Networking on WARP

Chris Hunter
Rice University

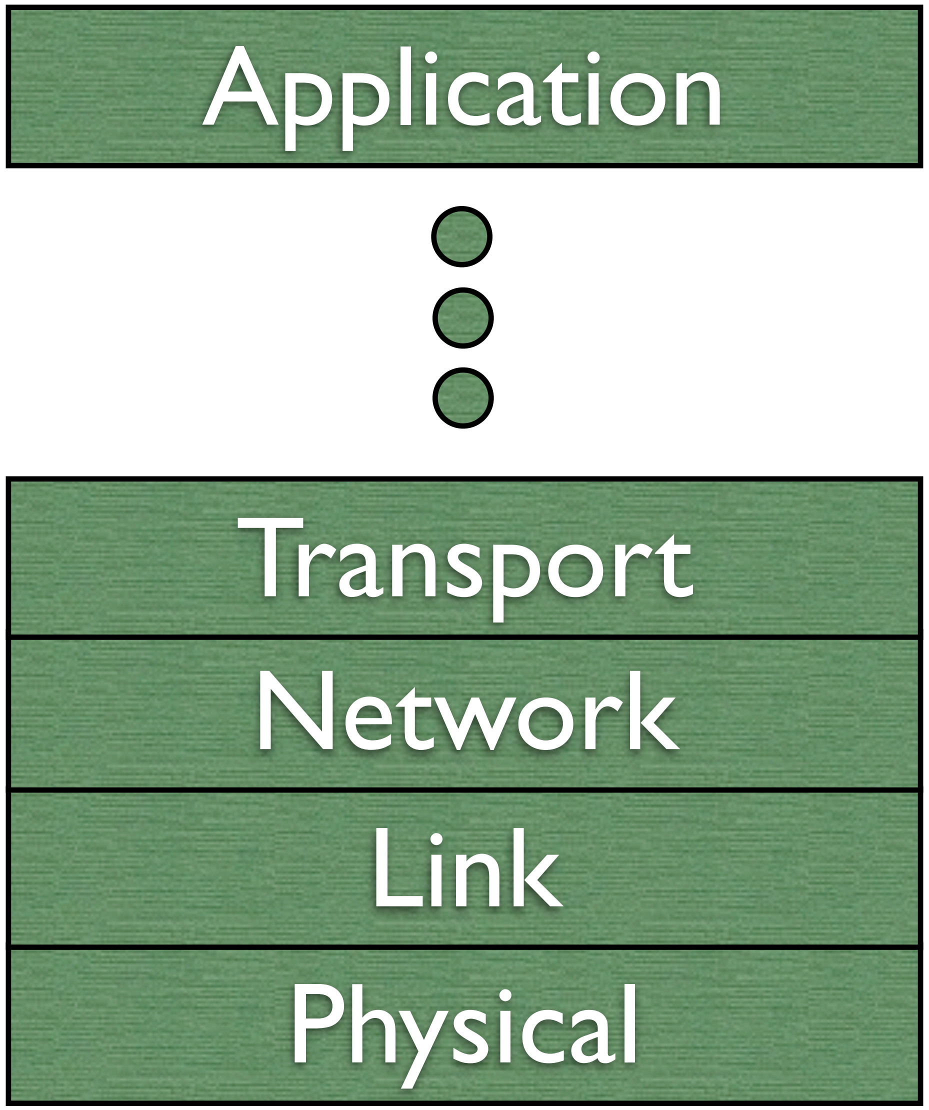
WARP Workshop at DySPAN 2011
May 3, 2011

warp.rice.edu

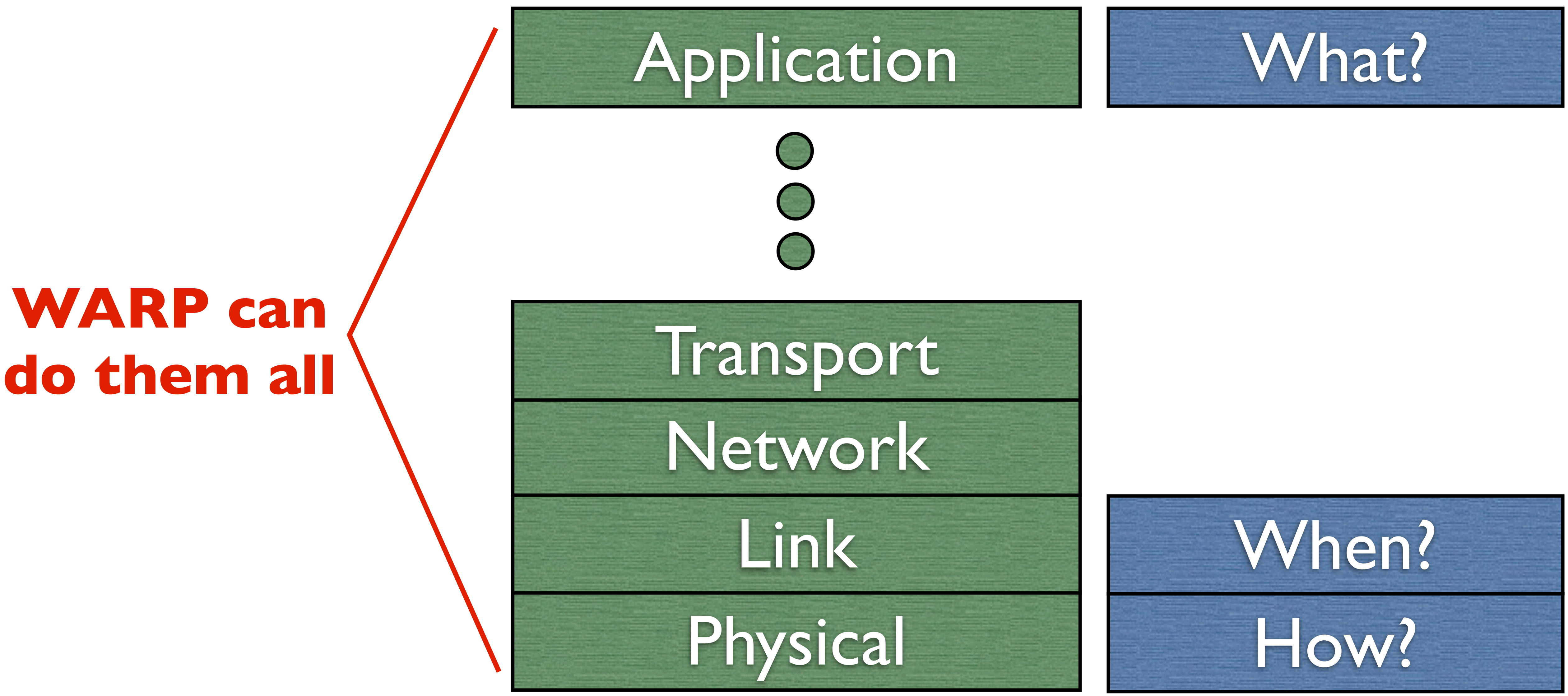
The OSI Model

Application
Presentation
Session
Transport
Network
Link
Physical

The OSI Model



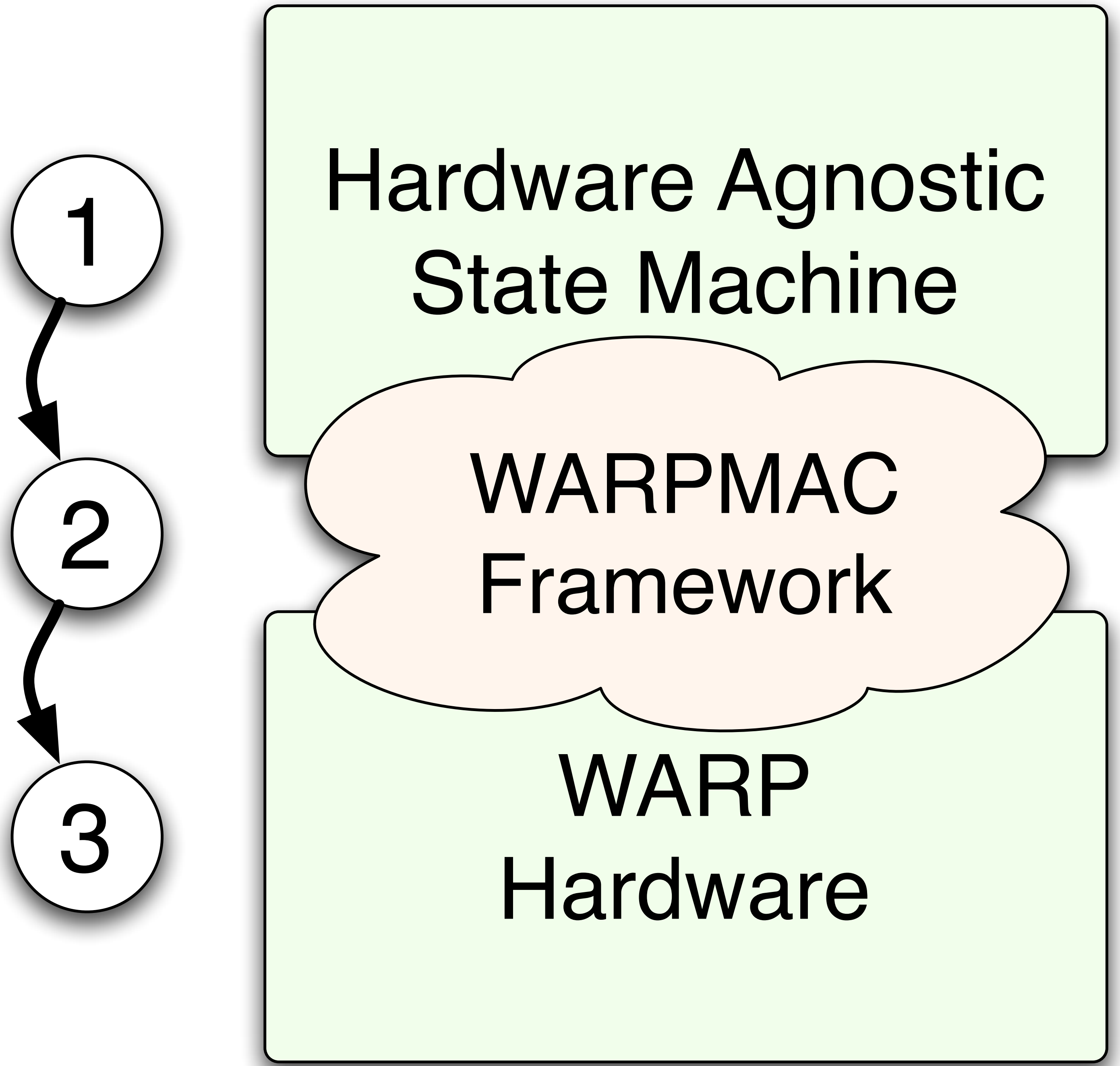
The OSI Model



Outline

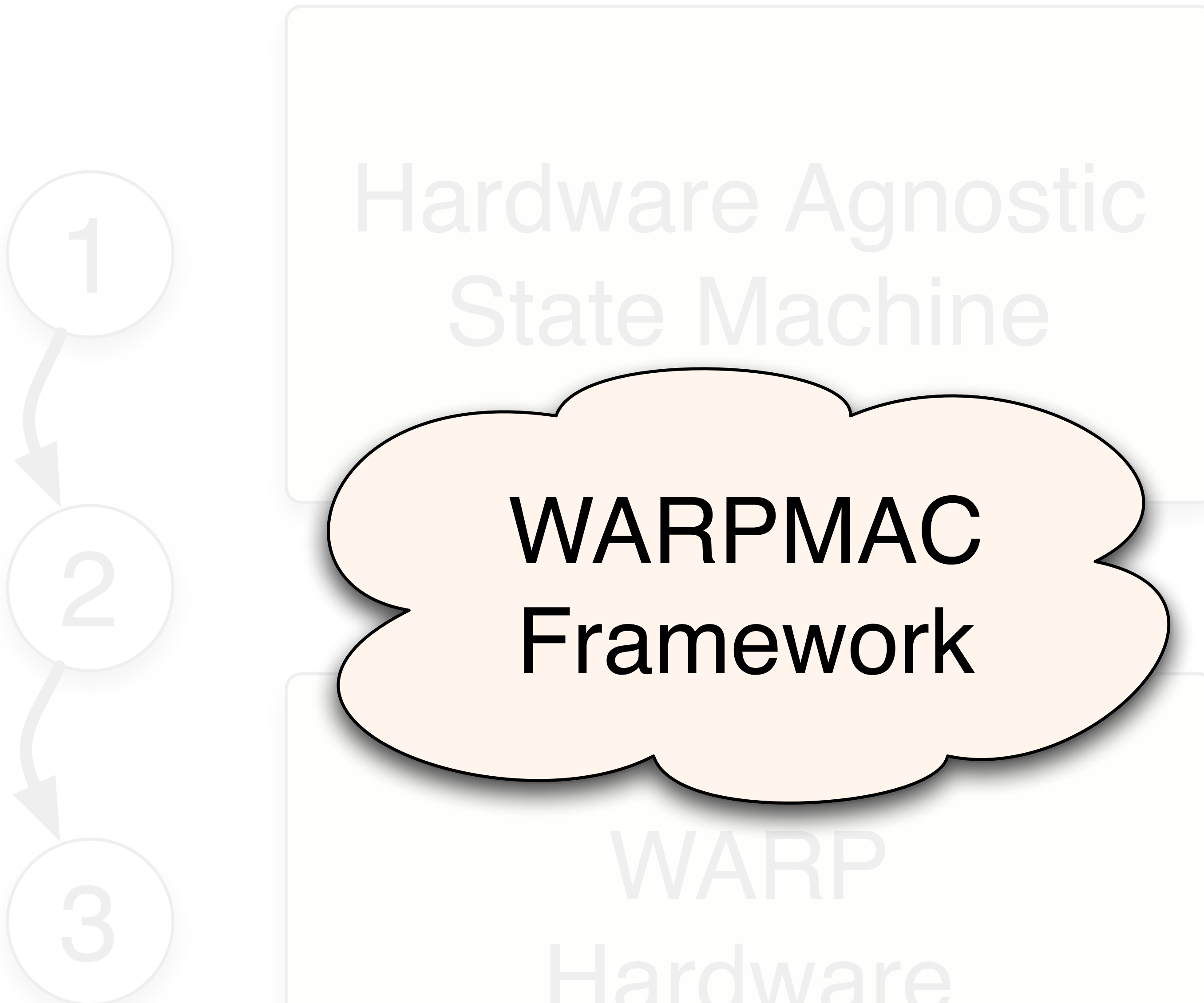
- Design Realization
- Example
- Lab Exercises

Design Realization



- Program high-level MAC behavior independent of hardware
- Use the WARPMAC framework to stitch the MAC to hardware

Design Realization



- No way to "lock" the framework and have it support all possible future MAC layers

Solution: *WARPMAC must grow with new algorithms*

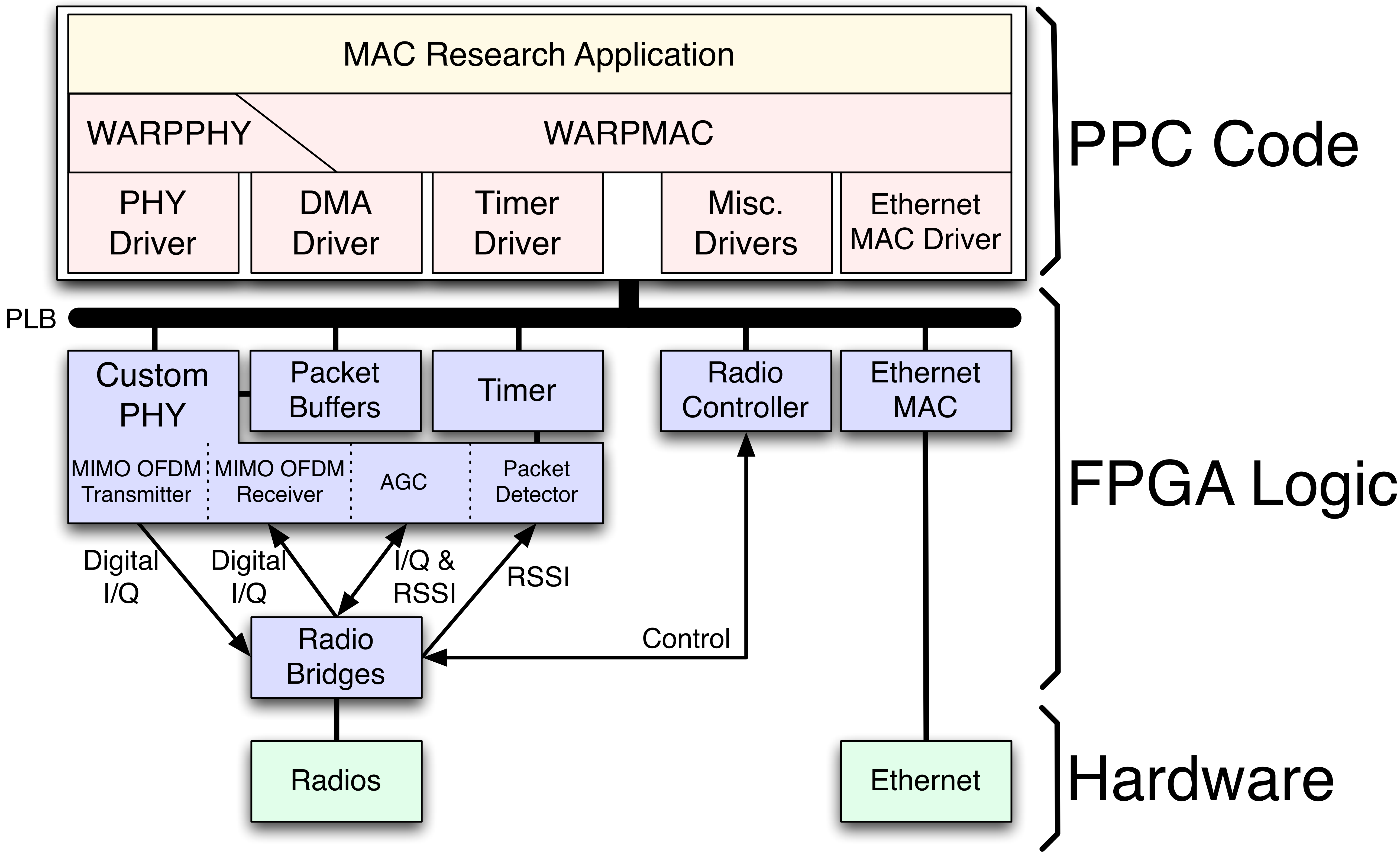
Problem: *How do we maintain sync between designs?*

Reference Designs

Reference Designs

- Snapshots of the WARP repository
- Free, open-source releases at regular intervals
  - Today's exercises are Reference Design v16.1
- Reference design is an example of:
  - a working PHY
  - a working MAC
  - the way all the pieces fit together
  - stuff that we use for our research

Reference Designs




User Code
WARPMAC
WARPPHY
Drivers

PHY Driver:

- Configure very low-level parameters
  - Correlation thresholds
  - FFT scaling parameters
  - Filter coefficients

Radio Controller Driver:

- Set center frequency
- Switch from Rx to Tx mode and vice versa

PHY Control:

- Provides control over PHY commonalities
  - General initialization command
  - Configure constellation order
  - Configure coding rate
  - “Start” and “Stop” the PHY

Mostly PHY agnostic

Completely PHY dependent

MAC Control:

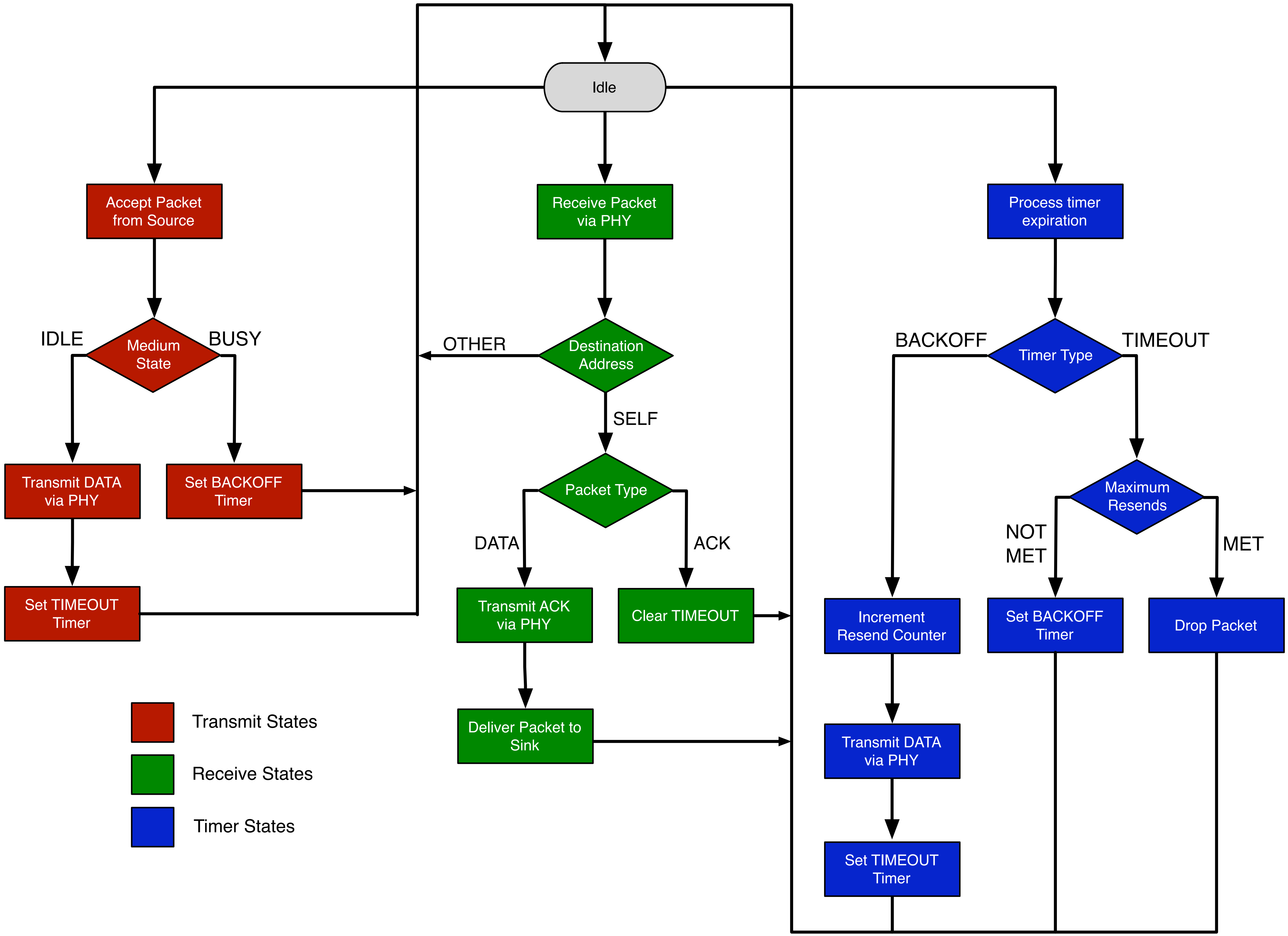
- Provides control over MAC commonalities
  - Timers for timeouts, backoffs, etc.
  - Carrier-sensing functions
  - Register user callbacks for event-driven operation

User-level MAC Algorithms:

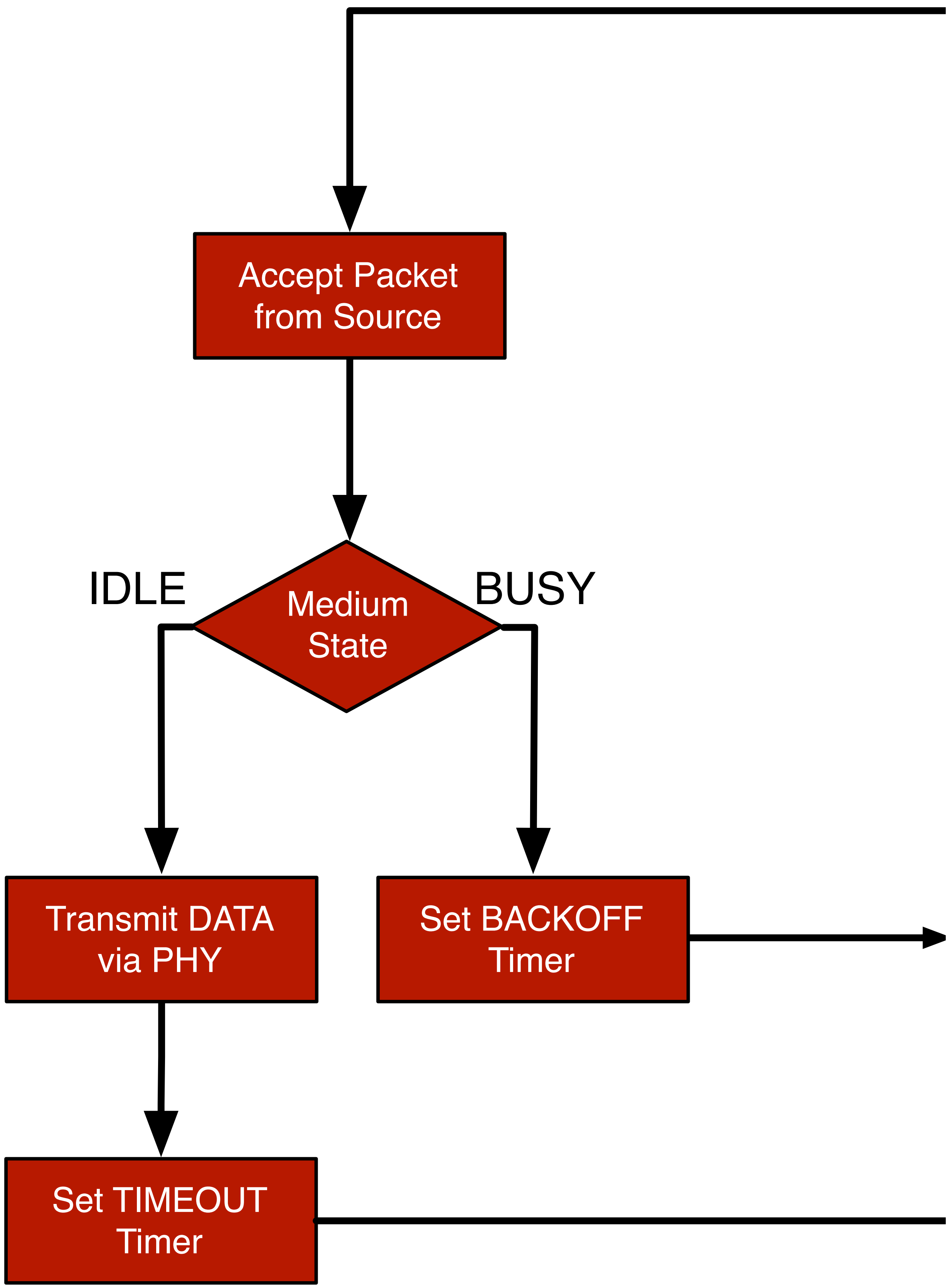
- High-level MAC algorithms
- Some examples so far:
  - Aloha
  - CSMA/CA
  - MAC Workshop Exercises
  - Distributed On-demand Cooperation (DOC)

An example: CSMA

- Carrier-Sense Multiple Access
- Serves as a foundation for a large class of other random access protocols
- Fairly simple algorithm


Idle
Accept Packet from Source
Medium State
IDLE
BUSY
Transmit DATA via PHY
Set BACKOFF Timer
Set TIMEOUT Timer
Receive Packet via PHY
Destination Address
OTHER
SELF
Packet Type
DATA
ACK
Transmit ACK via PHY
Clear TIMEOUT
Deliver Packet to Sink
Process timer expiration
Timer Type
BACKOFF
TIMEOUT
Maximum Resends
NOT MET
MET
Increment Resend Counter
Set BACKOFF Timer
Drop Packet
Transmit DATA via PHY
Set TIMEOUT Timer
Transmit States
Receive States
Timer States

Transmit States



Transmit States

warpmac_emacRx_handler

- Starts DMA transfer from EMAC to PHY

dataFromNetworkLayer_callback

- Constructs Macframe header for data packet

If medium is idle {

warpmac_prepPhyForXmit

- Configures PHY
- Copies Macframe header into PHY's buffer

warpmac_startPhyXmit

- Disables packet detection
- Starts radio controller's transmit state machine

warpmac_finishPhyXmit

- Polls PHY and waits for it to complete
- Enables packet detection and radio reception

- Starts a timeout timer
- Decrements remaining resend counter

}

If medium is busy {

- Starts a backoff timer

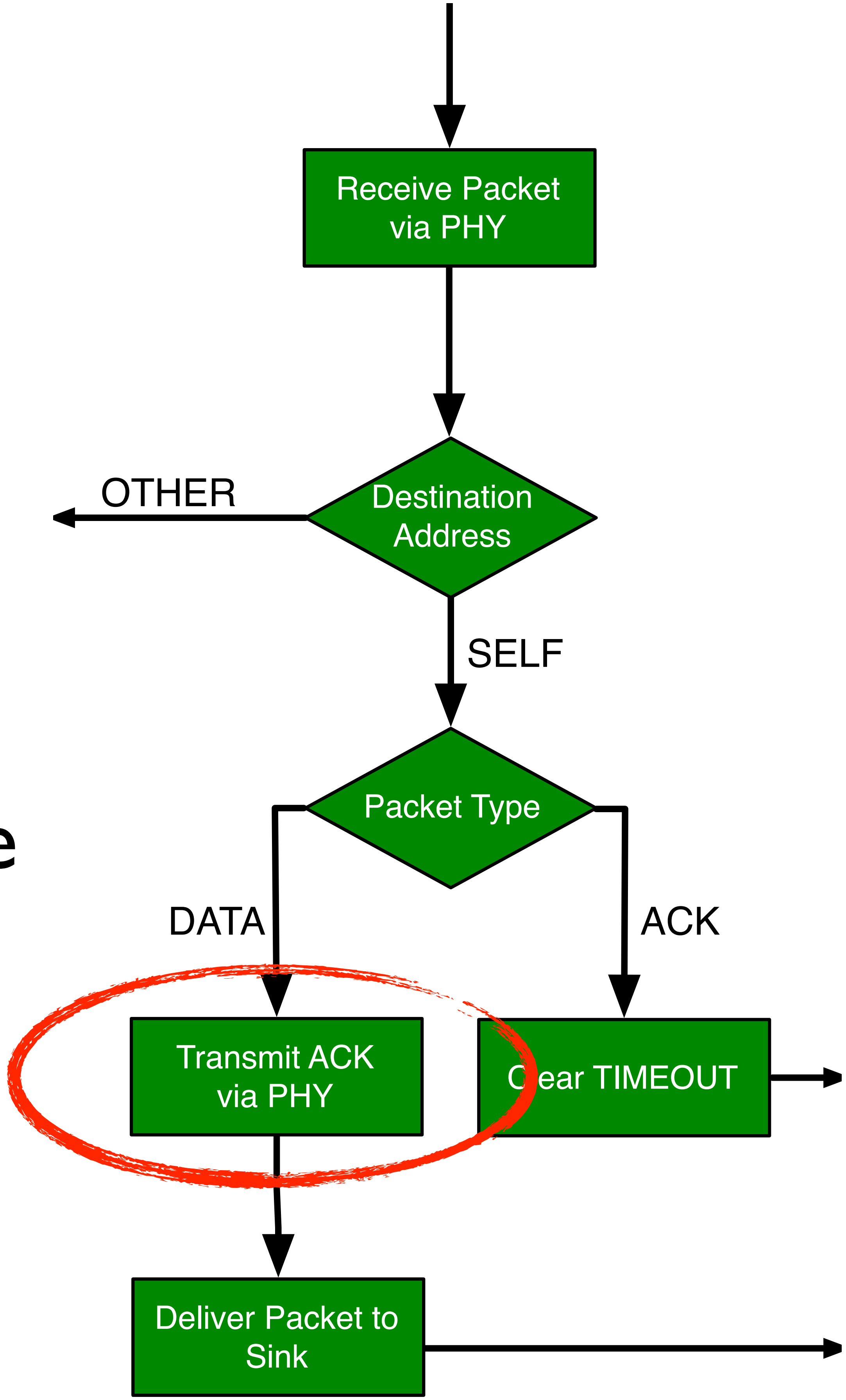
}

- Clears EMAC

WARPMAC

User-Code

Receive States



How can we use WARPMAC to keep turn-around-time (TAT) small?

Receive States

warpmac_pollPhy

- Copies header into Macframe

phyRx_goodHeader_callback

- Checks address/type fields of Macframe header

If data {

- Polls PHY receiver and waits for a "Good" or "Bad" state

If Good {

Send acknowledgement

warpmac_prepPktToNetwork

- Starts DMA transfer from PHY to EMAC

warpmac_finishPhyXmit

- Polls PHY and waits for it to complete
- Enables packet detection and radio reception

warpmac_startPktToNetwork

- Polls DMA and waits for it to complete
- Starts EMAC transmission

}

}

If acknowledgment {

- Clears timeout timer

}

- Resets PHY

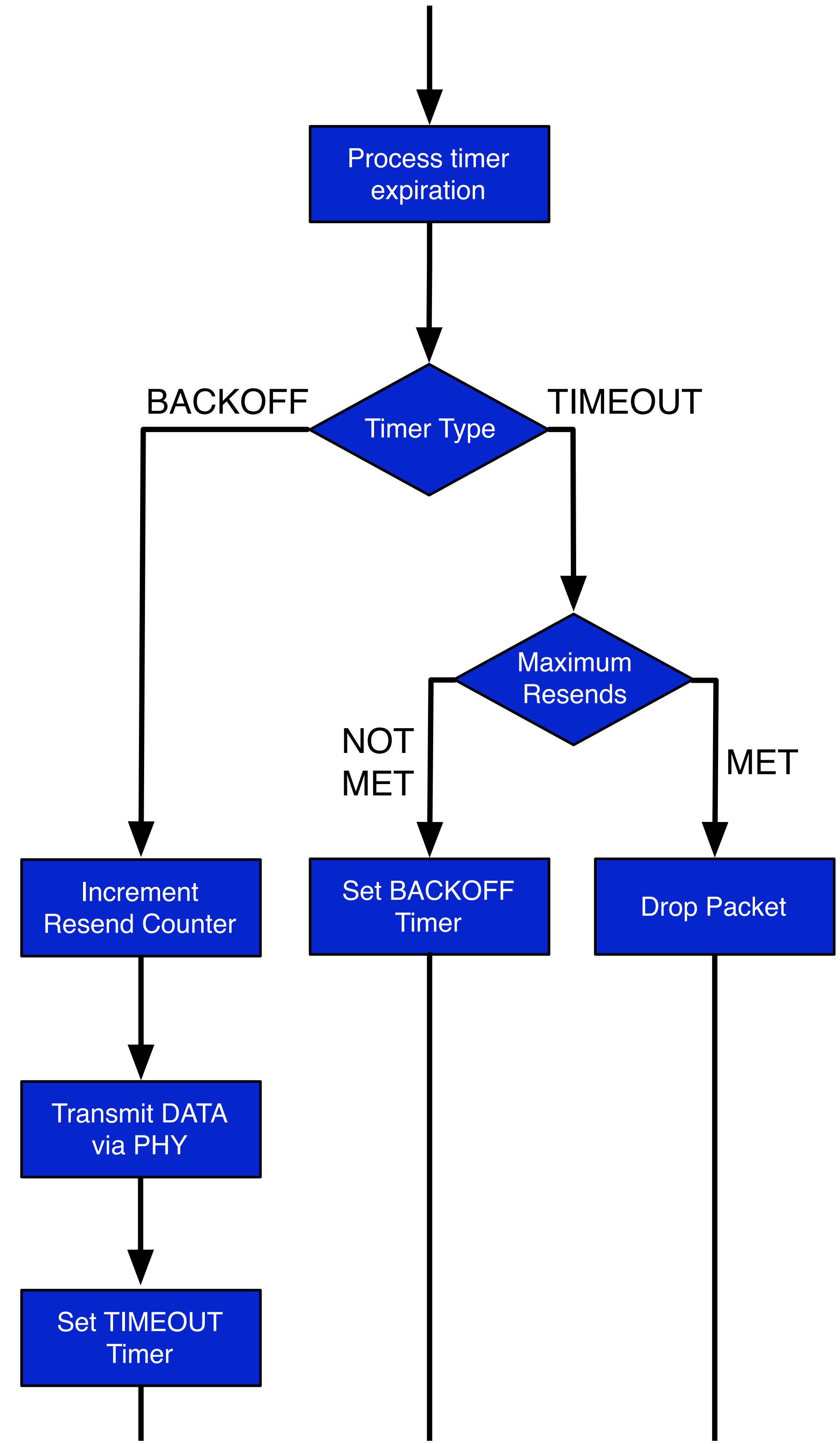
Fast Turn-Around Time (TAT)

1) Software calls
2) Hardened "autoresponder"

WARPMAC

User-Code

Timer States



Timer States

warpmac_pollTimer

- Checks each timer status and calls relevant callbacks

timer_callback

- Checks timer type

If timeout {

- Starts a backoff timer

}

If backoff {

warpmac_prepPhyForXmit

- Configures PHY
- Copies Macframe header into PHY's buffer

warpmac_startPhyXmit

- Disables packet detection
- Starts radio controller's transmit state machine

warpmac_finishPhyXmit

- Polls PHY and waits for it to complete
- Enables packet detection and radio reception

- Starts a timeout timer
- Decrements remaining resend counter

}

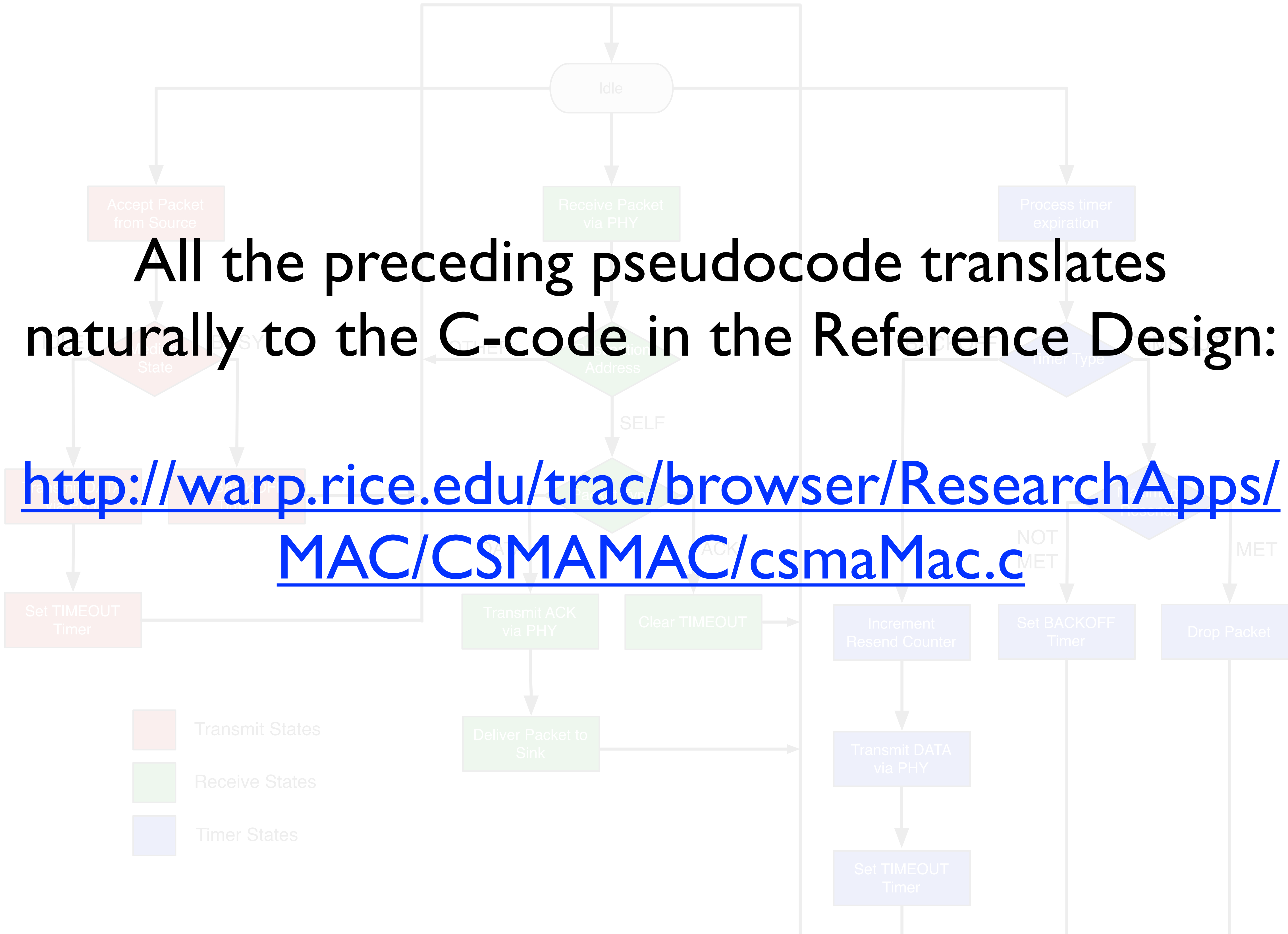
- Clears timers

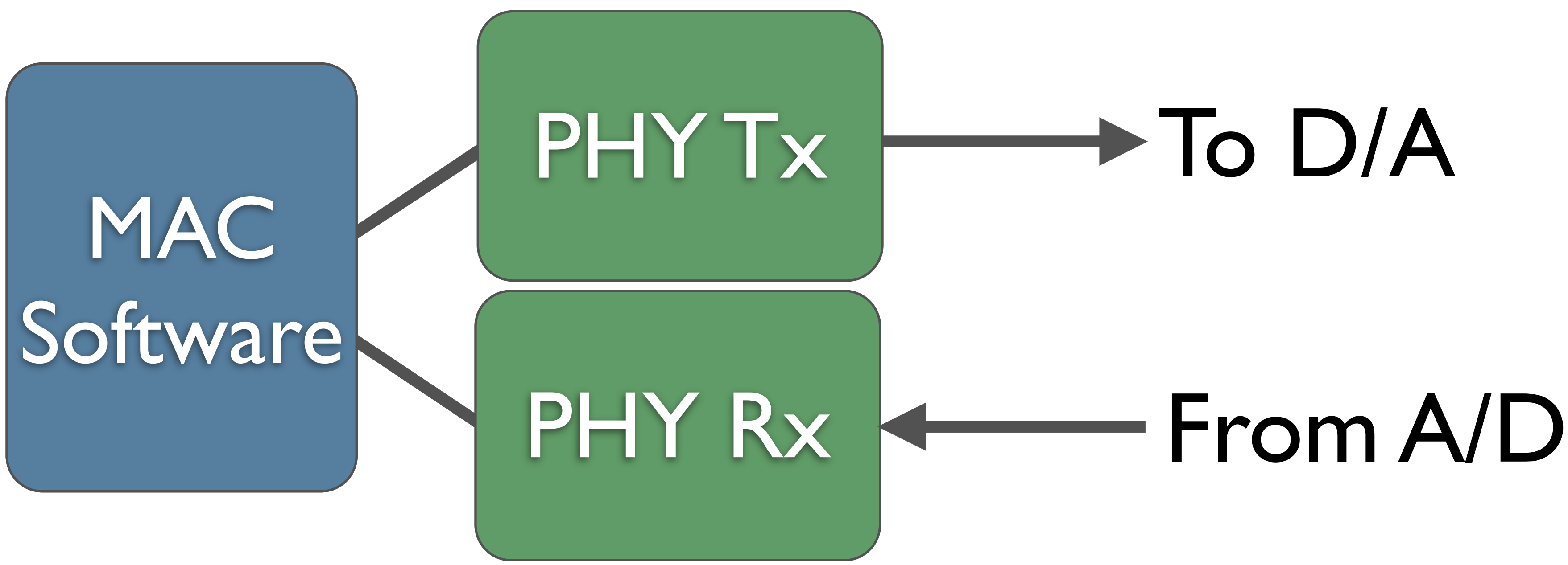
WARPMAC

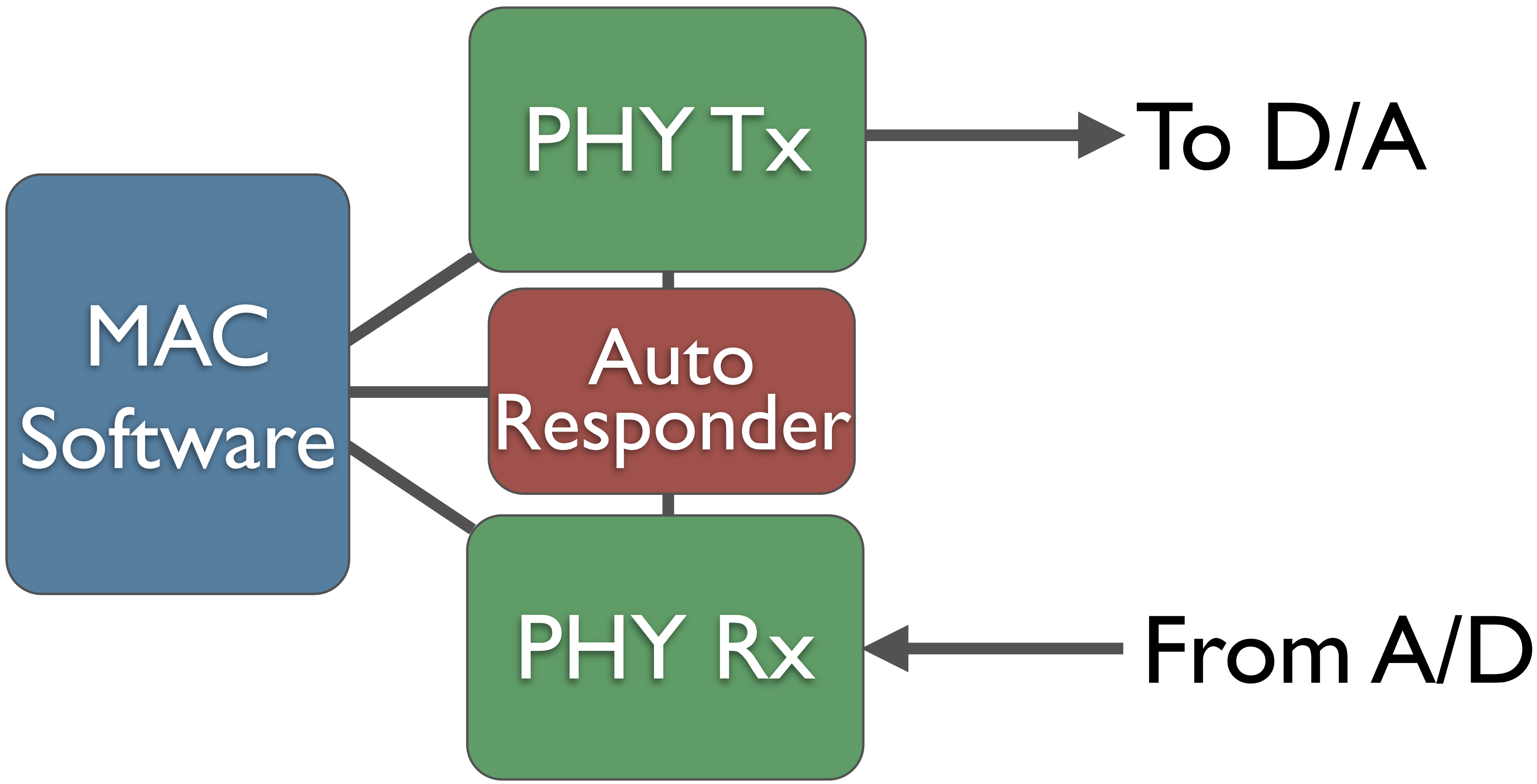
User-Code

All the preceding pseudocode translates naturally to the C-code in the Reference Design:

http://warp.rice.edu/trac/browser/ResearchApps/MAC/CSMAMAC/csmaMac.c

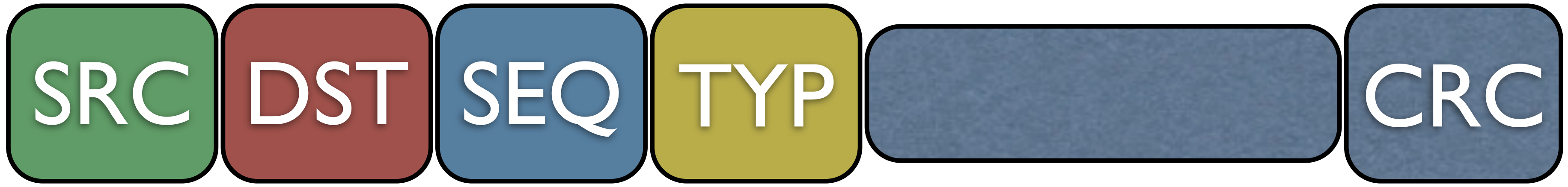



MAC
Software
PHY Tx
PHY Rx
To D/A
From A/D

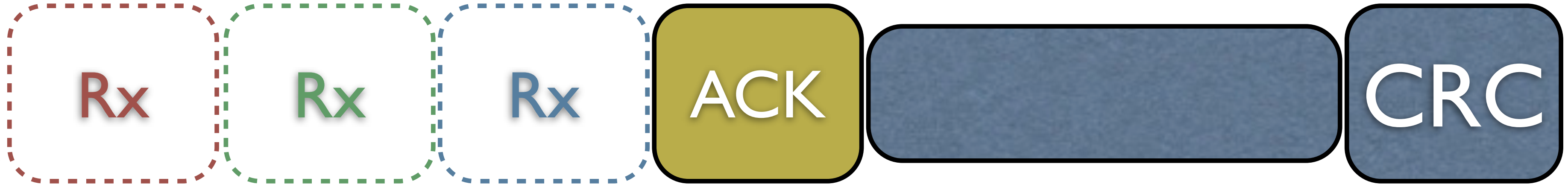


MAC specifies packet **templates**, Rx packet **conditions** and Tx header **substitution**. PHY initiates transmission automatically.

PACKET FORMAT



TEMPLATES$_{\text{ACK}}$



CONDITIONS

Template	Addressed to Node 1	Good Payload	Bad Payload	Packet Type
ACK	✓	✓		DATA

PACKET FORMAT

SRC	DST	SEQ	TYP		CRC

TEMPLATES$_{ACK}$

Rx	Rx	Rx	ACK		CRC

CONDITIONS

Template	Addressed to Node 1	Good Payload	Bad Payload	Packet Type
ACK	✓	✓		DATA

Rx PACKET

0	1	15	DATA		CRC

PACKET FORMAT

SRC	DST	SEQ	TYP		CRC

TEMPLATES$_{\text{ACK}}$

Rx	Rx	Rx	ACK		CRC

CONDITIONS

Template	Addressed to Node 1	Good Payload	Bad Payload	Packet Type
ACK	✓	✓		DATA

Rx PACKET

0	1	15	DATA		CRC

Tx PACKET

Rx	Rx	Rx	ACK		CRC

PACKET FORMAT

SRC	DST	SEQ	TYP		CRC

TEMPLATES$_{ACK}$

Rx	Rx	Rx	ACK		CRC

CONDITIONS

Template	Addressed to Node 1	Good Payload	Bad Payload	Packet Type
ACK	✓	✓		DATA

Rx PACKET

0	1	15	DATA		CRC

Tx PACKET

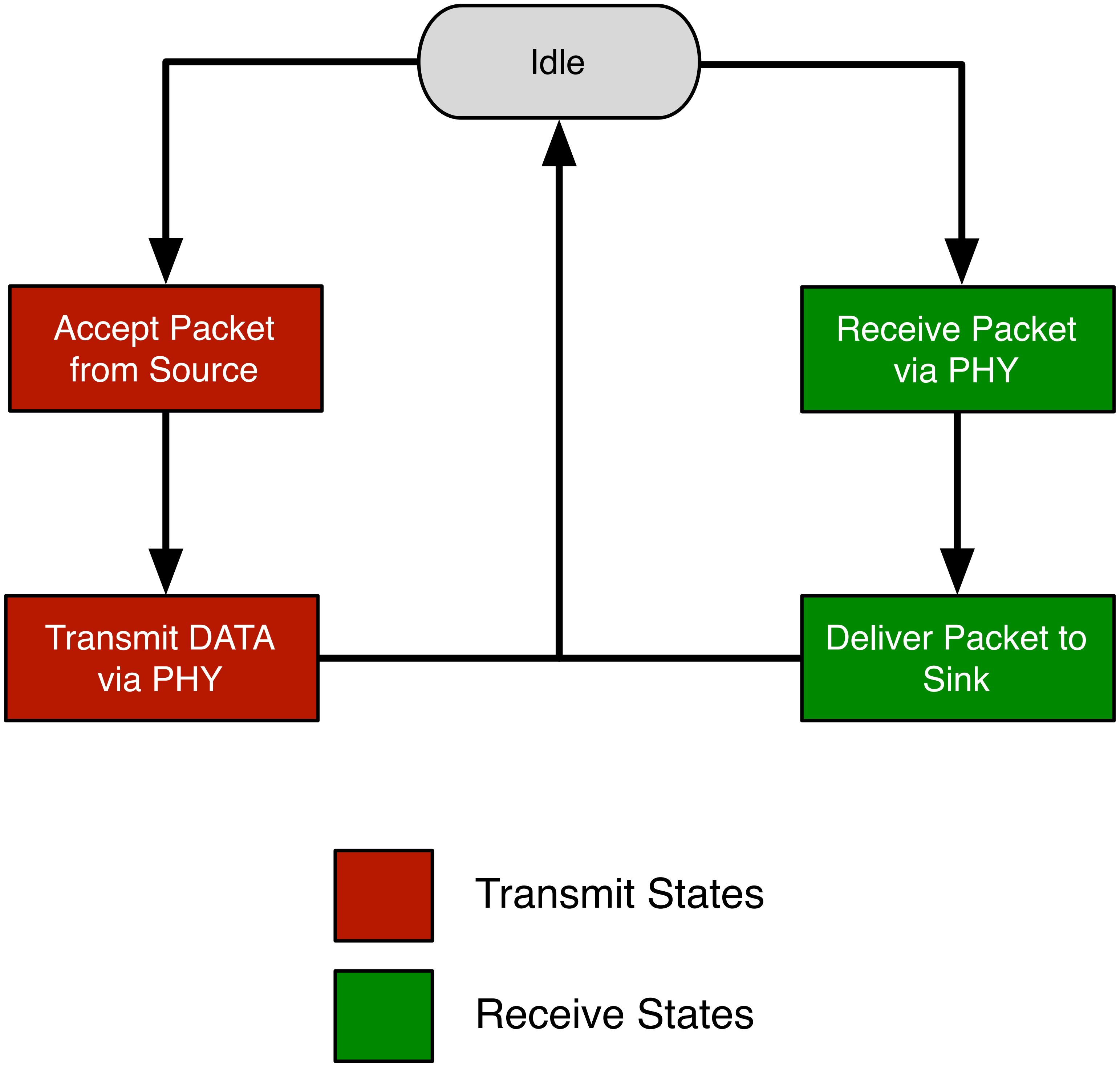
1	0	15	ACK		CRC

Questions?

Lab Exercises

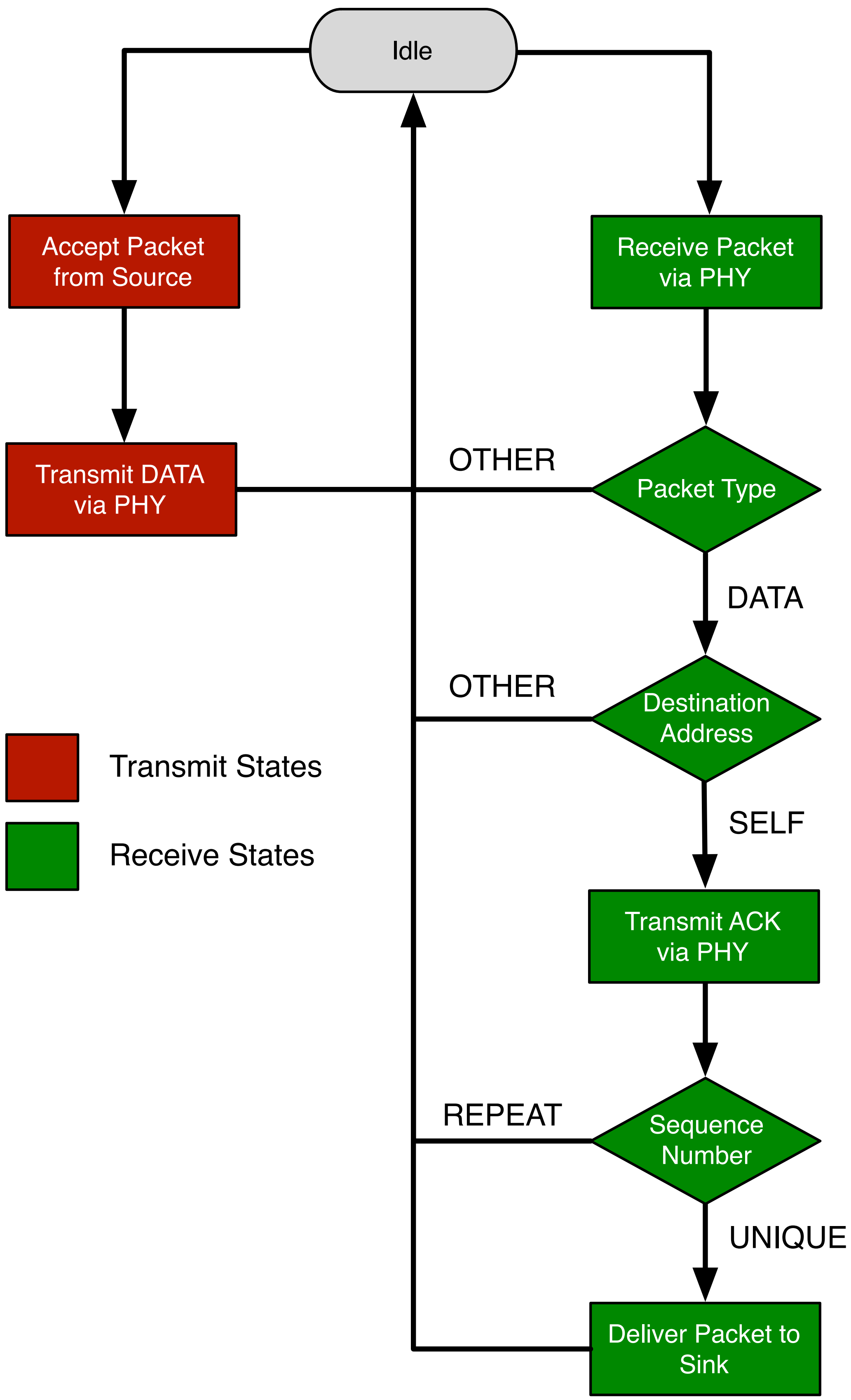
noMAC	Too simple to be a MAC; just puts packets over the air
halfMAC SW	Reception-half of a MAC (using software calls for ACKs)
cogMac	"Cognitive" MAC example (using autoresponder for ACKs)

noMac

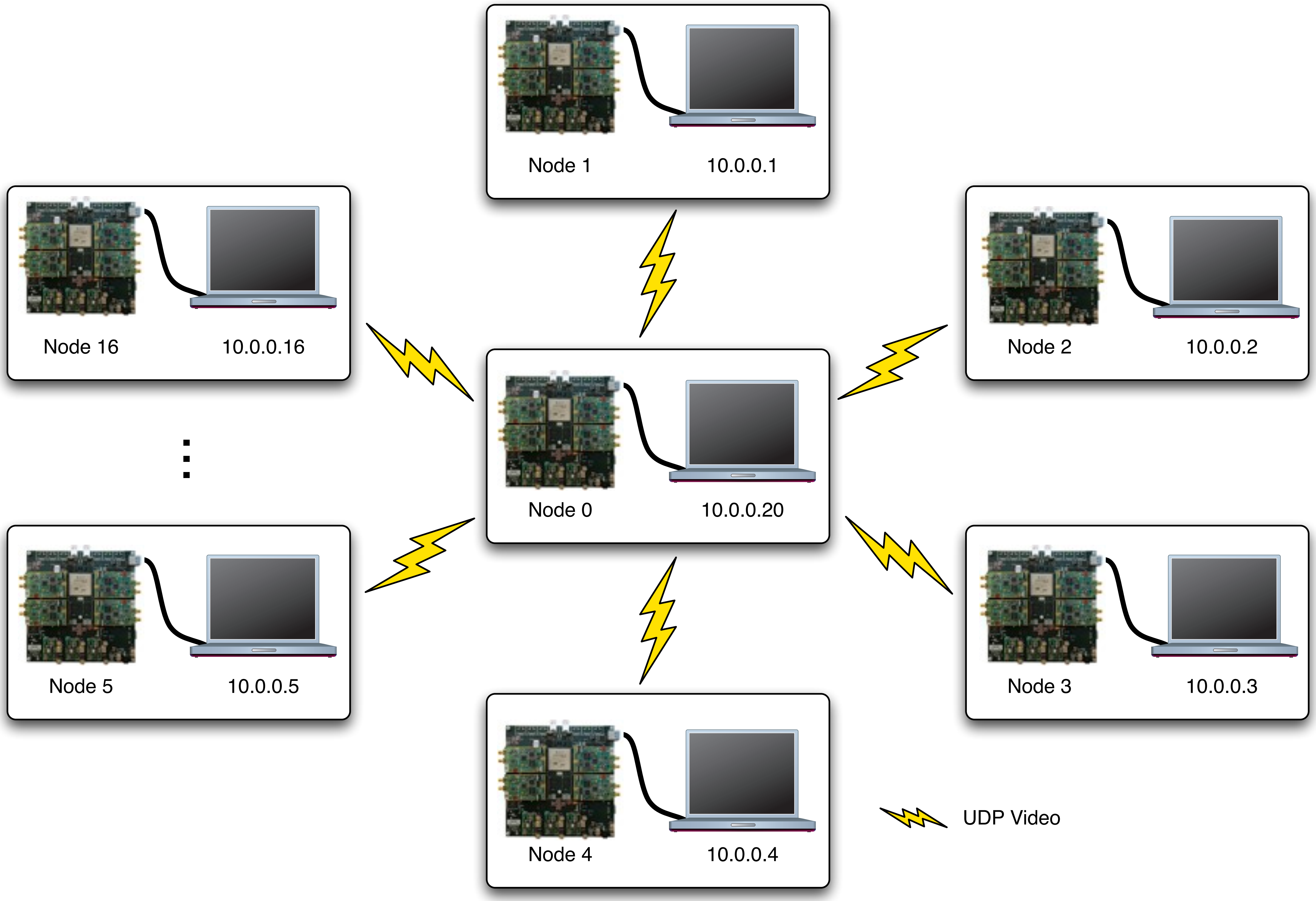


To test your noMac code, ping 10.0.0.20

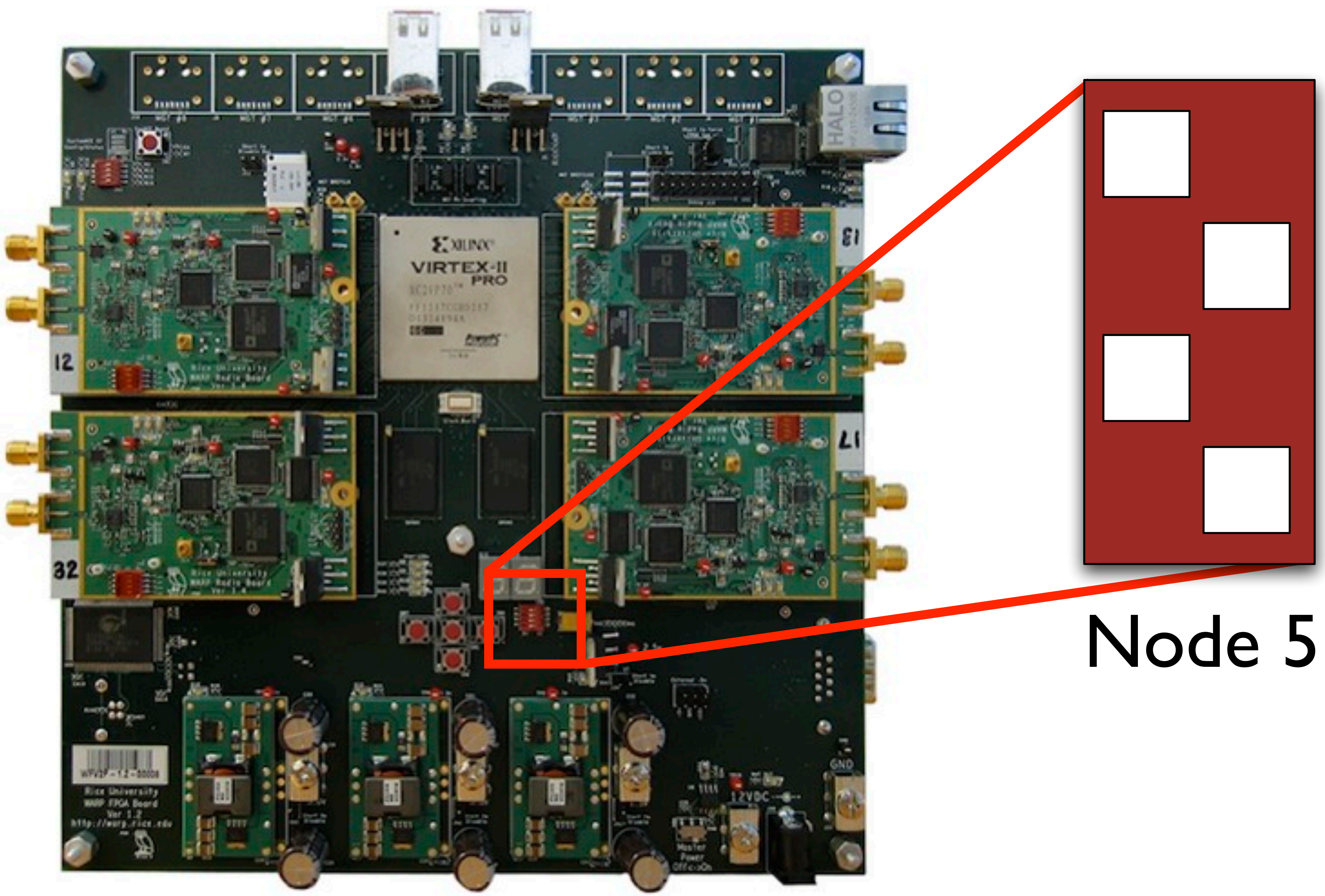
halfMac


Idle
Accept Packet from Source
Receive Packet via PHY
Transmit DATA via PHY
OTHER
Packet Type
DATA
OTHER
Destination Address
SELF
Transmit States
Receive States
Transmit ACK via PHY
REPEAT
Sequence Number
UNIQUE
Deliver Packet to Sink

halfMac

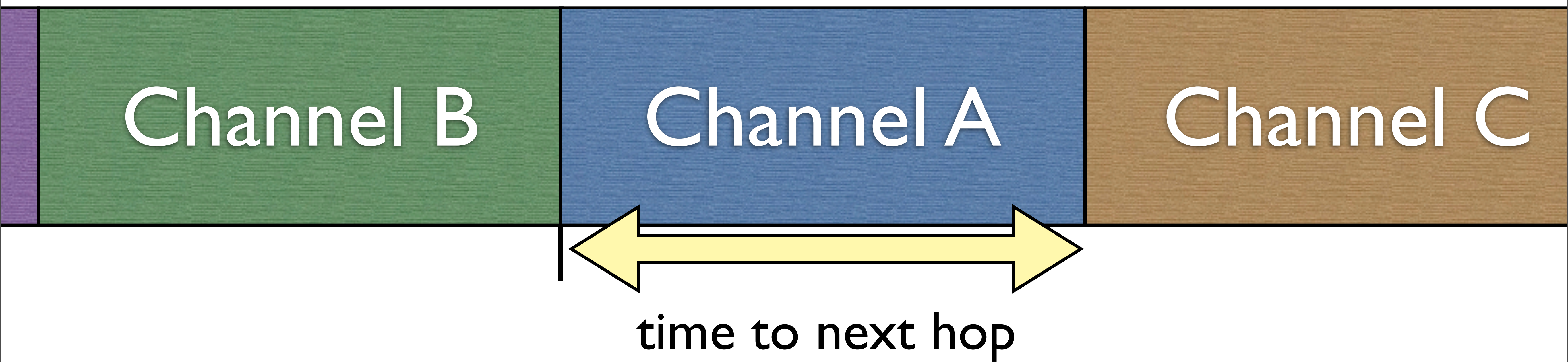

Node 1
10.0.0.1
Node 16
10.0.0.16
Node 2
10.0.0.2
Node 0
10.0.0.20
Node 5
10.0.0.5
Node 3
10.0.0.3
Node 4
10.0.0.4
UDP Video

halfMac

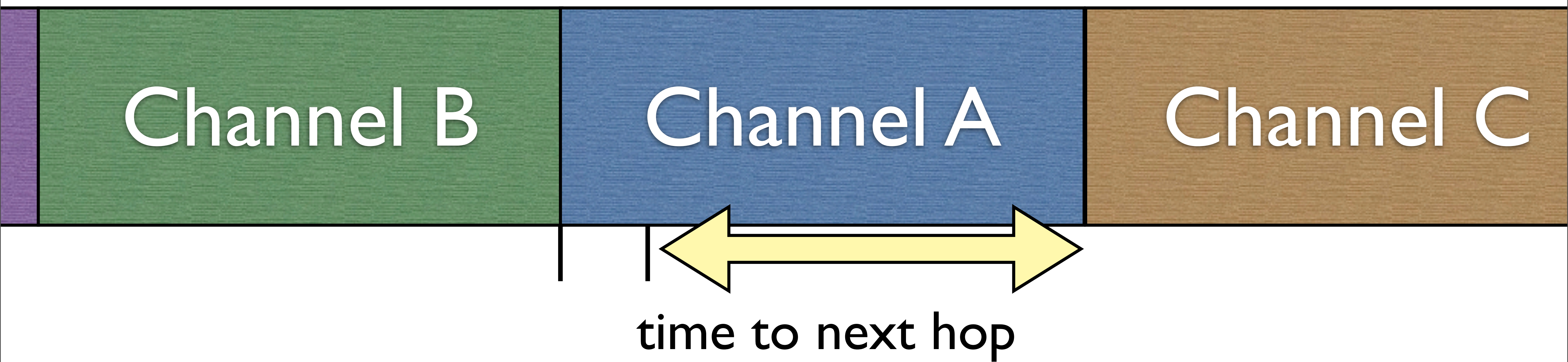


Node 5

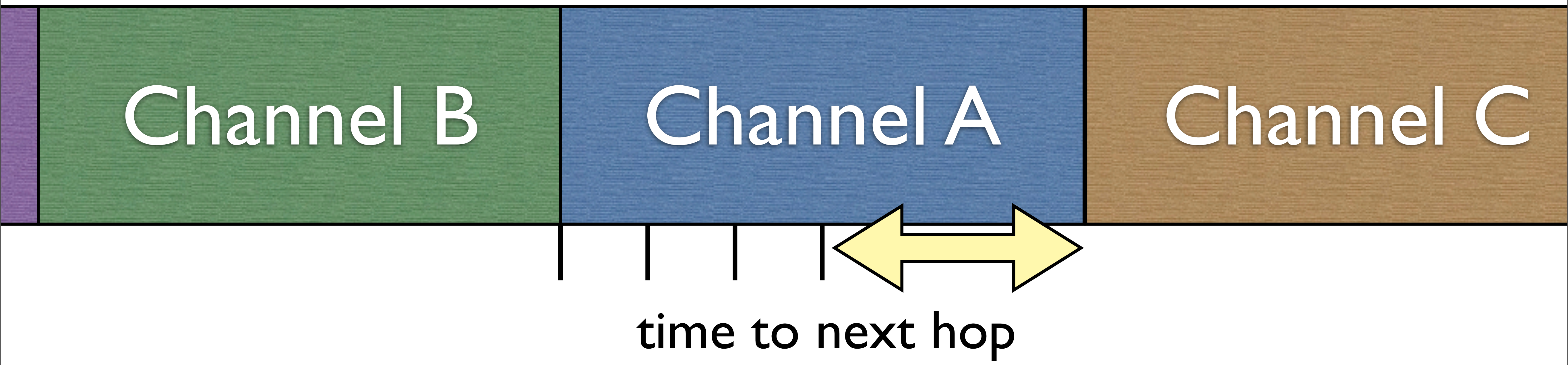
cogMAC



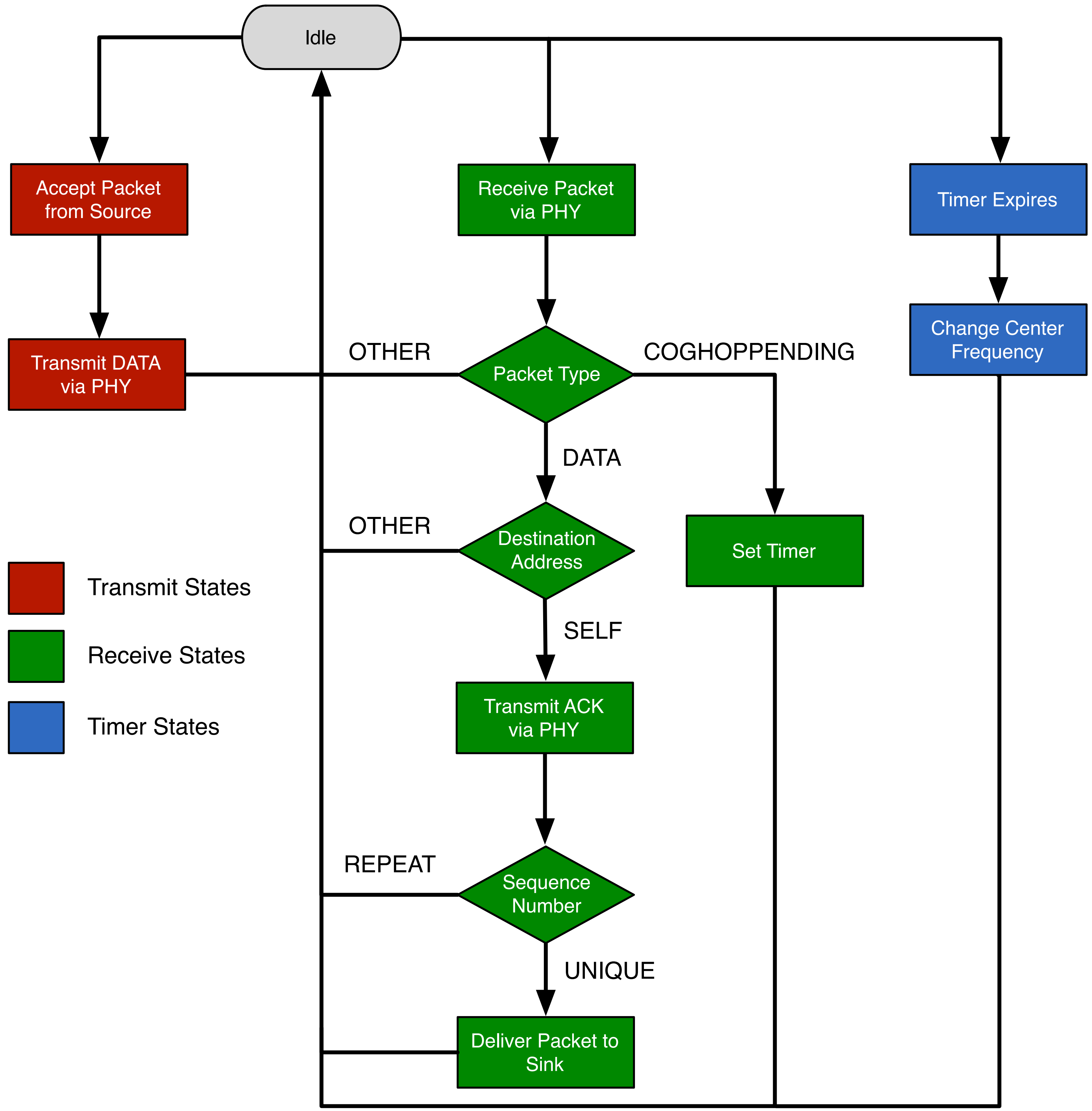
cogMAC



cogMAC



cogMAC


Idle
Accept Packet from Source
Transmit DATA via PHY
Receive Packet via PHY
Timer Expires
Change Center Frequency
OTHER
Packet Type
COGHOPPENDING
DATA
OTHER
Destination Address
Set Timer
SELF
Transmit ACK via PHY
REPEAT
Sequence Number
UNIQUE
Deliver Packet to Sink
Transmit States
Receive States
Timer States

Logistics

- WARPMAC API: http://warp.rice.edu/WARP_API
- Contacting us
    - Support & technical questions
        - http://warp.rice.edu/forums/
    - Hardware sales
        - Mango Communications (http://mangocomm.com/)