コンピュータシステムA - ハードウェアを中心に -

#8 bit, Byte, データ表現、互換性

Yutaka Yasuda

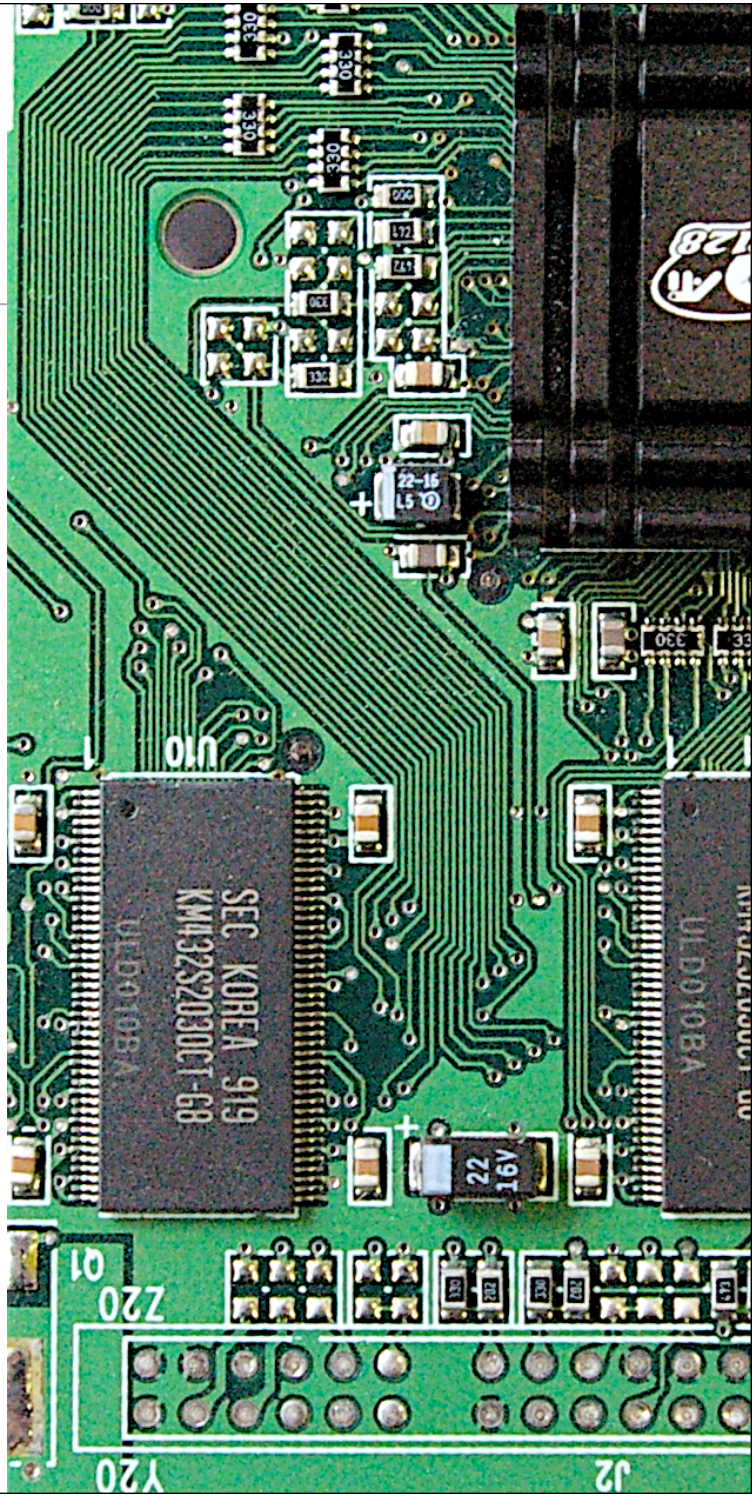
bit : データの最小単位

- 1bit = 最小状態の単位 = 二進一桁
- コンピュータ内部は電気配線

  配線に電気が通っている、いない、だけで処理（状態は2種）
- 二値（二進）動作にうまく対応

  二進一桁を配線一本で実現

  「0と1（二進数）で動作」の実体
- 1bit = 二進一桁 = 配線一本



Byte : データの標準枠

- Byte （バイト）

  コンピュータが扱うデータの基本単位（歴史的経緯）

  bit を 8 つまとめて 1 Byte とする

  0-255までの256種類の値が入る

  255を超える値は二桁（2Bytes）使う

- アルファベットは 1 バイトでおさまる
- 漢字は（普通は） 2 バイトを要する

  「フロッピー1枚は新聞何枚に相当し、、」

単位：Kilo, Mega, Giga, Tera

- メモリ量などに巨大な桁を扱う事が多い
- 欧米的 1000 倍単位
- コンピュータ固有の装置では 1024 単位の場合が多い

単位	読み	日本	1000	1024	誤差
K : Kilo	キロ	千	1000	1024	2.4%
M : Mega	メガ	100万	1,000,000	1,048,576	4.9%
G : Giga	ギガ	10億	1,000,000,000	1,073,741,824	7.4%
T : Tera	テラ	1兆	1.000.000.000.000	1,099,511,627,776	10.0%
P : Peta	ペタ	1000兆	1.000.000.000.000.000	1,125,899,906,842,620	12.6%

距離感

- 12 桁、15 桁のスケール感を把握する
- もし 1mm 幅で 1 バイトのメモリができたとする

  このメモリを並べて K, M, G, P バイトのメモリを実現した場合、どの程度の長さになるか？
- 1TB ディスクの広大さを想像する

1	虫眼鏡？
1K	1メートル
1M	1キロメートル
1G	京都〜盛岡間（1000キロ）
1T	月までの三倍弱ほど

Byte量：音楽CDは何バイトあるか？

- さまざまなもののバイト数
- 広辞苑 (第二版)

  24字 x 50行 x 4段 x 2400ページ=11,520,000 字

  一文字 2 Bytesとして 23 Mega Bytes (MB)
- 音楽CD

  44KHz x 65536段階(2Bytes) x 2ch = 176KB/sec

  176KB x 3600sec = 633,600 KB = 634MB

  さまざまなものが bit にかわる姿を想像できたろうか？

N 進法（位取り記数法）

- 一桁を幾つの記号で回すか、を意味する
- 右端のドラムが一周まわると、左隣のダイヤルが一つ進む

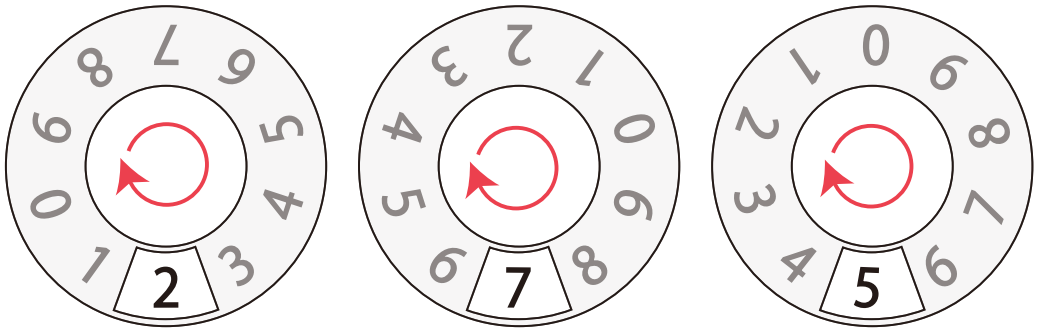
10進法はドラムに10種の記号がある数え方。

2進法は2種しか記号がない。短い周期で桁があがるだけで、回り方は同じ。

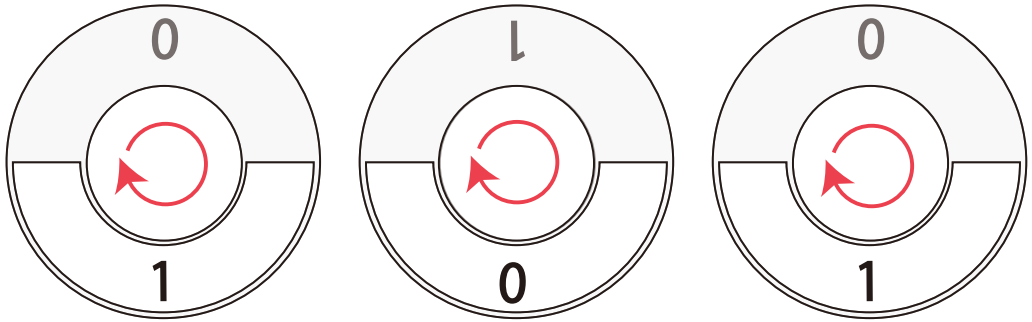
N 進法（位取り記数法）

- 一桁を幾つの記号で回すか、を意味する
- 右のダイヤルを一周まわすと、左隣のダイヤルが一つ進む

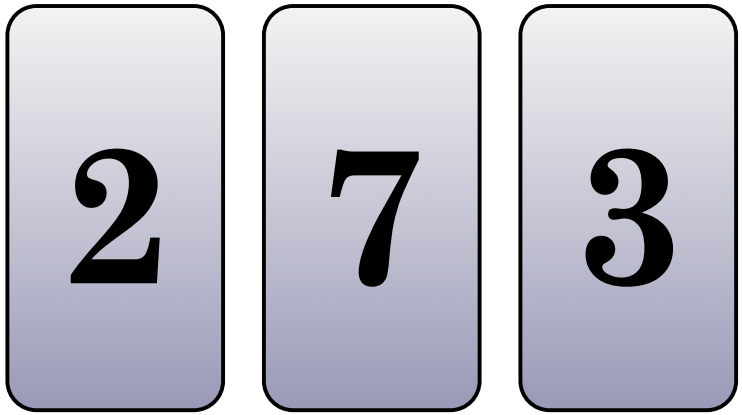
10進法はダイヤルに10種の記号がある数え方。ダイヤルを275 step 進めると右図のようになる。



2進法は2種しか記号がない。5 step 進めると右図の状態になる。101 と表記する。



2進表記

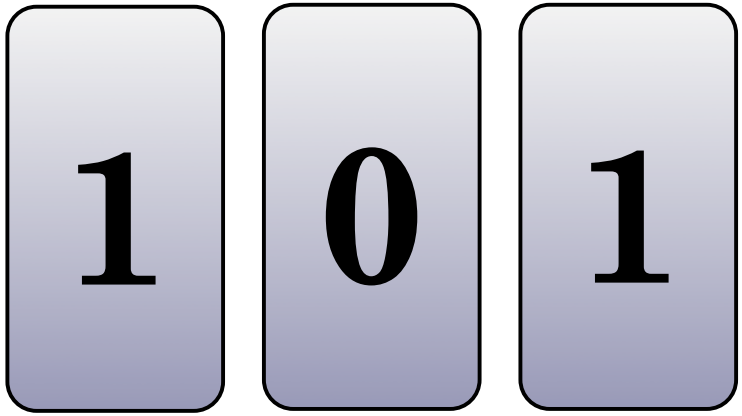


10進の
3桁め

10進の
2桁め

10進の
1桁め

100 x 2
10 x 7
+ 1 x 3



2進の
3桁め

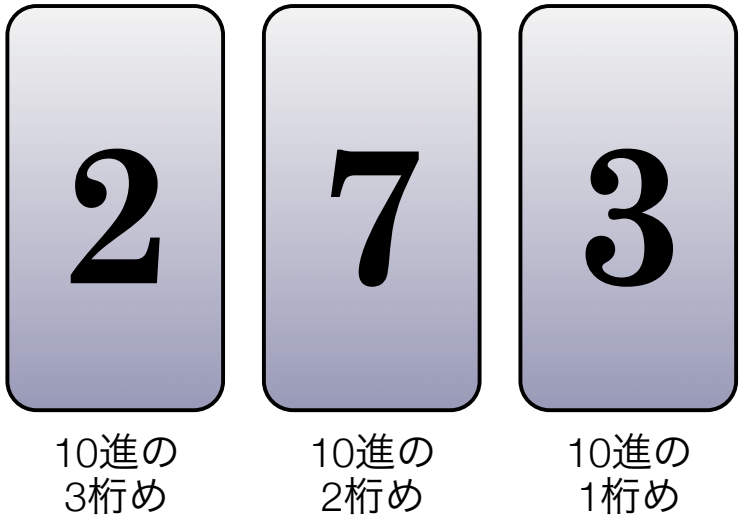
2進の
2桁め

2進の
1桁め

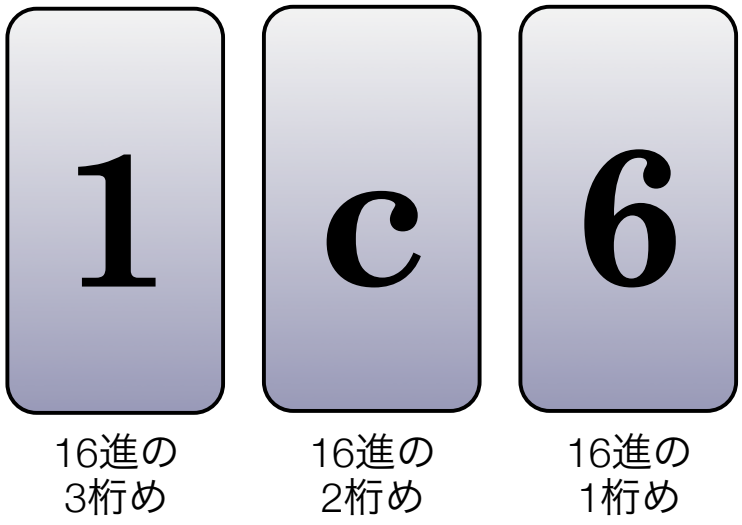
4 x 1
2 x 0
+ 1 x 1

10進	2進
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011

16進表記



$$
\begin{aligned}
100 \times 2 \\
10 \times 7 \\
+ \quad 1 \times 3
\end{aligned}
$$



$$
\begin{aligned}
256 \times \quad 1 \\
16 \times 12 \\
+ \quad 1 \times \quad 6
\end{aligned}
$$

10進	16進
0	0
1	1
2	2
....	
9	9
10	a
11	b
12	c
13	d
14	e
15	f
16	10
17	11
18	12
19	13

確認

- デジタルは 0, 1 である（二値である）は間違い
- デジタル：数値で処理するところが重要
- コンピュータが二値である理由

  組み合わせ数が少ないので回路が単純になる

  中間的な電圧を利用するなど技術的に複雑になる（結果的に速度を上げられない）
- 3値や多値ができても不思議はないが、今は無い
- フラッシュメモリなど部分的実用例はある

データ表現

データ表現

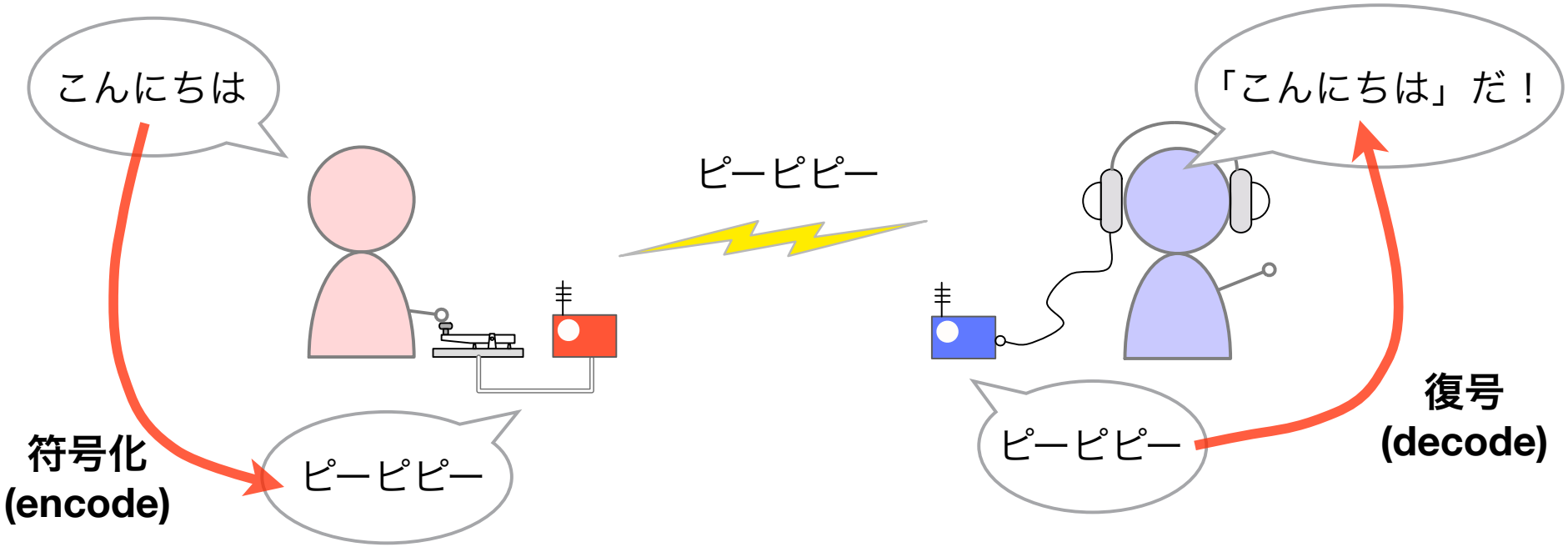
- コンピュータはbitの集まりだけを処理できる

  どのような情報でもbitに変えることができればコンピュータで処理できる

- データを bit に対応させる方法について知ろう

文字のデータ化（encode, decode）

- 文字をデータに変換する
- モールス信号



機械（電鍵および無線機）は文字を扱えないので人間が文字を符号に変換している

モールス信号

- 短音と長音の組み合わせで文字を表現
- 相手と共通の符号化パターンを用いる事が重要
- 違う符号表を用いると？

A	・－	イ	・－
B	－・・・	ロ	・－・－
C	－・－・	ハ	－・・
D	－・・	ニ	－・－・
E	・	ホ	－・・

フォーマット（書式）

- データの解釈には解釈（復号）ルールが必要
- つまりデータにはフォーマット（書式）がある

  フォーマットを間違えて解釈すると間違った結果が導き出される

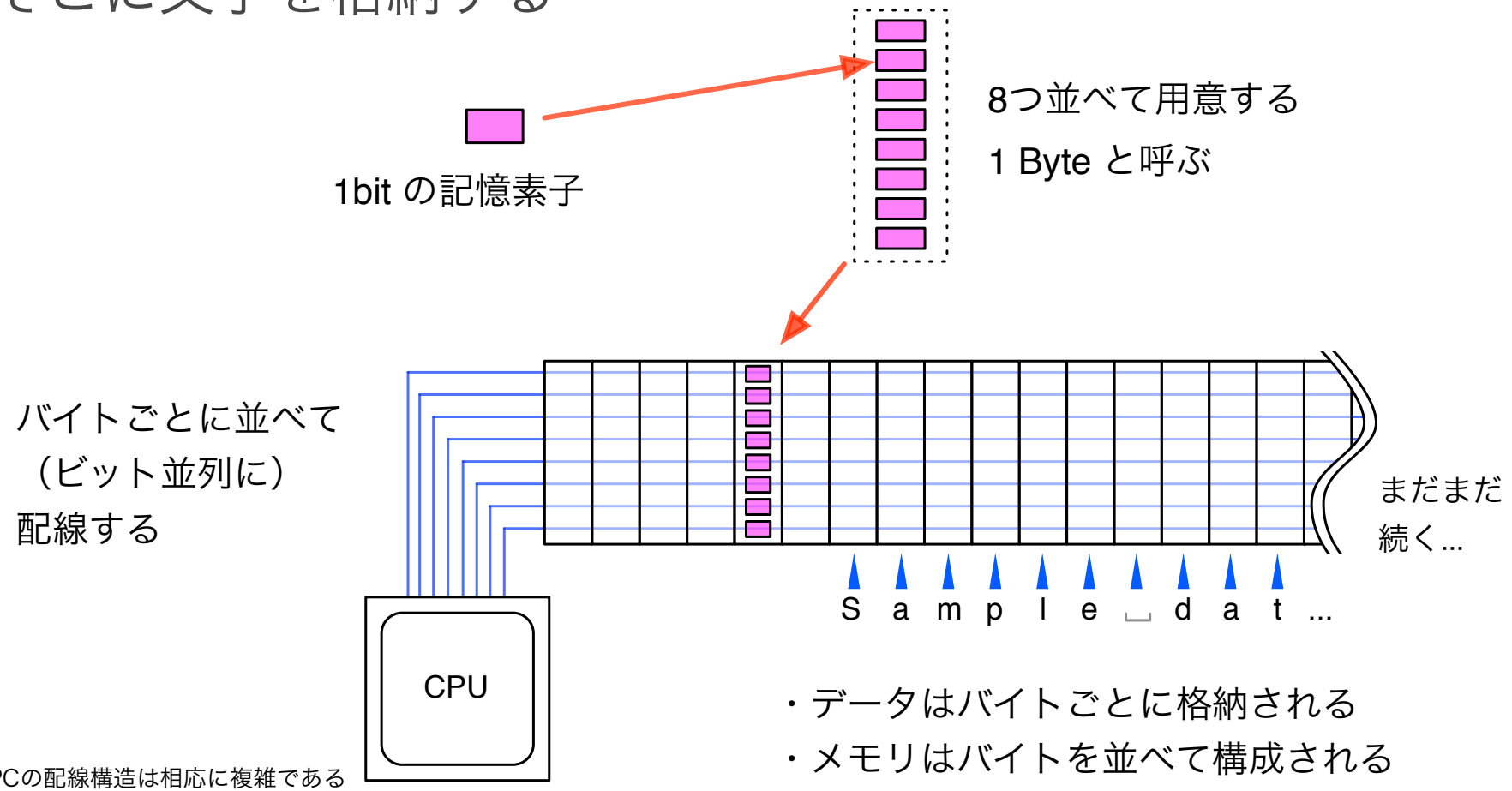
  異なるアプリケーションでデータが扱えない理由

  （データにおける）「互換性」という概念の実体

  いわゆる文字化けの原因

文字のデータ表現

- メモリはバイトが並んだものと考える
- そこに文字を格納する



・データはバイトごとに格納される
・メモリはバイトを並べて構成される

※実際のPCの配線構造は相応に複雑である

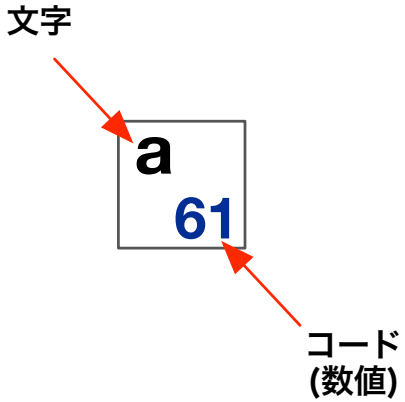
文字のデータ表現

- メモリ

  数値を格納

- 文字に番号を振る

  ASCII （右表）



つまりメモリに格納されるのは文字ではなく文字に相当する値である

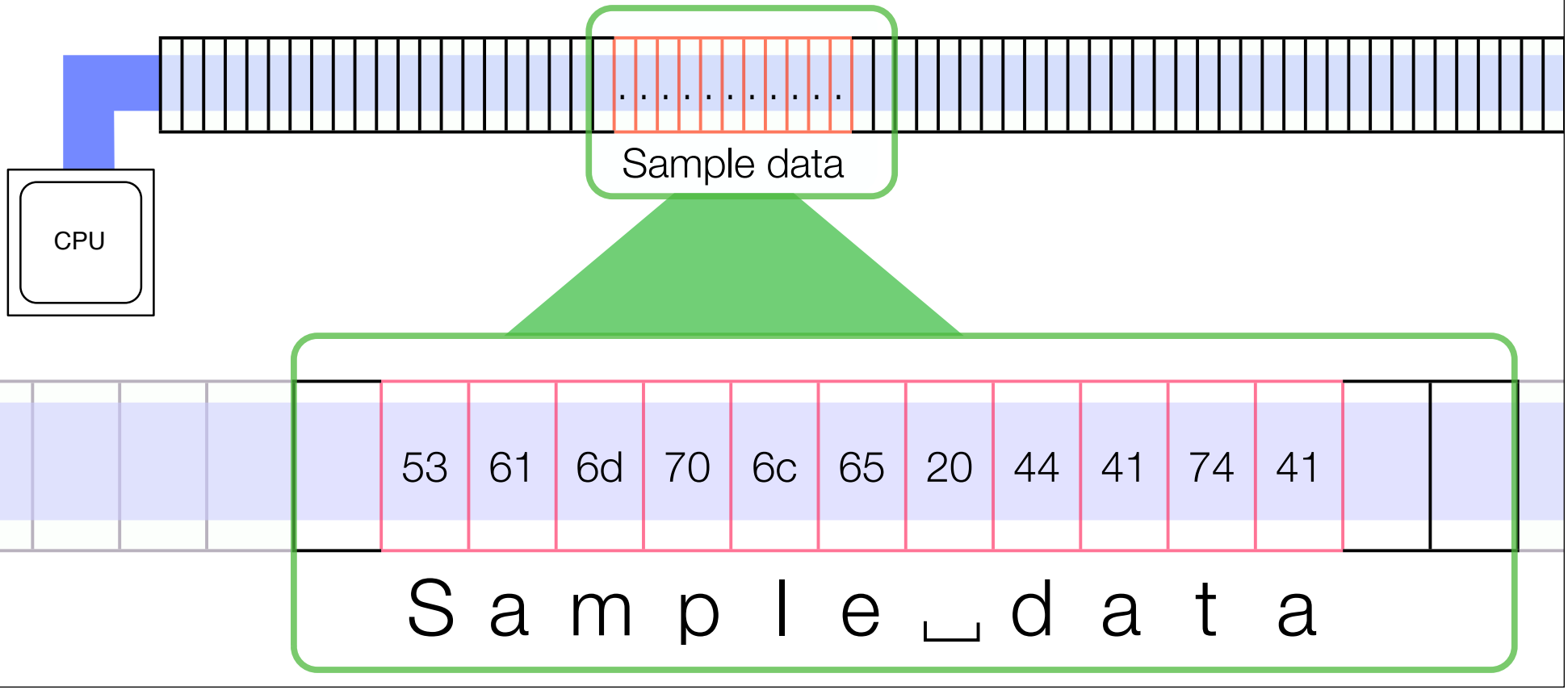
ASCII文字は 1 バイトで表現されているが、その実体は数値である。
つまり 'a' は番号 61 の文字。
61 は 16 進数表記なので、10進数で表記すると 97 番文字となる。

\0 00	10	sp 20	0 30	@ 40	P 50	` 60	p 70
01	11	! 21	1 31	A 41	Q 51	a 61	q 71
02	12	" 22	2 32	B 42	R 52	b 62	r 72
03	13	# 23	3 33	C 43	S 53	c 63	s 73
04	14	$ 24	4 34	D 44	T 54	d 64	t 74
05	15	% 25	5 35	E 45	U 55	e 65	u 75
06	16	& 26	6 36	F 46	V 56	f 66	v 76
07	17	' 27	7 37	G 47	W 57	g 67	w 77
08	18	(28	8 38	H 48	X 58	h 68	x 78
\t 09	19	) 29	9 39	I 49	Y 59	i 69	y 79
\n 0a	1a	* 2a	: 3a	J 4a	Z 5a	j 6a	z 7a
0b	1b	+ 2b	; 3b	K 4b	[5b	k 6b	{ 7b
0c	1c	, 2c	< 3c	L 4c	\ 5c	l 6c	| 7c
0d	1d	- 2d	= 3d	M 4d	] 5d	m 6d	} 7d
0e	1e	. 2e	> 3e	N 4e	^ 5e	n 6e	~ 7e
0f	1f	/ 2f	? 3f	O 4f	_ 5f	o 6f	7f

文字のデータ表現

- 文字列 ‘Sample data’ を格納する

  メモリのどこかに、一文字を一バイトずつ詰める



日本語（漢字）データと文字化け

漢字データは一文字二バイト

JIS

1b 24 42 34 41 3b 7a 1b 28 42

漢字はじまり　英字はじまり

漢 字

Shift JIS

8a bf 8e 9a

漢 字

Unicode (UTF-16)

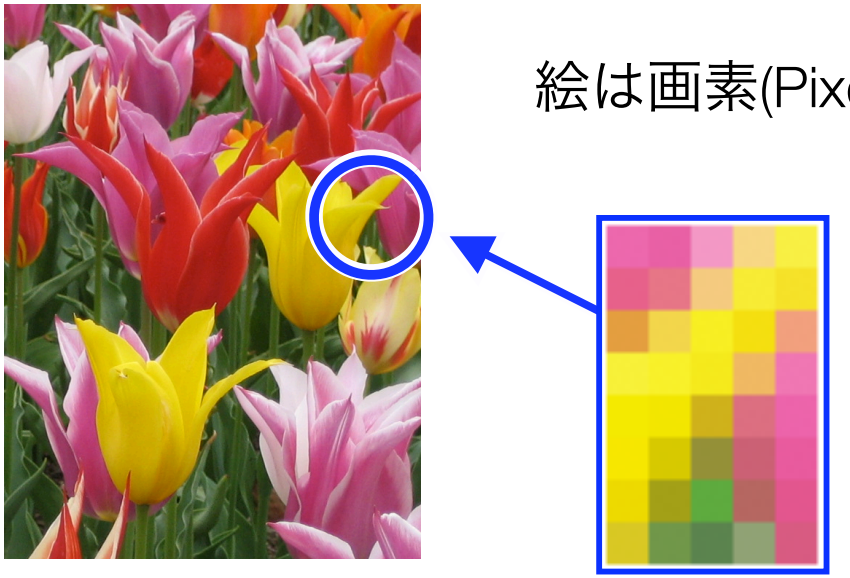
fe ff 6f 22 5b 57

BOM

漢 字

BOM : バイト順マーク（feff は正順）

画像のデータ表現



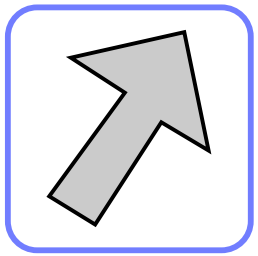
絵は画素(Pixel : Picture Element)ごとに分解

一画素ごとに赤・緑・青 (RGB) に
色分解して各色256段階で記録
最大 16,777,216 色

	赤	緑	青
→	229	83	158
→	242	231	0
→	80	155	46

動画も簡単にデータ化できますね？

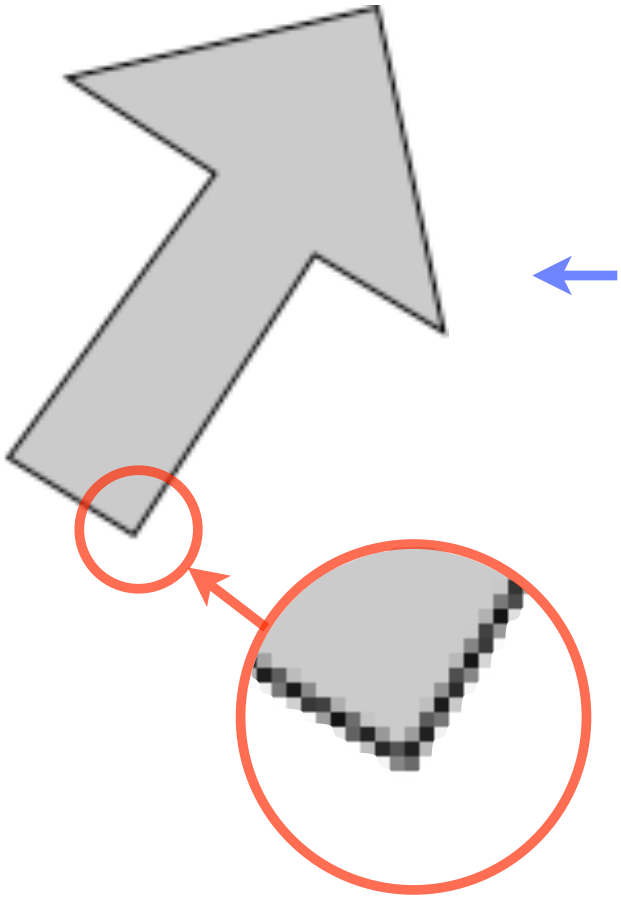
保存形式：ビットマップ vs ベクター



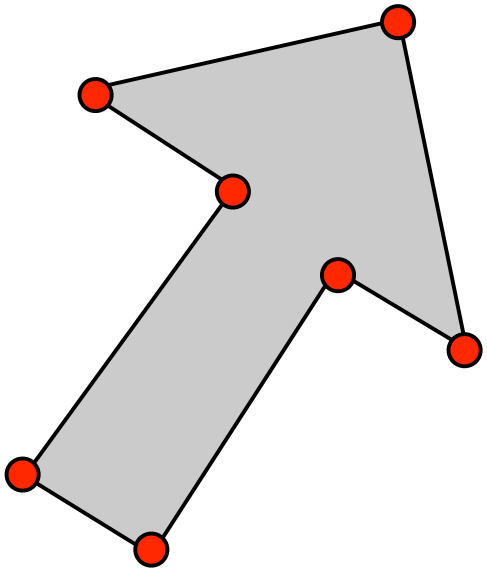
この画像を

← ビットマップで作る

ベクターで作る →



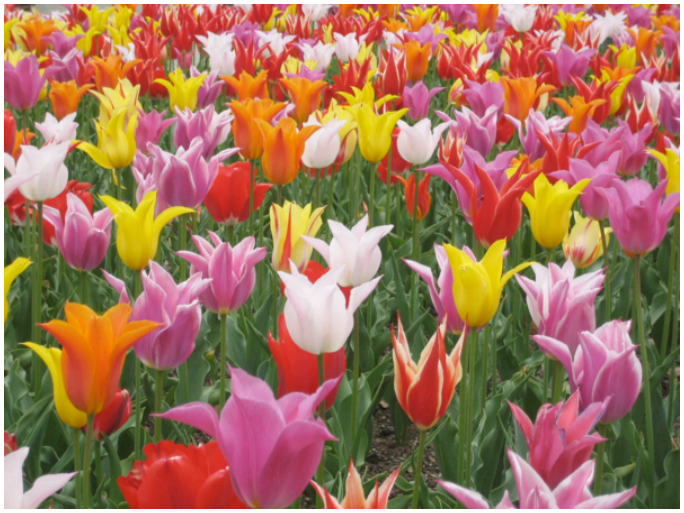
Pixel の集合体



座標位置や描画方法
の情報の集合体

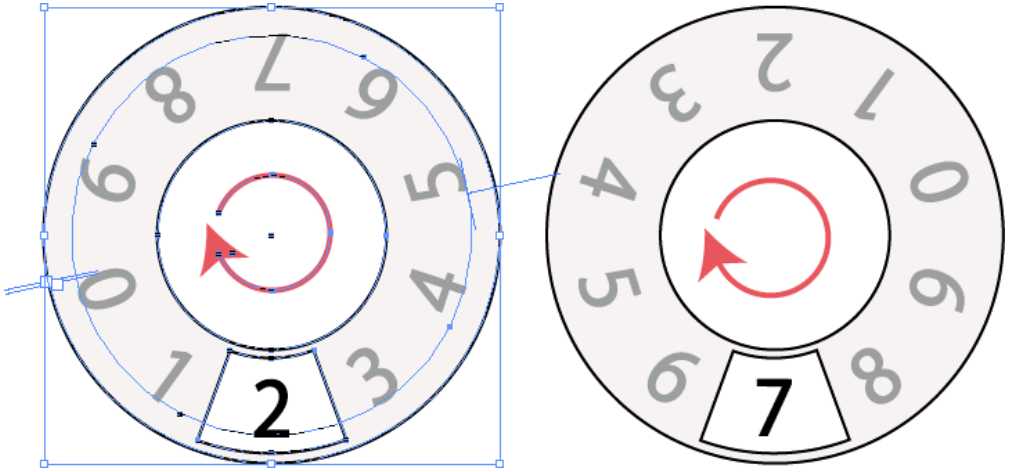
(109.29, 55.70)
(174.82, 15.26)
(269.24, 161.64)
(336.69, 121.19)
(302.01, 290.68)
(140.13, 254.08)
(217.21, 204.01)

保存形式：ビットマップ vs ベクター



Pixel の集合体に適する

縮めて見せることはでき
ても拡大すると粗くなる



座標位置や描画方法を指定
して作る画像に適する

拡大・縮小に問題がない

保存形式：ビットマップ vs ベクター

- ビットマップ形式の例

  （Windows の）ビットマップ

  JPEG ：圧縮（後述）

  GIF
- ベクター形式の例

  (Windowsの)メタファイル

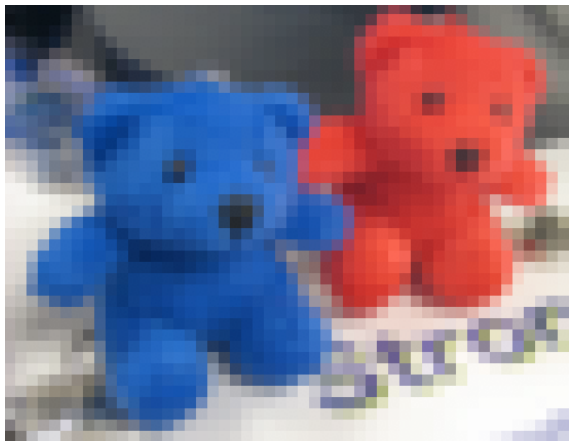
  Flash

  PDF

  Illustrator (PostScript)

圧縮（の前に）

オリジナル	1/5 縮小	1/20 縮小
400 x 312 x 3 = 374KB	80 x 62 x 3 = 14.9KB	20 x 16 x 3 = 960B



縮小して保存、拡大して表示すれば、少ないデータ量で同じ絵を得られる。
データ量は幾らでも減る。ただしディティールは失われる。

JPEGにおける圧縮

374 x 369 pixel image

40.9KB (1/10)

10.7KB (1/40)

8.4KB (1/50)

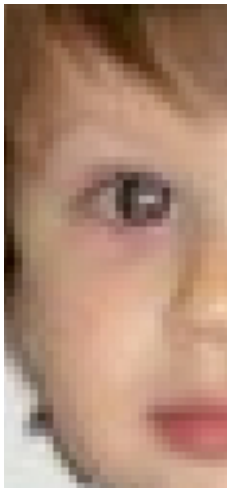
品質＝高い
データ量＝多い

品質＝低い
データ量＝少ない

無圧縮：374 x 369 x 3 = 414,018 バイト (414KB)

GIF

374 x 369 pixel image

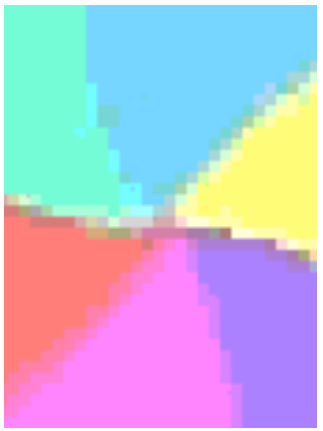


JPEG : 40.9KB

○

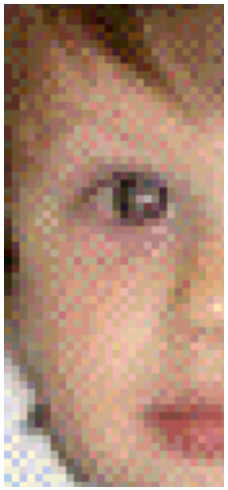
JPEG
vs
GIF

177 x 190 pixel image



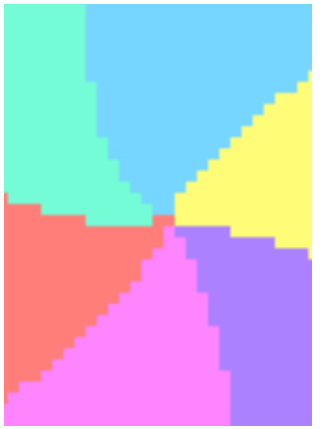
JPEG : 5.5KB

×



GIF : 83.8KB

×

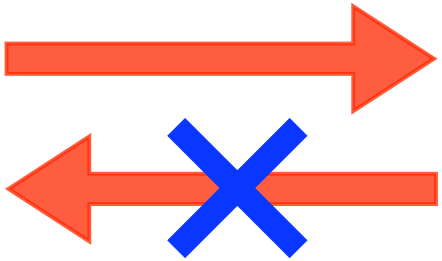


GIF : 1.8KB

○

可逆圧縮と非可逆（不可逆）圧縮

無圧縮：約 400KB



非可逆圧縮

JPEG：40.9KB

JPEG
vs
GIF

無圧縮：約 34KB

可逆圧縮

GIF：1.8KB

画像の圧縮（まとめ）

- ビットマップ画像のフォーマット

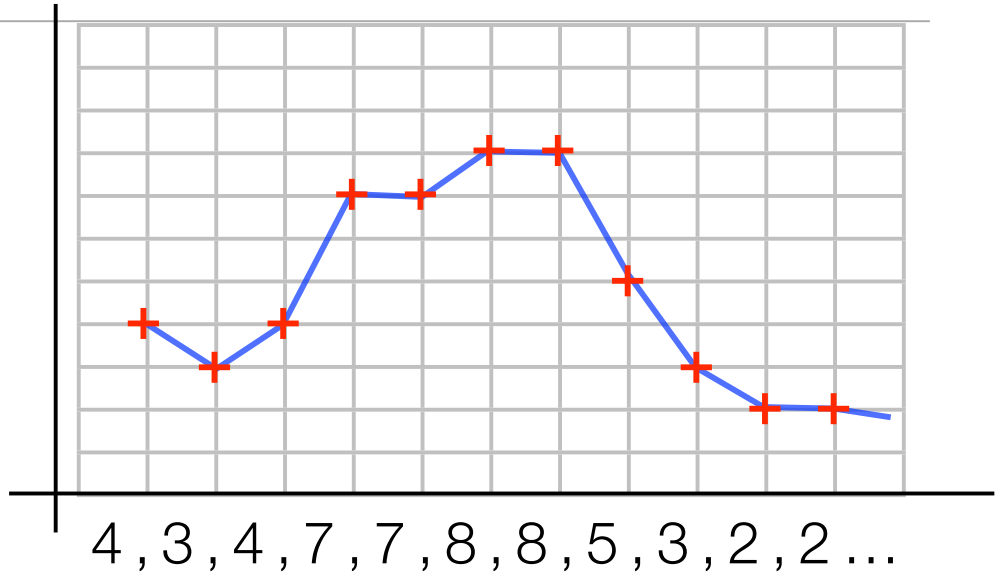
  （Windows の）Bitmap, JPEG, GIF などなど

- 圧縮を行う場合もある

  可逆：非圧縮の状態に戻せる（元の情報が失われない）
  ≒ 無駄（冗長）な表現をしない

  非可逆：情報が失われるが圧縮率が高い（場合が多い）
  ≒ 詳細を省いて情報量を減らす

音声のデジタル表現



- サンプリング

  標本化と量子化

  CDは44KHz, 16bit

- MP3

  非可逆圧縮の一つ

  人間が聞き取りにくい音の情報を削除する→音質劣化

  CD音源を 1/8〜1/15 程度に圧縮

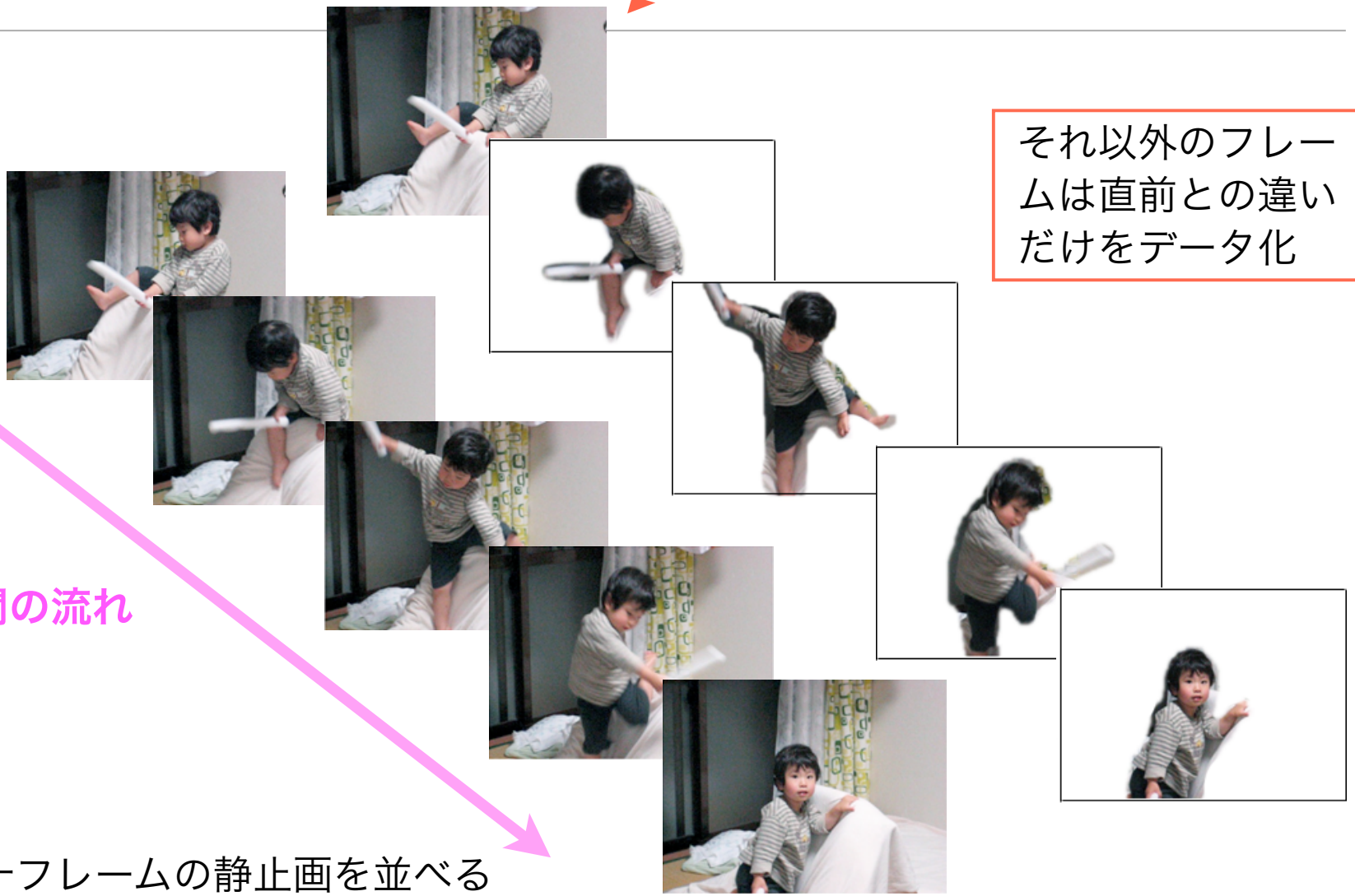
- AAC, ATRAC, WMA などなど他多数

  圧縮率と品質のよりよい両立を求めて

動画の表現

ときどき全情報を含むフレーム（キーフレーム）を用意する

それ以外のフレームは直前との違いだけをデータ化



- 毎秒数十フレームの静止画を並べる
- 直前のフレームとの違いだけをデータ化する

動画の表現

- WMV, MPEG, QuickTime, H.264 など各種あり

  非圧縮ではDVD 2.4GBに 720×486 画素 24bit 色 30fps は 75 秒 しか入らない (※)

- 符号化方式も重要だが、帯域のことも

  DVD : 11Mbps, BlueRay : 36Mbps

  DV : 30Mbps

  地上波デジタル : 80Mbps以下程度

※インターレースのことなど考慮すべきものは多いがここでは単純さを優先した

まとめ：デジタルデータとフォーマット

- その実体は数値（記号）の列

  音声：111,121,122,89,80,82,75….

  静止画：10,240,22,30,34,80…

  音声付き動画：12,33,45,1123,488…

  文字：33,38,42,60,32,39,55,80…

- これだけでは利用できない（意味が汲み取れない）

  符号化ルールとデータは常に一体

- このルールがフォーマット（書式）を生む

補足：数値の表現

補足：数値の表現

- 数値の表現

    整数・浮動小数

- ハードウェア（主としてCPU）が直接処理できるか否か

- あり得る全ての数値を処理できるわけではない

    処理速度・メモリ消費量（トレードオフ）

    そのように作られた処理系もある

整数

- 整数

  二進表記

  1Byte：8bit, 0〜255

  4Byte：32bit, 0~4G（9桁、約43億）

  8Byte：64bit, 0〜??（19桁）

10進表記	4bit 2進表記
15	1111
14	1110
13	1101
12	1100
11	1011
10	1010
9	1001
8	1000
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000

整数

- 符号付き整数（負数の表現）

  負数は 2 の補数
  （1引いてビット反転）

  符号なし 1Byte：8bit, 0〜255

  符号あり 1Byte：8bit, -128〜127

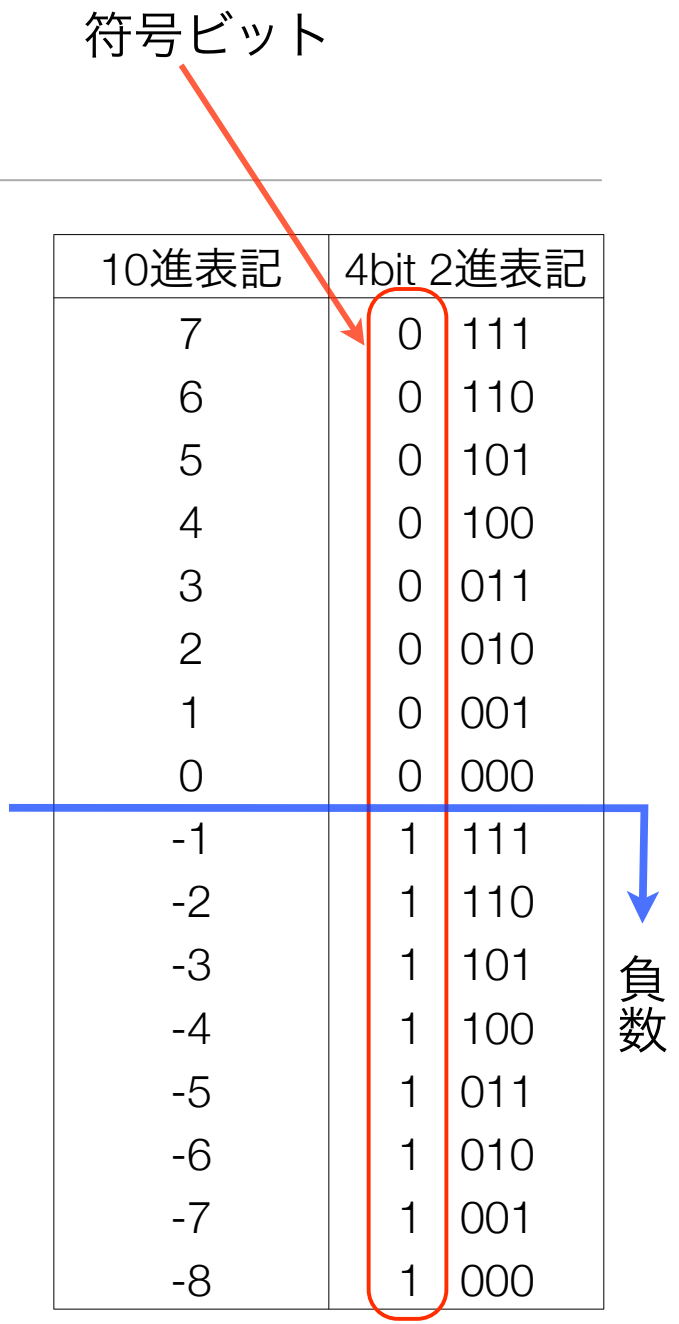
Sは符号 (sign)

S	7bit データ

- なぜ？

  負数を単純に加算することができる

  桁あふれをどう処理するかはソフトが判断すればよい

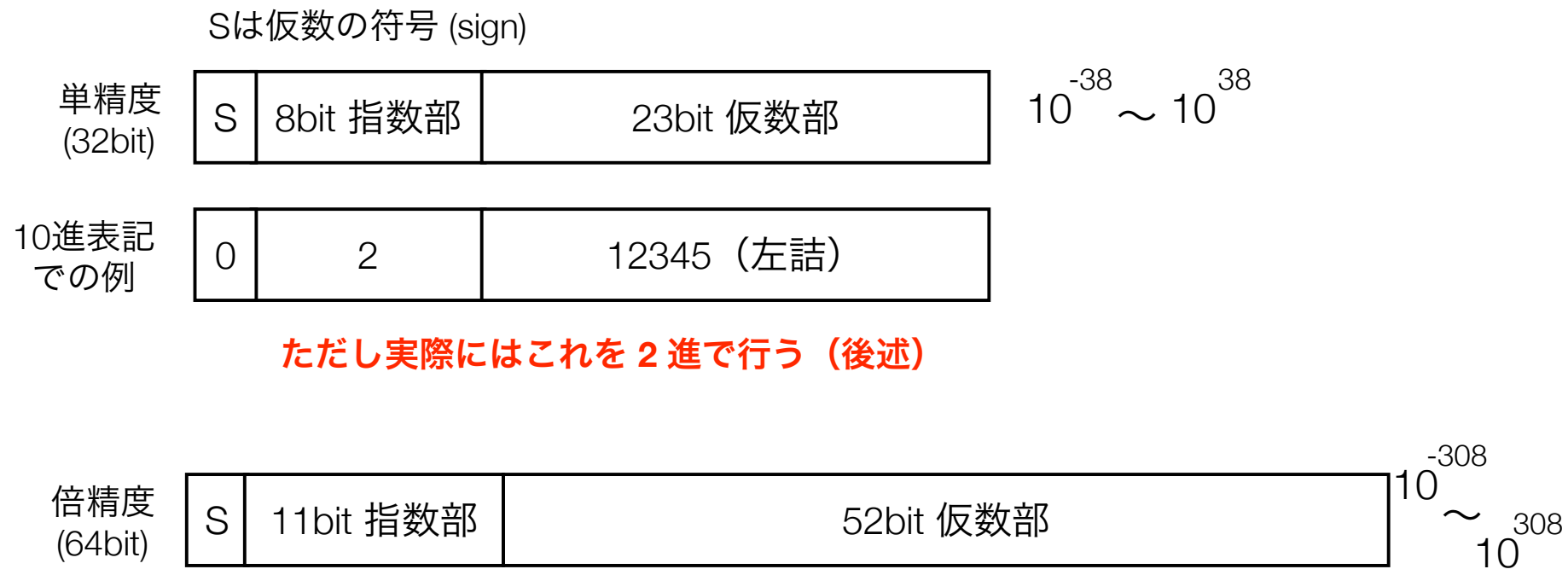


10進表記	4bit 2進表記
7	0 111
6	0 110
5	0 101
4	0 100
3	0 011
2	0 010
1	0 001
0	0 000
-1	1 111
-2	1 110
-3	1 101
-4	1 100
-5	1 011
-6	1 010
-7	1 001
-8	1 000

少数

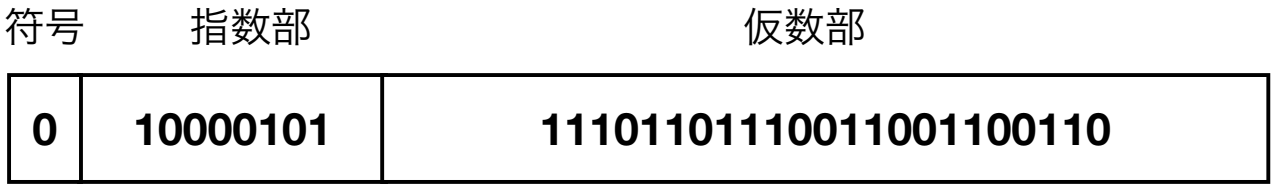
- 浮動小数点表現

  123.45 → 1.2345 x 10^2
  指数部(^2)と仮数部(12345)に分けて表現



実際の浮動小数点表現（参考）

- 123.45 の浮動小数点表記は？



10000101=128+5　**1**　**111011 . 0111001100110011**

略された 1 をつけて、6桁めに小数点

バイアスの127を
引いた 6 桁めに少数点

1111011 = 123（整数部）
続く 011100... が小数部
（つまり 0.25 + 0.125 + 0.0625...）

表現できた数値は 123.4499969...　（有効桁数は24bitつまり7桁少し）

他の数値の例

0.1 = 0, 01111011, 10011001100110011001101（誤差がある）

0.5 = 0, 01111110, 00000000000000000000000（割り切れた）

誤差・有効桁数

- 有効桁数に注意

Microsoft Excel 2004 for Macintosh での数値の扱い

乗数 (Nとする)	2のN乗	2のN乗に +1 した結果
8	256	257
16	65,536	65,537
32	4,294,967,296	4,294,967,297
64	18,446,744,073,709,600,000	18,446,744,073,709,600,000

どこでおかしくなったのだろうか？

48	281,474,976,710,656	281,474,976,710,657
49	562,949,953,421,312	562,949,953,421,313
50	1,125,899,906,842,620	1,125,899,906,842,620
51	2,251,799,813,685,250	2,251,799,813,685,250

末尾はそもそも 4 であるはずだが、15桁位置で切り捨てられ、+1 も反映されない

デジタル化による利益

まとめ：デジタルデータとフォーマット（再掲）

- その実体は数値（記号）の列

  音声：111,121,122,89,80,82,75….

  静止画：10,240,22,30,34,80…

  音声付き動画：12,33,45,1123,488…

  文字：33,38,42,60,32,39,55,80…

- これだけでは利用できない（意味が汲み取れない）

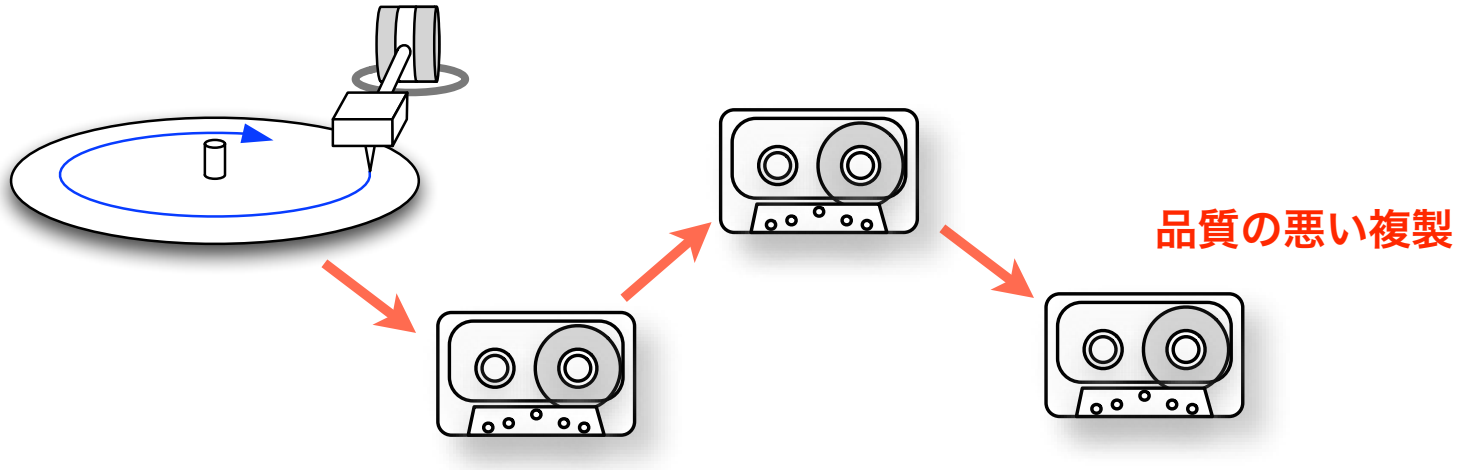
  符号化ルールとデータは常に一体

- このルールが**フォーマット（書式）**を生む

デジタル化による利益

- 数値による表現
  - 文字・映像・音声
  - 連続的な値の変化ではなく、離散的な数値として表現
- そうすることの利益は？

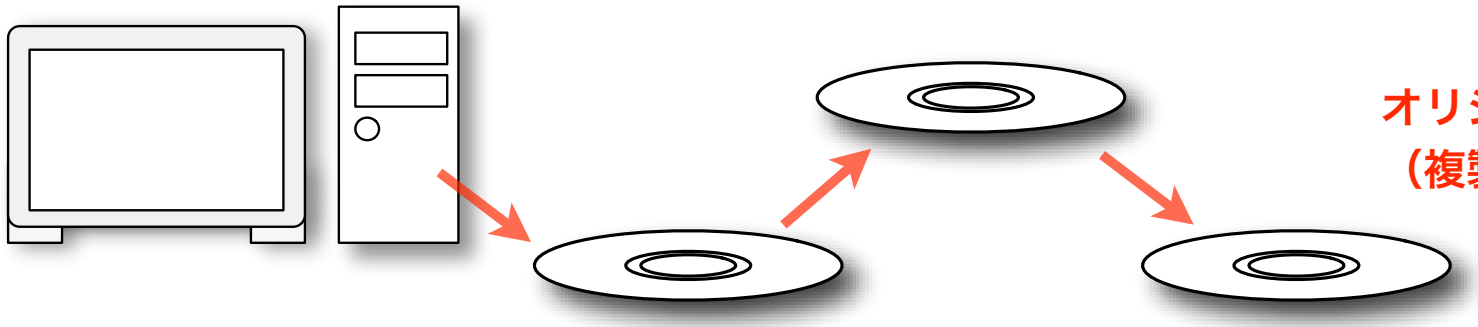
完全な複製（復習）



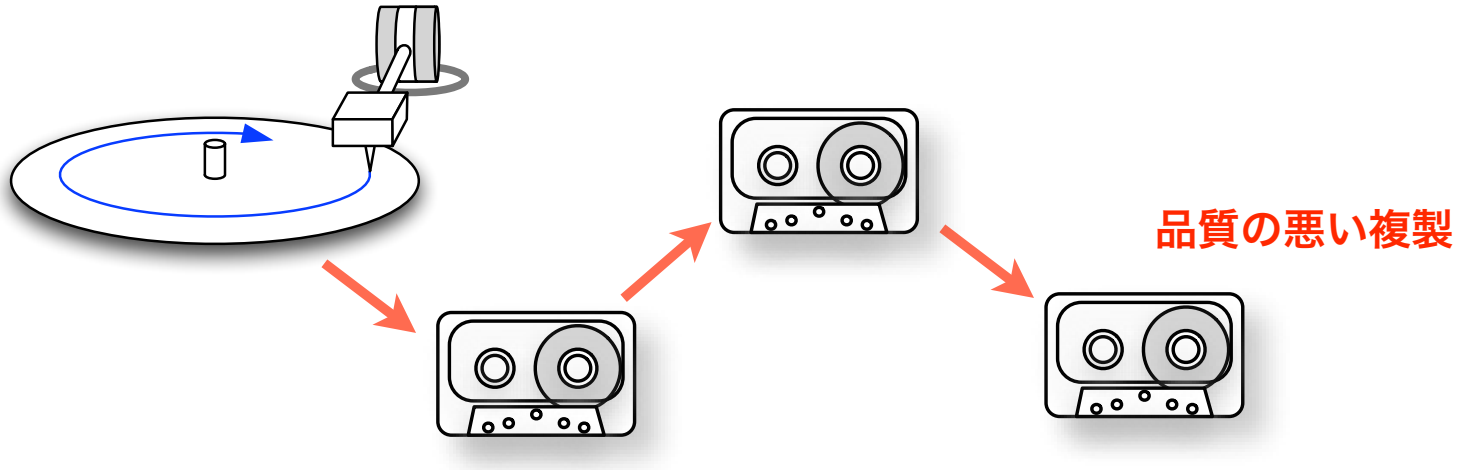
レコード・テープ　テープレコーダーを使ってレコード・テープ間の複製をとる

複製

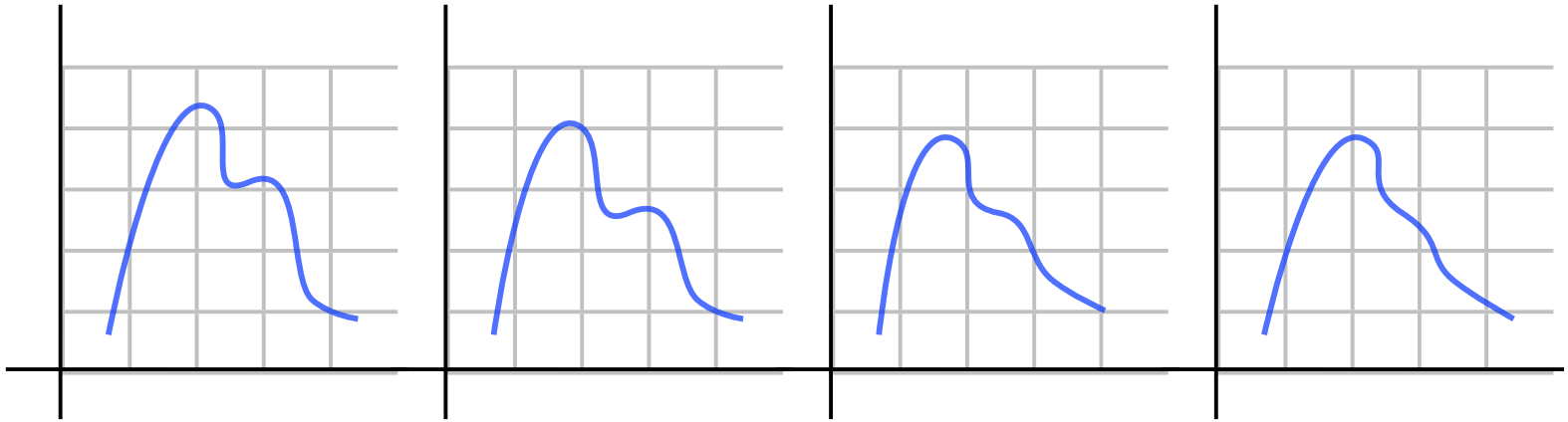
CD/CD-R　パソコンを使って CD / CD-R 間の複製をとる



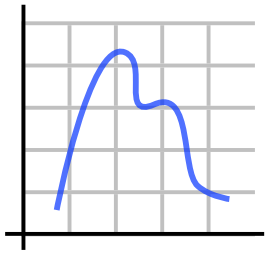
完全な複製（復習）



レコード・テープ　テープレコーダーを使ってレコード・テープ間の複製をとる



完全な複製（復習）



このサンプリング時点では誤差が生じる

4,3,4,7,7,8,8,5,3,2,2…

4,3,4,7,7,8,8,5,3,2,2…

4,3,4,7,7,8,8,5,3,2,2…

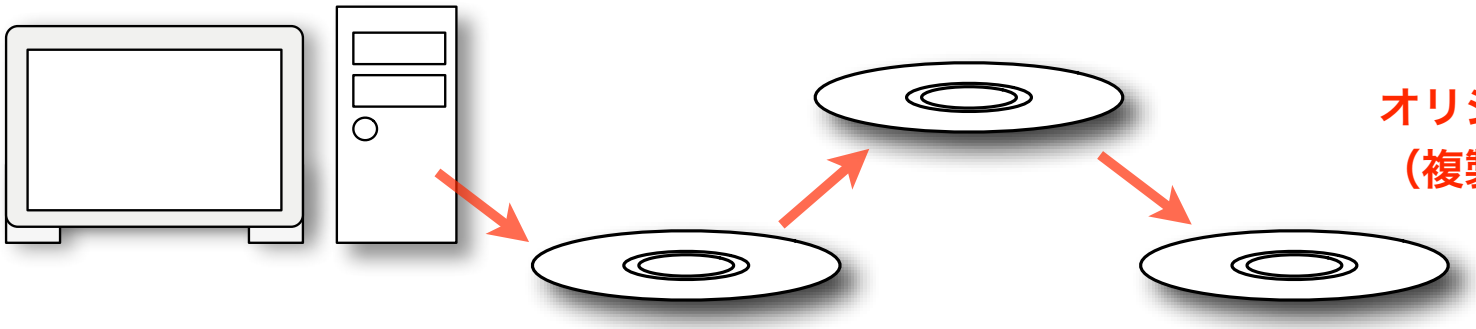
4,3,4,7,7,8,8,5,3,2,2…

扱うのは波形ではなく数値

いつまでもオリジナルと同一

CD/CD-R

パソコンを使って CD / CD-R 間の複製をとる



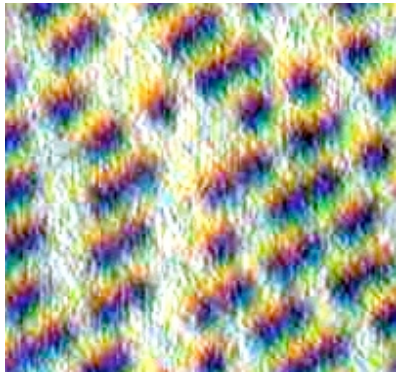
オリジナルと同一（複製と呼ぶべきか？）

ノイズへの抵抗・完全な複製（復習）

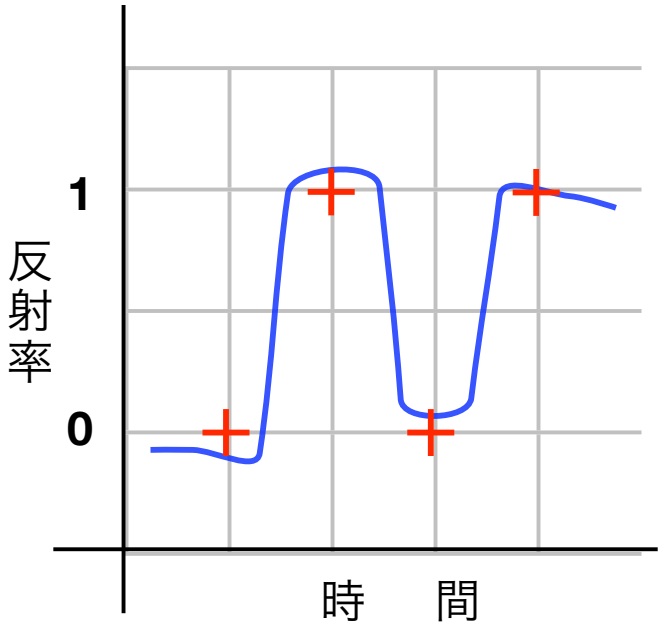
- 1/2量子化単位以下の狂いであれば正しい値が得られる

  二値化されている場合は 0/1 を違えなければ良い

- 再複製の際に狂いが継承（蓄積）されない



CDのピット長は9種類



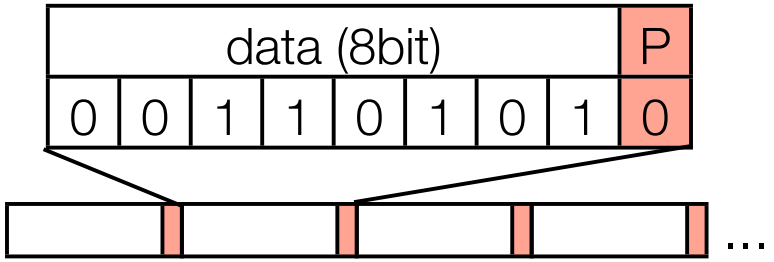
0 or 1 を間違えない程度に反射率の違いを検出できれば良い

最大ピット長の 1/9 以下程度の誤差で長さ検出できれば良い

誤り検出・訂正

- 違った値が得られた場合の検出・修正が可能
- 修正のための冗長な情報を付加
- 誤り検出の例：
- パリティ（奇偶性）- 1 bit 付加

  1 bit の誤りを検出可能（2 bit の同時誤りは駄目）



- チェックサム

  学生番号の合計は常に最下桁がゼロ（試してみよ）

- CRC（Cyclic Redundancy Check）

誤り検出・訂正

- 誤りを正せるような情報を加える
- 誤り訂正の例：
- 縦横チェックサム
- メモリにおける ECC （Error Correcting Code） ㈱

  64bit のデータに 8bit のECC情報を付加
  1bit の誤りを検出・修正
  2bit の誤りは検出のみ（修正不可能）

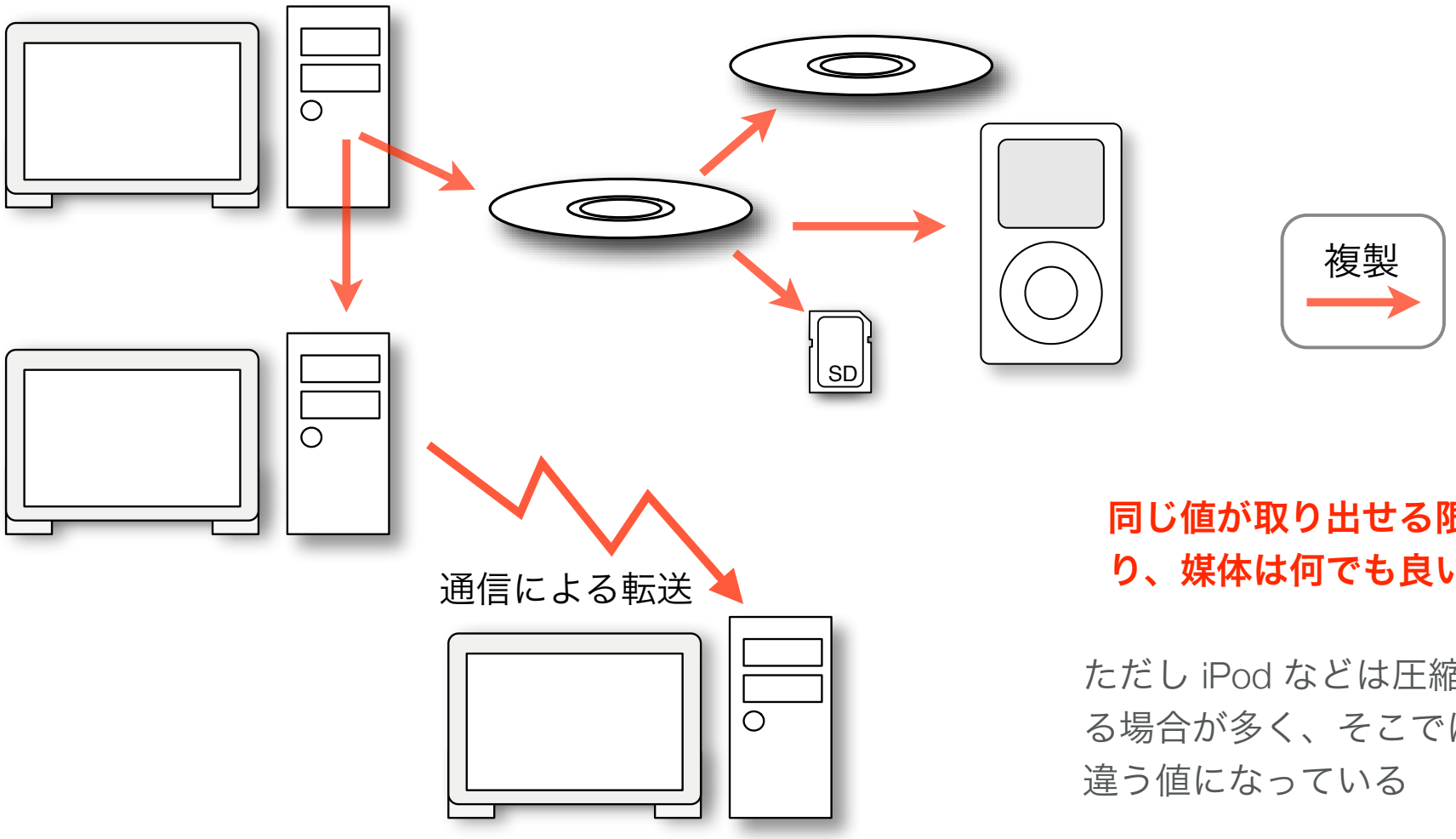
- CIRC : CD
- より多くの付加情報によってより広範囲な修正に対応

*ECC メモリは Error check and correct memory の略？

メディアの非依存性（復習）

CD/CD-R

パソコンを使って CD / CD-R / iPod / メモリカード 間の複製をとる



同じ値が取り出せる限り、媒体は何でも良い

ただし iPod などは圧縮する場合が多く、そこでは違う値になっている

まとめ：デジタルデータの特徴

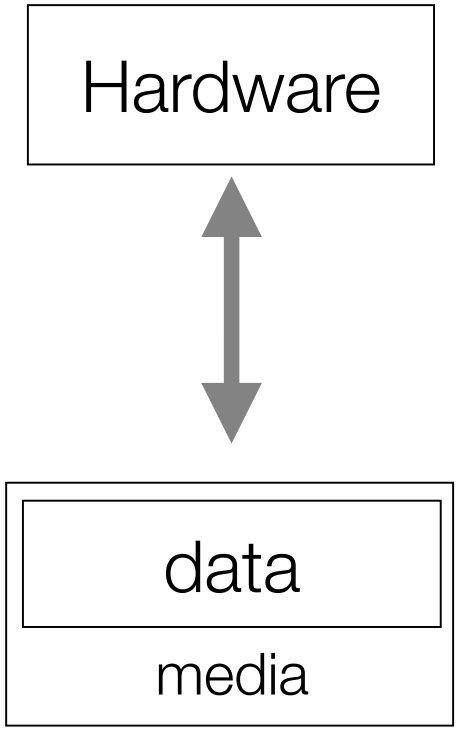
- 完全な複製
  - 複製・通信・保存に伴う劣化の回避
  - 完全さの検証も可能
- 不完全なデータ化
  - 初期ノイズの発生（近似でしかない）
- 考え方
  - 初めに精度を決めることでそれ以後の精度以内の変化をゼロにした
- 利益
  - 数学的なテクニックが多く適用可能に
  - コンピュータによる知的な自動処理が可能に

全体のまとめ

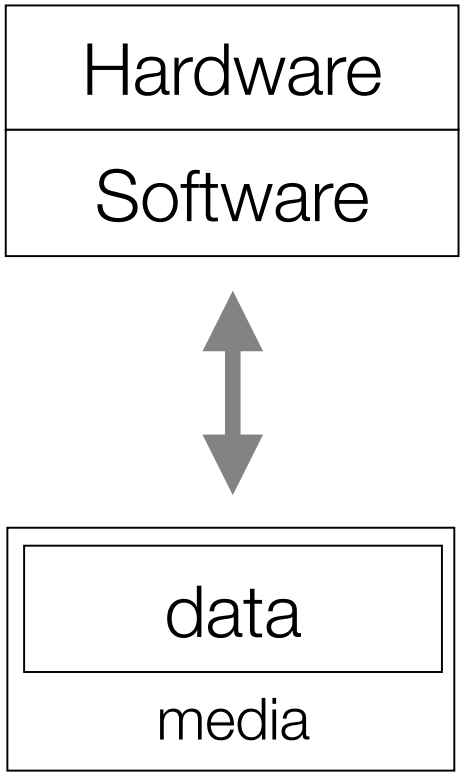
- デジタルデータのメリット
  - 完全な複製
  - デジタルコンピュータによる自動処理
- デジタルデータとフォーマットの関係
  - デジタル化で情報はメディア（物理的制約）からは自由になったがフォーマットが重要になった
  - 互換性という概念

デジタルシステムの柔軟性

アナログシステムとデジタルシステム

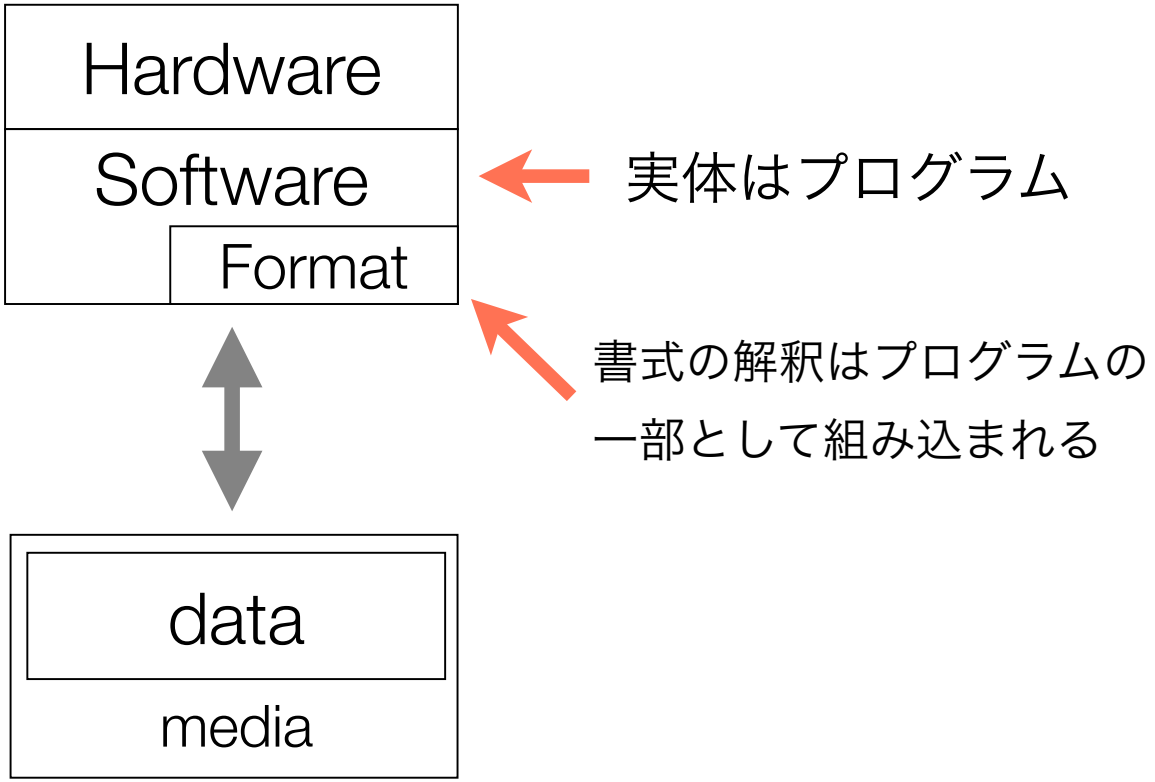


典型的なアナログシステム
（レコードプレーヤーなど）



典型的なデジタルシステム
（コンピュータなど）

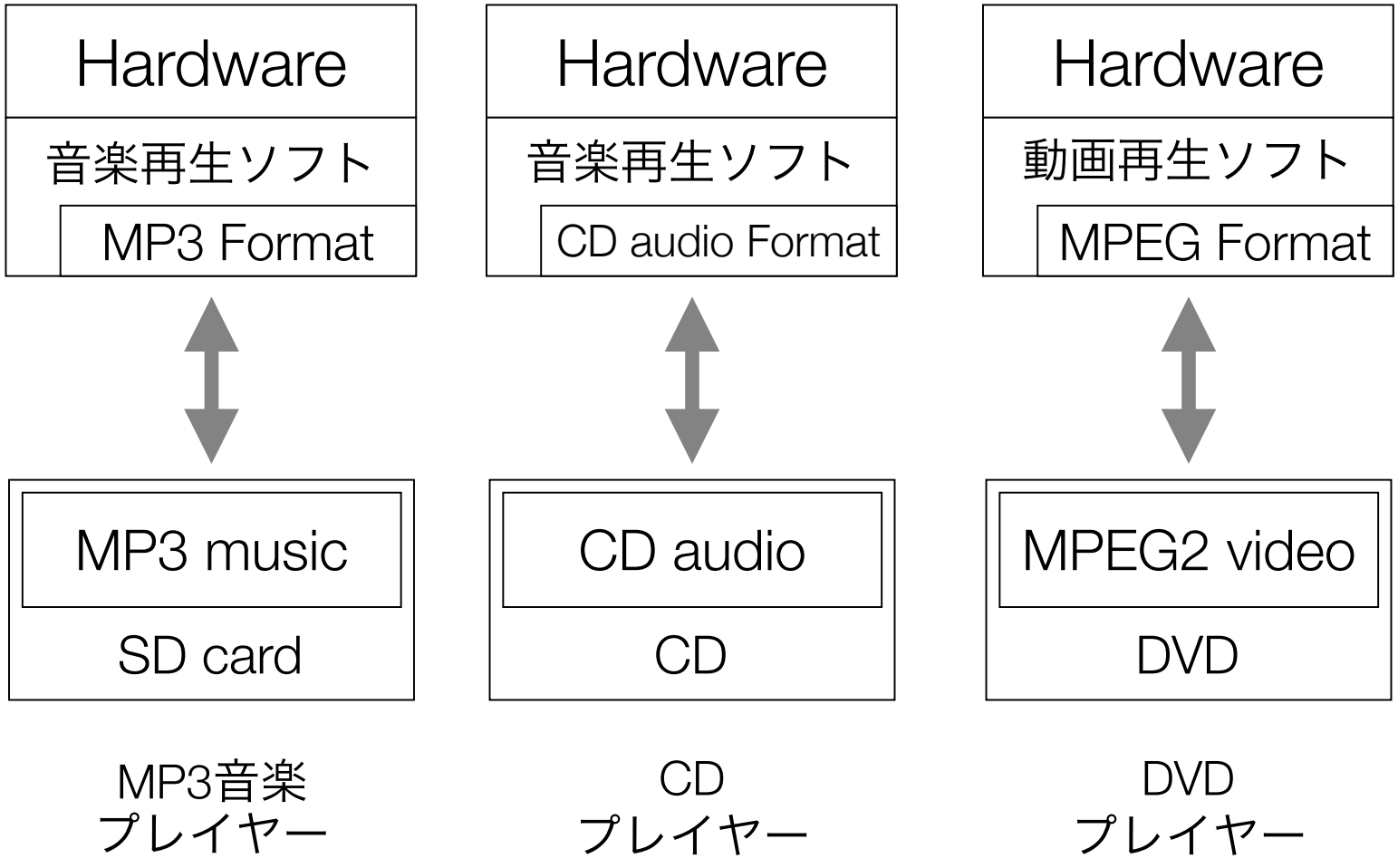
書式とデータの関係



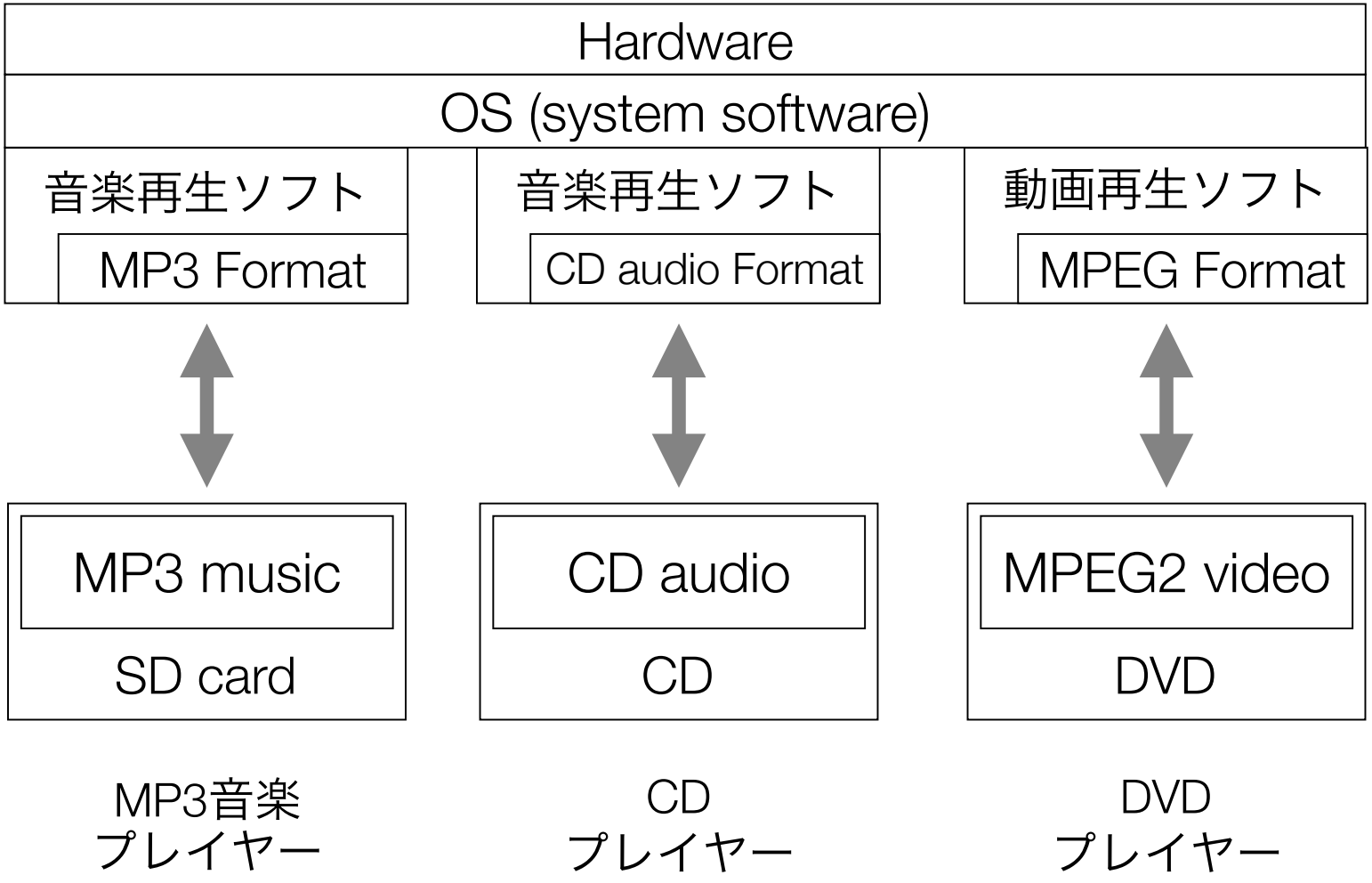
典型的なデジタルシステム

（コンピュータなど）

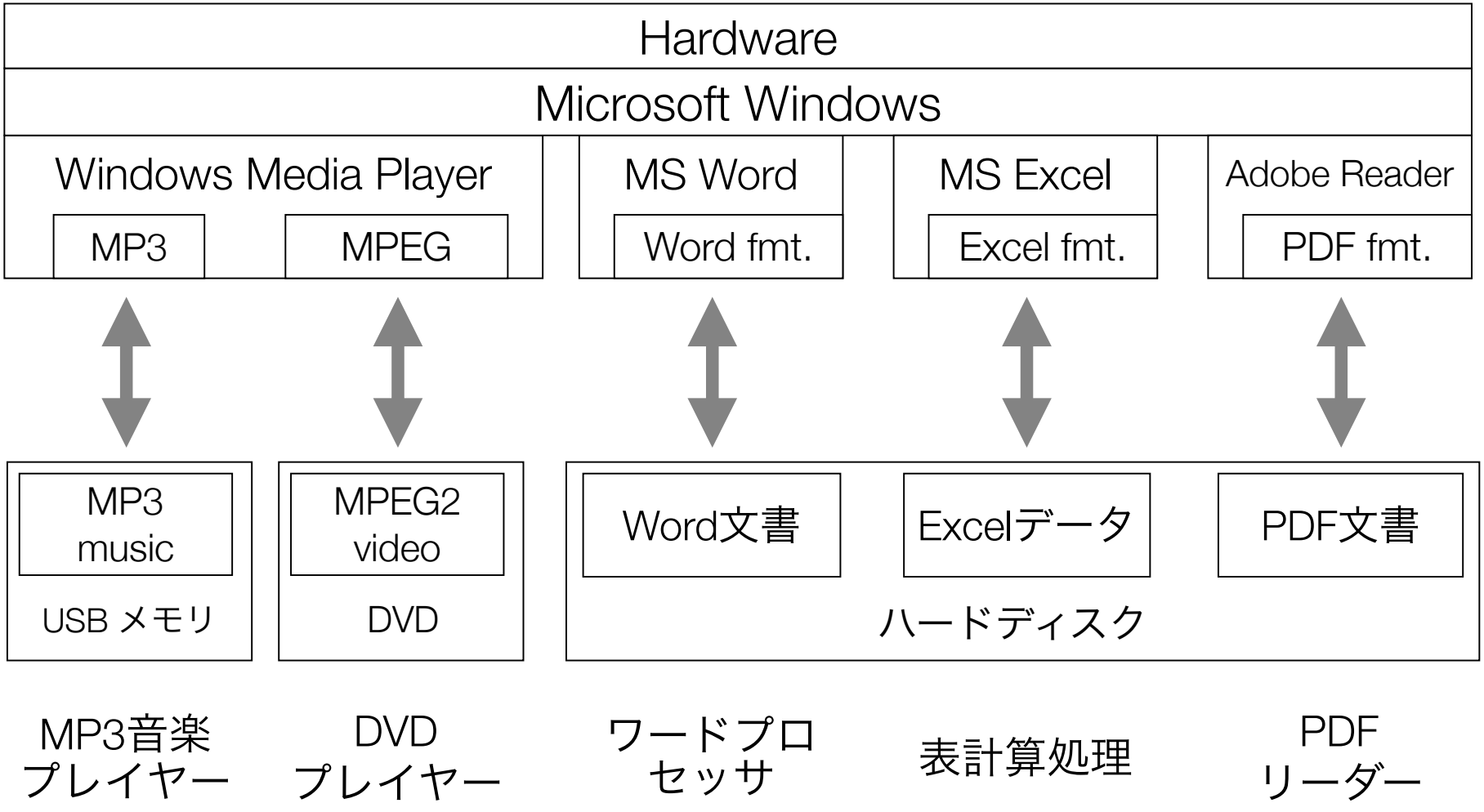
デジタルシステムの柔軟性



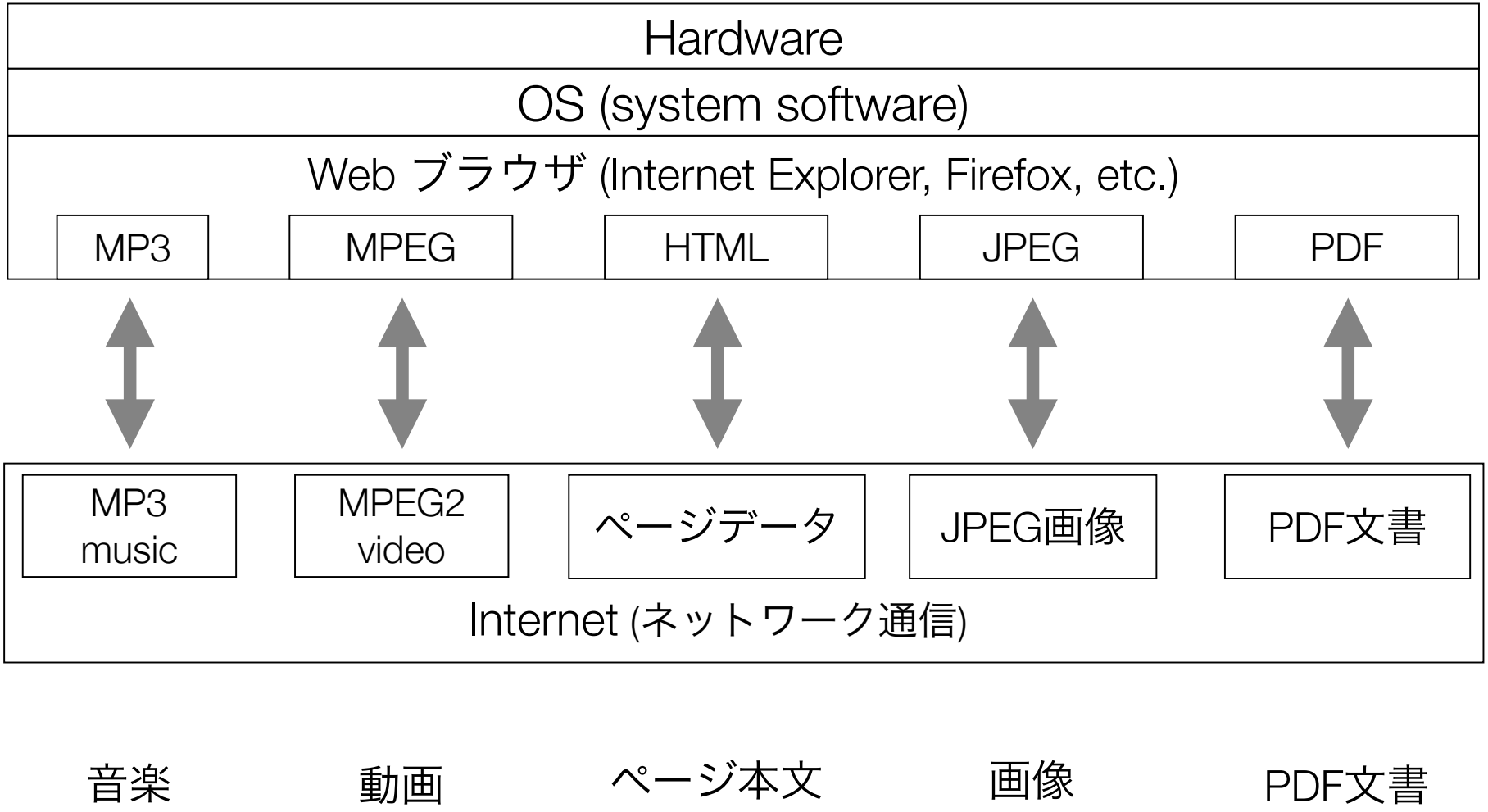
PC : 汎用デジタル処理システム



いつも使っている Windows パソコン



Web ページ閲覧におけるデータ処理



デジタル化のインパクト

- 汎用性
  - 情報はフォーマットと値で表現される
- 汎用(generic)のものに特定(specific)の機能を載せる
  - 汎用データ通信網に特定用途サービスを載せる
  - このサービスを汎用コンピュータに特定用途アプリケーション・ソフトウェアを載せて実現
  - ソフトウェアを入れ替えて新しい機能を実現可能
  - ソフトウェアで対応することの柔軟性

全体のまとめ

- デジタルデータのメリット
  - 完全な複製
  - デジタルコンピュータによる自動処理
- デジタルデータとフォーマットの関係
  - デジタル化で情報はメディア（物理的制約）からは自由になったがフォーマットが重要になった
  - 互換性という概念
- デジタル化のインパクト
  - ソフトウェアによる柔軟性

事例紹介

- HTML5 の動画フォーマット
  - Google / WebM の VP8 （元 On2 の Ogg Theora）が意味するものは何か？
  - Apple は H.264 を推している
  - 何故か？

事例紹介

- HTML5 の動画フォーマット
  - Google / WebM の VP8 （元 On2 の Ogg Theora）が意味するものは何か？
  - Apple は H.264 を推している
  - 何故か？
- 互換性
  - 全ての環境で使えるフォーマットが我々には必要だ