CMPN206: Multimedia

Lecture 9: Video Compression I

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2014

Agenda

- Video Compression
- Motion Compensation
- H.261
- H.263

Acknowledgments: Most slides are adapted from Richard Ladner, from Li and Drew, and from Khaled Sayood.

Video

- A video is made up of many *frames* or images that are displayed in rapid order
- 25-30 frames per second allows the human visual system to integrate frames and *perceive* discrete displacements as a continuous motion
- It has many applications:
  - Video conferencing
    - Real time video, very low delay
  - Live broadcast
    - Modest delay is tolerable
  - Video on demand/DVD
    - Encoding can be more complex than decoding
    - Random access is required

Video Compression

- The amount of data in *uncompressed* video is *huge*, e.g. in HDTV at 1920×1080 30 fps with 8 bits per channel gives 1.5 Gbps
- We need to use compression to reduce the amount of storage and network bandwidth required
- In *image compression*, we used *spatial redundancy* where pixel values do not change much and can be predicted from nearby pixels
- In *video compression*, we will use both *spatial redundancy* and *temporal redundancy*

Redundancy in Videos

- *Spatial redundancy*:
  - Pixel values within the *same frame* are close to nearby pixel values
- *Temporal redundancy*:
  - Pixel values in one frame are close to nearby pixel values in other *nearby frames*

- We can use *predictive coding* based on previous frames.
- Compression proceeds by *subtracting* images: subtract in time order and code the residual error.
- It can be done even better by *searching* for just the *right parts* of the image to subtract from the previous frame.

Temporal Prediction

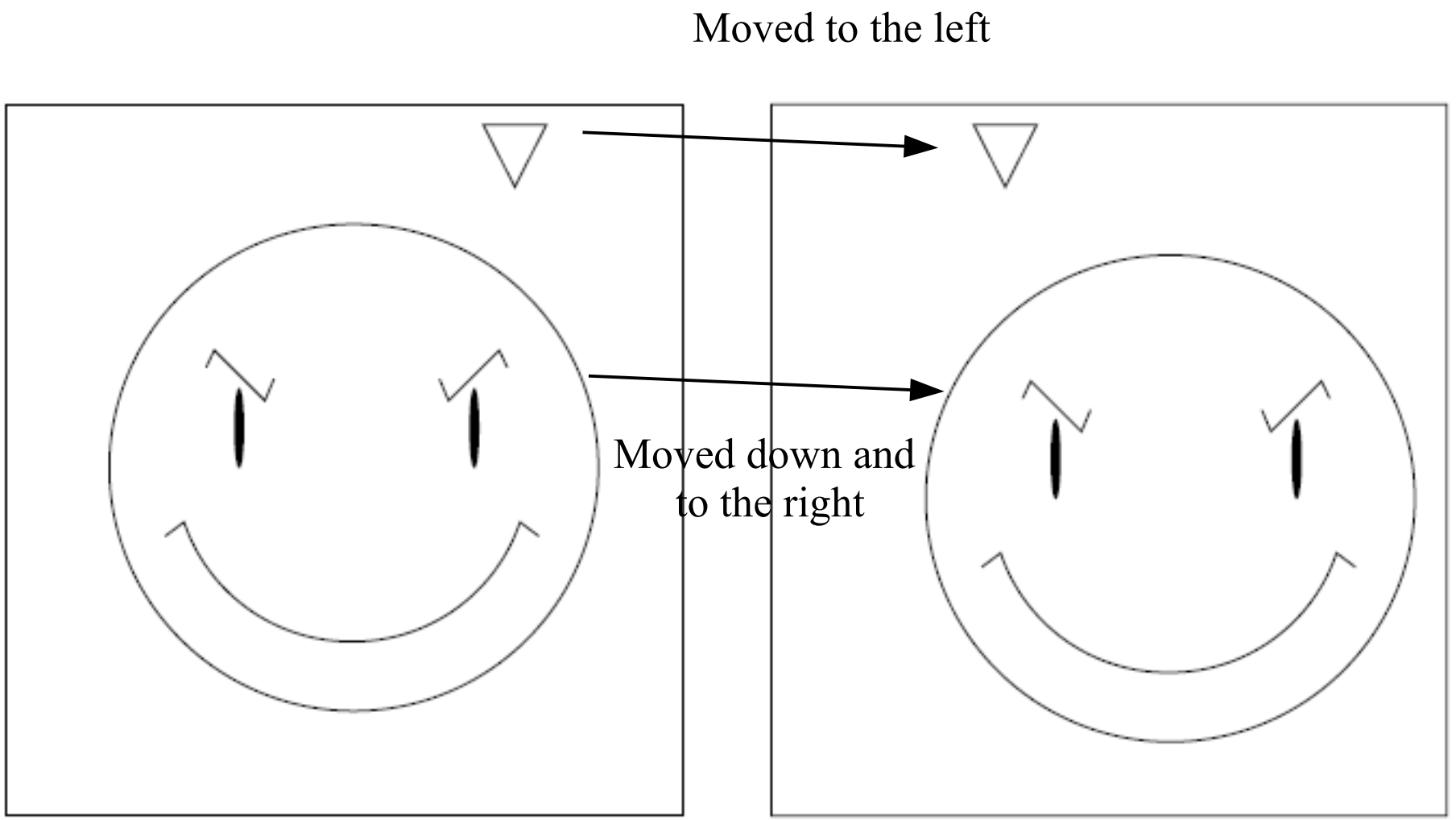


FIGURE 18. 1 **Two frames of a video sequence.**

What happens if we take the *difference* of the two frames?

Temporal Prediction

Problem! The difference image actually has *more* information!!

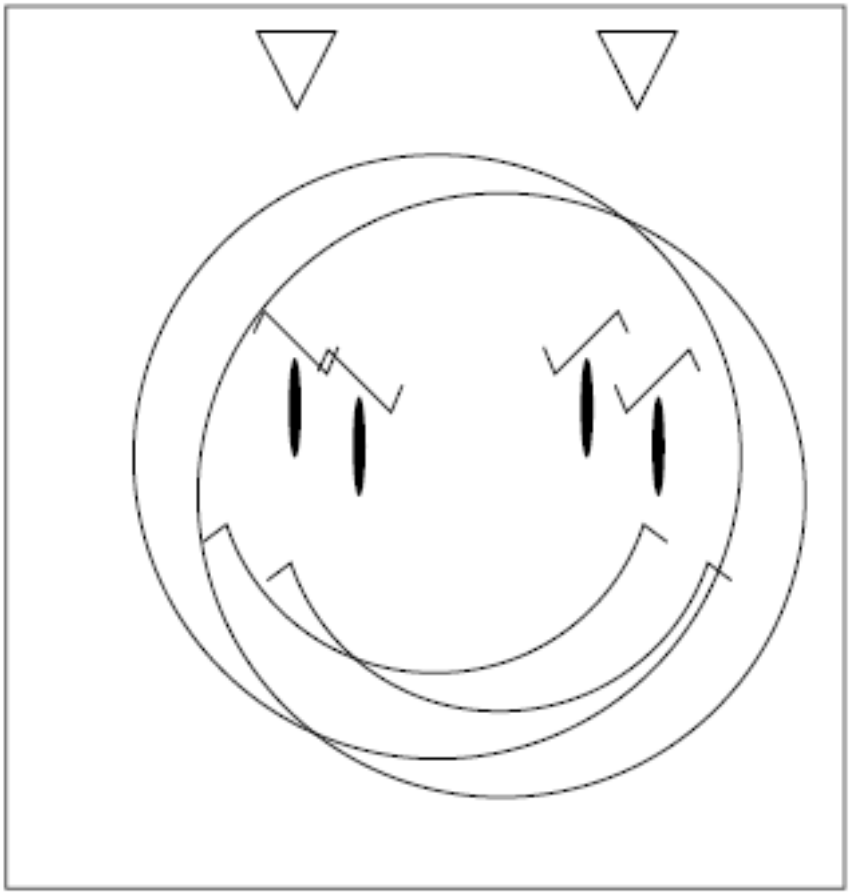


FIGURE 18. 2 **Difference between the two frames.**

Solution: Divide the image into *blocks* and find the *best matching* block

Motion Compensation

- *Motion Estimation*: divide the image into *macroblocks* and find the best matching block in nearby frames
- *Motion Compensation Prediction*: derive a *prediction* of the blocks in the current frame based on the motion estimation
- *Prediction Error Derivation*: code the difference between the prediction and the current block value

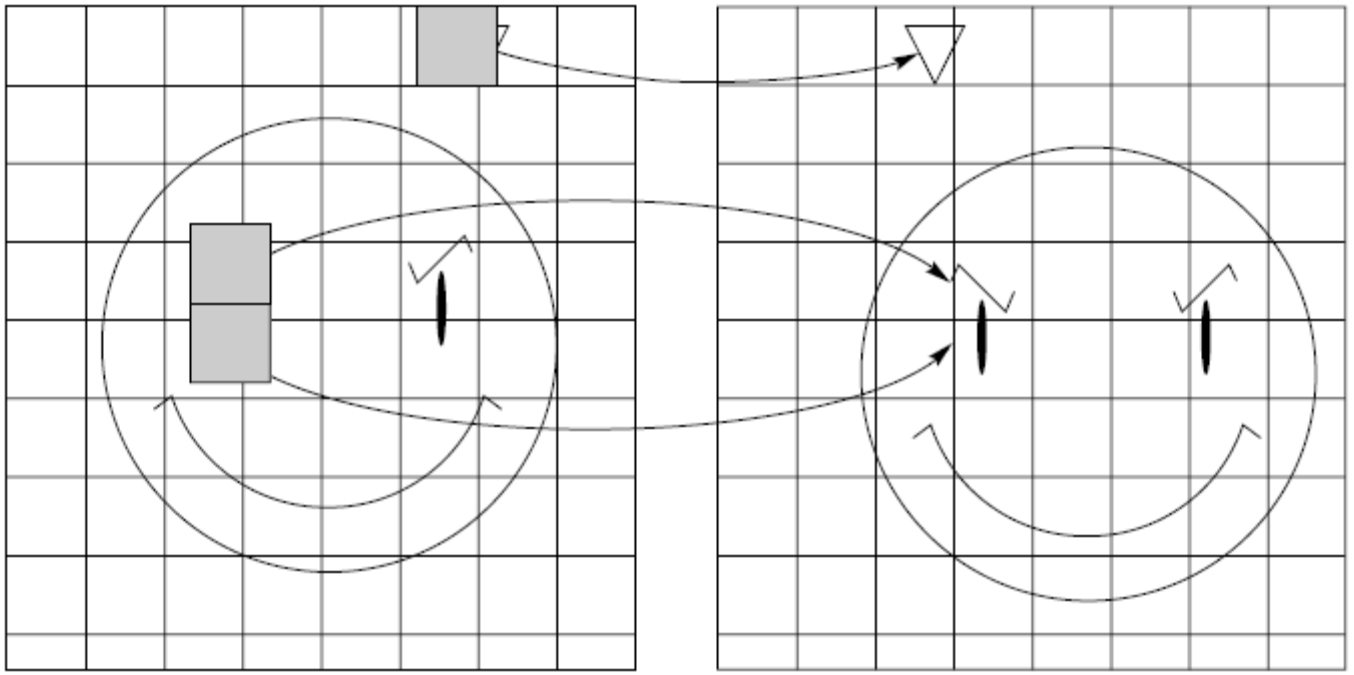


FIGURE 18. 3 **Motion-compensated prediction.**

Example

- For each *macroblock* in the current (*target*) search for the best matching block in the previous (*reference*) frame and estimate the *motion vector* i.e. the displacement from the target to reference frame
- Code the difference between the macroblock and its match
- In this case, only need to code and send the *motion vectors*

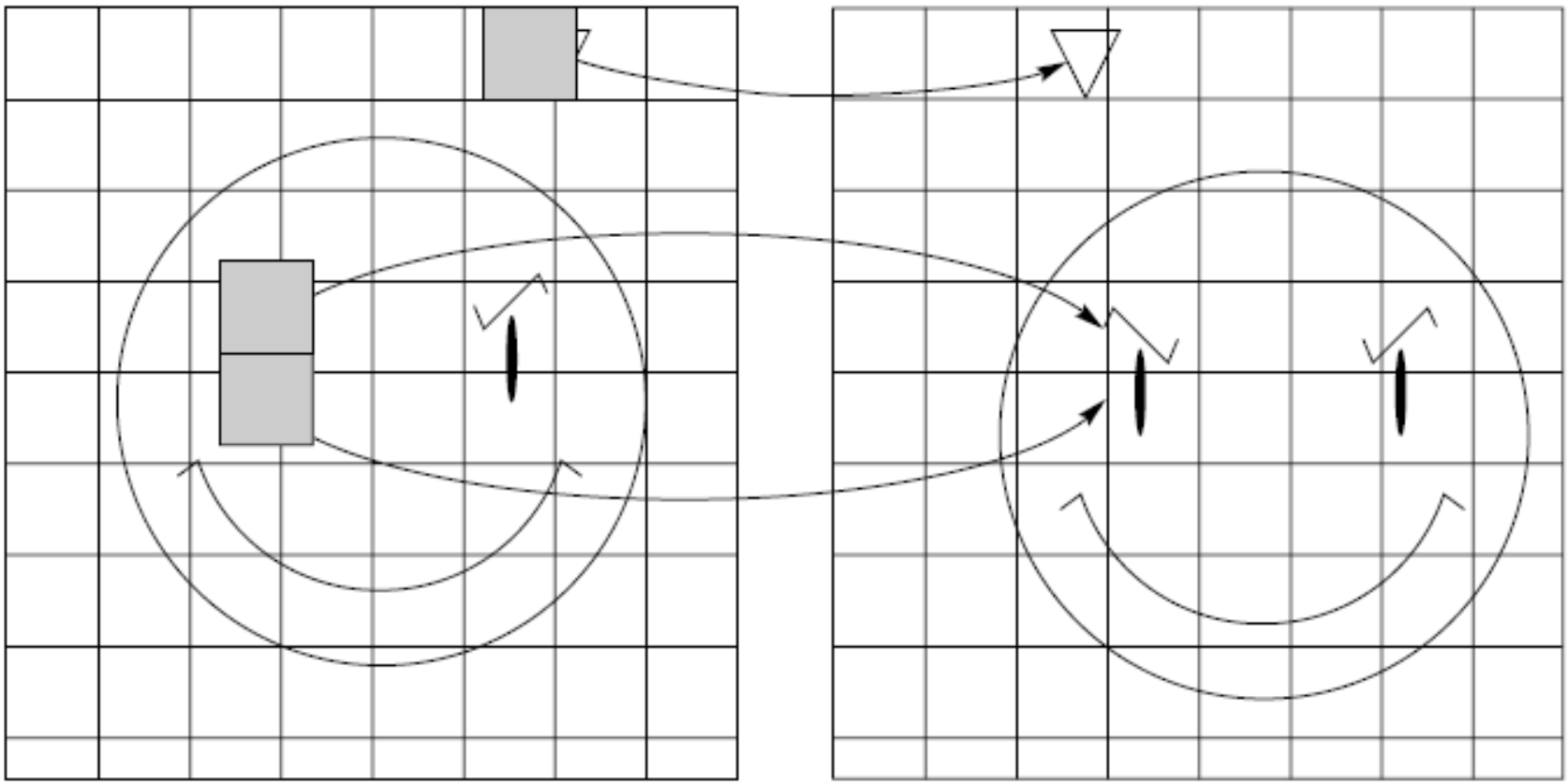
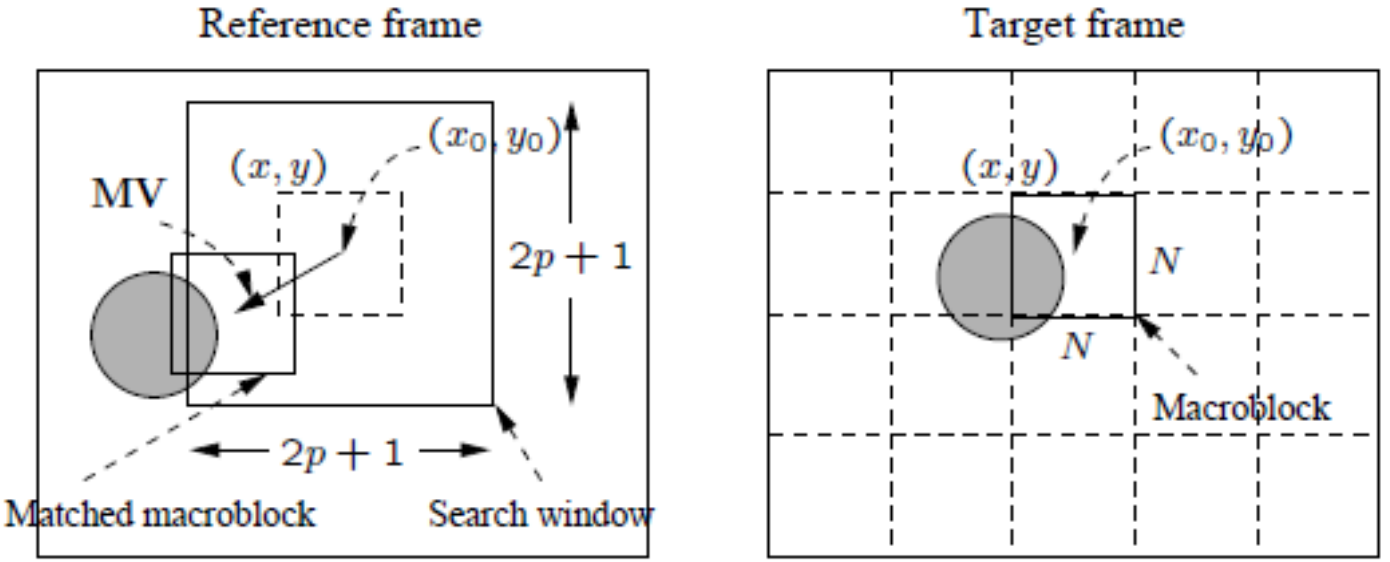


FIGURE 18. 3 **Motion-compensated prediction.**

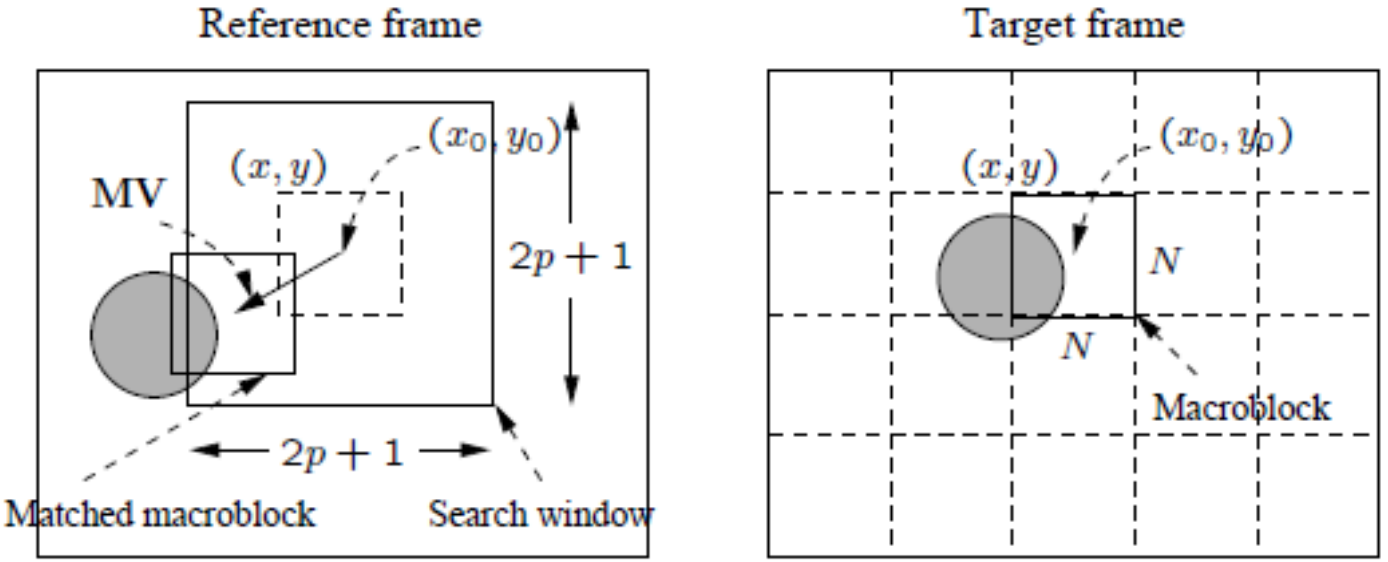
Motion Estimation

- Divide each frame into *macroblocks* of size $N \times N$
  - By default, $N = 16$ for luminance images, and $N = 8$ for chrominance images (when chroma subsampling of 4:2:0 is used)
- The current frame is called the *target* frame
- For each macroblock in the target frame, search for the most similar macroblock in previous and/or future frames, called *reference* frames



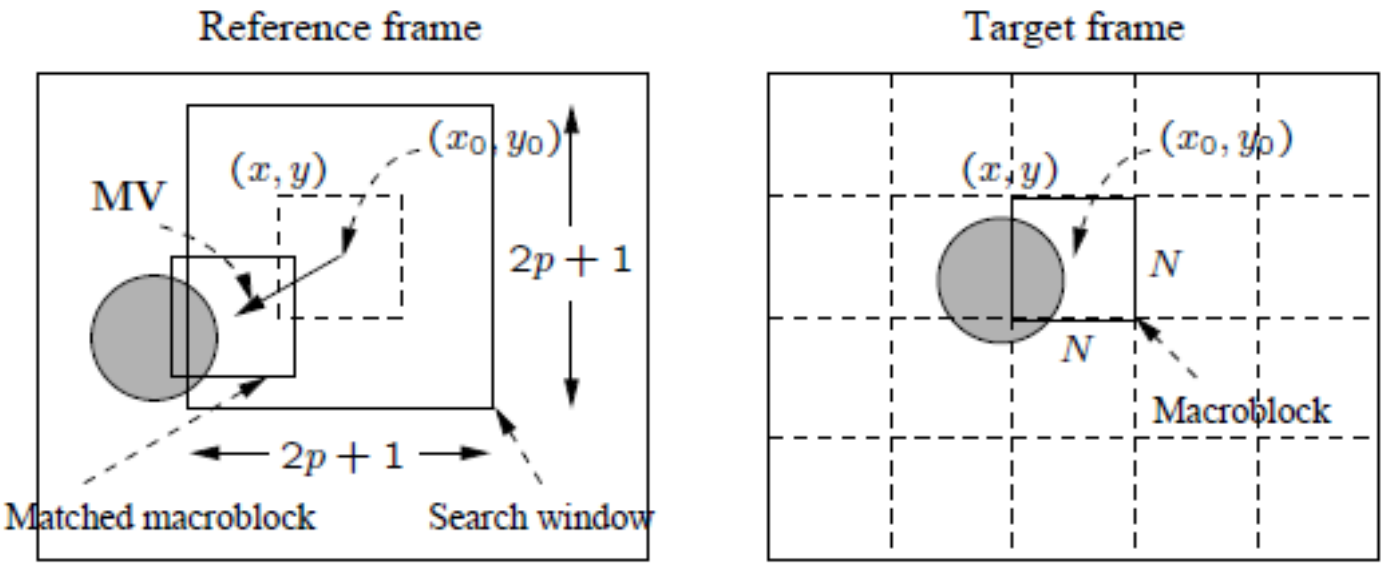
Motion Estimation

- The *displacement* between the target macroblock and the reference macroblock is called the *motion vector* e.g.
  - If the target macroblock (in the target frame) is at location (16, 32) and the reference macroblock (in the reference frame) is at location (10, 40), then the motion vector MV = (10, 40) – (16, 32) = (–6, 8) which is the *displacement vector* from the *target* to the *reference* macroblocks



Motion Estimation

- The *search* for the MV is usually restricted to a small neighborhood of the target macroblock with horizontal and vertical displacements of $[-p, p]$ i.e. a search window of size $(2p + 1) \times (2p + 1)$
- The *best matching* macroblock is the one whose *distance* is smallest from the target macroblock



Motion Estimation

- The *distance* measure is usually:
  - Sum of Absolute Differences (SAD)

$$SAD(i,j)=\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}\left|C(x+k,y+l)-R(x+i+k,y+j+l)\right|$$

where:

- *k and l* are indices for pixels in a macroblock
- *i* and *j* are horizontal and vertical displacements
- C ($x + k, y + l$) are pixels in the target macroblock
- C ($x + i + k, y + j + l$) are pixels in the reference macroblock

  - Sum of Squared Differences (SSD)

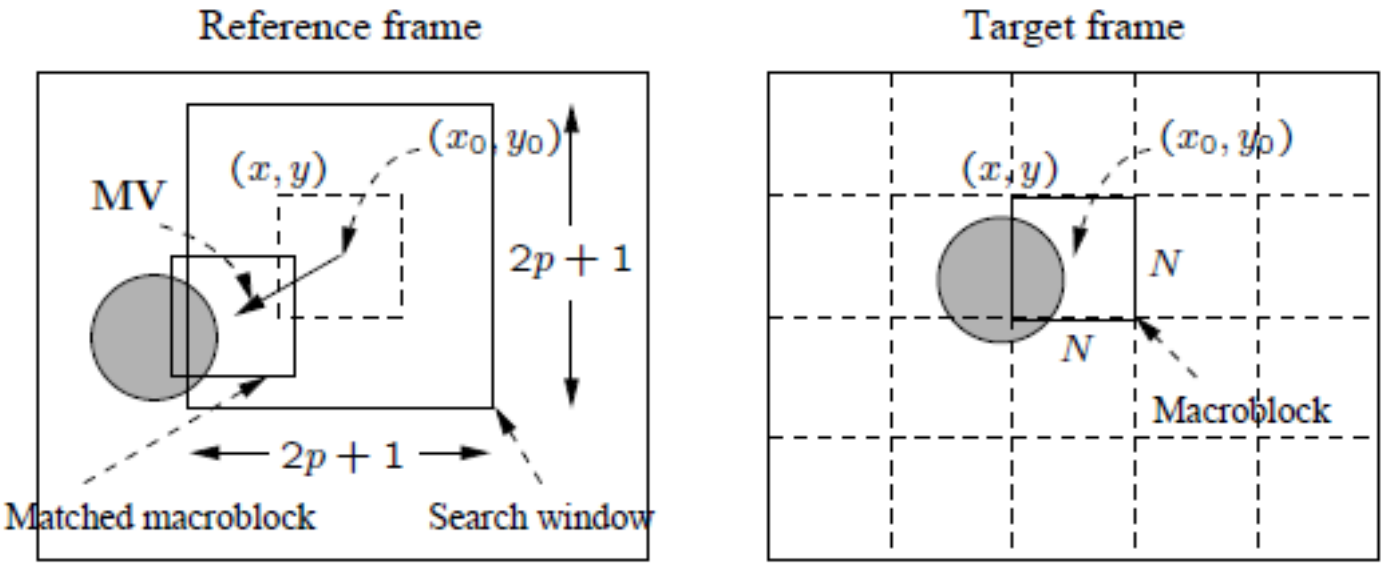
$$SAD(i,j)=\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}\left(C(x+k,y+l)-R(x+i+k,y+j+l)\right)^2$$

Motion Estimation

- The *search* looks for the *motion vector* (u, v) that minimizes the distance measure

$$MV = (u, v) = \arg\min_{i \in [-p, p],\, j \in [-p, p]} SAD(i, j)$$

$$MV = (u, v) = \arg\min_{i \in [-p, p],\, j \in [-p, p]} SSD(i, j)$$



Motion Vector Search

- Sequential Search
- Logarithmic Search
- Hierarchical Search

1. Sequential Search

- Search the whole $(2p + 1)\times(2p + 1)$ window in the reference frame around the location of the macroblock in the target frame
- The vector (i, j) that offers the least SAD (or SSD) is designated as the motion vector $MV = (u, v)$ for the macroblock in the target frame

```
begin
  min MAD = LARGE NUMBER; /* Initialization */
  for i = -p to p
    for j = -p to p
      cur SAD = SAD(i; j);
      if cur SAD < min SAD
        min SAD = cur SAD;
        u = i; /* Get the coordinates for MV. */
        v = j;
end
```

1. Sequential Search

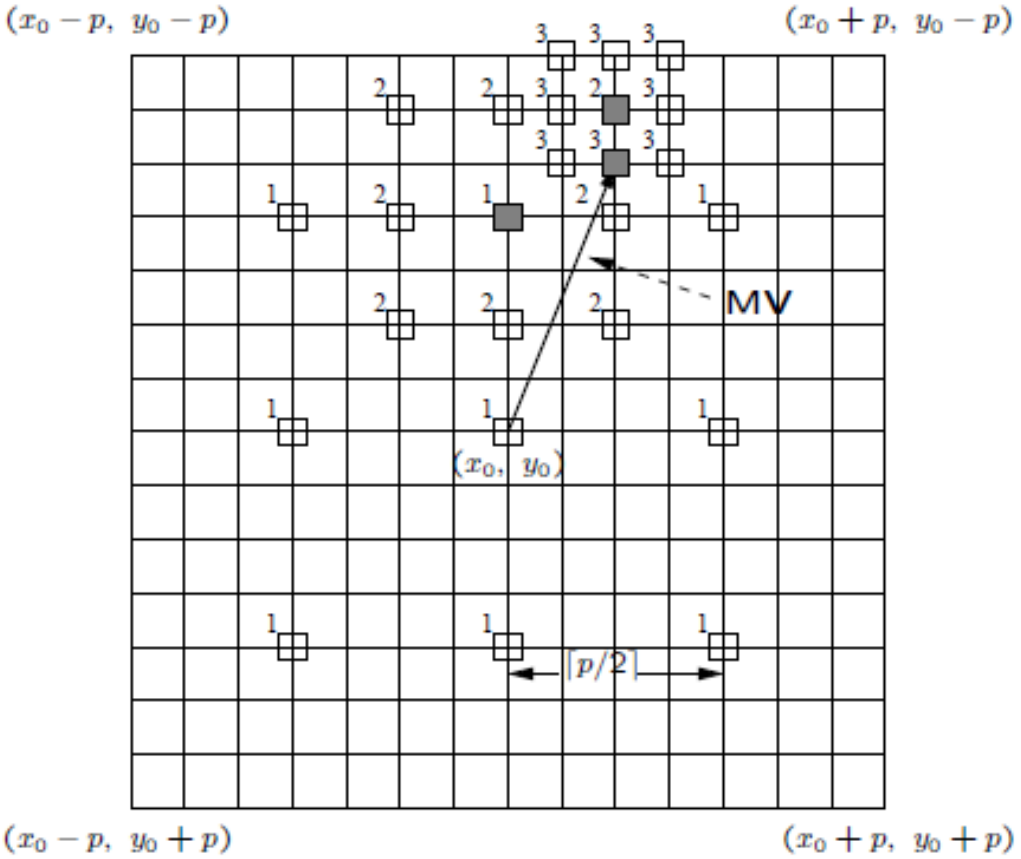
- Very *costly*: assuming each pixel requires three operations (subtraction, absolute value, addition), the number of operations = $(2p + 1) \times (2p + 1) \times N^2 \times 3 = O(p^2 N^2)$

- For a video with 720×480 with $p = 15$ and $N = 16$ and 30 fps, the number of operations required for *each* motion vector is $31^2 \times 16^2 \times 3$
- Since the number of macroblocks per frame is $720 \times 480 / N^2$, the total number of operations per second:
$(2p + 1)^2 \times N^2 \times 3 \times 720 \times 480 / N^2 \times 30 = 29.89 \times 10^9$

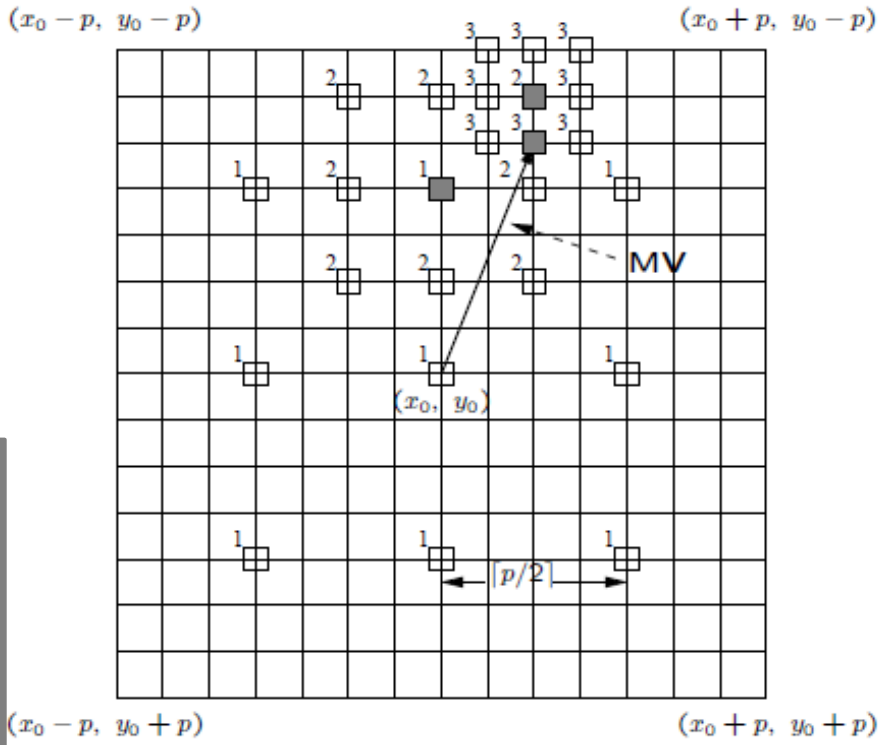
2. Logarithmic Search

- Similar to binary search:
  - Check only 9 locations in the first iteration (marked as 1)
  - The search is repeated around the location that yields the *minimum*, and 9 more locations are searched (marked as 2) with *half* the offset, and so on

- Suboptimal (*not* guaranteed to return the *best* match)



2. Logarithmic Search




```
begin
 offset = p / 2

 Specify nine macroblocks within
 the search window in the Reference
 frame, they are centered at (x0; y0)
 and separated by offset horizontally
 and/or vertically;

 while last != TRUE
  Find one of the nine specified
   macroblocks that yields minimum SAD;
  if offset = 1 then last = TRUE;
  offset = offset / 2 ;
  Form a search region with the new
   offset and new center found;
end
```

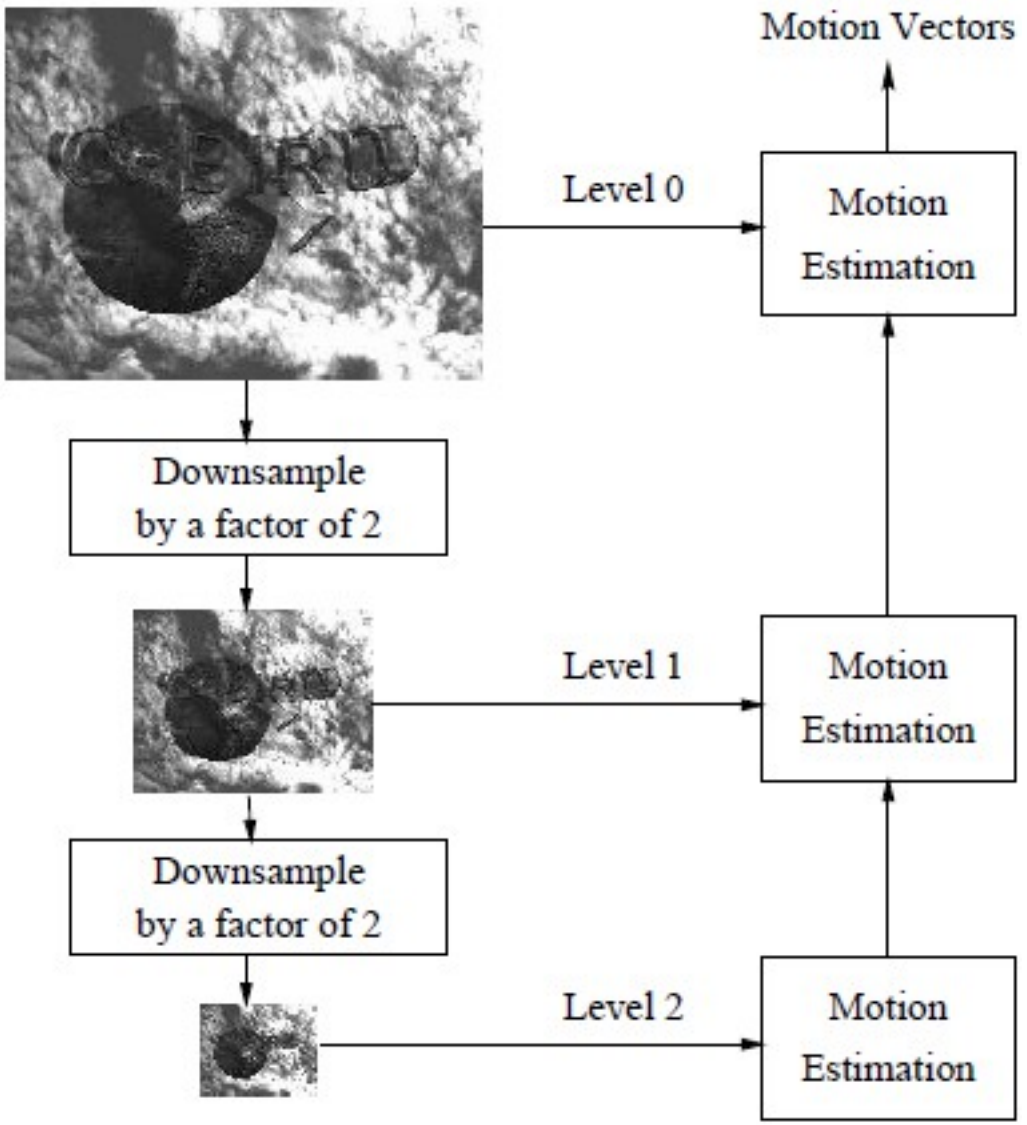
2. Logarithmic Search

- The number of reference macroblocks to be examined is reduced from $(2p + 1)^2$ down to $9\ (\log_2 p + 1)$ i.e. 9 for each *offset* value
- In fact, it can be even reduced to $8\ (\log_2 p + 1) + 1$ because the central macroblock can be reused in the next iteration
- The total number of operations per macroblock is reduced to $= (8\ (\log_2 p + 1) + 1) \times N^2 \times 3 = O(\log_2 p\ N^2)$

- For a video with 720×480 with $p = 15$ and $N = 16$ and 30 fps, the total number of operations per second:
$(8\ (\log_2 p + 1) + 1) \times N^2 \times 3 \times 720 \times 480 / N^2 \times 30 = 1.25 \times 10^9$

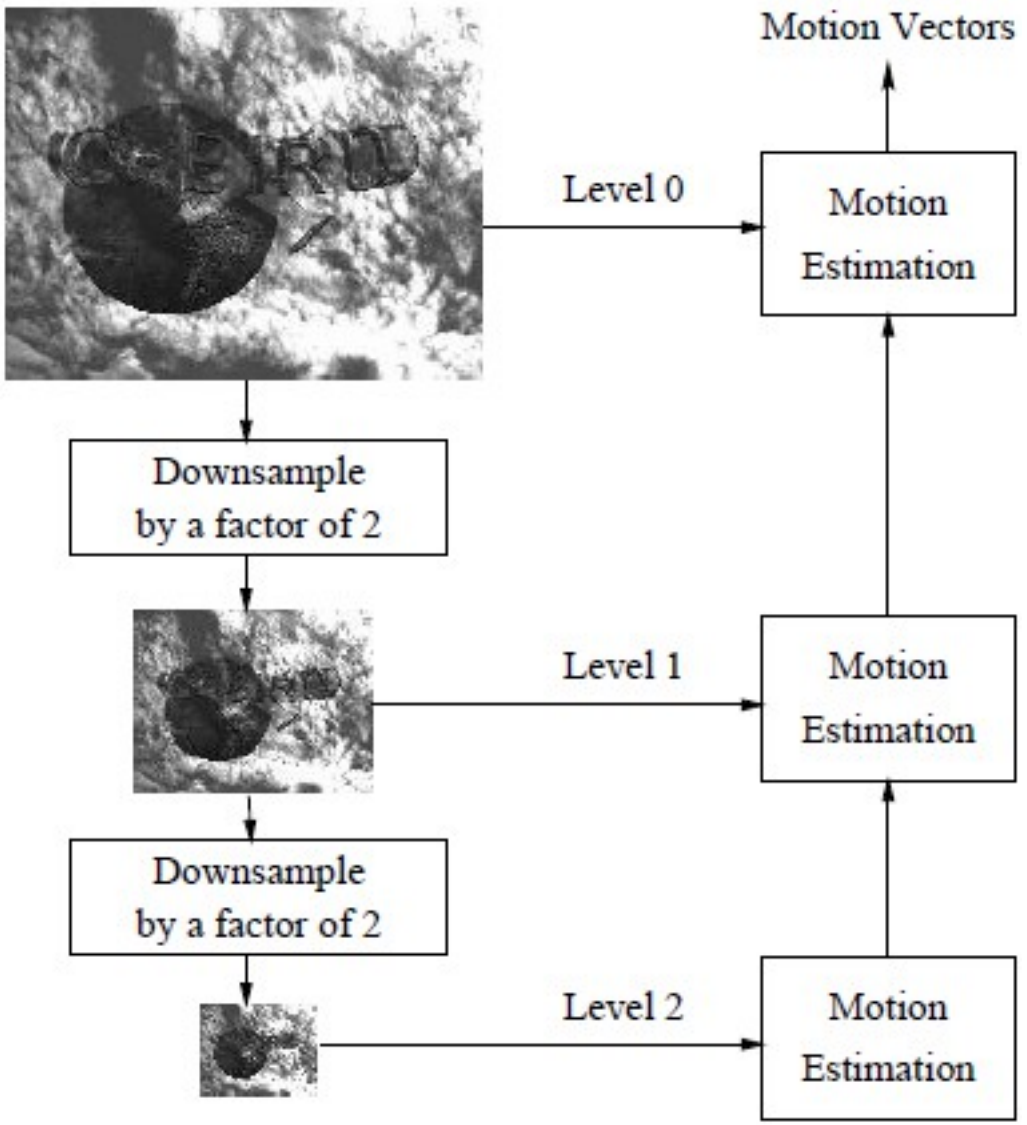
3. Hierarchical Search

- Uses a hierarchical (or *multiresolution*) approach
- The frames are first *downsampled* by factors of 2 e.g. for two levels, produce two versions at half and quarter resolution
- The macroblocks get *smaller* at each level, and so does p, and thus the number of operations required is reduced
- Search starts at the *highest* level i.e. *smallest* resolution, and is refined by at lower levels i.e. higher resolutions



Example

- Using *two* levels
- For a target macroblock whose center is at (x_0, y_0)
- Transform the center to level 2: $(x_0^2, y_0^2) = (x_0 / 4, y_0 / 4)$
- Find the initial estimate of the motion vector $MV^2 = (u^2, v^2)$ at level 2
- Transform the center in the reference frame by MV^2 to level 1: $(x_0^1, y_0^1) = (2(x_0^2 + u^2), 2(y_0^2 + v^2))$
- Check the 9 locations around (x_0^1, y_0^1) and repeat the same at level 0 …



3. Hierarchical Search

- The size of the reference macroblocks are not the same anymore
- The total number of operations per macroblock is reduced to

$$= (2\,(p/4+1))^2 \times (N/4)^2 + 9\,(N/2)^2 + 9\,N^2) \times 3$$

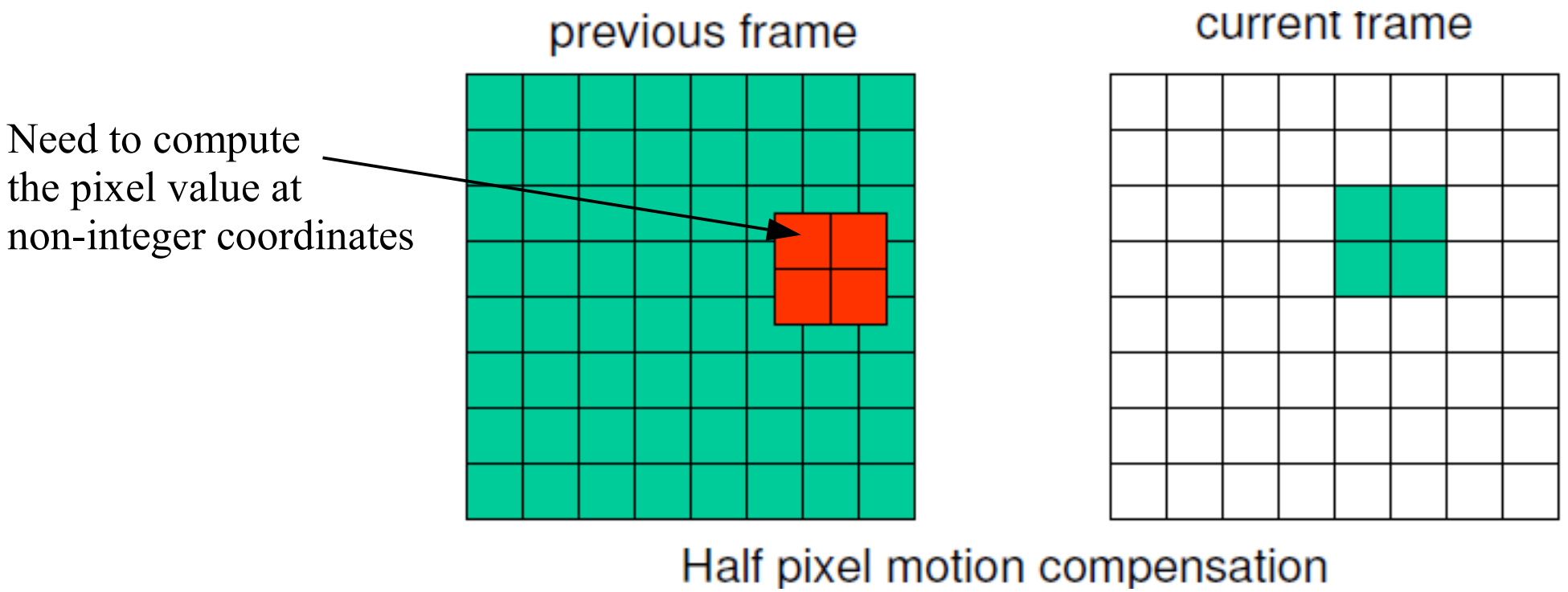
Level 2 Sequential Search 9 blocks at Level 1 9 blocks at Level 0

- For a video with 720×480 with $p = 15$ and $N = 16$ and 30 fps, the total number of operations per second: 0.51×10^9

Search Method	*OPS_per_second* for 720 × 480 at 30 fps	
	$p = 15$	$p = 7$
Sequential search	29.89×10^9	7.00×10^9
2D Logarithmic search	1.25×10^9	0.78×10^9
3-level Hierarchical search	0.51×10^9	0.40×10^9

Fractional Motion Estimation

- Instead of having motion vectors with *integer* coordinates, we can have *fractional* motion vectors e.g. at half or quarter pixels
- This can produce better results to find a better matching block, but we need to compute *subpixel* values
- *Linear interpolation* can be used for that



Fractional Motion Estimation

- Calculate the value of the interpolated pixel as the *weighted average* of the overlapping pixels

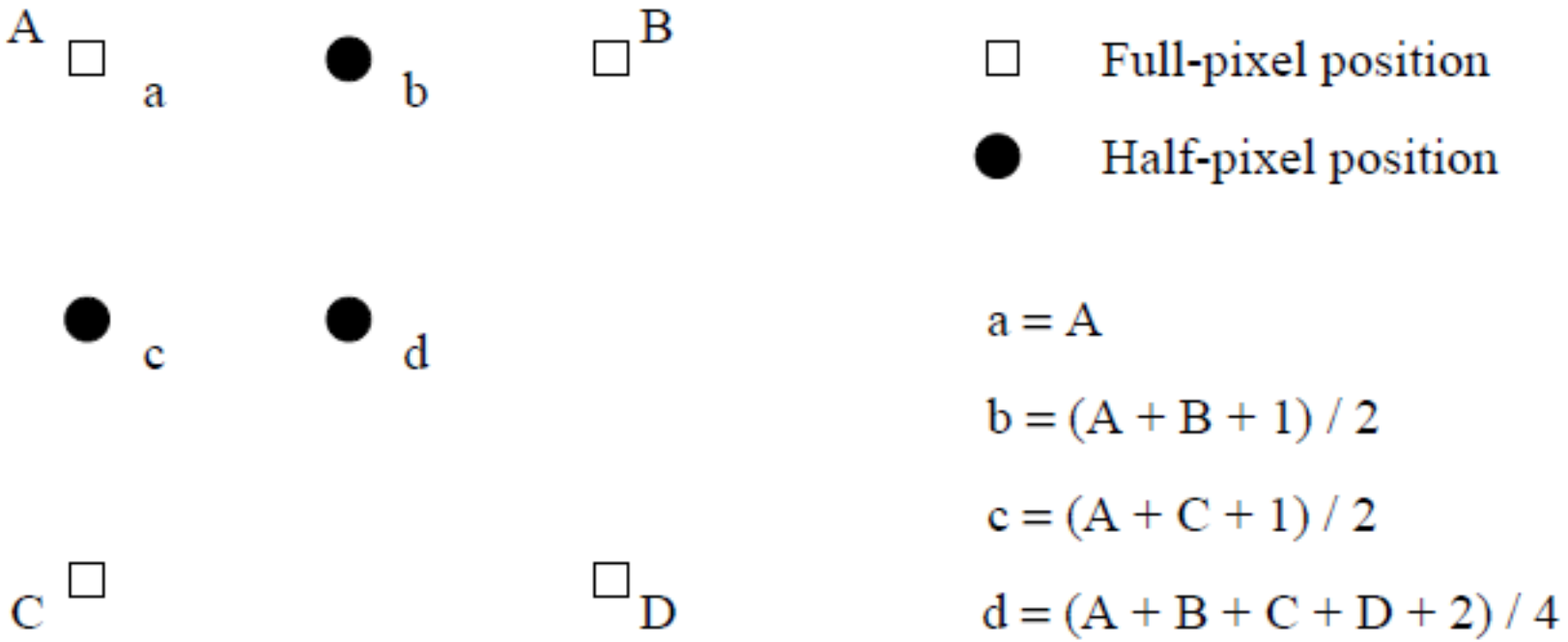


Fig. 10.12: Half-pixel Prediction by Bilinear Interpolation

Motion Compensation

- *Motion Estimation*: find the best motion vector
- *Motion Compensation Prediction*: derive a *prediction* of the blocks (and motion vector) in the target frame based on the estimated motion vector
- *Prediction Error Derivation*: code the difference between the prediction and the current block value

- Will look at specific implementations of motion compensation in different video coding techniques

H.261

- An earlier digital video compression standard, its principle of MC-based compression is retained in all later video compression standards.
- The standard was designed for videophone, video conferencing and other audiovisual services over ISDN.
- The video codec supports bit-rates of $p \times 64$ kbps, where p ranges from 1 to 30 (Hence also known as p*64).
- Require that the delay of the video encoder be less than 150 msec so that the video can be used for real-time bidirectional video conferencing.
- Started in 1988 by the CCITT
- Adopted as a standard by ITU-T in 1990

H.261 Video Formats

- Common Intermediate Format (CIF) and Quarter CIF (QCIF)
- 4:2:0 chroma subsampling
- 30 fps non-interlaced (*progressive*)

Video format	Luminance image resolution	Chrominance image resolution	Bit-rate (Mbps) (if 30 fps and uncompressed)	H.261 support
QCIF	176×144	88×72	9.1	required
CIF	352×288	176×144	36.5	optional

H.261 Frame Sequence

- Two types of frames:
  - I-frames: Intra-frames or Independent frames
    - Treated as independent images
    - Compressed with JPEG-like *transform coding*
    - Only *spatial redundancy*
    - Sent a couple of times per second

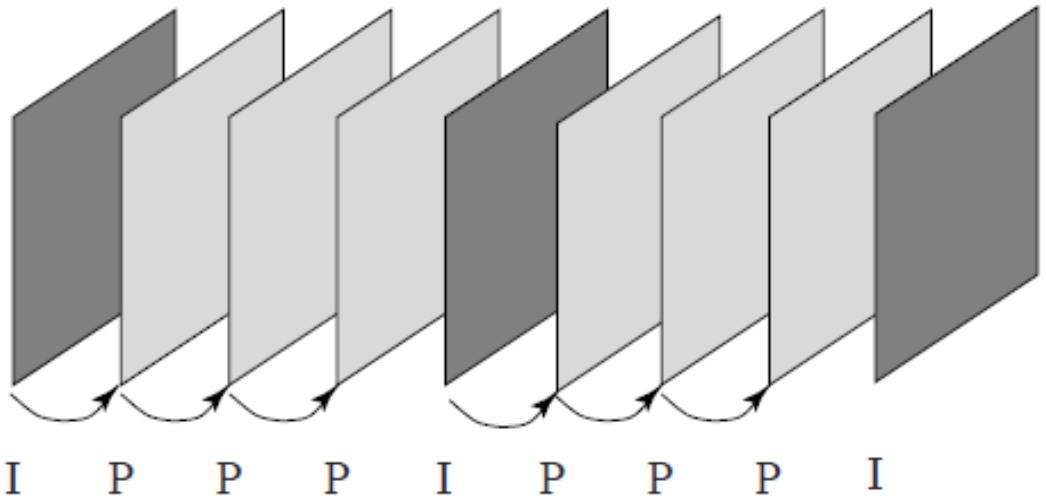


Fig. 10.4: H.261 Frame Sequence.

H.261 Frame Sequence

- Two types of frames:
  - P-frames: Inter-frames or Predicted frames
    - Depend on previous I-frames and/or P-frames
    - Uses *temporal redundancy* with motion compensation
    - Motion compensation with *integer coordinates* and search windows with $p = 15$

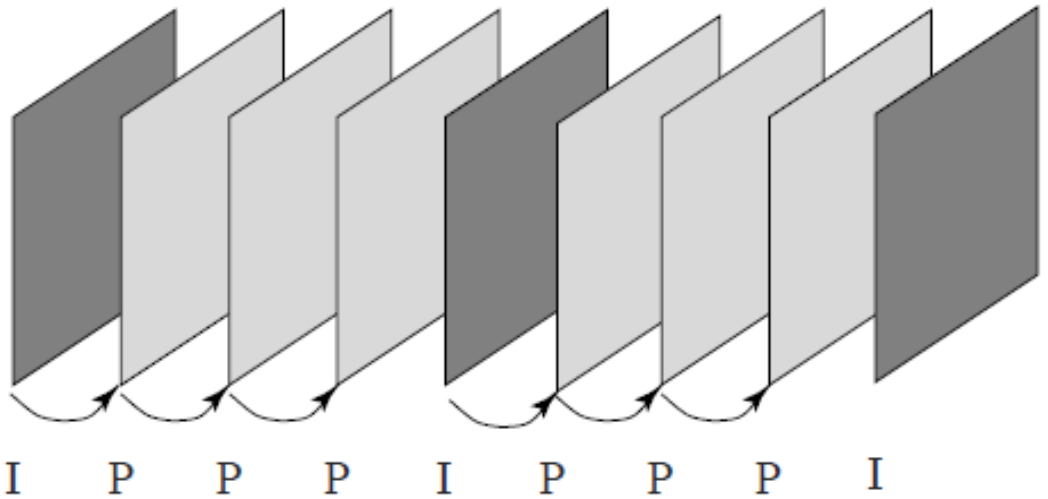


Fig. 10.4: H.261 Frame Sequence.

I-frame Coding

- I-frames are coded like JPEG
- Macroblocks are 16×16 for the *Y* channel (luminance) and 8×8 for *Cb* and *Cr* (chrominance)
- Each 8×8 block is transformed with 2D DCT, then quantized in a zigzag order, and entropy coded

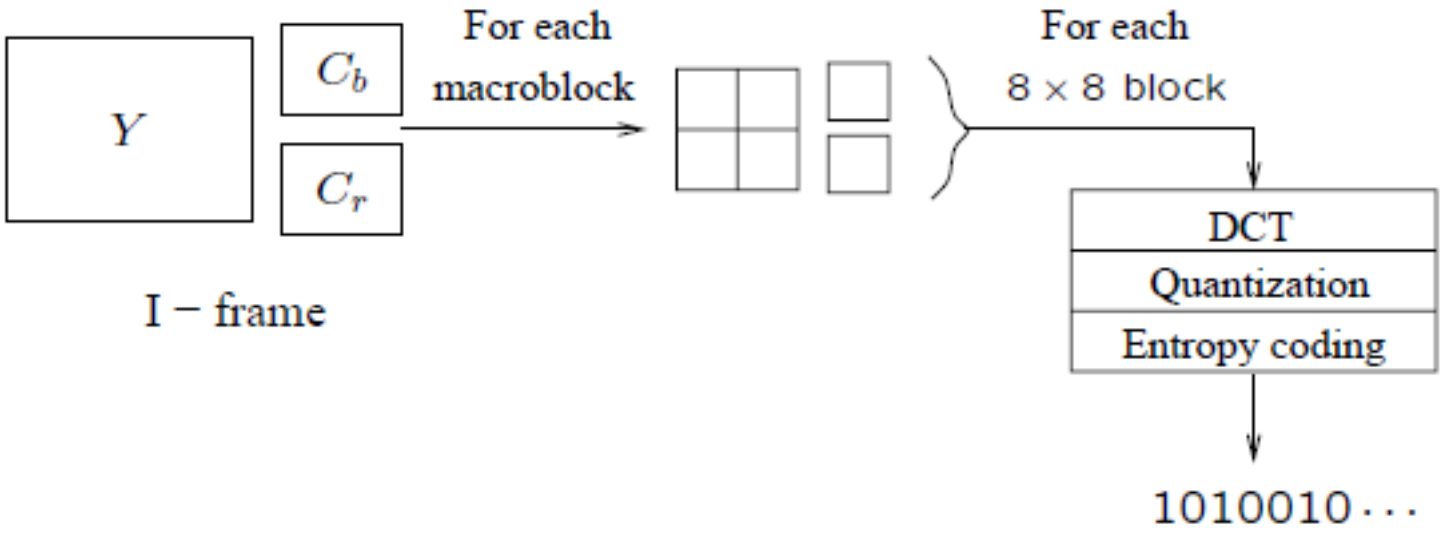
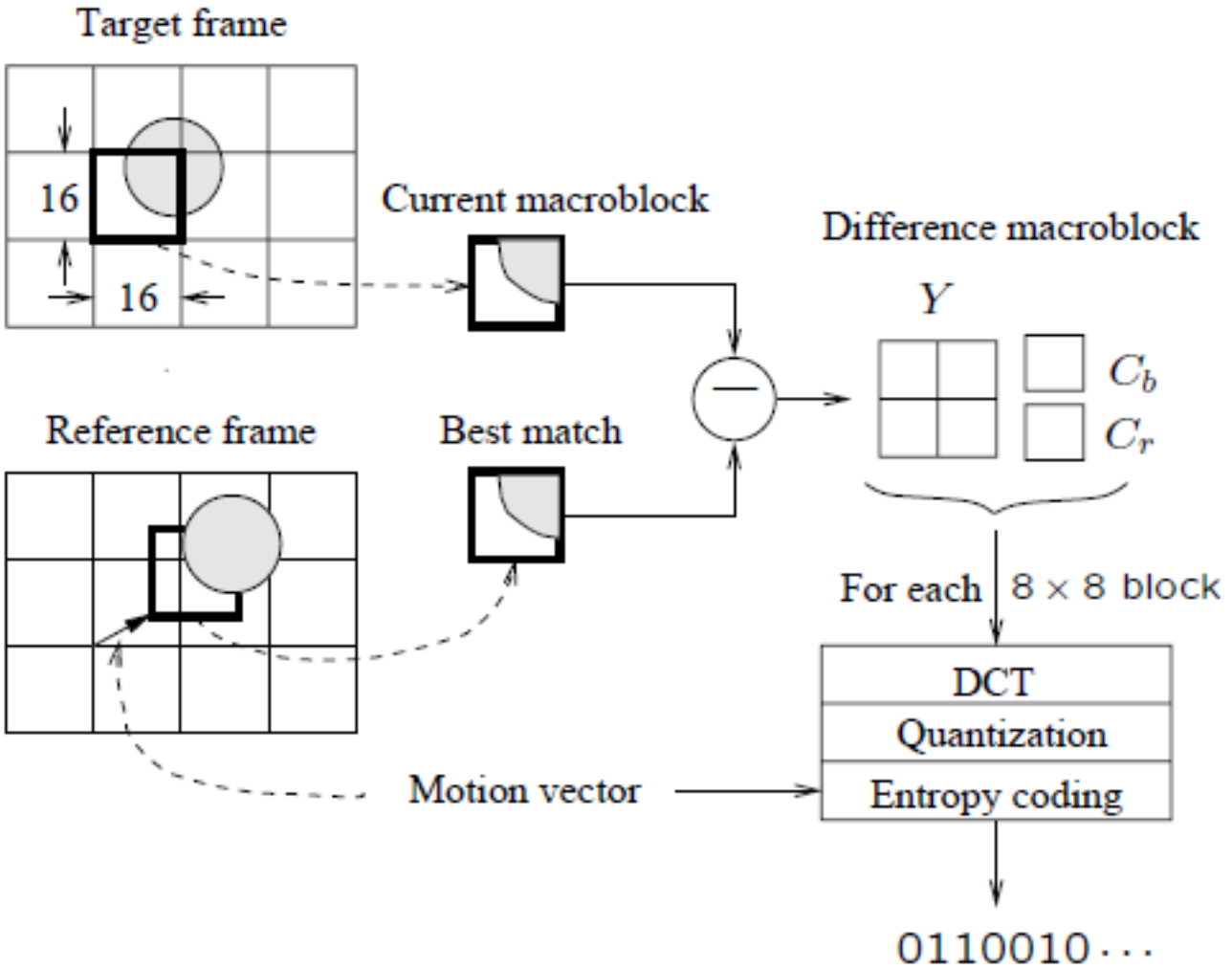


Fig. 10.5: I-frame Coding.

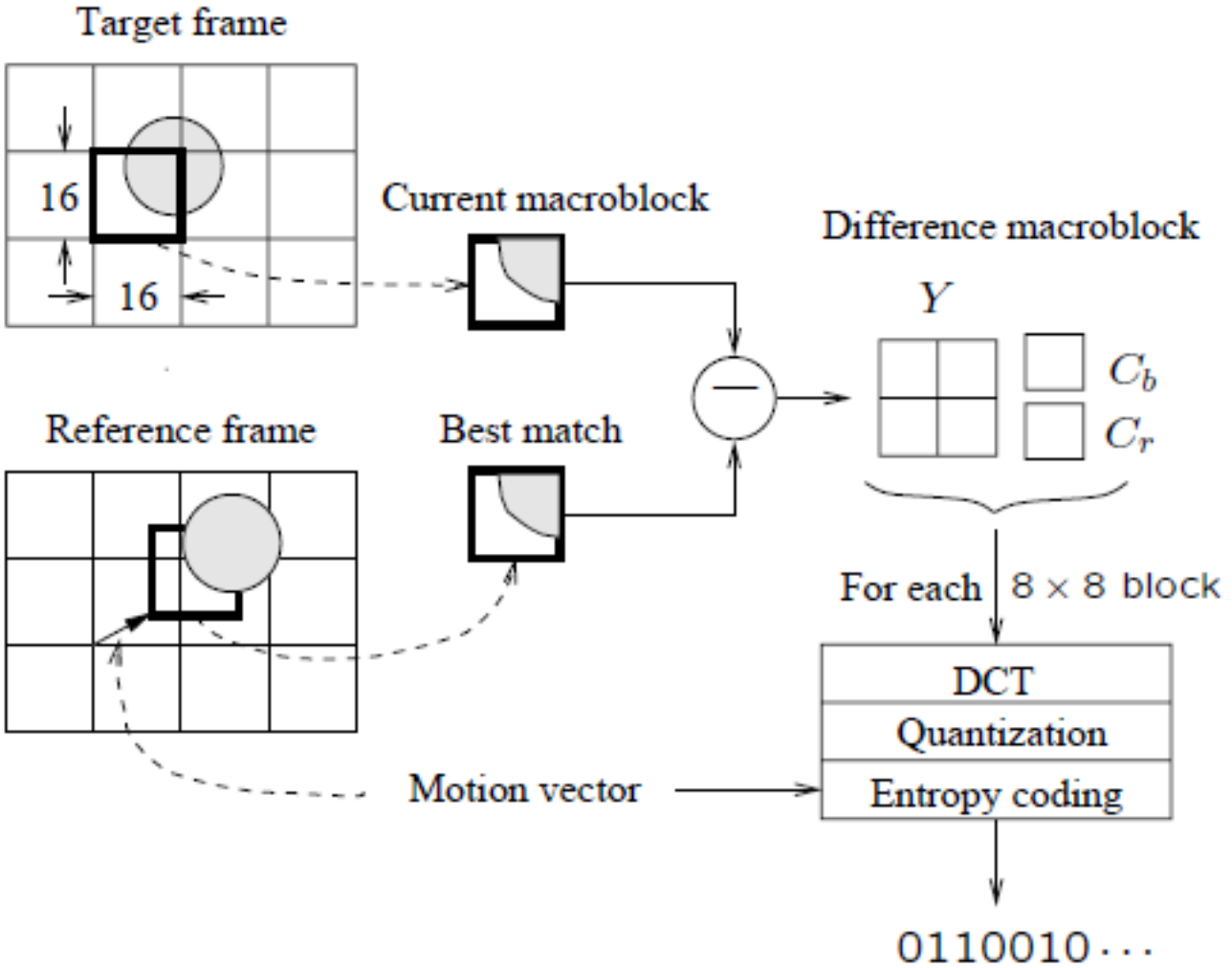
P-frame Coding

- For each 16×16 macroblock in the target frame, a *motion vector* is allocated
- The *difference* between the target macroblock and its best match in the *decoded* reference frame is computed to measure the *prediction error*
- The *difference* macroblock is coded with DCT, zigzag quantization, and entropy coding



P-frame Coding

- The motion vector is not coded directly, instead the *difference* between the current $MV_{current}$ and the MV of the previous macroblock $MV_{previous}$ is coded: $MVD = MV_{previous} - MV_{current}$
- If the *prediction error* is too large, the target macroblock is coded directly. This is called a non-motion compensated macroblock



H.261 Quantization

- Unlike JPEG which uses a *quantization table*, H.261 uses a constant *quantization step* for all DCT coefficients
- The DC coefficient:

$$QDCT = round\left(\frac{DCT}{8}\right)$$

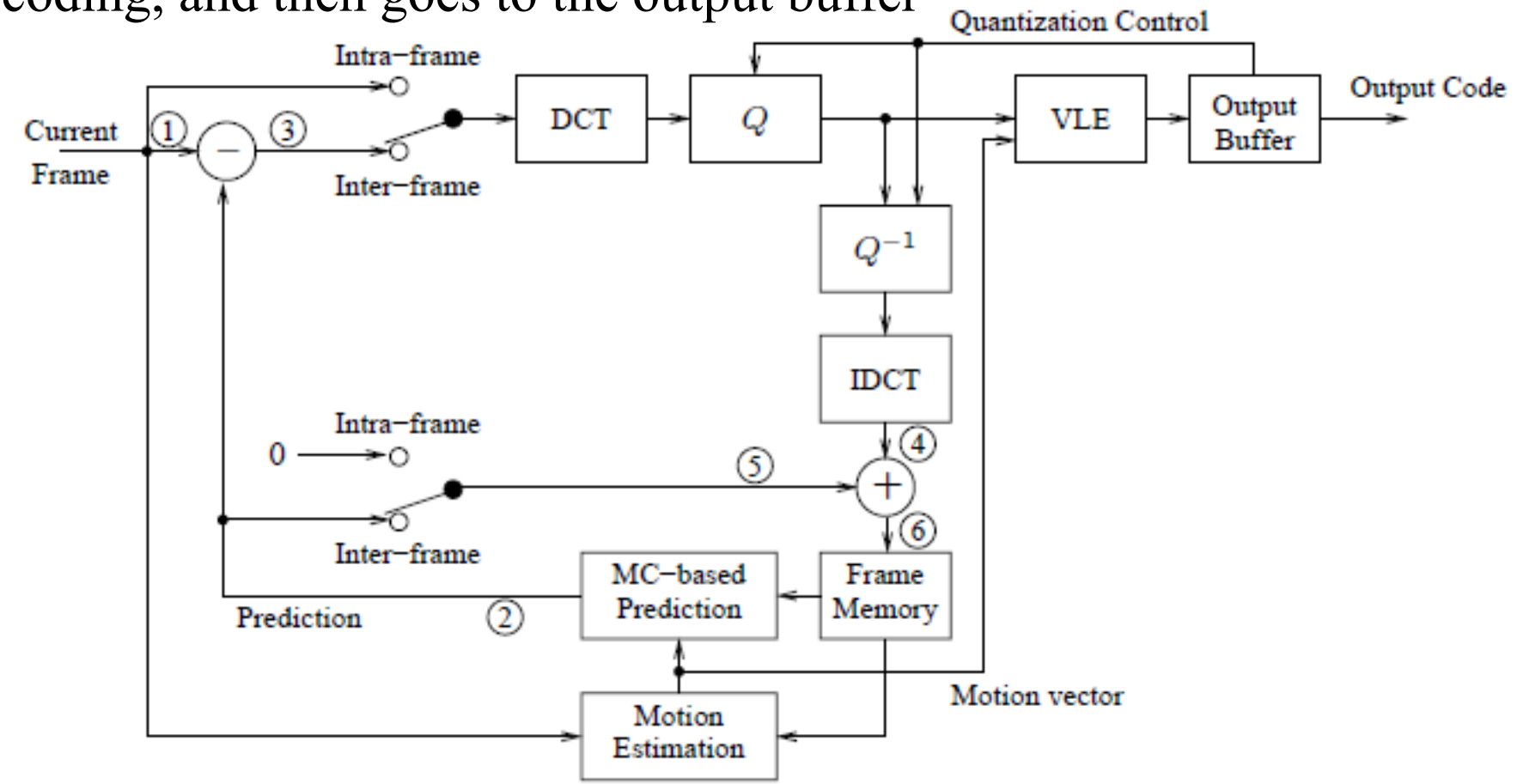
- The AC coefficients:

$$QDCT = \left\lfloor \frac{DCT}{2 \times scale} \right\rfloor$$

where *scale* is an integer in the range [1, 31] and defines the *quality* or the *bit rate* of the video i.e. higher scale means higher quantization step and lower quality and bit rate

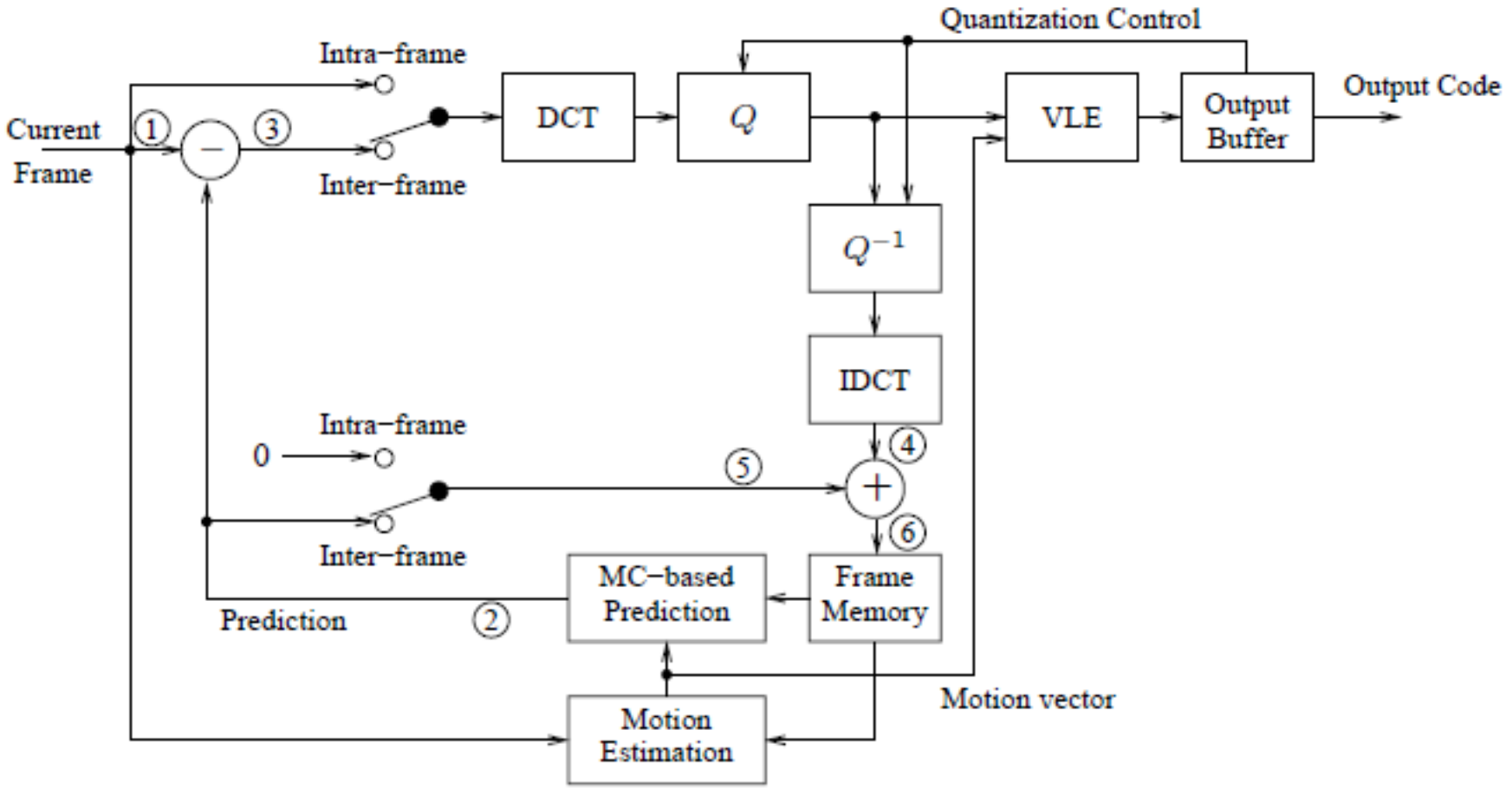
H.261 Encoder

- Assume we are encoding three frames: I, P_1, P_2
- The I-frame comes at (1)
- There is no prediction, so (2) is zero
- The I-frame goes at (3) and goes through DCT, quantization, entropy coding, and then goes to the output buffer



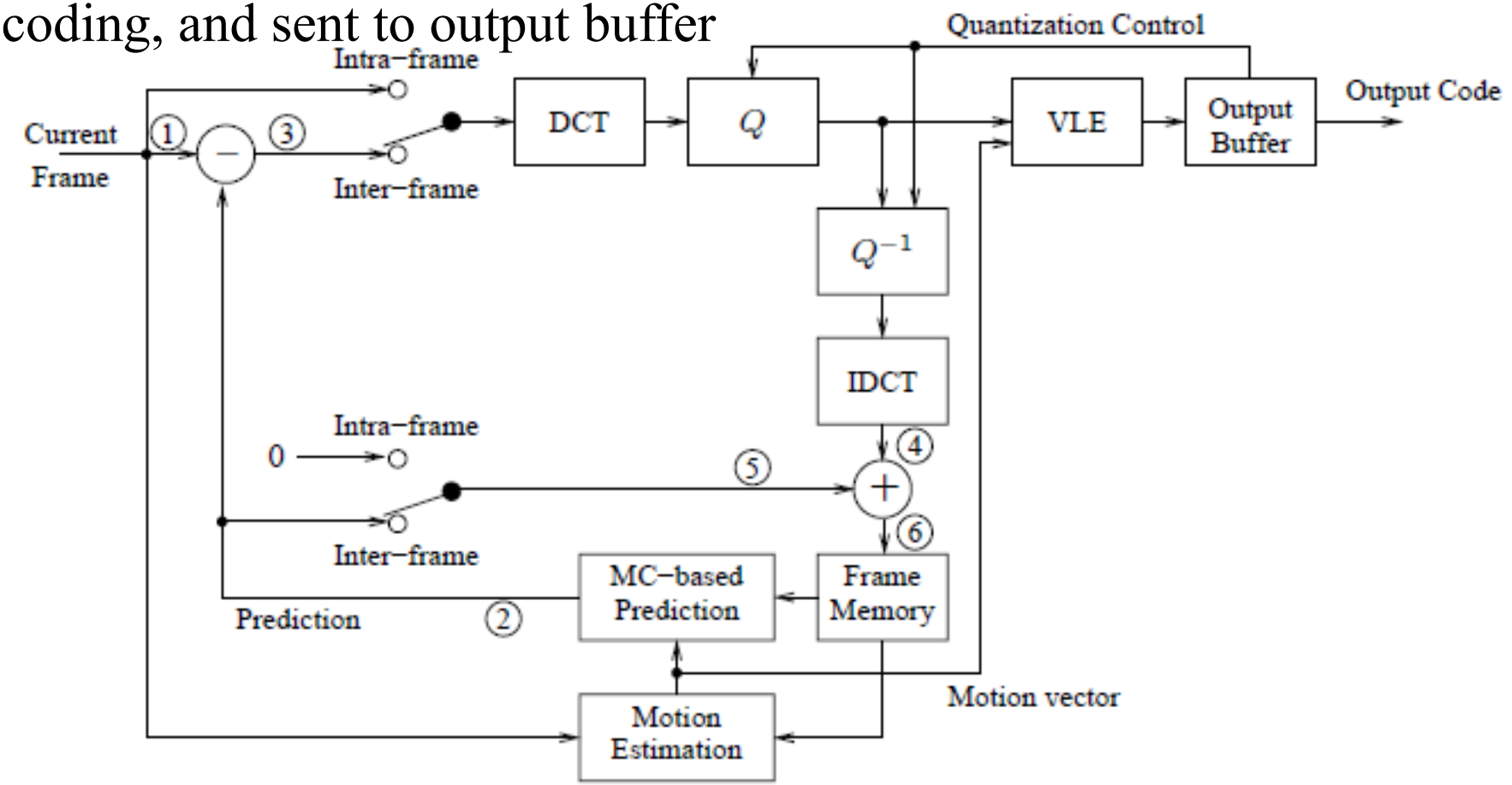
H.261 Encoder

- It also goes through inverse Quantization and IDCT and reaches (4)
- It is added to zero at (5) and *reconstructed* at (6) which is to be stored in the frame memory
- Frames in the frame memory will be used for the motion estimation of the next frames



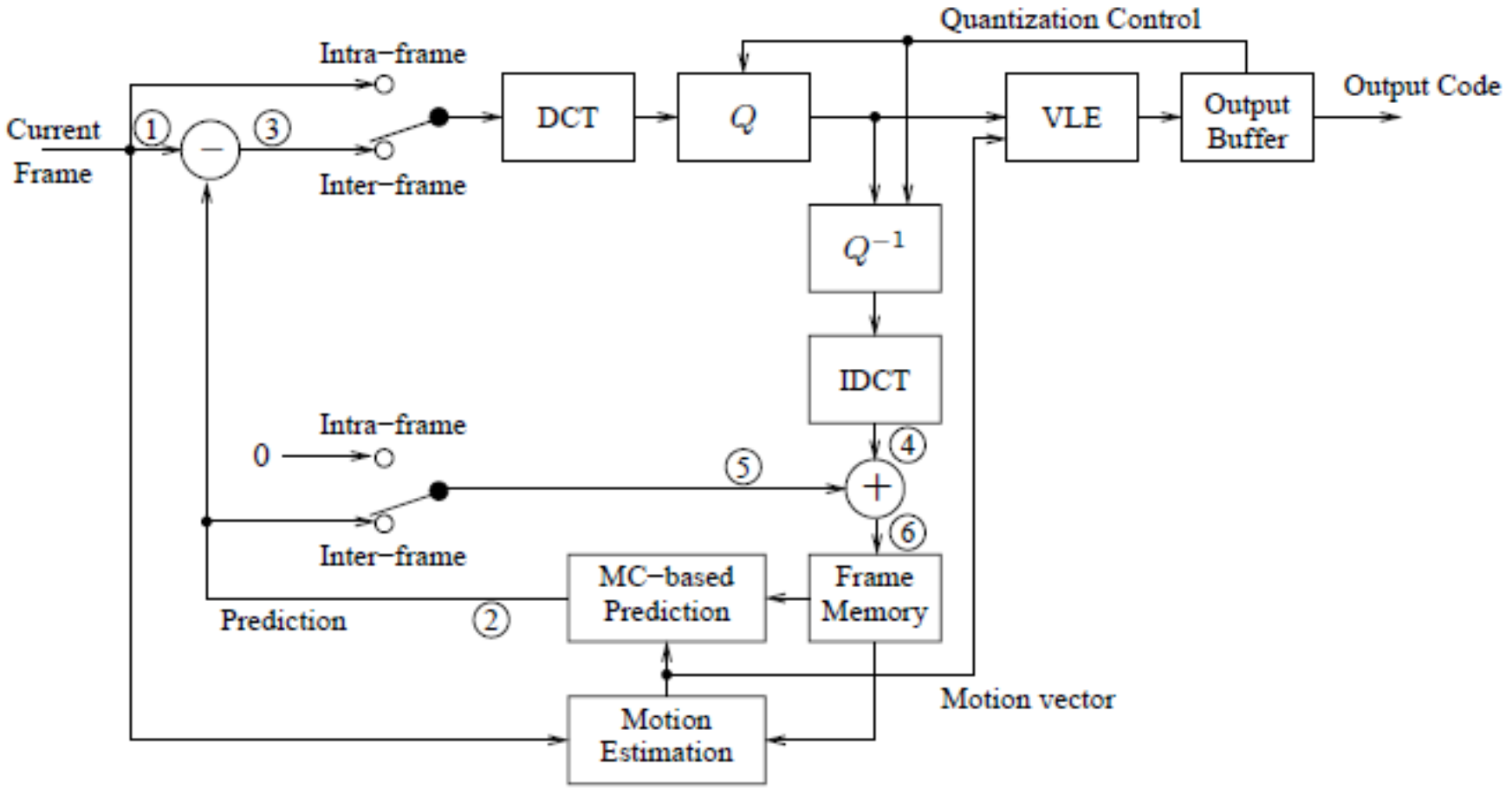
H.261 Encoder

- The first P-frame P_1 arrives at (1)
- Motion compensation is applied to macroblocks of P_1 by searching the *decoded* I-frame at (2)
- The difference between P_1 and its motion-compensated blocks is computed at (3), which is then sent to DCT, quantization, entropy coding, and sent to output buffer



H.261 Encoder

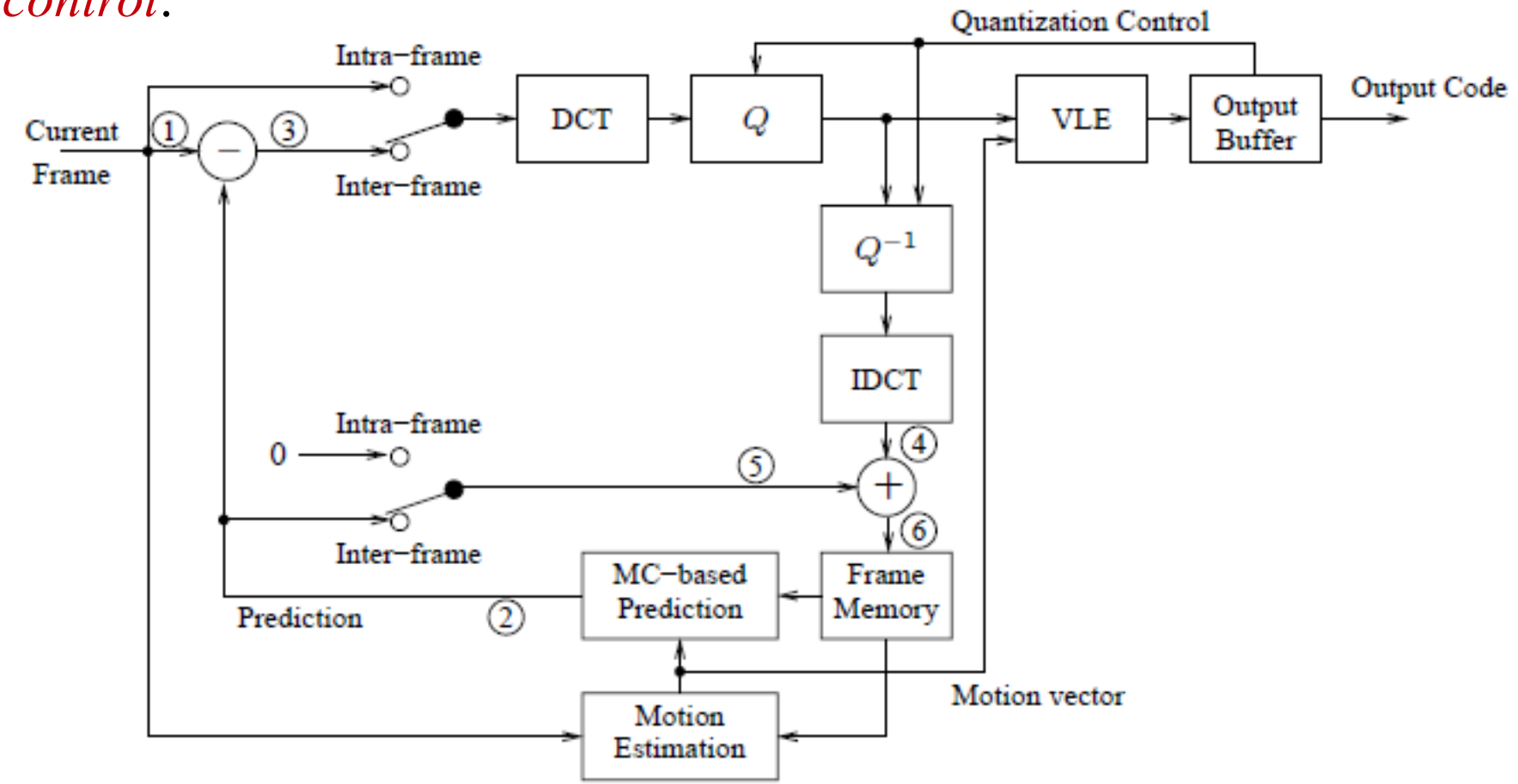
- It is also sent to inverse quantization and IDCT to reach (4)
- It is added to the *decoded* reference frame at (5) (with motion estimation) to reconstruct the P_1 frame at (6)
- The decoded frame is stored in the frame memory for future matching



H.261 Encoder

- The same is done to P_2 but with using *decoded* version of P_1 for motion estimation and compensation

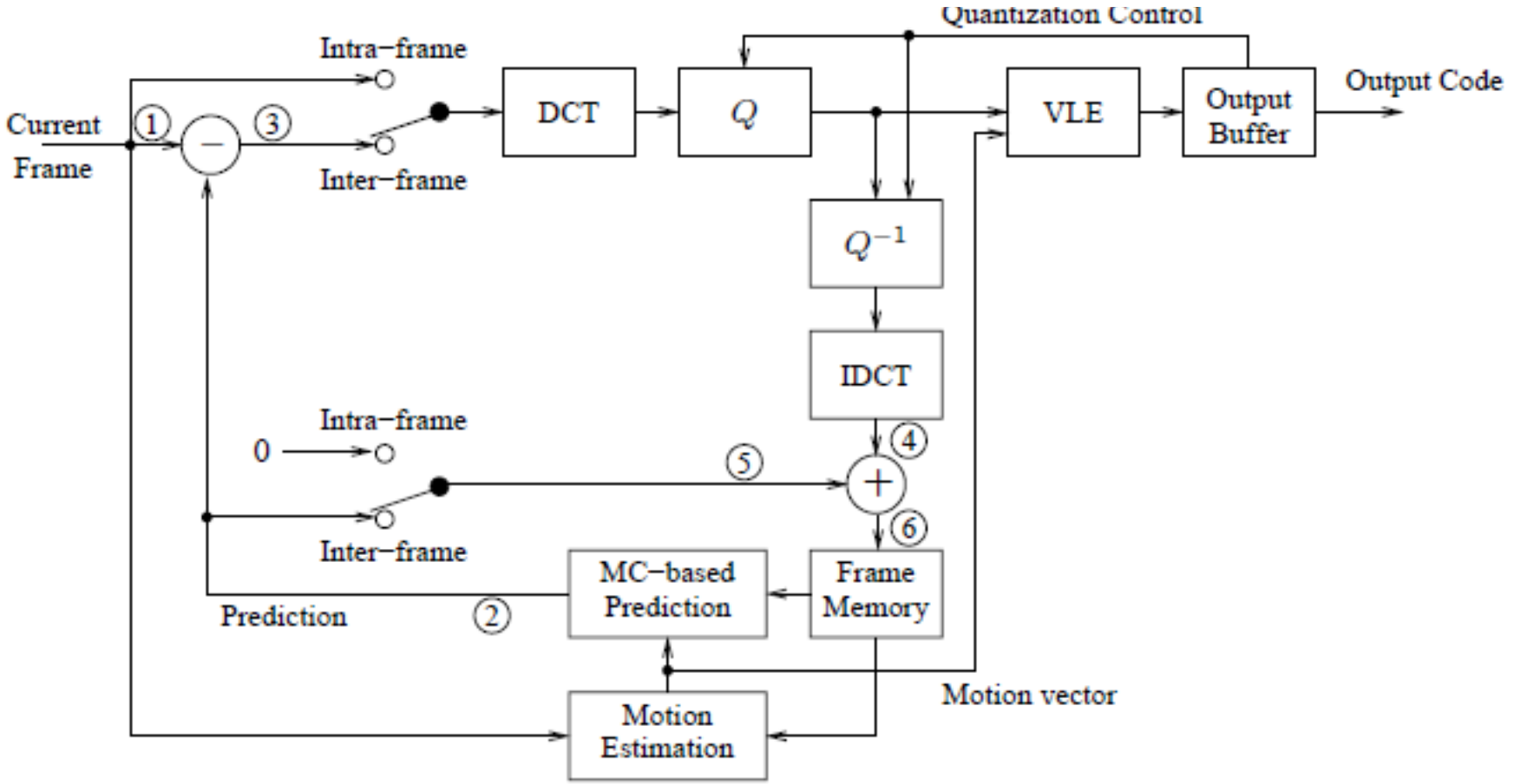
- When the *output buffer* is full, it adjusts the *quantization step* by making it larger to reduce the bit rate. This is called *encoding rate control*.



H.261 Encoder

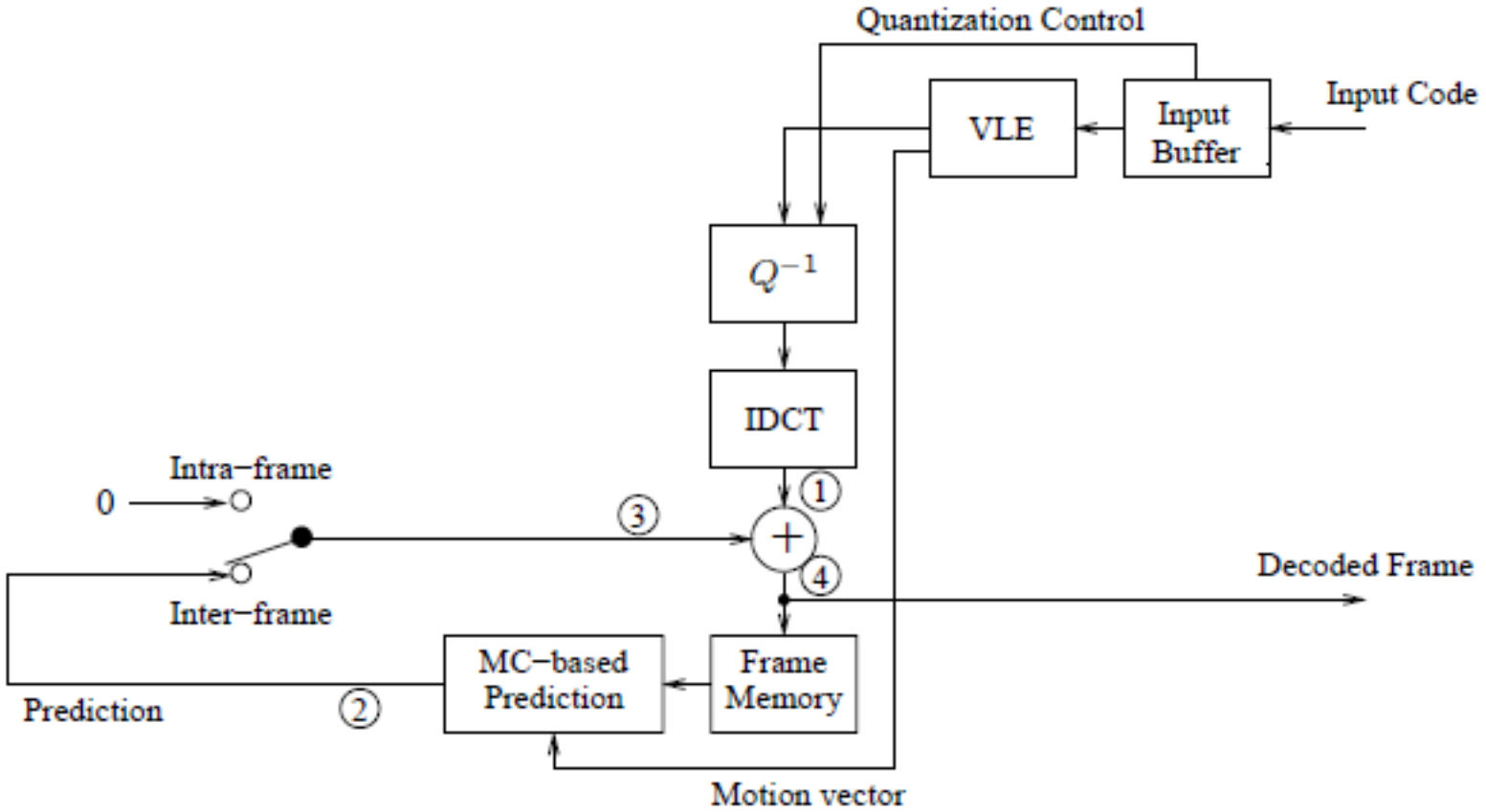
Table 10.3: Data Flow at the Observation Points in H.261 Encoder

Current Frame	Observation Point					
	1	2	3	4	5	6
I	I			$\tilde{I}$	0	$\tilde{I}$
P_1	P_1	P_1'	D_1	$\tilde{D}_1$	P_1'	$\tilde{P}_1$
P_2	P_2	P_2'	D_2	$\tilde{D}_2$	P_2'	$\tilde{P}_2$



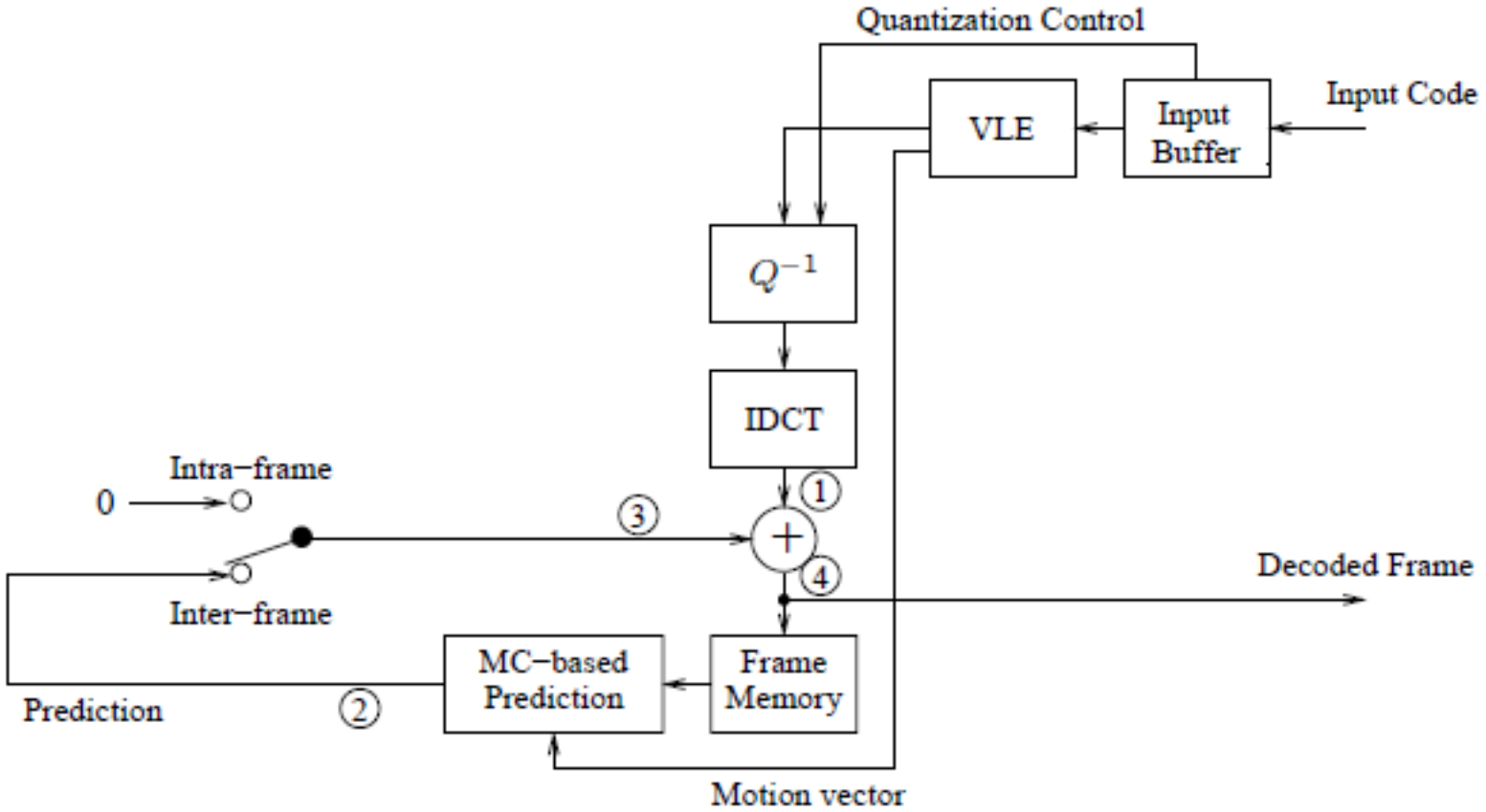
H.261 Decoder

- The encoded bit stream enters the *input buffer*
- The content of the I-frame I undergoes the entropy decoding, inverse quantization, and IDCT to reach point (1)
- It is added to the prediction of zero at (3) to produce the *decoded* I-frame at point (4), which is also stored in the frame memory to be used for the motion compensation of the following frames



H.261 Decoder

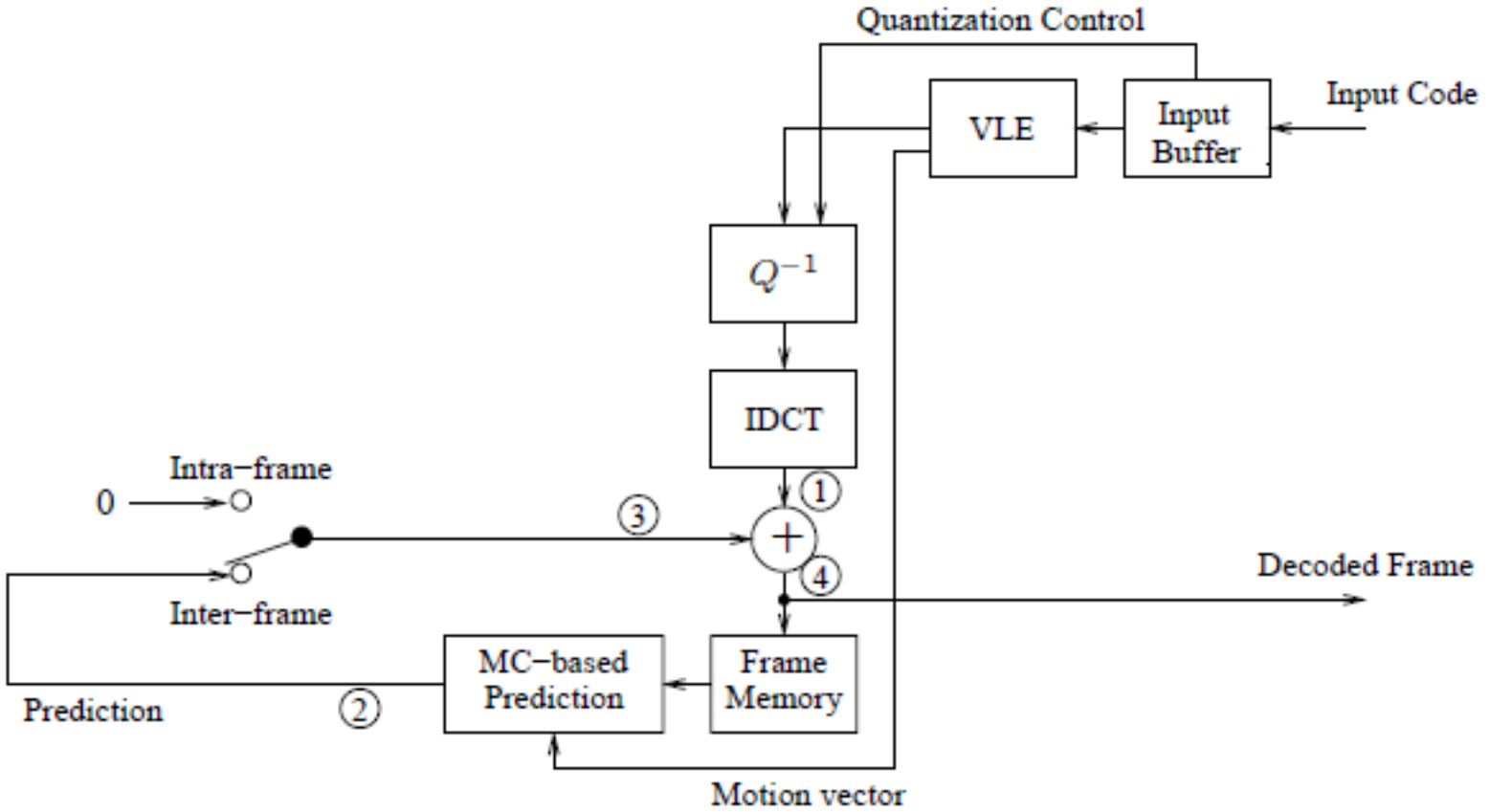
- The content of the P-frame P_1 (difference blocks and motion vectors) undergoes the entropy decoding, inverse quantization, and IDCT to reach point (1)
- It is added to the motion compensated prediction of the previous frame at (3) to produce the *decoded* P-frame at point (4), which is also stored in the frame memory to be used for motion compensation



H.261 Decoder

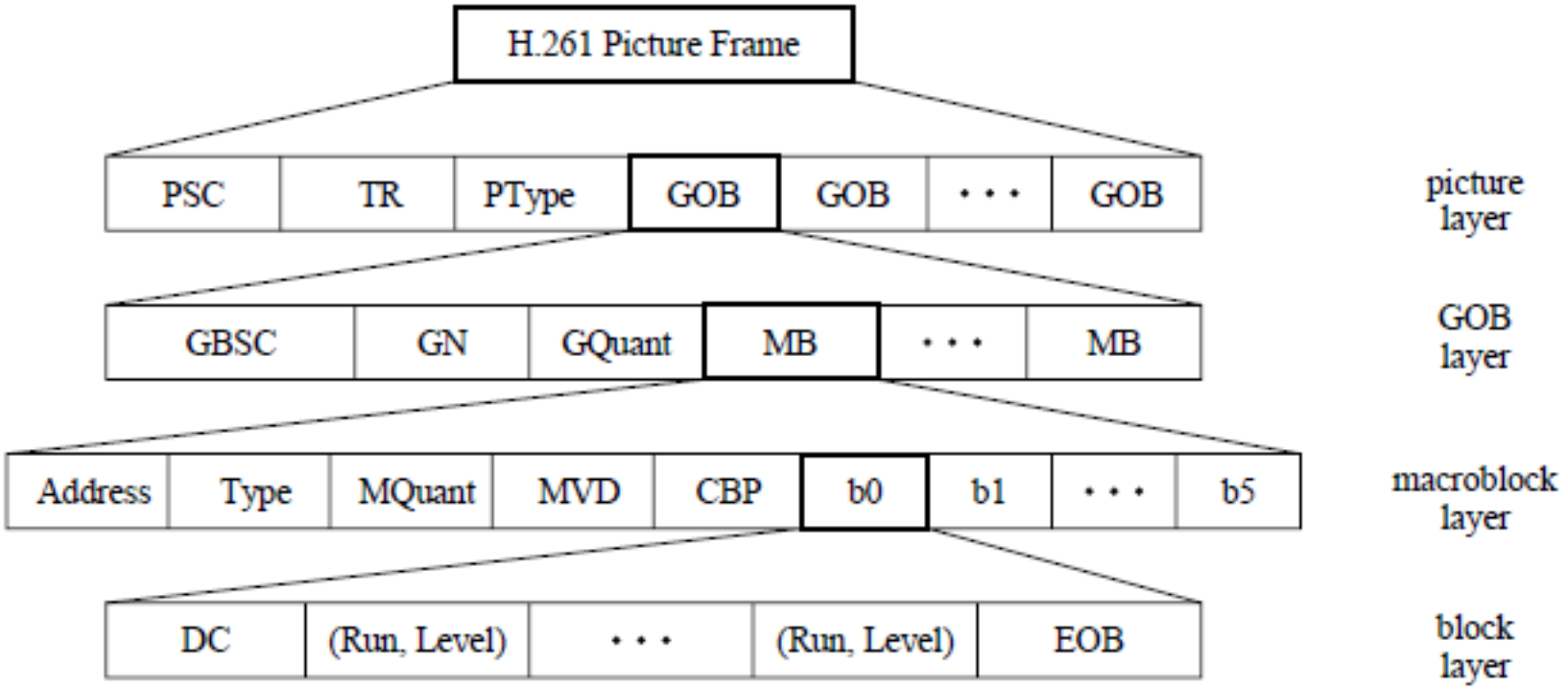
Table 10.4: Data Flow at the Observation Points in H.261 Decoder

Current Frame	Observation Point			
	1	2	3	4
I	$\tilde{I}$		0	$\tilde{I}$
P_1	$\tilde{D}_1$	P_1'	P_1'	$\tilde{P}_1$
P_2	$\tilde{D}_2$	P_2'	P_2'	$\tilde{P}_2$



H.261 Bitstream

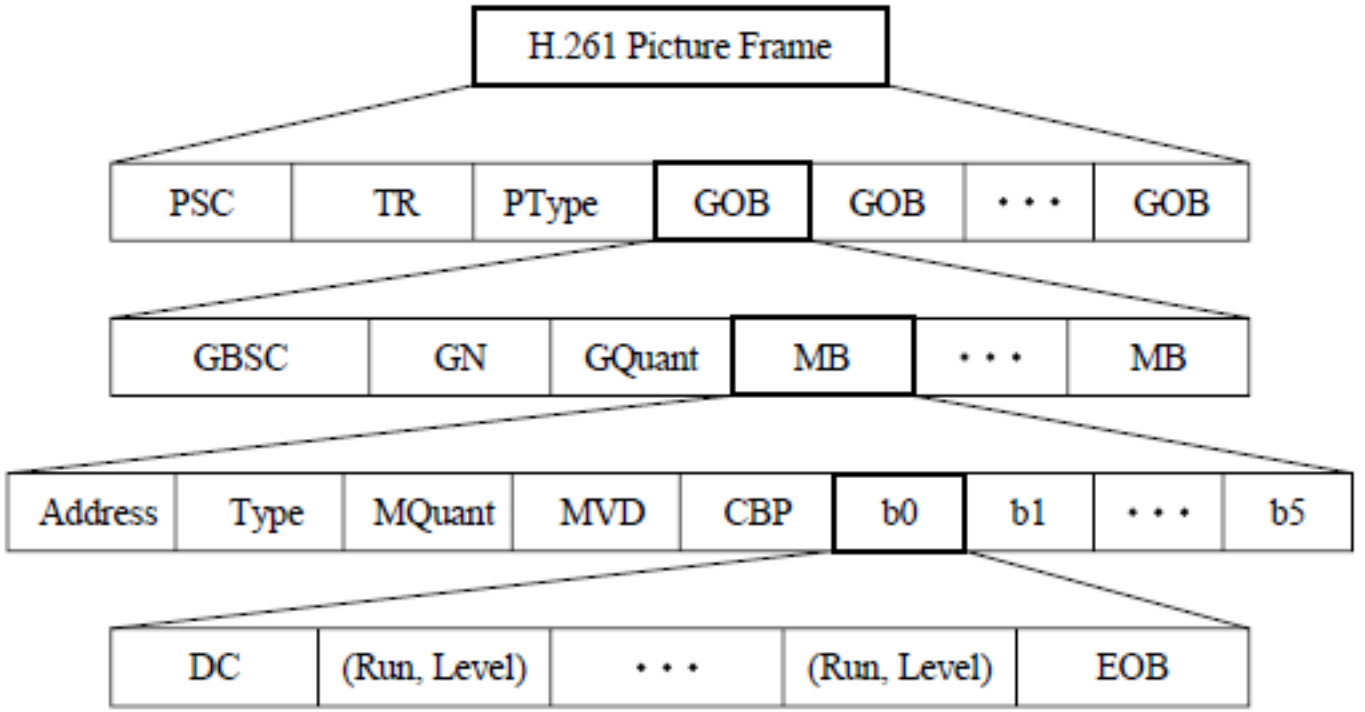
- Four layers: Picture, Group of Blocks (GOB), Macroblock, and Block layers



PSC:	Picture Start Code	TR:	Temporal Reference
PType:	Picture Type	GOB:	Group of Blocks
GBSC:	GOB Start Code	GN:	Group Number
GQuant:	GOB Quantizer	MB:	Macro Block
MQuant:	MB Quantizer	MVD:	Motion Vector Data
CBP:	Coded Block Pattern	EOB:	End of Block

H.261 Bitstream: Picture Layer

- Picture Start Code (PSC) delineates boundaries between pictures
- Temporal Reference (TR) provides a timestamp for the picture
- Picture Type (PType) specifies the type of the picture e.g. CIF or QCIF



H.261 Bitstream: GOB Layer

- Pictures are divided into regions of 11×3 macroblocks (regions of 176×38 pixels) each is called a Group of Blocks
- CIF contains 2×6 = 12 GOBs (352×288 pixels) and QCIF contains 1×3 = 3GOBs (176×144 pixels)

GOB 0
GOB 1
GOB 2

QCIF

GOB 0	GOB 1
GOB 2	GOB 3
GOB 4	GOB 5
GOB 6	GOB 7
GOB 8	GOB 9
GOB 10	GOB 11

CIF

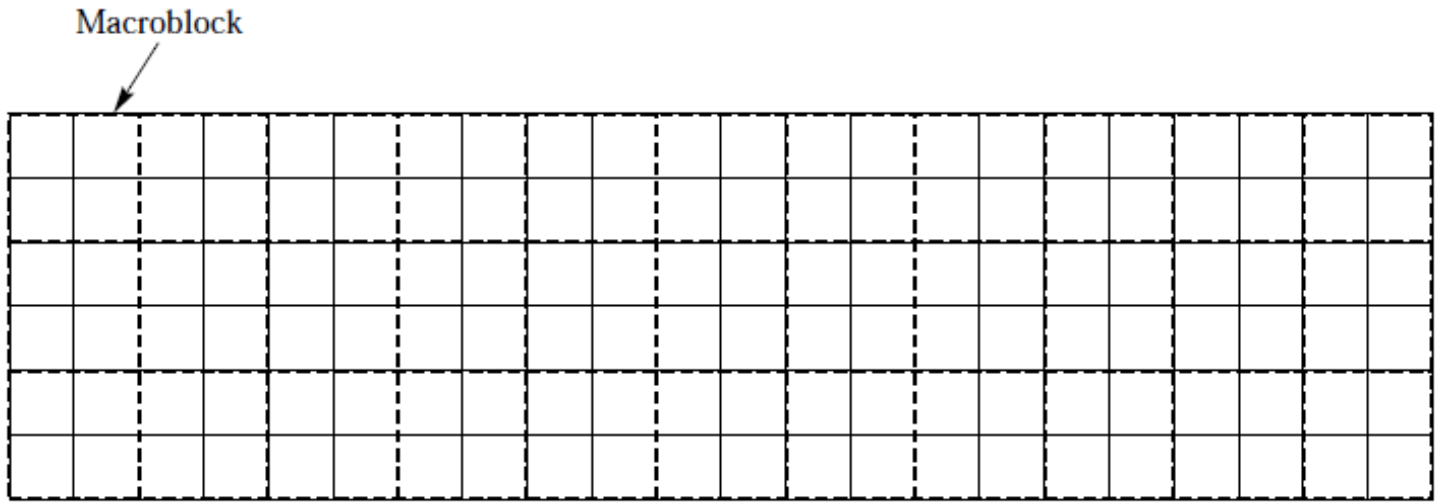
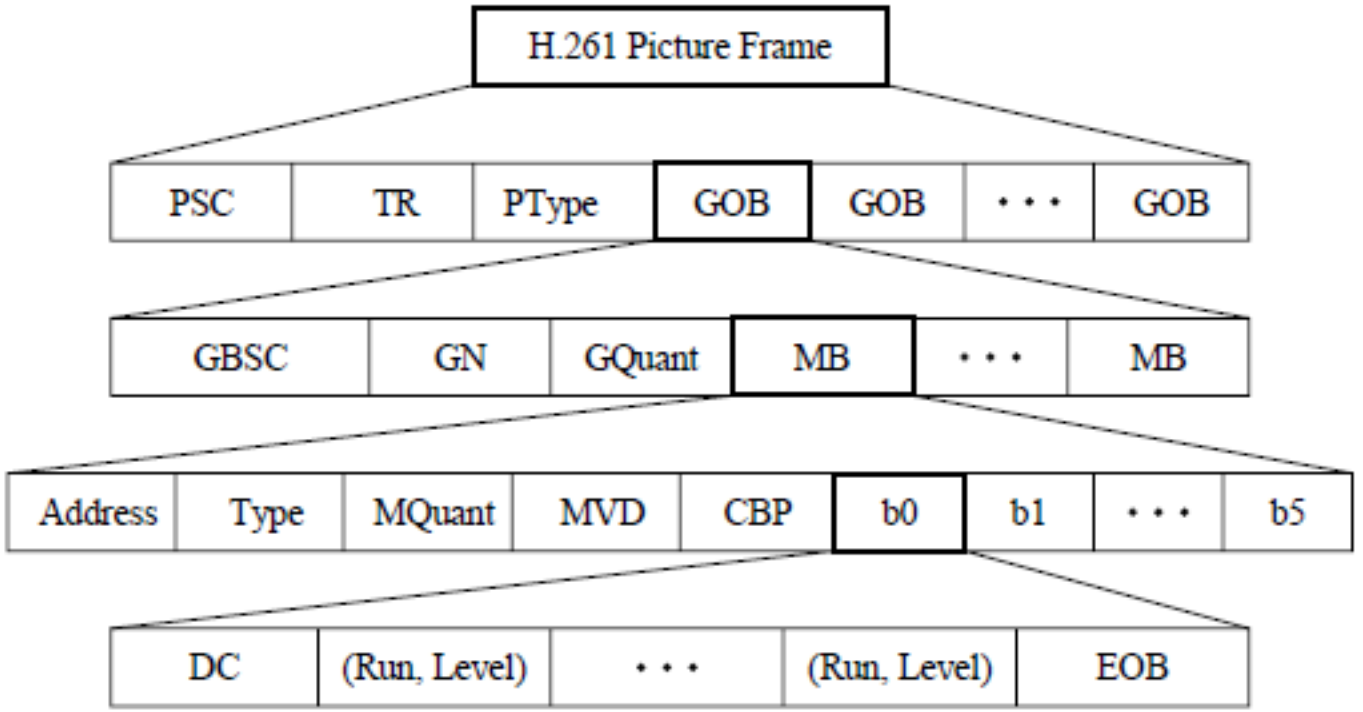


FIGURE 18. 12 **A GOB consisting of 33 macroblocks.**

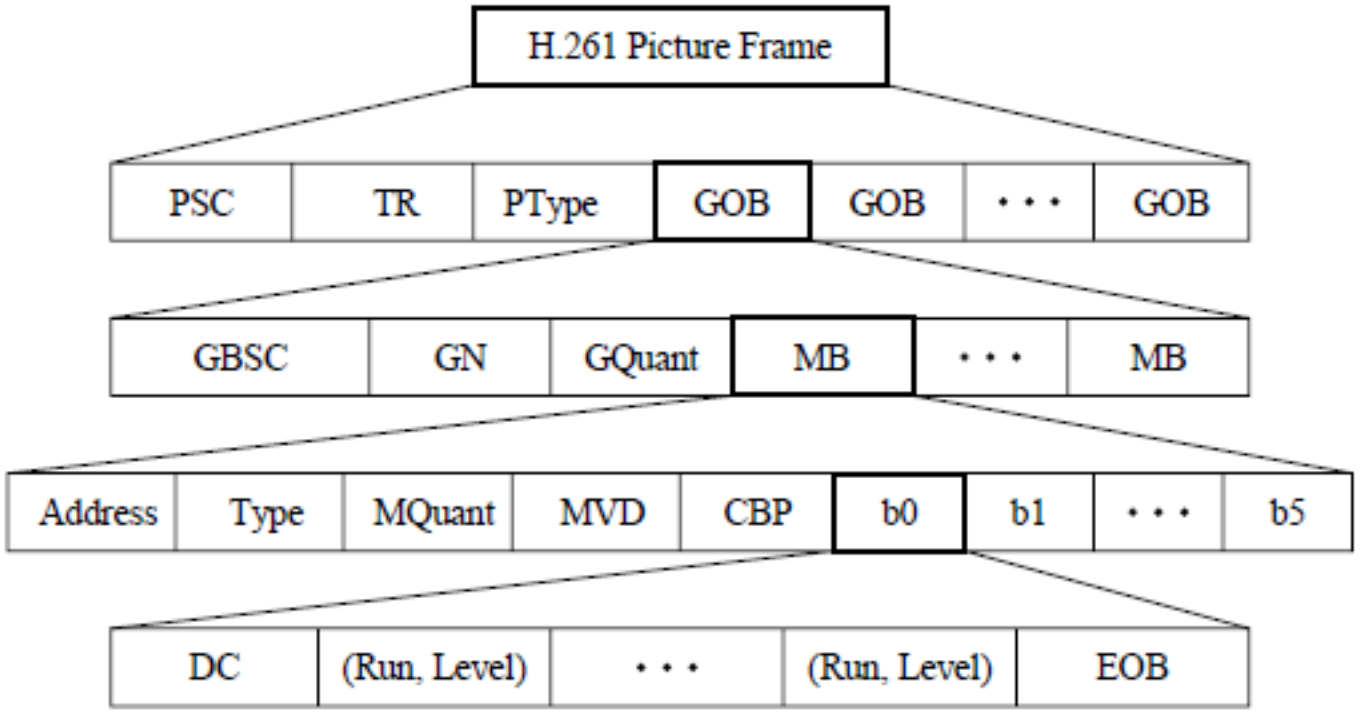
H.261 Bitstream: GOB Layer

- Each GOB has its Start Code (GBSC) and Group Number (GN)
- GQuant specifies the *scale* used for the quantization
- If some *errors* are introduced, the decoder can recover from the first identifiable GOB



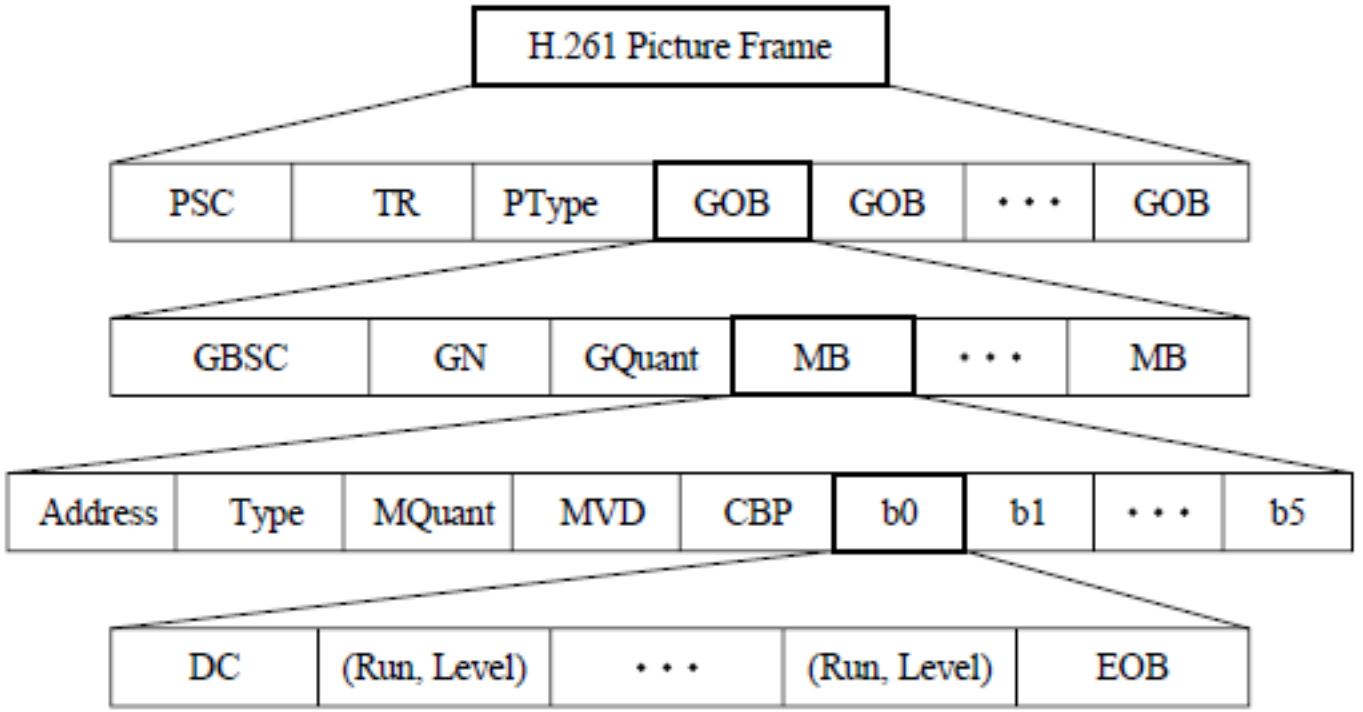
H.261 Bitstream: MB Layer

- Each Macroblock (MB) has its Address indicating its position in the GOB, its own quantizer (MQuant) and six 8×8 image blocks (4 *Y*, 1 *Cb* and 1*Cr*)
- Type indicates whether it's inter- or intra-frame, motion-compensated or non-motion-compensated MB
- MVD is the motion vector data for this MB



H.261 Bitstream: Block Layer

- Contains the DCT coefficients for each block
- Starts with the DC coefficient
- Followed by run-length coded AC coefficients (run, level) pairs which indicates a zero run followed by a non-zero value
- *run* is in the range [0, 63] and *level* [–127, 127] – {0}



H.261 Summary

- Two types of frames:
  - I-frames
  - P-frames
- JPEG-like compression in I-frames
- Motion compensation in P-frame

H.263

- An improved video coding standard for video conferencing and other audiovisual services transmitted on Public Switched Telephone Networks (PSTN).
- Uses predictive coding for inter-frames to reduce temporal redundancy and transform coding for the remaining signal to reduce spatial redundancy (for both Intra-frames and inter-frame prediction).
- Adopted by the ITU-T in 1995

H.263 Video Formats

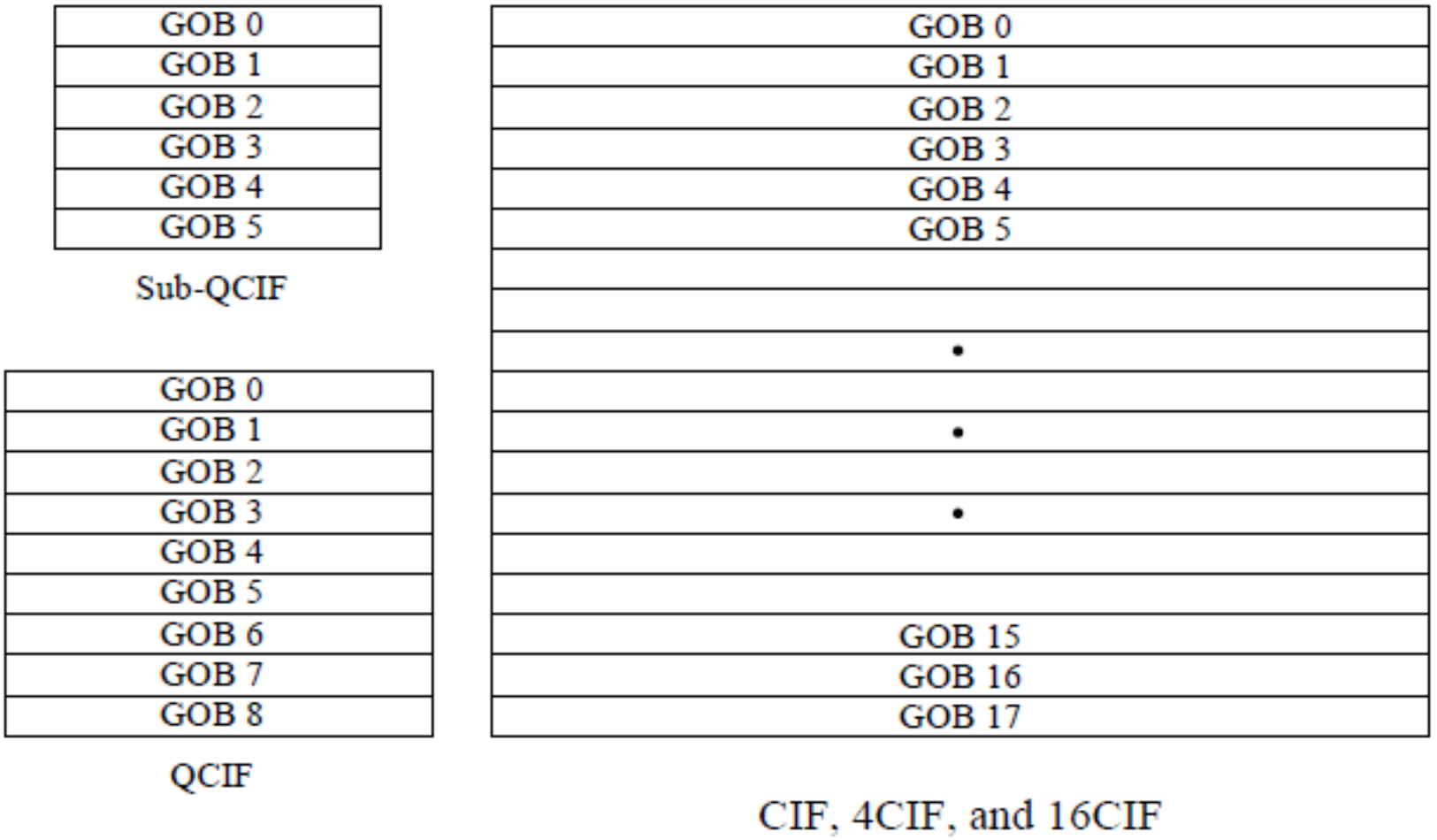
- Supports more video formats than H.261

Table 10.5 Video Formats Supported by H.263

Video format	Luminance image resolution	Chrominance image resolution	Bit-rate (Mbps) (If 30 fps and uncompressed)	Bit-rate (kbps) BPPmaxKb (compressed)
sub-QCIF	128 × 96	64 × 48	4.4	64
QCIF	176 × 144	88 × 72	9.1	64
CIF	352 × 288	176 × 144	36.5	256
4CIF	704 × 576	352 × 288	146.0	512
16CIF	1,408 × 1,152	704 × 576	583.9	1024

H.263 GOBs

- Supports Group of Blocks, but unlike H.261, they don't have a fixed size
- They always start and stop at the left and right boundary



H.263 Motion Compensation

- The horizontal and vertical components of the motion vector MV are predicted from the median values of the horizontal and vertical components, respectively, of MV1, MV2, MV3 from the *previous*, *above* and *above and right* MBs.
- For the macroblock with MV(u, v):

$$u_p = median(u_1, u_2, u_3)$$
$$v_p = median(v_1, v_2, v_3)$$

- The difference DMV is coded:

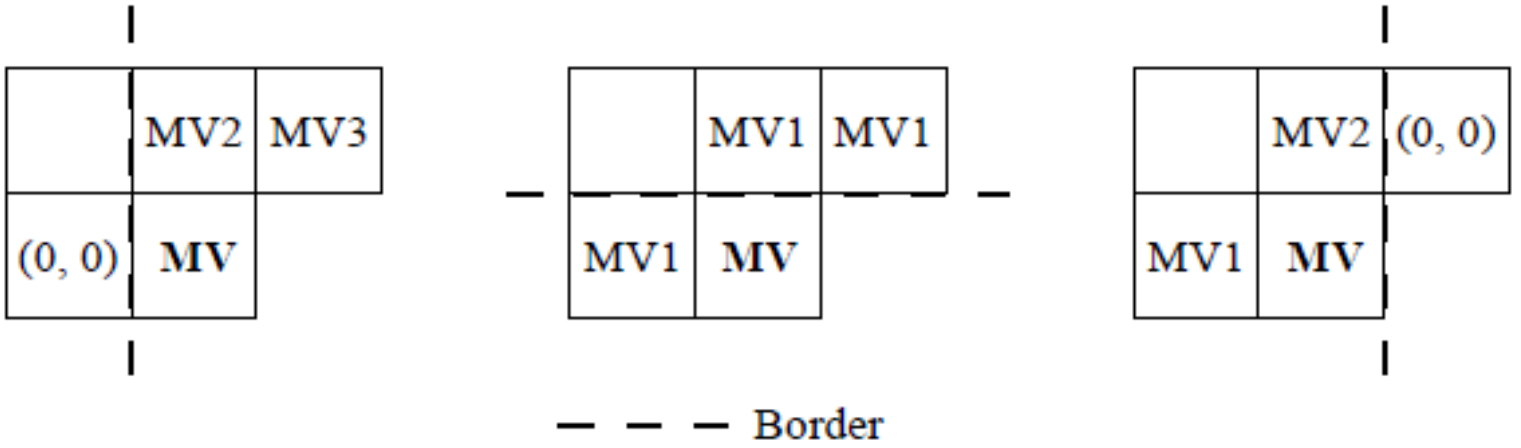
$$DMV = (u - u_p, v - v_p)$$

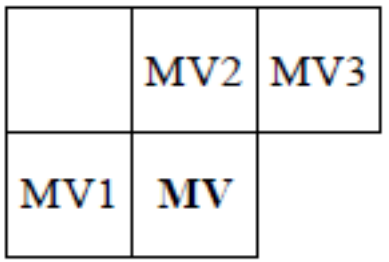
	MV2	MV3
MV1	**MV**	

MV	Current motion vector
MV1	Previous motion vector
MV2	Above motion vector
MV3	Above and right motion vector

H.263 Motion Compensation

- The MV prediction at the *border* is handled in a special way

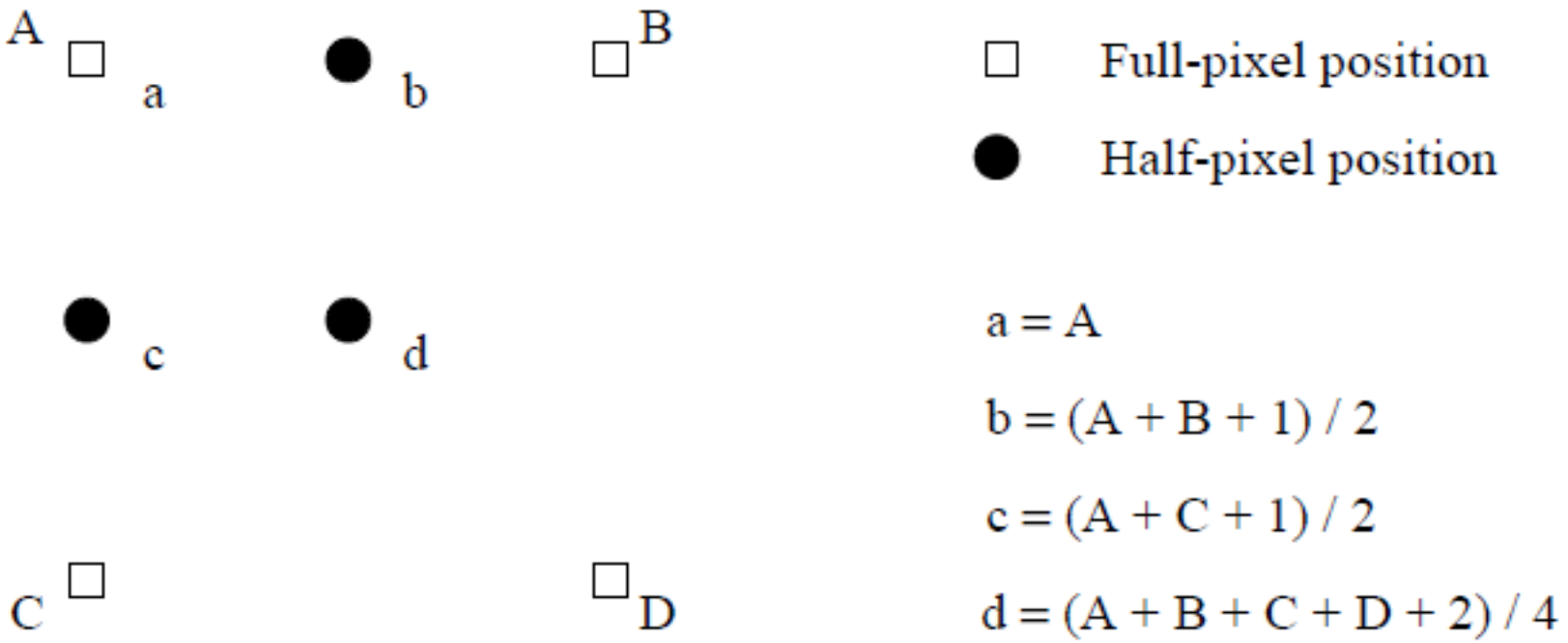




MV Current motion vector
MV1 Previous motion vector
MV2 Above motion vector
MV3 Above and right motion vector

H.263 Motion Compensation

- To improve accuracy, *fractional* motion estimation is used where the motion vector have *half-pixel* precision
- The default range for the vertical and horizontal component of the MV is [–16, 15.5]
- The pixel values needed at half-pixel positions are obtained using *linear interpolation*



H.263 Optional Modes

- H.263 has some other optional operating *modes*, influenced by the MPEG standards:
  - Unrestricted MV Mode
  - Advanced Prediction Mode
  - PB-frames Mode

PB-frames

- A PB-frame consists of two pictures coded together:
  - A P-frame coded from the previous I- or P-frame
  - a B-frame (bidirectional frame) coded using *both* the previous I-frame and the current P-frame
- Bi-directional motion compensation yields better prediction and compression results, and is used in the MPEG standard

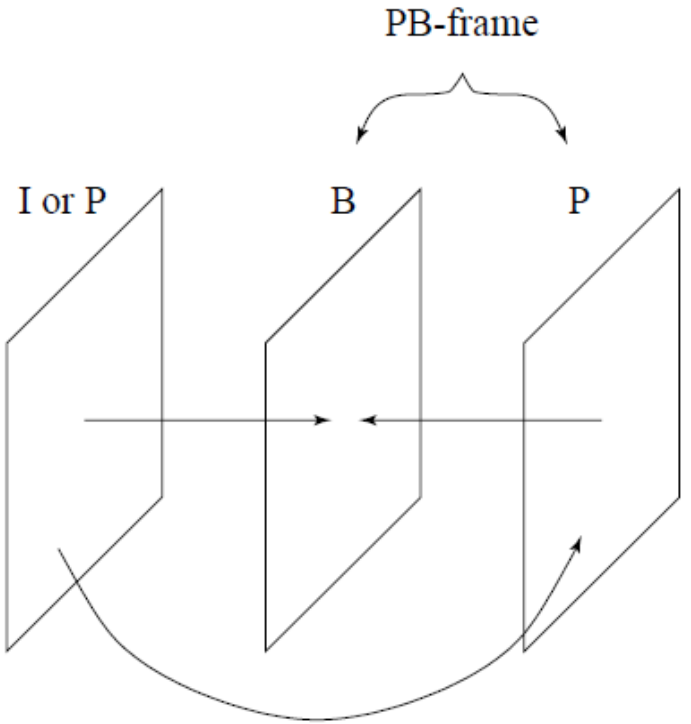


Fig. 10.13: A PB-frame in H.263.

Further Improvements

- H.263+ and H.263++ added more improvements and enhancements to the H.263 standard
  - Redefines the unrestricted MV mode
  - Defines a new *slice* structure like that in MPEG
  - Improved PB-frame mode
  - …

Recap

- Video Compression
- Motion Compensation
- H.261
- H.263

- Next: Video Compression II

- More information: **FM** Ch. 10, **IDC** Ch. 18.