Networking on WARP

Chris Hunter & Patrick Murphy
Rice University

WARP Workshop
December 2, 2007

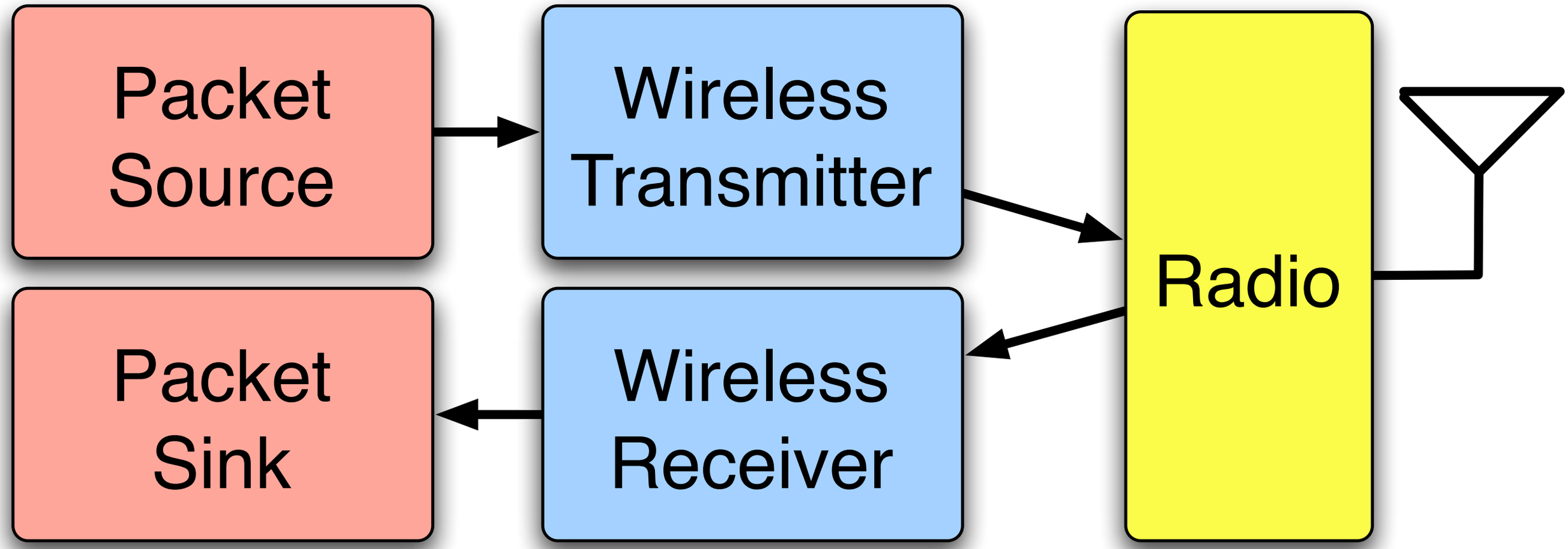
warp.rice.edu

Today's Agenda

- Questions from yesterday?
- Networking on WARP lecture
- Lab 4 - Simple "MAC" layer
- Lab 5 - Unidirectional MAC
- Lab 6 - Channel-hopping MAC
- Workshop wrap-up

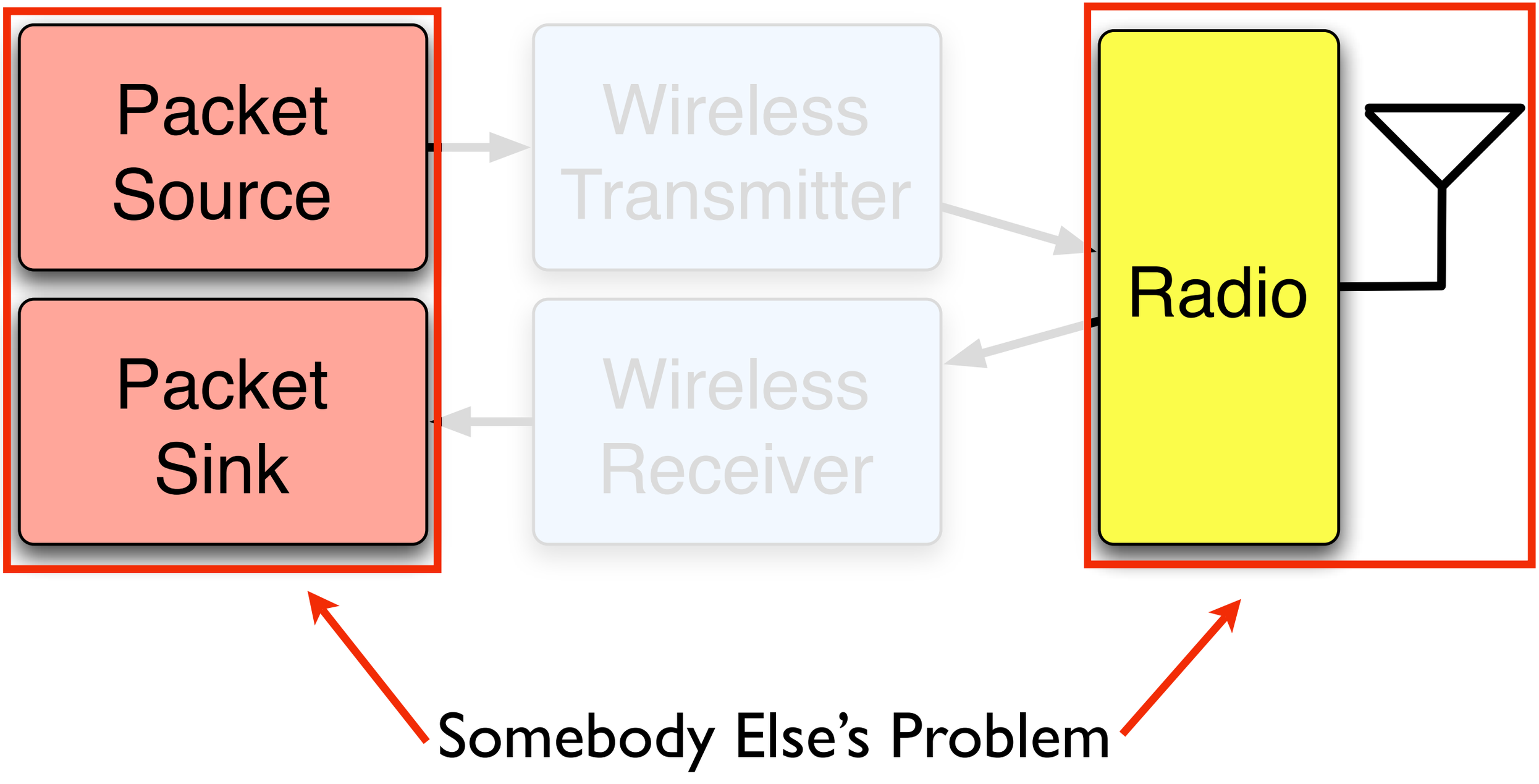
Physical Layer Basics

Simple Wireless Node



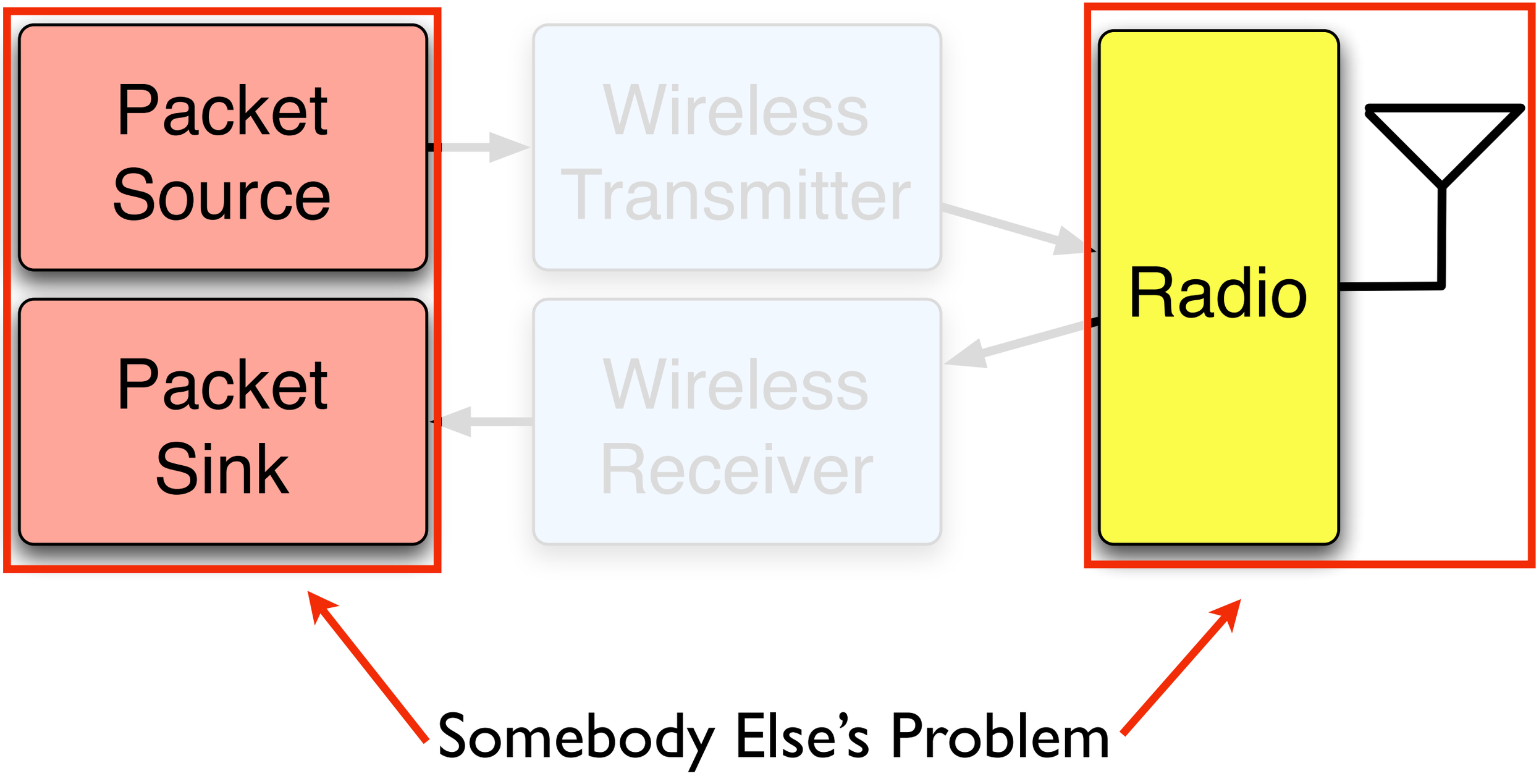
Physical Layer Basics

Simple Wireless Node



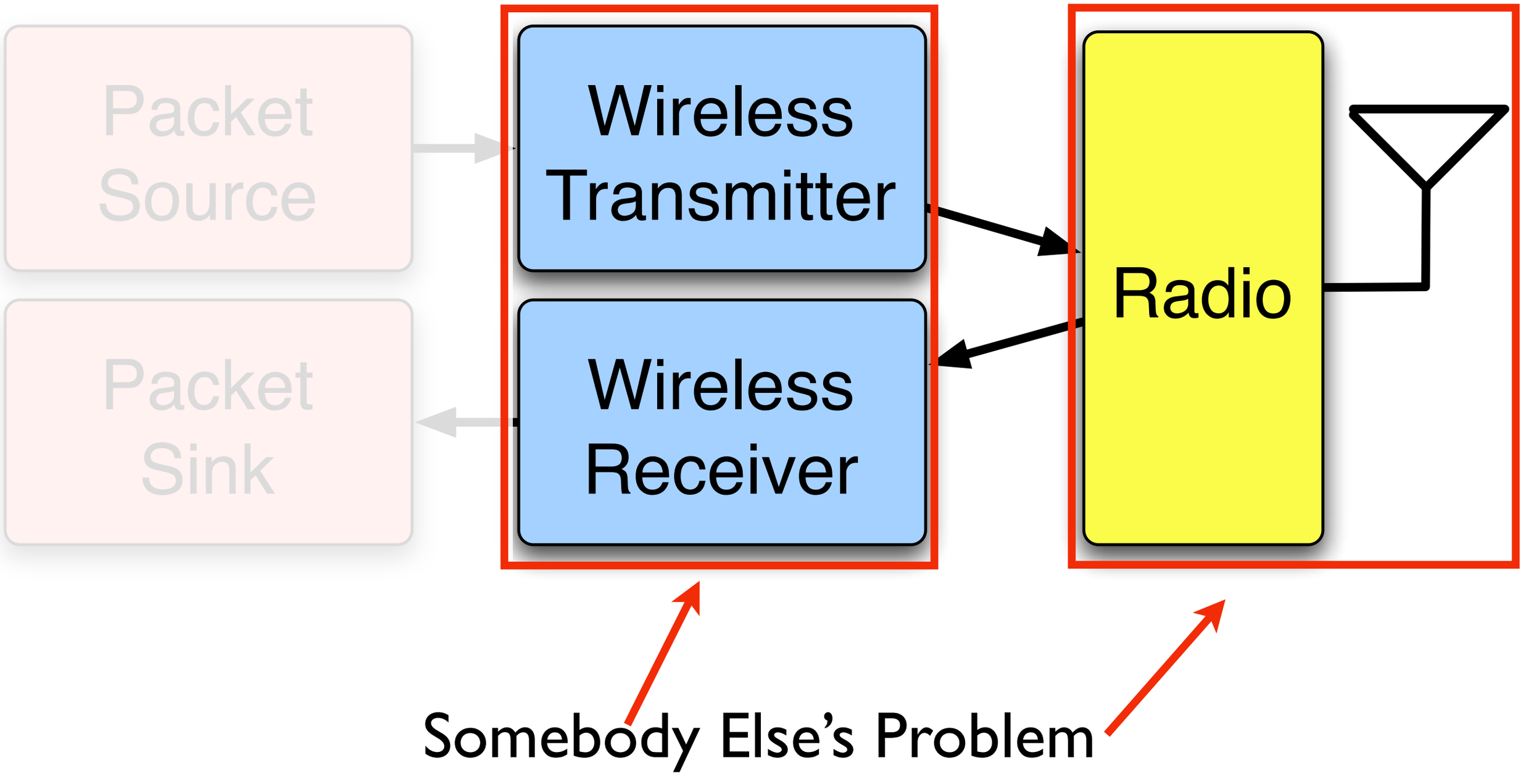
Network Layer Basics

Simple Wireless Node



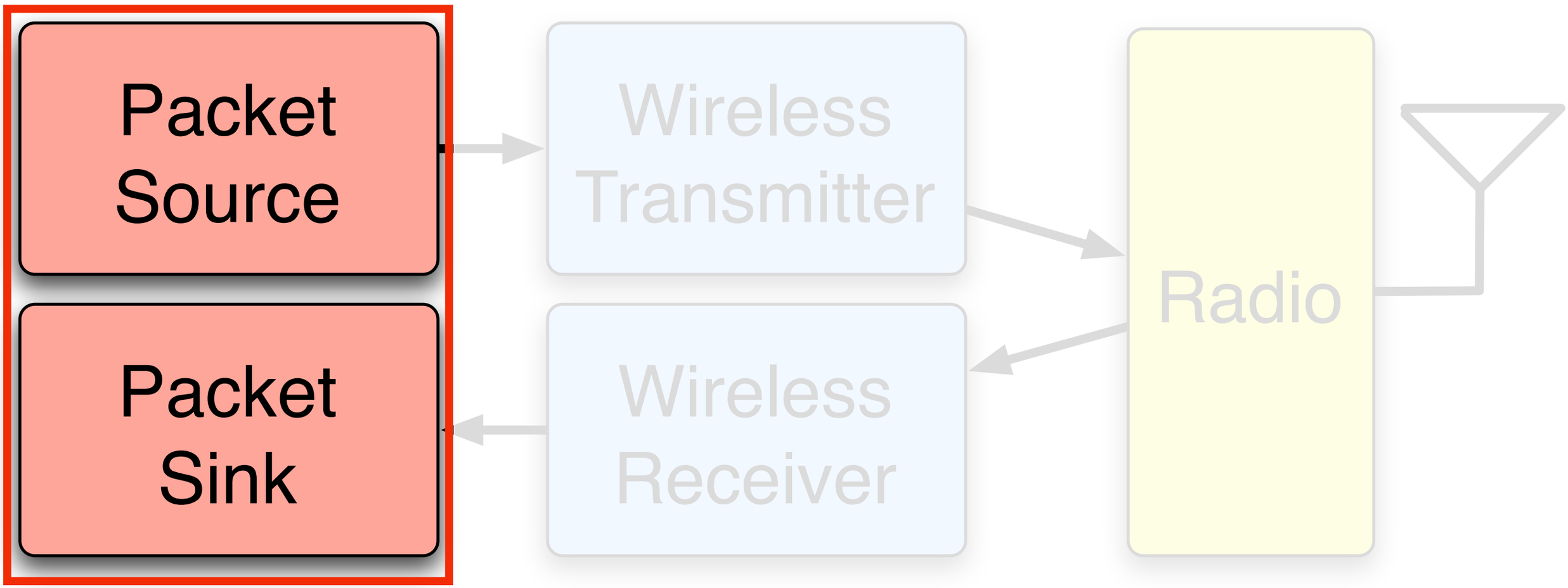
Network Layer Basics

Simple Wireless Node



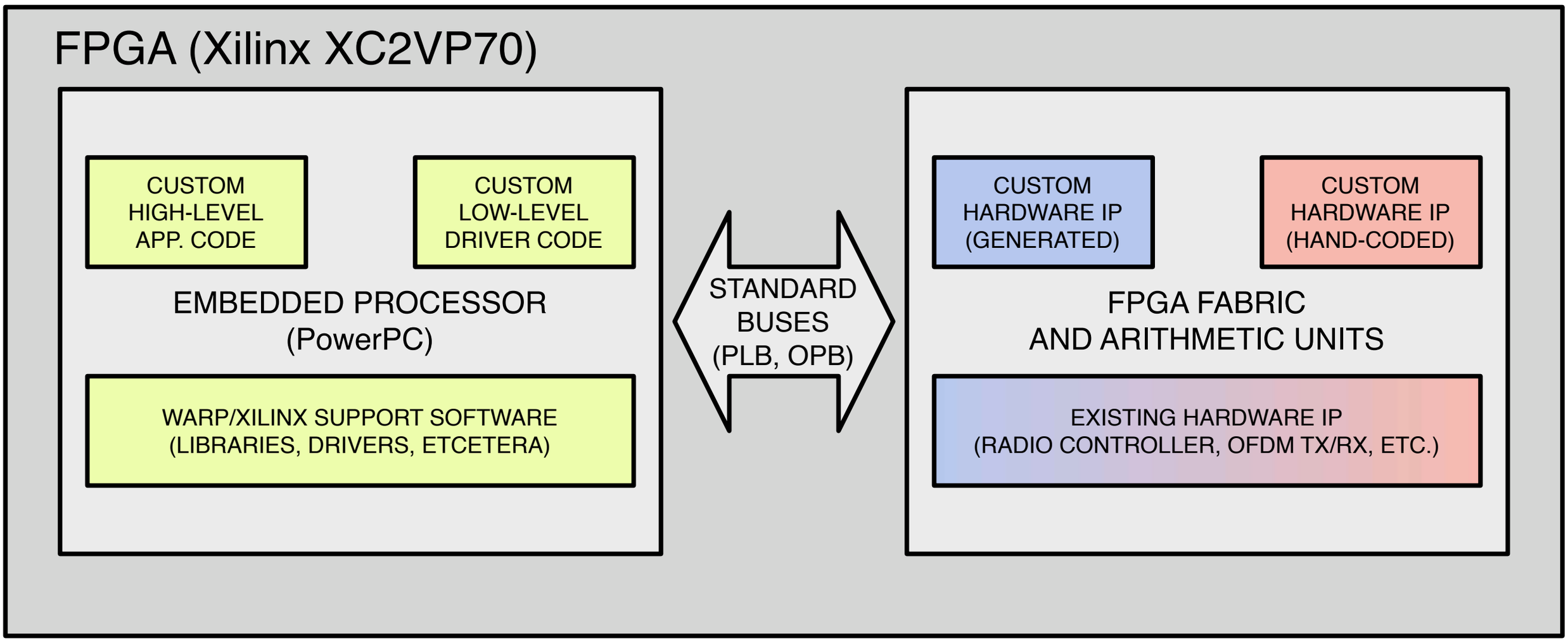
Network Layer Basics

Simple Wireless Node

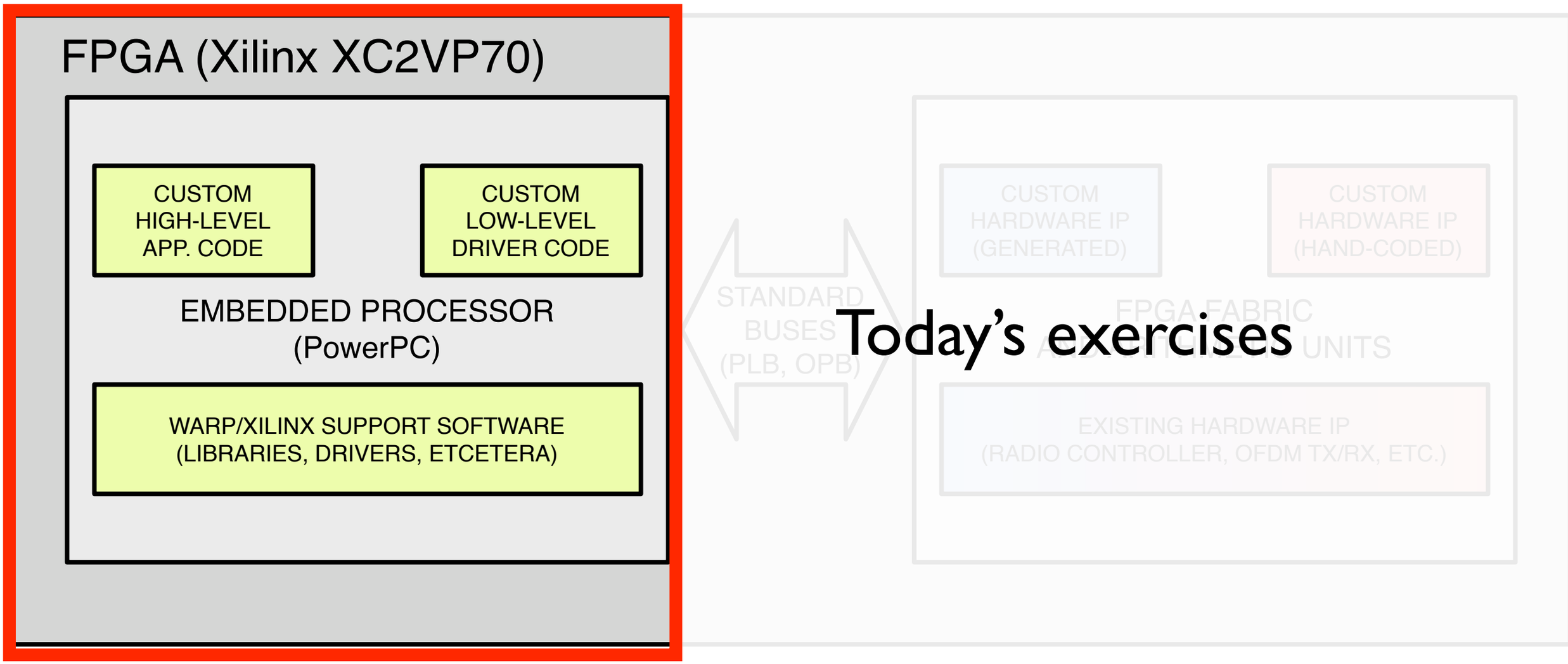


Targeting WARP Hardware

(Understanding the Development Environment)

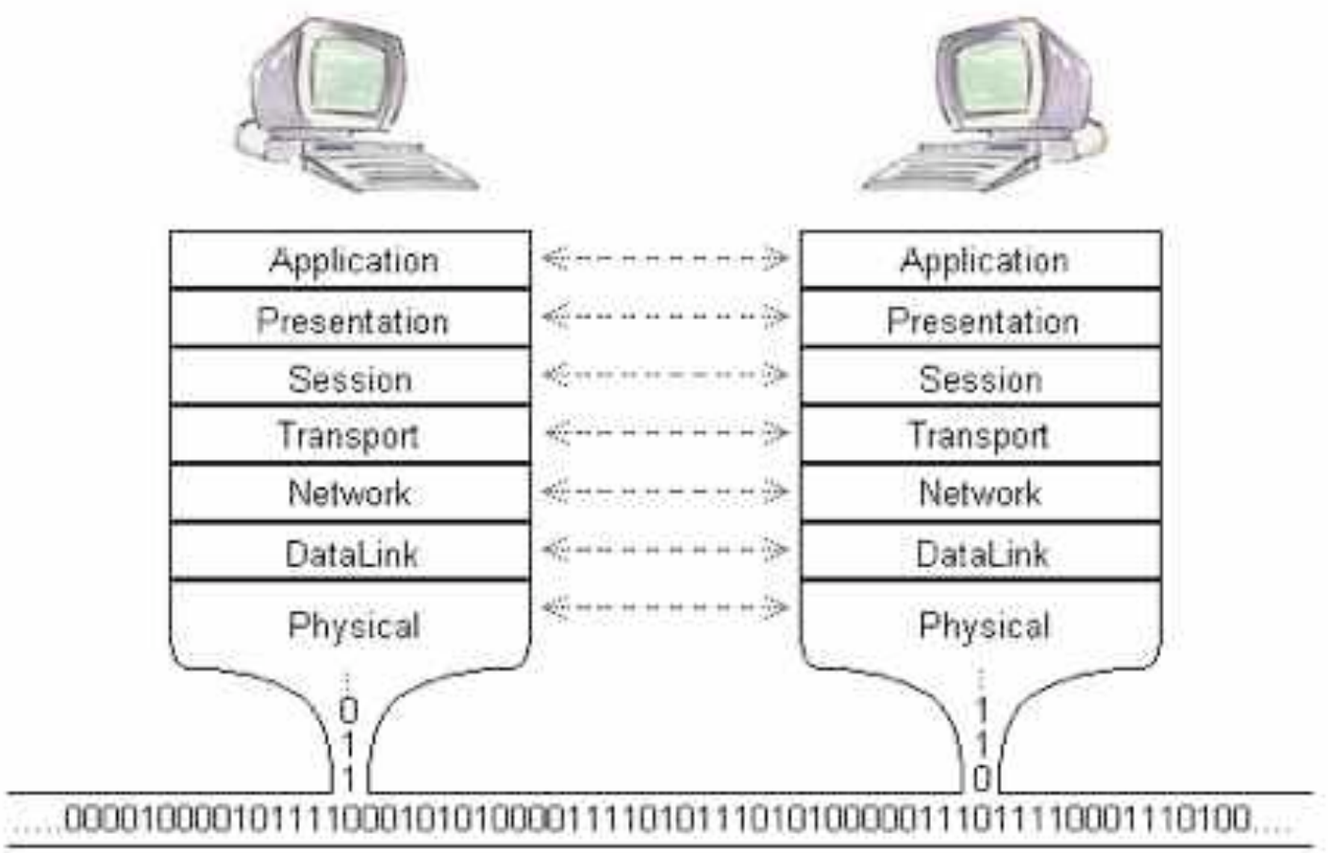


Targeting WARP Hardware
(Understanding the Development Environment)



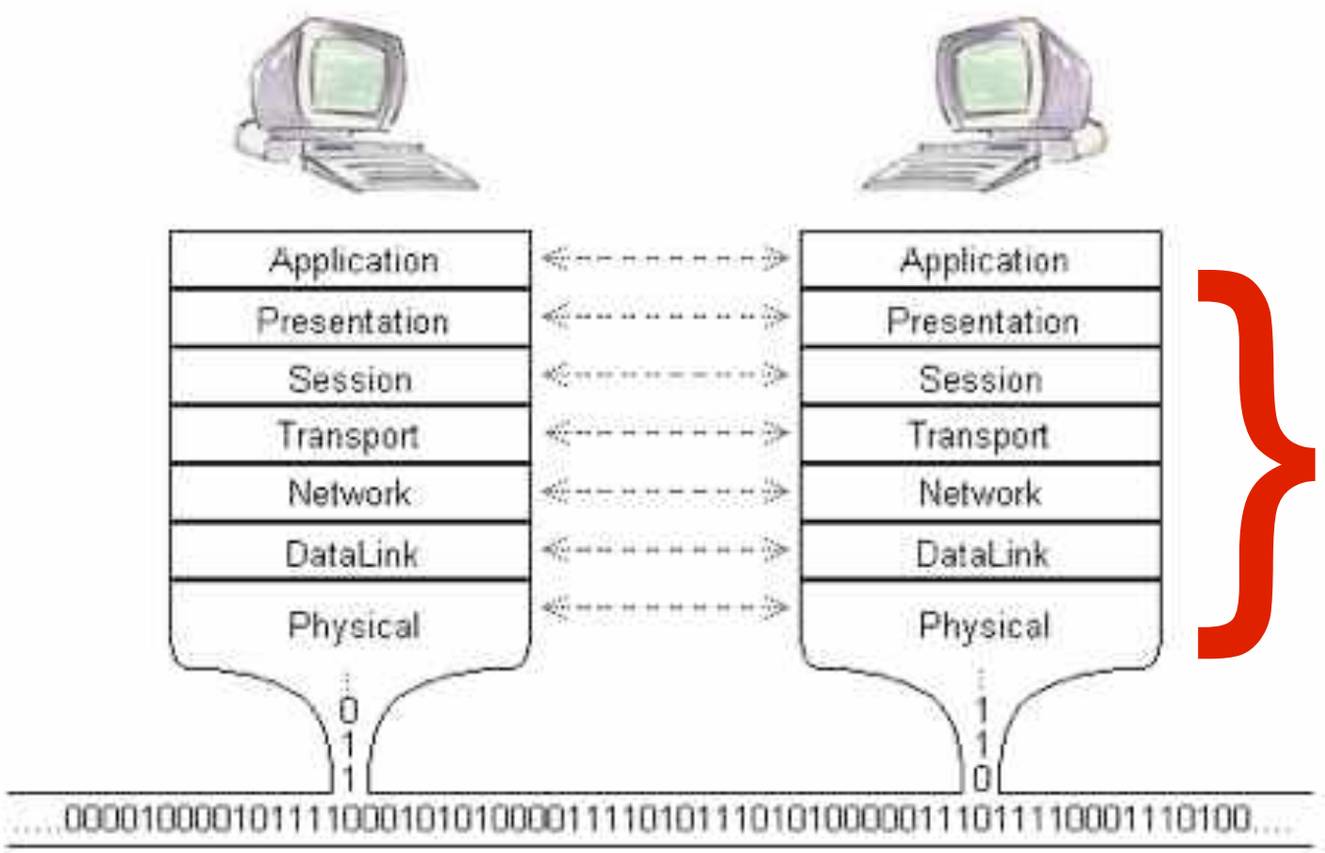
Today's exercises

Some Perspective - The OSI Model



Source: http://upload.wikimedia.org/wikipedia/en/f/ff/Osi_model_trad.jpg

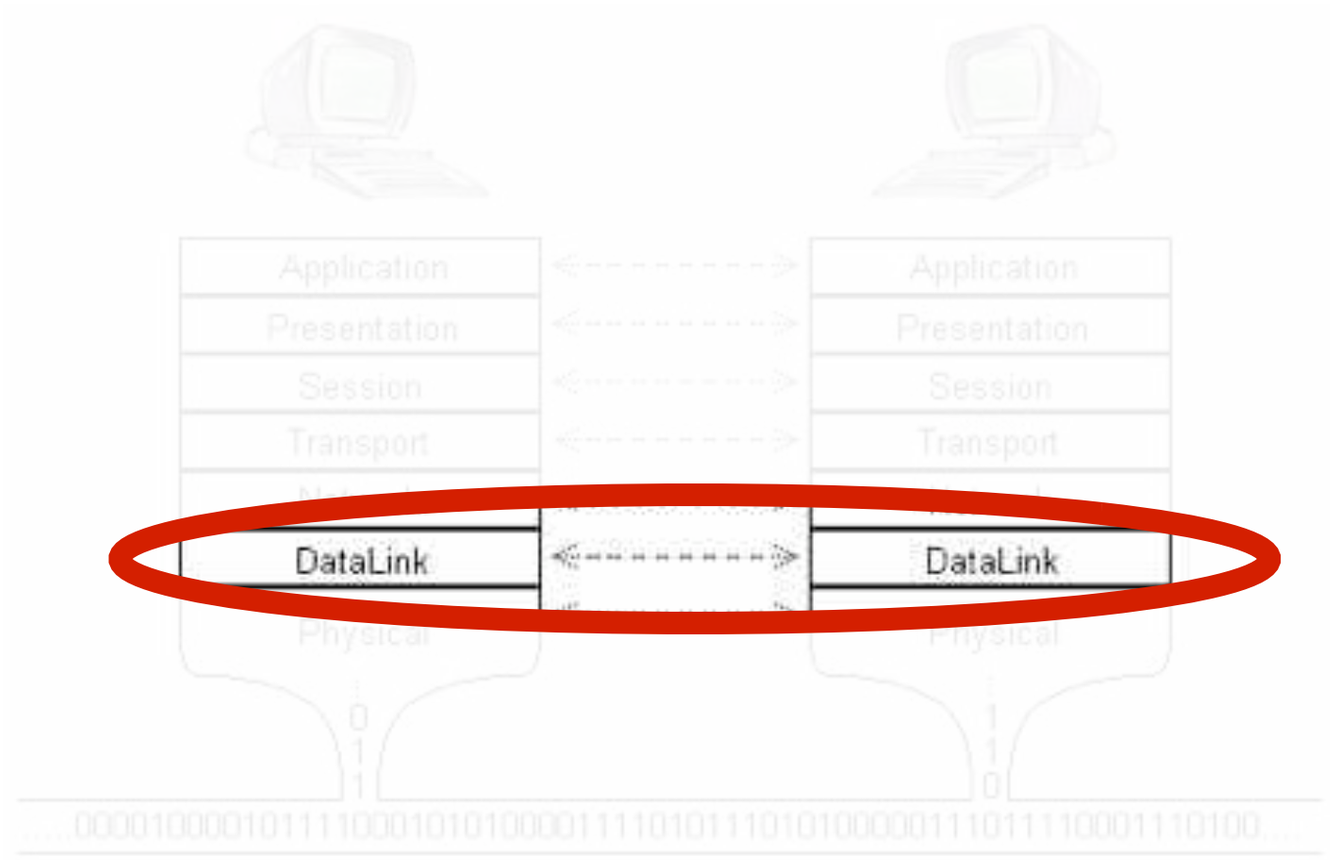
Some Perspective - The OSI Model



WARP can do them all

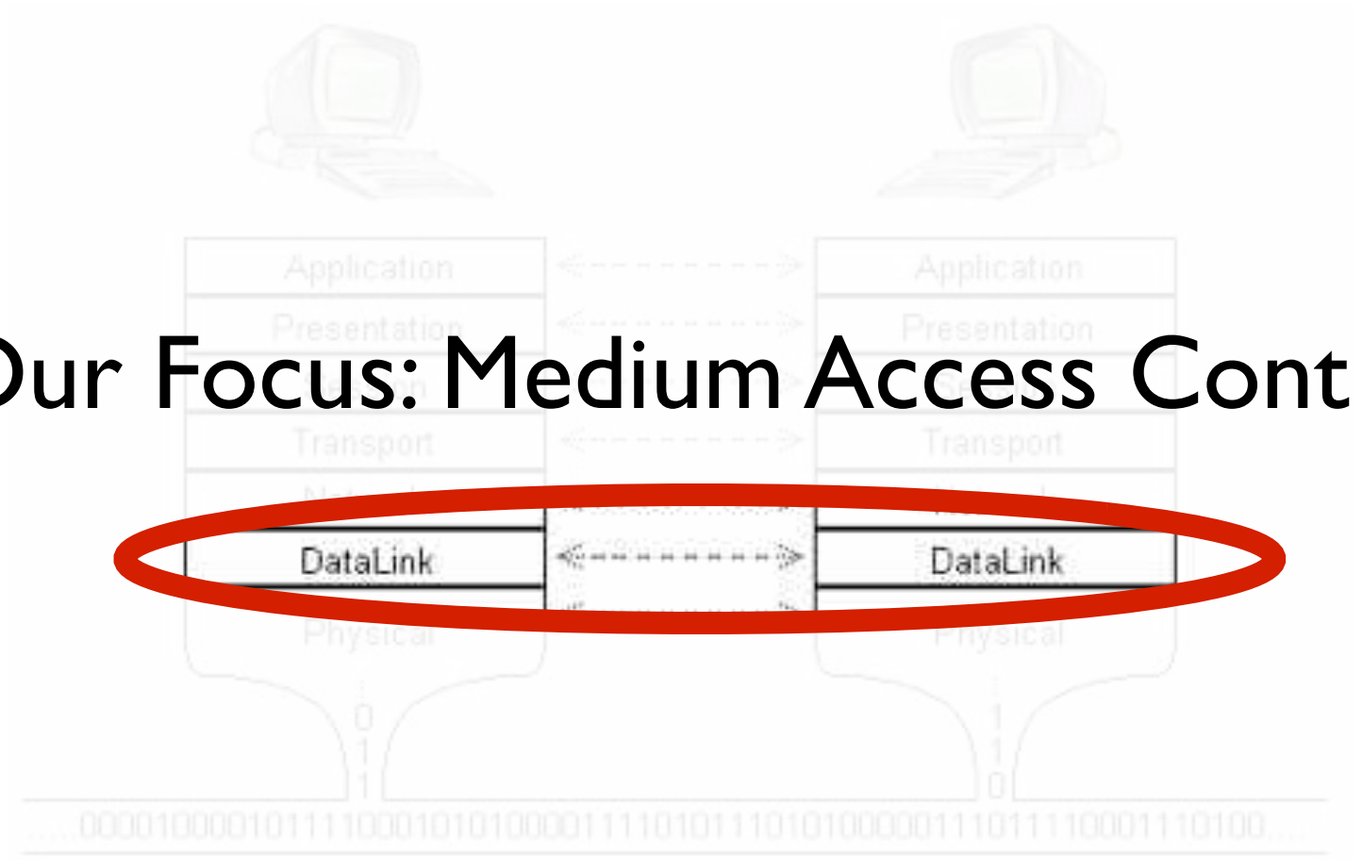
Source: http://upload.wikimedia.org/wikipedia/en/f/ff/Osi_model_trad.jpg

The OSI Model



Source: http://upload.wikimedia.org/wikipedia/en/f/ff/Osi_model_trad.jpg

The OSI Model

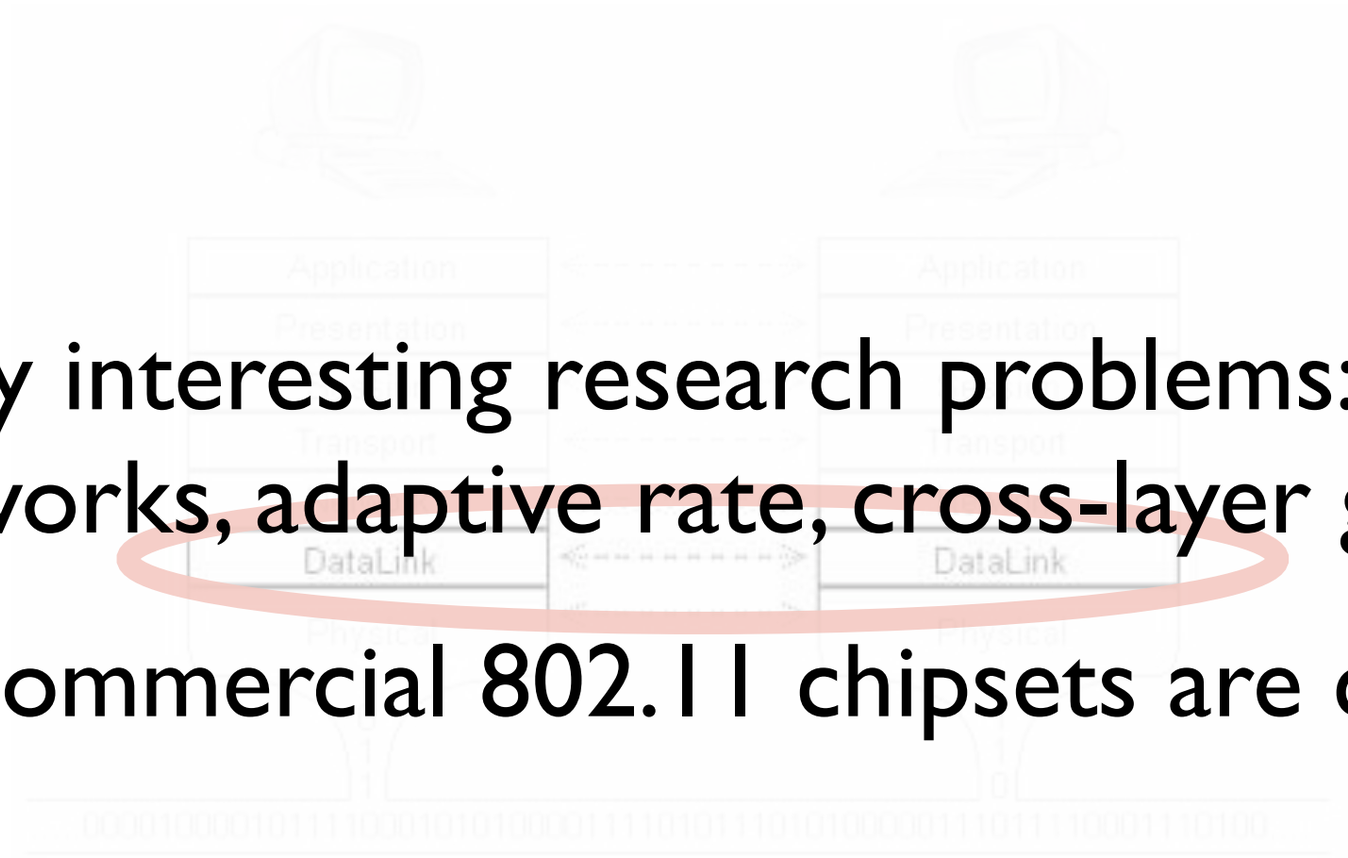


Our Focus: Medium Access Control

Source: http://upload.wikimedia.org/wikipedia/en/f/ff/Osi_model_trad.jpg

The OSI Model

- Why?
- Many interesting research problems: mesh networks, adaptive rate, cross-layer gains, etc.
- All commercial 802.11 chipsets are closed



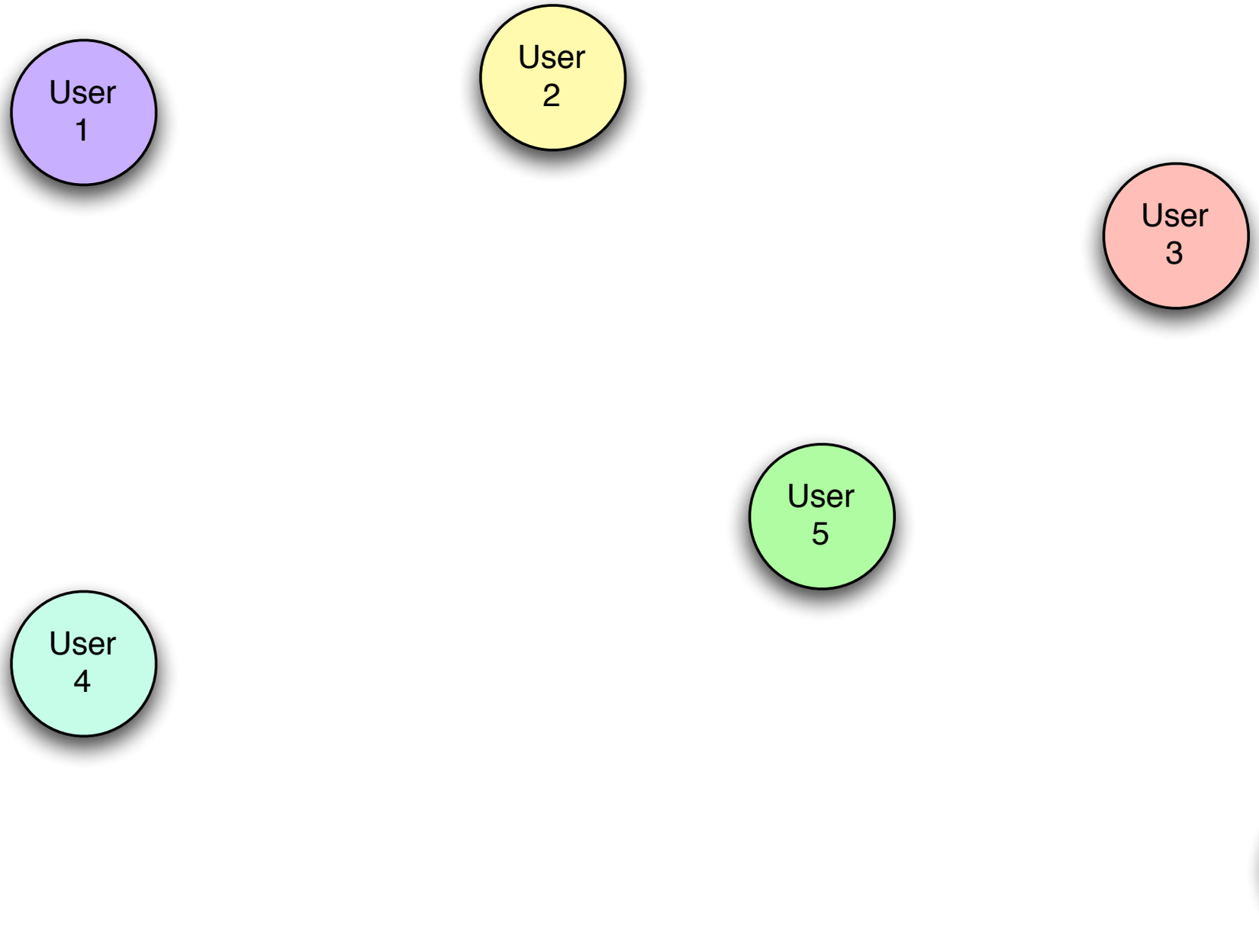
Source: http://upload.wikimedia.org/wikipedia/en/f/ff/Osi_model_trad.jpg

Outline

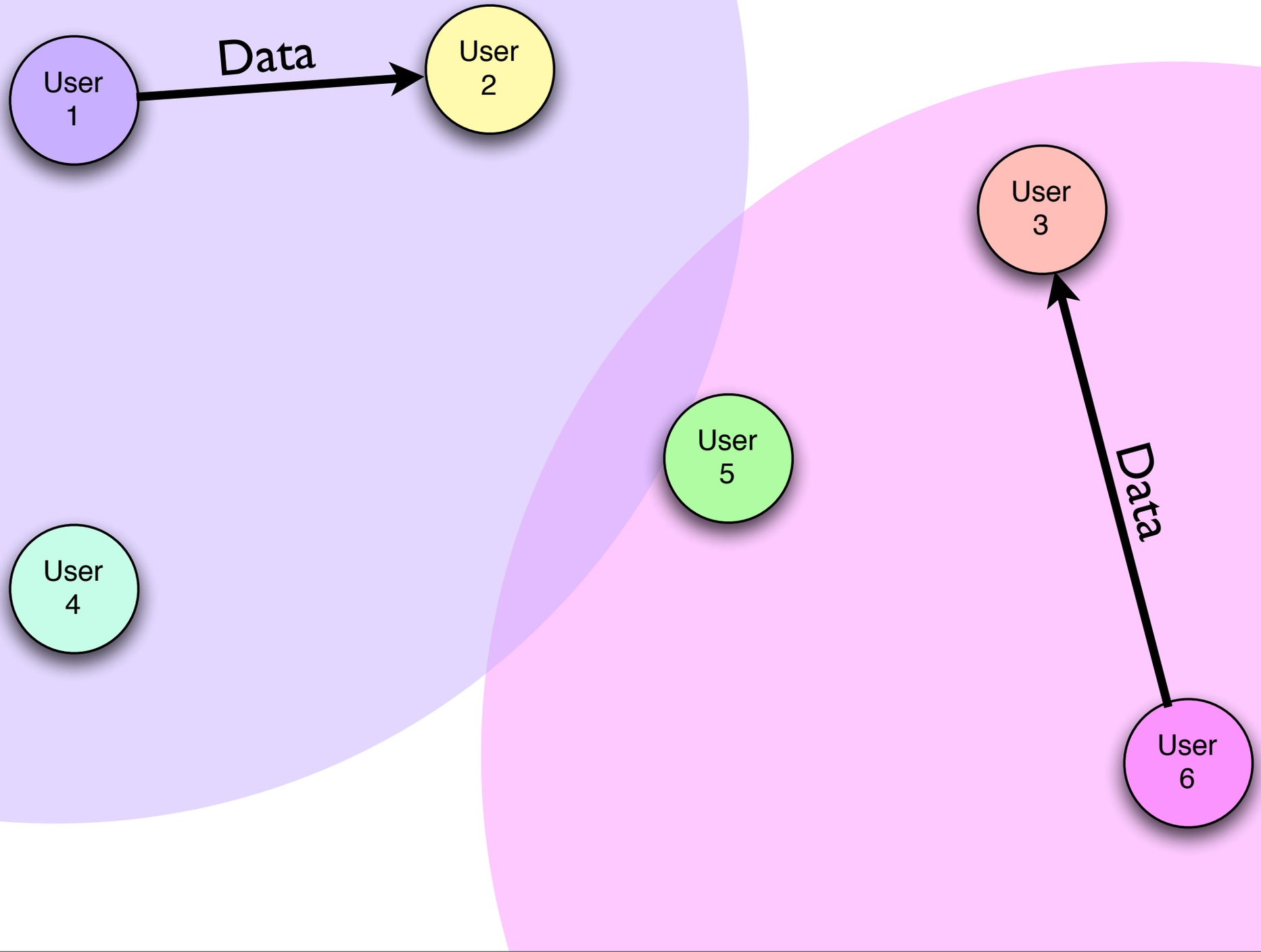
- Overview of Medium Access Control
- Design Realization
- WARPMAC Framework
- Detailed Example
- Lab Exercises

Medium Access Control Overview

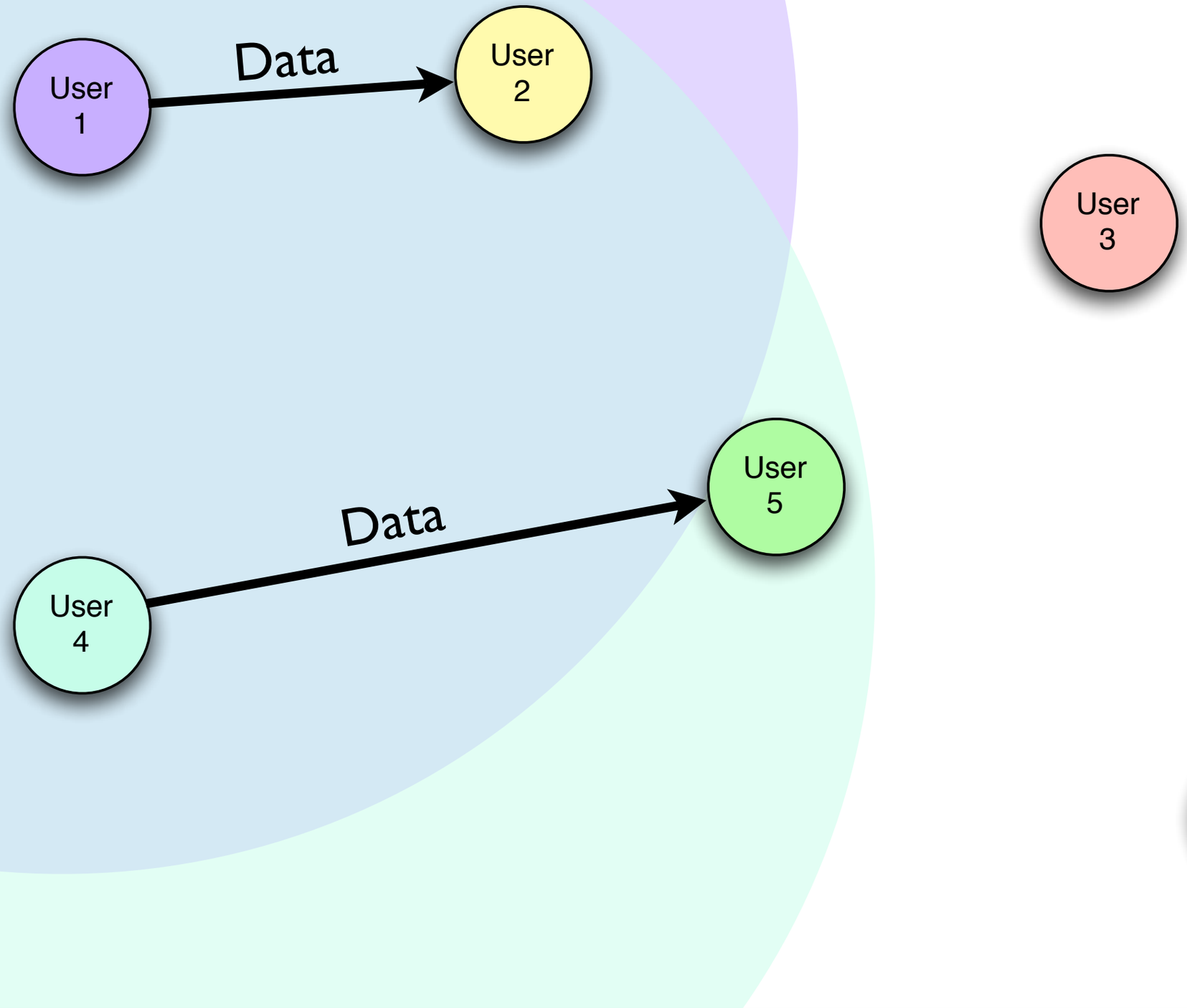
What is a MAC?



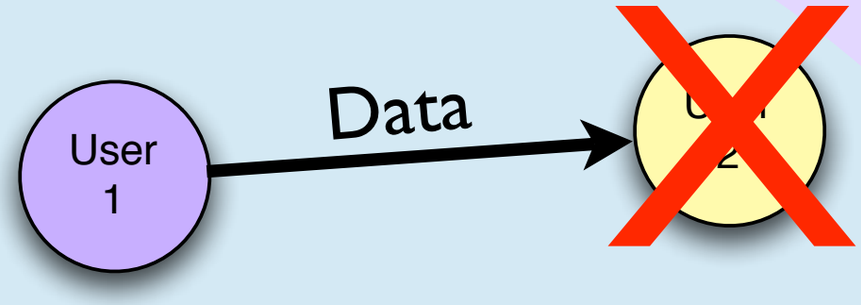
What is a MAC?

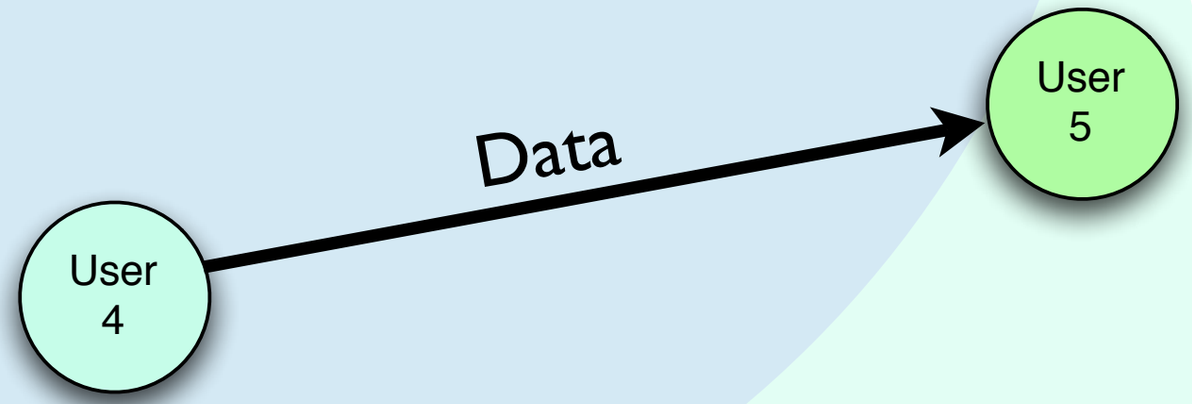


What is a MAC?

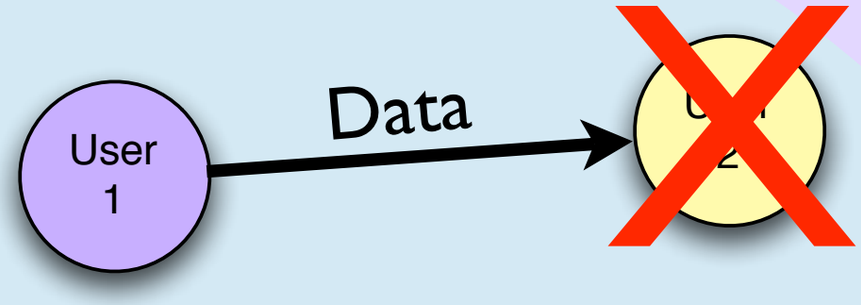


What is a MAC?

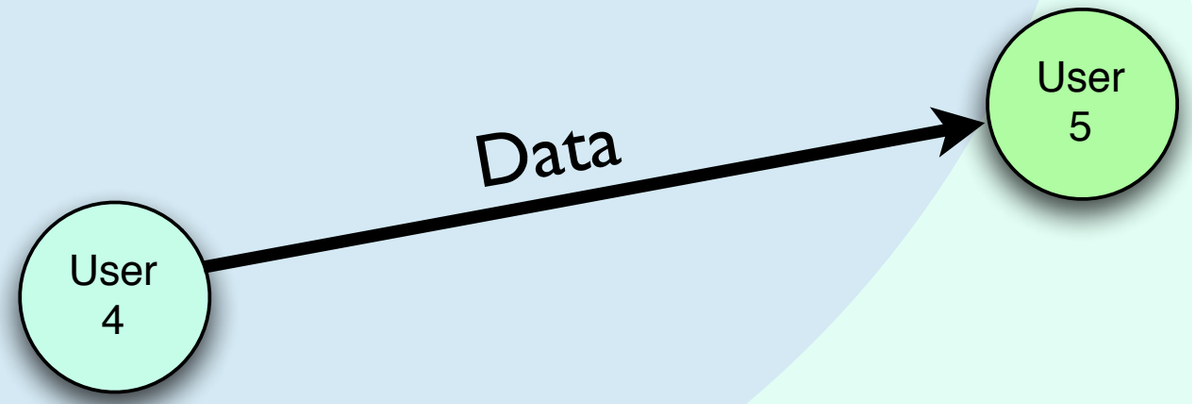




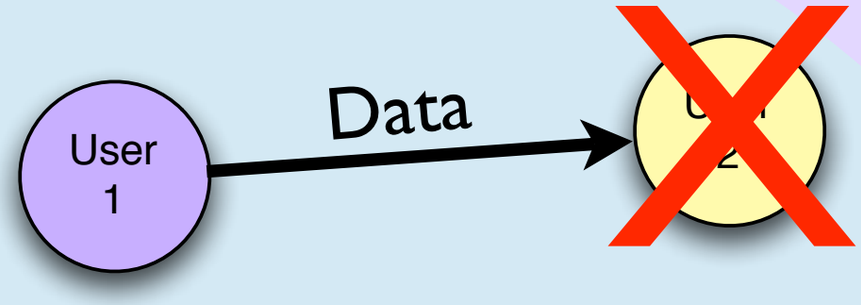
What is a MAC?



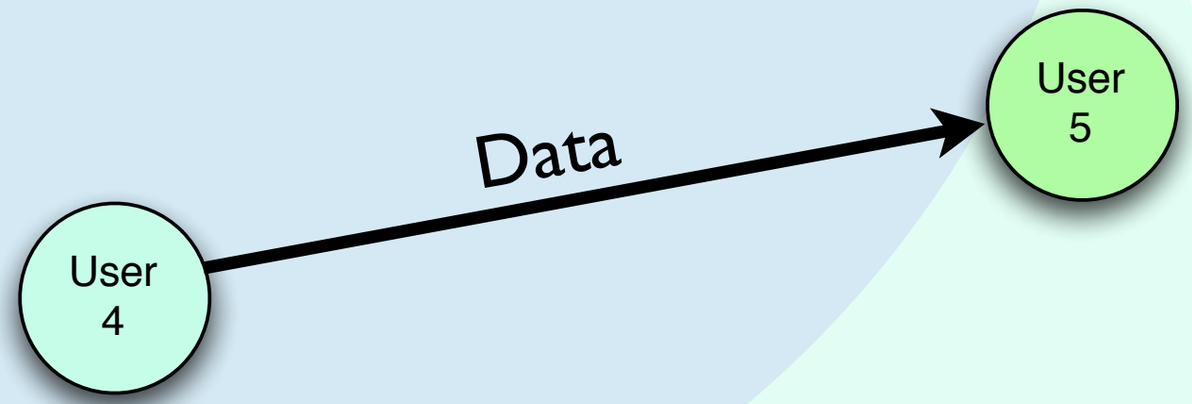
Received a jumbled packet... infer a packet collision



What is a MAC?

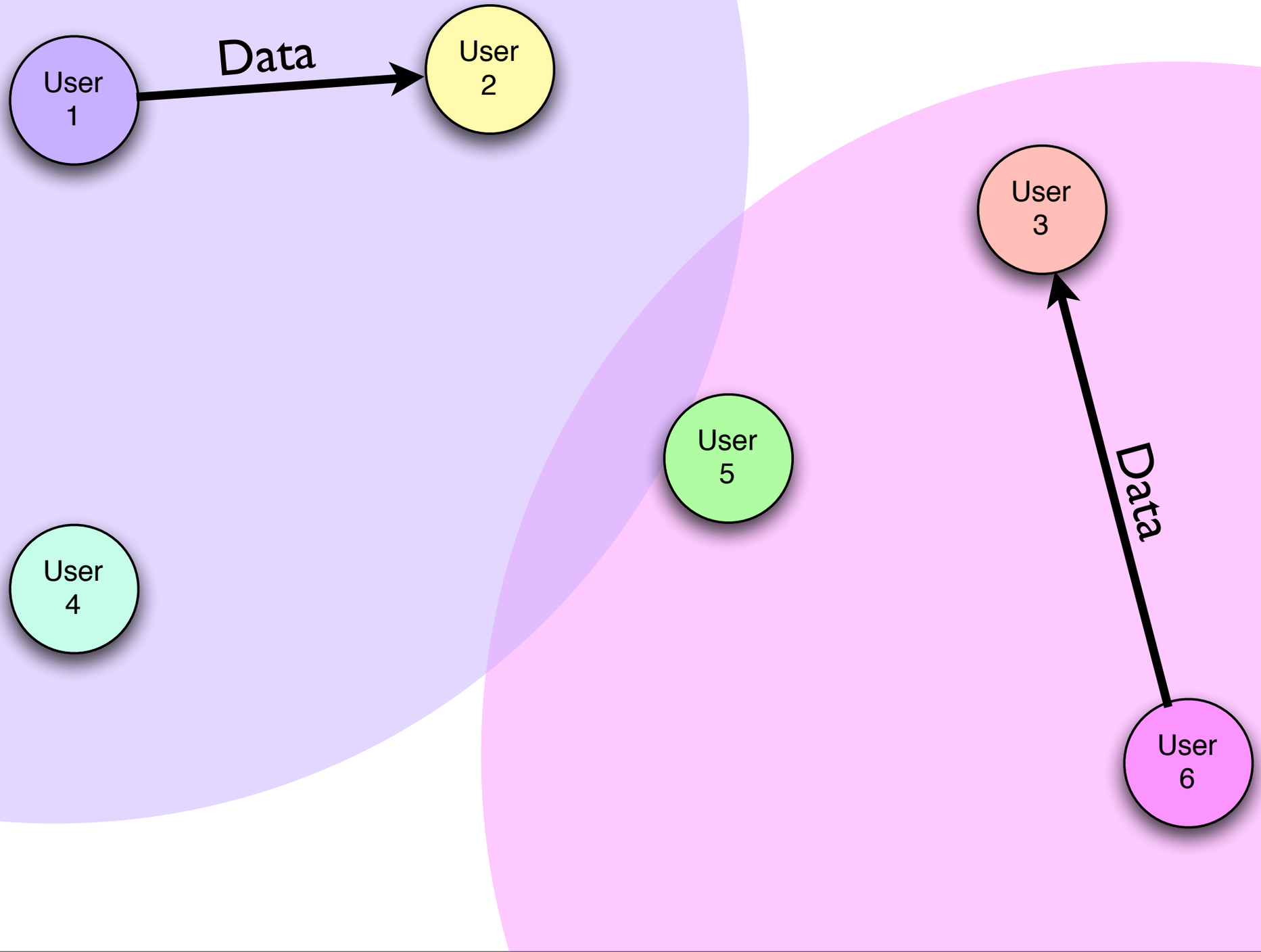


Received a jumbled packet... infer a packet collision

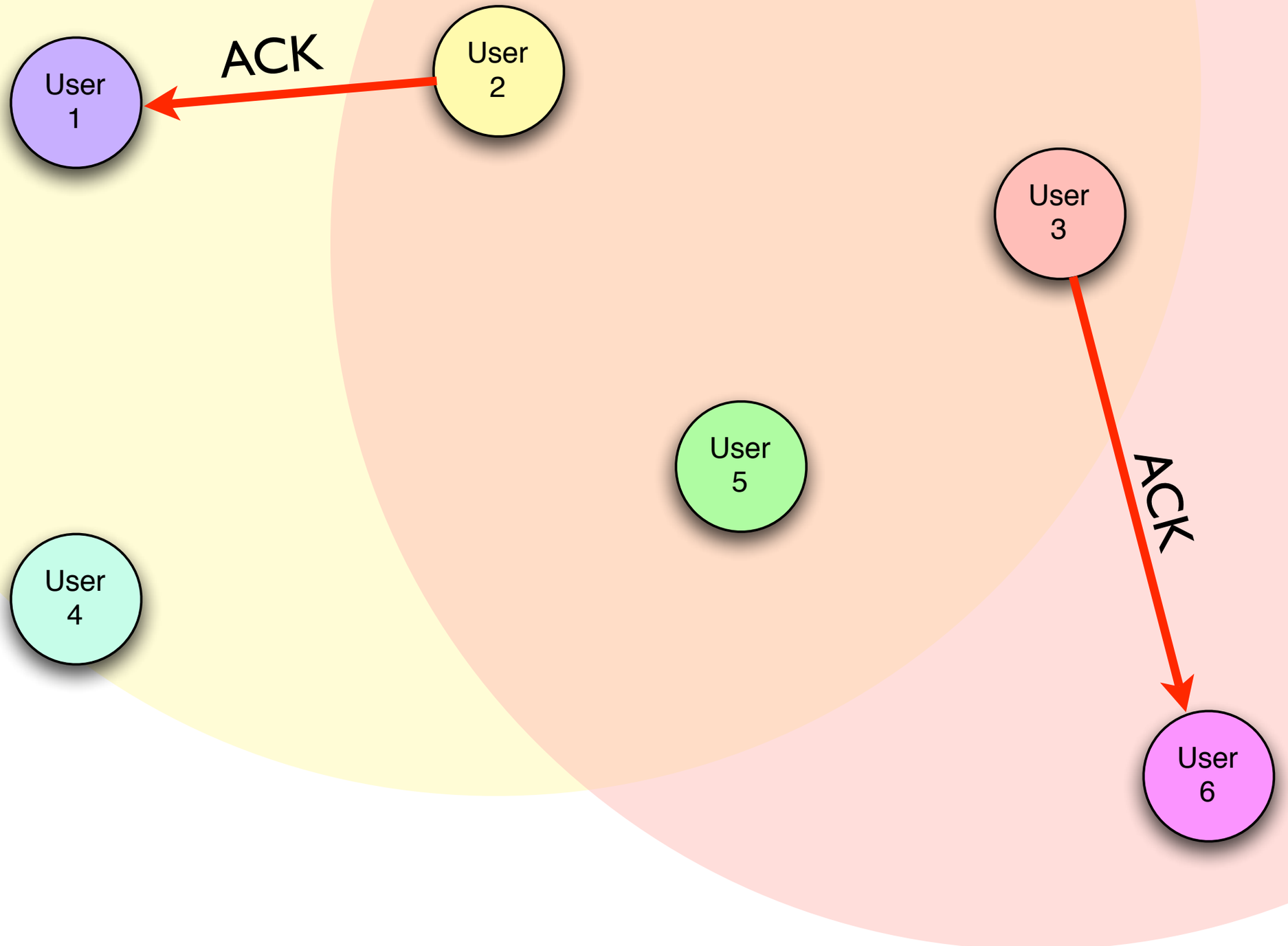


What if we ACK every transmit, and retransmit when we receive no ACK?

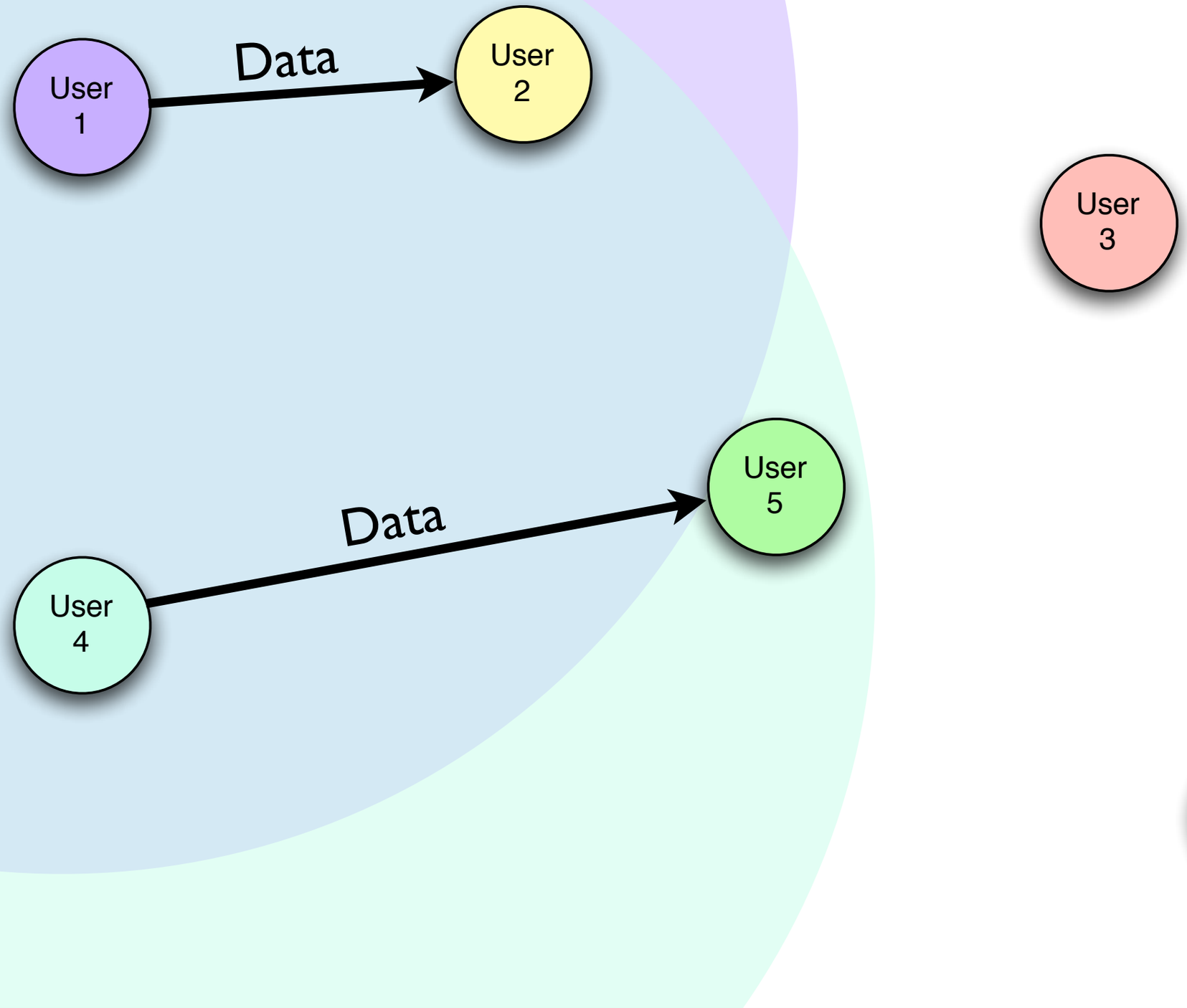
What is a MAC?



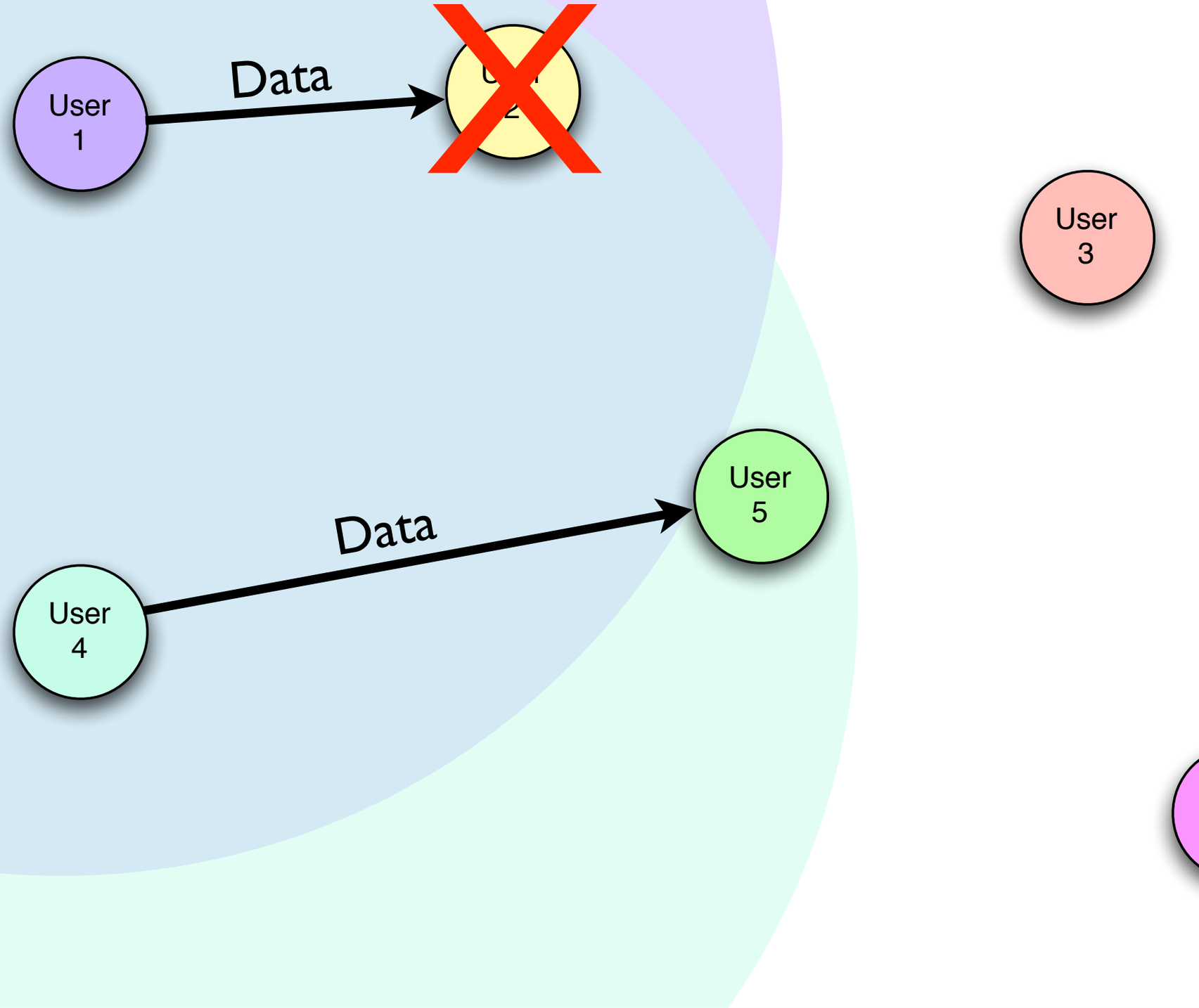
What is a MAC?



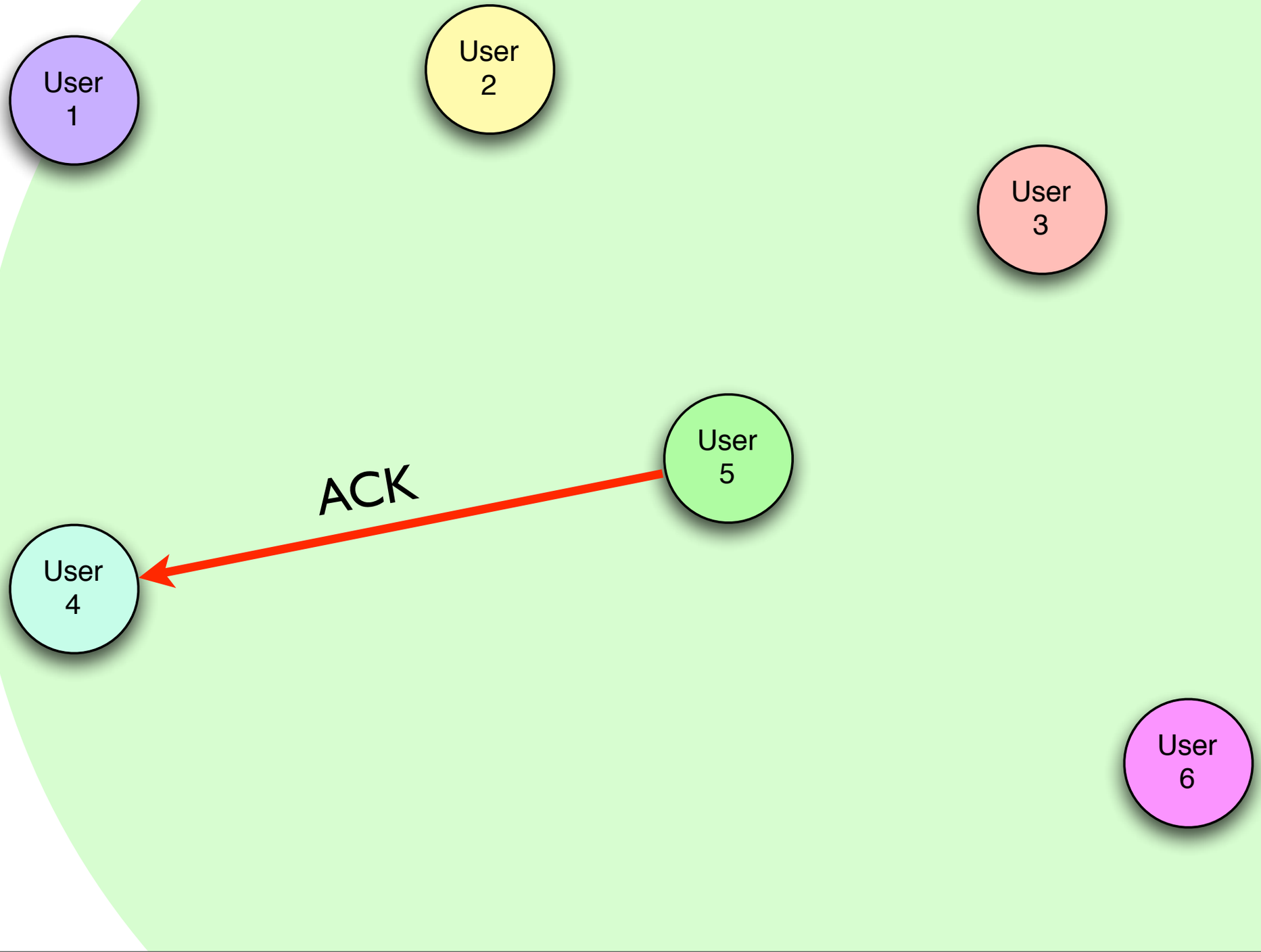
What is a MAC?



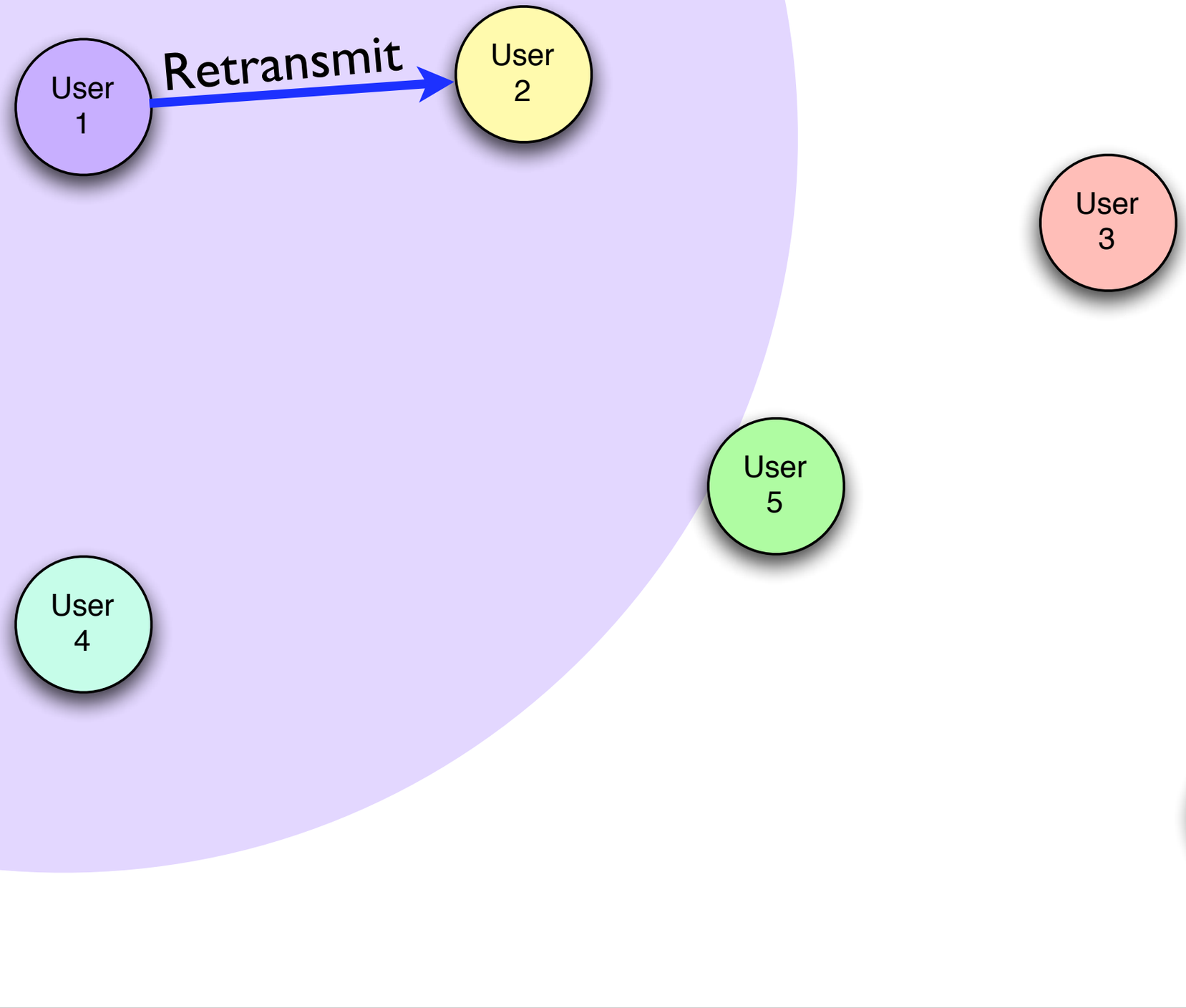
What is a MAC?



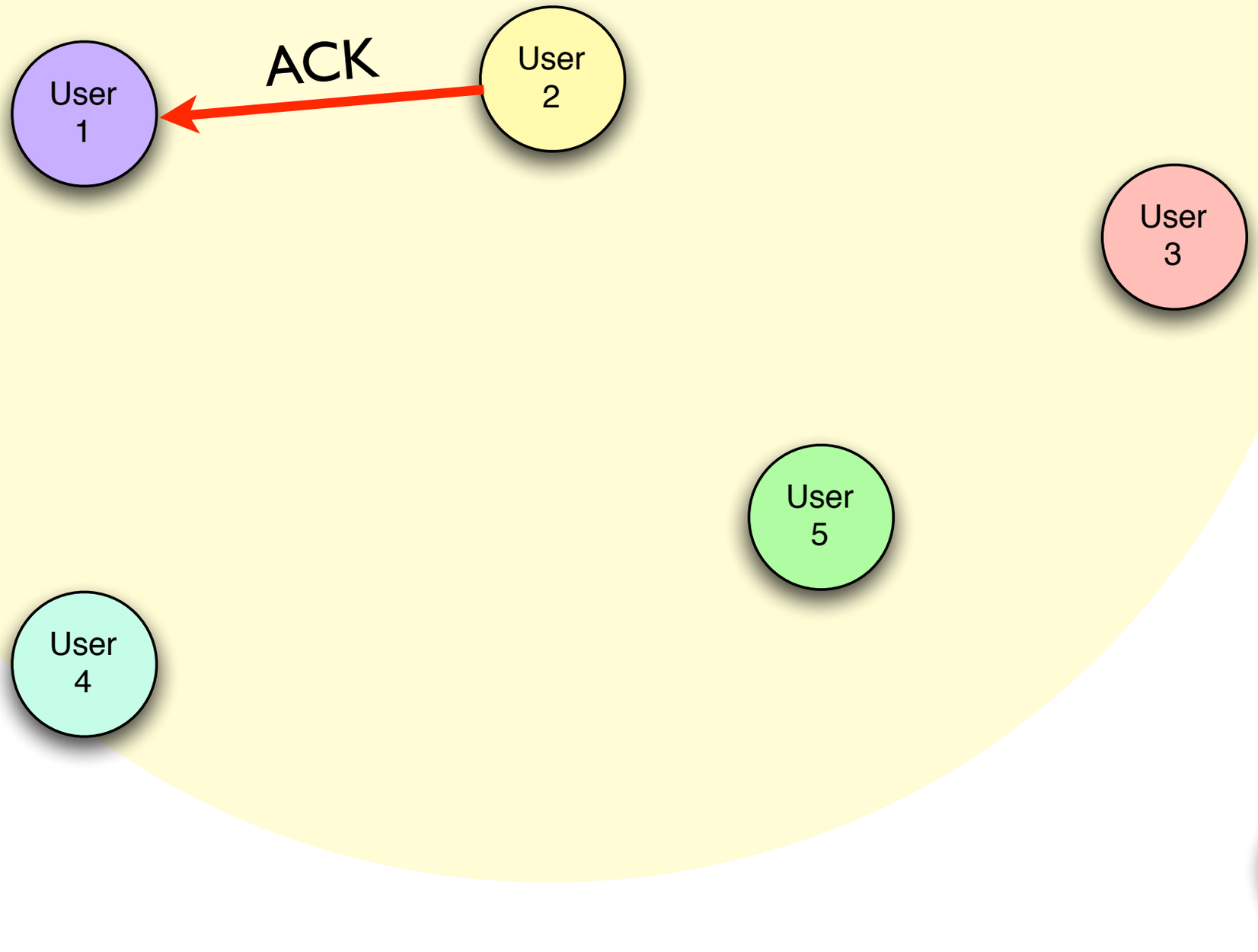
What is a MAC?



What is a MAC?



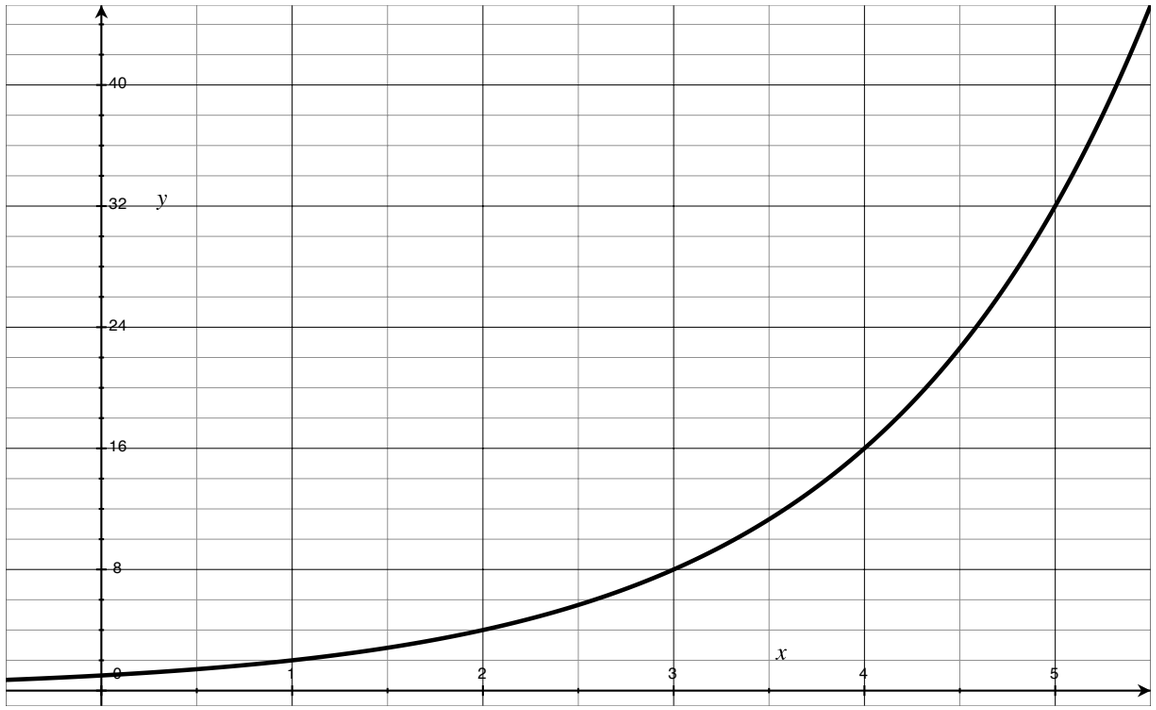
What is a MAC?



Random Backoffs

- **PROBLEM:** Retransmissions can collide *ad infinitum!*

- **SOLUTION:** Wait a random amount of time before a retransmit



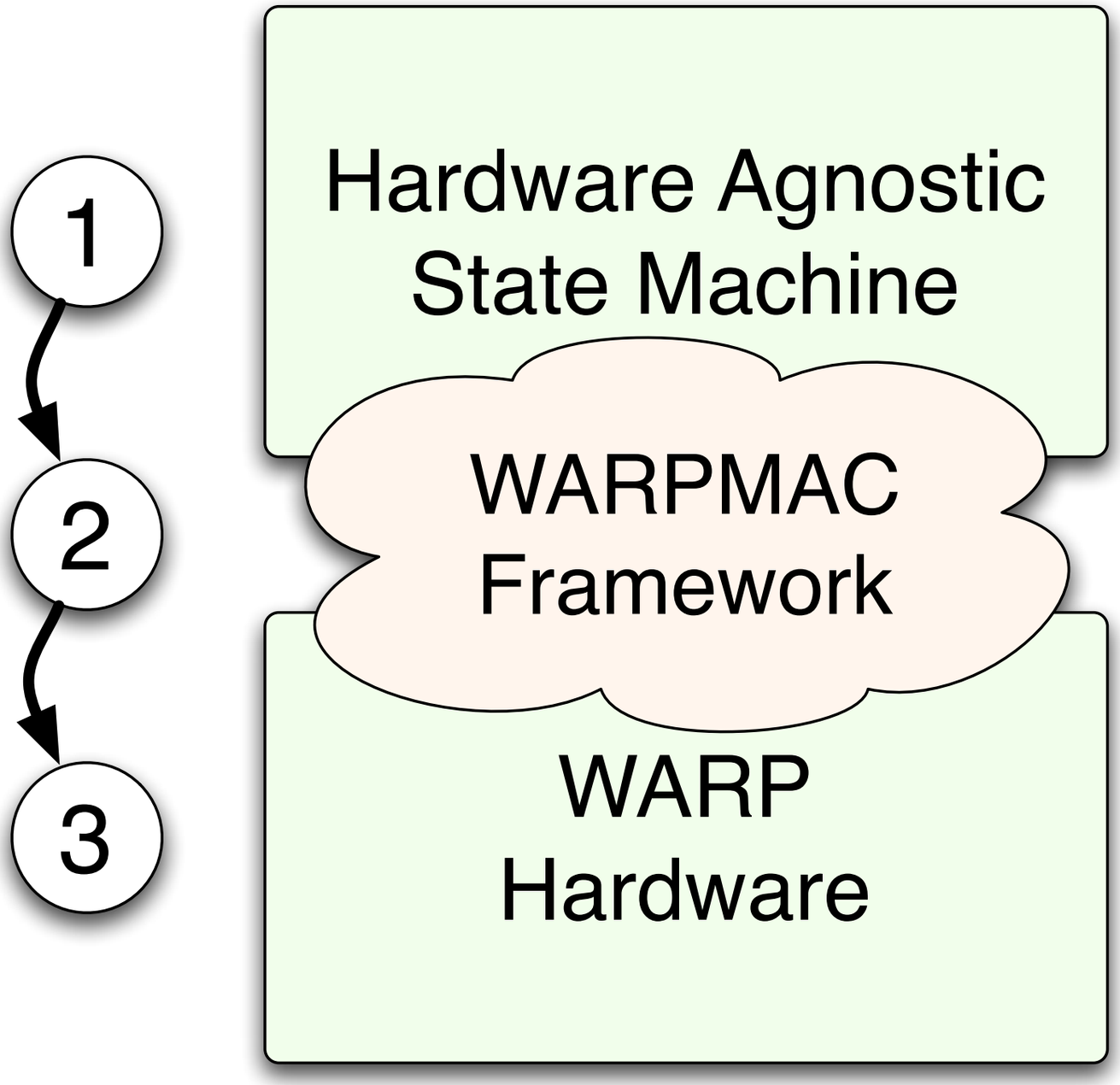
Contention Window increases over time

Other Important Details

- Carrier Sense Multiple Access (CSMA)
  - Listen to the medium before sending
- Request to Send / Clear to Send (RTS/CTS)
  - “Reserve” the medium with a short packet before sending a long one

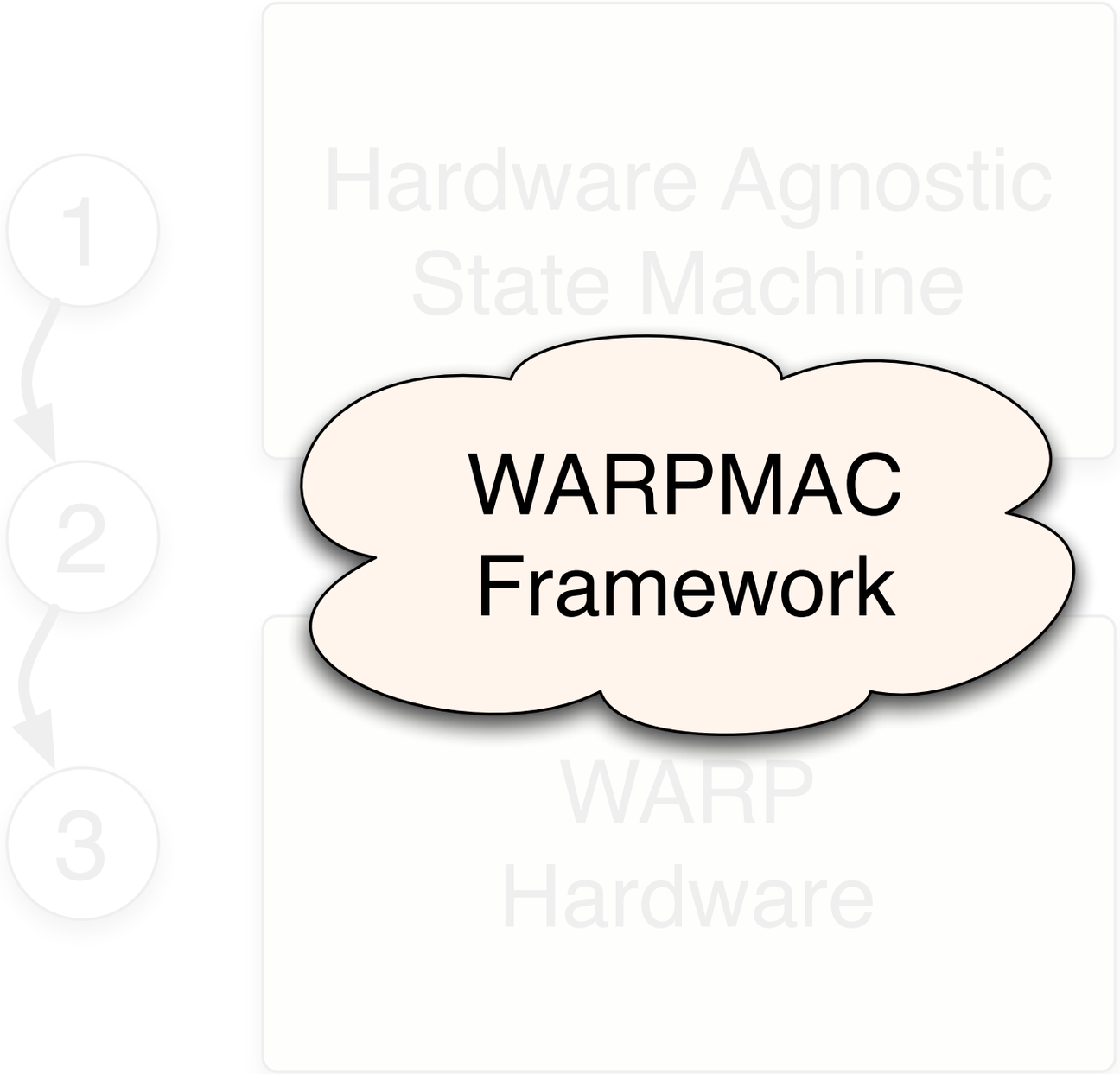
Design Realization

Design Realization



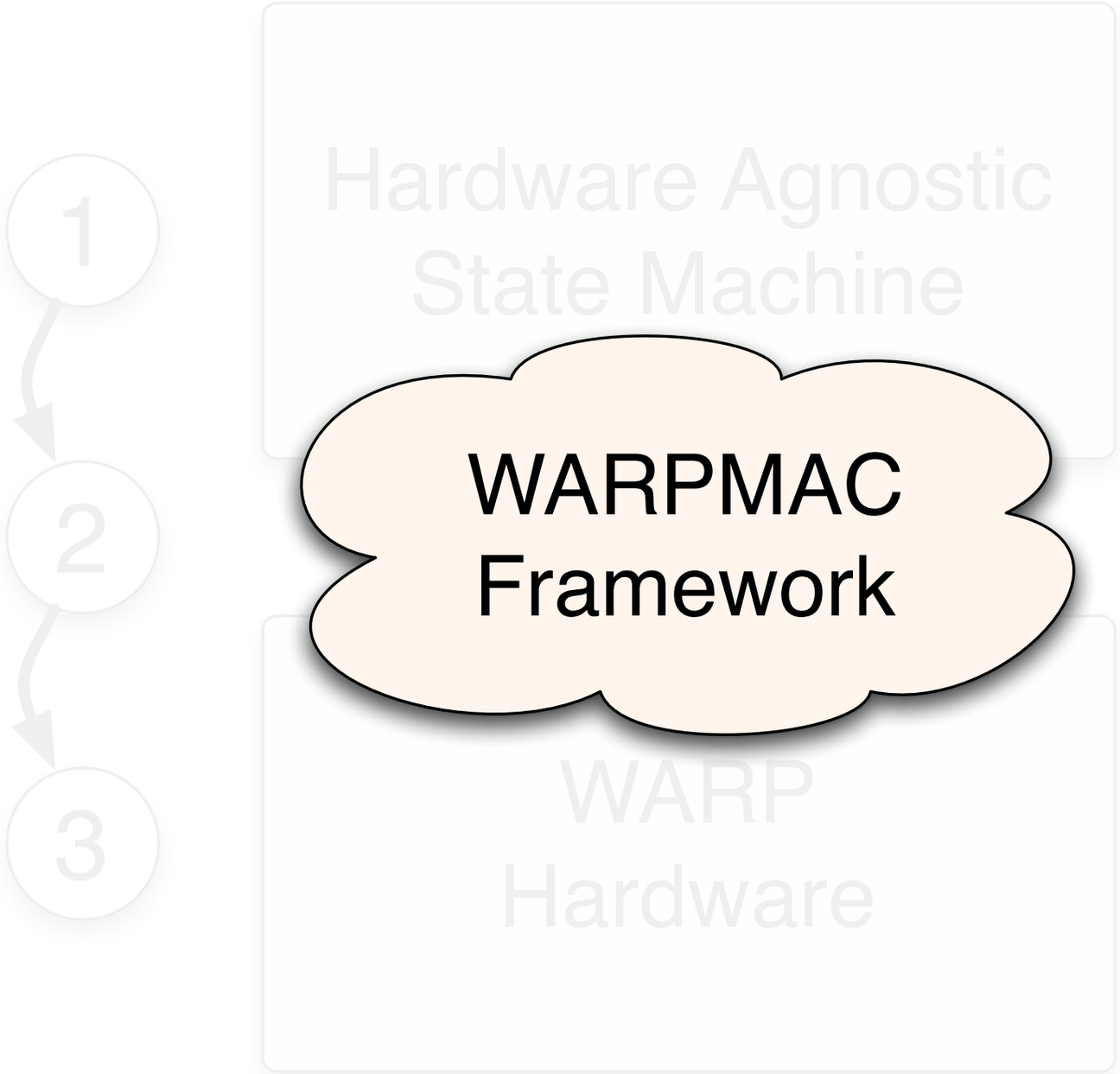
- Program high-level MAC behavior independent of hardware
- Use the WARPMAC framework to stitch the MAC to hardware

Design Realization



- "Driver" analogy is not entirely accurate
- No way to "lock" the framework and have it support all possible future MAC layers

Design Realization



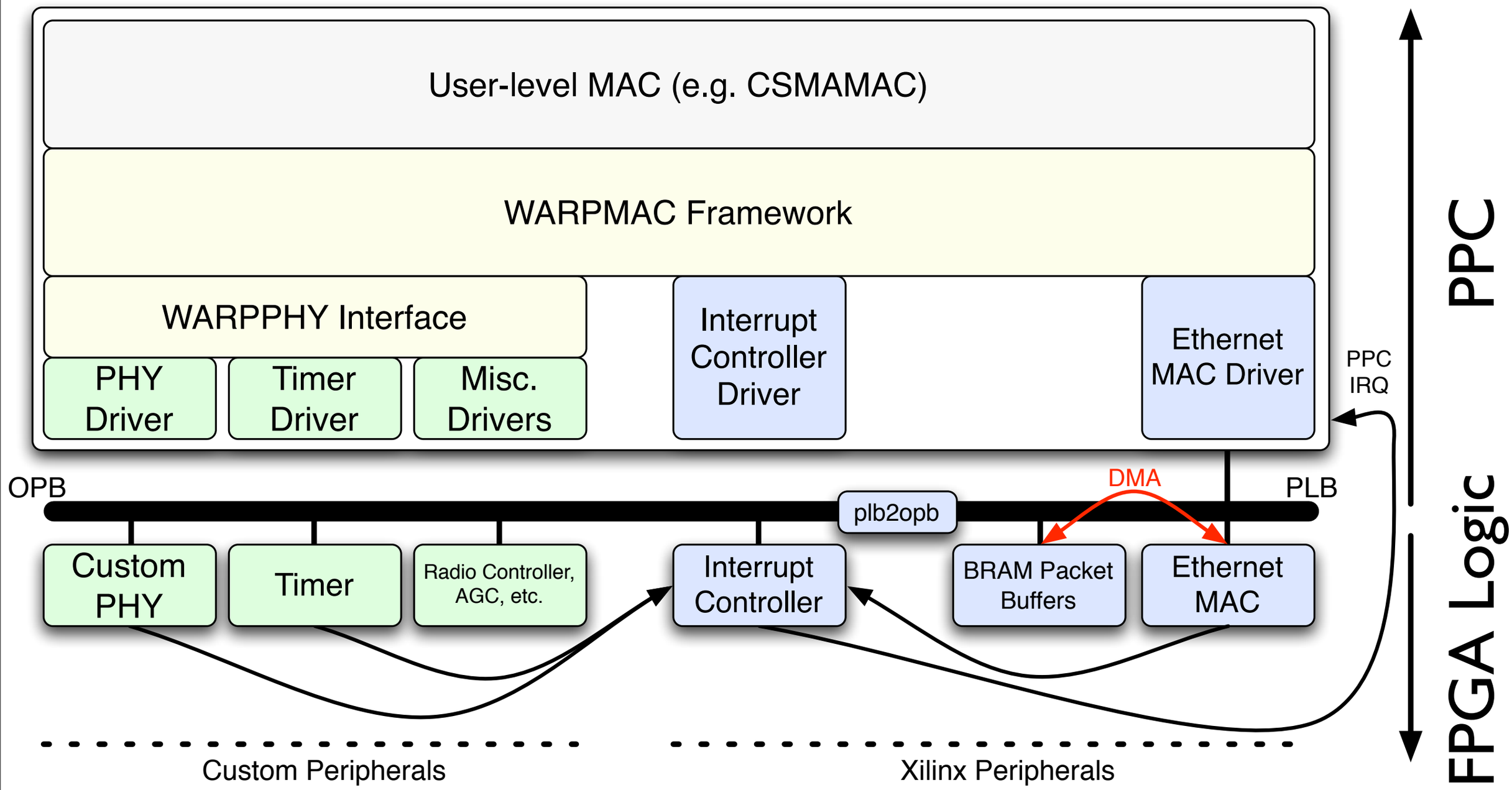
- "Driver" analogy is not entirely accurate
- No way to "lock" the framework and have it support all possible future MAC layers

Solution: *WARPMAC must grow with new algorithms*

WARPMAC

Framework

System Diagram




User Code
WARPMAC
WARPPHY
Drivers


User Code
WARPMAC
WARPPHY
Drivers


User Code
WARPMAC
WARPPHY
Drivers

PHY Driver:

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


User Code
WARPMAC
WARPPHY
Drivers


User Code
WARPMAC
WARPPHY
Drivers

Radio Controller Driver:

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


User Code
WARPMAC
WARPPHY
Drivers


User Code
WARPMAC
WARPPHY
Drivers

PHY Control:

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


User Code
WARPMAC
WARPPHY
Drivers


User Code
WARPMAC
WARPPHY
Drivers

Mostly PHY agnostic

Completely PHY dependent


User Code
WARPMAC
WARPPHY
Drivers

MAC Control:

- Provides control over MAC commonalities
  - Timers for timeouts, backoffs, etc.
  - Carrier-sensing functions
  - Register user callbacks to ISRs
  - Etc.


User Code
WARPMAC
WARPPHY
Drivers


User Code
WARPMAC
WARPPHY
Drivers

User-level MAC Algorithms:

- High-level MAC algorithms
- Some examples so far:
  - Aloha
  - Carrier-sensing MAC
  - Opportunistic Auto-Rate (OAR)
  - MAC Workshop Exercises


User Code
WARPMAC
WARPPHY
Drivers

An example: ALOHA

- Simple, usable MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:

An example: ALOHA

- Simple, usable MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:

Packet to send? Just send it

An example: ALOHA

- Simple, usable MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:

Packet to send? Just send it

Received a packet? Send an ACK

An example: ALOHA

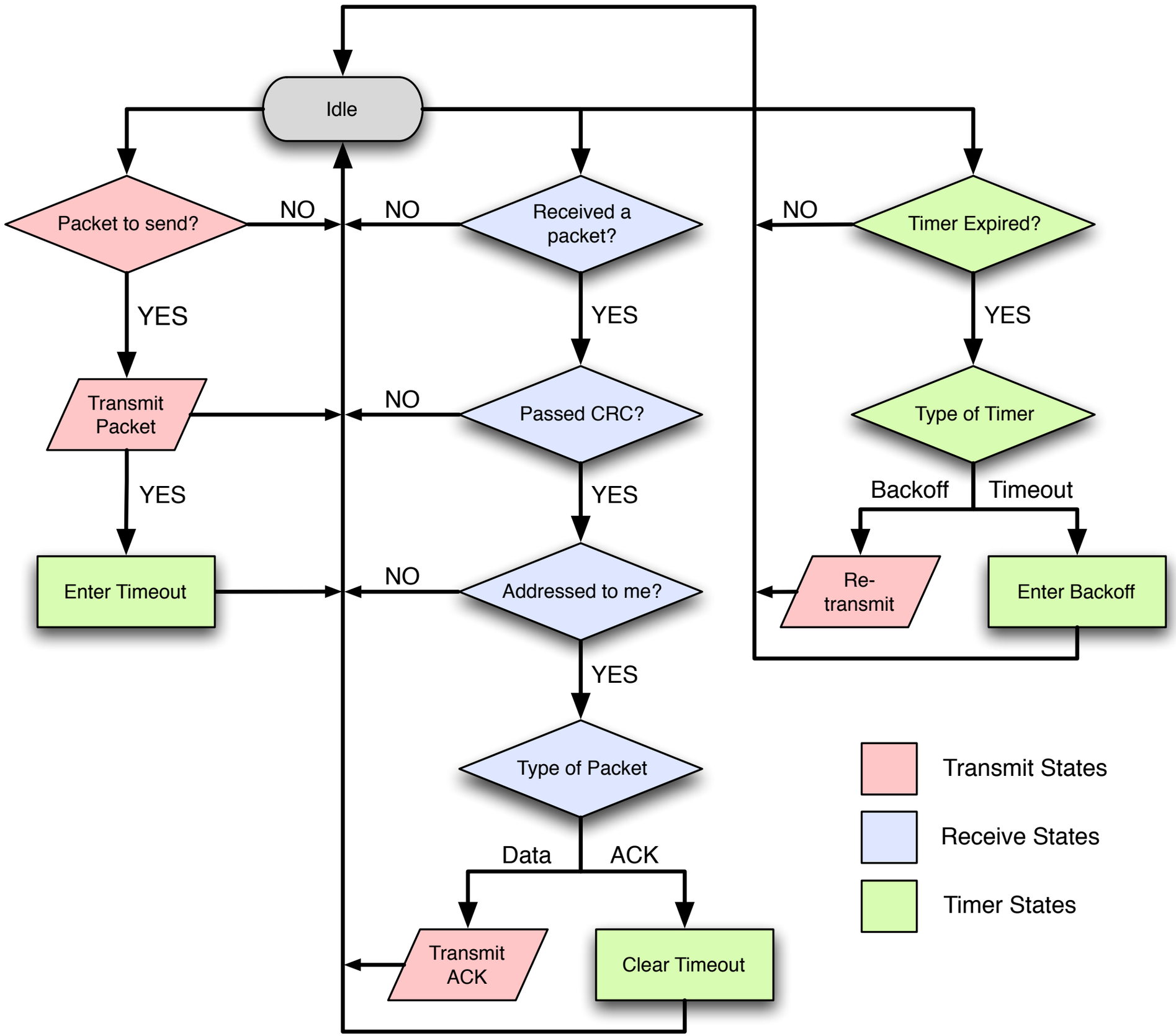
- Simple, usable MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:

Packet to send? Just send it

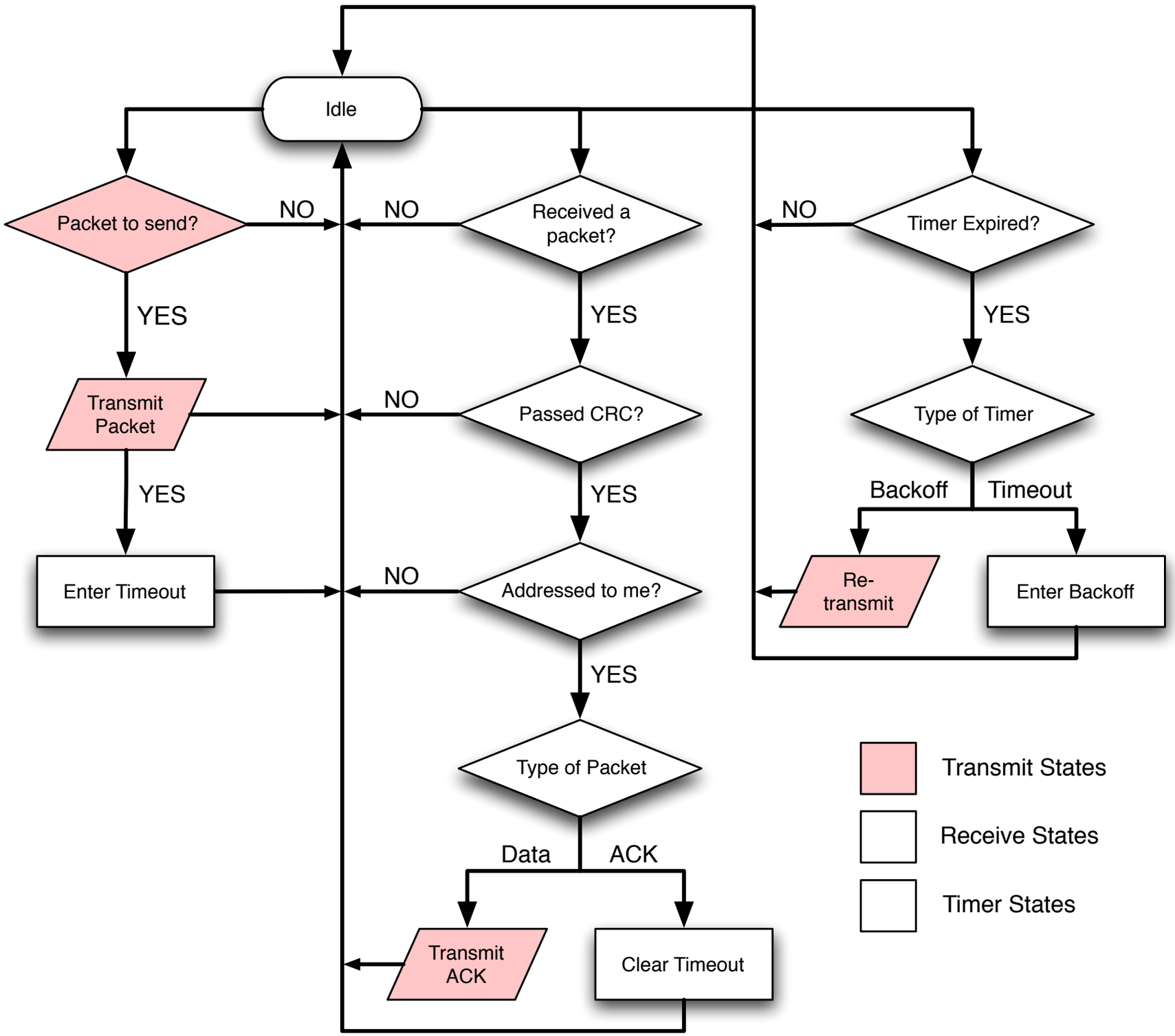
Received a packet? Send an ACK

Received no ACK? Backoff and resend

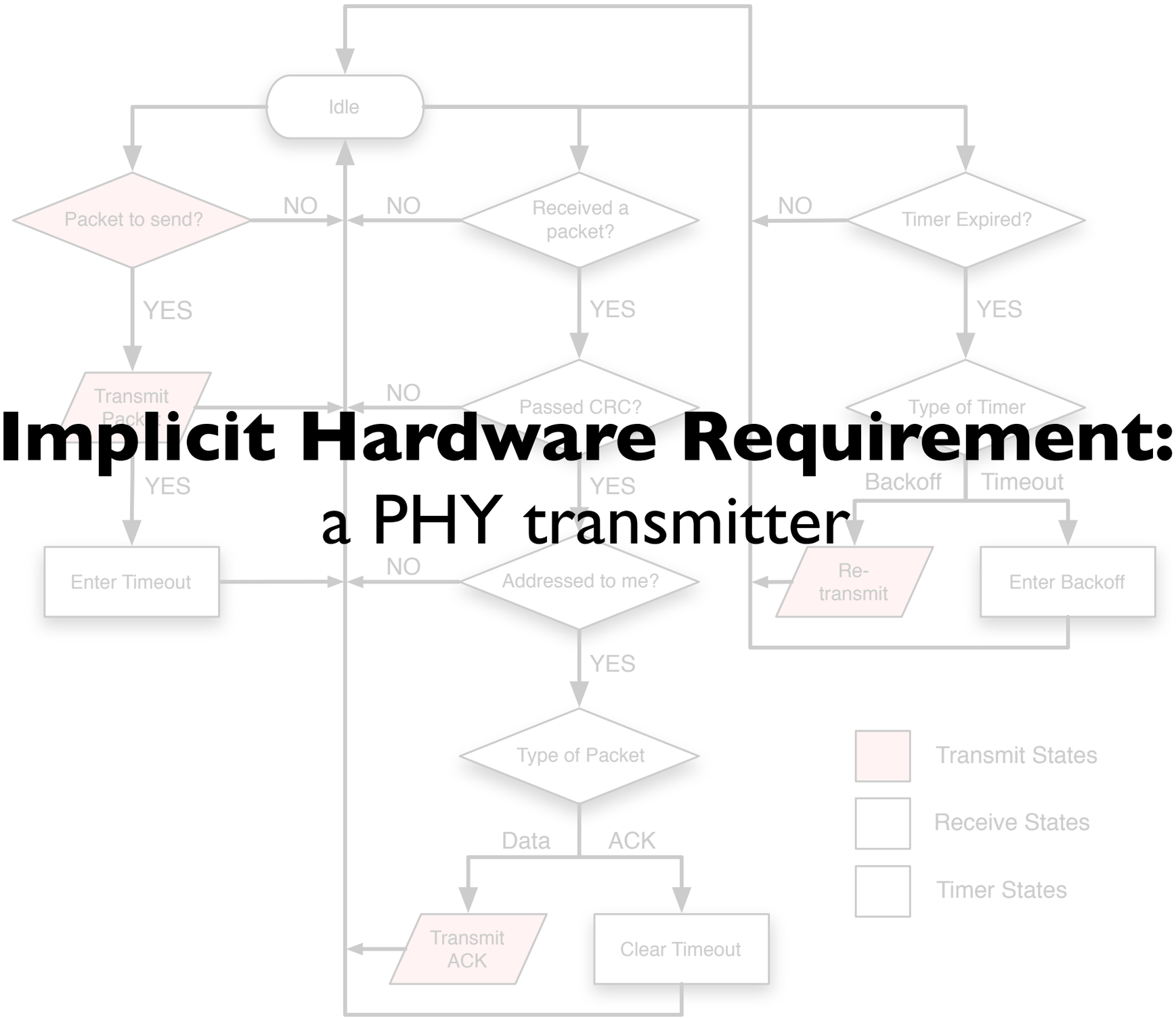
An example: ALOHA



An example: ALOHA

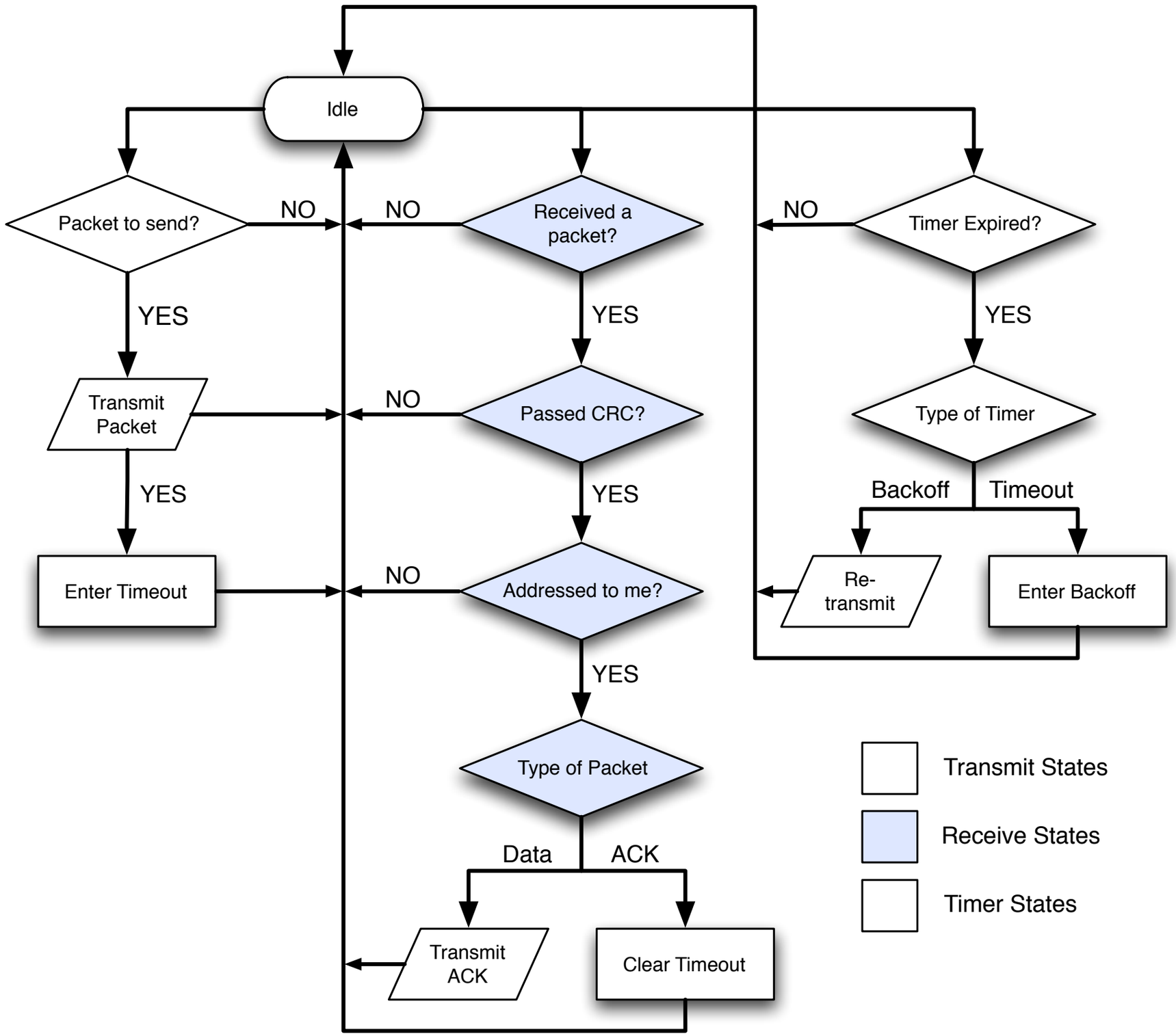


An example: ALOHA

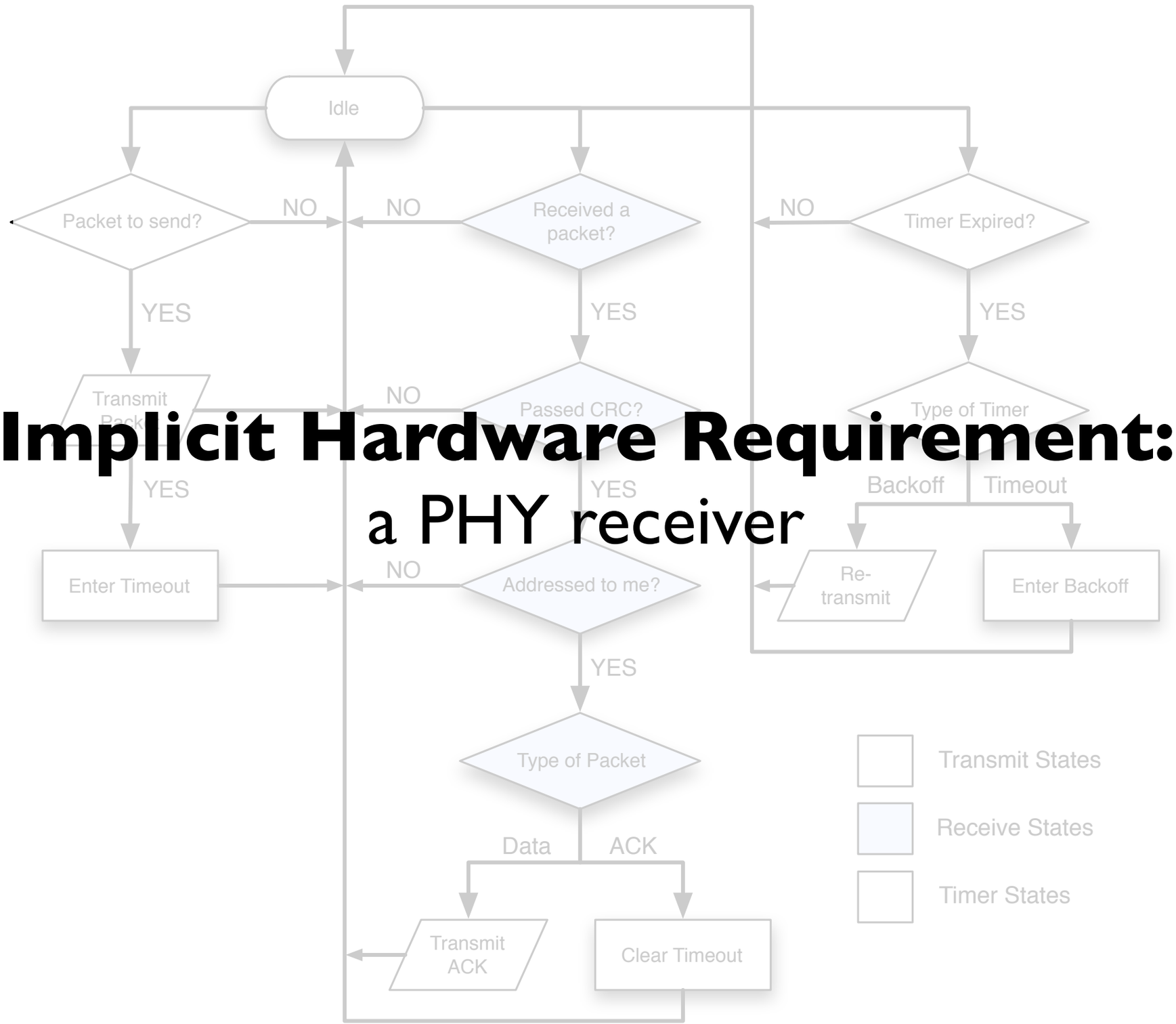


Implicit Hardware Requirement:
a PHY transmitter

An example: ALOHA

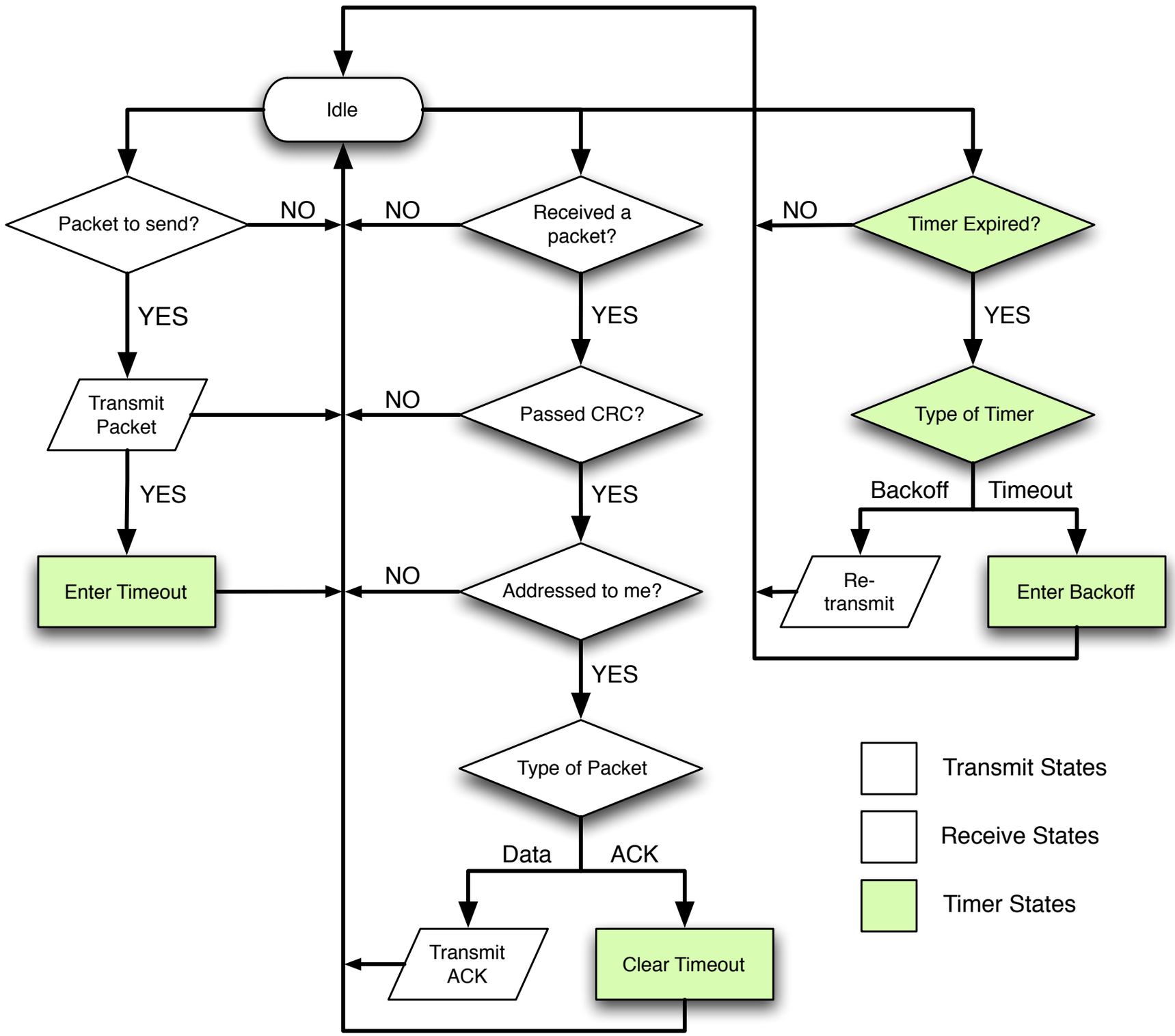


An example: ALOHA

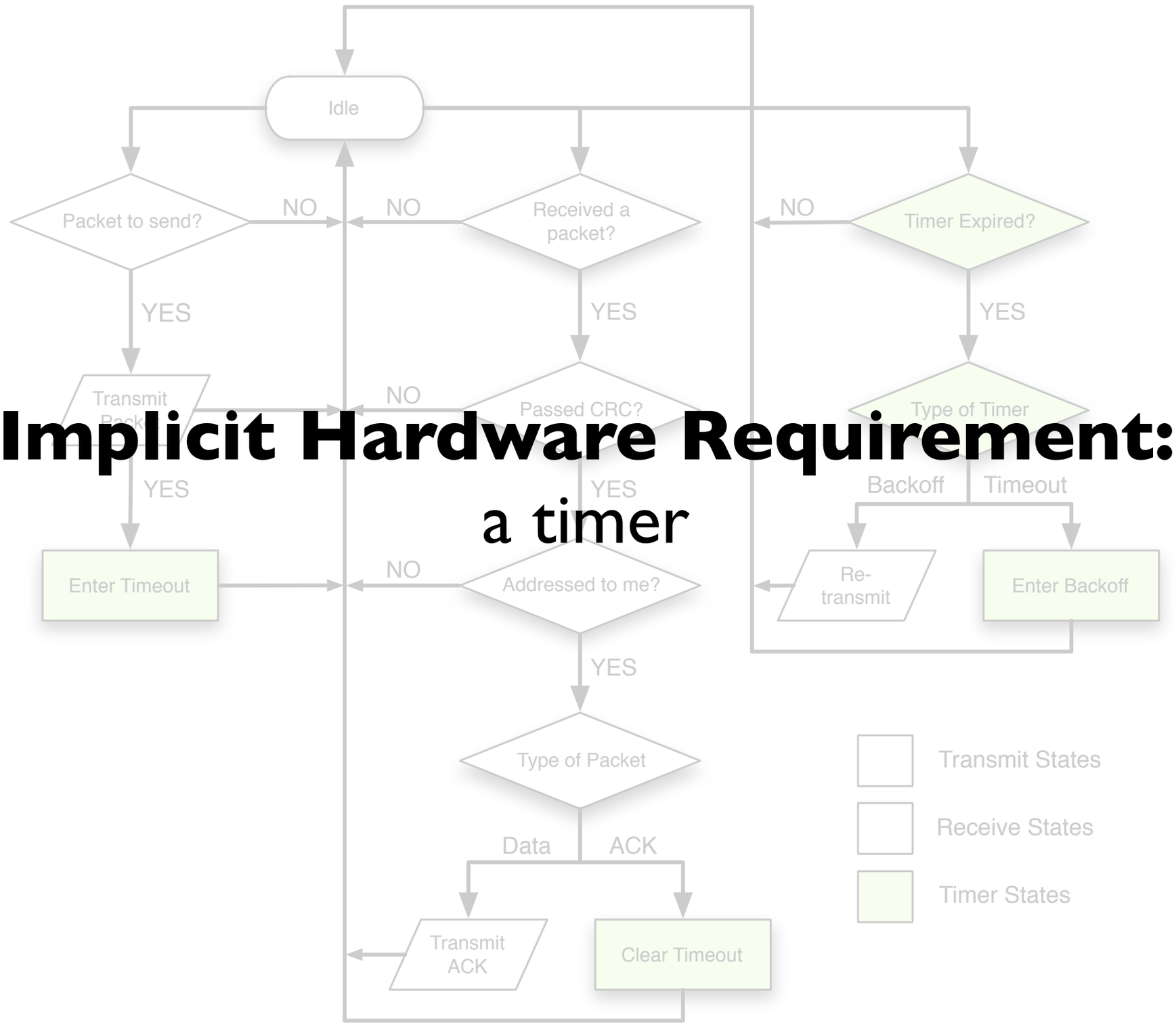


Implicit Hardware Requirement:
a PHY receiver

An example: ALOHA



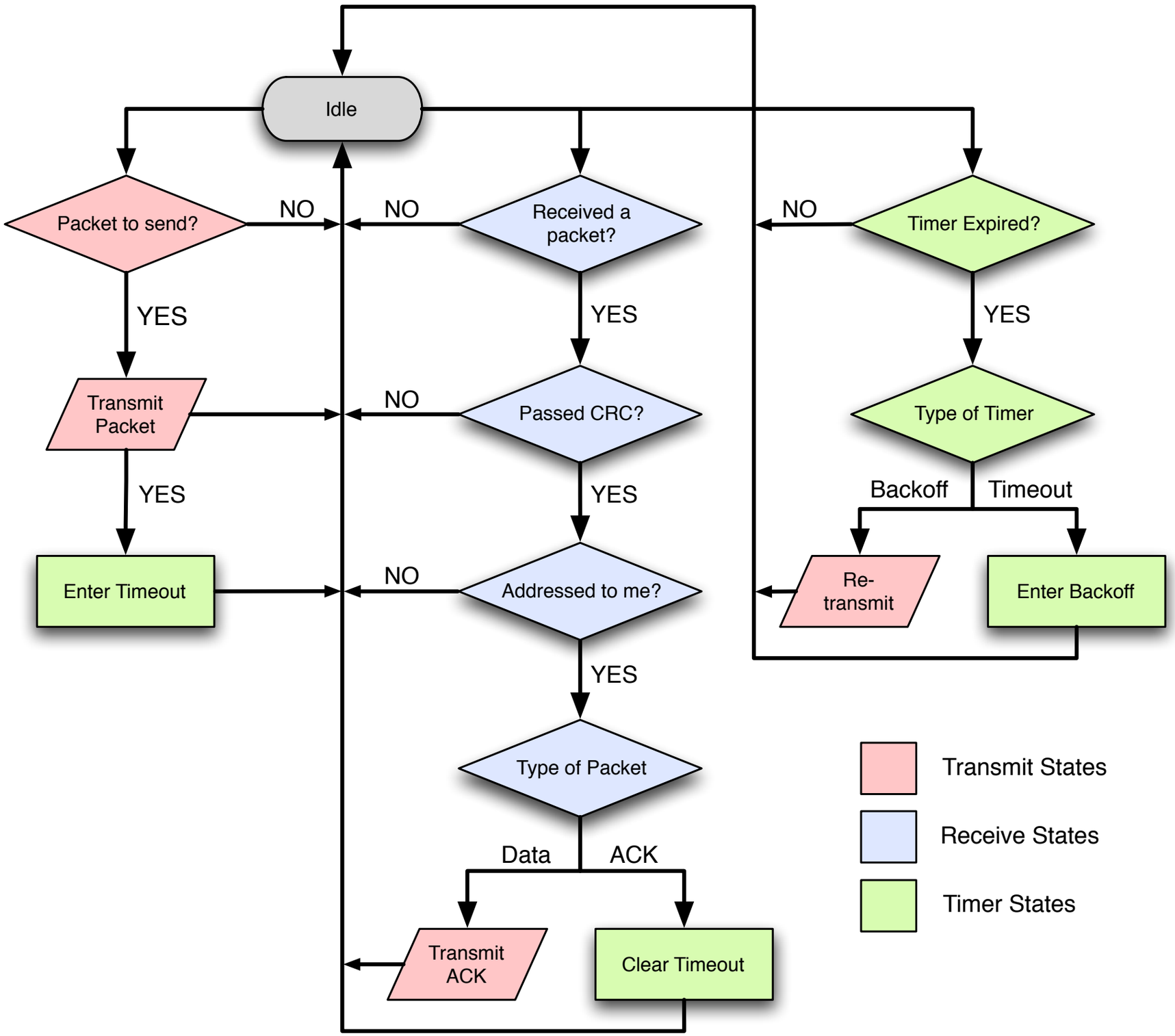
An example: ALOHA



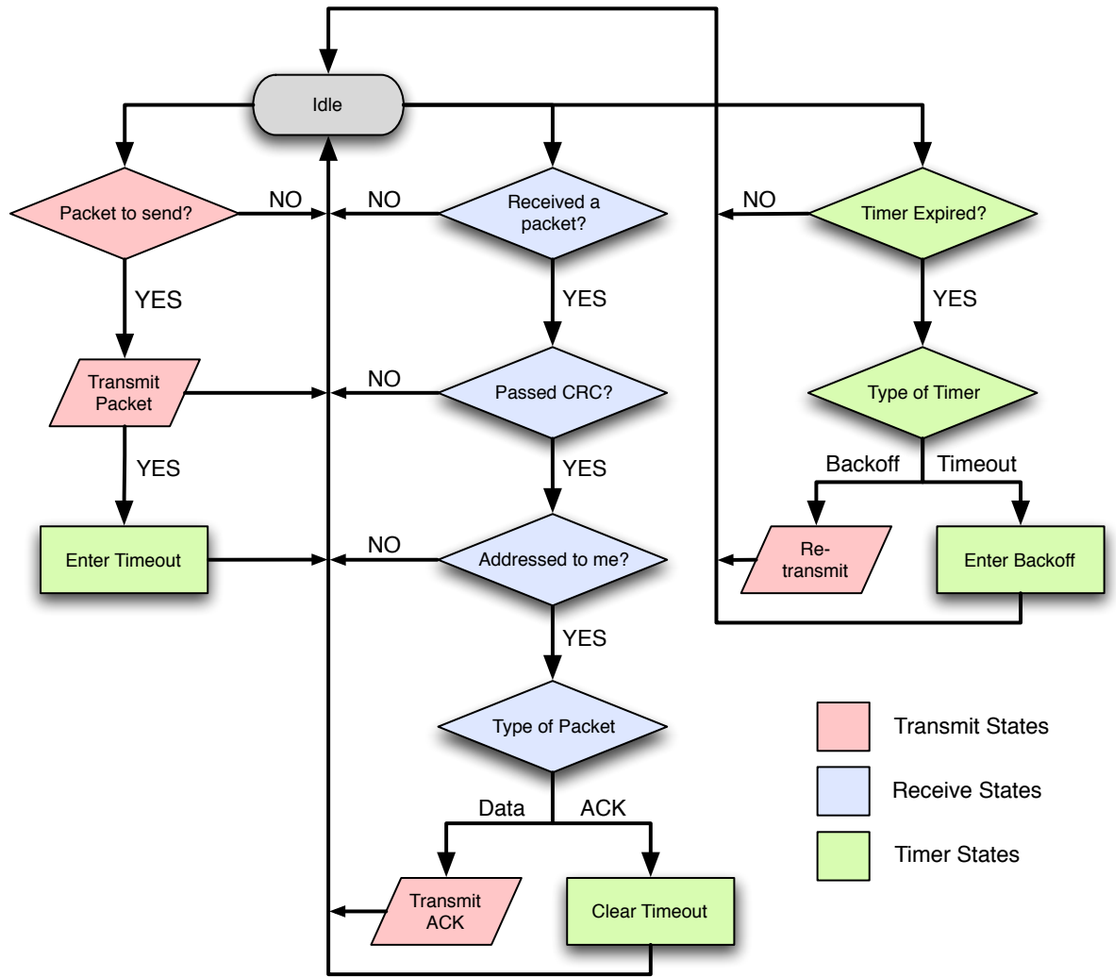
Implicit Hardware Requirement:

a timer

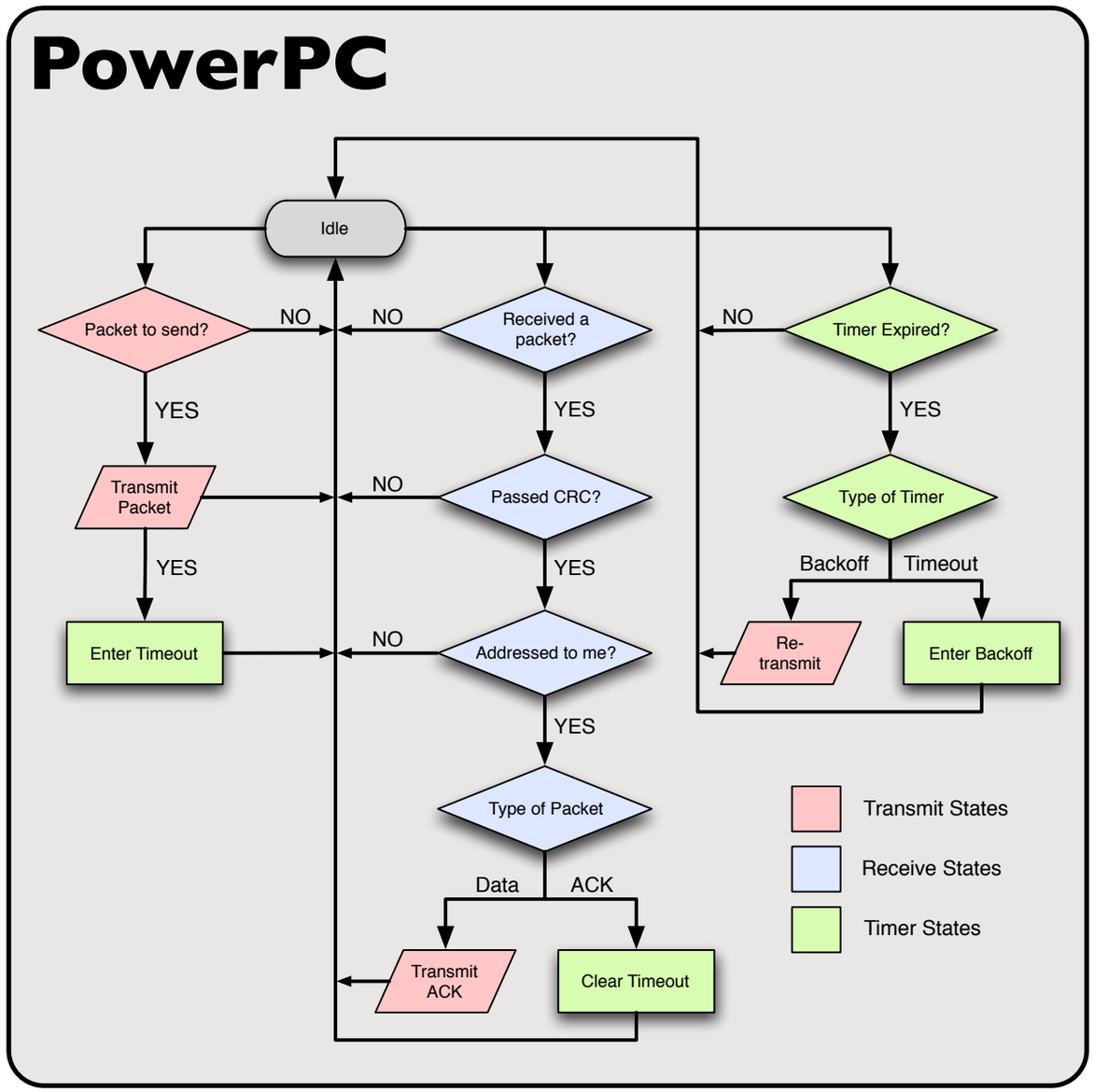
An example: ALOHA



Hardware Requirements



Hardware Requirements

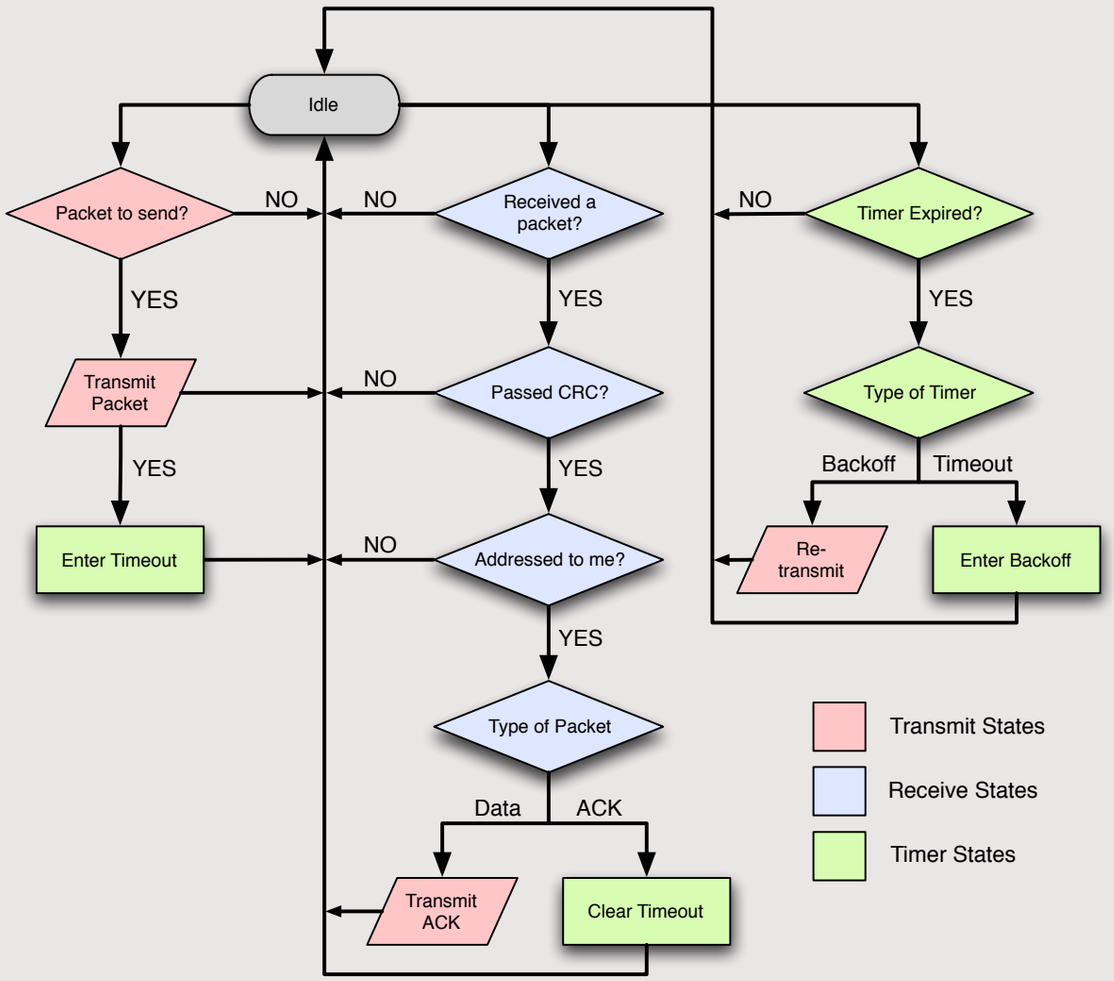


Hardware Requirements

PHY Transmitter

PHY Receiver

PowerPC



Hardware Timer

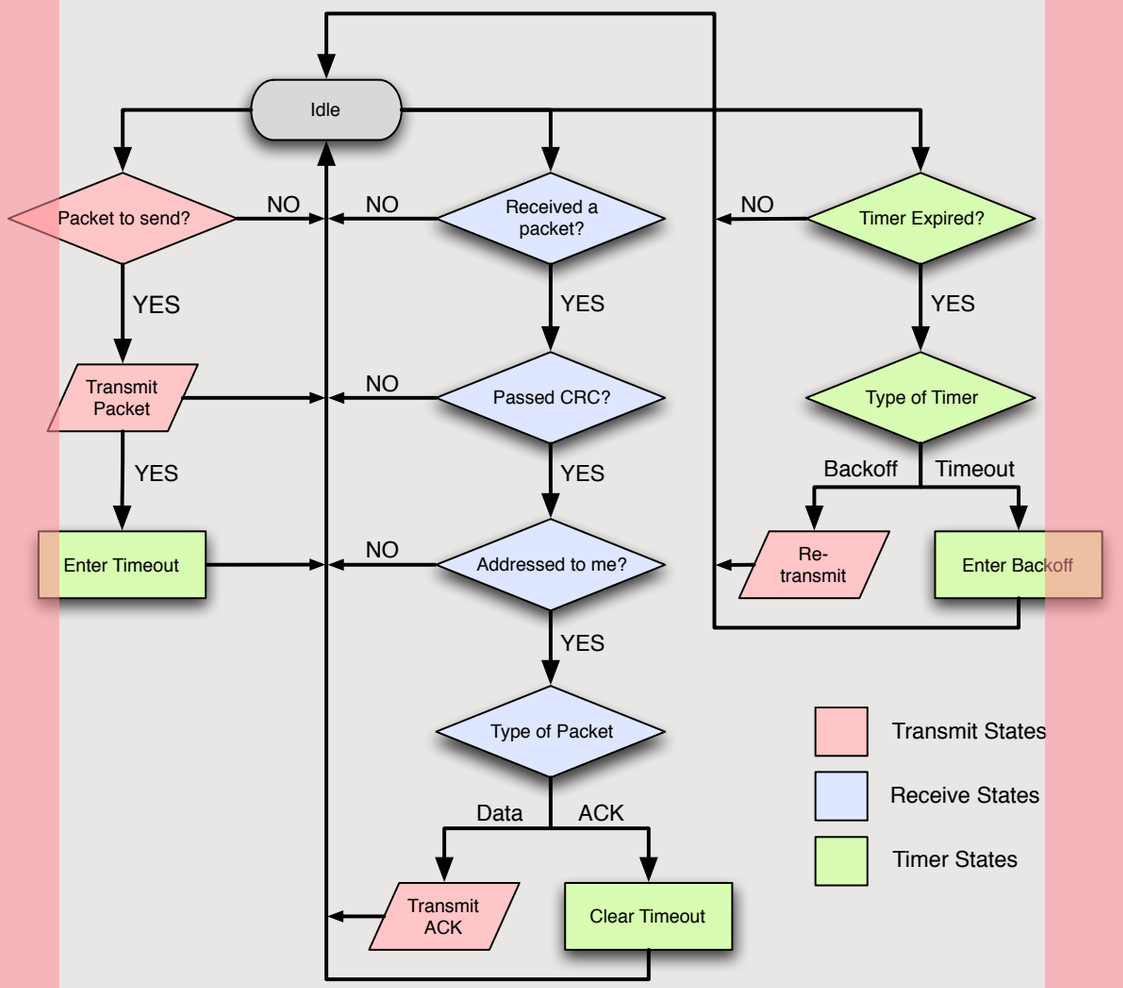
Ethernet

Hardware Requirements

PHY Transmitter

PHY Receiver

PowerPC



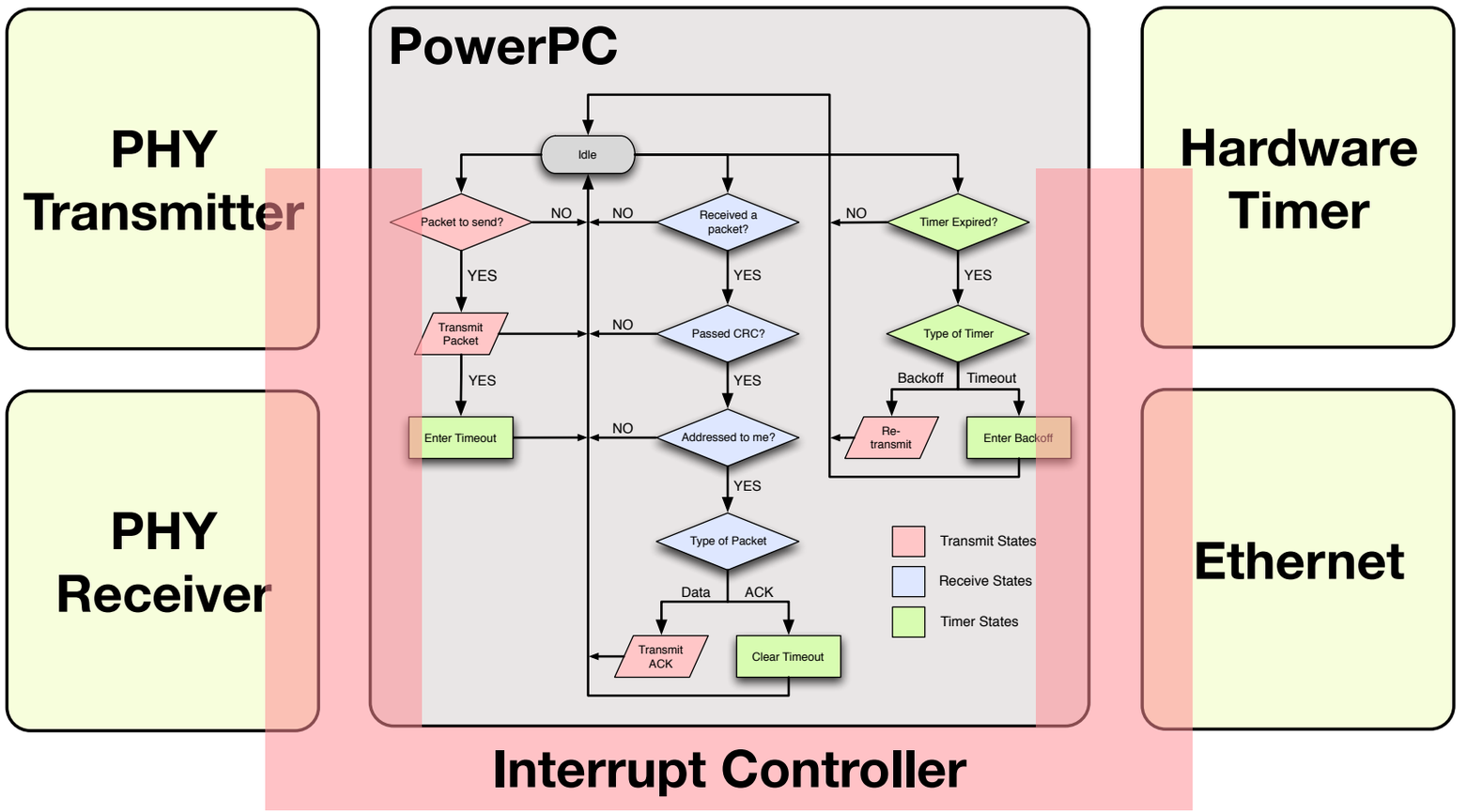
Hardware Timer

Ethernet

Interrupt Controller

Hardware Platform

Hardware Platform



Hardware Platform

Radio Controller

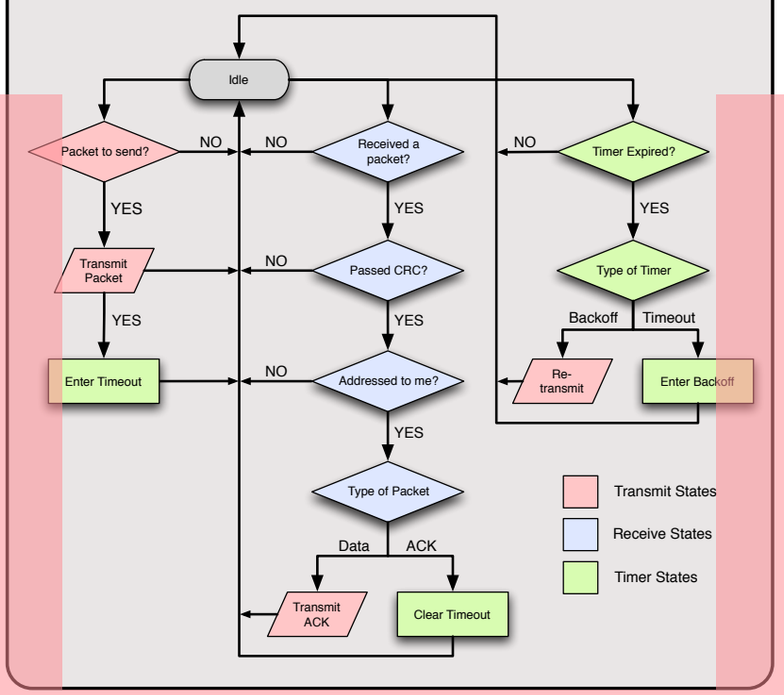
Packet Detection

Automatic Gain Control

PHY Transmitter

PHY Receiver

PowerPC



Hardware Timer

Ethernet

Interrupt Controller

Hardware Platform

Radio Controller

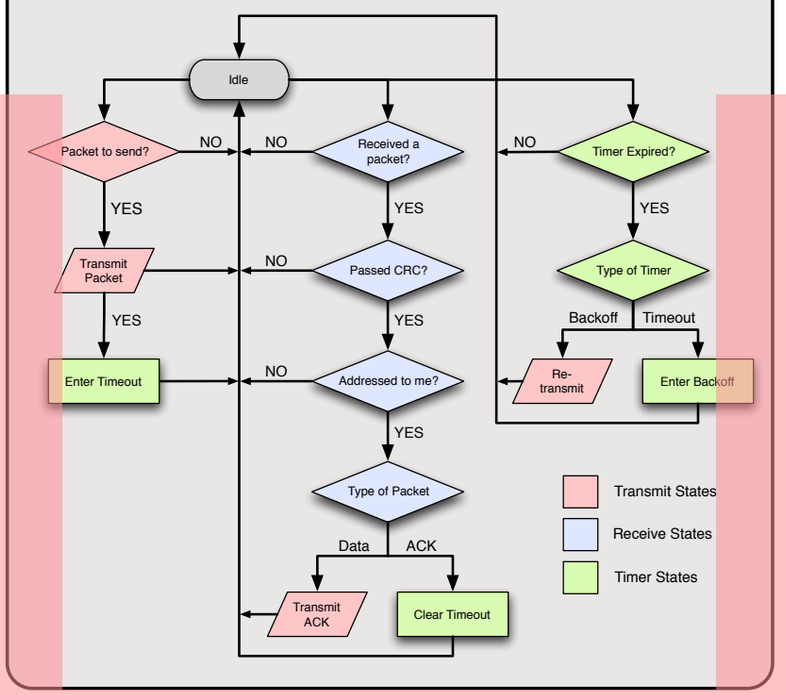
Packet Detection

Automatic Gain Control

PHY Transmitter

PHY Receiver

PowerPC



Interrupt Controller

Hardware Timer

Ethernet

Push-buttons

LEDs

RS232 Serial

Hardware Platform

Radio Controller

Packet Detection

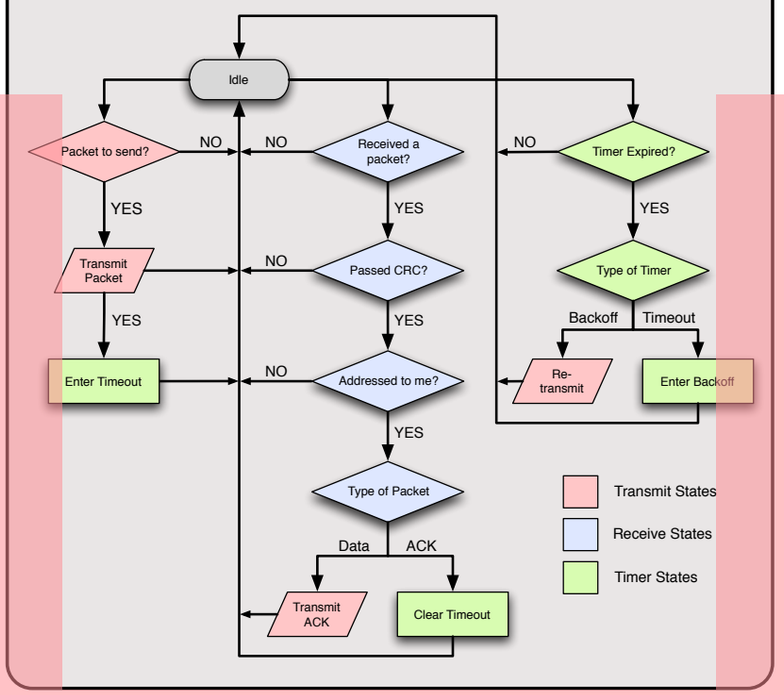
Automatic Gain Control

PowerPC

Control Code

PHY Transmitter

PHY Receiver



Hardware Timer

Ethernet

Interrupt Controller

Push-buttons

LEDs

RS232 Serial

Hardware Platform

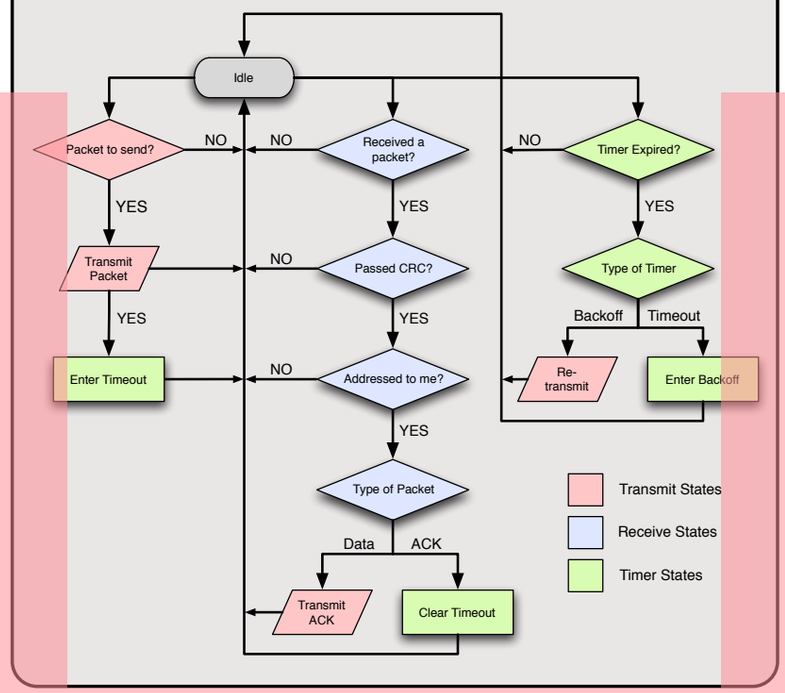
Radio Controller

PowerPC

Control Code

Push-buttons

PHY Transmitter



Hardware Timer

Packet Detection

PHY Receiver

Ethernet

LEDs

Automatic Gain Control

Interrupt Controller

RS232 Serial

Processor Busses (OPB/PLB)

Hardware Platform

WARPMAC

PowerPC

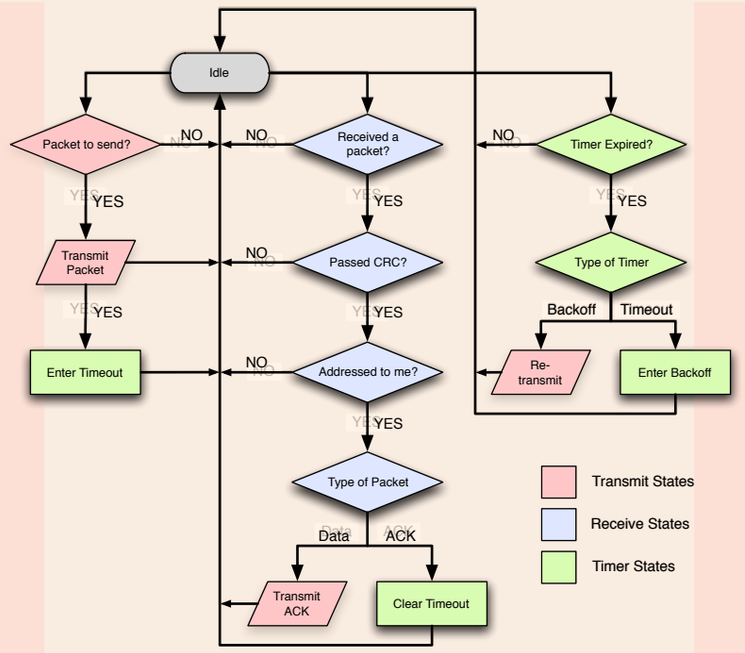
Control Code

Radio Controller

PHY Transmitter

Hardware Timer

Push-buttons



Packet Detection

PHY Receiver

Ethernet

LEDs

Automatic Gain Control

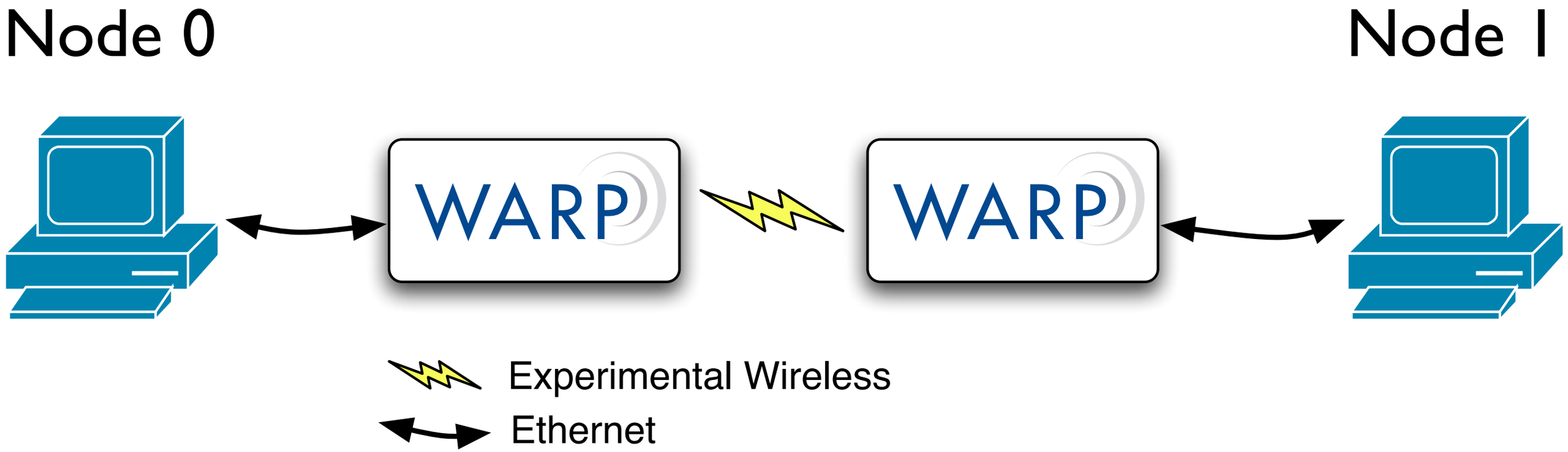
Interrupt Controller

RS232 Serial

Processor Busses (OPB/PLB)

Detailed Example

CSMA


Node 0
Node 1
WARP
WARP
Experimental Wireless
Ethernet

WARPMAC Structs

Macframe:

phyHeader header */* Another struct */*

unsigned char isNew */* Flag for new packets */*

Fully documented in API (http://warp.rice.edu/WARP_API)

WARPMAC Structs

phyHeader:

```
unsigned char fullRate; /* Payload modulation rate */
unsigned char reserved4; /* Unused */
unsigned short int length; /* Payload length */
unsigned char pktType; /* Packet type */
unsigned char destAddr[6]; /* Destination address */
unsigned char srcAddr[6]; /* Source address */
unsigned char currReSend; /* Re-send count */
unsigned char reserved1; /* Unused */
unsigned char reserved2; /* Unused */
unsigned char reserved3; /* Unused */
unsigned short int checksum; /* CRC placeholder */
```

Fully documented in API (http://warp.rice.edu/WARP_API)

```
int main(){
    unsigned char unitSel = 0;

    //Read Dip Switch value from FPGA board
    //This value will be used as an index into the routing table for other nodes
    myID = warpmac_getMyId();

    //Create an arbitrary address for this node
    unsigned char tmpAddr[6] = {0x14,0x24,0x64,0x53,0xe2,0xc2+myID};

    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<10;i++){
        routeTable[i].addr[0] = myAddr[0];
        routeTable[i].addr[1] = myAddr[1];
        routeTable[i].addr[2] = myAddr[2];
        routeTable[i].addr[3] = myAddr[3];
        routeTable[i].addr[4] = myAddr[4];
        routeTable[i].addr[5] = myAddr[5]+i-myID;
    }

    //Initialize the framework
    warpmac_init();

    warpmac_setMacAddr(&myAddr);

    warpmac_setMaxResend(8);
    warpmac_setMaxCW(1);
    warpmac_setTimeout(160);
    warpmac_setSlotTime(9);

    warpmac_setRxBuffer(&rxBuffer,0);
    warpmac_setTxBuffer(1);

    memcpy(txBuffer.header.srcAddr,myAddr,6);

    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);
    warpmac_setTimerHandler(timerExpire);
    warpmac_setEmacHandler(ethernet_callback);

    warpmac_setChannel(CHZ_2,8);
```

```
int main(){
    unsigned char antSel = 0;

    //Read Dip Switch value from FPGA board
    //This value will be used as an index into the routing table for other nodes
    myID = warpmac_getMyId();

    //Create an arbitrary address for this node
    unsigned char tmpAddr[6] = {0x14,0x24,0x63,0x53,0xe2,0xc2+myID};

    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<10;i++){
        routeTable[i].addr[0] = myAddr[0];
        routeTable[i].addr[1] = myAddr[1];
        routeTable[i].addr[2] = myAddr[2];
        routeTable[i].addr[3] = myAddr[3];
        routeTable[i].addr[4] = myAddr[4];
        routeTable[i].addr[5] = myAddr[5]+i-myID;
    }

    //Initialize the framework
    warpmac_init();

    warpmac_setMacAddr(&myAddr);

    warpmac_setMaxResend(8);
    warpmac_setMaxCW(1);
    warpmac_setTimeout(160);
    warpmac_setSlotTime(9);

    warpmac_setRxBuffer(&rxBuffer,8);
    warpmac_setTxBuffer(1);

    memcpy(txBuffer.header.srcAddr,myAddr,6);

    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);
    warpmac_setTimerHandler(timerExpire);
    warpmac_setEmacHandler(ethernet_callback);

    warpmac_setChannel(CHZ_2,8);
```

- Reads the value from the dip switch on the FPGA board for use as identification
- This function also displays the value on the seven-segment displays

```
int main(){
    unsigned char antSel = 0;

    //Read Dip Switch value from FPGA board
    //This value will be used as an index into the routing table for other nodes
    myID = warpmac_getMyId();

    //Create an arbitrary address for this node
    unsigned char tmpAddr[6] = {0x14,0x24,0x63,0x53,0xe2,0xc2+myID};

    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<10;i++){
        routeTable[i].addr[0] = myAddr[0];
        routeTable[i].addr[1] = myAddr[1];
        routeTable[i].addr[2] = myAddr[2];
        routeTable[i].addr[3] = myAddr[3];
        routeTable[i].addr[4] = myAddr[4];
        routeTable[i].addr[5] = myAddr[5]+i-myID;
    }

    //Initialize the framework
    warpmac_init();

    warpmac_setMacAddr(&myAddr);

    warpmac_setMaxResend(8);
    warpmac_setMaxCW(1);
    warpmac_setTimeout(160);
    warpmac_setSlotTime(9);

    warpmac_setRxBuffer(&rxBuffer,0);
    warpmac_setTxBuffer(1);

    memcpy(txBuffer.header.srcAddr,myAddr,6);

    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);
    warpmac_setTimerHandler(timerExpire);
    warpmac_setEmacHandler(ethernet_callback);

    warpmac_setChannel(CHZ_2,0);
```

- Defines an arbitrary address, based on the node ID
- Specifies a crude “routing table” to allow nodes to communicate with one another using only the node IDs

```
int main(){
    unsigned char antSel = 0;

    //Read Dip Switch value from FPGA board
    //This value will be used as an index into the routing table for other nodes
    myID = warpmac_getMyId();

    //Create an arbitrary address for this node
    unsigned char tmpAddr[6] = {0x16,0x24,0x63,0x51,0xe2,0xc2+myID};

    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<10;i++){
        routeTable[i].addr[0] = myAddr[0];
        routeTable[i].addr[1] = myAddr[1];
        routeTable[i].addr[2] = myAddr[2];
        routeTable[i].addr[3] = myAddr[3];
        routeTable[i].addr[4] = myAddr[4];
        routeTable[i].addr[5] = myAddr[5]+i-myID;
    }

    //Initialize the framework
    warpmac_init();

    warpmac_setMacAddr(&myAddr);

    warpmac_setMaxResend(8);
    warpmac_setMaxCW(5);
    warpmac_setTimeout(160);
    warpmac_setSlotTime(9);

    warpmac_setRxBuffer(&rxBuffer,0);
    warpmac_setTxBuffer(1);

    memcpy(txBuffer_header.srcAddr,myAddr,6);

    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);
    warpmac_setTimerHandler(timerExpire);
    warpmac_setEthHandler(ethernet_callback);

    warpmac_setChannel(CHAN_2,0);
```

- Initializes the framework
  - Initializes PHY, radio, AGC, packet detection, interrupts, etc.
- Sets specific parameters
  - 8 resends
  - Maximum contention window of 5 * (Slot-time)
  - 160 usec timeout
  - 9 usec Slot-time

```
int main(){
    unsigned char antSel = 0;

    //Read Dip Switch value from FPGA board
    //This value will be used as an index into the routing table for other nodes
    myID = warpmac_getMyId();

    //Create an arbitrary address for this node
    unsigned char tmpAddr[6] = {0x16,0x24,0x61,0x51,0xe2,0xc2+myID};

    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<10;i++){
        routeTable[i].addr[0] = myAddr[0];
        routeTable[i].addr[1] = myAddr[1];
        routeTable[i].addr[2] = myAddr[2];
        routeTable[i].addr[3] = myAddr[3];
        routeTable[i].addr[4] = myAddr[4];
        routeTable[i].addr[5] = myAddr[5]+i-myID;
    }

    //Initialize the framework
    warpmac_init();

    warpmac_setMacAddr(&myAddr);

    warpmac_setMaxResend(8);
    warpmac_setMaxCW(1);
    warpmac_setTimeout(1600);
    warpmac_setSlotTime(9);

    warpmac_setRxBuffer(&rxBuffer,0);
    warpmac_setTxBuffer(1);

    memcpy(txBuffer_header.srcAddr,myAddr,6);

    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);
    warpmac_setTimerHandler(timerExpire);
    warpmac_setEmacHandler(ethernet_callback);

    warpmac_setChannel(CH2_2.4);
```

```
memcpy(myAddr,tmpAddr,6);

//Fill an arbitrary routing table so that nodes know each others' addresses
unsigned char i;
for(i=0;i<10;i++){
    routeTable[i].addr[0] = myAddr[0];
    routeTable[i].addr[1] = myAddr[1];
    routeTable[i].addr[2] = myAddr[2];
    routeTable[i].addr[3] = myAddr[3];
    routeTable[i].addr[4] = myAddr[4];
    routeTable[i].addr[5] = myAddr[5]+i-myID;
}

//Initialize the framework
warpmac_init();

warpmac_setMacAddr(&myAddr);

warpmac_setMaxResend(4);
warpmac_setMax(30);
warpmac_setTimeout(100);
warpmac_setSlotTime(9);

warpmac_setRxBuffer(&rxBuffer,0);
warpmac_setTxBuffer(1);

memcpy(txBuffer.header.srcAddr,myAddr,6);

warpmac_setGoodPacketHandler(receiveGoodPacket);
warpmac_setBadPacketHandler(receiveBadPacket);
warpmac_setTimerHandler(timerExpire);
warpmac_setEmacHandler(ethernet_callback);

warpmac_setChannel(CHZ_2,0);

warpmac_enableCSMA();
warpmac_enableEthernetInterrupt();

//Set the modulation scheme use for base rate (header) symbols
warpmac_setBaseRate(QPSK);


while(1){
}
```

```
memcpy(myAddr,tmpAddr,6);

//Fill an arbitrary routing table so that nodes know each others' addresses
unsigned char i;
for(i=0;i<10;i++){
    routeTable[i].addr[0] = myAddr[0];
    routeTable[i].addr[1] = myAddr[1];
    routeTable[i].addr[2] = myAddr[2];
    routeTable[i].addr[3] = myAddr[3];
    routeTable[i].addr[4] = myAddr[4];
    routeTable[i].addr[5] = myAddr[5]+i-myID;
}

//Initialize the framework
warpmac_init();

warpmac_setMacAddr(&myAddr);

warpmac_setMaxResend(4);
warpmac_setMax(20);
warpmac_setTimeout(100);
warpmac_setSlotTime(9);
```

```
warpmac_setRxBuffer(&rxBuffer,0);
warpmac_setTxBuffer(1);

memcpy(txBuffer.header.srcAddr,myAddr,6);

warpmac_setGoodPacketHandler(receiveGoodPacket);
warpmac_setBadPacketHandler(receiveBadPacket);
warpmac_setTimerHandler(timerExpire);
warpmac_setEmacHandler(ethernet_callback);

warpmac_setChannel(CHAN_2_4);

warpmac_enableCSMA();
warpmac_enableEthernetInterrupt();

//Set the modulation scheme use for base rate (header) symbols
warpmac_setBaseRate(QPSK);


while(1){
}
```

- Tells WARPMAC to receive wireless packets into a particular buffer
- Tells WARPMAC to send wireless packets from a particular buffer
- Registers user interrupt handlers with the frameworks

```
memcpy(myAddr,tmpAddr,6);

//fill an arbitrary routing table so that nodes know each others' addresses
unsigned char i;
for(i=0;i<10;i++){
    routeTable[i].addr[0] = myAddr[0];
    routeTable[i].addr[1] = myAddr[1];
    routeTable[i].addr[2] = myAddr[2];
    routeTable[i].addr[3] = myAddr[3];
    routeTable[i].addr[4] = myAddr[4];
    routeTable[i].addr[5] = myAddr[5]+i-myID;
}

//Initialize the framework
warpmac_init();

warpmac_setMacAddr(&myAddr);

warpmac_setMaxResend(4);
warpmac_setMax(20);
warpmac_setTimeout(100);
warpmac_setSlotTime(9);

warpmac_setRxBuffer(&rxBuffer,0);
warpmac_setTxBuffer(1);

memcpy(txBuffer.header.srcAddr,myAddr,6);

warpmac_setGoodPacketHandler(receiveGoodPacket);
warpmac_setBadPacketHandler(receiveBadPacket);
warpmac_setTimerHandler(timerExpire);
warpmac_setEmacHandler(ethernet_callback);

warpmac_setChannel(CHZ_2,0);

warpmac_enableCSMA();
warpmac_enableEthernetInterrupt();

//Set the modulation scheme use for base rate (header) symbols
warpmac_setBaseRate(QPSK);


while(1){
}
```

```
    routeTable[i].addr[5] = myAddr[5]+i-myID;
}

//Initialize the framework
warpmac_init();

warpmac_setMacAddr(&myAddr);

warpmac_setMaxResend(8);
warpmac_setMaxCW(1);
warpmac_setTimeout(160);
warpmac_setSlotTime(9);

warpmac_setRxBuffer(&rxBuffer,0);
warpmac_setTxBuffer(1);

memcpy(txBuffer_header.srcAddr,myAddr,6);

warpmac_setGoodPacketHandler(receiveGoodPacket);
warpmac_setBadPacketHandler(receiveBadPacket);
warpmac_setTimerHandler(timerExpire);
warpmac_setEmacHandler(ethernet_callback);

warpmac_setChannel(CHZ_2,8);

warpmac_enableCSMA();
warpmac_enableEthernetInterrupt();

//Set the modulation scheme use for base rate (header) symbols
warpmac_setBaseRate(QPSK);


while(1){
}

return;
}
```

```
warpmac_setChannel(GHZ_2,8);

warpmac_enableCSMA();
warpmac_enableEthernetInterrupt();

//Set the modulation scheme use for base rate (header) symbols
warpmac_setBaseRate(QPSK);

while(1){
}
```

- Sets the frequency band to 802.11 channel 8 of the 2.4GHz band
- Enable carrier-sensing mode of WARPMAC
- Enable the Ethernet interrupt
- Set the base modulation rate to QPSK (must be agreed upon by all nodes in the network)
- Spins forever in a while loop, waiting for an interrupt

Case 1:

Packet received from Ethernet

```
int ethernet_callback(Xuint32 length, char* payload){
    warpmac_disableEthernetInterrupt();
    txBuffer.header.currReSend = 0;
    txBuffer.isNew = 1;
    txBuffer.header.length = length;
    txBuffer.header.pktType = DATAPACKET;

    //Set the modulation scheme for the packet's full-rate symbols
    txBuffer.header.fullRate = QPSK;

    //Copy in the packet's destination MAC address
    //Hard-coded as this node's partner node
    memcpy(txBuffer.header.destAddr,routeTable[(myID+1)%2].addr,6);


    if(warpmac_carrierSense()){
        warpmac_sendOfdm(&txBuffer);
        warpmac_setTimer(TIMEOUT);
    }
    else{
        warpmac_setTimer(BACKOFF);
    }
    return 0;
}
```

- Disables the Ethernet interrupt line until this frame is dealt with
- Metadata and header information is filled in
  - isNew = 1, since it is a new packet
  - Length, packet type, full rate modulation order and the destination MAC address are filled into the header
- If the medium is free, the packet is sent and a timeout begins

Case 2:

"Bad" packet received from OFDM

```
int receiveBadPacket(Macframe* packet) {
    warpmac_incrementLEDLow();
}
```

- If we receive a packet that fails checksum
  - Blink the bottom LEDs
- This way we can have a visualization of channel quality

Case 3:

"Good" data packet received from OFDM

```
int receiveGoodPacket(Macframe* packet) {

    warpmac_incrementLEDHigh();

    if(warpmac_addressedToMe(packet)){
        Macframe ackPacket;
        switch(packet->header.pktType){
            case ACKPACKET:

                if(warpmac_inTimeout()){
                    warpmac_clearTimer(TIMEOUT);
                    txBuffer.header.currReSend = 0;
                    txBuffer.isNew = 0;
                    warpmac_enableEthernetInterrupt();
                }

                break;
            case DATAPACKET:
                warpmac_leftHex(packet->header.currReSend);
                ackPacket.header.length = 0;
                ackPacket.header.pktType = ACKPACKET;
                ackPacket.header.fullRate = QPSK;
                memcpy(ackPacket.header.srcAddr,myAddr,6);
                memcpy(ackPacket.header.destAddr,packet->header.srcAddr,6);
                warpmac_setTxBuffer(2);
                warpmac_sendOfdm(&ackPacket);
                warpmac_setTxBuffer(1);

                packet->header.currReSend = 0;
                packet->isNew = 1;
                warpmac_phyInterruptClear();
                warpmac_sendEthernet(packet);
                packet->header.currReSend = 0;
                packet->isNew = 0;

                break;
        }
    }
return 0;
}
```

- Blink the top LEDs
- If destination address is equal to my source address and the type is a data packet
  - Create an acknowledgment and send it
  - Send the packet over Ethernet

Case 4:

"Good" acknowledgment packet received from OFDM

```
int receiveGoodPacket(Macframe* packet) {

    warpmac_incrementLEDHigh();

    if(warpmac_addressedToMe(packet)){
        Macframe ackPacket;
        switch(packet->header.pktType){
            case ACKPACKET:

                if(warpmac_inTimeout()){
                    warpmac_clearTimer(TIMEOUT);
                    txBuffer.header.currReSend = 0;
                    txBuffer.isNew = 0;
                    warpmac_enableEthernetInterrupt();
                }

                break;
            case DATAPACKET:
                warpmac_leftHex(packet->header.currReSend);
                ackPacket.header.length = 0;
                ackPacket.header.pktType = ACKPACKET;
                ackPacket.header.fullRate = QPSK;
                memcpy(ackPacket.header.srcAddr,myAddr,6);
                memcpy(ackPacket.header.destAddr,packet->header.srcAddr,6);
                warpmac_setTxBuffer(2);
                warpmac_sendOfdm(&ackPacket);
                warpmac_setTxBuffer(1);

                packet->header.currReSend = 0;
                packet->isNew = 1;
                warpmac_phyInterruptClear();
                warpmac_sendEthernet(packet);
                packet->header.currReSend = 0;
                packet->isNew = 0;

                break;
        }
    }
return 0;
}
```

- Blink the top LEDs
- If destination address is equal to my source address and the type is an acknowledgment
  - If a timeout is currently running (i.e., the node is waiting on an ACK)
    - Stop the timer
    - Turn Ethernet interrupts back on (they were disabled in the ethernet handler)

Case 5:

Timeout timer expires

```
int timerExpire(unsigned char timerType){
    int status;
    switch(timerType){
        case TIMEOUT:
            if(txBuffer.isNew){
                status = warpmac_incrementResend(&txBuffer);
                if(status == 0){
                    warpmac_enableEthernetInterrupt();
                    return 0;
                }
                warpmac_setTimer(BACKOFF);
                return 0;
            }
            break;
        case BACKOFF:
            if(warpmac_carrierSense()){
                warpmac_sendOfdm(&txBuffer);
                warpmac_setTimer(TIMEOUT);
            }
            else{
                warpmac_setTimer(BACKOFF);
            }
            break;
            return 0;
    }
}
```

- Increment the resend field of the packet
- Enter a backoff
- Re-enable Ethernet interrupts if maximum retransmissions were met

Case 6:

Backoff timer expires

```
int timerExpire(unsigned char timerType){
    int status;
    switch(timerType){
        case TIMEOUT:
            if(txBuffer.isNew){
                status = warpmac_incrementResend(&txBuffer);
                if(status == 0){
                    warpmac_enableEthernetInterrupt();
                    return 0;
                }
                warpmac_setTimer(BACKOFF);
                return 0;
            }
            break;
        case BACKOFF:
            if(warpmac_carrierSense()){
                warpmac_sendOfdm(&txBuffer);
                warpmac_setTimer(TIMEOUT);
            }
            else{
                warpmac_setTimer(BACKOFF);
            }
            break;
            return 0;
    }
}
```

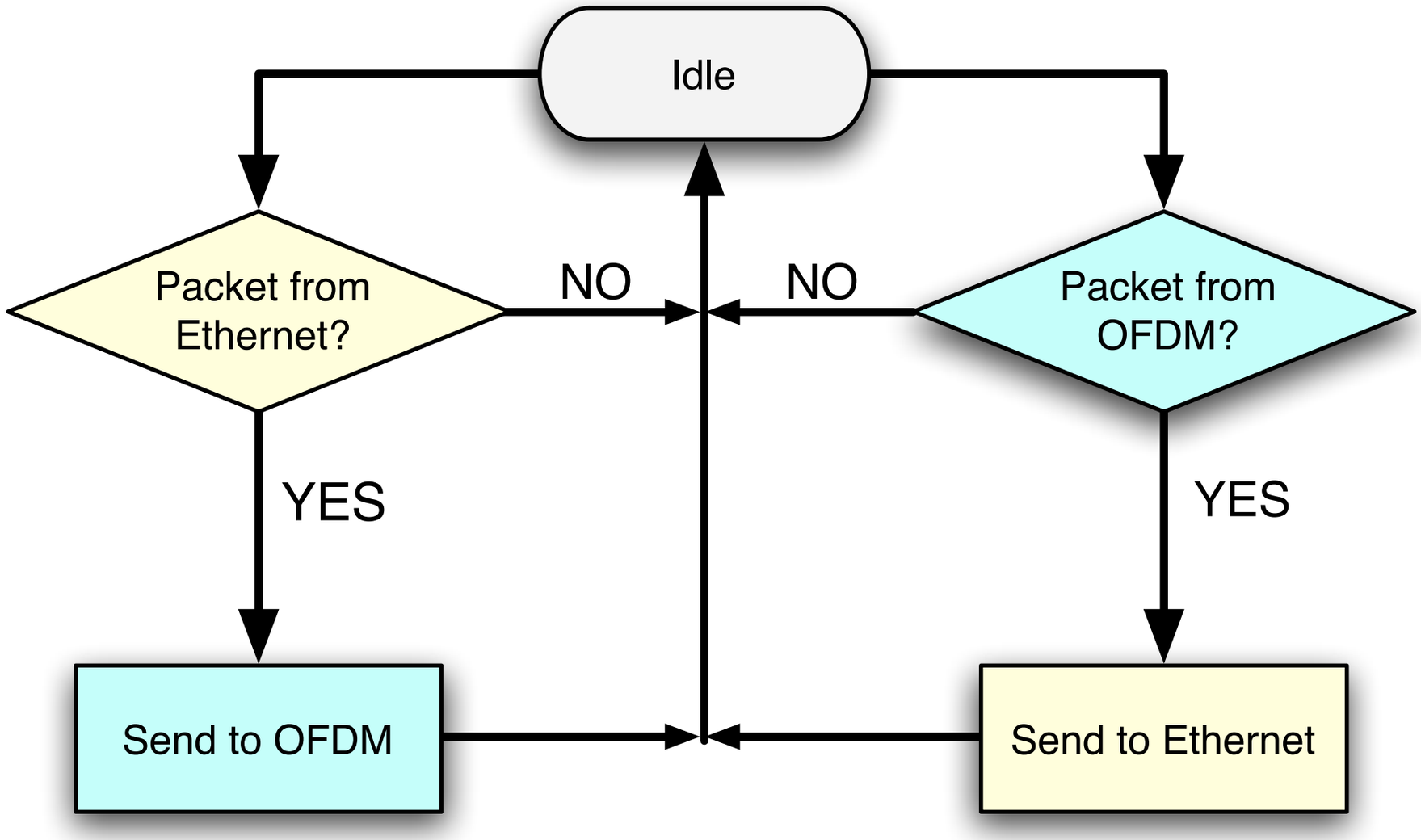
- If the medium is free
  - Send it over OFDM
  - Enter a timeout
- Otherwise, start another timeout

Questions?

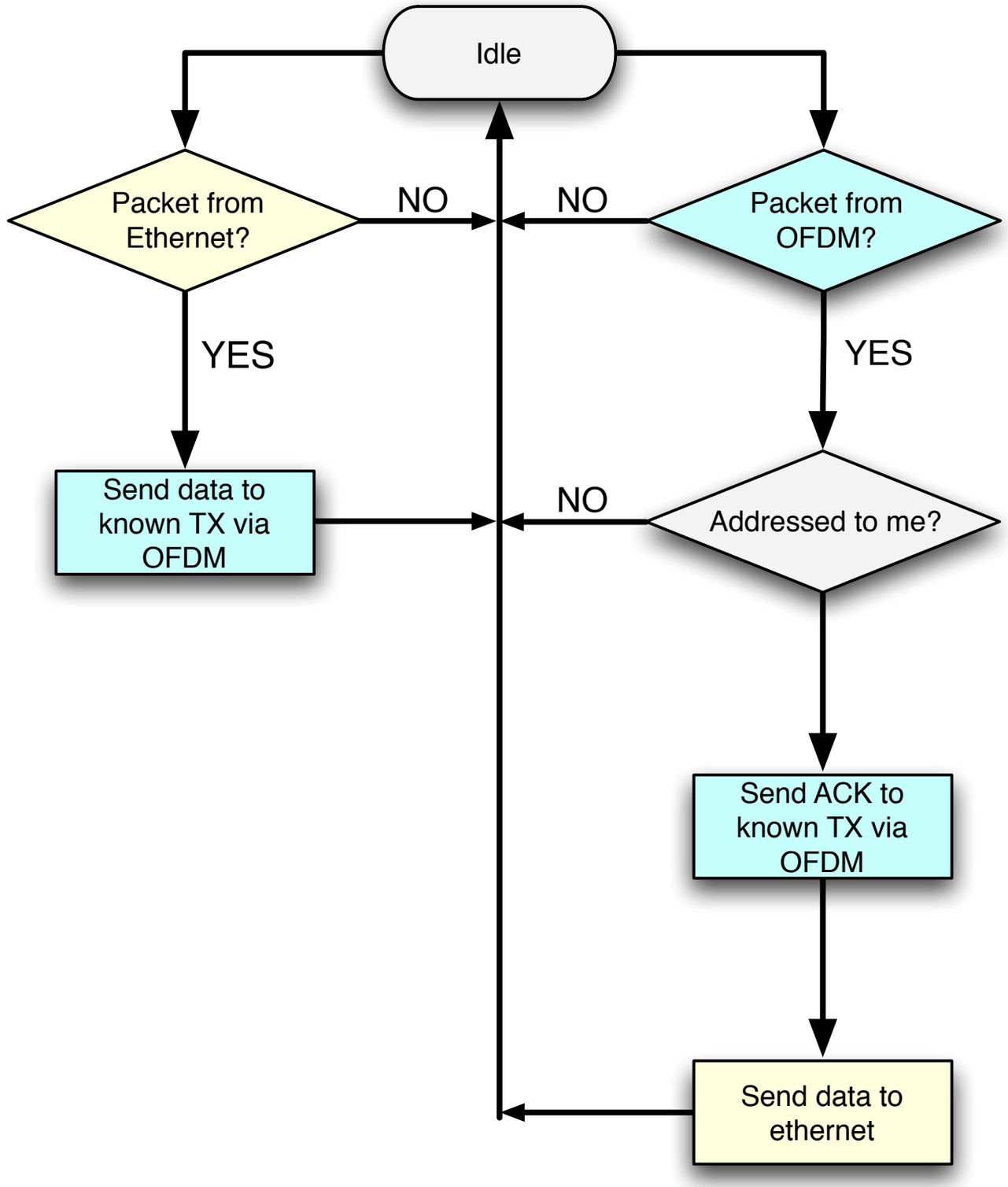
Lab Exercises

- Lab 4 - noMAC: Simple “MAC” layer
- Lab 5 - uniMAC: Unidirectional MAC
- Lab 6 - hopMAC: Channel-hopping MAC

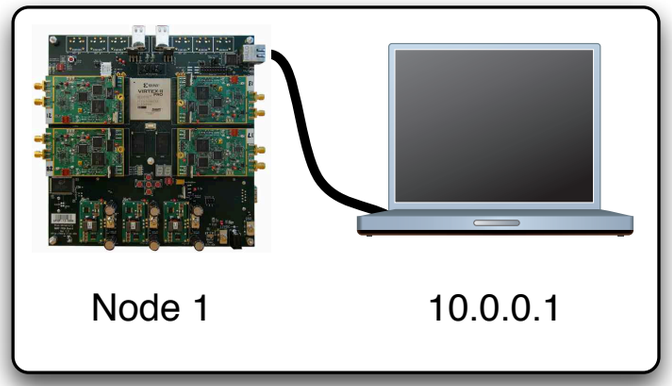
noMac

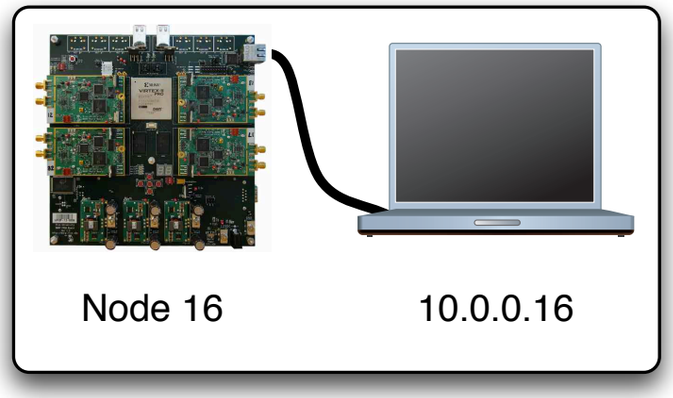

Idle
Packet from Ethernet?
NO
NO
Packet from OFDM?
YES
YES
Send to OFDM
Send to Ethernet

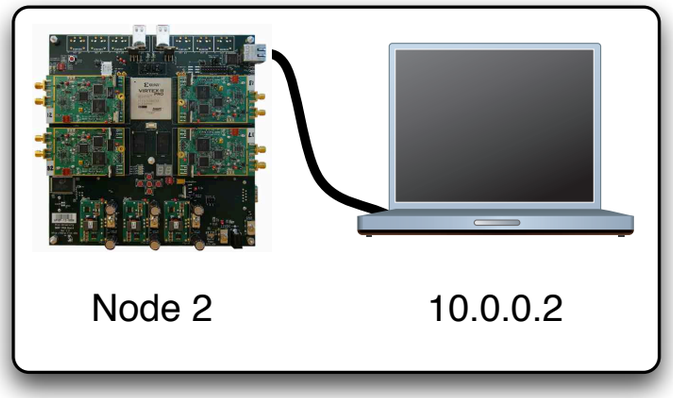
uniMac

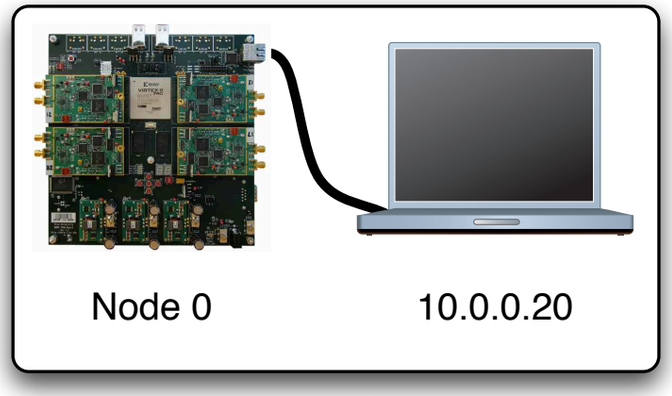

Idle
Packet from Ethernet?
NO
NO
Packet from OFDM?
YES
YES
Send data to known TX via OFDM
NO
Addressed to me?
Send ACK to known TX via OFDM
Send data to ethernet

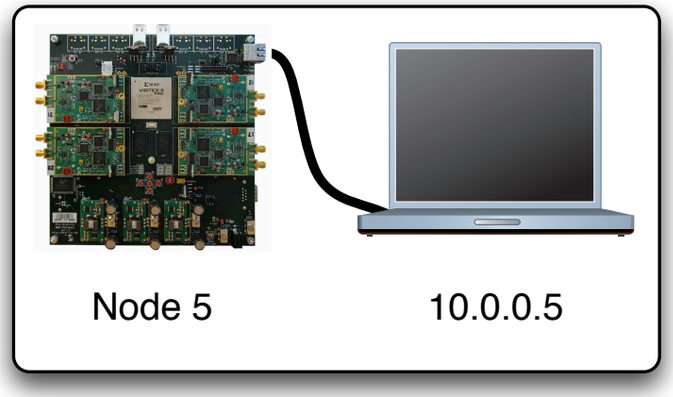
uniMac

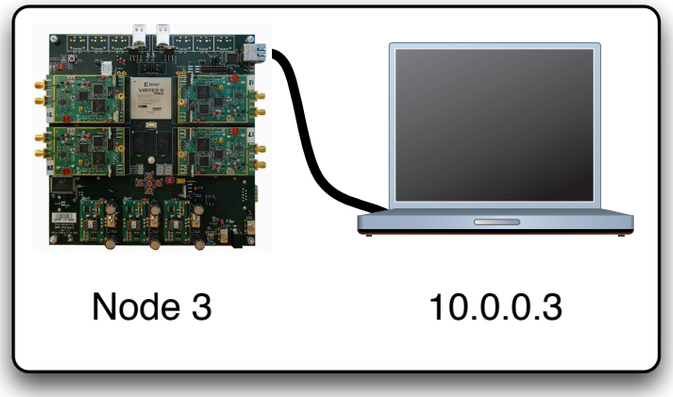


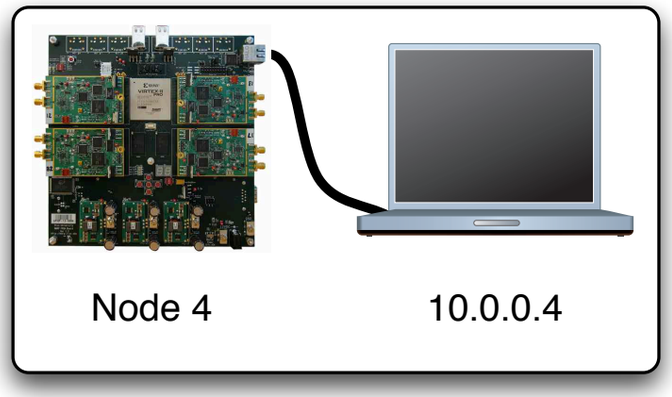






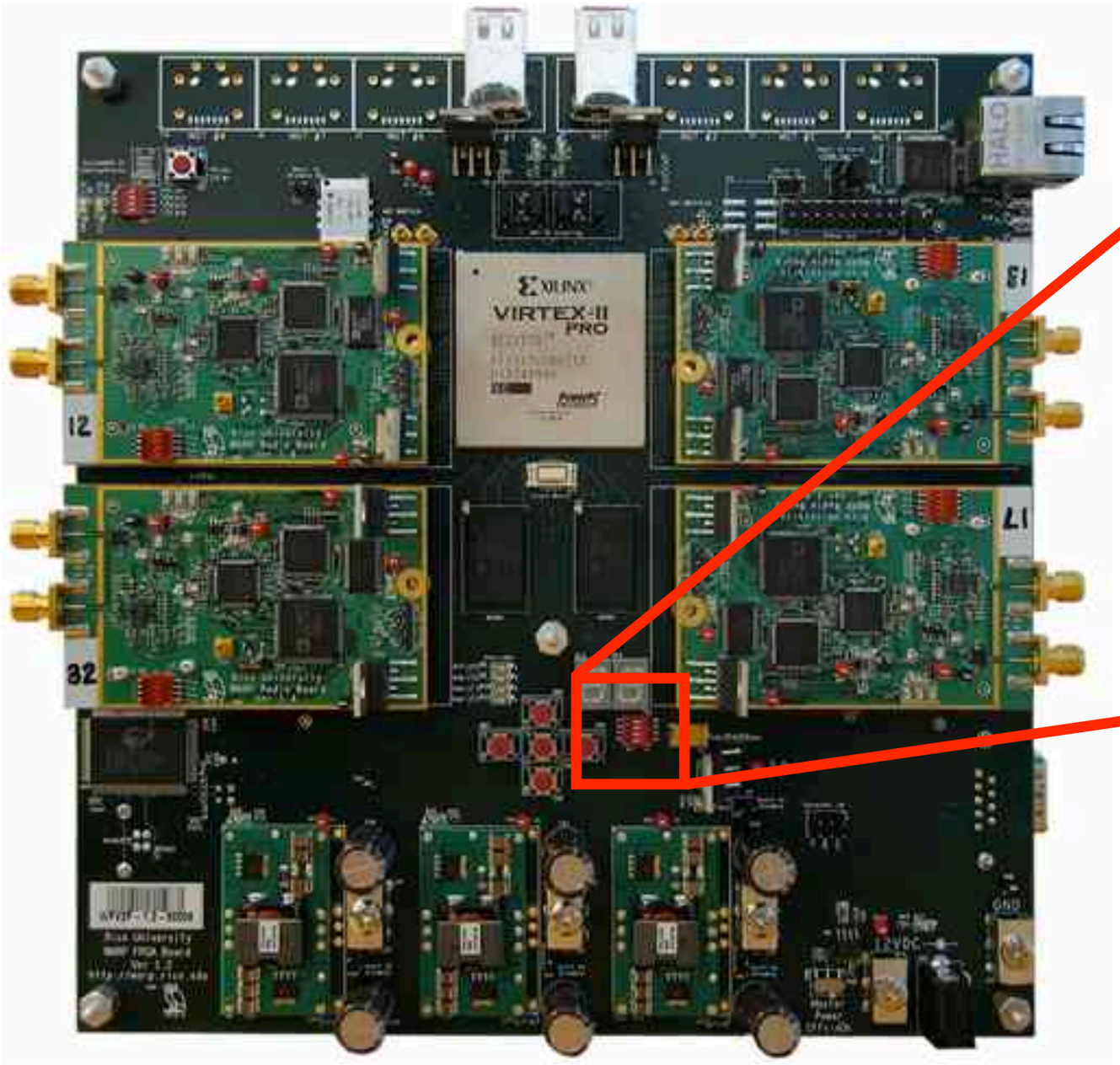


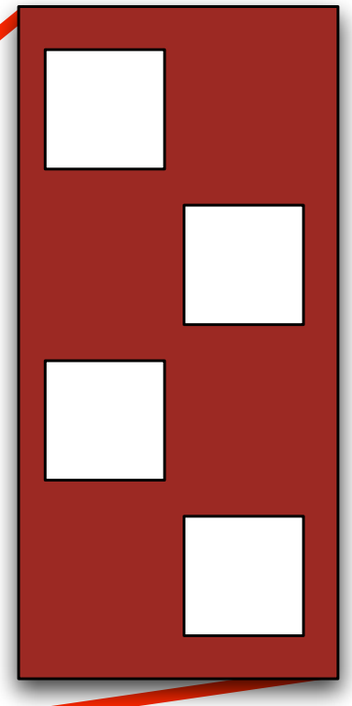




UDP Audio

uniMac Lab



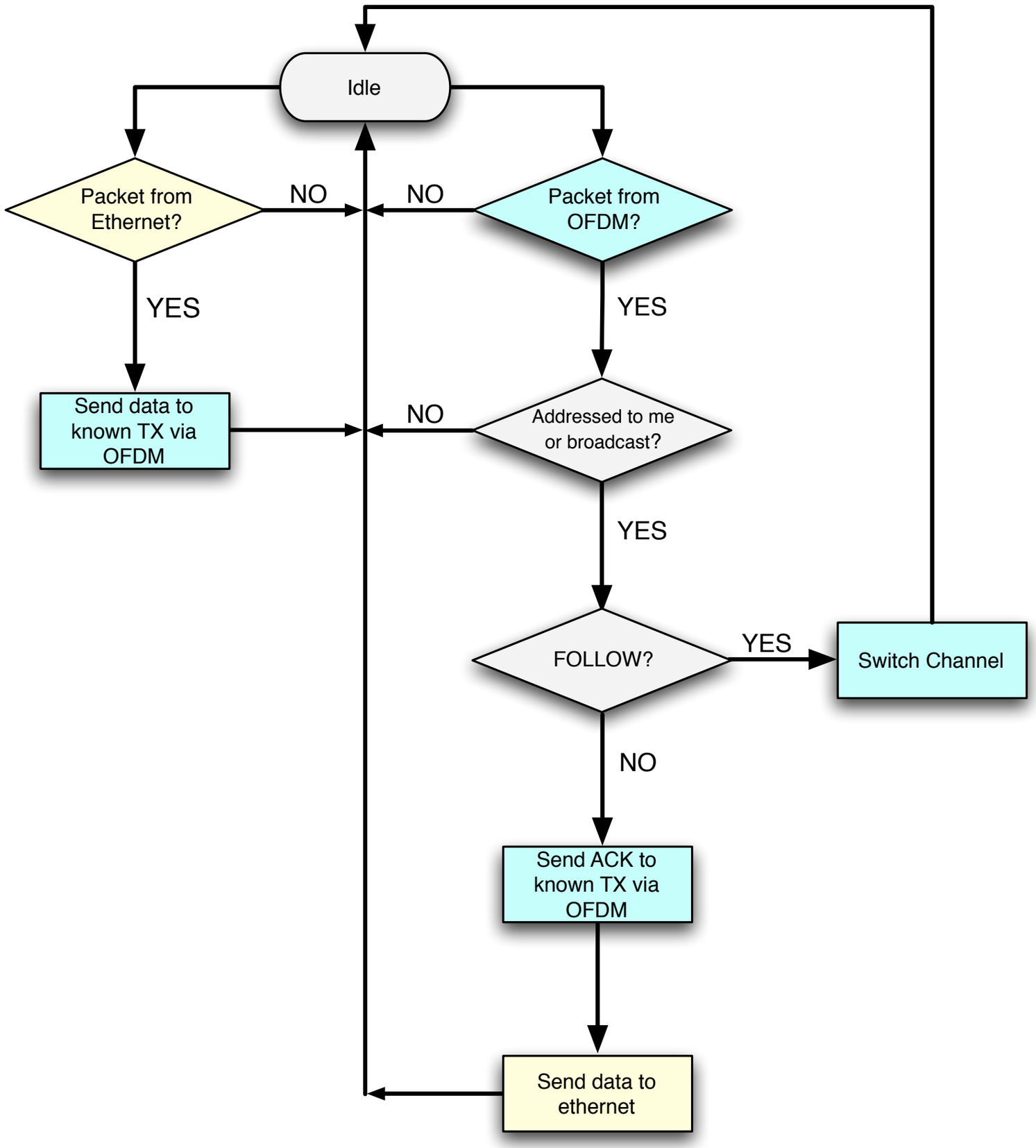


Most Significant Bit (MSB)

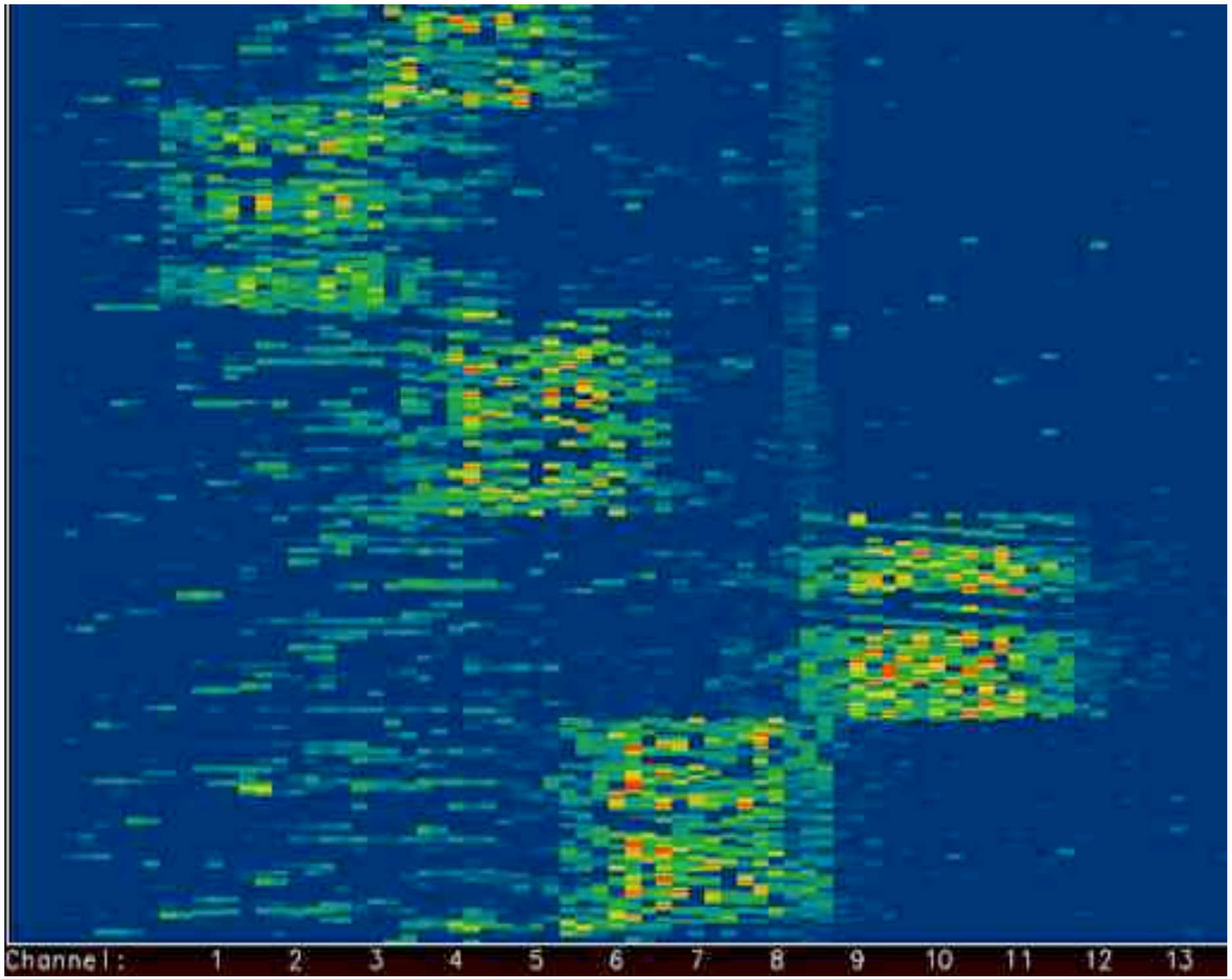
Least Significant Bit (LSB)

Node 5

hopMac


Idle
Packet from Ethernet?
NO
NO
Packet from OFDM?
YES
YES
Send data to known TX via OFDM
NO
Addressed to me or broadcast?
YES
FOLLOW?
YES
Switch Channel
NO
Send ACK to known TX via OFDM
Send data to ethernet

hopMac Lab



Questions?

Remember to use the API:

http://warp.rice.edu/WARP_API

Logistics

- Review forms
- Contacting us
  - Support & technical questions
    - http://warp.rice.edu/forums/
  - Hardware sales
    - http://warp.rice.edu/
    - warp-project@rice.edu