Overview of Day 1

- **Some background on OpenAirInterface**
- **OAI RAN (openairinterface5g) Software Architecture**
  - Functional entities
    - Focus today on L1 procedures for eNodeB
  - Real-time-scheduling
    - Example with USRP RF
- **SIMD Processing and Optimizations used in OAI**

BACKGROUND ON OAI

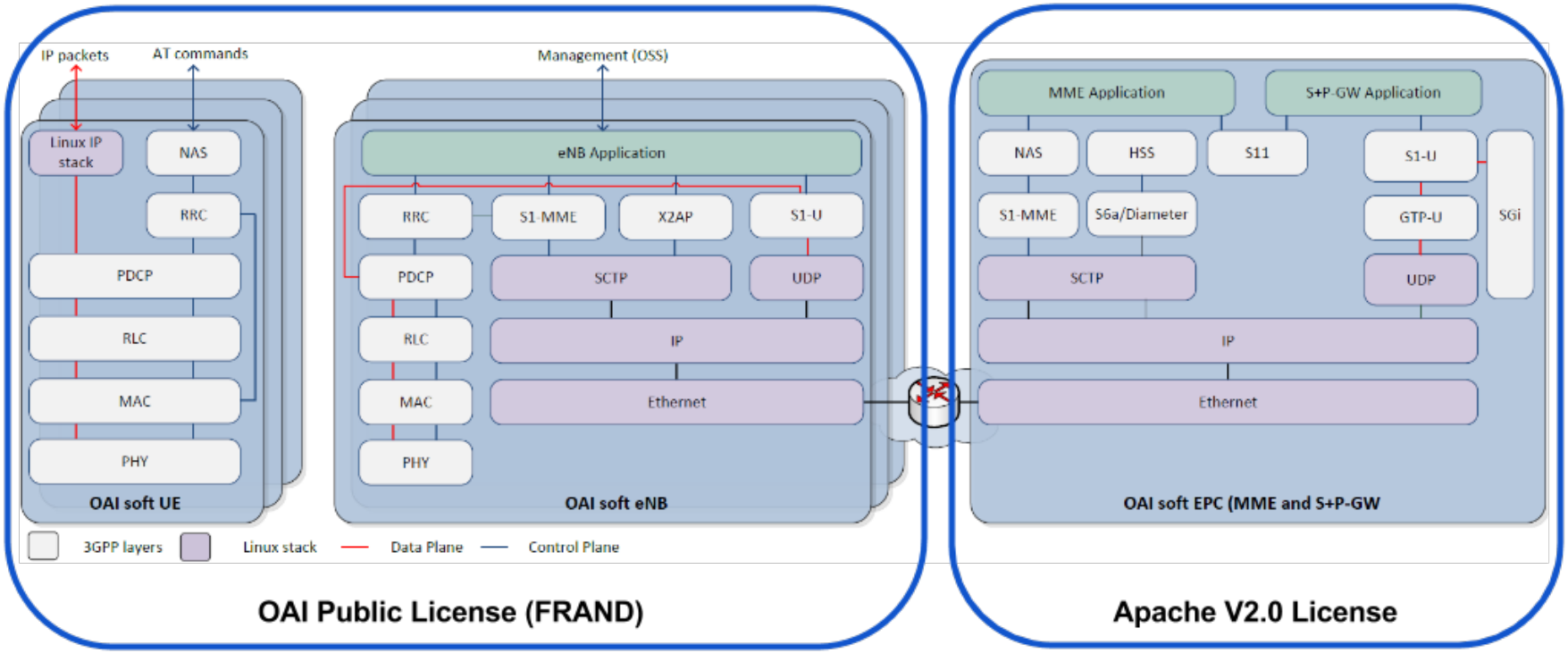
A bit about R. Knopp

- **From Montreal, Canada**
- **B.Eng (Hons) (1992), M. Eng. McGill University (1993), Dr. Sc. (1997) EPFL**
  - Work in Digital Communications, Wireless systems and protocols, real-time prototyping methods
- **Currently Professor @ EURECOM, Sophia Antipolis Technopole, France (near the city of Nice)**
  - Research center in Communication Systems, Security and Data Science
  - 4-5th year Engineering curriculum(~120 students) + PhD (70 students)

Commoditization of 3GPP Radio Systems and Open-Source : OAI Software Alliance

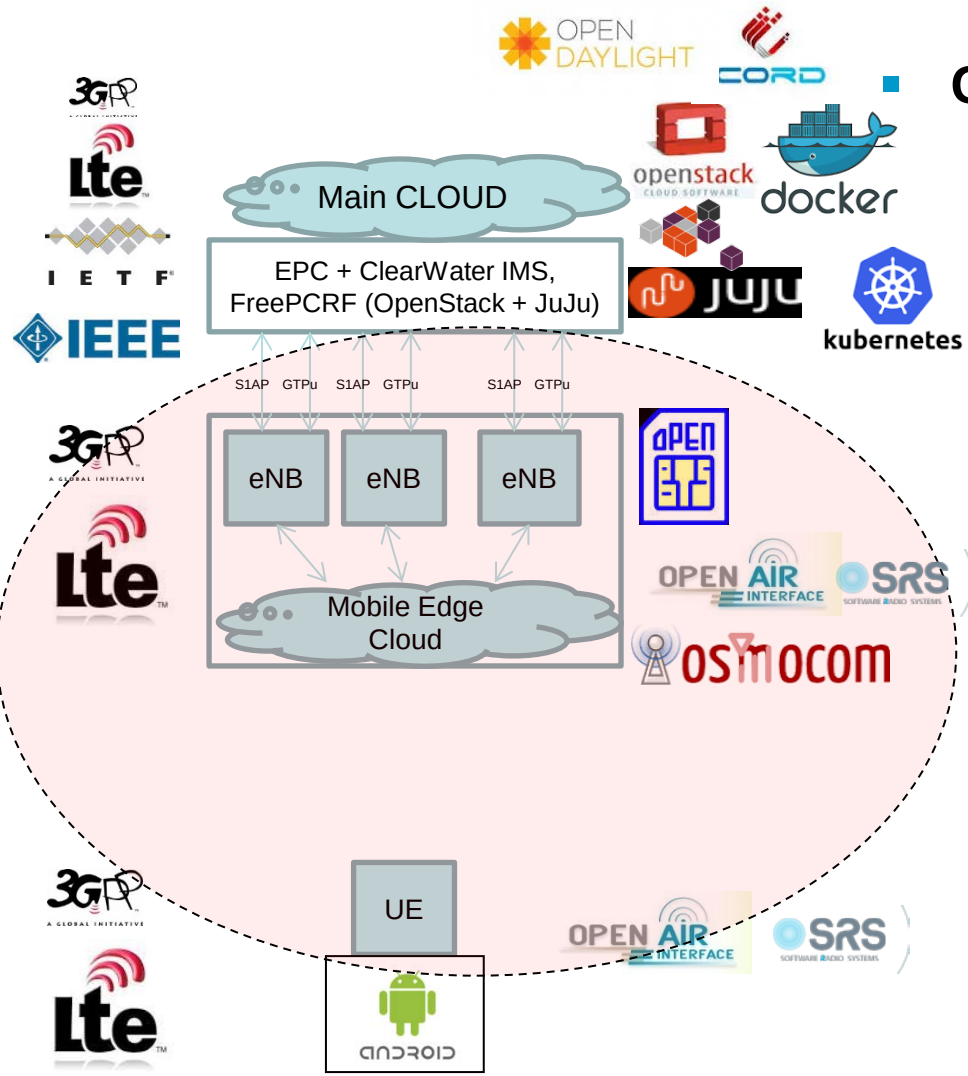
- **Today it is feasible to put a fully-compliant 4G eNodeB and EPC in a commodity x86 or ARM-based computer (or data center for a pool of eNodeBs)**
  - Emergence of “radio”-hackers in addition to commercial vendors
  - OAI Alliance
    - launched in 2014 as a “Fonds de Dotation”
    - 3GPP strategic members in 2015-2017 (Orange, TCL-Alcatel, Ercom, Nokia)
    - Many associate members (Cisco, B-COM, INRIA, IMT, TNO, III, Rutgers WINLAB, U. Washington, BUPT, IITH etc.)
- **Coupling this with an open-source community makes for a very disruptive technology for the onset of 5G**
  - What we’re building
    - Community of individual developers, academics and major industrials embracing open-source for 5G
  - What we hope to become
    - A strong voice and maybe a game-changer in the 3GPP world
      - ☞ Real impact from “the little guys” on 3GPP systems

OSA Components



- **FRAND License allows committing software with patent rights into OSA and still keep licensing rights -> Inline with 3GPP fair use licensing policy**
- **We aim to work closely with ETSI on implications of open-source for licensing/certification**
- **Future 5G Core Network developed within eNB/UE repository will inherit FRAND license**

Bringing open-source into the RAN



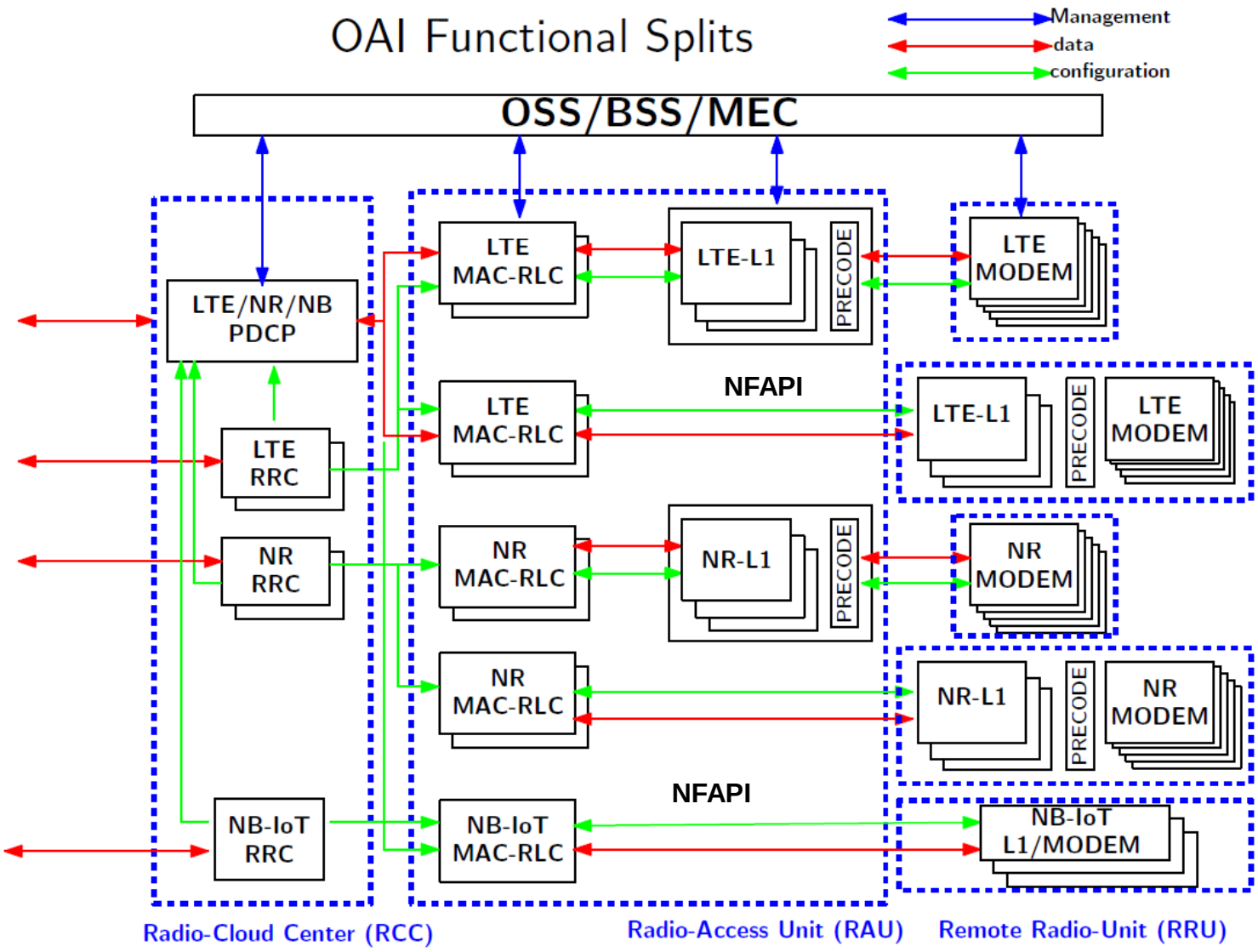
- **Challenges for 4.5G/5G**
  - FRAND License for open-source
    - Allow 3GPP members to contribute to open-source and still perceive royalties
    - Compatible with
      - academic/research/prototyping use
      - commercial use in devices, data centers and testing equipment
    - Approval by Nokia, Orange, TCL and Ericsson (observer)
  - Community coexistence/synergy with standardization process
    - Use of open-source in prototyping phase of 5G
    - Open-source community following 3GPP process
    - Community representation in 3GPP via OSA

OAI Offering

- **eNodeB software for x86/ARM**
  - 4G Implementation
    - Rel 10 (with many missing and incomplete features under development by the community)
    - Some extensions to Rel-14
      - ☞ LTE-M, NB-IoT under dev
      - ☞ Sidelink eNodeB procedures under dev
  - 5G Rel-15 underway in OAI community
- **UE Software for x86/ARM**
  - 4G Implementation
    - Rel 10 (with many missing and incomplete features under development by the community)
    - Some extensions to Rel-14
      - ☞ Sidelink UE procedures under dev
- **EPC Software**
  - Rel10 3GPP MME/HSS/S-PGW
  - Rel14 extensions under development
  - Transition to 5G core service-based architecture

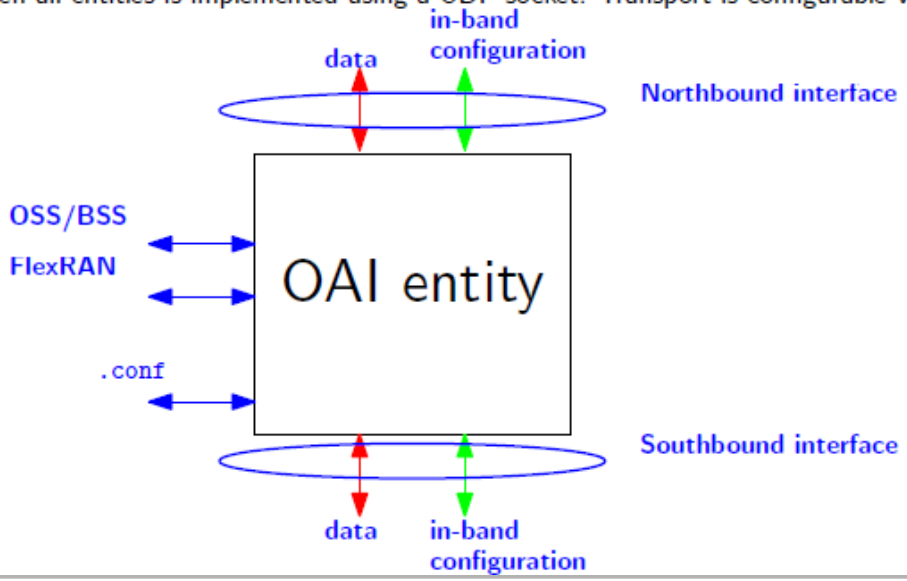
RAN ARCHITECTURE

Current vRAN Roadmap in OAI



Current Functional Entities

- OAI currently implements the following entities in openairinterface5g
  - LTE-MODEM (eNB 36.211 OFDM modulation/demodulation)
  - LTE-L1 (eNB 36.211/212/213)
  - LTE-MACRLC (eNB 36.321/322)
  - LTE-PDCP (eNB PDCP/GTPU 36.323)
  - LTE-RRC (eNB RRC/SCTP 36.331)
- Each entity comprises
  - a northbound interface (backhaul/midhaul/fronthaul and configuration)
  - a southbound interface (midaul/fronthaul and configuration)
  - one or two management interfaces
  - Three computing nodes
    - **Radio Cloud Center (RCC)** : multiple RRC/PDCP entities
    - **Radio-Access Unit (RAU)**: multiple MACRLC entities with medium-latency midhaul and L1 entities with low-latency fronthaul.
    - **Remote Radio-Unit (RRU)**: Equipment at radio site. Varying degrees of processing elements depending on fronthaul/midhaul interface.
- Each entity has a configuration which is a local file or received via the management interface
- default interface between all entities is implemented using a UDP socket. Transport is configurable via a dynamically-loadable networking device



NGFI fronthaul splits today in OAI

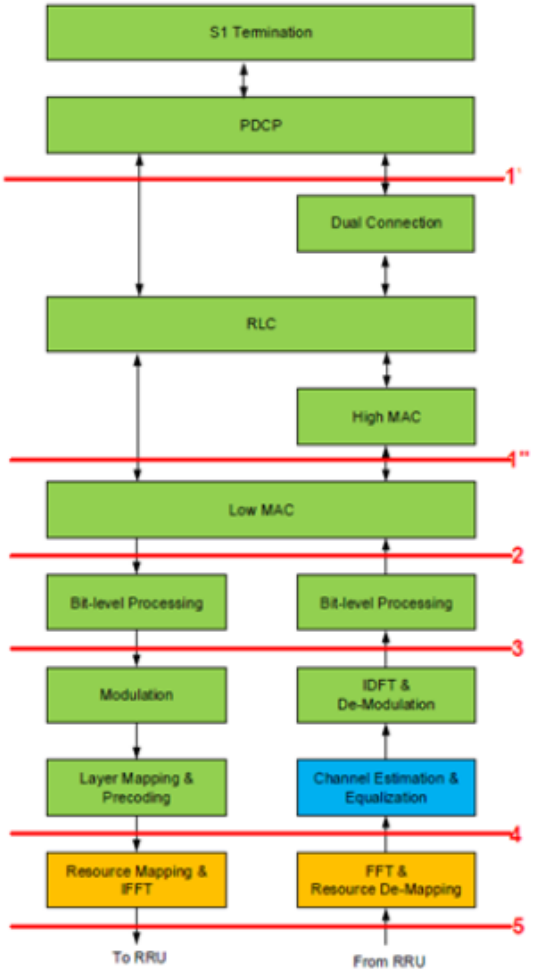
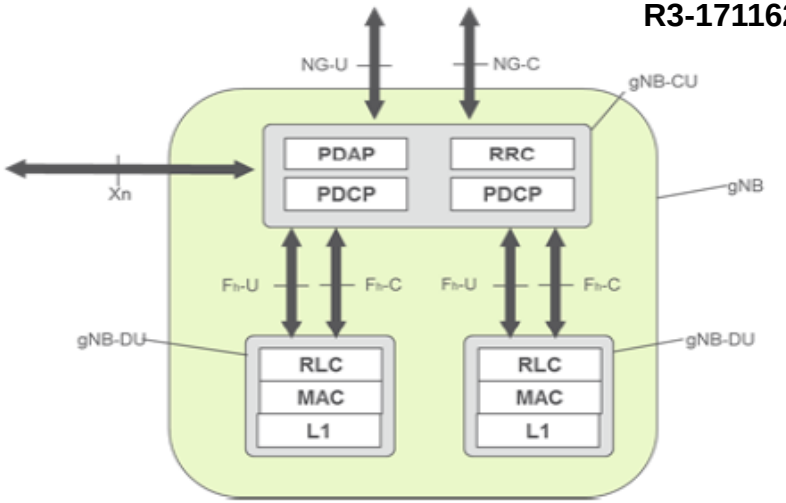


Figure 3-1: Division Plans for the RCC-RRS Interface

- Current OAI implementation (RRU/RCC) supports either
  - IF5 time-domain fronthaul (> 1 GbE required)
  - IF4.5 split (FFTs) (280 Mbit/s/antenna port fronthaul 20 MHz carrier) per carrier/sector
  - Soon IF2 (NFAPI)
  - IF1 for PDCP/RRC soon (3GPP Fh-C/Fh-U)

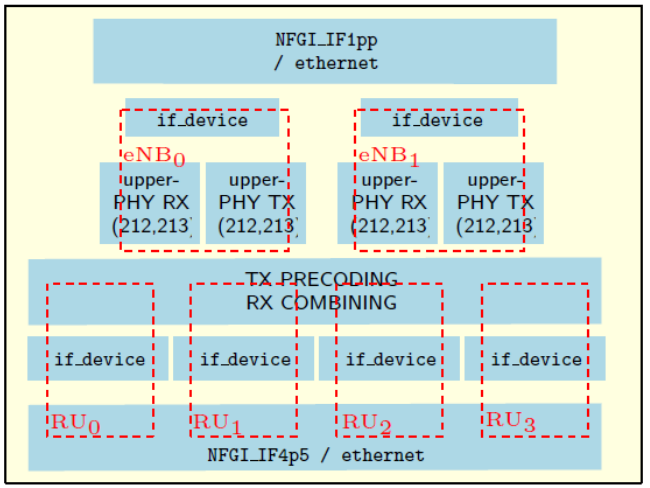
NR study item
R3-171162



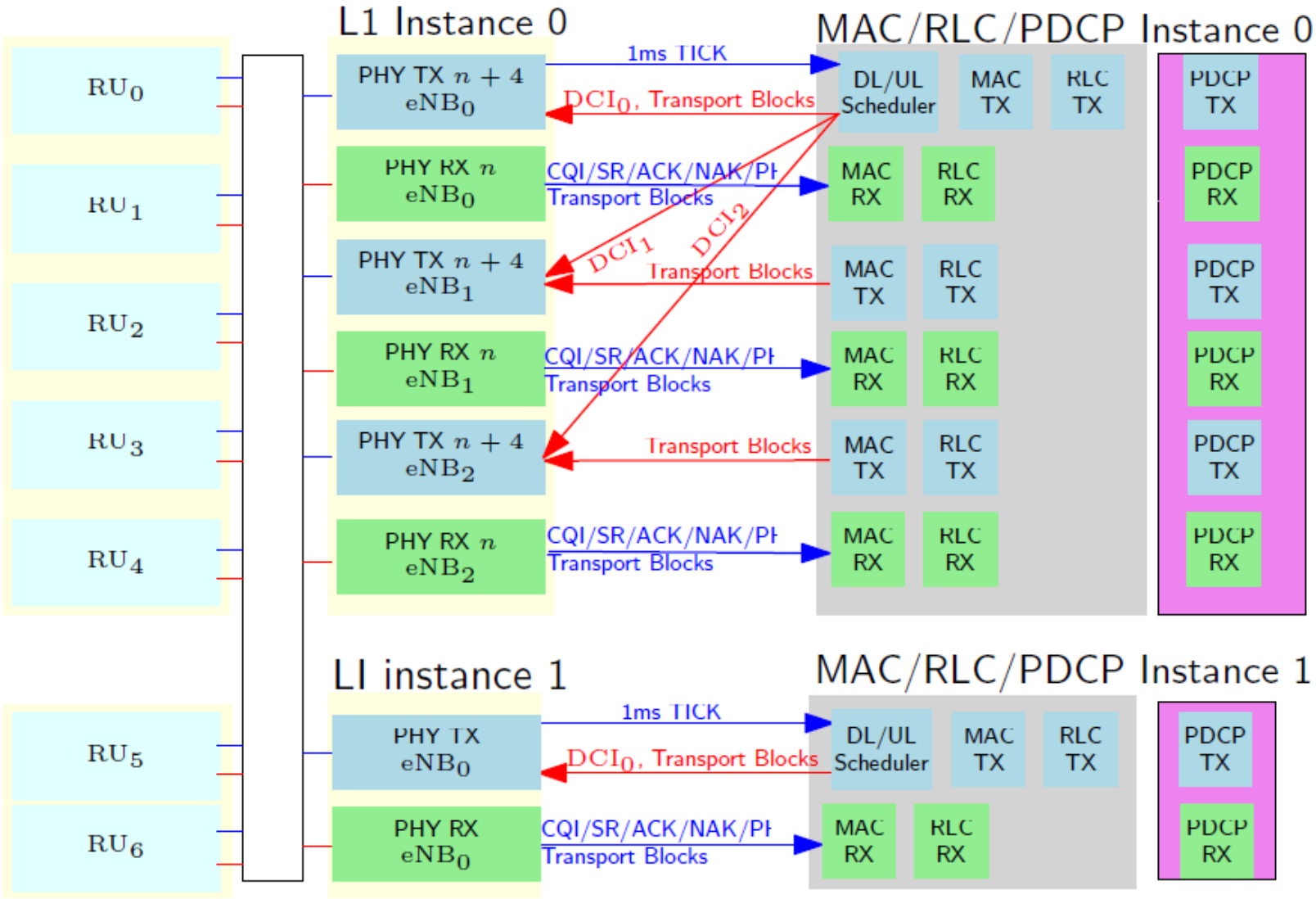
RRU/RAU

- **OAI current lower-layer functional split**
  - a network of radio units (L1-low)
  - a precoding function and switching function
  - Regular (virtualized) eNB functions

Example: RAU with NGFI_IF1pp xhaul (MAC/PHY split) northbound, NGFI_IF4p5 fronthaul southbound, 2 vCell logical interfaces (2 L1/L2 instances, or 1 L2 instance and 2 CCs), 4 RRUs with NGFI_IF4p5



OAI RAN (CU-DU) Architecture



RU and L1 Instances

- *Radio Unit* (RU) is
  - an entity managing a set of **physical** antennas. It can have a *local RF unit* or *remote RF unit*
  - performs precoding of multiple eNB TX streams and OFDM modulation (TX) and demodulation (RX) (part of 36.211)
- *L1 Instance* (indexed by `Mod_id`, or `enb_mod_id`) is a separate set of threads and contexts for the eNB/gNB procedures. There is one MAC/RLC entity associated to all :1 component carriers.
- *L1 Component Carrier* (indexed by `CC_id`) is
  - a software entity managing the L1 procedures (36.213,36.212,36.211) and can act on
    - sectored antenna component
    - Rel10+ component carrier
    - virtual cell for DAS or Massive-MIMO array
  - each L1 instance is managed by one or two threads which operate on a subframe (TX and RX) and can have a *local RU* or *remote RU*
  - if a remote radio unit the eNB performs the 36.213 specifications only (HARQ, etc.) and connects to the remainder via the IF2 midhaul interface.

RU and L1 Instances

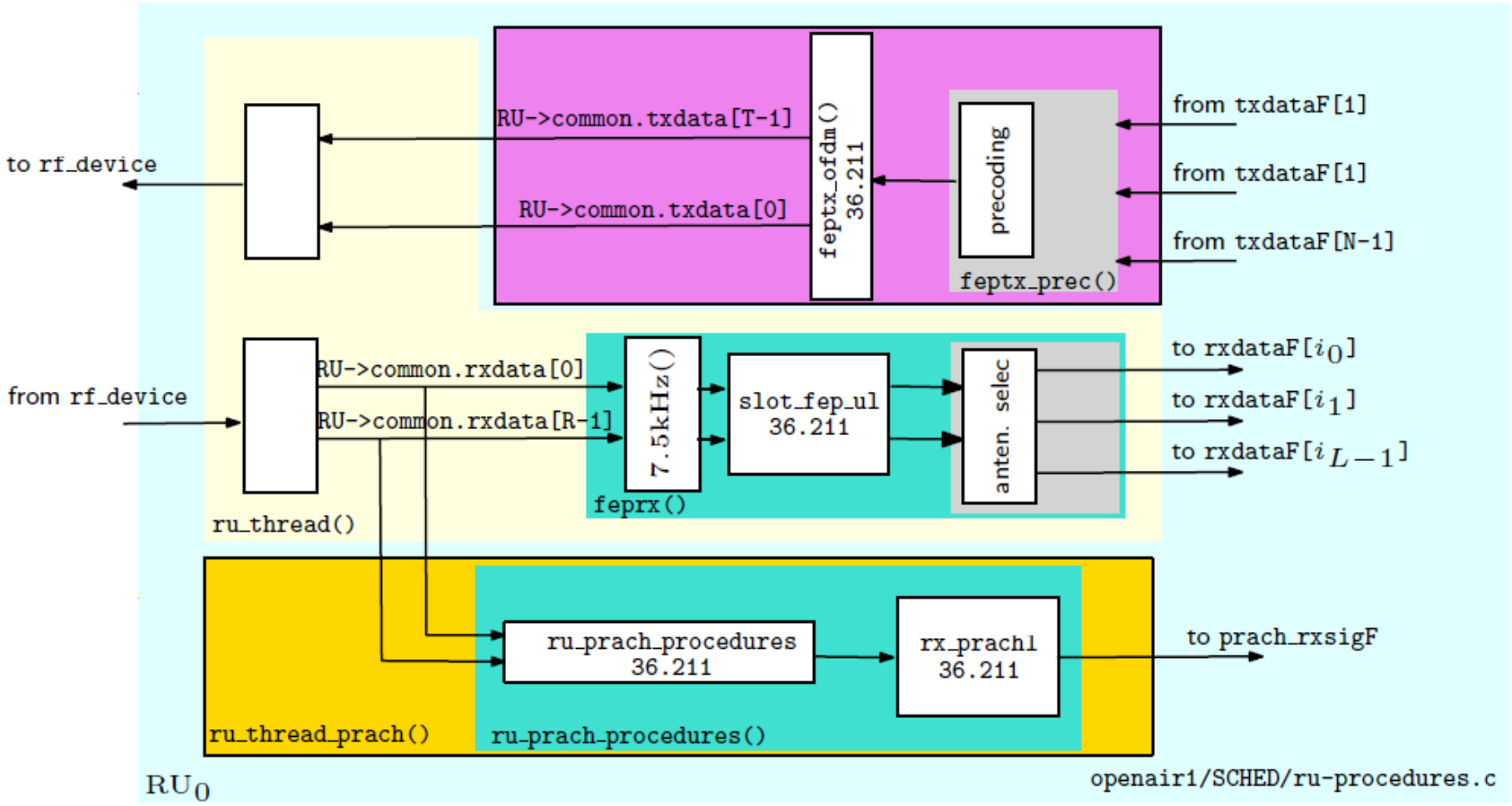
- RU may have both an `if_device` for fronthaul and an `rf_device` for interconnection with a local RF unit
- if the `rf_device` is absent, it must have a southbound fronthaul interface (either IF5 or IF4p5) depending on the local processing of the remote RU
- if the `if_device` is absent, it must have a southbound RF interface and `rf_device`.
- three types of L1 processing are performed by the RU
  - subset of common L1 procedures from 36.211 specifications
  - fronthaul compression/decompression
  - framing
- on TX
  - A-law compression for (NGFI_RAU_IF4p5, NGFI_RAU_IF5)
  - A-law decompression (for NGFI_RRU_IF4p5 and NGFI_RRU_IF5)
  - OFDM modulation and cyclic prefix insertion (for NGFI_RRU_IF4p5,NGFI_RAU_IF5,3GPP_eNodeB_BBU,3GPP_eNodeB)
  - Precoding (for NGFI_RAU_IF5, NGFI_RAU_IF4p5,3GPP_eNodeB_BBU,3GPP_eNodeB)

RU and L1 Instances

- on RX
  - A-law compression for (NGFI_RRU_IF4p5, NGFI_RRU_IF5)
  - A-law decompression (for NGFI_RAU_IF4p5 and 3GPP_eNodeB_BBU)
  - cyclic prefix removal, frequency-shifting, OFDM demodulation, PRACH DFT (for NGFI_RRU_IF4p5, NGFI_RAU_IF5, 3GPP_eNodeB_BBU, 3GPP_eNodeB)
- On TX path
  - L1 instances/component carriers operate on a set of logical antenna ports (0-3 for TM1-6, 4 for eMBMS, 5 for TM7, 6 for positioning, 7-8 for TM8, etc.)
  - each L1 instance has a list of RUs and the logical antenna ports are mapped to the physical antennas attached to the RUs via the precoding function

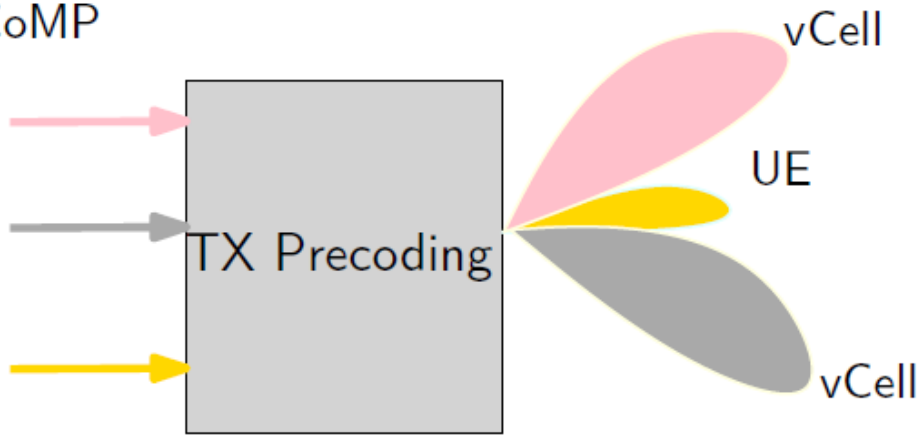
RADIO SEGMENT (RU)

RU Procedures (monolithic eNB/gNB)

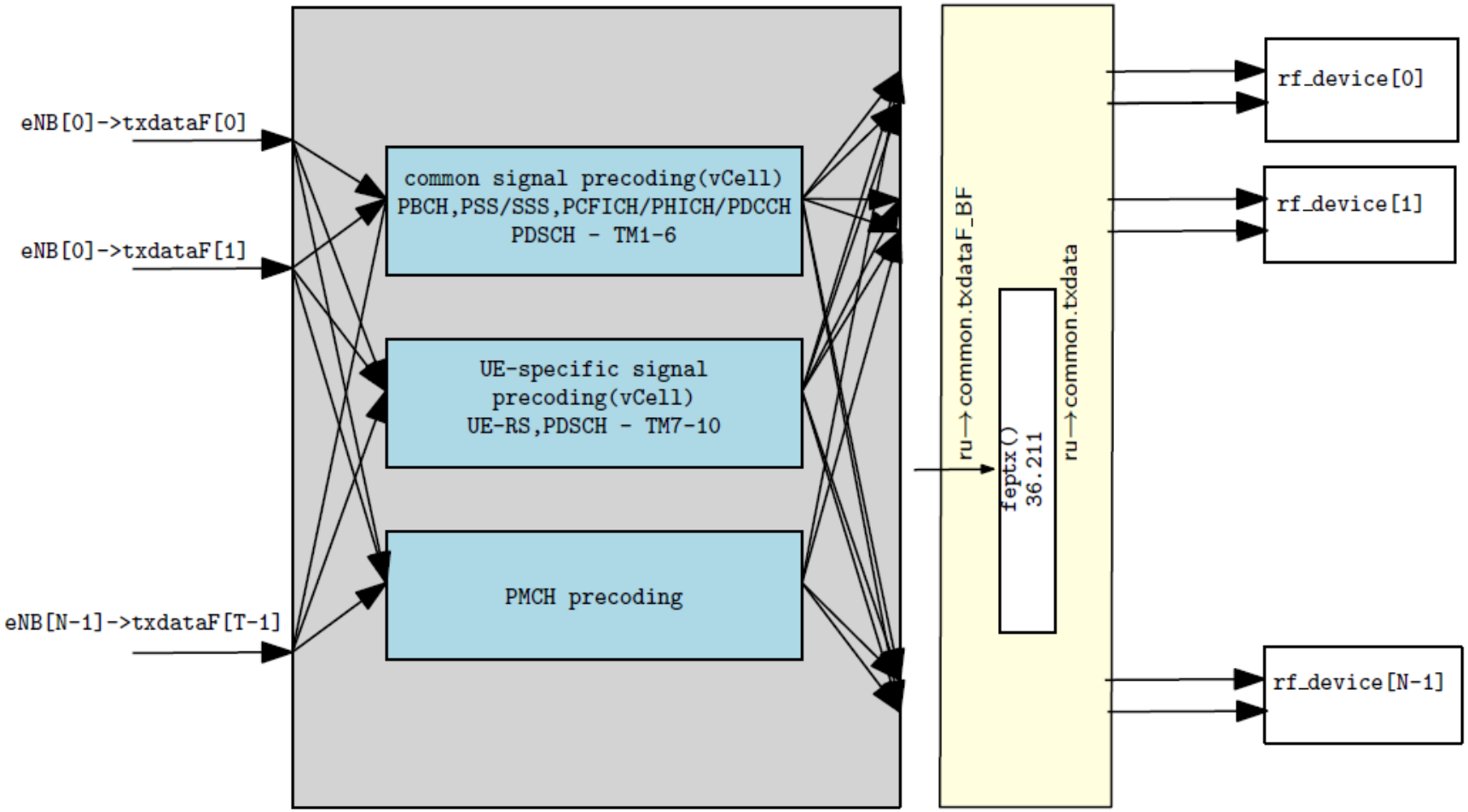


TX Precoding

- Spatio-temporal filtering for muli-cell (vCell) and multi-user transmission. Input and output are frequency-domain signals.
- can be applied to Rel-10/11/12/13 physical channels and Rel-8 common channels
  - UE-specific precoding (TM7-10)
  - vCell-specific precoding (PDCCH + TM1-6) for groups of UEs
  - PMCH vCells
- Precoding applicable to
  1. indoor DAS
  2. outdoor co-localized arrays (e,g, Massive-MIMO)
  3. outdoor CoMP

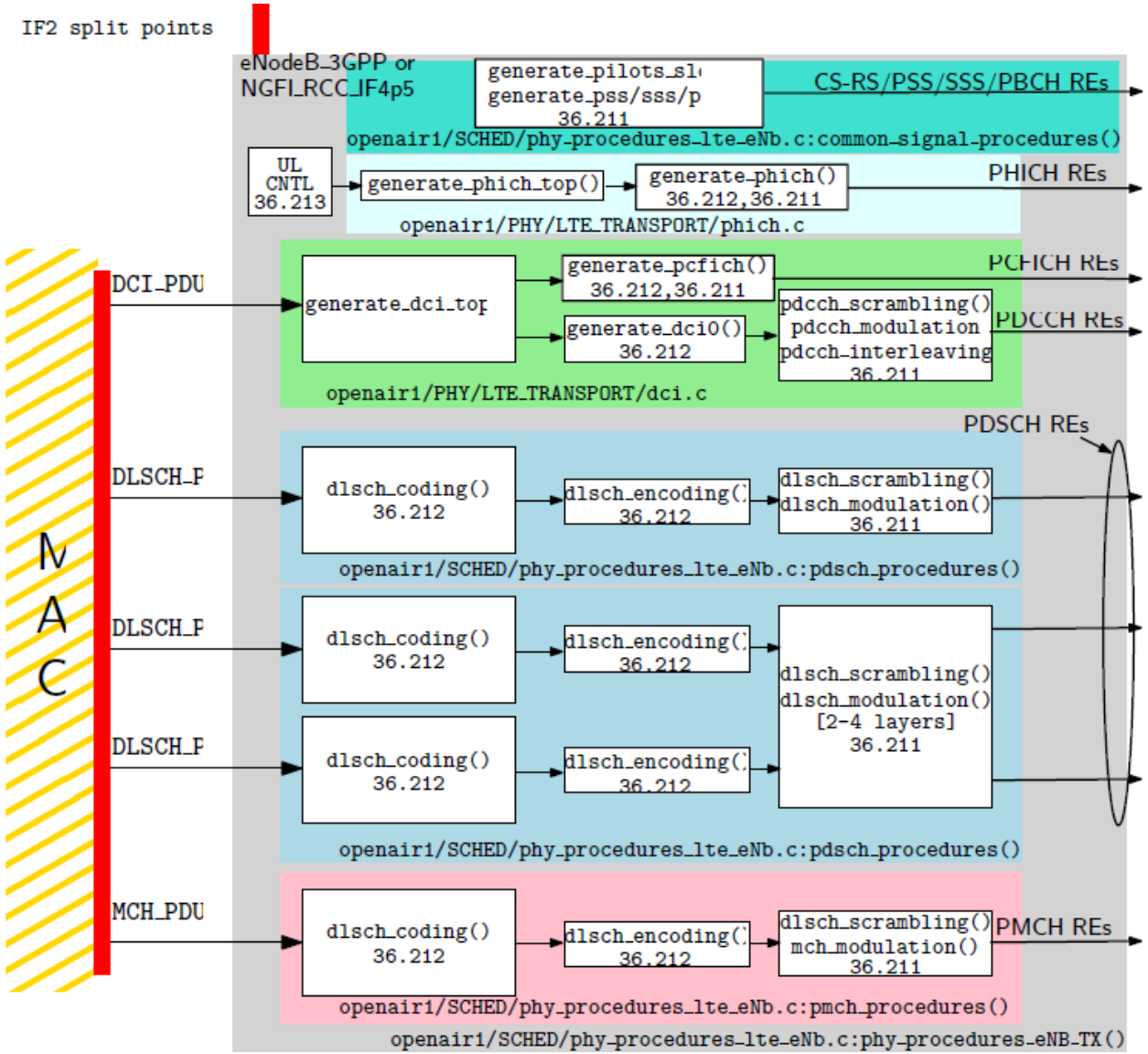


TX Precoding (monolithic eNB/gNB)

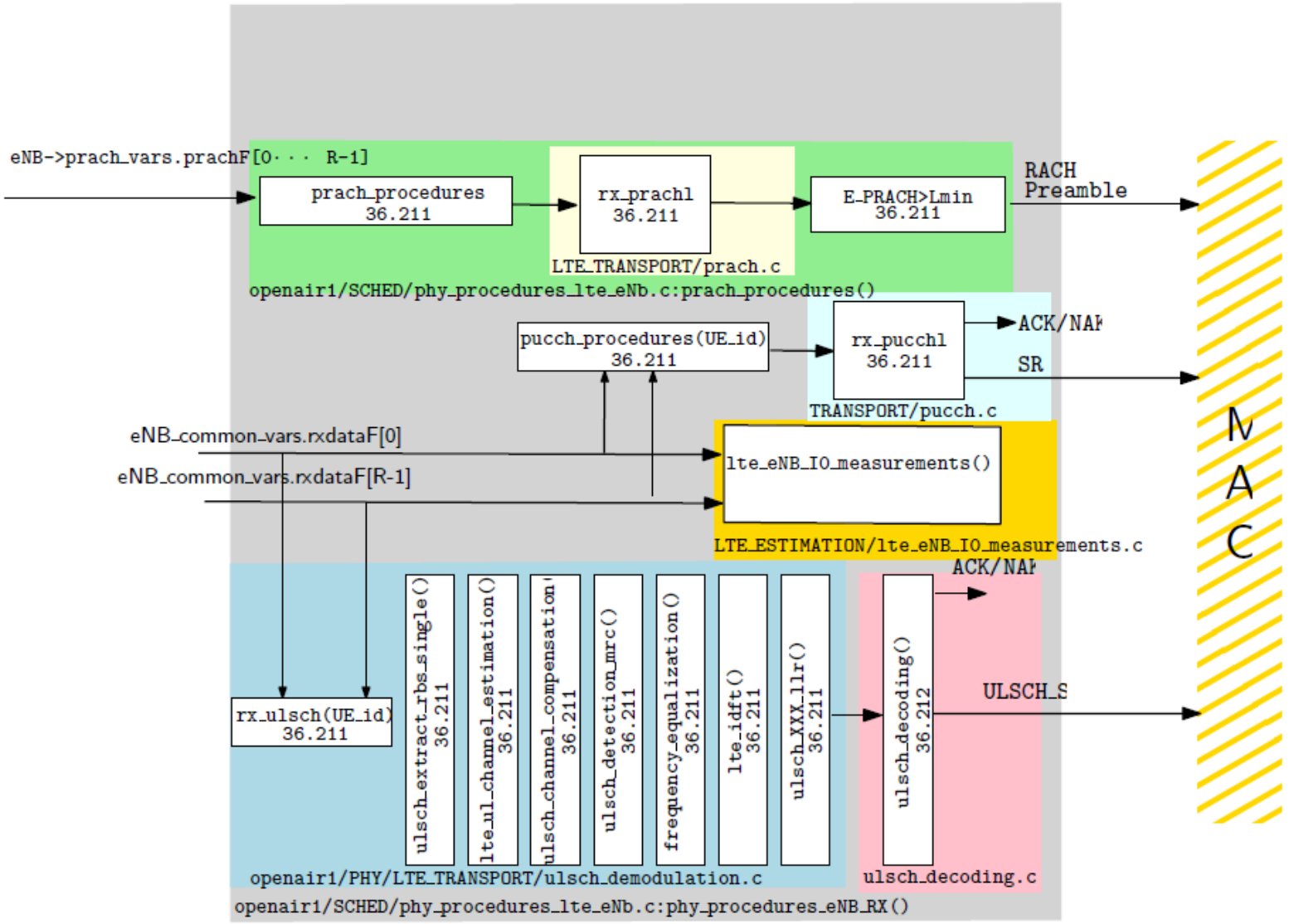


LAYER 1 HIGH SEGMENT (ENB/GNB)

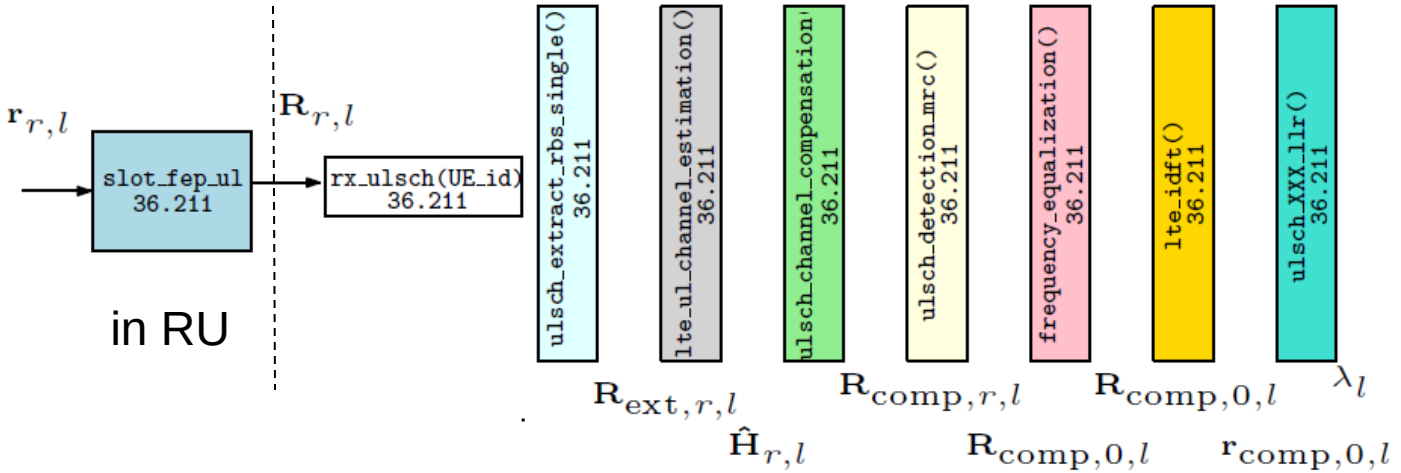
Layer 1 TX per eNB instance



Layer 1 RX per eNB instance



RU/eNodeB OAI PUSCH Procedures



$\mathbf{R}_{r,l} = \mathrm{DFT}_{N_{\mathrm{fft}}}(\mathbf{r}_{r,l} \odot \mathbf{F}_{7.5}), r = 0, 1, \cdots, R-1, l = 0, 1, \cdots, N_{\mathrm{symb}} - 1$ (eNB_common_vars→rxdataF[][])

$R_{\mathrm{ext},r,l}(n) = R_{r,l}(12\mathrm{firstPRB} + n), n = 0, 1, \cdots, 12N_{\mathrm{PRB}} - 1$ (eNB_pusch_vars→ulsch_rxdataF_ext[][])

$\hat{\mathbf{H}}_{r,l} = \mathbf{R}_{\mathrm{ext},r,l} \odot \mathbf{DRS}^*_l(\mathrm{cyclicShift}, n_{\mathrm{DMRS}(2)}, n_{\mathrm{PRS}})$, (eNB_pusch_vars→drs_ch_estimates[])

$\mathbf{R}_{\mathrm{comp},r,l} = \hat{\mathbf{H}}_r \odot \mathbf{R}_{\mathrm{ext},r,l} 2^{-\log_2 |H\max|}, \hat{\mathbf{H}}_r = \frac{1}{2}(\hat{\mathbf{H}}_{r,3} + \hat{\mathbf{H}}_{r,10})$
(eNB_pusch_vars→ulsch_rxdataF_comp)

$\mathbf{R}_{\mathrm{comp},0,l} = \frac{1}{R}\sum_{r=0}^{R-1} \mathbf{R}_{\mathrm{comp},r,l}$

$R_{\mathrm{comp},0,l}(n) = R_{\mathrm{comp},0,l}(n)\dot{Q}_8\left(\frac{1}{|\hat{H}(n)|^2 + I_0}\right), \hat{H}(n) = \sum_{r=0}^{R-1} \hat{H}_r(n)$

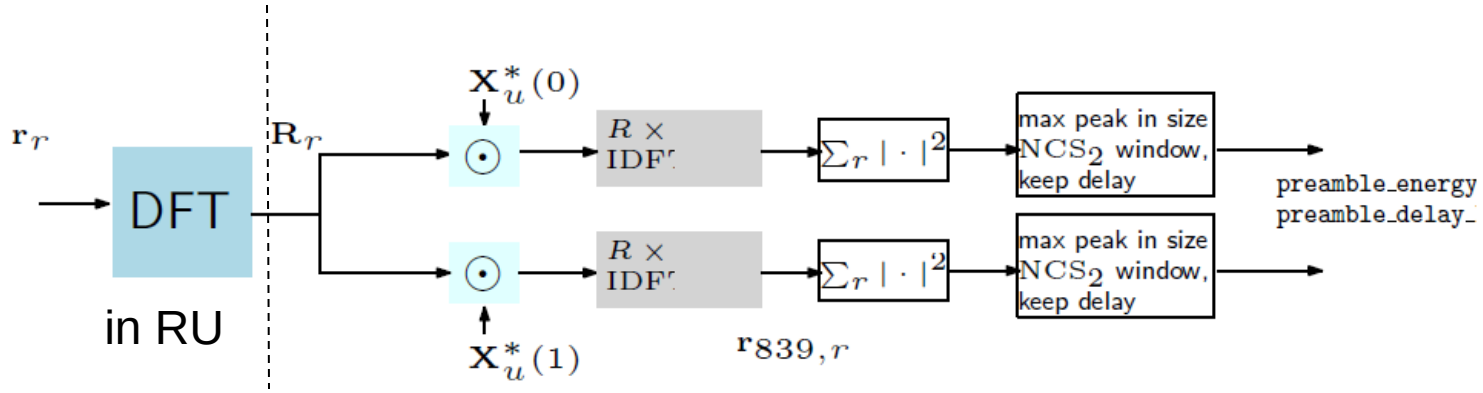
$\mathbf{r}_{\mathrm{comp},0,l} = \mathrm{IDFT}_{12N_{\mathrm{PRB}}}(\mathbf{R}_{\mathrm{comp},0,l})$

QPSK : $\lambda_l(2n) = \mathrm{Re}(r_{\mathrm{comp},0,l}(n)), \lambda_l(2n+1) = \mathrm{Im}(r_{\mathrm{comp},0,l}(n))$ (eNB_pusch_vars→ulsch_llr)

16QAM : $\lambda_l(4n) = \mathrm{Re}(r_{\mathrm{comp},0,l}(n)), \lambda_l(4n+2) = \mathrm{Im}(r_{\mathrm{comp},0,l}(n))$

$\lambda_l(4n+1) = |\mathrm{Re}(r_{\mathrm{comp},0,l}(n))| - 2\overline{|h(n)|}, \lambda_l(4n+3) = |\mathrm{Im}(r_{\mathrm{comp},0,l}(n))| - 2\overline{|h(n)|}$

RU/eNodeB PRACH procedures



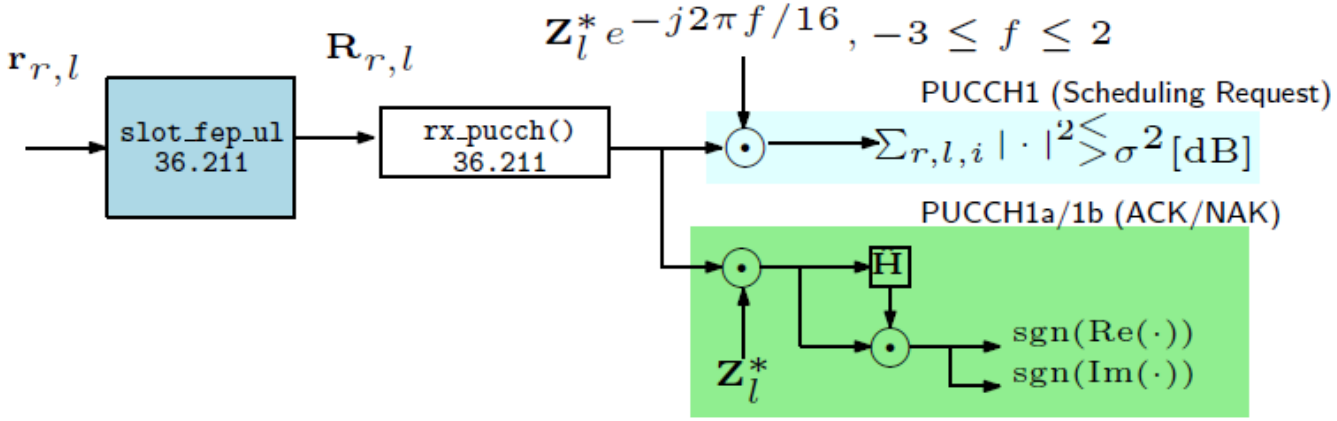
$\mathbf{R}_r = \mathrm{DFT}_{N_{\mathrm{PRACH}}}(\mathbf{r}_r), r = 0, 1, \cdots, R-1$ (lte_eNB_prach_vars→rxsigF[])

$\mathbf{R}_{\mathrm{comp},r} = \mathbf{R}_r \odot \mathbf{X}_u^*[i], r = 0, 1, \cdots, R-1$ (lte_eNB_prach_vars→prachF[])

$\mathbf{r}_{839,r} = \mathrm{IDFT}_{1024}(\mathbf{R}_{\mathrm{comp},r}), r = 0, 1, \cdots, R-1$ (lte_eNB_prach_vars→prach_ifft[])

- PRACH detection is a quasi-optimal non-coherent receiver for vector observations (multiple antennas)
- correlation is done in the frequency-domain, number of correlations (in the example above 2) depends on *zeroCorrelationConfig* configuration parameter
- peak-detection (for delay estimation) is performed in each NCS time-window

eNodeB PUCCH (1/1A/1B) Procedures

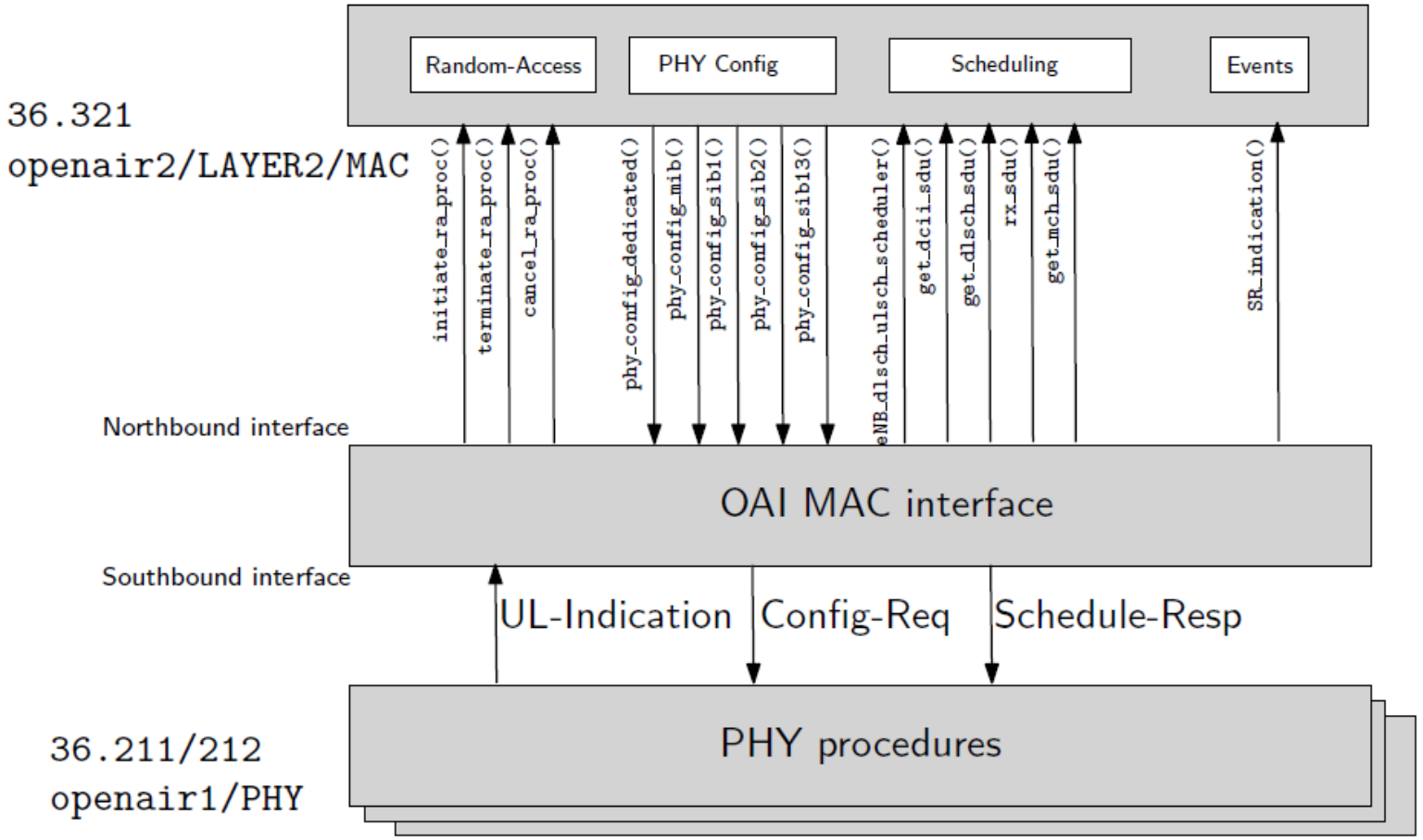


- PUCCH1 detection is a quasi-optimal non-coherent receiver (energy detector) for vector observations (multiple antennas) for scheduling request. Care is taken to handle residual frequency-offset.
- PUCCH1A/1B detection is quasi-coherent based on a rough channel estimate obtained on the 3 symbols without data modulation.
- In both cases, correlation is done in the frequency-domain

PHY/MAC INTERFACE

OAI IF1'' – MAC/PHY Interface

- OAI IF1" is the interface between the 36.321 Medium-Access (MAC) Layer Procedures and the 36.213 Physical Layer Procedures. It links several PHY instances to one MAC instance.
- It is a configurable (dynamically loadable) module which can implement an (N)FAPI P5/P7 or a simpler interface.



OAI IF1'' – MAC/PHY

- The PHY end uses three basic messages
  - CONFIG_REQ: this provides the cell configuration and UE-specific configuration to the PHY instances. This comprises the following FAPI P5/P7 messages
    1. CONFIG.request
    2. UE_CONFIG.request
  - UL_INDICATION This is an uplink indication that sends all UL information received in one TTI, including PRACH, if available. It also provides the subframe indication for the DL scheduler. It maps to the following FAPI P7 messages
    1. SUBFRAME.indication
    2. HARQ.indication
    3. CRC.indication
    4. RX_ULSCH.indication
    5. RX_SR.indication
    6. RX_CQI.indication
    7. RACH.indication
    8. SRS.indication
  - SCHEDULE_REQUEST This message contains the scheduling response information and comprises the following FAPI P7 messages
    1. DL_CONFIG.request
    2. UL_CONFIG.request
    3. TX.request
    4. HI_DCI0.request
- The module is registered both by PHY and MAC and can implement different types of transport (NFAPI, function call, FAPI over UDP, etc.). During registration, fuction pointers for the different messages are provided for the module to interact with either PHY or MAC or both if they are executing in the same machine. Note that for a networked implementation (e.g. NFAPI), there are north and south components running in different machines.

OAI IF1'' – MAC/PHY

- The PHY-layer timing is assumed to be
  1. wait for subframe indication n from HW
  2. trigger PRACH if n has PRACH (parallel thread)
  3. trigger UE specific RX procedures for n if n is UL
  4. assemble `UL_INDICATION` and send to MAC
  5. wait for `SCHEDULE_REQUEST`
  6. do TX procedures if $n+4$ is TX and RX programming if $n+4+k$ is UL
- The MAC-layer timing is assumed to be
  1. do all UL processing for subframe n if n is UL after unraveling of UL_INDICATION in MAC module
  2. wait for call to `eNB_dlsch_ulsch_scheduler`
  3. do DL scheduling for $n+4$ if it is DL
  4. do UL scheduling for $n+8$ if it is UL
  5. return from eNB_dlsch_ulsch_scheduler
  6. let MAC module form SCHEDULE_REQUEST

RU AND LAYER 1 PROCESS SCHEDULING

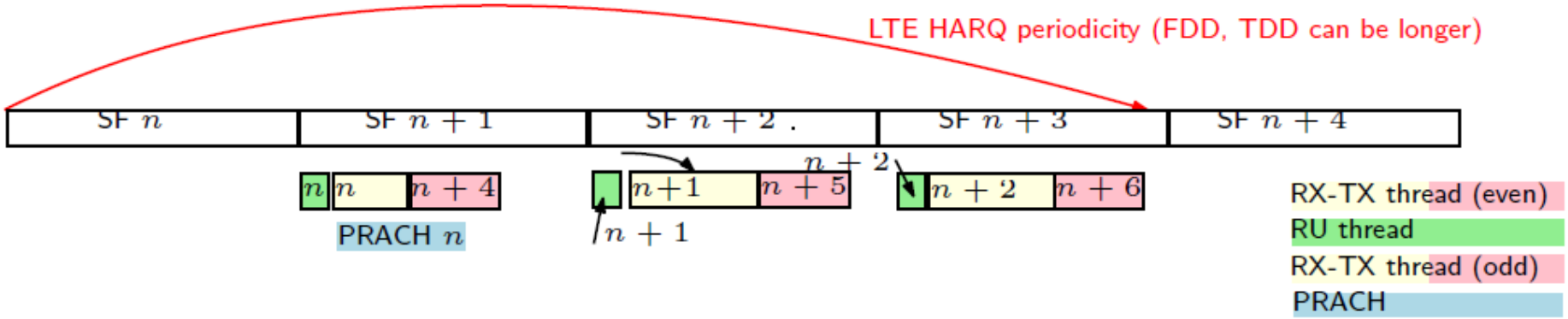
RU Process Scheduling

- Threads (all in `targets/RT/USER/lte-ru.c`)
  - `ru_thread`: Thread per RU which sequentially performs
    - read from south interface (RF or IF fronthaul)
    - RX processing for subframe n (if necessary).
    - wakeup eNBs that are waiting for signal (if necessary)
    - wait for eNB task completion (if necessary)
    - do TX processing for subframe $n + 4$ (if necessary). Note that this can spawn multiple worker threads for very high order spatial processing (e.g. massive-MIMO or DAS for UDN)
    - do outgoing fronthaul (RF or IF fronthaul)
  - `ru_thread_prach`: Thread for PRACH processing in remote RU (DFT on RX, IF4p5 RRU)
  - `ru_thread_asynch`: Thread for asynchronous reception from fronthaul interface (TX direction in RRU).
- Synchronization on fronthaul interface
  - *synch_to_ext_device* : synchronizes to incoming samples from RF or Fronthaul interface using blocking read
  - *synch_to_other* : synchronizes via POSIX mechanism to other source (other CC, timer) which maintains real-time.

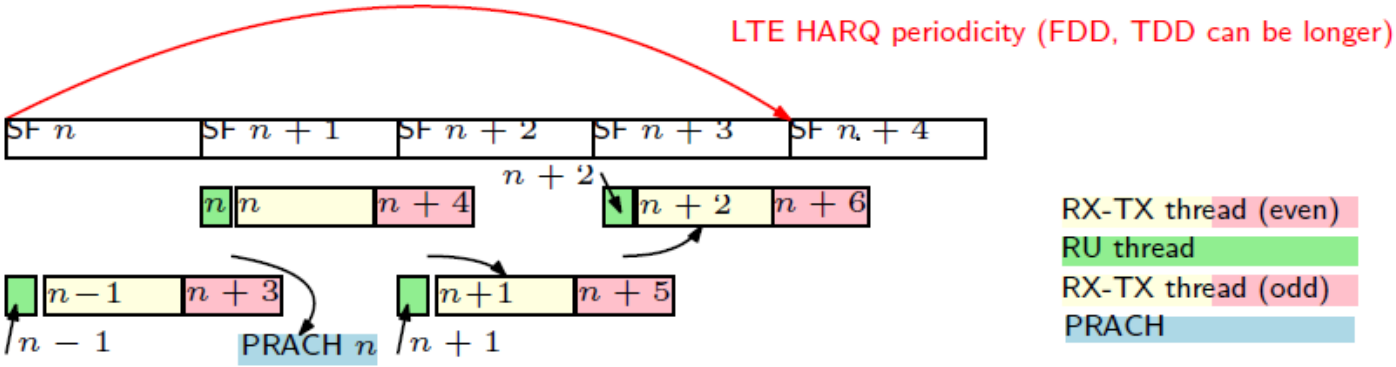
L1 Process Scheduling

- Threads (all in `targets/RT/USER/lte-enb.c`)
  - multi RX/TX thread mode (optional)
    - `eNB_thread_rxtx`: 2 threads per CC/Instance which do both RX procedures for subframe n and TX procedures for subframe $n+4$. One operates on even subframes, one on odd. This allows 1ms subframe processing to use multiple-cores.
  - common RU-eNB RX/TX thread (default if single RU/eNB)
    - calls `eNB_top`: procedure per CC/Instance which sequentially
      - blocks on signal from RU
      - RX/TX processing for subframe n and $n+4$
      - signals completion to RU
  - `eNB_prach`: Thread per CC_id/Instance for PRACH processing

Timing ("Almost" Single-Thread Mode)



Timing (Even/Odd Threading Mode)



- The current processing requires approximately 1ms peak in each direction (basically 1 core RX, 1core TX). The current architecture will work on a single core if the sum of RX and TX procedures is limited to 1ms. It can fit on 2 cores if the sum of RX,TX and PRACH is less than 2ms.
- three threads, RX-TX even, RX-TX odd and PRACH. RX-TX blocks until woken by the RU thread with a new RX subframe n that is linked to this eNB process. The RX-TX thread performs ue-specific processing for subframe n and then TX common and ue-specific processing for subframe $n+4$ (frequency-domain generation only). This insures the data dependency between TX $n+4$ and RX n is respected. The duration of this thread should be less than 2ms which can compensate some jitter on the RX processing.

Multi-threading lower-layer operations

- **Front-end Processing (RU)**
  - Parallelizing even/odd slots in Fourier Transforms (TX and RX in RU)
- **Back-end Processing (L1)**
  - Parallelizing Segments in Turbo-encoder / Rate-Matching
  - Parallelizing Segments in Rate-Matching Inversion / Turbo-Decoder
- **Run worker threads in parallel to main thread in “single-thread” mode**

SIMD PROCESSING IN OAI

SIMD in OAI

- **All key time-critical routines are coded using fixed-point processing and SIMD vector extensions in x86-64 or ARM architecture**
  - FFTs and basic routines use 16-bit AVX2 (downgraded to SSE4 on AVX2-less targets)
  - Turbo encoder uses a combination of AVX2 and MMX
  - Turbo decoder uses 8-bit SSE4 (128-bit). Will be upgraded to AVX2. But this takes effort and time …
  - Viterbi decoder uses 8-bit SSE4
  - ARM Neon largely written
- **Soon upgraded to AVX512 on recent IA**
- **New routines to come**
  - LDPC encode/decode for NR

Key linear processing operations which can be vectorized easily with SIMD

- **Inner-products (synchronization, projection operations)**
- **Componentwise products (compensation/equalization, channel estimation)**
- **FFT/IFFTs (OFDM TX/RX)**
- **Peak search in vector (synchronization)**
- **Vector addition, subtraction, energy, absolute value**

Coding with SIMD

- **SIMD = Single Instruction Multiple Data**
  - Operate on vectors of samples with parallel instructions
- **Good for linear processing in front-end and trellis-processing (turbo/Viterbi decoding)**

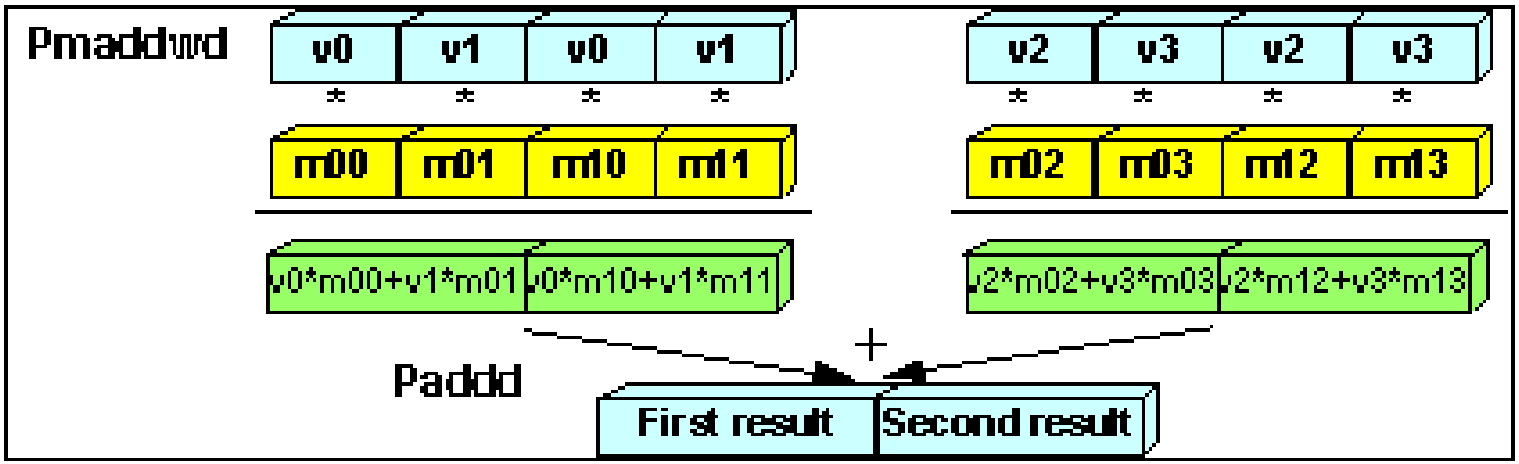
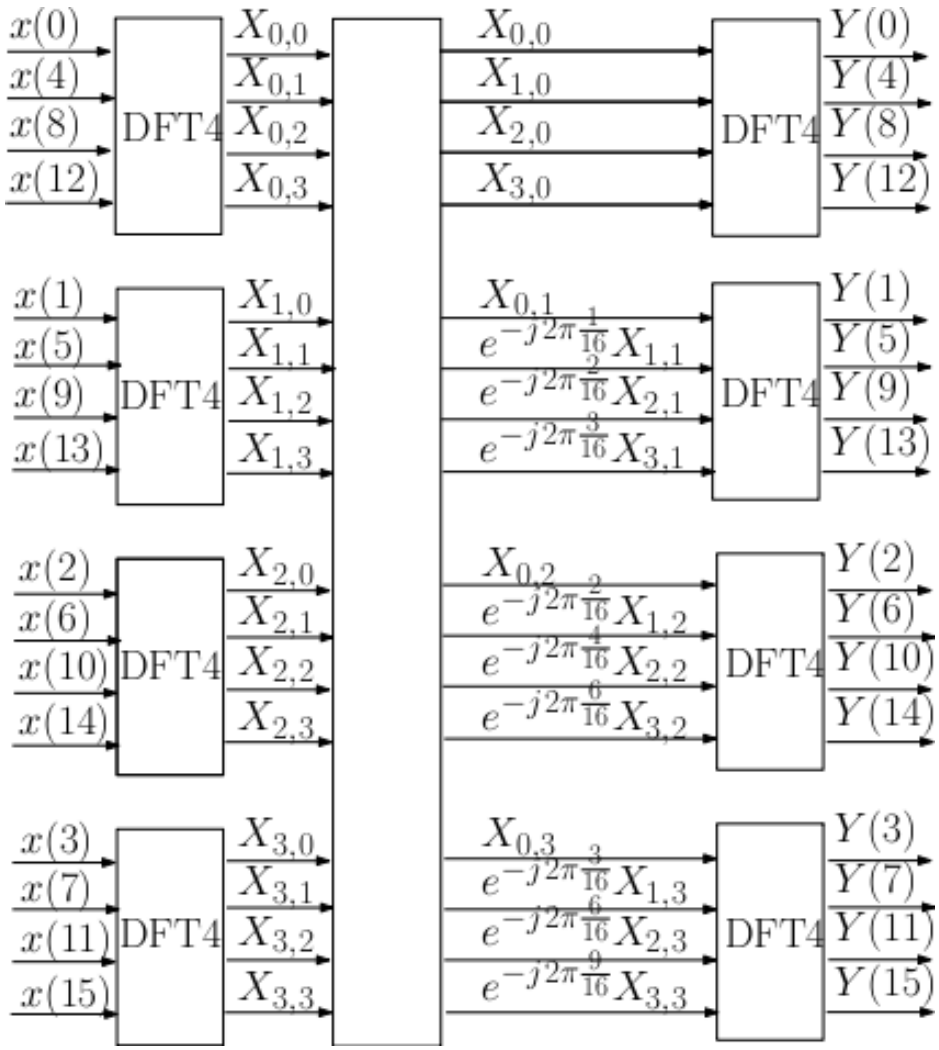


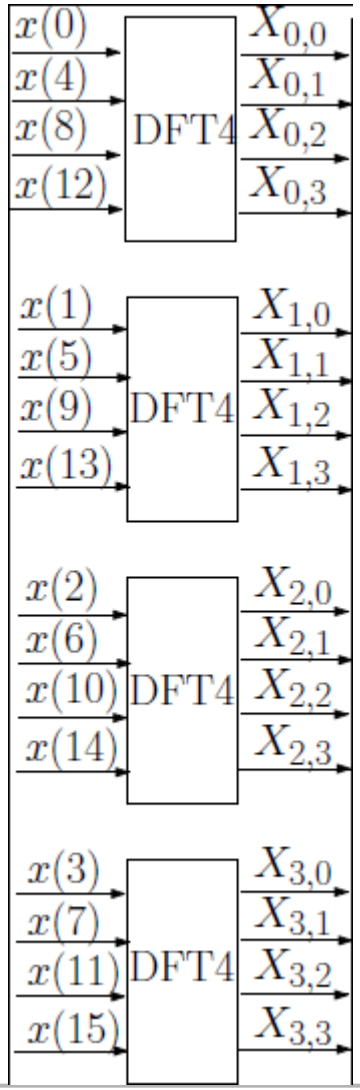
Figure 5. MMX Technology Matrix-Vector Multiplication

OAI Radix-4 DFT

- **Basic idea is to optimize**
  - the 16-point DFT with SIMD
  - the 16-bit complex radix-4 and radix-2 butterflies
  - Build up 64,128,256,512,1024,2048 from there
- **Two butterfly stages (radix 4), one with twiddles**
- **One permutation**



OAI DFT with SSE4



=

```
register __m128i x1_flip,x3_flip,x02t,x13t;
register __m128i xtmp0,xtmp1,xtmp2,xtmp3;

 // First stage : 4 Radix-4 butterflies without input twiddles
 x02t    = _mm_adds_epi16(x128[0],x128[2]);
 x13t    = _mm_adds_epi16(x128[1],x128[3]);
 xtmp0   = _mm_adds_epi16(x02t,x13t);
 xtmp2   = _mm_subs_epi16(x02t,x13t);

 // multiplication of x1 and x3 by sqrt(-1)
 x1_flip = _mm_sign_epi16(x128[1],*(__m128i*)conjugatedft);
 x1_flip = _mm_shufflelo_epi16(x1_flip,_MM_SHUFFLE(2,3,0,1));
 x1_flip = _mm_shufflehi_epi16(x1_flip,_MM_SHUFFLE(2,3,0,1));
 x3_flip = _mm_sign_epi16(x128[3],*(__m128i*)conjugatedft);
 x3_flip = _mm_shufflelo_epi16(x3_flip,_MM_SHUFFLE(2,3,0,1));
 x3_flip = _mm_shufflehi_epi16(x3_flip,_MM_SHUFFLE(2,3,0,1));

 x02t    = _mm_subs_epi16(x128[0],x128[2]);
 x13t    = _mm_subs_epi16(x1_flip,x3_flip);
 xtmp1   = _mm_adds_epi16(x02t,x13t);  // x0 + x1f - x2 - x3f
 xtmp3   = _mm_subs_epi16(x02t,x13t);  // x0 - x1f - x2 + x3f
```

Performance and Extensions

- **OAI Radix-4 DFTs provide essentially the same performance as FFTW**
  - Approx 300 cycles for 64-point DFT on a x86-64 core i5/i7/Xeon machine (gcc generated)
  - For larger DFTs, both are slightly less than the closed IPP libraries from Intel (based on SPIRAL code generator)
  - OAI doesn't make use of FFTW only because we use the DFT optimizations for teaching SIMD coding to our Engineering diploma students.
- **OAI also provides all the mixed-radix DFTs that are required for UE and eNB DSP**
  - $2^a3^b5^c$ for SC-FDMA (uplink transmission formats)
  - 2^N3 for random-access preamble (PRACH TX at UE and RX at eNB)

Turbo Decoding

- **The primary (by far) bottleneck in an HSPA/LTE receiver is the turbo decoder**
- **SIMD parallelization can be done in different ways**
  - OAI adopts a scalable SIMD *max-logmap* approach whereby code blocks (8-bit LLRs) are split in 16 (SSE4) or 32 (AVX2) equal size sub-blocks
  - integer x86 SIMD arithmetic is used to efficiently perform the forward-backward BCJR recursions (16 or 32-way parallel add, subtract, max) and extrinsic information computation
  - Care must also be taken when it comes to the interleaving and data reforming operations in a turbo decoder. These can also benefit from SIMD
- **x86-64 is much more efficient for Turbo decoding than legacy x86 because of the fact that the number of available registers is doubled**

RTOS issues

- **Low-latency radio applications for PHY (e.g. 802.11x,LTE) should run under an RTOS**
  - eCos/MutexH for generic GNU environment
  - RTAI for x86
  - VXWorks ($$$)
- **Example: RTAI / RT-PREEMPT kernel can achieve worst-case latencies below 30μs on a loaded-PC. More than good enough for LTE, but not 802.11x because of MAC timing.**
- **Should make use of POSIX multithreading for SMP**
  - Rich open-source tool chains for such environments (Linux, BSD, etc.)
  - Simple to simulate on GNU-based systems for validation in user-space
  - Allow each radio instance to use multiple threads on common HW

Issues with standard Linux Kernels

- **Scheduler latency**
  - Kernel is not pre-emptible
  - Overhead in disabling/enabling interrupts
- **Mainstream kernel solutions**
  - Kernel preemption (RT-PREEMPT) – mainstream until 2.6.32 (patches afterwards)
  - Latency reduction (soft-RT kernels)
  - Version >3.14
- **Patches / dual-OS solution**
  - ADEOS + RTAI/Xenomai

RTAI/Xenomai

- **Real-time nano-kernel tightly integrated with Linux kernel**
  - Linux runs as a low-priority thread under RTAI/Xenomai and and both share same memory space
  - Both RTAI and Xenomai make use of ADEOS which is a hardware abstraction framework which allows several OS to share HW resources
    - Low-level Scheduler allows Linux kernel to run in background
  - Both RTAI and Xenomai provide user-space real-time functionality in addition to kernel-only applications
    - Provide various API flavours to resemble classical OS (e.g. Posix, VXWorks, RTDM, etc.)

RT-PREEMPT and recent Linux Low-Latency Kernels

- **The RT-Preempt patch and out-of-the-box Linux kernel (>3.14) converts Linux into a fully preemptible kernel. The magic is done with:**
  - Making in-kernel locking-primitives (using spinlocks) preemptible though reimplementation with rtmutexes.
  - Critical sections protected by i.e. spinlock_t and rwlock_t are now preemptible. The creation of non-preemptible sections (in kernel) is still possible with raw_spinlock_t (same APIs like spinlock_t).
  - Implementing priority inheritance for in-kernel spinlocks and semaphores. For more information on priority inversion and priority inheritance please consult Introduction to Priority Inversion.
  - Converting interrupt handlers into preemptible kernel threads: The RT-Preempt patch treats soft interrupt handlers in kernel thread context, which is represented by a task_struct like a common user space process. However it is also possible to register an IRQ in kernel context.
  - Converting the old Linux timer API into separate infrastructures for high resolution kernel timers plus one for timeouts, leading to user space POSIX timers with high resolution.
- **In our experience, these solutions provide slightly lower-performance w.r.t. RTAI, but the other advantages greatly outweigh this penalty.**