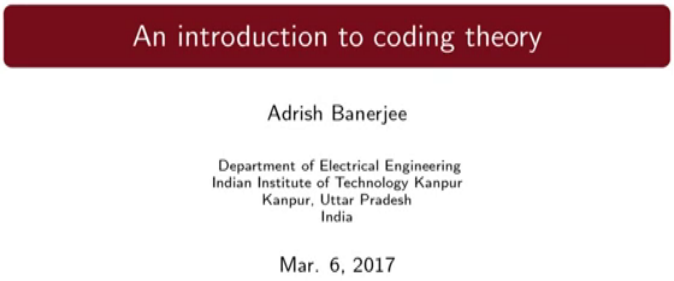
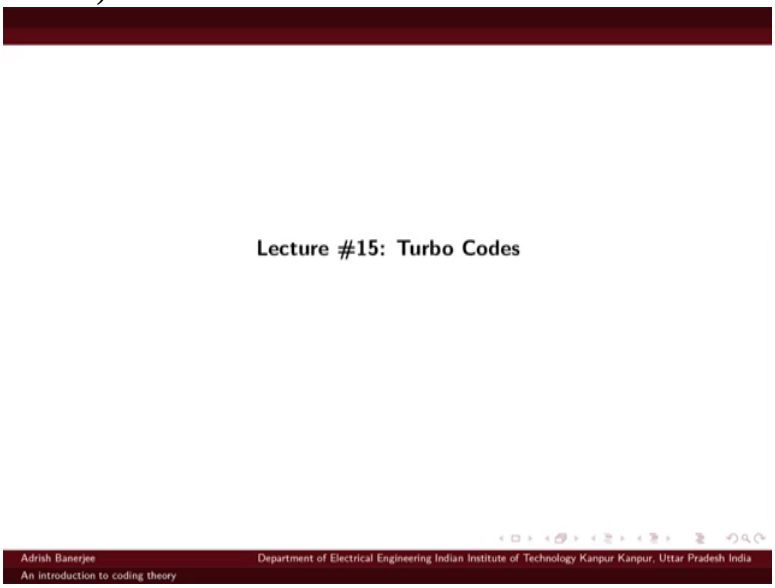
**An Introduction to Coding Theory**
**Professor Adrish Banerji**
**Department of Electrical Engineering**
**Indian Institute of Technology, Kanpur**
**Module 07**
**Lecture Number 27**
**Turbo Codes**

(Refer Slide Time 00:13)




So today we are going to

(Refer Slide Time 00:15)

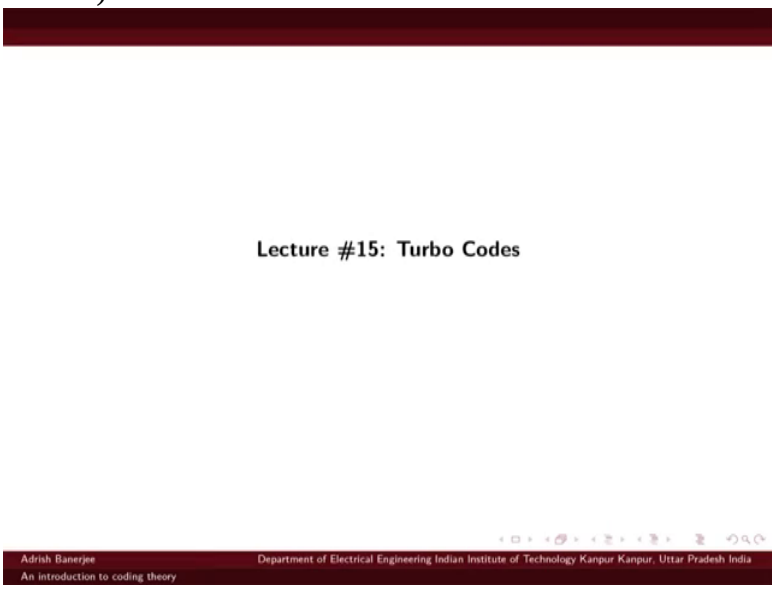



talk about a class of parallel

(Refer Slide Time 00:18)

concatenated convolutional code called turbo codes.

(Refer Slide Time 00:23)

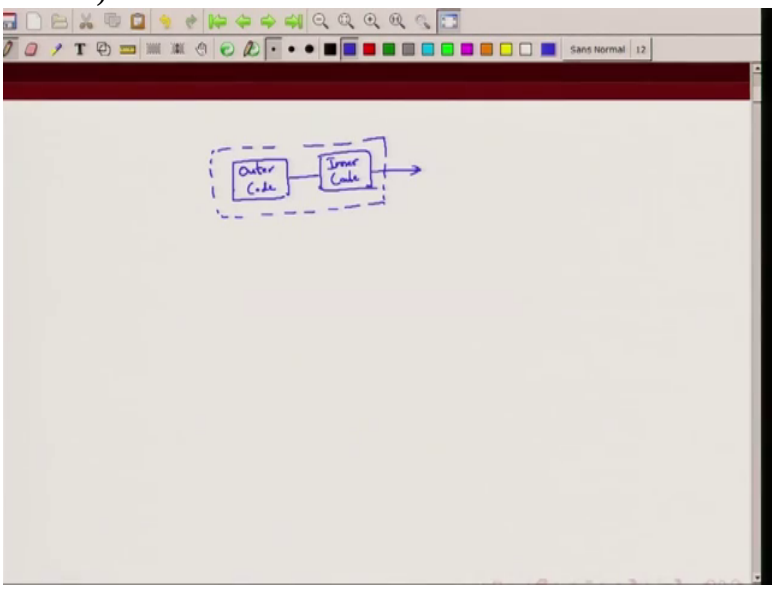



Now what is concatenation? In

(Refer Slide Time 00:26)

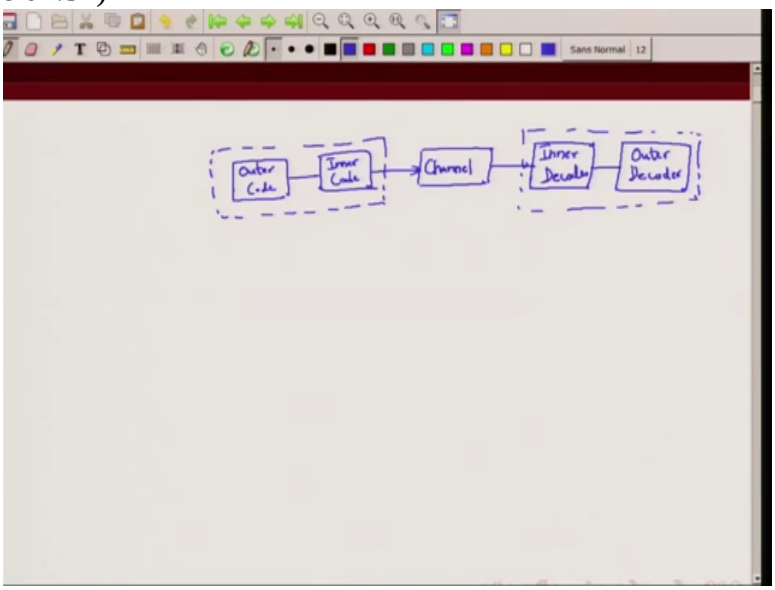
concatenation we take small codes and we combine them to create a more powerful code. Now this idea of concatenation was proposed by David Forney and what he did was he took two codes, so there was one code called outer code and then he had another code called inner code and, so this is your, you are creating a more powerful code using 2 codes

(Refer Slide Time 00:59)



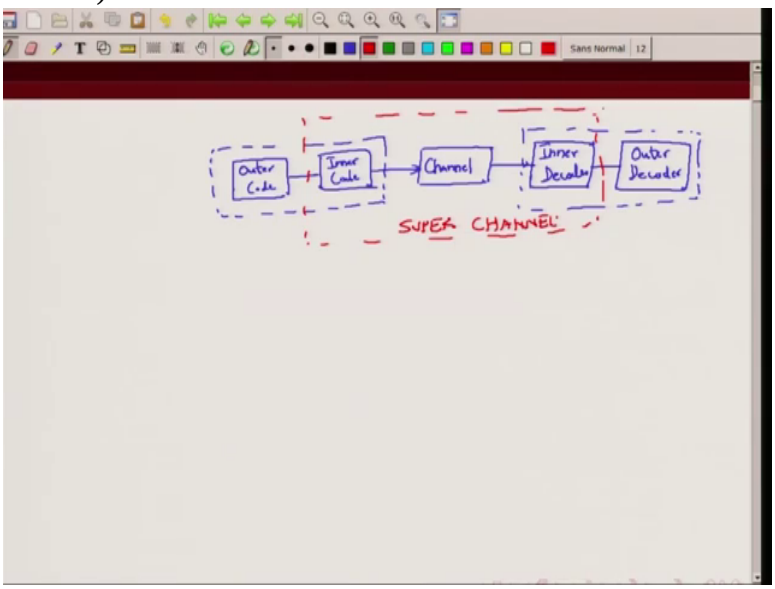

and you send your coded bits through a channel and you have the decoder for inner code. So you have inner decoder and then you have outer decoder. So this is your,

(Refer Slide Time 01:32)




I could also view this as, this as your super channel, this as your super channel. These are codes on that.

(Refer Slide Time 01:47)

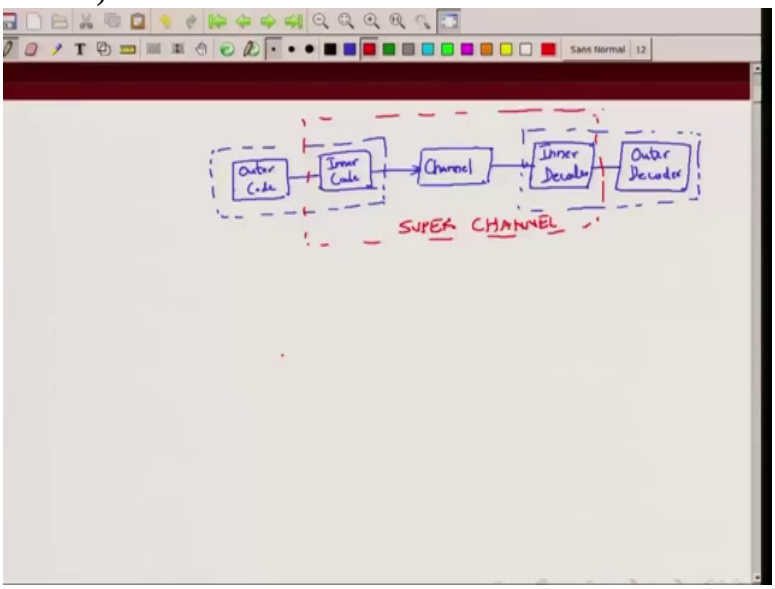



Now this is a very simple idea of combining 2 relatively simpler code

(Refer Slide Time 01:54)

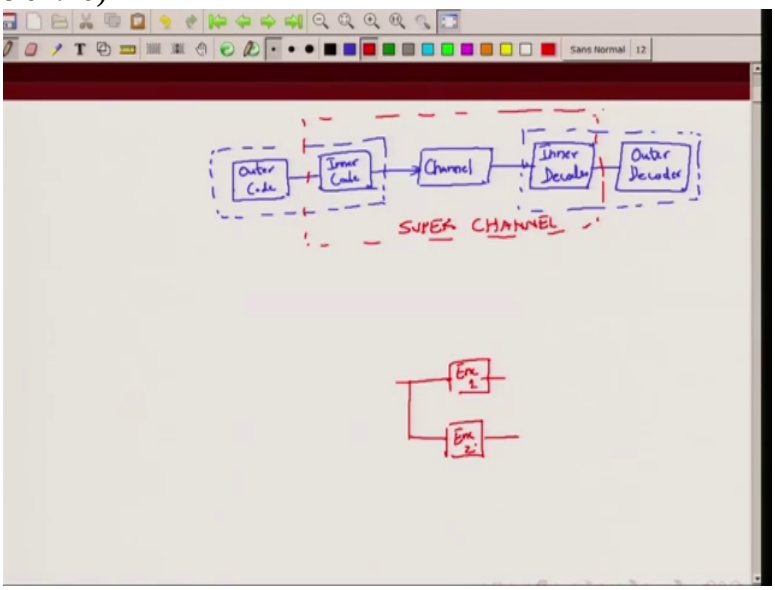
to create a more powerful code and these codes

(Refer Slide Time 02:01)



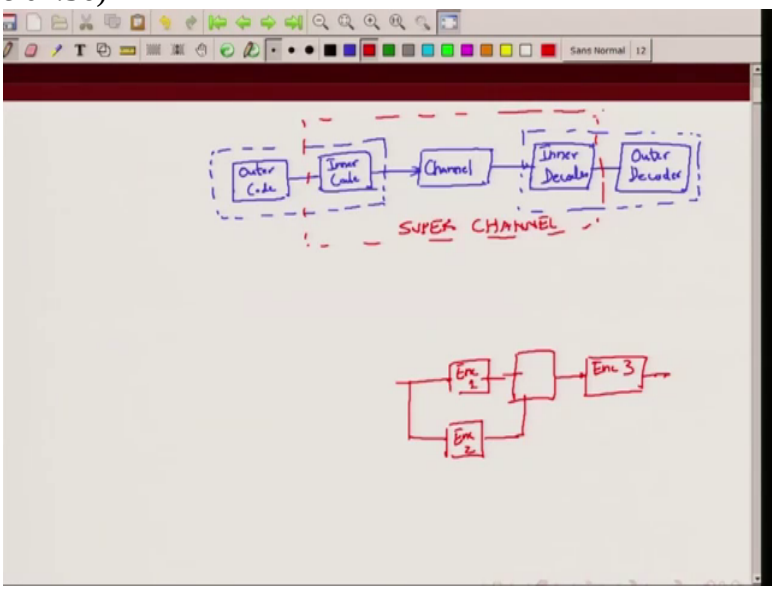

could be block codes, convolutional codes. You could use any one of them. Now this is one class of concatenated codes. There are various ways in which you could concatenate code. This is an example where two codes are serially connected. If you see, the output of the first encoder is given as input of second encoder. Now there could be a configuration where 2 encoders are connected in parallel, so you have 2 encoders. They are connected in parallel. This is your encoder 1, encoder 2, could have like this.

(Refer Slide Time 02:40)




Or you could have a combination of both. So you have some parallel concatenation, (()) is a combiner, and then you have another code, encoder 3 which is serial. So this is combination of parallel concatenation and

(Refer Slide Time 02:56)

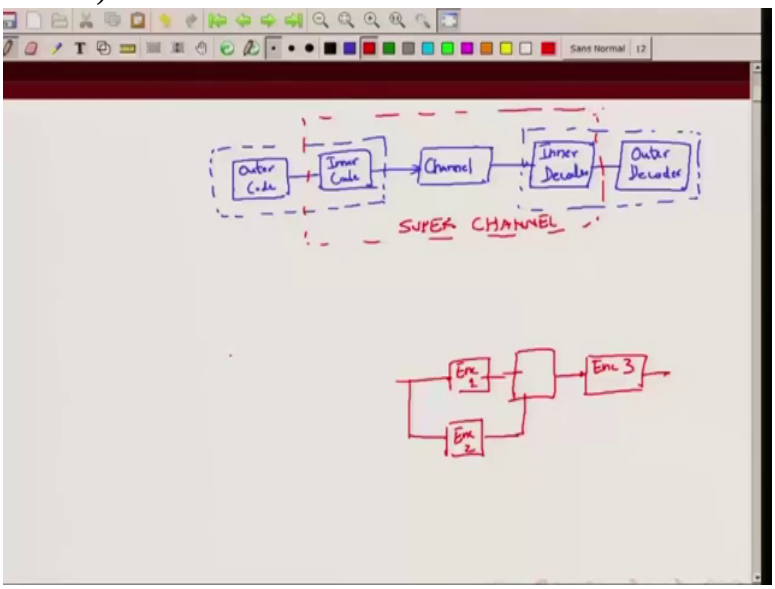



serial concatenation. This is like

(Refer Slide Time 02:59)

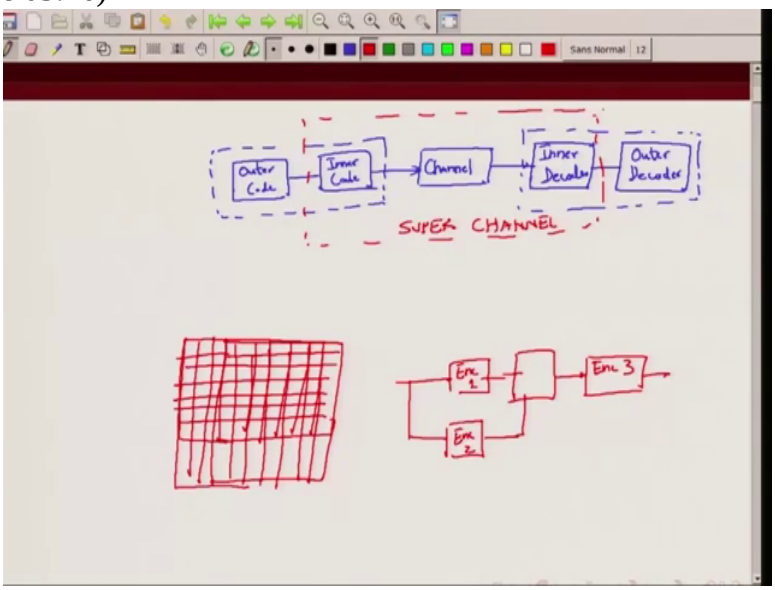
hybrid concatenation. So there are various ways in which you can concatenate two codes. There is

(Refer Slide Time 03:05)




a configuration which is called a product code. So you have a, let's say k cross k information bits, you add some parity bits here. Corresponding to each of these k bits you add some parity bits. And then you can read this information sequence column wise also. And you can add parity bits corresponding to these columns. So you can have codes like this. This is called incomplete product code. And then you could also have checks on checks, so that's a product code. So there are various ways in which you could concatenate

(Refer Slide Time 03:46)

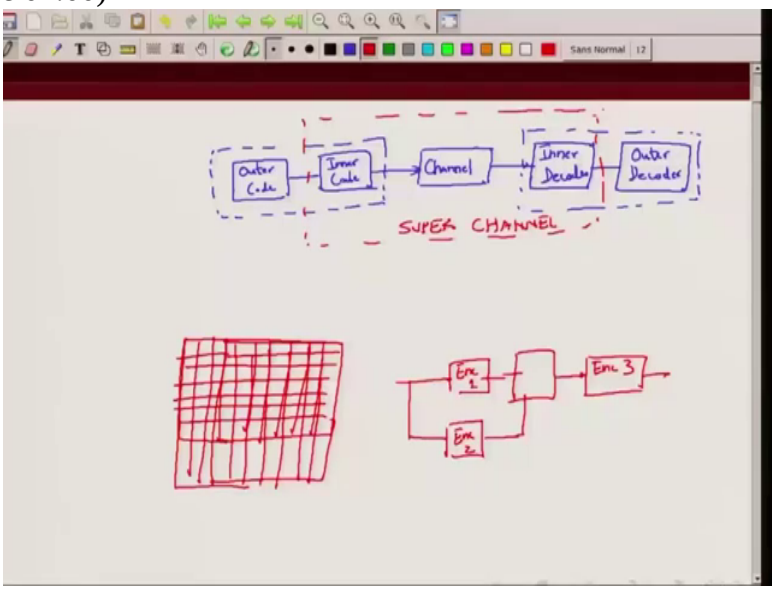



or combine 2 smaller codes to create a more powerful code. In this lecture we are going

(Refer Slide Time 03:53)

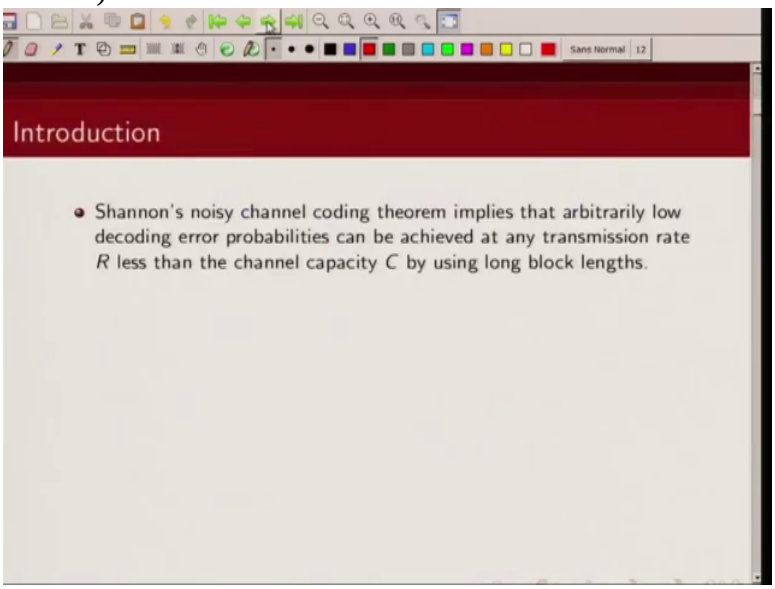
to talk of one such class of concatenated codes which is called turbo codes.

(Refer Slide Time 04:00)



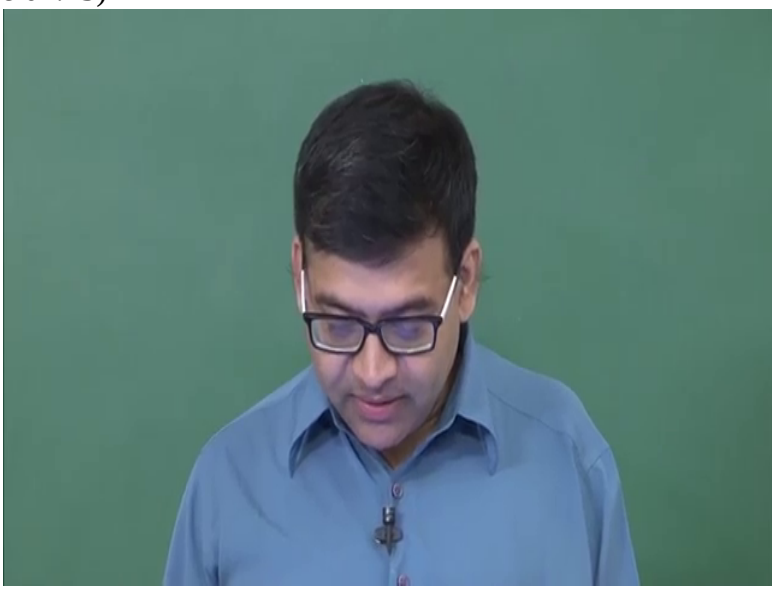

It's essentially a class of parallel concatenated codes.

(Refer Slide Time 04:05)



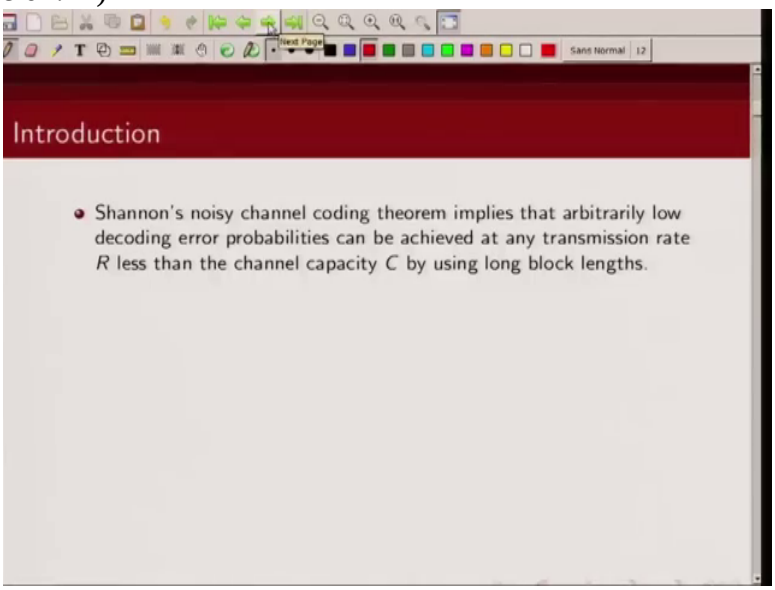

Now if you recall, Shanon in his celebrated Noisy Channel Coding Theorem has mentioned that as long as we choose our transmission rate to be below channel capacity, we can reliably communicate over a communication link.

(Refer Slide Time 04:23)



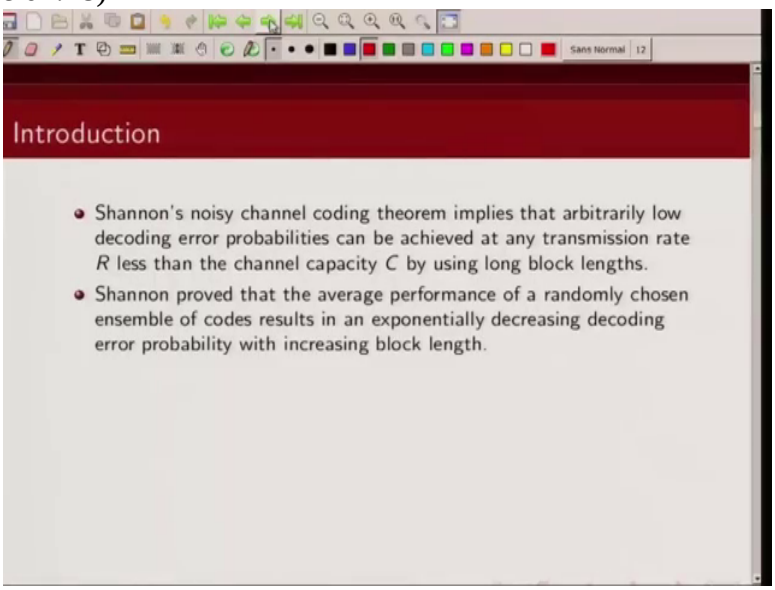
And his proof was basically based on fact that if you randomly select a codeword of very large length, then you can show that probability of error will go to zero as long as your transmission rate is below channel capacity and you transmit with long codewords. Now

(Refer Slide Time 04:47)



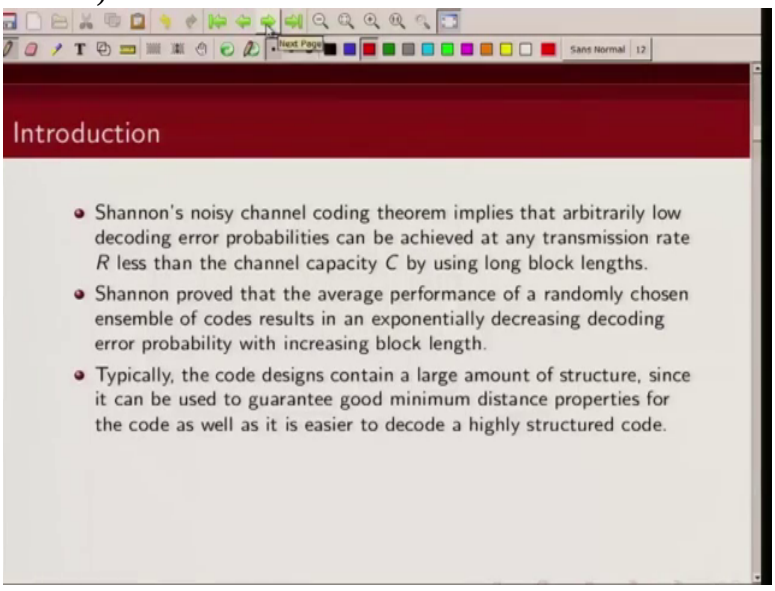

people

(Refer Slide Time 04:48)




from 1948 onwards were trying to design codes close to channel capacity. And the problem was if you design a very random like code which does not have any decoding structure then the decoding complexity is very very large. However if you put lot of structure into the code then it is not really random and its performance is not very good. So how do you

(Refer Slide Time 05:16)

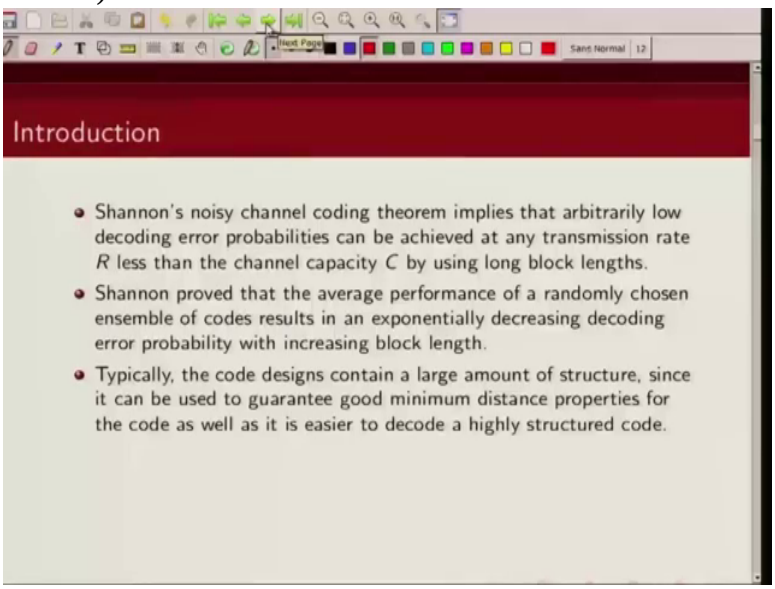



design a code which has enough randomness into it but then

(Refer Slide Time 05:21)

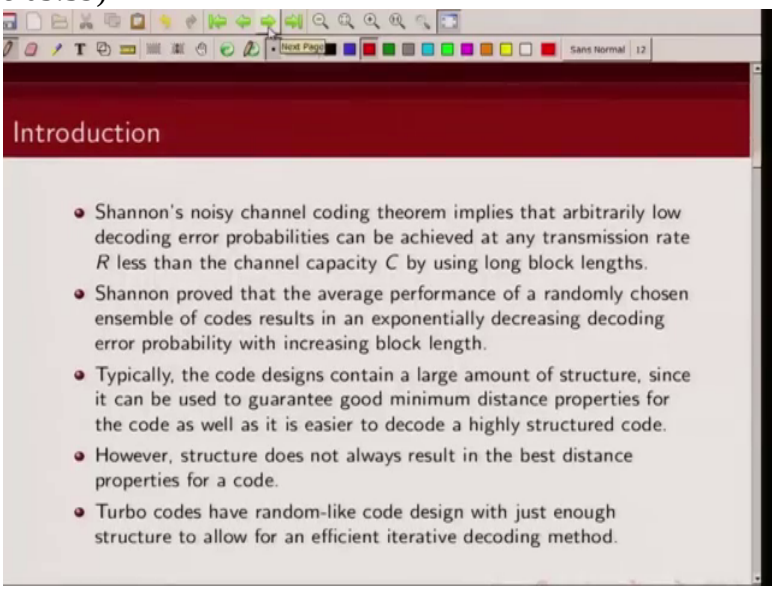
enough structure also which can be exploited for decoding the code? So that's basically the challenge

(Refer Slide Time 05:30)




to design a code which has good minimum distance but then we should be able to decode it also.

(Refer Slide Time 05:39)

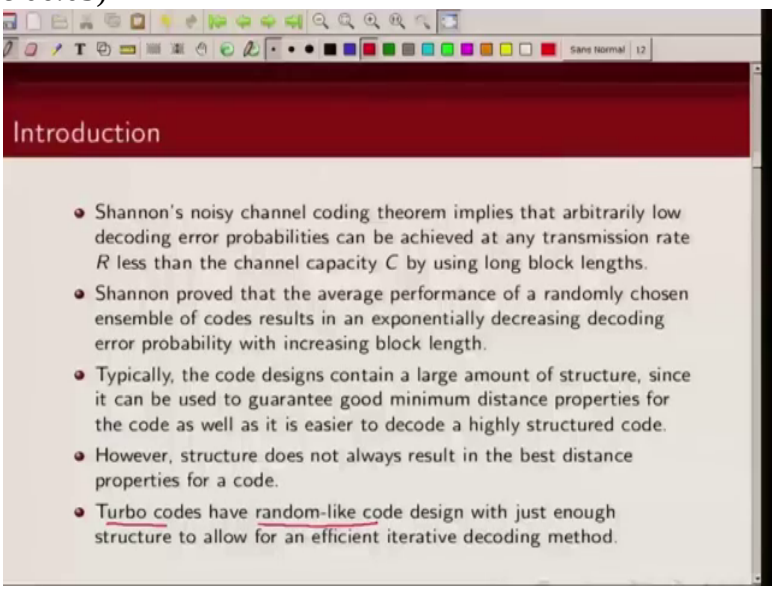



Now these turbo codes are class of codes which are random like and we will show you how these random look like because of a inherent interleaver structure fitted in this parallel concatenation structure and

(Refer Slide Time 05:57)

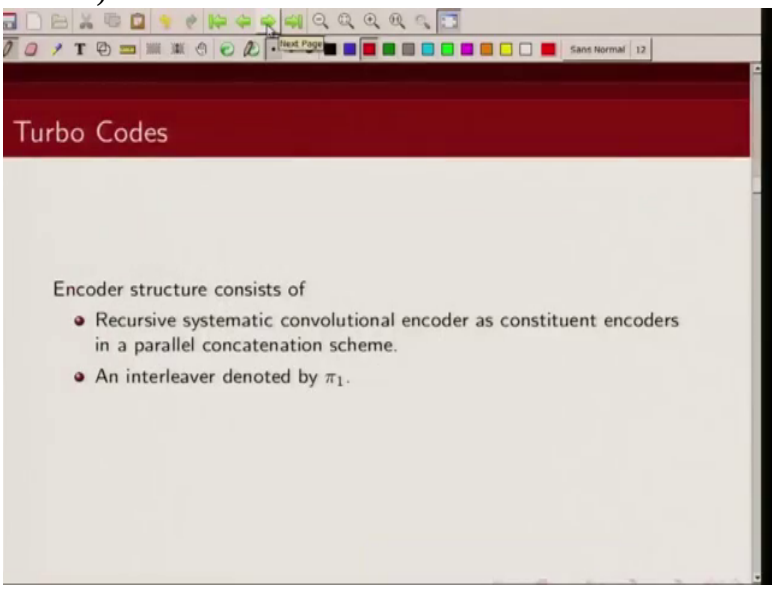
there is enough structure in the code which allows us to do efficient decoding of these

(Refer Slide Time 06:05)



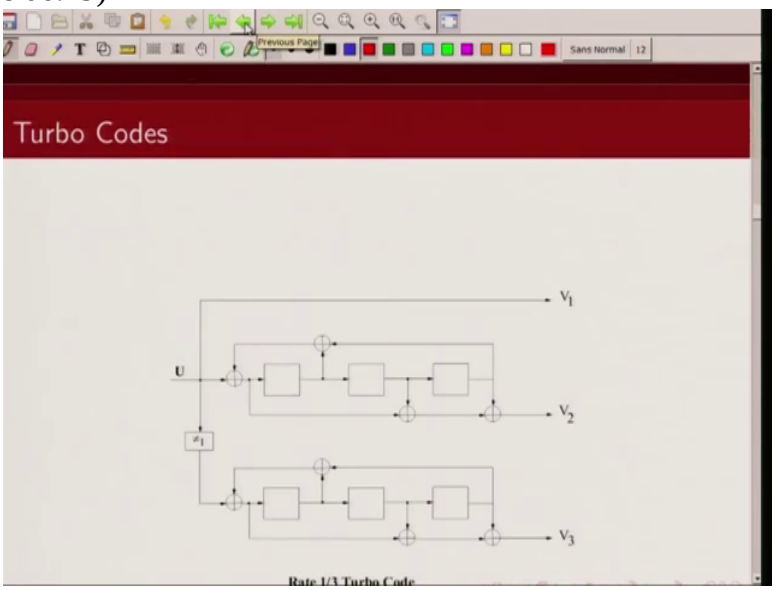

error correcting codes.

(Refer Slide Time 06:07)



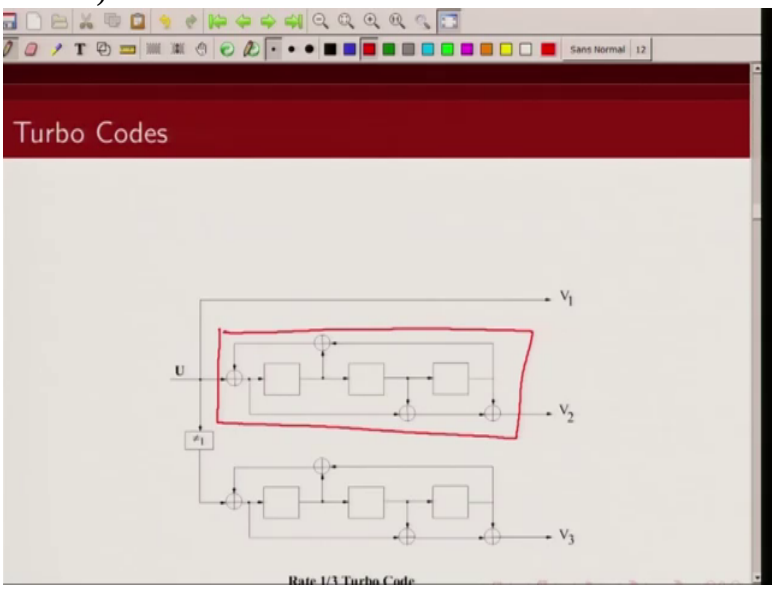

So what does an encoder of a turbo code looks like? It consists of parallel concatenation. May be first I will show you the diagram.

(Refer Slide Time 06:19)



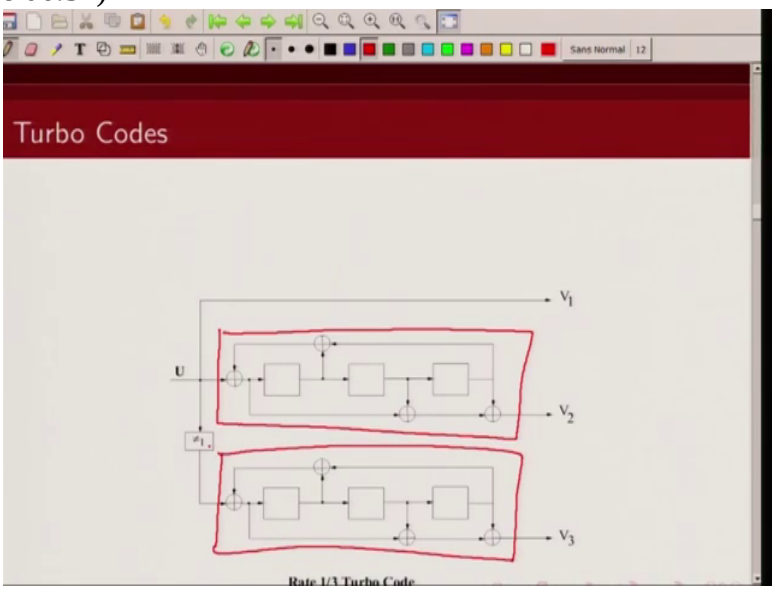

It consists of parallel concatenation of 2 encoders, this is one encoder,

(Refer Slide Time 06:26)



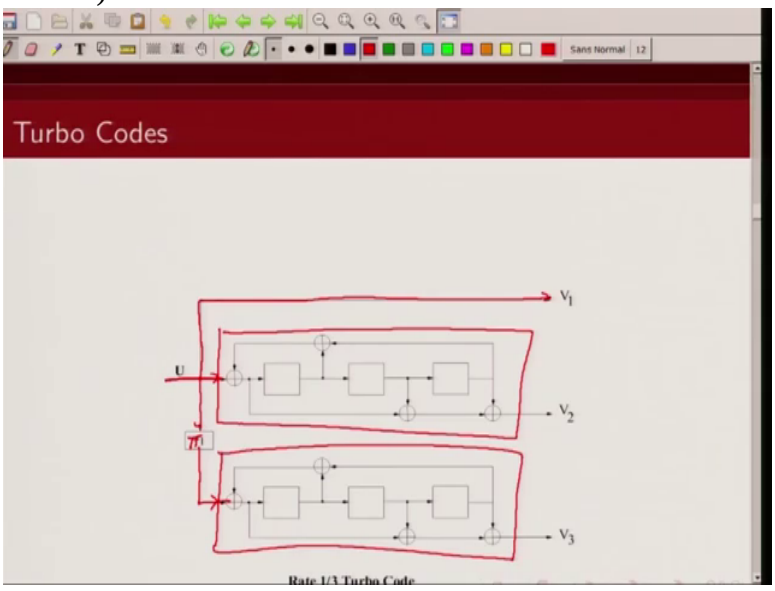

this is second encoder. And note

(Refer Slide Time 06:31)



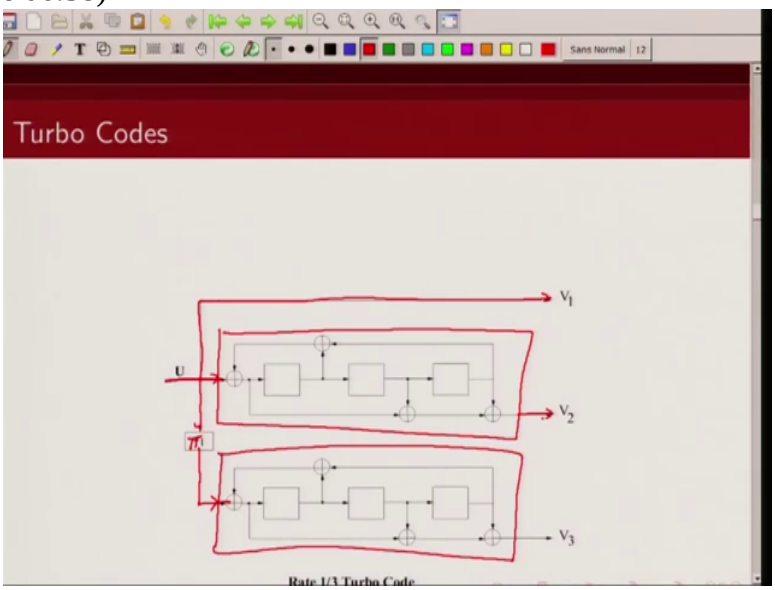

the same information bit is going to 2 encoders. So this information bit is coming here and the same information bit after getting interleaved, interleaved is nothing but reordering of the message bits, it is entering this particular encoder and this is your systematic code so you have your

(Refer Slide Time 06:54)



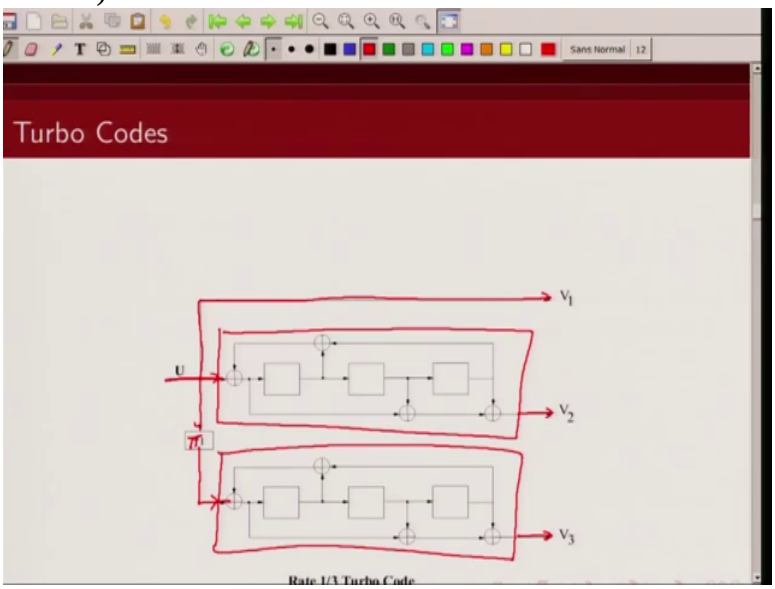

information bits here and this encoder is generating this

(Refer Slide Time 06:58)



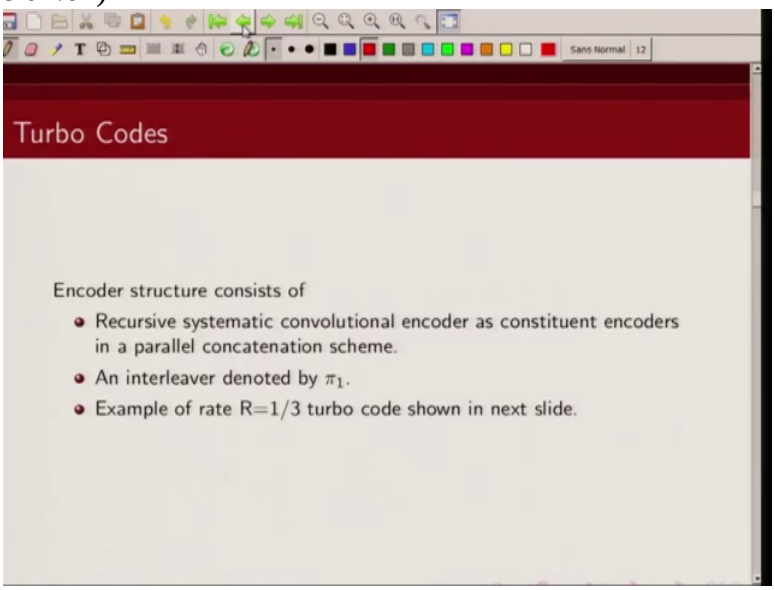

party bit, this encoder is generating

(Refer Slide Time 07:00)



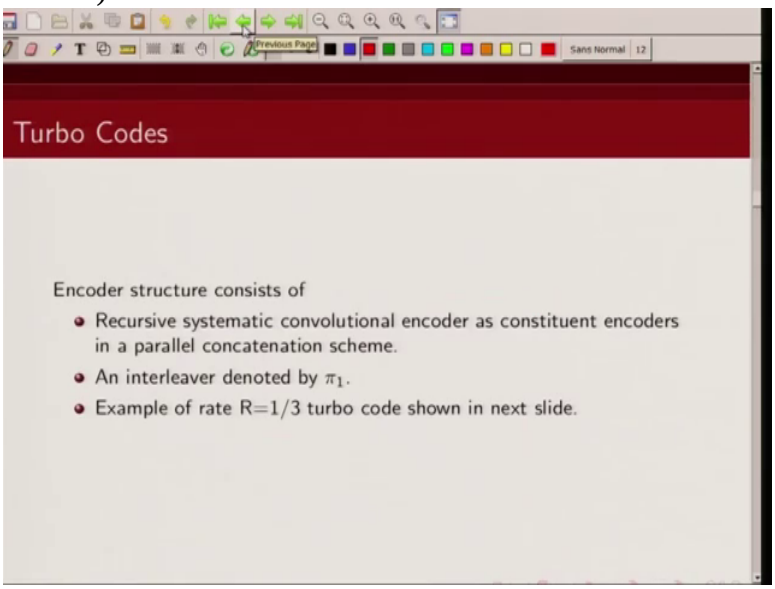

this parity bit.

(Refer Slide Time 07:02)




So

(Refer Slide Time 07:03)

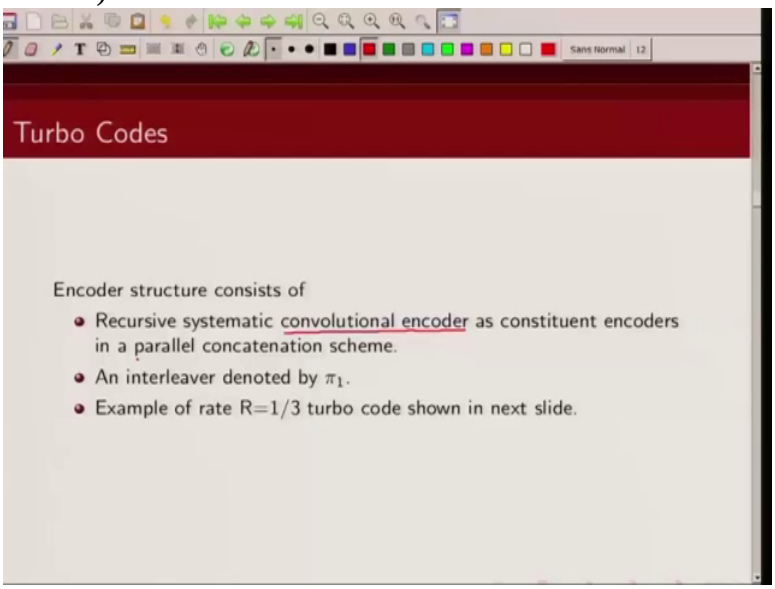



if you notice this encoder consists of convolutional encoder as a consistent encoder in a

(Refer Slide Time 07:12)

parallel concatenation scheme.

(Refer Slide Time 07:15)

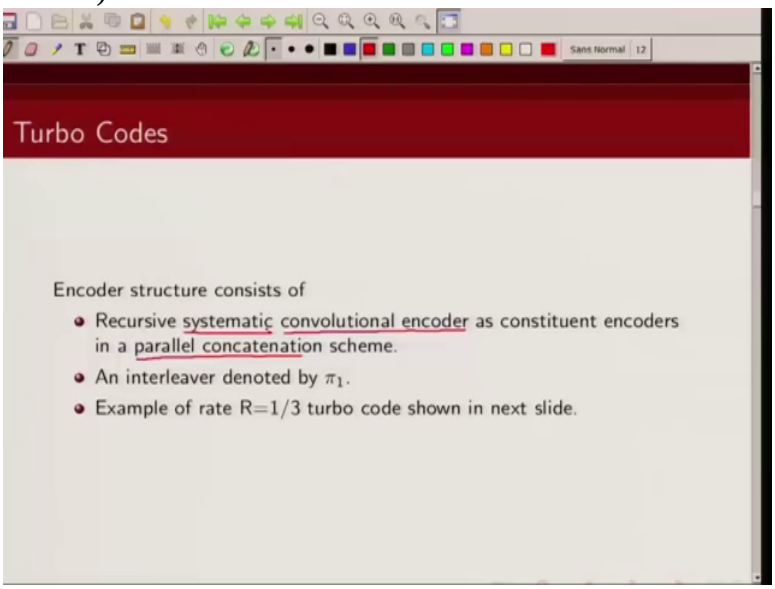



So the key is parallel concatenation. So the same input is going to both these convolutional encoders. Next thing to remember is we are using systematic

(Refer Slide Time 07:29)

convolutional encoder. Why are we using

(Refer Slide Time 07:33)

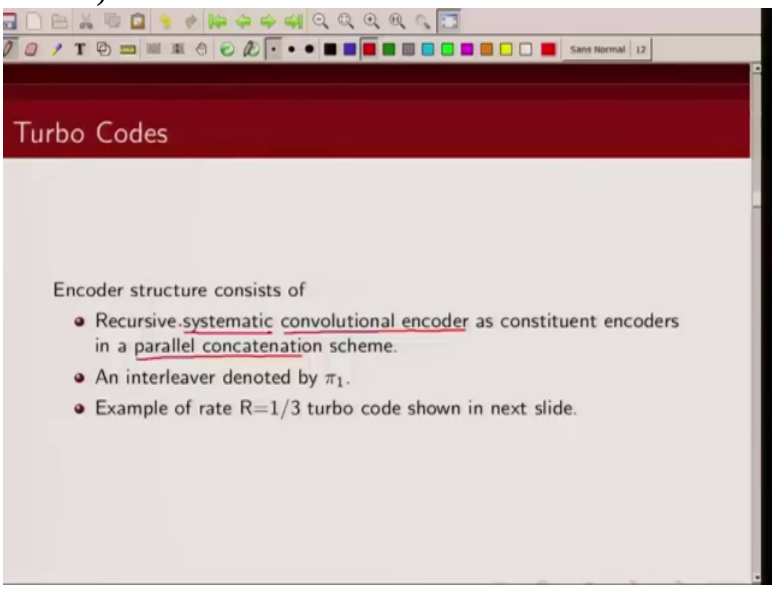



systematic convolutional encoder? When we talk of the convergence properties of turbo code then we will mention basically, these systematic encoders

(Refer Slide Time 07:43)

have better convergence properties under iterative decoding algorithm and that's why initially

(Refer Slide Time 07:51)

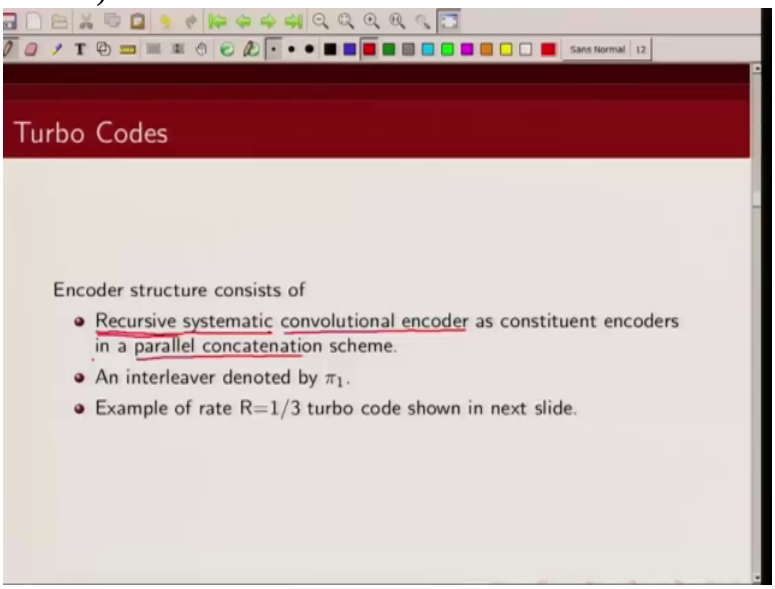



proposed turbo codes use systematic convolutional encoder. Subsequently people have also worked on designing, design of non systematic turbo codes but the ones which were initially proposed in 1993 by Berrou et al used systematic convolutional encoder. The third thing to note, very crucial is the use of recursive encoders.

(Refer Slide Time 08:20)

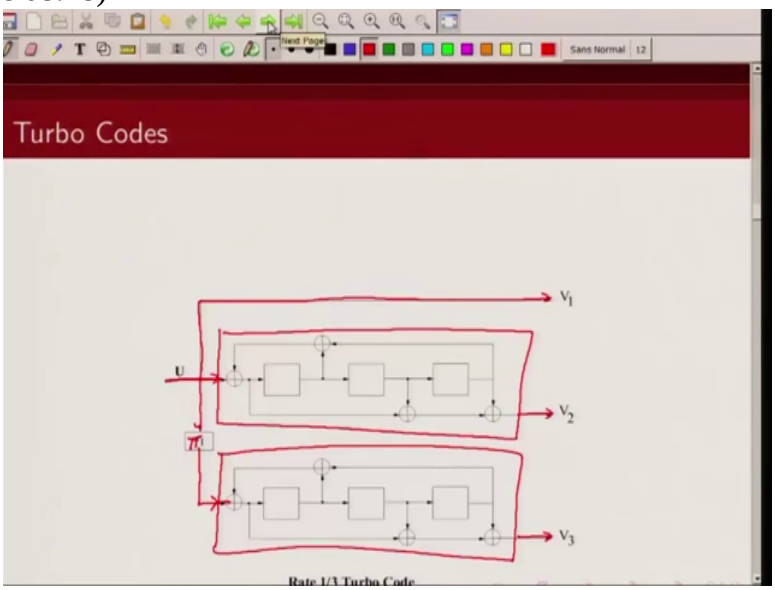
What are recursive encoders? These are feedback encoders. So there is a feedback from the

(Refer Slide Time 08:27)



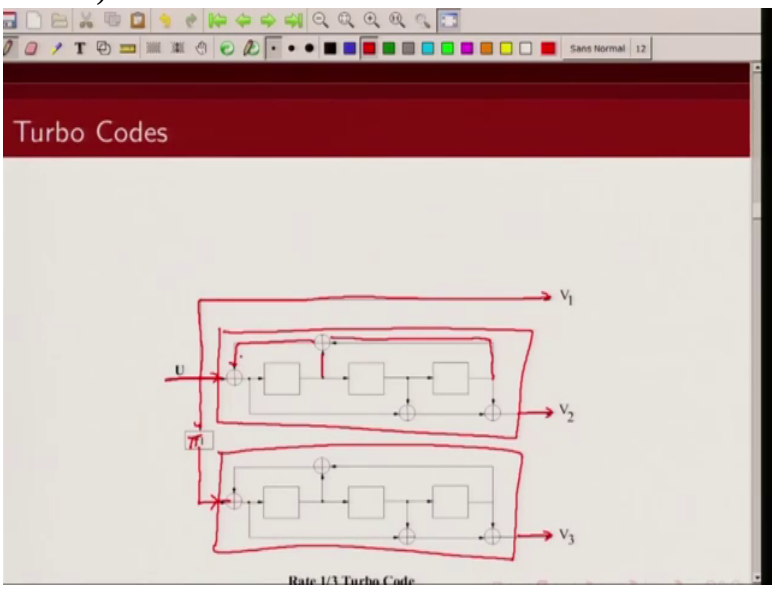

output

(Refer Slide Time 08:28)



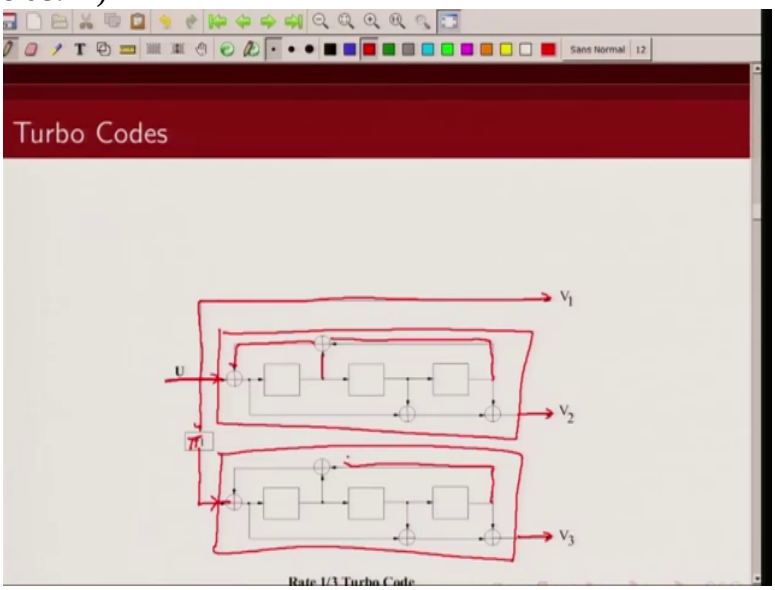

to the input. Note here there is a feedback from the output going to the input, there is a feedback from the output going to the input, so these

(Refer Slide Time 08:38)




are all feedback encoders. Now why do we need

(Refer Slide Time 08:42)

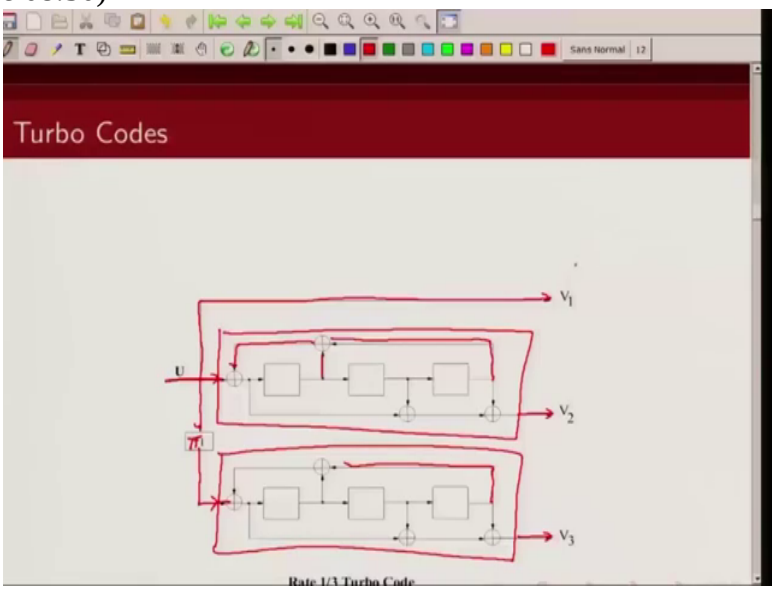



feedback encoders?

(Refer Slide Time 08:44)

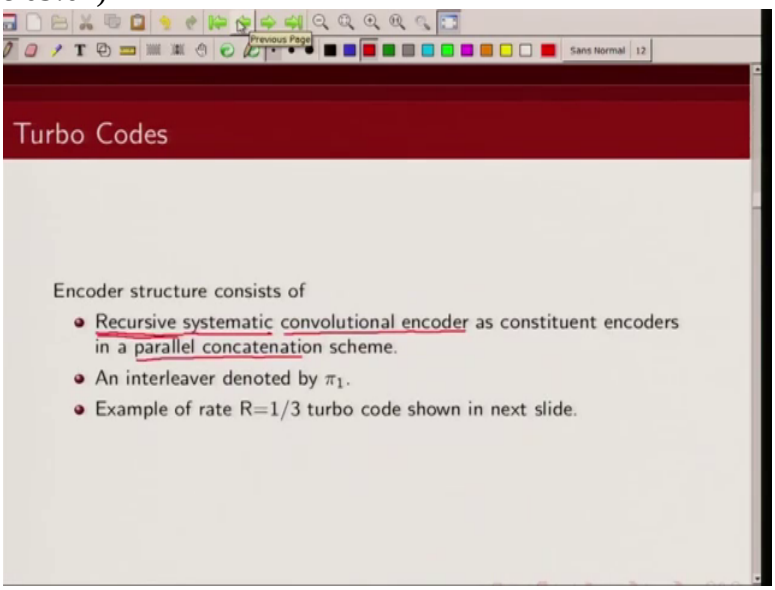
I will just give a very simple example to illustrate.

(Refer Slide Time 08:50)



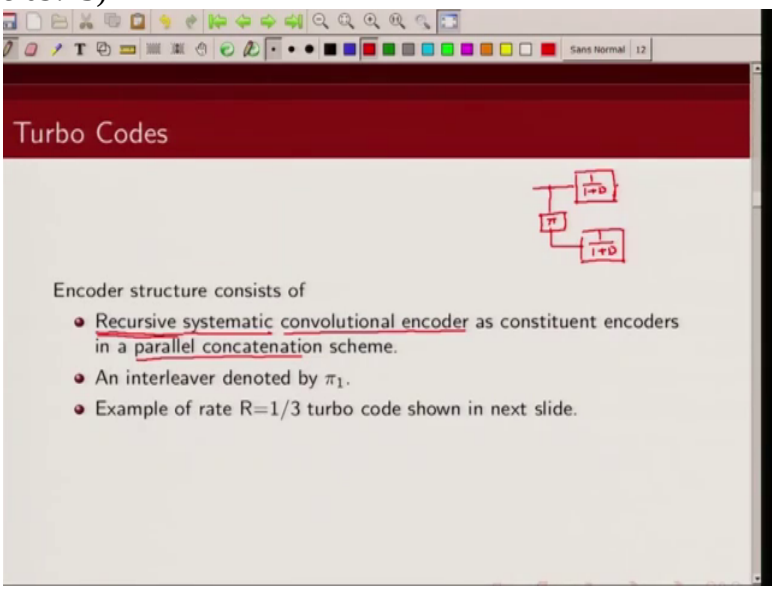

We know that we would like to have a large minimum distance of these codes. Because larger the minimum distance better is the error correcting capability of the code. So let's just consider a very simple case.

(Refer Slide Time 09:01)



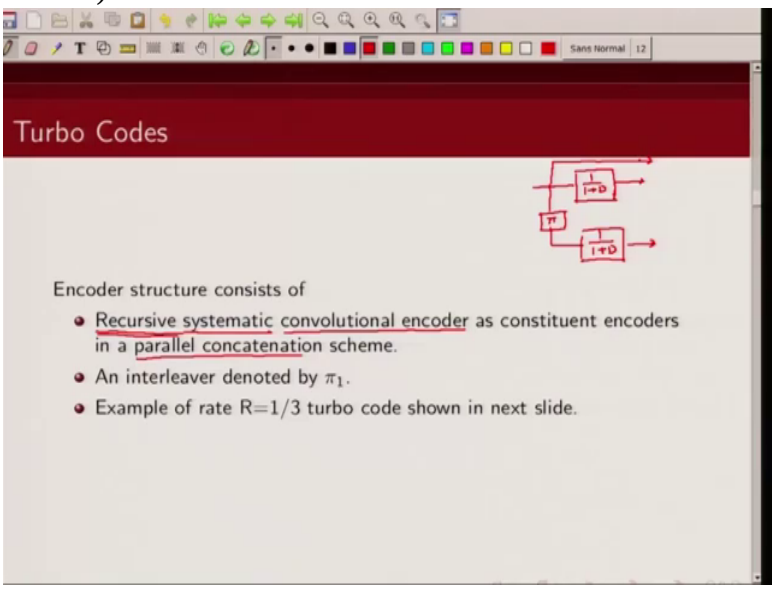

Let's just consider that you have a memory 1 code. So let's just see, memory 1 code would be g of d, so this could be let's say 1 plus d, 1, 1 plus d.

(Refer Slide Time 09:23)



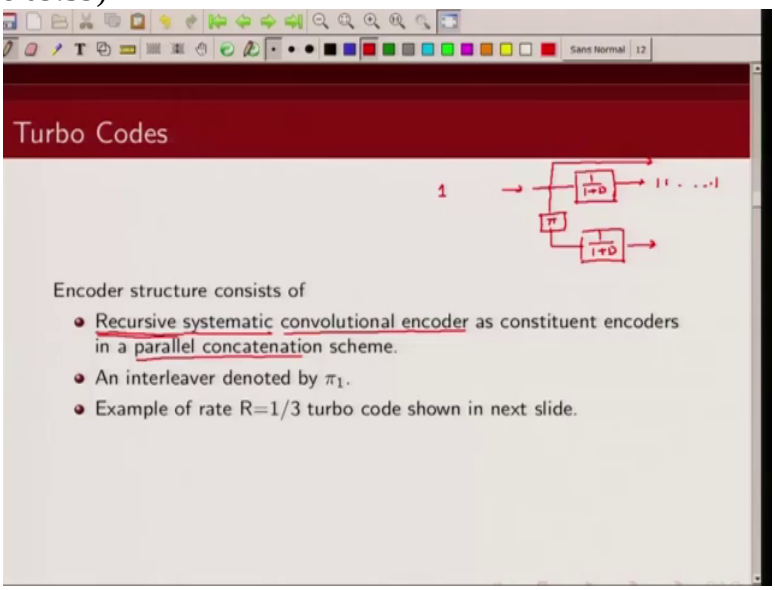

Now usually the input and this is systematic, right, so this comes here, so usually the information

(Refer Slide Time 09:35)



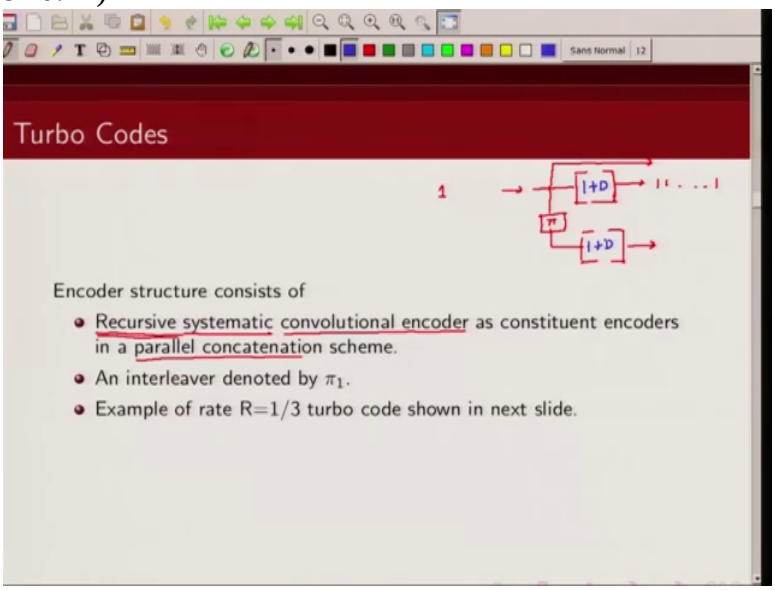

sequence that have low weight can create low output weight sequence also. So let's look at weight 1 sequence. Let's say 1 and all zeroes. So that's just one. Now if this input comes in here to this particular encoder, note this will produce infinite length of

(Refer Slide Time 09:59)



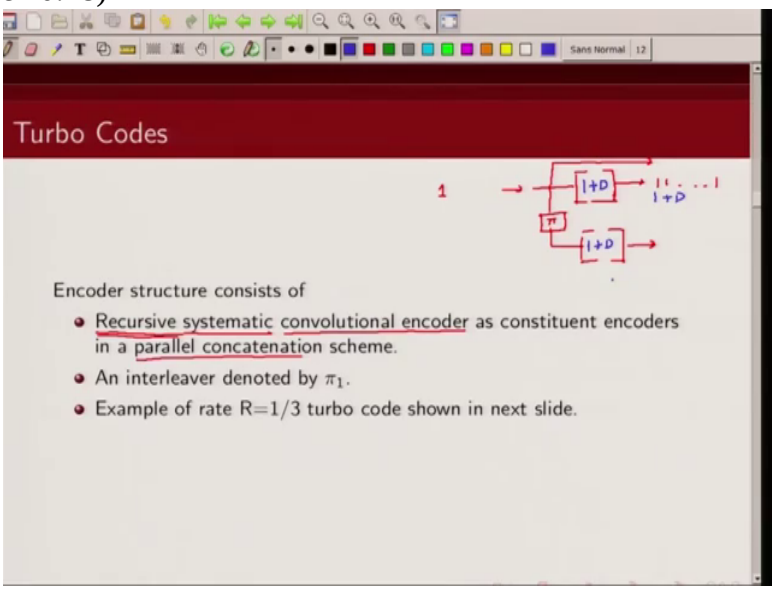

1. And same this 1 which comes out of here after interleaving this will also create a large number of 1s. Whereas if instead of recursive encoder if we would have used a non recursive encoder, let's say if we had used a feed forward encoder, 1, 1 plus d,

(Refer Slide Time 10:22)



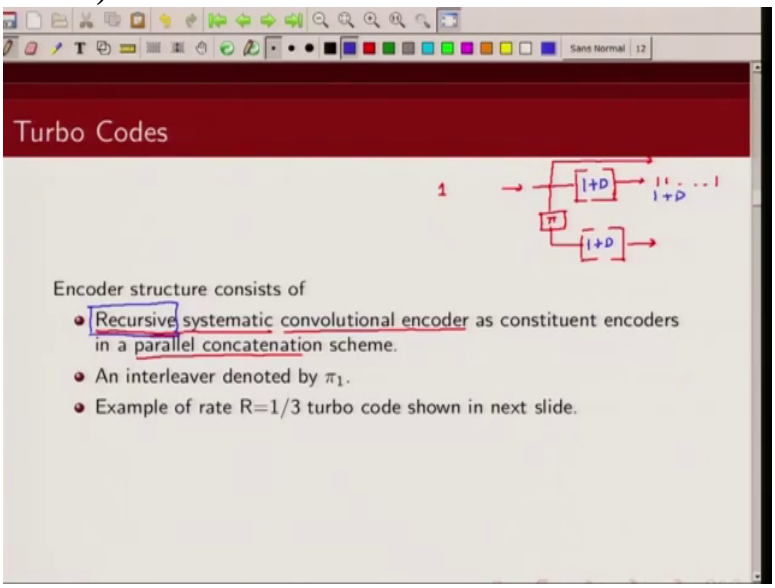

then what would have happened? This would have been just 1 plus d and of course

(Refer Slide Time 10:28)




this would have been 1 plus d some, or some shifted version of that. So you can see that we can get better distance by using feedback convolutional encoder. So that's why this is very important that, and this was a key innovation

(Refer Slide Time 10:49)

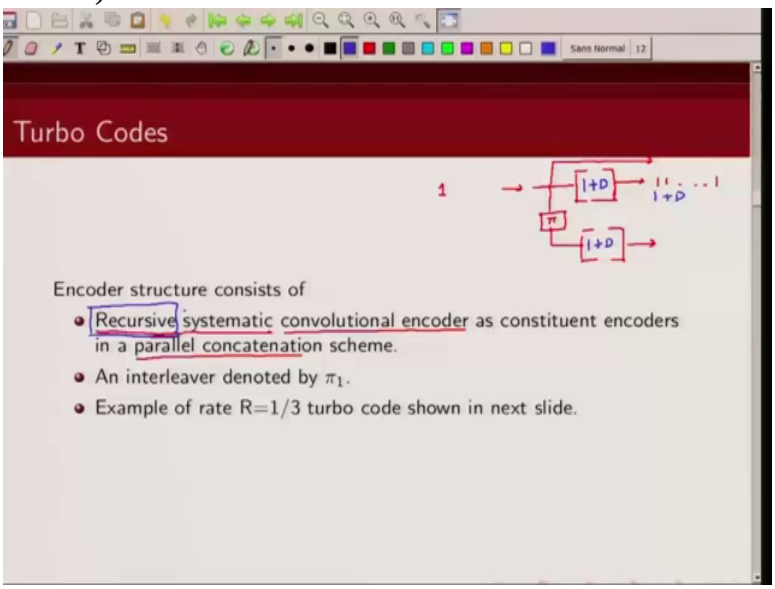



for this code that, when they

(Refer Slide Time 10:52)

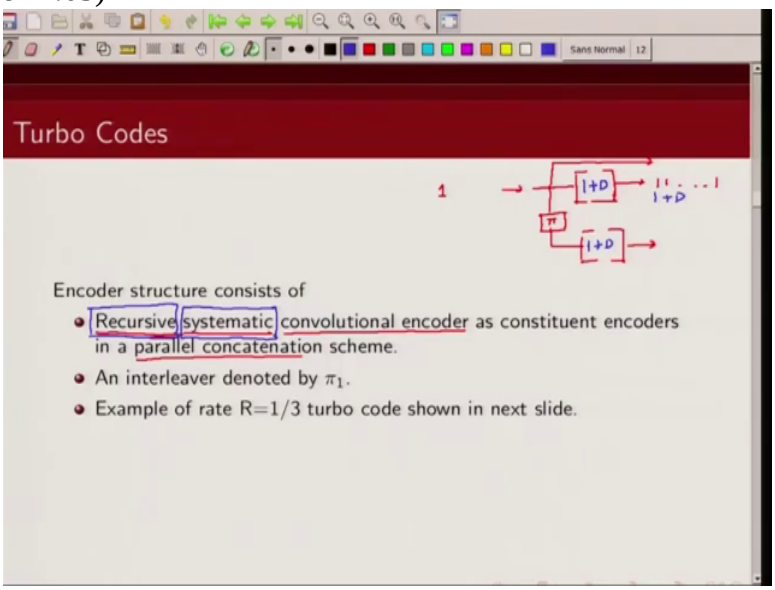
used 2 parallel concatenated codes, the convolutional encoder which was used was recursive convolutional encoder. It was

(Refer Slide Time 11:01)




systematic because systematic codes have better convolutional

(Refer Slide Time 11:05)

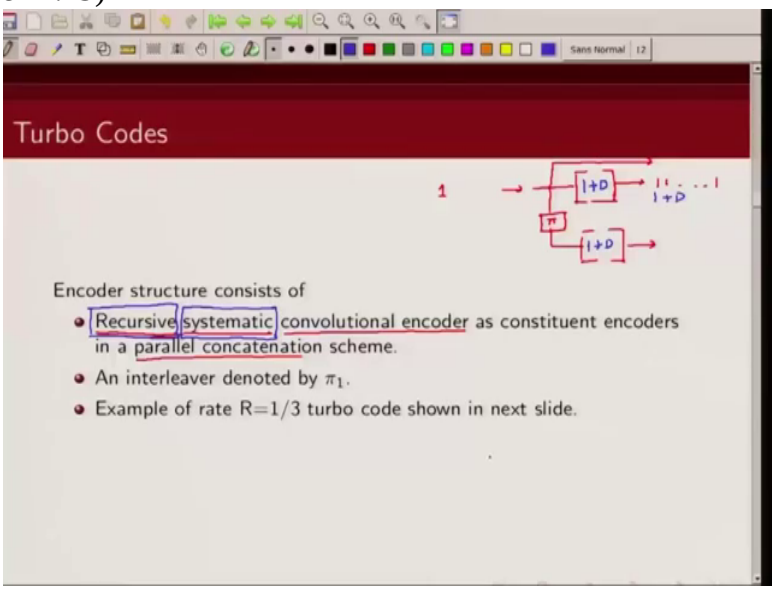



property. So remember these 2 things. So what are the components of the turbo code? So first thing I said it is the

(Refer Slide Time 11:20)

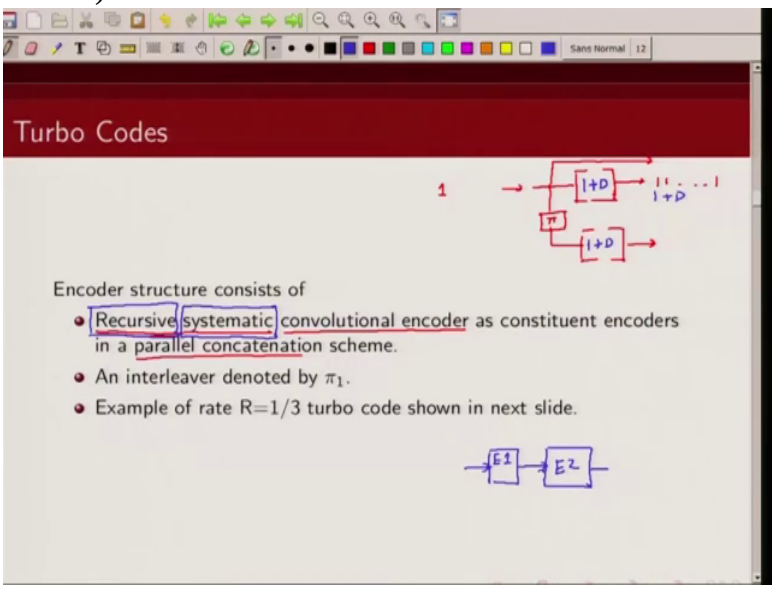
concatenation of two or more encoders in a parallel fashion as opposed to serial concatenation. In serial concatenation

(Refer Slide Time 11:29)



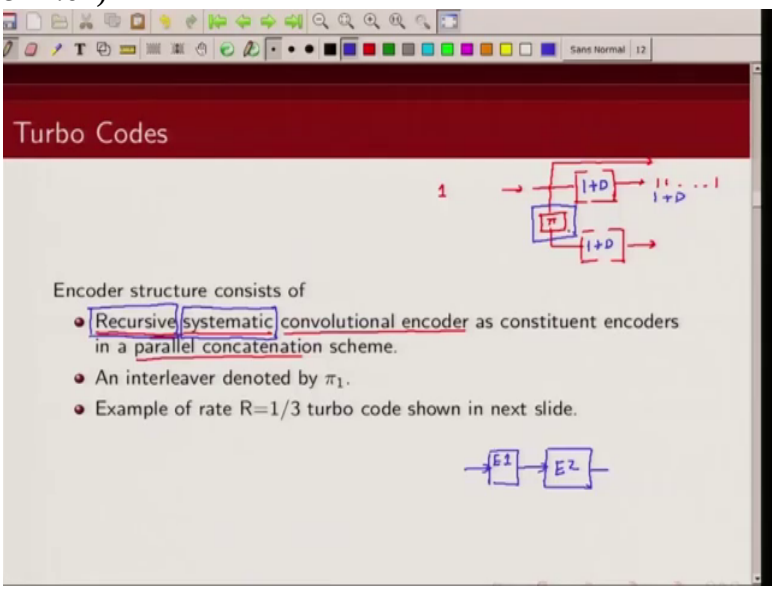

what happens is, so there are bits coming in here, this output of first encoder that is fed as input to the second

(Refer Slide Time 11:48)




encoder. So this is your encoder 1, encoder 2, so the output of encoder 1 is fed as input to the second encoder. But in parallel concatenation you are sending the same information parallely to both the encoders. Now what is the role of interleaver and what is

(Refer Slide Time 12:02)

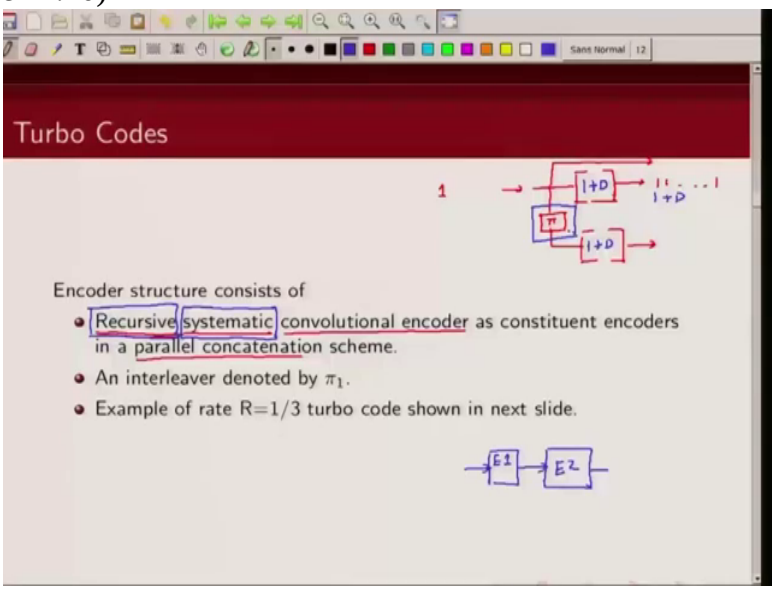



interleaver? As I said interleaver just permutes the bit. So it just reorders the bits. So some bits which were there

(Refer Slide Time 12:12)

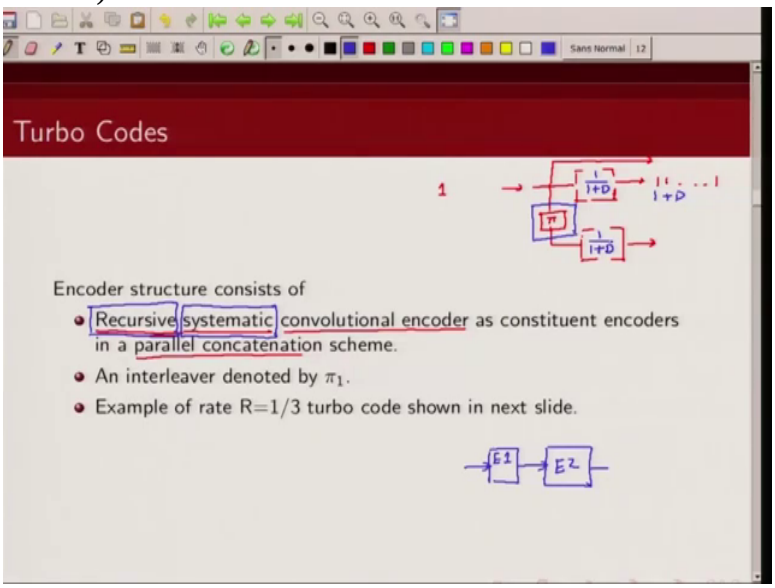
in the, let's say bit location 1, may be it will put in bit location 56 and it just shuffles bits here. Now

(Refer Slide Time 12:20)



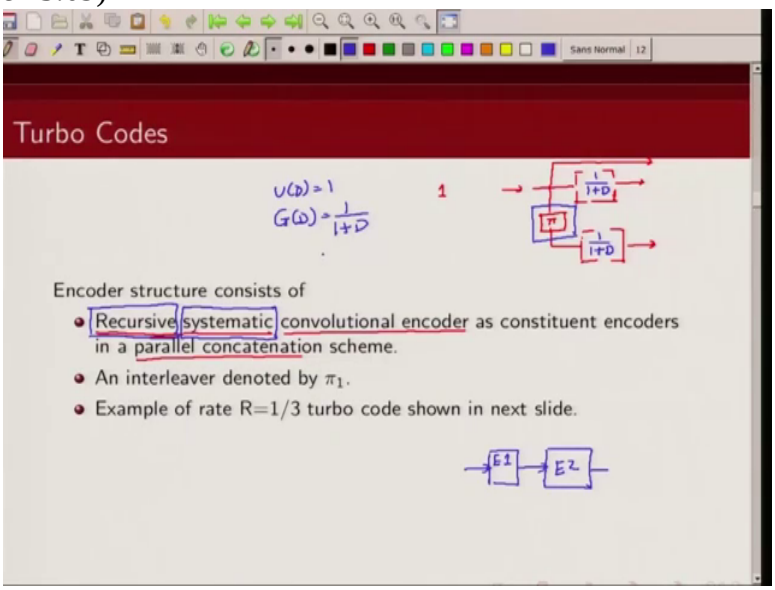

the design of interleaver and the role of interleaver is very very crucial for turbo codes. So let's illustrate that again with a simple example. So what we are considering is this recursive or feedback encoders and we are considering a very simple

(Refer Slide Time 12:43)



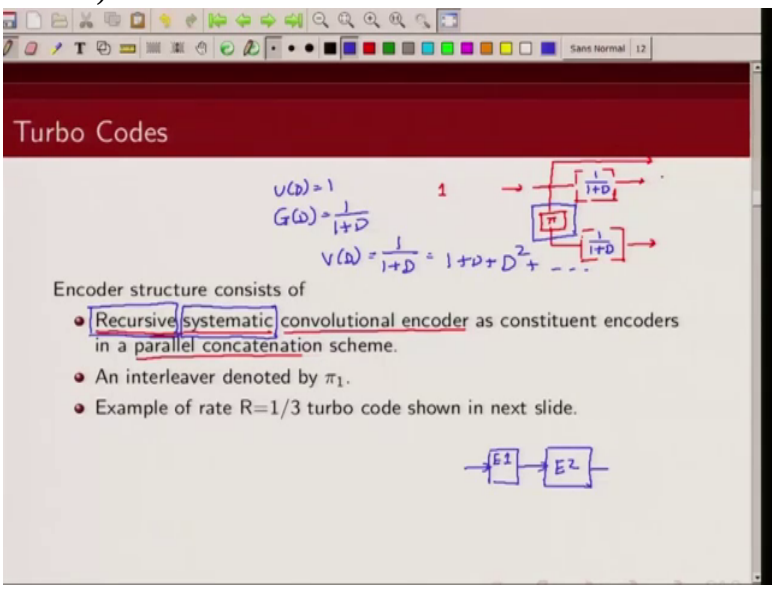

feedback encoders which is memory 1s 2 state convolutional encoder. Now as I said a input 1 sequence cannot terminate this encoder. If my u of d is 1, and g of d is 1 by 1 plus d,

(Refer Slide Time 13:09)



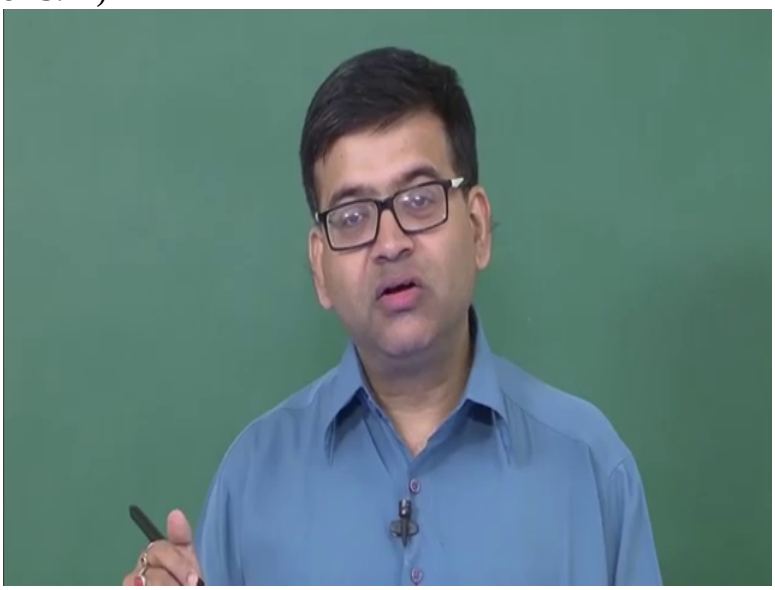

then what is my v of d? It is u times g, which 1, 1 plus d. And this is 1 plus d plus d square. So this I am

(Refer Slide Time 13:20)



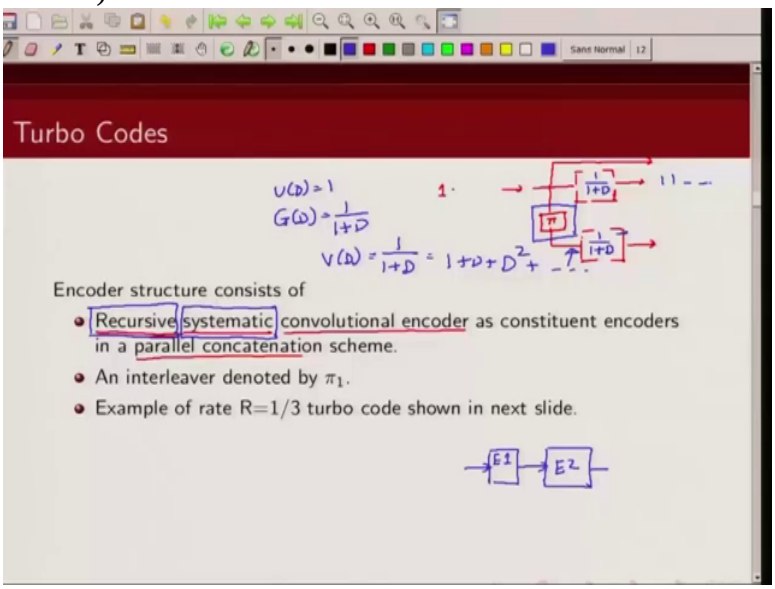

getting a string of 1s, the same thing when you permute it does not change the weight distribution. It just reorders the bit. So again the weight, one sequence that you will get here cannot terminate this encoder. So I will get a large output weight. Now let's look at weight

(Refer Slide Time 13:44)



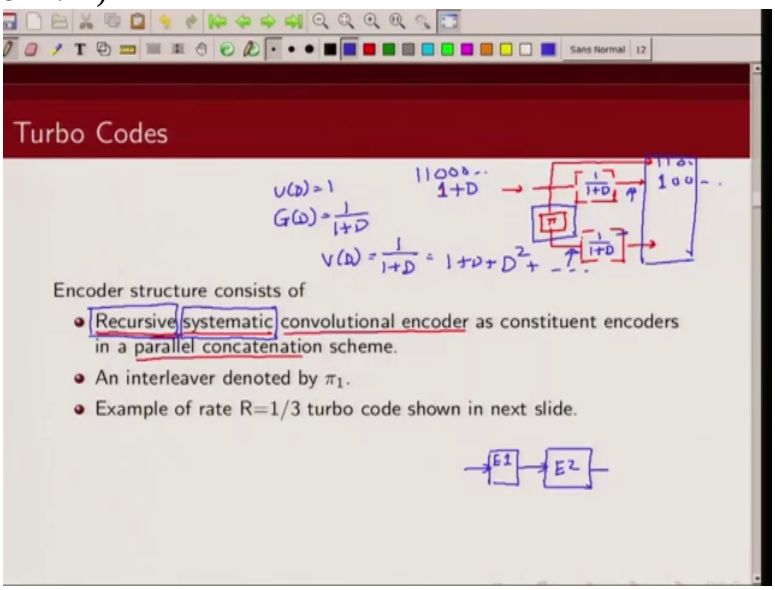
2 sequences. Let's say if I have one

(Refer Slide Time 13:46)




1 and all zero. So let's say one 1 and all zero, that is one plus d. So if my input is 1 plus d, then what happens to the output at the first decoder? Now this sequence can terminate this encoder and what I will get here is I will just get a 1. I will just get a 1 because my u d is 1 plus d and g d is 1 by 1 plus d. So what I will get here is 1.So I am getting parity bit out of this parity check bit I am just getting 1 and all zeros here. Now to have a code with overall weight large, remember my input is only 2 weight, it, my input is 1 1 and all zeroes so here also I will get 1 1 and all zeroes.

(Refer Slide Time 14:42)

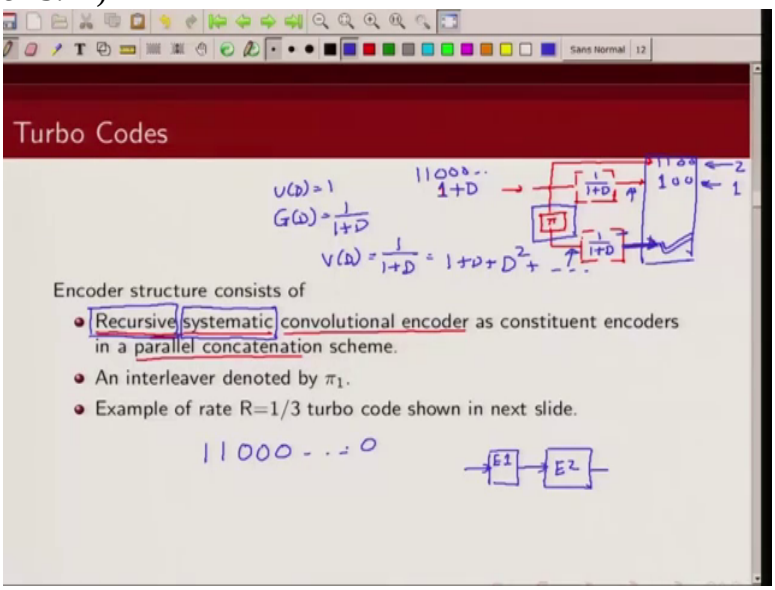



So here I am getting only weight 2. Here I am getting only weight 1. Now to have an overall weight large what would you like? You would

(Refer Slide Time 14:52)

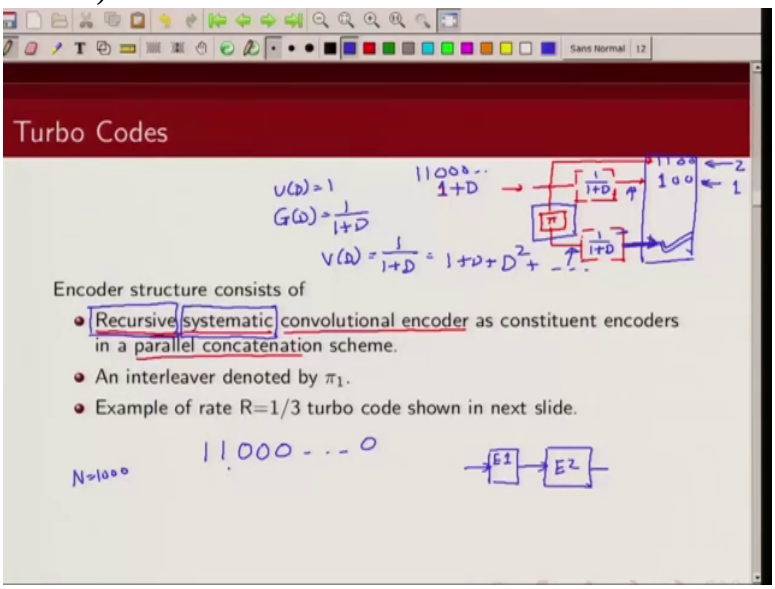
like to have a large number of 1s coming out from this parity check bit. Now how can you do that? Now remember, what is interleaver? Interleaver is just a permuter, it is just shuffling bits. So if you had bits which were like this, let's say 1, 1 and all zeroes, and

(Refer Slide Time 15:17)



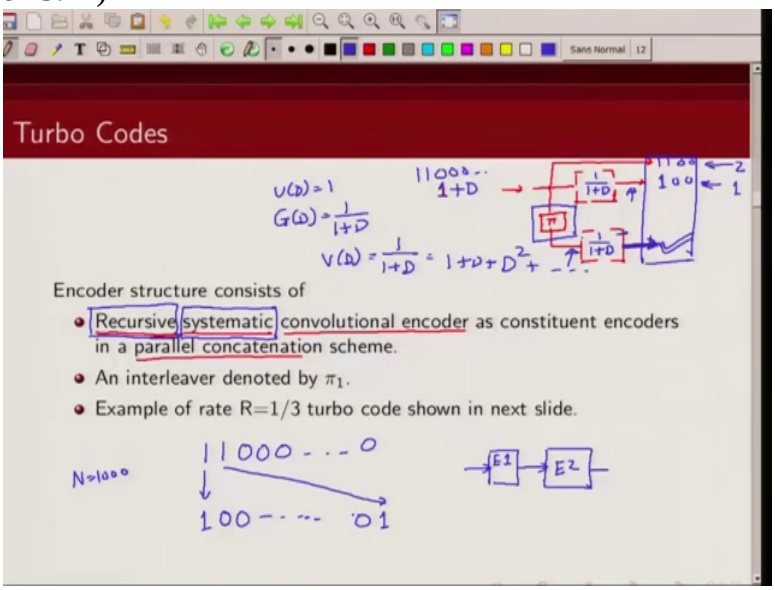

typically the block sizes are very large, I mean 1000, 10000 let's say 1000 or something like that.

(Refer Slide Time 15:24)



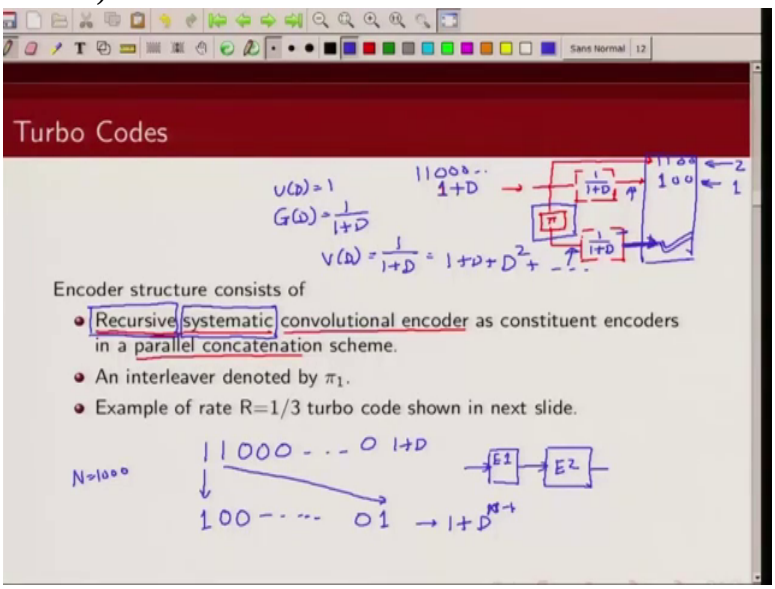

And let's say your interleaver what it did, it did was it kept this one here but moved this one to the last location and in between you had all zeroes.

(Refer Slide Time 15:42)




So this was your 1 plus b and this becomes 1 plus d raised to power let's say n minus 1 because

(Refer Slide Time 15:54)

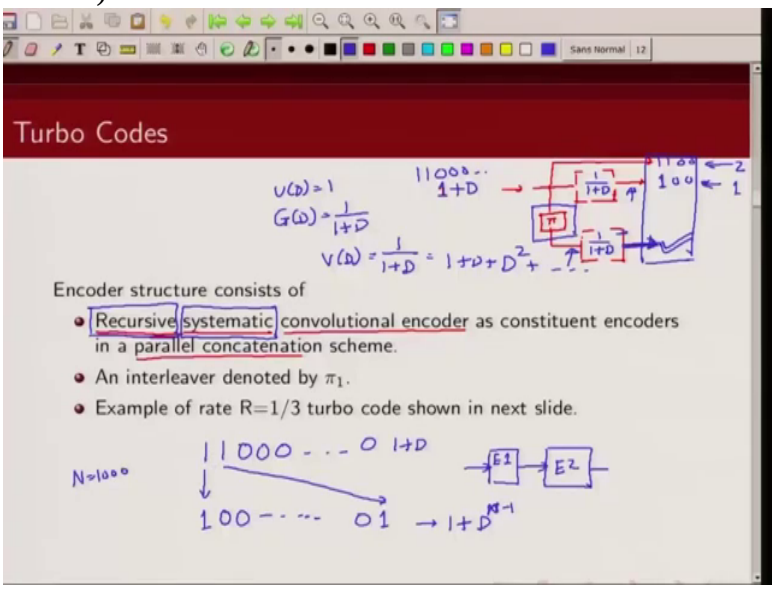



the way you designed this

(Refer Slide Time 15:56)

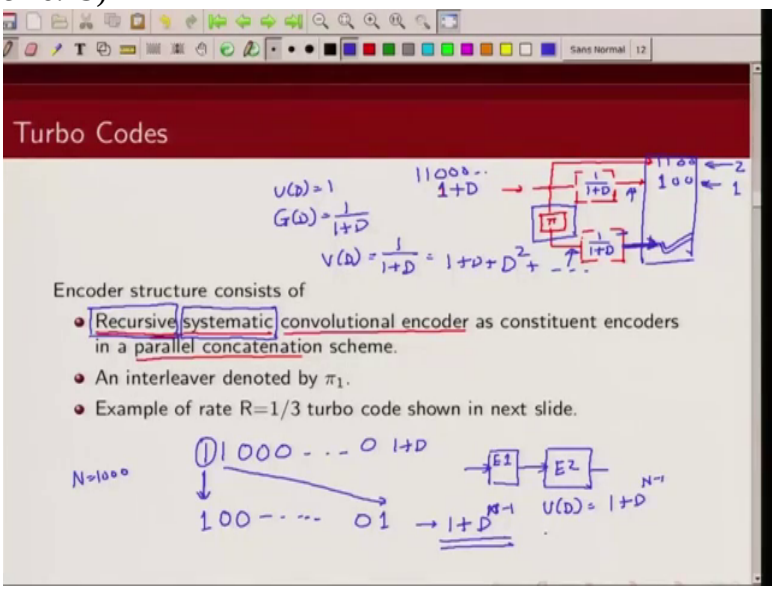
interleaver

(Refer Slide Time 15:58)



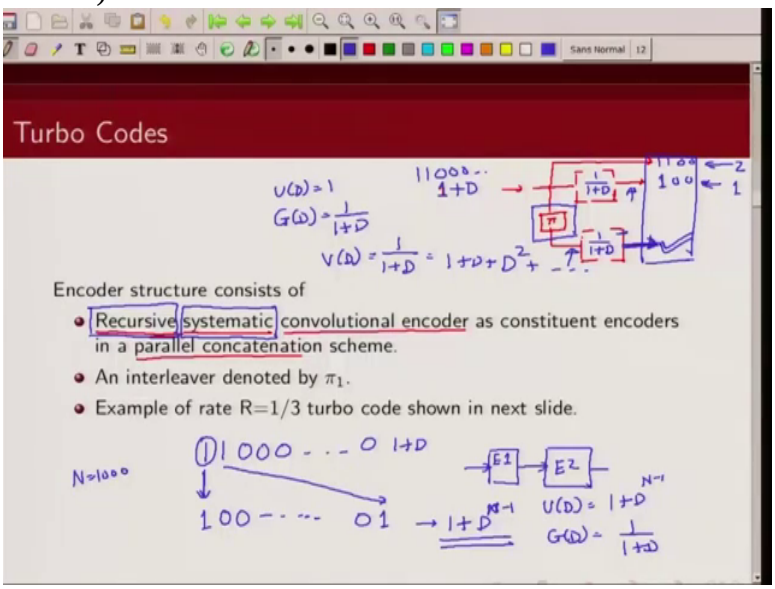

it rearranged the bits in such a way that, let's say this bit was put here, this bit was put here and other zeroes were put in between. Now if you feed in this sequence to this encoder, so your u D is 1 plus d raised to power n minus 1.

(Refer Slide Time 16:19)




g d is 1 plus d. What will you

(Refer Slide Time 16:24)

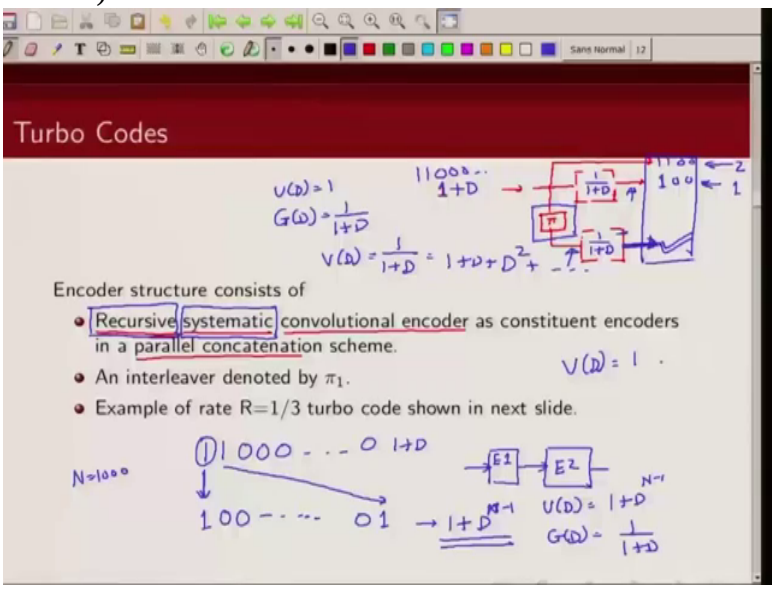



get? You will get a, output sequence which will have large number of 1s. So what I did was I just designed my interleaver in such a way that 2 adjacent 1's

(Refer Slide Time 16:39)

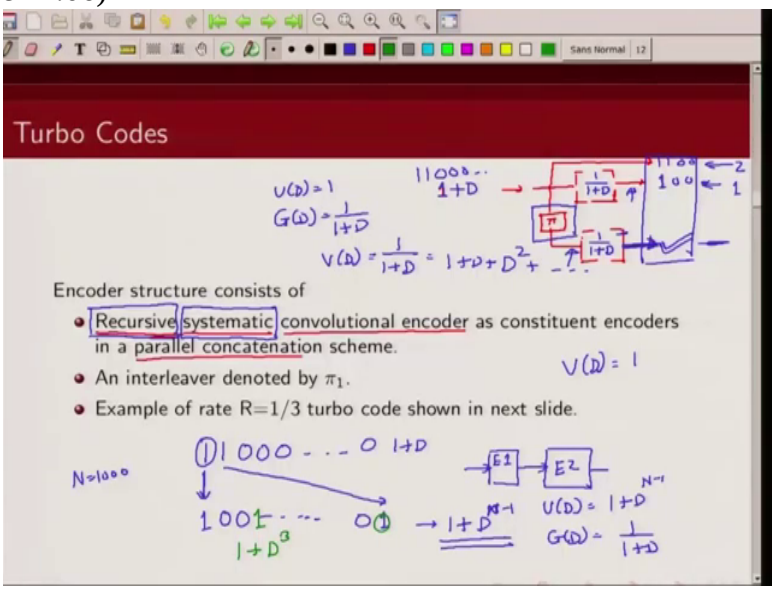
were spread out and as a result you

(Refer Slide Time 16:43)



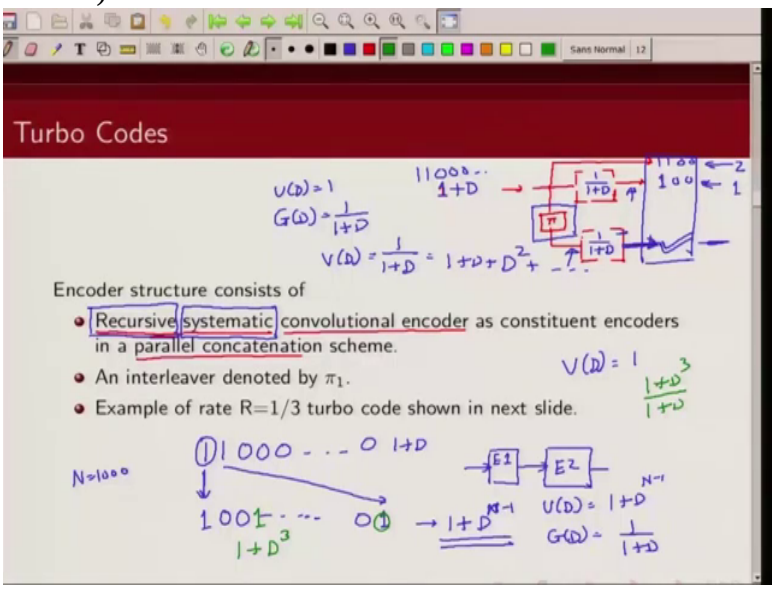

noticed that I am getting a large number of 1s coming out from this parity bit. Contrast it with the situation where let's say instead of shuffling this bit here if I would have kept this bit let's say here. And this would have been here and this would have been zero and this would have been 1 plus d cube.

(Refer Slide Time 17:06)



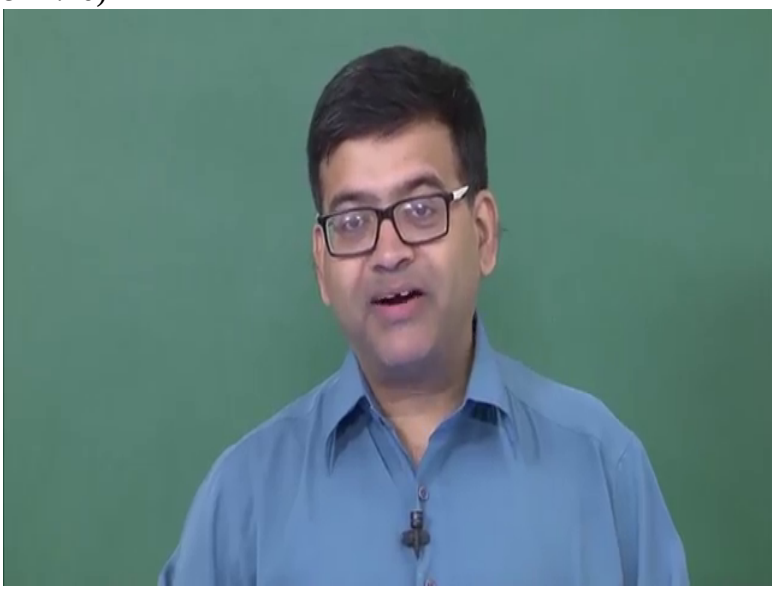

Then what would have get? I would have got 1 plus d cube by 1 plus d. So this would be

(Refer Slide Time 17:12)



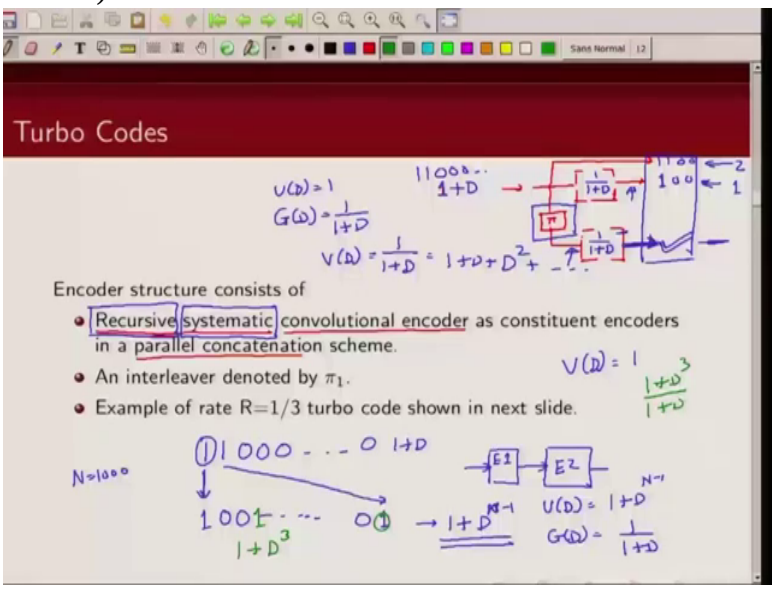

1 plus d plus d square. So I would have only got 3 1's here but now I am getting a large number of 1s. So I hope I am able to convey the role of interleaver. The design of interleaver is very very crucial. You should design the interleave in such a way such that the adjacent ones are separated in such a way such that the adjacent ones are separated out far apart

(Refer Slide Time 17:40)



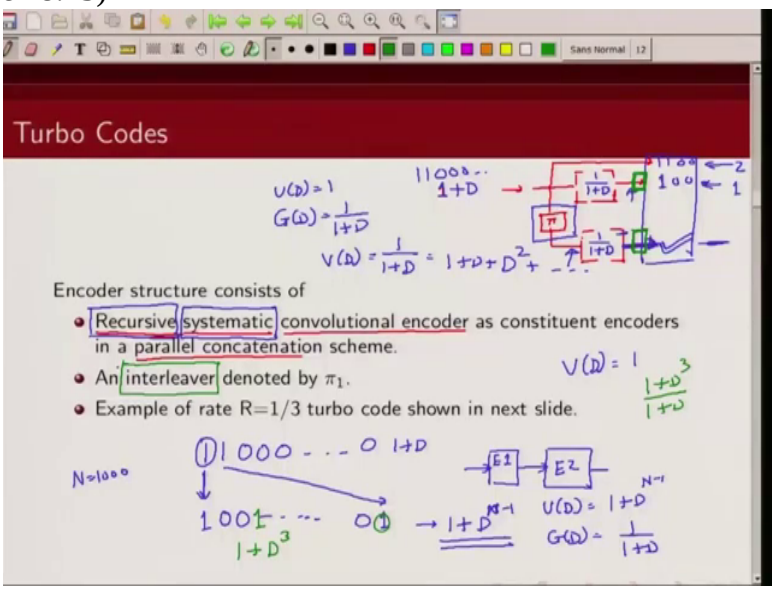
so that after the interleaving, when the same information sequence is passing through the encoder, it generates a large parity, a parity sequence with large weight. Ideally

(Refer Slide Time 17:55)



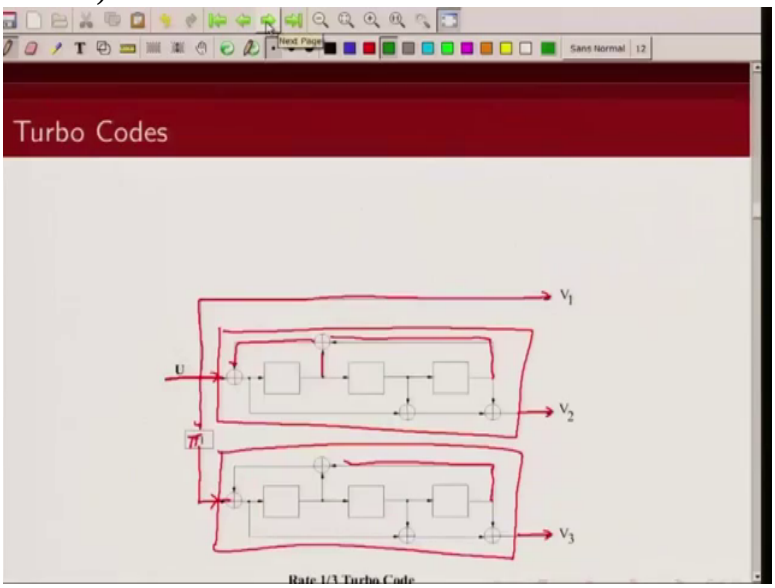

what you would like is if the parity sequence is coming out here at low weight, then this should generate large weight parity sequence. And if this guy is generating parity sequence for low weight then this sequence, this should generate a parity sequence with large weight. So again I cannot emphasize enough the role of interleaver

(Refer Slide Time 18:23)



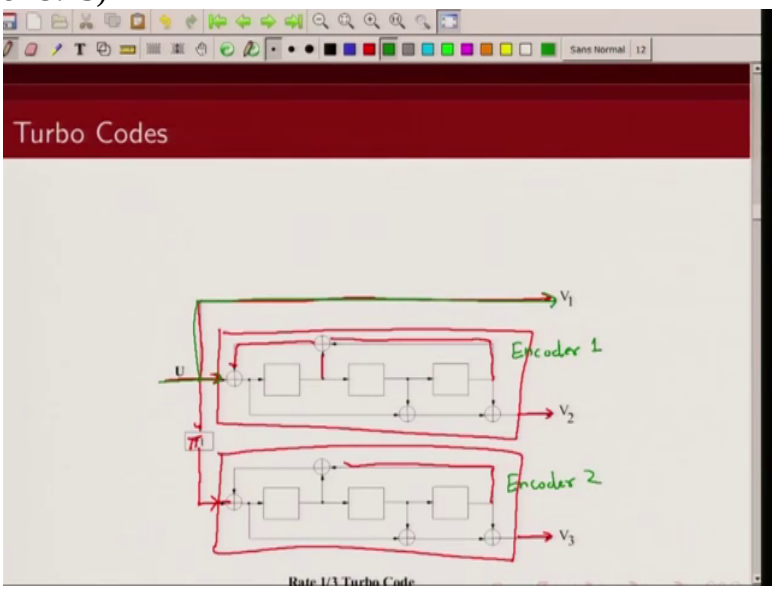

because this is a very, very crucial component of turbo encoder. And it also helps in some sense, randomizing the code length; make the code look random like, if you want to use that word. So let's look at the

(Refer Slide Time 18:40)



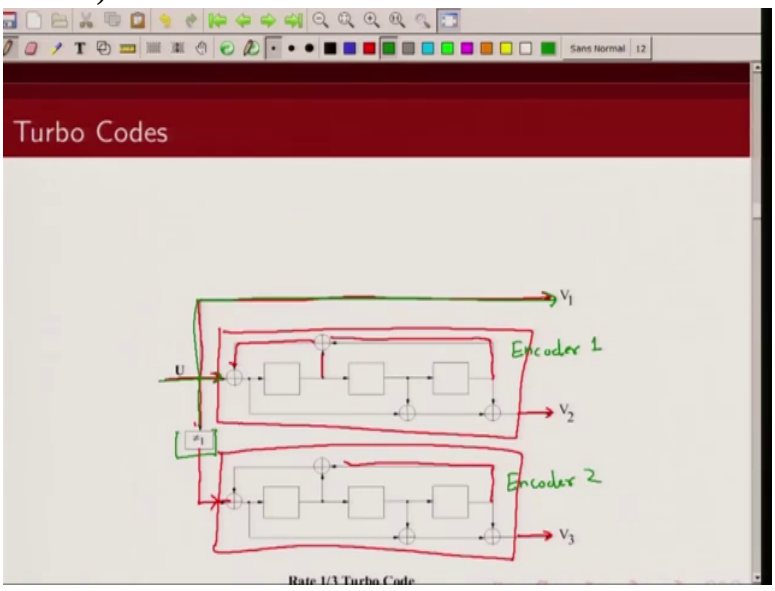

structure of turbo code again. So you note here I have 2 encoders. This is my encoder 1 and this is encoder 2. Both of them are recursive or feedback encoders, you can see that. Each one of them has 8 states in this particular example. Now information sequence is coming here. It's the systematic code so the same information is coming out directly

(Refer Slide Time 19:13)



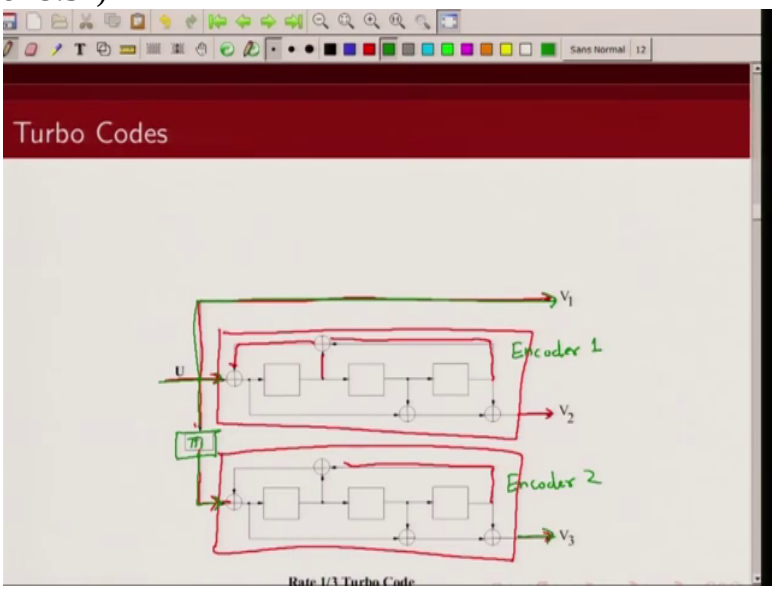

as one of the output. And then the input to the second encoder, this is interleaved. Now interleaving is

(Refer Slide Time 19:24)



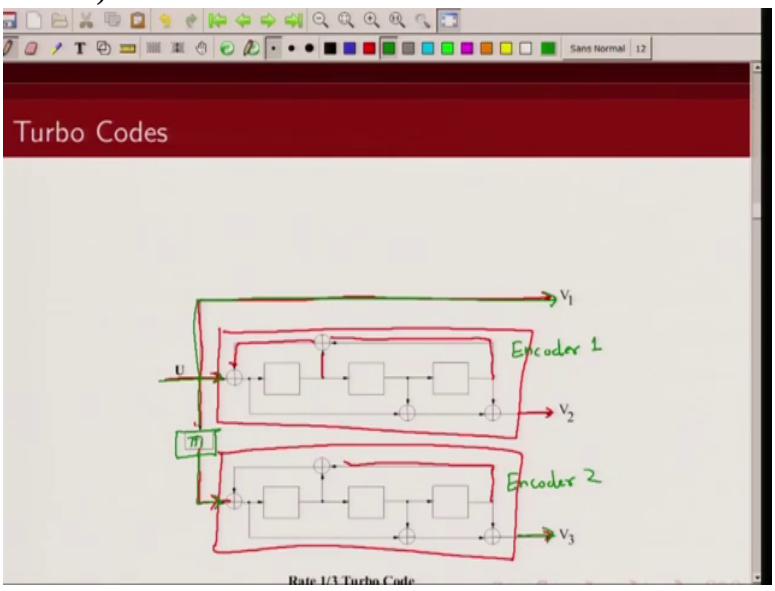

nothing but as I said, the reordering of the bits. Reordered bits are coming to the second encoder which will then generate a set of parity bits.

(Refer Slide Time 19:34)



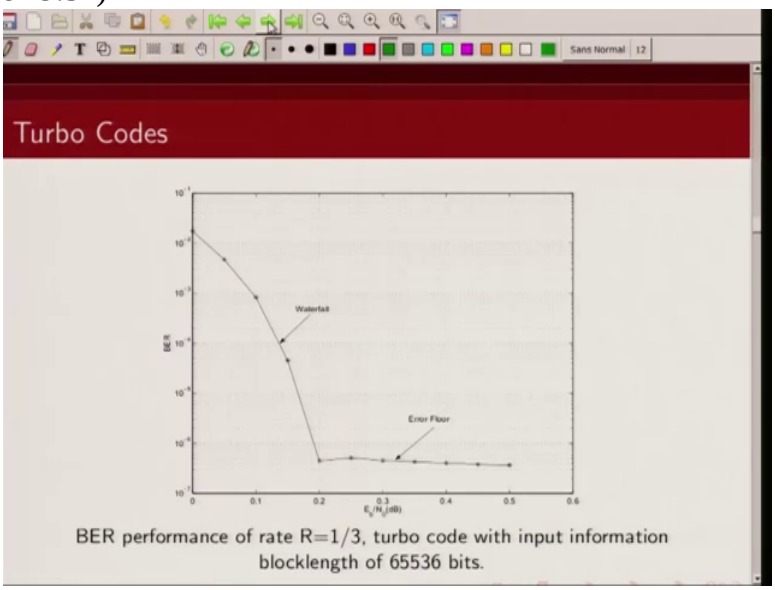

So this is a structure of a parallel concatenated code. In this particular example we are using convolutional

(Refer Slide Time 19:47)



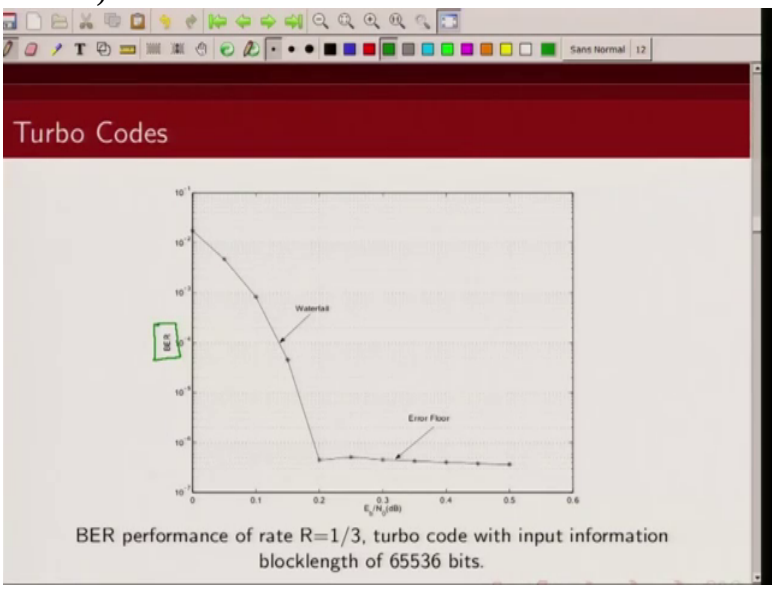

encoders as constituent encoders.

(Refer Slide Time 19:51)



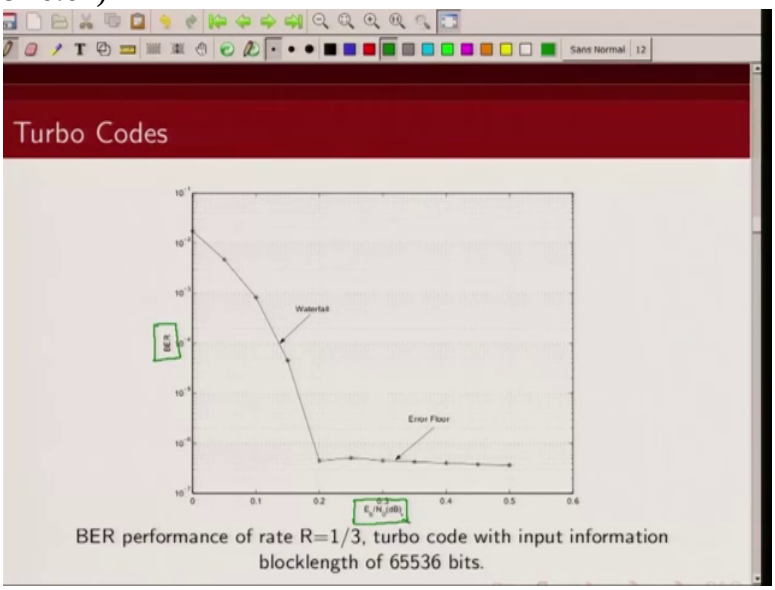

Now in this graph I have plotted bit error rate performance

(Refer Slide Time 19:59)



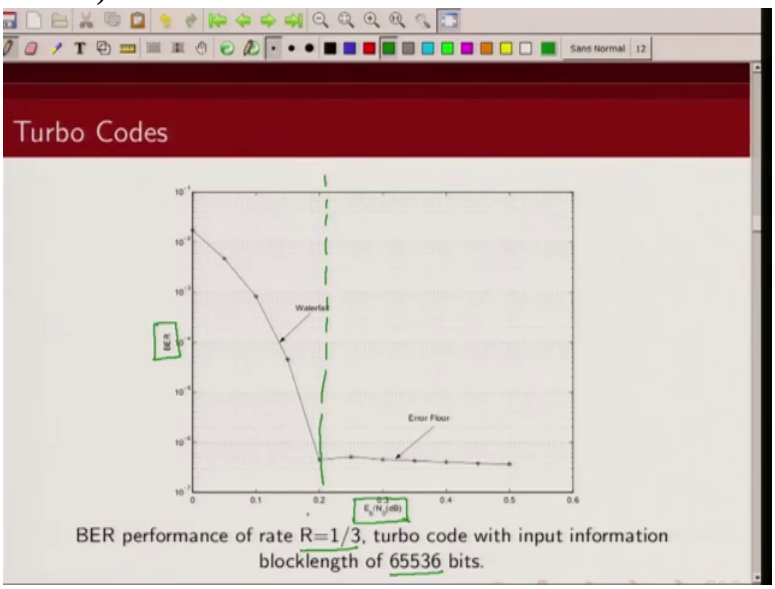

versus signal to noise

(Refer Slide Time 20:02)




ratio for a rate one third turbo code and for a block size more than 65000. Now note that typical performance. So there is a region where

(Refer Slide Time 20:20)

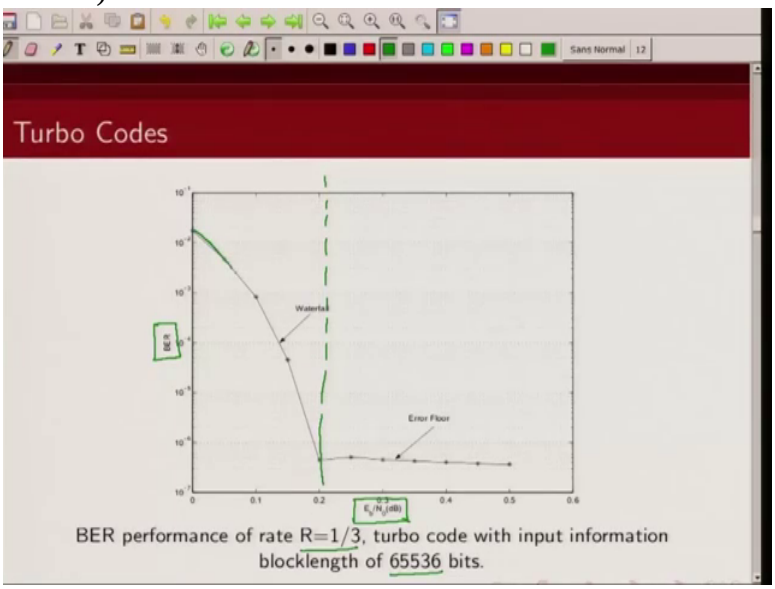



there is a steep fall in

(Refer Slide Time 20:23)

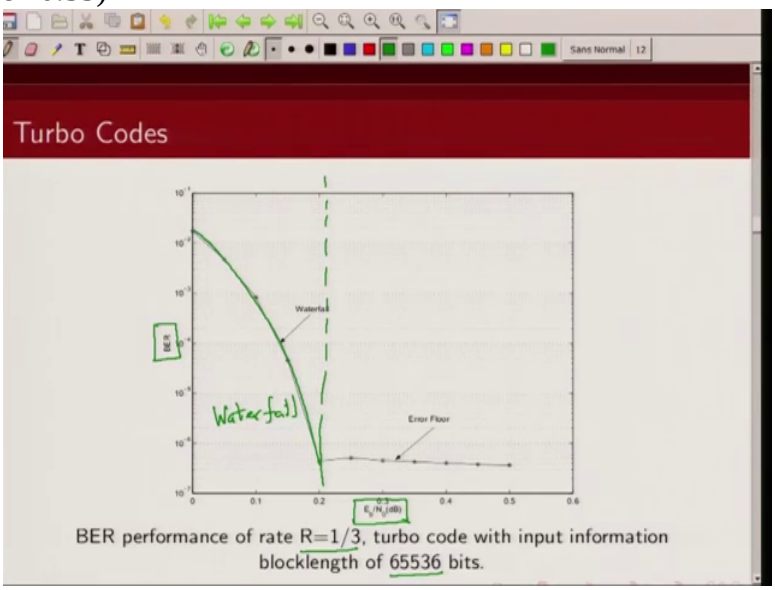
bit error rate performance. And this region

(Refer Slide Time 20:26)



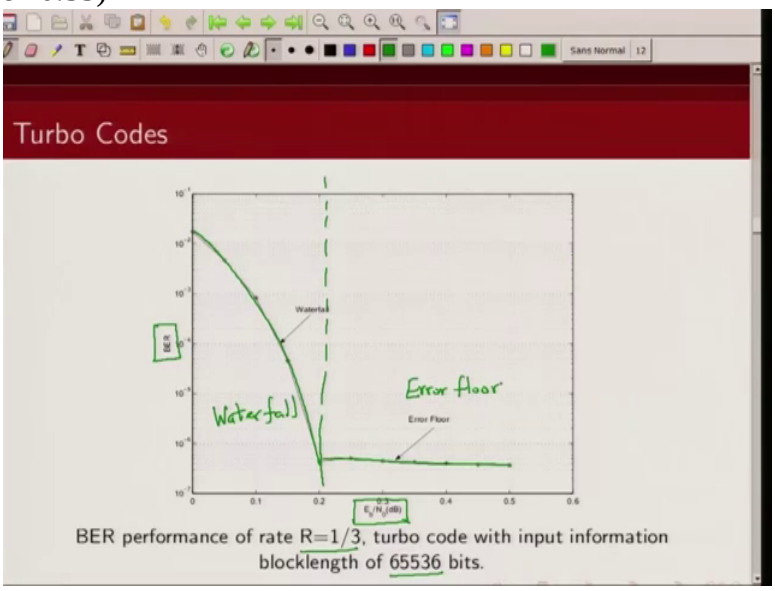

we call it waterfall region, so this is called waterfall region. So this is like a typical waterfall

(Refer Slide Time 20:35)




that performance is coming like this. And then there is a region where there is hardly any improvement in bit error rate in spite of increasing the signal to noise ratio. And this is known as error flow region.

(Refer Slide Time 20:55)

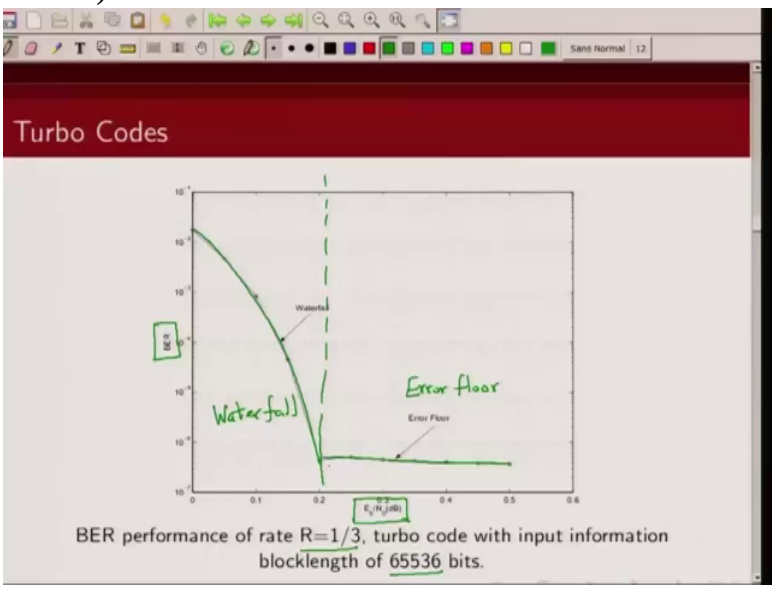



Now performance in the waterfall region is governed by the

(Refer Slide Time 21:01)

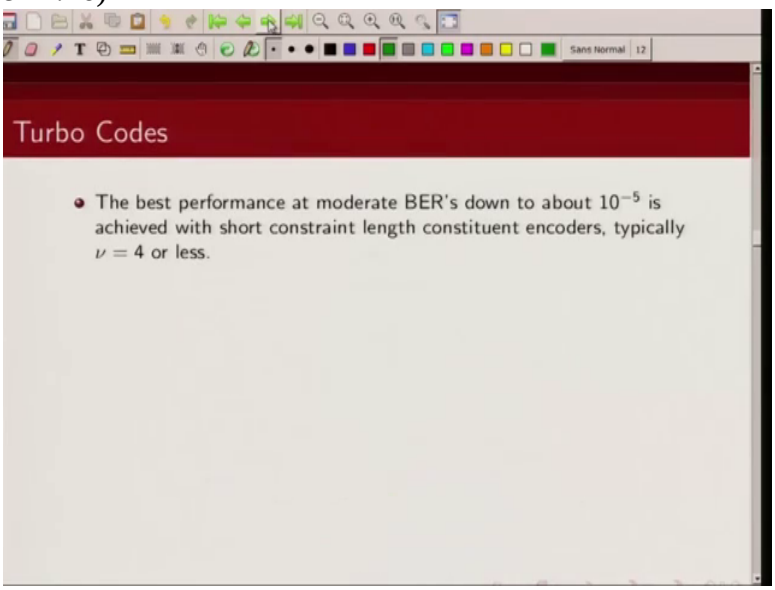
convergence behavior of the constituent encoders under iterative decoding algorithm and we will discuss this in subsequent lectures. And

(Refer Slide Time 21:14)




performance in the error flow region is governed by the distance spectrum of the turbo codes.

(Refer Slide Time 21:26)

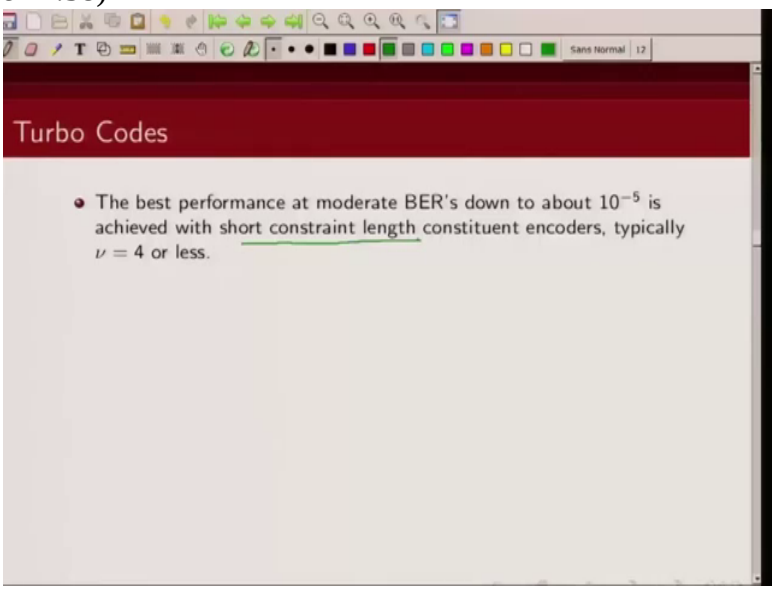



So typically the constituent encoders, the convolution constituent encoders that gives good performance are the ones

(Refer Slide Time 21:35)

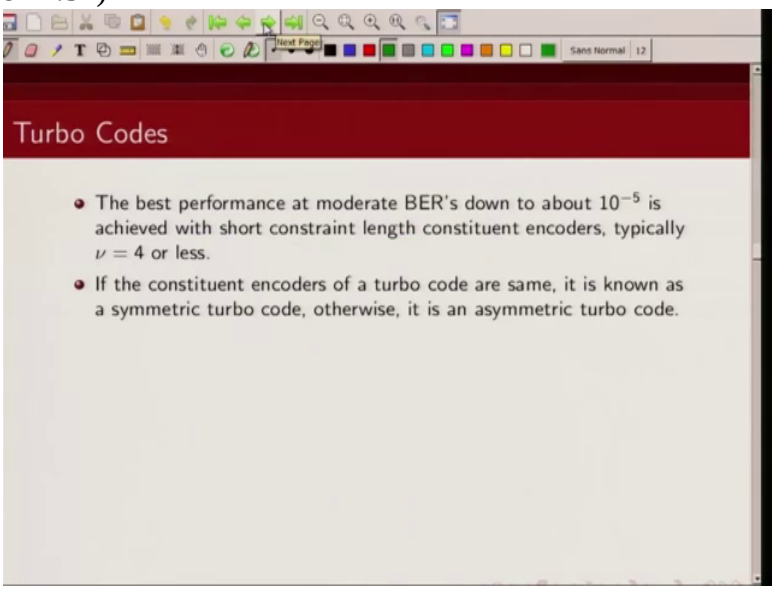
which have short constraint length

(Refer Slide Time 21:38)



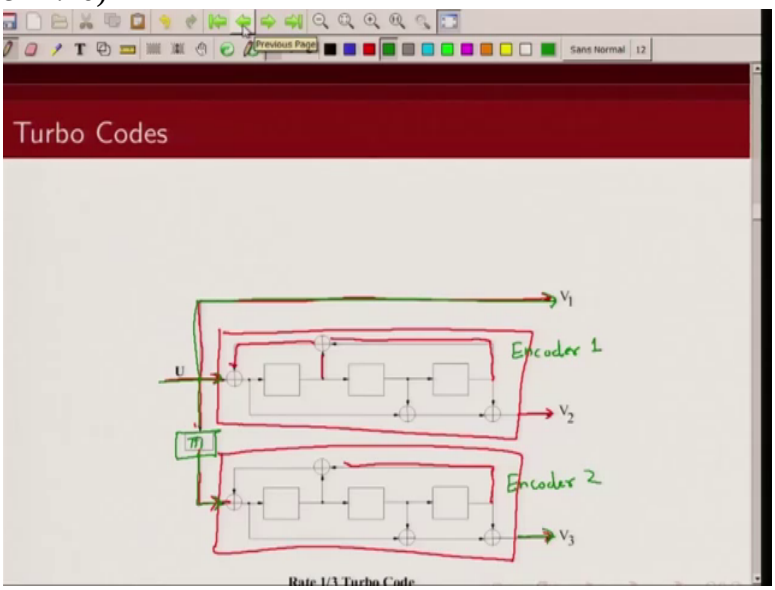

typically 3, 4 like memory 3, memory 4. These are the ones that give good performance. If we use a more stronger code with memory 5, 6 then typically the performance in the waterfall region is not good.

(Refer Slide Time 21:57)



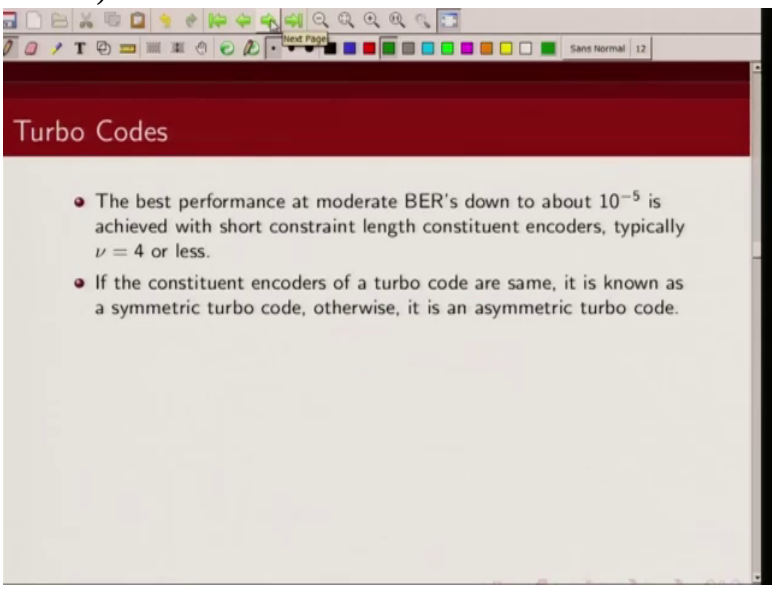

Now what I have shown you is a parallel concatenation of 2 encoders. Now these 2 encoders can be the same encoders like what I have shown here. If you

(Refer Slide Time 22:10)



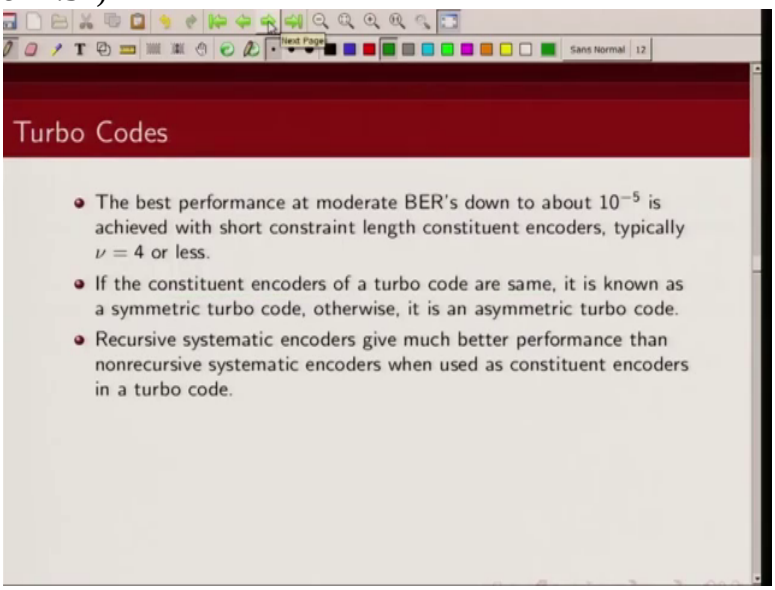

see here, both of encoder 1 and encoder 2 are same. Now they could be same or they could be different. So if

(Refer Slide Time 22:19)




the constituent encoders are same, we call it a systematic symmetric turbo code and if these constituent encoders are different, we call it asymmetric turbo code.

(Refer Slide Time 22:37)

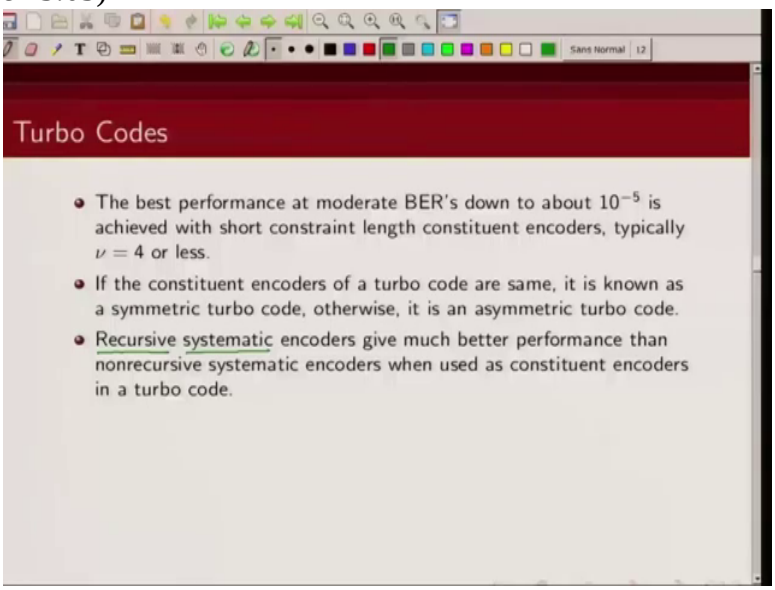



I already mentioned this point that choice of recursive and systematic encoders is very very crucial because for a recursive encoder a weight 1 pattern cannot terminate a encoder where as for a

(Refer Slide Time 22:57)

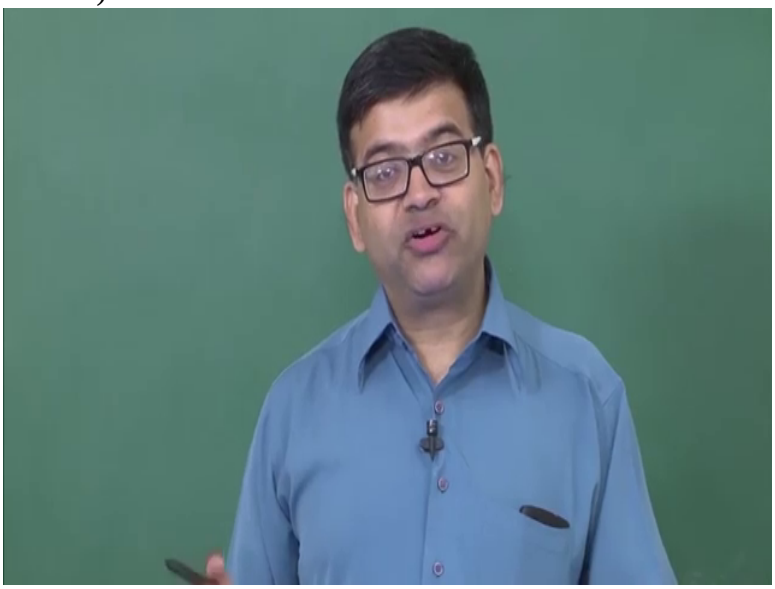
feed forward encoder a weight 1 pattern can terminate the encoder.

(Refer Slide Time 23:03)



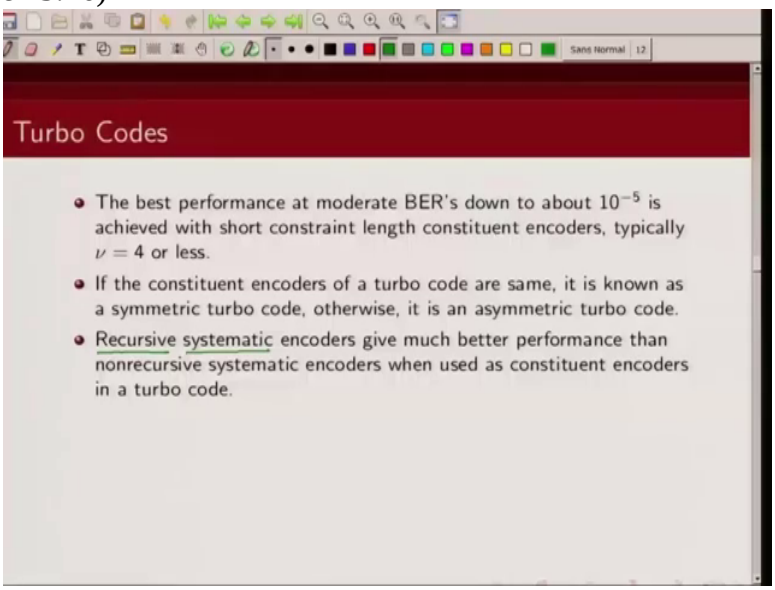

That's why we should use a recursive encoder and systematic encoders

(Refer Slide Time 23:09)



help in initial performance in waterfall region and that's why they are preferred over

(Refer Slide Time 23:16)

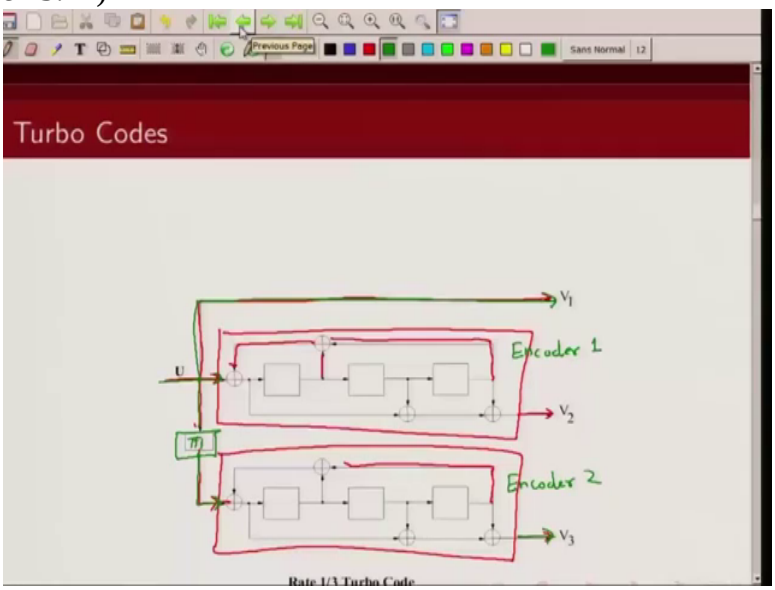



non systematic encoders. Especially if you are interested in

(Refer Slide Time 23:22)

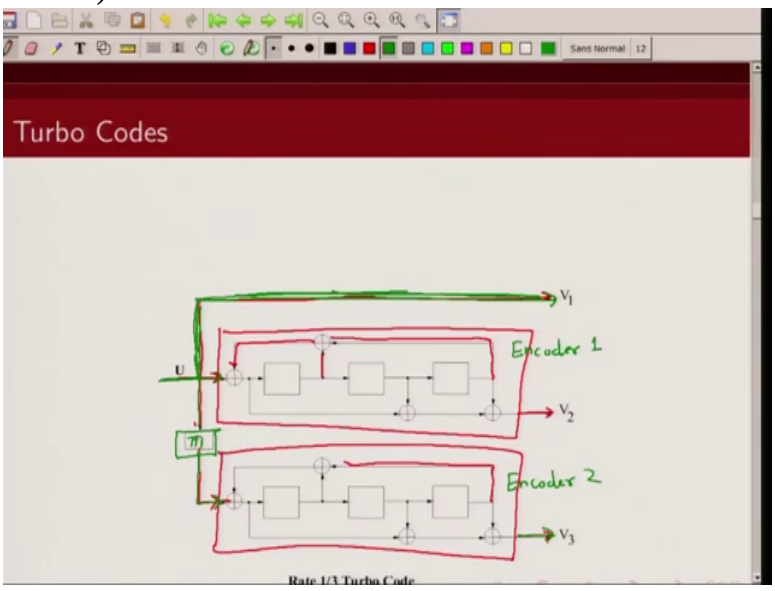
performance in the waterfall region.

(Refer Slide Time 23:27)



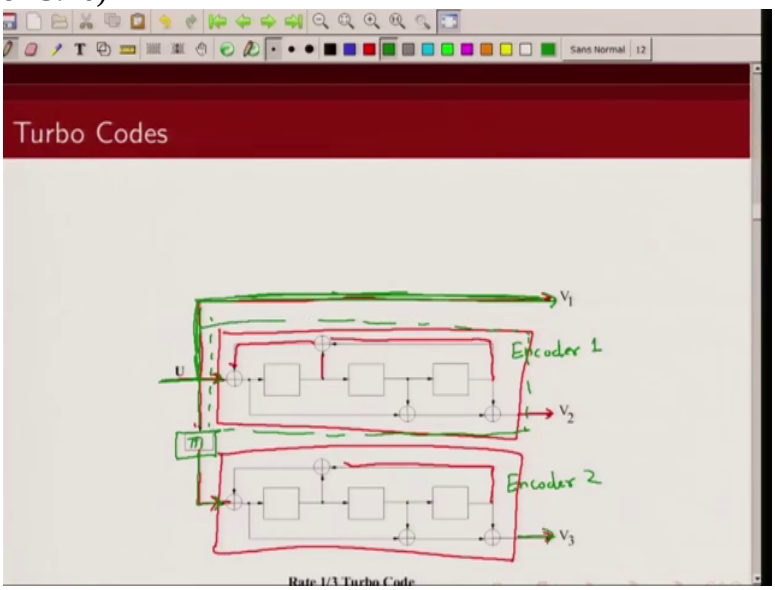

So what we are doing here is, this is information bit, right? This is the information bit that is coming out in the output.

(Refer Slide Time 23:39)



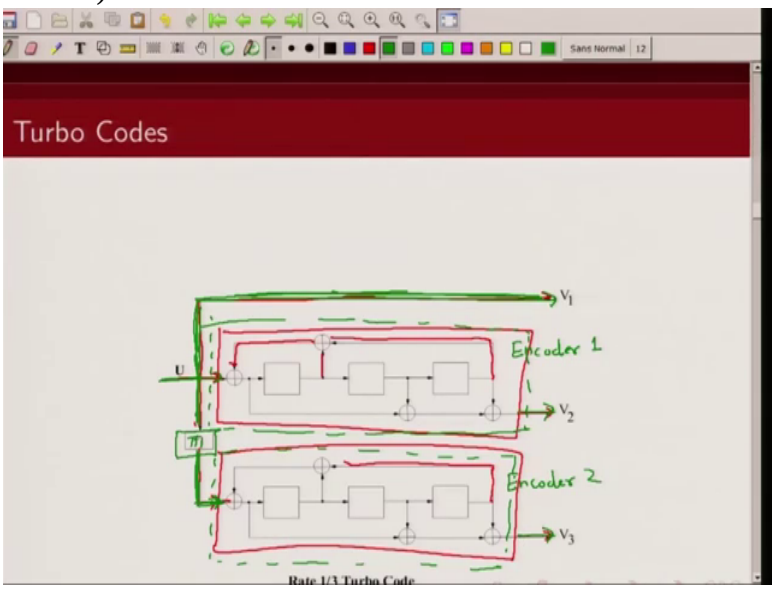

And then you have this encoder

(Refer Slide Time 23:46)




which is generating this parity bit, right? Now this interleaved bit, interleaved information sequence is coming here and what this encoder 2 generates, this encoder 2 generates

(Refer Slide Time 24:04)

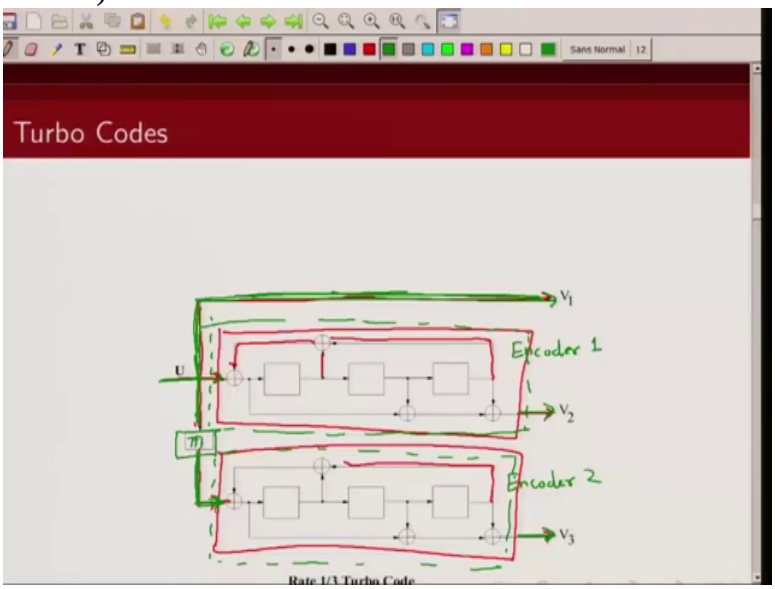



this parity bit. Please note that we are not sending the information bit corresponding to second encoder.

(Refer Slide Time 24:13)

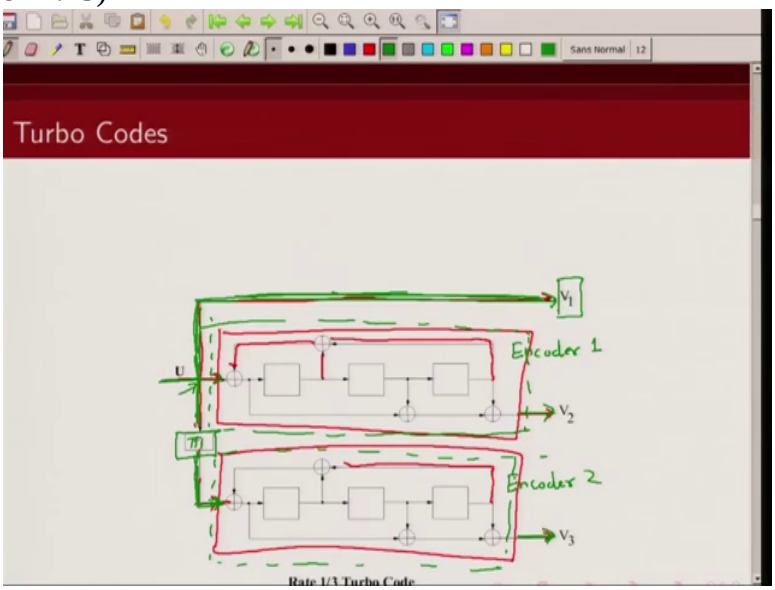
Why? Because the

(Refer Slide Time 24:17)



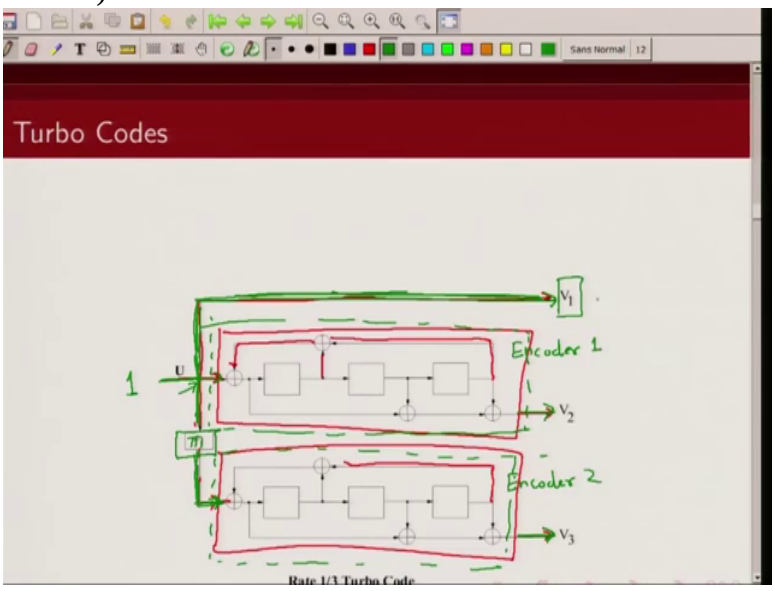

information bits corresponds to second encoder is nothing but interleaved version of the information sequence that is being fed to encoder 1 and that's why we are not sending, we are not sending the information bits here. We are not sending the information bits of the second encoder. We are not sending it across the channel that is because from this information bit,

(Refer Slide Time 24:43)



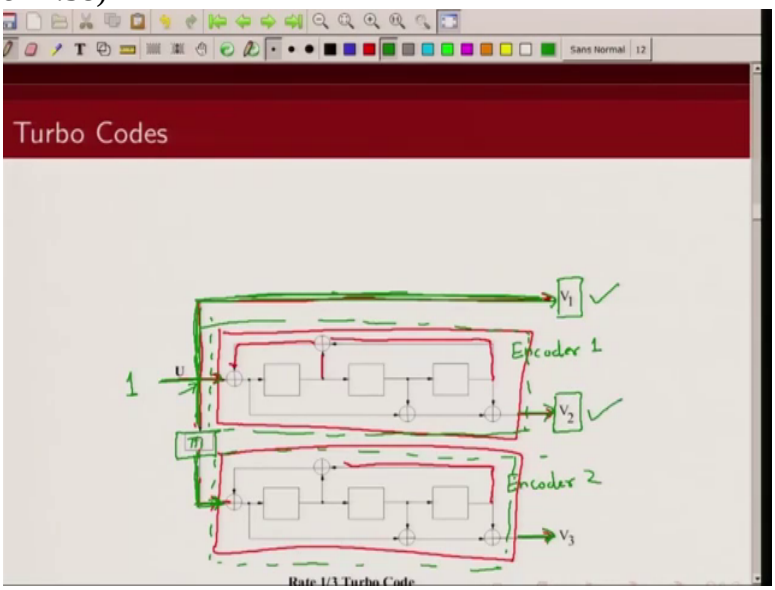

if we just interleave we can get information bits for the second encoder. So you can see this is the rate one third code because there is just one input

(Refer Slide Time 24:54)



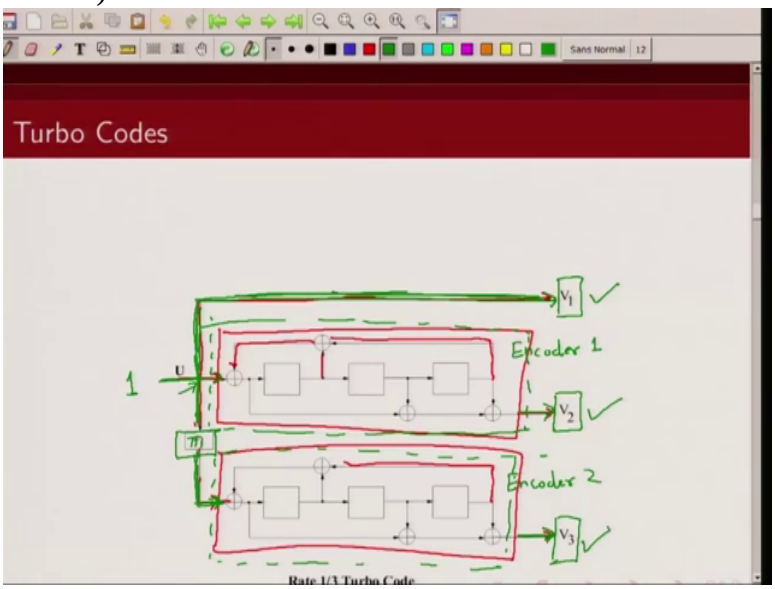

and there are 3 outputs. This is one output,

(Refer Slide Time 24:58)



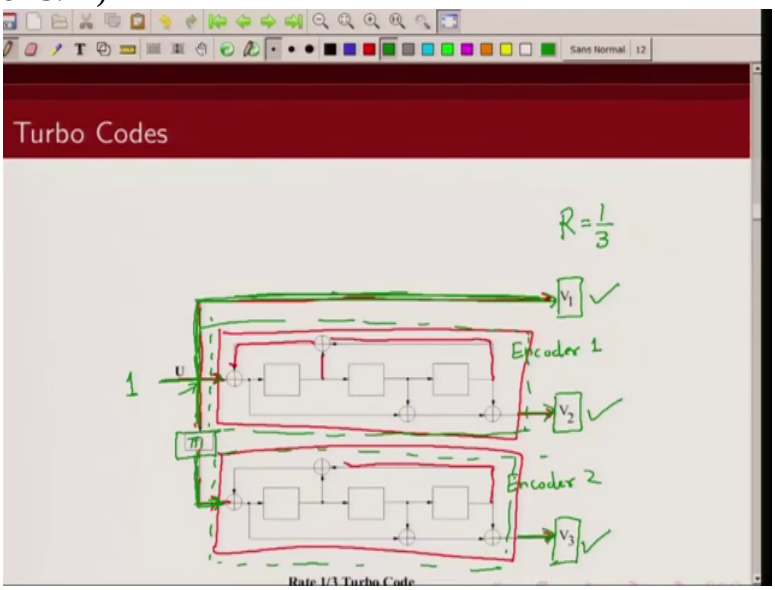

this is one output and this is one

(Refer Slide Time 25:01)



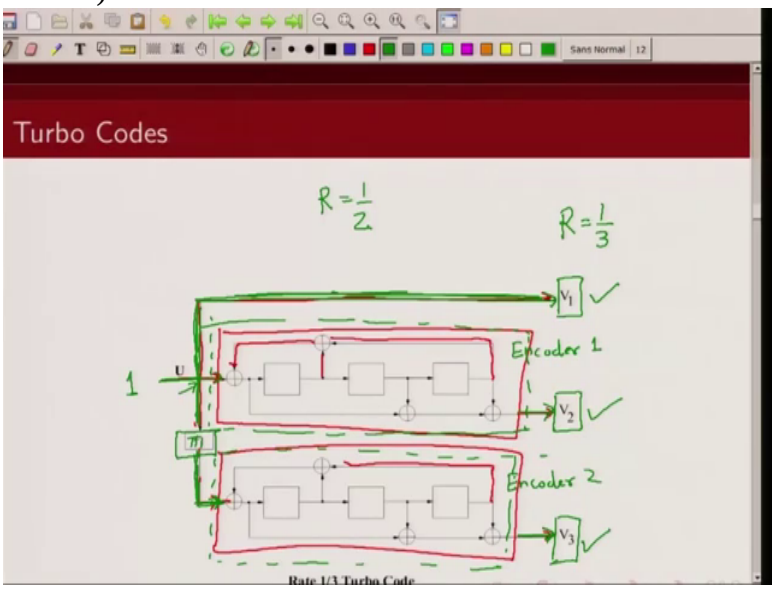

output, right? Now how do you get different rate codes? This is a rate 1 by 3 code. How do you get

(Refer Slide Time 25:12)



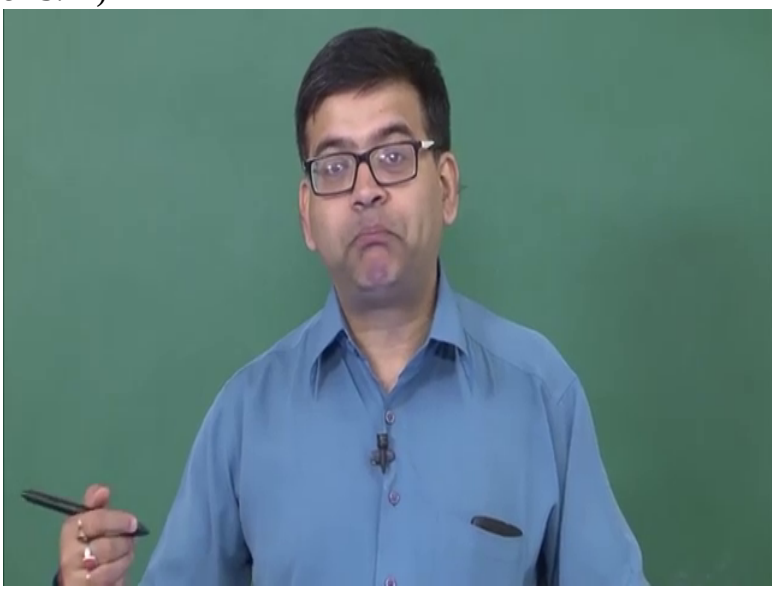

other rates? Let's say I want to design a rate one half turbo

(Refer Slide Time 25:17)



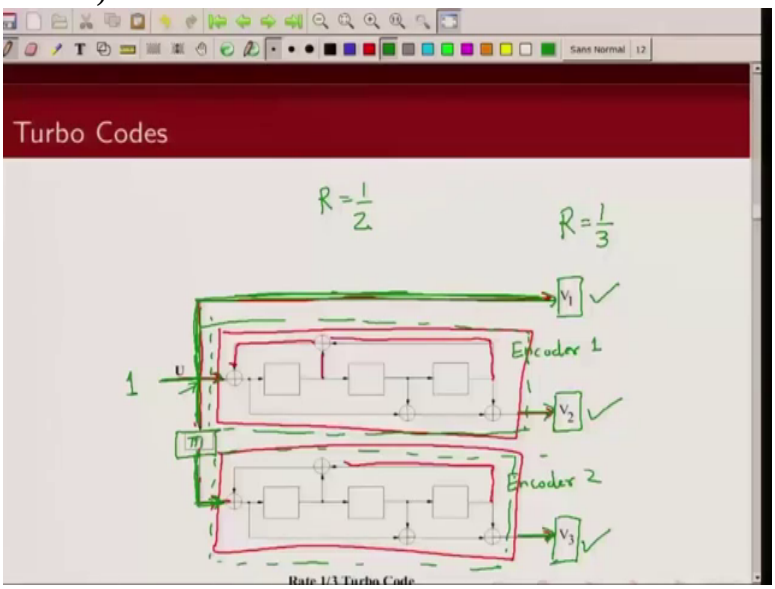

code. How do I do that? So I will do what, I can do, let's say what is called

(Refer Slide Time 25:24)



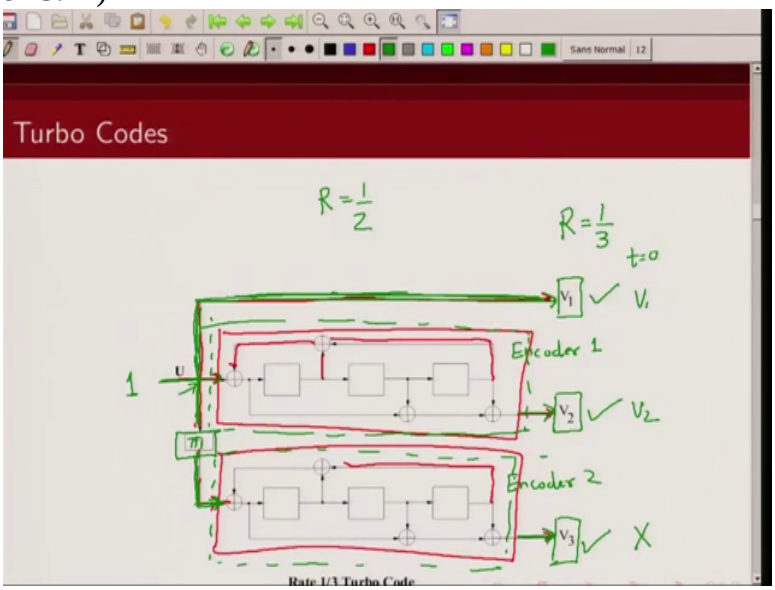
puncturing. So what do you do in puncturing? You remove some of the bits; don't send some of the bits. So let's say

(Refer Slide Time 25:32)



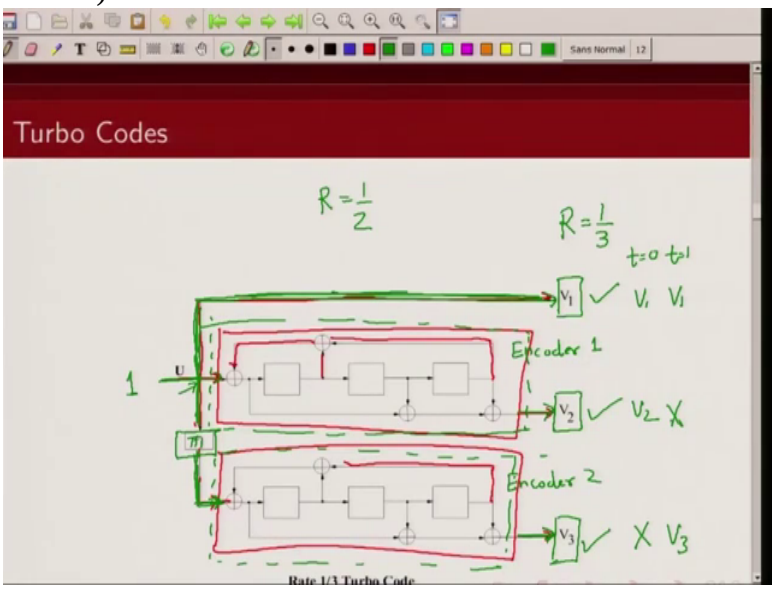

at time t equal to 0, I transmit v 1, and I transmit v 2. So I am transmitting 2 bits. I do not transmit

(Refer Slide Time 25:42)



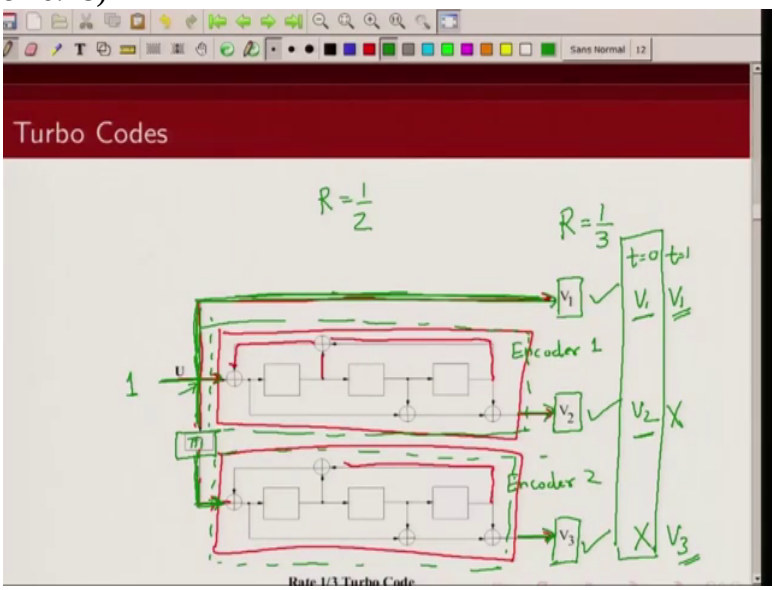

v 3. Then at time t equal to 1, I transmit v 1 and v 3. I do not transmit

(Refer Slide Time 25:52)



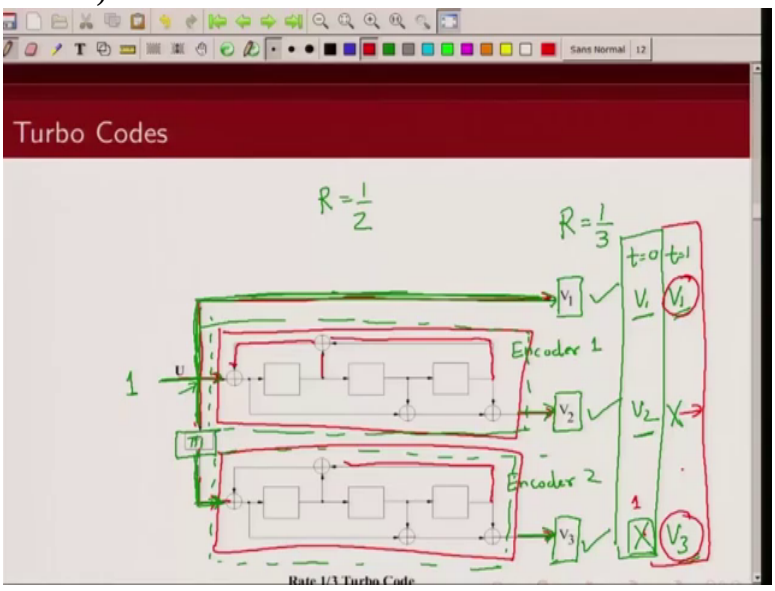

v 2. And this I repeat. So at every odd time I transmit v 1 and v 2 and every even, at t equal to 0, 2, 4 I transmit v 1 and v 2 and for other time I transmit v 1 and v 3. And at the receiver what I will do is, so at let's say I receive this bit, since I am not transmitting all

(Refer Slide Time 26:18)




the 3 bits so what I will do is for the bit that was not transmitted, I will assume that the bit is equally likely to be 0 or 1. So I would put it as likelihood ratio of this bit being 1 or 0 to be same. So, so that's what I will do. And similarly here, since I am getting received bit corresponding to only this and this, what I will assume here is I will feed to decoder that

(Refer Slide Time 26:49)

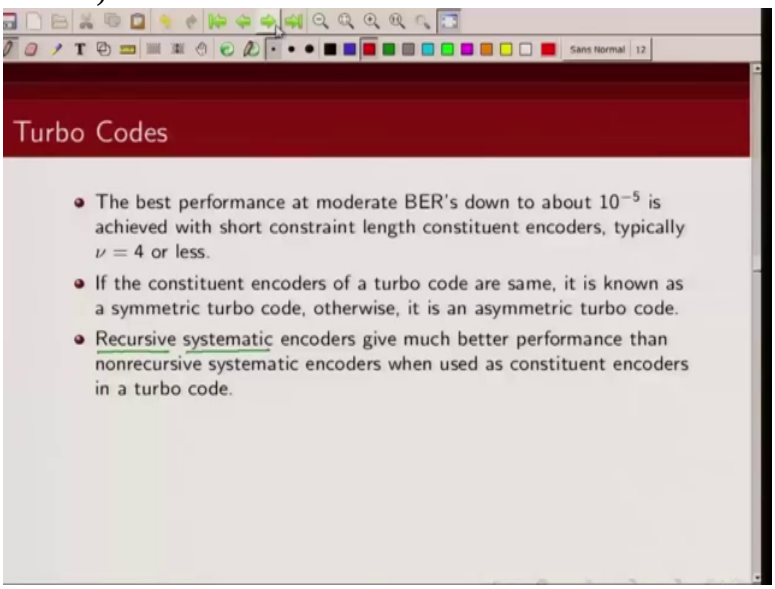



it is equally likely that this particular bit is 0 or 1 because I don't have any other information about this bit. So this is known as puncturing. So I am removing some of the bits. I am not sending some of the bits. So if you are interested in getting higher rates you can do

(Refer Slide Time 27:07)

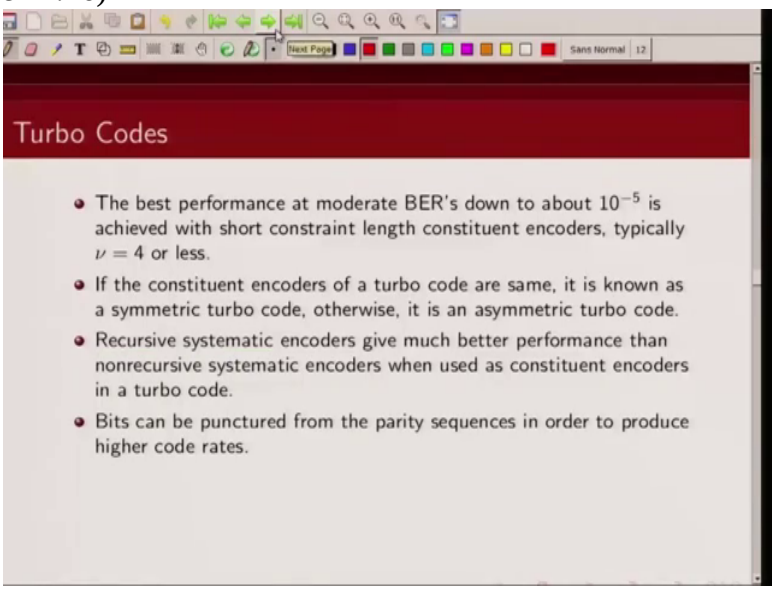
puncturing to get higher rate codes. And that's what I mean when I said

(Refer Slide Time 27:14)



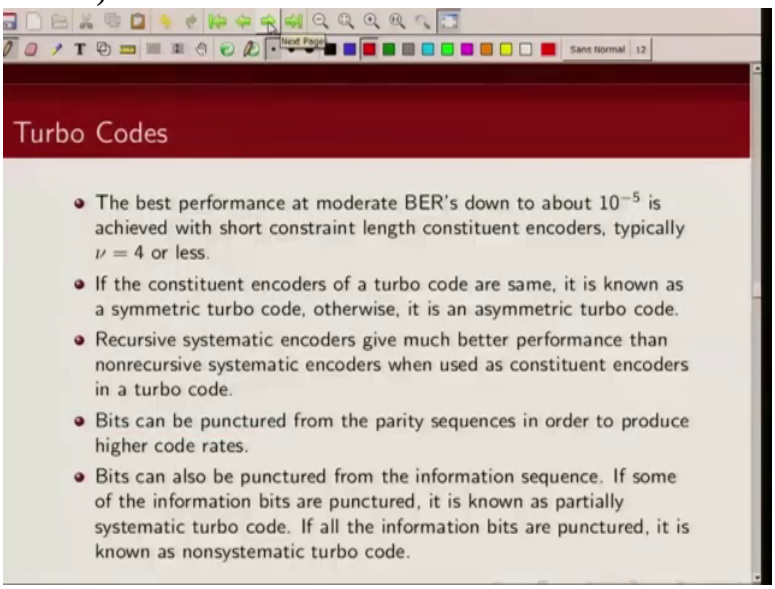

we could, bits can

(Refer Slide Time 27:16)



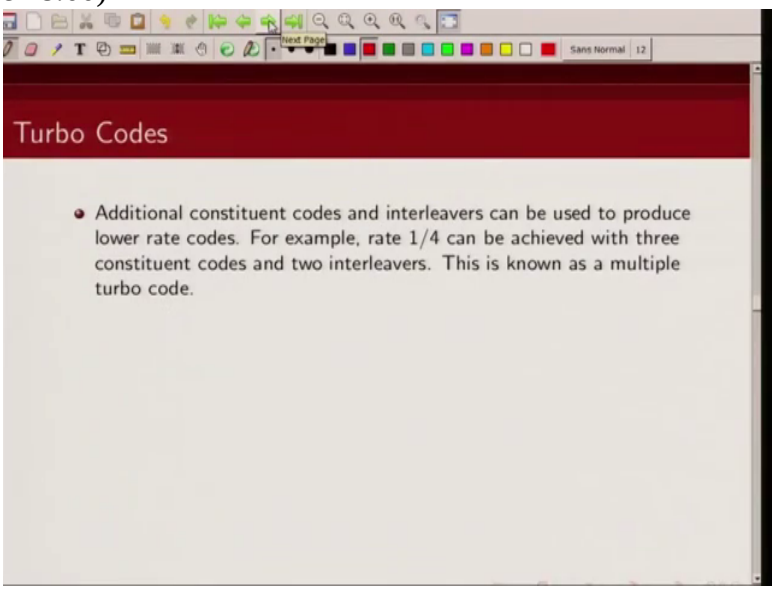

be punctured from the parity sequence to produce higher code rates.

(Refer Slide Time 27:25)



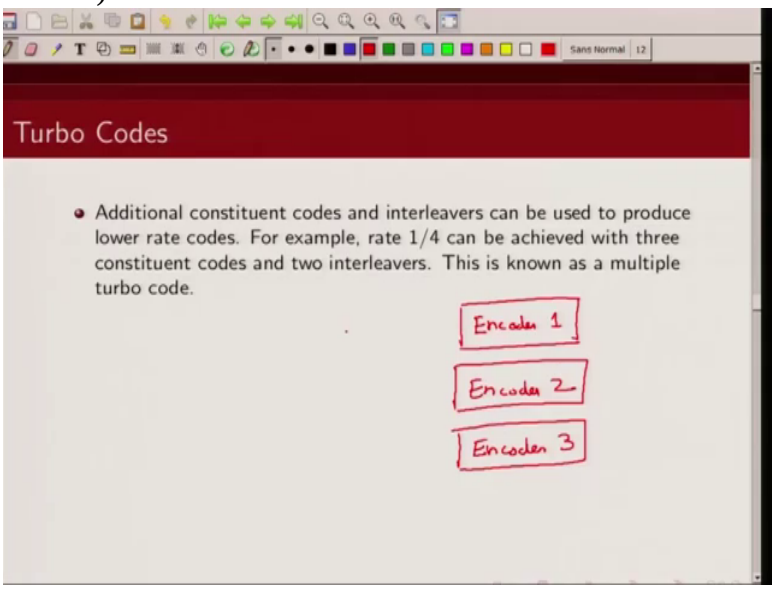

We could also do puncturing of the information bits. So I showed you an example of puncturing of the parity bits. I could also decide not to send the information bits. Now if some of the information bits are not sent, that means they are punctured, this particular code is known as partially systematic turbo code. And if I do not send any information bit, then of course you already know this is a non systematic code. Now

(Refer Slide Time 28:00)



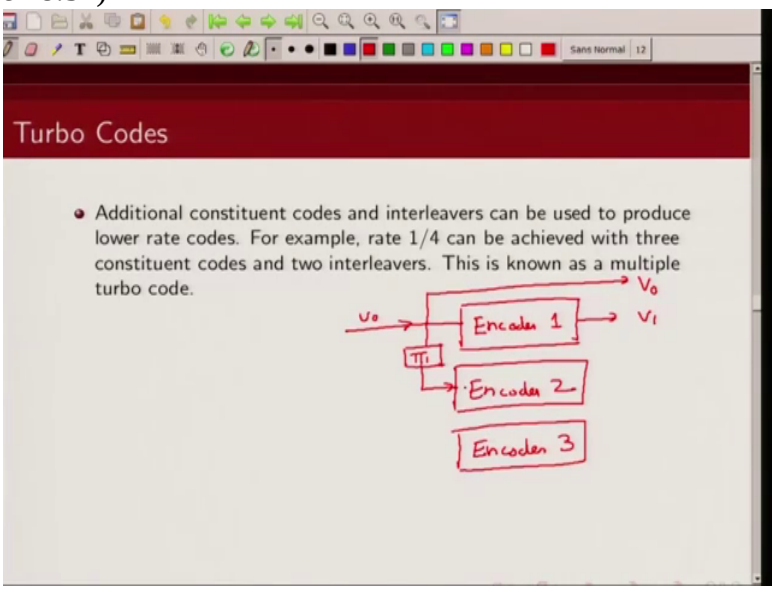

the encoder diagram that I showed you was using 2 encoders and 1 interleaver. Now this structure can be extended for multiple encoders and multiple interleavers. So I could have a situation, let's say I have encoder 1, I have encoder 2, I can have encoder 3, what I can do is I can have

(Refer Slide Time 28:39)



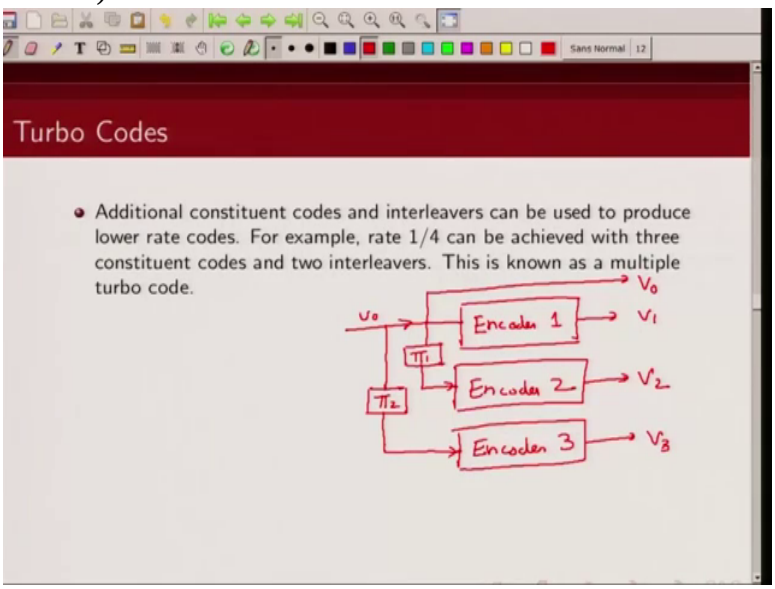

information sequence coming in, let's say u. So this is directly going out, this is v 0, this is v 1 and then I have interleaver which I am denoting by this pi so interleaved

(Refer Slide Time 28:57)




bits are coming to this encoder 2, it is generating some parity bits v 2. Then I have another set of interleaver. I am just denoting it by pi 2 and it's being fed to encoder 3, am I get (()).

(Refer Slide Time 29:13)

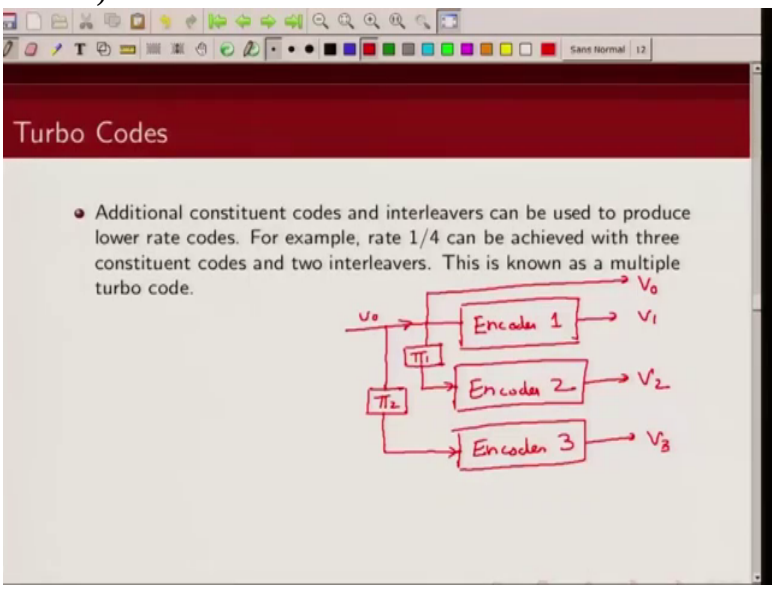



So the structure I have shown you for 2 encoders

(Refer Slide Time 29:17)

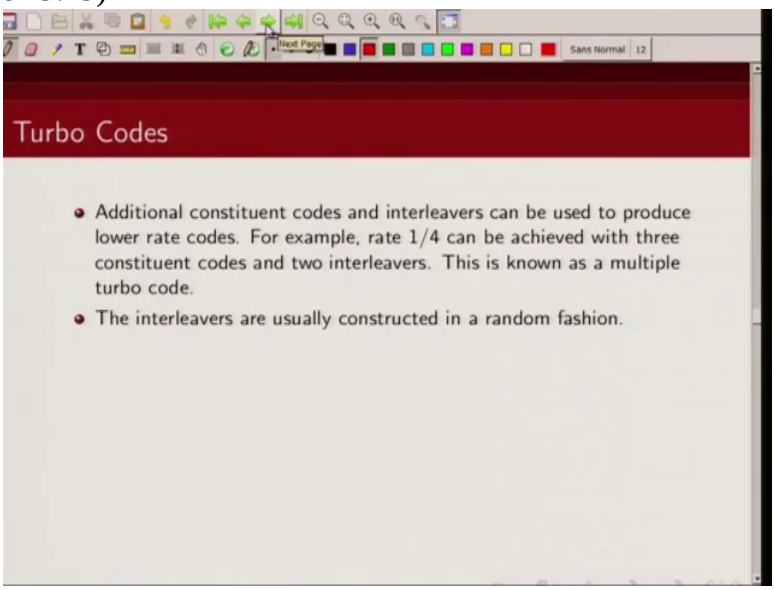
and 1 interleaver can be extended for multiple encoders

(Refer Slide Time 29:23)




multiple interleavers. So this will result in a rate 1 by 4, what we call multiple turbo code. So we use the term multiple turbo code. So this is the extension of the conventional turbo code which uses 2 encoders and 1 interleaver.

(Refer Slide Time 29:49)

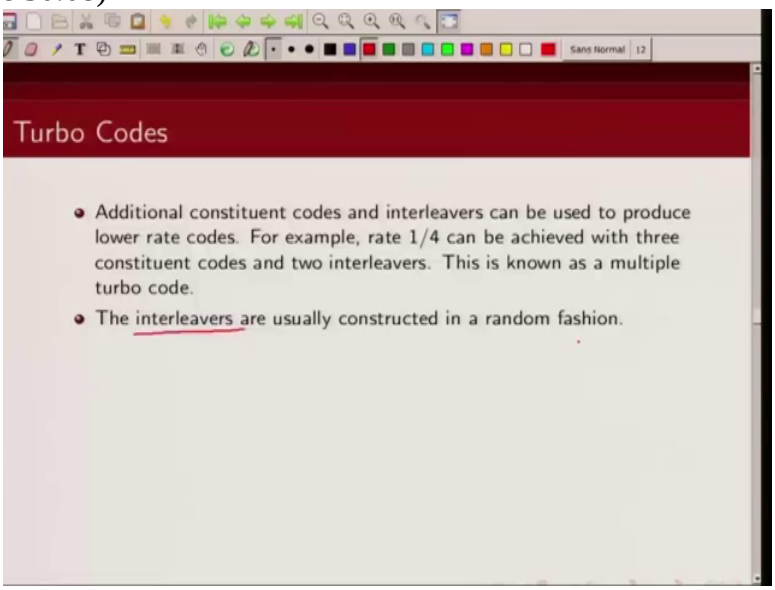



Now I have shown you that design of interleaver is very, very crucial. And there are

(Refer Slide Time 29:58)

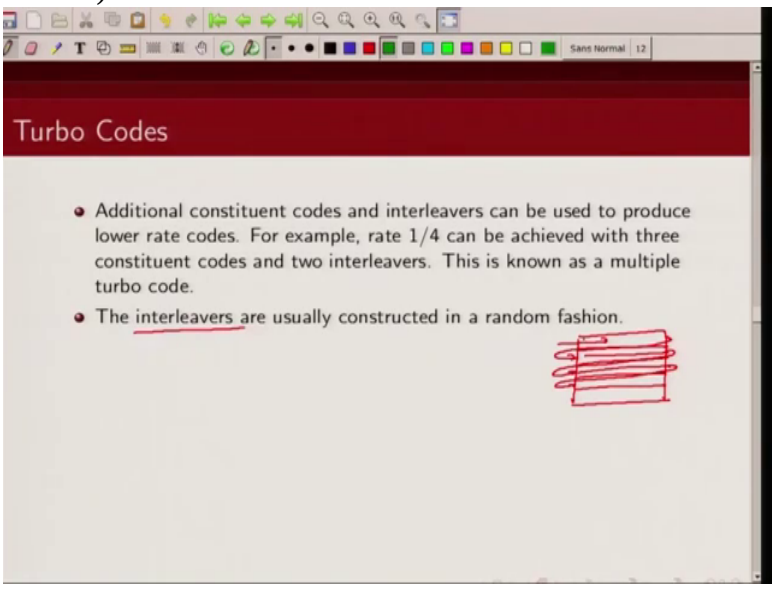
multiple techniques you can use for designing interleaver. A very simple way is what is called block interleavers. So you,

(Refer Slide Time 30:08)



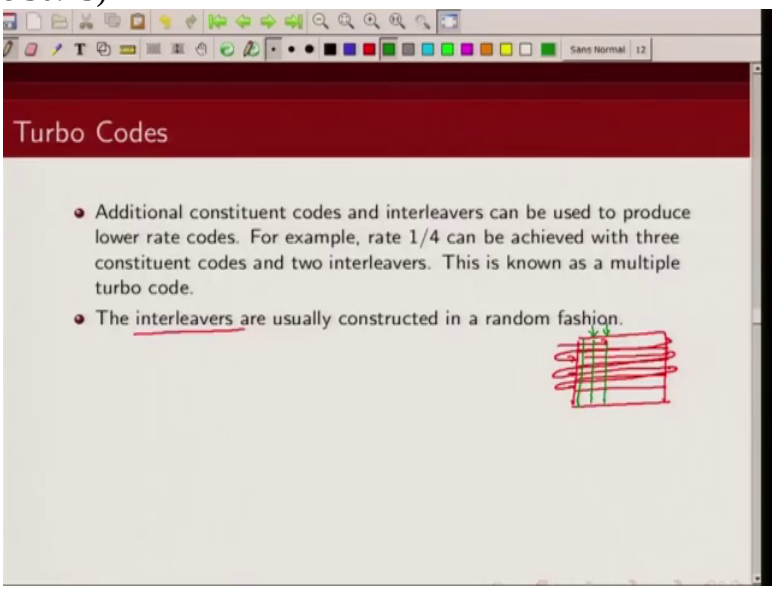

the information that you want to interleave or you want to send to the second encoder, what you can do is you can store that data block wise, row wise. So you fill that data row wise. So once you come here, you then fill up, so this is how you fill up the data, Ok. So you are filling up the data row wise. But when reading out, you could read the data column wise. So I can

(Refer Slide Time 30:36)



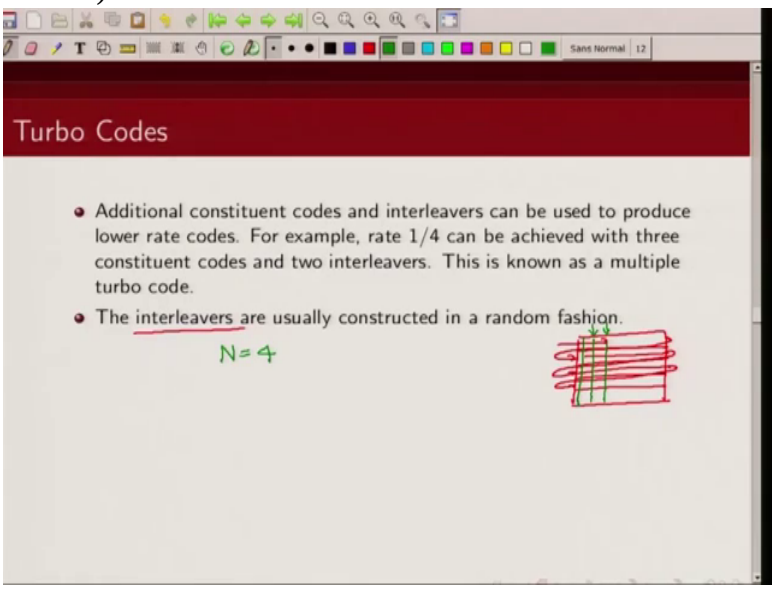

read data like this and then I come here, read this data from here, so this is

(Refer Slide Time 30:43)



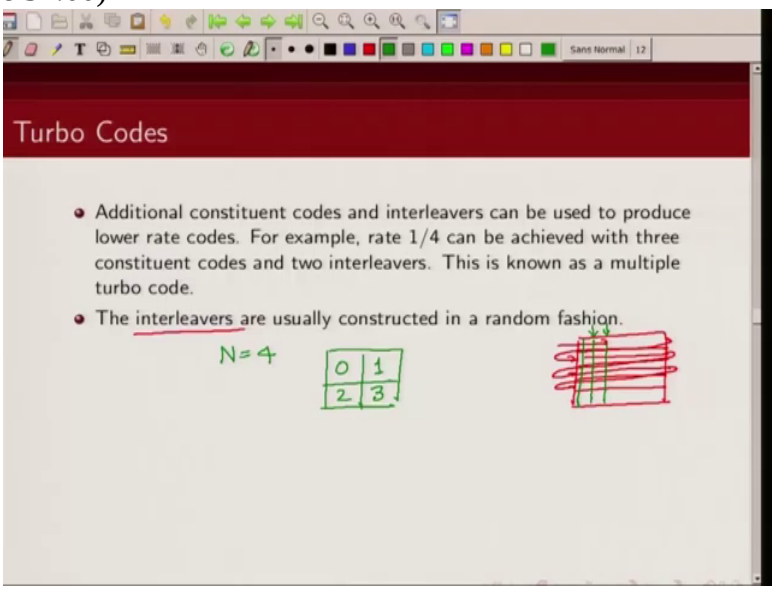

known as block interleaver. A very simple example, let us say my block size is 4. So what I can do

(Refer Slide Time 30:51)



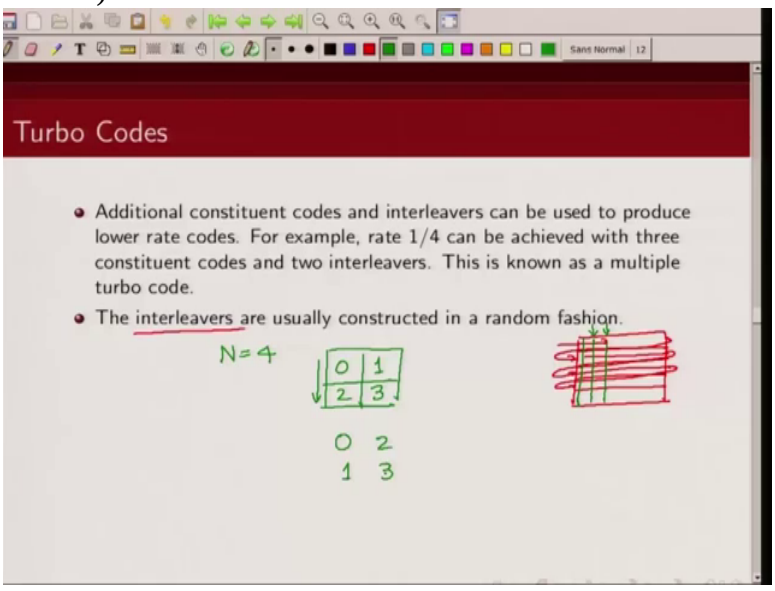

is I can put data like this. First data I can put it here, second data I can put it here, third data I can put here and the fourth

(Refer Slide Time 31:00)



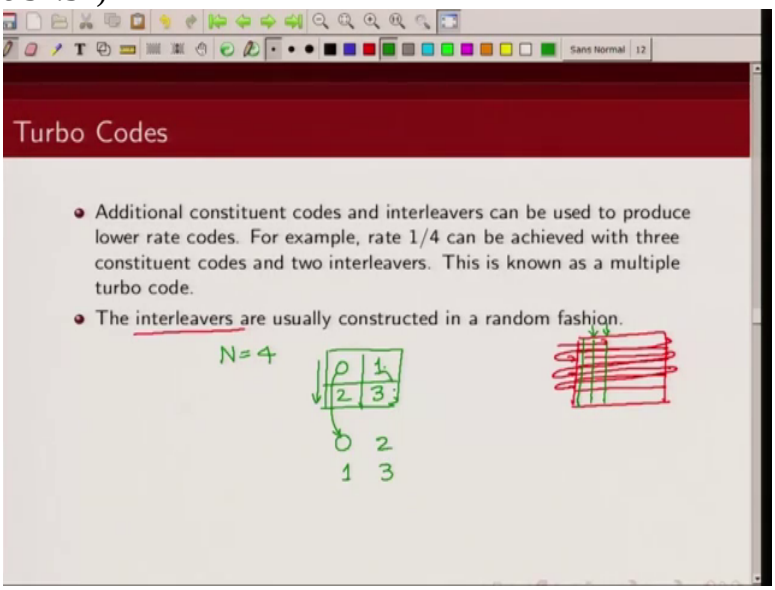

data I put here. But while reading instead of reading row wise I can read it column wise. So I first read 0, then I read second, I read one and I read third.

(Refer Slide Time 31:14)



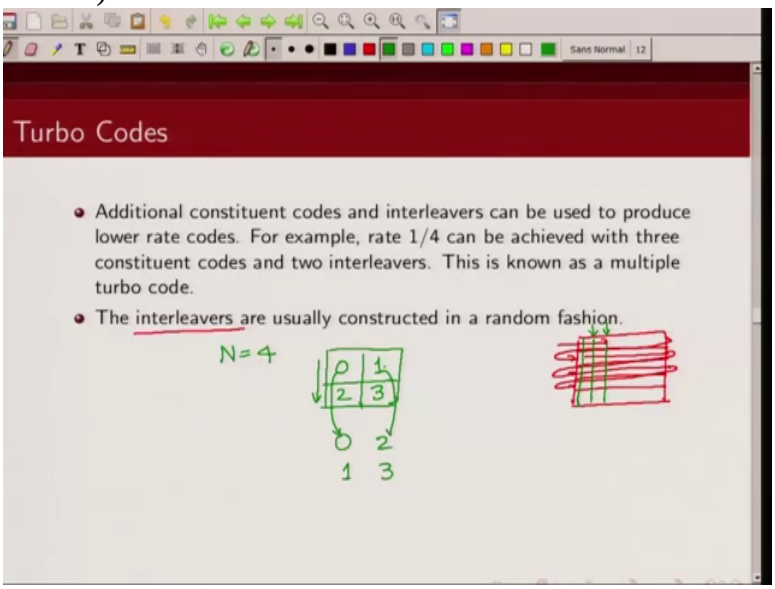

So what did I do? So this was my original data. But interleaved version is now, what was there in, so in location this, same bit is there but in this location now I am having

(Refer Slide Time 31:32)




what was

(Refer Slide Time 31:33)

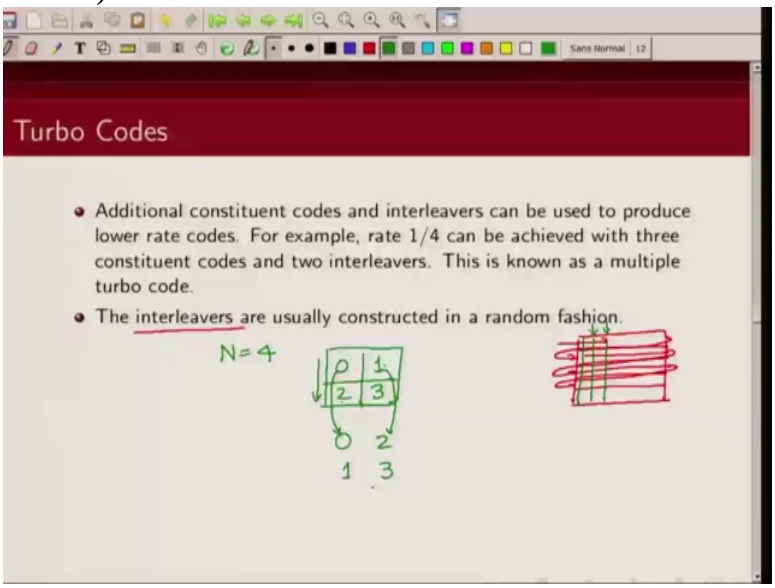



earlier there in this location and in this location I am having what was earlier I had in this location. So interleaving is nothing but just reordering of data. You could also design it randomly. I will decide Ok, first bit I will put in this location, second bit I will put in this location. I just have to ensure that every bits are put in distinct location.

(Refer Slide Time 31:55)

No two bits are put in same location, right. Ideally what you would like is if there are like bit streams coming in which have consecutive 1s, they should be spread out after interleaving. That's you would like

(Refer Slide Time 32:10)

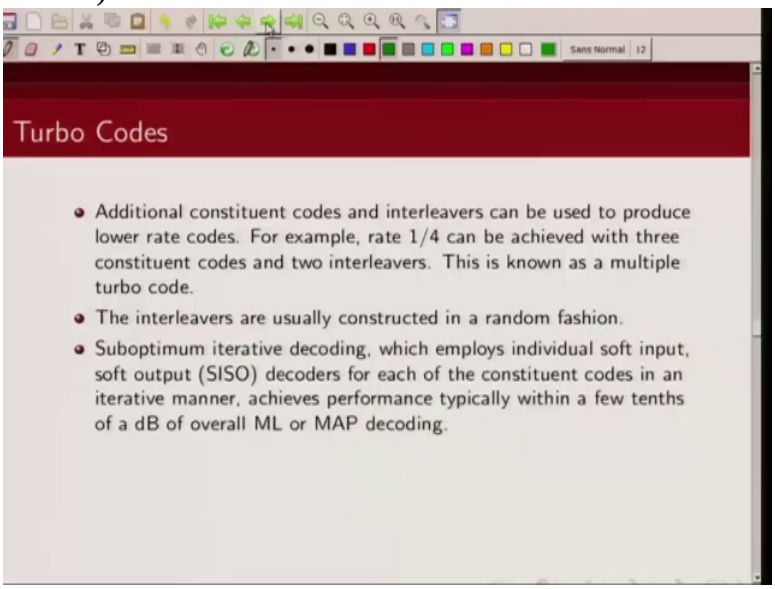



to do. And there are class of interleavers like s random interleavers which imposes some additional constraints which will allow that

(Refer Slide Time 32:19)

two adjacent bits are at least spread by some amount,

(Refer Slide Time 32:23)

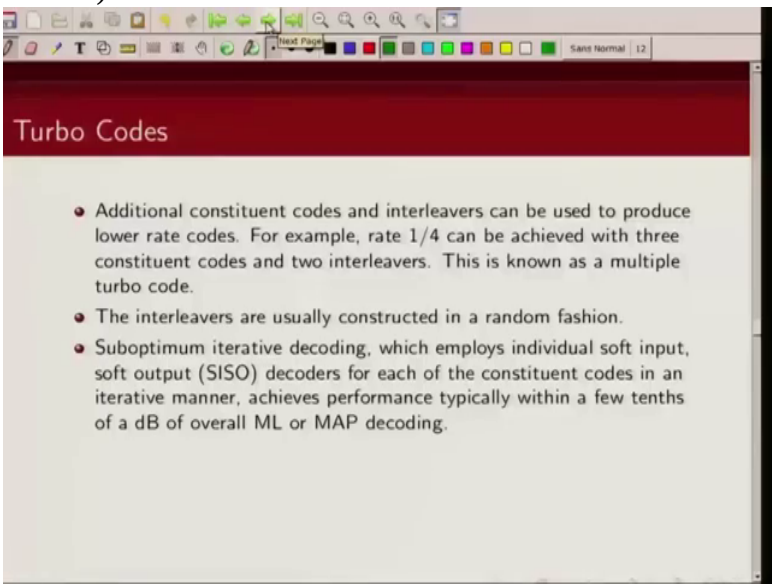



Ok. So that's interleaver. So you have seen how we are constructing the encoder. It is a parallel concatenation of 2 encoders, right? And you have a interleaver. Now how do we decode these codes? We are going to use an iterative algorithm. So we are first going to

(Refer Slide Time 32:45)

decode one encoder, then we are going to decode second encoder. And these decoders will pass on information to, among each other. So after the first encoder decodes, it will pass on some information to second decoder saying Ok, I think these are the information bits. Now the second encoder will take that input and process and it will give a new set of estimates to the first encoder saying, Oh no I think these are the values of information sequence. And this process happens in a iterative fashion. In fact the name turbo code comes from the decoding structure. The turbo engine has this feedback mechanism and decoding structure of this turbo code is also the feedback. So from one encoder to other encoder and this is basically going around. So

(Refer Slide Time 33:39)

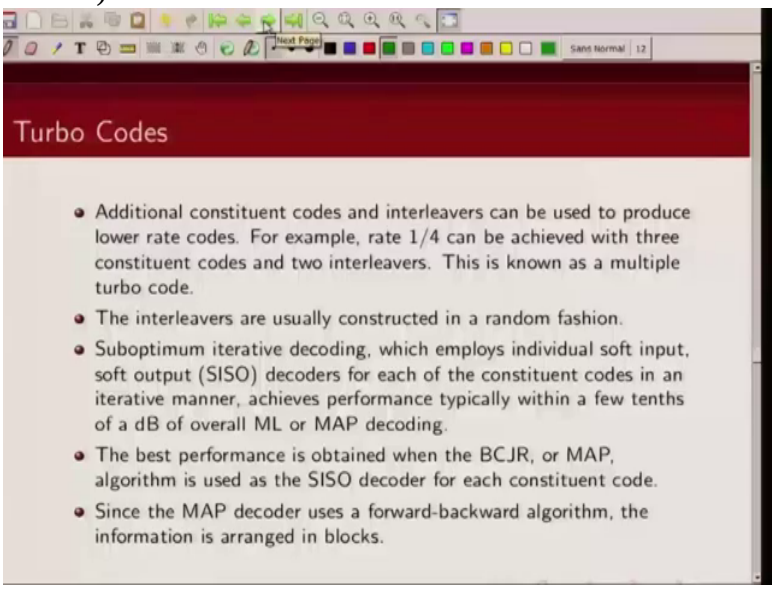


from there this name turbo codes have come. So we are going to use iterative decoding algorithm and each of these component encoders are going to use this B C G R algorithm and we already, when we talked decoding of convolutional code, we already talked of how B C G R algorithm works. In the next lecture we will talk about how

(Refer Slide Time 34:04)

the turbo decoding algorithm works.

(Refer Slide Time 34:07)

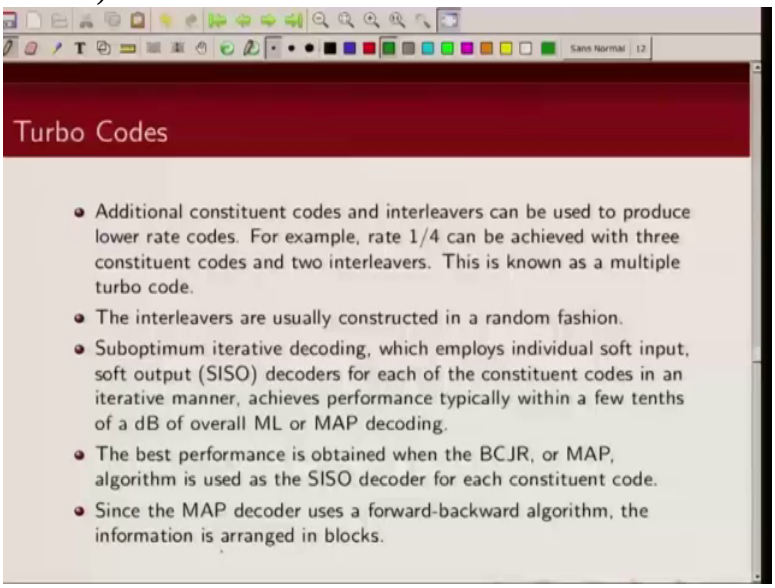



And this decoding, as I said happens by block fashion. So you wait until you receive

(Refer Slide Time 34:14)

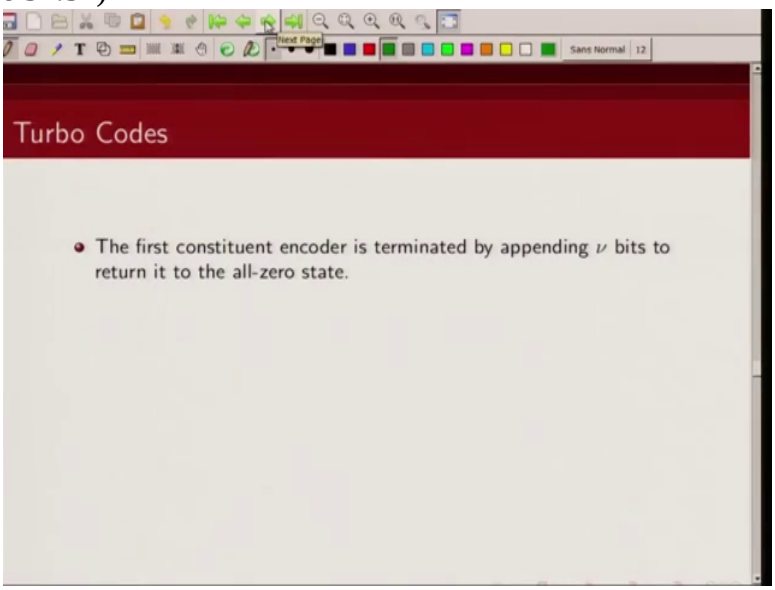
n bits of data because remember this interleaving operation, those it can be parallelized, typically done in block of data. So you process, you decode the bits also in block by block fashion. So the information can be arranged

(Refer Slide Time 34:30)



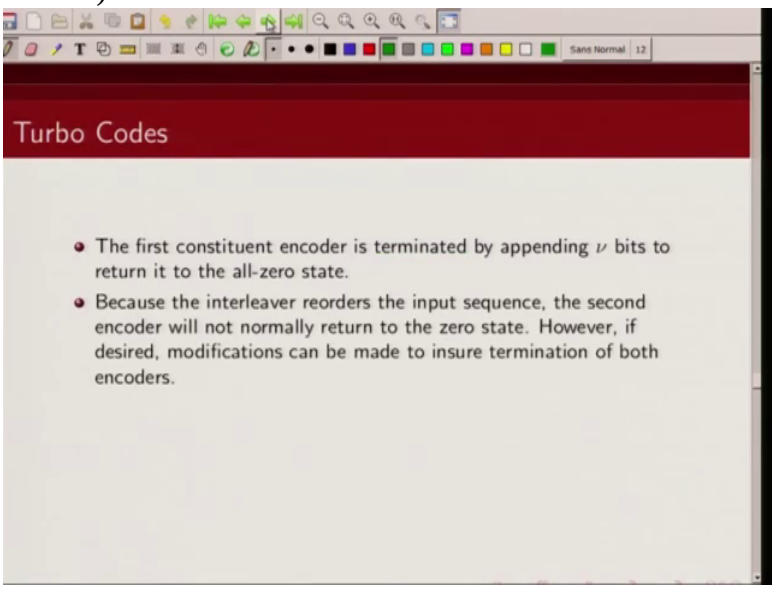

in blocks of data

(Refer Slide Time 34:34)




So typically for better performance, as I mentioned we typically tried, initially the encoder is in all zero state and we like to bring the encoder back to all zero state after we have transmitted our information sequence so that is known as termination and we terminate this turbo encoder by appending some tail bits to our information bits. Now because

(Refer Slide Time 35:07)

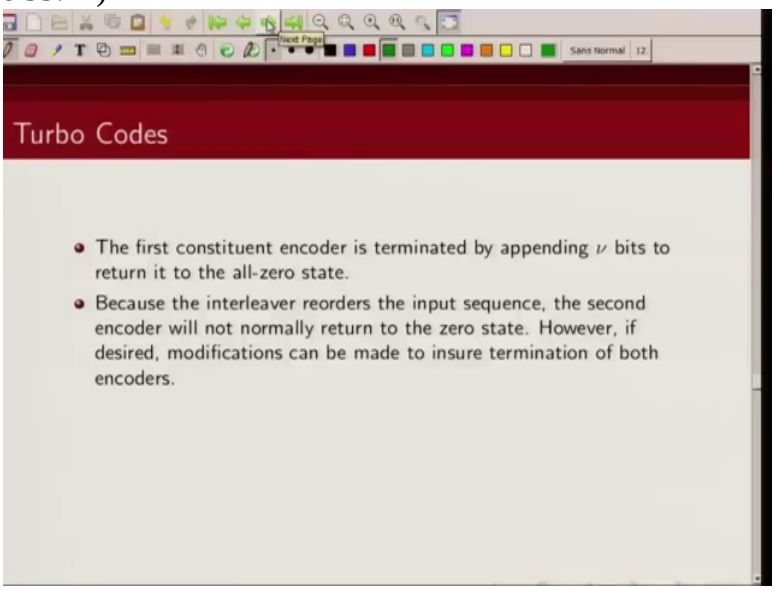



of the interleaver it is not always possible to have the same tail bits terminate both the encoders. You could design an interleaver which will allow you to terminate both the encoders, that's possible. But in general it is not possible

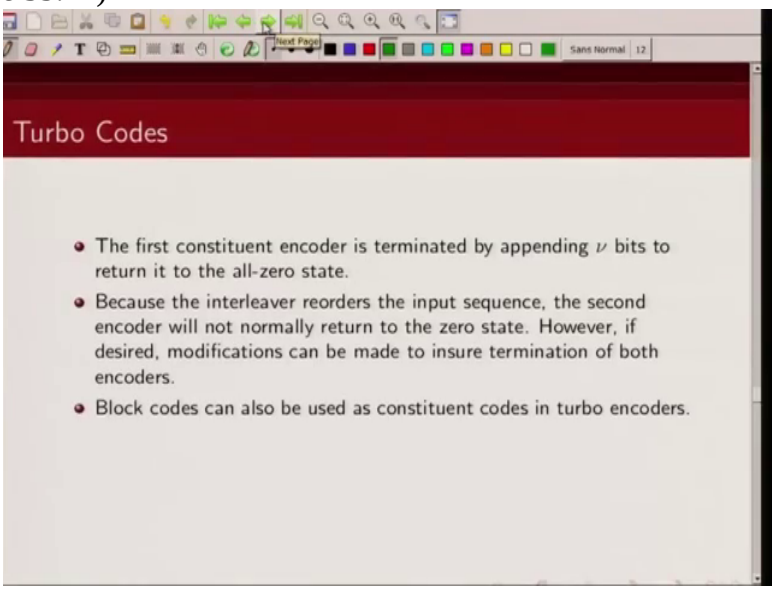
(Refer Slide Time 35:24)

to terminate both the encoders with same terminate bits. So you could either terminate first encoder, leave the second encoder unterminated or you could send additional bits to terminate the second encoder.

(Refer Slide Time 35:41)



(Refer Slide Time 35:42)

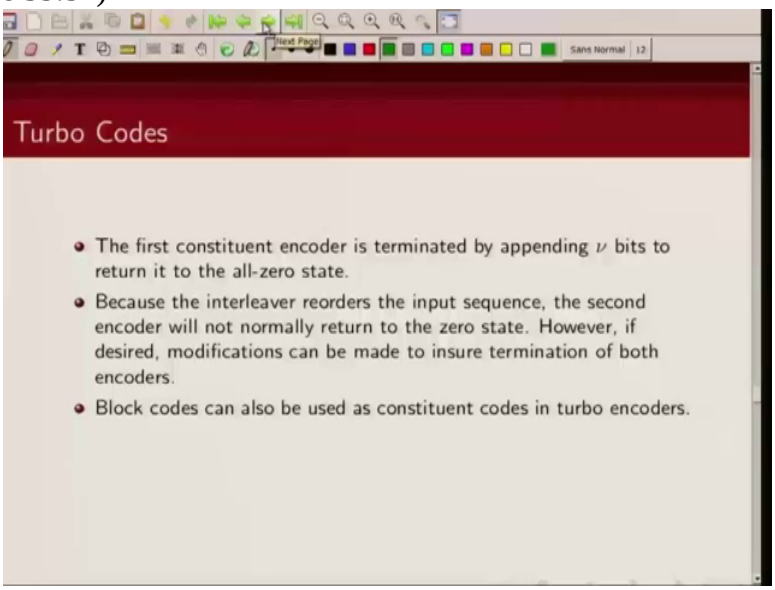



So far I have shown you that we are using recursive systematic convolutional encoder in a

(Refer Slide Time 35:49)

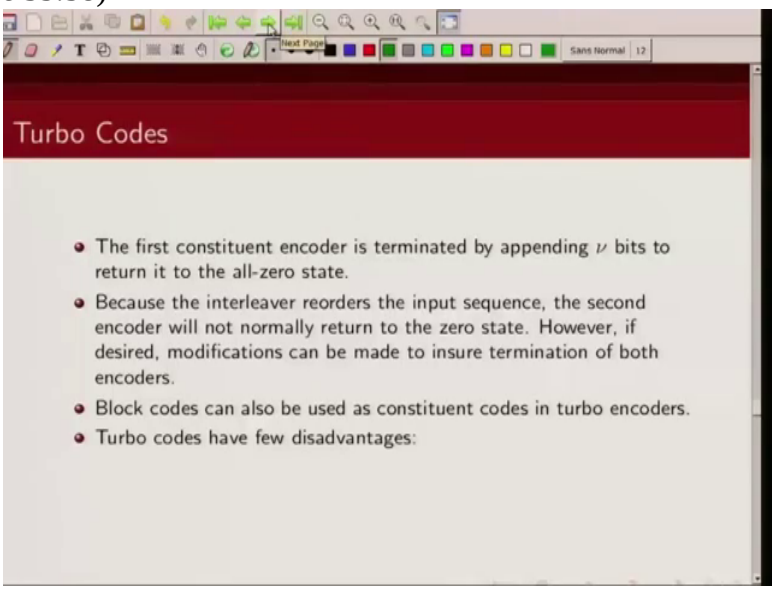
parallel configuration. We could also design same thing using block codes as well. And we are not going to discuss

(Refer Slide Time 35:57)



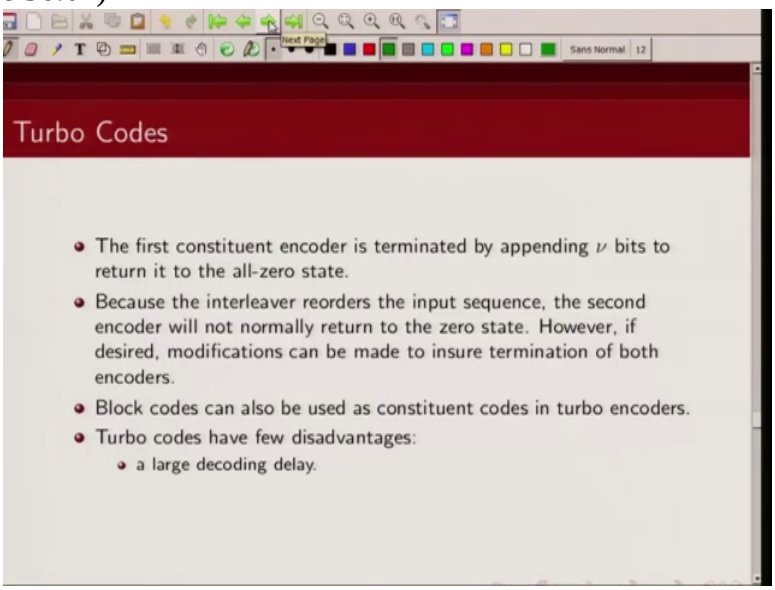

those

(Refer Slide Time 35:58)



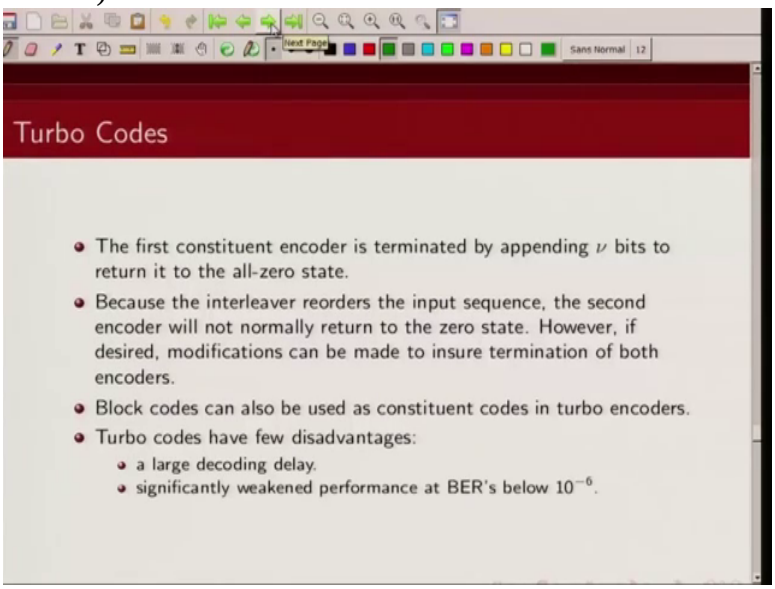

in this lecture. Now there are some disadvantages of turbo code. Though their performance is extremely good, there are some

(Refer Slide Time 36:07)




inherent problems with turbo code.

(Refer Slide Time 36:09)

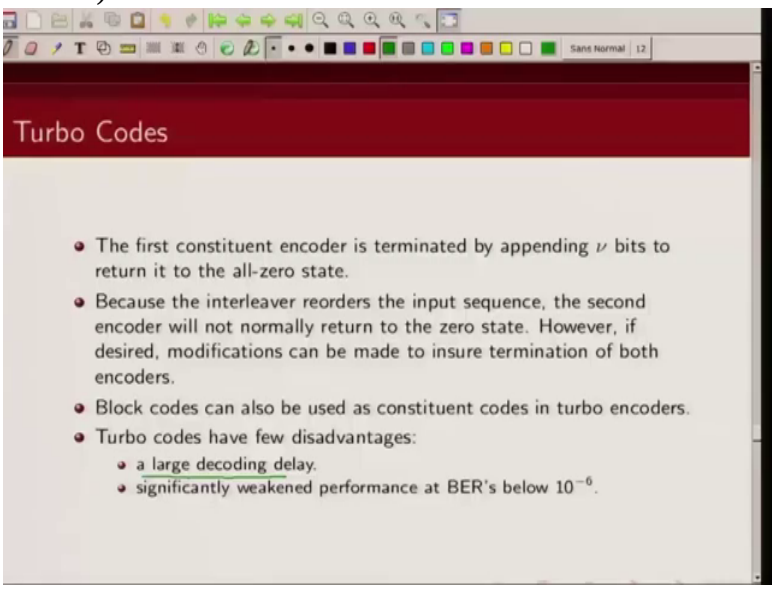



One is they have large decoding delay. Typically the performance is good for large block size.

(Refer Slide Time 36:18)

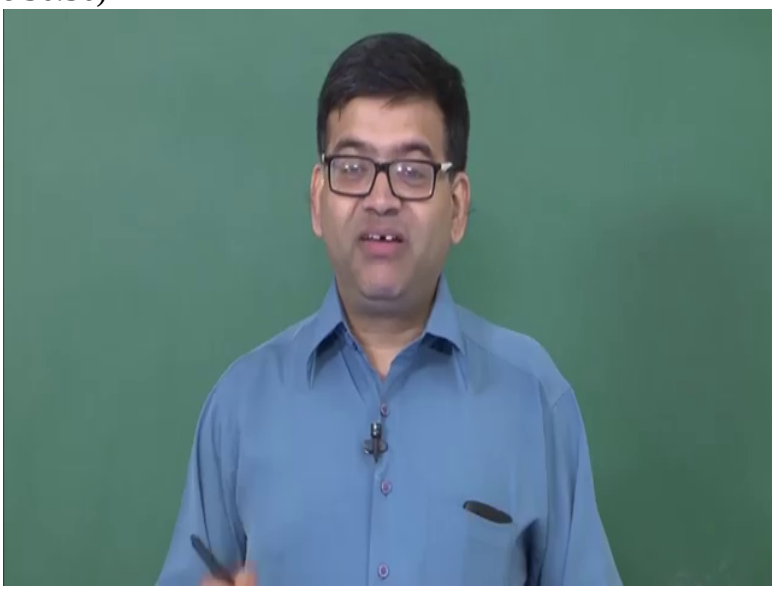
So when you are dealing with large block size and there is lot of delay involved because you are doing block by block processing

(Refer Slide Time 36:27)



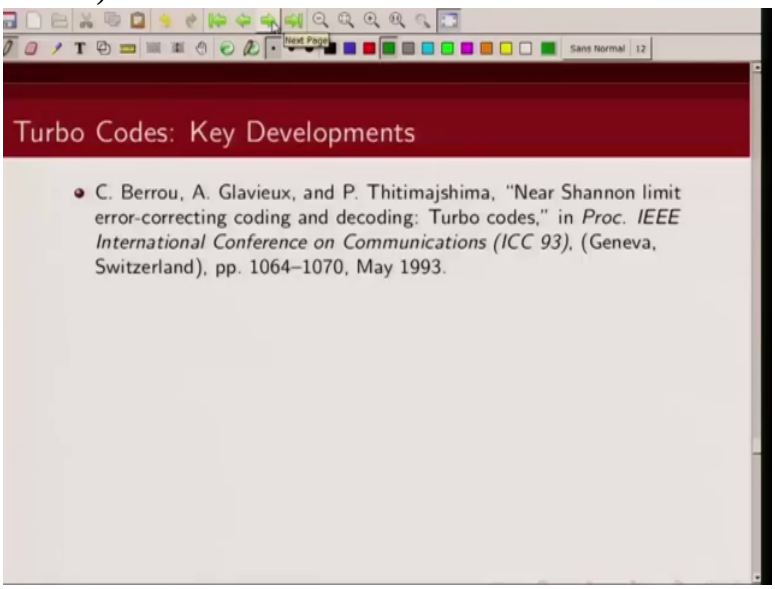

and second thing is typically if you are looking at very, very low bit error rate performance, there

(Refer Slide Time 36:36)



you just saw that because of error floor their performance is not very good at very, very low bit error rate. Now

(Refer Slide Time 36:48)



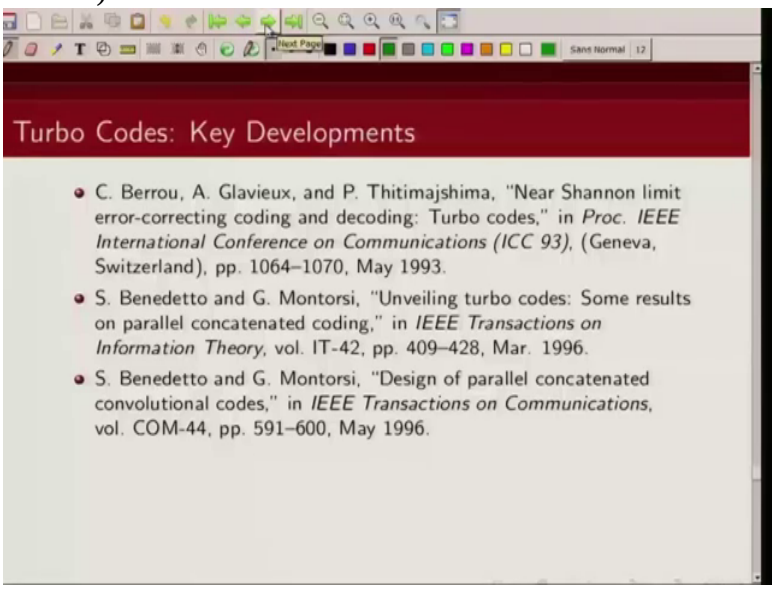

I will just conclude this lecture by giving you some references of some key early development in turbo codes and you can read those references.

So this is where turbo codes were initially introduced by paper by Berrou et al in I C C 1993. When this paper came people didn't actually believe the results because they were very good performing codes but subsequently multiple research groups were able to reproduce

(Refer Slide Time 37:18)

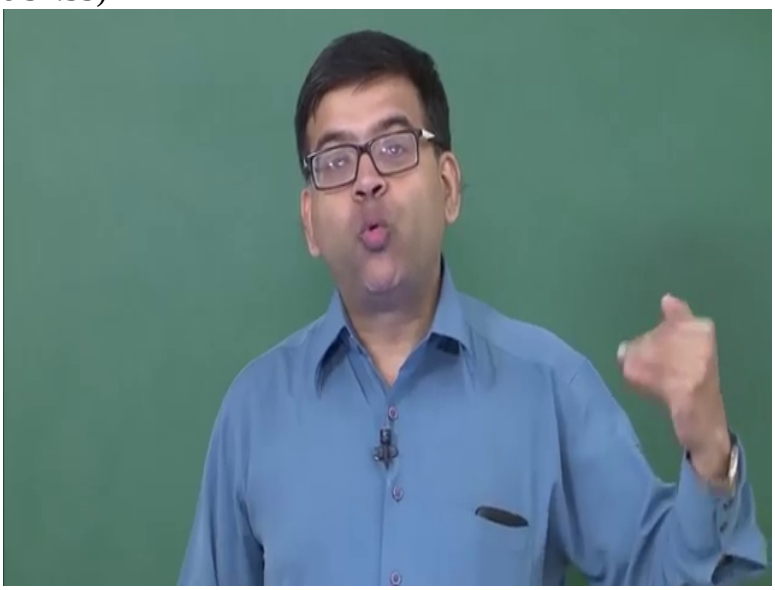
their results.

(Refer Slide Time 37:20)



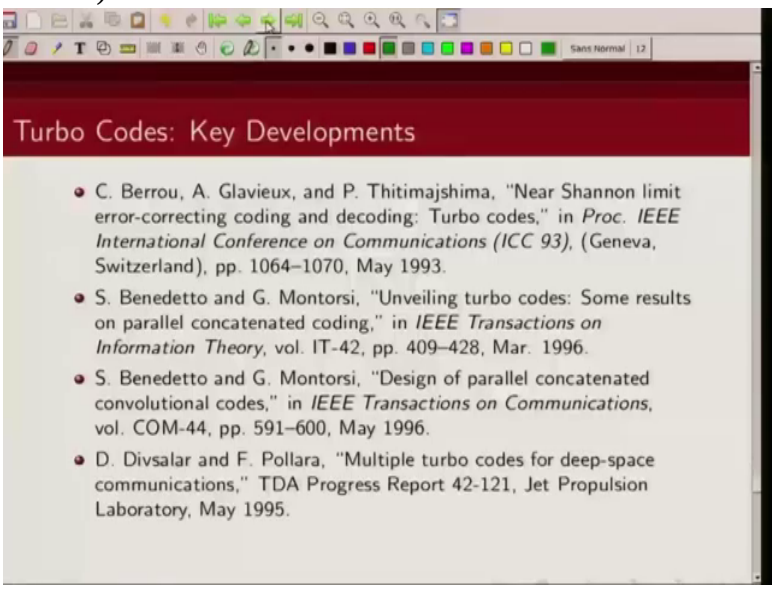

Now the next set of papers, these 2 papers by Benedetto et al, Unveiling Turbo Codes, some results on parallel concatenated coded and Design of parallel concatenated convolutional codes, these were one of the first papers to explain why

(Refer Slide Time 37:35)



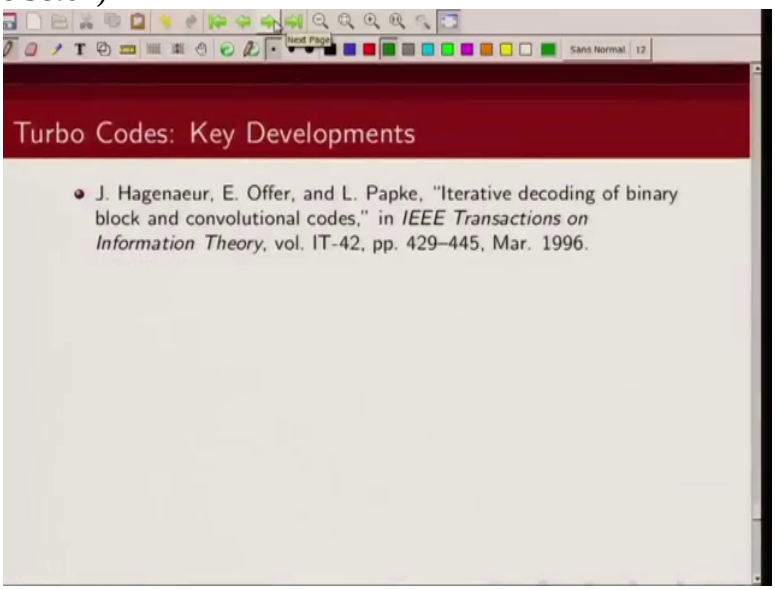
the performance of turbo codes is not very good at high signal to noise ratio and why do they suffer from error floor.

(Refer Slide Time 37:47)



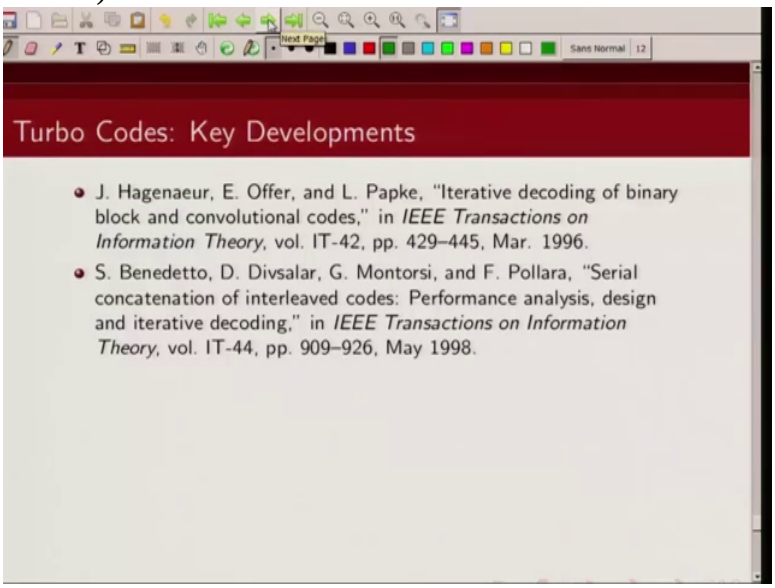

These were the first paper where they generalized by Divsalar at J P L lab where they generalized 2 multiple turbo codes and this paper also gave some design rules of how to design turbo codes.

(Refer Slide Time 38:04)




This is a nice, by Hagenaeur this is nice paper on iterative decoding of binary block convolutional code, this explains the B C G R algorithm and turbo decoding, B C G R algorithm in a very nice way.

(Refer Slide Time 38:22)

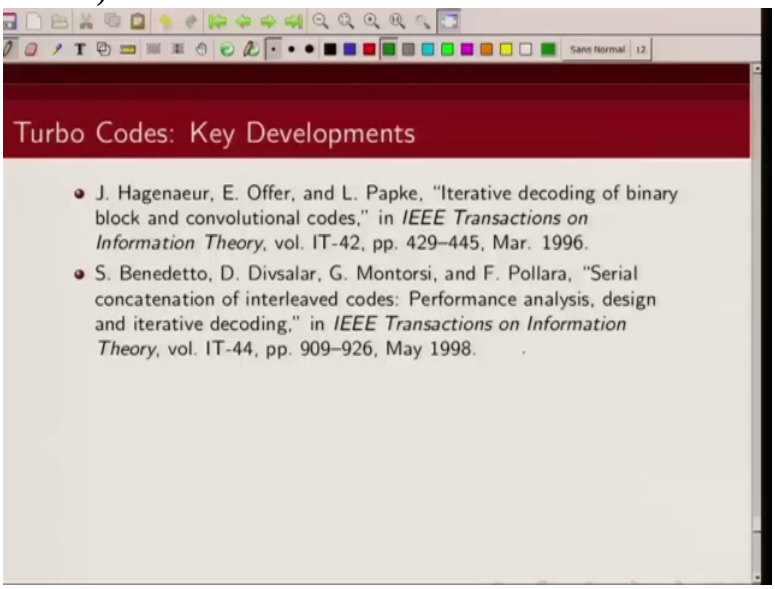



Now in this lecture we talked about parallel

(Refer Slide Time 38:27)

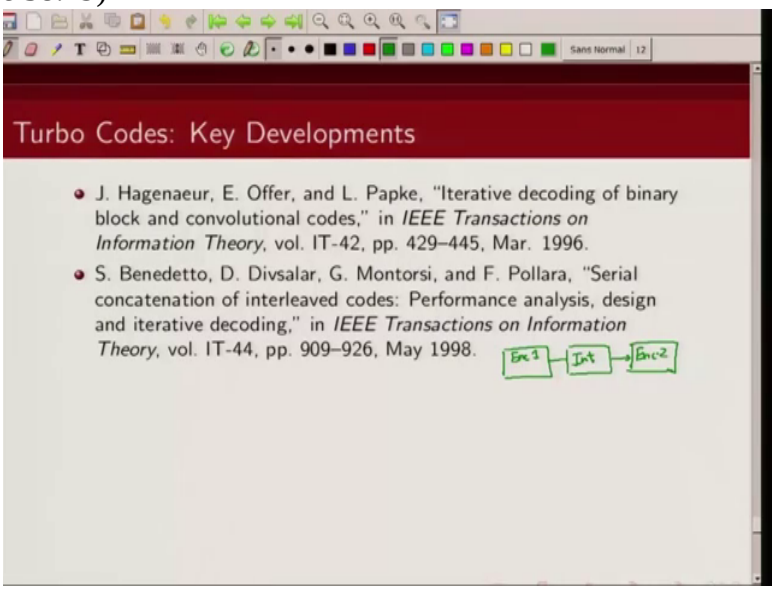
concatenation. There is a different variant where you could have a serial concatenation.

(Refer Slide Time 38:32)



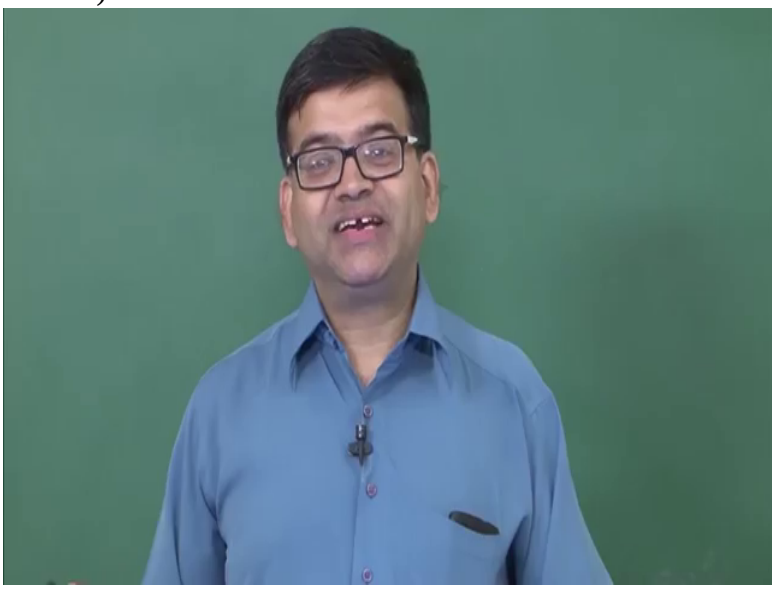

So you could have a encoder 1 here, then you could have an interleaver here, interleaver and you could have another encoder. So this variation

(Refer Slide Time 38:49)



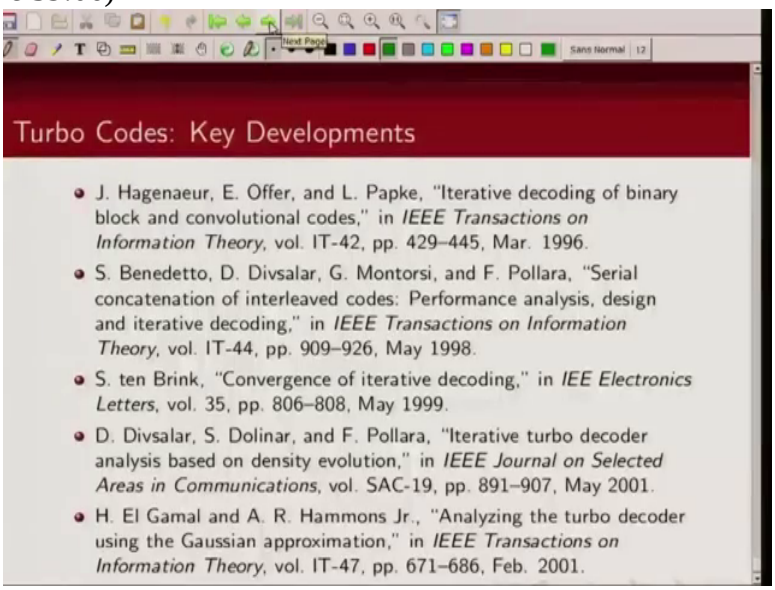

was proposed by Benedetto and others and this serial

(Refer Slide Time 38:54)



concatenation though has little inferior performance in the Waterfall region, it has better performance in the error floor region. And finally these 3

(Refer Slide Time 39:06)

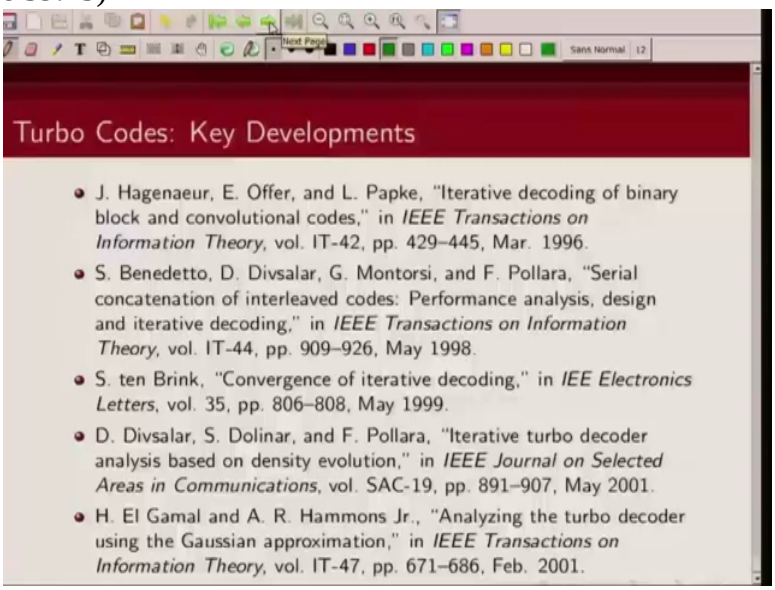



papers that you have discussed the behavior of iterative

(Refer Slide Time 39:12)

decoding algorithm. Why choice of some encoders is good, how should you choose the encoders, so that you have better performance in the waterfall region, this was

(Refer Slide Time 39:25)




nicely explained in these three papers and each one of them used different techniques? ten Brink used mutual information, Divsalar used evolution of the densities of the expressed information and El Gamal used mean and variance of the expressed information and they tracked it to get information about the

(Refer Slide Time 39:50)

convergence of the turbo code. So with this I am going to conclude my discussion on turbo codes, thank you.