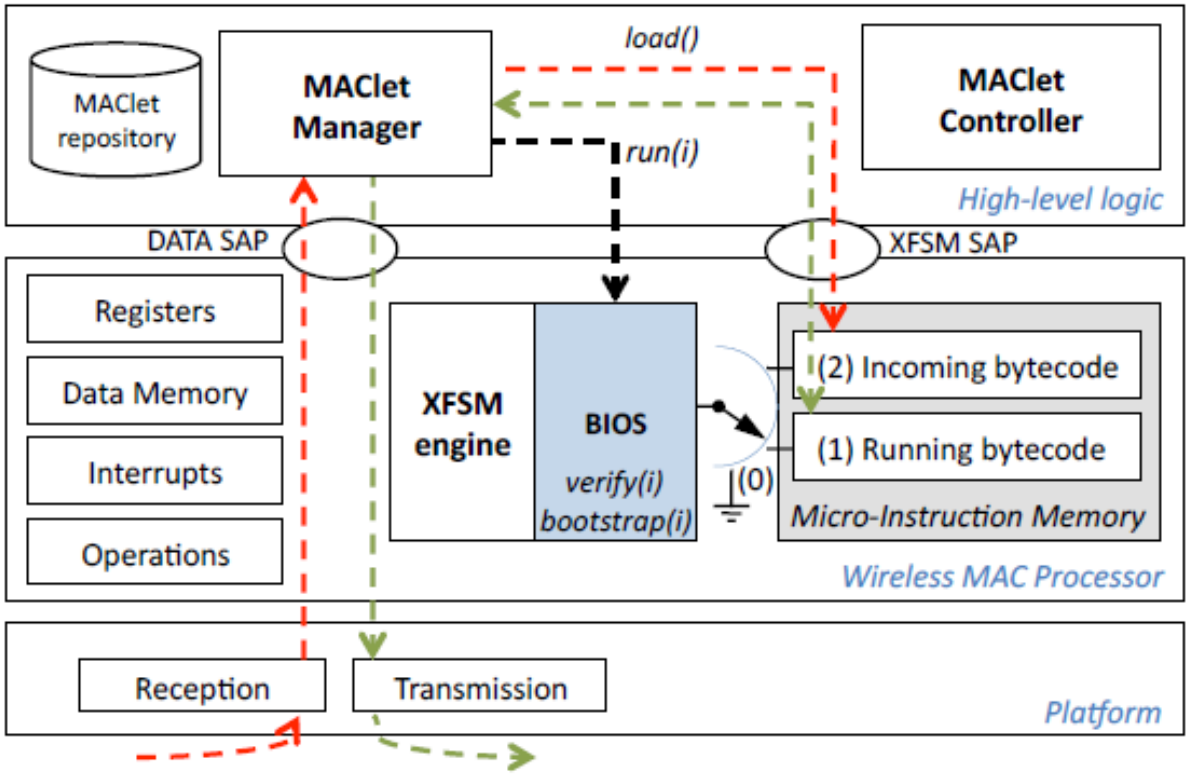
Code injection in radio HW platform

Machine Language Bytecode Example
(DCF, 544 bytes)

3- upload «MAClet» on device

- **Developed local control agents**
  - Can upload via PC interface or via packet (akin to active networks)
- **Manages two state machines**
  - Can upload while another MAC is running

- **WMP Control Primitives**
  - load(XFSM)
  - run(XFSM)
  - verify(XFSM)
  - switch(XFSM1, XFSM2, ev, cond)
- **Further primitives**
  - Synchro support for distributed start of same MAC operation
  - Distribution protocol

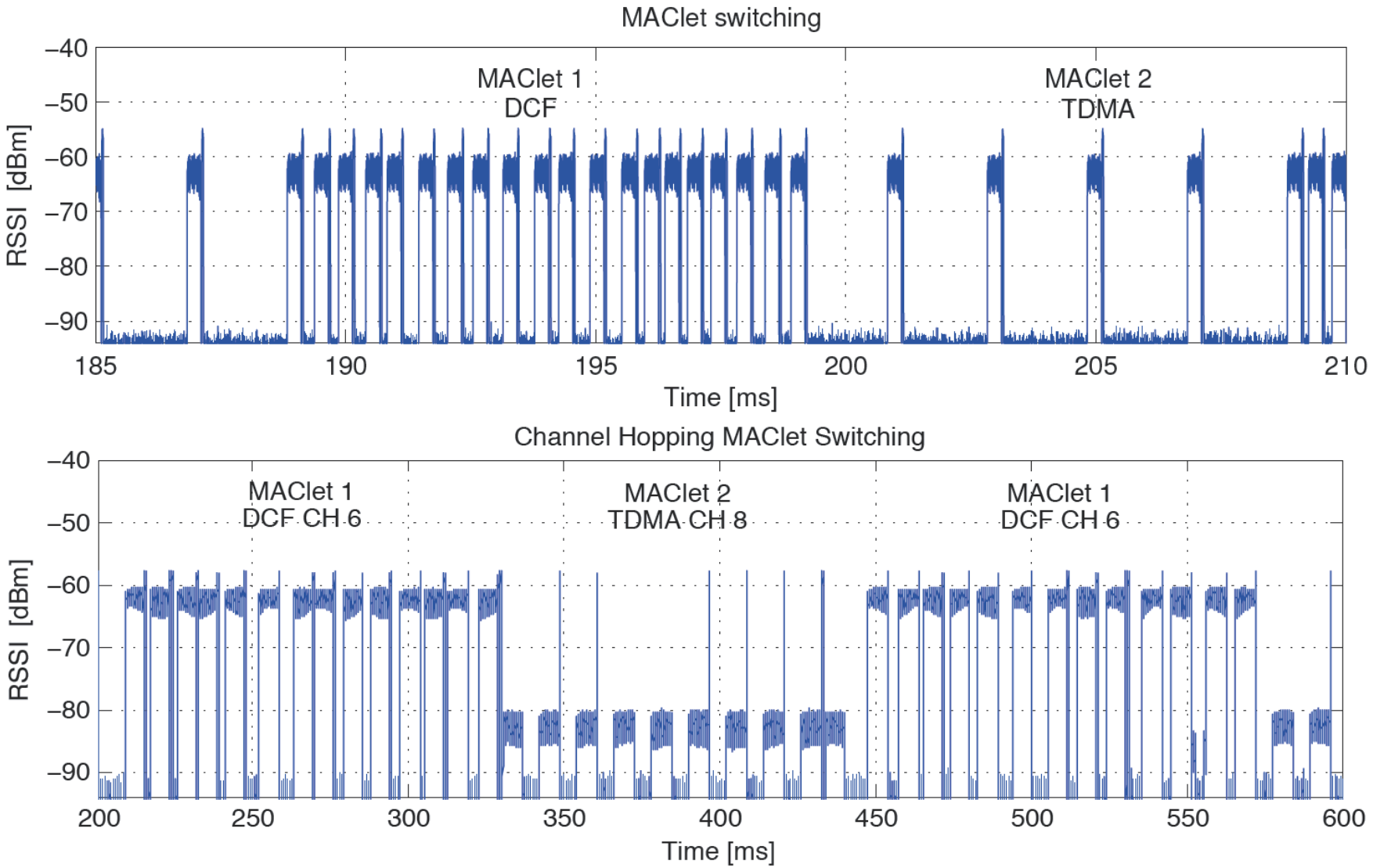


"Bios" state machine: DEFAULT protocol (e.g. wifi) which all terminals understand

4- switch from «old» to «new» MAC

Permits MAC «multi-threading»!! Switching time = less than 0.2 us over cheap broadcom!

(plus channel switching time if required)

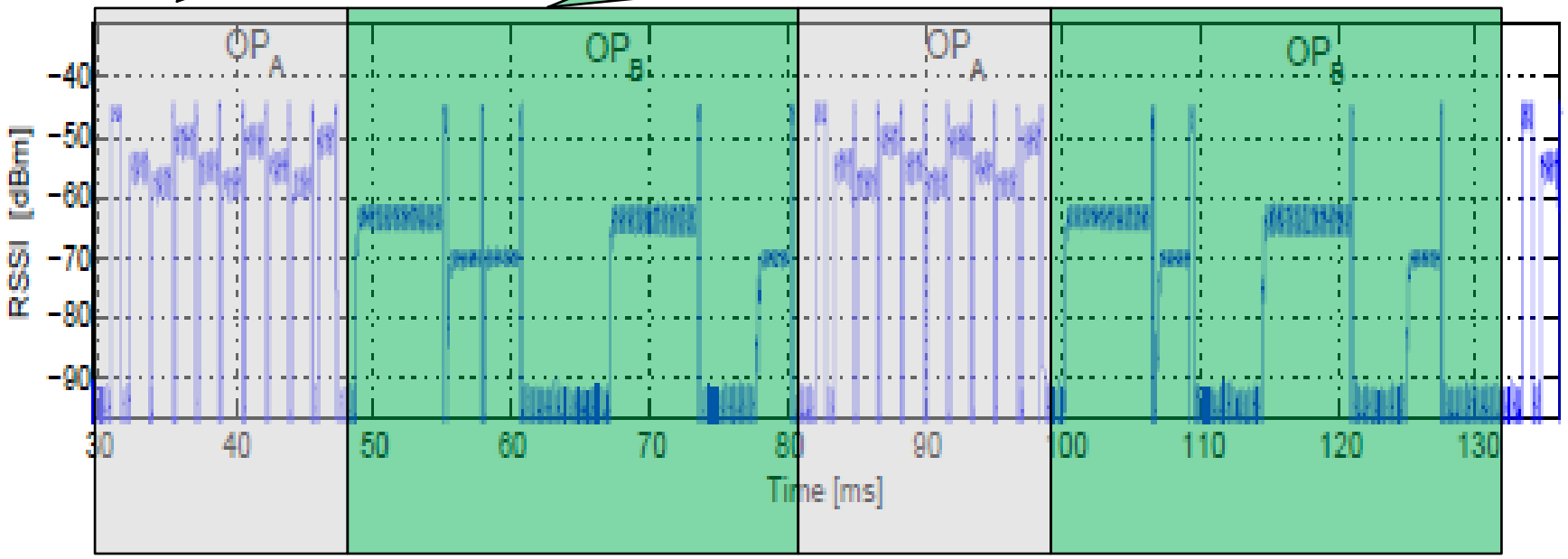


MAC protocols' virtualization

time slicing of different MAC protocols on same device/channel

Time «slice» dedicated to OP_A, best effort traffic → chooses DCF-like

Time «slice» dedicated to OP_A, guaranteed traffic → chooses TDM-like



trivial idea, isolation guaranteed, any (custom) MAC protocol in any tenant' slice…
but today hard to implement (and non standard)

Two proof of concept implementations

PLATFORM

- **Ultra-cheap commodity WLAN NIC: Broadcom Airforce54g 4311/4318**
- **Implementation at a glance:**
  - Deleted original 802.11 firmware
    - we do NOT want a "firmware MAC" to hack!
  - Replaced with [once for all developed in assembly language]:
    - Implementation of actions, events, conditions (in part reusing existing HW facilities)
    - Implementation of a MAC engine (XFSM executor)

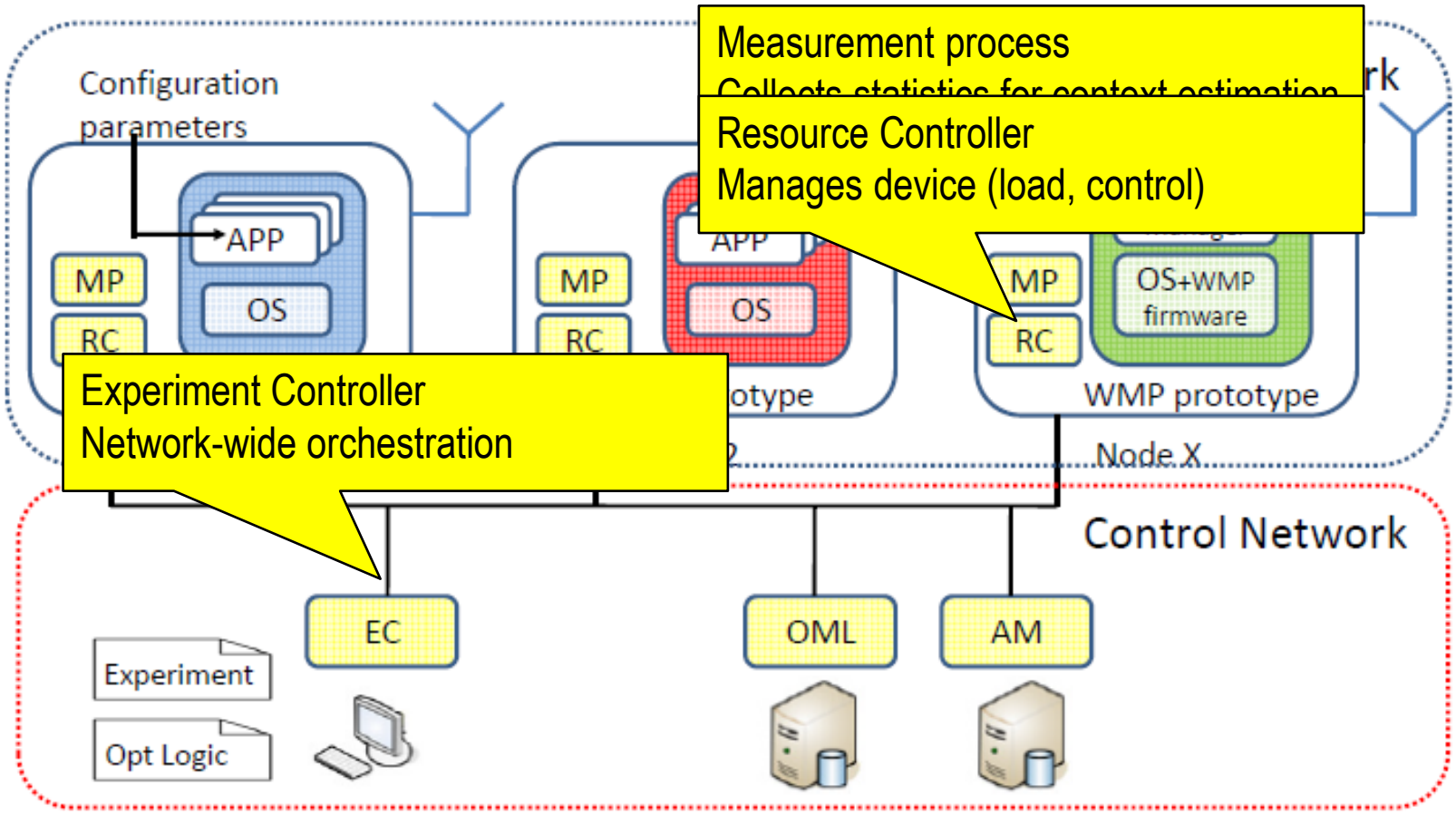
OUR GOAL

- **Show viability on commodity HW cards with poor resources**
  - general purpose processor (88 MHz),
  - 4KB data memory
  - 32 KB code memory

- **WARP SDR board**
  - Much more powerful
  - Not easy: required to «clean» several existing inter-dependencies among primitives

- **Paves the road towards future PHY/MAC extensions**
  - Antenna control, OFDM, MIMO, etc
- **Show portability of MAC programs**
  - With same API, MAC programs for Broadcom perfectly install and run on WARP!!

SDN-like controller (adapting the OMF framework)



Public-domain

➔ **Supported by the FLAVIA EU FP7 project**
- ⇨ http://www.ict-flavia.eu/

➔ **Ongoing integration in the CREW EU FP7 federated testbed**
- ⇨ http://www.ict-flavia.eu/

➔ **Public domain release**
- ⇨ **Project page:** http://wmp.tti.unipa.it
- ⇨ **Download:** https://github.com/ict-flavia/Wireless-MAC-Processor

➔ **Released distribution:**
- ⇨ Binary image for WMP
- ⇨ You DO NOT need it open source!
  Remember the "hard-coded" device philosophy…
  - ➔ Conveniently mounted and run on Linksis or Alix
- ⇨ Source code for everything else
- ⇨ Manual & documentation, sample programs

Conclusions and Future steps

➔ **Platform agnostic Wireless MAC protocol programming is technically possible (and viable)**

⇨ Key insight: XFSM as MAC programming abstraction

➔ **Decouples wireless innovation from their deployment**

⇨ Vendor's role: improve wireless primitives implemented in the card

⇨ Operator/deployer's role: «use» primitives available to program the desired wireless access operation

➔ **Permits context-specific MAC protocols**

⇨ Upload at run time the «most appropriate» protocol for your specific environment

➔ **What about wireless PHY and cross-layer?**

⇨ Open challenge: programming PHY radio behavior using platform-agnostic abstractions

→XFSM appear not sufficient; must integrate DAGs?