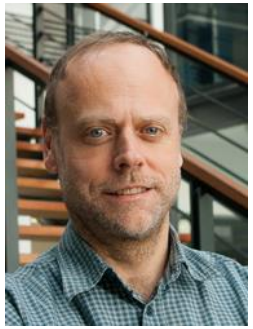
# SDR tools for experimentation: And Overview of the use of NI USRPs and emerging O-RU

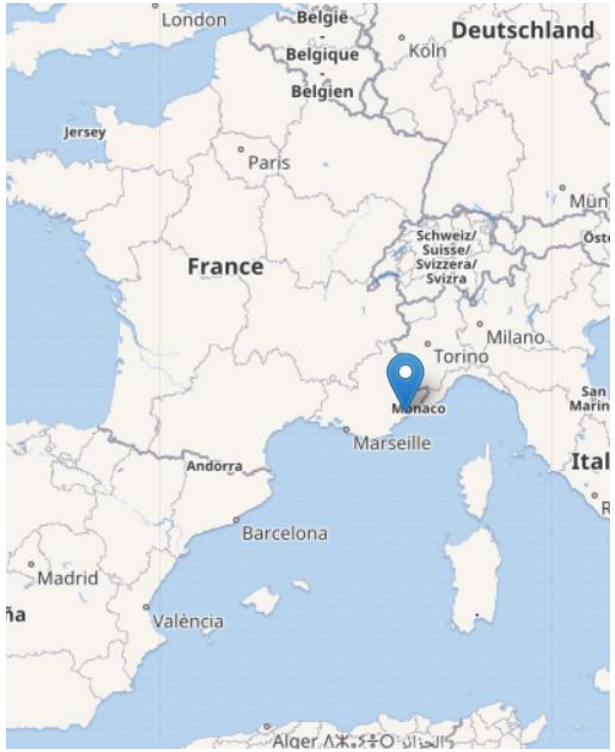
Prof. Raymond KNOPP, EURECOM

# $ whoami



**Raymond Knopp**

- Dept. Head Communication Systems @ **EURECOM**
- President of the OpenAirInterface Software Alliance
- Long-time SDR and RAN enthusiast

# Outline

- High-level descriptions of SDRs when used as 3GPP radio-units
  - Fronthaul systems and protocols and associated challenges
  - Capabilities of off-the-shelf USRPs
- A look at UHD real-time interfacing for 5G NR
- An ECPRI example (AW2S devices)
- Transitioning software radios like OAI to O-RAN Open Fronthaul Interface devices

# Software-Defined Radio (SDR)

- The canonical SDR is the National Instruments USRP (Universal Software Radio Peripheral)
- Basic research tool for experimentation with radio-access network processing and protocols
- Real-time interconnection of a PC-based application with a radio-frequency (RF) front-end
- Many commonalities with current commercial radio-units
  - Similar RF chipsets (high-end USRPs like N3x0, X4x0)
  - FPGA logic making use of FPGA SoC (e.g. AMD-Xilinx Zynq Ultrascale)
- Main differences with cellular industry-grade radio-units
  - Research focus
  - No built-in power circuits but wideband tuning
  - Different open-source fronthaul protocol (USRP Hardware Driver)
    - To make a USRP look like an eCPRI / O-RAN RRU, you need to either replace UHD or make a software wrapper which uses UHD under-the-hood (inefficient)

# 5G-capable USRPs

- **B210**
  - USB3 interface
  - ~40 MHz bandwidth (single antennas TX/RX)
  - Difficult to do MIMO above 10 MHz which is not really interesting for 5G
- **X300**
  - 1 or 2x10G PCIe, fronthaul interfaces
  - 80 MHz bandwidth 2x2 MIMO
- **N300/N310**
  - 1 or 2x10 G fronthaul interface
  - 100 MHz bandwidth 2x2 MIMO (RF impairments, high LO leakage)
  - 60 MHz bandwidth 2x2 MIMO (N300), 4x4 MIMO (N310), (without impairments)
- **N320/N321**
  - 1 or 2x10G, 1x40 G fronthaul
  - 2x2 MIMO 100 MHz without RF impairments (to be verified still)
  - 2x2 MIMO 200 MHz with RF impairments (high LO leakage, to be verified still)
- X410
  - Up to 100G fronthaul (or 10/25/40 G), in practice still 2x10G with UHD
  - 4x4 MIMO up to 400 MHz without RF impairments (uses AMD-Xilinx RFSoC RF)

# Why go beyond UHD support for experimentation

- Often we need to build custom solutions for experimentation
  - USRP + custom RF power circuits
  - Need additional (real-time) interfacing to control external circuits
  - Need to calibrate combined solutions
  - Field-deployable and even outdoor solutions
- This takes time and effort … off-the-shelf solutions exist
- All-in-one radio-units (e.g. O-RU, CPRI or AW2S eCPRI) combine a UHD-like interface with high-power and field-deployable RF solutions
  - Usually limited to certain bands
  - sometimes require more complex synchronization and management protocols to function (e.g. PTP, netconf)

slicessc

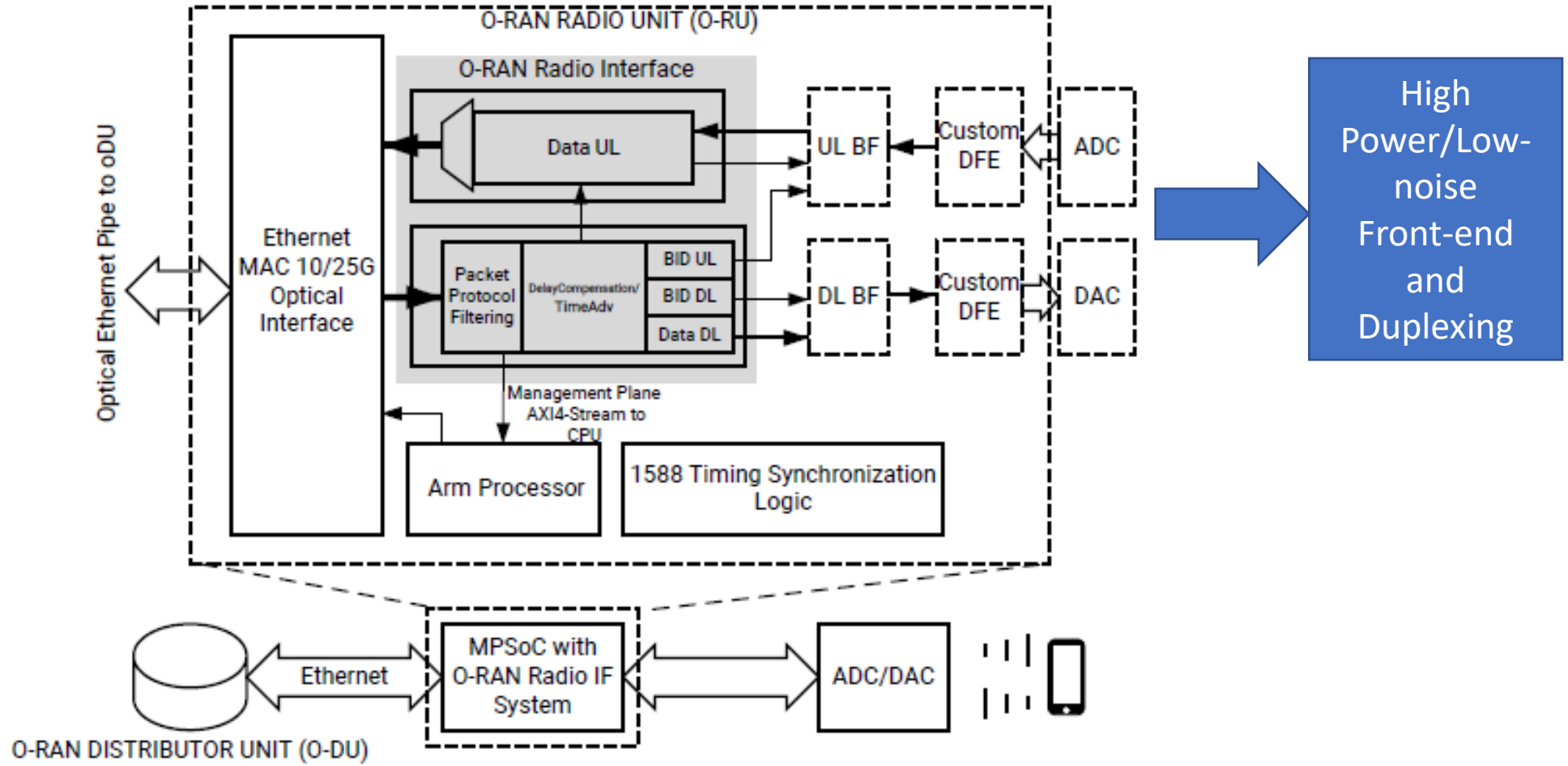
# 3GPP radio units are SDRs

Typical radio unit

- Multi-channel RF front-end, amplification (power, low-noise), up/downconversion, A/D and D/A conversion
- Digital processing (RF impairments, linearization, etc.)
  - for OFDM-tragetted frequency-domain devices, extra Fourier transforms and cyclic prefix handling (e.g. O-RAN Open Fronthaul 7.2 protocol)
- real-time interfacing with baseband processing
  - Compression/Decompression
  - Control Plane Protocol (UHD control, ETSI ORI, ORAN FHI C-plane)
  - Packetization and transport
    - Packetization = radio protocol (UHD streamer, O-RAN Open Fronthaul U-plane, proprietary over eCPRI/CPRI)
    - Transport over standard bus protocol (e.g. USB3/C, PCIe)
    - Transport over Ethernet (ECPRI, CPRI,
    - Or a combination of both (PCIe + ECPRI, PCIe + CPRI )
- Management protocol
  - Manual via telnet/ssh (e.g. USRP, AW2S)
  - Full management system via netconf (O-RAN RU)
- Synchronization protocol/method

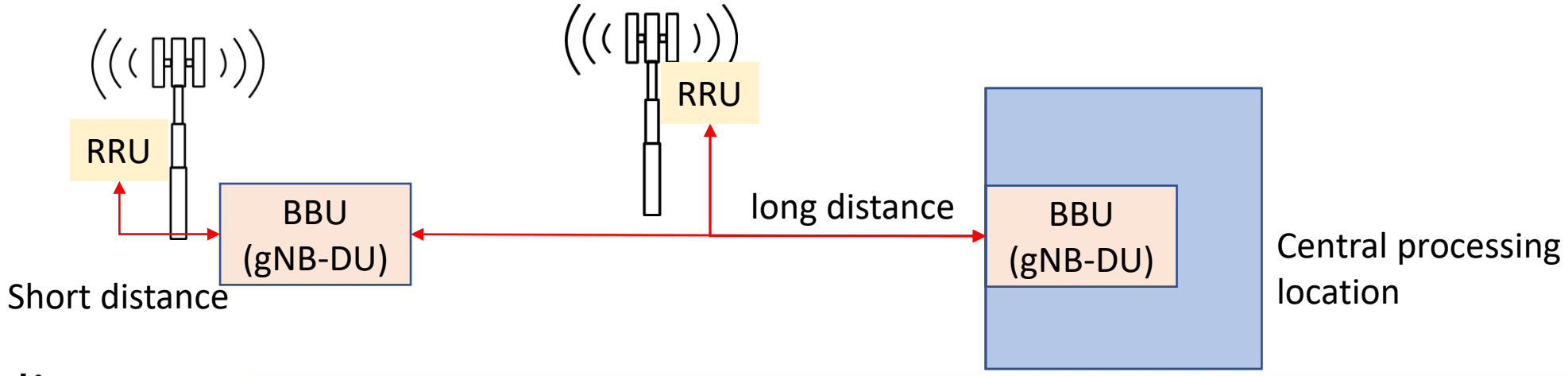
slicessc

# Example (AMD O-RAN IP for FPGA)



# Fronthaul systems and protocols

- Fronthaul = transport of radio signals over a network
- Why ?
  - To allow distance between the baseband processing and the antenna (e.g. 10-10000m)
  - To allow for statistical multiplexing of multiple radio sites sharing common fiber links
  - To share processing between radio-units and baseband units



# Fronthaul Systems and Protocols : Types

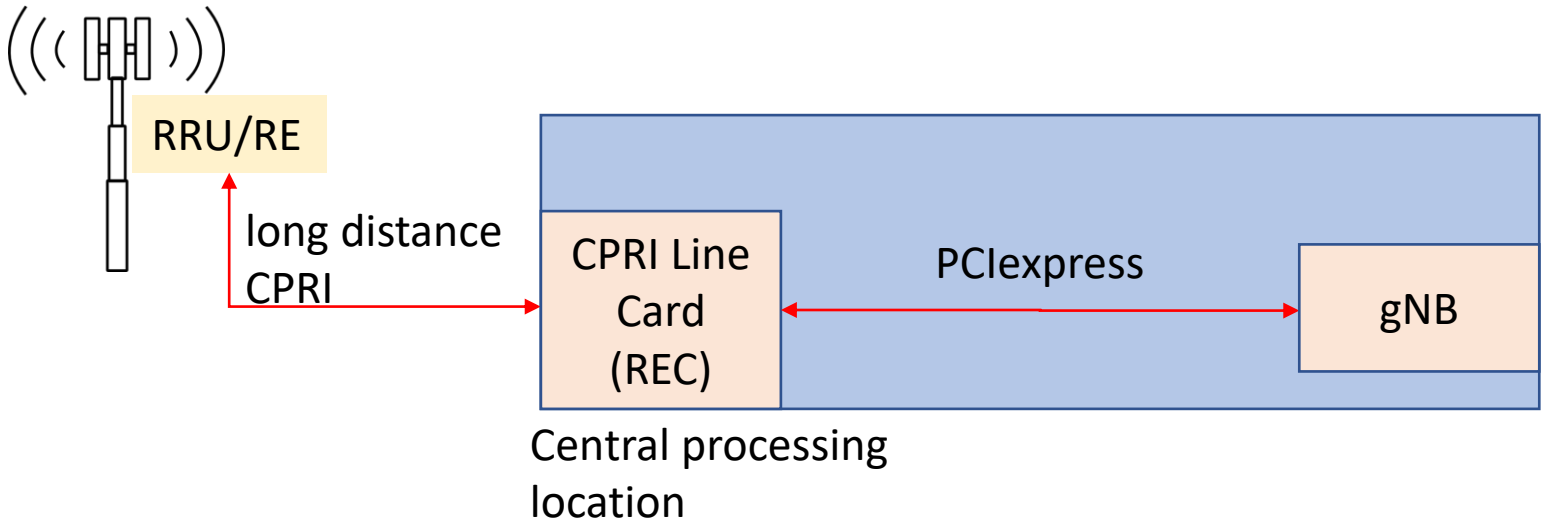
- Types
  - Analog
    - Radio over Fiber (RoF) is used by some equipment vendors today
    - Modulate an RF or intermediate-frequency (IF) signal on an optical carrier
    - Can be efficient but will always be a proprietary solution
    - Can be good to minimize latency/costs
  - Digital
    - Packetize digitized signals at some point in the transceiver chain
    - Can reuse standard Ethernet equipment for radio
    - More flexible than analog solution (reuse existing fiber infrastructure, e.g. gPON)

# Fronthaul Systems and Protocols : Types

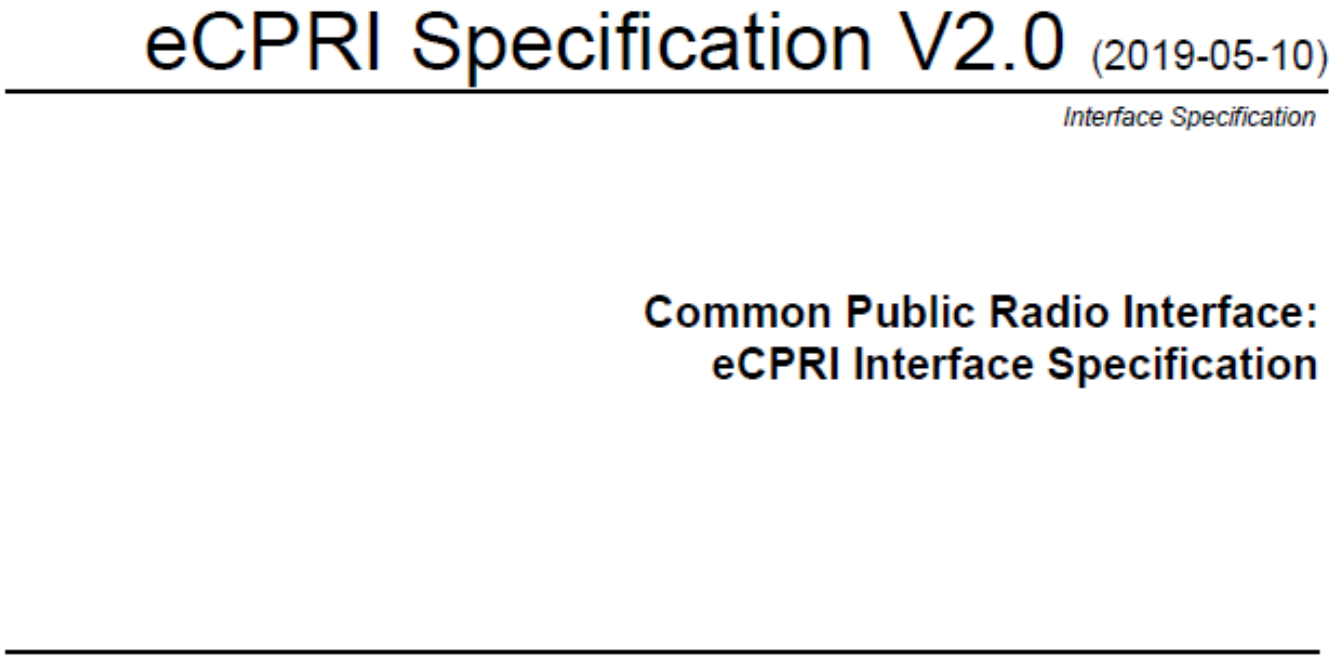
- Types of Digital Fronthaul Protocols
  - non-Ethernet based (CPRI)
    - CPRI is a partial non-3GPP specification (vendor-specific extensions)
    - Solely point-to-point
    - Good implementations in FPGA (e.g. AMD-Xilinx)
    - Usually coupled with PCIexpress on  BBU side
  - Ethernet-based (eCPRI)
    - Raw or UDP
    - Also a partial non-3GPP specification (vendor-specific extensions)
    - Can be switched
  - For eCPRI, packetization (transport) can be as simple as fixed-rate time-domain I/Q samples or much more complex variable-rate frequency-domain signals (for OFDM/SC-FDMA waveforms)

slicessc

# Typical CPRI scenario

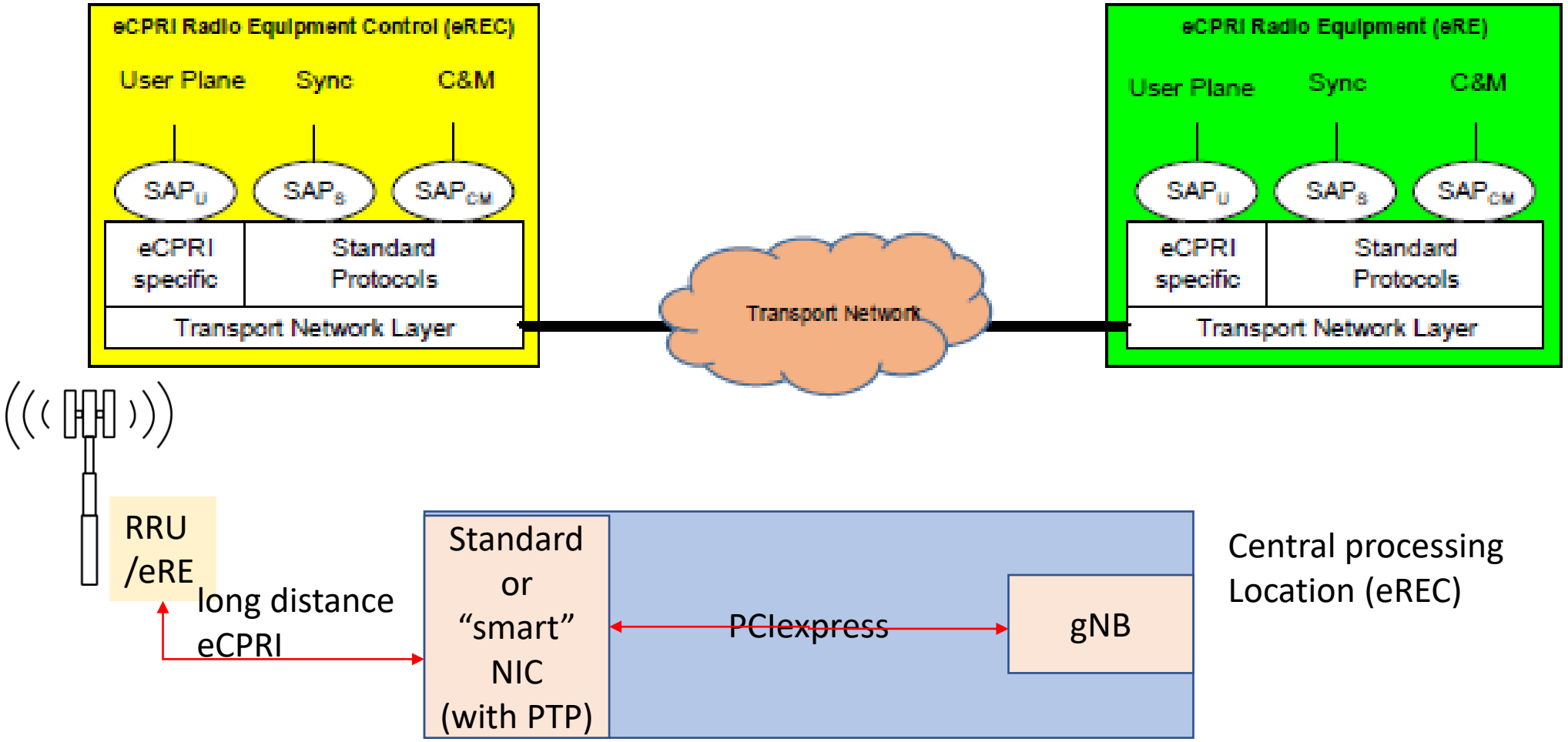


# eCPRI Specification




- Developed by Ericsson, Huawei, NEC and Nokia
- Meant to evolve P2P CPRI into a networked fronthaul
- Does not specify but implies
  - Control and management
  - Synchronization
- Clearly not meant to ensure any type of interoperability between vendors
  - Many vendor specific formats

# eCPRI


eCPRI Radio Equipment Control (eREC)
User Plane
Sync
C&M
SAP_U
SAP_S
SAP_CM
eCPRI specific
Standard Protocols
Transport Network Layer
Transport Network
eCPRI Radio Equipment (eRE)
User Plane
Sync
C&M
SAP_U
SAP_S
SAP_CM
eCPRI specific
Standard Protocols
Transport Network Layer
RRU /eRE
long distance eCPRI
Standard or "smart" NIC (with PTP)
PCIexpress
gNB
Central processing Location (eREC)

# Standard vs. “Smart NIC”

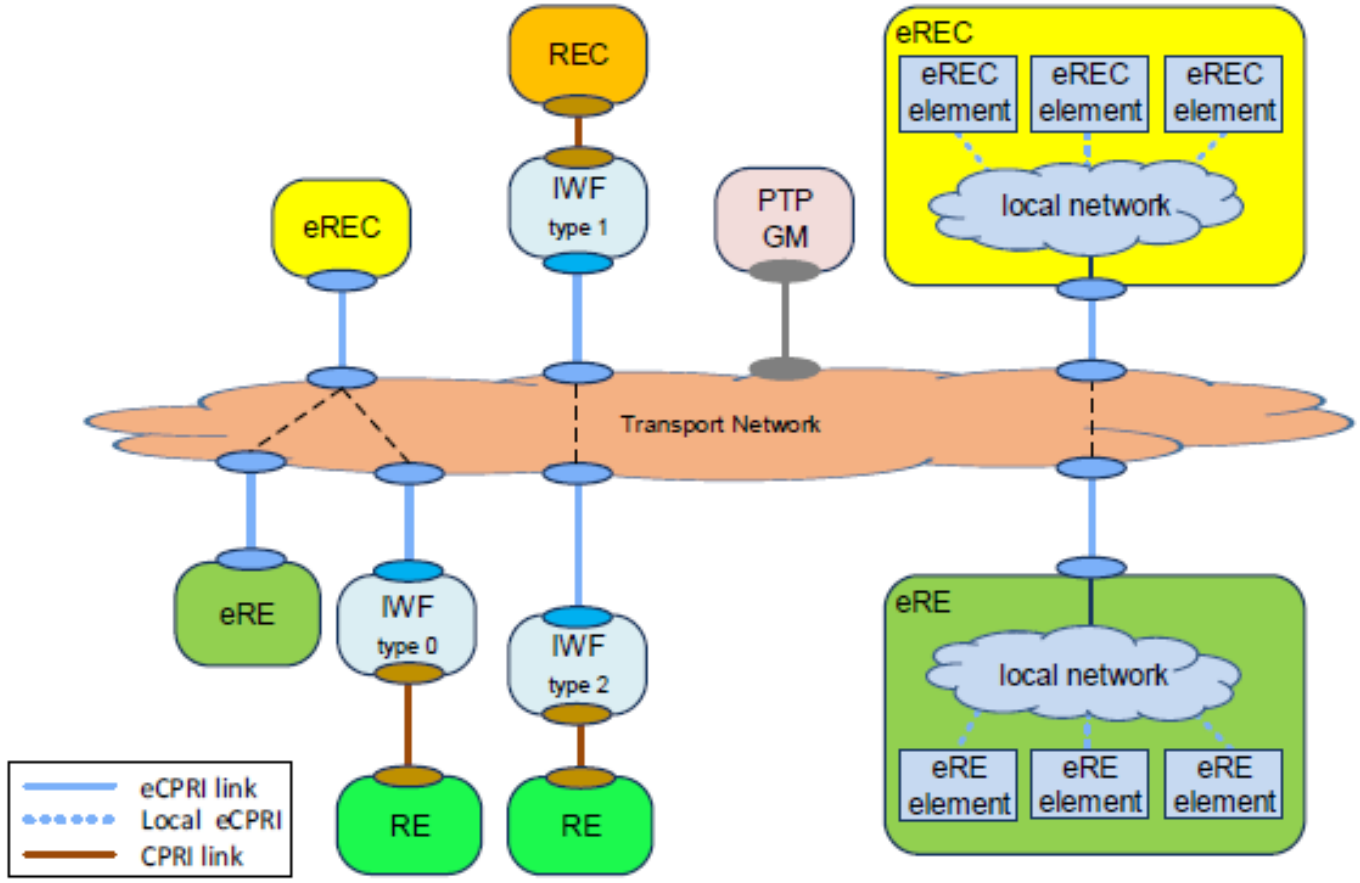
- Packet processing for eCPRI can be done in software (e.g. OAI IF5 or O-RAN FHI) and then a standard NIC is sufficient
- This requires using CPU cores to handle I/O from the NIC and to do the parsing of the protocol fields
- To be efficient various data-plane acceleration techniques are used
  - SIMD
  - DPDK (O-RAN FHI) for almost zero-copy between NIC and application
  - Recent Linux kernel extensions for minimal copy
  - Especially needed for Split-7 O-RAN solutions
- “Smart” NICs are emerging (e.g. AMD-Xilinx T1)
  - eCPRI and even O-RAN FHI packet parsing are offloaded to FPGA and DMA over PCIe is used to provide “stripped” PDUs to baseband processing unit

slicessc

# DHCP, Synchronization and Management

- DHCP is often a requirement for O-RU and eCPRI
- Sync-plane is typically PTPv2 or SyncE
- Grandmaster functionality can be co-located with central processing nodes (eREC)
- Management is open, but typically based on netconf/yang nowadays

# Current CPRI/eCPRI Networking




CPRI nodes (LTE) remain P2P but use interworking functions (IWF) to integrate with eCPRI network

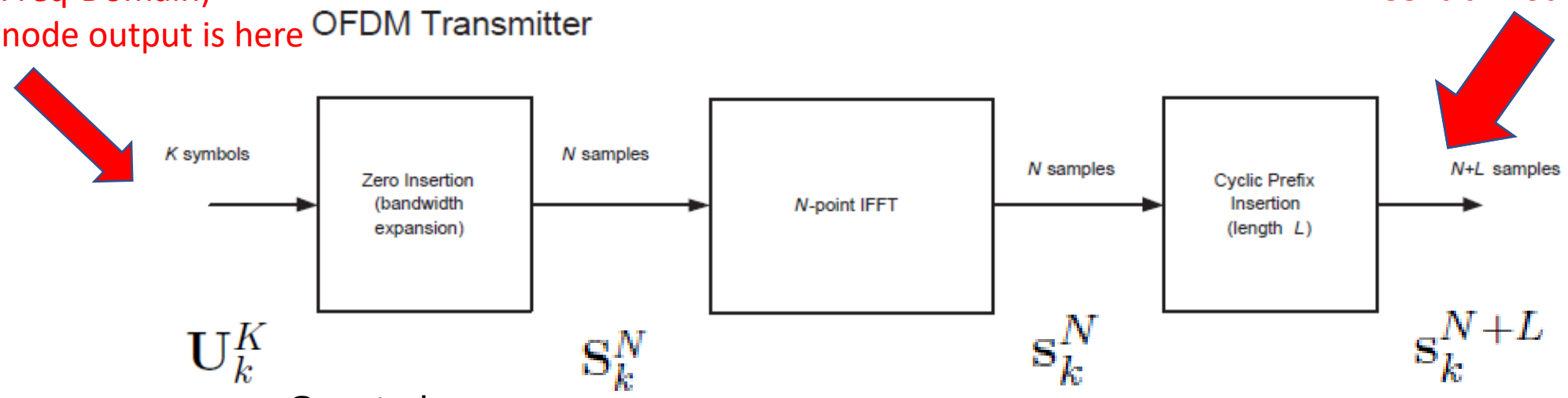
# O-RAN Functional Split

- eCPRI allows for a transport protocol which provides a functional split between the radio-unit (eRE) and central processing node (eREC)
  - Adopted in O-RAN Open Fronthaul Interface
- Usefulness is debatable
  - Can allow for sharing bandwidth/load between radio-units on common network/fiber (statistical multiplexing)
    - To be proven in large-scale deployments. Network has to be dimensioned for full-load or service could suffer
  - Frequency-domain processing allows for compression of fronthaul link
    - Can pack more radio bandwidth into a 10/20/40/100G link instead of using multiple links
  - Significant increase in protocol complexity compared to simple time-domain protocol

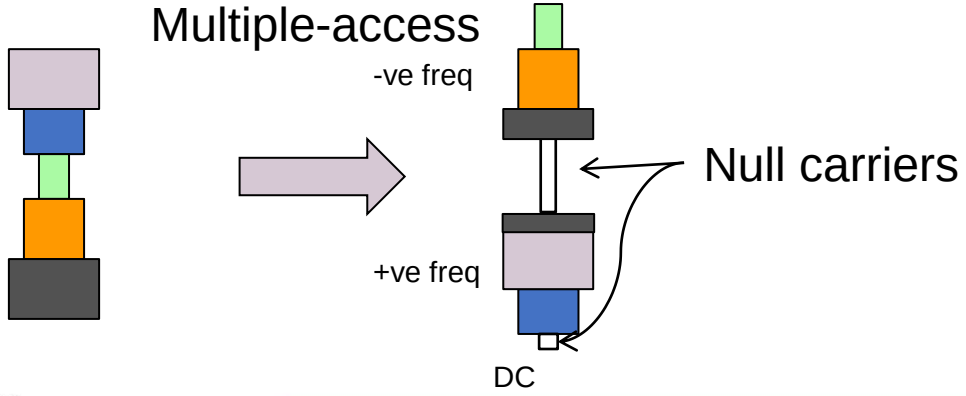
# OFDMA

Split-7 (Freq-Domain)
Central node output is here

Split-8 (Time-Domain)
Central Node output is here

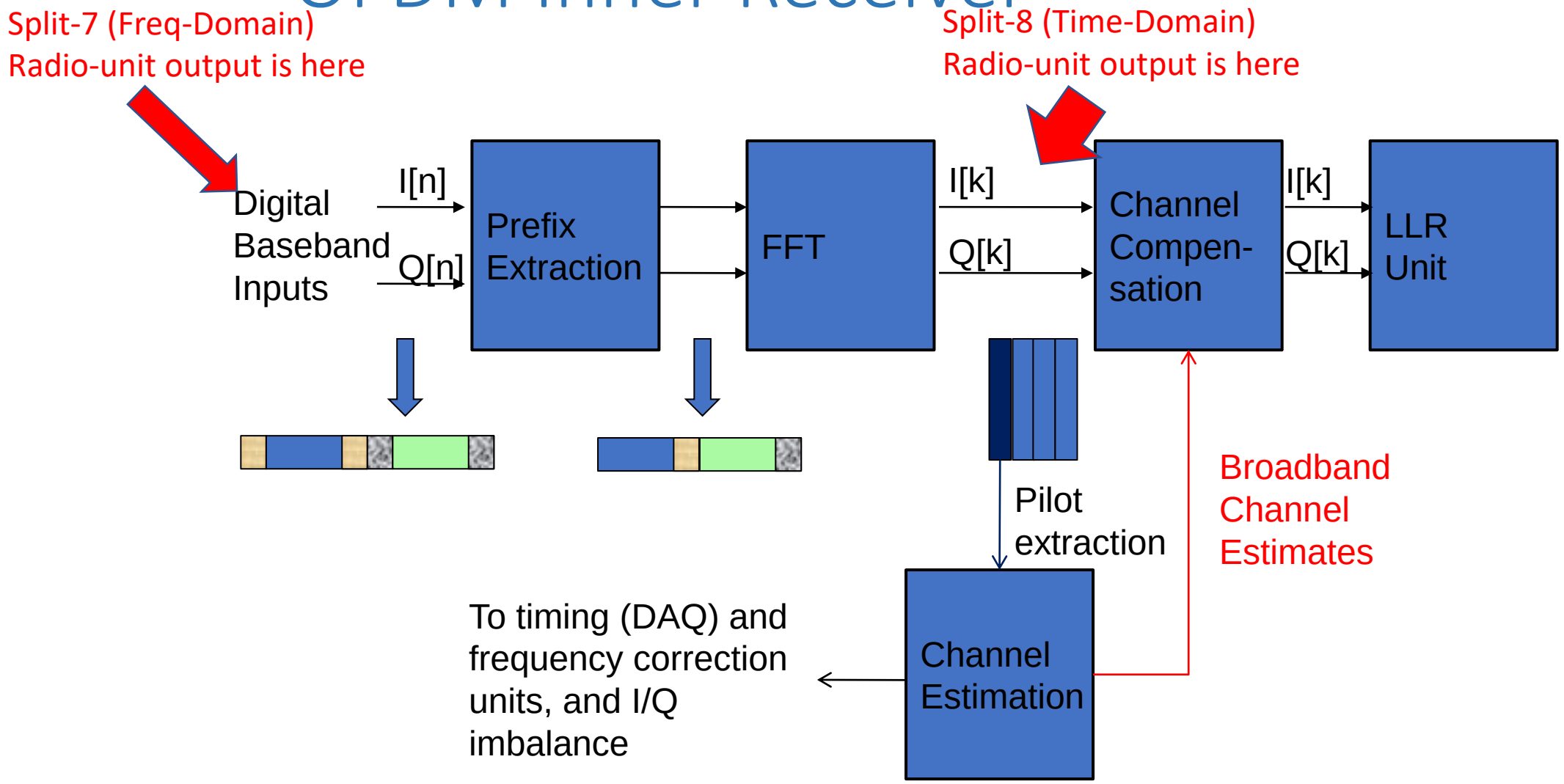



Spectral Shaping and Multiple-access



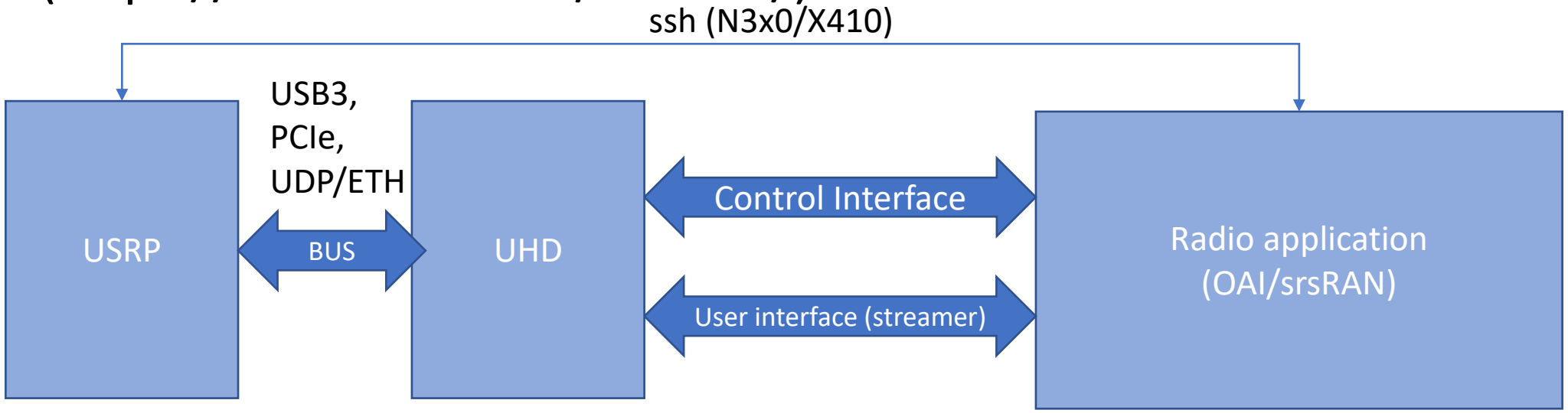

$\mathbf{U}_k^{K_i}$ Is taken from a QAM alphabet, which can be different for each $i$, and with a different amplitude (power). Each chunk can be for a different user and/or service class.

# OFDM Inner Receiver



# UHD Interfacing for 5G NR (OAI examples)

- Start with describing how UHD can be used for the 5G air-interface
- See online USRP Hardware Driver and USRP Manual (https://files.ettus.com/manual/)



# In OAI

- Between the OAI main thread there is a UHD wrapper
  - targets/ARCH/USRP/USERSPACE/LIB/usrp_lib.cpp
- Contains all the needed functions to interface with UHD
  - Set frequency
  - Set gain
  - Write to TX
  - Receive from RX
  - etc

# UHD Interfacing for 5G NR (OAI examples)

- Control interface
  - Carrier frequencies
  - Amplifier gains
  - Sampling frequencies
  - RF impairment corrections
  - Fronthaul format (float, int16, etc.)
  - Start/stop streaming
- User-plane interface (TX/RX streamers)
  - Timestamped packet transfers
  - Basically “wrapped” sendto/recvfrom socket interface, multi-threaded under-the-hood, libusb for usb3, linux (or DPDK) UDP sockets (jumbo frames) for Ethernet.

# Example snippets

```
int trx_usrp_set_freq(openair0_device *device, openair0_config_t *openair0_cfg, int dont_block)
{
  usrp_state_t *s = (usrp_state_t *)device->priv;
  pthread_t f_thread;
  printf("Setting USRP TX Freq %f, RX Freq %f, tune_offset: %f, dont_block: %d\n",
         openair0_cfg[0].tx_freq[0],openair0_cfg[0].rx_freq[0],
         openair0_cfg[0].tune_offset, dont_block);
  // spawn a thread to handle the frequency change to not block the calling thread
  if (dont_block == 1)
    pthread_create(&f_thread,NULL,freq_thread,(void *)device);
  else {
    uhd::tune_request_t tx_tune_req(openair0_cfg[0].tx_freq[0], openair0_cfg[0].tune_offset);
    uhd::tune_request_t rx_tune_req(openair0_cfg[0].rx_freq[0], openair0_cfg[0].tune_offset);
    s->usrp->set_tx_freq(tx_tune_req);
    s->usrp->set_rx_freq(rx_tune_req);
  }
  return(0);
}
```

# UHD Streaming principles

- Receive: Ask for `nsamps` in `buff`

```
n = s->rx_stream->recv(buff,nsamps,s->rx_md);
```

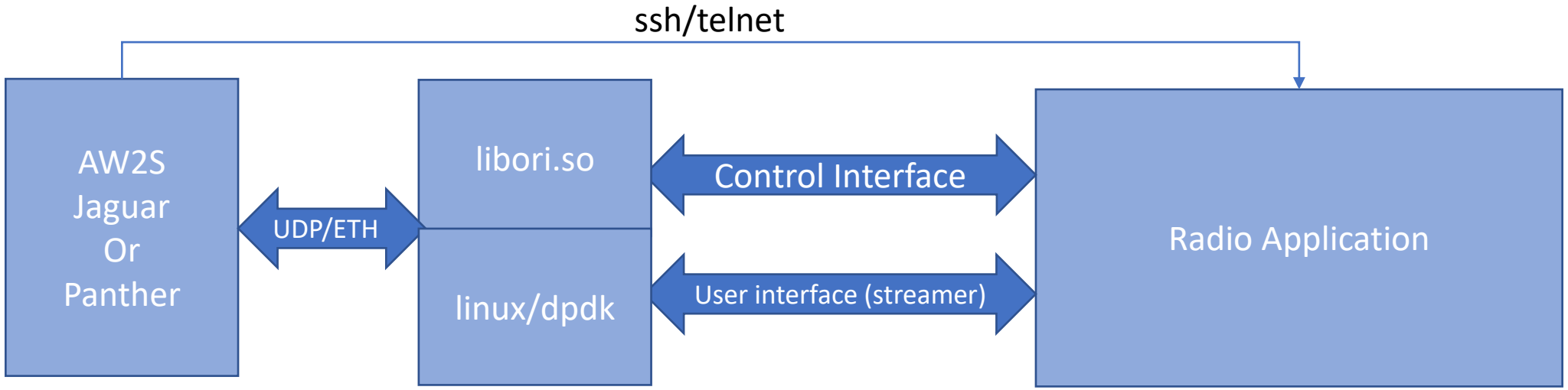
- Blocks and returns samples and timestamp when available
- Timestamp is used to derive basic frame timing information (frame,slot in 4G/5G)
- Can be used as the basic timer of the radio application
  - In main application thread we
    - Call the receive stream to get an appropriate size (e.g. 1 slot or 500us of signal)
    - Upon return, we save the timestamp, say slot $N$, and spawn RX processing (if any) for $N$ and TX processing for $N+k$
    - The resulting TX signal use the TX streamer with the appropriate timestamp

```
ret = s->tx_stream->send(buff, nsamps, s->tx_md);
```

- UHD hides the underlying socket mechanisms under the rx/tx_stream API
- Advantage: simple API
- Disadvantage : you don't really know what happens under-the-hood (number of underlying threads, CPU/OS optimizations are hidden). For instance, on Ethernet-based TX in OAI, we need to add a queing thread because tx_stream->send can block

# What about eCPRI (AW2S example)

- control plane is partial ETSI ORI implementation with 5G extensions



# ETSI ORI

- Control and Management procedures to go alongside CPRI (eCPRI now)
- Similar spirit to O-RAN but under ETSI and …
  - No specification of user-plane transport. Defaults to CPRI which is basically proprietary

# Snippets from OAI aw2sori.c

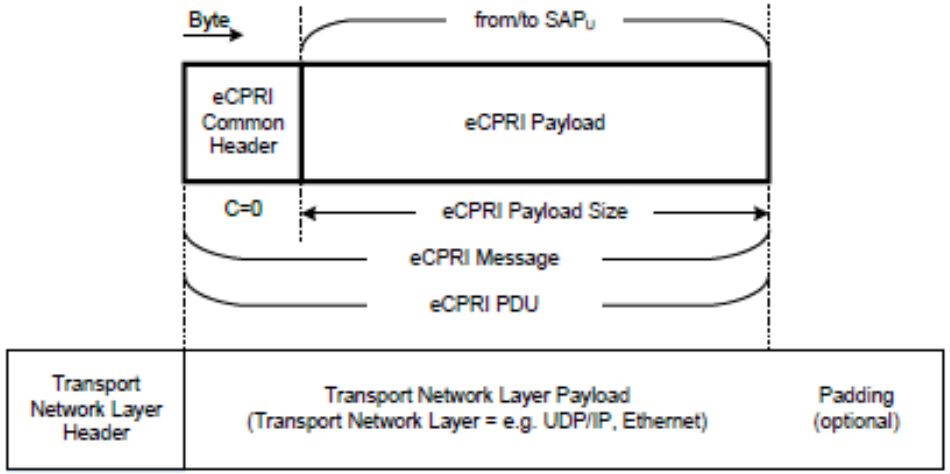
```
 else if (openair0_cfg->duplex_mode == duplex_mode_TDD && openair0_cfg->nr_flag == 1) {
    txParams.TxNRTDD.antPort = ORI_FindObject(ori, ORI_ObjectType_AntennaPort, 0, NULL);
    txParams.TxNRTDD.axcW = 1;
    txParams.TxNRTDD.axcB = 0;
    txParams.TxNRTDD.chanBW = openair0_cfg->tx_bw/100e3;
    txParams.TxNRTDD.AWS_arfcn = to_nrarfcn(openair0_cfg->nr_band,(long long int)openair0_cfg-
>tx_freq[0],openair0_cfg->nr_scs_for_raster,(uint32_t)openair0_cfg->tx_bw);
    txParams.TxNRTDD.maxTxPwr = 430-((int)openair0_cfg->tx_gain[0]*10);

    printf("AW2S: Configuring for NR TDD, NRARFCN %u, Power %d, BW %d\n",
        txParams.TxNRTDD.AWS_arfcn,txParams.TxNRTDD.maxTxPwr,txParams.TxNRTDD.chanBW);
  }
  else {
    aw2s_oricleanup(device);
    return -1;
  }
  result = ORI_ObjectCreation(ori, txTypeRef, txParams, txParamList, num_txparams, txParamResult,
&tx0, &RE_result);
  if(RE_result != ORI_Result_SUCCESS) {
     printf("ORI_ObjectCreation (txParams0.TxEUtra/NR/FDD/TDD) failed with error: %s
(%s,%s,%s,%s,%s,%s\n", ORI_Result_Print(RE_result),
```

# Streaming with ECPRI (UDP)

- Very simple:
  1. open a regular UDP socket with the RRU and provide information to RRU about gNB fronthaul networking
  2. Turn on RX and TX streaming (very similar to UHD) via ORI_ObjectStateModification on the desired RX and TX channels to “unlock” the streamers
  3. AW2S RRU will start streaming in RX via the socket using a “Vendor Specific” packet format which mimics UHD timestamp mechanism.
     - Like UHD, it is also a synchronous protocol with the RRU as a synchronization source. Multiple RRU are synchronized by GPS. PTP is not used and gNB doesn’t need PTP (like O-RAN)
  4. Use regular recv to grab each packet
- RX packets
  - Transport headers (Ethernet, UDP), ECPRI header, Application header (64-bit Timestamp, antenna port, 256 32-bit I/Q Samples (16 bit I, 16 bit Q), no compression

# ECPRI Packet format



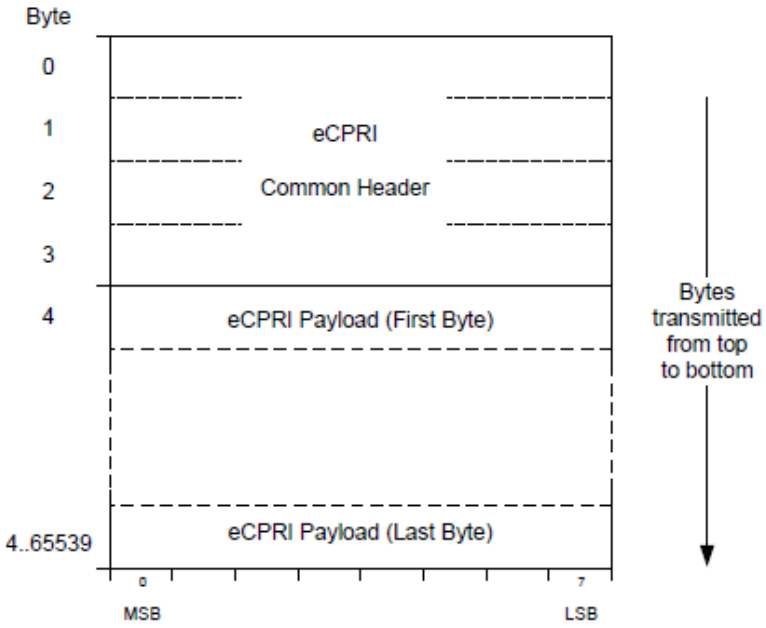




Figure 7: eCPRI message format

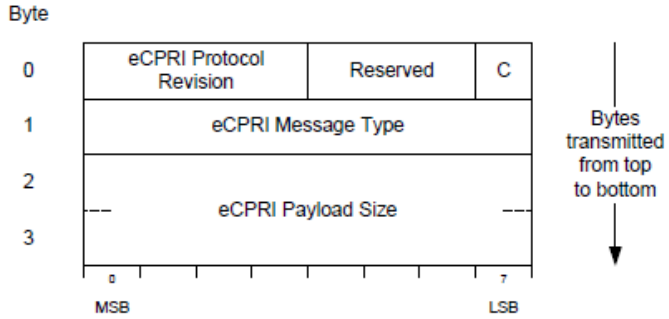



Figure 8: eCPRI Common Header format

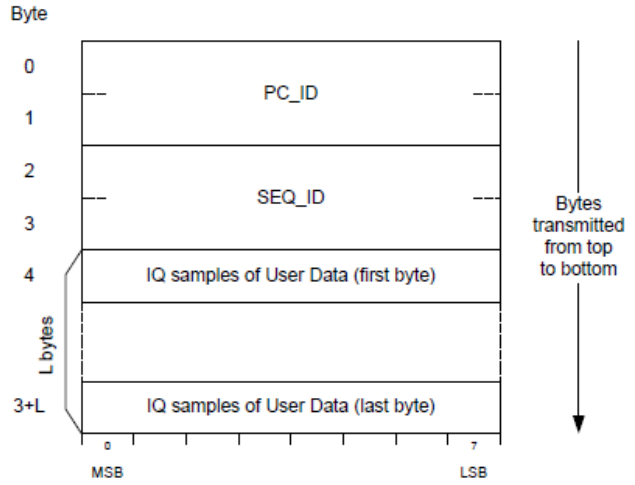



Figure 11: IQ Data Transfer message format

# OAI snippet

```
  buff2=(void*)&buff_tx[2] - APP_HEADER_SIZE_BYTES;
// ECPRI Protocol revision + reserved bits (1 byte)
  *(uint8_t *)buff2 = ECPRIREV;
   // ECPRI Message type (1 byte)
  *(uint8_t *)(buff2 + 1) = 64;
  openair0_timestamp TS = timestamp + fhstate->TS0;
  TS = (6*device->sampling_rate_ratio_d*TS)/device->sampling_rate_ratio_n;
  TS -= device->txrx_offset;
  int TSinc = (6*256*device->sampling_rate_ratio_d)/device->sampling_rate_ratio_n;
  int len=256;
  for (int offset=0;offset<nsamps;offset+=256,TS+=TSinc) {
    // OAI modified SEQ_ID (4 bytes)
    *(uint64_t *)(buff2 + 6) = TS;
    if ((offset + 256) <= nsamps) len=1024;
    else len = (nsamps-offset)<<2;
    // ECPRI Payload Size (2 bytes)
    *(uint8_t *)(buff2 + 2) = len>>8;    *(uint8_t *)(buff2 + 3) = len&0xff;
    for (int aid = 0; aid<nant; aid++) {
      // ECPRI PC_ID (2 bytes)
      *(uint16_t *)(buff2 + 4) = aid;
// … some IQ sample processing removed here
      bytes_sent = sendto(eth->sockfdd[0],buff2,
                          UDP_PACKET_SIZE_BYTES(len>>2), sendto_flag,
                          (struct sockaddr*)&eth->dest_addrd,
                          eth->addr_len);
```

ECPRI message type is “Vendor Specific” ☺

SEQ_ID contains UHD-like timestamp

PC_ID contains Antenna Port

# Some notes on non O-RAN ECPRI in OAI

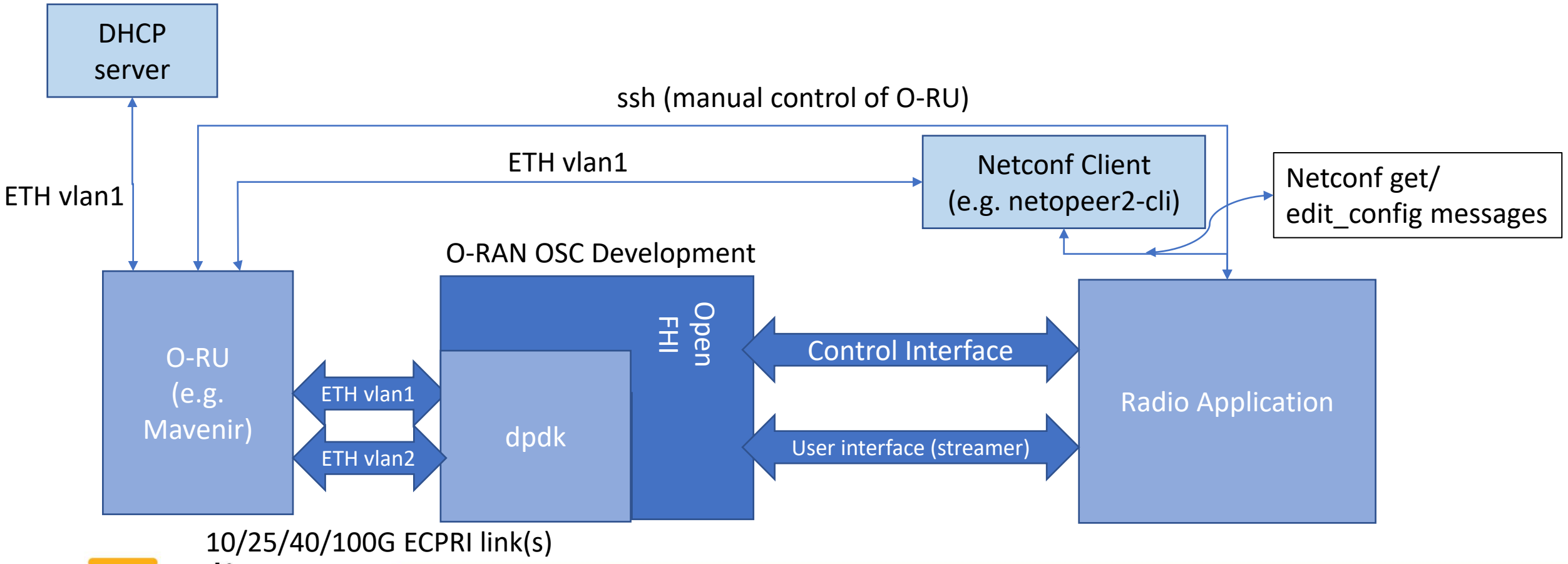
- Simple implementation of a limited set of ECPRI PDUs for AW2S RRU
  - One vendor-specific format for AW2S
- Makes use of basic linux sockets (no DPDK)
  - One RX thread (CPU core) constantly listening to a socket - `recvfrom()`
    - Linux kernel is quite efficient in the end. A single 3GHz Xeon core is sufficient for 4 antenna streaming @ 61.44 Msps (50 MHz 5G channel)
  - One TX thread (CPU core) with queue-based messages triggering `sendto()`
  - One socket per RRU (can aggregate multiple RRU on common synch)
  - So, 2 CPUs cores per socket is enough for a 4-antenna RRU. This is very likely less than what is used by UHD (hard to tell) and certainly less than DPDK

# Now O-RAN Open FHI

- O-RAN FHI is quite a bit more complex than AW2S ECPRI, but to their credit :
  - SW version is **fully open-source and distributed by O-RAN software community** (Intel implementation)
- Really optimized for Intel-based x86-64 (will not run as intended today on AMD, some routines use AVX512 and would need to be rewritten for AVX2)
- Still uses ECPRI as transport, but the application packets are more complex (control and user plane use the ECPRI protocol). Recall, control for AW2S ECPRI made use of ORI for control.
- Main differences
  - **Management protocol based on netconf and yang data models is required on compliant O-RU**
  - Raw Ethernet is used on most O-RU ,at least ones we have started using
  - Usually 2-3 different VLAN tags required (need sriov): management, control/user plane
  - DHCP is required on compliant O-RU
  - PTP is required and NIC in O-DU (gNB) needs PTP HW timestamping support
  - Need to intervene higher in the RAN L1 protocol (multiple physical channels)
    - Non-zero TX PRBs need to be sent according O-RAN protocol
    - Non-zero RX PRBs for PUSCH/PUCCH/SRS are requested from O-RU along with special PRACH PDUs (different OFDM numerology)
    - C-plane packets contain configuration information for transmitted and requests PDUs (a bit like (N)FAPI

slicessc

# High-level overview of O-RAN FHI integration



# Example configuration @ EURECOM

```
[eurecom@babylon ~]$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP mode DEFAULT group
default qlen 1000
    link/ether 34:80:0d:01:7c:78 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:11:22:33:44:66, vlan 2, tx rate 10000 (Mbps), max_tx_rate 10000Mbps,
spoof checking off, link-state auto
    vf 1 MAC 00:11:22:33:44:66, vlan 1, tx rate 10000 (Mbps), max_tx_rate 10000Mbps,
spoof checking off, link-state auto
    vf 2 MAC 00:11:22:33:44:77, vlan 1, tx rate 10000 (Mbps), max_tx_rate 10000Mbps,
spoof checking off, link-state auto
```

Virtual interfaces (sriov) created for O-RAN FHI
Seen from machine acting as gNB (O-DU)

# Example of netconf get (with netopeer2-cli)

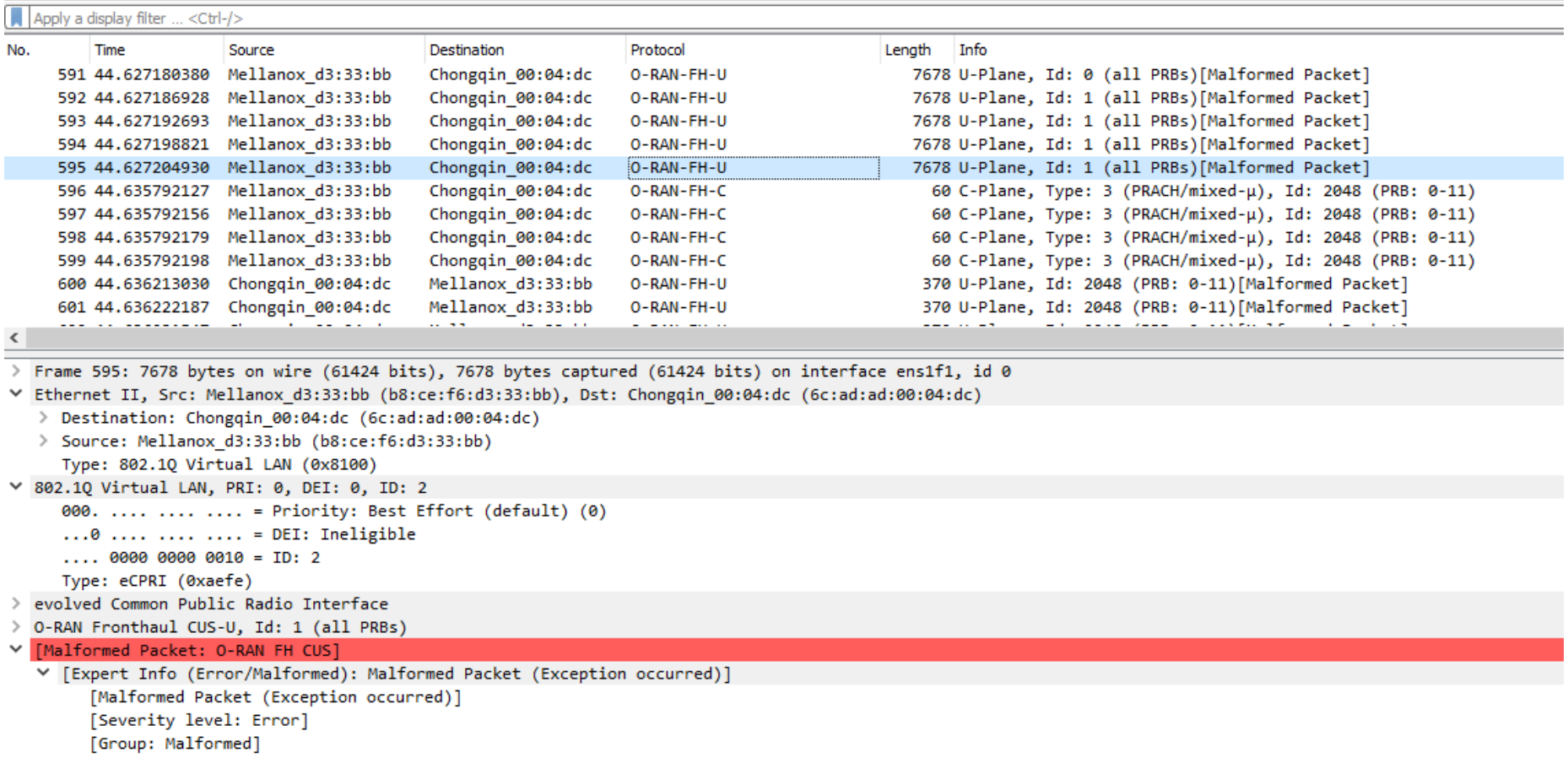
```
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <hardware xmlns="urn:ietf:params:xml:ns:yang:ietf-hardware">
    <component>
      <name>ZillnkRU</name>
      <class xmlns:o-ran-hw="urn:o-ran:hardware:1.0">o-ran-hw:O-RAN-RADIO</class>
      <description>Zillnk Radio Unit</description>
      <contains-child>FPGAMainTempSensor</contains-child>
      <contains-child>PA0TempSensor</contains-child>
      <contains-child>PA1TempSensor</contains-child>
      <contains-child>PA2TempSensor</contains-child>
      <contains-child>PA3TempSensor</contains-child>
      <hardware-rev>1.00</hardware-rev>
      <firmware-rev>1.00</firmware-rev>
      <software-rev>1.00</software-rev>
      <serial-num>Z12260120213500033l</serial-num>
      <mfg-name>Zillnk</mfg-name>
      <model-name>Zillnk O-RU</model-name>
      <is-fru>true</is-fru>
      <uuid>11111111-1111-1111-1111-111111111111</uuid>
      <state>
        <state-last-changed>1970-01-01T01:00:33+01:00</state-last-changed>
        <admin-state>unlocked</admin-state>
        <oper-state>enabled</oper-state>
```

Piece of M-plane capture from **Mavenir O-RU** (manufacturer **Zillnk**)

M-Plane required to

- Get O-RU capabilities
- Edit configuration
- Change state of O-RU (inactive to running)
- Montior events/faults

# Example wireshark trace of O-RAN Open FHI




Capture from Foxconn O-RU

# Conclusions

- Provided an overview of current fronthaul interfaces and their integration in OAI codebase
  - UHD
  - ECPRI
  - ECPRI + ORAN Open FHI
- Experimental networks (SLICES-RI, PAWR evolutions) will go beyond UHD in some deployment scenarios
- Even USRPs may soon start to support O-RAN Open FHI for experimentation with O-RAN based protocols (X410 devices)

Scientific Large-scale Infrastructure
for Computing Communication
Experimental Studies
**Starting Communities**

# Thank you

slicessc

www.slices-sc.eu