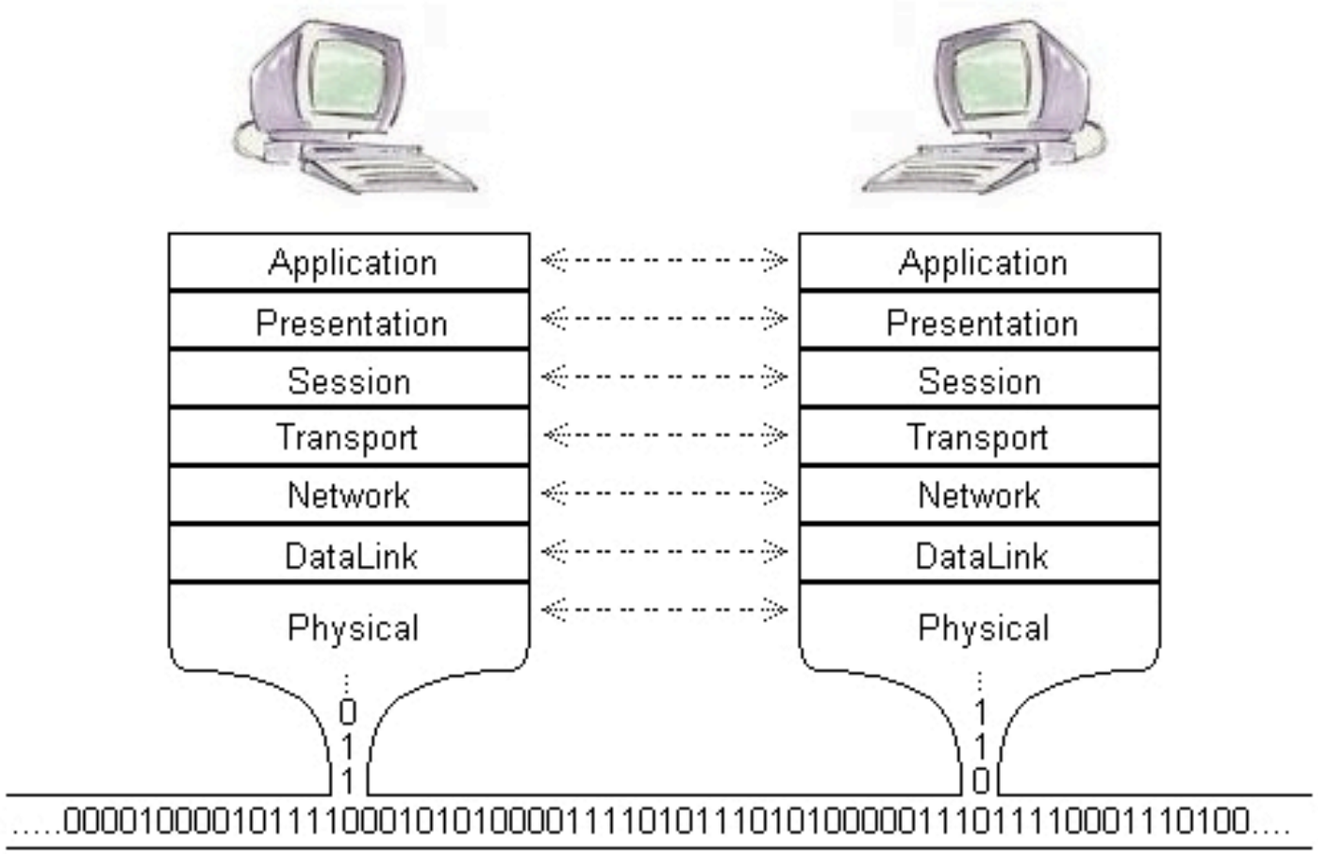
Networking on WARP

Chris Hunter
Rice University

WARP Workshop
March 23, 2007

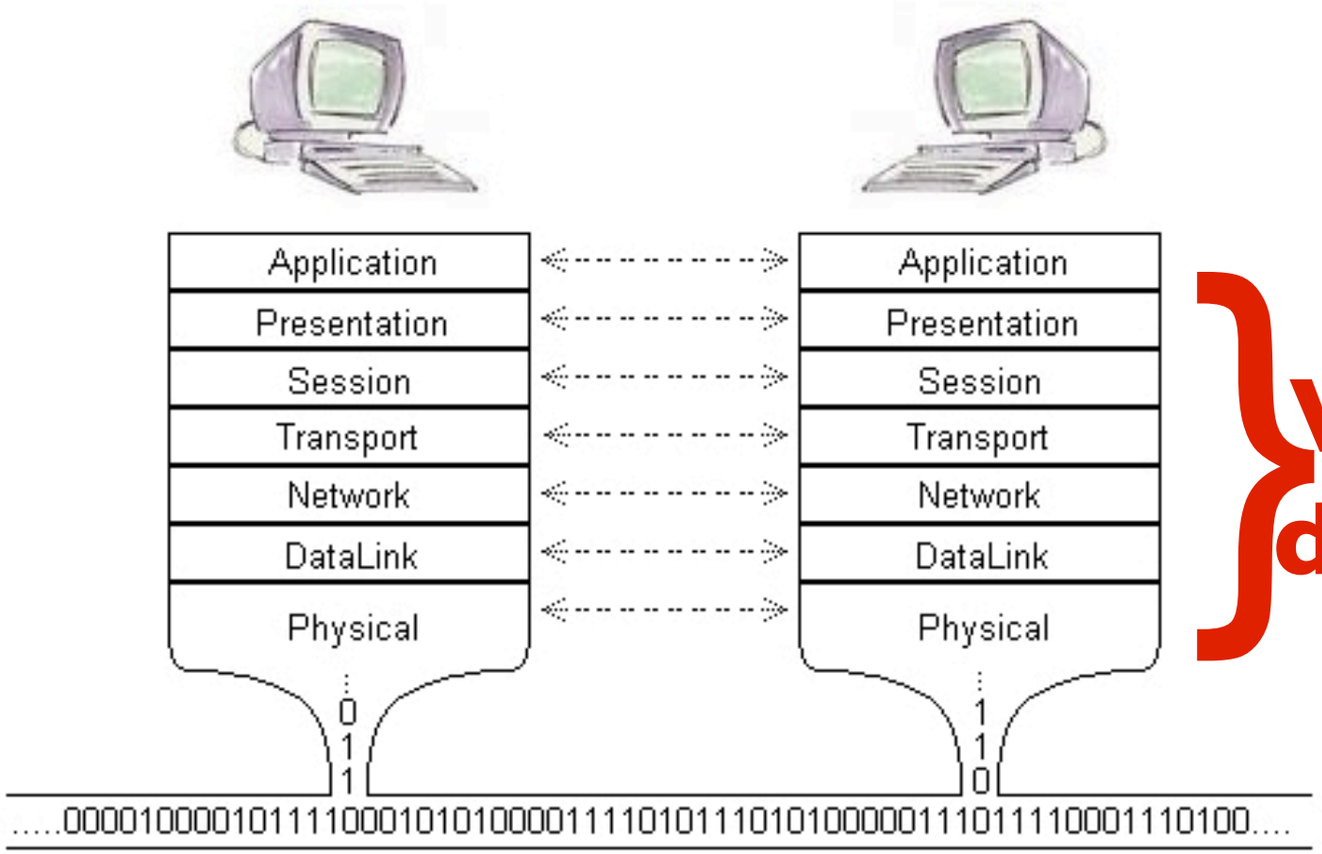
warp.rice.edu

Some Perspective - The OSI Model



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

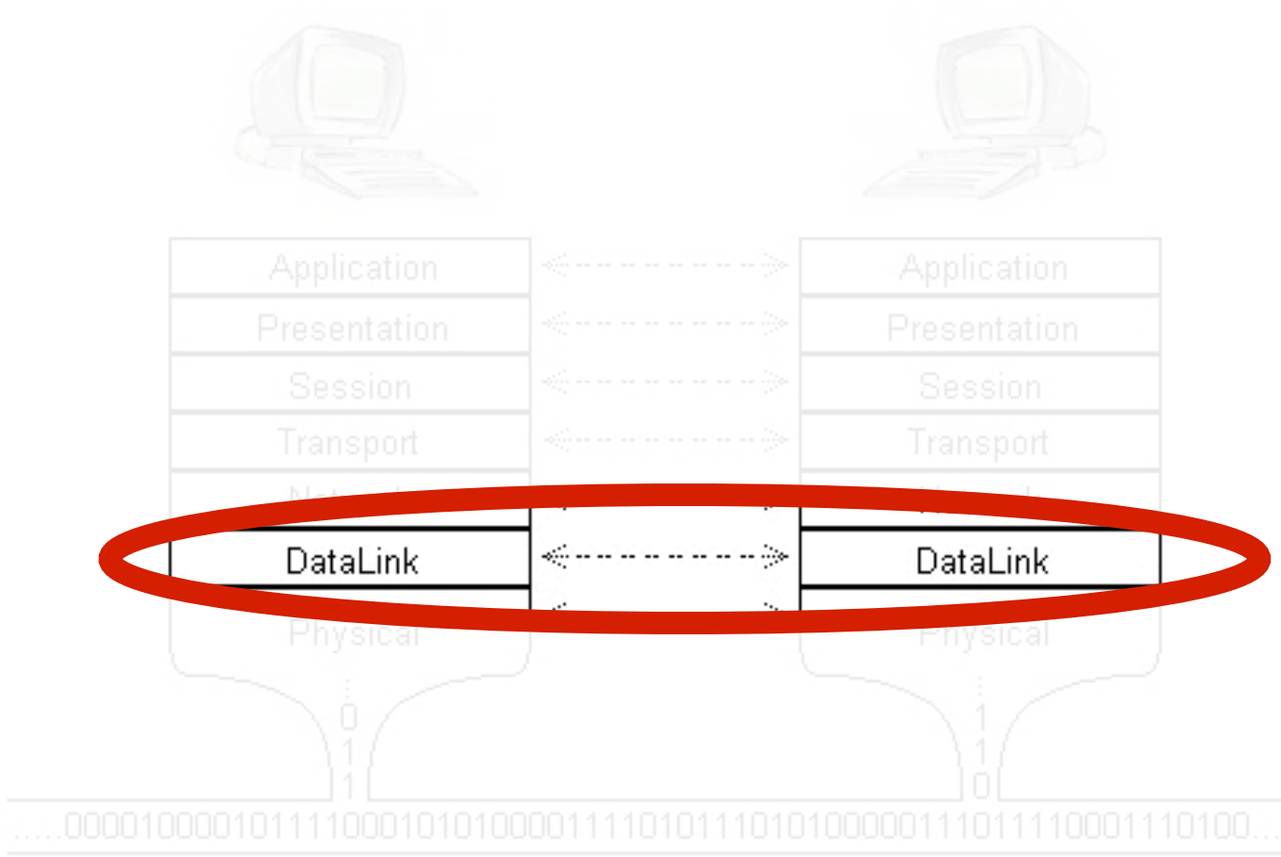
Some Perspective - The OSI Model



WARP can do them all

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

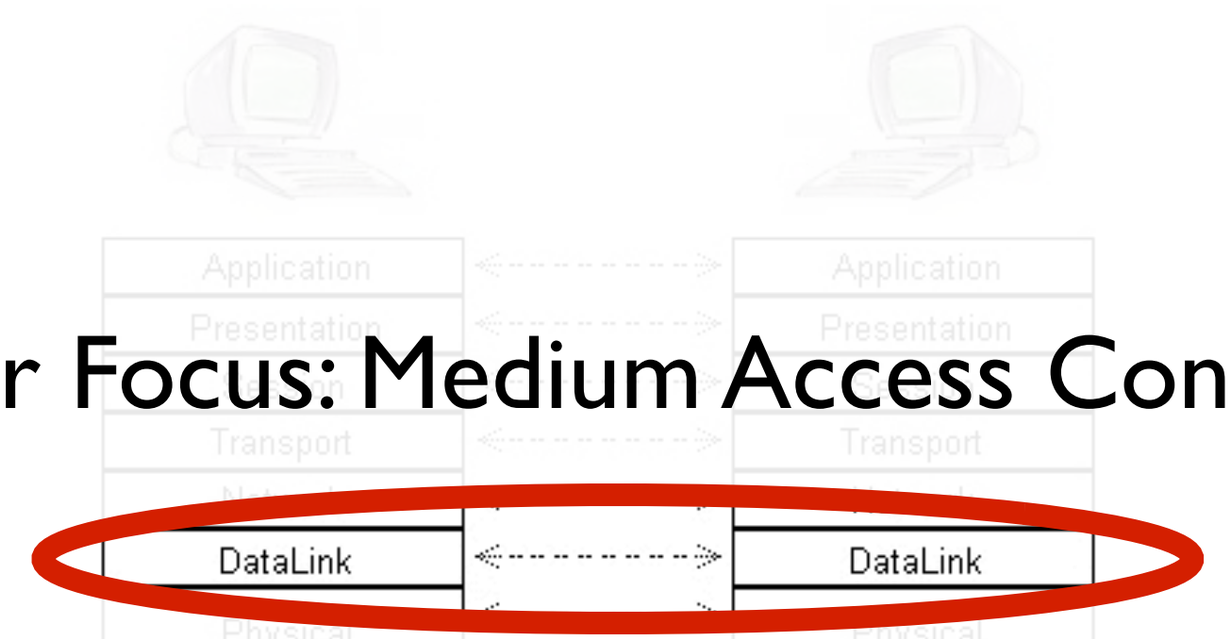
The OSI Model



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, MIMO, cross-layer gains, etc.
- All commercial 802.11 chipsets are closed

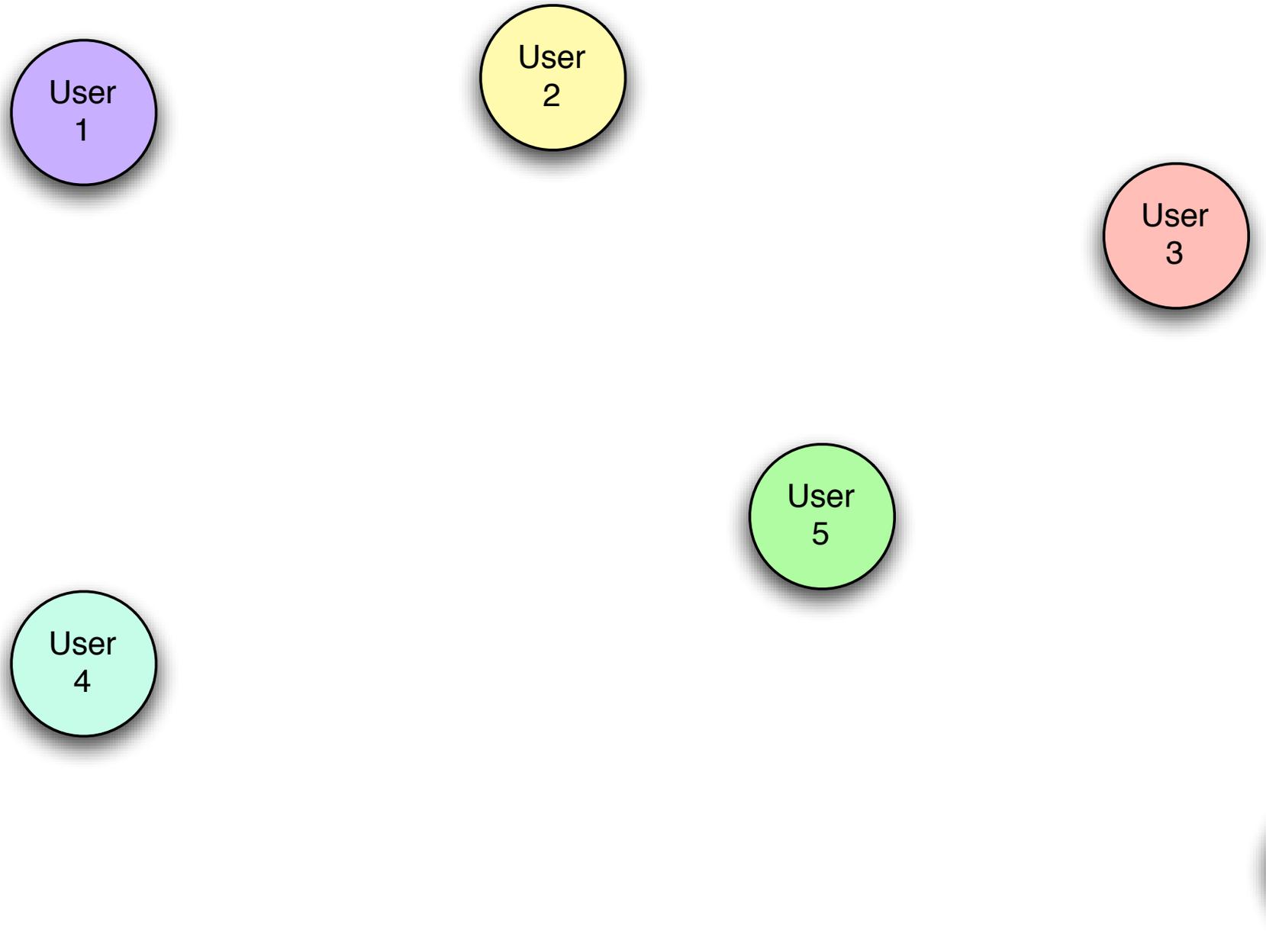
Source: http://upload.wikimedia.org/wikipedia/en/f/ff/Osi_model_trad.jpg

Outline

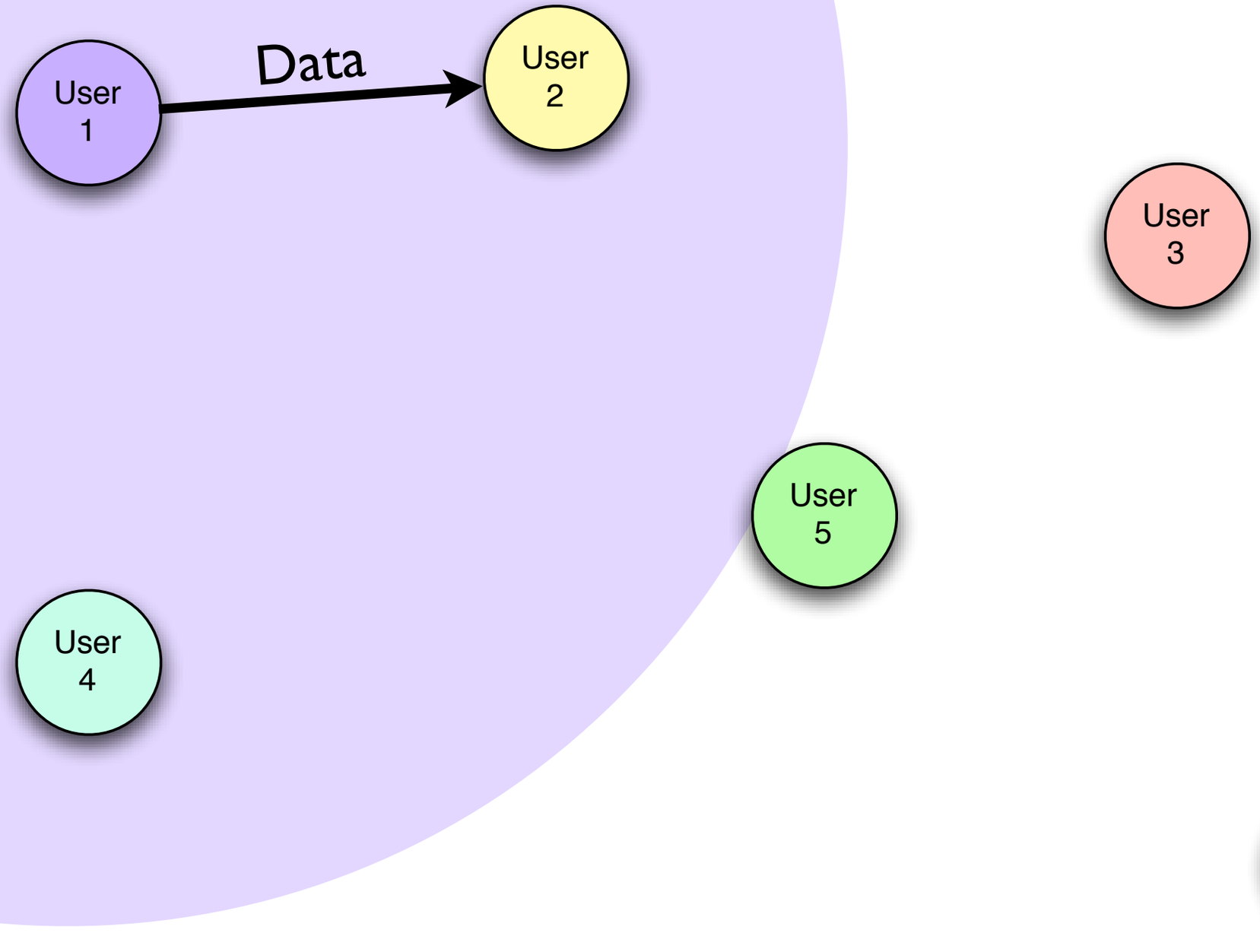
- Overview of Medium Access Control
- Design Realization
- WARPMAC Framework
- Detailed Example
- Lab Exercises

Medium Access Control Overview

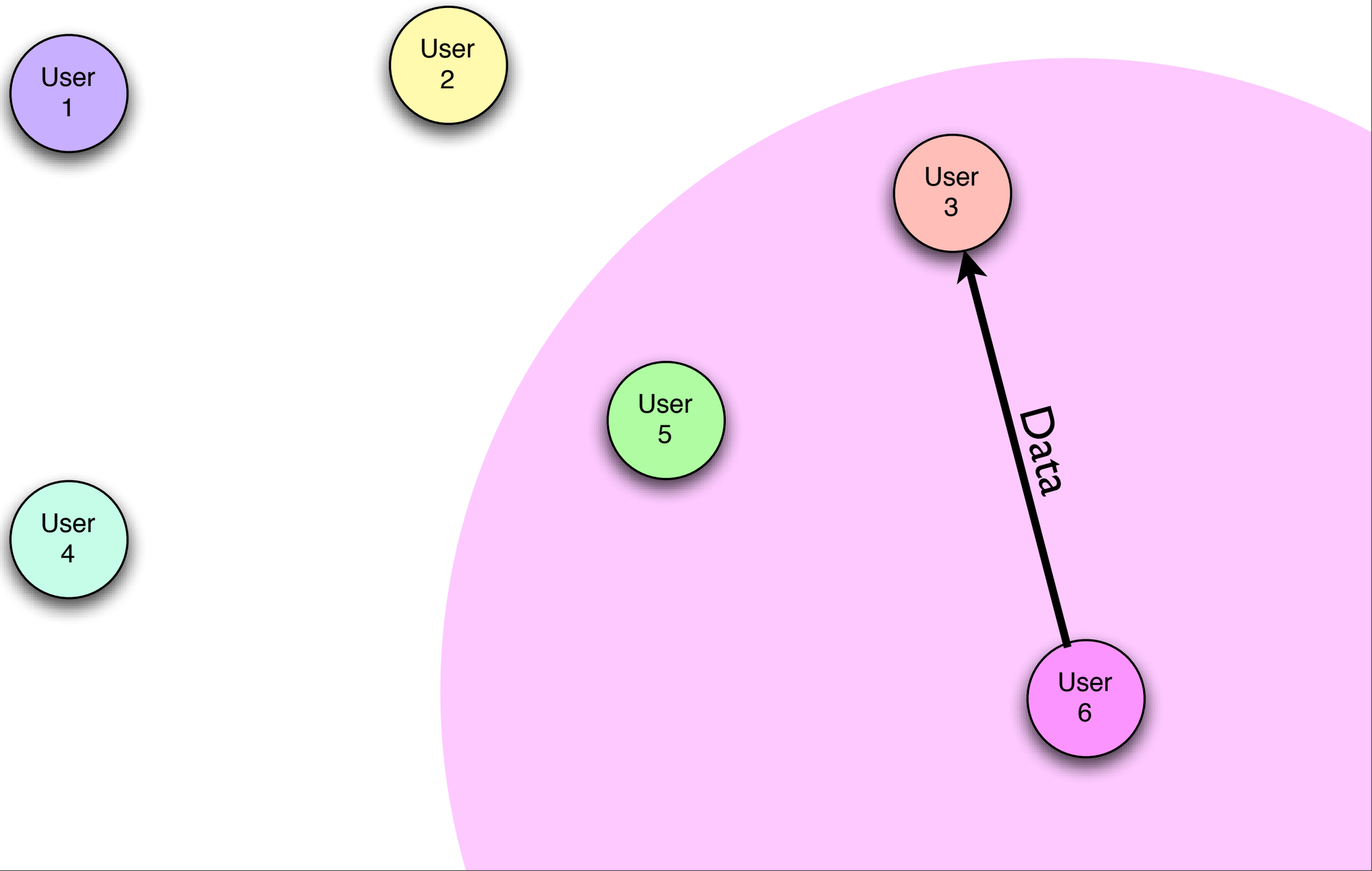
What is a MAC?



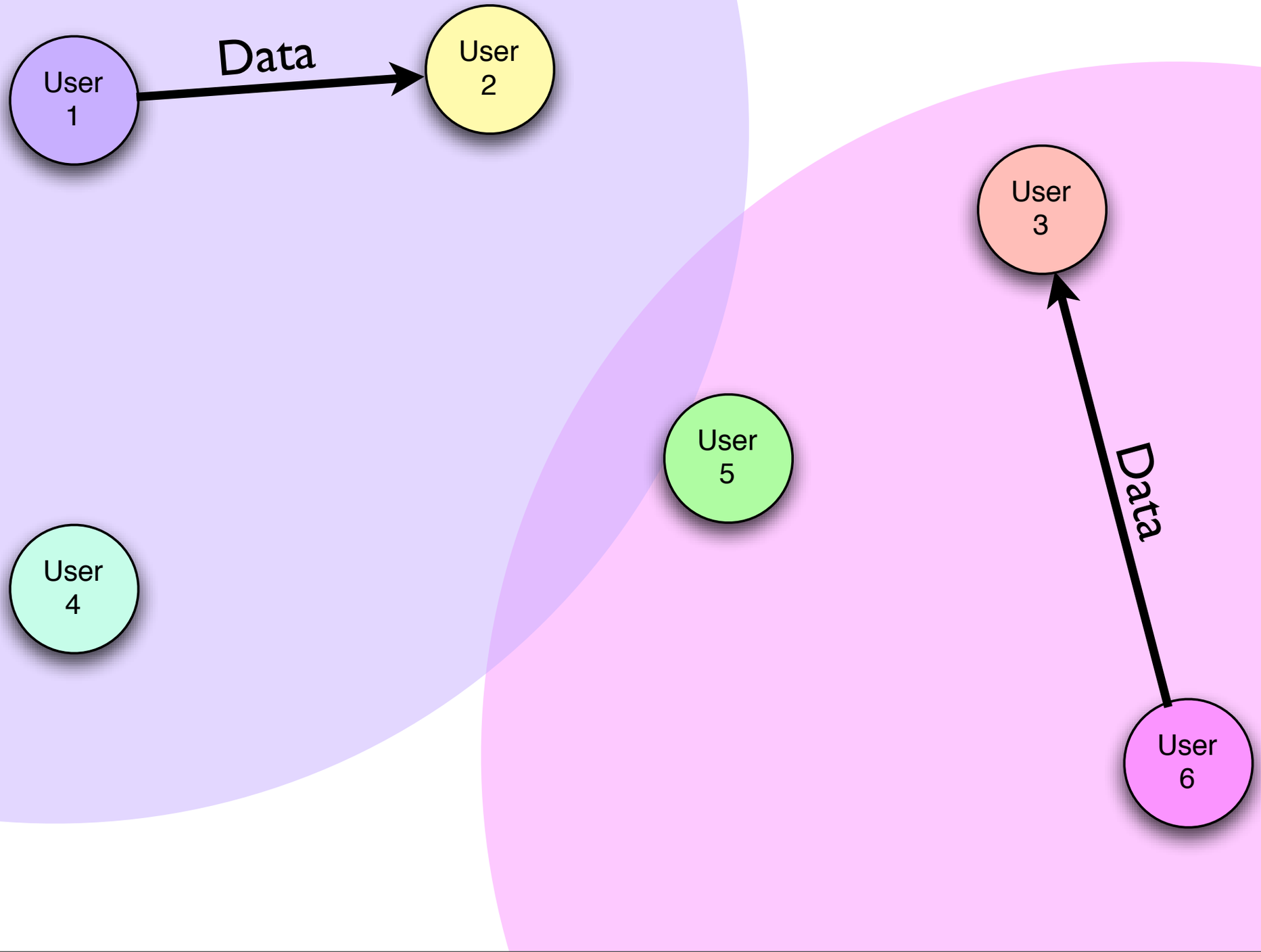
What is a MAC?



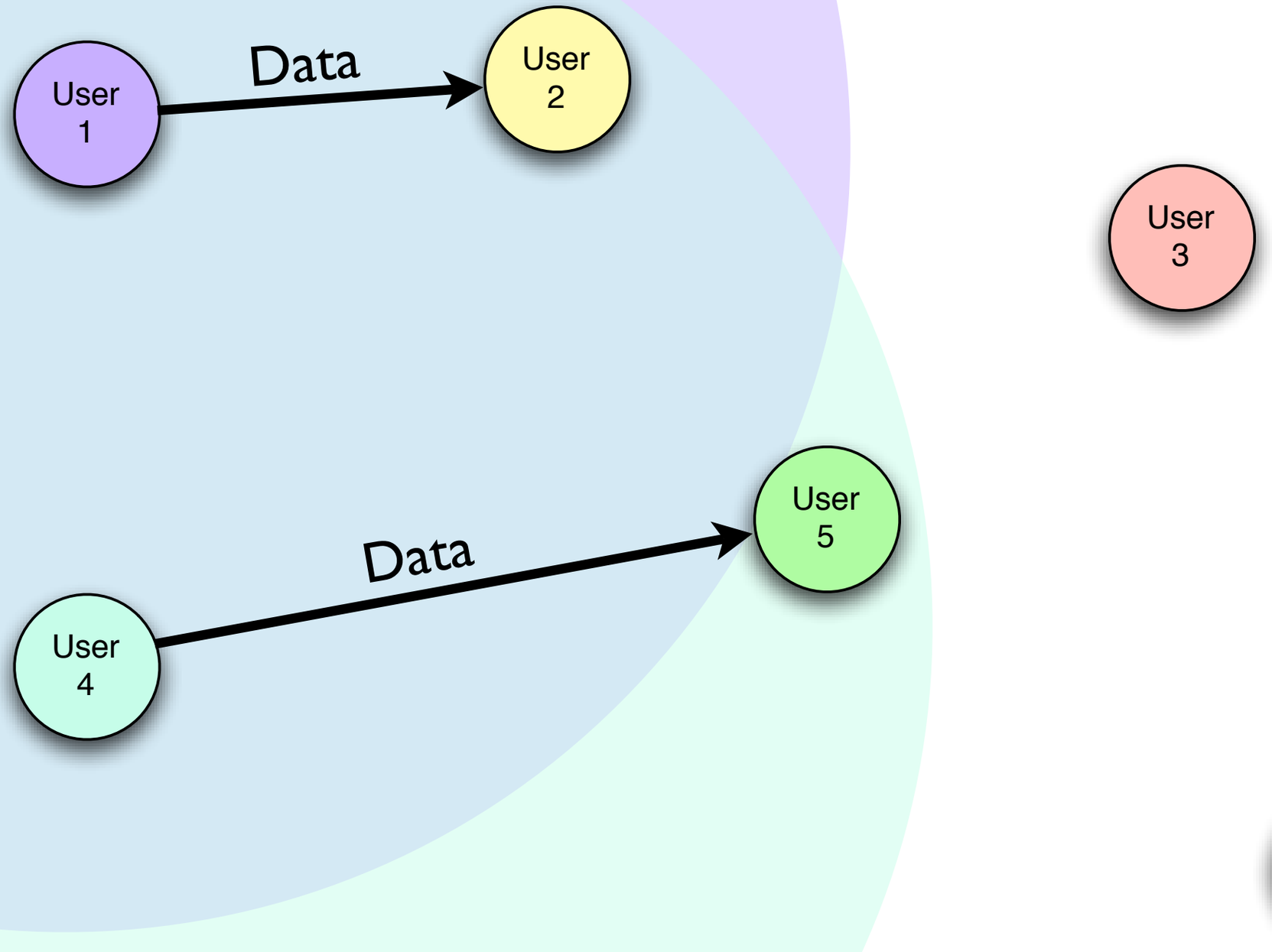
What is a MAC?



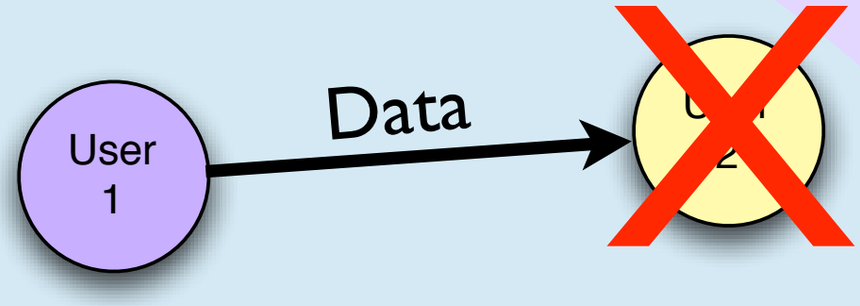
What is a MAC?

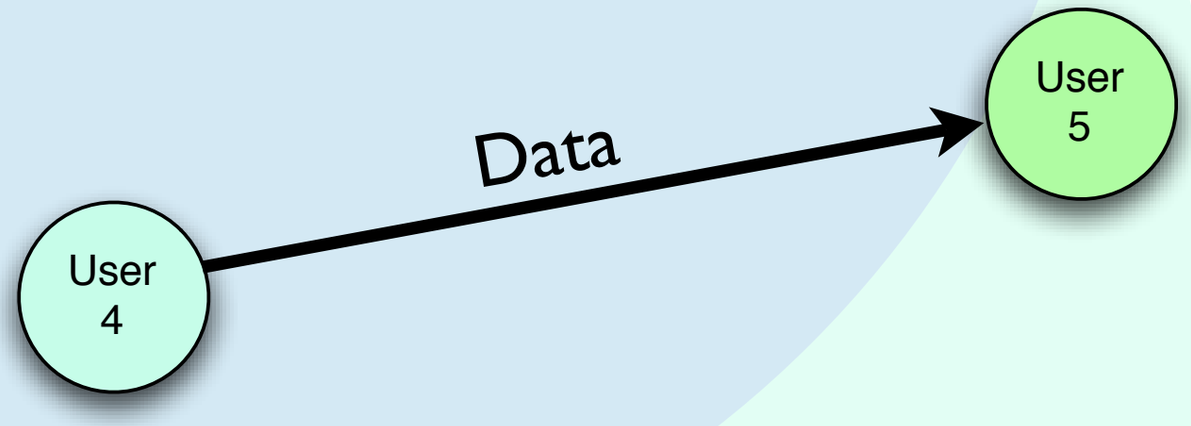


What is a MAC?

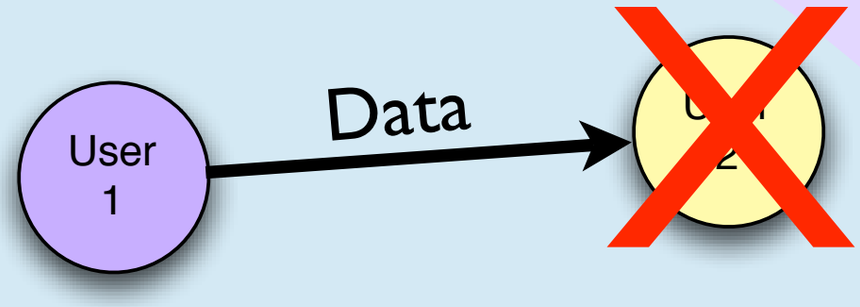


What is a MAC?

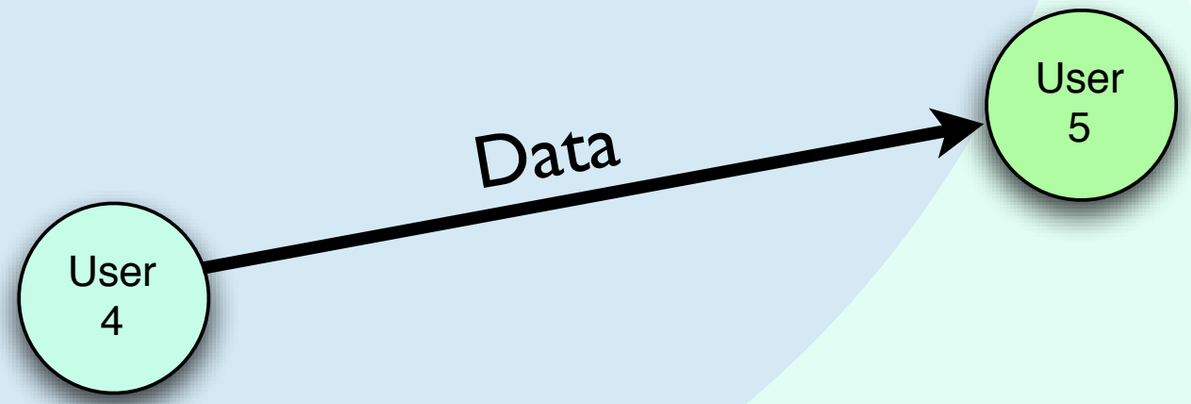




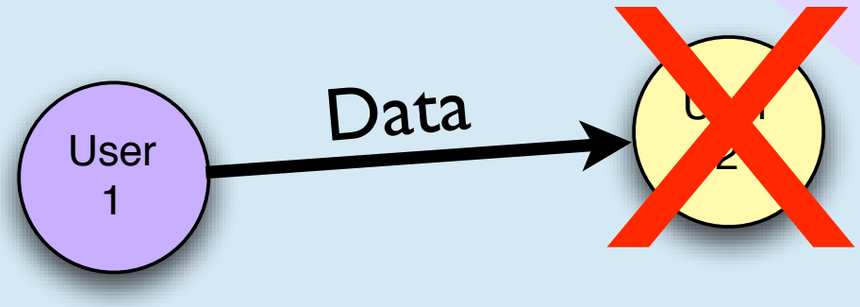
What is a MAC?



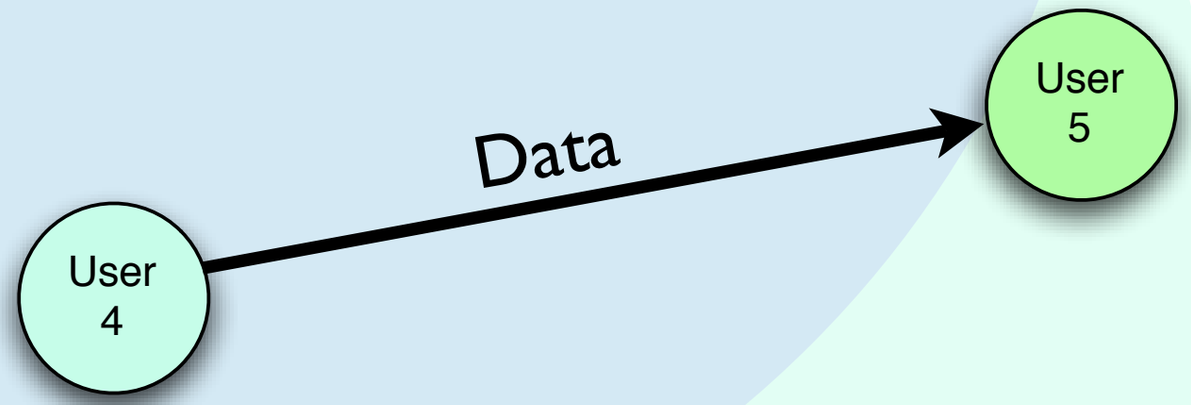
Received a jumbled packet... infer a packet collision



What is a MAC?

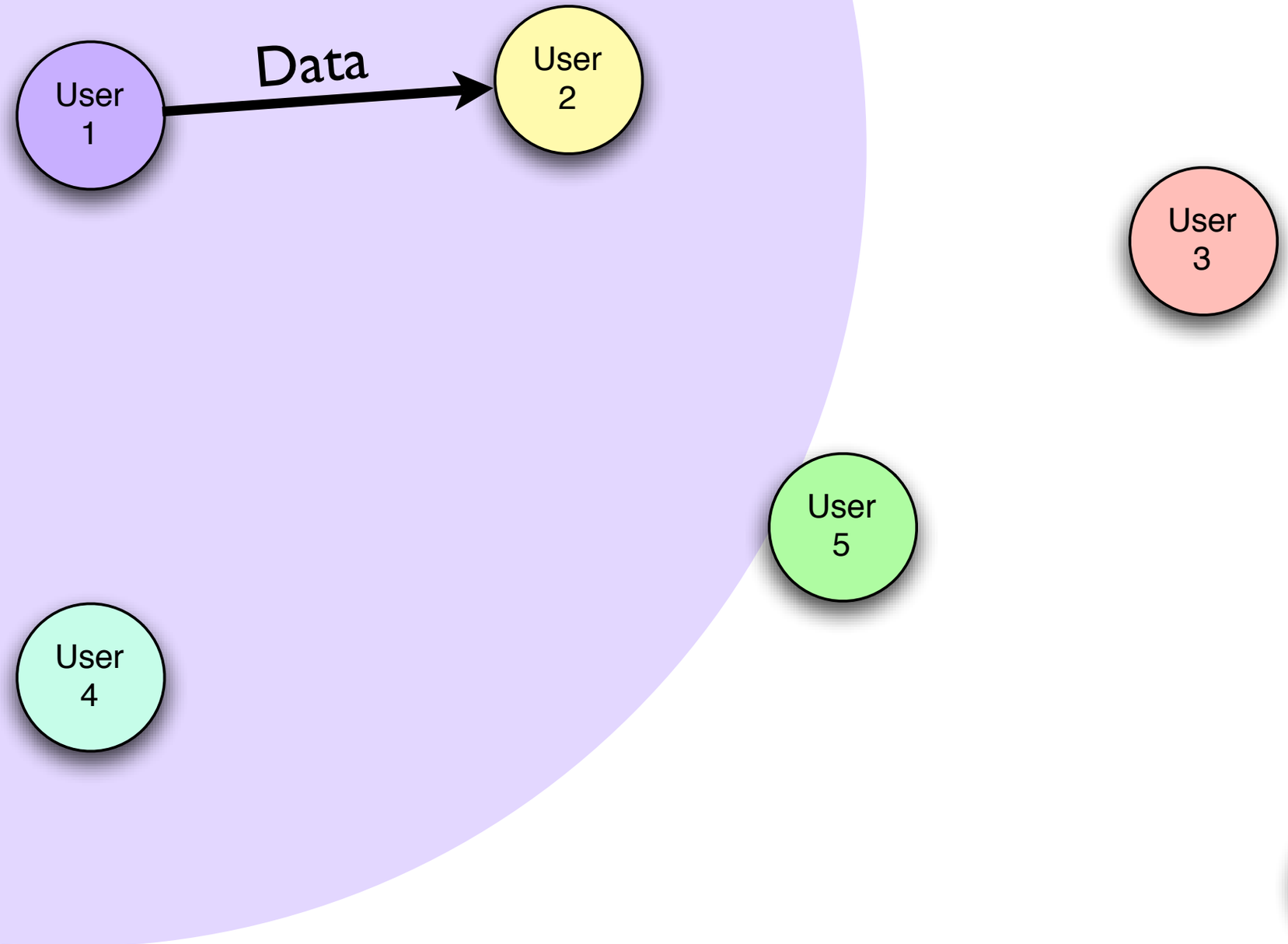


Received a jumbled packet... infer a packet collision

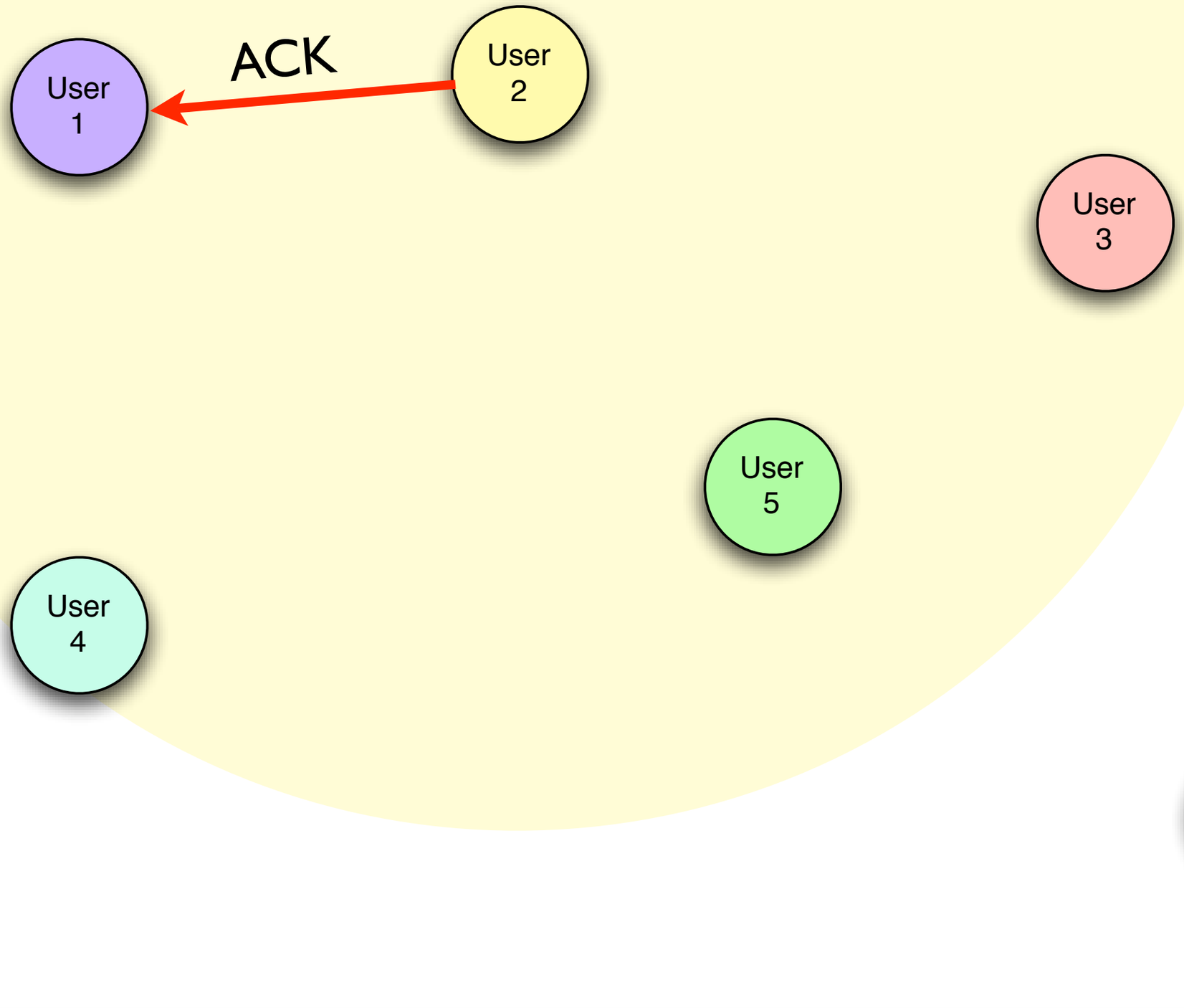


What if we ACK every transmit, and retransmit when we receive no ACK?

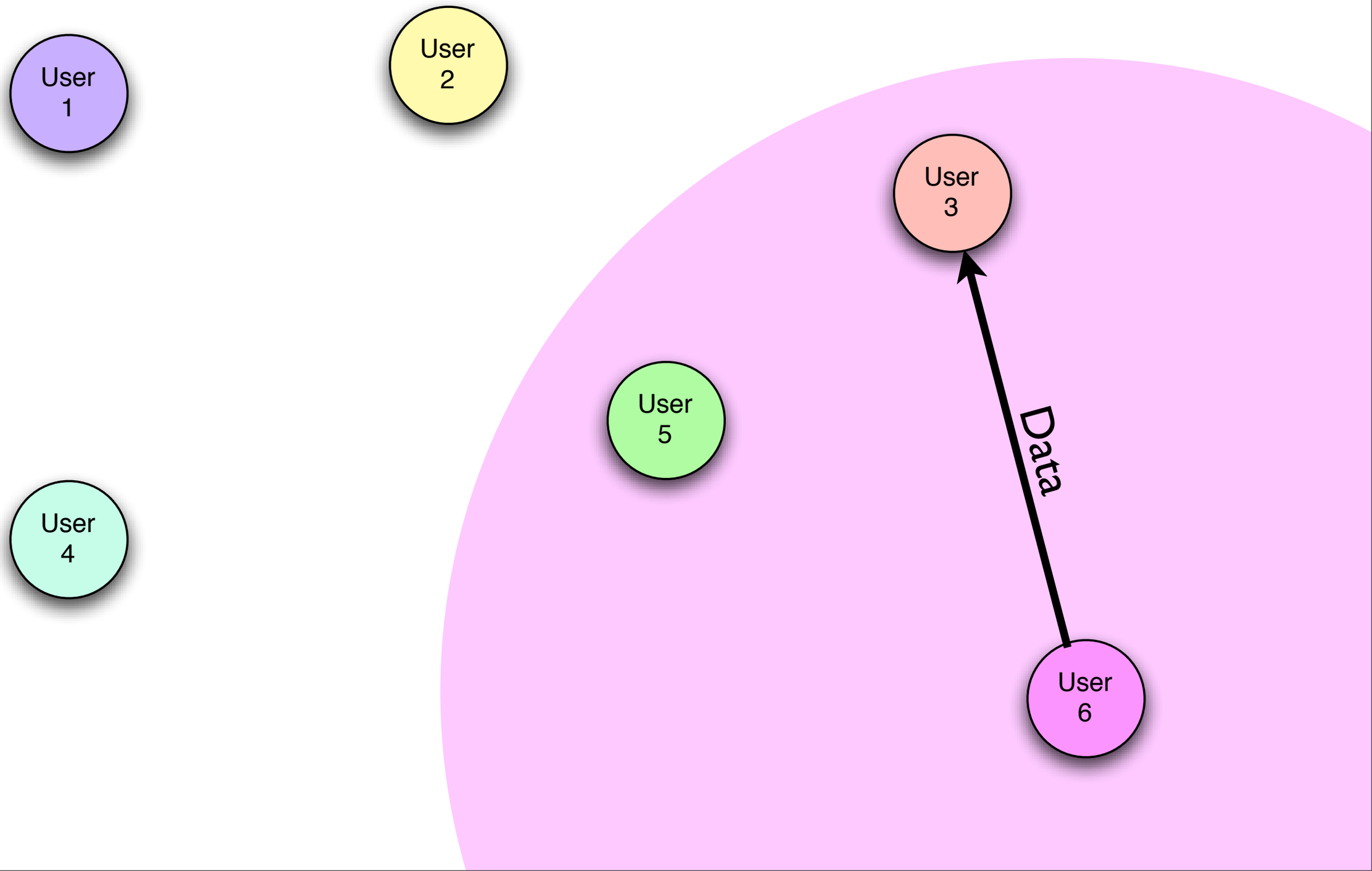
What is a MAC?



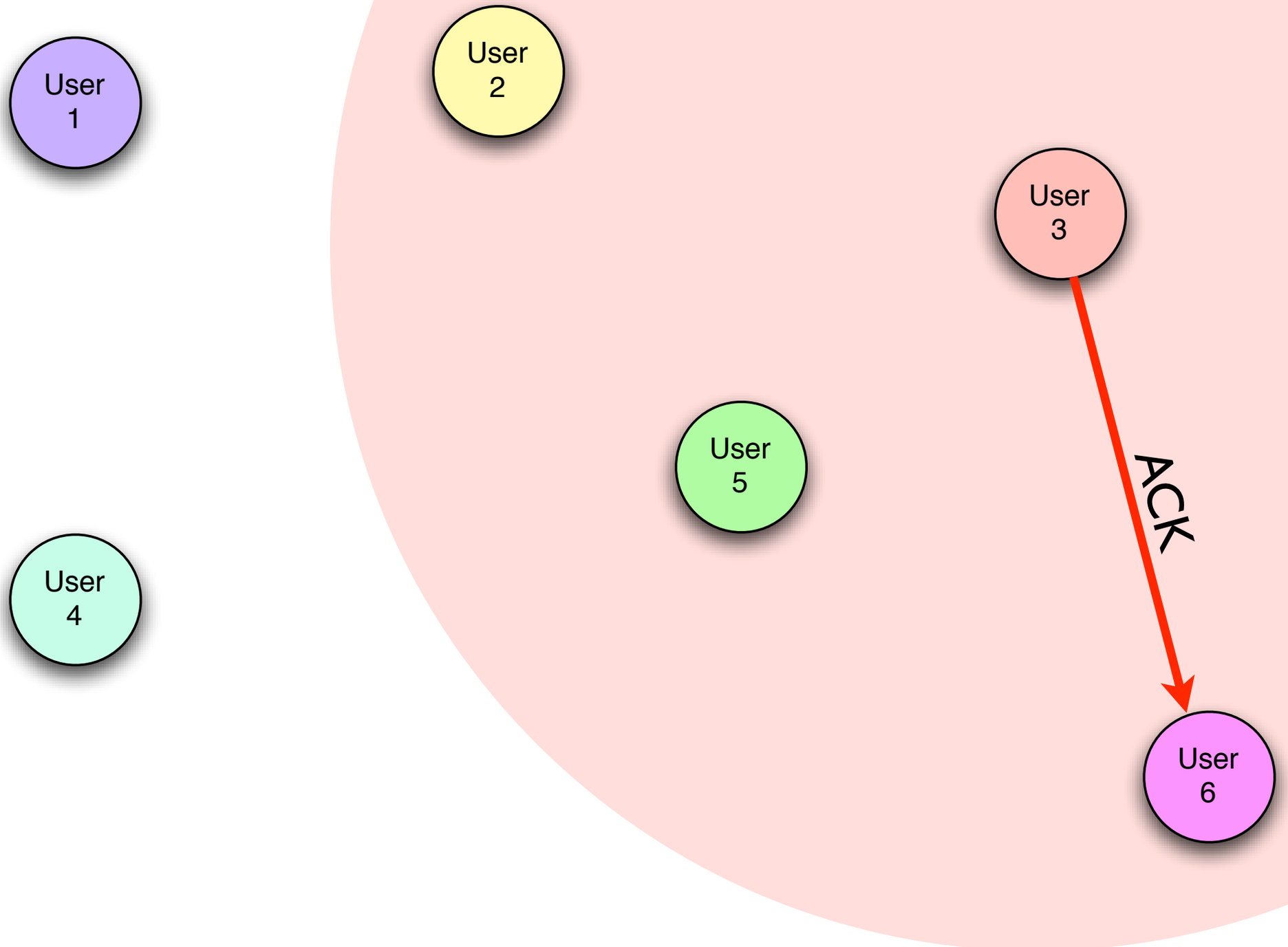
What is a MAC?



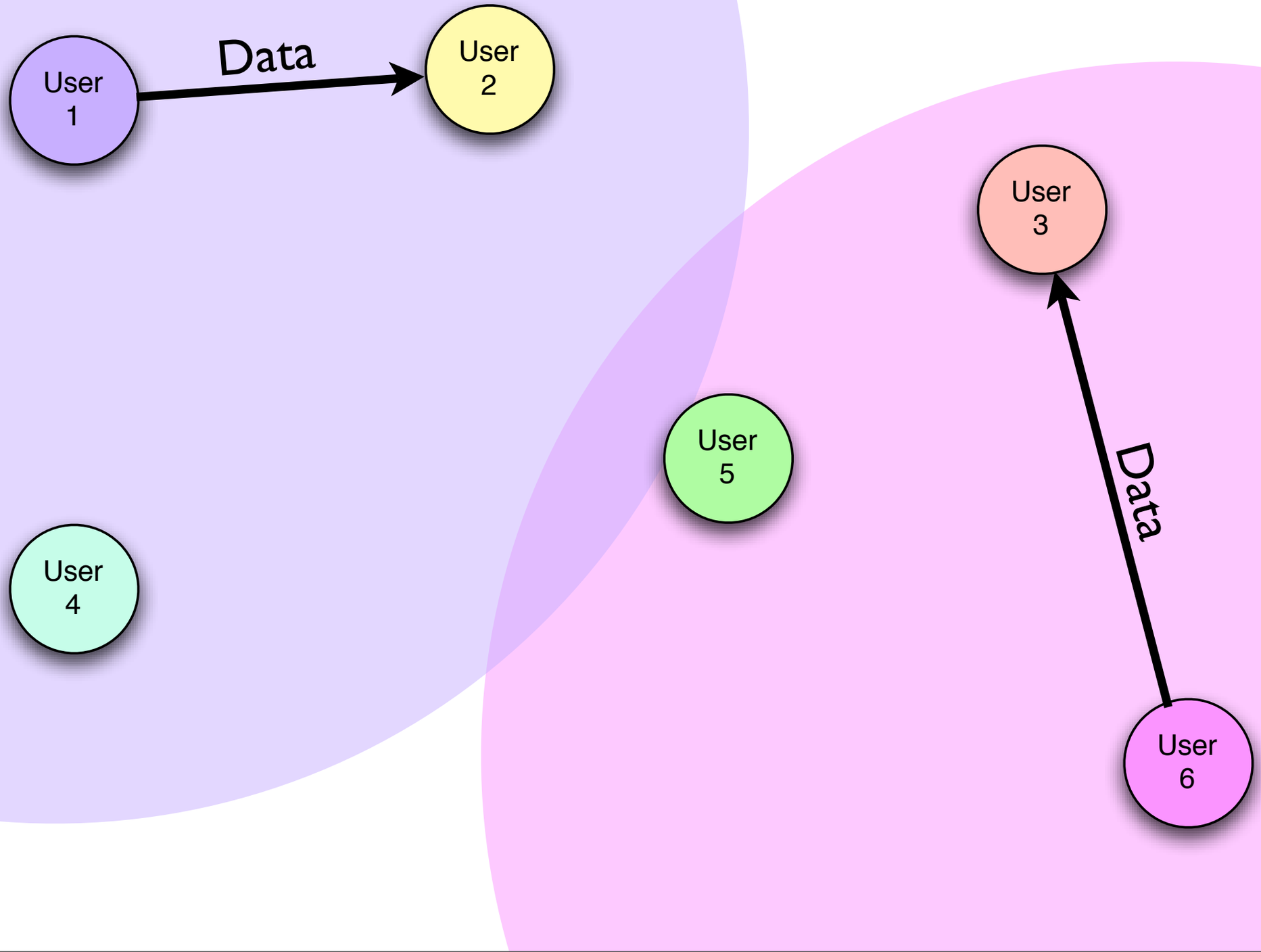
What is a MAC?



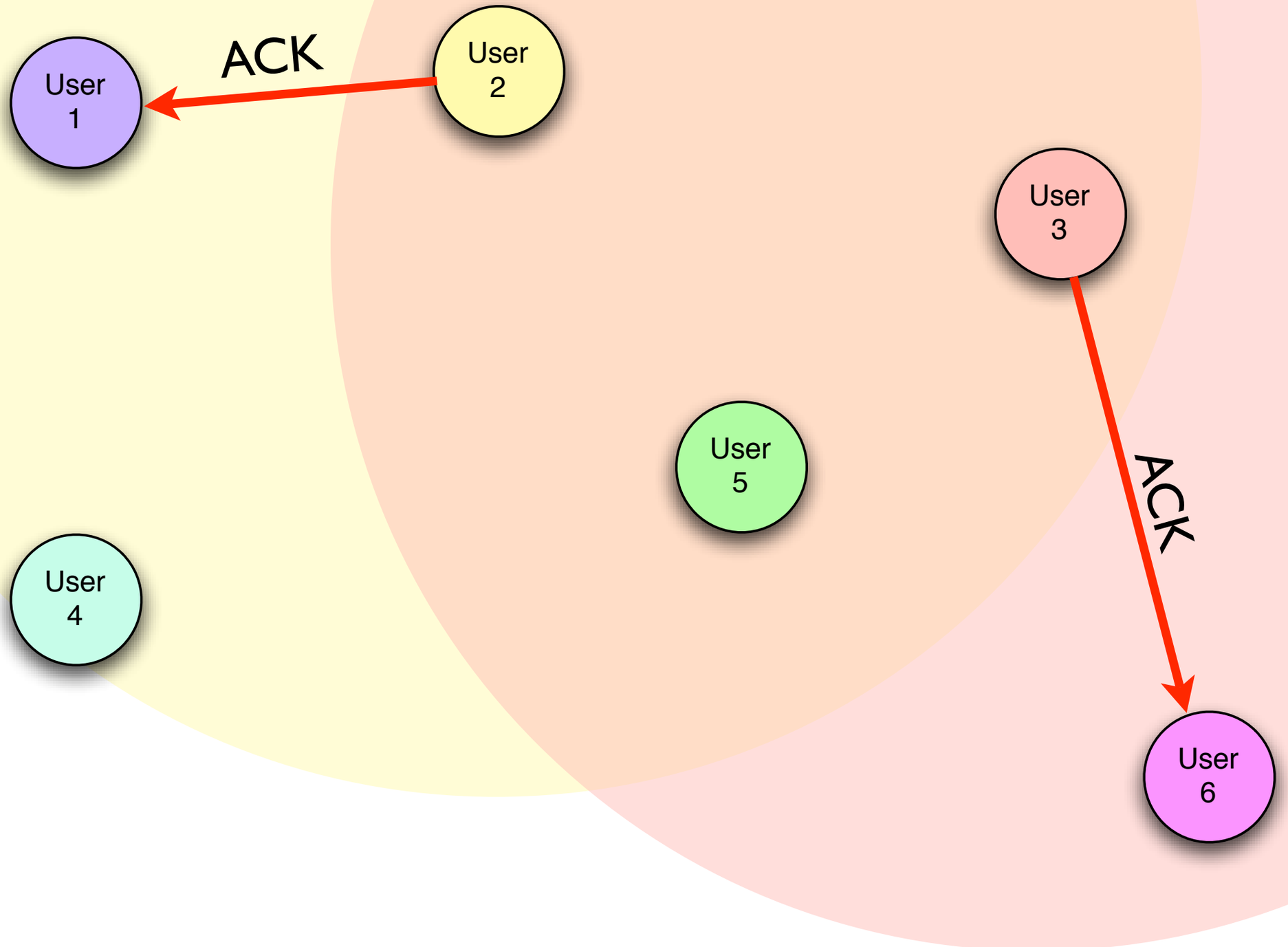
What is a MAC?



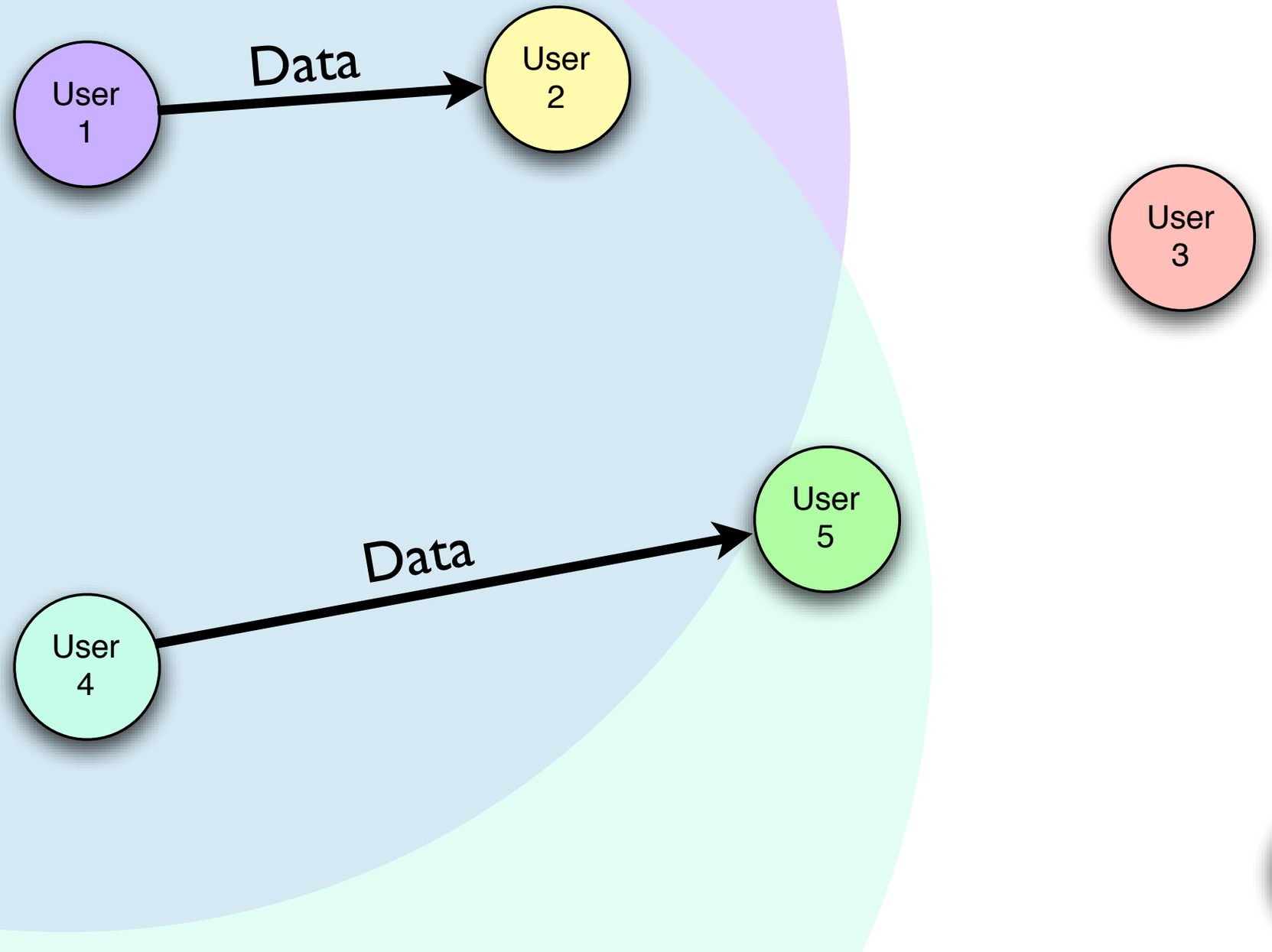
What is a MAC?



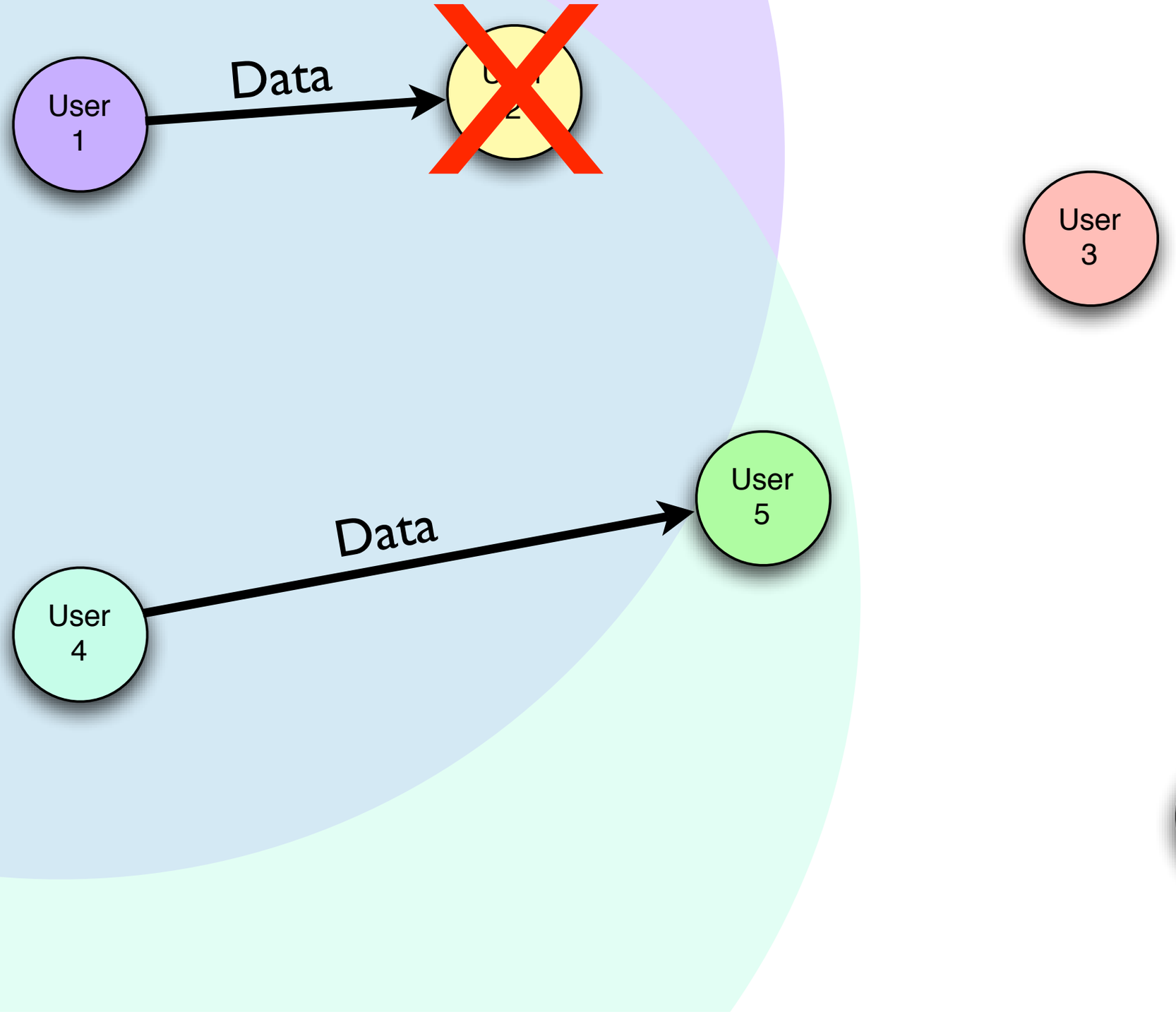
What is a MAC?



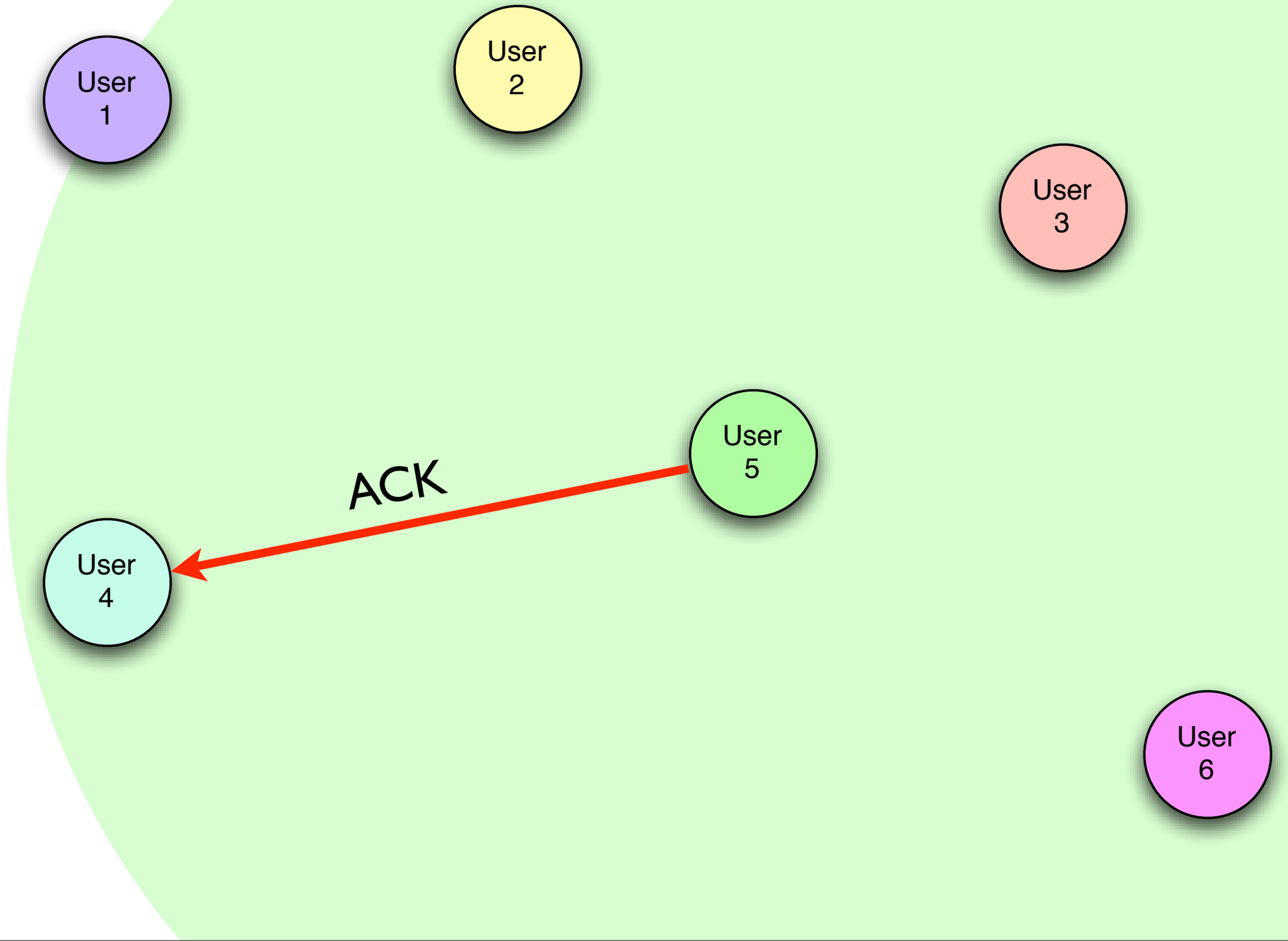
What is a MAC?



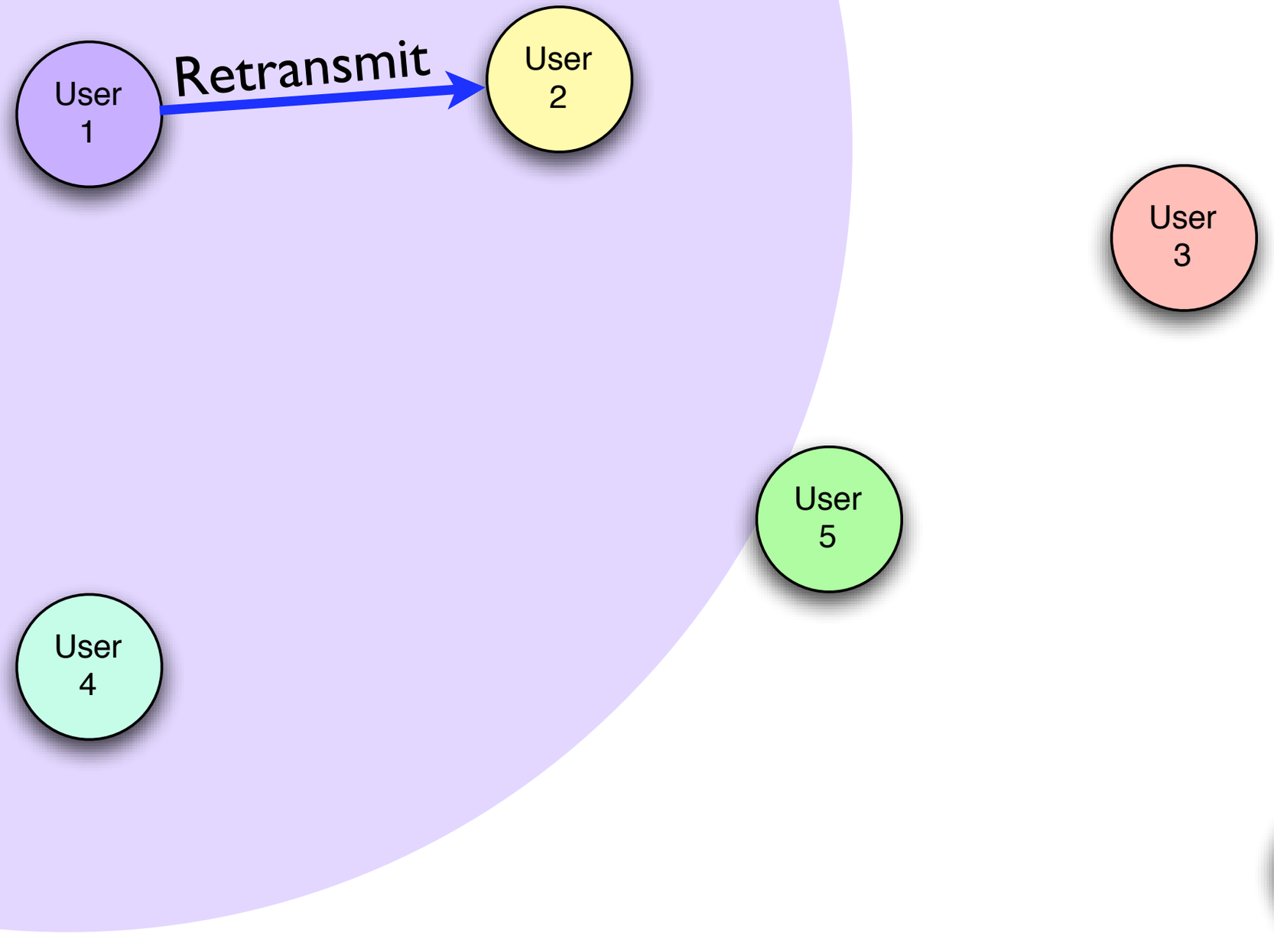
What is a MAC?



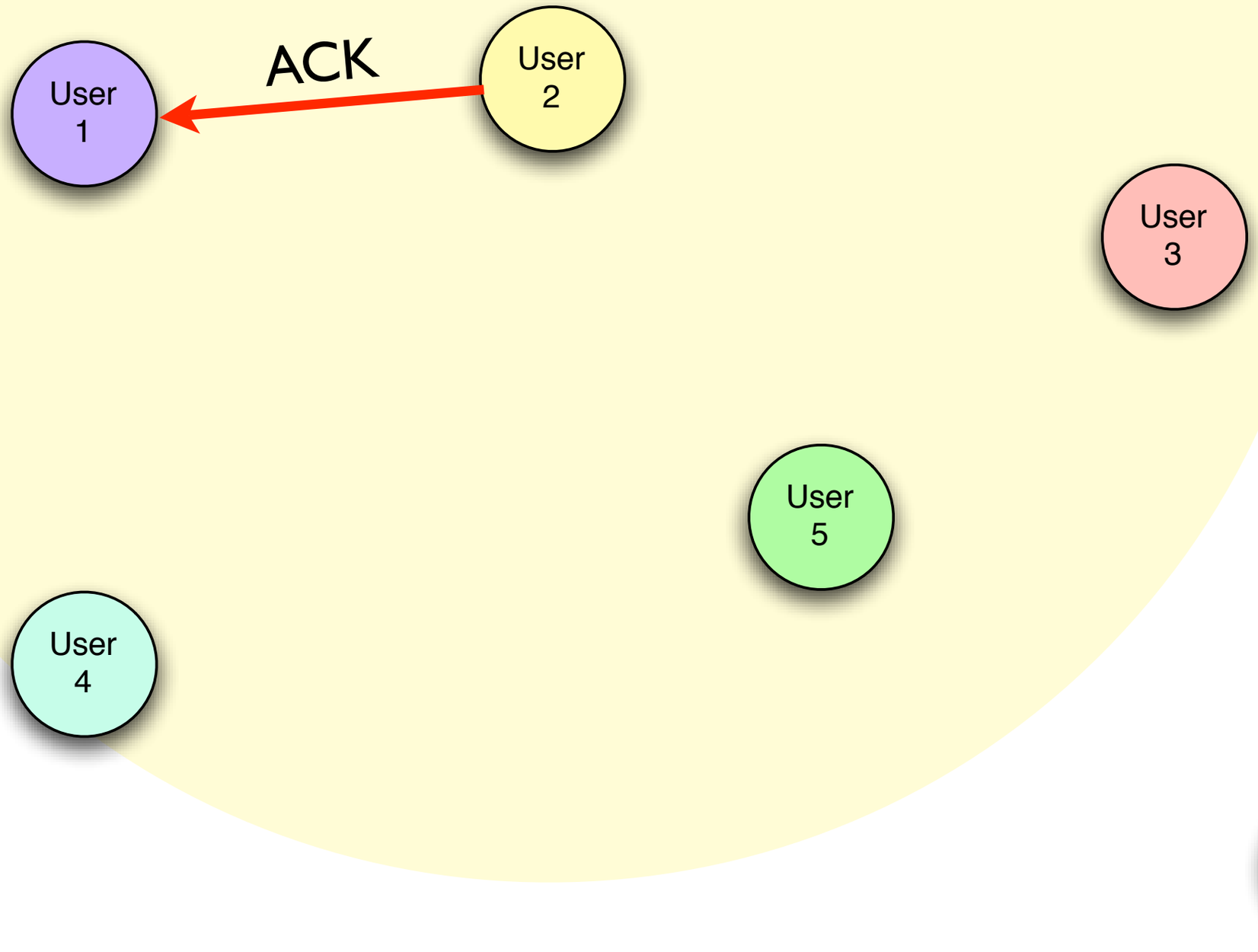
What is a MAC?



What is a MAC?



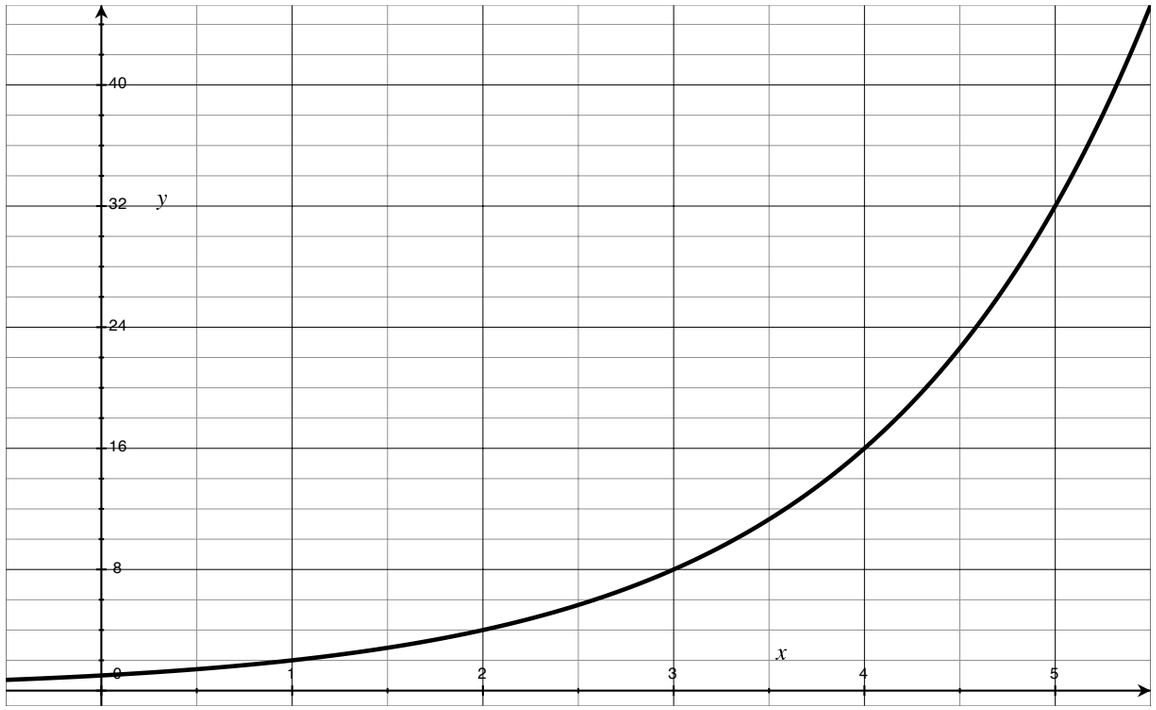
What is a MAC?



Random Backoffs

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

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



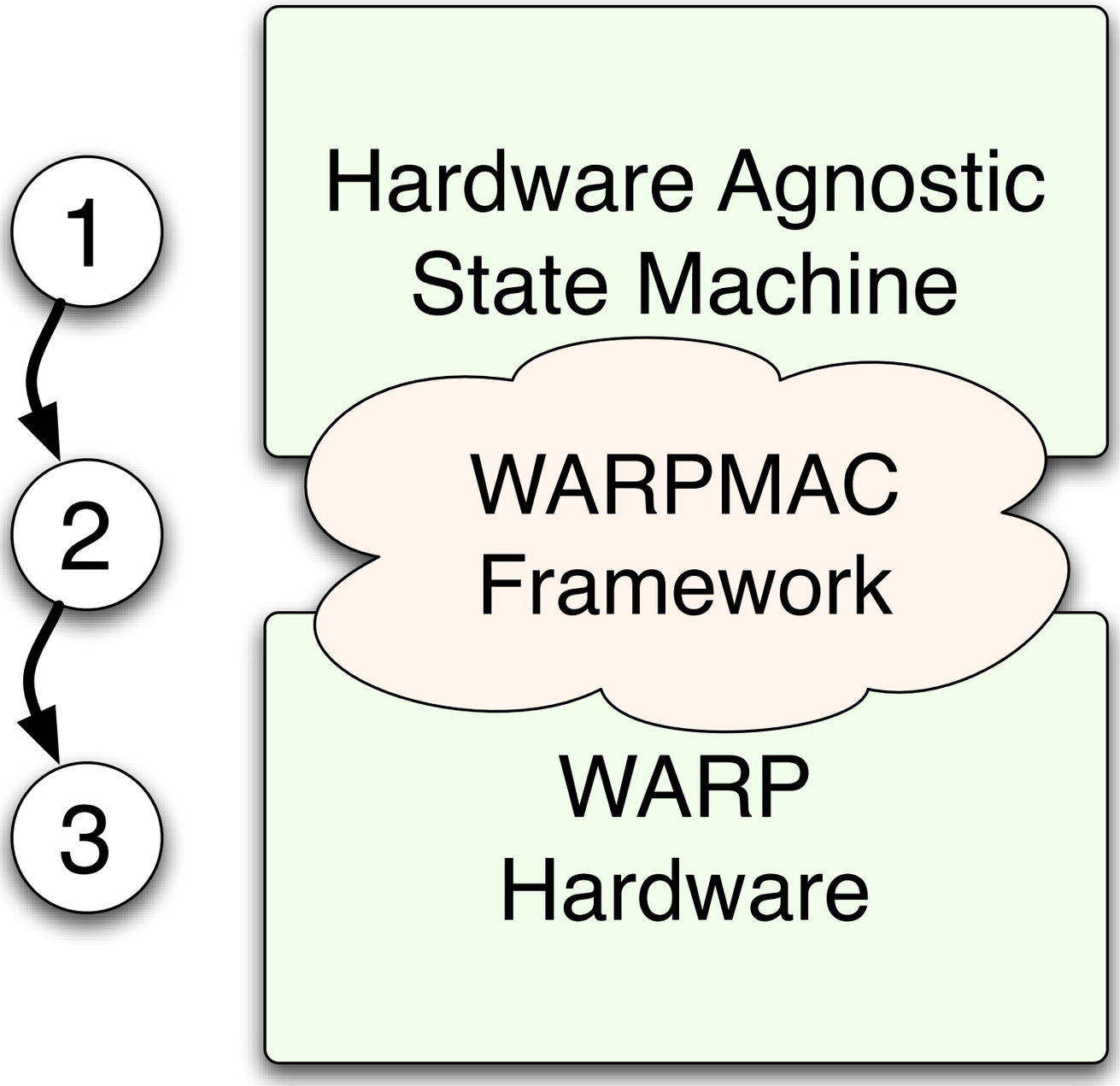
Contention Window increases over time

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

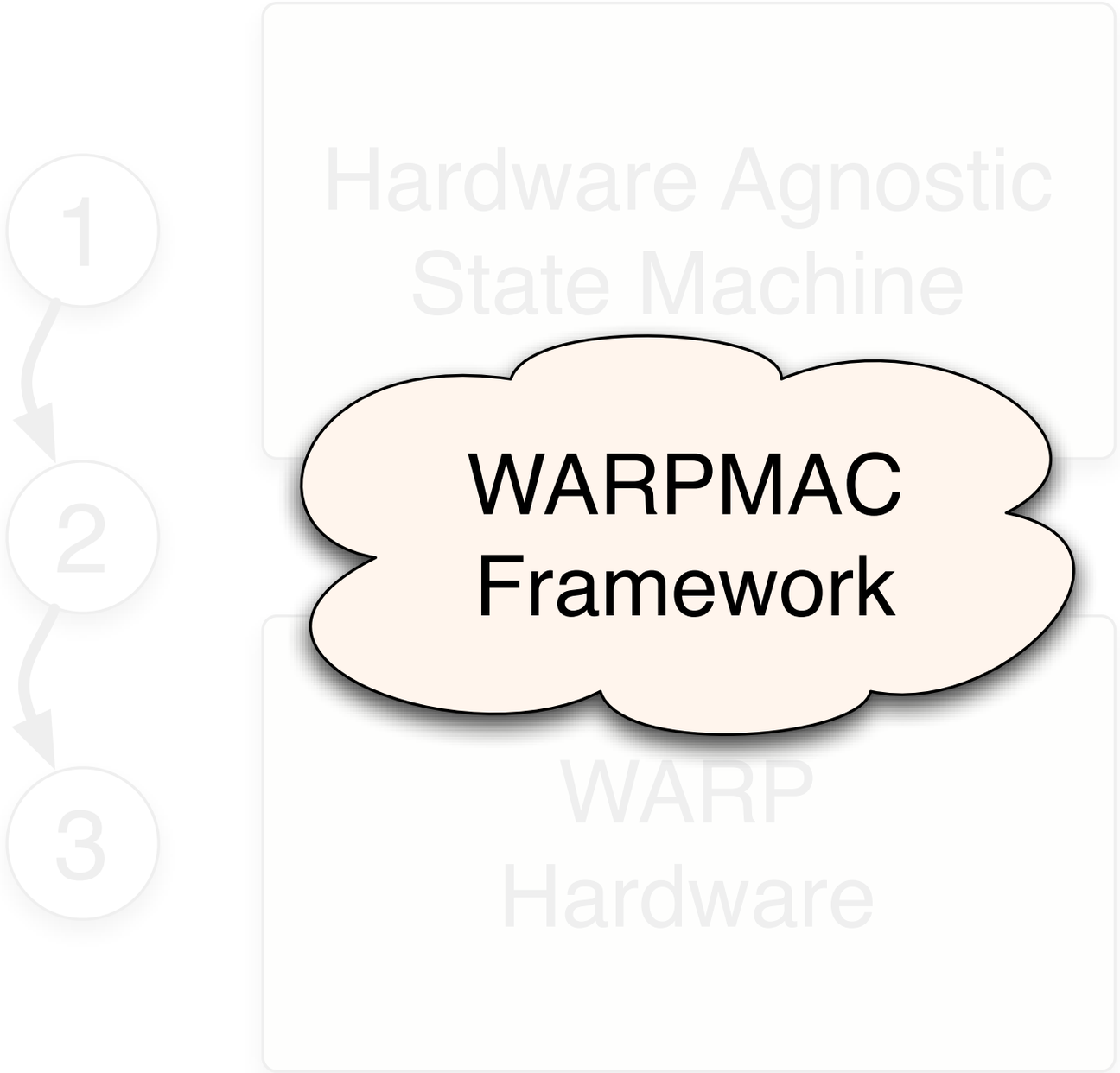
Design Realization

Design Realization



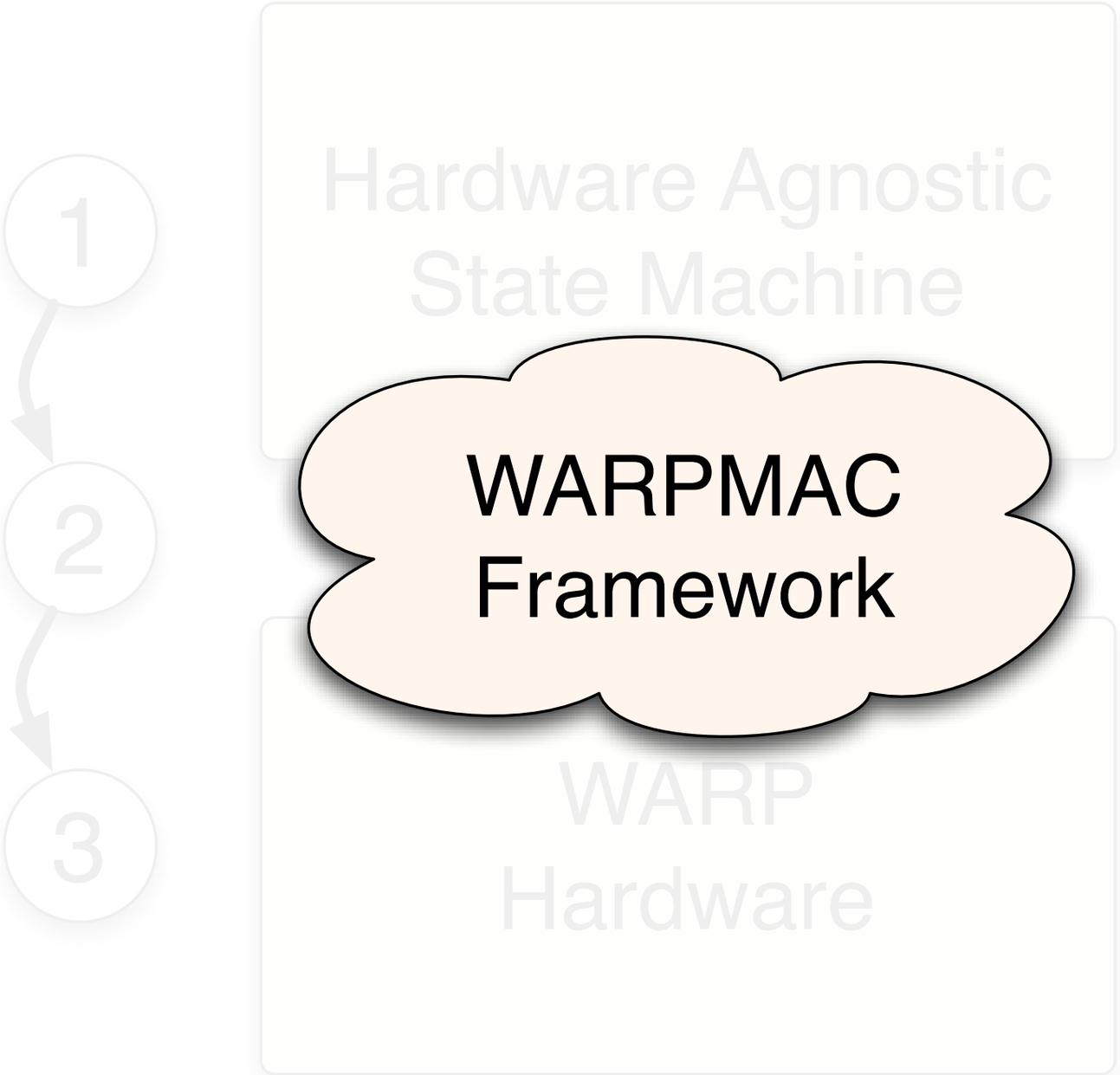
- Program high-level MAC behavior independent of hardware
- Use the WARPMAC framework to stitch the MAC to hardware

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



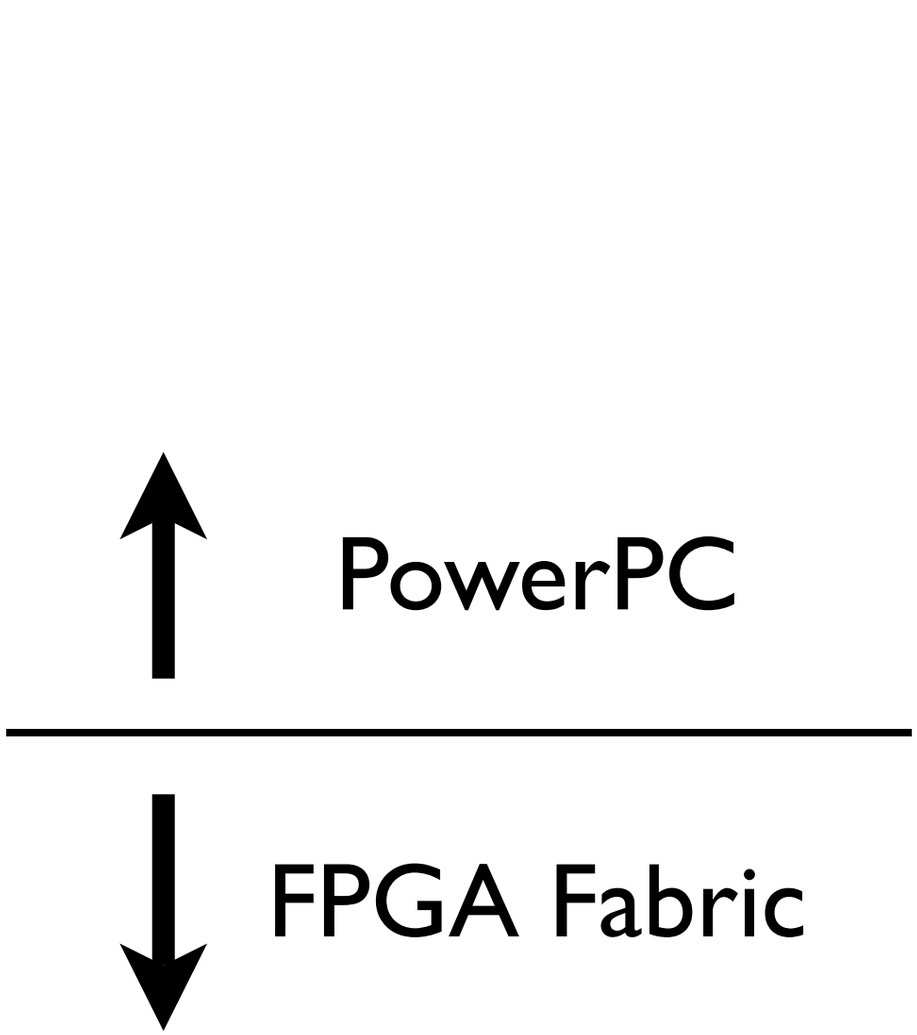
- "Driver" analogy is not entirely accurate
- No way to "lock" the framework and have it support all possible future MAC layers

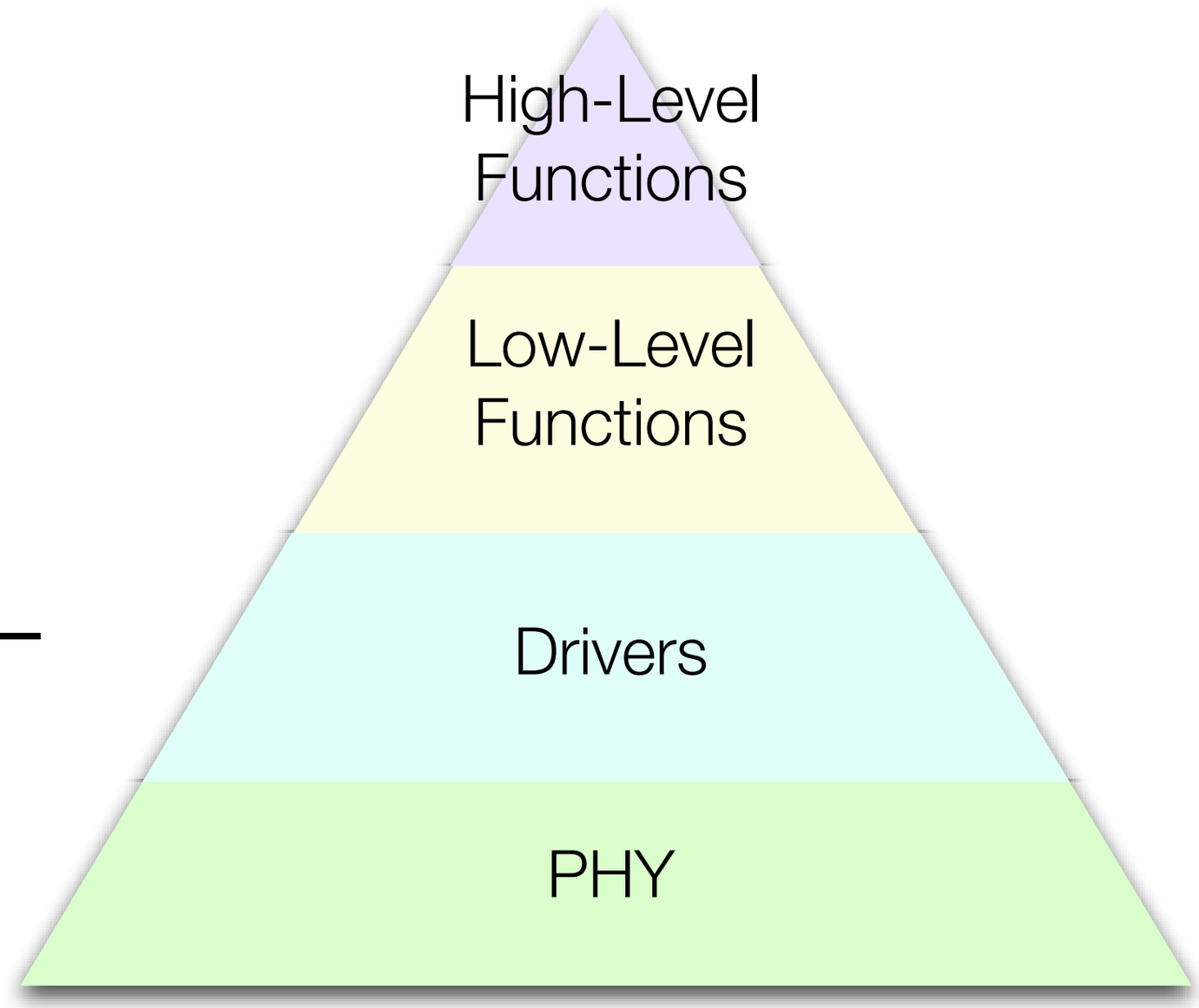
Solution: *WARPMAC must grow with new algorithms*

WARPMAC

Framework

WARPMAC

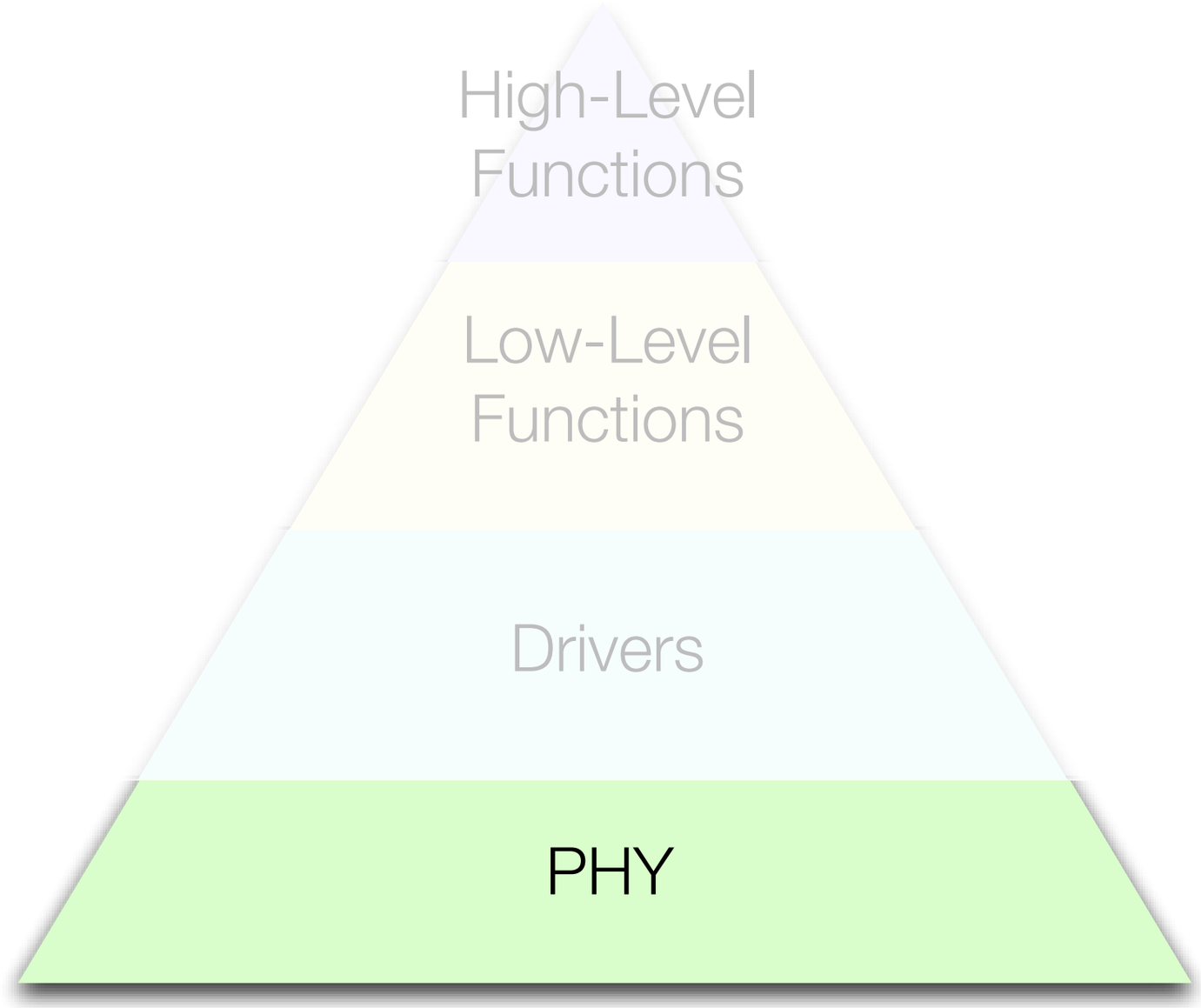

PowerPC
FPGA Fabric


High-Level Functions
Low-Level Functions
Drivers
PHY

WARPMAC

Current Offering:

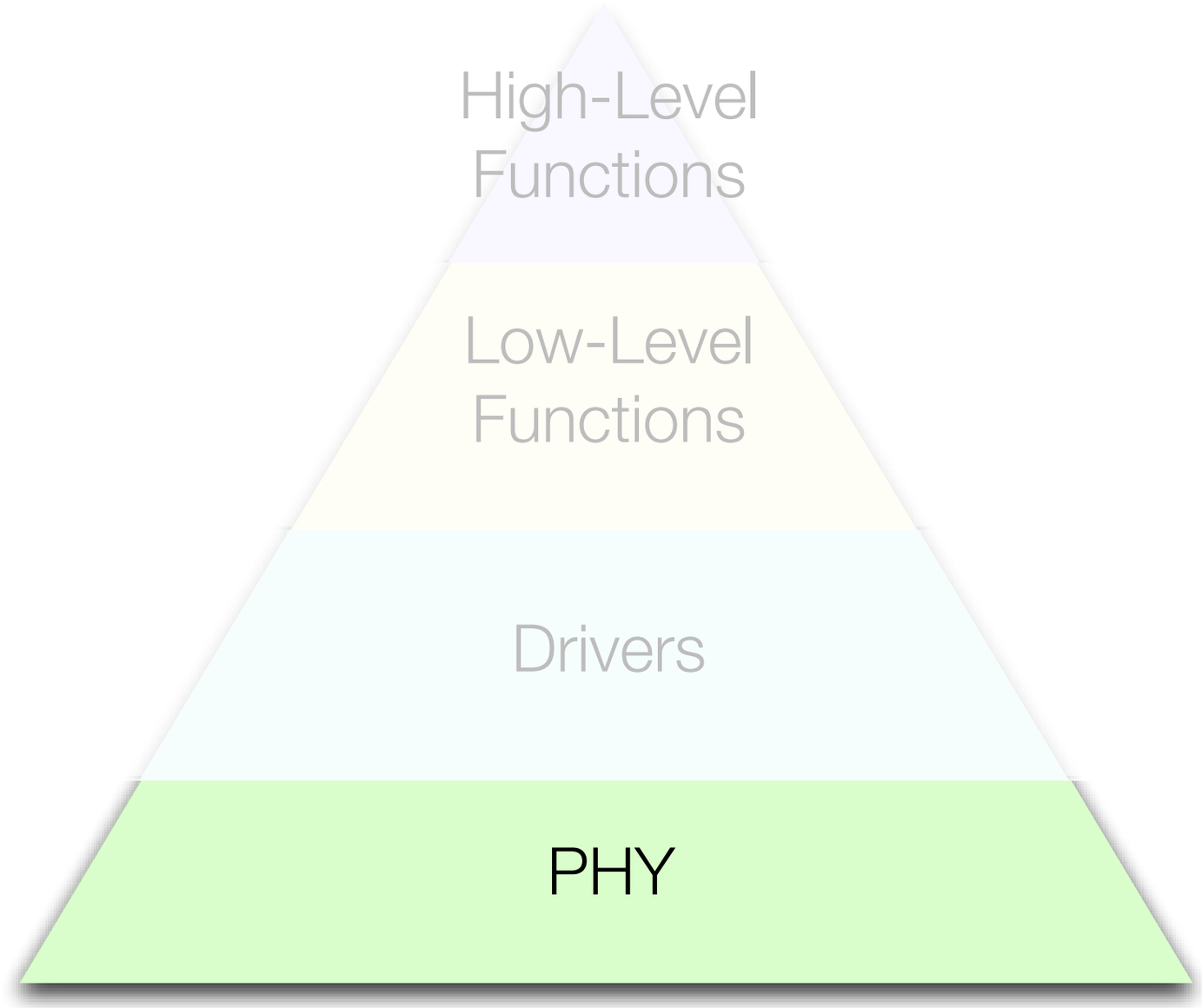
- Custom SISO & MIMO OFDM Transceivers
- Flexible data rate starting at 15Mbps
- Hardware CRC
- Hardware CSMA



WARPMAC

In General:

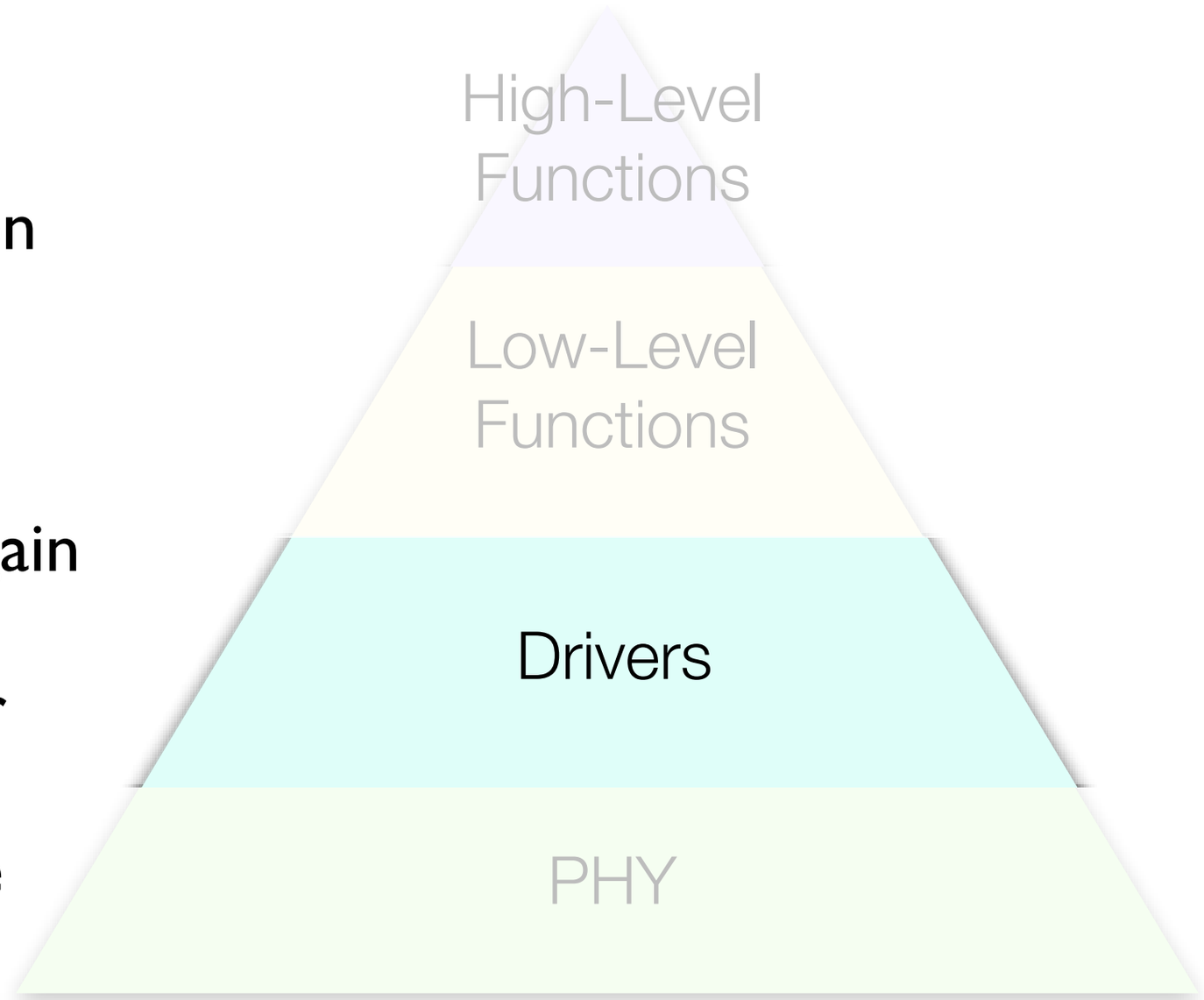
- SISO/MIMO, wide/ narrow band are all possible



WARPMAC

PHY Driver:

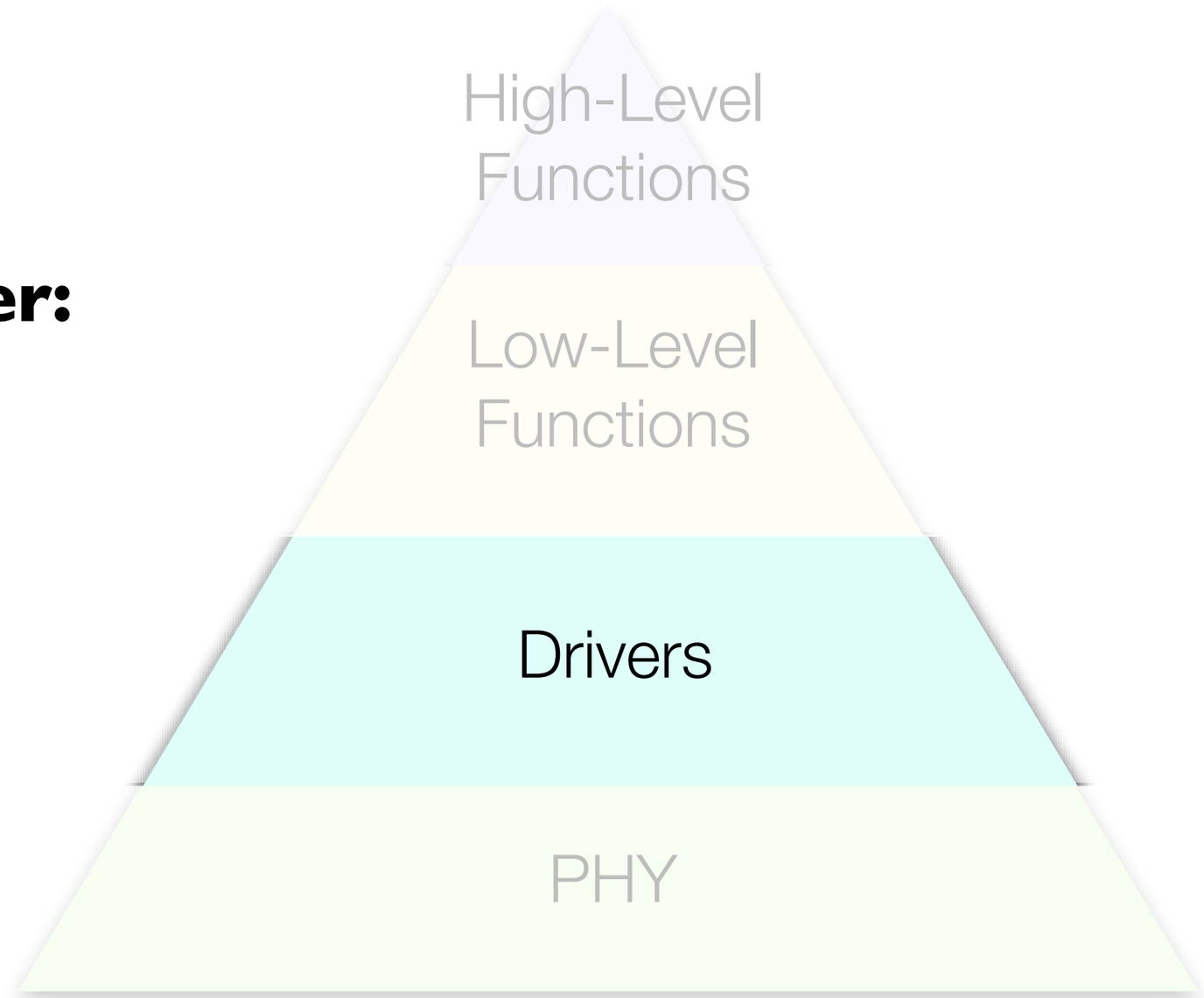
- Configure constellation size
- Thesholds in packet detection, automatic gain control, cross-correlation in receiver
- “Start” and “Stop” the PHY



WARPMAC

Radio Controller Driver:

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



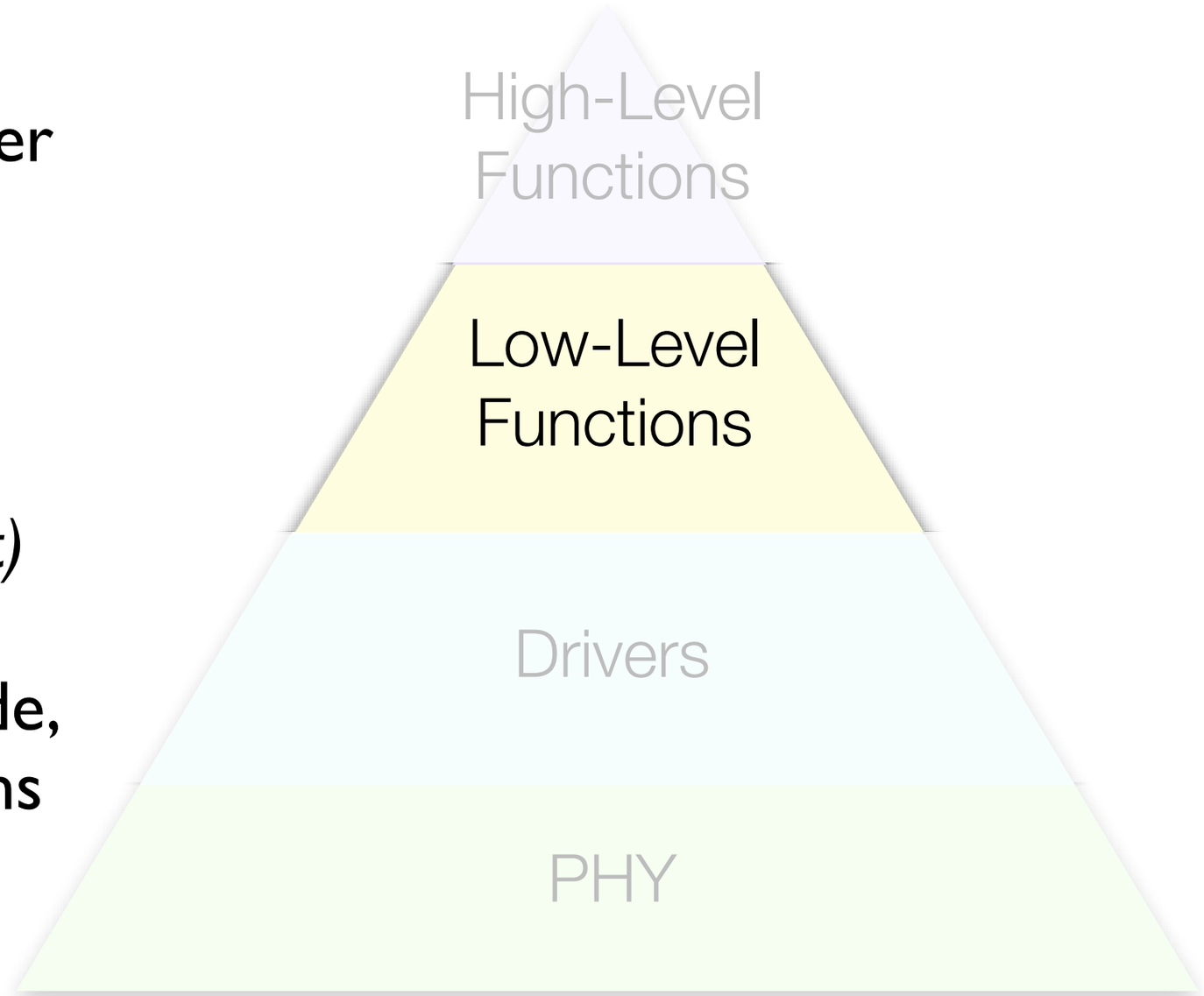
WARPMAC

- Wraps driver calls for another layer of abstraction

- For example:

  warpmac_sendOfdm(myPacket)

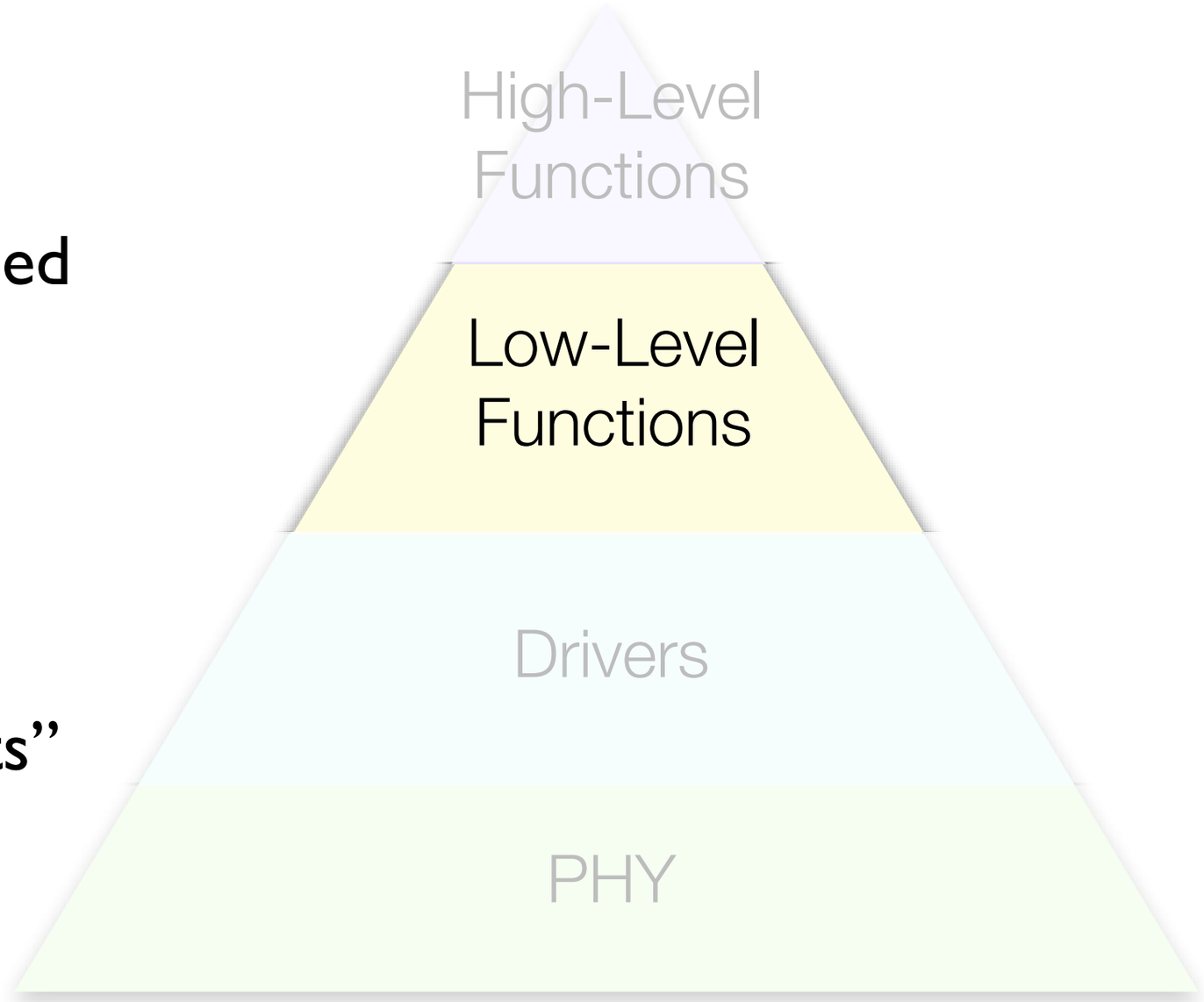
  puts radio into transmit mode, loads payload into PHY, begins transmit



WARPMAC

Interrupt Handling:

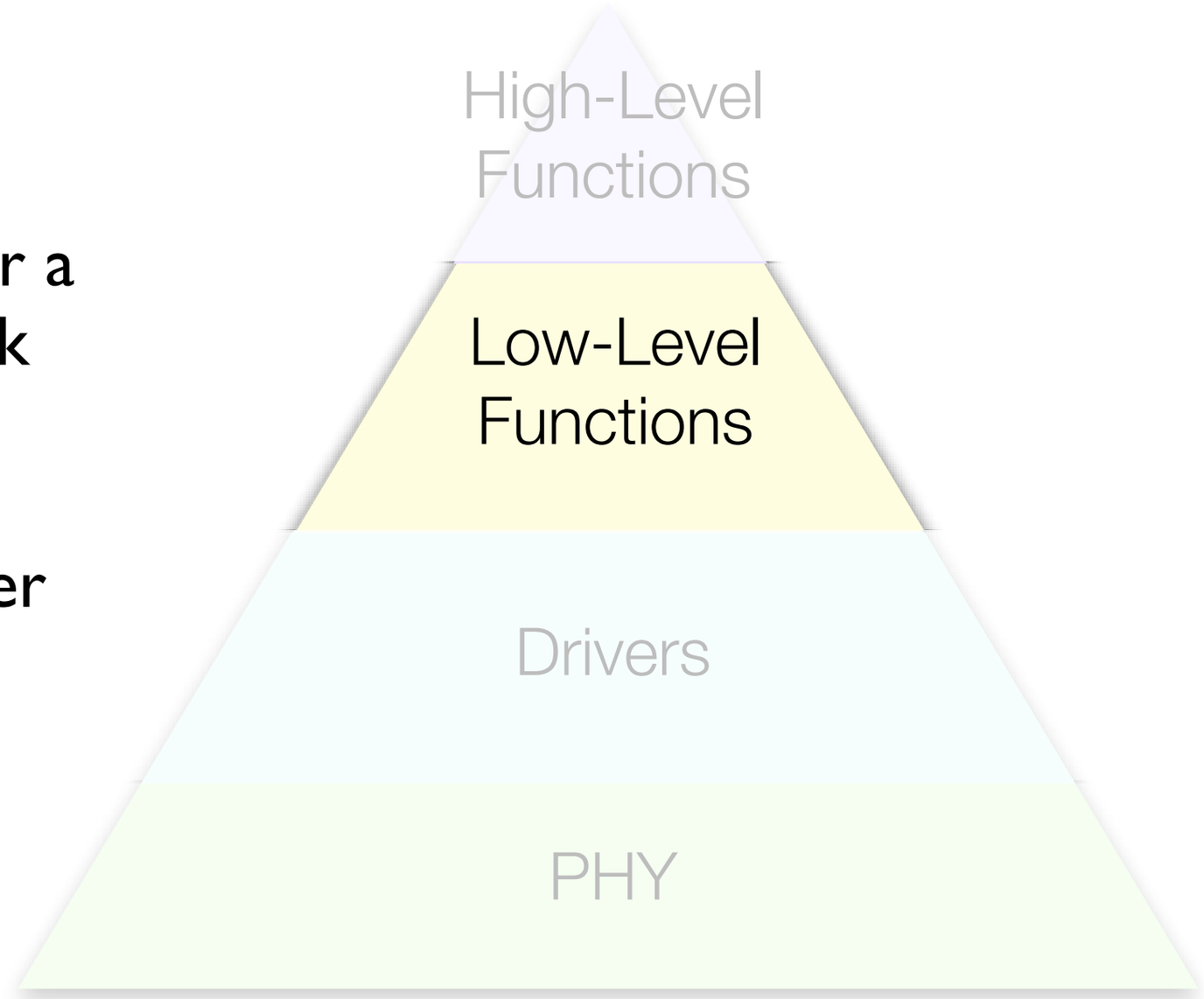
- Register functions to be called upon:
  - Reception of "Good" Packets
  - Reception of "Bad Packets"
  - Expiration of a timer



WARPMAC

Timer Control

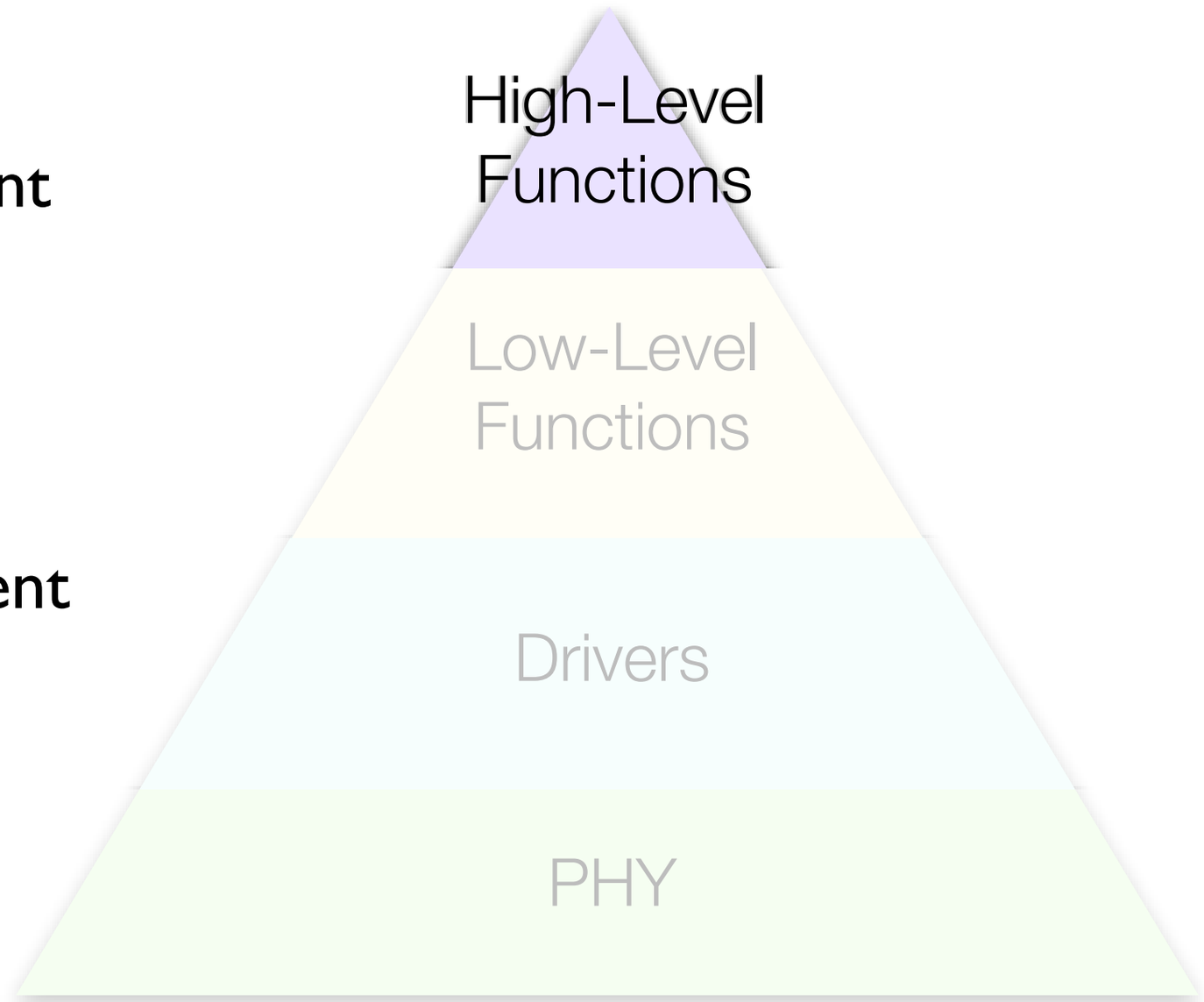
- Start a count down for a certain number a clock cycles
- User-registered handler will be called upon expiration



WARPMAC

- All the functions necessary to implement the ALOHA protocol

- For example, timer control function now abstracted to implement binary exponential backoff

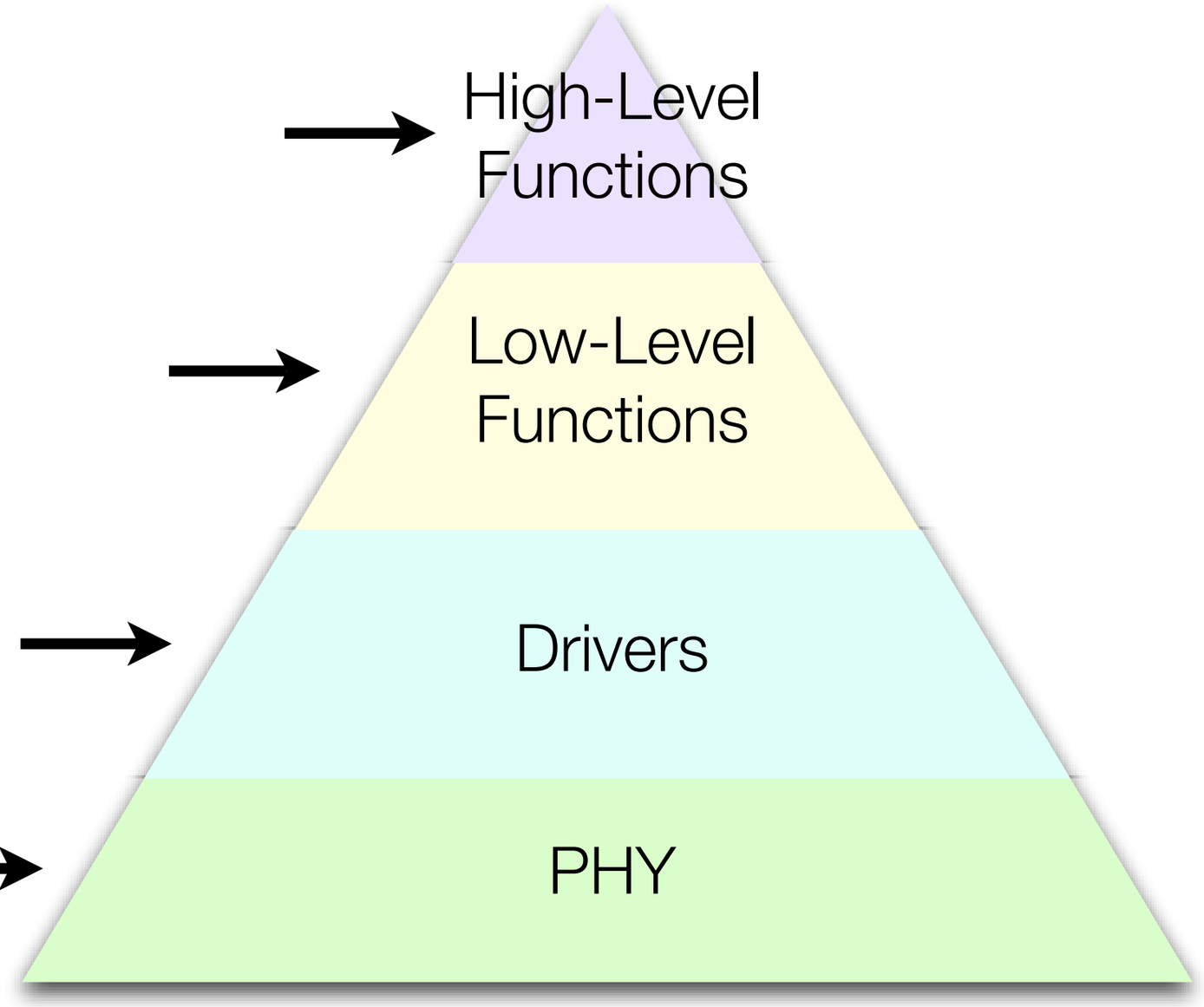


WARPMAC

Implementing Novel MACs:

- "Level" of WARPMAC to use is MAC dependent
- New PHYs, MACs, and lower-level functions will be added to the WARP repository:

  http://warp.rice.edu/trac



An example: ALOHA

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

An example: ALOHA

- Simplest 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

- Simplest 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

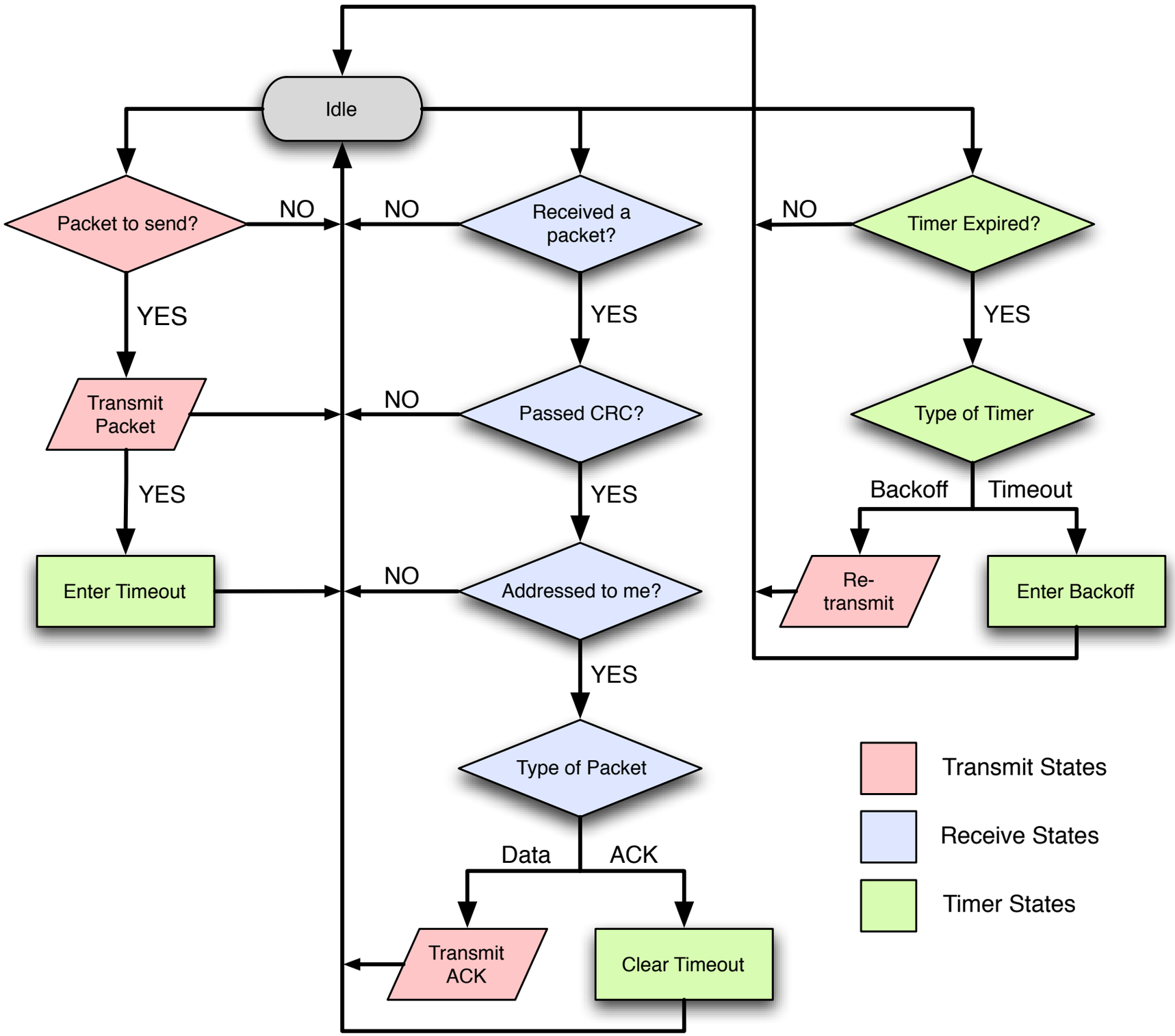
- Simplest MAC
- Serves as a foundation for a large class of other random access protocols
- The algorithm is simple:

Packet to send? Just send it

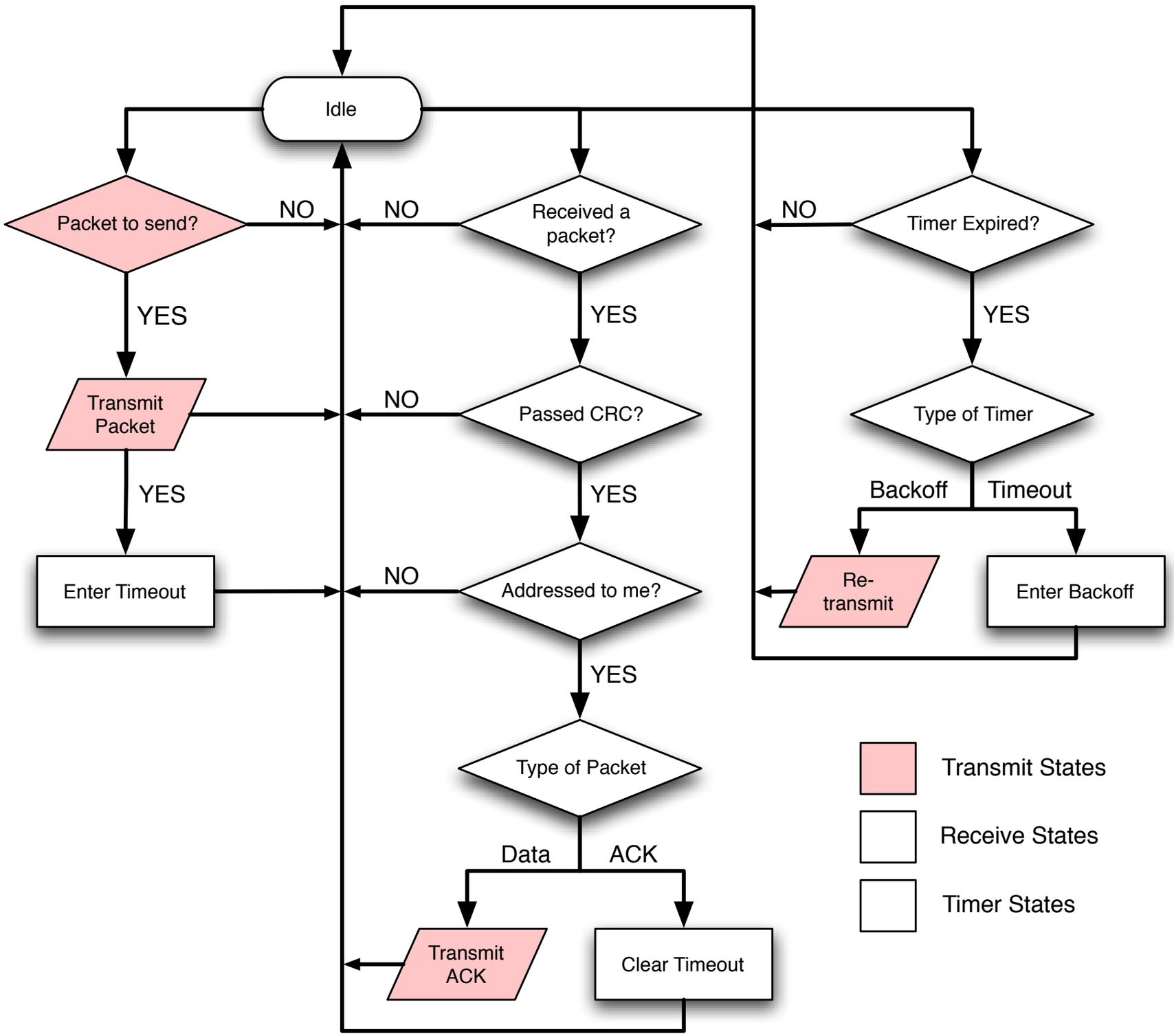
Received a packet? Send an ACK

Received no ACK? Backoff and resend

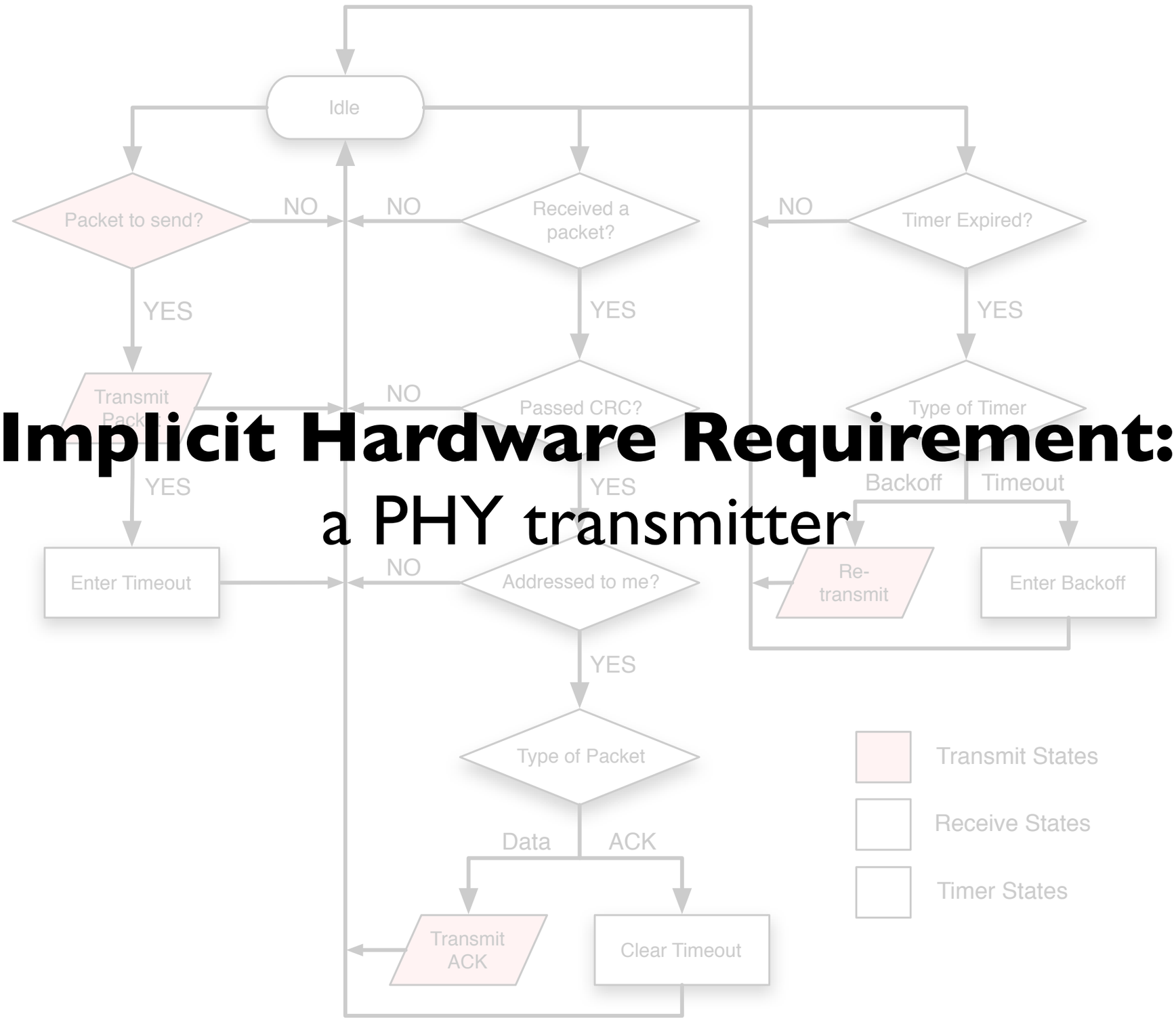
An example: ALOHA



An example: ALOHA

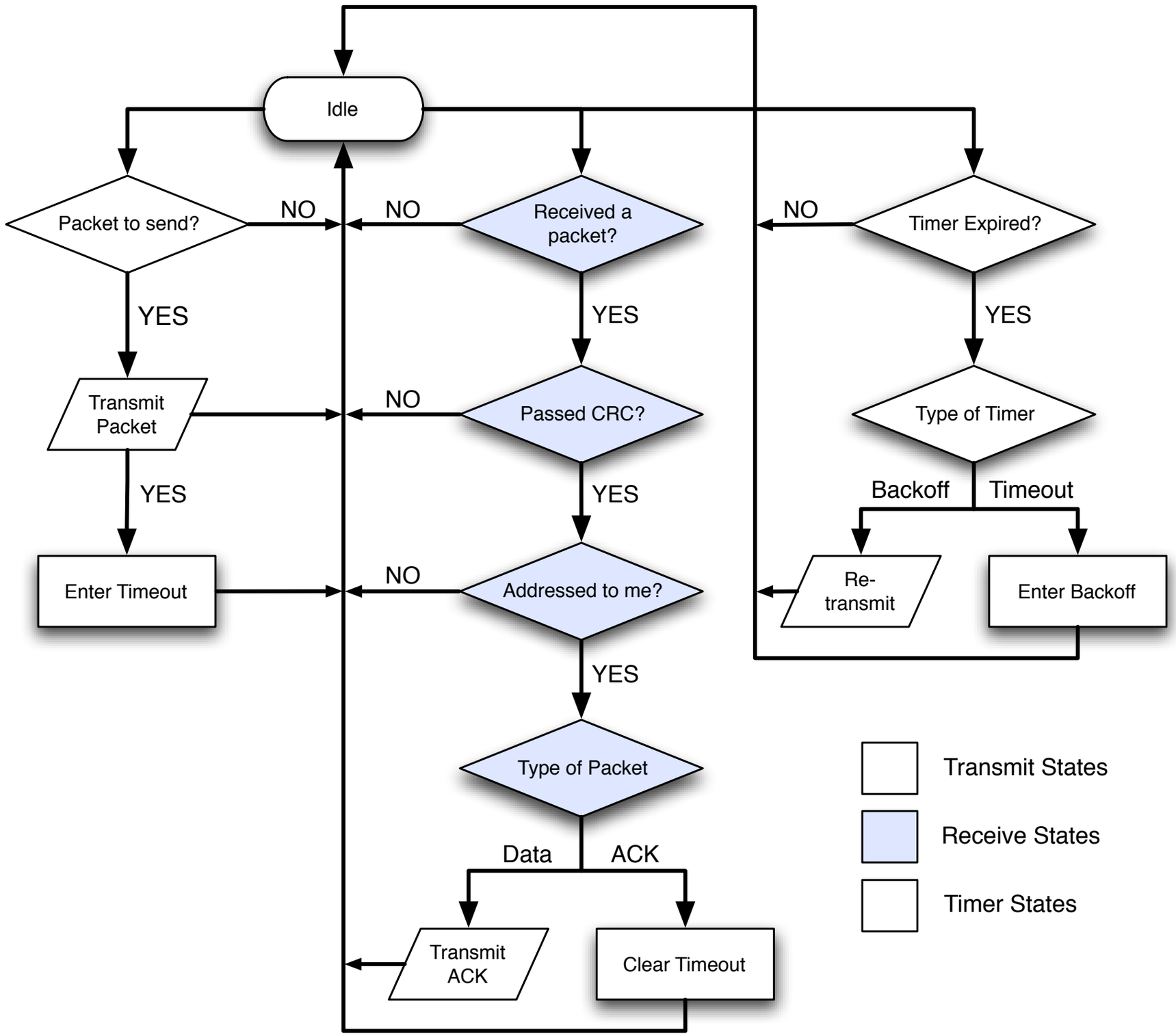


An example: ALOHA

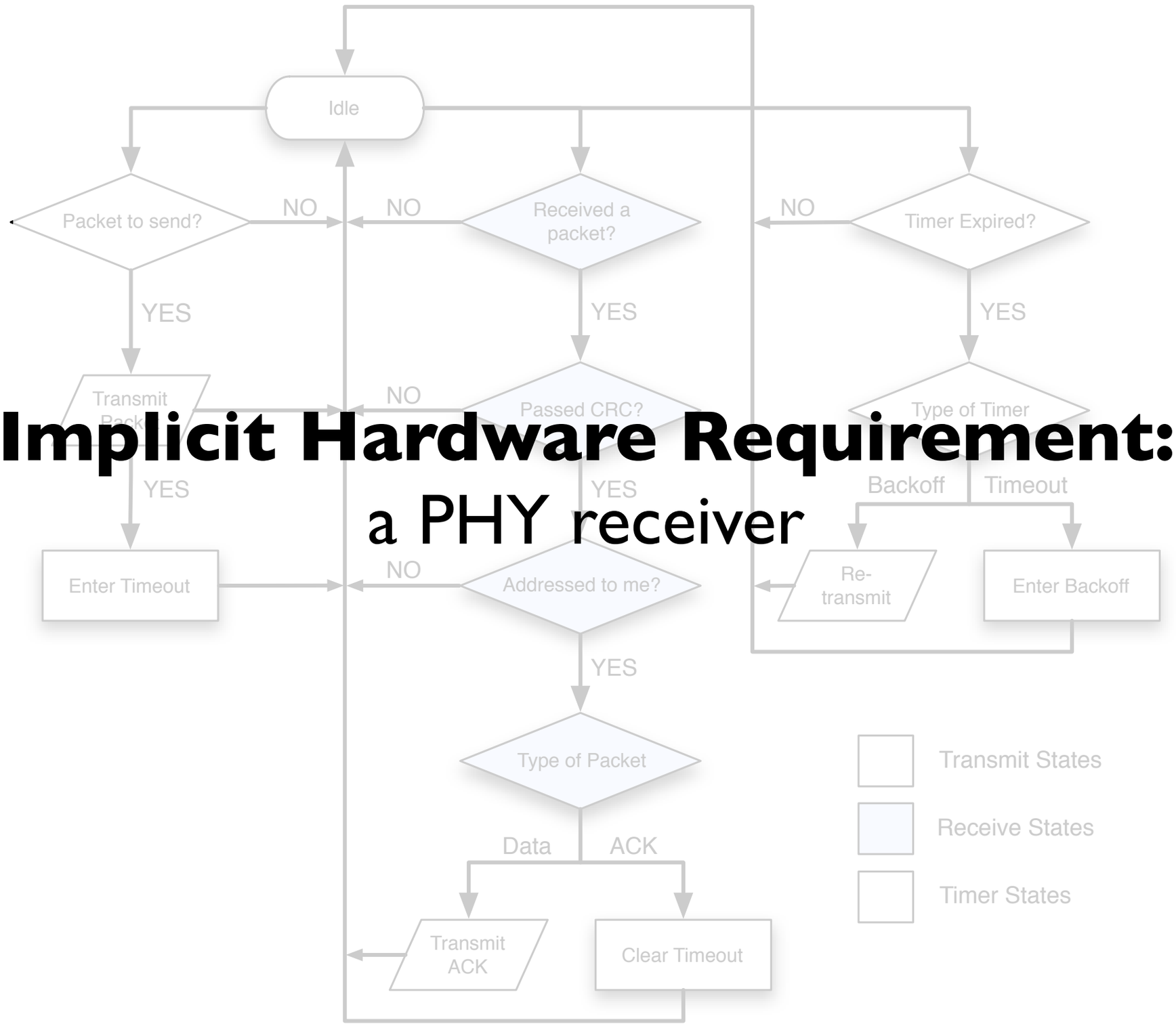


Implicit Hardware Requirement:
a PHY transmitter

An example: ALOHA

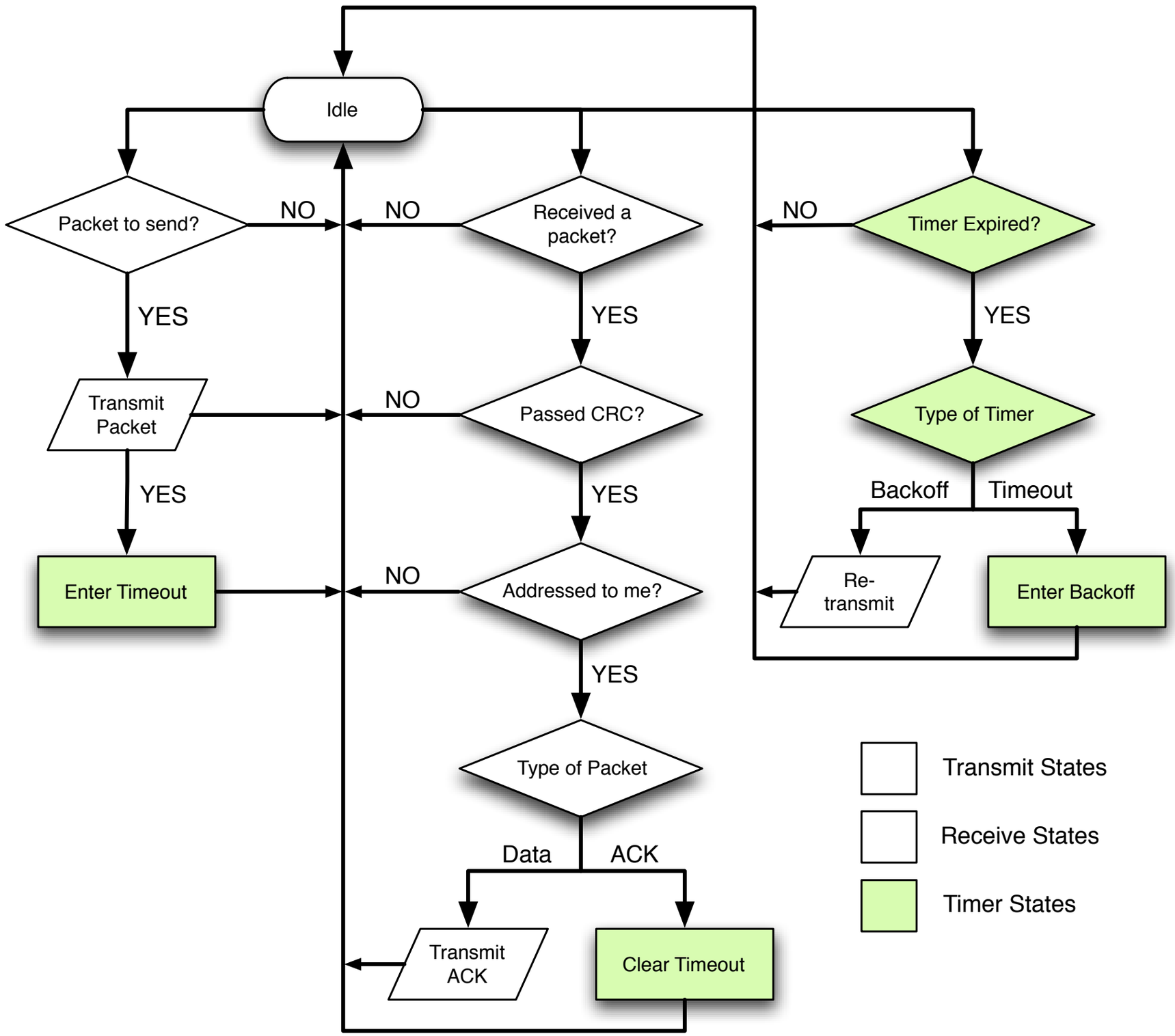


An example: ALOHA

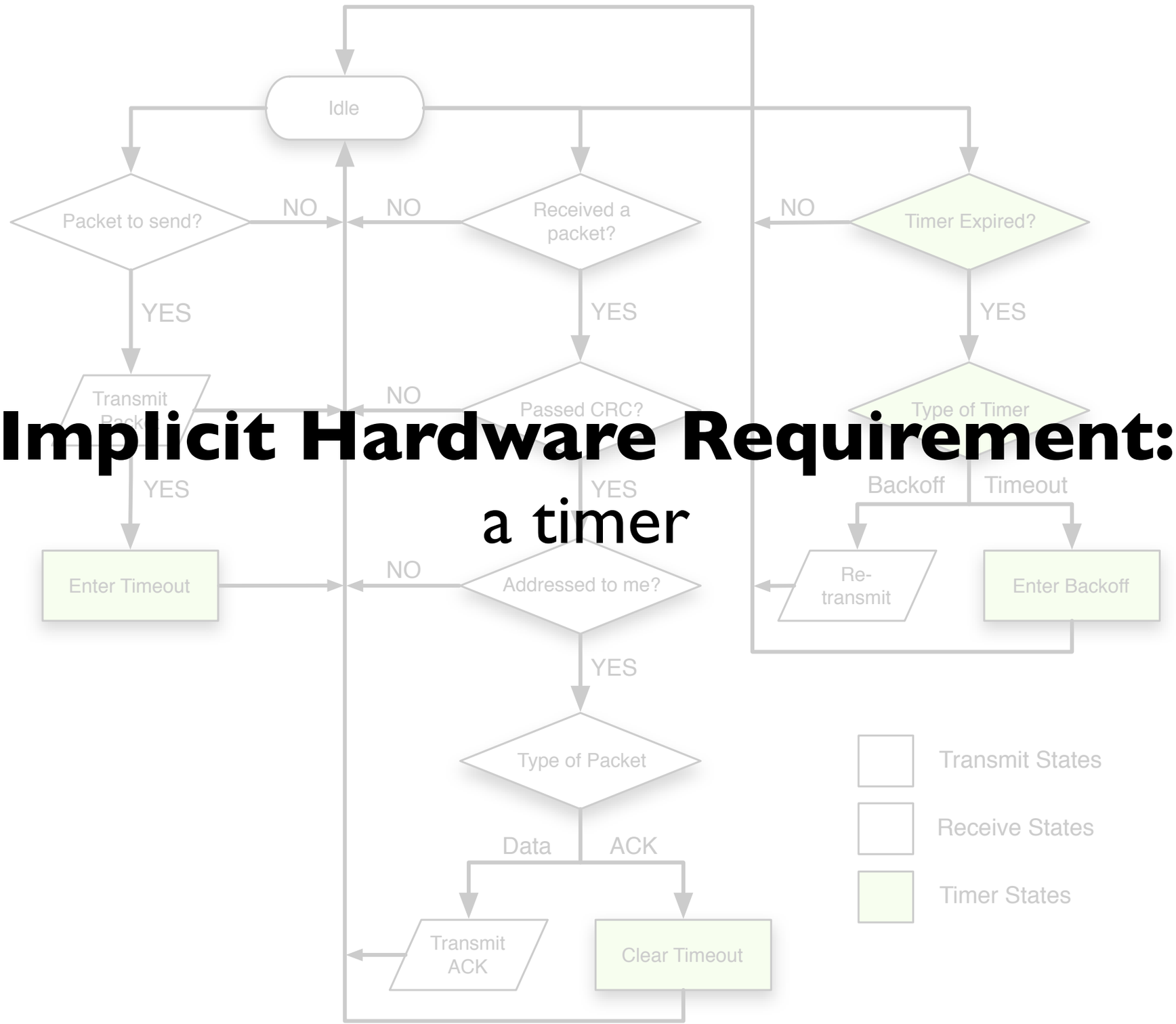


Implicit Hardware Requirement:
a PHY receiver

An example: ALOHA



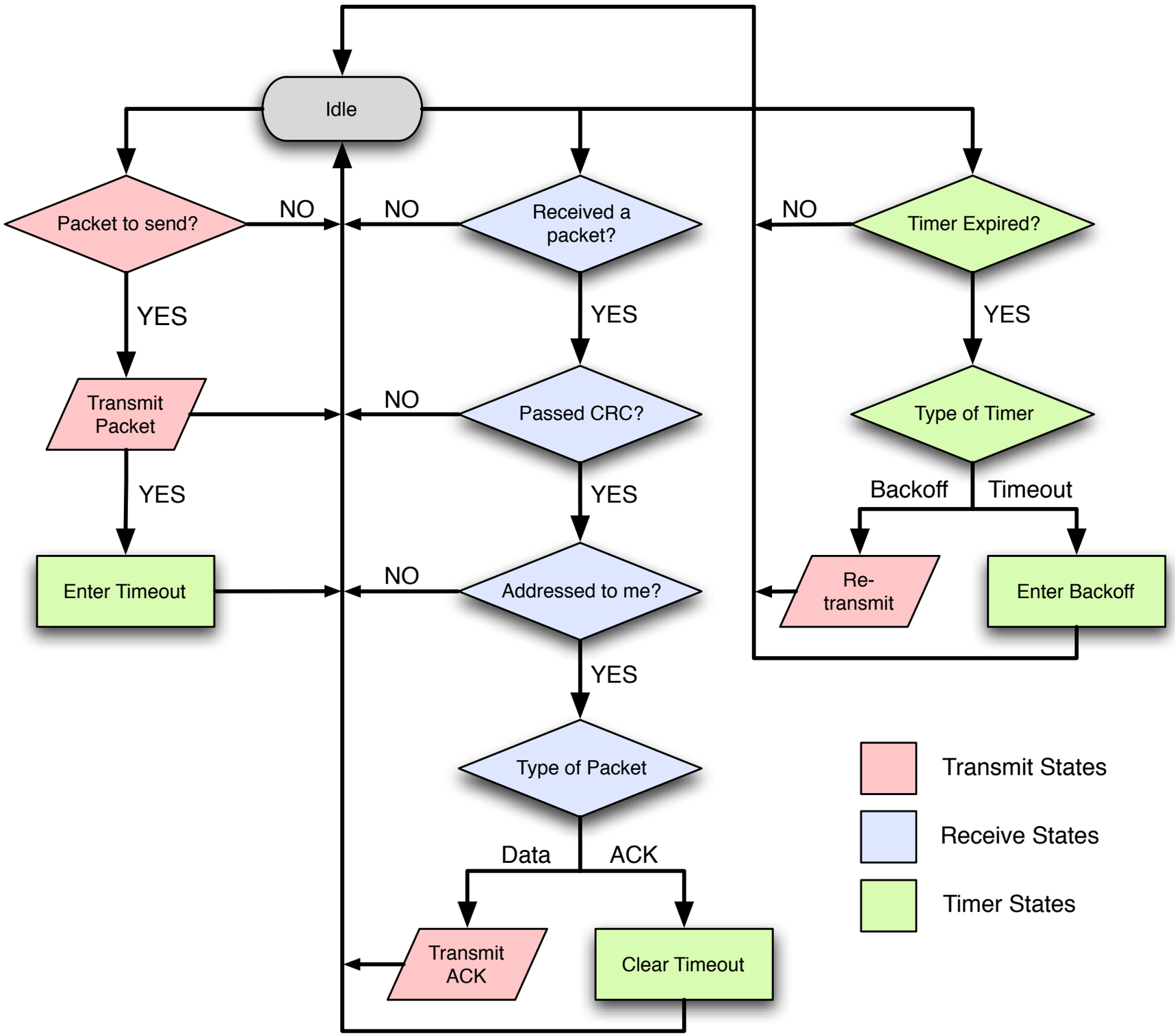
An example: ALOHA



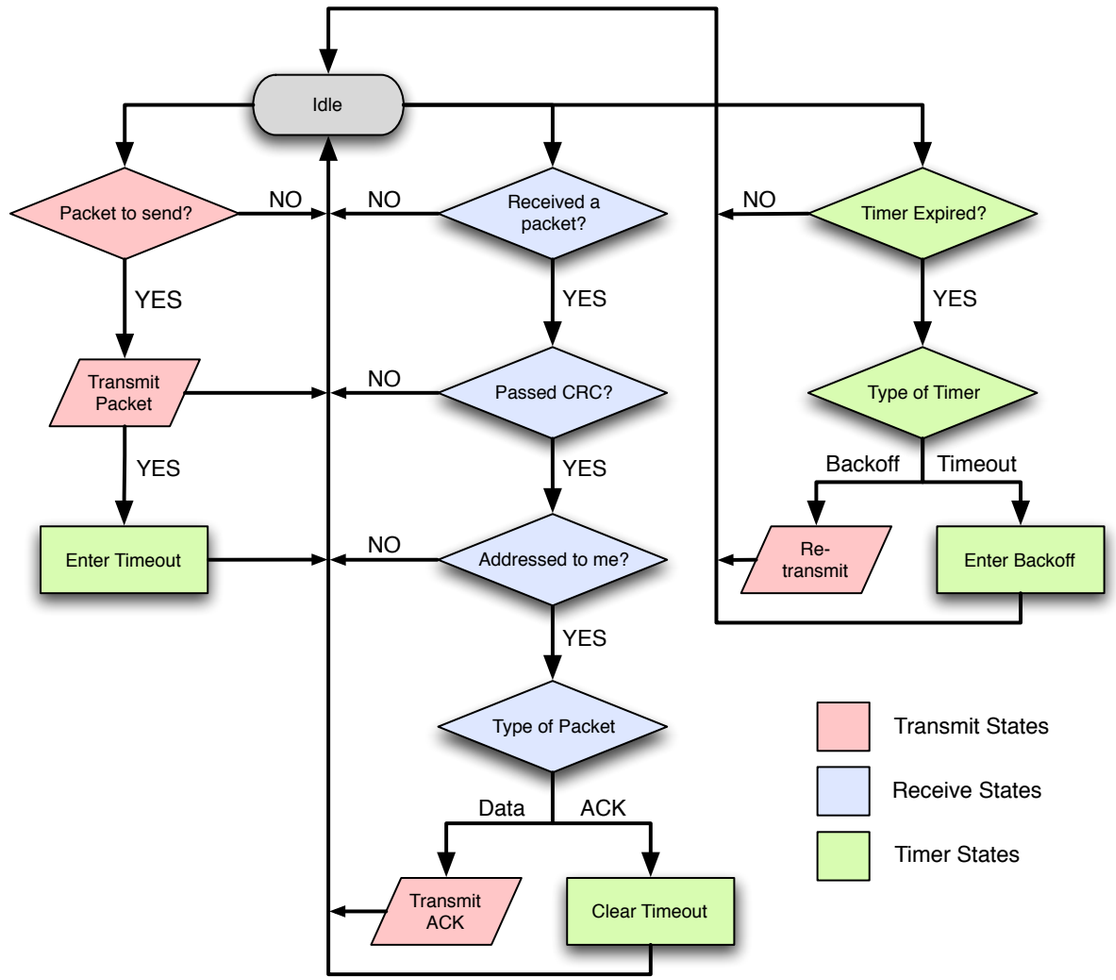
Implicit Hardware Requirement:

a timer

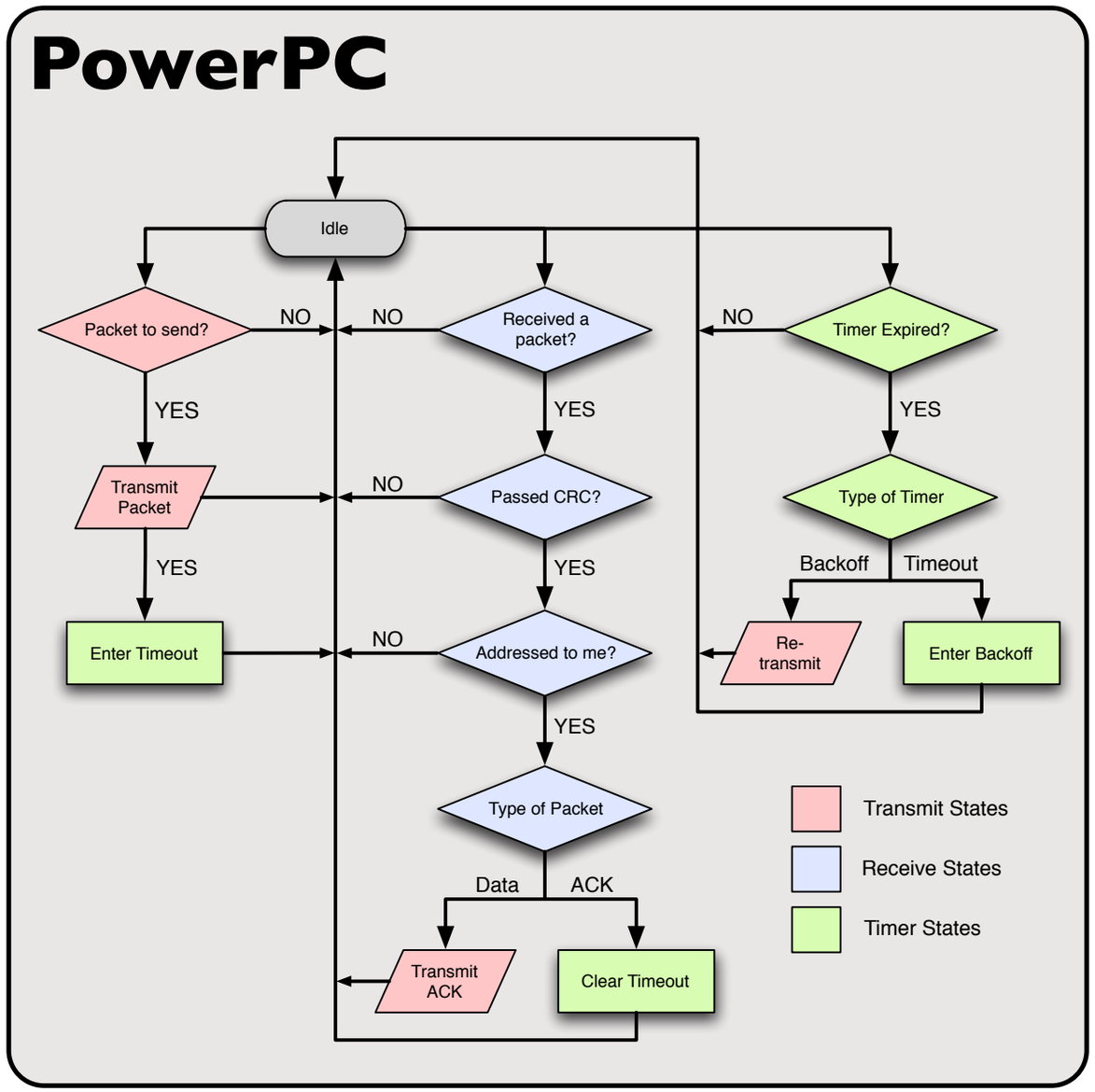
An example: ALOHA



Hardware Requirements



Hardware Requirements

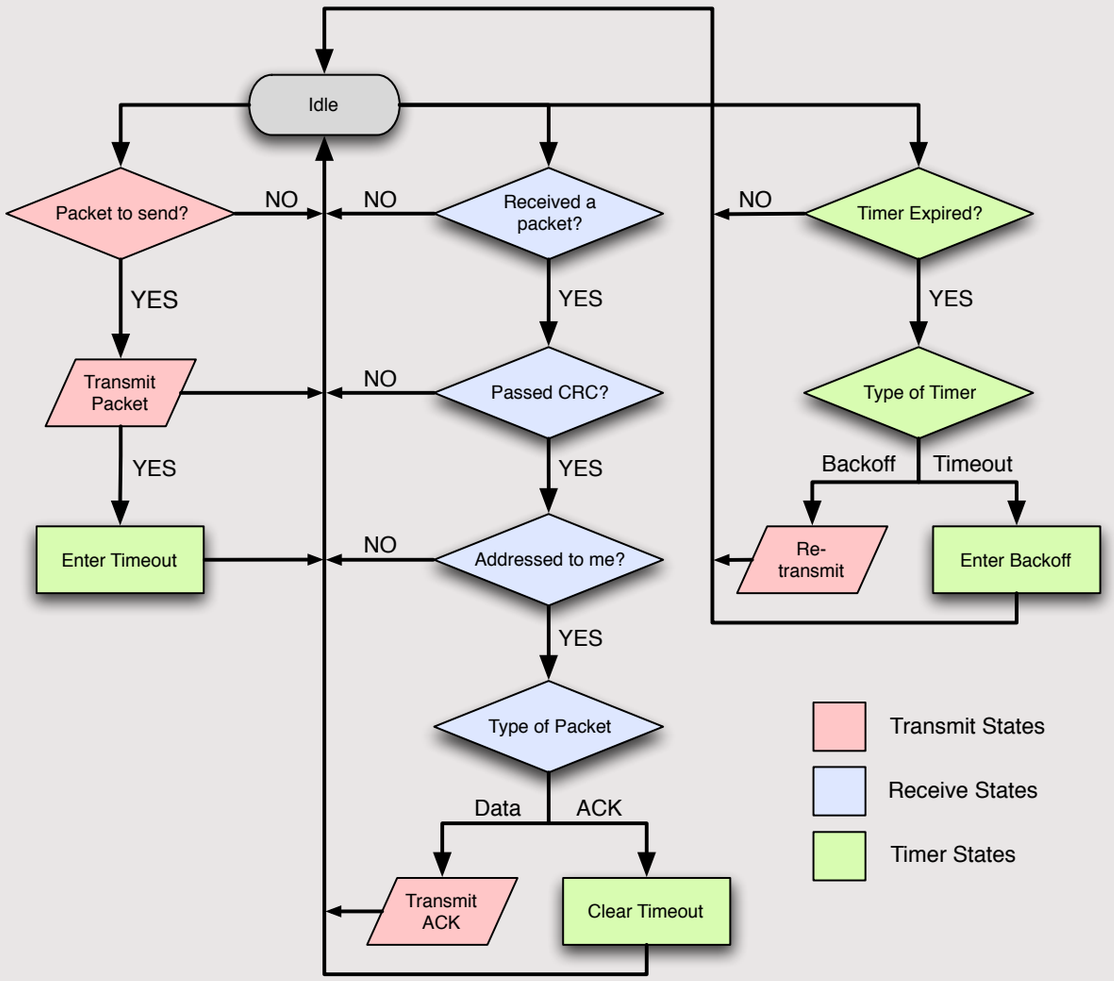


Hardware Requirements

PHY Transmitter

PHY Receiver

PowerPC



Hardware Timer

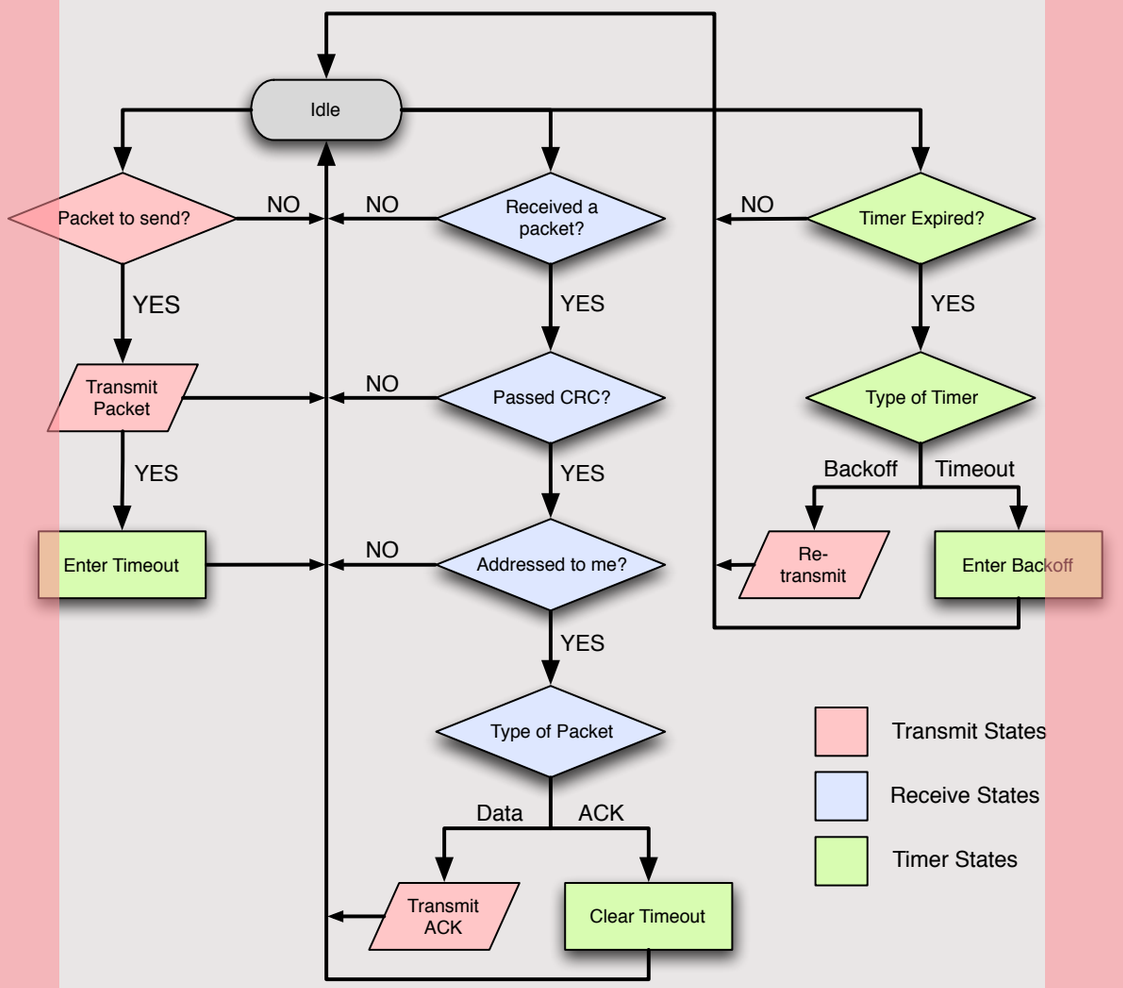
Ethernet

Hardware Requirements

PHY Transmitter

PHY Receiver

PowerPC



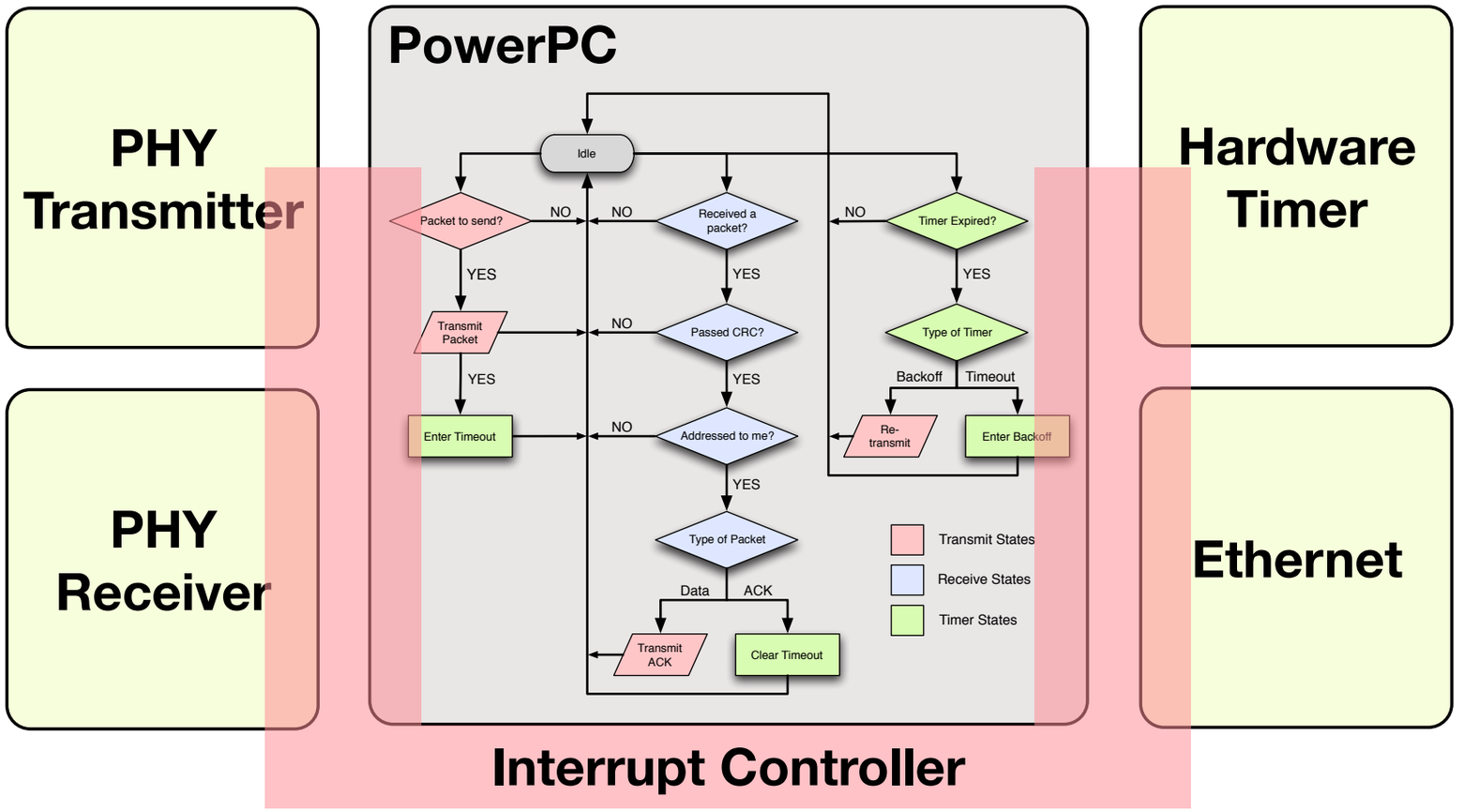
Hardware Timer

Ethernet

Interrupt Controller

Hardware Platform

Hardware Platform



Hardware Platform

Radio Controller

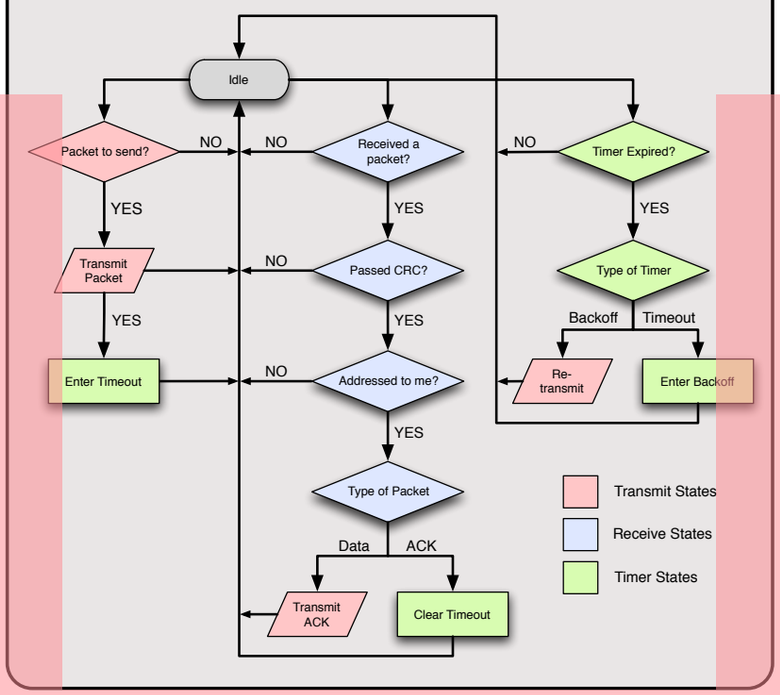
Packet Detection

Automatic Gain Control

PHY Transmitter

PHY Receiver

PowerPC



Hardware Timer

Ethernet

Interrupt Controller

Hardware Platform

Radio Controller

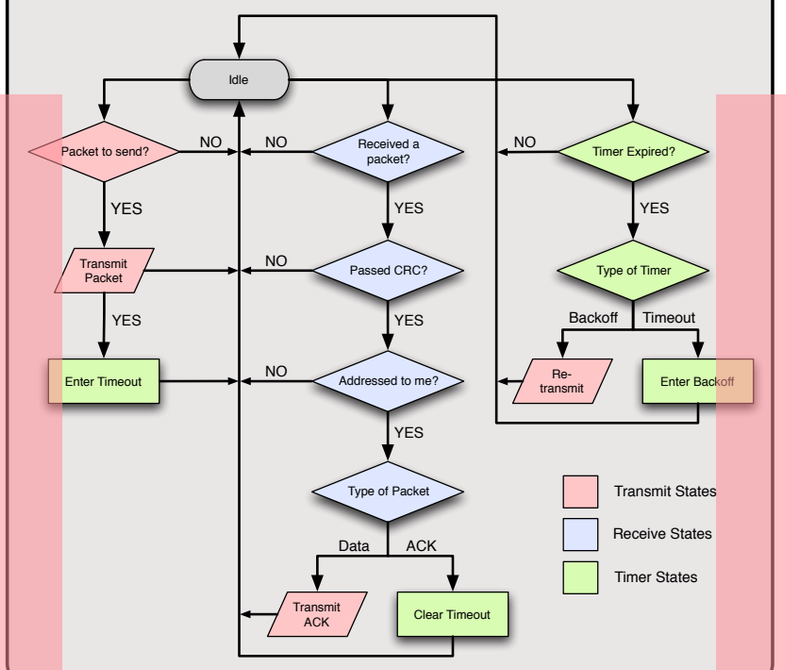
Packet Detection

Automatic Gain Control

PHY Transmitter

PHY Receiver

PowerPC



Interrupt Controller

Hardware Timer

Ethernet

Push-buttons

LEDs

RS232 Serial

Hardware Platform

Radio Controller

Packet Detection

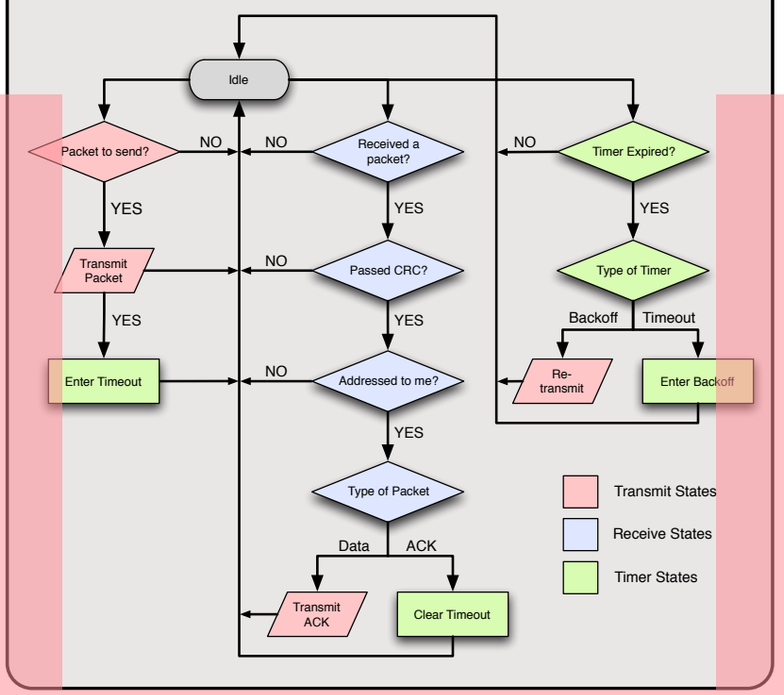
Automatic Gain Control

PHY Transmitter

PHY Receiver

PowerP

Control Code



Hardware Timer

Ethernet

Interrupt Controller

Push-buttons

LEDs

RS232 Serial

Hardware Platform

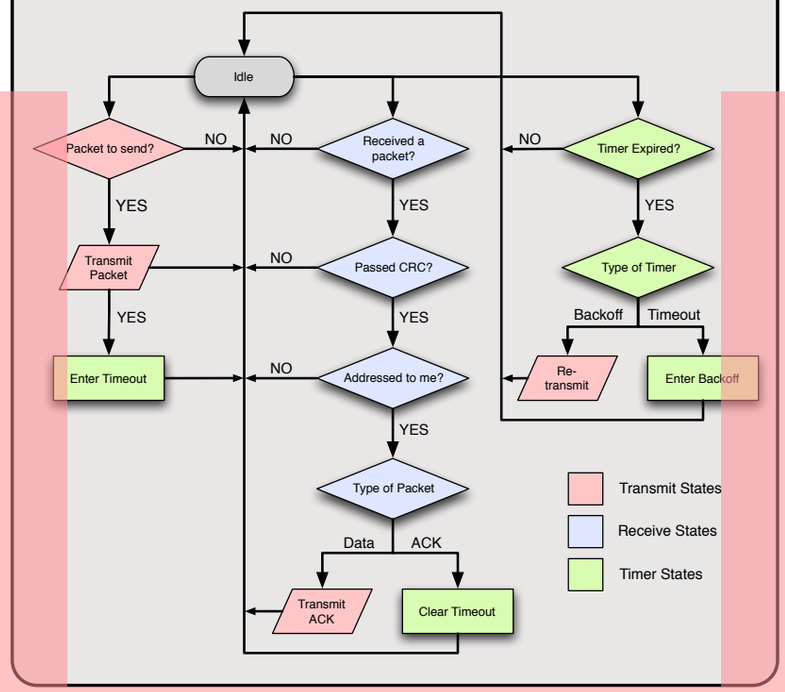
Radio Controller

PowerP

Control Code

Push-buttons

PHY Transmitter



Hardware Timer

Packet Detection

PHY Receiver

Ethernet

LEDs

Automatic Gain Control

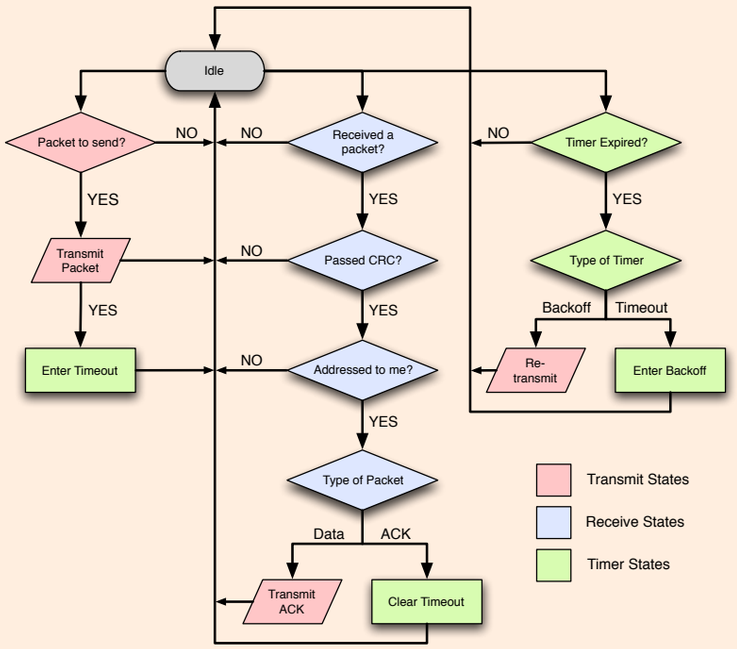
Interrupt Controller

RS232 Serial

Processor Busses (OPB/PLB)

One extreme: Hide the Hard Stuff

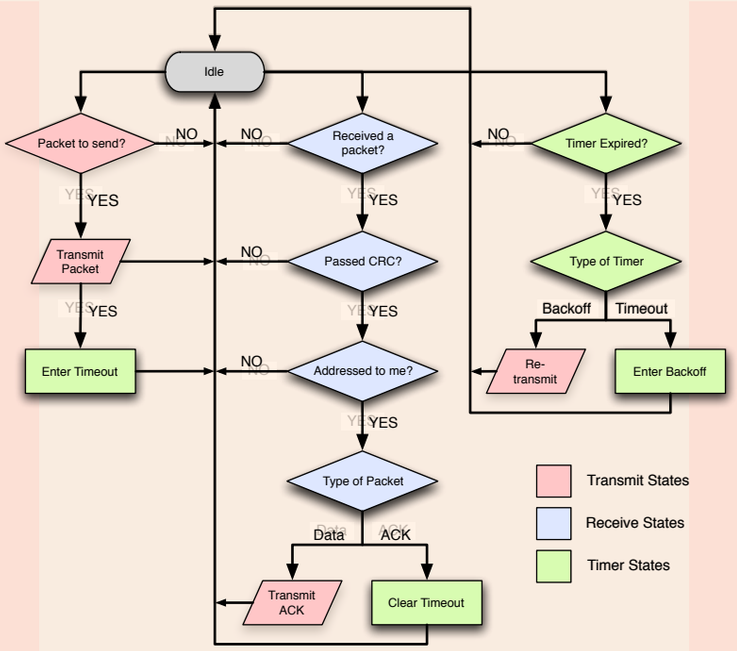
WARPMAC



Somewhere in between

WARPMAC

Control Code



Radio Controller

PowerP

PHY Transmitter

Packet Detection

PHY Receiver

Automatic Gain Control

Hardware Timer

Ethernet

Interrupt Controller

Push-buttons

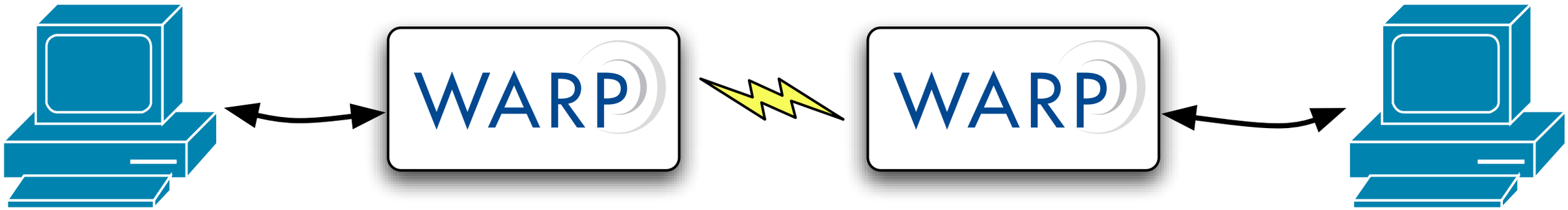
LEDs

RS232 Serial

Processor Busses (OPB/PLB)

Detailed Example

CSMA



Experimental Wireless

Ethernet

```
    warpmac_setSlotTime(9);
    warpmac_enableSequencing();


    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);


    warpmac_setTimerHandler(timerExpire);
    warpmac_setChannel(6);
    warpmac_enableSisoMode();
    warpmac_enableCSMA();

    while(1){
            if(txBuffer.isNew==0){
                    warpmac_pollEthernet(ethernet_callback);
            }
    }
    pthread_exit (NULL);
}

int main(){
    xilkernel_main();
}
```

- Launches the Xilinx kernel
- Kernel will launch the thread specified in Software Platform Settings in XPS
- For this project, that thread is “myMac_main”

```
/*MYMAC_MAIN- function is instantiating the MAC framework and looping in an idle state*/
void* myMac_main(){

    //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] = {0x16,0x24,0x63,0x53,0xe2,0xc2+myID};
    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<16;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_setMaxCW(4);
    warpmac_setTimeout(400);
    warpmac_setSlotTime(9);
    warpmac_enableSequencing();


    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);


    warpmac_setTimerHandler(timerExpire);
    warpmac_setChannel(6);
    warpmac_enableSisoMode();
    warpmac_enableCSMA();

    while(1){
            if(txBuffer.isNew==0){
                    warpmac_pollEthernet(ethernet_callback);
            }
    }
    pthread_exit (NULL);
}
```

```c
/*MYMAC_MAIN- function is instantiating the MAC framework and looping in an idle state*/
void* myMac_main(){

    //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] = {0x16,0x24,0x63,0x53,0xe2,0xc2+myID};
    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<16;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_setMaxCW(4);
    warpmac_setTimeout(400);
    warpmac_setSlotTime(9);
    warpmac_enableSequencing();


    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);


    warpmac_setTimerHandler(timerExpire);
    warpmac_setChannel(6);
    warpmac_enableSisoMode();
    warpmac_enableCSMA();

    while(1){
            if(txBuffer.isNew==0){
                    warpmac_pollEthernet(ethernet_callback);
            }
    }
    pthread_exit (NULL);
}
```

- 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

```
/*MYMAC_MAIN- function is instantiating the MAC framework and looping in an idle state*/
void* myMac_main(){

    //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] = {0x16,0x24,0x63,0x53,0xe2,0xc2+myID};
    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<16;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_setMaxCW(4);
    warpmac_setTimeout(400);
    warpmac_setSlotTime(9);
    warpmac_enableSequencing();


    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);


    warpmac_setTimerHandler(timerExpire);
    warpmac_setChannel(6);
    warpmac_enableSisoMode();
    warpmac_enableCSMA();

    while(1){
            if(txBuffer.isNew==0){
                    warpmac_pollEthernet(ethernet_callback);
            }
    }
    pthread_exit (NULL);
}
```

- 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

```
/*MYMAC_MAIN- function is instantiating the MAC framework and looping in an idle state*/
void* myMac_main(){

    //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] = {0x16,0x24,0x63,0x53,0xe2,0xc2+myID};
    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<16;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_setMaxCW(4);
    warpmac_setTimeout(400);
    warpmac_setSlotTime(9);
    warpmac_enableSequencing();


    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);


    warpmac_setTimerHandler(timerExpire);
    warpmac_setChannel(6);
    warpmac_enableSisoMode();
    warpmac_enableCSMA();

    while(1){
            if(txBuffer.isNew==0){
                    warpmac_pollEthernet(ethernet_callback);
            }
    }
    pthread_exit (NULL);
}
```

- Initializes the framework
  - Initializes PHY, radio, AGC, packet detection, interrupts, etc.
- Sets specific parameters
  - 4 resends
  - Maximum contention window of 4 * (Slot-time)
  - 9 usec Slot-time
  - 400 usec timeout
  - Enables sequencing to reduce packet duplicates

```
/*MYMAC_MAIN- function is instantiating the MAC framework and looping in an idle state*/
void* myMac_main(){

    //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] = {0x16,0x24,0x63,0x53,0xe2,0xc2+myID};
    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<16;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_setMaxCW(4);
    warpmac_setTimeout(400);
    warpmac_setSlotTime(9);
    warpmac_enableSequencing();


    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);


    warpmac_setTimerHandler(timerExpire);
    warpmac_setChannel(6);
    warpmac_enableSisoMode();
    warpmac_enableCSMA();

    while(1){
            if(txBuffer.isNew==0){
                    warpmac_pollEthernet(ethernet_callback);
            }
    }
    pthread_exit (NULL);
}
```

- Sets handlers to be called on certain events:
  - Packets that pass checksum
  - Packets that fail checksum
  - Timer expiration
- Sets specific parameters
  - Sets the channel of operation
  - Enables SISO operation of the MIMO core (MIMO is still in development)
  - Enables hardware carrier-sensing

```
/*MYMAC_MAIN- function is instantiating the MAC framework and looping in an idle state*/
void* myMac_main(){

    //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] = {0x16,0x24,0x63,0x53,0xe2,0xc2+myID};
    memcpy(myAddr,tmpAddr,6);

    //Fill an arbitrary routing table so that nodes know each others' addresses
    unsigned char i;
    for(i=0;i<16;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_setMaxCW(4);
    warpmac_setTimeout(400);
    warpmac_setSlotTime(9);
    warpmac_enableSequencing();


    warpmac_setGoodPacketHandler(receiveGoodPacket);
    warpmac_setBadPacketHandler(receiveBadPacket);


    warpmac_setTimerHandler(timerExpire);
    warpmac_setChannel(6);
    warpmac_enableSisoMode();
    warpmac_enableCSMA();

    while(1){
            if(txBuffer.isNew==0){
                    warpmac_pollEthernet(ethernet_callback);
            }
    }
    pthread_exit (NULL);
}
```

- Loop forever, polling ethernet when packet has been freed
  - isNew specifies whether or not the packet is still undergoing retransmissions
- At this point, we are in the “idle” state, ready to process a number of cases

Case 1:

Packet received from Ethernet

```
int* ethernet_callback(Xuint8 * frame, Xuint32 length){
    /*This function is called by the ethernet MAC drivers
    when a packet is available to send. This function fills
    the Macframe transmit buffer with the packet and sends
    it over the OFDM link*/

    warpmac_allocatePayload(&txBuffer,length);
    txBuffer.pktRev = ACKMAC;
    txBuffer.length = length;
    txBuffer.pktType = DATAPACKET;
    memcpy(txBuffer.srcAddr,myAddr,6); //Copy MAC address into source field
    memcpy(txBuffer.destAddr,routeTable[(myID+1)%2].addr,6);
    memcpy(txBuffer.data, frame, length);


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

- Allocates memory in the frame where the Ethernet payload should be copied
- Fills in header information
- Type is marked as data
- Destination address is hardcoded to ID to 0 if node is 1, and vice versa

```
int* ethernet_callback(Xuint8 * frame, Xuint32 length){
    /*This function is called by the ethernet MAC drivers
    when a packet is available to send. This function fills
    the Macframe transmit buffer with the packet and sends
    it over the OFDM link*/

    warpmac_allocatePayload(&txBuffer,length);
    txBuffer.pktRev = ACKMAC;
    txBuffer.length = length;
    txBuffer.pktType = DATAPACKET;
    memcpy(txBuffer.srcAddr,myAddr,6); //Copy MAC address into source field
    memcpy(txBuffer.destAddr,routeTable[(myID+1)%2].addr,6);
    memcpy(txBuffer.data, frame, length);


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

- If the medium is idle,
  - send the packet over OFDM
  - enter a timeout
- If the medium is busy,
  - enter a backoff

Case 2:

"Bad" packet received from OFDM

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

int receiveGoodPacket(Macframe* packet) {
    /*This function processes the received packet to see if it
    was addressed to this node, and sends an ACK if it is a
    data packet. Also, it pushes the received packet out to
    ethernet.*/

    warpmac_incrementLEDHigh();
    //IMPORTANT NOTE: Even though this function is passed a Macframe structure,
    //the data field is not yet filled. The user's MAC should process the packet
    //based on the header information, send out an ACK if necessary, and then
    //copy the payload out of the PHY with the warpmac_writePacket command.

    if(warpmac_addressedToMe(packet)){
    Macframe ackPacket;
        switch(packet->pktType){
            case ACKPACKET:
                if(warpmac_inTimeout()){
                    warpmac_clearTimer(TIMEOUT);
                    warpmac_freePayload(&txBuffer);

                    //Delay is necessary to give receiver time to push packet over ethernet...
                    //this will be removed when DMA is implemented
                    usleep(125);
                }
                break;
            case DATAPACKET:
                ackPacket.pktRev = ACKMAC;
                ackPacket.length = 0;
                ackPacket.pktType = ACKPACKET;
                memcpy(ackPacket.srcAddr,myAddr,6);
                memcpy(ackPacket.destAddr,packet->srcAddr,6);

                warpmac_sendOfdm(&ackPacket);
                warpmac_allocatePayload(packet,packet->length);
                warpmac_copyPayload(packet);
                warpmac_sendEthernet(packet);
                warpmac_freePayload(packet);
                break;
        }
    }
    return 0;
}
```

- 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

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

int receiveGoodPacket(Macframe* packet) {
    /*This function processes the received packet to see if it
    was addressed to this node, and sends an ACK if it is a
    data packet. Also, it pushes the received packet out to
    ethernet.*/

    warpmac_incrementLEDHigh();
    //IMPORTANT NOTE: Even though this function is passed a Macframe structure,
    //the data field is not yet filled. The user's MAC should process the packet
    //based on the header information, send out an ACK if necessary, and then
    //copy the payload out of the PHY with the warpmac_writePacket command.

    if(warpmac_addressedToMe(packet)){
    Macframe ackPacket;
        switch(packet->pktType){
            case ACKPACKET:
                if(warpmac_inTimeout()){
                    warpmac_clearTimer(TIMEOUT);
                    warpmac_freePayload(&txBuffer);

                    //Delay is necessary to give receiver time to push packet over ethernet...
                    //this will be removed when DMA is implemented
                    usleep(125);
                }
                break;
            case DATAPACKET:
                ackPacket.pktRev = ACKMAC;
                ackPacket.length = 0;
                ackPacket.pktType = ACKPACKET;
                memcpy(ackPacket.srcAddr,myAddr,6);
                memcpy(ackPacket.destAddr,packet->srcAddr,6);

                warpmac_sendOfdm(&ackPacket);
                warpmac_allocatePayload(packet,packet->length);
                warpmac_copyPayload(packet);
                warpmac_sendEthernet(packet);
                warpmac_freePayload(packet);
                break;
        }
    }
    return 0;
}
```

- Blink the top LEDs
- If destination address is equal to my source address
  - Create an acknowledgment and send it
  - Allocate space for the payload, copy the payload, send it over Ethernet, and free the allocated space

Case 4:

"Good" acknowledgment packet received from OFDM

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

int receiveGoodPacket(Macframe* packet) {
    /*This function processes the received packet to see if it
    was addressed to this node, and sends an ACK if it is a
    data packet. Also, it pushes the received packet out to
    ethernet.*/

    warpmac_incrementLEDHigh();
    //IMPORTANT NOTE: Even though this function is passed a Macframe structure,
    //the data field is not yet filled. The user's MAC should process the packet
    //based on the header information, send out an ACK if necessary, and then
    //copy the payload out of the PHY with the warpmac_writePacket command.

    if(warpmac_addressedToMe(packet)){
    Macframe ackPacket;
        switch(packet->pktType){
            case ACKPACKET:
                if(warpmac_inTimeout()){
                    warpmac_clearTimer(TIMEOUT);
                    warpmac_freePayload(&txBuffer);

                    //Delay is necessary to give receiver time to push packet over ethernet...
                    //this will be removed when DMA is implemented
                    usleep(125);
                }
                break;
            case DATAPACKET:
                ackPacket.pktRev = ACKMAC;
                ackPacket.length = 0;
                ackPacket.pktType = ACKPACKET;
                memcpy(ackPacket.srcAddr,myAddr,6);
                memcpy(ackPacket.destAddr,packet->srcAddr,6);

                warpmac_sendOfdm(&ackPacket);
                warpmac_allocatePayload(packet,packet->length);
                warpmac_copyPayload(packet);
                warpmac_sendEthernet(packet);
                warpmac_freePayload(packet);
                break;
        }
    }
    return 0;
}
```

- Blink the top LEDs
- If destination address is equal to my source address
  - If a timeout is currently running (i.e., the node is waiting on an ACK)
    - Stop the timer
    - Free the transmitted packet from further retransmits
    - Wait for the other node to get ready

Case 5:

Timeout timer expires

```
int timerExpire(unsigned char timerType){
    /*This function is responsible for handling TIMEOUTs and BACKOFFs.
    It is registered using the warpmac_setTimerHandler function in
    myMac_main. The job responsibilities of this function are to:
    -increase the contention window upon the expiration of a TIMEOUT
    -initiate a BACKOFF timer upon the expiration of a TIMEOUT
    -retransmit a packet upon the expiration of a BACKOFF*/
    int status;

    switch(timerType){
        case TIMEOUT:
            if(txBuffer.isNew){
                status = warpmac_incrementResend(&txBuffer);
                if(status == 0){
                    return 0;
                }
                warpmac_setTimer(BACKOFF);
                return 0;
            }
                break;
        case BACKOFF:

    if(warpmac_carrierSense()){ //This is somewhat of an overkill
        warpmac_sendOfdm(&txBuffer);
        warpmac_setTimer(TIMEOUT);
    }
    else{
        warpmac_setTimer(BACKOFF);
    }
    break;
    return 0;
    }
}
```

- Increment the resend field of the packet
- Enter a backoff

Case 6:

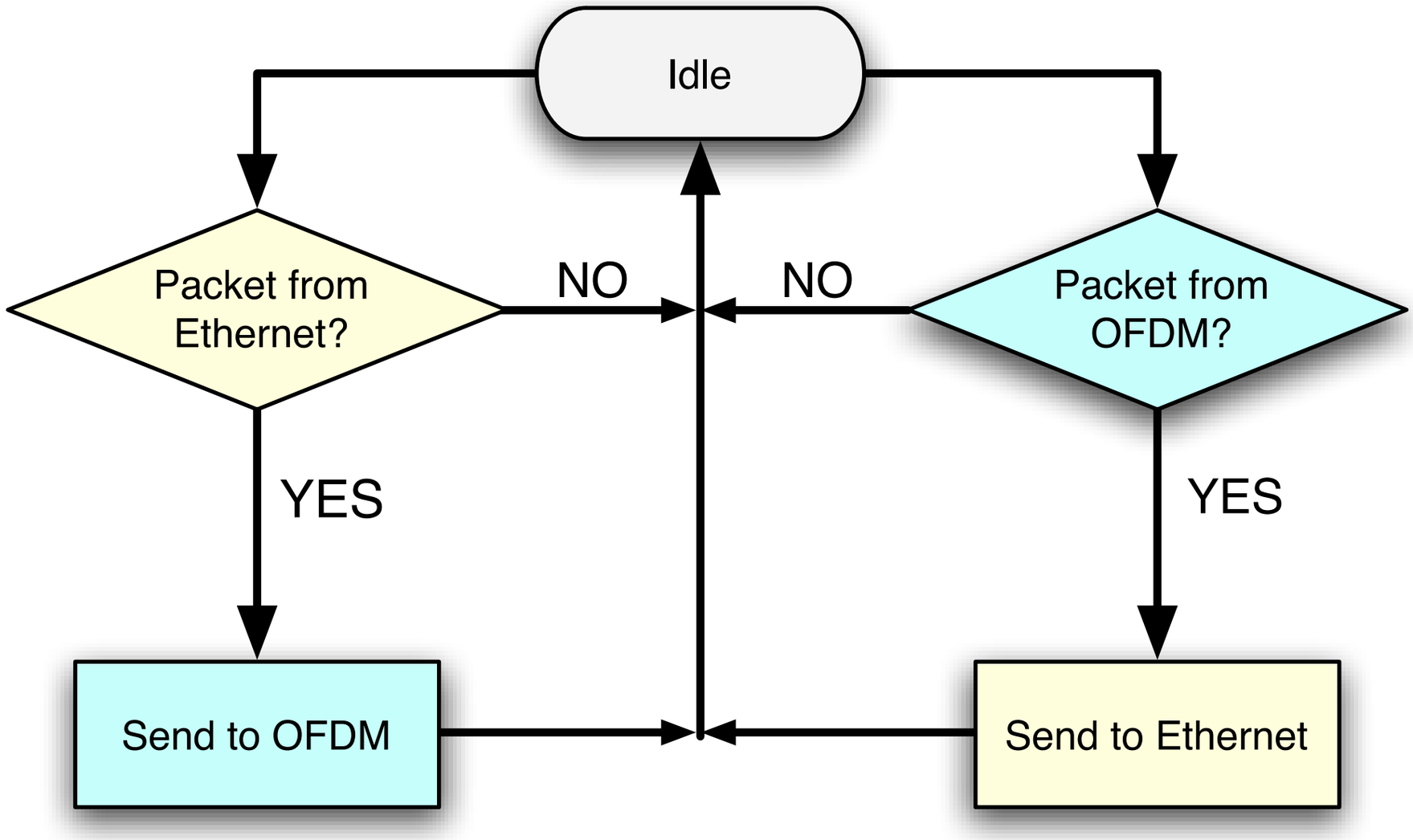
Backoff timer expires

```
int timerExpire(unsigned char timerType){
    /*This function is responsible for handling TIMEOUTs and BACKOFFs.
    It is registered using the warpmac_setTimerHandler function in
    myMac_main. The job responsibilities of this function are to:
    -increase the contention window upon the expiration of a TIMEOUT
    -initiate a BACKOFF timer upon the expiration of a TIMEOUT
    -retransmit a packet upon the expiration of a BACKOFF*/
    int status;

    switch(timerType){
        case TIMEOUT:
            if(txBuffer.isNew){
                status = warpmac_incrementResend(&txBuffer);
                if(status == 0){
                    return 0;
                }
                warpmac_setTimer(BACKOFF);
                return 0;
            }
                break;
        case BACKOFF:

    if(warpmac_carrierSense()){ //This is somewhat of an overkill
        warpmac_sendOfdm(&txBuffer);
        warpmac_setTimer(TIMEOUT);
    }
    else{
        warpmac_setTimer(BACKOFF);
    }
    break;
    return 0;
    }
}
```

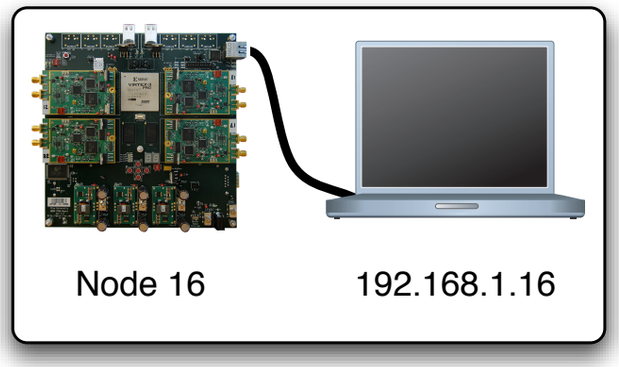
- If the medium is free
  - Send it over OFDM
  - Enter a timeout
- Otherwise, start another timeout

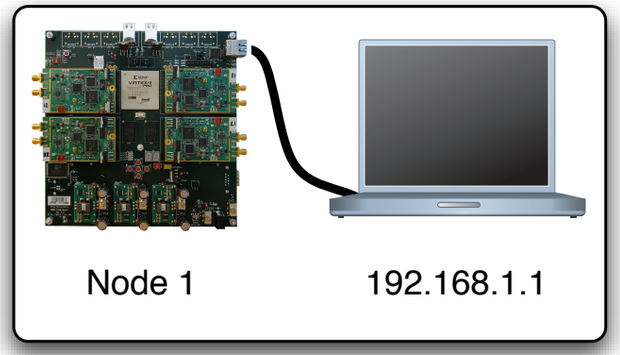
Labwork

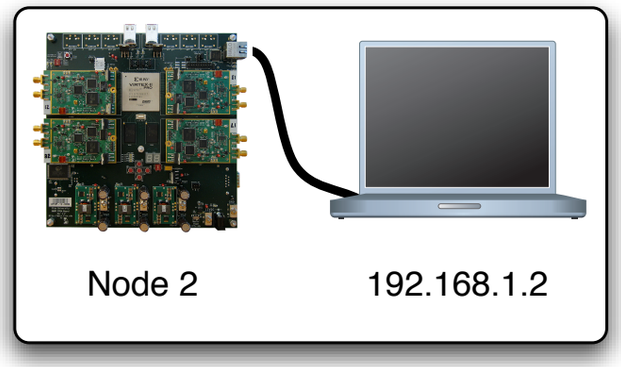
noMac

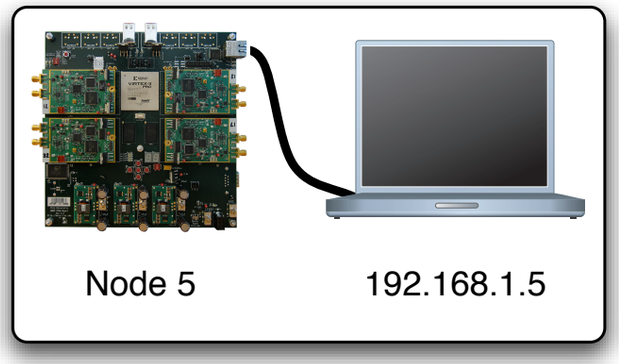

Idle
Packet from Ethernet?
NO
NO
Packet from OFDM?
YES
YES
Send to OFDM
Send to Ethernet

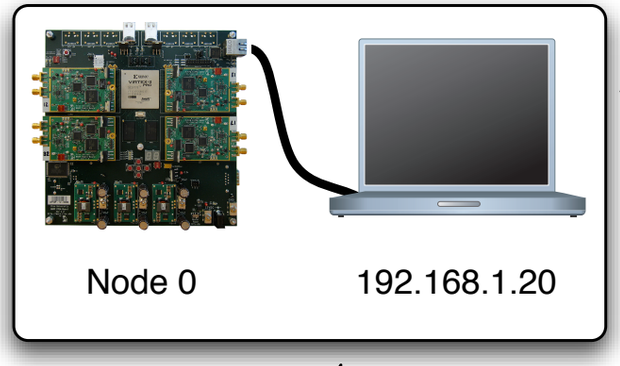
uniMac

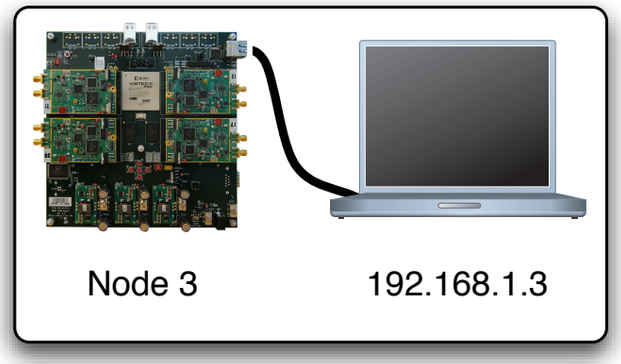


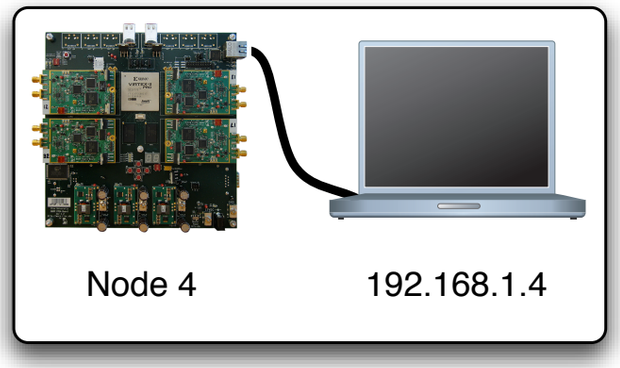






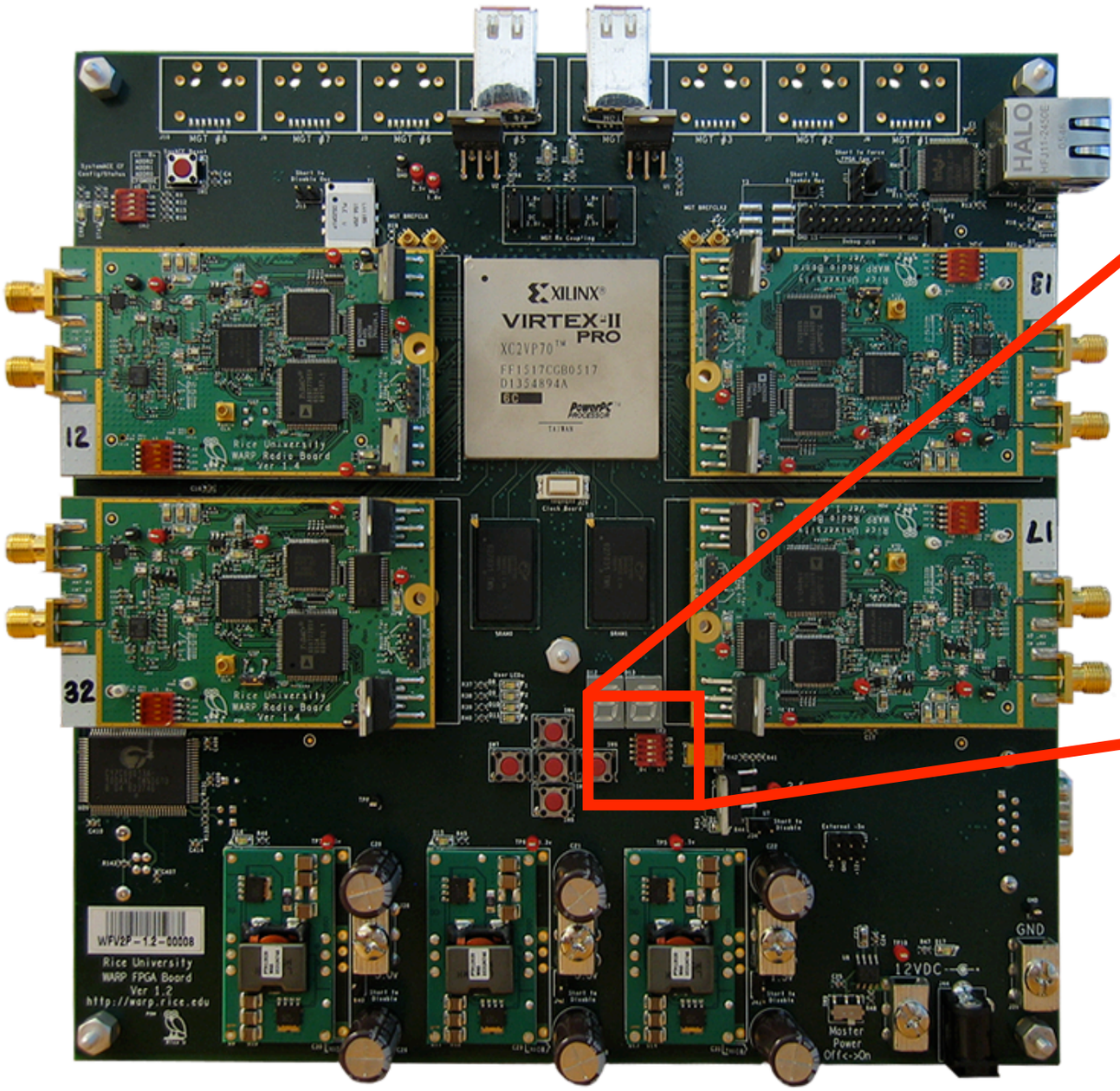


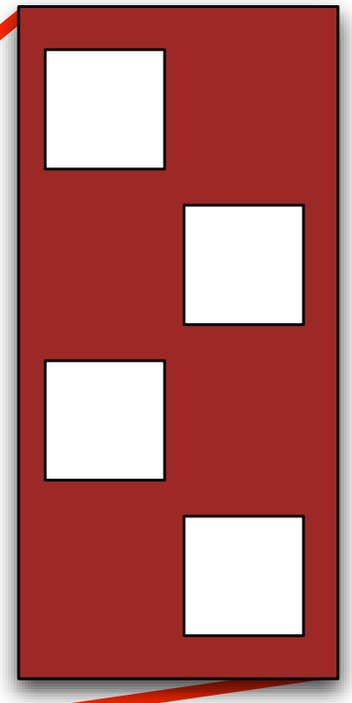




UDP Audio

uniMac Lab



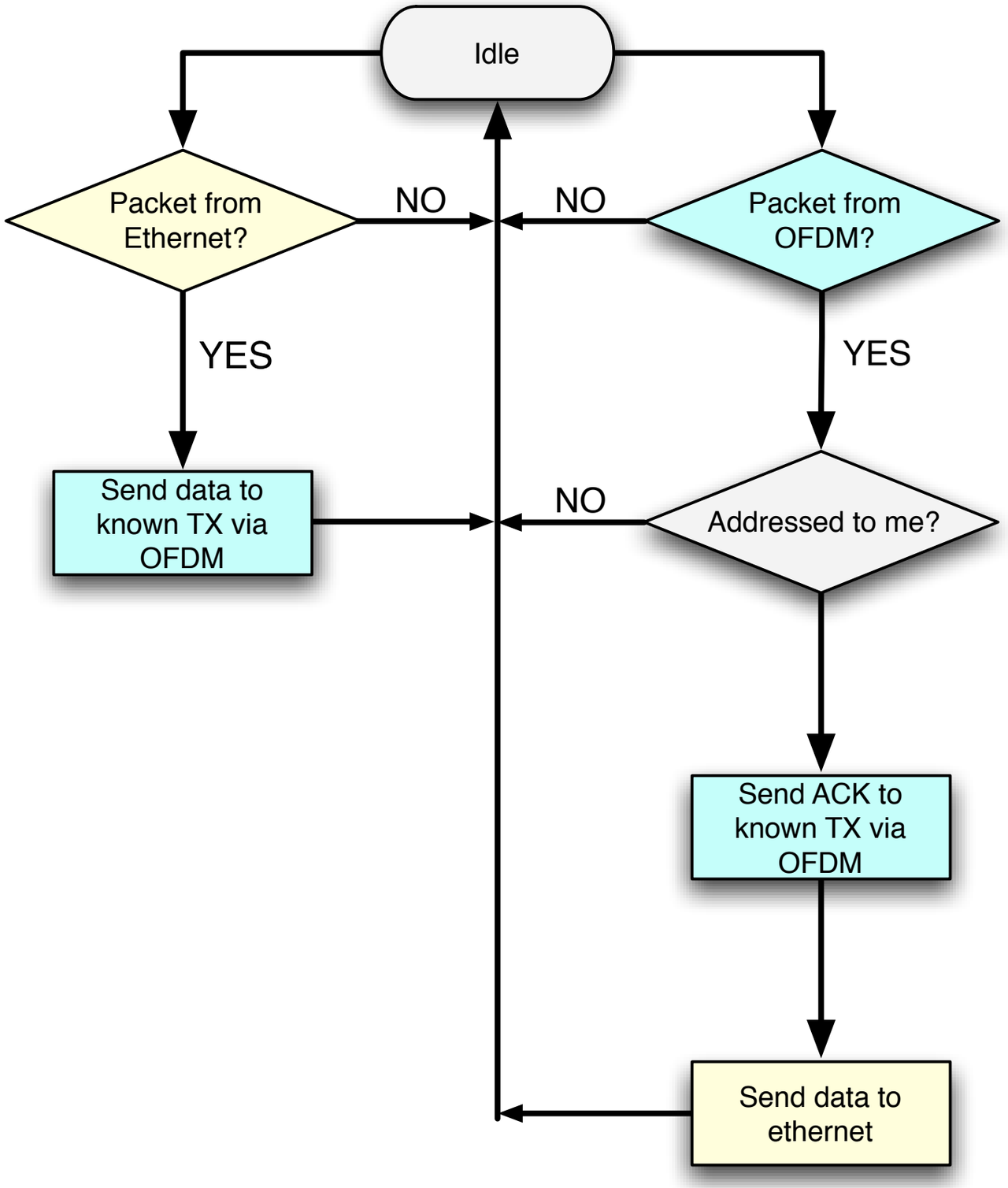


Most Significant Bit (MSB)

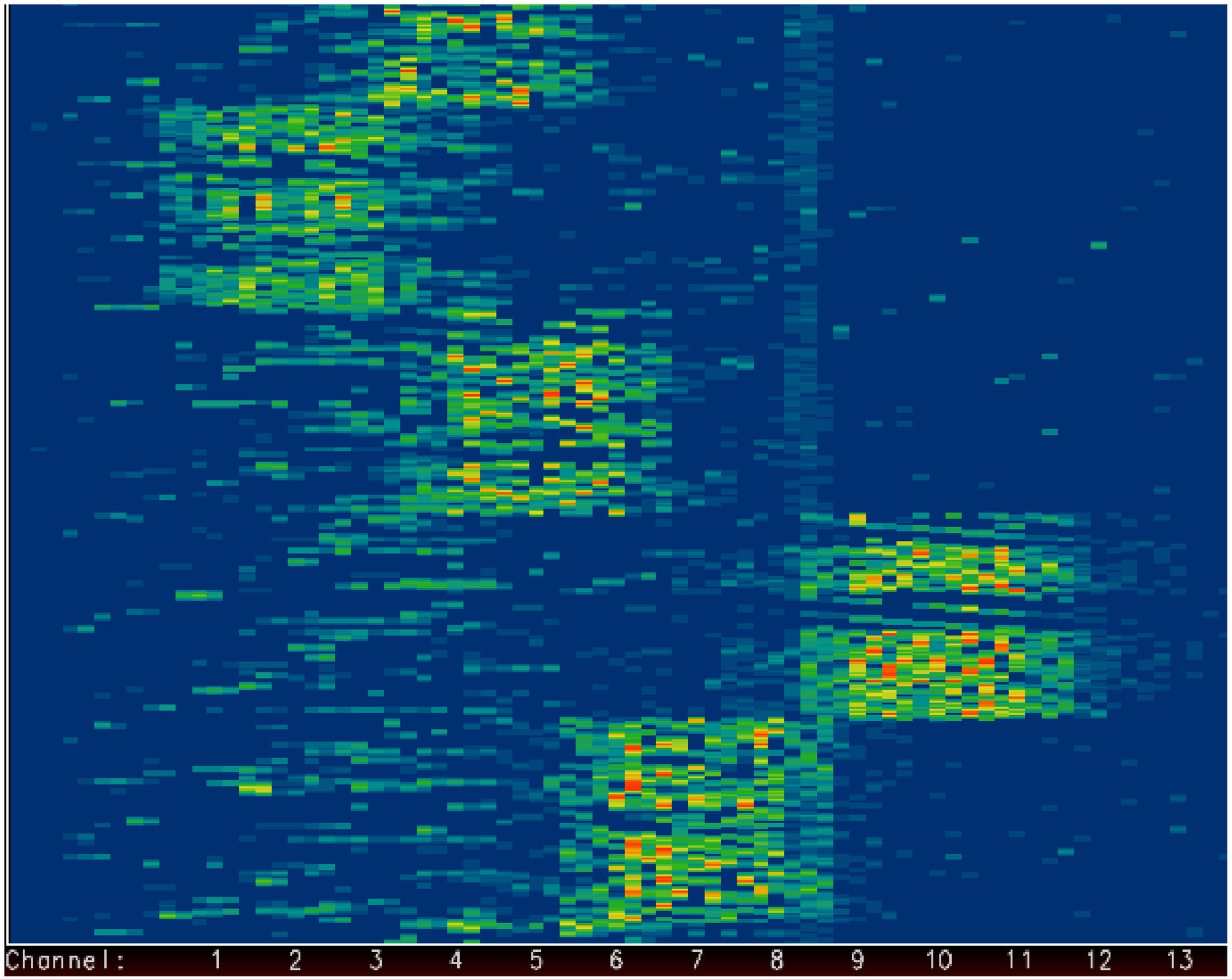
Least Significant Bit (LSB)

Node 5

uniMac Lab


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

lemmingMac Lab



lemmingMac Lab