Daten und Information

Repräsentation von Daten

- Nicht alle Daten, die interessant sind, sind von Natur aus als Bitfolgen repräsentiert.
- Wir haben gerade ein Beispiel gesehen: ein Programm in einer Höheren Programmiersprache liegt als Text vor.
- Bislang haben wir aber nur davon gesprochen, wie man Bitfolgen abspeichern kann, z.B. in Registern.
  - Teilweise interpretiert als Binär- oder als Zweierkomplementzahl
- Ansatz: einfach alle Buchstaben durchnummerieren.

Beispiel

- $A \mapsto 1$
- $B \mapsto 2$
- $C \mapsto 3$
- D …
- …

H	A	L	L	O
↓	↓	↓	↓	↓
8	1	12	12	15
↓	↓	↓	↓	↓
1000	0001	1100	1100	1111

Code / Codierung / Dekodierung

- **Definition.** Sei A eine Menge. Eine injektive Abbildung $c: A \to \{0,1\}^+$ heißt *(binärer) Code* oder *Codierung* für A. Die Umkehrfunktion $d = c^{-1}$ heißt *Dekodierung* nach A.
- Eine Decodierung ist i.A. eine partielle Funktion, d.h. nicht jede mögliche Bitfolge in $\{0,1\}^+$ kann auch einem Element in A zugewiesen werden.
- Es gilt für alle $a \in A$: $d\big(c(a)\big) = a$
  - Das heißt, wenn ich ein Element aus A zuerst mit c codiere und anschließend wieder mit d dekodiere, erhalte ich wieder das ursprüngliche Element.

ASCII: American Standard Code for Information Interchange

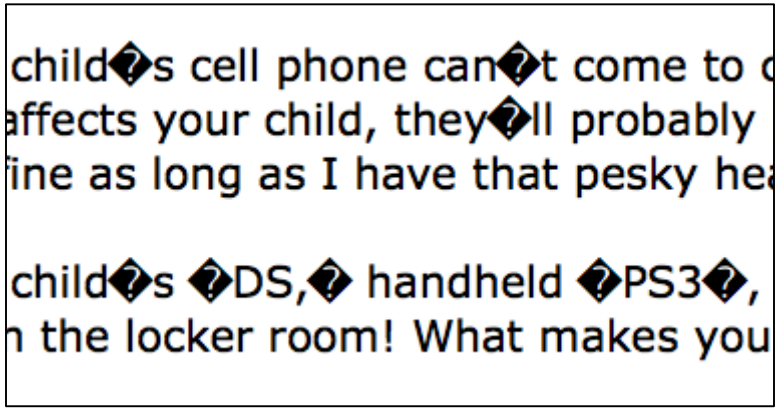
Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	“	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	‘	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Unterschiedliche Character Encodings

- Warum reicht ASCII nicht?
- Viele Sprachen verwenden weitere oder ganz andere Symbole
- Unicode Standard
  - Enthält aktuell (Version 15.0) 1.114.112 Zeichen
  - Definiert verschiedene Codierungen für Teilmengen dieser Zeichen
  - Beispiel: UTF-8
    - Interessant: manche Zeichen werden mit einem Byte codiert, andere mit 2, 3 oder 4.

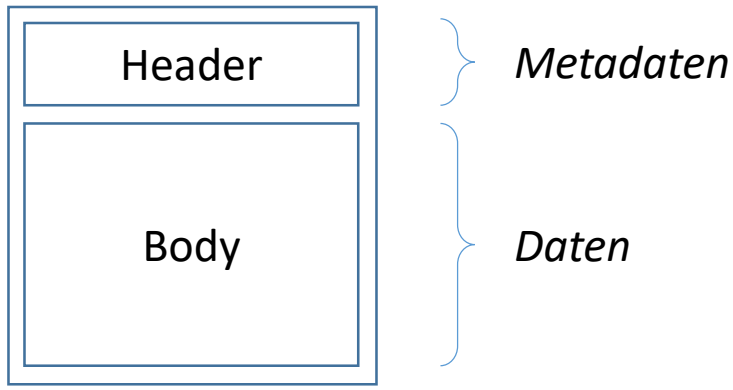
Welche Codierung wurde verwendet?

- Für jede Menge A kann es natürlich mehr als eine Codierung geben.
- Woher weiß man dann, welchen Dekodierer man zum Herstellen des Ursprungselements verwenden muss?



Metadaten

- Im Allgemeinen sieht man einer Bitfolge nicht an, von welcher Codierung sie erzeugt wurde.
- Diese Information muss daher gesondert gespeichert werden.
- Dazu gibt es in der Praxis verschiedene Möglichkeiten, z.B.:
  - Festlegung in einem Standard
  - „Header“-Sektion in standardisiertem Dateiformat, z.B.:



Metadaten 2

- Neben der Codierung gibt es bei Texten aber noch andere Arten von Metadaten, die nützlich sind, z.B:
  - Semantische Informationen
    - Welcher Teil des Textes ist eine Überschrift? Wo hört ein Abschnitt auf? etc.
  - Typographische Informationen
    - Font, Fettdruck, Kursivsatz, etc.
  - Layoutinformationen
    - Wie breit ist der Rand?
- Bei Dokumenten, die eine Kombination mehrerer Medien (Texte, Bilder, Videos, etc.) erlauben, sind solche Informationen zur korrekten Darstellung um so wichtiger.

Codierung anderer Medien

- Texte haben den Vorteil, dass sie inhärent aus Folgen von diskreten Zeichen aufgebaut sind.
  - Die Codierung eines Textes kann dann als Folge codierter Zeichen realisiert werden
- Andere Medien bestehen aber nicht per-se aus diskreten Einheiten, sondern sind kontinuierlich:

räumlich

zeitlich

Diskretisierung

- Ansatz: unterteile das Medium künstlich in diskrete Einheiten
- Dabei geht zwangsläufig Information verloren.
- Je feiner die Auflösung, desto weniger Information geht verloren
- Beispiele:
  - Bilder → Pixel („picture elements“)
  - Audio → Sample

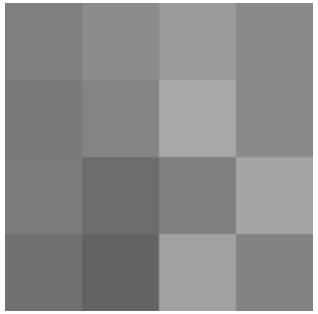
Beispiel: Pixelauflösung

Originalfoto

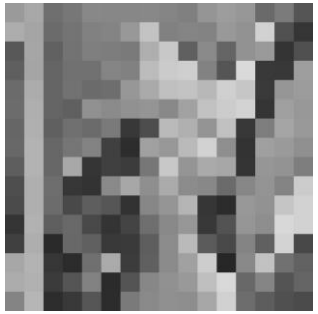
Unterteilung in diskrete Einheiten

Repräsentation jedes Quadrats durch Mittelwert

4x4 Pixel

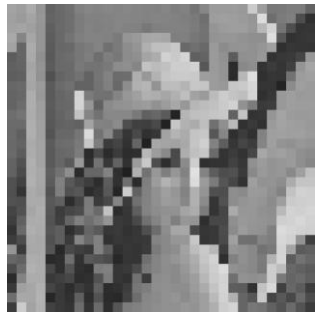


16x16 Pixel



128x128 Pixel

64x64 Pixel

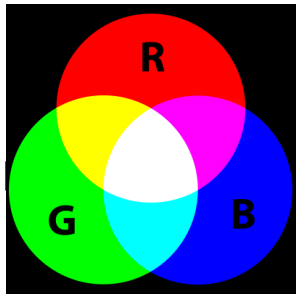


32x32 Pixel

Je höher die Auflösung, d.h. je mehr Pixel wir zur Repräsentation des Originalbilds bei der Diskretisierung verwenden, desto näher ist das Ergebnis am Original.

Codierung von Bildern

- Einfachster Ansatz:
  - Diskretisierung des Bildes in Pixel
  - Auflösung = Metadaten
  - Codiere die Farbe eines jeden Pixel, z.B. von oben nach unten und von links nach rechts
- RGB-Farbmodell
  - Drei Grundfarben: rot, grün und blau
  - Alle anderen Farben als additive Mischung der Grundfarben in unterschiedlichen Mischverhältnisse
  - z.B. violett = 1 rot + 0 gelb + 1 blau



Beispiel

- Metadaten:
  - Höhe: 1
  - Breite: 4

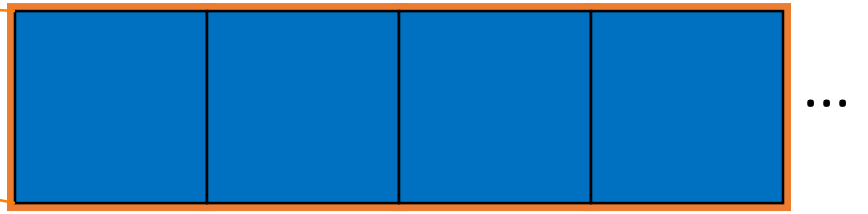
- Daten:
  - 255, 0, 0, 255, 128, 0, 34, 188, 220, 0, 0, 0

R,G,B des 1. Pixels | R,G,B des 2. Pixels | R,G,B des 3. Pixels | R,G,B des 4. Pixels

- Hierbei bedeutet „255“ maximaler Beitrag der jeweiligen Grundfarbe zur Mischung, und „0“ gar kein Beitrag → drei Bytes pro Pixel benötigt

Effizientere Codierungen

Speicherbedarf Gesamtbild:
900 x 600 x 3 Bytes = 1,6 MB



- Viele Pixel direkt nebeneinander haben dieselbe Farbe
- Idee: nutze dies für eine platzsparendere Codierung:
  10, 0, 40, 96,...

Anzahl der Wdh. | R,G,B des 4. Pixels

Lauflängencodierung

- Dieser Beobachtung trifft nicht nur auf Pixel zu.
- Allgemein: alle Bitfolgen, z.B. 1111 1110 0000 1000 0001 1111
- Da die Folge nur aus 0 und 1 besteht, wechseln sich darin immer Sequenzen desselben Bits ab.
- Idee: um die Folge zu kodieren, reicht es, die Länge der Sequenzen zu kodieren:
  - Im Beispiel oben: 7, 5, 1, 6, 5 → 111 101 001 110 101
  - Aus dieser Information allein lässt sich bereits wieder die gesamte Originalfolge rekonstruieren
  - In diesem Beispiel braucht man für diese Lauflängencodierung aber nur 15 statt wie im Original 24 Bits.
- Man muss wissen, ob die erste Sequenz aus 0‘en oder 1‘en besteht.
  - Das kann man aber einfach vorher vereinbaren / standardisieren.
- Man muss außerdem wissen, wieviele Bits für die Kodierung der Sequenzlängen verwendet werden.
  - Im Beispiel oben jeweils drei Bits (111, 101,…)
  - Als mögliche Sequenzlänge sollte auch 0 erlaubt sein.

Datenkompression

- **Definition.** Sei c eine Codierung $c: \{0,1\}^+ \to \{0,1\}^+$. Gilt für eine Bitfolge $b \in \{0,1\}^+$, dass die Anzahl der Bits von $c(b)$ kleiner ist als die Anzahl der Bits von b selbst, so nennt man c eine *Komprimierung* oder *Kompression* und die Umkehrfunktion $d = c^{-1}$ eine *Dekomprimierung* oder *Dekompression*.
- Die Lauflängencodierung ist also eine Methode der Datenkompression.

Grenzen von Komprimierungen

- **Satz.** Es gibt keine Codierung c, die für jede beliebige Bitfolge b eine Komprimierung darstellt.
- **Beweis.** Sei $n \in \mathbb{N}$. Dann gibt es 2^n verschiedene Bitfolgen der Länge n, aber nur $\sum_{k=1}^{n-1} 2^k = 2^n - 2$ verschiedene Bitfolgen, deren Länge kleiner als n ist. Aufgrund des Pidgeon-Hole-Prinzips kann daher also nicht jede Bitfolge der Länge n injektiv auf eine Bitfolge mit einer kürzeren Länge abgebildet werden.

Andere effiziente Codierungen

- Eine andere einfache Art Daten effizient (also möglichst platzsparend) zu kodieren, basiert auf folgender Überlegung:
  - Betrachtet man eine Sequenz von Elementen einer Menge A, kommen nicht alle Elemente unbedingt gleich oft darin vor.
  - Wenn wir die jedes Element als eine unterschiedliche Bitfolge kodieren, ist es sinnvoll, für die besonders häufig auftretenden Element eine möglichst kurze Bitfolge zu verwenden.

Beispiel

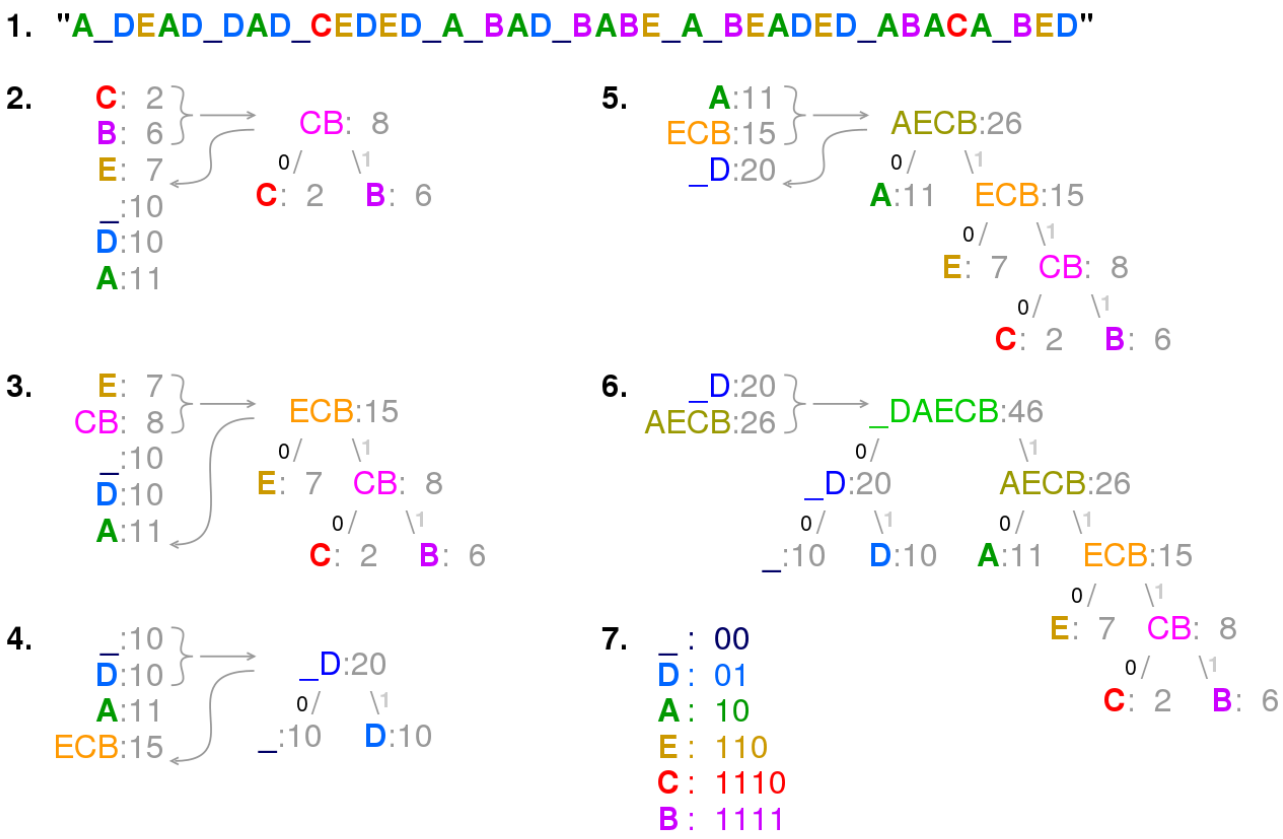
- Sei $A = \{A, B, C, D, R\}$ und $s =$ ABRACADABRA.
- Würden wir für die Kodierung der Elemente von A jeweils dieselbe Anzahl an Bits verwenden – z.B. 3 pro Element – bräuchten wir zur Kodierung der Sequenz s genau $11 \cdot 3 = 33$ Bits.
- Verwendet man jedoch stattdessen folgende Kodierung, benötigt man nur 23 Bits:
  0 110 111 0 100
  0 101 0 110 111 0

	A	B	C	D	R
Codierung	0	110	100	101	111
Häufigkeit in s	5	2	1	1	2
Benötigte Bits (gesamt)	5	6	3	3	6

Huffman-Coding

- Verfahren:
  1. Erstelle eine Liste aller Elemente zusammen mit ihrer Vorkommenshäufigkeit
  2. Sortiere die Liste nach den Häufigkeiten
  3. Entferne die zwei Elemente mit dem kleinsten Häufigkeitswert:
     - Ordne einem der beiden das 0-Bit und dem anderen das 1-Bit zu
  4. Erzeuge ein neues Element, dass diese beiden Elemente als Unterelemente enthält
     - Dieses neue Element erhält als Häufigkeitswert die Summe der Häufigkeitswerte der beiden zuvor entfernten Elemente
  5. Füge das neue Element so in die Liste ein, dass sie nach wie vor korrekt sortiert ist
  6. Wiederhole die Schritte 3 bis 5, bis nur noch ein Element übrig ist
  7. Die Bitkodierungen können dann für jedes Unterelement abgelesen werden.

Beispiel



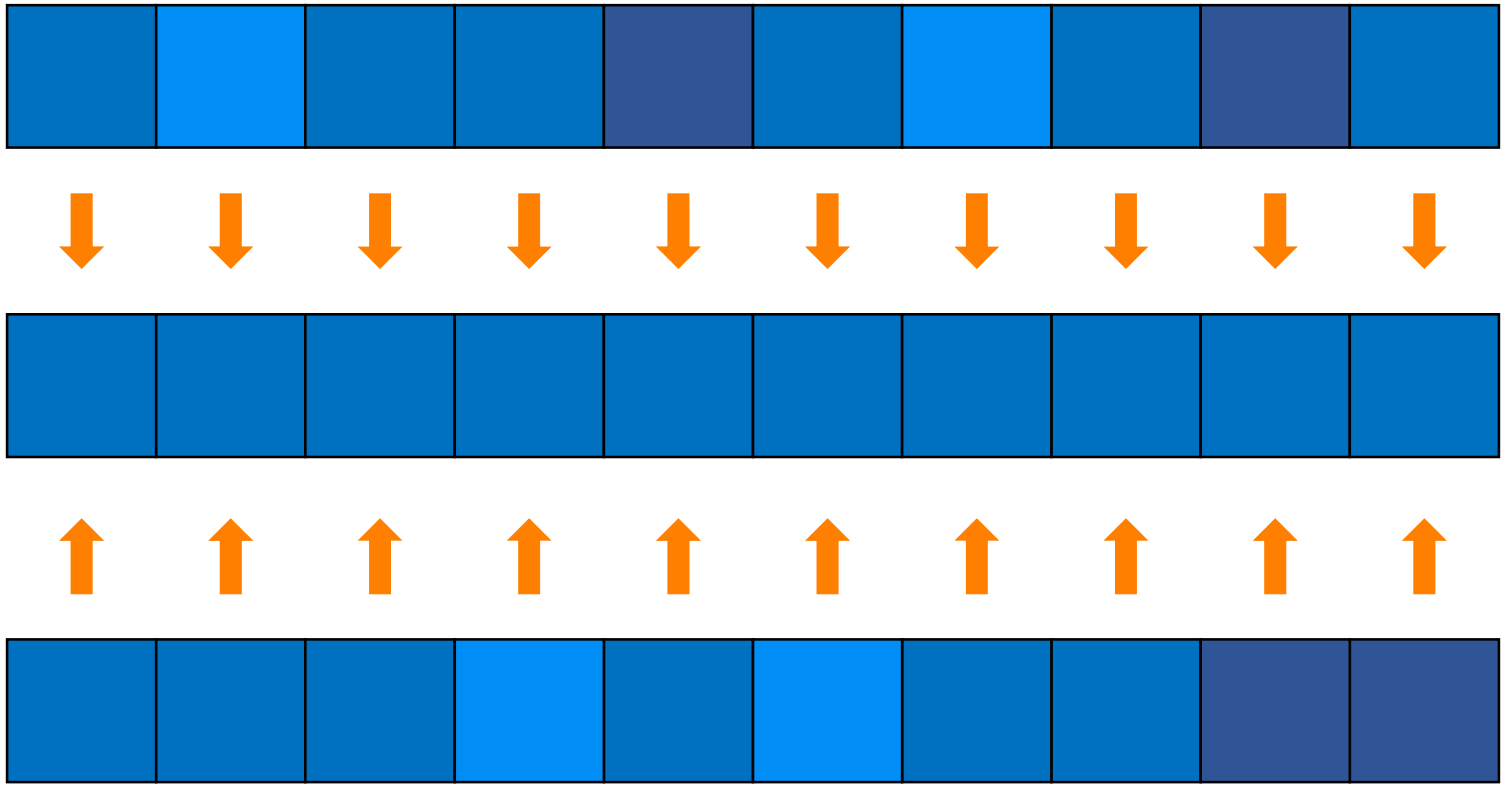
Verlustbehaftete Komprimierung

- Wenn man das zu codierende Element vor der Kodierung ein wenig vorverarbeitet, kann man eventuell noch besser *Kompressionsraten* erreichen.
  - Kompressionsrate = $\frac{\text{Länge der Bitfolge nach der Komprimierung}}{\text{Länge der Bitfolge vor der Komprimmierung}}$
- Beispiel: wenn wir die Sequenz ABRACADABRA vorverarbeiten zu der Sequenz AAAAAAAAAAA, lässt sich diese wesentlich effizienter komprimieren.
  - Nachteil: die Vorverarbeitung verändert die Sequenz in inakzeptabler Weise

Verlustbehaftete Kompression von Bildern

- Unter anderen Umständen kann eine geeignete Vorverarbeitung aber durchaus unauffälliger ausfallen, z.B. bei Bildern.

Da Pixel bei modernen Bildschirmen sehr klein sind, kann das Auge kleine Farbunterschiede nicht wahr-nehmen.



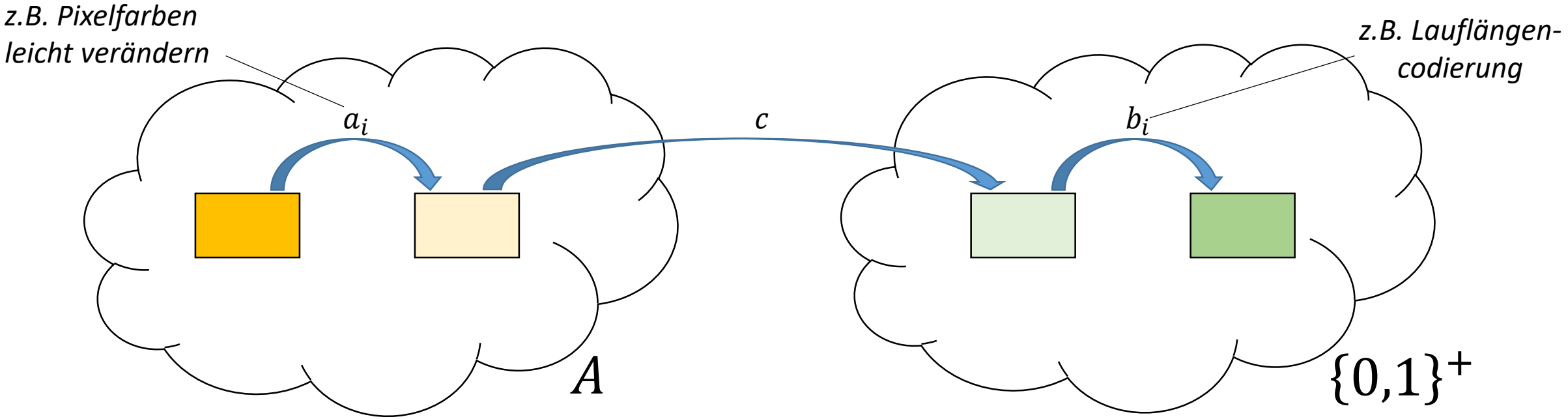
Verändert man die Blautöne gering-fügig, lässt sich das Bild besser kom-primieren, aber anschließend lässt sich nicht mehr feststellen, wie das Bild im Original ausgesehen hat. Diese Information ist verloren (daher „verlustbehaftete“ Kompression).

Menschliche Wahrnehmung

- Ändern wir zuerst die Pixel bevor wir das Bild codieren, codieren wir eigentlich nicht das Bild selbst, sondern eine leicht abgeänderte Variante davon.
- Dabei nutzt man die Limitierungen des menschlichen Wahrnehmungsapparats aus: wenn die Änderungen dezent genug sind, fallen sie praktisch nicht auf.
- Bei Texten ist eine derartige verlustbehaftete Komprimierung daher nicht ohne Weiteres möglich.
- Generelle Abwägung: Originaltreue vs. Kompressionsrate

Funktionale Sicht

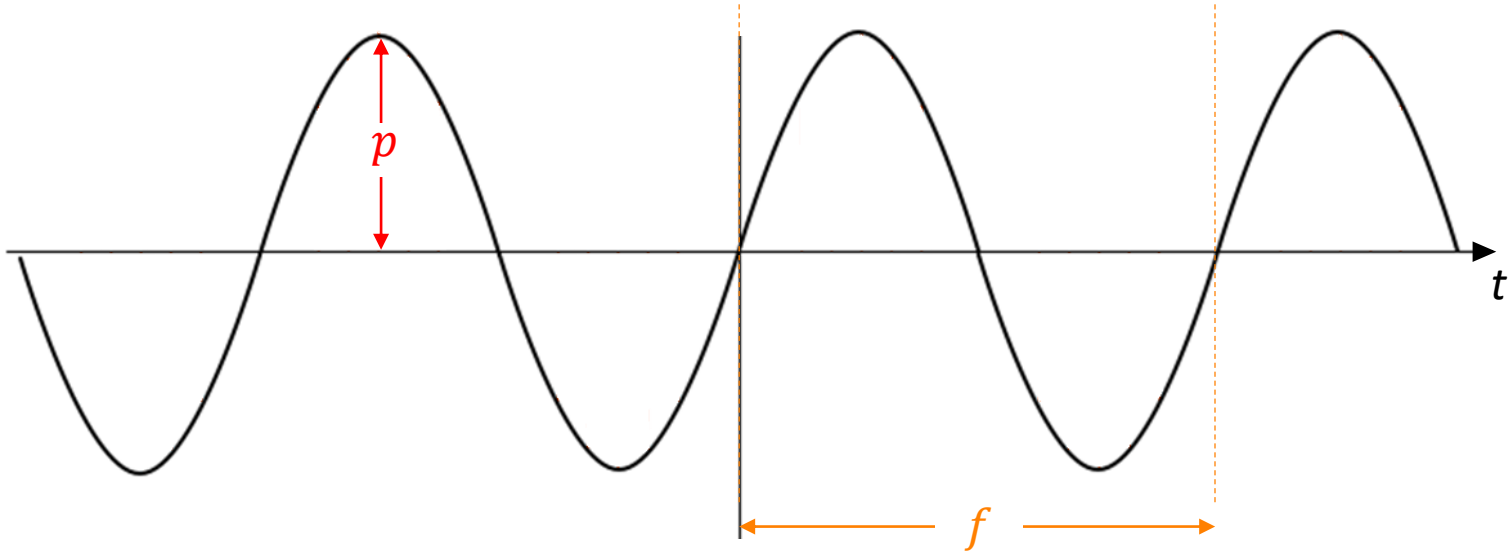
- Wir betrachten die Menge A, Abbildungen $a_1, \dots a_n$ von $A \to A$, Codierungen $b_1, \dots, b_m$ von $\{0,1\}^+$ und eine Codierung $c: A \to \{0,1\}^+$ von A. Dann können diese wie folgt kombiniert werden.



Audio

- Wie bei Bildern auch muss ein kontinuierlicher Klang erst in diskrete Einheiten unterteilt werden, damit er digital gespeichert werden kann
  - Dabei geht Information verloren → Approximation
- Wie entsteht überhaupt Klang? Die Luft wird in Schwingungen versetzt, die von unserem Ohr aufgefangen werden.
- Einfachster Fall: gleichmäßiges Vor- und Zurückschwingen über einen längeren Zeitraum
  - Sinuswelle
- Diskretisierung von Audio: Sampling

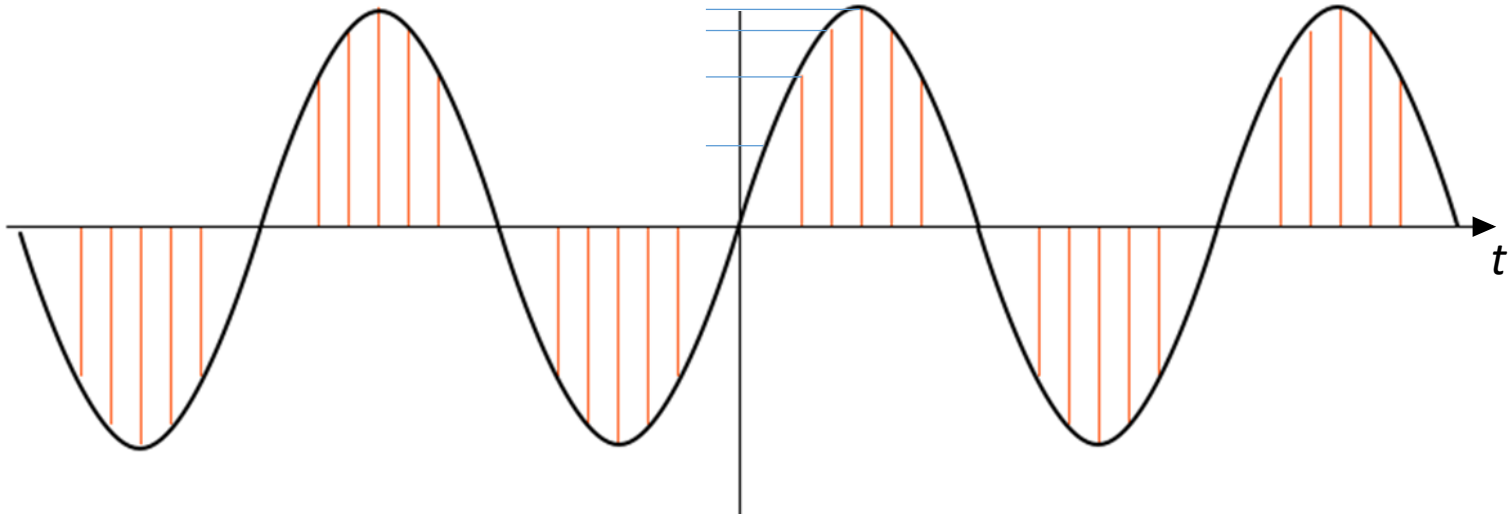
Akustisches Sinussignal



- Frequenz f: Tonhöhe
- Amplitude: Schalldruck (im Verhältnis zu Bezugspegel 0dB: Lautstärke)

Sampling

- In regelmäßigen Intervallen wird das Signal gemessen (rote Linien).

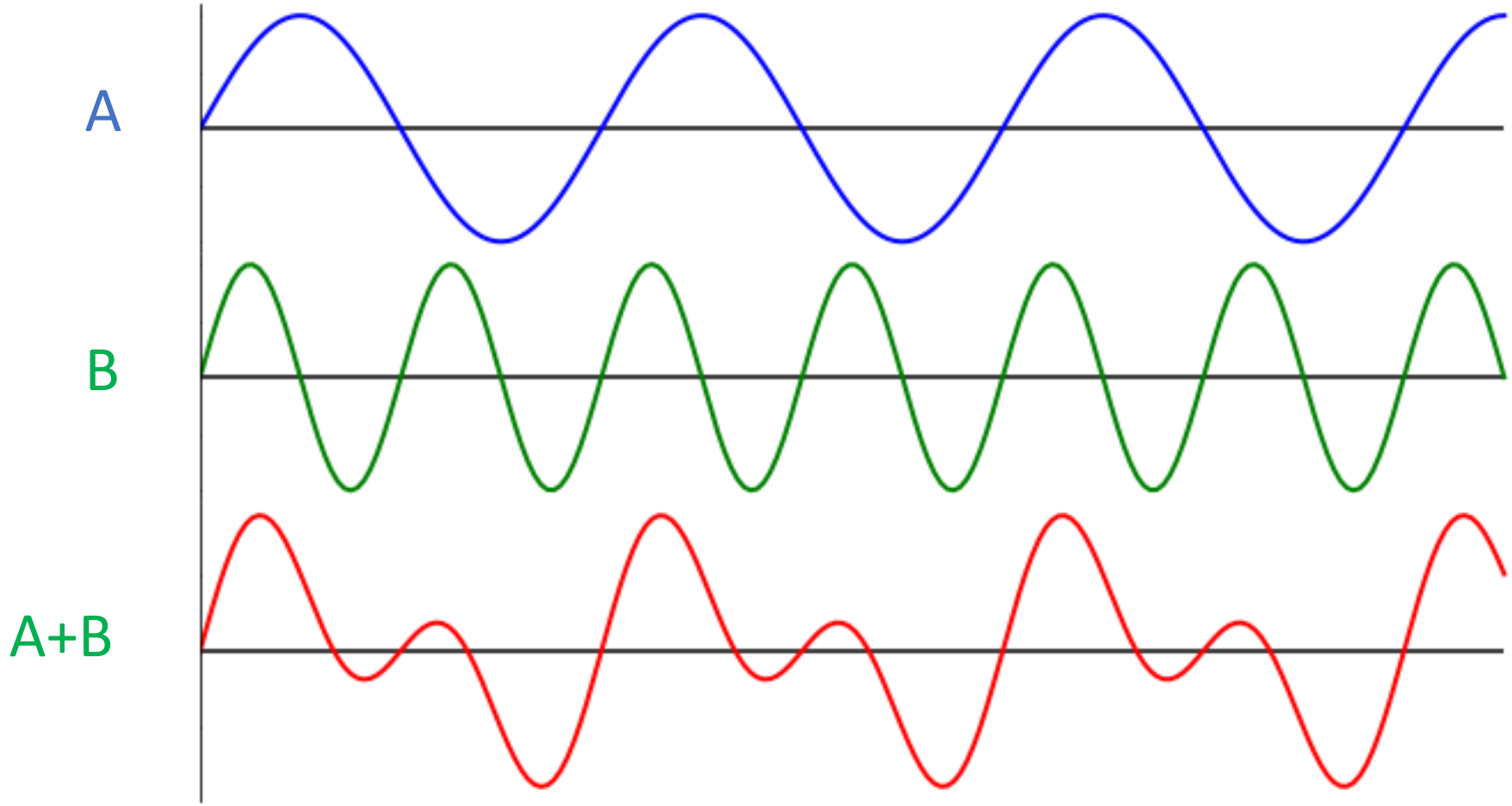


- Die diskreten Messpunkte stellen eine Annäherung an das tatsächliche Signal dar.
- Da ein Audiosignal auch bzgl. der Y-Achse kontinuierlich ist, muss auch hier eine Diskretisierung vorgenommen werden (blaue Linien).
- Mehr Messungen (sampling rate) und mehr Bits zur Repräsentation der Messwerte bedeuten eine bessere Annäherung.

Komplexe Klänge

- Sinuswellen klingen unschön, und interessante Klänge sind komplexer.
- Denoch bleibt das Prinzip auch bei diesen dasselbe:
  - Mittel Hilfe eines Mikrophons werden Schallwellen in analoge Signale überführt.
  - Der Wert dieses Signals wird in regelmäßigen Abständen gesamplet (z.B. 44.1 kHz).

Komplexe Klänge als Summe unterschiedlicher Sinusschwingungen



Speicherbedarf - Beispiel

- Wir betrachten ein Musikstück von genau 3 Minuten Länge.
- Als Samplingrate wählen wir 48 kHz.
- Jeder gesampelte Wert soll in 16 Bits gespeichert werden.
- Wieviel Speicherplatz benötigt das gesamte Musikstück?

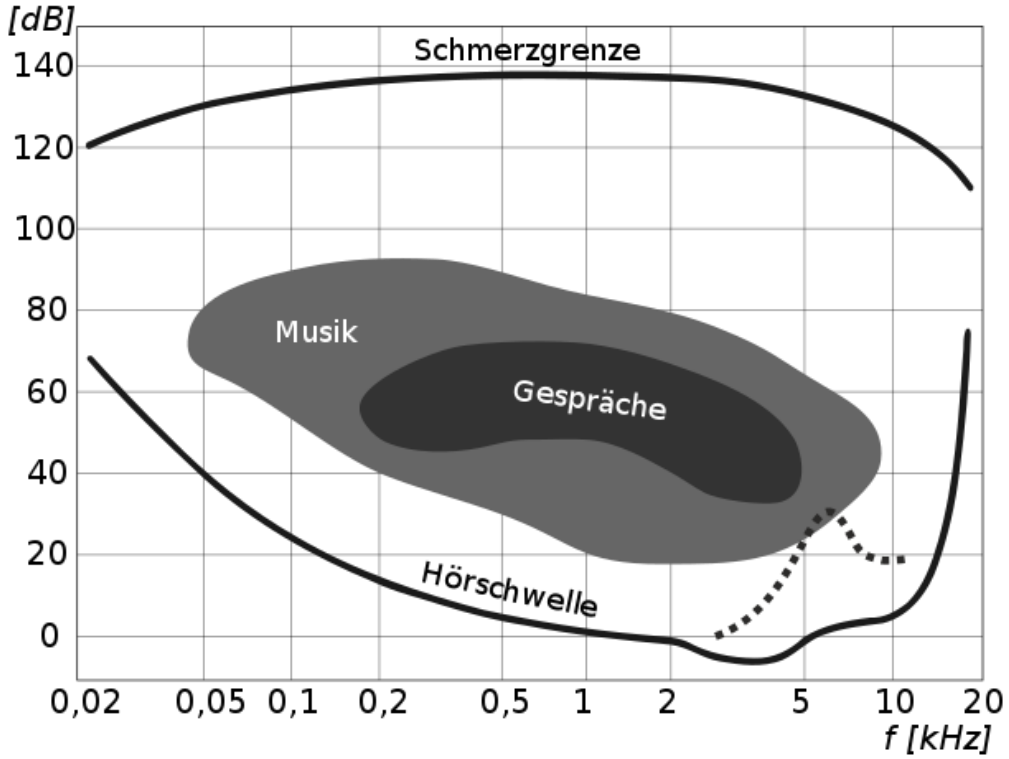
$$3 \cdot 60s \cdot 48000\,\frac{1}{s} \cdot 16\ \text{Bits} = 138240000\ \text{Bits}$$

→ Man benötigt ca. 16.5 MB. Das ist recht viel.

Verlustbehaftete Audio-Kompression

- Bei Bildern konnten wir ausnutzen, dass geringe Farbschwankeungen vom menschlichen Augen nicht wahrgenommen werden können?
- Gibt es ähnliche Möglichkeiten auch bei Audio?
  - Menschliche Hörschwelle
  - Maskierungseffekt

Hörschwelle

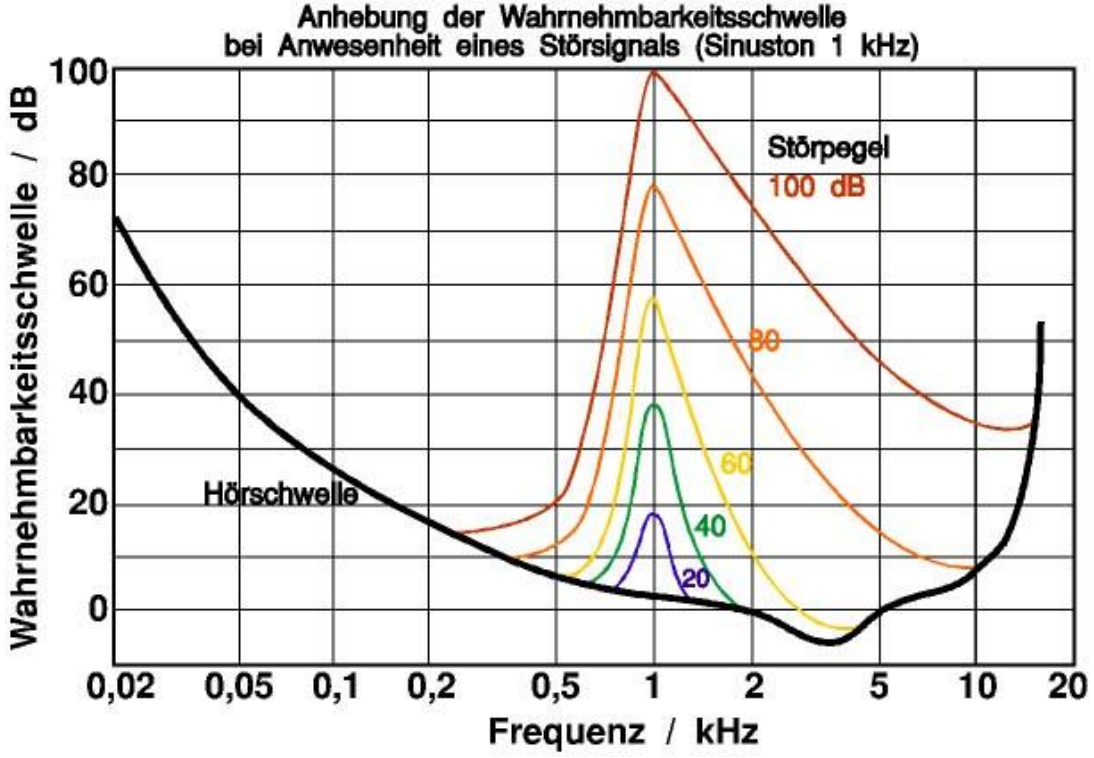


- Das menschliche Gehör kann Töne und Klänge nur in einem gewissen Frequenzbereich wahrnehmen
  - ca. 20 Hz bis 20000 Hz
- Je nach Frequenz muss das Audiosignal außerdem eine bestimmte Lautstärke (Schalldruckpegel) haben, um hörbar zu sein.

Maskierungseffekt

- Ob wir einen Ton mit einer bestimmten Frequenz und Lautstärke wahrnehmen können, hängt auch davon ab, welche anderen Signale zur gleichen Zeit erklingen.

Beispiel: Einfluss eines Störsignals



- Die Hörschwelle verändert sich in Anwesenheit eines Sinustons von 1 kHz wie dargestellt.
- Je nach Lautstärke dieses Störsignals können andere Frequenzen nur noch schwerlich wahrgenommen werden.

Anwendung bei Komprimierung

- Audio-Kompressionsverfahren wie z.B. MP3 machen sich diese Erkenntnisse der Psychoakustik zu Nutze.
- Frequenzen, die im Originalsignal zwar anwesend und auch gemessen werden können, werden beim Codieren einfach weggelassen, wenn sie vom menschlichen Gehör nicht wahrgenommen werden können.
- Dadurch reduziert sich die zu speichernde Datenmenge erheblich.
- Typischer Speicherplatzbedarf für ein 3-Minuten-Musikstück bei MP3-Kompression: ca. 3 bis 5 MB.
  - Das hängt natürlich auch (wie oben) von Samplefrequenz und Bitrate ab.

Beispiele für Komprimierungen

	Allgemeine Bitfolgen	**Bilder**	**Audio**
Unkomprimiert		BMP, TIFF, RAW	LPCM (wav, aiff)
Verlustfreie Komprimierung	gzip, xz, 7zip, DEFLATE (zip)	GIF, PNG, WEBP	FLAC
Verlustbehaftete Komprimierung		JPEG, WEBP	MP3, AAC

Zusammenfassung (Codierung)

- Eine binäre Codierung erlaubt es, Elemente einer Menge A als Folgen von Bits zu repräsentieren.
- Eine Codierung muss *unterschiedliche* Elemente von A auch auf *unterschiedliche* Bitfolgen abbilden (Injektivität der Codierungsabbildung).
- Kontinuierliche Elemente müssen erst diskretisiert werden.
  - Spezifische Verfahren je nach Typ des Elements (Bild, Audio,…)
  - Dabei geht bereits Information verloren (lediglich Annäherung)
- Für eine Menge gibt es i.A. mehrere Codierungsmöglichkeiten
- Unterschiedliche Mengen (z.B. die Menge aller englischen Texte, die Menge aller Bilder, etc.) benötigen in der Regel auch eigene, speziell auf sie zugeschnittene Codierungen.
  - Ein Bild-Dekodierer kann keine Musikstücke dekodieren.

Zusammenfassung (Komprimierung)

- Man ist oft an möglichst effizienten Codierungen interessiert, d.h. an solchen, die die Elemente von A auf möglichst kurze Bitfolgen abbilden.
- Lauflängen-Codierung und Huffman-Coding als Beispiele
- Ein Vor- oder Nachverarbeitungsschritt kann dabei helfen.
- Sind diese Verarbeitungsschritte nicht injektiv, spricht man von einer verlustbehafteten Komprimierung.
  - Hier können Limitierungen der menschlichen Wahrnehmung genutzt werden.

Intuitiver Informationsbegriff

- Achtung, eine wichtige Information für Sie!
  - Saarbrücken liegt nahe Frankreich.
  - Saarbrücken liegt genau auf dem 7. Längengrad Ost.
- Welche der beiden Aussagen hat einen höheren Informationsgehalt?
- Intuitiv kann man sagen:
  - Ist der Inhalt einer Nachricht bereits bekannt, oder kann man sich diesen leicht selbst denken, ist der Informationsgehalt gering.
  - Enthält die Nachricht jedoch Neuigkeiten, die zu einem gewissen Grad vielleicht sogar überraschend sind, ist der Informationsgehalt höher.

Ein Ratespiel

- Drei Nachrichten über dem Alphabet $\{A, B, C, D\}$
- Nachricht 1): `A A A A A A A A A A A A A A A A.`
- Nachricht 2): `A B A B A C A B A C A B A B A B.`
- Nachricht 3): `A B D B C D D A B D C D C A D D.`
- Welcher Buchstabe kommt jeweils als nächstes?
- Der Inuition der vorherigen Folie folgend hätte 1) einen geringen, 2) einen mittleren und 3) den höchsten Informationsgehalt.

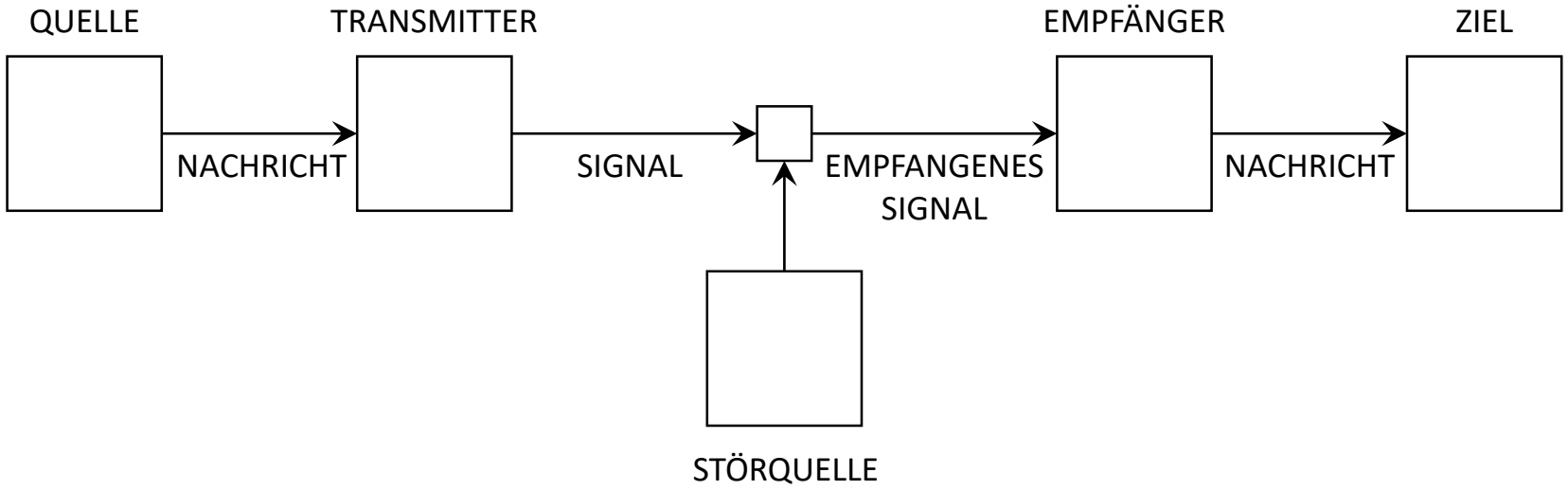
Zusammenhang zwischen Kompression und Informationsgehalt einer Nachricht

- Bei der Lauflängencodierung lassen sich lange Sequenzen desselben Bits besonders effizient komprimieren. Strecken, in denen sich 0 und 1 stets abwechseln, können überhaupt nicht komprimiert werden.
- Beim Huffman-Coding wird das am häufigsten vorkommende Element am effizientesten kodiert. Kommen hingegen alle Elemente ungefähr gleich oft vor, erreicht man keine nennenswerte Komprimierung.
- Wir erweitern unsere Intuition des Informationsbegriffs entsprechend: eine Nachricht lässt sich genau dann gut komprimieren, wenn sie einen geringen Informationsgehalt hat.

Der Informationsbegriff in der Informatik

- Der Begriff der „Information“ besitzt keine allgemein anerkannte und einheitliche Definition. Er wird in unterschiedlichen Disziplinen auch unterschiedlich verwendet.
- Die in der Informatik am weitesten verbreitete Auffassung geht auf Claude Shannon zurück, der sich in den 1940er-Jahren mit technischen Fragen der Nachrichtenübertragung auseinandergesetzt hat.
- Dabei wird ein rein syntaktisches Verständnis von Information probagiert ohne dabei semantische Aspekte zu berücksichtigen.

Shannon‘s „Noisy Channel“ Modell



- Um eine Nachricht von einer Quelle an ein bestimmtes Ziel zu übertragen, muss sie derart codiert werden, dass sie über einen Übertragungskanal versendet werden kann.
- Das empfangene Signal wird dekodiert, um die Originalnachricht wieder herzustellen.
- Es kann u.U. aber während der Übertragungen zu Störungen kommen, die das Signal unbemerkt verändern.

Beispiele

- Das Modell ist bewusst abstrakt und allgemein gehalten. Quelle und Ziel können je nach Fall ganz unterschiedlicher Natur sein:
  - Übertragung einer Sprachnachricht per Telefon von einer Person zu einer anderen.
  - Übertragung von Messdaten eines Sensors an eine Kontrollstation über das Internet.
  - Speichern und späteres wieder Auslesen eines Dokuments auf einer Computerfestplatte.
  - etc.

Modell der Quelle und der Nachricht

- Aus diesem Grund wird auch der Begriff der Quelle selbst und der Nachricht hier stark abstrakt gefasst:
- Gegen ein Alphabet $A = \{a_1, \dots, a_k\}$. Eine Quelle produziert eine Nachricht als Sequenz von Elementen aus A.
  - Beispiel für eine solche Nachricht: $a_{10}, a_{42}, a_3, a_{10}, a_{27}, \dots, a_{54}$
- Shannon geht weiterhin davon aus, dass in einer Nachricht nicht alle diese Elemente gleich wahrscheinlich sind, d.h. in der Regel auch nicht gleich häufig vorkommen.

Informationsgehalt einer Nachricht

- Wir wissen vom Huffman-Coding bereits, dass es für eine effiziente Komprimierung sinnvoll ist, häufig vorkommende Elemente mit möglichst wenig Bits zu repräsentieren.
- Wenn wir davon ausgehen, dass eine Nachricht gerade dann viel Information enthält, wenn sie sich nicht gut komprimieren lässt, kann mit die Anzahl der verwendeten Bits als Maß für den Informationsgehalt verwenden.

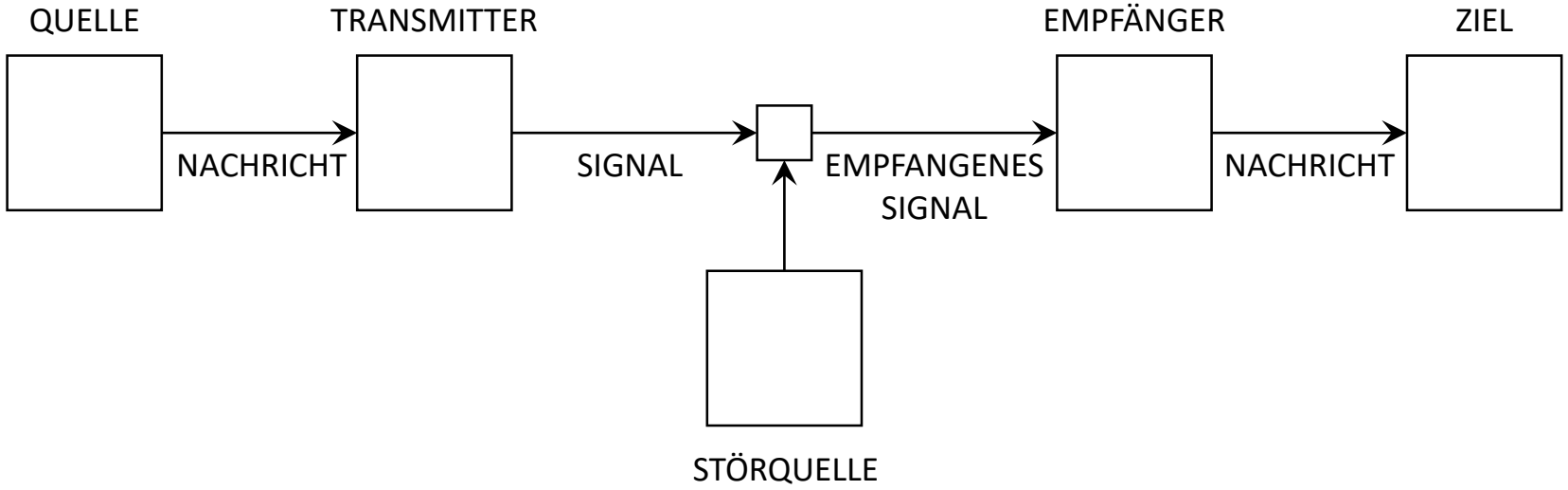
Beispiel

- Sei Q eine Quelle, die Nachrichten über einem Alphabet A versendet. Wir wissen außerdem, welche Elemente aus A oft in Nachrichten vorkommen und welche selten.
- Die Nachrichten werden durch den Transmitter effizient kodiert, z.B. mit Huffman-Coding.
- Der Empfänger erhält nun zwei konkrete Nachrichten m_1, m_2 von Q, die beide aus 20 Element von A bestehen. Jedoch sind sie unterschiedlich lang:
  - Länge(m_1) = 102 Bits
  - Länge(m_2) = 168 Bits
- Das bedeutet, dass Nachricht m_2 mehr Information im Shannon‘schen Sinne enthält.

Zusammenfassung (Informationsbegriff)

- Information als Konzept hat keine einheitliche Definition
- In der Informatik wird der Begriff oft mit Bezug auf Shannon’s Kommunikationsmodell verwendet.
- Intuitiv ist eine Nachricht dann reich an Information, wenn sie nicht einfach vorhersagbar ist.
- Dadurch gibt es auch eine Verbindung zur Komprimierung: Nachrichten mit höherem Informationsgehalt lassen sich schwieriger komprimieren.
- Bitlänge einer effizient codierten Nachricht als Maß ihres Informationsgehalts.

Übertragungsfehler



- Im „Noisy Channel“-Modell wird angenommen, dass das Signal während der Übertragung einem äußeren Einfluss ausgesetzt sein kann (Störquelle), der das Signal verändert.
- Bei der Übertragung von Bitfolgen bedeutet dies, dass es gelegentlich vorkommen kann, dass ein Bit gekippt wird (z.B. gesendet: 0, empfangen: 1 – oder umgekehrt).

Beispiel: Auswirkung von Bitfehlern

- Textnachricht: *„Ich würde dich wirklich gerne heute Abend treffen...“*
- Antwort: ...

	Unicode-Zeichen	UTF-16 Kodierung
	U+1F61A KISSING FACE WITH CLOSED EYES	11011000 00111101 11011110 00011010
	U+1F612 UNAMUSED FACE	11011000 00111101 11011110 00010010

- Wie kann der Empfänger wissen, ob die Nachricht korrekt ist?

Einfacher Ansatz: Wiederholungskodierung

- Idee: sende jedes einzelne Bit dreimal hintereinander
- Beispiel: 00101 → 000 000 111 000 111
- Der Empfänger könnte dies wie folgt dekodieren:

Empfangene Bits	...dekodiert als
Drei Nullen	0
Zwei Nullen, eine Eins	0
Eine Null, zwei Einsen	1
Drei Einsen	1

- Aber selbst dann kann der Empfänger sich nie 100%-ig sicher sein.

Paritätstest für Blöcke von i Bits

- Der Hauptnachteil der Wiederholungskosten ist das erhöhte Datenvolumen
- Alternativer Ansatz: *Paritätstest* beim Senden einer Nachricht s.
- Idee:
  - Kodierung:
    - Unterteile die zu sendende Nachricht s in Blöcke von i Bits.
    - Sende zusätzlich auch das XOR der Bits eines jeden Blocks.
  - Dekodierung
    - Unterteile die empfangene Nachricht r in Blöcke von i+1 Bits.
    - Überprüfe ob das i+1-te Bit dem XOR der anderen Bits entspricht.
- Da XOR einer Summe modulo 2 entspricht, nennt man das Paritätsbit auch *Checksumme, Prüfsumme* oder *Prüfziffer*.

Beispiel

- s = 0101001011010101
- i = 4
- Blöcke: 0101 0010 1101 0101
- Paritätsbits:
  - $0101 \rightarrow 0 \oplus 1 \oplus 0 \oplus 1 = 0$
  - $0010 \rightarrow 0 \oplus 0 \oplus 1 \oplus 0 = 1$
  - $1101 \rightarrow 1 \oplus 1 \oplus 0 \oplus 1 = 1$
- s' = 0101<u>0</u>0010<u>1</u>1101<u>1</u>0101<u>0</u>

- r = 01010011011101000011
- Blöcke: 01010 01101 11010 00011
- Paritätstest:
  - 0101 → 0
  - 0110 → 0
  - 1101 → 1
  - 0001 → 1
- Aber: falsche Sicherheit, wenn die Anzahl der gekippten Bits innerhalb eines Blocks gerade (und > 0) ist.

Kosten fehlererkennender Codes

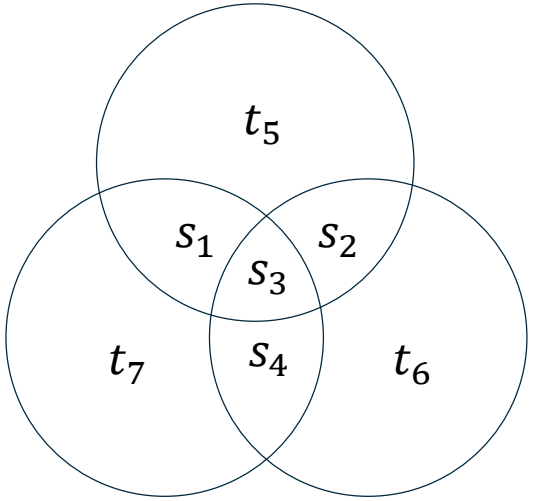
- Generell ist das Ziel fehlererkennender und –korrigierender Codes, eine *Datenredundanz* zu erzeugen, sodass dieselbe Information mehrfach codiert wird.
- Vergleicht man Wiederholungskodierung und Paritätstest, erkennt man folgenden Zusammenhang:
  Eine höhere Übertragungstreue muss man mit mehr Bits „bezahlen“.
- Im Beispiel:
  - Wiederholungskodierung = gute Effektivität, 200% höheres Datenvolumen
  - Paritätsbit = mäßige Effektivität, 25% höheres Datenvolumen

Hamming-Code

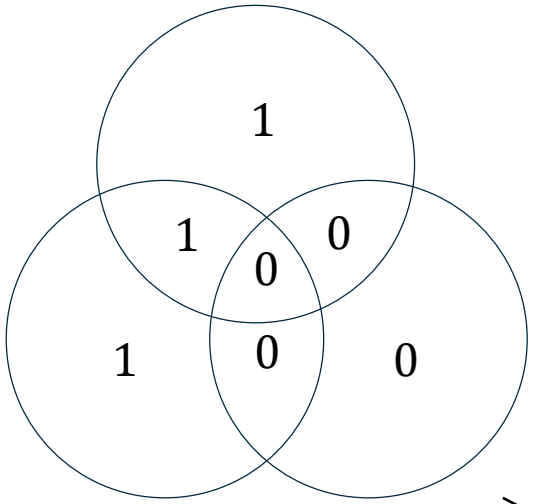
- Ein Mittelweg (gute Effektivität, mäßiger Datenanwuchs) basiert ebenfalls auf Paritätsbits: der Hamming-Code
- Wie zuvor wird die zu codierende Bitfolge in Blöcke geteilt.
- Statt eines einzelnen Paritätsbits, das aus dem gesamten Block berechnet wird, erzeugt man aus Teilen des Blocks mehrere Paritätsbits.
- Jedes Bit aus dem Block geht dabei in die Berechnung mehrerer Paritätsbits mit ein.

(7,4)-Hamming-Code

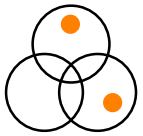
- Warum (7,4)?
  - 7 = 4 + 3 → Blockgröße = 4 Bits, Anzahl der Paritätsbits = 3
- Sende für jeden Block $s_1\ s_2\ s_3\ s_4$ drei zusätzliche Paritätsbits $t_5\ t_6\ t_7$
  - $t_5 = s_1 \oplus s_2 \oplus s_3$
  - $t_6 = s_2 \oplus s_3 \oplus s_4$
  - $t_7 = s_1 \oplus s_3 \oplus s_4$

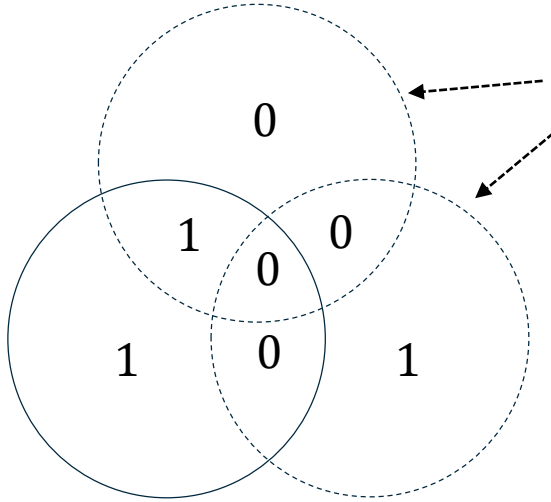


Auswirkungen von Übertragungsfehlern



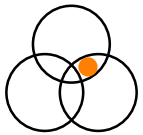
Beispiel: zwei Bits Kippen

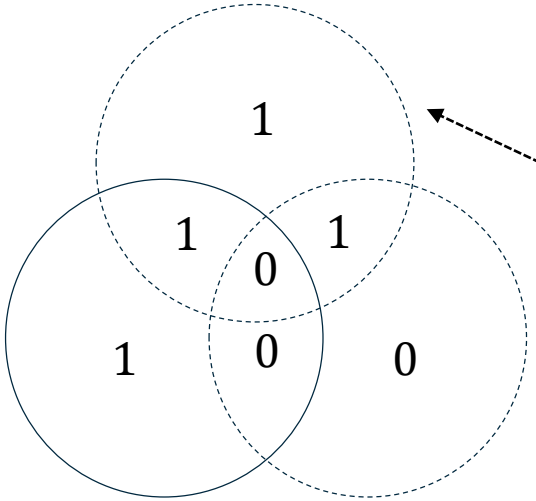




Der Empfänger kann feststellen, dass zwei der Paritätsbits inkorrekt sind...

Beispiel: ein Bit kippt





...das kann aber unterschiedliche Ursachen haben.

Aber: wenn das Kippen eines Bits schon eher unwahrscheinlich ist, ist das Kippen zweier Bit noch unwahrscheinlicher.

Dekodierung der Bits $s_1\ s_2\ s_3\ s_4$

- Der Empfänger überprüft, ob die Paritätsbits t_5, t_6, t_7 korrekt sind.
- Ist genau eines der Paritätsbits falsch, ist es sinnvoll anzunehmen, dass dieses Bit selbst gekippt wurde → $s_1 s_2 s_3 s_4$ bleiben unverändert
- Bei zwei oder drei inkorrekten Paritätsbits verfahren wir so:

t_5	t_6	t_7	**Wahrscheinlichste Ursache**	**Dekodiere Block als**
korrekt	inkorrekt	inkorrekt	s_4 wurde gekippt	$s_1\ s_2\ s_3 \overline{s_4}$
inkorrekt	korrekt	inkorrekt	s_1 wurde gekippt	$\overline{s_1}\ s_2\ s_3\ s_4$
inkorrekt	inkorrekt	korrekt	s_2 wurde gekippt	$s_1\ \overline{s_2}\ s_3\ s_4$
inkorrekt	inkorrekt	inkorrekt	s_3 wurde gekippt	$s_1\ s_2\ \overline{s_3}\ s_4$

Weitere Anwendungen fehlererkennender und –korrigierender Codes

- Wie auf Folie 293 gesehen, lässt sich das Noisy Channel – Modell nicht nur auf Datenfernübertragung anwenden.
- Daher sind in der Praxis auch in anderen Fällen Methoden zur Erkennung und Korrektur von Fehlern in Codes implementiert.
  - EAN (European Article Number – Strichcode auf Produkten)
  - ISBN
  - QR- Code

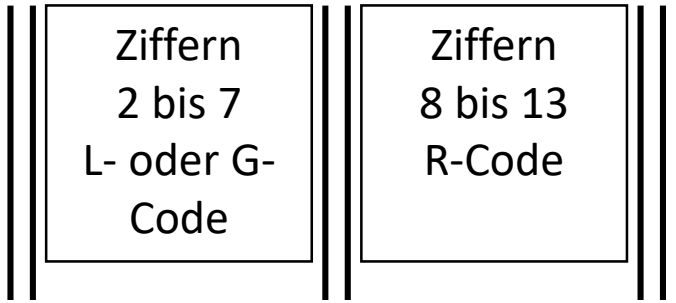
European Article Number

- Strichcode auf Produkten
- Kodiert eine Produktnummer aus 13 (EAN-13) oder 8 (EAN-8) Ziffern, standardisiert durch die Organisation GS1.
- Herstellen können sich selbst und ihre Produkte in einer internationalen Datenbank registrieren.
- Jedes Geschäft speichert zudem noch Preise für alle Artikels seines Sortiment.

Redundante Kodierung der Ziffern

Ziffer	L-Code	G-Code	R-Code
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

- Verwendung der L-, G-, R-Codes:



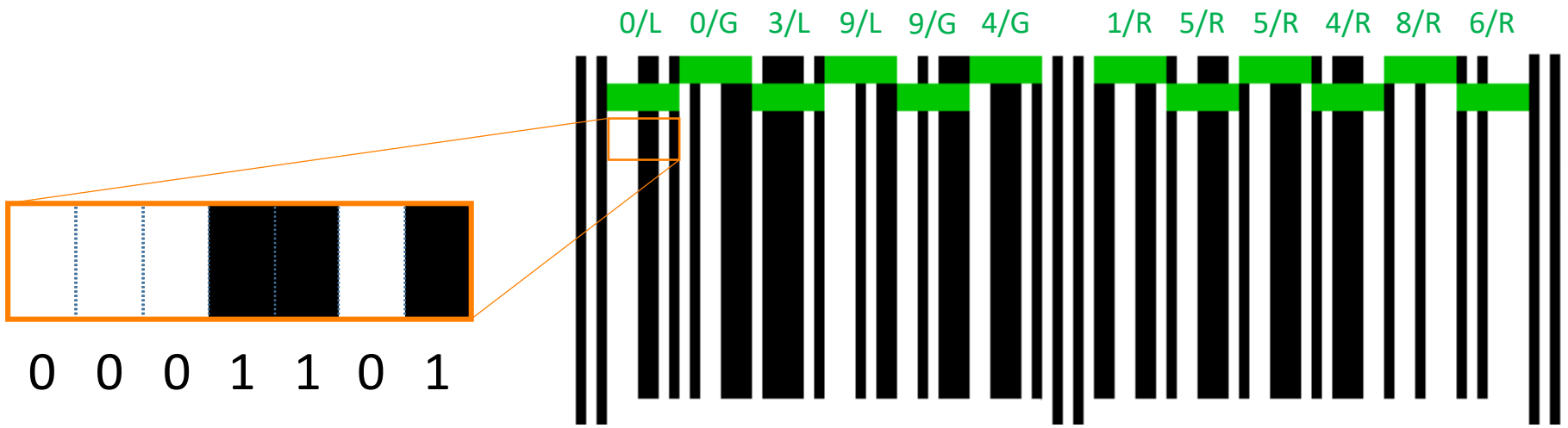
Ziffer 1	Muster	Ziffer 1	Muster
0	LLLLLL	5	LGGLLG
1	LLGLGG	6	LGGGLL
2	LLGGLG	7	LGLGLG
3	LLGGGL	8	LGLGGL
4	LGLLGG	9	LGGLGL

Prüfziffer

- Die dreizehnte Ziffer im EAN-Code ist eine Prüfziffer.
- Zu ihrer Berechnung benötigt man zunächst die gewichtete Summe S der ersten zwölf Ziffern $z_1, \dots, z_{12}$ wie folgt: $S = 3z_1 + z_2 + 3z_3 + z_4 + 3z_5 + z_6 + 3z_7 + z_8 + 3z_9 + z_{10} + 3z_{11} + z_{12}$
- Die Prüfziffer ist dann die Differenz zwischen S und der nächstgrößeren, ohne Rest durch 10 teilbaren Zahl.
  - Ist S selbst ohne Rest durch 10 teilbar, ist die Prüfsumme 0.

Beispiel

- Bit 0: weißer Strich, Bit 1: schwarzer Strich



Aus dem Muster LGLLGG folgt: die erste Ziffer ist 4.

Somit lautet die gesamte EAN:

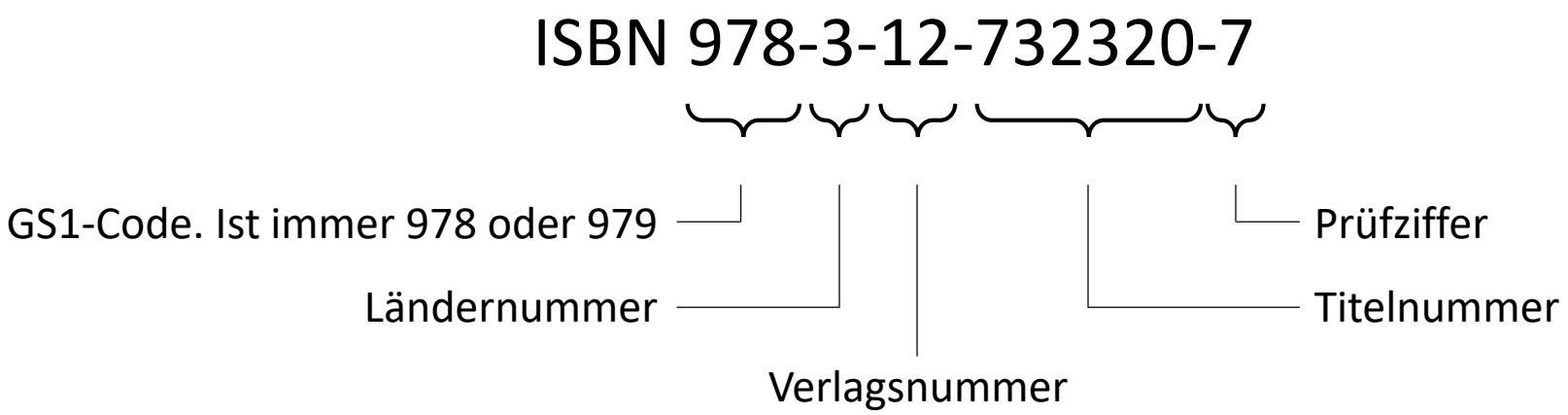
4 0 0 3 9 9 4 1 5 5 4 8 6

- Stimmt die Prüfziffer 6?

$$3 \cdot (4 + 0 + 9 + 4 + 5 + 4) + (0 + 3 + 9 + 1 + 5 + 8) = 104$$

Internationale Standardbuchnummer (ISBN)

- Die ISBN-Nummer wird im Buchhandel zur Kennzeichnung von Veröffentlichungen verwendet.
- Es gibt sie in zwei Varianten, ISBN-10 und deren Erweiterung ISBN-13.
- ISBN-13 ist ein Strichcode, der die EAN-Methode verwendet.
- Lediglich die Interpretation der 13 Ziffern ist anders:



Zusammenfassung fehlererkennende –und korrigierende Codes

- Bei der Übertragung von Bitfolgen kann es gelegentlich zu Fehlern kommen.
- Um dem entgegenzuwirken, kodiert man dieselbe Information mehrfach redundant.
- Dazu gibt es verschiedene Ansätze: Wiederholungskodierung, einfaches Paritätsbit, Hamming-Code, u.a.
- Die Anwendungsmöglichkeiten sind breit gefächert, z.B. beim Scannen von Strichcodes, Speichern und Lesen von Daten, aber insbesondere natürlich auch beim Datenaustausch zwischen mehreren Rechnern.