EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT

Medieninformatik / Human-Computer Interaction

Grundlagen der Multimediatechnik

Informationstheorie, Textcodierung und -komprimierung

05.11.2021, Prof. Dr. Enkelejda Kasneci

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Termine und Themen

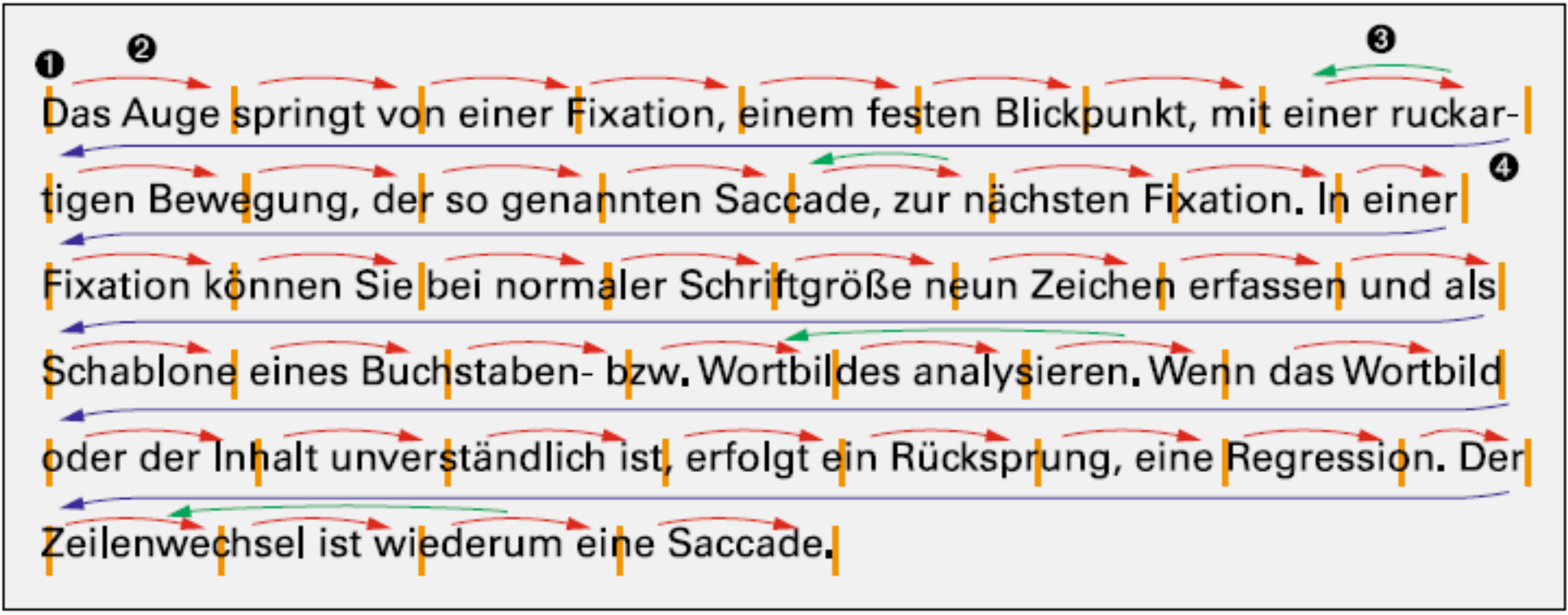
22.10.2021	Einführung
29.10.2021	Menschliche Wahrnehmung – visuell, akustisch, haptisch, …
05.11.2021	Informationstheorie, Textcodierung und -komprimierung
12.11.2021	Bildverbesserung
19.11.2021	Bildanalyse
26.11.2021	Grundlagen der Signalverarbeitung
03.12.2021	Bildkomprimierung
10.12.2021	Videokomprimierung
17.12.2022	Audiokomprimierung
14.01.2022	Videoanalyse
21.01.2022	Dynamic Time Warping
28.01.2022	Gestenanalyse
04.02.2022	Tiefendatengenerierung
11.02.2022	FAQ mit den Tutoren
15.02.2022	Klausur (noch nicht bestätigt)

Was passiert beim Lesen?

- Moderne geübte Leser nehmen Information sehr schnell auf:
  - Leseanfänger buchstabieren
  - Mittelalter:
    Lesen mit Mundbewegung → Lesetempo = Sprachtempo
  - Heute: Verschiedene Lesegeschwindigkeiten („Überfliegen“),
    Wechselbeziehung zwischen Inhalt und Lesetempo

- Untersuchungen zur Augenbewegung seit Émile Javal (1879):
  - **Fixationen** (ca. 250 ms) und
    Sakkaden (Sprünge, ca. 15 ms, 7 - 9 Zeichen)
  - Fixation: ca. 3 Buchstaben links und 14 Buchstaben rechts

Textwahrnehmung



❶ Fixation (Blickpunkt)
❷ Sakkade (Vorsprung)
❸ Regression (Rücksprung)
❹ Sakkade (Zeilenwechsel)

Texterkennung

- **Wortüberlegenheitseffekt**
  - In 10 ms werden erkannt:
    - maximal vier einzelne Buchstaben
    - zwei ganze Wörter
  - Wörter werden als **Bilder** gespeichert und dann identifiziert
- **Verständnis im Leseprozess integriert:**
  - Vorwissen ermöglicht schnelleres Lesen (kurze Fixationszeiten)
  - Typografische und orthografische Fehler verlangsamen Lesetempo
- **Satzabschlusseffekt**
  - Längere Fixation am Satzende
- **Konsequenzen**
  - Typografie (Bilddarstellung der Worte) ist wichtig!
  - Klare Führung für das Auge
  - Optisch deutliche Wortgrenzen und Textstruktur

Beispiel zur Wortüberlegenheit

Gmäeß eneir Sutide eneir elgnihcesn Uvenisterät ist es nchit witihcg in wlecehr Rneflogheie die Bstachuebn in eneim Wrot snid, das ezniige was wcthiig ist, ist dass der estre und der leztte Bstabchue an der ritihcegn Pstoiion snid. Der Rset knan ein ttoaelr Bsinöldn sien, tedztorm knan man ihn onhe Pemoblre lseen.

(Kursiert vielfach im Web; Originalquelle unbekannt)

Achtung, die genannte Studie existiert nicht!

- Die im Text gemachte Aussage ist auch nicht belegbar.
- Der erlebbare Effekt beruht auf Wortüberlegenheit zusammen mit inhaltlicher Vorhersagbarkeit.

Zeichen, Schriftart und Glyphen

- **Zeichen** = Element eines Zeichenvorrats
- **Zeichensatz** (character code) = Abbildung zwischen Zeichen und ganzen Zahlen (**Kodierung**)
- **Schriftart** (type face, font) = Genau festgelegte bildliche Darstellungen (**Glyphen**) für alle Zeichen des Zeichenvorrats
- **Beispiel:** Zeichen = ‚H' = ASCII-Zeichen Nr. 72

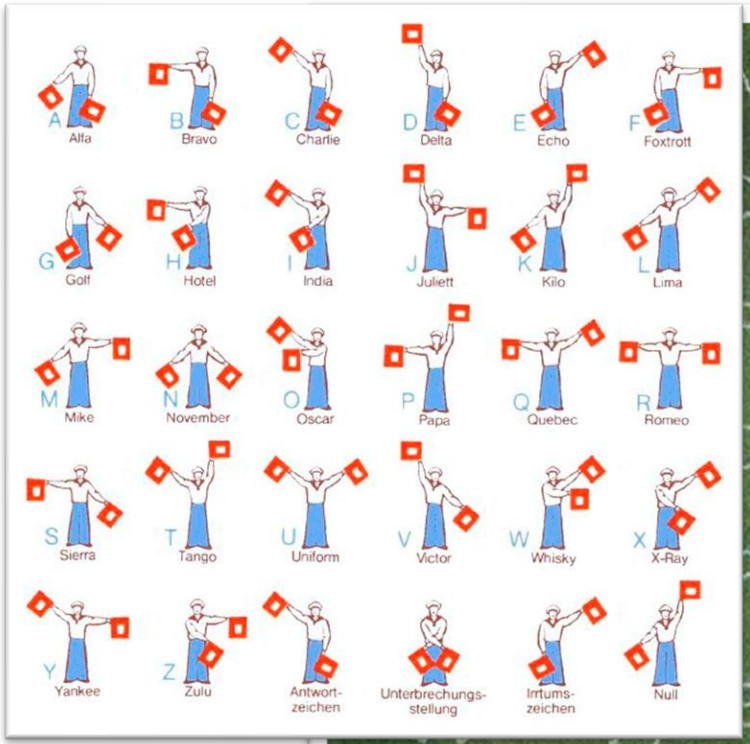
Schriftart „Times“ H Schriftart: „Helvetica“ H

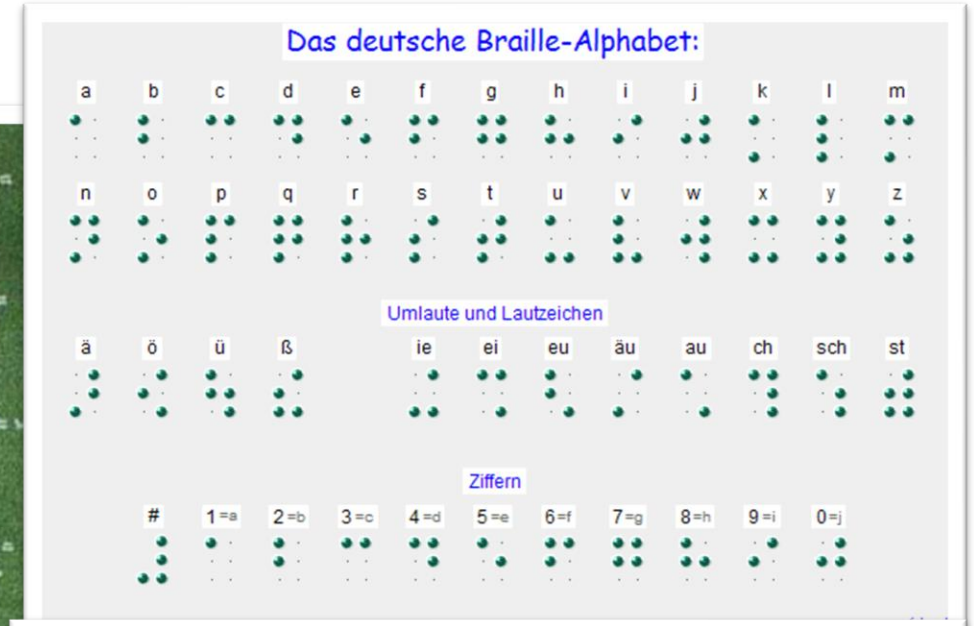
- Glyphen hochwertiger Schriftarten berücksichtigen viele Erkenntnisse der Gestaltungslehre.
  - Beispiel: „Optischen Mitte“ Liegt bei Fonts ca. 3% über der geometrischen Mitte

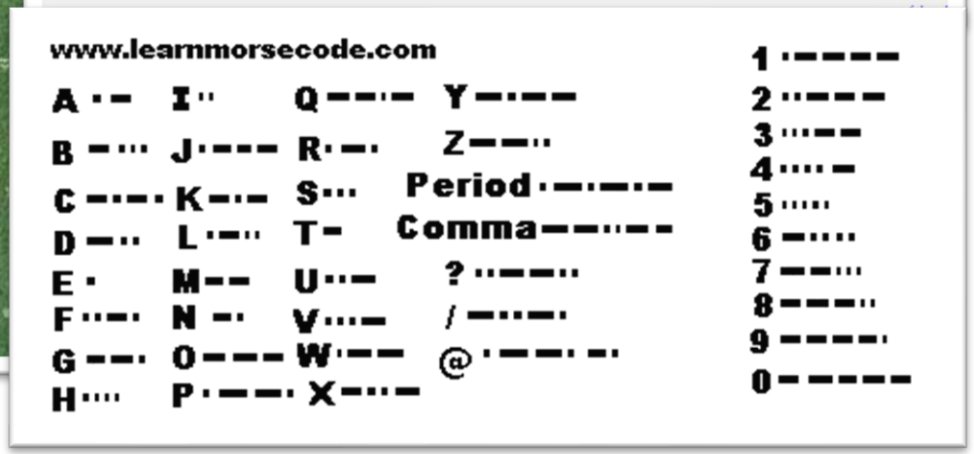
geometrische vs. optische Mitte

KODIERUNG UND BINÄRDARSTELLUNG

Kodierungen







http://crypto.over-blog.fr

Wertedarstellung

- Allgemeine Formel zu einer Basiskonvertierung

$$Z = \sum_{i=n}^{m} z_i b^i$$

$$n,\ m\ \in \mathbb{Z}$$

- Die Basis b ist z.B.
  - Binär: 2 (0,1)
  - Oktal: 8 (0-7)
  - Dezimal: 10 (0-9)
  - Hexadezimal: 16 (0-F)

Binäre Zahlendarstellung

- Bekannte Größe: 1 Byte = 8 Bit
  d.h. jedes Byte repräsentiert eine Zahl zwischen (inklusive) 0 und (inklusive) 255

$$Z = \sum_{i=0}^{7} z_i 2^i$$

$$z_i \in \{0, 1\}$$

- Jede Zahl kann einem Zeichen zugewiesen werden, siehe ASCII-Tabelle

H	a	l	l	o
01001000	01100001	01101100	01101100	01101111

B	a	l	l	o
01000010	01100001	01101100	01101100	01101111

Beispiel: ASCII-Zeichensatz

- **ASCII** („**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange“) ist ein 7-Bit-Code, der jedem 7-stelligen Binärwort eindeutig ein druckbares Zeichen zuordnet
- Daraus resultiert ein Alphabet für die Darstellung von Zeichensätzen (Buchstaben, Ziffern, Sonderzeichen)

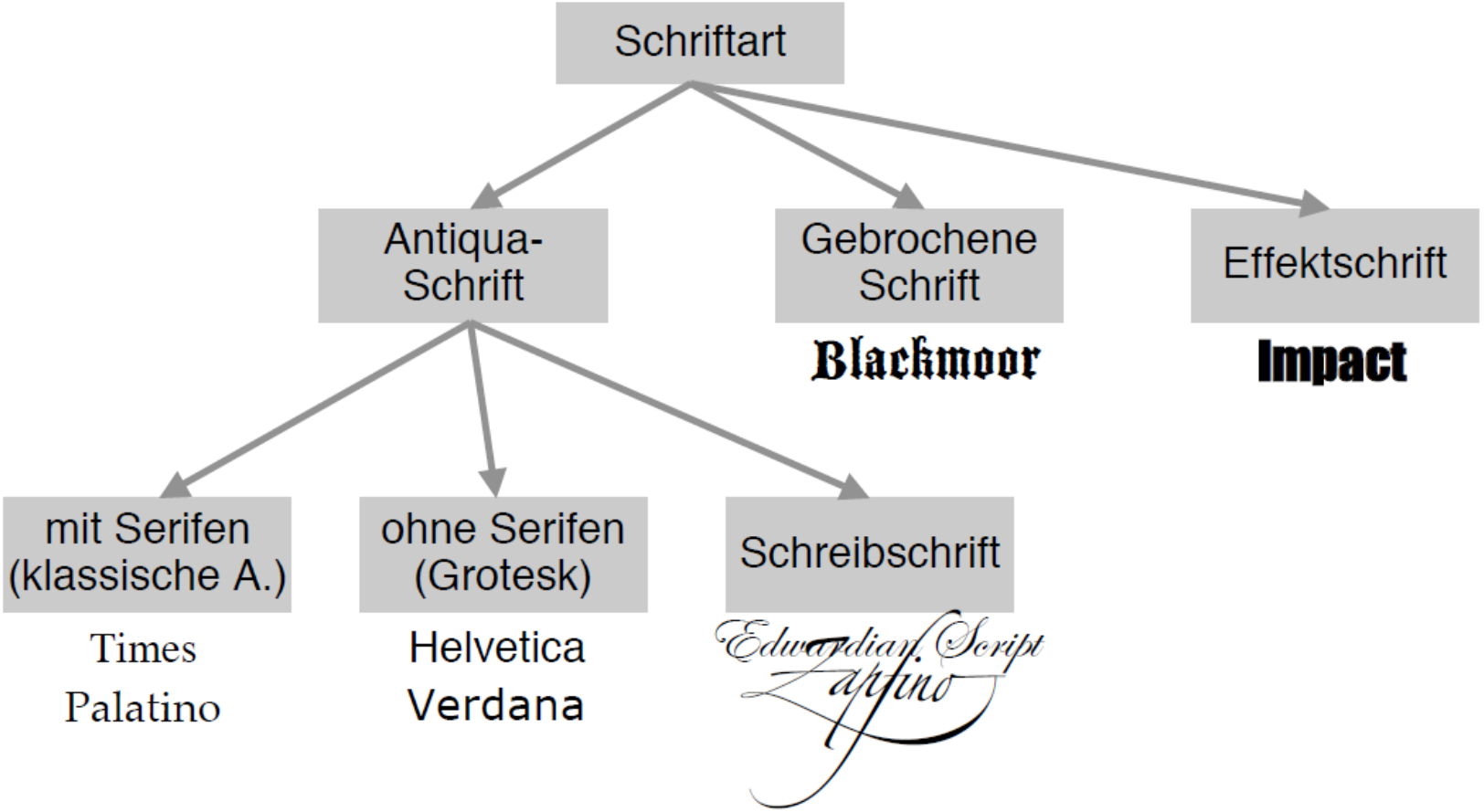
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	010		!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3	011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	110	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	N	o
7	111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Beispiel: Unicode Schriftfamilien (Ausschnitt)

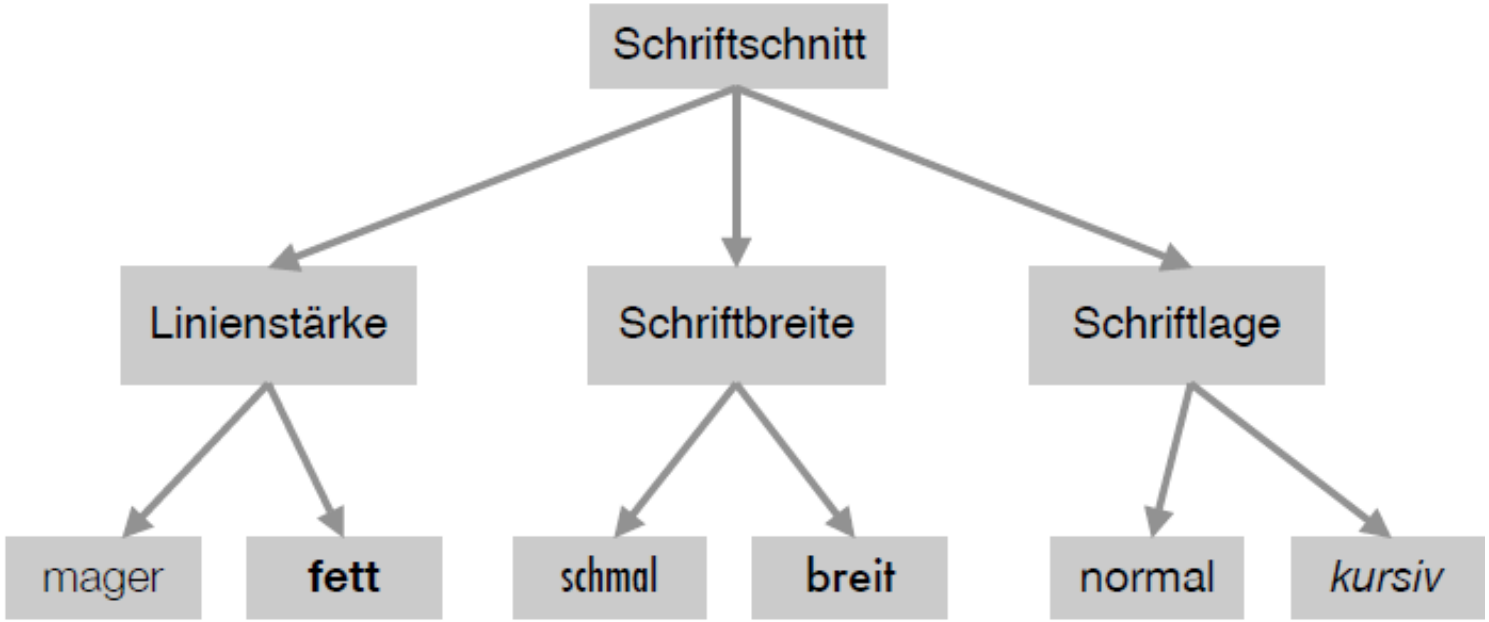
European Scripts	African Scripts	South Asian Scripts	Indonesia & Oceania Scripts
Armenian	***Bamum***	***Ahom***	***Balinese***
Armenian Ligatures	Bamum Supplement	***Bengali and Assamese***	***Batak***
Caucasian Albanian	***Bassa Vah***	***Brahmi***	***Buginese***
Cypriot Syllabary	***Coptic***	***Chakma***	***Buhid***
Cyrillic	*Coptic in Greek block*	***Devanagari***	***Hanunoo***
Cyrillic Supplement	Coptic Epact Numbers	Devanagari Extended	***Javanese***
Cyrillic Extended-A	***Egyptian Hieroglyphs (1MB)***	***Grantha***	***Rejang***
Cyrillic Extended-B	***Ethiopic***	***Gujarati***	***Sundanese***
Elbasan	Ethiopic Supplement	***Gurmukhi***	Sundanese Supplement
Georgian	Ethiopic Extended	***Kaithi***	***Tagalog***
Georgian Supplement	Ethiopic Extended-A	***Kannada***	***Tagbanwa***
Glagolitic	***Mende Kikakui***	***Kharoshthi***	**East Asian Scripts**
Gothic	***Meroitic***	***Khojki***	***Bopomofo***
Greek	Meroitic Cursive	***Khudawadi***	Bopomofo Extended
Greek Extended	Meroitic Hieroglyphs	***Lepcha***	***CJK Unified Ideographs (Han) (35MB)***
Ancient Greek Numbers	***N'Ko***	***Limbu***	CJK Extension-A (6MB)
Latin	***Osmanya***	***Mahajani***	CJK Extension B (40MB)
Basic Latin (ASCII)	***Tifinagh***	***Malayalam***	CJK Extension C (3MB)
Latin-1 Supplement	***Vai***	***Meetei Mayek***	CJK Extension D
Latin Extended-A	**Middle Eastern Scripts**	Meetei Mayek Extensions	CJK Extension E (3.5MB)
Latin Extended-B	***Anatolian Hieroglyphs***	***Modi***	(see also Unihan Database)
Latin Extended-C	***Arabic***	***Mro***	***CJK Compatibility Ideographs***
Latin Extended-D	Arabic Supplement	***Multani***	CJK Compatibility Ideographs Supplement
Latin Extended-E	Arabic Extended-A	***Ol Chiki***	***CJK Radicals / KangXi Radicals***
Latin Extended Additional	Arabic Presentation Forms-A	***Oriya (Odia)***	CJK Radicals Supplement
Latin Ligatures	Arabic Presentation Forms-B	***Saurashtra***	CJK Strokes
Fullwidth Latin Letters	***Aramaic, Imperial***	***Sharada***	Ideographic Description Characters
IPA Extensions	***Avestan***	***Siddham***	
Phonetic Extensions	***Carian***	***Sinhala***	
Phonetic Extensions Supplement		Sinhala Archaic Numbers	

Schriftarten



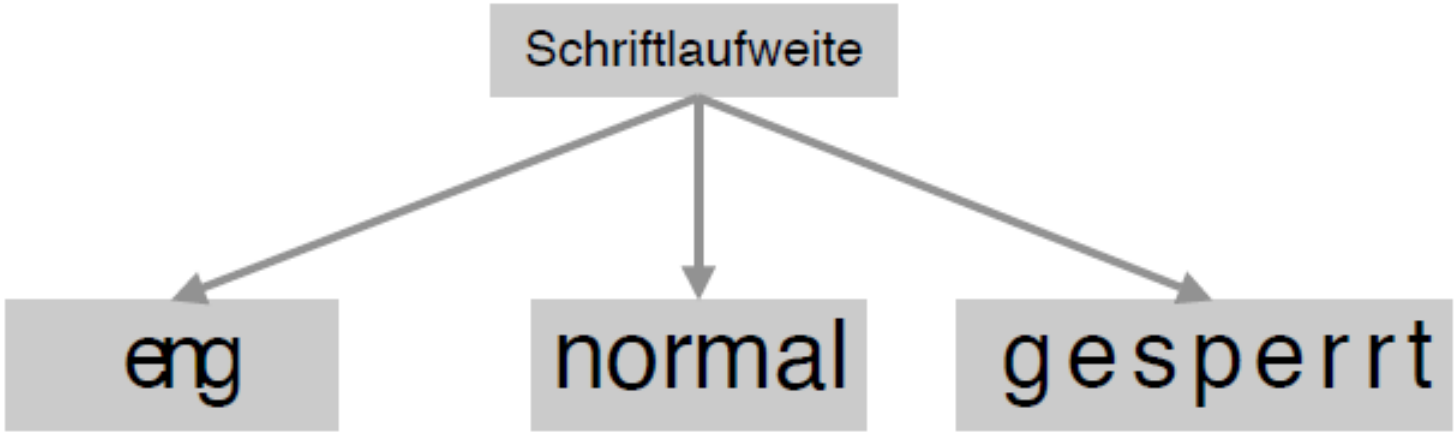
- **Antiqua:** Schriftarten mit gerundeten Bögen, die auf dem lateinischen Alphabet basieren und sich ursprgl. auf Vorbilder der römischen Antike bezogen
- Feinere Unterscheidung der klassischen Antiqua-Schriften nach DIN
  - z.B. Palatino = Französische Renaissance-Antiqua, Times = Barock-Antiqua

Schriftschnitt



- Weitere Schnitte im professionellen Satz, z.B. halbfett
- Gute Schriften existieren oft als **Schriftfamilien** (z.B. Univers: 21 Schnitte)
- Zahlen nach dem Schriftnamen (z.B. Univers 55) bezeichnen die Linienstärke.
- **Hybridschriften** haben Varianten mit und ohne Serifen (z.B. Lucida)

Laufweite

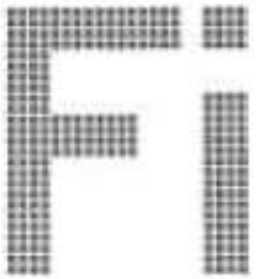


- „Normal“ = Abstand der Glyphen gemäß Dickte

- Proportionalschrift: Dickte individuell je Zeichen
- **Nicht-Proportionalschrift: Dickte gleich für alle Zeichen**

Bitmap- und Vektor-Schriften

- **Bitmap-Schrift** (-Font)
  - Glyphe ist aus einzelnen Punkten aufgebaut
  - Farbe vorgegeben
  - Spezifische Größe (und Auflösung)



- **Vektor-Schrift** (-Font)
  - Glyphe ist als graphische Kontur gegeben
    - Punkte und Verbindungslinien
  - Füllung durch beliebige Farben möglich
  - Skalierbar auf beliebige Größe

Quellen

Falls nicht anders angegeben:

(1) Joachim Böhringer, Peter Bühler, Patrick Schlaich:
Kompendium der Mediengestaltung: Konzeption und Gestaltung für Digital- und Printmedien (X.media.press); Springer, 2008.

(2) Joachim Böhringer, Peter Bühler, Patrick Schlaich:
Kompendium der Mediengestaltung: Produktion und Technik für Digital- und Printmedien (X.media.press); Springer, 2008.

(3) Ulrich Schmidt:
Professionelle Videotechnik: Grundlagen, Filmtechnik, Fernsehtechnik, Geräte- und Studiotechnik in SD, HD, DI, 3D; Springer, 2009.

(4) Rainer Malaka, Andreas Butz, Heinrich Hußmann:
Medieninformatik: Eine Einführung; Pearson Studium, 2009.

(5) Vorlesungsmaterialien „Digitale Medien“:
Prof. Dr. Heinrich Hußmann, LMU München, 2009/10.

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT

Eingebettete Systeme

Grundlagen der Multimediatechnik

Informationstheorie und Textkompression

16.11.2020, Prof. Dr. Enkelejda Kasneci

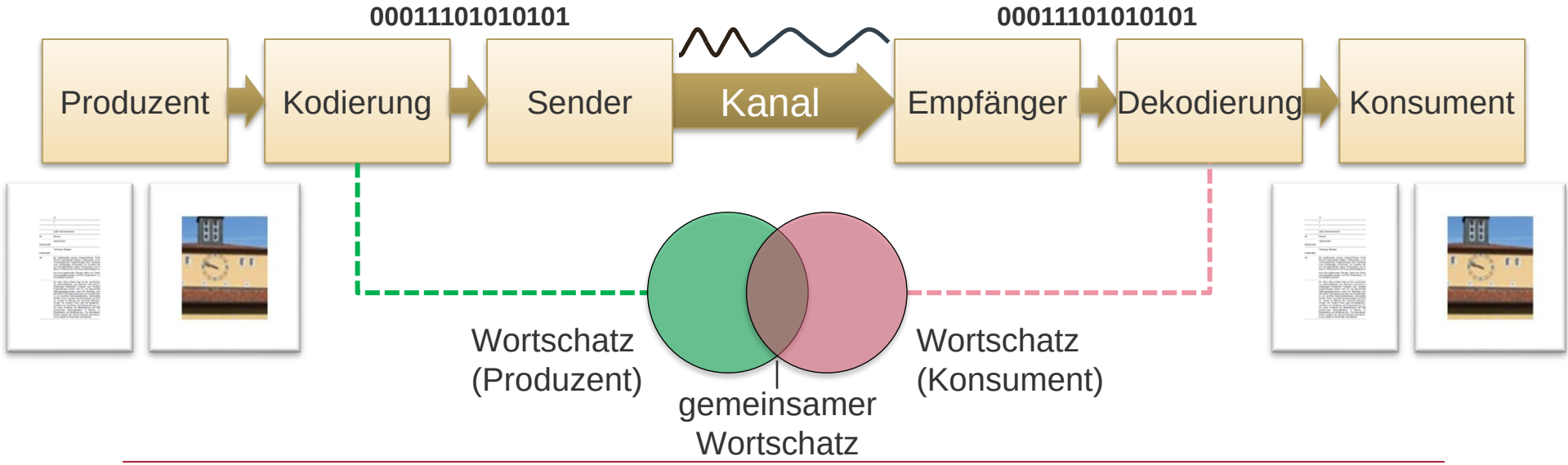
Übersicht

- Informationstheorie und Entropie
- Kodierung & Binärdarstellung
- Textkomprimierung
  - Lauflängenkodierung (RLE)
  - Huffman-Kodierung
  - Burrows-Wheeler

INFORMATIONSTHEORIE

Informationstheorie

- **Informationstheorie:** Systeme, Modelle und Methoden zur **Beschreibung der Entstehung und Übermittlung von Daten**
- Informationen können nur dann sinnvoll weitergegeben werden, wenn zwischen **Produzent** und **Konsument Abmachungen** über **Syntax**, **Semantik** und **Pragmatik** getroffen wurden oder bestehen (vgl. Einführung).
  - **Syntax:** Festlegung der zugelassenen Zeichen und Zeichenfolgen
  - **Semantik:** Beschreibung der Bedeutung der Zeichen und Zeichenfolgen
  - **Pragmatik:** Bedeutung (für den Menschen) in der Anwendung



Information als messbare Größe

- Betrachten wir dazu das Beispiel einer Datenleitung:
  0001110101010111000110100110001010000111010101001001001001010101….

- Was kann beobachtet werden?
  - Welche Zeichen treten auf?
    Zeichenvorrat, Alphabet der Quelle, hier „0“ und „1“
  - Wie häufig treten die Zeichen auf?
    Schätzwerte für die Wahrscheinlichkeiten, hier $P(0)$ und $P(1)$
  - Wie häufig treten Kombinationen von Zeichen, z. B. „00“ oder „010“, auf?
    Schätzwerte für die Verbundwahrscheinlichkeiten,
    z. B. $P(0,0)$, $P(0,1,0)$ *usw.*

Modell: Diskrete gedächtnislose Quelle

- Eine **diskrete gedächtnislose Quelle** X setzt in jedem Zeittakt ein Zeichen x_i aus dem Zeichenvorrat, dem Alphabet $X = \{x_1, x_2, \ldots, x_N\}$, mit der Wahrscheinlichkeit $P(x_i) = p_i$ ab. Die Auswahl der Zeichen geschieht unabhängig voneinander.

- **Beispiel: gedächtnislose Binärquelle**
  - Zeichenvorrat: $X = \{x_1, x_2\}$
  - Wahrscheinlichkeiten: $0 \leq p_1 \leq 1$ und $p_2 = 1 - p_1$

- Fragestellung: Wie kann man mit möglichst **wenig Aufwand** an Symbolen bzw. Zeichen **möglichst viel Information übertragen**?

Informationsgehalt eines Zeichens

Axiomatische Definition:

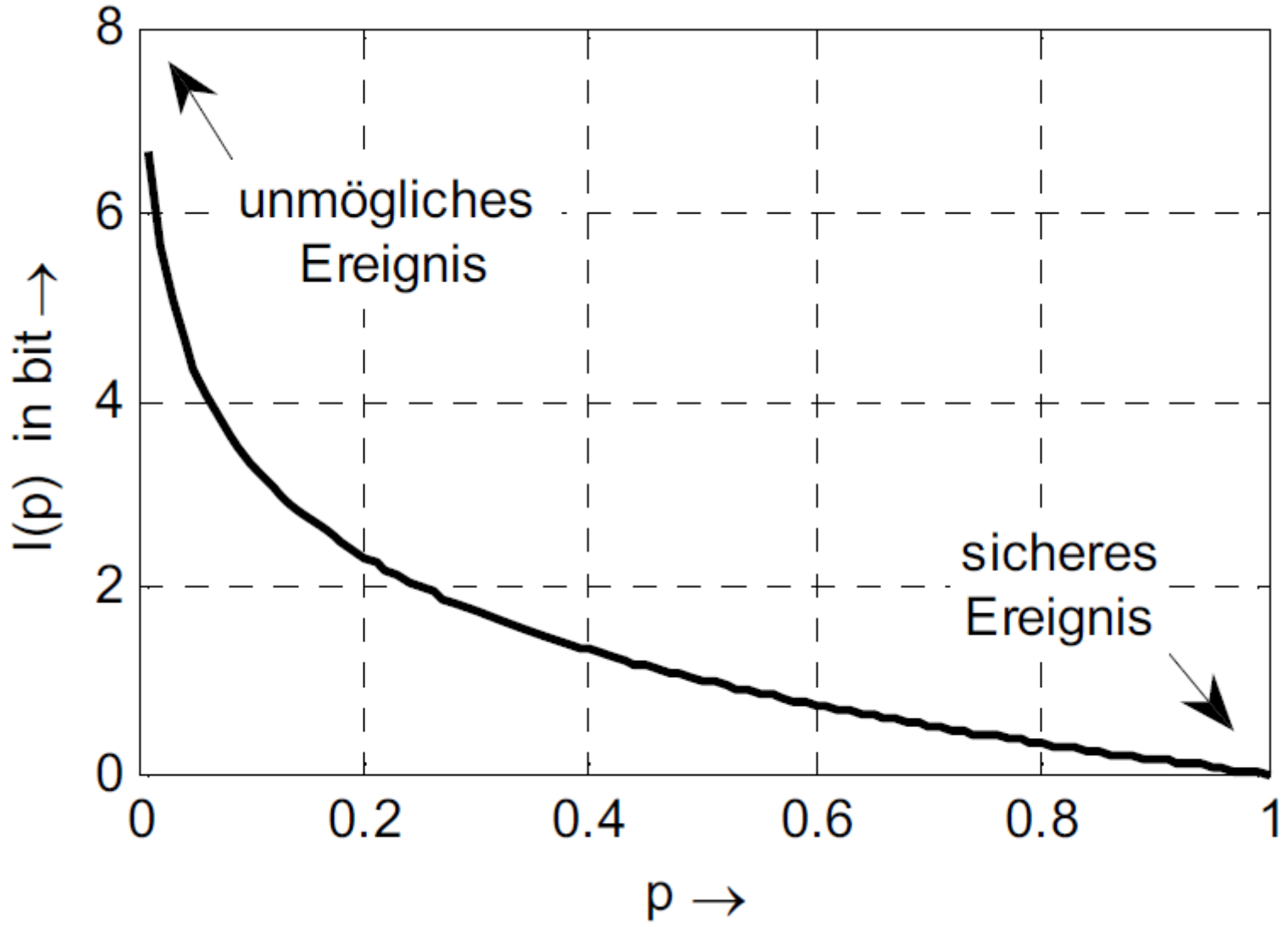
1. Der Informationsgehalt eines Zeichens $x_i \in X$ mit der Wahrscheinlichkeit p_i ist ein nicht-negatives Maß, das nur von der Wahrscheinlichkeit des Zeichens abhängt:
   $$I(p_i) \geq 0$$
2. Die jeweiligen Informationsgehalte eines unabhängigen Zeichenpaares (x_i, x_j) mit der Verbundwahrscheinlichkeit $P(x_i, x_j) = p_i \cdot p_j$ addieren sich:
   $$I(p_i, p_j) = I(p_i) + I(p_j)$$
3. Der Informationsgehalt ist eine stetige Funktion der Wahrscheinlichkeiten der Zeichen.

Definition: Der **Informationsgehalt eines Zeichens** mit der Wahrscheinlichkeit p einer gedächtnislosen Binärquelle lautet:

$$I(p) = \log_2(1/p)\ bit = -\log_2(p)\ bit$$

Dabei ist $1\ bit$ der Informationsgehalt, der in einer Auswahl aus zwei gleich wahrscheinlichen Möglichkeiten enthalten ist (gedächtnislose Binärquelle mit Wahrscheinlichkeit $p_1 = 0{,}5$).

Informationsgehalt eines Zeichens



Entropie

Informationsgehalt bezieht sich auf ein einzelnes Zeichen
→ Aussage über eine Datenquelle ist notwendig!

Definition: Eine diskrete gedächtnislose Quelle X mit dem Zeichenvorrat $X = \{x_1, x_2, \dots, x_N\}$ und den zugehörigen Wahrscheinlichkeiten $p_1, p_2, \dots, p_N$ besitzt den **mittleren Informationsgehalt**, die ***Entropie*** **[Shannon, 1948]**

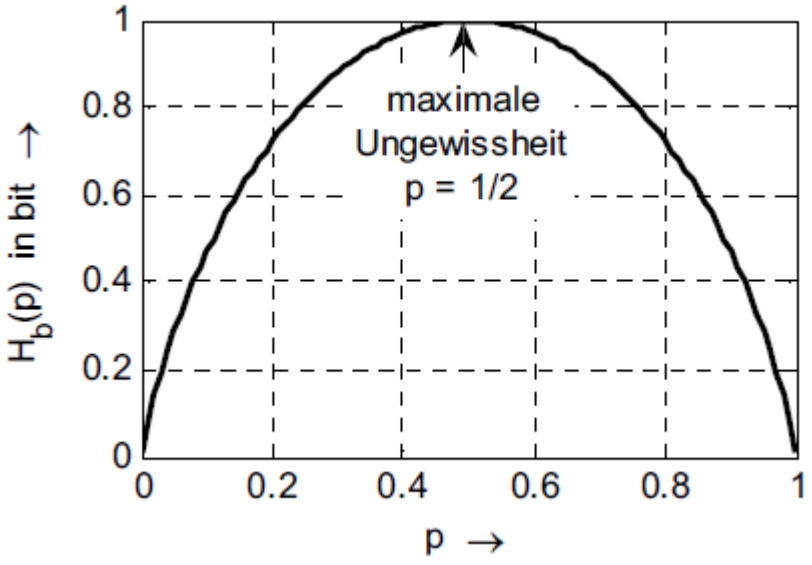
$$H(X) = -\sum_{i=1}^{N} p_i \cdot \log_2(p_i)\ bit$$

Beispiel:

Zeichen	a	b	c	d
p_i	$1/2$	$1/4$	$1/8$	$1/8$
$I(p_i)$	$1\ bit$	$2\ bit$	$3\ bit$	$3\ bit$
$H(X)$	$0{,}5 \cdot 1\ bit + 0{,}25 \cdot 2\ bit + 0{,}125 \cdot 3\ bit + 0{,}125 \cdot 3\ bit = 1{,}75\ bit$			

Entscheidungsgehalt

Definition: Die **Entropie** einer diskreten gedächtnislosen Quelle wird **maximal**, wenn alle N Zeichen des Zeichenvorrats **gleichwahrscheinlich** sind.
Der **Maximalwert** der **Entropie** ist gleich dem **Entscheidungsgehalt** des Zeichenvorrats: $H_0 = \log_2 N \; bit$



Entropie der Binärquelle

Redundanz

- **Definition:** Die Differenz zwischen Entropie einer Quelle und dem Entscheidungsgehalt ihres Zeichenvorrats nennt man **Redundanz**.

$$R = H_0 - H(X)$$

- **Beispiel** (vorvorherige Folie):

$$H(X) = 1{,}75\ bit$$
$$H_0 = \log_2 4 = 2\ bit$$
$$R = 2\ bit - 1{,}75\ bit = 0{,}25\ bit$$

→ Nachrichtenquelle enthält „gewisses Maß“ an Ordnung

→ Zeichen lassen sich „geringfügig“ vorhersagen

- Bei einer Quelle mit fester Zeichen-Auftrittswahrscheinlichkeit ist das Redundanzmaß basierend auf Entscheidungsgehalt nicht zielführend

Durchschnittliche Wortlänge und Redundanz

- **Definition:** Die **durchschnittliche Wortlänge $\bar{L}$** beschreibt die nach Auftrittswahrscheinlichkeiten gewichtete Summe der jeweiligen Wortlängen $|c(x_i)|$ einer Codierung $c(x_i)$ eines Einzelzeichens x_i

$$\bar{L} = \sum_{i=1}^{N} p_i \cdot |c(x_i)|$$

- **Beispiel:**

Zeichen	a	b	c	d
p_i	$1/2$	$1/4$	$1/8$	$1/8$
Kodier. C_1	00	01	10	11
$\bar{L}_{C_1}$	$0{,}5 \cdot 2\ bit + 0{,}25 \cdot 2\ bit + 0{,}125 \cdot 2\ bit + 0{,}125 \cdot 2\ bit = 2\ bit$			
Kodier. C_2	0	10	110	111
$\bar{L}_{C_2}$	$0{,}5 \cdot 1\ bit + 0{,}25 \cdot 2\ bit + 0{,}125 \cdot 3\ bit + 0{,}125 \cdot 3\ bit = 1{,}75\ bit$			

- **Redundanz:** $R = \bar{L} - H(X)$

  $R_{C_1} = 2\ bit - 1{,}75\ bit = 0{,}25\ bit$ und $R_{C_2} = 1{,}75\ bit - 1{,}75\ bit = 0\ bit$

KODIERUNG UND KOMPRESSION

Kodierung und Kompression
Kategorien

- **Entropiekodierung**
  - Keine Berücksichtigung der Datensemantik
  - Verlustfreie Kompression
- **Quellenkodierung**
  - Berücksichtigung der Datensemantik
  - Verlustfreie oder verlustbehaftete Kompression
  - Domänen-Transformation
- **Kanalkodierung**
  - Aufbereiten der Daten für Kompressionskanal / Übertragung
  - Einführung von Redundanz(!), z.B. ECC, Viterbi, Reed-Solomon
- **Hybride Kodierung**
  - Kombination unterschiedlicher Kodierungsverfahren

Kodierung und Kompression
Kategorien und Techniken

Entropiekodierung	Lauflängenkodierung (RLE) Huffman-Kodierung Arithmetische Kodierung	
Quellenkodierung	Prädiktion	DPCM DM
	Transformation	FFT DCT
	Layered Coding	Bit Position Subsampling Subband Coding
	Vektor-Quantisierung	
Hybride Kodierung	JPEG MPEG H.26x Proprietäre Formate, z.B. QuickTime	

Kodierung und Kompression
Lauflängenkodierung

- **Lauflängenkodierung: Run-length Encoding (RLE)**
- **Annahme: Häufige Abfolge identischer Symbole**
  - Beispiel: Schwarzer Text auf weißem Hintergrund

  → wwwwwwwwwwwwBwwwwwwwwwwwwBBBwwwwww
  wwwwwwwwwwwwwwwwwwwwwwwwwwBwwwwwwwww
- **Idee:** Ersetze Folge gleicher Zeichen durch 1 Zeichen + Zähler
- **Kodierung einer Rekonstruktionsvorschrift**
  - 12 weiße, 1 schwarzer, 12 weiße, 3 schwarze etc. Pixel

  → 12w**1B**12w**3B**25w**1B**14w

  → 18 Zeichen statt 68
- **Verlustfreies Kompressionsverfahren**
  - Kein Datenverlust durch Kompression
  - Entfernen redundanter Information durch Ersetzen der Daten durch Rekonstruktionsvorschrift
  - **Syntaktische Trennung von Wiederholungsindikatoren** und unveränderten Zeichen notwendig → entweder **Trennzeichen** (z.B.: #) oder **Kodierung in Maschinenworten** (z.B.: 1 Byte Zeichen, 1 Byte Zähler)
  - Bei geringer Häufigkeit von Wiederholungen ineffizient (verschlechternd)

  → Verallgemeinerung des Verfahrens: „Dictionary Coders“

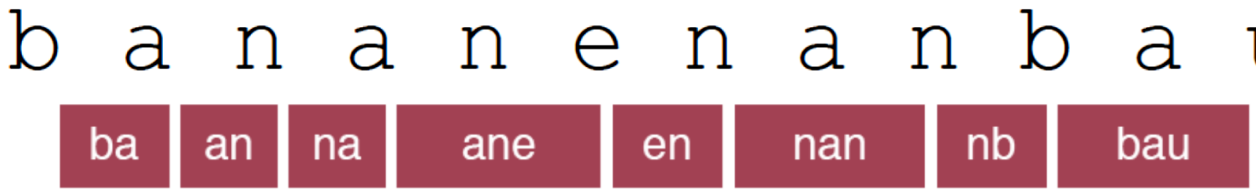
Kodierung und Kompression
Sequenzkodierung

- **Ermitteln von Sequenzen** in Zeichenfolge
- **Ablegen von wiederholten Sequenzen** in Wörterbuch
- **Ersetzen wiederholter Sequenzen durch Zeiger** auf Wörterbucheintrag
- **Erweitern** der **Wörterbucheinträge**

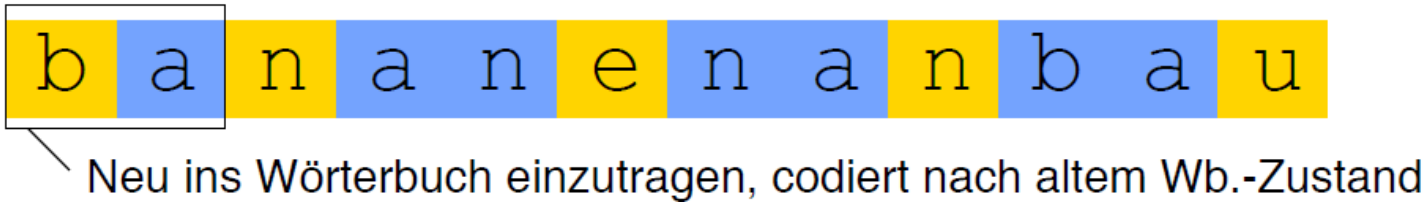
- **Praktische Algorithmen:**
  - Abraham Lempel, Jacob Ziv (Israel), Ende 70er-Jahre
    - **LZ77- und LZ78-Kodierung**
  - Verbessert 1984 von A. Welch = **LZW-Kodierung** (Lempel/Ziv/Welch)
  - Basis vieler semantikunabhängiger Kompressionsverfahren (z.B. UNIX „compress“, Zip, gzip, V42.bis)
  - Verwendet in vielen Multimedia-Datenformaten (z.B. GIF)

Limpel-Ziv-Welch-Kodierung (LZW)

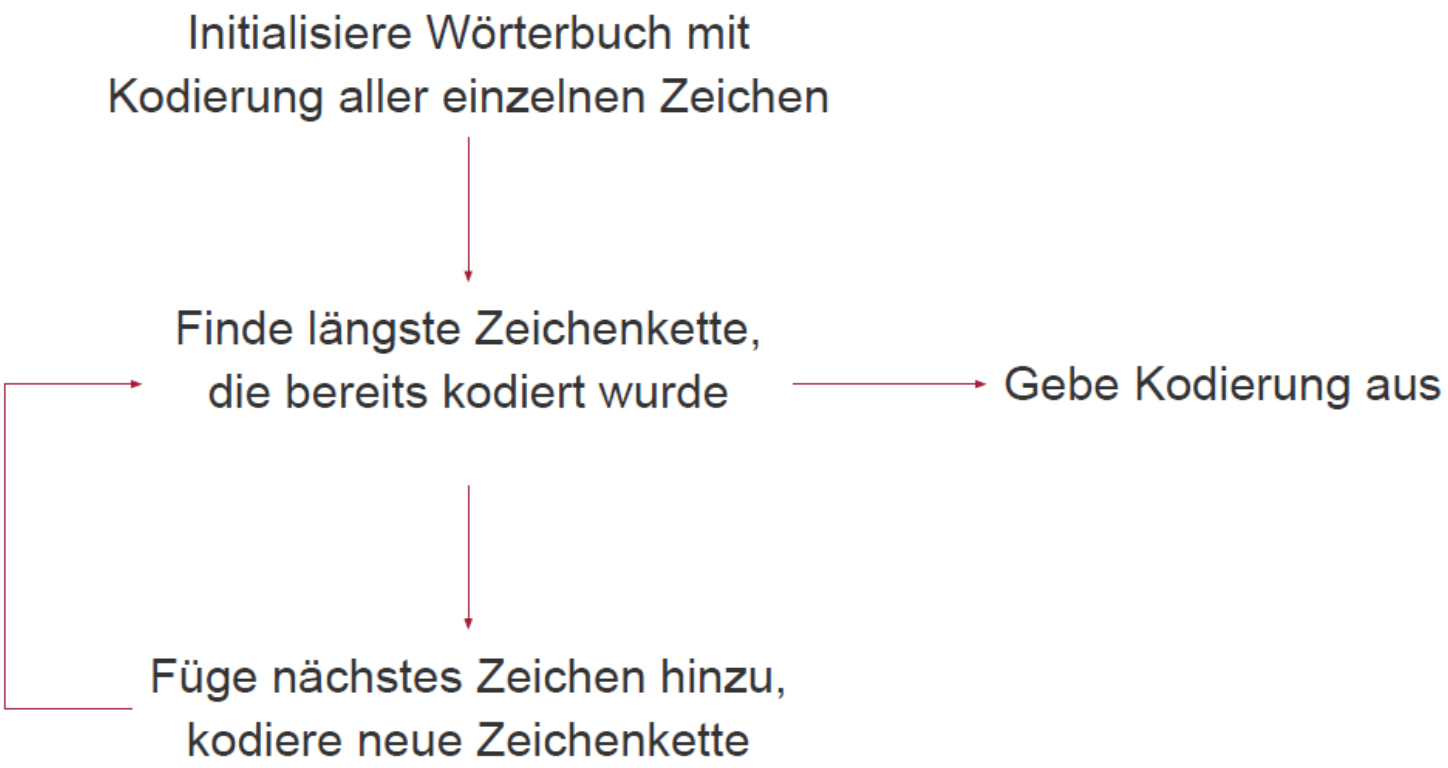
- Nicht alle Teilworte ins Wörterbuch, sondern nur eine „Kette“ von Teilworten, die sich um jeweils ein Zeichen überschneiden
- Sequentieller Aufbau:
  - Neu einzutragendes Teilwort = Kürzestes („erstes“) noch nicht eingetragenes Teilwort
- **Beispiel:**



- Codierung



Limpel-Ziv-Welch-Kodierung (LZW) Algorithmus



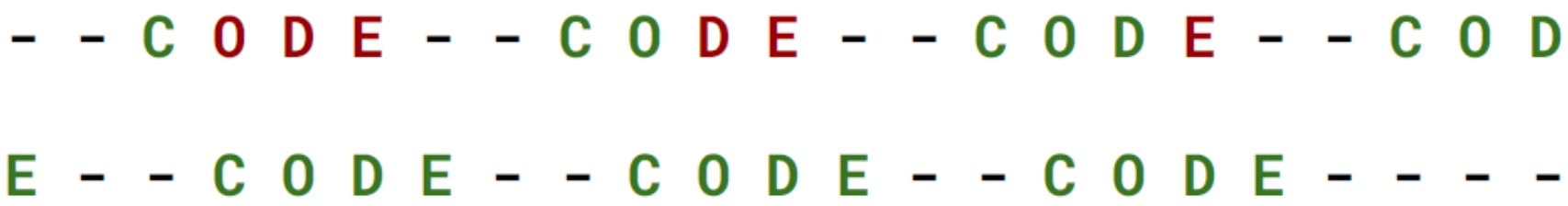
LZW-Kodierung
Beispiel: bananenanbau

Lese Zeich.	Neuer Wörterbucheintrag	Ausgabe	akt. Puffer
b (98)			"b"
a (97)	("ba", 256)	98 ("b")	"a"
n (110)	("an", 257)	97 ("a")	"n"
a (97)	("na", 258)	110 ("n")	"a"
n (110)			"an"
e (101)	("ane", 259)	257 ("an")	"e"
n (110)	("en", 260)	101 ("e")	"n"
a (97)			"na"
n (110)	("nan", 261)	258 ("na")	"n"
b (98)	("nb", 262)	110 ("n")	"b"
a (97)			"ba"
u (117)	("bau", 263)	256 ("ba")	"u"
EOF		117 ("u")	

LZW-Kodierung Effektivität

- Eigenschaften die LZW nutzt:
  - Symbol-Häufigkeit
  - Symbol-Wiederholungen
  - Wiederholende Muster

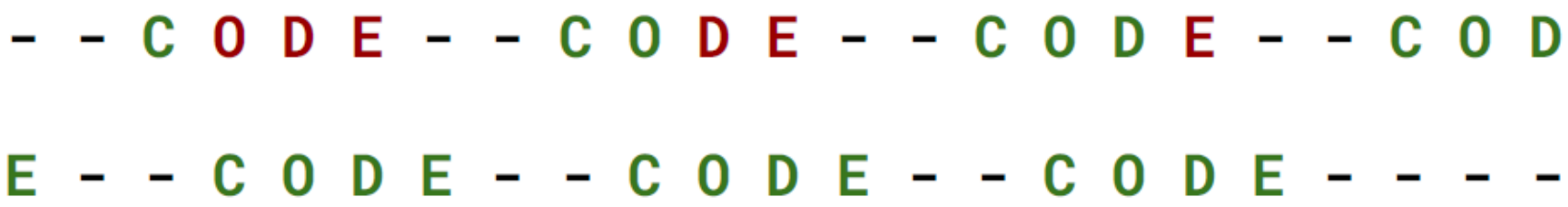
- Eigenschaft, die LZW **nicht** nutzt:
  - Positionsbedingte Redundanz



LZW-Kodierung Effektivität

- Eigenschaften die LZW nutzt:
  - Symbol-Häufigkeit
  - Symbol-Wiederholungen
  - Wiederholende Muster

- Eigenschaft, die LZW **nicht** nutzt:
  - Positionsbedingte Redundanz



Vergleich der Kompresssionsansätze

LZW	Huffman	RLE
Nutzt Häufigkeit von Symbol-Kombinationen aus	Nutzt Symbol-Häufigkeiten besser aus, aber scheitert an Häufigkeiten von Kombinationen (ZE, GC)	
Wörterbuch ist immer gleich und muss nicht in Kodierung enthalten sein	Muss für jede Quelle einen neuen Huffman-Baum erstellen	
Robust gegenüber der Eingabe		Nutzt Symbol Wiederholungen besser aus, aber ist in z.B in Texten nicht praktikabel
Ist bei hinreichend langer Eingabe effizienter		

Kodierung und Kompression
Unterstützung der RLE durch Transkodierung

- **RLE benötigt gleichbleibende Zeichen** oder Zeichensequenzen
- **Transkodierung der Eingabedaten**
  - **Beispiel:** Delta-Codierung von 2,4,6,8,9 → 2,2,2,2,1
    - Original nicht mittels RLE komprimierbar
    - Delta-kodiertes Signal mittels RLE komprimierbar: (2222,1)

  → einfache Hybridverfahren

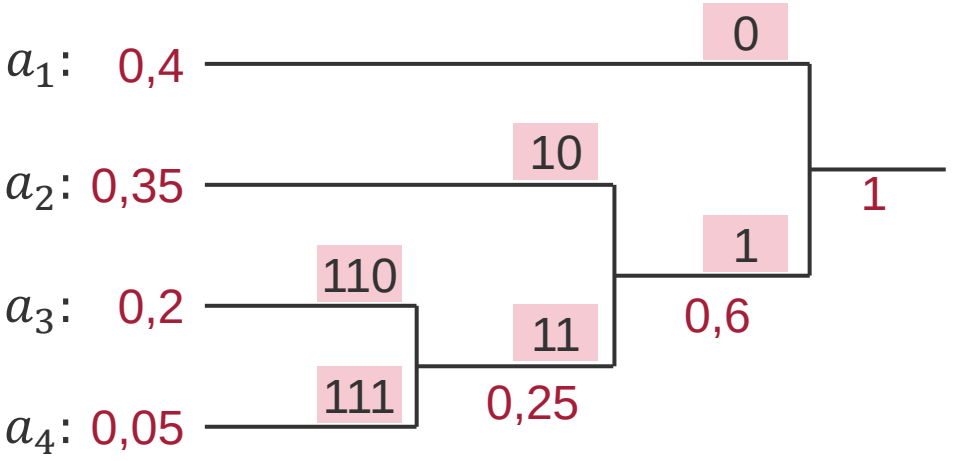
Huffman-Kodierung

- Berücksichtigung der **Häufigkeit von Symbolen**
  - häufige Symbole → kurze Repräsentation
  - seltene Symbole → lange Repräsentation

  → Transformation des Alphabets
- **Beispiel: Morse-Code**
  - e am häufigsten → kurze Kodierung durch ·
  - y eher selten → längere Kodierung durch – – · ·

Kodierung und Kompression
Huffman-Kodierung

- **Gegeben**: Sequenz aus 4 Zeichen a_1, a_2, a_3, a_4 mit $p(a_1, a_2, a_3, a_4) = 0{,}4;\ 0{,}35;\ 0{,}2;\ 0{,}05$
- **Aufbau eines Kodierungsbaums unter Berücksichtigung der Wahrscheinlichkeiten** des **Auftretens der Zeichen**
  - Beginn bei den Blättern (Zeichen)
  - Iterative Konstruktion des Baumes
  - Bewertung der Kanten und Ableitung des Codes



Symbol	Code
a_1	0
a_2	10
a_3	110
a_4	111

Huffman-Kodierung

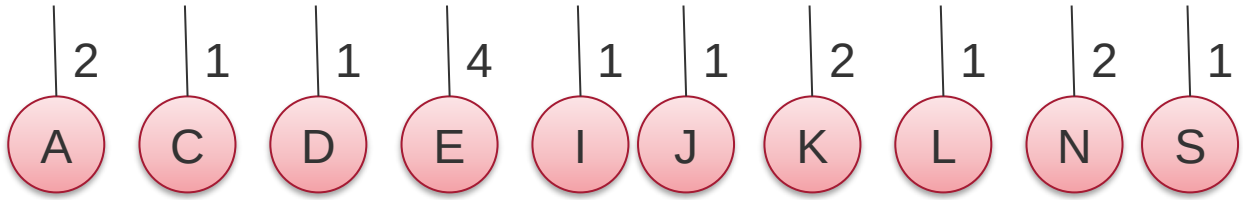
- Anstatt eine Bitfolge zu speichern, wird der Weg zum Wert in einem Baum gespeichert
- **Vorgehensweise:**
  - Gegeben: Wortmenge $M = \{w \in D\}$
    mit Häufigkeit $d(w) = |w \in D|$
  - **Erstelle Wald von Bäumen** mit jeweils einem Knoten, die Kante wird mit der Häufigkeit markiert.
  - **Fasse jeweils kleinste Häufigkeiten zusammen**, die Kante erhält die Summe der Häufigkeiten der beiden Subknoten
  - Dies wird fortgesetzt, **bis es nur noch einen Baum** gibt
- Um nun einen Wert auszulesen, muss nur der Pfad angegeben werden
  - **1 für den rechten Knoten, 0 für den linken Knoten**

Huffman-Kodierung - Beispiel

- Zu Kodieren: Wortfolge ENKELEJDAKASNECI

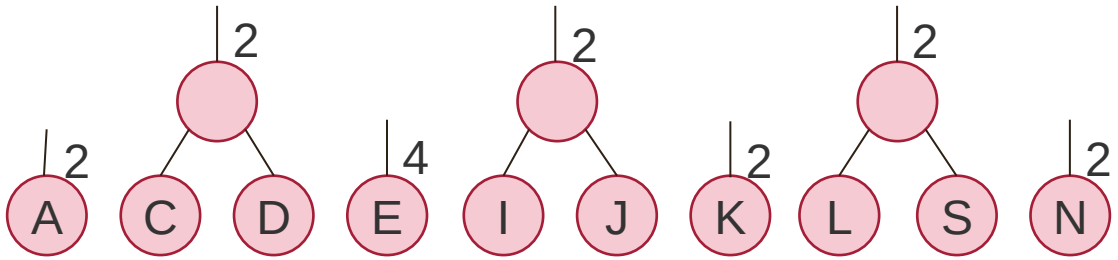
A	C	D	E	I	J	K	L	N	S
2	1	1	4	1	1	2	1	2	1

?

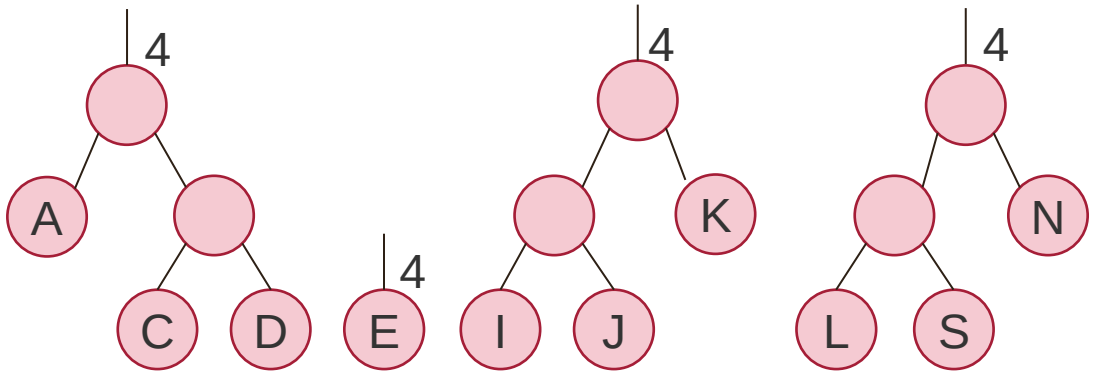


Huffman-Kodierung Beispiel

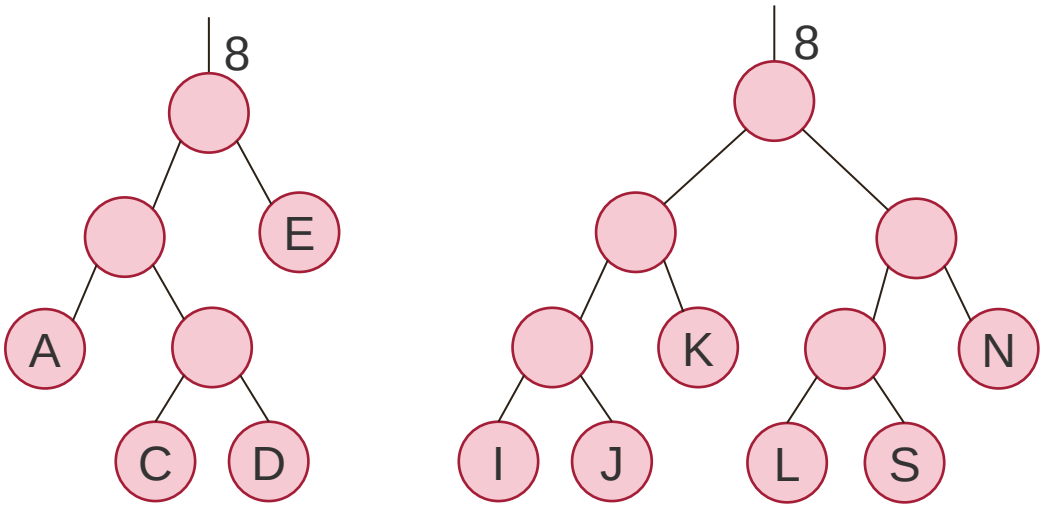
- In jedem Iterationsschritt werden die Teilbäume mit den geringsten Wahrscheinlichkeiten zusammengefasst
- Teilbäume mit gleichen Wahrscheinlichkeiten können beliebig kombiniert werden
- → **Verschiedene Lösungen möglich!**



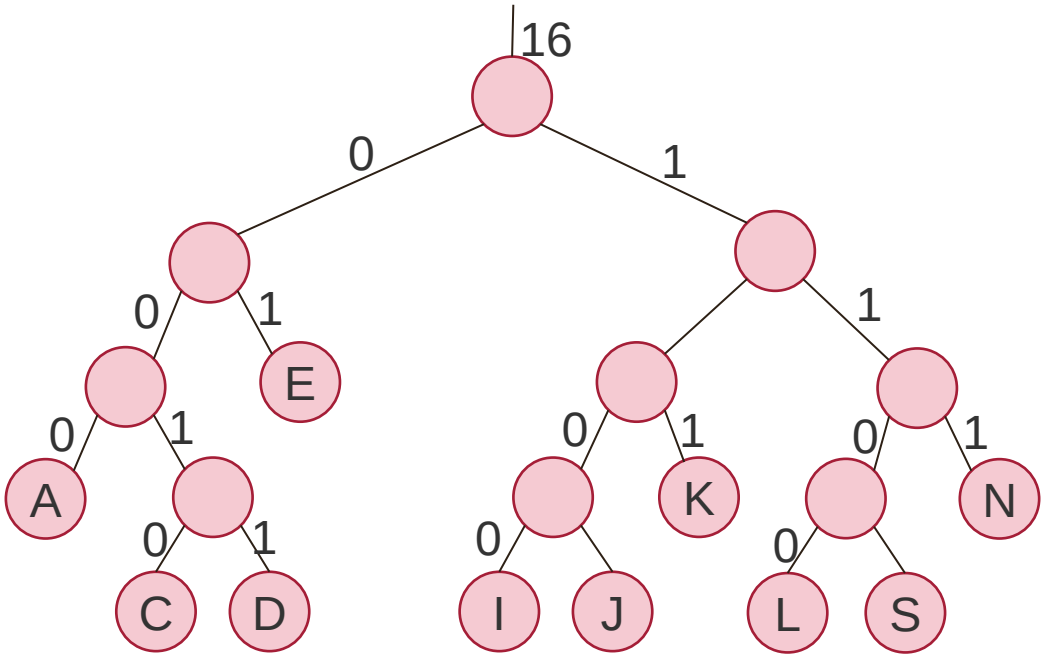
Huffman-Kodierung Beispiel



Huffman-Kodierung Beispiel



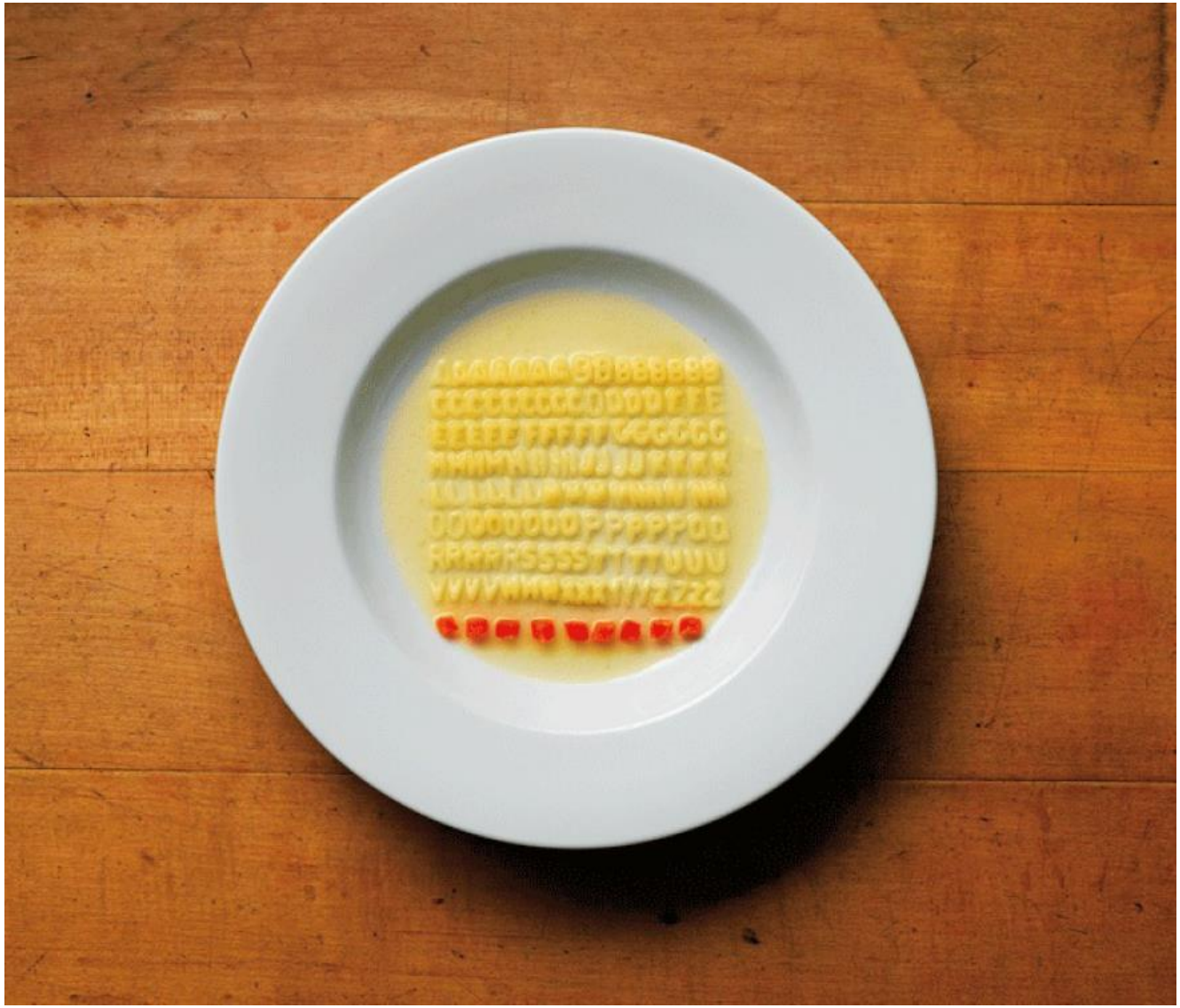
Huffman-Kodierung Beispiel



Zeichen	Code
A	000
C	0010
D	0011
E	01
I	1000
J	1001
K	101
L	1100
S	1101
N	111

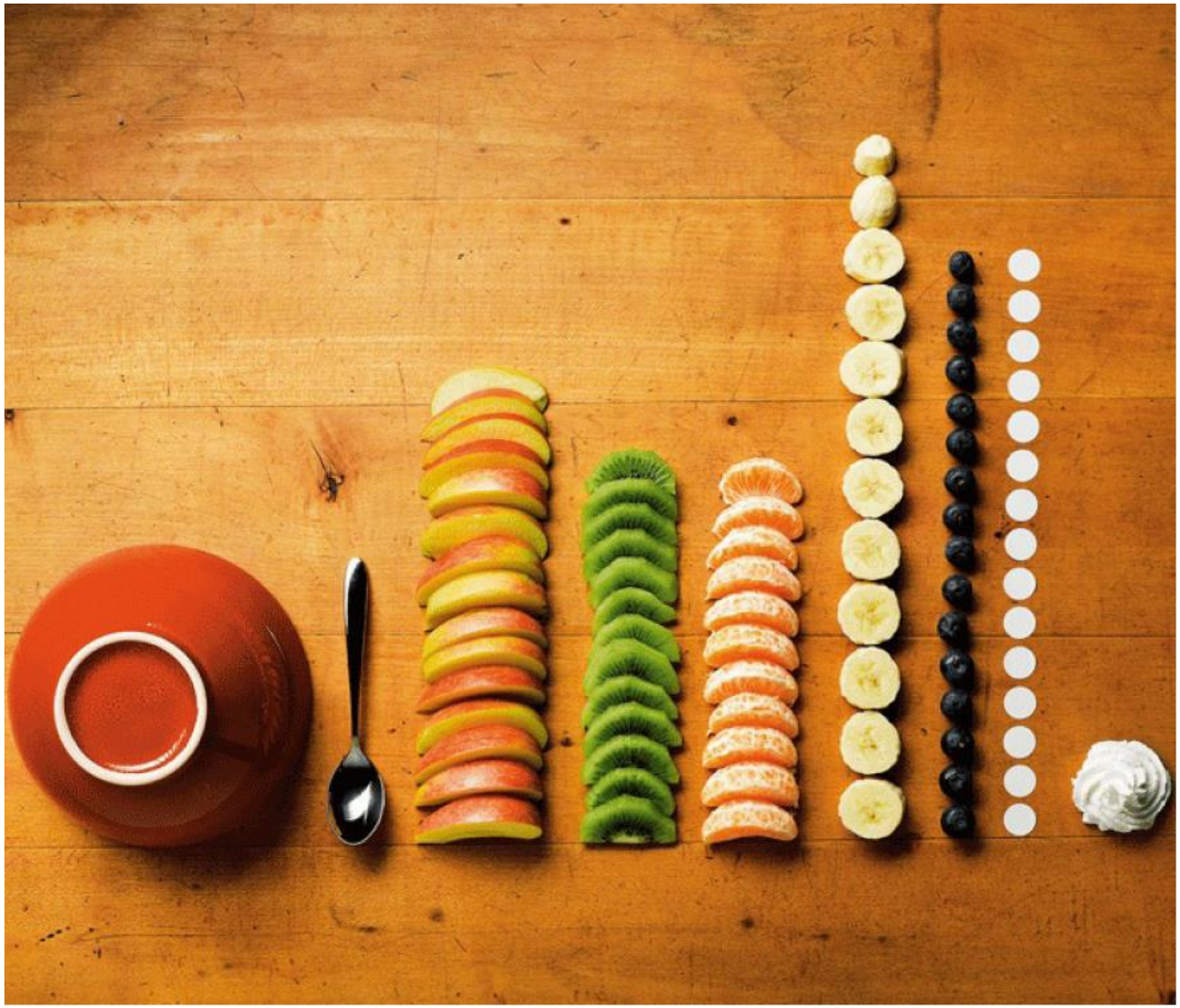
TEXTKOMPRIMIERUNG DURCH SORTIERUNG?

Quelle: Ursus Wehrli, Kunst aufräumen Kein & Aber Verlag, 2002



Quelle: Ursus Wehrli, Kunst aufräumen Kein & Aber Verlag, 2002

Quelle: Ursus Wehrli,
Kunst aufräumen
Kein & Aber Verlag,
2002



Quelle: Ursus Wehrli, Kunst aufräumen Kein & Aber Verlag, 2002

Quelle: Ursus Wehrli,
Kunst aufräumen
Kein & Aber Verlag,
2002

Quelle: Ursus Wehrli, Kunst aufräumen Kein & Aber Verlag, 2002

Komprimieren durch Sortieren?

- **Motivation:**

- Datenmenge ist sehr groß
- Bandbreite für die Übertragung im Vergleich klein

→ Vollständige Übertragung der Datenmenge (zeitlich) nicht effizient

Lösung

Komprimieren durch Sortieren?

- **Idee:**
  - Sortieren des Eingabetextes
  - Anwendung einer Entropiekodierung
    - Lauflängenkodierung
    - Huffman-Kodierung

- **Beispiel:**
  - HalloBallo
  - aaBHllllloo
  - aaBH4loo

- **Problem: Rückwärtstransformation zur Dekodierung**

Burrows-Wheeler-Transformation

Transformation

1. Erstellen einer **zyklischen Permutation** des Eingabestrings
2. **Lexikographisch Sortieren** der Permutationen
3. **Ausgeben des letzten Buchstabens** jeder Permutation und eines **Transformationsvektors**

Rücktransformation

- **Rücktransformation** mit Hilfe des **Transformationsvektors**

Zusätzliche Schritte

- „Move-to-front“-Kodierung
- Huffman-Kodierung

Burrows-Wheeler-Transformation Beispiel: 1. Permutieren des Eingabetextes

Beispiel: HalloBallo

H	a	l	l	o	B	a	l	l	o
a	l	l	o	B	a	l	l	o	H
l	l	o	B	a	l	l	o	H	a
l	o	B	a	l	l	o	H	a	l
o	B	a	l	l	o	H	a	l	l
B	a	l	l	o	H	a	l	l	o
a	l	l	o	H	a	l	l	o	B
l	l	o	H	a	l	l	o	B	a
l	o	H	a	l	l	o	B	a	l
o	H	a	l	l	o	B	a	l	l

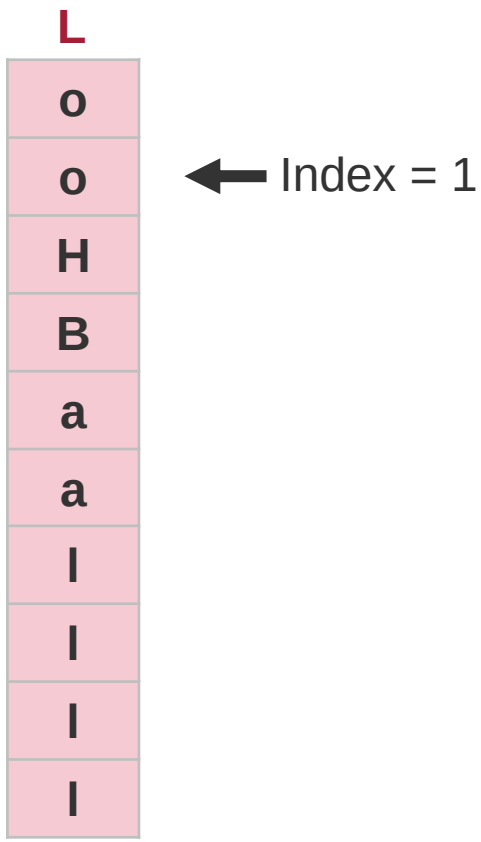
Burrows-Wheeler-Transformation Beispiel: 2. Zeilenweises Sortieren der Matrix

Ausgabe:

F									L
B	a	l	l	o	H	a	l	l	o
H	a	l	l	o	B	a	l	l	o
a	l	l	o	B	a	l	l	o	H
a	l	l	o	H	a	l	l	o	B
l	l	o	B	a	l	l	o	H	a
l	l	o	H	a	l	l	o	B	a
l	o	B	a	l	l	o	H	a	l
l	o	H	a	l	l	o	B	a	l
o	B	a	l	l	o	H	a	l	l
o	H	a	l	l	o	B	a	l	l

← Index = 1

Burrows-Wheeler-Rücktransformation Beispiel: Ausgangspunkt: Vektor L und Index



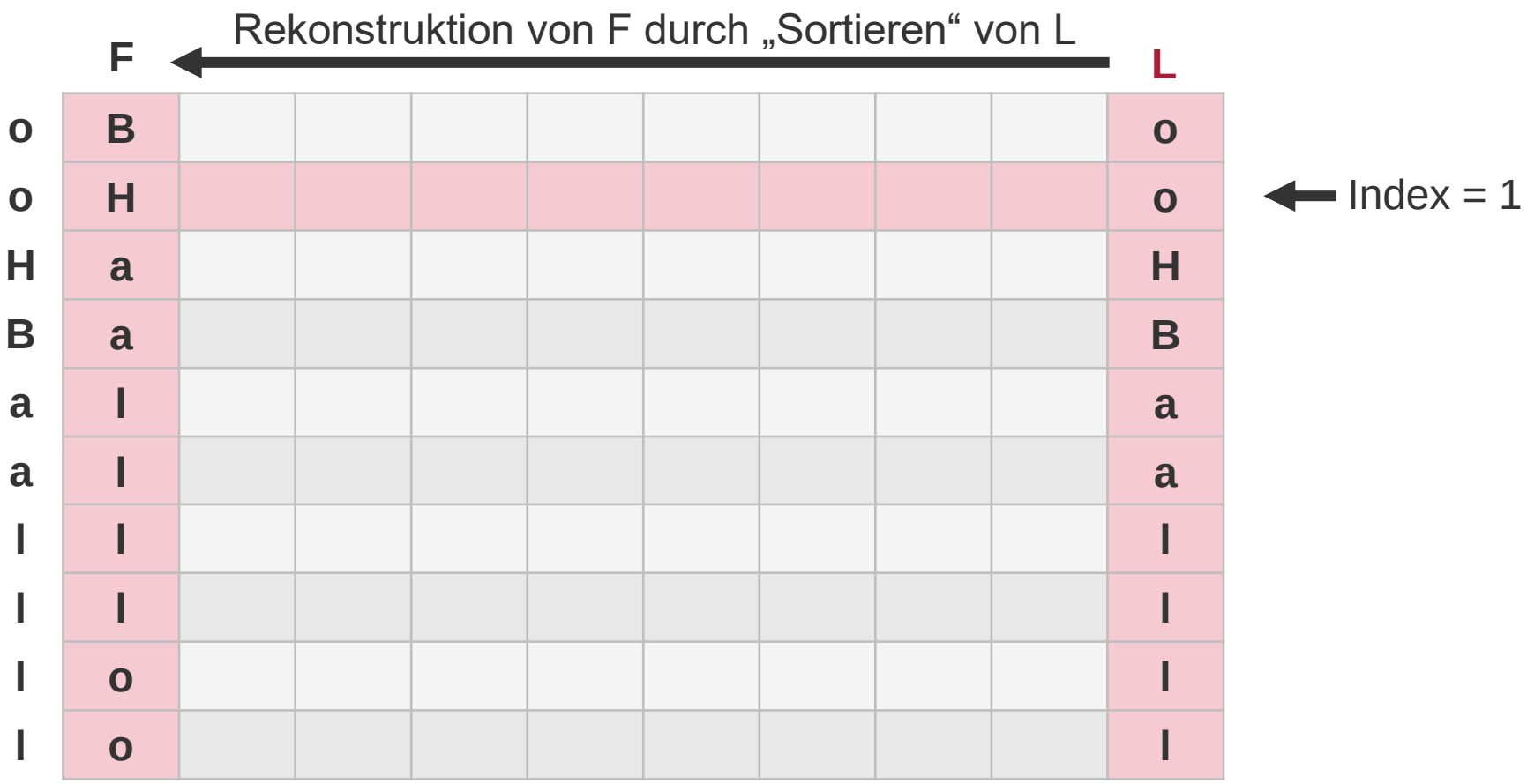
Burrows-Wheeler-Rücktransformation Beispiel: 1. Sortieren von L

Rekonstruktion von F durch „Sortieren“ von L

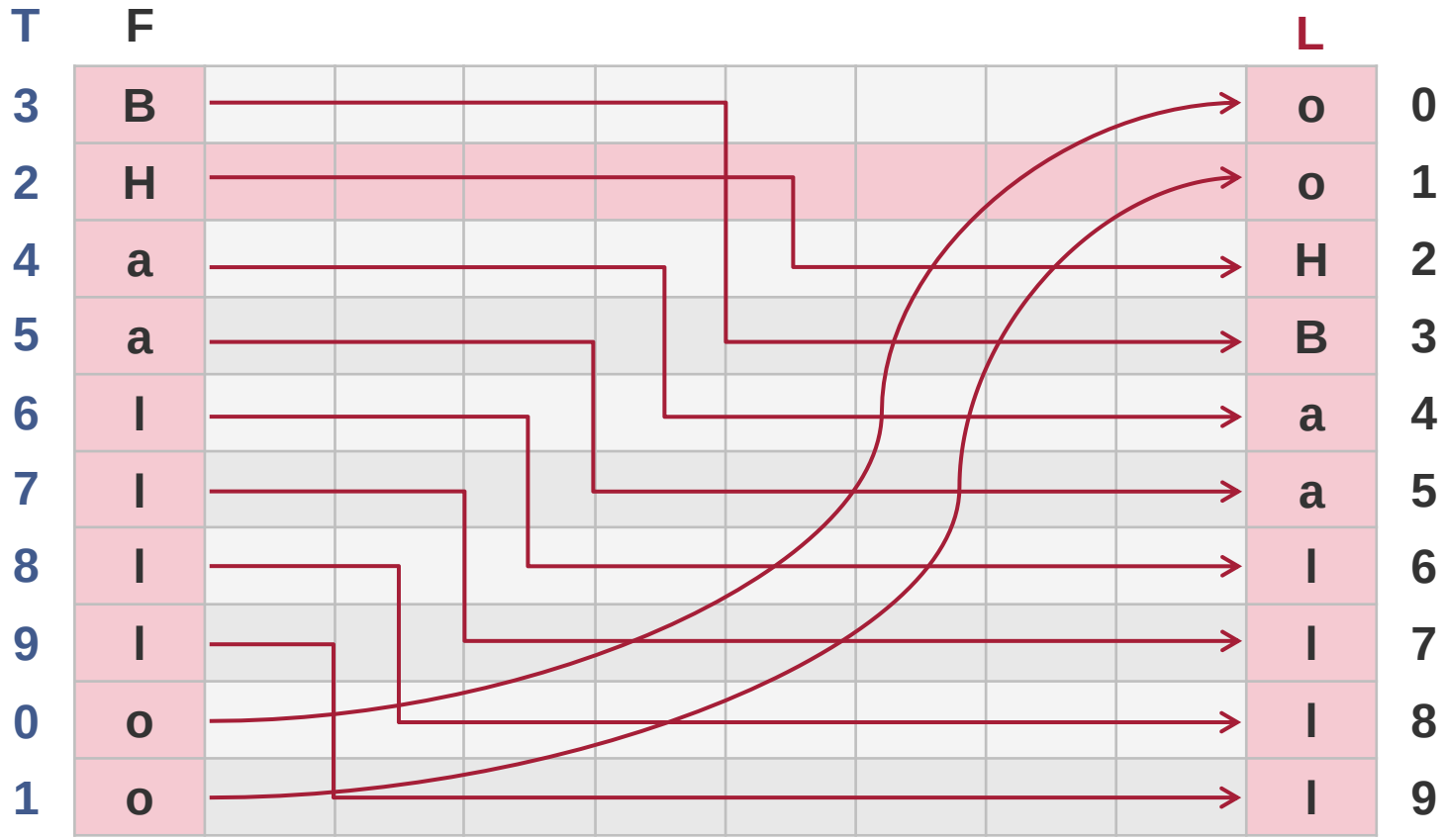
F									L
B									o
H									o
a									H
a									B
l									a
l									a
l									l
l									l
o									l
o									l

Index = 1

Burrows-Wheeler-Rücktransformation Beispiel: 1. Sortieren von L



Burrows-Wheeler-Rücktransformation Beispiel: 2. Konstruktion des Transformationsvektors (T)



Burrows-Wheeler-Rücktransformation Beispiel: 3. Rücktransformation des kodierten Textes

L	T	F									
o	3 → 0	B									
o	2 → 1	H									← Index = 1
H	4 → 2	a									
B	5 → 3	a									
a	6 → 4	l									
a	7 → 5	l									
l	8 → 6	l									
l	9 → 7	l									
l	0 → 8	o									
l	1 → 9	o									

Burrows-Wheeler-Rücktransformation Beispiel: 3. Rücktransformation des kodierten Textes

	T	F									
0 →	3	B									
1 →	2	H									← Index = 1
2 →	4	a									
3 →	5	a									
4 →	6	l									
5 →	7	l									
6 →	8	l									
7 →	9	l									
8 →	0	o									
9 →	1	o									

Burrows-Wheeler-Rücktransformation Beispiel: 3. Rücktransformation des kodierten Textes

	T	F									
0 →	3	B									
1 →	2	H	a	l	l	o	B	a	l	l	o ← Index = 1
2 →	4	a									
3 →	5	a									
4 →	6	l									
5 →	7	l									
6 →	8	l									
7 →	9	l									
8 →	0	o									
9 →	1	o									

Burrows-Wheeler-Rücktransformation Beispiel: Alternative Schreibweise der Rücktransformation

Wort: BANANA, **Transformationsvektor:** ANNB$AA

Output: B

Codierter Text	A	N	N	B	$	A	A
Position	1	2	3	4	5	6	7
Sortierter Text	$	A	A	A	B	N	N
Sortierte Position	5	1	6	**7**	4	2	3

Gehe zu Index 4 in sortiertem Text

Burrows-Wheeler-Rücktransformation Beispiel: Alternative Schreibweise der Rücktransformation

Wort: BANANA**, Transformationsvektor:** ANNB$AA

Output: BA

Codierter Text	A	N	N	B	$	A	A
Position	1	2	3	4	5	6	7
Sortierter Text	$	A	A	A	B	N	N
Sortierte Position	5	1	6	7	4	2	3

Dann zu Index 7

Burrows-Wheeler-Rücktransformation Beispiel: Alternative Schreibweise der Rücktransformation

Wort: BANANA, **Transformationsvektor:** ANNB$AA

Output: BAN

Codierter Text	A	N	N	B	$	A	A
Position	1	2	3	4	5	6	7
Sortierter Text	$	A	A	A	B	N	N
Sortierte Position	5	1	6	7	4	2	3

Nächstes Zeichen ergibt sich aus Position des aktuellen Buchstabens
Solange den Vorgang wiederholen bis $ erreicht → Output: BANANA$

Vorteile der BWT

- Schnell anwendbar
- Auch für große Datenmengen schnell berechenbar
- Geringer Speicherbedarf
- Kein Informationsverlust bei Trans-/Rücktransformation

Burrows-Wheeler-Transformation: Backend I: MTF

„Move-to-front“-Kodierung (MTF)

- Algorithmus (Prinzip):
  - Schreibe das komplette Alphabet in eine Zeichenkette a.
  - Für jedes Zeichen z der Eingabe:
    - Gib die Position von z in a aus
    - Entferne z aus a und füge es vorne wieder an
- Eigenschaft: Nach kurzer „Einarbeitungsphase“ werden relativ häufig kleine Zahlen ausgegeben → gut für anschließende Komprimierung

Burrows-Wheeler-Transformation: Backend I: MTF Beispiel

Eingabe: o o H B a a l l l l, Alphabet: Hbalo (Alphabet vorgegeben)

Ausgabe	Aktuelles Alphabet
4	oHBal
0	oHBal
1	HoBal
2	BHoal
3	aBHol
0	aBHol
4	laBHo
0	laBHo
0	laBHo
0	laBHo

- → Ausgabe: **4012304000**

Burrows-Wheeler-Transformation: Backend II Huffman

- MTF
- Eingabe: 4012304000, Index = 1
- Dekodiere die MTF-Ausgabe mit Hilfe des Huffman-Codes
  - Füge den Index ‘1’ in die Ausgabe binär ein

- Huffman-Code:
  - 0 → 0
  - 3 → 1111
  - 4 → 10
  - 1 → 110
  - 2 → 1110
  - Ergebnis: 10 0 110 1110 1111 0 10 0 0 0, 0001

Bzip2

- **Verschiedene Methoden, werden mehrfach in Reihenfolge durchgeführt**
  - Lauflängenkodierung
    - Alle Sequenzen länger als 4 bis 255 Zeichen werden codiert: “AAAAAAABBBBCCCD” wird "AAAA\3BBBB\0CCCD"
  - Burrows-Wheeler-Transformation
    - Sortiert Blöcke von 900kB
  - „Move-To-Front“-Kodierung
    - Zeichen werden mit einem Index kodiert, wiederholende Zeichen werden nach dem ersten Index 0
      ooHBaallll wird zu 4012304000
  - Erneut Lauflängenkodierung
  - Huffman-Kodierung & Huffman-Tabellen
  - Base 1 Encoding
  - Delta Encoding
  - Sparse Bit Array

Zusammenfassung

- **Informationstheorie** bildet **Grundlage** für **Kodierungsverfahren**
- Entropiebasierte Komprimierungsverfahren gut für Texte geeignet
- Eine **partielle Sortierung** unterstützt **hohe Kompressionsraten**
  - Lauflängenkodierung, Move-to-Front, Huffman
- Verbreitete Verfahren nutzen eine **Kombination** von unterschiedlichen **Komprimierungsansätzen**
  → hybride Verfahren
  - z.B.: Bzip2