Lesson learnt from WebP.

What's next?

Pascal Massimino
skal@google.com

ALLIANCE FOR OPEN MEDIA RESEARCH

Symposium 2019

Plan

- lessons learnt from VP8 -> WebP codec
- research direction and experiments for “WebP v2”
- results (+demo?)

Motivation

WebP, HEIF, AVIF ...

Motivation

WebP, HEIF, AVIF …

most recent **Image** codecs originate from **Video** codec.

Motivation

WebP, HEIF, AVIF …

most recent **Image** codecs originate from **Video** codec.

Is it a always a good choice?

Lessons learnt from VP8 -> WebP

Lessons learnt from VP8 -> WebP

Two main use-cases for image compression:

- “Capture” [device -> storage / CDN]

Lessons learnt from VP8 -> WebP

Two main use-cases for image compression:

- “Capture” [device -> storage / CDN]
- “Web consumption” [CDN -> mobile device]

Lessons learnt from VP8 -> WebP

Two main use-cases for image compression:

- “Capture” [device -> storage / CDN]
- “Web consumption” [CDN -> mobile device]

“WebP”

Web image format

important peculiarities

Web image format important peculiarities

- incremental decoding
- memory consumption
- small format overhead
- interleaved chunk data for early display
- efficient lossy/lossless transparency
- efficient lossless coding
- preview
- light ‘animation’ format (!= video)
- efficient in software, more than hardware

Web image format important peculiarities

- incremental decoding
- memory consumption
- small format overhead
- interleaved chunk data for early display
- efficient lossy/lossless transparency
- efficient lossless coding
- preview
- light ‘animation’ format (!= video)
- efficient in software, more than hardware

WEBP v2 !!

WebP v2: experimentations

Goal:

v2 = like v1 …

“Web-consumption”, not “Capture”.

WebP v2: experimentations

Goal:

v2 = like v1 …

… but ‘more’.

“Web-consumption”, not “Capture”.

WebP v2: experimentations

Goal:

v2 = like v1 …

… but ‘more’. And speed.

“Web-consumption”, not “Capture”.

WebP v2: experimentations

Goal:

v2 = like v1 …

… but ‘more’. And speed.

And HDR.

“Web-consumption”, not “Capture”.

WebP v2: how do we improve upon v1?

What can we do differently than AV1?

WebP v2: how do we improve upon v1?

- floating partitioning
- small-context residual coding
- non-classic residuals
- custom predictors
- CfL
- lossy/lossless alpha
- more filters
- more predictors
- interruptibility
- custom CSP transform
- ANS + adaptive multi-symbol dictionaries
- tiles

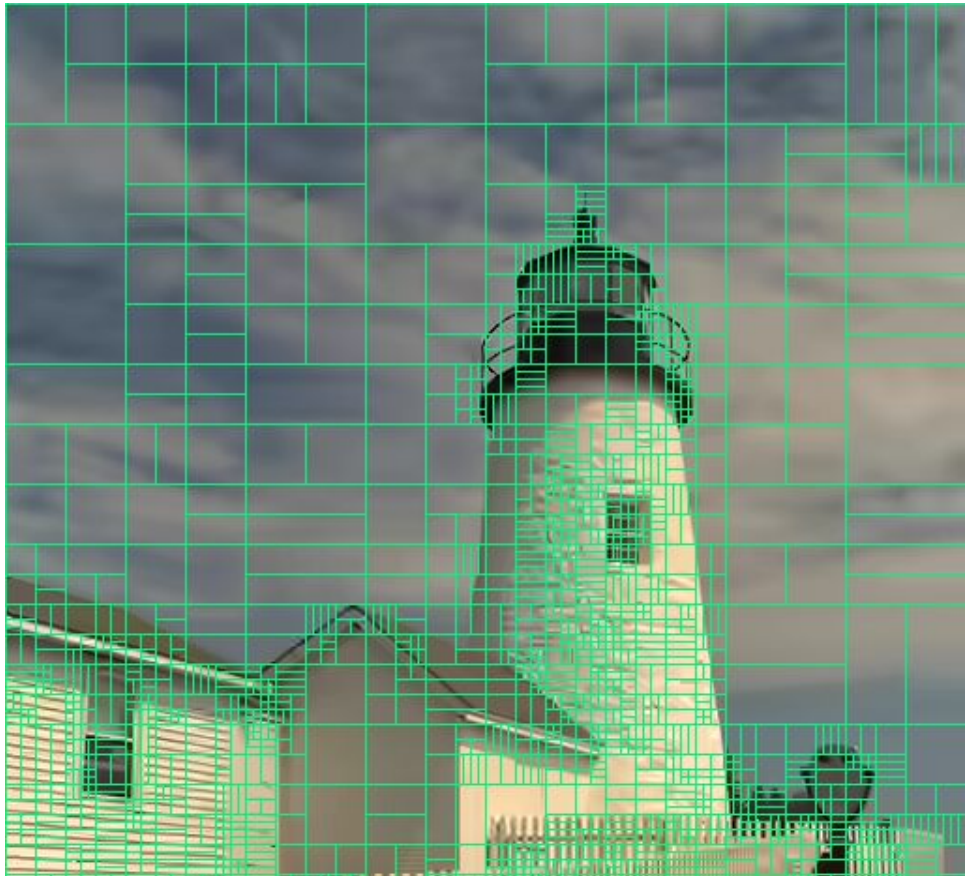
WebP v2: how do we improve upon v1?

- floating partitioning [wip]
- small-context residual coding [go]
- non-classic residuals [failed so far]
- custom predictors [failed so far]
- CfL [go]
- lossy/lossless alpha [go]
- more filters [wip]
- more predictors [failed so far]
- interruptibility [go]
- custom CSP transform [go]
- ANS + adaptive SIMD multi-symbol dictionaries [go]
- tiles [go]

WebP v2: how do we improve upon v1?

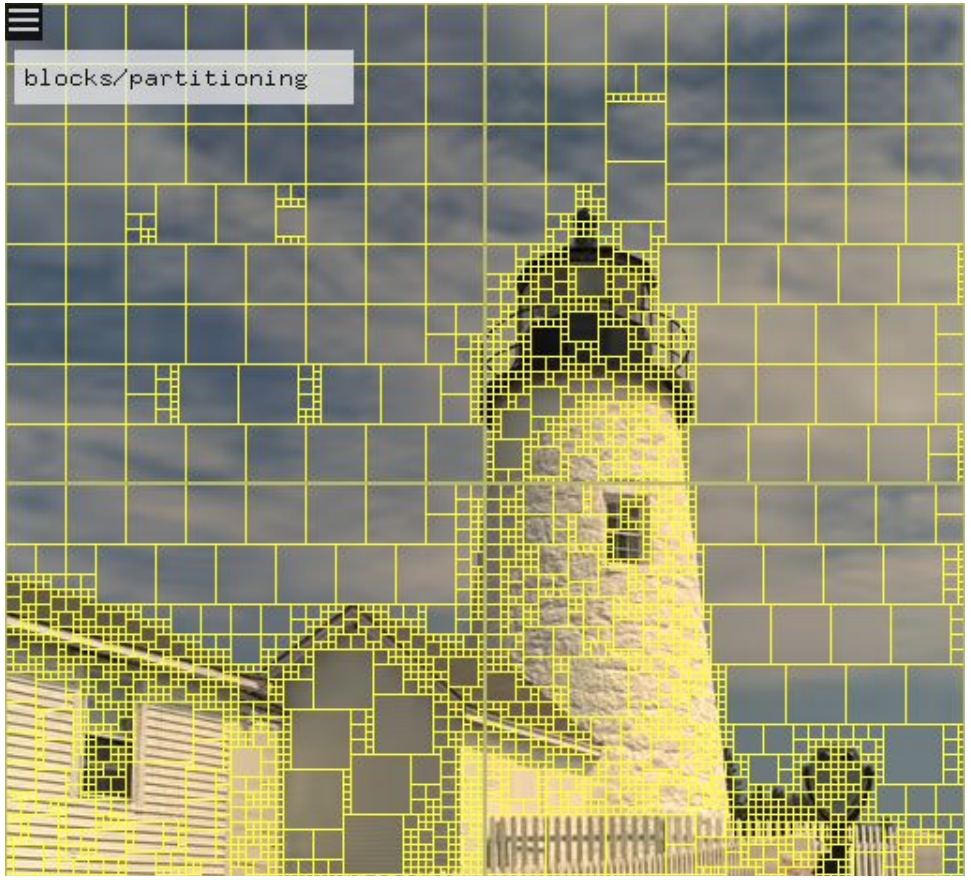
- floating partitioning [wip]
- small-context residual coding [go]
- non-classic residuals [fail]
- custom predictors [fail so far]
- CfL [go]
- lossy/lossless alpha [go]
- more filters [wip]
- more predictors
- interruptibility [go]
- custom CSP transform [go]
- ANS + adaptive multi-symbol dictionaries [go]
- tiles

classic AV1 block partitioning



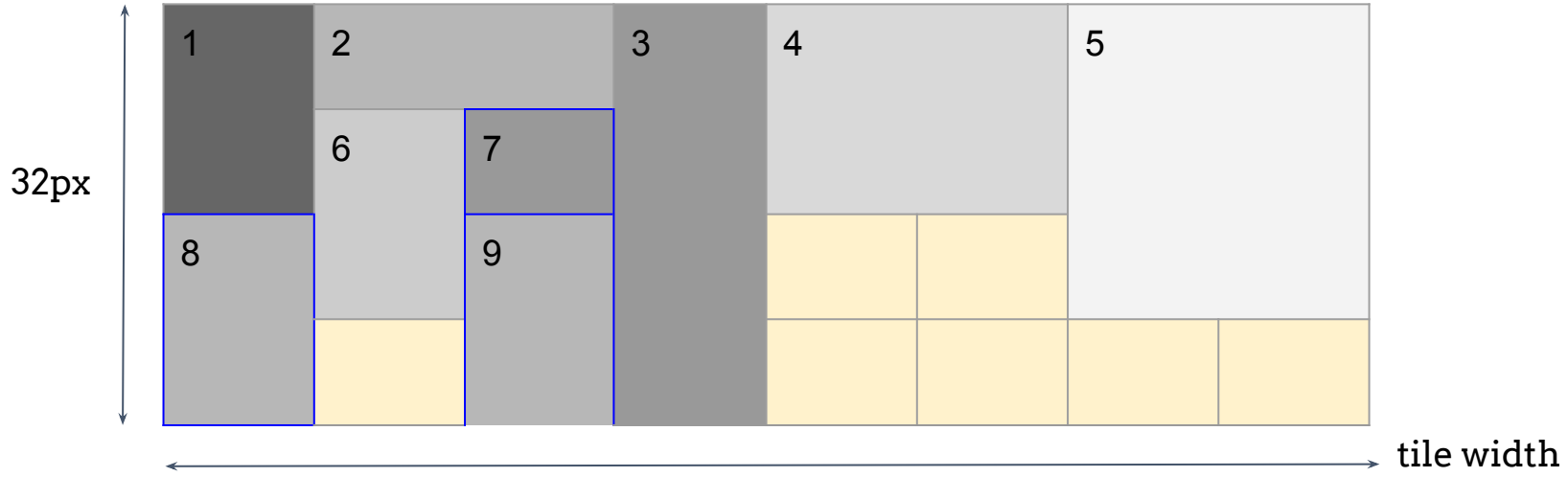
(low quality)

floating block-partitioning



floating block-partitioning

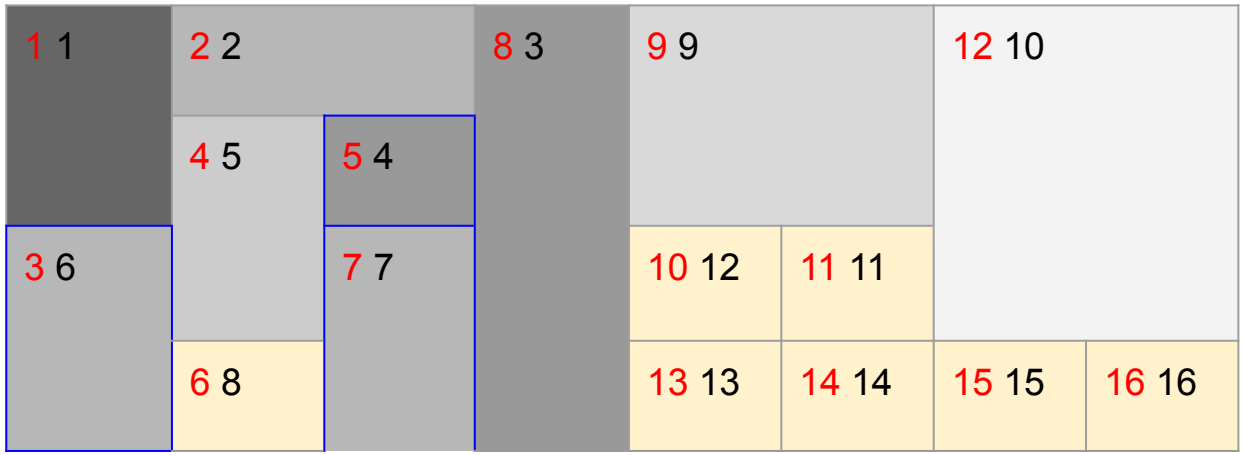
Parsing order = lexicographic order



X-Y sorted
Buffer = 32 px-high rolling cache (max block = 32x32)
Memory = O(32 * tile_width)

floating block-partitioning

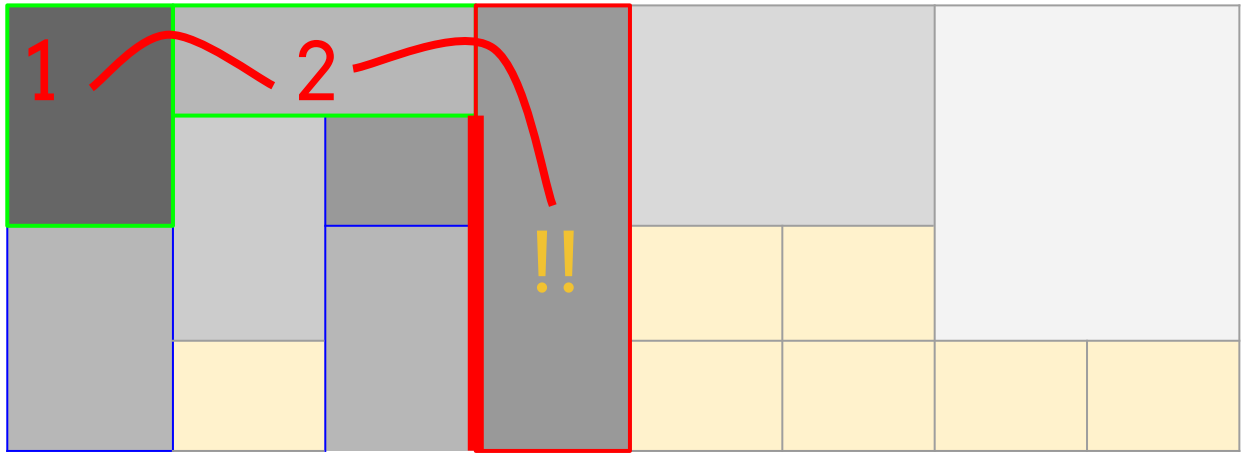
Parsing order != decoding order



Strategy: try to maximize the left-sample availability

floating block-partitioning

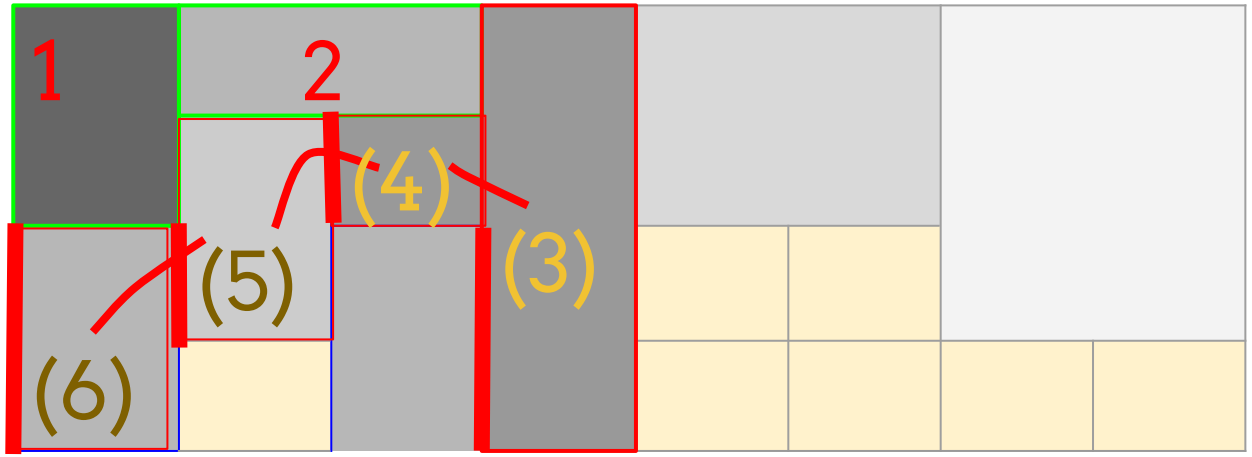
Parsing order != decoding order



Strategy: try to maximize the left-sample availability

floating block-partitioning

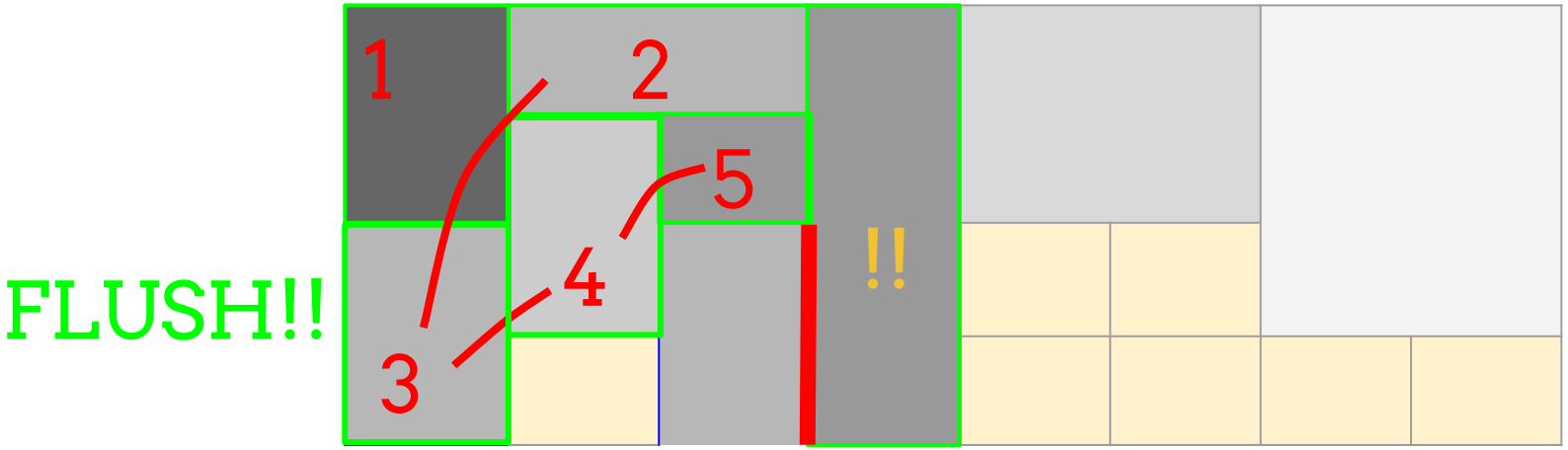
Parsing order != decoding order



Strategy: try to maximize the left-sample availability

floating block-partitioning

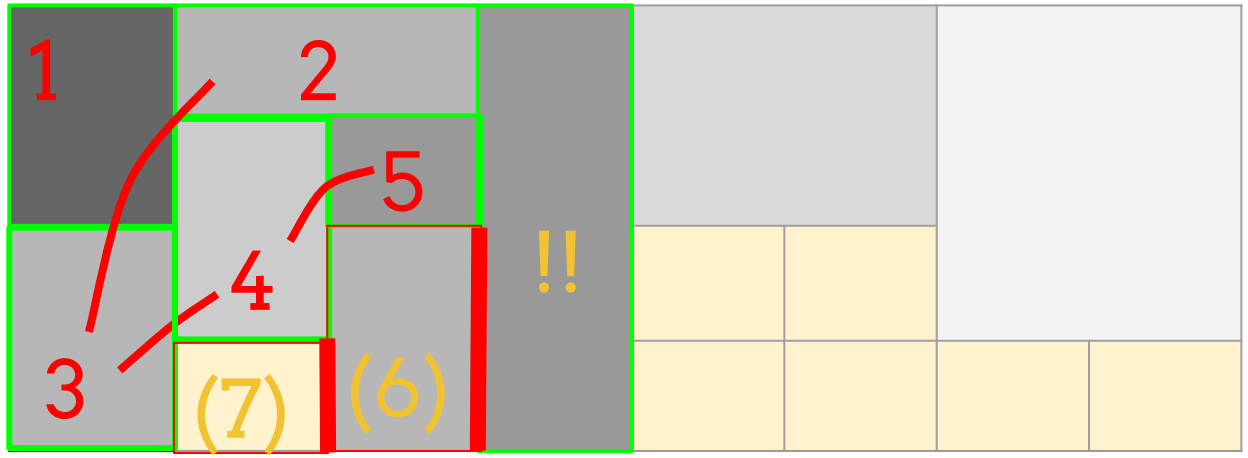
Parsing order != decoding order



Strategy: try to maximize the left-sample availability

floating block-partitioning

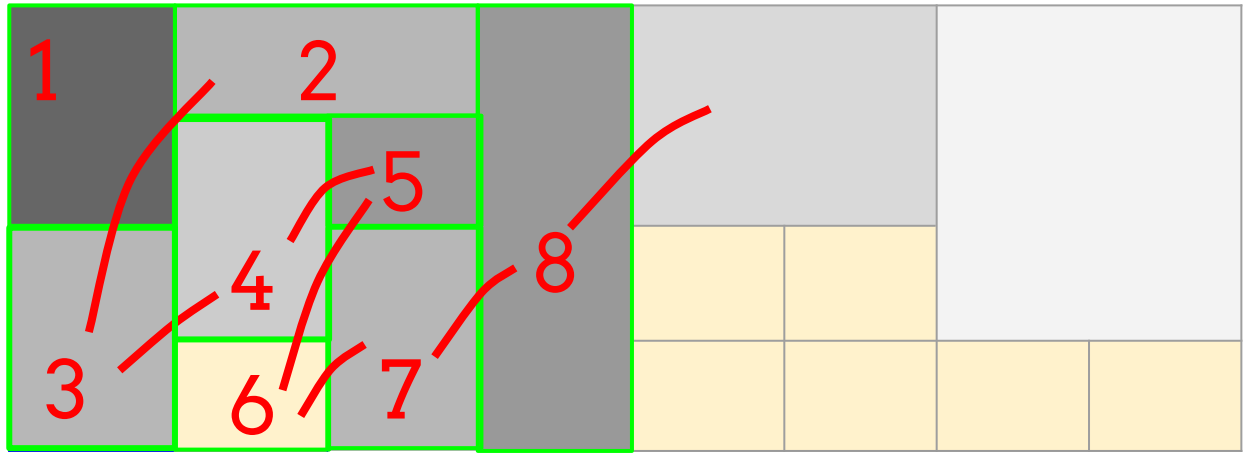
Parsing order != decoding order



Strategy: try to maximize the left-sample availability

floating block-partitioning

Parsing order != decoding order



Strategy: try to maximize the left-sample availability

floating block-partitioning

Problem:

the search space is HUGE

How to do RD-Opt with this vast search space??

Floating partitioning algo

Algo for finding a partitioning of a 32x32 section:

- use variance to label 4x4 blocks with four buckets.

Variance of input 4x4 blocks:

```
14.0 12.5 12.0 11.8 11.3  8.1 11.1 10.1
14.6 12.0 13.3 12.6 11.9  9.9 13.3  8.7
12.2 14.6 12.6 15.0 10.3  9.2 11.5 11.2
74.7 80.8 103.0 118.5 80.1 16.6 13.2 20.5
37.4 33.4 39.2 35.6 34.6 59.8 114.7 93.4
34.5 29.9 33.1 30.2 33.4 30.0 32.4 25.2
32.1 29.9 37.1 34.5 34.7 33.7 29.9 21.7
32.9 31.5 29.6 36.1 35.9 28.7 33.3 29.4
```

Floating partitioning algo

Algo for finding a partitioning:

- use variance to label 4x4 blocks with four buckets.

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

2 2 3 3 2 0 0 0

1 1 1 1 1 2 3 2

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 0

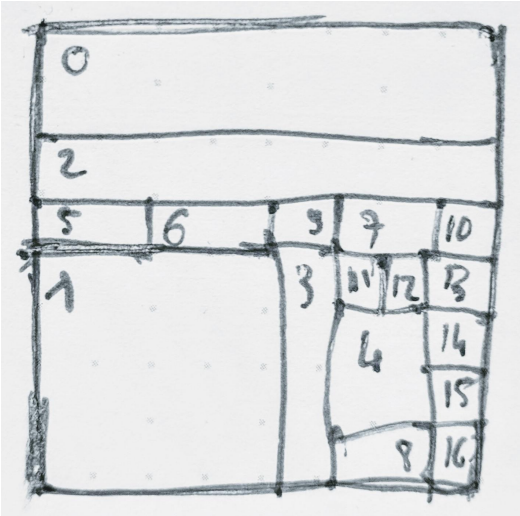
1 1 1 1 1 1 1 1

- lay down boxes with same labels,
- starting from the largest down to the smallest (finishing fill with 4x4 boxes).

Floating partitioning algo

Algo for finding a partitioning:

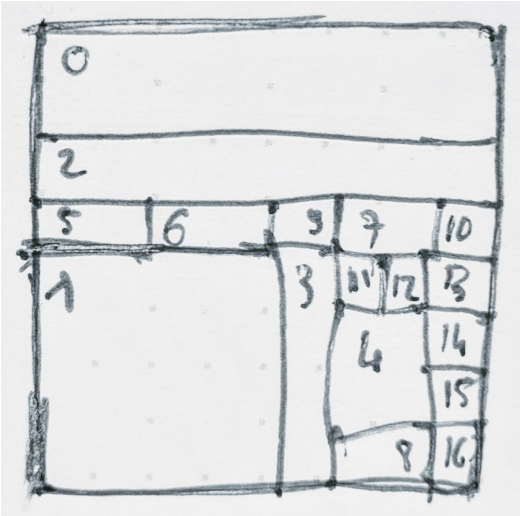
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
2 2 3 3 2 0 0 0
1 1 1 1 1 2 3 2
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1

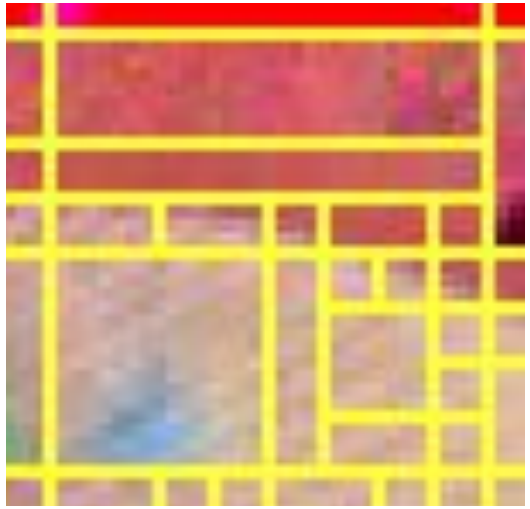


Floating partition algo

Algo for finding a partitioning:

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
2 2 3 3 2 0 0 0
1 1 1 1 1 2 3 2
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1





Floating partitioning algo

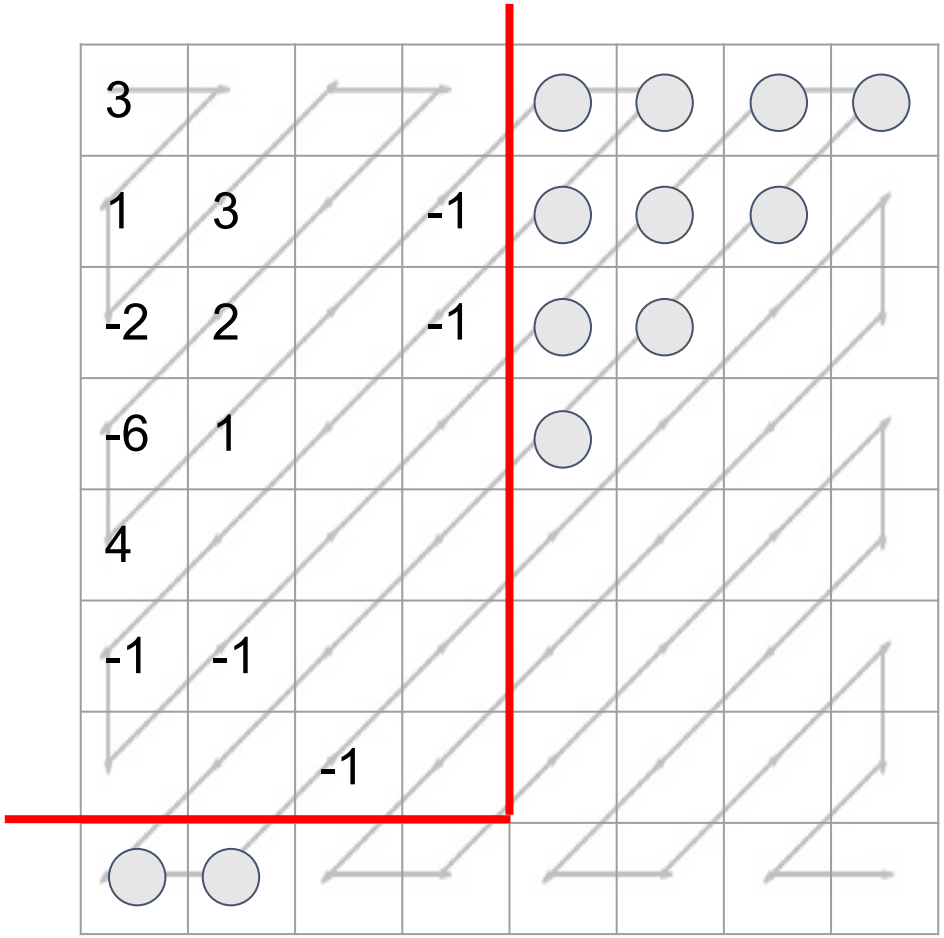
- Variance isn’t necessary a good metric
- too many ‘small’ blocks for filling gaps
- so many other algos to try!

Floating partitioning algo

-> Still a lot of potential trading geometry vs residuals

Residual coding

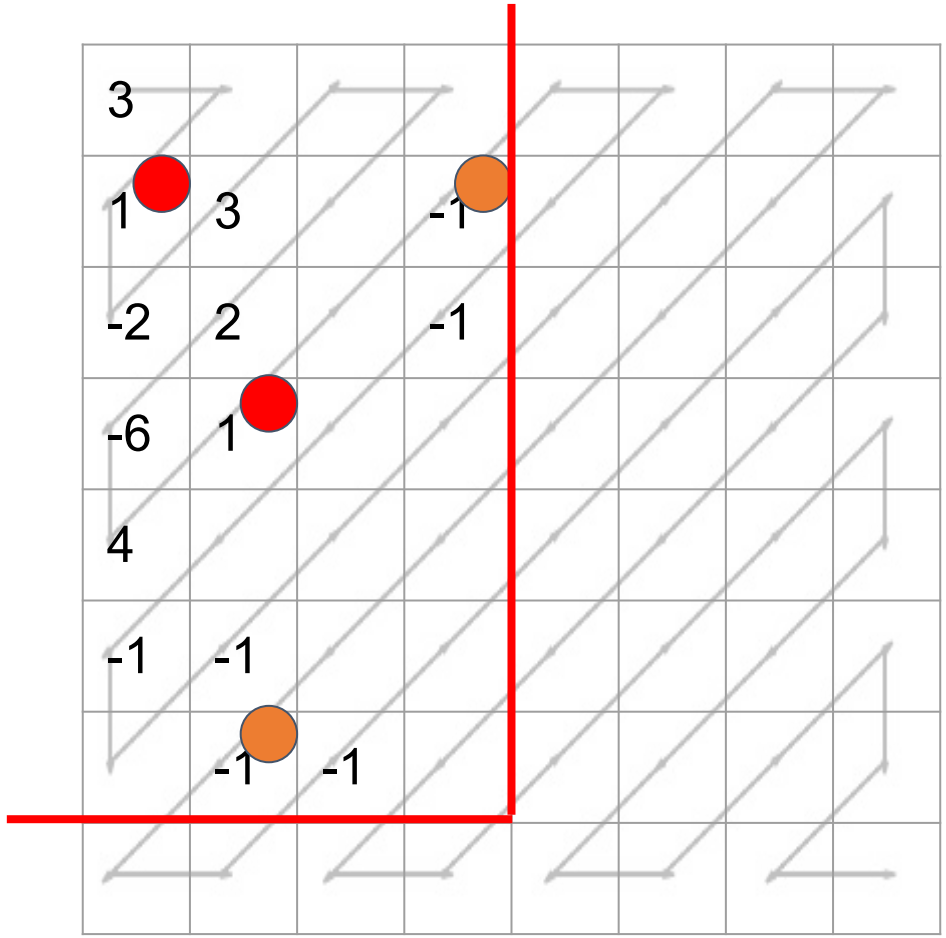
Residual coding



Bounds: use Adaptive Bit to say if the residuals are bounded in X/Y. If bounded, store bounds as range.

Residual: parse as zigzag but skip anything that is outside the box: ○

Residual coding



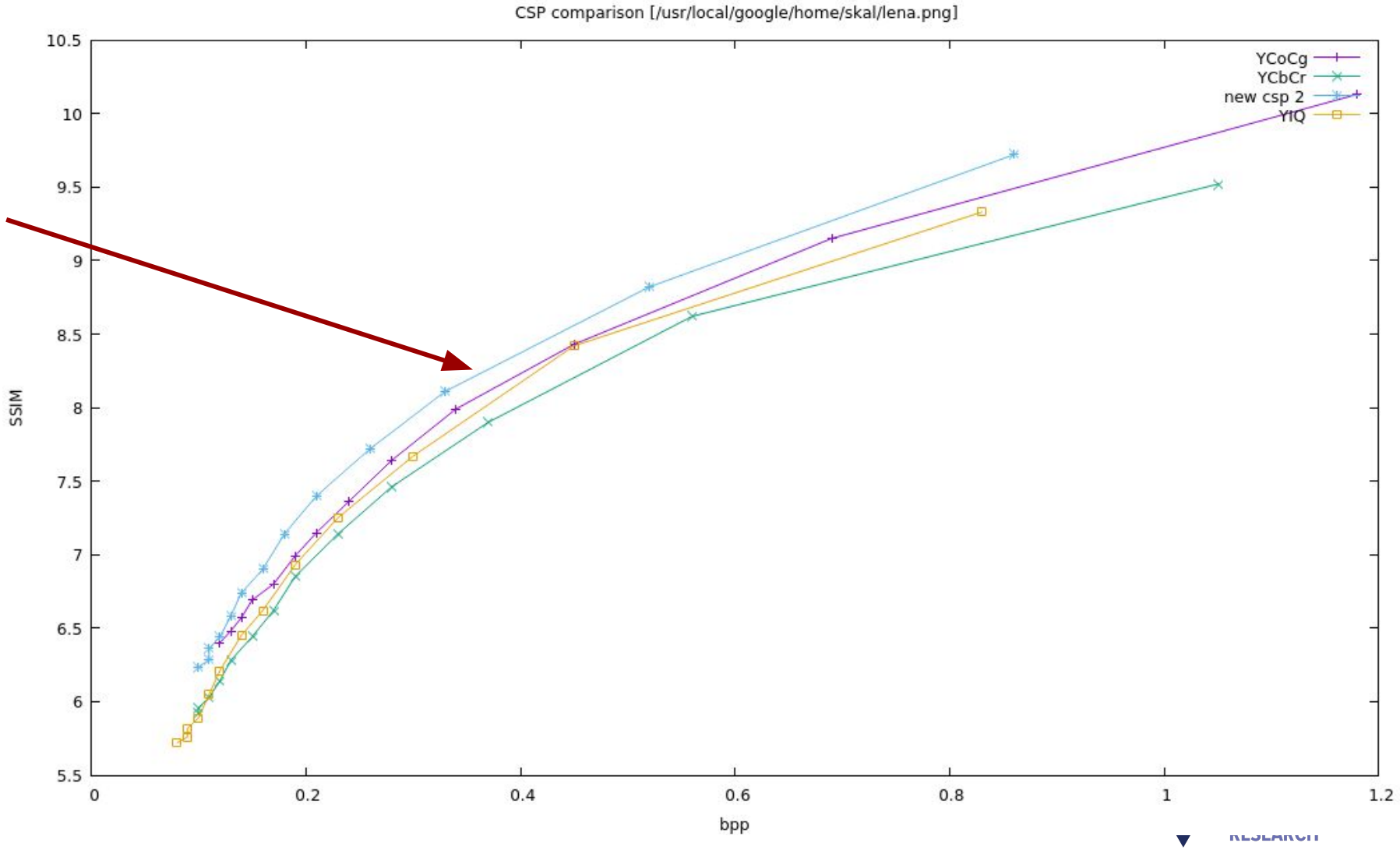
EOB: Adaptive Bit, but only if we have already touched both sides of the bounding box.

Only 1s after When finding a 1, ABit that indicates whether all elements after are 1s.

Custom CSP transform

Custom CSP transform

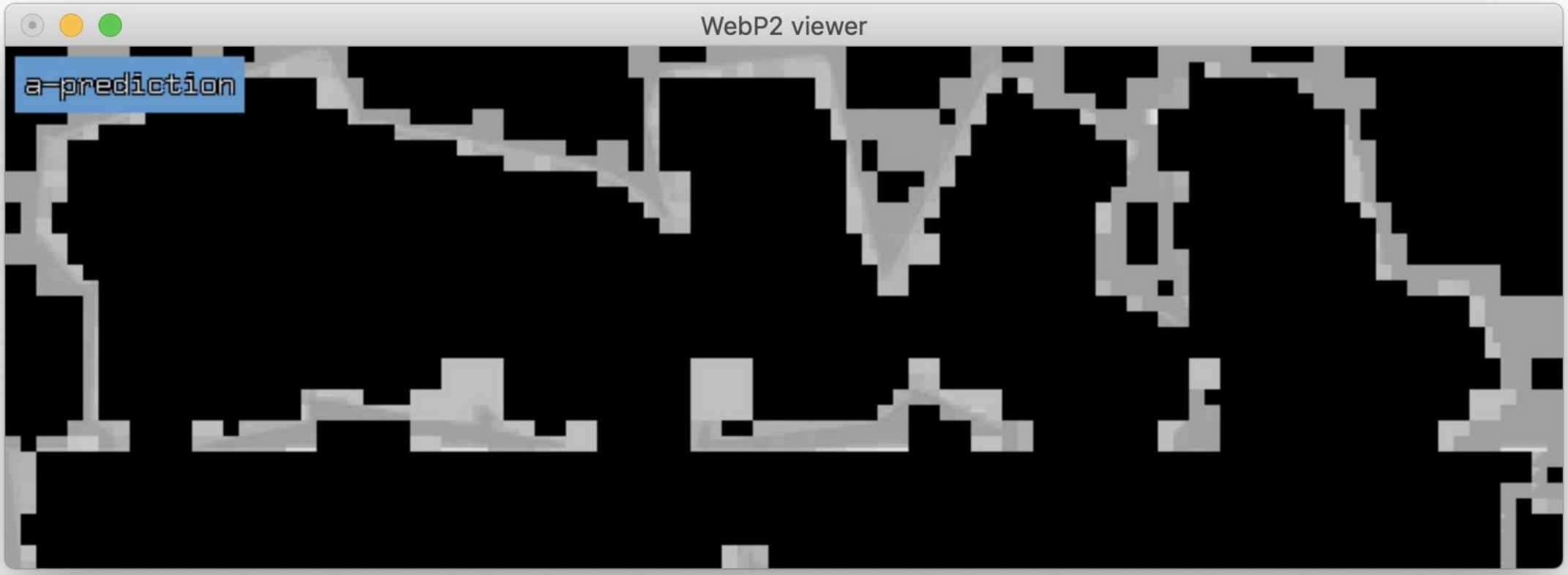
Use PCA to tight-fit the color transform matrix.



Lossy-lossless alpha mix

Lossy-lossless alpha mix

Lossy-lossless alpha mix



Triangle-based preview

218 bytes.

In the header.

Triangle-based preview

[grid = 64 x 64, nb_colors = 5 nb_pts = 187]

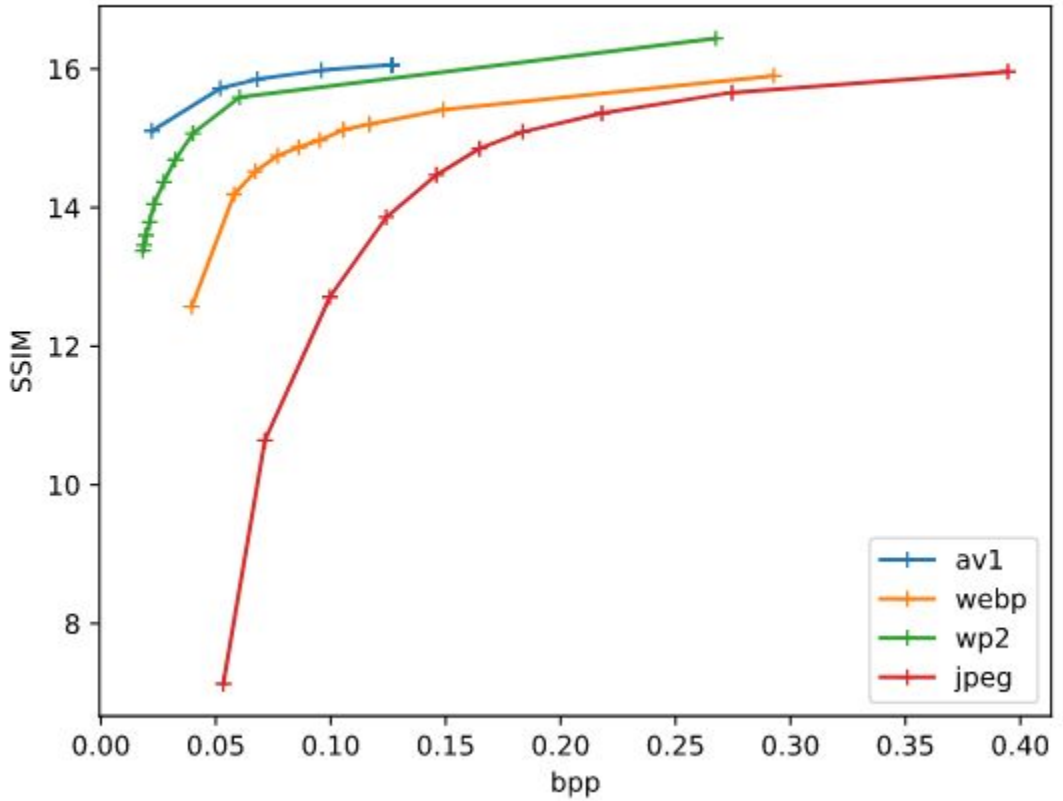
Colormap (w/ use counts): (50) (39) (33) (34) (31)

[ICIP 2018 Paper.]

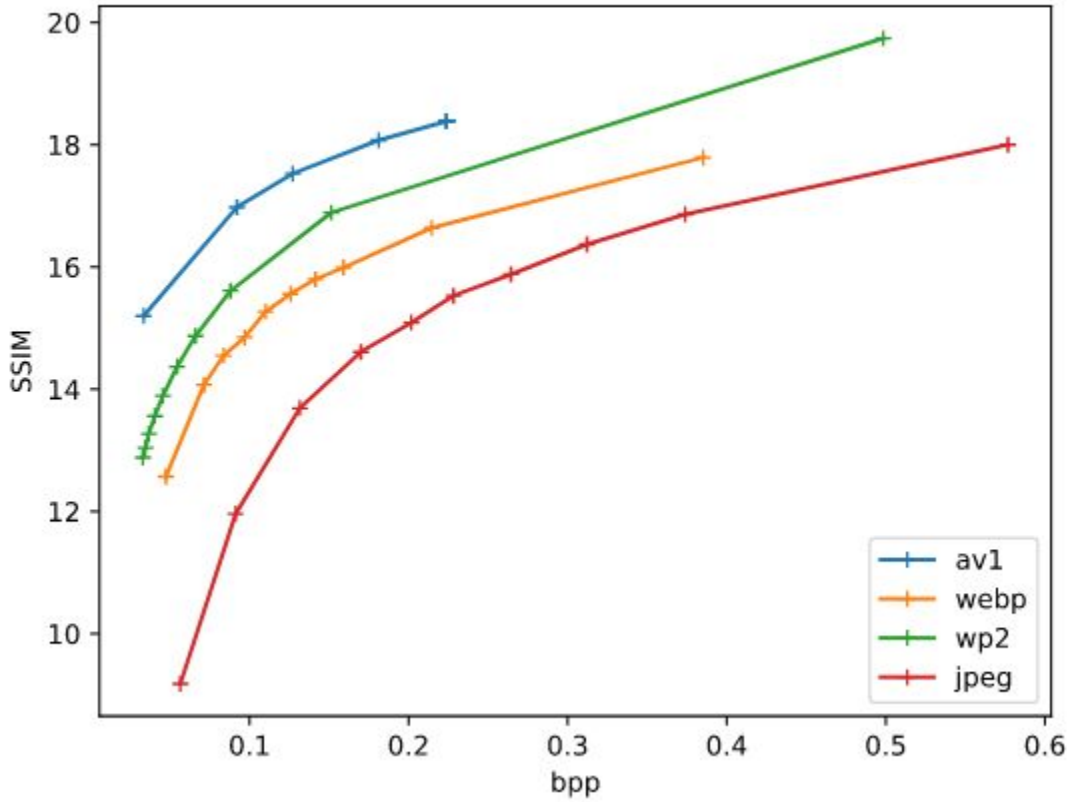
WebP v2:

results so far

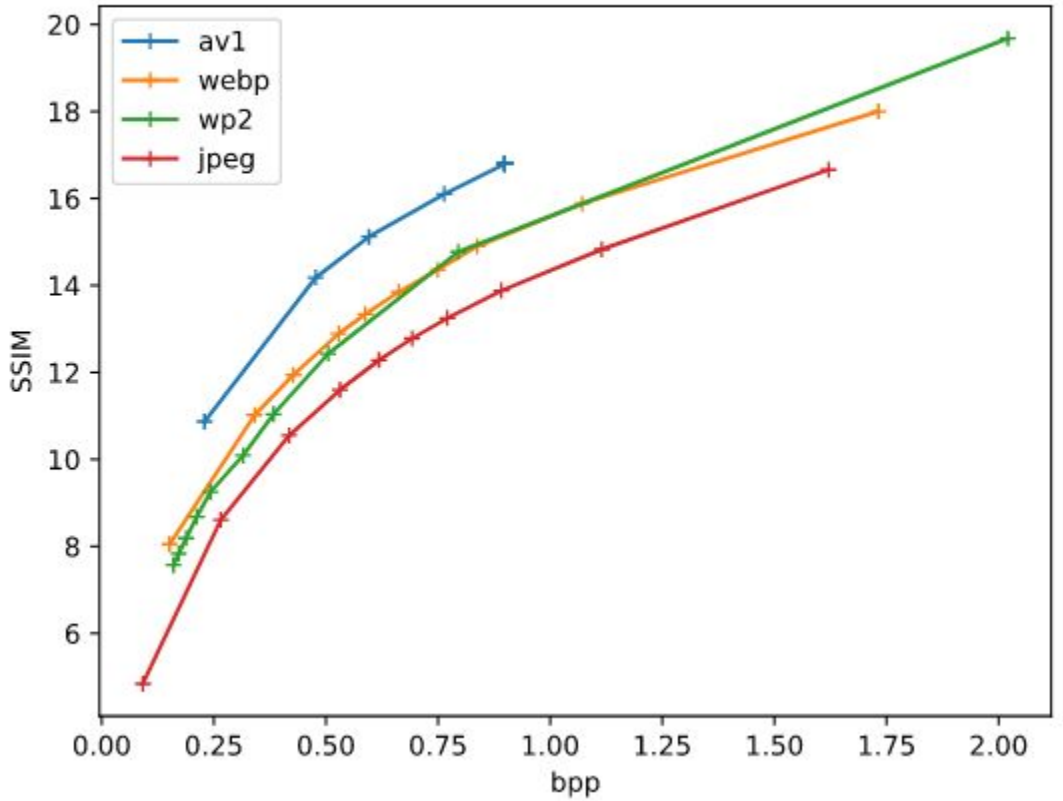
WebP v2: results so far. The Good.



WebP v2: results so far. The Bad.



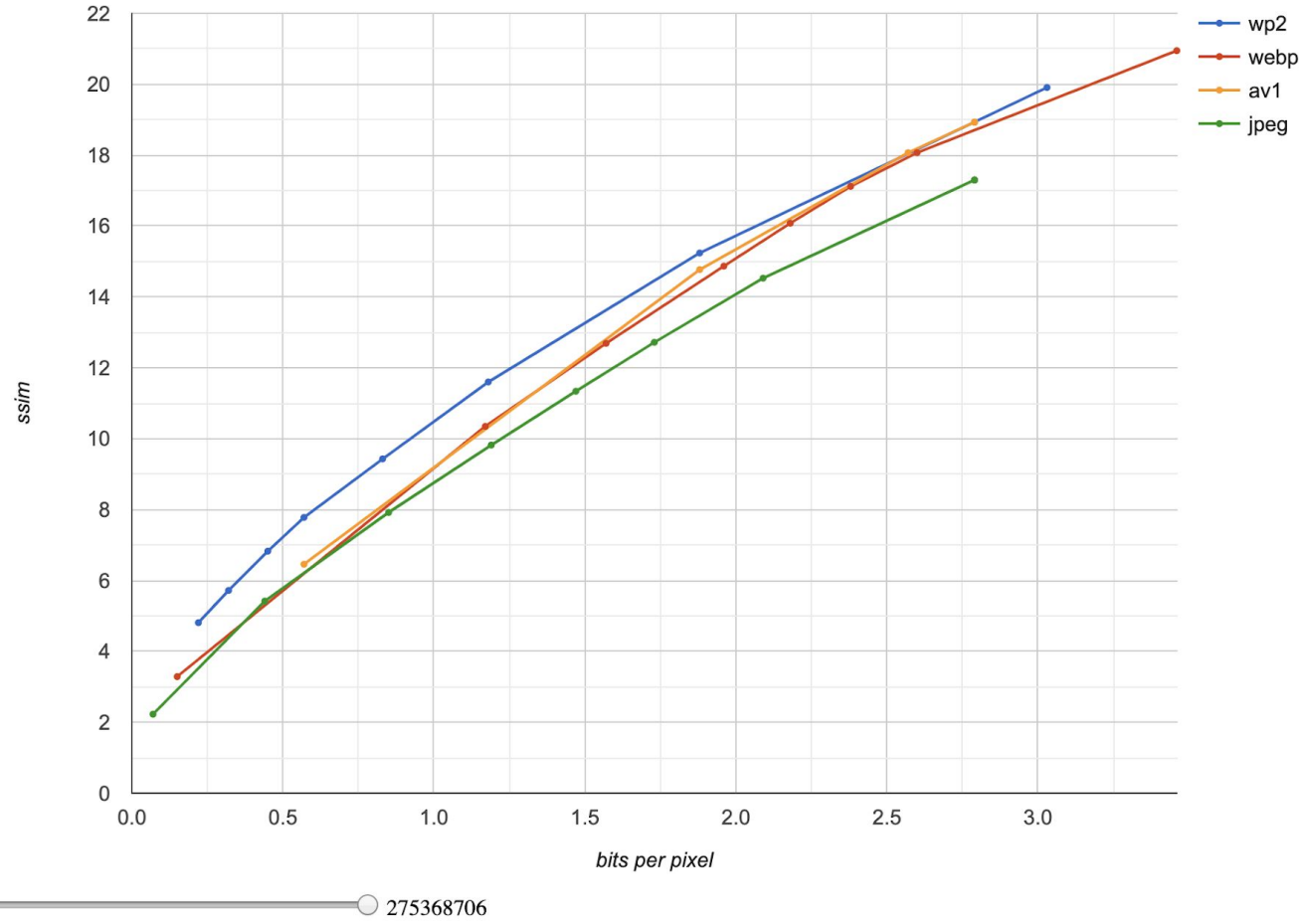
WebP v2: results so far. The Ugly.



also good

Metric: ○ PSNR ◉ SSIM

Image: ddlfpppdncpllfjjc



275368706

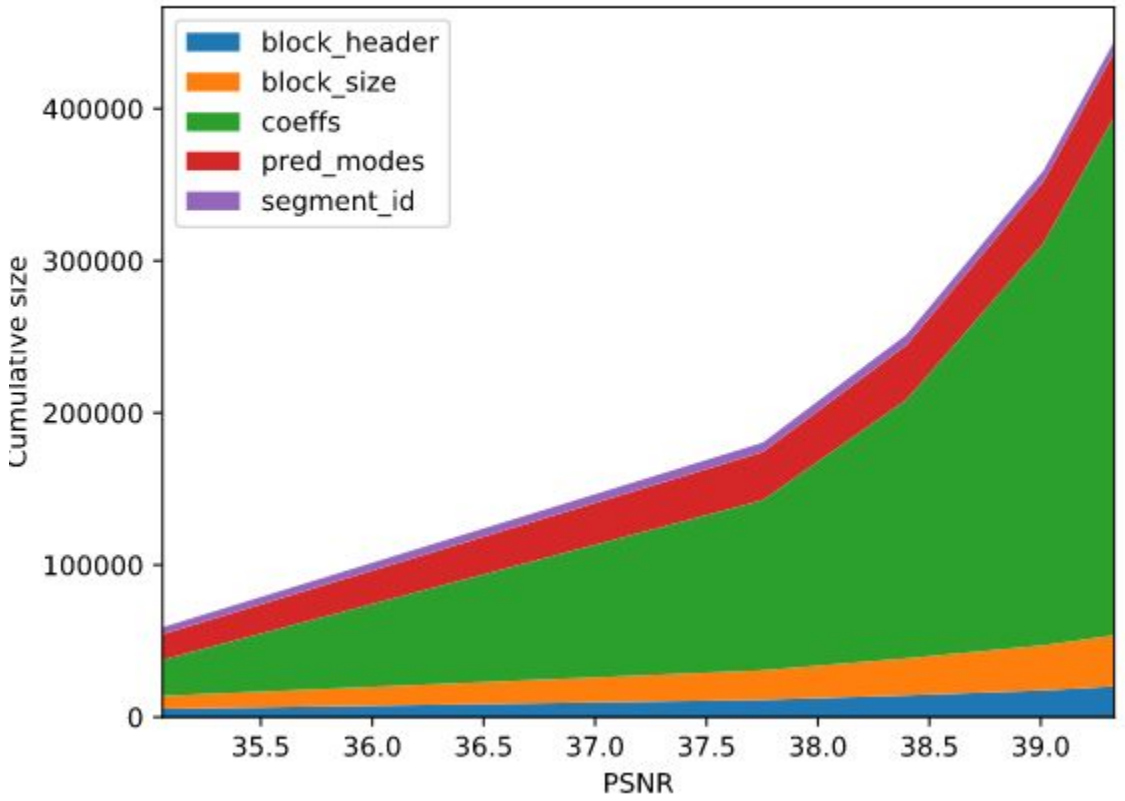
Source image

jpeg: quality 85 - bpp 2.79 - ssim 17.30

ALLIANCE FOR OPEN MEDIA RESEARCH Symposium 2019

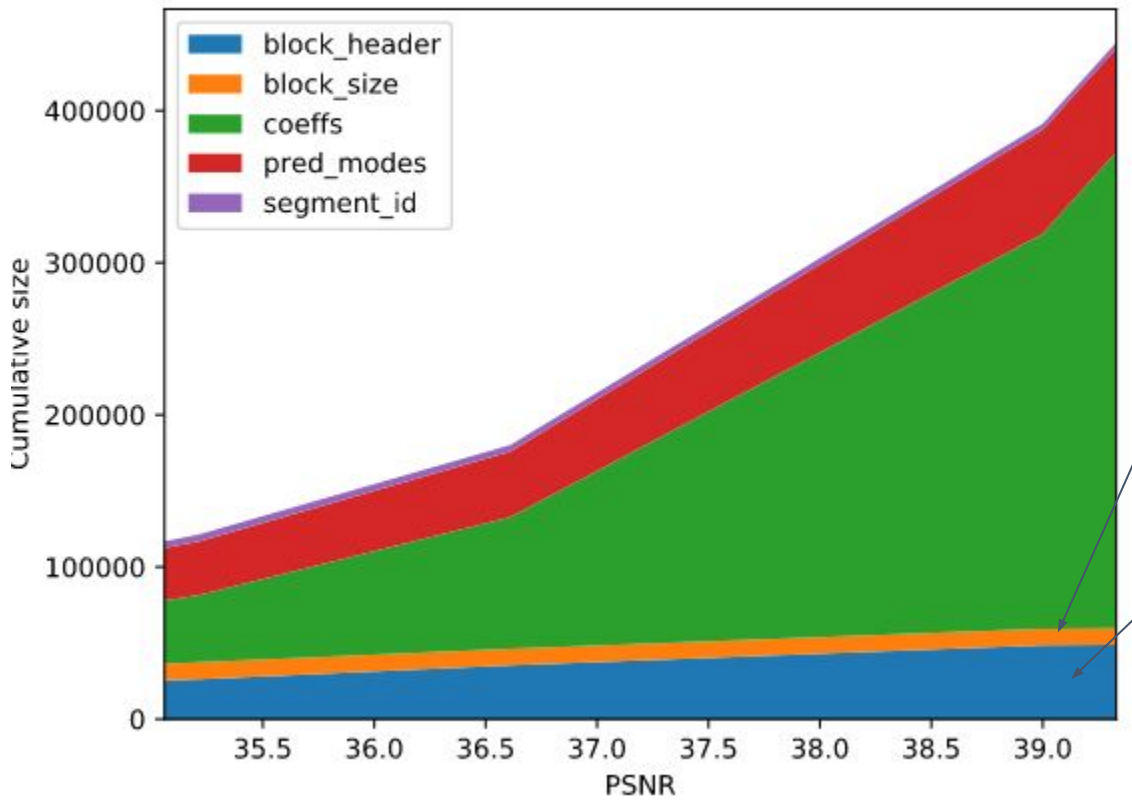
Syntactic decomposition

AV1



Syntactic decomposition

WP2



block size coding seems more efficient!

at the detriment of block header

trading geometry vs residuals!

Enc Speed comparison

```
> ./examples/rd_curve kodim19.png -nomt -av1 -jpeg -webp -ssim

# Q       {size (bytes), bpp, psnr (dB), SSIM*, enc-time (sec), dec-time (sec)}
#      |            WP2                          |            WebP                         |            AV1                          |            JPEG
0.0          5074 0.10 27.07  6.50 1.79 0.10      5028 0.10 26.49  6.44 0.04 0.00      8305 0.17 30.15  7.98  5.28 0.02      4315 0.09 22.65 5.12  0.01 0.00
12.1         5776 0.12 27.50  6.69 1.86 0.10     13026 0.27 30.42  8.10 0.04 0.00     29446 0.60 35.15 11.92 12.23 0.02     11653 0.24 28.51 7.17  0.01 0.00
24.3         6834 0.14 28.24  6.99 1.81 0.09     18850 0.38 31.72  9.09 0.03 0.00     47852 0.97 37.74 14.02 18.20 0.03     19015 0.39 30.71 8.55  0.01 0.00
36.4         8308 0.17 29.04  7.32 1.83 0.09     24882 0.51 32.88 10.06 0.04 0.00     54919 1.12 38.48 14.61 20.71 0.03     25183 0.51 31.94 9.38  0.01 0.00
48.6        11780 0.24 30.17  7.96 1.70 0.11     31518 0.64 34.04 11.04 0.04 0.00     54919 1.12 38.48 14.61 20.71 0.04     30969 0.63 32.97 10.12 0.02 0.00
60.7        17264 0.35 31.79  9.04 1.79 0.11     37818 0.77 34.99 11.79 0.04 0.00     54919 1.12 38.48 14.61 20.86 0.03     36423 0.74 33.78 10.72 0.01 0.00
72.9        28386 0.58 34.12 10.80 1.92 0.10     44738 0.91 35.93 12.52 0.05 0.00     54919 1.12 38.48 14.61 20.95 0.03     46192 0.94 35.07 11.67 0.02 0.00
85.0        65536 1.33 39.15 14.45 2.28 0.11     73180 1.49 38.92 14.84 0.05 0.01     54919 1.12 38.48 14.61 21.22 0.03     65399 1.33 37.25 13.18 0.02 0.00
```

WP2
120x

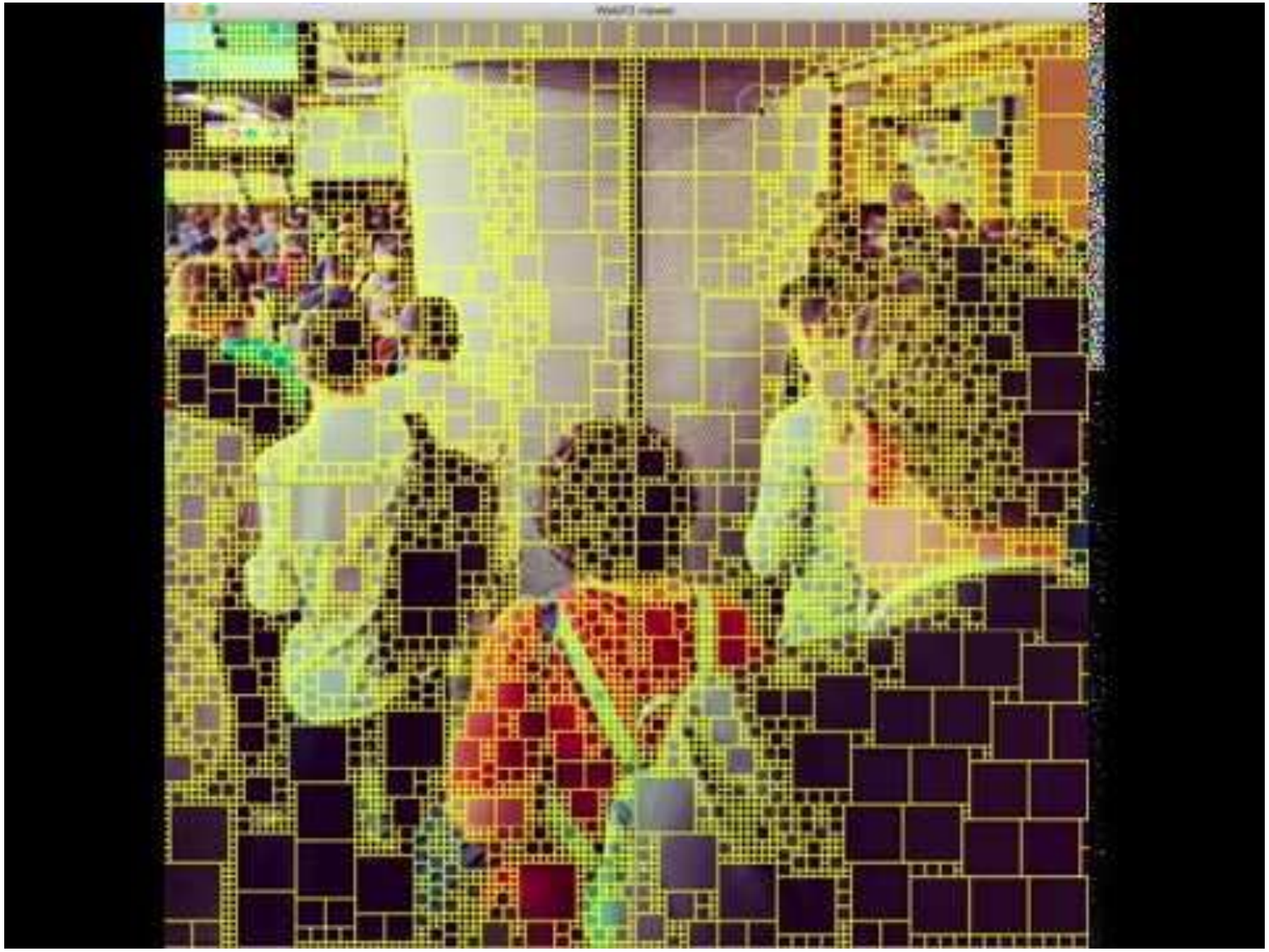
WebP
3x

AV1
1200x

jpeg
= ref

WebP v2: demo

[video]



ALLIANCE FOR OPEN MEDIA RESEARCH Symposium 2019

Conclusion

Plan for 2020:

- finalize the decoding tools for experiments
- release the code base as starting point

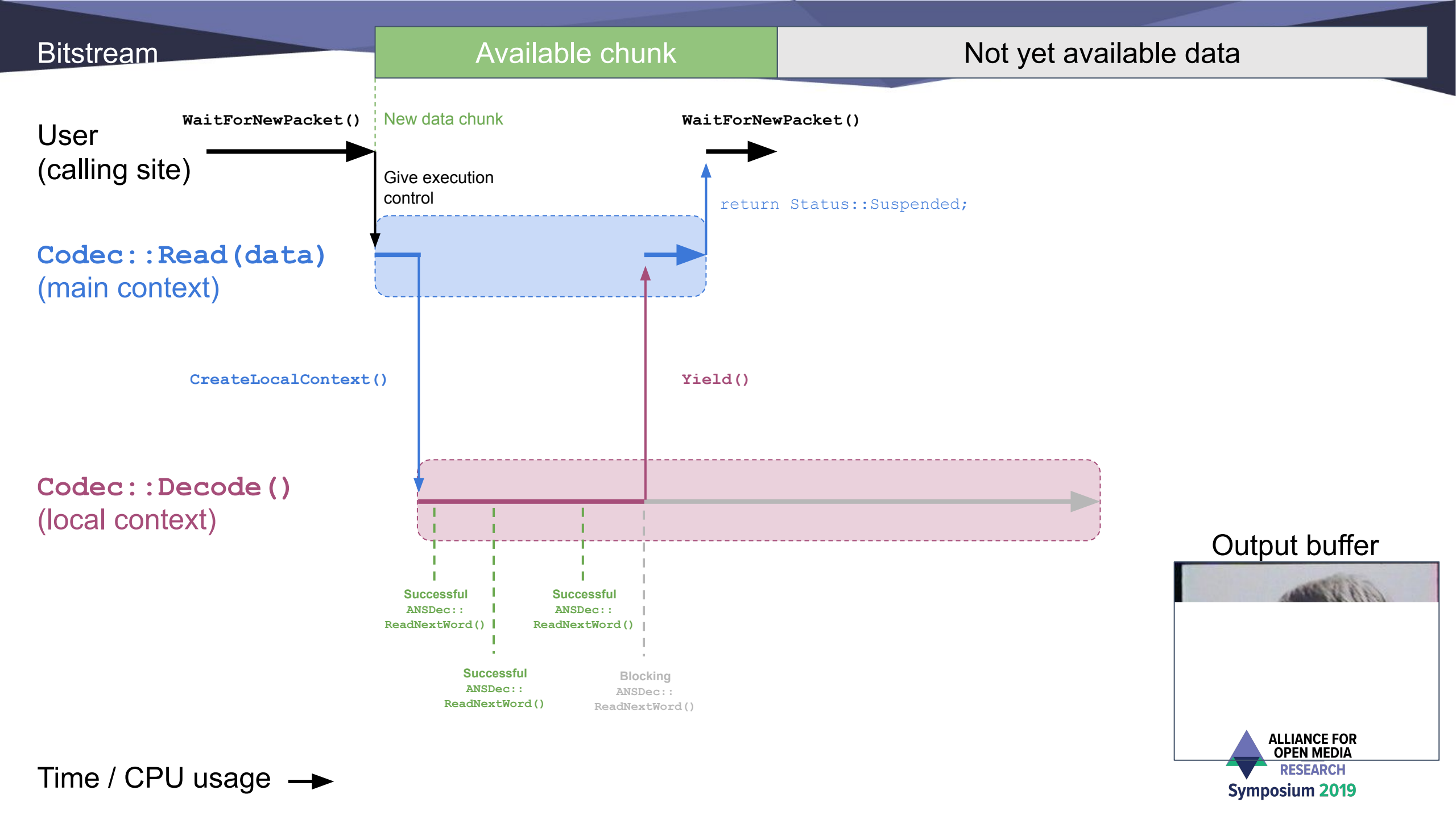
Thanks!

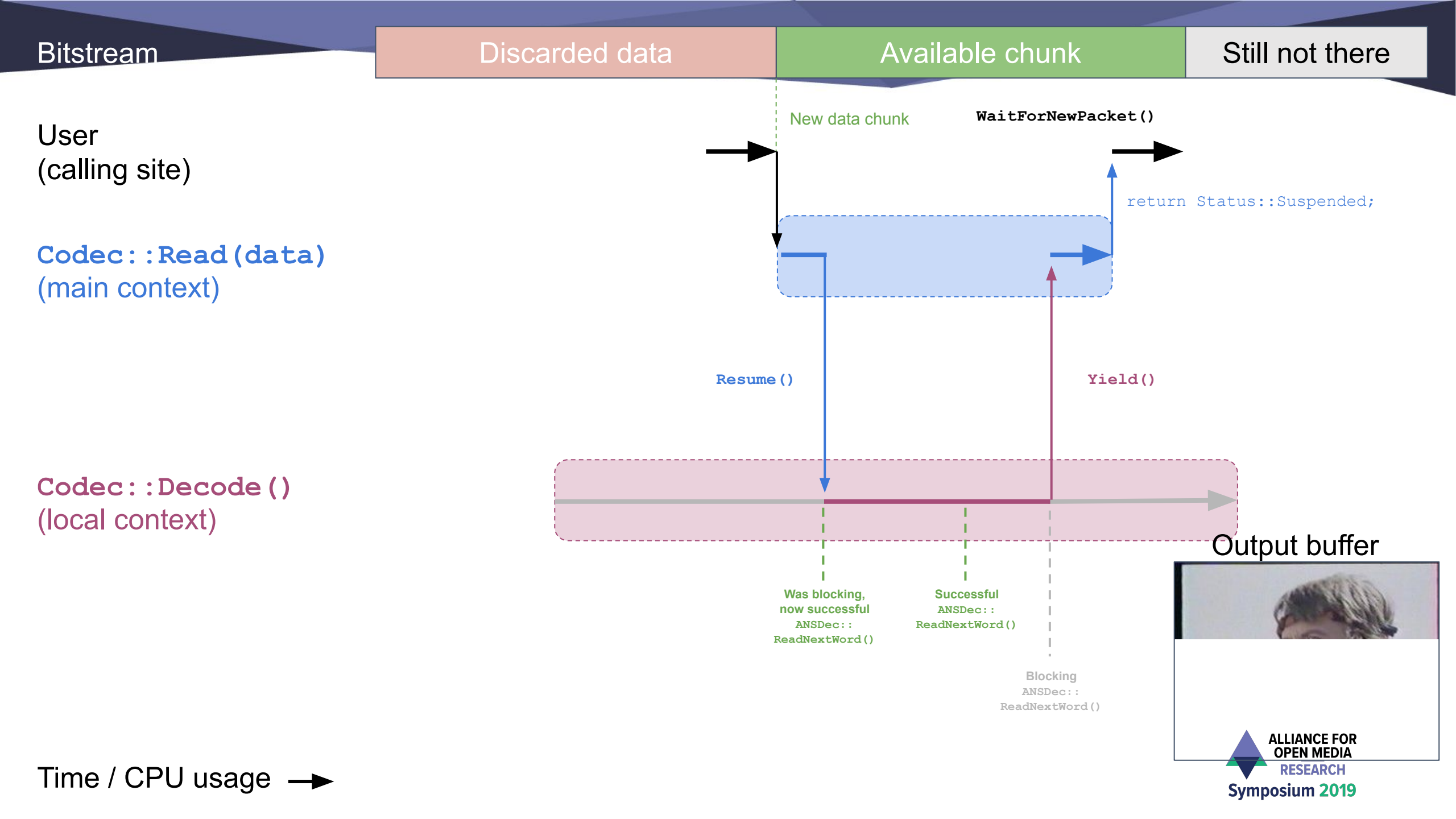
Questions?

Extra material

incremental decoding

using fiber / coroutines to pass control around between codec and network.


Bitstream
Available chunk
Not yet available data
User
(calling site)
WaitForNewPacket()
New data chunk
WaitForNewPacket()
Give execution
control
return Status::Suspended;
Codec::Read(data)
(main context)
CreateLocalContext()
Yield()
Codec::Decode()
(local context)
Successful
ANSDec::
ReadNextWord()
Successful
ANSDec::
ReadNextWord()
Successful
ANSDec::
ReadNextWord()
Blocking
ANSDec::
ReadNextWord()
Output buffer
Time / CPU usage
ALLIANCE FOR
OPEN MEDIA
RESEARCH
Symposium 2019


Bitstream
Discarded data
Available chunk
Still not there
User
(calling site)
Codec::Read(data)
(main context)
Codec::Decode()
(local context)
New data chunk
WaitForNewPacket()
return Status::Suspended;
Resume()
Yield()
Was blocking, now successful ANSDec::ReadNextWord()
Successful ANSDec::ReadNextWord()
Blocking ANSDec::ReadNextWord()
Output buffer
Time / CPU usage
ALLIANCE FOR OPEN MEDIA
RESEARCH
Symposium 2019

Bitstream

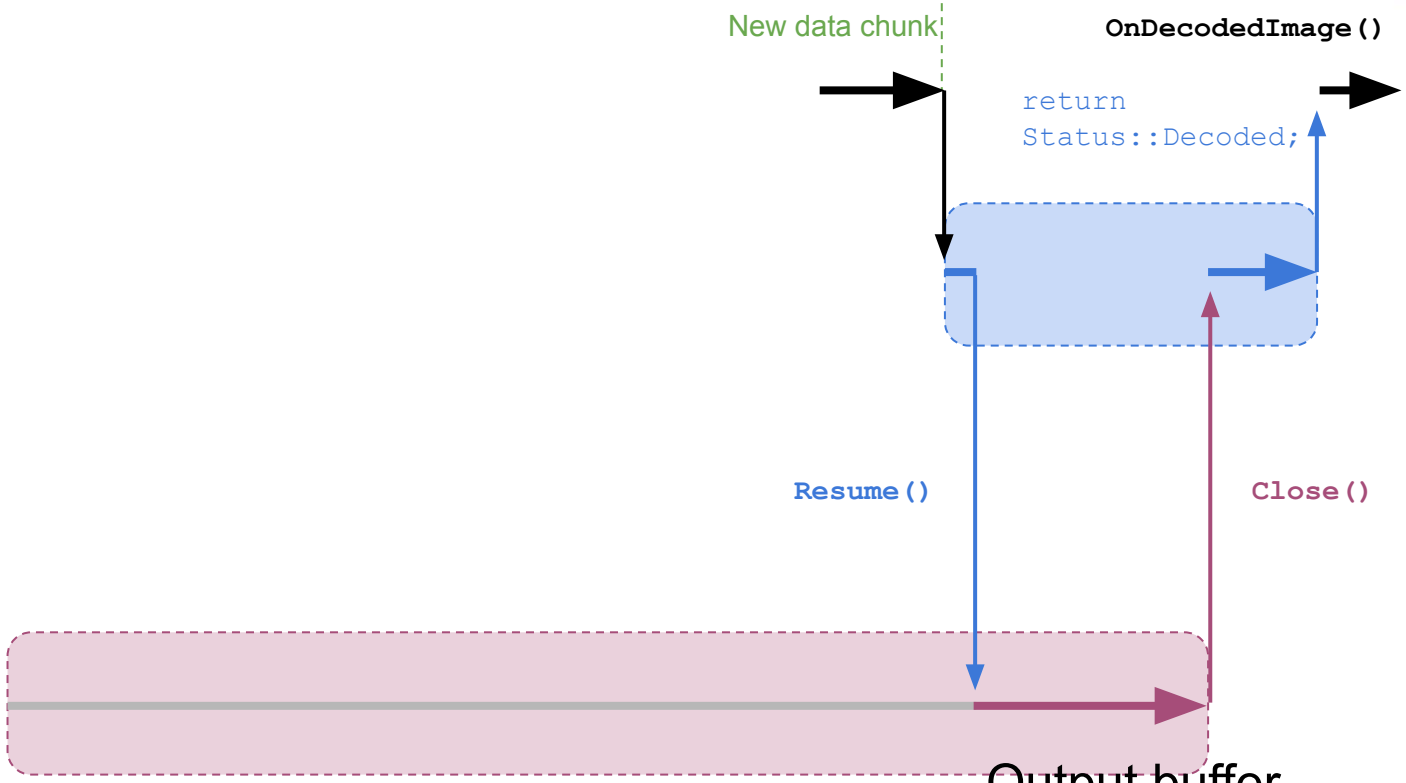
Discarded data

Available chunk

User
(calling site)

`Codec::Read(data)`
(main context)

`Codec::Decode()`
(local context)



Time / CPU usage →

RESEARCH
Symposium 2019

Incremental decoding

Don’t assume you have the complete data for the whole frame

one must be able to quickly suspend / resume the decoding with as few work as possible

-> check points

-> coroutines in the bit-reader’s TryReadNext()

Corollary: good decoding error trapping and reporting is critical

Memory consumption

Video decoding = several buffers (Ref, Alt-ref, etc.)

WebP = O(width) memory consumption

Blit to screen ASAP

animation = 1 buffer only

Hardware = difficult for images

Hardware decoding is:

- per-frame oriented, non-interruptible
- tricky to re-configure
- non-parallelizable
- unstable, sandboxed
- has transfer overhead

Hardware = difficult for images

WebP experiment with Android vp8 hardware:

only 50% faster, but a lot of extra system complexity

-> Let’s target software decoding !