

# Theory and Design of Turbo and Related Codes

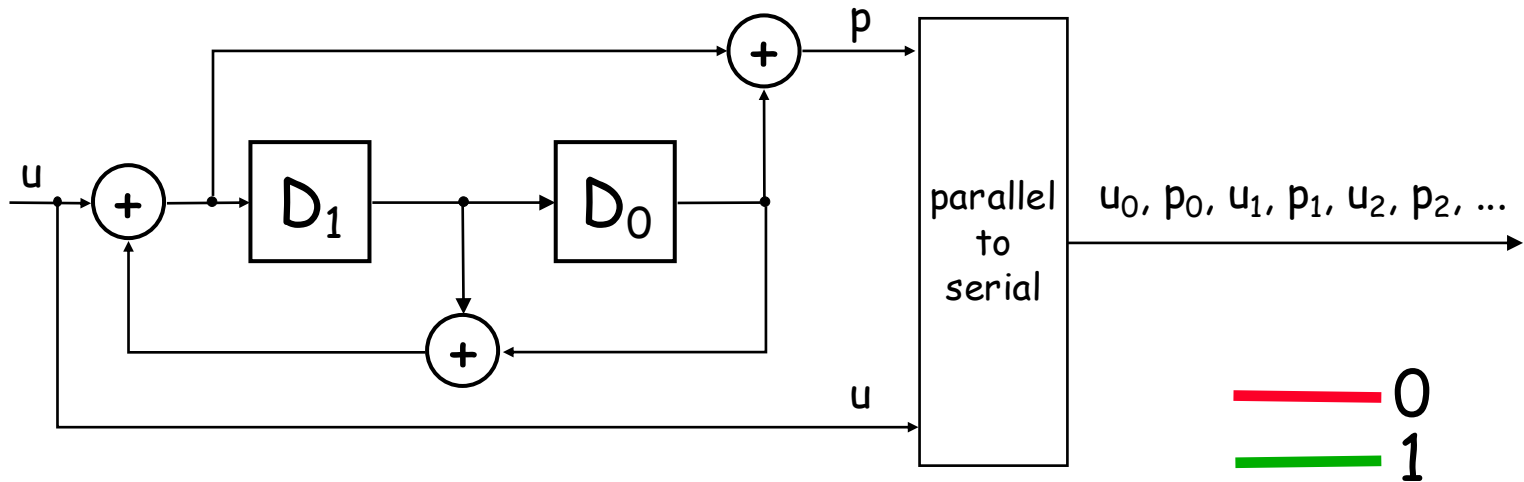
## *Lecture 8*

*Gottfried Lechner & Jossy Sayir*

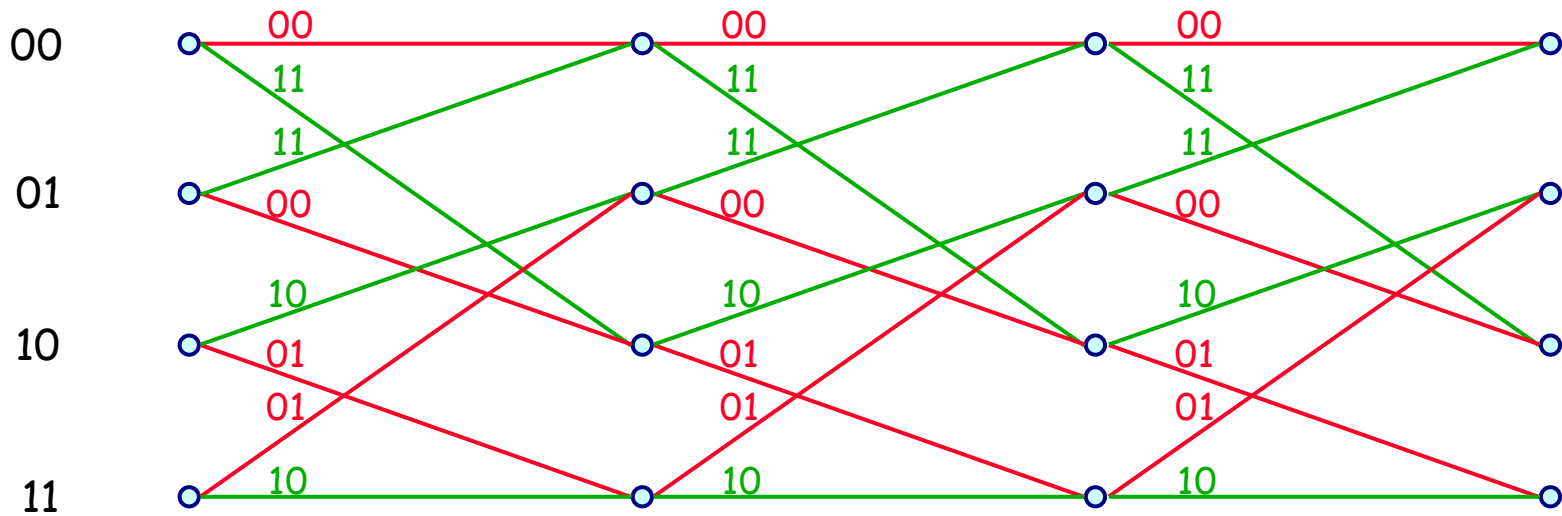
<http://userver.ftw.at/~jossy/turbo/index.html>

- We introduced another type of linear codes - **convolutional codes**
- These codes can be described using a **trellis diagramm**
- We derived two algorithms:
  - **Viterbi** algorithm - find the most likely **codeword**
  - **BCJR** algorithm - find the most likely **symbol**

# Trellis of a Convolutional Code Example

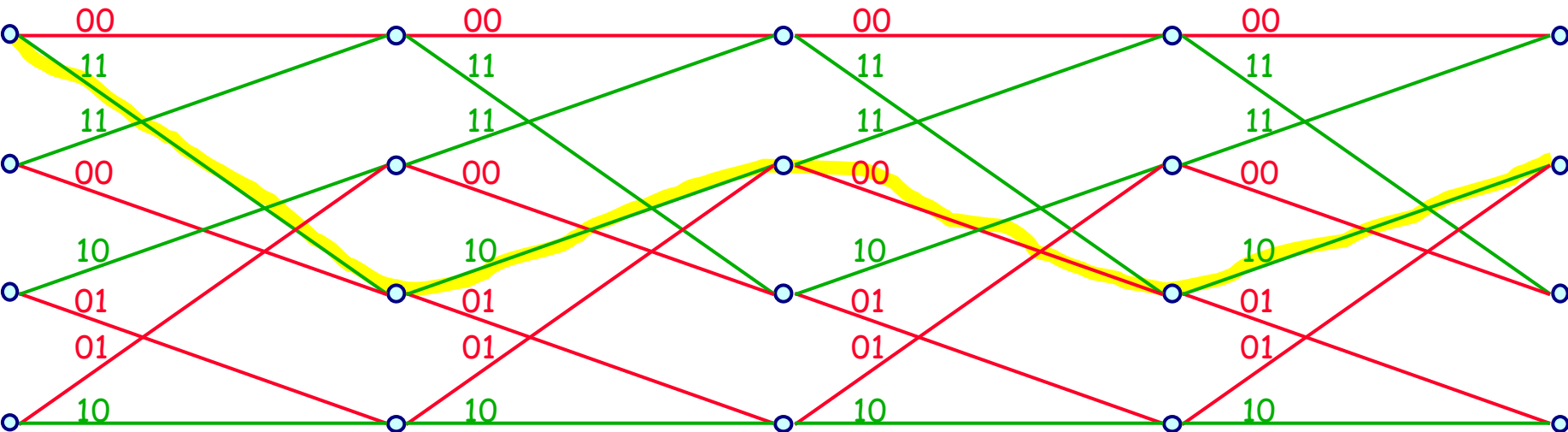


$D_1 D_0$



# Convolutional Encoding

$u =$	1	1	0	1				
$x =$	1	1	1	0	0	0	1	0
$t =$	-1	-1	-1	+1	+1	+1	-1	+1



Changing the domain from probabilities to the negative logarithm of probabilities results in the following relations.

$$z_i = -\log(p_i)$$

$$p_1 * p_2 \leq \min(p_1, p_2)$$

$$z_1 + z_2 \geq \max(z_1, z_2)$$

$$p_1 + p_2 \geq \max(p_1, p_2)$$

$$-\log(\exp(-z_1) + \exp(-z_2)) \leq \min(z_1, z_2)$$

- Find the **path** in the trellis with the **highest probability** (given the received vector  $y$ )
- $\text{path}_{\text{dec}} = \operatorname{argmax}_{\text{path}} \{P_{\text{path}}\}$
- $\text{path}_{\text{dec}} = \operatorname{argmin}_{\text{path}} \{-\log(P_{\text{path}})\}$

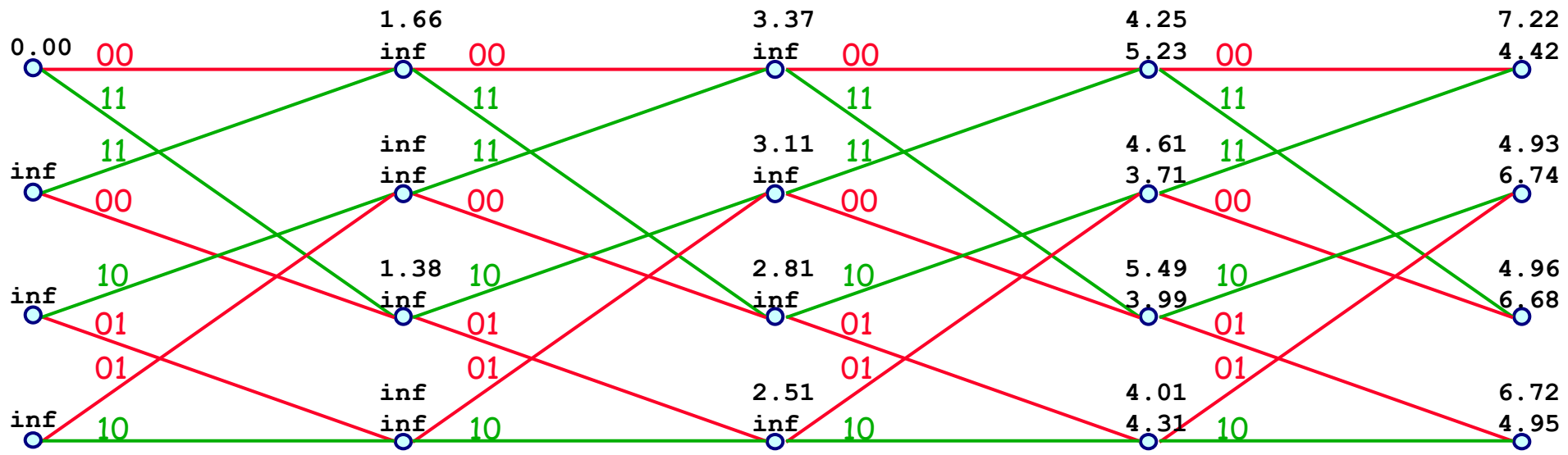


# Viterbi Decoding Example

$u =$	1		1		0		1	
$x =$	1	1	1	0	0	0	1	0
$t =$	-1	-1	-1	+1	+1	+1	-1	+1
$L_y =$	-1.73	+1.19	+0.04	-1.16	+1.84	+0.64	-4.05	-0.47

-log of propability

00	1.66	1.71	0.88	2.97
01	2.25	1.13	1.20	2.73
10	0.79	1.73	1.80	0.94
11	1.38	1.15	2.12	0.71



upper branch  
lower branch

- Find for every trellis stage the **symbol** with the **highest probability** (given the received vector  $y$ )
- This probability can be separated in
  - probability of path that ends at time  $k$
  - probability of path that begins at time  $k+1$
  - transition of time  $k$  to time  $k+1$

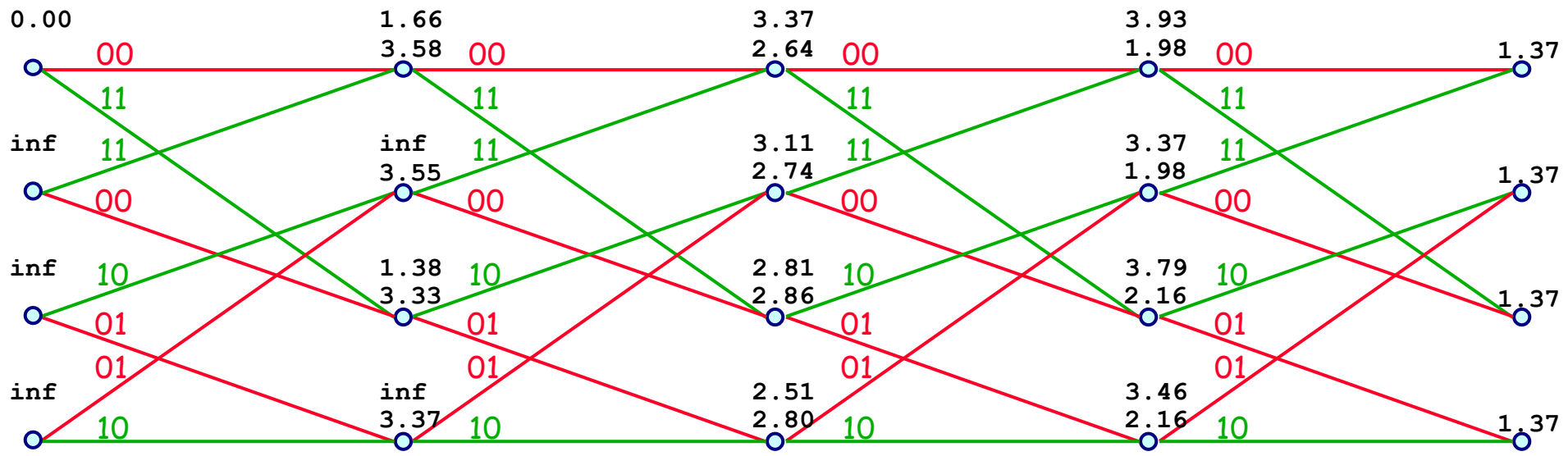


# BCJR Decoding Example

$$L_y = \begin{matrix} -1.73 & +1.19 & +0.04 & -1.16 & +1.84 & +0.64 & -4.05 & -0.47 \end{matrix}$$

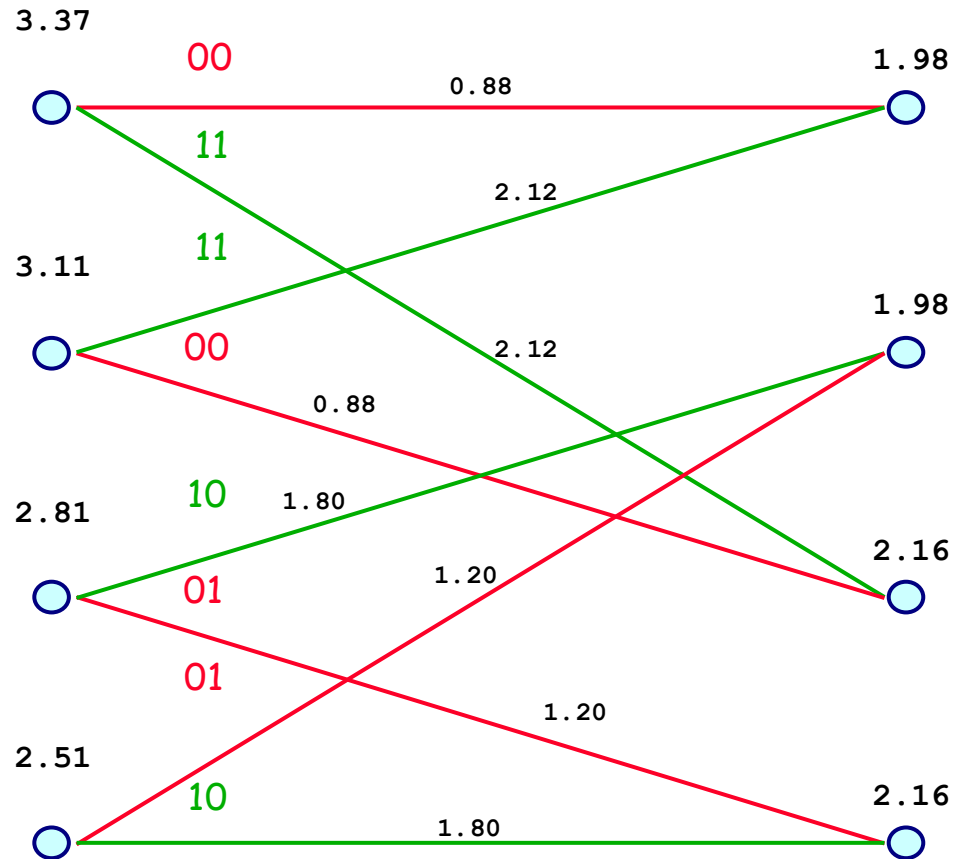
-log of propability

00	1.66	1.71	0.88	2.97
01	2.25	1.13	1.20	2.73
10	0.79	1.73	1.80	0.94
11	1.38	1.15	2.12	0.71



forward recursion  
backward recursion

# BCJR Decoding Example cont.



$$\begin{aligned}
 3.37 + 0.88 + 1.98 &= 6.23 \\
 3.11 + 0.88 + 2.16 &= 6.15 \\
 2.81 + 1.20 + 2.16 &= 6.17 \\
 2.51 + 1.20 + 1.98 &= 5.69 \\
 &4.65
 \end{aligned}$$

$$\begin{aligned}
 3.37 + 2.12 + 2.16 &= 7.65 \\
 3.11 + 2.12 + 1.98 &= 7.21 \\
 2.81 + 1.80 + 1.98 &= 6.59 \\
 2.51 + 1.80 + 2.16 &= 6.47 \\
 &5.49
 \end{aligned}$$

$$\begin{aligned}
 \text{LLR} &= -4.65 + 5.49 = 0.84 \\
 &\Rightarrow \text{decide for 0}
 \end{aligned}$$

- The **complexity** of trellis decoding is approximately **proportional to the "size"** of the trellis, where the size is the number of transitions (=block length) times the number of states.
- For a convolutional code with memory length  $m$  and block length  $N$  the complexity is

$$\propto N \times 2^m$$

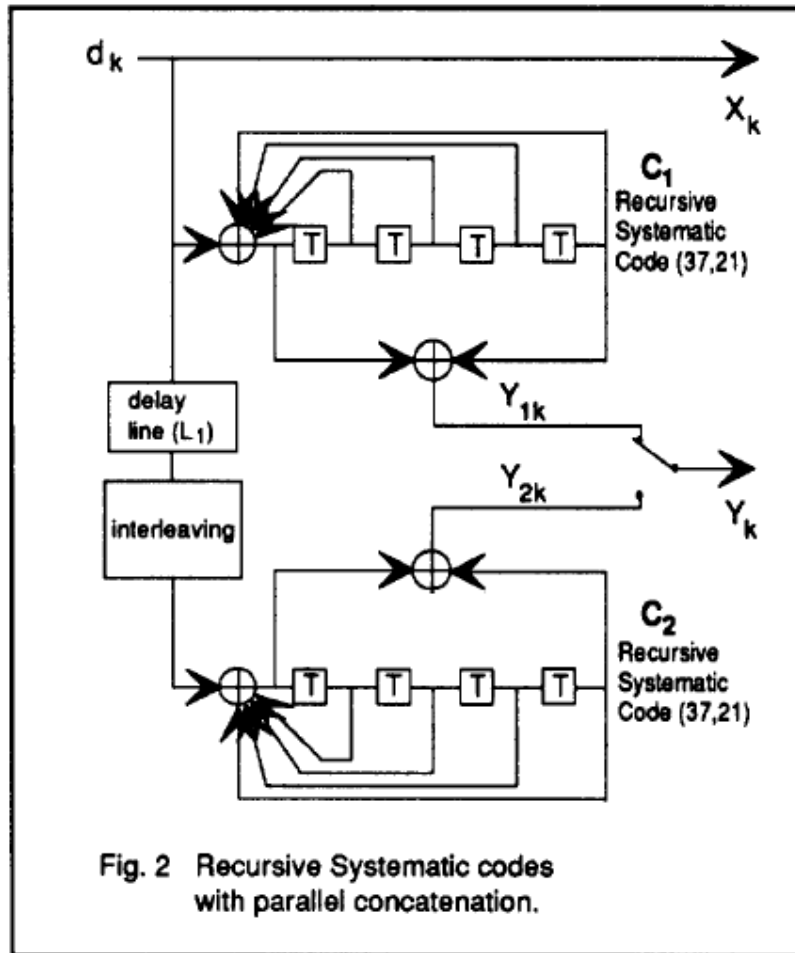
and the complexity per symbol is

$$\propto 2^m$$

- Therefore, the memory length is **limited by complexity constraints!**

- In principal the block length of convolutional codes is **infinite**. However, the performance of this code is not very good.
- In fact only the neighbouring symbols influence the current symbol and that reduces the "**effective block length**".
- The "effective block length" can be increased by increasing the length of the memory, but this would lead to more states and therefore, higher complexity per symbol...

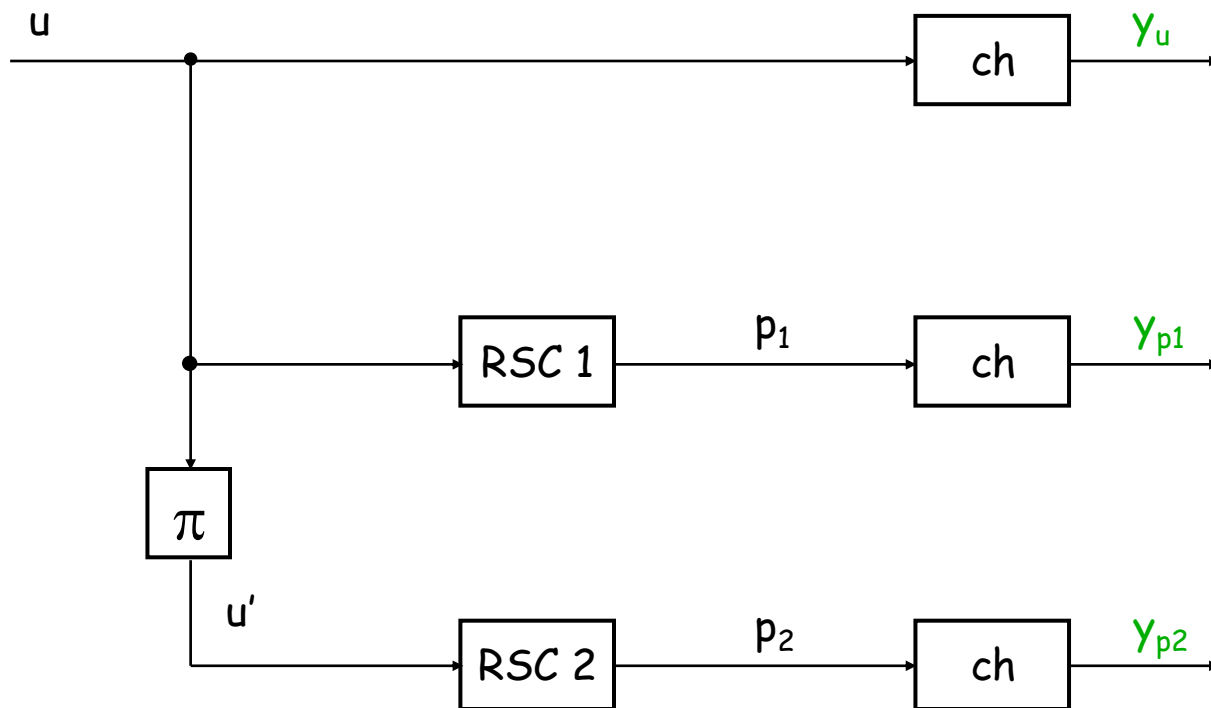
# Parallel Concatenation - Turbo Codes



By using two parallel concatenated systematic codes, separated by an interleaver, the "effective block length" is increased.

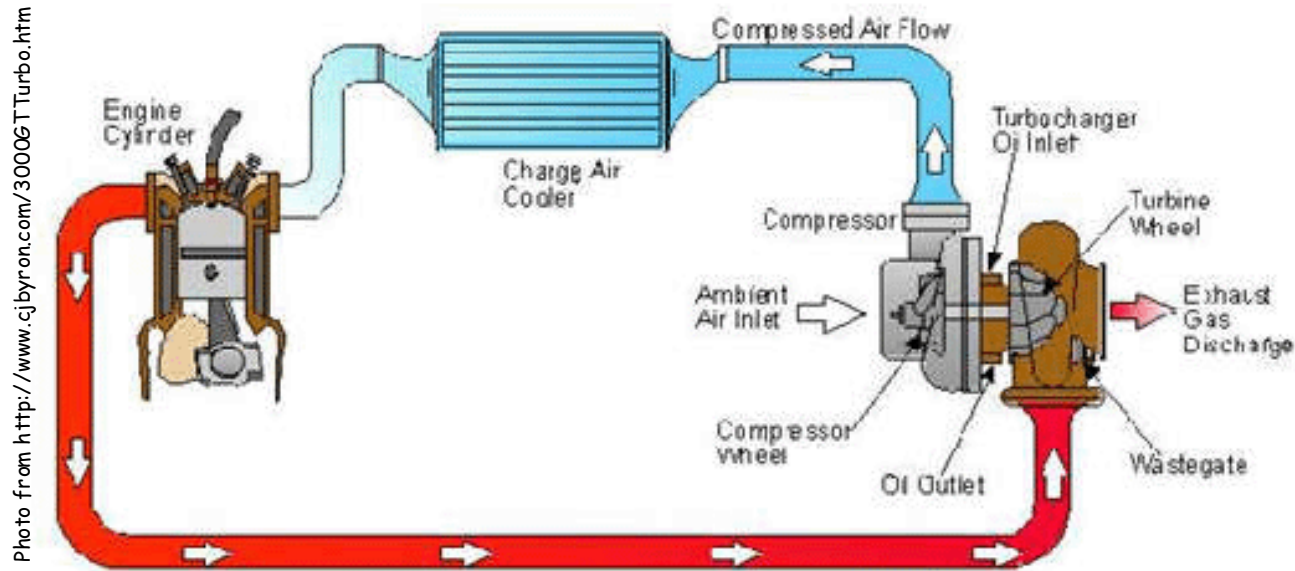
C. Berrou, A. Glavieux and P. Thitimajshima "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes", ICC 1993, Genève, Switzerland

# Turbo Codes – Encoder



How can this code be decoded?

# Why "Turbo"?

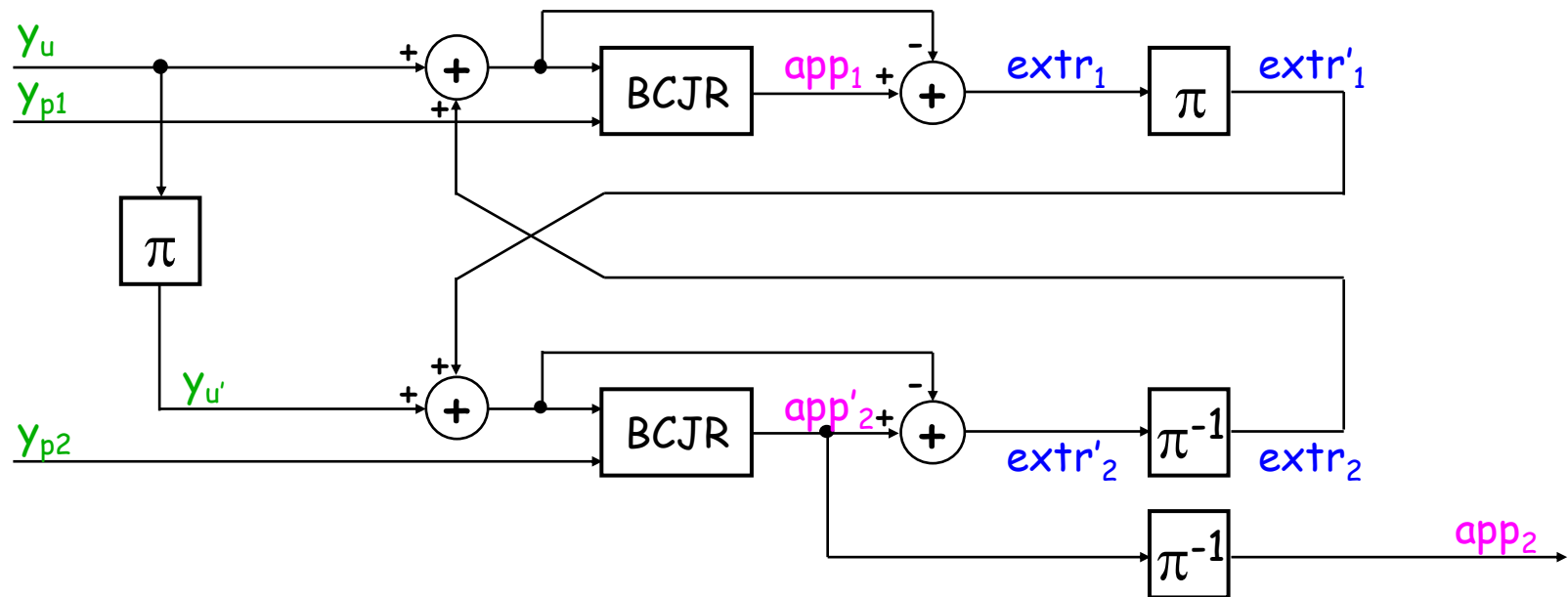


The turbo engine uses a **feedback** to increase the performance of the overall system.

The turbo decoder works similar.

# How to Decode a Turbo Code

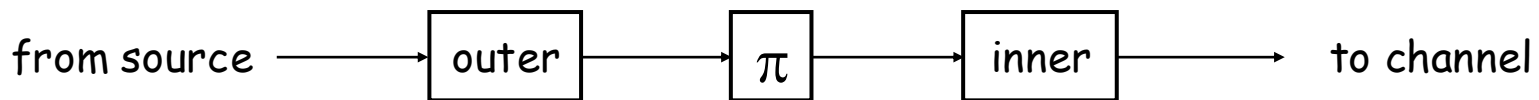
- The key idea is **not to decode the complete code** (which would be complicated), but to decode the two component codes alternating and feed back information gained by the other decoder.



- By decoding every component code instead of the overall code we achieve a decoding **complexity per symbol that is constant**.
- Using a larger interleaver results in a larger block length.
- Using longer (more memory) component codes results in higher decoding complexity.

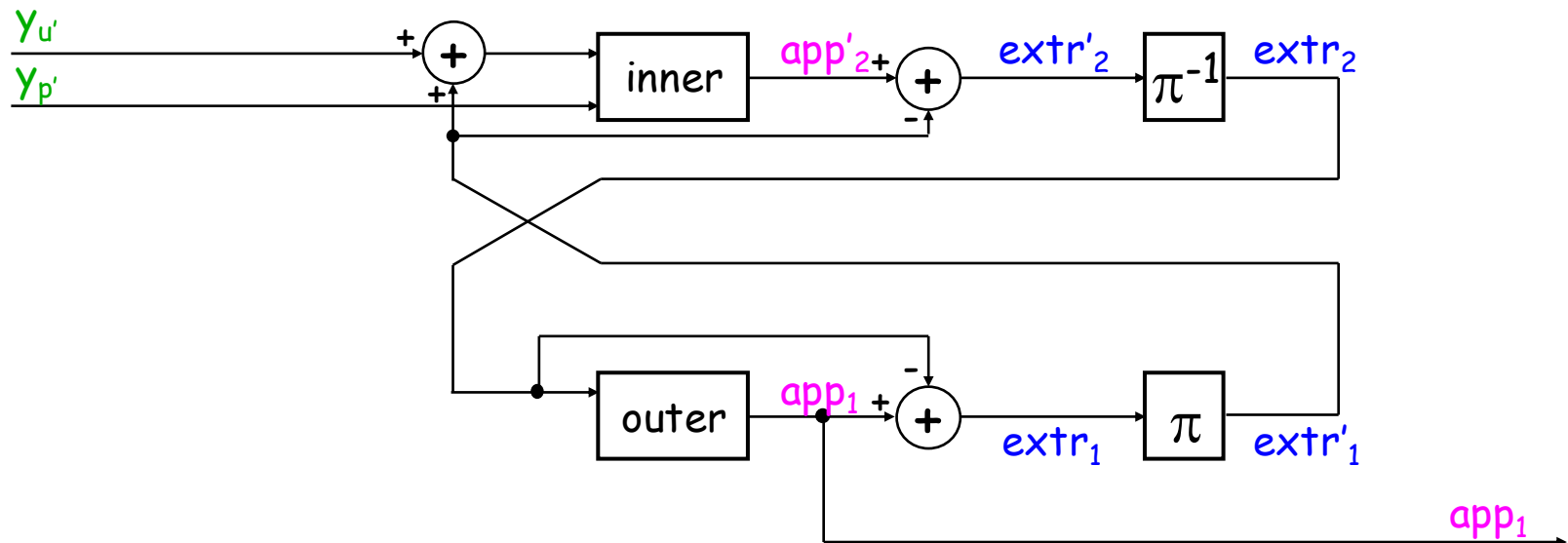
# Serial Concatenation

- Parallel concatenation is not the only way of combining codes.
- Instead of using two codes in parallel we use two codes that are **serially concatenated** (again separated by an interleaver)

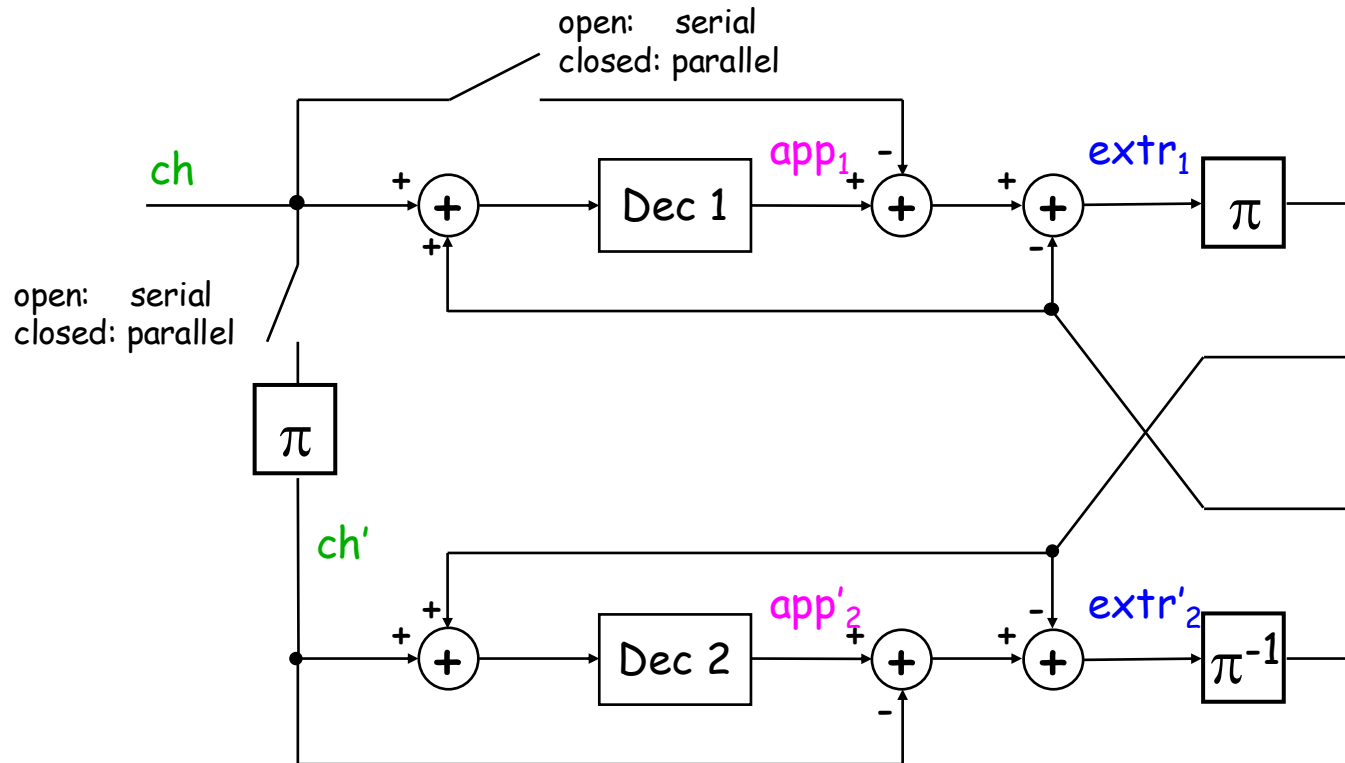


# Serial Concatenation - Decoder

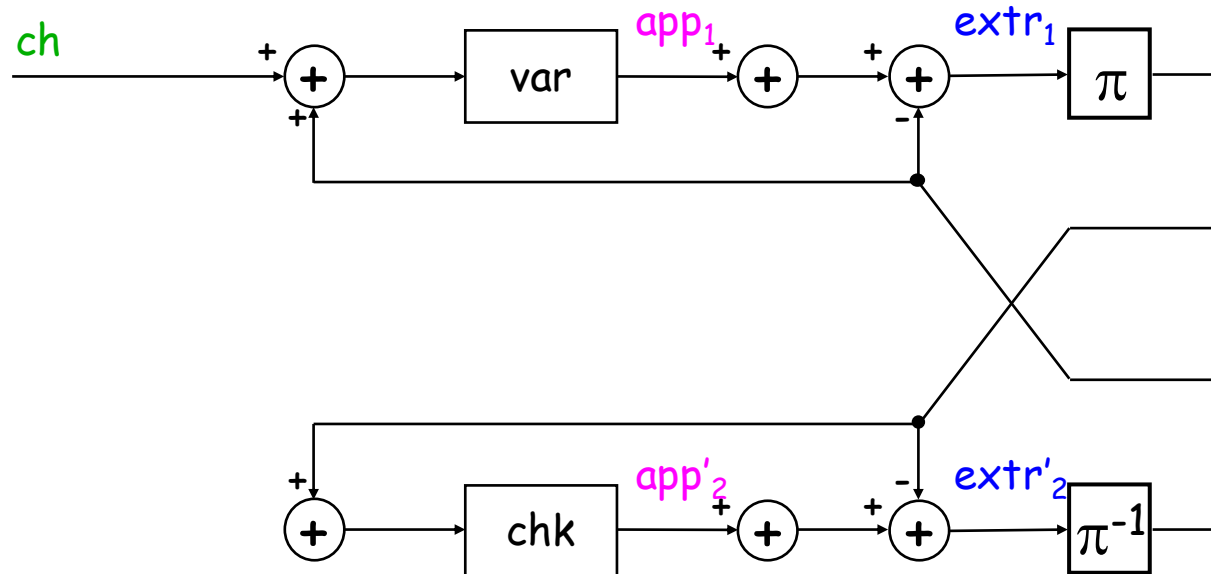
- Decoding is similar to parallel concatenated codes. But here, the outer code receives no messages directly from the channel.



# General Decoding Model



The decoder of parallel concatenated, serial concatenated and LDPC codes is a special case of this decoding model.



The decoder structure of an LDPC code is the same as for a serial concatenated code. But the LDPC encoder can not be represented like this.

## Factor graph of a convolutional code

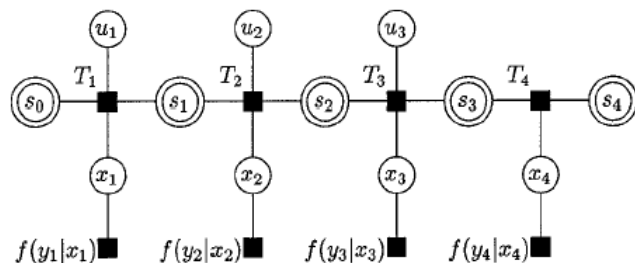


Fig. 13. The factor graph in which the forward/backward algorithm operates: the  $s_i$  are state variables, the  $u_i$  are input variables, the  $x_i$  are output variables, and each  $y_i$  is the output of a memoryless channel with input  $x_i$ .

## Factor graph of a turbo code

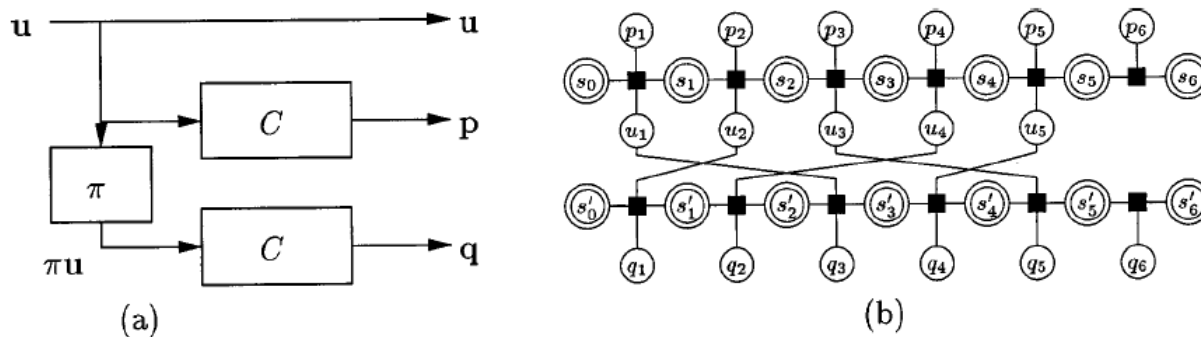


Fig. 16. Turbo code. (a) Encoder block diagram. (b) Factor graph.

B.J. Frey, F.R. Kschischang, and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," IEEE Trans. Inform. Theory, Feb. 2001.