Systeme II

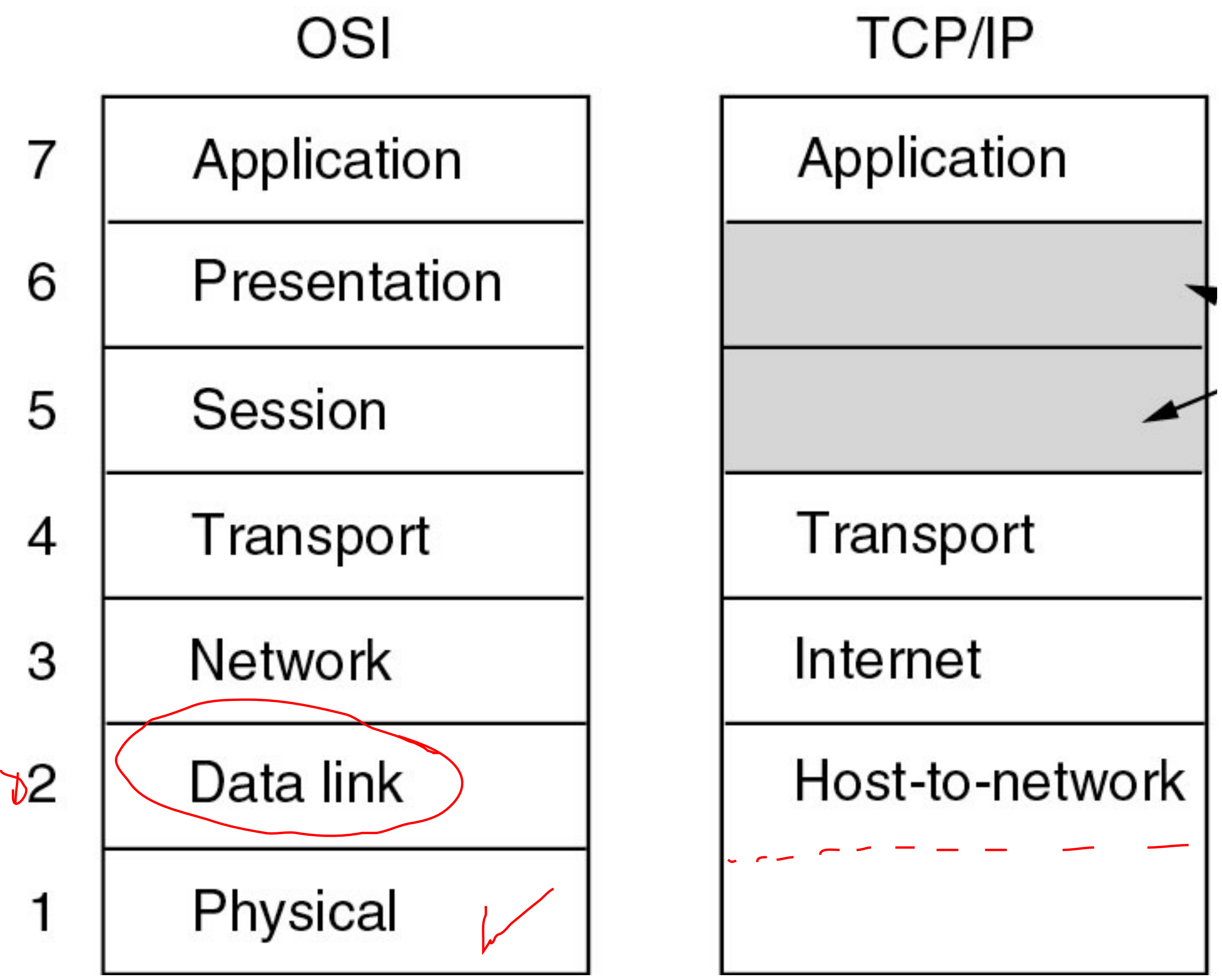
3. Die Datensicherungsschicht

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

Version 14.05.2013

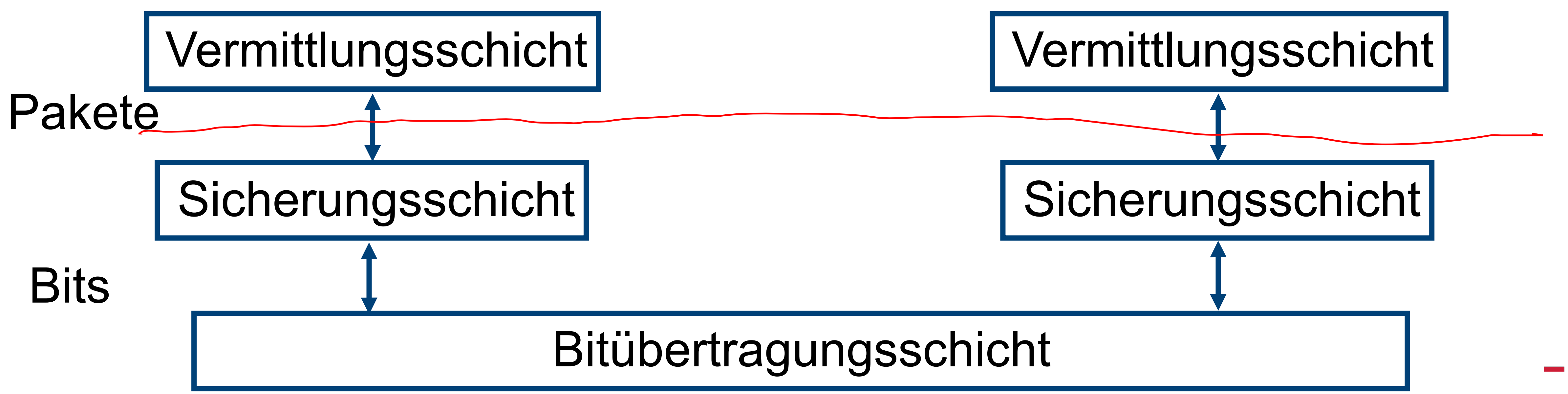
Die Sicherungsschicht

- Aufgaben der Sicherungsschicht (Data Link Layer)
  - Dienste für die Vermittlungsschicht
  - Frames
  - Fehlerkontrolle
  - Flusskontrolle



Dienste der Sicherungsschicht

- Situation der Sicherungsschicht
  - Die Bitübertragungsschicht überträgt Bits
  - Aber unstrukturiert und möglicherweise fehlerbehaftet
- Die Vermittlungsschicht erwartet von der Sicherungsschicht
  - Fehlerfreie Übermittlung
  - Übermittlung von strukturierten Daten
    - Datenpakete oder Datenströme
  - Störungslosen Datenfluss



Mögliche Dienste der Sicherungsschicht

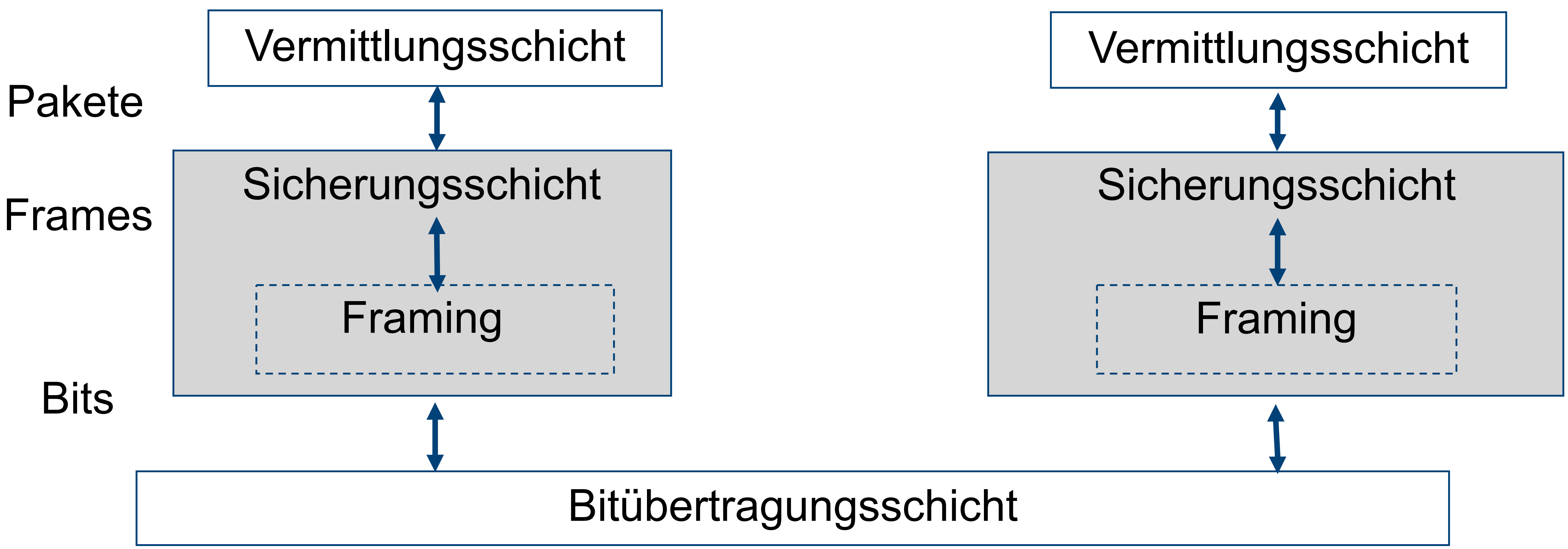
- Verlässlicher Dienst?
  - Das ausgelieferte und das empfangene Paket müssen identisch sein
  - Alle Pakete sollen (irgendwann) ankommen
  - Pakete sollen in der richtigen Reihenfolge ankommen
  - Fehlerkontrolle ist möglicherweise notwendig
- Verbindungsorientert?
  - Ist die Punkt-zu-Punktverbindung in einem größerem Kontext?
  - Reservierung der Verbindung notwendig?
- Pakete oder Datenströme (Bitströme)?

Unterscheidung: Dienst und Implementation

- Beispiel
  - Verbindungsloser und verlässlicher Dienst wird durch die Vermittlungsschicht gefordert
  - Sicherungsschicht verwendet intern verbindungsorientierten Dienst mit Fehlerkontrolle
- Andere Kombinationen sind möglich

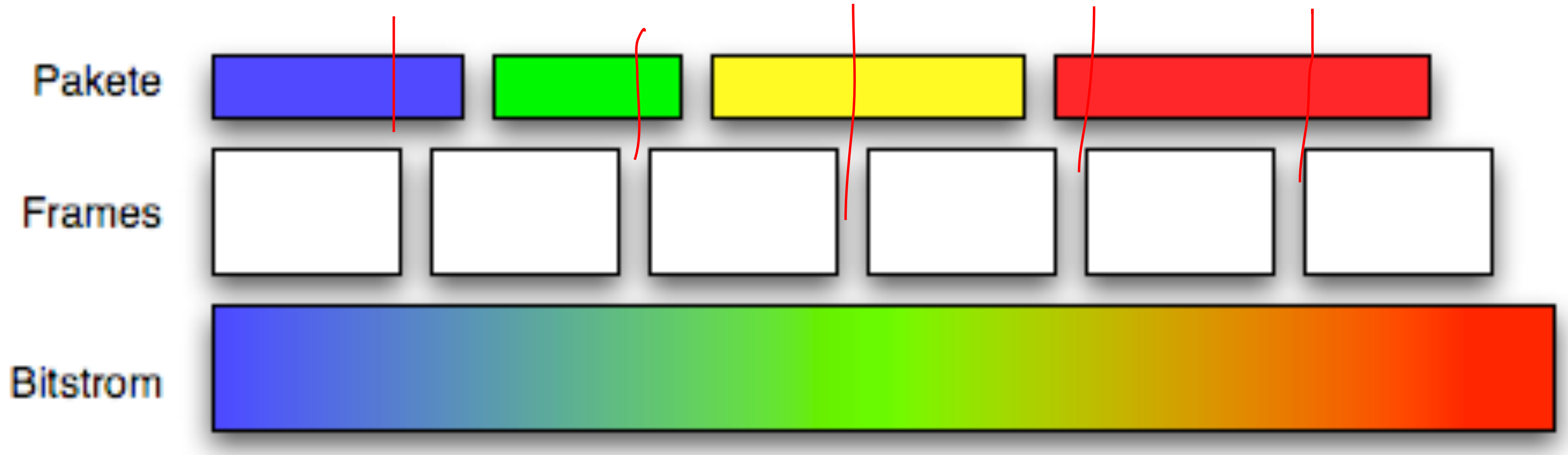
Frames

- Der Bitstrom der Bitübertragungsschicht wird in kleinere “Frames” unterteilt
  - Notwendig zur Fehlerkontrolle
  - Frames sind Pakete der Sicherungsschicht
- Frame-Unterteilung (Fragmentierung) und Defragmentierung sind notwendig
  - Falls die Pakete der Vermittlungsschicht größer sind als die Frames



Frames

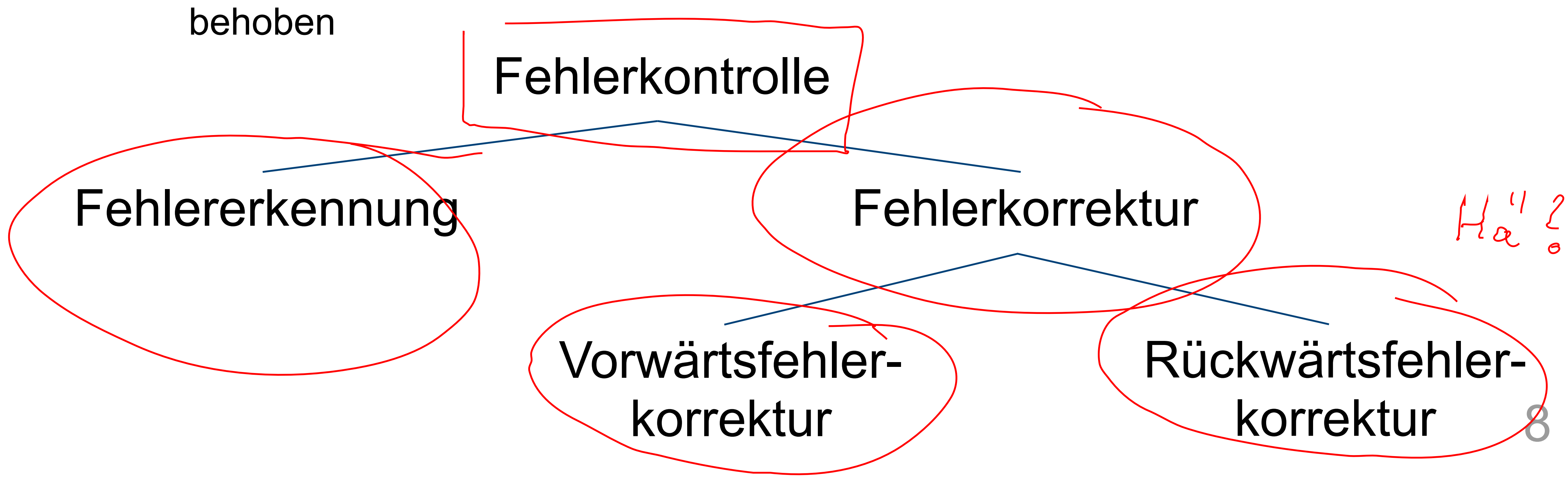
- Die Sicherungsschicht zwischen der Bitübertragungsschicht mit Bitstrom und der Vermittlungsschicht mit Paketen



- Pakete werden in Framegröße fragmentiert

Fehlerkontrolle

- 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



Verbindungsaufbau

- Nutzen von Verbindungen
  - Kontrolle des Verbindungsstatus
    - Korrektheit des Protokolls
  - Fehlerkontrolle
    - Verschiedene Fehlerkontrollverfahren vertrauen auf gemeinsamen Kontext von Sender und Empfänger
- Aufbau und Terminierung von Verbindungen
  - "Virtuelle Verbindungen"
    - Es werden keine Schalter umgelegt
    - Interpretation des Bitstroms
  - Kontrollinformationen in Frames
  - Besonders wichtig bei drahtlosen Medien
- Das Problem wird im Rahmen der Transportschicht ausführlich diskutiert
  - Vgl. Sitzungsschicht vom OSI-Modell

Flusskontrolle

- Problem: Schneller Sender und langsamer Empfänger
  - Der Sender lässt den Empfangspuffer des Empfängers überlaufen
  - Übertragungsbandweite wird durch sinnlosen Mehrfachversand (nach Fehlerkontrolle) verschwendet
- Anpassung der Frame-Sende-Rate an dem Empfänger notwendig

Langsamer Empfänger

Schneller Sender

Frames

- Wo fängt der Frame an und wo hört er auf?
- Achtung:
  - Die Bitübertragungsschicht kann auch Bits liefern, wenn der Sender tatsächlich nichts sendet
  - Der Empfänger
    - könnte das Rauschen auf dem Medium interpretieren
    - könnte die Folge 00000000.... liefern
  - Daten oder Kontrollinformation?

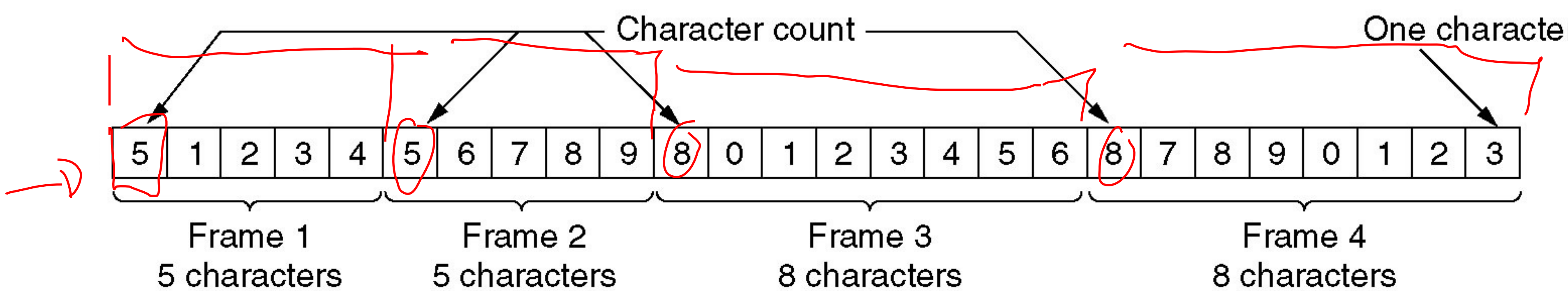
0000000 000000 000000 1

Übertragener Bitstrom 0110010101110101110010100010101010101010101100010

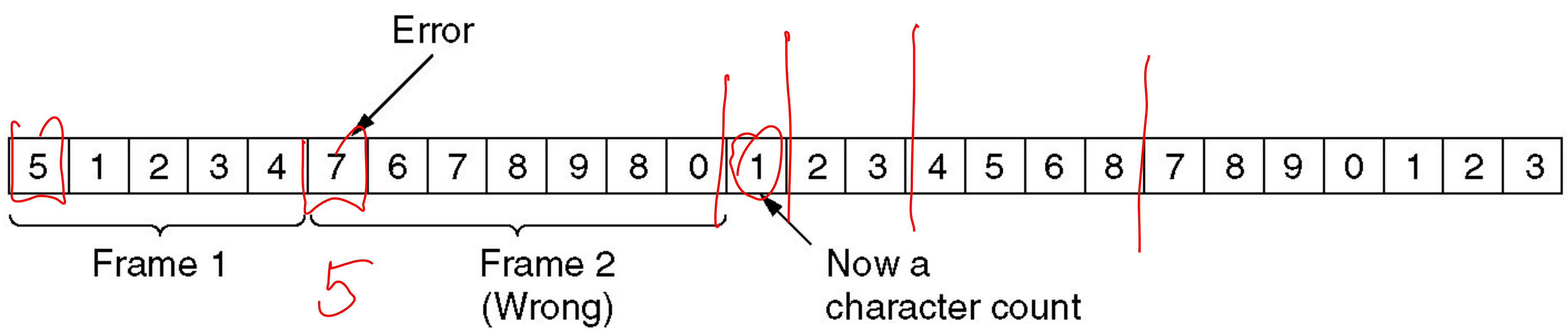
Frame-Anfang? Frame-Ende?

Frame-Grenzen durch Paketlängen?

- Idee: Ankündigung der Bitanzahl im Frame-Header

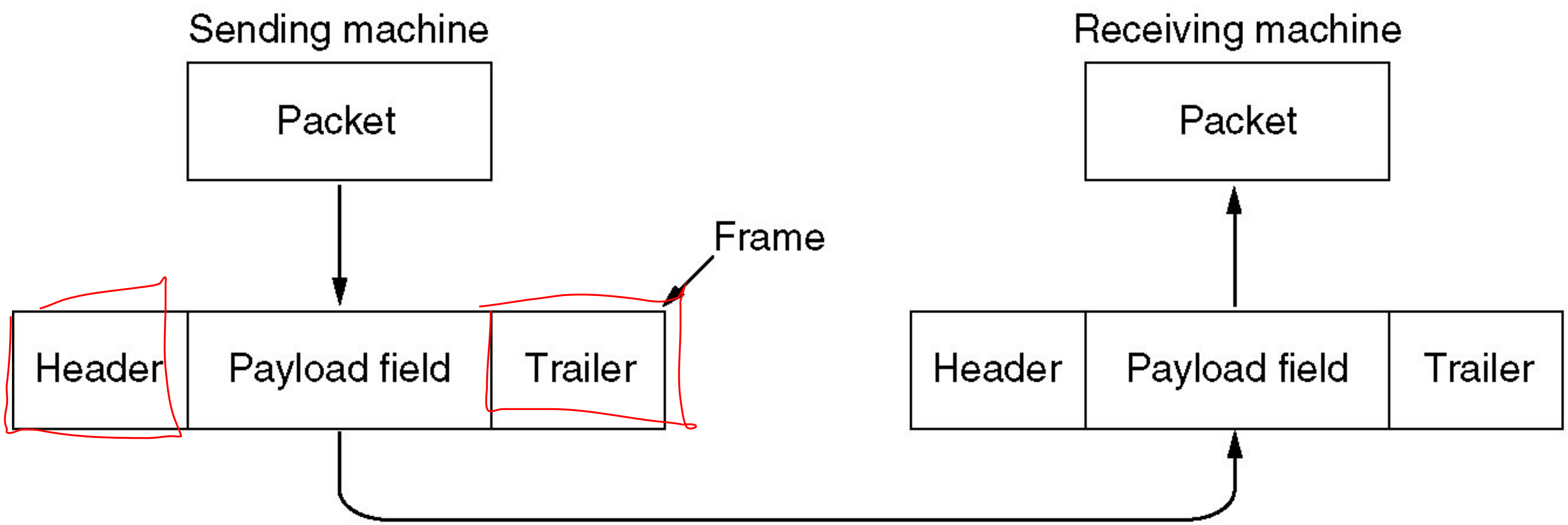


- Problem: Was, wenn die Frame-Länge fehlerhaft übertragen wird?
  - Der Empfänger kommt aus dem Takt und interpretiert neue, sinnlose Frames
  - Variable Frame-Größen mit Längeninformation sind daher kein gutes Konzept



Header und Trailer

- Header und Trailer
  - Zumeist verwendet man Header am Anfang des Frames, mitunter auch Trailer am Ende des Frames
  - signalisieren den Frame-Beginn und das Frame-Ende
  - tragen Kontrollinformationen
    - z.B. Sender, Empfänger, Frametypen, Fehlerkontrollinformation



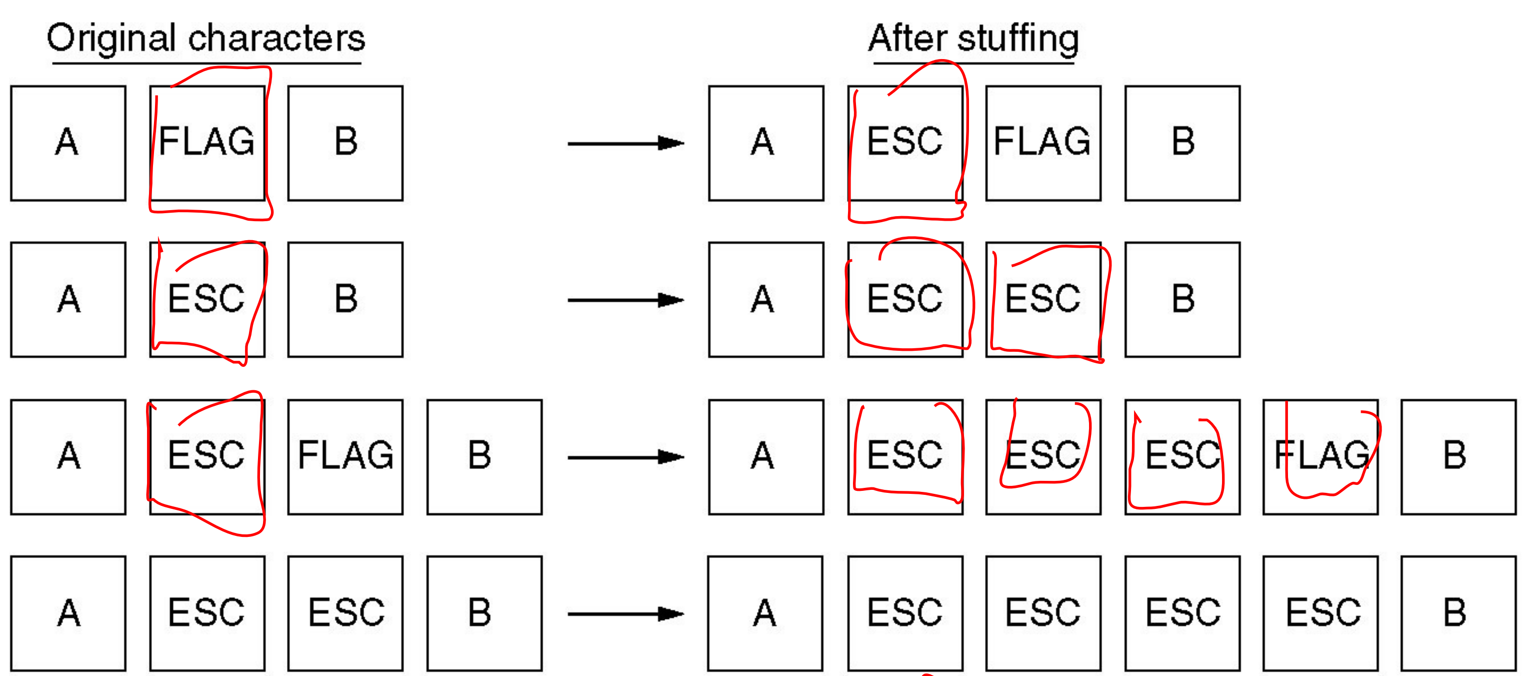
Flag Bytes und Bytestopfen

- Besondere “Flag Bytes” markieren Anfang und Ende eines Frames

FLAG	Header	Payload field	Trailer	FLAG

- Falls diese Marker in den Nutzdaten vorkommen
  - Als Nutzdatenbyte mit Sonderzeichen (Escape) markieren
    - Bytestopfen (byte stuffing)
  - Falls Sonderzeichen und “Flag-Byte” erscheinen, dito,
    - etc., etc.

$$\frac{1}{128} \cdot 2 + \frac{127}{128} \cdot 1 = 1{,}009$$



Frames durch Bit-Sequenzen/Bitstopfen

- Bytestopfen verwendet das Byte als elementare Einheit
  - Das Verfahren funktioniert aber auch auf Bitebene
- Flag Bits und Bitstopfen (bit stuffing)
  - Statt flag byte wird eine Bit-Folge verwendet
    - z.B.: 01111110
  - Bitstopfen
    - Wenn der Sender eine Folge von fünf 1er senden möchte, wird automatisch eine 0 in den Bitstrom eingefügt
      - Außer bei den Flag Bits
- Der Empfänger entfernt eine 0 nach fünf 1ern

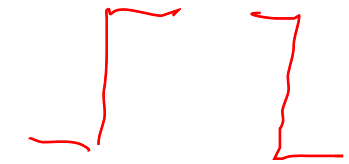
Originale Nutzdate	(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
Nach dem Bitstopfen	(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0
	Stuffed bits
Nach der "Entstopfung"	(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

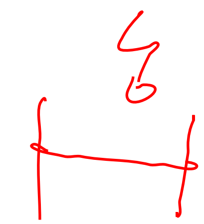
$\frac{n}{5}$ + 20%

1111
1111
011110
011110

11111 0 → 1111100
11111 1 → 1111101

Frames durch Code-Verletzung





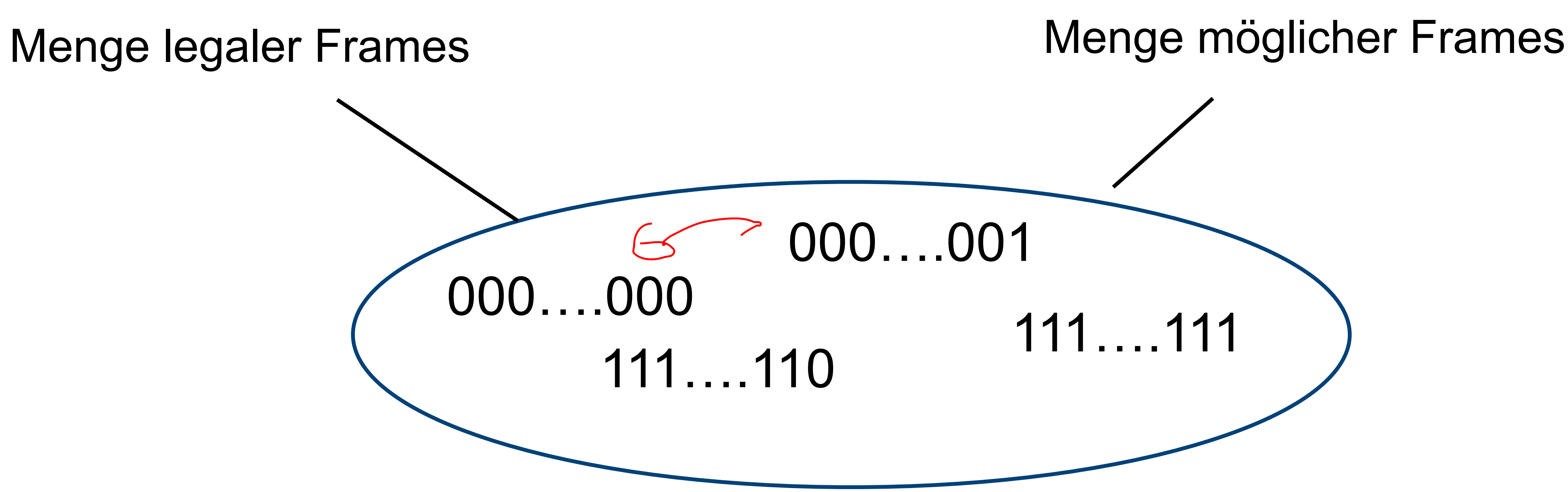
- Möglicher Spielraum bei Bitübertragungsschicht bei der Kodierung von Bits auf Signale
  - Nicht alle möglichen Kombination werden zur Kodierung verwendet
  - Zum Beispiel: Manchester-Kodierung hat nur tief/hoch und hoch/tief–Übergang → Ethernet
- Durch “Verletzung” der Kodierungsregeln kann man Start und Ende des Rahmens signalisieren
  - Beispiel: Manchester – Hinzunahme von hoch/hoch oder tief/tief
    - Selbsttaktung von Manchester gefährdet?
- Einfache und robuste Methode
  - z.B. verwendet in Ethernet
  - Kosten? Effiziente Verwendung der Bandbreite?

Fehlerkontrolle

- Aufgaben
  - Erkennung von Fehlern (fehlerhafte Bits) in einem Frame
  - Korrektur von Fehlern in einem Frame
- Jede Kombination dieser Aufgaben kommt vor
  - Erkennung ohne Korrektur
    - Löschen eines Frames ohne weiter Benachrichtigung (drop a frame)
    - Höhere Schichten müssen sich um das Problem kümmern
  - Korrektur ohne Erkennung
    - Es werden bestmöglich Bitfehler beseitigt, möglicherweise sind aber noch Fehler vorhanden
    - Sinnvoll, falls Anwendung Fehler tolerieren kann
      - Beispiel: Tonübertragung
    - Prinzipiell gerechtfertigt, weil immer eine positive Restfehlerwahrscheinlichkeit bleibt

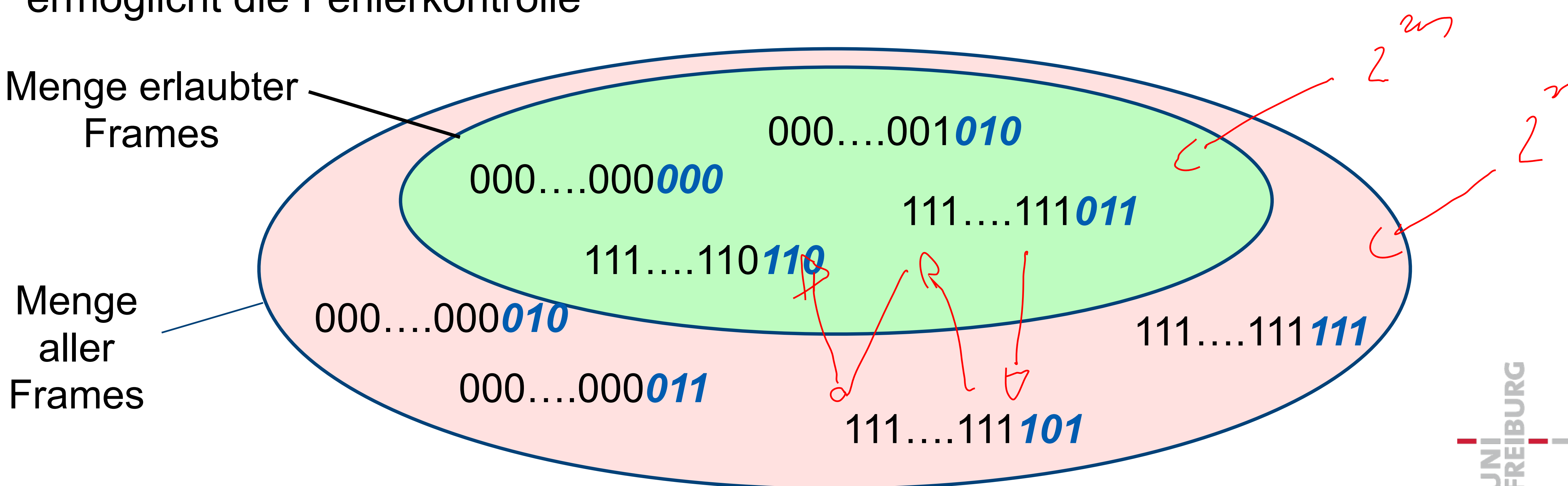
Redundanz

- Redundanz ist eine Voraussetzung für Fehlerkontrolle
- Ohne Redundanz
  - Ein Frame der Länge m kann ~~2m~~ 2^m mögliche Daten repräsentieren
  - Jede davon ist erlaubt
- Ein fehlerhaftes Bit ergibt einen neuen Dateninhalt



Redundanz

- Kernidee:
  - Einige der möglichen Nachrichten sind verboten
  - Um dann 2^m legale Frames darzustellen
    - werden mehr als 2^m mögliche Frames benötigt
    - Also werden mehr als m Bits in einem Frame benötigt
  - Der Frame hat also Länge n > m
  - r = m - n sind die redundanten Bits
    - z.B. Im Header oder Trailer
- Nur die Einschränkung auf erlaubte und verbotene (legal/illegal) Frames ermöglicht die Fehlerkontrolle



Einfachste Redundanz: Das Paritätsbit

Odd 1 1 1 ← $C = \overline{x_1 \oplus x_2}$

EVEN 1 1 0

- Eine einfache Regel um ein redundantes Bit zu erzeugen
(d.h. n=m+1)

$C = x_1 \oplus x_2$

- Parität
  - Odd parity
    - Eine Eins wird hinzugefügt, so dass die Anzahl der 1er in der Nachricht ungerade wird (ansonsten eine Null)
  - Even parity
    - Eine Eins wird hinzugefügt, so dass die Anzahl der 1er in der Nachricht gerade wird (ansonsten wird eine Null hinzugefügt)

1 1 0

1 0 1

1 1 1

= 0

- Beispiel:
  - Originalnachricht ohne Redundanz: 01101011001
  - Odd parity: 011010110011
  - Even parity: 011010110010

0 0 0

1 1 0

1 0 1

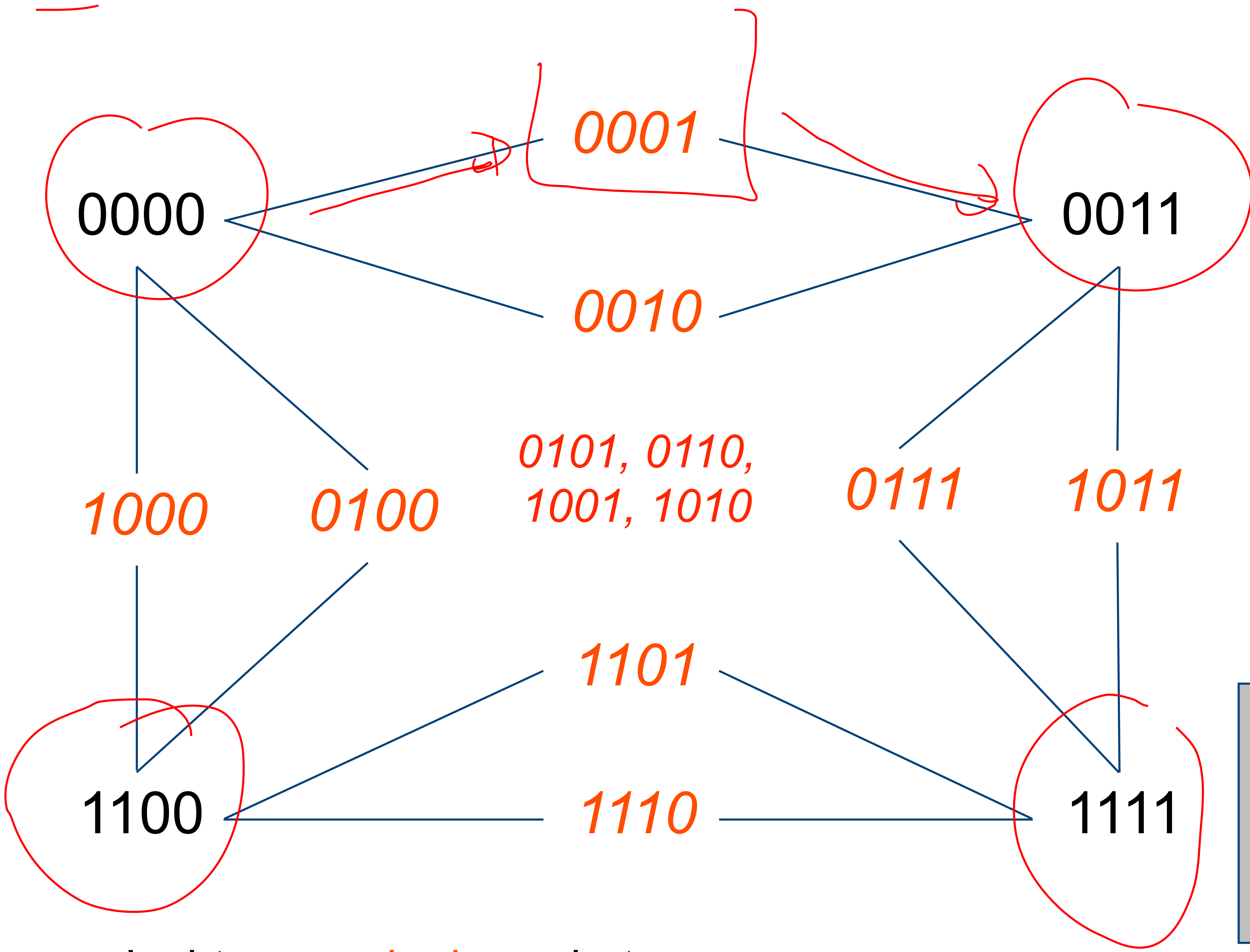
0 1 1

Der Nutzen illegaler Frames

- Der Sender sendet nur erlaubte Frames
- In der Bitübertragungsschicht könnten Bits verfälscht werden
- Hoffnung:
  - Legale Frames werden nur in illegale Nachrichten verfälscht
  - Und niemals ein legaler Frame in einen anderen Legalen
- Notwendige Annahme
  - In der Bitübetragungsschicht werden nur eine bestimmte Anzahl von Bits verändert
    - z.B. k Bits pro Frame
  - Die legalen Nachrichten sind verschieden genug, um diese Frame-Fehlerrate zu erkennen

Veränderung der Frames durch Bitfehler

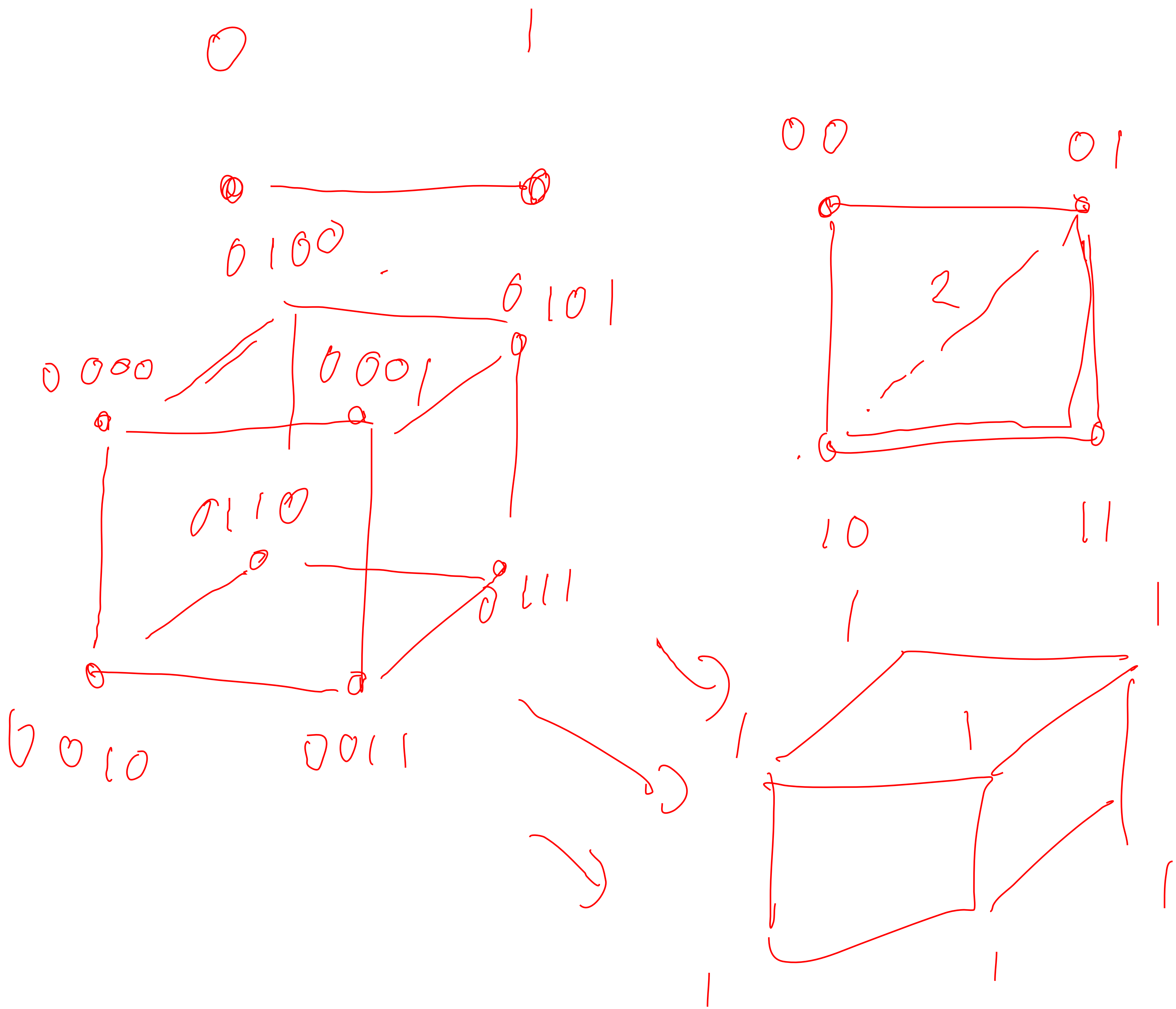
- Angenommen die folgenden Frames sind erlaubt: 0000, 0011, 1100, 1111



Kanten verbinden Frames, die sich nur in einem Bit unterscheiden

Ein einfacher Bitfehler kann erlaubte Frames nicht in einen anderen erlaubten Frame umformen!

uvxy – erlaubt *abcd* – verboten



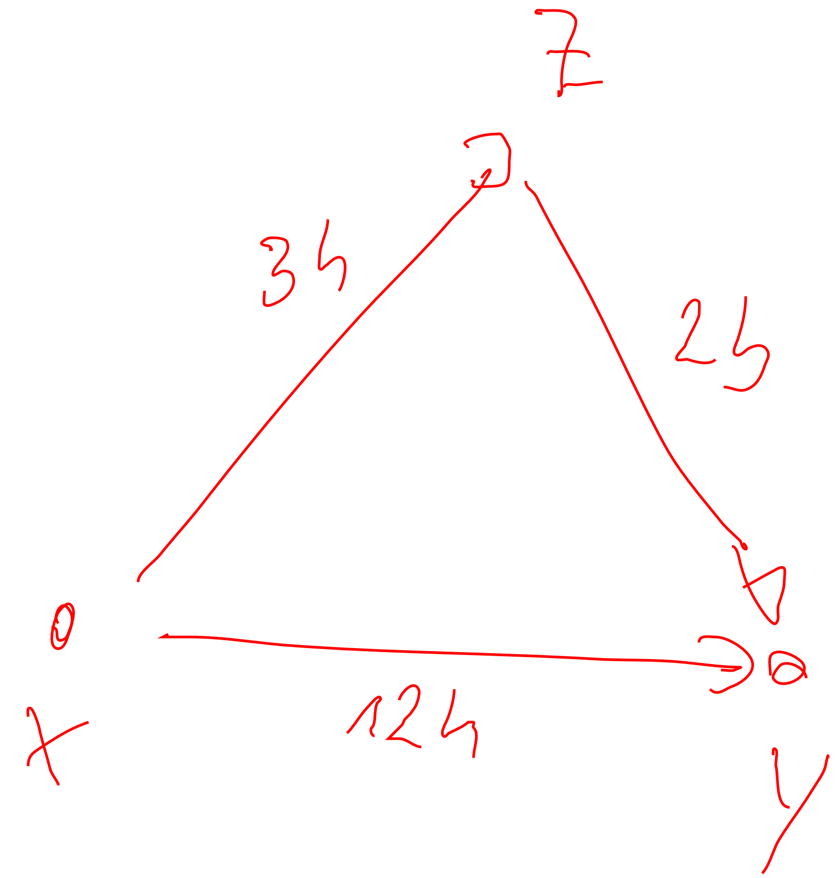
0
1
0100
0101
0000
0001
0110
0111
0010
0011
00
01
10
11
2

Hamming-Distanz

- Der "Abstand" der erlaubten Nachrichten zueinander war immer zwei Bits
- Definition: Hamming-Distanz
  - Seien $x = x_1, \ldots, x_n$ und $y = y_1, \ldots, y_n$ Nachrichten
  - Dann sei $d(x,y)$ = die Anzahl der 1er Bits in x XOR y
- Intuitiver: die Anzahl der Positionen, in denen sich x und y unterscheiden

Hamming-Distanz

- Die Hamming-Distanz ist eine Metrik
  - Symmetrie
    - d(x,y) = d(y,x)
  - Dreiecksungleichung:
    - d(x,y) ≤ d(x,z) + d(z,y)
  - Identität
    - d(x,x) = 0 und
      d(x,y) = 0 gdw. x = y



- Beispiel:
  - x=   0011010111
  - y=   0110100101
  - x XOR y=   0101110010
  - d(x,y) = 5

Hamming-Distanz von Nachrichtenmengen

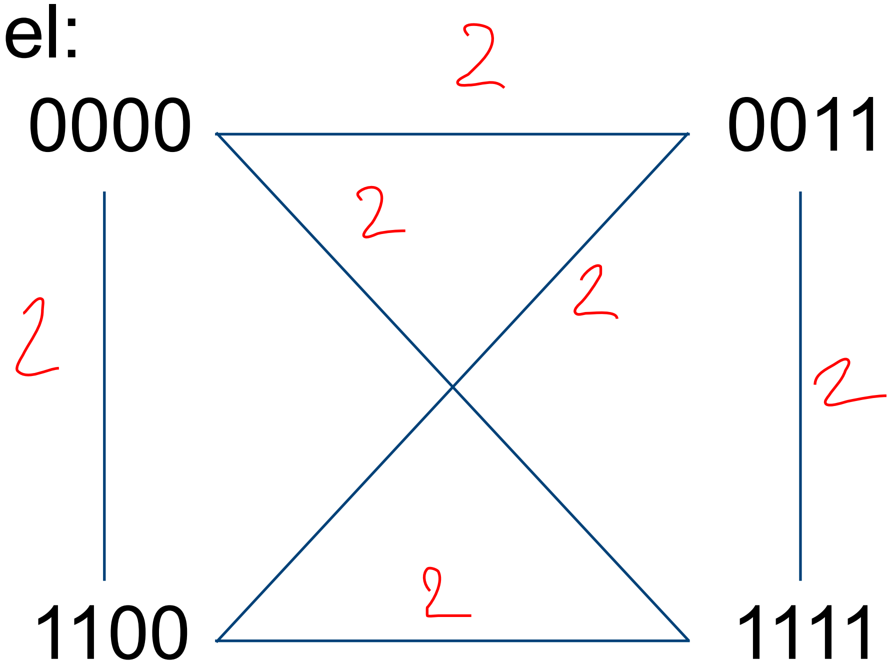
000000
16
111111

- Die Hamming-Distanz einer Menge von (gleich langen) Bit-Strings S ist:

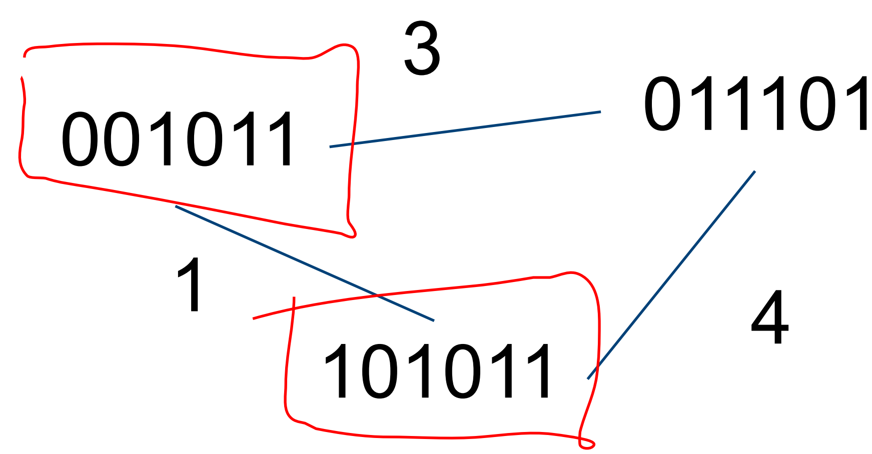
$$d(S) = \min_{x,y \in S, x \neq y} d(x,y)$$

  - d.h. der kleinste Abstand zweier verschiedener Wörter in S

Beispiel:



Alle Abstände sind 2



Ein Abstand ist 1!

CoNe Freiburg

Erkennung und Korrektur mit Hamming-Distanzen

011
010
110
2

- 1. Fall d(S) = 1
  - Keine Fehlerkorrektur
  - Legale Frames unterscheiden sich in nur einem Bit
- 2. Fall d(S) = 2
  - Dann gibt es nur x, y ∈ S mit d(x,y) = 2
  - Somit ist jedes u mit d(x,u) = 1 illegal,
    - wie auch jedes u mit d(y,u) = 1

x —— 1 Bit Unterschied —— u —— 1 Bit Unterschied —— y

  - 1-Bit-Fehler
    - können immer erkannt werden
    - aber nicht korrigiert werden

CoNe Freiburg

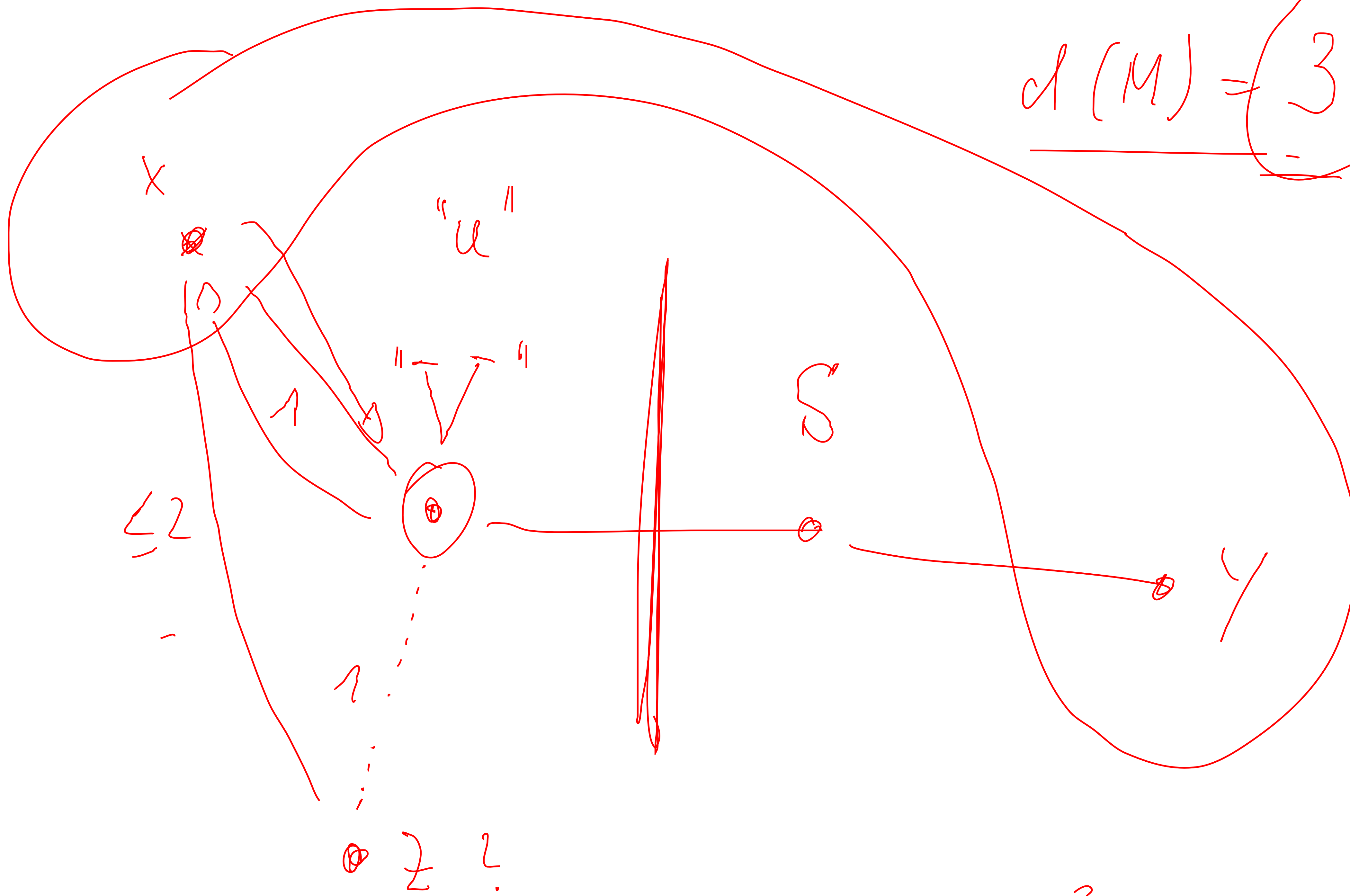
Erkennung und Korrektur mit Hamming-Distanzen

- 3. Fall d(S) = 3
  - Dann gibt es nur $x, y \in S$ mit $d(x,y) = 3$
  - Jedes u mit $d(x,u) = 1$ illegal und $d(y,u) > 1$

x —— 1 Bit Unterschied —— u —— 1 Bit Unterschied —— s —— 1 Bit Unterschied —— y

  - Falls u empfangen wird, sind folgende Fälle denkbar:
    - x wurde gesendet und mit 1 Bit-Fehler empfangen
    - y wurde gesendet und mit 2 Bit-Fehlern empfangen
    - Etwas anderes wurde gesendet und mit mindestens 2 Bit-Fehlern empfangen
  - Es ist also wahrscheinlicher, dass x gesendet wurde, statt y

$d(M) = 3$



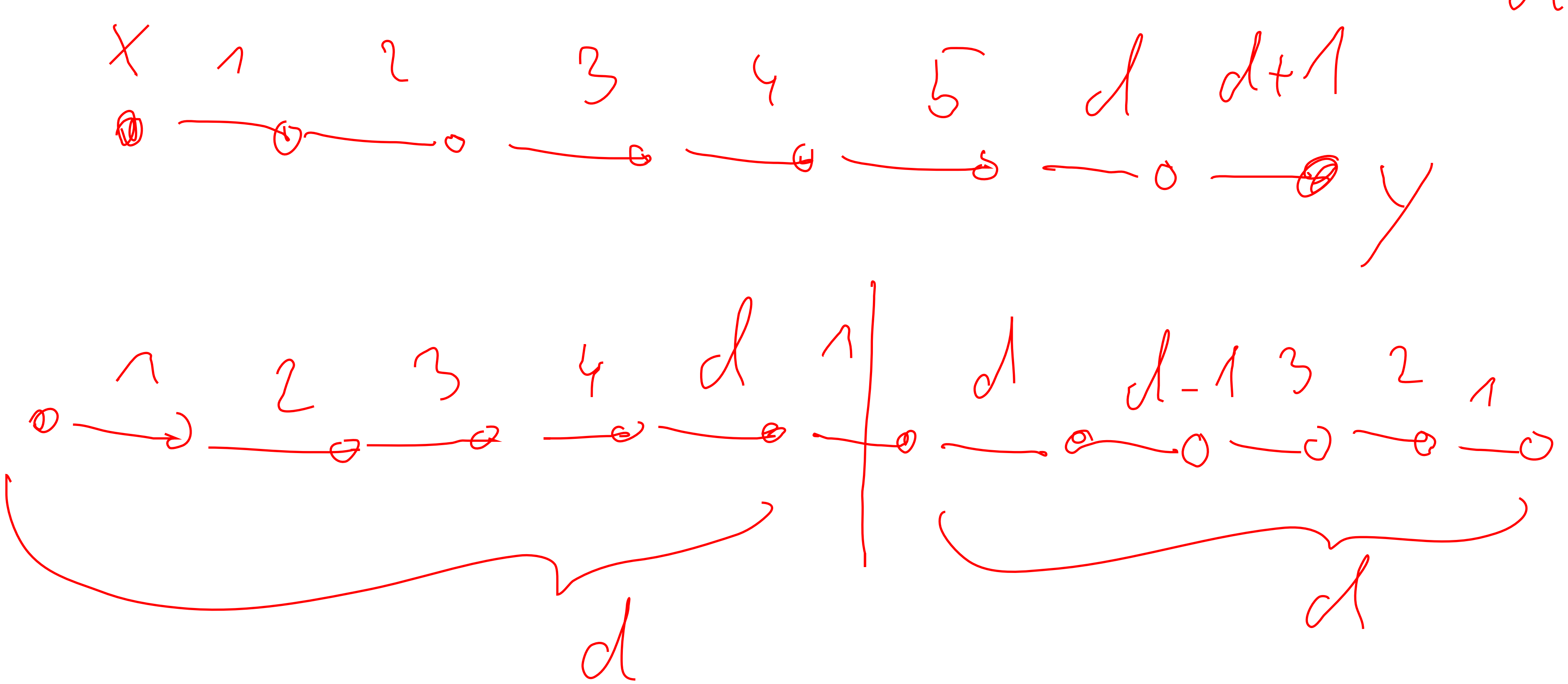
$z \notin M$?

CoNe Freiburg

Erkennung und Korrektur mit Hamming-Distanzen

- Um d Bit-Fehler zu erkennen ist eine Hamming-Distanz von d+1 in der Menge der legalen Frames notwendig
- Um d Bit-Fehler zu korrigieren, ist eine Hamming-Distanz von 2d+1 in der Menge der legalen Frames notwendig

$d(S) = d+1$



$d(S) = 2d+1$

UNI FREIBURG

Codebücher und Kodierungen

- Die Menge der legalen Frames $S \in \{0,1\}^n$ wird **das Code-Buch** oder einfach Kodierung genannt.
  - Die Rate R eines Codes S ist definiert als
    - Die Rate charakterisiert die Effizienz des Codes

$$R_S = \frac{\log |S|}{n}$$

$s_1 \leftrightarrow 000$

$s_2 \quad 001$

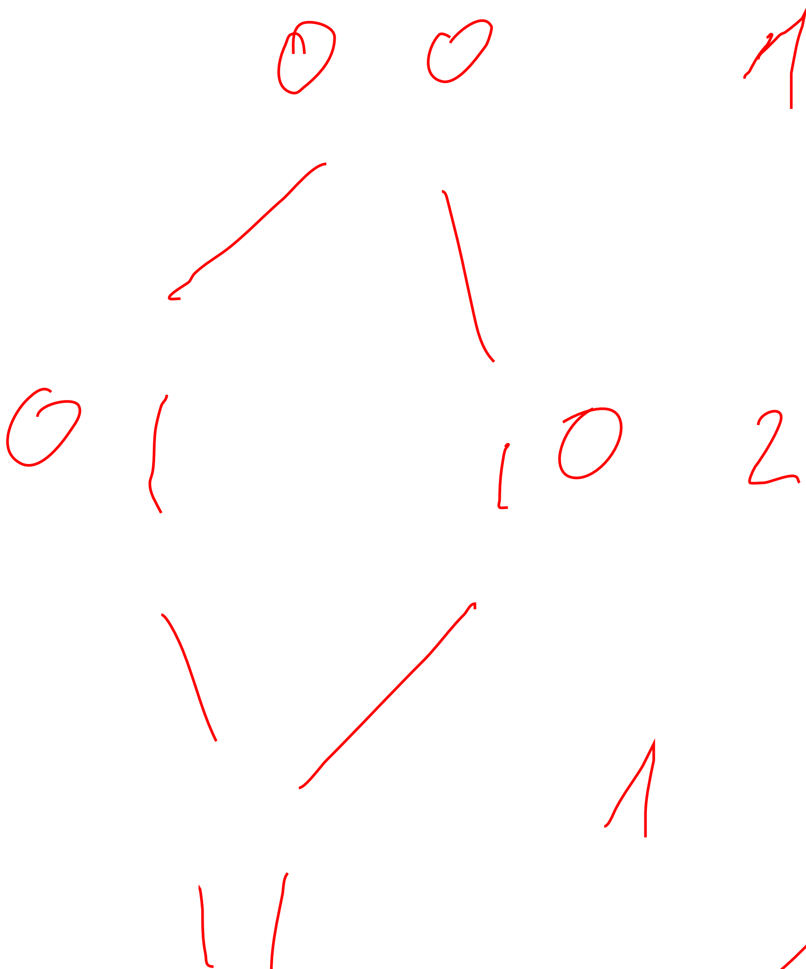
$s_3 \quad 010$

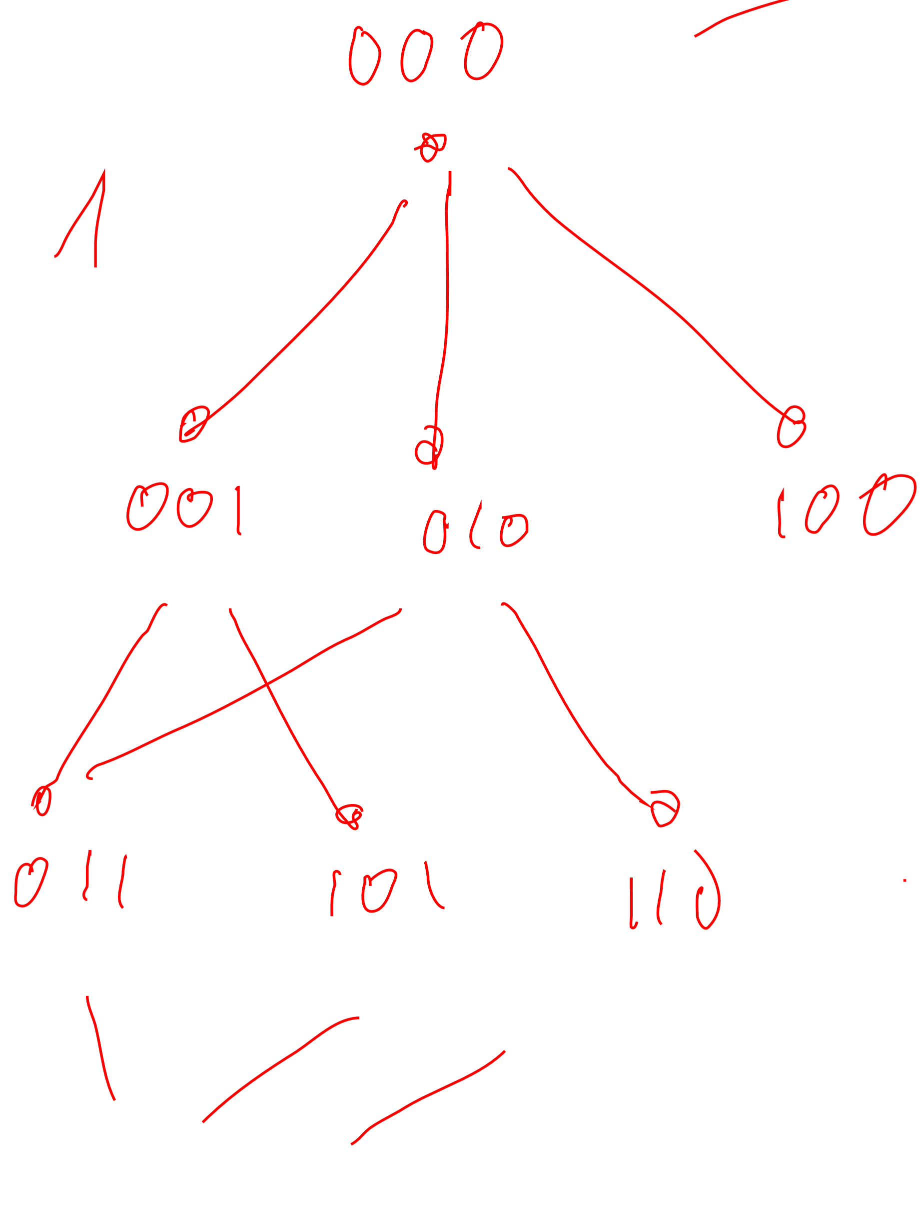
$\log |S|$

  - Die Distanz δ des Codes S ist definiert als
    - charakterisiert die Fehlerkorrektur oder Fehlererkennungsmöglichkeiten

$$\delta_S = \frac{d(S)}{n}$$

- Gute Codes haben hohe Raten und hohe Distanz
  - Beides lässt sich nicht zugleich optimieren





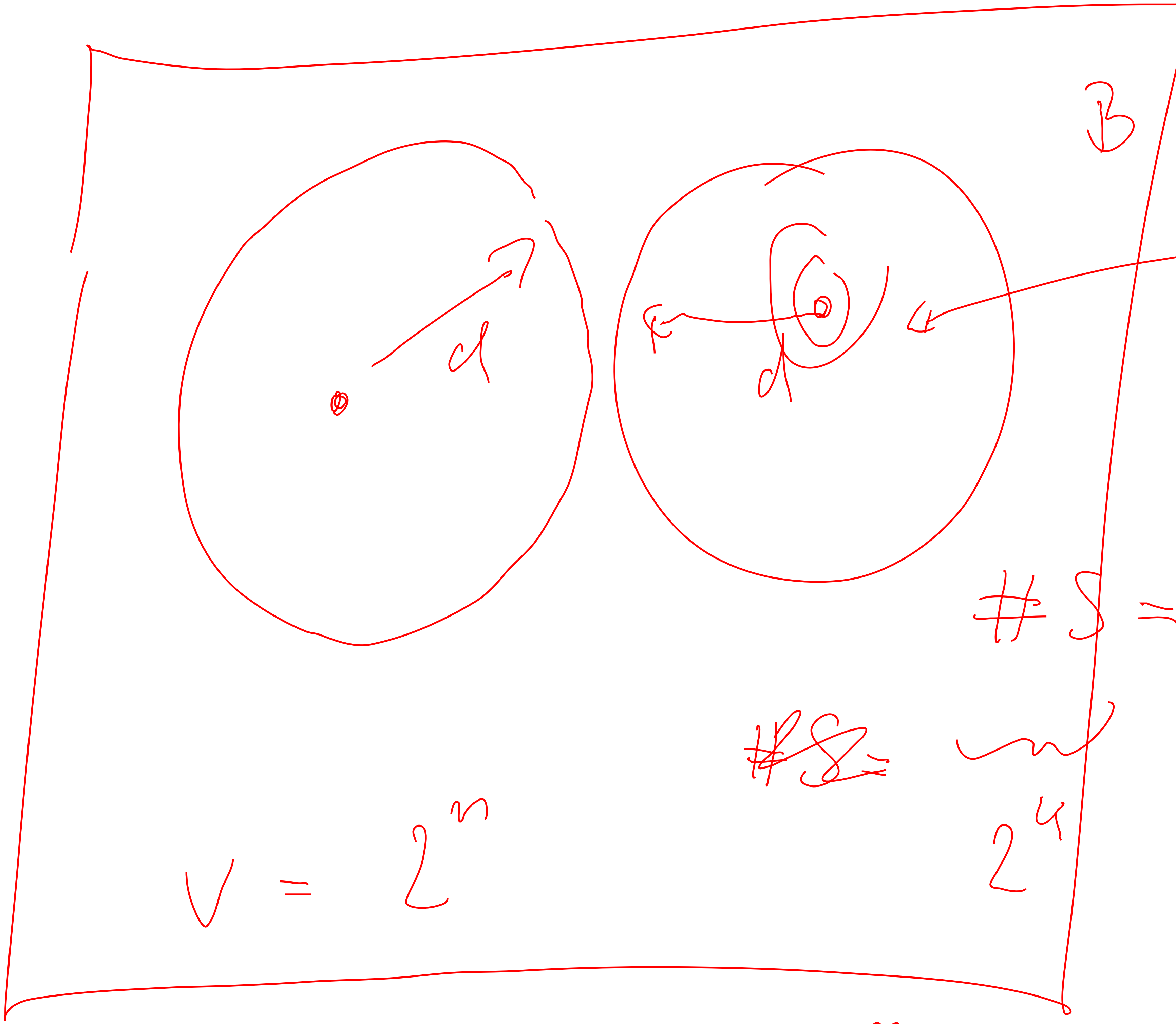
$n \to$	1	2	3	4	5	6	7
	1	1	1	1	1	1	
$\to$	1	2	3	4	5	6	
		1	3	6	10	15	
			1	4	10	20	
				1	5	15	
			1		1	6	
						1	

3

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

3

1



$$\geq \sum_{i=0}^{d} \binom{n}{i}$$

$$\#S = \frac{V}{B} \qquad V \leftarrow 2^n$$

$$2^{n-k} = \frac{2^n}{2^k} = B = \sum_{i=0}^{d} \binom{n}{i}$$

Block-Codes

- Block-Codes kodieren k Bits Originaldaten in n kodierte Bits
  - Zusätzlich werden n-k Symbole hinzugefügt
  - Binäre Block-Codes können höchstens bis zu t Fehler in einem Code-Wort der Länge n mit k Originalbits erkennen, wobei (Gilbert-Varshamov-Schranke):

$$2^{n-k} \geq \sum_{i=0}^{t} \binom{n}{i}$$

  - Das ist eine theoretische obere Schranke
- Beispiele
  - Bose Chaudhuri Hocquenghem (BCH) Codes
    - basierend auf Polynomen über endlichen Körpern (Galois-Körpern)
  - Reed Solomon Codes
    - Spezialfall nichtbinärer BCH-Codes