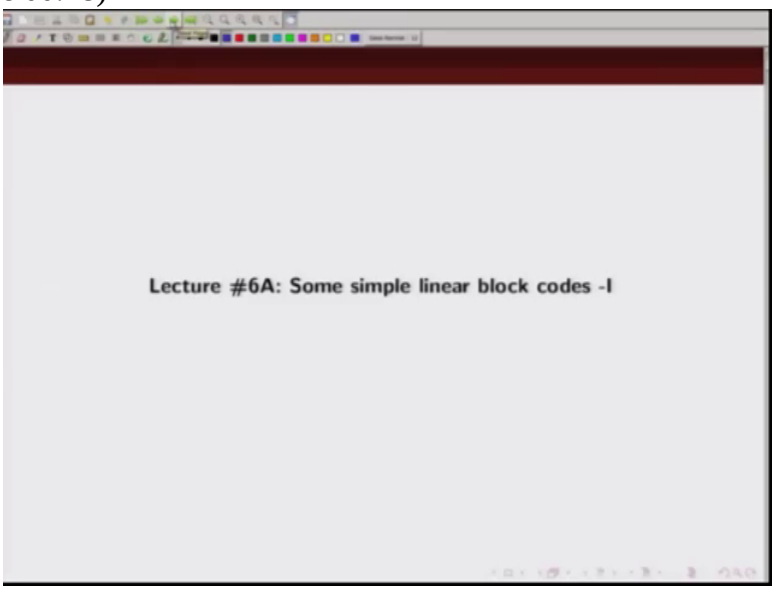
An Introduction to Coding Theory
Professor Adrish Banerji
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Module 3
Lecture Number 11
Some Simple Linear Block Codes-I

(Refer Slide Time 00:14)

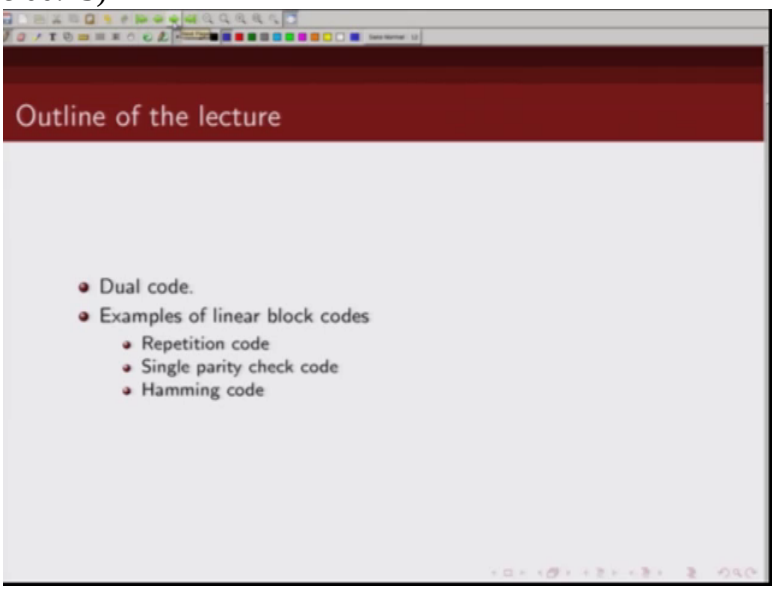
So today we will discuss about some very simple block codes.

(Refer Slide Time 00:18)



So this is the outline of today's talk. Before I discuss some examples

(Refer Slide Time 00:25)

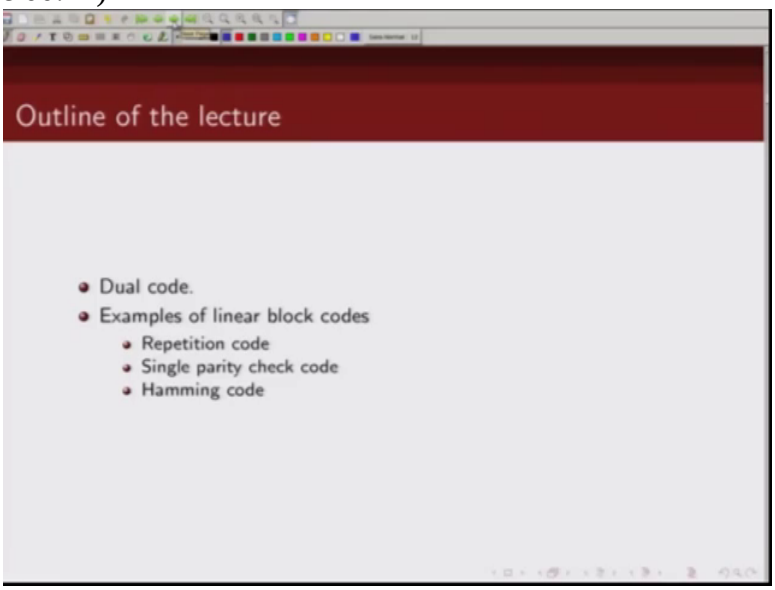


of linear block code I will first describe what do I mean by dual of a code. And then I will move on and describe some very simple linear block codes such as repetition code, single parity check code,

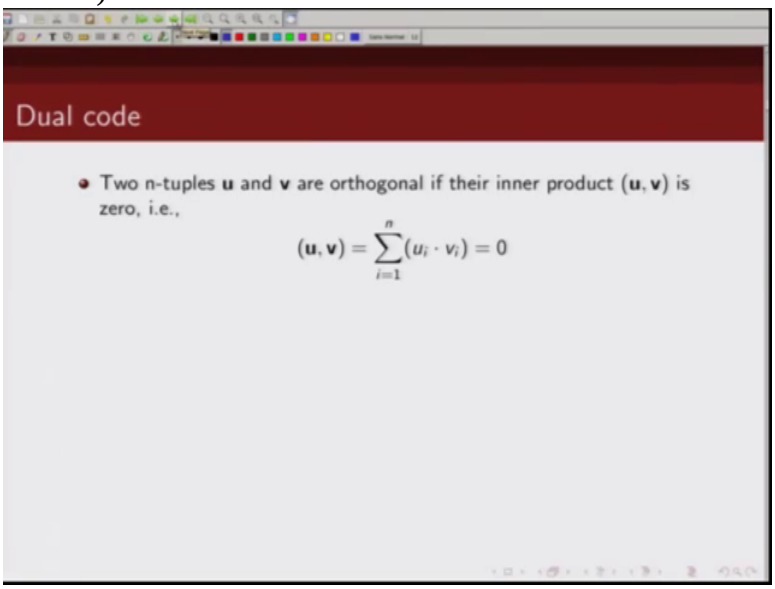
(Refer Slide Time 00:40)

Hamming code.

(Refer Slide Time 00:42)

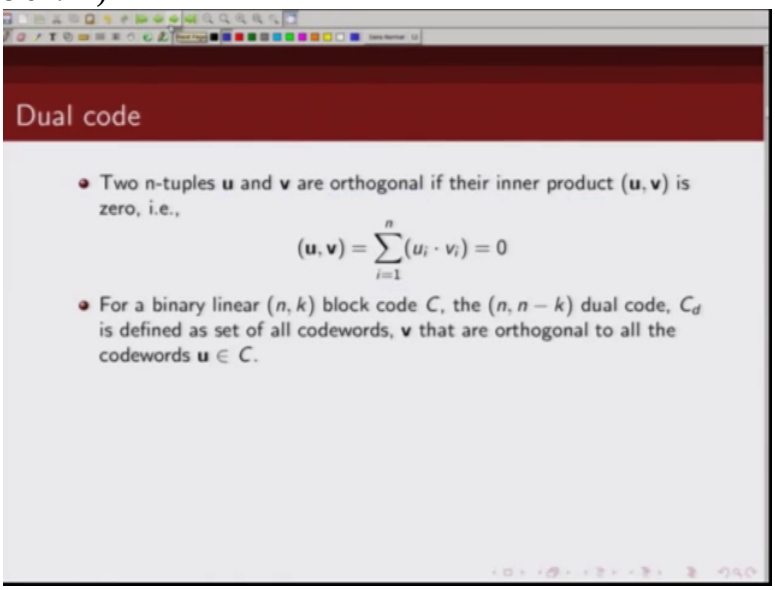


(Refer Slide Time 00:43)



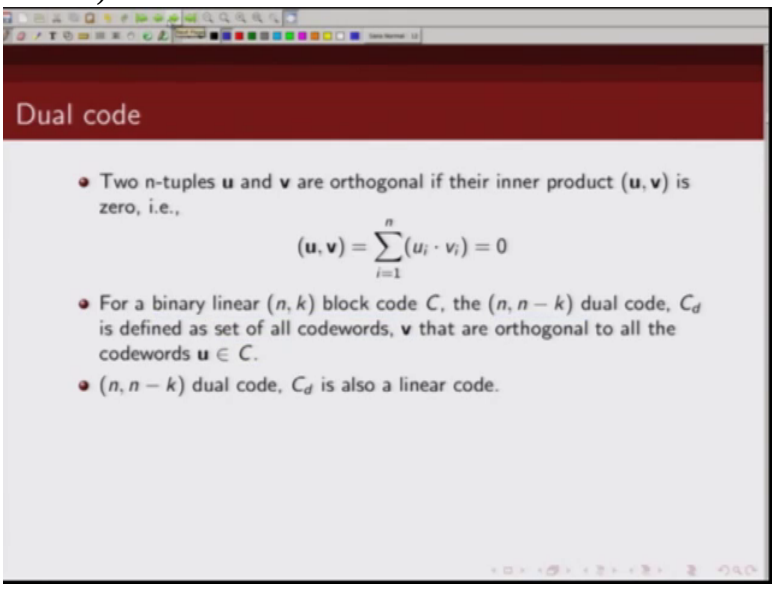
So before I discuss what is dual code, I would like to define what do I mean by two vectors u and v being orthogonal. So two vectors u and v are orthogonal if their inner product which is defined like this, so component wise dot product, if that inner product is zero, we call that these n tuples u and v as orthogonal.

(Refer Slide Time 01:12)



So, for a binary linear n k code, its dual has a parameter n and n minus k and it has the following properties. So a dual code has, is defined such that its set of codewords v are orthogonal to the set of codewords of the original code C. So if v is the codeword which belongs to the dual of the code c then v would be orthogonal to codewords u which belong to the original code C. We can show that,

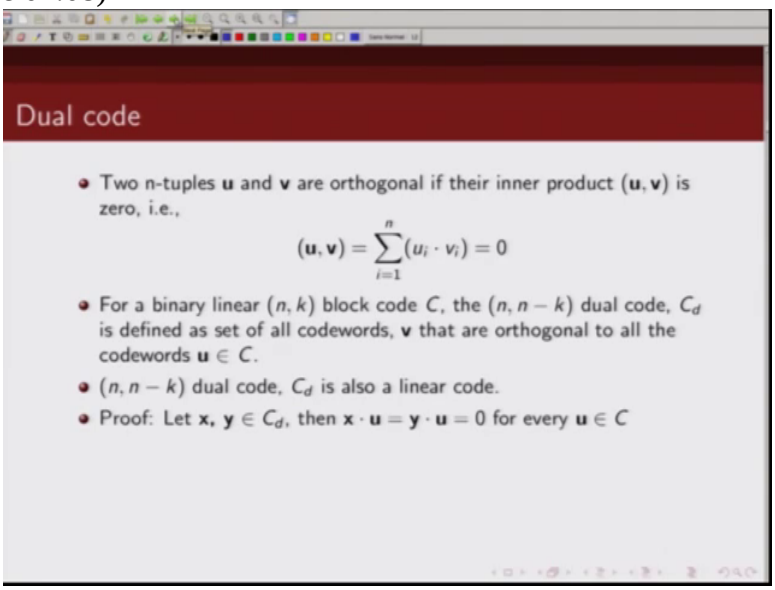
(Refer Slide Time 01:59)



for a linear block code C the dual code is also a linear code.

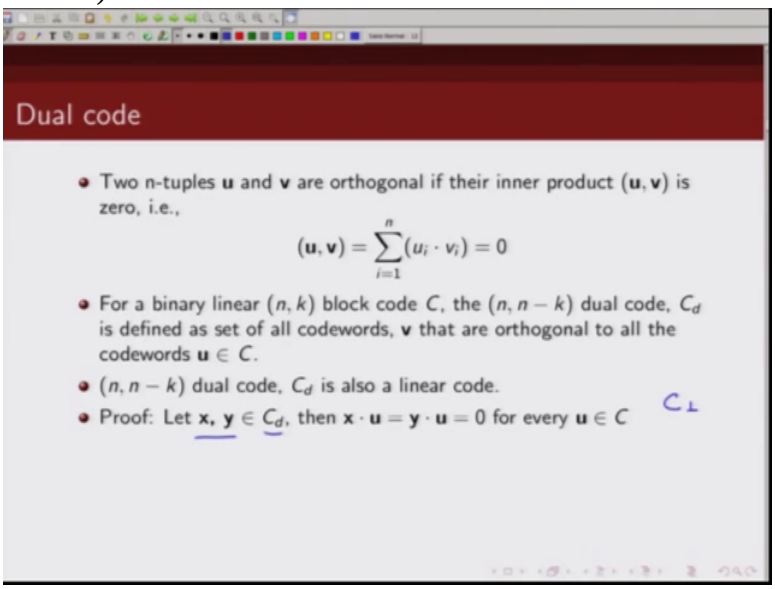
So let's

(Refer Slide Time 02:08)



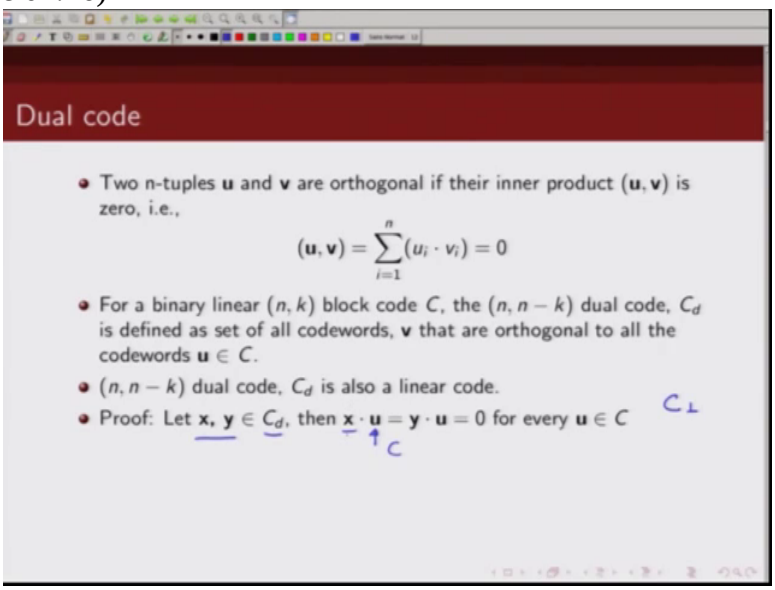
take x and y to codewords which belongs to this dual code of C which we are denoting by C d, sometimes the people use this notation also for the dual. Now if x and

(Refer Slide Time 02:26)



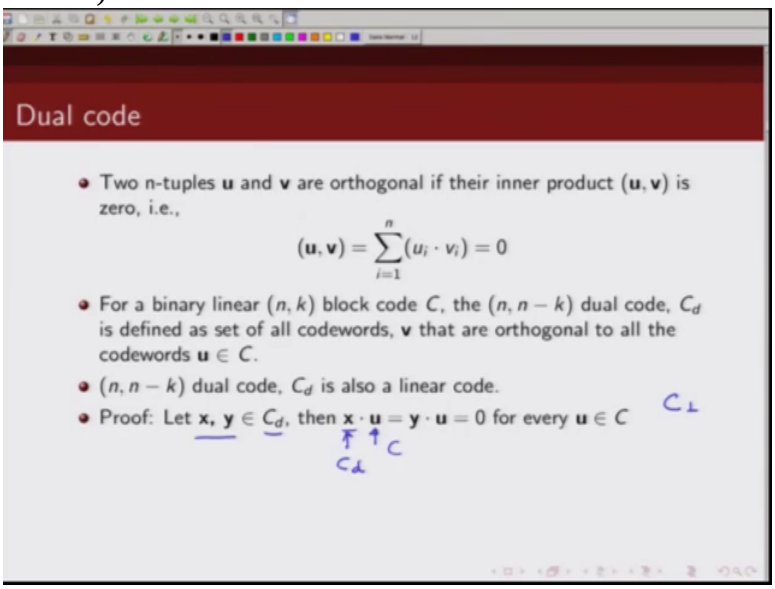
y belongs to the dual code then we know that any codewords belonging to the dual code, they are orthogonal to the codewords in the original code C. So if u belongs to C and x belongs

(Refer Slide Time 02:46)



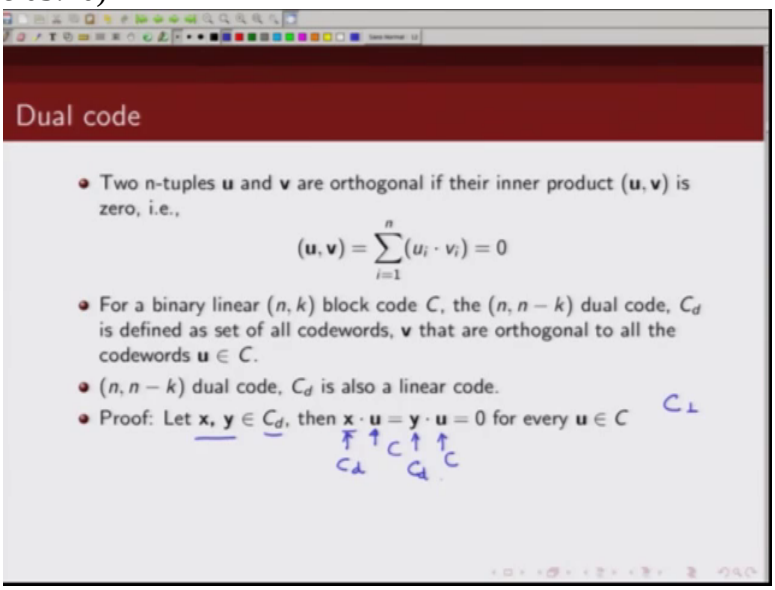
to dual code C of d then

(Refer Slide Time 02:50)



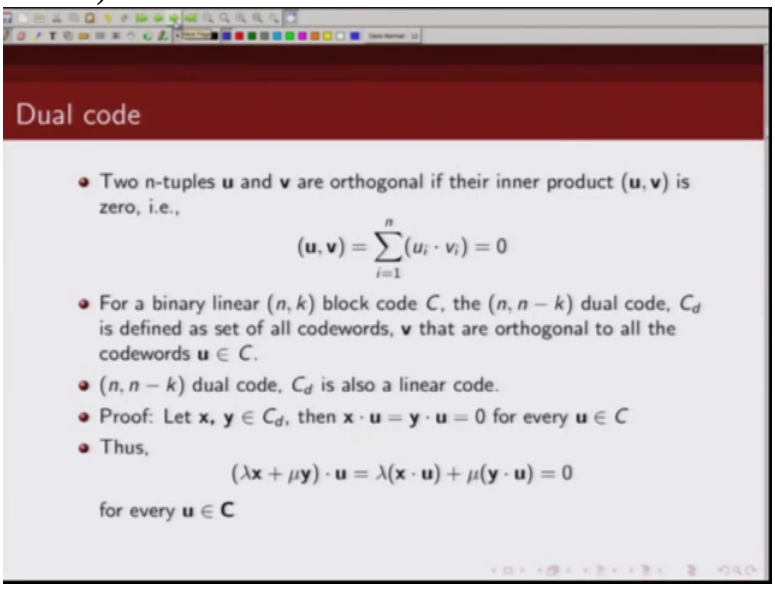
x dot u will be zero because the codeword x is orthogonal to codeword u. Similarly codeword y which belongs to the dual code and codeword u which belongs to original code u, since they are orthogonal,

(Refer Slide Time 03:10)



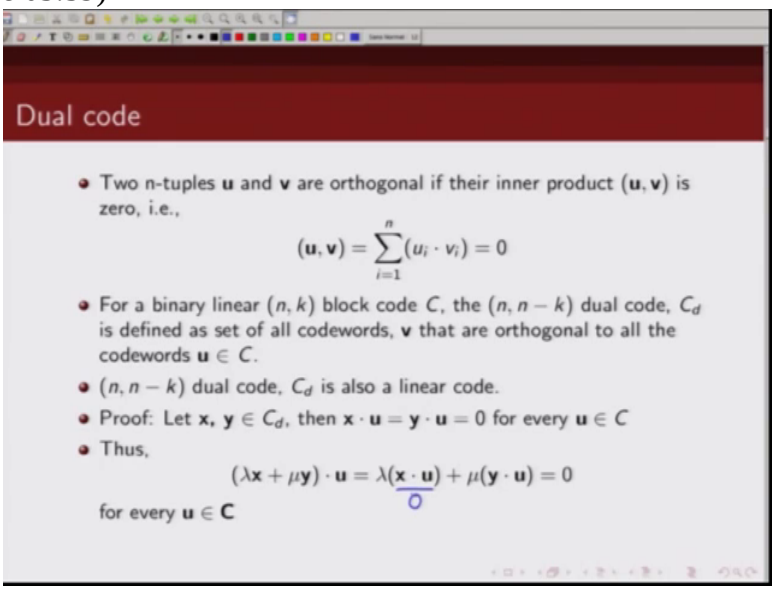
their dot product will be zero. So we can write x dot u is equal to y dot u is equal to zero. This follows from the property that a code which belongs to the dual code is orthogonal to the codewords in the original code C. So therefore some binary

(Refer Slide Time 03:33)



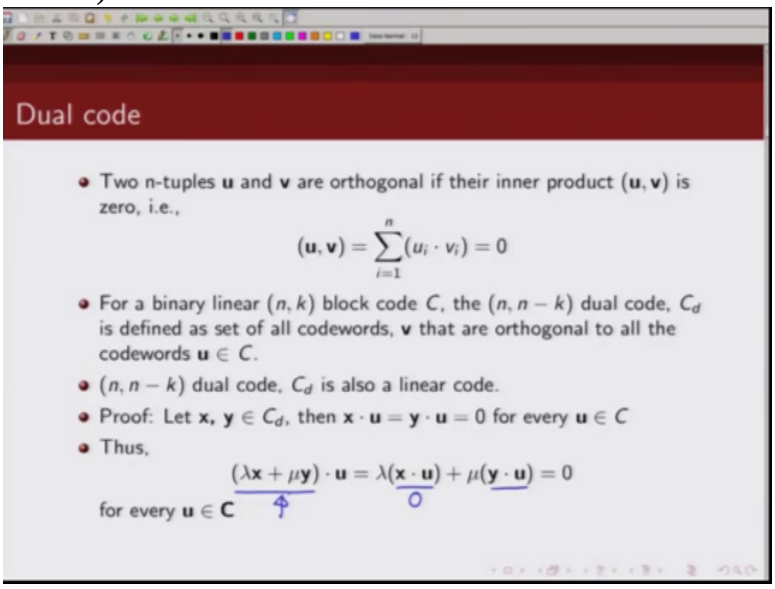
lambda and mu we can write this lambda plus mu times y dot product with u as lambda times x dot mu plus mu times lambda dot u. Now what is x dot u? Now x dot u is zero because x belongs to the

(Refer Slide Time 03:59)



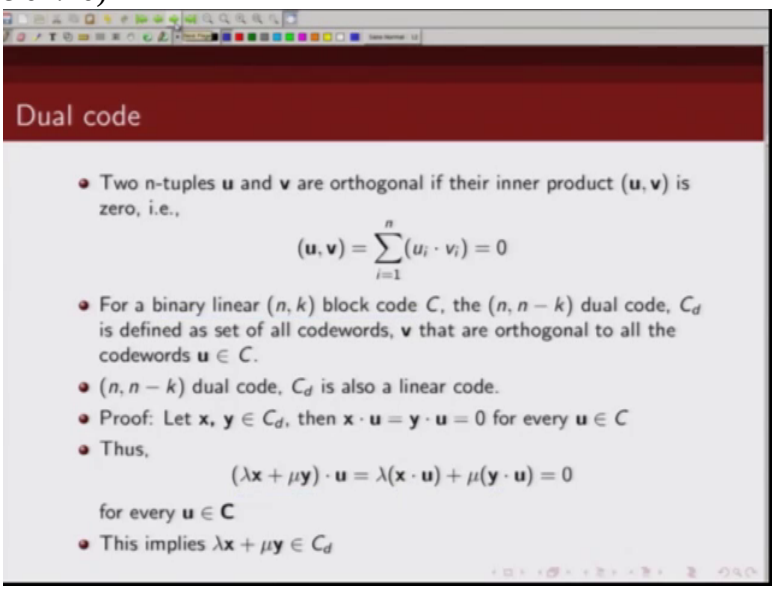
dual code and u belongs to the original code C. So they are orthogonal. That's why x dot u is zero. Similarly y dot u is also zero. Hence we can write lambda x plus mu y dot product with the code u is also zero. So we have shown basically this is also, this belongs to the dual

(Refer Slide Time 04:25)



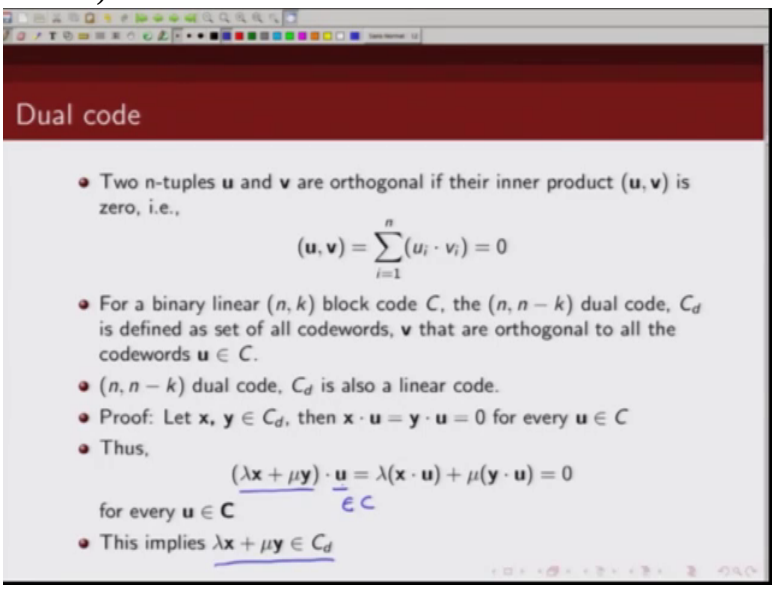
code. So we have shown that if our original code, linear block code is, n k code is linear the dual code is also linear.

(Refer Slide Time 04:40)



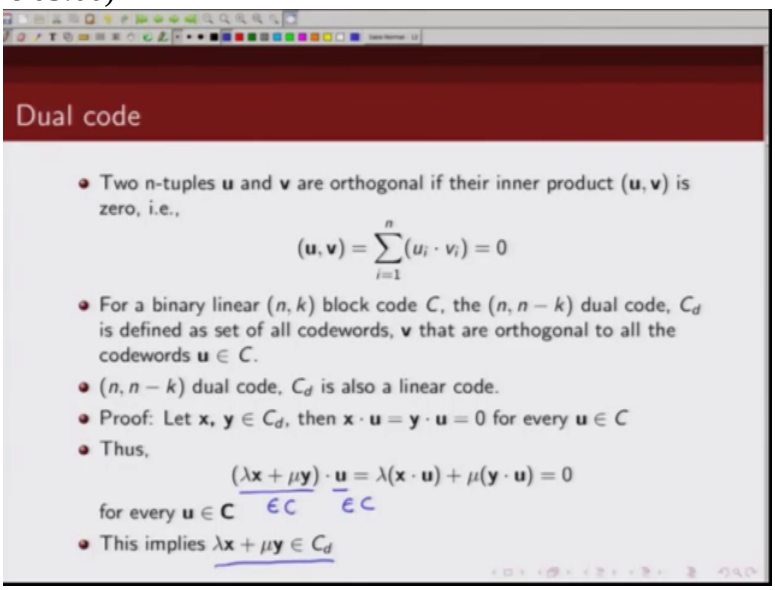
And why does this belong to the dual code? That is because this is orthogonal to the codeword which belongs to the, u belongs to C and since this is orthogonal to

(Refer Slide Time 04:55)



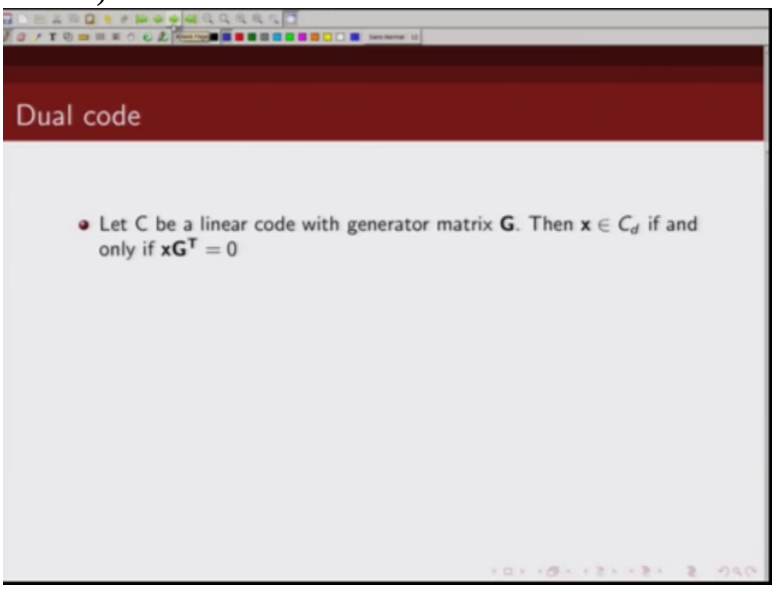
the codeword which belongs to C, so this must belong to the dual code.

(Refer Slide Time 05:00)



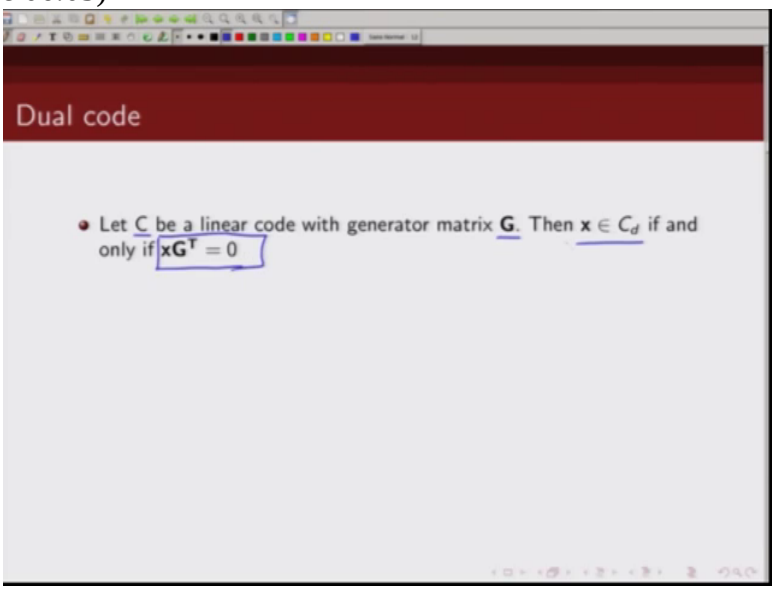
The next property which we are going to show

(Refer Slide Time 05:03)



is, if we have a linear block code, we denote it by C, whose generator matrix is given by this capital G and if x belongs to the dual code of this original code C. So the claim that we are making is if x belongs to the dual code then this relation holds and if this relation holds x belongs to dual code. That is what we mean by if and only if. So if x belongs to the dual code then x G transpose should be zero and if x G transpose is zero x should belong to dual code. So we are going to prove that if x belongs to dual code then this relation holds and further we will show if this condition holds then x will belong

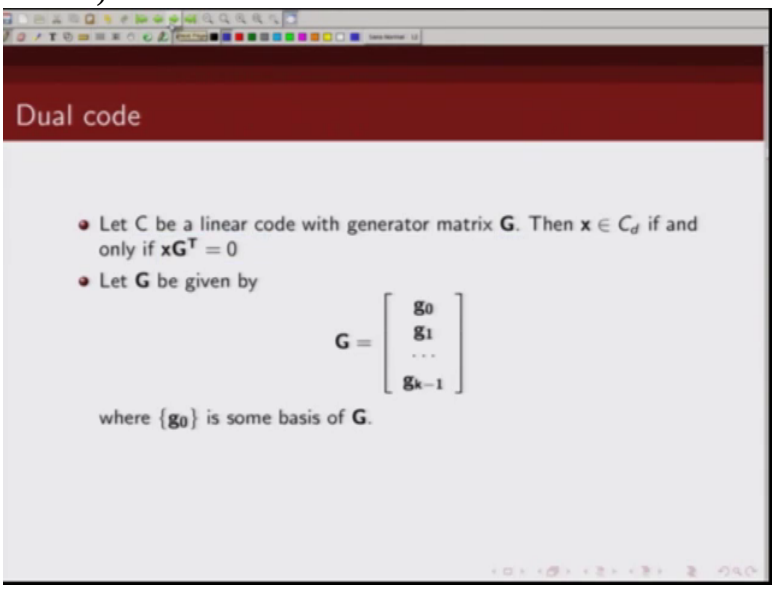
(Refer Slide Time 06:05)



to the dual code.

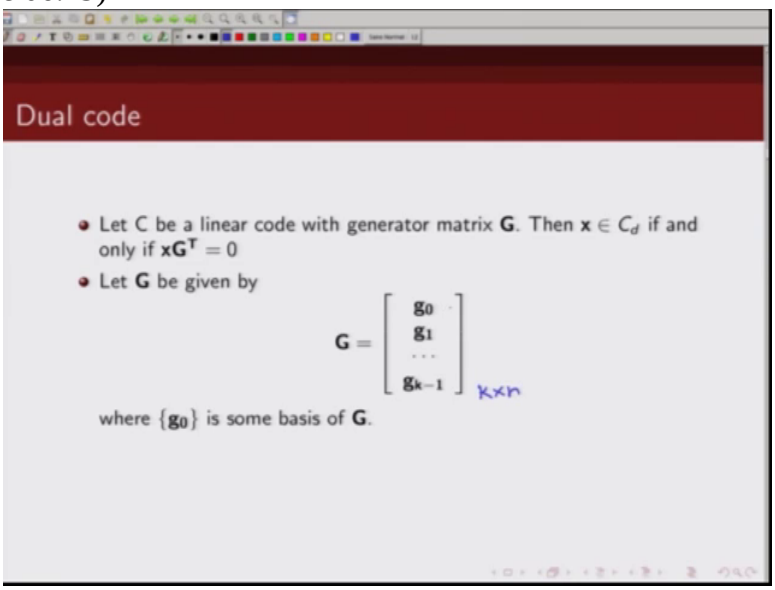
So let us

(Refer Slide Time 06:10)



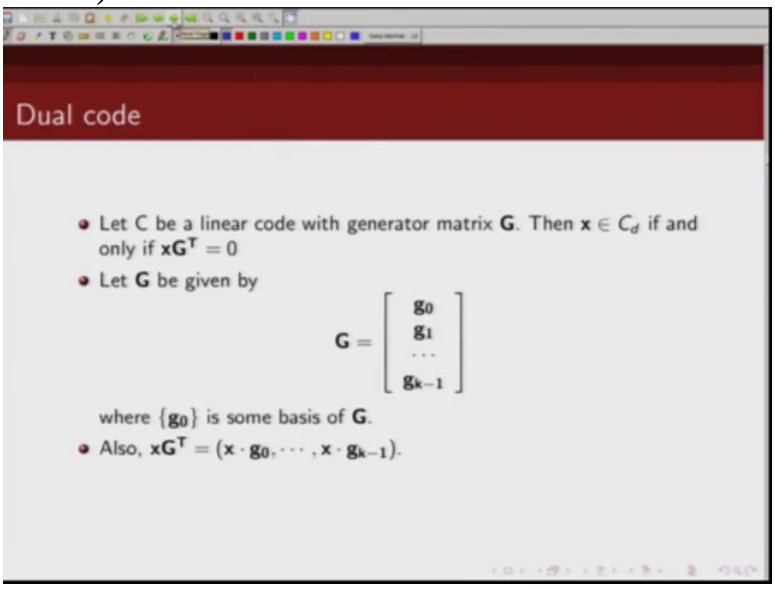
write a generator matrix of a linear block code C. As you know this is a k cross n matrix and this g 0,

(Refer Slide Time 06:19)



g 1, g 2 g k minus 1 are these k generators each of length up to n these are length n so this G is k cross n matrix and any codeword can be generated using these generators g 0, g 1, g 2, g k minus 1. Now

(Refer Slide Time 06:40)

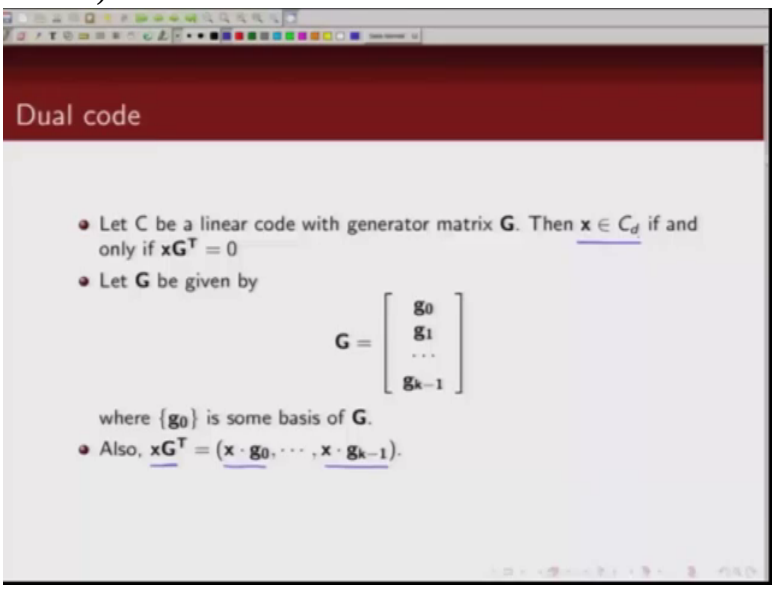


x G transpose is basically given by inner product of x with g 0, x with g 1, up to x with g k minus 1. Now what happens if x belongs to the dual code? If x belongs to the dual code,

(Refer Slide Time 07:04)

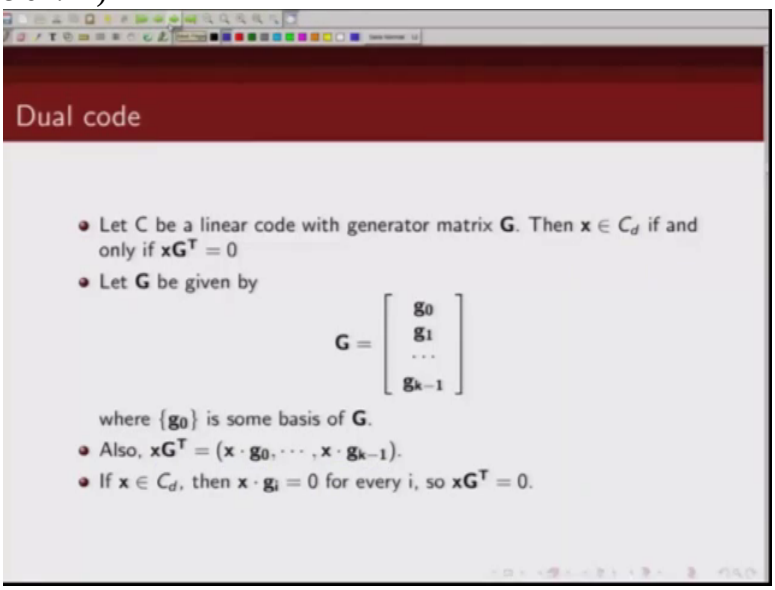
then set of codewords which belongs to the dual code, they are orthogonal to the set of codewords which belongs to the original code C.

(Refer Slide Time 07:16)



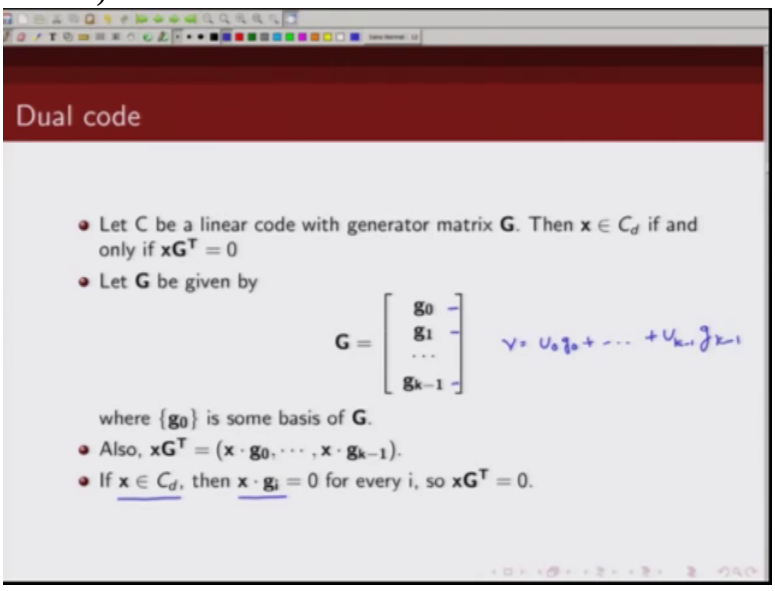
Now

(Refer Slide Time 07:17)



we know that if x belongs to the dual code then x, inner product of x with g i should be zero. Why, because the original code is generated using these generator sequences g 1, g 2, g k minus 1.Any linear combination of this g 0, g 1, g k minus 1 will give me my coded sequence. v is we can write v 0 g 0 plus dot dot dot v k minus 1 g k minus 1 right? So

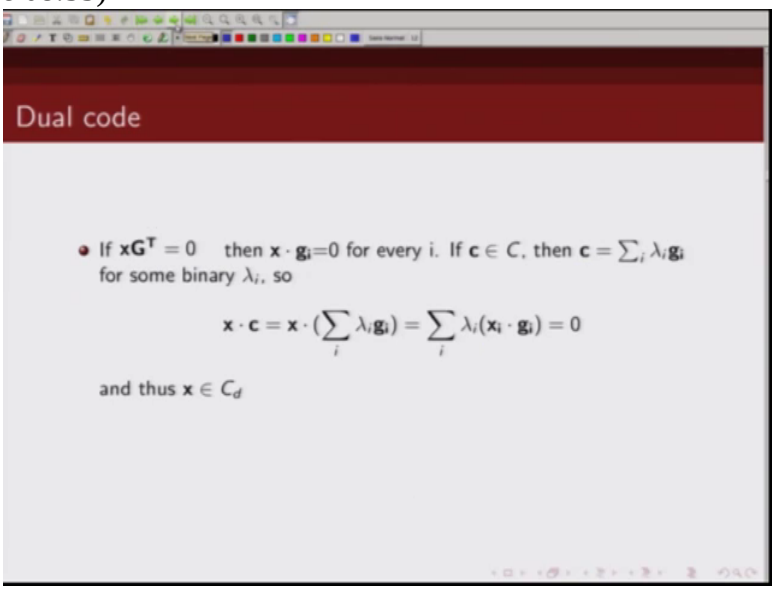
(Refer Slide Time 07:50)



if x belongs to the dual code then inner product uh of x with g i's would be zero and hence x G transpose will be zero. So what we have shown is, if x belongs to dual code, from the property that the codewords in the dual code and the codewords in the original codes, they are orthogonal to each other, from that property we get this condition that the inner product of x with g i's will be zero or in matrix form we can then write, because x, inner product of x with g i's is nothing but x G transpose. So then x G transpose would be zero.

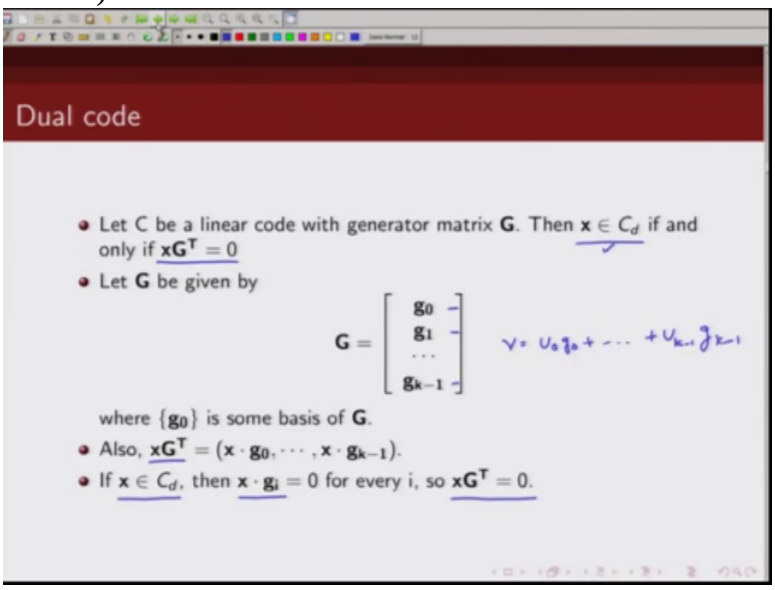
Next we are going to show, if x G transpose is zero then x must belong to the dual code.

(Refer Slide Time 08:55)



Now if x G transpose is zero, then this condition holds. That inner product of x with g i is zero for i equal to zero to k minus 1. Now what is a codeword? Codeword is obtained by linear combinations of the, these generators, g 0, g 1, g 2, g k minus 1. So if C belongs to the, linear block code C then C is essentially generated by linear combinations of these generator sequences where these lambda i's are zeroes and 1's because we are talking about binary code. Now note what do we want to show? We want

(Refer Slide Time 09:43)

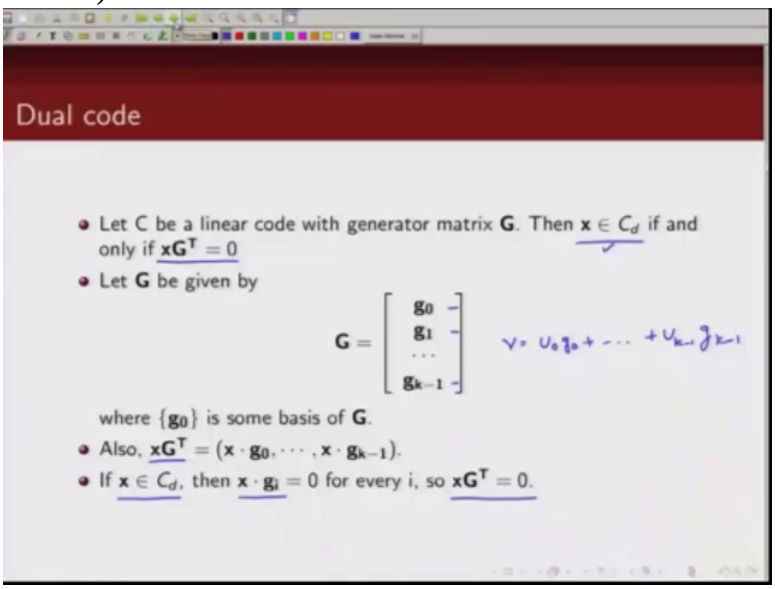


to show, if x G transpose is zero then x must belong to the dual code. And when will x belongs to dual code? We have to show that a codeword which belongs

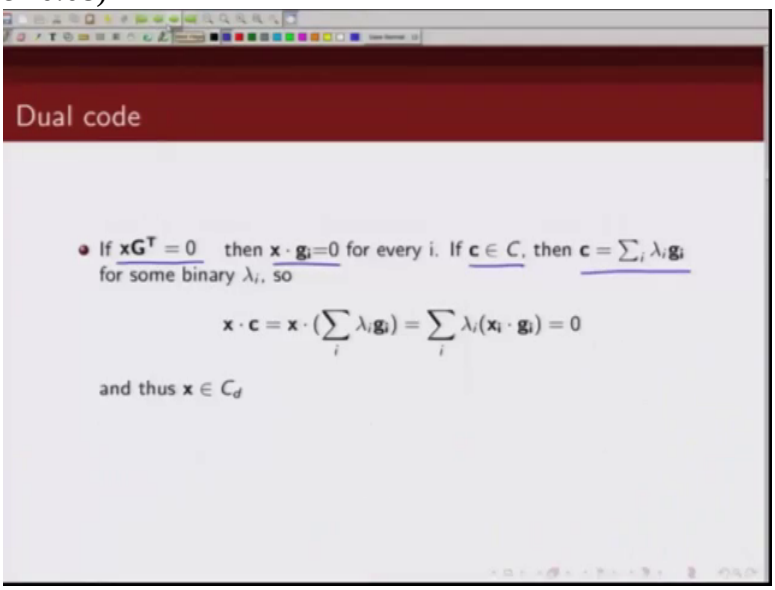
(Refer Slide Time 09:59)

to original linear block code c and set of codewords which belong to the dual code, they are orthogonal to each other. Or

(Refer Slide Time 10:07)

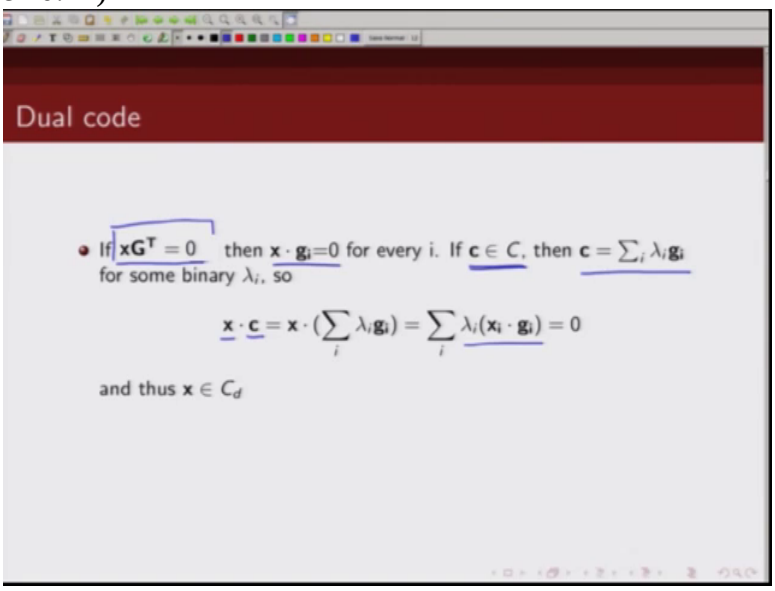


(Refer Slide Time 10:08)



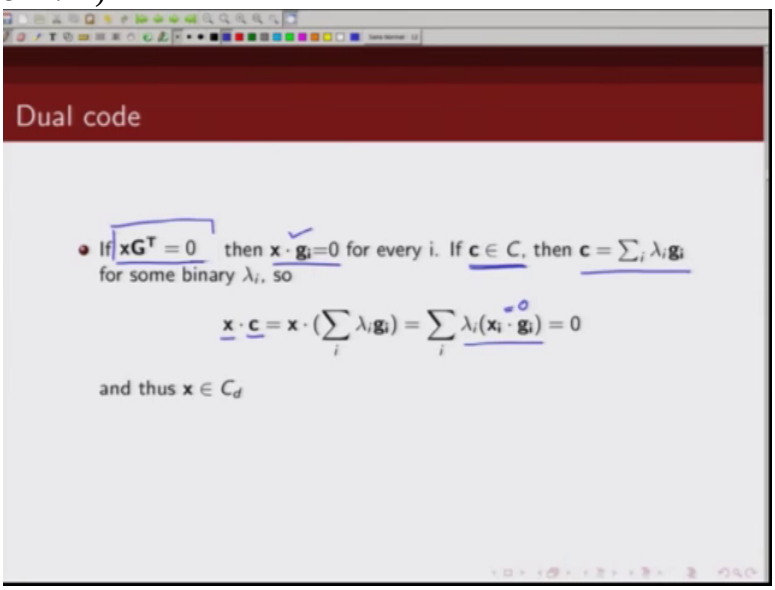
in other words, their dot product is zero. So let's take a dot product; this is x which you want to show that it belongs to the dual code and c is a codeword in the original code C. So what is x dot c? x dot c can be written as x dot summation lambda i g i. This, I can write as summation over i, lambda i x i dot g i. And what do I know from this condition? That x G transpose is zero, I know from this condition that

(Refer Slide Time 10:47)



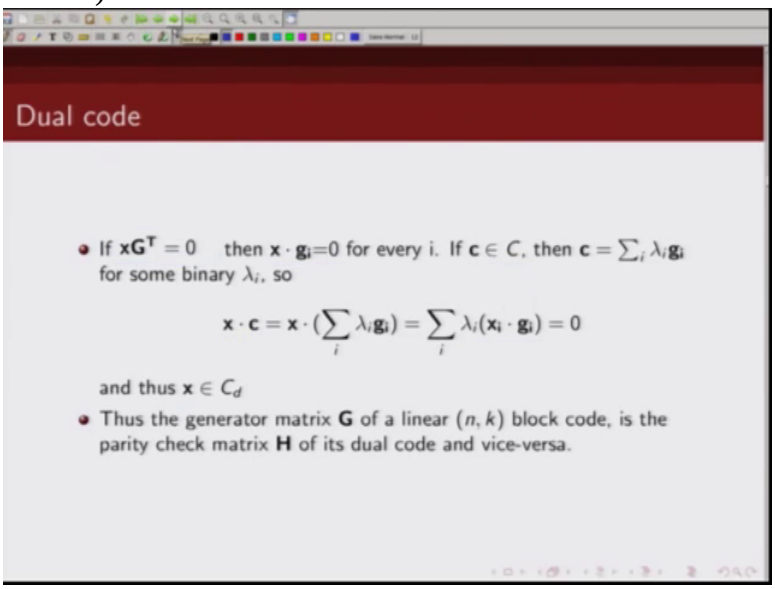
inner product of x with g i is zero. That means this term is equal to zero. Then what I have shown? I have shown that x dot c is equal to zero. That means x and c are orthogonal to each other. So if c belongs to original code this capital C, then x must belong to the dual code of this code C.

(Refer Slide Time 11:17)



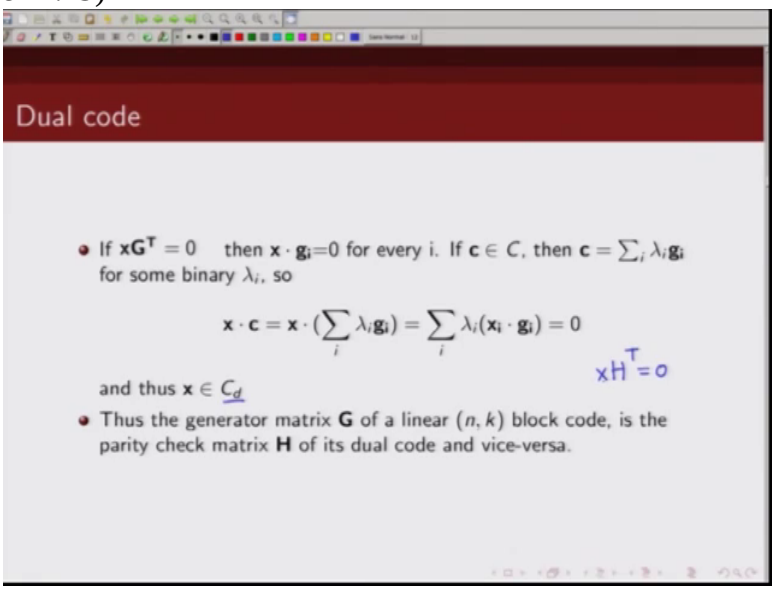
So hence we have shown that x belongs to the dual code of C. Now we

(Refer Slide Time 11:27)



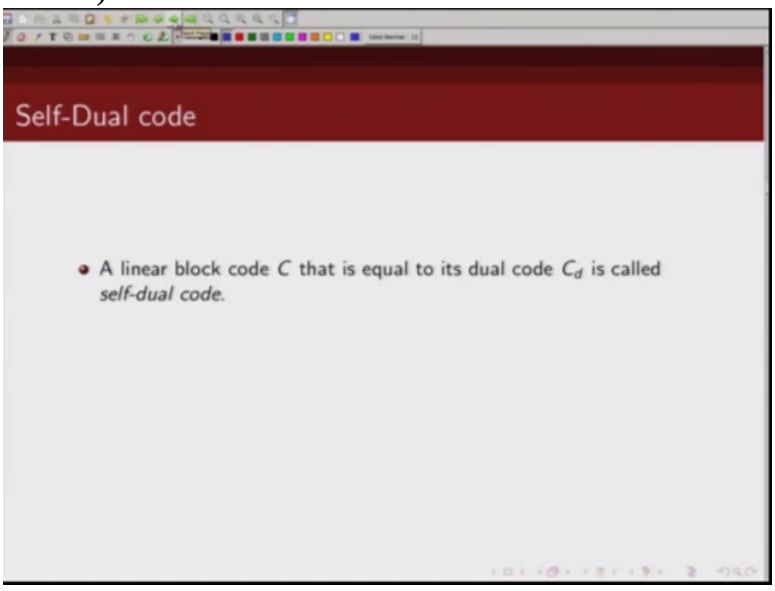
also know that if x belongs to a particular code, let's say, dual code then we know the condition that, if, for any valid codeword we know that, let's say x is a valid codeword then we know this property holds; that

(Refer Slide Time 11:49)



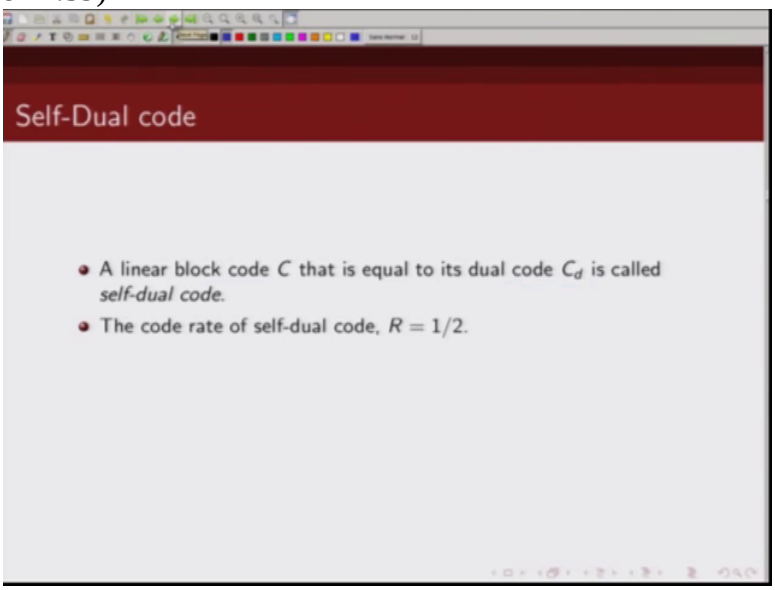
uh codeword parity check matrix transform is basically equal to zero. So if we compare this form with this, you can immediately guess that the generator matrix for the dual code is given by the parity check matrix of the original linear blockcode C. And the parity check matrix of the dual code is given by this generator matrix of the original code C. So this is what I am saying here. The generator matrix G of a linear blockcode is the parity check matrix for dual code and vice versa.

(Refer Slide Time 12:44)



Ok, now what is a self dual code? If a linear block code is same, is equal to its dual code then it is called a self dual code. So as you can make out

(Refer Slide Time 12:59)

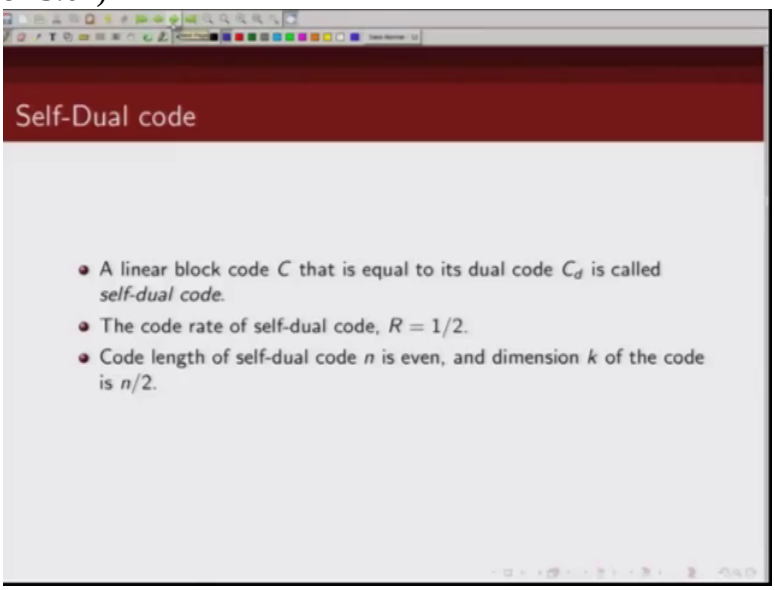


the rate for the self dual

(Refer Slide Time 13:02)

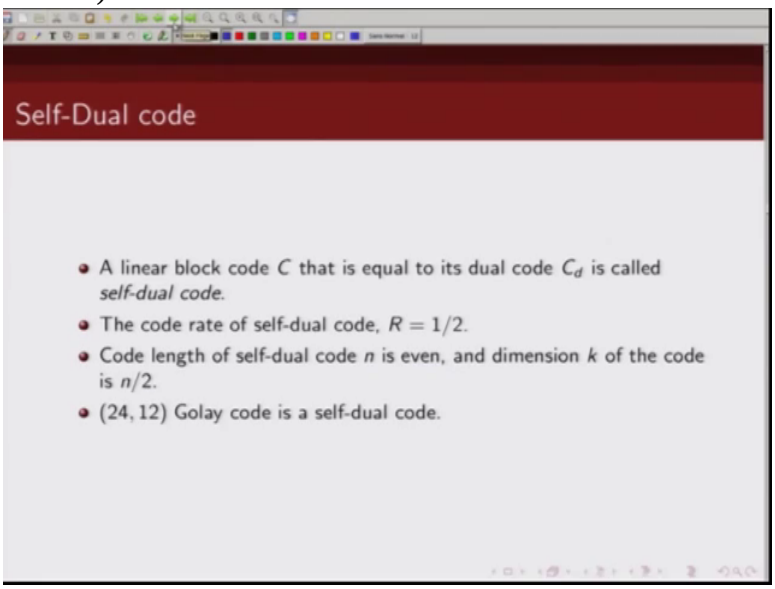
code should be half because k should be n by 2. So

(Refer Slide Time 13:07)



n is always even for a self dual code and the dimension of the code is always n by 2. An example of self dual

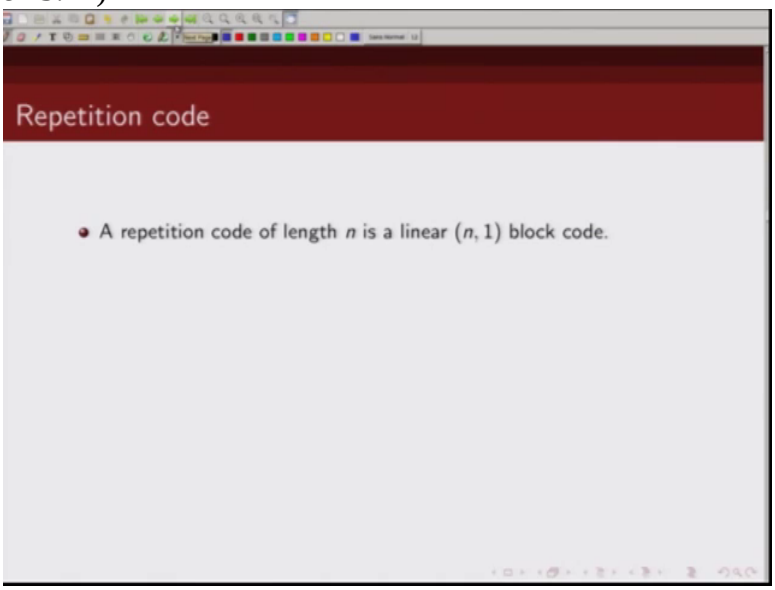
(Refer Slide Time 13:16)



is 24 12 Golay code.

Ok now that we have

(Refer Slide Time 13:21)

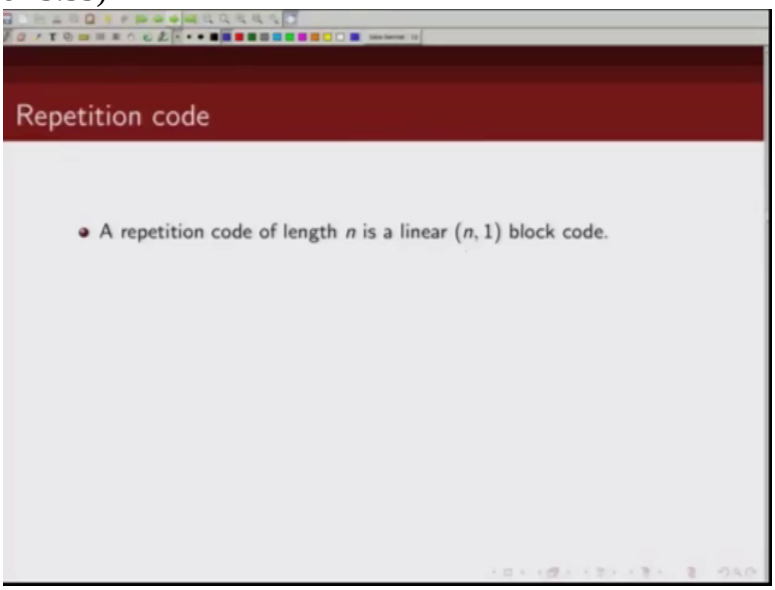


defined dual code let's now come to the other topic which is some examples of linear

(Refer Slide Time 13:28)

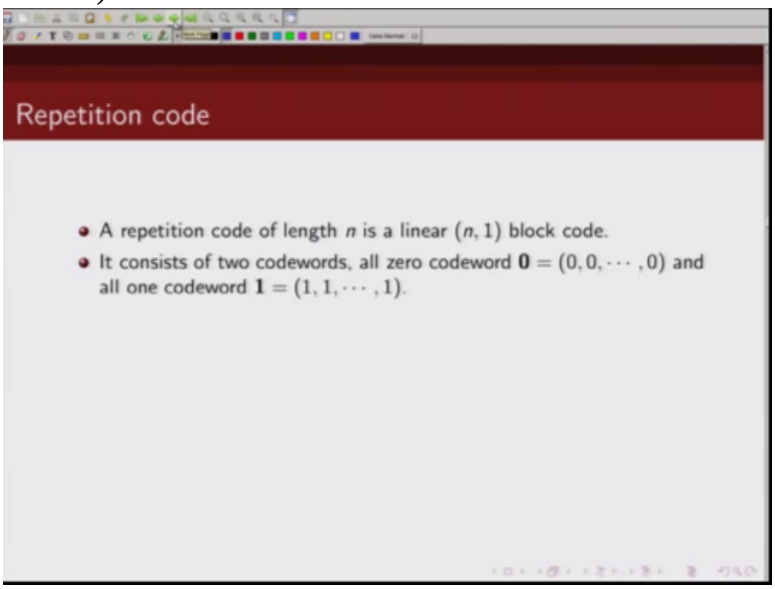
block code. So a very simple example is the repetition code. So for repetition

(Refer Slide Time 13:35)



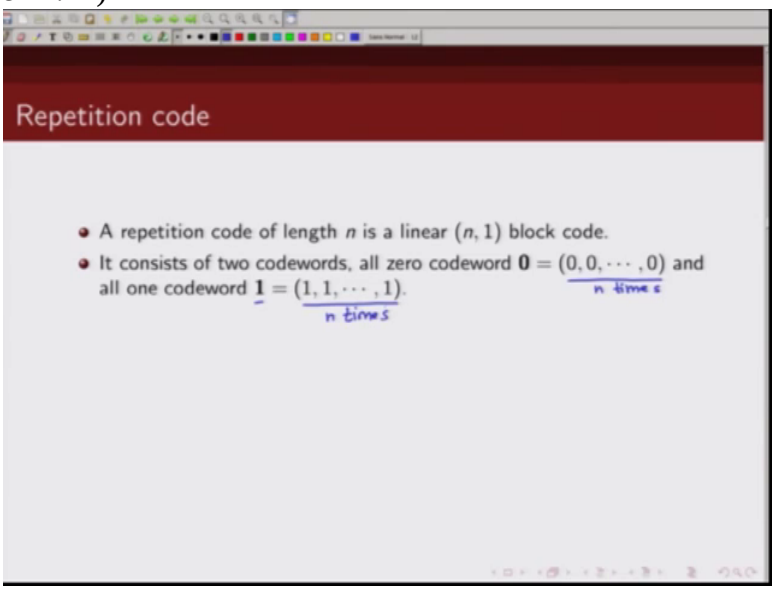
code k is 1 and let's say we have a rate 1 by n repetition code. So the codeword length is n, there is one input bit and n outputs. And how is

(Refer Slide Time 13:51)



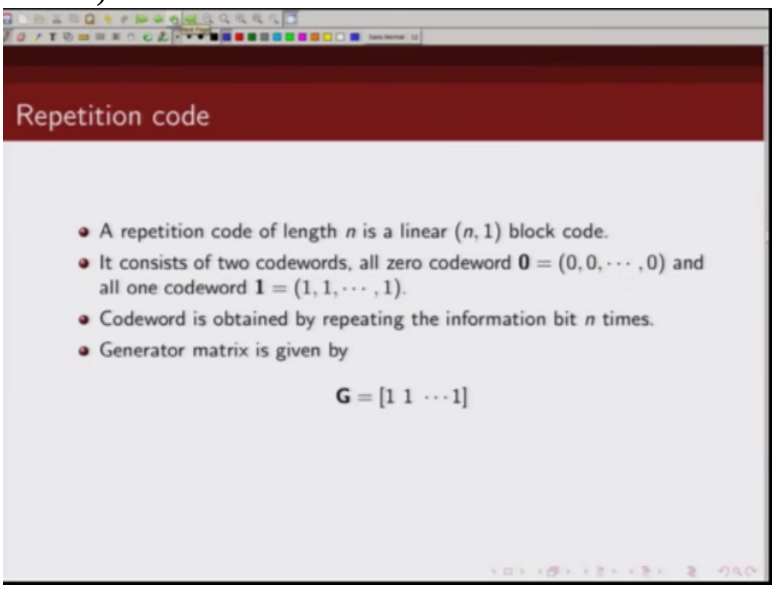
repetition code generated? So we repeat the same information n times. So for a binary repetition code, it will have two codewords, 0 and 1. So if the information sequence, information bit is 0, the output will be all zeroes repeated n times. And if the input is 1, this will be, output will be all 1 repeated n times. So there are only two

(Refer Slide Time 14:24)



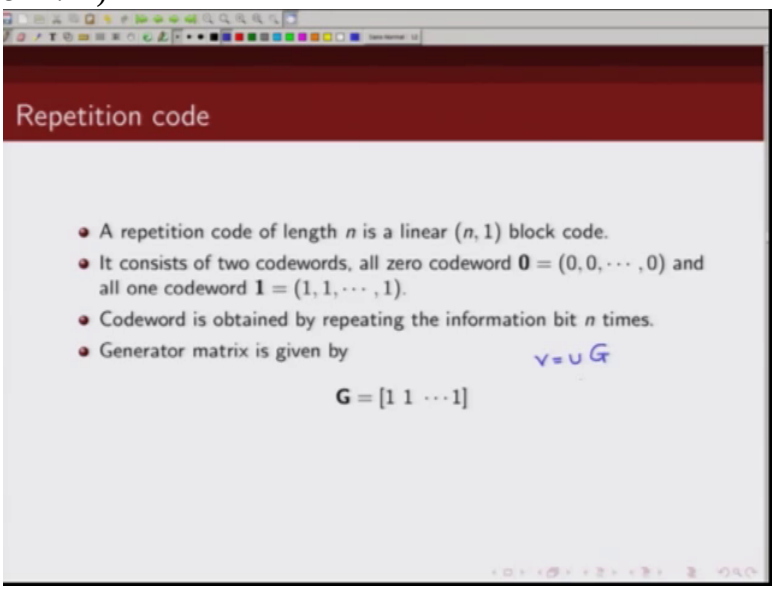
codewords in our repetition code. So we can then

(Refer Slide Time 14:31)



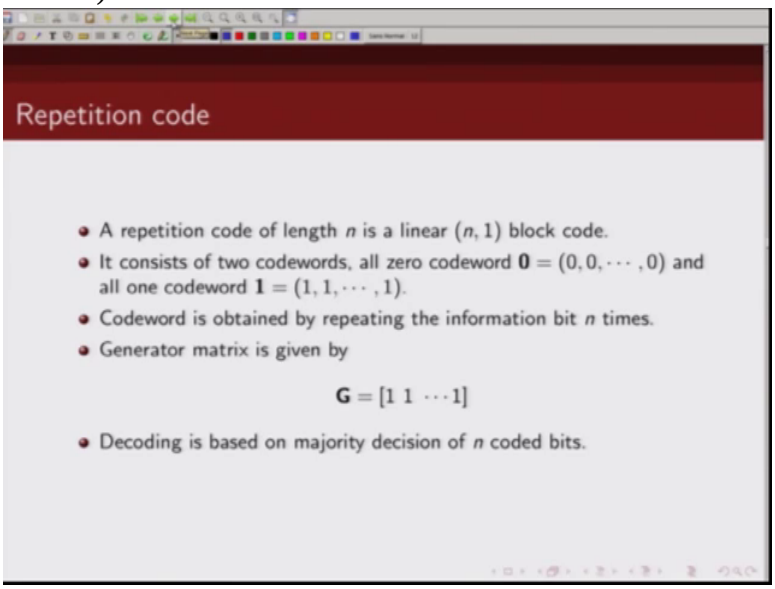
write down our generator matrix. How is generator matrix? If v is the coded sequence, u is the information sequence then generator matrix; they are related to generator matrix in this particular way.

(Refer Slide Time 14:44)



So, since our output is the bit repeated n times, the generator matrix for repetition code will be all 1s. So this is 1 cross n. Now how do we decode

(Refer Slide Time 15:01)

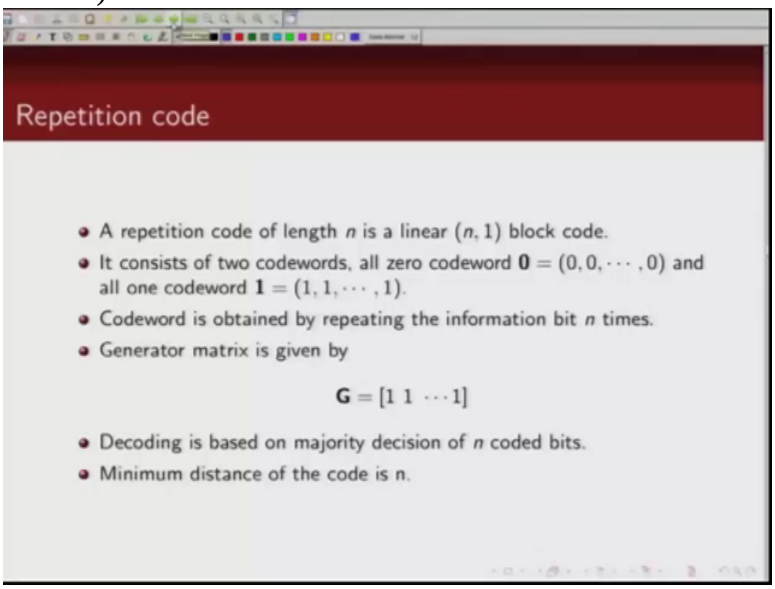


a code which is encoded using repetition code? So what we do is we take a majority decision. So, let's say, for zero, we are sending n zeroes

(Refer Slide Time 15:13)

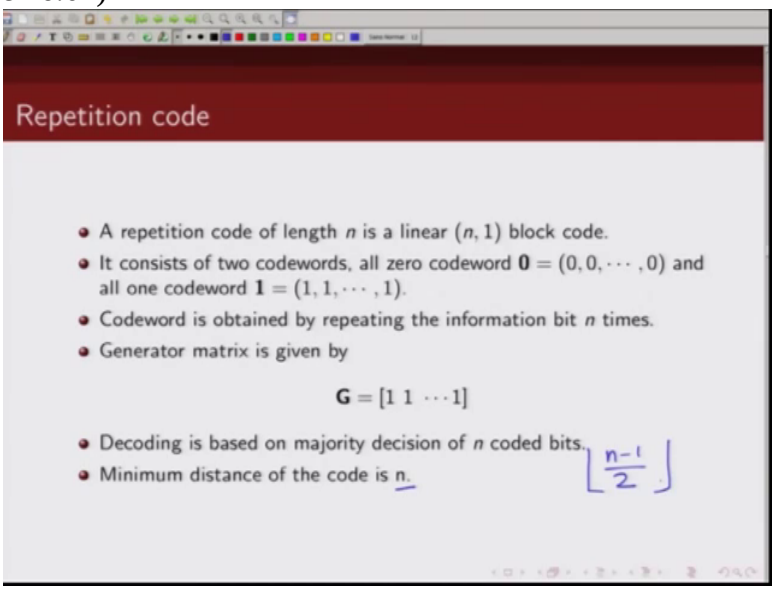
and for 1 we are sending n 1's. So at the receiver, when some of the bits get flipped or they are changed, what do we notice is that we look at block of n bits and we see what is the majority? Is it 0 or 1? If it is 0, if majority of the bits are 0 we decide in favor of 0. If it is a 1, we decide in favor of 1. And

(Refer Slide Time 15:41)



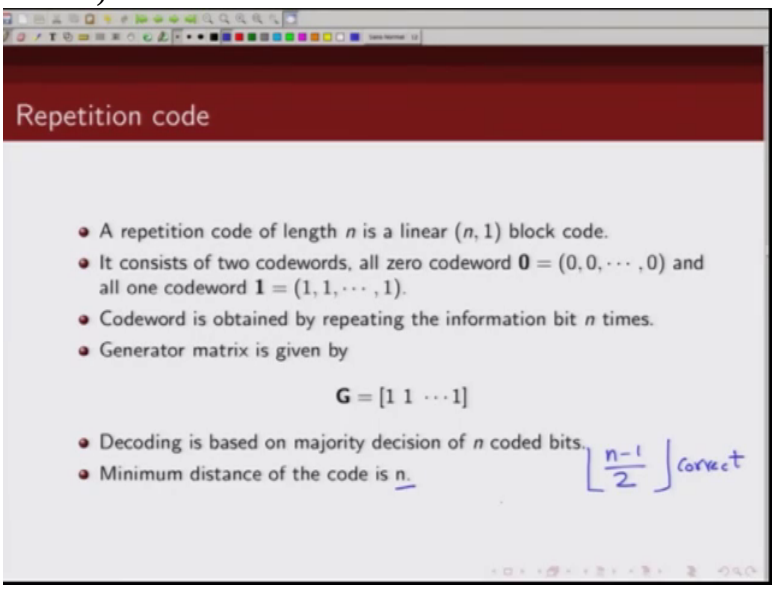
we can see the minimum distance of the code is n. Because there are only two codewords, all 0's codewords and all 1's codewords. So it can correct n minus 1 by 2, floor of that, it can correct so many errors,

(Refer Slide Time 16:01)



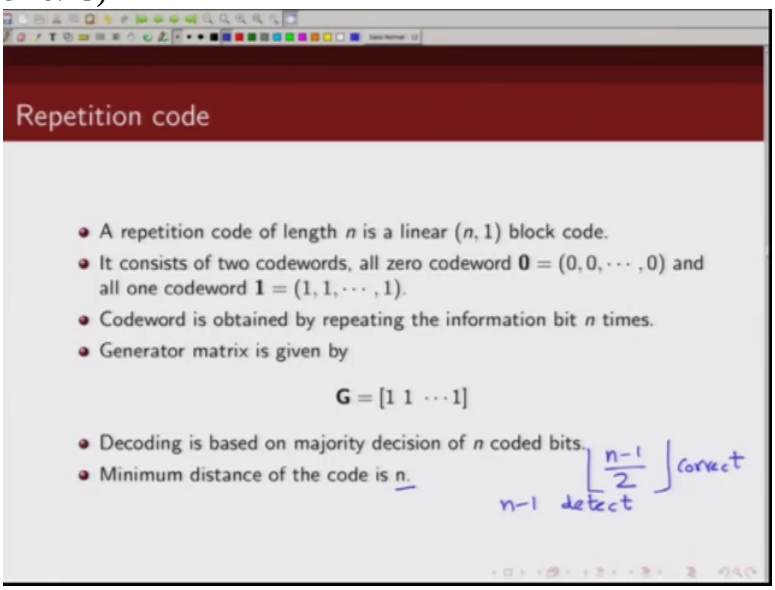
it can correct so many errors and it can detect

(Refer Slide Time 16:06)



n minus 1 errors. Because any error pattern

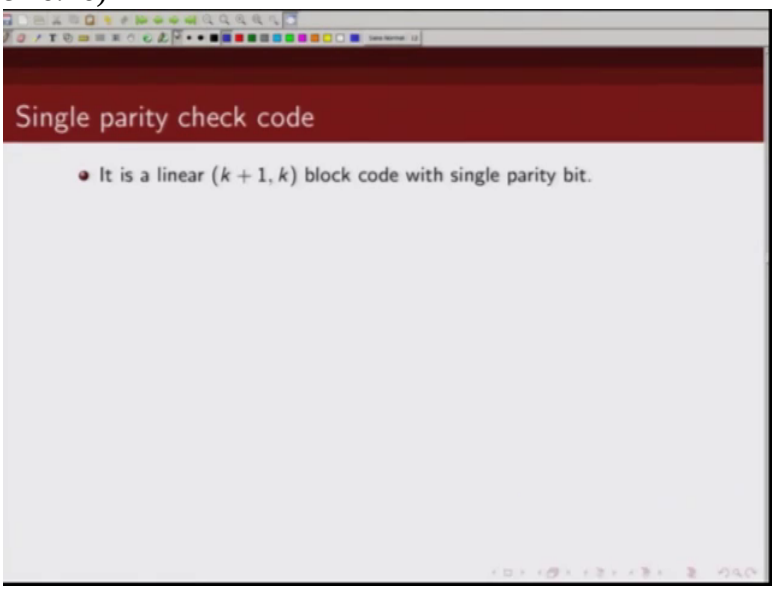
(Refer Slide Time 16:13)



of weight less than n would not change it into any valid codeword. So all error patterns up to weight n minus 1 can be detected by this repetition code.

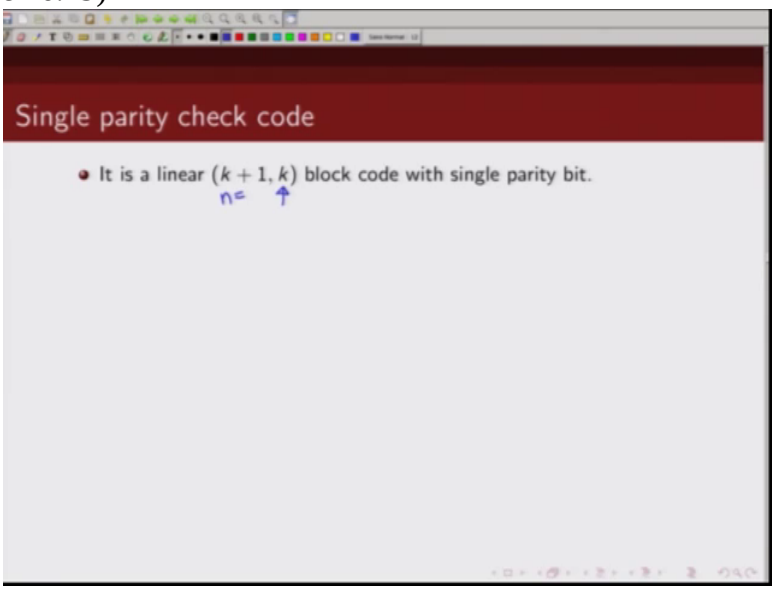
Let's look at

(Refer Slide Time 16:26)



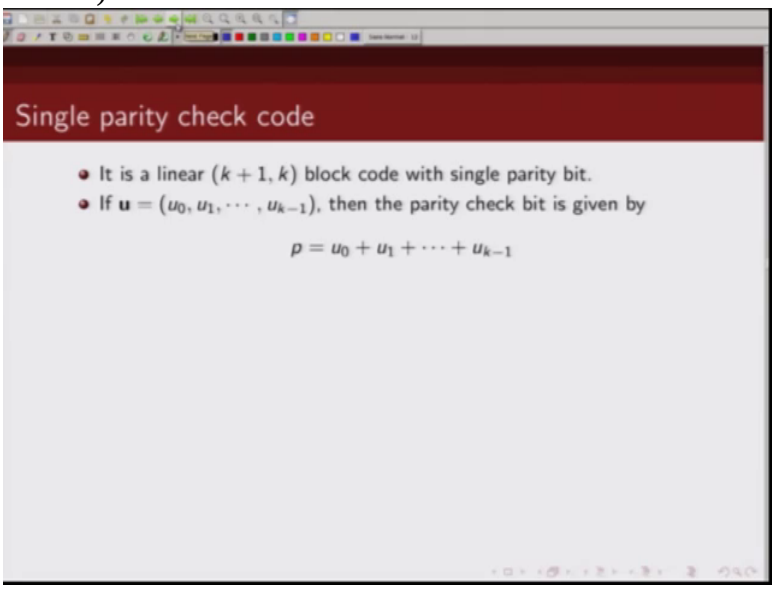
another example of linear block code. This is single parity check code. As the name suggests we are adding one single parity check bit. So the information sequence length is k and n here is given by k plus 1. And how do we generate this

(Refer Slide Time 16:49)



single parity check bit? This is as follows.

(Refer Slide Time 16:53)

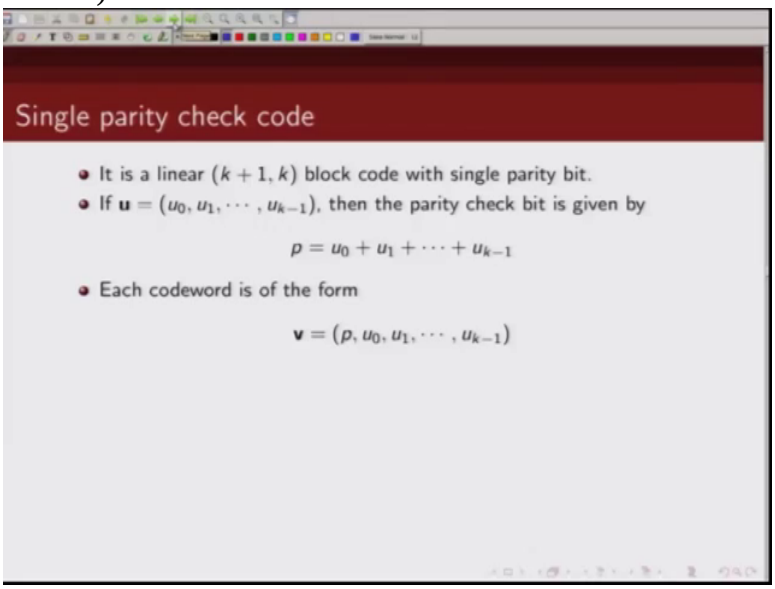


So if your information sequence is given by this, so you have a k-bit information sequence. Let's call it u 0, u 1, u 2, u k minus 1, then we generate this additional parity bit in this fashion. So p is equal to u 0 plus u 1 plus u 2 up to u k minus 1. So in other words, if information sequence is even parity, that means some of them basically add up to zero, then p will be zero or else if the information sequence has odd parity, p will be 1. So as you can make out, this single parity check code will always have even weight

(Refer Slide Time 17:41)

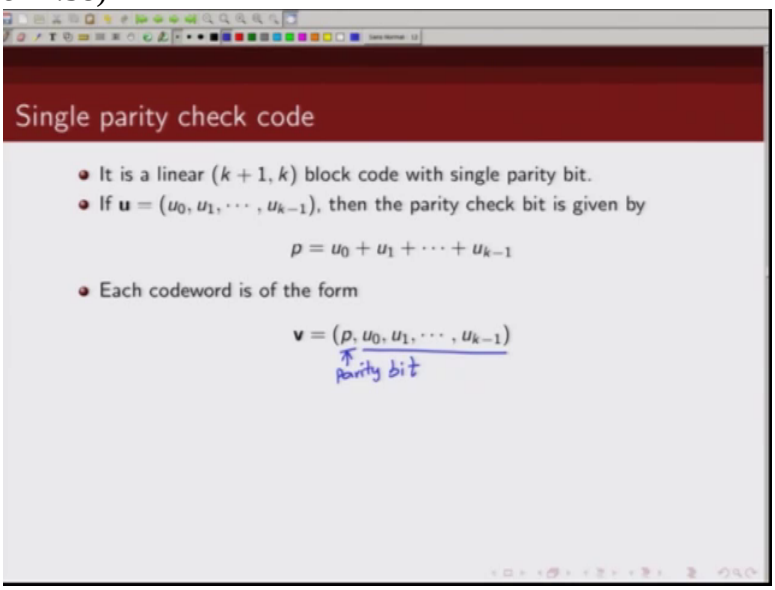
codewords. So each codeword

(Refer Slide Time 17:43)

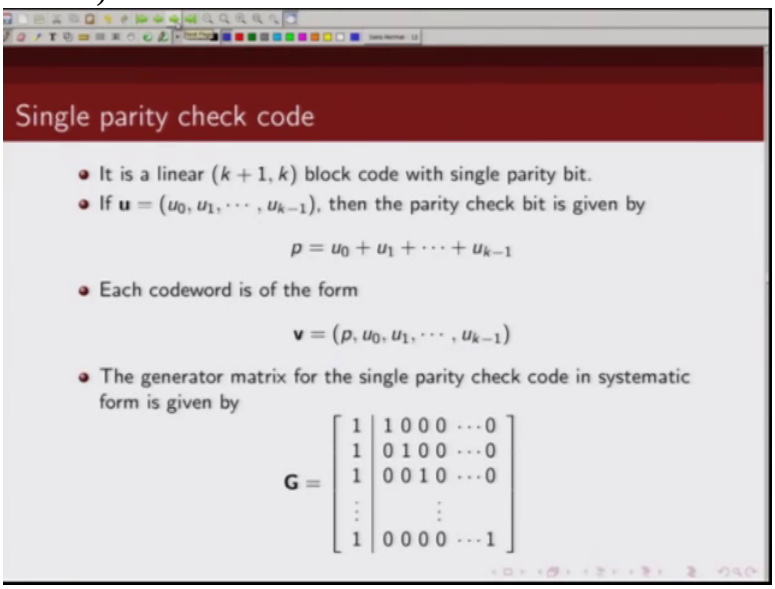


I can write in this particular fashion, so I have these k information bits and one parity bit.

(Refer Slide Time 17:58)

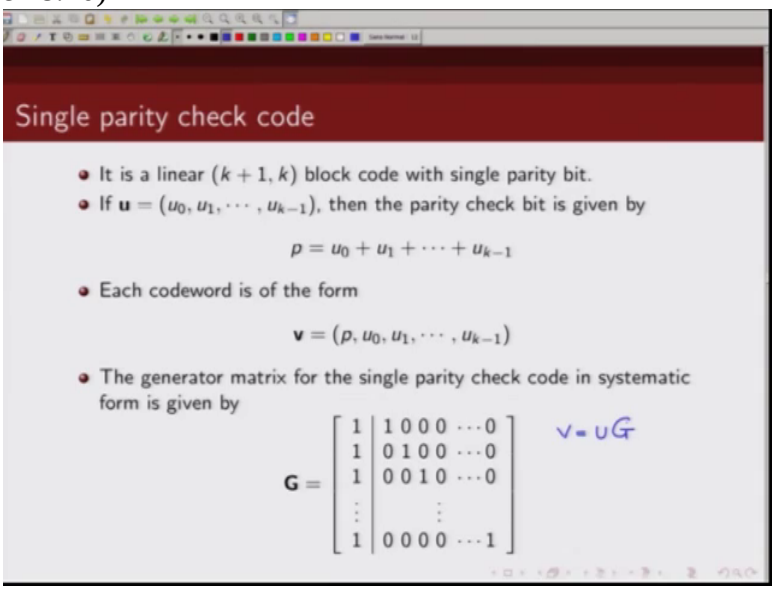


(Refer Slide Time 17:59)



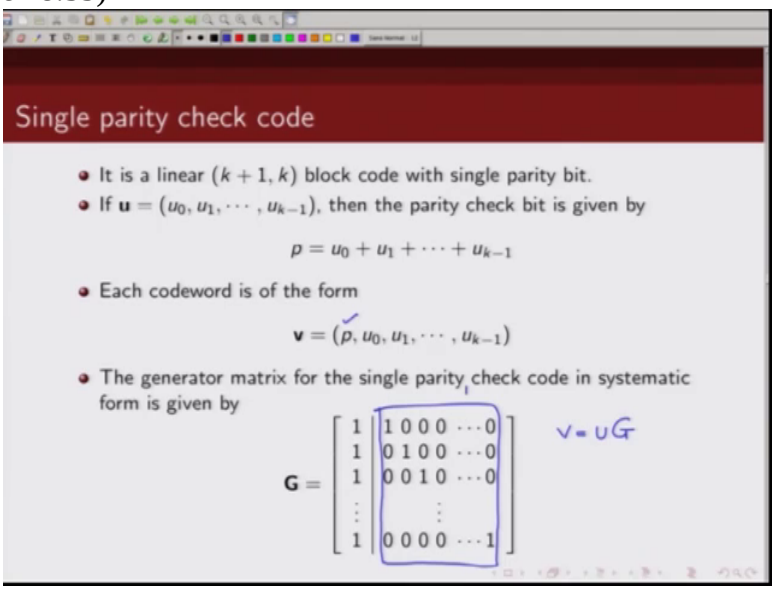
So I can write the same thing in the form of a generator matrix as we know our coded sequence can be written as input times

(Refer Slide Time 18:10)



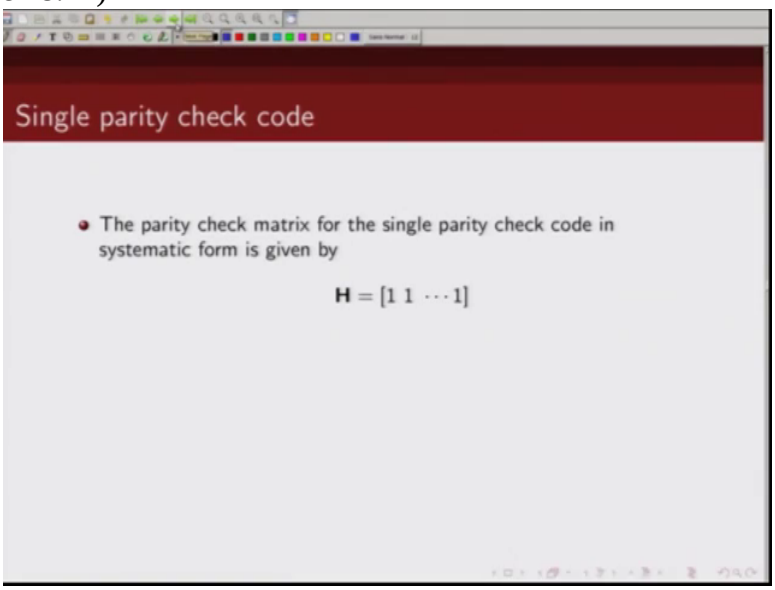
generator matrix. So you can see first bit is a parity bit which is sum of all u i's so first column would be all 1's and then this will be an identity matrix. Because the second bit is u 0, third bit is u 1, u 2 and so on. So this is a identity matrix.

(Refer Slide Time 18:35)



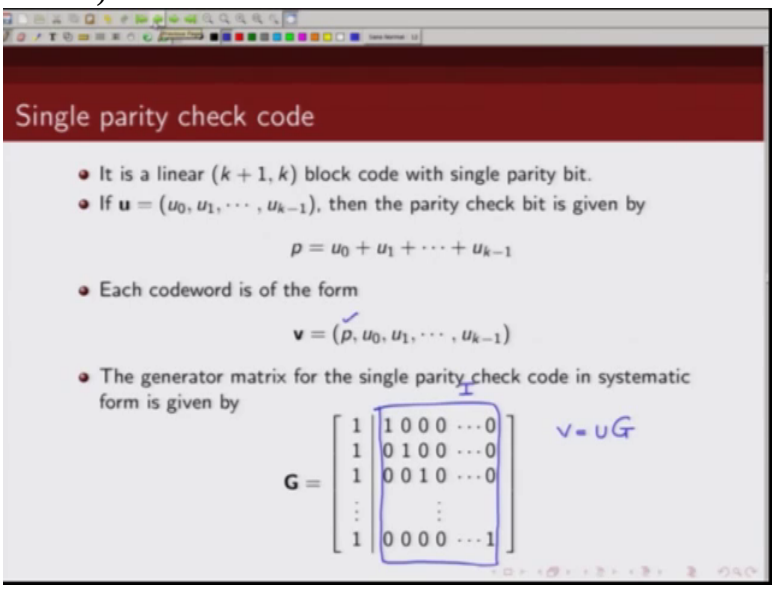
Ok?

(Refer Slide Time 18:41)



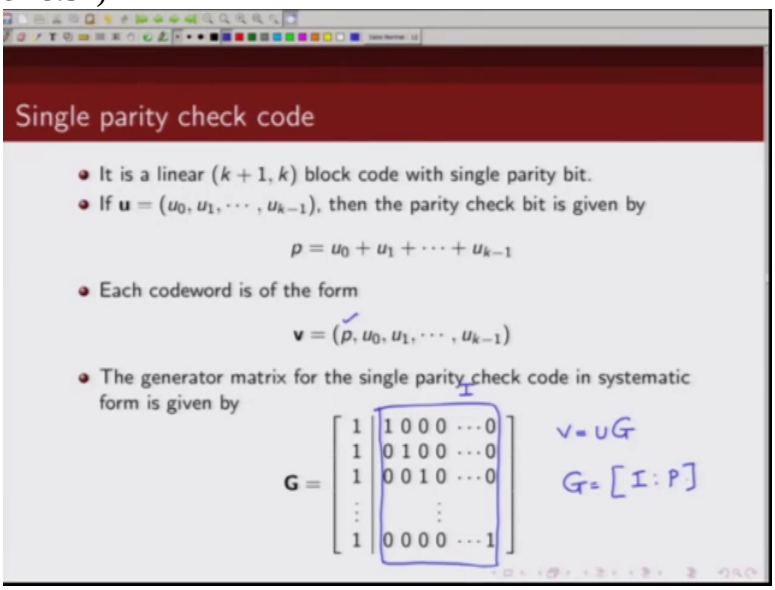
Now we can write down the parity check matrix for this also. As you can see this is,

(Refer Slide Time 18:47)



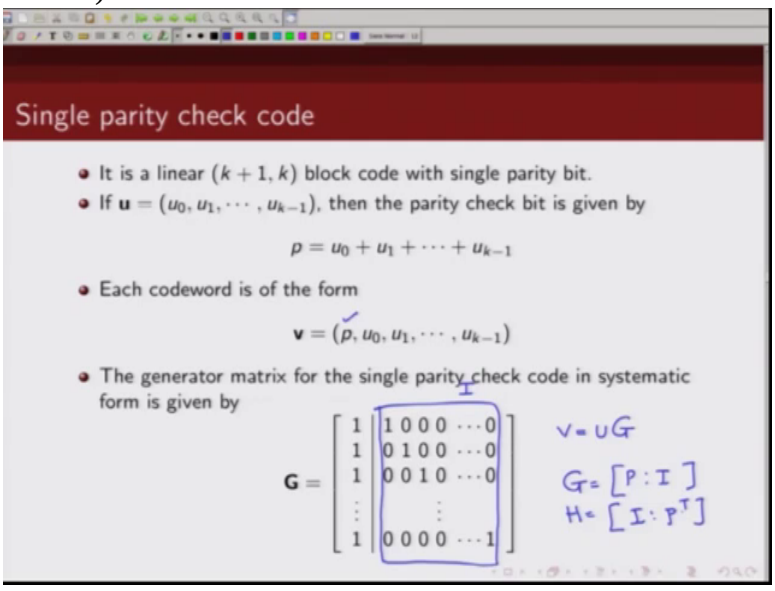
this generator matrix is in systematic form. So what would be

(Refer Slide Time 18:54)



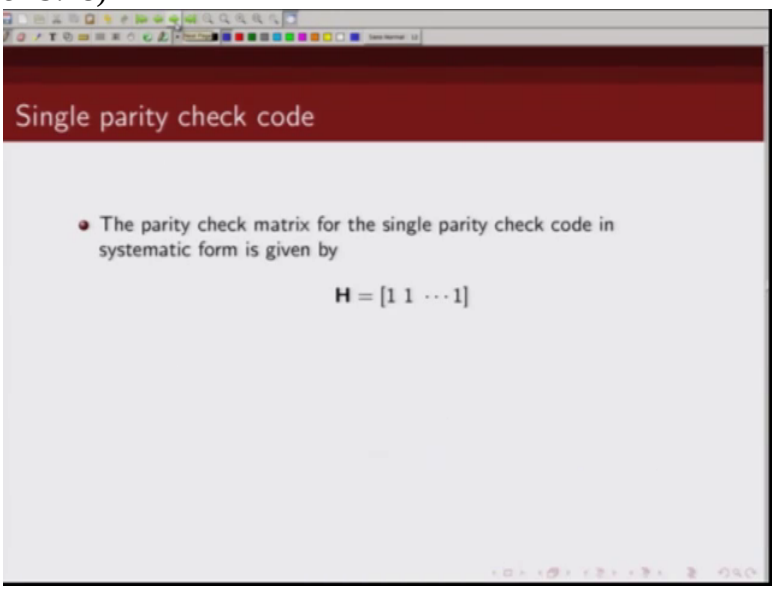
the corresponding, this is of the form like this. I have P I so parity check matrix will be I P transpose,

(Refer Slide Time 19:11)



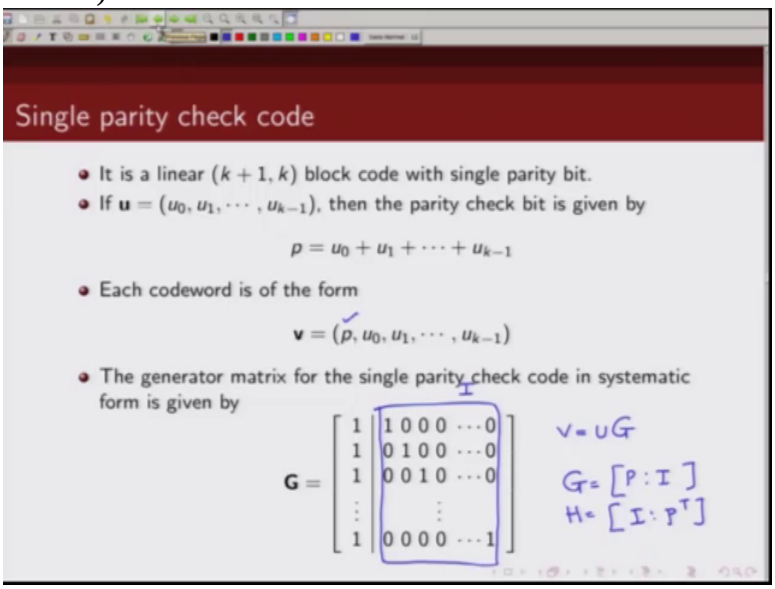
right? So what I have here is

(Refer Slide Time 19:18)



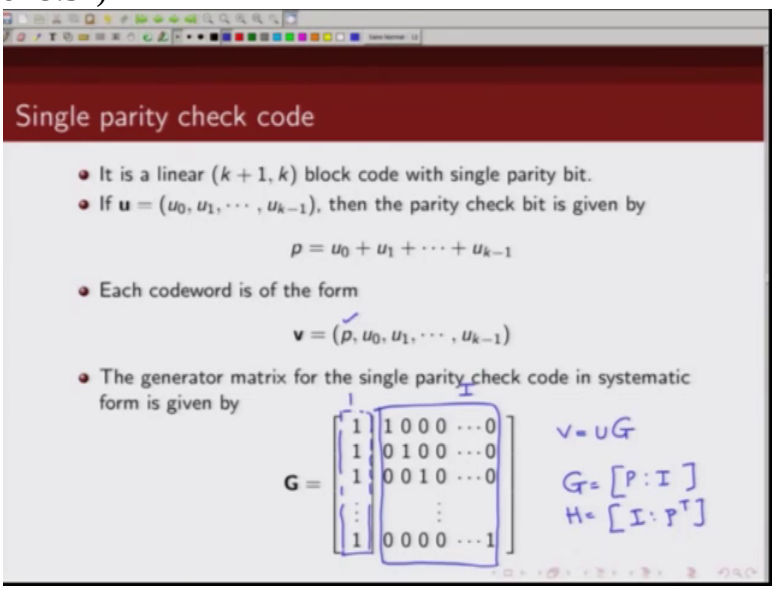
this is my P transpose and this is my I. This

(Refer Slide Time 19:27)



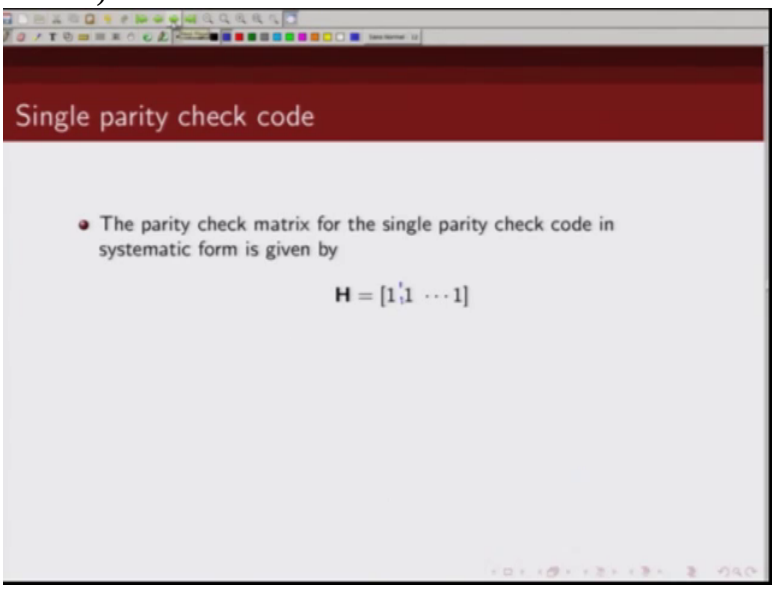
part is my, this part is my P and

(Refer Slide Time 19:34)

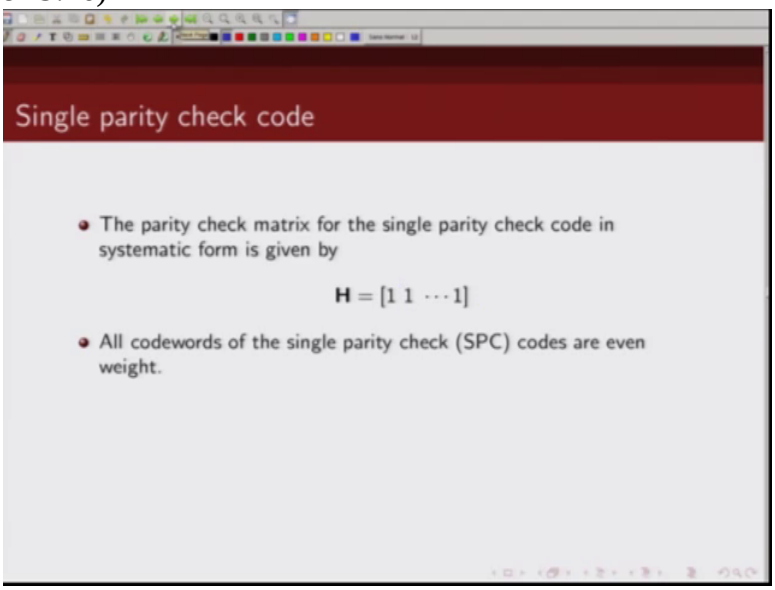


this part is my I. So parity check matrix would be I P transpose that's this.

(Refer Slide Time 19:41)

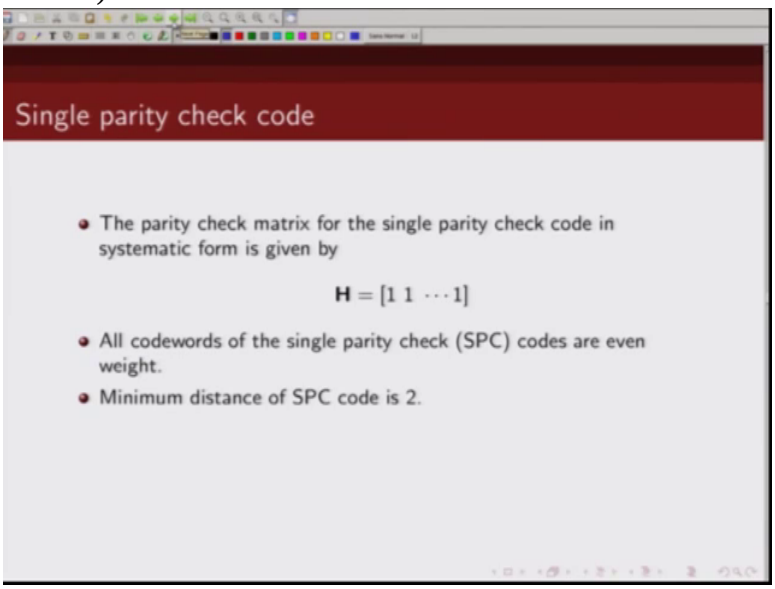


(Refer Slide Time 19:46)



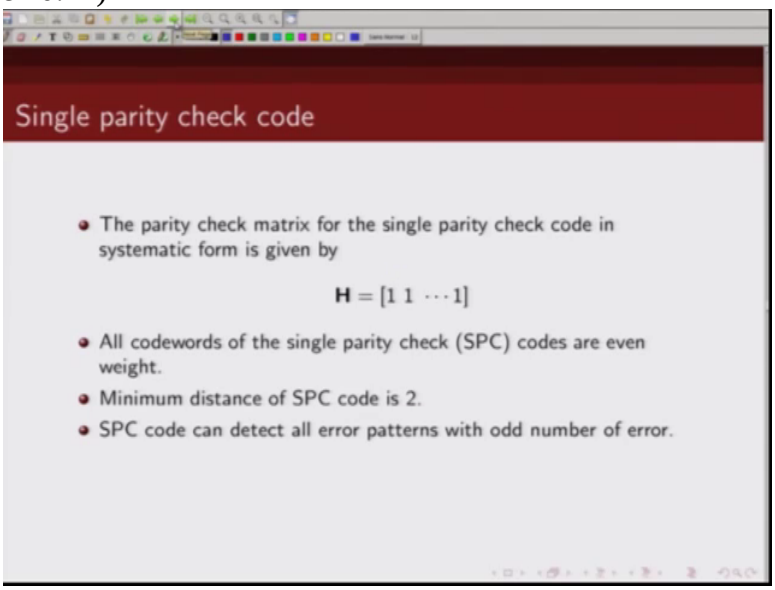
As I mentioned, a very interesting property of single parity check code, all codewords of single parity check codes are of even weight. And

(Refer Slide Time 20:01)



we can see the minimum distance of the code is 2. Simple, very simple way to check is look at the columns of the parity check matrix? What is the minimum number of columns that add up to zero? In this case 2. Any two columns will add up to zero. So the minimum distance of this code is 2.

(Refer Slide Time 20:24)

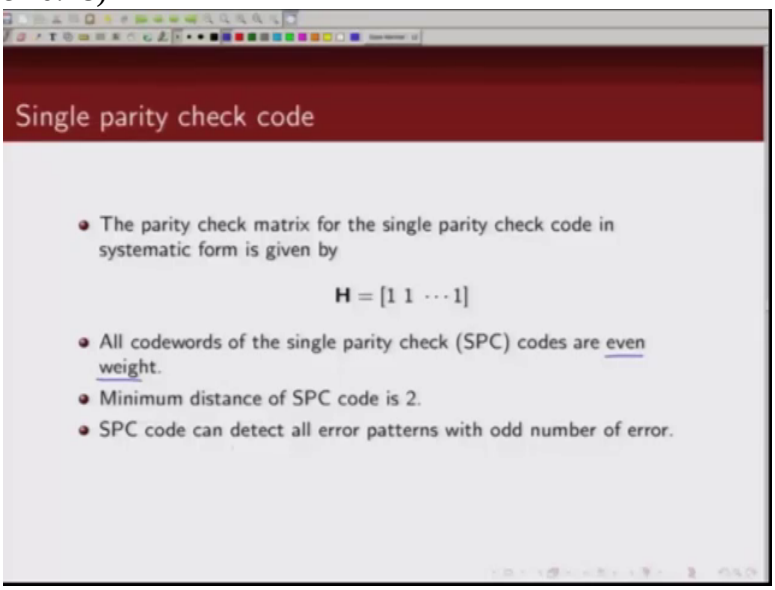


Now since all codewords have even weight any odd code, any odd error pattern can be determined by single parity check code, because any odd

(Refer Slide Time 20:40)

pattern, any odd weight pattern is not a valid codeword. So a single parity check

(Refer Slide Time 20:48)

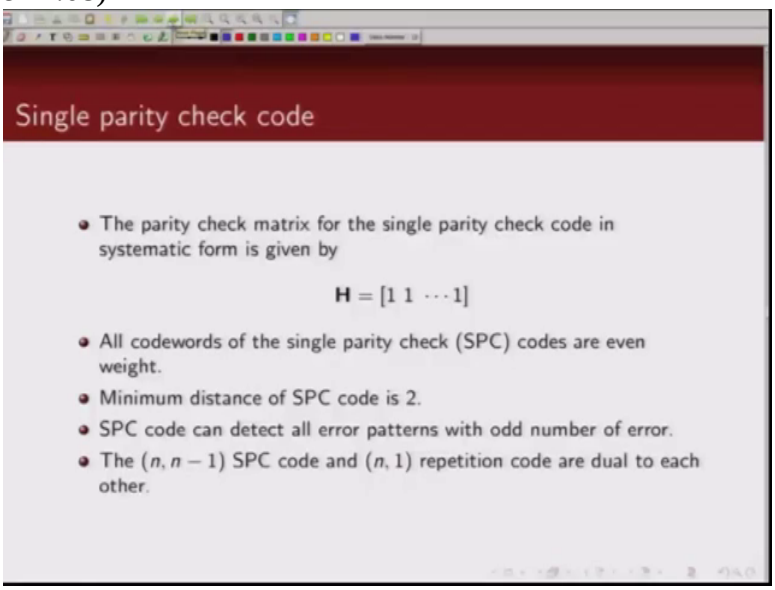


code can detect all error patterns which have even numbers of errors, odd numbers of errors, right? So as long as error pattern has odd weight,

(Refer Slide Time 21:02)

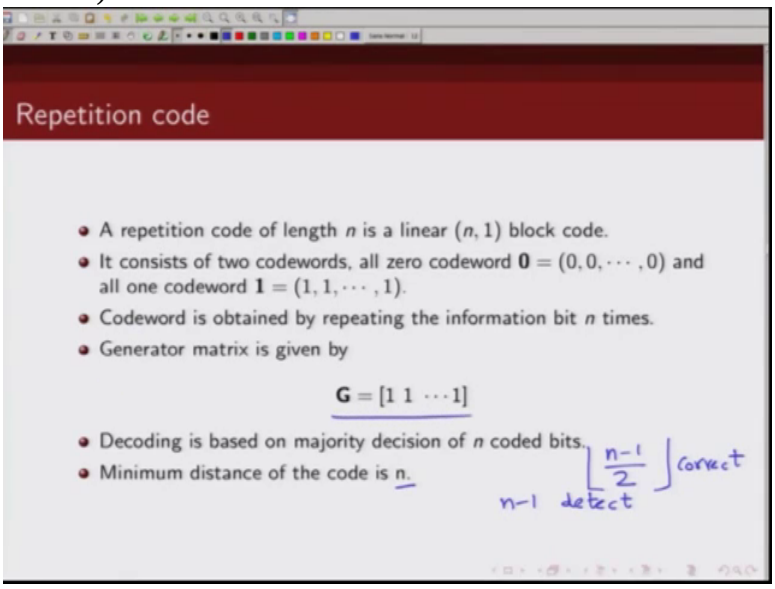
single parity check code can detect it. And it is not

(Refer Slide Time 21:08)



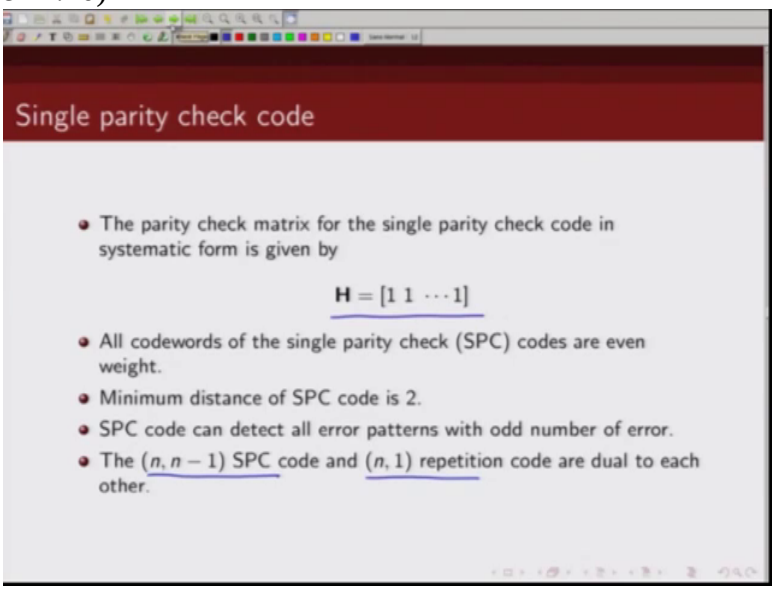
very difficult to see that single parity check code and repetition codes are dual to each other. You can see basically uh, the parity check matrix of single parity check code is same as generator matrix of the repetition

(Refer Slide Time 21:29)



code. And similarly the parity check matrix of uh, repetition code is same as the generator matrix of the single parity check code and you can check the dimensions of the single parity check

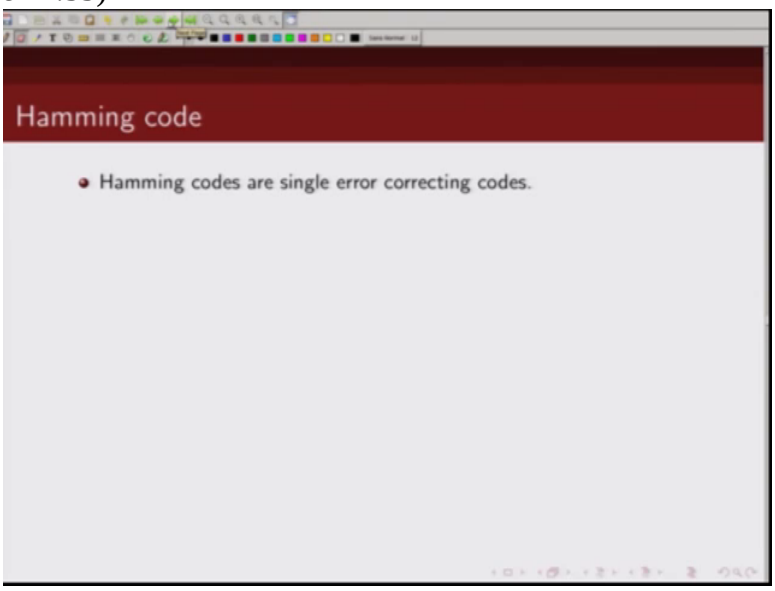
(Refer Slide Time 21:40)



code is n minus 1 and this dimension is 1. So single parity check codes and repetition code are dual to each other.

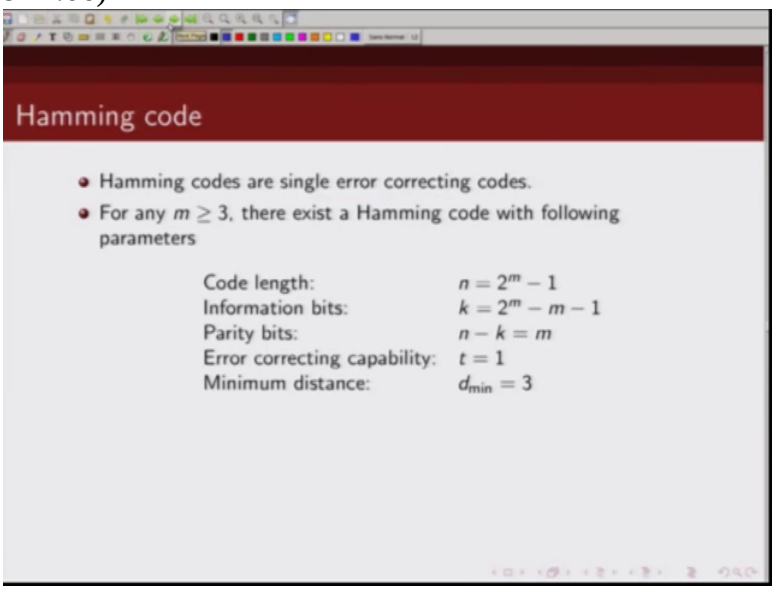
Next we consider

(Refer Slide Time 21:55)



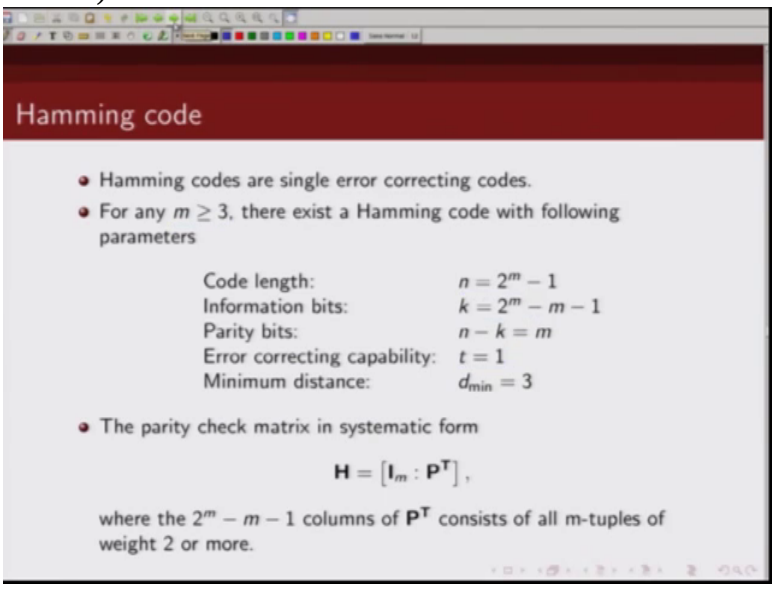
Hamming codes. So Hamming codes are single error correcting codes. These are single error correcting codes and these are described by

(Refer Slide Time 22:06)



these parameters. So for any m greater than equal to 3, the codeword length is given by this. The number of parity bits is equal to m. So the number of information bits are n minus k, n minus m so this is 2 to power n minus m minus 1. So these many number of information bits. It has minimum distance of 3. So it can correct single error and it can detect 2 errors. So how do we describe Hamming code? We can describe

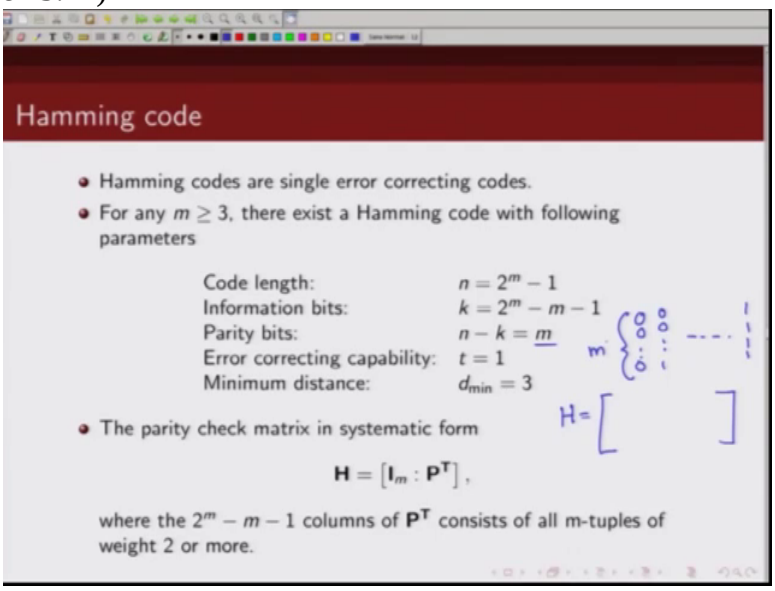
(Refer Slide Time 22:48)



the Hamming code by generator matrix or the parity check matrix. So a parity check matrix of a Hamming code consists of all non-zero m tuples. So if the number of parity bits is, as I said is, m, then codeword length is 2 raised to power m minus 1 and the entries in the parity check matrix are all non-zero m tuples. Now how many m-tuples do we have? We have total

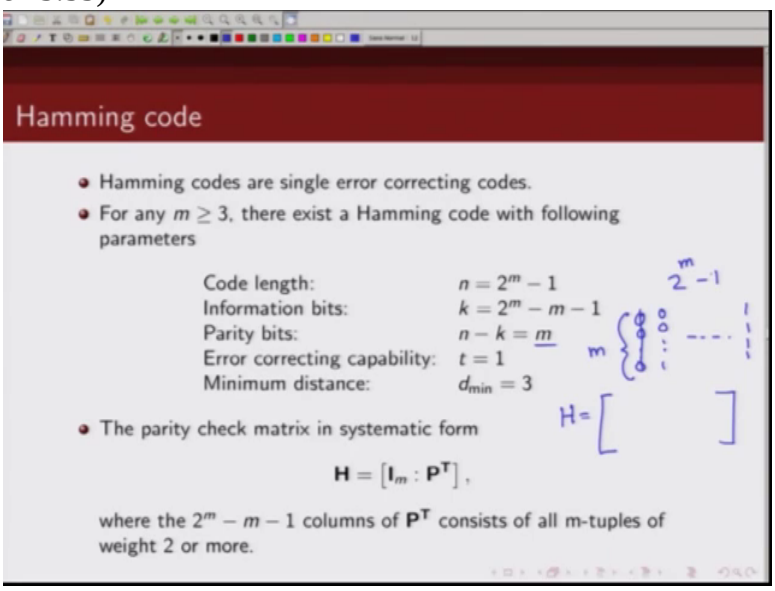
2 raised to power m m-tuples and out of those, so these are like 0 0 0, 0 0 1, you can go on up to all 1s. And this is m times, m. Now if we

(Refer Slide Time 23:44)



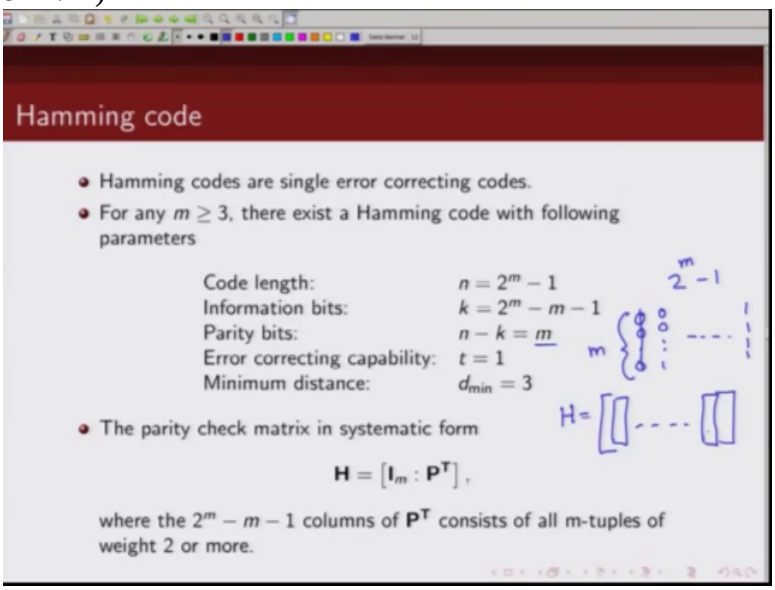
remove the all zero one, so total number of non-zero m-tuples is basically given by 2 raised to power m minus 1. So the

(Refer Slide Time 23:55)

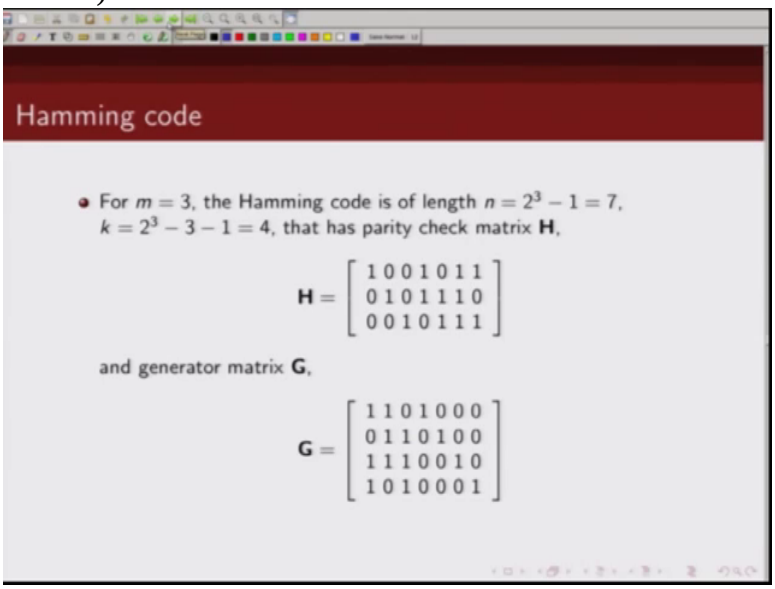


columns of the parity check matrix of Hamming code are nothing but non-zero m-tuples. So there are total 2 m of them, 2 m minus 1 of them,

(Refer Slide Time 24:11)



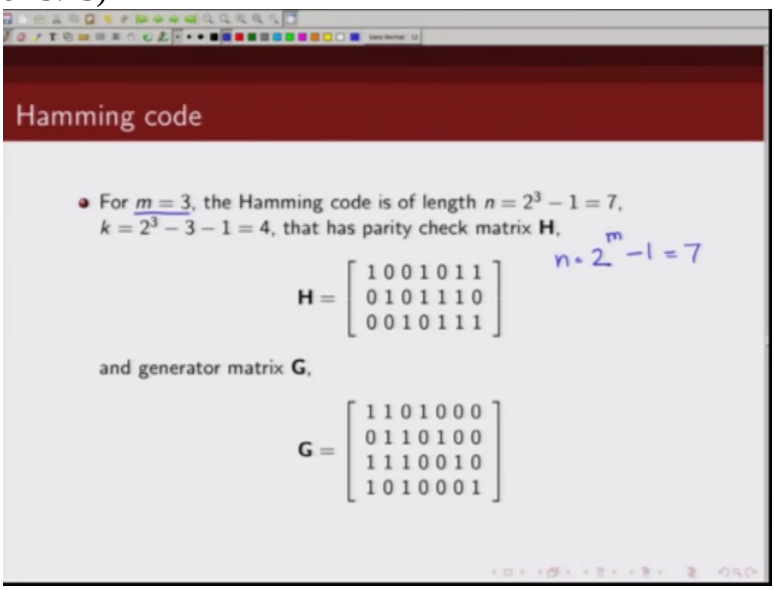
(Refer Slide Time 24:55)



Ok. And if you want to write a parity check matrix of a Hamming code in systematic form, then we can write it in this way. So parity check matrix will have one identity and then some matrix p since identity matrix will consist of all patterns, all m-tuples of weight 1, p t should consist of all m-tuples of weight more than 1. So it consists of all m-tuples of weight 2, 3, 4 up to m.

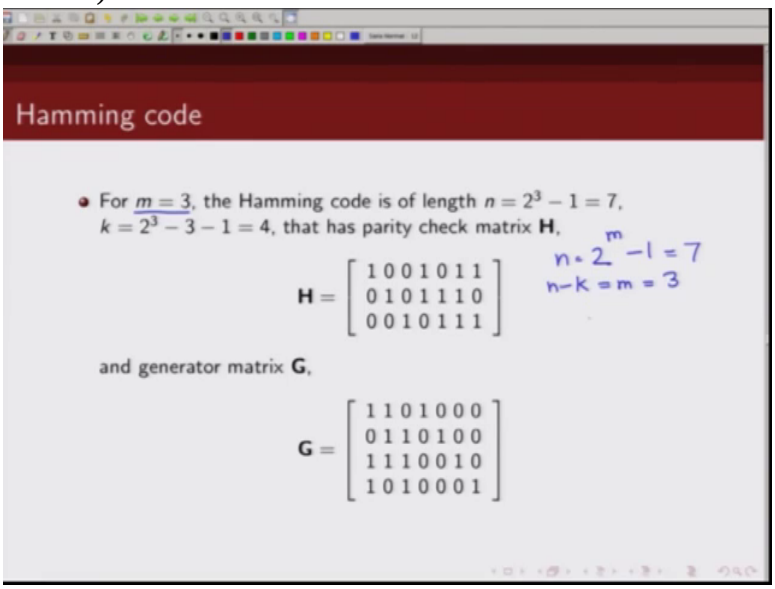
So let us take an example. Let us consider m equal to 3. So if m equal to 3, we know m is 2 raised to the power m minus 1. So the length of the block code in this case will be 7.

(Refer Slide Time 25:13)



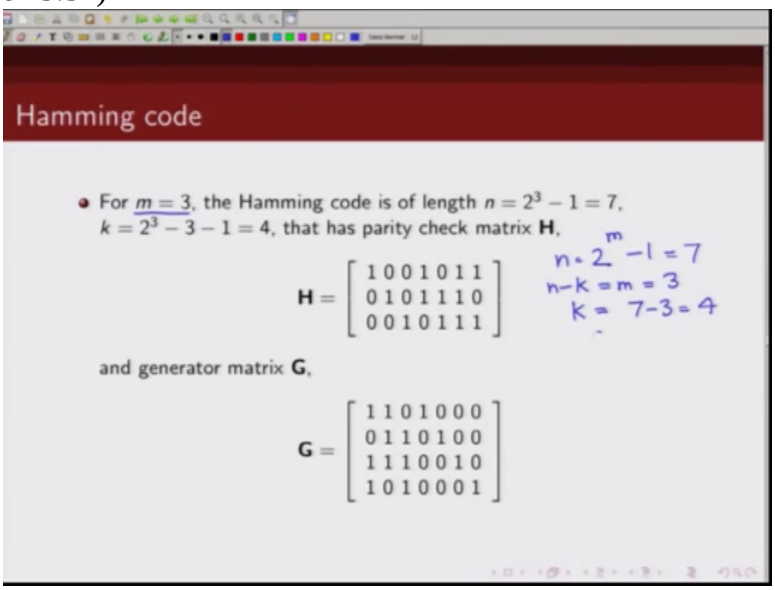
Now how many number of parity bits? Number of parity bits is given by m. So this will be 3. So then

(Refer Slide Time 25:23)



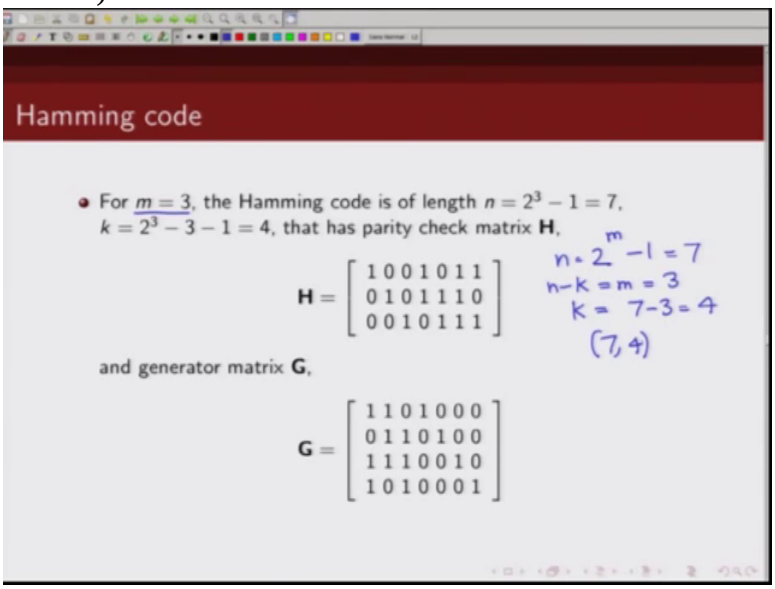
what is the number of information bits? It is 7 minus 3 that is 4. So this is a

(Refer Slide Time 25:31)



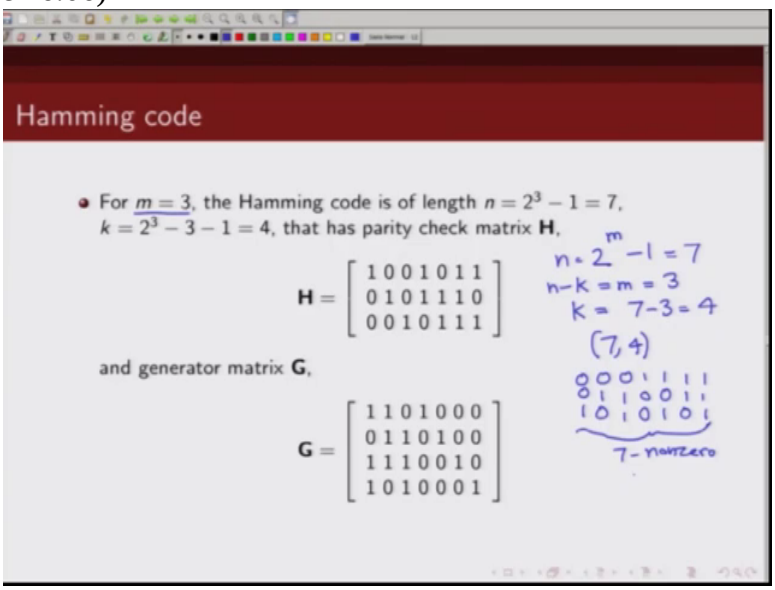
7 4 code. Now as I said,

(Refer Slide Time 25:35)



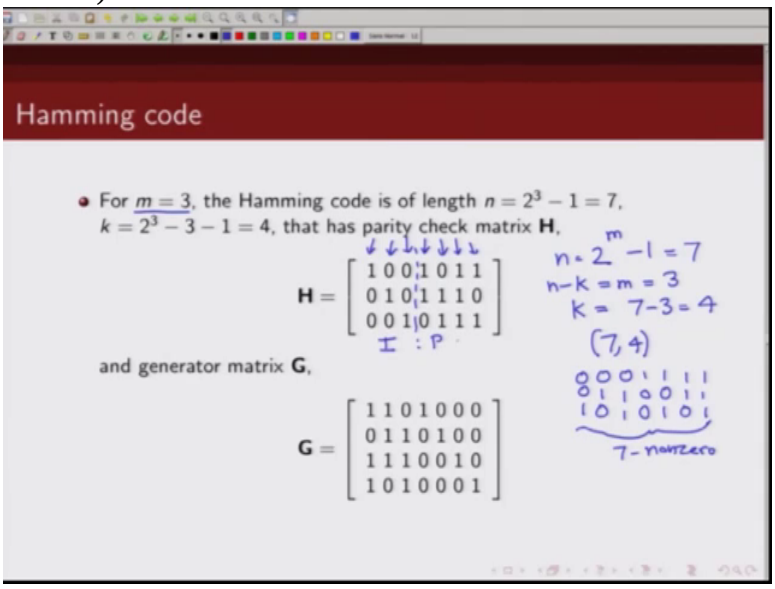
the parity check matrix consists of all non-zero m-tuples. So m here is 3. So let's list all non-zero m-tuples. So we can write 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1. So these are the seven non-zero m tuples,

(Refer Slide Time 26:06)



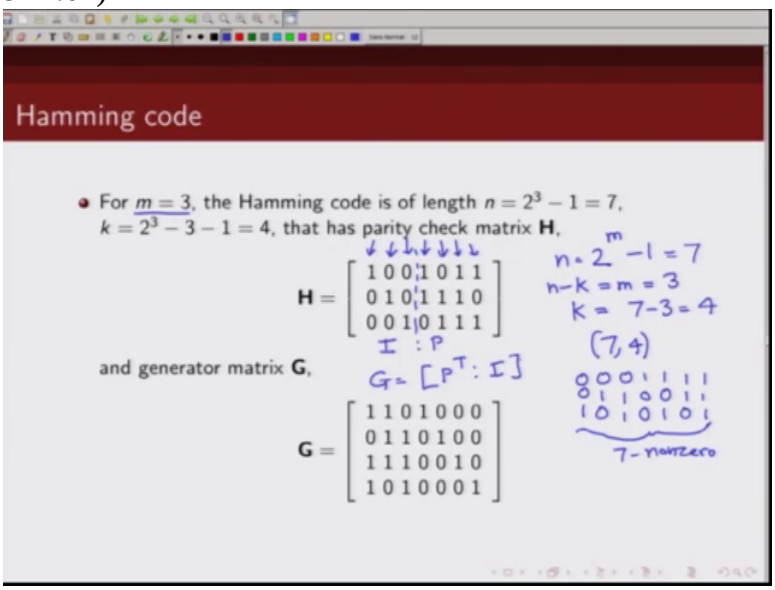
right and if you want to write your parity check matrix in a systematic form, what will you do? So this is your i, so I am writing these patterns which are like 1, 2 and 3, I am writing these patterns and then what I write here is the other m-tuples. So you can see this is m tuple of weight 1, these three are m-tuple of weight 1 and then this is m-tuple of weight 2, and this is m-tuple of weight 3. Now since this matrix is of the form I and P, I can

(Refer Slide Time 26:54)



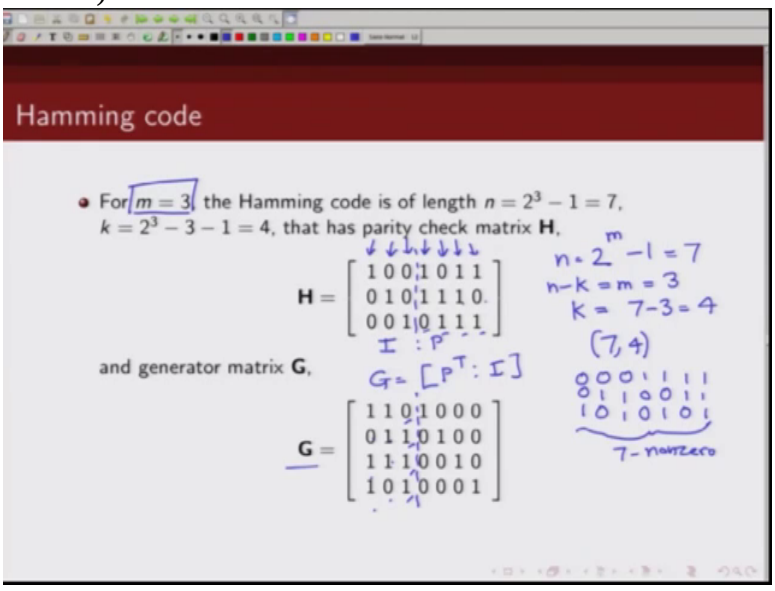
write the generator matrix. This will be P transpose I. So

(Refer Slide Time 27:01)



we can write this, so P transpose, so 1 0 1 1 will come here as 1 0 1 1. 1 1 1 0 will come as 1 1 1 0. And 0 1 1 1 will come as 0 1 1 1. And then this is a identity matrix. So this is our generator matrix for this 7 4 Hamming code. The point to be noted here is, once you fix your m, all

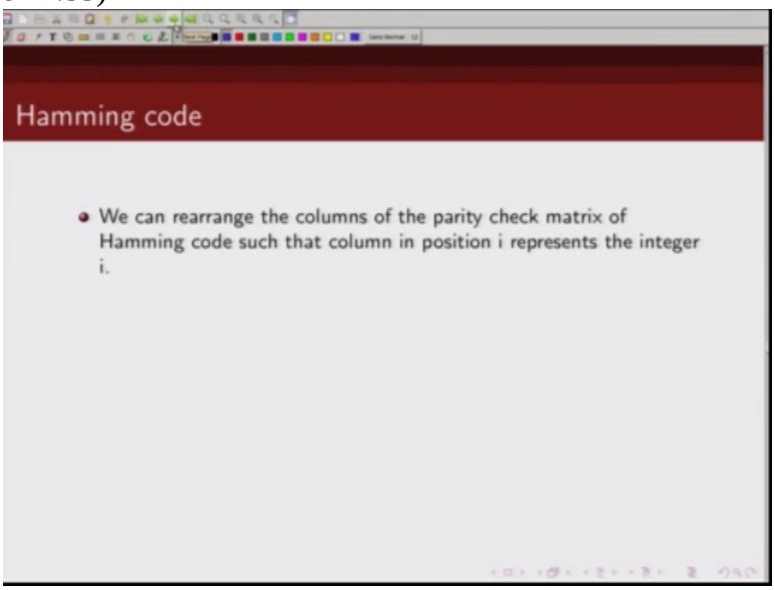
(Refer Slide Time 27:32)



other code parameters are fixed. n is fixed, right because n is 2 raised to power m minus 1. And number of parity bits is equal to m. For the Hamming code, once you fix m, the other parameters of the code are fixed.

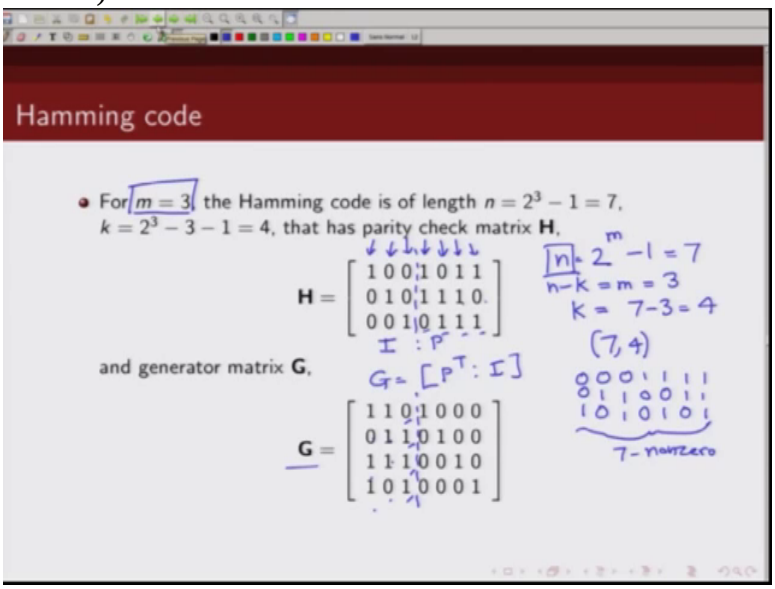
Now let us

(Refer Slide Time 27:53)



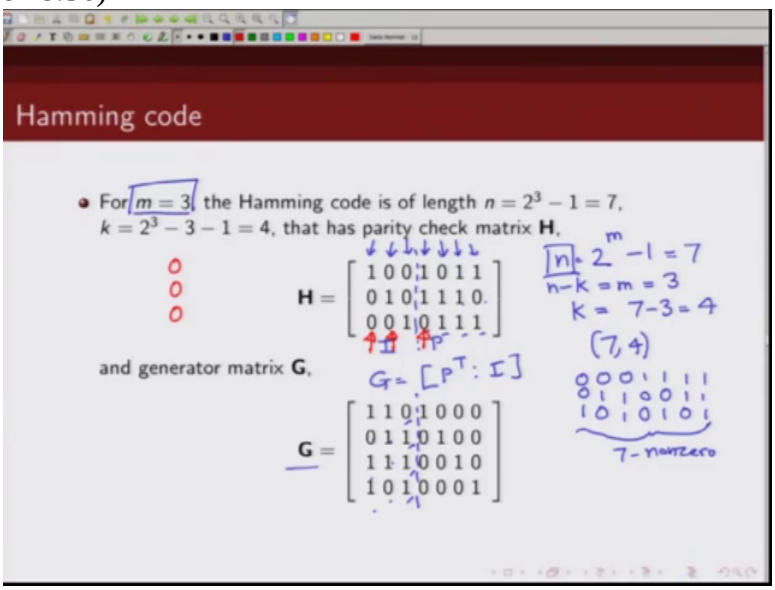
see how we can correct errors using Hamming code. As we said this is,

(Refer Slide Time 28:01)



this has minimum distance 3. You can see here. So let's take any three columns. Let's just take this column; column number 1, column number 2 and column number 4. You can see column number 1, 2 and 4, they will add up to 0 0 0, that is the minimum number of columns that will add up to zero,

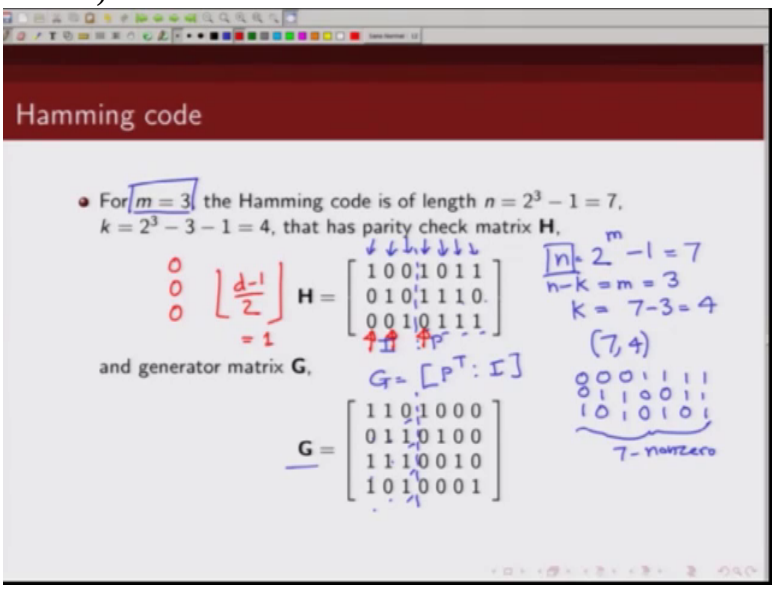
(Refer Slide Time 28:36)



so which means the minimum distance of this code is 3. If the minimum distance of this code is 3, what is the error correcting capability of this code? It is d minus 1 by 2, floor of that and this comes out to be 1. So Hamming code can correct single error.

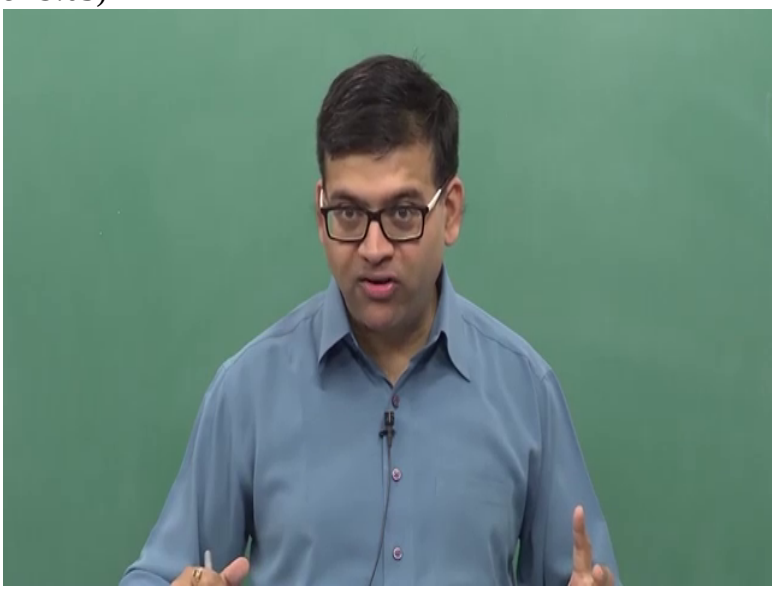
So let's see how we can use

(Refer Slide Time 29:01)



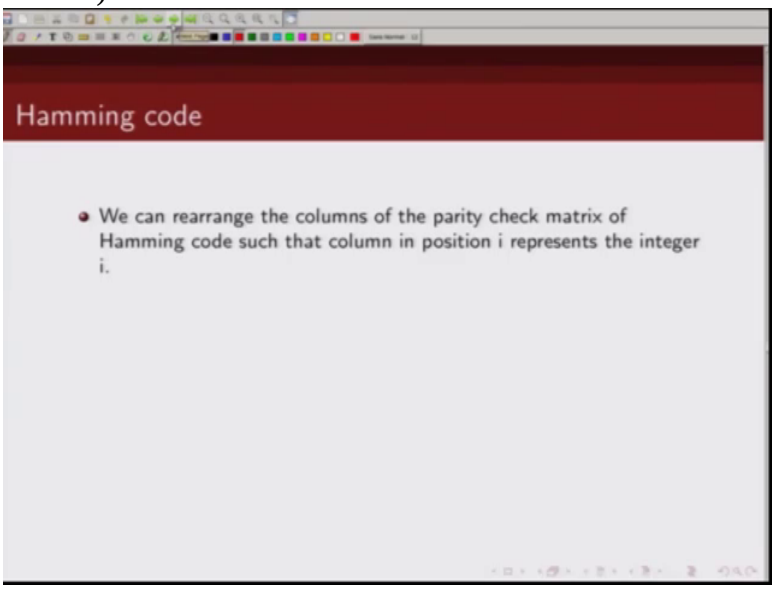
Hamming codes to

(Refer Slide Time 29:03)



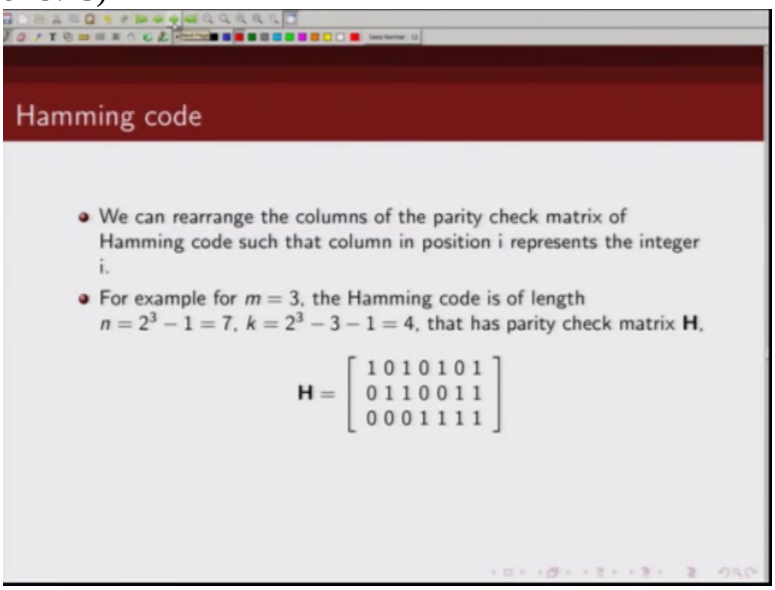
correct single error. So first thing what we will

(Refer Slide Time 29:09)



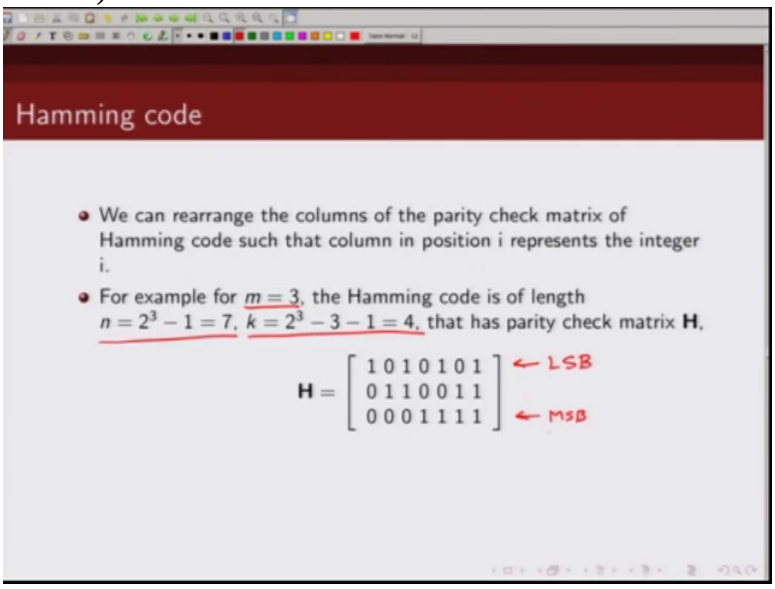
do is, we will rearrange the columns of parity check matrix of Hamming code such that column in position i represent interior i.

(Refer Slide Time 29:25)



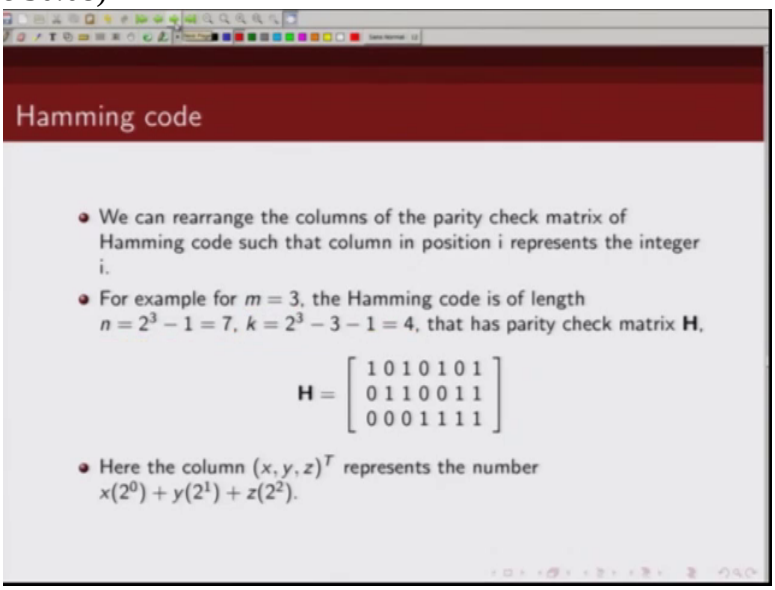
So as we said, the parity check matrix of Hamming code are all non-zero m-tuples. So we will arrange them in such a way such that the ith column represent ith bit. Now what do I mean by that? So let's go back to our same example m equal to 3, so in this case n is equal to 7 and k is equal to 4. Now note the way I have arranged this? So this is my, this is my M S B and this is my L S B, least significant bit and the

(Refer Slide Time 30:04)



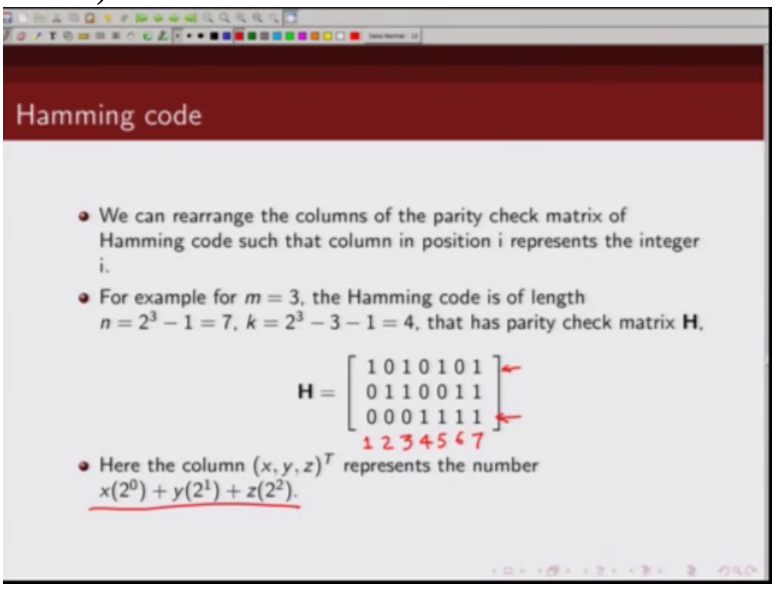
most significant bit. So

(Refer Slide Time 30:08)



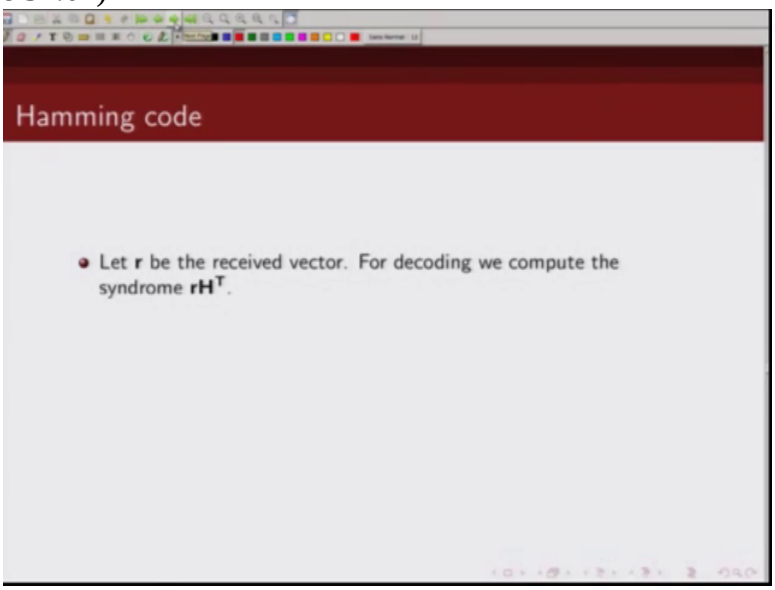
each column represents a number in this particular range. So this is the least significant, this is least significant bit; this is the most significant bit. So this is 0 0 1, that's binary 1, so this is my column number 1. This is 0 1 0, that's 2. 0 1 1, that's 3. This is 4, 5, 6, 7. Note what I

(Refer Slide Time 30:37)



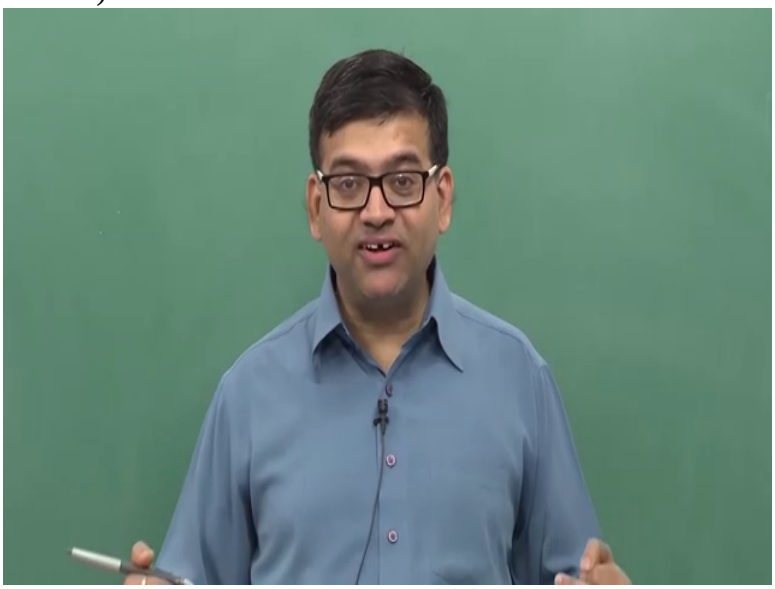
did was I rearranged a column of the parity check matrix of Hamming code such that now ith column represents ith, uh number basically, so column in ith position represents the integer i. Now if I do that then

(Refer Slide Time 31:02)



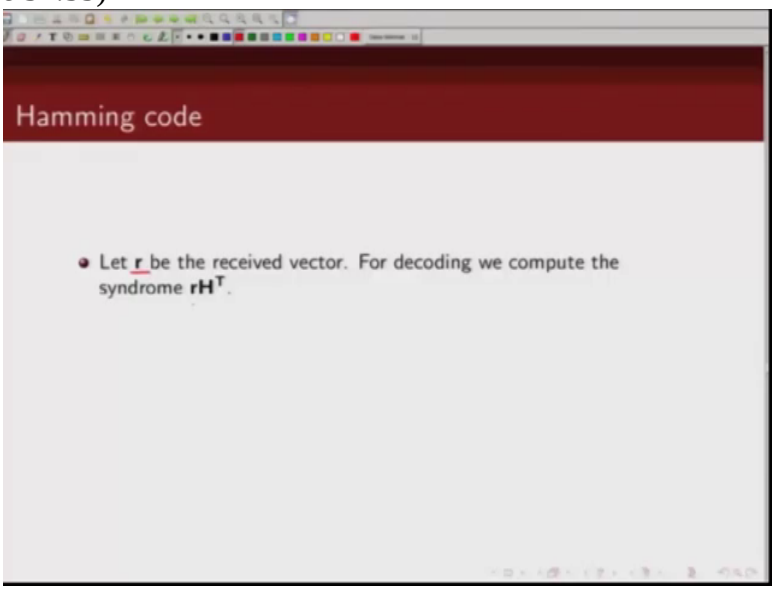
let's see how this helps us in locating the position of the error. So let's say r is my received vector. This is my received n bit vector and I am interested in knowing whether r is in error and if r is in error, because it's a single error correcting code, let us say a single error has happened then I am interested in

(Refer Slide Time 31:28)



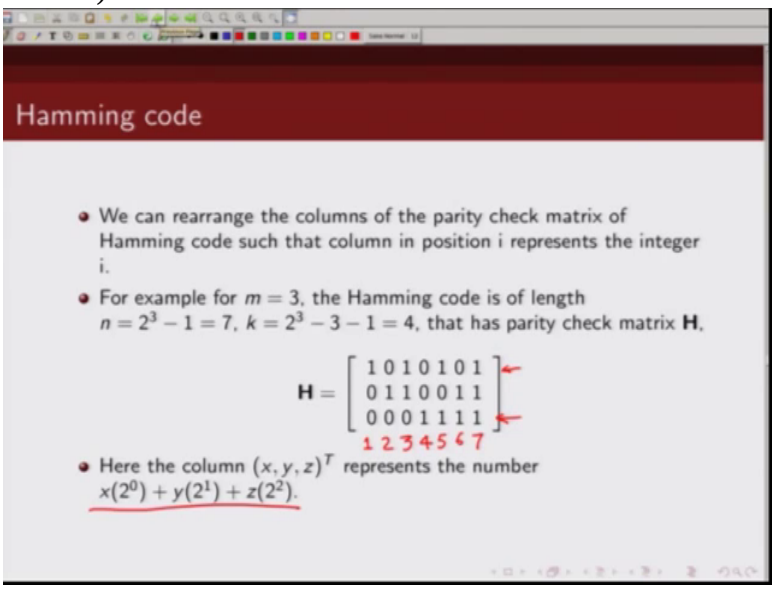
correcting this single error. So what do we do? The first thing to find out whether

(Refer Slide Time 31:35)



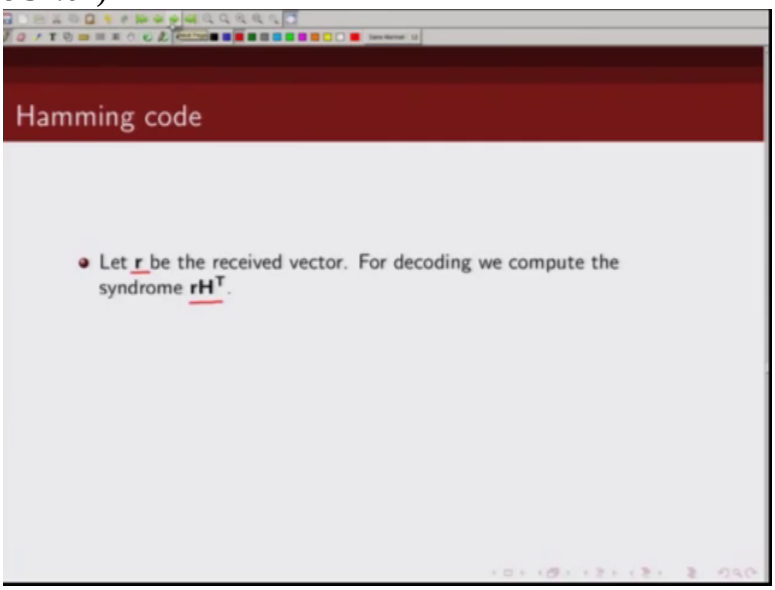
error has occurred or not, first thing that we do is we compute the syndrome. How do you compute the syndrome? We take r H transpose and what is our H? H is

(Refer Slide Time 31:49)

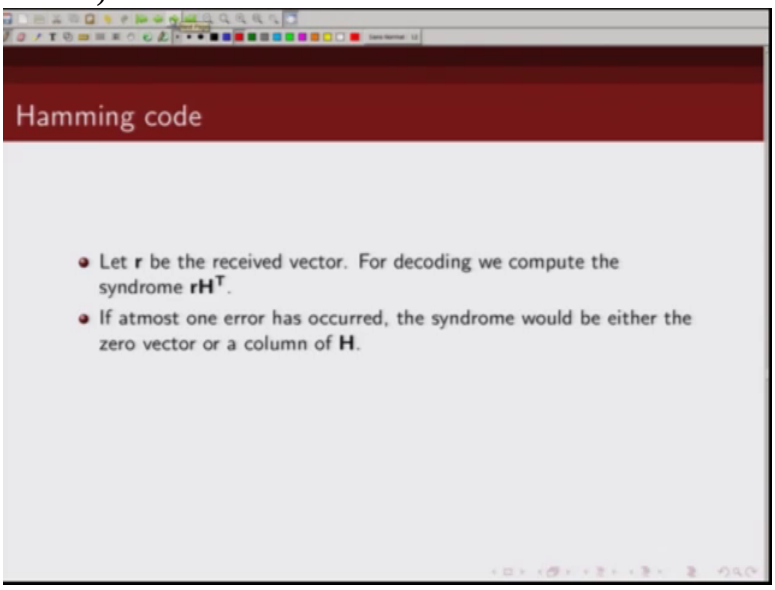


our matrix, this matrix which is arranged, where the columns are arranged in such a way such that column in ith position represents integer i.

(Refer Slide Time 32:02)

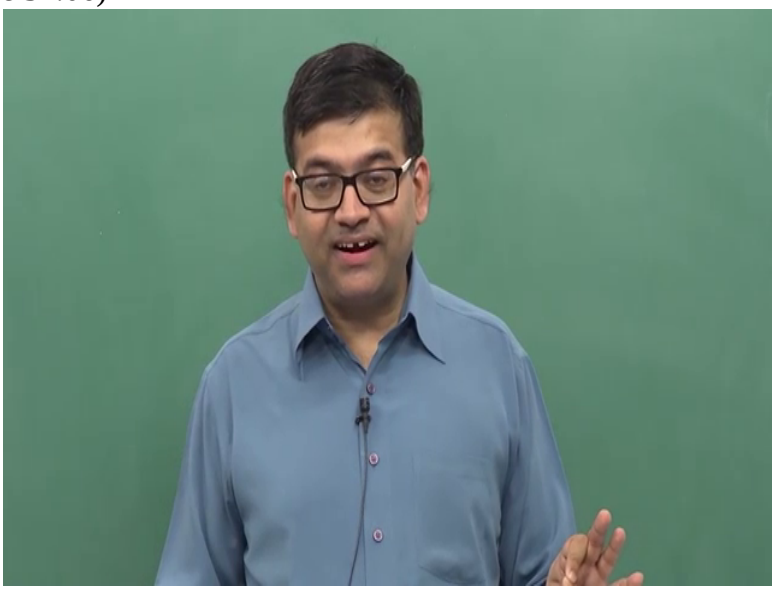


(Refer Slide Time 32:04)



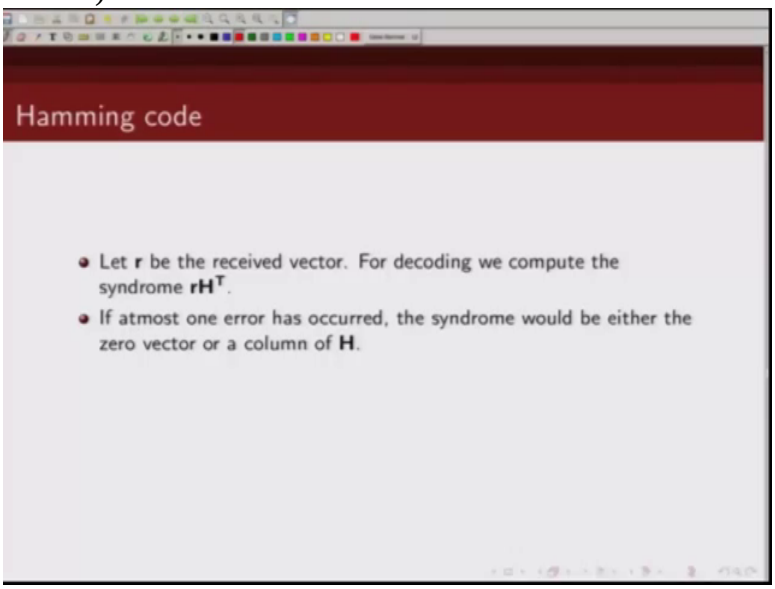
So, since it is single error correcting code,

(Refer Slide Time 32:06)



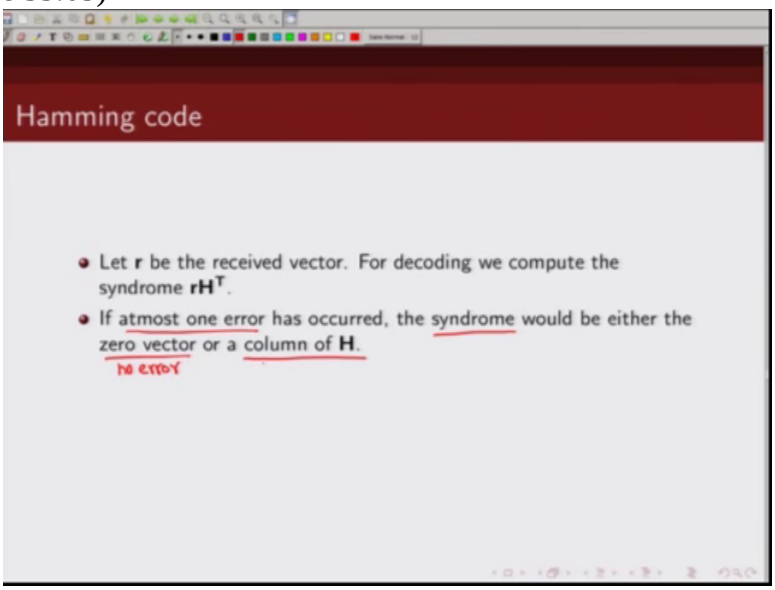
it can only correct at most one error. So we assume that say, at most

(Refer Slide Time 32:11)



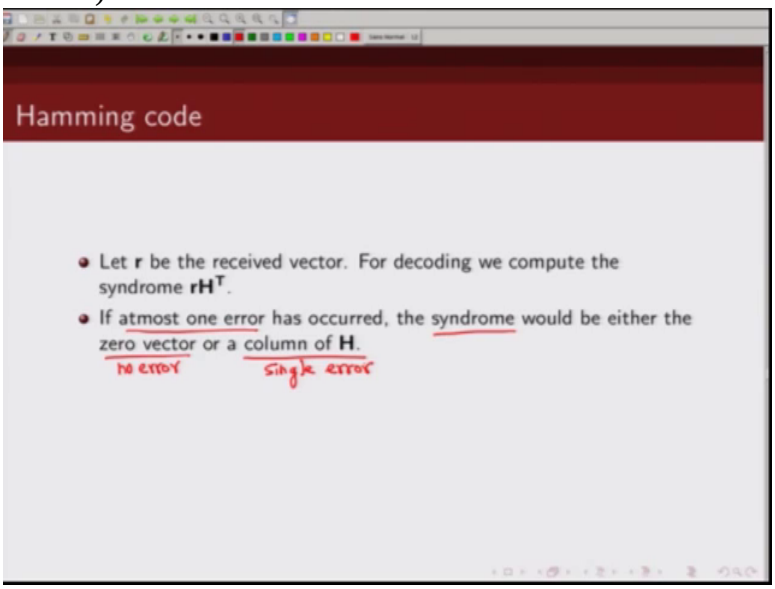
one error has occurred, at most. Of course if there is no error, then syndrome would be an all-zero vector and if a single error has happened, the syndrome would have been a non-zero vector. Now what you will observe is, if a single error happens then that single error will be, whatever is the syndrome, from there we can find out which bit location is in error. So in case of Hamming code, if single error has happened and you have arranged your parity check matrix in such a way such that column in position i represent integer i then the syndrome would be either zero vector in case there is no error or otherwise in case of single error it would be

(Refer Slide Time 33:05)



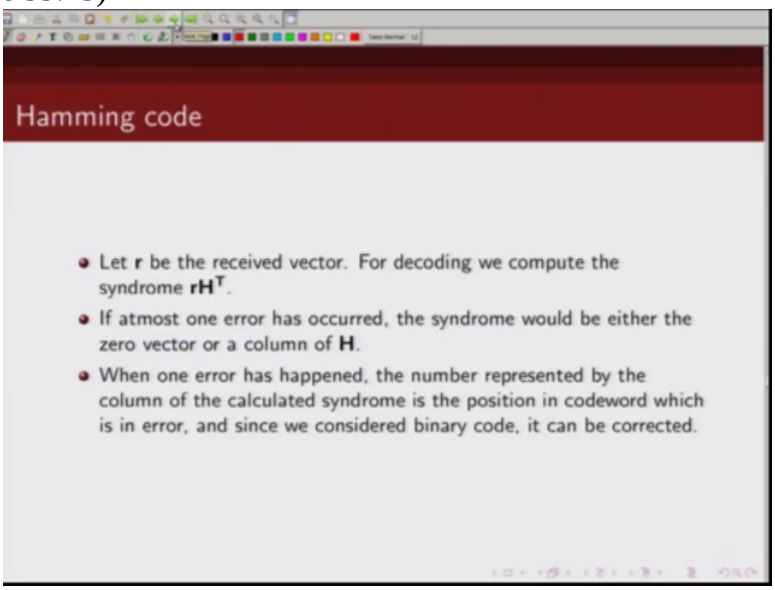
column of this parity check matrix.

(Refer Slide Time 33:11)



So

(Refer Slide Time 33:13)

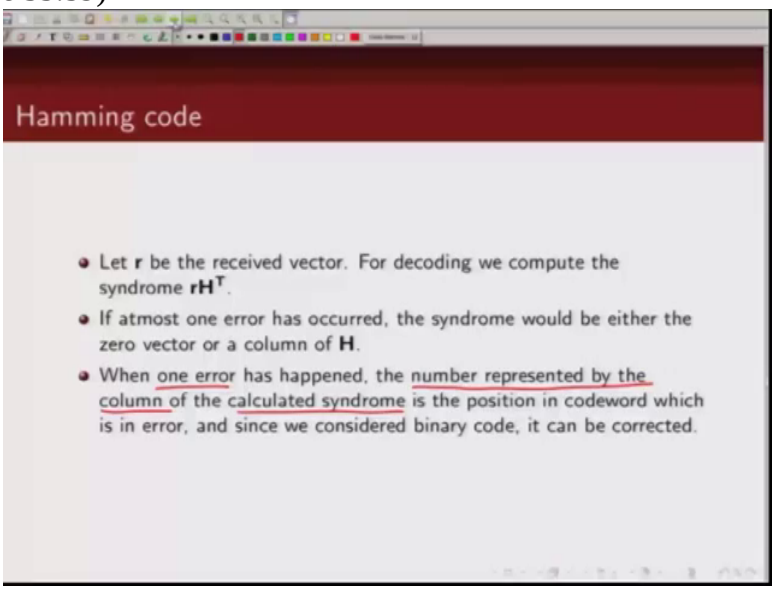


So when a single error happens, the number represented by the column of the calculated syndrome is the position in the codeword where the error has happened. So see, so nice, so by looking at the syndrome

(Refer Slide Time 33:34)

and comparing it with the column of the parity check matrix, our reordered parity check matrix, we can find out the location where the error has happened. And since for the binary code basically if we can locate the error, so the bit can either be 0 and 1, so whatever the bit is, we just flip that bit and get our

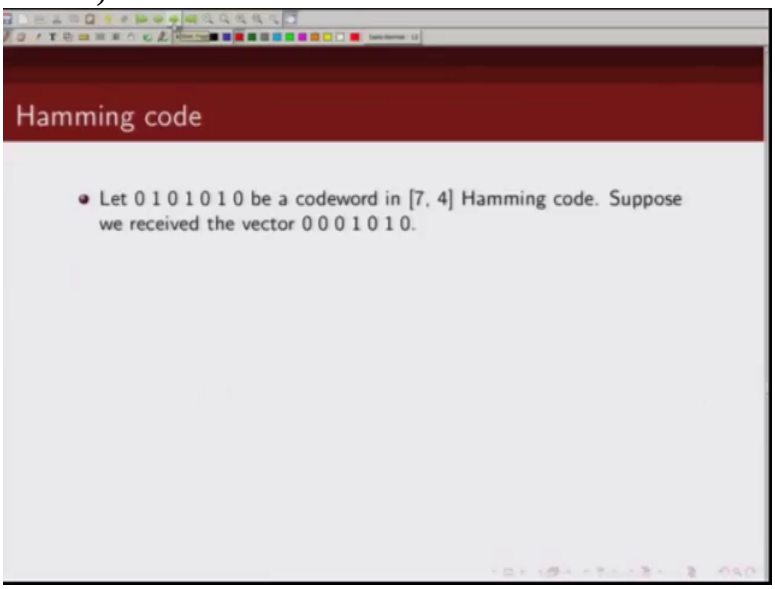
(Refer Slide Time 33:59)



corrected codeword.

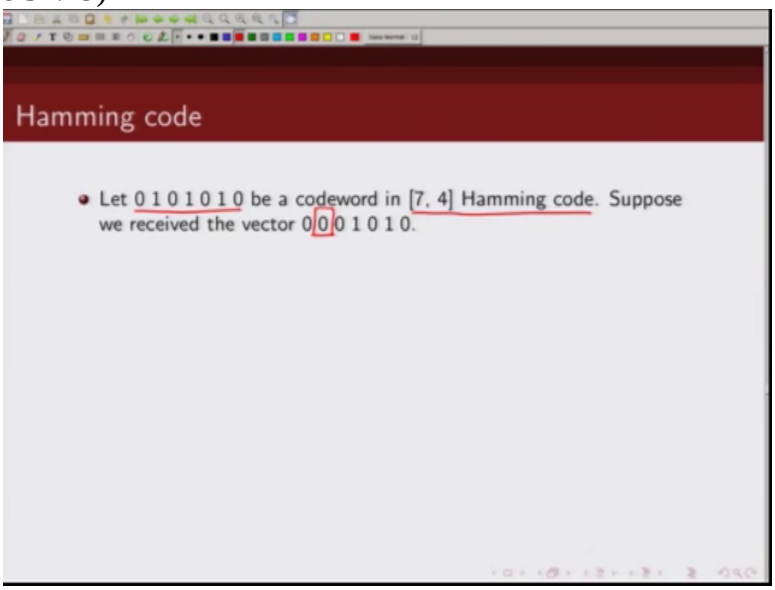
So let's take an example.

(Refer Slide Time 34:02)



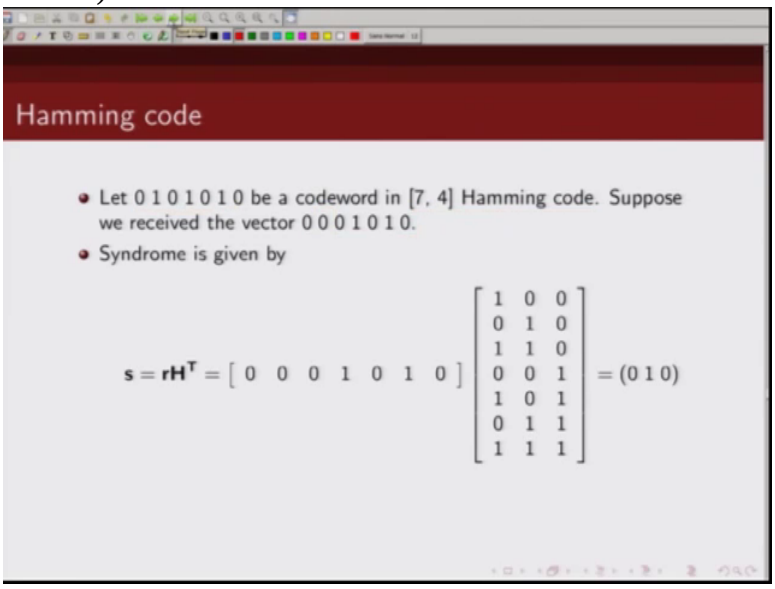
So let's say this is a codeword for this 7 4 Hamming code and let's assume that a single error has happened and this bit got changed from 1

(Refer Slide Time 34:19)



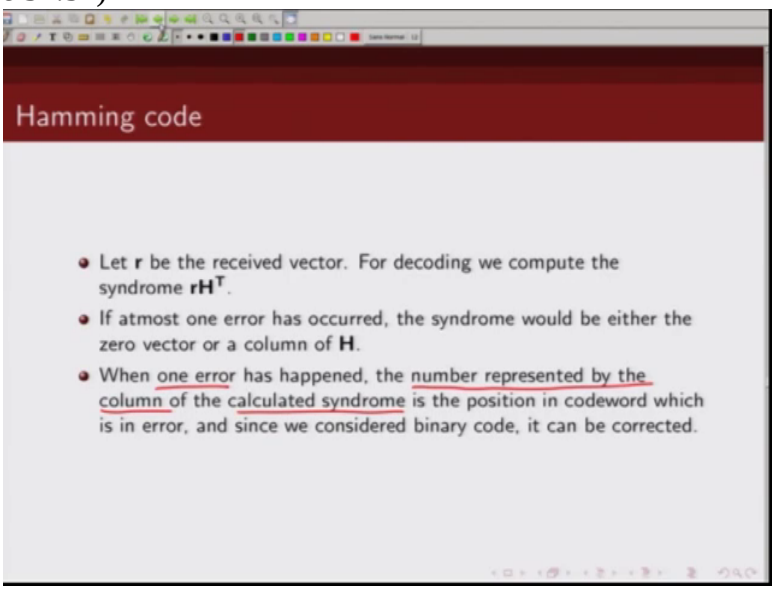
to 0. Now given this received sequence I am interested in finding out what is my estimated codeword. First of all I am interested in finding out whether an error has happened or not and if the error has happened I am interested in correcting that error. So

(Refer Slide Time 34:41)



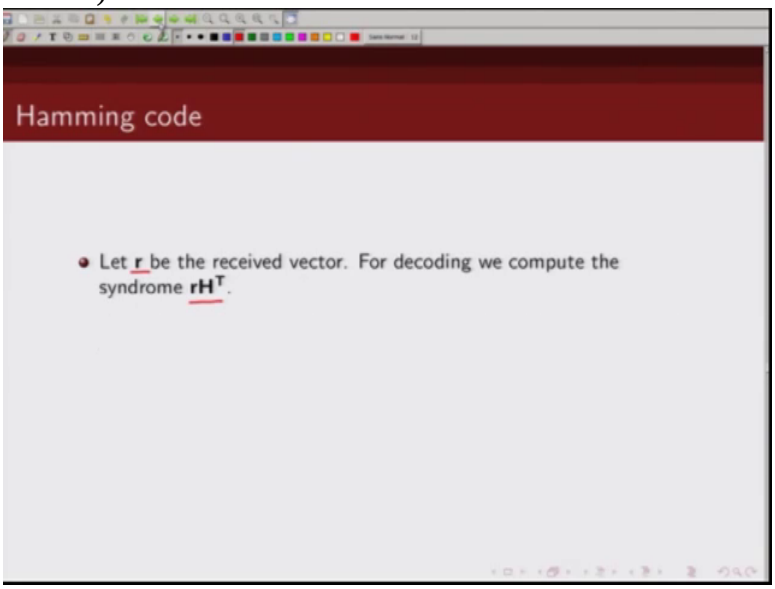
as I said, the first thing I am going to do is I am going to compute the syndrome. So what is my r? This is my r. And what is my H? H matrix is basically my

(Refer Slide Time 34:57)



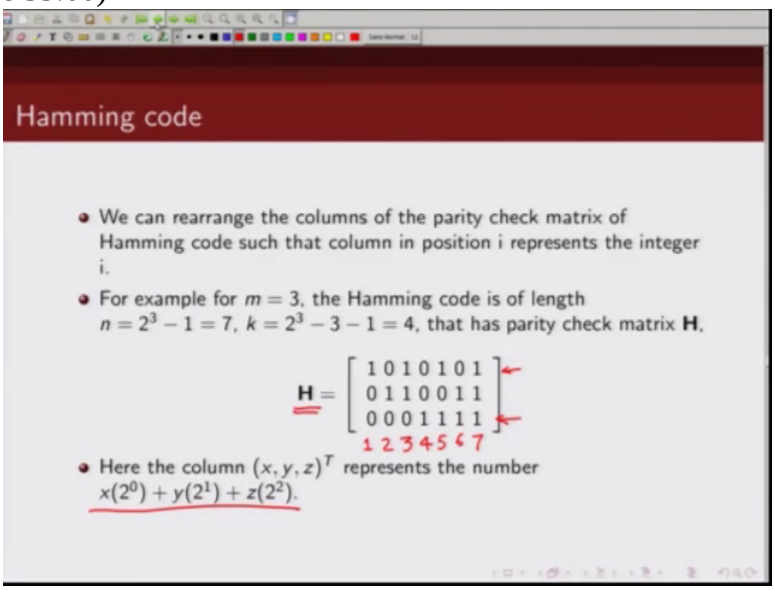
parity check

(Refer Slide Time 34:59)



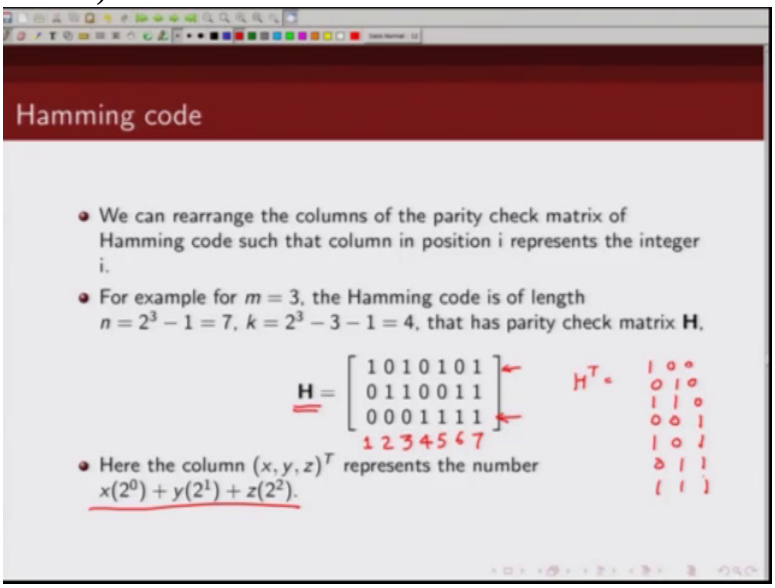
matrix

(Refer Slide Time 35:00)



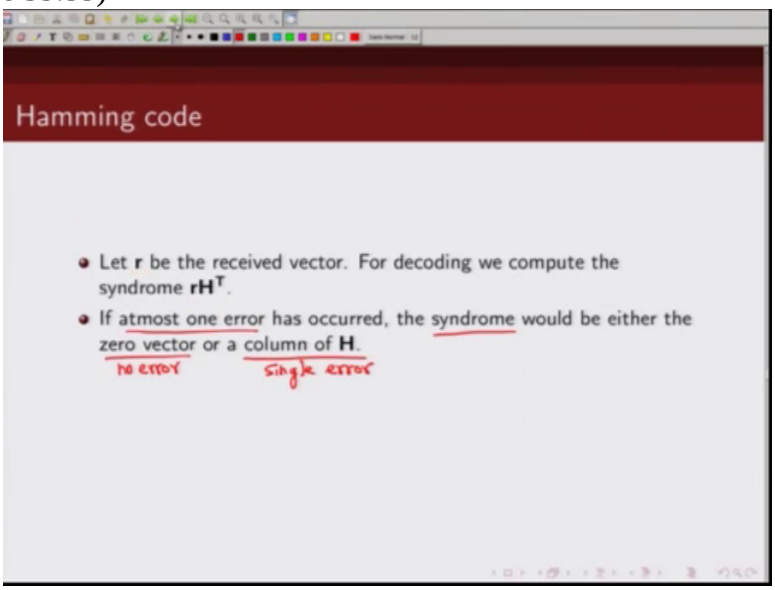
of the Hamming code where the columns are arranged in such a way that column in ith position represents ith integer. So if we take H transpose we get 1 0 0, 0 1 0, 1 1 0, 0 0 1, 1 0 1, 0 1 1, 1 1 1, right and that's what I get here.

(Refer Slide Time 35:32)

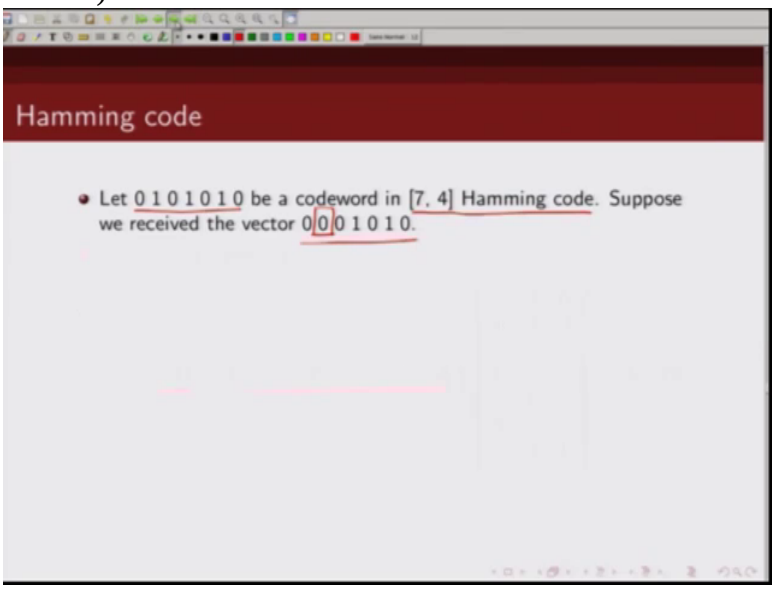


I get here.

(Refer Slide Time 35:33)

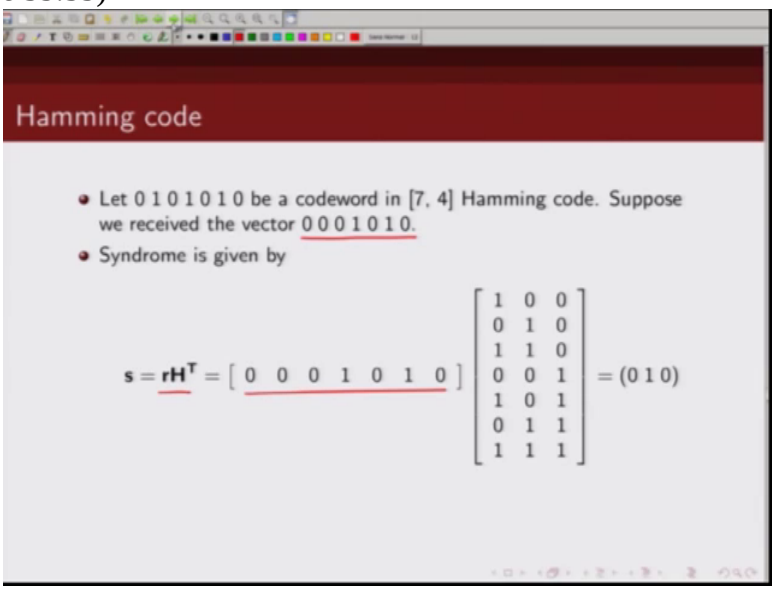


(Refer Slide Time 35:34)



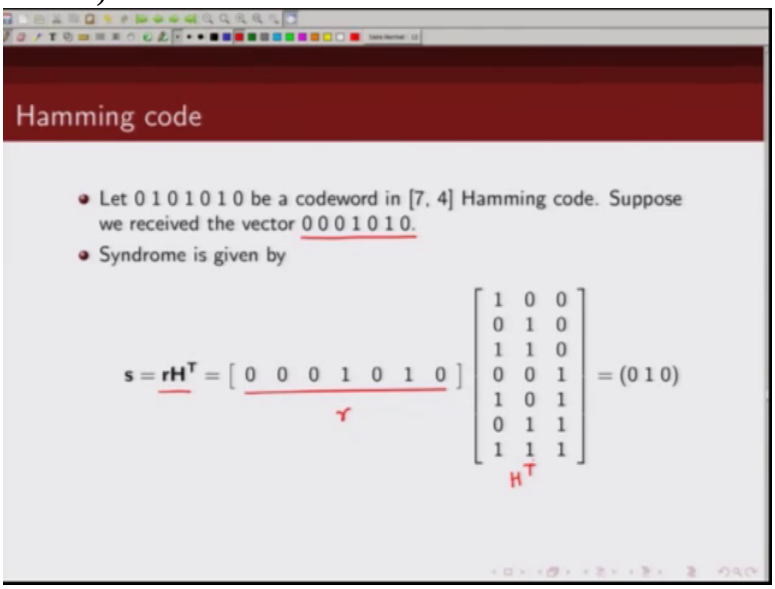
So this

(Refer Slide Time 35:35)



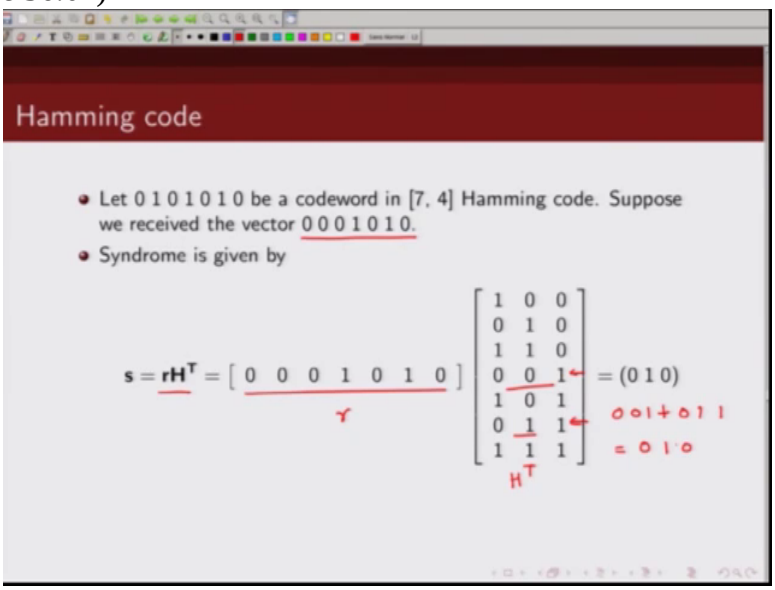
is my r, this is my H transpose, now H transpose,

(Refer Slide Time 35:40)



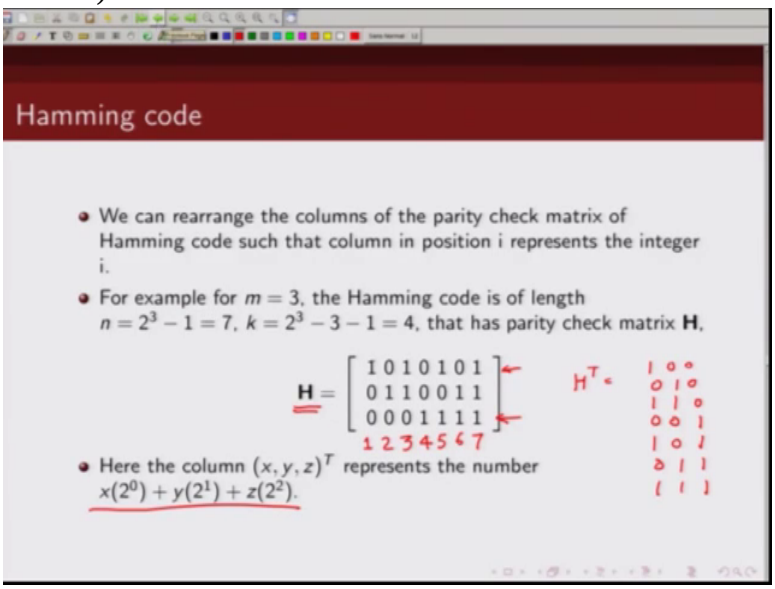
so this is 1, 2, 3, this row will participate this row, so my syndrome will be this plus this, this row plus this row. This will be 0 0 1 plus 0 1 1 which is 0 1 0.

(Refer Slide Time 36:02)



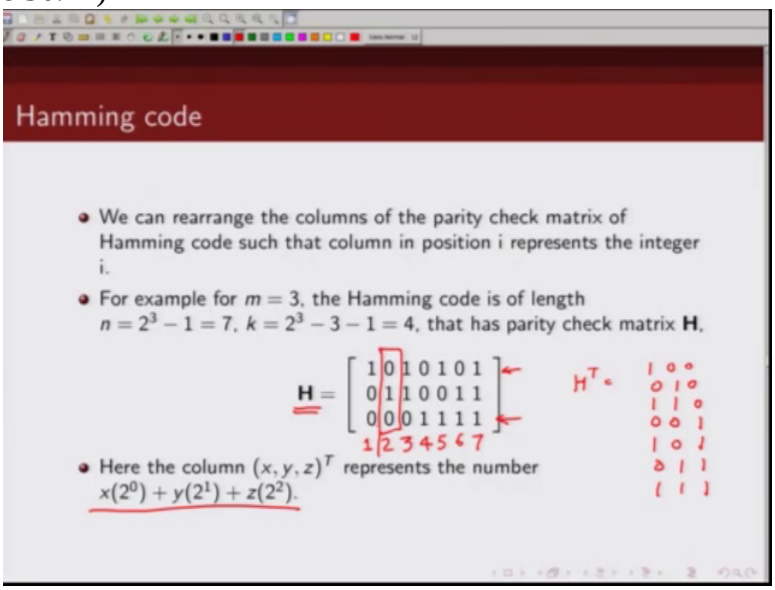
So my syndrome here is 0 1 0. Now what do I do? I go back and check which column is 0 1 0. So if I go back to my parity check matrix I notice that

(Refer Slide Time 36:19)



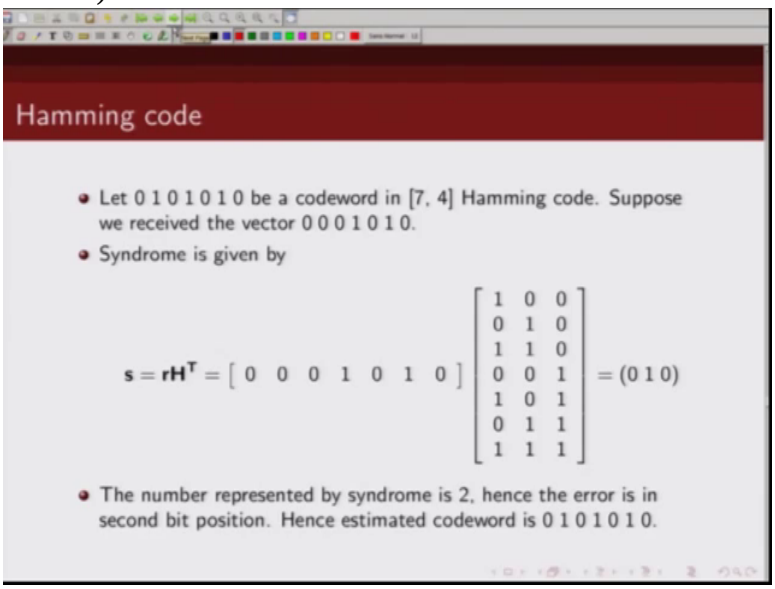
0 1 0 is this second column

(Refer Slide Time 36:24)



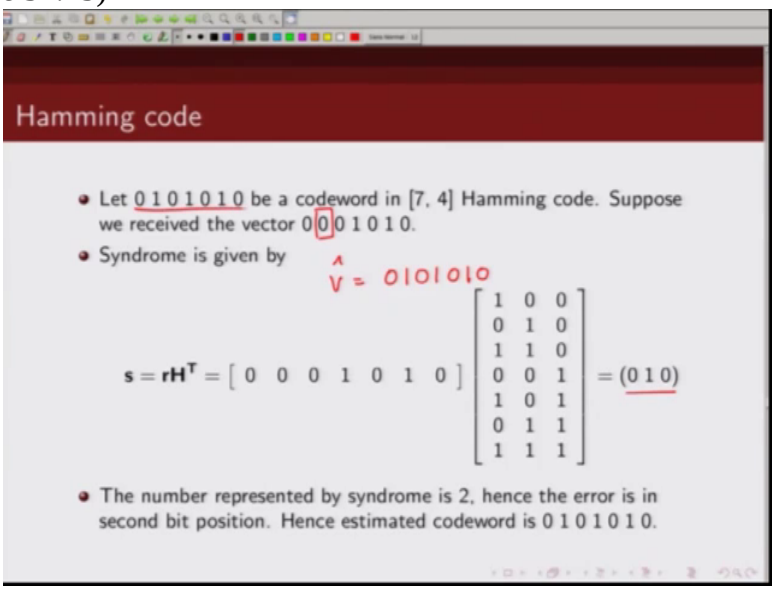
of my parity check matrix, right? So what does it indicate then? It indicates that second bit of my codeword is in error. So then I can go ahead and correct this error. So this second bit,

(Refer Slide Time 36:44)



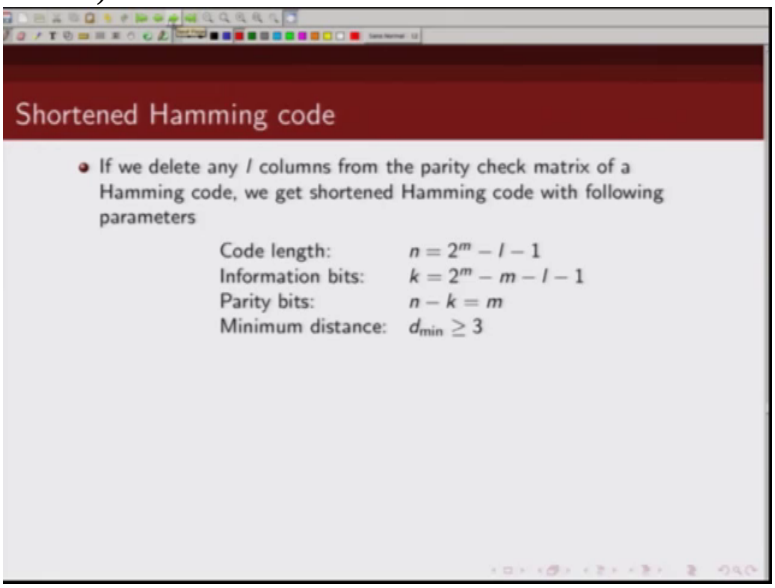
from the syndrome I can find out that this syndrome is nothing but the column in the second location. So my second bit is in error. And since we are talking about binary code so then this 0 should have been 1. So my estimated codeword is then 0 1 0 1 0 1 0. And you can see that's what I had transmitted. This is the same

(Refer Slide Time 37:15)



codeword that I have transmitted. You can yourself verify, change some other bit location, introduce some other single error and you will see, from the syndrome you can find out the location of your error. So it has a very simple

(Refer Slide Time 37:34)

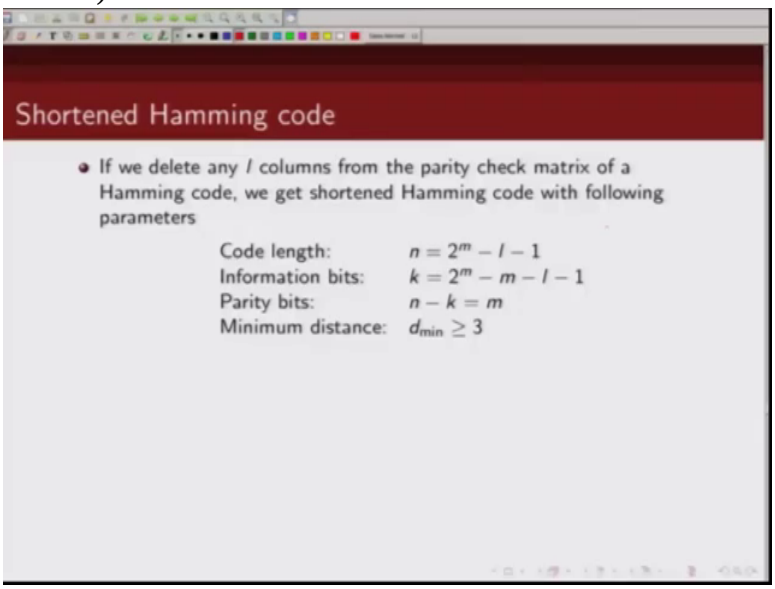


decoding algorithm. Now as I said before, in case of Hamming code, once I fix my m,

(Refer Slide Time 37:42)

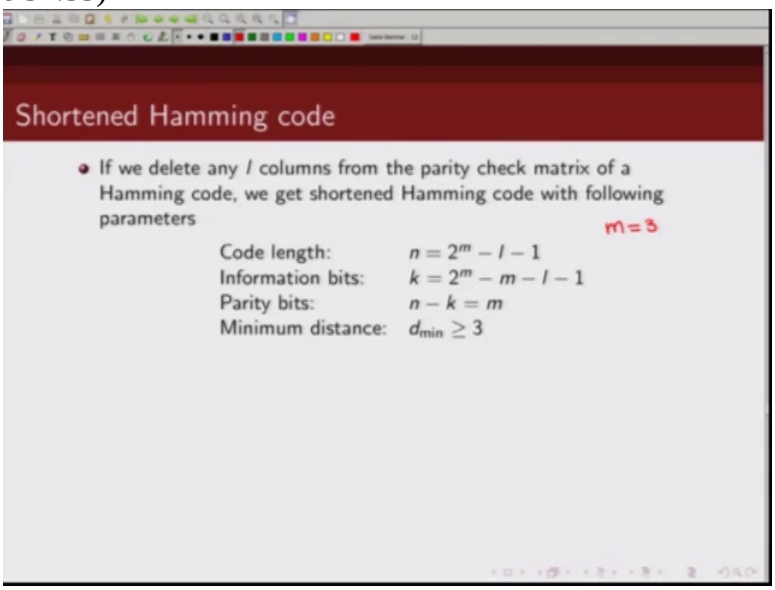
my code parameters are fixed. So for example, the examples that we were dealing with, we had

(Refer Slide Time 37:52)



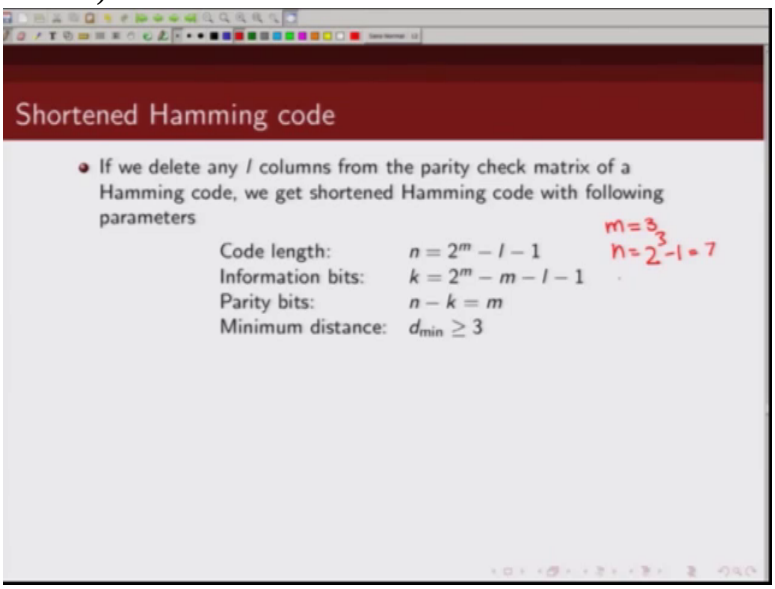
taken m equal to 3. So once

(Refer Slide Time 37:55)



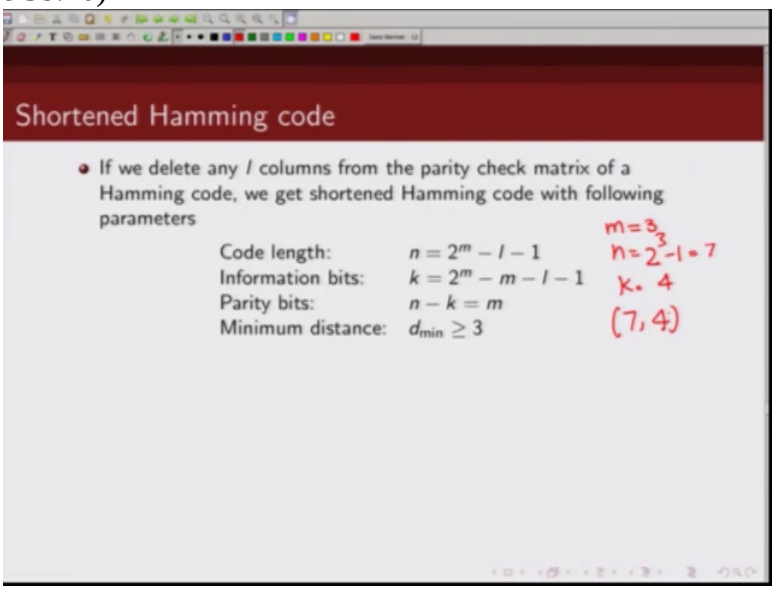
the moment I fixed m equal to 3, my n is fixed which is 2 raised to power 3 minus 1 that is 7 so k is

(Refer Slide Time 38:03)



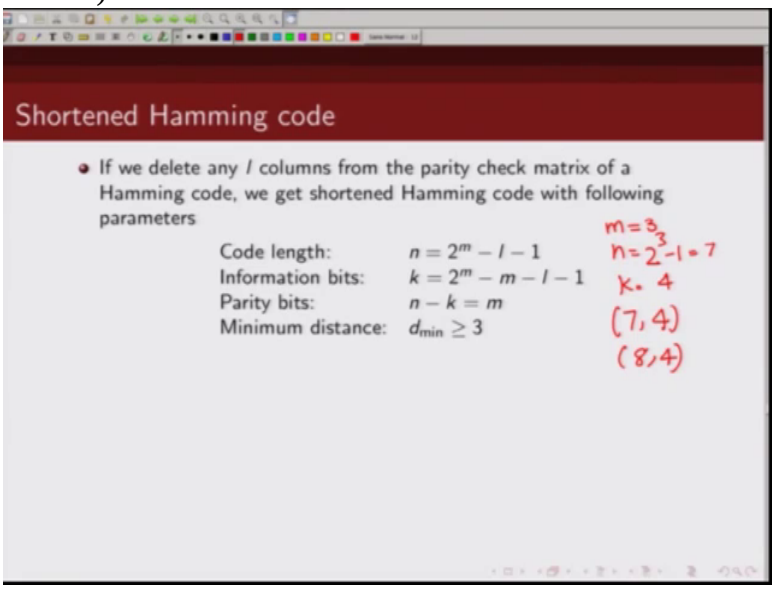
7 minus 3 that's 4. So this becomes a 7 4 code, right? What

(Refer Slide Time 38:10)



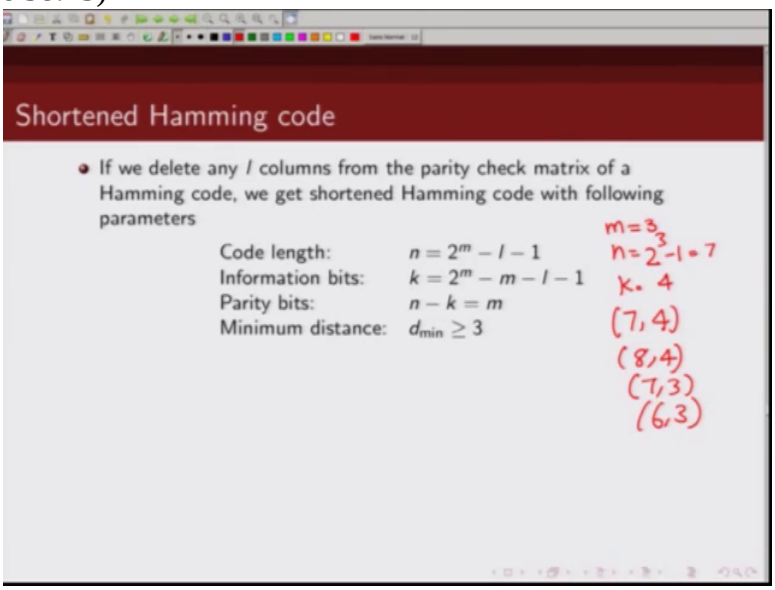
if I am interested in designing, let's say 8 4 code or

(Refer Slide Time 38:16)



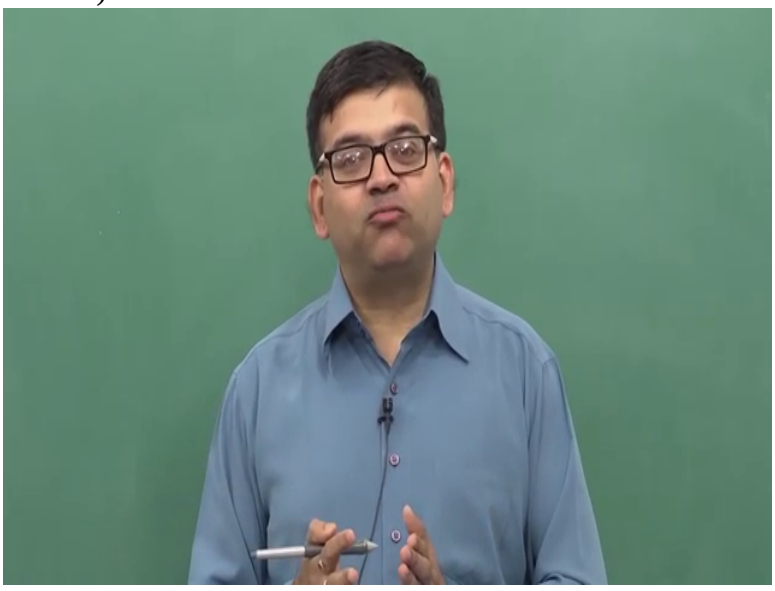
let's say I am interested in 7 3 codes, 6 3 codes. So there are various techniques

(Refer Slide Time 38:23)



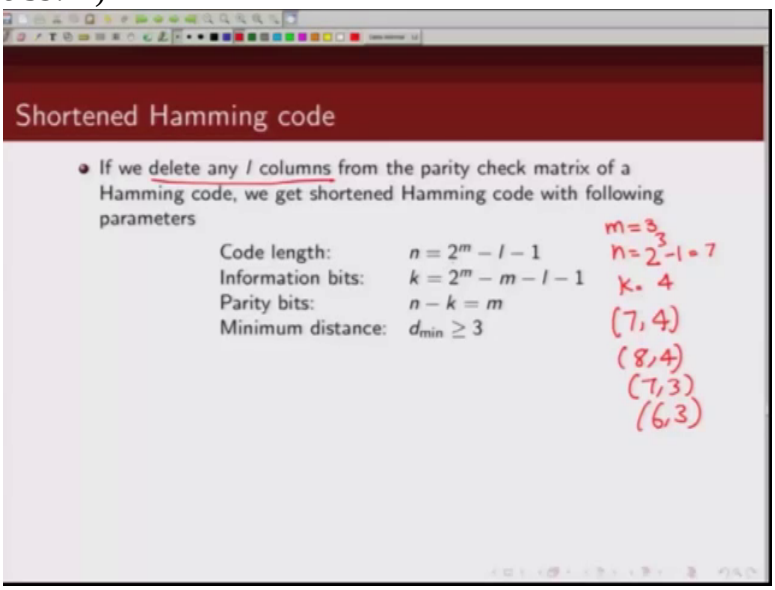
available to tweak the parameters of the code, we are going to talk about some of them, we are not exhaustively covering all possible ways of lengthening or shortening a code, we just give you just few examples to illustrate this idea of changing the parameters of the code. So the first thing we are considering is a shortened Hamming code. So how do you get a shortened Hamming code? You delete l columns from the parity check matrix of your Hamming code. Now if I delete l columns, that means my codeword length is decreased by l.

(Refer Slide Time 39:08)



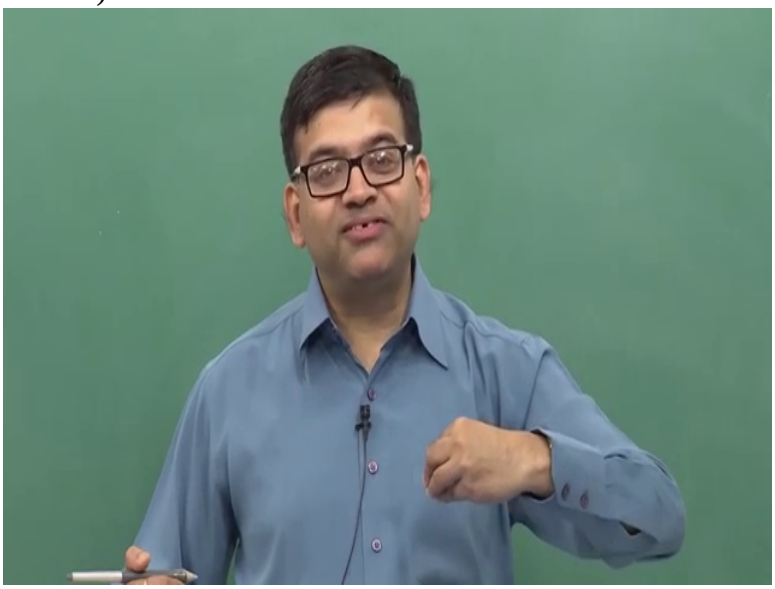
So my new codeword length is then

(Refer Slide Time 39:11)



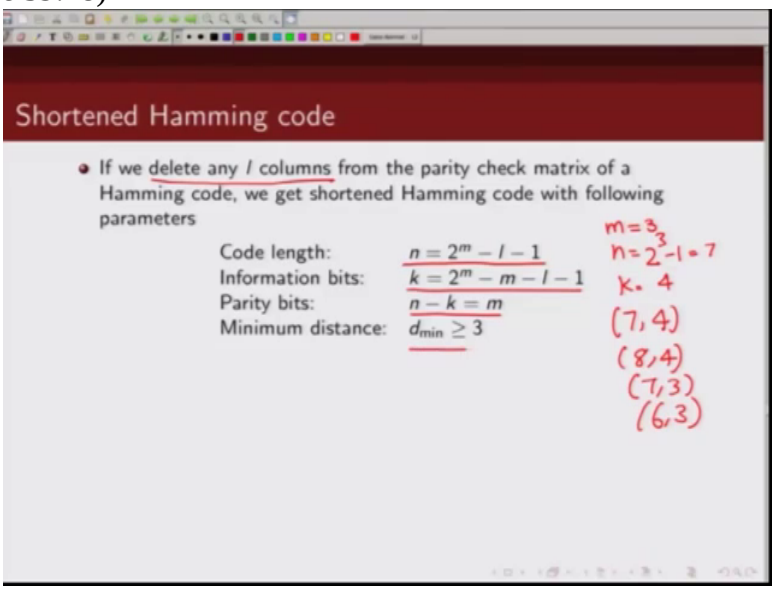
2 m minus 1 minus l. So this is my codeword length. Now number of rows are still the same. That's m so number of parity bits are still same. But since I deleted l coded bits so number of information bits also got decreased by l. Now minimum distance of Hamming code is 3. If I am deleting some columns, then minimum distance cannot

(Refer Slide Time 39:46)



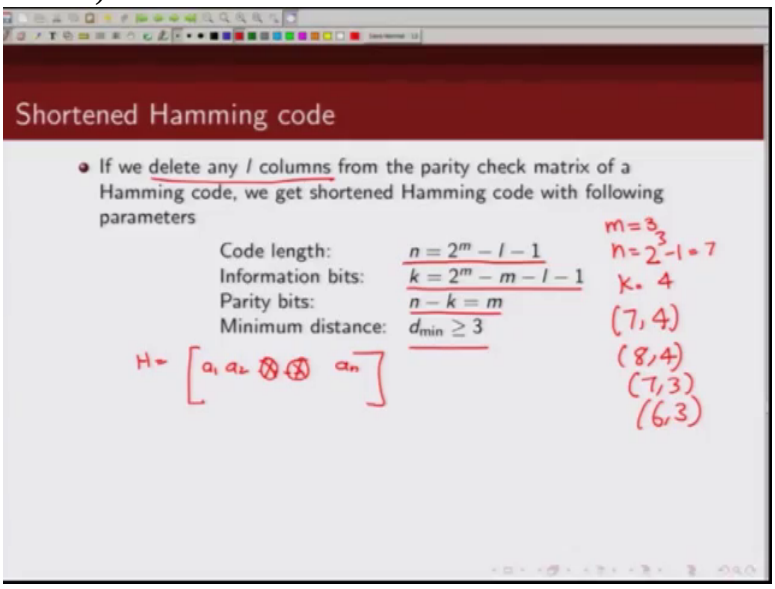
be less than what

(Refer Slide Time 39:48)



was the original minimum distance of the code. Why, because you can see if I have a parity check matrix here and let's say I have these columns, let's call it a 1, a 2... a n. Now if I am removing some of the columns I can carefully

(Refer Slide Time 40:11)

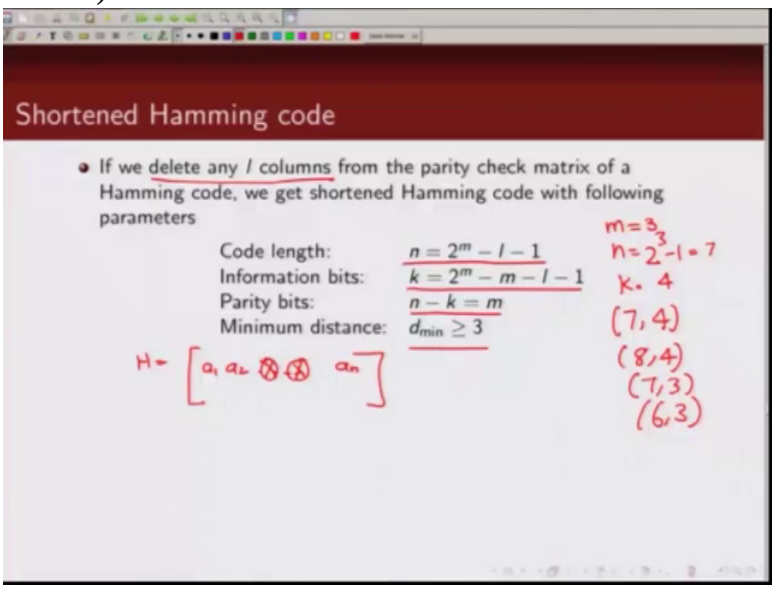


remove those columns which

(Refer Slide Time 40:14)

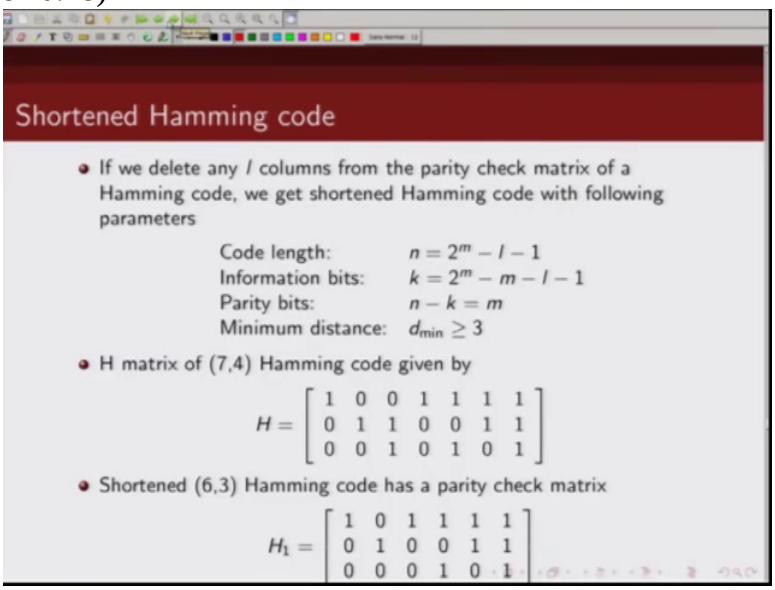
are causing the sum of, let's say d columns to be linearly dependent. If I remove some of those columns I can possibly increase my minimum distance. And that's why I wrote

(Refer Slide Time 40:30)



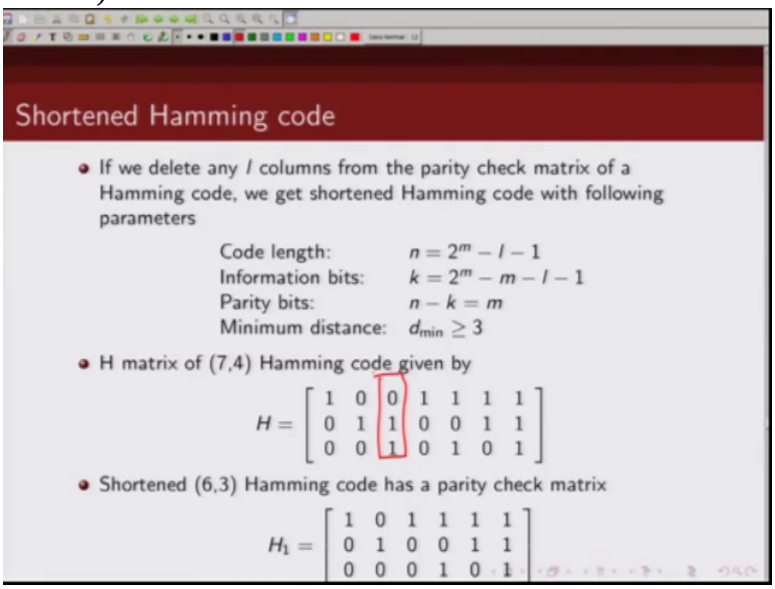
by whenever you delete columns on the parity check matrix, the minimum distance is at least what was there in the original code. This is one example of a 7 4 Hamming code. So if I delete, let's say I delete

(Refer Slide Time 40:48)



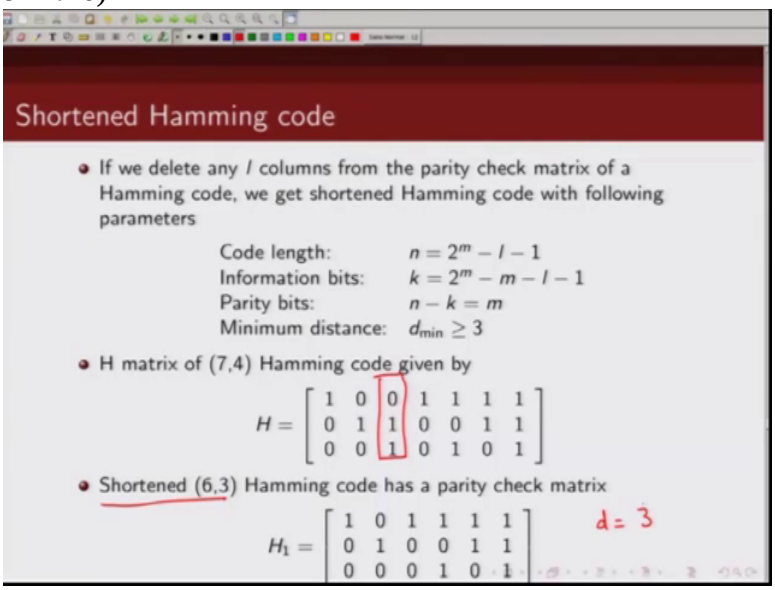
this, if I delete

(Refer Slide Time 40:52)



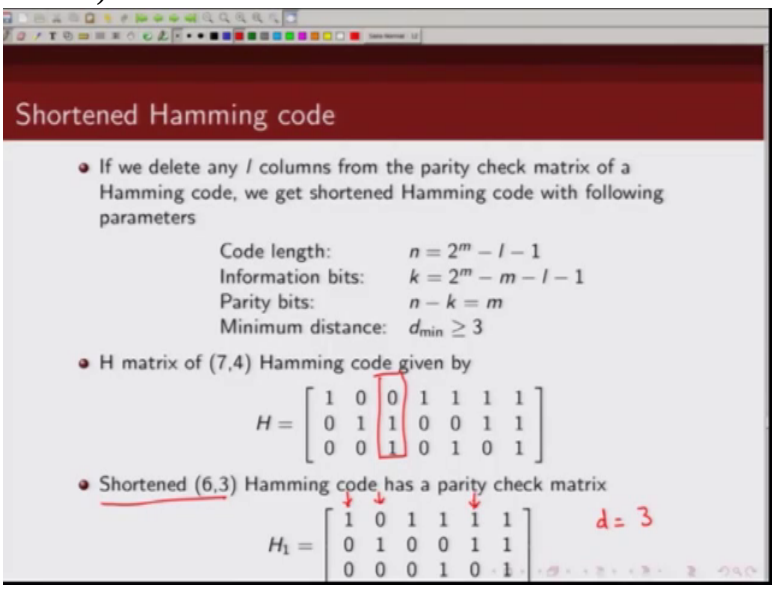
this column, what do I get? I get a shortened 6 3 Hamming code. So when I delete this my n becomes 6 from 7 and my information bits also reduce from 4 to 3. This code also has minimum distance 3, you can see 3 columns will add up to zero.

(Refer Slide Time 41:16)



Let's say this column, this column, and this column add up to zero so minimum distance is still 3.

(Refer Slide Time 41:24)

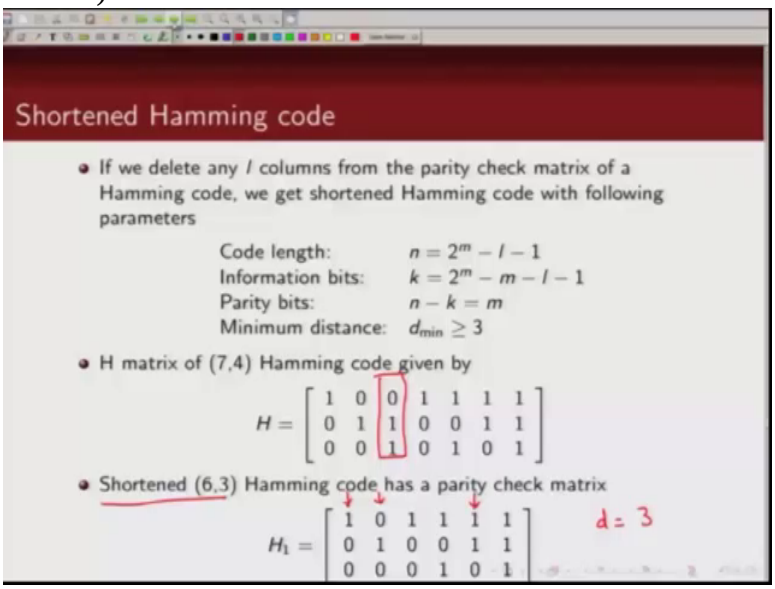


Or this one. So whenever you want shortened code, make sure you remove

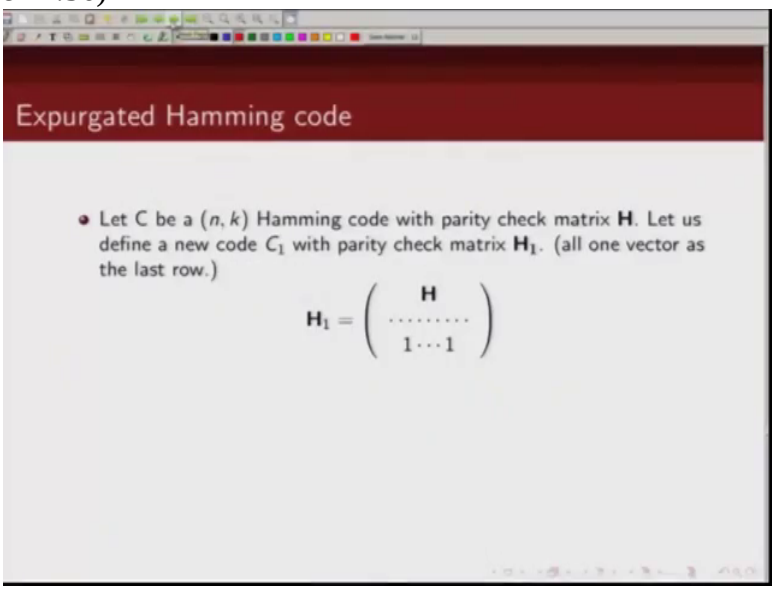
(Refer Slide Time 41:31)

those columns in a judicial way so that potentially you can increase the minimum distance. In this particular example, if I delete one column I cannot increase the Hamming distance but if I delete more columns from this parity check matrix I can potentially increase the minimum distance of the code.

(Refer Slide Time 41:55)

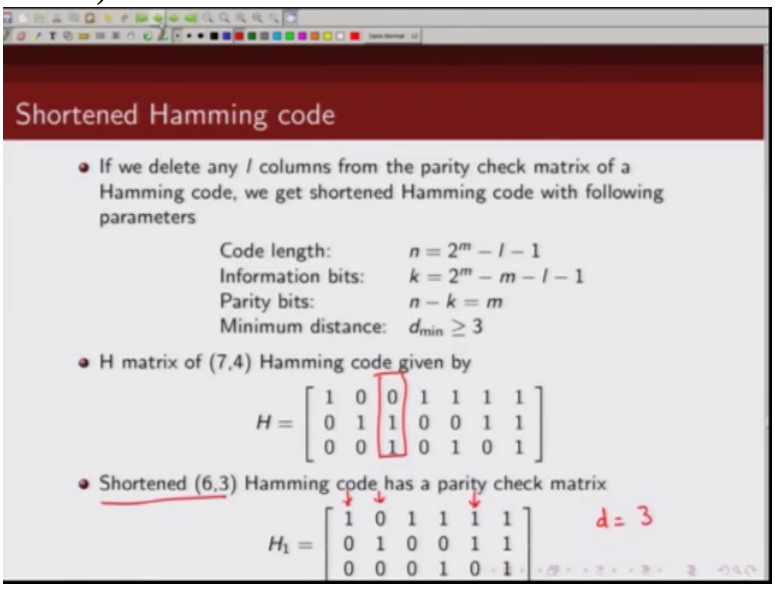


(Refer Slide Time 41:56)



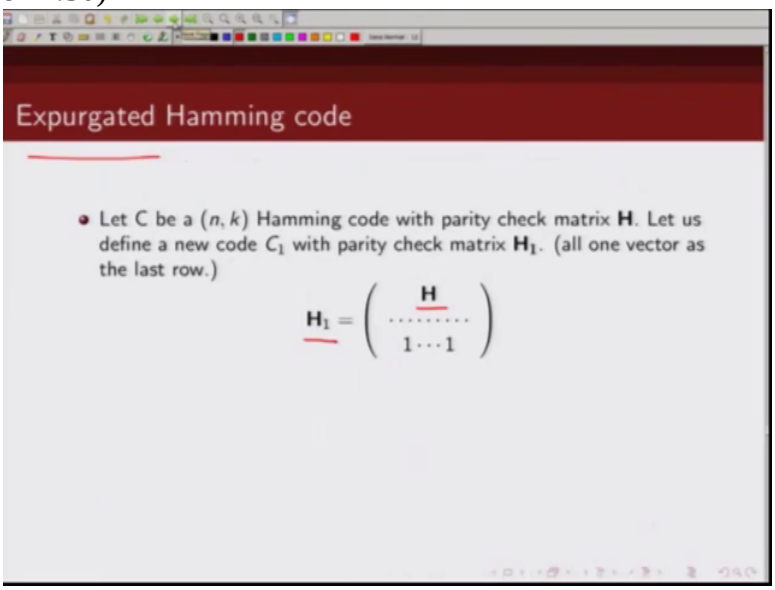
Another class of code is what is called expurgated codes. So what is expurgated Hamming code? So H is the parity check matrix of Hamming code, then I define a new parity check matrix which has all 1's at the last row. Now note that the parity check matrix

(Refer Slide Time 42:22)



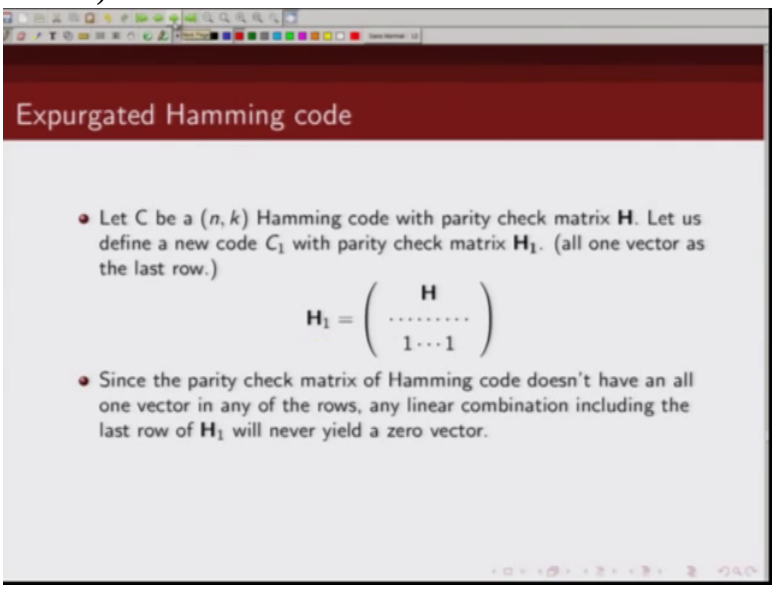
of Hamming code does not have all 1's. So when I add an all 1 row,

(Refer Slide Time 42:30)



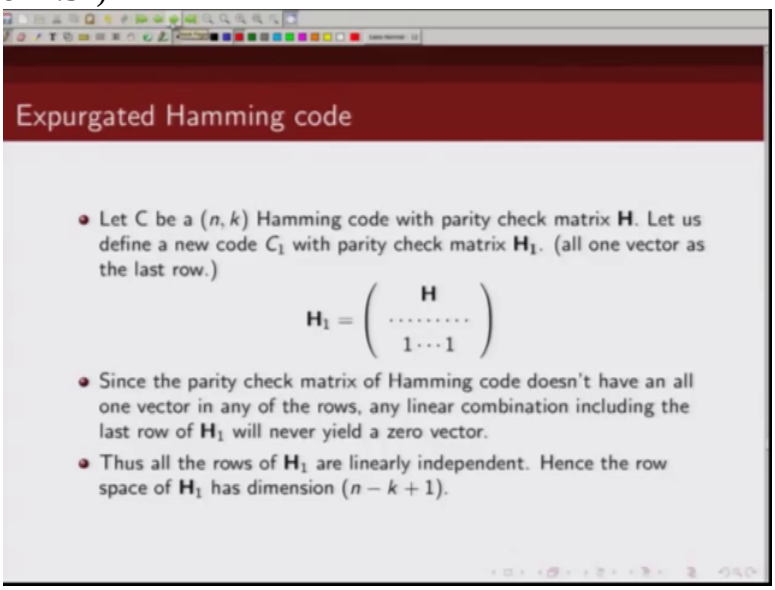
essentially the rank of the matrix will be one more than the rank of the original matrix H because

(Refer Slide Time 42:40)



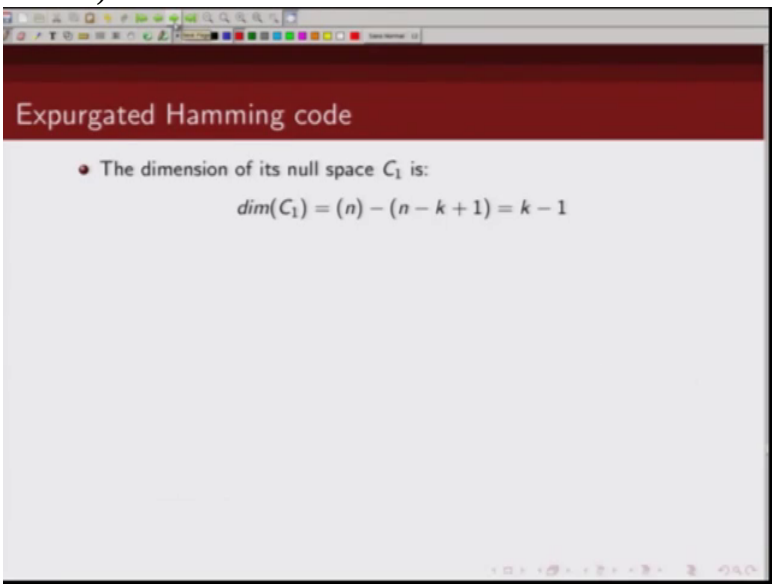
any linear combination of last row with the earlier parity check matrix H would not be dependent. Hence

(Refer Slide Time 42:52)



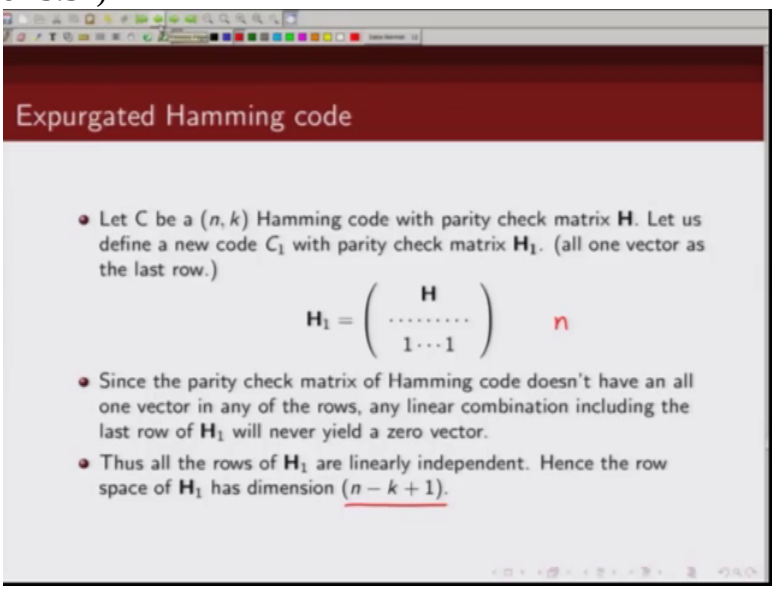
the rank of this new matrix H 1 is n minus k plus 1. And what is the number of columns of this matrix? That is n. So rank of this matrix is n minus k plus 1. Then what is the dimension of the null space of this? Or what is the dimension

(Refer Slide Time 43:14)



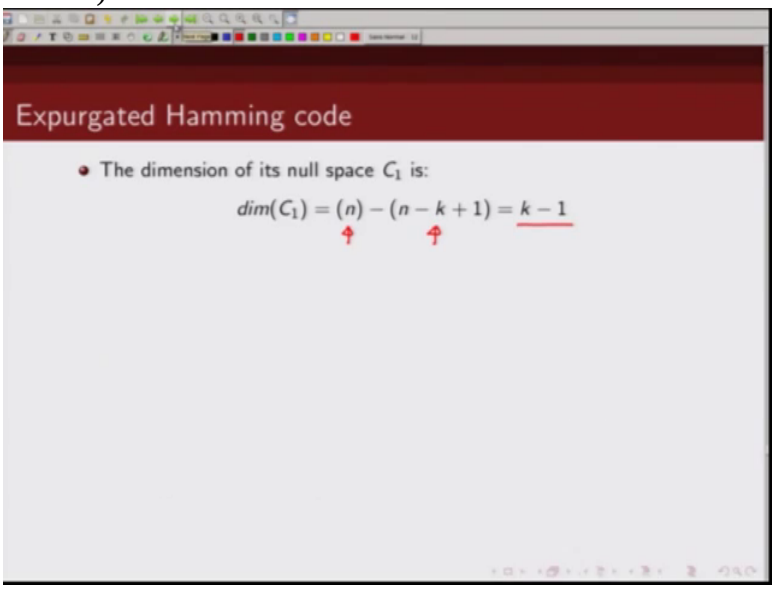
of this code sequence here? So that is basically, this is n minus dimension of H matrix so that becomes k minus 1. So the new code that we derive by adding

(Refer Slide Time 43:31)



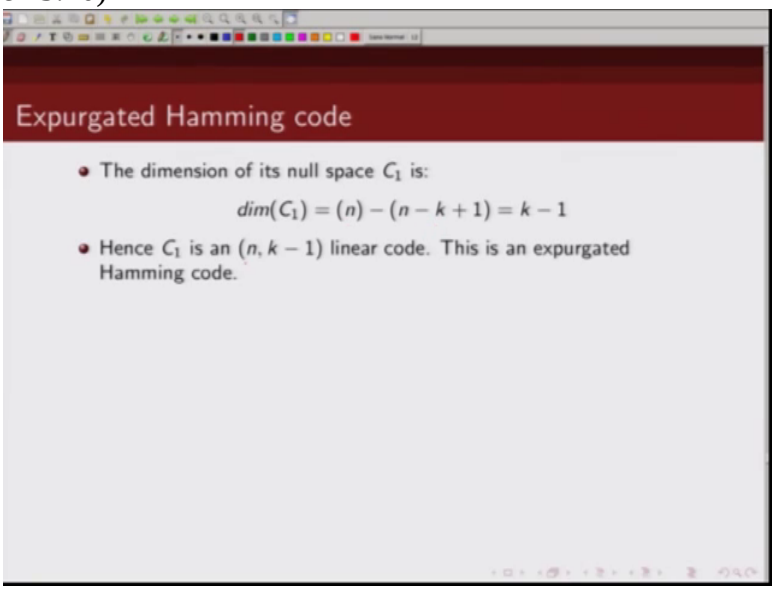
an all 1 row in the parity check matrix of the Hamming code, we get a new

(Refer Slide Time 43:37)



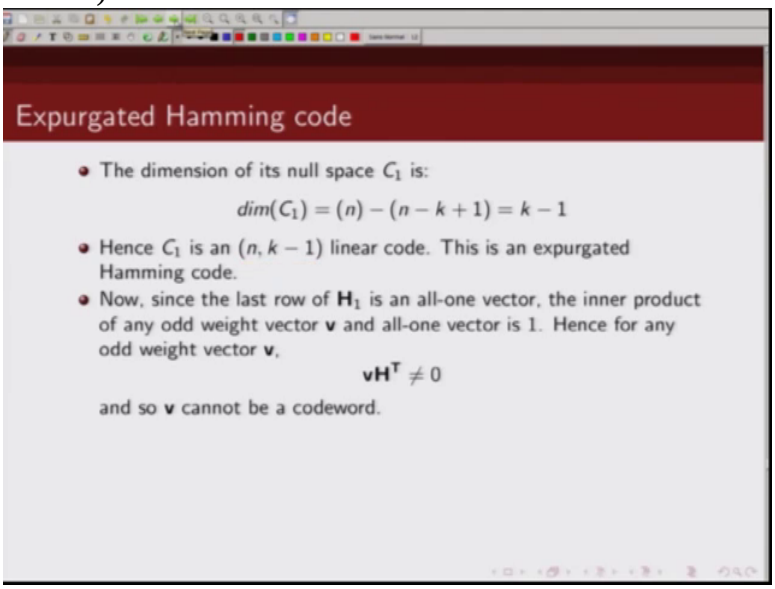
code which is a n k minus 1

(Refer Slide Time 43:40)



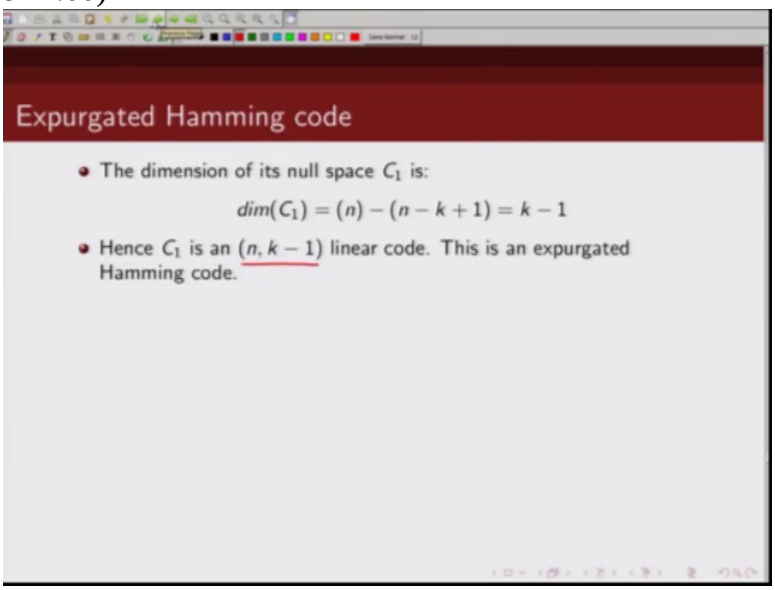
code and this is known as expurgated Hamming code. Now one of

(Refer Slide Time 43:48)



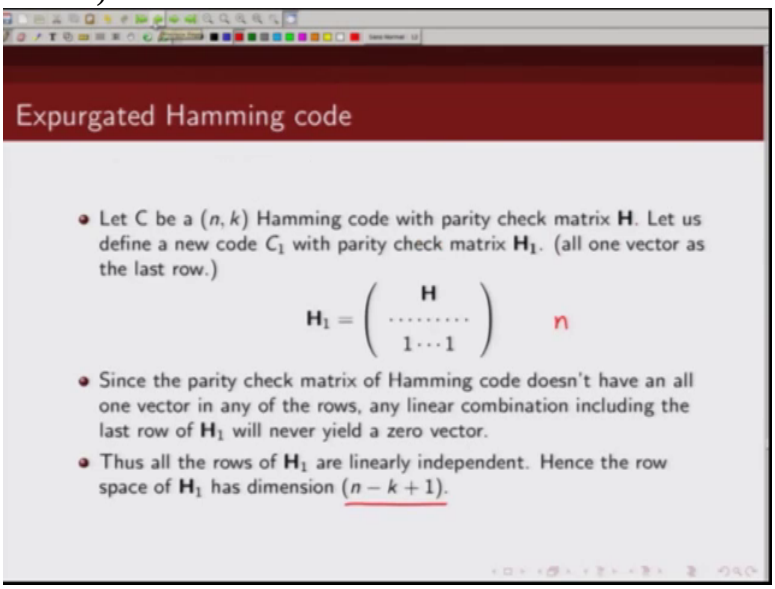
the interesting property of this code is this code contains all even weight codewords. It is not very difficult to prove. If

(Refer Slide Time 44:00)



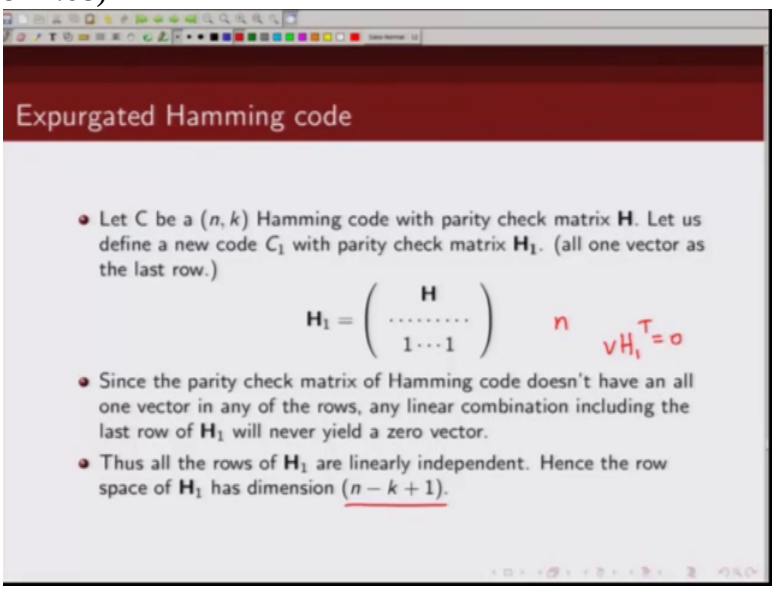
v is a valid codeword

(Refer Slide Time 44:03)



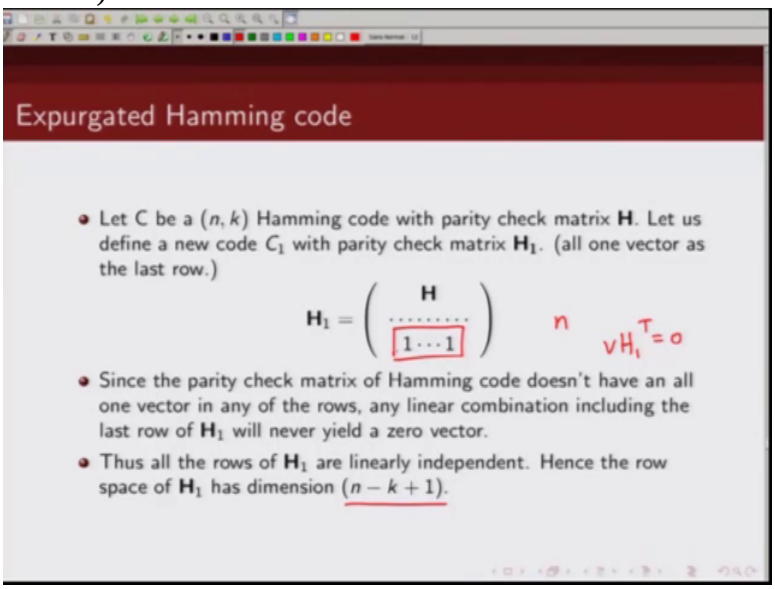
then we know v H transpose should be

(Refer Slide Time 44:08)



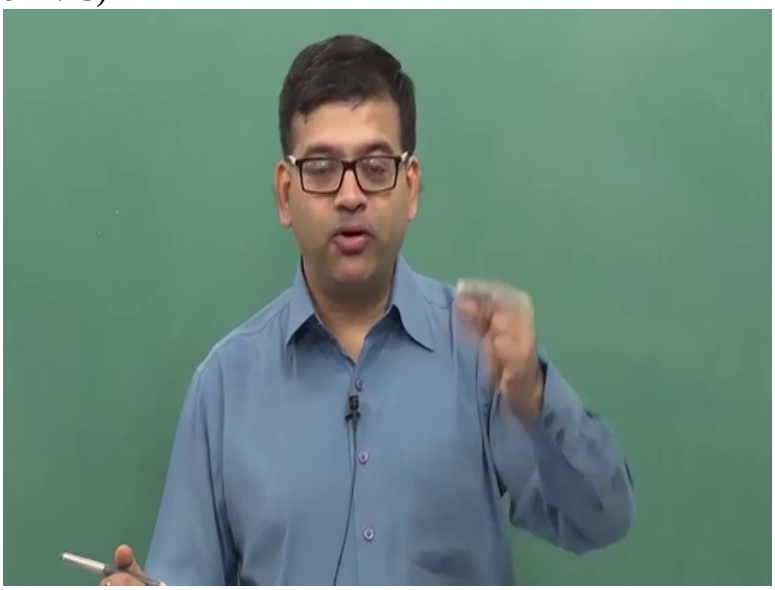
zero. And last row of H 1 has all 1's. So when we take transpose,

(Refer Slide Time 44:16)



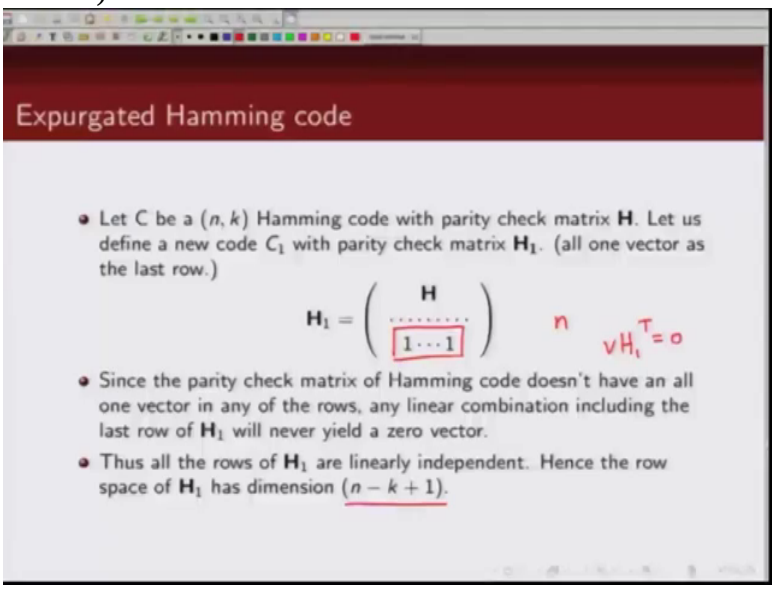
the first column will be all 1's. And when we do v H 1 transpose what we will get is, let's say the components of v are

(Refer Slide Time 44:25)



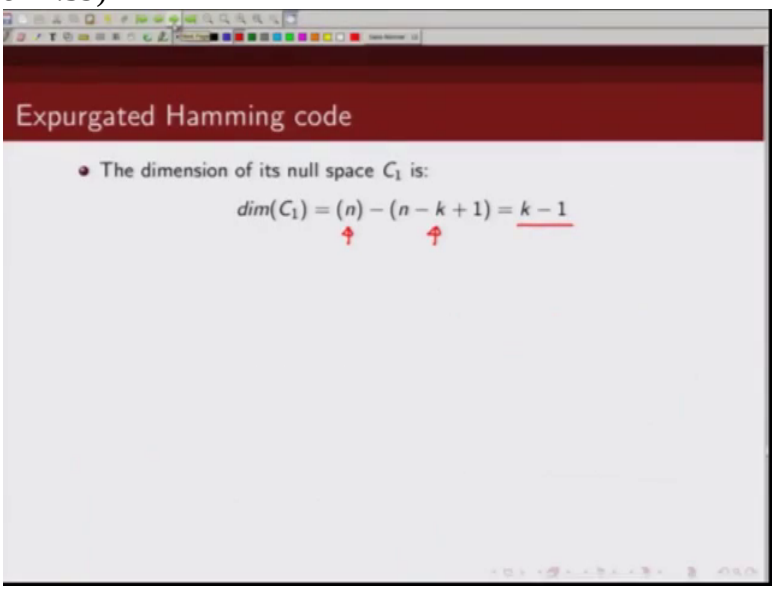
v 0, v 1, v 2 so v n minus 1 so what you will get is basically sum of these

(Refer Slide Time 44:36)



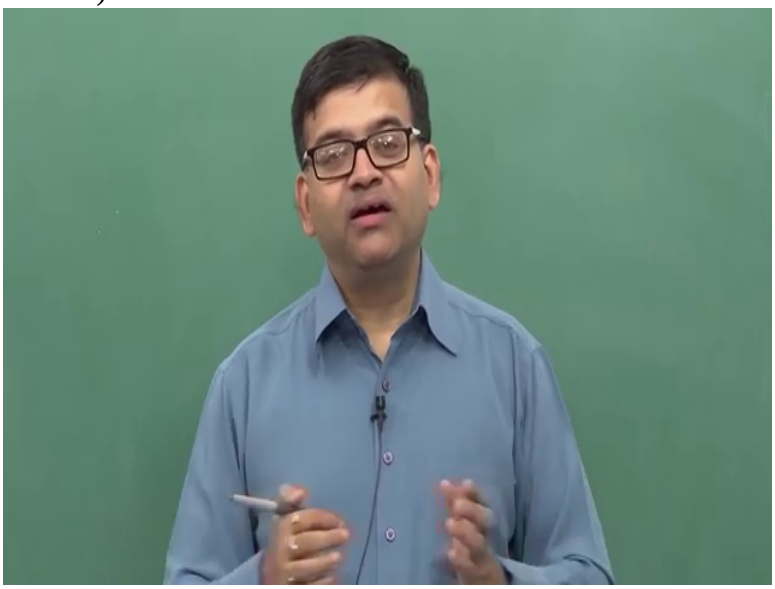
component of codewords,

(Refer Slide Time 44:39)



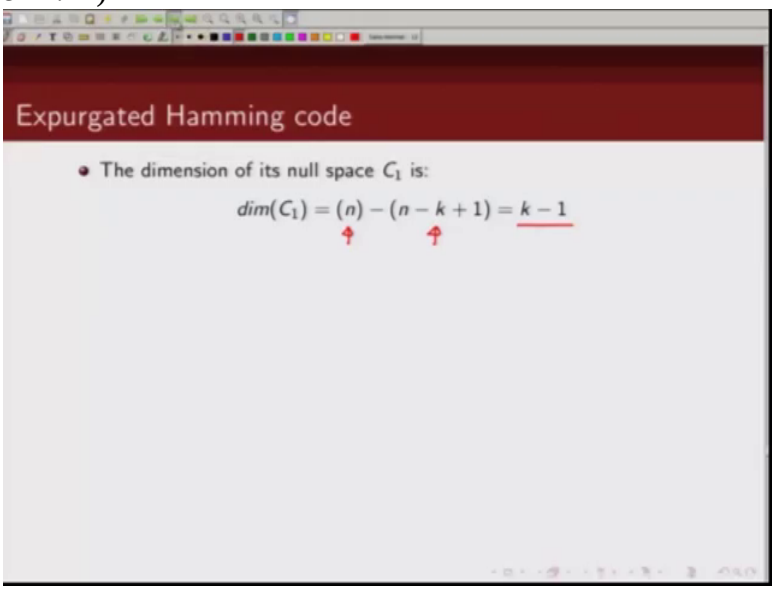
they should add up to zero and this will happen only when

(Refer Slide Time 44:44)

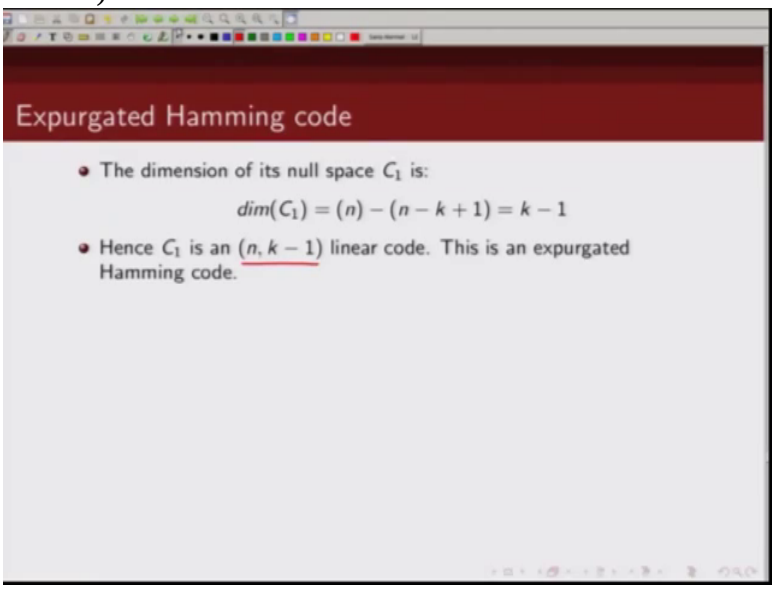


v has even weight.

(Refer Slide Time 44:47)

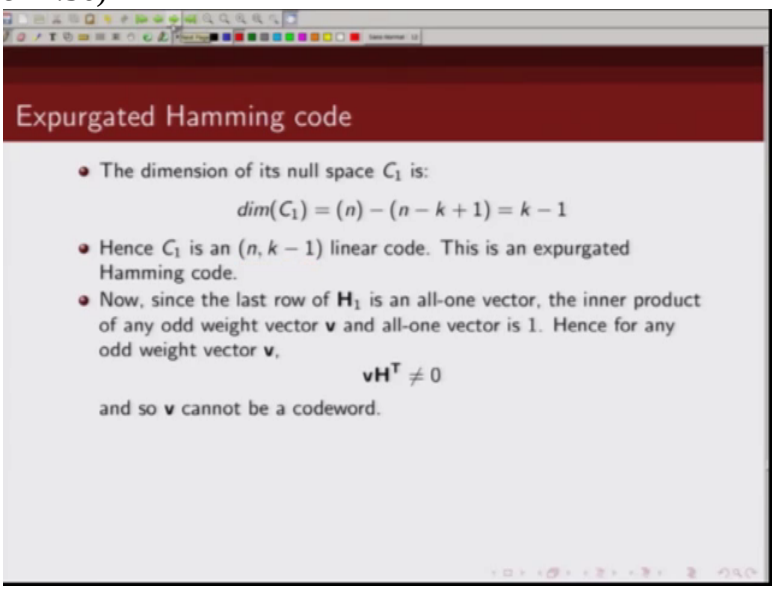


(Refer Slide Time 44:48)



So this new code that we generated, expurgated code is basically has all even weight codewords.

(Refer Slide Time 44:56)

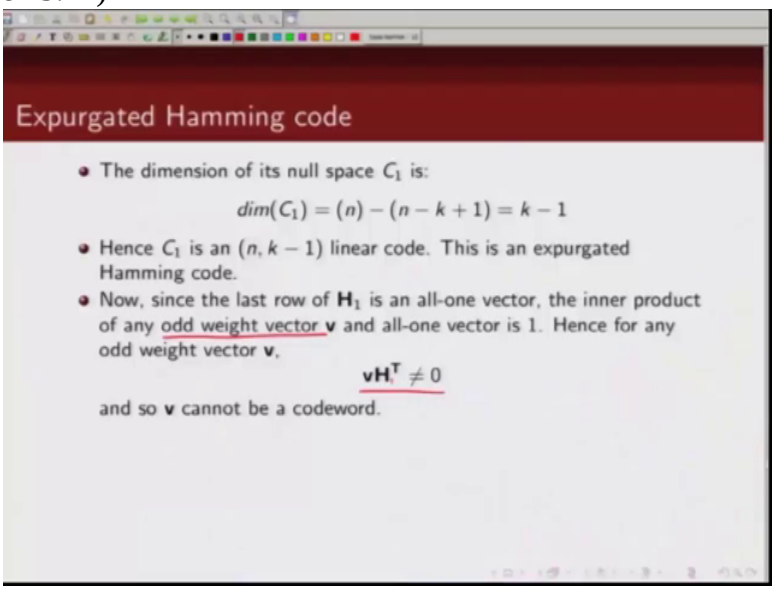


So if you have an odd weight vector then v H transpose cannot be zero because of all these, all 1's

(Refer Slide Time 45:07)

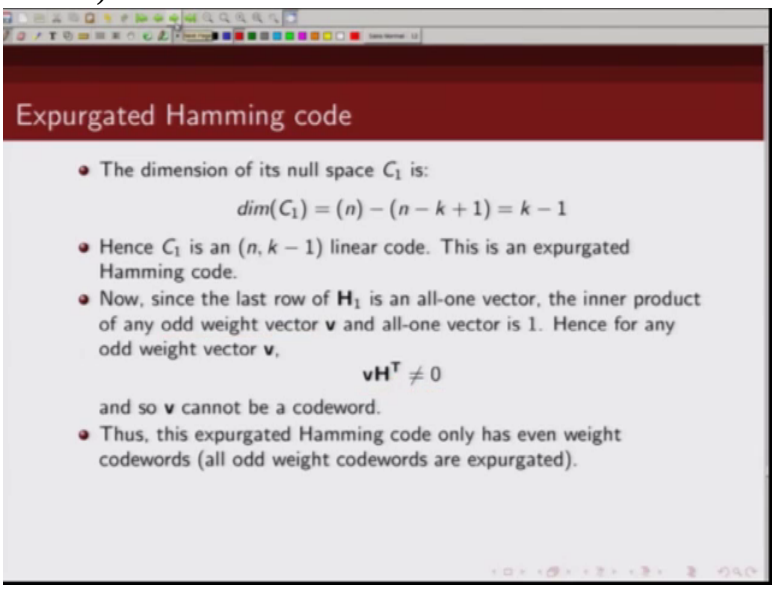
in the parity check matrix

(Refer Slide Time 45:12)



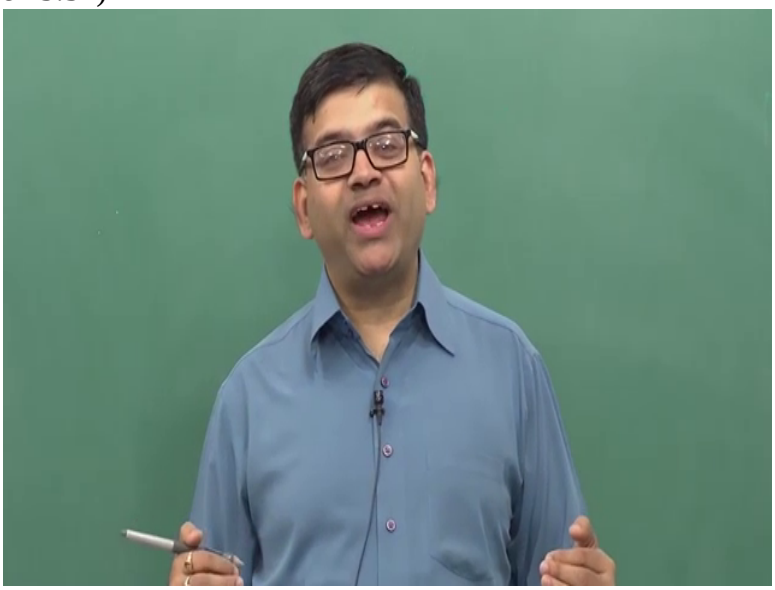
of this new matrix H 1, expurgated code parity check matrix, Ok. So v H 1 transpose cannot be zero if v has odd weight. So you can see that

(Refer Slide Time 45:29)



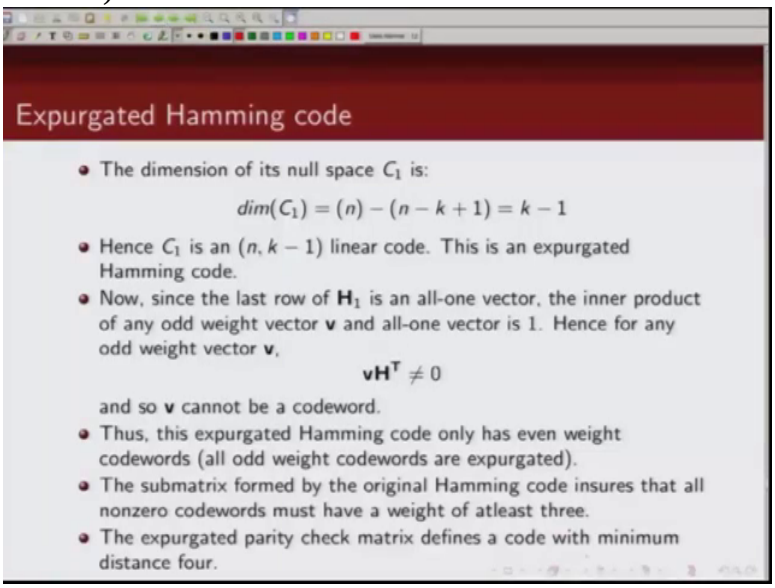
this is one way of

(Refer Slide Time 45:31)



getting rid of odd codewords. So by adding an additional all 1 rows in the parity check matrix of the original code, we actually got rid of all odd weight codewords. So the minimum distance of this expurgated Hamming code would be then 4. Why? Original code has minimum distance 3, but now we got rid of all odd weight codewords. So the minimum distance of this new code, expurgated Hamming code would be 4.

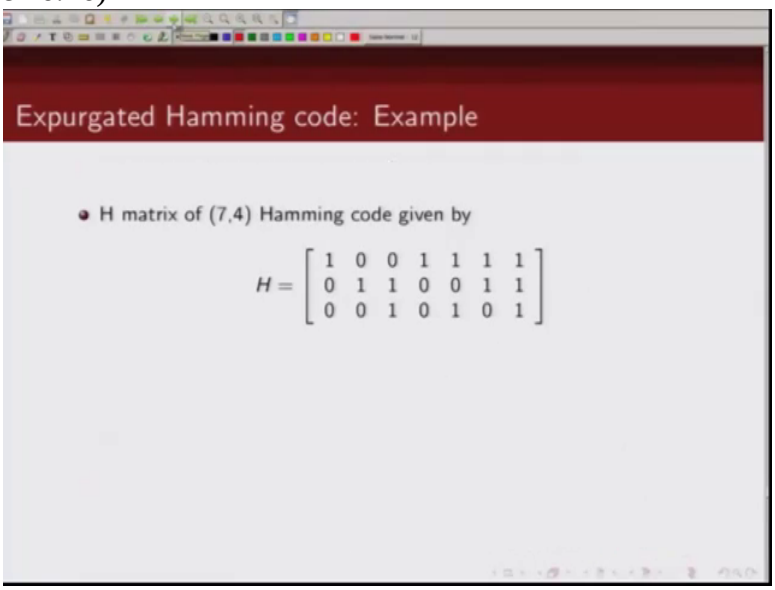
(Refer Slide Time 46:11)



So minimum distance is 4.

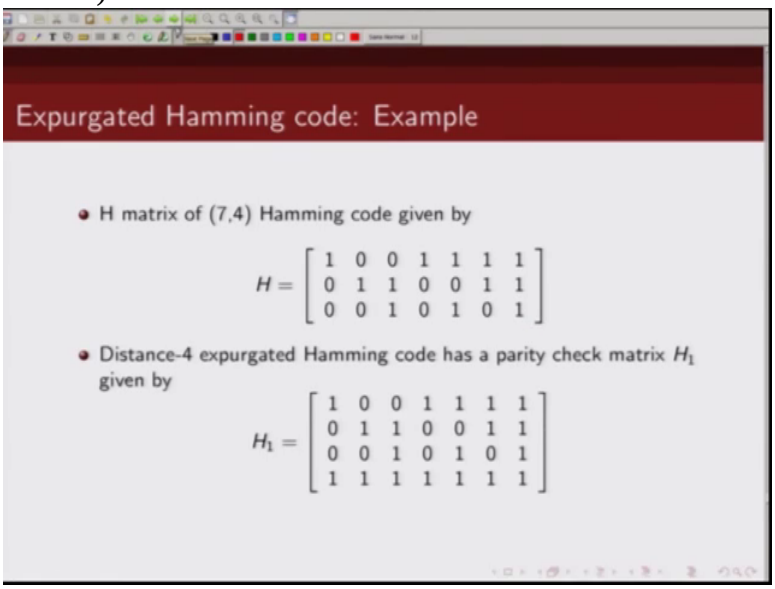
So this is

(Refer Slide Time 46:16)



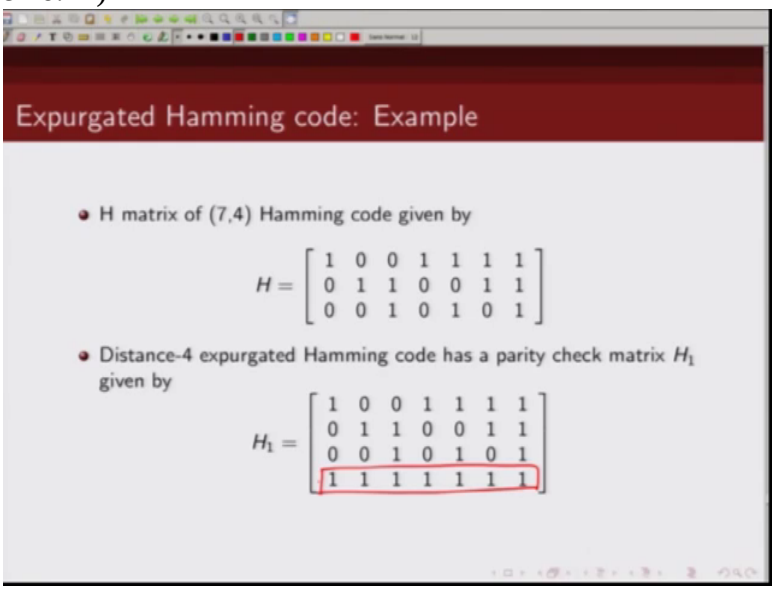
an example, this is the parity check matrix of the original Hamming code and if we add

(Refer Slide Time 46:23)



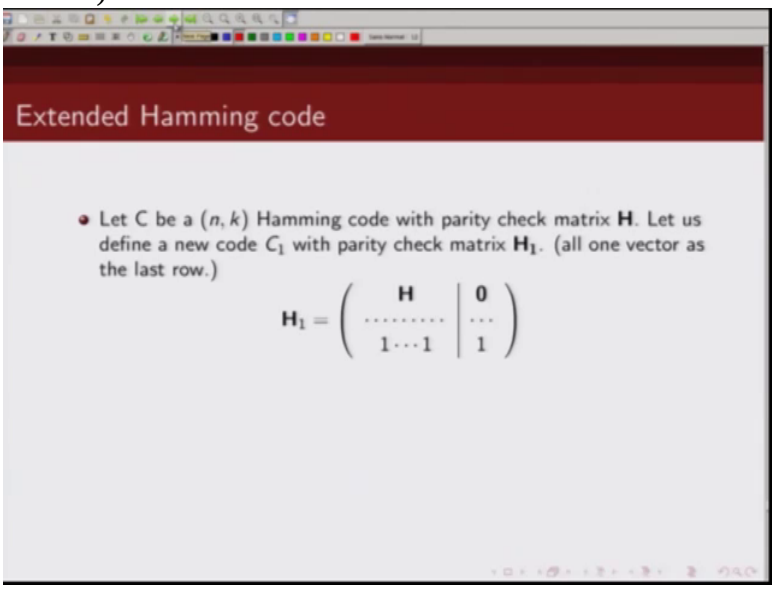
all 1 rows in the parity check matrix, we get

(Refer Slide Time 46:27)



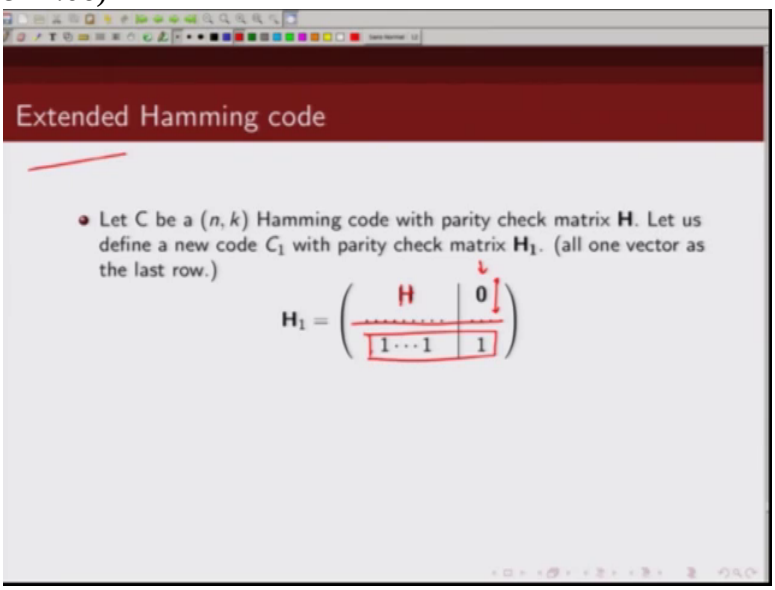
parity check matrix of expurgated code. Finally

(Refer Slide Time 46:32)



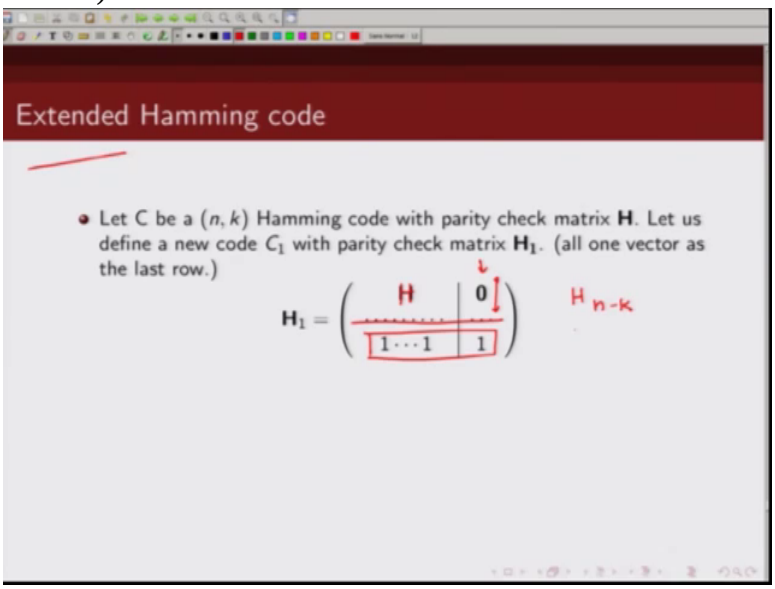
I will conclude this lecture with another class which is basically extension; it is called extension of the code. So this is an example of extended Hamming code. So how do I generate an extended Hamming code? So note, so this is the parity check matrix of the original Hamming code. I add an zero, I add an additional column which is zero here, so these are all zero here and then I add 1 row which is

(Refer Slide Time 47:06)



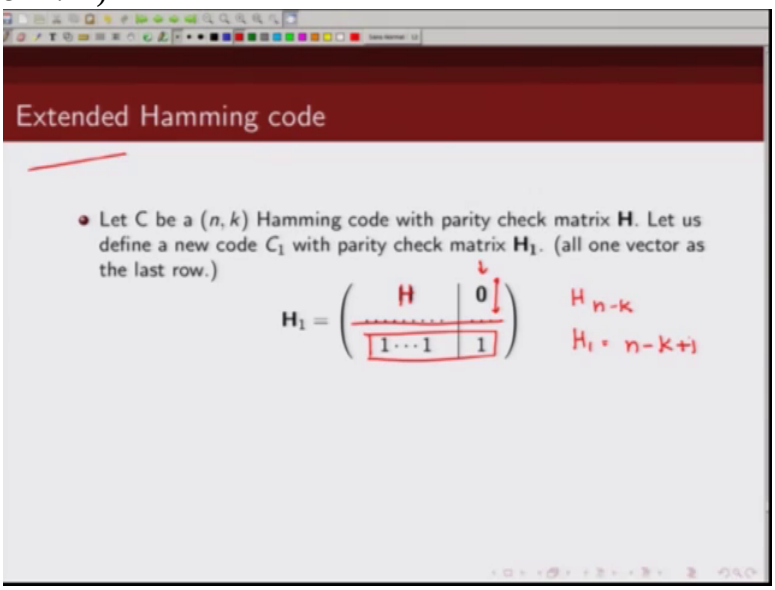
all 1. Now it is not very difficult to see that the rank of the matrix will be if , the rank of the H is n minus k, the rank of this matrix

(Refer Slide Time 47:18)



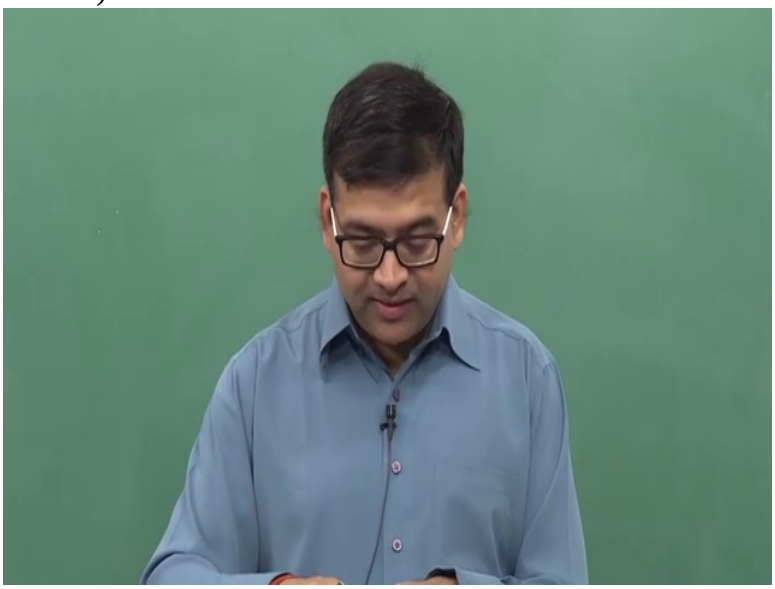
H 1 will be n minus k plus 1

(Refer Slide Time 47:22)



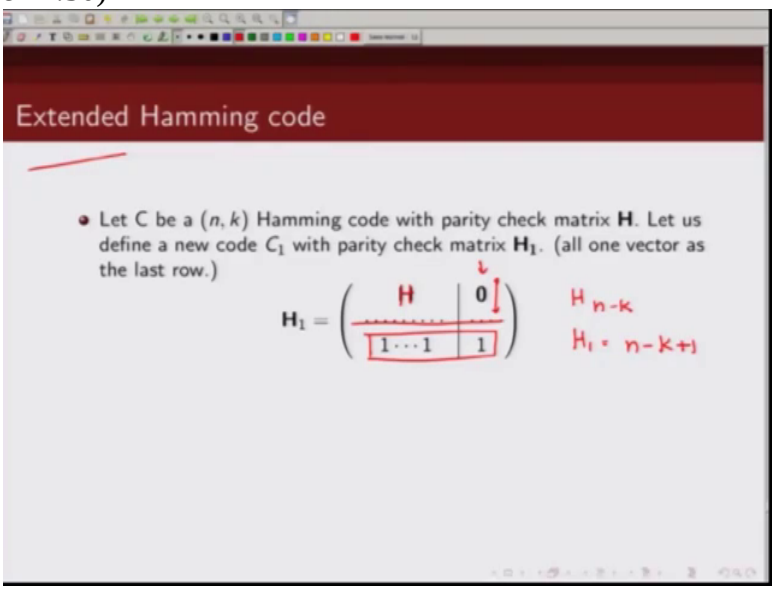
because the parity check matrix

(Refer Slide Time 47:25)



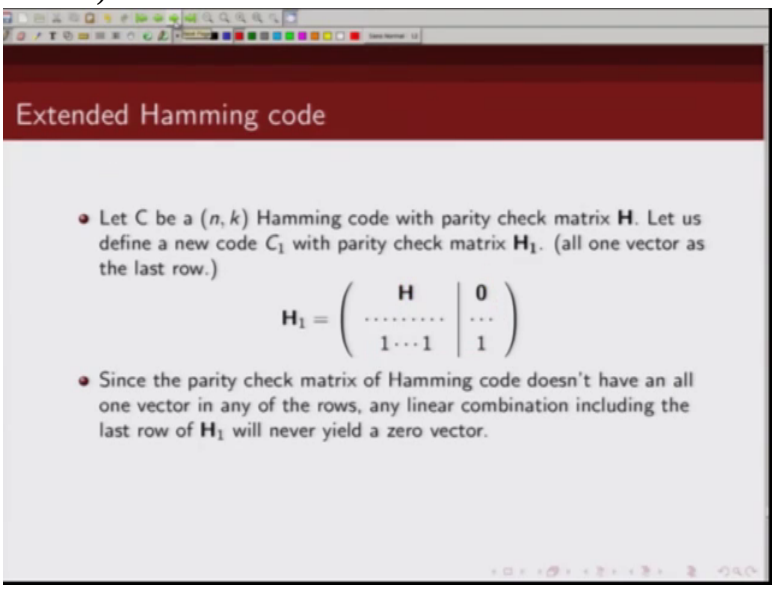
of the Hamming code did not have an all 1 row and then

(Refer Slide Time 47:30)



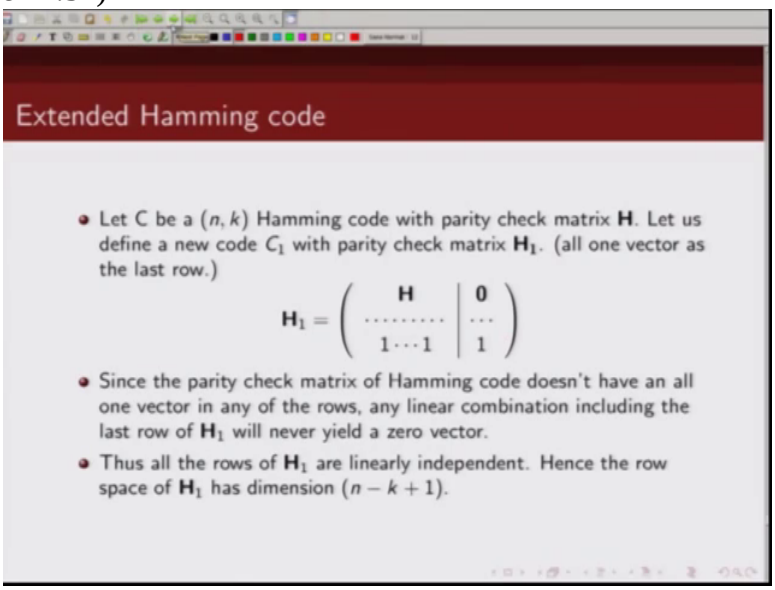
these are all zeroes and this is 1. So any linear combination of this row with these rows basically would not, would not change the, would not decrease

(Refer Slide Time 47:42)



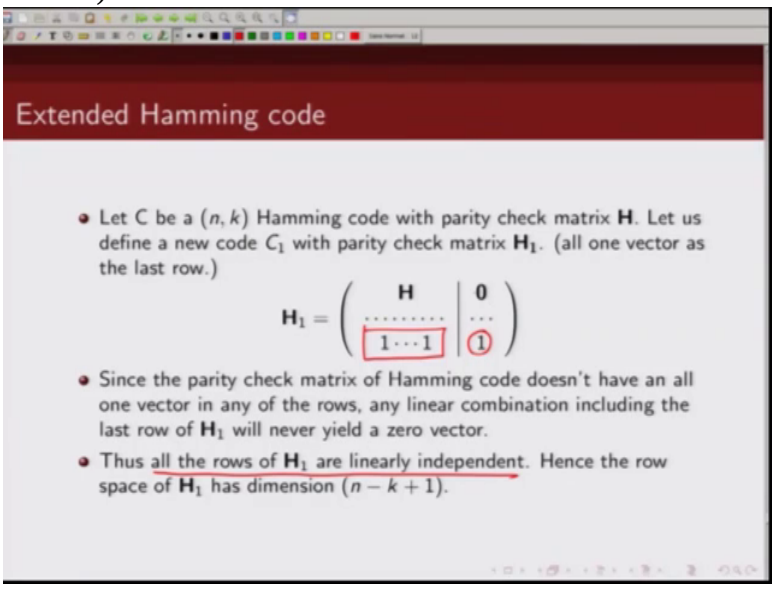
the Hamming distance actually. So this will have, rank of this matrix would be one more than the rank of the original parity check matrix H. So what

(Refer Slide Time 47:52)



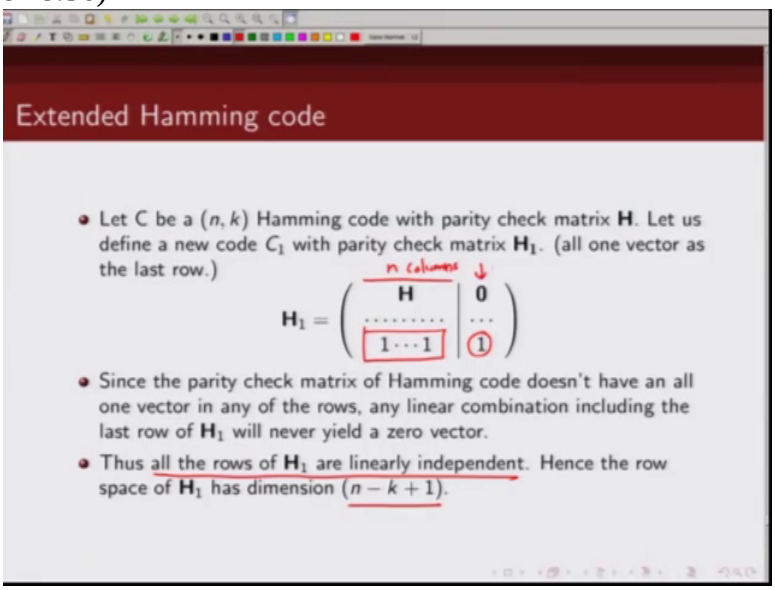
we are saying then is, all rows of this parity check matrix H 1are linearly independent. As I said that is because original H matrix did not have all 1s here and this is 1 here, this is 0 here so if we take linear combination

(Refer Slide Time 48:12)



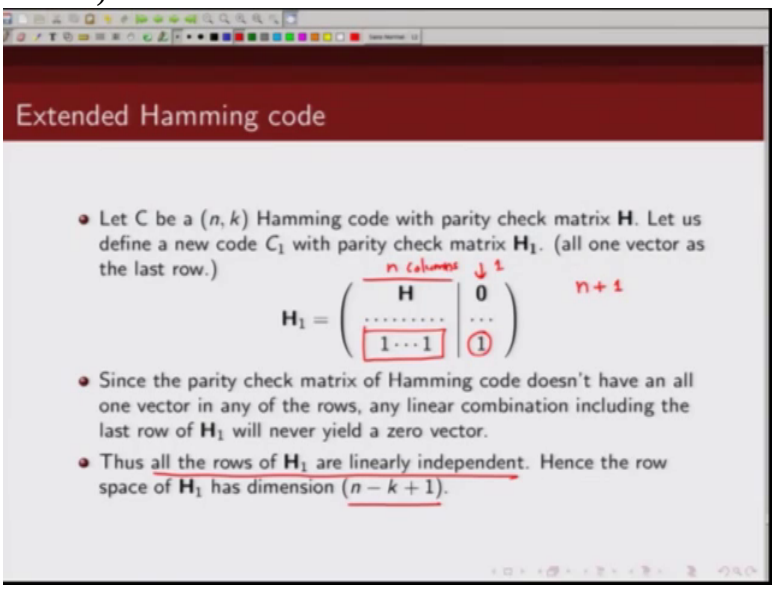
of this with any of the rows of the original parity check matrix they would not be linearly dependent. So all rows are linearly independent, hence the rank of this matrix is n minus k plus 1. And what is the number of columns? The original matrix had n columns. And we added one column, one more column.

(Refer Slide Time 48:36)



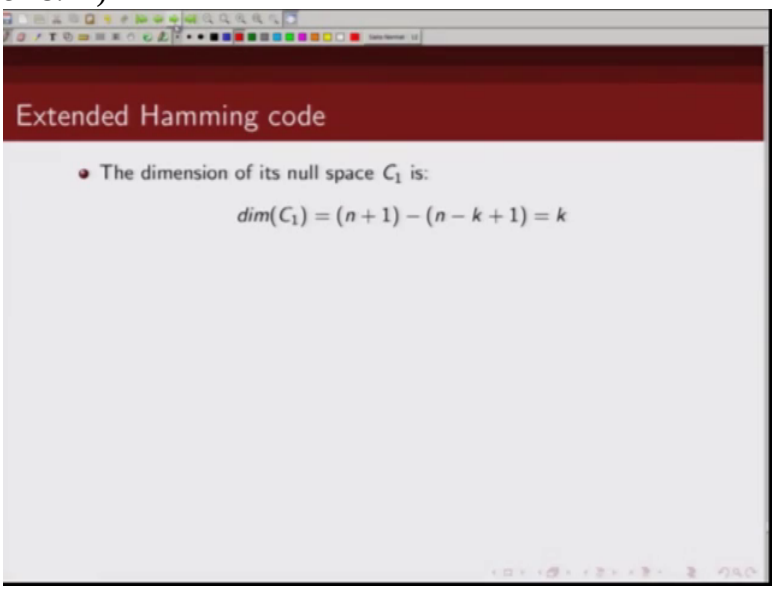
So number of columns here is n plus 1. So this will define the code of length

(Refer Slide Time 48:42)



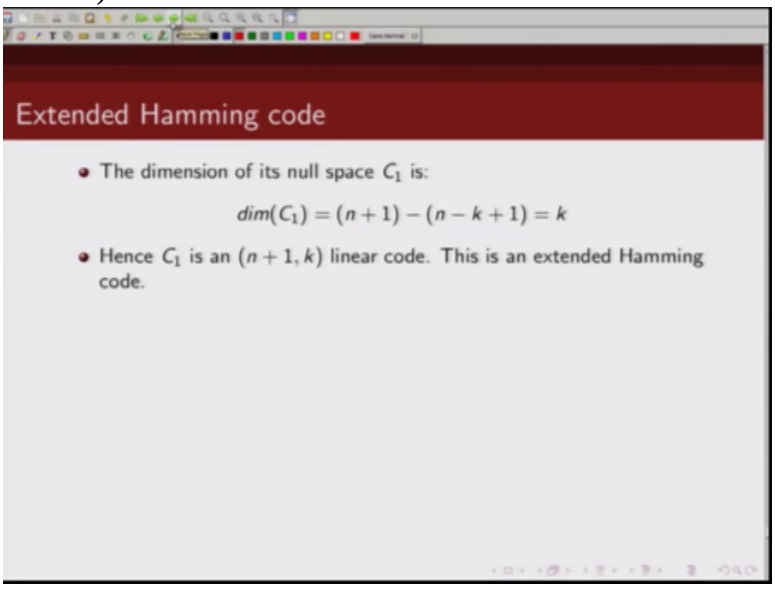
n plus 1. So then we can find out

(Refer Slide Time 48:47)



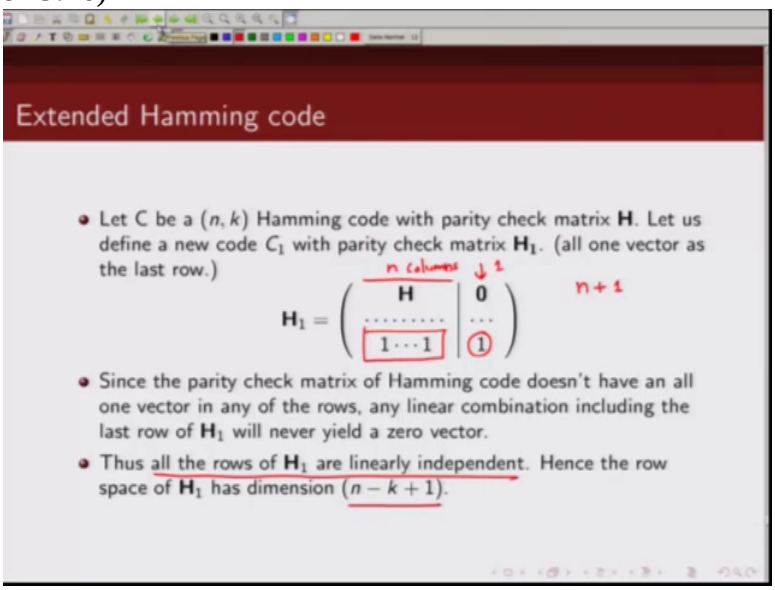
what is the dimension of null space or so, this is number of of coded bits, this is the rank of the new parity check matrix so number of coded, so the dimension of null space is k. So So this will then generate an

(Refer Slide Time 49:04)



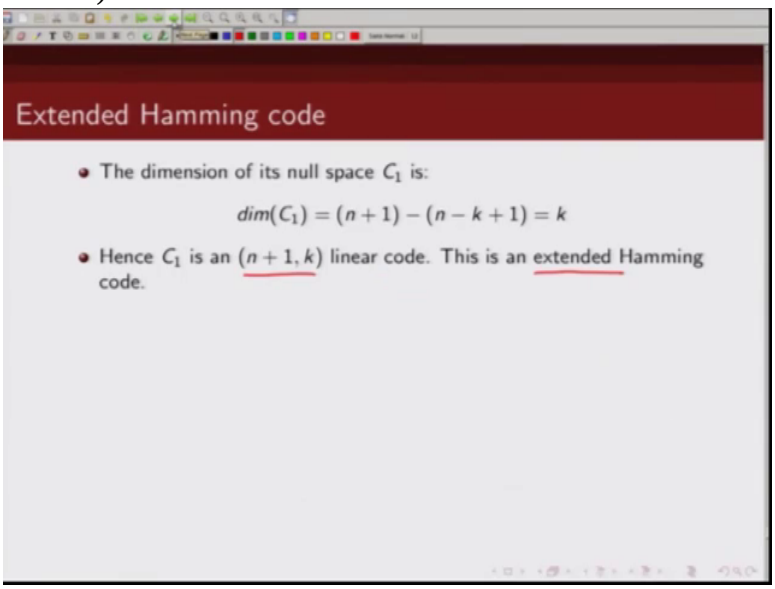
n plus 1 k linear block code. And this is known as extension code or extended Hamming code. If this H is matrix,

(Refer Slide Time 49:16)



parity check matrix with Hamming code, the code described by H 1 which is given by this would be extended Hamming code. And again

(Refer Slide Time 49:28)

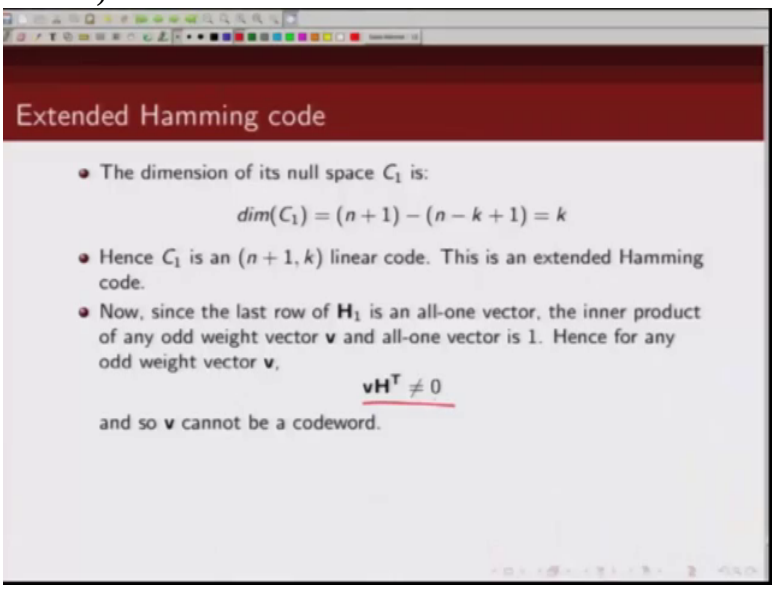


we can see that the extended Hamming code will have only even codewords. That is because

(Refer Slide Time 49:37)

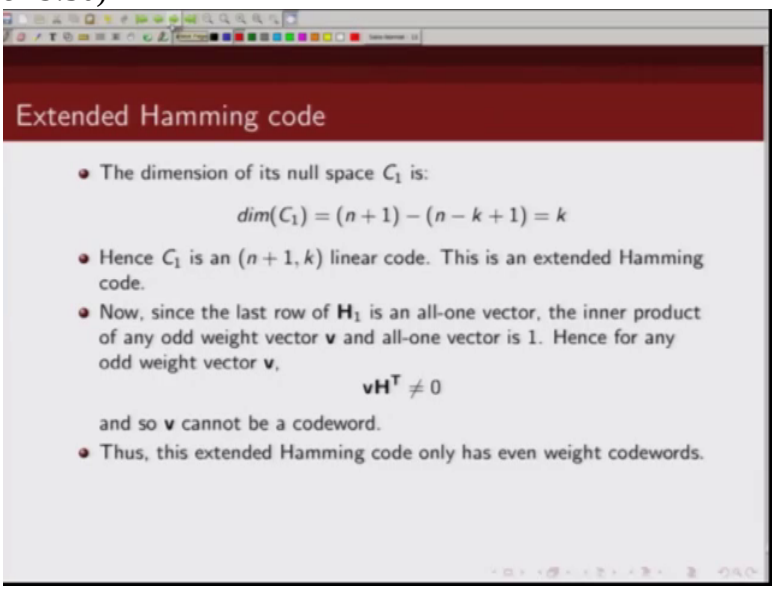
the last row of the parity check contains all 1's. So v H transpose cannot be zero if v

(Refer Slide Time 49:46)



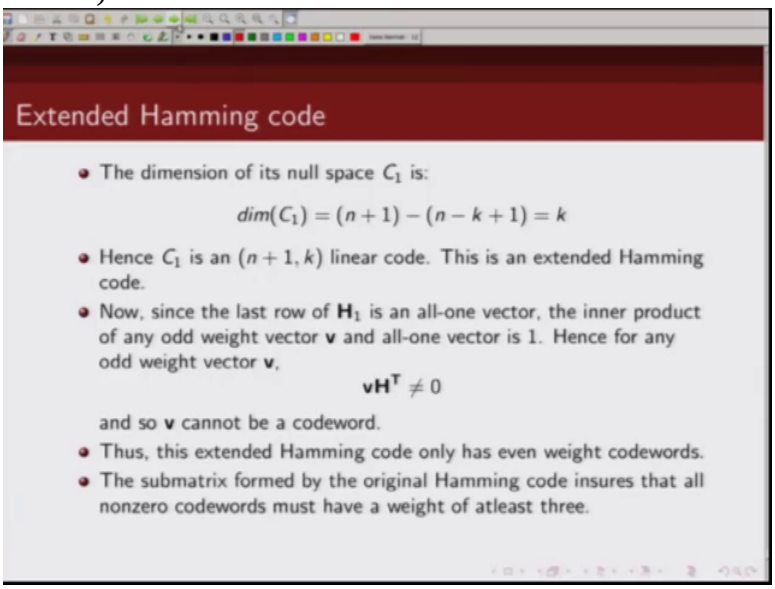
has odd weight. So the

(Refer Slide Time 49:50)



minimum distance of extended Hamming code is 4. Now please

(Refer Slide Time 49:55)

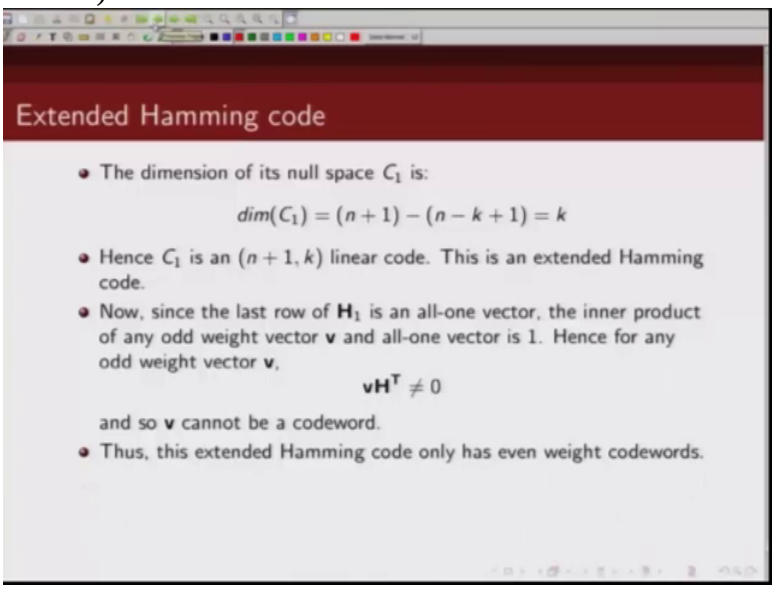


note that some of the techniques what we mentioned here, extension, shortening, expurgated, that's valid for

(Refer Slide Time 50:03)

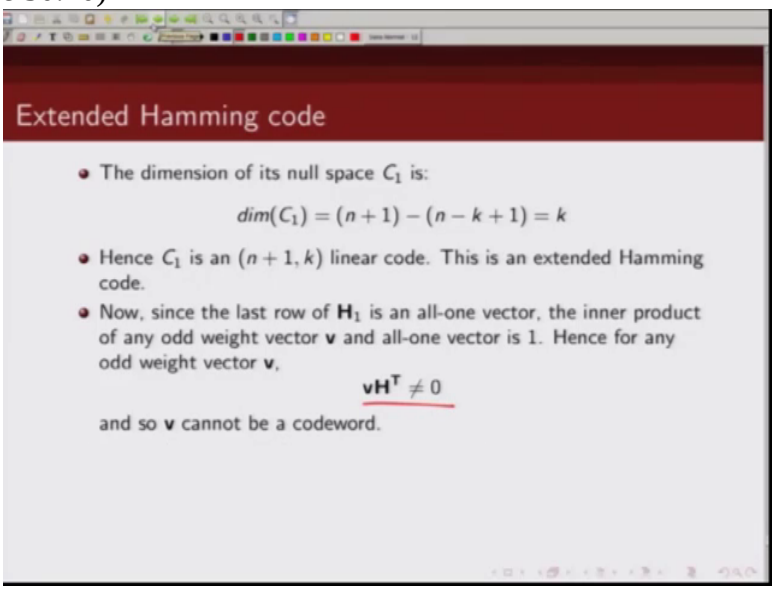
any other linear block codes too

(Refer Slide Time 50:06)



and these are some of the ways in which we can change

(Refer Slide Time 50:10)



the code parameters. So with this we conclude this

(Refer Slide Time 50:14)

lecture, thank you.