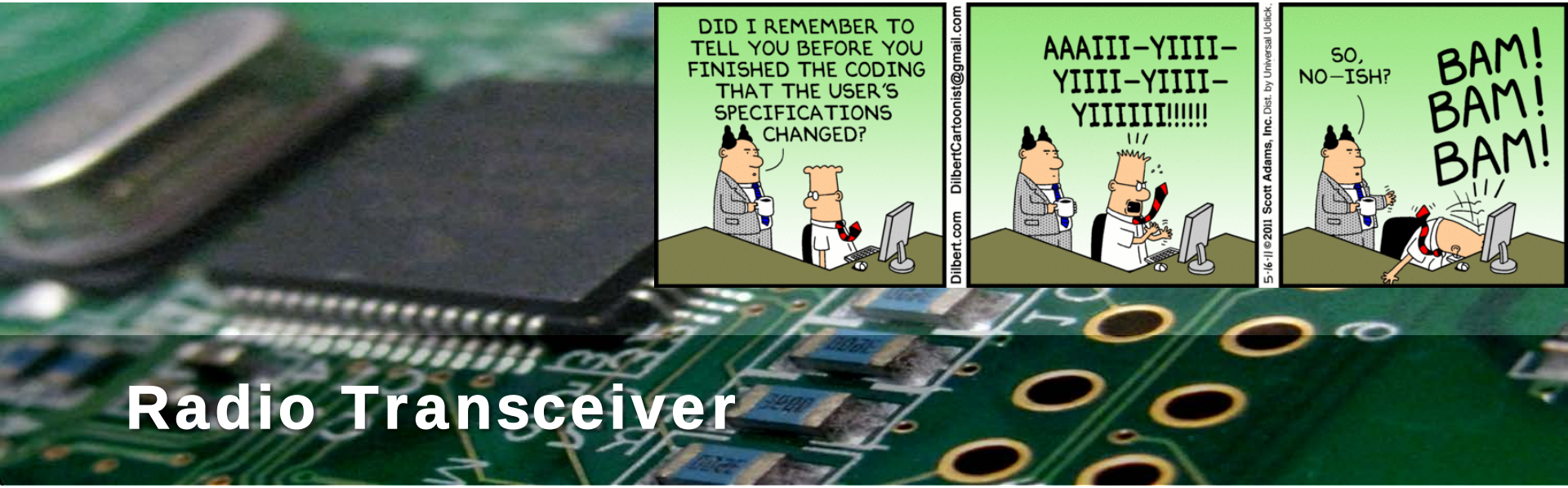
Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur

Source: Dilbert.com



Radio Transceiver

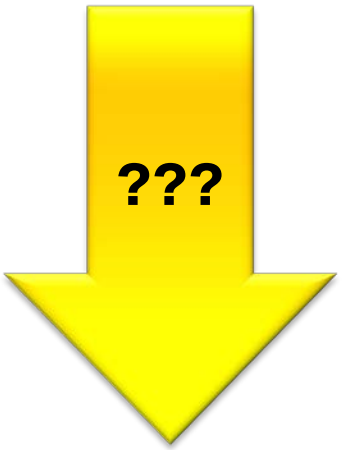
"Send or receive, that's the question."

Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

Learning Goals

- Ability to 'assist' robot with Remote board
  - Driving, escaping, ...
  - Remote controller
  - Two modes: assisted, autonomous

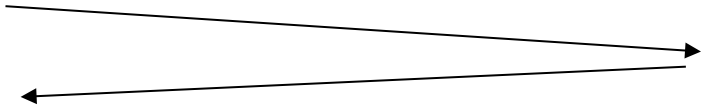
- Mounting hardware
- Communication, Systems, ...
- Enhancing projects with 2.4 GHz Radio
- Send/Receive radio messages
  - Shell usage
- Protocol definition
  - Format



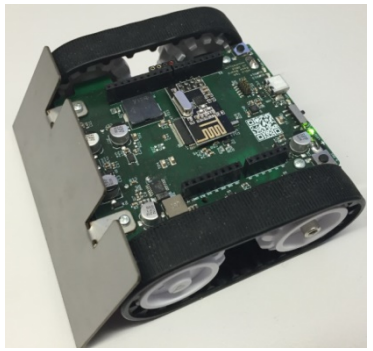
Communication Goal

- Communication between Remote and Robot
  - Buttons, Joystick
  - Desired speed
  - Switch between manual and auto mode
- Receiver acknowledges message

Packet with Data

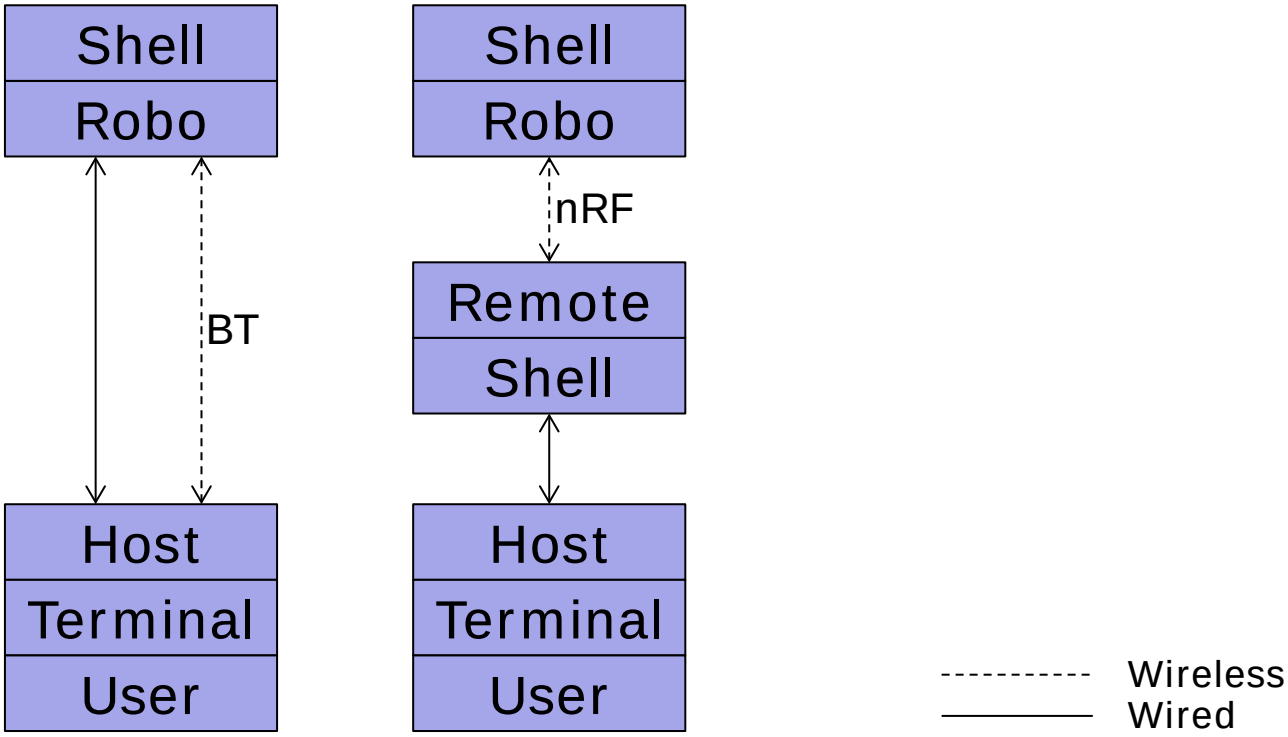


Acknowledge



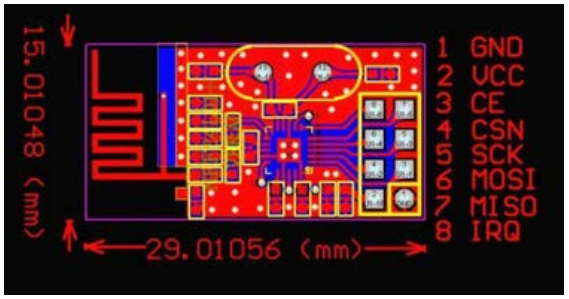
Requirements and Use Cases

- Wireless communication link between robot and host/remote controller
  - User sends 'status' to robot
  - Remote control using joystick & buttons

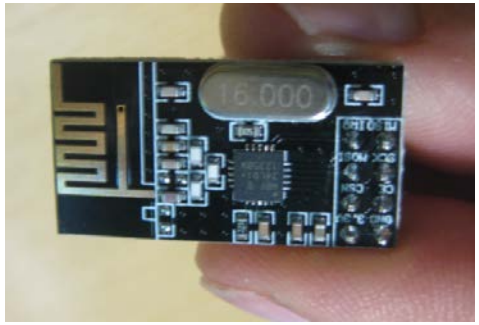


Nordic Semiconductor nRF24L01+

- 2.4 GHz Transceiver it's a free band
- Proprietary (Nordic) Protocol ('ShockBurst') with optional auto-acknowledge
- Max 32 Bytes of payload
- SPI Interface, max 10 MHz clock
- 128 channels
- 250 kBit, 1 Mbit, 2 Mbit



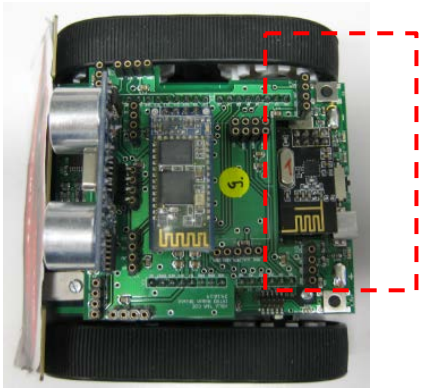
Source: DX.com



3.3V must be constant, otherwise communication will be interrupted

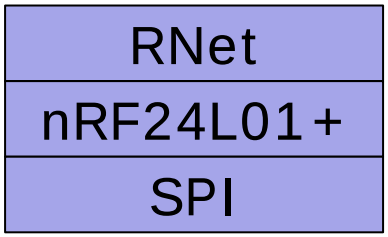
nRF Module Installation

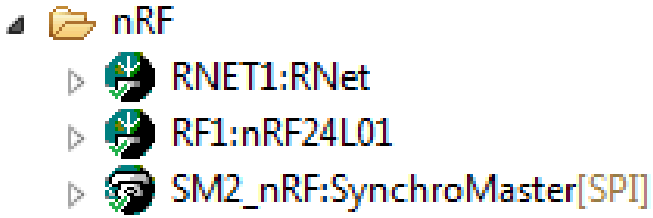
- Robot: module on base board **or** shield
  - ➔ need it on base board with ToF Shield!
- Remote: bottom side



RNet Network and Components

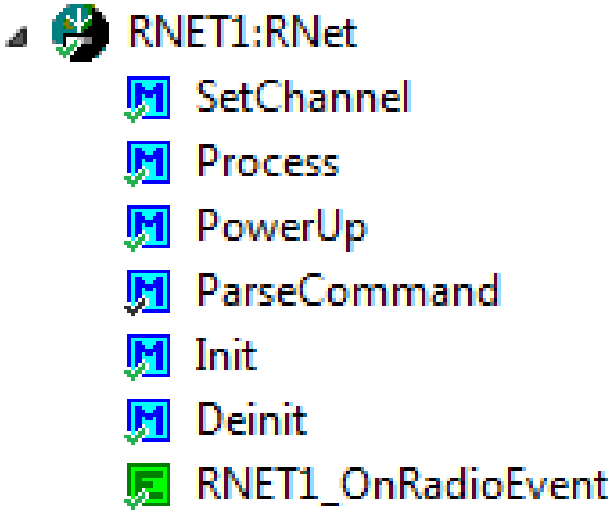
- Implementation of Radio Network Stack
- Supports nRF24L01+
- Short (8bit) and 16bit addresses — choose in the software
- Optional ACK and Retry — only optional, can choose if you want to have a acknowledge
- Remote Std I/O
- FreeRTOS Tx and Rx Msg queues
- Simplified ISO network layers





RNet Component

- Configuration of stack

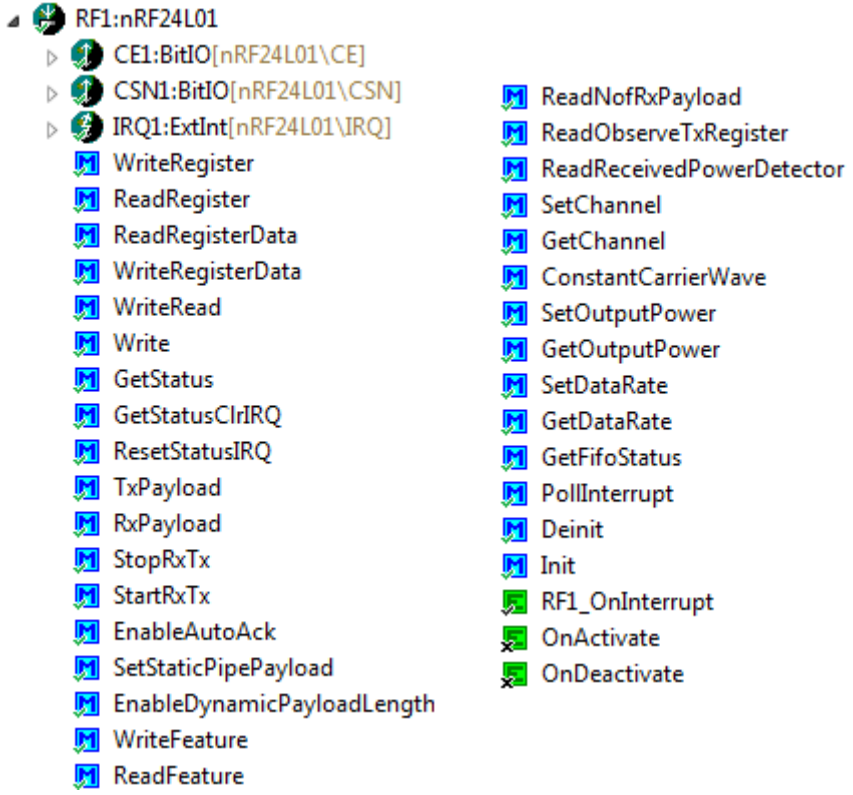


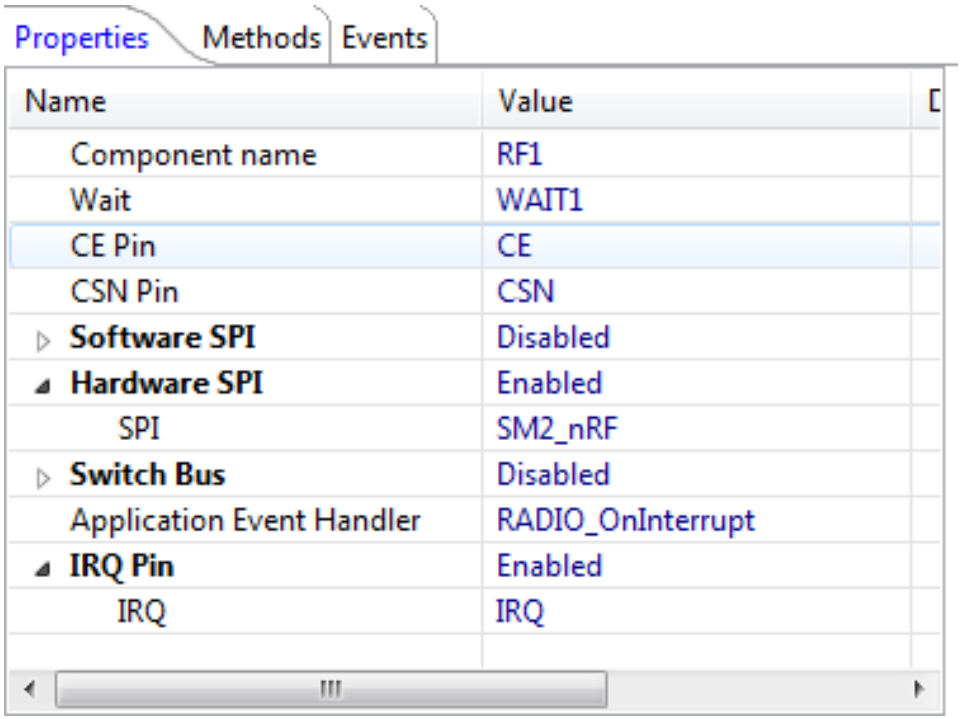
Component name	RNET1
Transceiver	
Transceiver Type	nRF24L01+
nRF24L01+	Enabled
nRF24L01+	RF1
Radio Channel	0
Data Rate	2000 kBit
Payload Size	32
Address (data pipe 0)	0x11, 0x22, 0x33, 0x44, 0x55
Address (data pipe 1)	0xB3, 0xB4, 0xB5, 0xB6, 0xF1
Address (data pipe 2)	0xF2
Address (data pipe 3)	0xF3
Address (data pipe 4)	0xF4
Address (data pipe 5)	0xF5
SMAC	Disabled
Network	
Address Size	8 Bits
Queues	
Rx Message Queue Size	6
Tx Message Queue Size	6
Message Queue Blocking Tim	200
Send Retry Count	3
Send Timeout (ms)	500
System	
SDK	KSDK1
Utility	UTIL1
RTOS	FRTOS1
Shell	Enabled
Remote StdIO	Enabled
Queue length	48
Queue Timeout (ms)	500
Shell	CLS1

there are 128, choose differenz channel!

nRF24L01+ Component

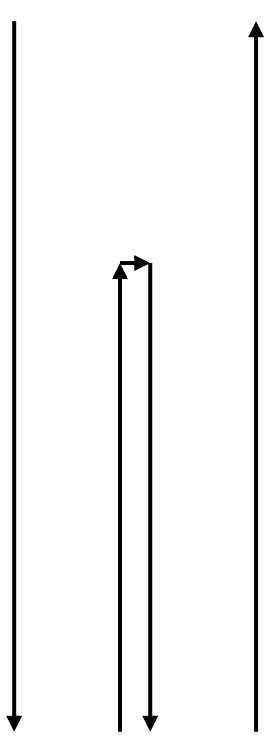
- IRQ Pin only on Robot, not on Joystick Shield/Remote
  - Polling/Gadfly synchronization
- CSN (chip select) and CE (chip enable Rx/Tx)

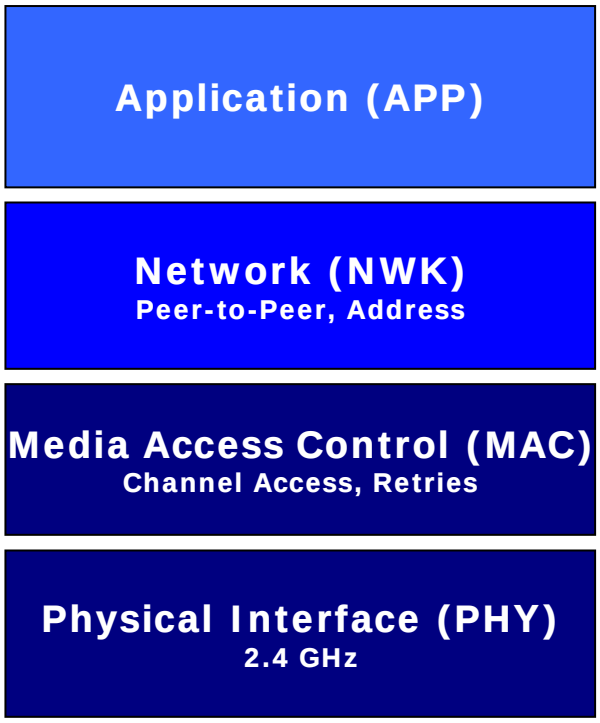




Simplified Network Layers

- Layers
  - Responsibility
  - Modularity
  - Scalability





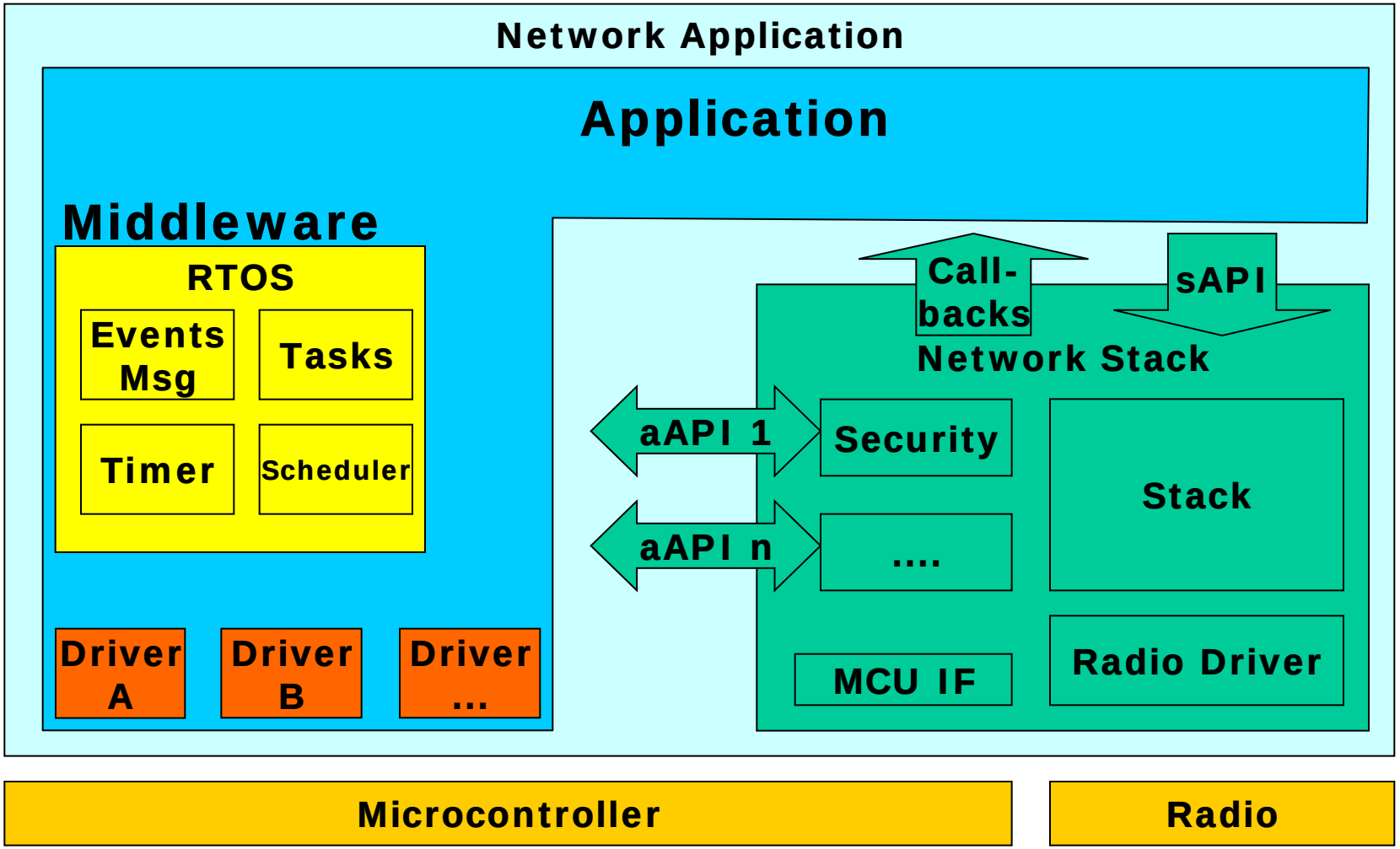
HandleRxMessage(src, msg);

Resend(MsgDsc); Route();
Connect(); Encrypt(); GetNodeAddress()

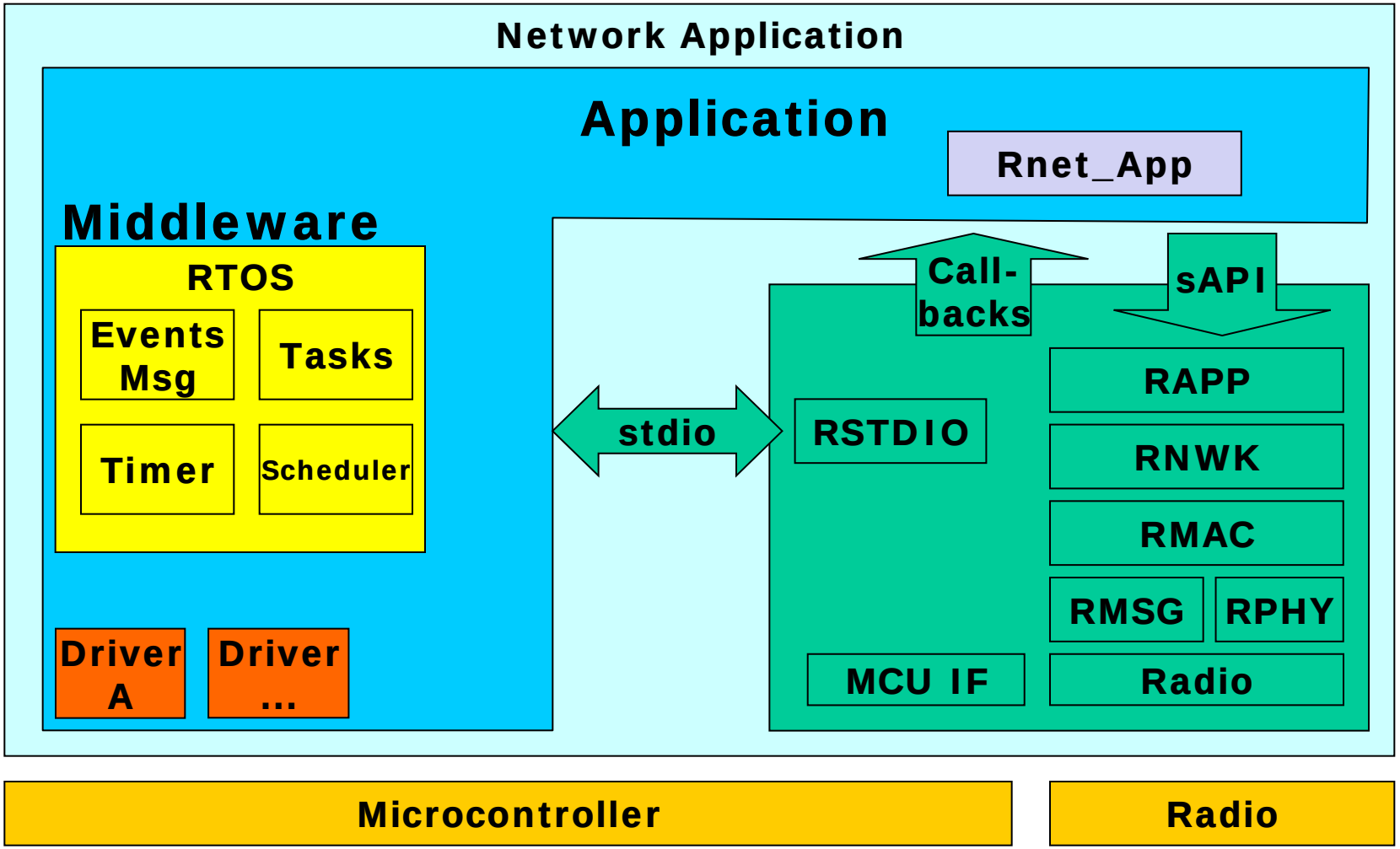
SendACK(); SequenceNumber();

SelectChannel(6); SetDataRate(1);
SendSPI();

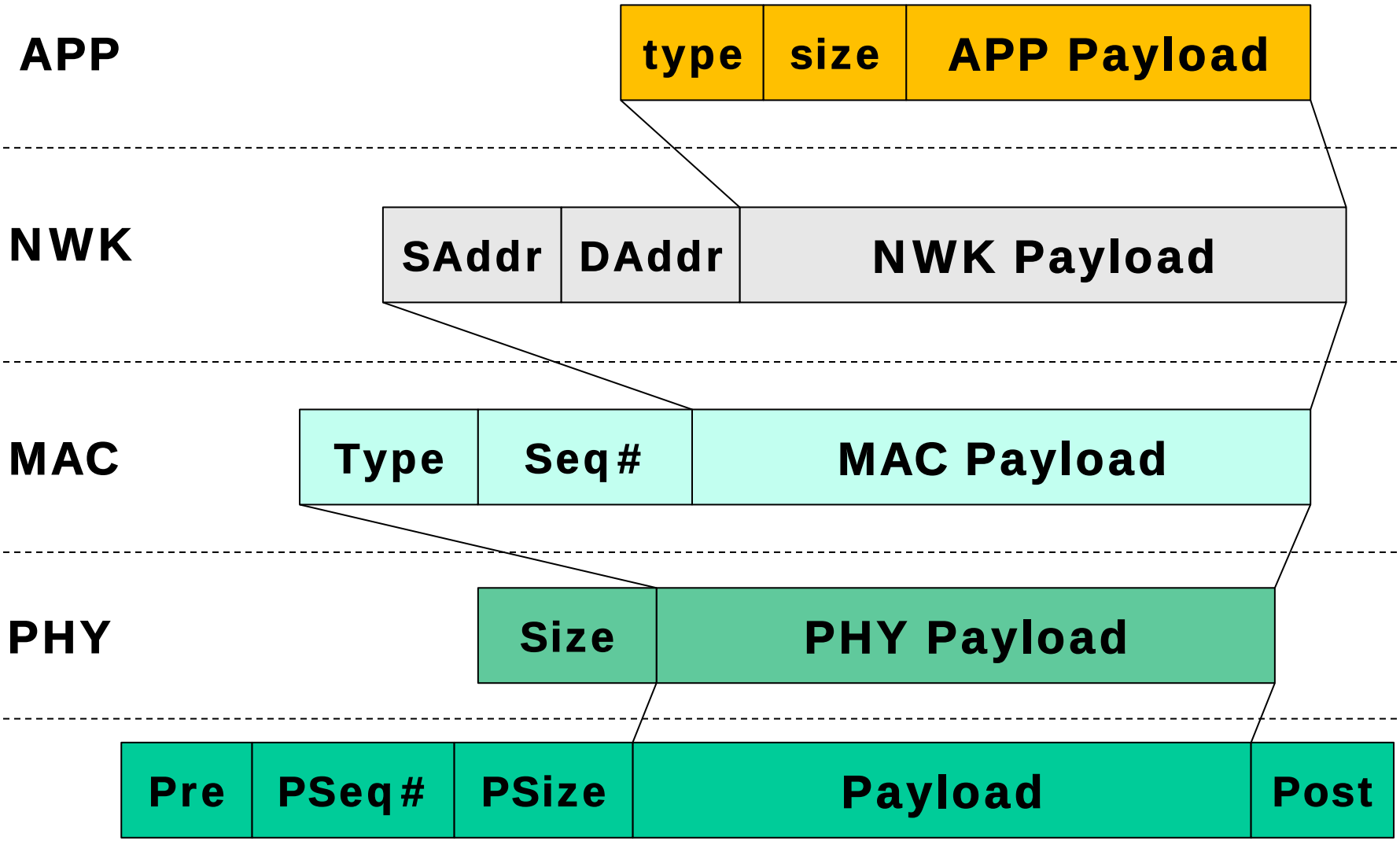
Typical Application + Network Stack



RNet Stack Architecture

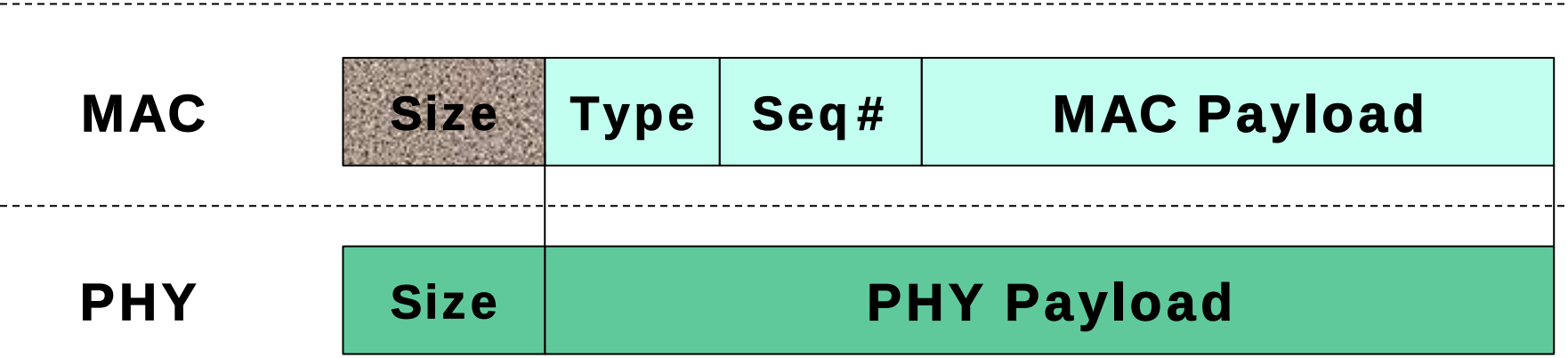


Payload Packaging



Copy-Less Stack Operation

- Packets are passed up and down the stack
- Data is added to the beginning and end
- Issue: copy
- Solution:
  - Same buffer is passed through stack
  - Modifications directly in 'physical' payload



Copy-Less Access Macros

```
/* payload format is:
 * PHY: <size><phy payload>
 * MAC:       <type><seq#><mac payload>
*/
#define RMAC_HEADER_SIZE   (2) /* <type><seq#> */
#define RMAC_PAYLOAD_SIZE  (RPHY_PAYLOAD_SIZE-RMAC_HEADER_SIZE)
#define RMAC_BUFFER_SIZE   (RPHY_BUFFER_SIZE)

/* PHY buffer access macros */
#define RMAC_BUF_IDX_TYPE                 (RPHY_BUF_IDX_PAYLOAD+0)
#define RMAC_BUF_IDX_SEQNR                (RPHY_BUF_IDX_PAYLOAD+1)
#define RMAC_BUF_IDX_PAYLOAD              (RPHY_BUF_IDX_PAYLOAD+2)

#define RMAC_BUF_TYPE(phy)                ((phy)[RMAC_BUF_IDX_TYPE])
#define RMAC_BUF_SEQN(phy)                ((phy)[RMAC_BUF_IDX_SEQNR])

uint8_t RMAC_PutPayload(uint8_t *buf, size_t bufSize, uint8_t payloadSize) {
  RMAC_BUF_TYPE(buf) = RMAC_MSG_TYPE_DATA;
  RMAC_ExpectedAckSeqNr = RMAC_SeqNr;
  RMAC_BUF_SEQN(buf) = RMAC_SeqNr++;
  return RPHY_PutPayload(buf, bufSize, payloadSize+RMAC_HEADER_SIZE);
}
```

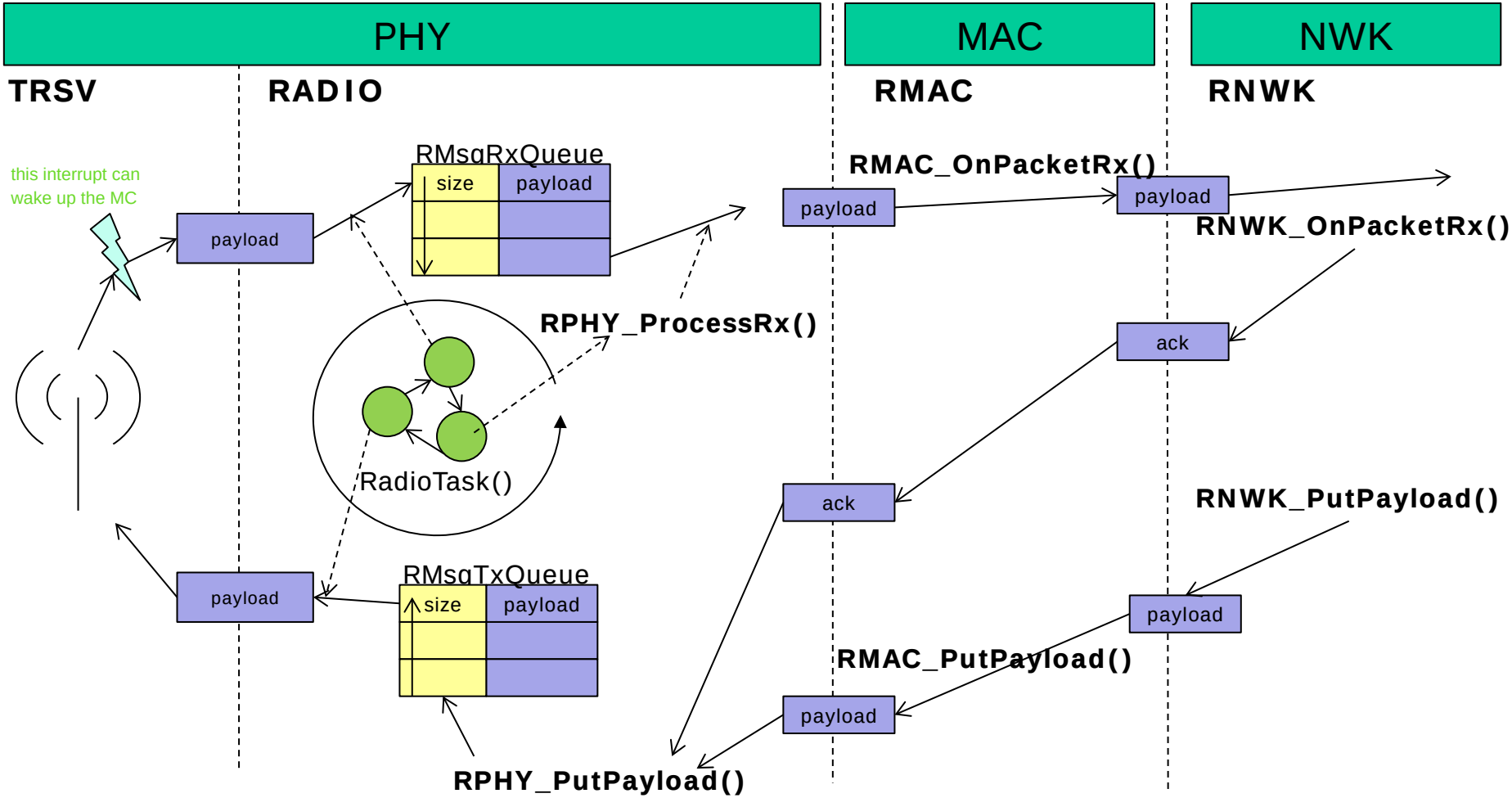
Example with Payload Access Macros

- Full payload buffer is passed through API
- Macros are accessing bytes inside buffer
- Flags for special handling (ACK, power down, …)

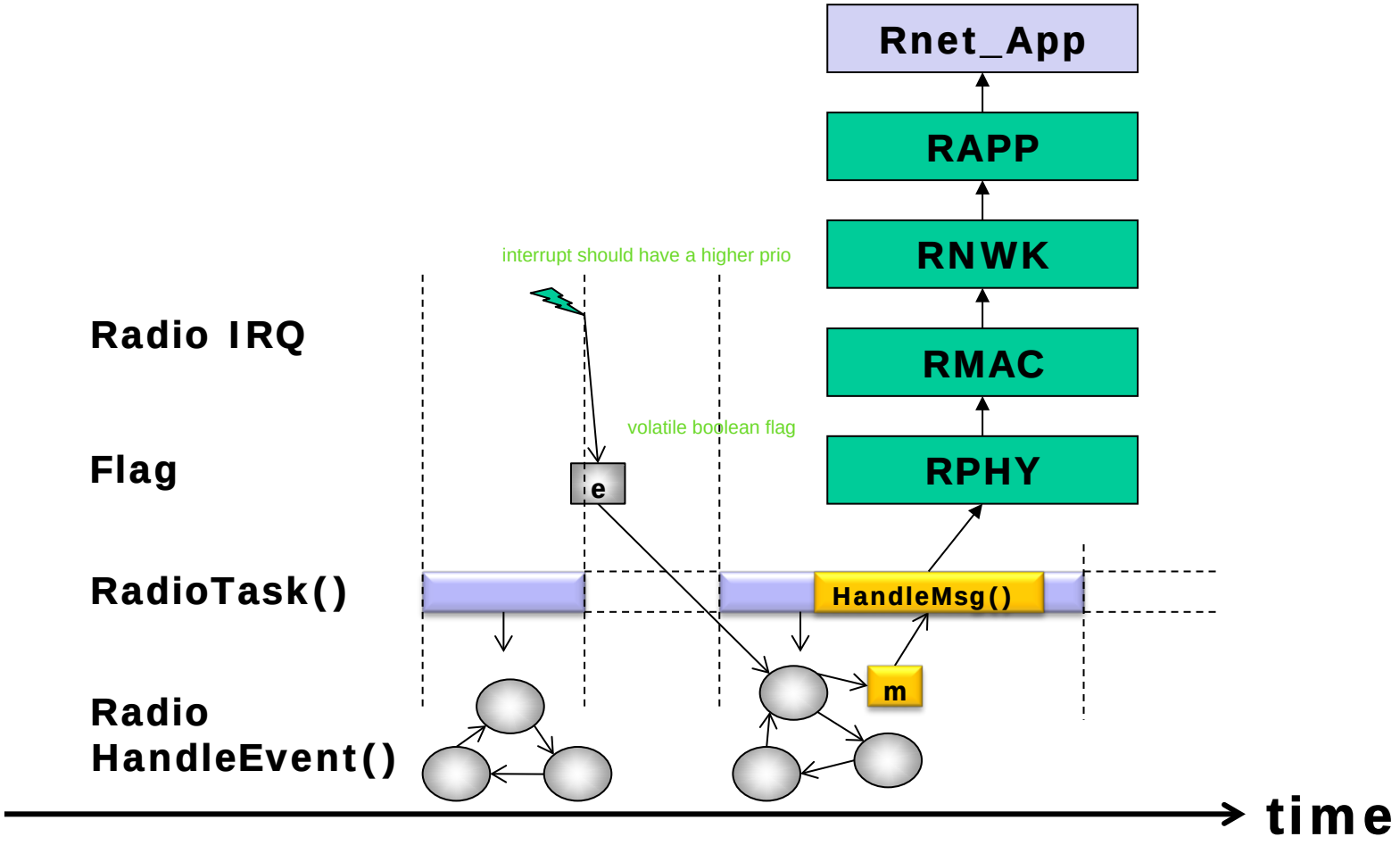
```
static uint8_t RMAC_SeqNr = 0;
static uint8_t RMAC_ExpectedAckSeqNr;
uint8_t RMAC_PutPayload(uint8_t *buf, size_t bufSize,
   uint8_t payloadSize, RPHY_FlagsType flags)
{
  if (flags&RPHY_PACKET_FLAGS_REQ_ACK) {
    RMAC_BUF_TYPE(buf) = RMAC_MSG_TYPE_DATA|RMAC_MSG_TYPE_REQ_ACK;
  } else {
    RMAC_BUF_TYPE(buf) = RMAC_MSG_TYPE_DATA;
  }
  RMAC_ExpectedAckSeqNr = RMAC_SeqNr;
  RMAC_BUF_SEQN(buf) = RMAC_SeqNr++;
  return RPHY_PutPayload(buf, bufSize, payloadSize+RMAC_HEADER_SIZE,
    flags);
}
```

checks if you requested an acknowledge

Payload Handling



Packet Handling



Radio Shell Commands

```
radio                          ; Group of radio commands
  help|status                  ; Shows radio help or status
  channel <number>             ; Switches to the given
                                 channel. Channel must be
                                 in the range 0..127
  power <number>               ; Changes output power to 0,
                                 -10, -12, -18
  sniff on|off                 ; Turns sniffing on or off
  writereg 0xReg 0xVal         ; Write a register
  flush                        ; Empty all queues
  printreg                     ; Print the radio registers
app                            ; Group of application commands
  help                         ; Shows radio help or status
  saddr 0x<addr>               ; Set source node address
  daddr 0x<addr>               ; Set destination node address
  send val <val>               ; Set a value to the destination node
  send (in/out/err)            ; Send a string to stdio using the
                                 wireless transceiver
```

Radio Shell Status

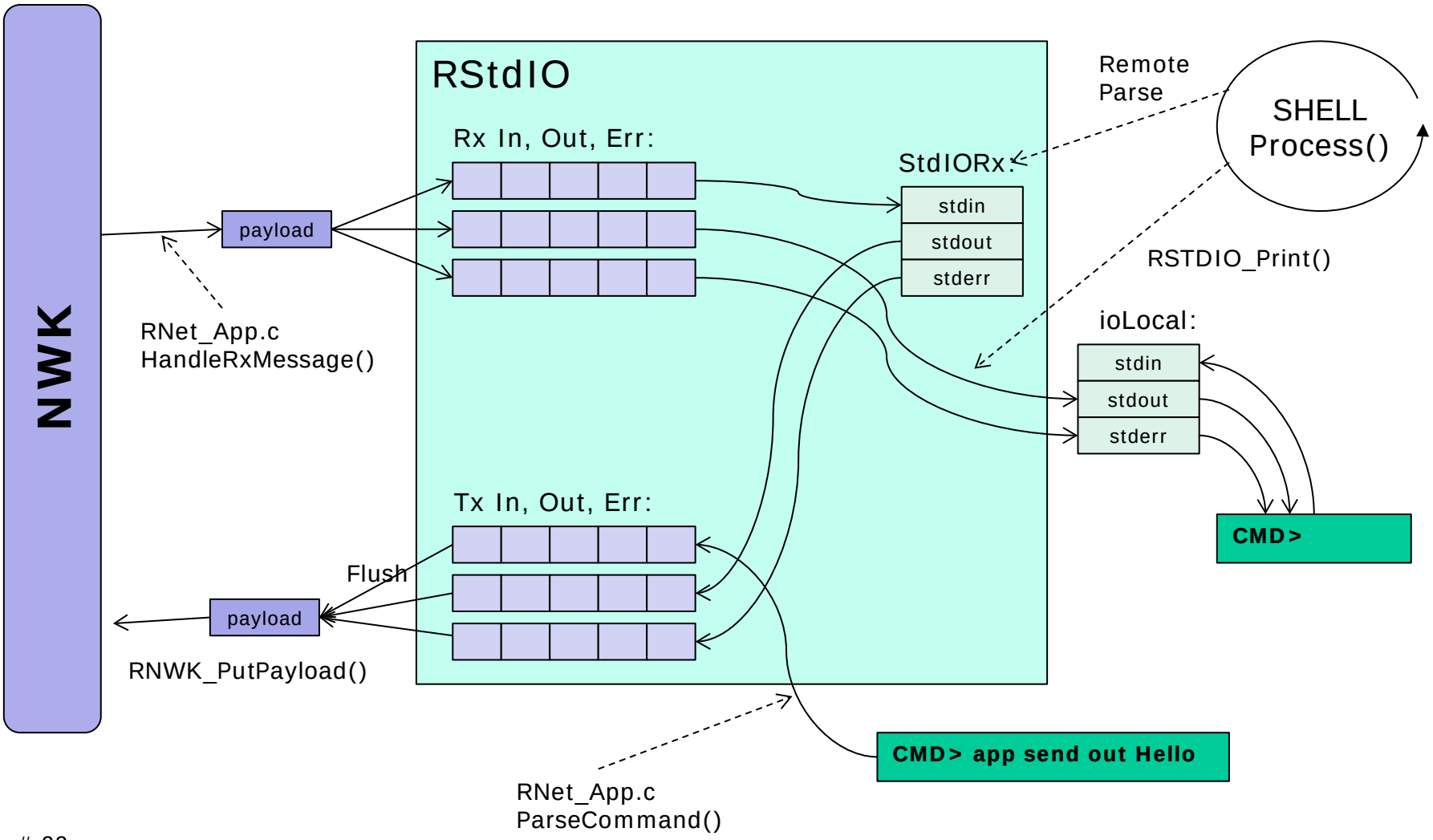
```
Radio          :
  state        : READY_TX_RX
  sniff        : no
  channel      : 0 (HW), 0 (SW)
  power        : 0 dBm
  data rate    : 1000 kbps
  STATUS       : 0x0E: RxFifoEmpty
  FIFO_STATUS: 0x11: TX_EMPTY RX_EMPTY
  OBSERVE_TX : 0 lost, 0 retry
rnwk           :
  addr         : 0xFF
app            :
  dest addr    : 0xFF
```

Radio Shell Status

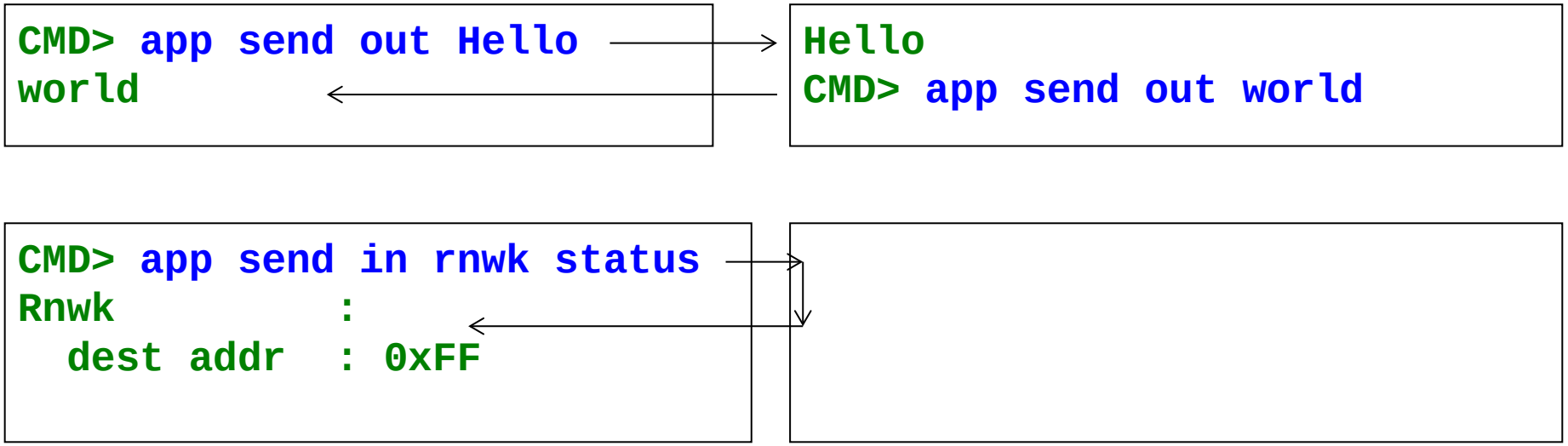
```
CMD> radio sniff on
CMD> app send val 0
Packet Tx flags: 0 size: 34 PHY data: 00 07 01 00 FF FF 04 01
 00  MAC size:7 type:01(NACK|DATA) s#:00 NWK src:FF dst:FF
 APP type:04 size:01
CMD> app send val 5
Packet Tx flags: 0 size: 34 PHY data: 00 07 01 01 FF FF 04 01
 05  MAC size:7 type:01(NACK|DATA) s#:01 NWK src:FF dst:FF
 APP type:04 size:01
```

```
CMD> radio sniff on
Packet Rx flags: 0 size: 34 PHY data: 00 07 01 00 FF FF 04 01
00  MAC size:7 type:01(NACK|DATA) s#:00 NWK src:FF dst:FF APP
type:04 size:01
Data: 0 from addr 0xFF
Packet Rx flags: 0 size: 34 PHY data: 00 07 01 01 FF FF 04 01
05  MAC size:7 type:01(NACK|DATA) s#:01 NWK src:FF dst:FF APP
type:04 size:01
Data: 5 from addr 0xFF
```

Application: Remote StdIO Bridge



Radio Shell Std I/O



Rnet App: Sending Example Message

```
/*!
 * \brief Send an application payload data block.
 * \param appPayload Size of application payload.
 * \param appPayloadSize Application payload size.
 * \param msgType Payload message type.
 * \param dstAddr destination address.
 * \param flags Packet flags.
 * \return Error code, ERR_OK for no failure.
 */
uint8_t RAPP_SendPayloadDataBlock(uint8_t *appPayload,
  uint8_t appPayloadSize, uint8_t msgType,
  RNWK_ShortAddrType dstAddr, RPHY_FlagsType flags);
```

Example:

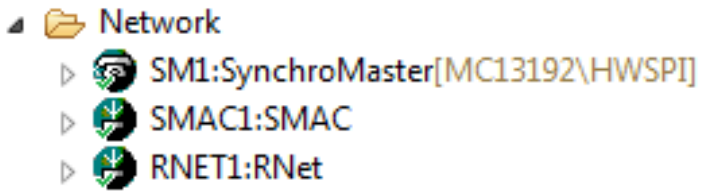
```
(void)RAPP_SendPayloadDataBlock(&val8, sizeof(val8),
  RAPP_MSG_TYPE_DATA, APP_dstAddr, RPHY_PACKET_FLAGS_NONE);
```

Rnet_App: Receiving Example Message

```
static uint8_t HandleRxMessage(RAPP_MSG_Type type, uint8_t size, uint8_t *data,
  RNWK_ShortAddrType srcAddr, bool *handled, RPHY_PacketDesc *packet)
{
  uint8_t buf[16], val;
  CLS1_ConstStdIOTypePtr io = CLS1_GetStdio();

switch(type) {
    case RAPP_MSG_TYPE_DATA:
      *handled = TRUE;
      val = *data; /* get data value */
      CLS1_SendStr((unsigned char*)"Data: ", io->stdOut);
      CLS1_SendNum8u(val, io->stdOut);
      CLS1_SendStr((unsigned char*)" from addr 0x", io->stdOut);
      buf[0] = '\0';
      UTIL1_strcatNum8Hex(buf, sizeof(buf), srcAddr);
      UTIL1_strcat(buf, sizeof(buf), (unsigned char*)"\r\n");
      CLS1_SendStr(buf, io->stdOut);
      return ERR_OK;
    default:
      break;
  } /* switch */
  return ERR_OK;
}
```

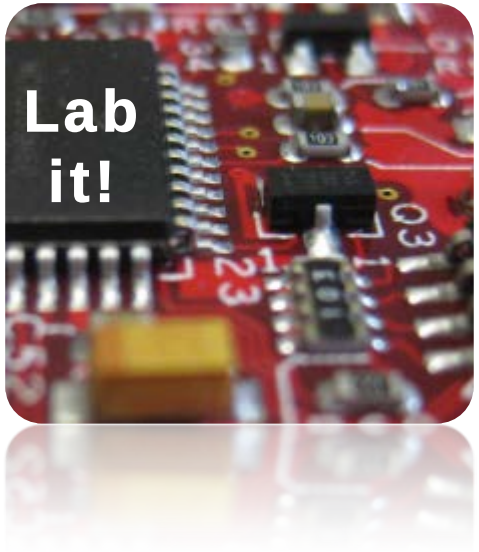
RNet Components and Application Files



- RNet_AppConfig.h
  - Stack configuration file (message types, payload size, ...)
  - Overwrite the defaults in the stack
- RNet_App.h
  - Header file for network part of application
- Rnet_App.c
  - Radio Task and message handling

Lab: Radio Transceiver

- Install Hardware
- Add Network components
  - SynchroMaster
  - nRF24L01+
  - RNet
- Inspect/understand RNet Architecture



- RNet_App.c
  - Integrate into application
  - Node Address
  - Channel
  - HandleDataRxMessage()
  - Send Message

Radio Recap Questions

- Which two kind of acknowledge exist with the nRF24L01+ and RNet stack?
- On the Remote there is no IRQ line for the nRF24L01+: what does this mean for the software?
- Can you explain what ‘asynchronous API’ means for the network stack?
- What are the pros and cons for copy-less stack operations?
- Why is the Remote StdIO implementation using FreeRTOS queues?