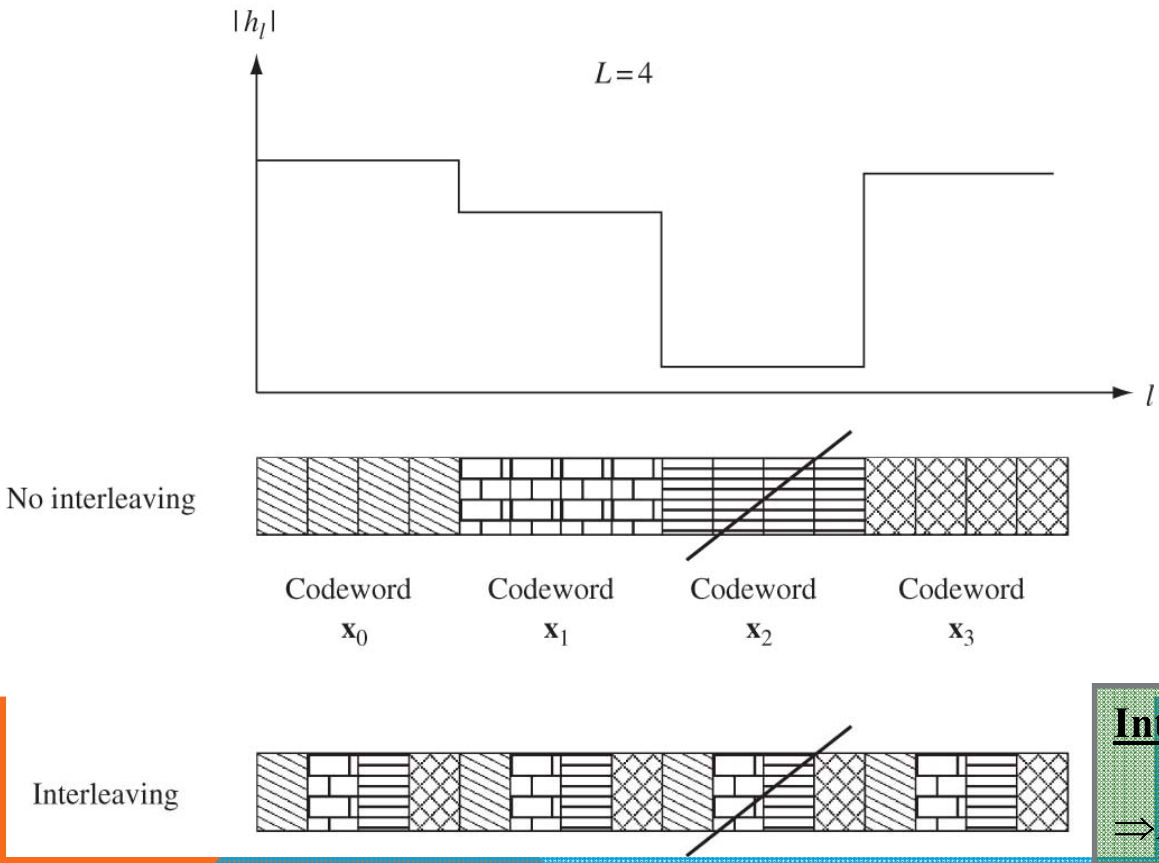
PERFORMANCE OF CODES

CONTEXT: TIME DIVERSITY

Time diversity can be obtained by interleaving and coding over symbols across different coherent time



Channel: time diversity/selectivity, but correlated across successive symbols

(Repetition) Coding… w/o interleaving: a full codeword lost during fade

Interleaving: of sufficient depth: (> coherence time)
$\Rightarrow$At most 1 symbol of codeword lost

Coding alone is not sufficient!

WHAT *IS* CHANNEL CODING?

Transforming signals to improve communications performance by increasing the robustness against channel impairments (noise, interference, fading, ..)

- It is a time-diversity technique, but can be broadly thought of as techniques to make better use of the degrees-of-freedom in channels (eg: space-time codes)

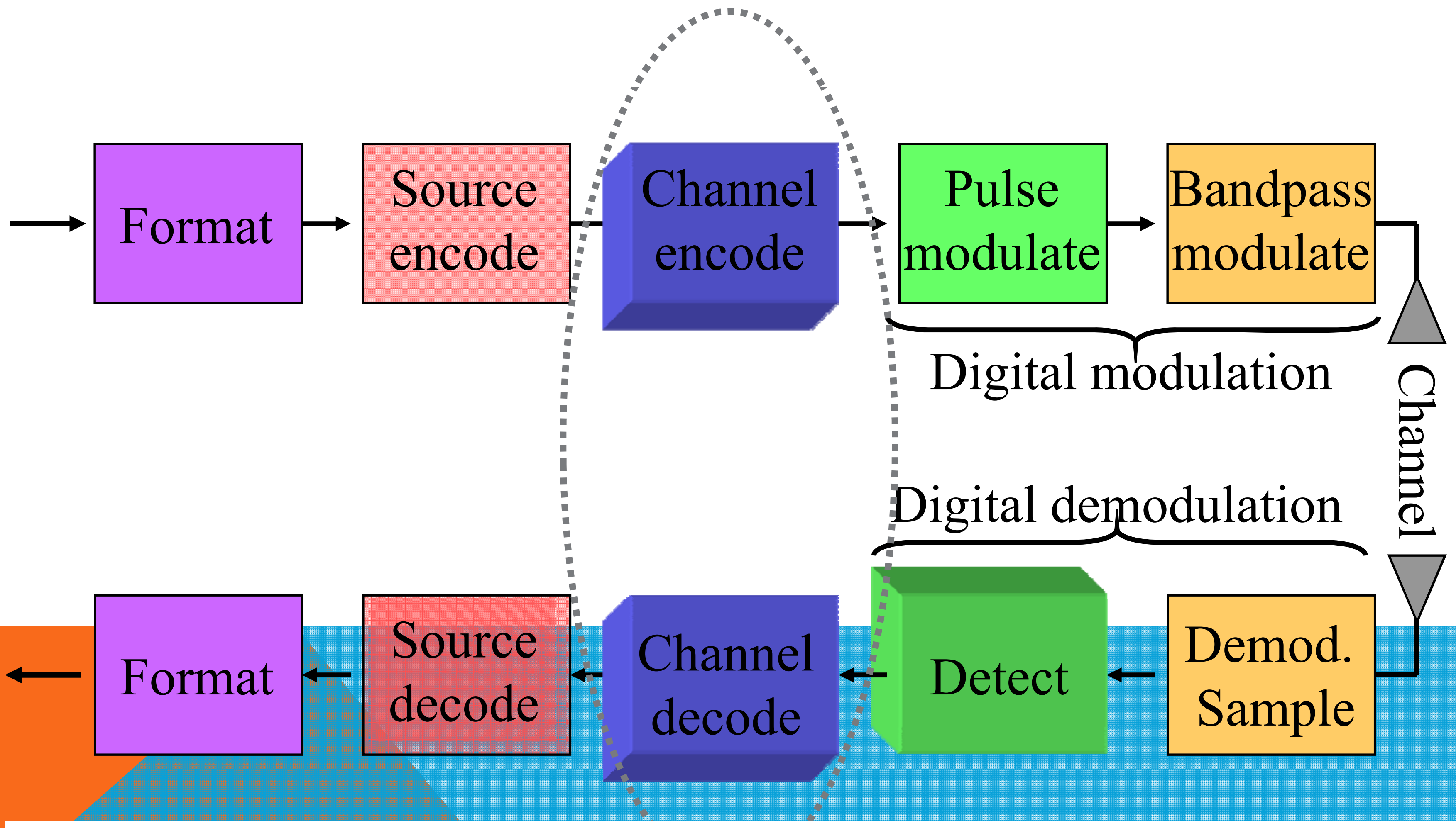
Waveform coding: Transforming waveforms to better waveforms

Structured sequences: Transforming data sequences into better sequences, having structured redundancy.

- "Better" in the sense of making the decision process less subject to errors.
- Introduce constraints on transmitted codewords to have greater "distance" between them

Note: Channel coding was developed in the context of AWGN channels & we shall study them in the same context

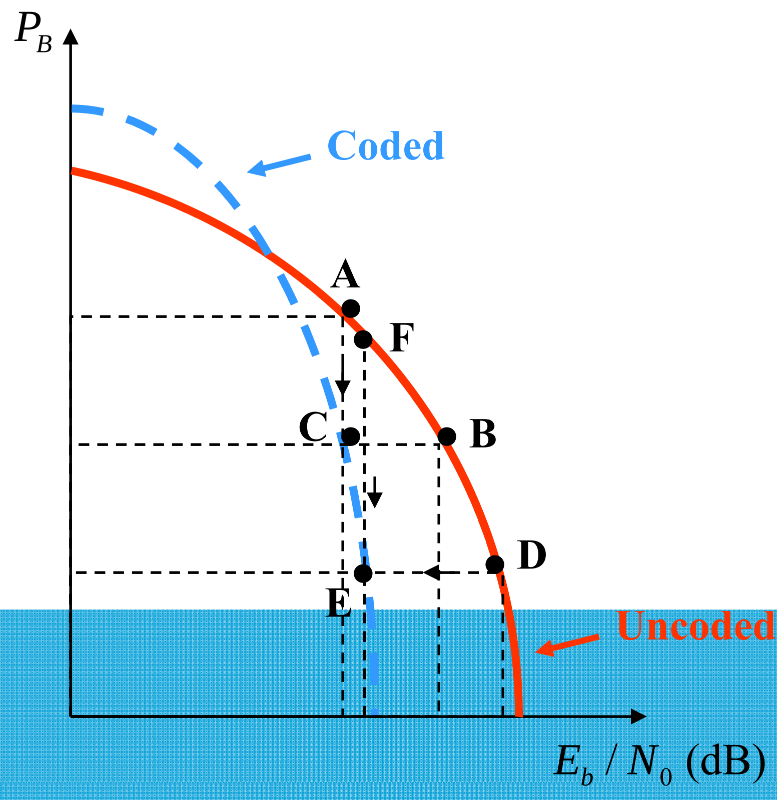
(MODIFIED) BLOCK DIAGRAM



CHANNEL CODING SCHEMES: BLOCK, CONVOLUTIONAL, TURBO

CODING GAIN: THE VALUE OF CODING…

- Error performance vs. bandwidth
- Power vs. bandwidth
- Data rate vs. bandwidth
- Capacity vs. bandwidth

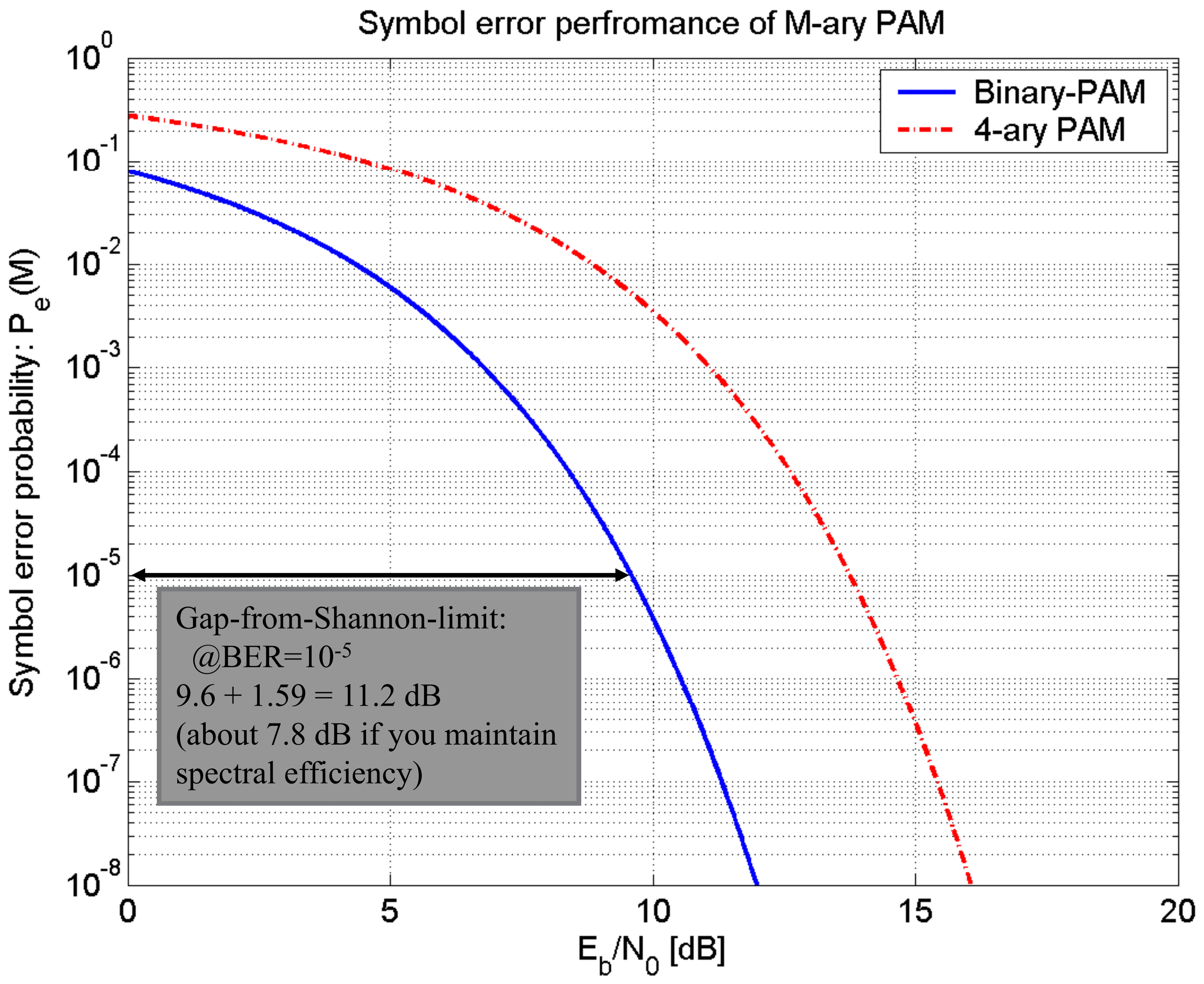


Coding gain:

For a given bit-error probability, the reduction in the Eb/N0 that can be realized through the use of code:

$$G\,[\text{dB}] = \left(\frac{E_b}{N_0}\right)_{\text{u}} [\text{dB}] - \left(\frac{E_b}{N_0}\right)_{\text{c}} [\text{dB}]$$

CODING GAIN POTENTIAL



Gap-from-Shannon-limit:
@BER=10^{-5}
9.6 + 1.59 = 11.2 dB
(about 7.8 dB if you maintain spectral efficiency)

THE ULTIMATE SHANNON LIMIT

Goal: what is min Eb/No for any spectral efficiency ($\rho \to 0$)?

Spectral efficiency $\rho = B/W = \log_2 (1 + SNR)$

- where SNR = E_s/N_o where E_s=energy per symbol
- Or **SNR = $(2^\rho - 1)$**
  - Eb/No = Es/No * (W/B)

    = SNR/ρ

Eb/

Lets try to appreciate what Shannon's bound means by designing some simple codes and comparing it to the Shannon bound

- Fix

Gap-to-capacity @ BER $=10^{-5}$:

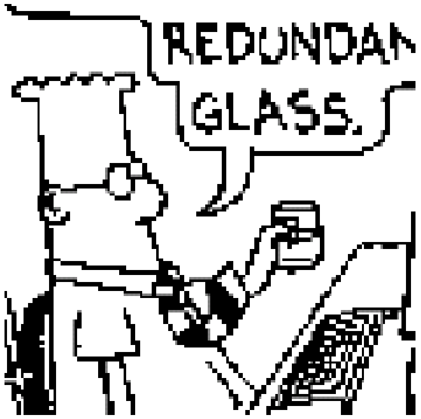
9.6dB + 1.59 = **11.2 dB** (without regard for spectral eff.)

or 9.6 – 1.76 = **7.84 dB** (keeping spectral eff. constant)

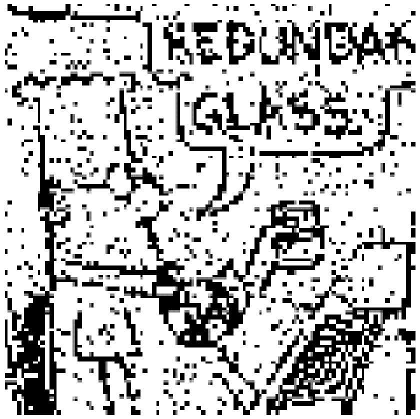
BINARY SYMMETRIC CHANNEL (BSC)

$$\begin{tikzcd} & 0 \arrow[r] \arrow[dr] & 0 & \\ x & & & y \\ & 1 \arrow[r] \arrow[ur] & 1 & \end{tikzcd}$$

$$\begin{aligned} P(y=0 \mid x=0) &= 1-f; & P(y=0 \mid x=1) &= f; \\ P(y=1 \mid x=0) &= f; & P(y=1 \mid x=1) &= 1-f. \end{aligned}$$



$$\begin{tikzcd} 0 \arrow[r, "(1-f)"] \arrow[dr] & 0 \\ 1 \arrow[r, "(1-f)"'] \arrow[ur, "f"'] & 1 \end{tikzcd}$$



Given a BER (*f*), we can construct a BSC with this BER…

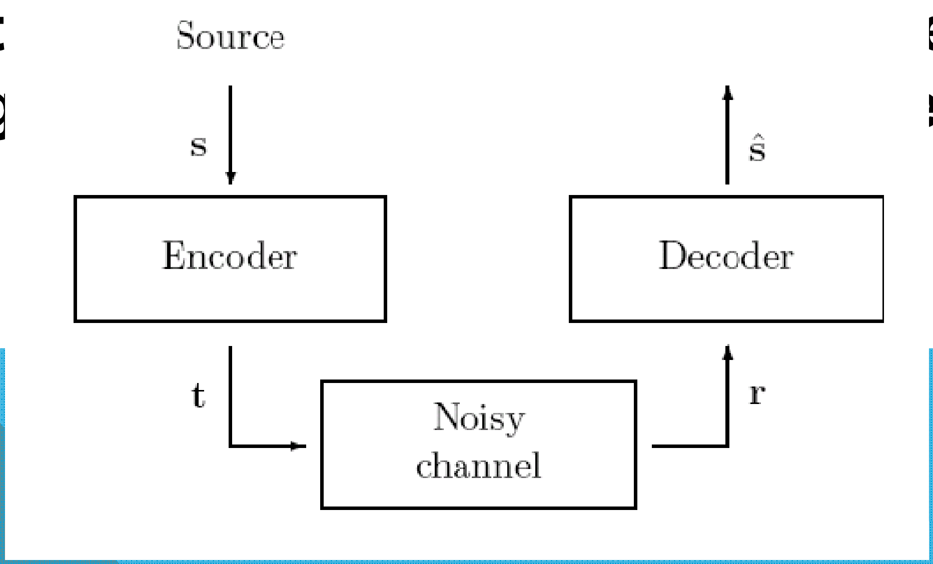
RELIABLE DISK DRIVE APPLICATION

We want to build a disk drive and write a GB/day for 10 years.

- => desired BER: 10^{-15}

<u>Physical</u> solution: use more reliable components, reduce noise

<u>System</u> solut **ect/correct**
errors (eng **channels)**



REPETITION CODE (R3) & MAJORITY VOTE DECODING

Source sequence $\mathbf{s}$	Transmitted sequence $\mathbf{t}$
0	000
1	111

source message $\mathbf{s} = 0\ 0\ 1\ 0\ 1\ 1\ 0$

AWGN:

$$\begin{array}{cccccccc}
\mathbf{s} & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
\mathbf{t} & \overbrace{000} & \overbrace{000} & \overbrace{111} & \overbrace{000} & \overbrace{111} & \overbrace{111} & \overbrace{000} \\
\mathbf{n} & 000 & 001 & 000 & 000 & 101 & 000 & 000 \\
\hline
\mathbf{r} & 000 & 001 & 111 & 000 & 010 & 111 & 000
\end{array}$$

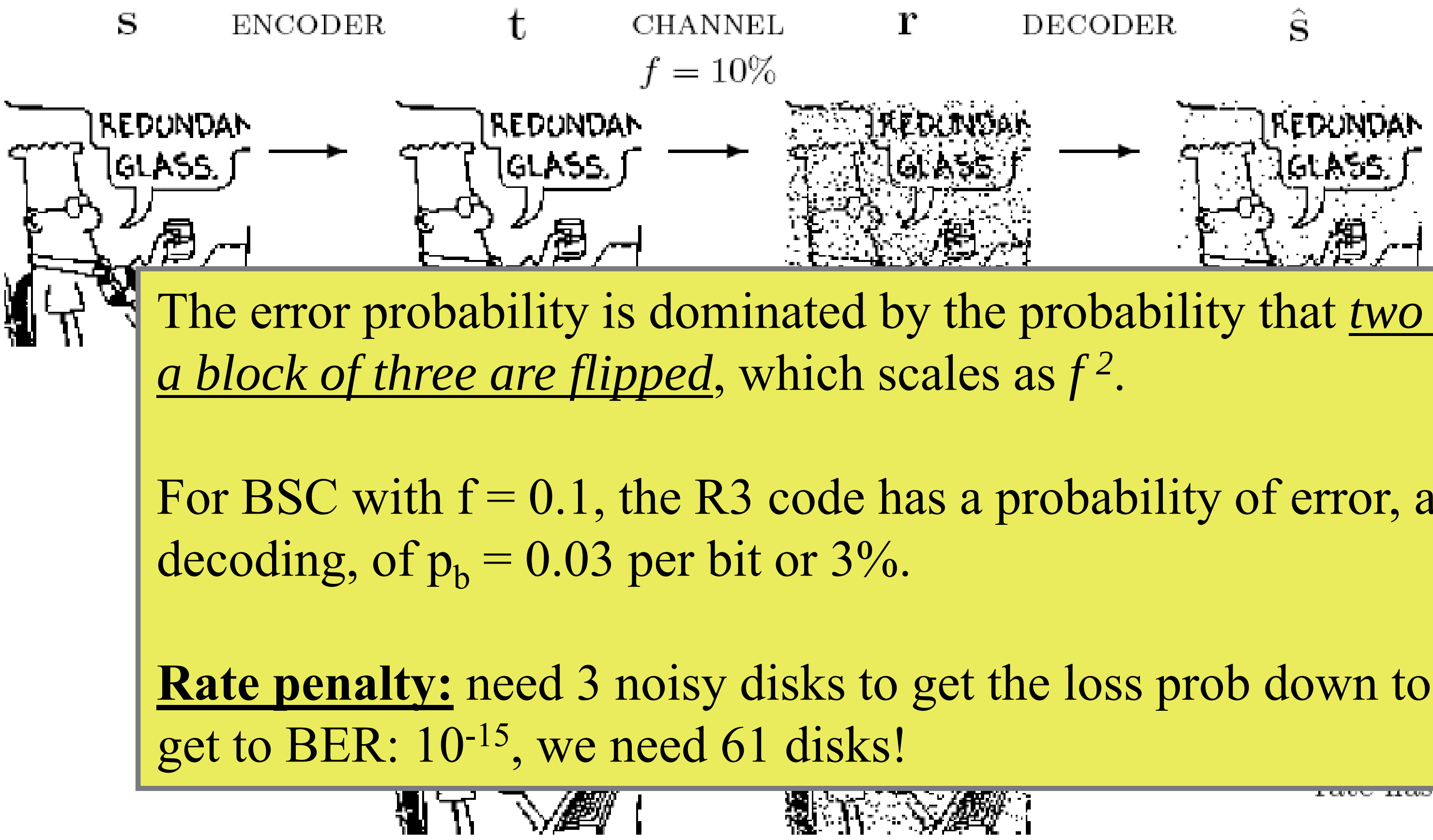
Received sequence $\mathbf{r}$	Likelihood ratio $\frac{P(\mathbf{r}\,\vert\,s=1)}{P(\mathbf{r}\,\vert\,s=0)}$	Decoded sequence $\hat{s}$
000	γ^{-3}	0
001	γ^{-1}	0
010	γ^{-1}	0
100	γ^{-1}	0
101	γ^{1}	1
110	γ^{1}	1
011	γ^{1}	1
111	γ^{3}	1

Algorithm 1.9. Majority-vote decoding algorithm for R_3. Also shown are the likelihood ratios (1.23), assuming the channel is a binary symmetric channel; $\gamma \equiv (1-f)/f$.

$\gamma \equiv \frac{(1-f)}{f}$ is greater than 1, since $f < 0.5$, so the winning hypothesis is the one with the most 'votes', each vote counting for a factor of γ in the likelihood ratio.

PERFORMANCE OF R3

s	0	0	1	0	1	1	0
t	000	000	111	000	111	111	000
n	000	001	000	000	101	000	000
r	000	001	111	000	010	111	000
ŝ	0	0	1	0	0	1	0
corrected errors		⋆					
undetected errors					⋆		



The error probability is dominated by the probability that *__two bits in a block of three are flipped__*, which scales as f^2.

For BSC with f = 0.1, the R3 code has a probability of error, after decoding, of $p_b = 0.03$ per bit or 3%.

__Rate penalty:__ need 3 noisy disks to get the loss prob down to 3%. To get to BER: 10^{-15}, we need 61 disks!

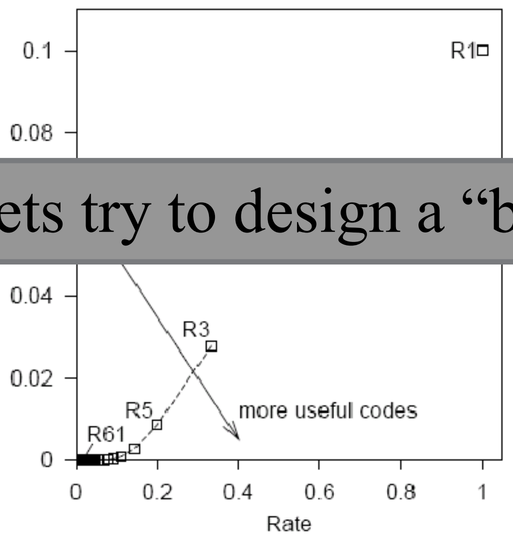
g 10 000

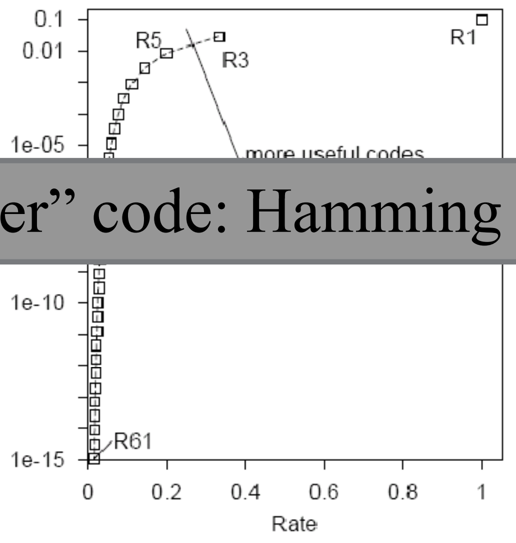
$f = 10\%$
nd the
lgorithm.
led bit
3%; the

CODING: RATE-BER TRADEOFF?

Repetition code R3:

s	0	0	1	0	1	1	0
t	000	000	111	000	111	111	000
n	000	001	000	000	101	000	000
r	000	001	111	000	010	111	000
$\hat{s}$	0	0	1	0	0	1	0
corrected errors		⋆					
undetected errors					⋆		





Lets try to design a "better" code: Hamming Code

Shannon: The perception that there is a necessary tradeoff between Rate and BER is illusory! It is not true upto a critical rate, the channel capacity!

[illegible]

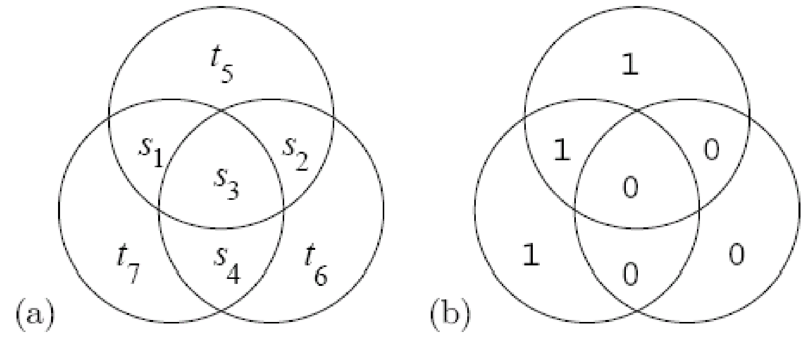
HAMMING CODE: LINEAR BLOCK CODE

A <u>block code</u> is a rule for converting a sequence of source bits s, of length K, say, into a transmitted sequence t of length N bits.

In a linear block code, the extra N-K bits are *<u>linear functions</u>* of the original K bits; these extra bits are called *<u>parity-check</u>* bits.

<u>(7, 4) Hamming code:</u> transmits N = 7 bits for every K = 4 source bits.

- The first four transmitted bits, $t_1t_2t_3t_4$, are set equal to the four source bits, $s_1s_2s_3s_4$.
- The parity ... parity within each circle (se



HAMMING CODE: (CONTD)

s	t	s	t	s	t	s	t
0000	0000000	0100	0100110	1000	1000101	1100	1100011
0001	0001011	0101	0101101	1001	1001110	1101	1101000
0010	0010111	0110	0110001	1010	1010010	1110	1110100
0011	0011100	0111	0111010	1011	1011001	1111	1111111

Table 1.14. The sixteen codewords $\{\mathbf{t}\}$ of the (7, 4) Hamming code. Any pair of codewords differ from each other in at least three bits.

Because the Hamming code is a linear code, it can be written compactly in terms of matrices as follows. The transmitted codeword $\mathbf{t}$ is obtained from the source sequence $\mathbf{s}$ by a linear operation,

$$\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}, \tag{1.25}$$

where $\mathbf{G}$ is the *generator matrix* of the code,

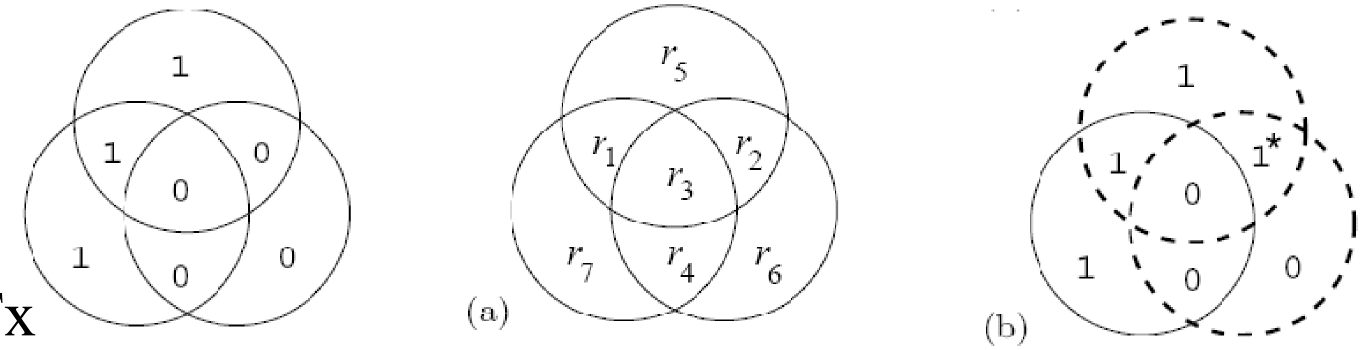
$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \tag{1.26}$$

HAMMING CODE: SYNDROME DECODING

If channel is BSC and all source vectors are equiprobable, then…

- … the optimal decoder identifies the source vector **s** whose encoding **t**(**s**) differs from the received vector **r** in the fewest bits.
- Similar to "closest-distance" decision rule seen in demodulation!

Can we do it more efficiently? Yes: <u>Syndrome decoding</u>



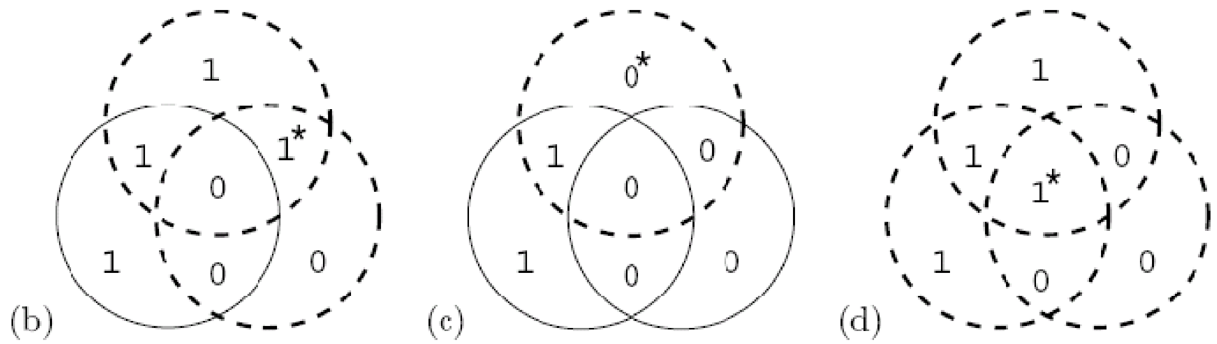
The decoding task is to find the smallest set of flipped bits that can account for these violations of the parity rules.

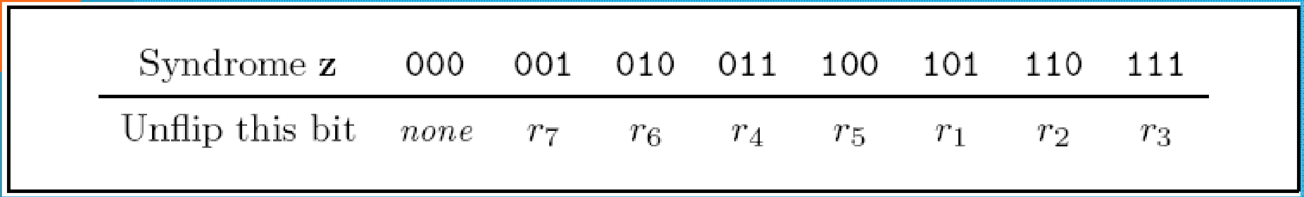
[The *<u>pattern of violations of the parity checks is called the syndrome</u>*: the syndrome above is **z = (1, 1, 0),** because the first two circles are `unhappy' (parity 1) and the third circle is `happy‘ (parity 0).]

SYNDROME DECODING (CONTD)

Can we find a unique bit that lies _inside_ all the `unhappy' circles and _outside_ all the `happy' circles?

- If so, the flipping of that bit would account for the observed syndrome.





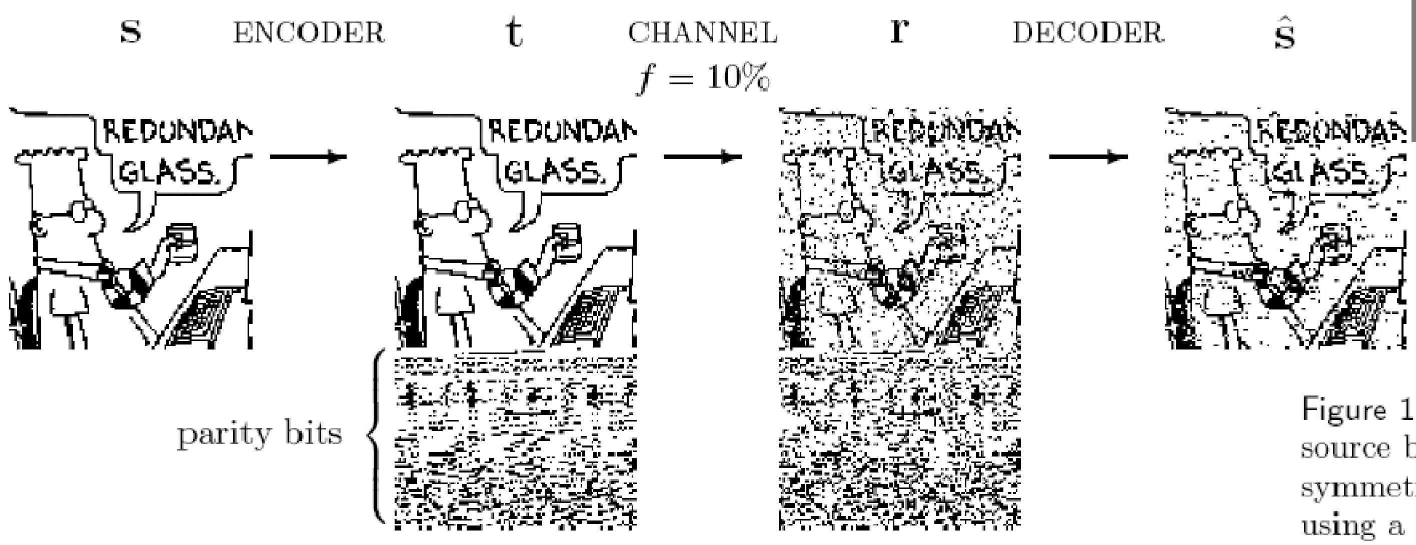
Syndrome $\mathbf{z}$	000	001	010	011	100	101	110	111
Unflip this bit	*none*	r_7	r_6	r_4	r_5	r_1	r_2	r_3

HAMMING CODE: PERFORMANCE

A decoding error will occur whenever the noise has flipped more than one bit in a block of seven.

The probability scales as O(f^2), as did the probability of error for the repetition code R3; but Hamming code has a greater rate, R = 4/7.

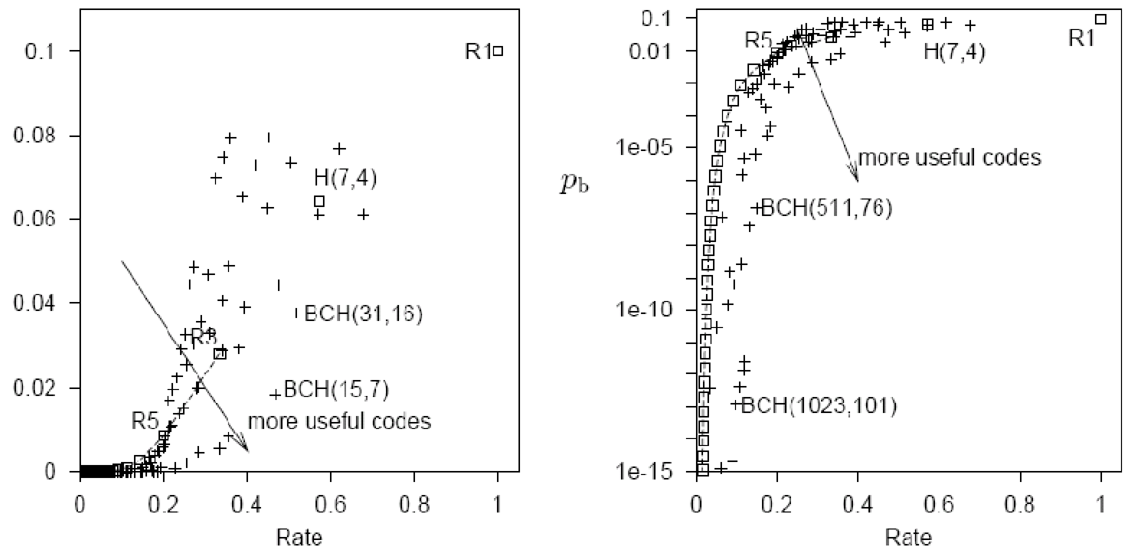
Dilbert Test: About 7% of the decoded bits are in error. The residual errors are *correlated*: often two or three successive decoded bits are flipped

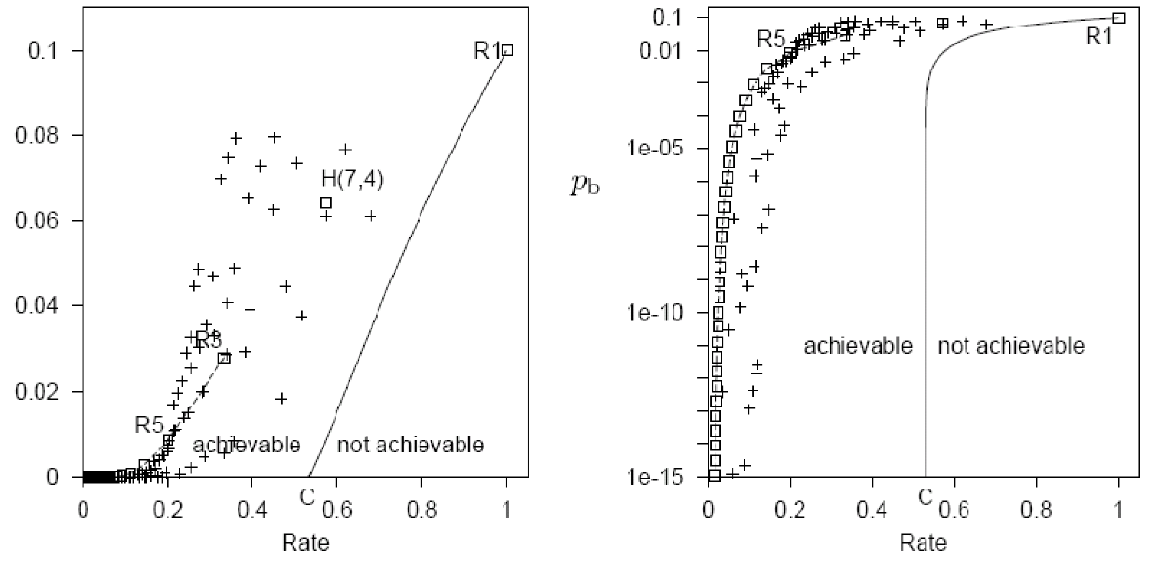


$$p_{\rm b} = \frac{1}{K} \sum_{k=1}^{K} P(\hat{s}_k \neq s_k).$$

Figure 1.17. Transmitting 10 000 source bits over a binary symmetric channel with $f = 10\%$ using a (7, 4) Hamming code. The probability of decoded bit error is about 7%.

SHANNON'S LEGACY: RATE-RELIABILITY OF CODES



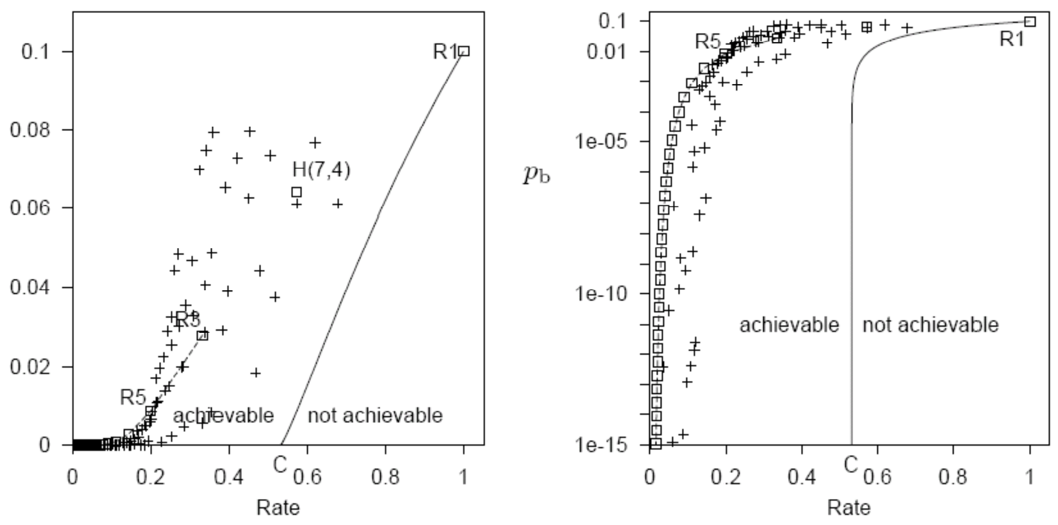


<u>Noisy-channel coding theorem</u>: defines achievable rate/reliability regions

Note: you can get BER as low as desired by designing an appropriate code within

The equation defining the Shannon limit (the solid curve) is $R = C/(1 - H_2(p_b))$, where C and H_2 are defined in equation (1.35).

SHANNON LEGACY (CONTD)



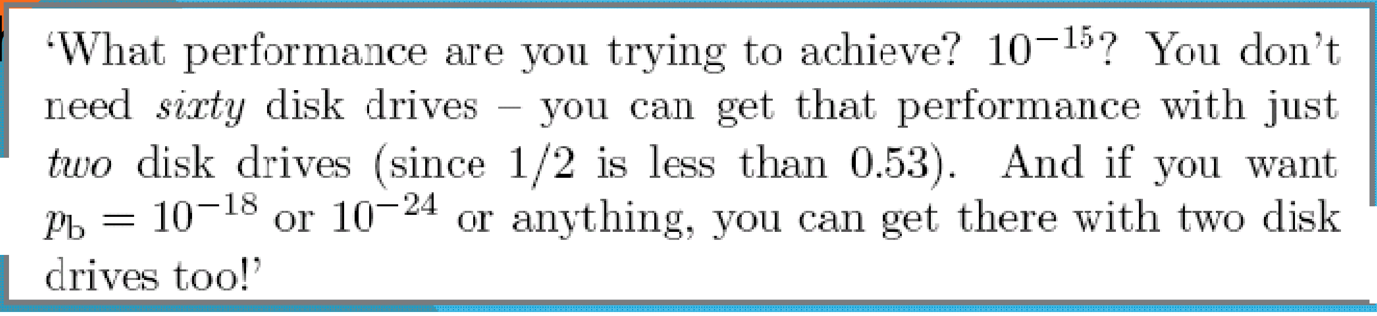
The equation defining the Shannon limit (the solid curve) is $R = C/(1 - H_2(p_b))$, where C and H_2 are defined in equation (1.35).

The maximum rate at which communication is possible with *arbitrarily small* p_b is called the *capacity* of the channel.

$$C(f) = 1 - H_2(f) = 1 - \left[f \log_2 \frac{1}{f} + (1-f) \log_2 \frac{1}{1-f} \right];$$

BSC(f) capacity:

f = 0.1 h


'What performance are you trying to achieve? 10^{-15}? You don't need *sixty* disk drives – you can get that performance with just *two* disk drives (since 1/2 is less than 0.53). And if you want $p_b = 10^{-18}$ or 10^{-24} or anything, you can get there with two disk drives too!'

CAVEATS & REMARKS

Strictly, the above statements might not be quite right:

Shannon proved his noisy-channel coding theorem by studying sequences of block codes with *ever-increasing block lengths*, and the required block length might be bigger than a gigabyte (the size of our disk drive),

… in which case, Shannon might say `well, you can't do it with those *tiny* disk drives, but if you had two noisy *terabyte* drives, you could make a single high-quality terabyte drive from them'.

Information theory addresses both the *limitations* and the *possibilities* of communication.

- Reliable communication at any rate beyond the capacity is *impossible*, and that reliable communication at *all rates up to capacity is possible*.

GENERALIZE: LINEAR CODING/SYNDROME DECODING

The first four received bits, $r_1r_2r_3r_4$, purport to be the four source bits; and the received bits $r_5r_6r_7$ purport to be the parities of the source bits, as defined by the generator matrix G.

- Evaluate the three parity-check bits for the received bits, $r_1r_2r_3r_4$, and see whether they match the three received bits, $r_5r_6r_7$.

The differences (modulo 2) between these two triplets are called the syndrome of the received vector.

- If the syndrome is zero then the received vector is a codeword, and the most probable decoding is given by reading out its first four bits.
- If the syndrome is non-zero, then the noise sequence for this block was non-zero, and the *syndrome is our pointer* to the most probable error pattern.

LINEAR CODING/SYNDROME DECODING (CONTD)

Coding: $\boxed{\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s},}$

G is the *generator matrix* of the code $\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \mathbf{I}_4 \\ \mathbf{P} \end{bmatrix}$$

- **Received vector & Syndome:** $\mathbf{r} = \mathbf{G}^{\mathsf{T}}\mathbf{s} + \mathbf{n}$

th tisfy

Lets now build linear codes from ground up (first principles)

The syndrome-decoding problem is to find the most probable noise vector **n** satisfying the equation $\boxed{\mathbf{H}\mathbf{n} = \mathbf{z}.}$

$$\mathbf{H}\mathbf{t} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Parity Check Matrix H:

the *parity-check matrix* **H** is given by $\mathbf{H} = \begin{bmatrix} -\mathbf{P} & \mathbf{I}_3 \end{bmatrix}$

in modulo 2 arithmetic, $-1 \equiv 1$, so

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

SOME DEFINITIONS

Binary field :

- The set {0,1}, under modulo 2 binary addition and multiplication forms a field.

Addition	Multiplication
$0 \oplus 0 = 0$	$0 \cdot 0 = 0$
$0 \oplus 1 = 1$	$0 \cdot 1 = 0$
$1 \oplus 0 = 1$	$1 \cdot 0 = 0$
$1 \oplus 1 = 0$	$1 \cdot 1 = 1$

- Binary field is also called *Galois field, GF(2).*

DEFINITIONS: FIELDS

Fields :

- Let F be a set of objects on which two operations ‘+’ and ‘.’ are defined.
- F is said to be a field if and only if

1. F forms a **commutative group under +** operation. The additive identity element is labeled “0”.

$$\forall a,b \in F \Rightarrow a+b=b+a \in F$$

2. F-{0} forms a **commutative group under .** operation. The multiplicative identity element is labeled “1”.

$$\forall a,b \in F \Rightarrow a \cdot b = b \cdot a \in F$$

The operations “+” and “.” **distribute**:

$$a \cdot (b+c) = (a \cdot b) + (a \cdot c)$$

DEFINITIONS: VECTOR SPACE *OVER* FIELDS

Vector space: (*note: it mixes vectors and scalars*)

- Let V be a set of **vectors** and F a fields of elements called **scalars**. V forms a vector space over F if:

1. **Commutative**:
2. **Closure**:
3. **Distributive**:
4. **Associative**:
5. **Identity Element**:

$$\forall \mathbf{u}, \mathbf{v} \in V \Rightarrow \mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u} \in F$$

$$\forall a \in F, \forall \mathbf{v} \in \mathbf{V} \Rightarrow a \cdot \mathbf{v} = \mathbf{u} \in \mathbf{V}$$

$$(a+b) \cdot \mathbf{v} = a \cdot \mathbf{v} + b \cdot \mathbf{v} \quad \text{and} \quad a \cdot (\mathbf{u} + \mathbf{v}) = a \cdot \mathbf{u} + a \cdot \mathbf{v}$$

$$\forall a, b \in F, \forall \mathbf{v} \in V \Rightarrow (a \cdot b) \cdot \mathbf{v} = a \cdot (b \cdot \mathbf{v})$$

$$\forall \mathbf{v} \in \mathbf{V},\ 1 \cdot \mathbf{v} = \mathbf{v}$$

VECTOR SPACES, SUBSPACES

Examples of vector spaces V_n

- The set of binary n-tuples, denoted by

$$V_4 = \{(0000),(0001),(0010),(0011),(0100),(0101),(0111),\\ (1000),(1001),(1010),(1011),(1100),(1101),(1111)\}$$

Vector subspace:

- A subset S of the vector space V_n is called a **subspace** if:
  - **Zero**: The all-zero vector is in S.
  - **Closure**: The sum of any two vectors in S is also in S.

Example: $\{(0000),(0101),(1010),(1111)\}$ is a subspace of V_4.

SPAN, BASES…

Spanning set:

- A collection of vectors $G = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ the linear combinations of which include all vectors in a vector space V, is said to be a spanning set for V or to span V.
  - Example:

$$\{(1000),(0110),(1100),(0011),(1001)\} \text{ spans } V_4.$$

Bases:

- A spanning set for V that has ***minimal cardinality*** is called a basis for V.
  - Cardinality of a set is the number of objects in the set.
  - Example:

$$\{(1000),(0100),(0010),(0001)\} \text{ is a basis for } V_4.$$

LINEAR BLOCK CODES ARE JUST SUBSPACES!

Linear block code (n,k)

- A set $C \subset V_n$ with cardinality 2^k is called a linear block code ***<u>if, and only if, it is a subspace</u>*** of the vector space V_n.

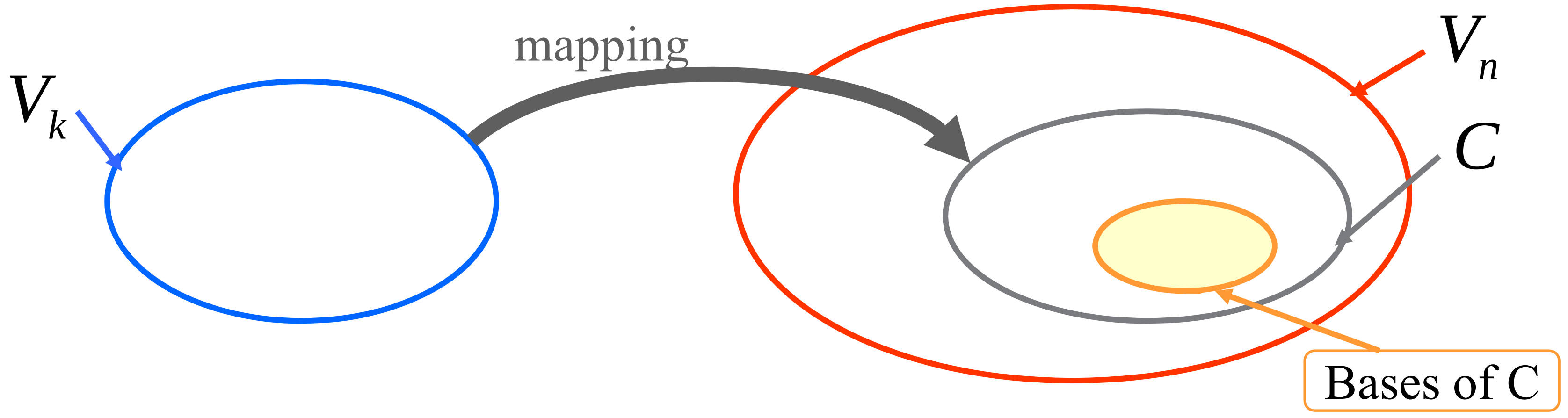
Members of C are called <u>codewords</u>.

The all-zero codeword is a codeword.

$$V_k \rightarrow C \subset V_n$$

Any *<u>linear combination of code-words is a codeword</u>*.

LINEAR BLOCK CODES – CONT’D



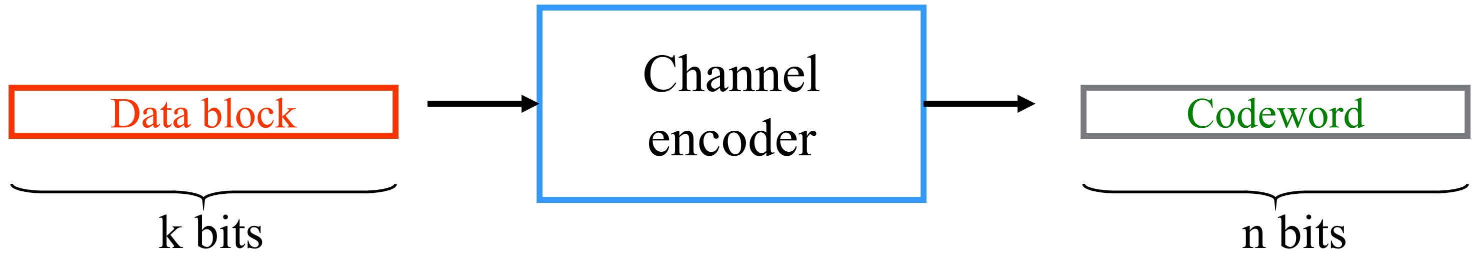
LINEAR BLOCK CODES – CONT'D

The information bit stream is chopped into blocks of k bits.

Each block is encoded to a larger block of n bits.

The coded bits are modulated and sent over channel.

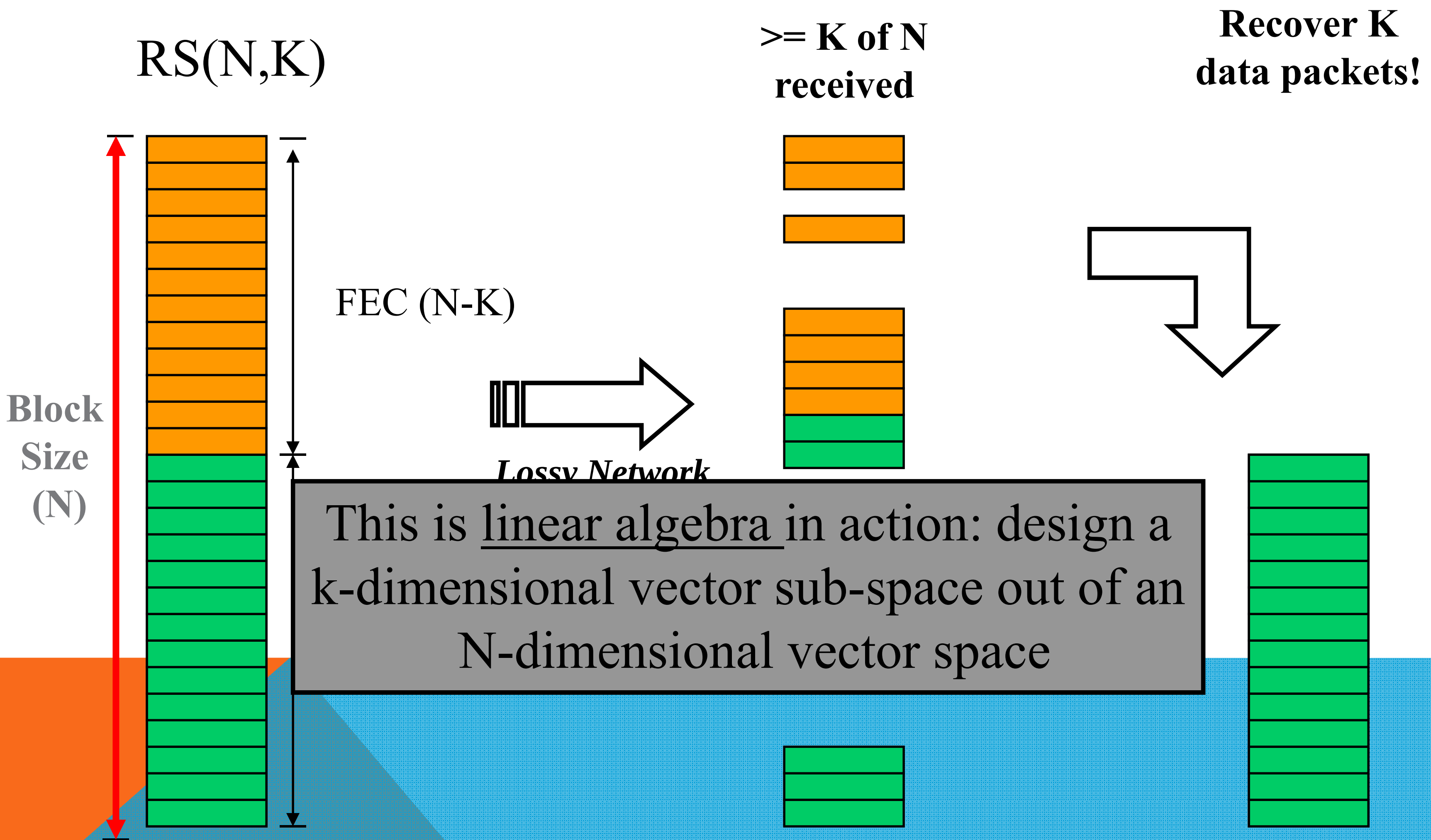
The reverse procedure is done at the receiver.



$n\text{-}k$ Redundant bits

$R_c = \frac{k}{n}$ Code rate

RECALL: REED-SOLOMON RS(N,K): LINEAR ALGEBRA IN ACTION…



This is <u>linear algebra </u>in action: design a k-dimensional vector sub-space out of an N-dimensional vector space

LINEAR BLOCK CODES – CONT'D

The *Hamming weight* (*w*) of vector U, denoted by w(U), is the number of non-zero elements in U.

The *Hamming distance* (*d*) between two vectors U and V, is the number of elements in which they differ.

The minimum distance of a block code is

$$d(\mathbf{U},\mathbf{V}) = w(\mathbf{U} \oplus \mathbf{V})$$

$$d_{\min} = \min_{i \neq j} d(\mathbf{U}_i, \mathbf{U}_j) = \min_i w(\mathbf{U}_i)$$

LINEAR BLOCK CODES – CONT'D

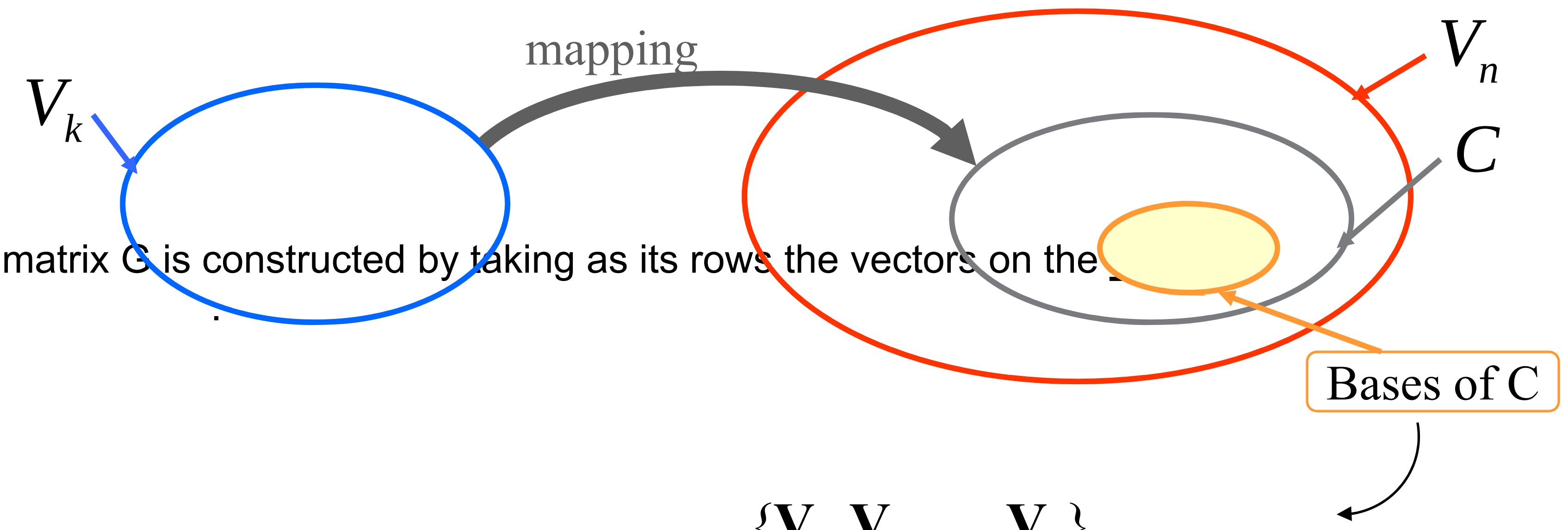
Error detection capability is given by

$$e = d_{\min} - 1$$

Error correcting capability t of a code, which is defined as the maximum number of guaranteed correctable errors per codeword, is

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

LINEAR BLOCK CODES –CONT'D



- A matrix G is constructed by taking as its rows the vectors on the

$$\{\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_k\}$$

$$\mathbf{G} = \begin{bmatrix} \mathbf{V}_1 \\ \vdots \\ \mathbf{V}_k \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & & \ddots & \vdots \\ v_{k1} & v_{k2} & \cdots & v_{kn} \end{bmatrix}$$

LINEAR BLOCK CODES – CONT'D

Encoding in (n,k) block code

$$\boxed{\mathbf{U} = \mathbf{m}\mathbf{G}}$$

- The rows of G are linearly independent

$$(u_1, u_2, \ldots, u_n) = (m_1, m_2, \ldots, m_k) \cdot \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \vdots \\ \mathbf{V}_k \end{bmatrix}$$

$$(u_1, u_2, \ldots, u_n) = m_1 \cdot \mathbf{V}_1 + m_2 \cdot \mathbf{V}_2 + \ldots + m_2 \cdot \mathbf{V}_k$$

LINEAR BLOCK CODES – CONT'D

Example: Block code (6,3)

$$\mathbf{G} = \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Message vector	Codeword
000	000000
100	110100
010	011010
110	101110
001	101001
101	011101
011	110011
111	000111

SYSTEMATIC BLOCK CODES

Systematic block code (n,k)

- For a systematic code, the first (or last) k elements in the codeword are information bits.

$$\mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k]$$

$$\mathbf{I}_k = k \times k \text{ identity matrix}$$

$$\mathbf{P}_k = k \times (n-k) \text{ matrix}$$

$$\mathbf{U} = (u_1, u_2, \ldots, u_n) = (\underbrace{p_1, p_2, \ldots, p_{n-k}}_{\text{parity bits}}, \underbrace{m_1, m_2, \ldots, m_k}_{\text{message bits}})$$

LINEAR BLOCK CODES – CONT'D

For any linear code we can find an matrix $\mathbf{H}_{(n-k)\times n}$ which its rows are *orthogonal* to rows of $\mathbf{G}$

$$\boxed{\mathbf{G}\mathbf{H}^T = \mathbf{0}}$$

Why? H *checks the parity* of the received word (i.e. maps the N-bit word to a M-bit *syndrome*).

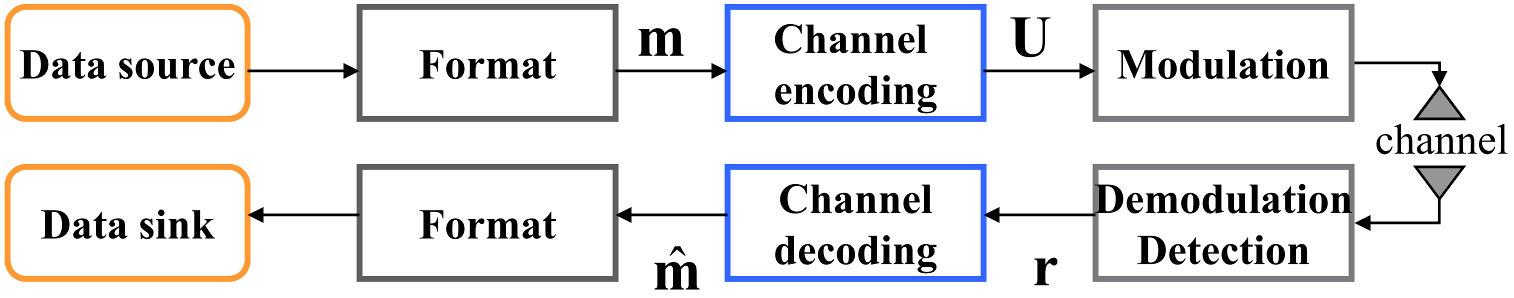
- Codewords (=**mG**) should have parity of 0 (i.e. null-space).

H is called the parity check matrix and its rows are linearly independent.

For systematic linear block codes:

$$\boxed{\mathbf{H} = [\mathbf{I}_{n-k} \vdots \mathbf{P}^T]}$$

LINEAR BLOCK CODES – CONT'D



$$\mathbf{r} = \mathbf{U} + \mathbf{e}$$

Syndrome testing:

- **S** is syndrome of **r**, corresponding to the error pattern **e**.

LINEAR BLOCK CODES – CONT'D

Error pattern	Syndrome
000000	000
000001	101
000010	011
000100	110
001000	001
010000	010
100000	100
010001	111

$\mathbf{U} = (101110)$ transmitted.

$\mathbf{r} = (001110)$ is received.

- The syndrome of $\mathbf{r}$ is computed:

$$\mathbf{S} = \mathbf{r}\mathbf{H}^T = (001110)\mathbf{H}^T = (100)$$

- Error pattern corresponding to this syndrome is

$$\hat{\mathbf{e}} = (100000)$$

- The corrected vector is estimated

$$\hat{\mathbf{U}} = \mathbf{r} + \hat{\mathbf{e}} = (001110) + (100000) = (101110)$$

There is a unique mapping from Syndrome $\leftrightarrow$ Error Pattern

STANDARD ARRAY: ERROR PATTERNS

Example: ***Standard array*** **for the (6,3) code**

codewords

000000	110100	011010	101110	101001	011101	110011	000111
000001	110101	011011	101111	101000	011100	110010	000110
000010	110110	011000	101100	101011	011111	110001	000101
000100	110000	011110	101010	101101	011010	110111	000110
001000	111100	⋮			⋮		⋮
010000	100100						
100000	010100				⋮		
010001	100101		⋯			⋯	010110

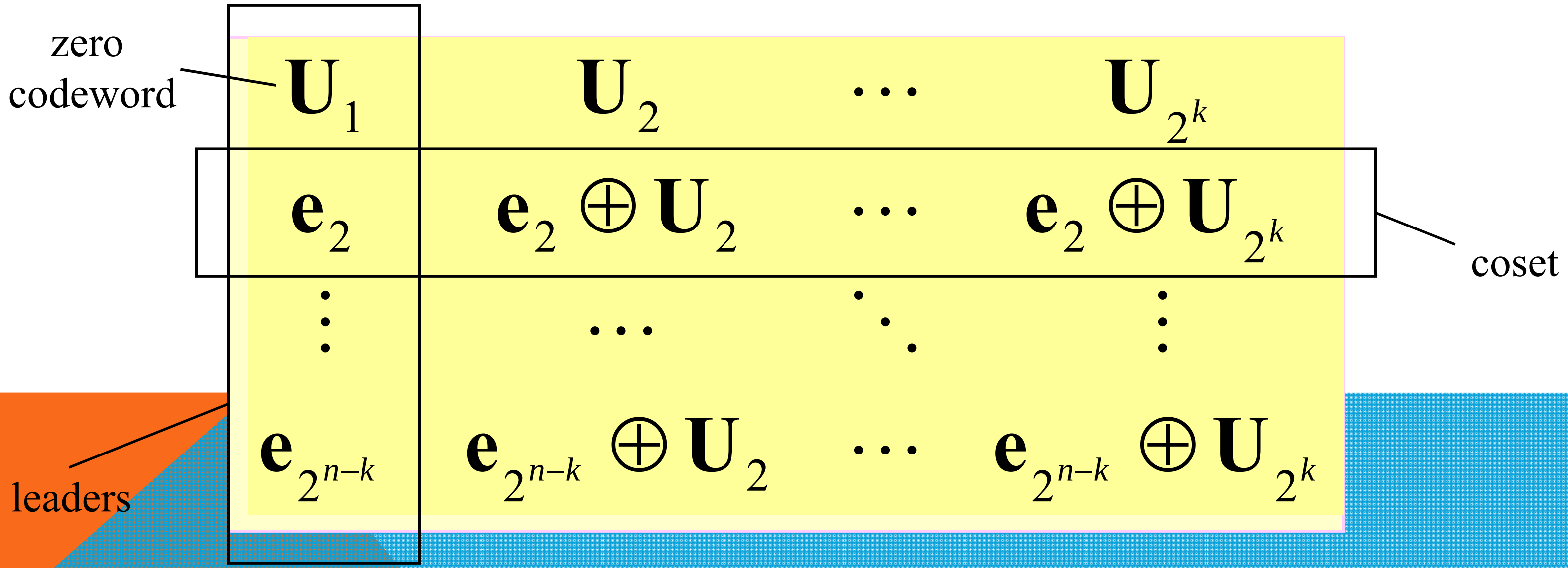
Coset:
Error pattern + codeword

Coset leaders
(error patterns)

LINEAR BLOCK CODES – CONT’D

Standard array

1. For row $i = 2, 3, \ldots, 2^{n-k}$, find a vector in V_n of minimum weight which is not already listed in the array.
2. Call this error pattern $\mathbf{e}_i$ and form the $i:\text{th}$ row as the corresponding coset



$$\begin{array}{cccc}
\mathbf{U}_1 & \mathbf{U}_2 & \cdots & \mathbf{U}_{2^k} \\
\mathbf{e}_2 & \mathbf{e}_2 \oplus \mathbf{U}_2 & \cdots & \mathbf{e}_2 \oplus \mathbf{U}_{2^k} \\
\vdots & \cdots & \ddots & \vdots \\
\mathbf{e}_{2^{n-k}} & \mathbf{e}_{2^{n-k}} \oplus \mathbf{U}_2 & \cdots & \mathbf{e}_{2^{n-k}} \oplus \mathbf{U}_{2^k}
\end{array}$$

LINEAR BLOCK CODES – CONT'D

Standard array and syndrome table decoding

1. Calculate syndrome $\mathbf{S} = \mathbf{r}\mathbf{H}^T$
2. Find the coset leader, $\hat{\mathbf{e}} = \mathbf{e}_i$, corresponding to $\mathbf{S}$.
3. Calculate $\hat{\mathbf{U}} = \mathbf{r} + \hat{\mathbf{e}}$ and corresponding $\mathbf{m}$.

- Note that $\hat{\mathbf{U}} = \mathbf{r} + \hat{\mathbf{e}} = (\mathbf{U} + \mathbf{e}) + \hat{\mathbf{e}} = \mathbf{U} + (\mathbf{e} + \hat{\mathbf{e}})$
  - If $\hat{\mathbf{e}} = \mathbf{e}$, error is corrected.
  - If $\hat{\mathbf{e}} \neq \mathbf{e}$, undetectable decoding error occurs.

HAMMING CODES

Hamming codes

- Hamming codes are a subclass of linear block codes and belong to the category of *perfect codes*.
- Hamming codes are expressed as a function of a ***single integer*** $m \geq 2$, i.e. *n* and *k* are derived from *m*:

Code length :	$n = 2^m - 1$
Number of information bits :	$k = 2^m - m - 1$
Number of parity bits :	$n\text{-}k = m$
Error correction capability :	$t = 1$

- The columns of the parity-check matrix, **H**, consist of all non-zero binary m-tuples.

HAMMING CODES

Example: Systematic Hamming code (7,4)

$$\mathbf{H} = \left[\begin{array}{ccc:cccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{array}\right] = [\mathbf{I}_{3\times 3} \; \vdots \; \mathbf{P}^T]$$

$$\mathbf{G} = \left[\begin{array}{ccc:cccc} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] = [\mathbf{P} \; \vdots \; \mathbf{I}_{4\times 4}]$$

CYCLIC BLOCK CODES

Cyclic **codes are a subclass of linear block codes.**

Encoding and syndrome calculation are easily performed using *feedback shift-registers*.

- Hence, relatively *long* block codes can be implemented with a reasonable complexity.

BCH and Reed-Solomon codes are cyclic codes.

CYCLIC BLOCK CODES

A linear (n,k) code is called a Cyclic code if all cyclic shifts of a codeword are also a codeword.

$$\boxed{\mathbf{U} = (u_0, u_1, u_2, \ldots, u_{n-1})}$$

"*i*" cyclic shifts of **U**

$$\boxed{\mathbf{U}^{(i)} = (u_{n-i}, u_{n-i+1}, \ldots, u_{n-1}, u_0, u_1, u_2, \ldots, u_{n-i-1})}$$

- Example:

$$\mathbf{U} = (1101)$$

$$\mathbf{U}^{(1)} = (1110) \quad \mathbf{U}^{(2)} = (0111) \quad \mathbf{U}^{(3)} = (1011) \quad \mathbf{U}^{(4)} = (1101) = \mathbf{U}$$

CYCLIC BLOCK CODES

Algebraic structure of Cyclic codes, implies expressing codewords in polynomial form

$$\boxed{\mathrm{U}(X) = u_0 + u_1 X + u_2 X^2 + \ldots + u_{n-1} X^{n-1} \qquad \text{degree}\,(n\text{-}1)}$$

- Relationship between a codeword and its cyclic shifts:

$$X\mathbf{U}(X) = u_0 X + u_1 X^2 + \ldots, u_{n-2} X^{n-1} + u_{n-1} X^n$$

$$= \underbrace{u_{n-1} + u_0 X + u_1 X^2 + \ldots + u_{n-2} X^{n-1}}_{\mathbf{U}^{(1)}(X)} + \underbrace{u_{n-1} X^n + u_{n-1}}_{u_{n-1}(X^n + 1)}$$

$$= \mathbf{U}^{(1)}(X) + u_{n-1}(X^n + 1)$$

- Hence:

$$\mathbf{U}^{(1)}(X) = X\mathbf{U}(X) \text{ modulo } (X^n + 1)$$

By extension

$$\boxed{\mathbf{U}^{(i)}(X) = X^i \mathbf{U}(X) \text{ modulo } (X^n + 1)}$$

CYCLIC BLOCK CODES

Basic properties of Cyclic codes:

- Let C be a binary (n,k) linear cyclic code
  1. Within the set of code polynomials in C, there is a unique monic polynomial with minimal degree is called the **generator polynomials.**
  2. Every code polynomial in C, can be expressed uniquely as
  3. The generator polynomial is a factor of

$\mathbf{g}(X)$

$r < n.$ $\mathbf{g}(X)$

$$\mathbf{g}(X) = g_0 + g_1 X + \ldots + g_r X^r$$

$\mathbf{U}(X)$

$\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}(X)$

$\mathbf{g}(X)$

$X^n + 1$

CYCLIC BLOCK CODES

4. The orthogonality of **G** and **H** in polynomial form is expressed as
   This means is also a factor of

$$\mathbf{g}(X)\mathbf{h}(X) = X^n + 1$$

5. The row of generator matrix is formed by the coefficients of the cyclic shift of the generator polynomial.

$$\mathbf{h}(X) \quad X^n + 1$$

$$i, i = 1, \ldots, k$$

$$"i-1"$$

$$\begin{bmatrix} g_0 & g_1 & \cdots & g_r & & & \mathbf{0} \\ & & & & & & \\ & & & g_0 & g_1 & \cdots & g_r & \\ \mathbf{0} & & & & g_0 & g_1 & \cdots & g_r \end{bmatrix}$$

$$\begin{bmatrix} \vdots \\ X^{k-1}\mathbf{g}(X) \end{bmatrix}$$

Toeplitz Matrix (like the circulant matrix): Efficient Linear Algebra Operations (multiplication, inverse, solution of Ax = b) etc possible

CYCLIC BLOCK CODES

Systematic encoding algorithm for an (n,k) Cyclic code:

1. Multiply the message polynomial by

$\mathbf{m}(X)$ X^{n-k}

2. Divide the result of Step 1 by the generator polynomial . Let be the **reminder**.

3. Add $\mathbf{g}(X)$ to $\mathbf{p}(X)$ to form the codeword

$\mathbf{p}(X)$ $X^{n-k}\mathbf{m}(X)$ $\mathbf{U}(X)$

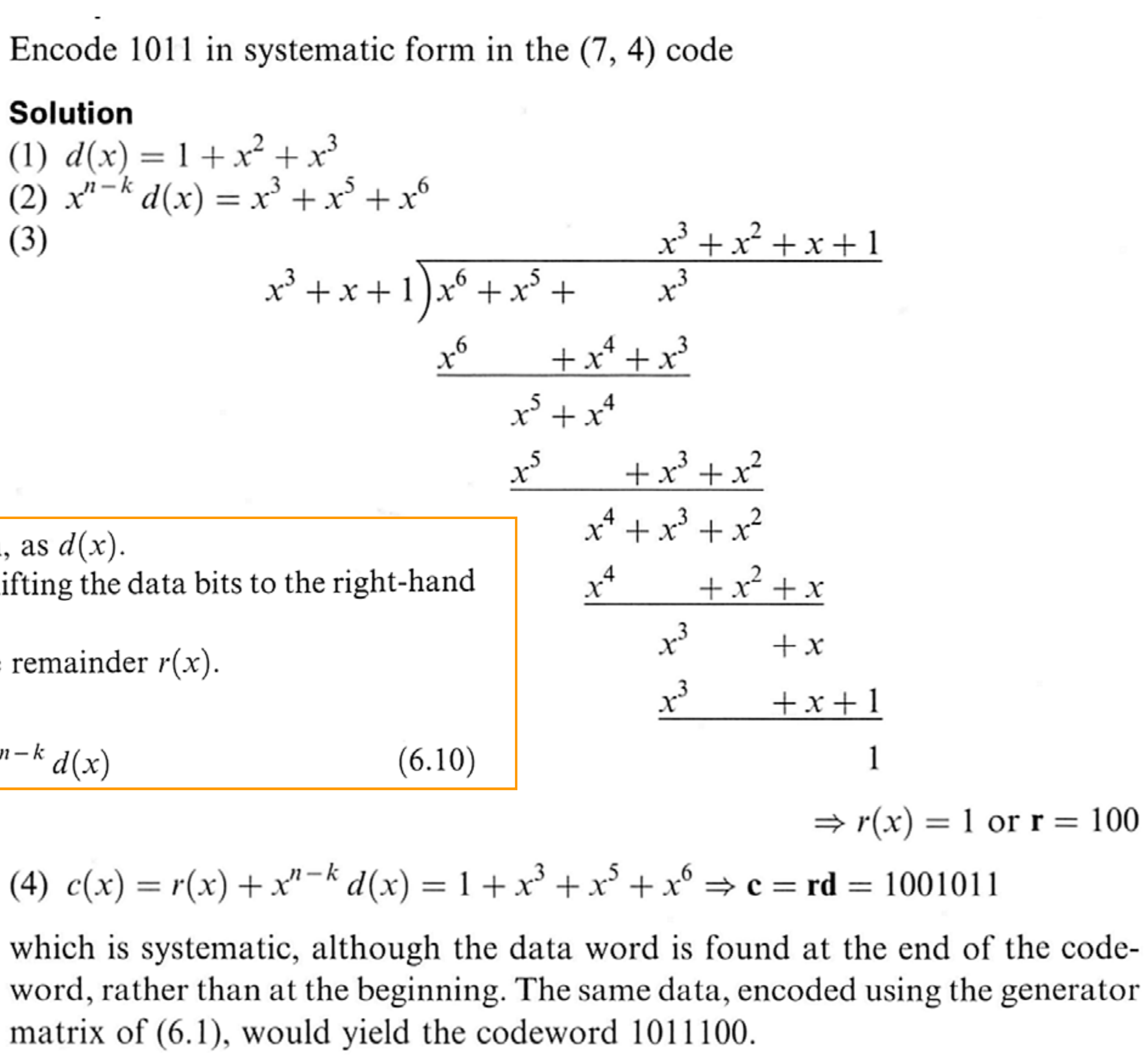
Remember CRC used to detect errors in packets?
"*Cyclic*" Redundancy Check: same idea!

CYCLIC BLOCK CODES

Example: For the systematic (7,4) Cyclic code with generator polynomial $\mathbf{g}(X) = 1 + X + X^3$

1. Find the codeword for the message $\mathbf{m} = (1011)$

EXAMPLE: ENCODING OF SYSTEMATIC CYCLIC CODES



Encode 1011 in systematic form in the (7, 4) code

Solution

(1) $d(x) = 1 + x^2 + x^3$

(2) $x^{n-k}\,d(x) = x^3 + x^5 + x^6$

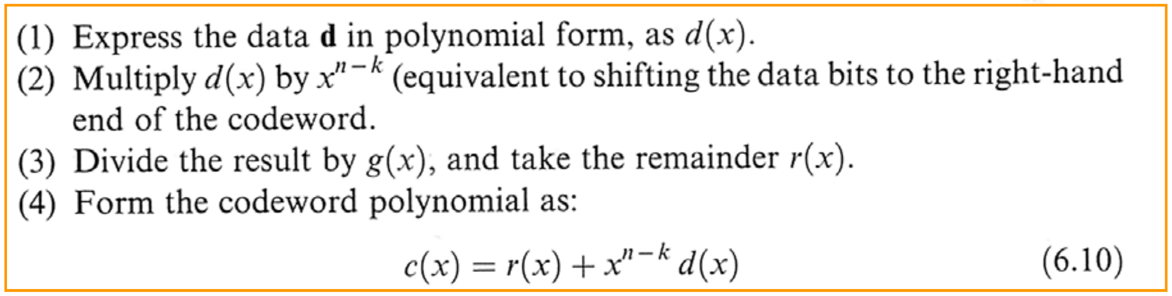
(3)

$$\begin{array}{r l}
 & \quad x^3 + x^2 + x + 1 \\
\cline{2-2}
x^3 + x + 1 & \!\!\big)\, x^6 + x^5 + x^3 \\
 & \underline{x^6 + x^4 + x^3} \\
 & \quad x^5 + x^4 \\
 & \quad \underline{x^5 + x^3 + x^2} \\
 & \qquad x^4 + x^3 + x^2 \\
 & \qquad \underline{x^4 + x^2 + x} \\
 & \qquad\quad x^3 + x \\
 & \qquad\quad \underline{x^3 + x + 1} \\
 & \qquad\qquad\qquad 1
\end{array}$$

$\Rightarrow r(x) = 1$ or $\mathbf{r} = 100$

(4) $c(x) = r(x) + x^{n-k}\,d(x) = 1 + x^3 + x^5 + x^6 \Rightarrow \mathbf{c} = \mathbf{rd} = 1001011$

which is systematic, although the data word is found at the end of the codeword, rather than at the beginning. The same data, encoded using the generator matrix of (6.1), would yield the codeword 1011100.



(1) Express the data **d** in polynomial form, as $d(x)$.
(2) Multiply $d(x)$ by x^{n-k} (equivalent to shifting the data bits to the right-hand end of the codeword.
(3) Divide the result by $g(x)$, and take the remainder $r(x)$.
(4) Form the codeword polynomial as:

$$c(x) = r(x) + x^{n-k}\,d(x) \tag{6.10}$$

Decoding cyclic codes

e	s
1000000	110
0100000	011
0010000	111
0001000	101
0000100	100
0000010	010
0000001	001

When the received word $\boldsymbol{r}$ is **1101101**,

$$r(x) = x^6 + x^5 + x^3 + x^2 + 1$$

We now compute $s(x) = \mathrm{mod}\left[r(x)/g(x)\right]$

$$\begin{array}{r|l}
 & x^3 \\
\cline{2-2}
x^3 + x^2 + 1 & x^6 + x^5 + x^3 + x^2 + 1 \\
 & x^6 + x^5 + x^3 \\
\cline{2-2}
 & \qquad\qquad\qquad x^2 + 1
\end{array}$$

$g(x)$

Hence, $\boldsymbol{s}$ = **101**. From Table 16.6, this gives $\boldsymbol{e}$ = **0001000**, and

$$\boldsymbol{c} = \boldsymbol{r} \oplus \boldsymbol{e} = \mathbf{1101101} \oplus \mathbf{0001000} = \mathbf{1100101}$$

Hence, from Table 16.5 we have

$$\boldsymbol{d} = \mathbf{1100}$$

Table 16.5

d	c
1111	1111111
1110	1110010
1101	1101000
1100	1100101
1011	1011100
1010	1010001
1001	1001011
1000	1000110
0111	0111001

In a similar way, we determine for $\boldsymbol{r}$ = **0101000**, $\boldsymbol{s}$ = **110** and $\boldsymbol{e}$ = **1000000**; hence $\boldsymbol{c} = \boldsymbol{r} \oplus \boldsymbol{e}$ = **1101000**, and $\boldsymbol{d}$ = **1101**. For $\boldsymbol{r}$ = **0001100**, $\boldsymbol{s}$ = **001** and $\boldsymbol{e}$ = **0000001**; hence $\boldsymbol{c} = \boldsymbol{r} \oplus \boldsymbol{e}$ = **0001101**, and $\boldsymbol{d}$ = **0001**.

CYCLIC BLOCK CODES

2. Find the generator and parity check matrices, **G** and **H**, respectively.

CYCLIC BLOCK CODES

Syndrome decoding for Cyclic codes:

- Received codeword in polynomial form is given by

Received codeword

$$\mathbf{r}(X) = \mathbf{U}(X) + \mathbf{e}(X)$$

Error pattern

- The syndrome is the reminder obtained by dividing the received polynomial by the generator polynomial.

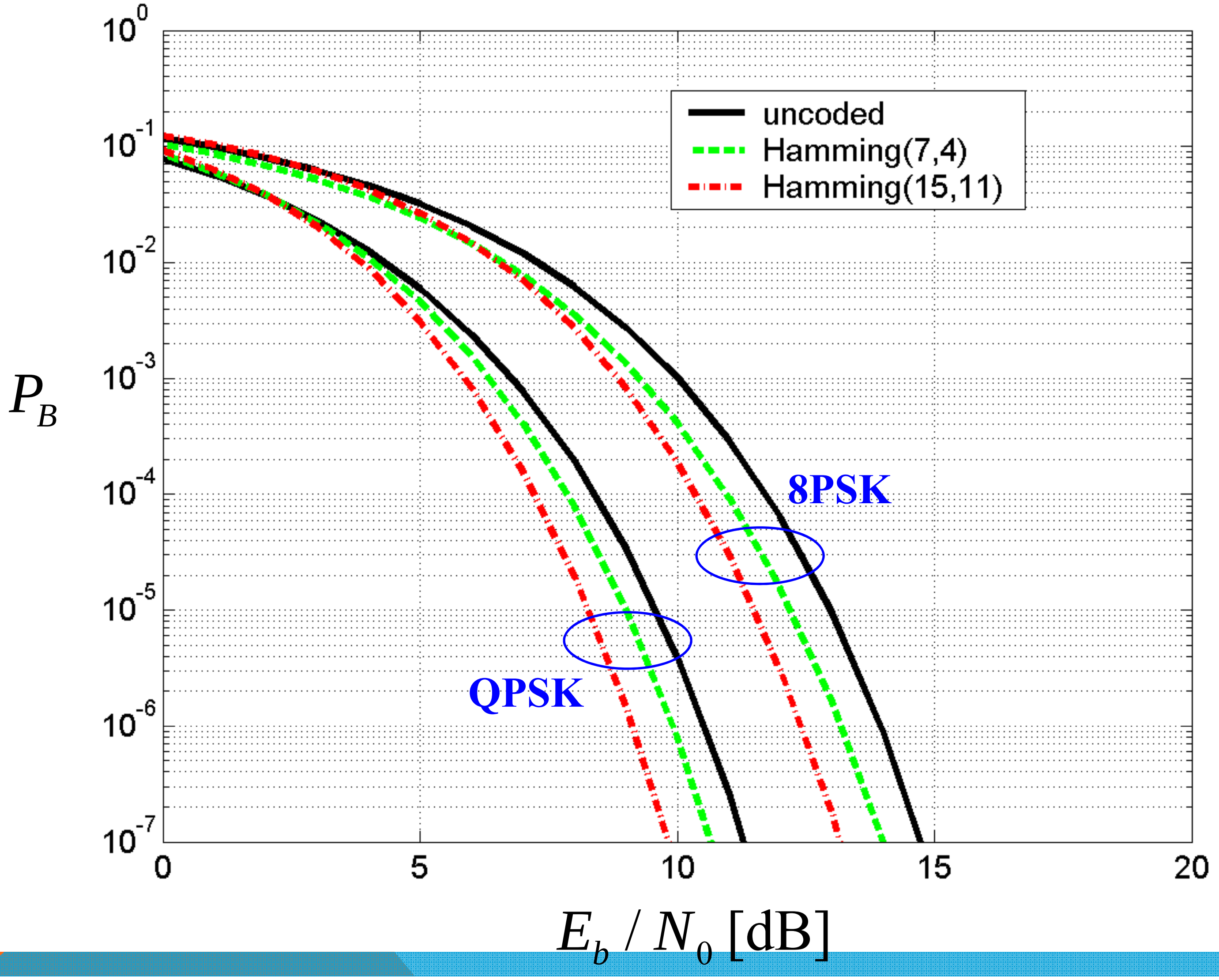
$$\mathbf{r}(X) = \mathbf{q}(X)\mathbf{g}(X) + \boxed{\mathbf{S}(X)}$$

Syndrome

- With syndrome and Standard array, error is estimated.

In Cyclic codes, the size of standard array is considerably reduced.

EXAMPLE OF THE BLOCK CODES



WELL-KNOWN CYCLIC CODES

(n,1) **Repetition codes.** High coding gain, but low rate

(n,k) **Hamming codes**. Minimum distance always 3. Thus can detect 2 errors and correct one error. $n=2^m-1$, $k = n - m$, $m \geq 3$

Maximum-length codes. For every integer $k \geq 3$ there exists a maximum length code (*n,k*) with $n = 2^k - 1$, $d_{min} = 2^{k-1}$. Hamming codes are dual of maximal codes.

BCH-codes. For every integer $m \geq 3$ there exist a code with $n = 2^m-1$, $k \geq n - mt$ and $d_{\min} \geq 2t+1$ where *t* is the error correction capability

(*n,k*) **Reed-Solomon (RS) codes**. Works with *k* symbols that consists of *m* bits that are encoded to yield code words of *n* symbols. For these codes $n = 2^m - 1$, number of check symbols $n - k = 2t$ and $d_{\min} = 2t+1$

BCH and RS are popular due to large d_{min}, large number of codes, and easy generation

REED-SOLOMON CODES (RS)

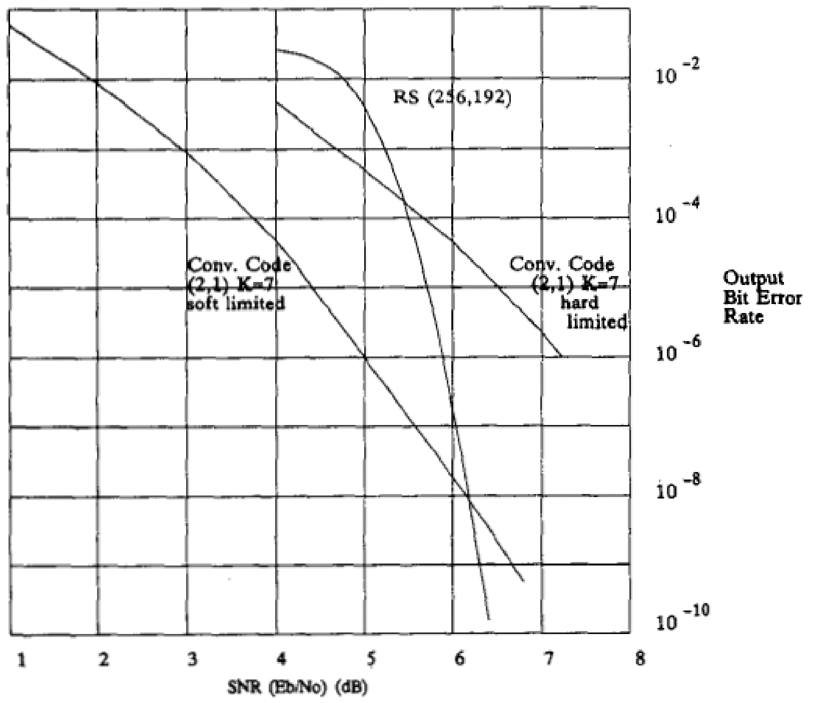


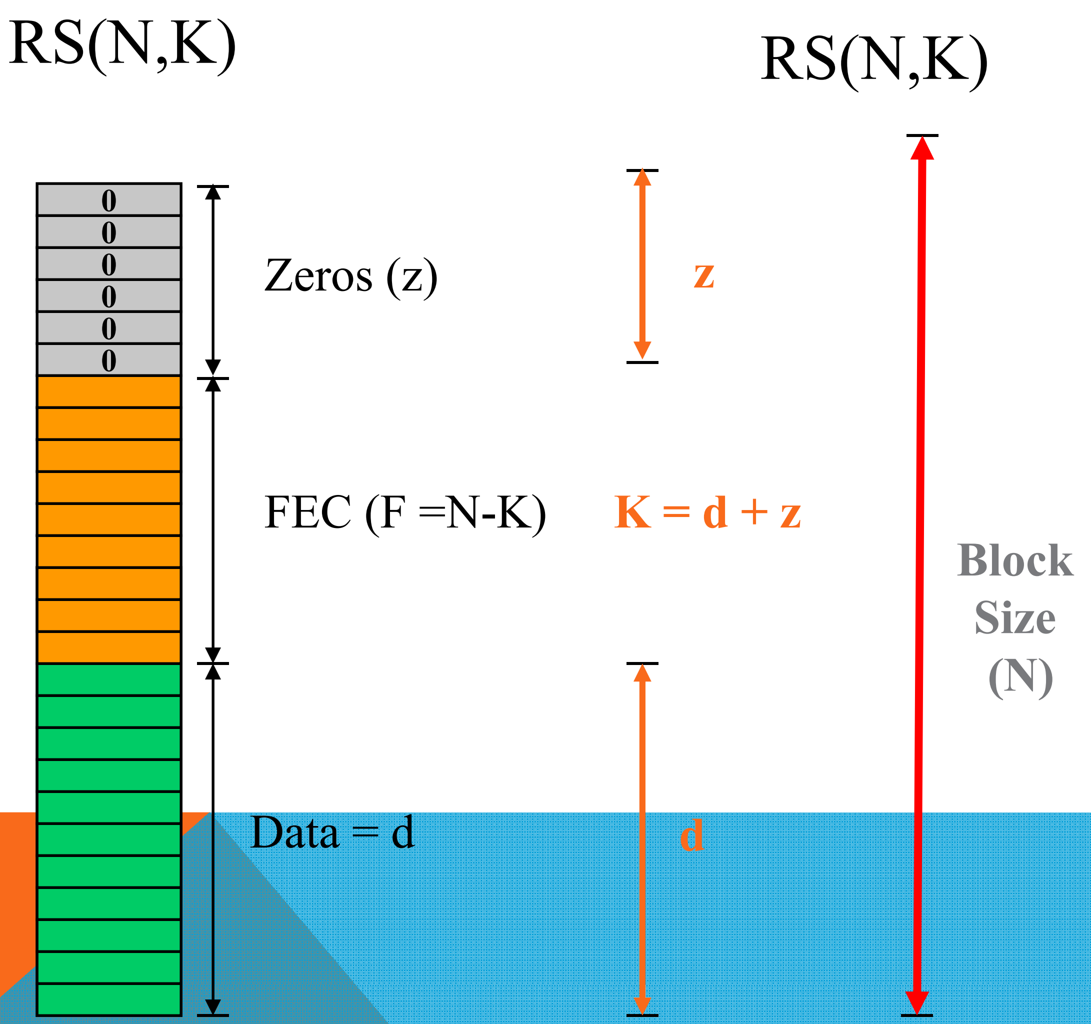
Fig. 3. RS Code versus Convolutional Code.

Group bits into L-bit symbols. Like BCH codes with symbols rather than single bits.

Can tolerate burst error better (fewer symbols in error for a given bit-level burst event).

Shortened RS-codes used in CD-ROMs, DVDs etc

SHORTENED REED SOLOMON CODES



RS-CODE PERFORMANCE

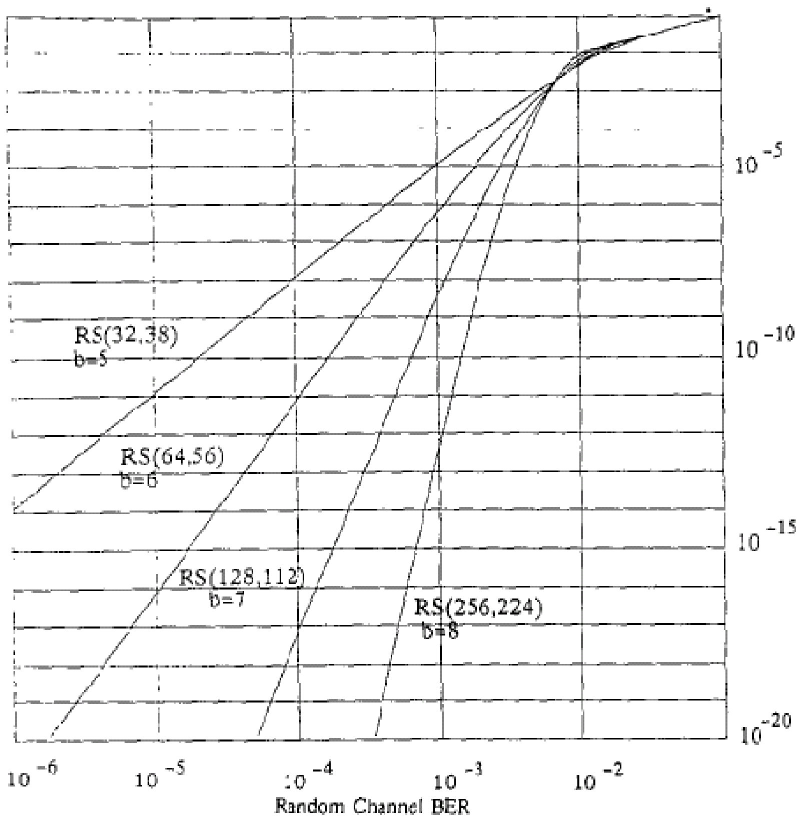


Fig. 5. RS Codes of 7/8-th Rate

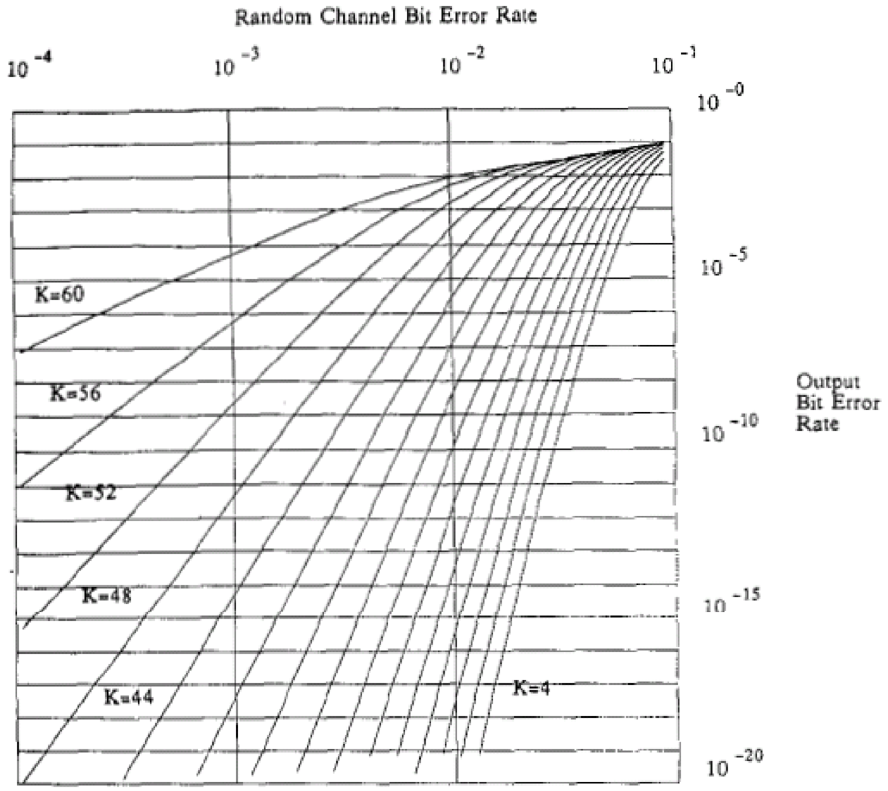
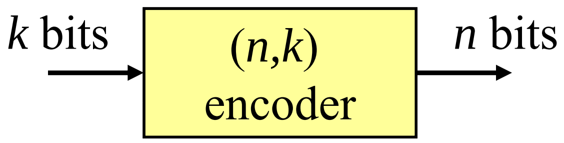


Fig. 2. RS(64,k) Random Digital Error Performance.

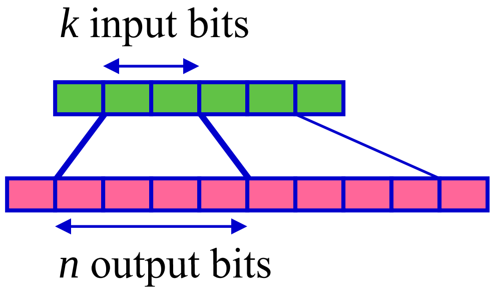
- **Longer blocks, better performance**
- **Encoding/decoding complexity lower for higher code rates (i.e. > ½): O{K(N-K) $\log_2 N$}.**

CONVOLUTIONAL CODES

BLOCK VS CONVOLUTIONAL CODING



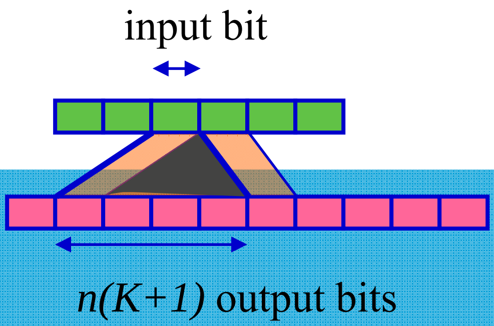
(n,k) block codes: Encoder output of
***n* bits depends only on the *k* input bits**



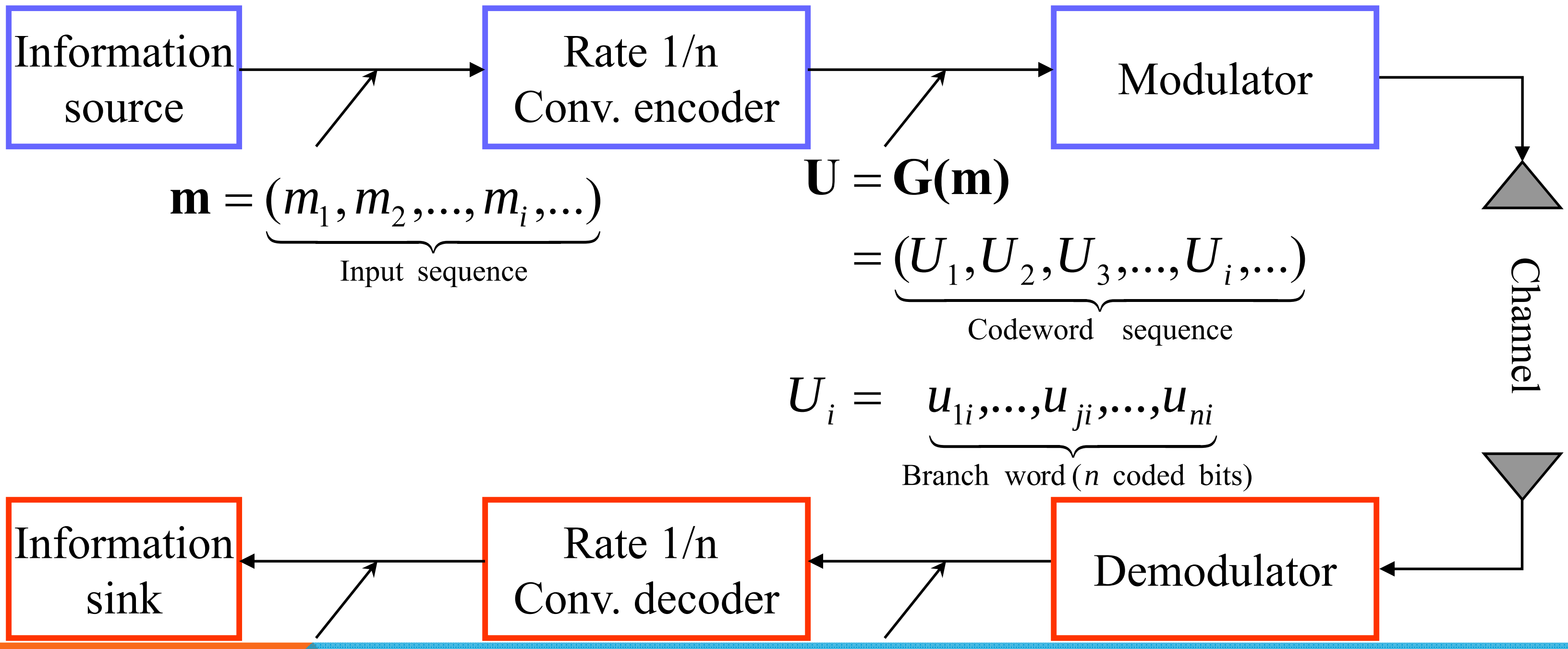
(n,k,K) convolutional codes:

each source bit influences *n(K+1)* encoder output bits

n(K+1) is the constraint length



BLOCK DIAGRAM: CONVOLUTIONAL CODING



$$\mathbf{m} = \underbrace{(m_1, m_2, \ldots, m_i, \ldots)}_{\text{Input sequence}}$$

$$\mathbf{U} = \mathbf{G(m)} = \underbrace{(U_1, U_2, U_3, \ldots, U_i, \ldots)}_{\text{Codeword sequence}}$$

$$U_i = \underbrace{u_{1i}, \ldots, u_{ji}, \ldots, u_{ni}}_{\text{Branch word } (n \text{ coded bits})}$$

$$\hat{\mathbf{m}} = (\hat{m}_1, \hat{m}_2, \ldots, \hat{m}_i, \ldots)$$

$$\mathbf{Z} = \underbrace{(Z_1, Z_2, Z_3, \ldots, Z_i, \ldots)}_{\text{received sequence}}$$

$$\underbrace{Z_i}_{\substack{\text{Demodulator outputs} \\ \text{for Branch word } i}} = \underbrace{z_{1i}, \ldots, z_{ji}, \ldots, z_{ni}}_{n \text{ outputs per Branch word}}$$

CONVOLUTIONAL CODES-CONT'D

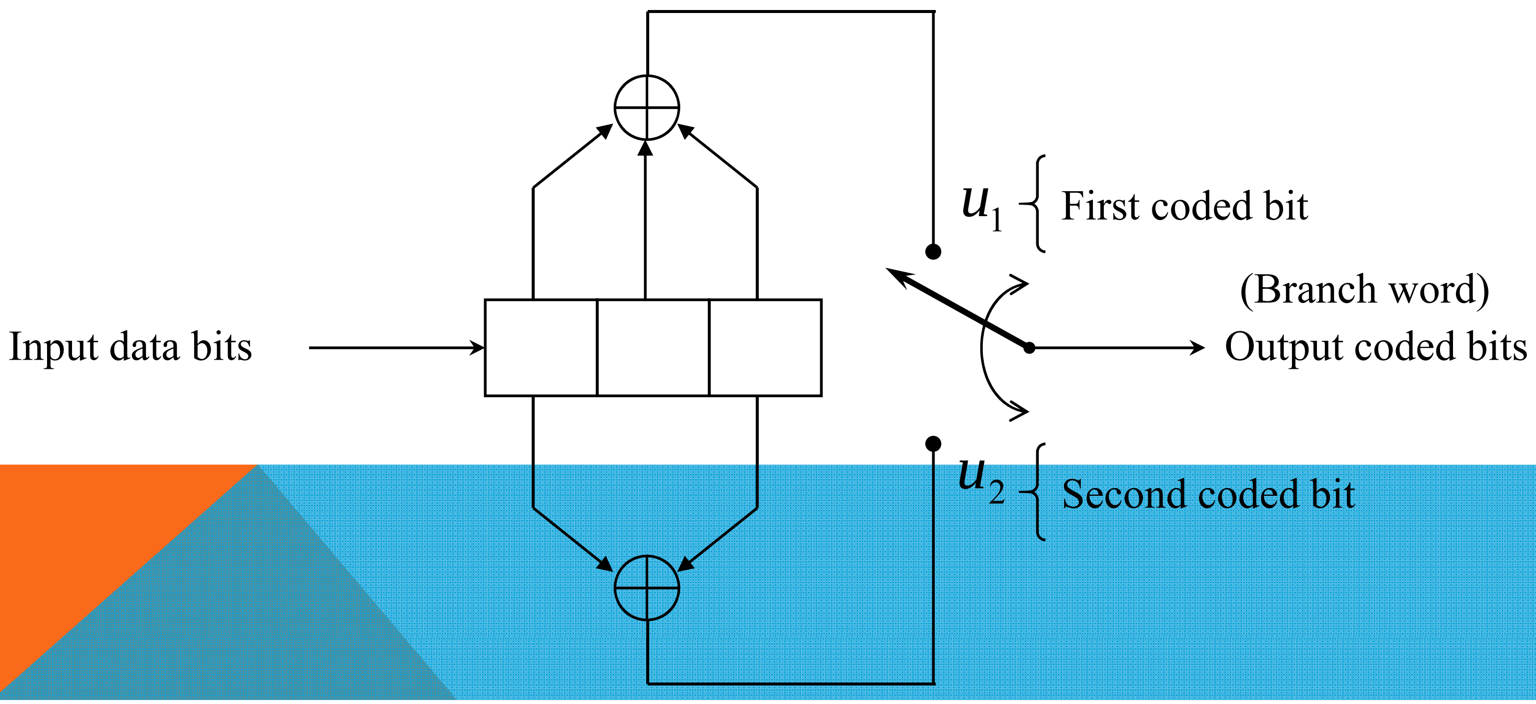
A Convolutional code is specified by three parameters (n,k,K) or $(k/n,K)$ where

- $R_c = k/n$ is the **coding rate**, determining the number of data bits per coded bit.
  - In practice, usually *k=1* is chosen and we assume that from now on.
- *K* is the **constraint length** of the encoder a where the encoder has *K-1* memory elements.

A RATE ½ CONVOLUTIONAL ENCODER

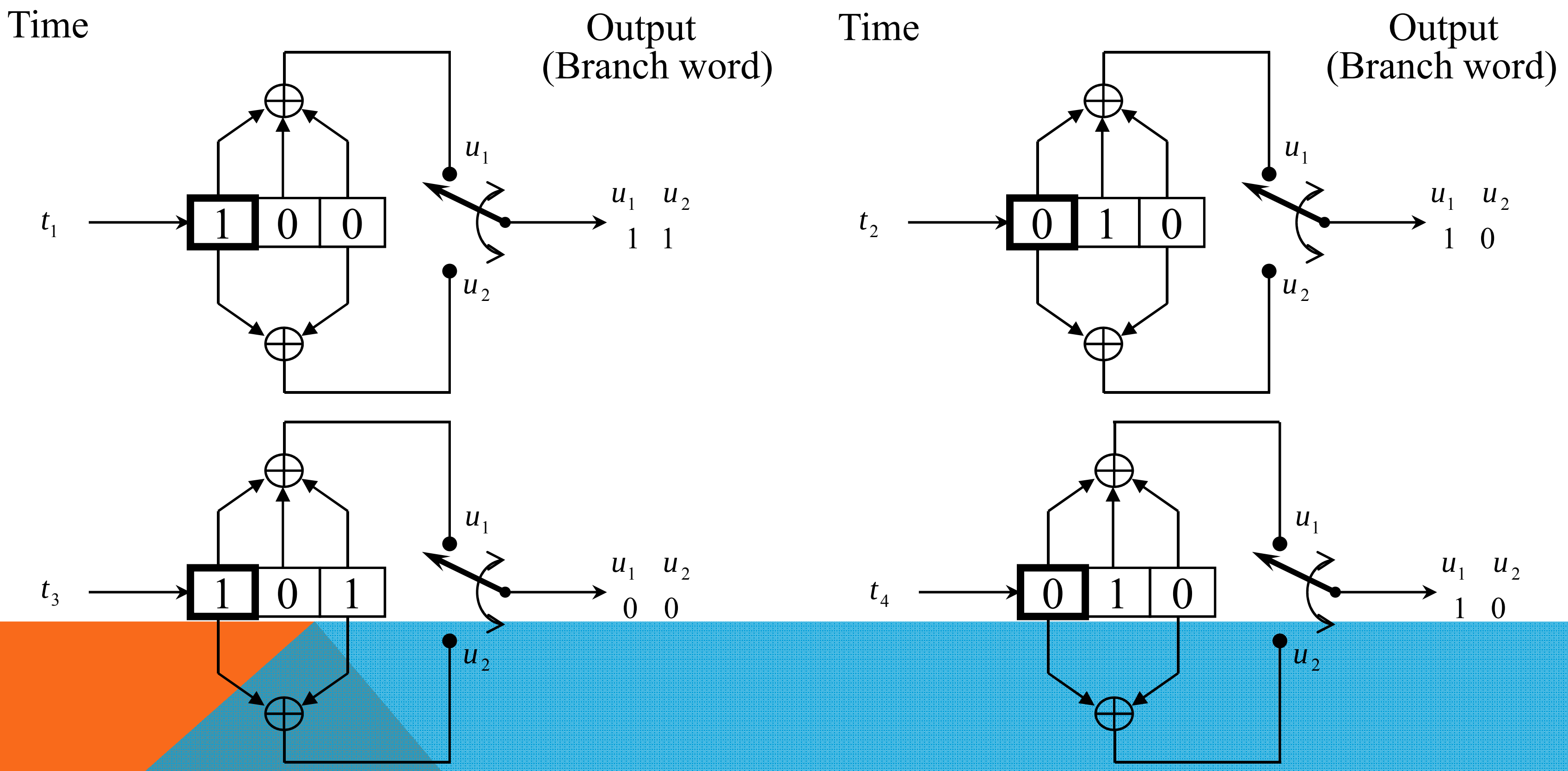
Convolutional encoder (rate ½, K=3)

- 3 bit shift-register where the first one takes the incoming data bit and the rest form the memory of the encoder.

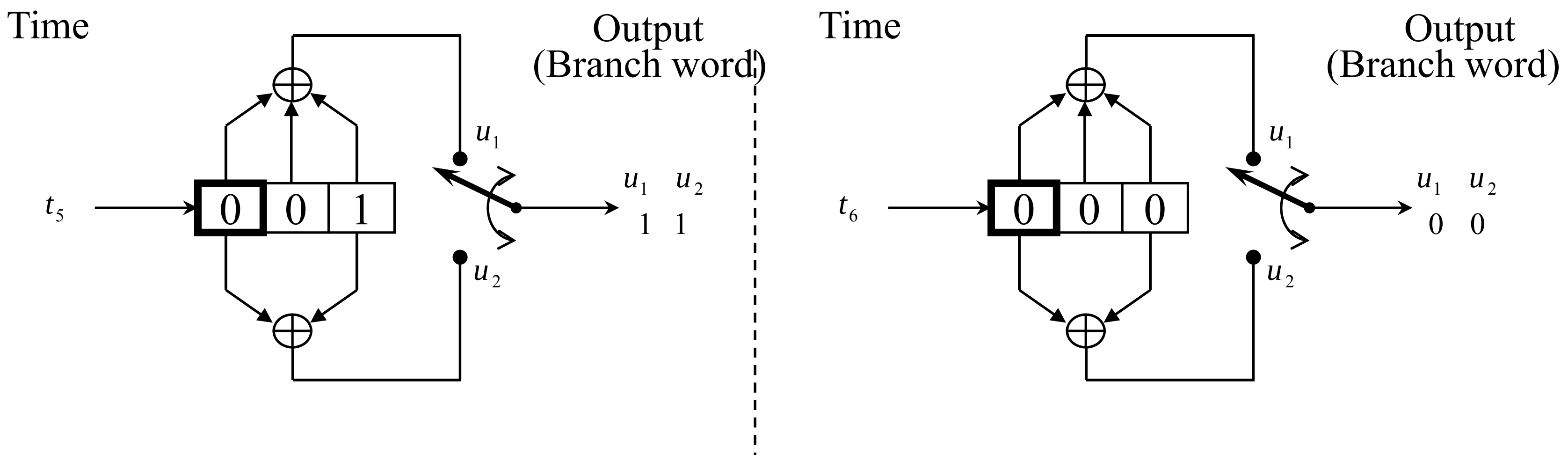


A RATE ½ CONVOLUTIONAL ENCODER

Message sequence: $\mathbf{m} = (101)$



A RATE ½ CONVOLUTIONAL ENCODER (CONTD)



$\mathbf{m} = (101)$ $\longrightarrow$ $\boxed{\text{Encoder}}$ $\longrightarrow$ $\mathbf{U} = (11\quad 10\quad 00\quad 10\quad 11)$

n = 2, k = 1, K = 3,

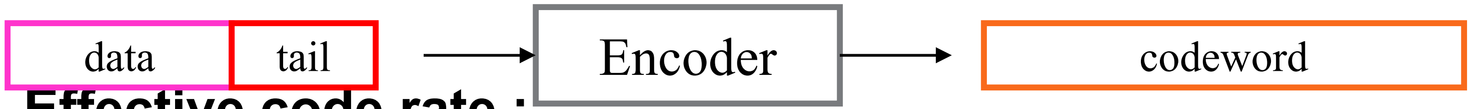
L = 3 input bits -> 10 output bits

EFFECTIVE CODE RATE

Initialize the memory before encoding the first bit (all-zero)

Clear out the memory after encoding the last bit (all-zero)

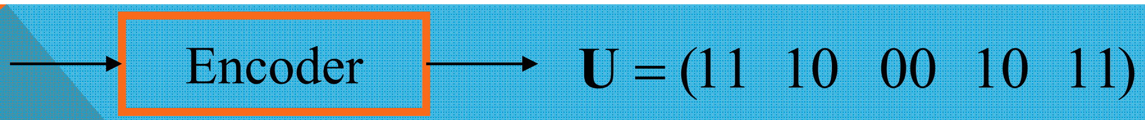
- Hence, a tail of zero-bits is appended to data bits.



Effective code rate :

- L is the number of data bits and *k=1* is assumed:

$$R_{eff} = \frac{L}{n(L+K-1)} < R_c$$



<u>Example</u>: n = 2, k = 1, K = 3, L = <u>*3 input bits.*</u>

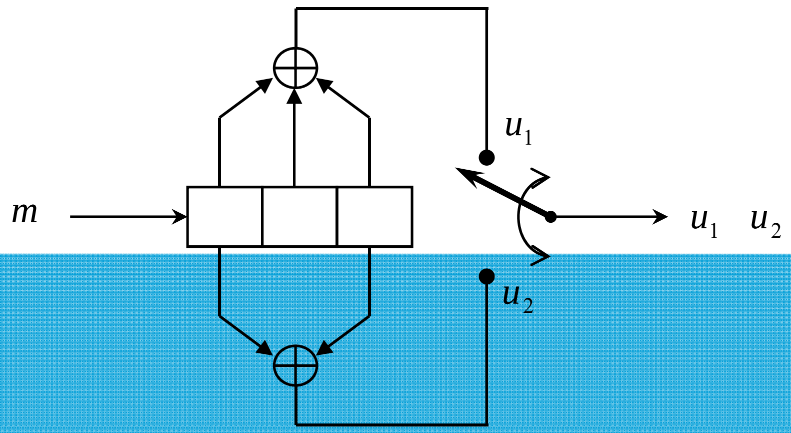
Output = n(L + K -1) = 2*(3 + 3 – 1) = *10 output bits*

ENCODER REPRESENTATION

Vector representation:

- We define n binary vector with *K* elements (one vector for each modulo-2 adder).
- The i:th element in each vector, is "1" if the i:th stage in the shift register is connected to the corresponding modulo-2 adder, and "0" otherwise.
  - Example:

$$\mathbf{g}_1 = (111)$$
$$\mathbf{g}_2 = (101)$$



ENCODER REPRESENTATION: IMPULSE RESPONSE

<u>Impulse response</u> representaiton:

- The response of encoder to a single "one" bit that goes through it.
  - Example:

Register contents	Branch word u_1	Branch word u_2
100	1	1
010	1	0
001	1	1

$$\text{Input sequence}: \quad 1 \quad 0 \quad 0$$

$$\text{Output sequence}: \; 11 \quad 10 \quad 11$$

Input **m**	Output				
1	11	10	11		
0		00	00	00	
1			11	10	11
Modulo-2 sum:	11	10	00	10	11

ENCODER REPRESENTATION: POLYNOMIAL

Polynomial representation:

- We define n generator polynomials, one for each modulo-2 adder. Each polynomial is of degree *K-1* or less and describes the connection of the shift registers to the corresponding modulo-2 adder.
  - Example:

$$\mathbf{g}_1(X) = g_0^{(1)} + g_1^{(1)}.X + g_2^{(1)}.X^2 = 1 + X + X^2$$

$$\mathbf{g}_2(X) = g_0^{(2)} + g_1^{(2)}.X + g_2^{(2)}.X^2 = 1 + X^2$$

The output sequence is found as follows:

$$\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}_1(X) \text{ interlaced with } \mathbf{m}(X)\mathbf{g}_2(X)$$

ENCODER REPRESENTATION –CONT'D

In more details:

$$\mathbf{m}(X)\mathbf{g}_1(X) = (1+X^2)(1+X+X^2) = 1+X+X^3+X^4$$

$$\mathbf{m}(X)\mathbf{g}_2(X) = (1+X^2)(1+X^2) = 1+X^4$$

$$\mathbf{m}(X)\mathbf{g}_1(X) = 1+X+0.X^2+X^3+X^4$$

$$\mathbf{m}(X)\mathbf{g}_2(X) = 1+0.X+0.X^2+0.X^3+X^4$$

$$\mathbf{U}(X) = (1,1)+(1,0)X+(0,0)X^2+(1,0)X^3+(1,1)X^4$$

$$\mathbf{U} = 11 \quad 10 \quad 00 \quad 10 \quad 11$$

STATE DIAGRAM

A finite-state machine only encounters a finite number of states.

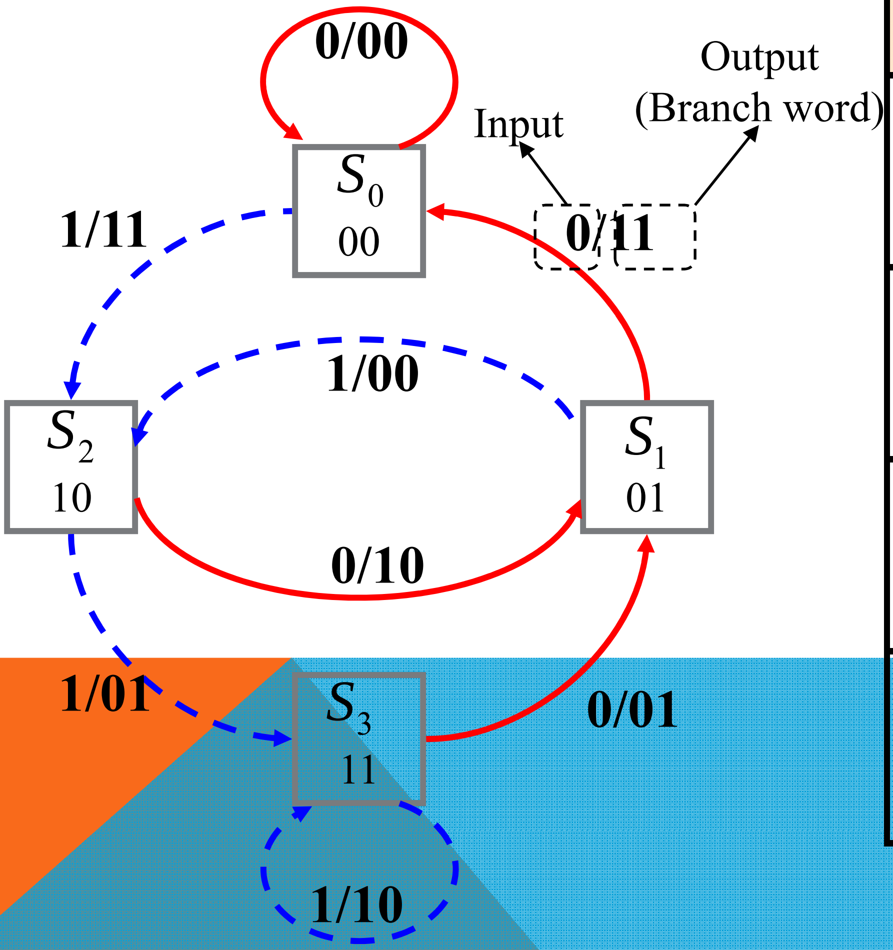
State of a machine: the smallest amount of information that, together with a current input to the machine, can predict the output of the machine.

In a convolutional encoder, the state is represented by the content of the memory.

Hence, there are states. (grows exponentially w/ constraint length)

$$2^{K-1}$$

STATE DIAGRAM – CONT’D

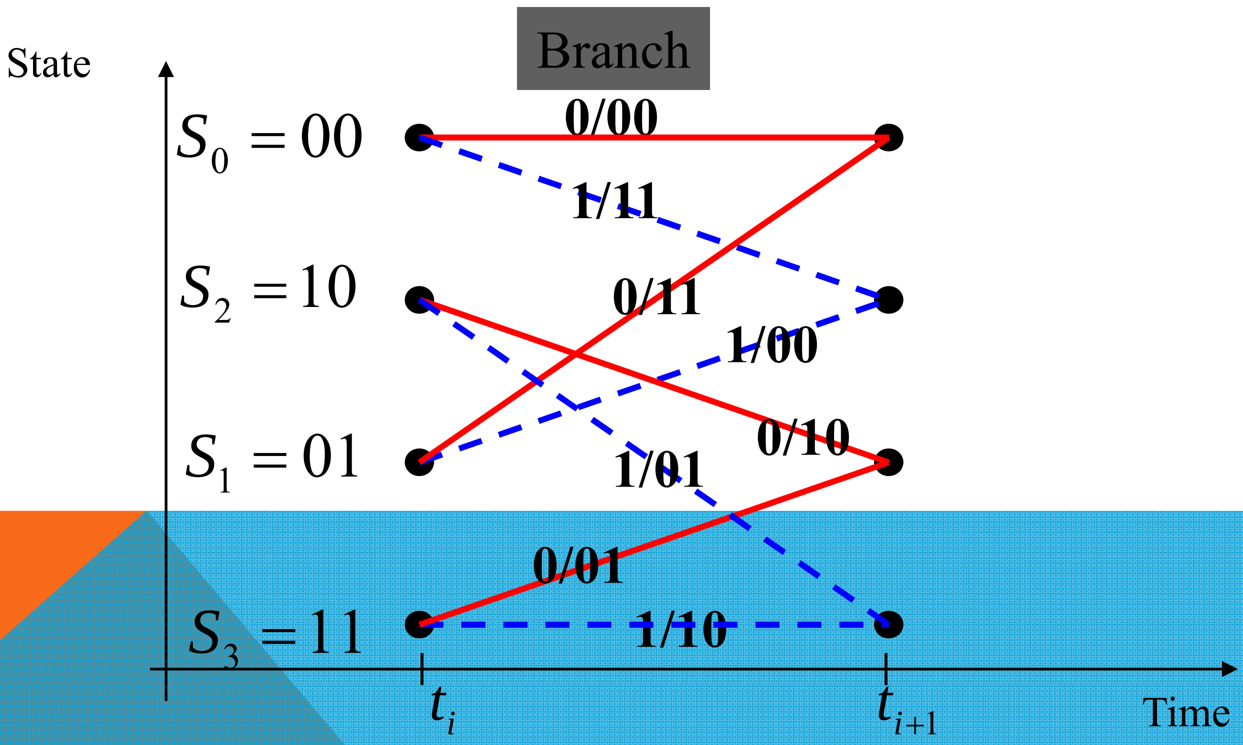


Current state	input	Next state	output
S_0 00	0	S_0	00
	1	S_2	11
S_1 01	0	S_0	11
	1	S_2	00
S_2 10	0	S_1	10
	1	S_3	01
S_3 11	0	S_1	01
	1	S_3	10

TRELLIS – CONT’D

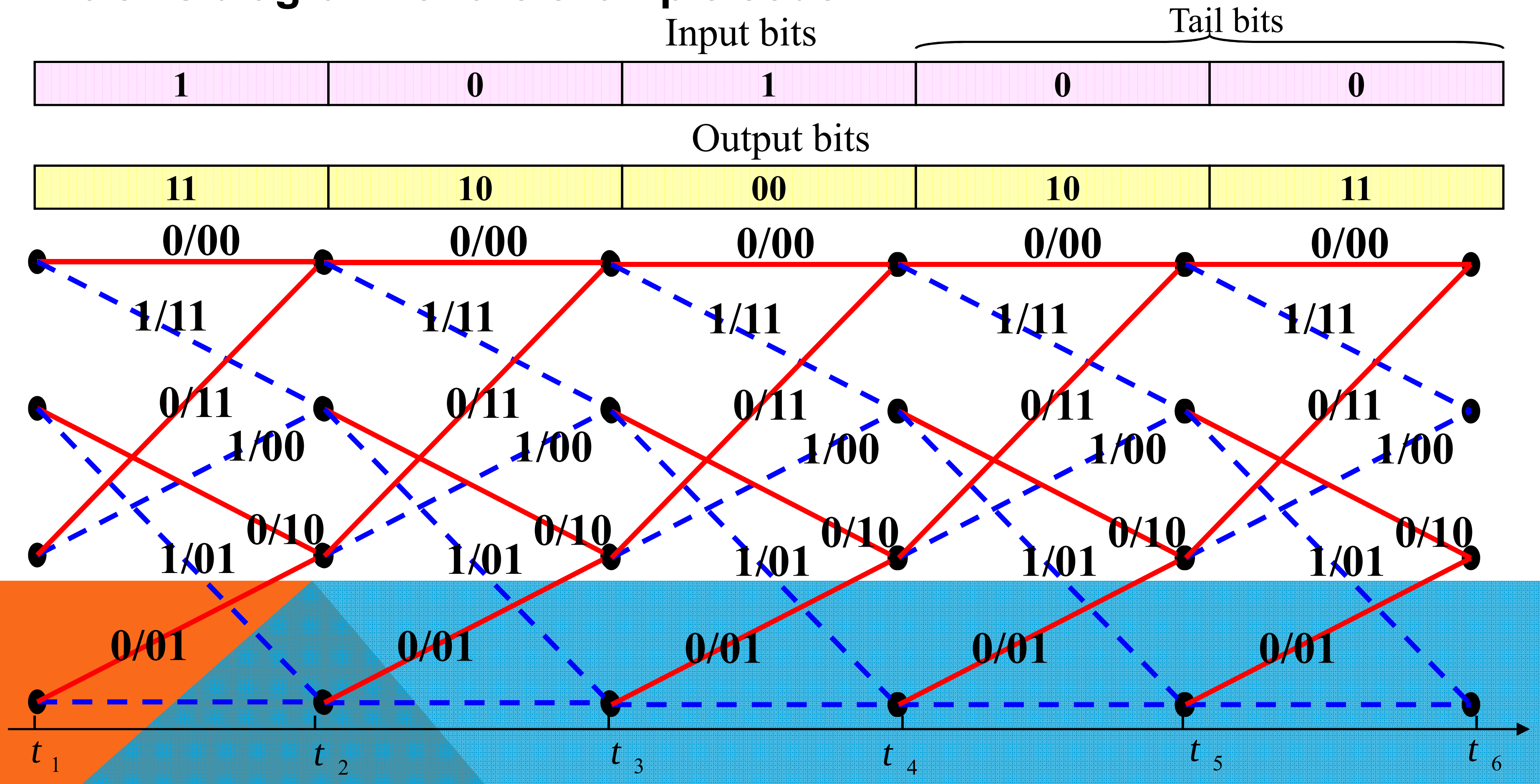
Trellis diagram is an extension of the state diagram that shows the passage of time.

- Example of a section of trellis for the rate ½ code

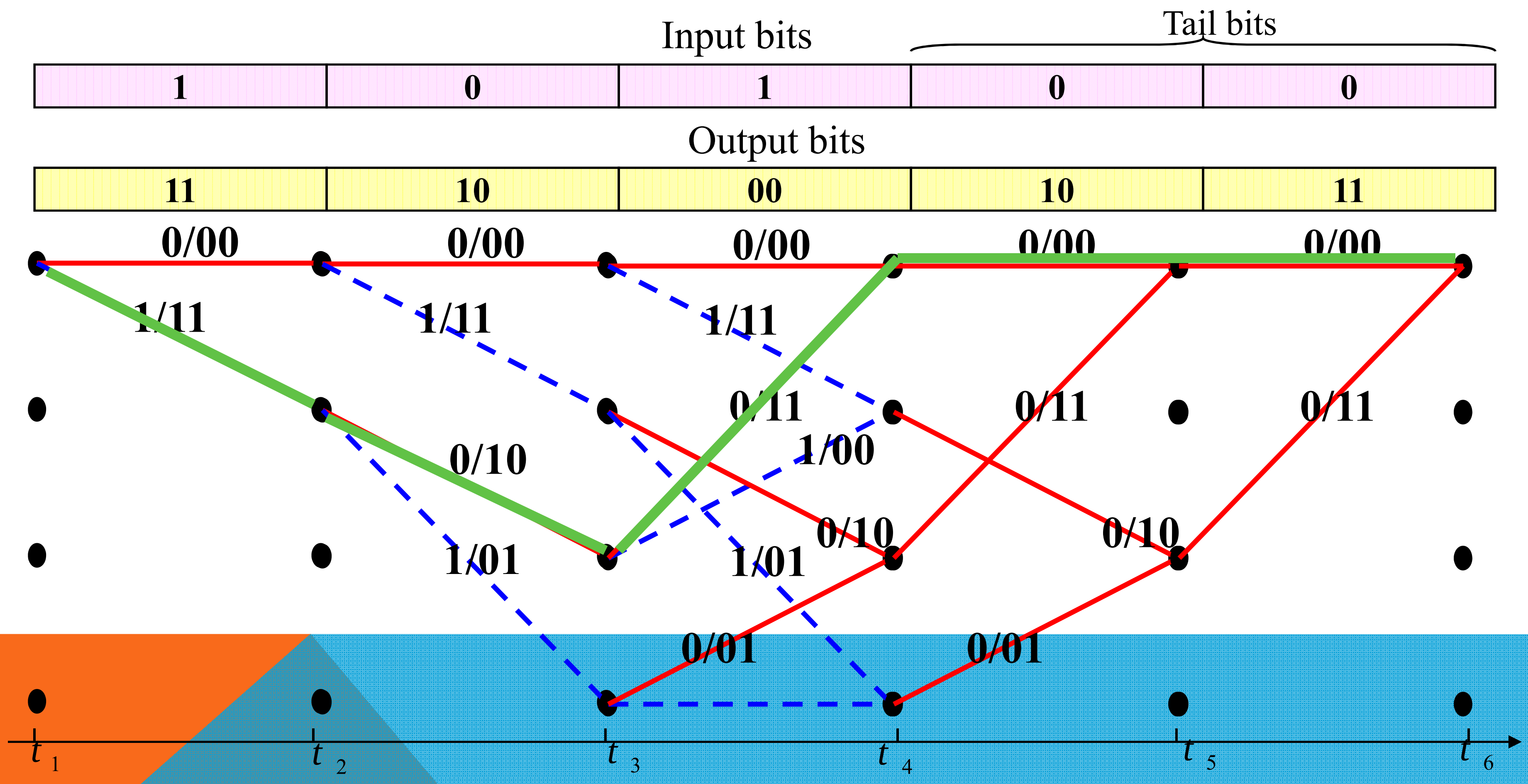


TRELLIS –CONT'D

A trellis diagram for the example code



TRELLIS – CONT'D



Path through the trellis

OPTIMUM DECODING

If the input sequence messages are equally likely, the optimum decoder which minimizes the probability of error is the *Maximum likelihood* decoder.

ML decoder, selects a codeword among all the possible codewords which maximizes the likelihood function $p(\mathbf{Z}|\mathbf{U}^{(m')})$ where $\mathbf{Z}$ is the *<u>received</u>* <u>sequence</u> and $\mathbf{U}^{(m')}$ is one of the possible codewords:

➢ML decoding rule:

$$\text{Choose } \mathbf{U}^{(m')} \text{ if } p(\mathbf{Z}|\mathbf{U}^{(m')}) = \max_{\text{over all } \mathbf{U}^{(m)}} p(\mathbf{Z}|\mathbf{U}^{(m)})$$

2^L codewords to search!!!

ML DECODING FOR MEMORY-LESS CHANNELS

Due to the independent channel statistics for memoryless channels, the likelihood function becomes

$$p(\mathbf{Z}|\mathbf{U}^{(m)}) = p_{z_1,z_2,\ldots,z_i,\ldots}(Z_1,Z_2,\ldots,Z_i,\ldots|U^{(m)}) = \prod_{i=1}^{\infty} p(Z_i|U_i^{(m)}) = \prod_{i=1}^{\infty}\prod_{j=1}^{n} p(z_{ji}|u_{ji}^{(m)})$$

and equivalently, the log-likelihood function becomes

$$\gamma_{\mathbf{U}}(m) = \underbrace{\log p(\mathbf{Z}|\mathbf{U}^{(m)})}_{\text{Path metric}} = \sum_{i=1}^{\infty} \underbrace{\log p(Z_i|U_i^{(m)})}_{\text{Branch metric}} = \sum_{i=1}^{\infty}\sum_{j=1}^{n} \underbrace{\log p(z_{ji}|u_{ji}^{(m)})}_{\text{Bit metric}}$$

The path metric up to time index $"i"$, is called the partial path metric.

➢ML decoding rule:

Choose the path with maximum metric among all the paths in the trellis.

This path is the “closest” path to the transmitted sequence.

AWGN CHANNELS

For BPSK modulation the transmitted sequence corresponding to the codeword $\mathbf{U}^{(m)}$ **is denoted by** $\mathbf{S}^{(m)} = (S_1^{(m)}, S_2^{(m)}, \dots, S_i^{(m)}, \dots)$ **where** $S_i^{(m)} = (s_{1i}^{(m)}, \dots, s_{ji}^{(m)}, \dots, s_{ni}^{(m)})$ **and** $s_{ij} = \pm\sqrt{E_c}$ **.**

The log-likelihood function becomes

$$\gamma_{\mathbf{U}}(m) = \sum_{i=1}^{\infty} \sum_{j=1}^{n} z_{ji} s_{ji}^{(m)} = < \mathbf{Z}, \mathbf{S}^{(m)} >$$

Inner product or correlation between **Z** and **S**

- Maximizing the correlation is equivalent to minimizing the Euclidean distance.

➢ML decoding rule:

Choose the path which with minimum Euclidean distance to the received sequence.

THE VITERBI ALGORITHM

The Viterbi algorithm performs Maximum likelihood decoding.

It find a path through trellis with the largest metric (maximum correlation or minimum distance).

- It processes the demodulator outputs in an iterative manner.
- At each step in the trellis, it compares the metric of all paths entering each state, and *keeps only the path with the largest metric, called the survivor, together with its metric*.
- It proceeds in the trellis by *eliminating the least likely paths.*

It reduces the decoding complexity to $L2^{K-1}$ **!**

THE VITERBI ALGORITHM - CONT'D

Viterbi algorithm:

A. Do the following set up:

- For a data block of *L* bits, form the trellis. The trellis has *L+K-1* sections or levels and starts at time t_1 and ends up at time t_{L+K}.
- Label all the branches in the trellis with their corresponding <u>*branch metric*</u>.
- For each state in the trellis at the time t_i which is denoted by $S(t_i) \in \{0,1,\ldots,2^{K-1}\}$, define $\Gamma\left(S(t_i),t_i\right)$ parameter (path metric)

B. Then, do the following:

THE VITERBI ALGORITHM - CONT'D

1. Set $\Gamma(0, t_1) = 0$ and $i = 2$.
2. At time t_i, compute the partial path metrics for all the paths entering each state.
3. Set $\Gamma(S(t_i), t_i)$ equal to the best partial path metric entering each state at time t_i.

   Keep the survivor path and delete the dead paths from the trellis.
4. If $i < L + K$, increase i by 1 and return to step 2.

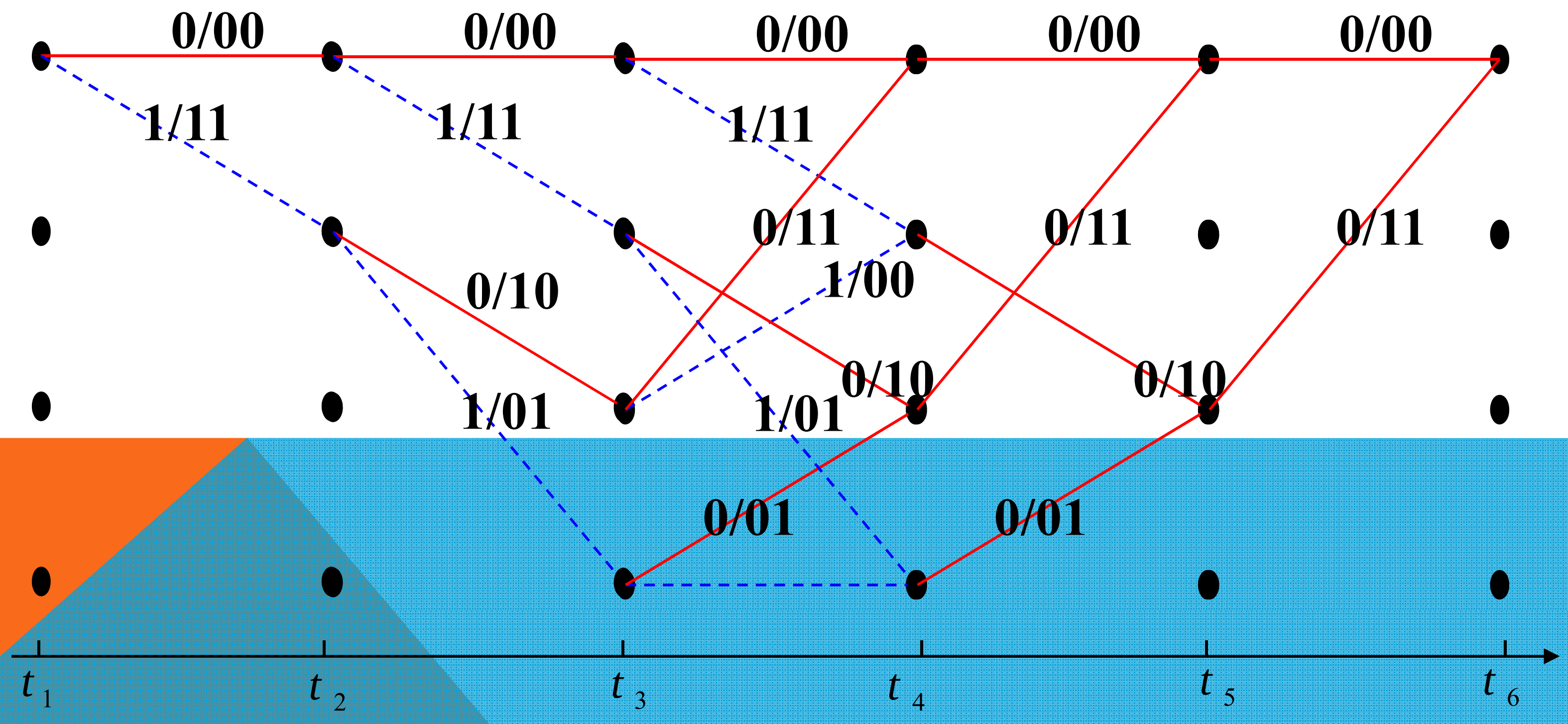
C. Start at state zero at time t_{L+K}. Follow the surviving branches backwards through the trellis. The path thus defined is unique and correspond to the ML codeword

EXAMPLE OF VITERBI DECODING

$\mathbf{m} = (101)$

$\mathbf{U} = (11 \quad 10 \quad 00 \quad 10 \quad 11)$

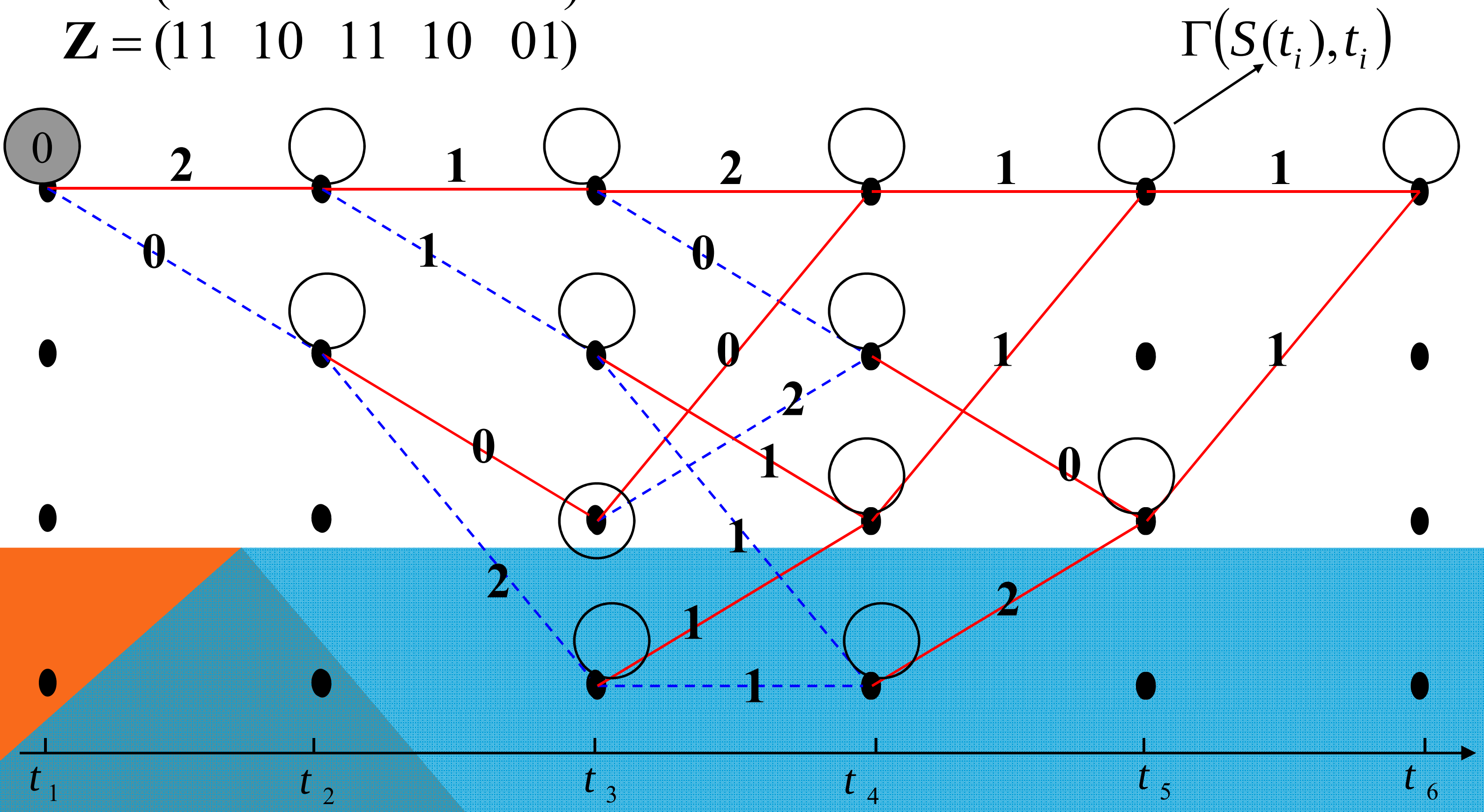
$\mathbf{Z} = (11 \quad 10 \quad 11 \quad 10 \quad 01)$



VITERBI DECODING-CONT'D

Label al the branches with the branch metric (Hamming distance)

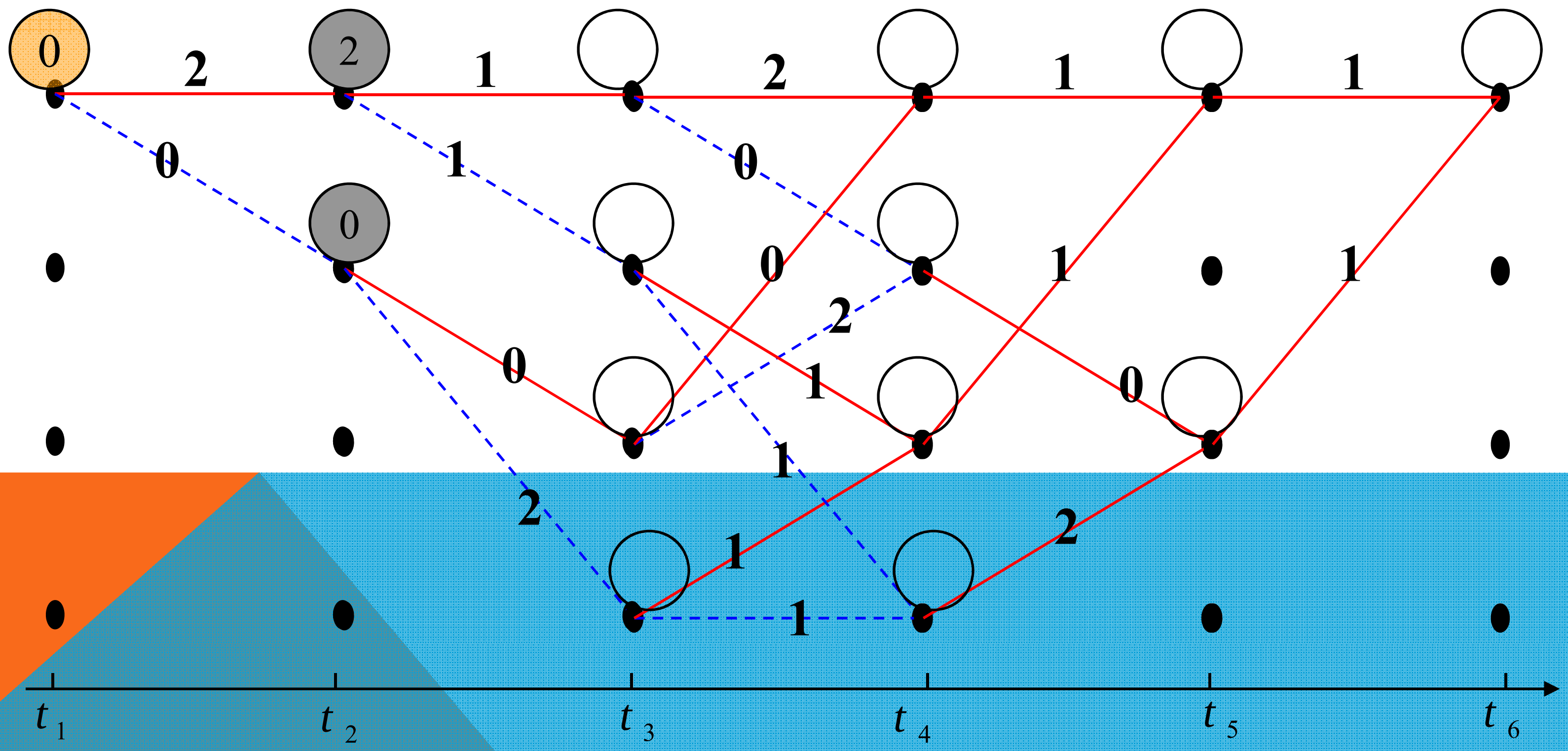
$$\mathbf{m} = (101)$$
$$\mathbf{U} = (11 \quad 10 \quad 00 \quad 10 \quad 11)$$
$$\mathbf{Z} = (11 \quad 10 \quad 11 \quad 10 \quad 01)$$



VITERBI DECODING-CONT'D

i=2

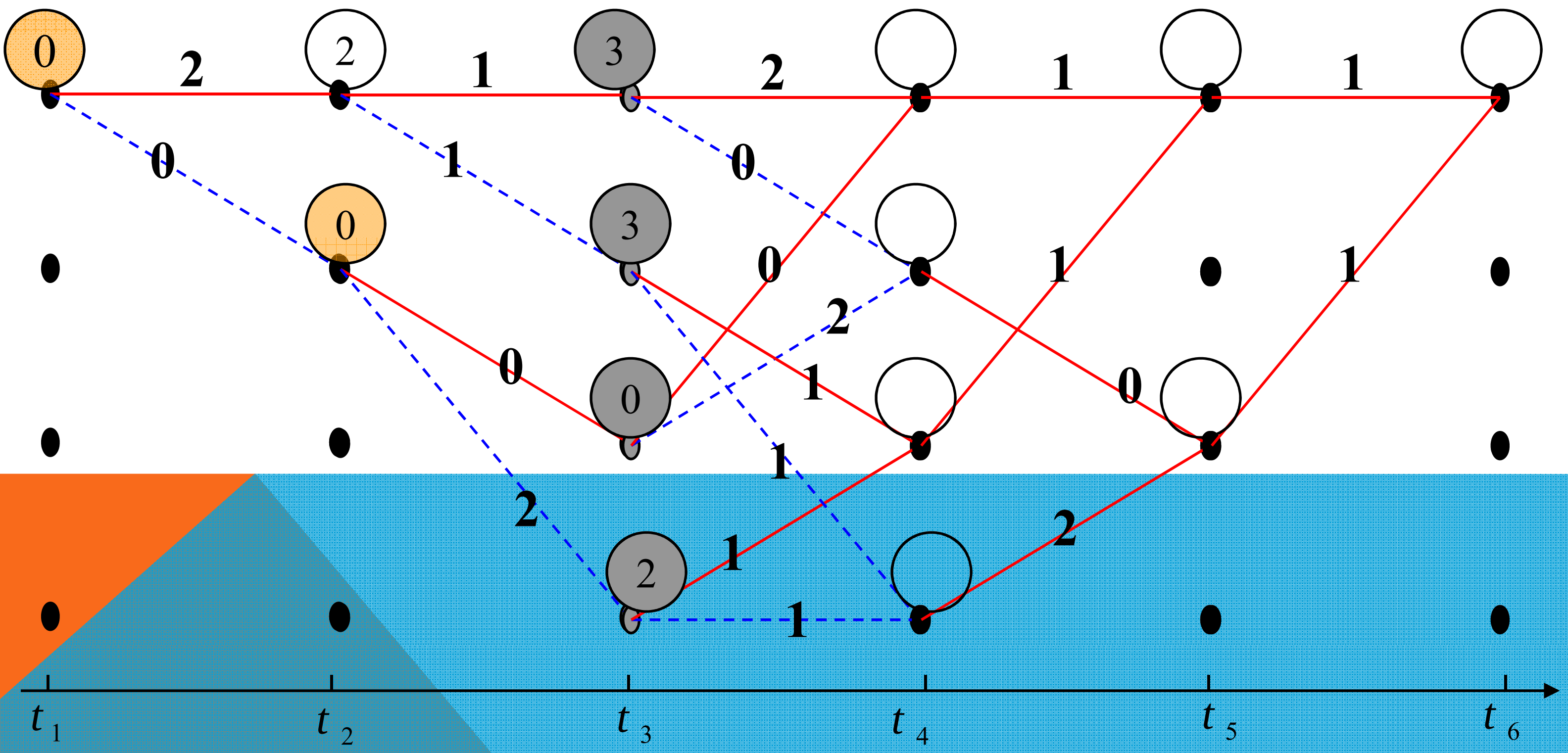
$$\mathbf{m} = (101)$$
$$\mathbf{U} = (11\ \ 10\ \ 00\ \ 10\ \ 11)$$
$$\mathbf{Z} = (11\ \ 10\ \ 11\ \ 10\ \ 01)$$



VITERBI DECODING-CONT'D

i=3

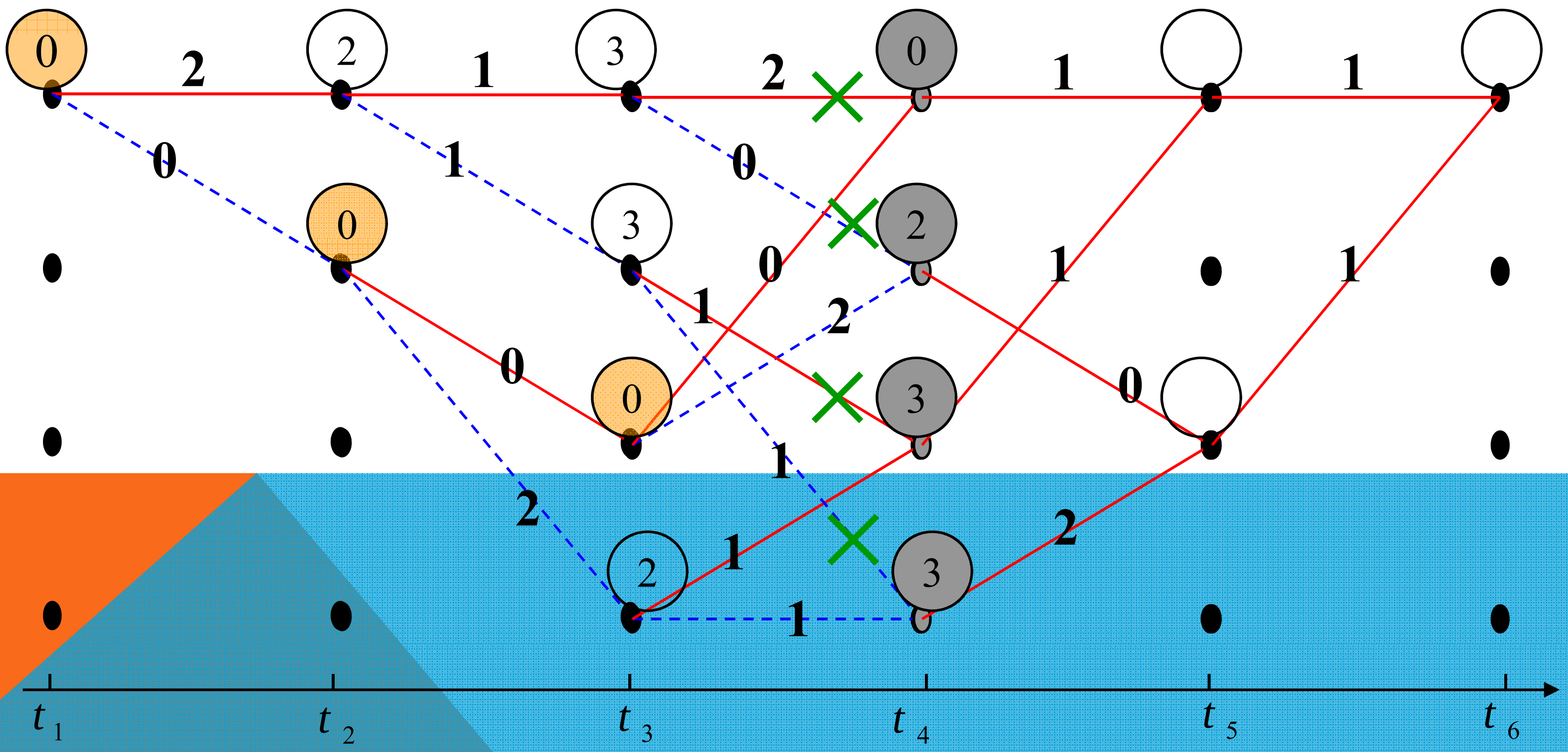
$$\mathbf{m} = (101)$$
$$\mathbf{U} = (11 \quad 10 \quad 00 \quad 10 \quad 11)$$
$$\mathbf{Z} = (11 \quad 10 \quad 11 \quad 10 \quad 01)$$



VITERBI DECODING-CONT'D

i=4

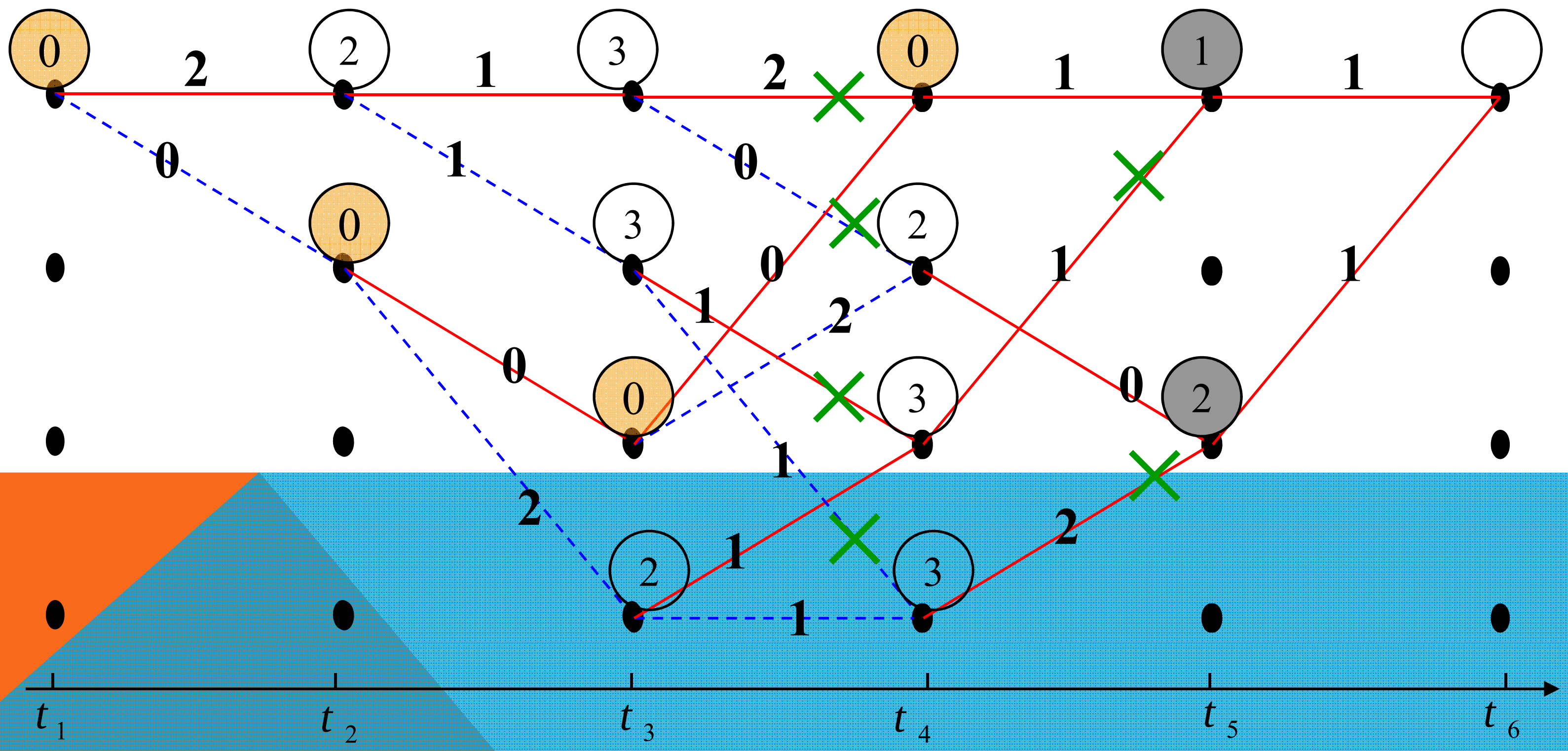
$$\mathbf{m} = (101)$$
$$\mathbf{U} = (11 \quad 10 \quad 00 \quad 10 \quad 11)$$
$$\mathbf{Z} = (11 \quad 10 \quad 11 \quad 10 \quad 01)$$



VITERBI DECODING-CONT'D

i=5

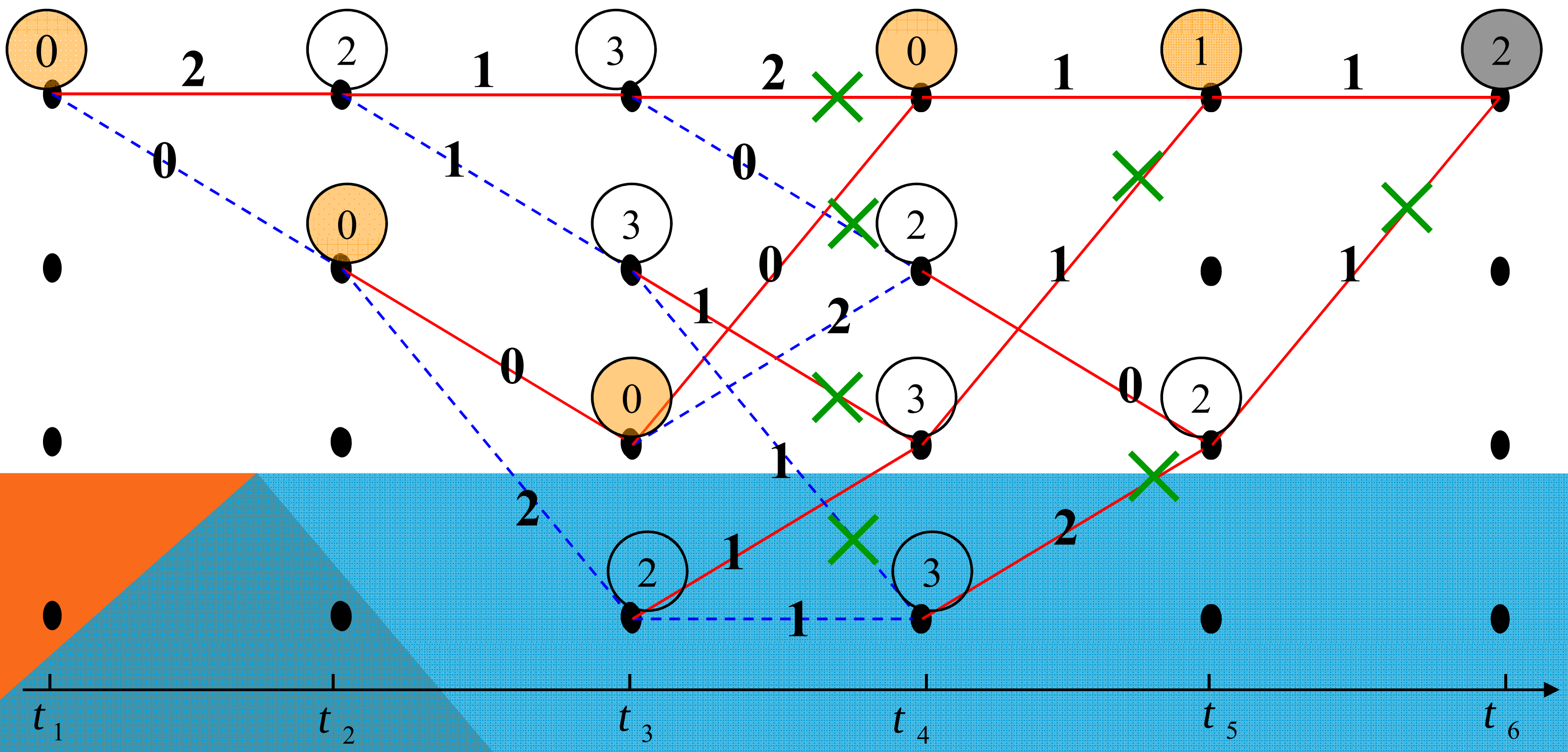
$$\mathbf{m} = (101)$$
$$\mathbf{U} = (11 \quad 10 \quad 00 \quad 10 \quad 11)$$
$$\mathbf{Z} = (11 \quad 10 \quad 11 \quad 10 \quad 01)$$



VITERBI DECODING-CONT'D

i=6

$$\mathbf{m} = (101)$$
$$\mathbf{U} = (11\ \ 10\ \ 00\ \ 10\ \ 11)$$
$$\mathbf{Z} = (11\ \ 10\ \ 11\ \ 10\ \ 01)$$



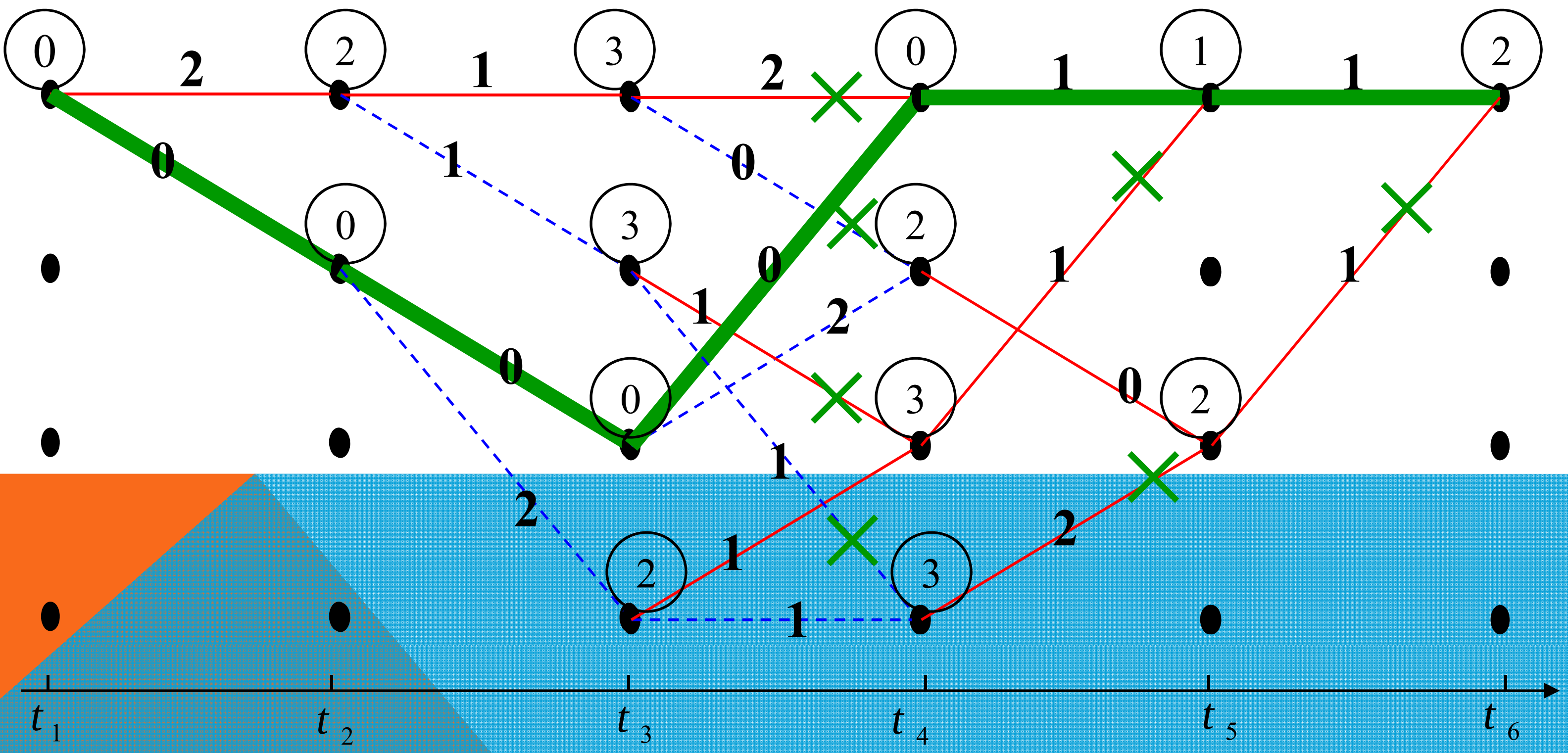
VITERBI DECODING-CONT'D

Trace back and then:

$$\hat{\mathbf{m}} = (100)$$
$$\hat{\mathbf{U}} = (11 \quad 10 \quad 11 \quad 00 \quad 00)$$

vs

$$\mathbf{m} = (101)$$
$$\mathbf{U} = (11 \quad 10 \quad 00 \quad 10 \quad 11)$$
$$\mathbf{Z} = (11 \quad 10 \quad 11 \quad 10 \quad 01)$$



SOFT AND HARD DECISIONS

Hard decision:

- The demodulator makes a firm or hard decision whether one or zero is transmitted and provides no other information reg. how reliable the decision is.
- Hence, its *output is only zero or one* (the output is quantized only to two level) which are called "*hard-bits*".

Soft decision:

- The demodulator provides the decoder with some *side information* together with the decision.
- The side information provides the decoder with a *measure of confidence for the decision*.
- The demodulator outputs which are called soft-bits, are quantized to more than two levels. (eg: 8-levels)

Decoding based on soft-bits, is called the "soft-decision decoding".

On AWGN channels, 2 dB and on fading channels 6 dB gain are obtained by using soft-decoding over hard-decoding!

PERFORMANCE BOUNDS …

Basic coding gain (dB) for *soft-decision* Viterbi decoding

Uncoded E_b/N_0 (dB)	Code rate P_B	K	1/3 7	1/3 8	1/2 6	1/2 7
6.8	10^{-3}		4.2	4.4	3.5	3.8
9.6	10^{-5}		5.7	5.9	4.6	5.1
11.3	10^{-7}		6.2	6.5	5.3	5.8
Upper bound			7.0	7.3	6.0	7.0

INTERLEAVING

Convolutional codes are suitable for memoryless channels with random error events.

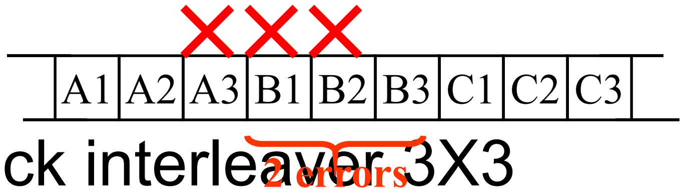
Some errors have bursty nature:

- Statistical dependence among successive error events (time-correlation) due to the channel memory.
  - Like errors in multipath fading channels in wireless communications, errors due to the switching noise, …

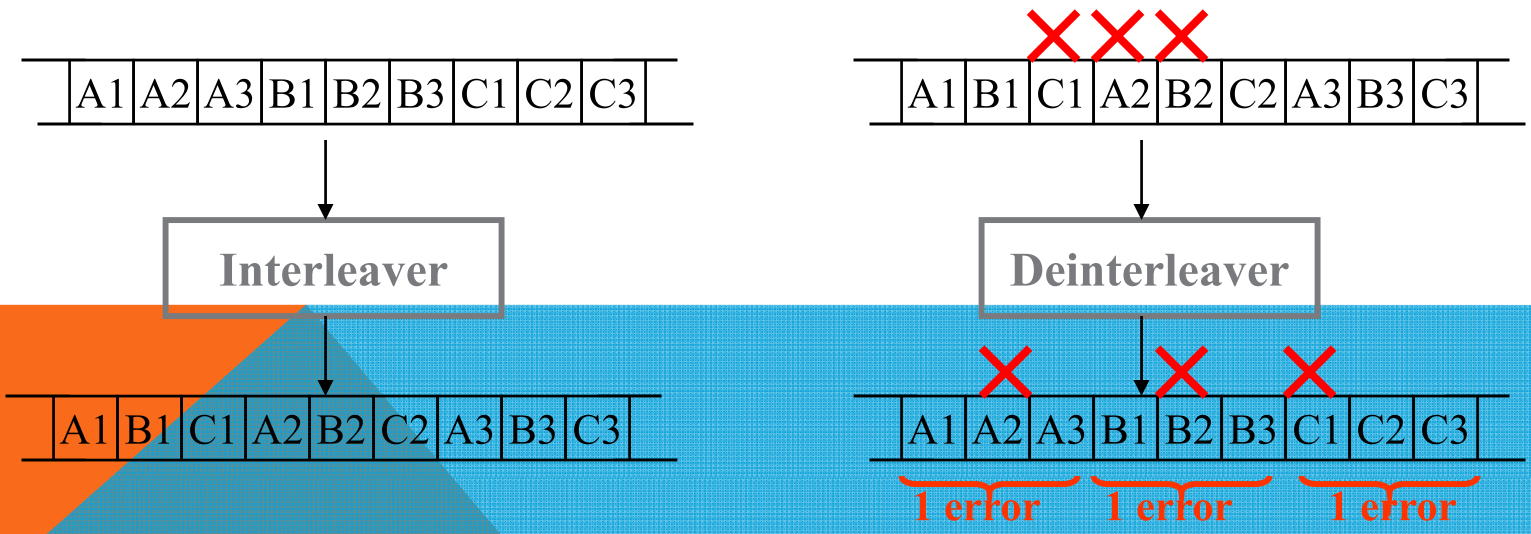
"Interleaving" makes the channel looks like as a memoryless channel at the decoder.

INTERLEAVING ...

- Consider a code with t=1 and 3 coded bits.
- A burst error of length 3 can not be corrected.



- Let us use a block interleaver 3X3



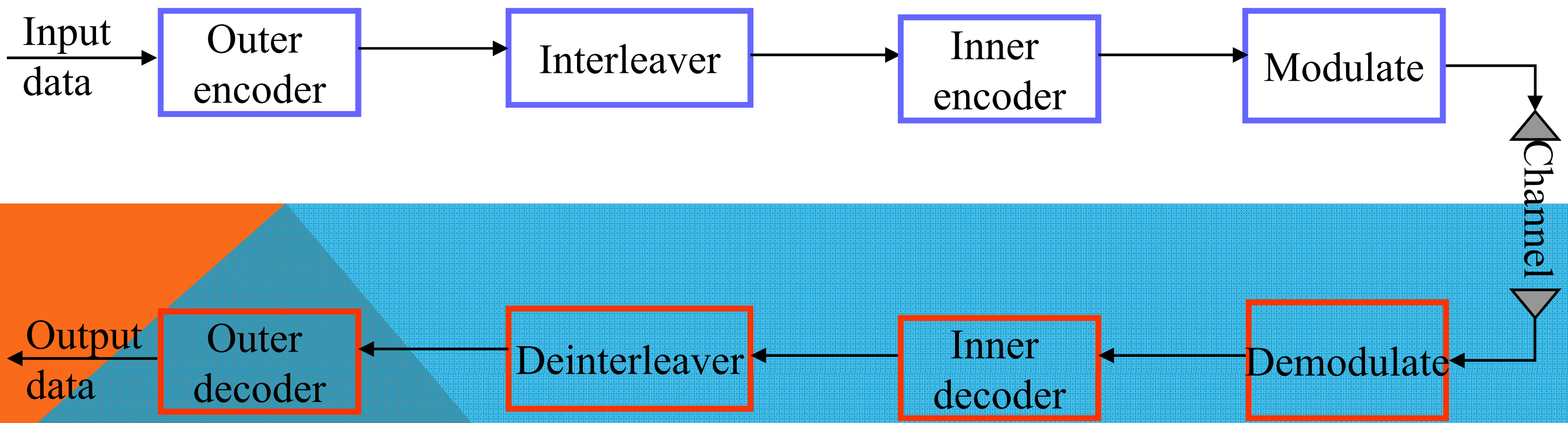
CONCATENATED CODES

CONCATENATED CODES

A concatenated code uses two levels on coding, an inner code and an outer code (higher rate).

- Popular concatenated codes: Convolutional codes with Viterbi decoding as the inner code and Reed-Solomon codes as the outer code

The purpose is to reduce the overall complexity, yet achieving the required error performance.



CONCATENATED CODES

$$C' \to \underbrace{C \to Q \to D}_{Q'} \to D'$$

Encoder-channel-decoder system C → Q → D can b viewed as defining a ***super-channel*** Q’ with a smaller probability of error, and with complex ***correlations*** among its errors.

We can create an encoder C’ and decoder D’ for this super-channel Q’.

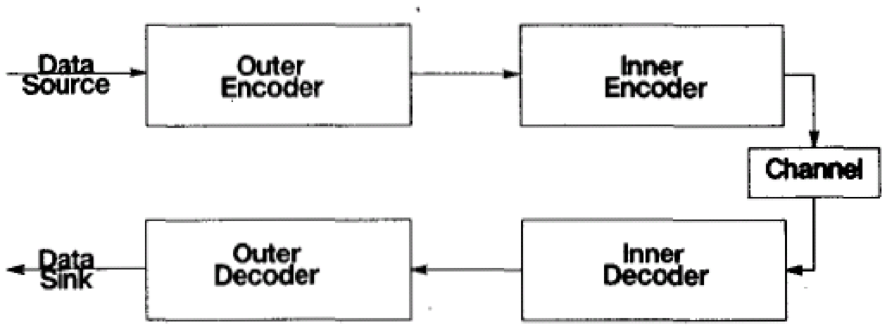


Fig. 4. Concatenated Code Schematic.

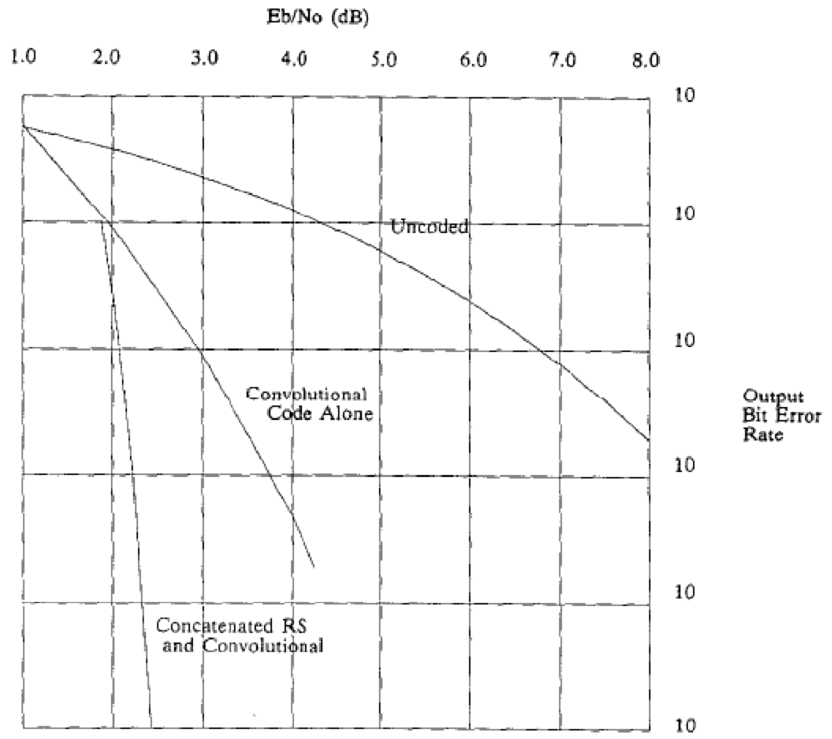


Fig. 10. Performance of RS(255,223) and (2,1), K=7 Conv. Code.

PRODUCT/RECTANGULAR CODES: CONCATENATION + INTERLEAVING

Some concatenated codes make use of the idea of *interleaving*.

Blocks of size *larger* than the block lengths of the constituent codes C and C'.

- After encoding the data of one block using code C',
- … the bits are *reordered* within the block in such a way that nearby bits are separated from each other once the block is fed to the second code C.

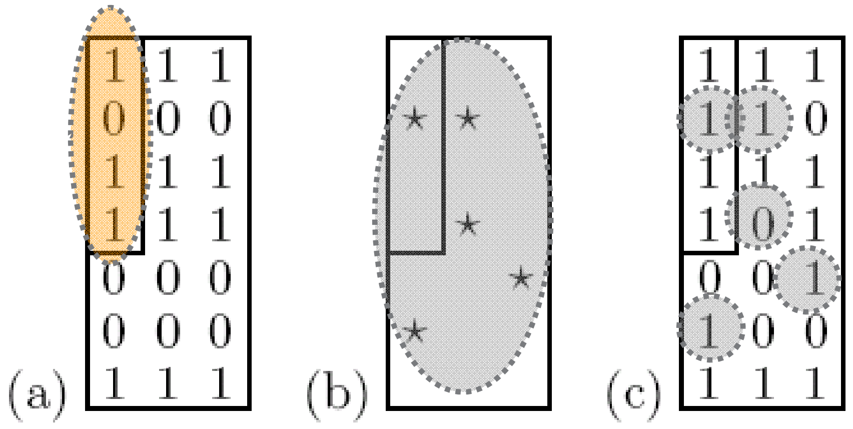
A simple example of an interleaver is a *rectangular code* or *product code* in which …

… the data: $K_2 \times K_1$ *rectangular block*, and …

… encoded *horizontally* using an (N_1,K_1) linear code,

… then *vertically* using a (N_2,K_2) linear code.

PRODUCT CODE EXAMPLE

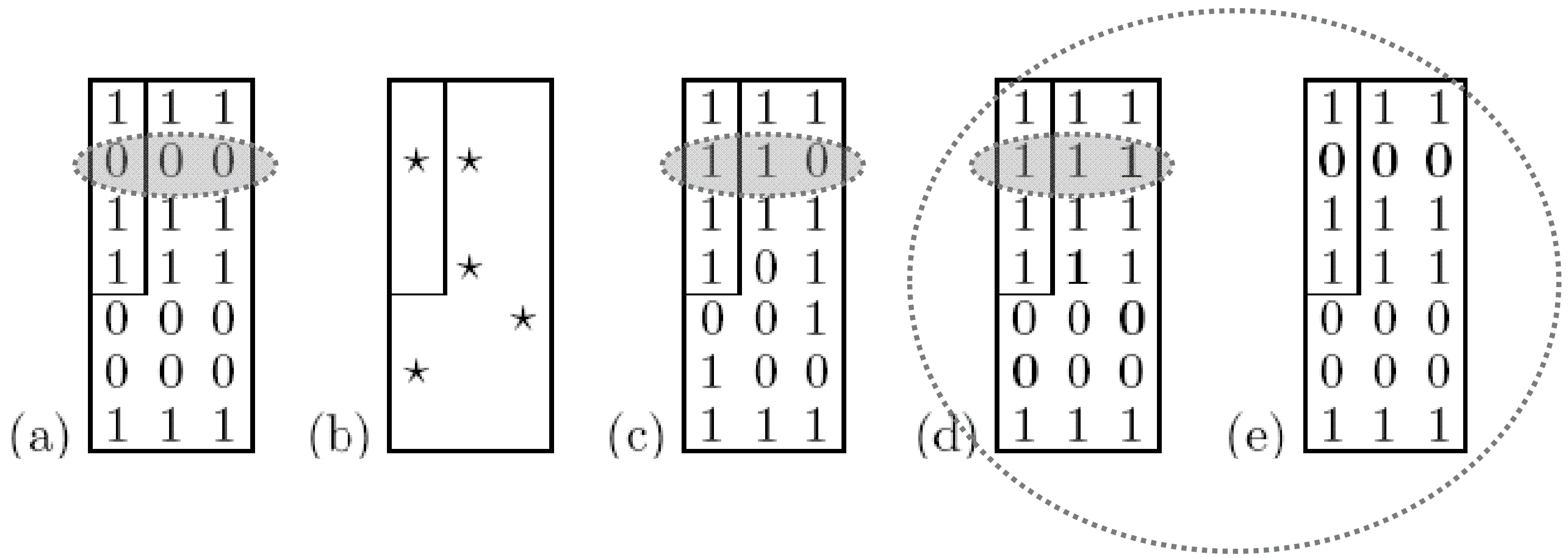


(a) A string 1011 encoded using a concatenated code w/ two Hamming codes, H(3, 1) ≡ Repetition code (R3) and H(7,4).

(b) a noise pattern that flips 5 bits.

(c) The received vector.

PRODUCT CODES (CONTD)



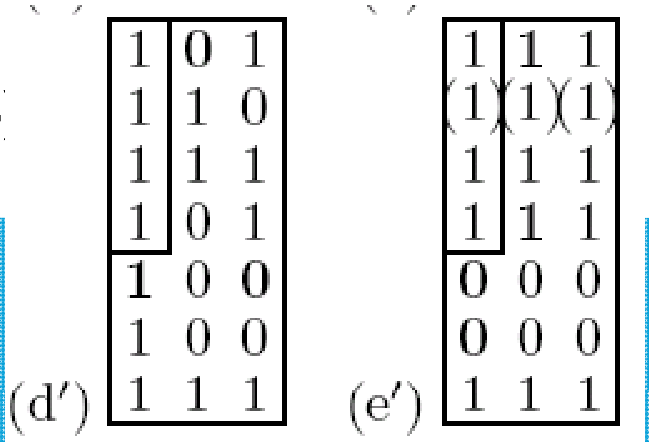
(d) After decoding using the horizontal (3, 1) decoder, and

(e) after subsequently using the vertical (7; 4

decoder.

The decoded vector matches the original.

Note: Decoding in the other order (weaker-code-first) leads to residual error in this example:



PRACTICAL EXAMPLE: COMPACT DISC

"Without error correcting codes, digital audio would not be technically feasible."

Channel in a CD playback system consists of a transmitting laser, a recorded disc and a photo-detector.

Sources of errors are manufacturing damages, fingerprints or scratches

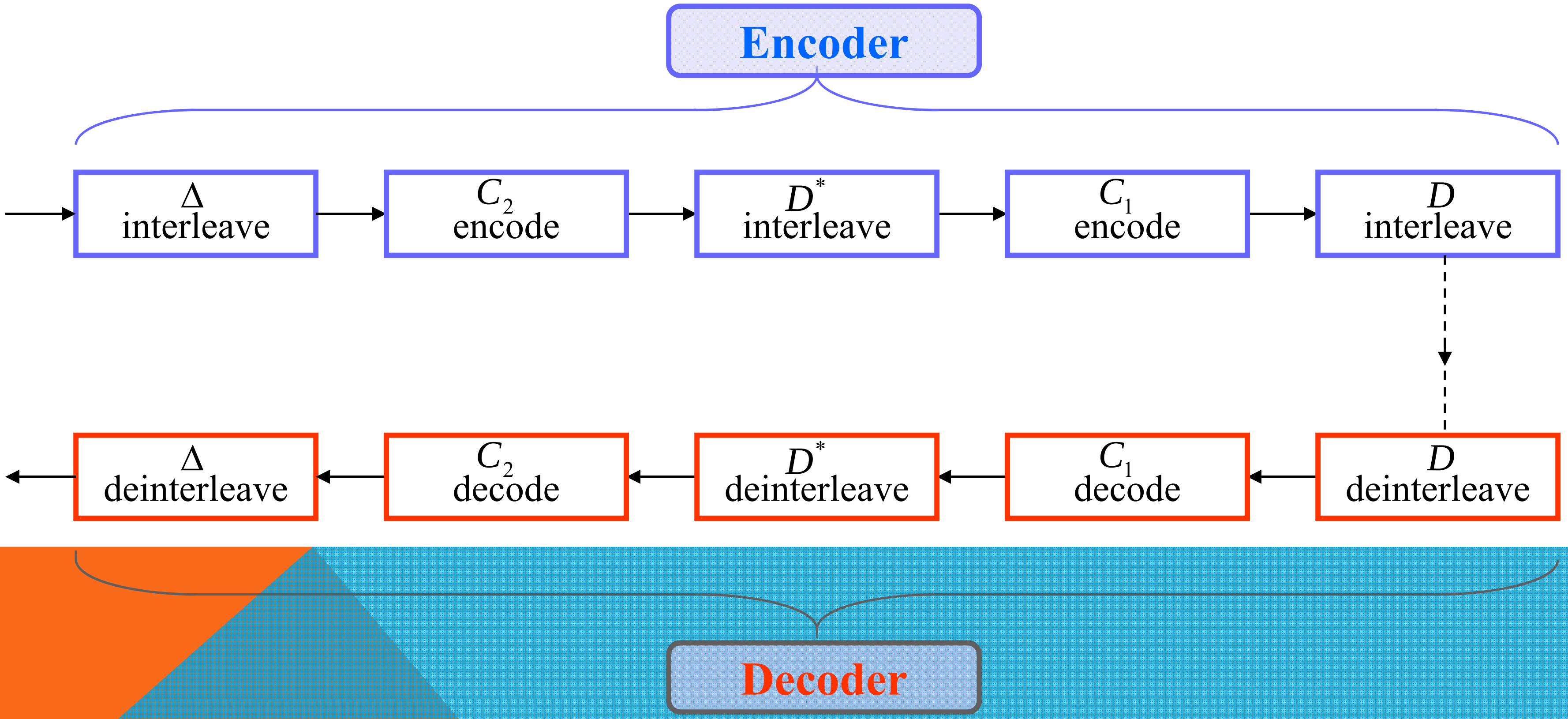
Errors have bursty like nature.

Error correction and concealment is done by using a concatenated error control scheme, called ***cross-interleaver Reed-Solomon code (CIRC).***

Both the inner and outer codes are ***shortened RS codes***

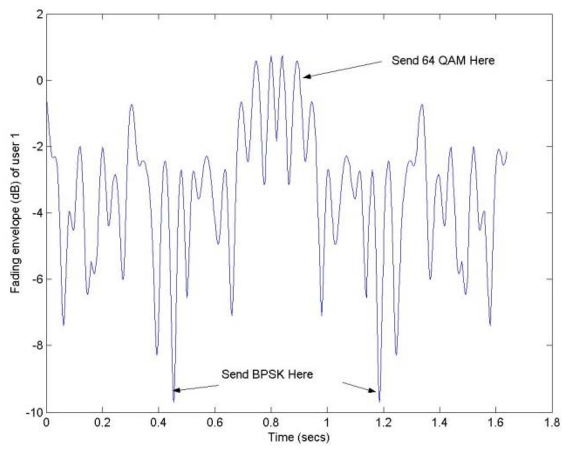
COMPACT DISC – CIRC ENCODER

CIRC encoder and decoder:



ADAPTIVE MODULATION AND CODING

ADAPTIVE MODULATION



Just vary the “M” in the MQAM constellation to the appropriate SNR

Can be used in conjunction with spatial diversity

ADAPTIVE MODULATION/CODING: MULTI-USER

Exploit *<u>multi-user</u>* diversity.

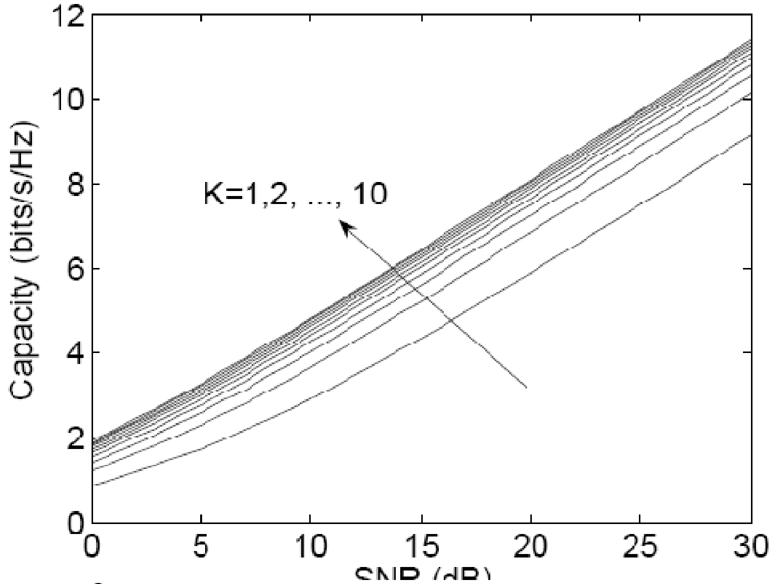
- Users with **<u>high SNR</u>**: use MQAM (large M) + high code rates
- Users with **<u>low SNR:</u>** use BPSK + low code rates (i.e. heavy error protection)

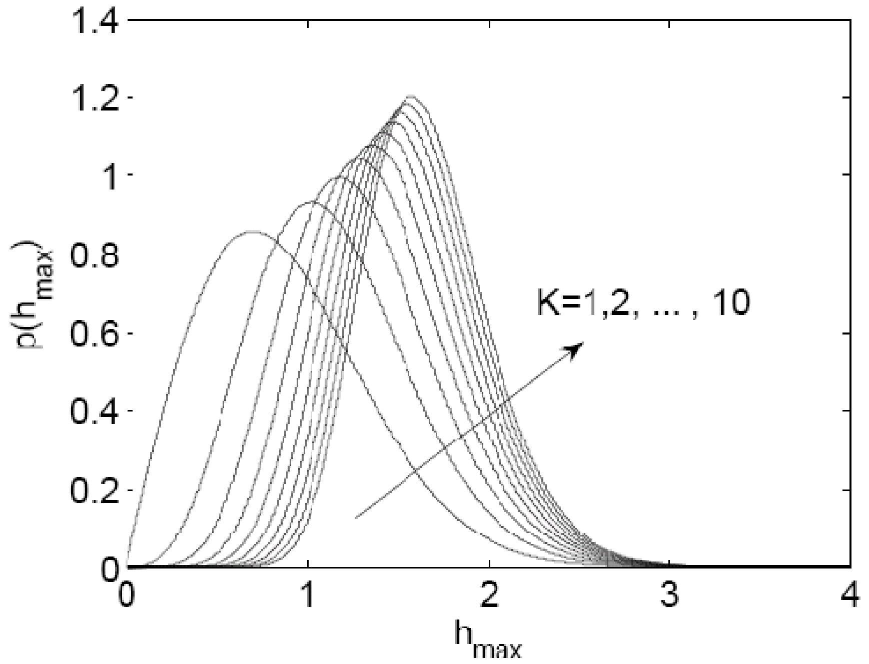
In any WiMAX frame, different users (assigned to time-frequency slots within a frame) would be getting a different rate!

- i.e. be using different code/modulation combos..

BASIS FOR ADAPTIVE MODULATION/CODING (AMC)

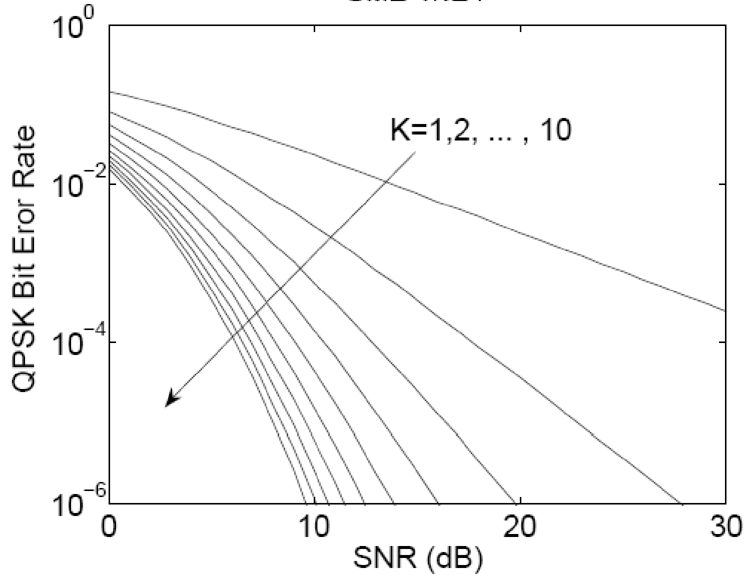
K-user system: the subcarrier of interest experiences i.i.d. Rayleigh fading: each user's channel gain is independent of the others, and is denoted by *h_k*.





Probability density function of h_{max}.

the maximum of the K users channel gains.



WIMAX: USES FEEDBACK & BURST PROFILES

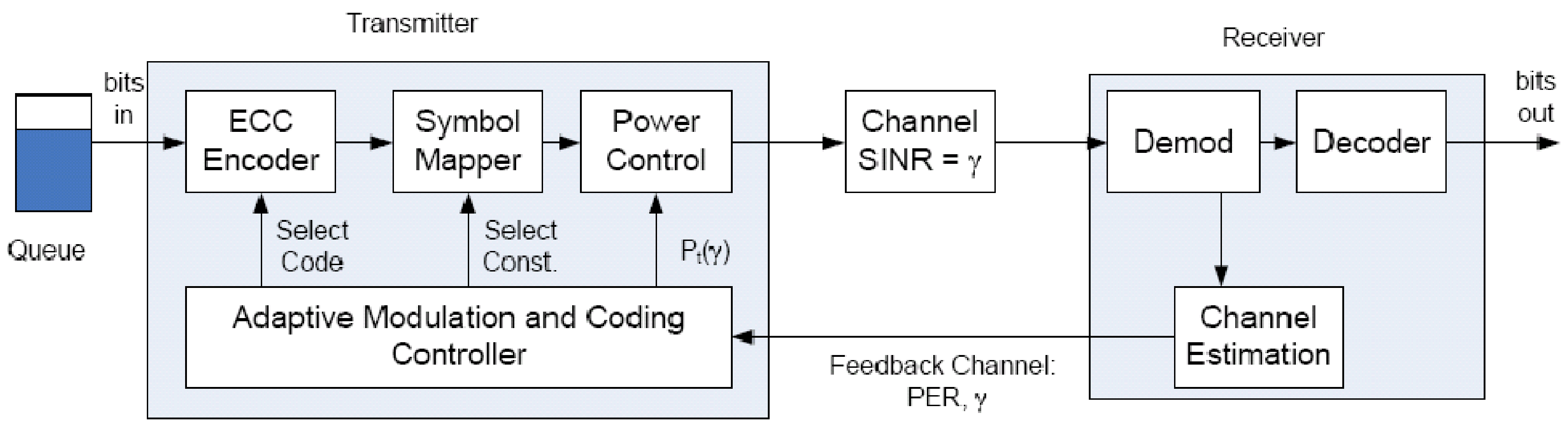


Figure 6.7: Adaptive Modulation and Coding Block Diagram.

Lower data rates are achieved by using a small constellation – such as QPSK – and low rate error correcting codes such as rate 1/2 convolutional or turbo codes.

The higher data rates are achieved with large constellations – such as 64QAM – and less robust error correcting codes, for example rate 3/4 convolutional, turbo, or LDPC codes.

Wimax burst profiles: 52 different possible configurations of modulation order and coding types and rates.

WiMAX systems heavily protect the feedback channel with error correction, so usually the main source of degradation is due to mobility, which causes channel estimates to rapidly become obsolete.

AMC CONSIDERATIONS

<u>BLER and Received SINR:</u> In adaptive modulation theory, the transmitter needs only to know the statistics and instantaneous channel SINR. From the channel SINR, it can determine the optimum coding/modulation strategy and transmit power.

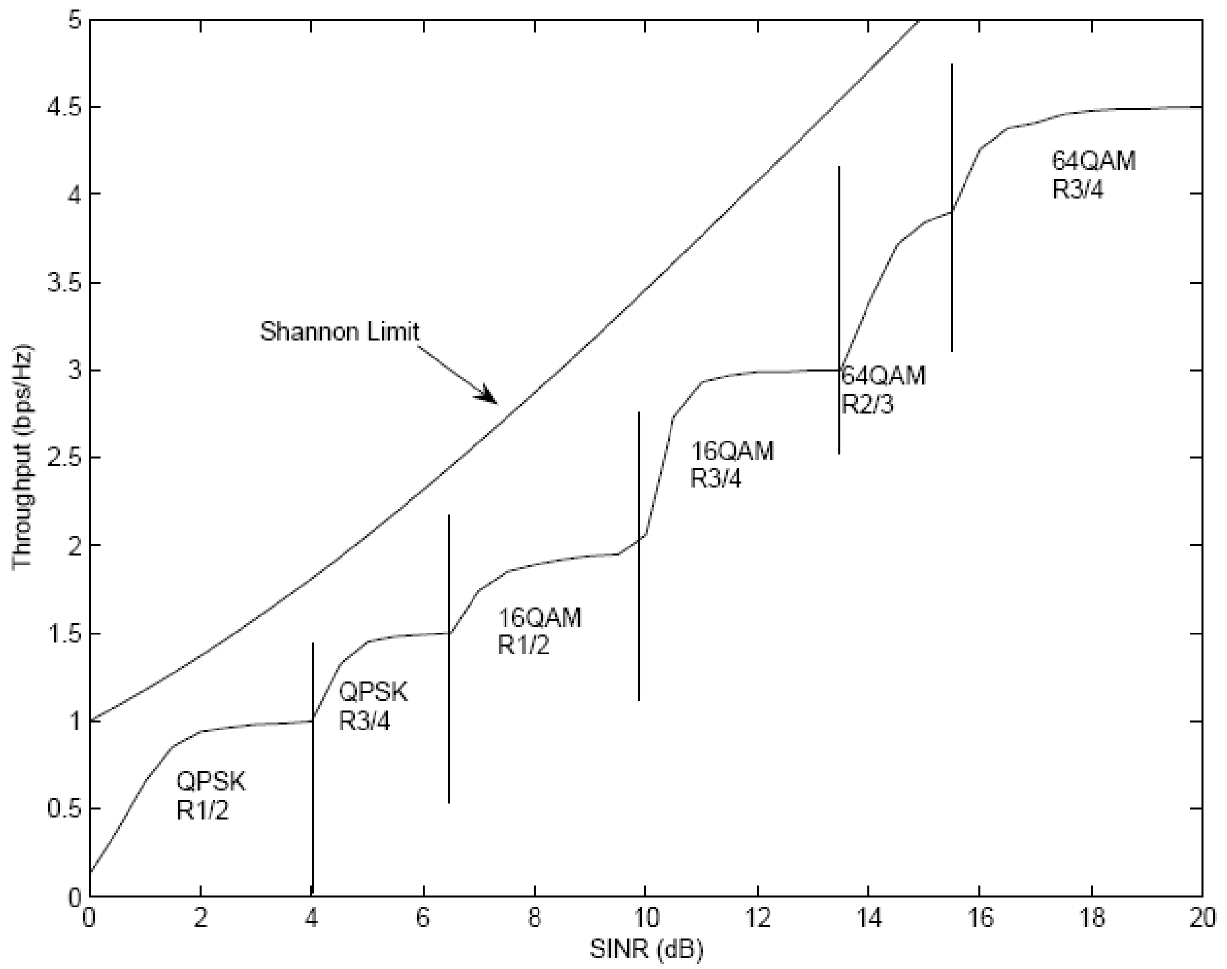
- In practice however, the BLER should be carefully monitored as the final word on whether the data rate should be increased (if the BLER is low) or decreased to a more robust setting.

<u>Automatic Repeat Request (ARQ):</u> ARQ allows rapid retransmissions, and Hybrid ARQ generally increases the ideal BLER operating point by about a factor of 10, e.g. from 1% to 10%.

- For delay-tolerant applications, it may be possible to accept a BLER approaching even 70%, if Chase combining is used in conjunction with HARQ to make use of unsuccessful packets.

<u>Power control vs. Waterfilling:</u> In theory, the best power control policy from a capacity standpoint is the so-called waterfilling strategy, in which more power is allocated to strong channels, and less power allocated to weak channels. In practice, the opposite may be true in some cases.

AMC VS SHANNON LIMIT



Optionally turbo-codes or LDPC codes can be used instead of simple block/convolutional codes in these schemes

MAIN POINTS

Adaptive MQAM uses capacity-achieving power and rate adaptation, with power penalty K.

Adaptive MQAM comes within 5-6 dB of capacity

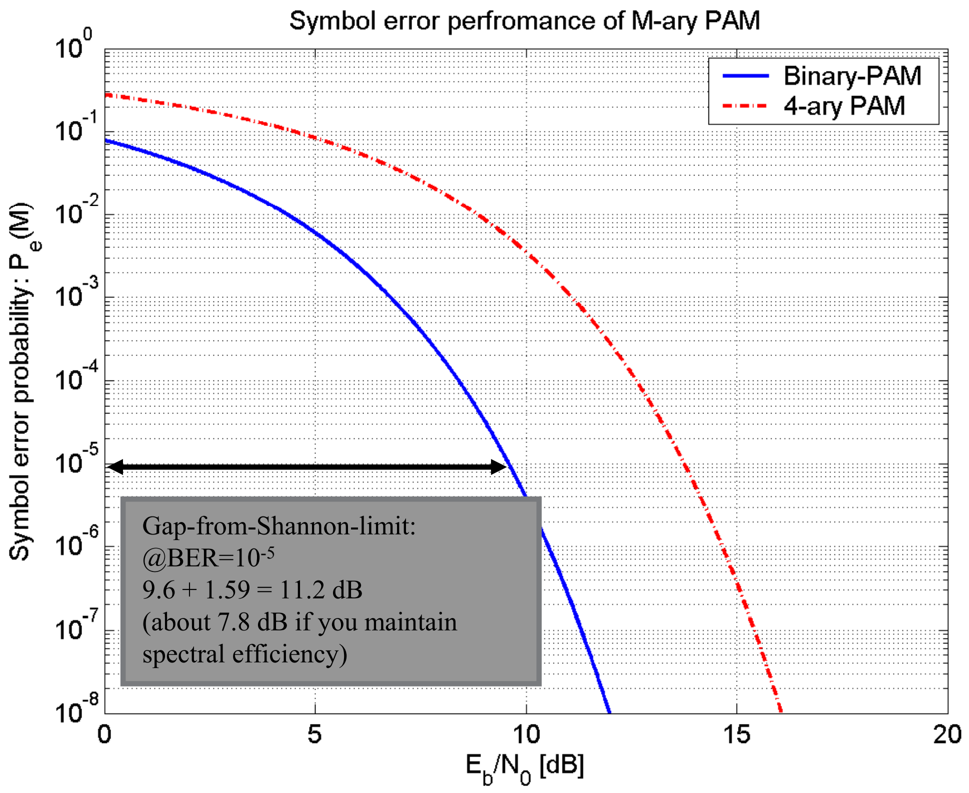
Discretizing the constellation size results in negligible performance loss.

Constellations cannot be updated faster than 10s to 100s of symbol times: OK for most dopplers.

Estimation error and delay lead to irreducible error floors.

TOWARDS THE SHANNON LIMIT!
LDPC, TURBO CODES, DIGITAL FOUNTAINS

RECALL: CODING GAIN POTENTIAL



Gap-from-Shannon-limit:
@BER=10^{-5}
9.6 + 1.59 = 11.2 dB
(about 7.8 dB if you maintain spectral efficiency)

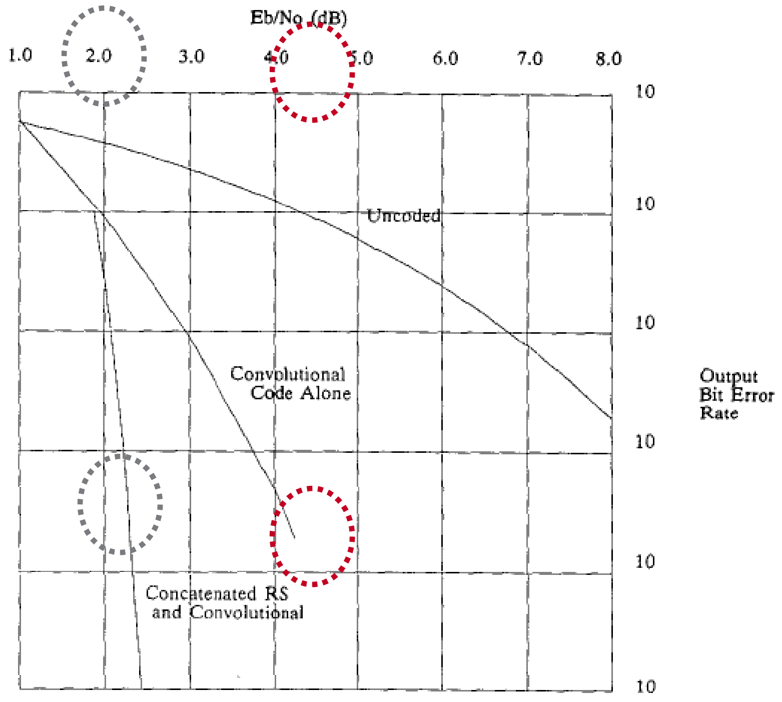


Fig. 10. Performance of RS(255,223) and (2,1), K=7 Conv. Code.

With convolutional code alone, @BER of 10^{-5}, we require Eb/No of 4.5dB or get a gain of 5.1 dB.

With concatenated RS-Convolutional code, BER curve ~ vertical cliff at an Eb/No of about 2.5-2.6 dB, i.e a gain of 7.1dB.

We are still 11.2 – 7.1 = 4.1 dB away from the Shannon limit ☹

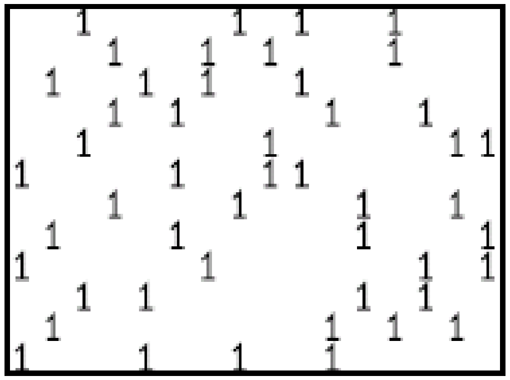
Turbo codes and LDPC codes get us within 0.1dB of the Shannon limit !! ☺

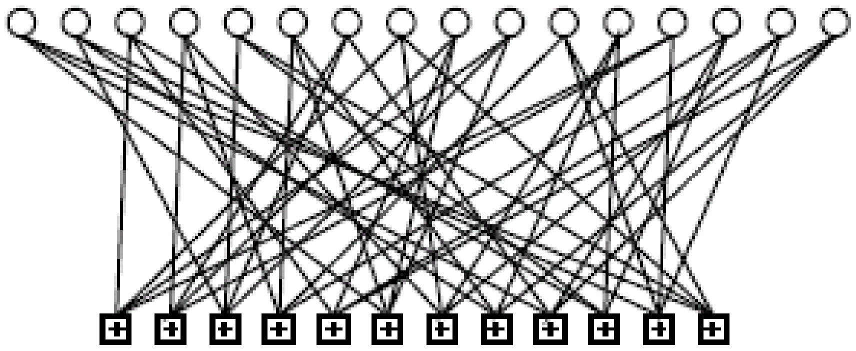
LOW-DENSITY PARITY CHECK (LDPC) CODES

LDPC

- *Low-Density Parity-Check* (LDPC) codes are a class of linear block codes characterized by sparse parity check matrices **H**
  - **H** has a low-density of 1's
- LDPC codes were originally invented by Robert Gallager in the early 1960's but were largely ignored until they were "rediscovered" in the mid-1990's by MacKay
- Sparseness of **H** can yield large minimum distance d_{min} and reduces decoding complexity
- Can perform within 0.0045 dB of Shannon limit
- These code are making their way into standards
  - Binary turbo: UMTS, cdma2000
  - Duobinary turbo: DVB-RCS, 802.16
  - LDPC: DVB-S2 standard.

EXAMPLE LDPC CODE

$H =$ 



A low-density parity-check matrix and the corresponding (bipartite) graph of a rate-1/4 low-density parity-check code with blocklength N =16, and M =12 constraints.

Each white circle represents a transmitted bit.

Each bit participates in j = 3 *constraints*, represented by squares $\boxplus$.

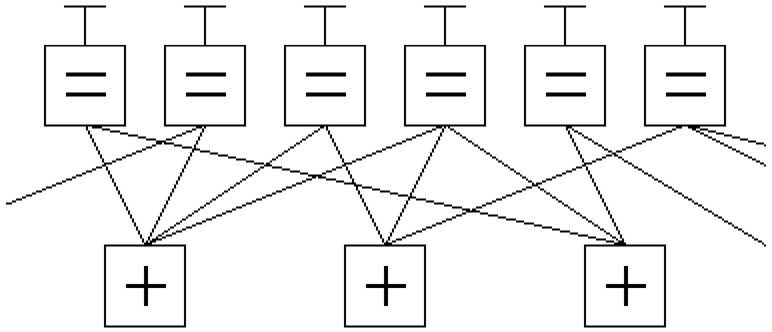
Each constraint forces the sum of the k = 4 bits to which it is connected to be even.

This code is a (16; 4) code. Outstanding performance is obtained when the blocklength is increased to N ≈ 10,000.

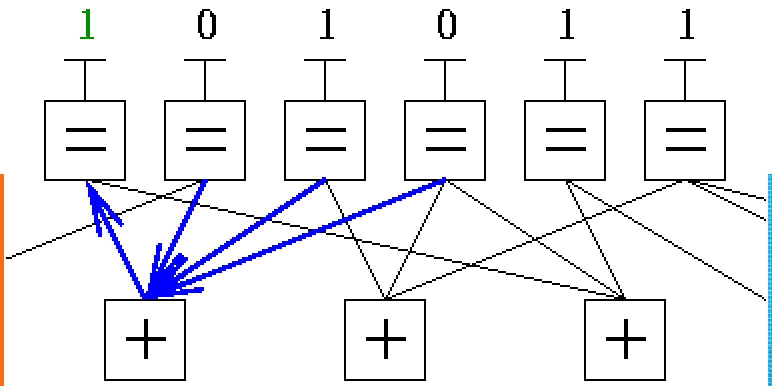
TANNER GRAPH

- A ***Tanner graph*** is a *bipartite* graph that describes the parity check matrix **H**
- There are two classes of nodes:
  - ***Variable-nodes:*** Correspond to bits of the codeword or equivalently, to columns of the parity check matrix
    - There are *n* v-nodes
  - ***Check-nodes:*** Correspond to parity check equations or equivalently, to rows of the parity check matrix
    - There are *m=n-k* c-nodes
  - ***Bipartite*** means that nodes of the same type cannot be connected (e.g. a c-node cannot be connected to another c-node)
- The ith check node is connected to the jth variable node iff the (i,j)th element of the parity check matrix is one, i.e. if h_{ij} =1
  - All of the v-nodes connected to a particular c-node must sum (modulo-2) to zero

A.K.A FACTOR GRAPH NOTATION



$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$



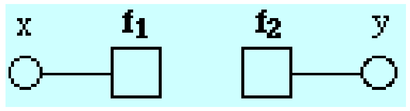
$$\mathbf{z} = \mathbf{H}\mathbf{r} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

FACTOR GRAPHS

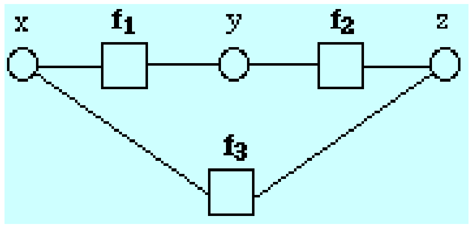
A factor graph shows how a function of several variables can be factored into a product of "smaller" functions.

For example, the function g defined by g(x,y)=xy+x can be factored into $g(x,y)=f_1(x)f_2(y)$ where $f_1(x)=x$ and $f_2(y)=y+1$.

The factor graph depicting this factorization:



Graph for function $g(x,y,z) = f_1(x,y)\ f_2(y,z)\ f_3(x,z)$.



Why Factor graphs?

1. Very *general*: variables and functions are arbitrary

2. Factorization => *Sum-Product Algorithm* can be applied

3. Third, many efficient algorithms are *special cases* of the Sum-Product Algorithm applied to factor graphs:

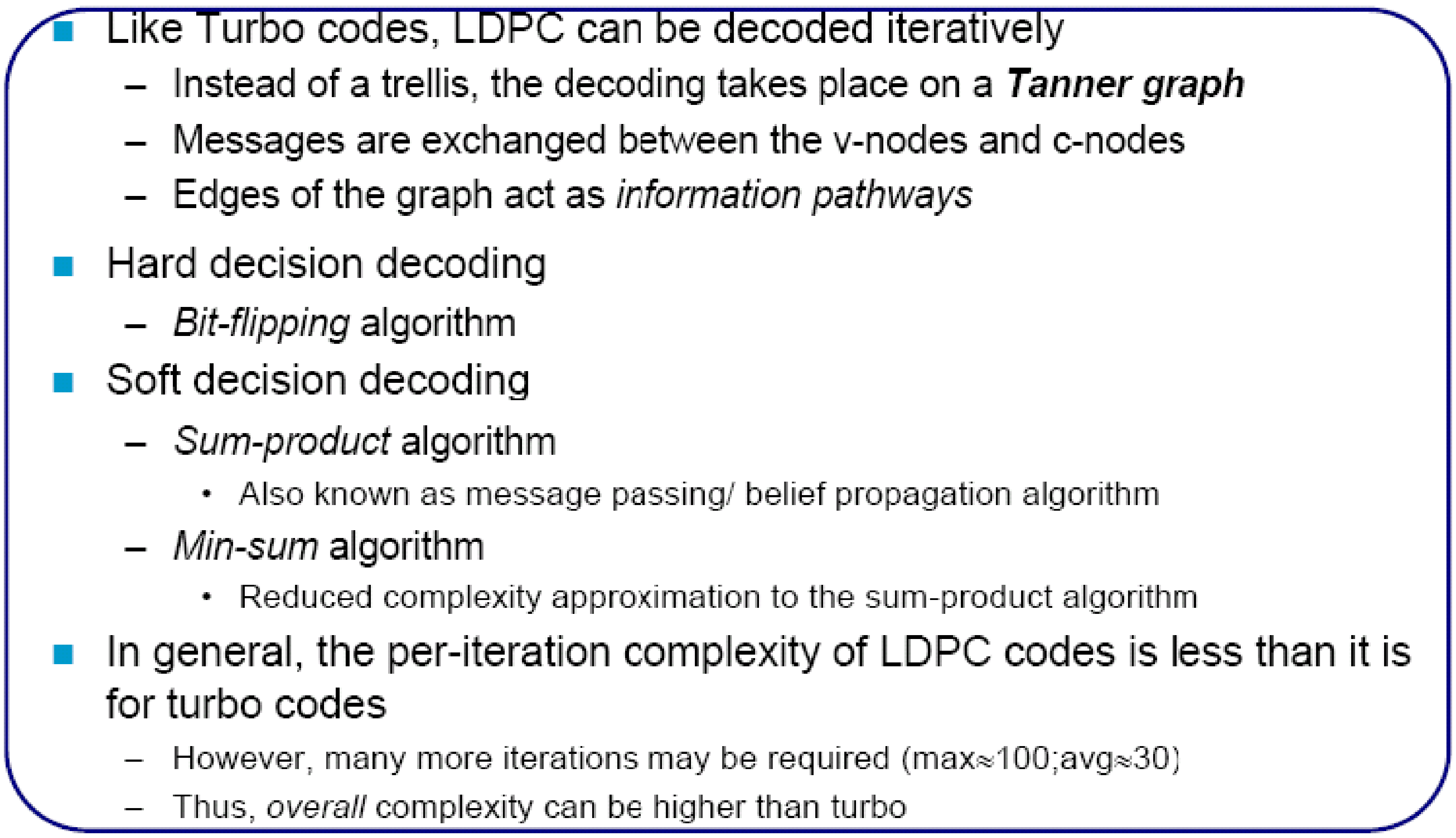
FFT (Fast Fourier Transform), Viterbi Algorithm, Forward-Backward Algorithm, Kalman Filter and Bayesian Network Belief Propagation.

Brings many good algorithms together in a common framework.

LDPC CODING CONSTRUCTIONS

- Around 1996, Mackay and Neal described methods for constructing sparse **H** matrices
- The idea is to randomly generate a $M \times N$ matrix **H** with weight d_v columns and weight d_c rows, subject to some constraints
- Construction 1A: Overlap between any two columns is no greater than 1
  - This avoids length 4 cycles
- Construction 2A: M/2 columns have d_v =2, with no overlap between any pair of columns. Remaining columns have d_v =3. As with 1A, the overlap between any two columns is no greater than 1
- Construction 1B and 2B: Obtained by deleting select columns from 1A and 2A
  - Can result in a higher rate code

LDPC DECODING: ITERATIVE



- Like Turbo codes, LDPC can be decoded iteratively
  - Instead of a trellis, the decoding takes place on a ***Tanner graph***
  - Messages are exchanged between the v-nodes and c-nodes
  - Edges of the graph act as *information pathways*
- Hard decision decoding
  - *Bit-flipping* algorithm
- Soft decision decoding
  - *Sum-product* algorithm
    - Also known as message passing/ belief propagation algorithm
  - *Min-sum* algorithm
    - Reduced complexity approximation to the sum-product algorithm
- In general, the per-iteration complexity of LDPC codes is less than it is for turbo codes
  - However, many more iterations may be required (max$\approx$100;avg$\approx$30)
  - Thus, *overall* complexity can be higher than turbo

REGULAR VS IRREGULAR LDPC CODES

- An LDPC code is ***regular*** if the rows and columns of H have uniform weight, i.e. all rows have the same number of ones (**d**$_v$) and all columns have the same number of ones (d$_c$)
  - The codes of Gallager and MacKay were regular (or as close as possible)
  - Although regular codes had impressive performance, they are still about 1 dB from capacity and generally perform worse than turbo codes
- An LDPC code is ***irregular*** if the rows and columns have non-uniform weight
  - Irregular LDPC codes tend to outperform turbo codes for block lengths of about $n>10^5$
- The degree distribution pair (λ, ρ) for a LDPC code is defined as

$$\lambda(x) = \sum_{i=2}^{d_v} \lambda_i x^{i-1}$$

$$\rho(x) = \sum_{i=1}^{d_c} \rho_i x^{i-1}$$

- λ_i, ρ_i represent the fraction of edges emanating from variable (check) nodes of degree *i*

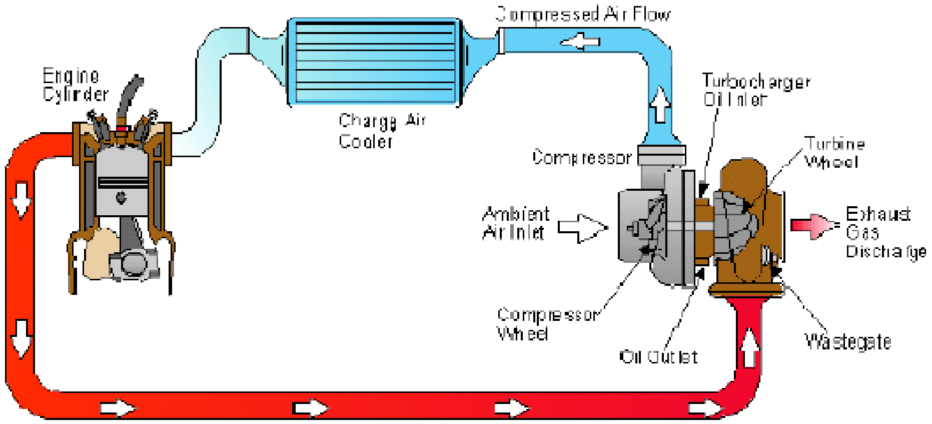
IRREGULAR LDPC CODES

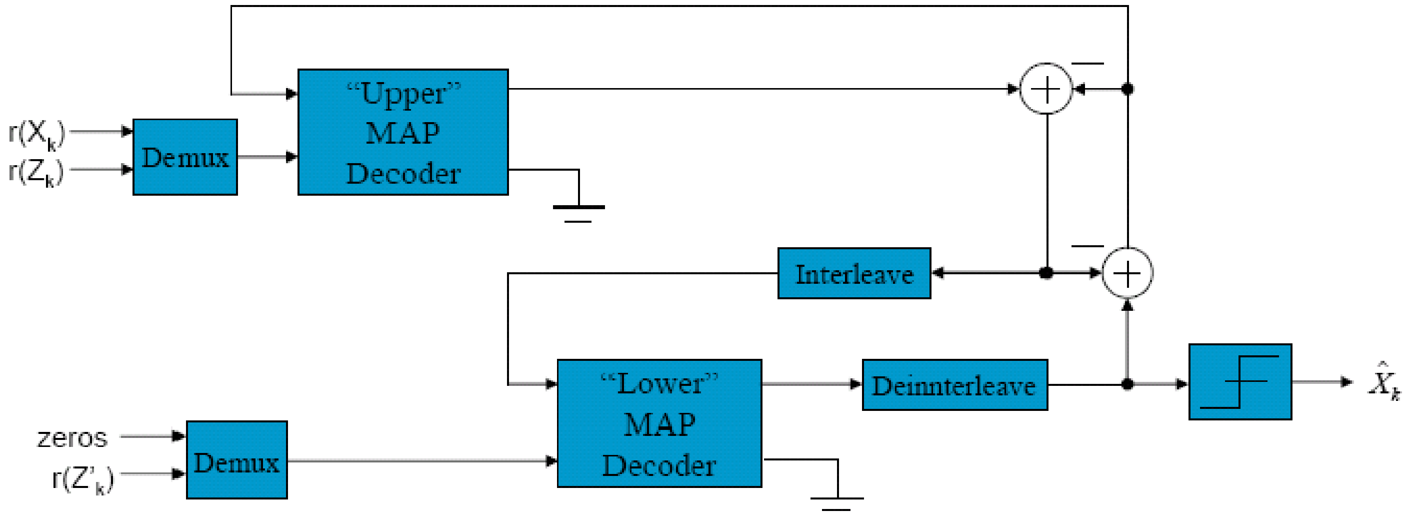
- Luby et. al. (1998) developed LDPC codes based on irregular LDPC Tanner graphs
- Message and check nodes have conflicting requirements
  - Message nodes benefit from having a large degree
  - LDPC codes perform better with check nodes having low degrees
- Irregular LDPC codes help balance these competing requirements
  - High degree message nodes converge to the correct value quickly
  - This increases the quality of information passed to the check nodes, which in turn helps the lower degree message nodes to converge
- Check node degree kept as uniform as possible and variable node degree is non-uniform
  - Code 14: Check node degree =14, Variable node degree =5, 6, 21, 23
- No attempt made to optimize the degree distribution for a given code rate

TURBO CODES

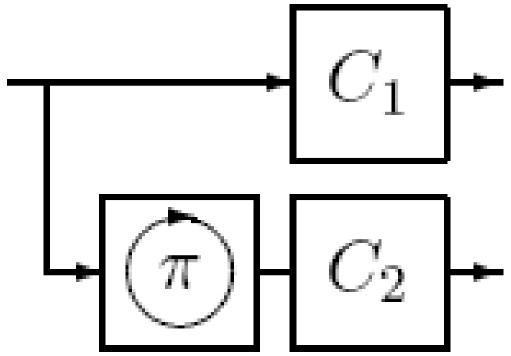
TURBO CODES

- Turbo codes get their name because the decoder uses feedback, like a turbo engine.





TURBO ENCODER



The encoder of a turbo code.

Each box C1, C2, contains a convolutional code.

The source bits are reordered using a permutation π before they are fed to C2.

The transmitted codeword is obtained by concatenating or interleaving the

outputs of the two convolutional codes.

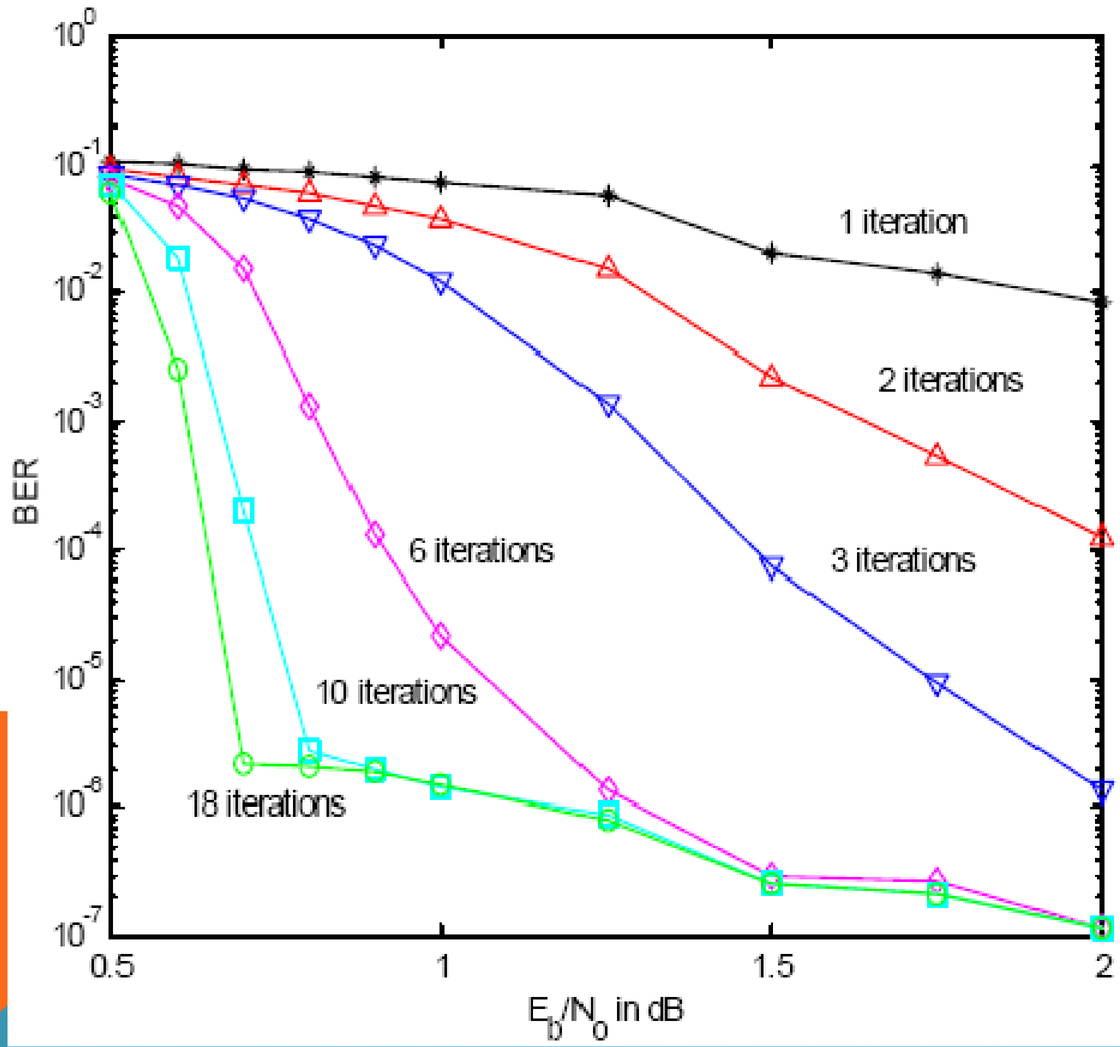
The random permutation is chosen when the code is designed, and fixed thereafter.

TURBO: MAP DECODING

- The goal of the maximum a posteriori (MAP) decoder is to determine P(u(t)=1 | **y**) and P(u(t)=0 | **y**) for each t.
  - The probability of each message bit, given the entire received codeword.
- These two probabilities are conveniently expressed as a log-likelihood ratio:

$$\lambda(t) = \log \frac{P[u(t)=1 \mid \mathbf{y}]}{P[u(t)=0 \mid \mathbf{y}]}$$

Performance as a Function of Number of Iterations

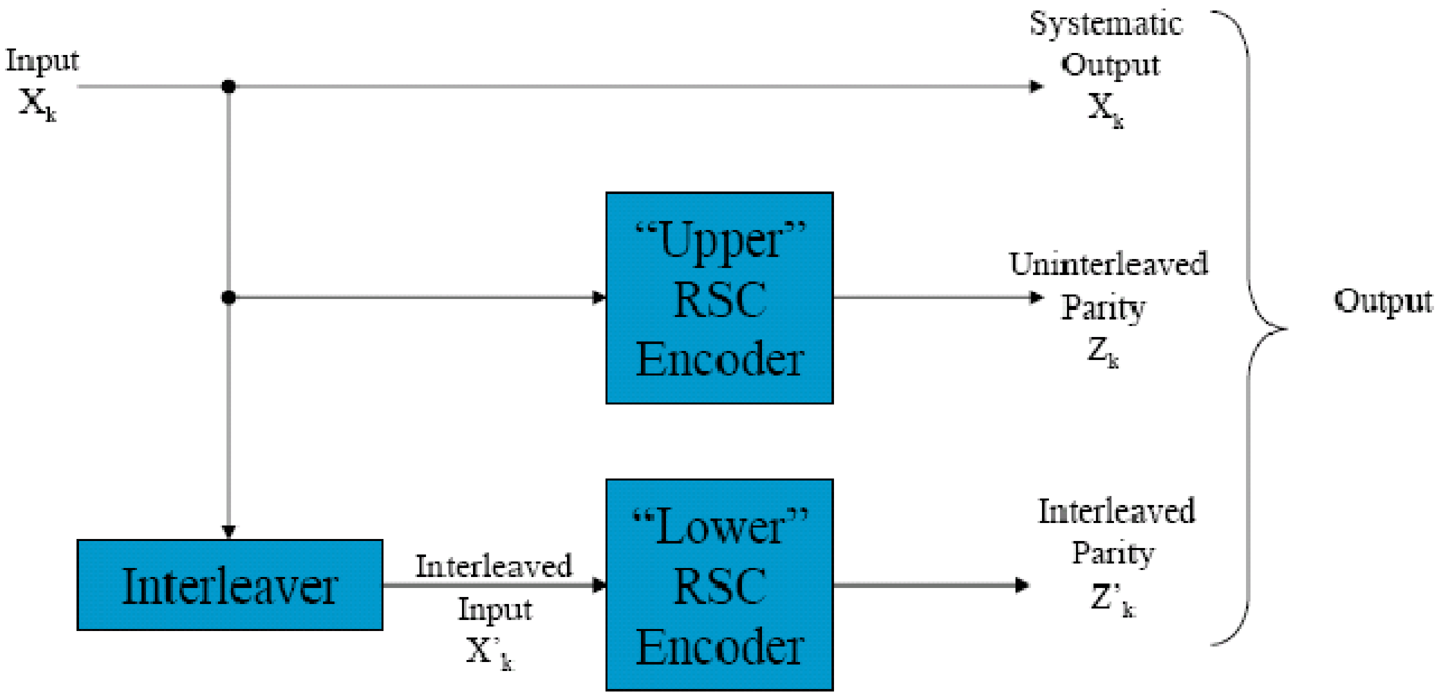


- K = 5
  - constraint length
- r = 1/2
  - code rate
- L= 65,536
  - interleaver size
  - number data bits
- Log-MAP algorithm

TURBO CODES: PERFORMANCE…

- Turbo codes have extraordinary performance at low SNR.
  - Very close to the Shannon limit.
  - Due to a low multiplicity of low weight code words.
- However, turbo codes have a BER “floor”.
  - This is due to their low minimum distance.
- Performance improves for larger block sizes.
  - Larger block sizes mean more latency (delay).
  - However, larger block sizes are not more complex to decode.
  - The BER floor is lower for larger frame/interleaver sizes
- The complexity of a constraint length K_{TC} turbo code is the same as a $K = K_{CC}$ convolutional code, where:
  - $K_{CC} \approx 2+K_{TC}+\log_2$(number decoder iterations)

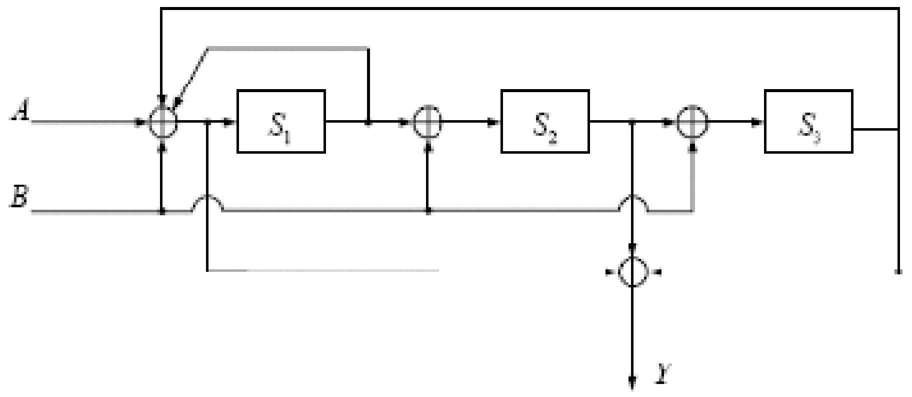
UMTS TURBO ENCODER



- From 3GPP TS 25 212 v6.6.0, Release 6 (2005-09)
  - UMTS Multiplexing and channel coding
- Data is segmented into blocks of L bits.
  - where $40 \leq L \leq 5114$

WIMAX: CONVOLUTIONAL TURBO CODES (CTC)

- The standard specifies an optional convolutional turbo code (CTC) for operation in the 2-11 GHz range.
- Uses same duobinary CRSC encoder as DVB-RCS, though without output W.



- Modulation: BPSK, QPSK, 16-QAM, 64-QAM, 256-QAM.
- Key parameters:
  - Input message size 8 to 256 bytes long.
  - r = {1/2, 2/3, 3/4, 5/6, 7/8}

DIGITAL FOUNTAIN ERASURE CODES

WHAT IS A DIGITAL FOUNTAIN?

A *digital fountain* is an ideal/paradigm for data transmission.

- Vs. the standard (TCP) paradigm: data is an *ordered* finite sequence of bytes.

Instead, with a digital fountain, a *k* symbol file yields an *infinite* data stream ("fountain"); once you have received any *k* symbols from this stream, you can quickly reconstruct the original file.

HOW DO WE BUILD A DIGITAL FOUNTAIN?

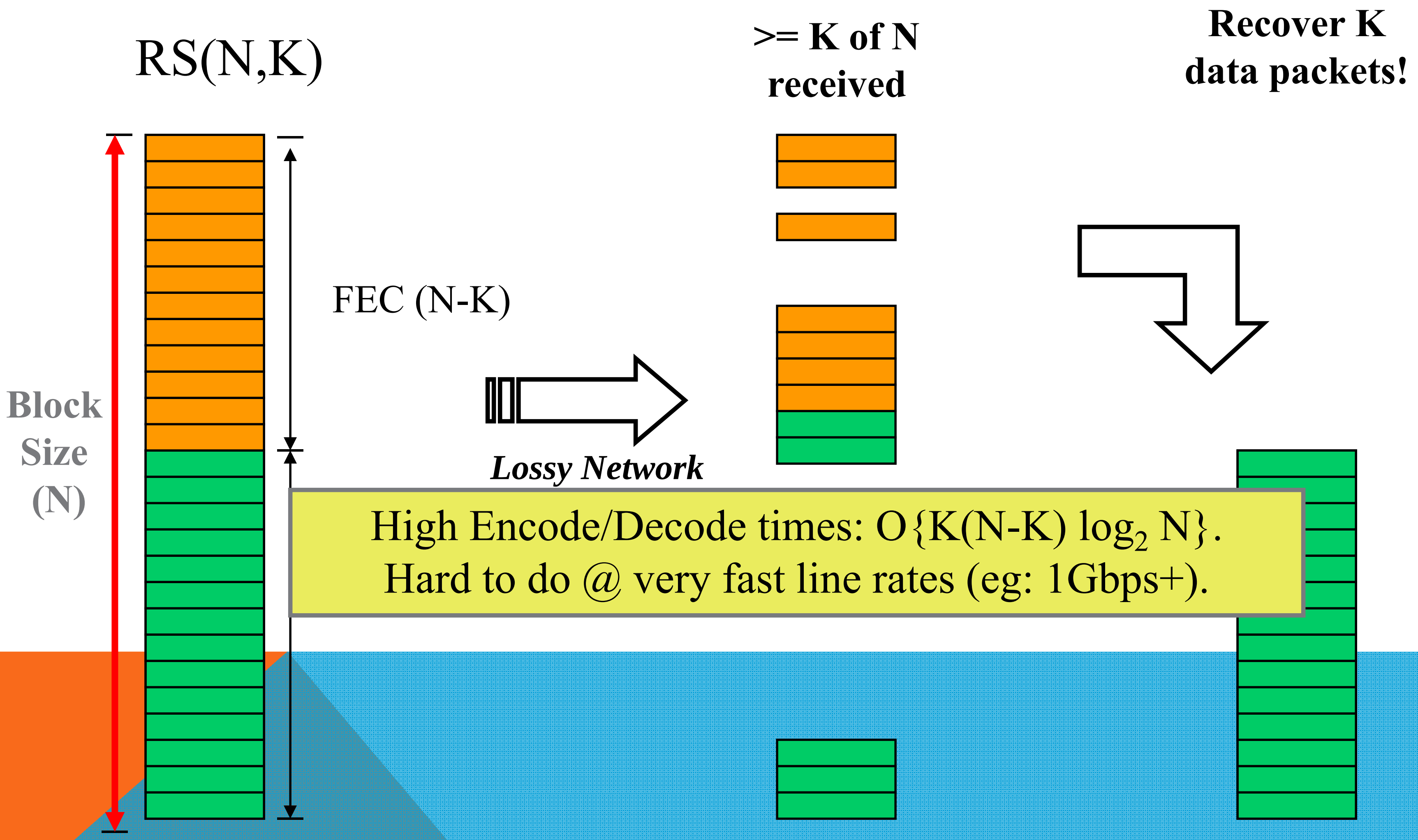
We can construct (approximate) digital fountains using erasure codes.

- Including Reed-Solomon, Tornado, LT, fountain codes.

Generally, we only come close to the ideal of the paradigm.

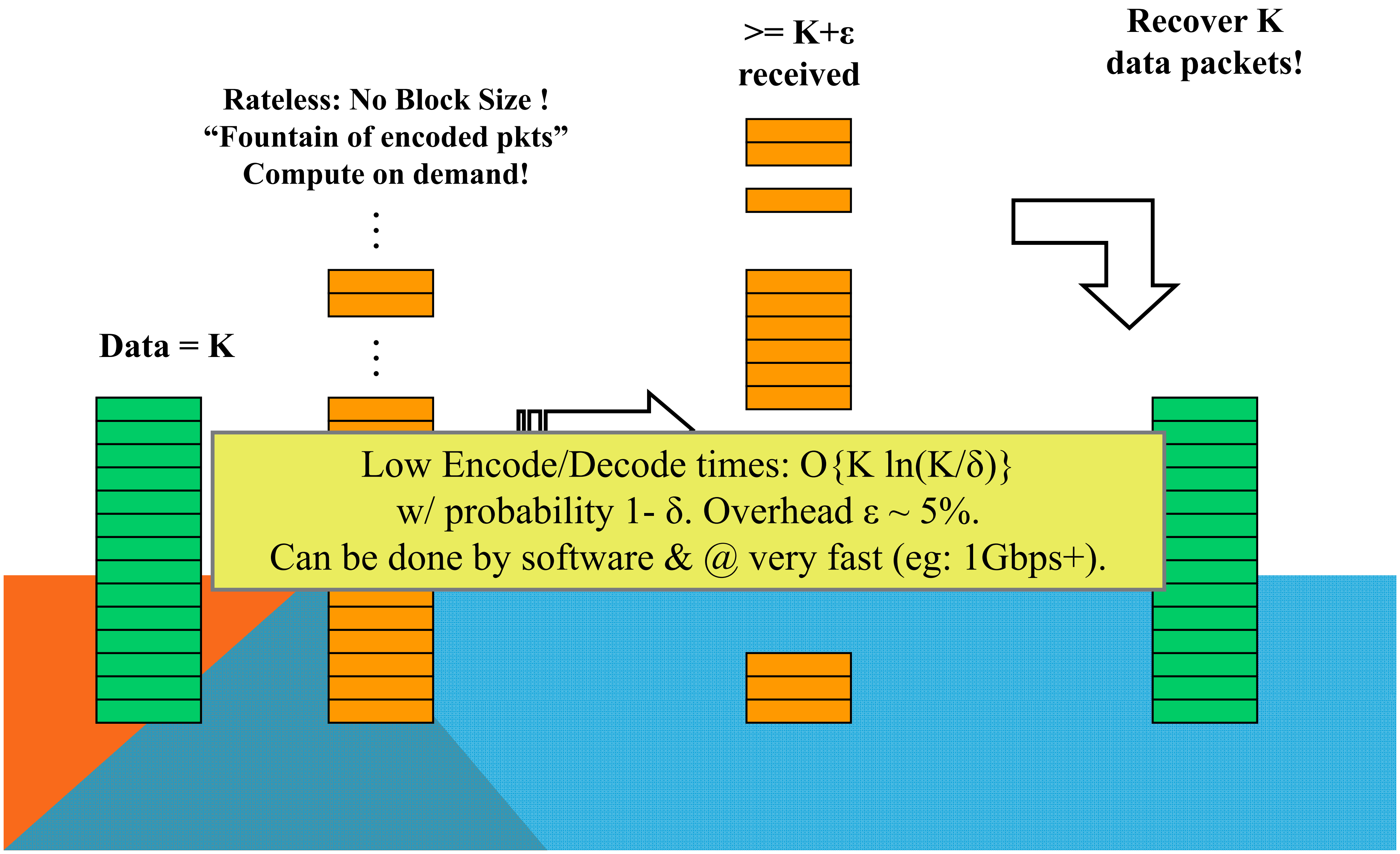
- Streams not truly infinite; encoding or decoding times; coding overhead.

FORWARD ERROR CORRECTION (FEC): EG: REED-SOLOMON RS(N,K)



High Encode/Decode times: O{K(N-K) $\log_2$ N}.
Hard to do @ very fast line rates (eg: 1Gbps+).

DIGITAL FOUNTAIN CODES (EG: RAPTOR CODES)



Low Encode/Decode times: O{K ln(K/δ)}
w/ probability 1- δ. Overhead ε ~ 5%.
Can be done by software & @ very fast (eg: 1Gbps+).

RAPTOR/RATELESS CODES

Properties: *Approximately* MDS

- "Infinite" supply of packets possible.
- Need $k(1+\varepsilon)$ symbols to decode, for some $\varepsilon > 0$.
- Decoding time proportional to $k \ln (1/\varepsilon)$.
- On average, $\ln (1/\varepsilon)$ (constant) time to produce an encoding symbol.

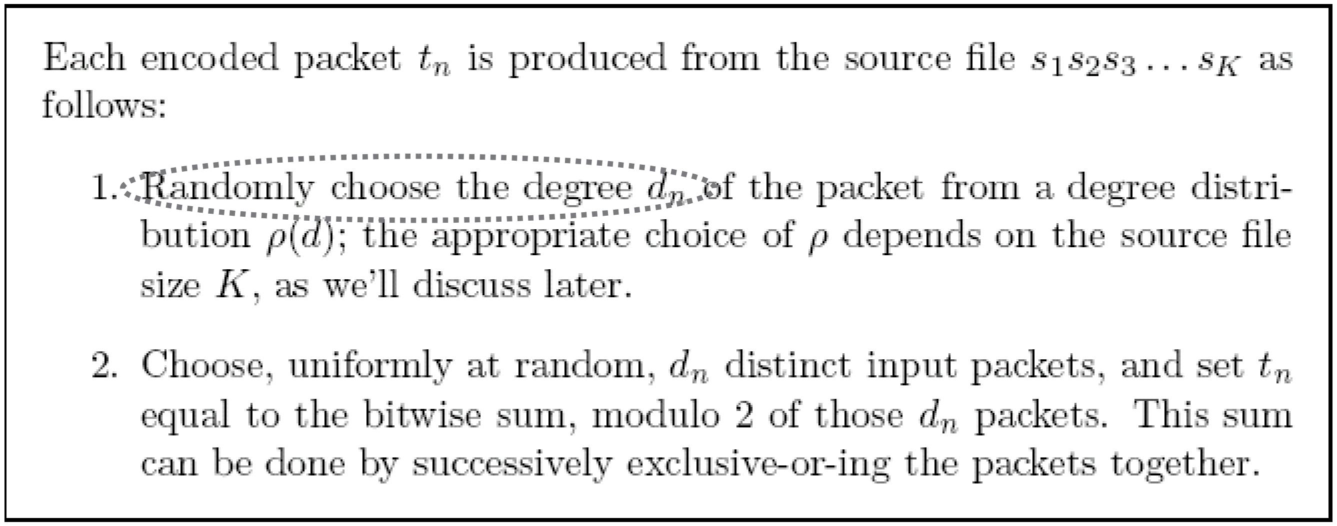
Key: Very fast encode/decode time compared to RS codes

- Compute new check packets on demand!

Bottomline: these codes can be made very efficient and deliver on the promise of the digital fountain paradigm.

DIGITAL FOUNTAIN ENCODER/DECODER

Encoder:

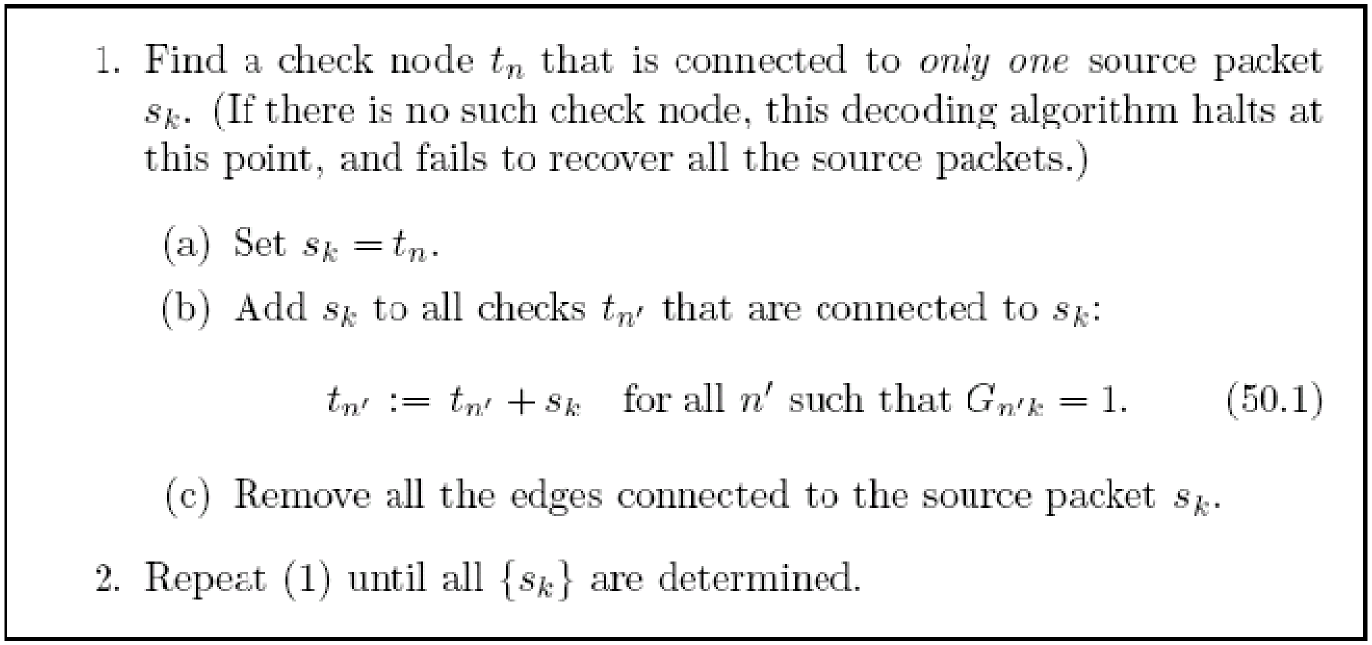


Each encoded packet t_n is produced from the source file $s_1 s_2 s_3 \ldots s_K$ as follows:

1. Randomly choose the degree d_n of the packet from a degree distribution $\rho(d)$; the appropriate choice of ρ depends on the source file size K, as we'll discuss later.

2. Choose, uniformly at random, d_n distinct input packets, and set t_n equal to the bitwise sum, modulo 2 of those d_n packets. This sum can be done by successively exclusive-or-ing the packets together.

- Decoder:



1. Find a check node t_n that is connected to *only one* source packet s_k. (If there is no such check node, this decoding algorithm halts at this point, and fails to recover all the source packets.)

   (a) Set $s_k = t_n$.

   (b) Add s_k to all checks $t_{n'}$ that are connected to s_k:

   $$t_{n'} := t_{n'} + s_k \quad \text{for all } n' \text{ such that } G_{n'k} = 1. \qquad (50.1)$$

   (c) Remove all the edges connected to the source packet s_k.

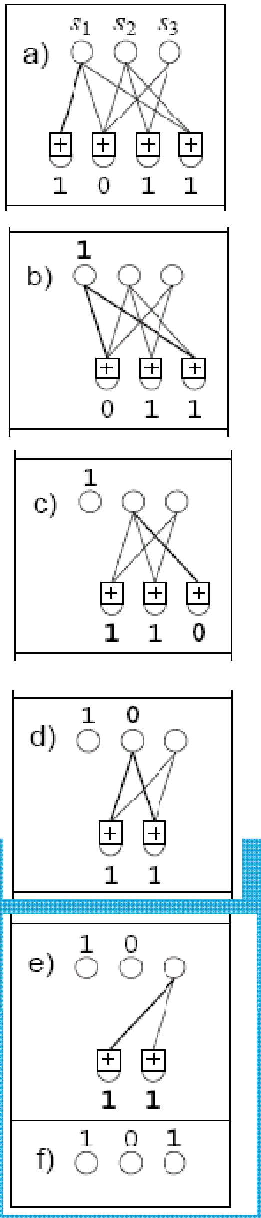
2. Repeat (1) until all $\{s_k\}$ are determined.

DIGITAL FOUNTAIN DECODING (EXAMPLE)

Received bits: 1011

- t_1 is of degree 1, $s_1 = t_1 = 1$
- t_2 & t_3 XOR'ed w/ $s_1 = 1$. Remove s_1's edges
- s_2 set to $t_4 = 0$ {degree = 1}

Repeat as before; $s_3 = 1$



First such code called "*Tornado*" code. Later: *LT-codes*; Concatenated version: "*Raptor code*"

ESOTERICS: ROBUST SOLITON DEGREE DISTRIBUTION

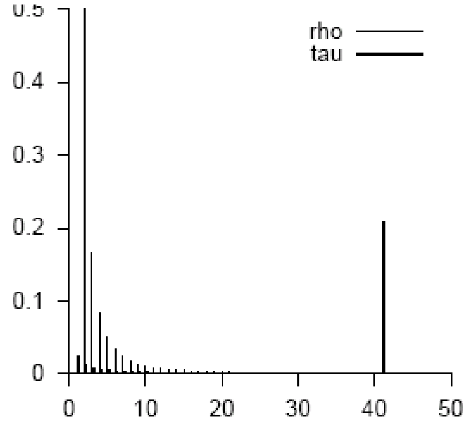


Figure 50.2. The distributions $\rho(d)$ and $\tau(d)$ for the case $K = 10\,000$, $c = 0.2$, $\delta = 0.05$, which gives $S = 244$, $K/S = 41$, and $Z \simeq 1.3$. The distribution τ is largest at $d = 1$ and $d = K/S$.

$$\begin{aligned} \rho(1) &= 1/K \\ \rho(d) &= \frac{1}{d(d-1)} \quad \text{for } d = 2, 3, \ldots, K. \end{aligned}$$

$$S \equiv c \ln(K/\delta)\sqrt{K}.$$

$$\tau(d) = \begin{cases} \frac{S}{K}\frac{1}{d} & \text{for } d = 1, 2, \ldots (K/S)-1 \\ \frac{S}{K}\ln(S/\delta) & \text{for } d = K/S \\ 0 & \text{for } d > K/S \end{cases}$$

$$\mu(d) = \frac{\rho(d) + \tau(d)}{Z},$$

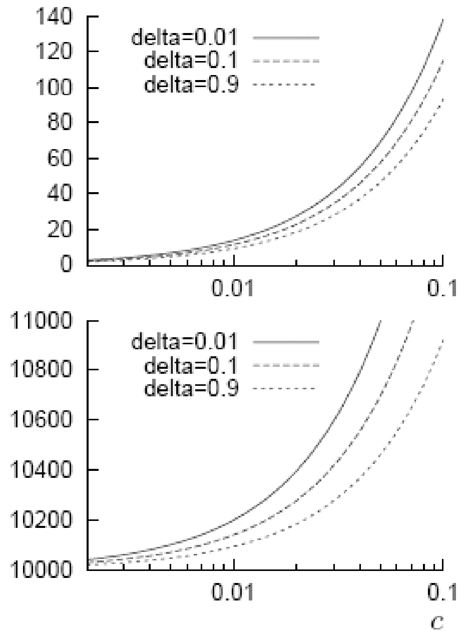


Figure 50.3. The number of degree-one checks S (upper figure) and the quantity K' (lower figure) as a function of the two parameters c and δ, for $K = 10\,000$. Luby's main theorem proves that there exists a value of c such that, given K' received packets, the decoding algorithm will recover the K source packets with probability $1 - \delta$.

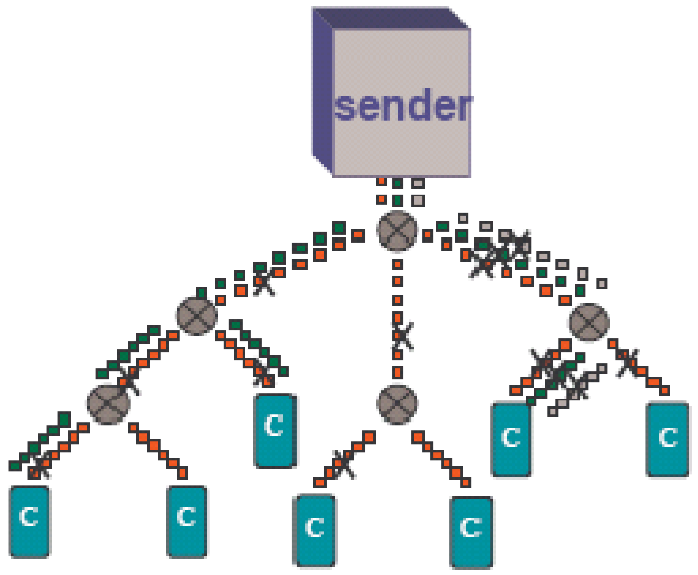
APPLICATIONS: RELIABLE MULTICAST

Many potential problems when multicasting to large audience.

- Feedback explosion of lost packets.
- Start time heterogeneity.
- Loss/bandwidth heterogeneity.

A digital fountain solves these problems.

- Each user gets what they can, and stops when they have enough: doesn't matter which packets they've lost
- Different paths could have diff. loss rates



<u>APPLICATIONS</u>: DOWNLOADING IN PARALLEL

Can collect data from multiple digital fountains for the same source seamlessly.

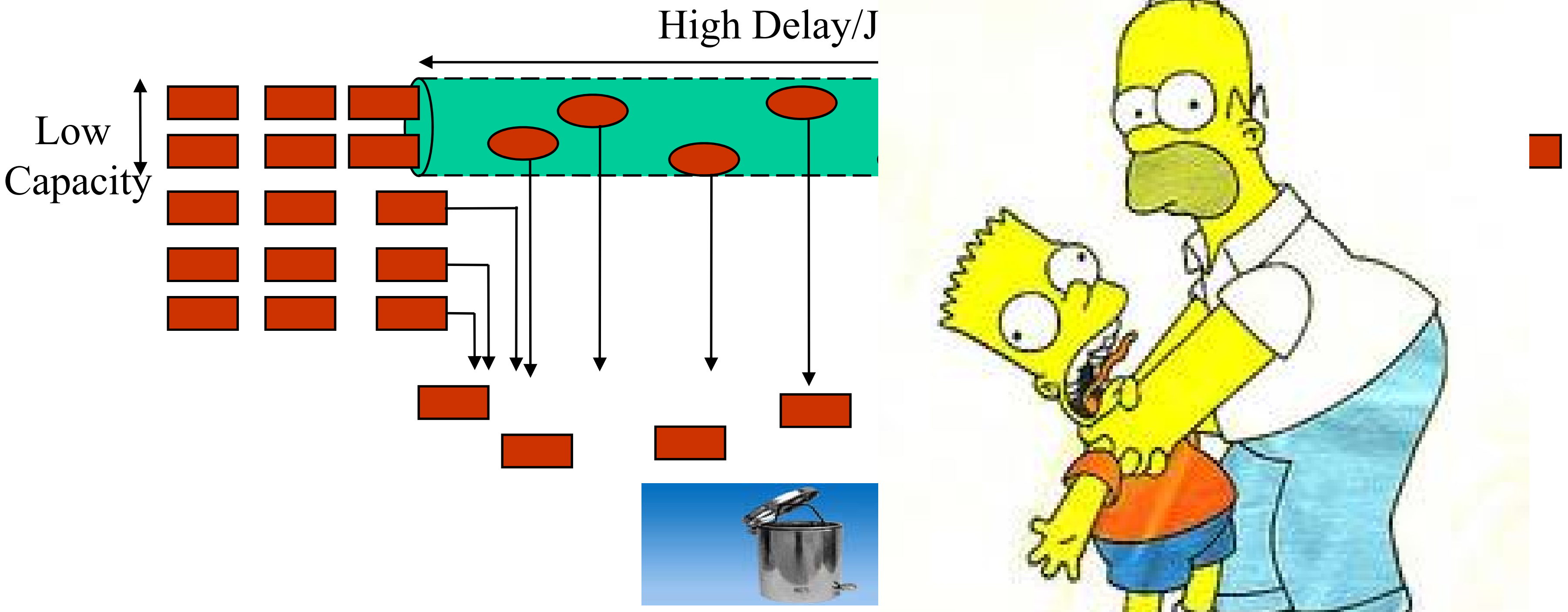
- Since each fountain has an "infinite" collection of packets, no duplicates.
- Relative fountain speeds unimportant; just need to get enough.
- Combined multicast/multi-gather possible.

Can be used for BitTorrent-like applications.

- Microsoft's "Avalanche" product uses randomized linear codes to do "network coding"
- http://research.microsoft.com/~pablo/avalanche.aspx

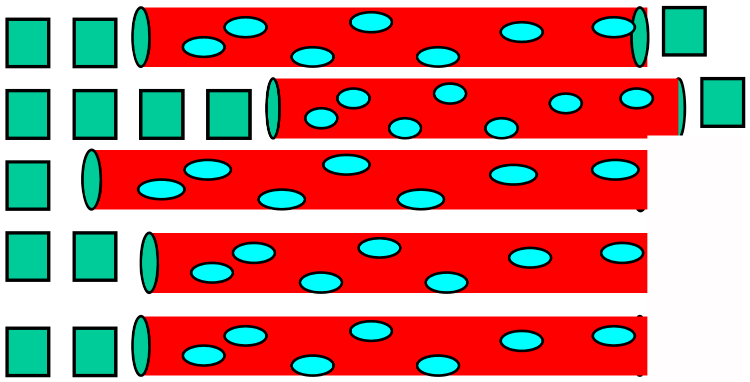
Used to deliver patches to security flaws rapidly; Microsoft Update dissemination etc

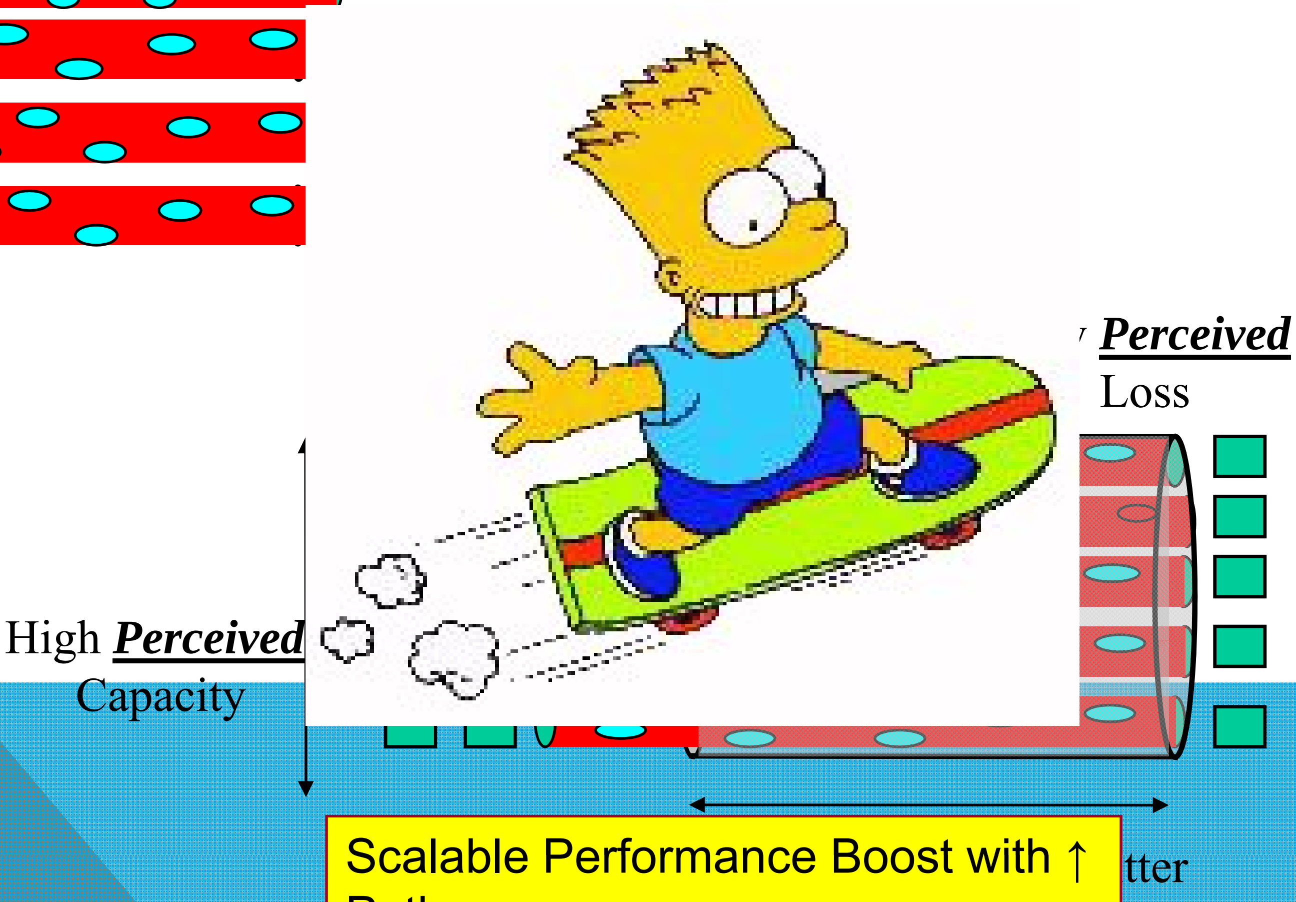
Single path: limited *capacity, delay, loss…*



Network paths usually have
- ***low e2e capacity,***
- ***high latencies and***
- ***high/variable loss rates.***

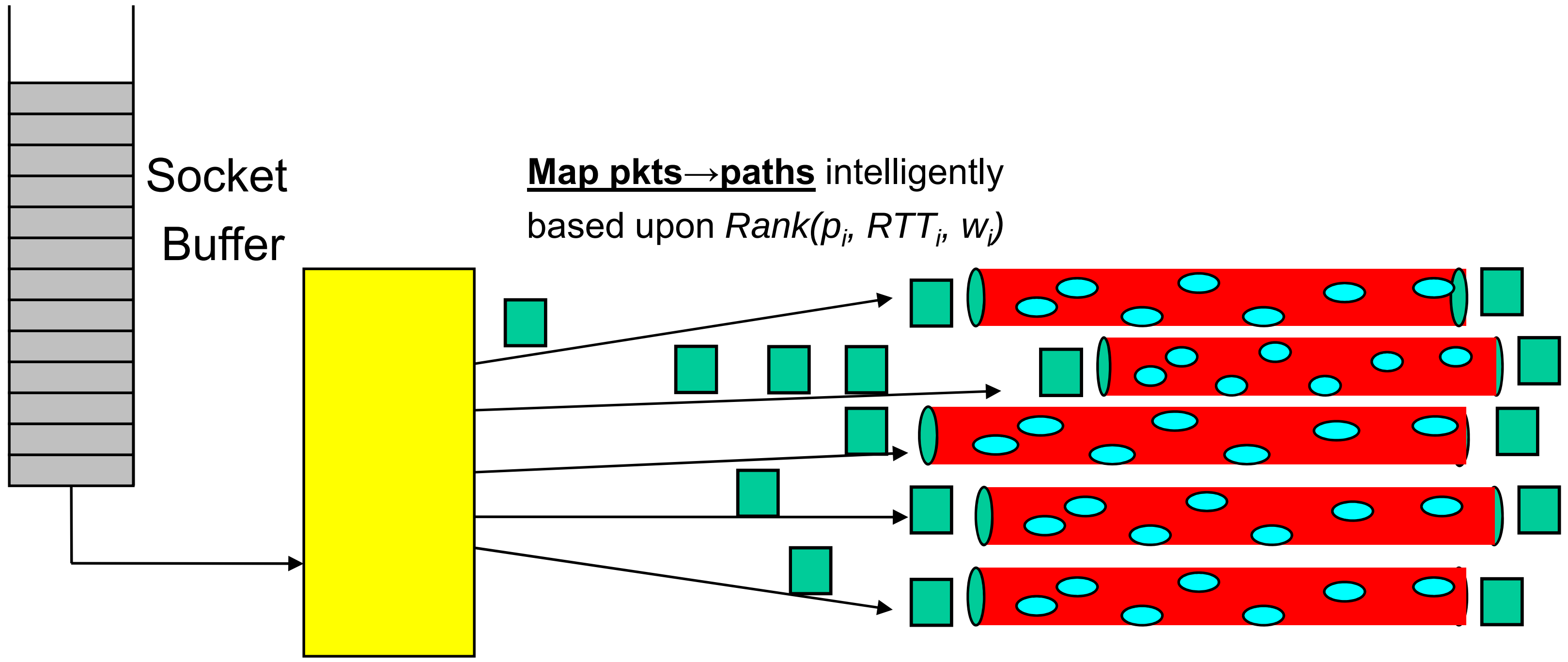
IDEA: AGGREGATE CAPACITY, USE ROUTE DIVERSITY!





Scalable Performance Boost with ↑ Paths

MULTI-PATH LT-TCP (ML-TCP): STRUCTURE


Socket
Buffer
Map pkts→paths intelligently
based upon *Rank(p$_i$, RTT$_i$, w$_i$)*

Per-path congestion control (like TCP)

Reliability @ aggregate, across paths
(FEC block = weighted sum of windows,
PFEC based upon weighted average loss rate)

*Note: these ideas can be applied to other link-level multi-homing,
Network-level virtual paths, non-TCP transport protocols (including video-streaming)*

SUMMARY

Coding: allows better use of degrees of freedom

- Greater reliability (BER) for a given Eb/No, or
- Coding gain (power gain) for a given BER.
- Eg: @ BER = 10^{-5}:
  - 5.1 dB (Convolutional), 7.1dB (concatenated RS/Convolutional)
  - Near (0.1-1dB from) Shannon limit (LDPC, Turbo Codes)
- Magic achieved through iterative decoding (belief propagation) in both LDPC/Turbo codes
  - Concatenation, interleaving used in turbo codes
- Digital fountain erasure codes use randomized LDPC constructions as well.

Coding can be combined with modulation adaptively in response to SNR feedback

Coding can also be combined with ARQ to form Hybrid ARQ/FEC

Efficient coding schemes now possible in software/high line rates => they are influencing protocol design at higher layers also:

LT-TCP, ML-TCP, multicast, storage (RAID, CD/DVDs), Bittorrent, Network coding in Avalanche (Microsoft Updates) etc