Systeme II

3. Die Datensicherungsschicht

Christian Schindelhauer
Technische Fakultät
Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg

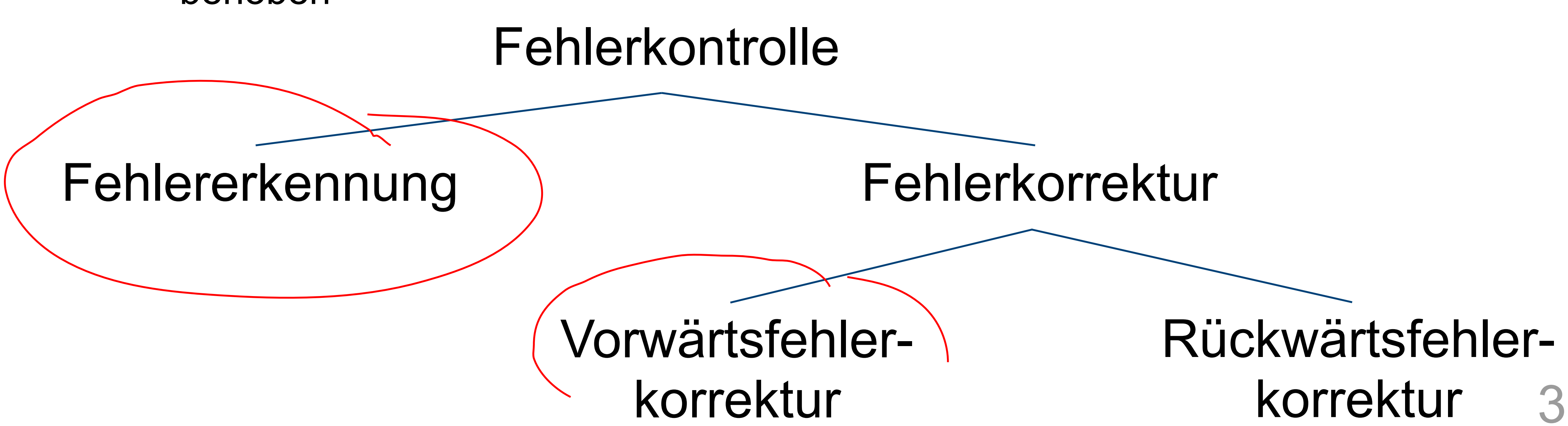
Version 14.05.2013

Fehlerkontrolle

CoNe Freiburg

0 1 0 1 0 1 1 1

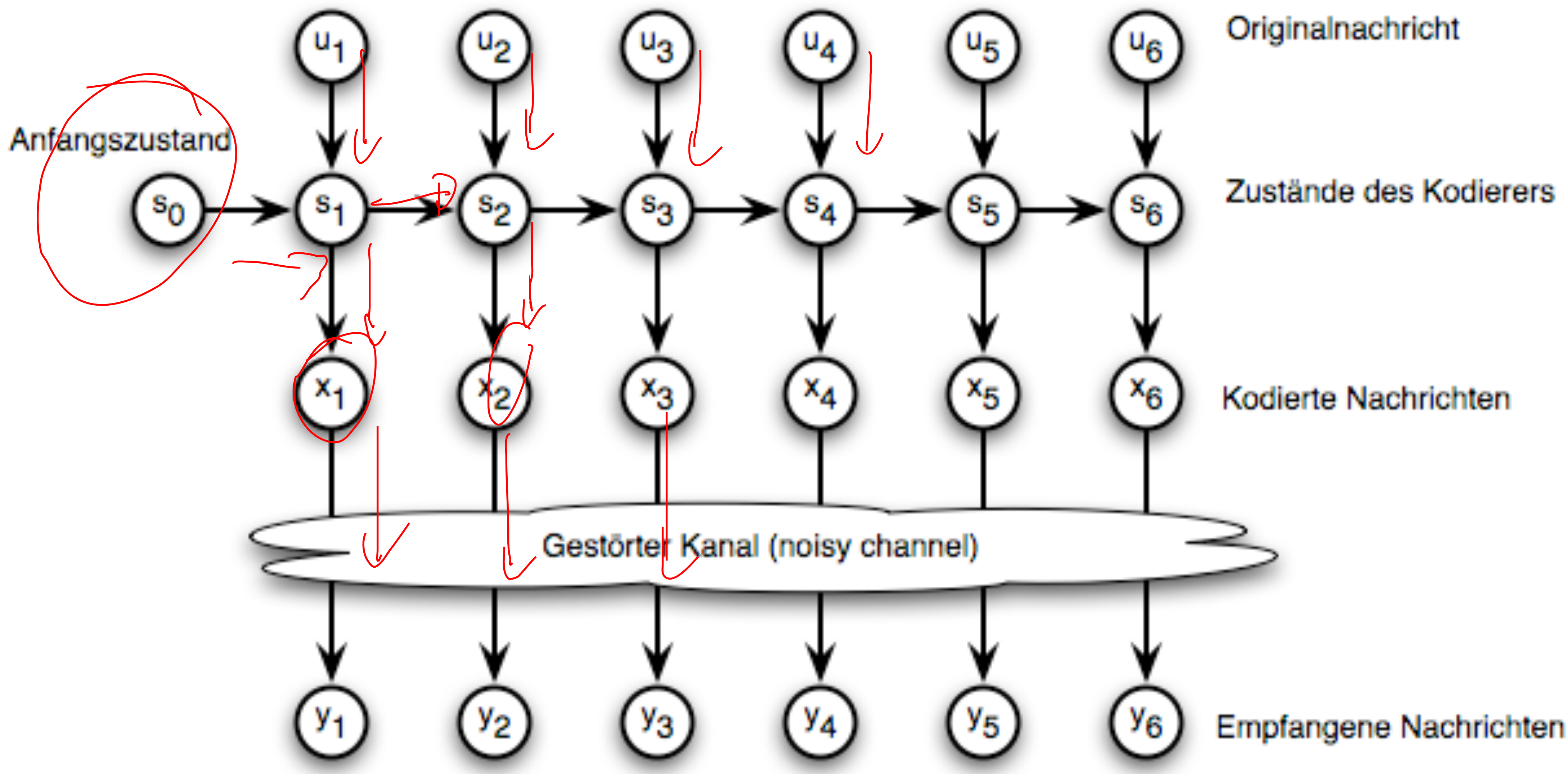
- Zumeist gefordert von der Vermittlungsschicht
  - Mit Hilfe der Frames
- Fehlererkennung
  - Gibt es fehlerhaft übertragene Bits?
- Fehlerkorrektur
  - Behebung von Bitfehlern
  - Vorwärtsfehlerkorrektur (Forward Error Correction)
    - Verwendung von redundanter Kodierung, die es ermöglicht Fehler ohne zusätzliche Übertragungen zu beheben
  - Rückwärtsfehlerkorretur (Backward Error Correction)
    - Nach Erkennen eines Fehlers, wird durch weitere Kommunikation der Fehler behoben



UNI FREIBURG

Faltungs-Codes

- Faltungs-Codes (Convolutional Codes)
  - Daten und Fehlerredundanz werden vermischt.
  - k Bits werden auf n Bits abgebildet
  - Die Ausgabe hängt von den k letzten Bits und dem internen Zustand ab.



Beispiel

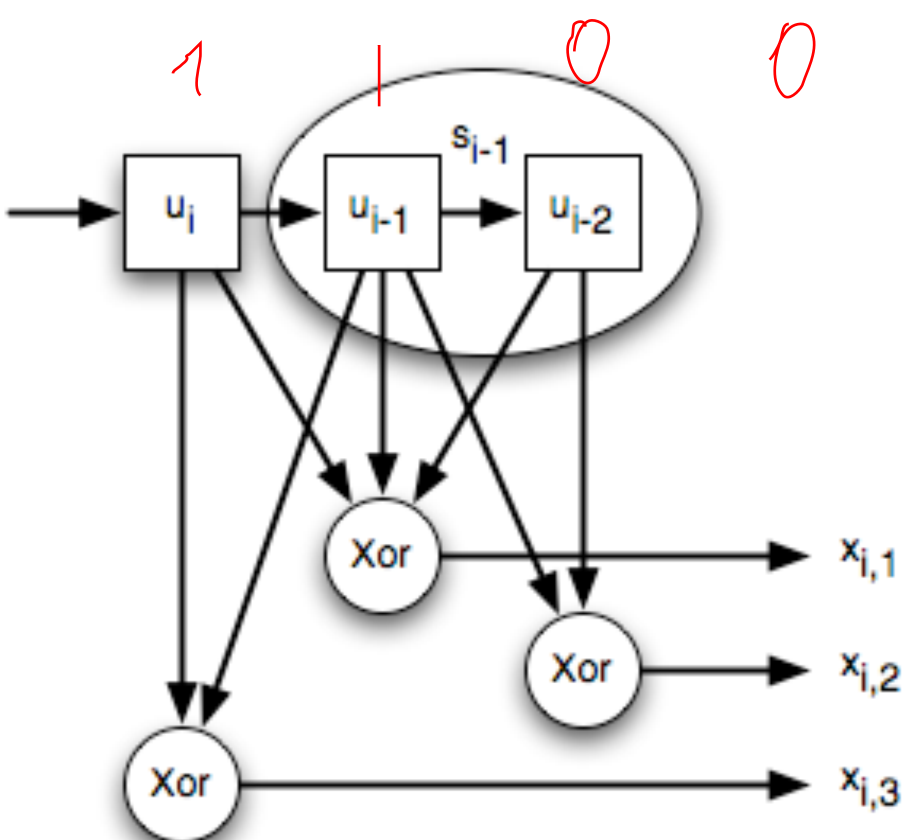
$(a \oplus b) \oplus c$

1 0 1

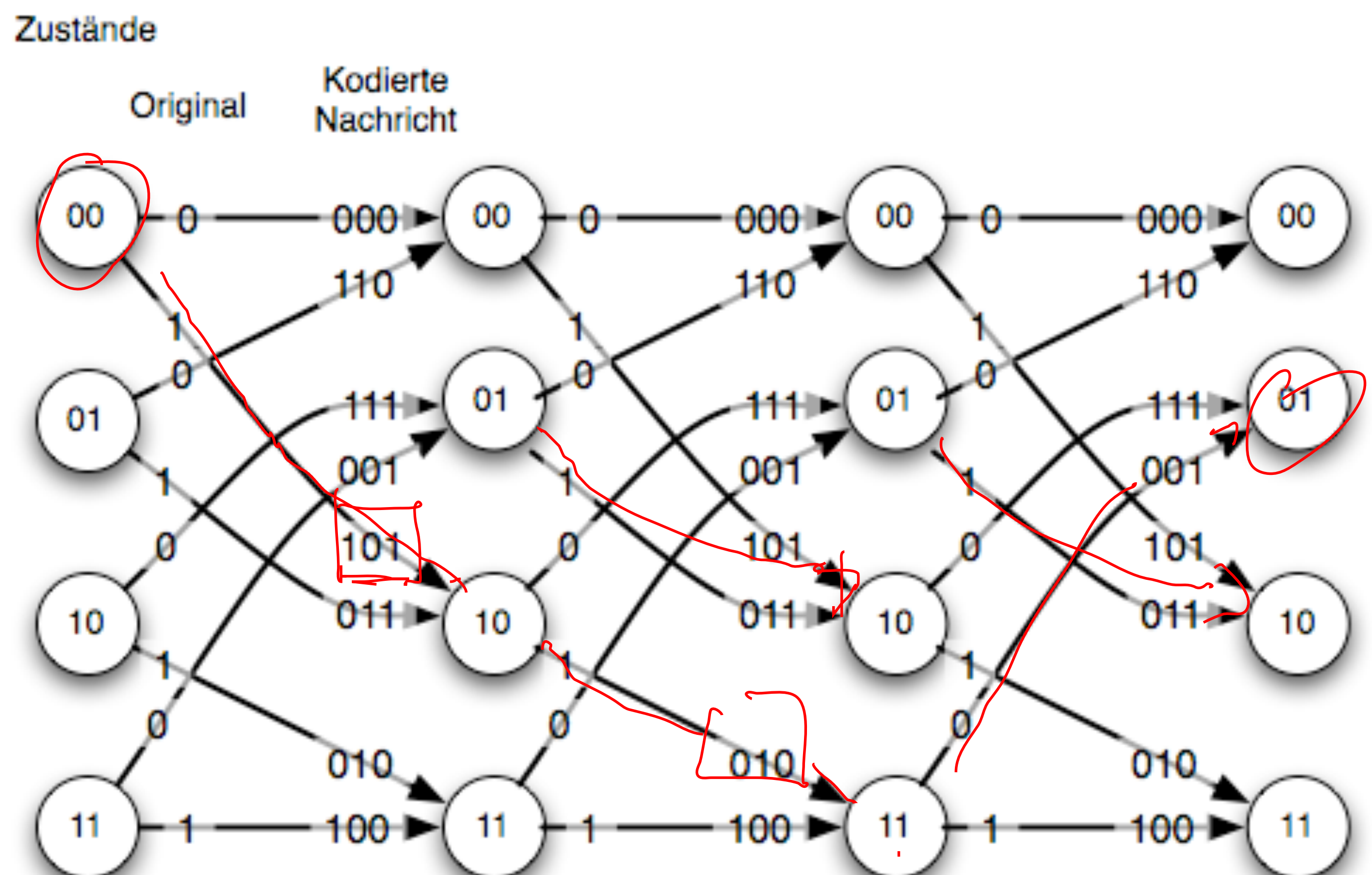
1 0 1 0 1 0 0 0 1

1 0 1 0 1 1 0 1 1

Faltungs-Kodierer



Trellis-Diagramm



UNI FREIBURG

Dekodierung der Faltungs-Codes: Algorithmus von Viterbi

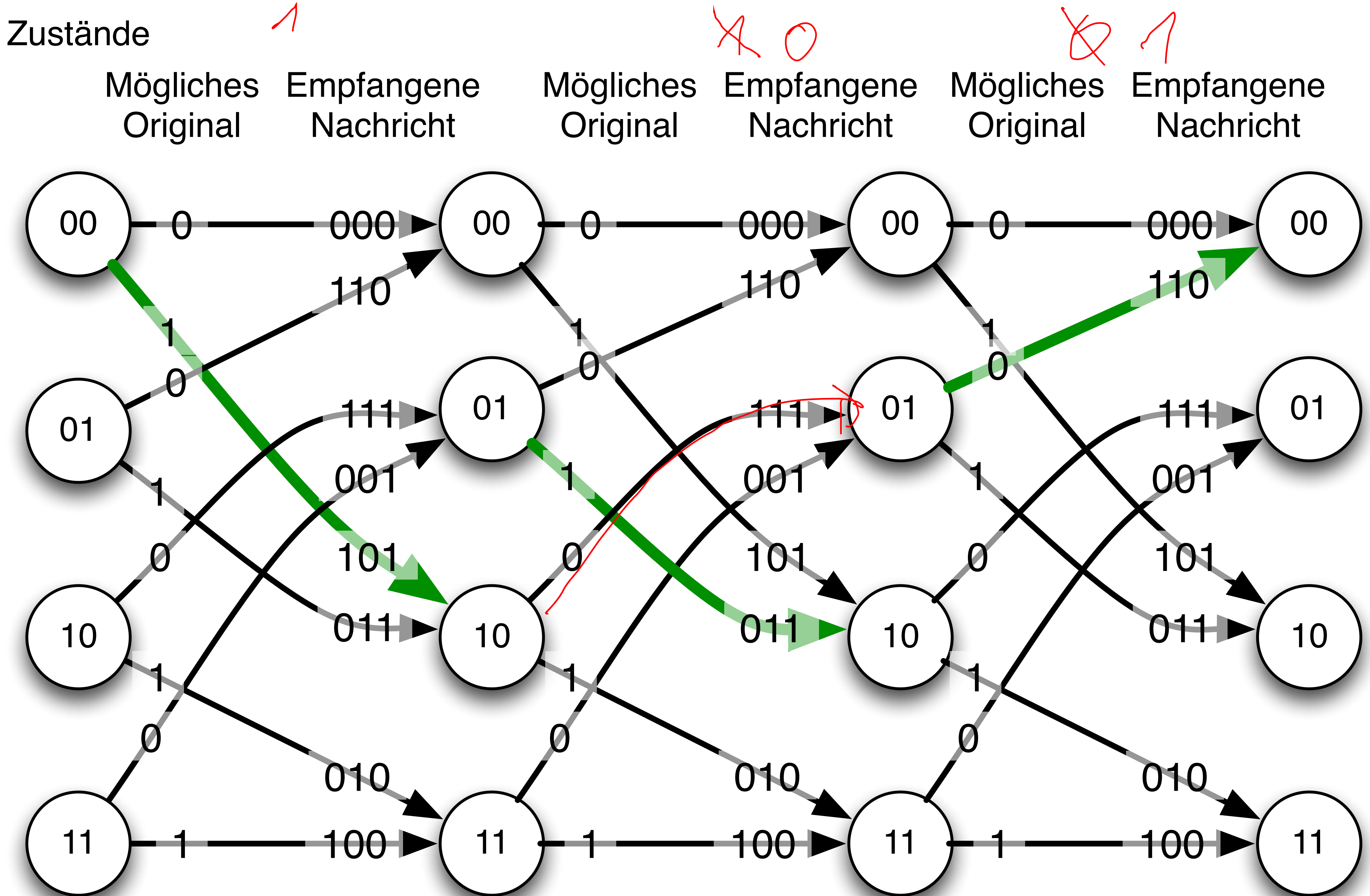
- Dynamische Programmierung
- Zwei notwendige Voraussetzungen für Dekodierung
  - (für den Empfänger) unbekannte Folge von Zuständen
  - beobachtete Folge von empfangenen Bits (möglicherweise mit Fehler)
- Der Algorithmus von Viterbi bestimmt die warscheinlichste Folge von Zuständen, welches die empfangenen Bits erklärt
  - Hardware-Implementation möglich

CoNe Freiburg

Dekodierung (I)

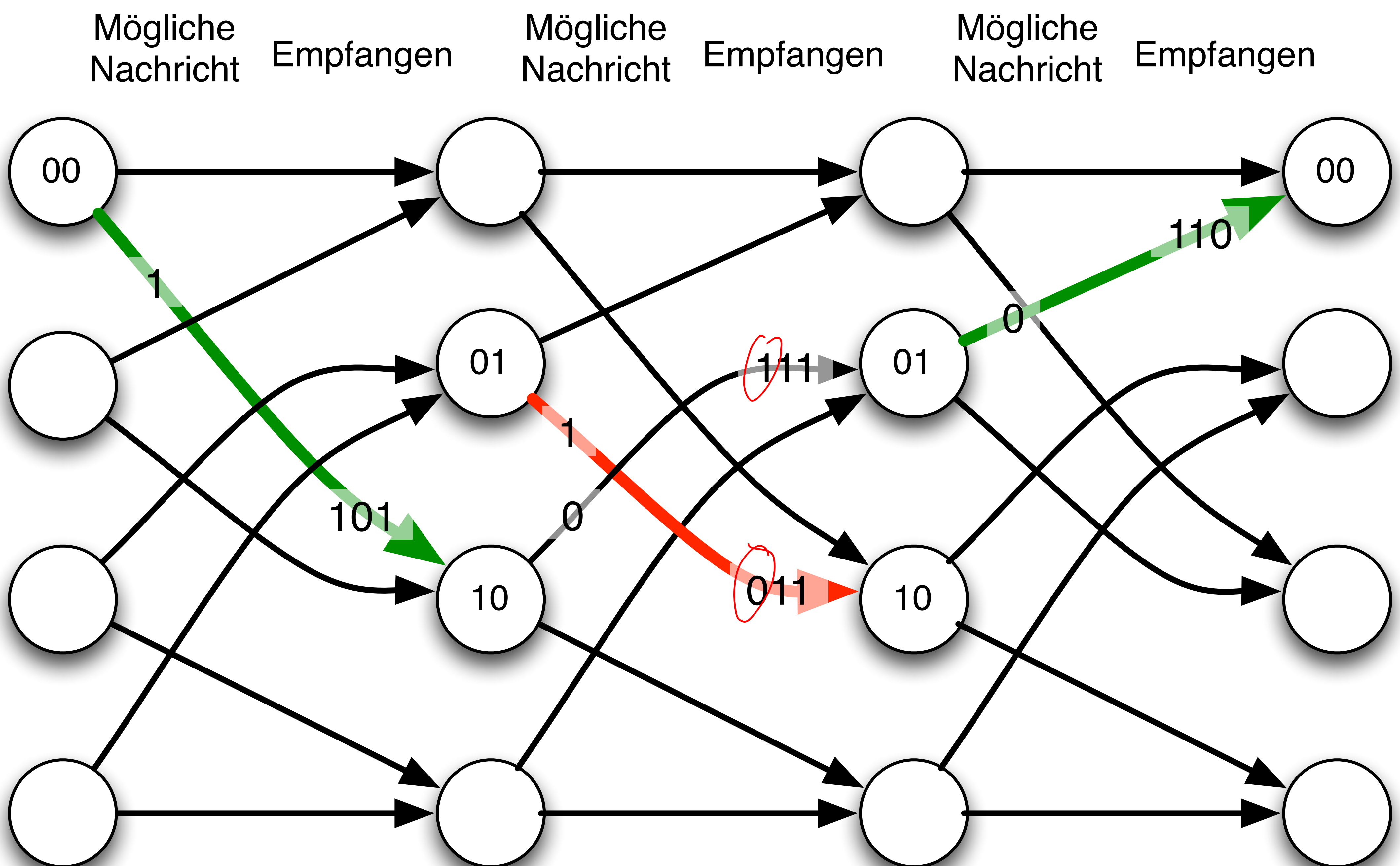
101 111 011

101 1011 110

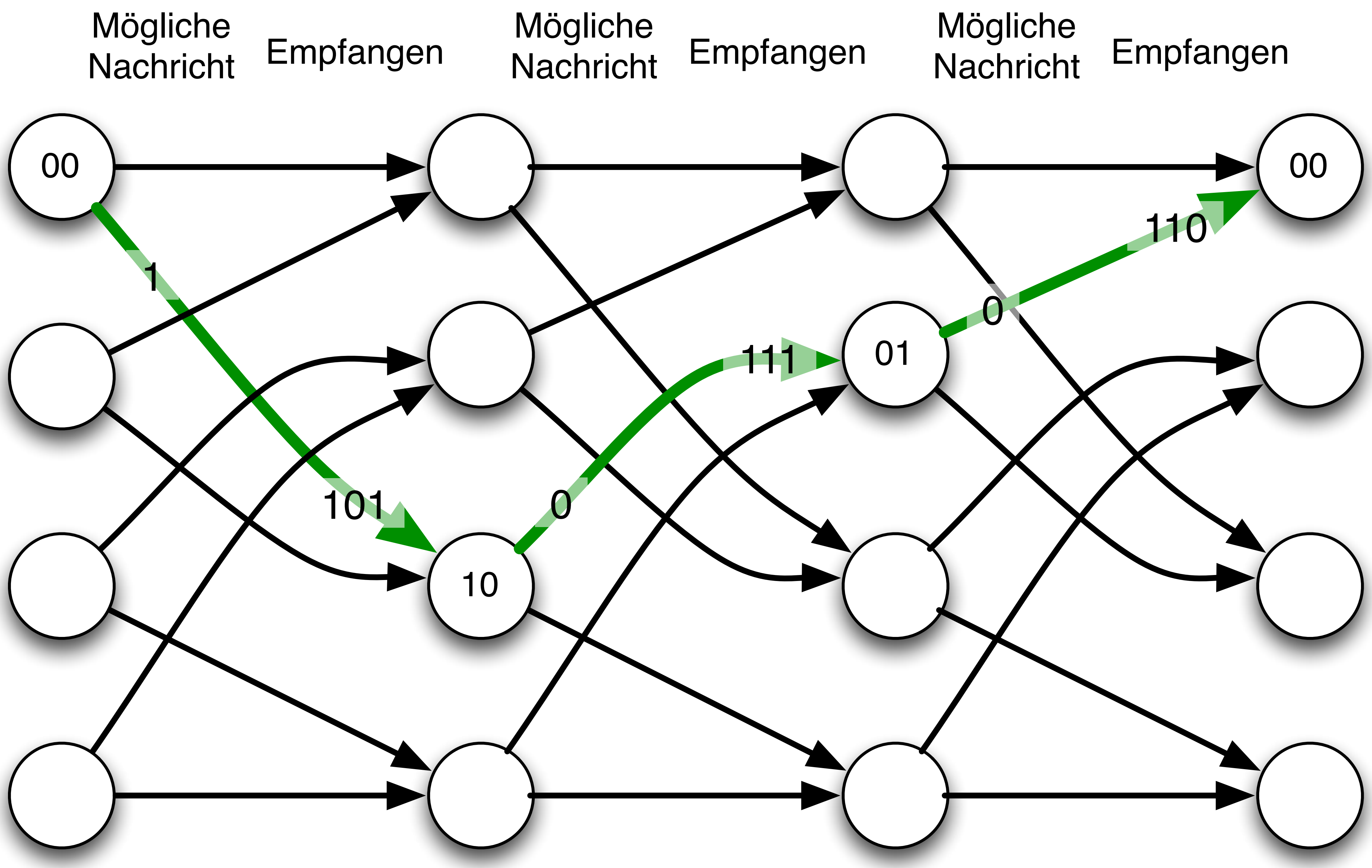


UNI FREIBURG

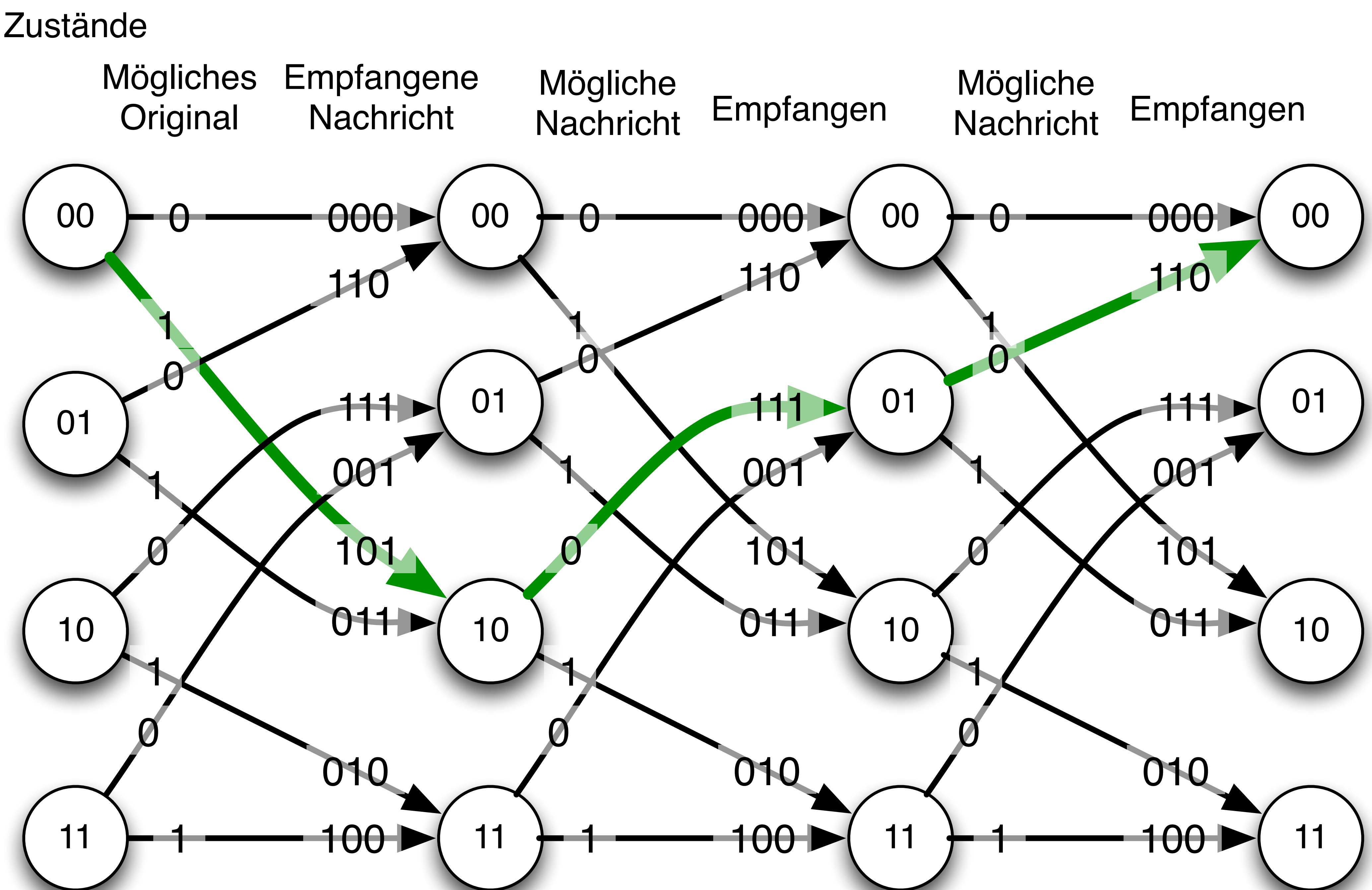
Dekodierung (II)



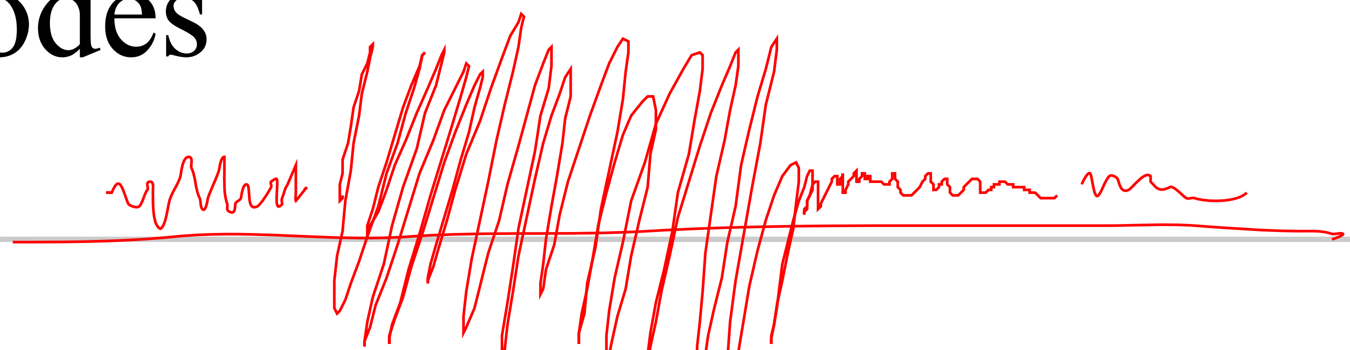
Dekodierung (III)



Dekodierung (IV)

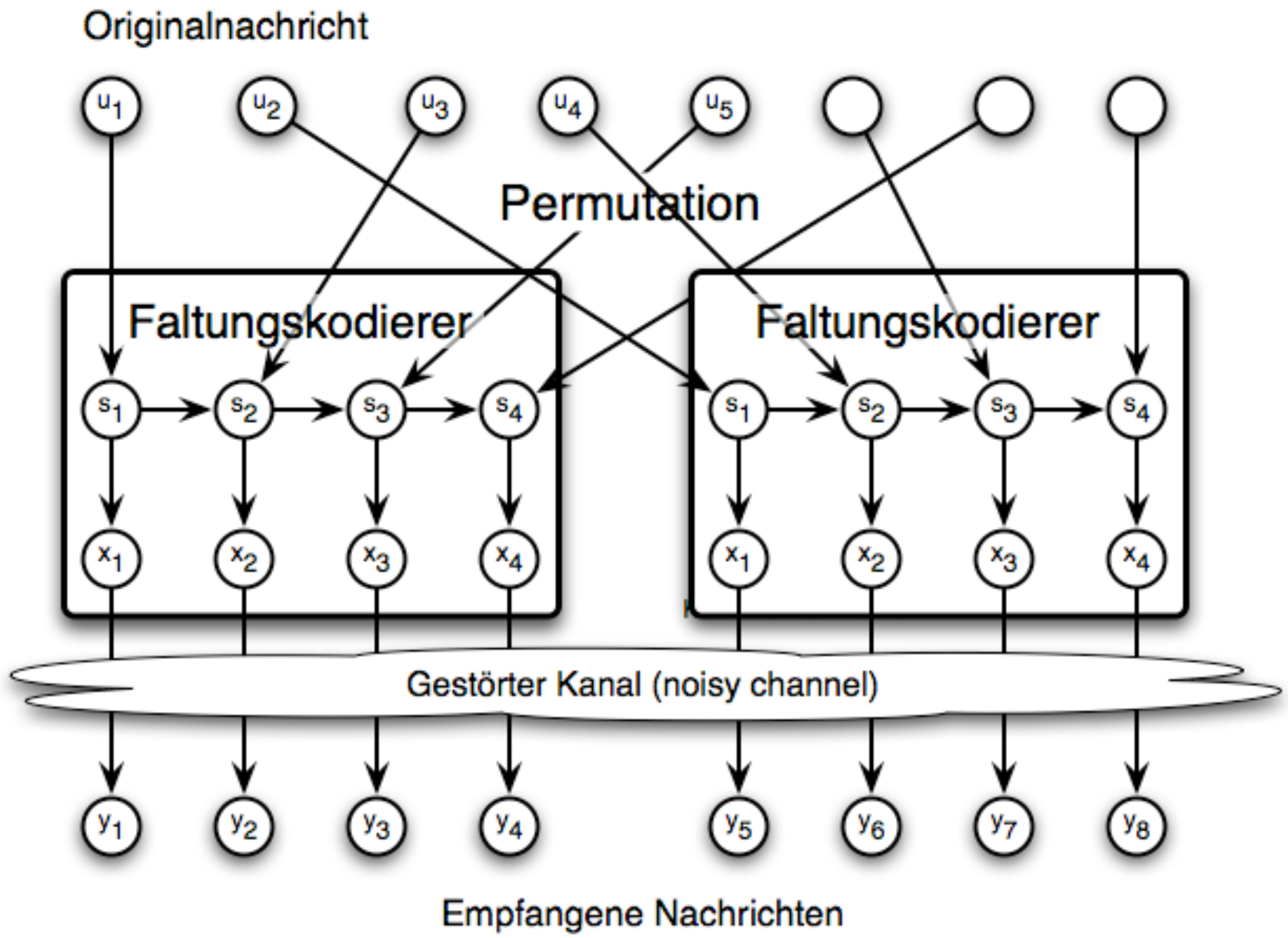


Turbo-Codes



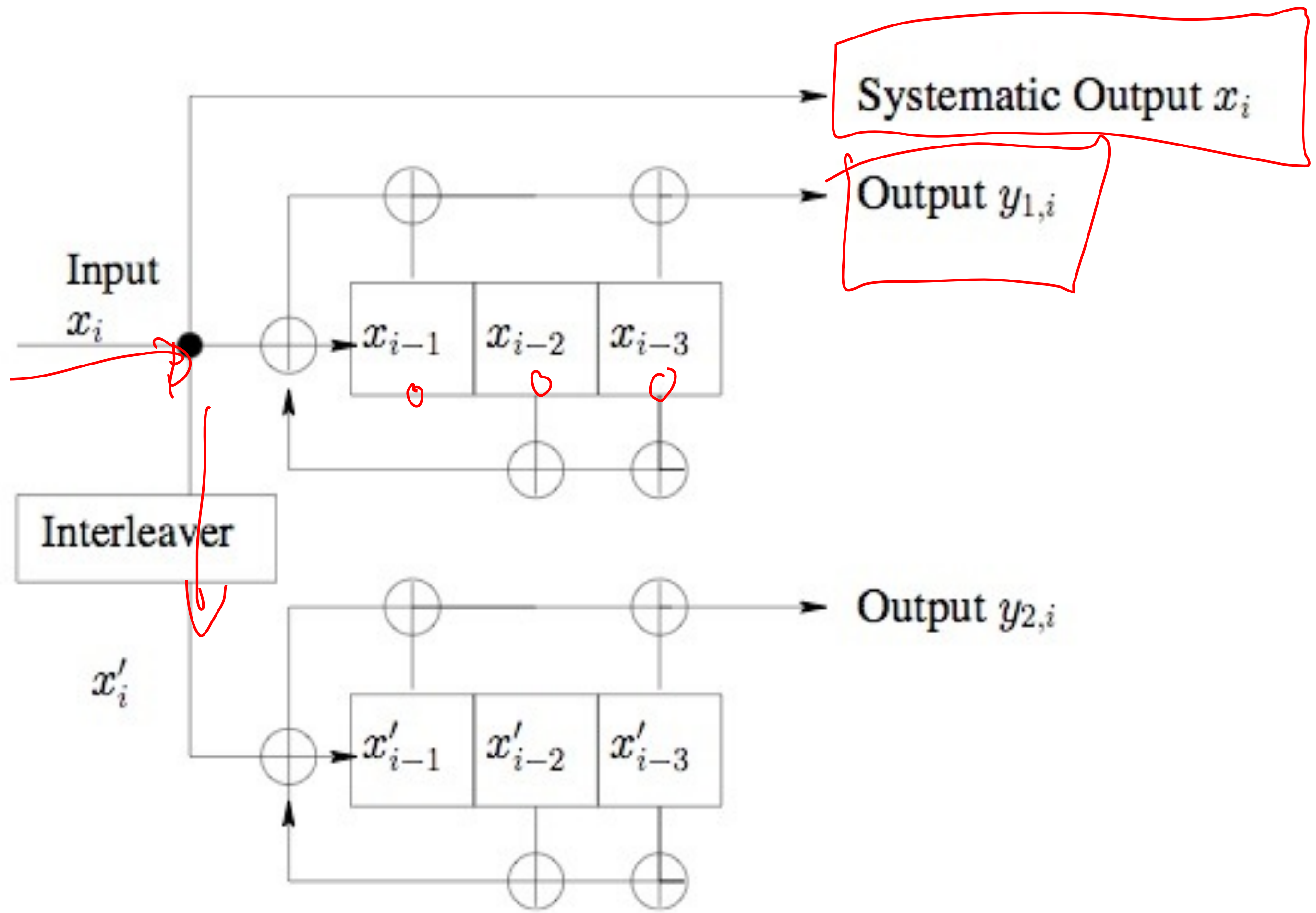
0101 ???????? 0110 10 1 010 110

- Turbo-Codes sind wesentlich effizienter als Faltungs-Codes
  - bestehen aus zwei Faltungs-Codes welche abwechselnd mit der Eingabe versorgt werden.
  - Die Eingabe wird durch eine Permutation (Interleaver) im zweiten Faltungs-Code umsortiert



Turbo-Codes

- Beispiel:
  - UMTS Turbo-Kodierer
- Dekodierung von Turbo-Codes ist effizienter möglich als bei Faltungscodes
- Kompensation von Bursts



Interleavers

- Fehler treten oftmals gehäuft auf (Bursts)
  - z.B.: Daten: 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - mit Fehler: 0 1 2 3 **? ? ? ? ?** 9 A B C D E F
- Dann scheitern klassische Kodierer ohne Interleavers
  - Nach Fehlerkorrektur (zwei Zeichen in Folge reparierbar):
    0 1 2 3 4 5 **?** 7 8 9 A B C D E F
- Interleaver:
  - Permutation der Eingabekodierung:

    0 1 2 3
    4 5 6 7
    8 9 A B
    C D E F
  - z.B. Row-column Interleaver:
    0 4 8 C 1 5 9 D 2 6 A E 3 7 B F
  - mit Fehler: 0 4 8 C **? ? ? ? ?** 6 A E 3 7 B F
  - Rückpermutiert: 0 **? ?** 3 4 **?** 6 7 8 **?** A B C D **?** F
  - nach FEC: 0 1 2 3 4 5 6 7 8 9 A B C D E F

Fehlererkennung: CRC

- Effiziente Fehlererkennung: Cyclic Redundancy Check (CRC)
- Praktisch häufig verwendeter Code
  - Hoher Fehlererkennungsrate
  - Effizient in Hardware umsetzbar
- Beruht auf Polynomarithmetik im Restklassenring Z_2
  - Zeichenketten sind Polynome
  - Bits sind Koeffizienten des Polynoms

Rechnen in Z_2

$0 - 1 = x$

$0 = x + 1$

$x = 1$

- Rechnen modulo 2:
- Regeln:
  - Addition modulo 2 = Xor = Subtraktion modulo 2
  - Multiplikation modulo 2 = And

A	B	A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

$\Leftrightarrow$

A	B	A - B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

And

  - Beispiel: $0 + (1 \cdot 0) + 1 + (1 \cdot 1) =$ 0

Polynomarithmetik modulo 2

- Betrachte Polynome über den Restklassenring Z_2
  - $p(x) = a_n x^n + \ldots + a_1 x^1 + a_0$
  - Koeffizienten a_i und Variable x sind aus $\in \{0,1\}$
  - Berechnung erfolgt modulo 2
- Addition, Subtraktion, Multiplikation, Division von Polynomen wie gehabt

$x + x = (1+1) \cdot x = 0$

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

$x^2 + 1 : x + 1 = x + 1$ Rest 0

$\underline{x^2 + x}$

$x + 1$

$x + 1$

Zeichenketten und Polynomarithmetik

$0x^3 + 1x^2 + 1x + 0 \cdot 1$ 0110

- Idee:
  - Betrachte Bitstring der Länge n als Variablen eines Polynoms
- Bit string: $b_n b_{n-1} \ldots b_1 b_0$

  Polynom: $b_n x^n + \ldots + b_1 x^1 + b_0$
  - Bitstring mit (n+1) Bits entspricht Polynom des Grads n
- Beispiel
  - A xor B = A(x) + B(x)
  - Wenn man A um k Stellen nach links verschiebt, entspricht das
    - B(x) = A(x) x^k
- Mit diesem Isomorphismus kann man Bitstrings dividieren

$1001 = 1 + x^3$

$000110 = x^2 + x$

$10 = x$

$(x^4 + x^2 + x + 1)$ x

$$\begin{array}{rr} & 1011 \\ + & 0010 \\ \hline & 1001 \end{array}$$

$$\begin{array}{r} x^3 + x + 1 \\ x \\ \hline x^3 \quad + 1 \end{array}$$

$10111 \cdot 10 =$

101110

10111 · 101001

10111
10111
10111

1001001111

$$
\begin{array}{l}
1000\,0 : 111 = 11 \\
\underline{111} \\
110 \\
\underline{111} \\
001
\end{array}
$$

G(x)

Rest $\boxed{001}$

CRC

Polynome zur Erzeugung von Redundanz: CRC

- Definiere ein Generatorpolynom *G(x)* von Grad *g*
  - Dem Empfänger und Sender bekannt
  - Wir erzeugen g redundante Bits
- Gegeben:
  - Frame (Nachricht) M, als Polynom M(x)
- Sender
  - Berechne den Rest der Division $r(x) = x^g M(x) \bmod G(x)$
  - Übertrage $T(x) = x^g M(x) + r(x)$
    - Beachte: $x^g M(x) + r(x)$ ist ein Vielfaches von G(x)
- Empfänger
  - Empfängt m(x)
  - Berechnet den Rest: m(x) mod G(x)

$1 \cdot x^8$

$x^g \cdot M(x) + r(x) \bmod G(x)$

$r(x)$

0

CRC Übertragung und Empfang

10110101 0

$+$ 00000 1 0 0 0

- Keine Fehler:
  - T(x) wird korrekt empfangen
- Bitfehler: T(x) hat veränderte Bits
  - Äquivalent zur Addition eines Fehlerpolynoms E(x)
  - Beim Empfänger kommt T(x) + E(x) an

$1 + x + x^2 + x^3 \ldots$

- Empfänger
  - Empfangen: m(x)
  - Berechnet Rest m(x) mod G(x)
  - Kein Fehler: m(x) = T(x),
    - dann ist der Rest 0
  - Bit errors: m(x) mod G(x) = (T(x)+ $\boxed{E(x)}$) mod G(x)

$$= \underbrace{T(x) \bmod G(x)}_{0} + \underbrace{\boxed{E(x)} \bmod G(x)}_{\text{Fehlerindikator}}$$

CoNe Freiburg

CRC – Überblick

Sender

Original Frame $M(x)$

Generator Polynom $G(X)$

$r(x) = x^g M(x) \bmod G(x)$

Sende $T(x) = x^g M(x) + r(x)$

Kanal

Addiert Fehlerpolynom $E(x)$

Kein Fehler:
$E(x) = 0$

Empfänger

Empfang: $m(x) = T(x) + E(x)$

Berechnet Rest $(T(x) + E(x)) \bmod G(x)$

Falls Rest = 0 : Kein Fehler

If Rest ≠ 0: Fehler!

UNI FREIBURG

b a

1 0 1 1 1 0 0 1 1 1 0 1 0 $= a + b \cdot x^n$

$b \cdot x^{n+1} + a$

$+ 1 \cdot x^n$

$b\,(x^{n+1} + x^n) + x^n$

Der Generator bestimmt die CRC-Eigenschaften

- Bit-Fehler werden nur übersehen, falls E(x) ein Vielfaches von G(x) ist
- Die Wahl von G(x) ist trickreich:
- Einzel-Bit-Fehler: $E(x) = x^i$ für Fehler an Position i
  - G(x) hat mindestens zwei Summenterme, dann ist E(x) kein Vielfaches von G(x) ist
- Zwei-Bit-Fehler: $E(x) = x^i + x^j = x^j (x^{i-j} +1)$ für i>j
  - G(x) darf nicht $(x^k + 1)$ teilen für alle k bis zur maximalen Frame-Länge
- Ungerade Anzahl von Fehlern:
  - E(x) hat nicht (x+1) als Faktor
  - Gute Idee (?): Wähle (x+1) als Faktor von G(x)
    - Dann ist E(x) kein Vielfaches von G(x)
- Bei guter Wahl von G(x):
  - kann jede Folge von r Fehlern erfolgreich erkannt werden
- Häufig:
  - G(x) wird als irreduzibles Polynom gewählt, dass heißt es ist kein Vielfache eines anderen (kleineren) Polynoms

CRC in der Praxis

$0\,g(x)$, $g(x)$, $g(x)\,x$, $g(x)+(x+1)$ ...

- Verwendetes irreduzibles Polynom gemäß IEEE 802:
  - $g(x) =$ $x^{32} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$
- Achtung:
  - Fehler sind immer noch möglich
  - Insbesondere wenn der Bitfehler ein Vielfaches von G(x) ist.
- Implementation:
  - Für jedes Polynom x^i wird $r(x,i)= x^i \bmod G(x)$ berechnet
  - Ergebnis von $B(x) \bmod G(x)$ ergibt sich aus
  - $b_0\, r(x,0) + b_1\, r(x,1) + b_2\, r(x,2) + \ldots + b_{k-1}\, r(x,k-1)$
  - Einfache Xor-Operation

Systeme II

3. Die Datensicherungsschicht

Christian Schindelhauer
Technische Fakultät
Rechnernetze und Telematik
Albert-Ludwigs-Universität Freiburg