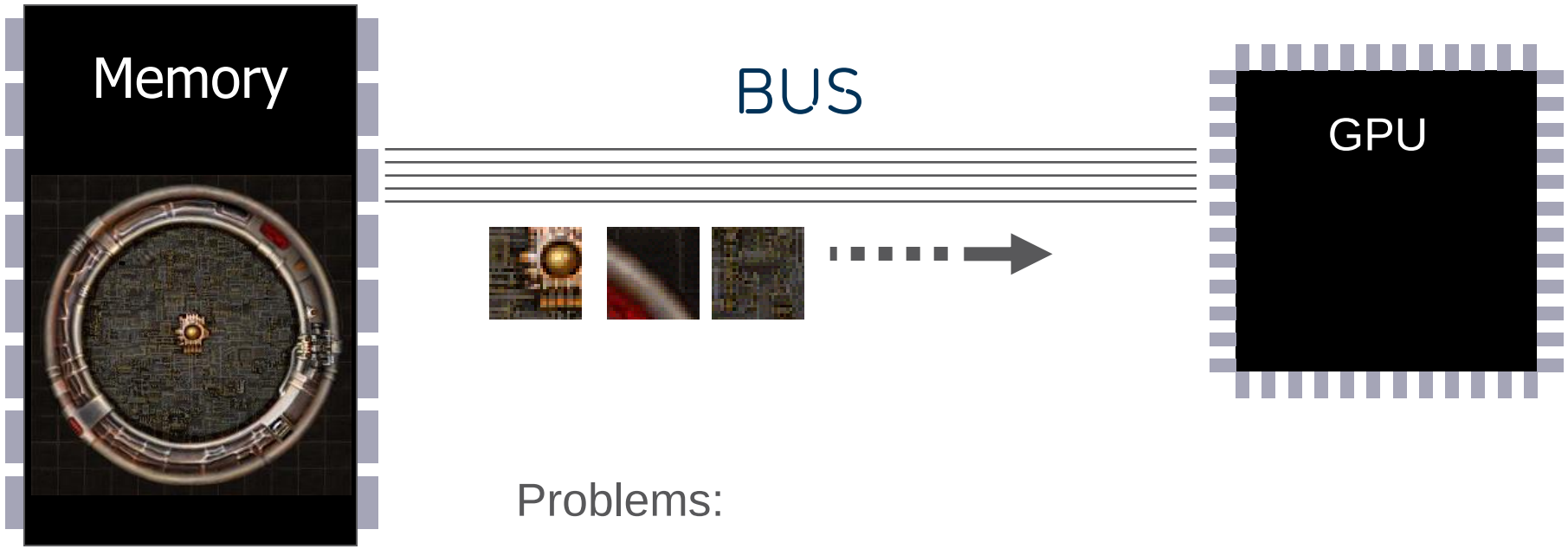
COMPRESSING ALREADY COMPRESSED TEXTURES

JACOB STRÖM AND PER WENNERSTEN
ERICSSON RESEARCH

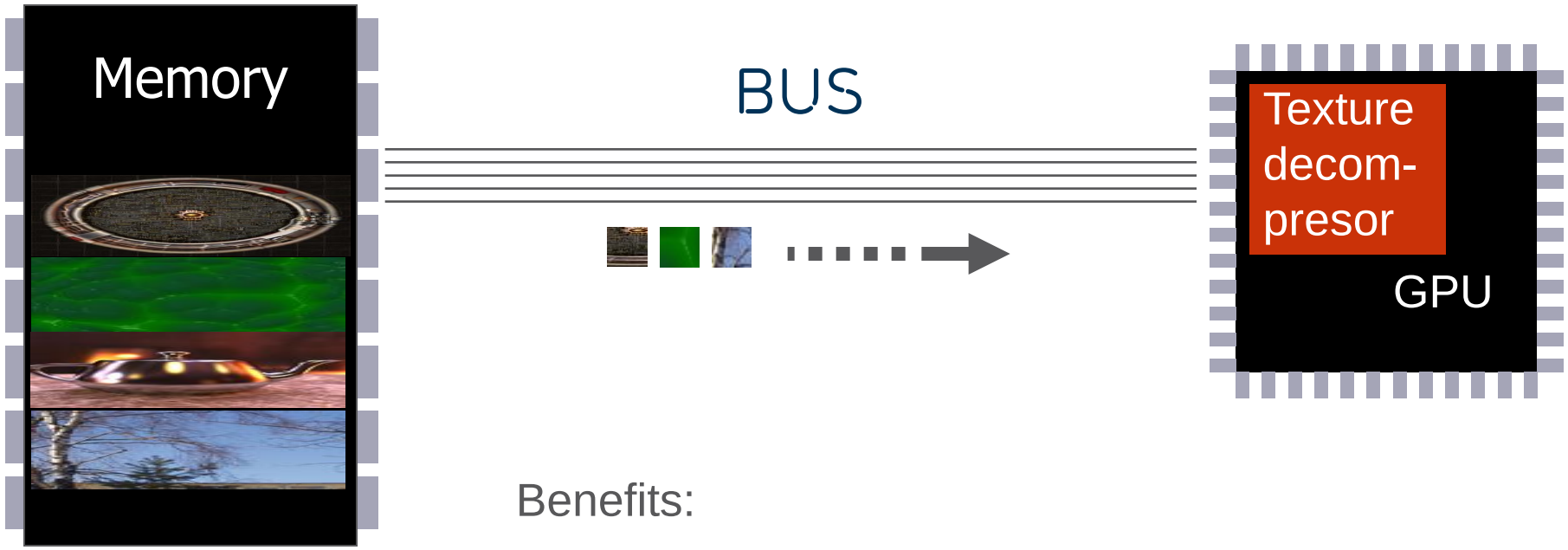
TEXTURING AND THE BUS



Problems:

- Memory can get full
- Bus can get full (performance bottleneck)

TEXTURE COMPRESSION HELPS



Benefits:

- More textures fit in memory
- Less traffic on bus = higher performance
- Less traffic on bus = lower power consumption

EVEN COMPRESSED TEXTURES TAKE TOO MUCH SPACE

THIS CAN BE A PROBLEM WHEN

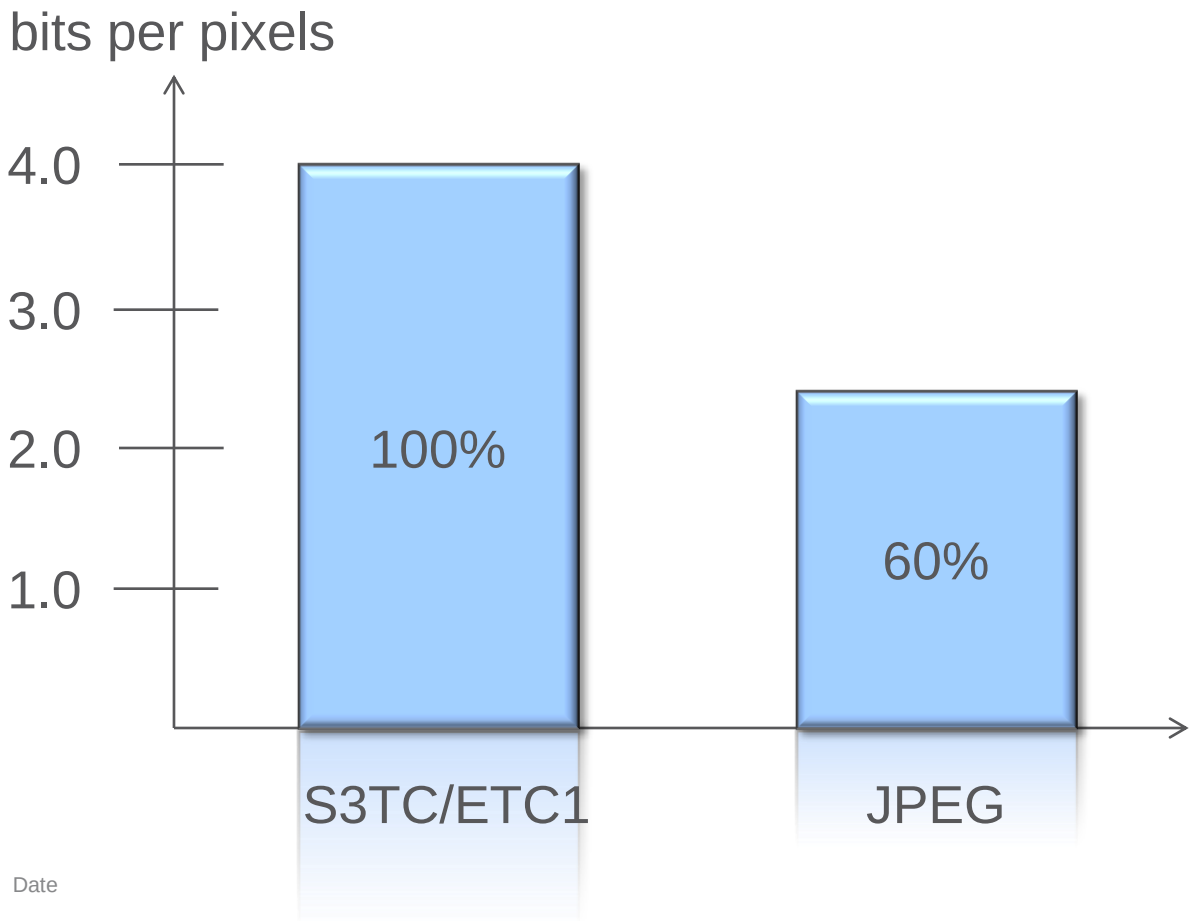
- There are file size restrictions
  - Android Market just increased maximum size from 50 MB to 4 GB.

- You want to download something over the air
  - Even short downloading times can be too long – a customer can lose interest

- You want to stream textures from disk/network
  - Textures may be consumed faster than your throughput

COMPARISON TO JPEG

› At equal quality, JPEG only uses 60% of the bits of texture compression methods such as S3TC and ETC1.



RANDOM ACCESS

› Texture compression needs random access, obtained by fixed rate coding.

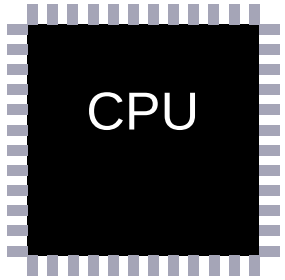
› JPEG does not have that constraint – simple areas can be coded with few bits.

TRADITIONAL WAYS OF SOLVING THIS

- van Waveren, J., 2006. *Real-Time Texture Streaming and Decompression*. Id Software Technical Report

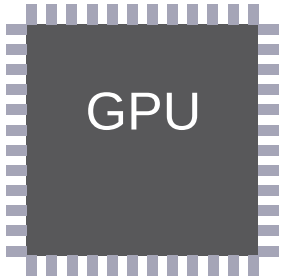
1. Store in JPEG-like format on disk

2. Read when needed for rendering.



3. Transcode on-the-fly to S3TC

4. Transfer to GPU memory

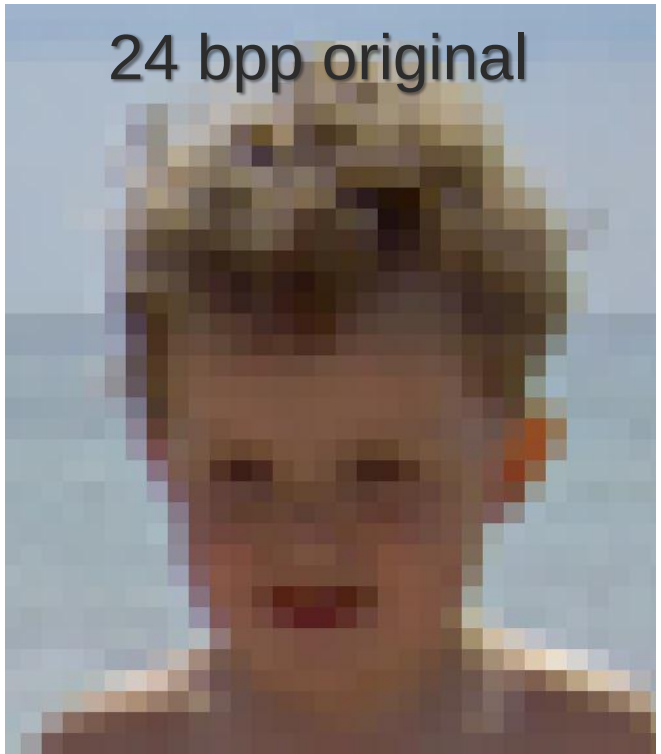


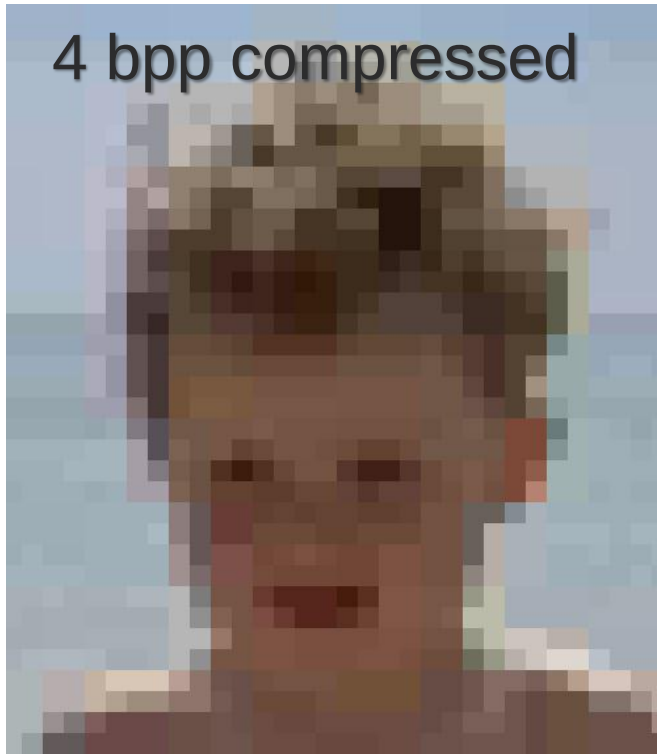
5. Render from S3TC.

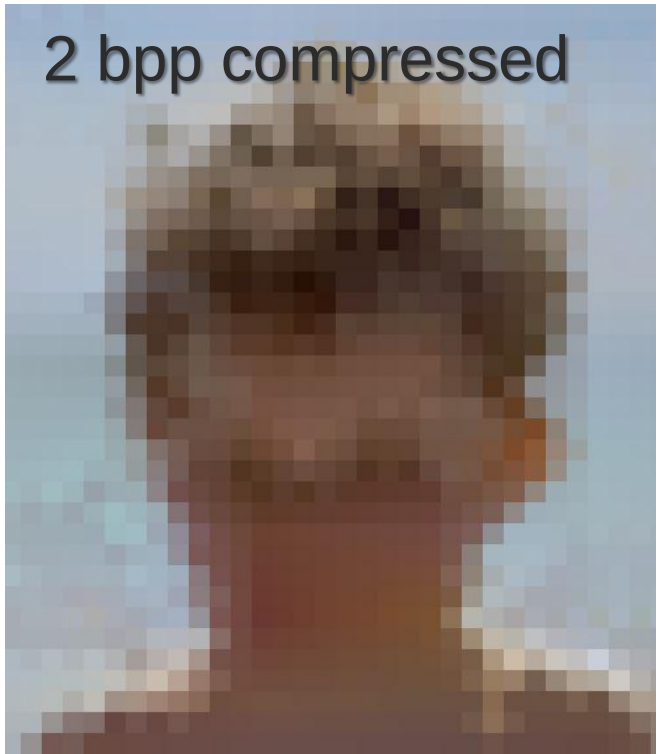
- Pro: Can utilize the compression efficiency of JPEG
- Con: Transcoding gives image artifacts both from JPEG and S3TC, and must happen fast (giving yet more artifacts)

TRADITIONAL WAYS OF SOLVING THIS

- Choose a lower bit rate texture compression
  - This may not give the reqiured quality
  - Resulting file sizes may still be too big
  - May not be available on some platforms

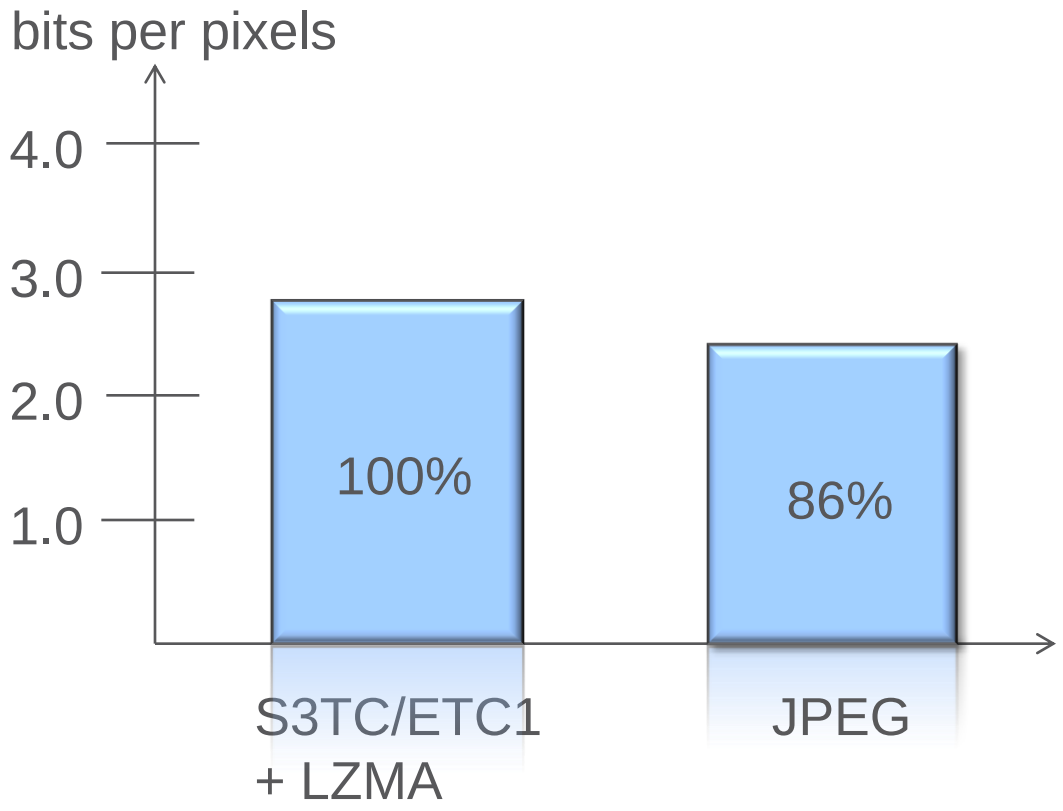






TRADITIONAL WAYS OF SOLVING THIS

- Compressing textures with ZIP, GZIP or LZMA
  - JPEG still 86% of the bits at equal quality



DOMAIN SPECIFIC KNOWLEDGE

- van Waveren also investigates compressing S3TC data using domain specific knowledge
- Each 4x4 block of S3TC data consists of colors and indices
  - colors are put into images and compressed with JPEG
  - indices are compressed using ZIP/LZA
- But:
  - Indices turn out to be hard to compress further than 75% of original size
  - Lossy coding of colors will give some additional artifacts

COULD WE DO BETTER?

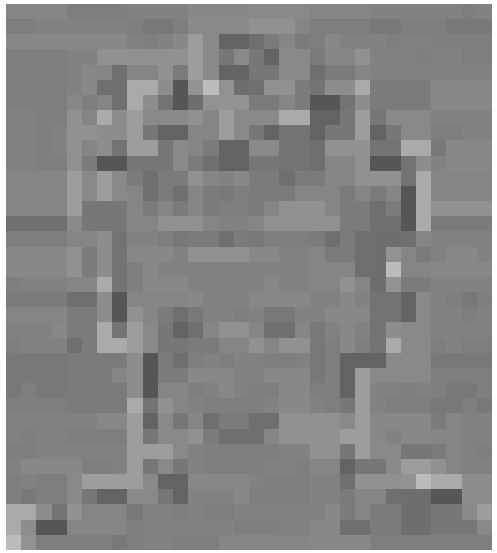
- Since the index data makes up 50% of the bits, we must compress them better than 75% if we want to reach good compression ratios
- Is there a way to better compress the index data?
- We chose to concentrate on ETC1, since we co-developed it and hence know it well
- It is also available on Android from version 2.2 and up, so it is widely spread
- We need to explain how ETC1 works first

BASIC IDEA ETC1

- Use only 12 bits to specify a “base color” for a 2x4 block
- Modify the luminance for each pixel in the block

12-bit “base color”

+



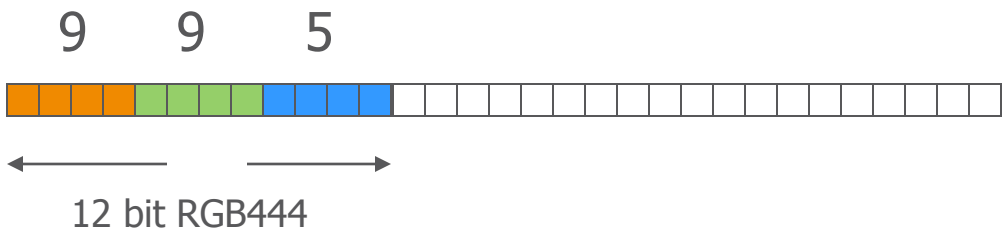
luminance modification

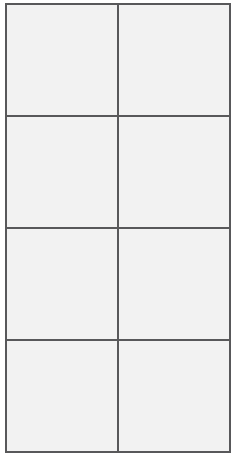
=

resulting image

BIT OUTLINE

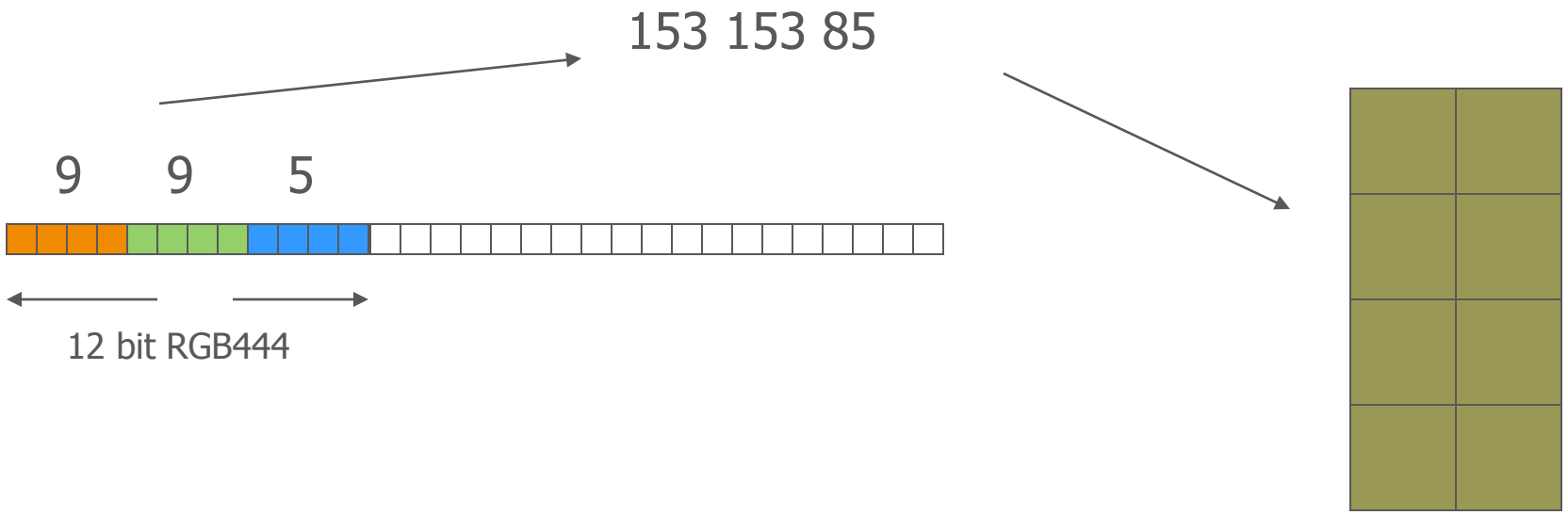
- Each 4x4 pixel block will be represented by 64 bits
- 15 bits will control each half-block (2 bits are mode bits)





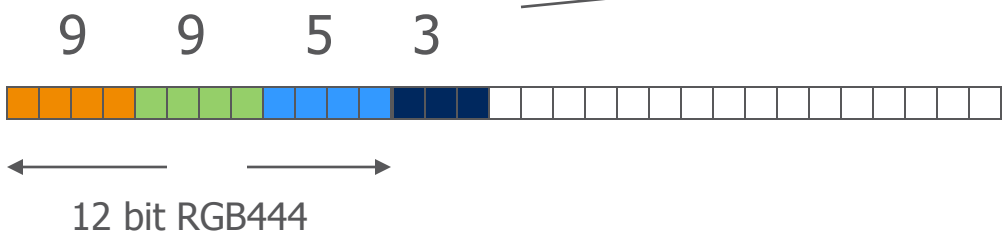
BIT OUTLINE

› First 12 bits is RGB444 which gives the base color for the entire block.



BIT OUTLINE

› Next 3 bits selects a table from a set of 8 tables



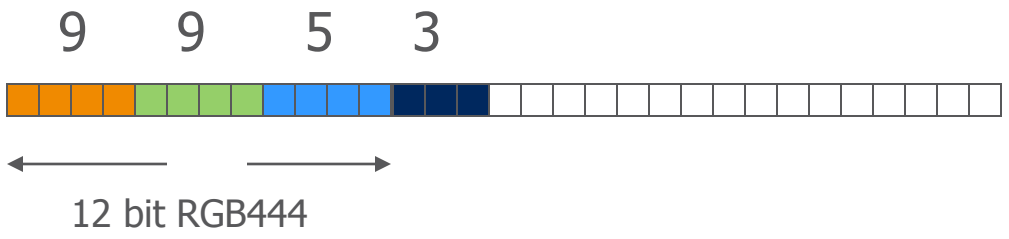
11	10	00	01
-8	-2	2	8
-17	-5	5	17
-29	-9	9	29
-42	-13	13	42
-60	-18	18	60
-80	-24	24	80
-106	-33	33	106
-183	-47	47	183

BIT OUTLINE

› Next 3 bits selects a table from a set of 8 tables.

11	10	00	01
-42	**-13**	**13**	**42**

11	10	00	01
-8	-2	2	8
-17	-5	5	17
-29	-9	9	29
-42	-13	13	42
-60	-18	18	60
-80	-24	24	80
-106	-33	33	106
-183	-47	47	183

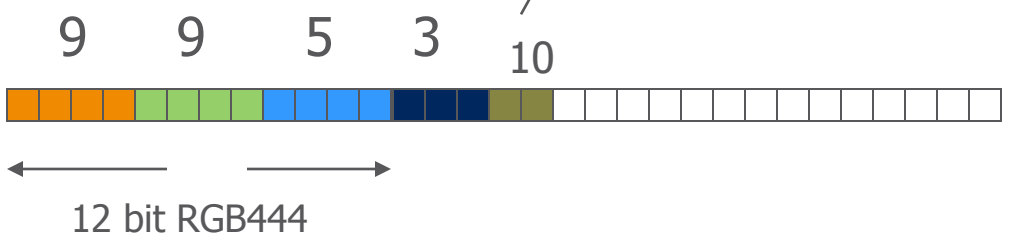


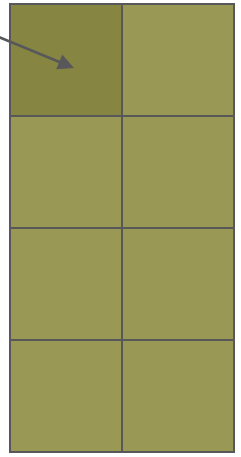
BIT OUTLINE

› The next 2 bits modifies the first pixel according to the table…

11	10	00	01
-42	**-13**	**13**	**42**

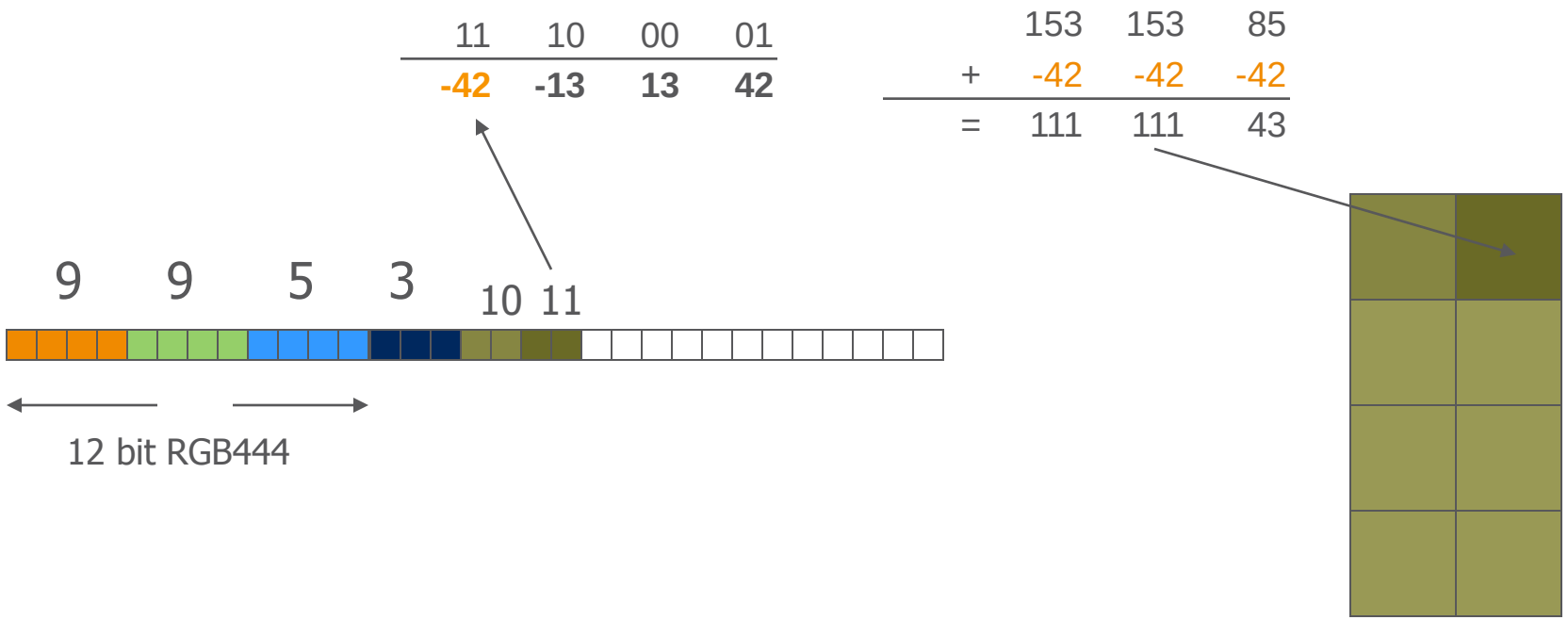
	153	153	85
+	-13	-13	-13
=	140	140	72





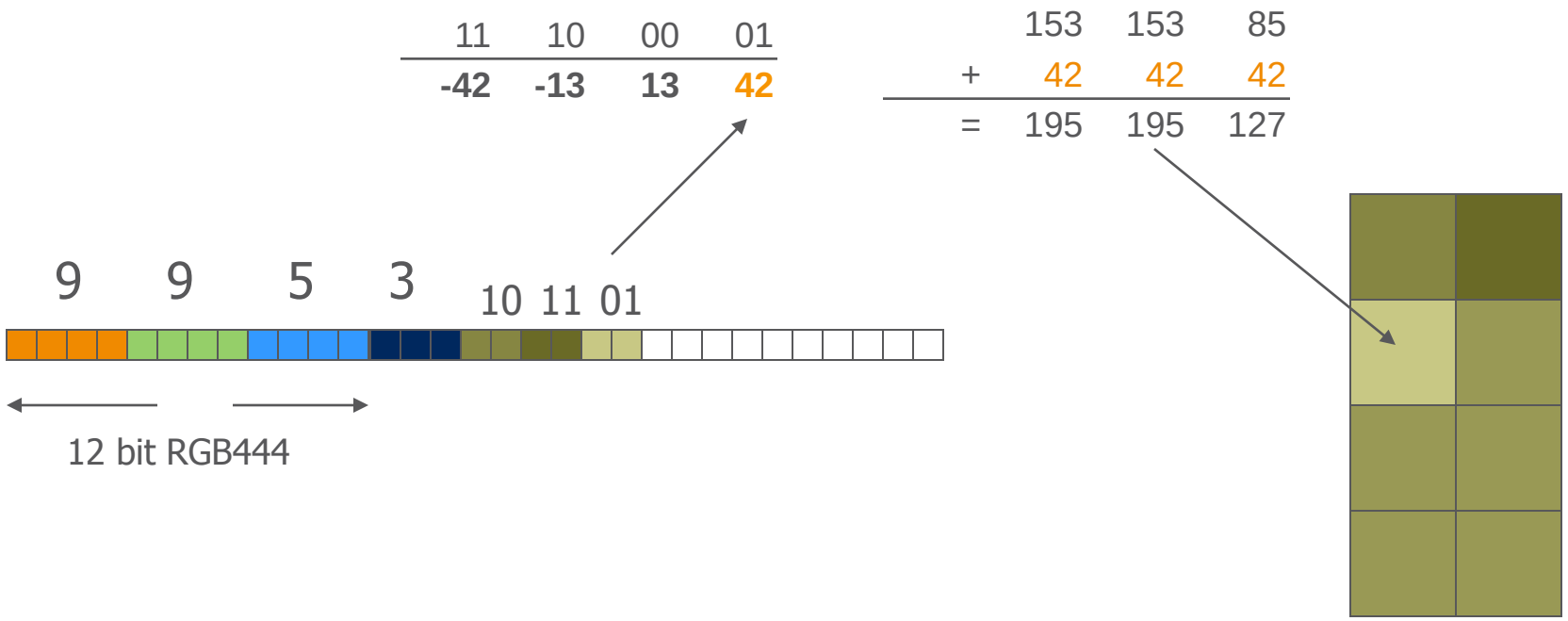
BIT OUTLINE

› The next 2 bits modifies the first pixel according to the table… and so on.



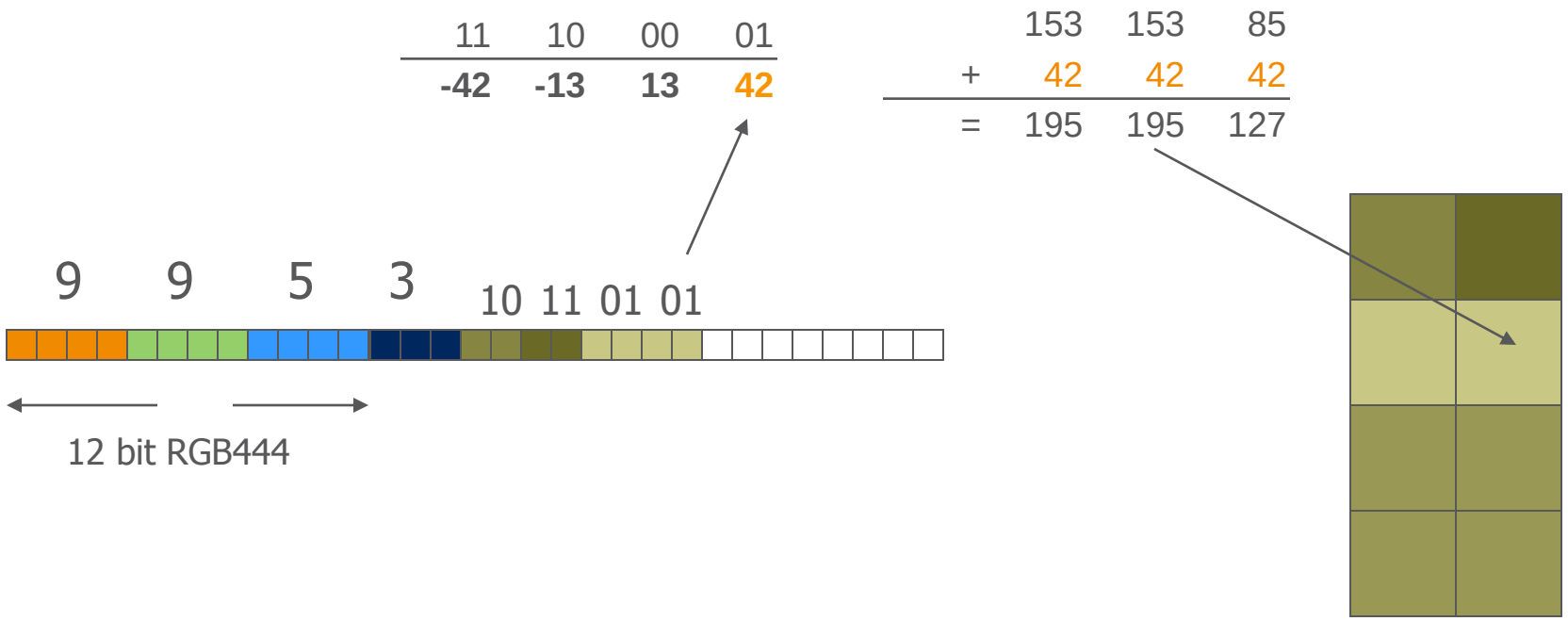
BIT OUTLINE

› The next 2 bits modifies the first pixel according to the table… and so on.



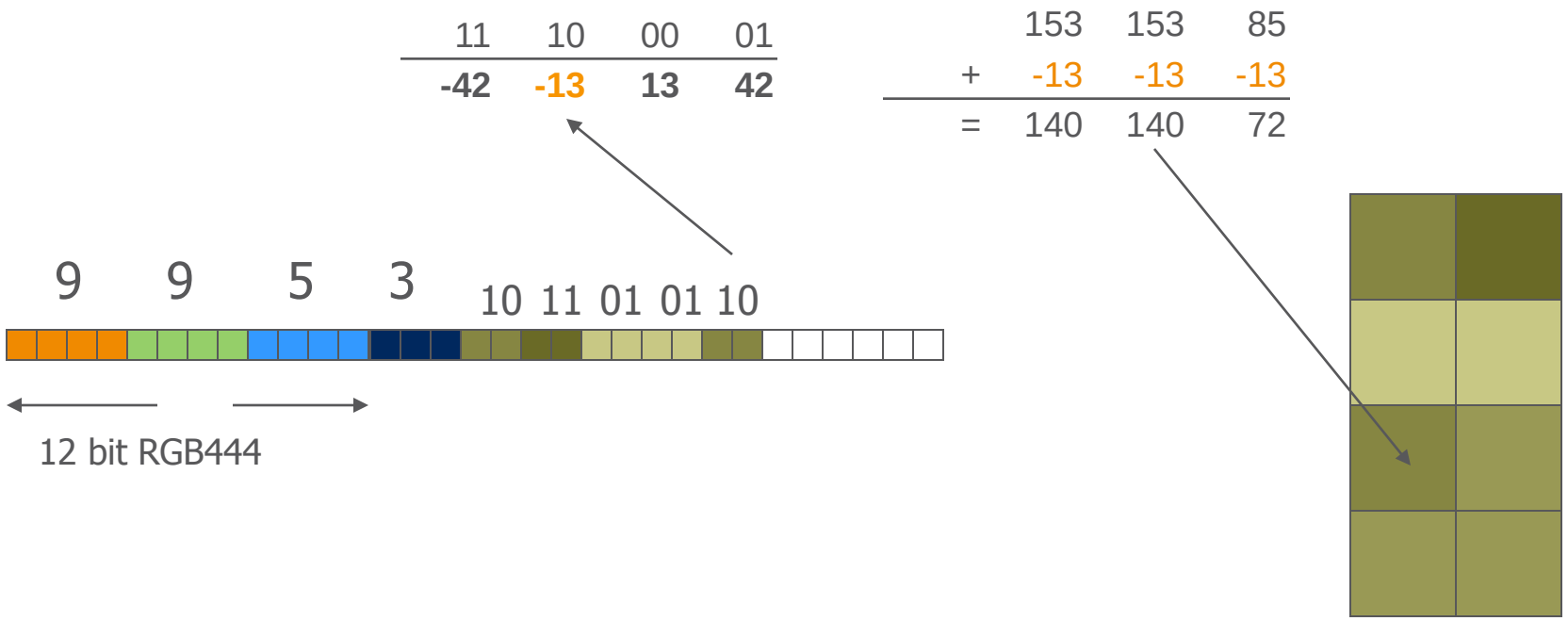
BIT OUTLINE

› The next 2 bits modifies the first pixel according to the table… and so on.



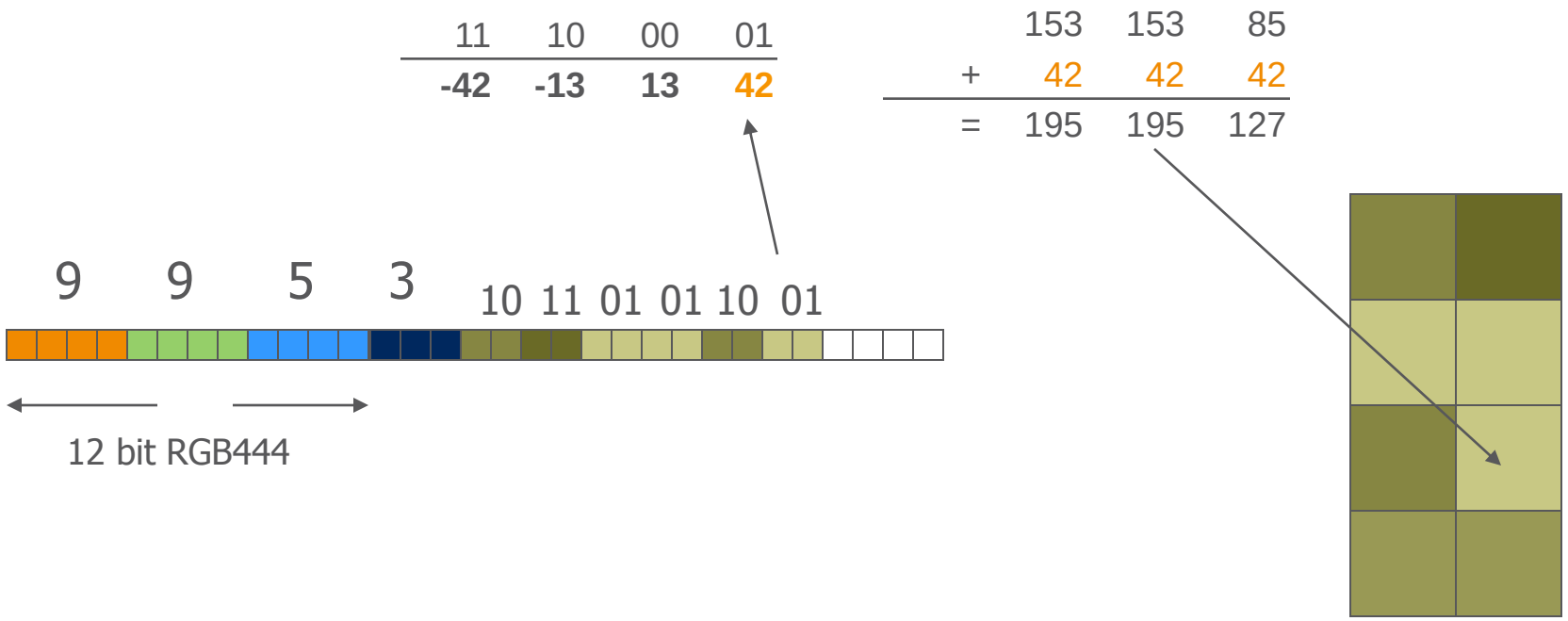
BIT OUTLINE

› The next 2 bits modifies the first pixel according to the table… and so on.



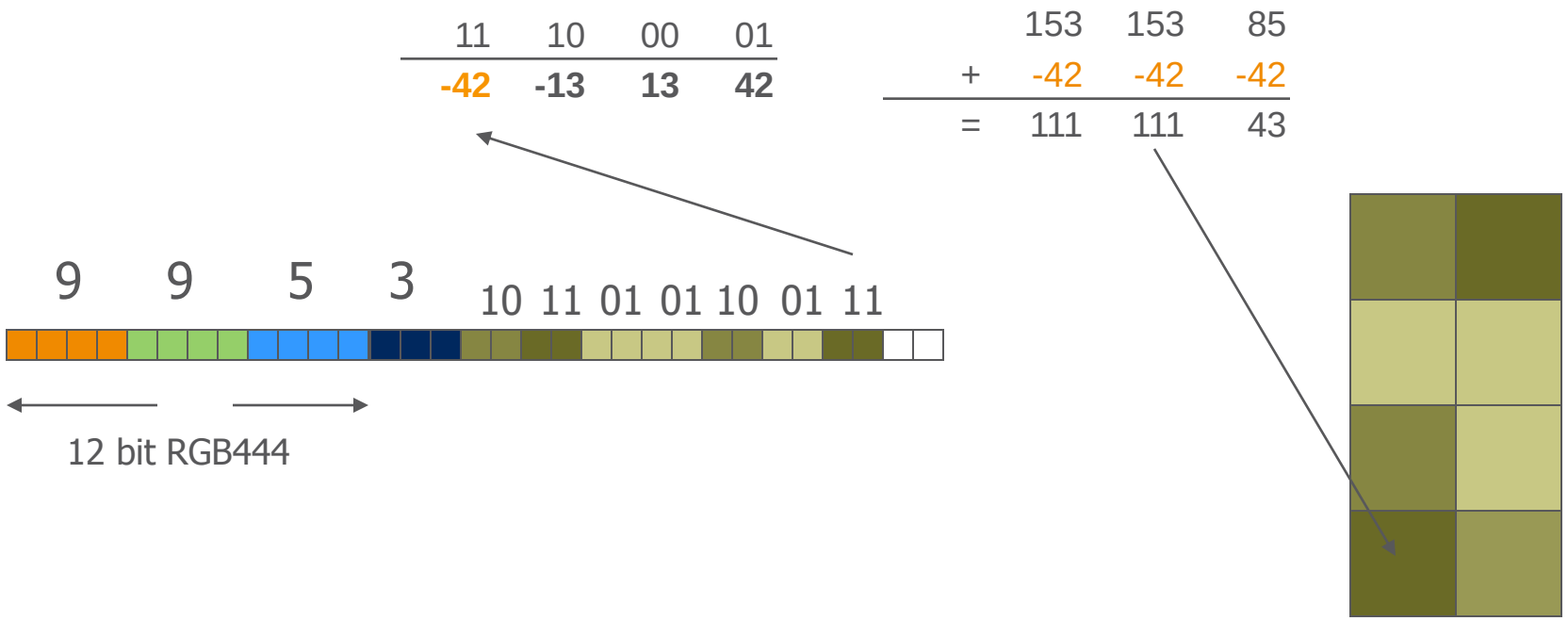
BIT OUTLINE

› The next 2 bits modifies the first pixel according to the table… and so on.



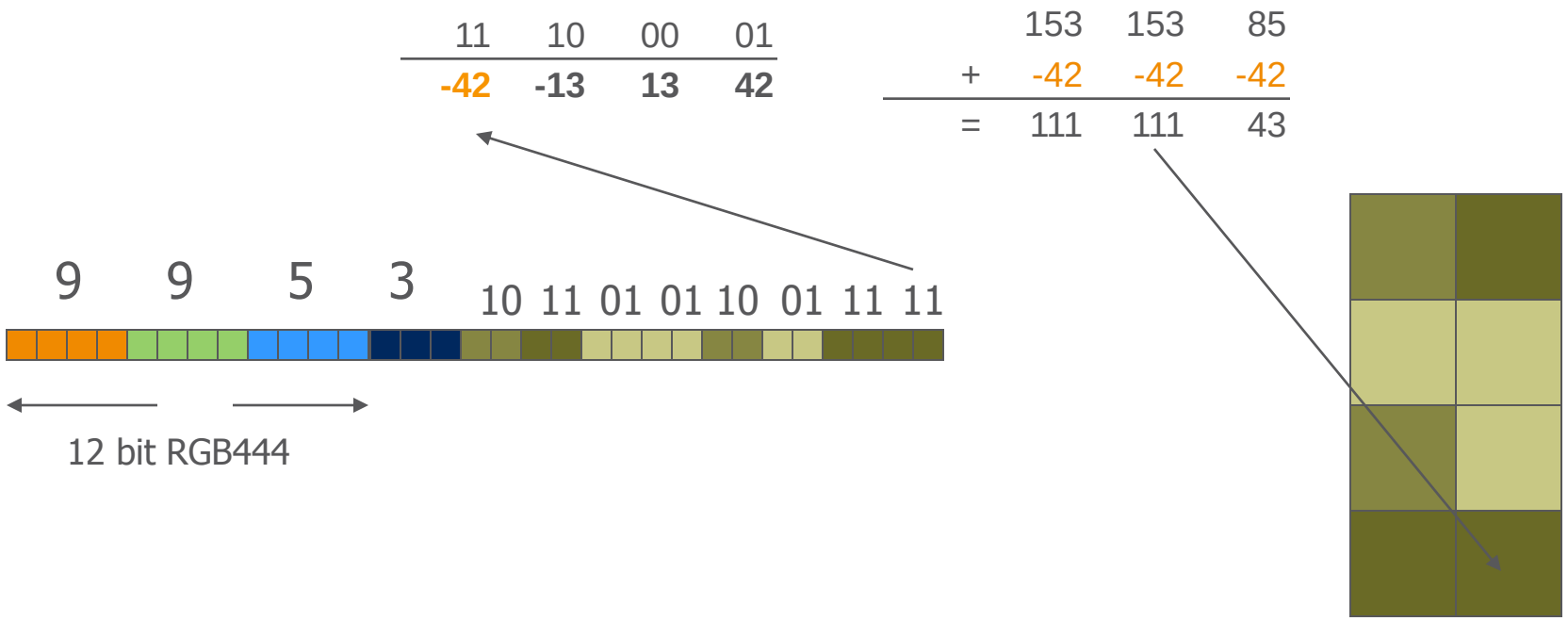
BIT OUTLINE

› The next 2 bits modifies the first pixel according to the table… and so on.



BIT OUTLINE

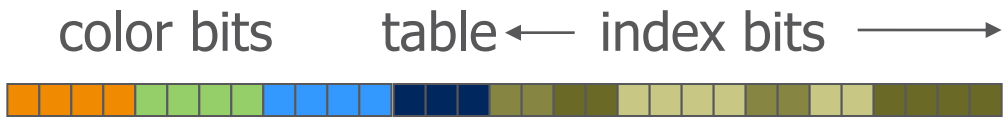
› The next 2 bits modifies the first pixel according to the table… and so on.



ETC1 WALKTHROUGH

› The other half-block is done similarly

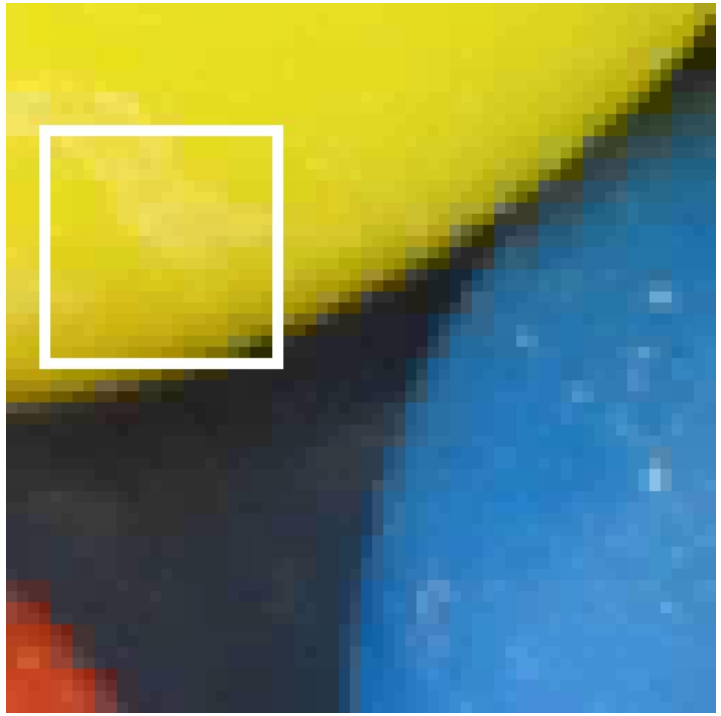
› We see that, as for S3TC, half of the bits are indices:



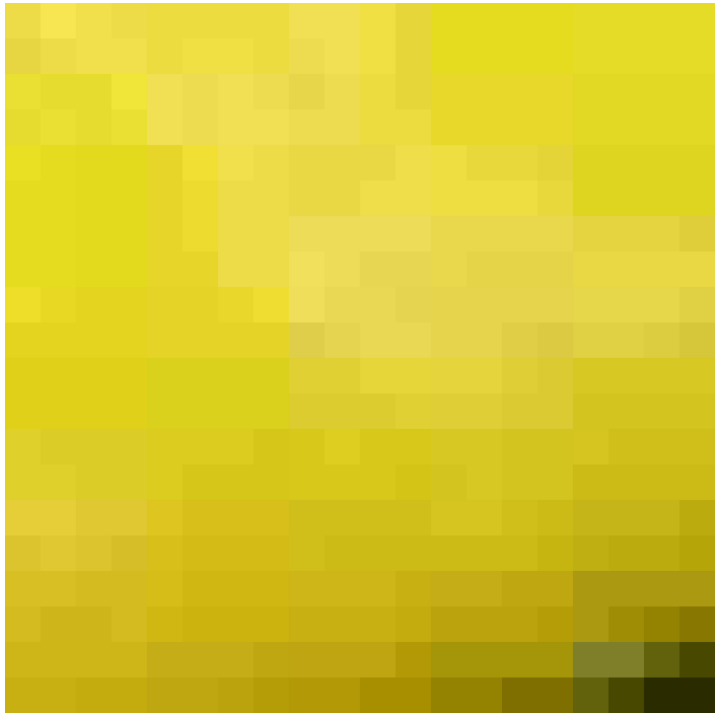
› In order to succeed we must compress these well.

INDEX DATA HARD

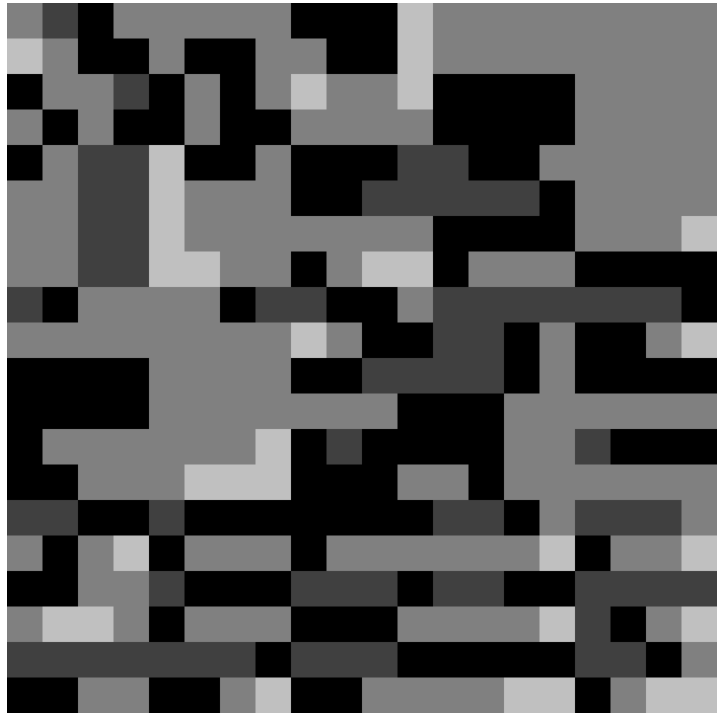
› Unfortunately, even in smooth areas, the index data looks very random and hard to compress



Decoded ETC1



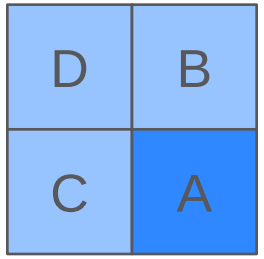
Zoomin of decoded image



Zoomin of index data

PREDICTION

- Many image compression techniques work by looking at the neighboring pixels. (PNG, H.264 among others)

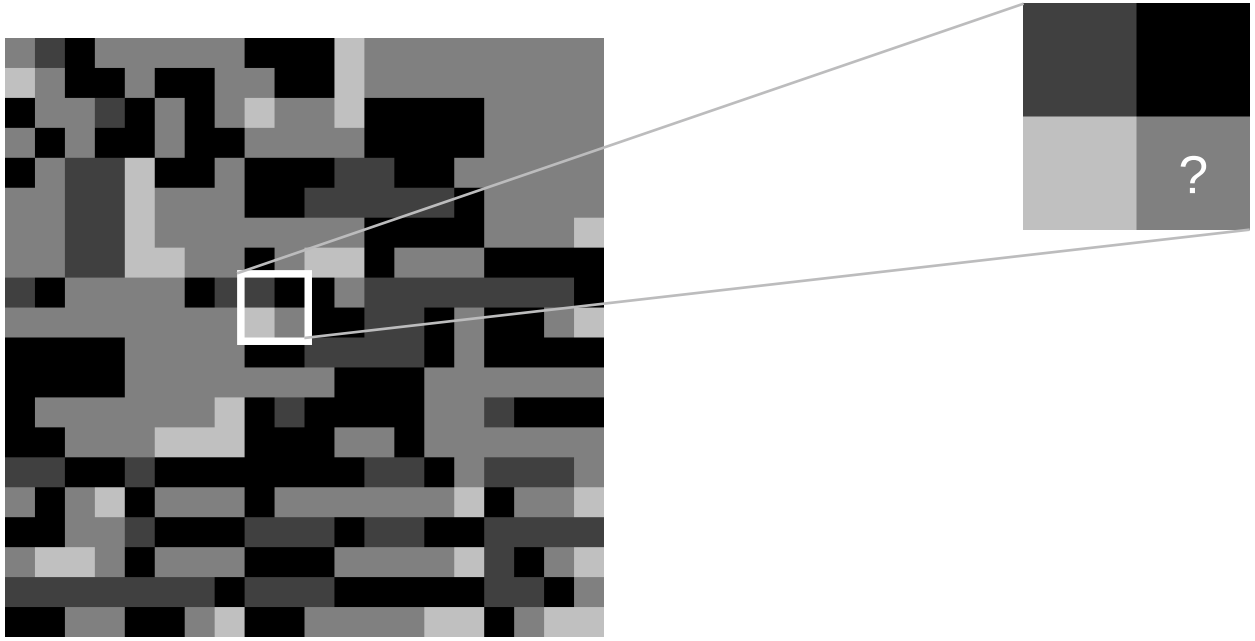


  - Instead of storing value A, a prediction is used:
  - $\hat{A}$ = prediction(B, C, D)
  - Then $\Delta A = A - \hat{A}$ is stored instead of A.

- The hope is that ΔA is easier to store than A, which is true if the prediction is good.

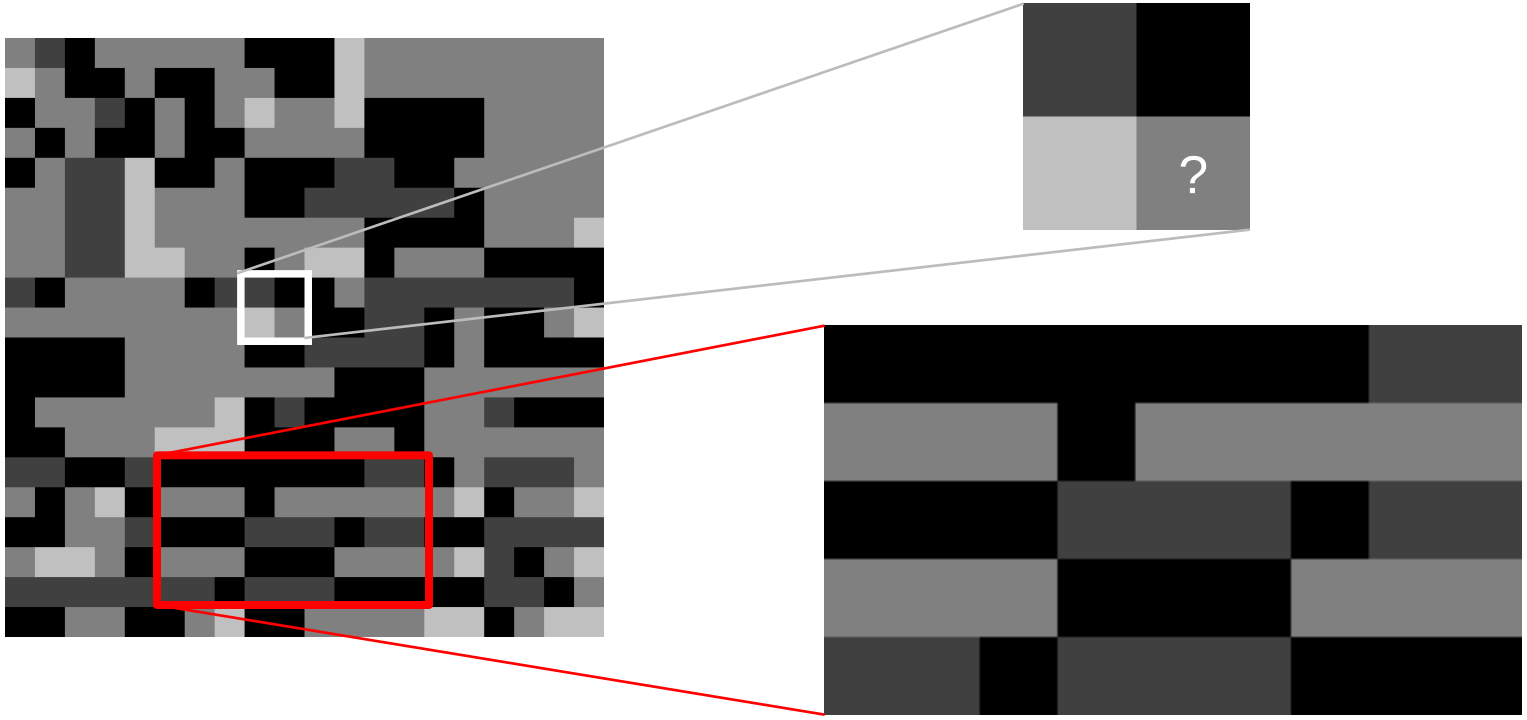
HOW TO PREDICT INDICES

› Guessing indices from surrounding indices not so easy



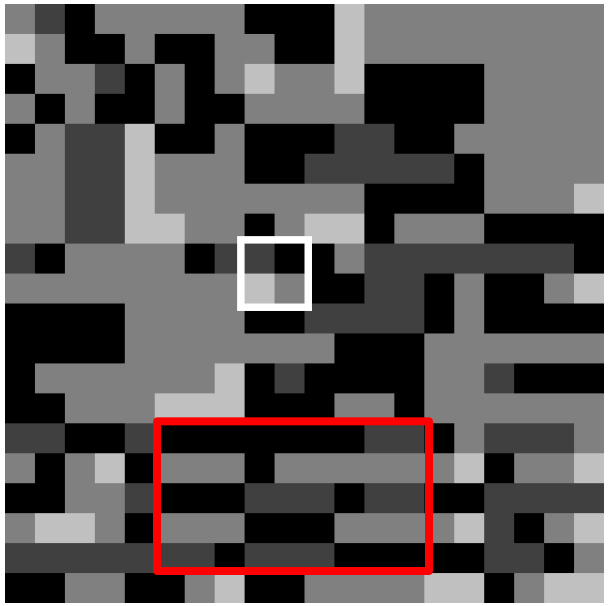
HOW TO PREDICT INDICES

- Guessing from surrounding pixels not so easy
- Some sort of structure seems to be there

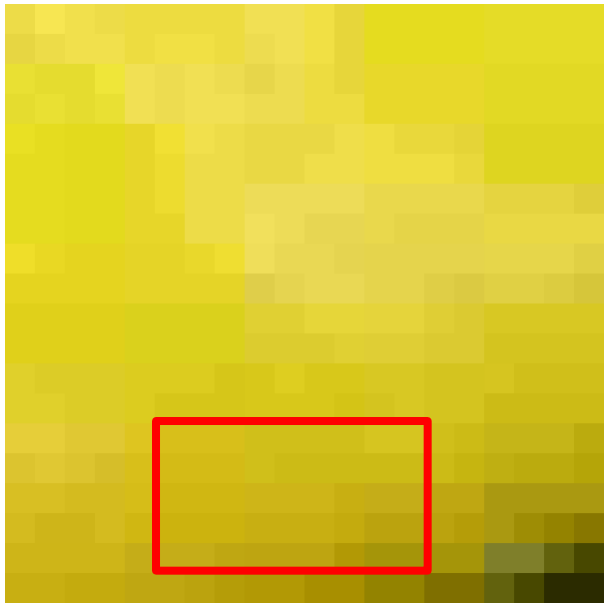


HOW TO PREDICT INDICES

- Guessing from surrounding pixels not so easy
- Some sort of structure seems to be there in gradient areas.



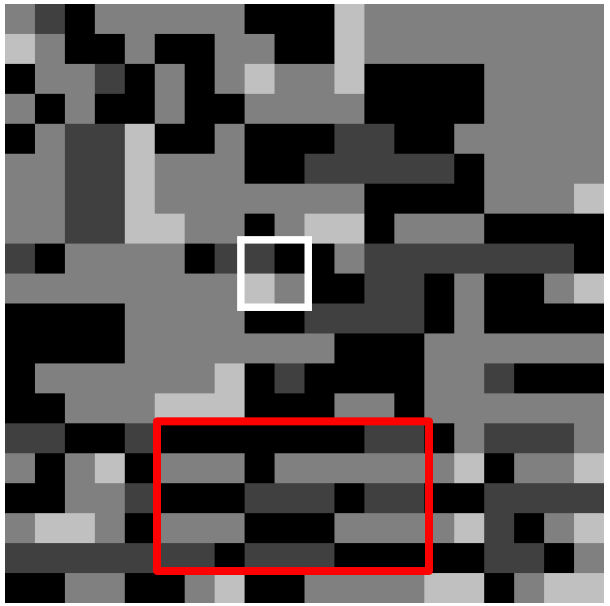
hard to predict



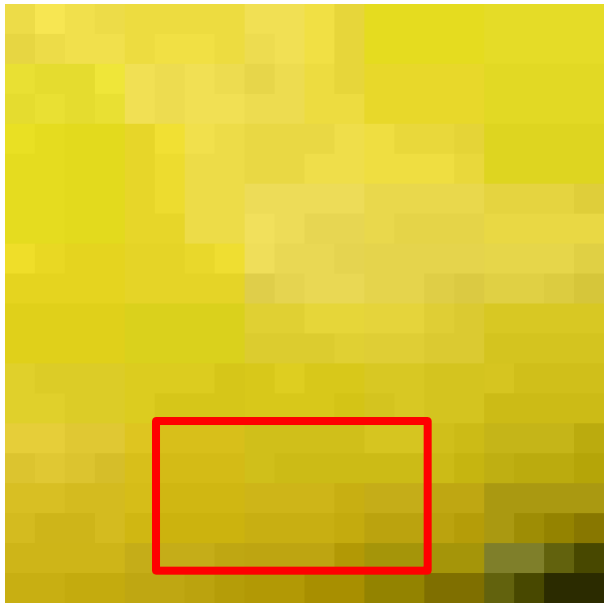
essy to predict

HOW TO PREDICT INDICES

- Guessing from surrounding pixels not so easy
- Some sort of structure seems to be there in gradient areas.
- Why not predict the color instead of the index?



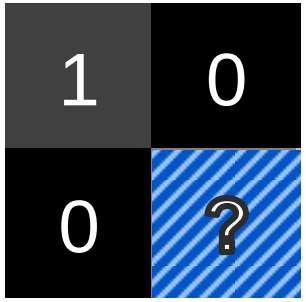
hard to predict



essy to predict

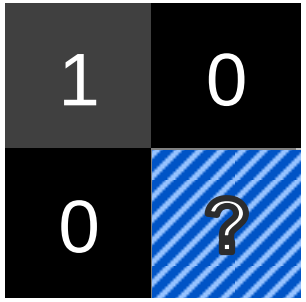
EXAMPLE

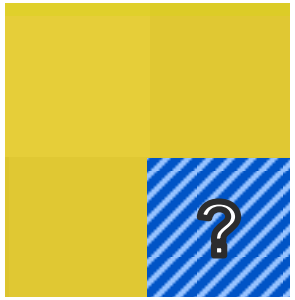
› To guess the index...



EXAMPLE

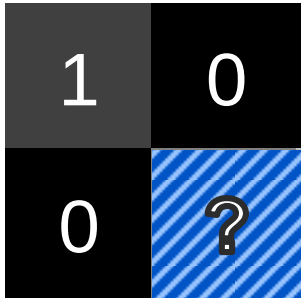
- To guess the index...
- ...look at the surrounding colors

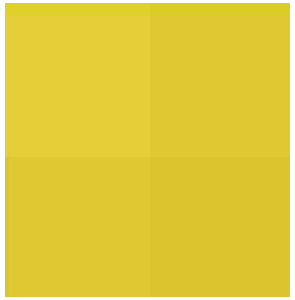




EXAMPLE

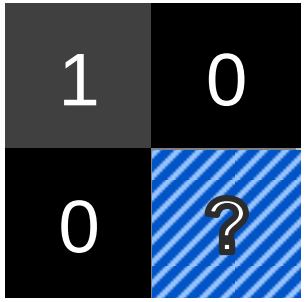
- To guess the index...
- ...look at the surrounding colors
- ... guess the color of the pixel

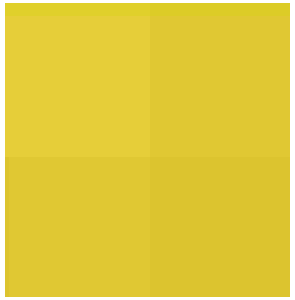




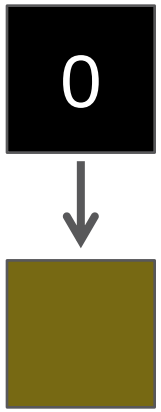
EXAMPLE

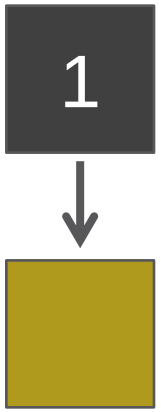
- To guess the index...
- ...look at the surrounding colors
- ... guess the color of the pixel

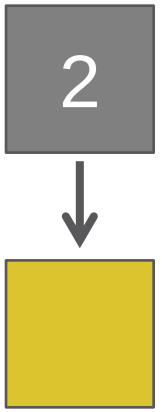


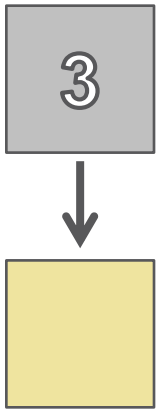


- Now calculate which colors each index would give...



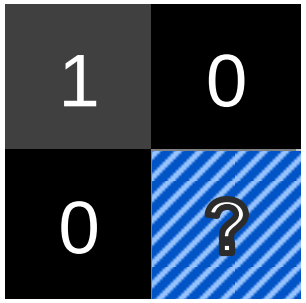


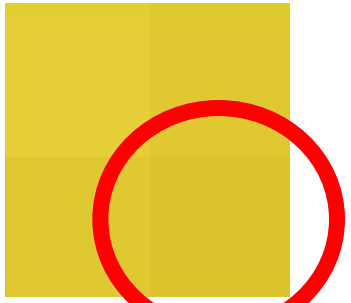




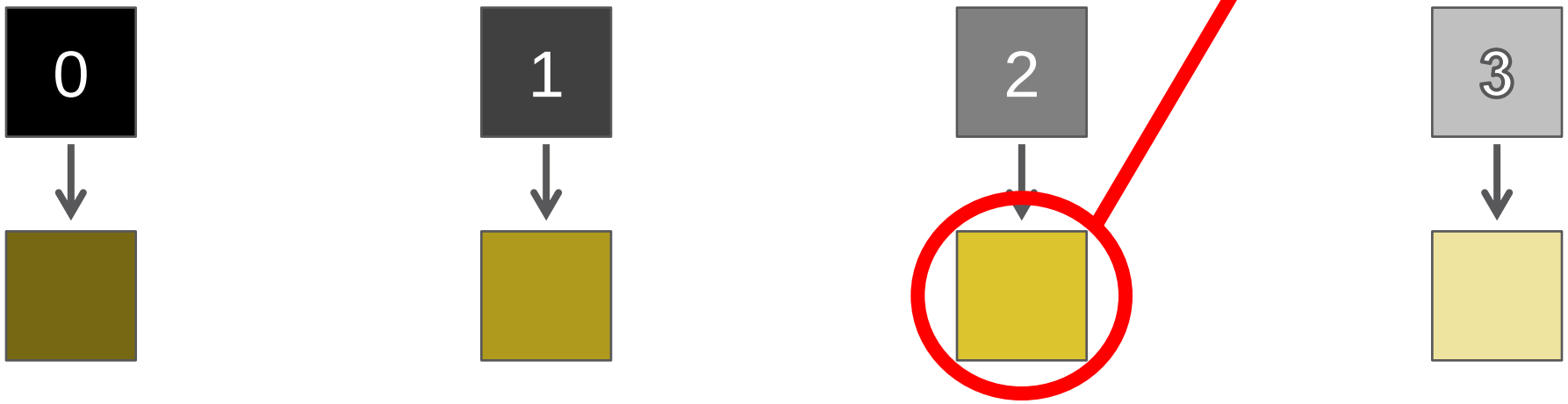
EXAMPLE

- To guess the index...
- ...look at the surrounding colors
- ... guess the color of the pixel



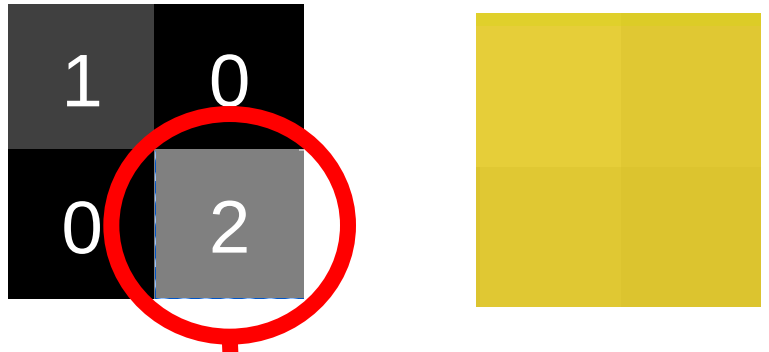


- Now calculate which colors each index would give...
- ... and pick the closest!

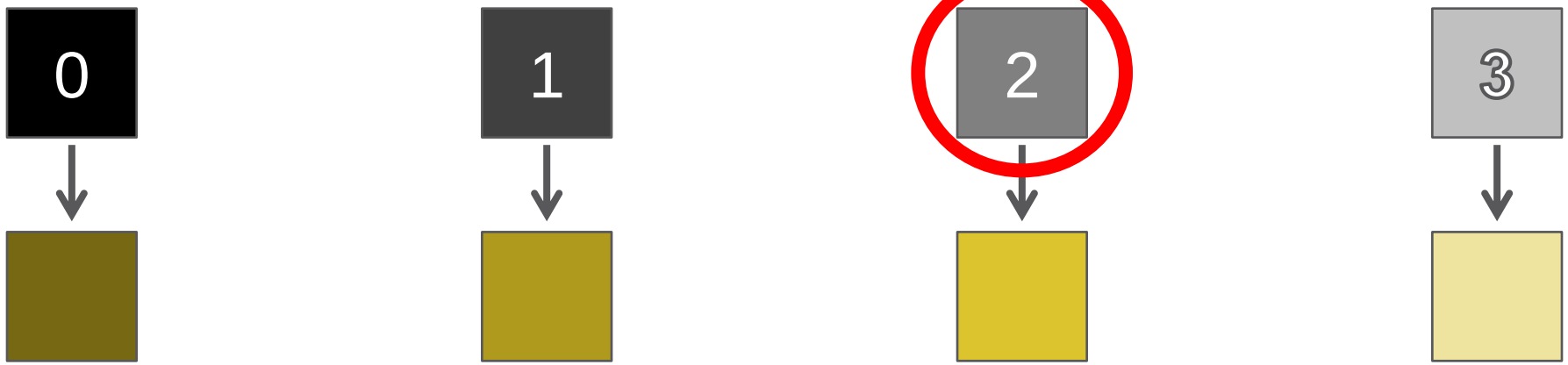


EXAMPLE

- To guess the index...
- ...look at the surrounding colors
- ... guess the color of the pixel



- Now calculate which colors each index would give...
- ... and pick the closest!



closest match

EXAMPLE

- Base color is (240, 130, 0)
- Predicted color is (249, 150, 25)
- Table = {-60, -18, 18, 60}

pixel index	modifier	color	$\|\|\text{color} - \text{prediction}\|\|^2$
0	-60	(180, 70, 0)	11786
1	-18	(222, 112, 0)	2798
2	18	(255, 148, 18)	89
3	60	(255, 190, 60)	2861

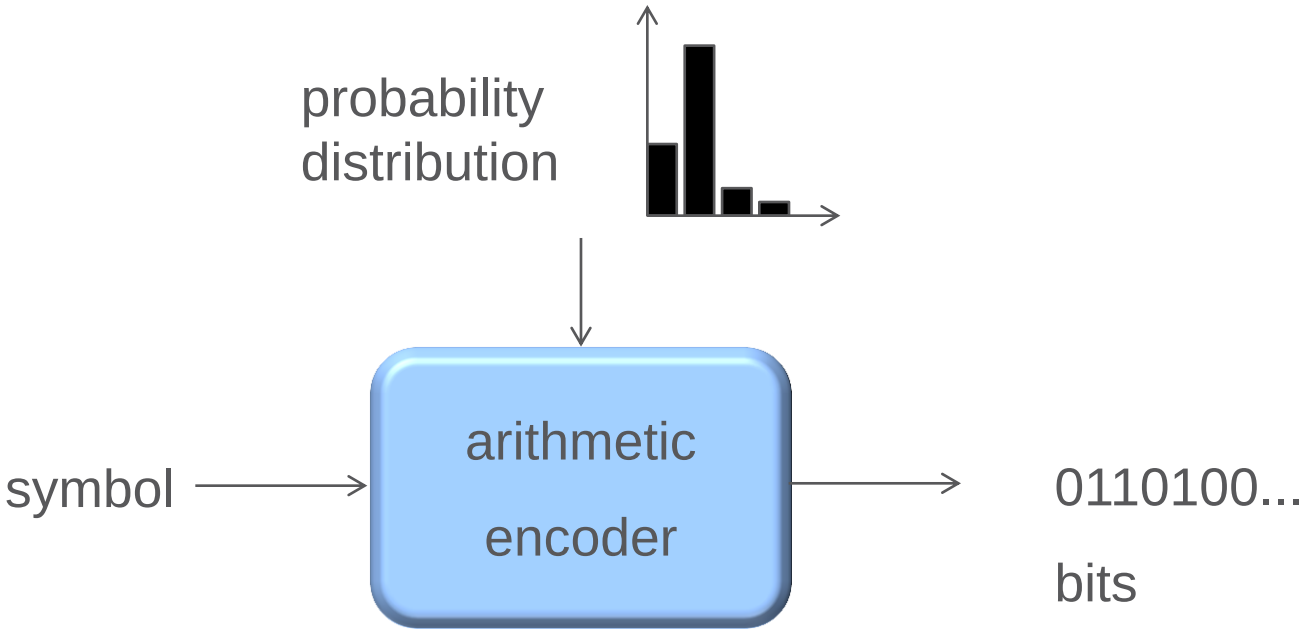
EXAMPLE

- Base color is (240, 130, 0)
- Predicted color is (249, 150, 25)
- Table = {-8, -2, 2, 8}

pixel index	modifier	color	$\|\|\text{color} - \text{prediction}\|\|^2$
0	-8	(232, 122, 0)	1698
1	-2	(238, 128, 0)	1230
2	2	(242, 132, 2)	902
3	8	(248, 138, 8)	434

ENCODING USING THE PREDICTION

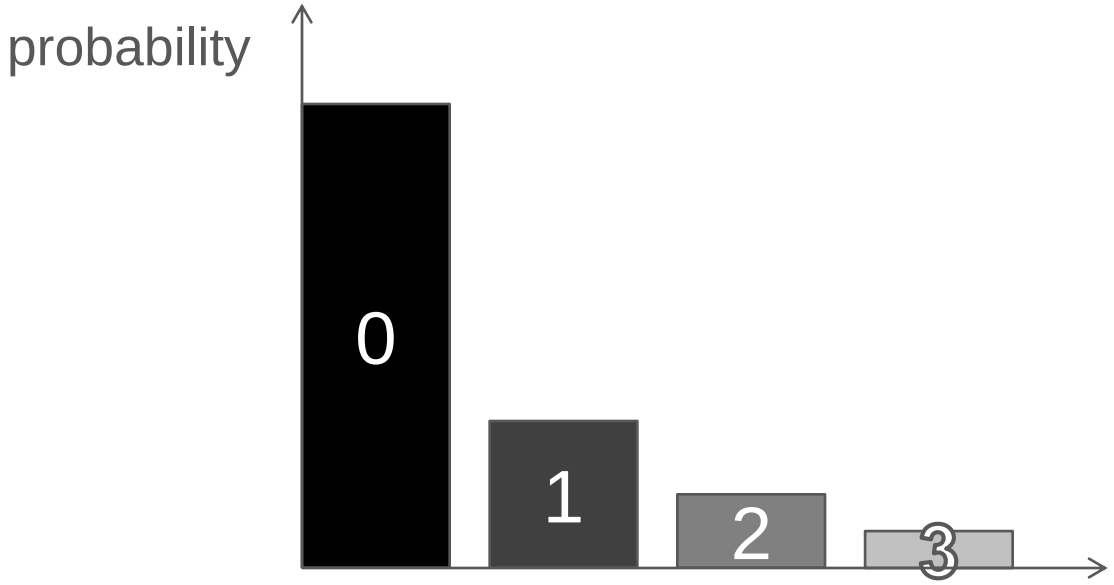
- We use an arithmetic coding.
- Given a probability distribution, an arithmetic coder can encode symbols to bits
- We use the arithmetic encoder as a black box:



ENCODING

› Depending on the predicted index, the probability distribution changes:

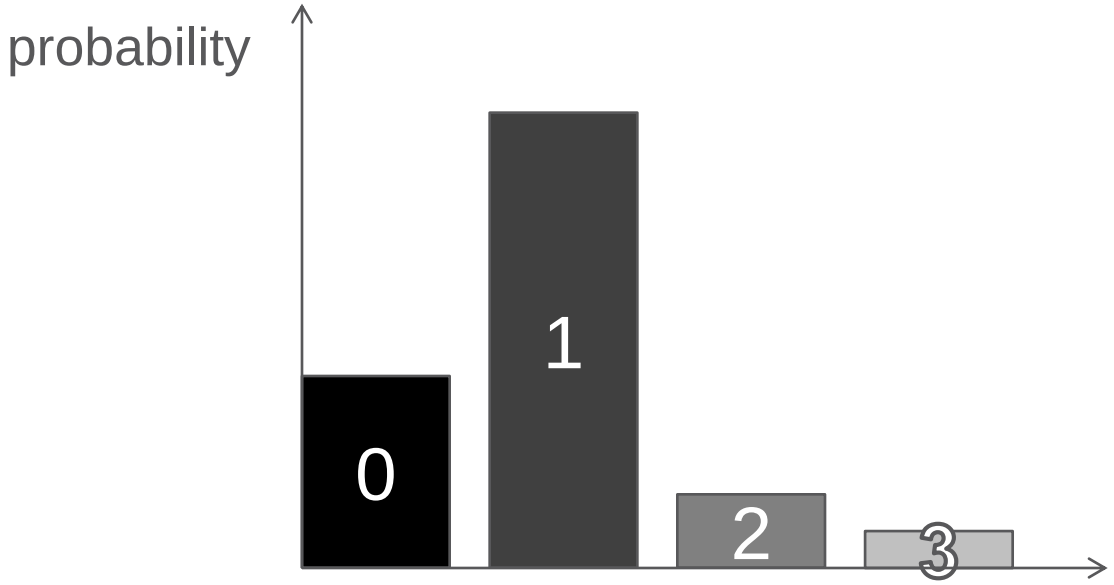
› If prediction is **0** the probability distribution may be



ENCODING

› Depending on the predicted index, the probability distribution changes:

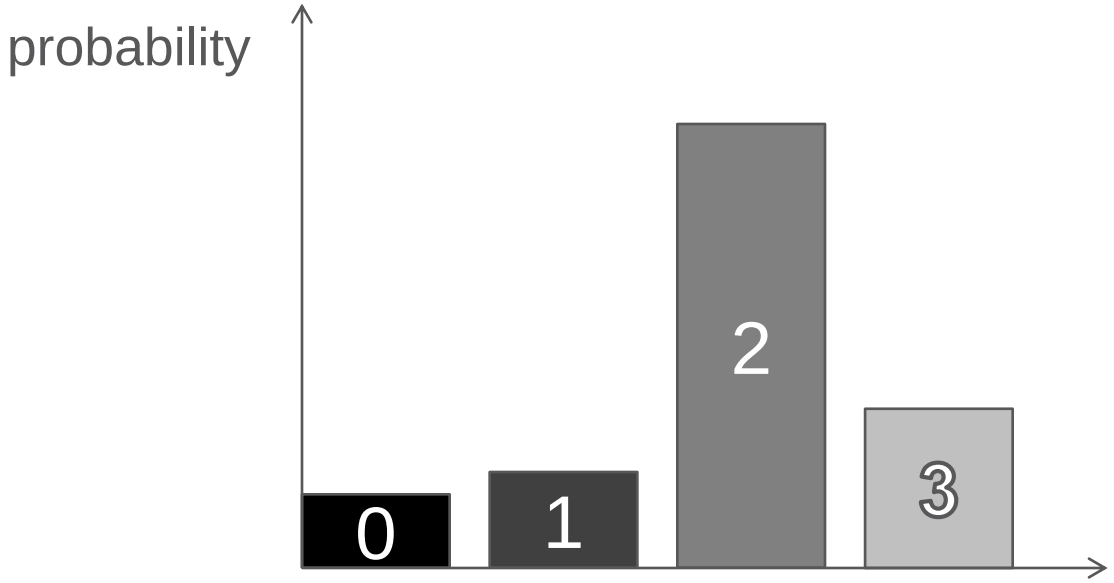
› If prediction is 1 the probability distribution may be



ENCODING

› Depending on the predicted index, the probability distribution changes:

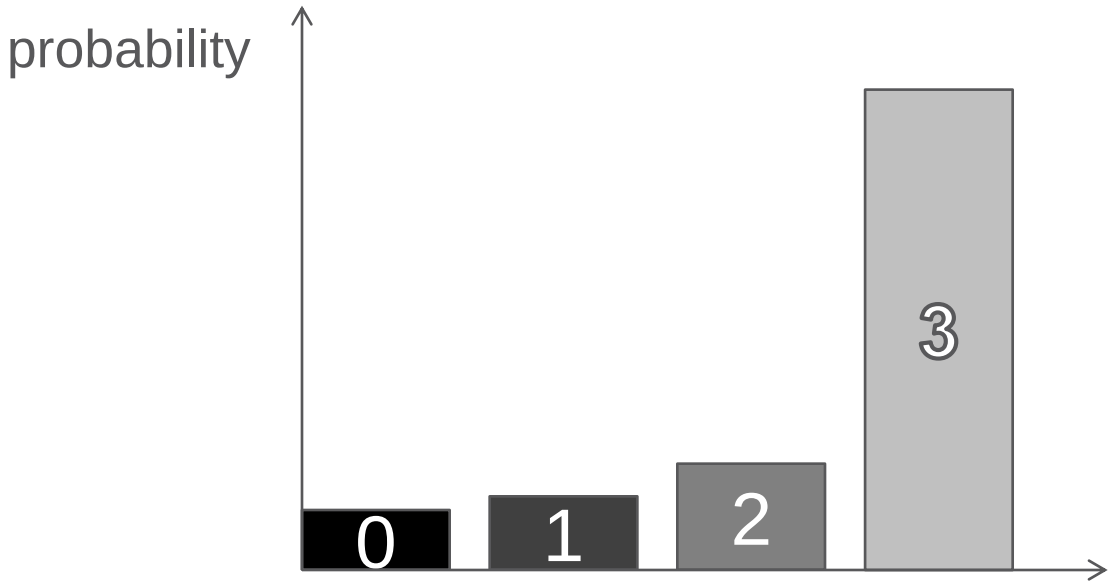
› If prediction is 2 the probability distribution may be



ENCODING

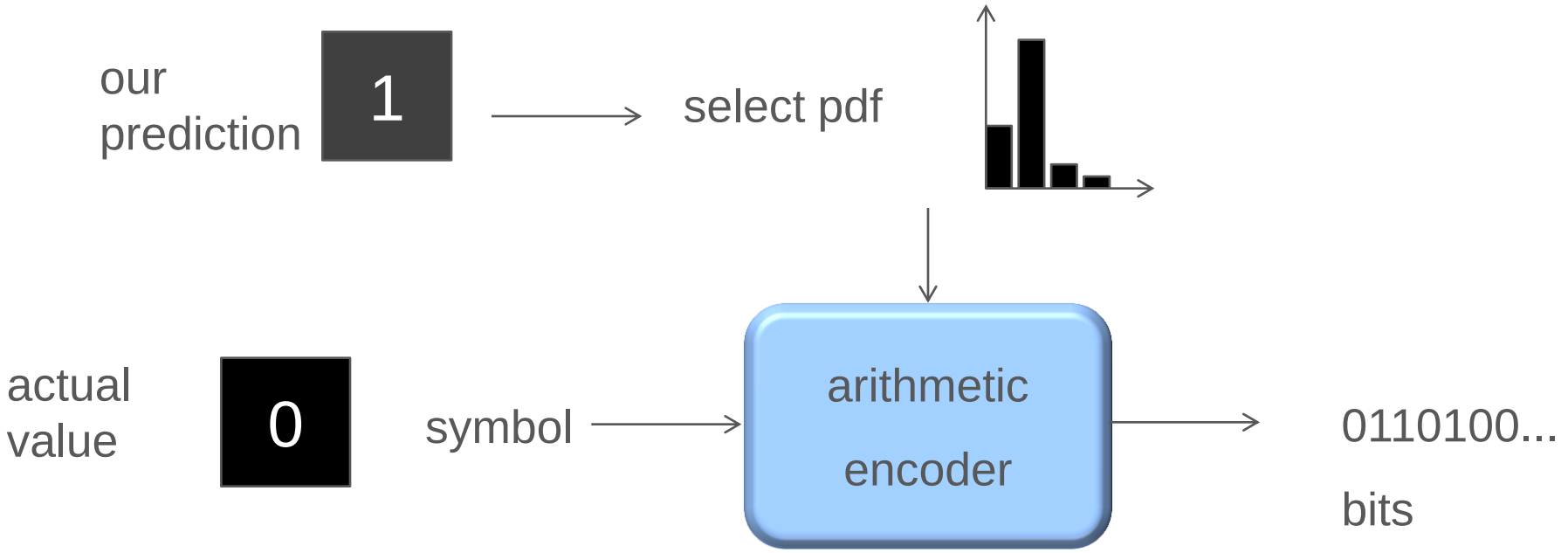
› Depending on the predicted index, the probability distribution changes:

› If prediction is 3 the probability distribution may be



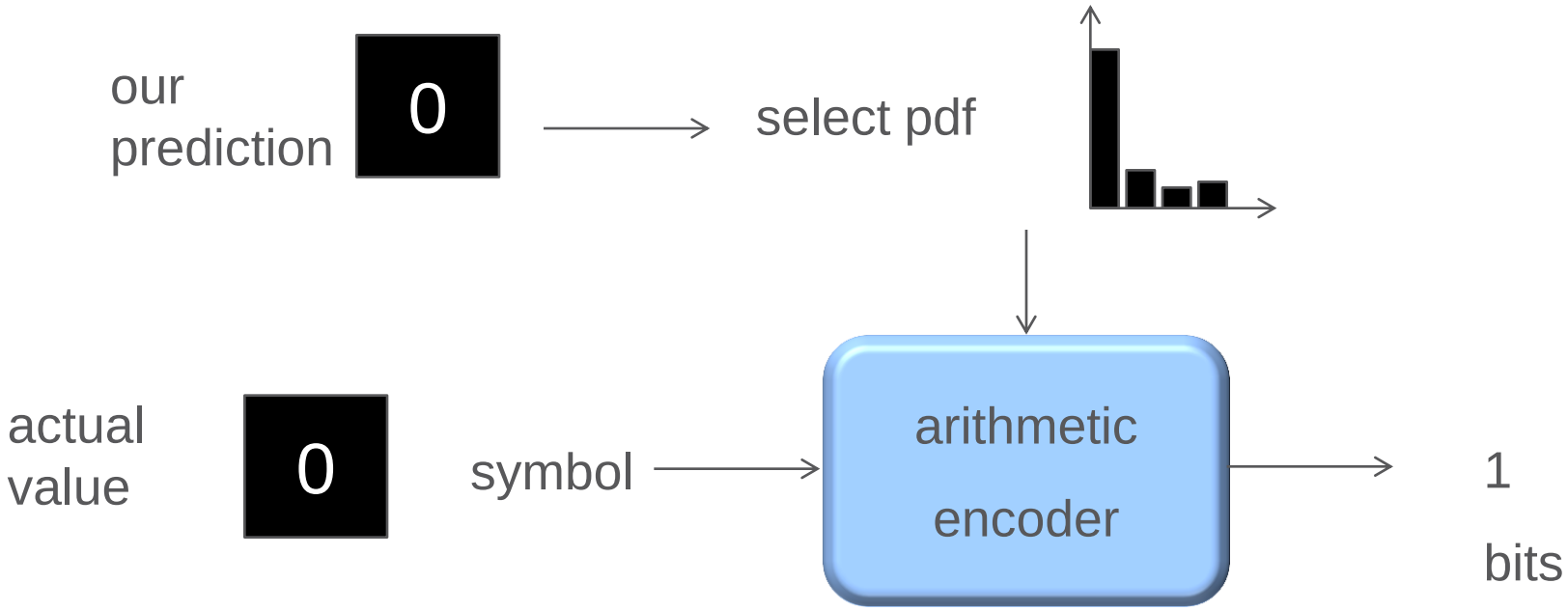
ENCODING USING THE PREDICTION

- We use an arithmetic coding.
- Given a probability distribution, an arithmetic coder can encode symbols to bits
- We use the arithmetic encoder as a black box:



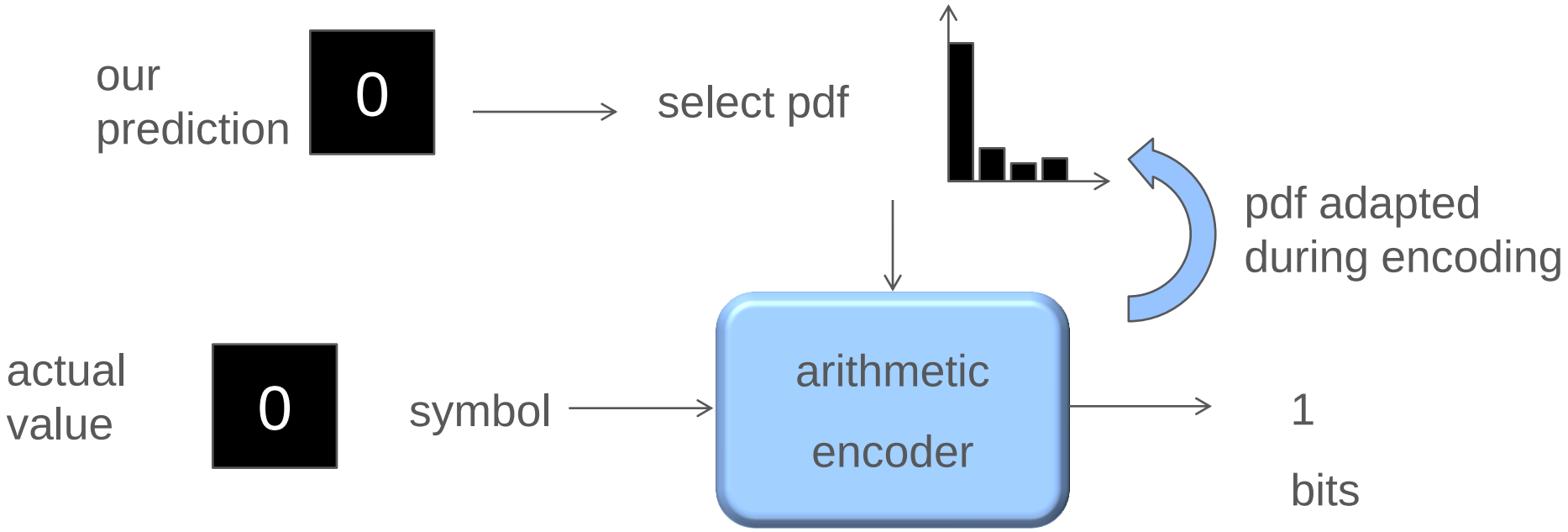
ENCODING USING THE PREDICTION

- We use an arithmetic coding.
- Given a probability distribution, an arithmetic coder can encode symbols to bits
- We use the arithmetic encoder as a black box:



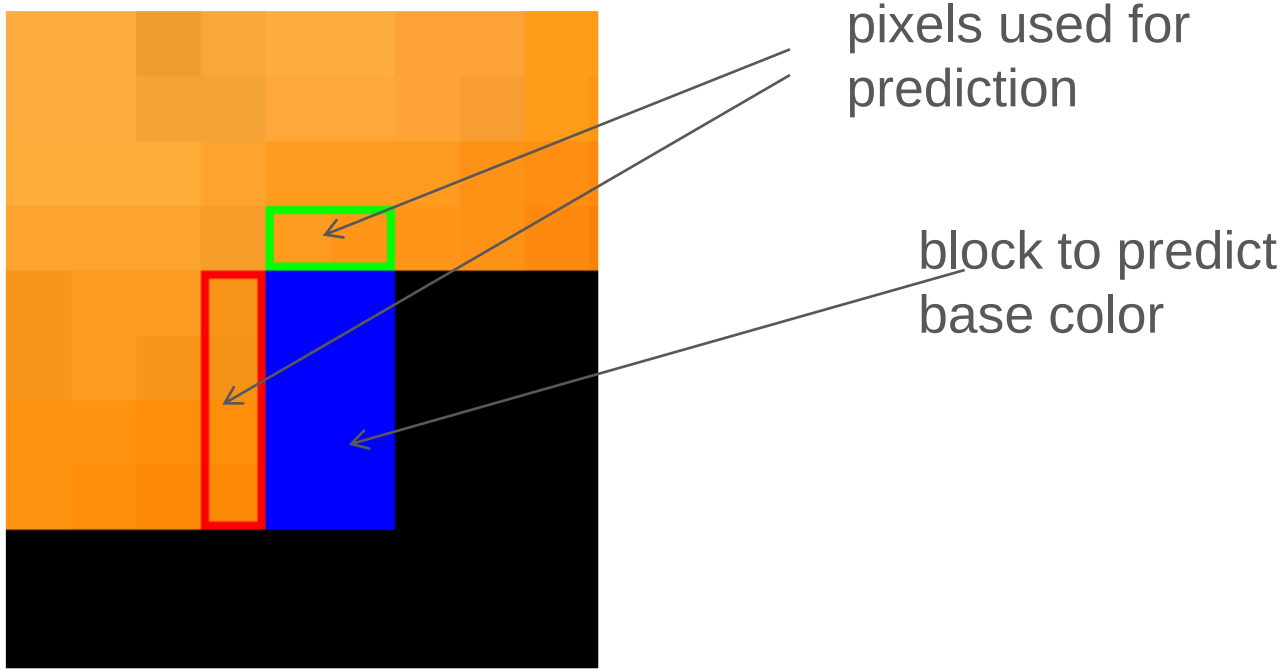
ENCODING USING THE PREDICTION

- We use an arithmetic coding.
- Given a probability distribution, an arithmetic coder can encode symbols to bits
- We use the arithmetic encoder as a black box:



OTHER PARAMETERS

- The base color is compressed in a similar way: It is predicted from surrounding pixels (not surrounding base colors), followed by arithmetic coding.
- For details about the other parameters, see the paper.



RESULTS

- We have tried our method on a set of 64 images containing game textures, natural images and synthetic images.
- Some of the images had a white background, so we report results also for the case when the white images were excluded.

	ETC1 +ZIP
all images (bpp)	2.85
nonwhite (bpp)	3.01

RESULTS

- We have tried our method on a set of 64 images containing game textures, natural images and synthetic images.
- Some of the images had a white background, so we report results also for the case when the white images were excluded.

	ETC1 +ZIP	ETC1 +LZMA
all images (bpp)	2.85	2.60
nonwhite (bpp)	3.01	2.76

RESULTS

- We have tried our method on a set of 64 images containing game textures, natural images and synthetic images.
- Some of the images had a white background, so we report results also for the case when the white images were excluded.

	ETC1 +ZIP	ETC1 +LZMA	ETC1 +proposed
all images (bpp)	2.85	2.60	2.21
nonwhite (bpp)	3.01	2.76	2.31

RESULTS

- We have tried our method on a set of 64 images containing game textures, natural images and synthetic images.
- Some of the images had a white background, so we report results also for the case when the white images were excluded.

	ETC1 +ZIP	ETC1 +LZMA	ETC1 +proposed	JPEG of equal quality
all images (bpp)	2.85	2.60	2.21	2.25
nonwhite (bpp)	3.01	2.76	2.31	2.35

RESULTS

- We have tried our method on a set of 64 images containing game textures, natural images and synthetic images.
- Some of the images had a white background, so we report results also for the case when the white images were excluded.
- Index data compressed down to 60% - 65% of original

	ETC1 +ZIP	ETC1 +LZMA	ETC1 +proposed	JPEG of equal quality
all images (bpp)	2.85	2.60	2.21	2.25
nonwhite (bpp)	3.01	2.76	2.31	2.35

LIMITATIONS

- Compared to texture streaming [van Waveren 2006]
  - We are on par with JPEG in the upper range (around 2.4 bpp).
  - JPEG can save more bits by increasing distortion (to around 1.2 bpp) – but our method is lossless and can therefore not trade distortion against bit rate.
  - Decoding time is around 300 ms for a 512x512 texture
  - But: Provides better quality since slow off-line texture compression can be used and double artifacts are avoided.
- Compared to low-bit rate texture compression
  - Our proposal does not save memory footprint during rendering
  - But: Can be combined with low-bit rate texture compression

FUTURE WORK

- Try method on S3TC, PVR-TC, ETC2 etc.
- Lower bit rate further by introducing errors. Can be done in a rate/distortion fashion (remove the bit that makes the least distortion).
- Make method more GPU-friendly so that the GPU can do the unpacking

THANKS

› Thanks to Andrew Norkin for valuable discussions on alternative binarizations.

› Thank you for listening!

ERICSSON