WiMAX Tutorial

GREE-SC 2013

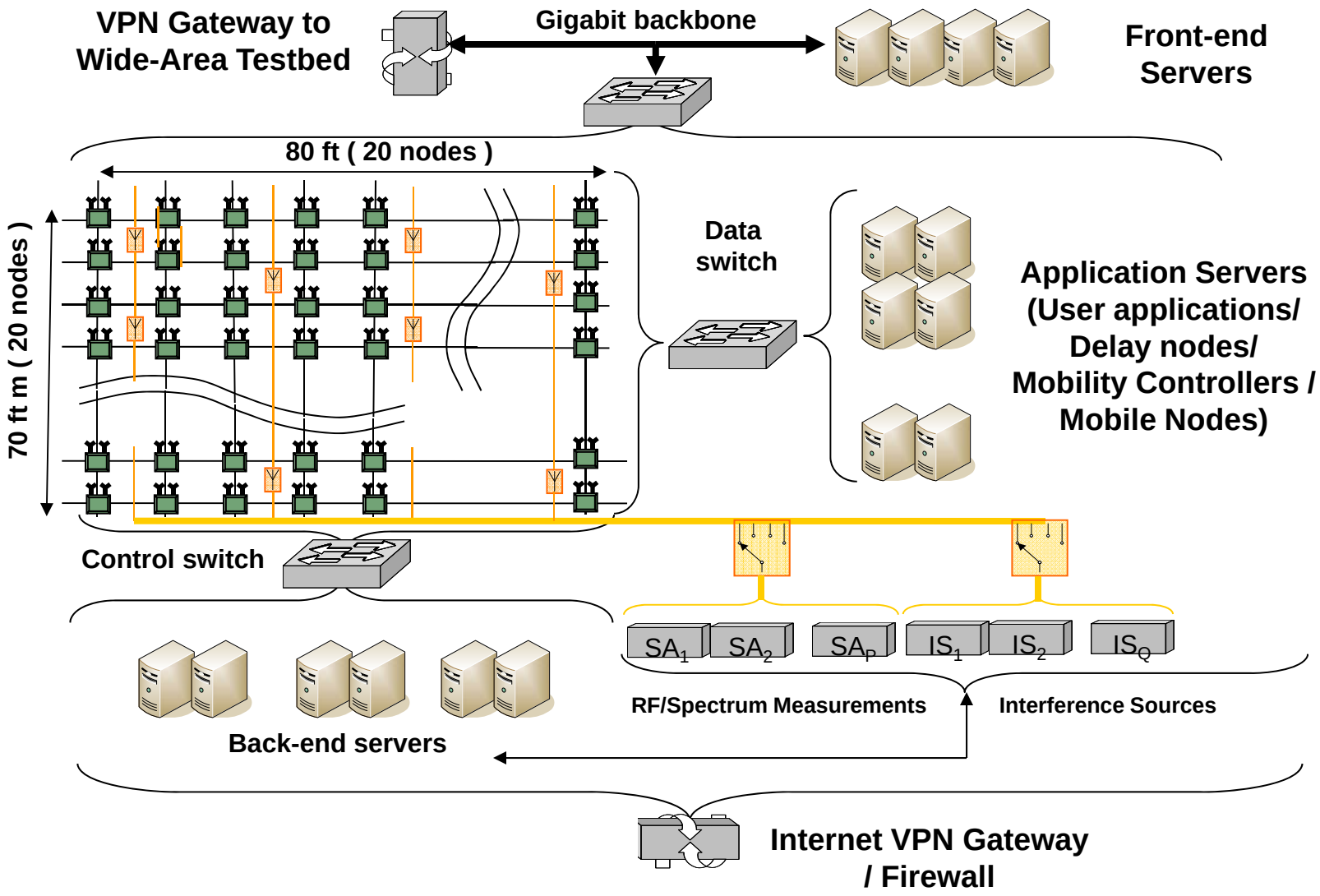
Ivan Seskar, Associate Director
WINLAB
Rutgers, The State University of New Jersey
Contact: seskar (at) winlab (dot) rutgers (dot) edu

RUTGERS

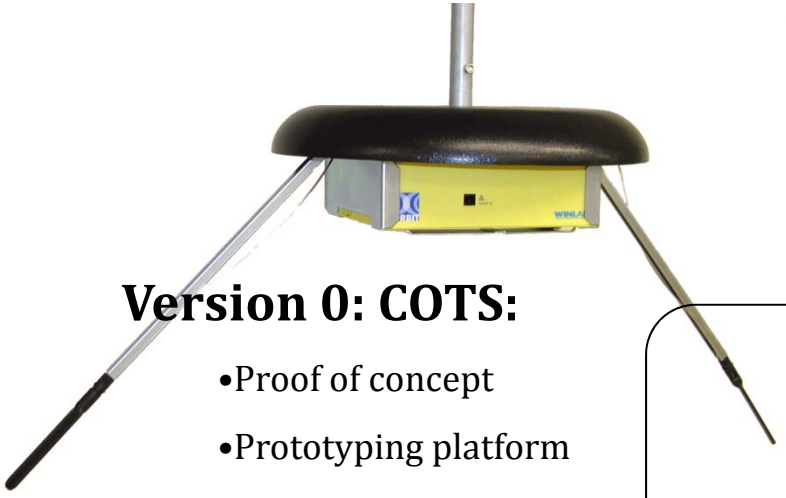
WINLAB

Hardware

GRID


VPN Gateway to Wide-Area Testbed
Gigabit backbone
Front-end Servers
80 ft (20 nodes)
70 ft m (20 nodes)
Data switch
Application Servers (User applications/ Delay nodes/ Mobility Controllers / Mobile Nodes)
Control switch
SA1
SA2
SAP
IS1
IS2
ISQ
RF/Spectrum Measurements
Interference Sources
Back-end servers
Internet VPN Gateway / Firewall

Radio Node



Version 0: COTS:

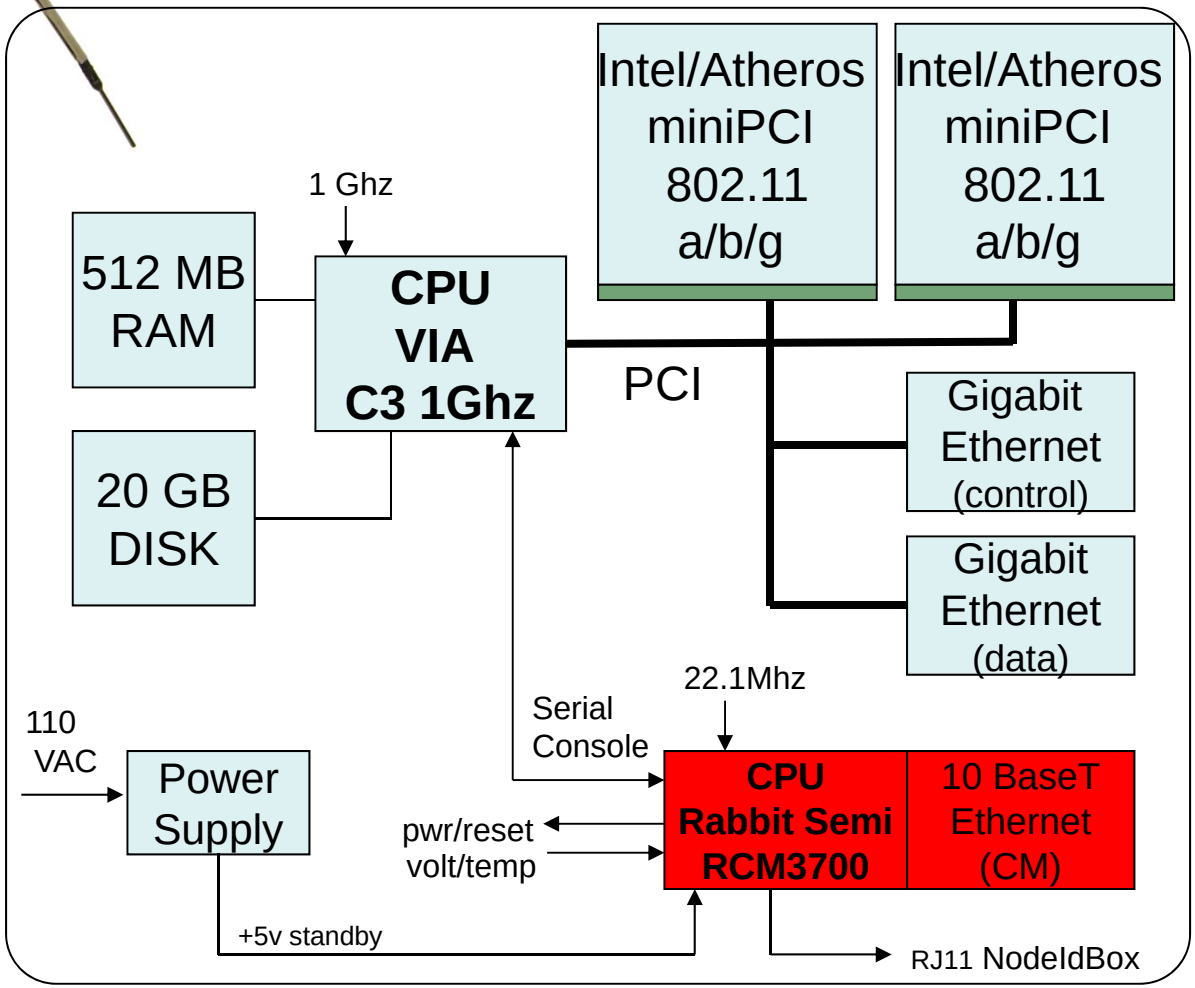
- Proof of concept
- Prototyping platform

Version 2: Custom design:

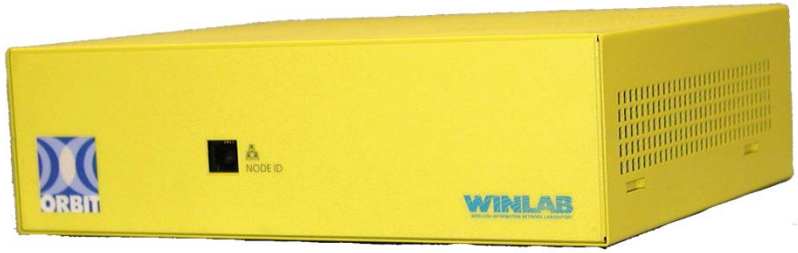
- Functional requirements
- Manageability
- Power consumption
- Cost

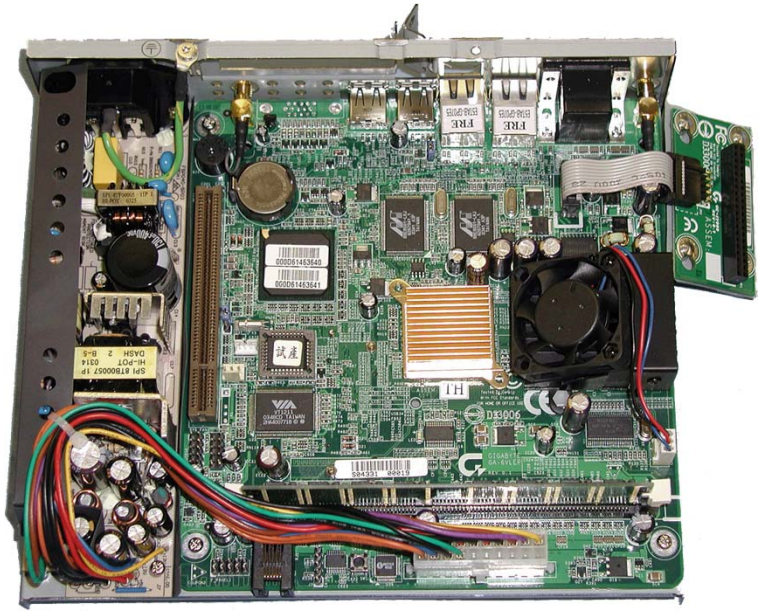
Other attached devices:

- Bluetooth
- ZigBee
- GNU Radio

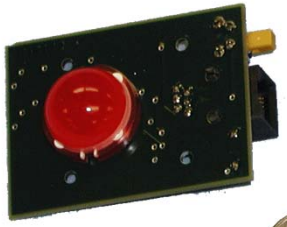


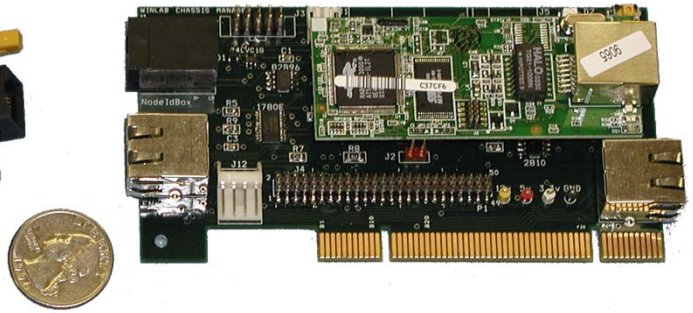
Radio Node Photo Album





ORBIT Radio Node
with integrated Chassis Manager



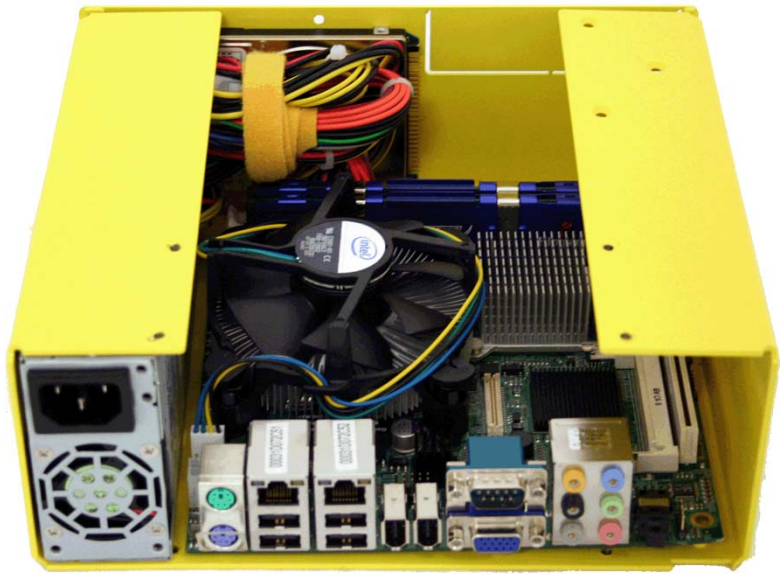


Non-Grid Node
Chassis Manager

ORBIT Radio Node (Version 3)

- Core 2 Quad with Q35 Express chipset
- 4 GB DDR2
- 2 x Gigabit Ethernet ports
- PCI-Express X16
- Mini-PCI socket
- 8 x USB 2.0
- 2 x COM

- Core 2 Duo with GM45 chipset
- 8 GB DDR3
- 2 x Gigabit Ethernet ports
- PCI-Express X16
- PCI Express mini socket
- Mini-PCI socket
- 8 x USB 2.0
- 2 x COM



ORBIT

Devices: 802.11 a/b/g

- Based on Atheros Dual Band Radio-on-a-Chip (5212)
- Dual-diversity with 0-18 dBm (1 dBm steps)
- PCI 2.3 and PC Card 7.1 host interfaces with DMA support
- Drivers: madwifi and ath5k

- Intel Dual Band 2915ABG
- Dual-diversity with -12-+20 dBm (1 dBm steps)
- Drivers: ipw2200

ORBIT

Devices: 802.11n and Bluetooth

- Belkin F8T003 and F8T010
- Bluetooth v1.1 compliant
- Range of 10m (100m)
- Driver: BlueZ

- Netgear WNDA3100
- Based on Atheros AR9170+AR9104
- 2x2 MIMO
- 6.5 - 300 Mbps
- Driver: ath9k

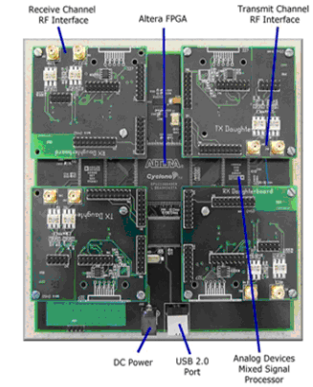
- D-LINK DWA-140
- Based on Ralink RT2870
- 2x2 MIMO
- 20/40 MHz support
- 6.5 - 300 Mbps
- Driver*: rt2x00

ORBIT

Devices: USRP/USRP2 with GNU Radio Platform

"Pentium" based SDR: Open-source GNU Radio Software - signal processing code on host computer in C++ (including FSK, PSK, AM, ASK, NBFM. WBFM, 802.11)

- IF 0-100 MHz (50 MHz transmit)
  - 128 MS/s DAC
  - 64 MS/s ADC
- USB bus (W = 8 MHz)
- Channelizer code in Altera Cyclone FPGA
- 2 RF board slots



- IF -200 MHz (80 MHz receive)
  - 100 MS/s 14-bit dual (IQ) ADCs
    400 MS/s 16-bit dual (IQ) DACs
- Gigabit Ethernet (W = 25 MHz)
- Bigger FPGA w/Multipliers (Xilinx Spartan 3) with 1 MB high-speed on-board SRAM and high speed serial expansion interface
- 1 RF board slot

Selection of RF daughtercards (DC-5.9 GHz): DC-30 MHz, 50-870 MHz (Rx only), 800-2400 MHz (Rx only), 400-500 MHz, 800-1000 MHz, 1150-1450 MHz, 1500-2100 MHz, 2300-2900 MHz, 2400-2500+4900-5840 and 50-2200 MHz

ZigBee Motes

Processor : Atmega (4MHz), MSP430 (8MHz)

Memory: 512 kb

Peripherals: Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller

Wireless Radio – CC2420 250kbps 2.4GHz IEEE 802.15.4 (ZigBee) Chipcon Wireless Transceiver (now Texas Instruments)

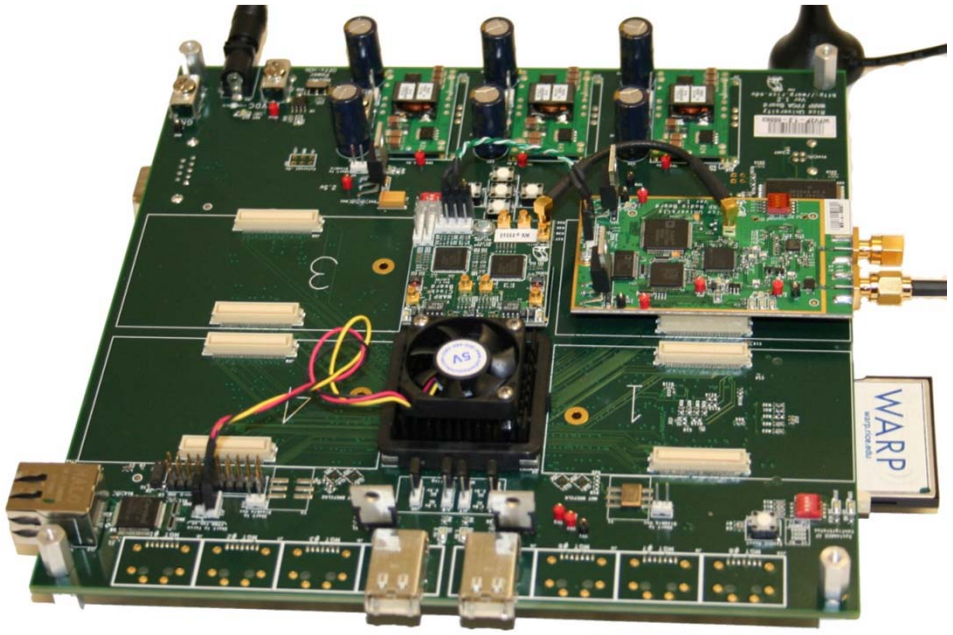
Sensors - Temperature, Light, Humidity

USB port - Programming and data collection via USB

ORBIT

WARP Platform (Rice University)

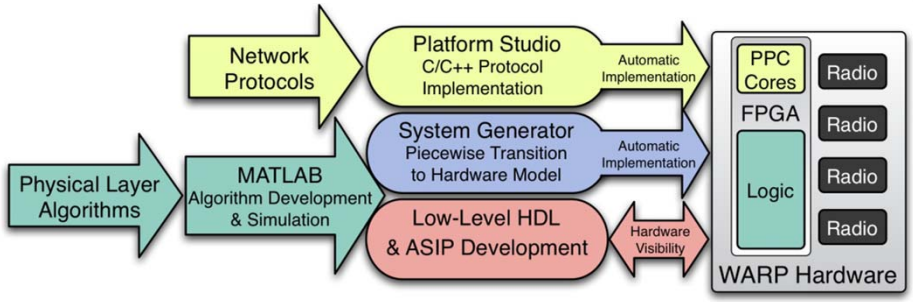
- Xilinx Virtex-II Pro (Xilinx XC2VP70) FPGA
- 10/100 Ethernet
- 4 Daughtercard Slots
- RS-232 UART
- 16-bit Digital I/O
- Radio dauthercard
  - 2 x 160MS/s 16-bit DAC
  - 2 x 65MS/s 14-bit dual-ADC
  - dual-band ISM/UNII RF (2400-2500MHz, 4900-5875MHz) - MIMO capable
  - 20 or 40MHz baseband bandwidth



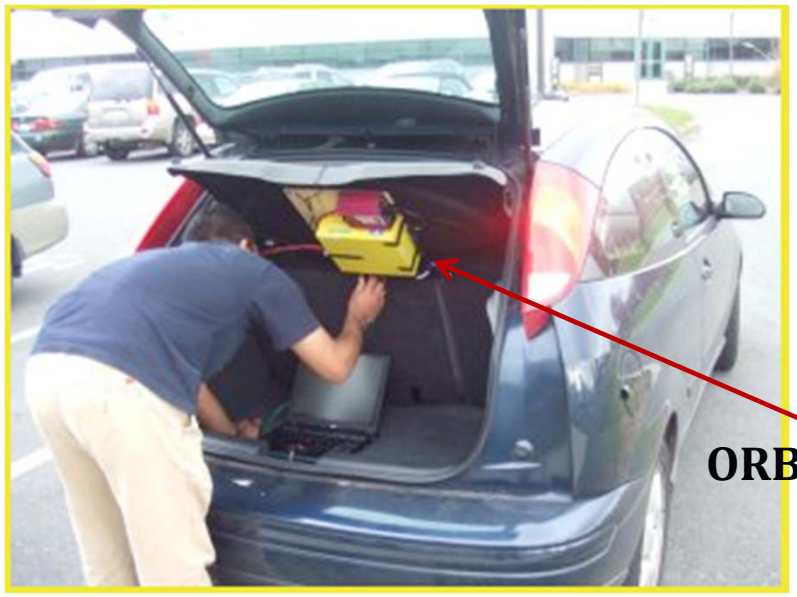
Design flows:

•Real-time – OFDM

•Non-real-time (interfaces for MATLAB) – SISO and MIMO

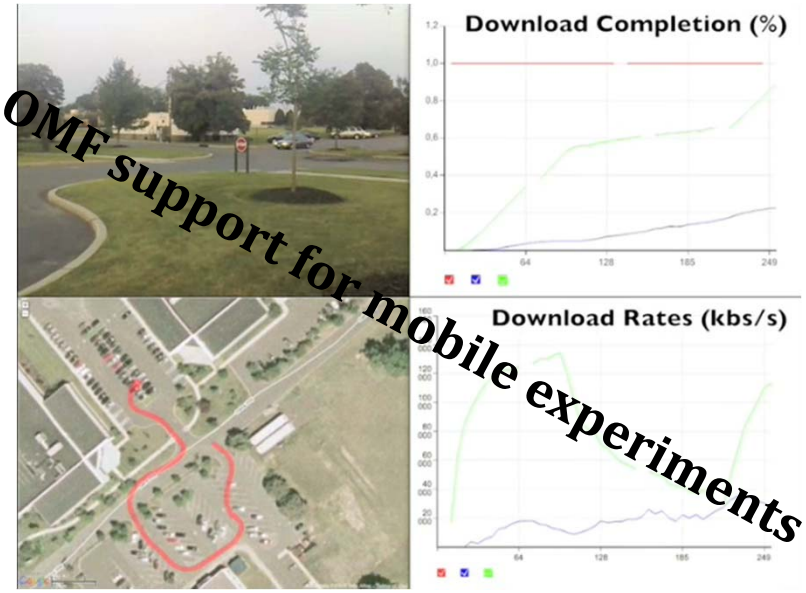


Mobile Platforms



ORBIT Node

Intel 5150/5350
mini-PCI express card for laptops with Linux driver

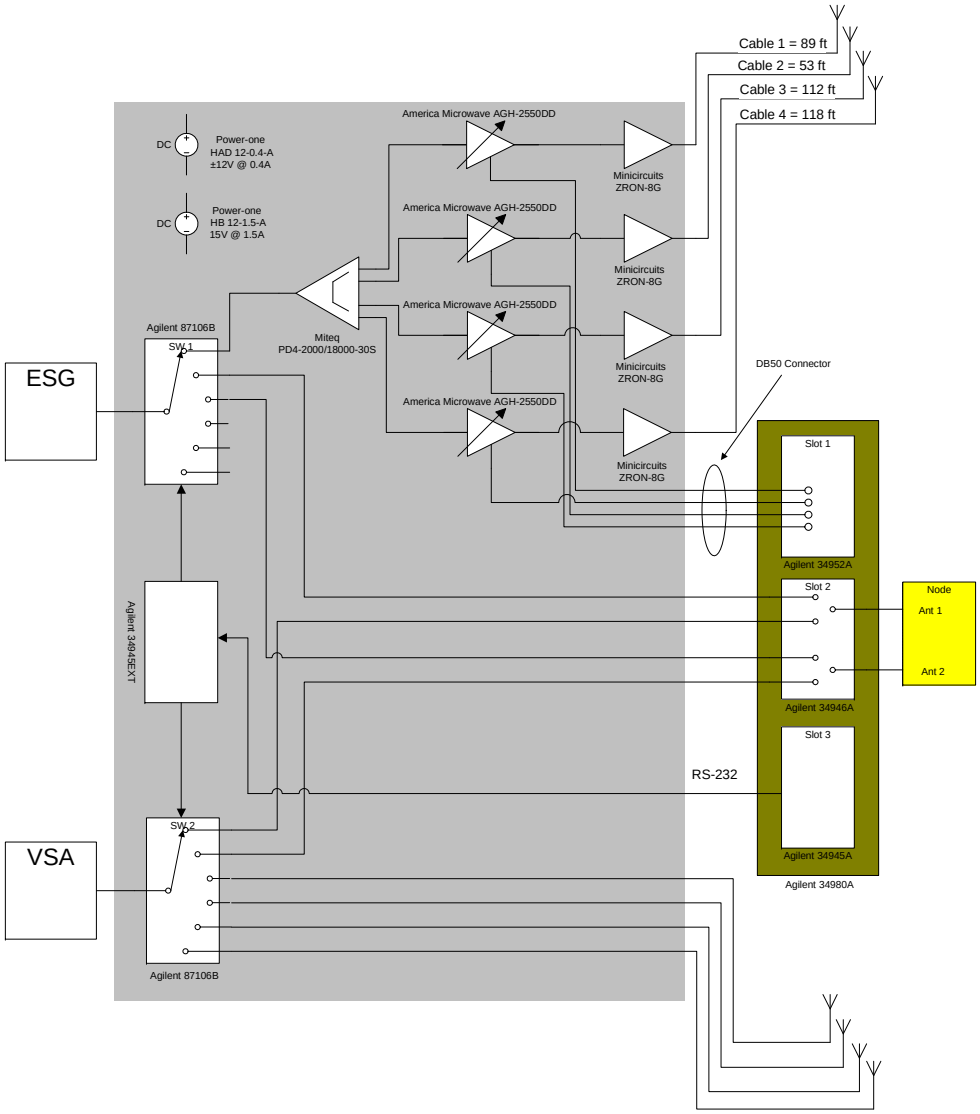


OMF support for mobile experiments

HTC EVO 4G
Android based portable platform

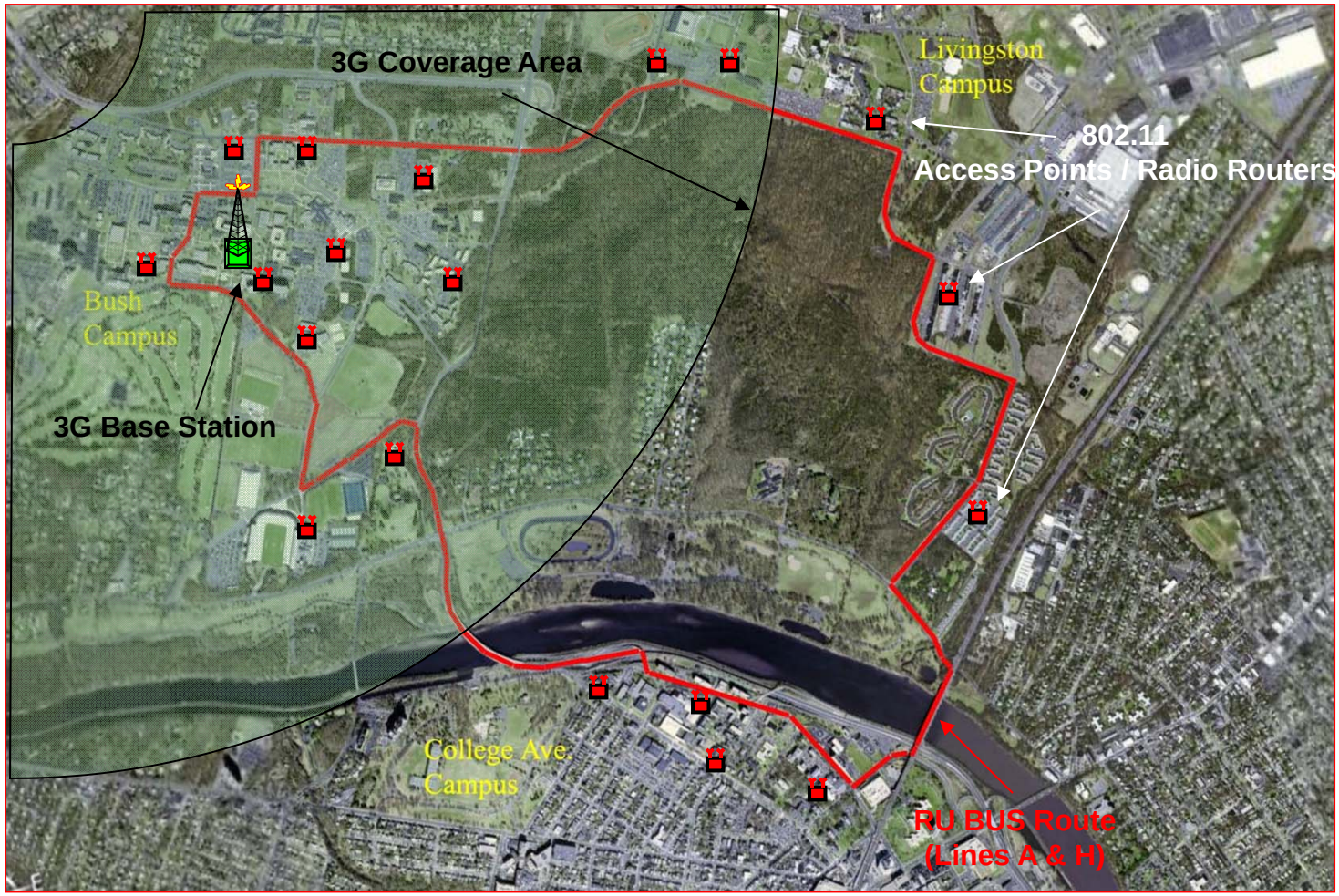
RF Interference grid

- Number of antennas providing spatial distribution of interference sources (BW = 40 MHz, f0=250 KHz – 6 GHz)
- Ideally @ each antenna, feed is a linear combination of 2-8 sources (cost issue)
- Variety of interference types *(W-CDMA, cdma2000, 1xEV-DO/DV, TD-SCDMA, cdmaOne, WiMax, GSM/EDGE, GPRS/EGPRS, 802.11, Bluetooth, GPS, enhanced multitone, NPR, AWGN, or up to 8 sec of arbitrary waveform generation)*
- Ability to observe actual RF signals



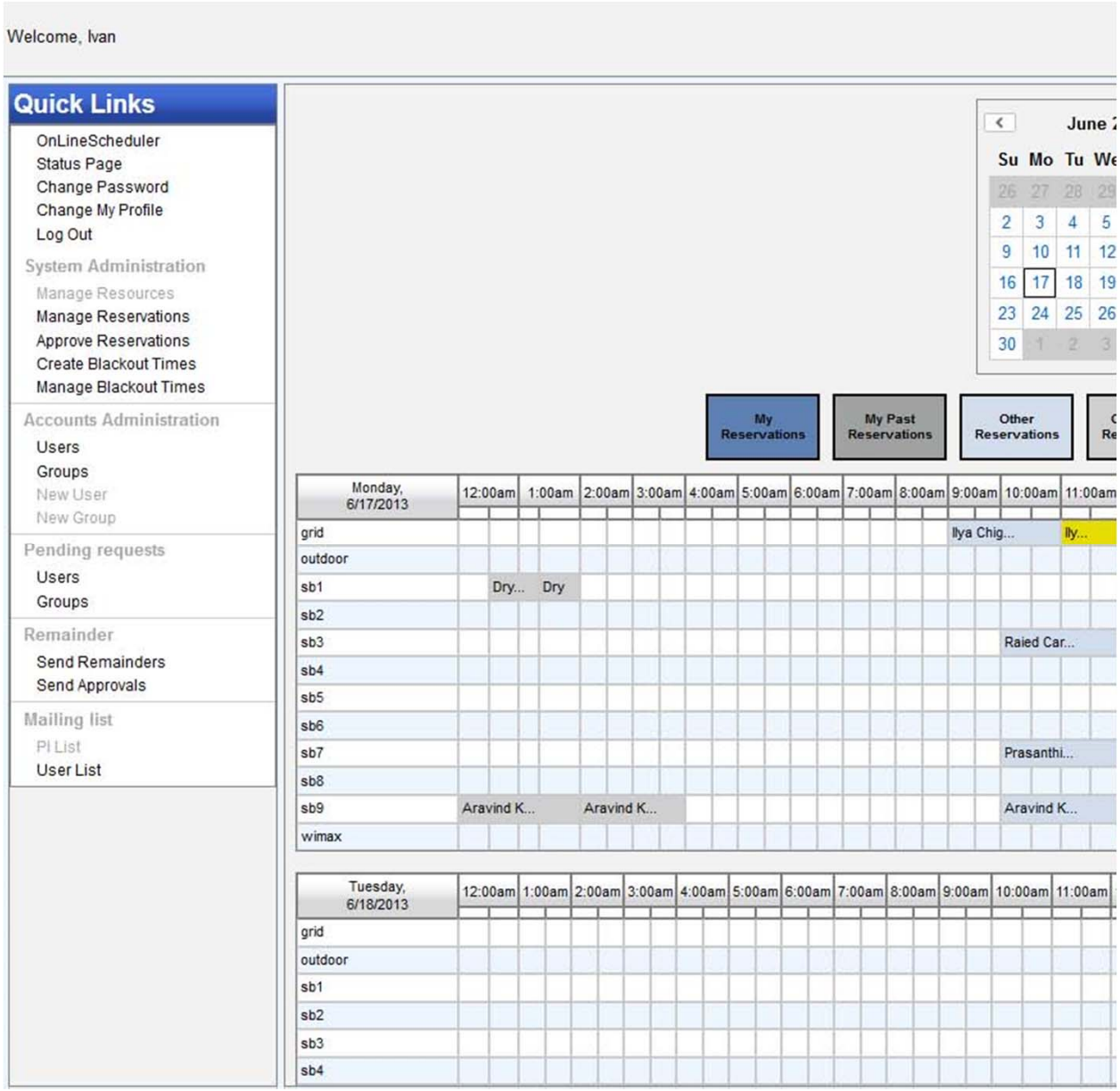
ORBIT Grid

ORBIT: Field Trial Plan (Phase II)

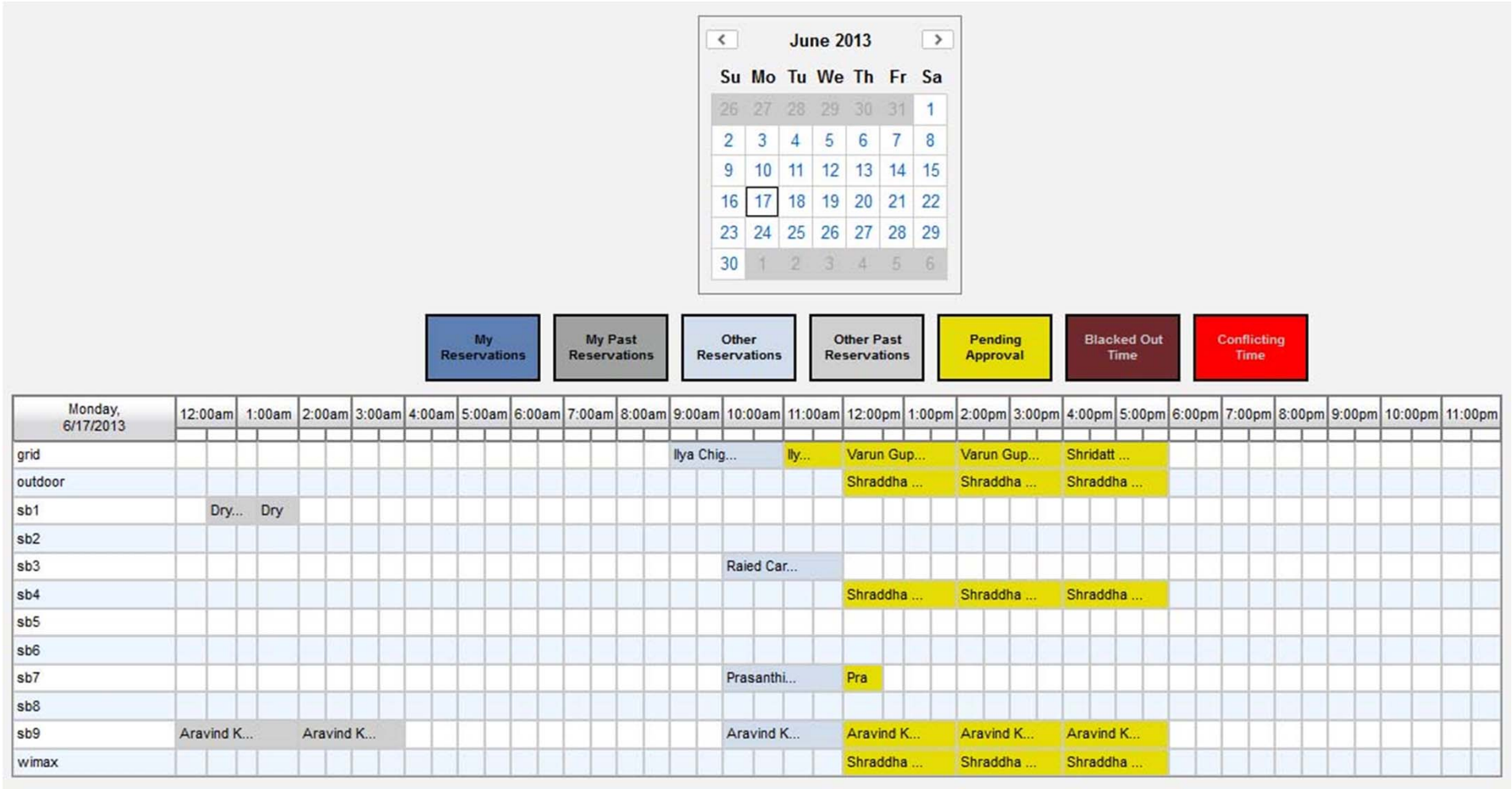


Software

Control Panel



Reservation System

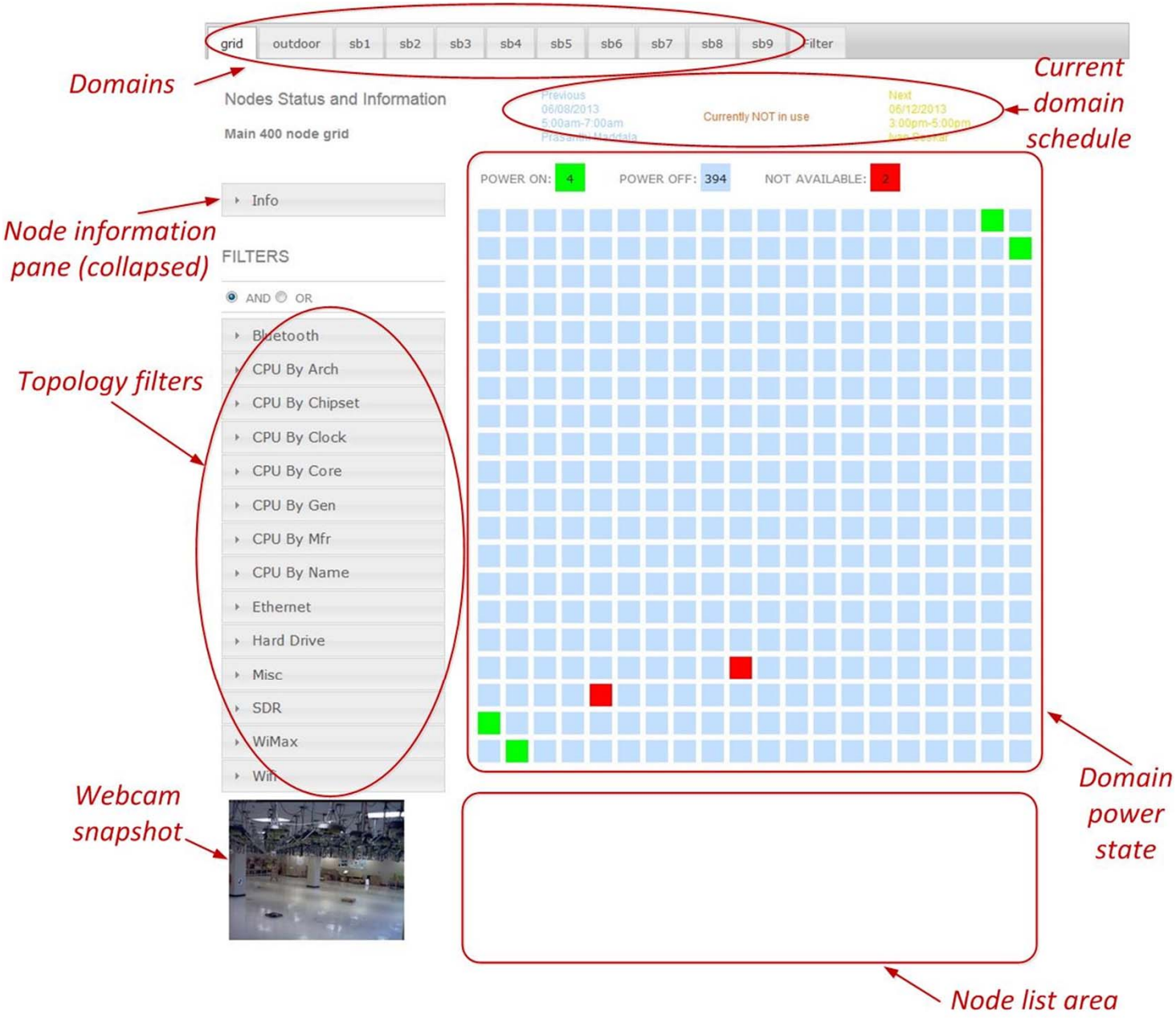


Auto-approval

- Two stage algorithm:
  - "Early bird" – runs once a day (at 2 PM) and resolves conflicts and approves first two hours for all users for the next day
    - (e.g if you ask for your first slot daily slot from 10-12 the next day , at 2 PM a day earlier you will know wheather you got it).
  - "Just in time" – for reservations made after 2 PM or for more than 2 hours per day per domain, the slots will be automatically approved at the beginning of the slot.
- Conflicts are resolved based on usage in the last two weeks
  - (the less you (ab)use it the more likely you are to get it ☺).
- Be aware of major conference deadlines

ORBIT

Status Page


grid
outdoor
sb1
sb2
sb3
sb4
sb5
sb6
sb7
sb8
sb9
Filter
Domains
Nodes Status and Information
Main 400 node grid
Previous
06/08/2013
5:00am-7:00am
Currently NOT in use
Next
06/12/2013
3:00pm-5:00pm
Current domain schedule
POWER ON: 4
POWER OFF: 394
NOT AVAILABLE: 2
Info
Node information pane (collapsed)
FILTERS
AND
OR
Topology filters
Bluetooth
CPU By Arch
CPU By Chipset
CPU By Clock
CPU By Core
CPU By Gen
CPU By Mfr
CPU By Name
Ethernet
Hard Drive
Misc
SDR
WiMax
Wifi
Webcam snapshot
Domain power state
Node list area

First Exercise: Scheduler, Login (ssh) and Status Page

ORBIT Management Framework

OMF is a framework to **use** and **manage** experimental platforms (testbeds)

Use

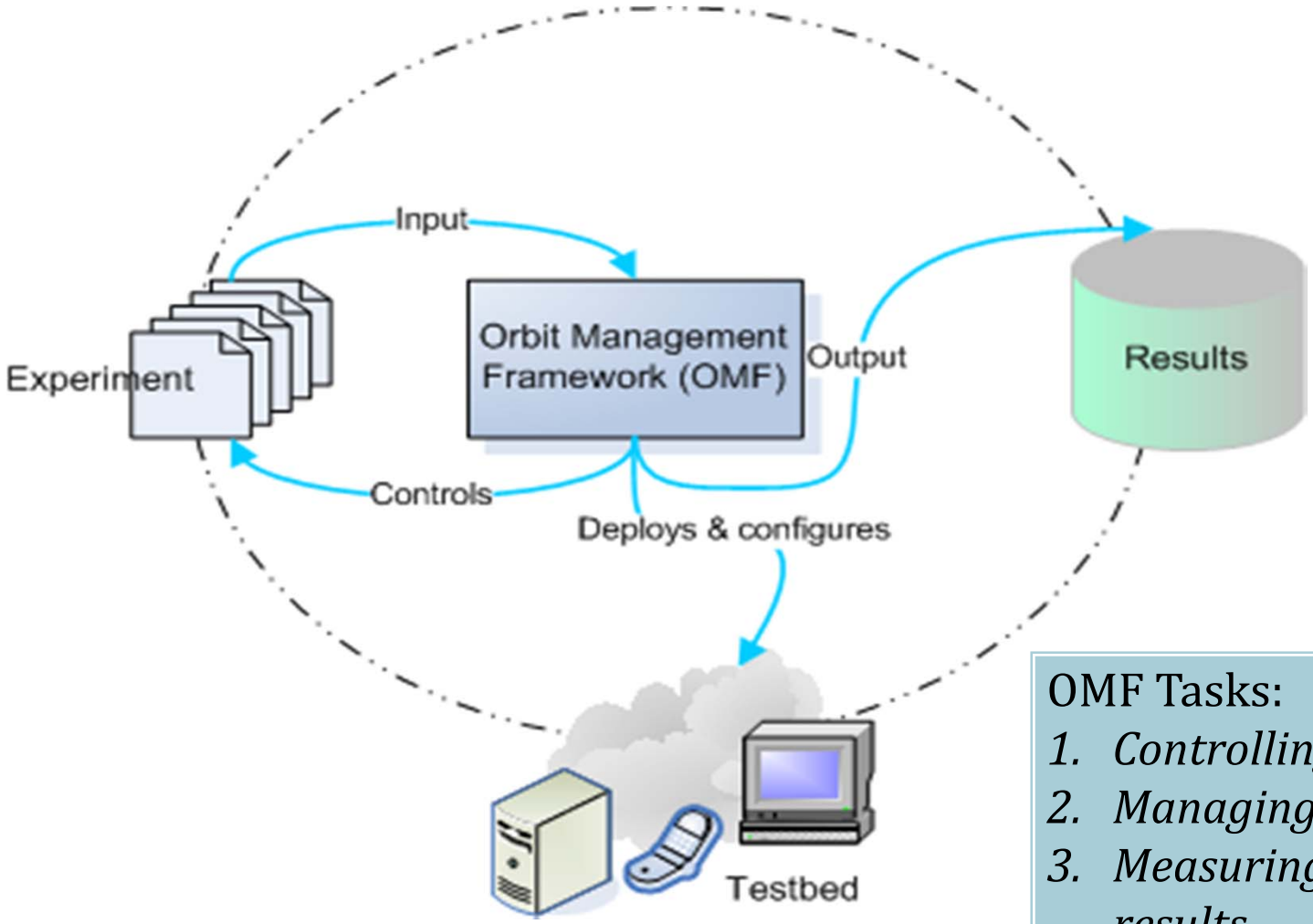
- support "*experiment cycles*" & scientific rigor
- validation, accuracy & reproducibility

Manage

- ease operation and maintenance tasks
- optimize resource utilization inside / across testbeds

Written mostly in Ruby

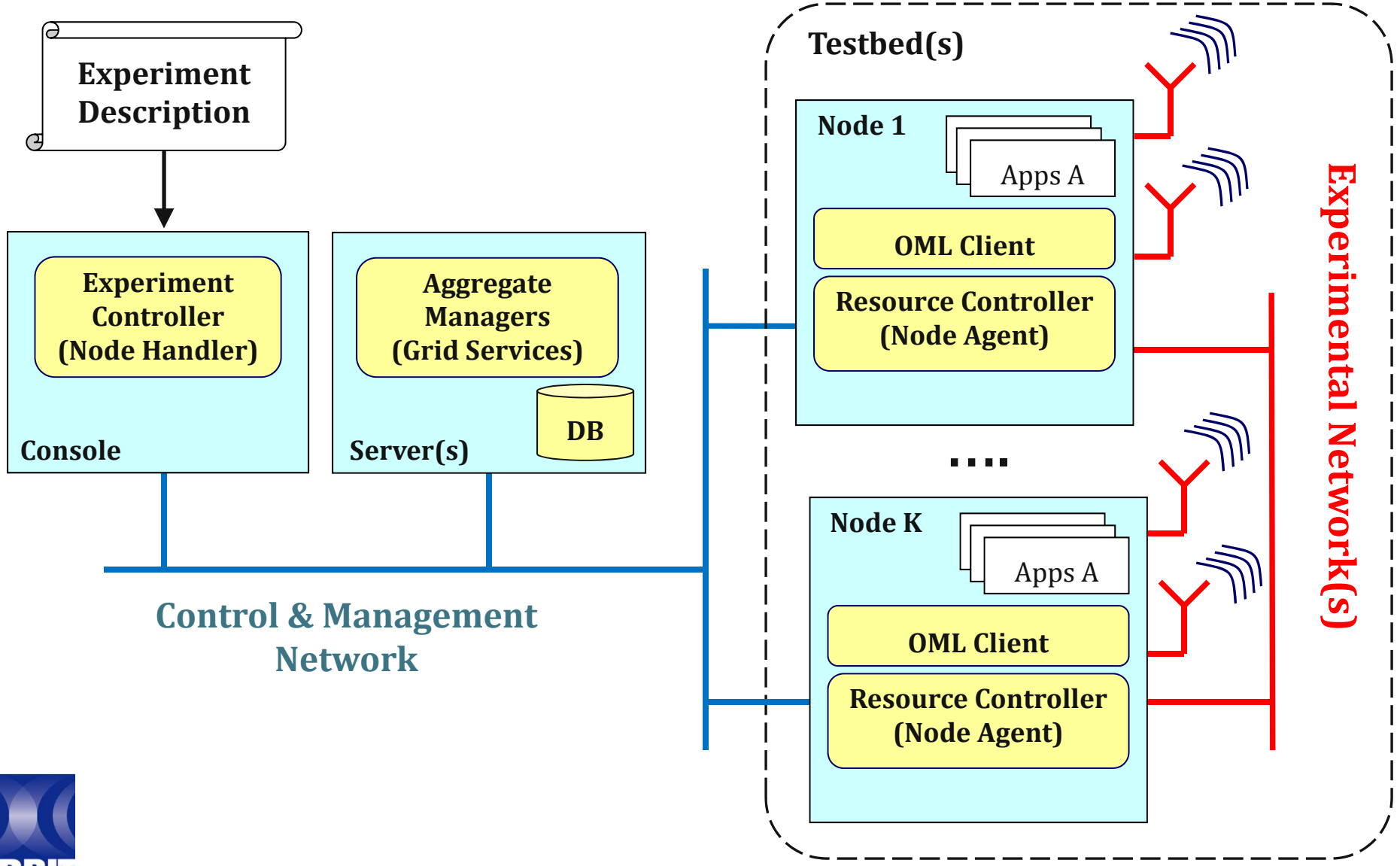
OMF Workflow



OMF Tasks:

1. *Controlling experiments*
2. *Managing the testbed*
3. *Measuring and collecting results*

OMF - Experimenter View



OMF Command

(aka “NodeHandler”)

omf [SUBCOMMAND] [ARGUMENT]...

Subcommand	***Description***
omf help	Display the help for using omf commands.
omf exec	Execute an experiment script.
omf load	Load a disk image on a given set of nodes.
omf save	Save a disk image from a given node into a file.
omf tell	Switch a given set of nodes ON/OFF.
omf stat	Returns the status of a given set of nodes

Second Exercise: Basic OMF commands

omf {tell, stat, load}

ORBIT Measurement Library

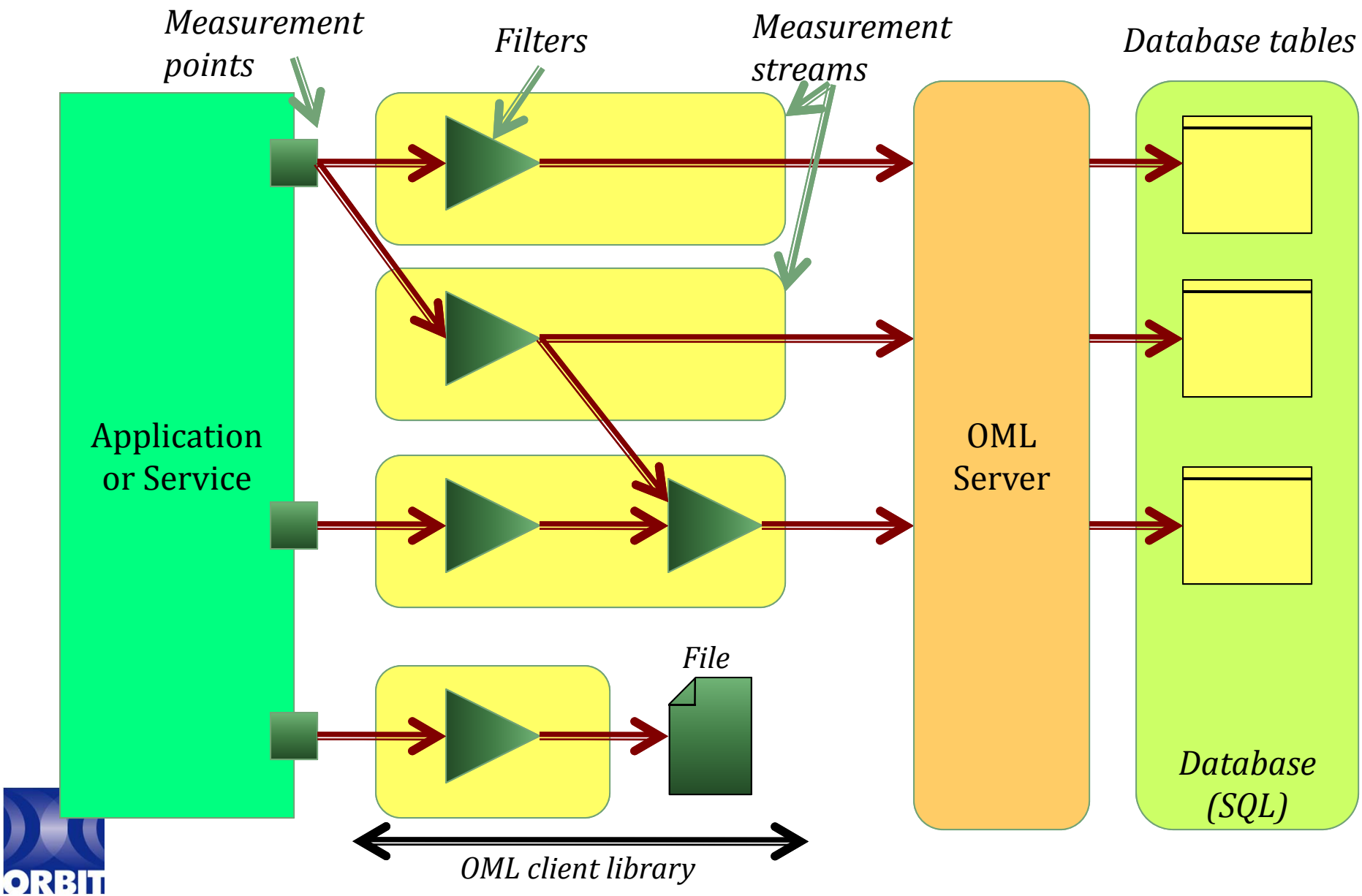
- Hardware with dedicated NIC for control/instrumentation
- Experimenters measure node, network & application performance (i.e. a lot of numbers).
- How to collect these numbers in real-time, in a distributed environment like ORBIT?

OML design goal: distributed software infrastructure to collect measurements in real-time while providing flexible (and dynamic) way to modify the collection process.

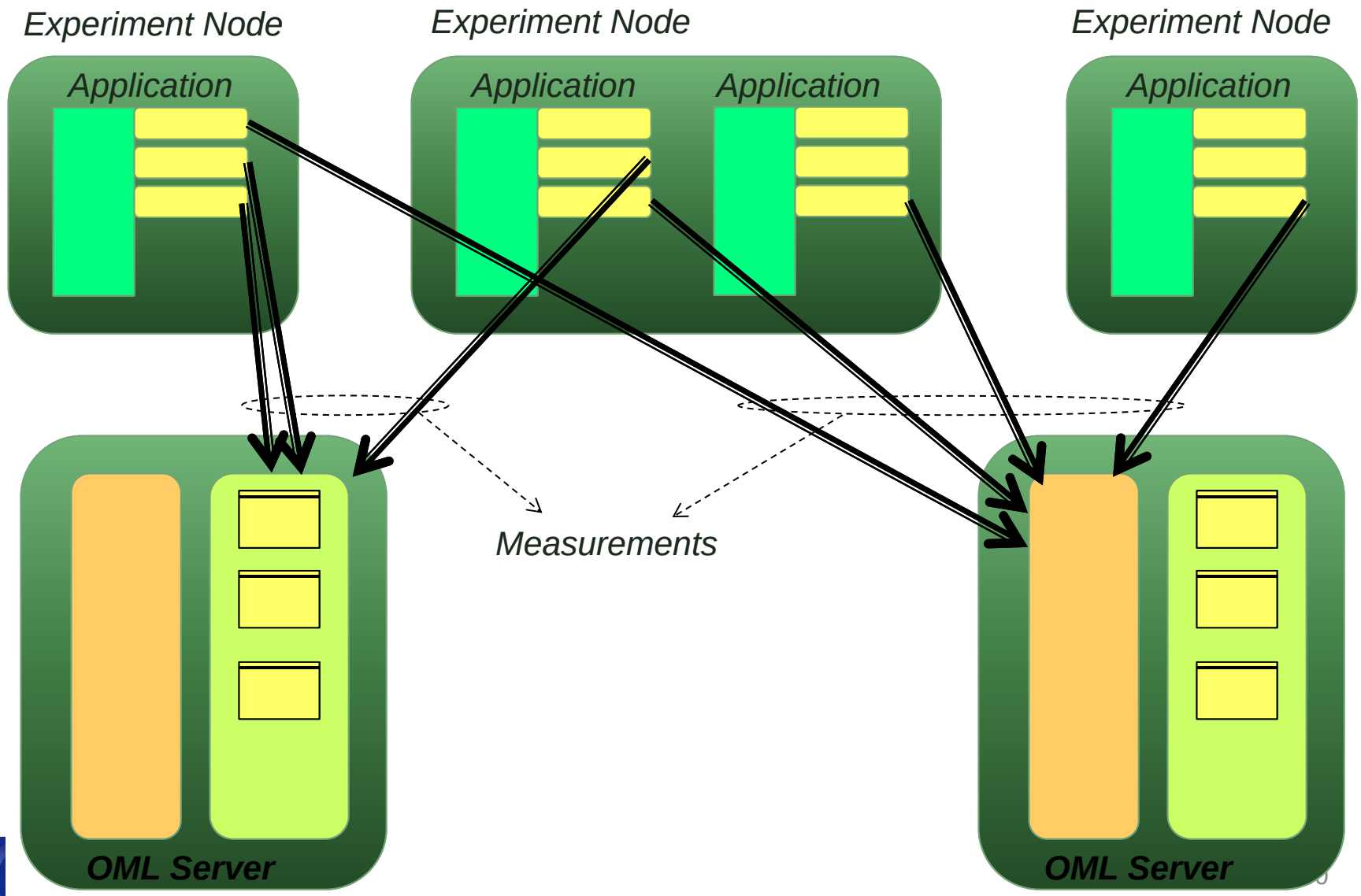
Orbit Measurement Library (OML)

- Push based architecture
- All experiment data in one place – including metadata
- Separation of concerns
  - Instrumentation
  - Collection
- Minimize collection overhead
  - Application CPU time
  - Experimental traffic interference
- Proxy server for disconnected operation

OML Client + Server



OML – Measurement Collection



Experimental Support

Applications

- Traffic Generation/Measurements
  - OTG … Traffic Generator
  - Iperf
- Monitoring
  - Libtrace
  - Libsigar
  - Spectrum Analyzer
  - GPS
  - (Weather)
- Components
  - TinyOS/Motes
  - (GnuRadio)

Filters

- Plug-in Architecture
- User extensibility

Current List

- ✓ Stddev
- ✓ Average
- ✓ First
- ✓ Histogram

ORBIT

OML2 Functions

- **omlc_init()** – used for OML initialization

  omlc_init(arg(0), &argc, argv, o_log);

- **oml_add_mp()** – called to register each measurement point with the container

  oml_mp = omlc_add_mp("udp_out", oml_def);

- **omlc_start()** - used to start the local collection daemon

- **omlc_process()** – used to specify the stages at which to collect the measurement points

  omlc_process(oml_mp, v);

OMF Experiment Description Language (OEDL)

- Domain-specific Language based on Ruby
- Two parts of experiment description (ED):
  - **Resource requirements and configuration**: specifies experimental resources
  - **Task description**: *state-machine* that enumerates tasks to perform
- At the moment nodes are identified by their 2D "location" (coordinates in the grid)

OEDL Commands

8 groups:

- Top-level commands
- Topology-specific commands
- Group-specific commands
- Prototype-specific commands
- Application-specific commands
- Execution-specific commands
- Resource Paths
- Testbed-specific commands

OEDL Top-level Commands: defProperty

defProperty(name, initialValue, description)

- **name**: name of the property. This name will be used to refer to this property in any consecutive OEDL commands.
- **initialValue**: the initial value of the property. This also determines the type of the property.
- **description**: Textual description. Used in Experiment Controller's help message, as well as for the default web interface.

Usage:
defProperty('rate', 300, 'Bits per second sent from sender')
defProperty('packetSize', 1024, 'Size of packets sent from sender')

OEDL Top-level Commands: prop

```
prop.propName
prop.propName = newValue
```

- propName: Name of experiment property.
- newValue: New value to assign to the property.

Usage:

```
defProperty('rate', 300, 'Bits per second sent from sender') ...
'rate' => prop.rate
...
[500, 1000, 2000].each { |newRate|
prop.rate = newRate 14
}
```

ORBIT

OEDL Top-level Commands: logging

```
debug(arg1, ...)
info(arg1, ...)
warn(arg1, ...)
error(arg1, ...)
```

- **arg1**: None or more strings to be logged

Usage:

```
info("Starting")
debug(i, " resource(s) are up")
```

Note: DEBUG and INFO log normal progress and can be ignored, while WARNING and ERROR report on abnormal behavior.

OEDL Top-level Commands: wait

wait(time)

- **time**: pause experiment execution for time seconds

Usage:

```
whenAllInstalled {
...
   [500, 1000, 2000].each { |newRate|
      prop.rate = newRate
      wait 30
   }
}
```

OEDL Topology Commands: defTopology

Used to specify topology consisting of a set of nodes and links each with certain characteristics

```
defTopology( name , arrayOfNodes = nil , &block = nil )
```

- **name**: Name of the defined topology.
- **arrayOfNodes**: (optional) array of resources (e.g. nodes) to include in this topology.
  - the list of valid definition patterns are:
    - [x,y]: Describes a single node at location x@y
    - [x1..x2, y]: Describes a set of nodes along a line starting at x1@y and ending at x2@y. For instance, [2..4, 5] defines the nodes [2,5], [3,5], [4,5].
    - [x, y1..y2]: Same as previous, but for the y coordinate.
    - [x1..x2, y1..y2]: This defines a rectangle area of nodes within the grid.
    - [[x1,y1], [x2,y2], [x3,y3]]: An arbitrary long list of single nodes.
- **block**: (optional) a block of commands that can be used to build/configure this topology.

ORBIT

OEDL Topology Commands: defTopology (cont'd)

Topology Sub-Commands	*Description*
addNode(x,y)	Add node at location x@y to the topology.
removeNode(x,y)	Remove node at location x@y from the topology.
addLink (x, y, spec)	Adds a link between nodes x and y and configures it with the characteristics defined in the 'spec'.'spec' is a hash with the following valid keys {:rate , :per, :delay, :asymmetric}
RemoveLink (x, y)	Severs the link between nodes x and y.
size()	Return the number of nodes in this topology.
getNode(index)	Return the node at the position index in this topology. Return nil if index is greater than the number of nodes in the topology.
getFirstNode()	Return the node at the 1st position in this topology.
getLastNode()	Return the node at the last position in this topology.
getRandomNode()	Return a random node from this topology.
getUniqueRandomNode()	Return a unique random node from this topology. When all the available nodes in this topology have been drawn, this method will return nil and output a warning message to the console.
eachNode(&block)	Execute the commands in block on each node within this topology.
setStrict()	Set the strict flag for this topology. By default, the strict flag is NOT set for a topology.
unsetStrict()	Clear the "strict" flag. By default, the strict flag is NOT set for a topology.
hasNode(x, y)	Return true if the node at location x@y is part of this topology, return false otherwise.

OEDL Topology Commands: defTopology (cont'd)

```
defTopology('test:topo:circle') { |t|
    nodeNum = 8
    xCenter = 10
    yCenter = 10
   radius = nodeNum
   # use simple 4-way algorithm to pick the nodes
   r2 = radius * radius
   t.addNode(xCenter, yCenter + radius)
   t.addNode(xCenter, yCenter - radius)
   (1..radius).each { |x|
       y = (Math.sqrt(r2 - x*x) + 0.5).to_i
       t.addNode(xCenter + x, yCenter + y)
       t.addNode(xCenter + x, yCenter - y)
       t.addNode(xCenter - x, yCenter + y)
       t.addNode(xCenter - x, yCenter - y)
   }
}
```

OEDL Group Commands: defGroup

defGroup(groupName, selector, &block = nil)

- **groupName**: name of the defined set of resources
- **selector**: selects the resources to be contained in this set. Group selector can be also defined with topology URI (i.e. set of nodes that form the topology)
- **block**: instructions for all resources in the group

Usage:

```
defGroup('sender1', [1, 1])      # set contains 1 resource
defGroup('sender2', [2, 1..8]) # set contains  8 resources [2,1], [2,2], ... [2,8]
defGroup('sender', ['sender1', 'sender2', [3, 1..8]]) {|node|
   node.prototype("test:proto:sender", {
      'destinationHost' => '192.168.1.1',
      ...
   }
   node.net.w0.mode = "master" #802.11 Master Mode
}
```

ORBIT

OEDL Group Commands: defGroup (cont'd)

addApplication	Install an application on a node
exec	Execute a command on all nodes in this group.
image	Check whether a node boots in the required image. (not available in version 4.4 of the NH)
netmask	This is the network mask resource path.
onNodeUp	Execute a block of commands when a node is up.
pxeImage(...)	Instructs a resource to boot from a network PXE image (recommended for expert users only).

OEDL Group Commands: group and allGroups

```
group(groupSelector).command()
group(groupSelector).resource_path = value
group(groupSelector).resource_path {...}
```

- **groupSelector:** set of resources to use.
- **command**: command to run for that set.
- **resource_path:** is the parameter to be set
- **value:** is the value to assign to the resource path parameter

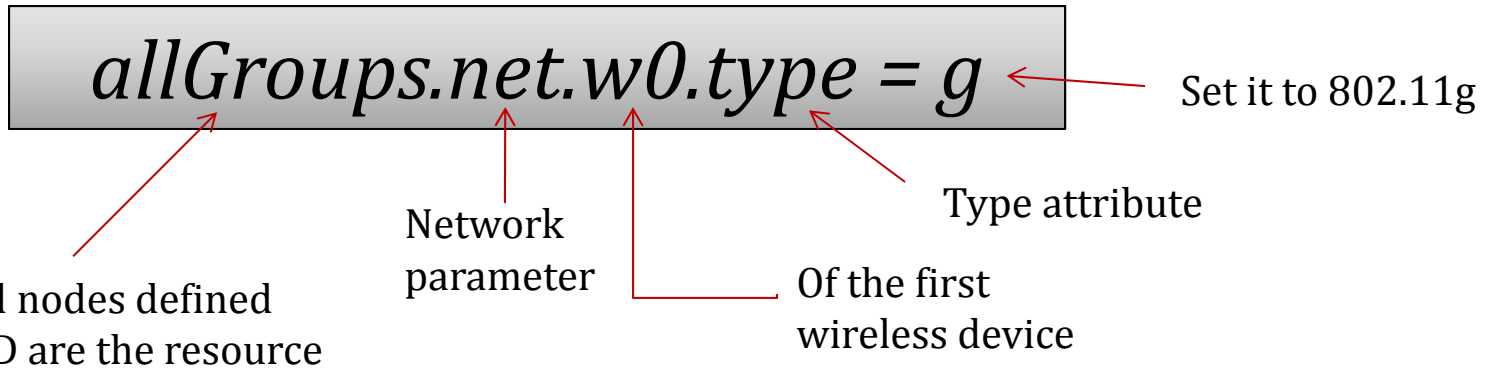
Usage:

```
group('sender1').startApplications
group(['s1', 'r1']).net.w0.essid = "orbit"
allGroups.net.w0 { |w|
   w.essid = "orbit“
}
```

Resource Paths

- A resource path allows the access and the value assignment of a specific configuration parameter of a resource
- **Can be** used in any section of the ED.
- Follow a hierarchical organization:

  <resource_selector>.<hierarchical_path>



net - network resource path

- {e0, e1} Ethernet interface
  - arp = true|false En/disable ARP
  - forward = true|false Enable forwarding
  - ip = address/netmask IP address of interface
  - up = true|false En/disable interface
- {w0, w1} Wireless interface
  - All the above
  - channel (intel only) = 1..11; 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161
  - frequency (intel only) = 2.412..2.462GHz (5 Mhz steps); 5.18GHz (20Mhz steps)
  - essid = arbitrary string
  - mode = master|managed|monitor, ad-hoc (intel only)
  - rts (atheros only) = packetSizeThreshold [bytes]
  - rate (intel only) = 1, 5, 11; 6, 9, 12, 18, 24, 36, 48, 54
  - tx_power = -12..15 dBm (intel), 0..20 dBm (atheros)
  - type = a/b/g

OEDL Testbed Commands (ORBIT): antenna

This command enables you to inject arbitrary waveform into the grid

```
antenna(selector).signal.resource = value
antenna(selector).signal {}
```

- **selector**: expression to select the antenna set.
- **signal**: path to noise generator.
- **resource**: antenna parameters that can be configured are:
  - bandwidth: Bandwidth of signal
  - channel: 802.11 channel
  - power: output power in dbM
  - on: on when set to true and off when set to false.
  - value: value to assign to parameter.

Usage:

```
antenna(4,4).signal {|s|
   s.bandwidth = 20
   s.channel = 36
   s.power = prop.noisePower
}
```

ORBIT

hello-world-wireless.rb

```
defProperty('res1', 'node1-1.grid.orbit-lab.org', "ID of sender node")
defProperty('res2', 'node1-2.grid.orbit-lab.org', "ID of receiver node")
defProperty('duration', 60, "Duration of the experiment")

defGroup('Sender', property.res1) do |node|
  node.addApplication("test:app:otg2") do |app|
    app.setProperty('udp:local_host', '192.168.0.2')
    app.setProperty('udp:dst_host', '192.168.0.3')
    app.setProperty('udp:dst_port', 3000)
    app.measure('udp_out', :samples => 1)
  end
  node.net.w0.mode = "adhoc"
  node.net.w0.type = 'g'
  node.net.w0.channel = "6"
  node.net.w0.essid = "helloworld"
  node.net.w0.ip = "192.168.0.2"
end

defGroup('Receiver', property.res2) do |node|
  node.addApplication("test:app:otr2") do |app|
    app.setProperty('udp:local_host', '192.168.0.3')
    app.setProperty('udp:local_port', 3000)
    app.measure('udp_in', :samples => 1)
  end
  node.net.w0.mode = "adhoc"
  node.net.w0.type = 'g'
  node.net.w0.channel = "6"
  node.net.w0.essid = "helloworld"
  node.net.w0.ip = "192.168.0.3"
end

onEvent(:ALL_UP_AND_INSTALLED) do |event|
  info "This is my first OMF experiment"
  wait 10
  allGroups.startApplications
  info "All my Applications are started now..."
  wait property.duration
  allGroups.stopApplications
  info "All my Applications are stopped now."
  Experiment.done
end
```

Third Exercise: “Hello World” (wireless)

omf exec test:exp:tutorial:hello-world-wireless -- --res1 node1-1.sb1.orbit-lab.org --res2 node1-2.sb1.orbit-lab.org

IEEE 802.16

(WiMax, WiBro)

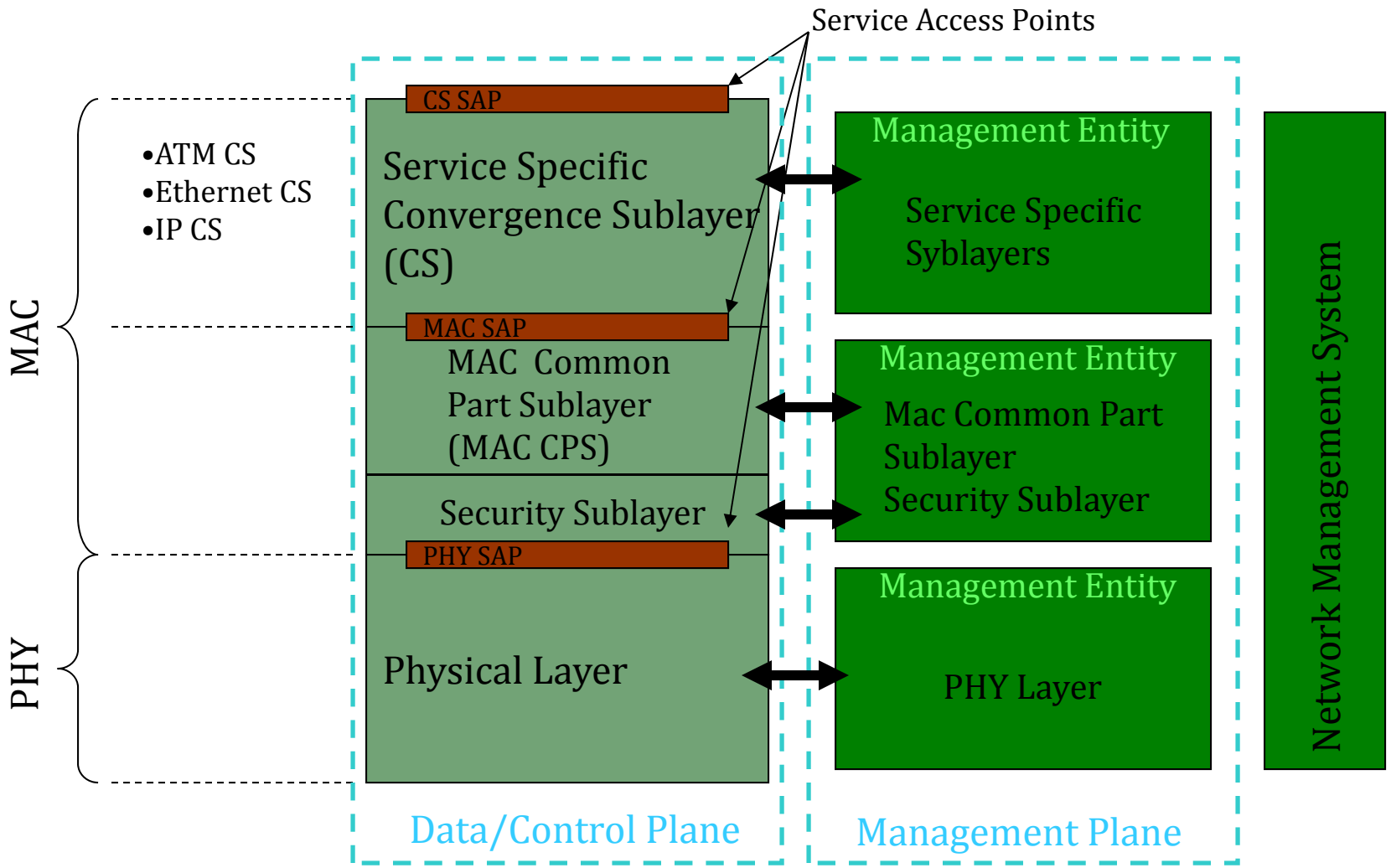
802.16 Key Features

- Common MAC with multiple (subscriber) adaptive PHY technologies
- Support for both TDD and FDD
- Modular design
- Multiple network topoplogies: both sigle-hop and multi-hop (PtP, PmP and Mesh)
- Multiple antenna technologies: omni, directional/sectorized, adaptive and MIMO
- Multiple encapsulated protocol payloads
- Flexible transmission policies (ARQ)
- Integrated QoS

802.16 Standards

- 802.16-2004
  - 802.16-2001/802.16REVd
  - 802.16a/b (2-11 Ghz, Mesh, non-line-of-sigth)
  - 802.16c (detailed system profiles)
- 802.16e: Mobile Wireless MAN
- 802.16f: Management Information Base
- 802.16g: Management Plane Procedures and Services
- 802.16h: Improved Coexistence Mechanisms for License-Exempt Operation
- 802.16i: Mobile Management Information Base
- 802.16j: Mobile Multihop Relay
- 802.16/Conformance0X: Conformance Test Standards
- 802.16.2: Licensed Coexistence
  - 802.16.2-2001 – WirelessMAN-SC
  - 802.16.2-2004 – exanded for 16a frequencies

IEEE 802.16 Reference Model



802.16 PHY Parameters

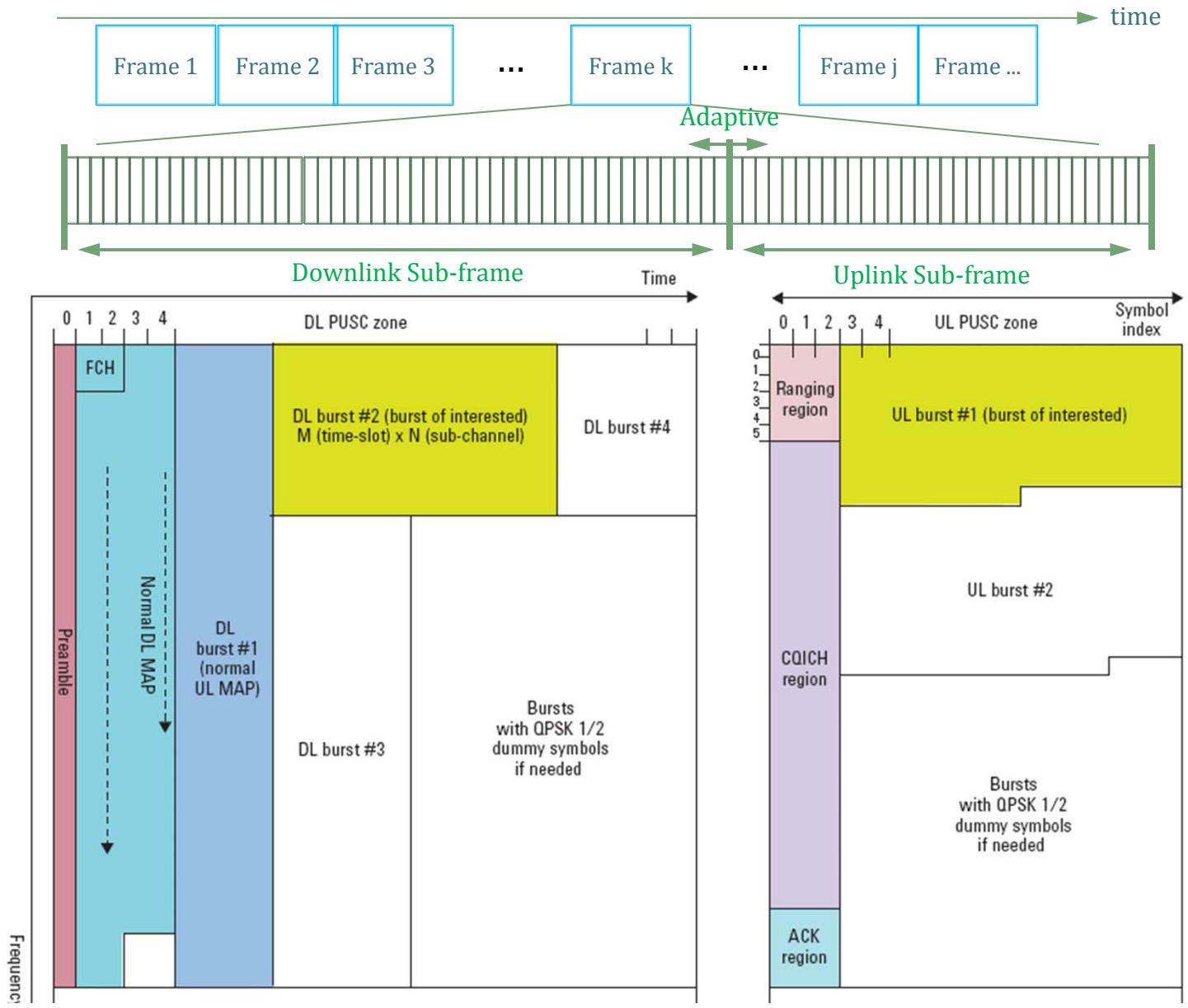
Parameters	Values		
Bandwidth (MHz)	7	5	10
Sampling Frequency (MHz)	8	5.6	11.2
FFT Size	1024	512	1024
Subcarrier spacing (kHz)	7.81	10.93	10.93
Symbol time (µs)	128	91.4	91.4
Guard time (µs)	16	11.4	11.4
OFDMA symbol time (µs)	144	102.8	102.8

BW	7		5		10	
Path	DL	UL	DL	UL	DL	UL
Used subcarriers	840	840	420	408	840	840
Data subcarriers	720	560	360	272	720	560
Sub-channels	30	35	15	17	30	35

802.16 MAC

- Connection orienteded
  - Connection ID (CID), Service Flows(SF)
- Channel access:
  - UL-MAP - defines uplink channel access and data burst profiles
  - DL-MAP - defines downlink data burst profiles
  - UL-MAP and DL-MAP are both transmitted in the beginning of each downlink subframe (FDD and TDD).
- CS hides details of payload protocol from MAC by:
  - Mapping payload protocol PDU to MAC service flow through set of configured rules - classifiers
  - Compressing redundant payload protocol headers
  - Delivering MSDU to MAC

OFDMA Frame Structure



802.16 Bandwidth requests and allocations

- Requests
  - Using ”contention request opportunities” as a response to multicast or broadcast poll
  - By sending ”BW request” in an allready granted slot
  - Piggyback a BW request message on a data packet
- Two modes of allocations:
    - Grant Per Subscriber Station (GPSS)
    - Grant Per Connection (GPC)
- Decision based on requested bandwidth and QoS requirements vs available resources and are realized through the UL-MAP.

QoS Classes

SF Class	Target Traffic	Parameters
UGS	CBR real-time	maximum sustained rate, maximum latency and tolerated jitter
ertPS	VoIP with silence suppression	maximum sustained rate, maximum latency and tolerated jitter
rtPS	Variable bit rate real-time	minimum reserved traffic rate, maximum sustained traffic rate
nrtPS	Non-real-time VBR traffic with no delay guarantee	minimum rate
BE	Everything else	minimum reserved traffic rate, maximum sustained traffic rate (not guaranteed)

ORBIT

802.16 Scheduling

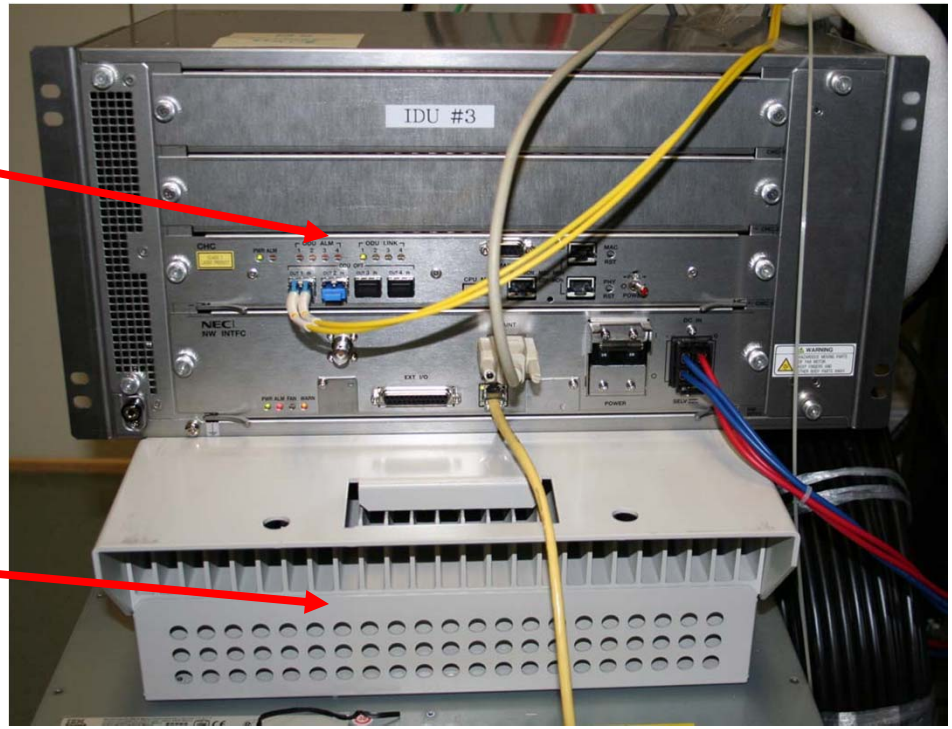
- Downlink scheduling is simple – queues in BS
- Uplink scheduling
  - Queues are distributed among SSs.
  - Queue states and QoS requirements are obtained through requests.
- Algorithms not defined in standard

NEC WiMax Hardware

Roof mounted Antenna

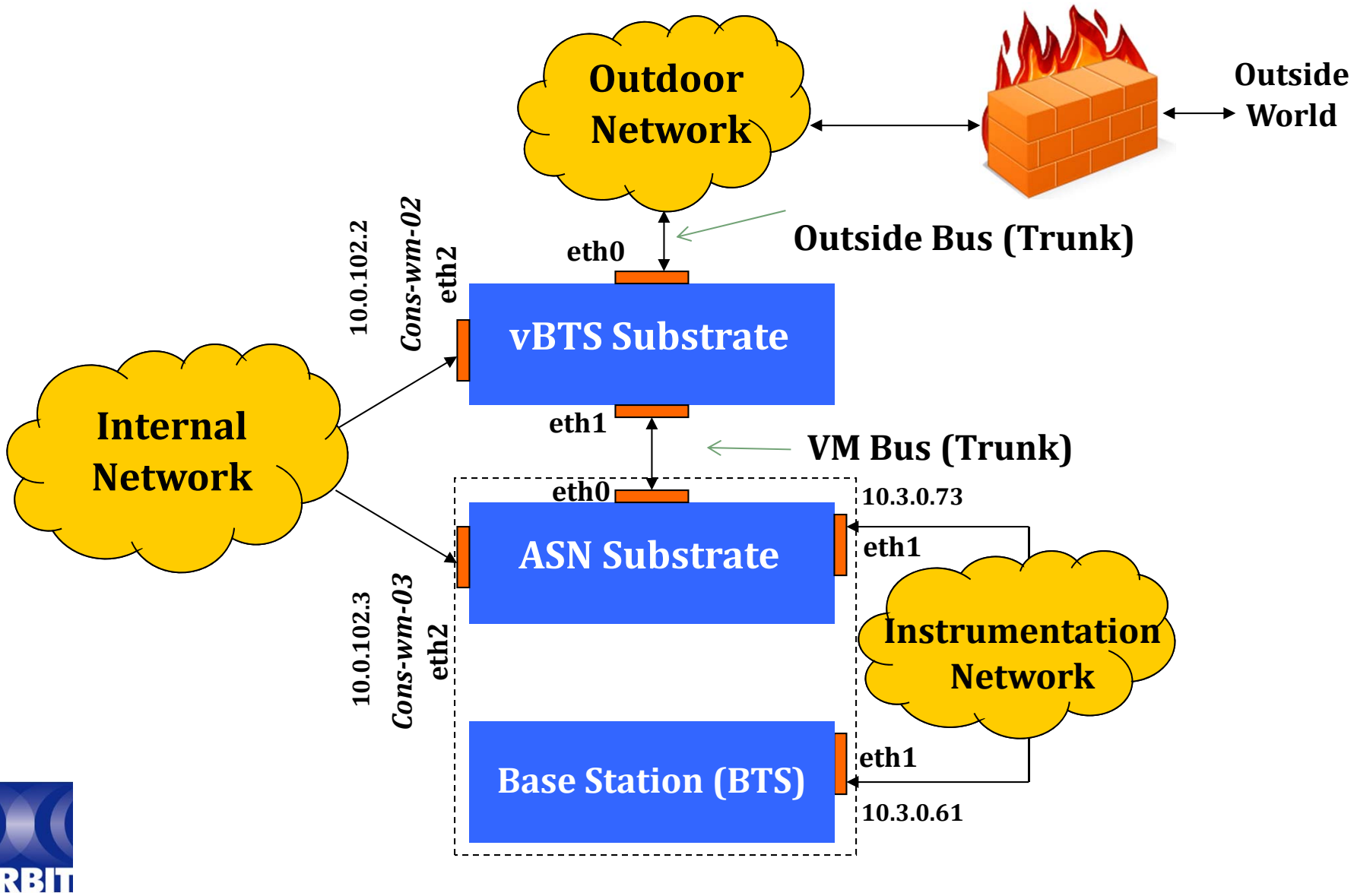
Basestation (IDU) Unit

RF (ODU) Amplifier

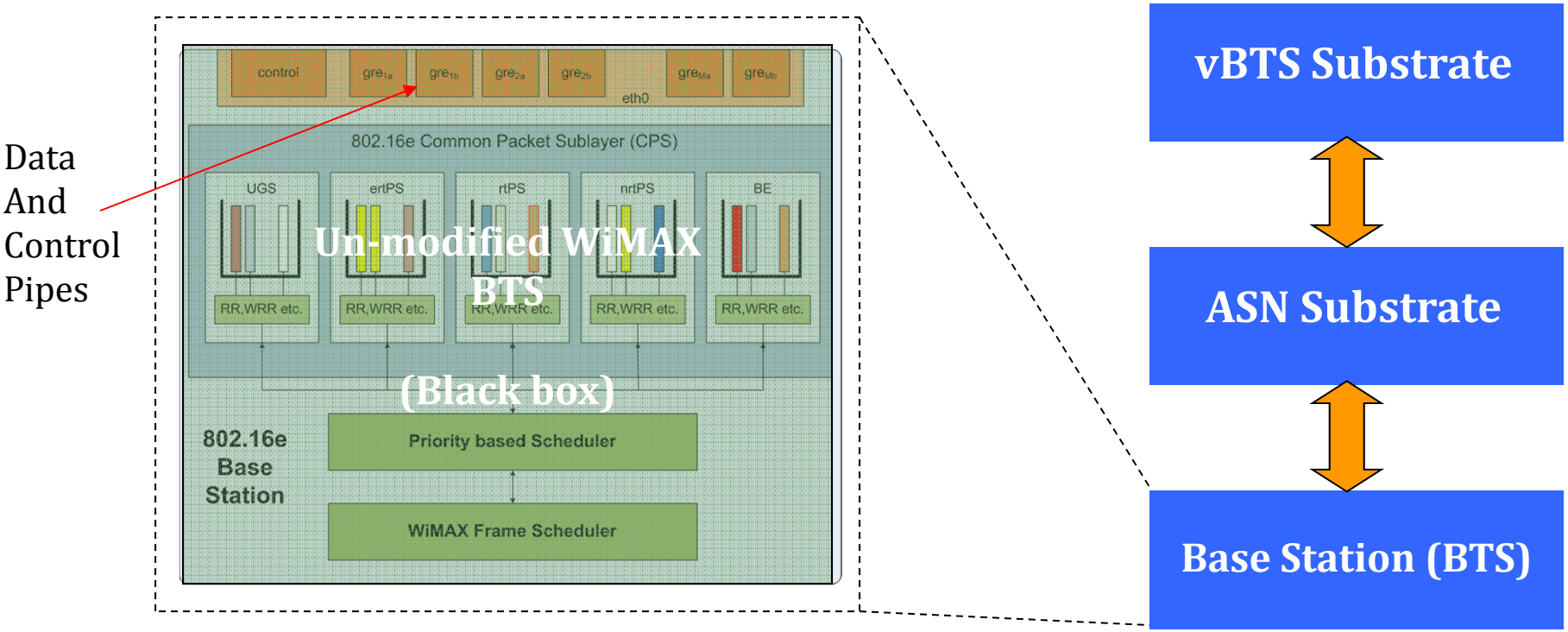


- Operational with an educational license
- Inherently IP based

WiMax Kit Architecture



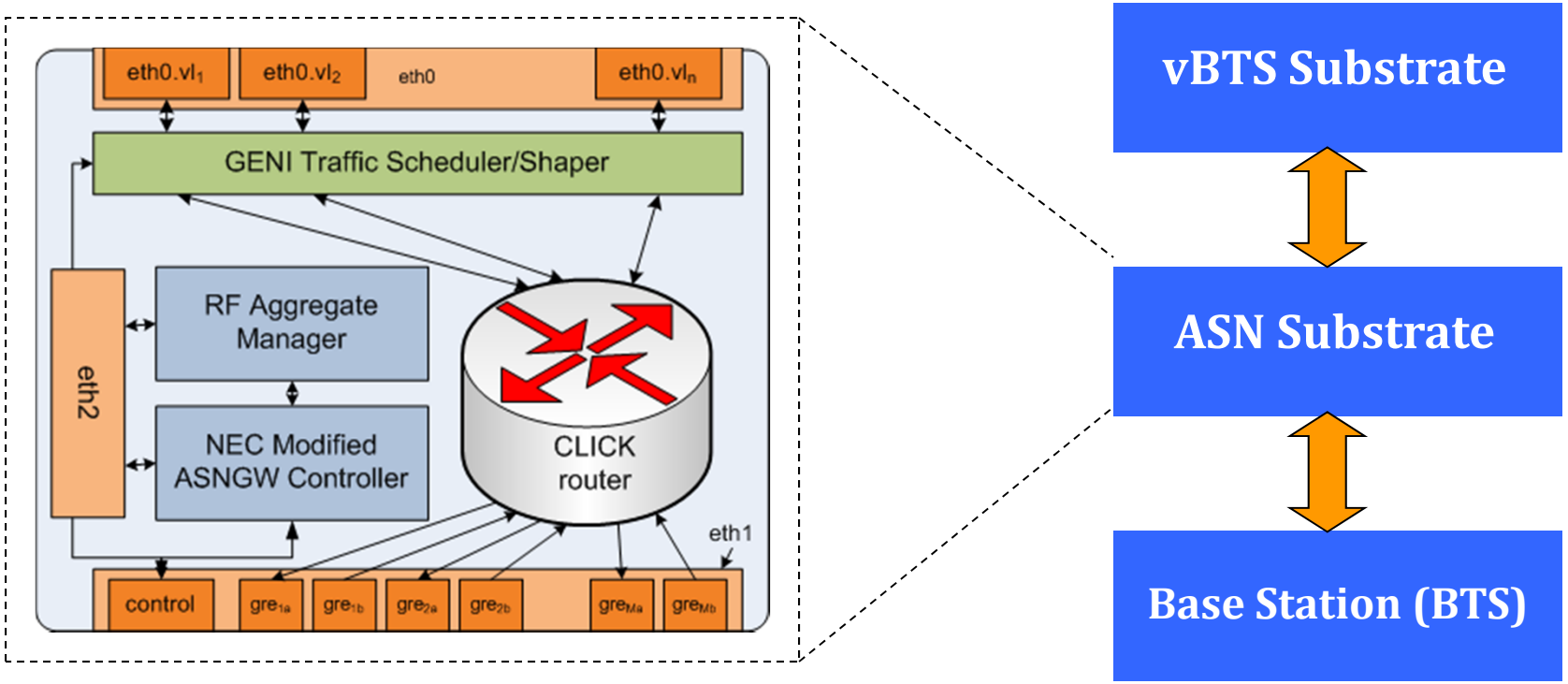
WiMax WiMax BTS



- The BTS itself is a black box
- Hence, the slice isolation mechanism and control framework is outside of this box

ORBIT

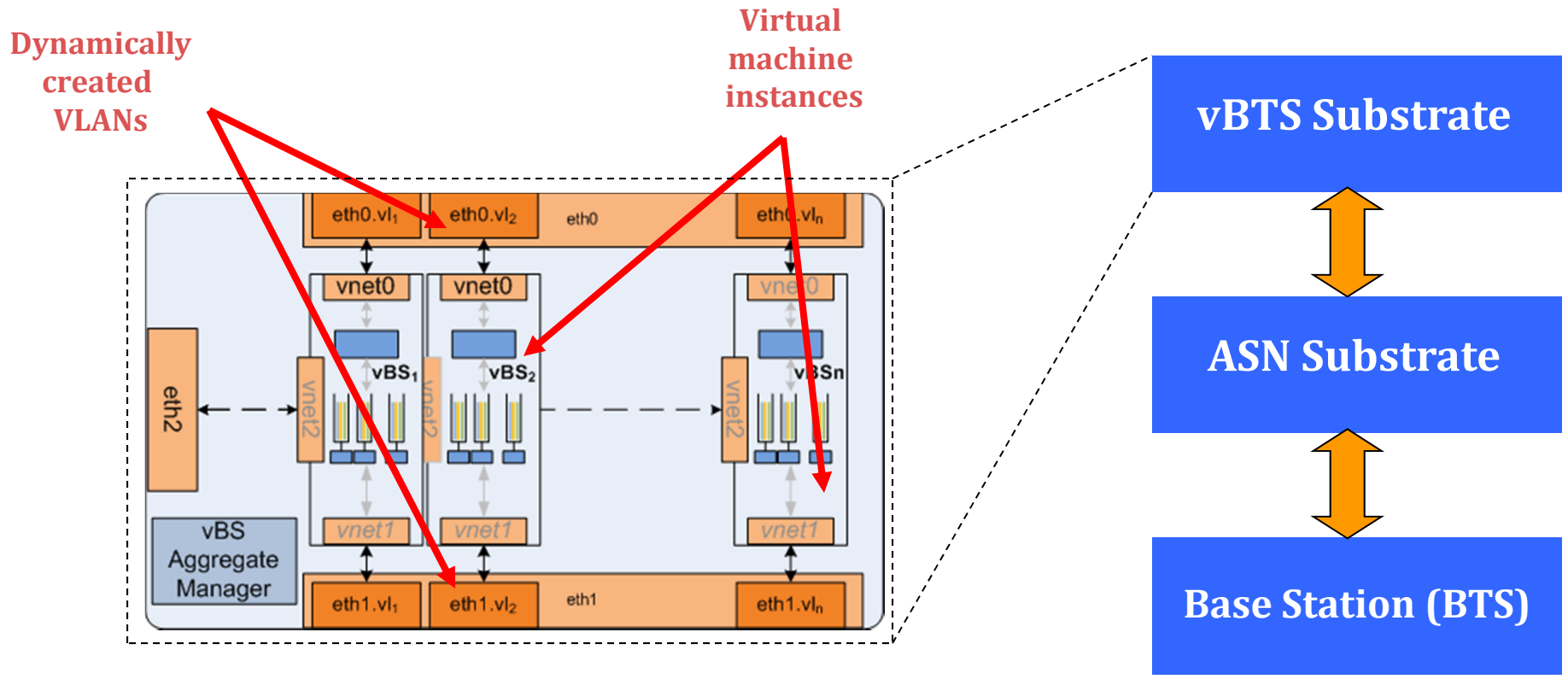
WiMax: ASN Packet Forwarding



- Removed all default IP routing, simplified ASN controller*
- All switching purely based on MAC addresses
- Implemented the *VNTS* shaping mechanism in click for slice isolation

** Work done at NEC*

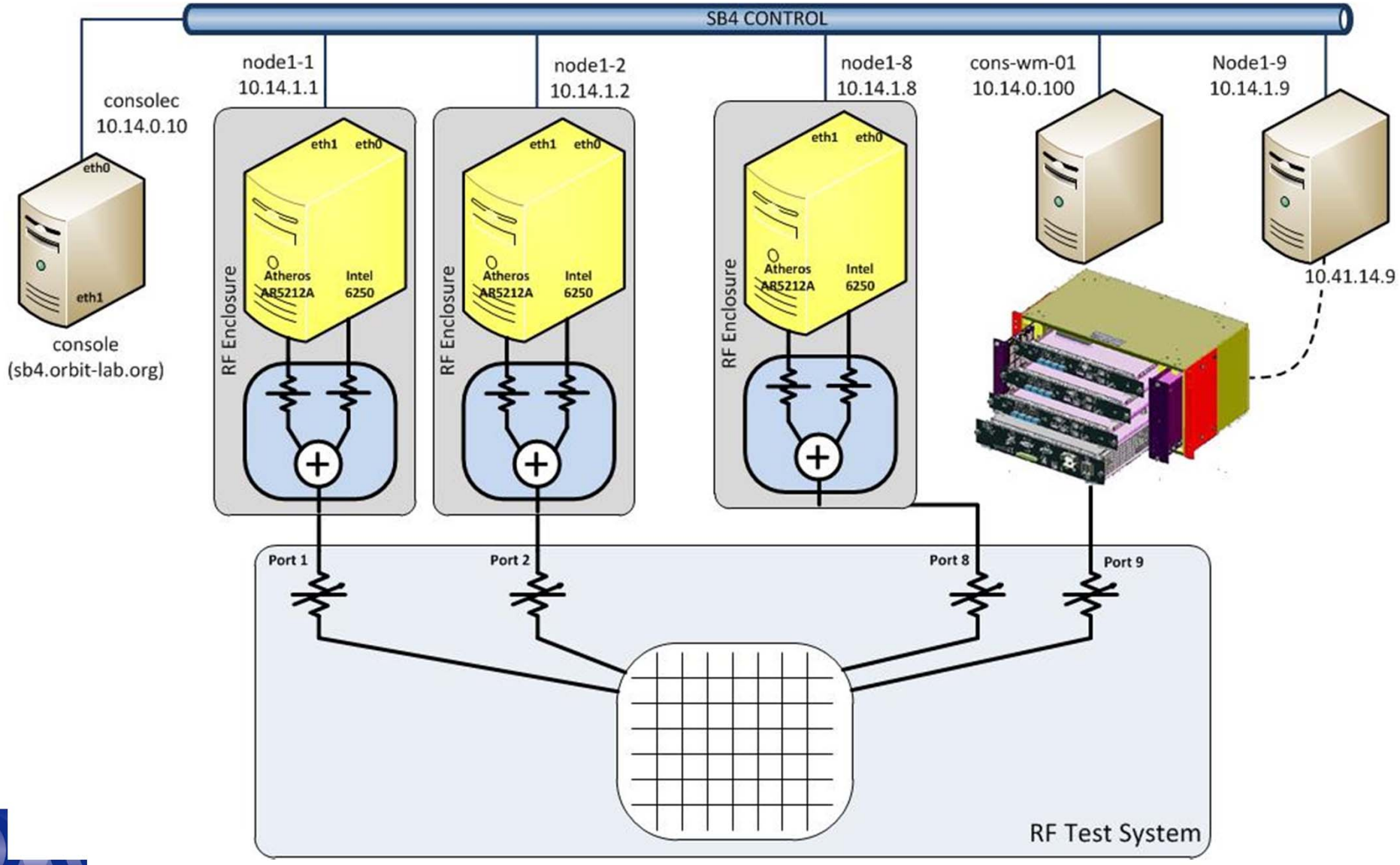
WiMax: vBTS Architecture



- Redirect all traffic from VLANs to individual slices
- Similar redirection from slices to outbound VLAN interfaces
- Grid services for creation, destruction, maintenance of slices, adding clients, slice allocation control …

ORBIT

SB4


SB4 CONTROL
consolec
10.14.0.10
eth0
eth1
console
(sb4.orbit-lab.org)
node1-1
10.14.1.1
node1-2
10.14.1.2
node1-8
10.14.1.8
cons-wm-01
10.14.0.100
Node1-9
10.14.1.9
10.41.14.9
RF Enclosure
eth1
eth0
Atheros
AR5212A
Intel
6250
Port 1
Port 2
Port 8
Port 9
RF Test System

Fourth Exercise: Connect to the BS

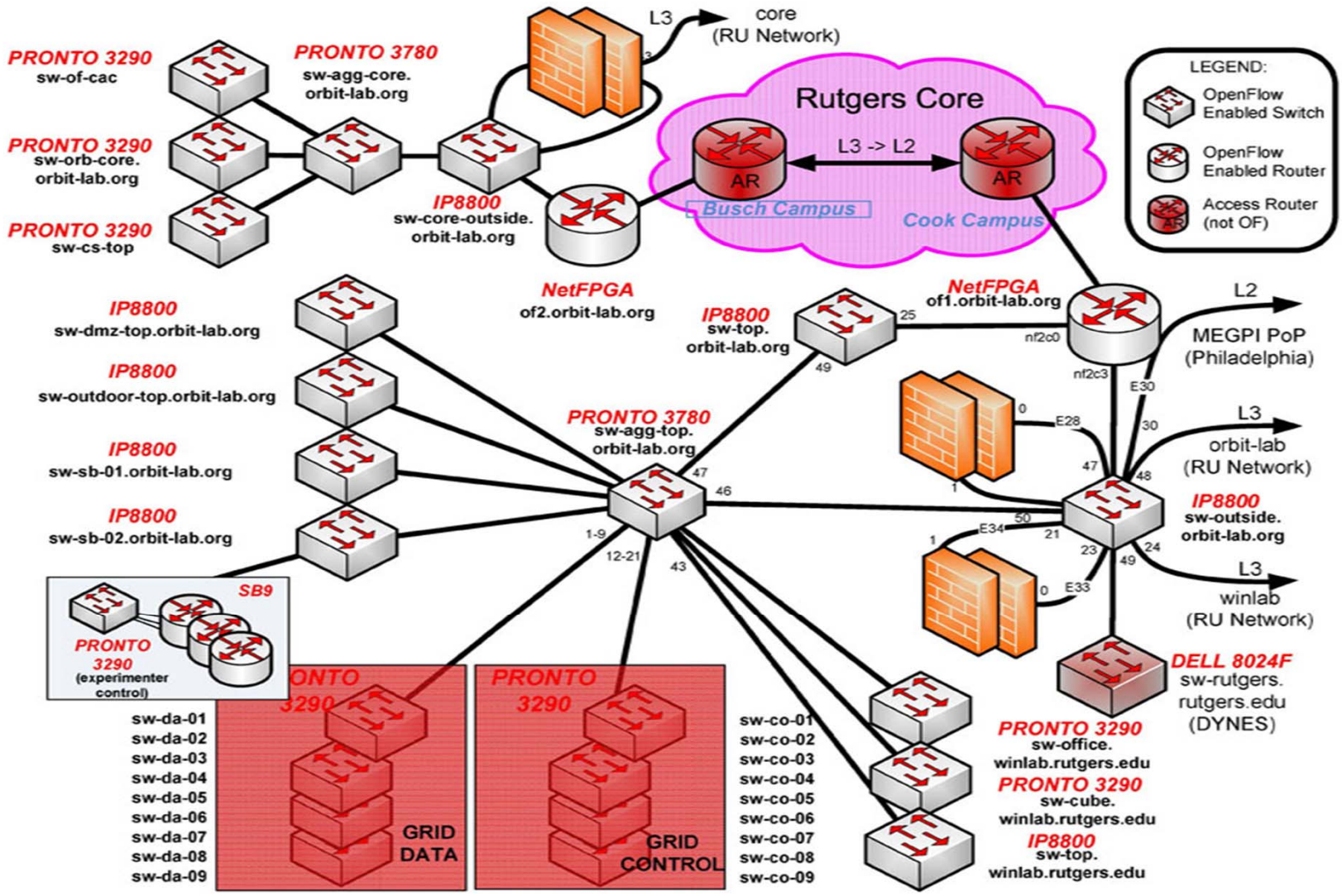
modprobe, wimaxd, wimaxcu
wimaxrf

Basic WiMAX Commands (Intel 6250)

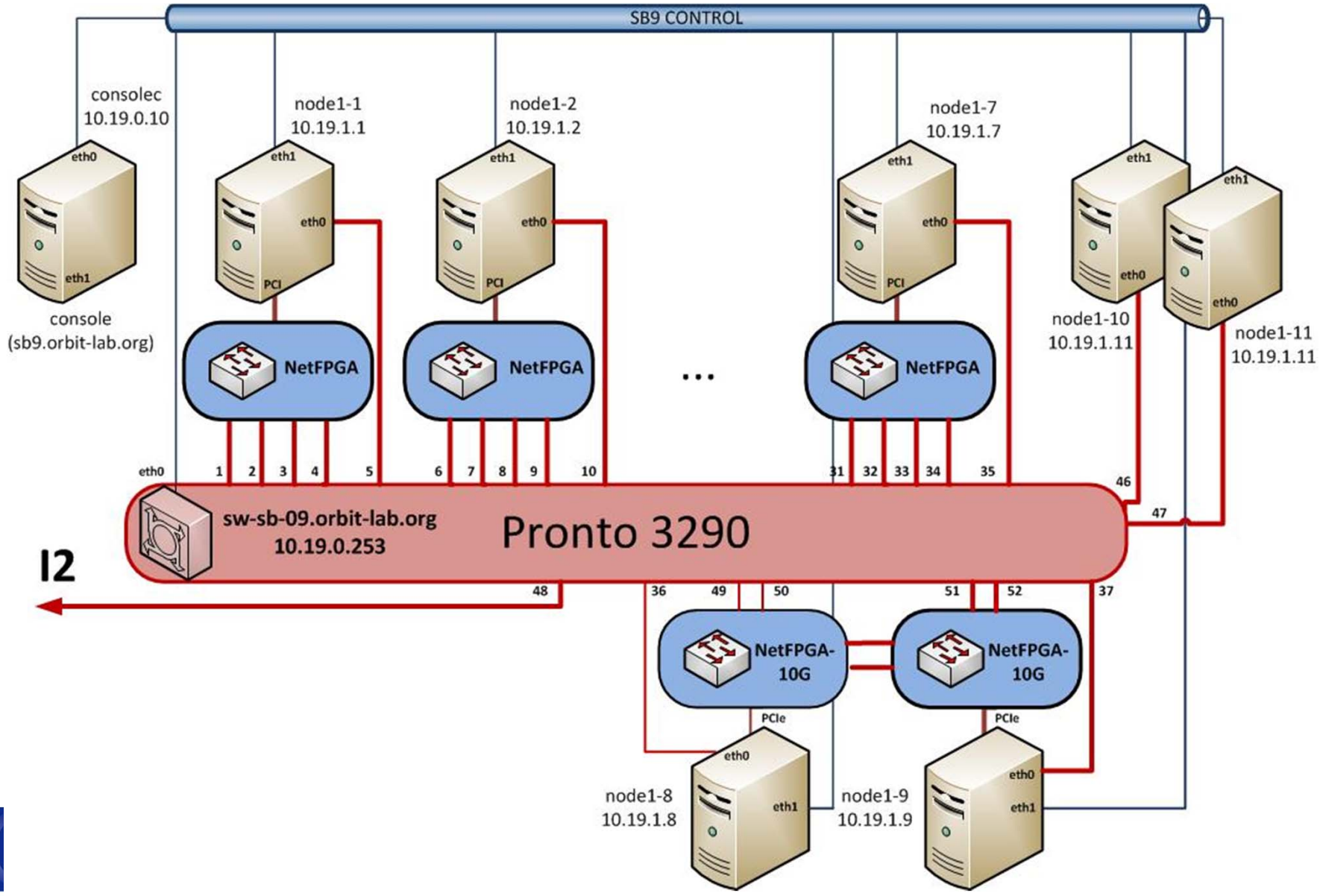
```
root@node1-1:~# lsusb
Bus 002 Device 003: ID 03fd:000f Xilinx, Inc.
Bus 002 Device 002: ID 8087:0020 Intel Corp. Integrated Rate Matching Hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 003: ID 8086:0186 Intel Corp. WiMAX Connection 2400m
Bus 001 Device 002: ID 8087:0020 Intel Corp. Integrated Rate Matching Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

root@node1-1:~# modprobe -i i2400m_usb

root@node1-1:~# ps -ef | grep wimax
root    15410    1  0 10:04 ?       00:00:00 /usr/bin/wimaxd -b -i wmx0
root    15438 15378  0 10:05 pts/0    00:00:00 grep --color=auto wimax

root@node1-1:~# wimaxcu connect network 51
Current Preferred Profile is:
                ID  : 51
                Name: GENI 4G
Connecting to GENI 4G Network...
Connection successful
```

OpenFlow in ORBIT

ORBIT SDN Deployment



SB9


SB9 CONTROL
consolec
10.19.0.10
console
(sb9.orbit-lab.org)
node1-1
10.19.1.1
node1-2
10.19.1.2
node1-7
10.19.1.7
node1-10
10.19.1.11
node1-11
10.19.1.11
NetFPGA
NetFPGA
NetFPGA
...
sw-sb-09.orbit-lab.org
10.19.0.253
Pronto 3290
I2
NetFPGA-
10G
NetFPGA-
10G
node1-8
10.19.1.8
node1-9
10.19.1.9
PCI
PCIe