Video Compression

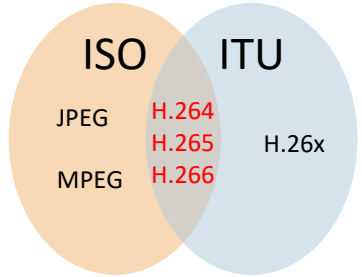
INSTRUCTOR: YAN-TSUNG PENG

DEPT. OF COMPUTER SCIENCE, NCCU

CLASS 4

Video Coding Standards: JPEG and MPEG

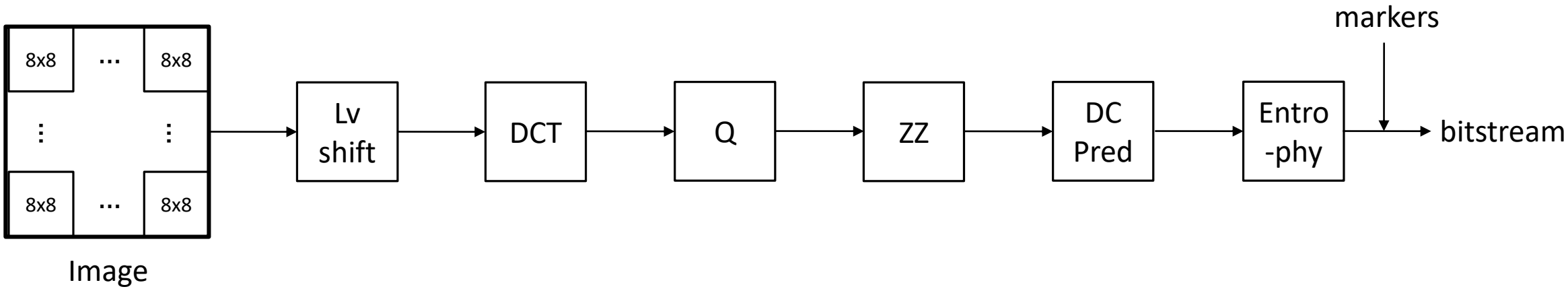
- Major video CODECs (encoder and decoder) used today conform to coding standards enacted by International Standards Organization (ISO) and the International Telecommunications Union (ITU).
  - ex: **H.264** (http://www.staroceans.org/e-book/ISO-14496-10.pdf)

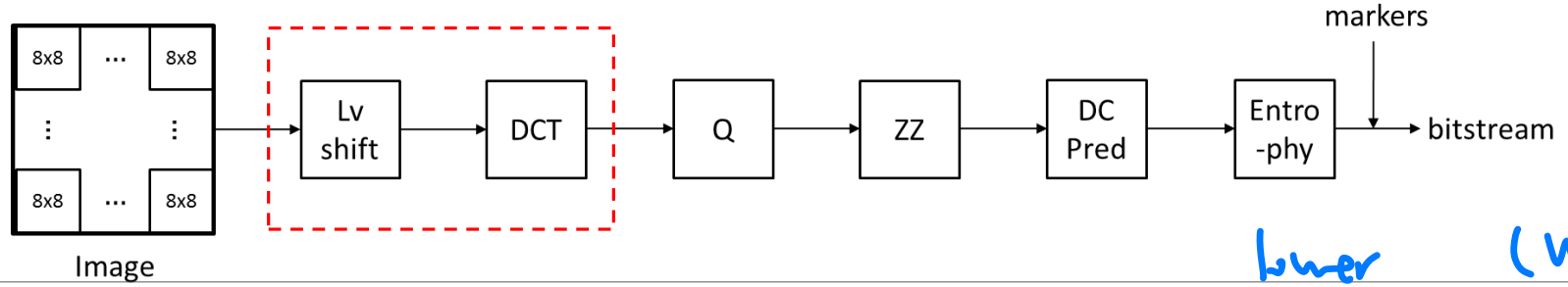


- Video codec development (ITU: H.26x, ISO: MPEG-x)
  - H.261 (1990): video telephony over CBR channels
  - JPEG (1992): still image compression for storage purposes
  - MPEG-1 (1993): compression for video and audio on storage purposes
  - H.263 (1995): video telephony over circuit- and packet-switched networks, supporting low to high bit rates
  - MPEG-2 (1995): compression and transmission for storage and broadcast applications
  - MPEG-4 (1998), H.263+ (1998), H.263++ (2001), H.264 (2003)
  - H.265 (2013), aka High Efficiency Video Coding (HEVC), developed by the Joint Collaborative Team on Video Coding (JCT-VC), experts from ITU and ISO/IEC Moving Picture Experts Group (MPEG), improving upon its predecessor, H.264
  - H.266 (2020), aka Versatile Video Coding (VVC), developed by the Joint Video Experts Team (JVET) from ITU and ISO/IEC collaboration

JPEG

- JPEG standard is released in 1992, providing a syntax and decoding process for still images.
- Baseline (Y, Cb, Cr)





lower (u,v) higher

smooth transformation

sharp, fine details

- Level shift: the input image is normalized to [-128, 127] (8-bit)
- DCT: discrete cosine transform
  - for an 8x8 block, there are 64 DCT coefficients.

domain

spatial ↓ freq.

freq. ↓ spatial

- Forward Transform (for N x N blocks)

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

↳ DCT coefficient ↳ pixel value ↳ block size

- Inverse Transform (for N x N blocks)

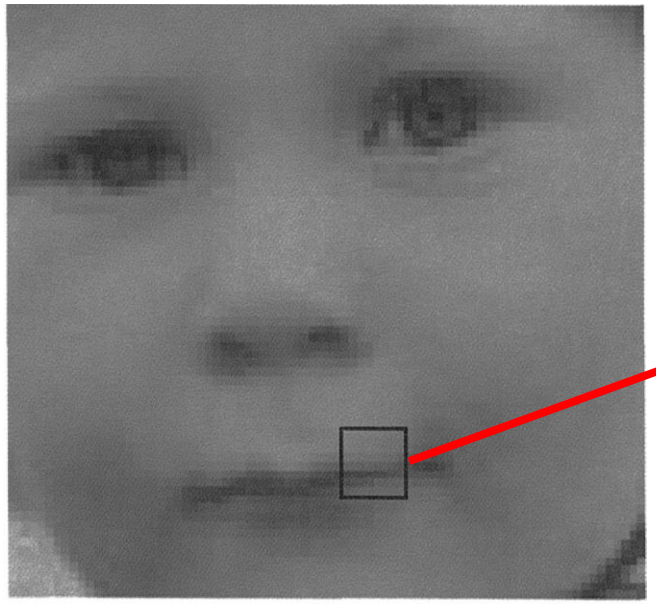
$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N}$$

$$u, v = 0, 1, \ldots N-1$$

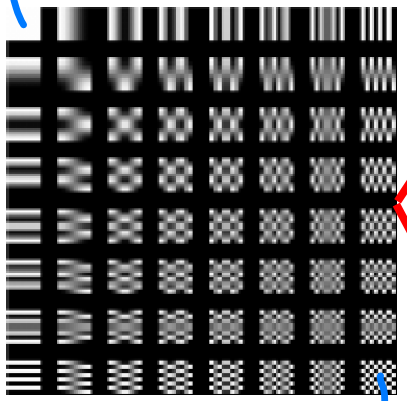
$$C(t) = \begin{cases} \frac{2}{\sqrt{N}}, & t = 0 \\ 2 \cdot \sqrt{\frac{2}{N}}, & t \neq 0 \end{cases}$$

$$x, y = 0, 1, \ldots N-1$$

Example of DCT



8x8 DCT



DC component

high frequency

DCT coefficients

967.5	-6.3	-1				1.3	0.1
-164.4	-71.3	1				0.2	-1.6
55.3	13.6	-				0.7	-0.6
81.8	-8.9	-				0.1	1
38.9	-12.9	-				0.3	1
-7.6	-8.4	0				0.1	2.1
-14.7	1.4	4				0.2	1.1
-22.3	5	4				0.2	-0.3

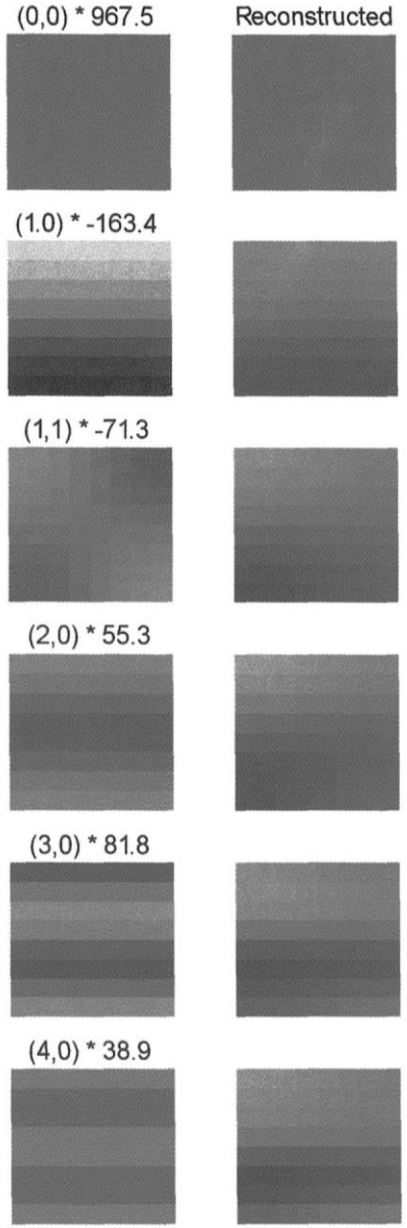
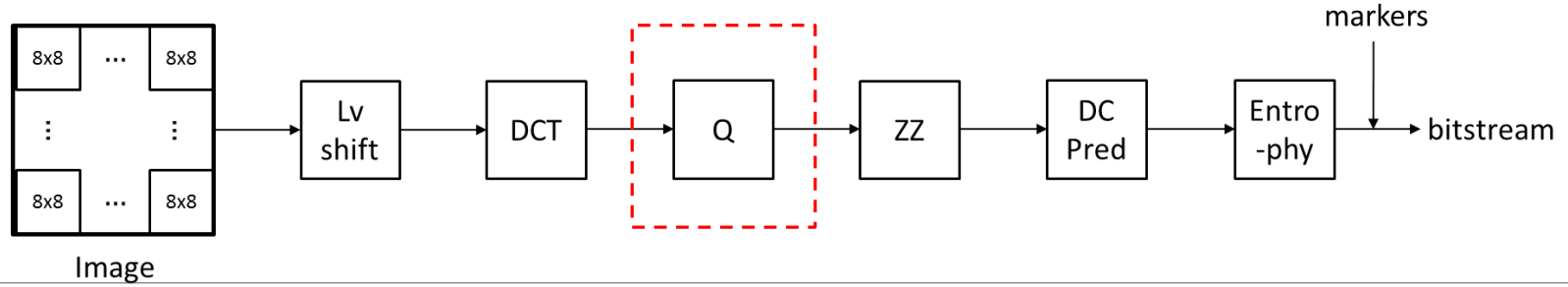


Image taken from “Video codec design developing image and video compression systems, wiley, by Iain E. G. Richardson”



- Quantization: For 64 DCT coefficients, they are quantized by integer division as:

$$Y_q^{ij} = \text{round}\left(\frac{Y^{ij}}{q_{ij}}\right)$$

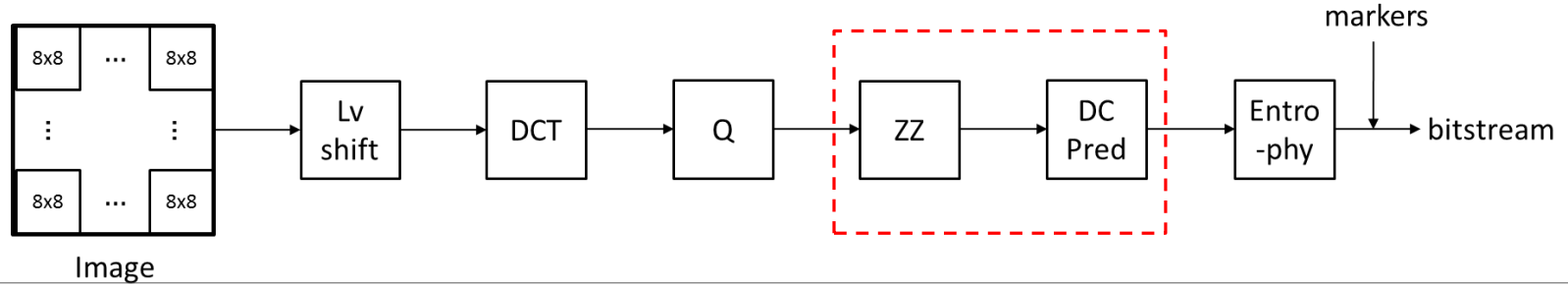
with a quantization table:

Y

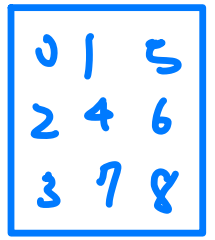
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

q



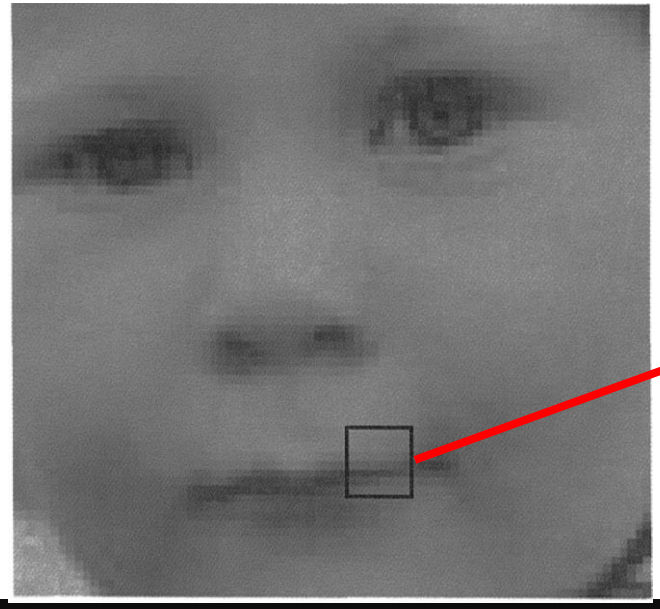
visit order:



- Zigzag reordering: A 8x8 block coefficients are rearranged in a zigzag scan order to regroup coefficients from low to high frequencies.

2D matrix → 1D array

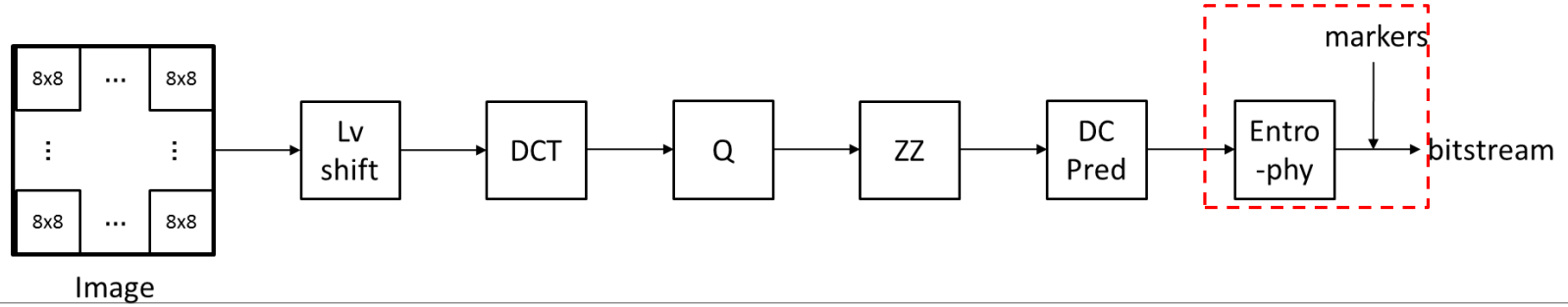
- DC differential prediction
  - a spatial prediction method that finds a correlation between the DC coefficients of neighboring blocks, assuming smooth transition between blocks



DC

967.5	-6.3	-10.7	-2.7	-1.2	-1.1	-1.3	0.1
-164.4	-71.3	1.8	4.2	-1.8	2.9	0.2	-1.6
55.3	13.6	-1.2	2.3	-0.6	2	0.7	-0.6
81.8	-8.9	-4.7	1.4	0.4	-0.5	-0.1	1
38.9	-12.9	-7.0	-1.3	-0.6	0.1	-0.3	1
-7.6	-8.4	0.8	2.6	0.6	-1.9	0.1	2.1
-14.7	1.4	4.1	1.2	0	-0.3	-0.2	1.1
-22.3	5	4.3	1.3	-0.6	-0.5	0.2	-0.3

All the other values except for DC are called ACs.

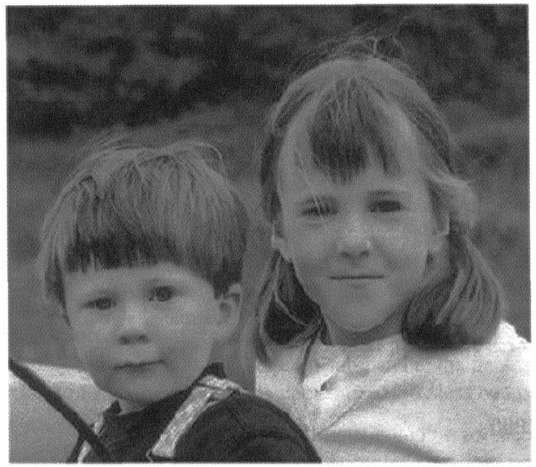


- Entropy coding: variable length coding for DC and AC values
- Marker insertion: headers
  - frame header: describing width, height, number of color channels
  - Restart interval markers (for resynchronization if an error occurs)

- Optional modes
  - Progressive encoding

DC only

DC + 2ACs



DC + all ACs

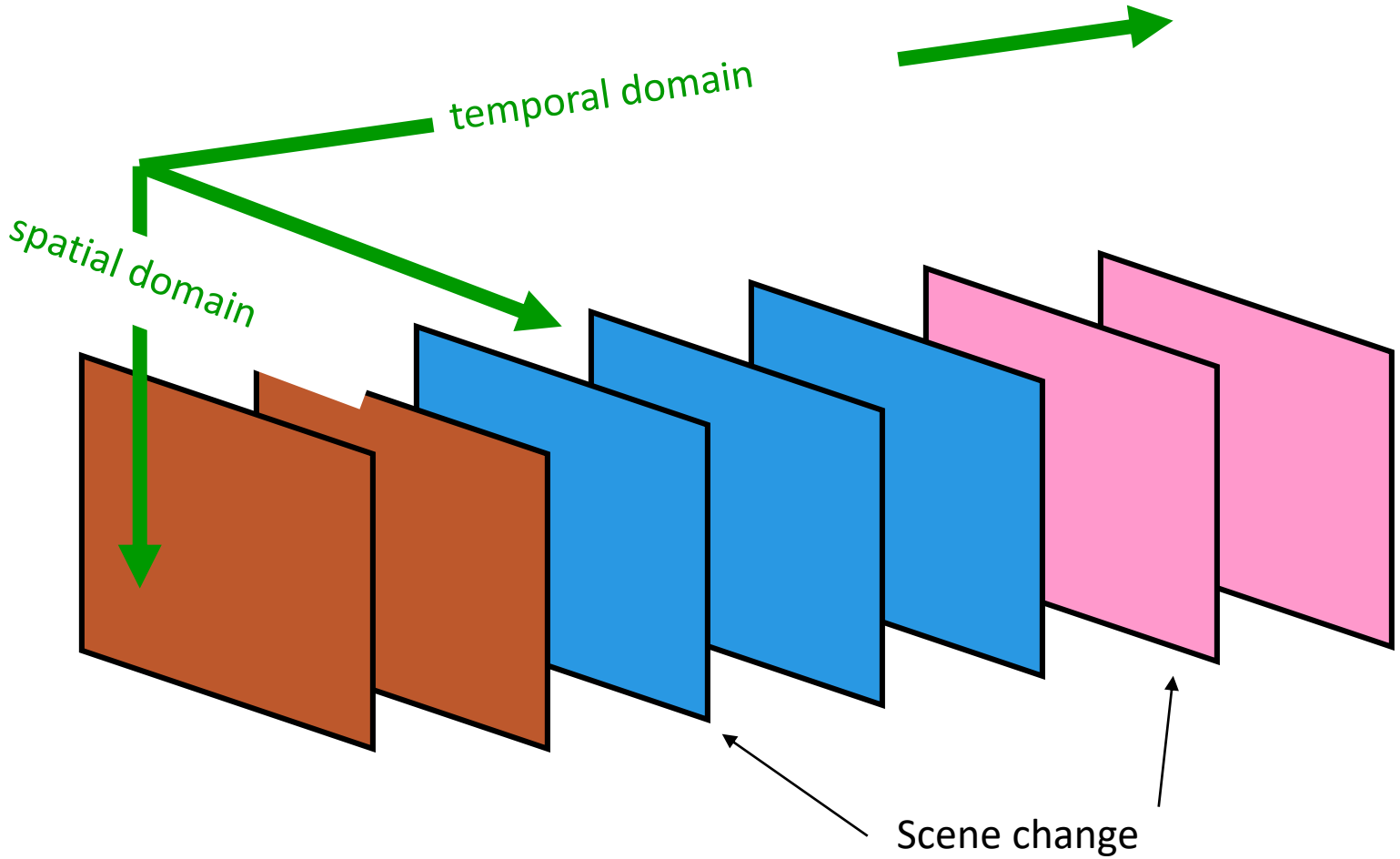
Images taken from Video Codec Design: Developing Image and Video Compression Systems, Iain E. G. Richardson

Video Predictive Coding

- Spatial Prediction intra-frame
  - In the spatial domain, neighboring pixels are used as predictors to predict the current pixel(s)
    - The differences are encoded

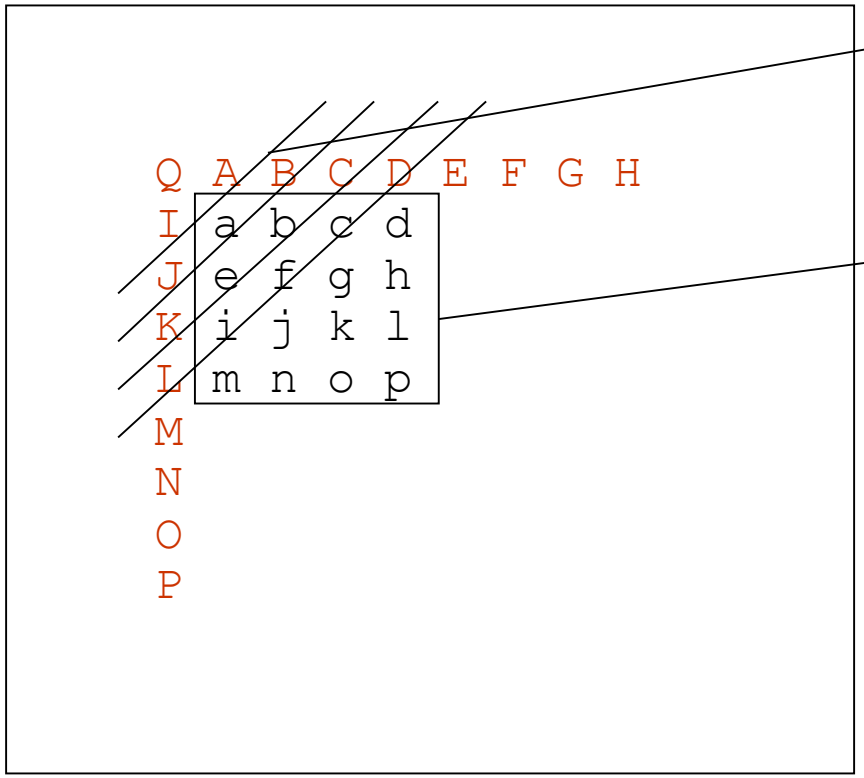
- Temporal Prediction inter-frame
  - In the temporal domain, pixels from the previous or future frames are used as predictors to predict the current pixel(s)
    - The differences are encoded

Predictive coding



spatial

Example of Spatial Prediction



Reconstructed Pixels:
A~Q

Pixels to be encoded

1. DC mode, predictor =
   (A+B+C+D+I+J+K+L)/8 -> a~p

2. 45° edge mode, predictor =
   (B+J)/2 -> a
   (C+K)/2 -> b, e
   (D+L)/2 -> c, f, i

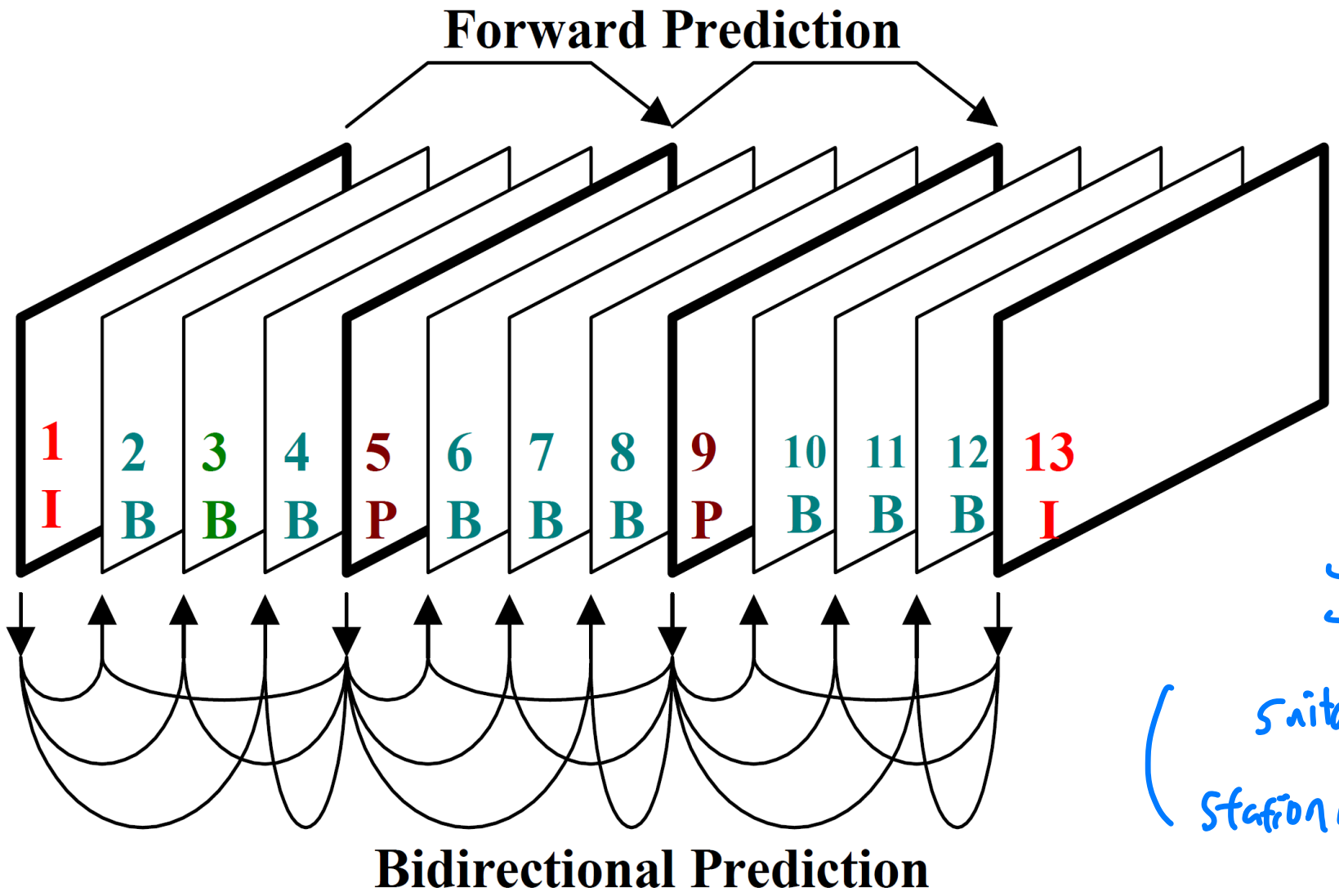
temporal

Video Frame Coding Type

- Intra-coded Frame (I Frame)
  - the coded frame does not require other frames to encode (keyframe)
  - Blocks can only be coded as intra-blocks

the more I frames, the better the video quality

- Inter-coded Frame
  - P Frame (Predicted Frame)
    - The coded frame requires previous frames to encode
    - In a P frame, blocks can be coded as intra- or inter-blocks
  - B Frame (Bidirectional predicted Frame)
    - the coded frame requires “previous and forward frames” or “forward frames only” to encode
    - In a B frame, blocks can be coded as intra or inter-blocks



IPB

(suitable for stationary recording)

Coding Order: $I_1 P_5 B_2 B_3 B_4 P_9 B_6 B_7 B_8 \ldots$ **Display Order:** $I_1 B_2 B_3 B_4 P_5 B_6 B_7 B_8 P_9 \ldots$

Alternative Coding Order: $I_1 P_5 B_3 B_2 B_4 P_9 B_7 B_6 B_8 \ldots$

Condition Replenishment

- Condition Replenishment
  - Image segmented into stationary, moving areas
  - Data transmitted only within moving area

Sprite

Foreground Object

pre-stored background

Decoded Frame

Sprite coding

🧠 What does “condition” mean here?

In this context, **“condition” refers to the state of each part of the image** — specifically, whether a region is:

- **Stationary** (not changing)
- **Moving** (changing)

🔄 What does “replenishment” mean?

“Replenishment” means **refreshing** or **updating** something — like refilling a supply.
So, in video compression:

- We only **replenish (update)** areas **where the condition has changed** (i.e., moving parts).
- Stationary areas **retain their previous values** — no need to resend them.

🧩 Putting it together:

> **Condition Replenishment** =
> “Only update the image data in regions whose condition (movement status) has changed.”

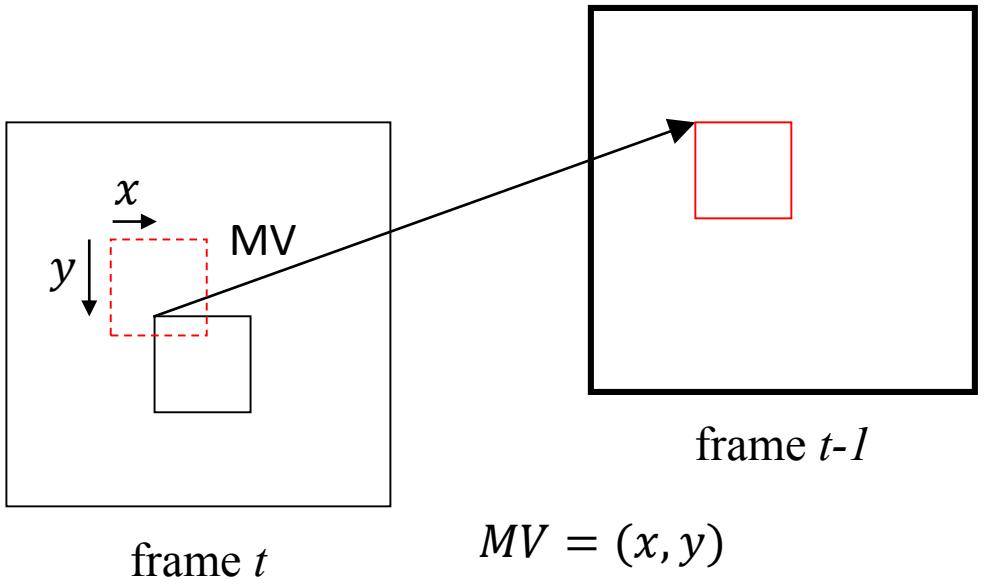
This allows the encoder to **save bandwidth** by **not transmitting redundant pixels** in areas that look the same as before.

Condition Replenishment & Motion Estimation

- Condition Replenishment
  - Image segmented into stationary, moving areas
  - Data transmitted only within moving area
- Motion Estimation
  - Use temporal redundancy
  - Pixels in previous picture are in different position in current picture

Introduction to Motion Estimation

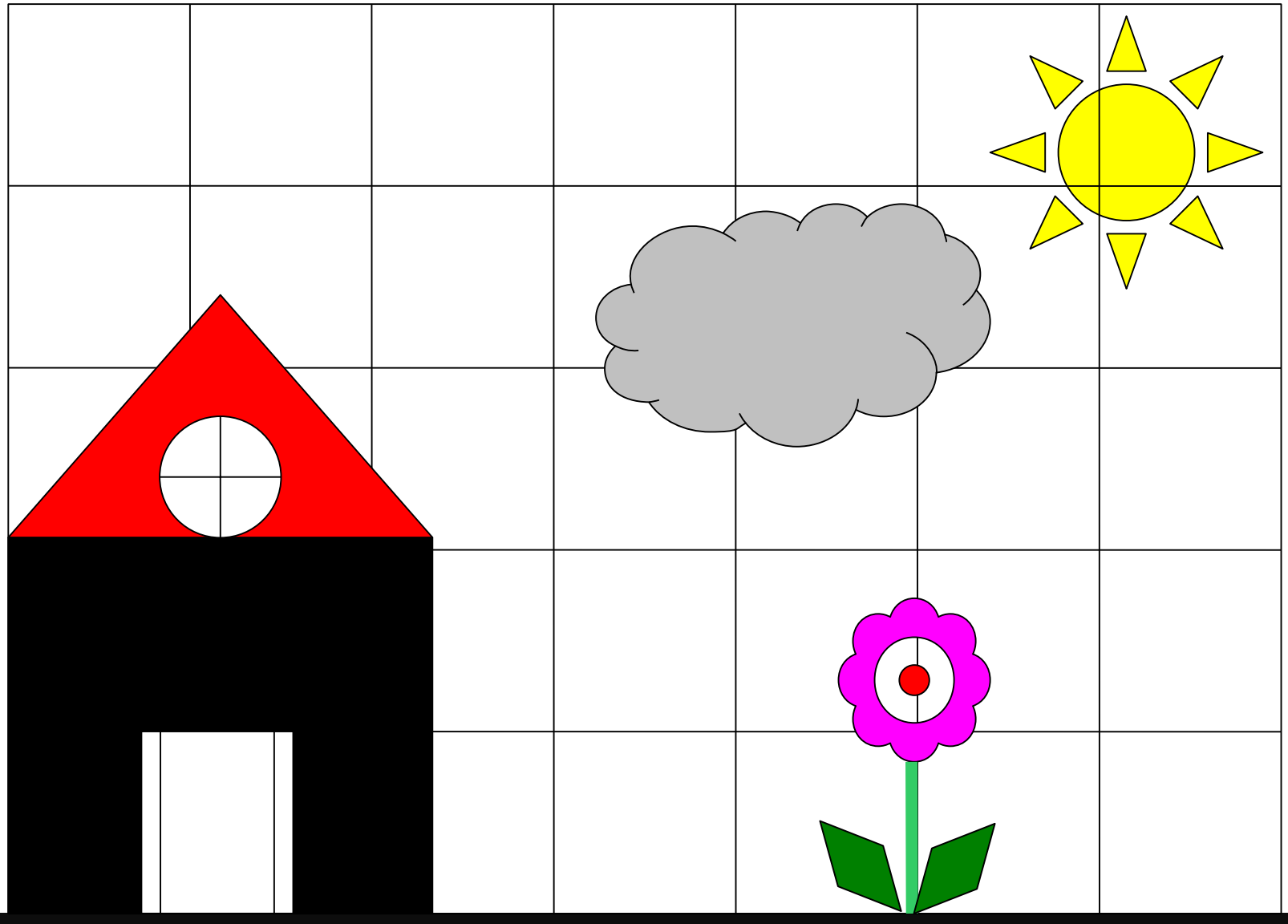
- Finding a match block in previous or future frames is called Motion Estimation.
- There are horizontal and vertical components for a motion vector as $MV = \begin{bmatrix} mv_x \\ mv_y \end{bmatrix}$
- Motion Vector (MV) search range is commonly squared.



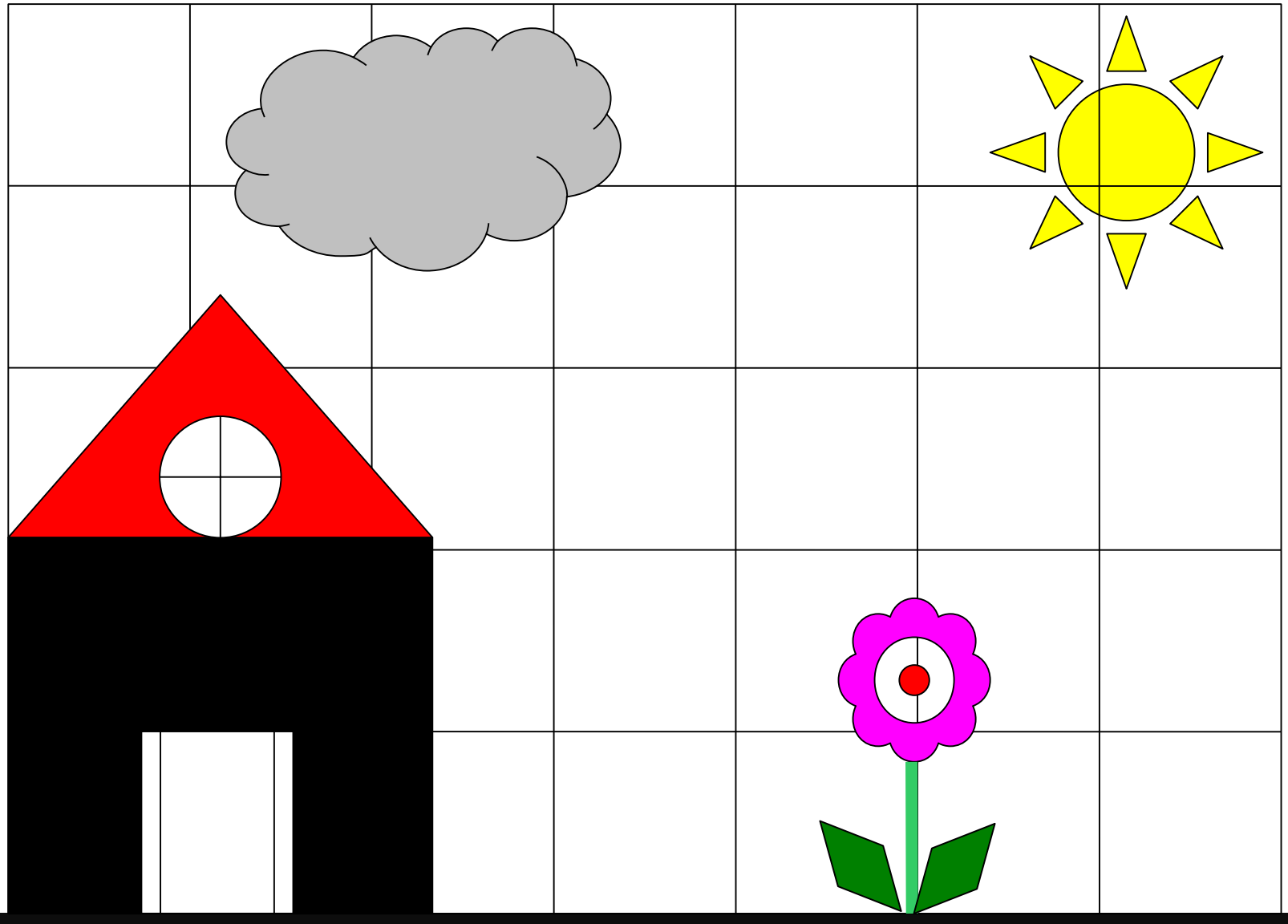
Search Range

- Suppose each frame is divided into blocks of size 16x16 pixels. To perform motion estimation for a block at position (200, 300) in the current frame, with a defined search range of ±32 pixels, the encoder will look for the best matching block in the previous frame within this range.
  - Horizontal Search: From 168 to 232 (200 - 32 to 200 + 32)
  - Vertical Search: From 268 to 332 (300 - 32 to 300 + 32)
- The encoder checks all possible blocks within this area in the reference frame to find the one that most closely matches the block in the current frame. The position of the best match relative to the original block's position defines the motion vector for that block.

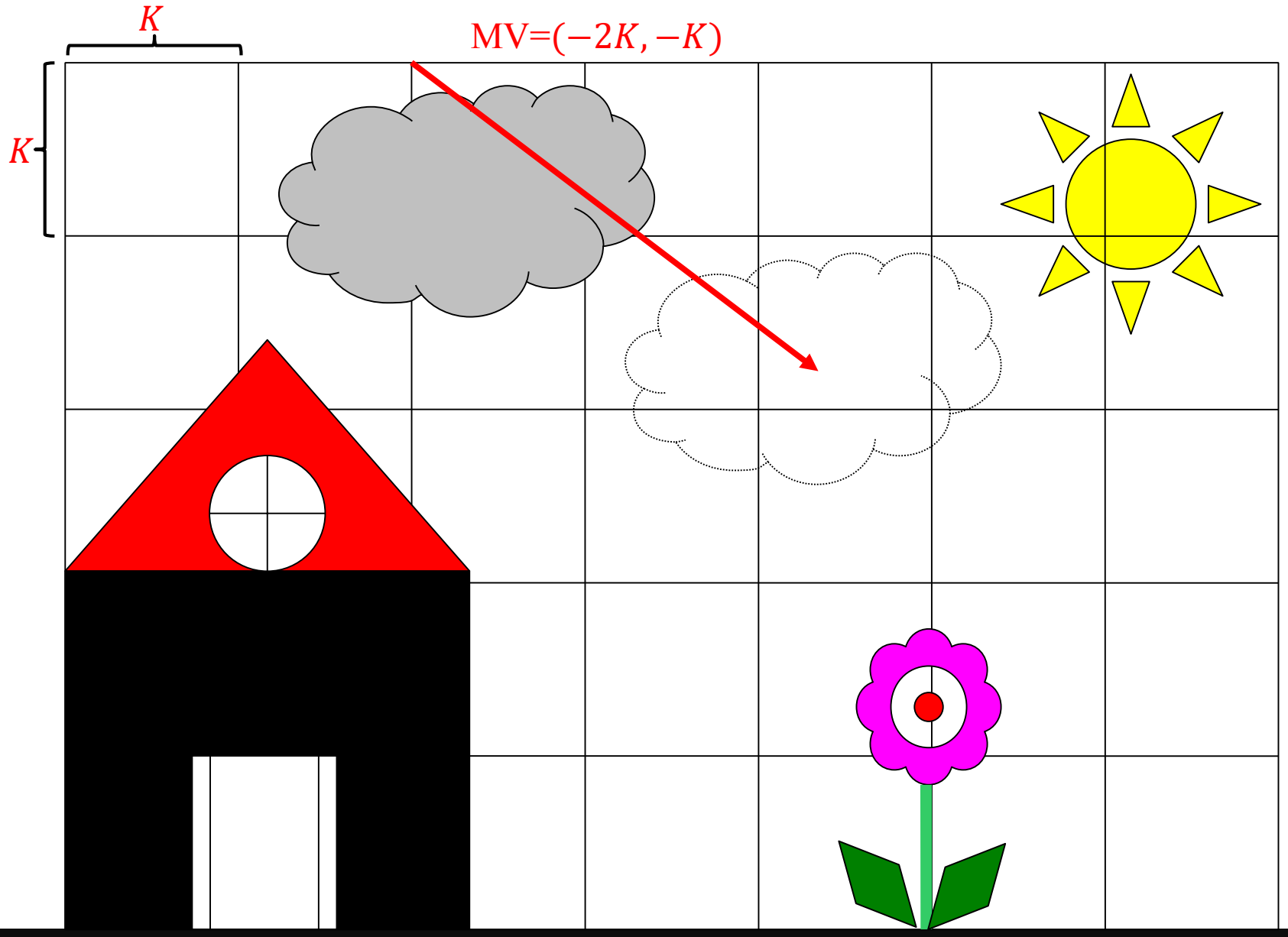
Frame *N-1*



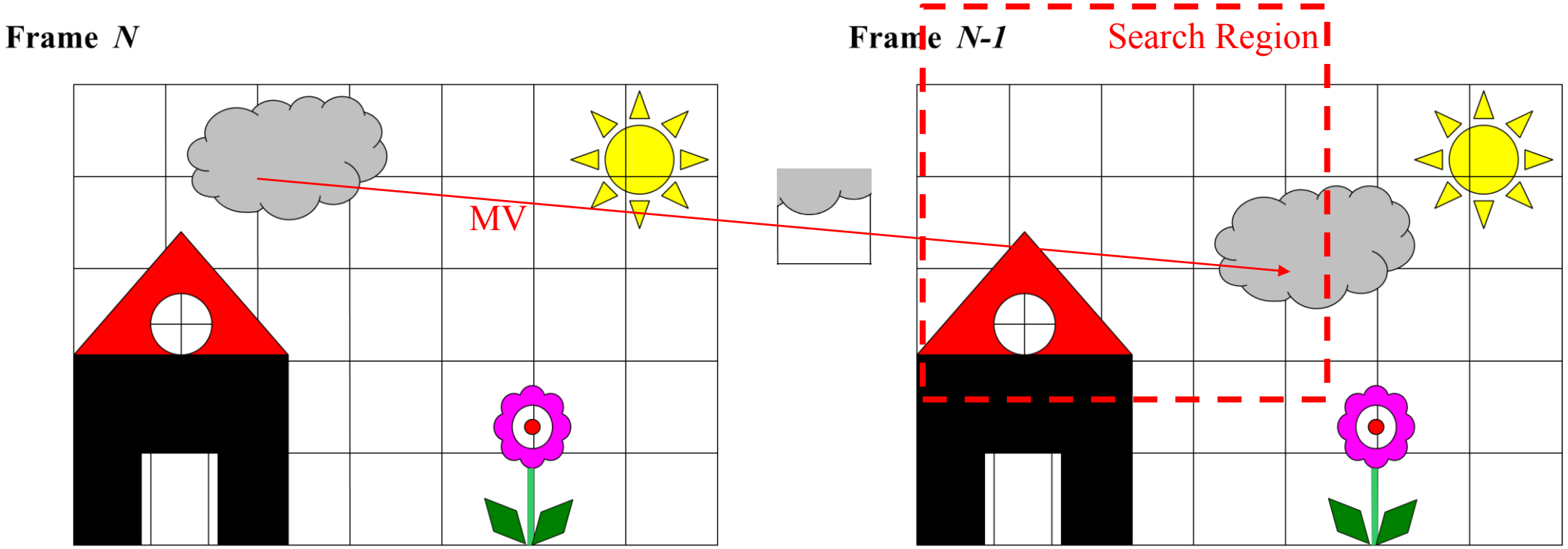
Frame *N*



Frame N

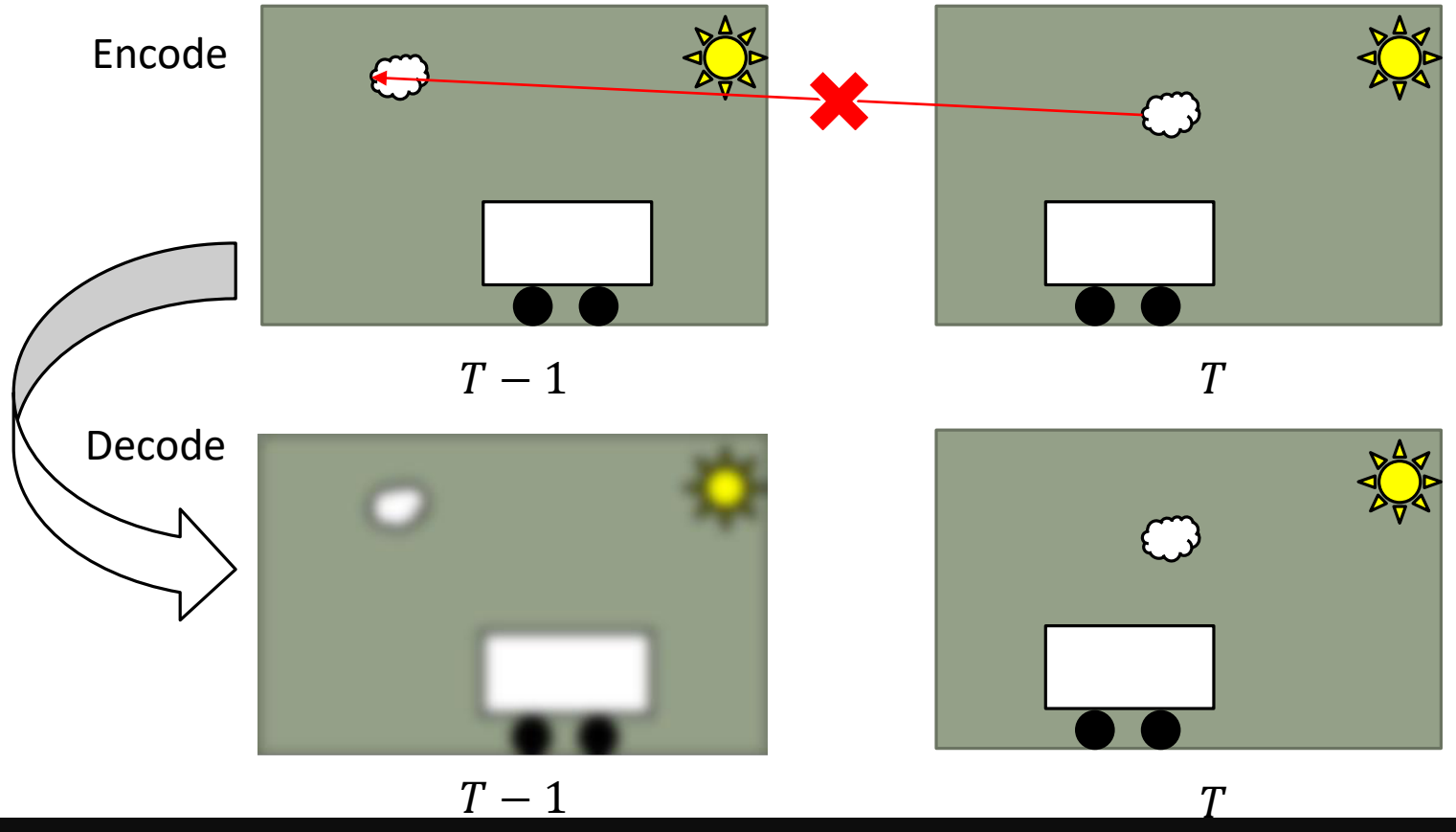


Block Matching



Video Encoder - Decoder

- Why do we need a decoding path in the encoding flow?



Consider what you have when decoding !!

Block Matching Criteria

- Underlying assumptions
  - Ambient lighting is unchanged
  - Objects are rigid (not changing in shapes)
  - Objects are translated in the 3D on a plane (not shrinking or enlarging in the 2D)
  - No objects are disappearing or appearing
- A similarity (dissimilarity) measure ε on a MV between blocks would be
  - $\varepsilon(MV) = \sum_{\forall(i,j)\in\Omega(x,y)} \delta\left(F_t(x+i, y+j), F_{t-1}\left(x+i+mv_x, y+j+mv_y\right)\right)$
    - $\Omega(x, y)$: search region centered in (x, y)
    - F_t , F_{t-1}: frame at time *t and t-1*
    - δ can be
      - correlation function
      - MSE
      - MAE
      - SAD

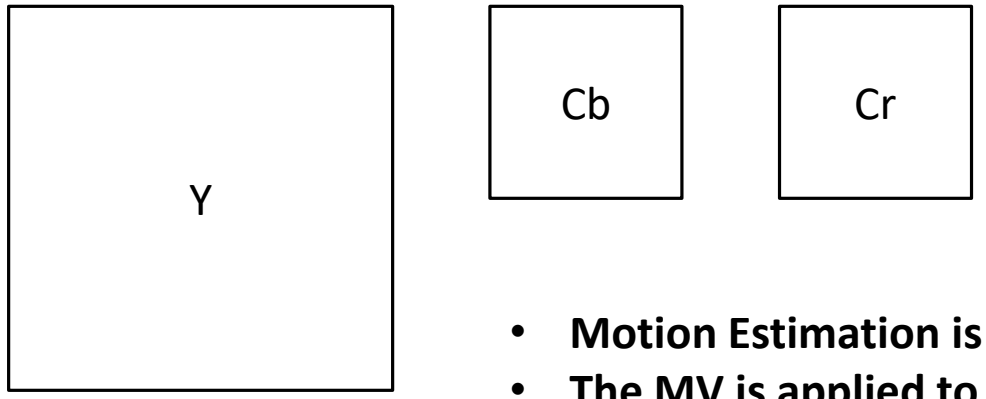
Block Similarity

- Mean Squared Error
  - $MSE = \frac{\sum_{M,N}[I_1(m,n)-I_2(m,n)]^2}{M \times N}$
- Mean Absolute Error
  - $MAE = \frac{\sum_{M,N}|I_1(m,n)-I_2(m,n)|}{M \times N}$
- Sum of Absolute Difference
  - $SAD = \sum_{M,N}|I_1(m,n) - I_2(m,n)|$

$\times (M \times N)$

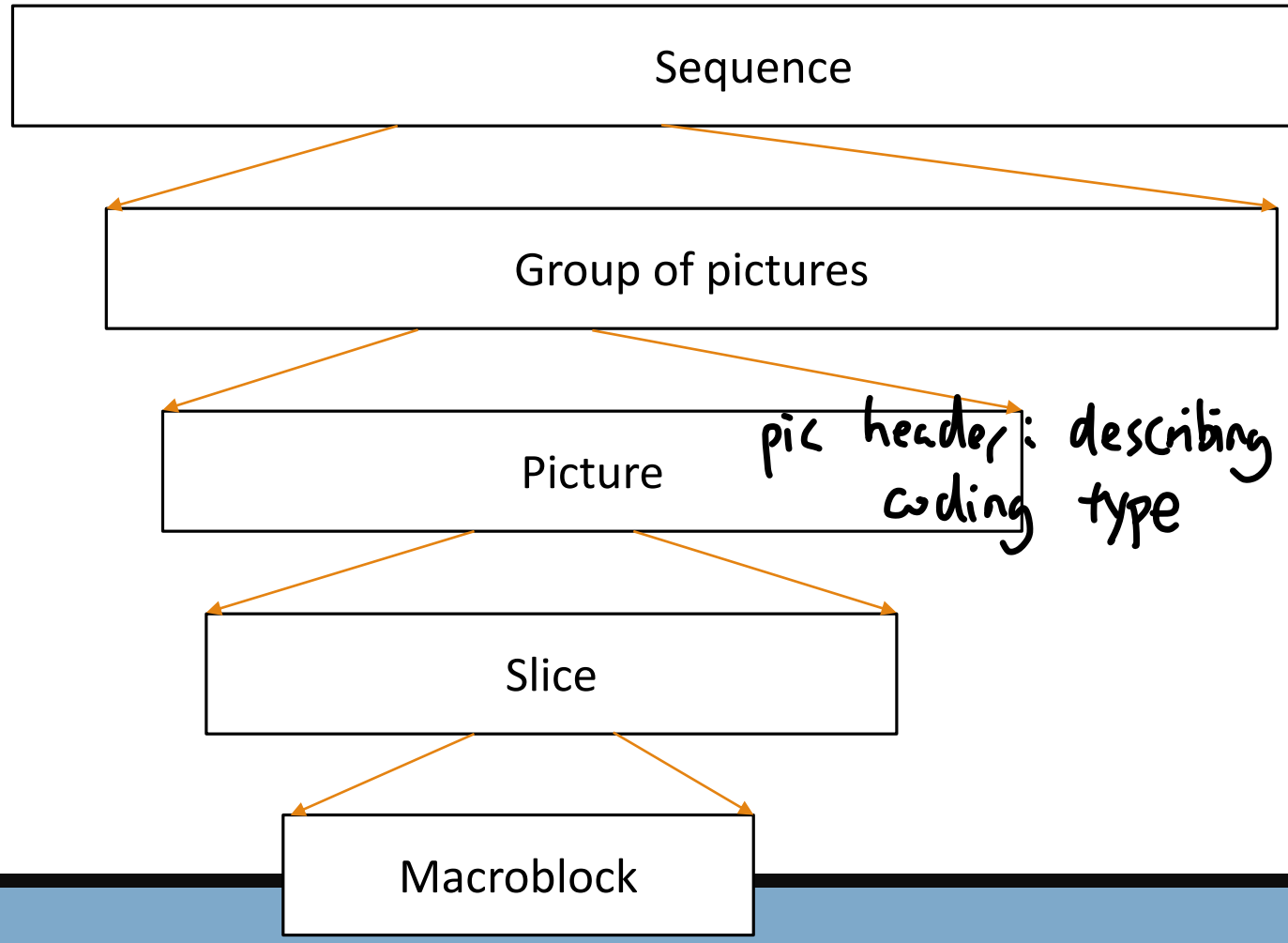
Block Partition

- In video coding, a basic processing unit is usually called "Macroblock" (MB).
  - Note that in HEVC, it has been replaced by "Coding Tree Unit" (CTU)
- It is used in most coding functional modules, such as transformation, quantization, motion estimation/compensation, deblocking, VLC coding, etc.
- For example, in H.263, mpeg4, and H.264. The size of a MB is 16x16, meaning 16x16 for Y, 8x8 for Cb and Cr (4:2:0)



- **Motion Estimation is usually done w.r.t Y**
- **The MV is applied to Cb and Cr**

Coding Syntax Hierarchy



Seq header: pic resolution, frame rate

A GOP contains one I followed by a series of Ps and Bs

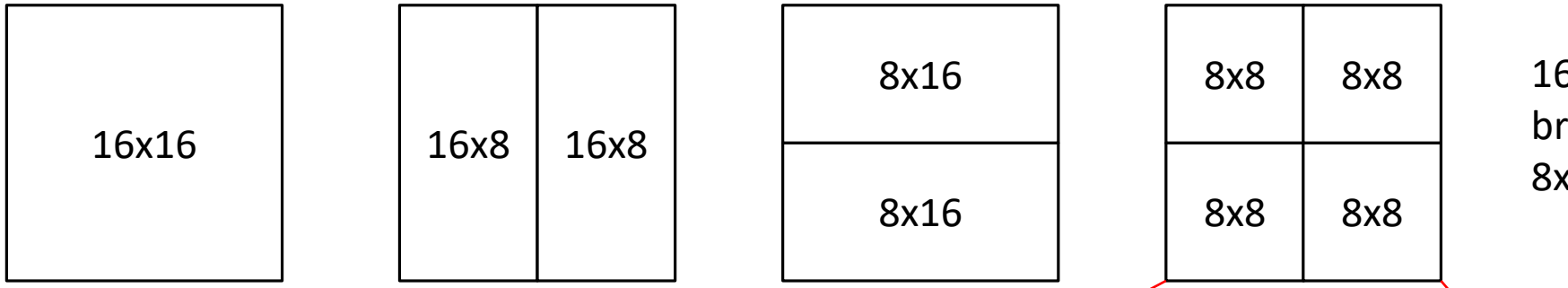
- All I, similar to Motion JPEG
- Only I and Ps: compression efficiency is relatively low but low computational complexity
- Large GOPs: compression efficiency is high but not good for unstable transmissions
- Small GOPs: compression efficiency is relatively low but good error resilience

Macroblock Partitions

Macro blocks

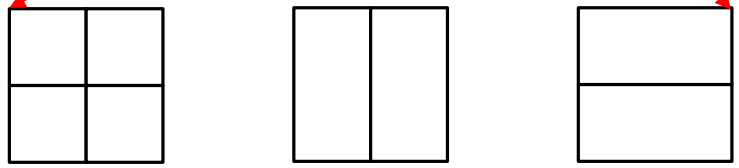
- We run Motion Estimation based on MB (H.263, MPEG4)
- A MB can be further divided into smaller blocks for ME (H.264)

Motion Estimation



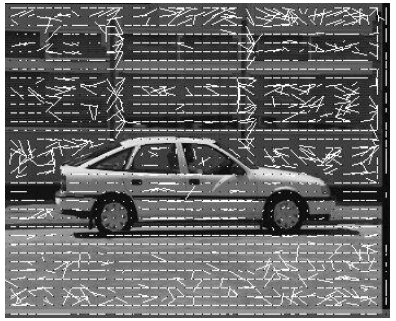
16x16 blocks can be broken into 8x8, 16x8, or 8x16.

8x8 blocks can be further broken into 4x4, 8x4, or 4x8.

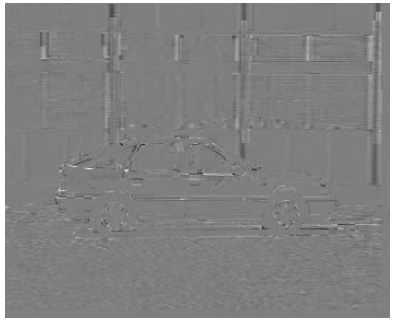


4x4 8x4 4x8

Motion Estimation (ME) and Motion Compensation (MC)



ME



MC Diff

Motion Estimation (ME) – Encoding

- **ME in the Encoding Process**
  - **Motion Estimation (ME)**:
    - **Block Comparison**: Calculate the difference between each block in the current frame and a set of candidate blocks from one or more reference frames. The difference is typically measured using a metric such as the Sum of Absolute Differences (SAD).
    - **Block Selection**: Identify the candidate block that shows the minimal difference from the current block. This is termed as the best-matching block.
    - **Difference Calculation and Motion Vector**: Compute the difference block, also known as the residual, by subtracting the best-matching block from the current block. Simultaneously, calculate the motion vector, which represents the displacement from the position of the best-matching block to the position of the current block.

difference block = current block - best matching block
(Residual)

🧱 1. Difference Block (Residual)

- It represents the **pixel-wise difference** between the **current block** and the **best-matching block** from the reference frame.
- It's a **block of pixel values**, showing what's **not explained** by motion alone.

📌 Formula (as shown in blue handwriting):

$$\text{Difference block} = \text{Current block} - \text{Best matching block}$$

🔧 **Why it matters**:

Even after matching motion, the blocks are not exactly the same, so we still need to send this residual to reconstruct the image accurately.

📍 2. Motion Vector (MV)

- It tells us **how far** and in **which direction** the best-matching block is located in the reference frame.
- It's a **single vector** (e.g., $MV = (mv_x, mv_y)$), not a block.

📌 Example:

If the best match is 4 pixels to the right and 2 pixels down, the MV is $(4, 2)$.

🔧 **Why it matters**:

It allows the decoder to **find the reference block** used to predict the current block.

Motion Estimation (ME) & Motion Compensation (MC) – Decoding

- **MC in the Encoding/Decoding Process**
- **Decoding Motion Vectors and Difference Blocks (Decoding)**:
  - **Decode the Data**: Retrieve and decode the encoded difference block and the motion vector from the compressed stream.
- **Reconstruction**:
  - **Apply Motion Vector**: Use the motion vector to locate and fetch the best-matching block from the referenced frame(s).
  - **Add the Difference**: Add the decoded difference block to this best-matching block. The result is the reconstructed block in the current frame.

reconstructed block = reference block (via MV) + difference block

Summary – ME/MC

- ME/MC in the encoding process

1. Calculate the difference between the current block and a set of candidate blocks in the reference frame(s).
2. Select the block with the lowest difference as the best-matching block.
3. Subtract the best-matching block from the current block to get the difference block and calculate the motion vector, which is the displacement from the position of the best-matching block to that of the current block.

- MC in the decoding process

1. Decode the difference block and its motion vector
2. Add the difference block to the best-matching block indicated by the motion vector and the position of the current decoded block.

Summary

- **Motion Estimation** is utilized during the encoding process to effectively reduce the amount of data needed by identifying areas of redundancy across successive frames.
- **Motion Compensation** complements this by using the motion vectors and difference blocks to reconstruct the frames during decoding. Together, ME and MC allow for significant video data compression by exploiting temporal redundancies, thus reducing the bandwidth and storage requirements for video transmission and storage.
- This mechanism is fundamental to modern video compression algorithms like MPEG and H.264/AVC, which are widely used in media streaming and broadcasting.

✅ ME (Motion Estimation) → Encoder side

- Finds the best-matching block from a reference frame.
- Computes:
  - The **motion vector** (where the block moved)
  - The **difference block** (residual between the current and matched block)
- The goal: **reduce redundant data before compression**.

✅ MC (Motion Compensation) → Decoder side

- Reconstructs the original block using:
  - The **motion vector** (to fetch the matching block)
  - The **difference block** (to restore fine detail)
- The goal: **accurately rebuild the video using compact motion info**.

🧠 So:

Component	Role	Side
ME	Analyze motion	Encoder
MC	Rebuild using motion	Decoder

🧠 Why MC is also in the encoder:

Even though **Motion Compensation (MC)** is mainly associated with **decoding**, the **encoder also performs MC** — but for a very specific reason:

🔁 Encoder must simulate the decoder

- The encoder must **predict what the decoder will reconstruct** — because the motion estimation is based on that.
- The decoder doesn't get the original reference frame — it only gets the **reconstructed** version (after quantization, transformation, etc.).
- So, the encoder **recreates that same decoded frame** using **inverse transform (T^{-1}), inverse quantization (Q^{-1})**, and **MC** — just like the decoder would.
- This way, the **reference used for ME/MC stays consistent** with what the decoder will have.

🔄 So what's happening in the encoder diagram?

1. Motion Estimation (ME) finds a matching block in a **previous decoded frame**.
2. Motion Compensation (MC) uses that block to build the **predicted current frame**.
3. The actual current frame is subtracted from the prediction to get the **residual**.
4. That residual is transformed, quantized, and encoded.
5. Meanwhile, the residual is also **decoded locally** (inverse Q and T) and added back to the prediction to **simulate what the decoder will reconstruct** — that becomes the new **reference frame**.

💡 **Summary:**

Role	Encoder	Decoder
ME	Finds best match	❌ Not used in decoder
MC	Builds predicted block	Reconstructs the predicted block
Residual	Created and encoded	Decoded and added
Inverse Q/T + MC	Simulates decoder's output	Actually reconstructs frame

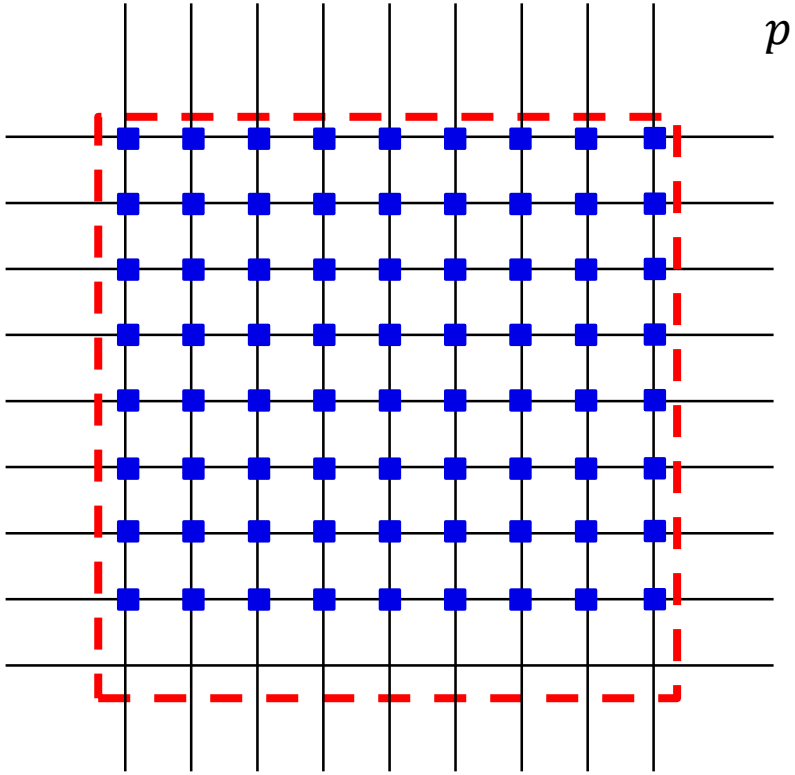
Block Matching (Motion Search) Algorithms

- Block matching is computationally expensive
  - Search is performed at every pixel position
  - Search can be performed over multiple reference frames
- There are several different search algorithms
  - Full Search
  - Three Step Search (TSS)
  - Cross Search
  - 2D Logarithmic Search
  - Hierarchical Motion Search
  - Diamond Search

Full Search

Search Range : $(2p + 1) \times (2p + 1)$

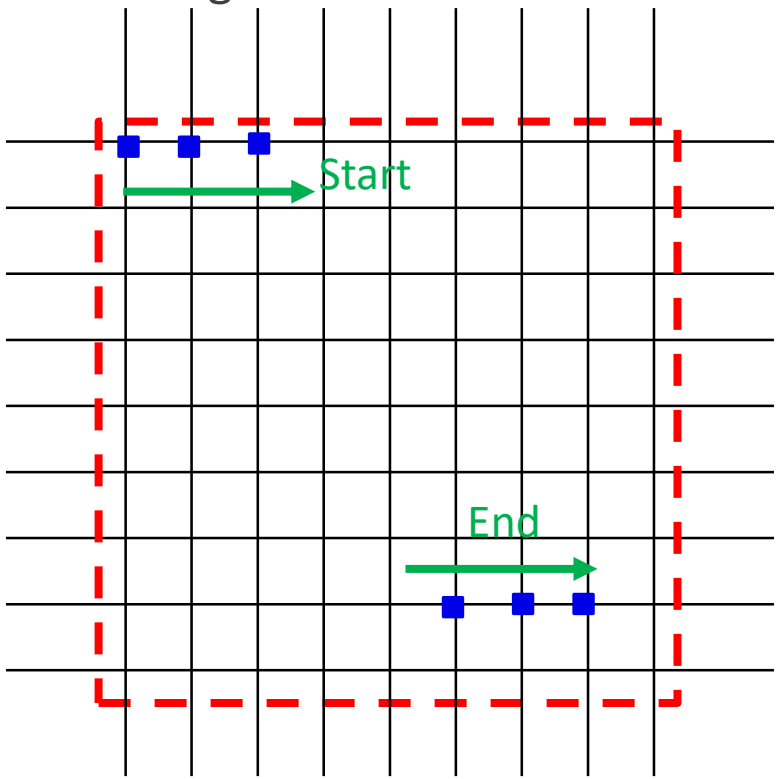
- Search point



$p = 4$

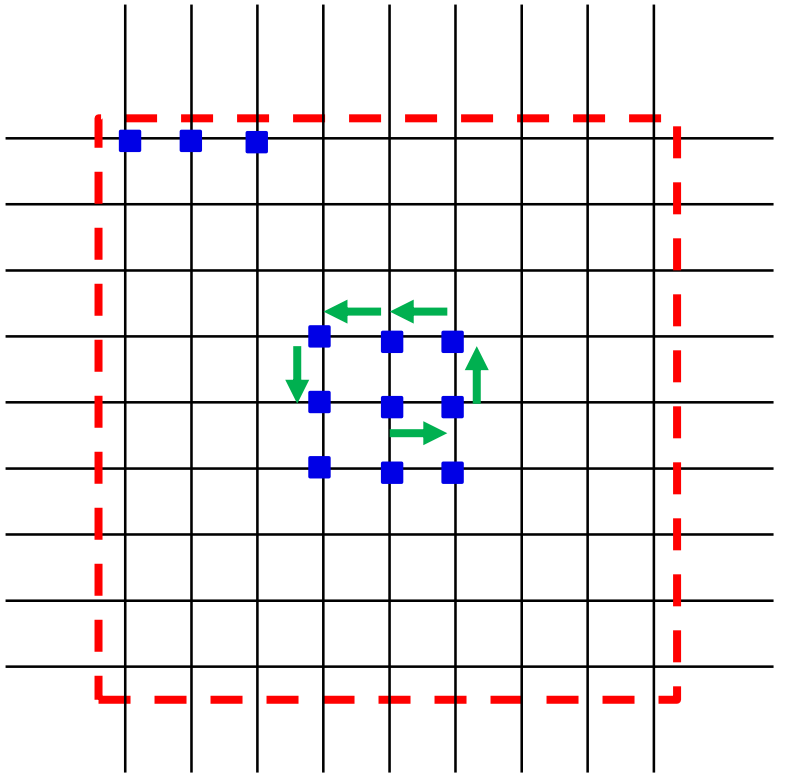
Full Search

- Computationally intensive
- Processing order



Raster order

Start from the center



Spiral order

we can set a threshold to reduce computation

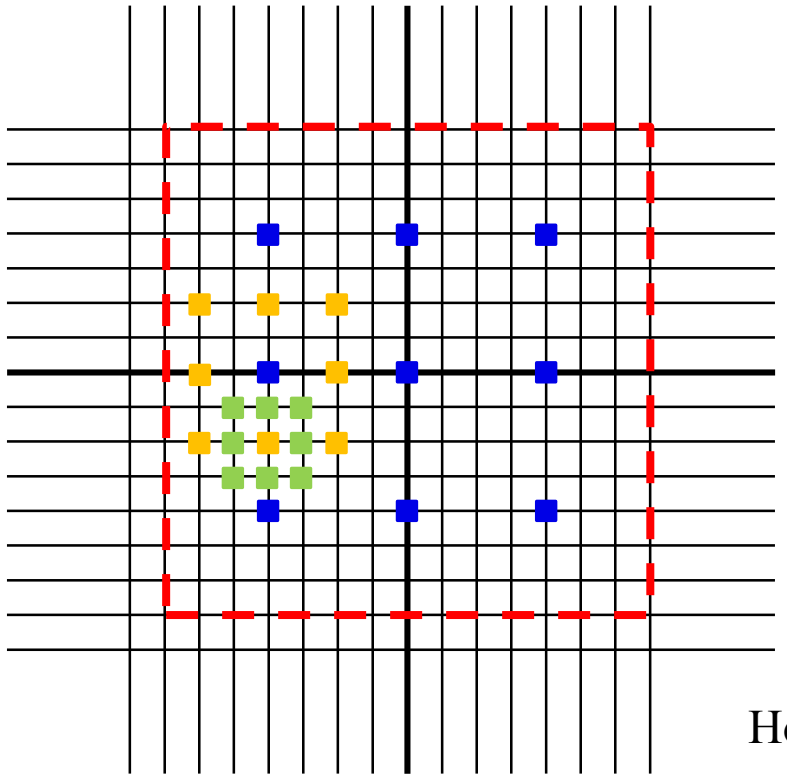
Three Step Search (N-Step Search)

sum of absolute difference

Search Range : $(2p+1) \times (2p+1), p = 7$

- 1st Search point
- 2nd Search point
- 3rd Search point

$N = 3$



1. Start origin: (0, 0), Step size: $S = 2^{N-1}$
2. Search 8 locations $\pm S$ pixels based on the origin
3. pick the loction with the smallest SAD and make it the new search origin (set $S = \frac{S}{2}$)
4. Repeat the above steps until $S = 1$

three-step search

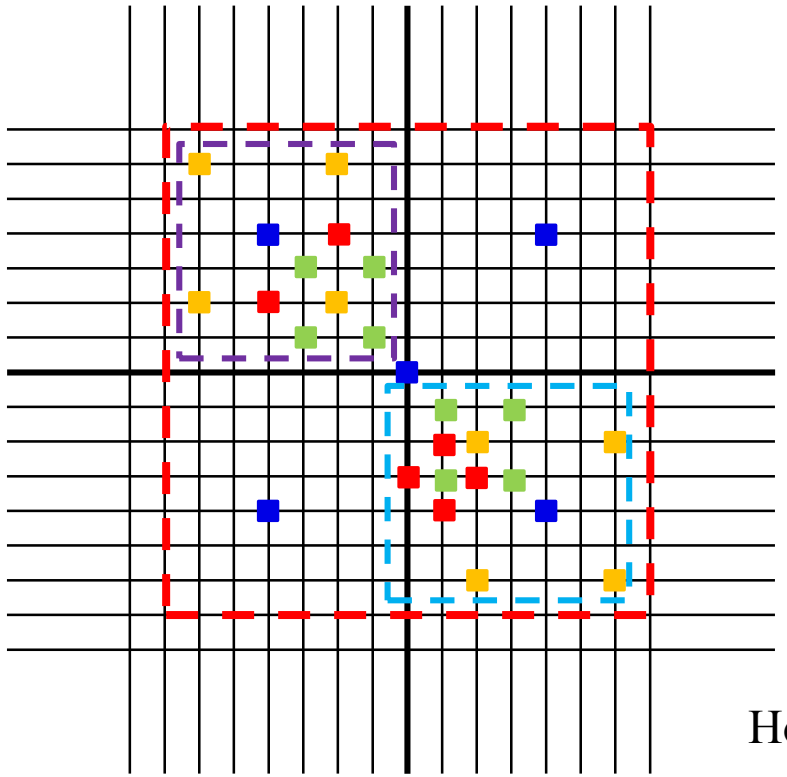
How many search comparisons are required for TSS?

Cross Search (N-Step Search)

Search Range : $(2p+1) \times (2p+1), p = 7$

$N = 3$

- 1st Search point
- 2nd Search point
- 3rd Search point
- 4th Search point
- Case 1
- Case 2



1. Start origin: (0, 0), Step size: $S = 2^{N-1}$
2. Search 4 locations like a × sign with $\pm S$ pixels away from the origin
3. pick the loction with the smallest SAD and make it a new search origin. If S > 1, make $S = \frac{S}{2}$; go to step 2.
4. If the best match is at the top left or bottom right of the ×, do four more points at a distance of ± 1 in × (case 1); otherwise (the best match is at the top right or bottom left), do four points at a distance of ± 1 in + (case 2);

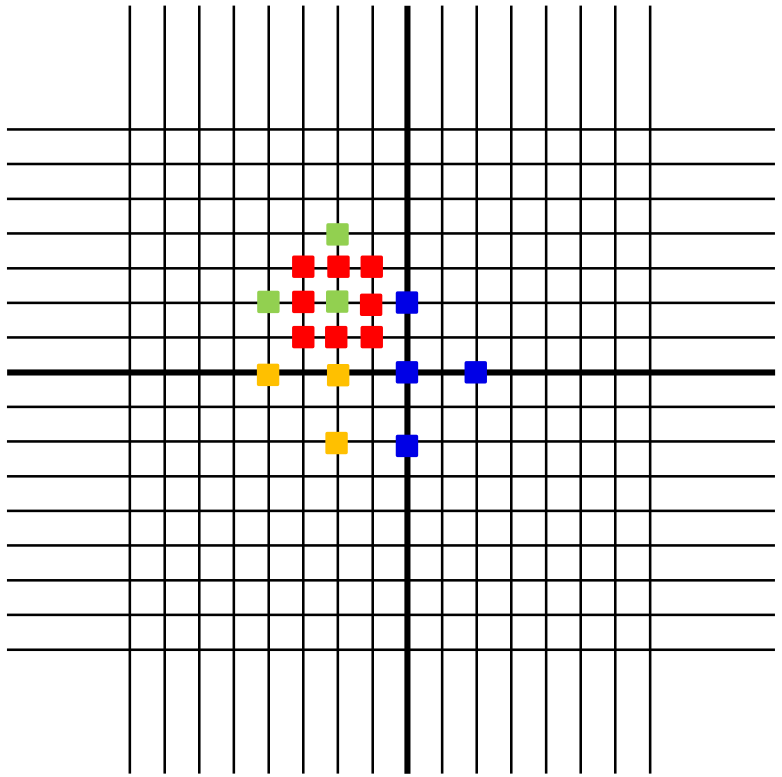
How many search comparisons are required for CS?

2D Logarithmic Search

search window = 15x15 = 225

Search Range : $(2p+1) \times (2p+1), p = 7$

- 1st Search point
- 2nd Search point
- 3rd Search point
- 4th Search point

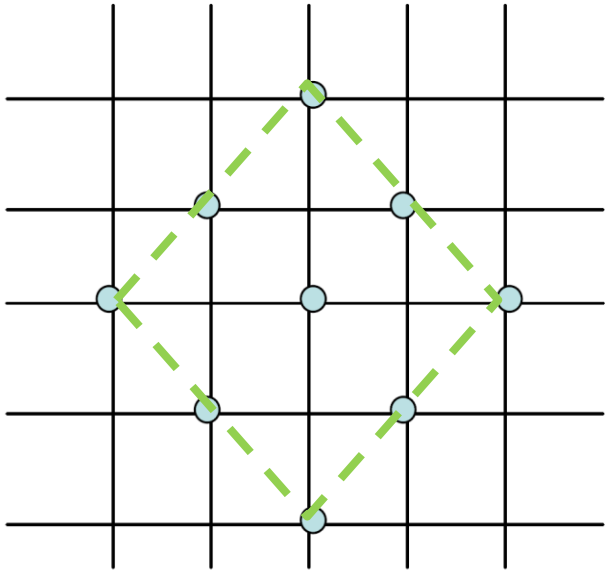


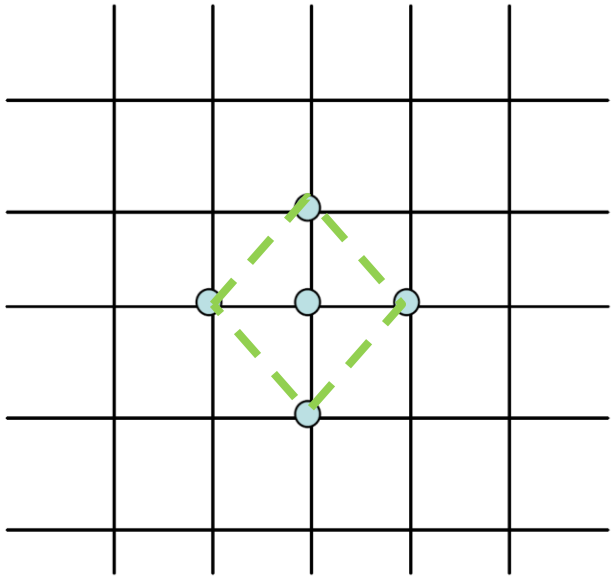
1. Start origin: (0, 0), Step size: $S = 2$
2. Search 4 locations like a + sign with $\pm S$ pixels away from the origin
3. pick the loction with the smallest SAD and make it a new search origin. If the best match is at the center of +, make $S = \frac{S}{2}$; **otherwise, keep S unchanged.**
4. If $S = 1$, search 8 locations 1 pixel away from the origin. The search result is the best match among these 9 locations (including the search origin); otherwise go to step 2.
5. Search will not proceed outside of the search range.

How many search comparisons are required for 2D log Search?

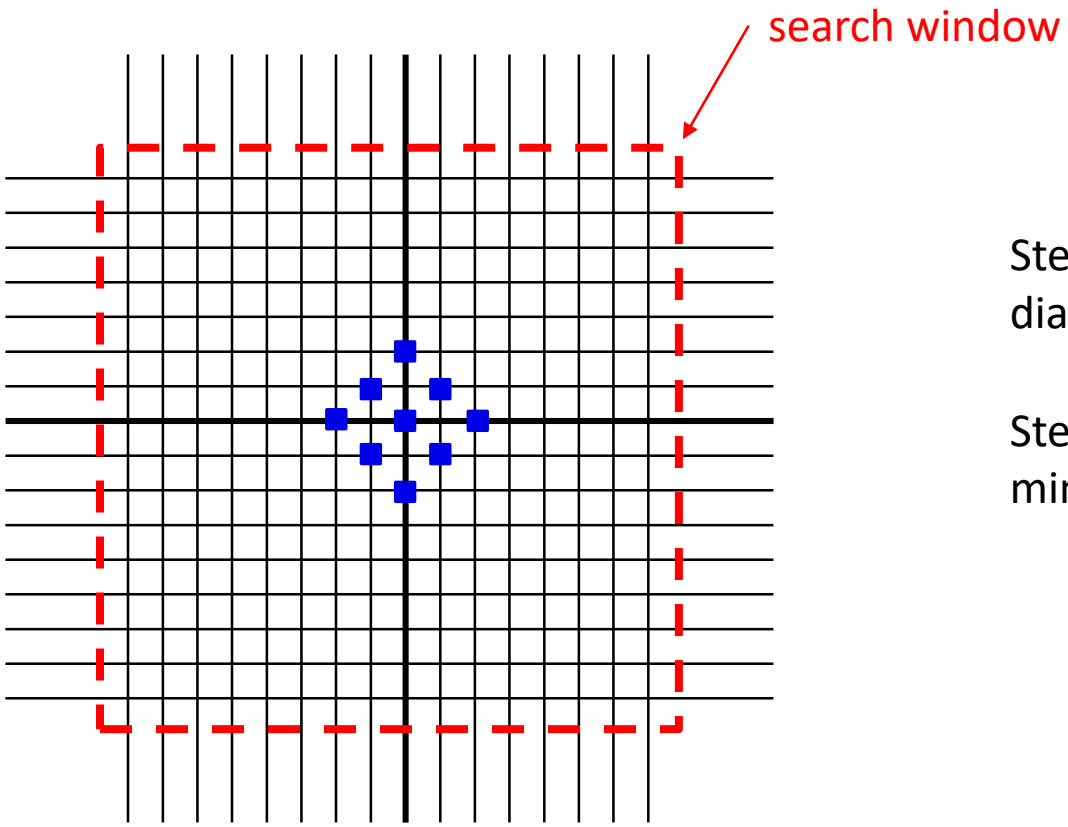
Diamond Search

- Large diamond search pattern of radius 2
- Small diamond search pattern of radius 1.





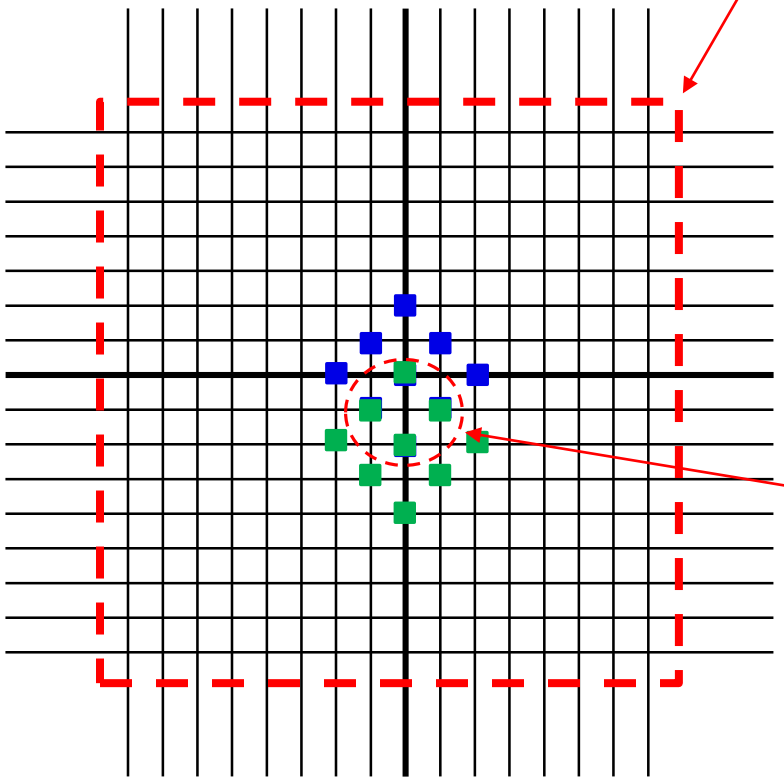
Diamond Search



Step 1: Search all the positions of the large diamond at the center of the search window

Step 2: check if the center position has the minimum error

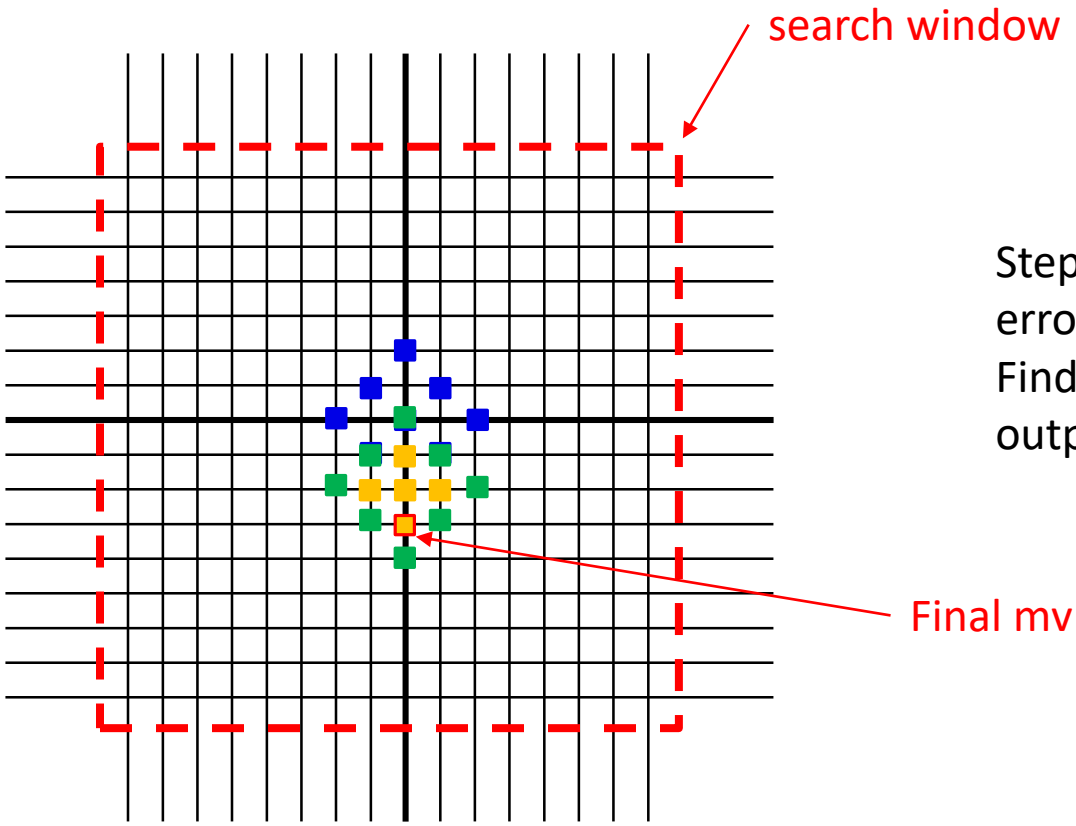
Diamond Search



Keep running Step 2 and 3 until the center has the minimum error

Step 3: If not, move to the minimum position and use it as the new center. Apply the large diamond to the new center position. Go back to Step 2.

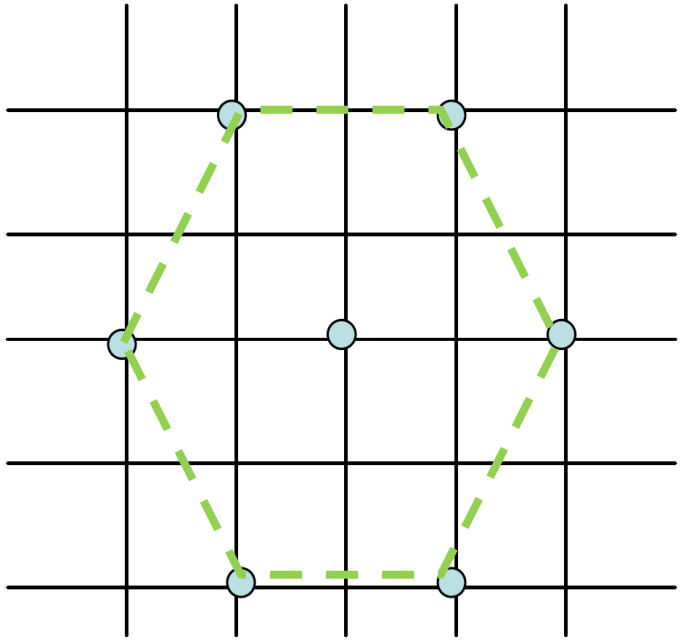
Diamond Search

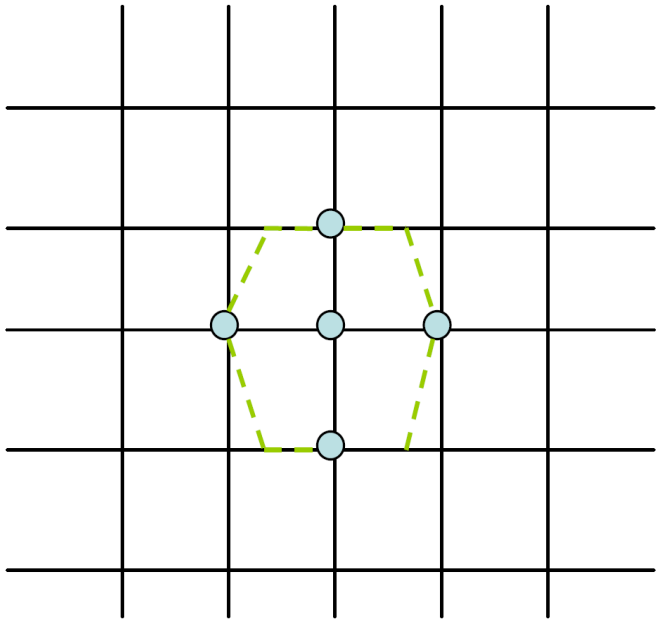


Step 4: The center position has the minimum error. Apply the small diamond to the center. Find the position with the minimum error and output the final mv.

Hexagon Search Algorithm

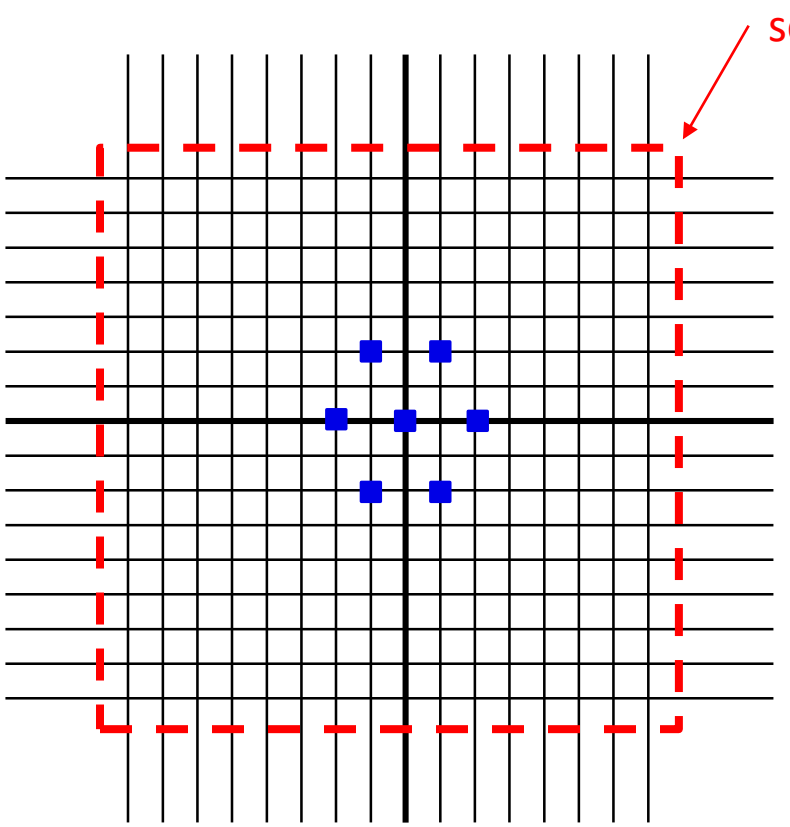
- Large hexagon search pattern of radius 2
- Small hexagon search pattern of radius 1.





The search steps are similar to those of diamond search

Hexagon Search Algorithm

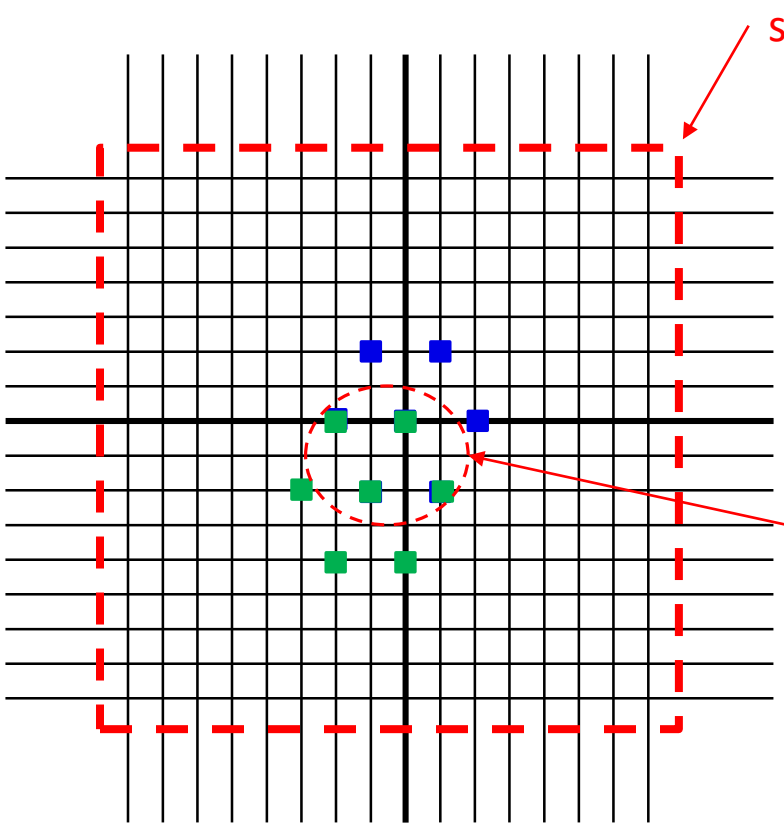


Step 1: Search all the positions of the large hexagon at the center of the search window

Step 2: check if the center position has the minimum error

C. Zhu, X. Lin and L-P. Chau, IEEE CSVT, 2002

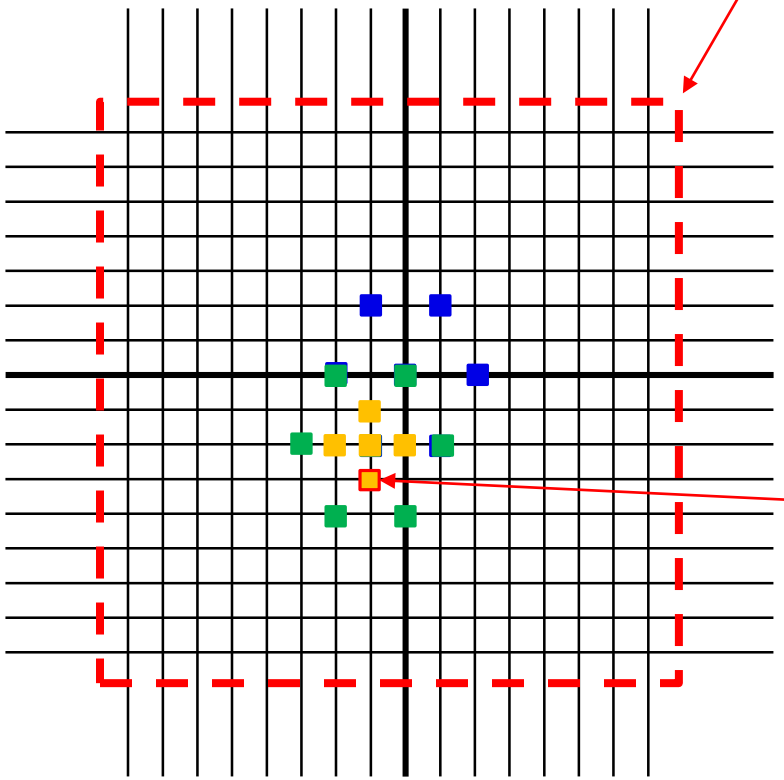
Hexagon Search Algorithm



Step 3: If not, move to the minimum position and use it as the new center. Apply the large diamond to the new center position. Go back to Step 2.

C. Zhu, X. Lin and L-P. Chau, IEEE CSVT, 2002

Hexagon Search Algorithm



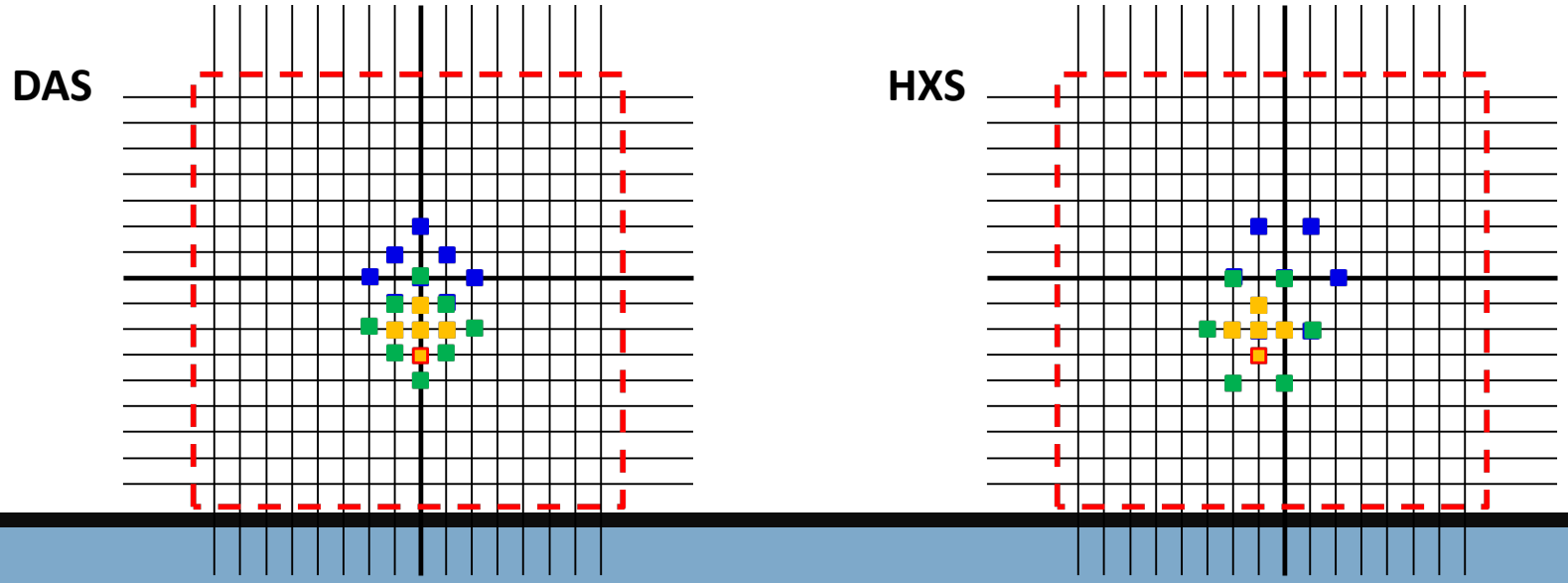
Keep running Step 2 and 3 until the center has the minimum error

Step 4: The center position has the minimum error. Apply the small hexagon to the center. Find the position with the minimum error and output the final mv.

C. Zhu, X. Lin and L-P. Chau, IEEE CSVT, 2002

Diamond Search (DAS) vs. Hexagon Search (HXS)

- There are fewer search points in HXS than DAS.
- For two large-pattern searches and one small-pattern search, HXS requires 14 search points, whereas DAS needs 18 search points.
- For longer MVs, HXS saves computation costs more.
- Overall, DAS does a slightly better job in terms of PSNR than HXS



Hierarchical Motion Search

- Hierarchical Motion Search, also known as Hierarchical Motion Estimation
  - A multi-layered approach to motion estimation, where the search for motion vectors is performed across different scales or resolutions of the video
  - It begins with analyzing **low-resolution** versions of the video frames and progressively refines the motion estimation in **higher resolutions**.

- **Steps in Hierarchical Motion Search**
  - **Creation of Image Pyramids**
    - Video frames are rescaled into multiple resolutions, creating a pyramid of images for each frame. The lowest resolution provides a rough and computationally cheap estimation, while each subsequent higher resolution offers finer details.
  - **Coarse-to-Fine Motion Estimation**
    - **Initial Search at Lower Resolution**: The search for matching blocks (regions in the frame with similar content) starts at the lowest resolution. This stage aims to quickly and efficiently identify potential areas of motion without the computational cost of processing high-resolution data.
    - **Refinement at Higher Resolutions**: Once a motion vector is estimated at a coarser level, it is used as a starting point for the search at the next higher resolution. This process is repeated until the highest resolution is reached. The initial estimates guide the search in finer resolutions, reducing the search range and computational load.
    - **Refined Motion Vector Calculation**
      - At each level of the pyramid, the motion vector is adjusted and refined based on more detailed image data.

Hierarchical Motion Search

Hierarchical Motion Search is utilized in various advanced video compression standards, such as MPEG-4, H.264/AVC, and HEVC.

Disadvantage: small objects could be eliminated because of downsampling

How many search comparisons are required for HMS?

Low Pass Filter

Gaussian

Down-sampling

$x \downarrow 2$
$y \downarrow 2$

Low Pass Filter

Gaussian

Down-sampling

$x \downarrow 2$
$y \downarrow 2$

Refinement: search 8 locations 1 pixel away from the origin.

MV

Motion Estimation

Refinement

Search Range : 31 × 31 (8)

× 2

Motion Estimation

Refinement

The best match from the previous level is used as the center

Search Range : 15 × 15 (8)

× 2

Motion Estimation

Full Search → initial 'coarse' mv

May find the global minimum

Search Range : 7 × 7 (49)

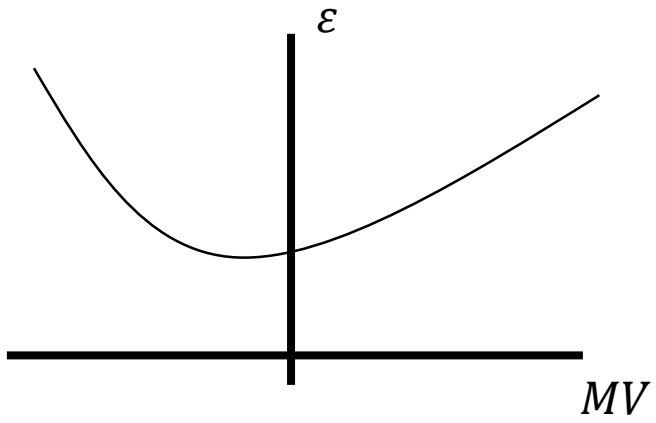
Global Minimum vs. Local Minima

$$\arg\min_{MV} \varepsilon(MV) = \arg\min_{MV} \sum_{\forall(i,j)\in\Omega(x,y)} \delta\left(F_t(x+i, y+j), F_{t-1}\left(x+i+mv_x, y+j+mv_y\right)\right)$$

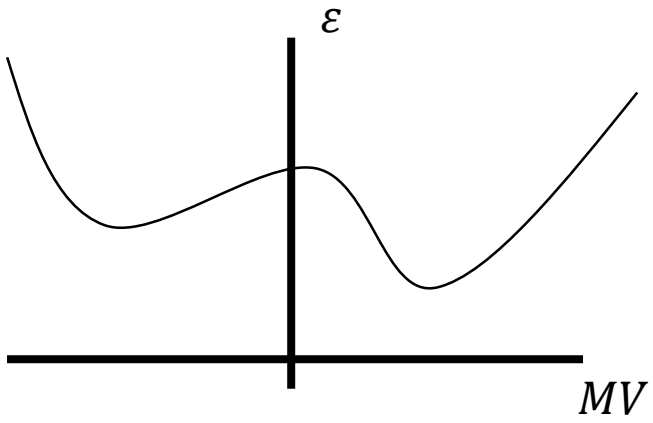
δ: distance metric

F_t: Frame at time t

Ideal Scenario

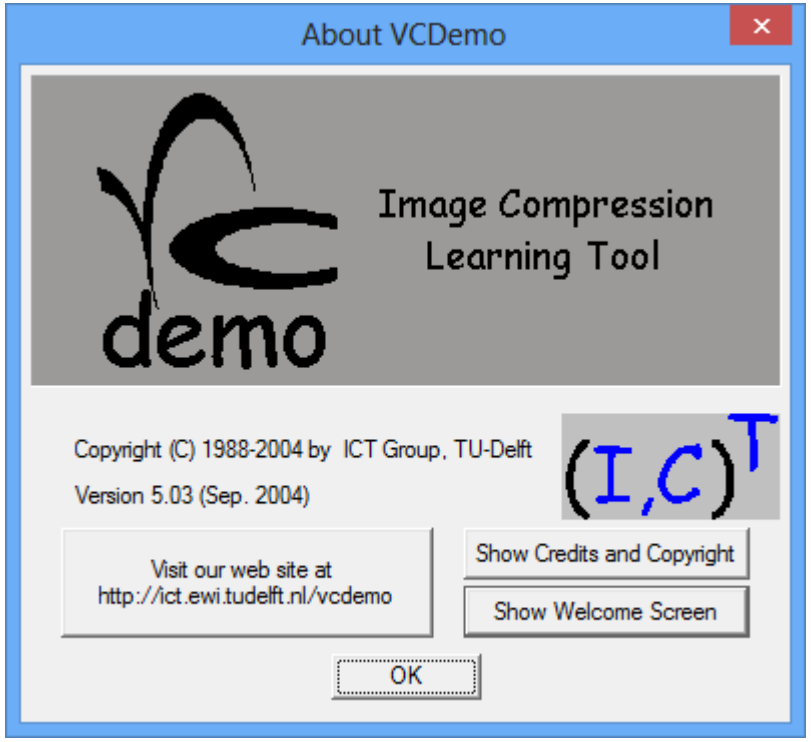


Practical Scenario



VcDemo

- VcDemo is an interactive image and video compression freeware software package for Windows.
- http://homepage.tudelft.nl/c7c8y/VcDemo.html
- You can use it to demonstrate results of different encoding options

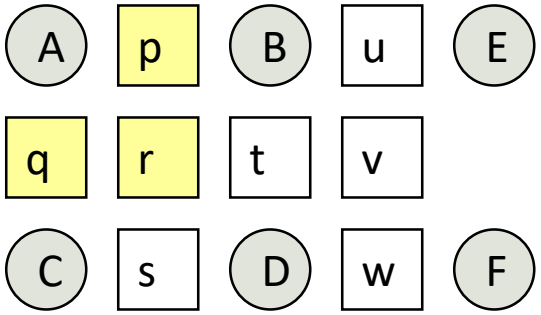


Sub-pixel Motion Estimation

- Usually, we assume that the best ME predictor can be found at an integral region displacement.
- Actually, a better predictor can be obtained by searching at a **sub-pixel offset** interpolated by integer pixels.
- Increasing the “depth” of sub-pixel interpolation leads to better block matching performance; however, it also increases computational complexity.
- One can first use integer ME to find a good predictor at the integer locations, and then refine it by searching the surrounding sub-pixel locations.
- It is proven that sub-pixel ME can substantially **outperform** integer ME in terms of video quality because objects not necessarily move by an integral number of pixels due to a high frame rate.

Half-pixel and Quarter-pixel Motion

- Error residuals are calculated with half-pixel or quarter-pixel resolution
  - pixel interpolation is required



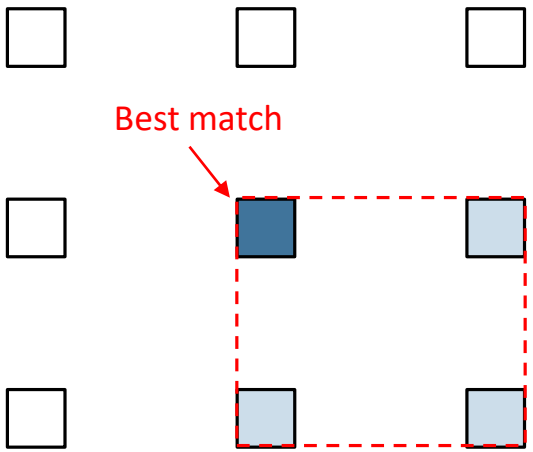
Half-pixel Motion

$p = (A+B+1)/2$
$q = (A+C+1)/2$
$r = (A+B+C+D+2)/4$

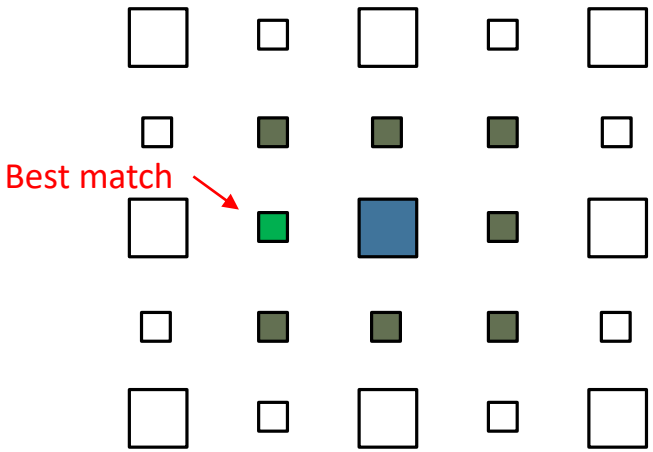
Compute the block similarity at each pixel/half-pixel/quarter-pixel position within the search window

Sub-pixel Motion Estimation - Half Pixel (Hpel) Search

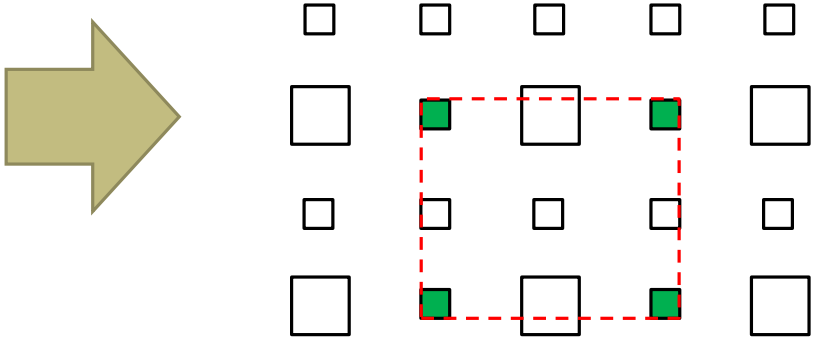
Assuming the block size is 4x4



Integer pixel search



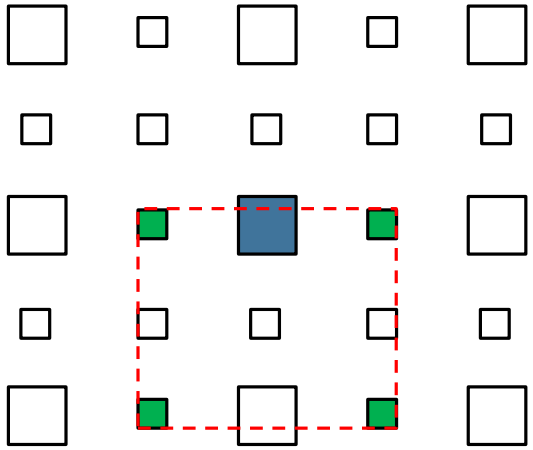
Half pixel (Hpel) search



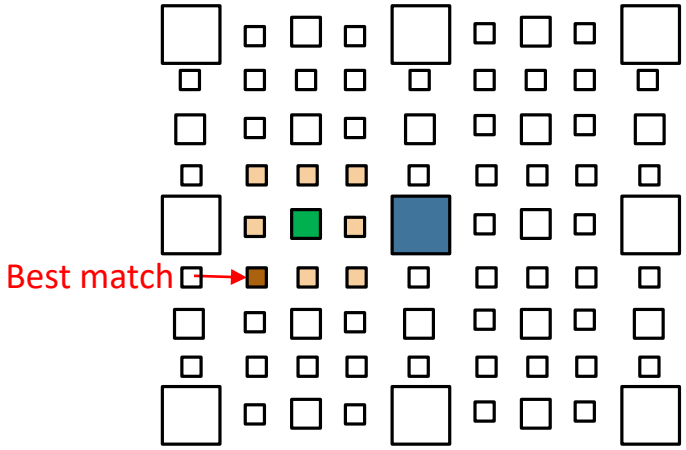
Half pixel (Hpel) search

Sub-pixel Motion Estimation – Quarter Pixel (Qpel) Search

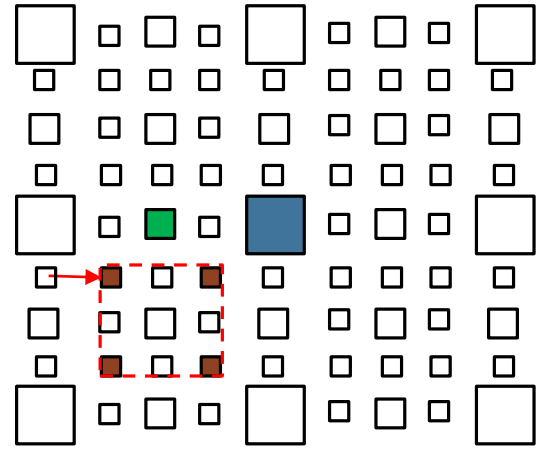
Assuming the block size is 4x4



Half pixel search



Qpel search

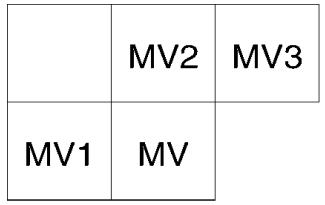


Qpel search

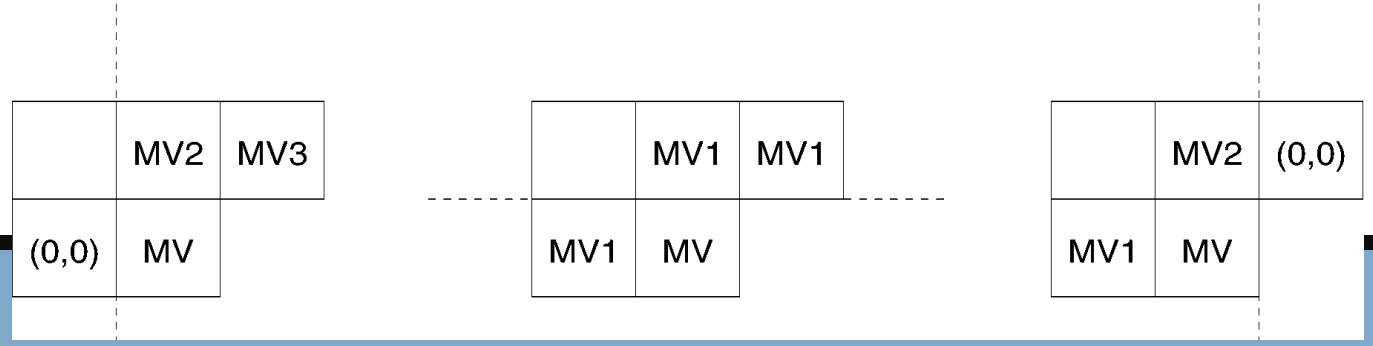
$\frac{1}{8}$ search can also be applied

Motion Vector Prediction

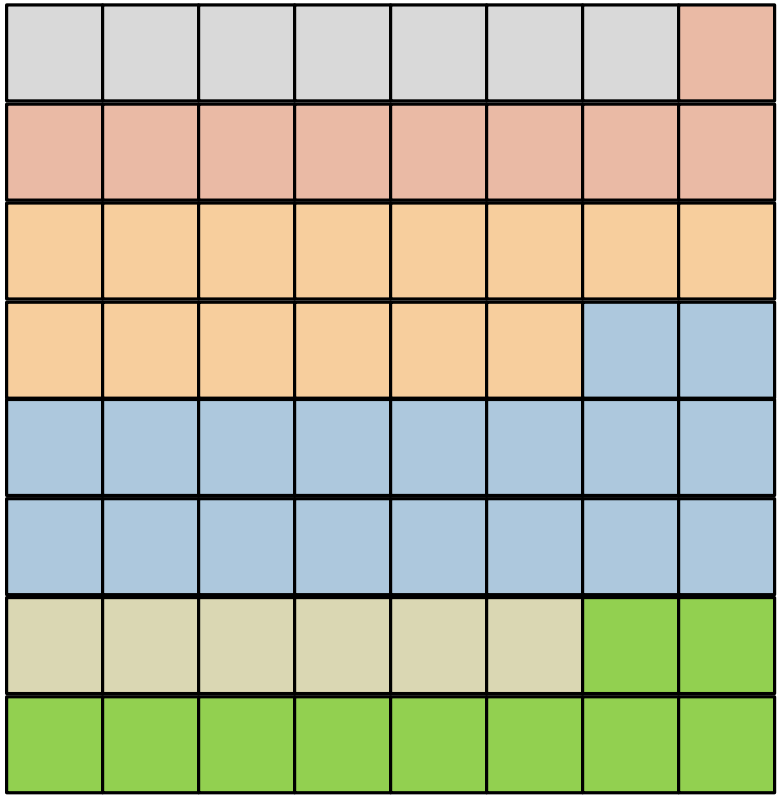
- MV predictor can be chosen as **Median(MV_1, MV_2, MV_3)**
  - There are horizontal and vertical components for a motion vector as $MV = \begin{bmatrix} mv_x \\ mv_y \end{bmatrix}$
  - MV Difference $MVD = \begin{bmatrix} mvd_x \\ mvd_y \end{bmatrix} = \begin{bmatrix} mv_x - mvp_x \\ mv_y - mvp_y \end{bmatrix}$
  - MVD is VLC-coded.

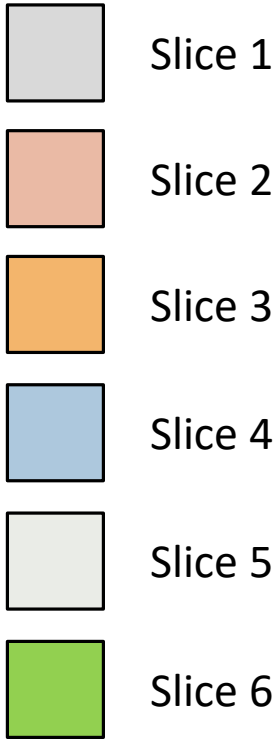


MV : Current motion vector
MV1 : Previous motion vector
MV2 : Above motion vector
MV3 : Above right motion vector
----- : Picture or GOB border



Slice Mode in Video Coding



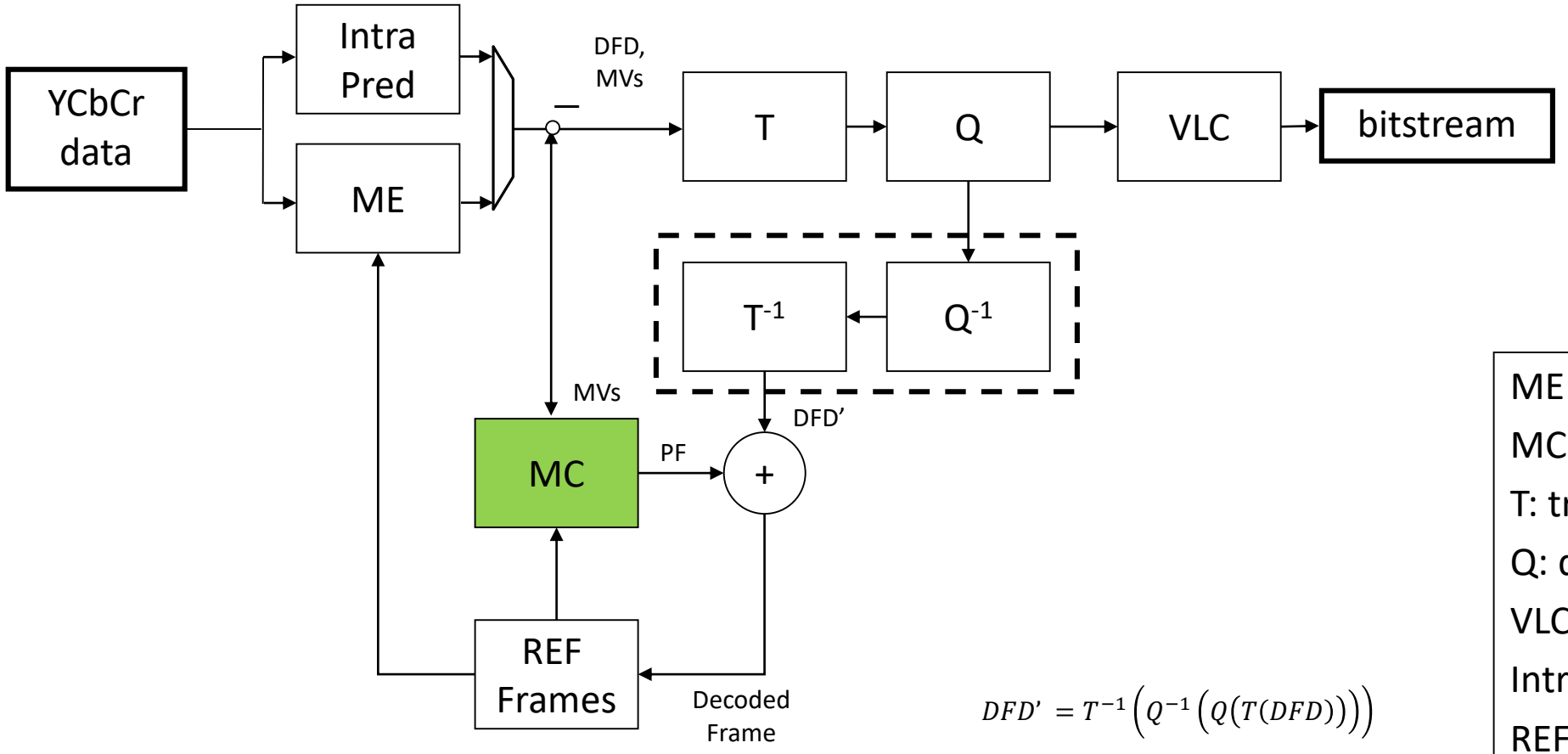


MV predictors can not cross different slices

Motion Compensation (MC)

- After ME, the current frame is predicted by blocks from the reference frame(s). The frame that consists of the predicted blocks can be called the predicted frame (PF).
- In MC, the current frame subtracted from the predicted frame can be called “motion-compensated residual (MCR)” or “displaced frame difference (DFD).”
- The MCR/DFD and MVs will be coded.

Motion Compensation

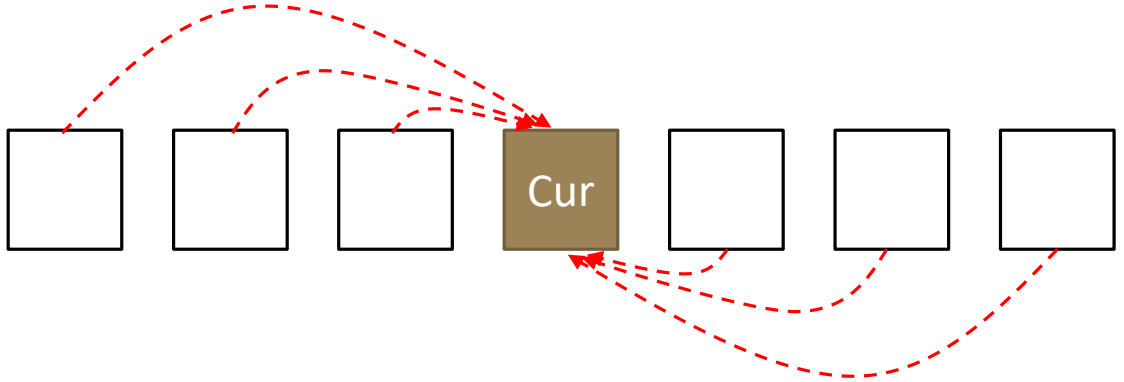


$$DFD' = T^{-1}\left(Q^{-1}\left(Q(T(DFD))\right)\right)$$

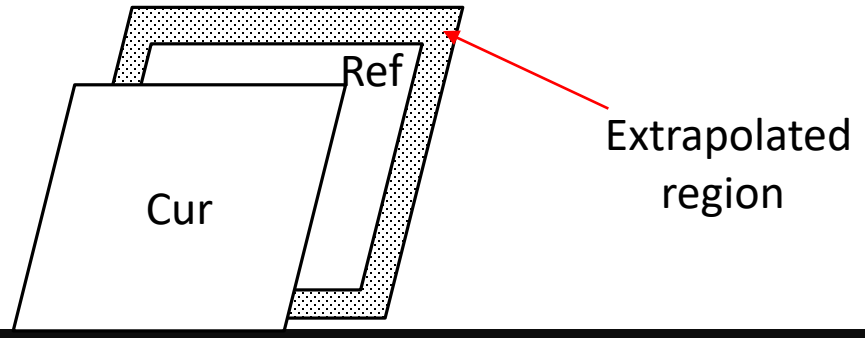
ME: motion estimation
MC: motion compensation
T: transform coding
Q: quantization
VLC: variable length coding
Intra Pred: Intra prediction
REF frame: reference frame

Misc. for ME

- Multiple Reference Frames
  - Allowing the encoder to choose a reference frame from a set of previously encoded-and-then-decoded frames



- Unrestricted motion vectors
  - Motion vectors pointing outside the frame boundaries



Forward/Backward

- Forward Prediction
  - It may fail when
    1. a significant time difference between the current and reference frames
    2. a scene change occurs
    3. moving objects appear or disappear in the reference frame
- Backward Prediction
  - It may help with the case 2 and 3 of forward prediction

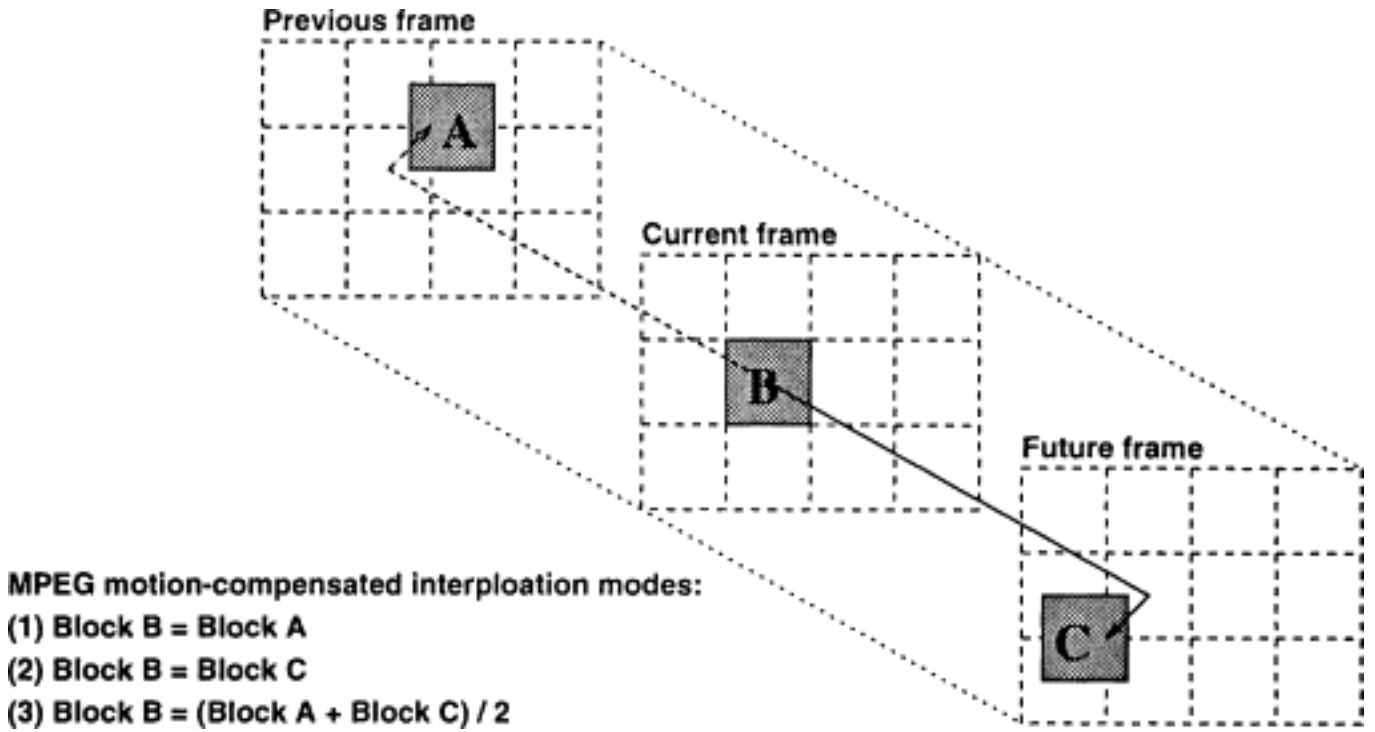
Bidirectional Prediction

- Bidirectional Prediction
  - It may outperform forward/backward prediction.
  - It carries out two motion estimation searches for each block, one of which is based on a previous frame and the other of which is on a future frame.
  - Once the two best matching blocks found, the final SAE is calculated by subtracting the average of the two matching blocks from the current block.
  - The encoder will choose which following mode works the best based on the SAE:
    - Forward prediction
    - Backward prediction
    - Bidirectional prediction

Bidirectional Prediction

- **Bidirectional prediction**
  - its performance is better than forward or backward prediction alone
- **Dual Motion Estimation Searches**:
  - For each block in the current frame, bidirectional prediction conducts two separate motion estimation searches:
    - **Forward Motion Estimation**: This search looks for a matching block in a previous frame.
    - **Backward Motion Estimation**: This search identifies a matching block in a future frame.
- **Selection of Best Matching Blocks**:
  - The process identifies the best matching blocks from the previous and future frames for each block in the current frame.
- **Calculation of SAE**:
  - Once the best matching blocks are found, the final SAE is computed by subtracting **the average of the two matching blocks** from the current block. The result represents the error or difference between the predicted block (from the average of the two matches) and the actual block in the current frame.
- **Mode Selection Based on SAE**:
  - The encoder then evaluates which prediction mode yields the lowest SAE for each block, thereby determining the most efficient way to encode the block. The options include:
    - **Forward Prediction**: Uses only the block from the previous frame.
    - **Backward Prediction**: Uses only the block from the future frame.
    - **Bidirectional Prediction**: Uses the average of blocks from previous and future frames.

Bidirectional prediction - Interpolation Mode



Hang et al., Motion Estimation for Image Sequence Compression, *in Handbook of Visual Communications*, 1995

Advantages of Bidirectional Prediction

- **Improved Compression Efficiency**: By using information from both previous and future frames, bidirectional prediction can more accurately reconstruct the current frame, reducing the amount of data needed to encode changes between frames.
- **Higher Quality Video Output**: This method can significantly reduce artifacts and improve the visual quality of the video, especially in sequences with complex motion patterns where forward or backward prediction alone might not be sufficient.
- **Flexibility and Adaptability**: The encoder's ability to choose between forward, backward, and bidirectional prediction modes on a block-by-block basis allows for a flexible approach that adapts to varying motion and content characteristics within a video sequence.

Bidirectional prediction is a key feature in modern video compression standards like H.264/AVC and HEVC, which are widely used in broadcasting, streaming, and other applications requiring efficient and high-quality video compression.