

画像情報特論 (11)

最近の動向 & プログラム

情報理工学専攻 甲藤二郎

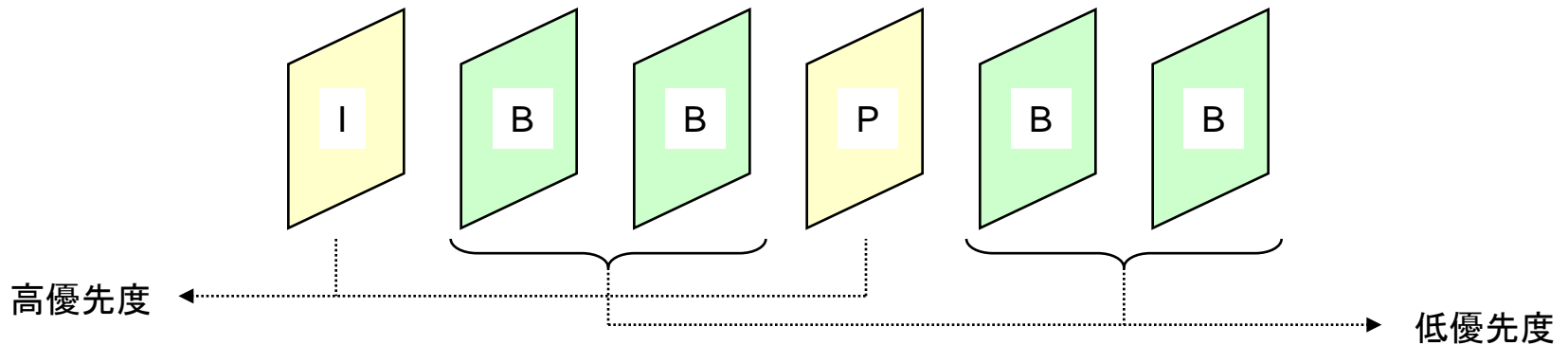
E-Mail: katto@waseda.jp

最近の動向

MDC (1)

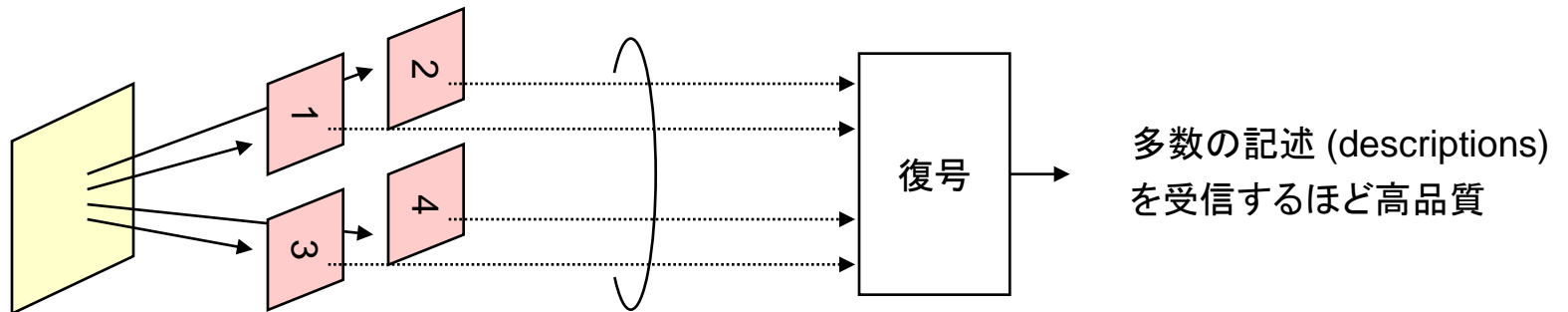
- スケーラブル符号化 (従来):

例: temporal scalability



- Multiple Description Coding:

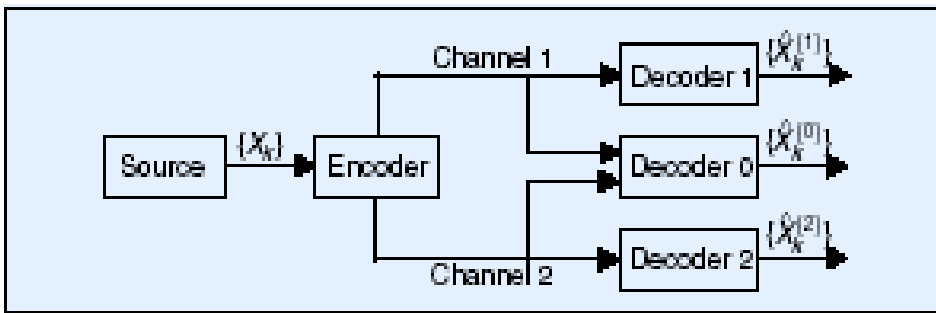
例: 空間サンプリング、複数コーデック、...



同一優先度 + マルチパス伝送

MDC (2)

• 理論



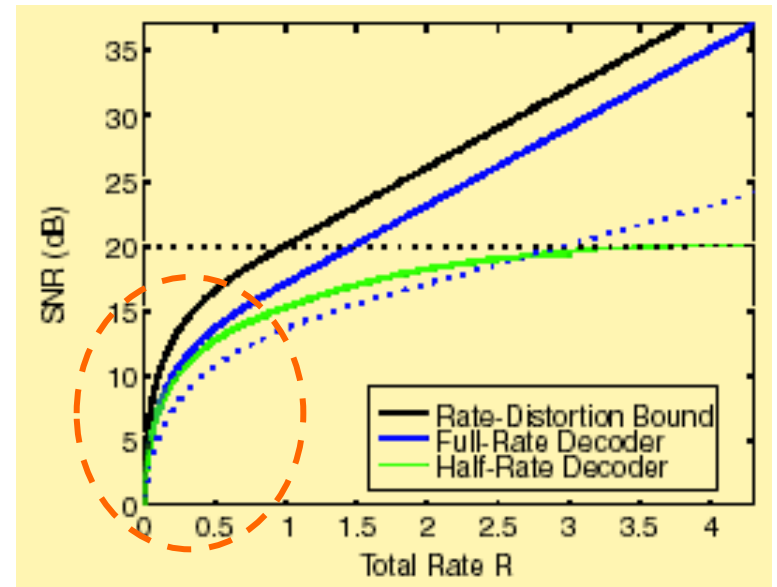
2チャンネル・3受信者のMDCモデル



For memoryless Gaussian source,

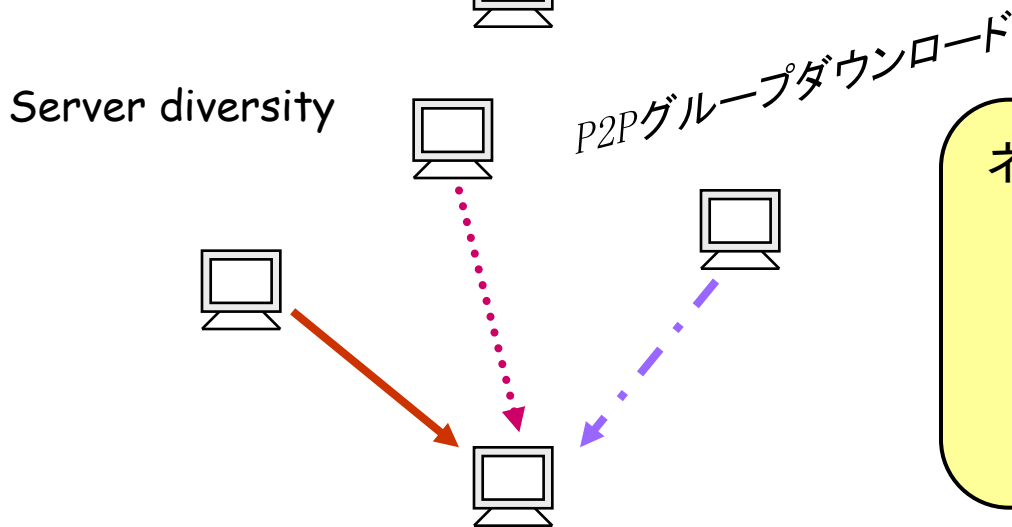
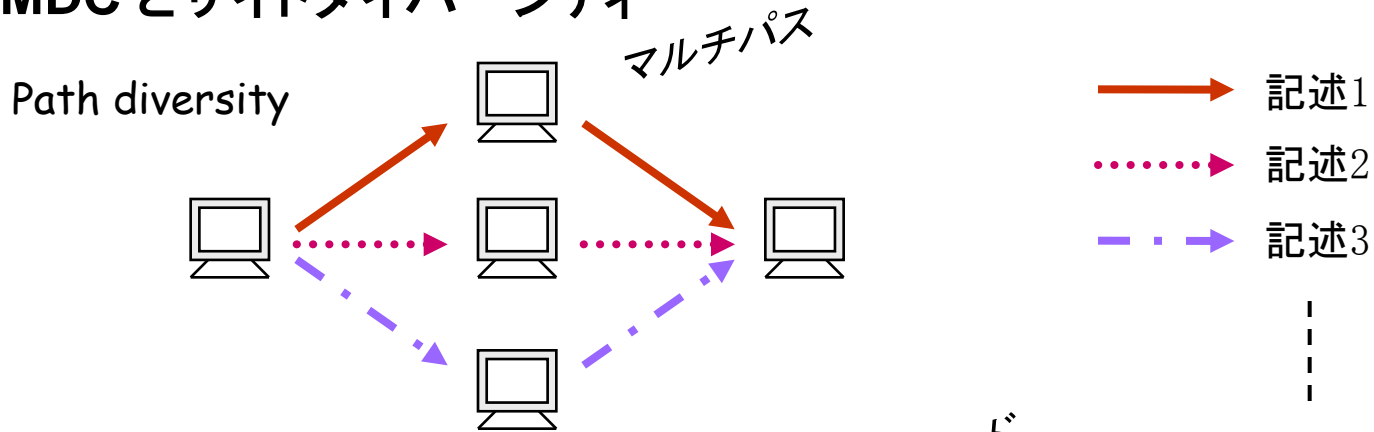
$$\begin{cases} D_i \geq \sigma^2 2^{-2R_i}, & \text{for } i=1,2 \\ D_0 \geq \sigma^2 2^{-2(R_1+R_2)} \cdot \gamma_D(R_1, R_2, D_1, D_2) \end{cases} \Rightarrow \gamma_D(R_1, R_2, D_1, D_2) = 1 \text{ if } D_1 + D_2 \geq \sigma^2 + D_0$$

D_1, D_2 のいずれか、もしくは両方が大きいときに成立



MDC (3)

• MDC とサイトダイバーシティ

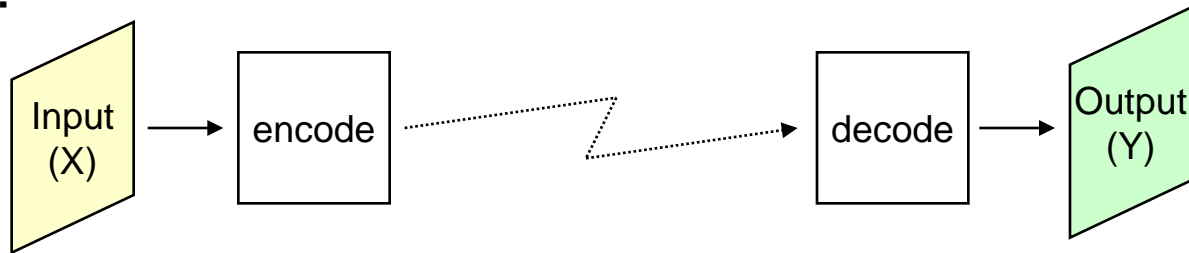


ネットワーク的な利点:

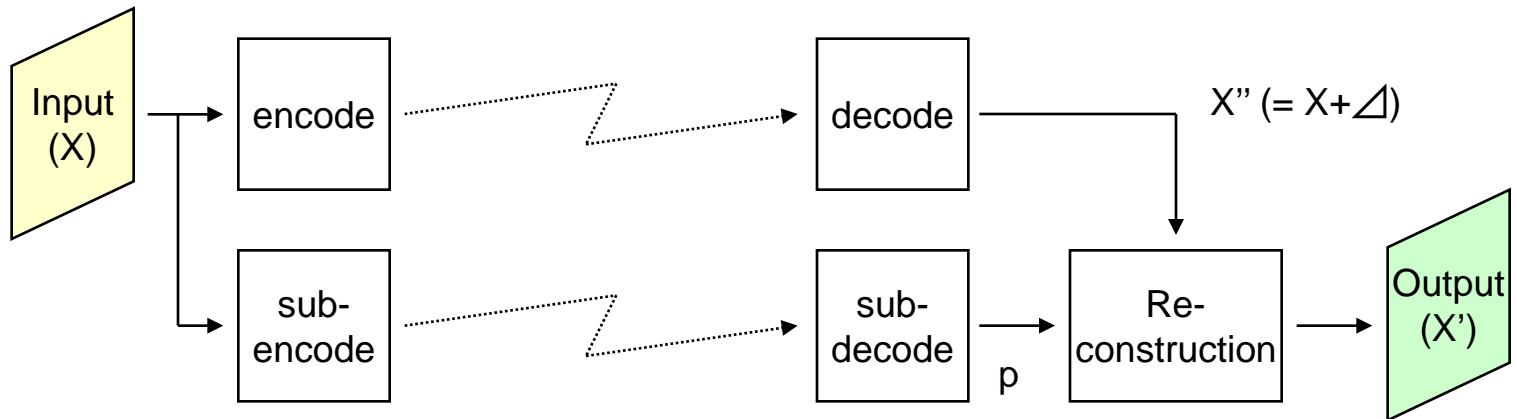
階層符号化のような階層間の依存性がない。
⇒ 使いやすい。各受信ノードが自由に取捨選択できる。

Distributed Source Coding (1)

- 従来:



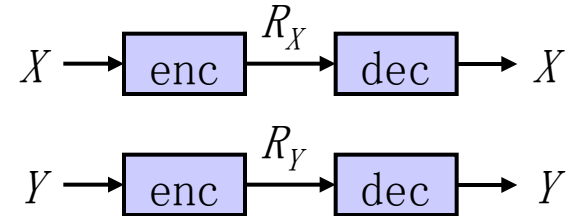
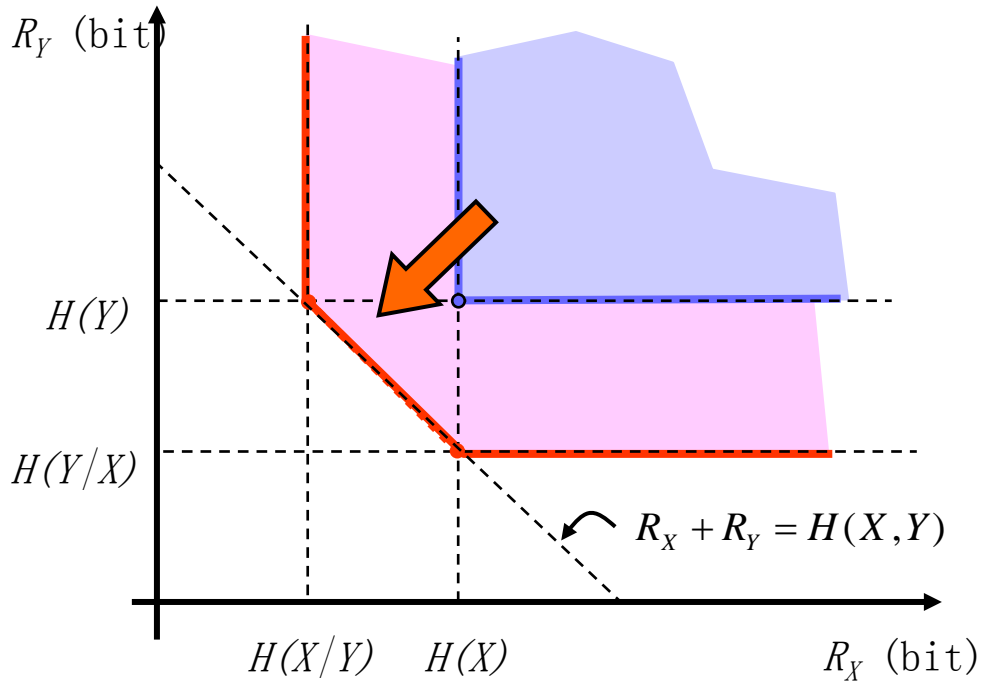
- Distributed Source Coding (Wyner-Ziv Coding):



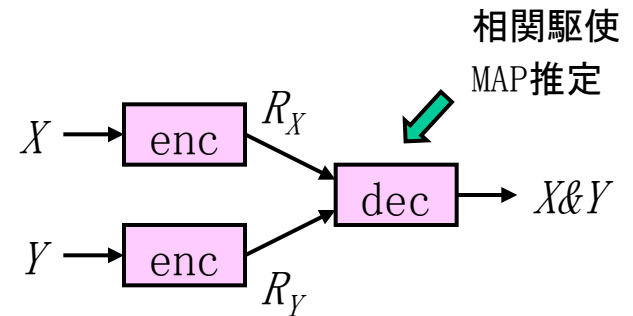
MAP推定: $X' = \arg \max P(X | X'', p)$

Distributed Source Coding (2)

- Slepian-Wolf の定理 (lossless)



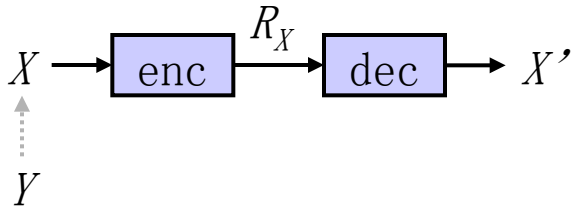
$$R_X \geq H(X), R_Y \geq H(Y)$$



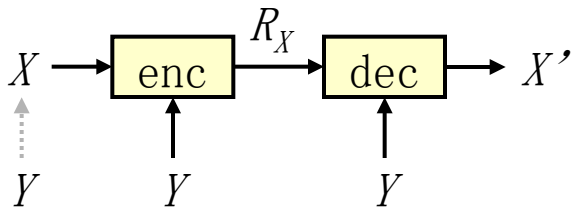
$$R_X + R_Y \geq H(X, Y)$$

Distributed Source Coding (3)

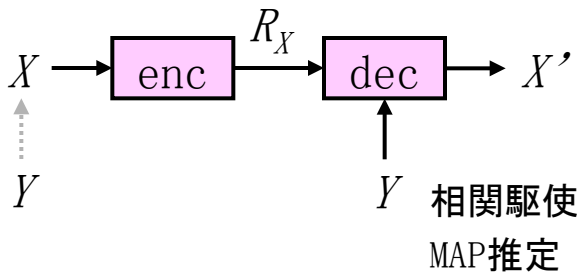
- Wyner-Ziv の定理 (lossy)



$$R_X \geq R(D_X) (\geq R_{X|Y}(D_X))$$



$$R_X \geq R_{X|Y}(D_X)$$

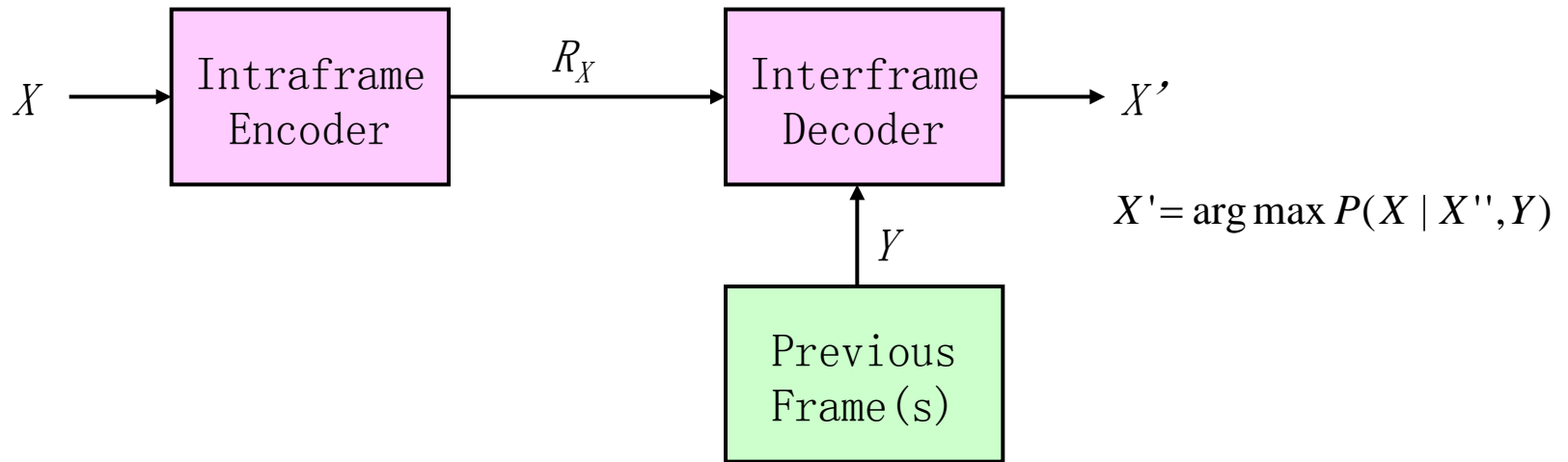


$$R_X \geq R_{X|Y}^{\text{WZ}}(D_X) \geq R_{X|Y}(D_X)$$

$$R_{X|Y}(D_X) + \Delta$$

Distributed Source Coding (4)

- Intraframe Encoder & Interframe Decoder



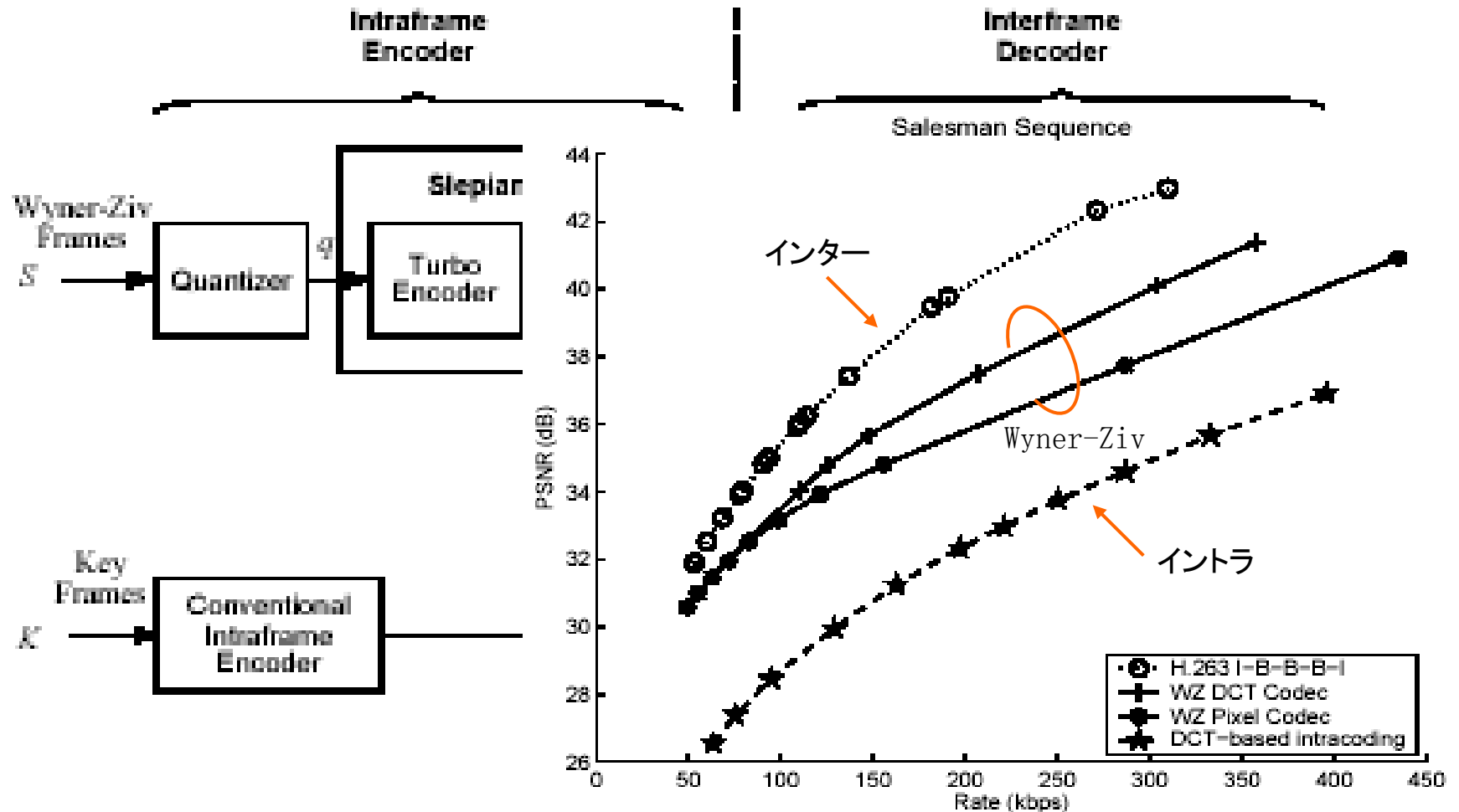
現状: Encoder の演算量は Decoder よりもはるかに大きい。

発想: Encoder をシンプルにして Decoder に演算量をシフトする。

特性: 現在のイントラ符号化よりは特性をよくできる。

Distributed Source Coding (5)

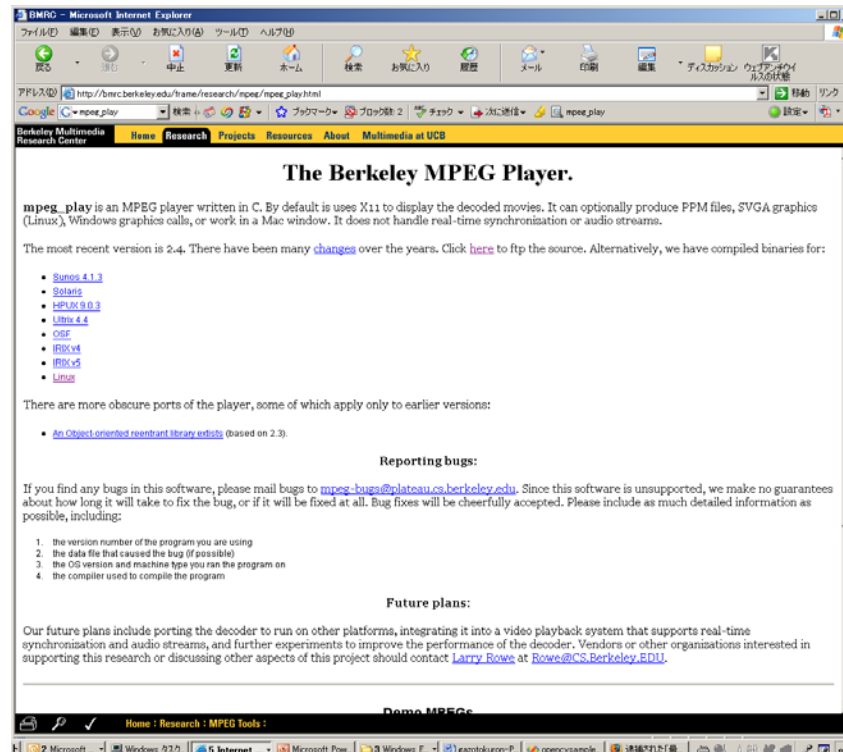
【Prof. Girod 氏資料参照】



プログラミング

mpeg_play

- (おそらく) 世界最初のソフトウェアデコーダ

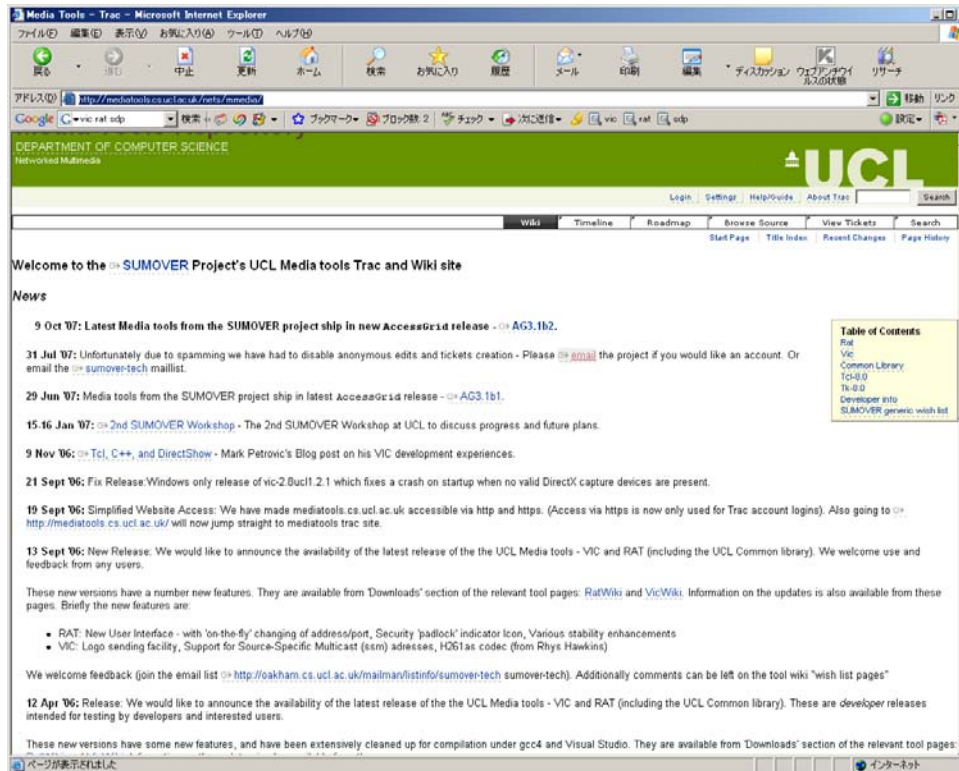


UCB

http://bmrc.berkeley.edu/frame/research/mpeg/mpeg_play.html

vic/rat/sdr

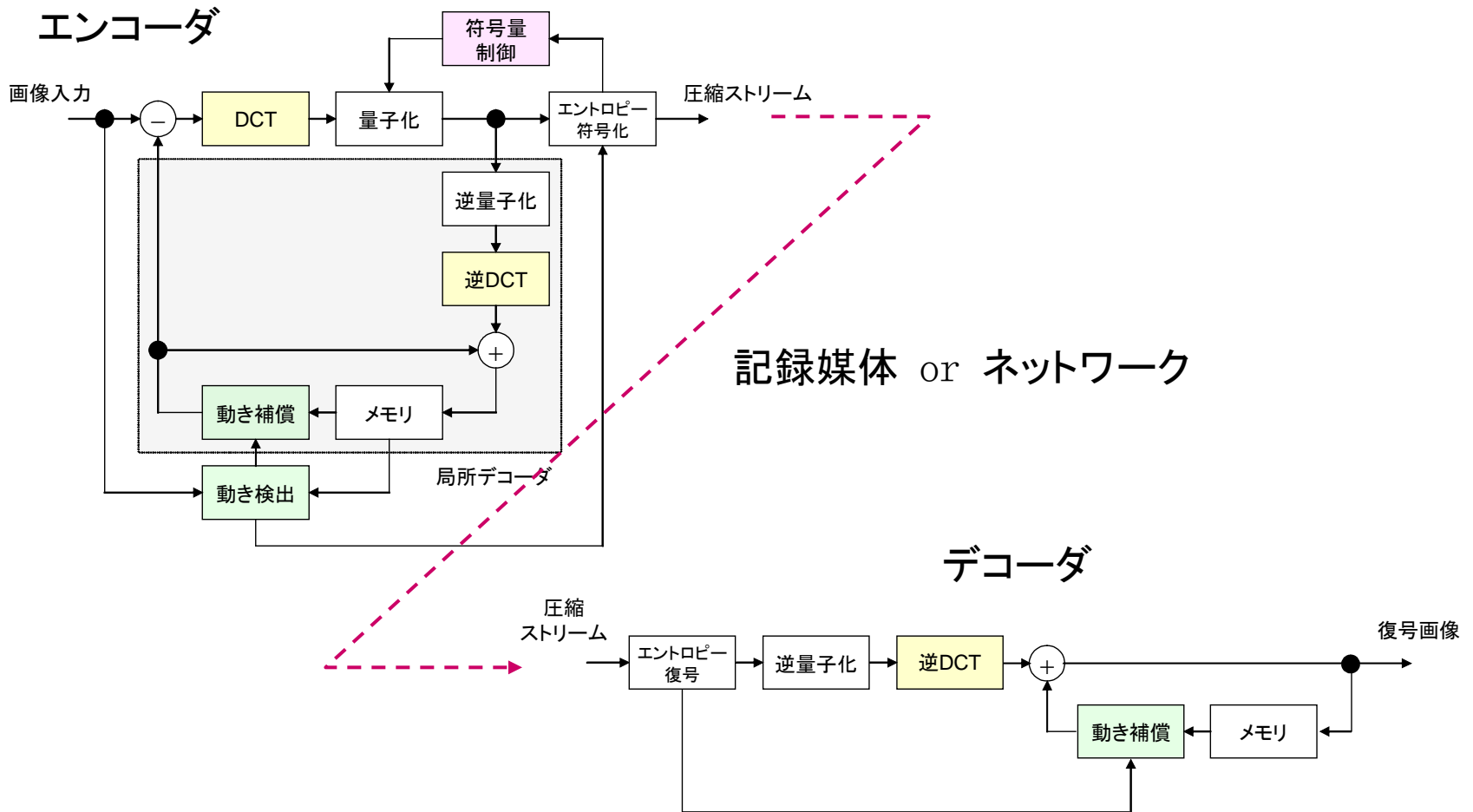
- (おそらく) 世界最初のストリーミングソフトウェア



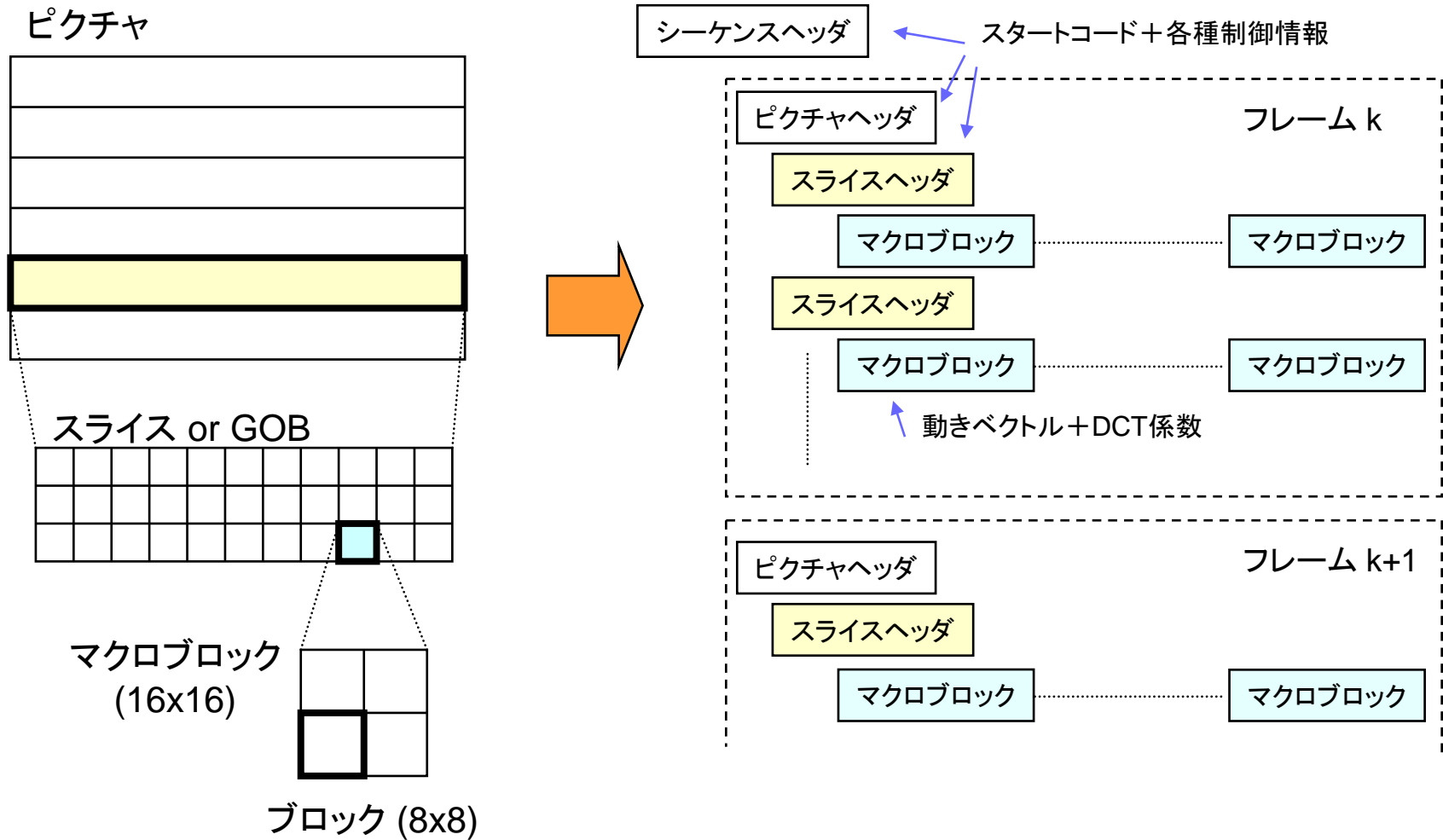
もともと UCB

<http://mediatools.cs.ucl.ac.uk/nets/mmedia/>

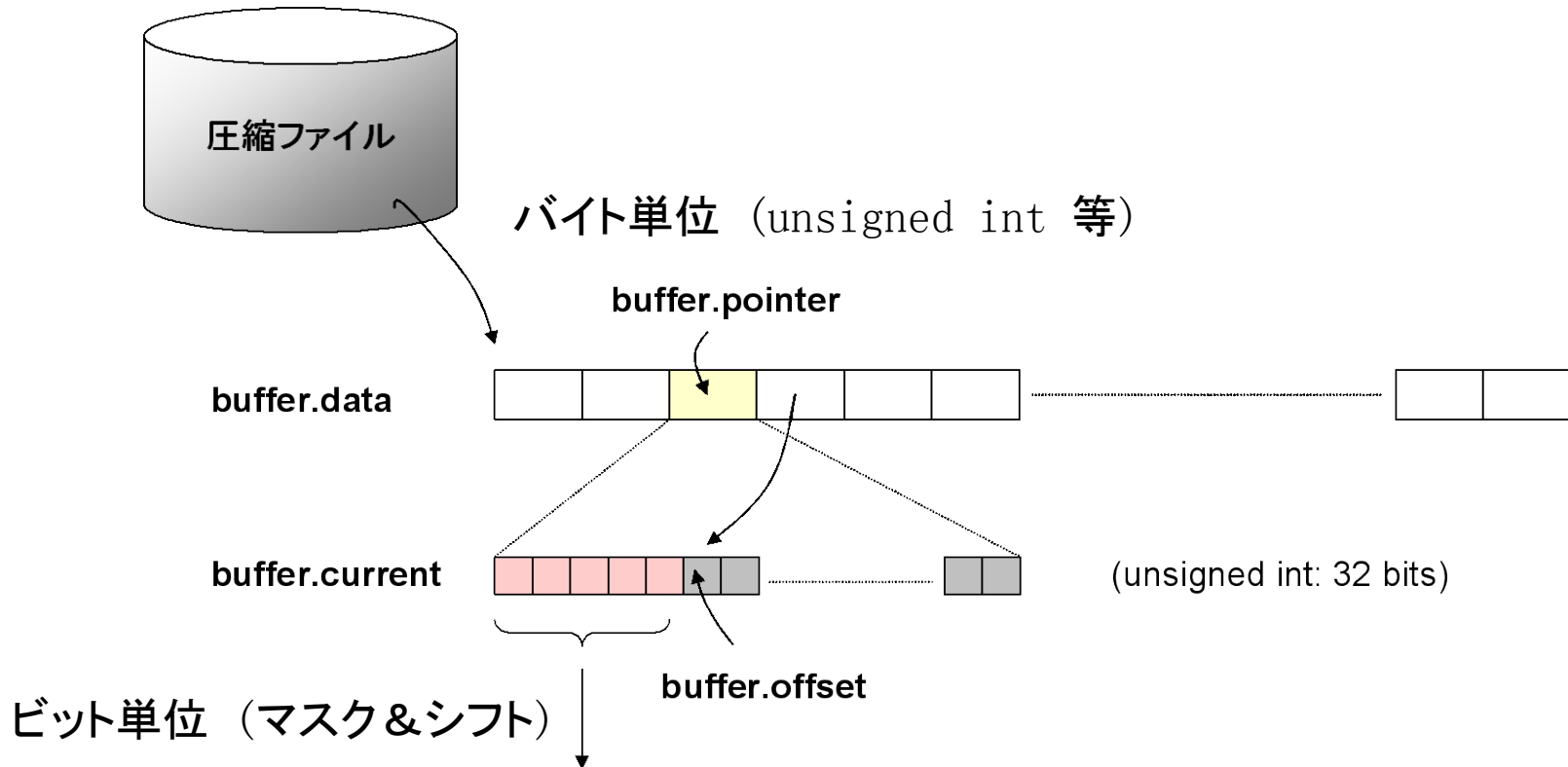
ビデオコーデックの基本構成



ビットストリーム構造



ビット処理

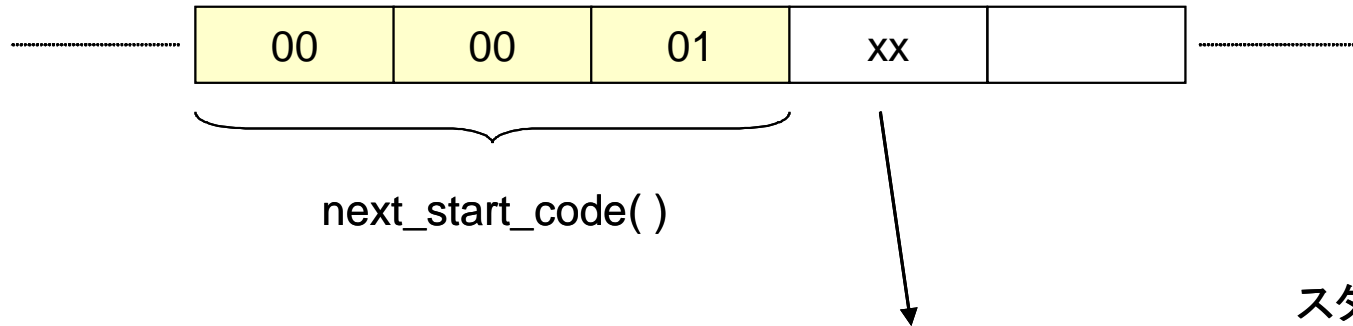


show_bits(): ビット表示のみ

get_bits(): ビット取得、current, offset 更新

スタートコードサーチ

buffer.data



#define SEQ_START_CODE	0x000001 b3
#define SEQ_END_CODE	0x000001 b7
#define GOP_START_CODE	0x000001 b8
#define PICTURE_START_CODE	0x000001 00
#define SLICE_MIN_START_CODE	0x000001 01
#define SLICE_MAX_START_CODE	0x000001 af

バイトアライン → ピクチャ境界、スライス境界、再同期ポイントの探索

テーブル参照

インデクス →
← 値

インデクス	値
0	A
1	B
2	C
⋮	⋮
n	Z

基本構成(デコーダ)

```
main() {  
    init();                // 初期化 (メモリ確保、各種パラメータ初期化)  
    while(1) {  
        read();           // 圧縮ファイル読み込み or ネットワーク受信  
        decode();        // ビデオ・デコード  
        display();       // 表示 (WIN32、X11 等)  
    }  
    close();              // 終了処理 (メモリ開放、表示系の終了処理)  
}
```

基本構成 (decode)

```
decode() { // ピクチャ処理
    picture_header_search(); // ピクチャヘッダ探索・復号
    while(slice) { // スライス処理
        slice_header_search(); // スライスヘッダ探索・復号
        while(macroblock) { // マクロブロック処理
            mb_header_decode(); // マクロブロックヘッダ復号
            motion_compensation(); // 動き補償予測
            while(block) { // ブロック処理
                i_huffman(); // ハフマン復号
                i_quantization(); // 逆量子化
                i_dct(); // 逆 DCT
                frame_update(); // フレームメモリ更新
            }
        }
    }
}
```

基本構成(エンコーダ)

```
main() {  
    init();           // 初期化 (メモリ確保、各種パラメータ初期化)  
    while(1) {  
        read();      // 画像ファイル読み込み or キャプチャ  
        encode();    // ビデオ・エンコード  
        write();     // 圧縮ファイル書き出し or ネットワーク送信  
    }  
    close();         // 終了処理 (メモリ開放、各種終了処理)  
}
```

基本構成 (encode)

```
encode() { // ピクチャ処理
    picture_header(); // ピクチャヘッダ書き出し
    while(slice) { // スライス処理
        slice_header(); // スライスヘッダ書き出し
        while(macroblock) { // マクロブロック処理
            motion_estimation(); // 動き検出
            motion_compensation(); // 動き補償予測
            mb_header(); // マクロブロックヘッダ書き出し
            while(block) { // ブロック処理
                dct(); // DCT (離散コサイン変換)
                quantization(); // 量子化
                huffman(); // ハフマン符号書き出し
                i_quantization(); // 逆量子化
                i_dct(); // 逆 DCT
                frame_update(); // フレームメモリ更新
            }
        }
    }
}
```

H.264 JMの場合 (エンコーダ)

```
int main(int argc, char **argv) {  
    // 各種初期化  
    alloc_xxx();  
    params = malloc(...);  
    Configure (argc, argv);  
    init_encoder(img);  
  
    // エンコード  
    encode_sequence(img);  
  
    // 終了処理  
    free_xxx();  
    free( params );  
}
```

H.264 JMの場合 (encode)

```
static void encode_sequence(ImageParameters *img) {  
    // ループ  
    for (curr_frame_to_code = 0; curr_frame_to_code < params->no_frames;  
        curr_frame_to_code++) {  
        // 各種パラメータ設定  
  
        // エンコード実行  
        encode_one_frame(img);  
  
        // 各種パラメータ更新  
    }  
}
```


H.264 JMの場合 (encode)

```
int encode_one_frame (ImageParameters *img) {
    // 初期化、画像データ読み込み
    put_buffer_frame (img);
    init_frame (img);
    ReadOneFrame (...);

    // 圧縮実行
    rc_init_frame(img->frm_no_in_file);
    frame_picture (frame_pic[0], &imgData, 0);  ———> code_a_picture()
                                                    ———> code_a_plane()
    // 結果書き出し、パラメータ更新
                                                    ———> encode_one_slice()
    frame_mode_buffer (img, params);
    write_non_vcl_nalu();
    writeout_picture (frame_pic[img->rd_pass]);
    rc_update_pict_frame_ptr(quadratic_RC, bits);
    UpdateDecoders (params, img, enc_picture);
    compute_distortion(&imgData);

    return ;
}
```