



# Обзор современных методов видеокодирования

Василий Шампоров  
Июль 2019

Internet of Things Group

# Видеокодек



Промежуточный формат

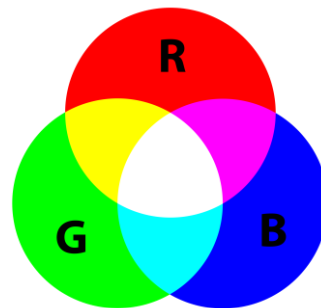
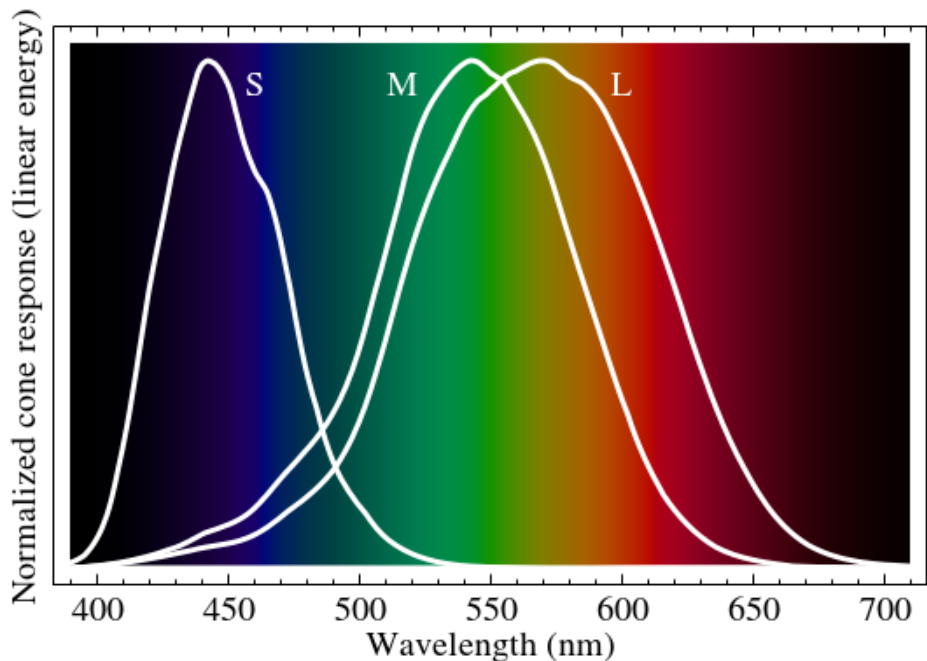


КОдирование

ДЕКОдирование

Кодек - преобразователь данных!

# Компьютерное представление цвета - RGB



На каждую цветовую компоненту - 1 байт

Красный (**R**):  
0-255 (**0x00** – **0xFF**)

Зеленый (**G**):  
0-255 (**0x00** – **0xFF**)

Синий (**B**):  
0-255 (**0x00** – **0xFF**)

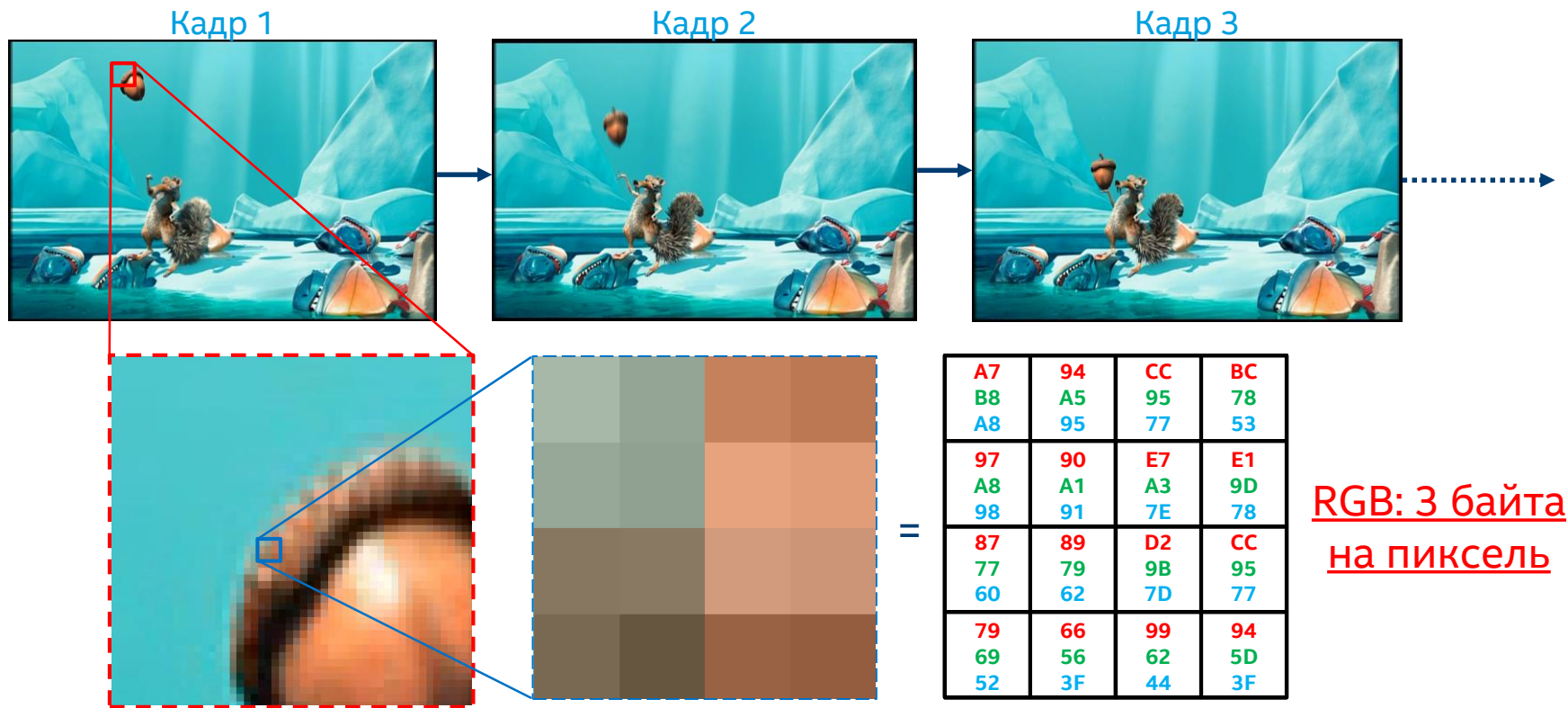


→ 215, 129, 43 → **0xD7812B**



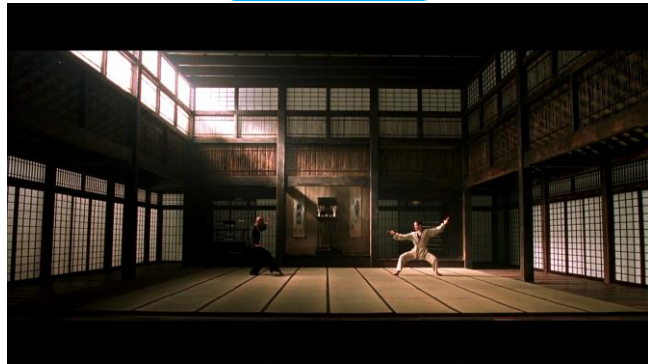
→ 0, 112, 192 → **0x0070C0**

# Компьютерное представление видео



# Проблема в цифрах

Целый кадр



Типичная продолжительность фильма: **2 ч**

Число кадров в секунду: **30**

Разрешение: **1920x1080**

Цветовая схема: **RGB**

Оригинальное RGB-видео



**1.22 ТБ**

$$2 \times 3600 \times 30 \times 1920 \times 1080 \times 3 \\ = \\ 1343692800000 \text{ байт}$$

Кадр, переданный с помощью кодека



**~2500 МБ**

Сжатие в **~500 раз!**

High Efficiency Video Codec (HEVC)

# Как сжать?



512x512

Оригинал



128x128

Сжатие: 16x



64x64

Сжатие: 64x

- Снижение разрешения всех RGB-компонент – неэффективное решение...



# Компьютерное представление цвета - YUV

**Y** – светимость (0 – 255)

} **Luma**

**U** – цветность синего (0 – 255)

**V** – цветность красного (0 – 255)

} **Chroma**

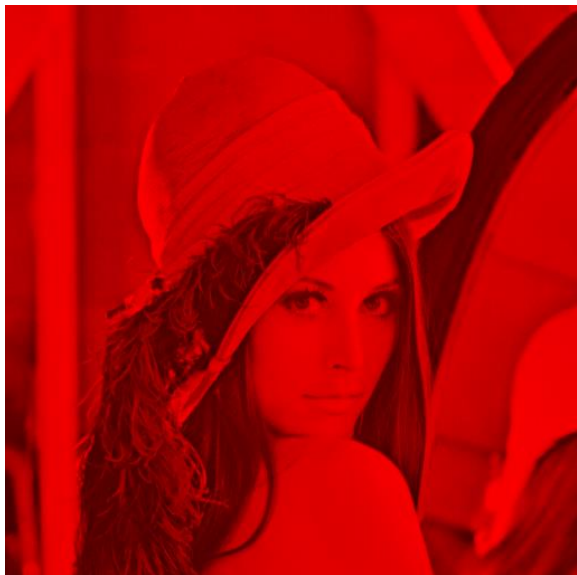
Семейство форматов YUV:

Y'UV, YUV, YCbCr (Rec. 601, Rec. 609, Rec. 2020, Rec. 2100), YPbPr, ...

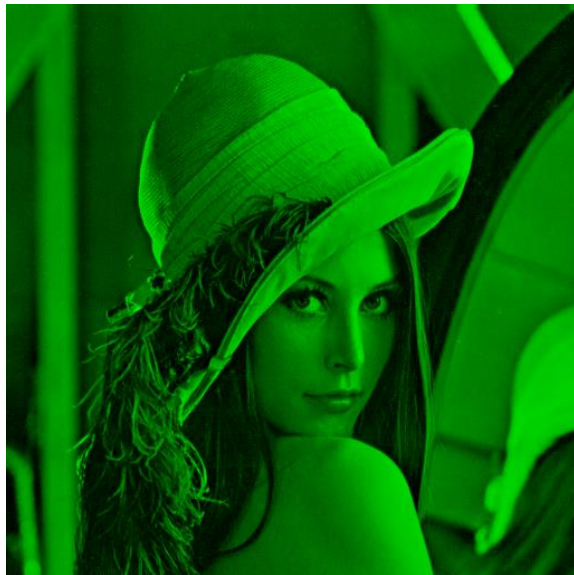
$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- Цвет в YUV однозначно соответствует цвету в RGB, и наоборот

# RGB



512x512  
R – красный  
1 байт на пиксель



512x512  
G – зеленый  
1 байт на пиксель



512x512  
B – синий  
1 байт на пиксель

Всего –  $1+1+1=3$  байта на пиксель



# YUV



512x512

Y – светимость  
1 байт на пиксель



512x512

U – цветность (синий)  
1 байт на пиксель

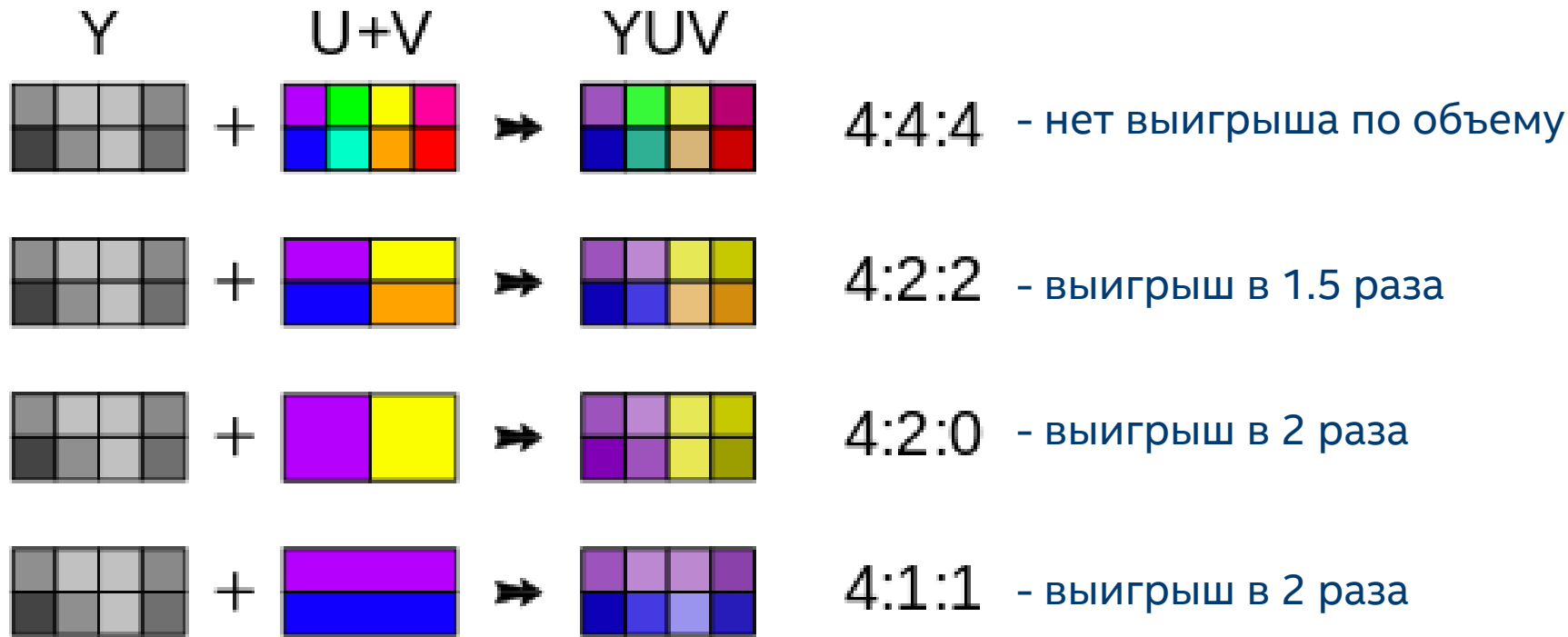


512x512

V – цветность (красный)  
1 байт на пиксель

Всего –  $1+1+1=3$  байта на пиксель

# YUV – прореживание цветности (downsampling)



- Глаз более чувствителен к изменениям Y, нежели к изменениям U и V

# YUV 4:2:0



512x512

Y – светимость  
1 байт на пиксель



256x256

U – цветность (синий)  
1/4 байт на пиксель



256x256

V – цветность (красный)  
1/4 байт на пиксель

Всего –  $1 + 1/4 + 1/4 = 1.5$  байта на пиксель



RGB 512x512  
**768 KB**



YUV 4:2:0 512x512  
**384 KB**





RGB 512x512  
**768 KB**



RGB 368x368  
**384 KB**

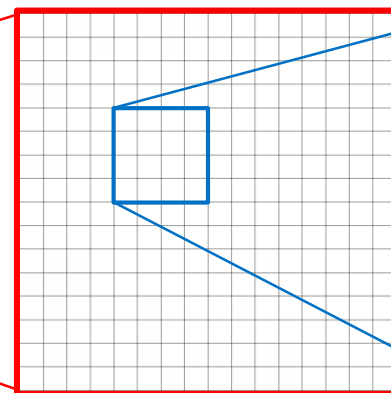
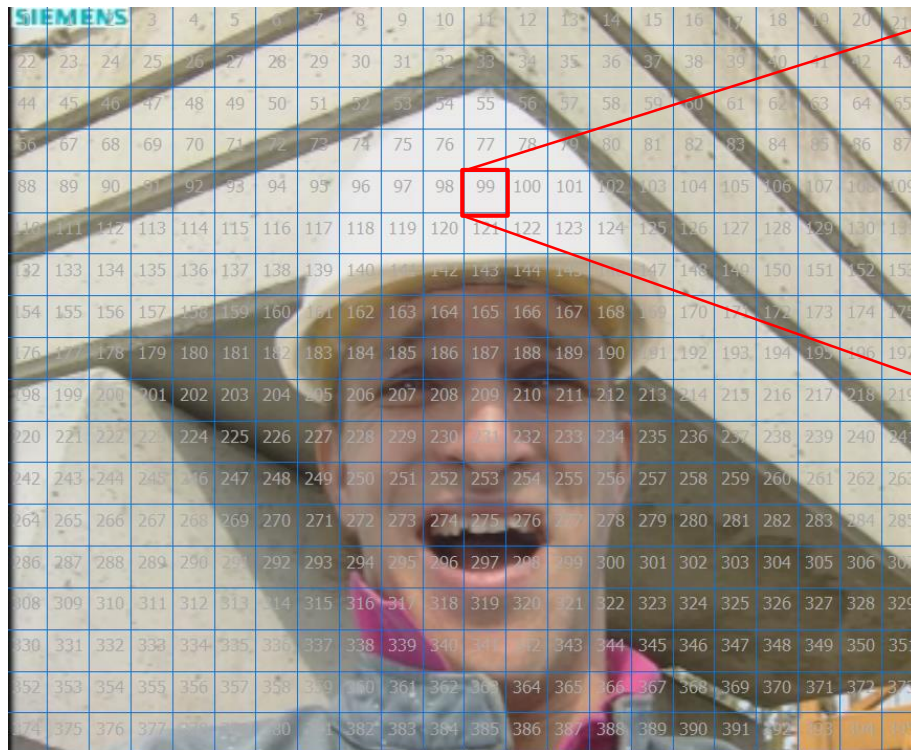
# Как сжать?



- Тестовый видеопоток – `foreman_352x288_300.yuv`
- Однородные и неоднородные элементы внутри кадров
- Статичные и движущиеся части картинки
- Резкая смена плана
- Дрожание камеры



# Пространственная избыточность



Блок 16x16

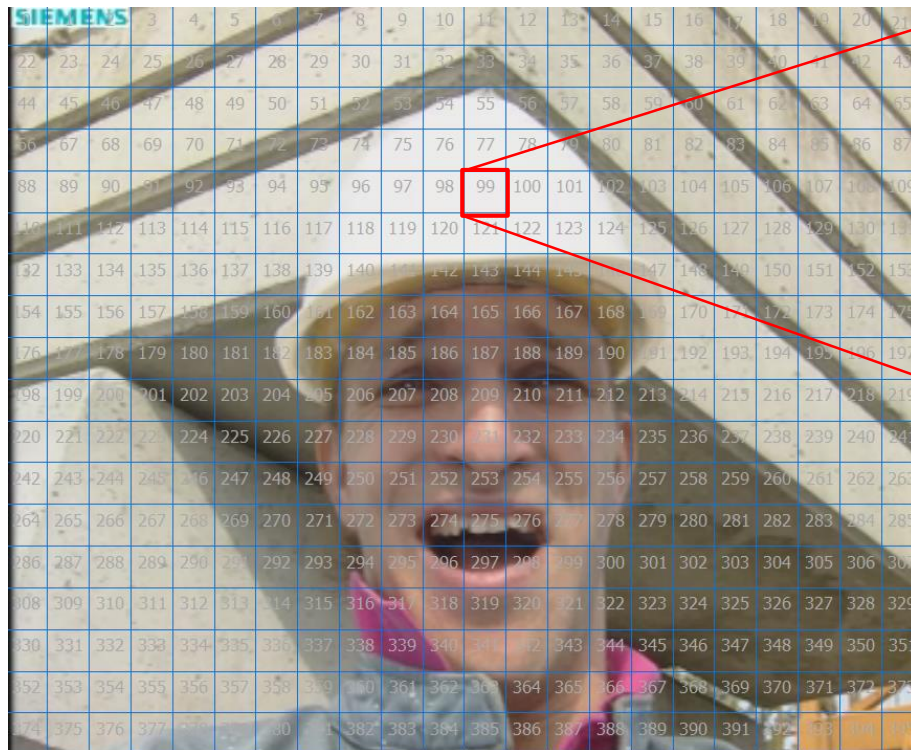
236	236	236	236
236	236	236	236
236	236	236	236
236	236	236	236

Блок 4x4 (luma)

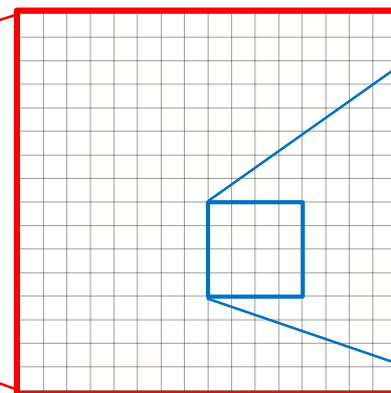
## Однородный блок:

- 16 одинаковых байт в блоке – избыточность!
- Вместо 16 байт можно обойтись 2 байтами – среднее значение “236” + некий символ «однородный блок»

# Пространственная избыточность



Кадр 24/300



Блок 16x16

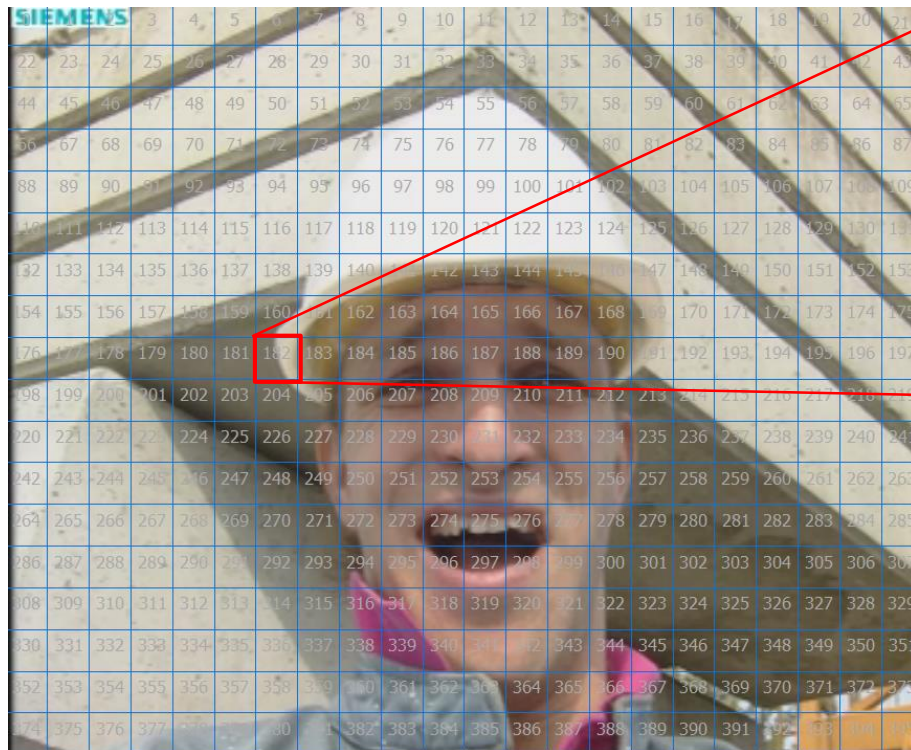
237	236	236	236
236	236	236	232
236	236	234	236
236	238	236	236

Блок 4x4 (luma)

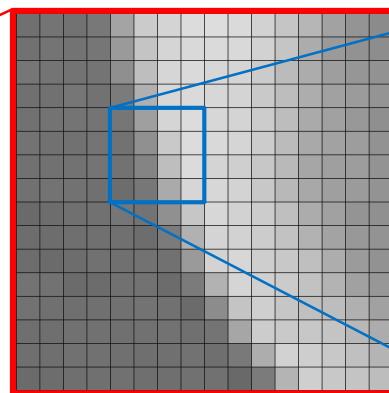
Практически однородный блок:

- Передача только среднего значения – максимальное сжатие с незначительной потерей качества

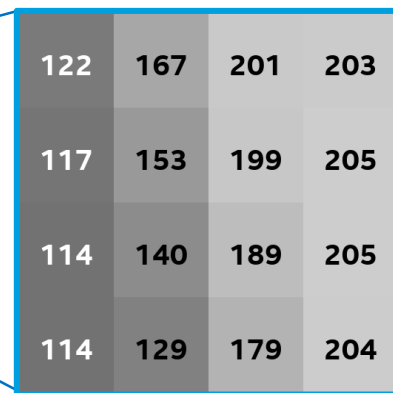
# Пространственная избыточность



Кадр 24/300



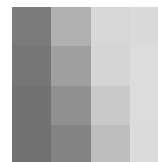
Блок 16x16



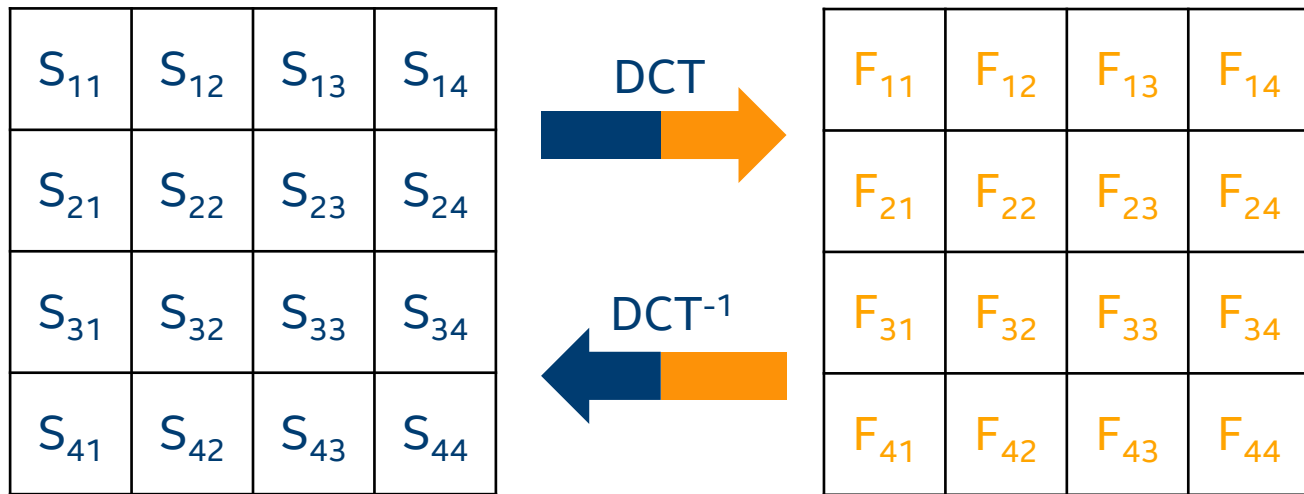
Блок 4x4 (luma)

## Неоднородный блок:

- Передача усредненного блока - неоправданная потеря качества



# Дискретное косинус-преобразование (DCT)

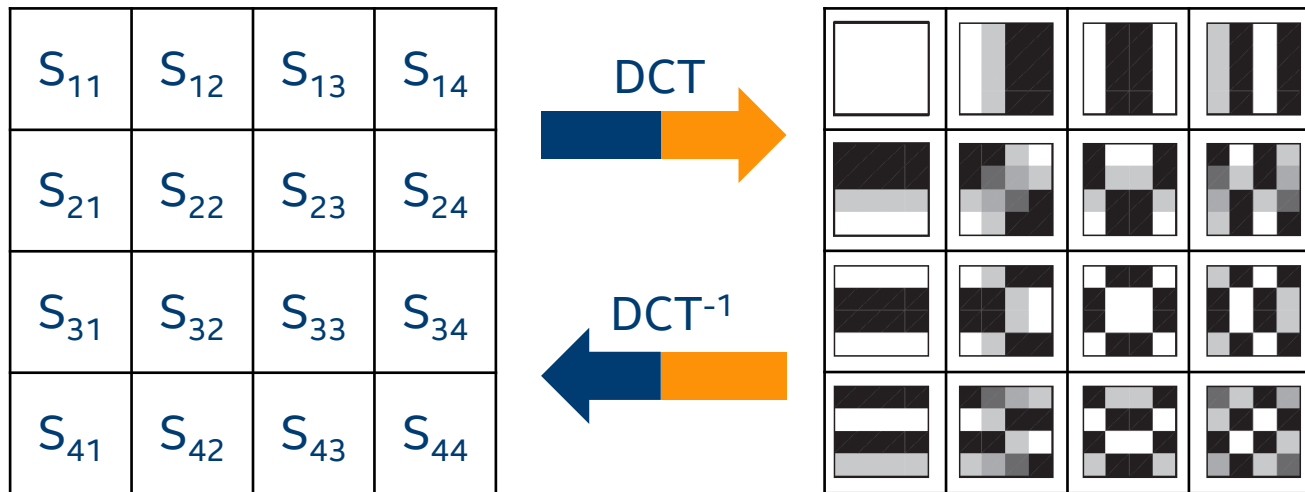


$$F_{vu} = \frac{2}{N} C_v C_u \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} S_{yx} \cos\left(v\pi \frac{2y+1}{2N}\right) \cos\left(u\pi \frac{2x+1}{2N}\right)$$

$$S_{yx} = \frac{2}{N} \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} C_v C_u F_{vu} \cos\left(v\pi \frac{2y+1}{2N}\right) \cos\left(u\pi \frac{2x+1}{2N}\right)$$

$$C_{u,v} = \begin{cases} \frac{1}{\sqrt{N}}, & (u, v = 0) \\ \frac{\sqrt{2}}{\sqrt{N}}, & (u, v \neq 0) \end{cases}$$

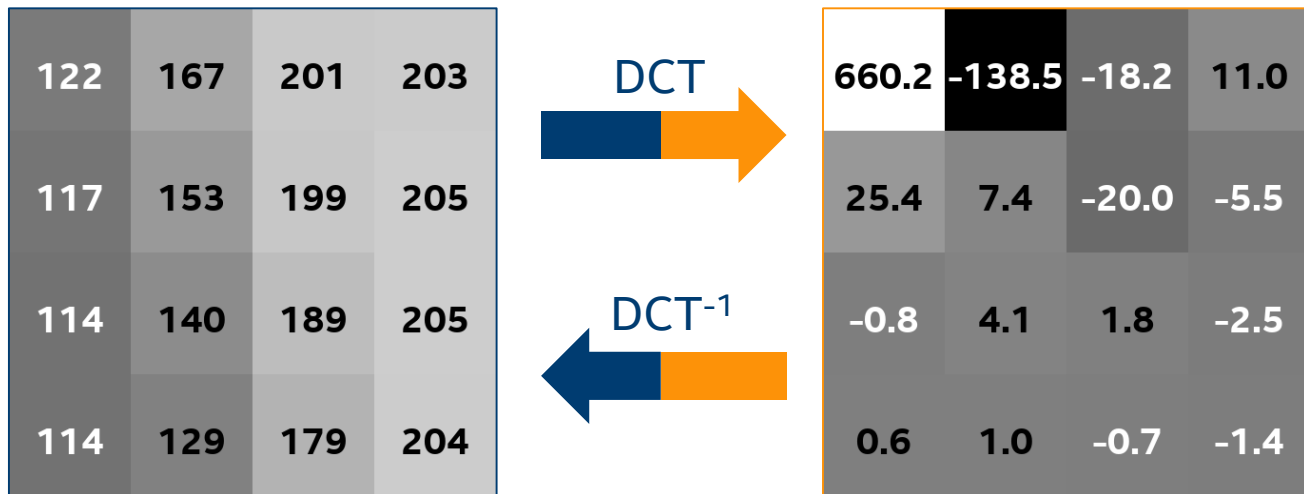
# Дискретное косинус-преобразование (DCT)



$$S_{yx} = \frac{2}{N} \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} C_v C_u F_{vu} \cos\left(v\pi \frac{2y+1}{2N}\right) \cos\left(u\pi \frac{2x+1}{2N}\right)$$

- Коэффициенты DCT соответствуют вкладу каждой моды в исходный блок

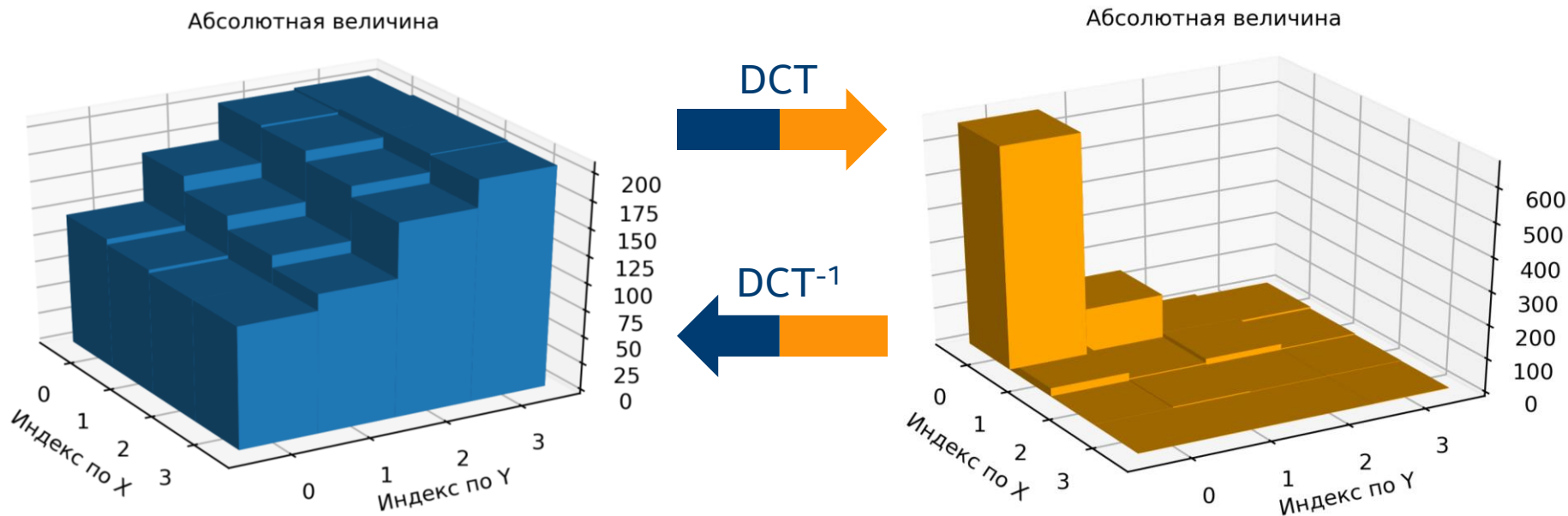
# Дискретное косинус-преобразование (DCT)



- Косинус-преобразование компактизует информацию о пикселях

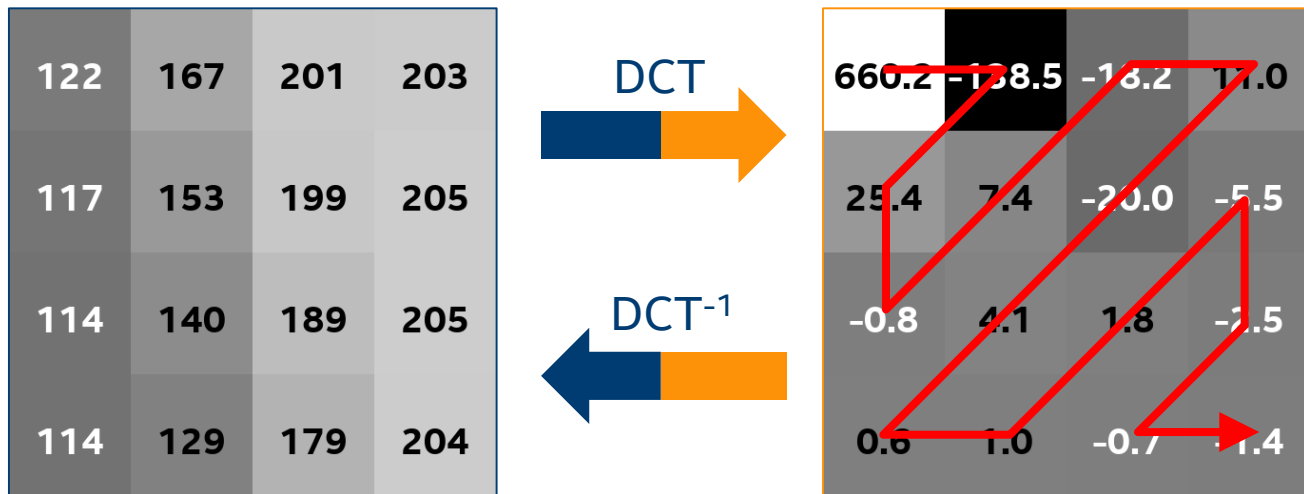


# Дискретное косинус-преобразование (DCT)



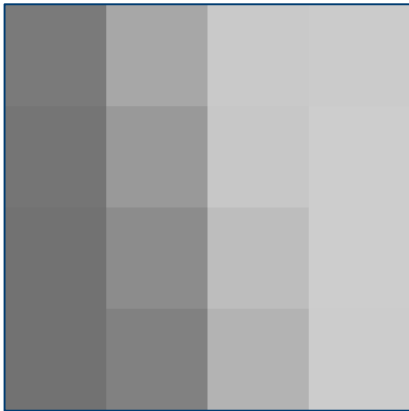
- Косинус-преобразование компактизует информацию о пикселях

# Дискретное косинус-преобразование (DCT)



- Наиболее значимыми оказываются коэффициенты с меньшими индексами
- Выгодно передавать коэффициенты в порядке «зигзага»

## Исходный блок



## Восстановленный блок

## Преобразованный блок



660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4



660.2	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

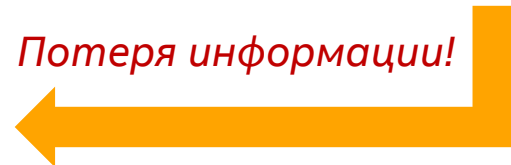
## Оптимизированный блок

Сохранение **1** коэффициента



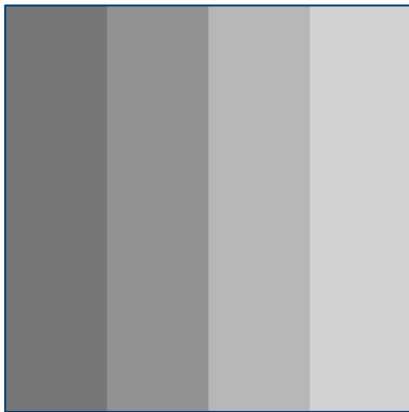
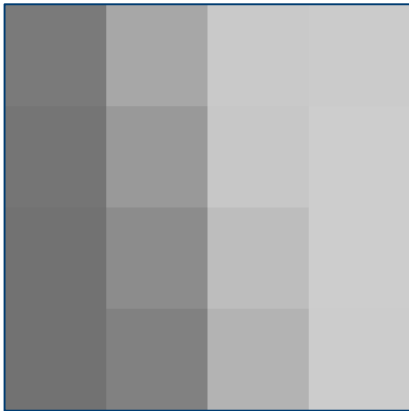
660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

Потеря информации!



Сжатие ~ 16x

Исходный блок



Восстановленный блок

DCT



Преобразованный блок

660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

DCT<sup>-1</sup>



660.2	-138.5	0	0
0	0	0	0
0	0	0	0
0	0	0	0

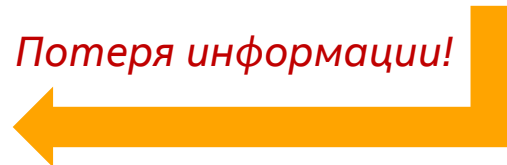
Оптимизированный блок

Сохранение 2 коэффициентов



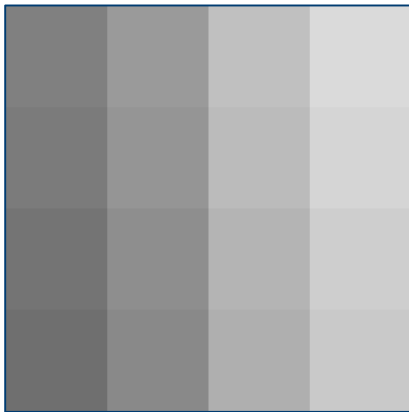
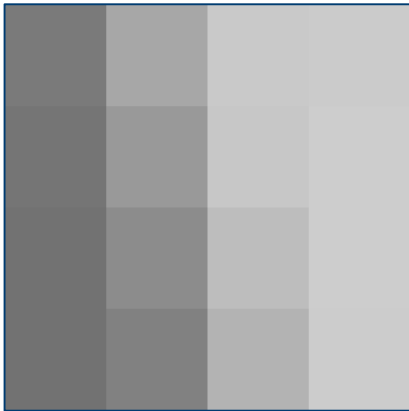
660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

Потеря информации!



Сжатие ~ 8x

Исходный блок



Восстановленный блок

DCT



Преобразованный блок

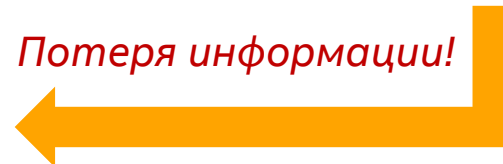
660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

Сохранение **3** коэффициентов



660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

Потеря информации!



Сжатие ~ 5x

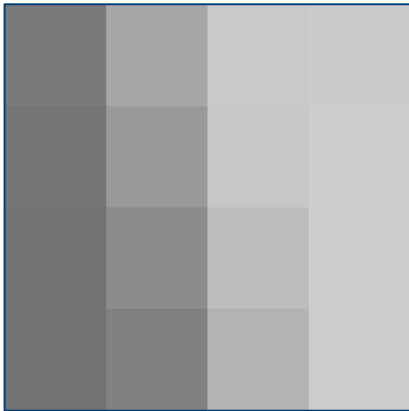
DCT<sup>-1</sup>



Оптимизированный блок

660.2	-138.5	0	0
25.4	0	0	0
0	0	0	0
0	0	0	0

Исходный блок



DCT



Преобразованный блок

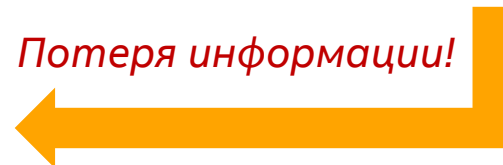
660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

Сохранение **6** коэффициентов



660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

Потеря информации!



Сжатие ~ 2.5x

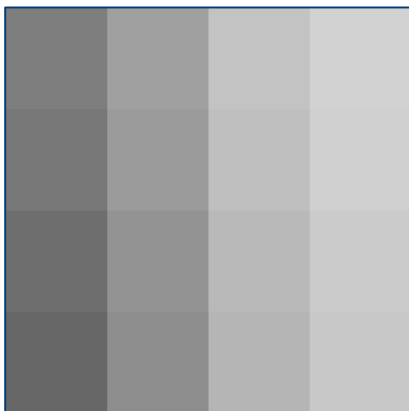
DCT<sup>-1</sup>



Оптимизированный блок

660.2	-138.5	-18.2	0
25.4	7.4	0	0
-0.8	0	0	0
0	0	0	0

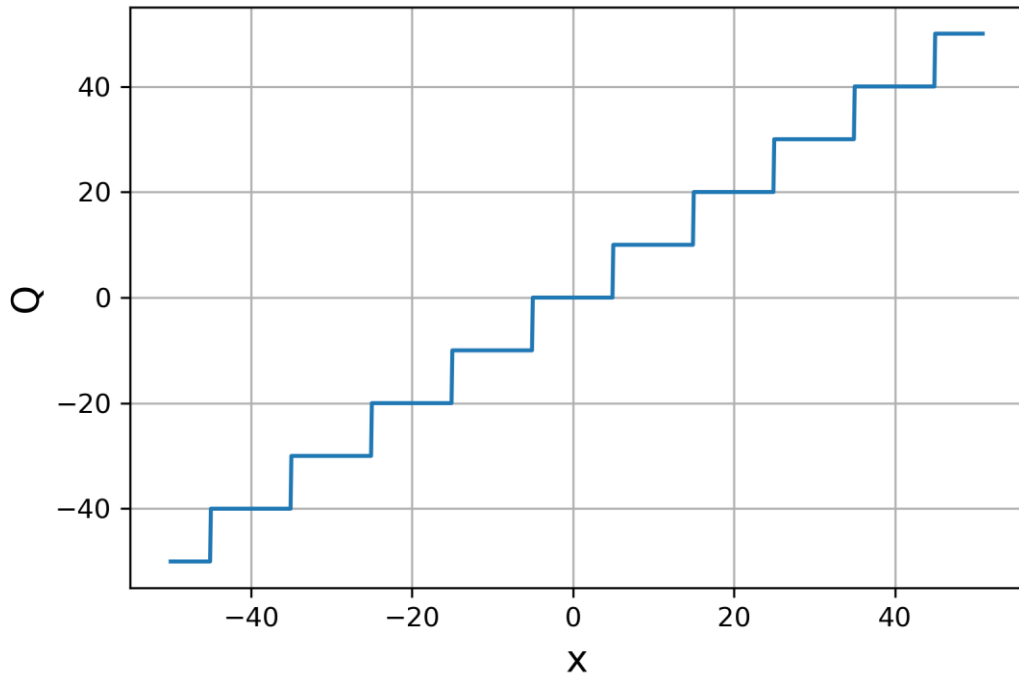
Восстановленный блок





# Квантизация коэффициентов ДКП

$Q(x, QP = 20)$



Линейный скалярный квантизатор:

$$\widetilde{F_{ij}} = Q(F_{ij}, QP)$$

$$Q(x, QP) = \text{round}\left(\frac{x}{QP}\right) * QP$$

$$\text{round}(t) = \text{sgn}(t) * \lfloor |t| \rfloor$$

- $QP$  – параметр квантизации, напрямую управляет качеством и степенью сжатия

660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

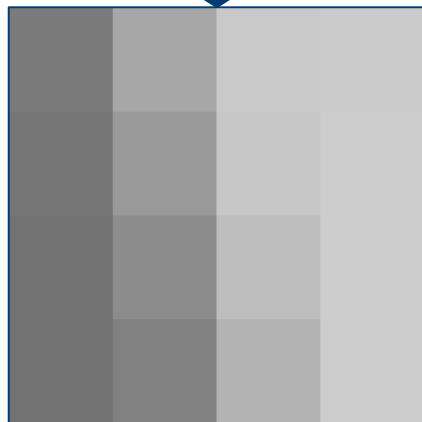
$$Q(F_{ij}, QP = 10)$$

Квантизация

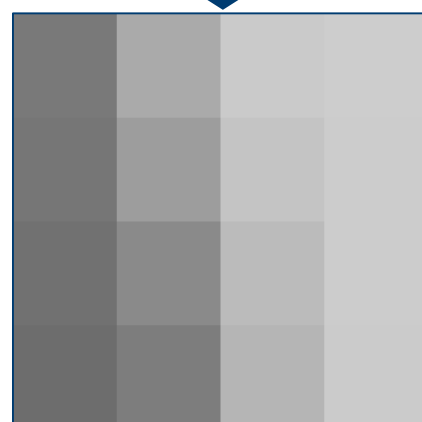
*Потеря информации!*

660	-140	-20	10
30	10	-20	-10
0	0	0	0
0	0	0	0

DCT<sup>-1</sup>



DCT<sup>-1</sup>



660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

$$Q(F_{ij}, QP = 10)$$

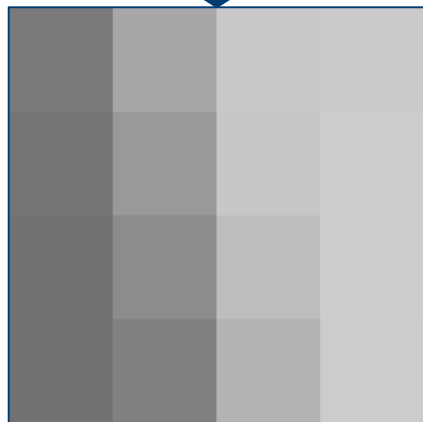
Квантизация

*Потеря информации!*

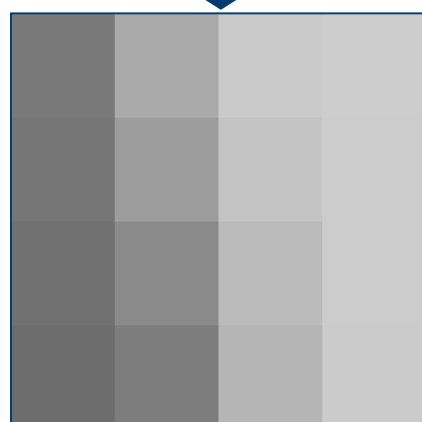
660	-140	-20	10
30	10	-20	-10
0	0	0	0
0	0	0	0

При **QP = 10**  
последние 3  
коэффициента в  
зигзаговом порядке  
равны 0 – передаем  
только первые **13**  
коэффициентов

DCT<sup>-1</sup>



DCT<sup>-1</sup>



*Сжатие ~ 1.2x*

660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

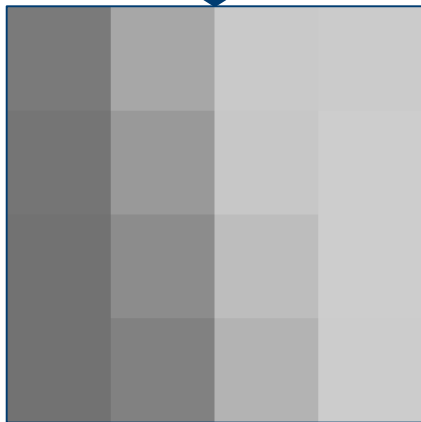
$$Q(F_{ij}, QP = 50)$$

Квантизация

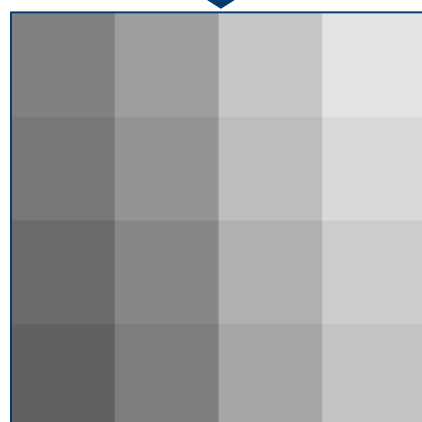
*Потеря информации!*

650	-150	0	0
50	0	0	0
0	0	0	0
0	0	0	0

DCT<sup>-1</sup>



DCT<sup>-1</sup>



660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

$$Q(F_{ij}, QP = 50)$$

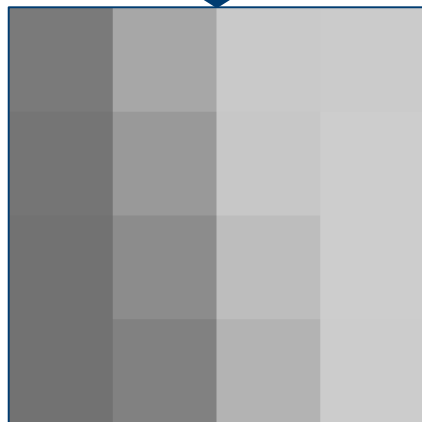
Квантизация

*Потеря информации!*

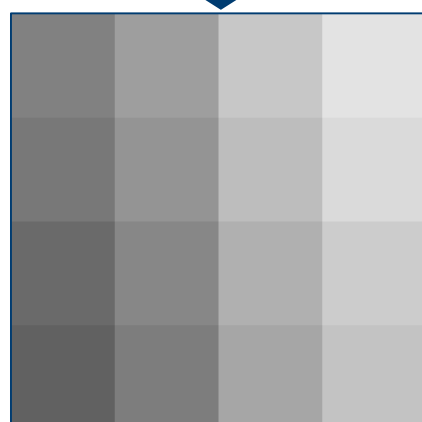
650	-150	0	0
50	0	0	0
0	0	0	0
0	0	0	0

При **QP = 50**  
последние 13  
коэффициентов в  
зигзаговом порядке  
равны 0 – передаем  
только первые **3**  
коэффициента

DCT<sup>-1</sup>



DCT<sup>-1</sup>



*Сжатие ~ 5x*

660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

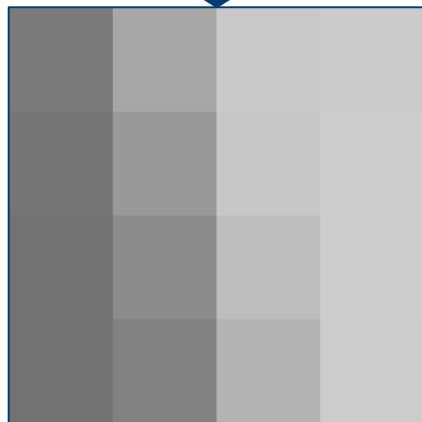
$$Q(F_{ij}, QP = 100)$$

Квантизация

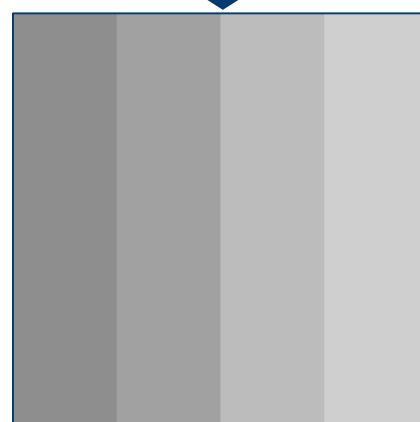
*Потеря информации!*

700	-100	0	0
0	0	0	0
0	0	0	0
0	0	0	0

DCT<sup>-1</sup>



DCT<sup>-1</sup>





660.2	-138.5	-18.2	11.0
25.4	7.4	-20.0	-5.5
-0.8	4.1	1.8	-2.5
0.6	1.0	-0.7	-1.4

$$Q(F_{ij}, QP = 100)$$

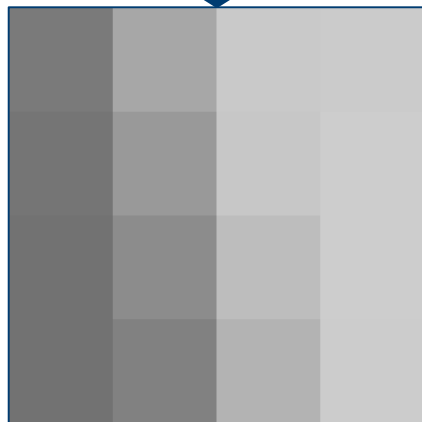
Квантизация

*Потеря информации!*

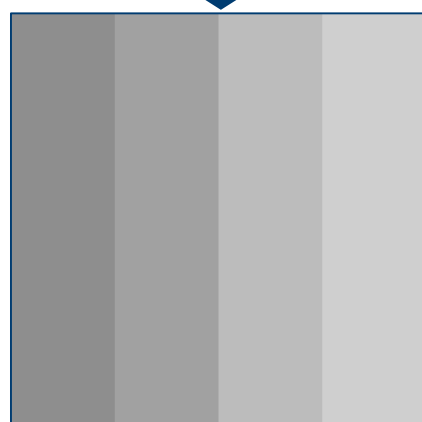
700	-100	0	0
0	0	0	0
0	0	0	0
0	0	0	0

При **QP = 100**  
последние 14  
коэффициентов в  
зигзаговом порядке  
равны 0 – передаем  
только первые **2**  
коэффициента

DCT<sup>-1</sup>

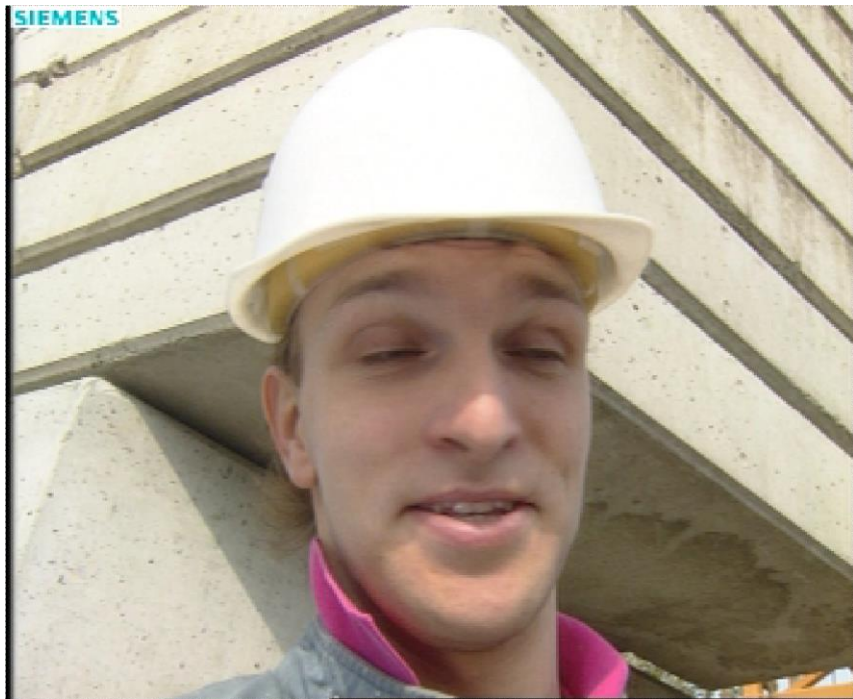


DCT<sup>-1</sup>



*Сжатие ~ 8x*

# Временная избыточность



Кадр 1/300



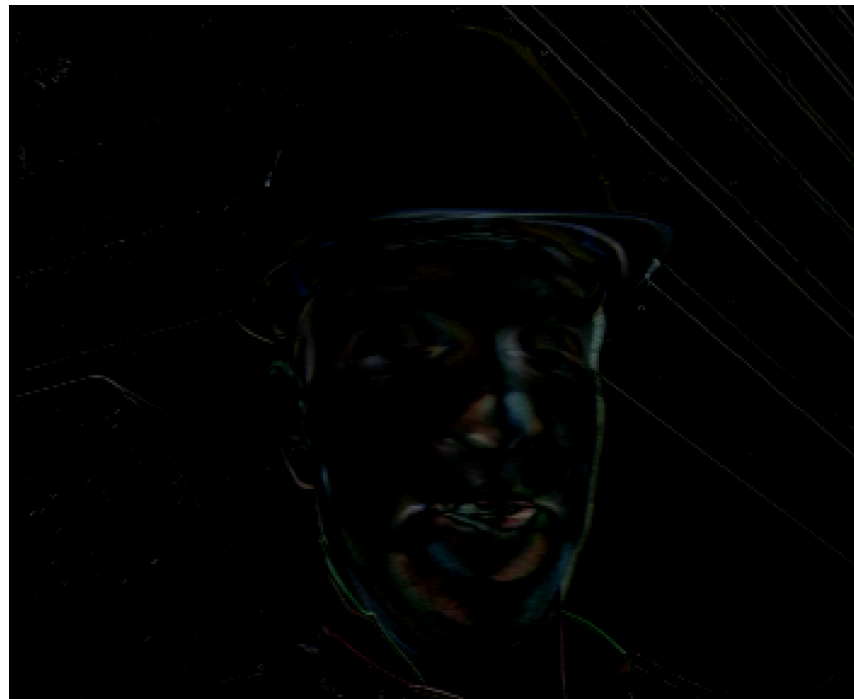
Кадр 2/300

- Обычно текущий кадр на видео похож на предыдущие

# Временная избыточность



Кадр 1/300



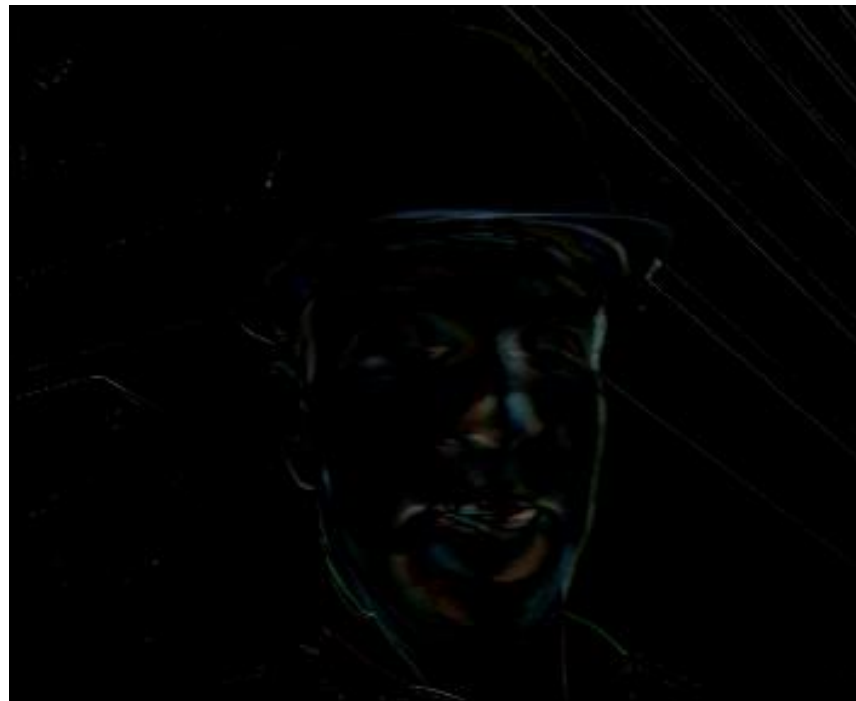
Разность кадра 2/300 и кадра 1/300

- Разность кадров кодируется эффективнее, чем целый кадр, но несет столько же информации

# Временная избыточность



Оригинальное видео

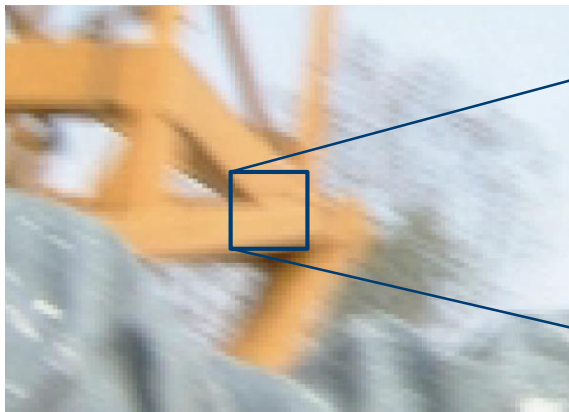


Видео, составленное из разностных кадров

- Кодирование разности менее эффективно для видео с выраженным движением объектов

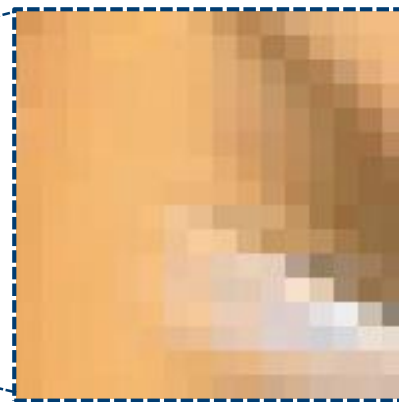
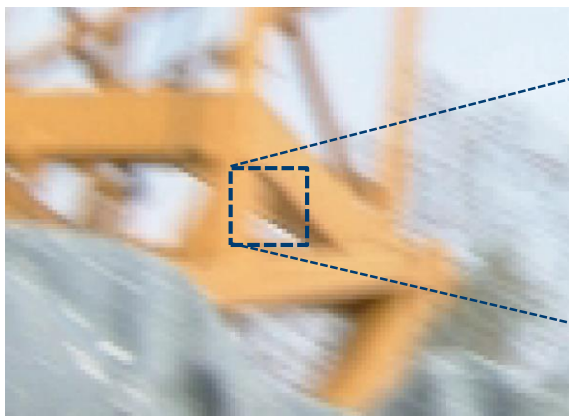
# Компенсация движения (motion compensation)

Кадр 192



Блок 16x16,  
(X;Y) = (128, 224)

Кадр 191

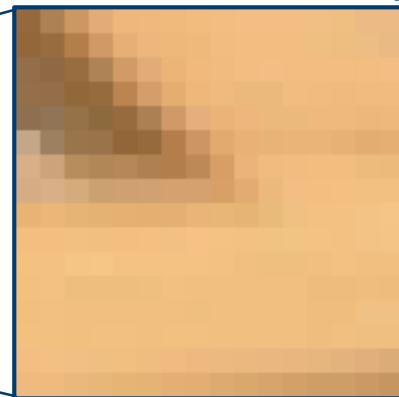
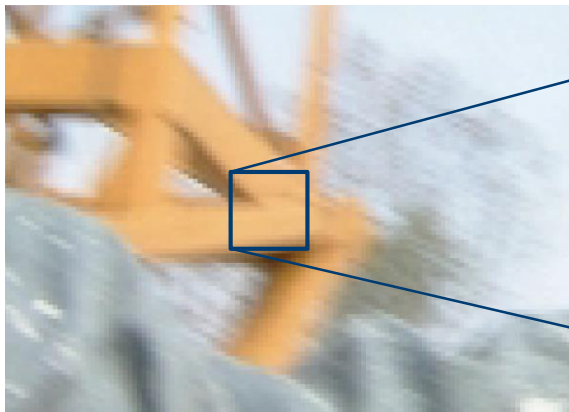


Совмещенные блоки  
на соседних кадрах  
недостаточно похожи!

Блок 16x16,  
(X;Y) = (128, 224)

# Компенсация движения (motion compensation)

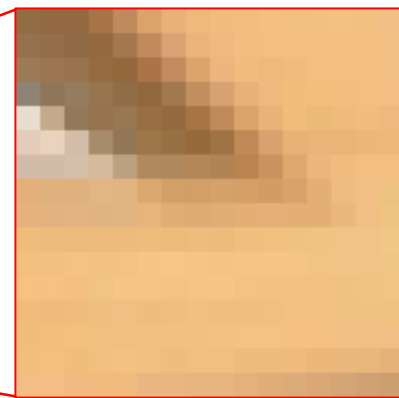
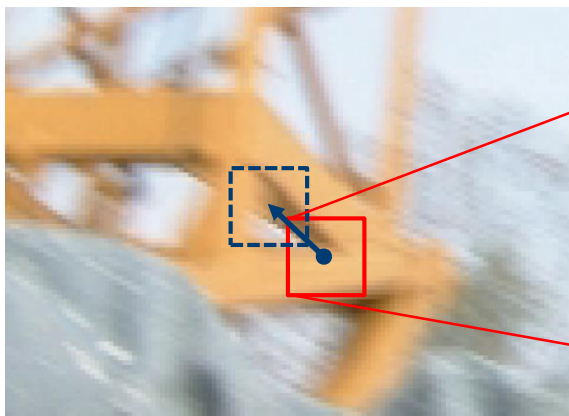
Кадр 192



Блок 16x16,  
(X;Y) = (128, 224)

Блок с прошлого кадра,  
сдвинутый относительно  
предыдущего, похож  
гораздо сильнее!

Кадр 191

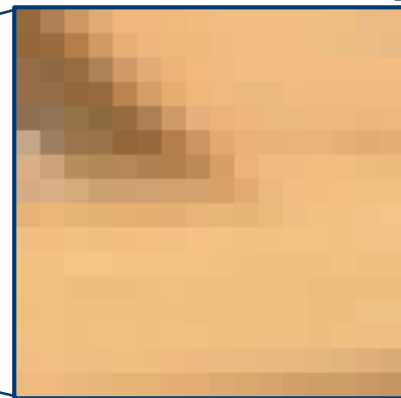
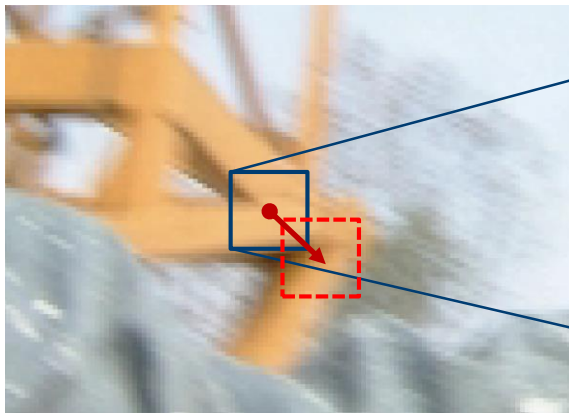


Блок 16x16,  
(X;Y) = (143, 233)



# Компенсация движения (motion compensation)

Кадр 192



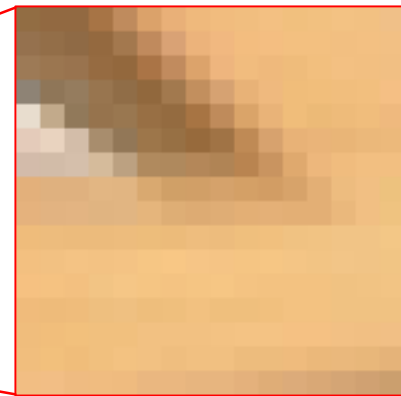
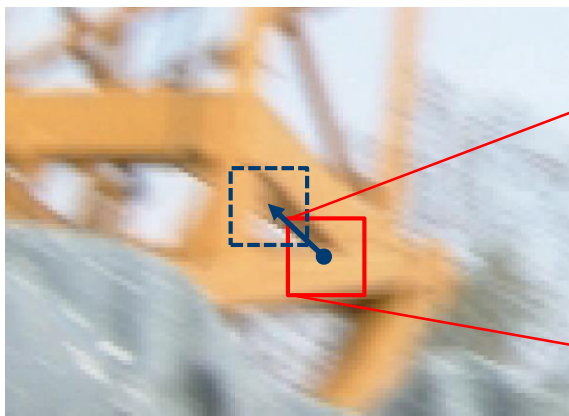
Блок 16x16,  
(X;Y) = (128, 224)

+

$\overrightarrow{MV} = (15;9)$

Вместе с каждым блоком  
передается вектор  
движения относительно  
прошлого кадра

Кадр 191



Блок 16x16,  
(X;Y) = (143, 233)

# Компенсация движения (motion compensation)



- Информация о векторах движения передается внутри закодированного видеопотока вместе с информацией о разностных пикселях
- Разностный блок вычисляется относительно текущего блока и блока на прошлом кадре, смещенного на вектор движения
- Процесс поиска энкодером правильных векторов движения – **«интер-предсказание» (inter prediction)**



# Внутрикадровое предсказание (intra prediction)

Кадр 99



Кадр 100



Кадр 101



Интер-предсказание

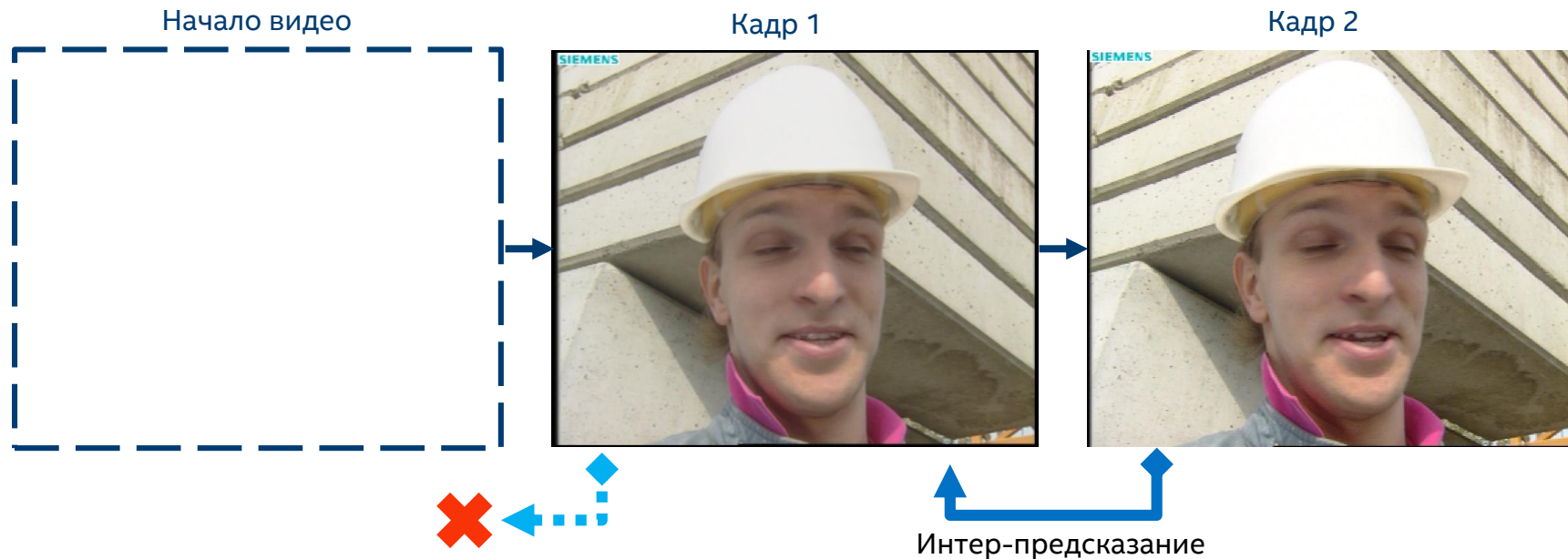


Интер-предсказание



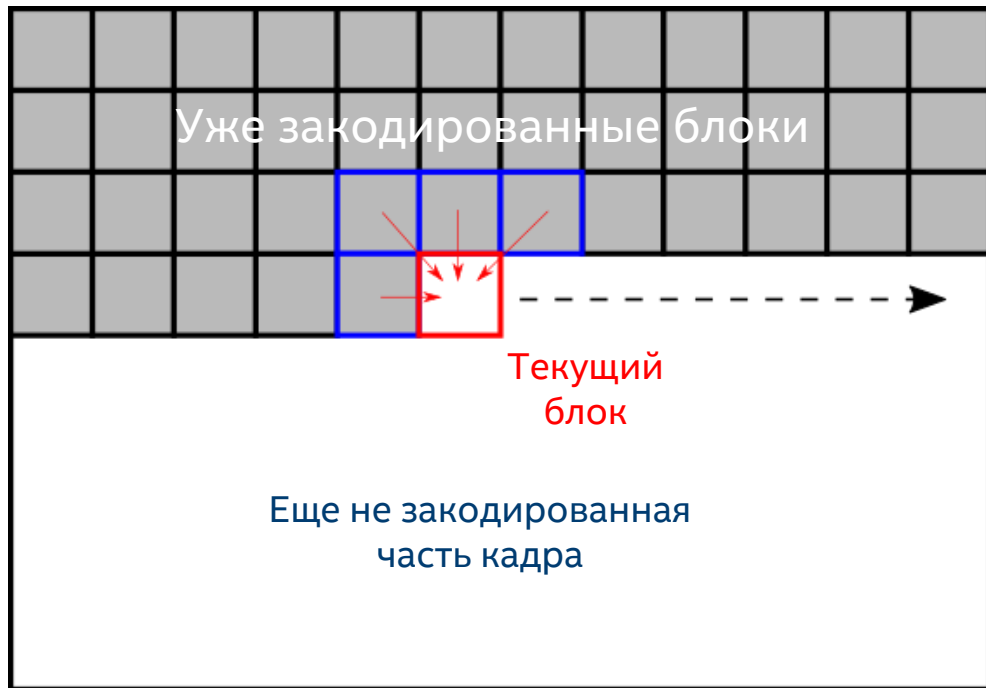
- Для некоторых кадров/частей кадров интер-предсказание невозможно или неэффективно

# Внутрикадровое предсказание (intra prediction)



- Для некоторых кадров/частей кадров интер-предсказание невозможно или неэффективно

# Внутрикадровое предсказание (intra prediction)

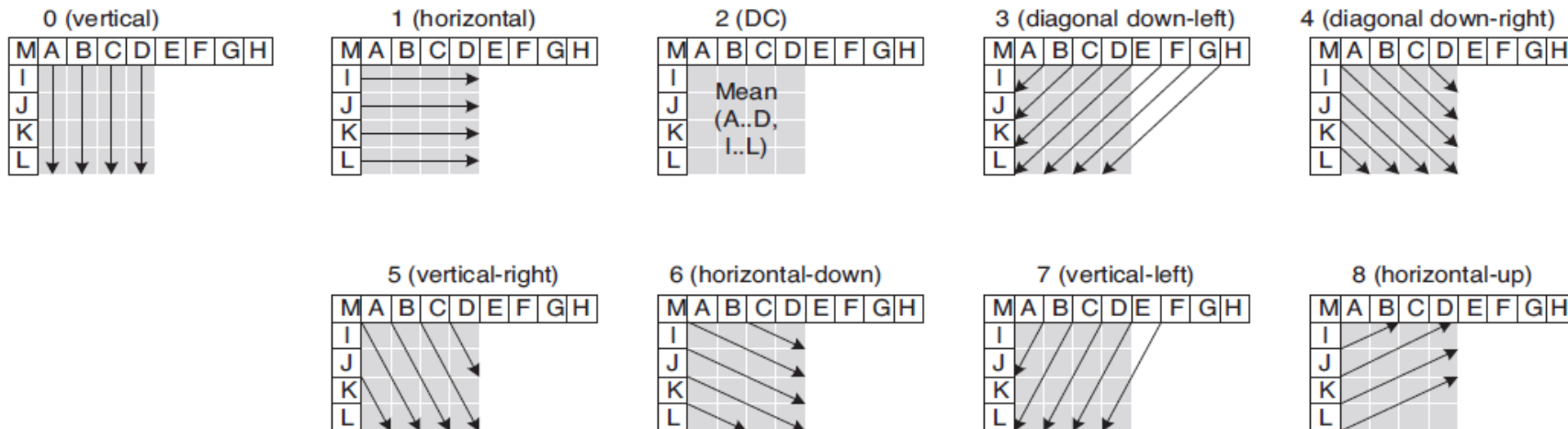


Кадр видео в процессе кодирования

- Обработка кадра происходит в блочном порядке, слева-направо, сверху-вниз
- Для текущего блока доступна информация о предыдущих закодированных блоках
- Соседние с текущим, уже закодированные блоки пикселей используются для формирования предсказания для текущего блока

# Внутрикадровое предсказание (intra prediction)

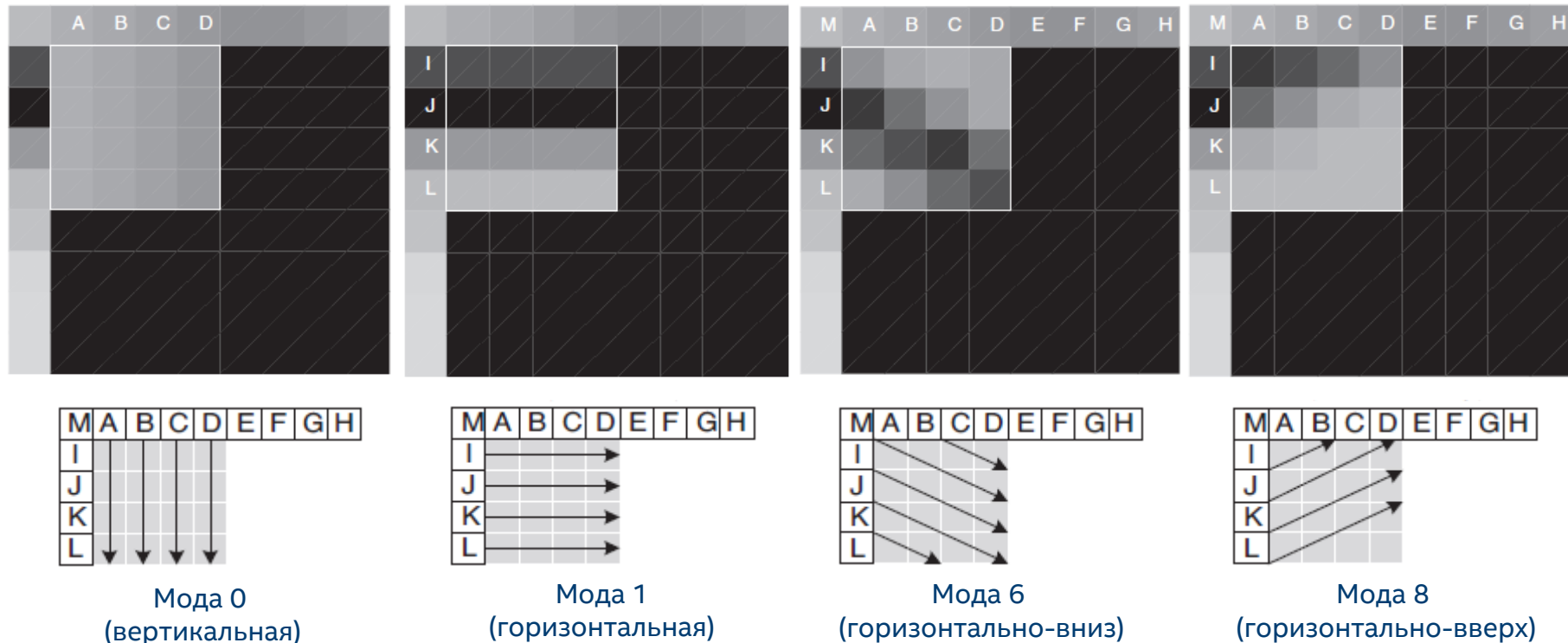
## Интра-моды кодака H.264 (AVC)



Источник: I. Richardson, "The H.264 Advanced Video Compression Standard, Second Edition"

- Интра-предсказанный блок составляется из экстраполированных соседних с блоком пикселей (уже закодированных/раскодированных)

# Внутрикадровое предсказание (intra prediction)

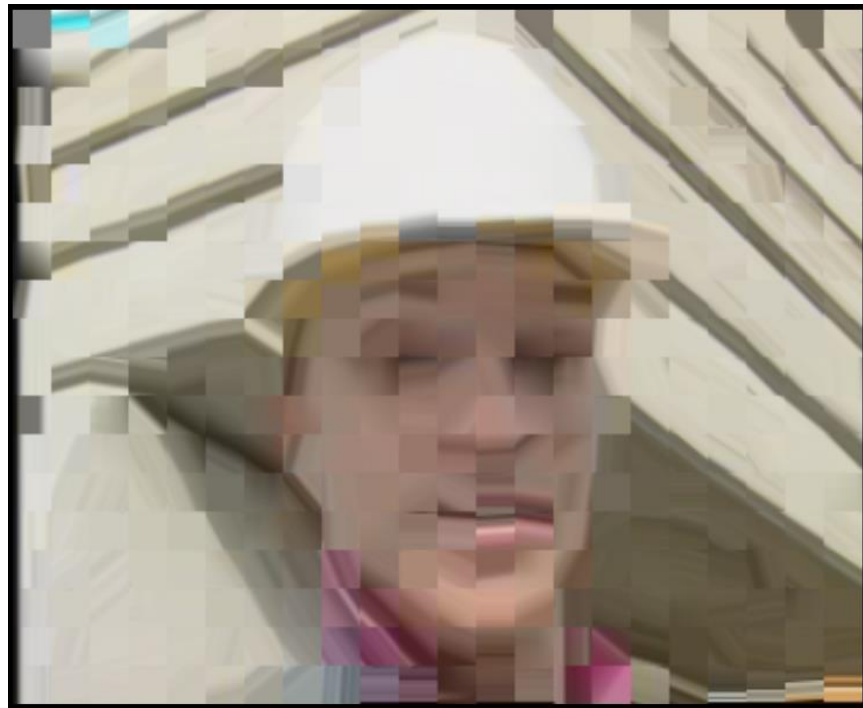


Источник: I. Richardson, "The H.264 Advanced Video Compression Standard, Second Edition"

# Внутрикадровое предсказание (intra prediction)



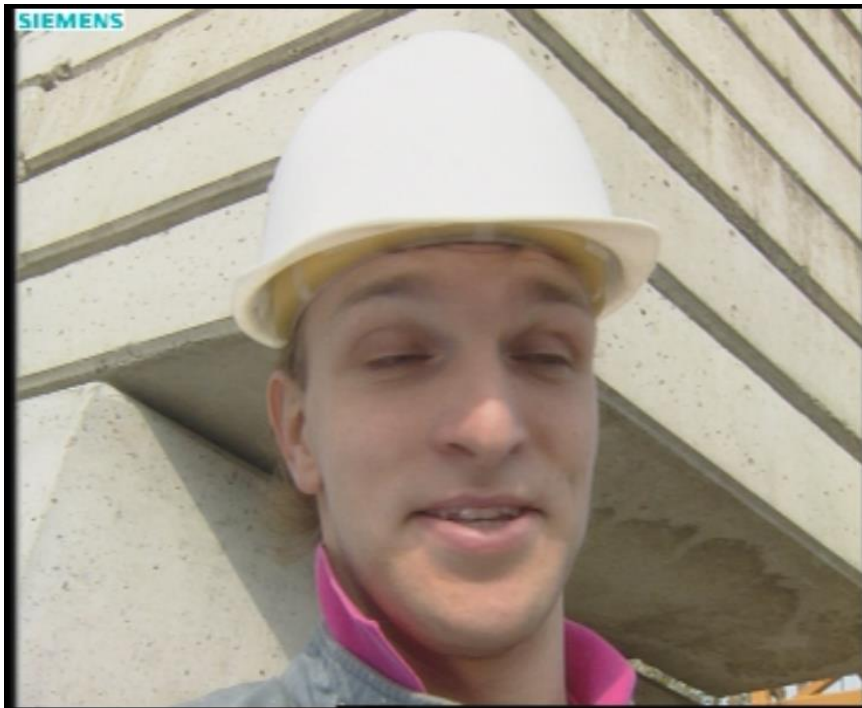
Оригинальный кадр



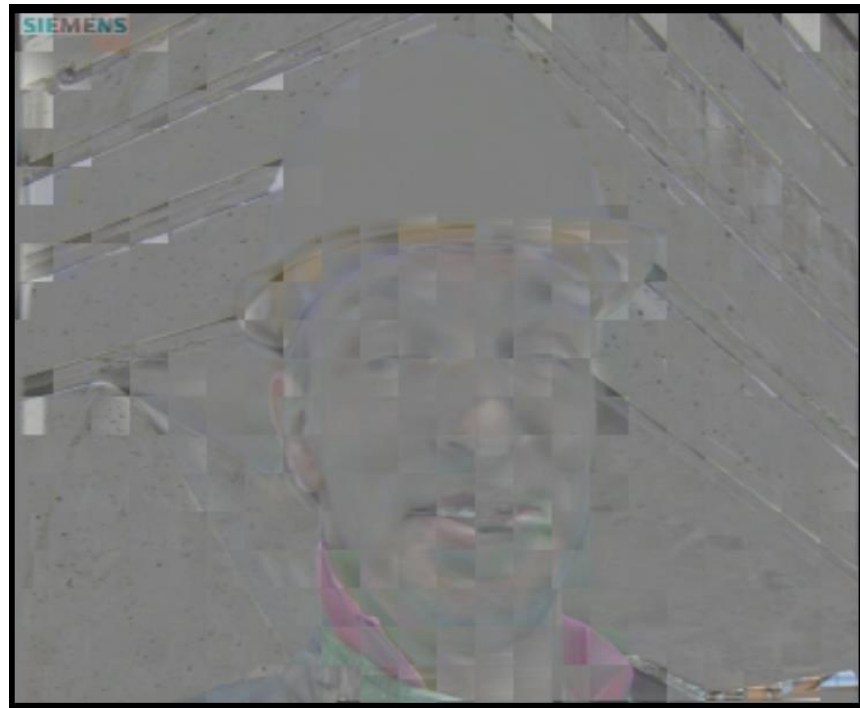
Интра-предсказанный кадр



# Внутрикадровое предсказание (intra prediction)



Оригинальный кадр



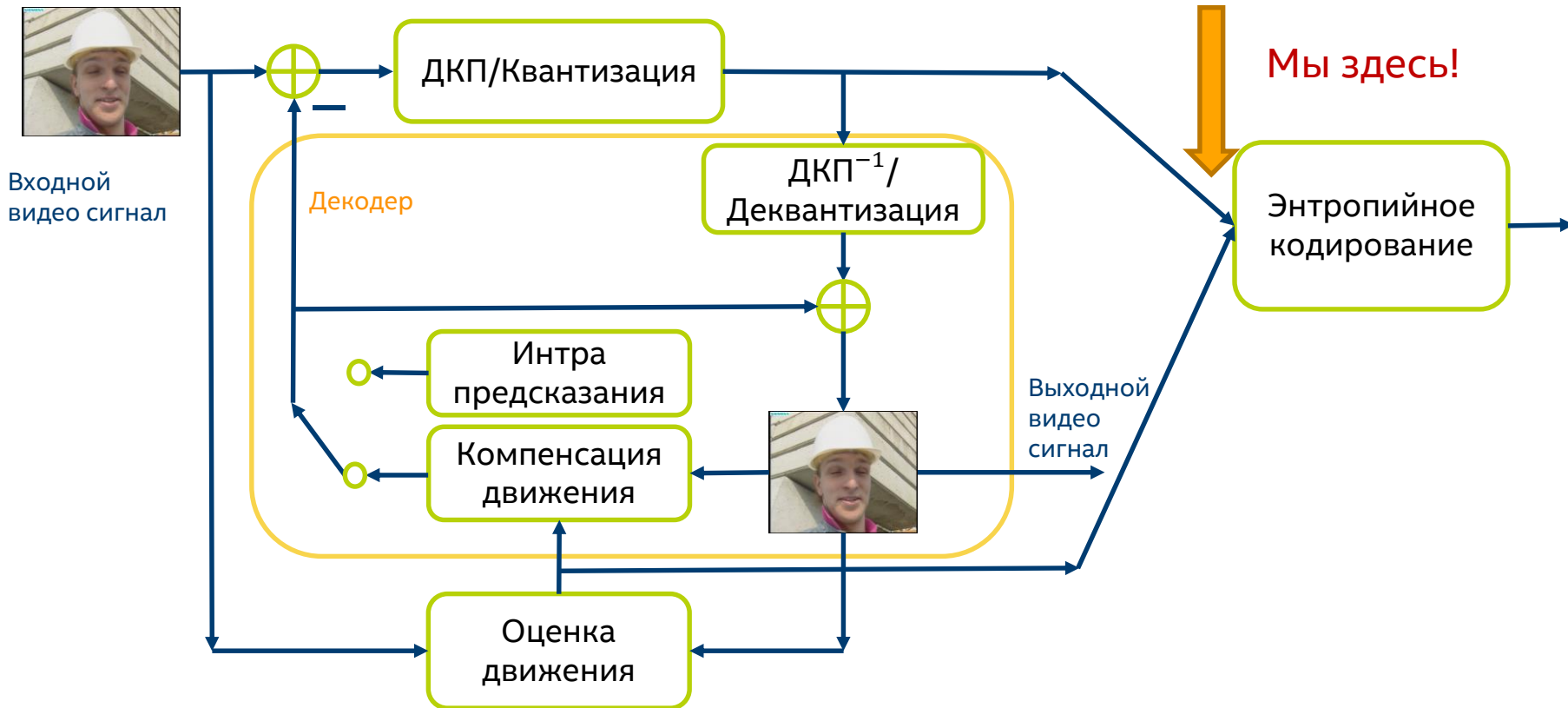
Разница оригинального и интра-предсказанного кадра

- Интра-предсказание менее эффективно, чем интер-предсказание

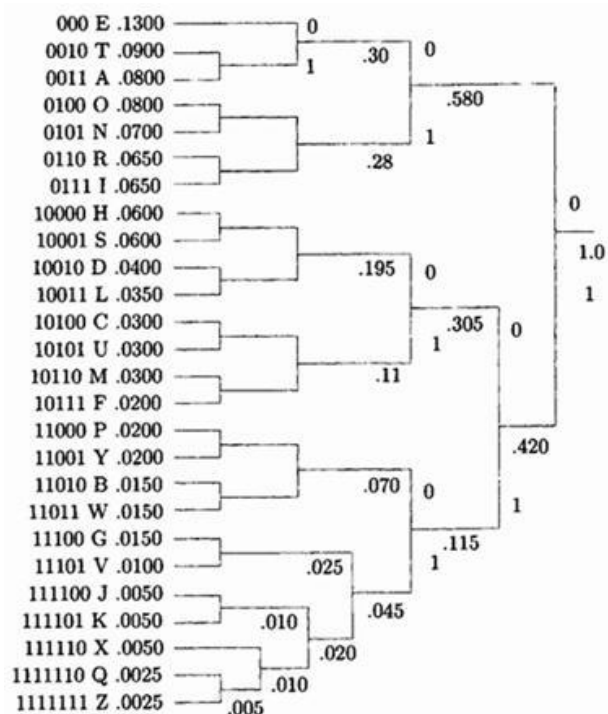




# Конвейер работы энкодера и декодера



# Энтропийное кодирование



Входное слово: **EDUCATED**

- Код Хаффмана:

000 10010 10101 10100 0011 0010 000 10010 (34 бита)

- Бинарный ASCII код:

01000101 01000100 01010101 01000011 01000001  
01010100 01000101 01000100 (64 бита)

Имеем практически двойное сжатие!

# Энтропийное кодирование



*Источник: Jan Ozer,  
Encoding H.264 Video for  
Streaming and Progressive  
Download*

- **CAVLC - кодирование переменной длины** на основе контекста

Менее эффективно с точки зрения качества при заданном размере, проще декодировать

- **CABAC - адаптивное арифметическое кодирование** на основе контекста

Более эффективно с точки зрения качества при заданном размере, сложнее декодировать

# Современные видеокодеки

Кодек	MPEG2	H.264 (AVC)	H.265 (HEVC)	VP8	VP9	AV1
Год первого издания	1996	2003	2013	2008	2013	2018
Лицензия	Платный	Платный	Платный	Открытый	Открытый	Открытый
Места применения	DVD, спутниковое ТВ	Всюду	Видео HD и Ultra HD-качества, FaceTime, HEIF	WebM, Youtube	WebM, Youtube (Ultra HD)	HTML5-видео, WebRTC

# Как сравнивают кодеки?



# Как сравнивают кодеки?

$$MSE = \frac{1}{WH} \sum_{i,j=0}^{W-1, H-1} (\widetilde{S}_{ij} - S_{ij})^2$$

- среднеквадратичная ошибка

$$PSNR_{dB} = 10 \log_{10} \frac{255^2}{MSE}$$

- отношение «сигнал-шум»

- $W$  – ширина кадра в пикселях,  $H$  – высота кадра в пикселях,
- $S_{ij}, \widetilde{S}_{ij}$  – значения пикселей сравниваемых кадров

# Как сравнивают кодеки?



512x512  
768 КБ

368x368  
(растянутая  
до 512x512)  
384 КБ



$$PSNR_{dB} = 30.72$$

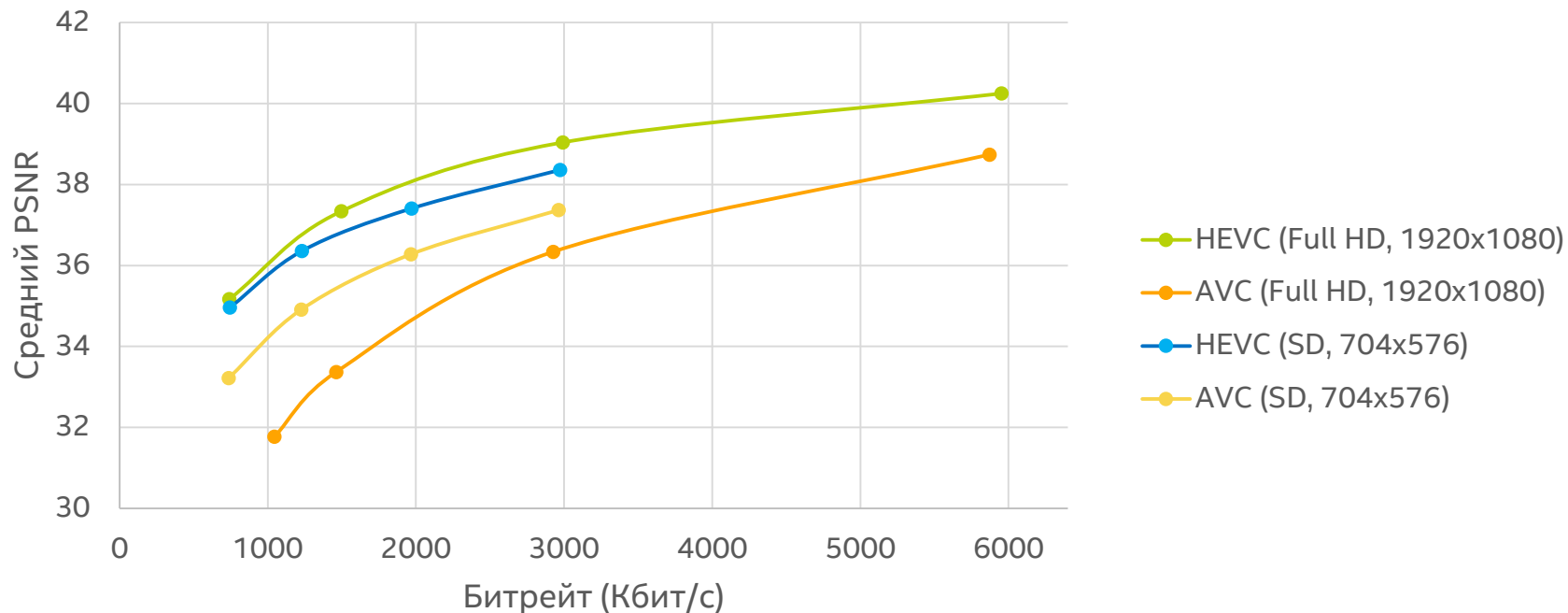
64x64  
(растянутая  
до 512x512)  
12 КБ



$$PSNR_{dB} = 19.23$$

# Как сравнивают кодеки?

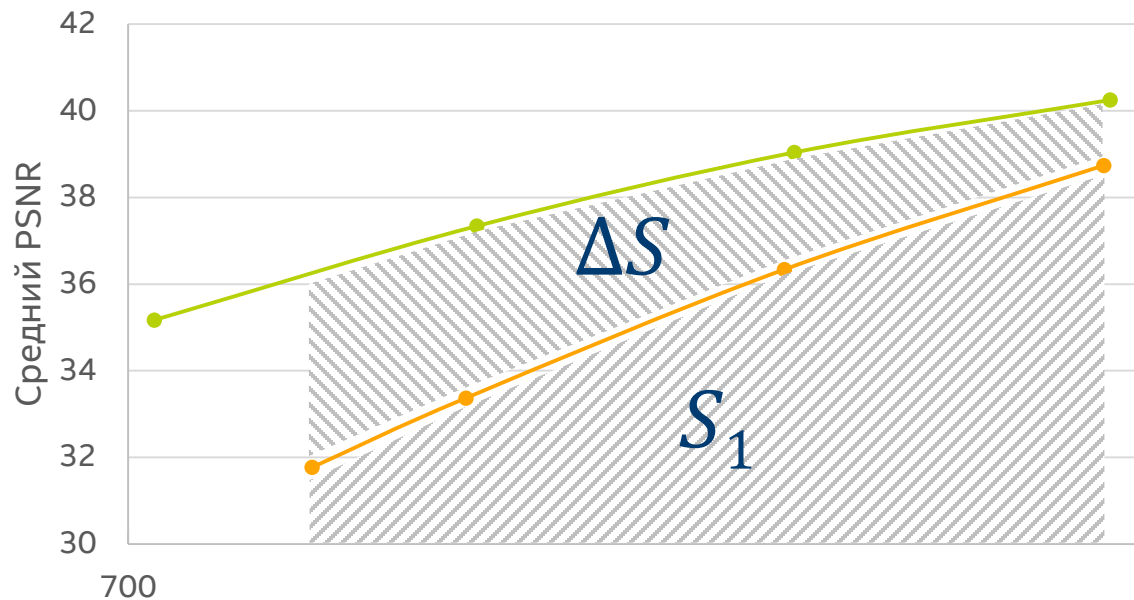
H.264 (AVC) vs. H.265 (HEVC)





# Как сравнивают кодеки?

H.264 (AVC) vs. H.265 (HEVC)



$$\text{BD-Rate} = \frac{\Delta S}{S_1} = \frac{S_2 - S_1}{S_1}$$

— HEVC (Full HD, 1920x1080)

— AVC (Full HD, 1920x1080)


Битрейт (Кбит/с) – логарифмическая шкала




# Intel® Media SDK







- <https://software.intel.com/en-us/media-sdk>
- A. K. A.: MSDK, QSV (Intel® Quick Sync Video), libmfx
- API и пользовательские библиотеки для приложений под Windows и Linux
- Аппаратное ускорение процессов кодирования, декодирования и обработки видео с помощью возможностей Intel® Graphics Technology
- Интегрирован в FFmpeg (h264\_qsv, hevc\_qsv)

# Intel® Media SDK

- Открытый исходный код (Linux): <https://github.com/Intel-Media-SDK/MediaSDK>






 Intel-Media-SDK / MediaSDK

 Watch 76  Star 318  Fork 208


 Code  Issues 116  Pull requests 36  Wiki  Security  Insights

The Intel® Media SDK <http://mediasdk.intel.com>

intel-media-sdk qsv mfx

 1,280 commits  10 branches  29 releases  81 contributors  MIT

Branch: master ▾ New pull request Find File Clone or download ▾

 mgonchar and onabiull vp9e: fix comparison of segment maps ... Latest commit 62911ba 2 hours ago

# Intel® Media SDK

- Начиная с Ubuntu 19.04, может быть установлен из основного репозитория Ubuntu



The screenshot shows the Ubuntu Packages website for the `intel-mediasdk` source package. The header is orange with the Ubuntu logo and a search bar. The breadcrumb trail is: » Ubuntu » Packages » disco (19.04) » Source » libs » intel-mediasdk. The main heading is "Source Package: intel-mediasdk (18.4.1-0ubuntu1) [universe]" with version and distribution information. Below this, it lists binary packages built from the source: `libmfx-dev` (development files), `libmfx-tools` (tools), and `libmfx1` (shared library). On the right, there are links for the package and Ubuntu resources.

ubuntu<sup>®</sup> packages

source package names ▼ Search [all options](#)

» Ubuntu » Packages » disco (19.04) » Source » libs » intel-mediasdk

Source Package: intel-mediasdk (18.4.1-0ubuntu1) [universe] [ disco ] [ eoan ]

The following binary packages are built from this source package:

- `libmfx-dev`  
Intel Media SDK -- development files
- `libmfx-tools`  
Intel Media SDK -- tools
- `libmfx1`  
Intel Media SDK -- shared library

Links for intel-mediasdk

Ubuntu Resources:

- Bug Reports
- Ubuntu Changelog

