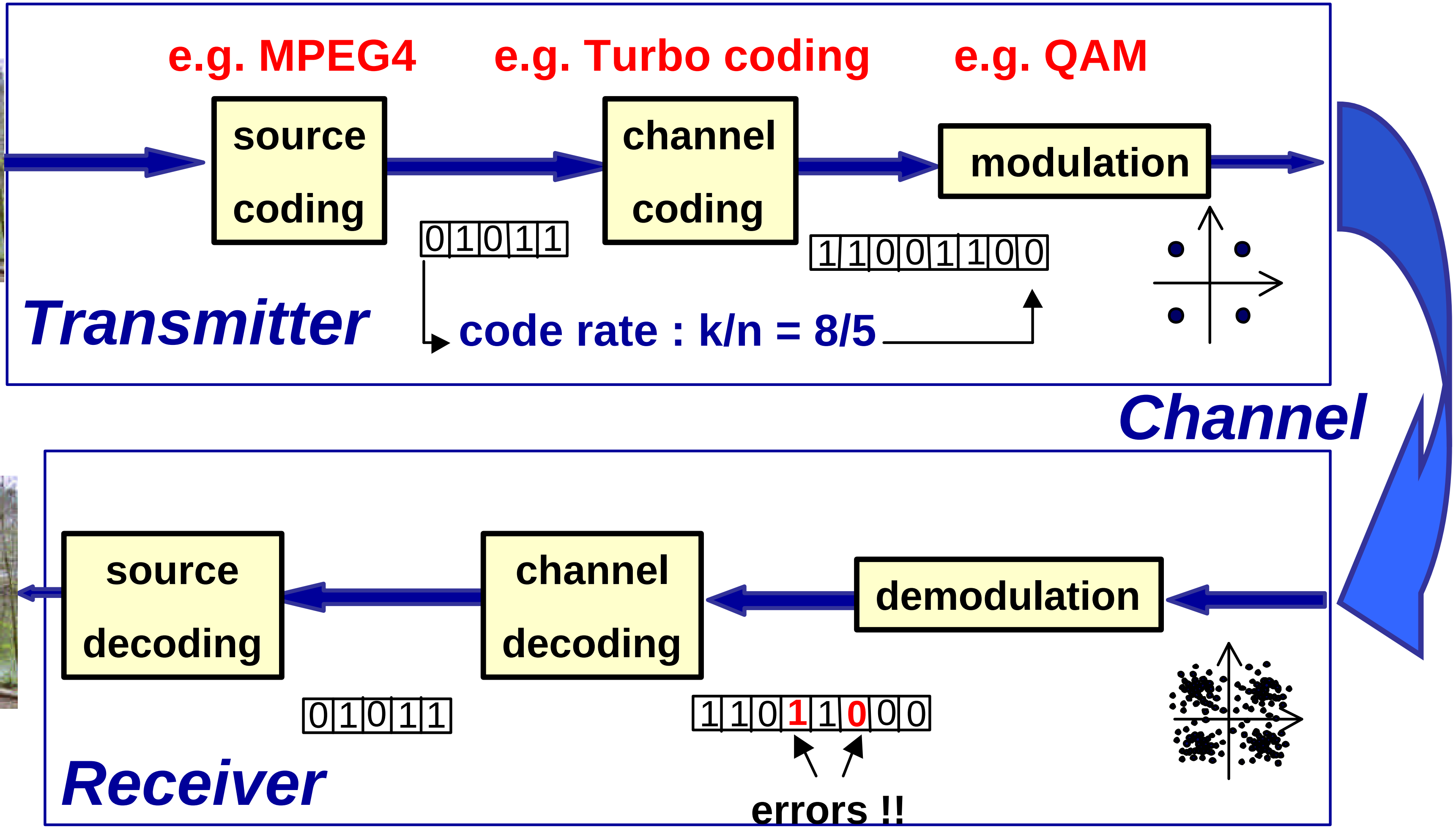
Advanced channel coding : a good basis

Alexandre Giulietti, on behalf of the T@MPO team

Errors in transmission are 'fowardly' corrected using channel coding



Broad, but not exhaustive, channel coding techniques overview

advanced channel coding

block codes
- BCH codes
- Reed-Solomon codes

concatenated codes
- block codes + convolutional codes
- block turbo (product) codes
- convolutional turbo codes (T@MPO)

'applied' codes
- trellis coded modulation
- space-time codes

convolutional codes
- Fano decoding algorithm
- Viterbi decoding algorithm

Block codes were the first

(n,k) linear block code

message bits	parity bits
k bits	n-k bits

coding : **codeword : v = u.G** (u: message)

decoding: **s = r . H^T** (r: received vector; H: parity-check matrix)

syndrome

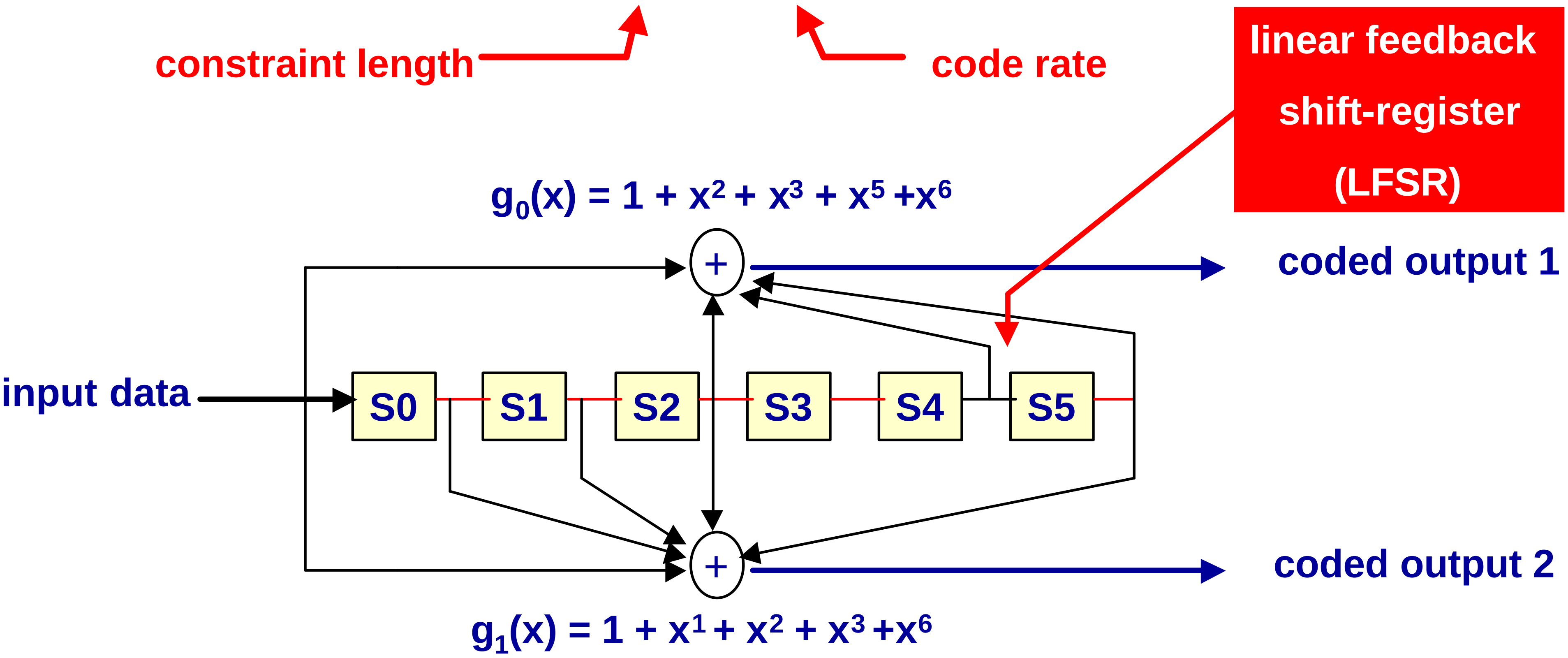
- 0 → no errors, **r = codeword**
- ≠0 → errors, decoding using syndrome

Ex.: (7,4) block code

(0000)	(0000000)
(1000)	(1101000)
(0100)	(0110100)
(1100)	(1011100)
(0010)	(1110010)
(1010)	(0011010)
(0110)	(1000110)
(1110)	(0101110)
(0001)	(1010001)
(1001)	(0111001)
(0101)	(1100101)
(1101)	(0001101)
(0011)	(0100011)
(1011)	(1001011)
(0111)	(0010111)
(1111)	(1111111)

Convolutional coding became famous in the 80's with Viterbi decoding

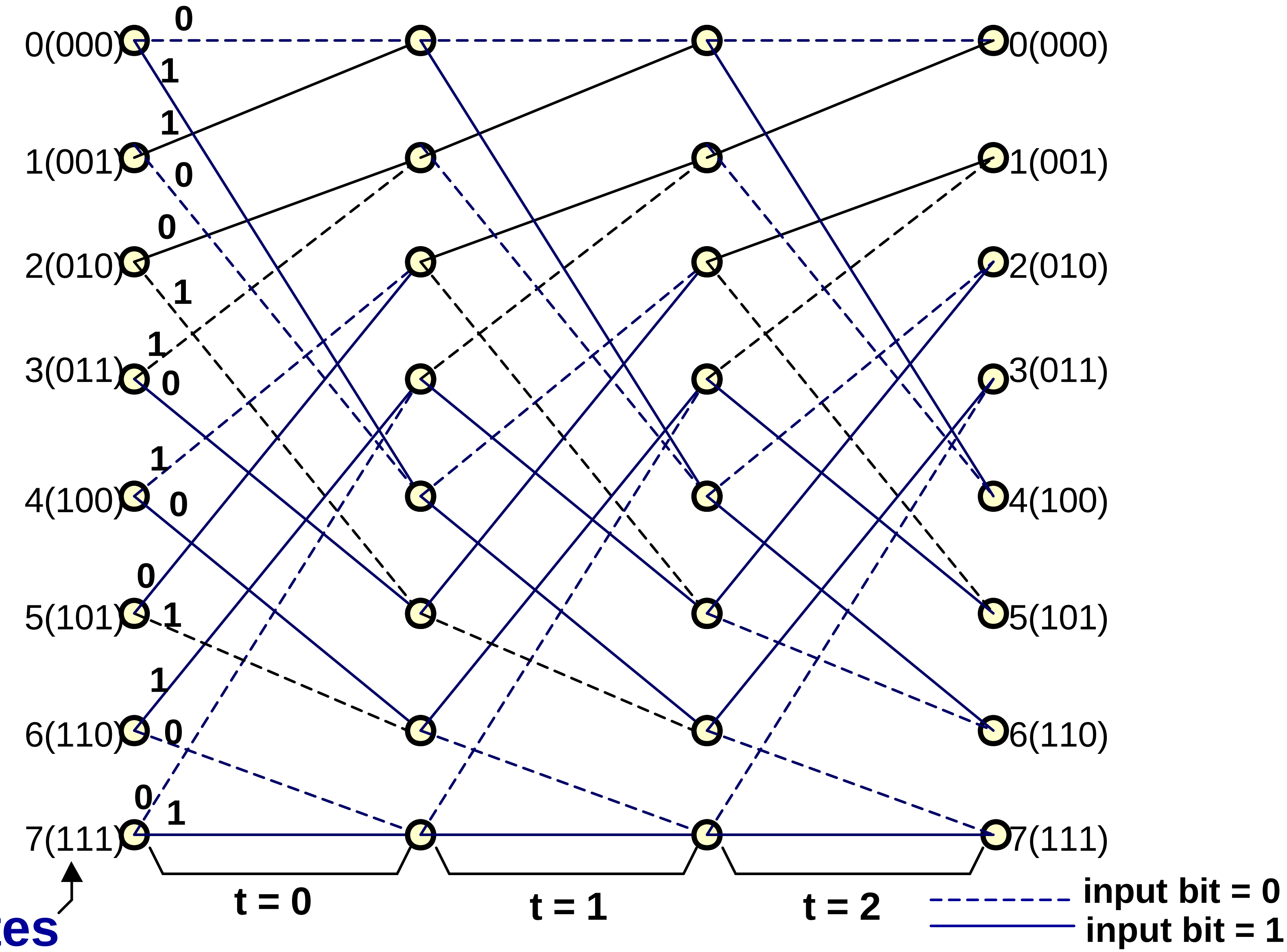
Example : K=7, k/n = 1/2, non-recursive, non-systematic



S0 S1 S2 S3 S4 S5 → 64 encoder states → 000000 , 010100, etc.

Convolutional coding: trellis example

K=4, k/n=1/2



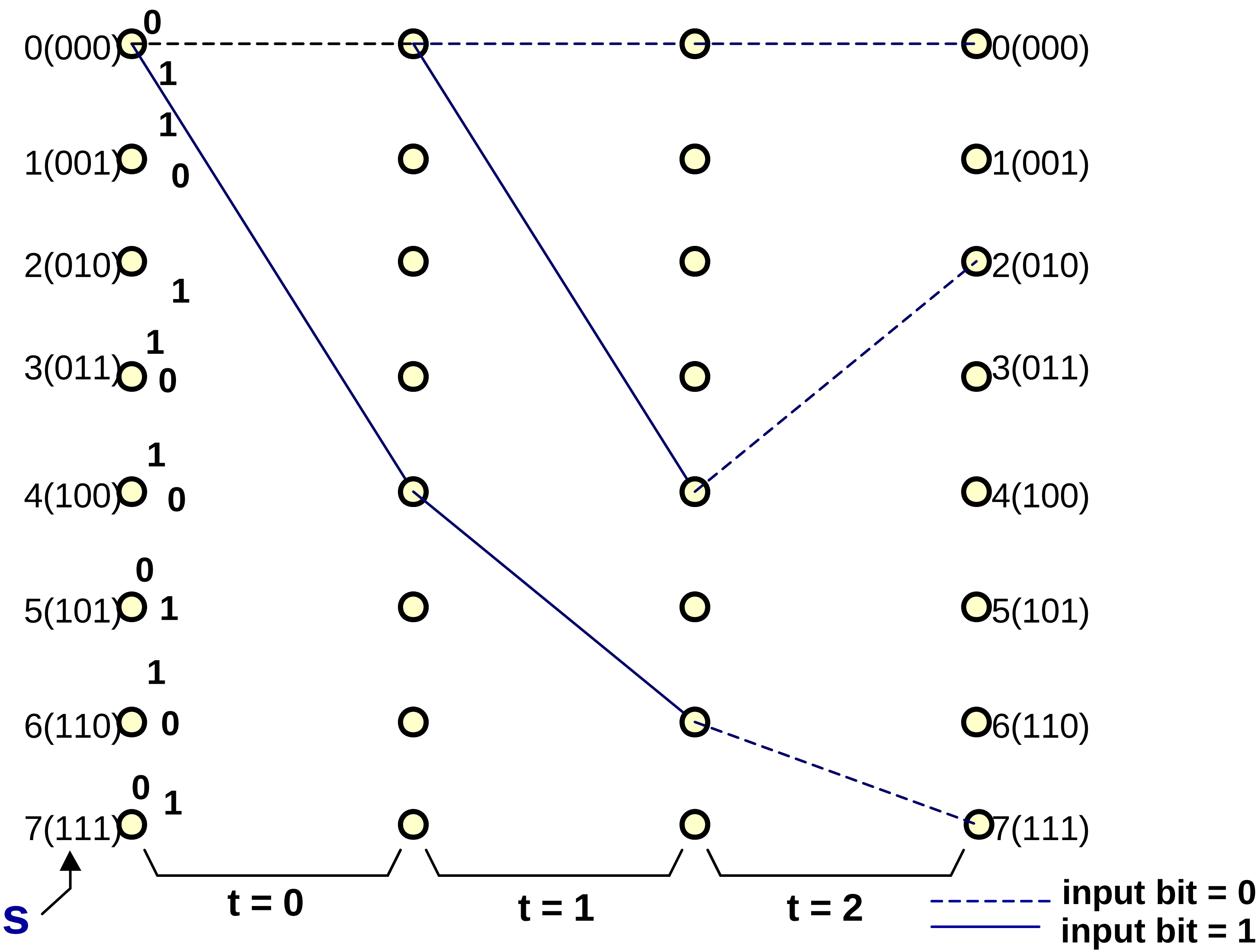
8 states

Convolutional coding: trellis example

input sequence :
010

input sequence :
110

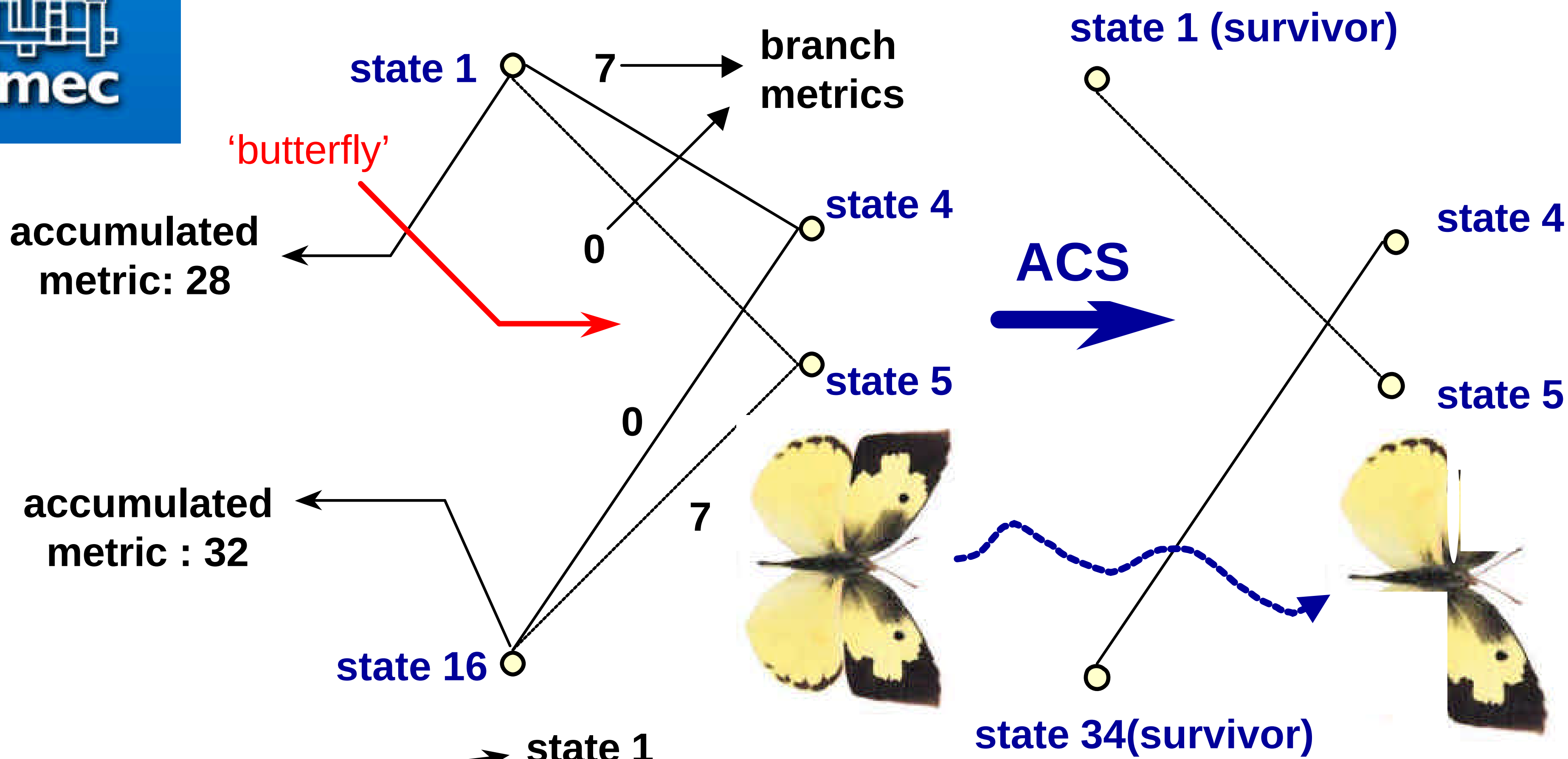
input sequence :
000



8 states

Alex - T@MPO team, DESICS/WISE

Viterbi decoding, ACS operation



ADD :

28 + 7 = 35	('0' branch)	state 1
28 + 0 = 28	('1' branch)	
32 + 0 = 32		
32 + 7 = 39		state 16

COMPARE :

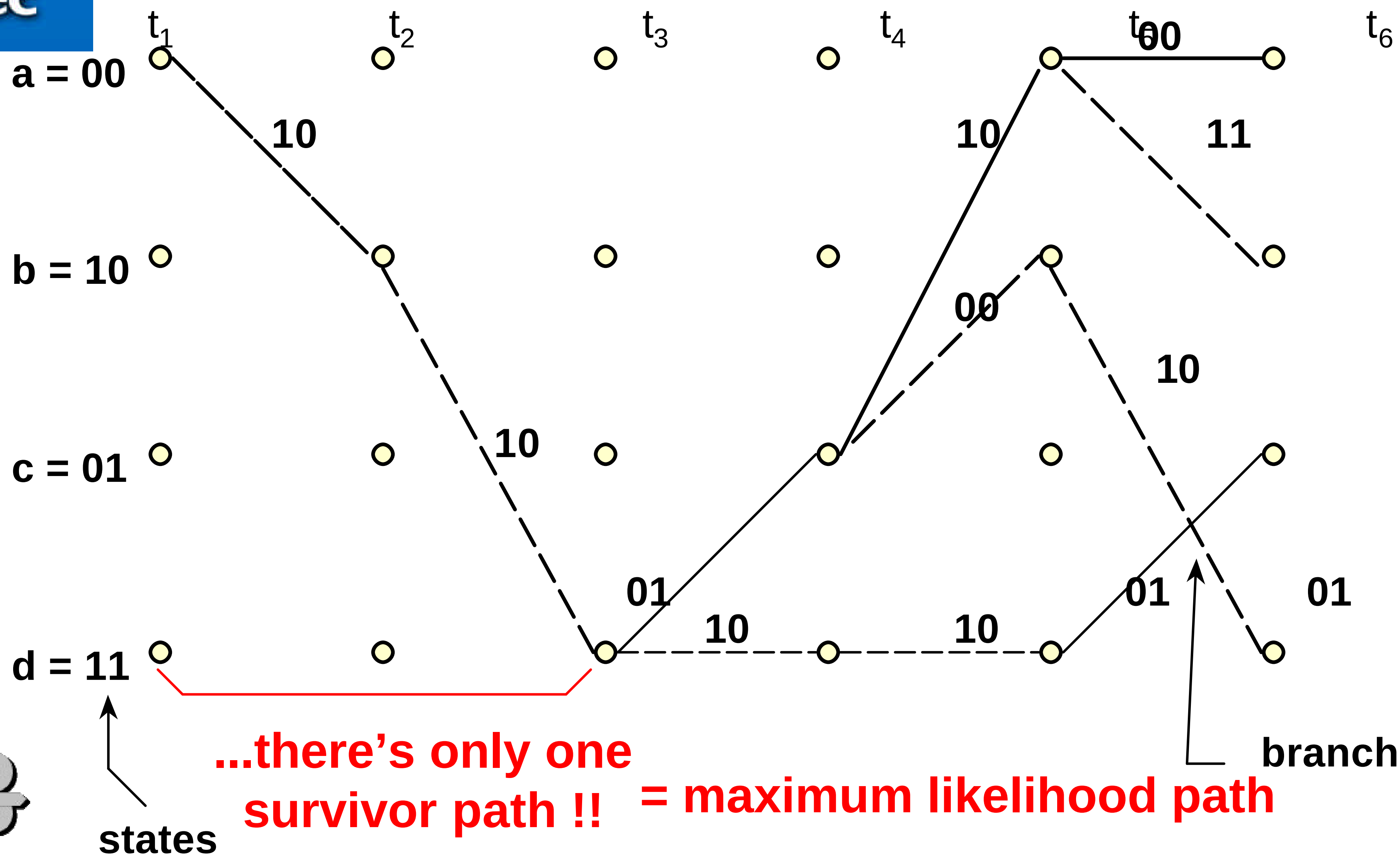
35 > 32 ('0' branch)
28 < 39 ('1' branch)

SELECT :

state 16 ('0' branch)
state 1 ('1' branch)

Viterbi decoding, ML path

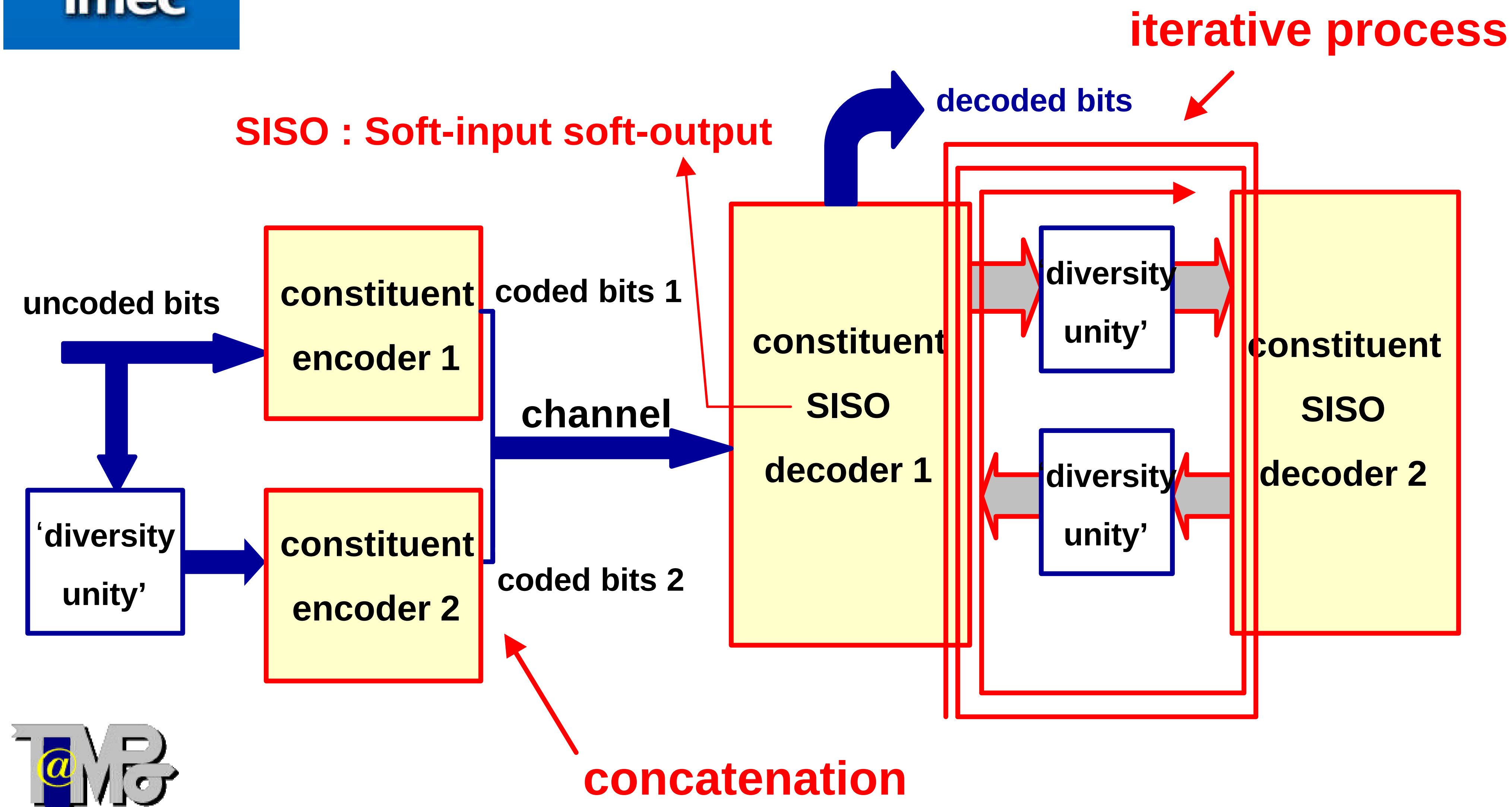
after a few (>4*K) steps ...



...there's only one survivor path !!

= maximum likelihood path

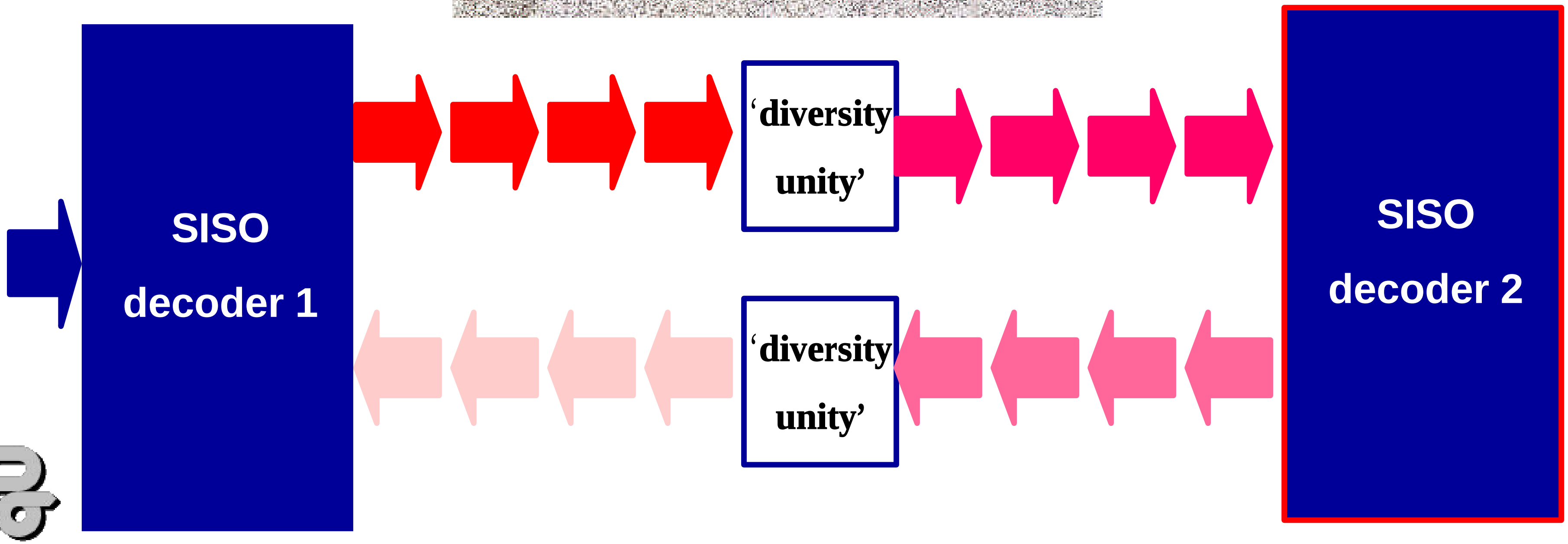
The path to advanced coding : concatenating and decoding iteratively

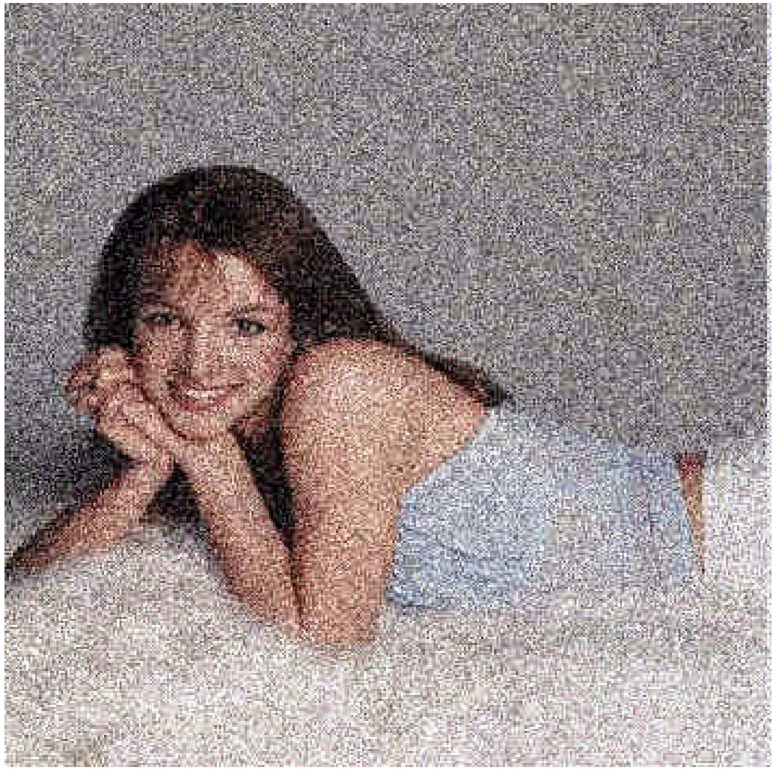


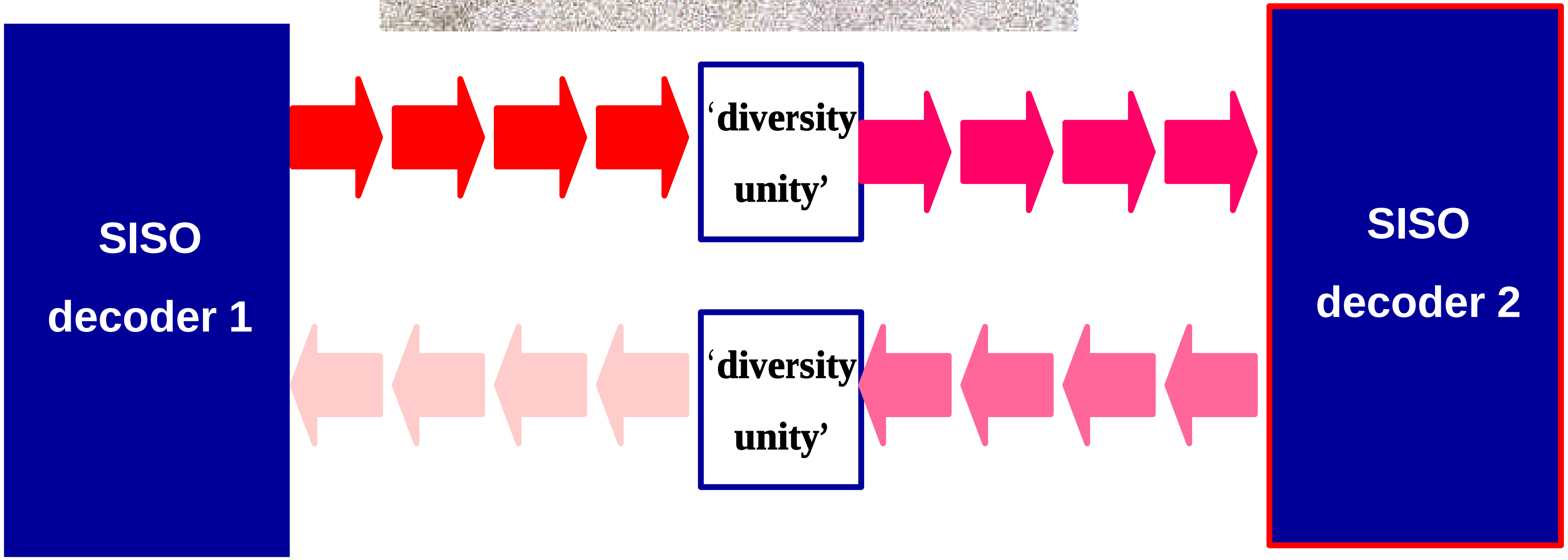
Does iterative decoding + concatenation really allow reliable communication ?

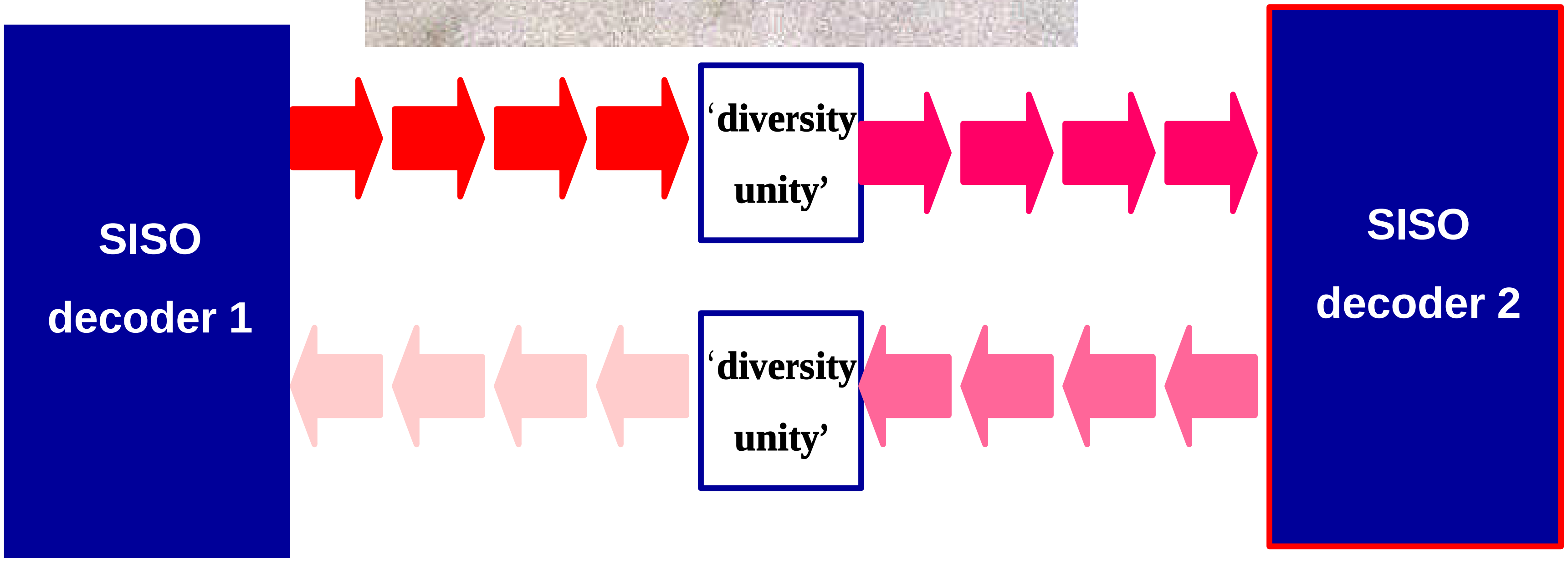
AWGN

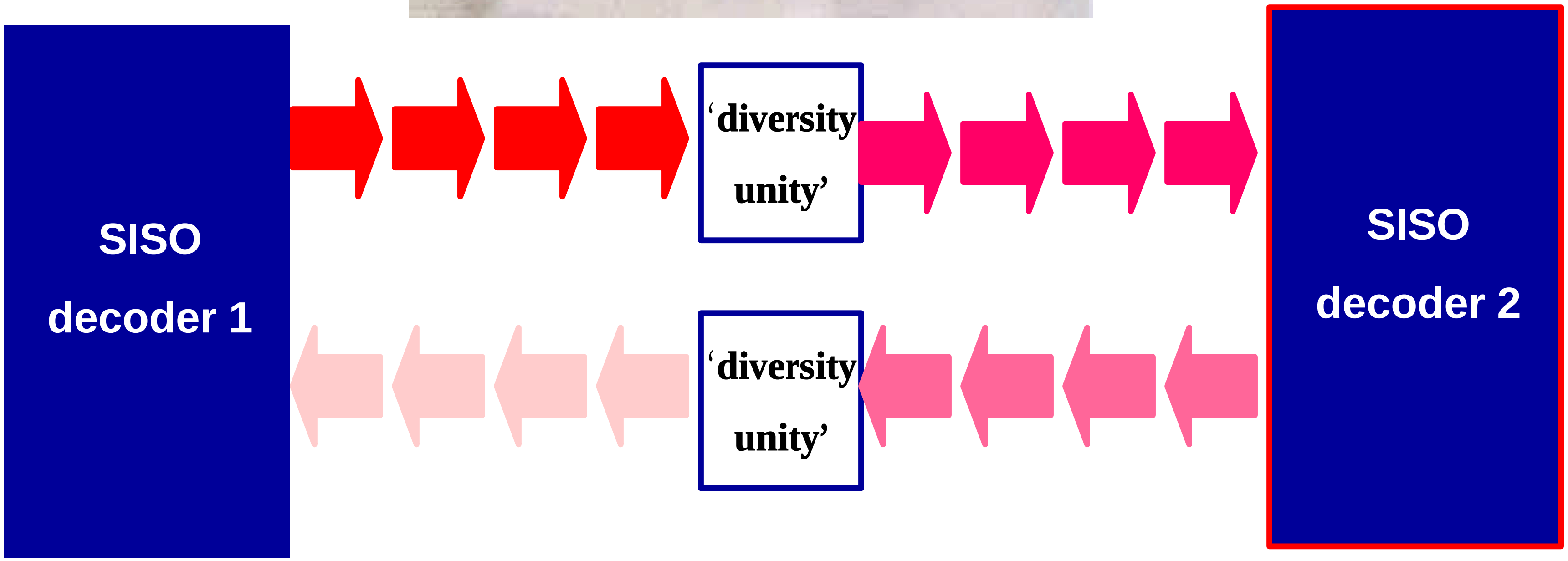
fading


SISO
decoder 1
'diversity
unity'
'diversity
unity'
SISO
decoder 2



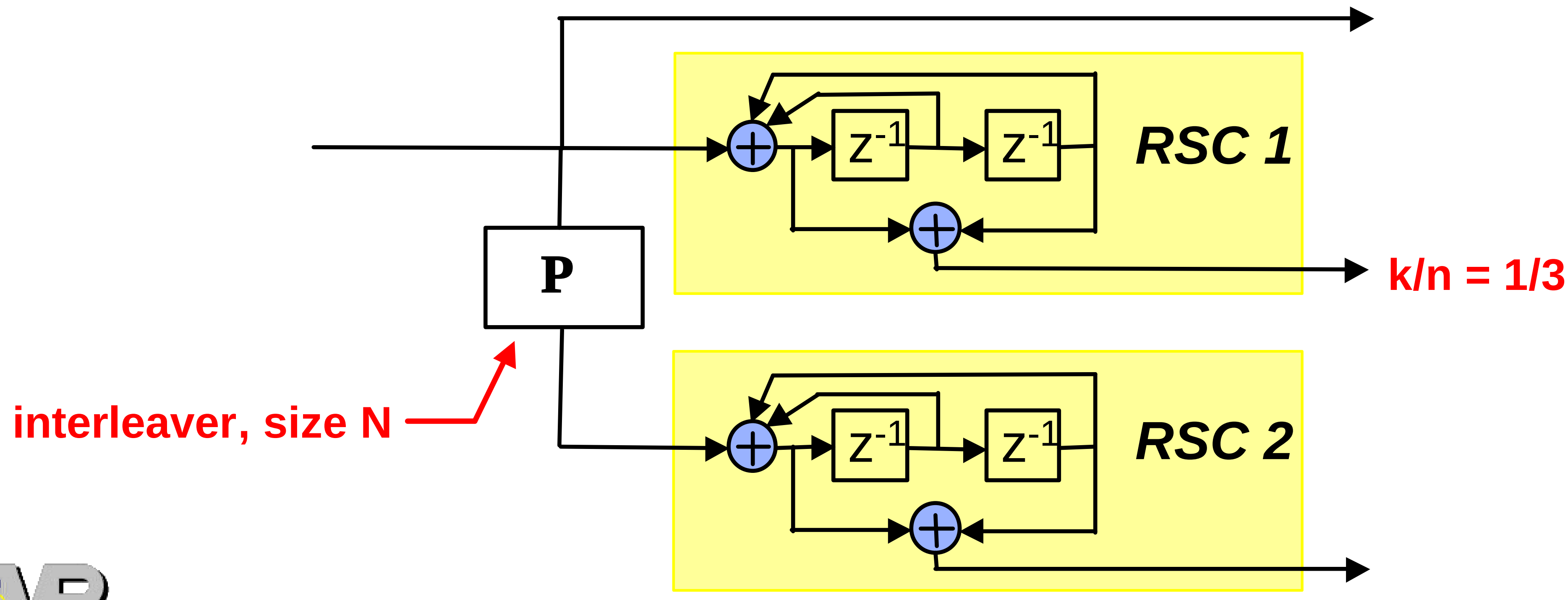

SISO
decoder 1
'diversity
unity'
'diversity
unity'
SISO
decoder 2


SISO
decoder 1
'diversity
unity'
'diversity
unity'
SISO
decoder 2


SISO
decoder 1
'diversity unity'
'diversity unity'
SISO
decoder 2

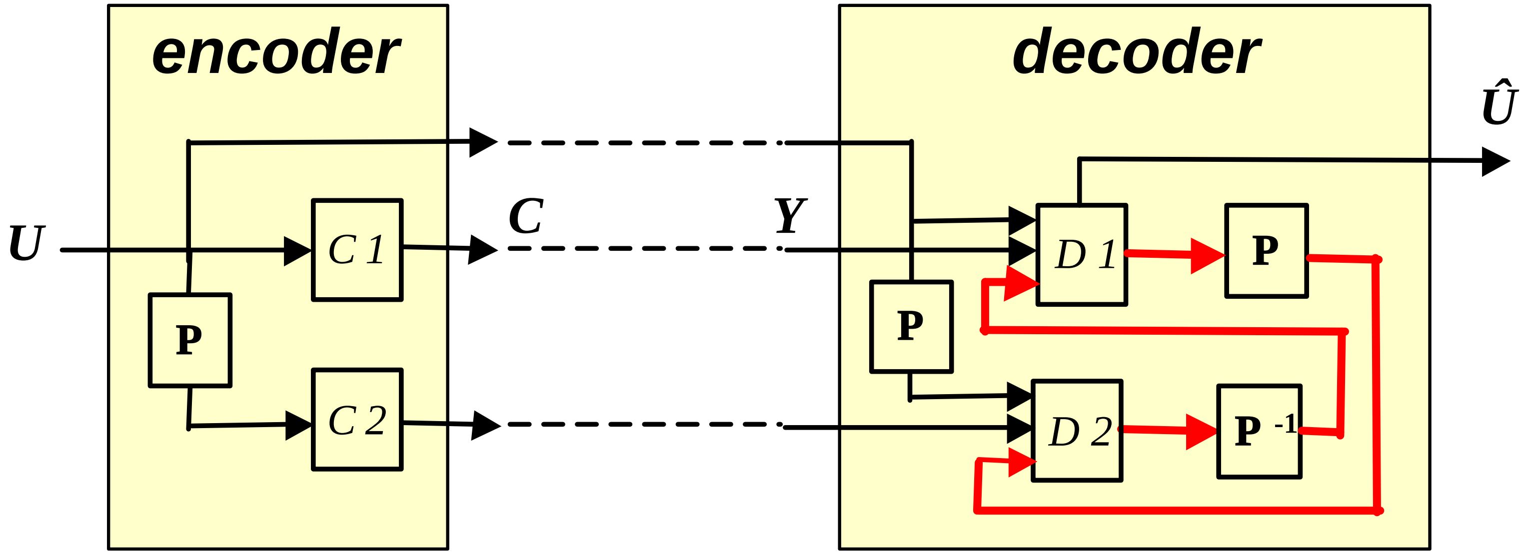
Parallel concatenated convolutional (*turbo*) codes(Glavieux,Berroux et Thitimajshima, 1993)

two recursive systematic convolutional codes RSC 1 and RSC 2 concatenated in parallel !



Turbo codes: coder/decoder architecture

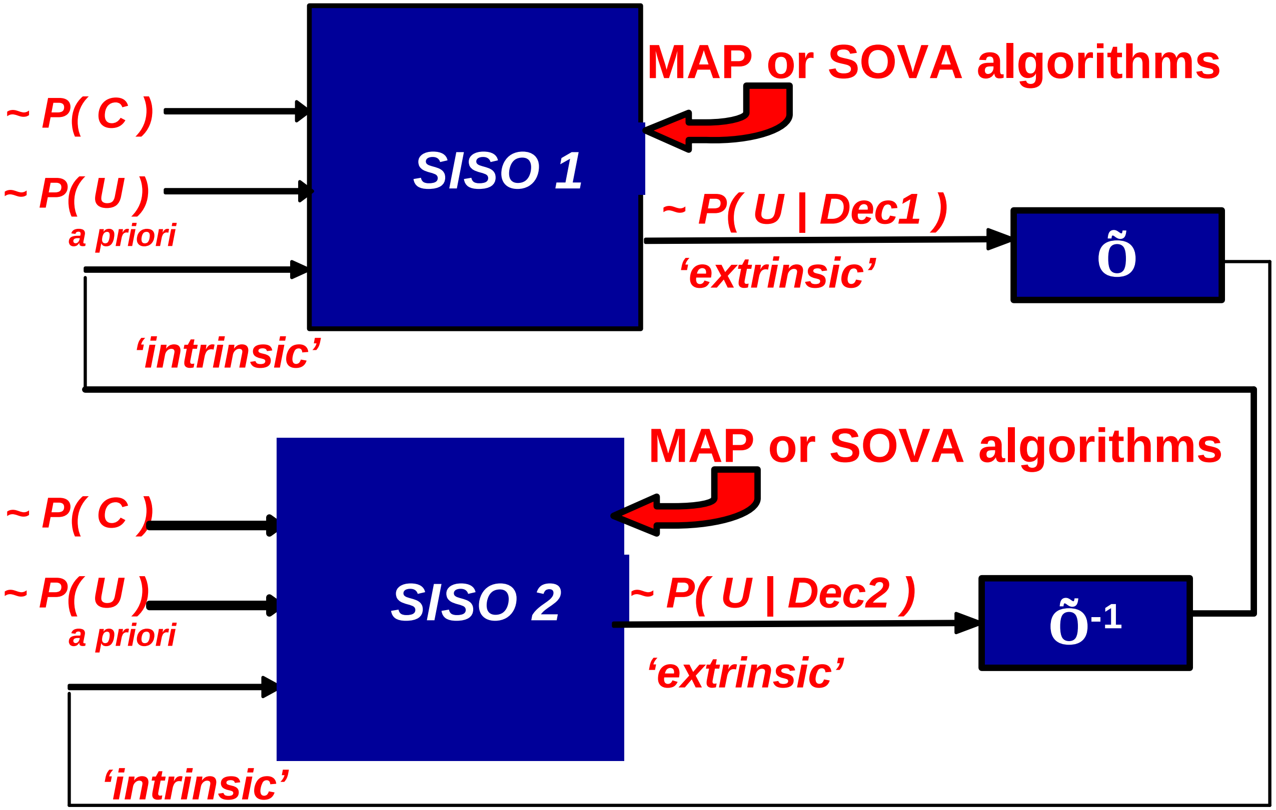
- **iterative decoding:** ***D1 ® D2 ® D1 ® D2 ® D1 ® D2 ®***
- **one decoder module (*D*) for each encoder (*C*)**
- **increasingly good solution:**



Turbo codes : soft-input soft-output (SISO) decoding

log-likelihood

information



amplitude = confiability

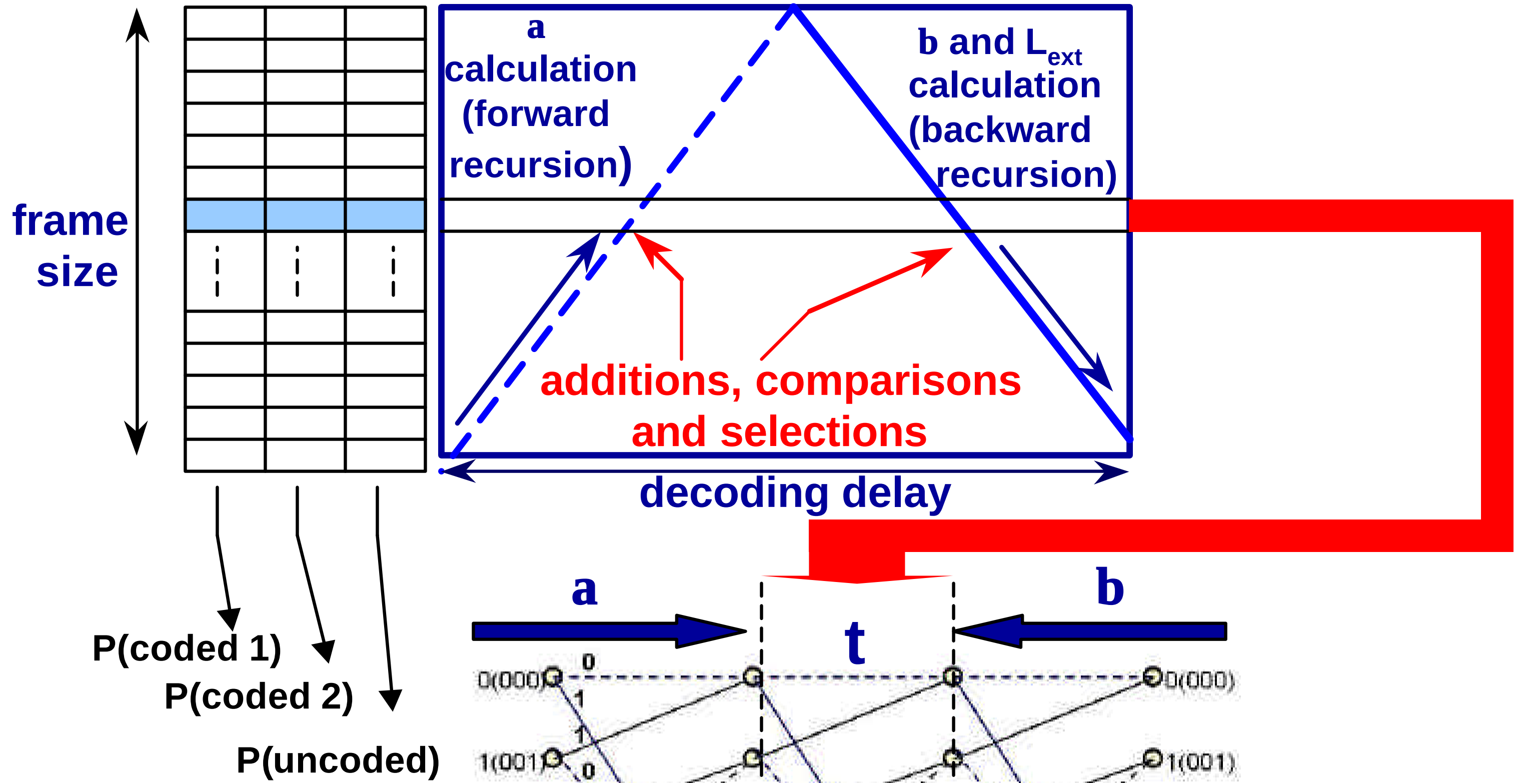
sign = hard-decision

$$L(u) = \log \frac{p(u=1 \mid R)}{p(u=0 \mid R)}$$

at the end ... $L(u) = Lc.y + Le_{dec1}(u) + Le_{dec2}(u)$

SISO = log-likelihood information amplifier!

The MAP algorithm is a hard nut to crack

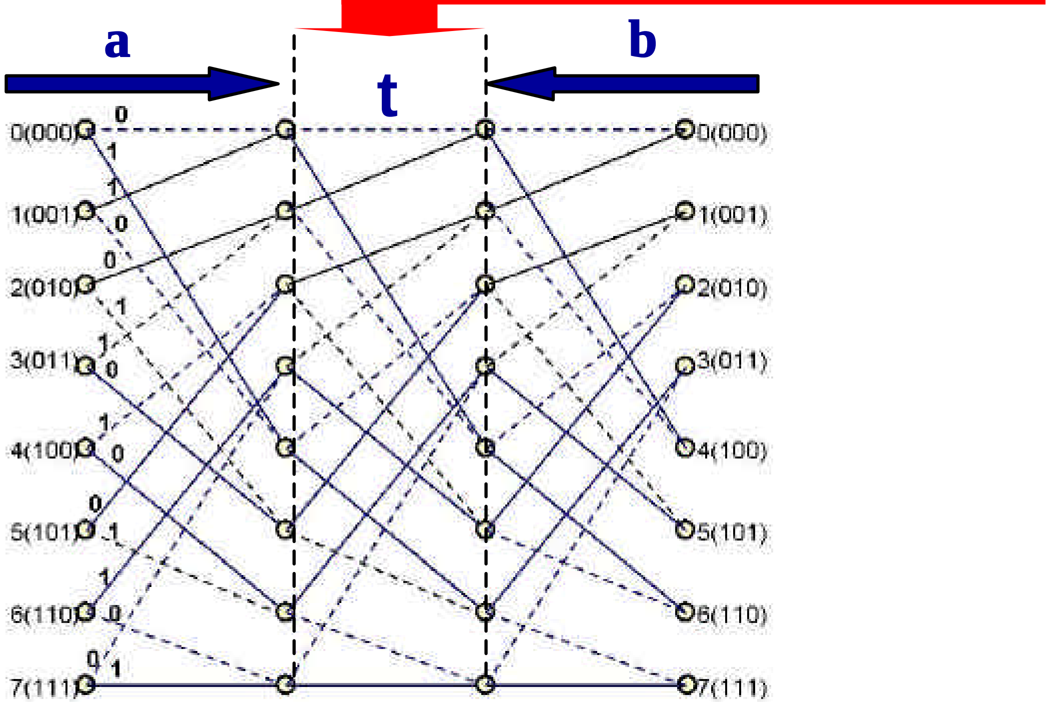


$$a_t = f(a_{t+1})$$

$$b_{t+1} = f(b_t)$$

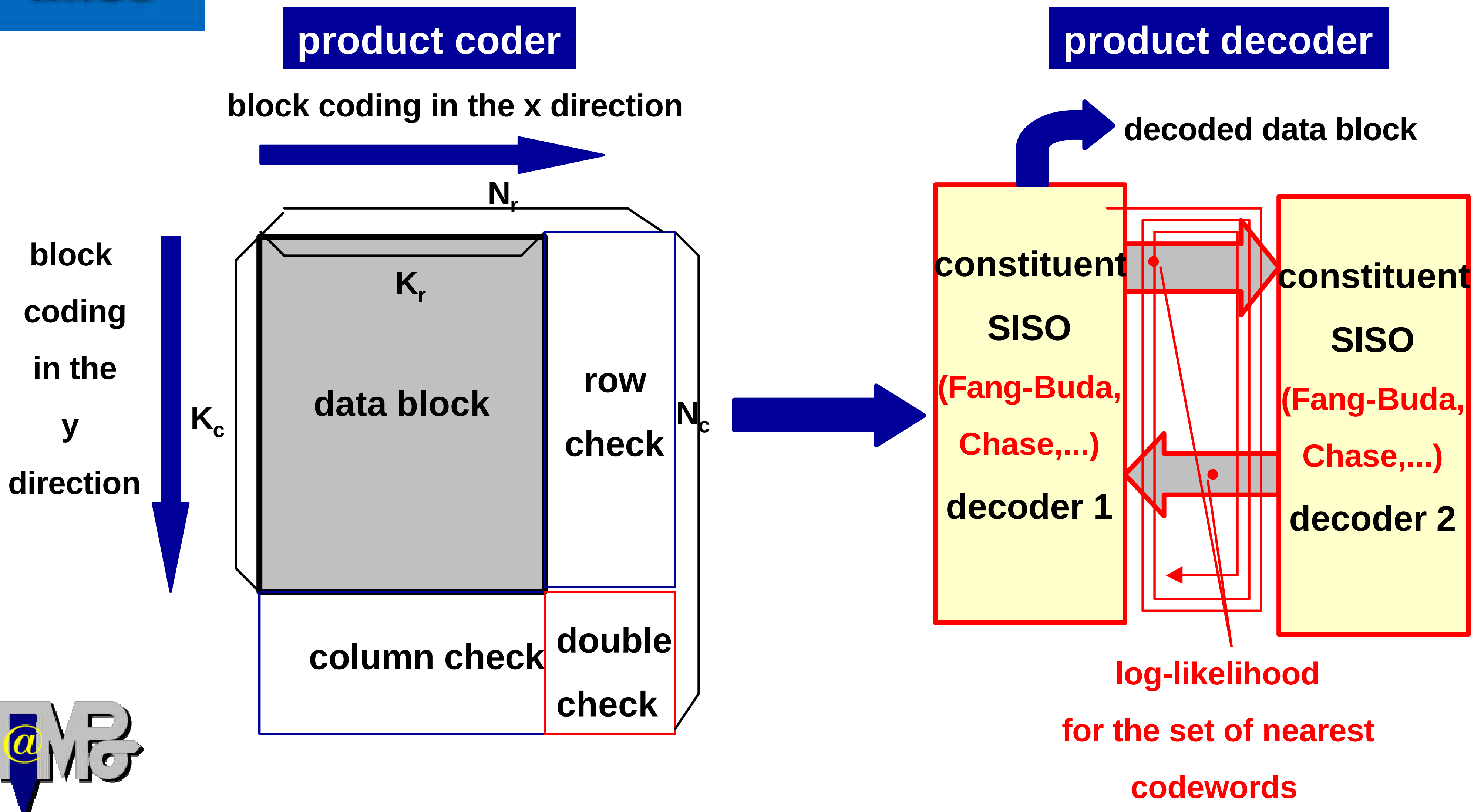
$$g_t = f(P(coded), P(uncoded), extr)$$

$$L(u) = \log \frac{a_1 b_1 g_1}{a_0 b_0 g_0}$$



Alex - T@MPO team, DESICS/WISE

Serially concatenated block (*product*) codes are brothers and rivals of turbo codes



Who corrects more errors ?
Selected algorithms

'product' codes (SCBC)

Fang-Buda
4 iterations
BCH x BCH codes:
(26,20)x(31,20) = 1 ATM cell
(42,29)x(42,29) = 2 ATM cells
k/n = 1/2

X

'turbo' codes (PCCC)

log-max MAP
6 iterations
UMTS codes
UMTS interleaver
1 and 2 ATM cells
k/n = 1/2 (punctured)

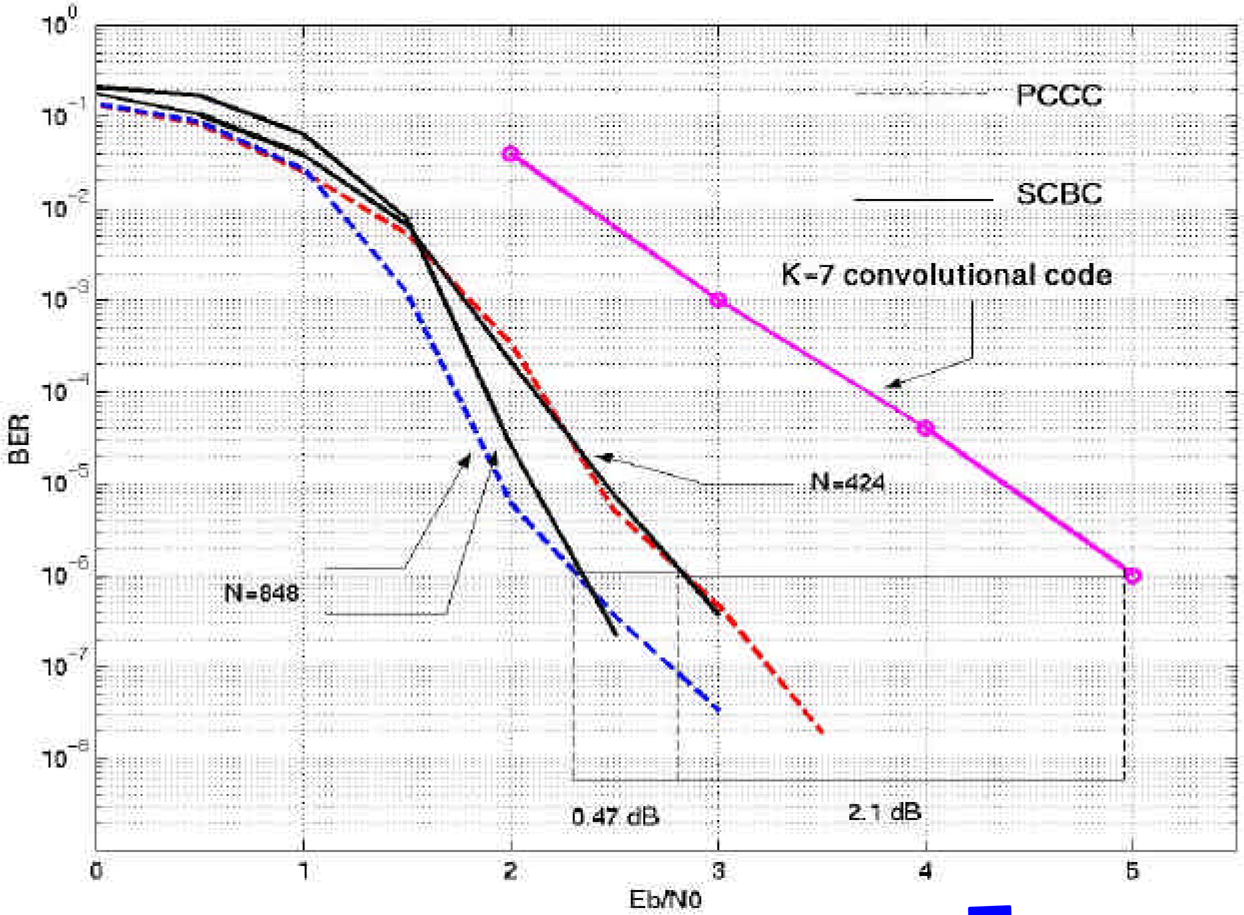
pre-processing

1. LLR sorting
2. H matrix permutation
3. Gaussian reduction on H
4. LLR permutation

systematic search for codewords

1. subset reordering with exclusion
2. coding to get list of codewords
3. reduced list of closest codewords
4. extrinsic calculation

Who corrects more errors ?
BER for QPSK and AWGN

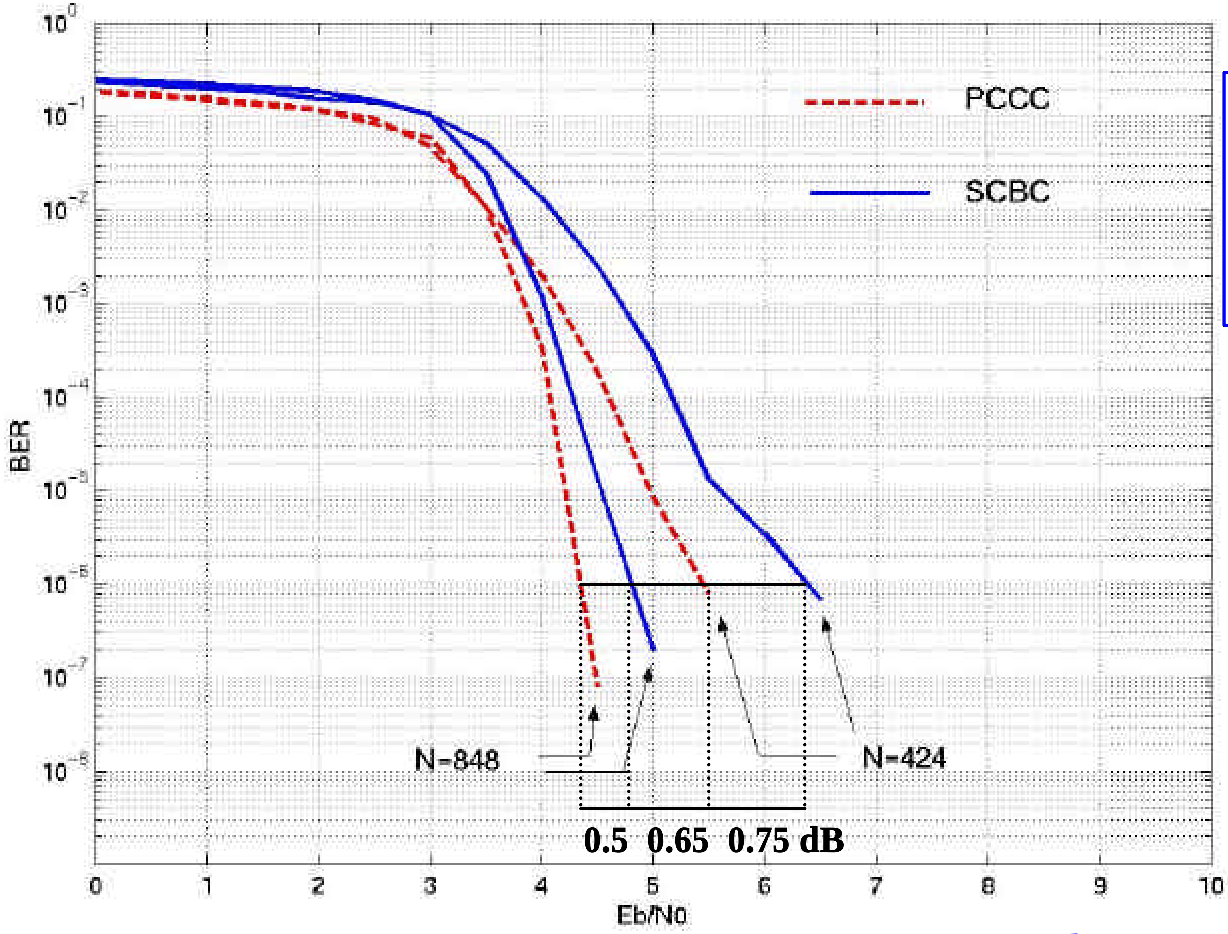


satellite to fixed terminal, LOS

k/n = 1/2

no difference for BER = 10^{-6}

Who corrects more errors ? BER for 16-QAM and AWGN

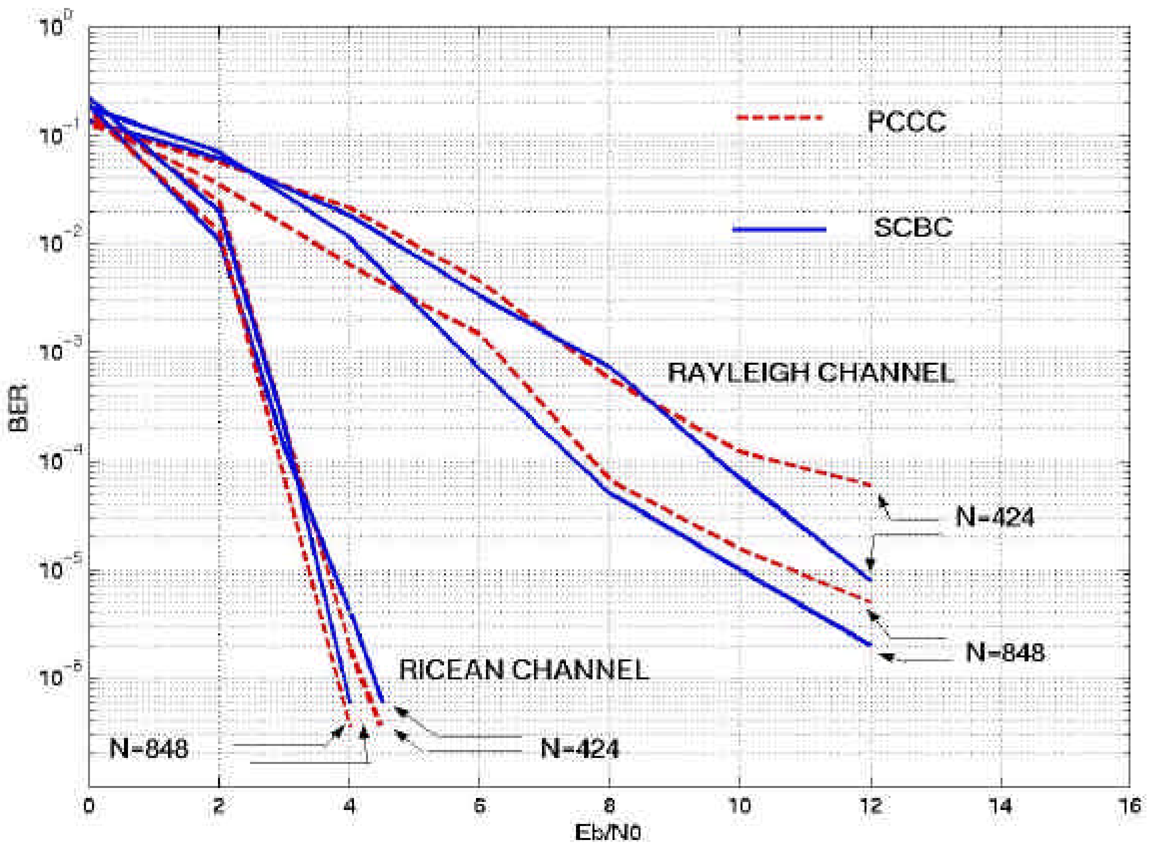


satellite to fixed terminal, LOS

k/n = 1/2

PCCCs perform between 0.5 and 0.75 dB better

Who corrects more errors ?
BER for fading, time varying channels



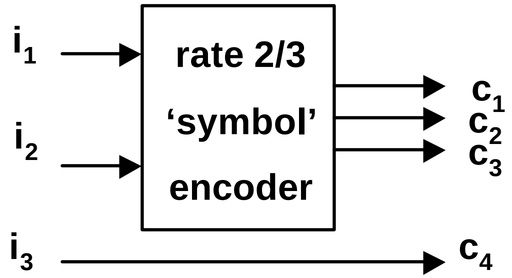
**satellite
to
mobile terminal,
LOS (Ricean)
or
non-LOS (Rayleigh)**

k/n = 1/2

**K = 5
terminal speed =
200 km/h
data rate =
1 Mb/s**

- **SCBCs perform better for the worst condition case (EbN0>12 dB)**
- **no difference for the Ricean (LOS) case**

Trellis coded modulation (Ungerboeck, 1982) : coding specific to modulation scheme improves bandwidth efficiency

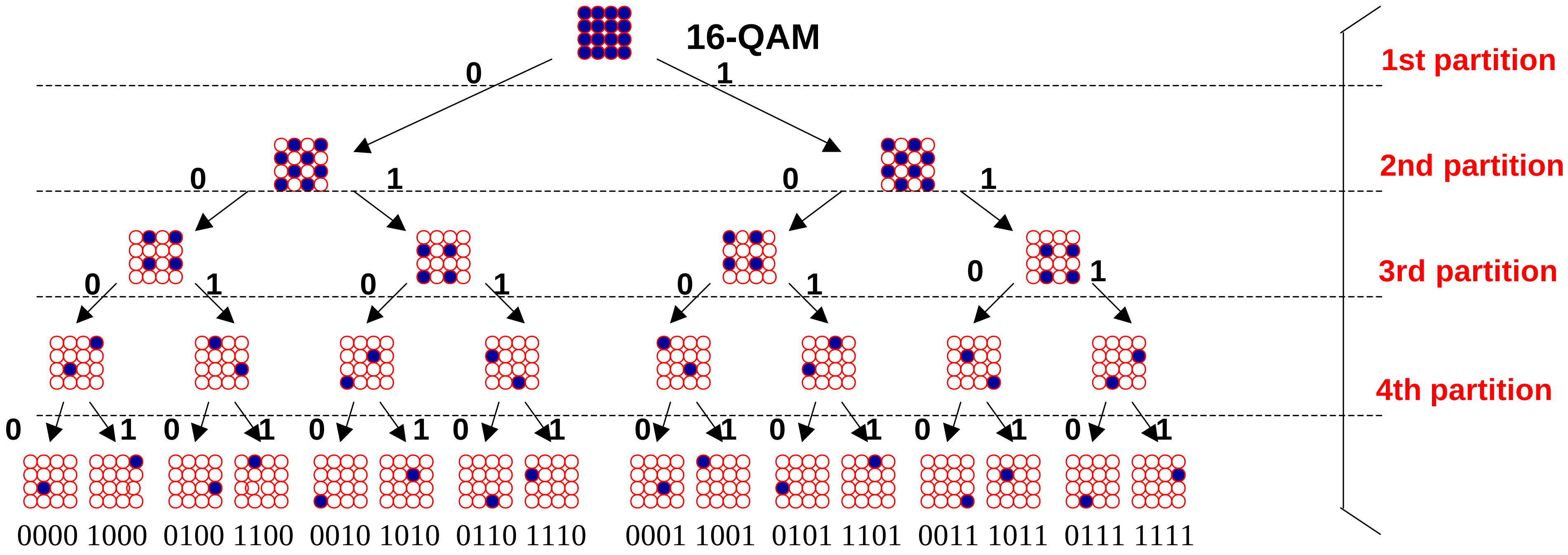


c_1 selects the 1st set partition

c_2 selects the 2nd set partition

c_3 selects the 4rd set partition

c_4 selects the 4th set partition



'less protected' bits are assigned to larger Euclidian distances ➡ **larger code rates with good error correction !**

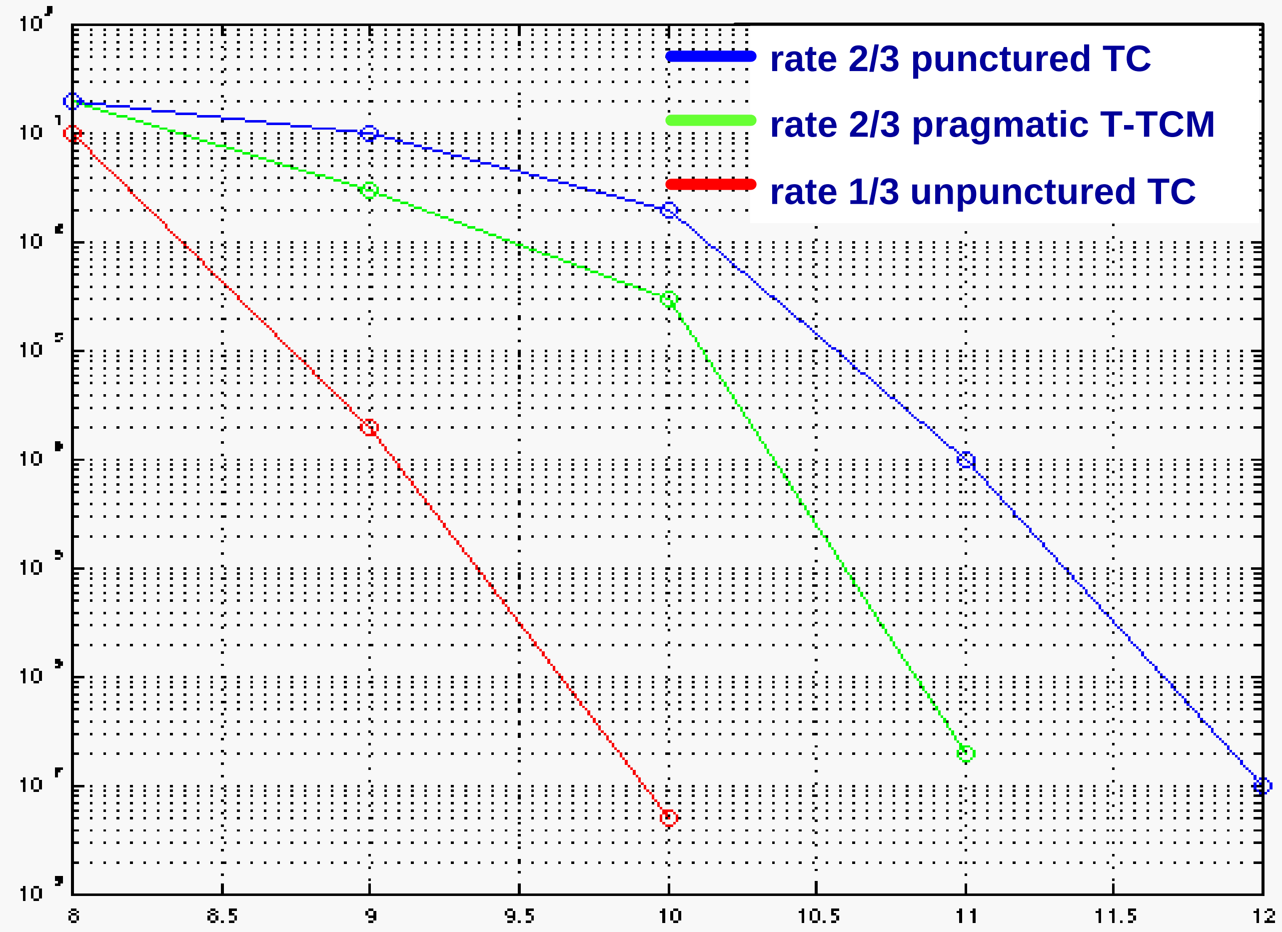
Turbo trellis coded modulation: from k/n = 1/3 to k/n = 2/3 and 3/4 with good performance

puncturing the 'classical' turbo code scheme and using modulation with higher constellations (16-QAM, 64-QAM)

- intelligent puncturing scheme taking into account the mapping $\rightarrow$ 'pragmatic approach' (Berrou)
- turbo code working on symbol level$\rightarrow$ 'symbol based trellis coded modulation (Robertson, Benedetto)

T-TCM !

Pragmatic T-TCM performances

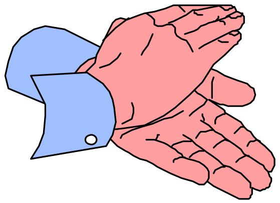


AWGN Channel

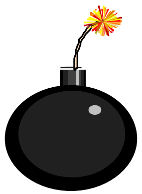
64-QAM - rate 2/3 - h= 4 bits/s/Hz

Ungerboeck mapping

We chose turbo codes for our channel coding/decoding ASIC



- + excellent coding gain
- + flexible (different coding rates, block sizes)
- + matches well with OFDM (WLAN)
- + leaves open the option for T-TCM in the future



- - complexity of the MAP algorithm
- - latency due to recursions and iterative process

IMEC's effort is solving the problem with T@MPO (to be shown in the rest of the day)