Efficient Stream Reduction on the GPU

David Roger, Ulf Assarsson, Nicolas Holzschuch

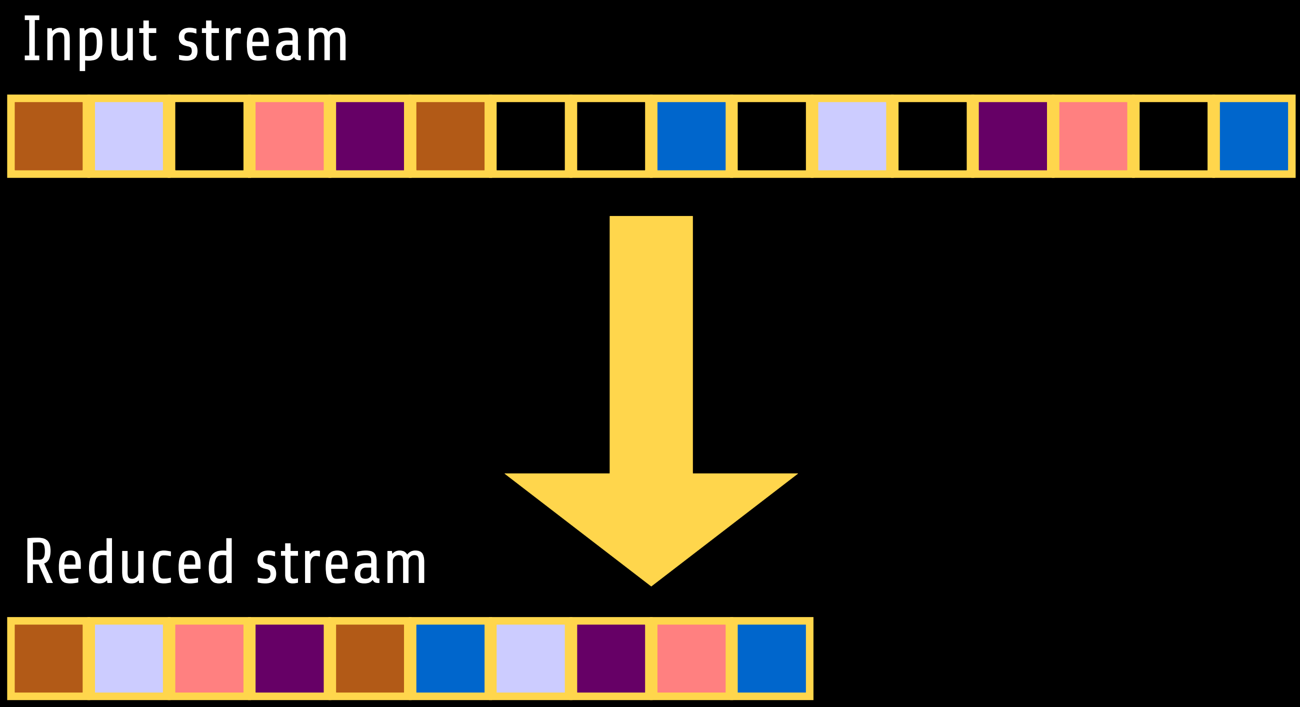
Grenoble University

Chalmers University of Technology

Cornell University

Stream Reduction

Removing unwanted elements from a stream



Applications

- Tree traversal:
  - Ray tracing
  - Collision detection
- Often the bottleneck

Sequential Algorithm

- Algorithm:

```
i=0
for j=0 to n-1 do
    if x[j] is valid then
        x[i]=x[j]
        i=i+1
```

- Easy: one single loop
- Linear complexity

On GPU

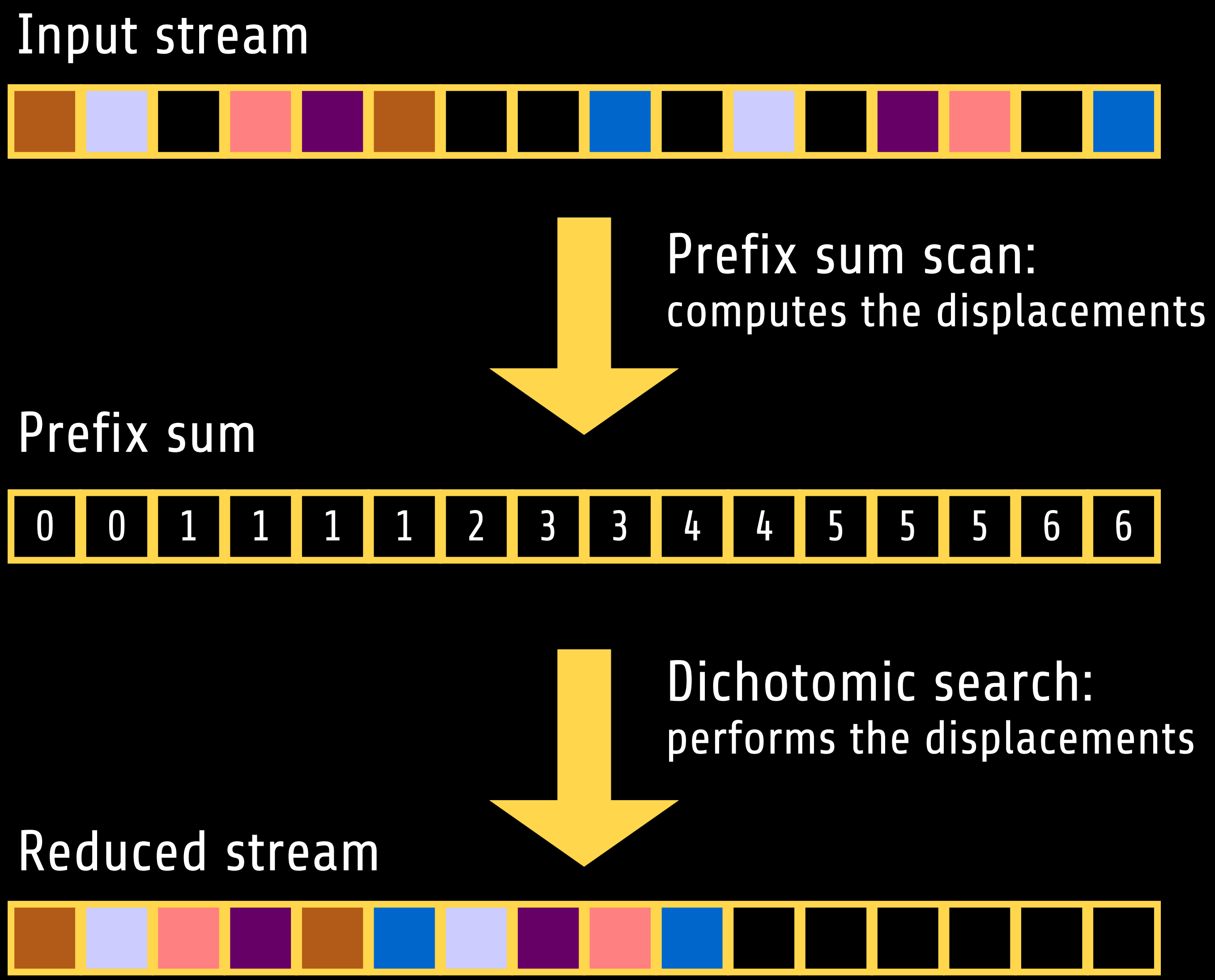
- Parallelism

- We assume no scatter
  - We will speak about scatter later

Talk Structure

- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

Previous works: Horn's Method



Previous works

- Prefix sum scan
  - Hillis and Steele, Horn: O(n log n)
  - Blelloch, Sengupta *et al.*, Harris *et al.*: O(n)
  - Sengupta *et al.* Hybrid: O(n)
- Dichotomic search: O(n log n)
- Overall complexity: O(n log n)

Other approaches

- Geometry shader + stream output
  - NV_transform_feedback
  - Input stream: vertices in a VBO
  - Geometry shader discards NULL elements
  - Output stream: vertices in a VBO
    - No fragments, no fragment shader
- Bitonic sort
  - Slow
- Sum scan + Scatter with vertex engine

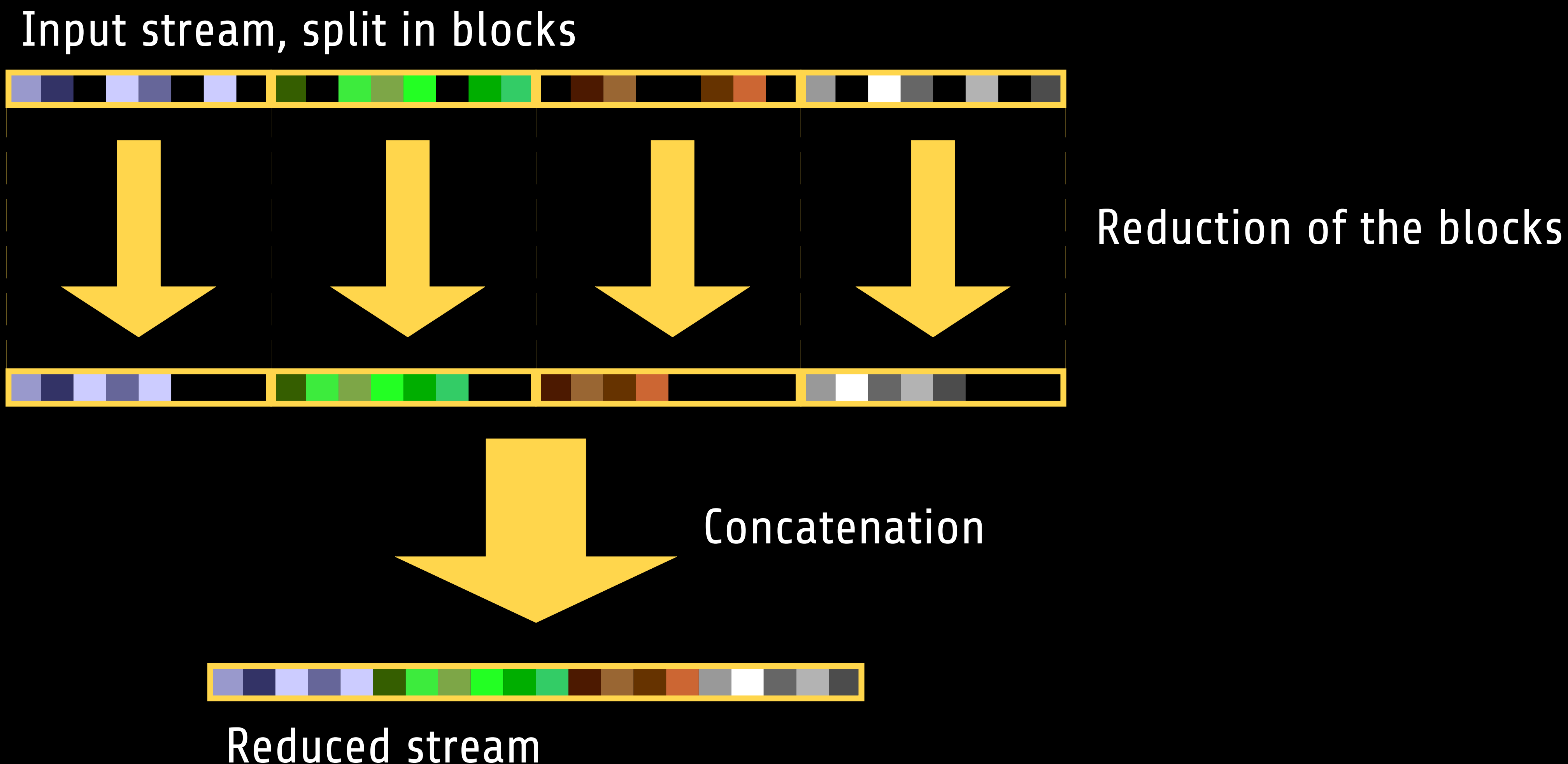
Talk Structure

- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

Talk Structure

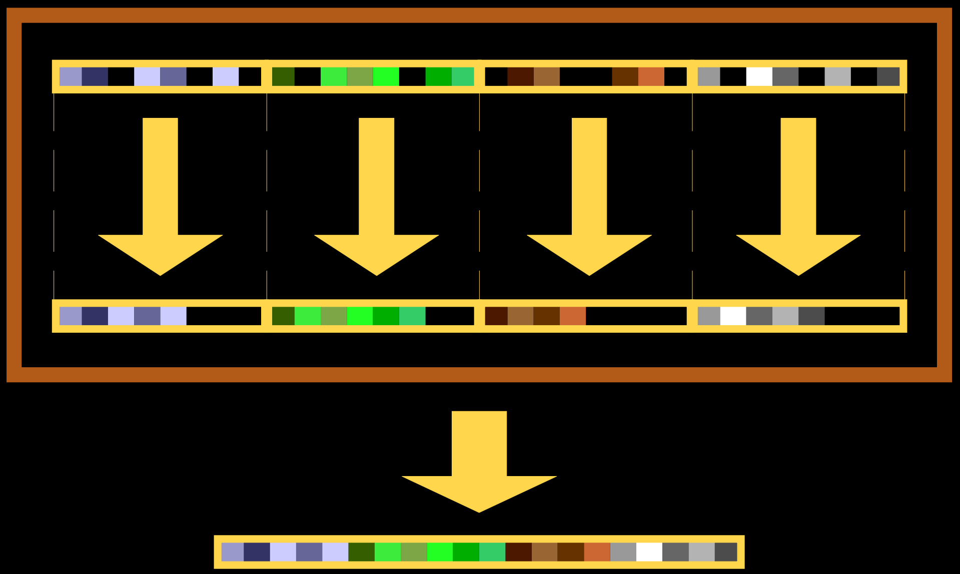
- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

Our approach



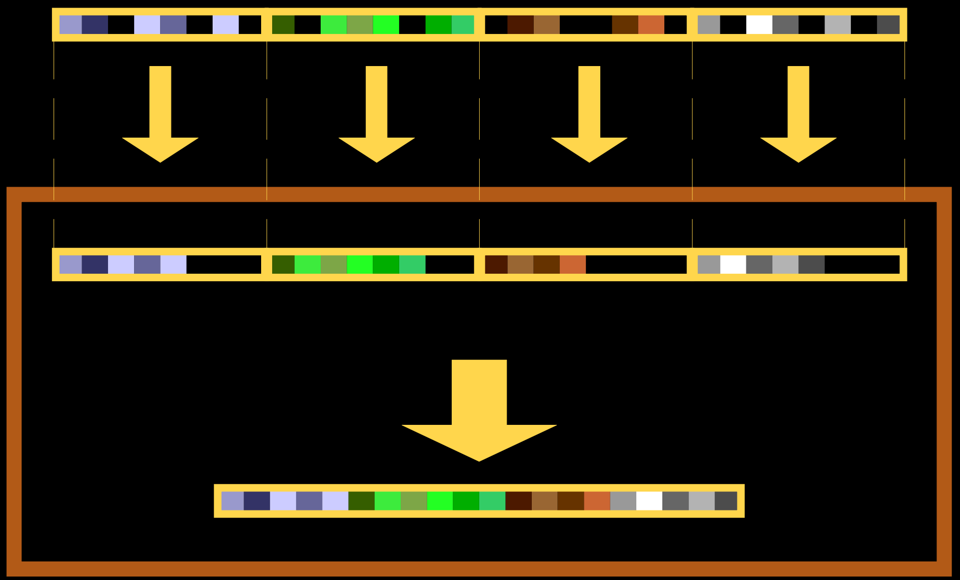
Reduction of the blocks

- In parallel
- Using previous works
  - Prefix sum scan
  - Dichotomic search
- Complexity
  - s: size of a block
  - One block: O(s log s)
  - n/s blocks: O(n log s)



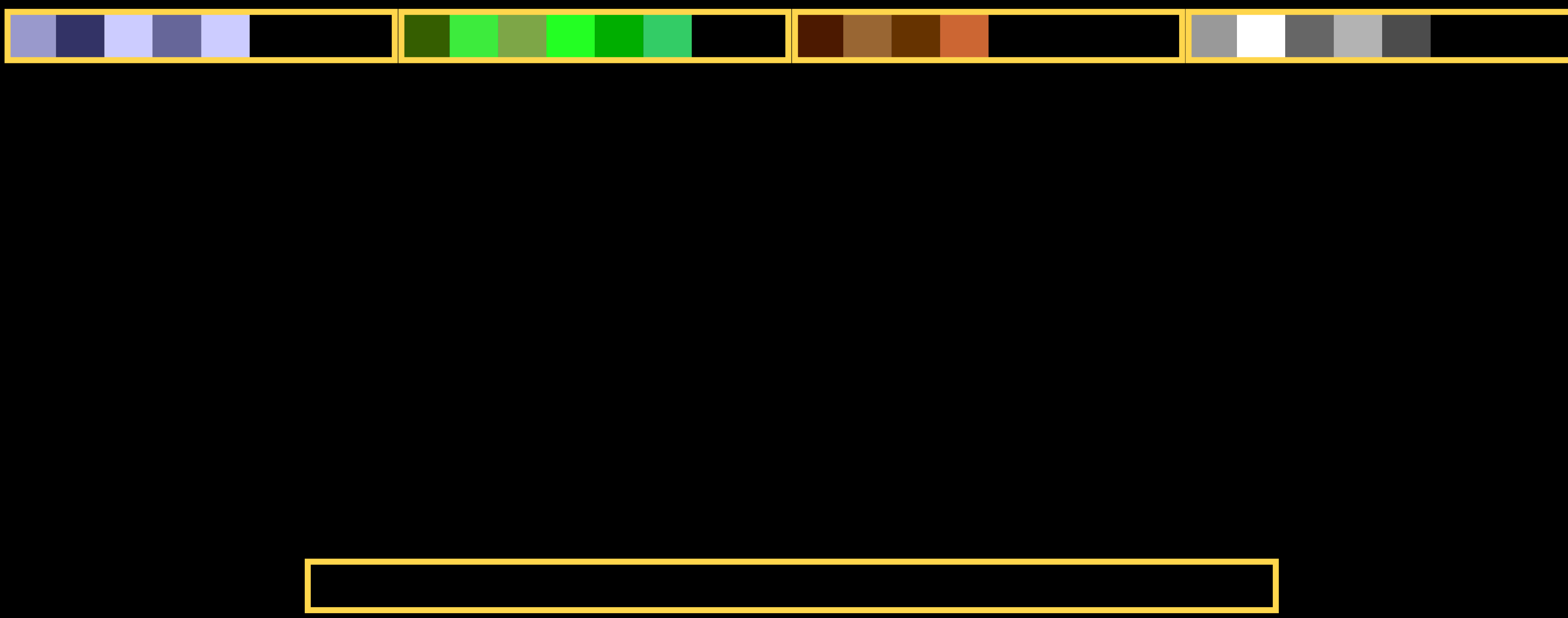
Concatenation of the blocks

- Prefix sum scan
  - Computes displacements of the blocks in parallel
- Line drawing
  - Segments extremities moved by scattering (vertex engine)
  - Other elements linearly interpolated (rasterization)
- Complexity: O(n)



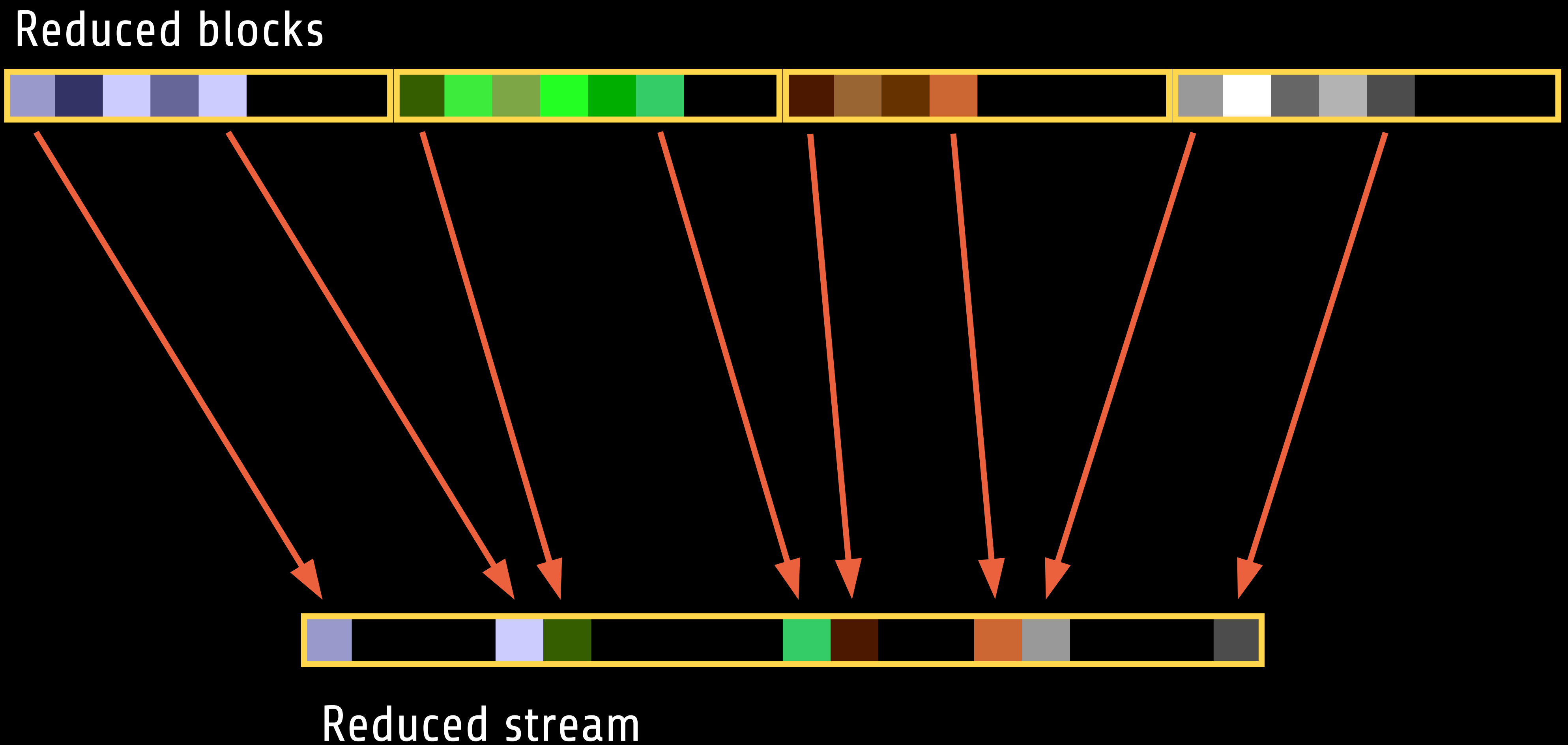
Concatenation of the blocks

Reduced blocks



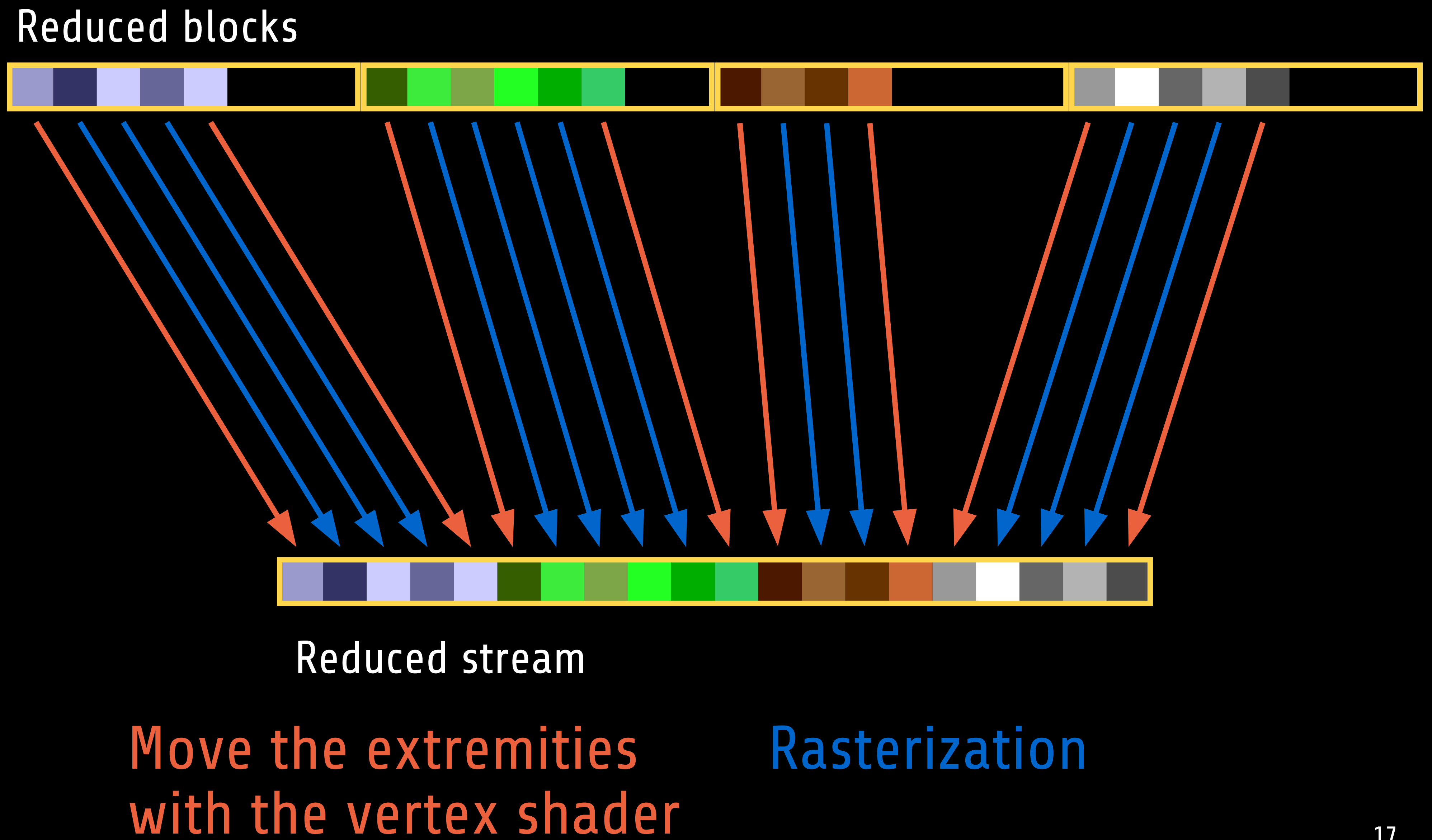
Reduced stream

Concatenation of the blocks



Move the extremities
with the vertex shader

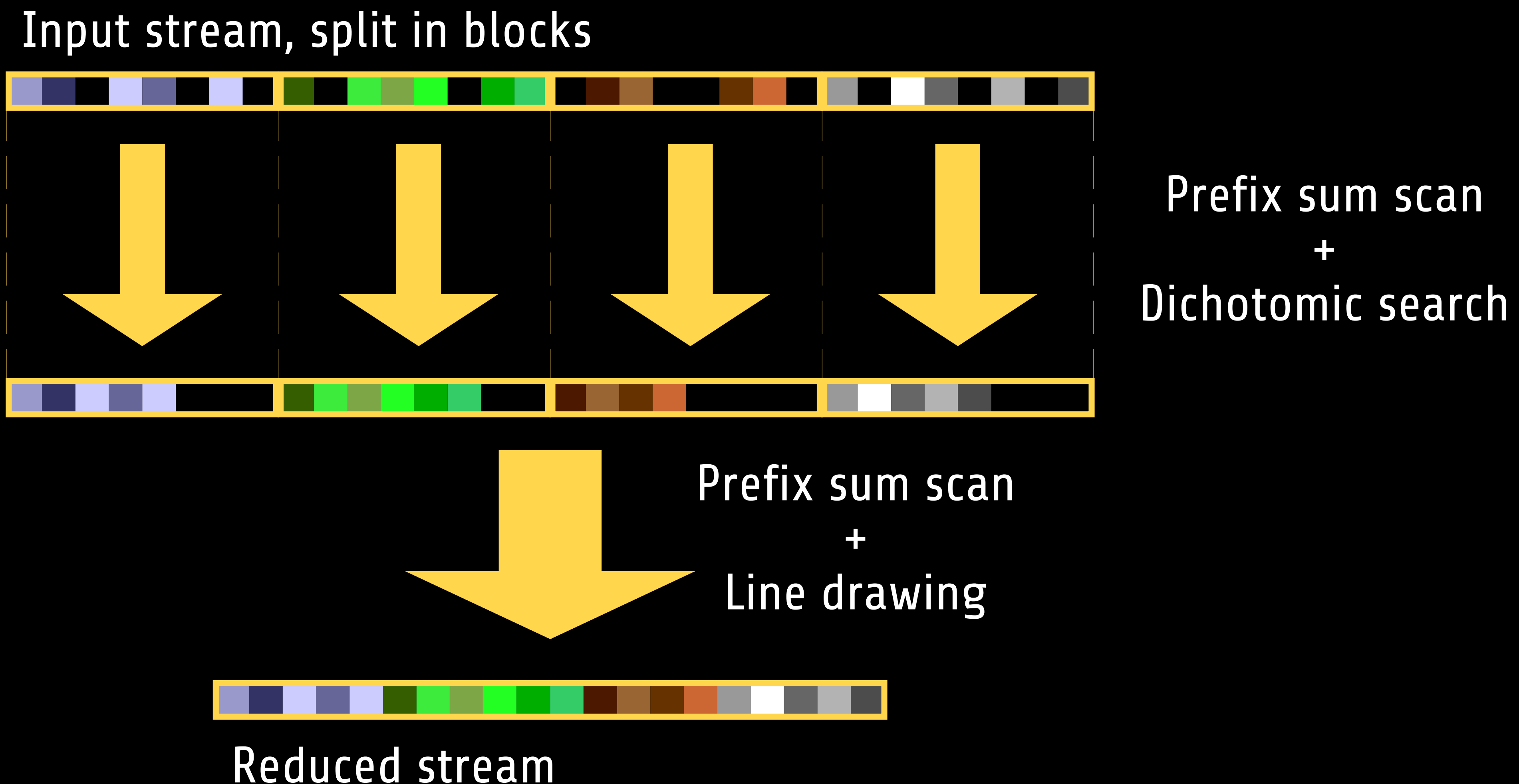
Concatenation of the blocks



Algortihmic complexity

- All previous works: $O(n \log n)$
- Our algorithm: $O(n \log s)$
  - s is the size of the blocks
  - s is a constant !

Overview



Input stream, split in blocks
Prefix sum scan
+
Dichotomic search
Prefix sum scan
+
Line drawing
Reduced stream

Why is it efficient ?

The key is block concatenation:

- Dichotomic search is avoided
- Vertex engine: scatter ... but lesser efficiency
  - Use it for a few elements (segment extremities)
  - Interpolate the other elements

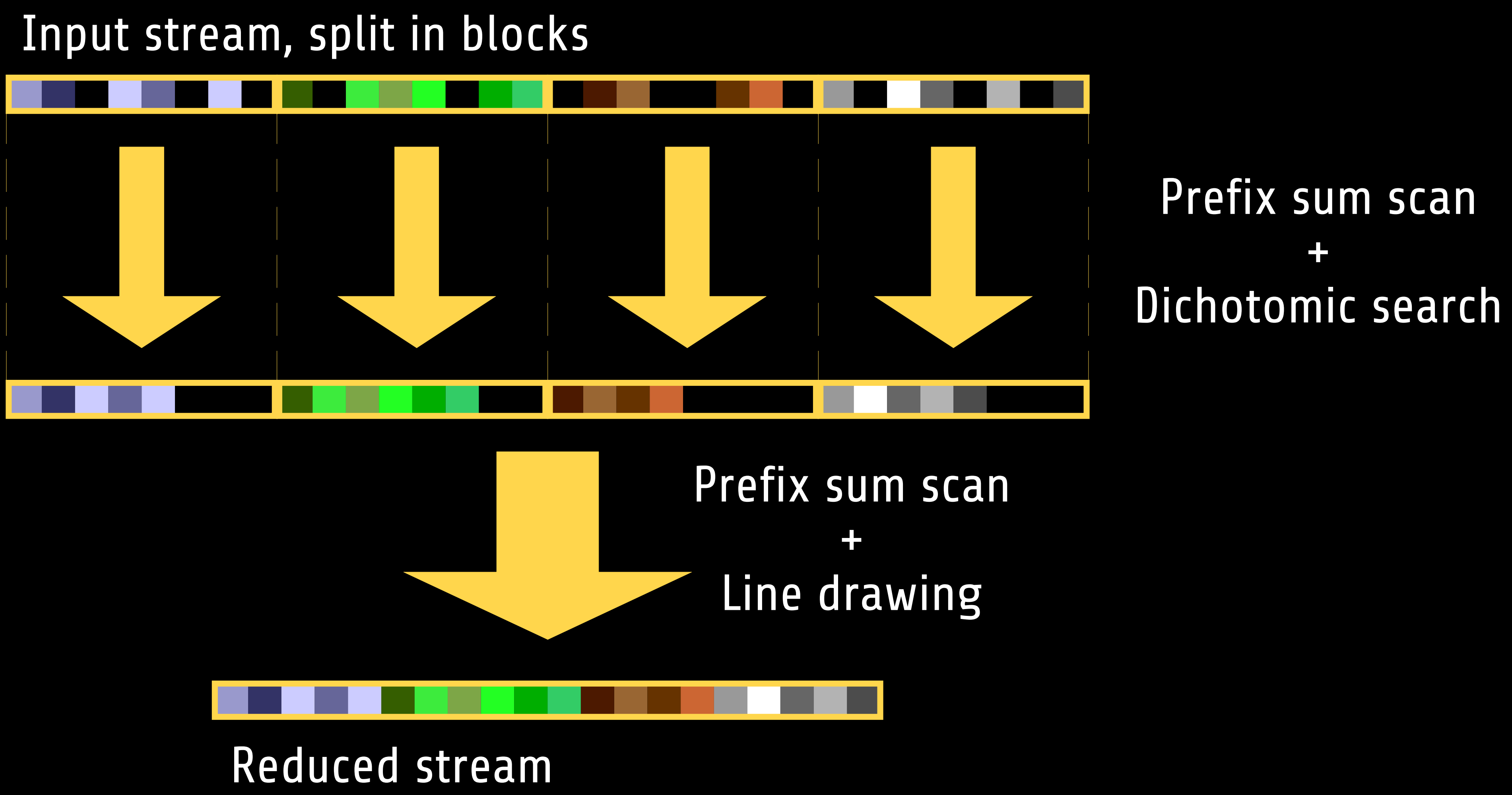
Talk Structure

- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

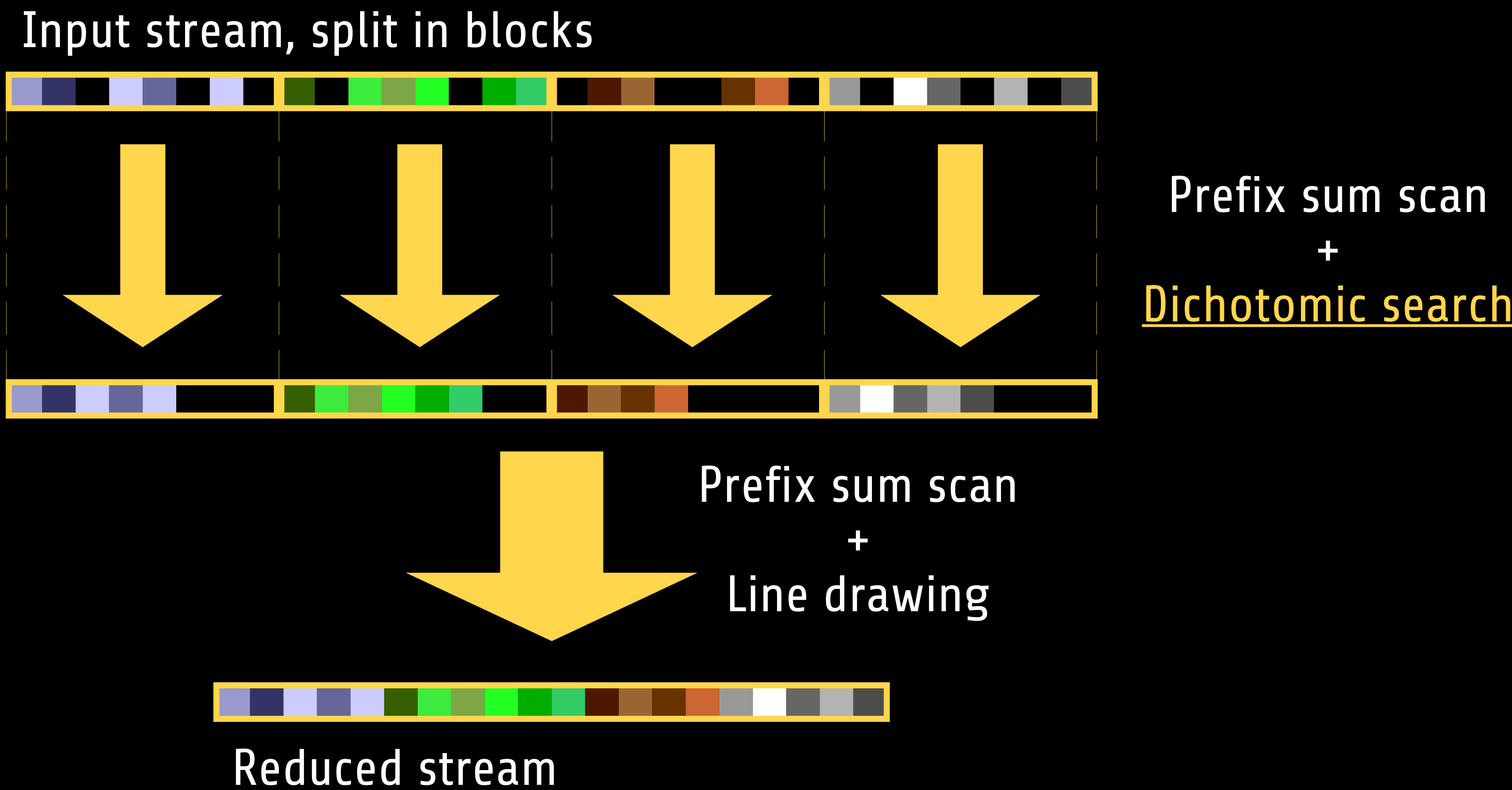
Talk Structure

- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

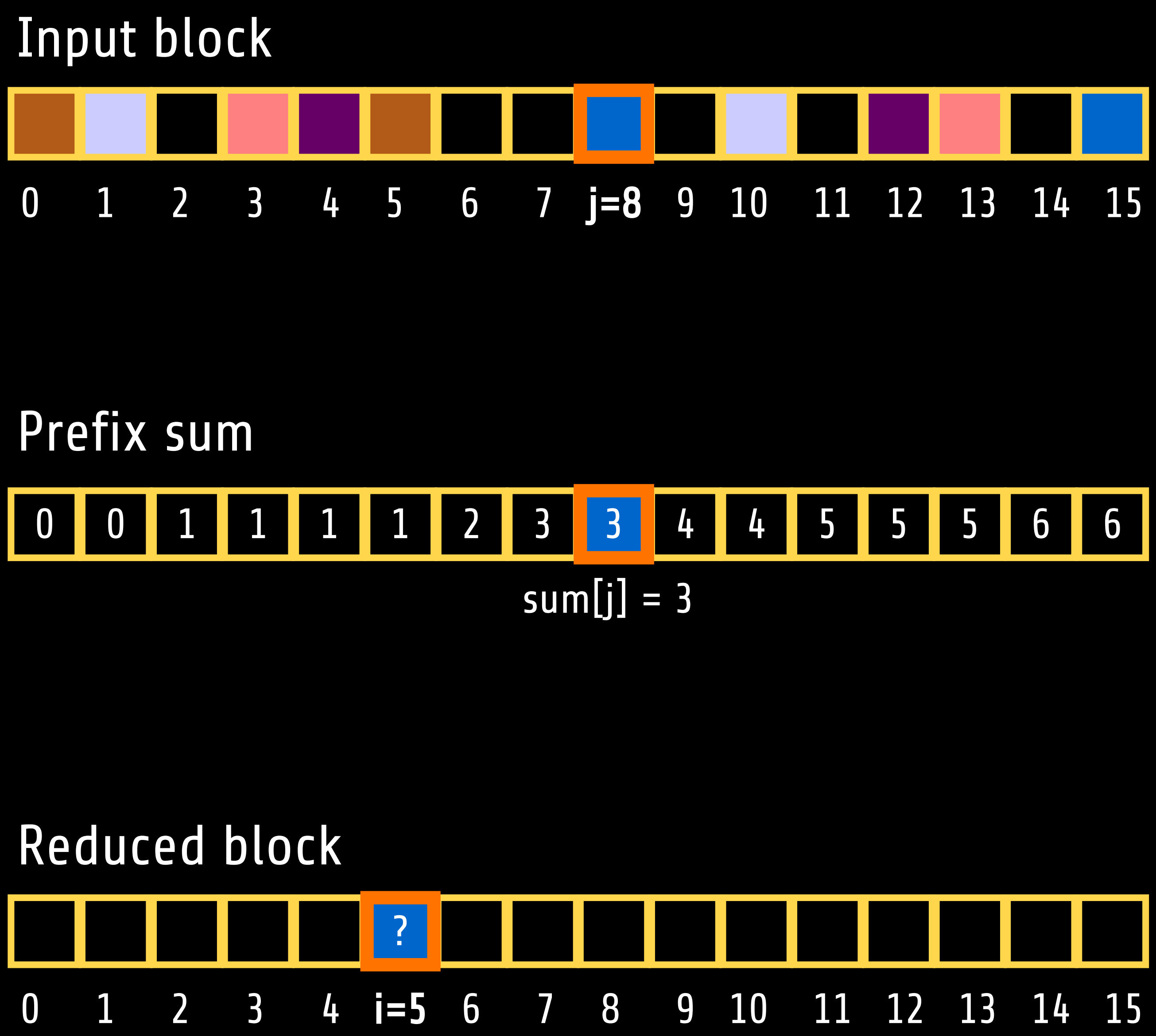
Overview


Input stream, split in blocks
Prefix sum scan
+
Dichotomic search
Prefix sum scan
+
Line drawing
Reduced stream

Overview


Input stream, split in blocks
Prefix sum scan
+
Dichotomic search
Prefix sum scan
+
Line drawing
Reduced stream

Dichotomic search details



Gather:
At output position i

Search j in input such as:
$i = j - sum[j]$

Search bounds:
$i + sum[i] \leq j \leq i + sum[15]$

Example:
$i = 5$
$6 \leq j \leq 11$
Search result $j = 8$

Dichotomic search pseudo-code

Search j_0 such as $i = j_0 - sum[j_0]$:

```
lowBound = i + sum[i]
upBound = i + sum[n-1]


if(upBound > n-1) discard


j = (lowBound + upBound)/2
found = j-sum[j]-i
```

```
while(found ≠ 0) {
    if (found < 0) lowBound = j
    else upBound = j

    j = (lowBound + upBound) / 2
    found = j-sum[j]-i
}
```

Dichotomic search improvement

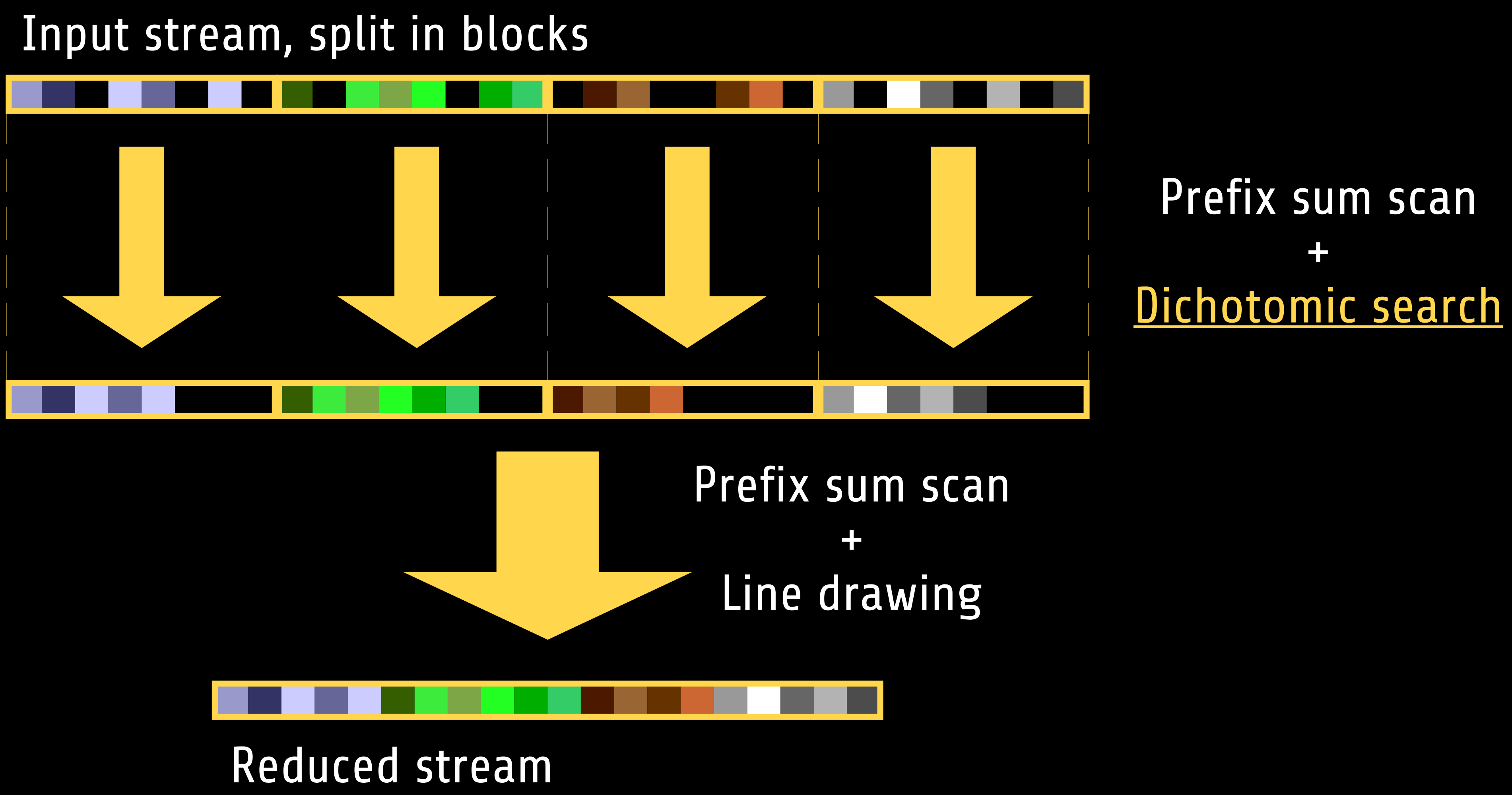
Search j_0 such as $i = j_0 - sum[j_0]$:

```
lowBound = i + sum[i]
upBound = i + sum[n-1]

if(upBound > n-1) discard

j = (lowBound + upBound)/2
found = j-sum[j]-i
```

```
while(found ≠ 0) {
    if (found < 0) lowBound = j - found
    else upBound = j - found

    j = (lowBound + upBound) / 2
    found = j-sum[j]-i
}
```

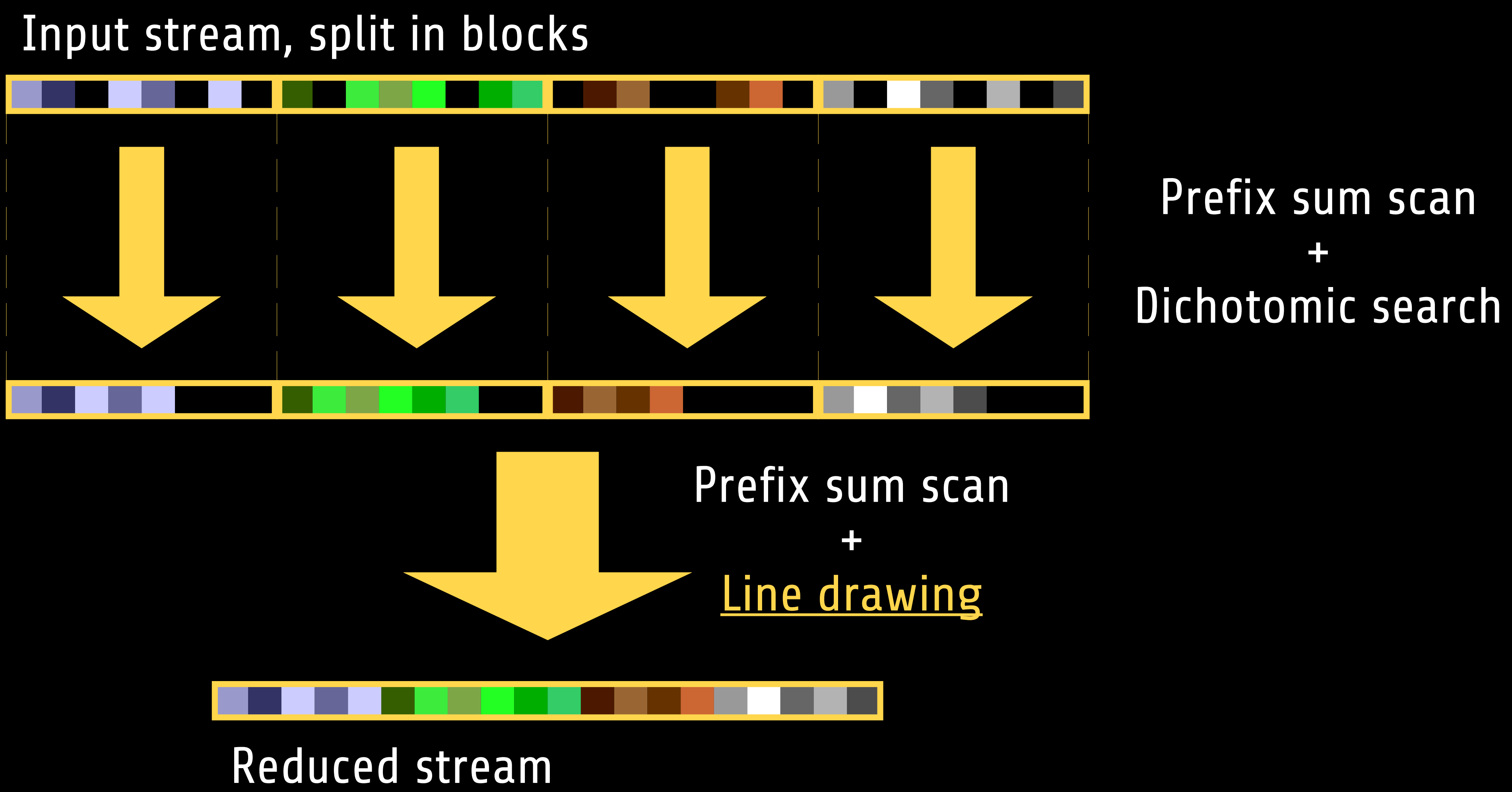
Because j – sum[j] is contracting!

Overview



Input stream, split in blocks
Prefix sum scan
+
Dichotomic search
Prefix sum scan
+
Line drawing
Reduced stream

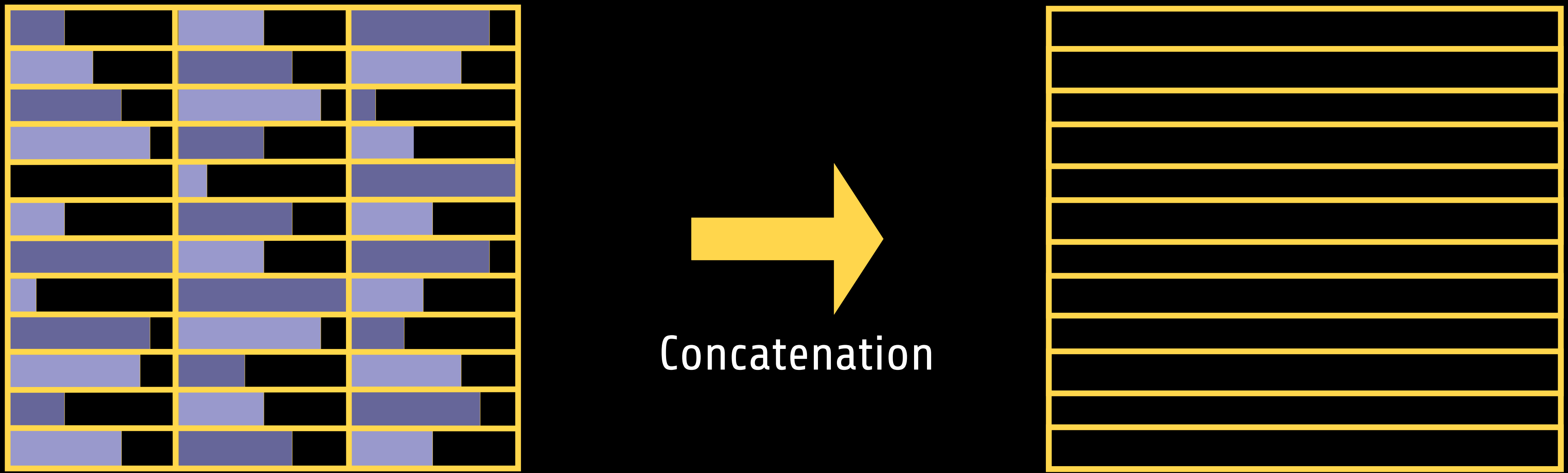
Overview



Input stream, split in blocks
Prefix sum scan
+
Dichotomic search
Prefix sum scan
+
Line drawing
Reduced stream

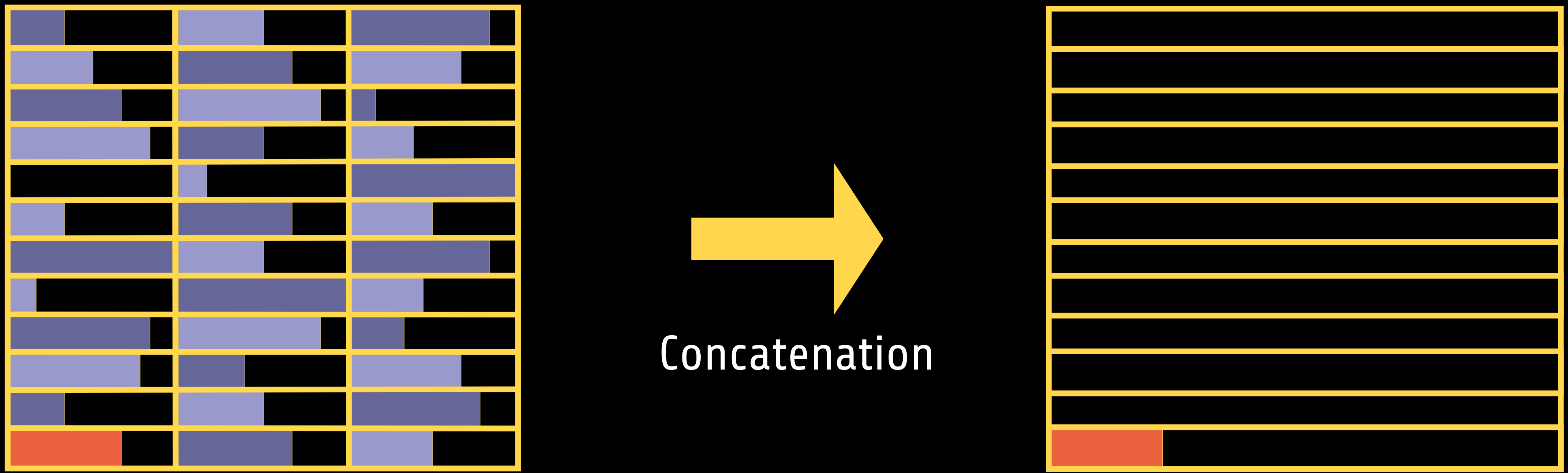
Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary



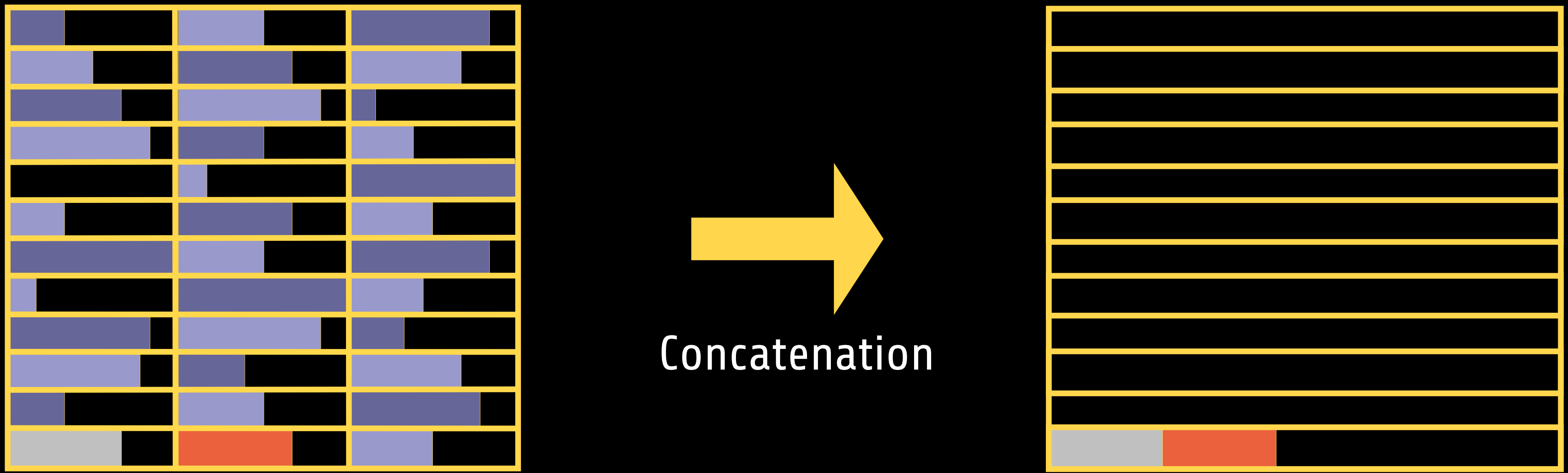
Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary



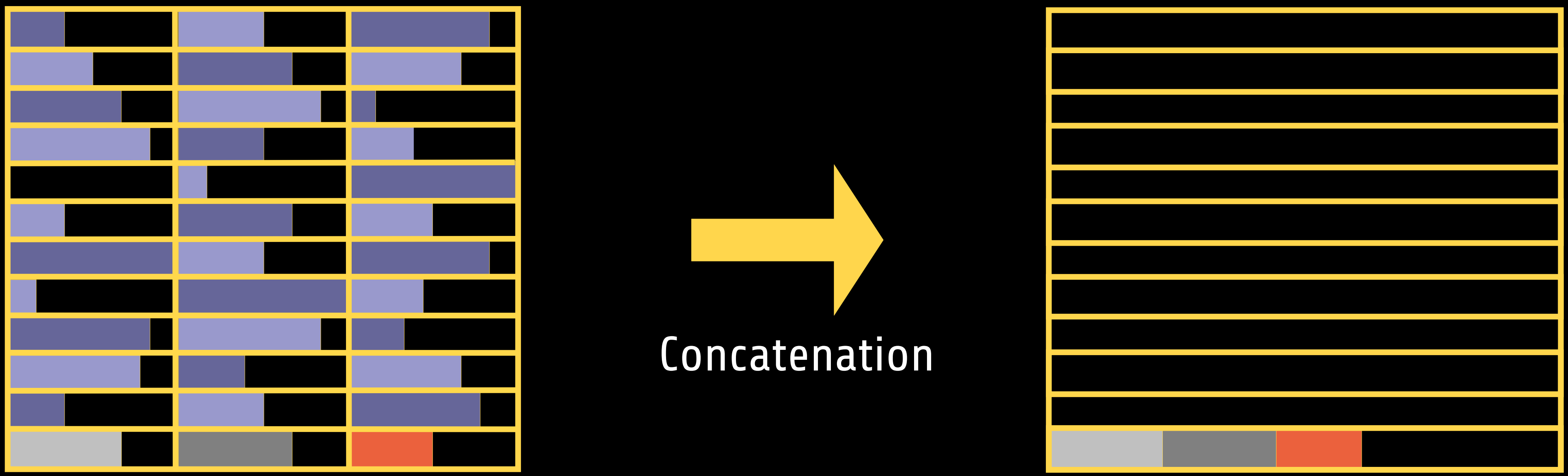
Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary



Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary



Lines wrapping

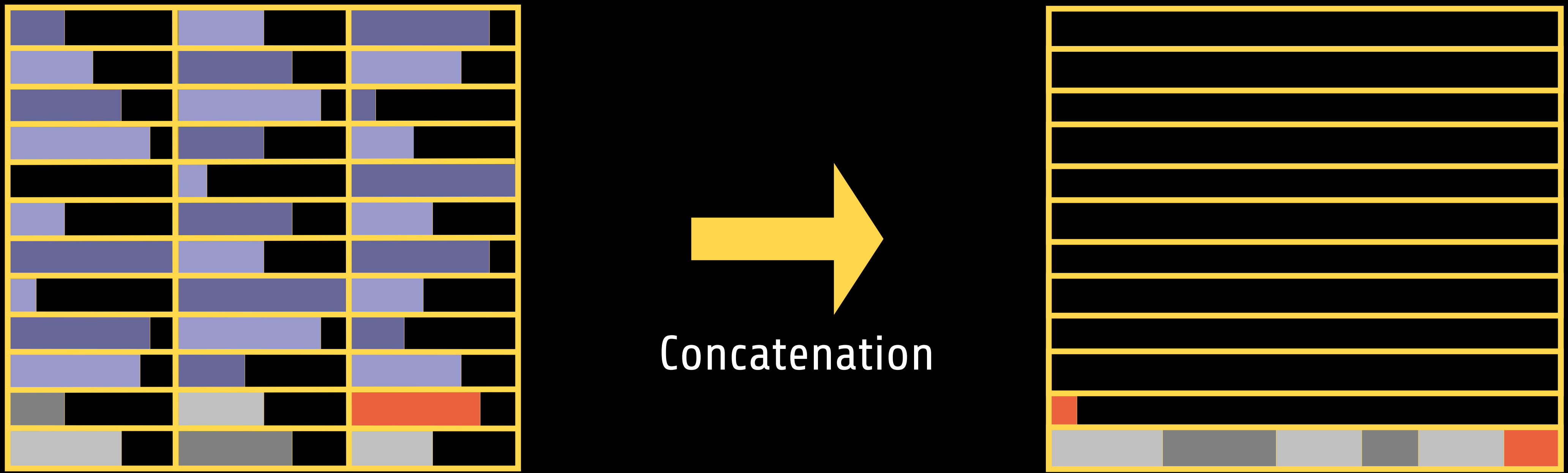
- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary

Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary

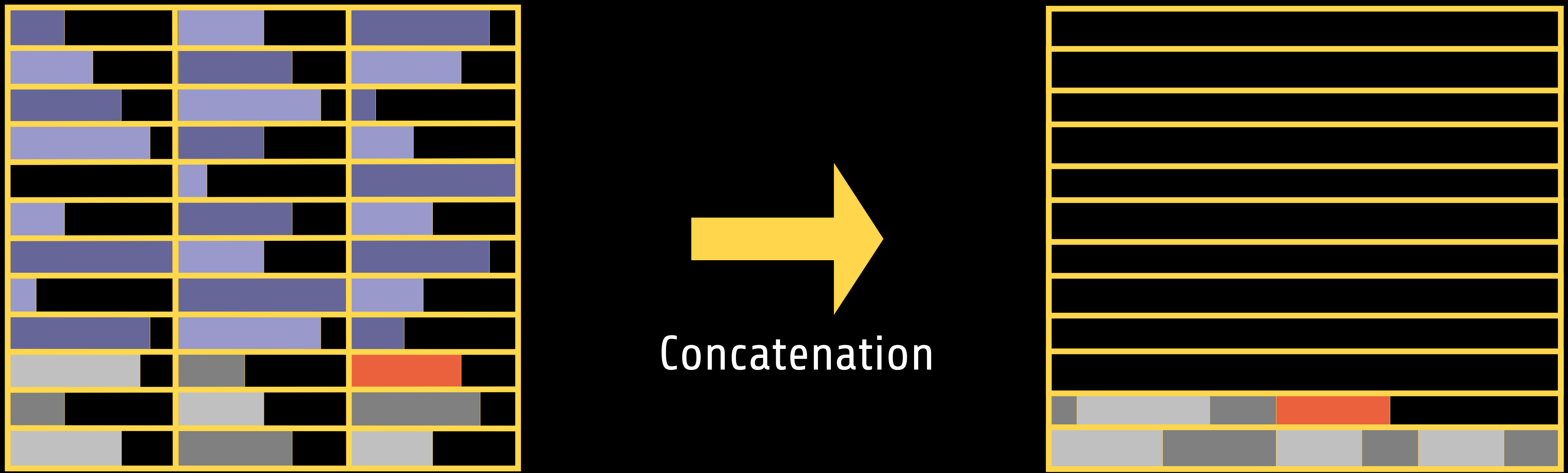
Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary



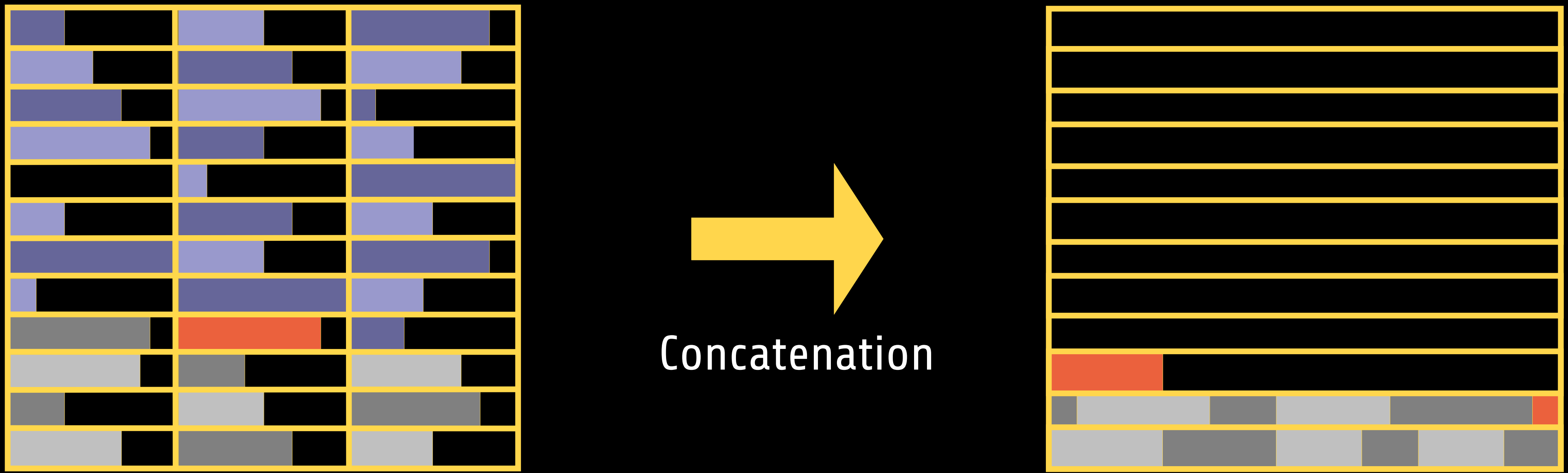
Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary



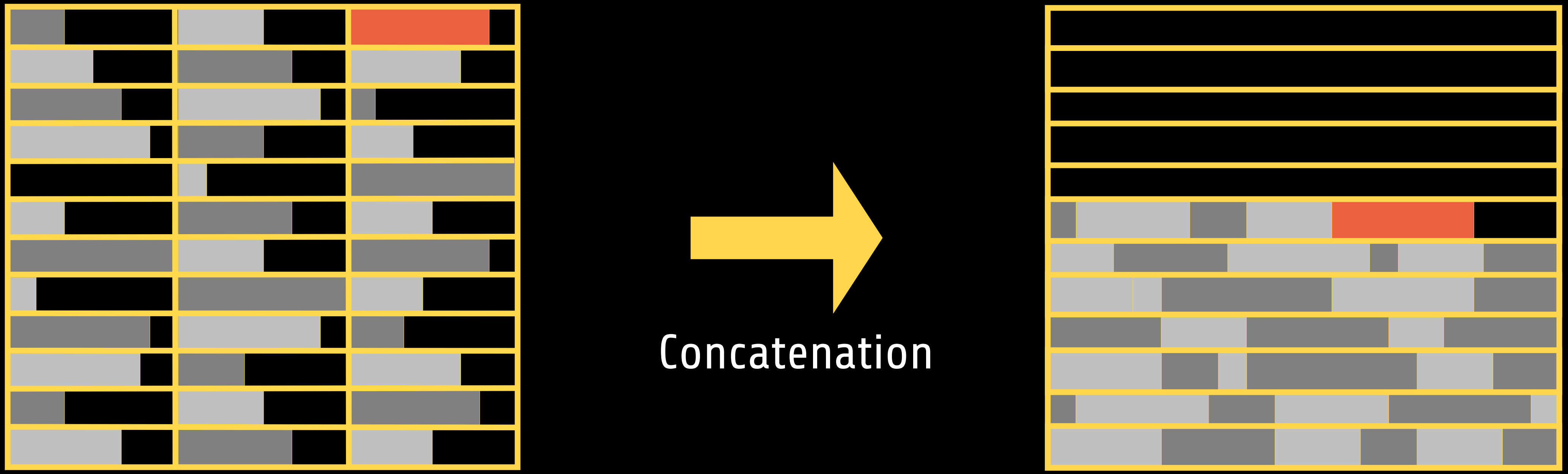
Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary



Lines wrapping

- We use 2D textures: wrap line segments
  - Split all segments in two
    - Or
  - Use geometry engine to split only when necessary



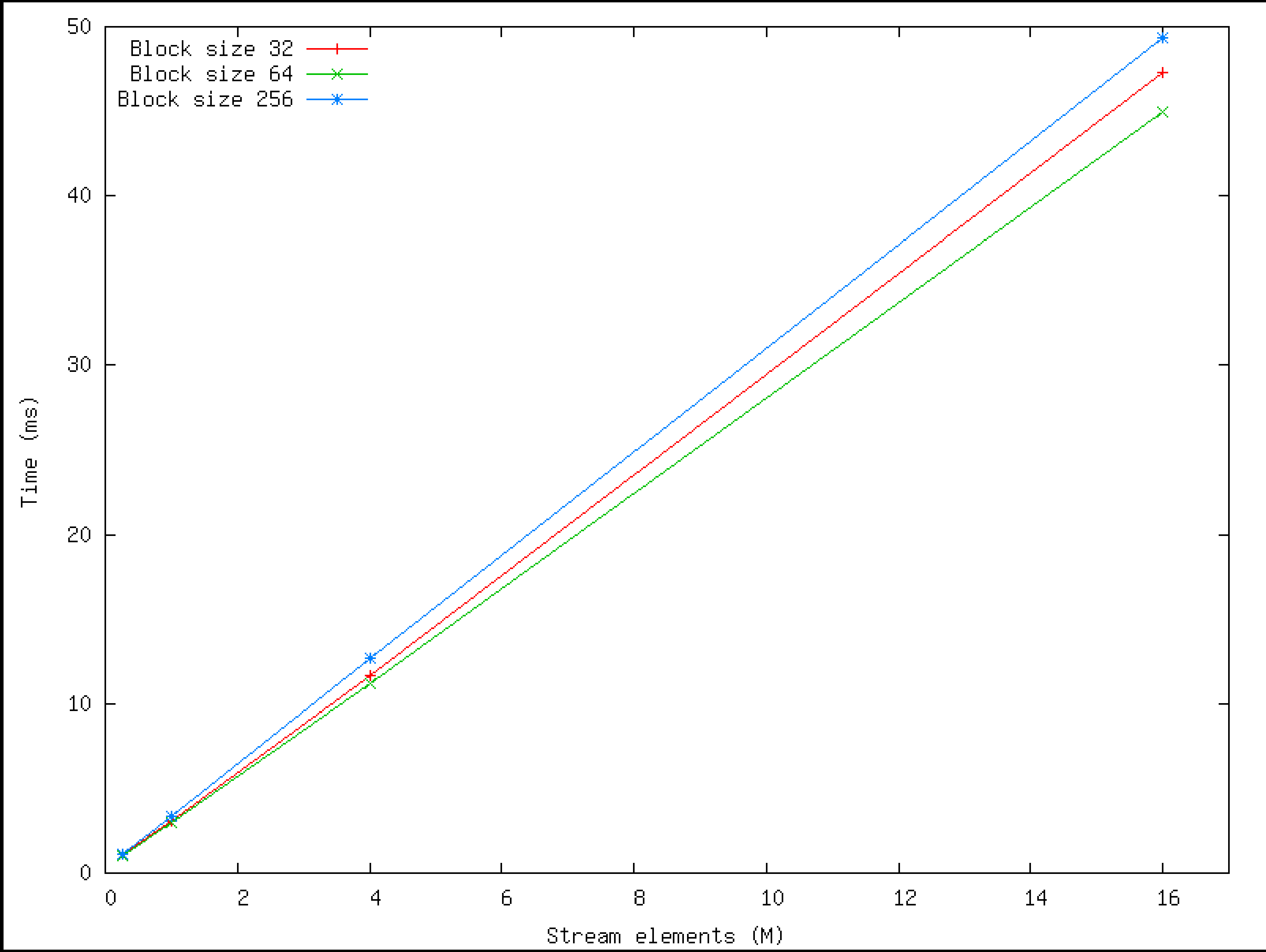
Talk Structure

- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

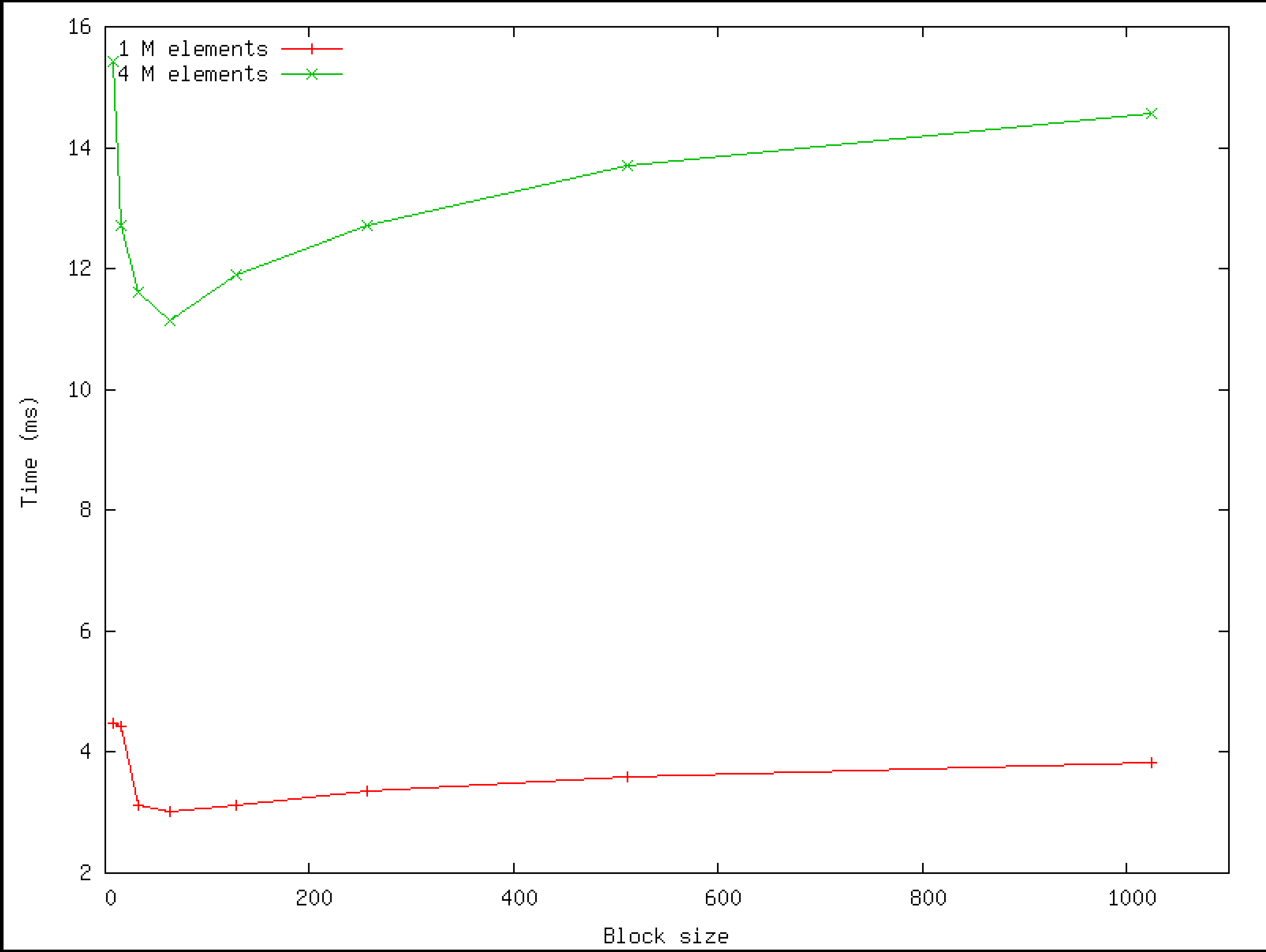
Talk Structure

- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

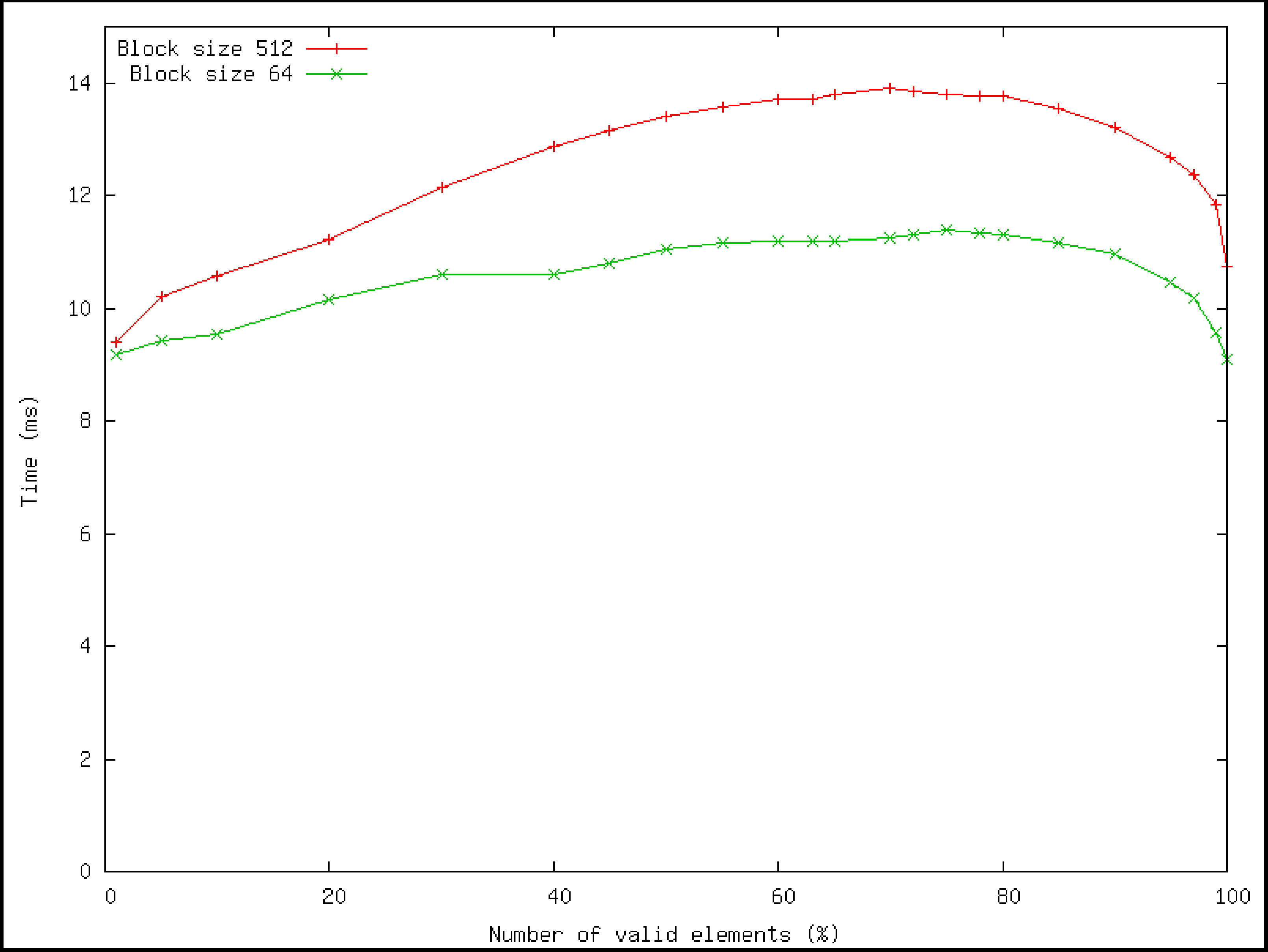
Behavior: linear complexity



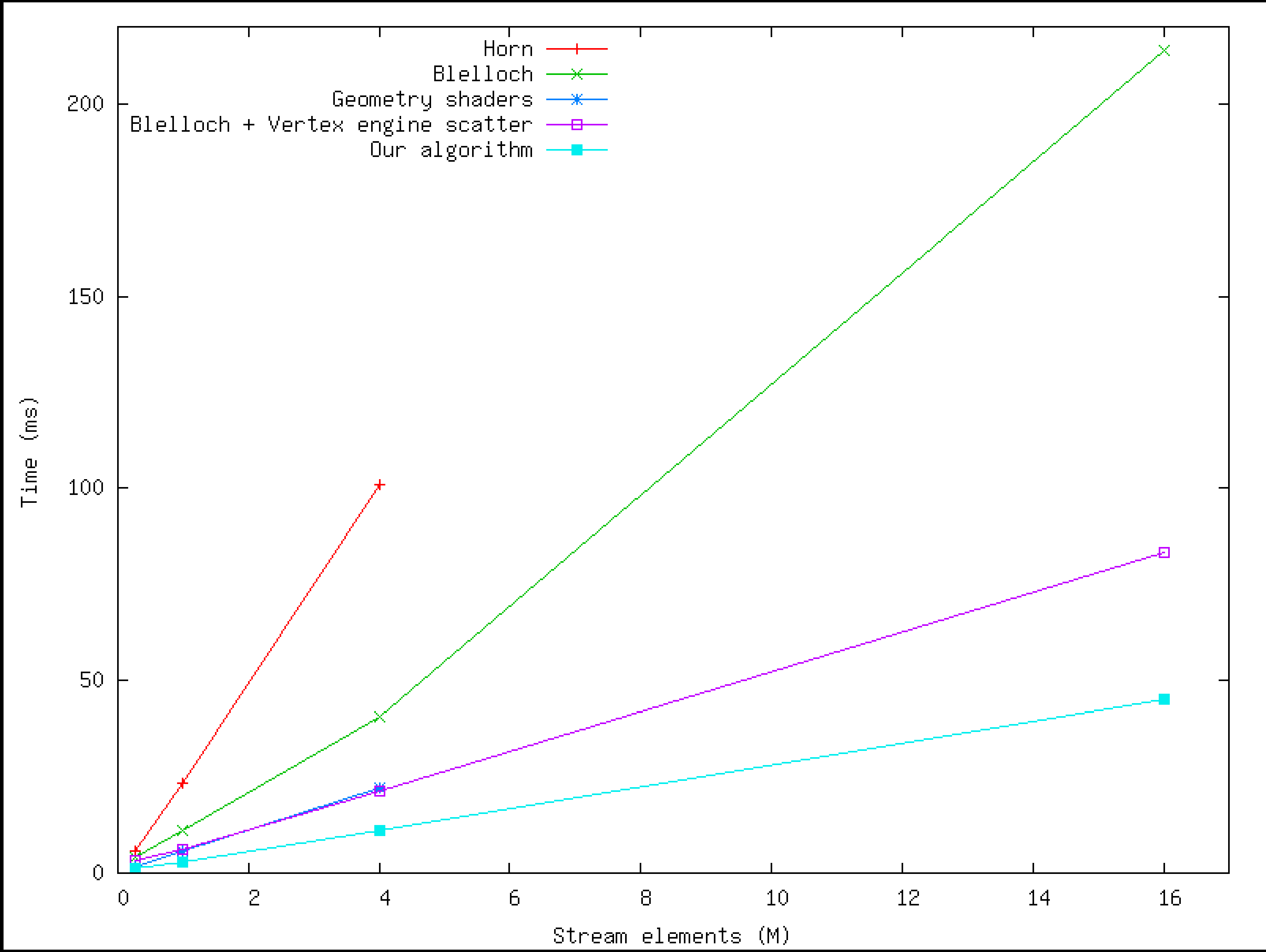
Behavior: block size



Behavior: fill ratio



Comparison with previous works



Talk Structure

- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

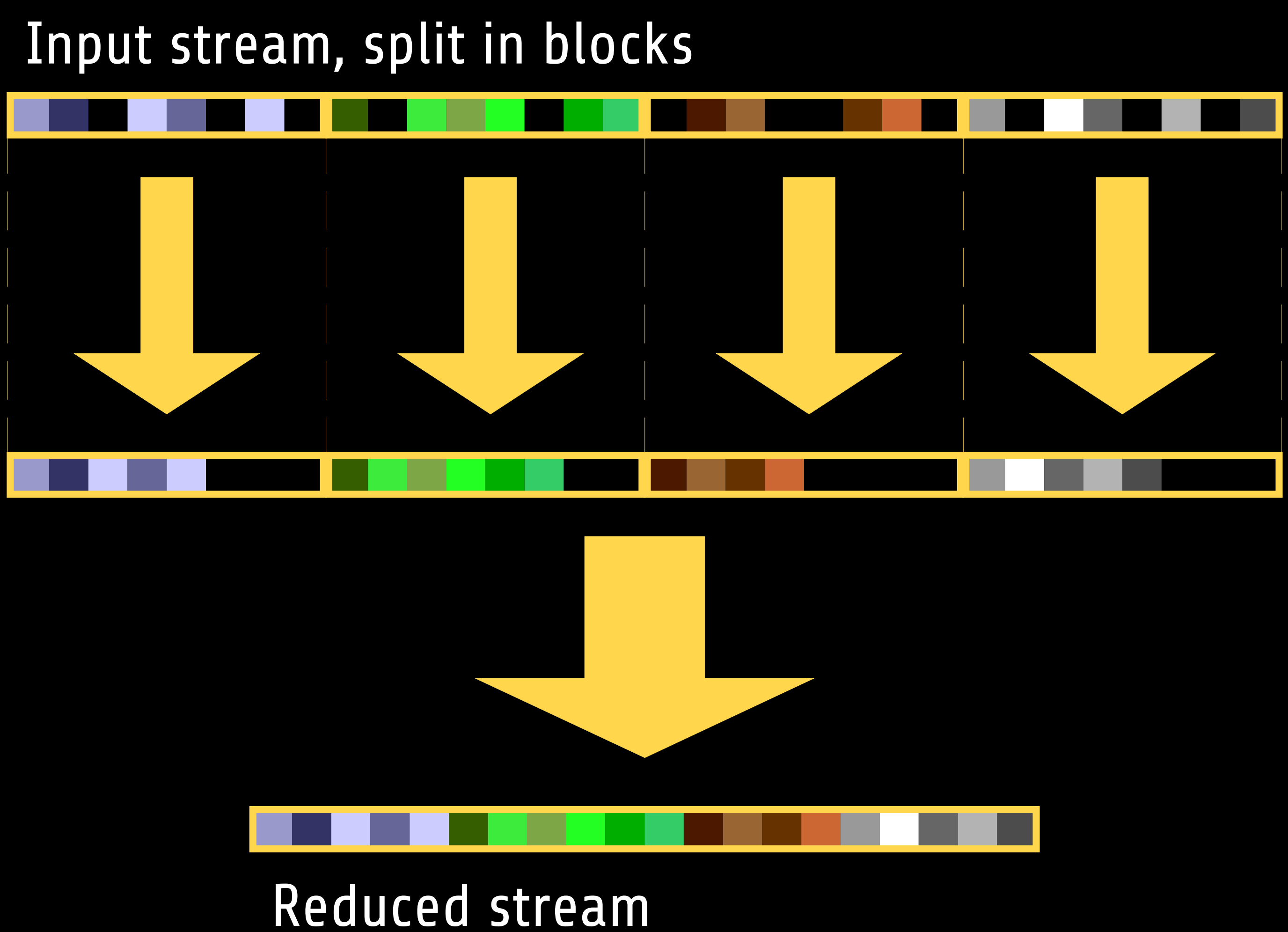
Talk Structure

- Previous Works
- Algorithm Overview
- Details and Implementation
- Results
- Future Works & Conclusion

Scatter ? (future work)

- Scatter available in CUDA
- Possible improvements

Scatter ? (future work)



Reduction of the blocks:

- without scatter:
  sum scan + search
  O(n log s)
- with scatter:
  sequential algo
  (loop over the block)
  O(n)

Concatenation:

- Simpler
- No wrapping

Scatter ? (future work)

- Overall complexity: O(n)
- … but other techniques in O(n)
  - Sum scan (Harris et al. or Sengupta et al.) + scatter
- Future work: tests with CUDA
  - Expected speed up ≥ 2.5

Conclusion

- Orthogonal to previous works:
  - We don't compete with them, we use them !
- Better asymptotic complexity
  - O(n) Vs O(n log n)
- Significant speed up
- Does not require scatter

Thank you