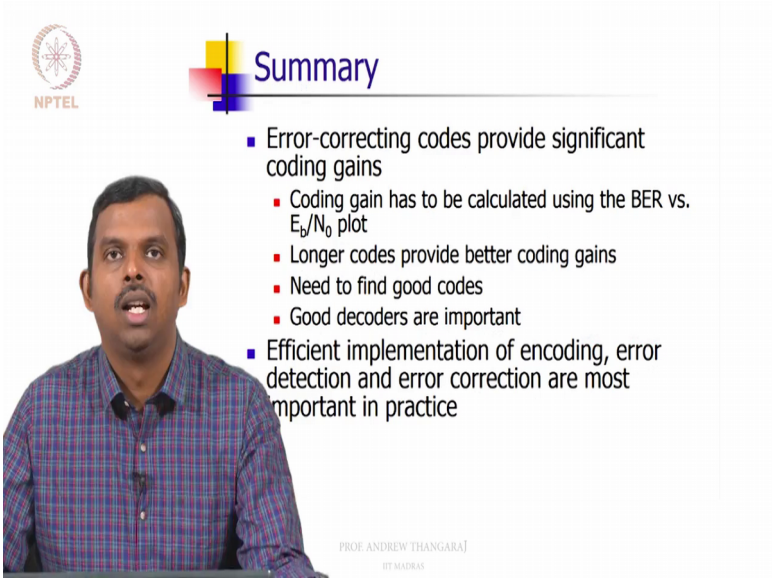
LDPC and Polar Codes in 5G Standard
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology Madras
A Brief Introduction to Linear Block Codes

(Refer Slide Time 00:16)



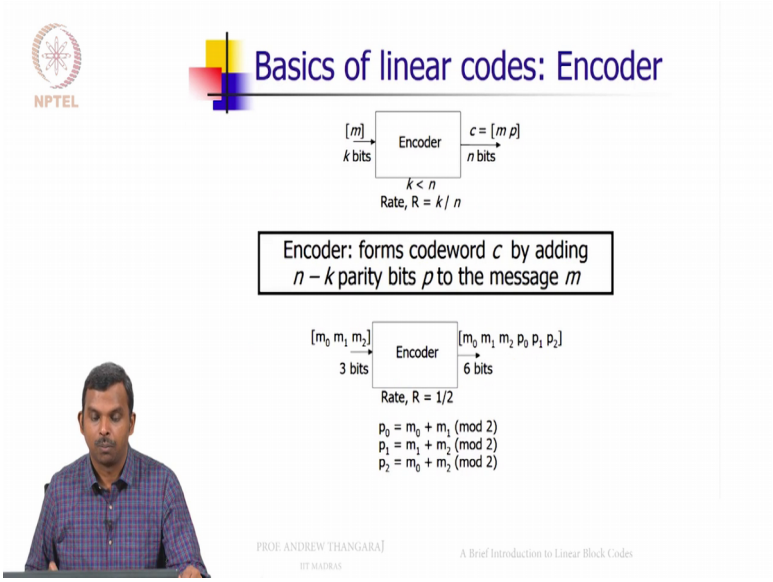
Ok. Welcome to this lecture. We are going to proceed talking about error control codes. In particular we will study linear error control code generator matrices, parity check matrices etc as the main topic in this lecture, Ok.

So I would like to begin with the summary of what we did in the previous lecture. We saw through some simple examples then that error control codes can provide coding gains, the coding gain which to be computed in the B E R versus E b over N naught plot and longer codes provide good coding gains but we need to find good codes, not every big code is going to be good.

Also once we have a code, we need to able to decode it well. We need, in particular soft decision decoders which are very important, Ok. So overall I think finding a good code and efficiently implementing and coding, decoding etc. is the main challenge and this has been overcome today and we will describe that in the rest of this course, Ok.

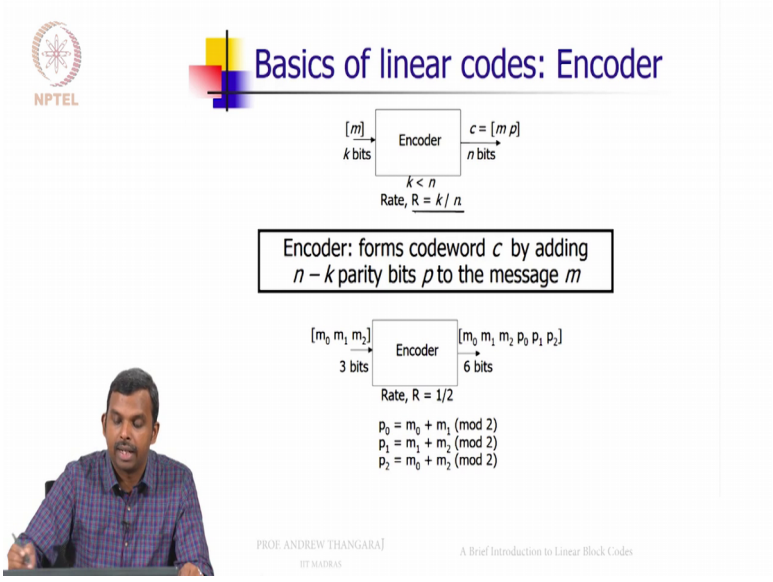
So let us proceed with today's lecture,

(Refer Slide Time 01:17)



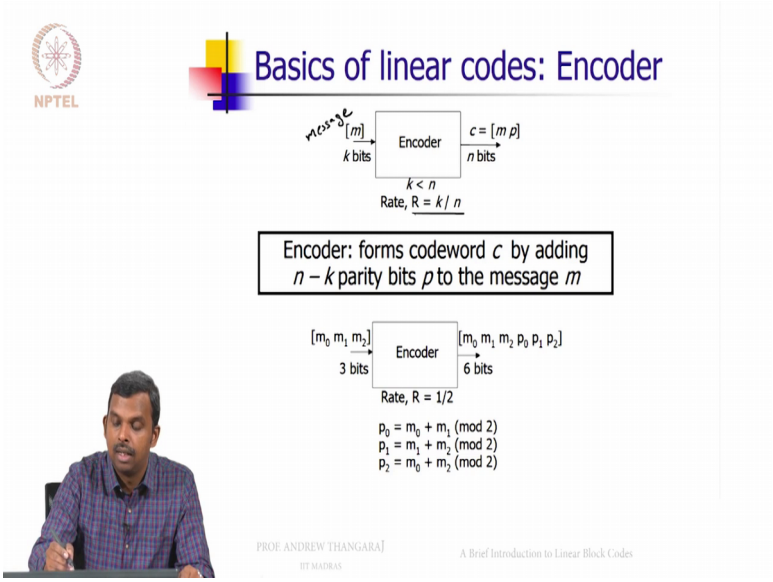
Ok. So today's lecture is on linear codes. We will begin by describing encoders for linear codes. So if you see here, this is a rate k by n code.

(Refer Slide Time 01:32)



Ok, there are k message bits coming in. This is the message m. There are k message bits

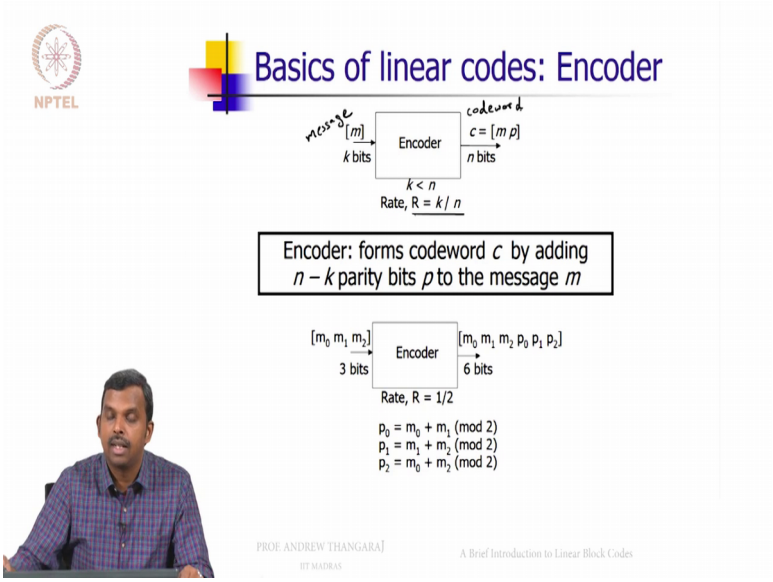
(Refer Slide Time 01:39)



and this is the codeword c.

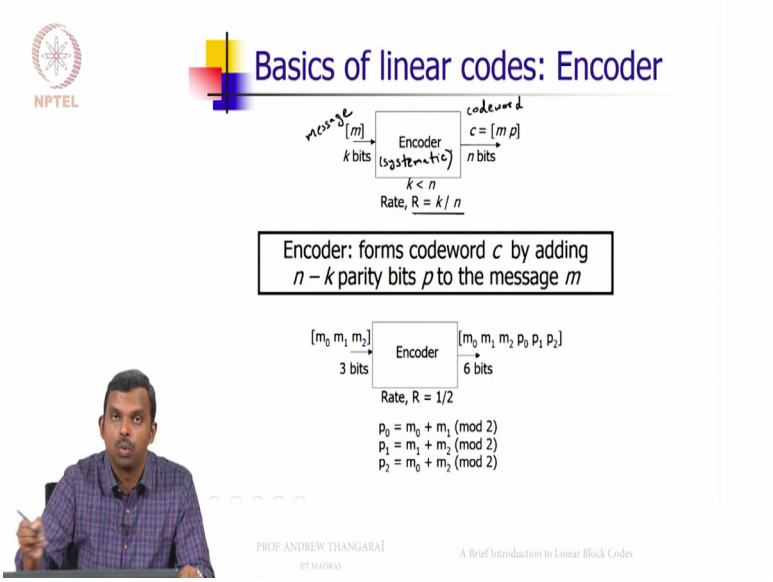
And you can see

(Refer Slide Time 01:45)



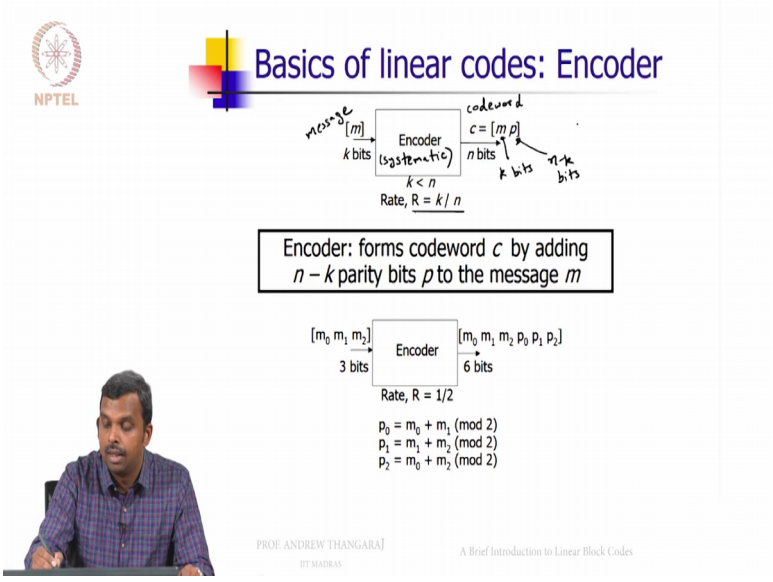
that I have assumed that the encoder is systematic in the sense that the message appears as a part of the codeword.

(Refer Slide Time 01:54)



Ok, message will appear in the part of the codeword. I have chosen n bits for the codeword, first k bits is the message and n minus k bits is the p

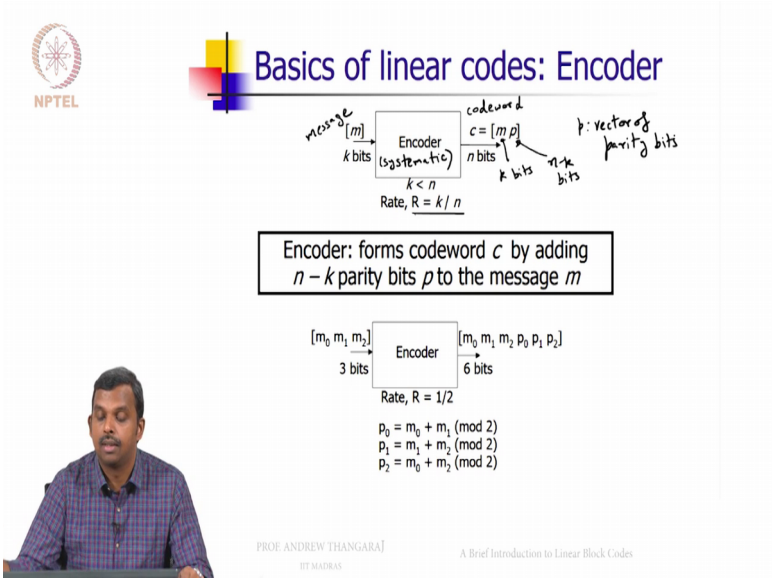
(Refer Slide Time 02:11)



part and this p vector is the set vector of parity bits, Ok.

So this is

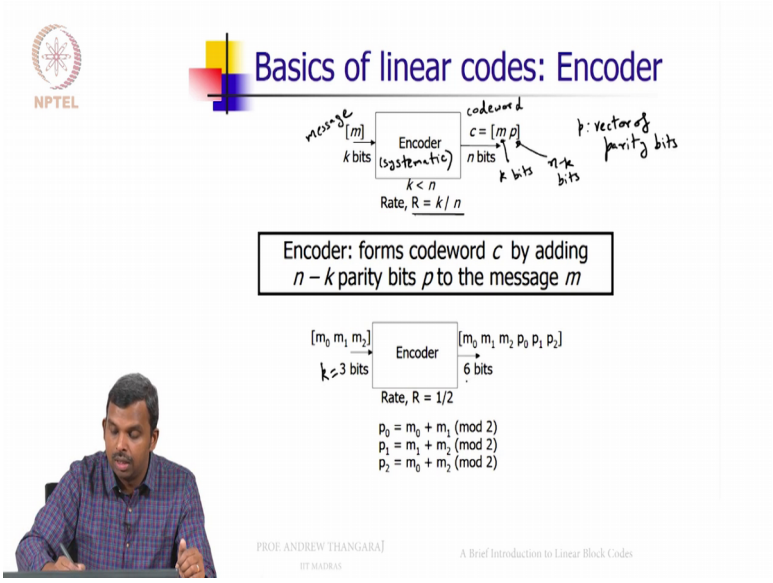
(Refer Slide Time 02:23)



the most common picture of error control code. You have a message. You send that message. Along with that message, to protect it from errors and provide coding gain we add parity bits and sent that as a codeword. Ok

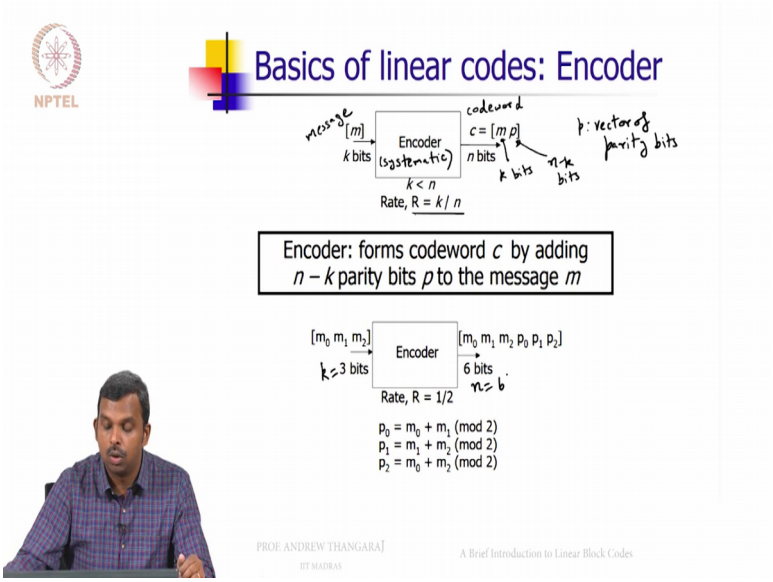
So the rate is k by n, k is less than n, this is the picture. So here is a simple example. So it is always good to have a simple representative example and then understand the whole thing. And here is an example of how a linear block code would work. You have a rate half, 3 bits for the

(Refer Slide Time 02:57)



message and 6 bits for the

(Refer Slide Time 03:00)

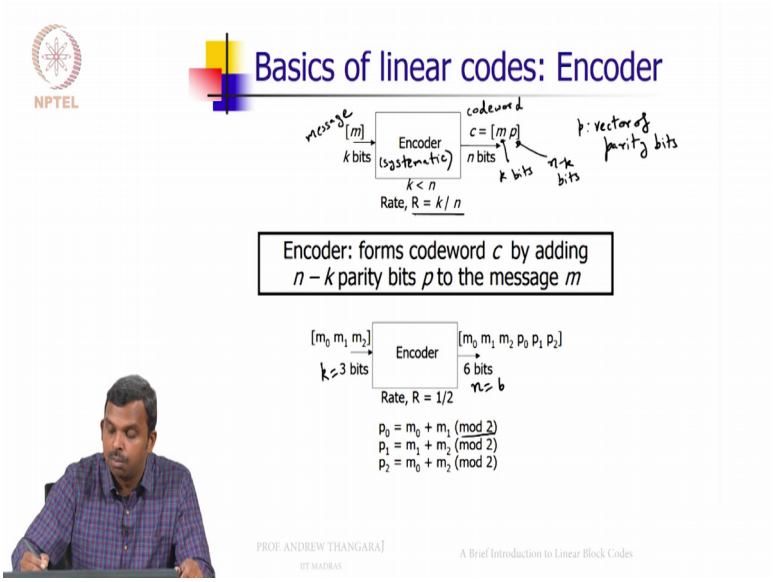


codeword, Ok.

And m 0, m 1 and m 2 are the three message bits and the codeword is composed of m 0, m 1, m 2 first and then 3 parity bits, p 0, p 1, p 2. And how are these parity bits computed? They are computed by linear operations, linear operations like addition etc. but there is this modulo 2.

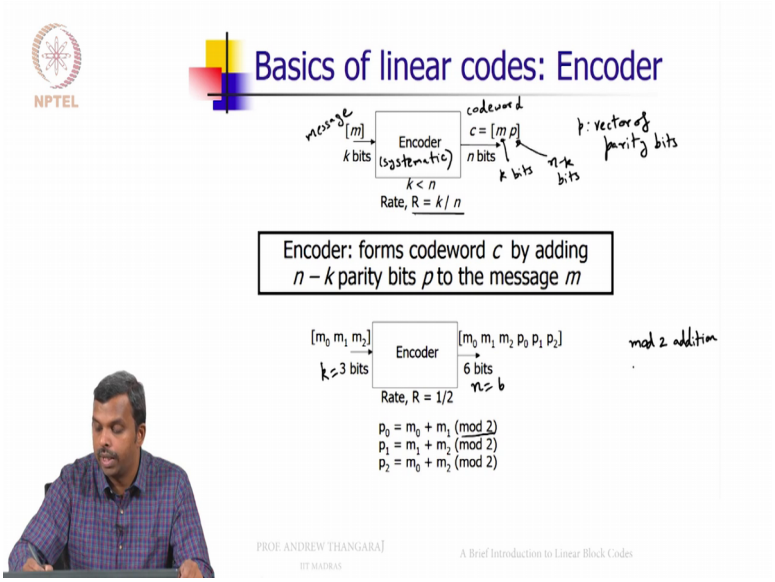
So what is this

(Refer Slide Time 03:21)



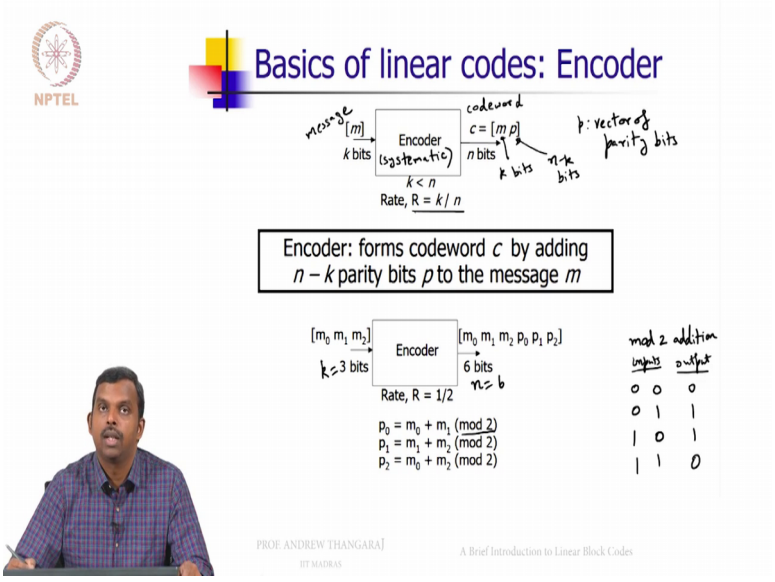
modulo 2 addition? Ok so this is quite

(Refer Slide Time 03:29)



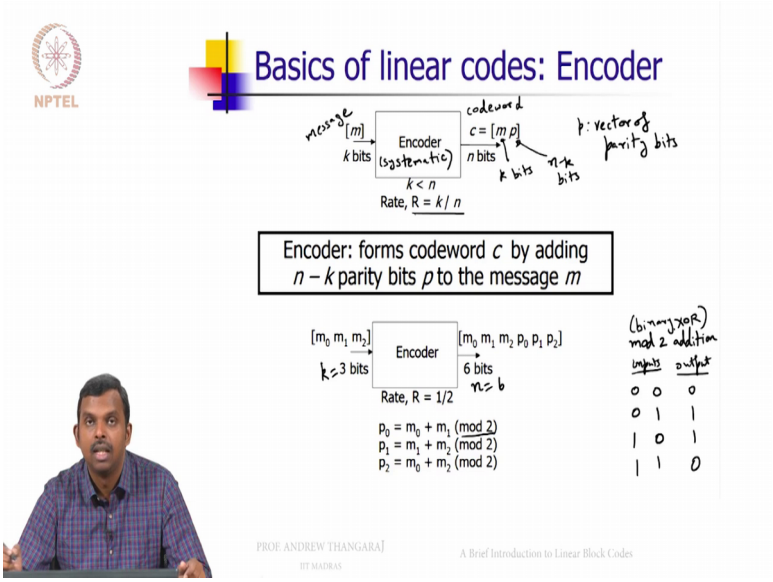
common. You have 2 inputs let us say, 0 and 1, inputs and the output of modulo 2 addition, if you have 0 0 as input, output is 0, 0 1 is input output is 1, 1 0 it is 1, 1 1 is 0.

(Refer Slide Time 03:49)



So it is modulo 2, you divide by 2 and take the reminder. So if you have 1 and 1, 1 plus 1 is 2 but 2 is 0 modulo 2, Ok. So this is also called binary XOR, Ok and this is

(Refer Slide Time 04:05)

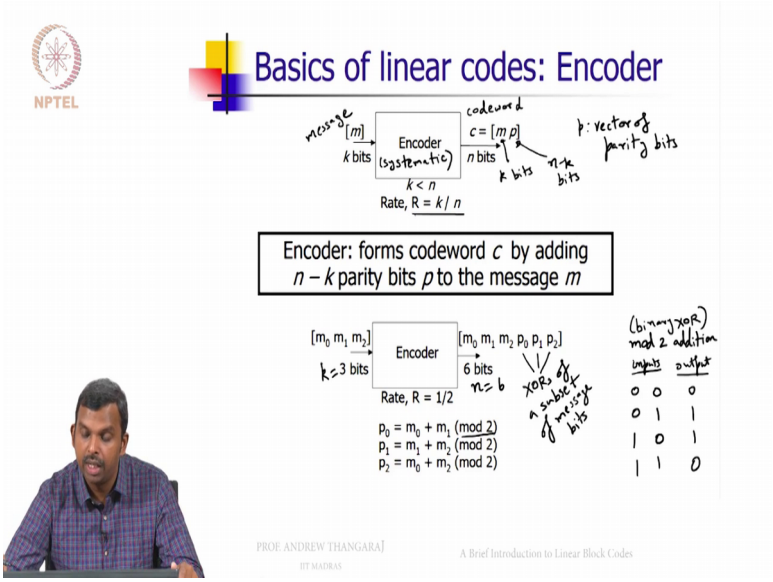


a very important bread and butter operation for us.

You should be comfortable with XOR, given 2 bits you should be able to quickly XOR it. XOR is exclusive OR. The bits are equal means the XOR becomes zero, the bits are not equal, 0 1 or 1 0, the XOR becomes 1, Ok. So that is the operation, Ok.

So that is what we are going to do here. So how are we doing this? So you can see p 0, p 1, p 2 are obtained as XORs or modulo 2 addition of a subset of message bits. So this is a generally true statement for

(Refer Slide Time 04:44)



any linear error control code. You have message bits appearing in the systematic encoder and then each parity bit is obtained as the XOR of subset of message bits, Ok.

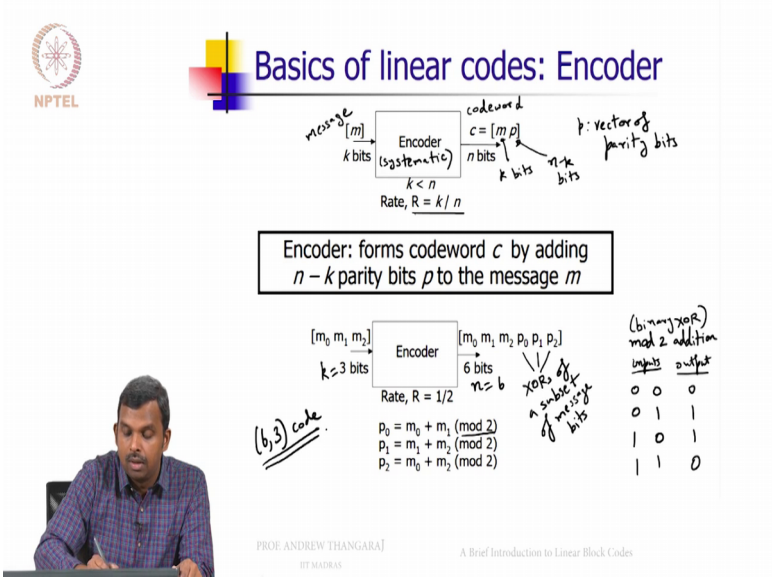
So remember I have just taken two at a time here. But supposing even if you have 3 bits you can make an XOR of all of them. You make the XOR of first two and then you make the XOR of the result with the third one, you can make an XOR of 3 bits as well, or 4 bits or how many ever bits you have, Ok.

So in this case we are just taking two at a time, for instance p 0 is the XOR of m 0 and m 1, p 1 is the XOR of m 1 and m 2 and p 2 is the XOR of m 0 and m 2, Ok. So all linear error control codes, when they have systematic encoding is described in this fashion. For instance even in the 5 G standard the L D P C codes are specified in this fashion. The encoder works like this.

So you have a message vector and then parity bits are computed as XORs of subsets of the message bits. You might have efficient ways of implementing these XORs particularly when k and n are large. But essentially the description is exactly the same, Ok. So this is an example of a linear code. I will use one more notation for linear code.

Such codes I will call them 6 comma 3 code, Ok.

(Refer Slide Time 06:01)



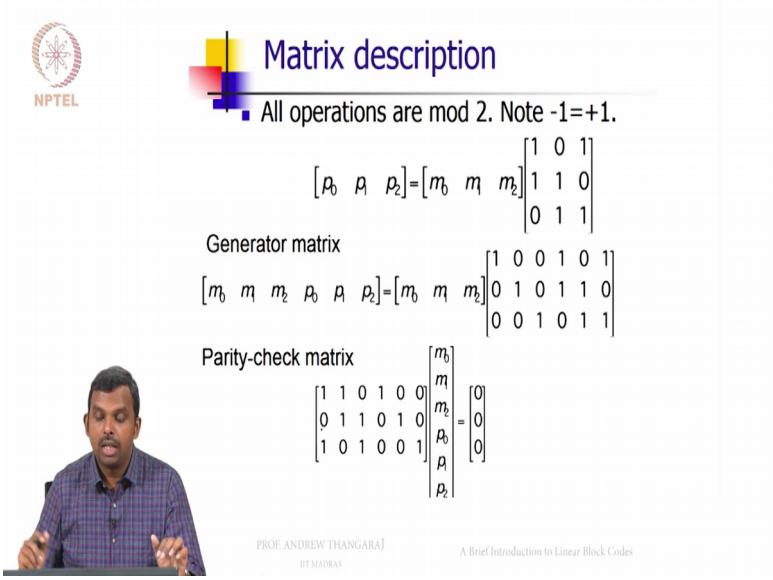
So 6 comma 3 means n is equal to 6, k is equal to 3. But this is not the only 6 comma 3 code. There are some other 6 comma 3 codes but this is reasonable one.

This specific 6 comma 3 code with this kind of an encoder we will use repeatedly in this course as a, as a representative of a generic linear block code. We will talk about decoding, encoding and all that with respect to this code, Ok.

So hopefully the picture is clear to you and also you can imagine even if k becomes very large, even if n becomes very large, k could be like 1000, n could be like 2000 or something like that, this operation is not too difficult to perform, Ok. You just need to know a subset of the message bits, which subset to pick and then XOR them together.

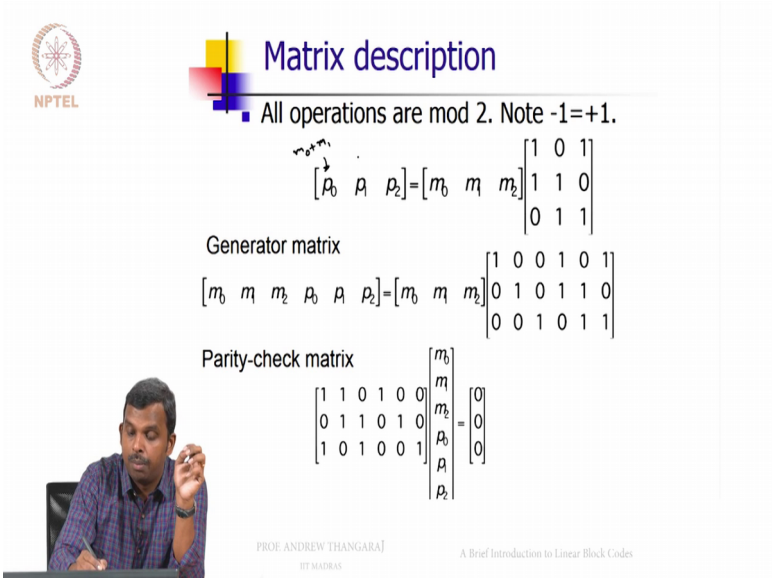
Ok so XORing is a reasonably simple operation to implement. One can do it, Ok so one can implement this encoder quite efficiently in practice. It could work pretty well, Ok. So this is the description of the linear code from an encoding point of view, Ok.

(Refer Slide Time 07:05)



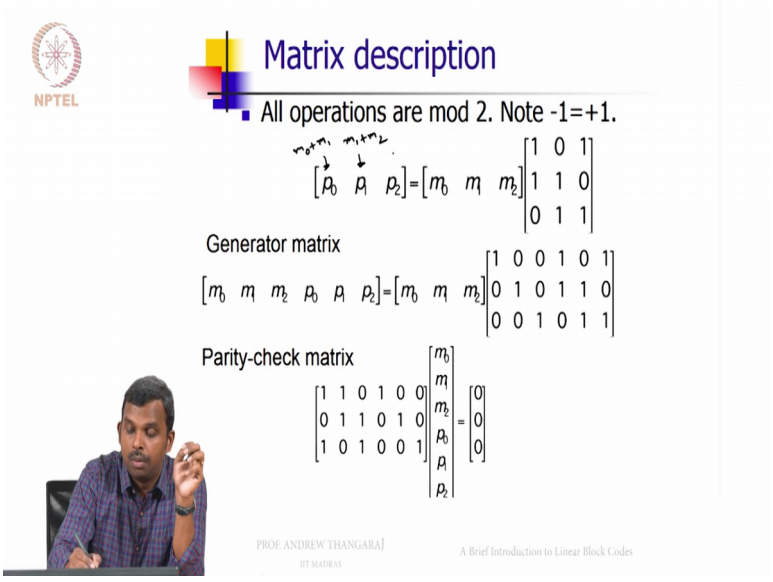
Now it turns out there is an important and useful matrix description for linear block codes and that is described here, Ok. So the three operations we formed to find the 3 parity bits, p 0, p 1 and p 2, p 0 was what? p 0 was m 0 plus m 1.

(Refer Slide Time 07:26)



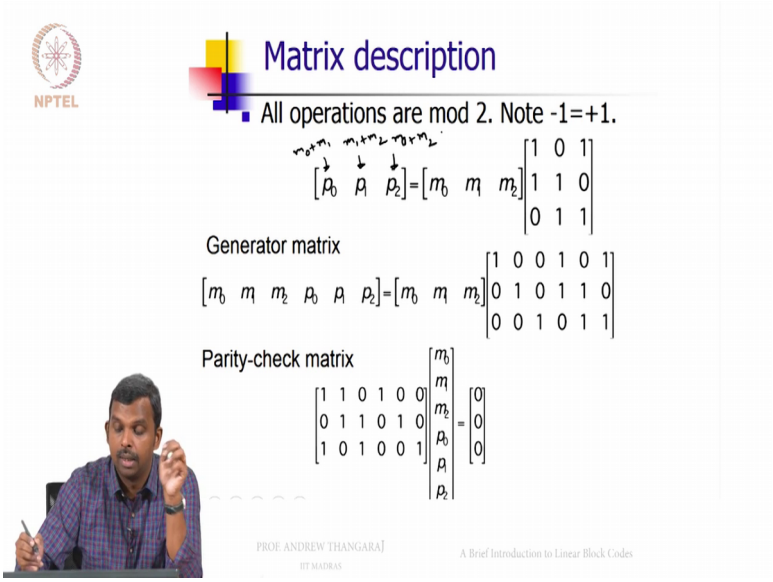
p 1 was m 1 plus m 2.

(Refer Slide Time 07:30)



And p 2 was m 1 plus, I am sorry m 0 plus m 2, right?

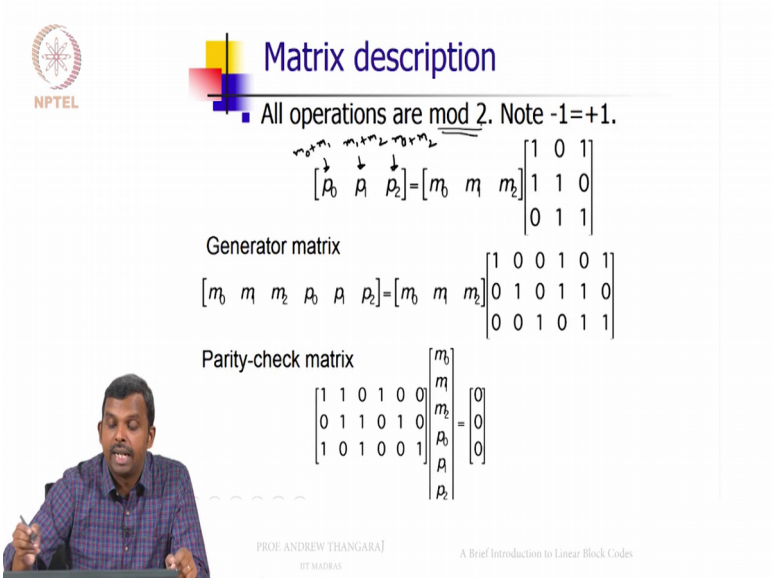
(Refer Slide Time 07:36)



So these three operations can be conveniently represented in this matrix form. So you take m 0, m 1 and m 2, then multiply it on the right with this matrix, 1 1 0, Ok right, m 0 m 1 m 2 multiplied by 1 1 0 will give you m 0 plus m 1. And m 0 m 1 m 2 multiplied by 0 1 1 will give you m 1 plus m 2. And m 0 m 1 m 2 multiplied by 1 0 1 will give you m 0 plus m 2.

Remember all operations are modulo 2, so I won't keep repeating this modulo 2

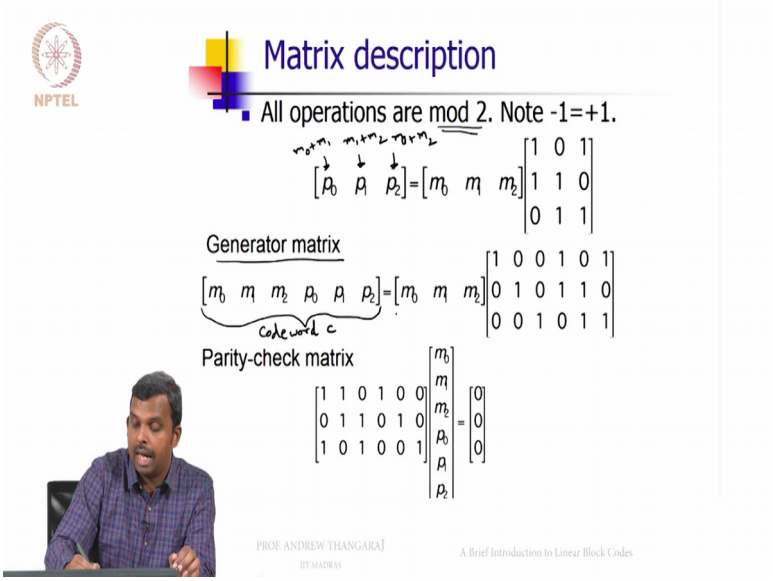
(Refer Slide Time 08:04)



again and again and so you have to assume that whenever I multiply matrices and all that I always do a modulo 2. So 2 becomes 0, so that is something important to know, Ok.

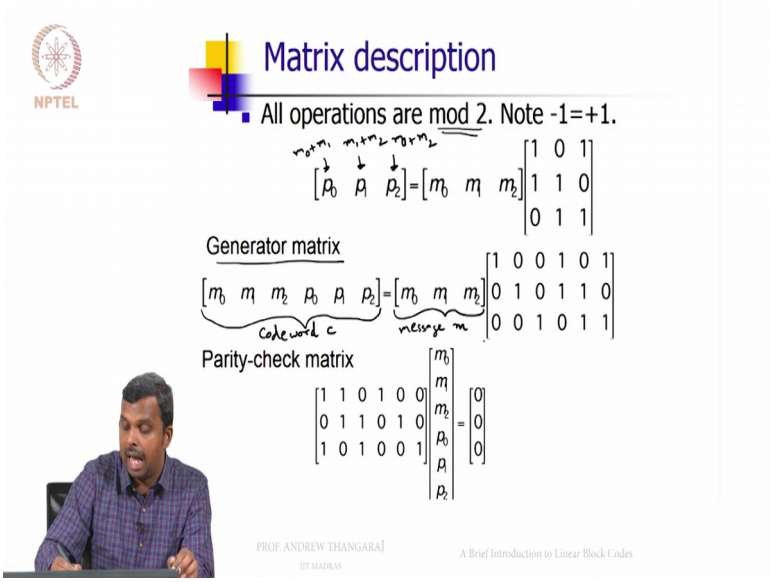
So the first thing one can do is to make a generator matrix Ok which will produce the codeword

(Refer Slide Time 08:30)



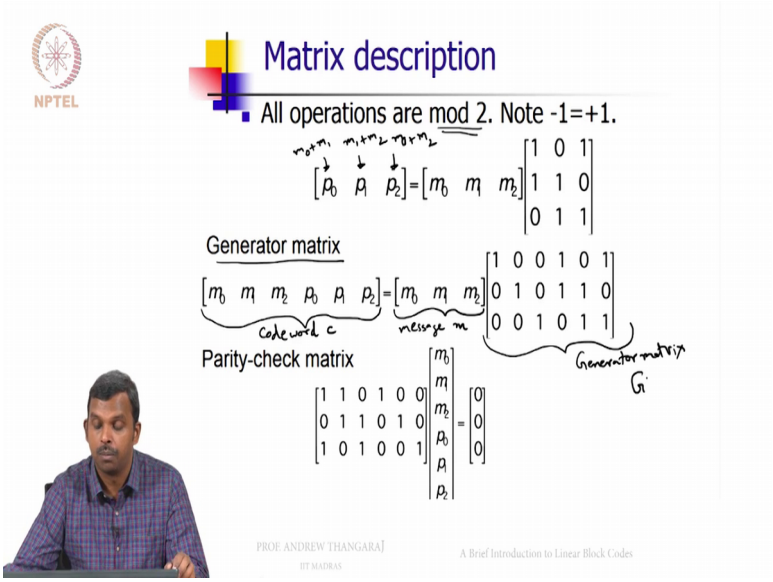
as the product of the message

(Refer Slide Time 08:37)



and this matrix, this matrix is called the generator matrix and denoted G,

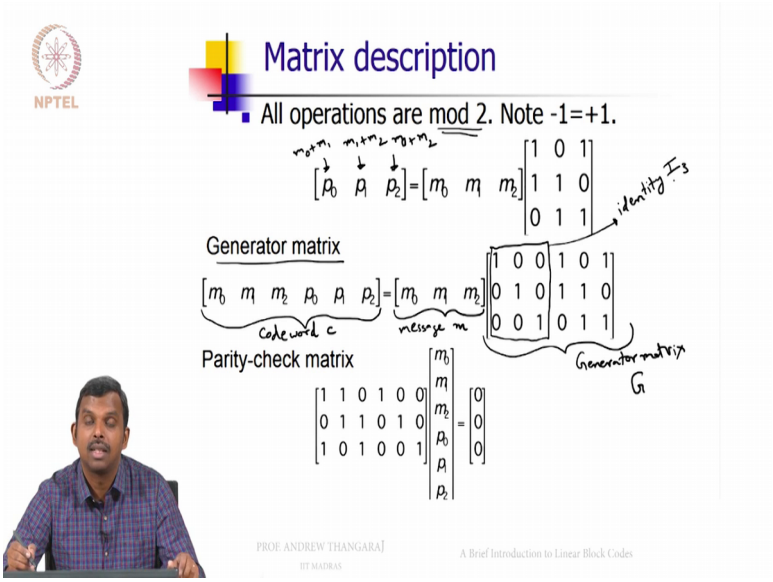
(Refer Slide Time 08:47)



Ok. So you can see where this comes from.

The first 3 bits of the codeword are simply rep ((())) the, just the message bits themselves. So you see you have the identity matrix here. So this part is the identity part, right, identity. I will denote this as I sub 3 to denote that this is a 3 by 3

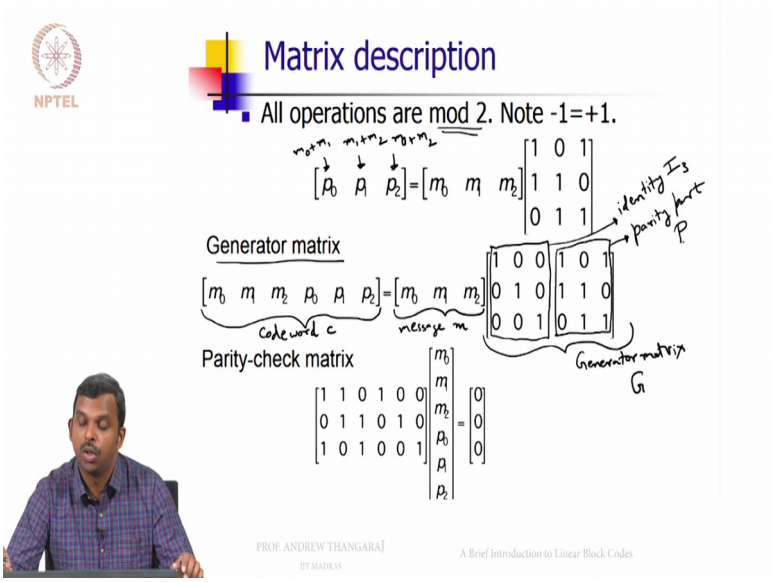
(Refer Slide Time 09:15)



identity matrix.

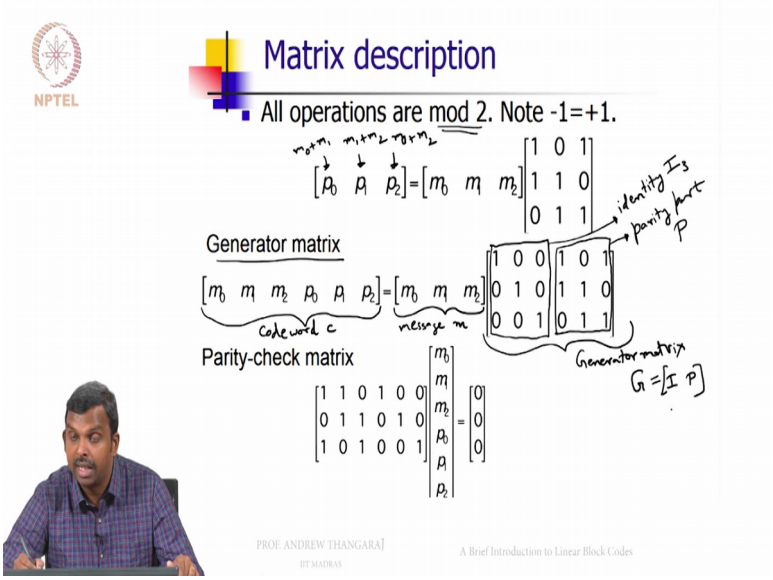
And then I have the parity part, Ok which is the same as what I wrote before. This is the parity part usually denoted P

(Refer Slide Time 09:33)



Ok. So this generator matrix G is I and then P. So this will be a

(Refer Slide Time 09:39)



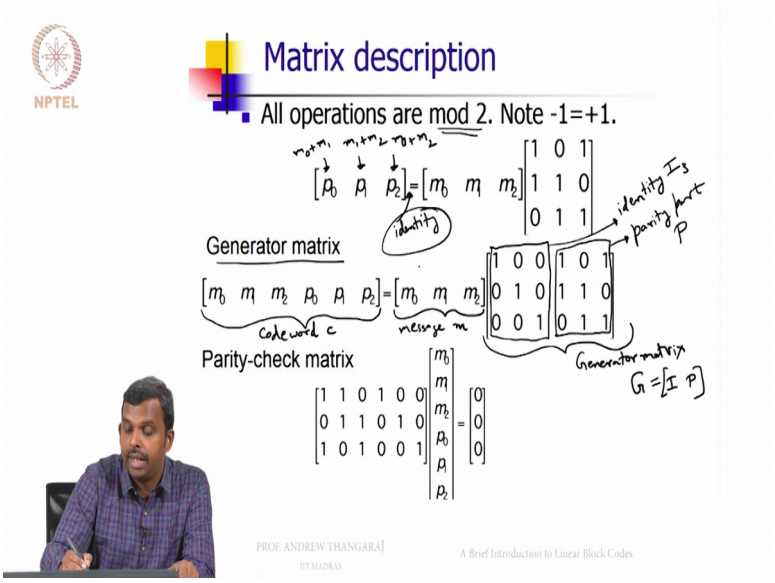
generic structure as we go through as well. So you will have identity part and then a parity part. The parity part is obtained by various XORs.

Ok so you can see clearly that this is the same as the previous operation and one can do it, Ok.

So another way to write this expression p 0 p 1 p 2 is message into this matrix that you have here is to take both parts to one side and equate it to zero, Ok. So this is also equally valid. If you do that, if you take it to this side, what will happen is you will get an identity here, Ok.

So you will get an identity here, Ok. Identity can come here

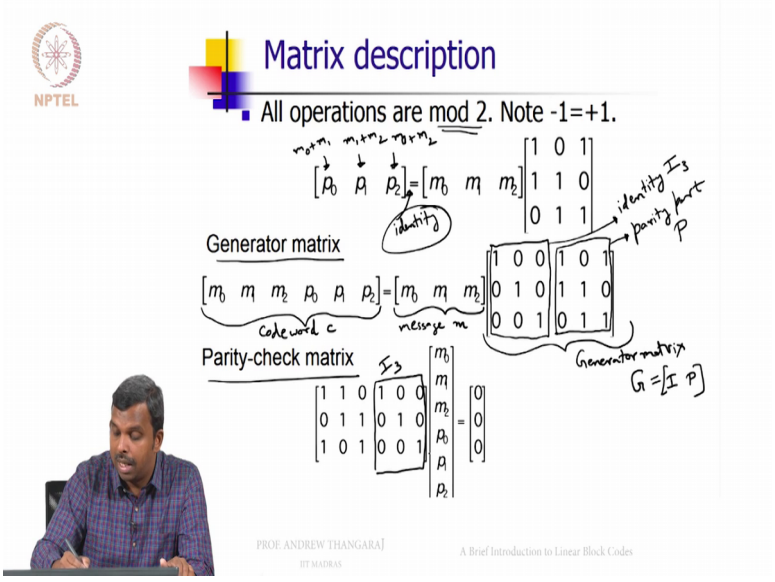
(Refer Slide Time 10:24)



and then you will get this parity check matrix, Ok. So you can see this is very much equivalent to that. I have taken both to one side and I have put identity for the p part and I have got equal to zero zero zero, Ok.

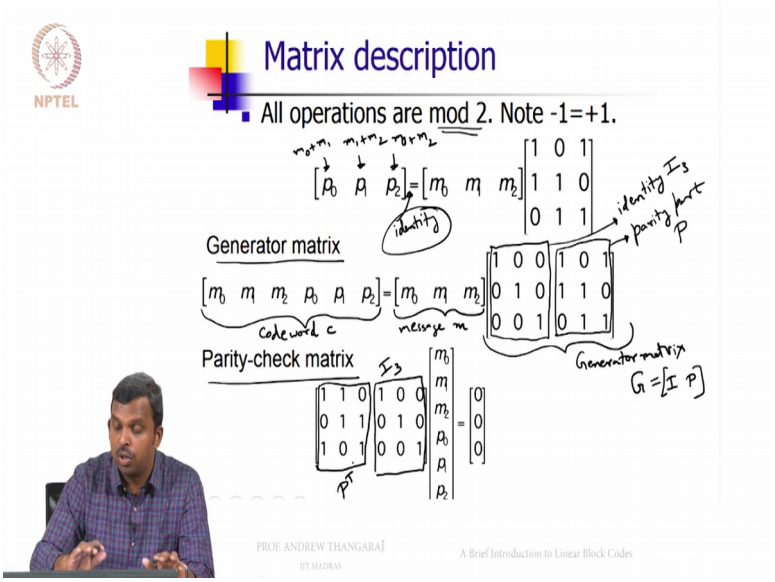
So this is the matrix here, so you have noticed you have the identity here I 3 and that

(Refer Slide Time 10:47)



part multiplies the P, Ok. And then m 0 m 1 m 2, this is the P. So actually if you look at it very closely, this is actually P transpose, Ok.

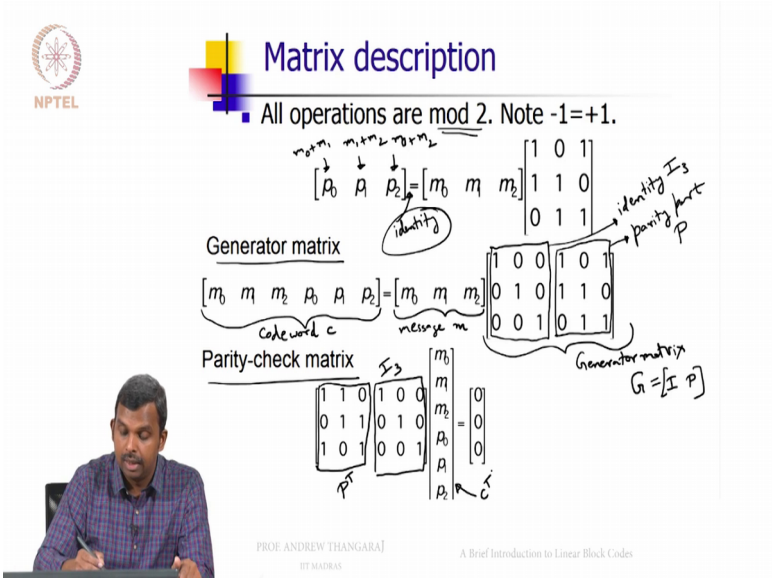
(Refer Slide Time 11:01)



So because I have shifted the left multiplication to right multiplication, Ok so the identity comes on the left, the matrix comes on the left and everything becomes a column, Ok so I have sort of taken transpose on both sides to get columns. And so this becomes P transpose and this is the identity matrix and then together you have this, Ok.

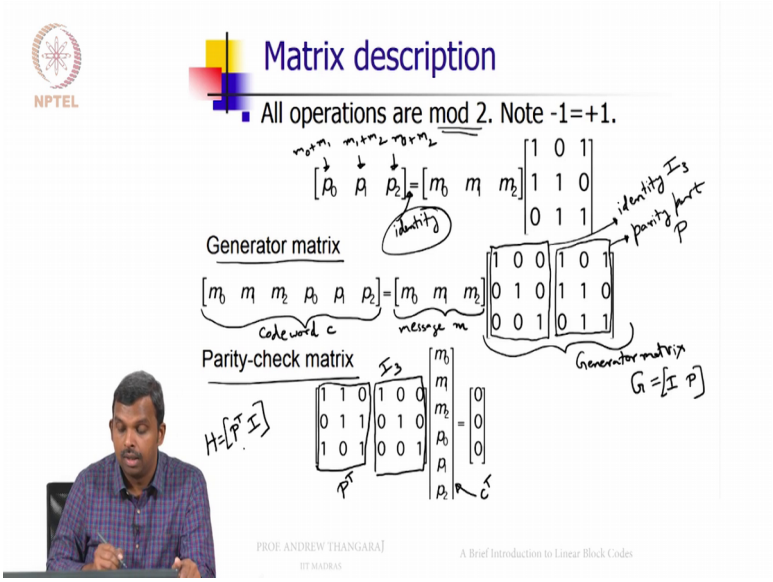
So this once again is c transpose, transpose of the codeword.

(Refer Slide Time 11:28)



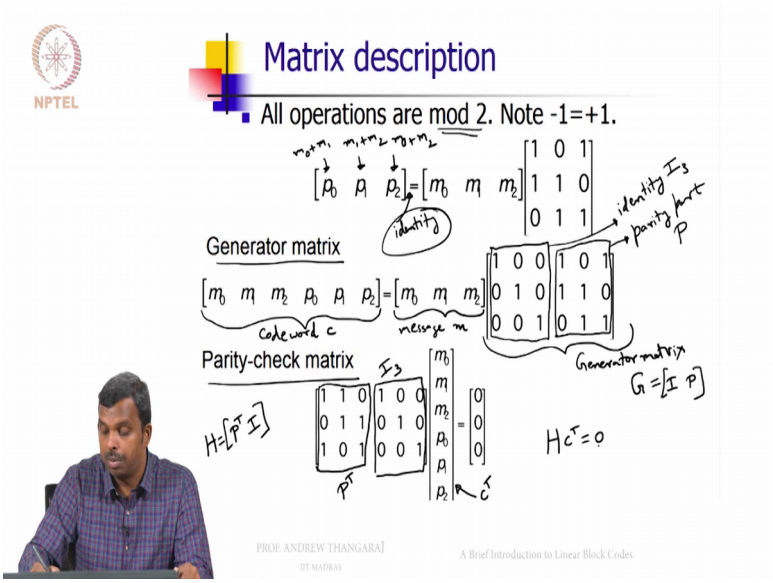
This entire matrix is called the parity check matrix H equals p transpose I. Ok

(Refer Slide Time 11:37)



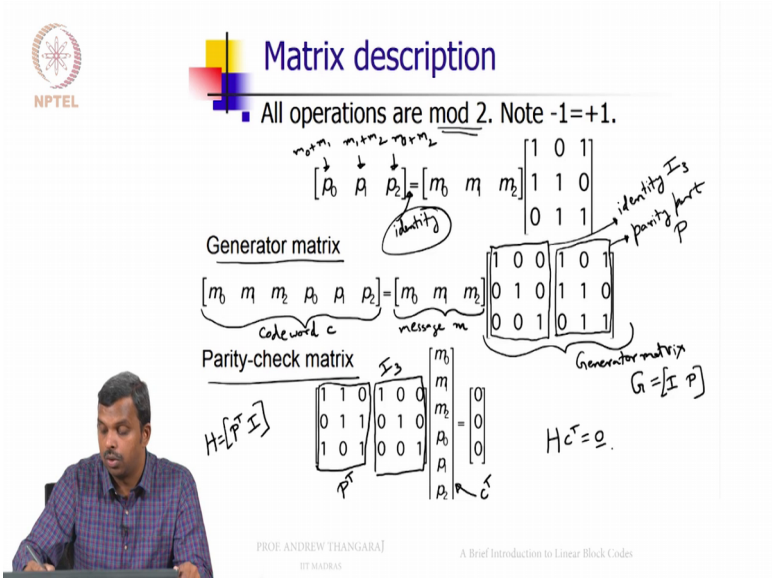
and once you have the parity check matrix what you have is this relationship, H times C transpose equals zero.

(Refer Slide Time 11:45)



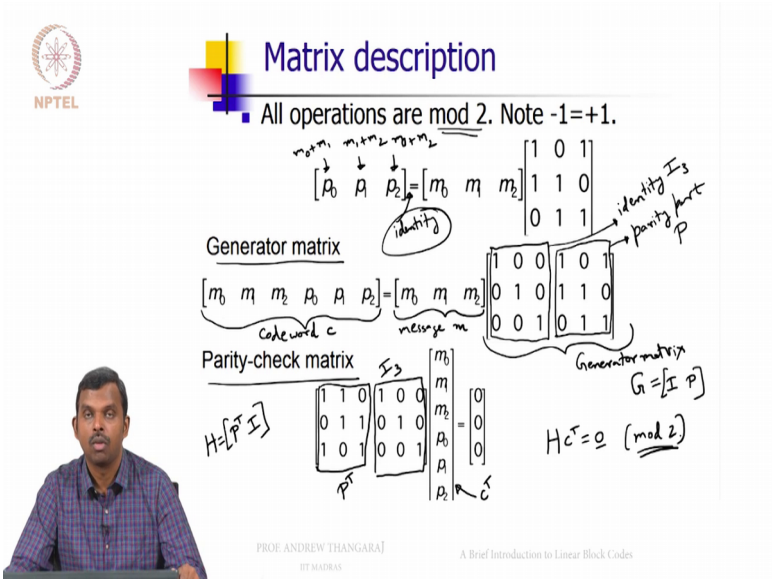
Remember zero is a vector zero here, Ok, so there are as many zeros as you need here. So

(Refer Slide Time 11:51)



I put an underscore to, I mean underline it to denote it is a vector and this is, remember also everything is modulo 2, Ok.

(Refer Slide Time 12:01)



So these 2 matrices play a very, very important role in describing codes, Ok. So typically codes are described using generator matrices and parity check matrices. And they may be more generic than having just this I P sort of structure. May be they are not systematic. May be they have some other sort of form. And we will use both these things when describing code.

So it turns out the polar codes are usually described with the generator matrix and the L D P C codes, the low density parity check codes of course are described using the parity check

matrix. So both of these are very good descriptions of error control codes. You can see clearly that they are equivalent.

If they all come from the same idea that you have parities being formed as XORs of subsets of the message bits and that gives you this complete picture of having a parity check matrix or a generator matrix, Ok.

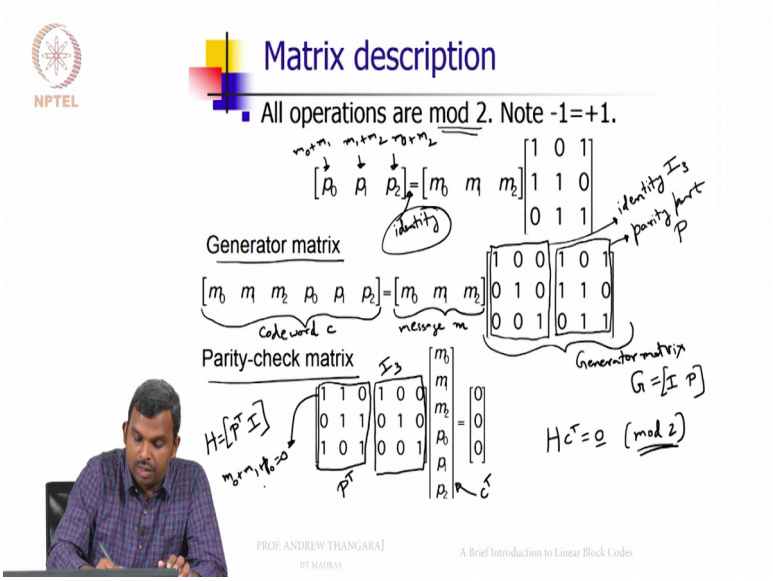
So the code is fully described with the parity check matrix. Hopefully you are convinced of that. See remember m 0 m 1 m 2 are the messages. So once they give the m 0, m 1 and m 2 how do you find p 0?

You have to find p 0 so that the first product, product with the first row equals zero, Ok. So if we take product with the first row, you get m 0 plus m 1 plus p 0 equals zero. Ok. So that gives you p 0 as m 0 plus m 1, Ok?

So remember this is all modulo 2. So minus 1 is same as plus 1, so m 0 plus m 1 plus p 1 equals zero is the same as, I am sorry, m 0 plus m 1 plus p 0 equals zero is the same as p 0 equals m 0 plus m 1.

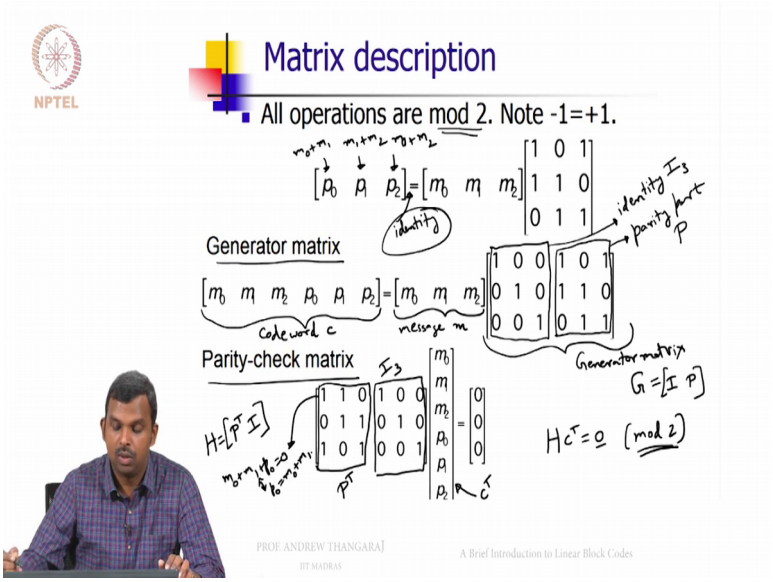
So maybe I should write that down for you. So if you look at the first row, the first row says m 0 plus m 1 plus p 0 equals zero which is

(Refer Slide Time 13:49)



the same as p 0 equals m 0 plus m 1.

(Refer Slide Time 13:56)

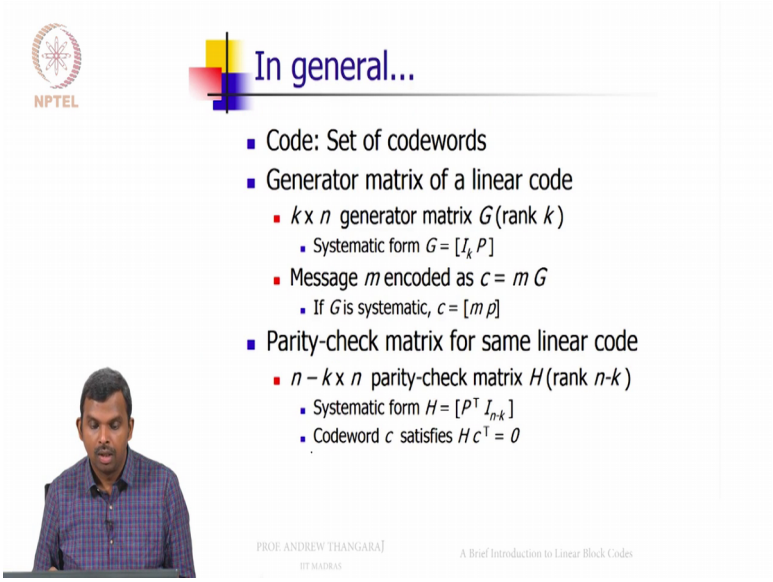


So hopefully you see how the whole thing is working out, every other parity bit is also conditioned in the same way, Ok.

So you can use the parity check matrix to perform encoding as well. You take the first row. That gives you a parity bit p 0; the second row gives you the parity bit p 1. The third row gives you the parity bit p 2 and so on. So this description is very, very important and in fact all linear codes are described in this way.

Even the 7 4 Hamming code has a description like this. I will show you that soon enough Ok. So this is generator and parity check matrix. It is quite important.

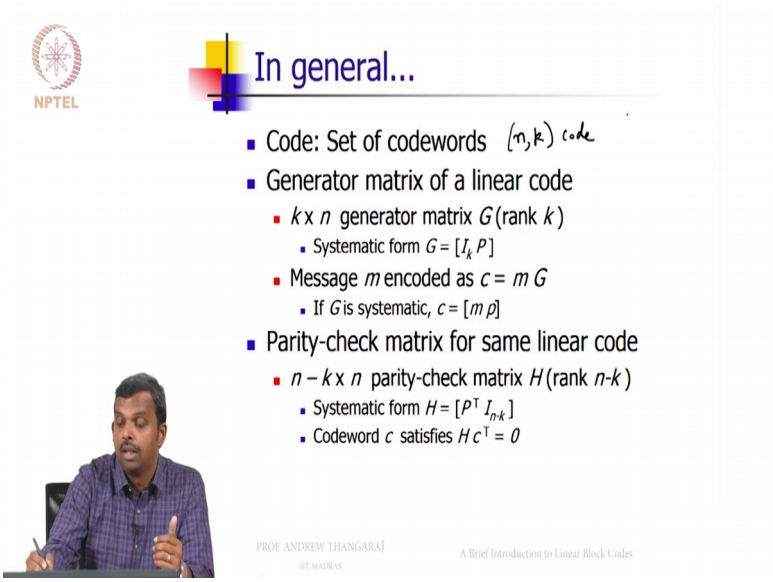
(Refer Slide Time 14:31)



Ok so let us generalize from what we had before. A code is technically defined as the set of all codewords. You take all the codewords together, you have a, you put them together in a set, Ok how many of codewords you have, that makes a code.

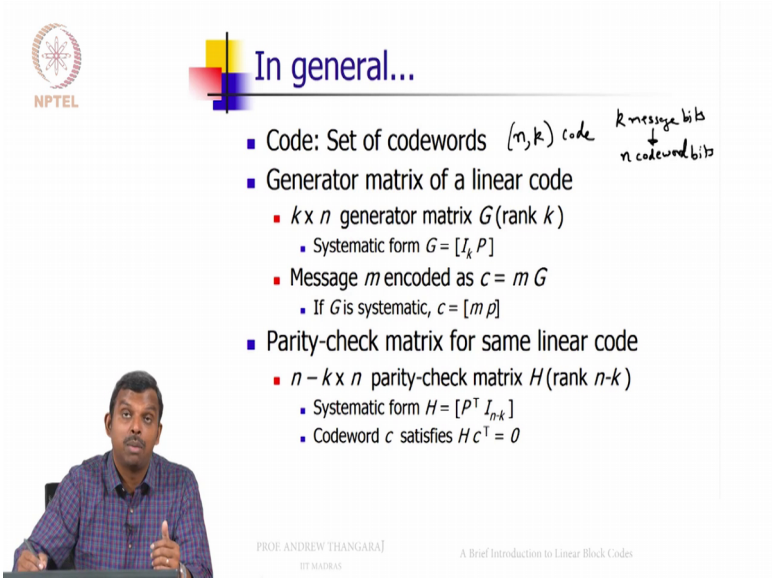
And typically one thinks of a n comma k code, Ok

(Refer Slide Time 14:52)



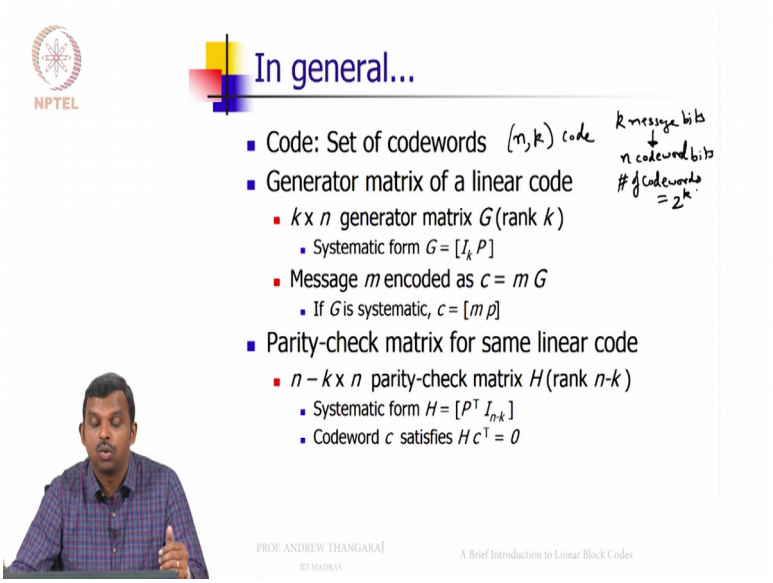
so this is k message bits to n codeword bits. That means

(Refer Slide Time 15:03)



you have n minus k parity bits. Number of codewords equals 2 power k,

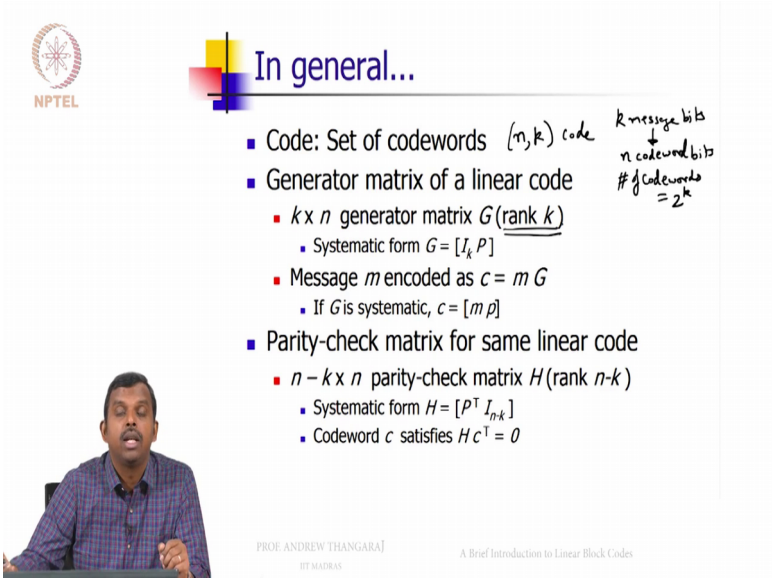
(Refer Slide Time 15:14)



Ok. So you have 2 power k codewords typically in a n k code and all of them make up the code, Ok.

So you have a generator matrix for the linear code which is a k cross n generator matrix of, we usually denote G, it should have rank k. It is the linear algebraic property

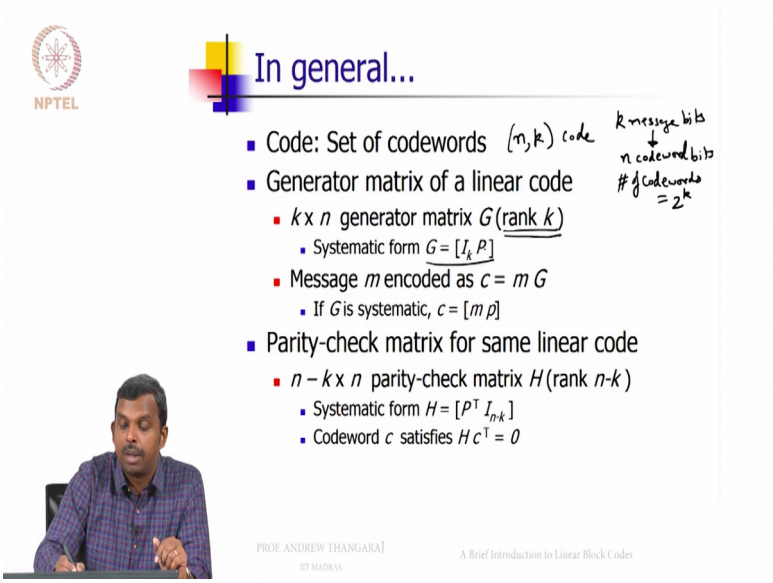
(Refer Slide Time 15:30)



over the vector space we are considering here, we do not have to talk too much about it in this class at least.

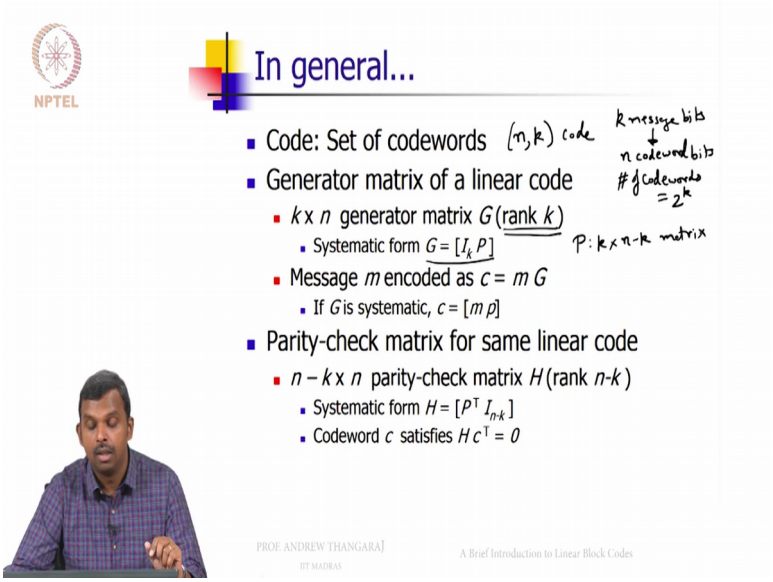
So in systematic form G can be written as I sub k, the identity part and

(Refer Slide Time 15:44)



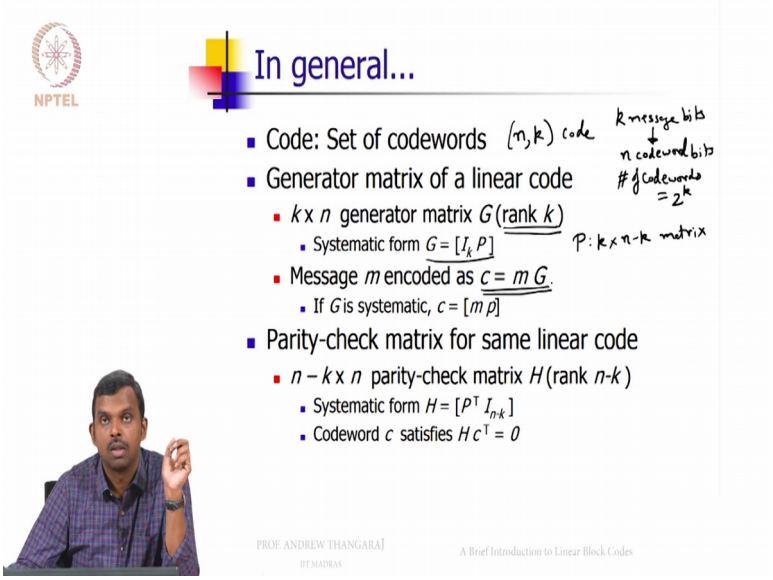
the parity part. Ok so the P is a k by n minus k matrix, Ok. So

(Refer Slide Time 15:52)



when you multiply m with G you get a codeword, Ok.

(Refer Slide Time 15:58)



So when you multiply m on the left side, m will multiply with the identity part to give m itself.

Then m will multiply with the capital P matrix to give you the parity part P, Ok. So this is exactly what we had before. One can generalize.

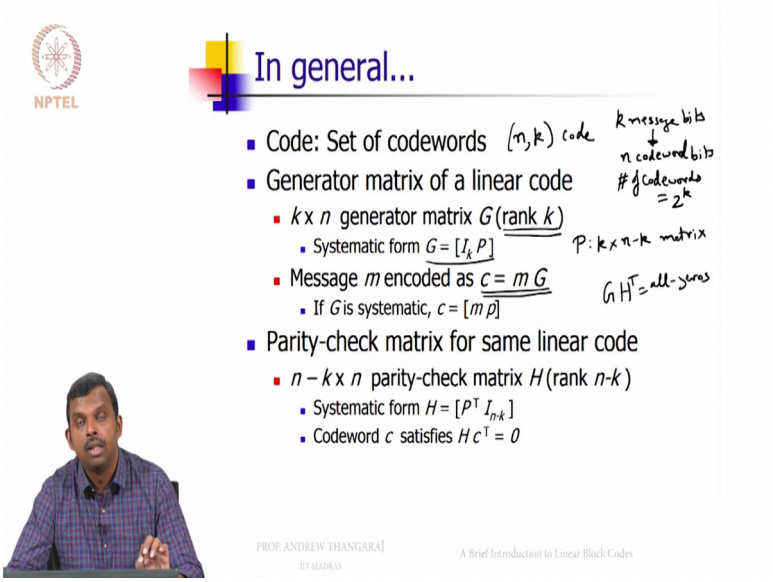
But now there will be some codes where G will not have this form, will not have I k P form, it will not be in systematic form but nevertheless that is a valid description of the code. It turns out; you can go from non-systematic forms to systematic form and all that.

But it is not, maybe it is not so important in this class but nevertheless you should know that G can have a more general form. In fact when we talk about polar codes I will describe them using the generator matrix and when I specify the generator matrix for the polar code you will see it is not in systematic form not in the I k P form, Ok.

But nevertheless you can always form the codeword as m times G, Ok. It won't have, m may be appearing by itself in the codeword but it will produce some other m-bit vector which you can transmit as the codeword, Ok.

So the parity check matrix for the same code is an n minus k cross n matrix usually denoted H. It needs to have rank n minus k and there is one important condition, G times H transpose has to be equal to all zero matrix, Ok.

(Refer Slide Time 17:17)



So every row of G should, if you take a dot product or if you multiply with every row of H you should get zero, Ok.

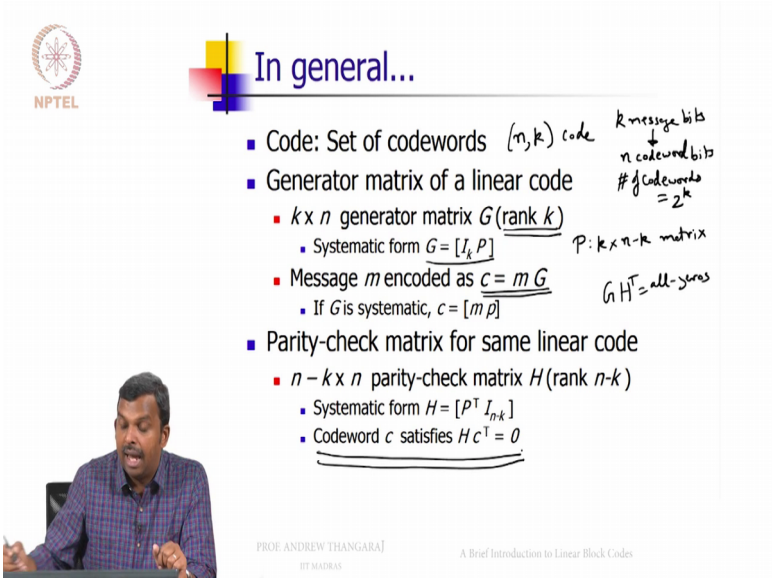
So that is the condition of the parity check matrix. It is said to describe the dual code and all that. In general in this class we will not talk too much about the linear algebraic properties of codes and linear codes. It is not so important to us but nevertheless it is good to do this and read it, read about it.

I will be putting up some additional lectures from prior courses which precisely describe these linear algebraic properties. If you are interested please go through them. But it is sort of additional reading as far as this class is concerned, Ok.

So if you have the generator matrix in systematic form I k P it turns out, a parity check matrix is really specified. You get H to be P transpose I n minus k. So it is a little bit of an exercise to check that G specified as I k P and H specified as P transpose I n minus k, if you take a transpose product of them you will get zero. It is something you can check.

And the most important property is this, Ok. Once you have the parity check matrix

(Refer Slide Time 18:23)



every code word satisfies H C transpose equals zero. So just like the generator matrix is useful in encoding, typically one uses the parity check matrix for decoding. You can also it for encoding. It is sort as the same as the generator matrix but typically one uses it for encoding.

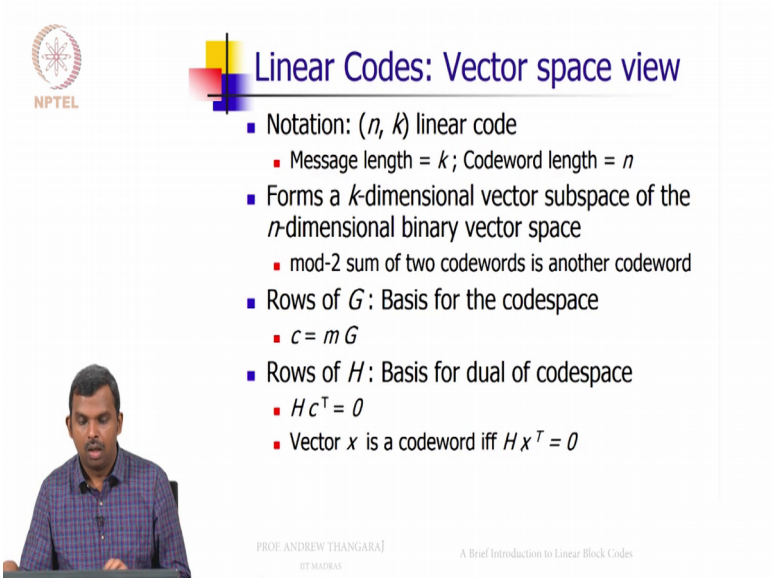
So for instance if I give an n-bit vector, some n-bit vector and ask you whether or not it is a valid codeword, Ok or does it belong to the list of all codewords that you have in the code? Does it belong to the code, if I ask you that question it is easiest to answer if you have the parity check matrix?

Ok, so what can you do, you can take the parity check matrix, multiply that vector transpose on the right and see if you get zero. If you get zero then it belongs to the code. If you do not get zero, it does not belong to the code, Ok.

So you can see that given a vector you can do this and you can use it in decoding also and it is quite efficient. On the other hand with the generator matrix may be you cannot answer that question immediately and quickly, Ok.

So this is a description and when we describe polar codes and L D P C codes I will describe them using generator and parity check matrices, Ok.

(Refer Slide Time 19:39)

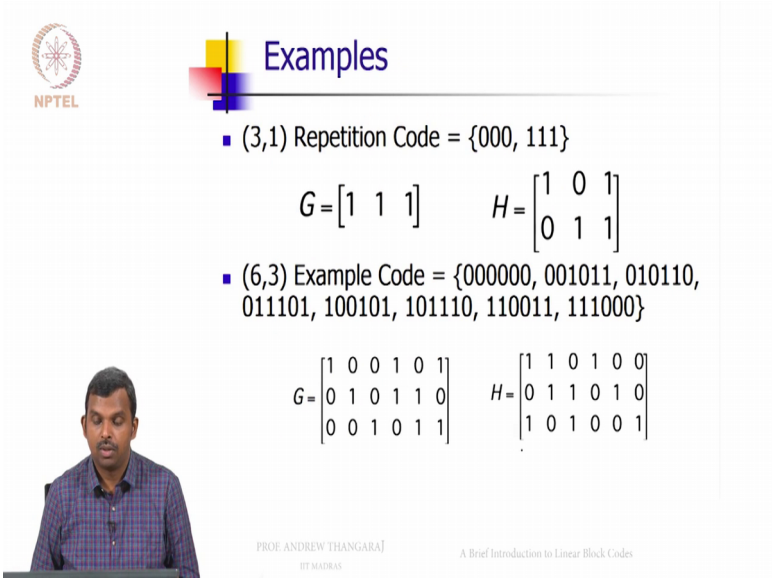


So like I mentioned there is a vector space view. I will just mention it briefly. There are other lectures that I will upload which will talk about this in more detail.

This is the crux of the story. It turns out a n k linear code form a k-dimensional vector subspace of the n dimensional binary vector space. Basically that means modulo 2 sum of two codewords is another codeword, Ok. So that is quite easy to see in the way described it.

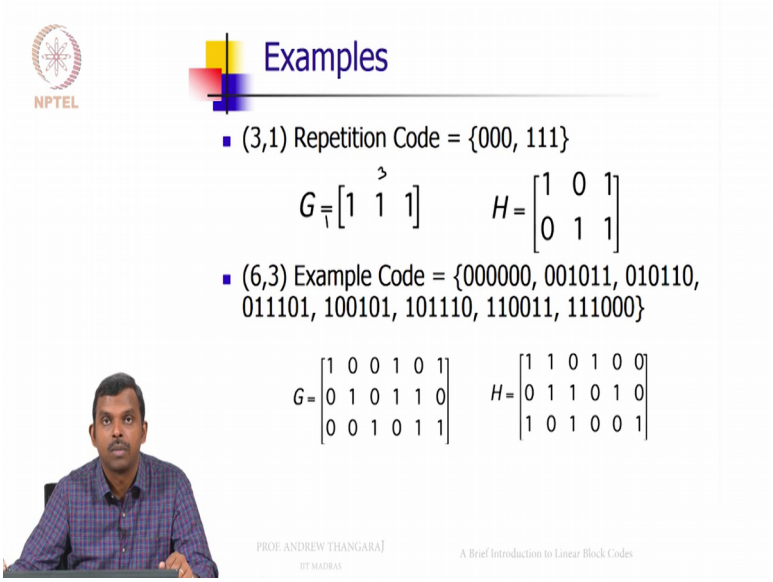
The rows of G are basis for the code space, the basis for the subspace which is the code space and the rows of H form the basis for the dual of the code space, Ok. So that is the whole story from a vector space point of view.

(Refer Slide Time 20:19)



So here are the couple of examples. I will provide one more example but this is the two simplest examples, one to two examples we have seen before in this class, Ok. The first one is the 3 comma 1 repetition code. The code itself has just 2 codewords, 0 0 0 and 1 1 1, Ok. The generator matrix is just 1 row, 1 1 1. Ok so you can see it is a 1 cross 3 matrix

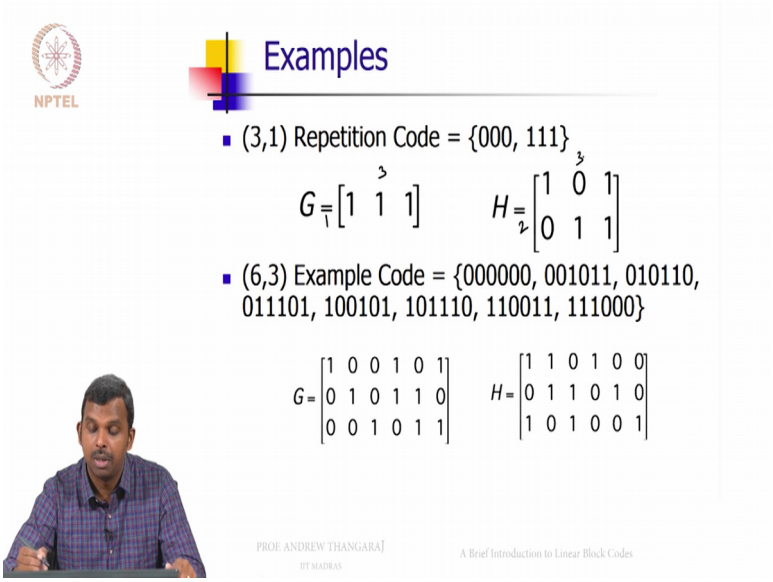
(Refer Slide Time 20:47)



Ok.

And if you multiply by 0 on the left you get 0 0 0, that is one codeword. You multiply by 1, you get 1 1 1, that is another codeword, Ok. Now the parity check matrix is a 2 cross 3 matrix.

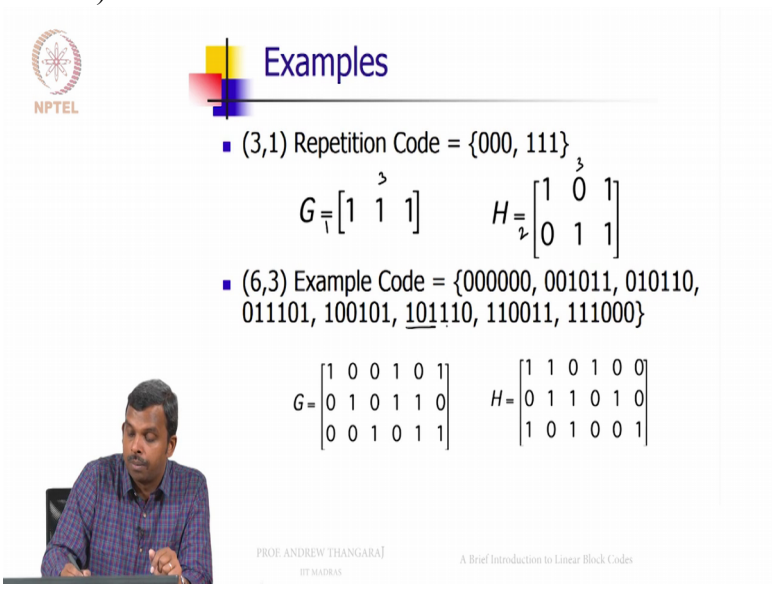
(Refer Slide Time 21:00)



Remember it is n minus k by n and you can check that this is the valid parity check matrix. So you multiply with G trans/transpose, G with H transpose you will get zero and that is the parity check matrix, Ok.

The next 6 comma 3 example code that we saw has this list of codewords. You can see the message part comes here for instance. You look at the message part here. This is the m part, Ok.

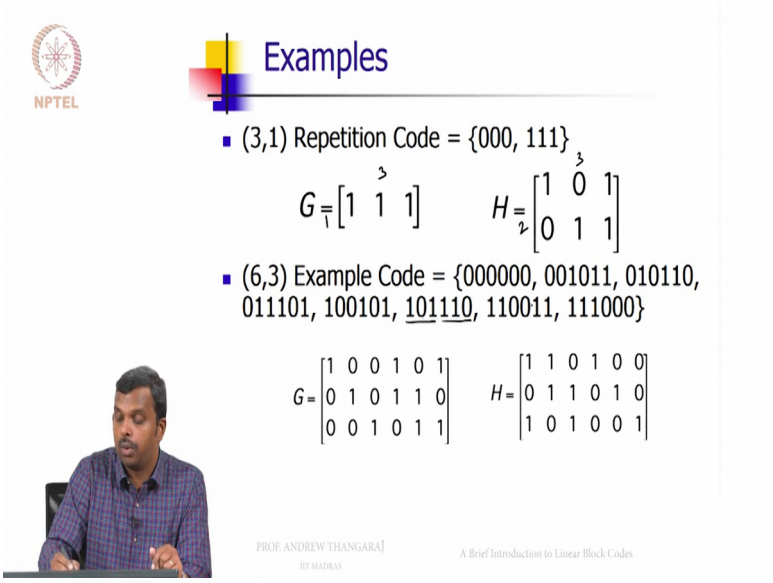
(Refer Slide Time 21:23)



And then you have the p 0. p 0 being XOR of 1 and 0, p 1 being XOR of 1 and 2 and then p 2 being the XOR of 1 and 3, Ok.

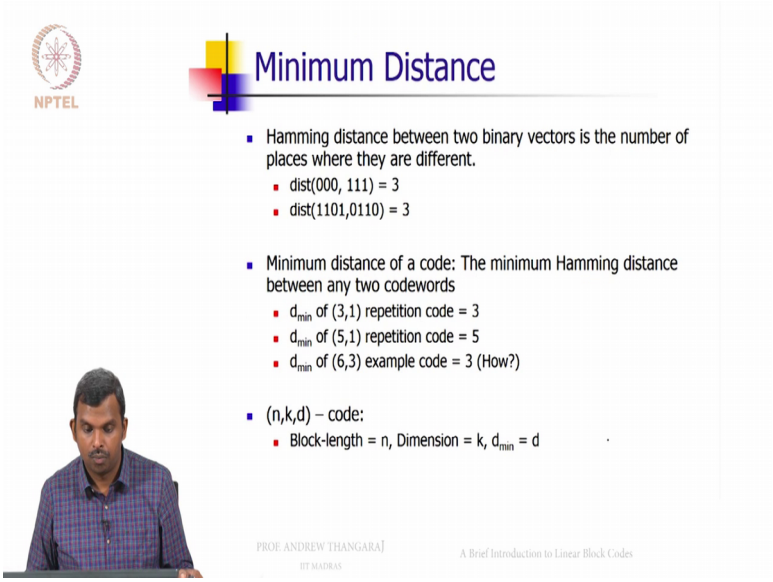
So that is a codeword.

(Refer Slide Time 21:35)



Likewise all the 8 different codewords have been listed out here and generator matrix is given here, the parity check matrix is given here, Ok so this how one can see examples quickly,

(Refer Slide Time 21:49)



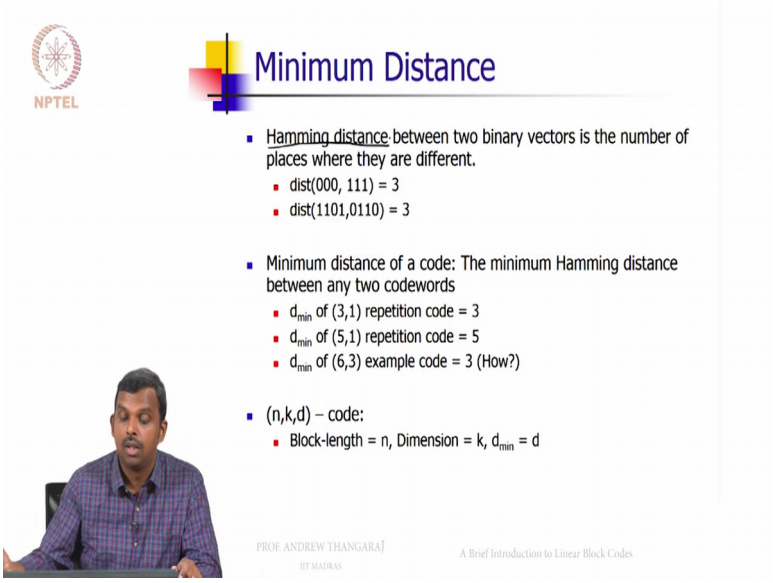
Ok.

So, Ok so the last idea that we will briefly introduce in this lecture is this of minimum distance, Ok. It is, it is not something that is very critical for us in this course. But it is an important design principle. It is very, very useful to know and have some intuition about the minimum distance and the role it plays in encoding, in decoding successfully. Ok.

So this is extremely important. Lot of people who design codes have intuition about minimum distance. It plays a role but as it turns out in modern codes the role is little bit, it is sort of used as a useful design criterion but people do not really try to optimize this as much as they used to do before. But nevertheless it is important to know the definition, Ok.

So the first definition you need to know to understand minimum distance is the definition of Hamming distance.

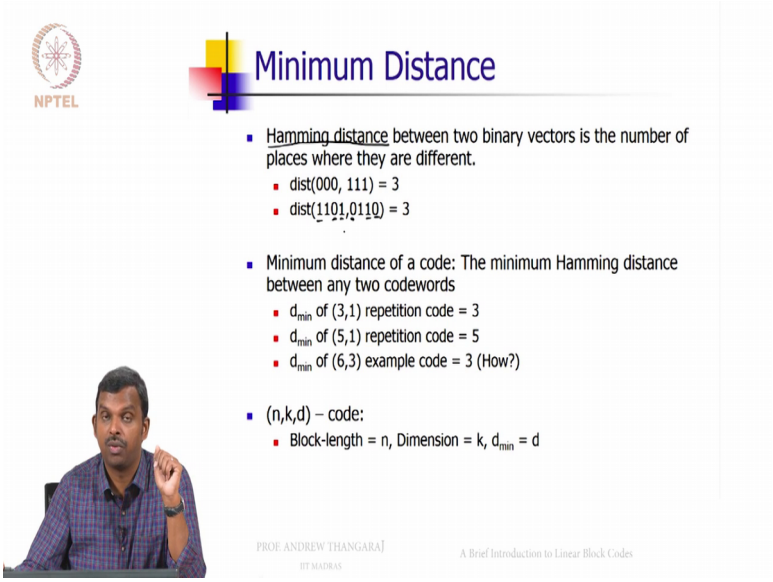
(Refer Slide Time 22:50)



So what is Hamming distance? If we take two different binary vectors of the same length, Hamming distance is the number of places where they are different, where they differ, Ok.

So here is a couple of examples. You can see 0 0 0 and 1 1 1 clearly differ in 3 places. This one may be is a little bit more difficult to write down but again the difference is in 3 places, Ok, in all these 3 positions they differ. So the Hamming, minimum, the Hamming distance between

(Refer Slide Time 23:16)

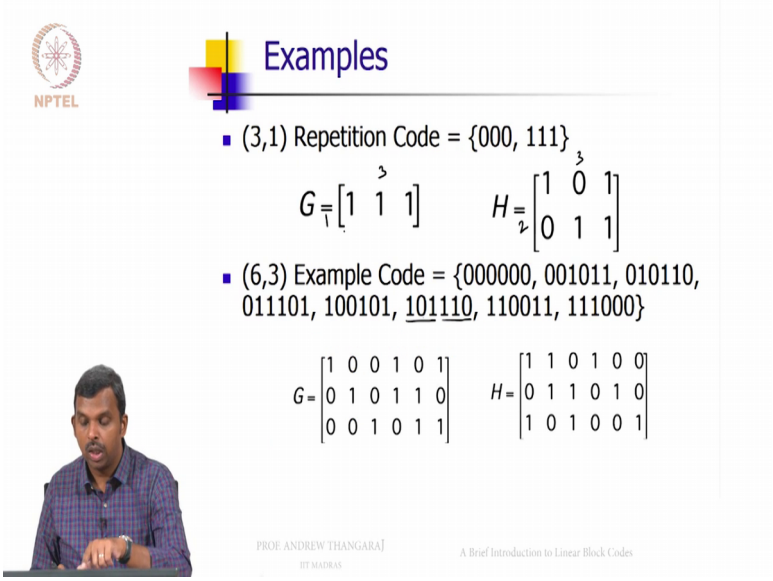


these vectors is 3, Ok.

Now what is the minimum distance of a code? You have lot of codewords in the code, Ok so you have 2 power k codewords. You take 2 at a time and find all possible distances, Ok and the least among those distances is the minimum distance of the code.

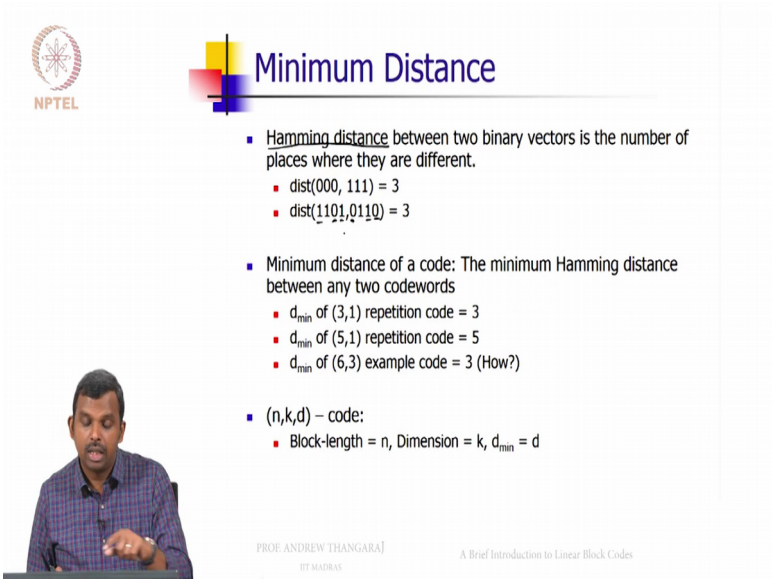
So one can, means this is not a very easy thing to calculate given a code but for some codes one can easily do it. For instance with the 3 comma 1 repetition code the minimum distance is clearly 3 because there are only 2 codewords 0 0 0 and 1 1 1 but what about the 6 comma 3 example code?

(Refer Slide Time 23:52)



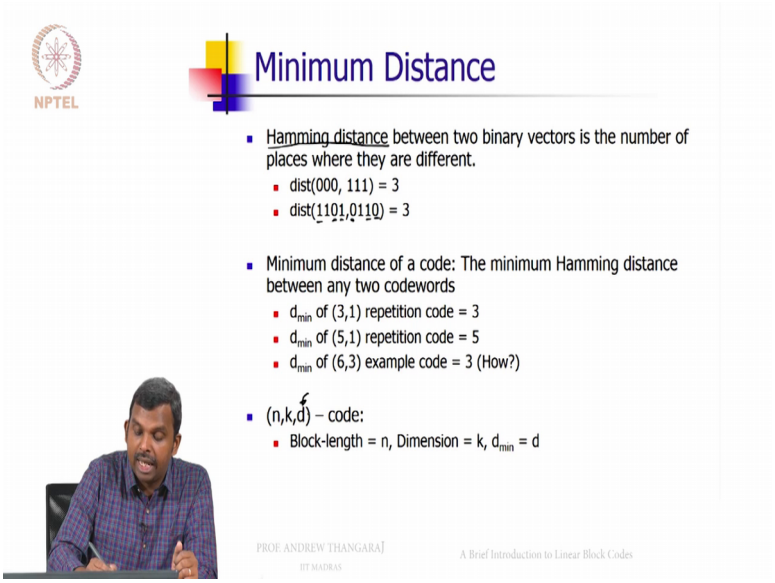
You look at this list of codewords here. What is the minimum distance? It is not immediate, right? So you have to go through all possible pairs, Ok. There are lot of pairs here. You can look at distances between any two of them and then find the minimum distance of that code.

(Refer Slide Time 24:08)



It turns out the answer is 3 for that code. Ok it is not very immediate, Ok but nevertheless this minimum distance is an important parameter. So typically people specify that along with

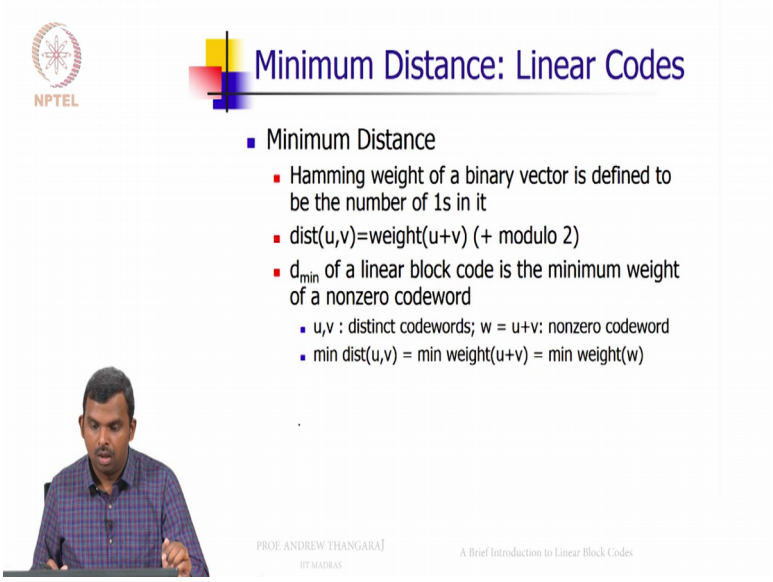
(Refer Slide Time 24:19)



the n k they will introduce that as a new parameter d, they will say it is a n k d code, block length is n, dimension is k or message length is k and then the minimum distance equals d, Ok.

But like I said in this course which is focused on L D P C Polar codes, the minimum distance won't make an explicit appearance. I will allude to it may be when I can later on, Ok,

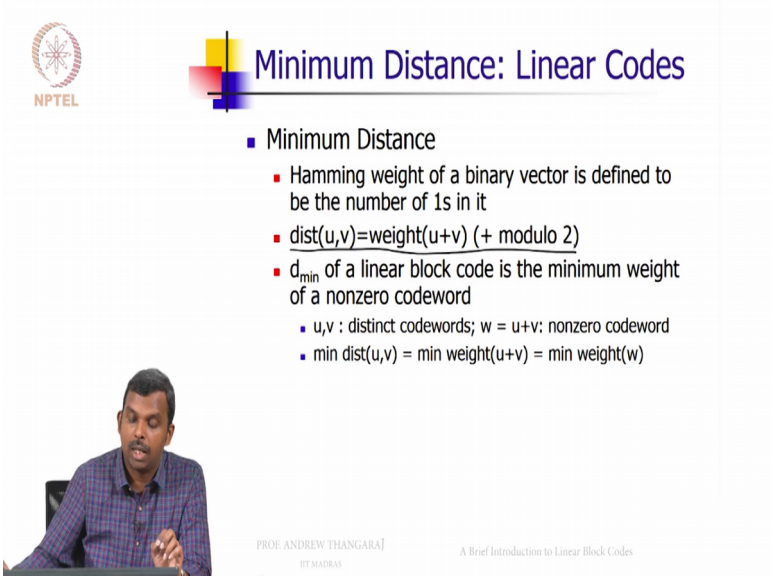
(Refer Slide Time 24:43)



alright. So it turns out for linear codes, because the sum of 2 codewords is also a codeword one can simplify some of the computation involved in the minimum distance.

So for instance this is an important identity, Ok.

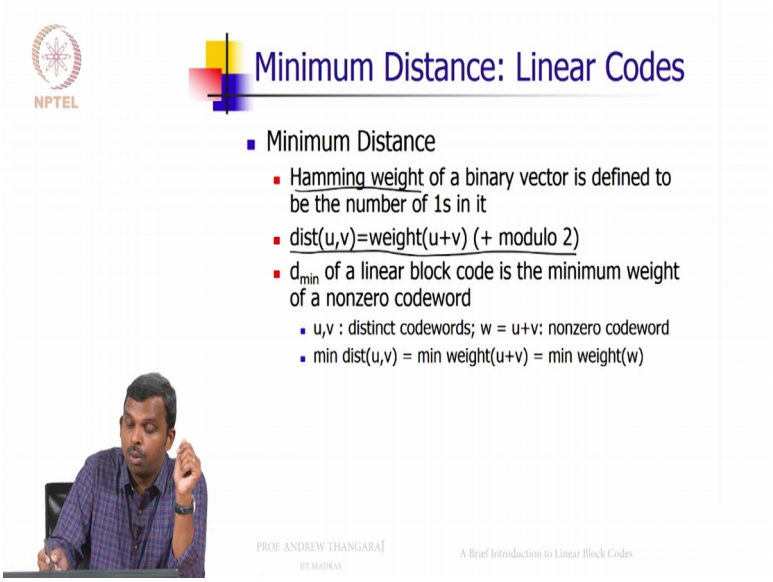
(Refer Slide Time 24:59)



So if you want to find the number of places where 2 vectors differ, you can take the XOR of two vectors, Ok and simply count the number of 1s in the XOR, Ok.

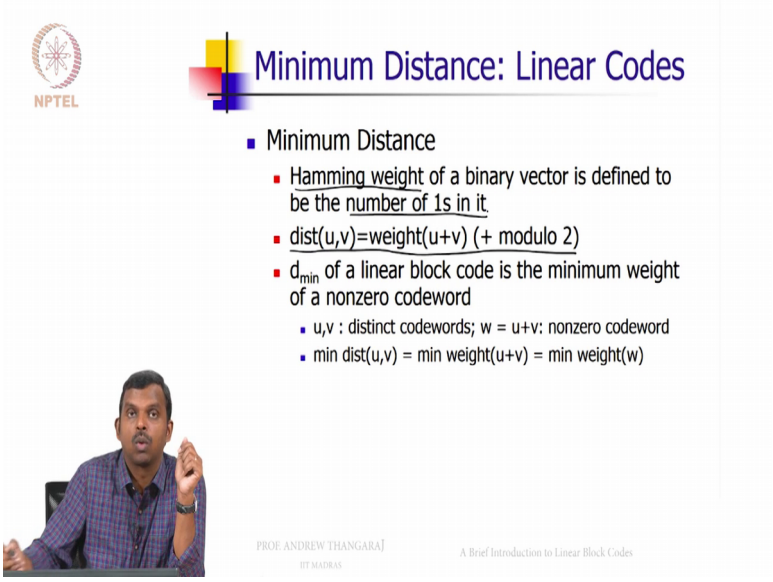
So remember Hamming weight is defined

(Refer Slide Time 25:12)



as the number of 1s in the vector,

(Refer Slide Time 25:14)



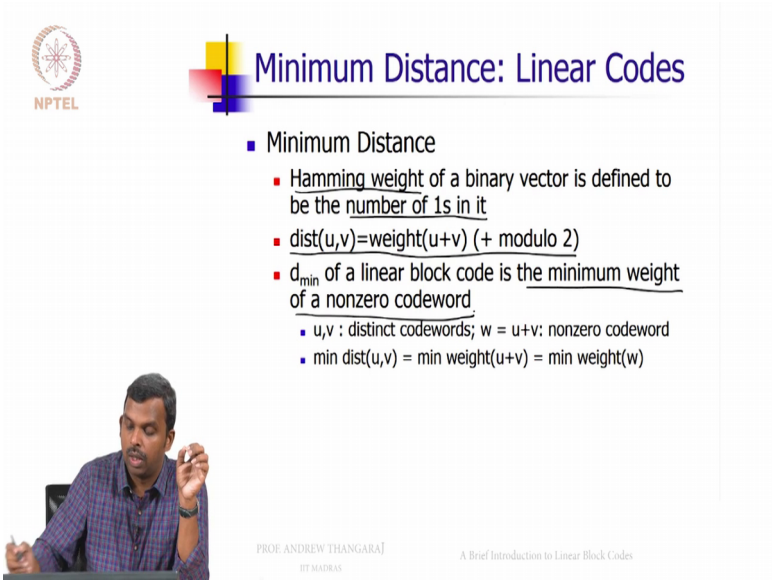
Ok. That is Hamming weight. Remember Hamming distance is between two vectors, number of places in which they differ. If you have 1 vector, Hamming weight is the number of 1s in it, Ok.

Now given 2 vectors u v there is a relationship between the Hamming distance between two u v and the weight of u plus v. So if you take the XOR of u and v, wherever they differ the XOR is going to be 1. Wherever they are the same, XOR is going to be 0. So if you take the

XOR and count the number of 1s you get the Hamming distance. And that is this relationship, Ok.

So if I have a linear block code the minimum distance is actually equal to the minimum weight of a non-zero codeword,

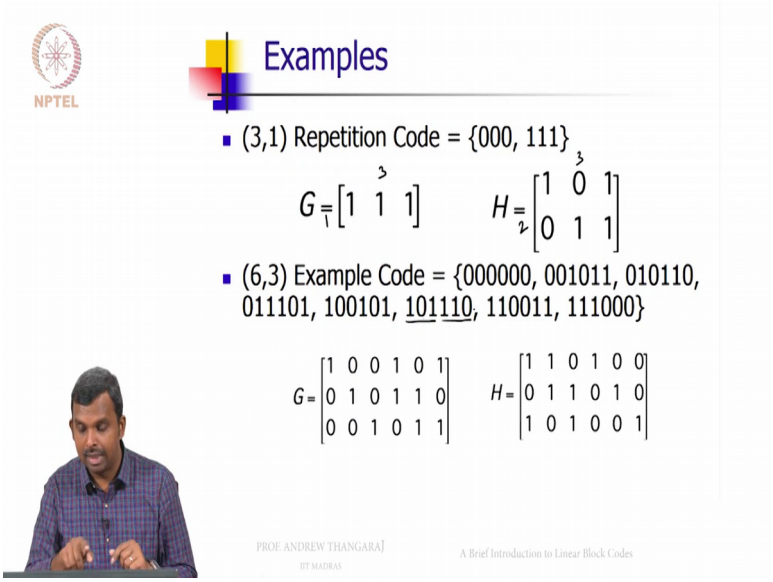
(Refer Slide Time 25:56)



Ok. It is easy to prove this. I have written a small proof for it here. The, for a linear code the minimum distance is the minimum weight of a non-zero codeword.

So in general if you want to have a large minimum distance you should make sure that there are no codewords of low weight in a linear block code. So if you go back and look

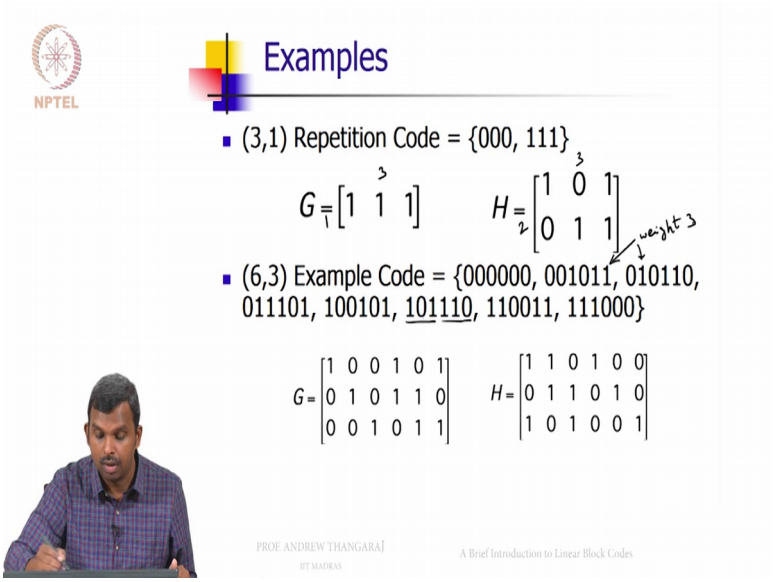
(Refer Slide Time 26:17)



in this example, if you want to find all possible pair wise distances in the 6 comma 3 example code, it is a lot of distances you have to compute.

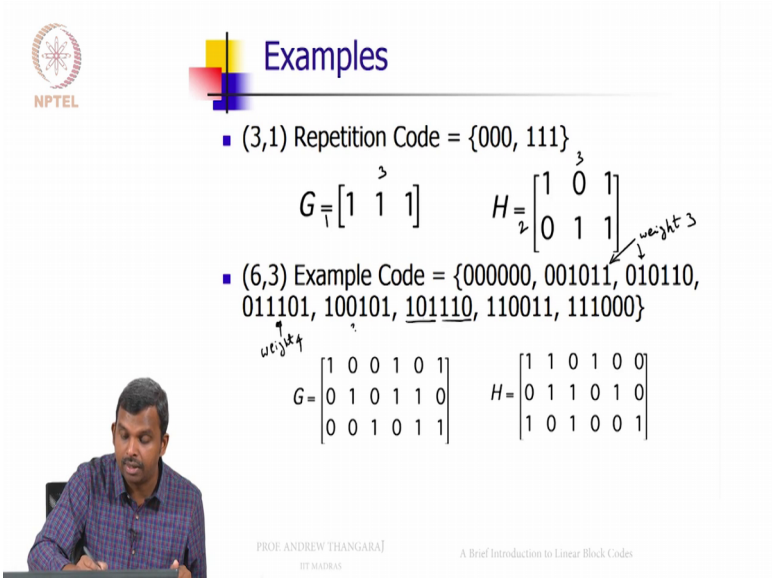
But if you just have to look at the codeword of minimum weight, you have codewords of weight 3, right all this is weight 3,

(Refer Slide Time 26:34)



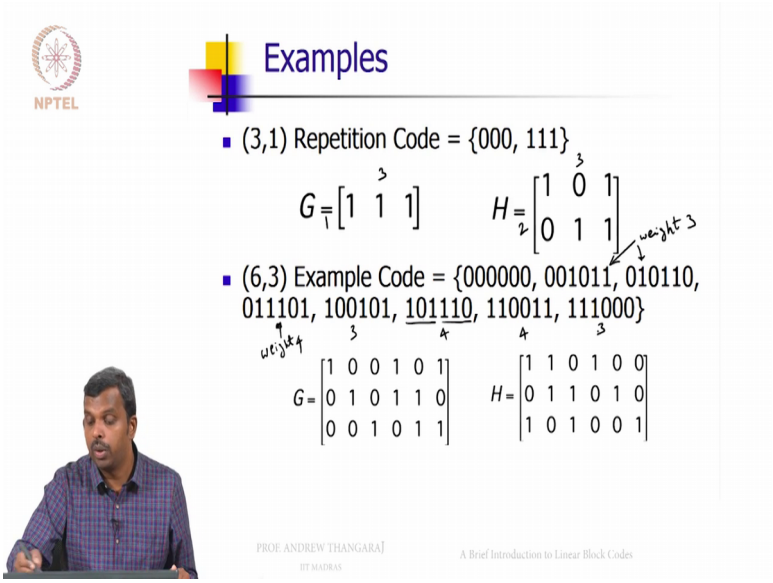
right this is weight 4, right, so this is weight 3,

(Refer Slide Time 26:42)



this is weight 4, weight 4, weight 3. So

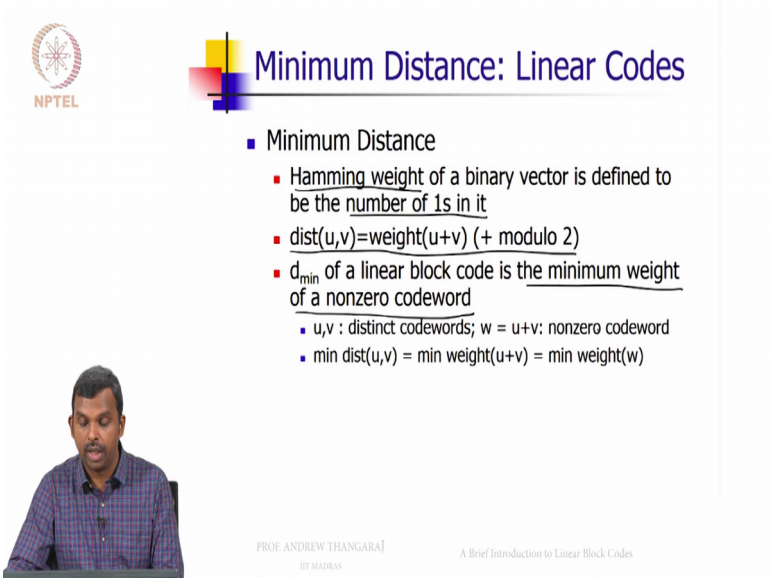
(Refer Slide Time 26:46)



you can quickly tell that the minimum weight of a non-zero codeword is 3 and since the code is linear the minimum distance of this code is also 3, Ok.

So that is a nice, little quick little calculation

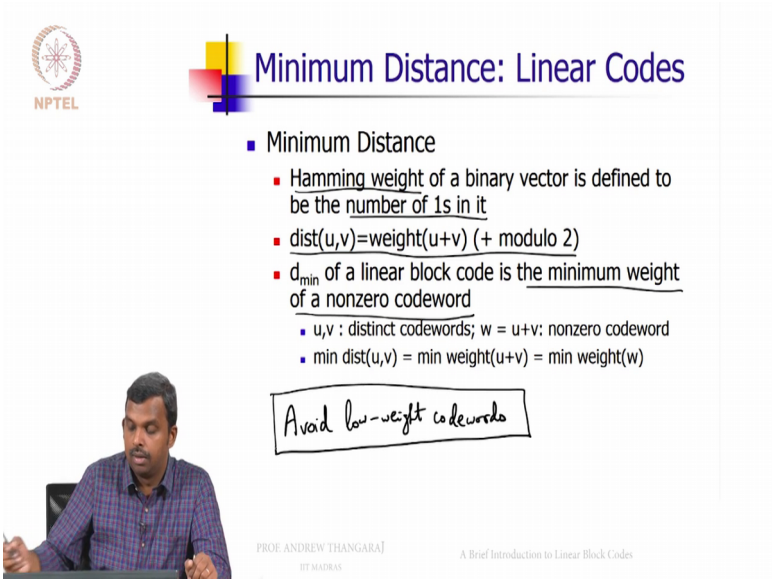
(Refer Slide Time 26:57)



that one can do and that comes from this nice relationship between, the special relationship between minimum distance and minimum weight of non-zero codewords in linear codes, Ok. This is a good thing to know.

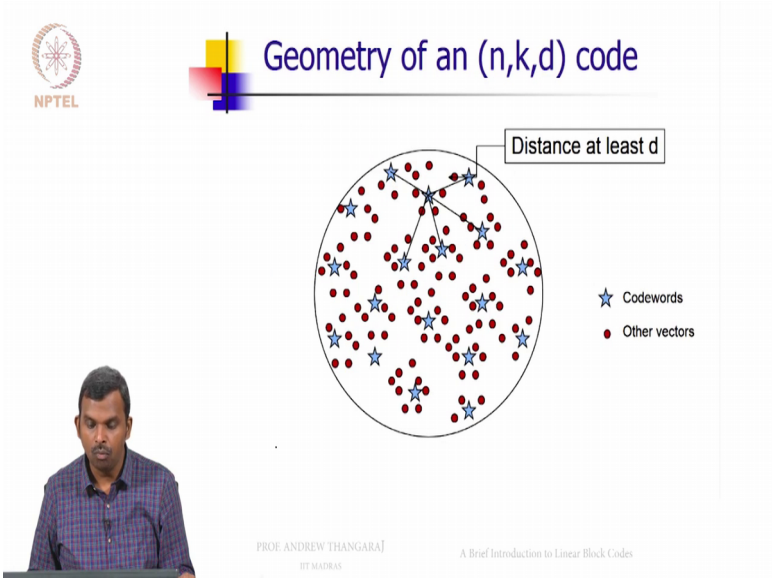
So in general you do not want to have too many low weight codewords in a code, Ok and that is a good design principle, Ok. So avoid low weight codewords, Ok. This is a good design principle for

(Refer Slide Time 27:32)



linear code, you want to design it and come up with a good code, Ok.

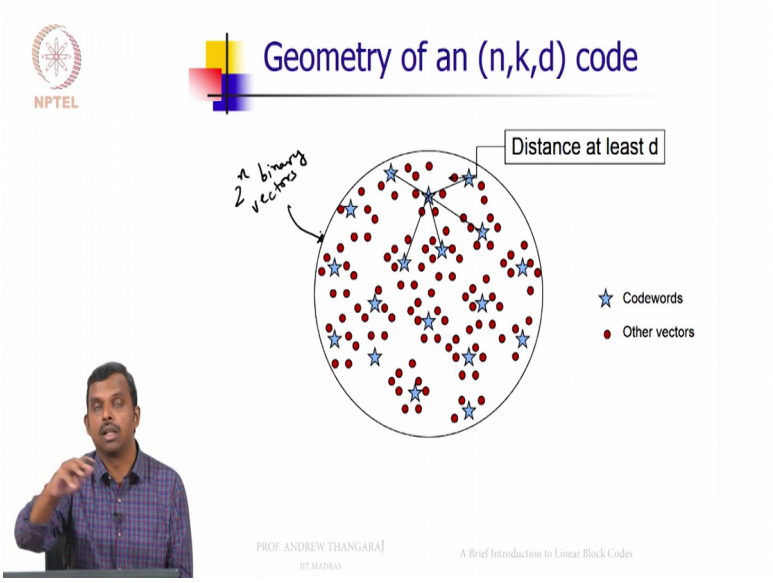
(Refer Slide Time 27:38)



So here is a picture and I think this picture is very good to have in your mind when you think of codes you should have a picture like this in your mind. And this picture is very, very, it gives you a certain intuition of how codes work, Ok.

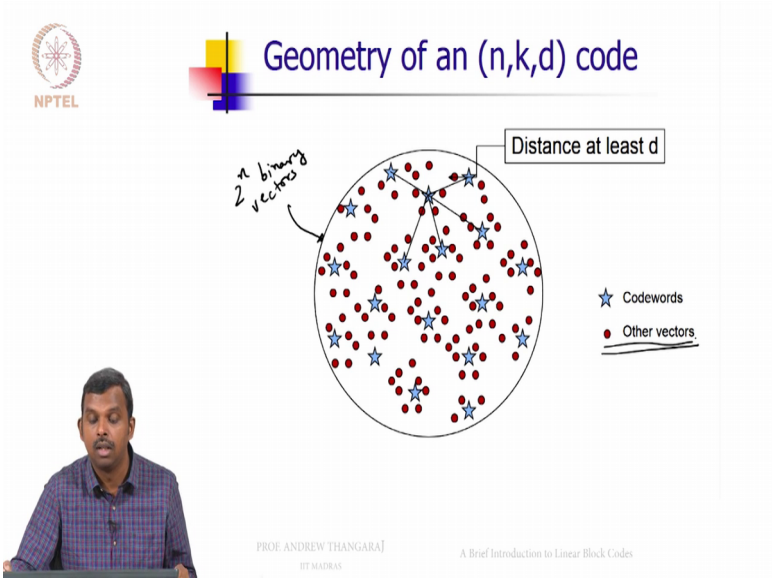
So you can think of the space of all 2 power n binary vectors. There are 2 power n binary vectors. So you put them all in a circle,

(Refer Slide Time 28:08)



Ok. So you imagine they are these small dots. All these vectors are

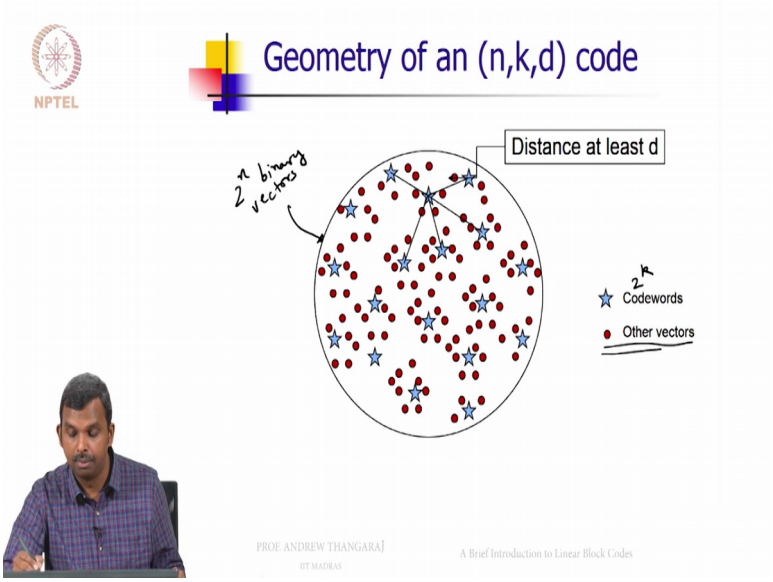
(Refer Slide Time 28:13)



these are the 2 power n binary vectors.

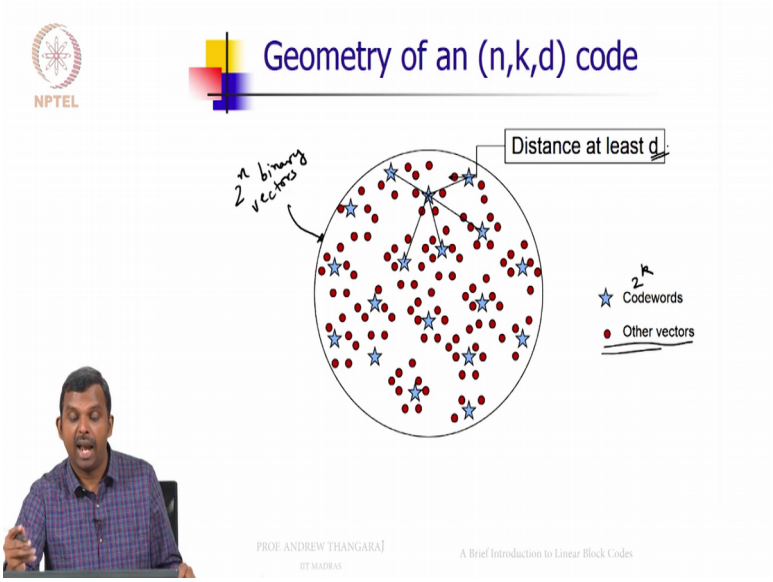
So out of these 2 power n binary vectors there are 2 power k codewords,

(Refer Slide Time 28:21)



Ok. Those I am denoting in stars, these blue stars, Ok and what do we know based on the distance d?

(Refer Slide Time 28:30)



Any two codewords are at least a distance d apart, Ok.

So that is what the minimum distance d means, Ok. And you can imagine at the transmitter when you are transmitting, you are transmitting one of the stars, right. The codeword is what is transmitted.

But of course noise gets added and what you receive is something else, something else away from the star, Ok. And in general when you want to decode you want, you are looking around from wherever you are you are looking around to see the star.

If there are too many stars you are going to get confused, Ok. So whenever you have a received word and you want to look around and try to find, say the closest star or something nearby you should not have too much confusion. Ok. You should know clearly which direction to go, Ok.

And if all these stars are too close by you are going to be confused. You won't know where to go, Ok. So that is actually a pretty good intuition to have about how modern decoders work.

So modern decoders will have the received word and start looking for these codewords, Ok and they look for them in multiple ways. And remember you have to be efficient also. They use some very clever ideas. We will discuss some of these ideas as we go along to search for the closest possible codeword in some sense, Ok

And if you have too many close possible codewords you get confused. So the code becomes bad. So if you design a good code you won't have typical received vectors. You won't have too many codewords contesting to the closest codeword. And your decoder will succeed, Ok.

So this kind of intuition about how the code works is very crucial and when I describe the decoder for these modern codes, I will urge you to keep this keep this picture in mind. And the modern codes use some very nice clever ideas to search for this code, stars so to speak which are nearby received vectors, Ok.

So we will stop here for now. In the next lecture I am going to be doing some Matlab coding to show you how soft decision maximum likelihood decoder works for the Hamming code and may be the example code that we had here, Ok. So we will do that in the next lecture, thank you.