コンピュータ・サイエンス1

第11回
アナログ情報とデジタル情報

人間科学科コミュニケーション専攻

白銀 純子

第11回の内容

- アナログ情報とデジタル情報

設問2

- やってみよう! の演習用文字コードを使った問題の1.の問題の解答を報告すること
  - 1. の問題: 62B7 2654 4C25 1D7F 720B

	62B7	2654	4C25	1D7F	720B
文字コードA	つ	け	さ	き	と
文字コードB	さ	い	え	ん	す
文字コードC	へ	よ	や	り	ひ

解答: 文字コードB, さいえんす

設問3

- 下記の中で正しい説明を全て選びなさい
  1. Unicodeを使ったとしても、機種依存文字はありえる
     - 機種依存文字は規定された文字集合に含まれない文字なのでありえる
  2. Unicodeでは、日本語の文字とギリシア語の文字で、同じ番号を使っていることがある
     - Unicodeは複数の言語の文字を統一して扱う規格なので、存在しない
  3. UTFで番号を割り振られている文字の中には、UCSの規定に含まれていない文字も存在する
     - UCSで規定された文字に番号をつける方法がUTFなので、存在しない
  4. Unicodeを使うと、1つの文書の中で日本語やフランス語、アラビア語など様々な言語を使うことができる
     - これがUnicodeの目的なので、できる

解答: 1, 4

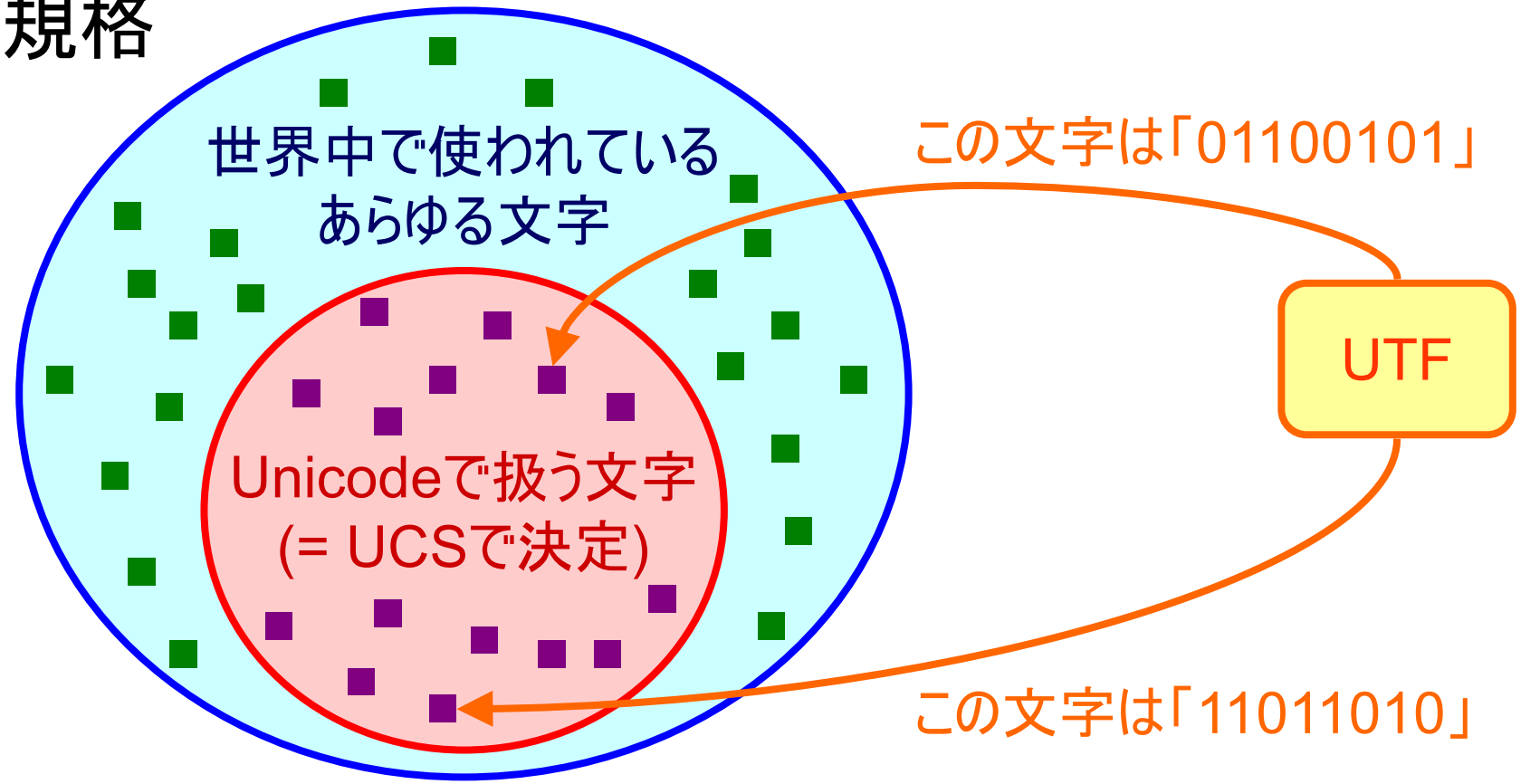
前回の質問の解答

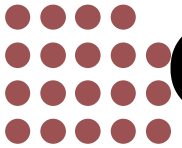
いろいろな文字

- スペース: 図形文字に入るか、制御文字に入るかは諸説もろもろ
- 半角カナ: 1バイト
  - ASCIIが使っていない8ビット(1バイト)の番号を使用
  - コンピュータの性能が低かった時代に、できるだけ資源(記憶容量や処理能力)を少なくするために作られたもの
  - 現在は全角でまったく問題ない処理能力(ケータイであっても)であり、逆に半角カナを使うことで文字化けの原因になりがち

UCSとUTF

- UCS: 世界中のあらゆる文字のうち、どの文字をUnicodeで扱うかを決めた規格
- UTF: UCSで決めた文字に対し、どのように0と1を割り当てるかを決めた規格





Question!

復習(浮動小数)

浮動小数点方式[1](p. 10)

- 小数: D × 10^nと表現できる
  - Ex. 1: $0.5 = 5 \times 10^{-1}$
  - Ex. 2: $-0.0625 = -6.25 \times 10^{-2}$
  - Ex. 3: $0.0000000084 = 8.4 \times 10^{-9}$

どの数でも「$\times 10^n$」の「10」の部分は同じ(実際のコンピュータでは「$\times 2^n$」)

小数を「$D \times 10^n$」の形と考え、「D」と「n」だけ記憶しておく

浮動小数点方式[2](p. 10)

- 浮動小数点方式:
  小数を「$D \times 10^n$」と考え、「D」と「n」を記憶することで小数を表す方式

Ex.

- D = 6.25, n = -3の場合: 0.00625
- D = 6.25, n = -2の場合: 0.0625
- D = 6.25, n = -1の場合: 0.625
- D = 6.25, n = 0の場合: 6.25

nの数値が何かで、小数点が仮数の中を動くように見えるから「浮動小数点」と名づけられた

D: 仮数部
n: 指数部
と呼ぶ

- 仮数部
  - ✓ 符号は「0」が「+」、「1」が「-」
  - ✓ 固定小数点方式
- 指数部
  - ✓ 符号は「0」が「+」、「1」が「-」(ただし、2の補数表現とは別の特殊な形で表現される)

大きな数の表現(p. 10)

- 浮動小数点方式を利用して表現

Ex.

- $2000000000000 = 2 \times 10^{12}$
- $-42500000000000000 = -4.25 \times 10^{17}$

指数部が「+」の数になる

コンピュータは「2」と「+12」, 「-4.25」と「+17」を記憶しておく
(実際には、「$\times 10^n$」ではなく「$\times 2^n$」で表現)

浮動小数の計算方法

1. 仮数部の有効桁数が小数点第X位の数にしたときにどうなるかを考える
   - 「X」は問題で与えられる
2. 1. の数になるために、小数点「.」が何桁分移動するかを数える
3. 2. の移動が右向きに移動であれば、2. の数に「-」(マイナス)をつける
   - 左向きに移動であれば、2. の数には何もしない
4. 数を「(1. の小数)×10(3. の数)」の形で表す

Ex. 1: 0.000000001234を、仮数部の有効桁数が小数点第2位の浮動小数として表すこと

1. 仮数部の数: 12.34
2. 12.34になるために、小数点は10回移動する
3. 2. の移動は右向きである→ 2. の回数を「-10」にする
4. 0.000000001234を浮動小数で表すと: 12.34×10^{-10}

復習(文字の表現)

日本語の文字(p. 15)

- ASCII
  - 1文字を8ビットで表現→全部で256文字分表現可能
  - 現状で128文字存在(128文字分利用されているので残りは128文字)
- 日本語
  - ひらがな: あ～ん(ゐ, ゑなどの旧字を含む), 濁音・半濁音, 小文字(「ぁ」, 「ぃ」など)
  - カタカナ: ア～ン(ヰ, ヱなどの旧字を含む), 濁音・半濁音(ヴを含む), 小文字(「ァ」, 「ィ」, 「ヵ」, 「ヶ」など)

169文字

ひらがな・カタカナだけでもASCIIでは表現できない

日本語文字集合の規格(p. 16)

- 現状での日本語文字集合の規格: JIS X 0208:1997
  - ひらがな・カタカナ・漢字・非漢字文字で6879個

JIS第1水準(使用頻度の高い漢字): 2965個
JIS第2水準(使用頻度の低い漢字): 3390個

- 2^{13} = 8192なので、13ビットで表現可能
- コンピュータ処理では、バイト単位(8ビット単位)が好都合

16ビット(2バイト)で日本語1文字を表現

ASCII文字との区別(p. 16)

- 日本語の文書
  - 日本語の2バイト文字
  - ASCIIの1バイト文字

混在

日本語の2バイト文字(JIS X 0208)とASCIIの1バイト文字は区別する必要
(1つの文書の中で、どれが2バイト文字でどれが1バイト文字か)

- モード切り替えによる区別方法
- ASCII文字の番号を避ける区別方法

モード切り替え(p. 16)

- 文字集合切り替えのための特別な記号を用意
  - ここから先はASCII文字
  - ここから先は日本語文字
  - ここから先は中国語漢字
  - etc.

エスケープシーケンス

通常の文書では頻繁に文字集合が切り替わることがなく、同じ文字集合に属する文字が現れることが多いという性質を利用

- 国際標準規格: ISO-2022
- 日本語に適用したもの: ISO-2022-JP

モード切り替えの問題(p. 17)

- 文書を先頭から順番に見ていく場合には問題ない
- 文書を途中から見ていくときに問題が生じる
  - 見始めた途中の文字が、ASCII文字か日本語文字か、エスケープシーケンスかが判別できない

Ex. 見始めた途中の文字が「70」番だった場合

- ASCII文字の「F」?
- 日本語文字の一部?
- 韓国語の一部?

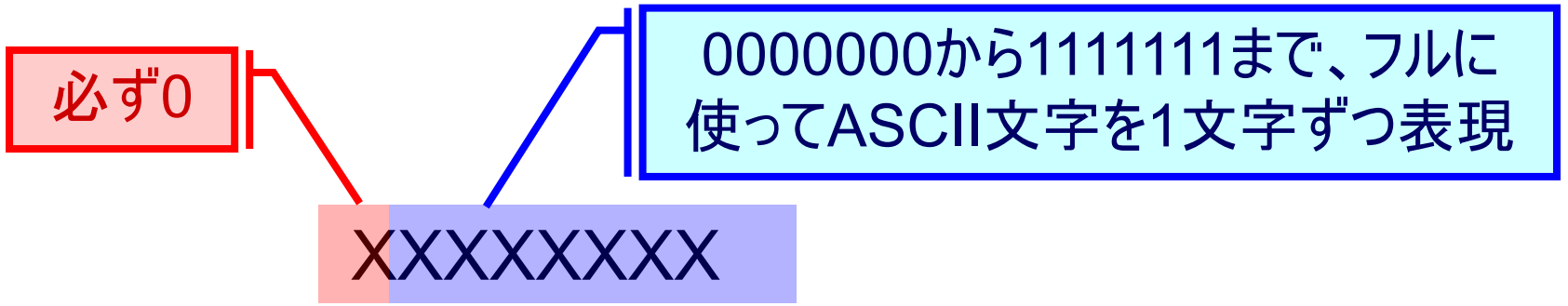
検索や置換などの文書処理に時間がかかる

ASCII文字の番号を避ける(p. 17)

- ASCIIで使われていない番号を2バイト文字の番号にあてる方法
  - EUC(日本語のものをEUC-JP)
    - 第1バイト(前半の8ビット)と第2バイト(後半の8ビット)両方でASCII領域が避けられている
  - SJIS(Shift JIS)
    - 第2バイト(後半の8ビット)ではASCII領域も使われている
  - 文章のバイト数は、単純に、日本語文字で2バイト、ASCII文字で1バイトで数えれば良い

EUCとSJIS[1](p. 17)

- ASCII文字: 8個の0と1で、1文字分を表現
  - 実際には、7個の0と1で1文字分を表現
  - 8ビット目は必ず0



= ASCII文字は、0XXXXXXX という形
Ex.「a」を0と1で表現すると: 01100001

8ビット目が1になる番号(1XXXXXXXという形の番号)は
ASCII文字ではない

言語圏ごとの文字コード(p. 18)

- これまでの多バイト文字の扱い: 異なる言語圏ごとに文字集合を作成

様々な文字集合ができてしまって不便

- コンピュータネットワークの国際化が進んだ
- コンピュータの資源が豊富になった

国際文字集合規格として各文字集合を統一化

- ASCII, ラテン文字, 日本語, 韓国語, 中国語, ベトナム語, ギリシャ文字, 記号, etc.

Unicode: どの文字を扱うかと文字の符号化の方法を決めた規格

- UCS(Universal multi-octet coded Character Set)でどの文字を扱うかを規定
- UTF(UCS Transformation Format)で文字の符号化の方法を規定

UTF-8(p. 18)

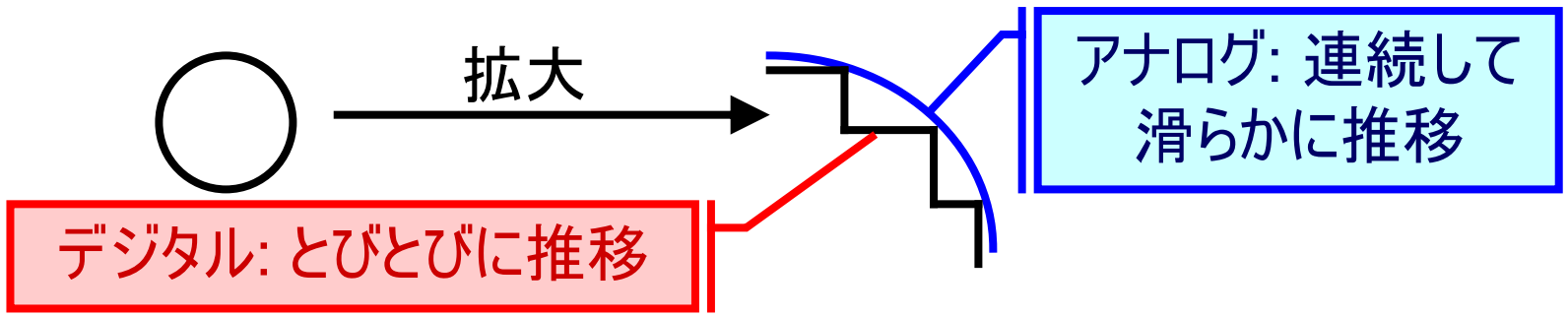
- Unicodeでの代表的な符号化方式(符号化方式はいくつか存在)
- 1文字を1～6バイトの可変長(文字によってバイト数が異なる)で符号化する方式
  - ASCIIやISO-2022-JP、Shift JIS、EUC-JPは1文字を同じバイト数で表現
- OS(WindowsやMacなどのオペレーティングシステム)でファイル名などの内部処理に利用
  - 半角英数を符号化した結果が、ASCII文字と全く同じになるため、従来のシステムと相性が良い

現在、UTF-8への移行が急速に進んでいる
- ただし、以前からのファイルを移行するのは大変なので、完全移行には時間がかかる
- 完全移行できたら、文字化けが起こらなくなる

アナログ情報とデジタル情報

アナログ情報とデジタル情報(p. 19)

- アナログ情報: 連続的な数値で表現できる情報
  - 物事を表現する数値の桁数が無限
  - Ex. アナログ時計: 1秒から2秒になるまで秒針が止まらず動き続ける
    (1秒と2秒の間も1.xxxx….秒が存在する)
- デジタル情報: 離散的な数値(とびとびの数値)で表現される情報
  - 物事を表現する数値の桁数が有限
  - Ex. デジタル時計: 1秒の次は2秒(1秒と2秒の間がない)



アナログ/デジタル情報の例(p. 19)

- アナログ情報
  - 現実世界の音
  - カセットテープやレコードに記録された音情報
  - 人の手で描いた絵
  - フィルムに記録された映像
- デジタル情報
  - CDやMDに記録された音情報
  - デジタルカメラが記録した画像情報
  - DVDに記録された音と動画の情報

アナログ情報のデジタル化(p. 20)

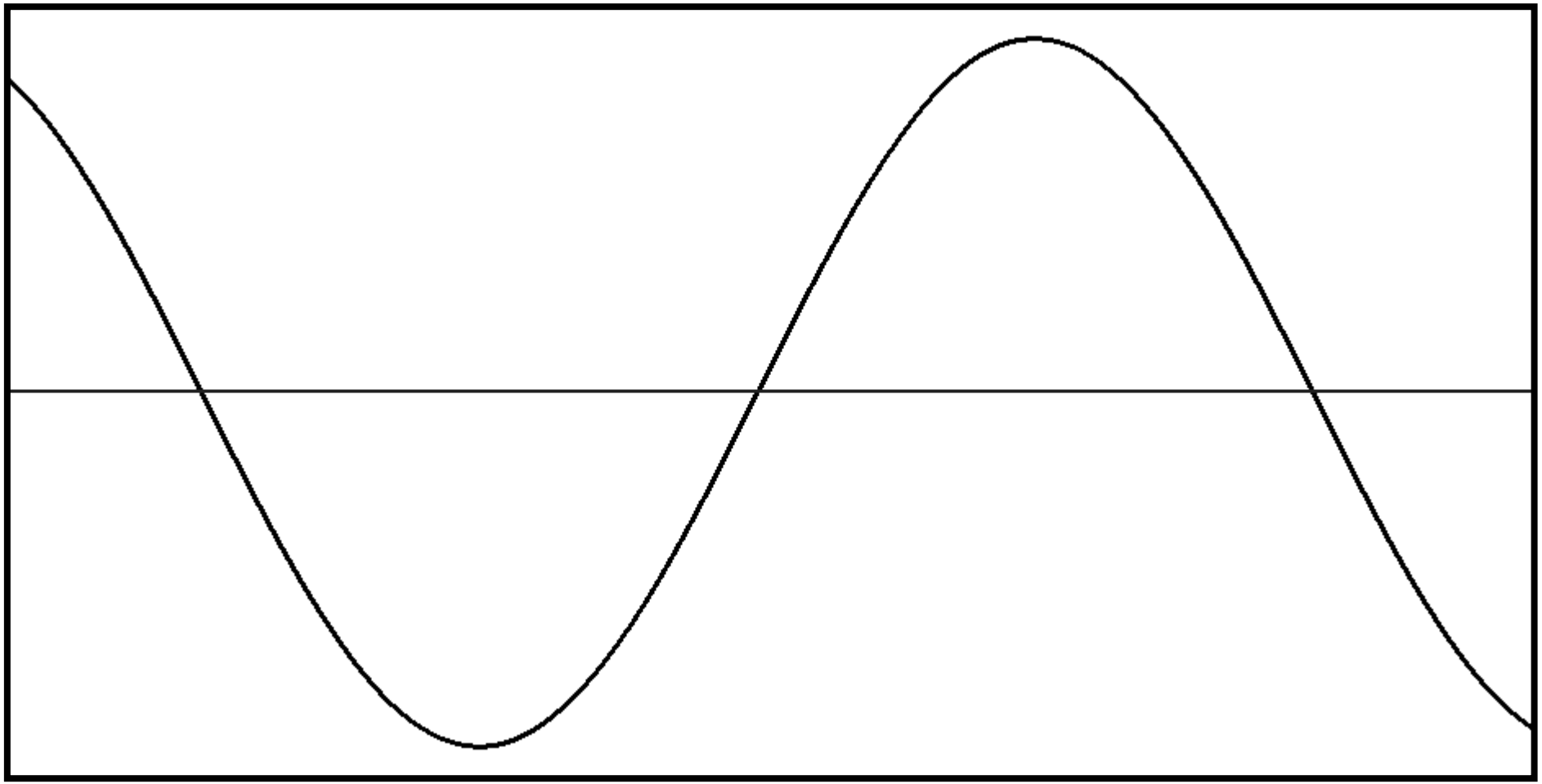
- アナログ情報: 数値化すると連続的
  - 音の強弱の変化
  - 画像の色の濃さ光の強弱の変化

グラフで表すと、なめらかな曲線(正弦波) = アナログ信号

デジタル化するには…
- 標本化
- 量子化

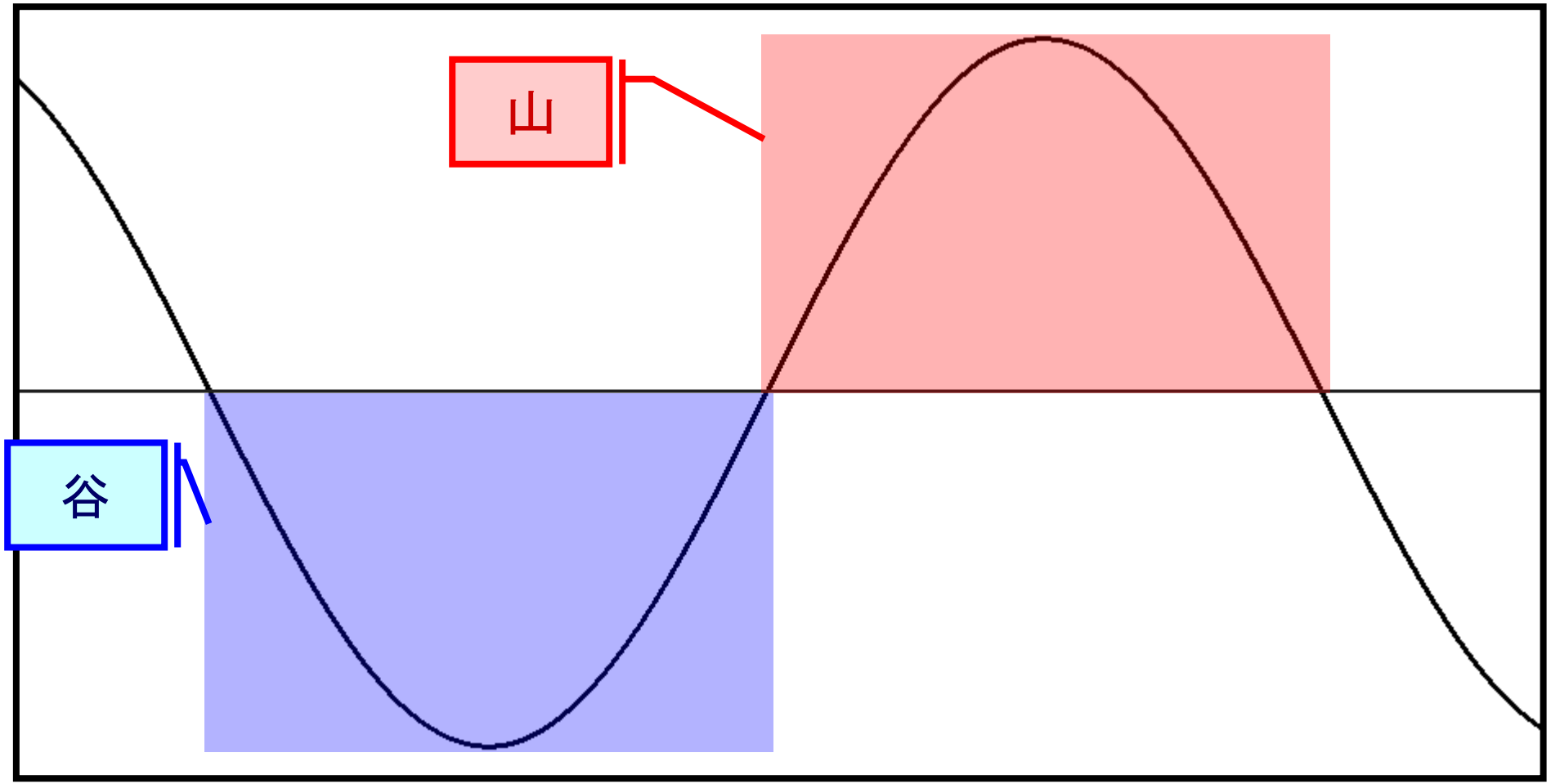
正弦波

- 波の形になっているグラフ
  - いくつかの特徴あり



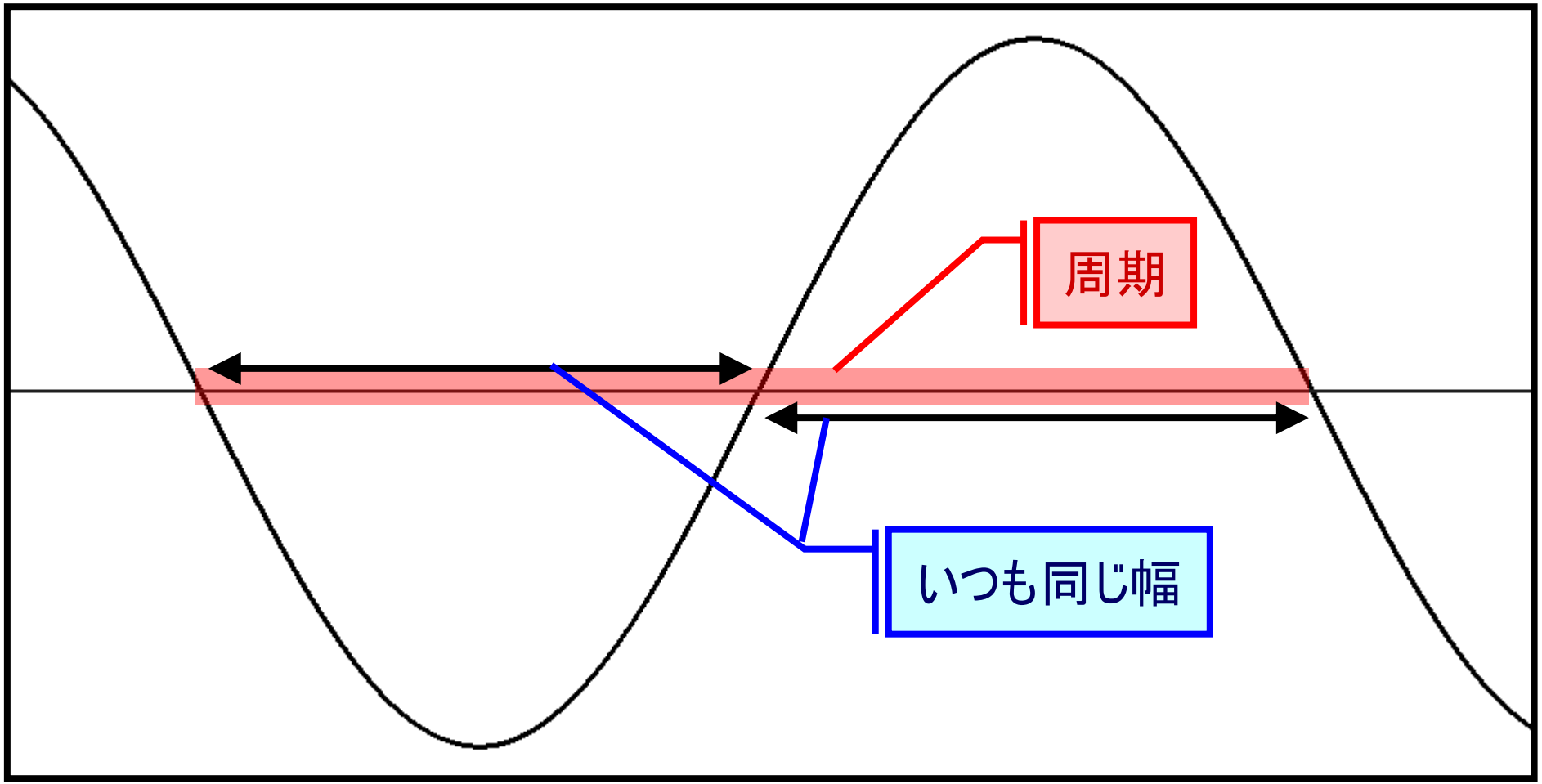
正弦波[特徴その1]

- 山と谷が交互に出現



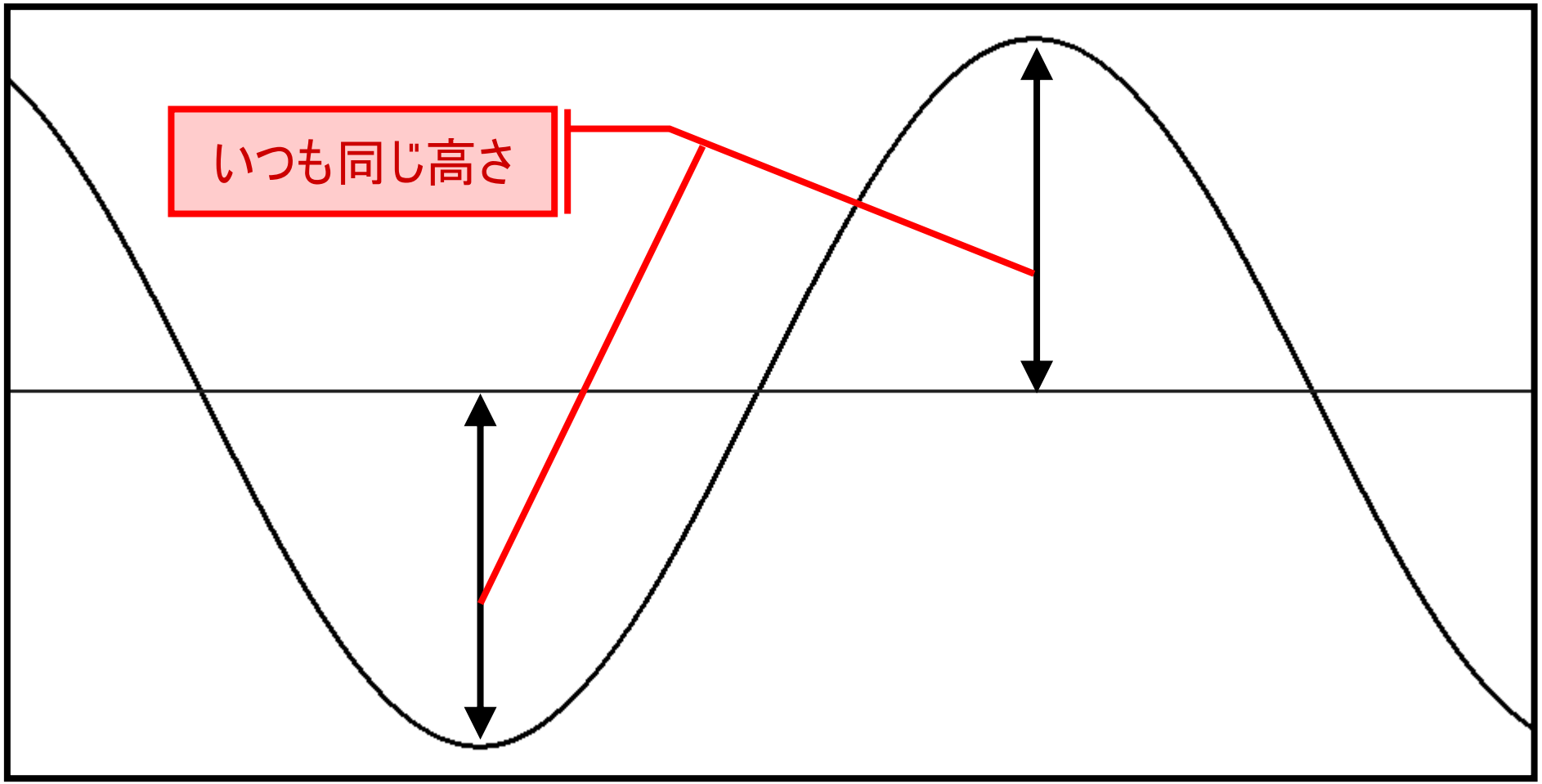
正弦波[特徴その2]

- 山と谷の幅をあわせたものを「周期」と呼び、山と谷の幅はいつも同じ



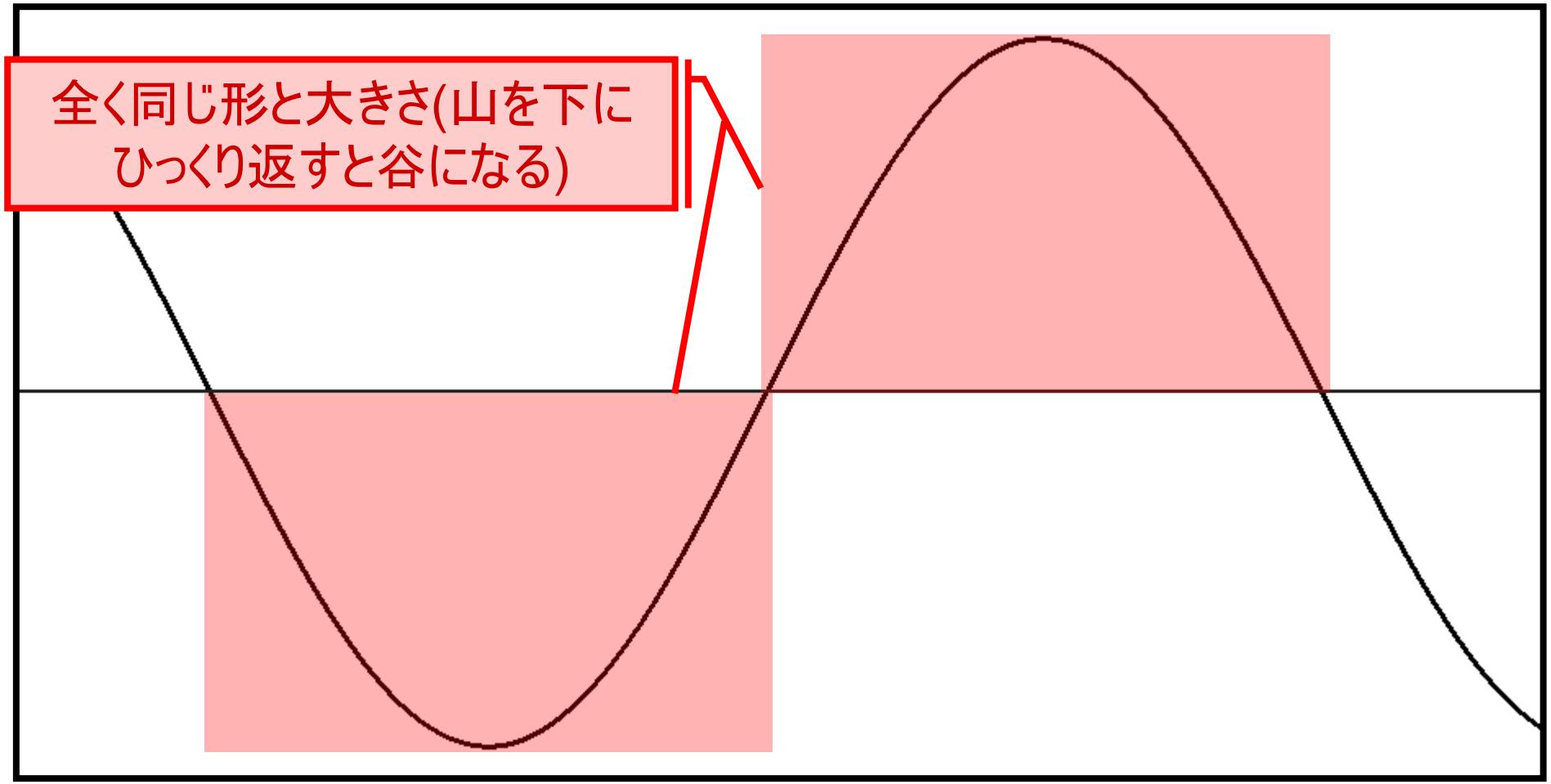
正弦波[特徴その3]

- 山と谷の高さ・低さはいつも同じ



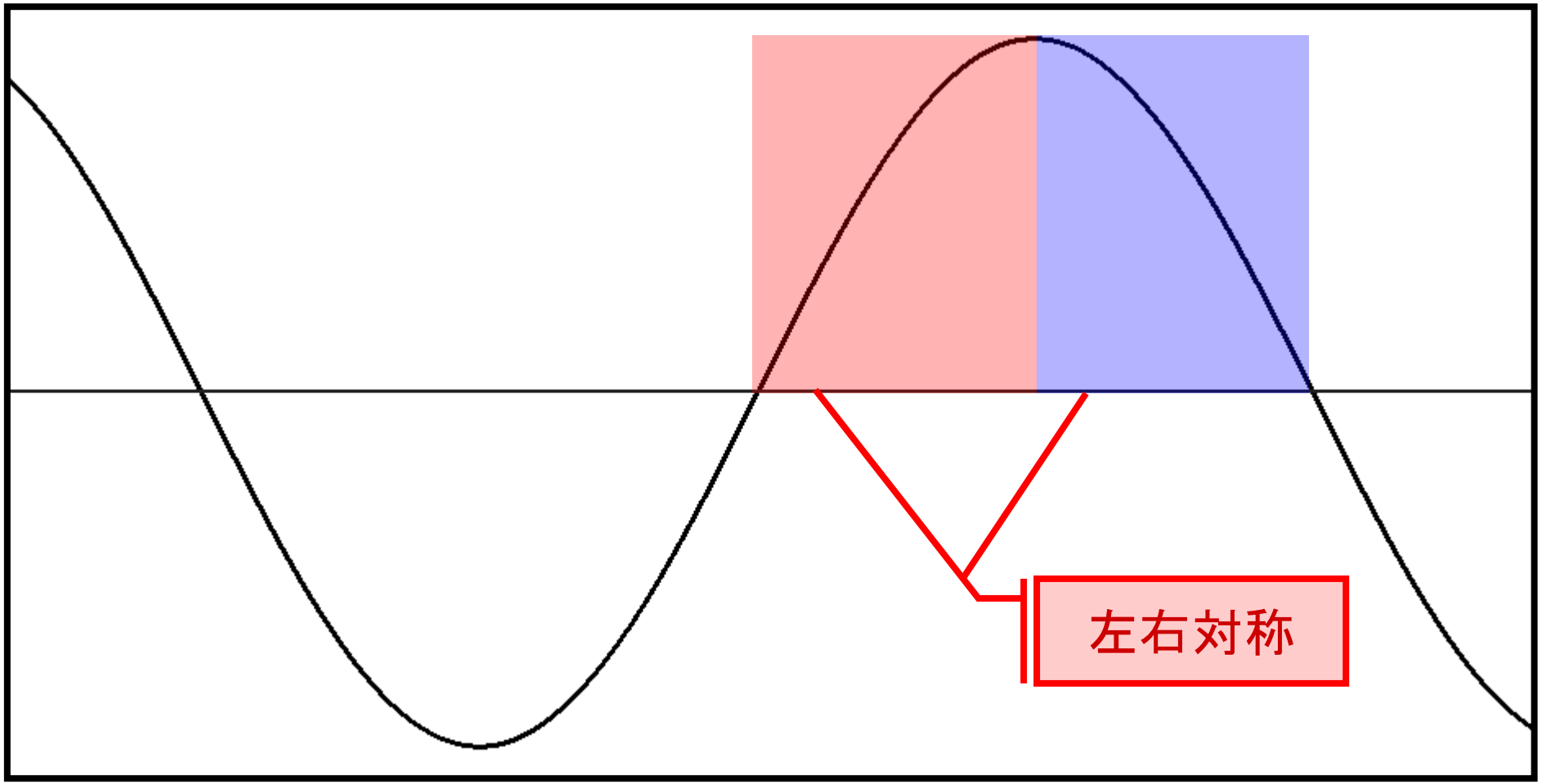
正弦波[特徴その4]

- 山と谷は全く同じ形・同じ大きさ



正弦波[特徴その5]

- 山(谷)は左右対称
  - 山(谷)を真ん中で折りたたむと、きれいに重なる



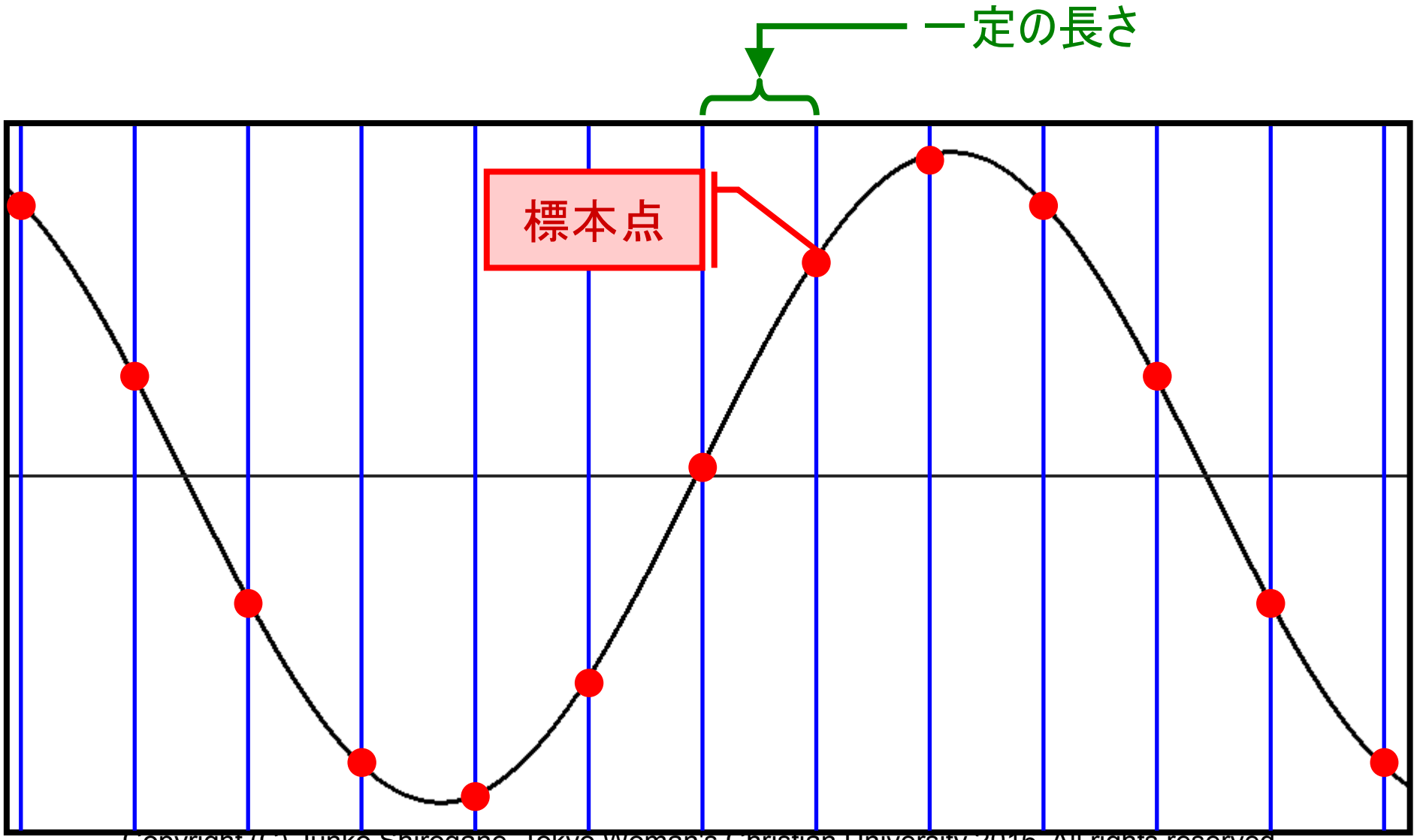
標本化[1](p. 20)

- アナログ信号: 様々な幅や高さの波(正弦波)の組み合わせ
  - 幅の狭い波・広い波、高い波・低い波
- 標本化: アナログ信号の横軸の一定の長さごとに、縦軸の値を調べること
  - 調べた点を「標本点」と呼ぶ
- 調べた以外のところの値は使用しない
  - 標本点の間隔を十分に細かくとれば、もとのアナログ信号の情報を完全に保持できる

どのくらいの間隔で標本化をすれば良いか???

標本化[2](p. 20)

- 標本化: 横軸を一定の長さで区切って縦軸の値を調べること



標本化[3](p. 20)

- 標本化の感覚をどうやって決めるか?
  - 1つの波の幅の半分より小さい間隔で標本点をとれば、元の波形を正確に表現可能(=方程式を算出可能)
  - 1つの波の幅の半分より大きい間隔で標本点をとれば、
    元の波形とは異なった波形が出現

エイリアシング

音や画像の情報: 波形の周期の長いもの・短いものを重ね合わせることで表現

波の中で最も狭い幅のものの半分よりも
狭い間隔で標本化することが重要

標本化定理[1](p. 21)

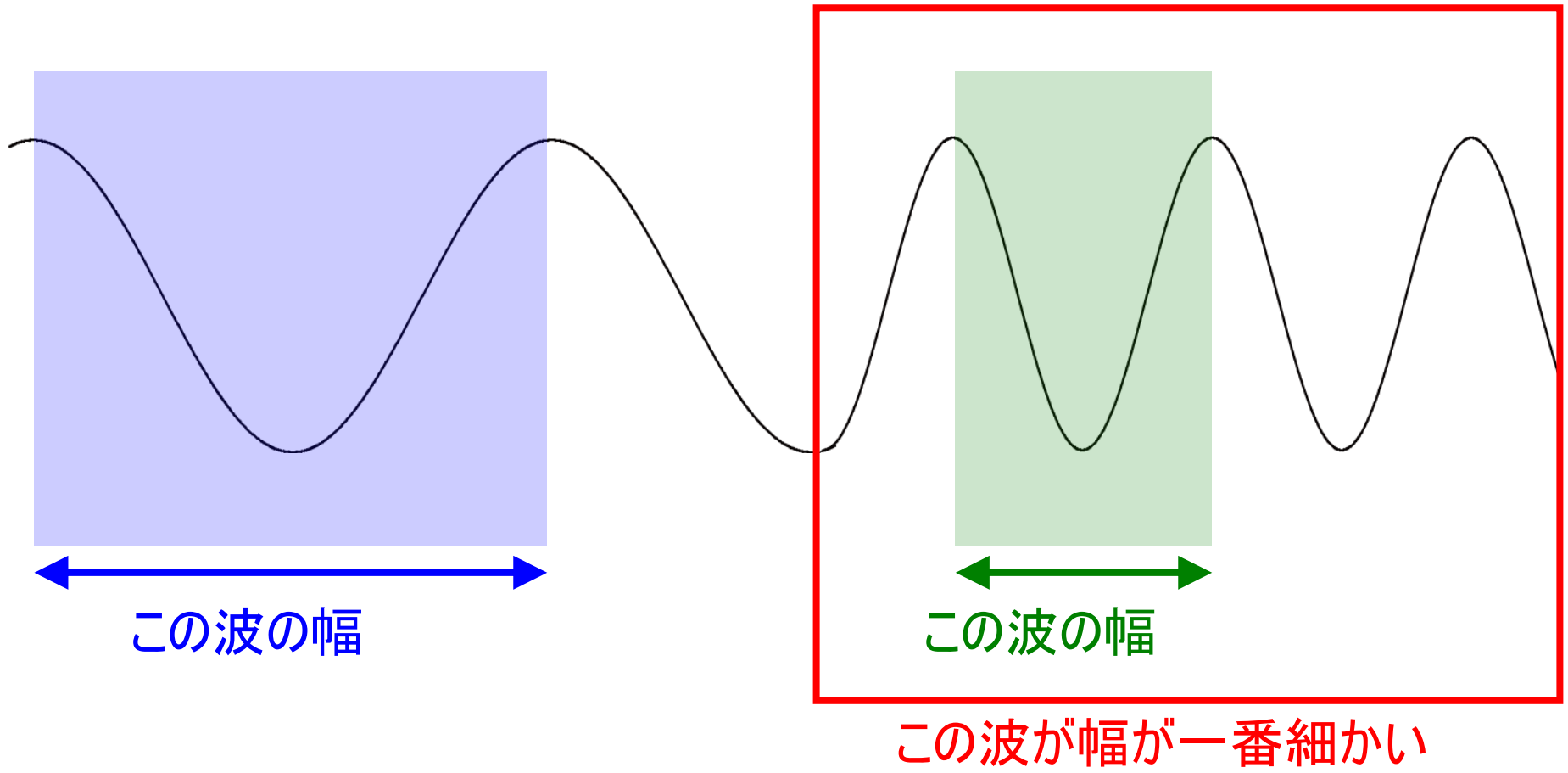
- 標本化定理
  - 最も狭い波の幅の半分よりも狭い幅で標本化すれば、標本から元の波を再現可能

標本化をするときに、どのくらいの間隔で標本点を取れば良いか？ を考えるときの目安になる定理

- 間隔が広すぎると、元の波が再現できず、正確に情報をコンピュータに取り込めない
- 間隔が狭すぎると、元の波は再現できるが、情報量が多すぎて、ファイルサイズなどが大きくなる

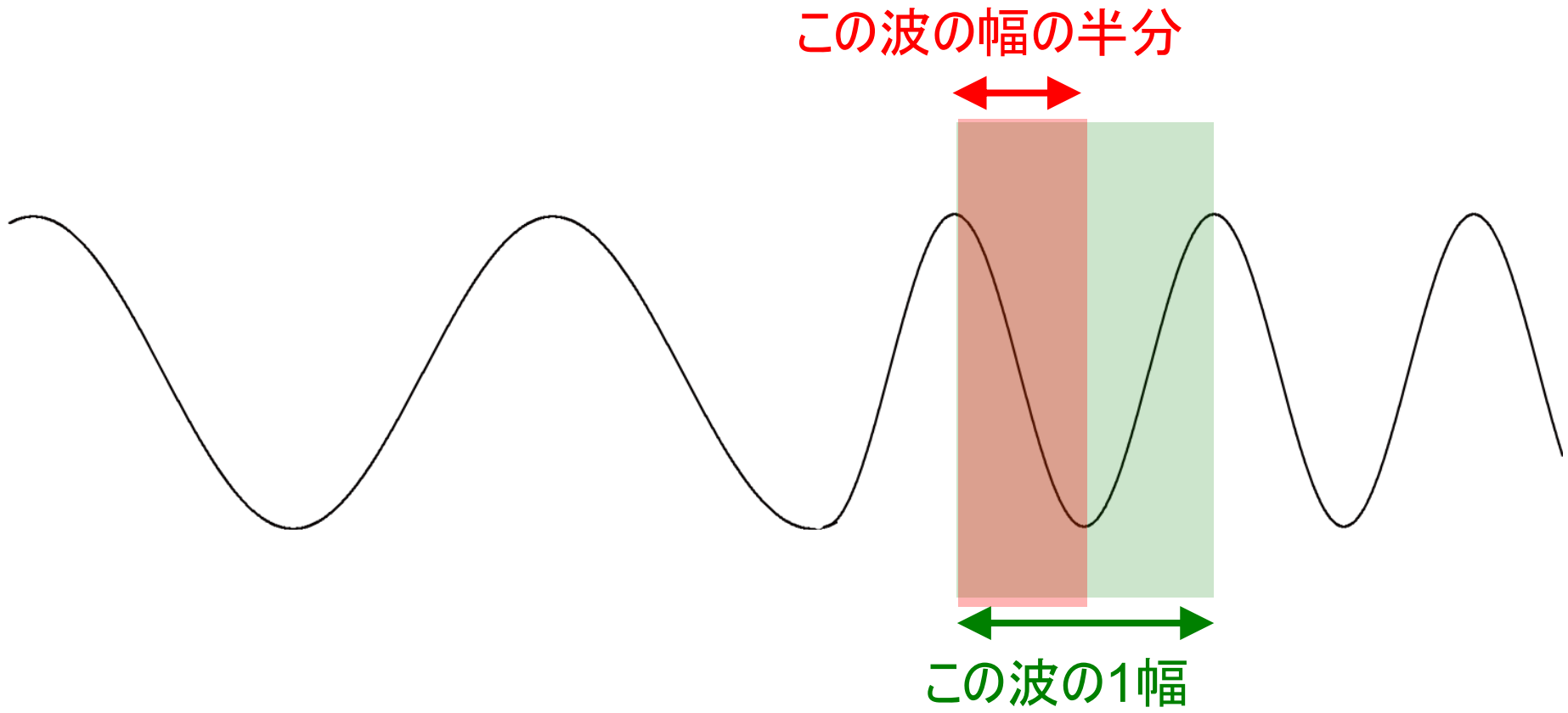
標本化定理[5](p. 21)

- アナログ信号の中のどの波が、一番幅が狭いか?



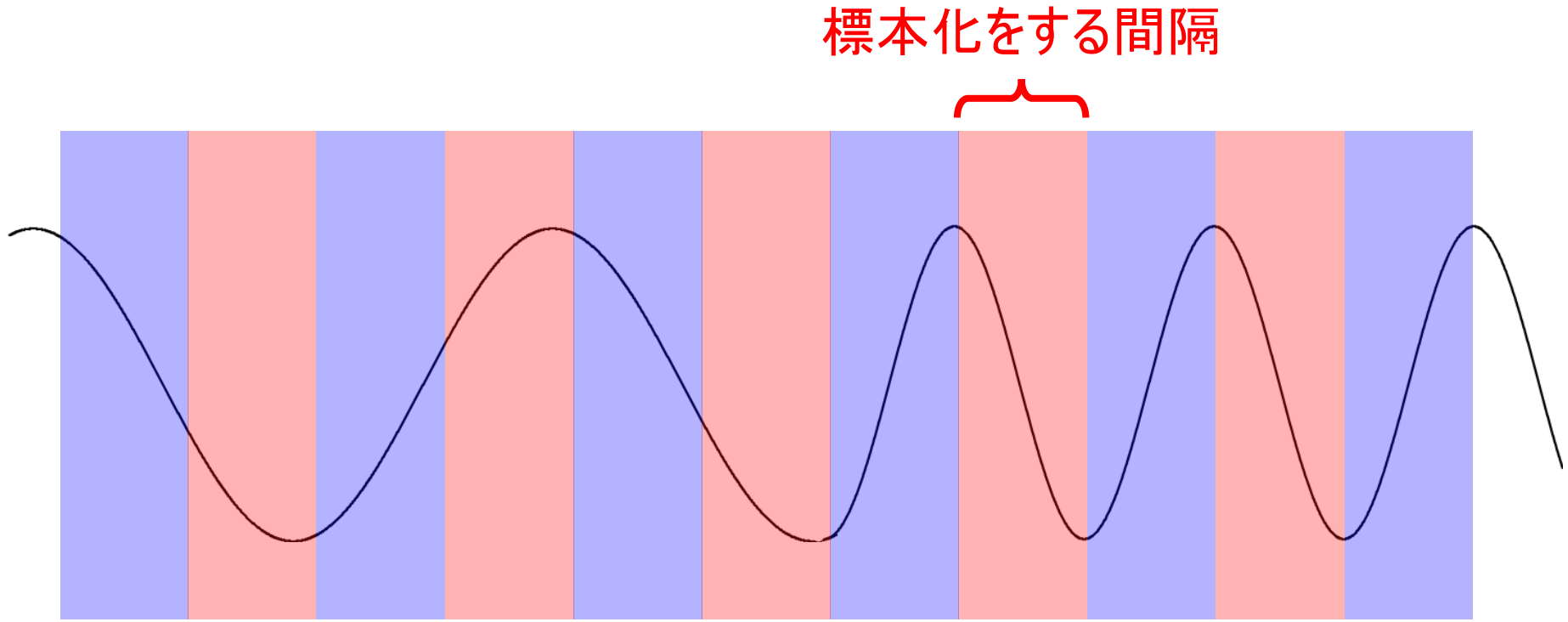
標本化定理[6](p. 21)

- 標本化をする間隔を決定
  1. アナログ信号の中で、最も幅の細かい波を見つける
  2. 1. の波の幅の半分より狭い間隔で標本化をする



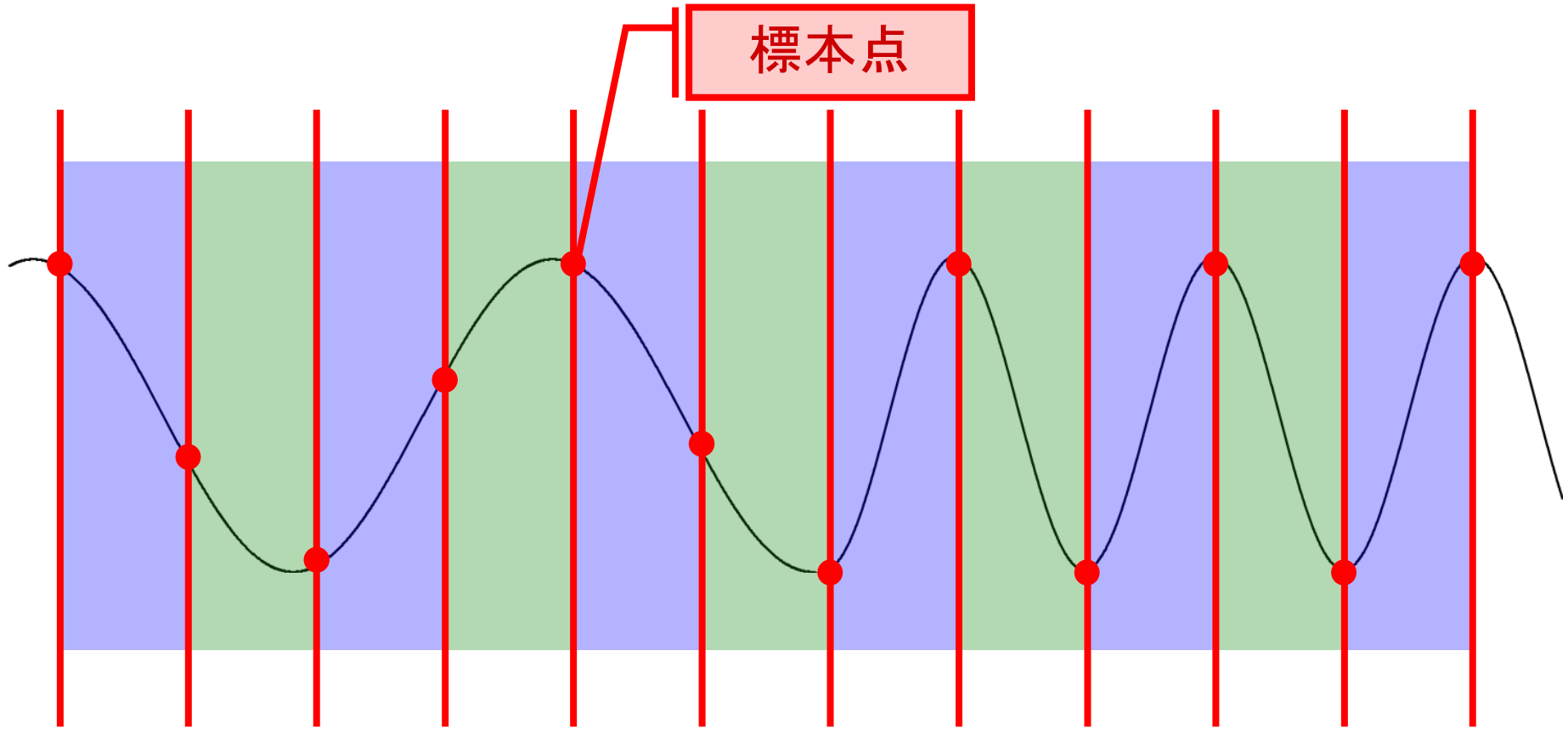
標本化定理[7](p. 21)

- 標本化をする間隔を決定
  1. アナログ信号の中で、最も幅の細かい波を見つける
  2. 1. の波の幅の半分より狭い間隔で標本化をする



標本化定理[8](p. 21)

- 標本化をする間隔を決定
  1. アナログ信号の中で、最も幅の細かい波を見つける
  2. 1. の波の幅の半分より狭い間隔で標本化をする



量子化[1](p. 22)

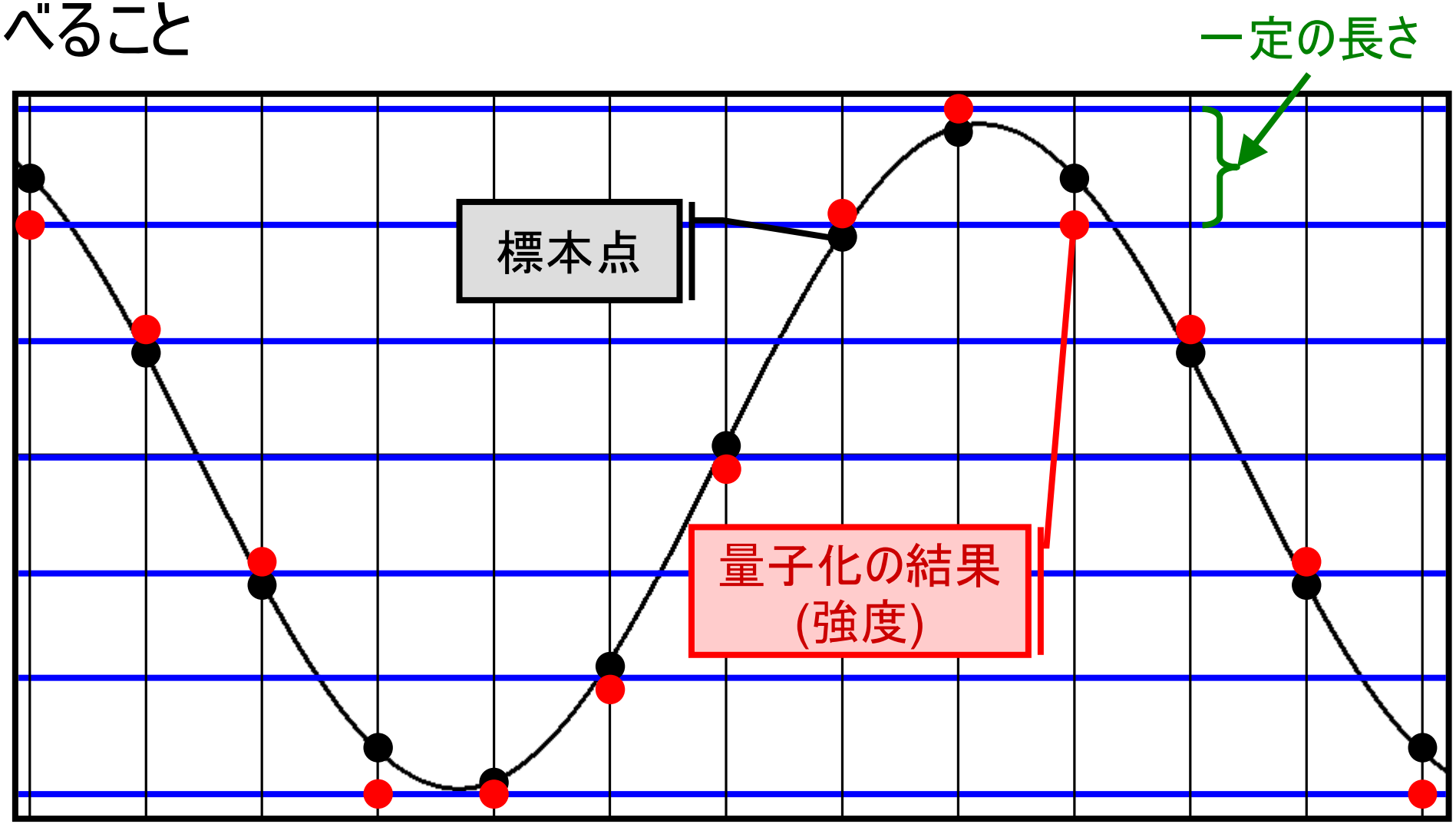
- 量子化: 縦軸の一定の長さごとに、横軸の値を調べること
  1. 横軸の値を調べるための縦軸の量(量子化レベル)をどの程度にするかを決める
  2. 標本化で取り出された標本点のy軸の値(強度)を最も近い量子レベルに置き換える

アナログ情報のデジタル化が完了

- 最終的に、量子化の結果(強度)の値をコンピュータに取り込む
  - もとのアナログの値とは異なる値が取り込まれる

量子化[2](p. 22)

- 量子化: 縦軸を一定の長さで区切り、各標本点と最も近い縦軸の値を調べること



音の符号化[原則](p. 23)

- 音: 空気などの分子の振動現象
- 波の横軸: 音の高さ
  - 人間の耳は、振動の中で20Hz～20kHzのものを聞き分け可能
  - Hz(ヘルツ): 1秒間の振動の回数, または1秒間の標本化の回数
    - 波の幅: 振動の間隔(1秒間の振動の回数)
  - 振動の回数が大きければ(波の幅が小さければ)高い音
  - 振動の回数が小さければ(波の幅が大きければ)低い音
- 波の縦軸: 音の強さ(大きさ)

人間の感覚で感知できる程度の波を扱えれば良い
(人間が聞き分けられない波は扱わなくても良い)

音の符号化[標本化](p. 23)

- 波の横軸: 音の高さ
- 20kHzの波形を再現するには、40kHzで標本化(1秒間に40000回標本化)
  - 20kHzの波: 正弦波の周期は1/20000
    - 標本化は、1/2周期で1回行う
      = 20kHzの波は1/40000間隔で標本化
      = 40kHzで標本化
  - 音楽では高い音も再現する必要: 音楽CDは44.1kHzで標本化
  - 固定電話では人間の声の高さ程度を再現: 8kHzで標本化

音の符号化[量子化](p. 23)

- 波の縦軸: 音の強さ(大きさ)
  - 固定電話: 8ビット(256種類)の量子化レベル
    - 0～255の256段階の数値で音の強さを表現
  - 音楽CD: 16ビット(65,536種類)の量子化レベル
    - 0～65,535段階の数値で音の強さを表現(音の強さをより細かく表現可能)

画像の符号化[1](p. 23)

- 標本化した画像: 点の集まりと考えられる
  - 点: 細かい正方形のマス目

    画素(ピクセル, pixel)
  - 画像の長方形のキャンバスは点の集まり
  - 1つ1つの点の大きさによって画像の質が決定
    - 点が大きければ粗い画像
    - 点が小さければきめの細かい画像

    ➡ 標本化の間隔により、点の大きさが決定
  - 1つ1つの点が何色かを記録しておくことで画像を表現

    ➡ 量子化の間隔により、画像中で利用可能な色の種類が決定
    (どの程度、微妙な色合いを表現するか)

画像の符号化[2](p. 23)

- カラー画像
  - コンピュータのディスプレイ: 赤(Red), 緑(Green), 青(Blue)の3つの光を利用
    - 3つの光にそれぞれ256段階の濃淡をつけ、3つの光を混ぜ合わせて色を作成
    - 256段階 = 8ビットで表現可能
    - 1つの色: 8ビット×3つの光 = 24ビットで表現

画像中の1つの点を24ビットで表現

カラー画像: 1つの画素を0～16,777,215までの数値で表現可能

Question!

データ圧縮と情報量

画像データの伝送[1](p. 25)

- 地上デジタル放送: 1440×1080の解像度でフレームレートが約30fps
  - fps(frame per second): 1秒あたりに切り替える画像の数

動画の世界では「フレーム」と呼ぶ

Ex. 1つの点を24ビット(3byte)で表現すると…
(1440×1080画素)×(3byte)×(30フレーム/秒)
=139968000byte/秒
≒140Mbyte/秒
(ただし、音声なし)

画像データの伝送[2](p. 25)

- デジタル放送: 140Mバイト/秒の伝送が必要
  - テレビ局から家庭に映像を送るときに、1秒間に140Mバイト送れないと、なめらかな映像が見られない
  - Ex. 家庭のコンピュータのネットワーク環境
    - ADSL: 50Mビット/秒(bps) = 6.25Mバイト/秒
    - 光: 100Mビット/秒(bps) = 12.5Mバイト/秒

理論上の速さで、実際はもっと遅い

140Mバイト/秒は莫大な伝送量!!

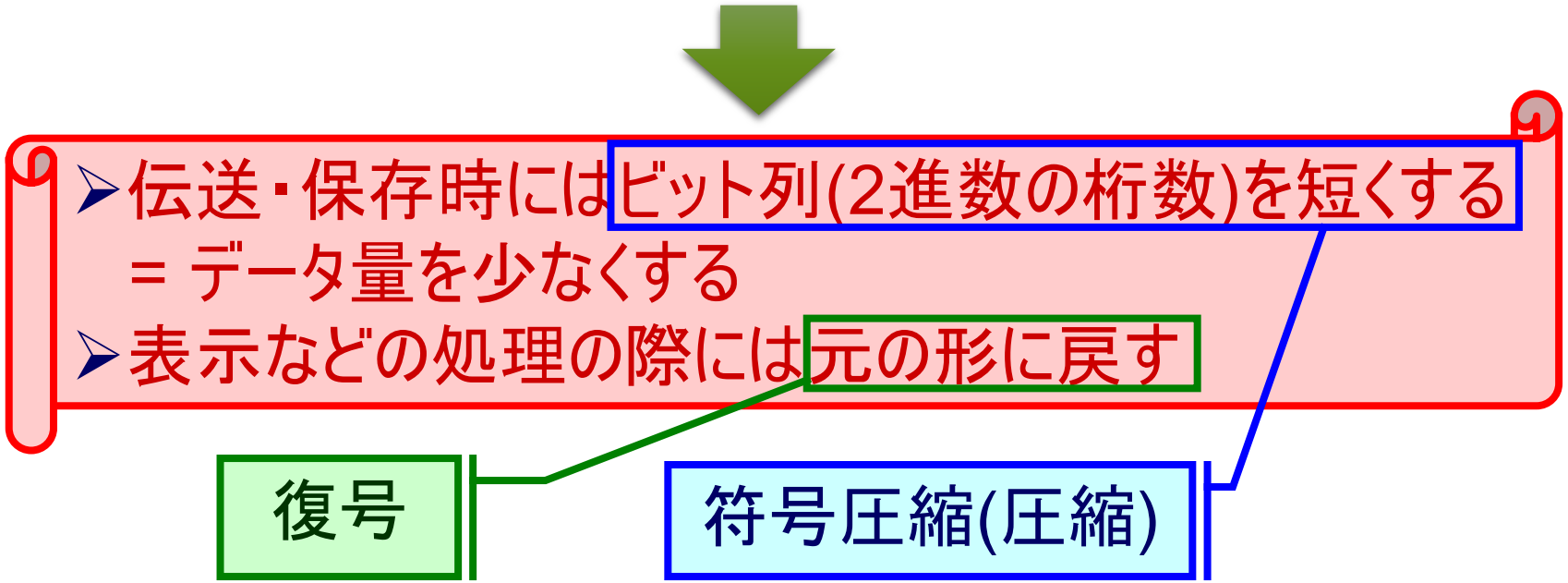
画像のデジタル表現[4](p. 25)

- 140Mバイト/秒は現実的に伝送不可能
  →データを圧縮して伝送
  - 圧縮: 決まった手順に従ってデータのサイズを小さくすること
- 静止画
  - 画像の中の特定の領域の色の変化は少ない
- 動画
  - 動画の中で実際に動く部分は大きくない
  - 動く部分を過去の様子から大体予測できる

デジタル放送では、この性質を使って圧縮
(圧縮技術: MPEG-2)

圧縮と復号(p. 26)

- 数値・文字・画像・音声: 符号化(2進数に変換)して処理
- 特に画像・音声はデータ量が多い
  - 伝送に時間が多く必要
  - 保存場所の容量が多く必要



可逆圧縮(p. 26)

- 可逆圧縮
  - 復号時に1ビットの違いもなく元のビット列が復元される圧縮法
    - 圧縮したファイルから、圧縮前のファイルを取り出すことができる
  - 非可逆圧縮に比べて、ビット列をあまり短くすることはできない
    - 非可逆圧縮よりもデータ量の減量分は少ない

非可逆圧縮(p. 26)

- 非可逆圧縮
  - 復号時に元のビット列とは多少の違いが生じてしまう圧縮法
    - 圧縮したファイルから、圧縮前のファイルを取り出すことができない
  - 可逆圧縮に比べて、ビット列がかなり短くなる
    - 可逆圧縮よりもデータ量の減量分が多い

数字・文字情報(p. 26)

- 復号したときに内容が変わってはならない
  - 文書中の1文字が抜け落ちたら…?
  - 数値の小数点の桁数が少なくなったら…?
- 一般的に(画像・音声に比べて)データ量は少ない
  - 圧縮によるデータの減量分はそれほど(画像・音声ほど)多くなくて良い

可逆圧縮が使われる

画像・音声情報(p. 26)

- 復号したときに内容が多少変わっても良い
  - 画像: 表示したときに見た目が変わらなければ良い
  - 音声: 聞いたときに元と同じように聞こえれば良い
- 一般的に(数値・文字に比べて)データ量が多い
  - 圧縮によるデータの減量分が(数値・文字に比べて)多いことが必要

非可逆圧縮が使われる

- 動画・音声のデータ量は膨大なので非可逆圧縮
- 静止画のデータ量はそれほど多くないので可逆圧縮・非可逆圧縮のどちらも使われる

非可逆圧縮の圧縮法

- 余分な情報を取り除くことで圧縮
  - 画像の場合、同じ色が集まっているところの色の情報など
  - 音声の場合、人間の耳には聞こえないような音

人間の感覚ではわからないものを削るので、見た目・聞いた感じでは品質は変わらない

- 画像は拡大すると、画質が落ちているのがわかる
- 音声は、良いスピーカーを使うと音質が落ちているのがわかる
- 圧縮率を上げる(ファイルサイズをより小さくする代わりに、取り除くものを多くする)と、質が落ちているのがわかる

可逆圧縮の圧縮法(p. 28)

- ハフマン符号化
- ランレングス符号化
- etc.

ハフマン符号化[1](p. 28)

- ハフマン符号化: データ内に出現する情報を統計的に処理し、ビット列の長さを変えて情報を表現

  これまで: どの情報も同じ長さのビット列で表現

  - 出現率の高い情報を短いビット列で表現
  - 出現率の低い情報を長いビット列で表現

ハフマン符号化[2](p. 28)

Ex. ある地方の320日の天気

天気	天気である日数	ビット列での表現
雨が降っていない	160日(2日に1日)	0
小雨が降っている	40日(8日に1日)	100
適度な雨が降っている	20日(16日に1日)	1010
やや強い雨が降っている	20日(16日に1日)	1011
非常に強い雨が降っている	20日(16日に1日)	1100
強い雨が降っている	20日(16日に1日)	1101
弱い雪が降っている	20日(16日に1日)	1110
大雪が降っている	20日(16日に1日)	1111

「10100100…」は、何日分のどんな天気?

ハフマン符号化[3](p. 28)

- 「10100100…」
  - 最初のビットが「1」なので、「雨が降っていない」という天気ではない
  - 2ビット目が「0」なので、「非常に強い雨」、「強い雨」、「弱い雪」、「大雪」という天気ではない
  - 3ビット目が「1」なので、「小雨」という天気ではない
  - 4ビット目が「0」なので、「やや強い雨」という天気ではない

1日目の天気は「適度な雨が降っている」

それぞれの情報を同じ長さのビット列にしなくても情報の表現は可能

※情報の種類ごとに割り当てるビット数を変える方式: 可変長符号

ハフマン符号化[4](p. 29)

- 8種類の天気を普通のやり方で現すと…？
  - 1種類を3ビットで表現: 320日では、320×3=960ビット

ハフマン符号化では…

雨が降っていない	160日	0	1ビット×160日
小雨が降っている	40日	100	3ビット×40日
適度な雨が降っている	20日	1010	4ビット×20日
やや強い雨が降っている	20日	1011	4ビット×20日
非常に強い雨が降っている	20日	1100	4ビット×20日
強い雨が降っている	20日	1101	4ビット×20日
弱い雪が降っている	20日	1110	4ビット×20日
大雪が降っている	20日	1111	4ビット×20日

合計: 760ビット

200ビット少なくなっている

※どれだけ少なくなるかは扱う情報によって違う

ランレングス符号化[1](p. 29)

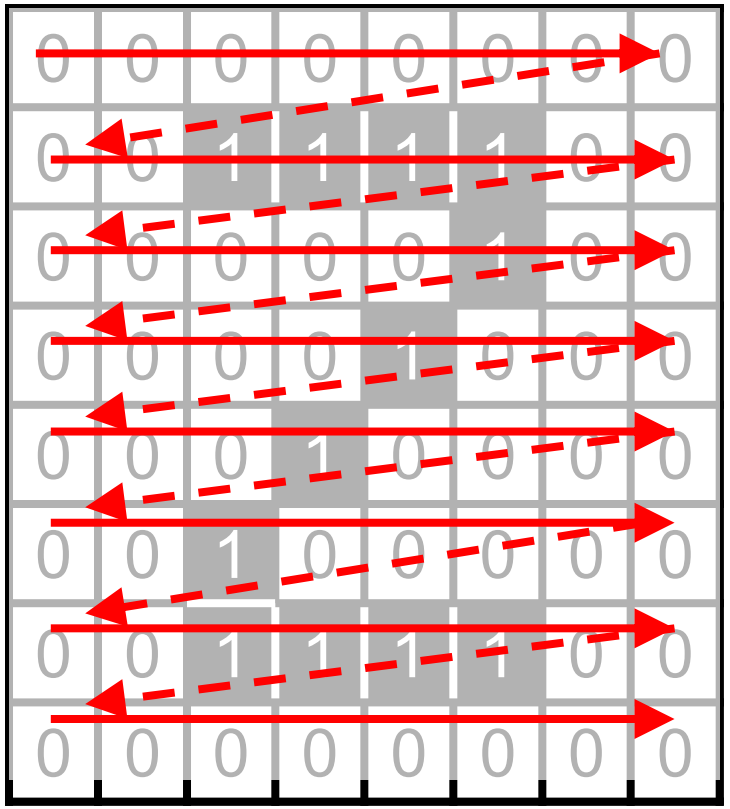
- ランレングス符号化: 白黒2値画像(灰色のない、白と黒のみの画像)の圧縮方法の1つ
  - 主にFAXで使われている

0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0

8×8=64ビット
(2値画像は1つ1つの点を1ビットで表現するため)

ランレングス符号化[2](p. 29)

- 2値画像: 白画素と黒画素はある程度固まっている

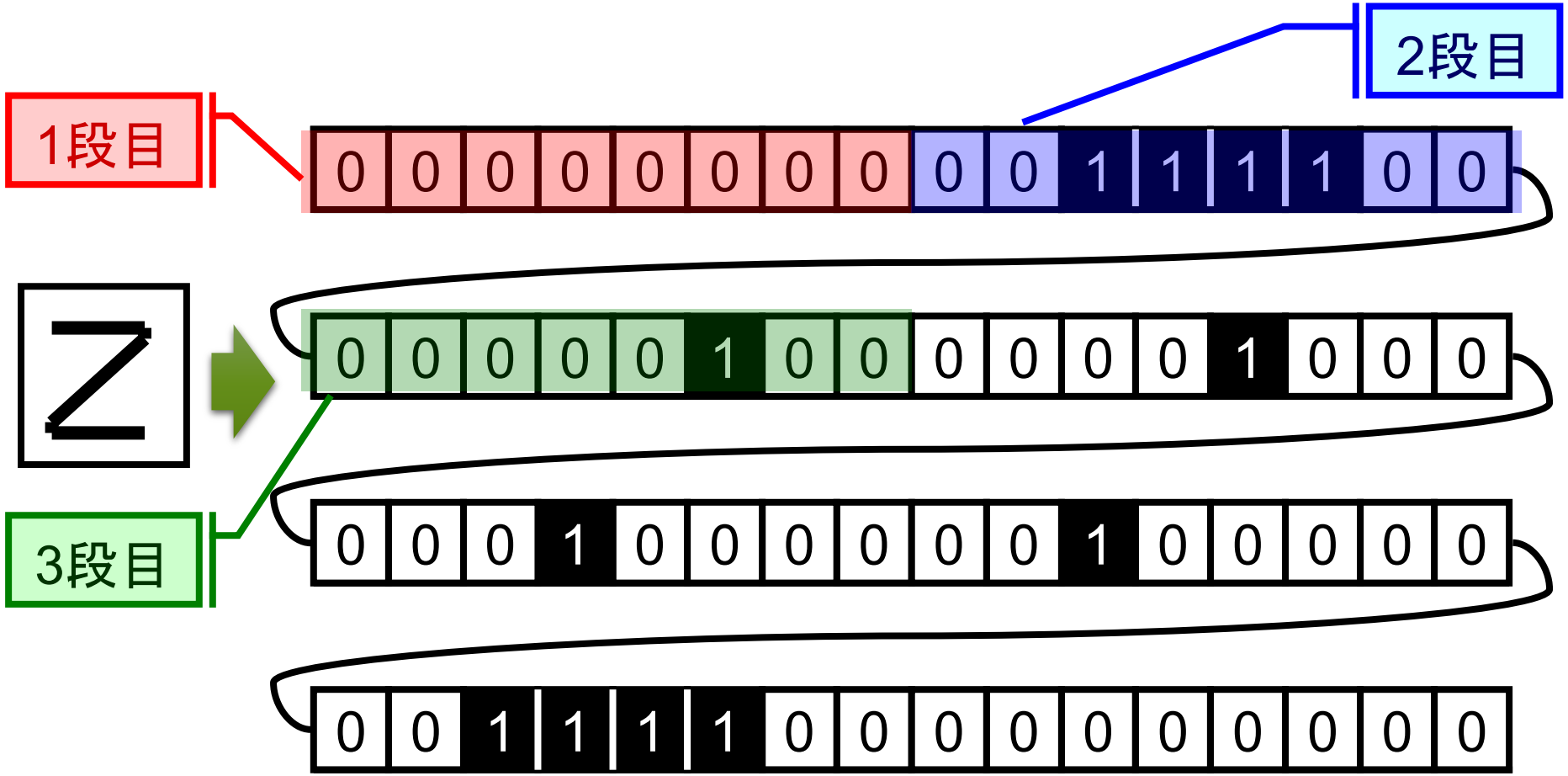


白画素が10個
黒画素が4個
白画素が7個
……
黒画素が4個
白画素が10個

矢印の方向に画素を見ていったとき

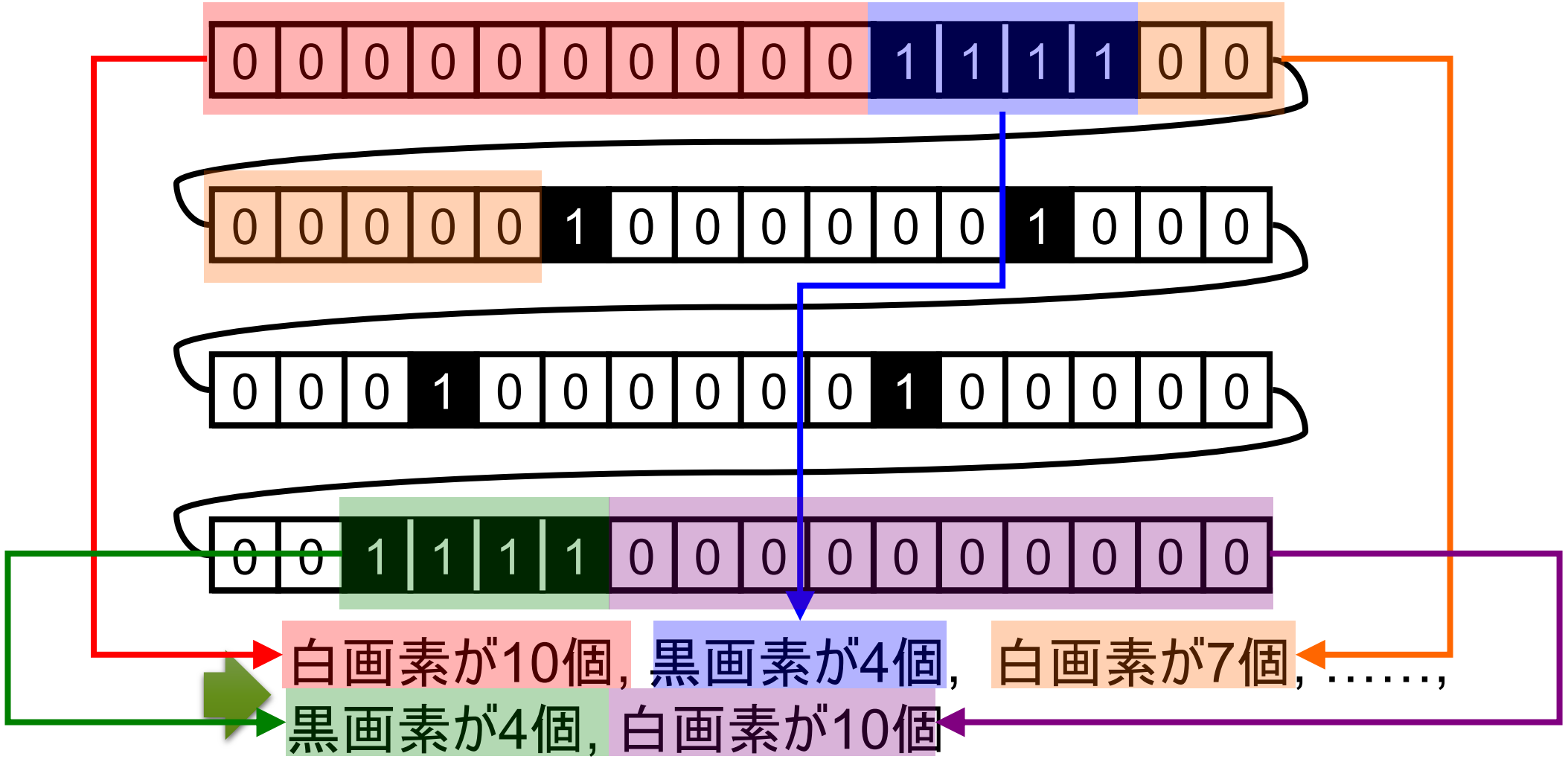
ランレングス符号化[圧縮][1]

- 画像の中の全ての画素を横一列に並べる
  - 横1段目の右に2段目、2段目の右に3段目…とつなげていく



ランレングス符号化[圧縮][2]

- 横一列に並べた画素について、左から順に、連続している白画素と黒画素の数を数えていく



ランレングス符号化[圧縮][3]

白画素が10個　黒画素が4個　白画素が7個　……
黒画素が4個　白画素が10個

画素の個数を2進数で表すと…

1010　0100　0111　0001　0110　0001　0110
0001　0110　0001　0111　0100　1010

52ビット

※個数の中で「1010」(2進数)が最も大きな個数なので、他の個数も、
2進数で表現したときの桁数を「1010」(4桁)にあわせる

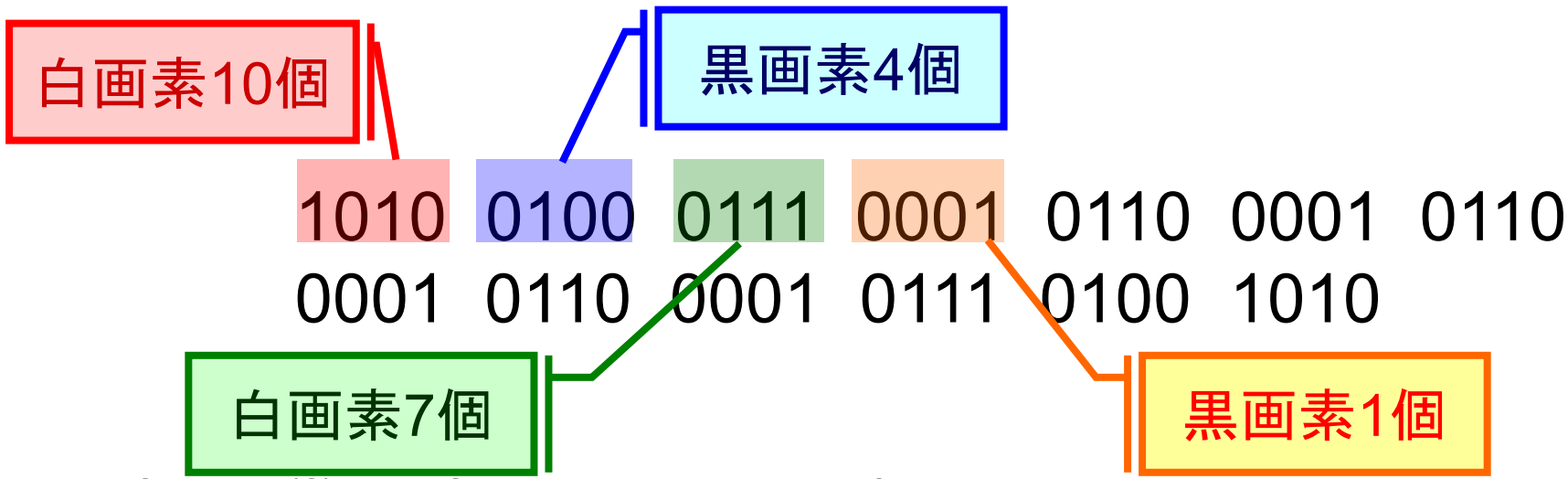
通常の方法で表す(1画素を1ビットで表す)と…
8×8=64ビット

12ビット少なくなっている

※どれだけ少なくなるかは扱う画像の内容によって違う

ランレングス符号化[復号][1]

- 圧縮したものを復号化するには…?
  - 画像の画素の数だけを表したもの(色の情報はない)
  - 白画素の並びと黒画素の並びは必ず交互になる
  - 画像の最も左上隅の画素は白であることが多い
    - 先頭の個数は、白画素の個数として扱う
    - 最も左隅の画素が黒の場合、圧縮時に、最初に出現する白画素の個数を0個としておく

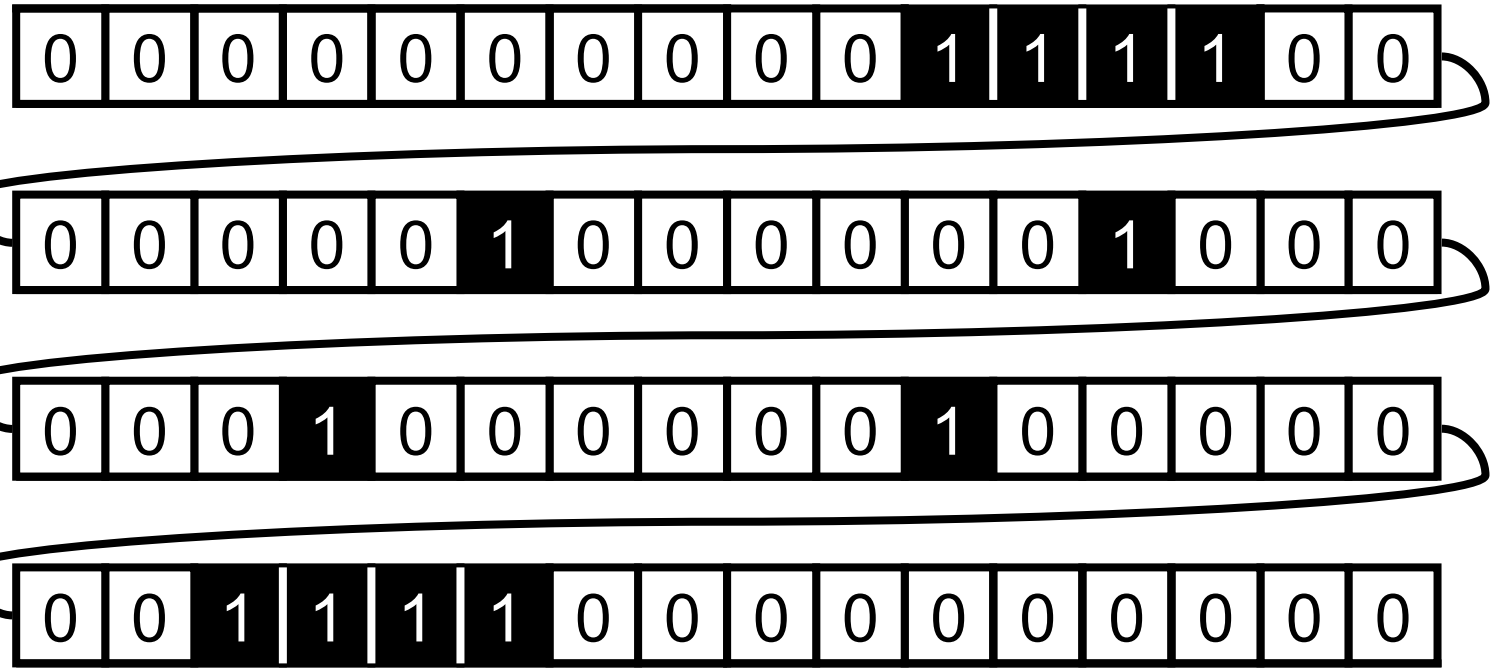


ランレングス符号化[復号][2]

- 画素の個数から、横一列の画素の並びが復元

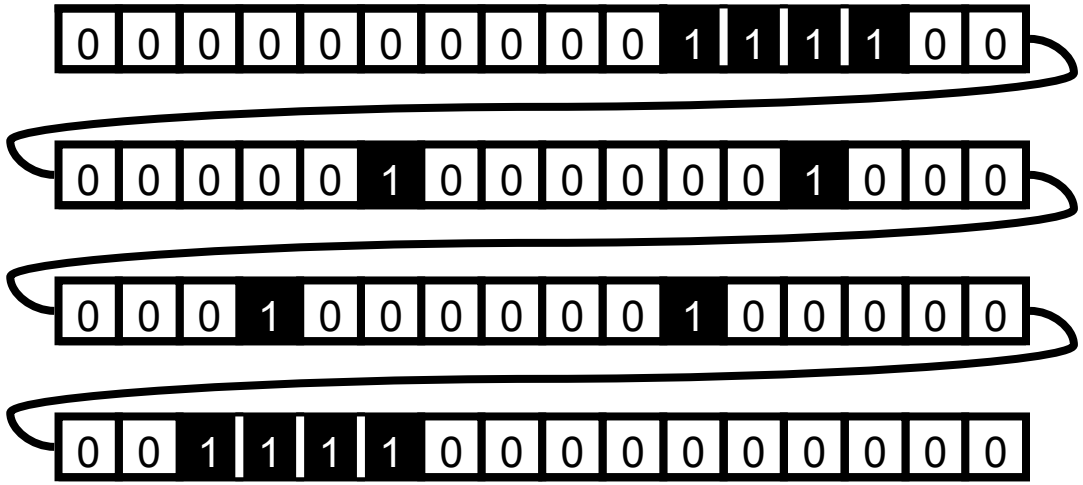
1010 0100 0111 0001 0110 0001 0110
0001 0110 0001 0111 0100 1010

白画素が10個, 黒画素が4個, 白画素が7個, ……,
黒画素が4個, 白画素が10個

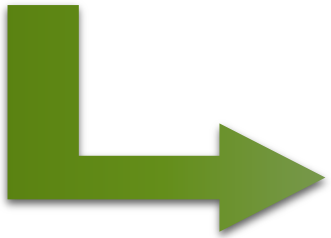


ランレングス符号化[復号][3]

- もとの画像の縦横の画素の数は記録されてある
  - 横一列の画素の並びが復元できると、もとの画像も復元できる



もとの画像は横8個、縦8個の画素



0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0

先頭が黒画素のとき[1]

- 前のスライドの画像の、白と黒を逆にした画像を考えると…

1	1	1	1	1	1	1	1
1	1	0	0	0	0	1	1
1	1	1	1	1	0	1	1
1	1	1	1	0	1	1	1
1	1	1	0	1	1	1	1
1	1	0	1	1	1	1	1
1	1	0	0	0	0	1	1
1	1	1	1	1	1	1	1

白画素・黒画素の個数を数えると…(前のスライドの画像の白と黒を逆にした画像なので…)

黒画素10個

白画素4個

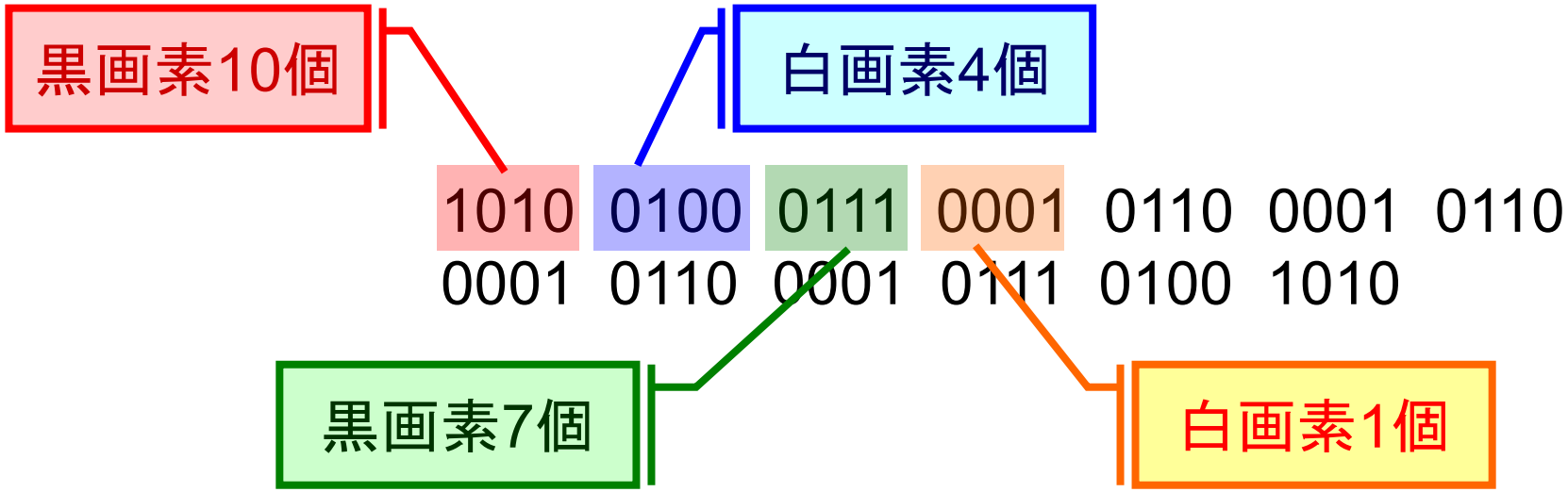
1010 0100 0111 0001 0110 0001 0110
0001 0110 0001 0111 0100 1010

黒画素7個

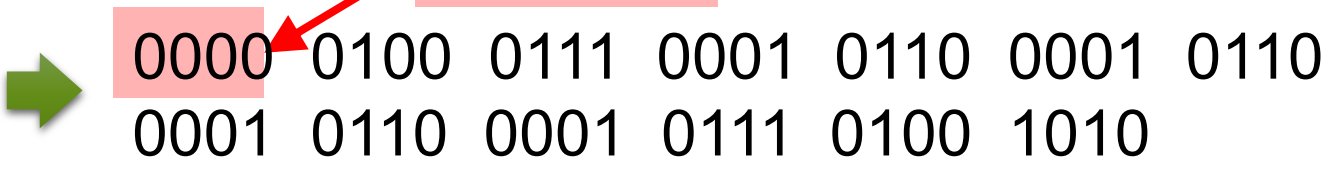
白画素1個

先頭が黒画素のとき[2]

- ランレングス符号化の考え方
  - 白画素・黒画素の個数だけを並べたとき、先頭の個数は白画素の個数とみなす



先頭の個数は黒画素の個数! …でもこれでは困る
→先頭の個数を「白画素0個」にすると良い



次々回(7月5日)

- 実習をするので24102教室で授業
  - スキャナで取り込んだ写真の標本化・量子化のレベルの違いの確認を実習

- 次回までに写真を1枚コンピュータに取り込んでおくこと
  - やり方はプリント&授業の資料のページで