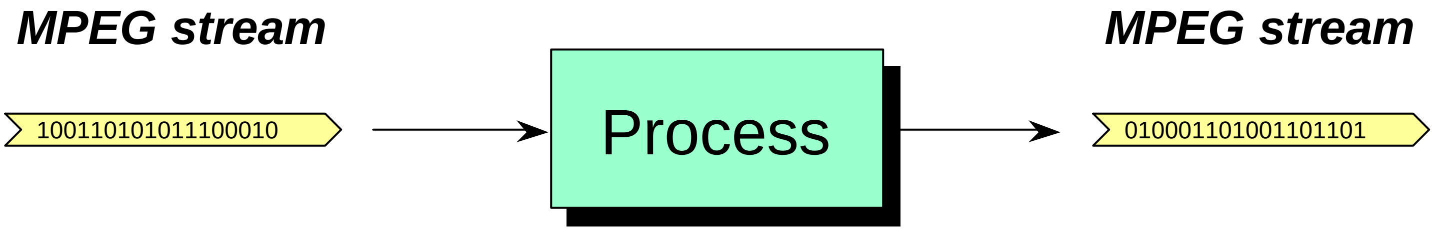
Compressed-Domain Video Processing

MIT 6.344
April 26, 2001

John Apostolopoulos and Susie Wee
Streaming Media Systems Group
Hewlett-Packard Laboratories

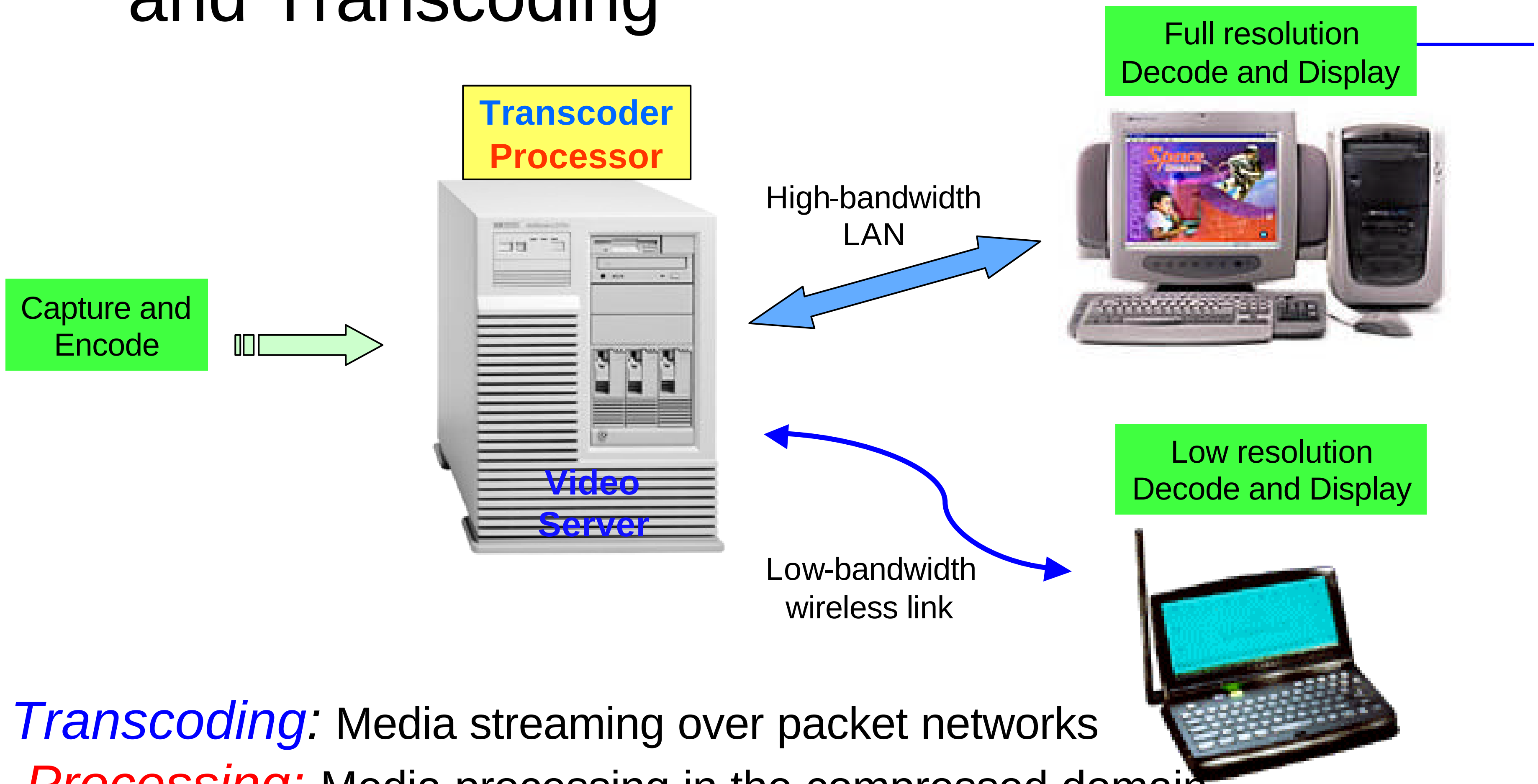
Problem Statement

Goal: Efficient signal processing of compressed MPEG video streams



High-quality video processing with low computational complexity and low memory requirements.

Compressed Domain Processing and Transcoding



Full resolution Decode and Display

Transcoder Processor

High-bandwidth LAN

Capture and Encode

Video Server

Low resolution Decode and Display

Low-bandwidth wireless link

Transcoding: Media streaming over packet networks

Processing: Media processing in the compressed domain

Outline

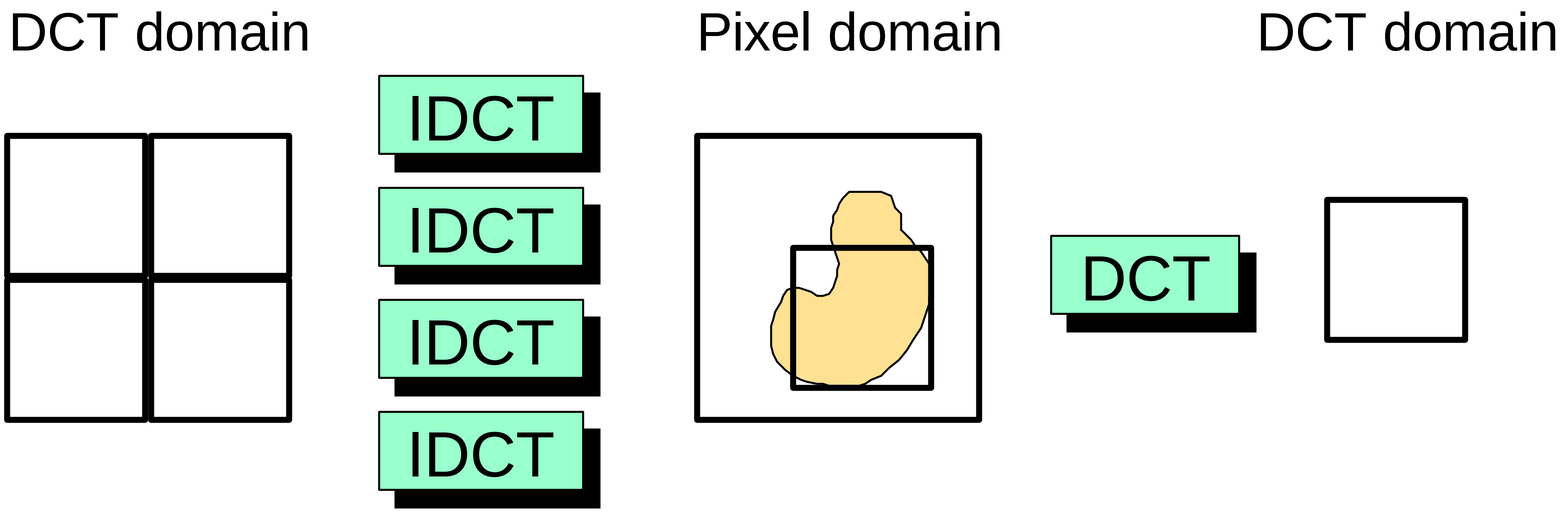
- Compressed-Domain Image Processing
- Compressed-Domain Video Processing
  - Review of MPEG basics
  - Manipulating temporal dependencies
  - Splicing
  - Reverse Play
  - MPEG-2 to H.263 Transcoding
- Review and Demo

CDP of Images

CDP Example

Basic translation problem:

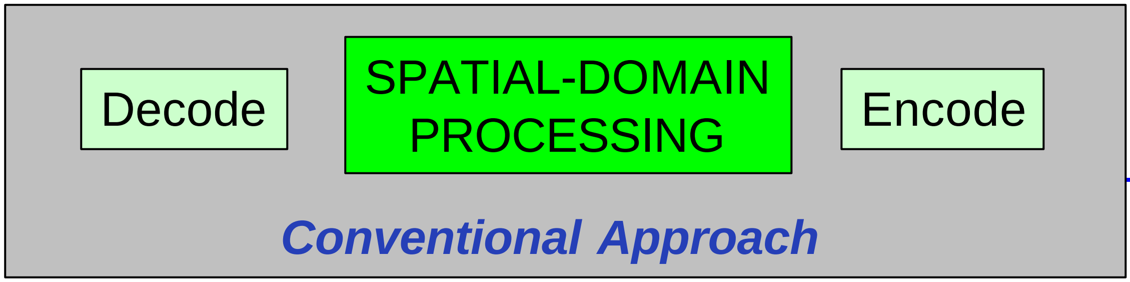
Given four 8x8 DCT blocks, find DCT of any 8x8 subblock.



DCT is good for compression,
but complicates many image processing tasks.

DCT-Domain Processing

- Many DCT-domain algorithms have been developed [Vasudev, Merhav, Shen] including: *translation, downscaling, filtering, masking, blue screen editing,* etc.
- Algorithms exploit:
  - *Signal processing properties* of DCT
  - *Matrix structures* used in fast DCT algorithms
  - *Sparseness* of quantized DCT coefficients
- *Exact* and *approximate* algorithms



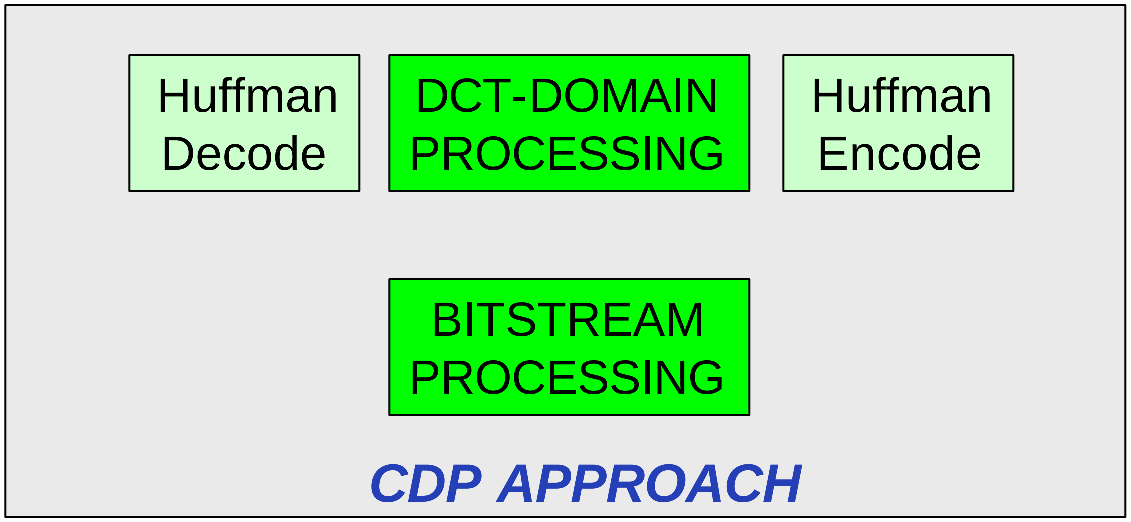
Decode

SPATIAL-DOMAIN PROCESSING

Encode

Conventional Approach

Compressed-Domain Processing



Huffman Decode

DCT-DOMAIN PROCESSING

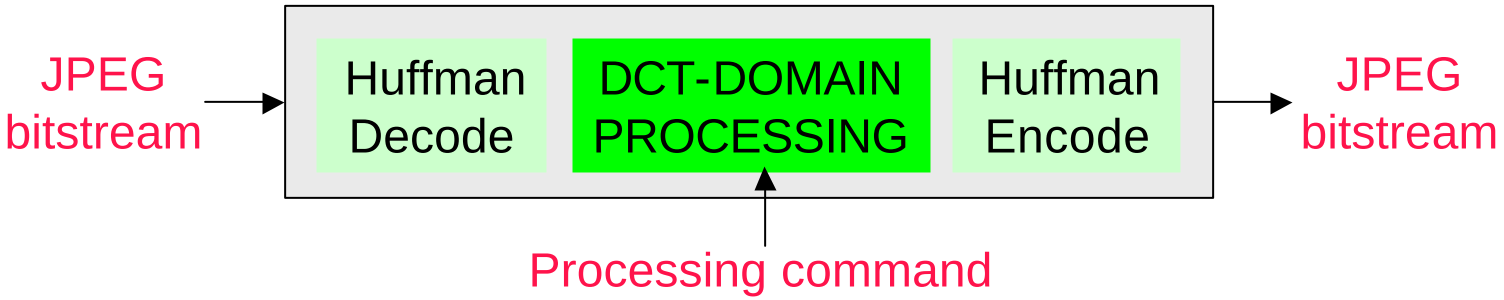
Huffman Encode

BITSTREAM PROCESSING

CDP APPROACH

GOAL: Perform equivalent spatial-domain image/video processing tasks using fast algorithms that operate directly on the compressed-domain data.

DCT-Domain Processing



Basic Concepts:

- Express the spatial-domain processing as a sequence of matrix operators
- Use the distributive property of the DCT to develop the equivalent DCT-domain operation
- Use a sparse matrix factorization to derive a fast algorithm

DCT-Domain Processing

- *Spatial Domain*: IDCT-Process-DCT
  - – Fast DCT/IDCT based on efficient factorization by Arai, et. al. requires 13 mults, 29 adds.

    IDCT $T' x = A_3'A_2'A_1' M' B_2' B_1' P' D' x$

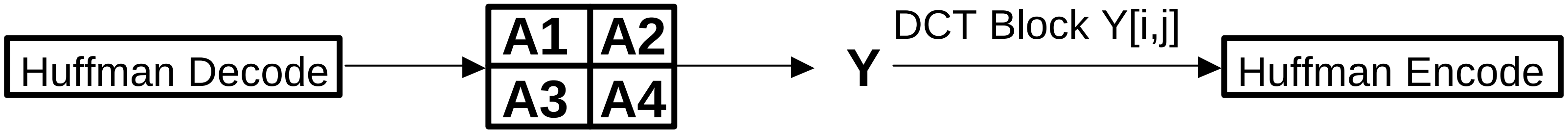
    Process $y = F T' x$

    DCT $T y = D P B_1 B_2 M A_1 A_2 A_3 y$
- *Compressed Domain*: partial decompression

$$TFT'x = DPB_1B_2MA_1A_2A_3FA_3'A_2'A_1'M'B_2'B_1'P'D'x$$

DCT-Domain Downscaling

Method:



$$Y = (S \; T) \begin{bmatrix} A1 & A2 \\ A3 & A4 \end{bmatrix} \begin{bmatrix} S^T \\ T^T \end{bmatrix}$$

S and T are fixed 8x8 matrices that differ only in the signs of their entries. For fast implementations, entries in S and T can be approximated by rounding off the entries to 0, 1/8, 1/4 and 1/2.

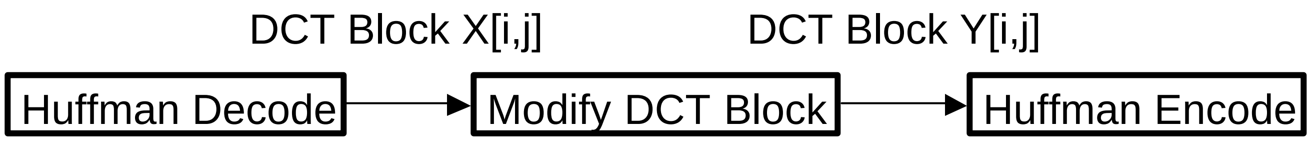
Downscale by 3 (use 9 blocks A1,A2,...,A9) and Downscale by 4 (use 16 blocks A1,A2,...,A16) can be performed in a similar manner.

DCT-Domain Downscaling

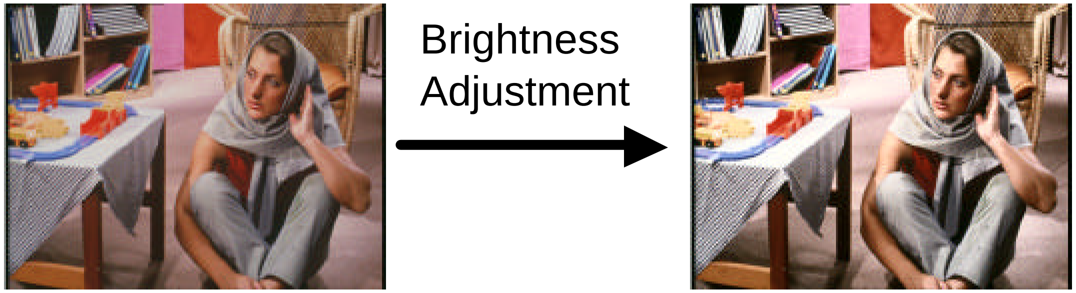
Downscaling by two	Cycles per output 8x8 block
Spatial domain	
- four 8x8 inverse DCT,	**5376**
- pixel average,	
- one 8x8 DCT	
DCT Domain	
- Smith	> 7700
- Chang	6192
- Our approach#1	2824
- Our approach (exact S, T)	3392
- Our approach (approx. S,T)	**880**
DCT Domain - sparse data	
- Smith	> 5250
- Chang	2096
- Our approach#1	902
- Our approach (exact S,T)	1752
- Our approach (approx. S,T)	**528**

Downscaling by 3:	
Spatial domain Approach	9392 ops
DCT Domain Approach	5728 ops
Downscaling by 4:	
Spatial Domain Approach	16224 ops
DCT Domain Approach	8224 ops

DCT-Domain Image Enhancement



Brightness Adjustment:



Spatial Domain:

y = x + b

64 adds per 8x8 block

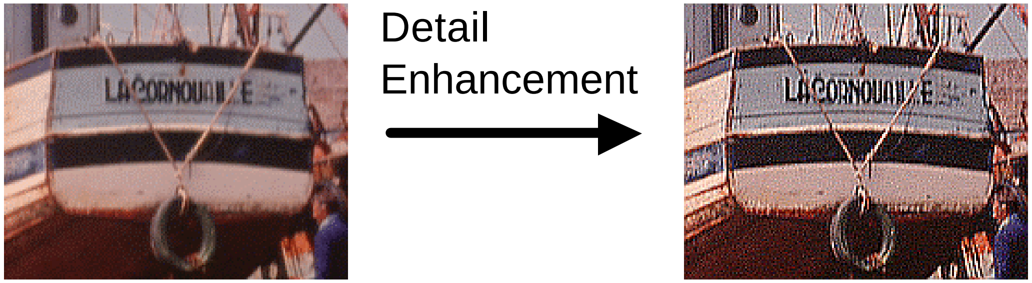
Transform Domain:

Y[i,j] = X[i,j] + B, i = j = 0

= X[i,j] elsewhere

1 add per 8x8 block

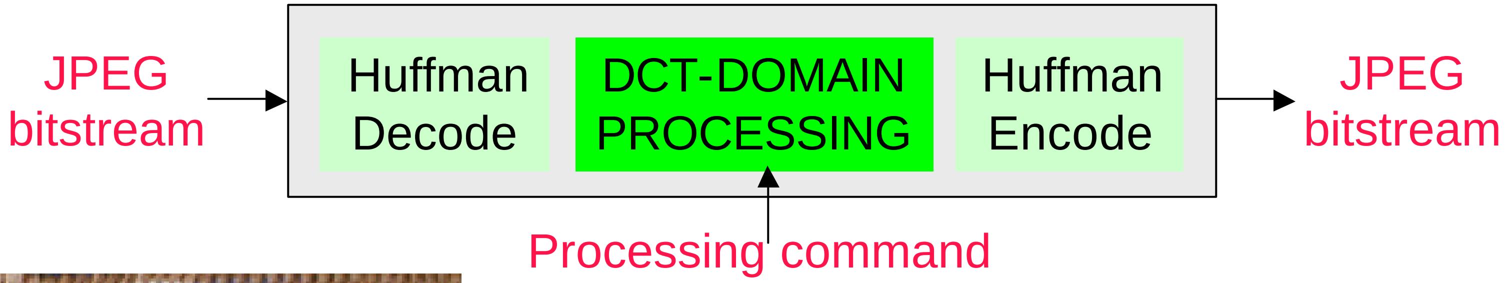
Detail Enhancement:



Transform Domain:

Manipulate X[i,j] for all (i,j) except (i = j = 0)

JPEG CDP Examples

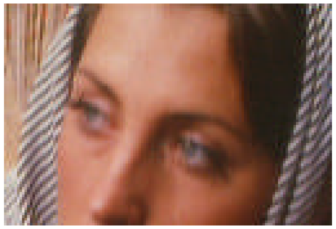


JPEG bitstream → Huffman Decode | DCT-DOMAIN PROCESSING | Huffman Encode → JPEG bitstream

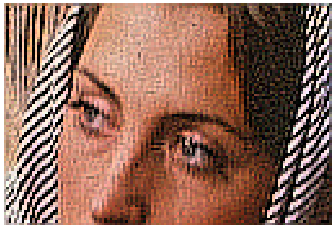
Processing command

Original JPEG Image

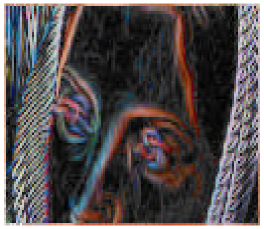
Downscaling
14X speedup



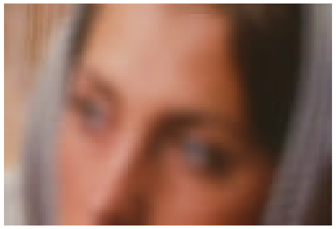
Tiling
4.5X speedup



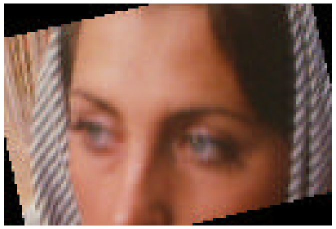
Sharpening
8X speedup



Edge detection
5X speedup



Filtering
2-4X speedup

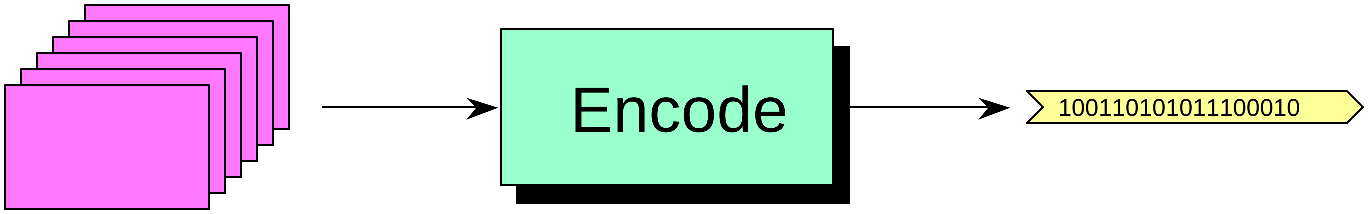


Rotate
1.5-3X speedup

Speedup data based on actual implementations.

CDP of Video

Video Compression



Raw video *MPEG stream*

HDTV quality

720 x 1280 pixels, 60 fps
~ 1 Gbps

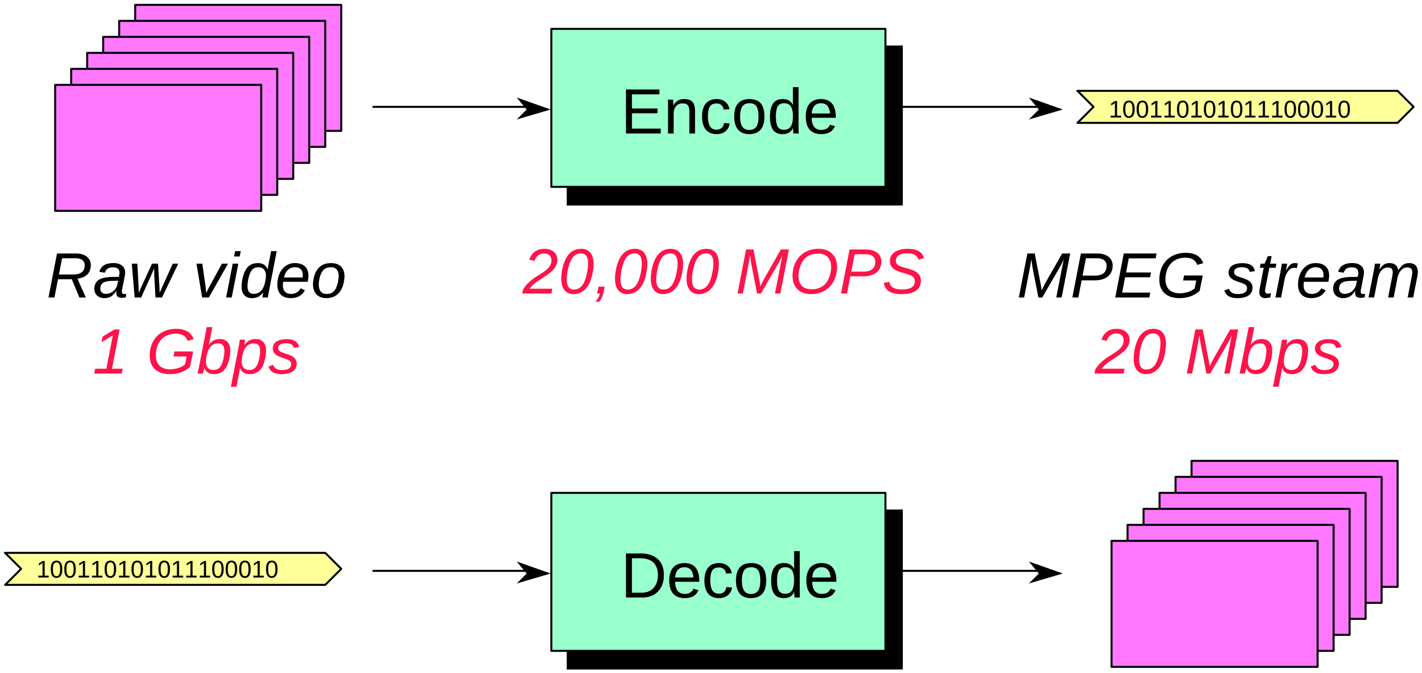
20 Gbytes / 2 hour movie
~ 20 Mbps

Broadcast quality

480 x 720 pixels, 30 fps
~ 250 Mbps

5 Gbytes / 2 hour movie
~ 5 Mbps

MPEG Video Compression



Encode

1001101010111100010

Raw video

1 Gbps

20,000 MOPS

MPEG stream

20 Mbps

1001101010111100010

Decode

MPEG stream

20 Mbps

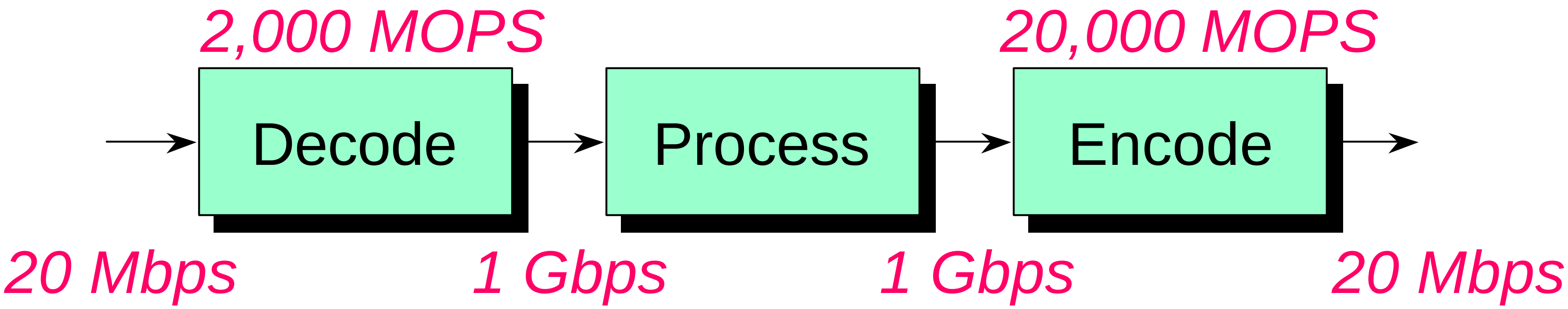
2,000 MOPS

Raw video

1 Gbps

Problem statement

How do we process compressed video streams?



- Difficulties:
  - Computational requirements: 22,000 MOP overhead from decode and encode operations (plus additional processing)
  - Bandwidth requirements: Need to process uncompressed data at 1 Gbps
  - Quality issues: Even without processing, the decode/encode cycle causes quality degradation.

HEWLETT PACKARD

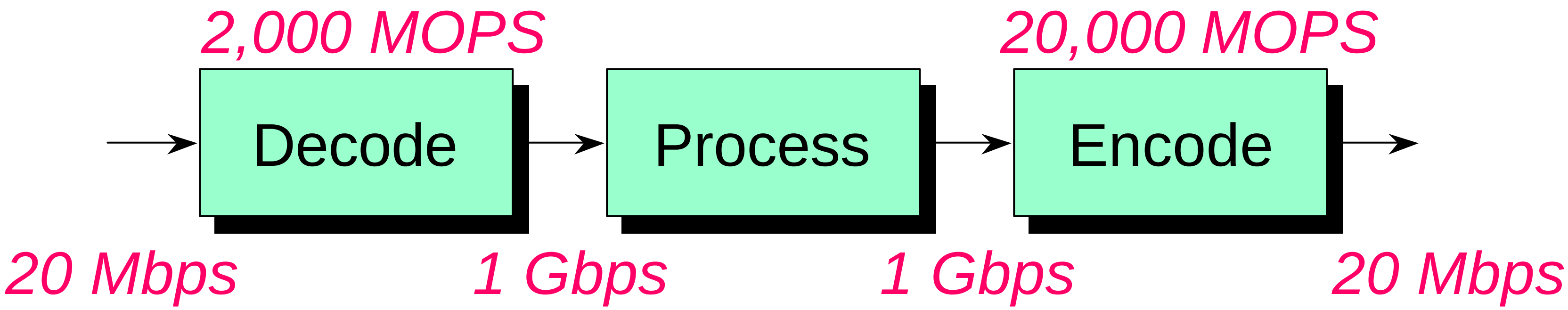
John Apostolopoulos
April 26, 2001

MPEG CDP

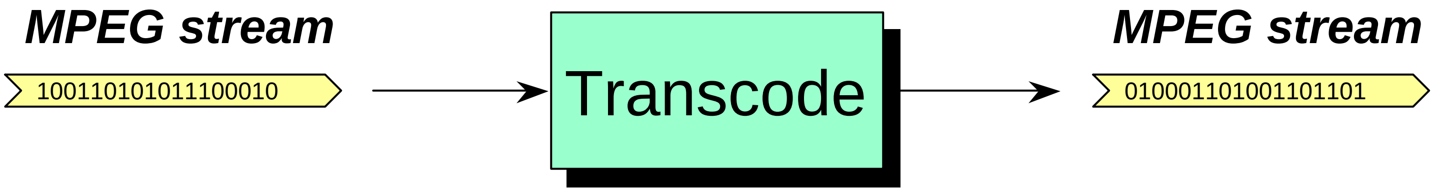
How do we process compressed video streams?

Use efficient compressed-domain processing (CDP) algorithms.

Naive Solution:



Ideal Solution:

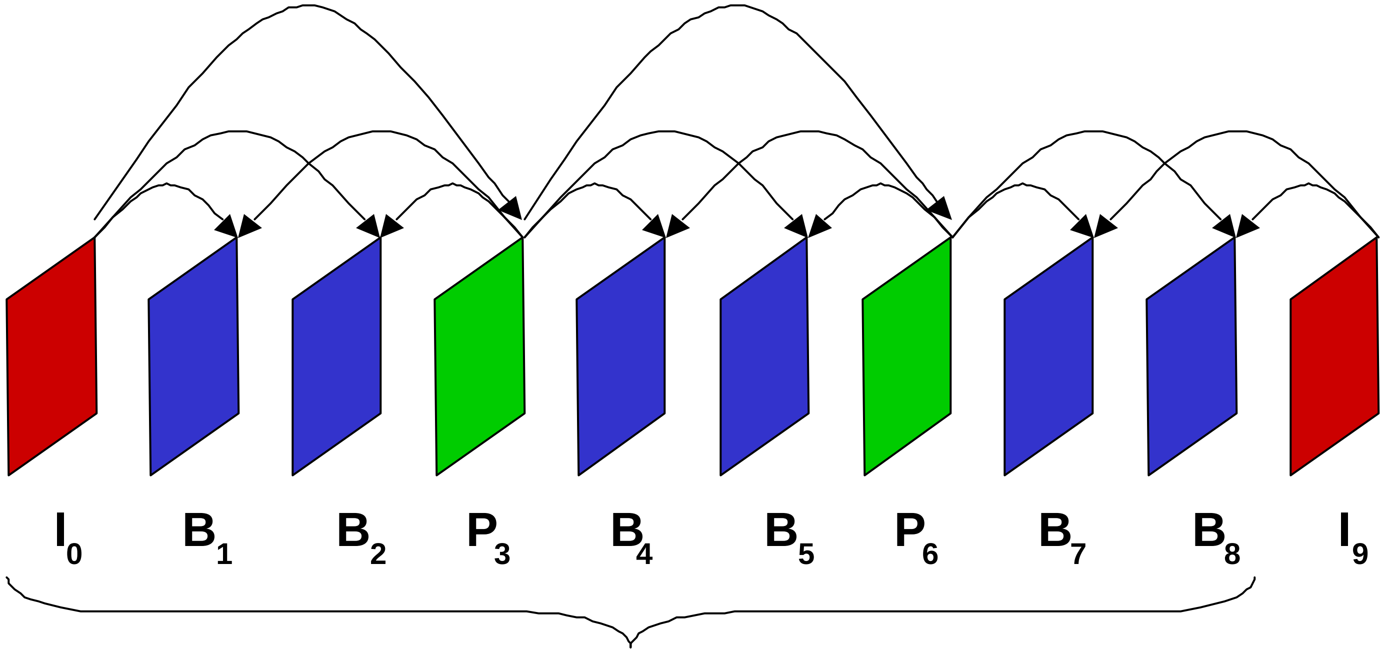


Outline

- Compressed-Domain Image Processing
- Compressed-Domain Video Processing
  - → Review of MPEG basics
  - Manipulating temporal dependencies
  - Splicing
  - Reverse Play
  - MPEG-2 to H.263 Transcoding
- Review and Demo

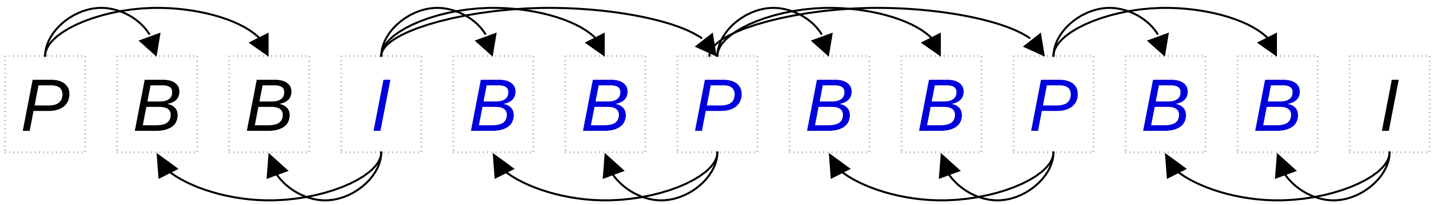
MPEG Group of Pictures (GOP) Structure

- Composed of I, P, and B frames
- Arrows show prediction dependencies
- Periodic I-frames enable random access

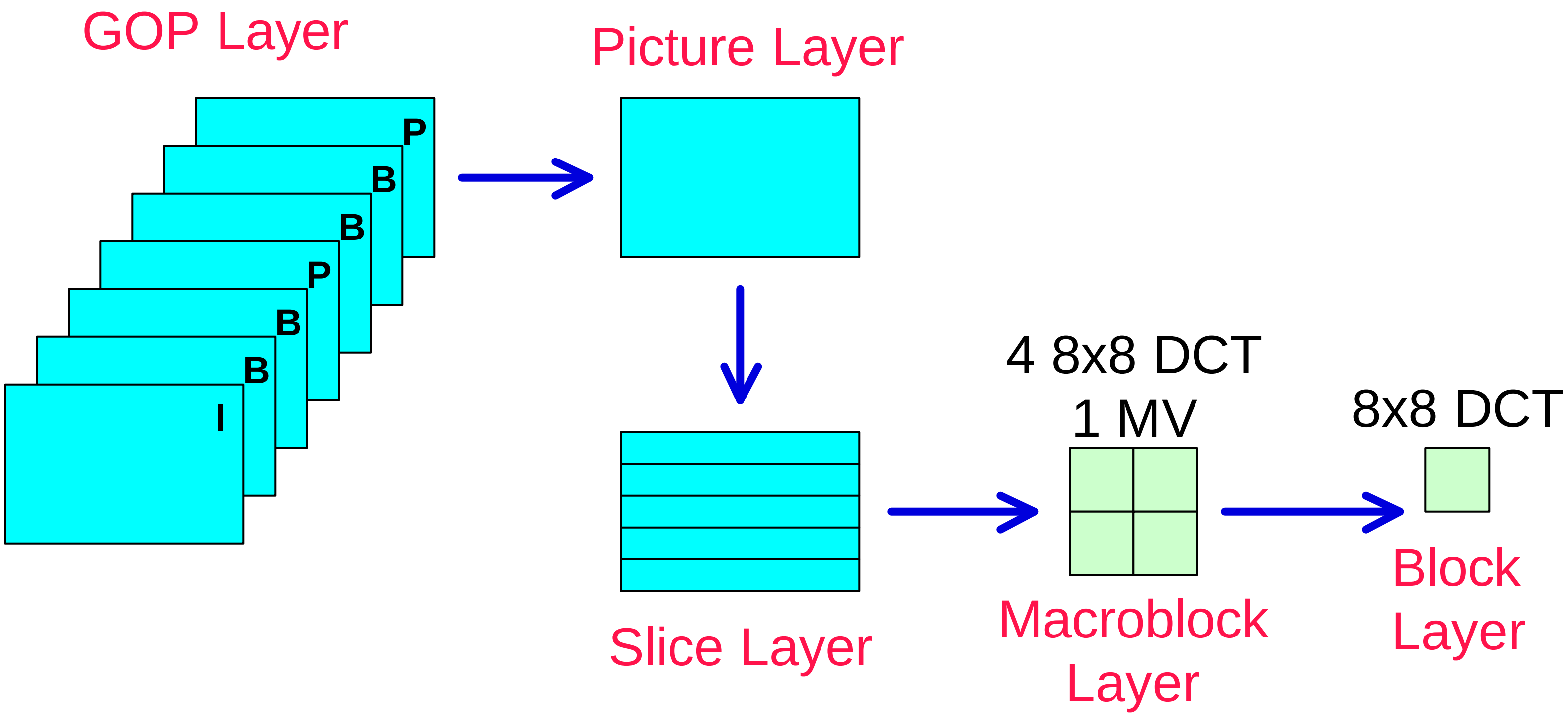


MPEG GOP

MPEG Structures



P B B I B B P B B P B B I



GOP Layer

Picture Layer

Slice Layer

4 8x8 DCT
1 MV

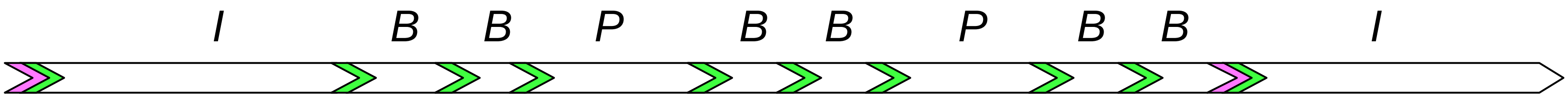
Macroblock Layer

8x8 DCT

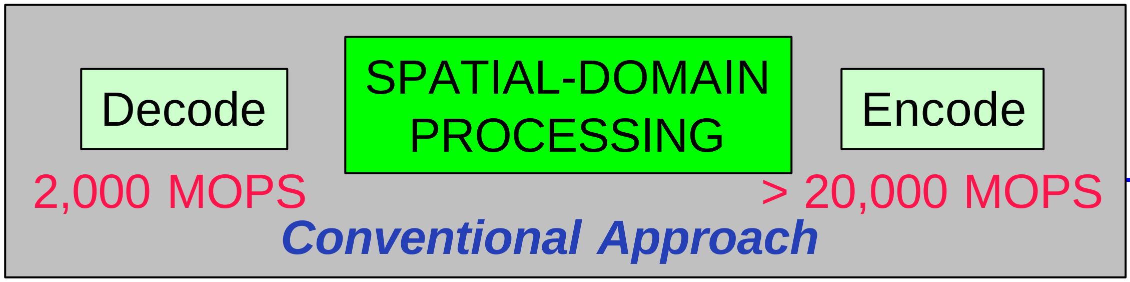
Block Layer

HEWLETT PACKARD

MPEG Bitstream



- GOP header >
- Picture header >
- Picture data 10110010
- Pictures in coding order
- Start codes (seq, GOP, pic, and slice start codes; seq end code)
  - Unique byte-aligned 32-bit patterns
  - *0x000001nm*   23 zeros, 1 one, 1-byte identifier
  - Enables random access



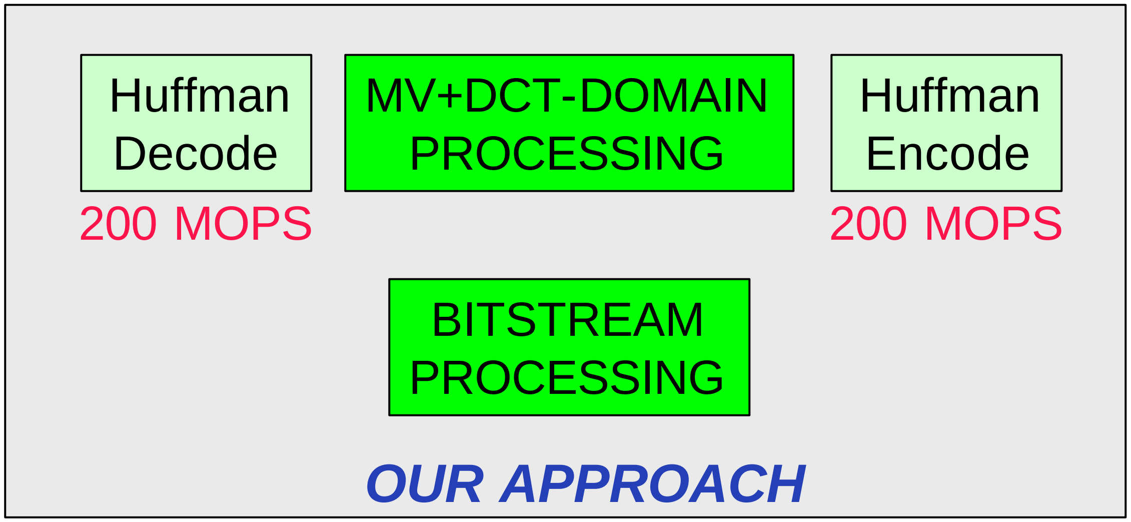
Decode | SPATIAL-DOMAIN PROCESSING | Encode

2,000 MOPS

> 20,000 MOPS

Conventional Approach

Compressed-Domain Processing



Huffman Decode | MV+DCT-DOMAIN PROCESSING | Huffman Encode

200 MOPS

200 MOPS

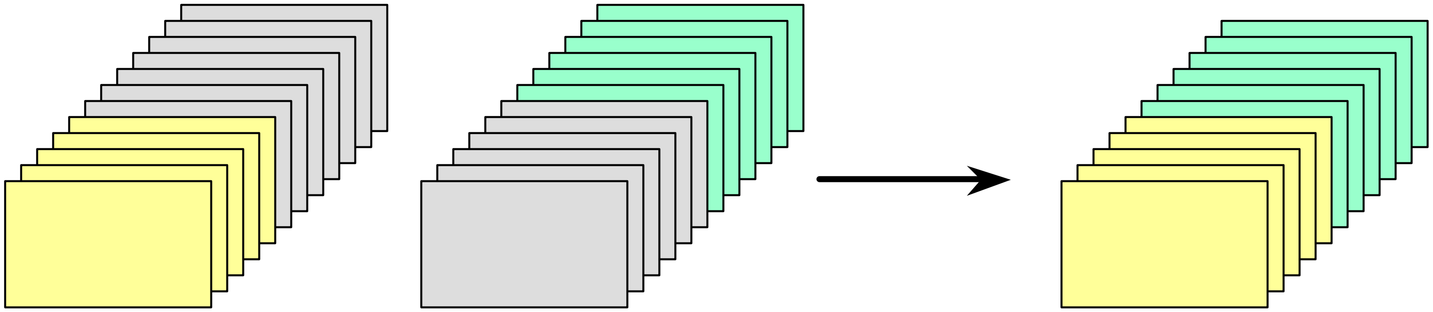
BITSTREAM PROCESSING

OUR APPROACH

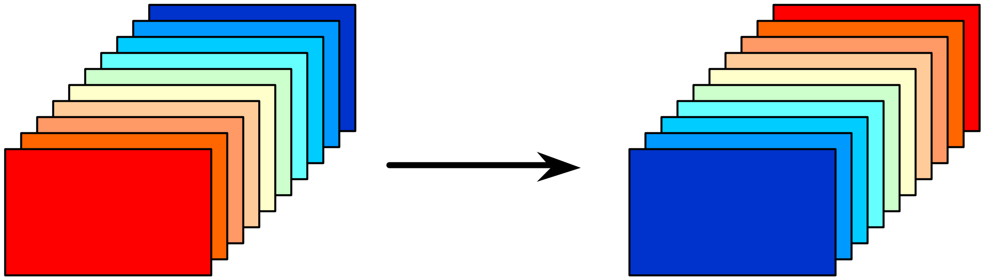
Develop algorithms to perform equivalent spatial-domain video processing tasks using fast algorithms that operate directly on the compressed-domain data.

Frame-Level Video Processing

Splicing



Reverse Play



Outline

- Compressed-Domain Image Processing
- Compressed-Domain Video Processing
  - Review of MPEG basics
  - Manipulating temporal dependencies
  - Splicing
  - Reverse Play
  - MPEG-2 to H.263 Transcoding
- Review and Demo

Manipulating Temporal Dependencies in Compressed Video

- Video compression uses temporal prediction

  → Prediction dependencies in coded data
- Many applications require changes in the dependencies
- Manipulating (modifying) temporal dependencies in the compressed video [Wee]
  - Adding
  - Removing
  - Changing

Temporal Prediction

Original Video Frames: $\mathbf{F} = \{ F_1, F_2, \ldots, F_n \}$

Coded Video Frames: $\hat{\mathbf{F}} = \{ \hat{F}_1, \hat{F}_2, \ldots, \hat{F}_n \}$

Each frame F_i is coded with two components

1. Prediction $P_i(\hat{\mathbf{A}}_\mathbf{i}, S_i)$

   Anchor frames $\hat{\mathbf{A}}_\mathbf{i} \subseteq \{ \hat{F}_1, \hat{F}_2, \ldots, \hat{F}_{i-1} \}$

   Side information S_i

2. Residual $R_i = F_i - P_i(\hat{\mathbf{A}}_\mathbf{i}, S_i)$

Coded Residual $\hat{R}_i$

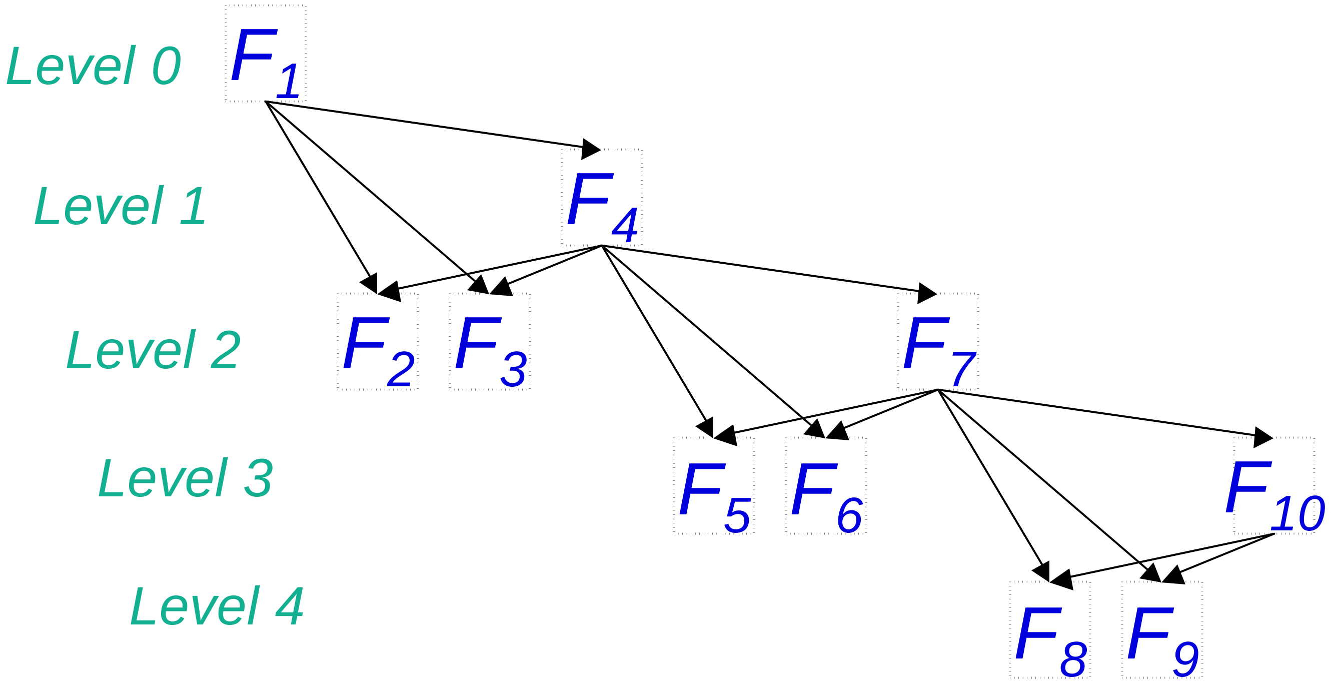
Reconstructed Frame $\hat{F}_i = P_i(\hat{\mathbf{A}}_\mathbf{i}, S_i) + \hat{R}_i$

Prediction Dependencies

Each coded frame is *dependent* upon its anchor frames.

$$\hat{F}_i = P_i\left(\hat{\mathbf{A}}_i, S_i\right) + \hat{R}_i$$

Prediction Depth



Level 0 F_1

Level 1 F_4

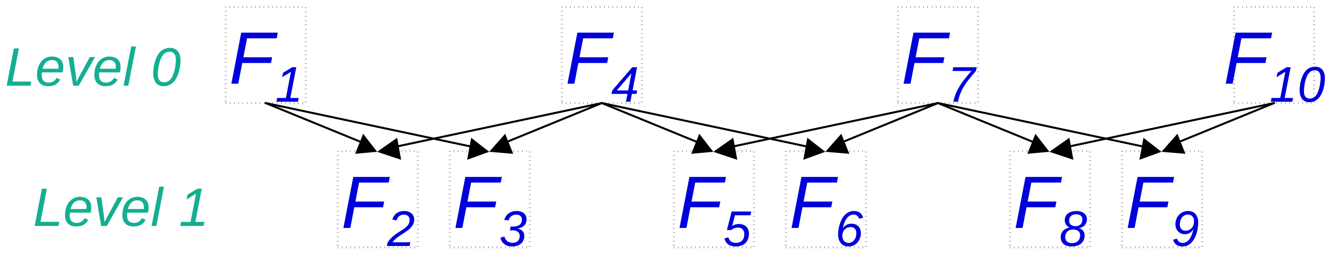
Level 2 F_2 F_3 F_7

Level 3 F_5 F_6 F_{10}

Level 4 F_8 F_9

Manipulating Dependencies

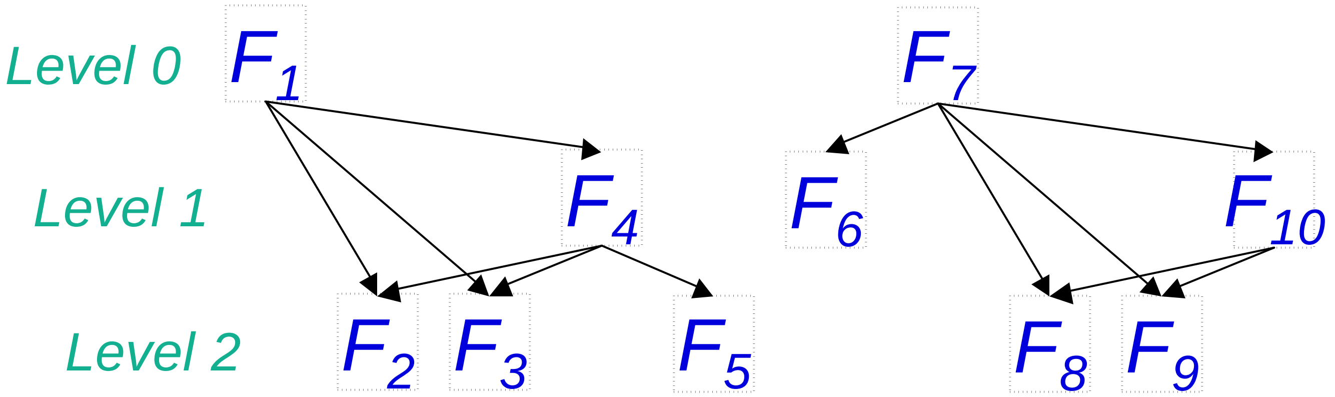
Fewer levels of dependence



Level 0 F_1 F_4 F_7 F_{10}

Level 1 F_2 F_3 F_5 F_6 F_8 F_9

Remove dependence between frames



Level 0 F_1 F_7

Level 1 F_4 F_6 F_{10}

Level 2 F_2 F_3 F_5 F_8 F_9

Problem Formulation

Compression algorithm has a set of prediction rules.

Choose prediction modes during encoding.

$$\hat{\mathbf{A}}_{\mathbf{i}}, S_i \qquad\qquad \hat{\mathbf{A}}'_{\mathbf{i}}, S'_i$$

$$\hat{F}_i = P_i\left(\hat{\mathbf{A}}_{\mathbf{i}}, S_i\right) + \hat{R}_i \qquad\qquad \hat{F}'_i = P'_i\left(\hat{\mathbf{A}}'_{\mathbf{i}}, S'_i\right) + \hat{R}'_i$$

Each choice yields:

- different compressed representation of frame F_i ,
- different distribution between P & R components,
- different set of prediction dependencies.

Manipulating Dependencies

Given a compressed representation with dependencies $\hat{\mathbf{A}}_{\mathbf{i}}, S_i$

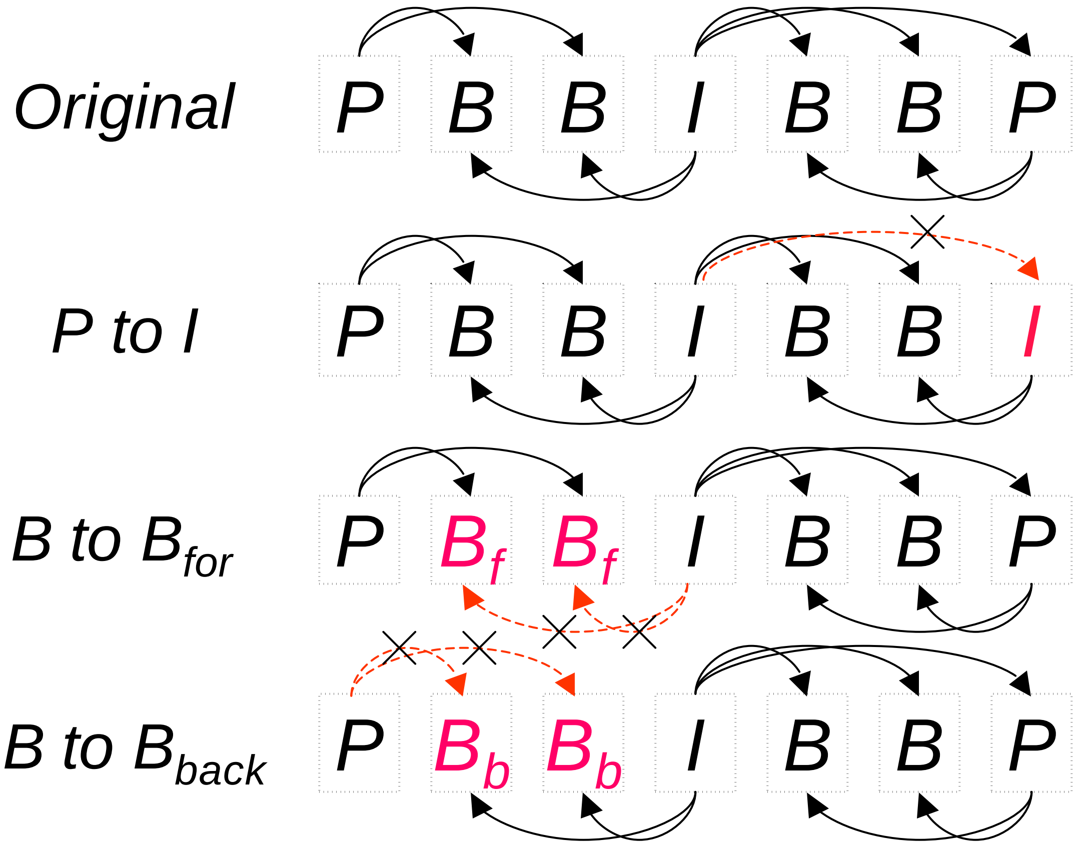
how do we create a new compressed representation with dependencies $\hat{\mathbf{A}}'_{\mathbf{i}}, S'_i$?

- Reconstruct frames $\hat{F}_i = P_i\left(\hat{\mathbf{A}}_{\mathbf{i}}, S_i\right) + \hat{R}_i$
- Compressed-domain approximation $\hat{F}_i \approx F_i$
- Calculate new prediction and residual

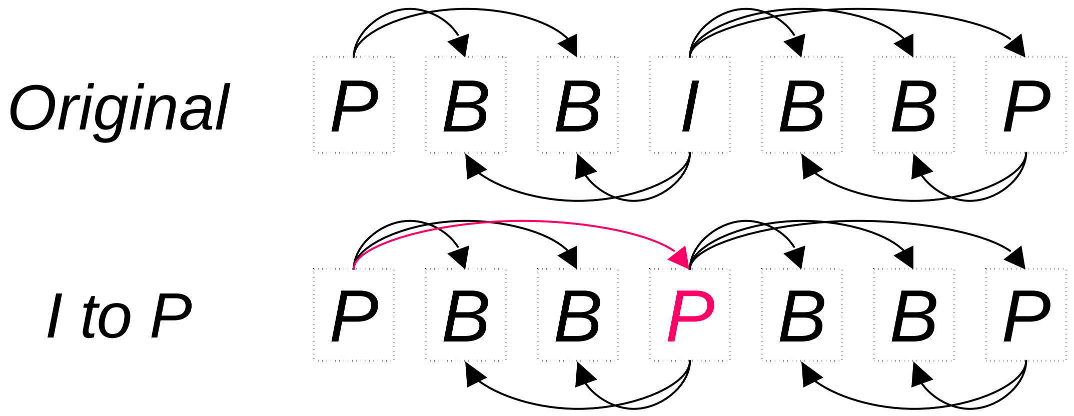
$$R'_i = F_i - P'_i\left(\hat{\mathbf{A}}'_{\mathbf{i}}, S'_i\right)$$

$$R'_i \approx \hat{R}_i + P_i\left(\hat{\mathbf{A}}_{\mathbf{i}}, S_i\right) - P'_i\left(\hat{\mathbf{A}}'_{\mathbf{i}}, S'_i\right)$$

Frame Conversions: Remove Dependencies



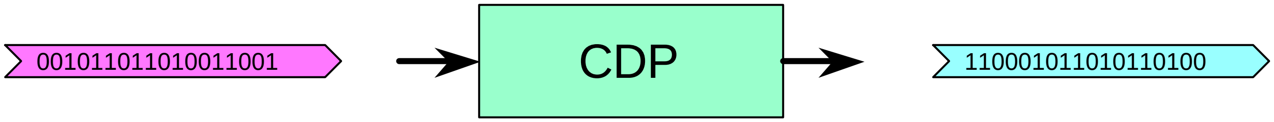
Frame Conversion: Add Dependencies



1. Find and code new MVs. *MV Resampling*
2. Calculate new prediction.
3. Calculate and code new residual.

Compressed-Domain Processing

MPEG standard addresses prediction rules, buffer requirements, coding order, and bitstream syntax.



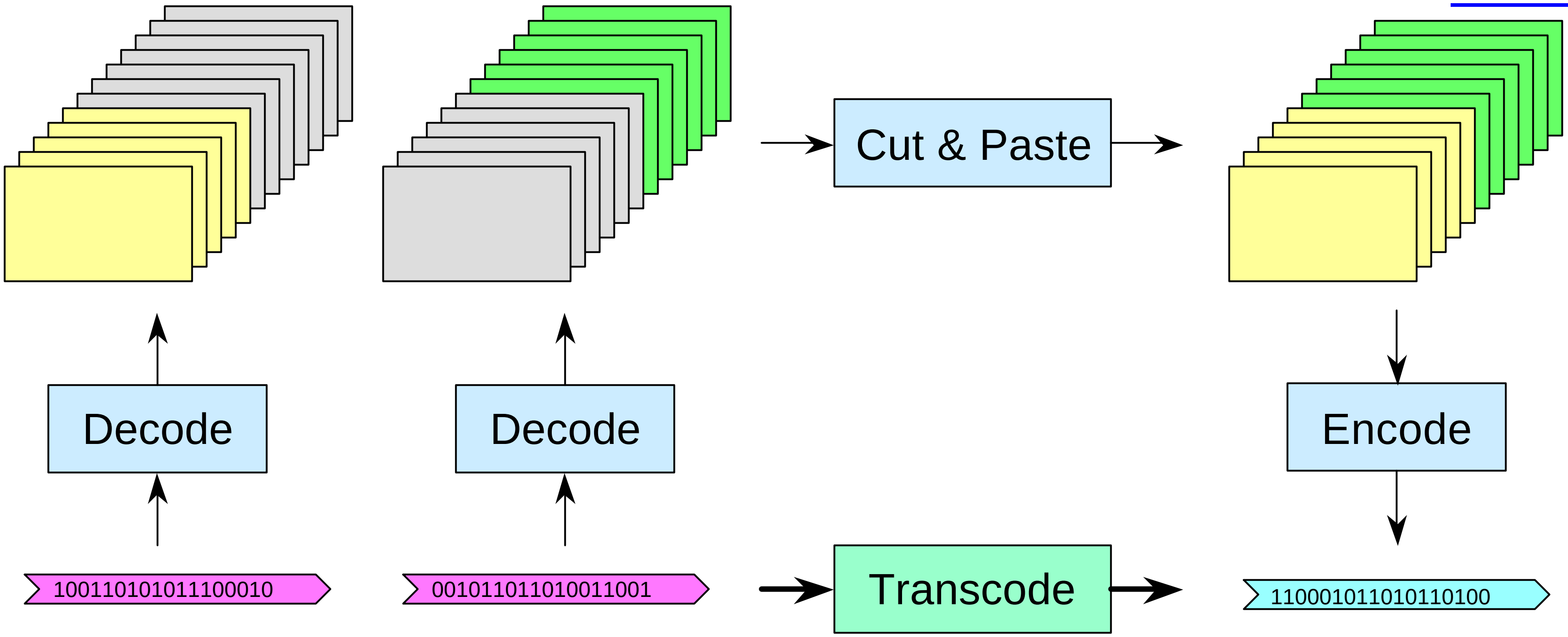
MPEG CDP steps

- Determine and perform appropriate frame conversions (i.e. manipulate dependencies).
- Reorder data.
- Perform rate matching.
- Update header information.

Outline

- Compressed-Domain Image Processing
- Compressed-Domain Video Processing
  - Review of MPEG basics
  - Manipulating temporal dependencies
  - → Splicing
  - Reverse Play
  - MPEG-2 to H.263 Transcoding
- Review and Demo

Compressed-Domain Splicing

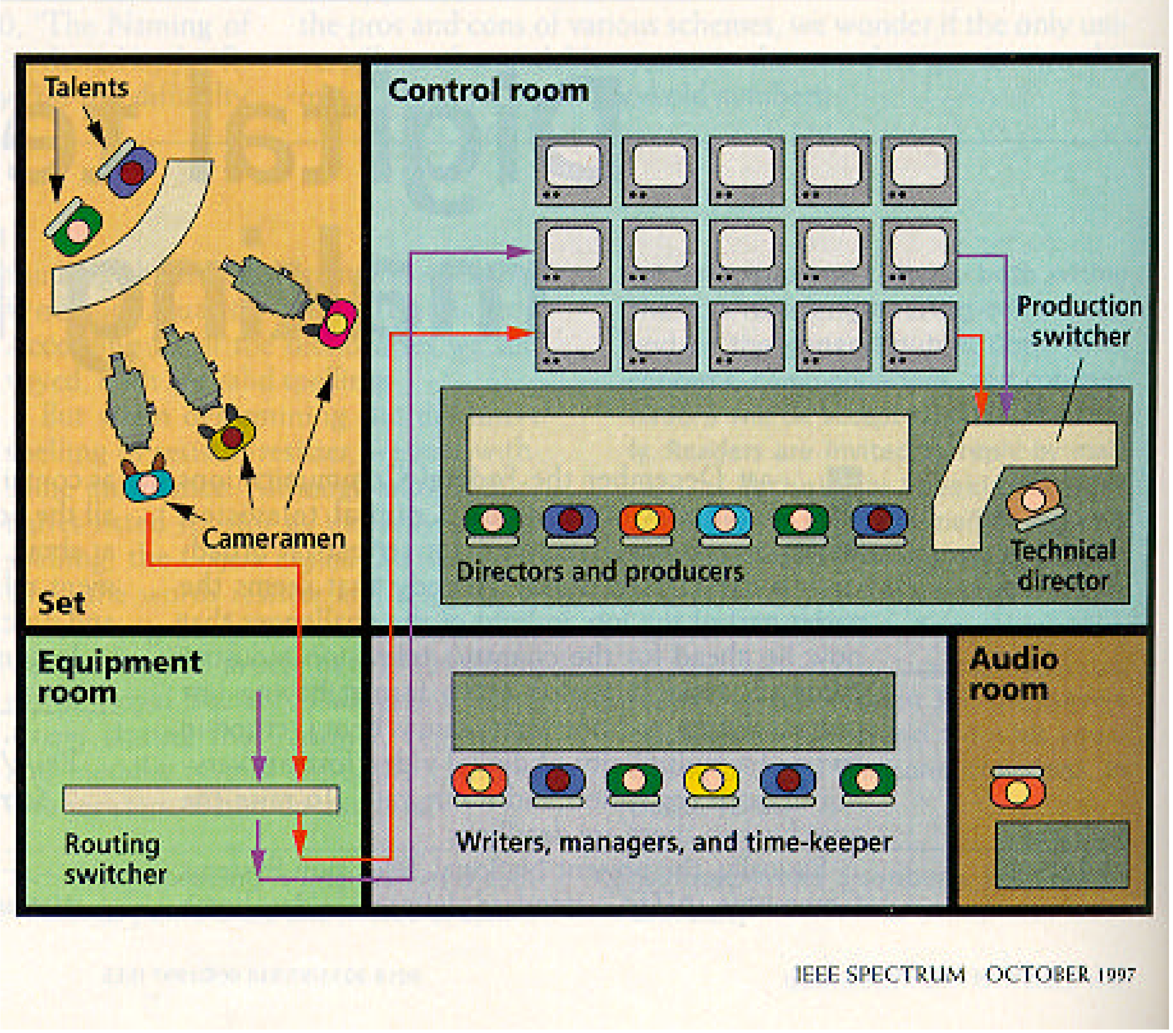


Application: Ad Insertion for DTV

Splicing: SMPTE standard

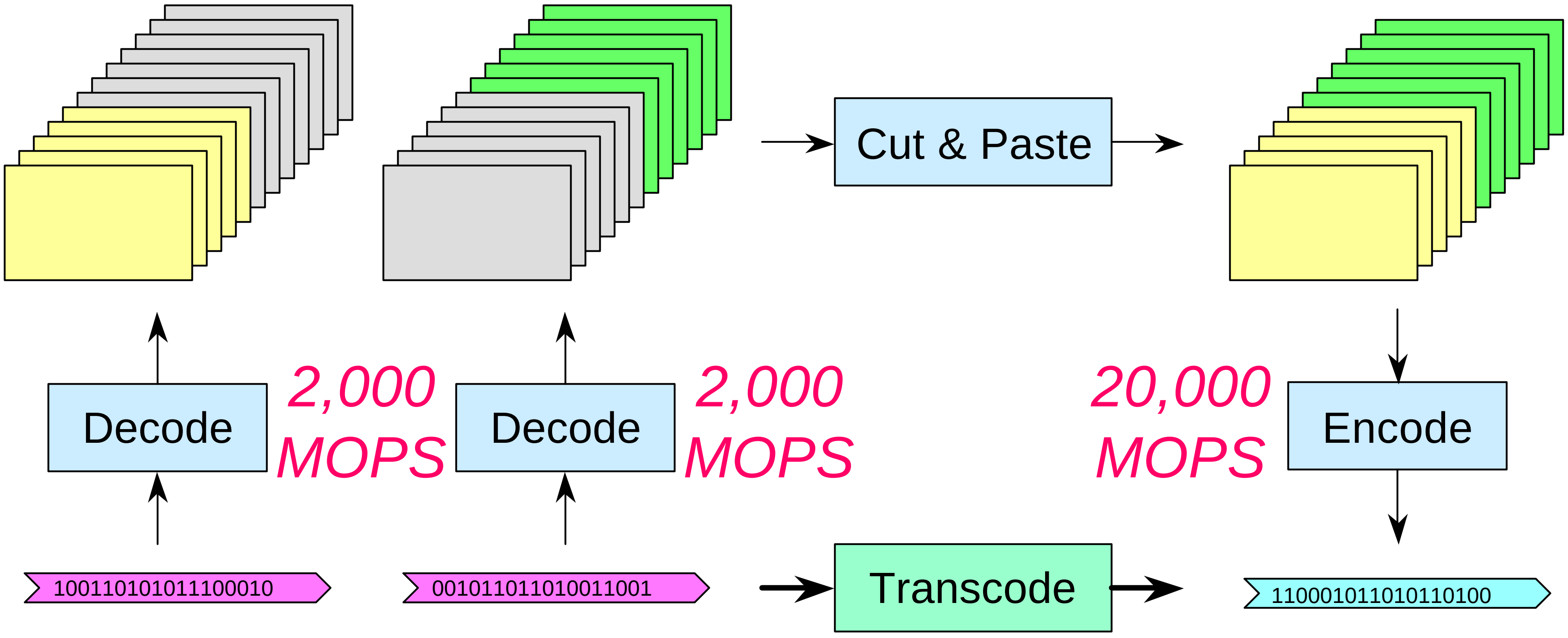
- SMPTE formed a committee on splicing.
- *Disadvantages:*
  - User must predefined splice points during encoding
    - ⇒ Complicated encoders.
  - Splice points can only occur on I frames
    - Not frame accurate.
- *Advantages:*
  - Simple cut-and-paste operation.

The TV Newsroom **IEEE Spectrum**



IEEE SPECTRUM OCTOBER 1997

Compressed-Domain Splicing



Cut & Paste

Decode

2,000 MOPS

Decode

2,000 MOPS

20,000 MOPS

Encode

1001101010111100010

0010110110100110011001

1100010110101110100

Transcode

< 2,000 MOPS

for one splice point every sec

Compressed-Domain Processing

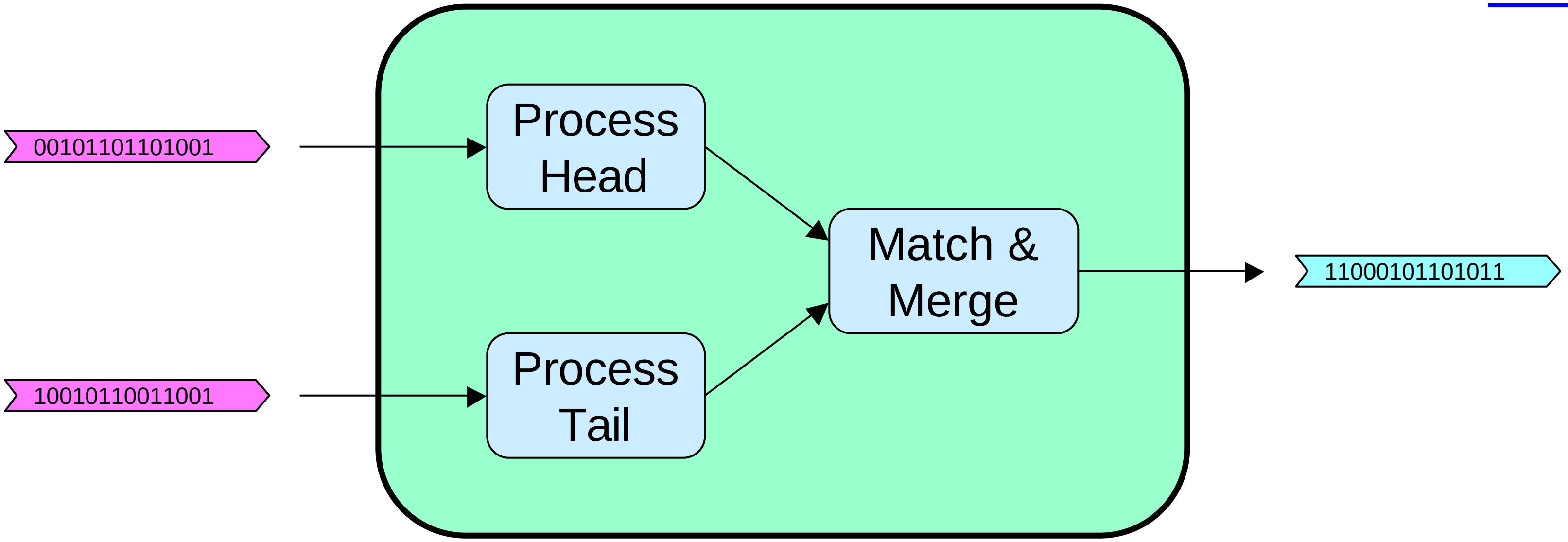
Can we do better by exploiting properties of

1) the MPEG compression algorithm

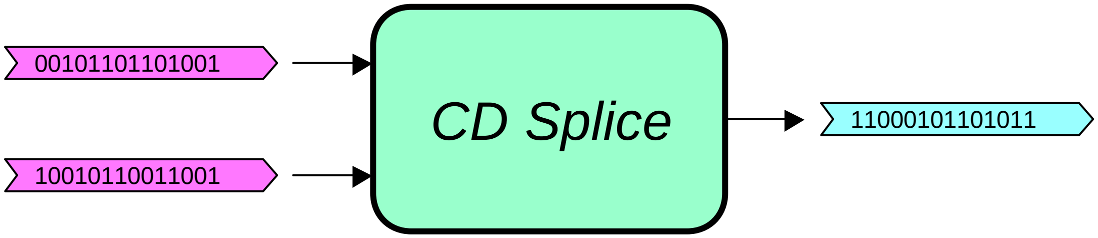
and

2) the splicing operation ?

Splicing: Our approach



< 2,000 MOPS *for one splice point every sec*



Splicing Algorithm (simplified overview)

Only process the GOPS affected by the splice.

Step 1: Process the head stream.

- Remove (backward) dependencies on dropped frames.

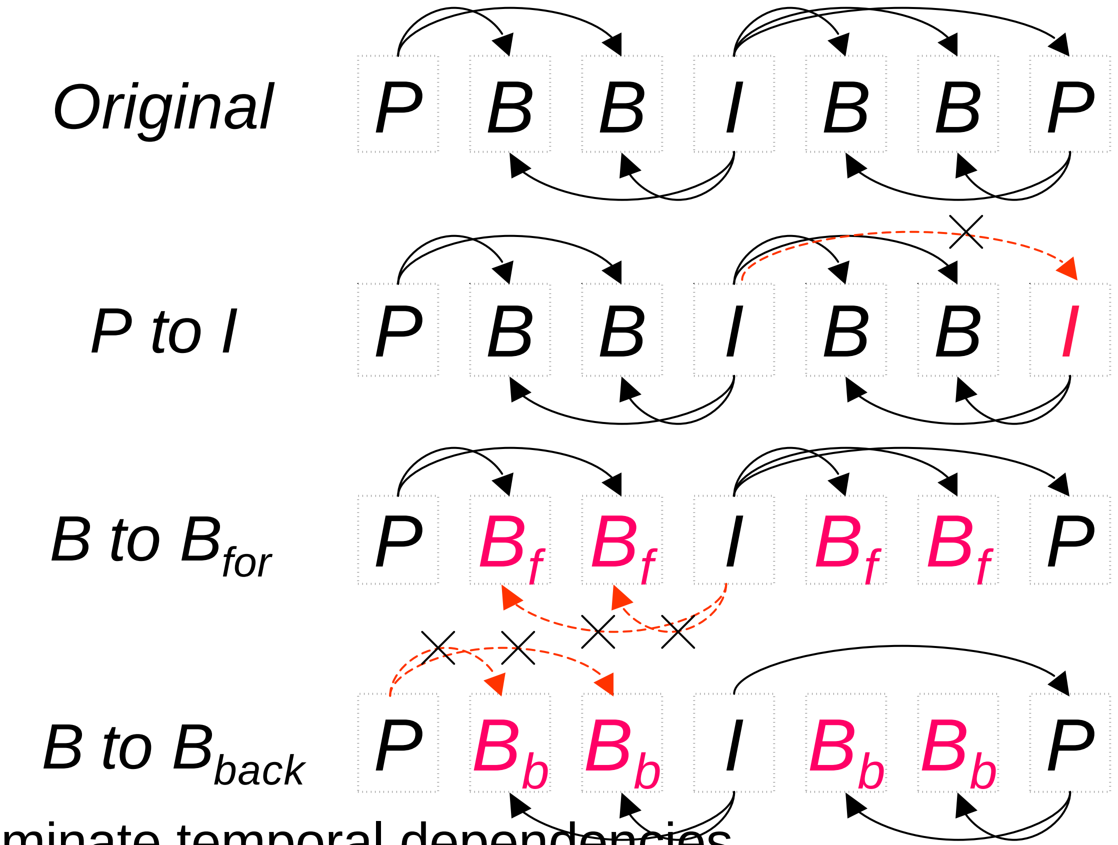
Step 2: Process the tail stream.

- Remove (forward) dependencies on dropped frames.

Step 3: Match & Merge the head and tail data.

- Perform rate matching.
- Reorder data appropriately.
- Update header information.

Frame Conversion: Remove Dependencies



Original P B B I B B P

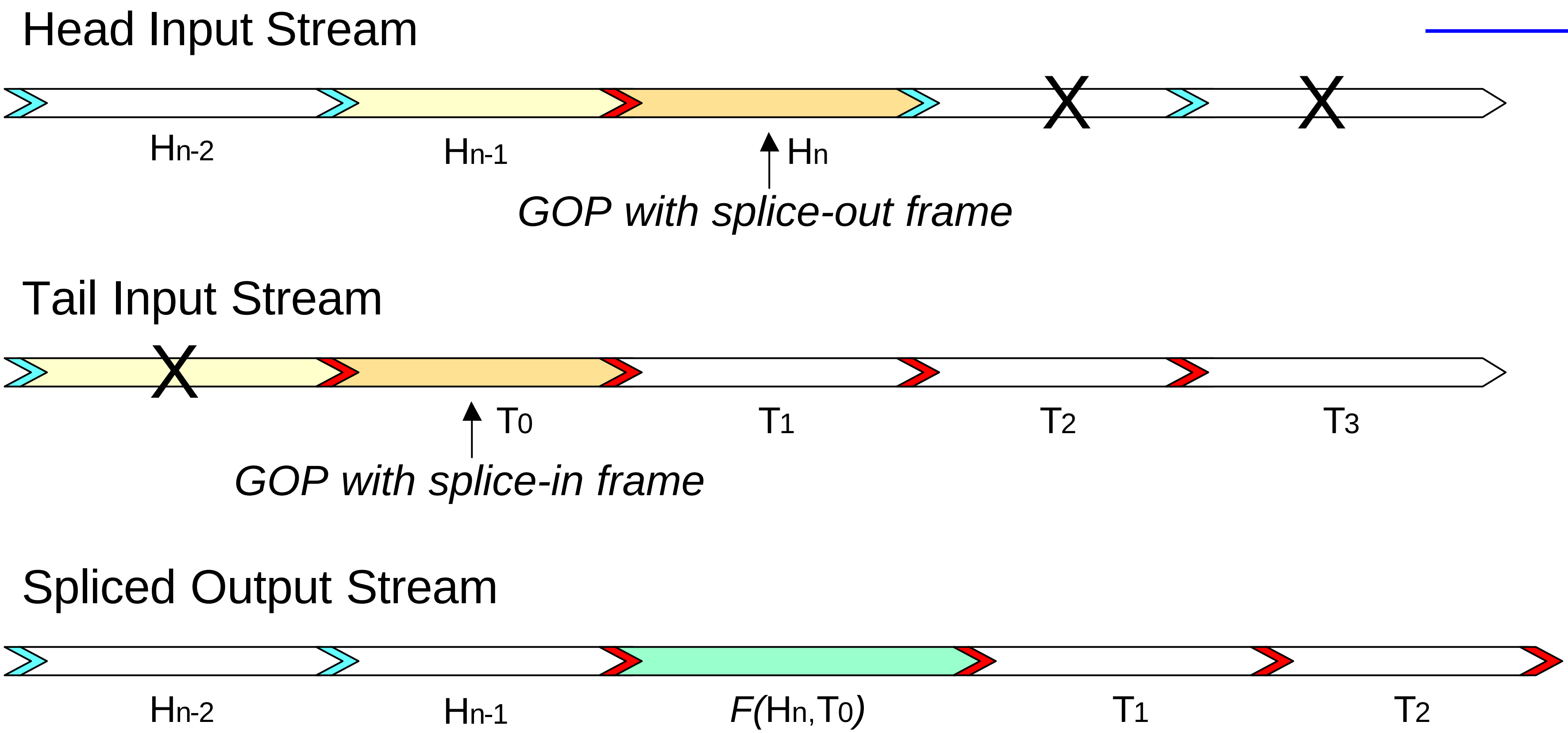
P to I P B B I B B I

B to B_{for} P B_f B_f I B_f B_f P

B to B_{back} P B_b B_b I B_b B_b P

1. Eliminate temporal dependencies.
2. Calculate new prediction (DCT-domain).
3. Calculate and code new residual.

Splicing



Head Input Stream

H_{n-2} H_{n-1} H_n

GOP with splice-out frame

Tail Input Stream

T_0 T_1 T_2 T_3

GOP with splice-in frame

Spliced Output Stream

H_{n-2} H_{n-1} $F(H_n, T_0)$ T_1 T_2

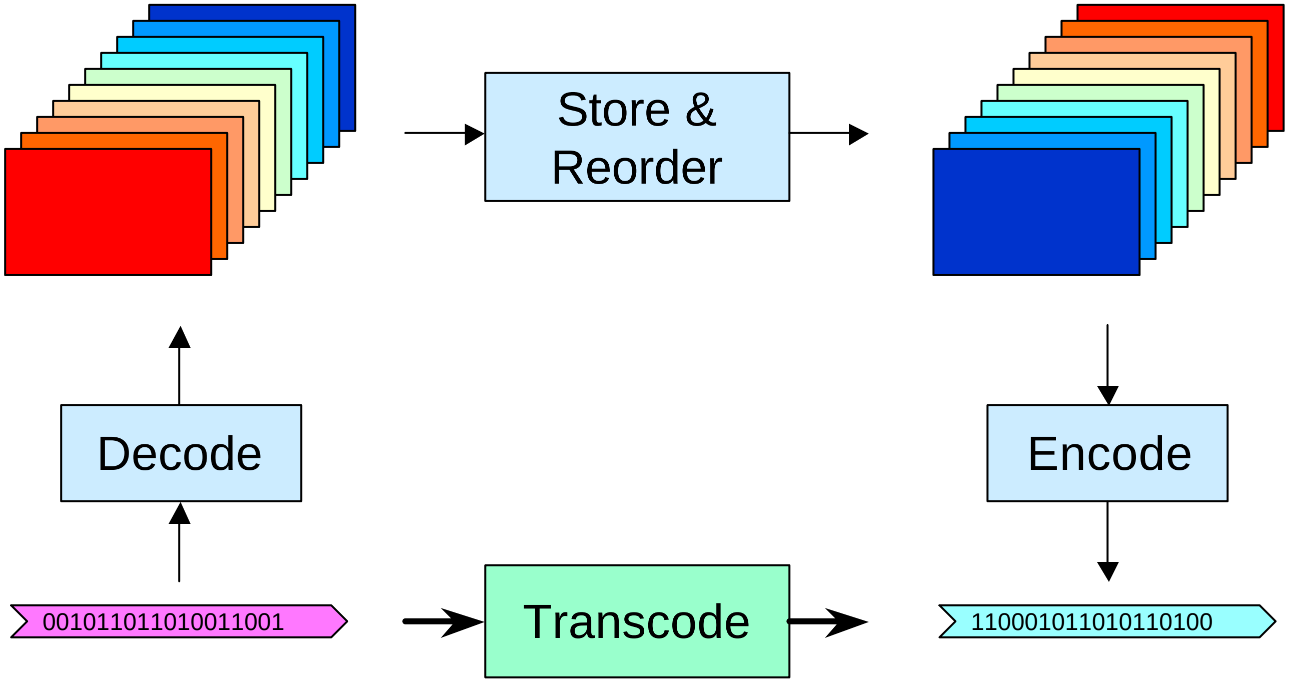
Splicing: Remarks

- Proposed Algorithm
  - Frame-accurate splice points
  - Only uses MPEG stream (Encoder is not affected.)
  - Frame conversions in MV+sparse DCT domain.
  - Rate control by requantization and frame conversion. (Do not insert synthetic frames.)
  - Quality only affected near splice points.
  - Computational scalability: Video quality can be improved if extra computing power is available.

Outline

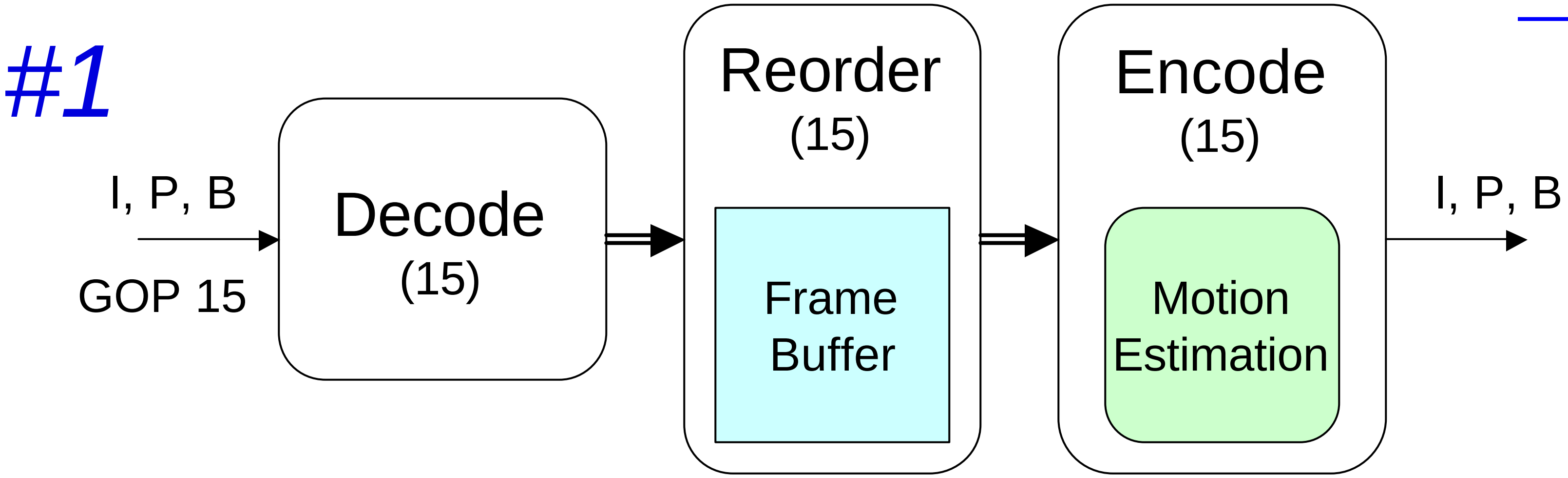
- Compressed-Domain Image Processing
- Compressed-Domain Video Processing
  - Review of MPEG basics
  - Manipulating temporal dependencies
  - Splicing
  - → Reverse Play
  - MPEG-2 to H.263 Transcoding
- Review and Demo

Reverse-Play Transcoding



Develop computation- and memory-efficient transcoding algorithms for *reverse play* of a given forward-play MPEG video stream.

Reverse-Play Architecture #1



- High memory requirements
  - Frame buffer must store 15 frames
- High computational requirements
  - Motion estimation dominates computational needs

Compressed-Domain Processing

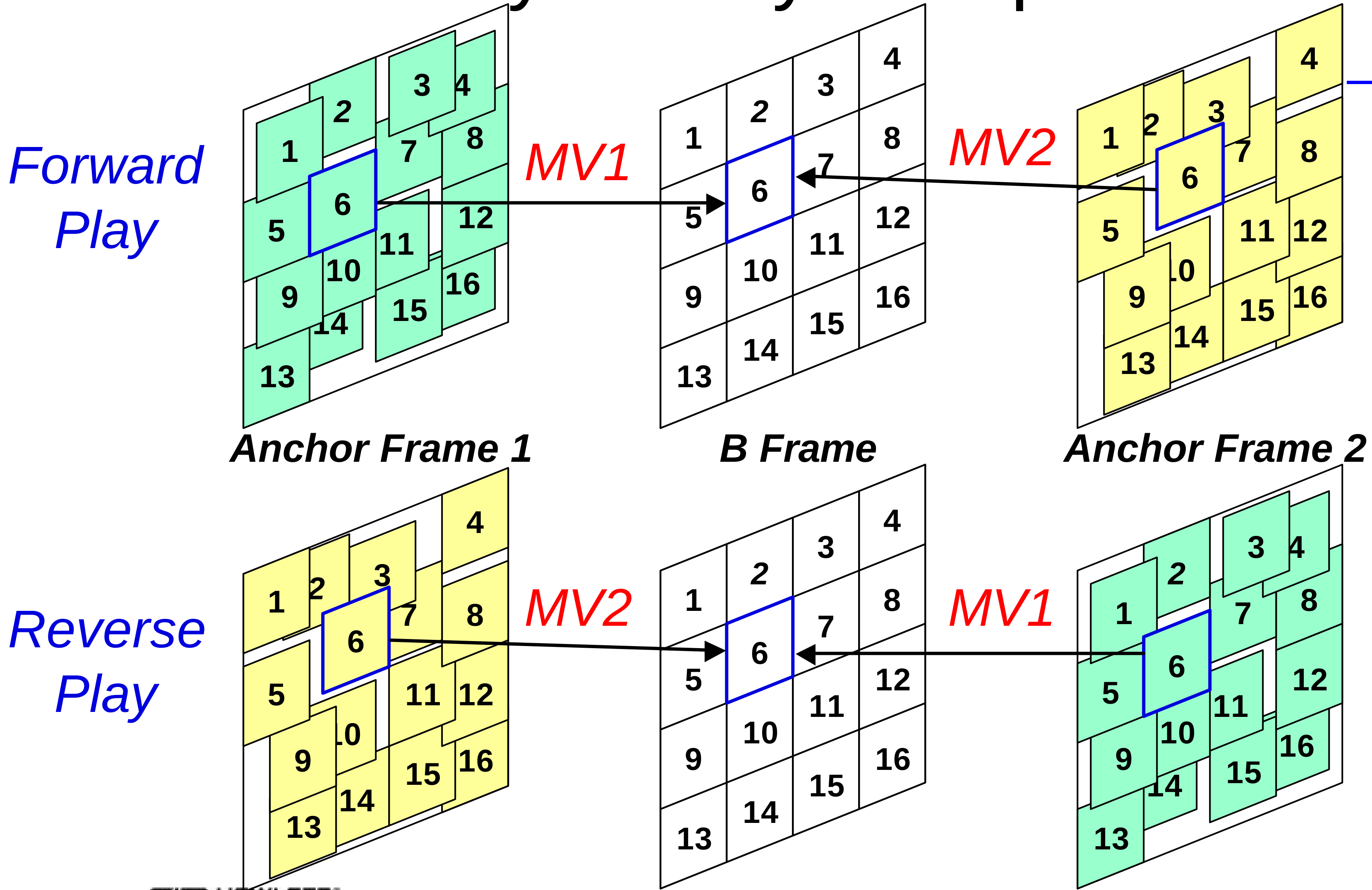
Can we do better by exploiting properties of

1) the MPEG compression algorithm

and

2) the reverse-play operation ?

B-Frame Symmetry: Swap MVs



Forward Play

Reverse Play

Anchor Frame 1

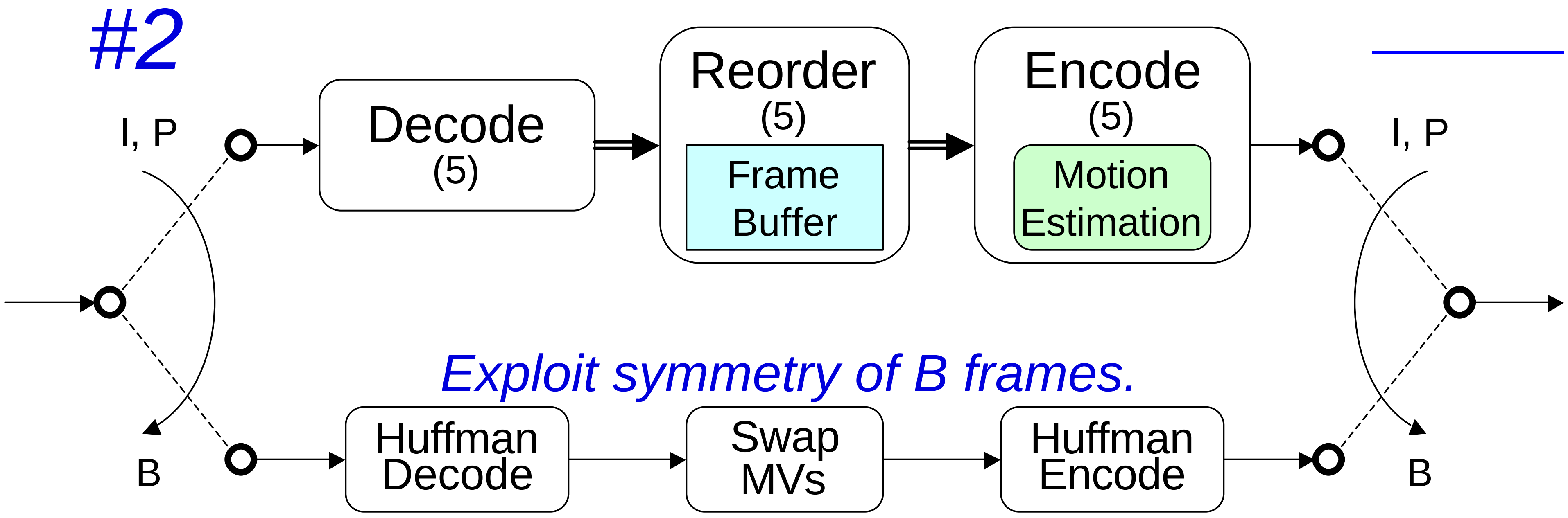
B Frame

Anchor Frame 2

MV1

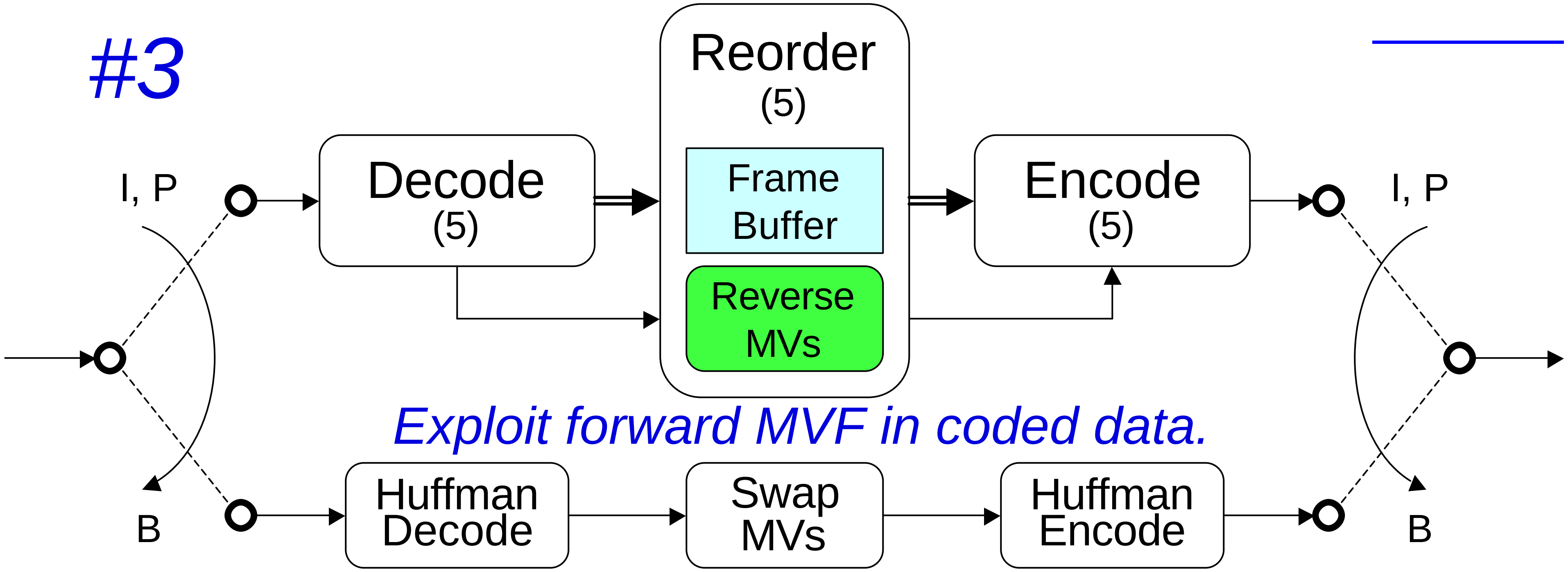
MV2

Reverse-Play Architecture #2



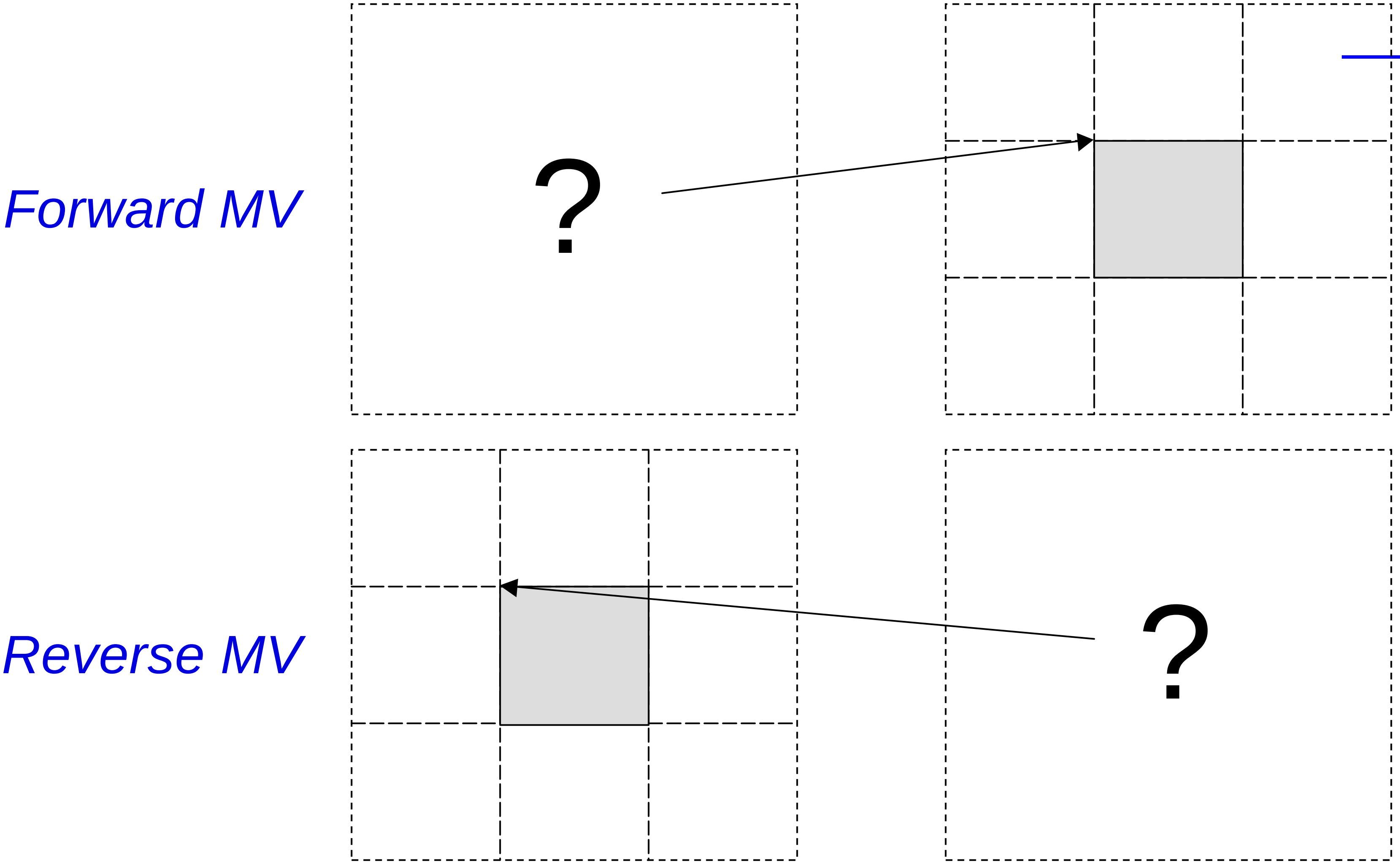
- Reduced memory requirements
  - Frame buffer must store 5 frames
- Reduced computational requirements
  - ME still dominates computational needs

Reverse-Play Architecture #3



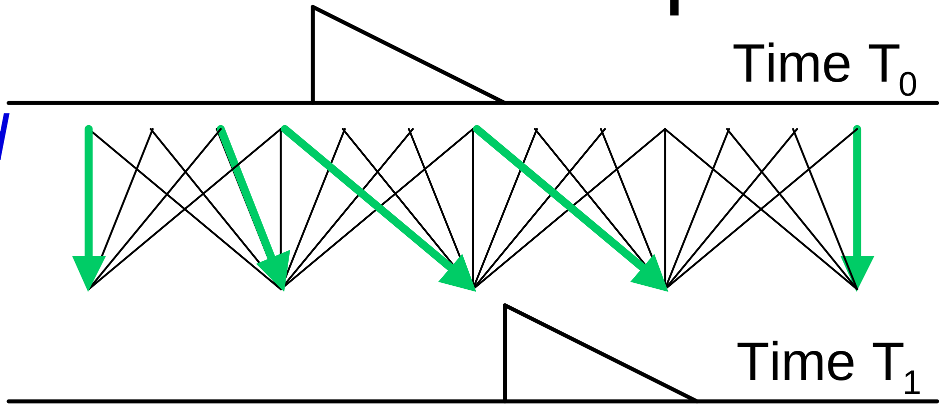
- Reduced memory requirements
- Reduced computational requirements

Forward versus Reverse MV's



Search Area: No correspondence

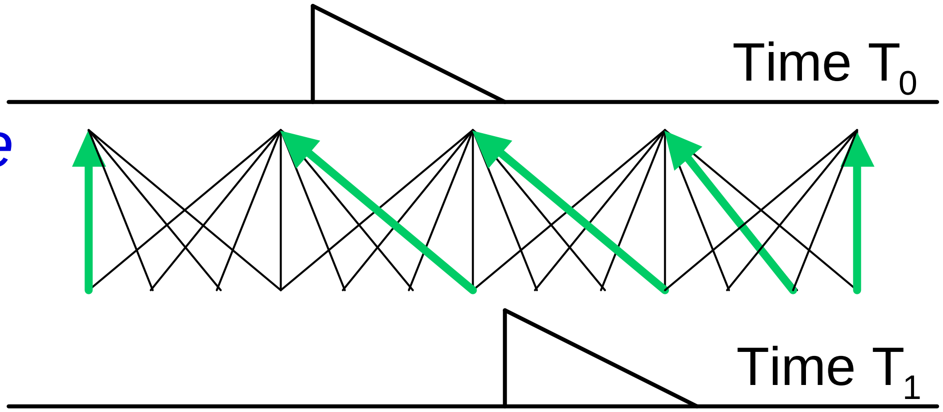
Forward search area



Time T_0

Time T_1

Reverse search area



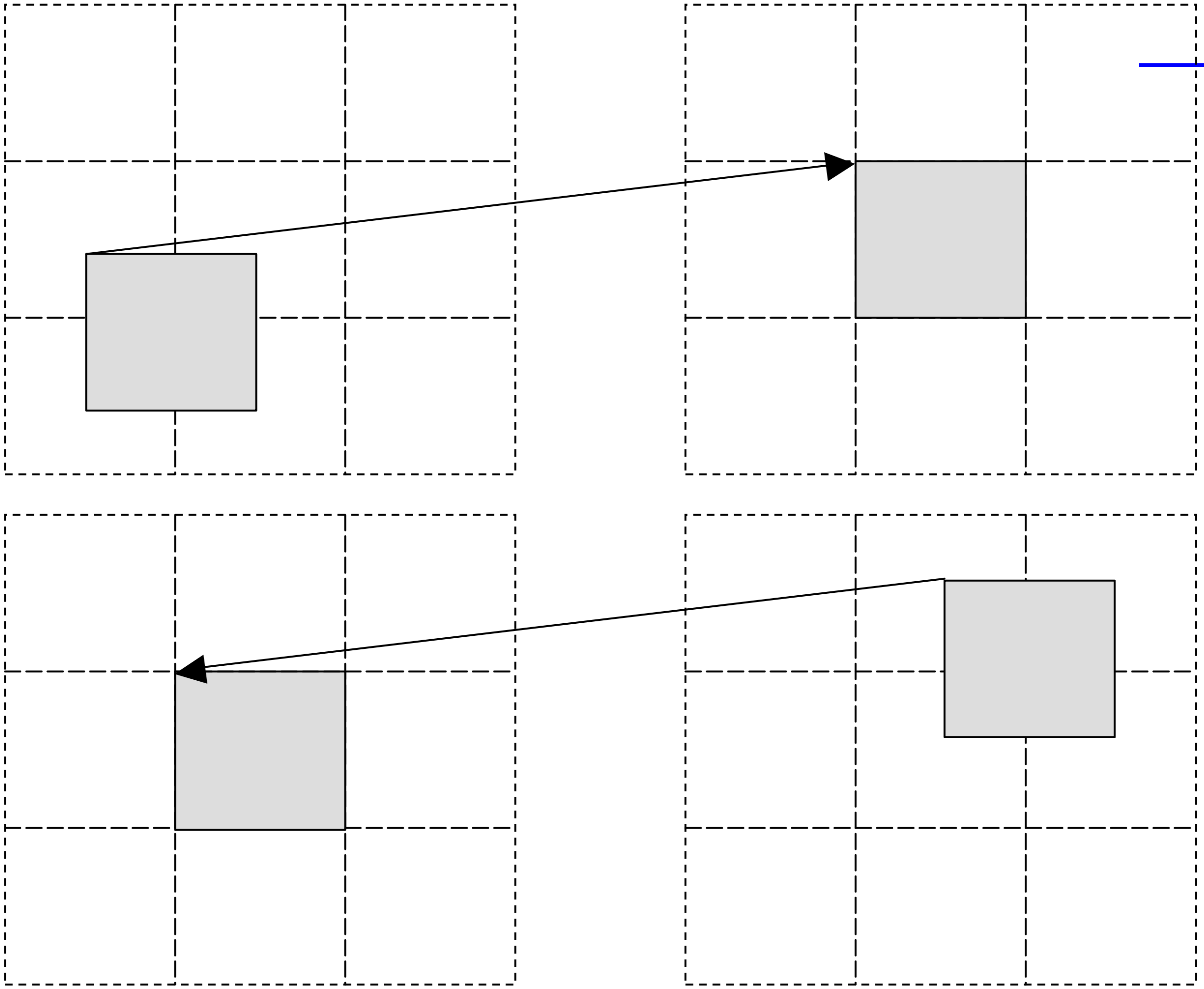
Time T_0

Time T_1

Interpret MVs as specifying a match between blocks.

Develop motion vector resampling methods.

In-place Reversal Method



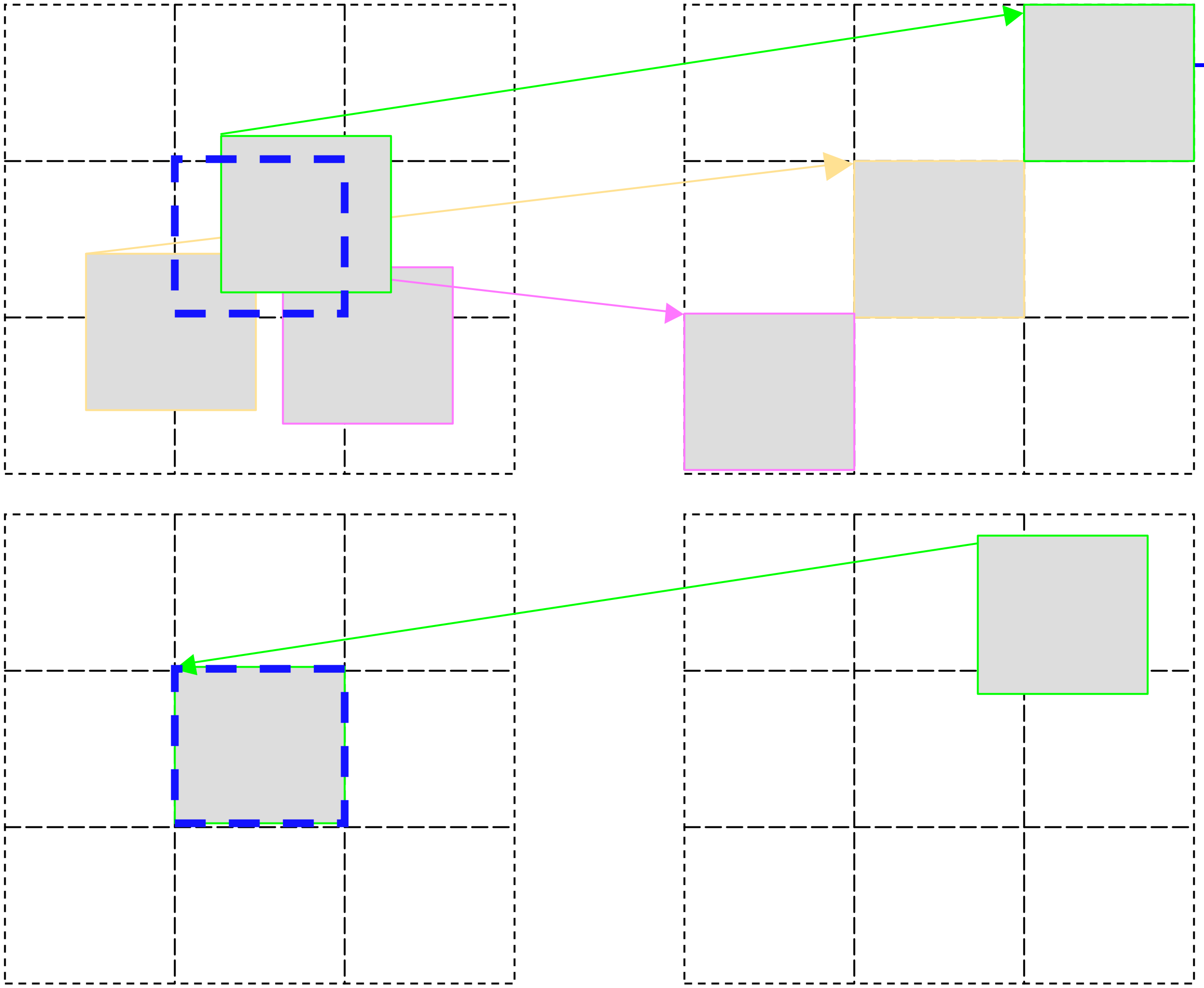
Forward MV

Reverse MV:
In-place
reversal

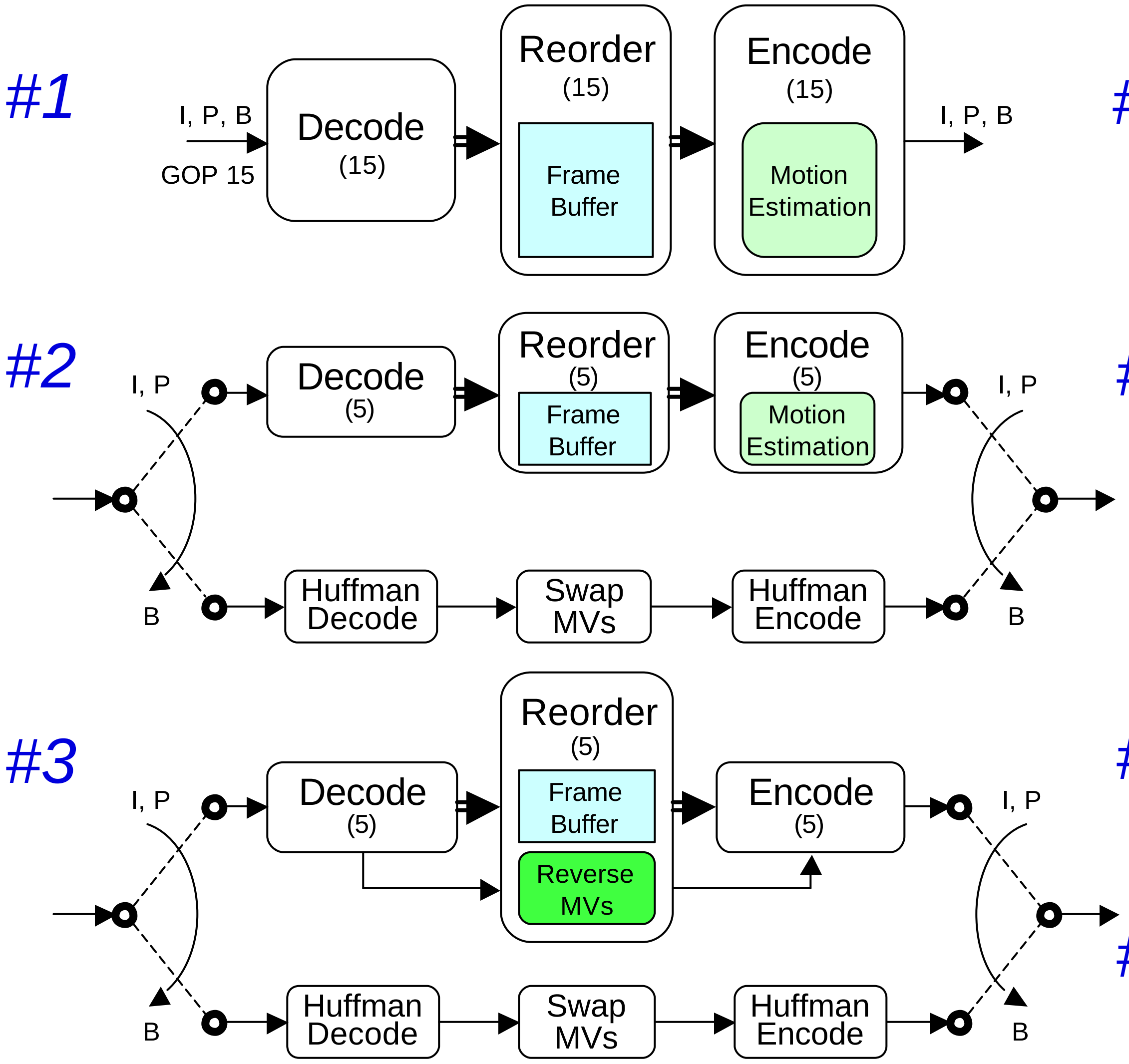
Maximum Overlap Method

Forward MV's overlapping block in question

Reverse MV: reversal of forward MV with max overlap



Computational Requirements



#1: 4200 Mcycles

Log Search ME

#2: 1030 Mcycles

Log Search ME

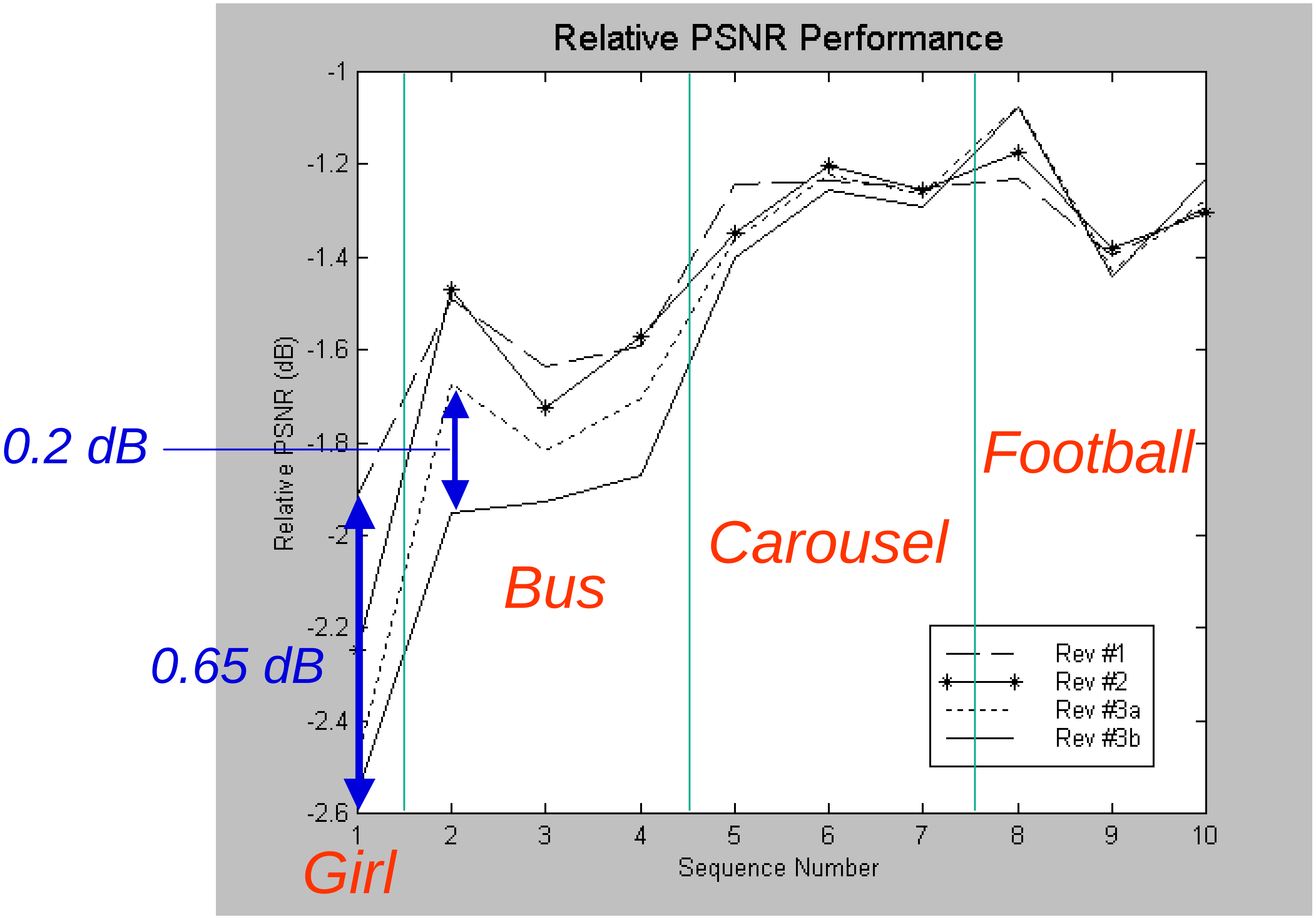
#3a: 420 Mcycles

Maximum Overlap

#3b: 330 Mcycles

In-place Reverse

Experimental Results



Observations

- Girl sequence showed largest PSNR loss (.65 dB) due to high detail and texture in source.
  - MV accuracy is important!
- Bus sequence benefits from maximum overlap because of large motions in source.
- Carousel has little performance loss because block MC does not match motion in source.
- Football has little performance loss due to blurred source.
  - When MV accuracy is not important, fast approximate methods do not sacrifice quality.

Reverse Play Summary

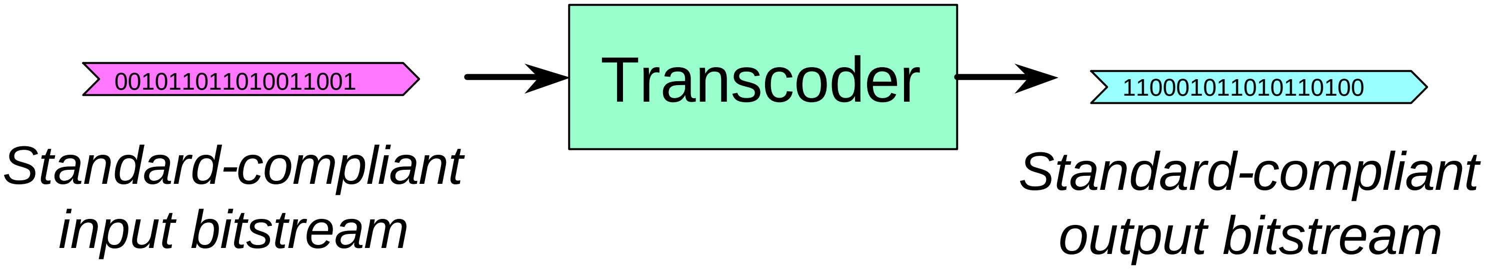
- Developed several compressed-domain reverse-play transcoding algorithms.
- CDP: Exploit properties of compression algorithm and reverse-play operation
  - Exploit symmetry of B frames.
  - Exploit MVF information given in forward stream.
- Order of magnitude reduction in computational requirements.
- Worst case 0.65 dB loss in PSNR quality over baseline transcoding.

Outline

- Compressed-Domain Image Processing
- Compressed-Domain Video Processing
  - Review of MPEG basics
  - Manipulating temporal dependencies
  - Splicing
  - Reverse Play
  - → MPEG-2 to H.263 Transcoding
- Review and Demo

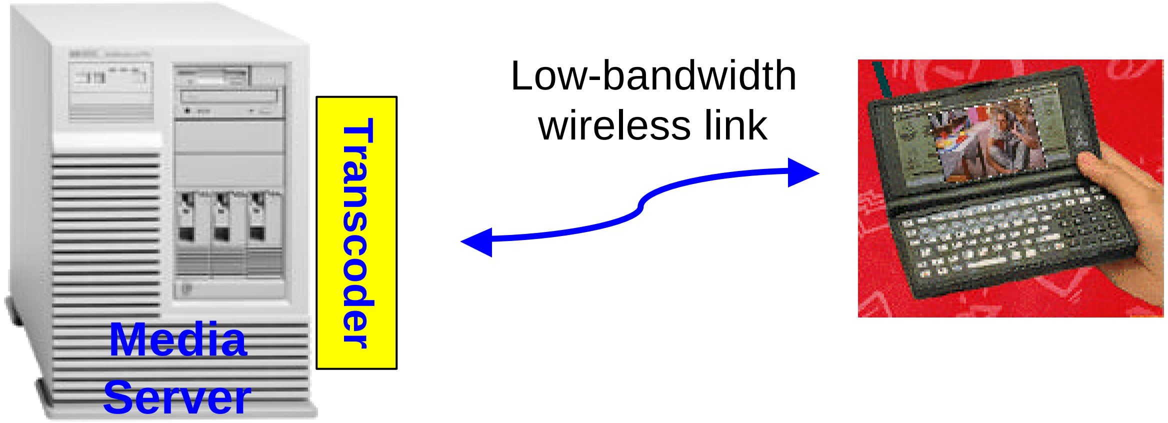
Motivation

- Video communication requires the seamless delivery of video content to users with a broad range of bandwidth and resource constraints.
- However, the source signal, communication channel, and client device may be incompatible.
- Therefore, efficient transcoding algorithms must be designed


0010110110100110001
Transcoder
1100010110101101000
Standard-compliant input bitstream
Standard-compliant output bitstream

MPEG2-to-H.263 Transcoding

Media Server
DTV or DVD content



MPEG-to-H.263 Transcoder

- Transcode MPEG video streams to lower-rate H.263 video streams.
- Interlace-to-progressive conversion.
- Order of magnitude reduction in computational requirements.

Stream DVD movies to portable multimedia devices.

Stream DTV program material over the internet.

Problem Statement

Develop a fast transcoding algorithm that adapts the bitrate, frame rate, spatial resolution, scanning format, and/or coding standard while preserving video quality

MPEG-2 input

- High bitrates
- DVD, Digital TV
- >1.5 Mbps, 30 fps
- Interlaced and progressive

H.263 output

- Low bitrates
- Wireless, internet
- <500 kbps, 10fps
- Progressive

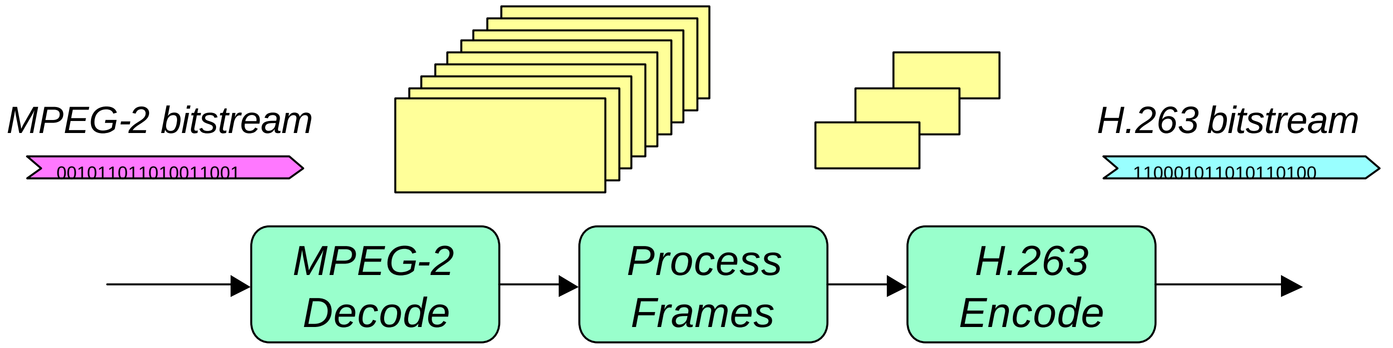
Important features:

- *Interlaced-to-progressive* transcoder.
- *Inter-standard* transcoder, e.g. MPEG-2 to H.263 (or MPEG-4)

Contributors: Wee, Apostolopoulos, Feamster (MIT)

Difficulties

- High cost of conventional transcoding



MPEG-2 bitstream

0010110110100110001

H.263 bitstream

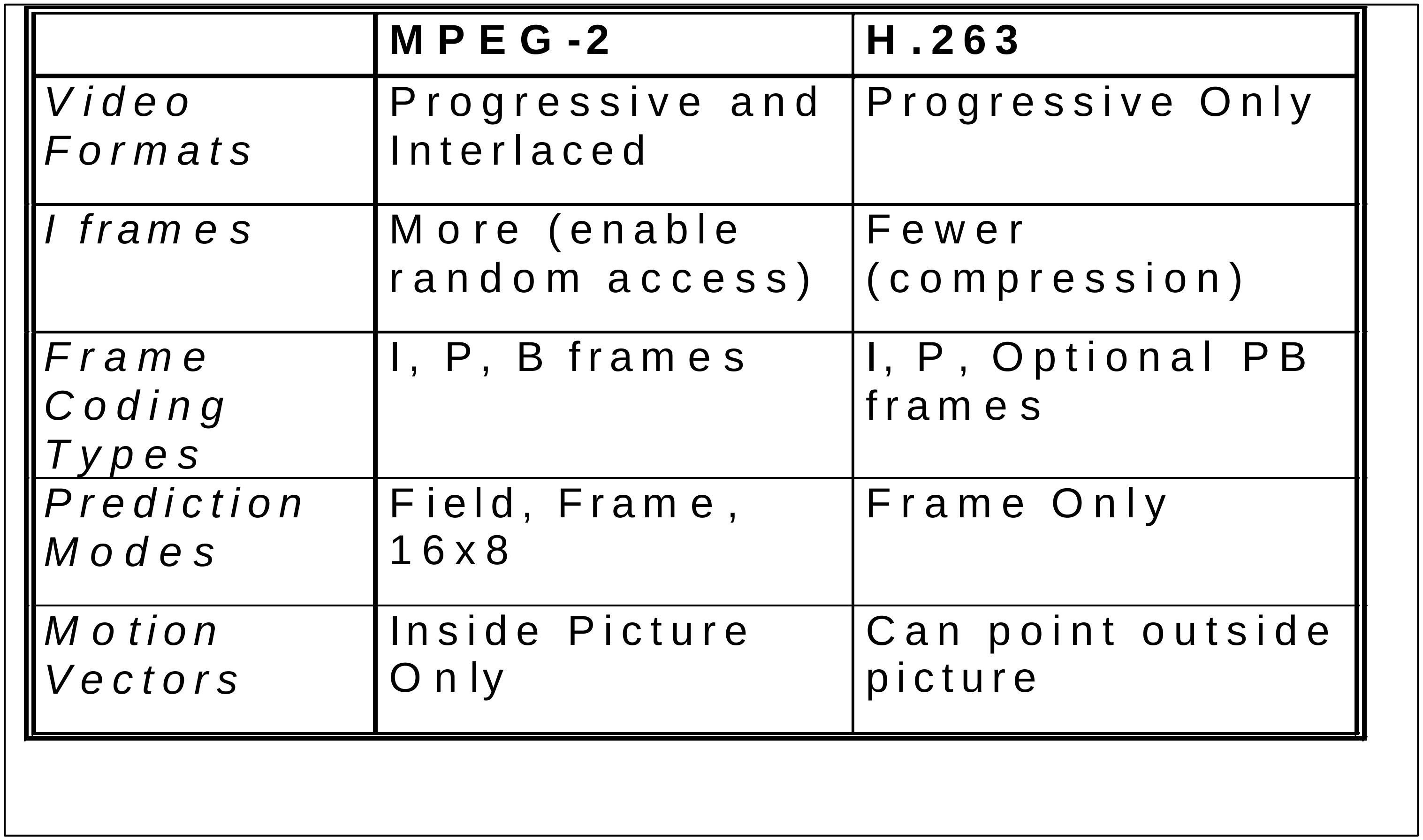
1100010110101101100

MPEG-2 Decode → *Process Frames* → *H.263 Encode*

- Differences in video compression standards
- Interlaced vs. progressive formats

HEWLETT® PACKARD

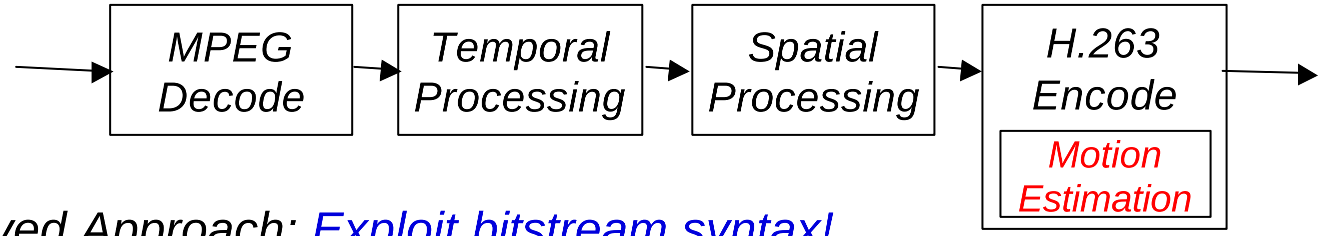
Issue: Standard Differences



	MPEG-2	**H.263**
Video Formats	Progressive and Interlaced	Progressive Only
I frames	More (enable random access)	Fewer (compression)
Frame Coding Types	I, P, B frames	I, P, Optional PB frames
Prediction Modes	Field, Frame, 16x8	Frame Only
Motion Vectors	Inside Picture Only	Can point outside picture

Development of Proposed Approach

Conventional



MPEG Decode

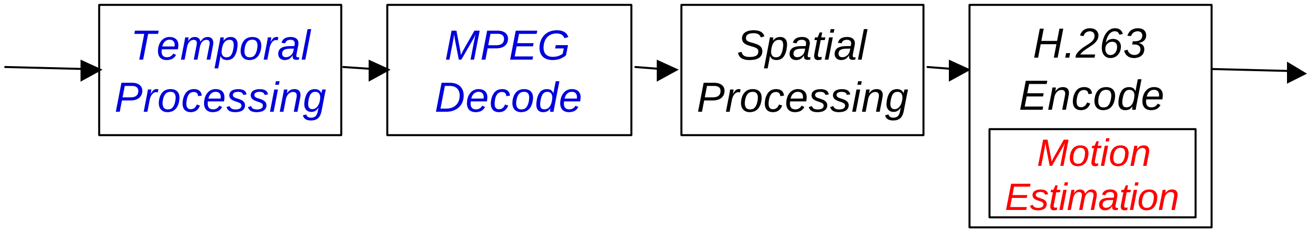
Temporal Processing

Spatial Processing

H.263 Encode

Motion Estimation

Improved Approach: Exploit bitstream syntax!



Temporal Processing

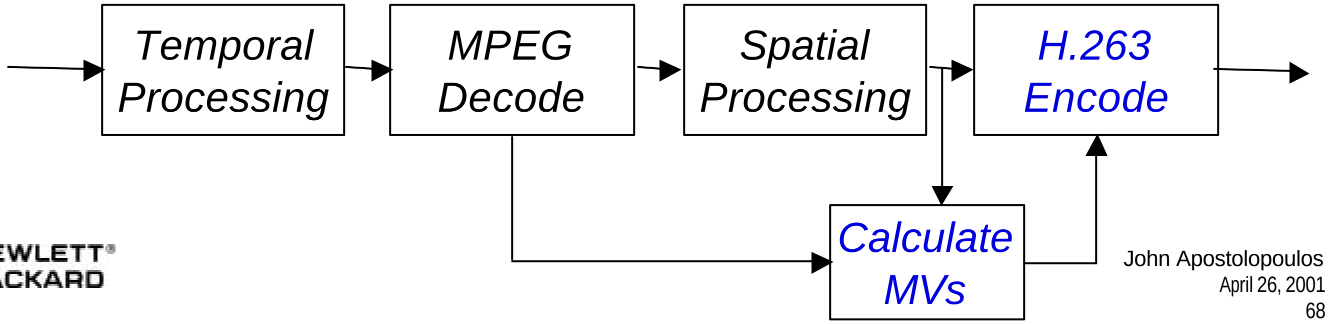
MPEG Decode

Spatial Processing

H.263 Encode

Motion Estimation

Proposed Aproach: Also exploit input coded data!



Temporal Processing

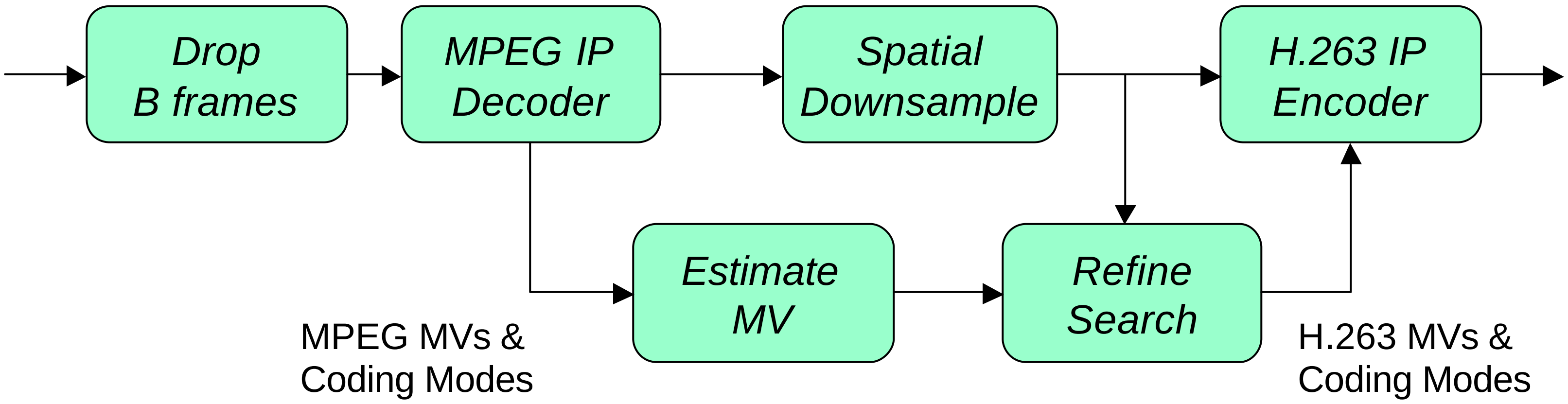
MPEG Decode

Spatial Processing

H.263 Encode

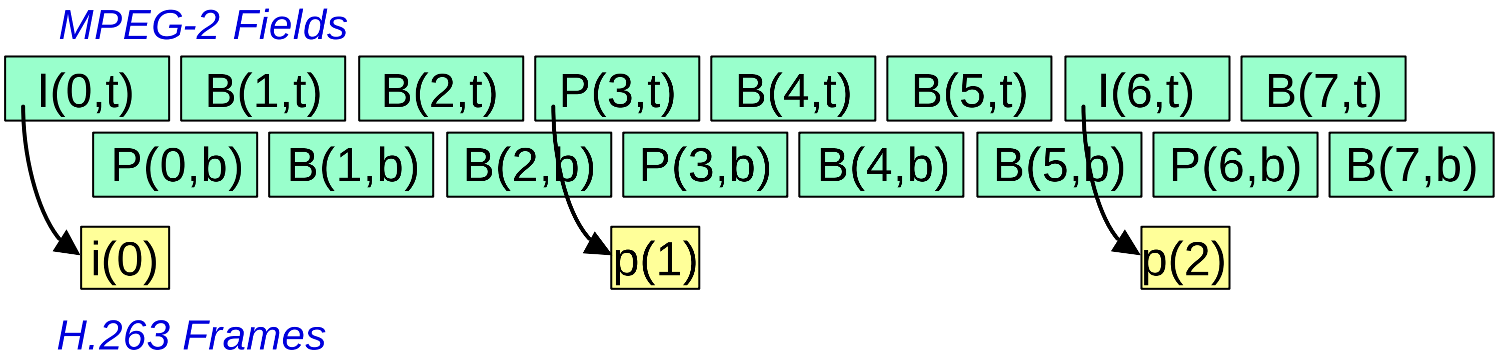
Calculate MVs

Block Diagram



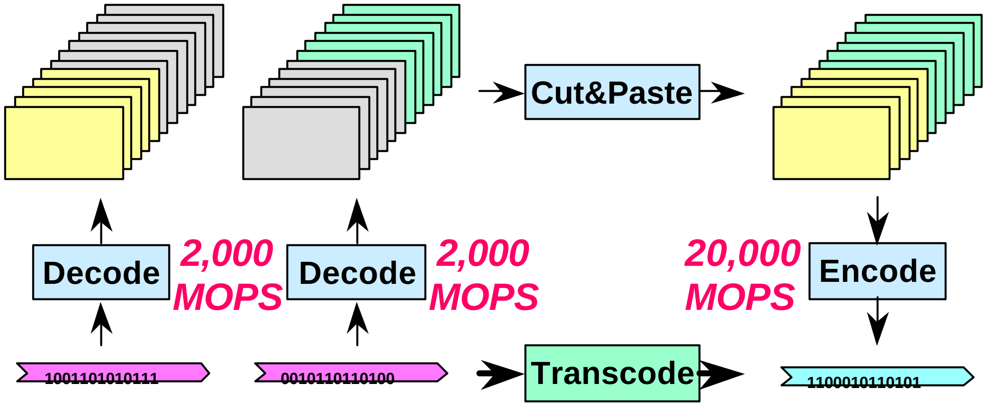
Issue: Match Prediction Modes

- Match MPEG-2 Fields to H.263 Frames



- Vertical downsampling => discard bottom field
- Horizontal downsampling => downsample top field
- Temporal downsampling => drop B frames
- Match MPEG-2 IPPPPIPPPP to H.263 IPPPPPPPPP
  - Problems
    - Convert MPEG-2 P fields to H.263 P frames
    - Convert MPEG-2 I fields to H.263 P frames

Compressed-Domain Splicing



< 2,000 MOPS
for one splice point every sec

Enables Ad Insertion for DTV

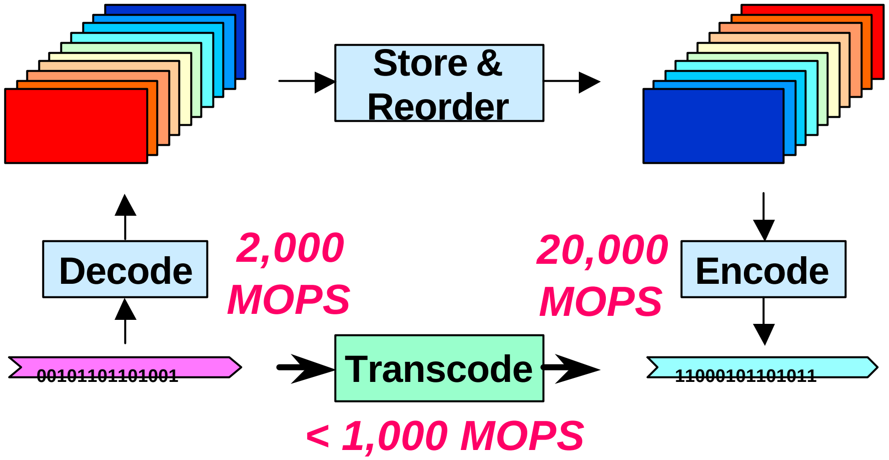
SMPTE splicing solution

- User must define splice points during encoding
  Specialized complex encoders.
- Restricted splice points

Splicing solution

- Works with any MPEG stream.
- Frame-accurate splice points

Reverse-Play Transcoding



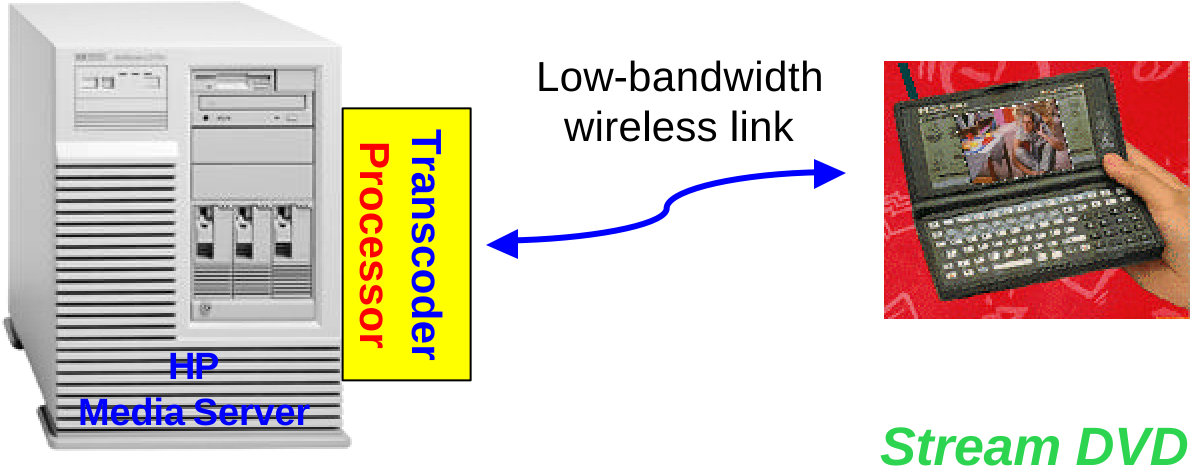
VCR functionality for DTV and Set Tops

Reverse-play solution

- Frame-by-frame reverse play.
- Works with any MPEG stream.
- Order of magnitude reduction in computational requirements.

HEWLETT PACKARD

MPEG2-to-H.263 Transcoding



MPEG-to-H.263 Transcoder

- Transcode MPEG video streams to lower-rate H.263 video streams.
- Interlace-to-progressive conversion.
- Order of magnitude reduction in computational requirements.

Stream DVD movies to portable multimedia devices.

Stream DTV program material over the internet.

Demo

References

Handouts:

- “Manipulating Temporal Dependencies in Compressed Video Data with Applications to Compressed-Domain Processing of MPEG Video”, S. Wee, ICASSP 1999.
- “Compressed-Domain Reverse Play of MPEG Video Streams”, S. Wee, B. Vasudev, SPIE Inter. Sym. On Voice, Video, and Data Communications 1998.
- “Field-to-frame Transcoding with Spatial and Temporal Downsampling”, S. Wee, J. Apostolopoulos, N. Feamster, ICIP 1999.

And references in above papers.