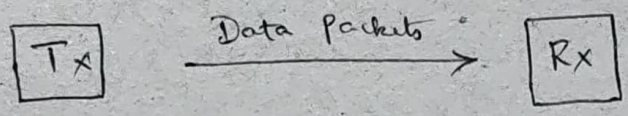
HARQ : What and Why ? (PDSCH & PUSCH)

HARQ → Hybrid Automatic Repeat Request.

Before we understand how the HARQ is defined in the specifications and what are the channels where the HARQ is being used, let us understand why do we need HARQ ?

HARQ is applicable to both Downlink (PDSCH) and Uplink (PUSCH).



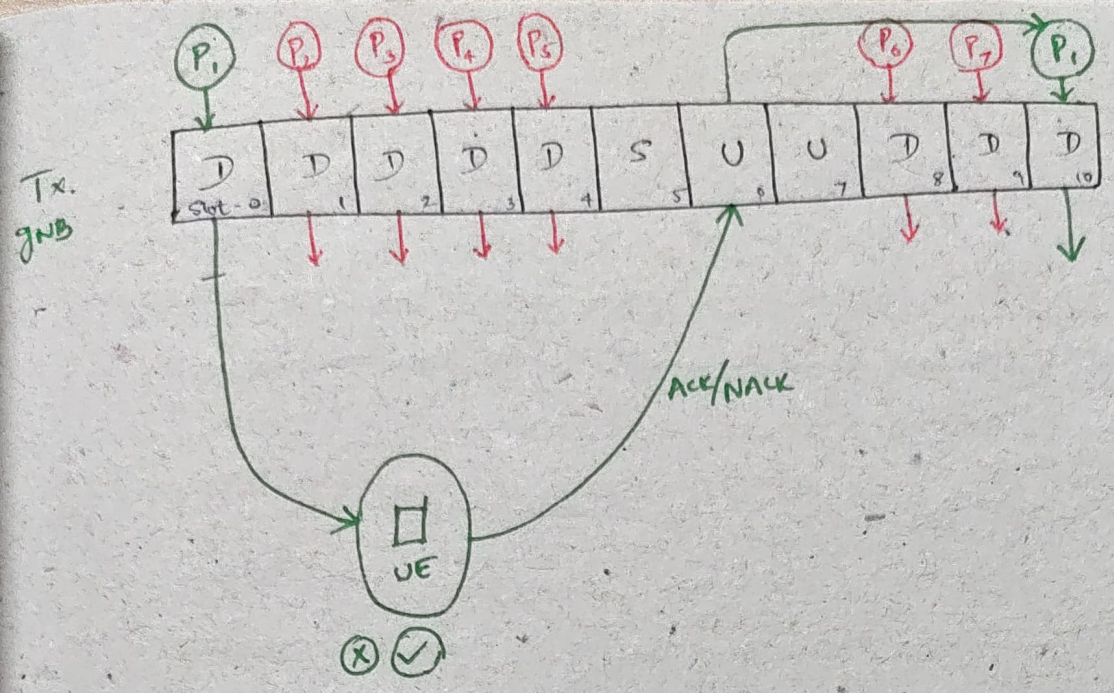
The transmitter sends a packet of data to the receiver. The received packets may or may not be correctly decoded at the Receiver. The receiver should communicate whether the transmitted packet got successfully decoded (ACK) or failed to decode (NACK).

If it is successfully decoded, then it's well and good and the transmitter will send the next data packet. If it fails to decode, then the transmitter should retransmit that packet back.

Now the question comes, whether the transmitter should transmit the same packet in the exact same way, or should the transmitter be modifying the bits and transmit in a different way ?

Another question comes, How many times the transmitter should keep transmitting the packet when it keeps failing ?

Another question, When the receiver is processing, should the transmitter wait for the first packet to get decoded or should it do something else ?



Consider the TDD slot structure above. Now, at slot 0, the transmitter transmits some packets (P_1) to the UE. The UE receives the packet (P_1). Now, the UE takes some time to process the packet (P_1). If the UE fails to decode this packet, it sends NACK in the next UL slot (slot 6) or wherever the ACK/NACK is scheduled to send back. If the UE successfully decodes this packet, then also it sends ACK to the gNB. Now, the gNB takes some time to process. If NACK is received, the gNB retransmits the same packet (P_1) once again to UE.

Considering there is only one UE connected to the BS. Now, what the BS should do at the remaining DL slots? Whether should it just keep waiting for (P_1) to get successfully decoded? or can it transmit (P_2), (P_3), (P_4), ... in the next DL slots?

So, all these procedures comes under HARQ.

Hybrid ?

When the receiver fails to decode the packet, it requests the transmitter to transmit once again. Then the transmitter transmits the same data back to the receiver. Now again the receiver will try to decode this packet. This is called as ARQ.

When the ARQ is combined with FEC (Forward Error correction). Here, when the receiver fails to decode the packet, here the transmitter may not transmit the packet the same way it was transmitted before.

This is called as HARQ. (We'll understand this part in detail later)

Now, lets understand what are the channels that requires HARQ process.

We have the following channels in 5G.

SSB → For Synchronization in DL
PRACH → For synchronization in UL

PDCCH → DL Control channel
PUCCH → UL Control channel

PDSCH → DL Shared channel
PUSCH → UL Shared channel

PTRS / CSI-RS / SRS → Reference Signals (RS)

The purpose of the Reference Signals is to find out how bad the channel is. (ie) when Reference Signals are transmitted, at the receiver side we may not receive the exact same signal (may be distorted). And this is what we suppose to find out. So, there is no necessity to retransmit the signal in case of decoding failure at the receiver side. So, HARQ is not applicable for RS.

The SSB and PRACH signals are designed for synchronization. SSB is broadcasted all the time with some periodicity. And if the UE fails to decode SSB, it can always try to decode the next SSB. UE cannot send ACK/NACK for SSB that is failed to decode. PRACH is either used for Timing Advance or for the Initial access (when UE tries to connect with gNB). So, when the gNB fails to decode PRACH, UE can reinitiate after a particular time. (ie) when the UE sends PRACH to gNB, and the gNB fails to decode (or) UE doesn't receive the response for PRACH, then UE can reinitiate PRACH again. (Something like retransmission). So, HARQ is not applicable for SSB/PRACH too.

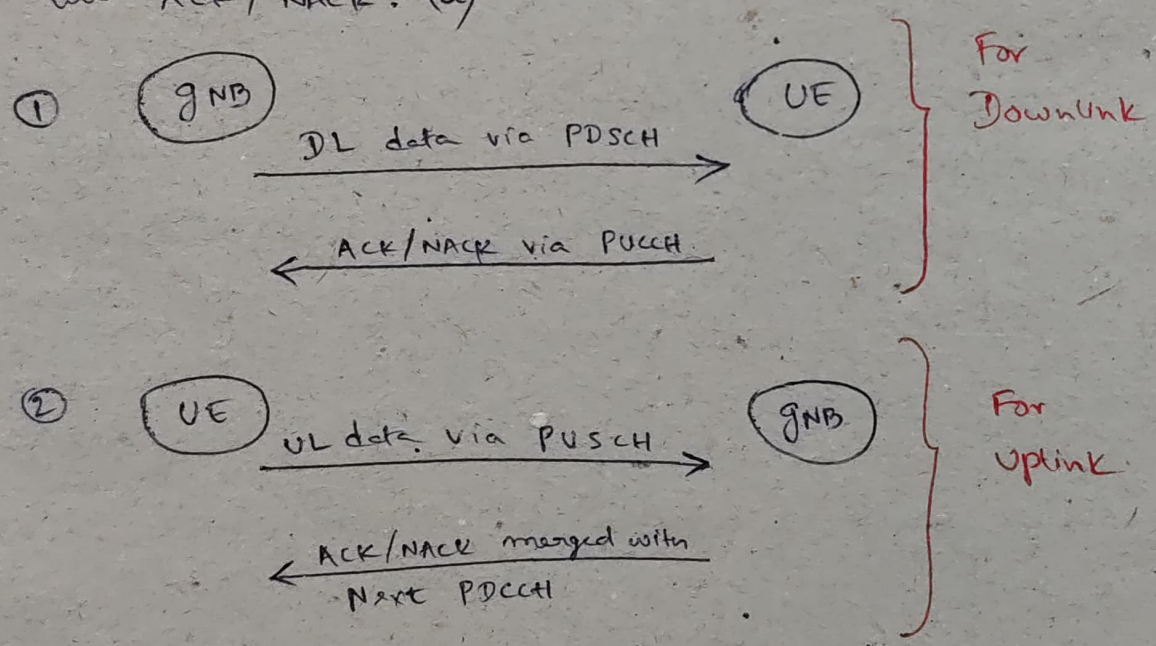
PDCCH goes, whenever there is data for either PDSCH or PUSCH. It could be DCI_0-0, DCI_0-1, DCI_1_0 or DCI_1-1. If the receiver fails to decode PDCCH, then the transmitter has to transmit everything again. Say, in Downlink, if it is dynamic allocation, gNB will send PDSCH along with PDCCH. So, if the UE fails to decode PDCCH, it wont be able to decode PDSCH also, coz the configuration of PDSCH is inside PDCCH. So, if the UE fails to decode PDCCH, the gNB has to send the complete data (PDSCH + PDCCH) again.

Similarly, PUSCH is sent from UE to gNB. But before that gNB sends PDCCH. Here if the UE fails to decode PDCCH,

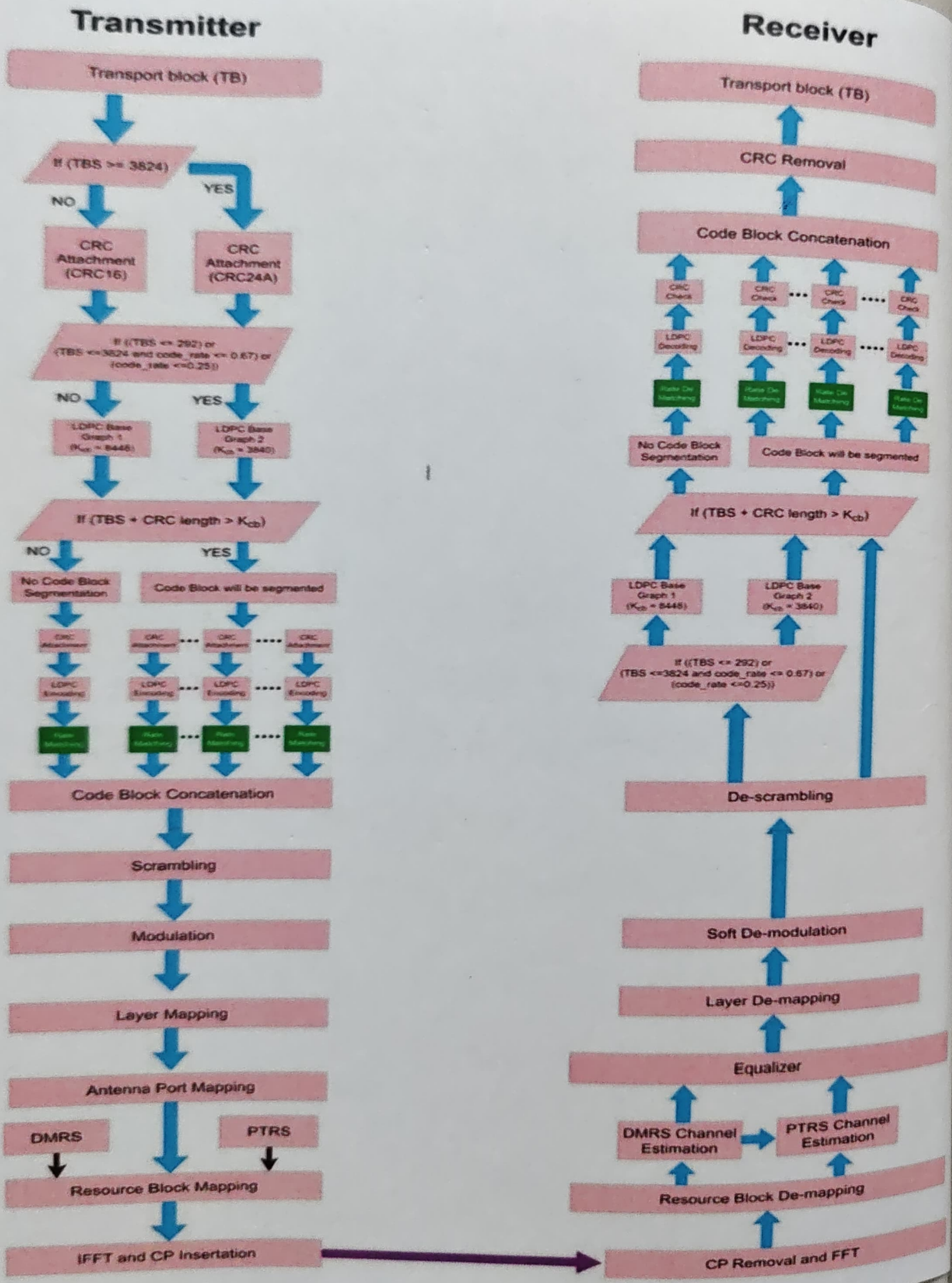
then it wont send the PUSCH, because there is no configuration available to send PUSCH. Hence the gNB will reinitiate the PDCCH again.

PUCCH contains either the channel state information (or) ACK/NACK. So if the gNB fails to decode PUCCH, it may or may not know what has failed. whether the PDCCH has failed / whether the PDSCH has failed / whether the PUCCH has failed. In this case also, it may reinitiate the data transmission.

If the PDSCH or PUSCH channel decoding failed, then there will always be ACK/NACK transmitted by the receiver. The way ACK/NACK transmitted depends on whether it is DL data / UL data. There are different ways to transmit the ACK/NACK. (ie)



So, if PDSCH / PUSCH failed at the receiver side, then the transmitter will reinitiate this data in the next slot whichever is allocated.


Transmitter
Transport block (TB)
If (TBS >= 3824)
NO
YES
CRC Attachment (CRC16)
CRC Attachment (CRC24A)
If ((TBS <= 292) or (TBS <=3824 and code_rate <= 0.67) or (code_rate <=0.25))
NO
YES
LDPC Base Graph 1 (K_{cb} = 8448)
LDPC Base Graph 2 (K_{cb} = 3840)
If (TBS + CRC length > K_{cb})
NO
YES
No Code Block Segmentation
Code Block will be segmented
Code Block Concatenation
Scrambling
Modulation
Layer Mapping
Antenna Port Mapping
DMRS
PTRS
Resource Block Mapping
IFFT and CP Insertation
Receiver
Transport block (TB)
CRC Removal
Code Block Concatenation
CRC Check
LDPC Decoding
Rate De Matching
No Code Block Segmentation
Code Block will be segmented
If (TBS + CRC length > K_{cb})
LDPC Base Graph 1 (K_{cb} = 8448)
LDPC Base Graph 2 (K_{cb} = 3840)
If ((TBS <= 292) or (TBS <=3824 and code_rate <= 0.67) or (code_rate <=0.25))
De-scrambling
Soft De-modulation
Layer De-mapping
Equalizer
DMRS Channel Estimation
PTRS Channel Estimation
Resource Block De-mapping
CP Removal and FFT

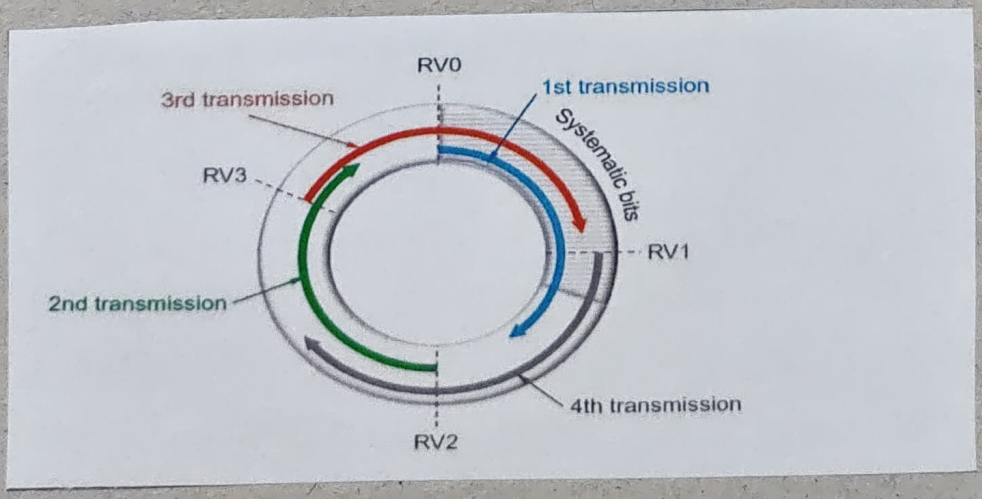
Now, lets understand how the chains are defined in the Physical layer. In this figure, single Transport Block is shown. But in downlink, there will be two Transport Blocks.

The data packets goes through this full chain at the transmitter side. At the receiver side, the received packets goes through the similar chain but the modules are in reverse order.

Be it PDSCH / PUSCH chain, the concept remains the same.

We have Rate Matching comes after code block segmentation. If there are multiple code blocks, the Rate Matching uses something called Redundancy version (Refer LDPC Rate Matching Topic). What is Redundancy version. The LDPC encoder here generates Systematic and Parity bits, which are put in a Circular buffer. Based on the code rate or the resources allocated, the Rate Matching module decides the bits to pick and then to proceed further for transmission.

So, based on the Redundancy version, it decides which bit to pick.



So, basically we put the data in a circular buffer. Based on the code Rate, we decide how many bits to pick. And based on the redundancy version, we decide where to start picking the bits. (ie) based on if it is RV0/RV1/RV2/RV3, we start picking the data bits from respective positions shown

in figure. In the initial transmission, the redundancy version is always 0, so the bits that are picked will go through the chain at the transmitter side. At the receiver side, we apply RV0 again, to decode the bits. If the receiver failed to decode, then the transmitter will reinitiate the chain once again at the next slot whichever is allocated, where the data will go through all these modules, but at the second time, the same redundancy version may not be used. We may use different redundancy version which is also communicated to the receiver.

The sequence in which the redundancy versions are used is

$$0 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 0 \rightarrow \ldots$$

So, this is the HARQ process where, in every retransmissions, instead of transmitting the same data after LDPC encoding, we transmit a different set of data from that buffer.

Now, at the receiver side, we have 2 options, where we can either <u>discard</u> the previous data (or) we can <u>keep</u> the soft bits, meaning just before LDPC decoding, we can wait for next transmission.

The first transmission is with RV0, next transmission is with RV2.

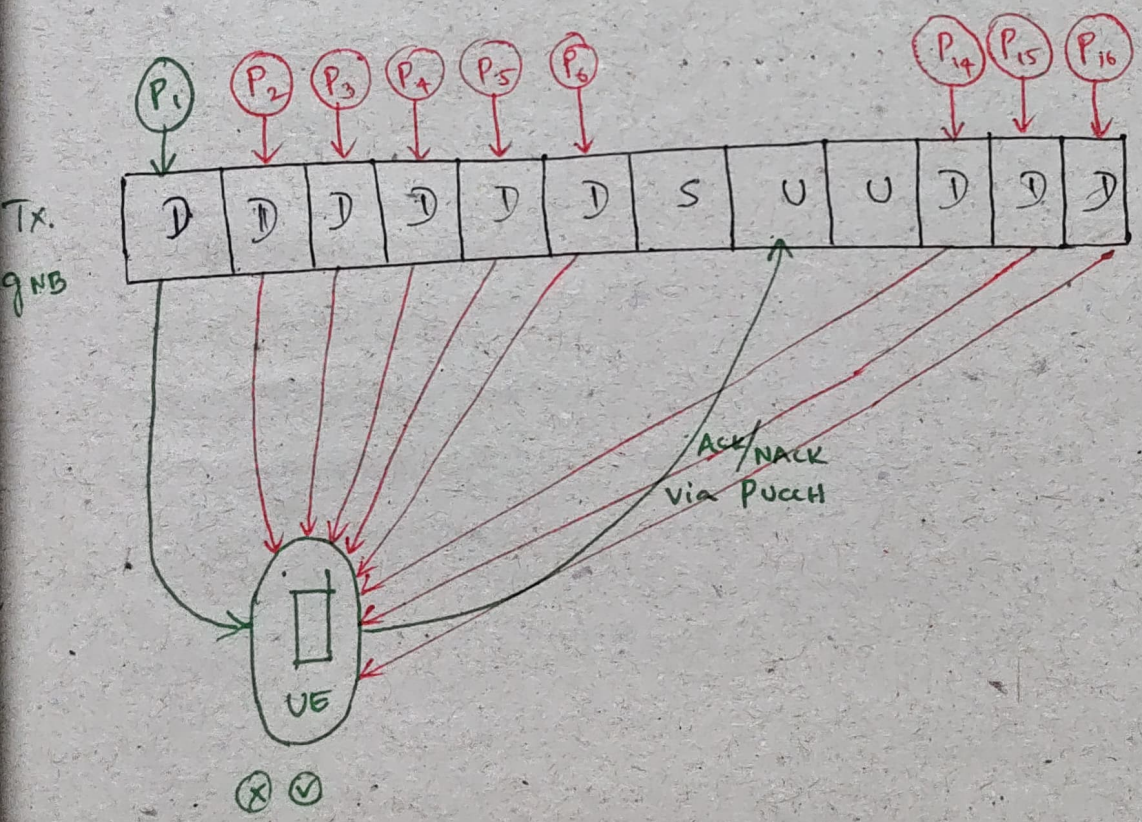
During first transmission, we store the soft bits and wait for the next transmission. In the next transmission, we try to decode independently (Only RV0 and RV3 are self decodable)

<u>Note</u>: Self decodable means, we dont have to combine with other set of data for decoding.

So, RV2 here is not self decodable. So we store the RV0 data and combine it with RV2 data, and then we try to decode and see if we are able to get the TB or not. If it still fails, we try with RV3 (Self decodable). Now, we can take a call whether to discard the previous data (or) keep it and combine with the new data and try to decode all three final combined data.

This is the <u>Single HARQ process</u> that is being followed with different redundancy version.

Consider the TDD slot structure as below.



The data packet P_1 transmitted by gNB in the DL slot, got received by the UE. Now, the gNB doesn't know whether the UE is able to successfully decode or not. The gNB will get to know it only when the UE signals it in the next UL slot. Now, gNB will take some time to process this UL data and find out whether should it have to retransmit that data P_1 (or) to transmit a new packet.

Now, the gNB will use the remaining DL slots to transmit the next consecutive packets P_2, P_3, P_4, ... and signals the UE that the packets P_2, P_3, P_4, ... are transmitted via the remaining DL slots, which are not same as the previous packet P_1. This way, the transmission becomes faster and the gNB and UE both may not have to wait for the first packet to get decoded.

Here, each packet transmission is called as <u>HARQ process</u>. The gNB (in downlink PDSCH) or the UE (in uplink PUSCH) may have <u>16 parallel HARQ</u> process. This can also be limited to 2/4/8, but <u>maximum</u> is 16 parallel HARQ processes that can be used by the gNB/UE in DL/UL.

So, without waiting for the previous ACK/NACK, we can keep transmitting 16 packets. At the receiver side, if P_1 fails and P_2 pass, then UE will store P_2. UE will keep waiting until all the packets are received (Pass) and combine all of them into a single buffer again. Once UE forms the single packet, it is sent to the upper layers for further processing. These are called the HARQ process

This is all about how the HARQ is defined, why do we use HARQ and why do we need multiple HARQ process.

HARQ : CBG based Transmission.

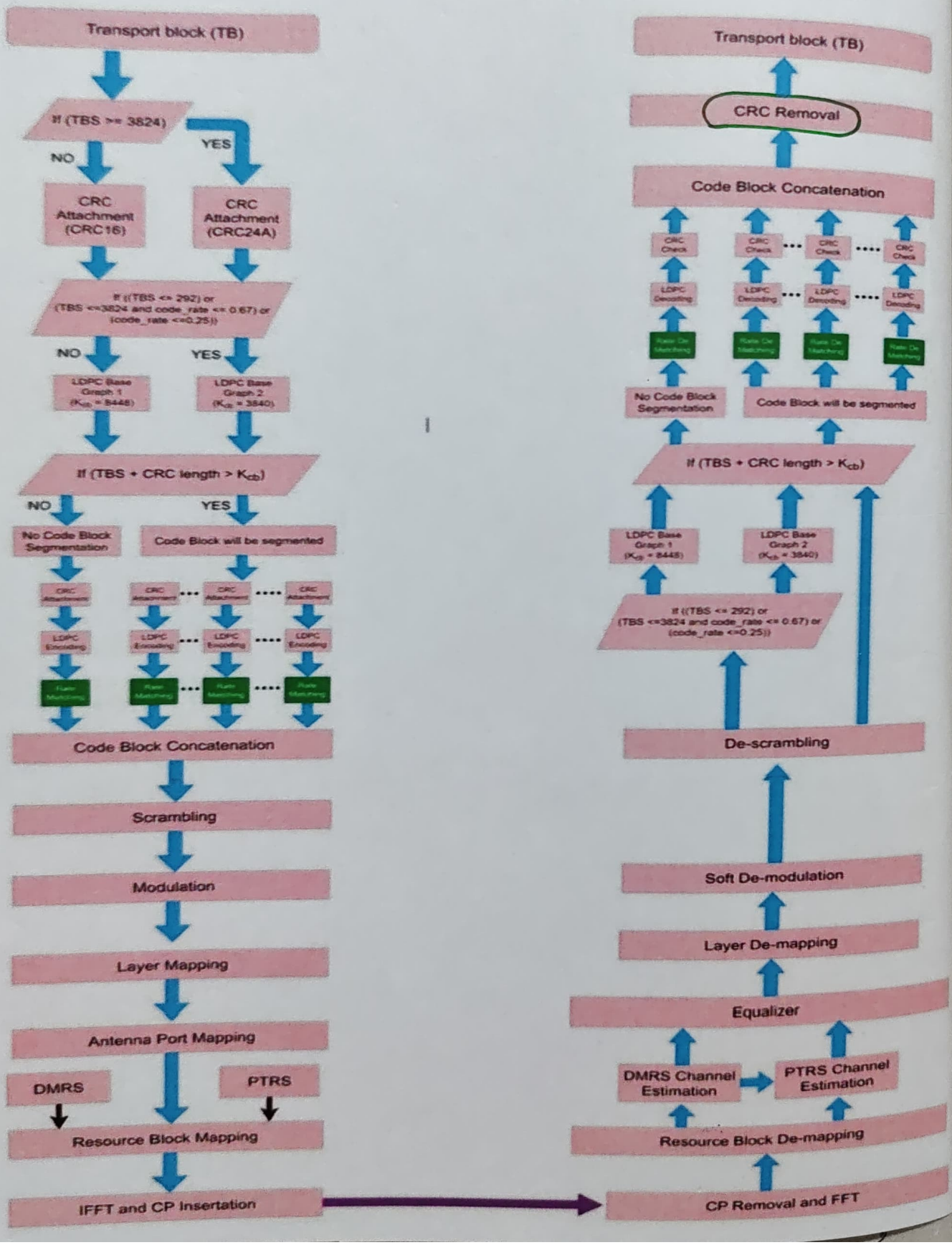
CBG → Code Block Group

(Refer figure): The Transport Block (TB) goes through the main CRC attachment. Then based on few conditions, we choose LDPC Base Graph. Based on the TB size and CRC length, we segment the TB into multiple Code Blocks (CBs). Then for each CB, we do a CRC attachment, LDPC encoding and Rate Matching. Followed by the rest of the modules in sequence.

At the Receiver side, we do reverse processing of the modules in sequence. Followed by Code Block segmentation, Rate DeMatching, LDPC decoding, CRC check, code block concatenation, CRC removal and get the TB. This TB is further sent to higher layers for processing.

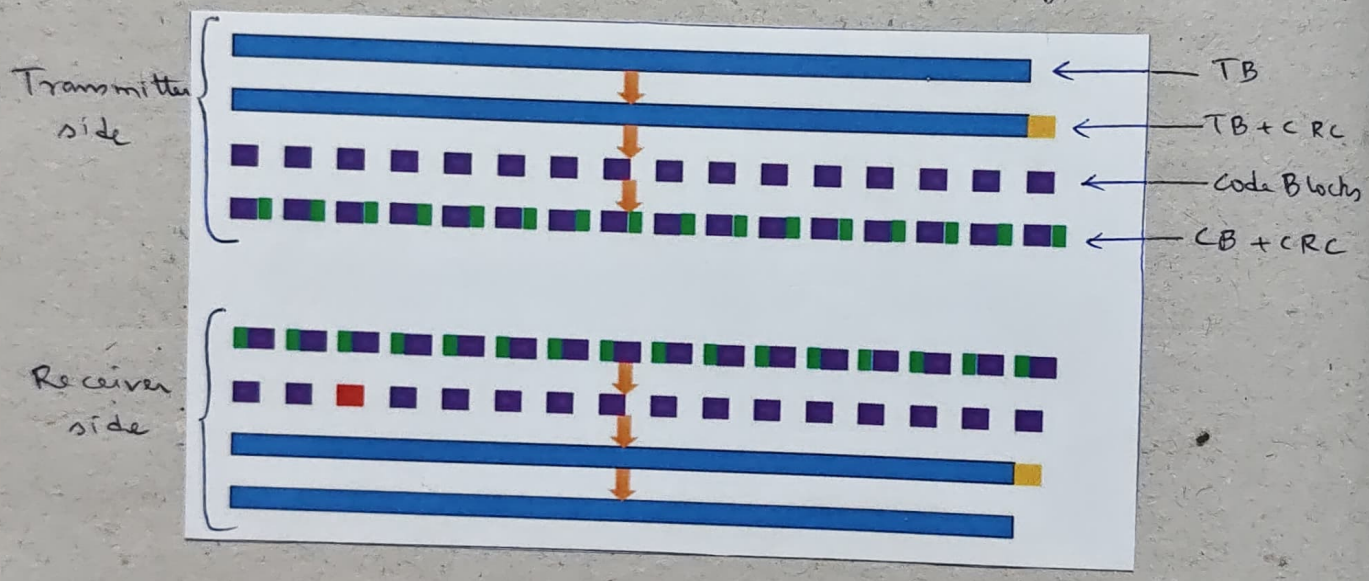
We've seen in the previous section, the ACK/NACK is being sent by the receiver is based on the main CRC removal. If the CRC pass, then we send ACK. If the CRC fails, then we send NACK to the transmitter. It could be PDSCH / PUSCH, gNB / UE, the rules remains the same.

Lets say, we have a very large TB (1,00,000 bits). This TB is further divided into multiple code Blocks. (say 15 CBs) at the transmitter side. Now, at the receiver side, lets say, only one out of 15 CBs fails and the remaining 14 CBs pass. Now, we form the final TB (concatenate the CBs, even though one CB has already failed). Eventually, the final CRC also fail. Now that the TB reception failed, the receiver


Transmitter
Transport block (TB)
If (TBS >= 3824)
NO
YES
CRC Attachment (CRC16)
CRC Attachment (CRC24A)
If ((TBS <= 292) or (TBS <=3824 and code_rate <= 0.67) or (code_rate <=0.25))
NO
YES
LDPC Base Graph 1 (K_{cb} = 8448)
LDPC Base Graph 2 (K_{cb} = 3840)
If (TBS + CRC length > K_{cb})
NO
YES
No Code Block Segmentation
Code Block will be segmented
Code Block Concatenation
Scrambling
Modulation
Layer Mapping
Antenna Port Mapping
DMRS
PTRS
Resource Block Mapping
IFFT and CP Insertation
Receiver
Transport block (TB)
CRC Removal
Code Block Concatenation
No Code Block Segmentation
Code Block will be segmented
If (TBS + CRC length > K_{cb})
LDPC Base Graph 1 (K_{cb} = 8448)
LDPC Base Graph 2 (K_{cb} = 3840)
If ((TBS <= 292) or (TBS <=3824 and code_rate <= 0.67) or (code_rate <=0.25))
De-scrambling
Soft De-modulation
Layer De-mapping
Equalizer
DMRS Channel Estimation
PTRS Channel Estimation
Resource Block De-mapping
CP Removal and FFT

sends NACK to the transmitter.

So, as we see, this is not a very optimized way of doing the data transmission. And we are also not fully utilizing the Code block level CRC that we have, because even if one out of 15 CB fails, we have to retransmit the full 1,00,000 bits here. So, instead of this, we can utilize the Code block level CRC and we can do ACK/NACK based on the code blocks. Lets understand this better with the below diagram.



Here, the TB goes through the main CRC attach, then further divided into multiple CBs. Then we add CRC to these CBs.

At the receiver, we remove the CB CRC and concatenate the multiple CBs into single TB. Then we remove the TB CRC. Finally we get the TB.

At the receiver, the CB colored red failed to decode. Because of this single CB failure, we have to retransmit the full TB again.

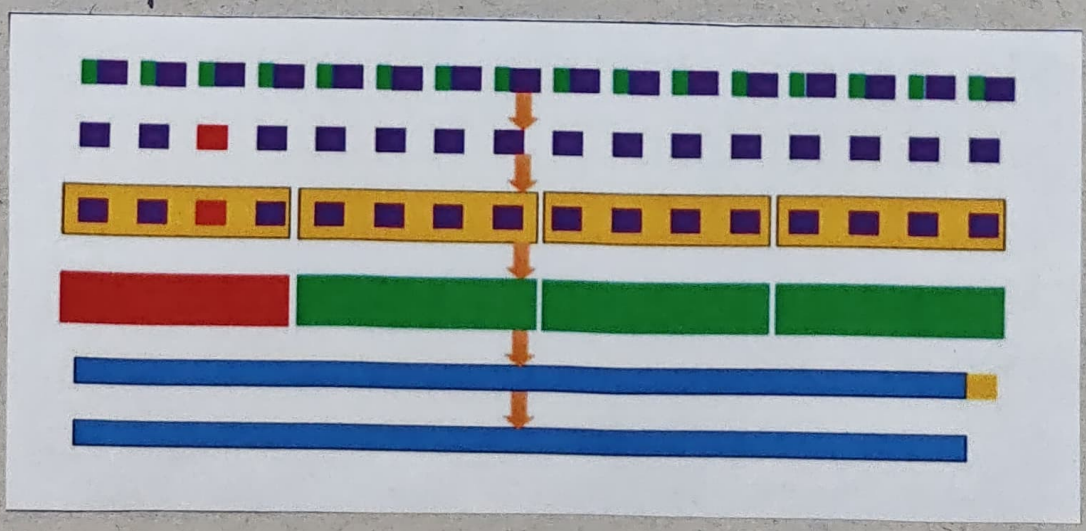
This can be optimized, by sending ACK/NACK for each Code block. In this figure, we have 16 code blocks. So

we send ACK/NACK for all the 16 CBs to the transmitter (1101111111111111). And the transmitter will find out that, out of 16, one particular CB failed and it will only retransmit that particular CB. This way, we can save the resources. But there is a problem here.

The problem is, the OVERHEAD of PDCCH (for DL ACK/NACK) and PUCCH (for UL ACK/NACK) increases heavily here. (ie) Instead of sending one ACK/NACK for the TB CRC, we have to send 16 ACK/NACKs for this particular case. The ACK/NACKs may go 50 (or) 60 based on the TB size. (ie) As the TB size increases, No. of CB increases, and the overhead of ACK/NACK also increases. So, this is also not a fully optimized way.

As we know, we always target 10% of BLER. (ie) more than 10% of TB should not fail. Thats how we keep the No. of layers / MCS/... in a way that, not more than 10% of the blocks should fail. So, on an average, 9 out of 10 TBs should always pass. And for those, we only need one ACK/NACK.

So, to do a fully optimized way, we can club the code blocks into a group, which is called as Code Block Group (CBG).



Here, instead of generating ACK/NACK for each CB, we group the CBs together. In this example, 4 CBs are grouped into one. This is called Code Block Group (CBG). Here the first CBG failed and the remaining three CBG has passed.

So here, instead of sending 16 ACK/NACKs, now we will send ACK/NACK for each CBG. So, the ACK/NACK would be of 4 bits (0111).

Also, when CBG based transmission is enabled, we dont send the final TB ACK/NACK.

Lets understand how to group the CBs into CBGs from the Specification itself.

5.1.7.1 UE procedure for grouping of code blocks to code block groups

If a UE is configured to receive code block group (CBG) based transmissions by receiving the higher layer parameter *codeBlockGroupTransmission* for PDSCH, the UE shall determine the number of CBGs for atransport block reception as

$$M = \min(N, C),$$

where N is the maximum number of CBGs per transport block as configured by *maxCodeBlockGroupsPerTransportBlock* for PDSCH, and C is the number of code blocks in the transport block according to the procedure defined in Subclause 7.2.3 of [5, TS 38.212].

Define $M_1 = \mathrm{mod}(C, M)$, $K_1 = \left\lceil \frac{C}{M} \right\rceil$, and $K_2 = \left\lfloor \frac{C}{M} \right\rfloor$.

If $M_1 > 0$, CBG m, $m = 0,1,\ldots,M_1 - 1$, consists of code blocks with indices $m \cdot K_1 + k, k = 0,1,\ldots,K_1 - 1$. CBG m, $m = M_1, M_1 + 1,\ldots,M - 1$, consists of code blocks with indices $M_1 \cdot K_1 + (m - M_1) \cdot K_2 + k, k = 0,1,\ldots,K_2 - 1$.

Above snippet shows the grouping of code blocks into code Block Groups. Lets understand this with an example.

Let take a TB size of 3,19,784 bits.

Adding TB-CRC of 24 bits.

$\Rightarrow$ 3,19,784 bits + 24 bits

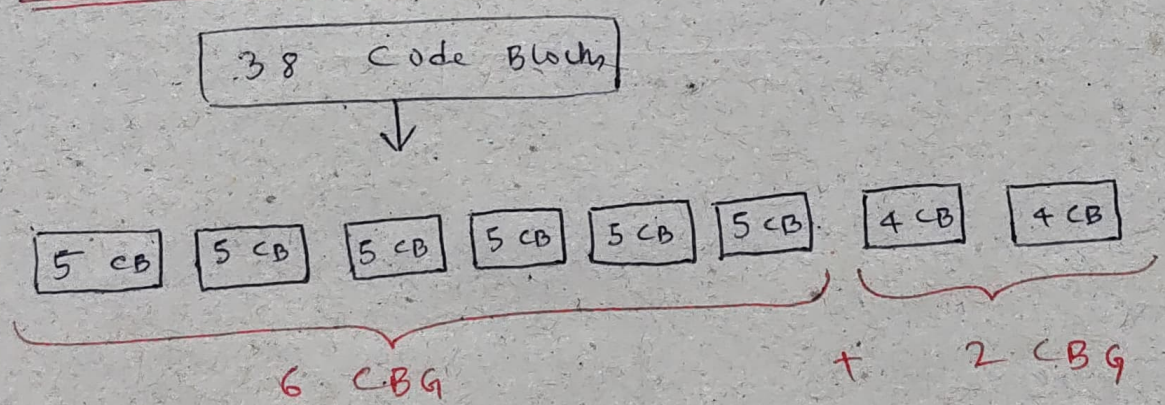
= 3,19,808 bits.

Now, this is divided into 38 code Blocks.

Now, we divide the code blocks into two different sizes. K_1 and K_2. As per the spec, the No. of CBG is 2/4/6/8. For our example, we take No. of CBG = 8.

$$C = 38$$
$$N = 8$$
$$M = \min(N, C) = 8$$
$$M_1 = \mathrm{mod}(C, M) = \mathrm{mod}(38, 8) = 6$$
$$K_1 = \left\lceil \frac{C}{M} \right\rceil = \left\lceil \frac{38}{8} \right\rceil = 5$$
$$K_2 = \left\lfloor \frac{C}{M} \right\rfloor = \left\lfloor \frac{38}{8} \right\rfloor = 4$$

So, basically what these formulas doing is that, we are supposed to divide 38 code Blocks into 8 different Code Block Groups (CBGs).



This is how the division is happening.

For each of the 8 CBG, we give the ACK/NACK to the transmitter.

CBGFI — Code Block Group Flush Indicator

As we've seen in the previous section, that the receiver may or may not combine the received soft information with the previous information in the Rate Matching. (ie) the reciver may / may not do soft combining.

The CBGFI indicates whether we need to flush the previous data or not. So, this is basically, in downlink, the gNB can indicate to the UE that it is retransmitting data for a particular CBG, and it can ask UE to flush the previous soft data from its buffer and then use only this new packets for decoding.

HARQ : Codebooks for PDSCH ACK / NACK

Lets understand the codebook for PDSCH HARQ. Basically, codebooks tells us how the ACK/NACK, are planned for the PDSCH. In which slot, the ACK/NACK for the PDSCH (DL) will go through the PUCCH/PUSCH (UL), and how the ACK/NACK is packed in the UL channel. Lets understand this with an example.

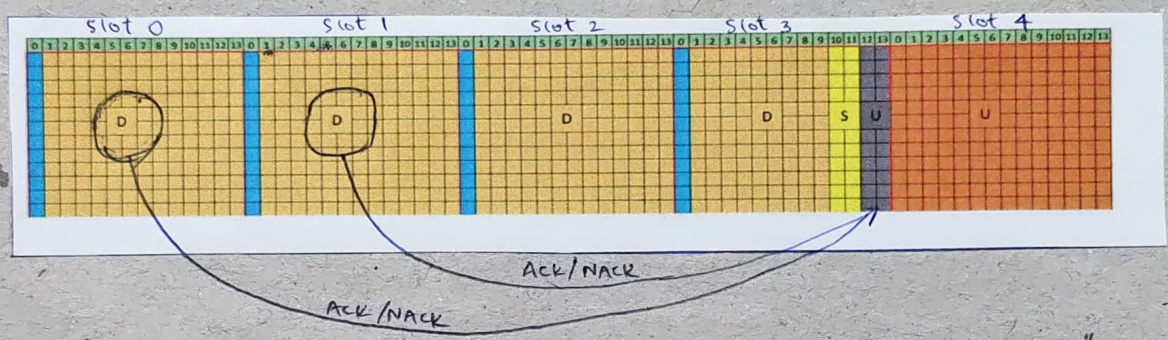


Figure shows the TDD Frame structure "DDDSU". As we see, we have 10D/2S/2U symbols in S-slot, 3 full DL-slots and 1 full UL-slot.

Assume, we want to schedule ACK/NACK in 2 U symbols of S-slot. Though the ACK/NACK can be planned in UL-slot as well (depending on the PUCCH format we choose). For this example, we assume the ACK/NACK to be scheduled in S-slot.

Now, lets take the DL symbols of slot 3. Now, can the UE send the ACK/NACK for these DL symbols in the UL symbols of same slot (slot 3)? This depends on the UE-capability. (i) Can the UE really process the DL symbols of slot 3 and schedule the ACK/NACK at the same slot (slot 3). (ii) The UE processing involves decode the PDCCH to get the DCI, followed by decode the PDSCH to process the data, followed by send the ACK/NACK with in the same slot (slot 3). There is also switching delay in slot 3. Also, there would be propagation delay (i.e. UE may receive data after some time depending on how far it is from gNB). And similarly, UE have to send UL data in advance. So, UE can't

possibly process all of the above in single slot duration. So, UE cant possibly schedule the ACK/NACK for the DL data in Slot 3 in the UL symbols of slot 3 itself.

Now, lets take the DL symbols of Slot 2. For this DL slot, it again depends on the UE capability. (ie) Can the UE be able to process the data in DL symbols of Slot 2 and schedule the ACK/NACK at the next slot (slot 3) itself. For our example, lets assume that the UE may not be able to send the ACK/NACK in the UL symbols of slot 3, by processing the data in the DL symbols of slot 2.

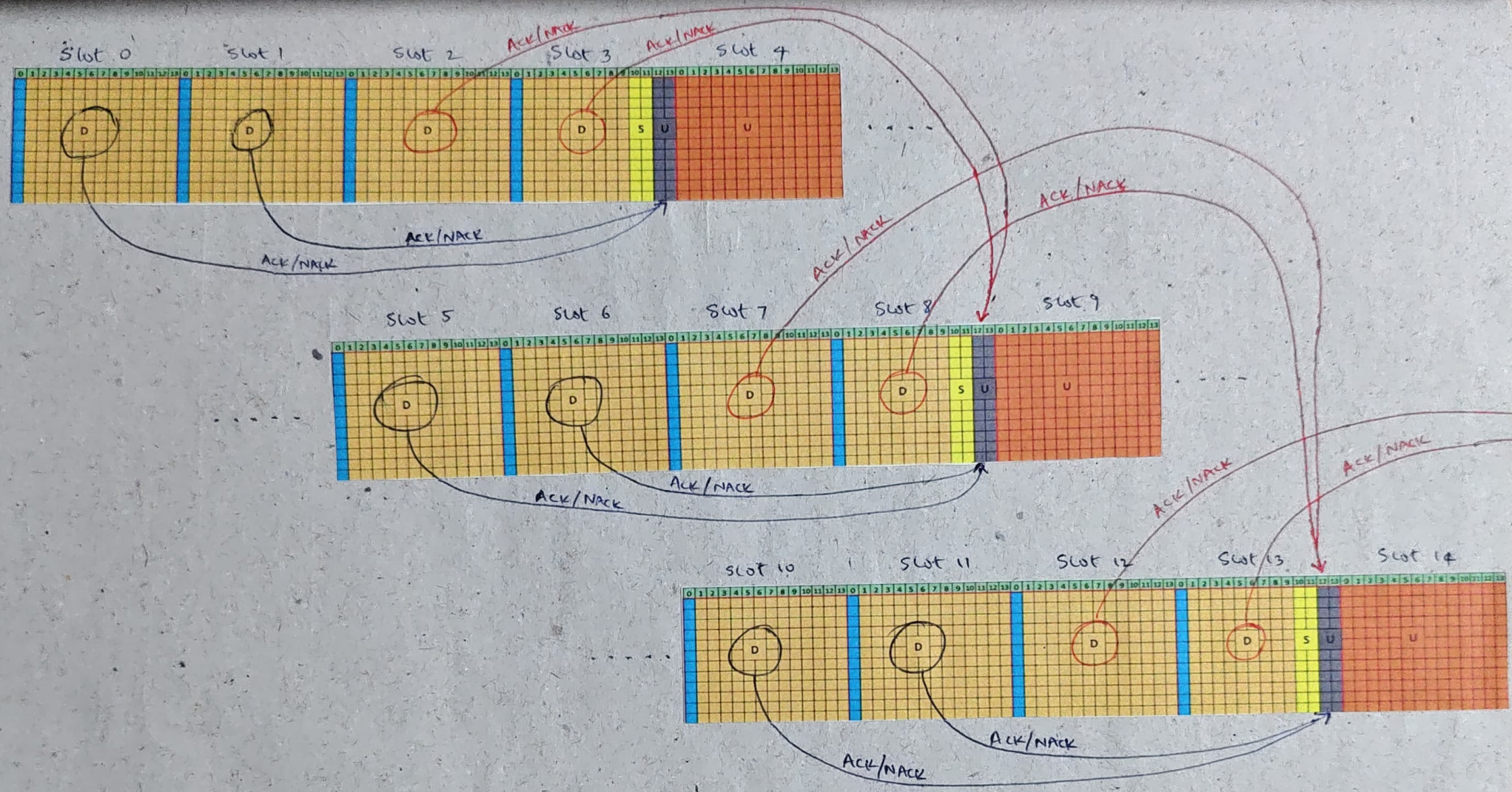
Now, lets take the DL symbols of Slot 1 and Slot 0. Lets assume that the UE can comfortably process and schedule the ACK/NACK as well in the UL Symbols of Slot 3.

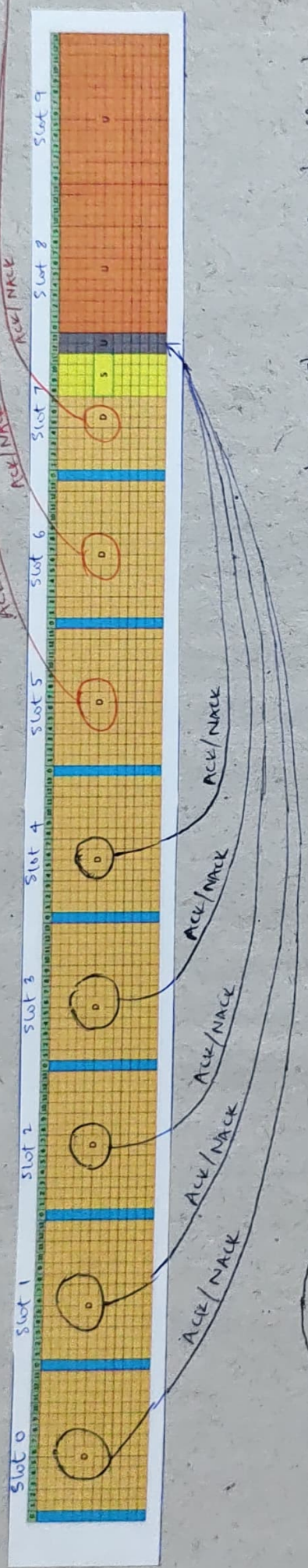
So, the ACK/NACK for DL symbols of slot 0 and Slot 1, goes via the UL symbols of slot 3.

Now, what about the ACK/NACK for the DL symbols of slot 2 and Slot 3? Where will it go?

If we look into the bigger picture, the ACK/NACK for the DL symbols of Slot 2 and Slot 3 goes via the next scheduled PUCCH resources (ie) UL symbols of slot 8. In this bigger picture, we see that, the scheduled PUCCH resources (ie) UL symbols of Slot 8 and Slot 13 carries the ACK/NACK's of the previous DL symbols of (Slots 2 and 3) and (Slots 7 and 8). This is what the codebook gives.

The codebook gives a planning on how the ACK/NACK's are mapped with the Downlink transmission.


Slot 0
Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
Slot 7
Slot 8
Slot 9
Slot 10
Slot 11
Slot 12
Slot 13
Slot 14
D
S
U
ACK/NACK



Lets take another example, with different TDD Frame structure "DDDDDDDSUU". In S-slot, we assume 6 DL symbols, 4 symbols for switching and 2 UL symbols. In this case also, assume PUCCH that is scheduled in slot 7 UL symbols carries the ACK/NACK.

Lets also assume, the ACK/NACKs for DL symbols in slots 5 and 6, cannot go via slot 7 UL symbols. And the ACK/NACKs for DL symbols in slot 0, 1, 2, 3, 4 are scheduled to go via slot 7 UL symbols.

Now the question comes, how does the ACK/NACKs are mapped to the PUCCH resources? And this is defined in the codebook.

In this example, we have 5 DL slots of ACK/NACK. Lets assume there are 4 UEs that are multiplexed either in frequency/time/MIMO in one DL slot. Lets assume it is a non CBG based transmission, hence we need only 1 bit to send. Lets also assume, the No. of TBs per UE is ≤ 4, hence we need only 1 bit to send.

So, $5 \times 4 \times 1 \times 1 = 20$ HARQ bits (or) ACK/NACKs has to be planned in this scenario.

Note: If the No. of TBs per UE is > 4, then we need 2 bits.

There are 2 types of codebooks defined...

- Type-1 → Semi-static
- Type-2 → Dynamic.

In the semi-static case, we fix the resources for HARQ (ACK/NACK) irrespective of whether there is data transmission or not.

Consider gNB sent (PDCCH + PDSCH) to the UE, and the UE fails to decode the PDCCH itself. Here the UE doesn't even know whether the gNB has sent the PDSCH or not, but the gNB is expecting ACK/NACK from the UE.

Keeping such problems in mind, Codebooks are defined. So, in semi-static case, UE have to send NACK for this scenario. Whereas in dynamic case, UE may not have to send NACK, instead gNB detects the DTX based on the transmission. Upto 4 DTX can be detected in dynamic case.

Whereas, in semi static case, whether the gNB sent the PDSCH or not, whether the UE decoded PDCCH or not, it has to allocate the resources and UE has to schedule the ACK/NACK.