https://hao-ai-lab.github.io/dsc204a-w24/

DSC 204A: Scalable Data Systems
Winter 2024

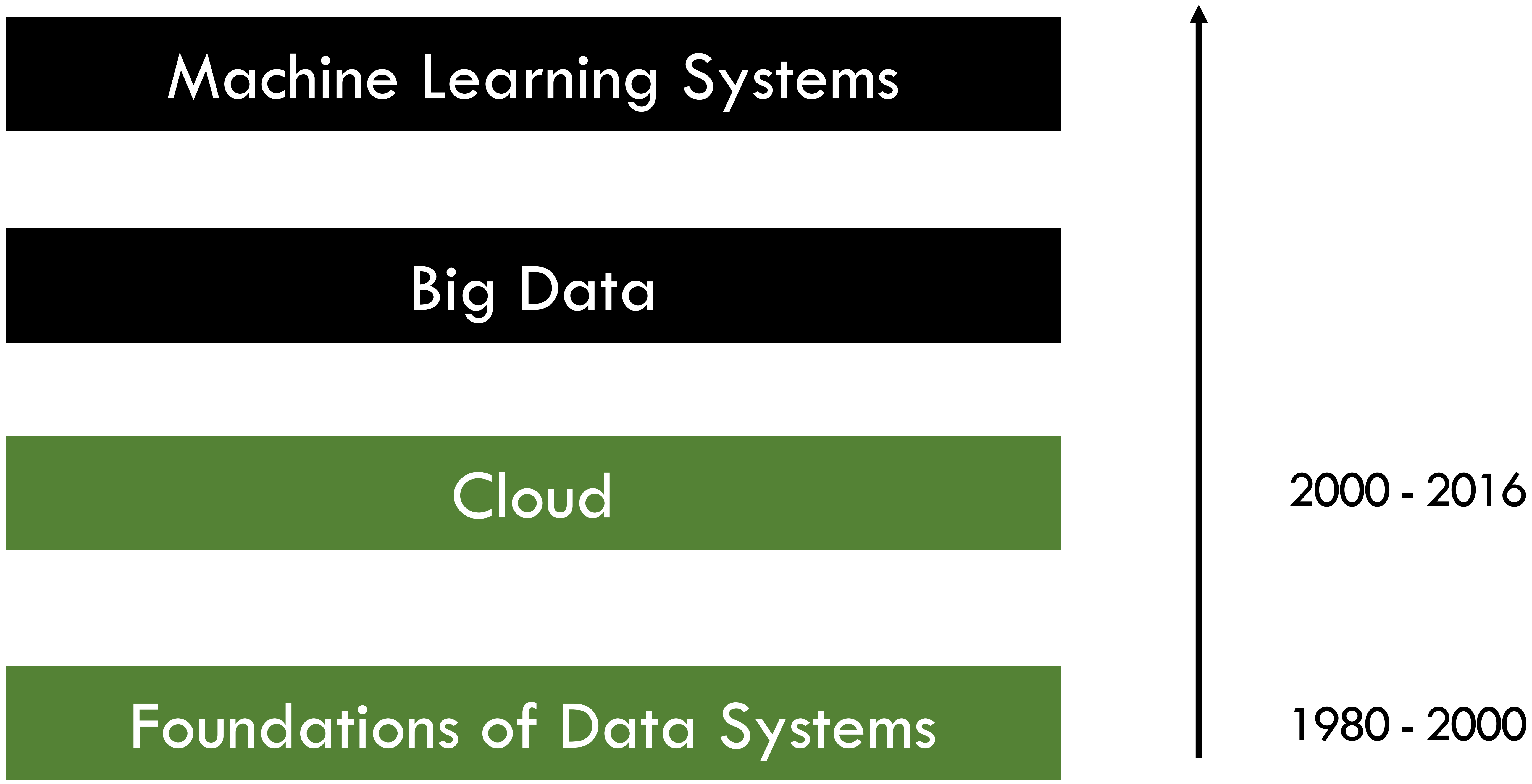
Machine Learning Systems

Big Data

Cloud

Foundations of Data Systems

Where We Are


Machine Learning Systems
Big Data
Cloud
Foundations of Data Systems
2000 - 2016
1980 - 2000

Recap: Networking

- Q1: True _ False x Protocols specify the implementation
- Q2: True _ False x Congestion control takes care of the sender not overflowing the receiver
- Q3: True x False _ A random access protocol is efficient at low utilization
- Q4: True _ False x At the data link layer, hosts are identified by IP addresses
- Q5: True x False _ The physical layer is concerned with sending and receiving bits

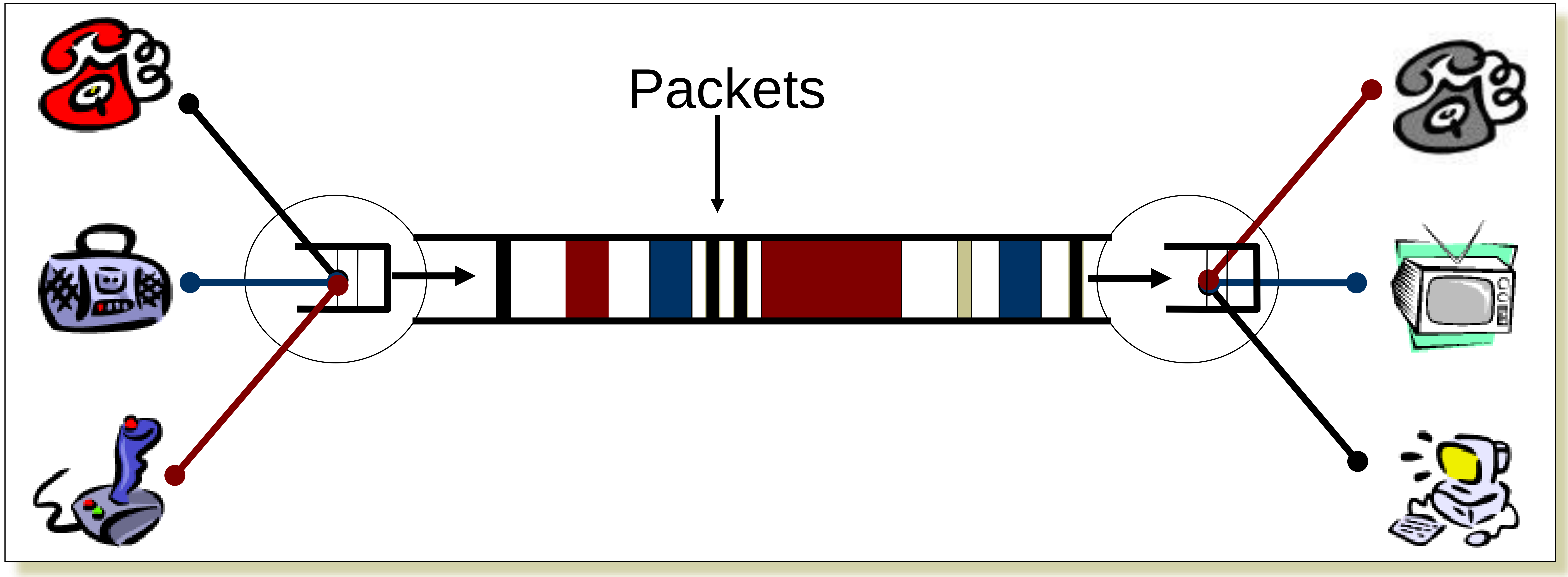
Recap: Networking

- Q1: True _ False X̲ Layering improves application performance
- Q2: True X̲ False _ Routers forward a packet based on its destination address
- Q3: True _ False X̲ “Best Effort” packet delivery ensures that packets are delivered in order
- Q4: True _ False X̲ Port numbers belong to network layer

Today's topic

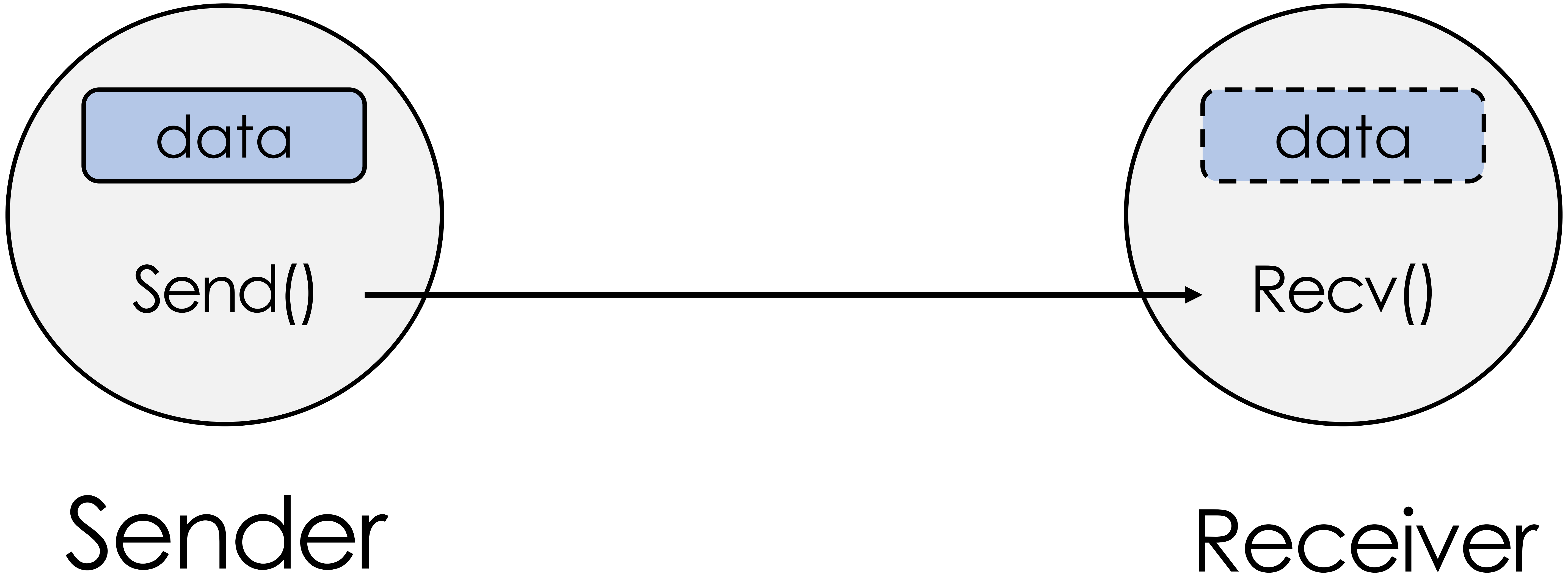
- Network Basics
- Layering and protocols
- **Collective communication**

Communication

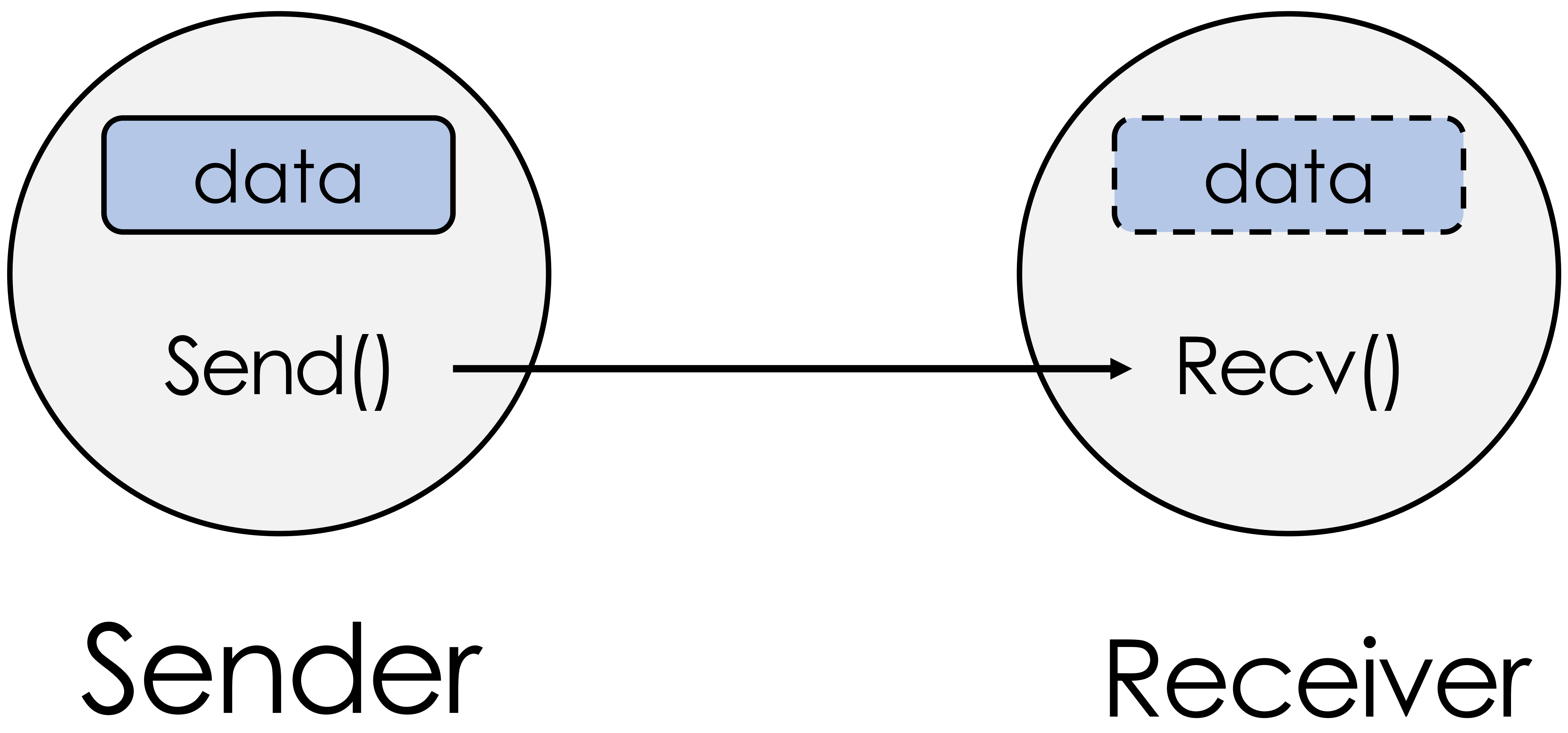


Communication: Point-to-point communication

1. Establish TCP connection
2. Application sends data
3. Data goes down 5 layers, through the network, and arrives in receiver



Program P2P communication: Very Simple in Ray




```python
def send(array: np.array):
    # Running on sender processs
    ref = ray.put(array)
    return ref

def receive(ref):
    # Running on receiver process
    array = ray.get(ref)
    print(array)
```

Case study: Gradient update in DL

Gradient / backward computation

$$\theta^{(t)} = \theta^{(t-1)} + \boxed{\varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})}$$

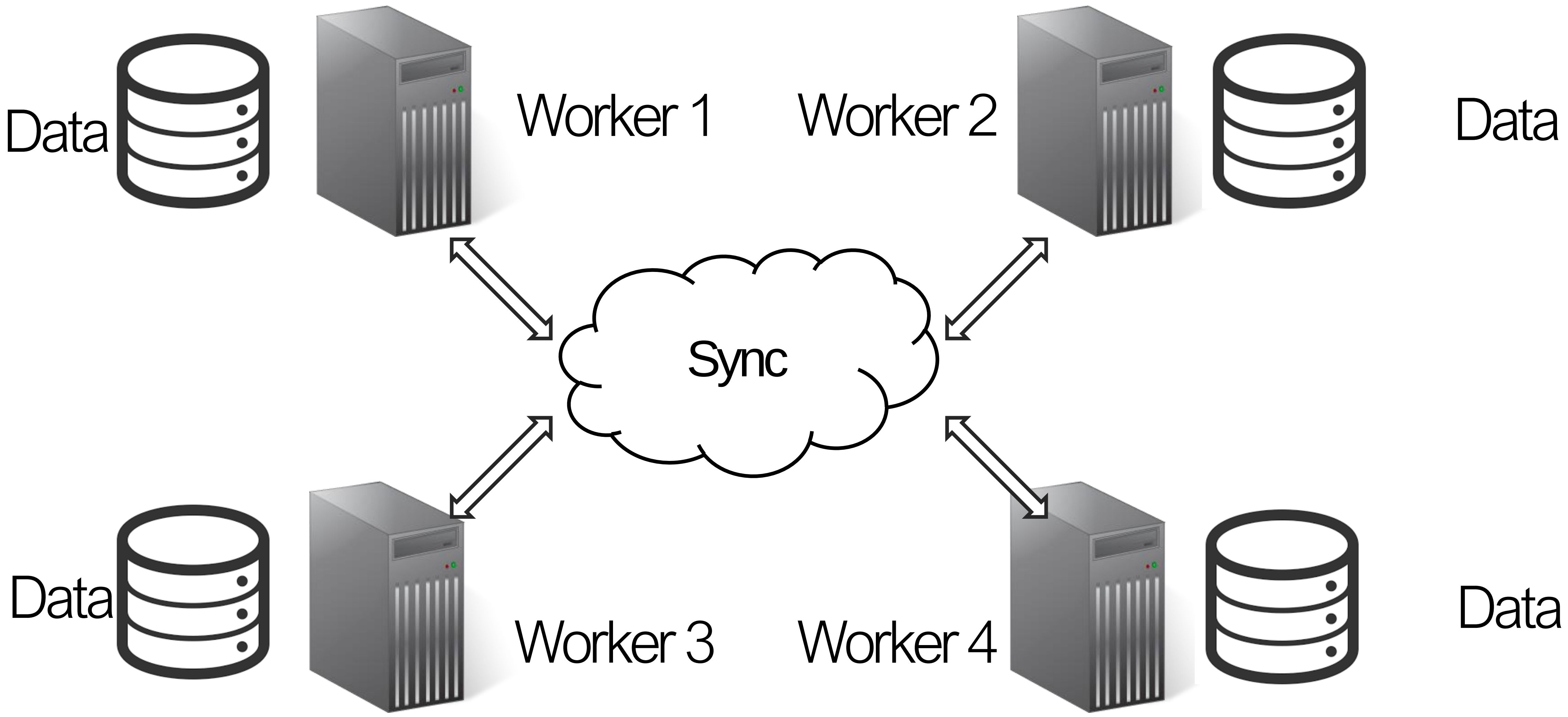
objective

data

What If Data is super Big?

What if Data is Super Big?

Big Data



Case study: Gradient update

Gradient / backward computation

$$\theta^{(t)} = \theta^{(t-1)} + \boxed{\varepsilon \cdot \nabla_{\mathcal{L}}(\theta^{(t-1)}, D^{(t)})}$$

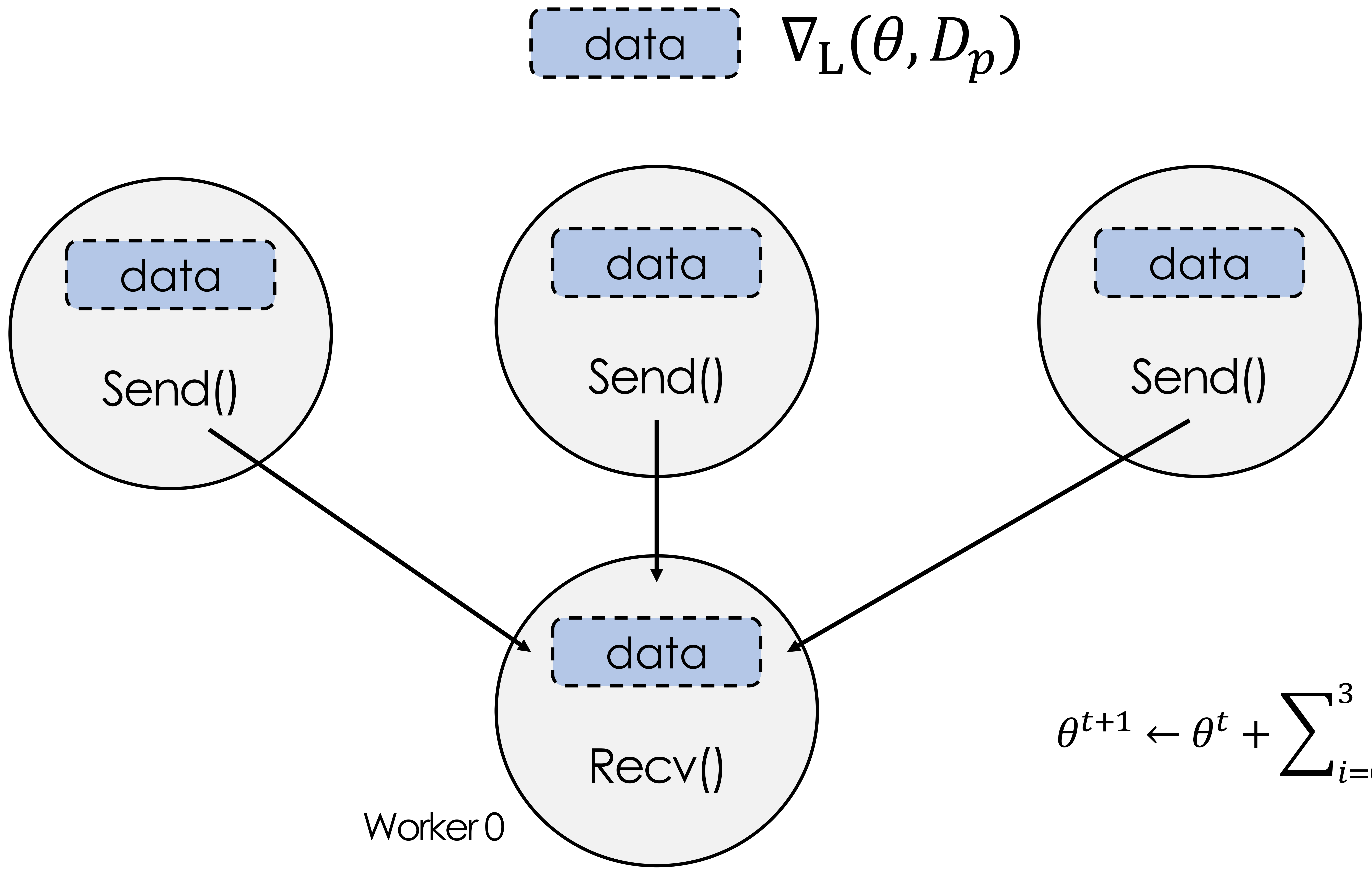
objective

data

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

How to perform this sum?

Collective Primitive: Reduce



$$\theta^{t+1} \leftarrow \theta^t + \sum_{i=0}^{3} \nabla\theta_i^t$$

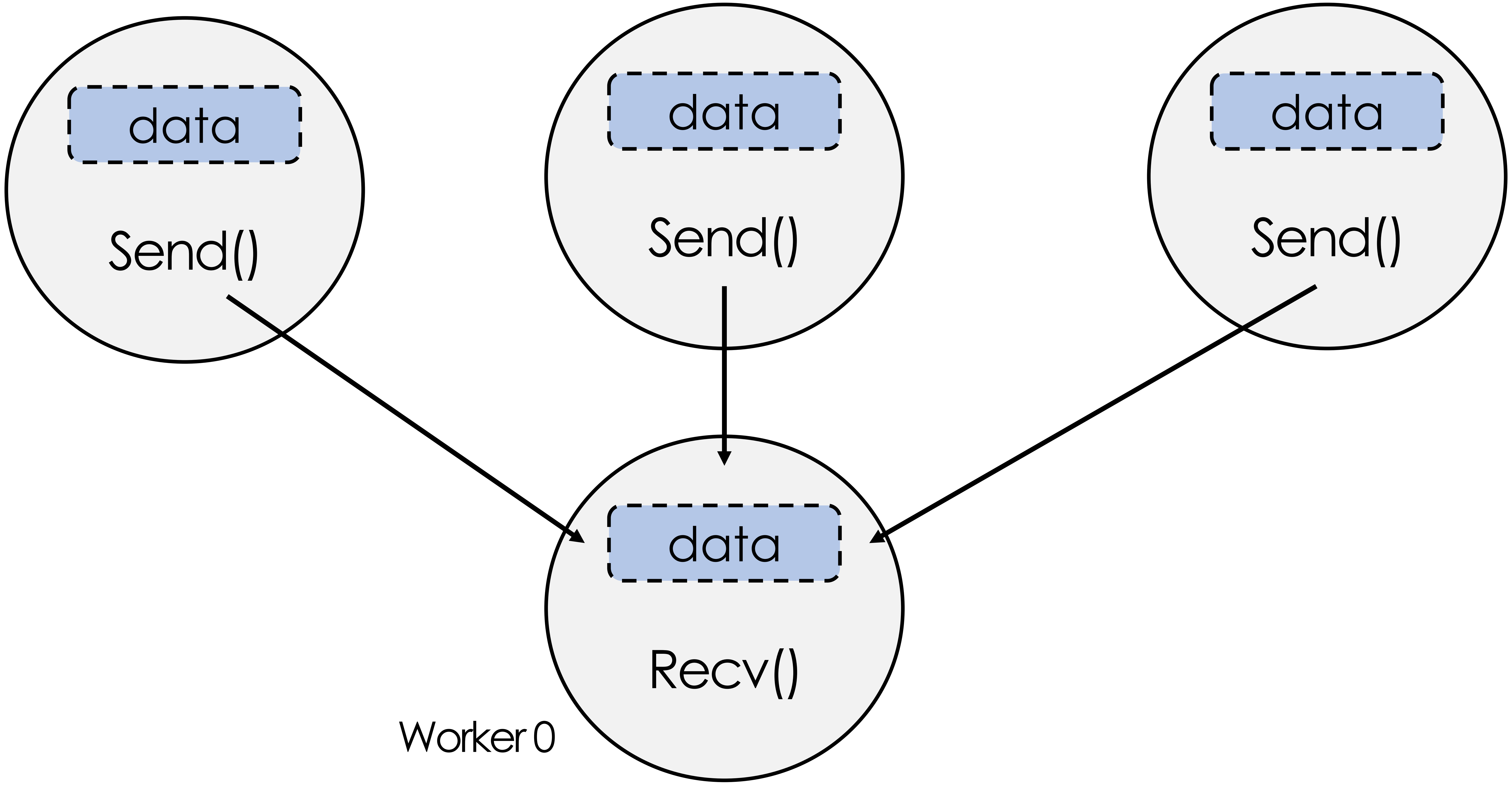
Program This?

```python
@ray.remote(num_gpus=1)
class GPUWorker:
  def __init__(self):
    self.gradients = cupy.ones((10,), dtype=cupy.float32)

  def put_gradients(self):
    return ray.put(self.gradients)

  def reduce_gradients(self, grad_id_refs):
    grad_ids = ray.get(grad_id_refs)
    reduced_result = cupy.ones((10,), dtype=float32)
    for grad_id in grad_ids:
      array = ray.get(grad_id)
      reduced_result += array
    result_id = ray.put(reduced_result)
    return result_id
```

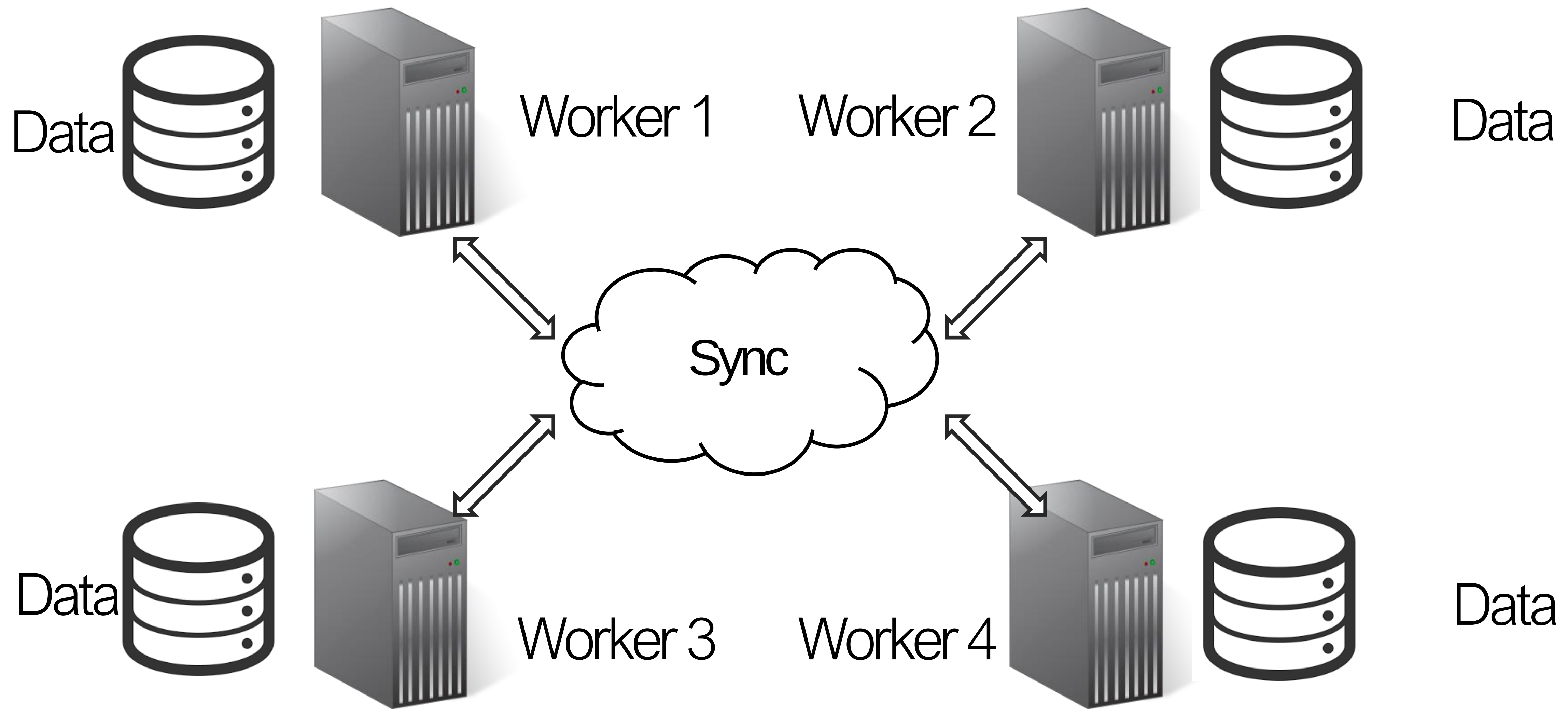
```python
# Allreduce the gradients using Ray APIs
# Let all workers to put their gradients into the Ray object store.
gradient_ids = [worker.put_gradients.remote() for worker in workers]
ray.wait(graident_ids, num_returns=len(graident _ids, timeout=None))

# Let worker 0 reduce the gradients
reduced_id_ref = workers[0].reduce_gradients.remote(gradient_ids)
```

Analyze Performance

- Message over networks: 3*N.
- Can we do better?



Not Yet Finished: Synchronization

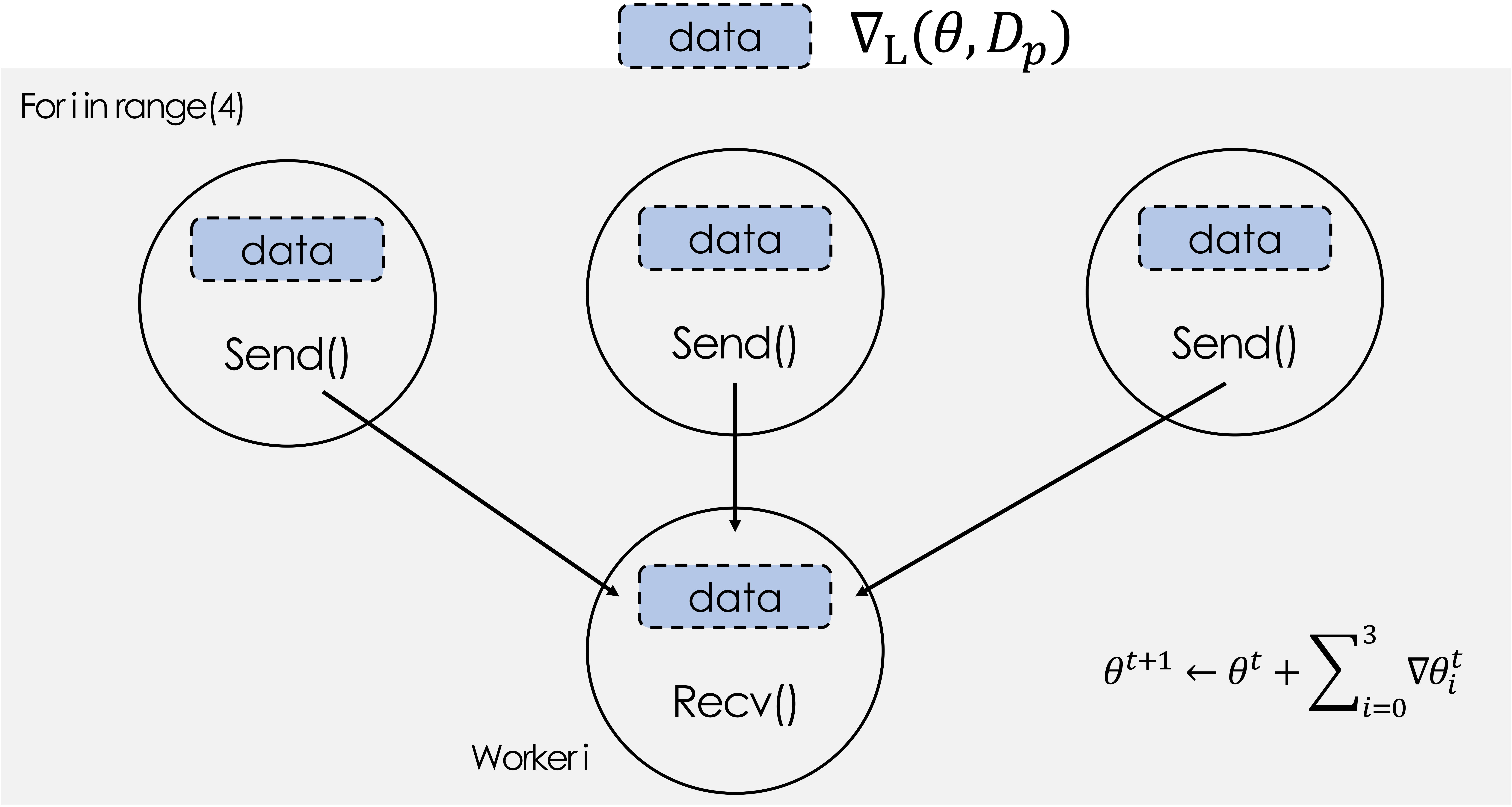


For each worker

$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$

How to perform this sum?

Problem: We need **All-**Reduce



Program This?

```python
@ray.remote(num_gpus=1)
class GPUWorker:
  def __init__(self):
      self.gradients = cupy.ones((10,), dtype=cupy.float32)

  def put_gradients(self):
      return ray.put(self.gradients)

  def reduce_gradients(self, grad_id_refs):
      grad_ids = ray.get(grad_id_refs)
      reduced_result = cupy.ones((10,), dtype=float32)
      for grad_id in grad_ids:
          array = ray.get(grad_id)
          reduced_result += array
      result_id = ray.put(reduced_result)
      return result_id

  def get_reduced_gradient(self, reduced_gradient_id_ref):
      reduced_gradient_id = ray.get(reduced_gradient_id_ref)
      reduced_gradient = ray.get(reduced_gradient_id)
      # do whatever with the reduced gradients
      return True
```

```python
# Allreduce the gradients using Ray APIs
# Let all workers to put their gradients into the Ray object store.
gradient_ids = [worker.put_gradients.remote() for worker in workers]
ray.wait(graident_ids, num_returns=len(graident _ids, timeout=None))

# Let worker 0 reduce the gradients
reduced_id_ref = workers[0].reduce_gradients.remote(gradient_ids)

# All others workers get the reduced gradients
results = []
for i, worker in enumerate(workers):
    results.append(worker.get_reduced_gradient.remote([red
uced_id_ref]))
ray.get(results)
```

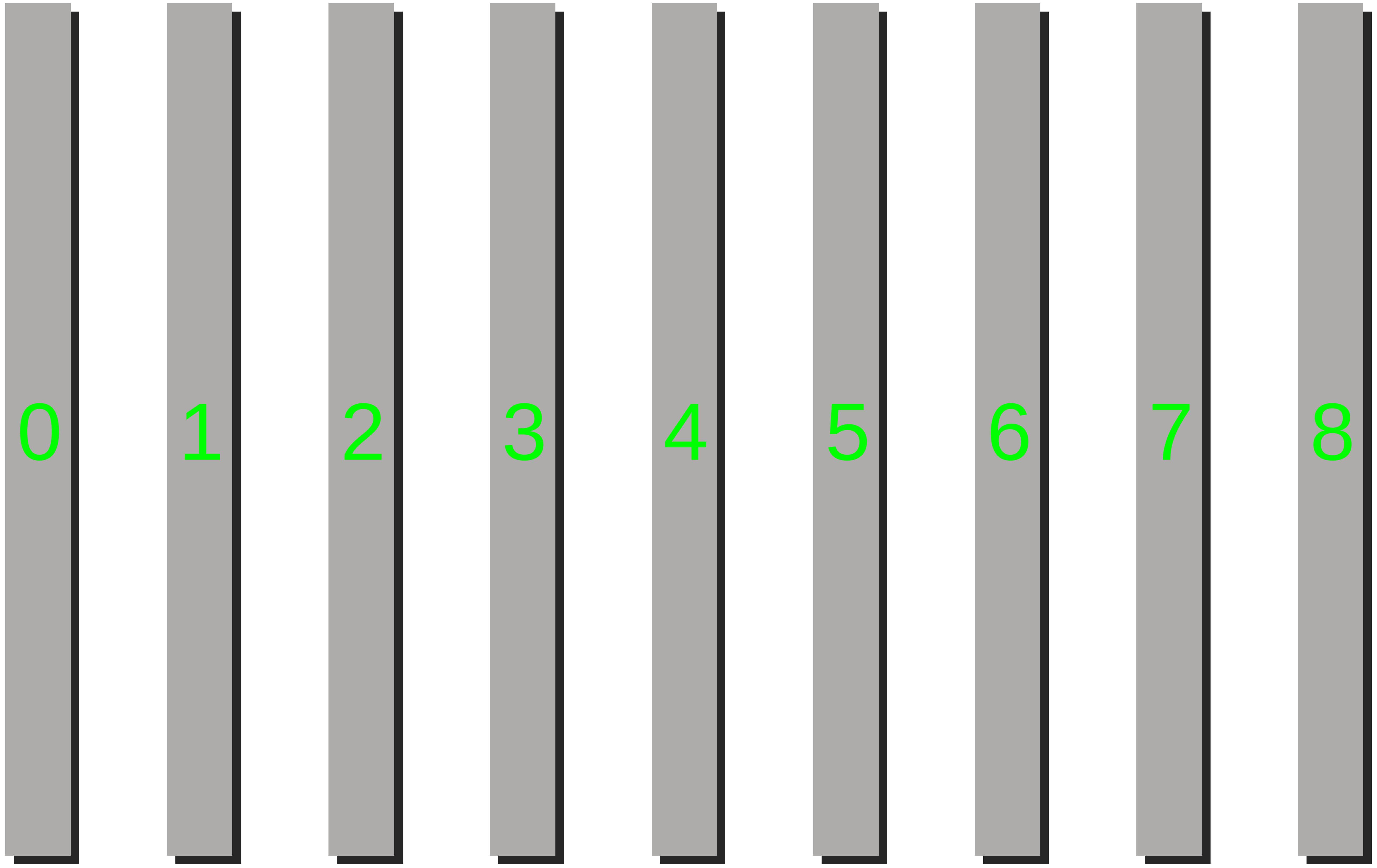
Performance

- Message size over networks:
  - Sum: 3N
  - Send Sum back: 3N
  - = 6N
- Can we do better?
  - Hint: we cannot do better than 3N

Why Collective Communication?

- Programming Convenience
  - Use a set of well-defined communication primitives to express complex communication patterns
- Performance
  - Since they are well defined and well structured, we can optimize them to the extreme
- ML Systems 🧡 Collective communication

Make it Formal



- A 1D **Mesh** of workers (or devices, or nodes)

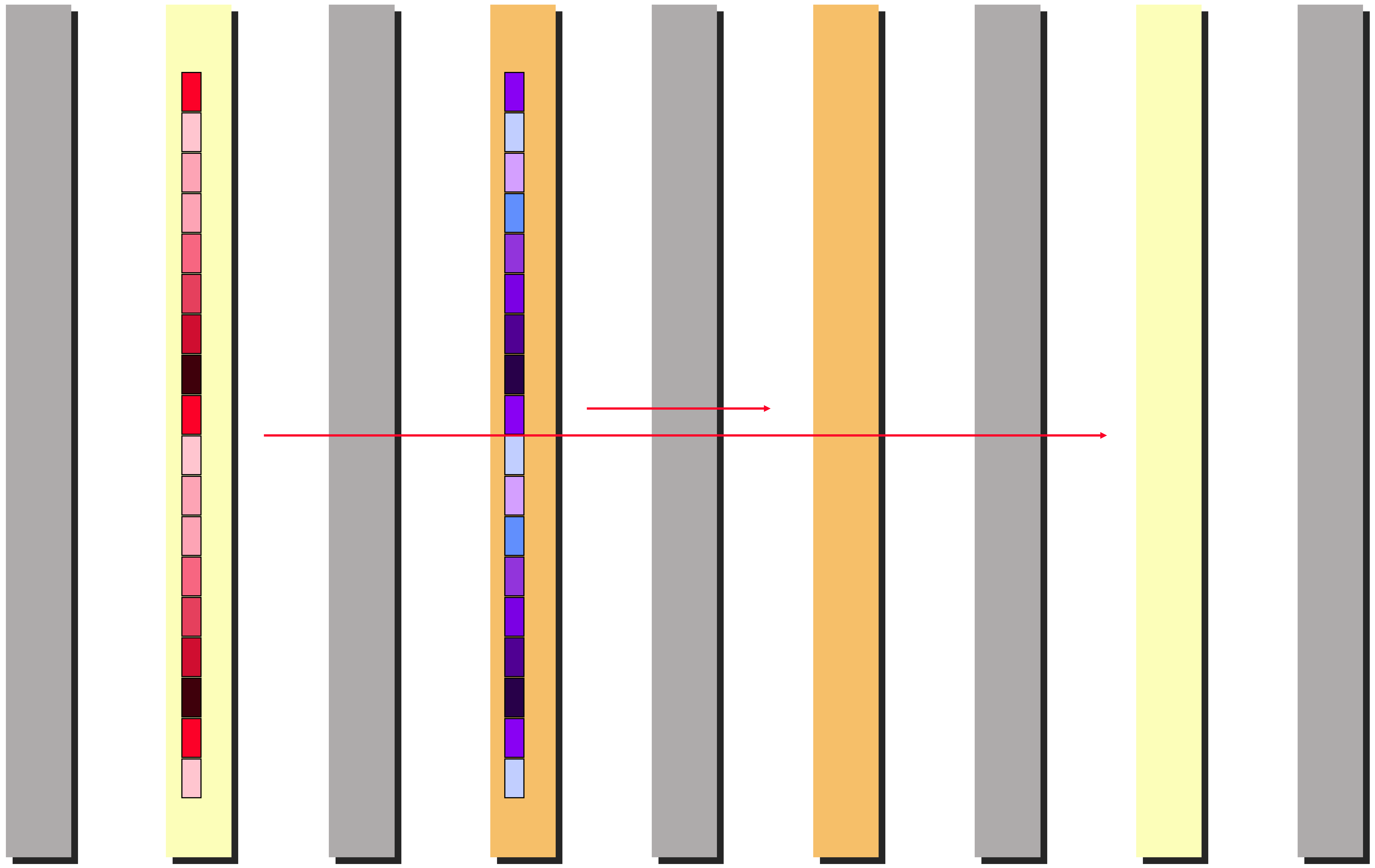
Model of Parallel Computation

- a node can send directly to any other node (maybe not true)
- a node can simultaneously receive and send
- cost of communication
  - sending a message of length n between any two nodes

$$\alpha + n\,\beta$$

  - if a message encounters a link that simultaneously accommodates M messages, the cost becomes

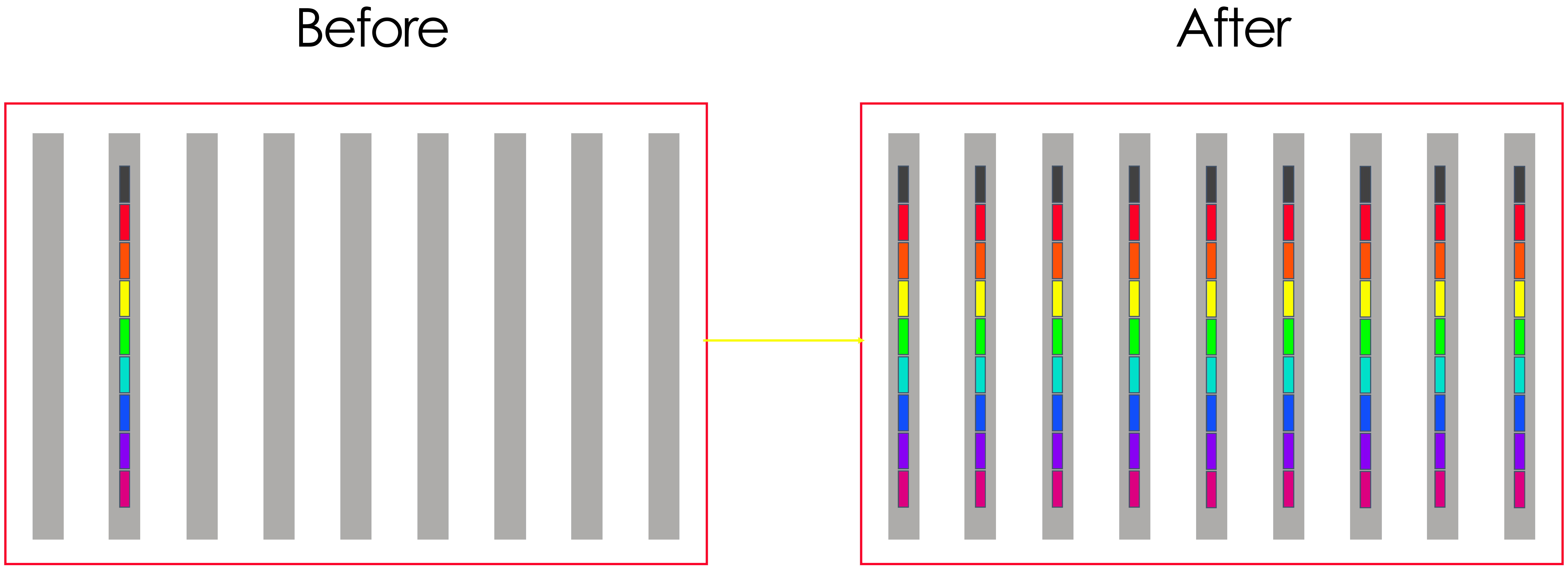
$$\alpha + Mn\,\beta$$



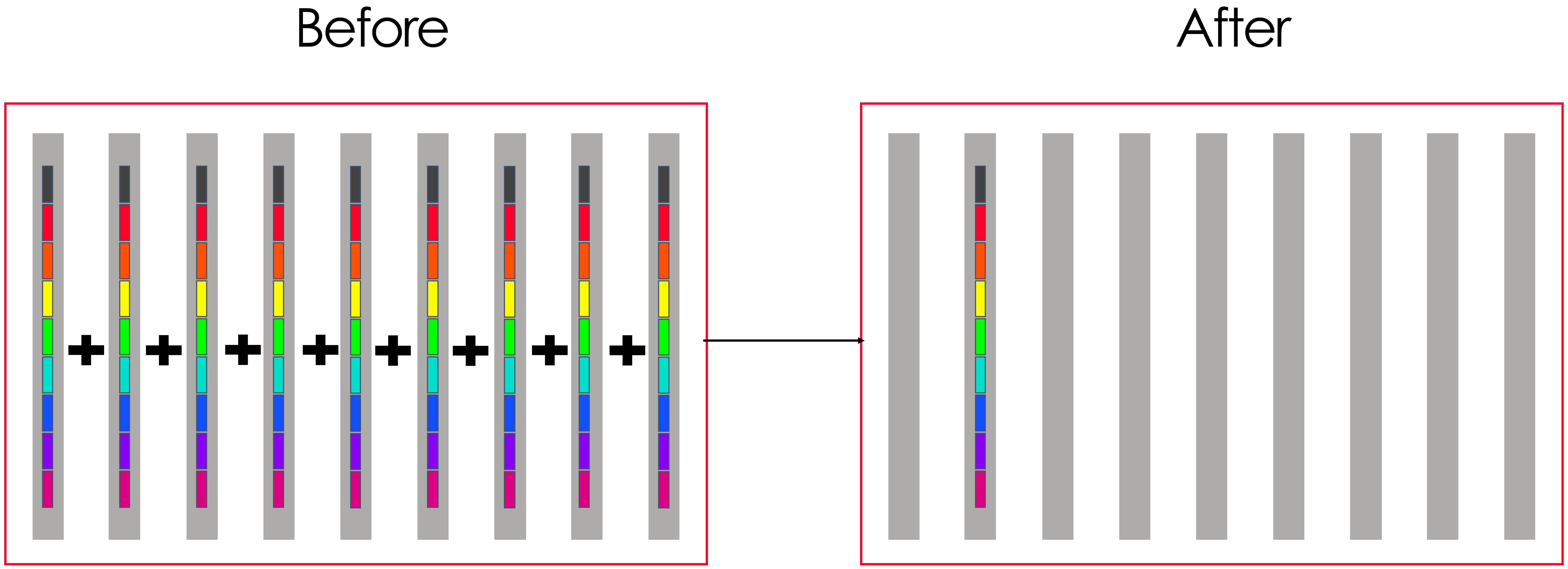
Collective Communications

- Broadcast
- Reduce(-to-one)
- Scatter
- Gather
- Allgather
- Reduce-scatter
- Allreduce

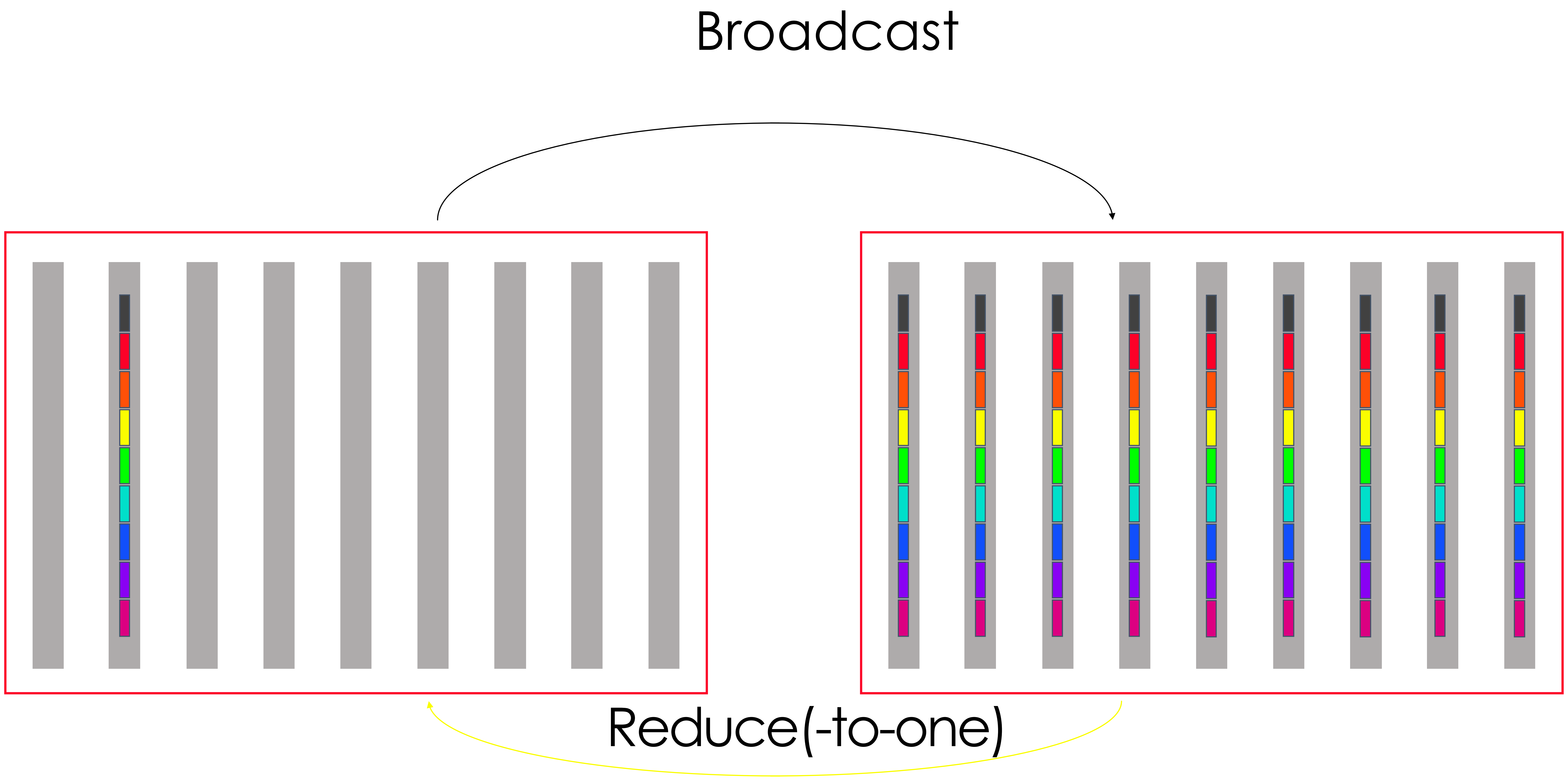
Broadcast



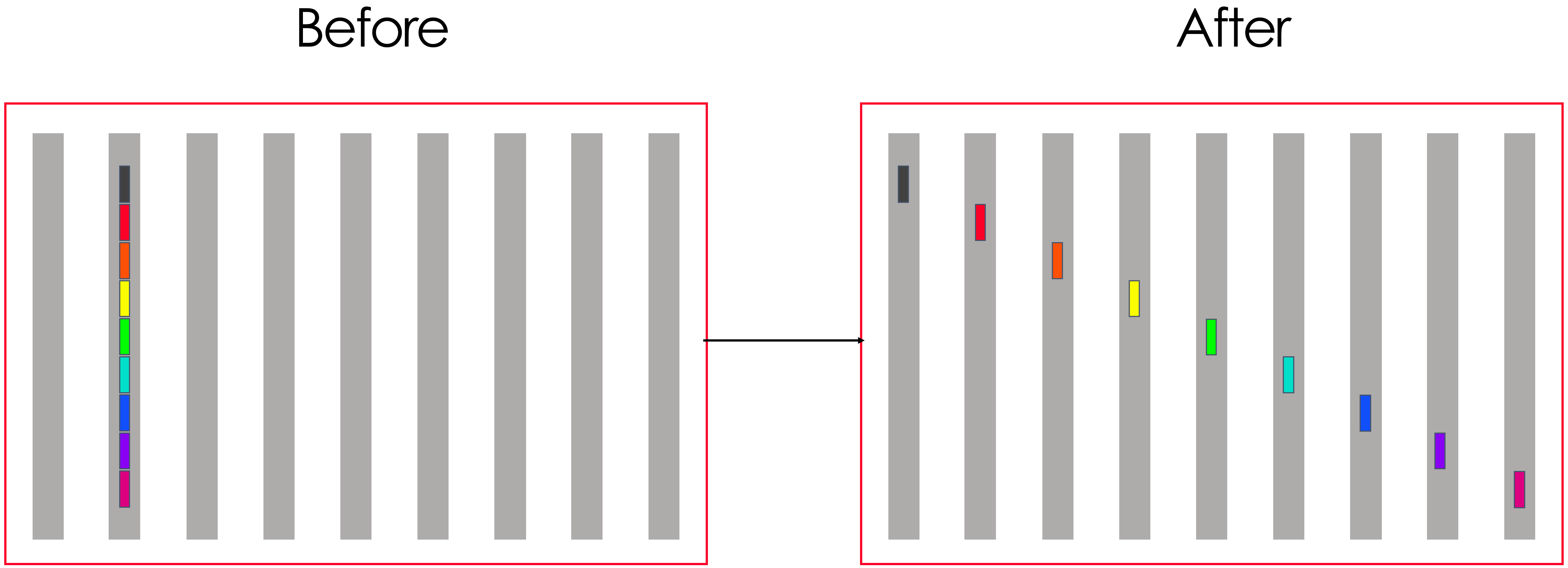
Reduce(-to-one)



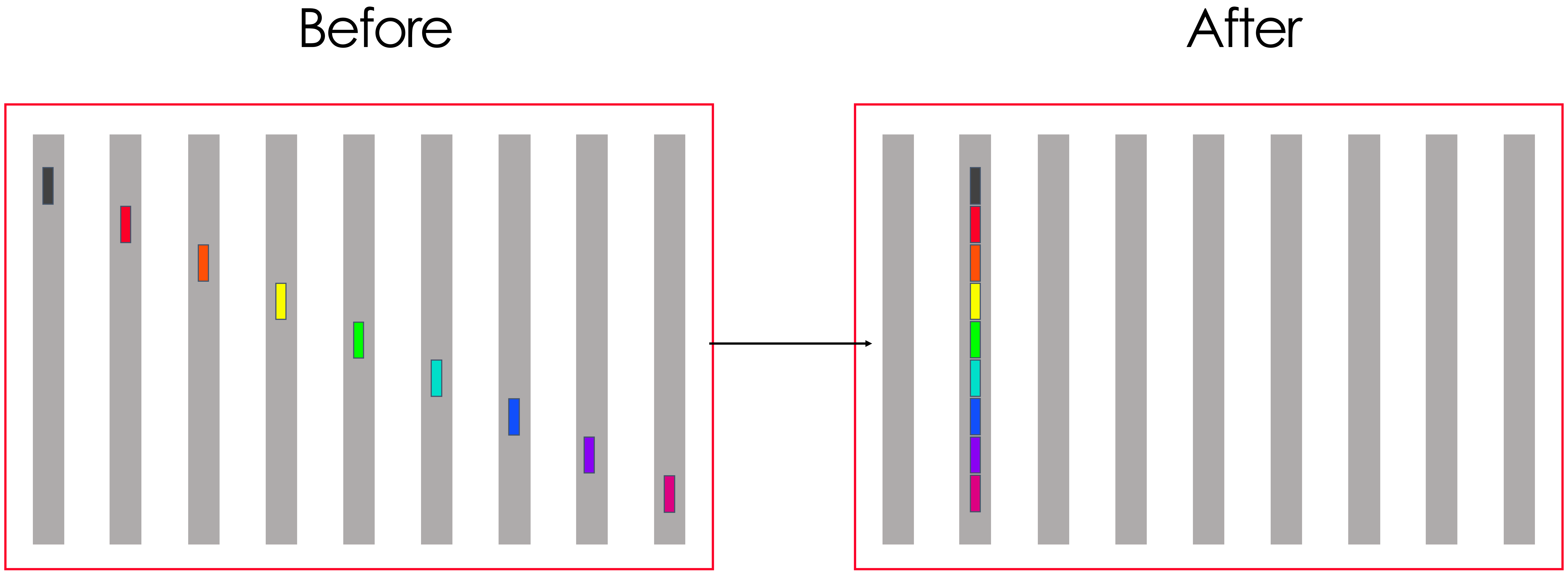
Broadcast/Reduce(-to-one)



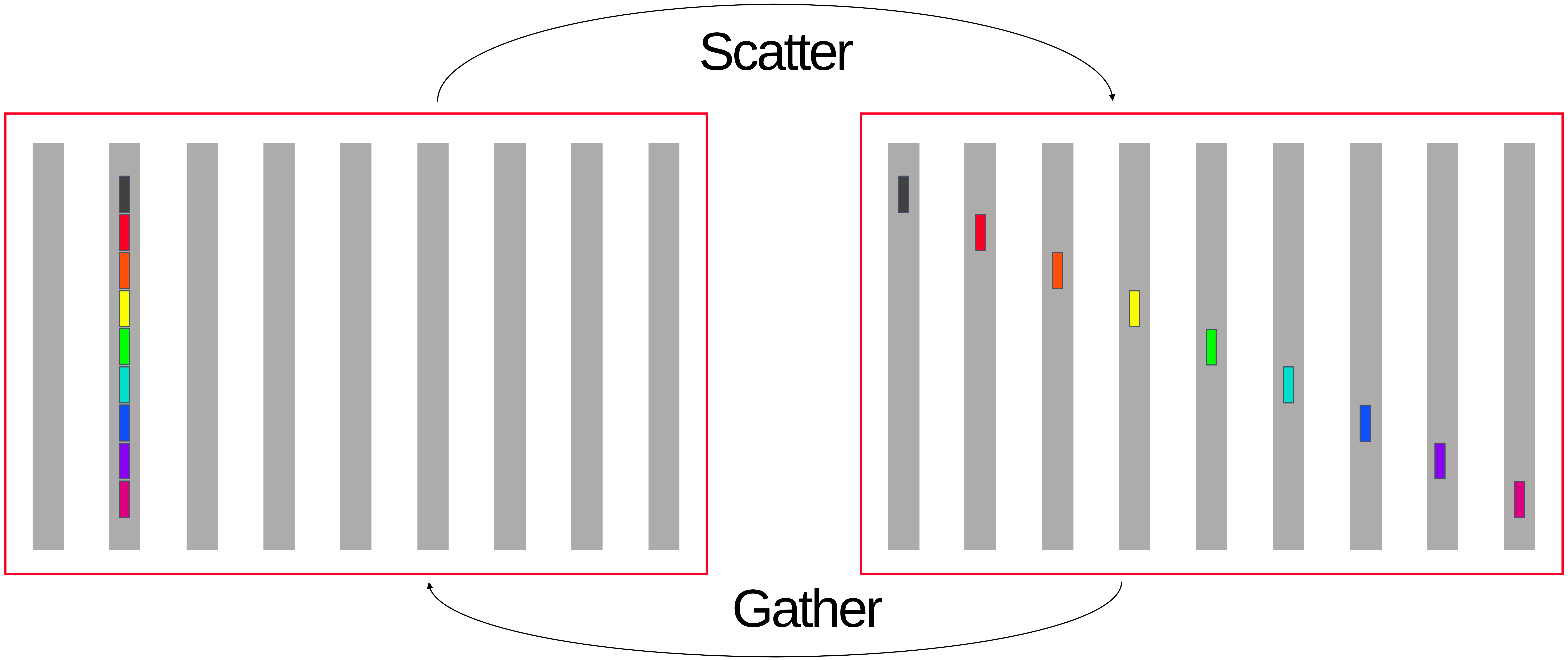
Scatter



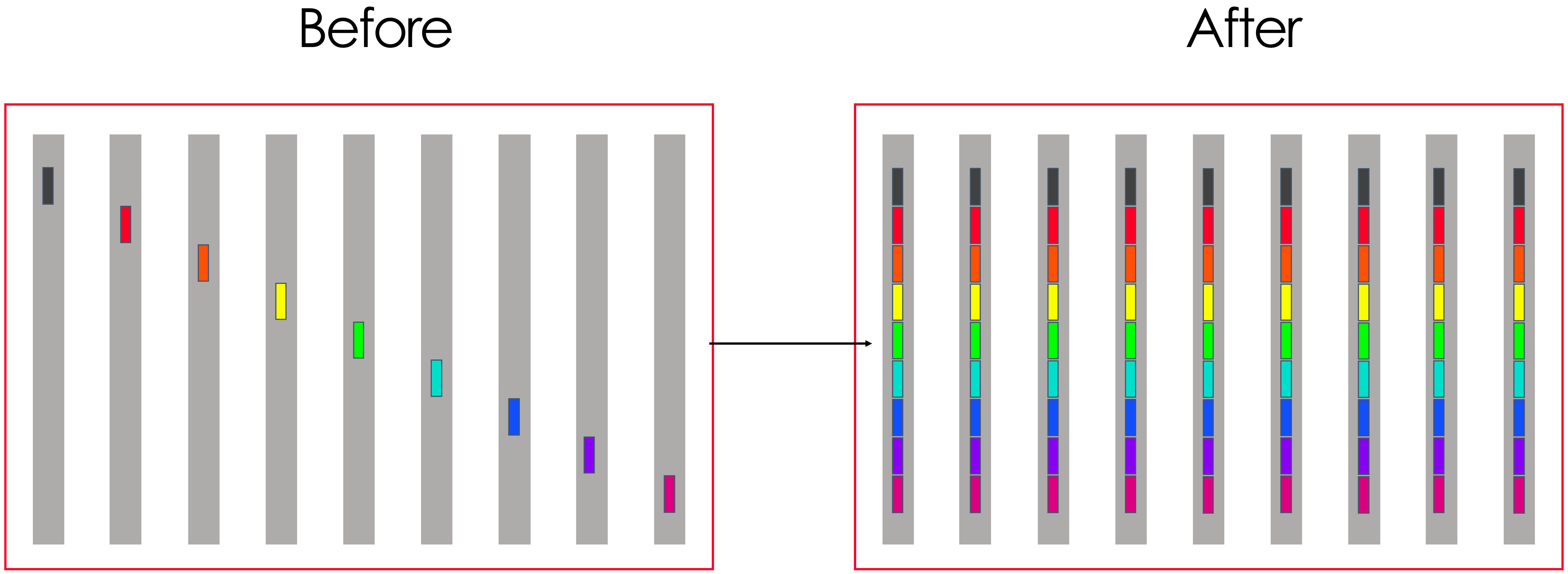
Gather



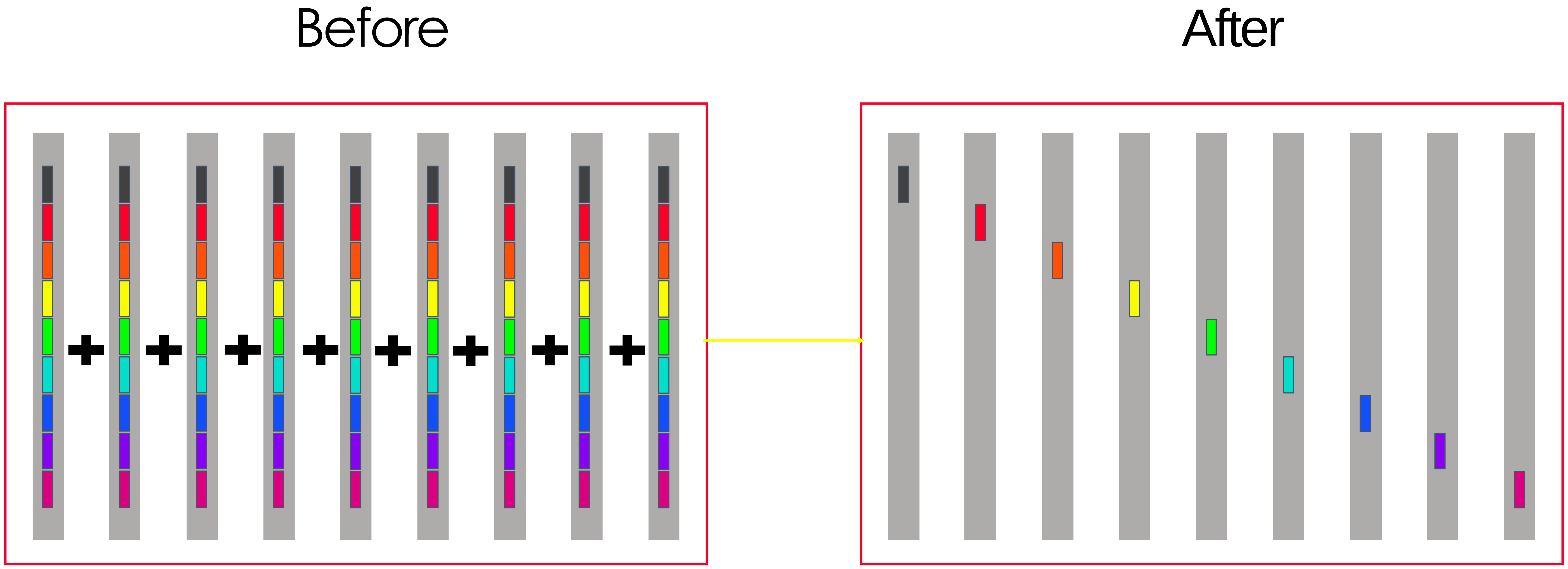
Scatter/Gather



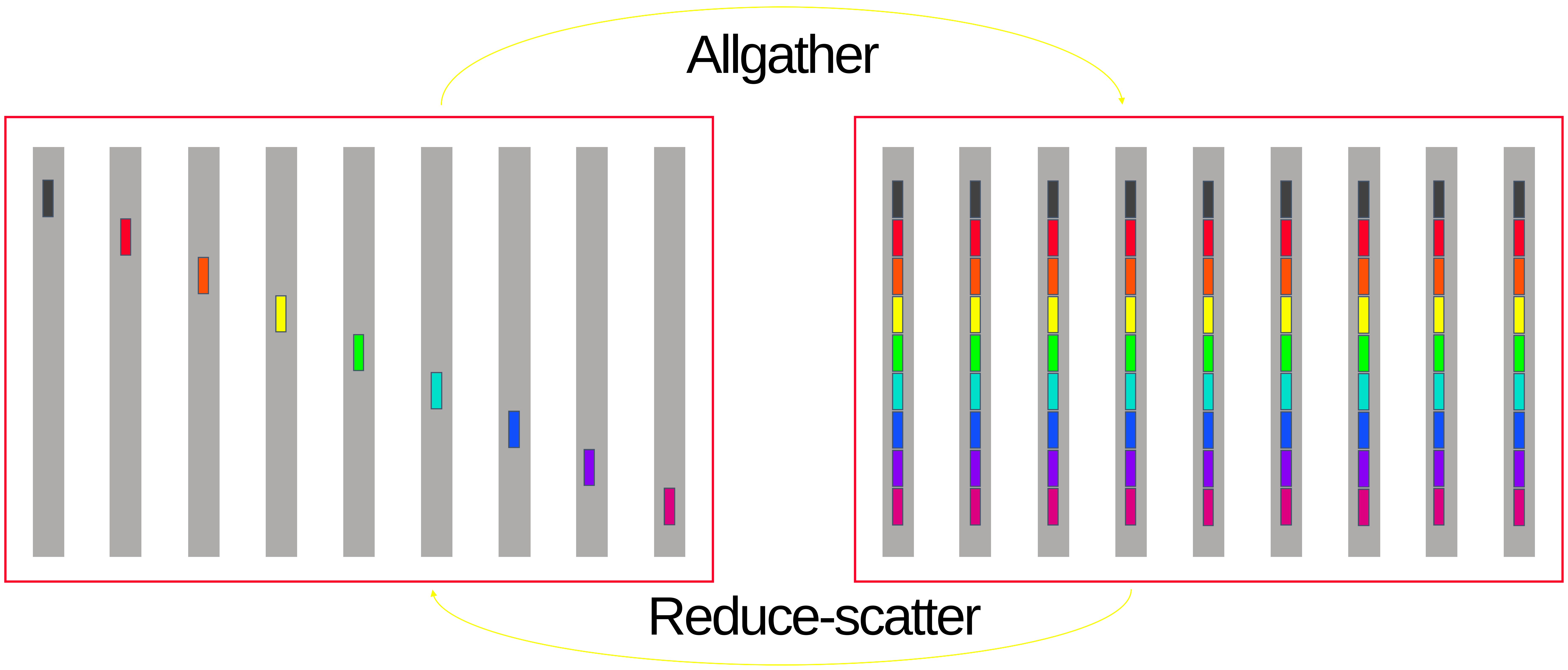
Allgather



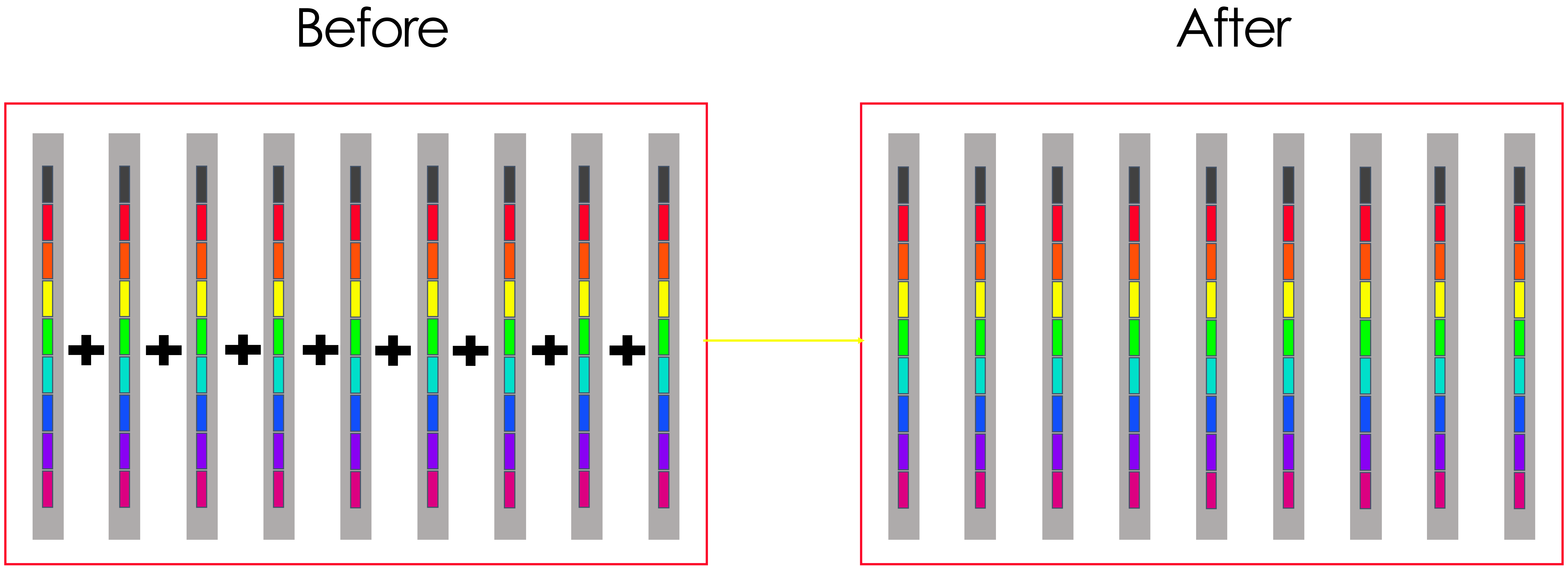
Reduce-scatter



Allgather/Reduce-scatter



Allreduce



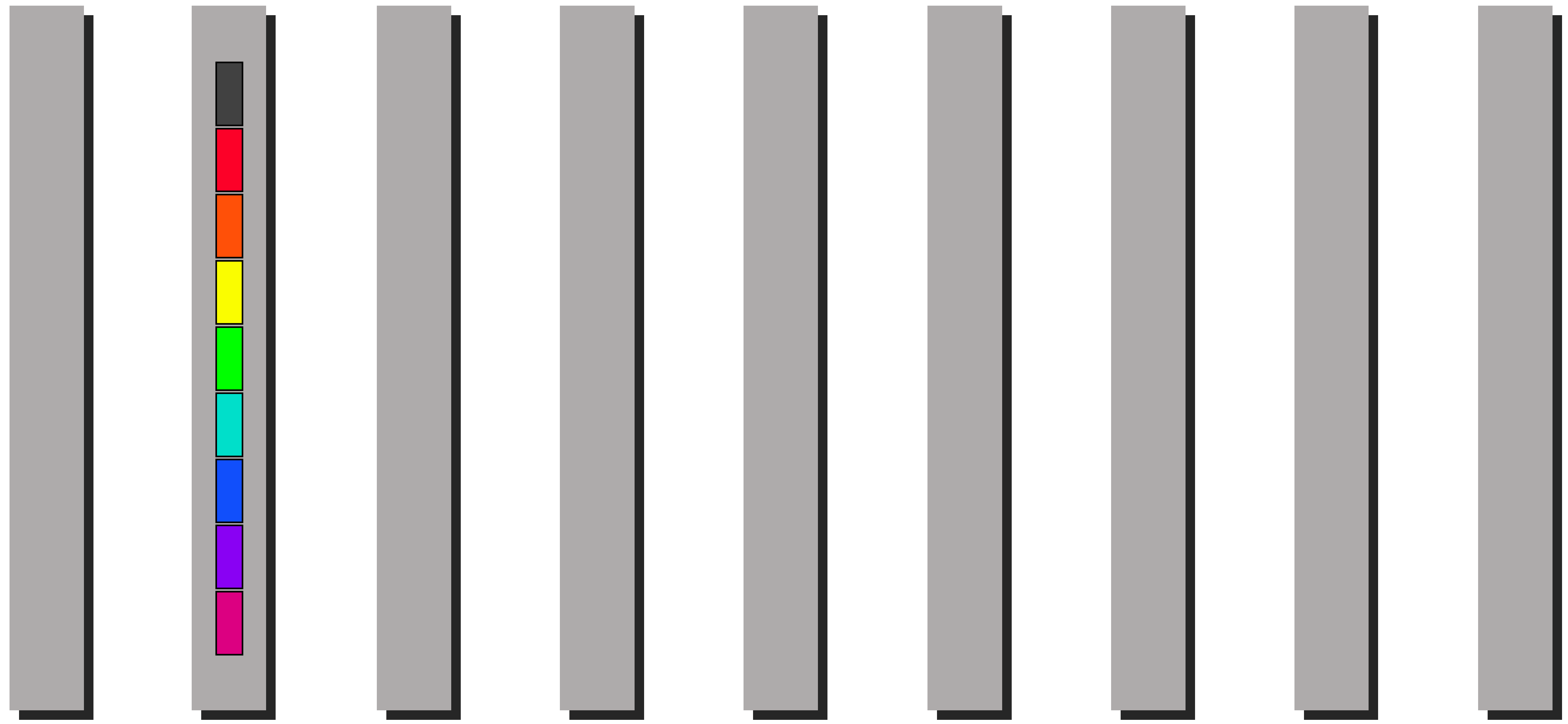
Two Family of Mainstream Algorithms/Implementations

- Small message: Minimum Spanning Tree algorithm
  - Emphasize **low latency**

- Large Message: Ring algorithm
  - Emphasize **bandwidth utilization**

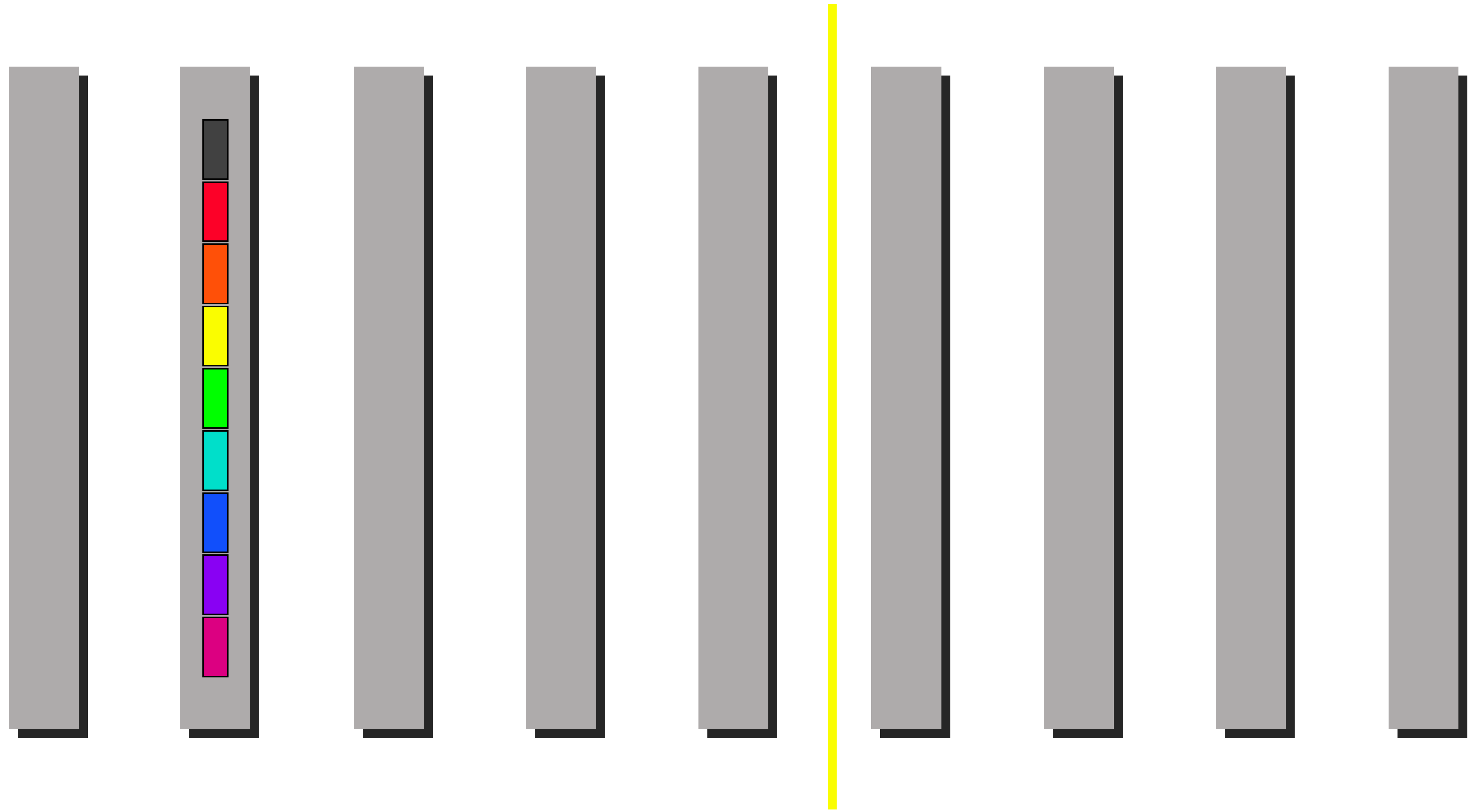
- There are 50+ different algorithms developed in the past 50 years by a community called “High-performance computing”
  - Last year Turing award

General principles



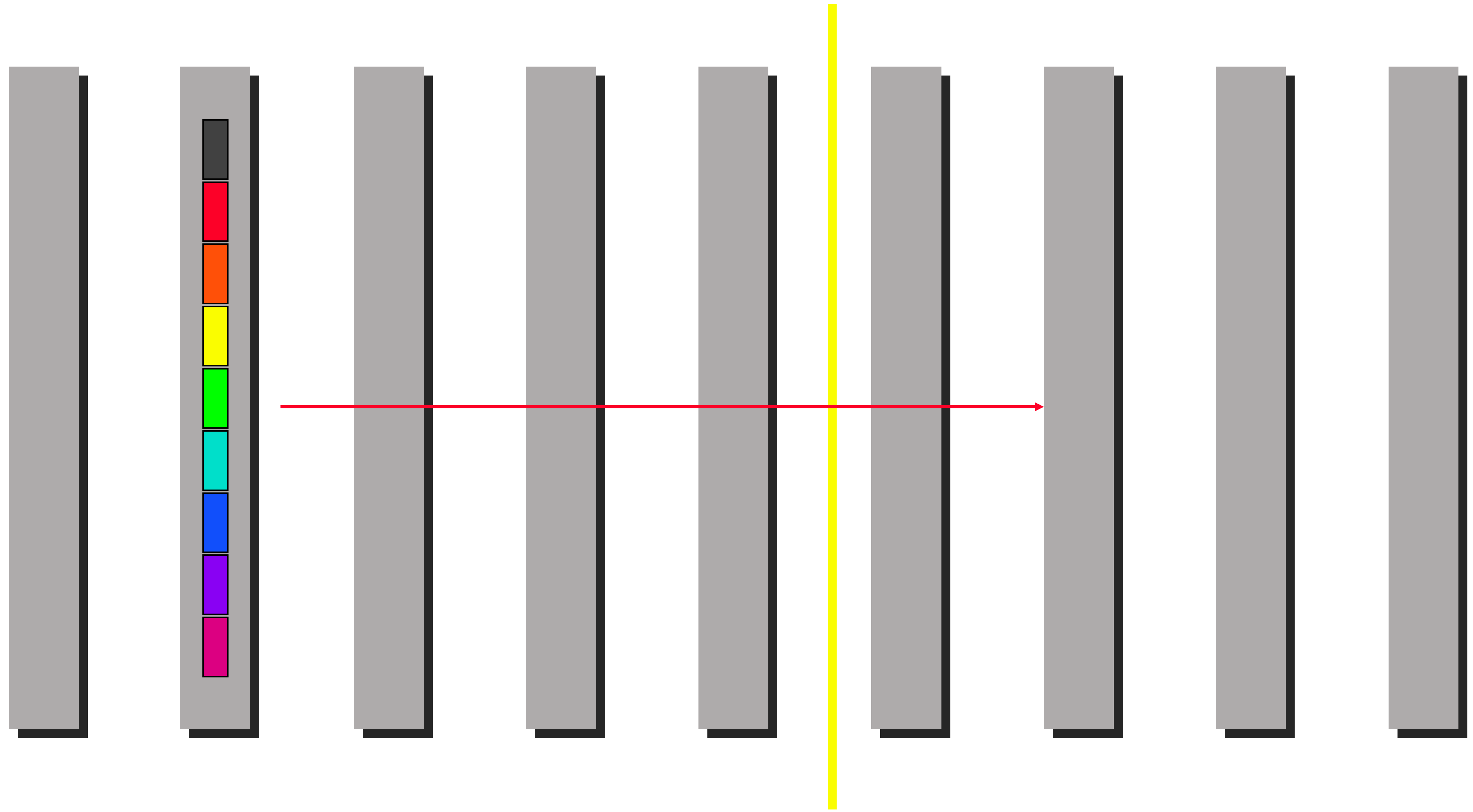
- message starts on one processor

General principles



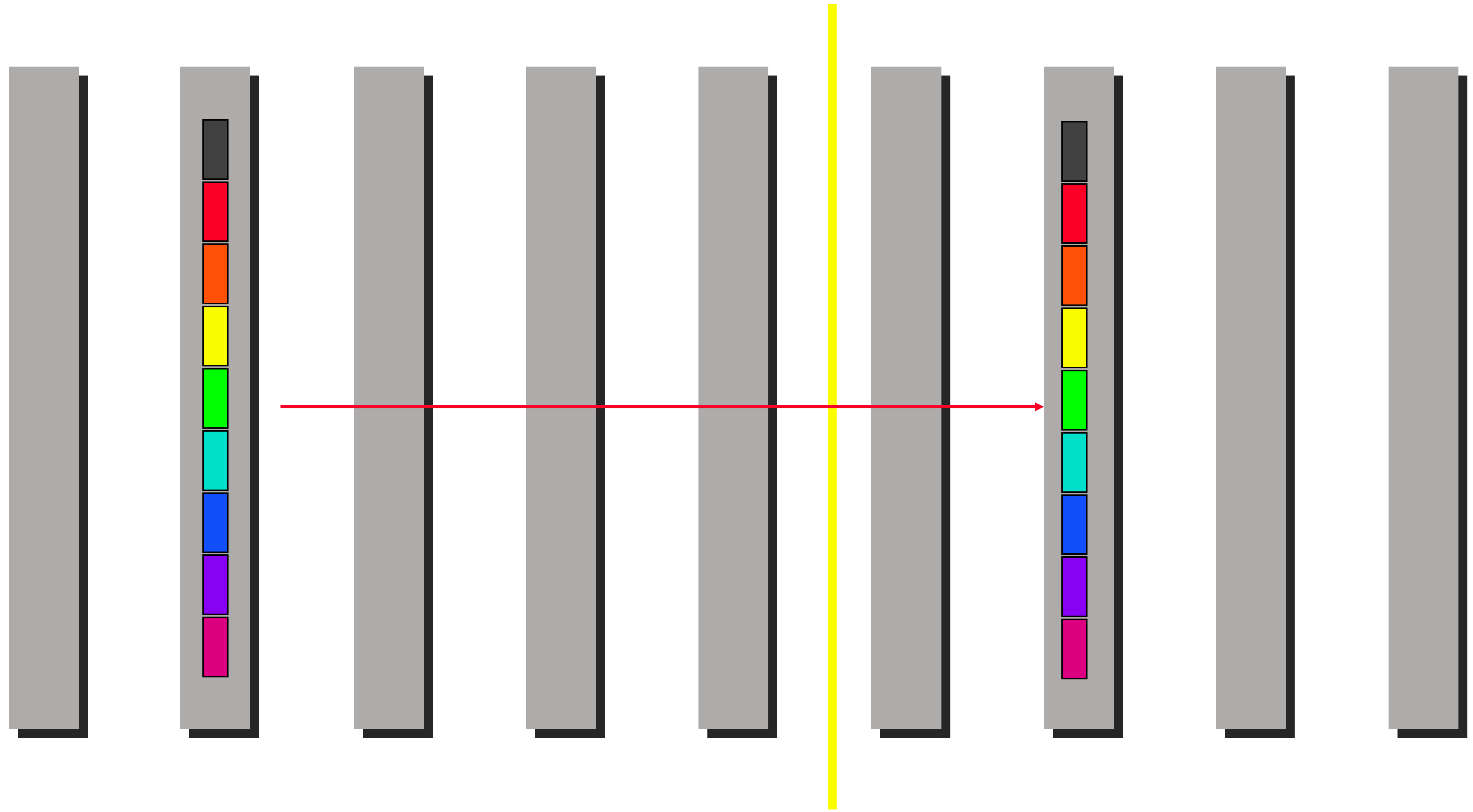
- divide logical linear array in half

General principles



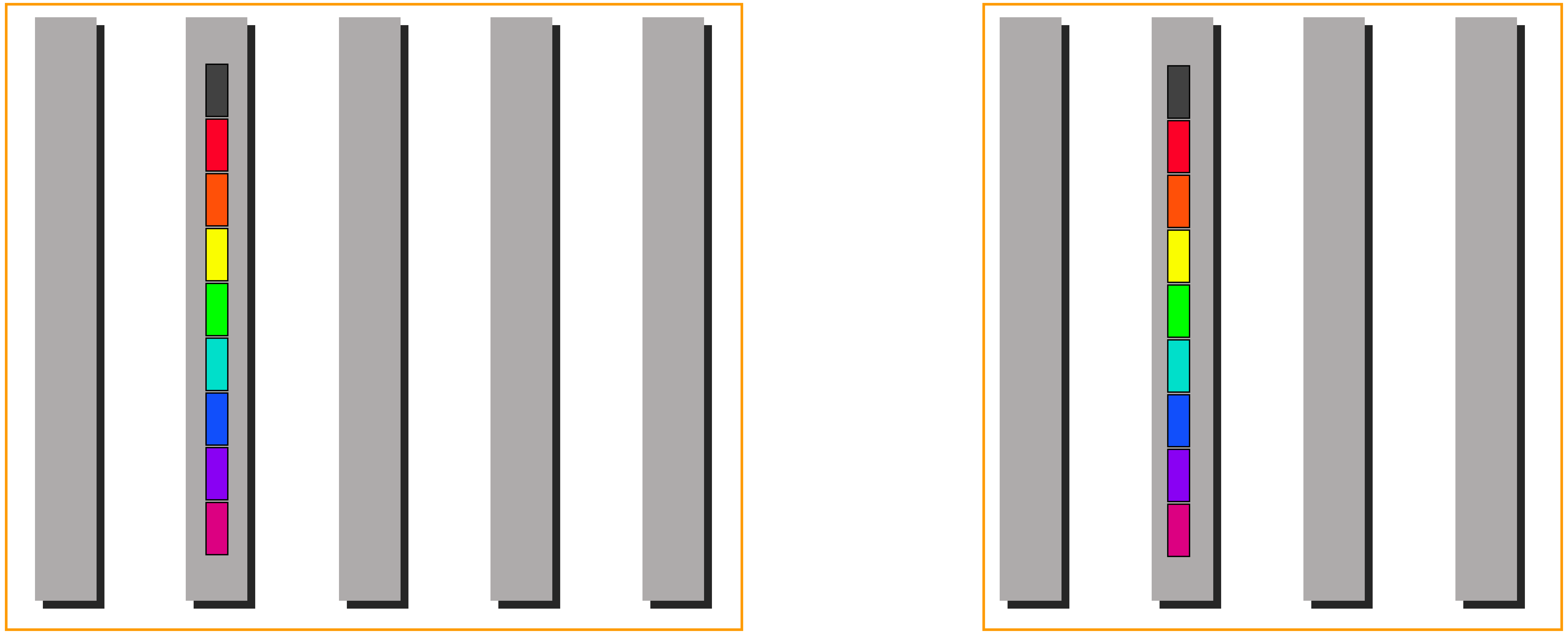
- send message to the half of the network that does not contain the current node (root) that holds the message

General principles



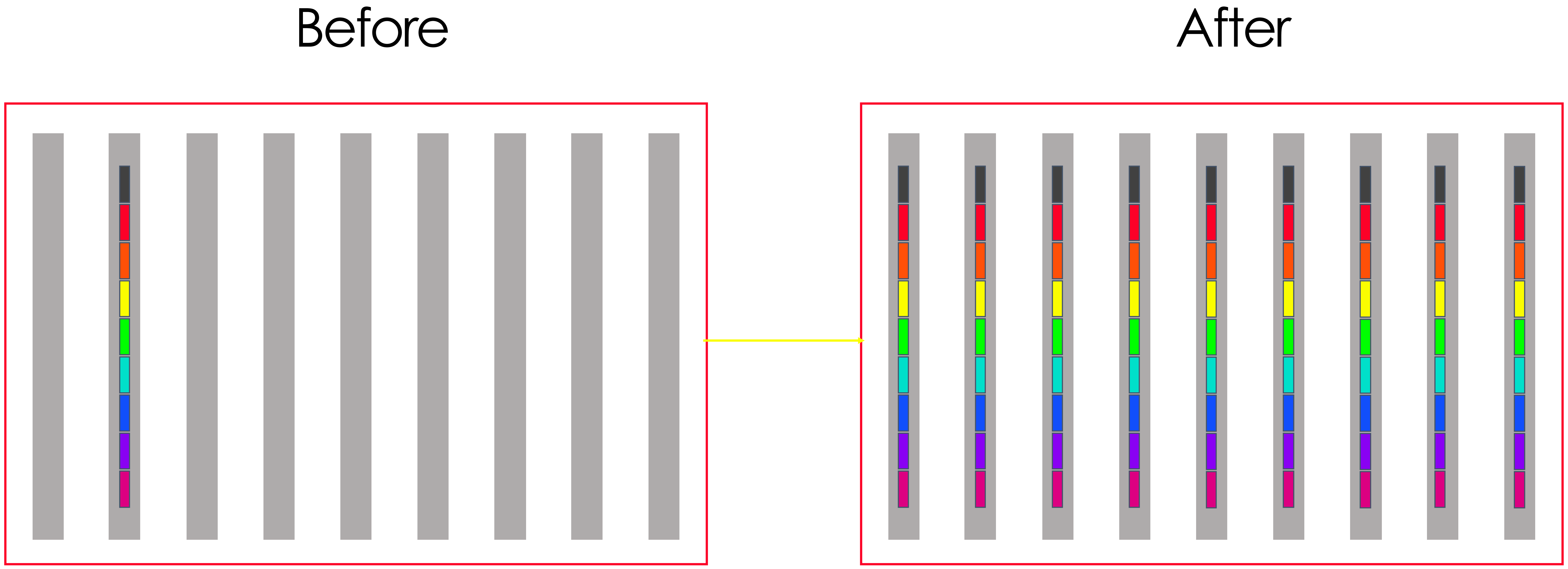
- send message to the half of the network that does not contain the current node (root) that holds the message

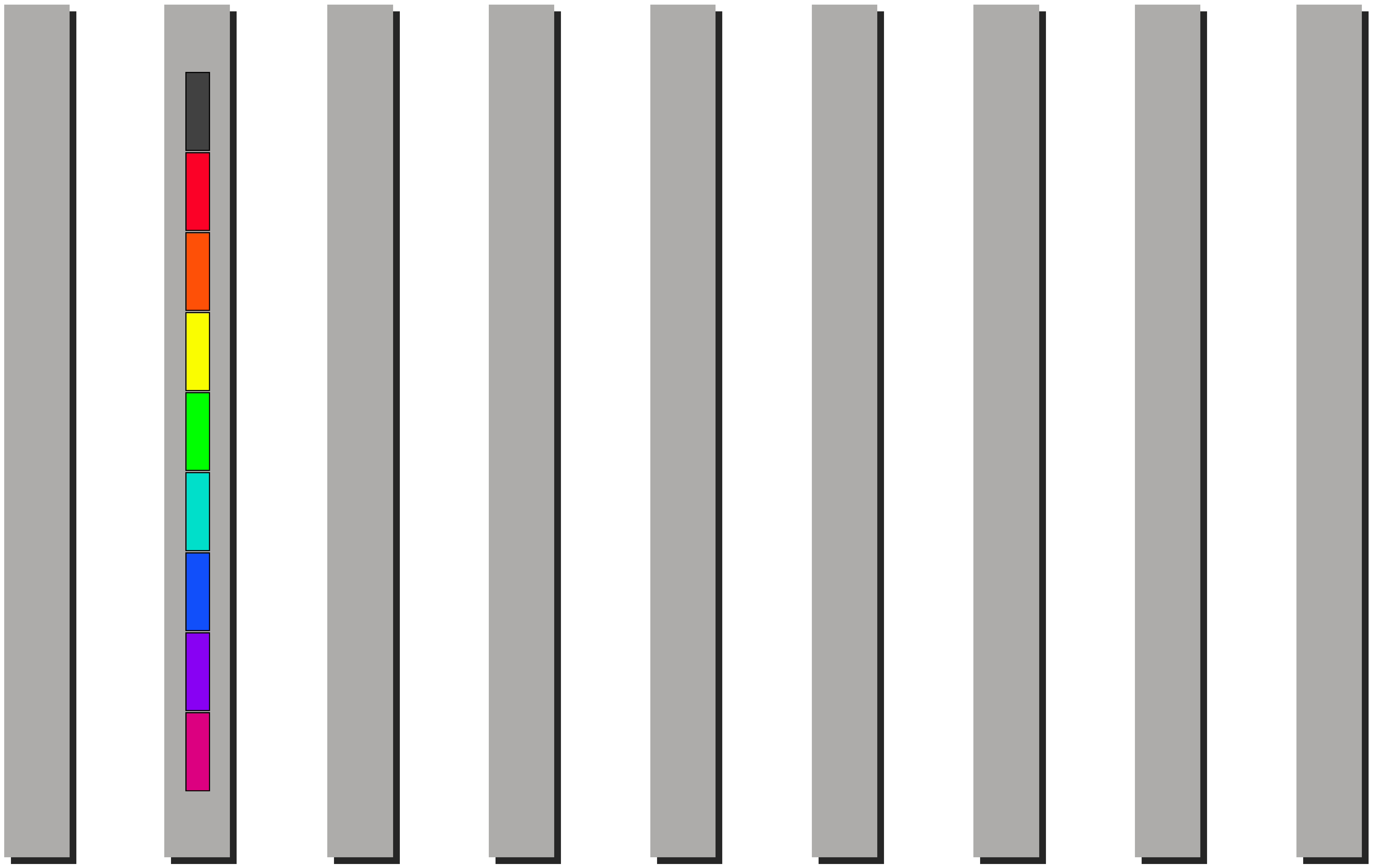
General principles

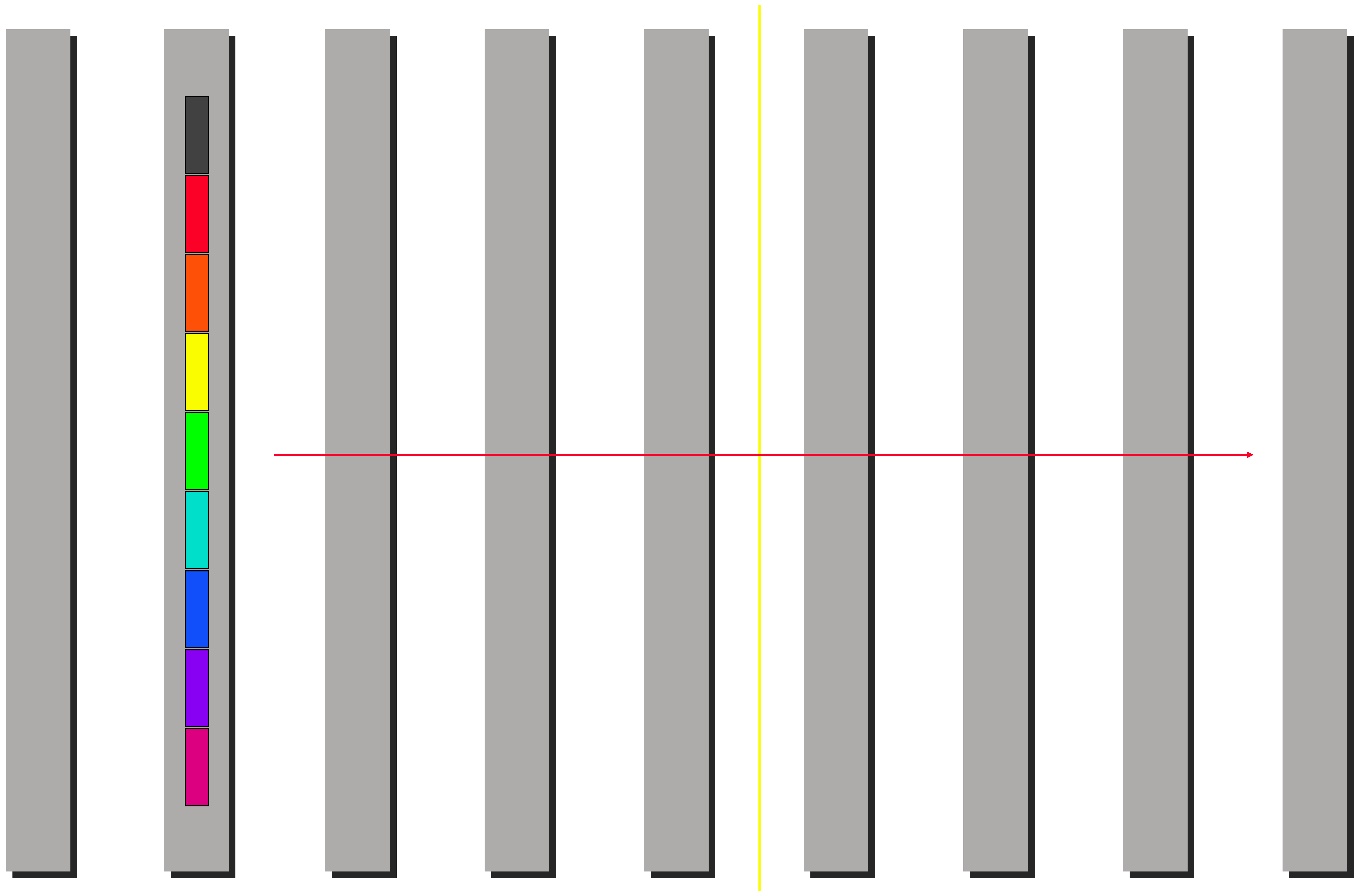


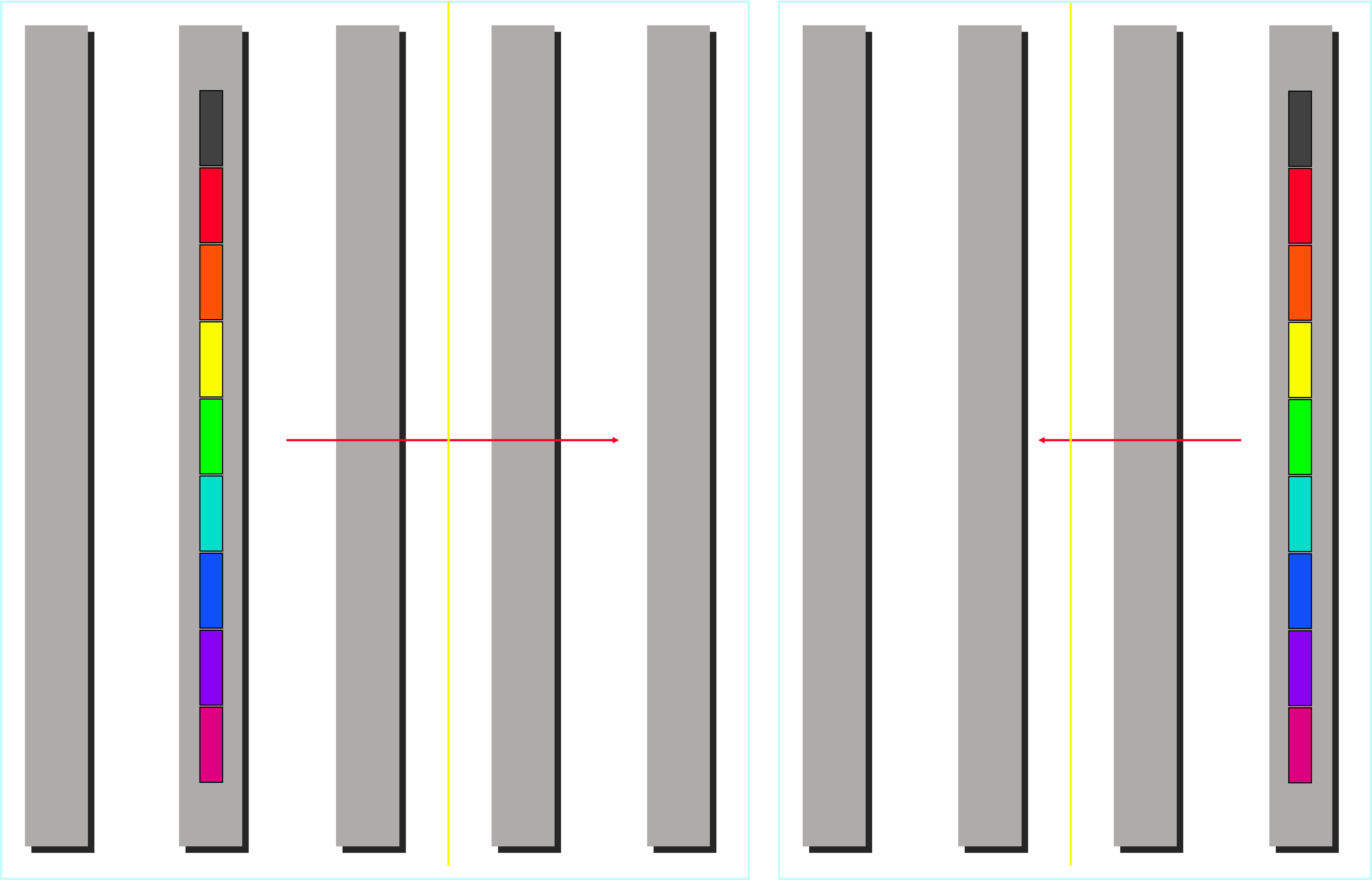
- continue recursively in each of the two halves

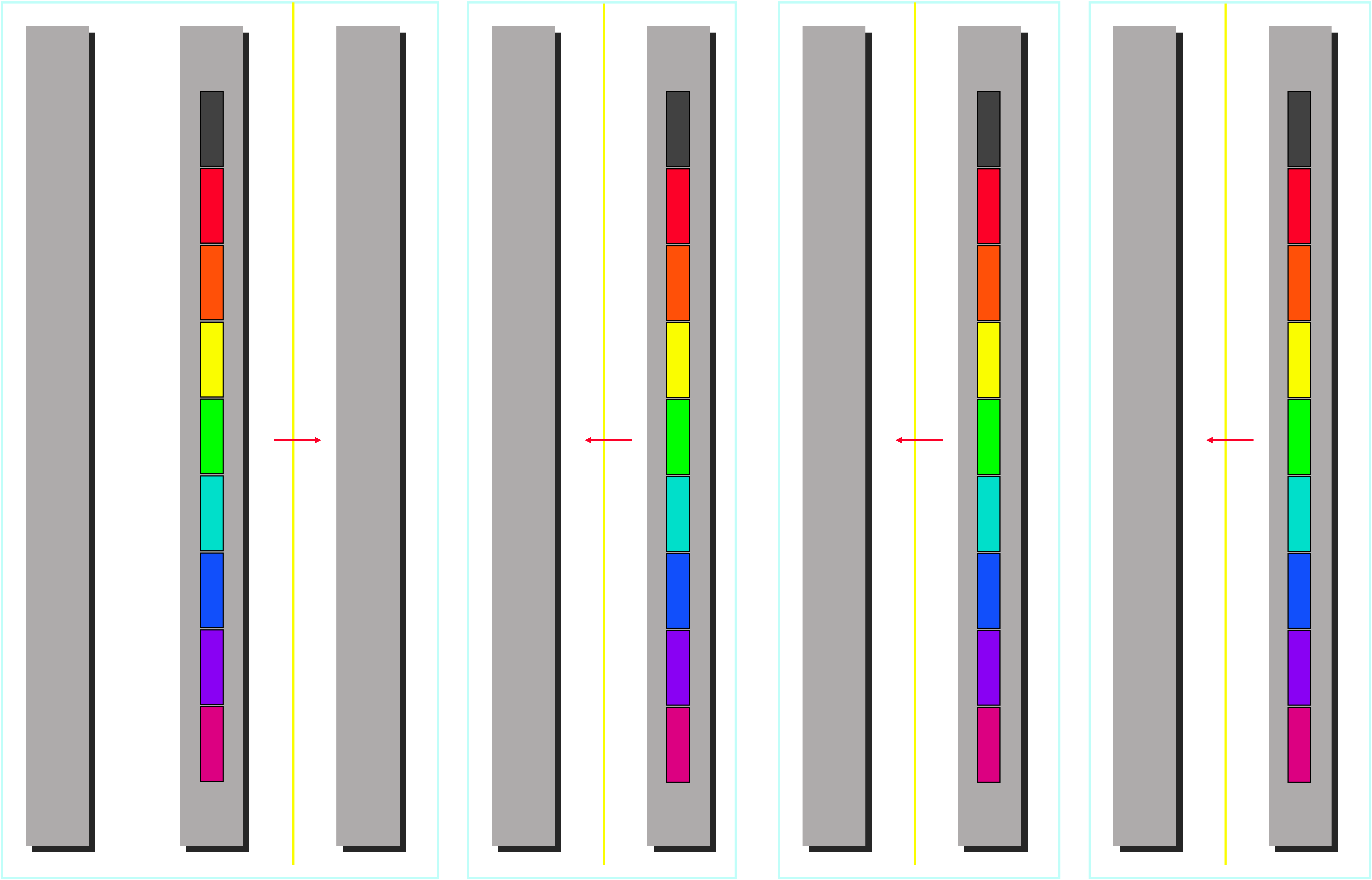
Broadcast

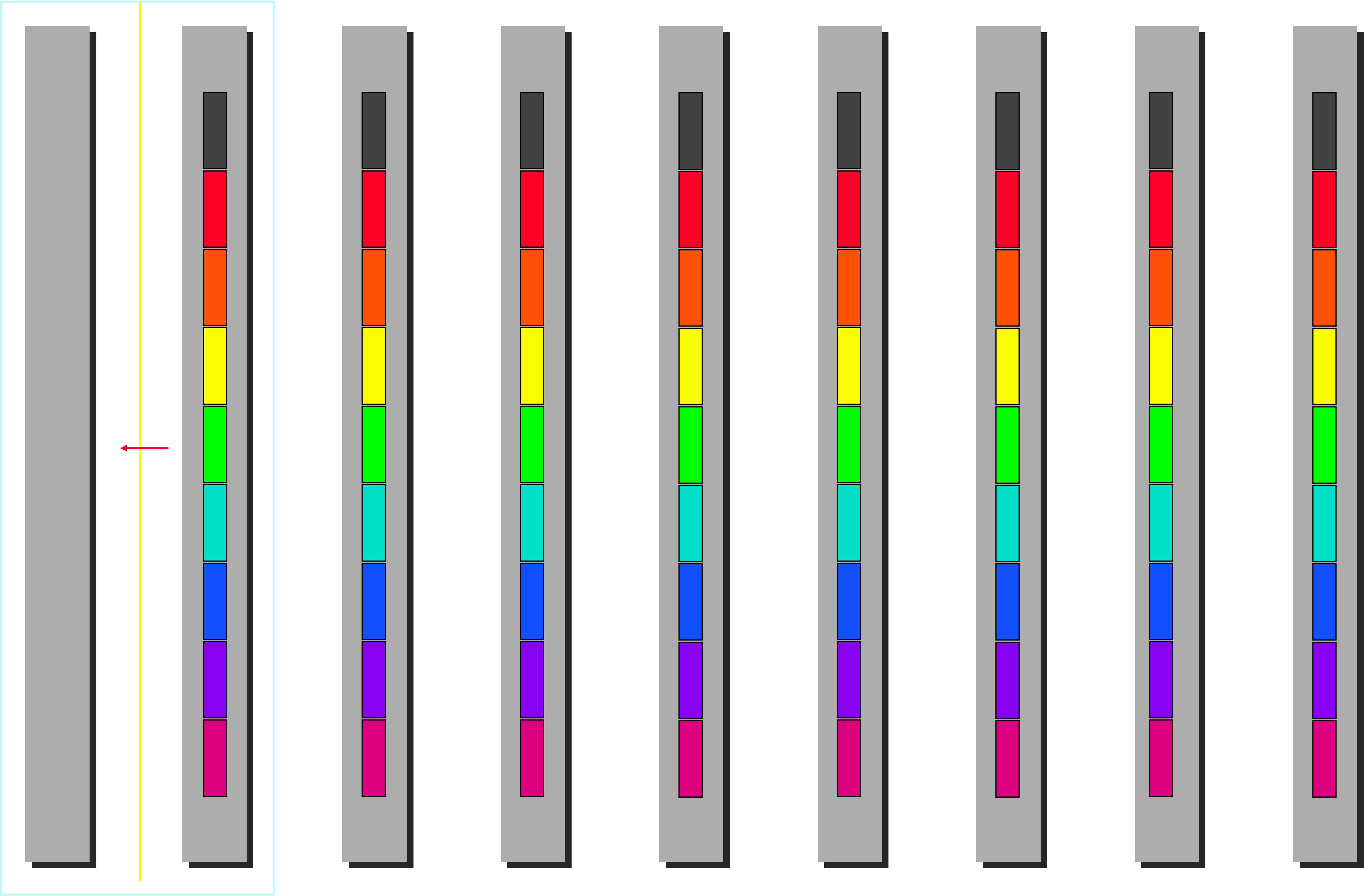


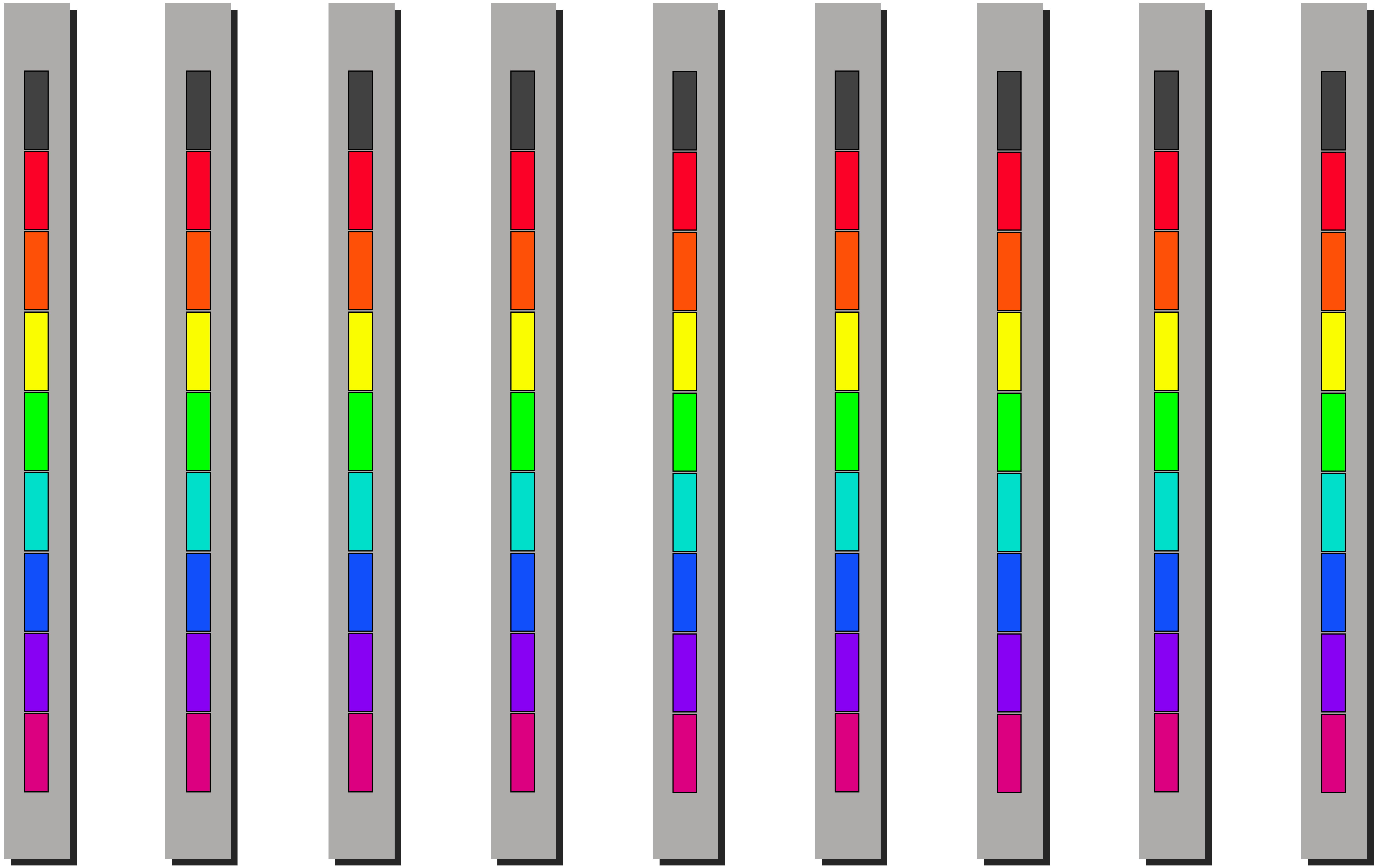








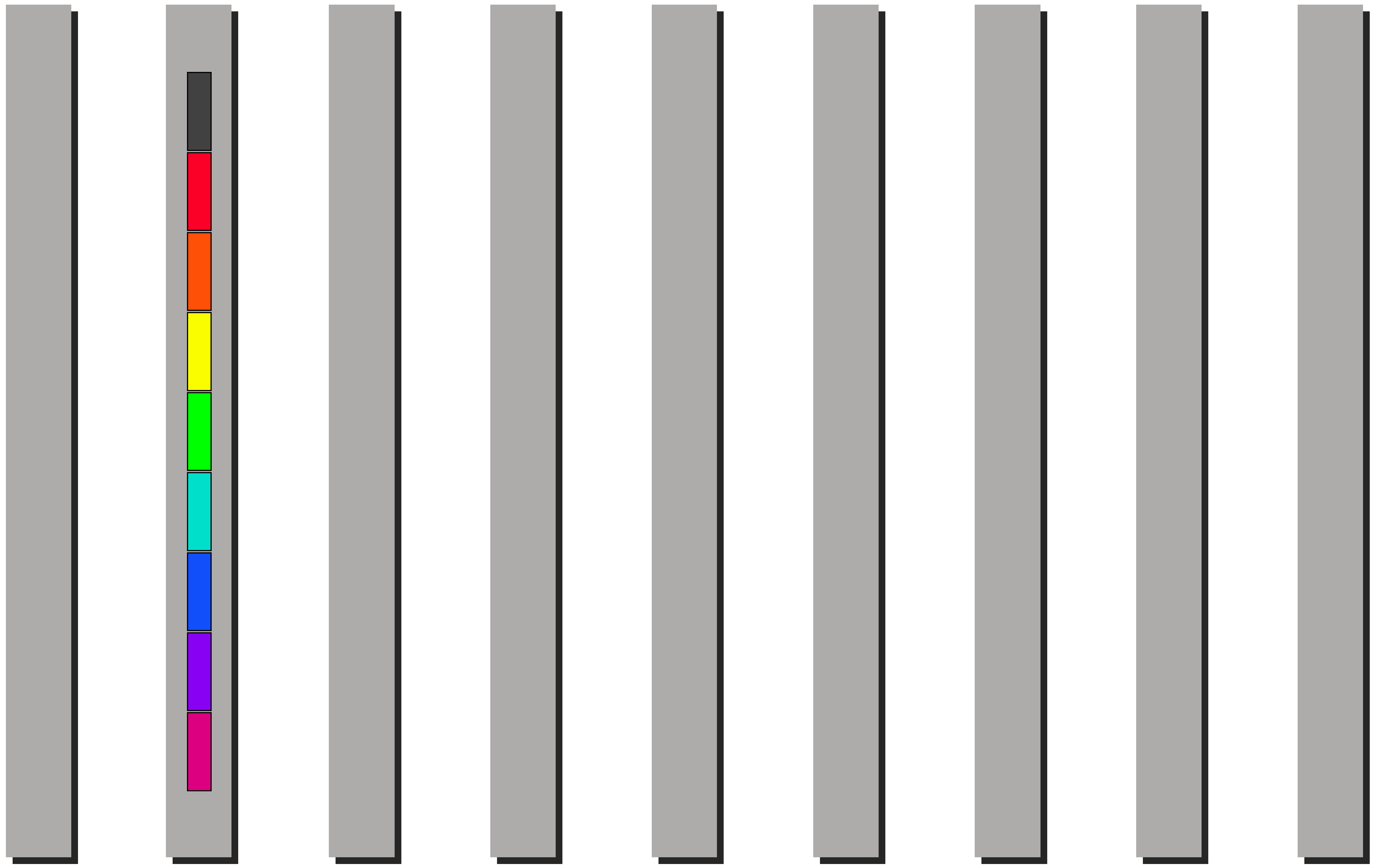


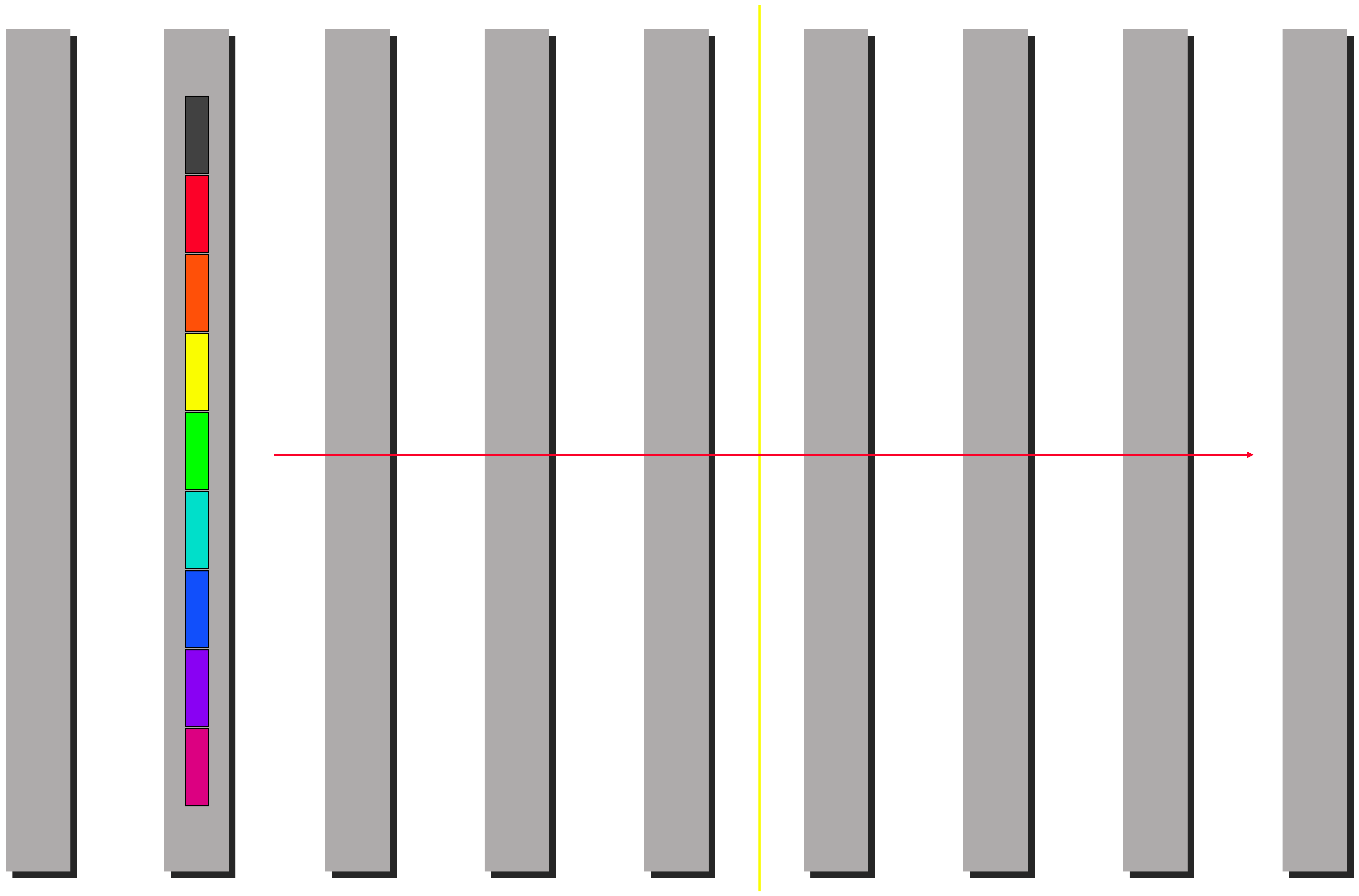


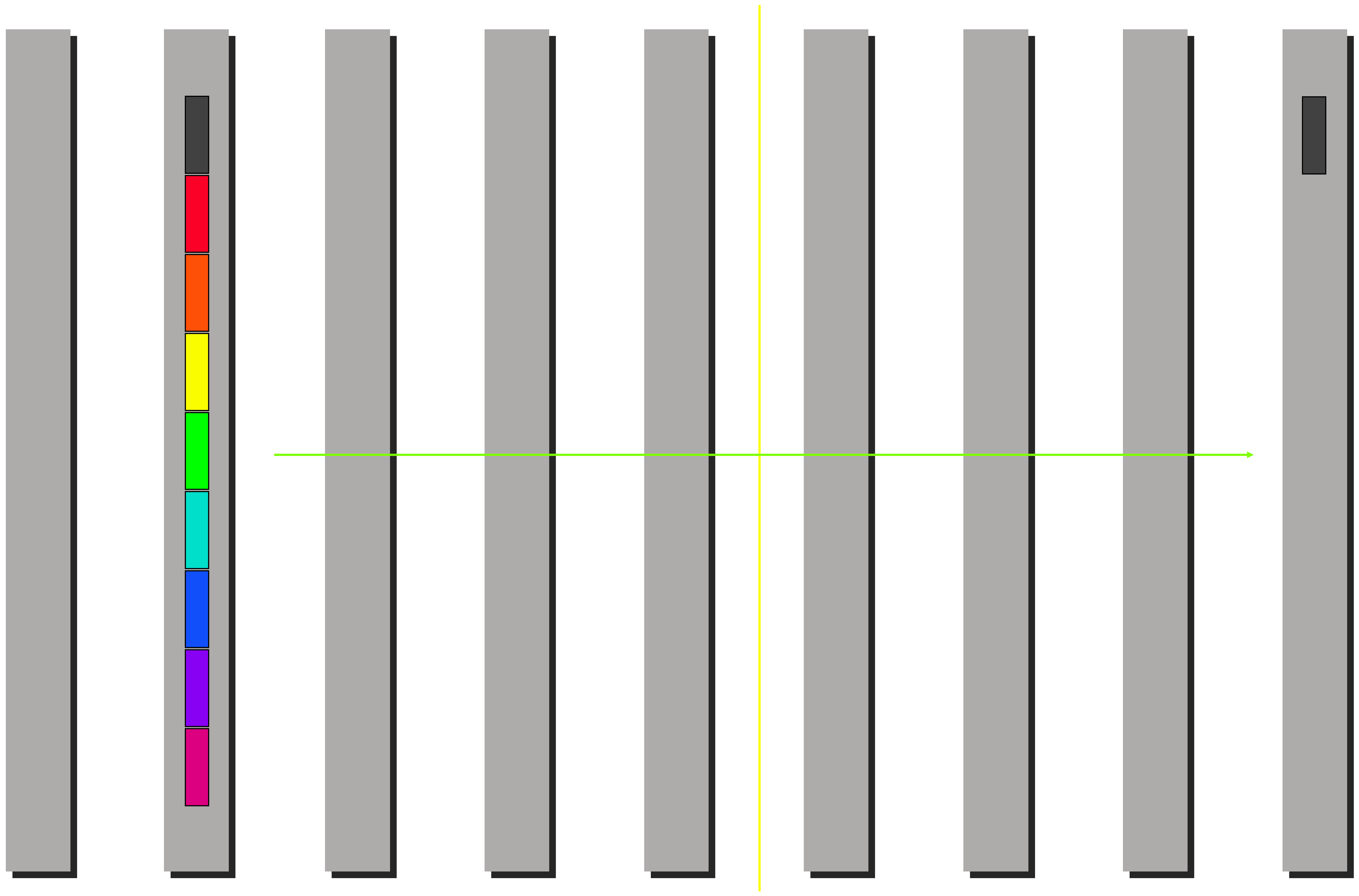
Let us view this more closely

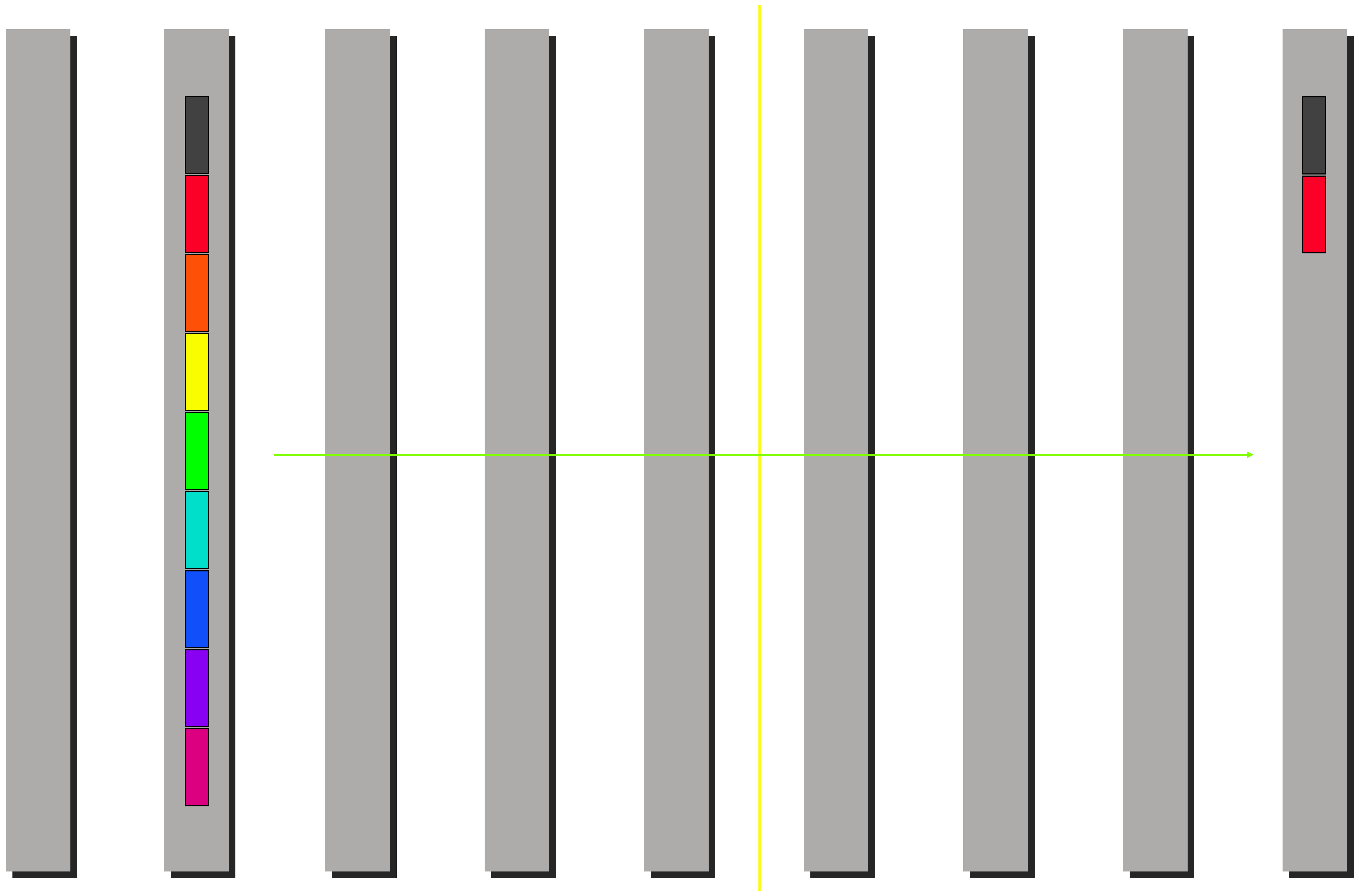
- Red arrows indicate startup of communication (leading to latency, α)

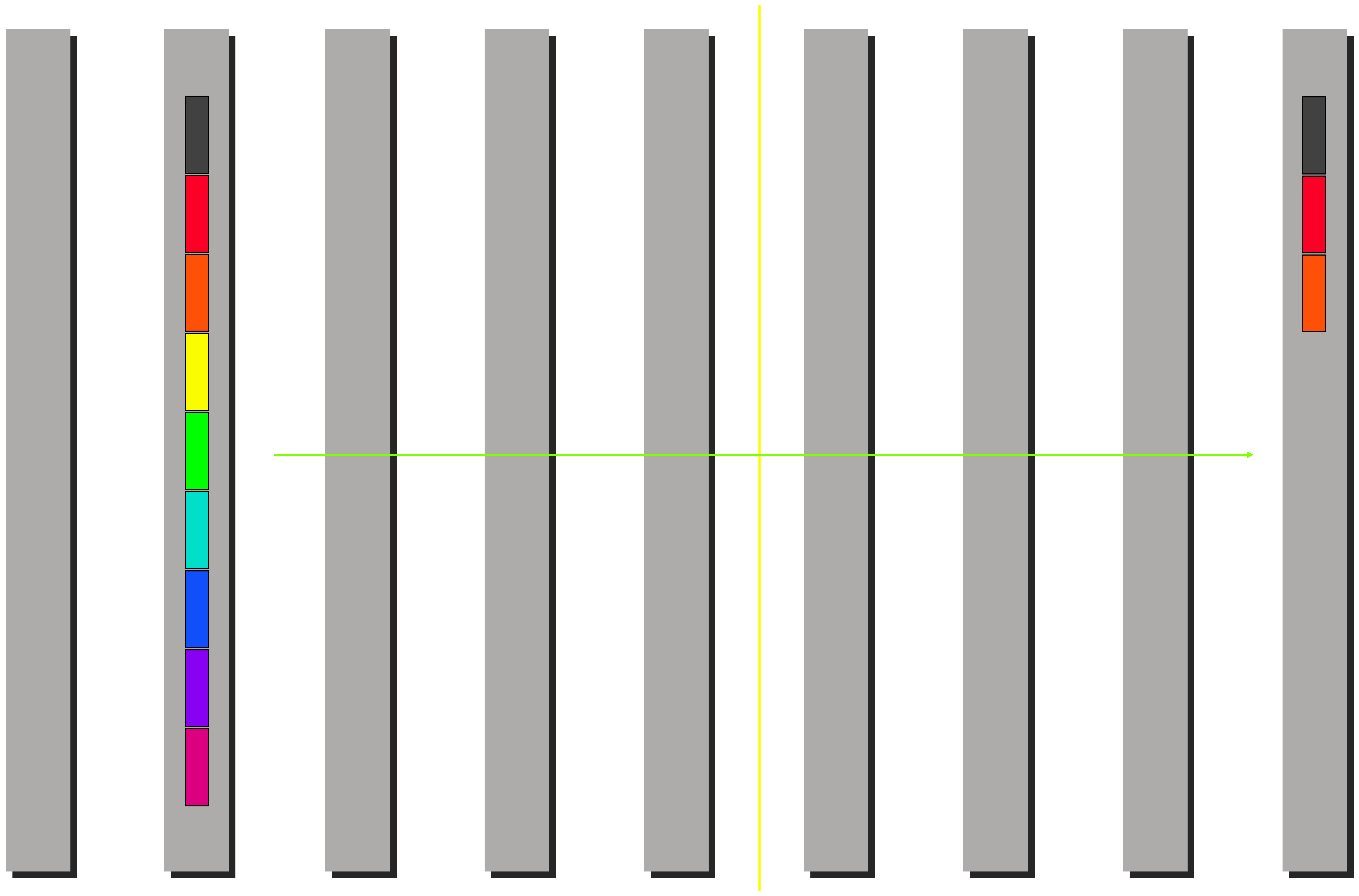
- Green arrows indicate packets in transit (leading to a bandwidth related cost proportional to β and the length of the packet

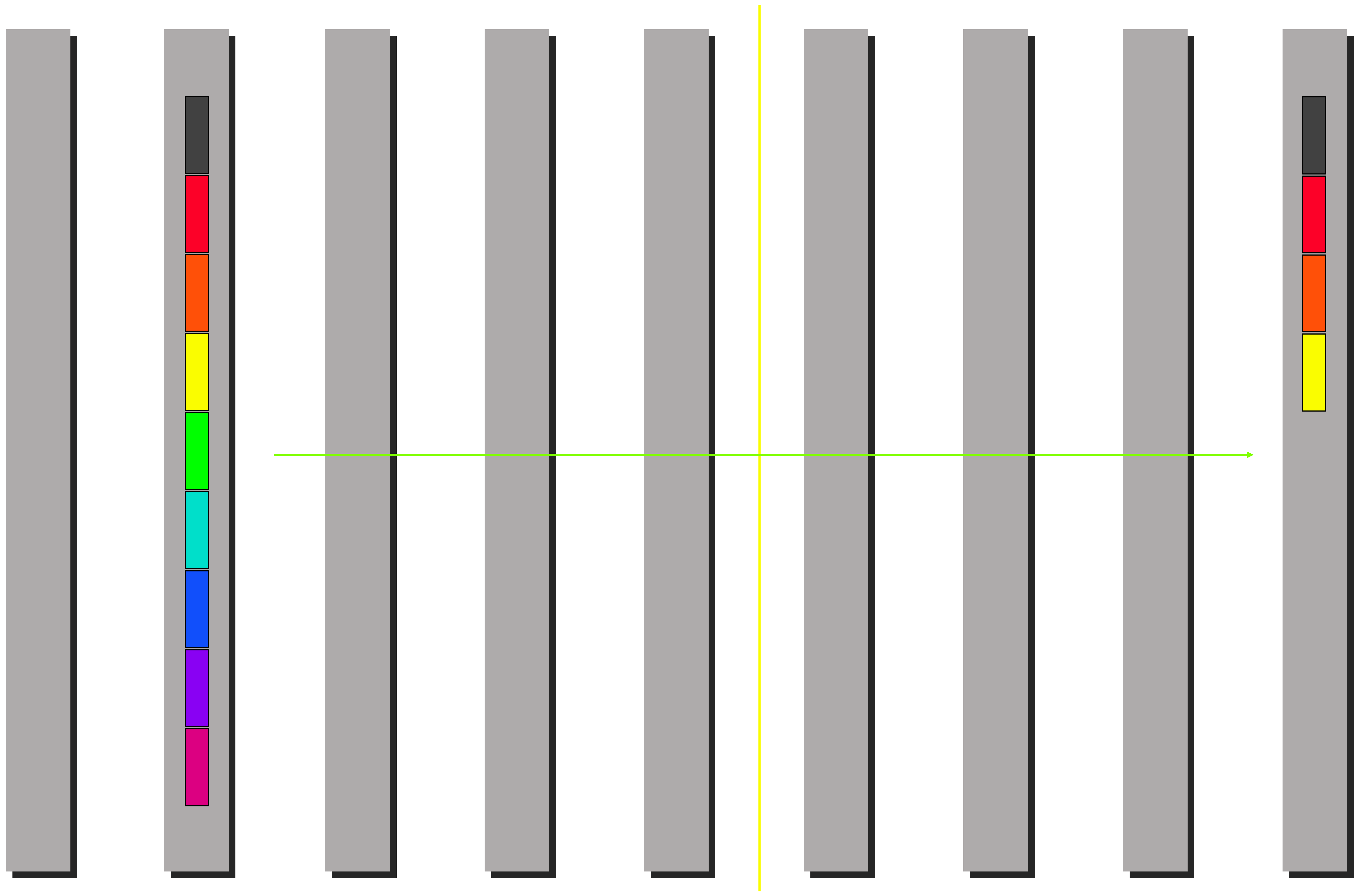


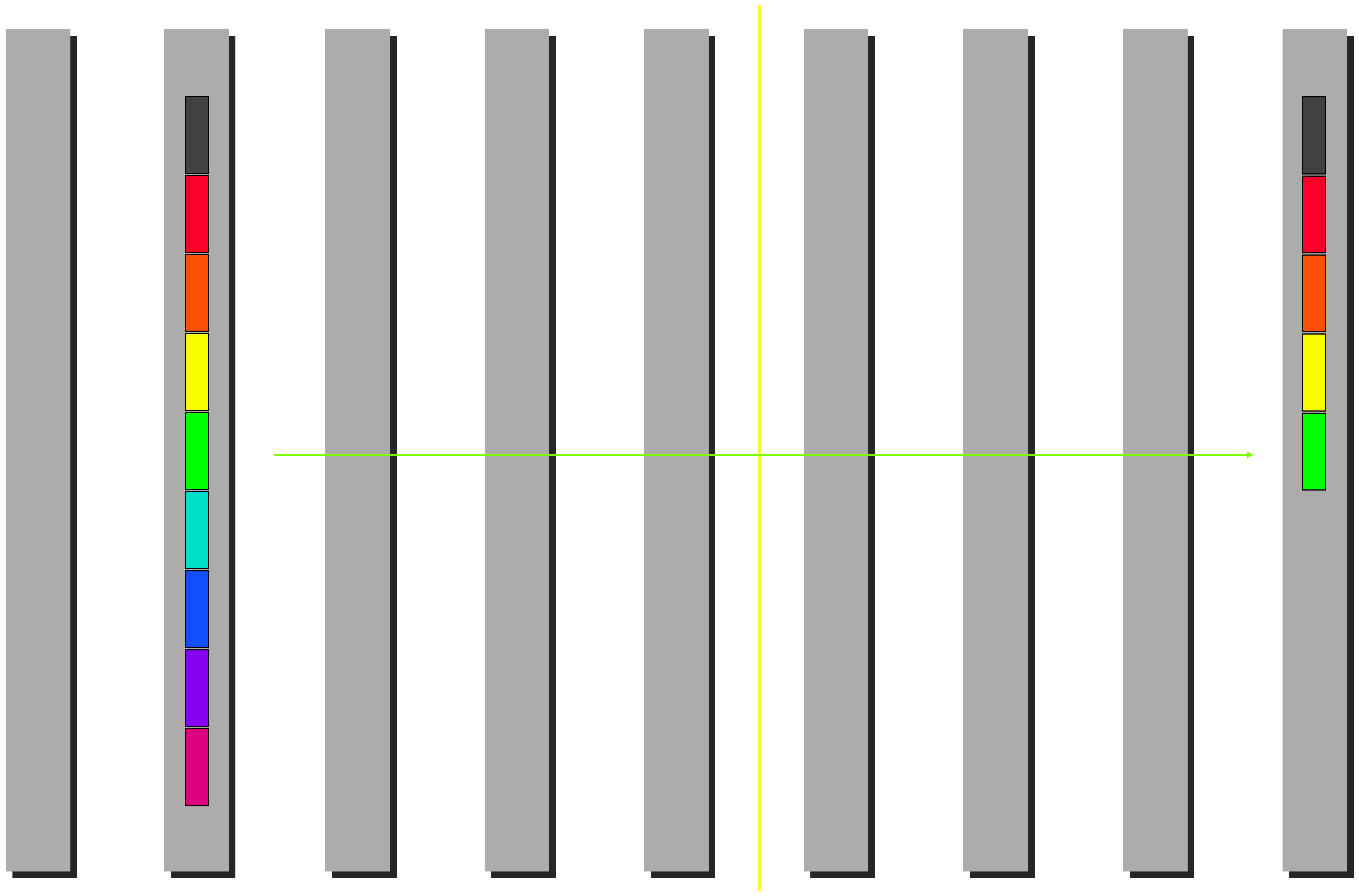


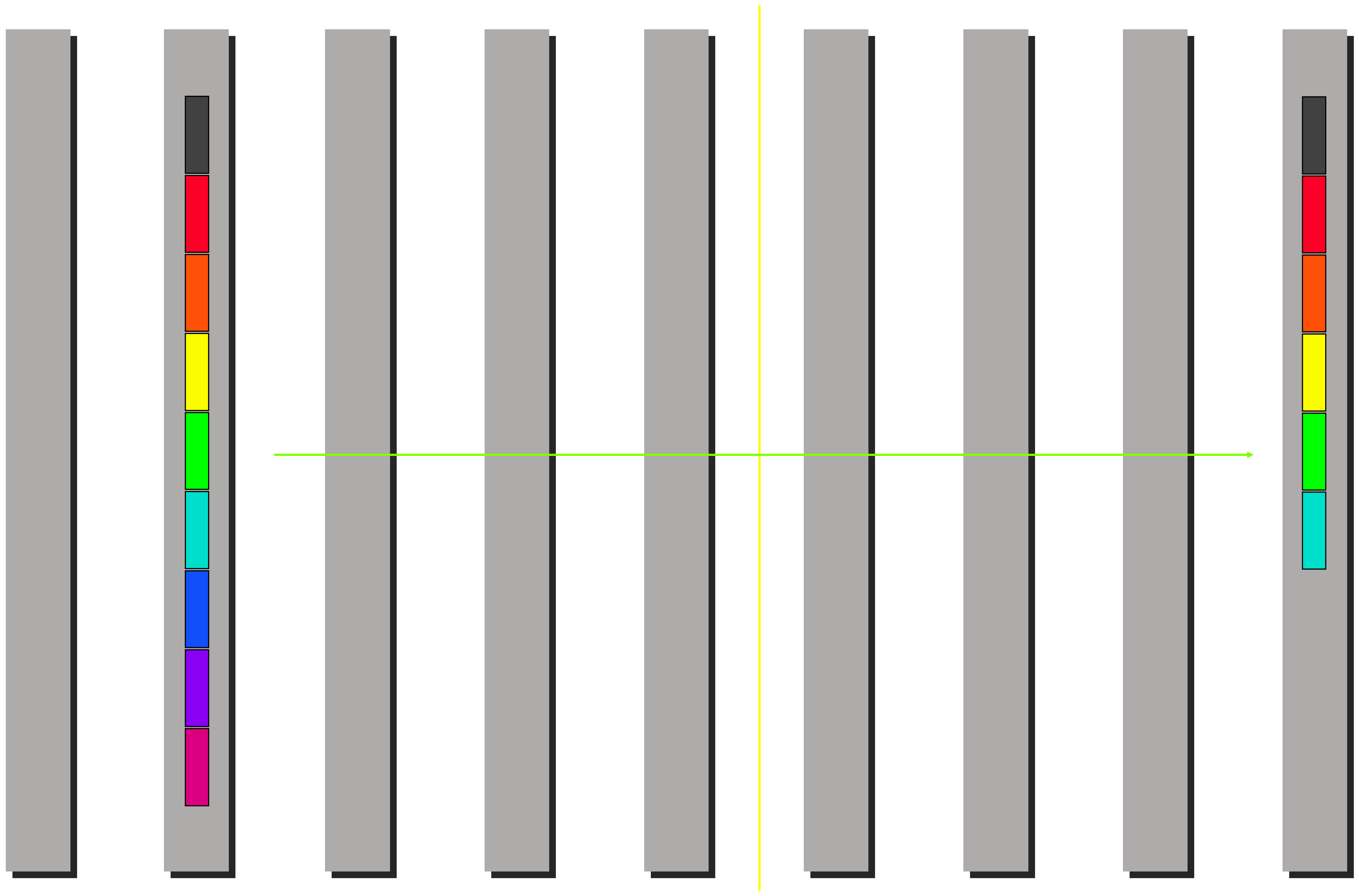


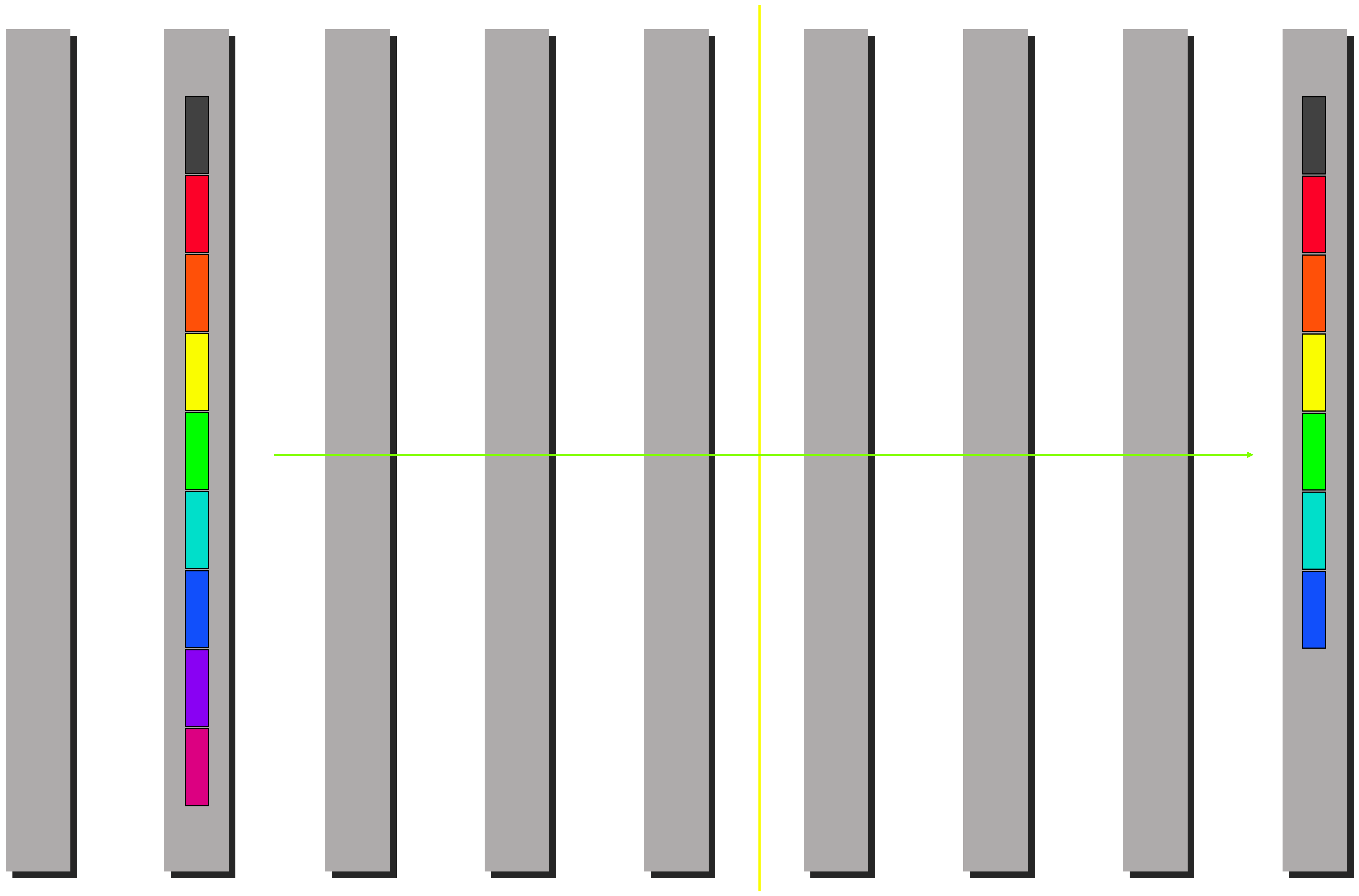


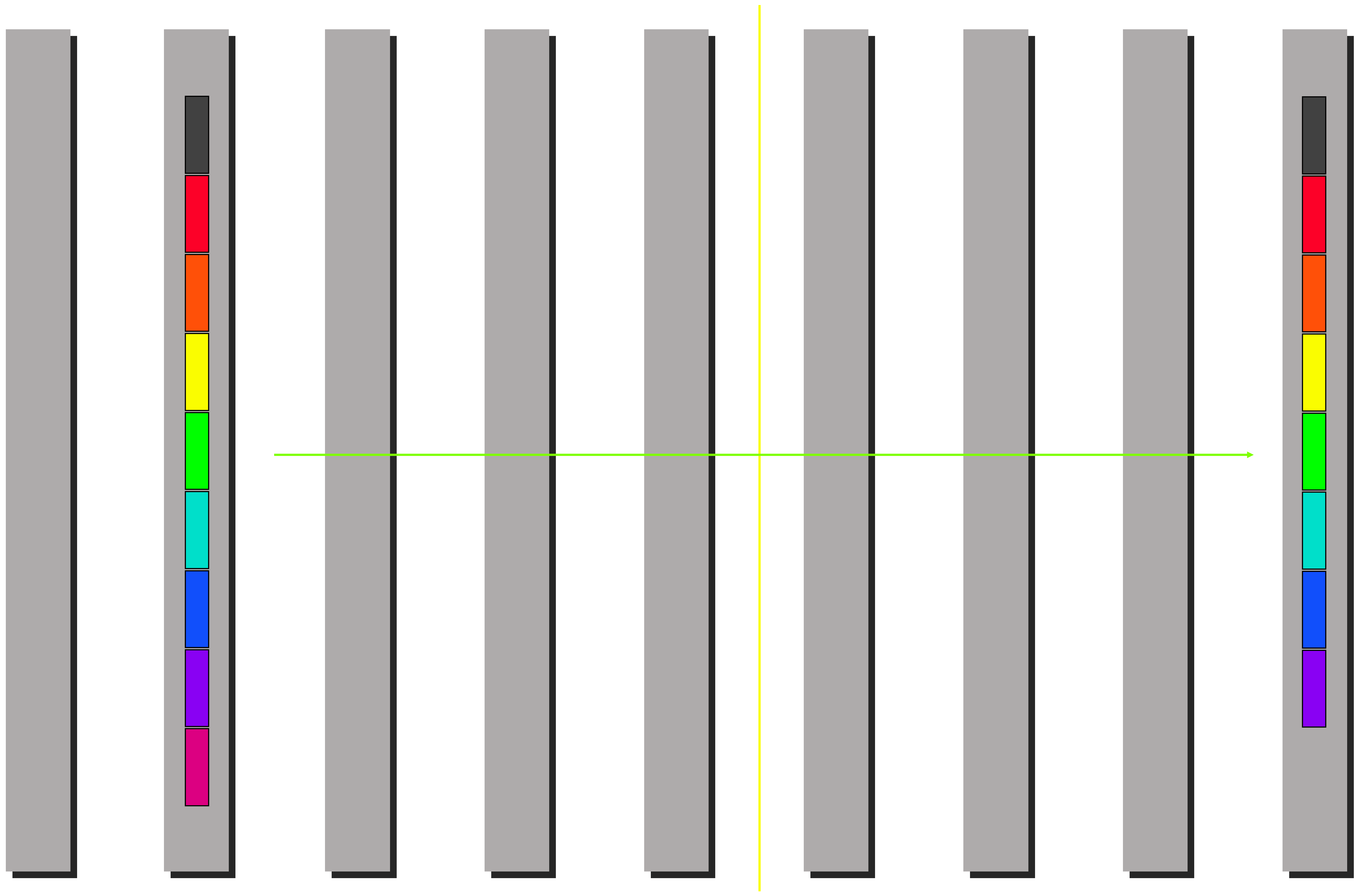


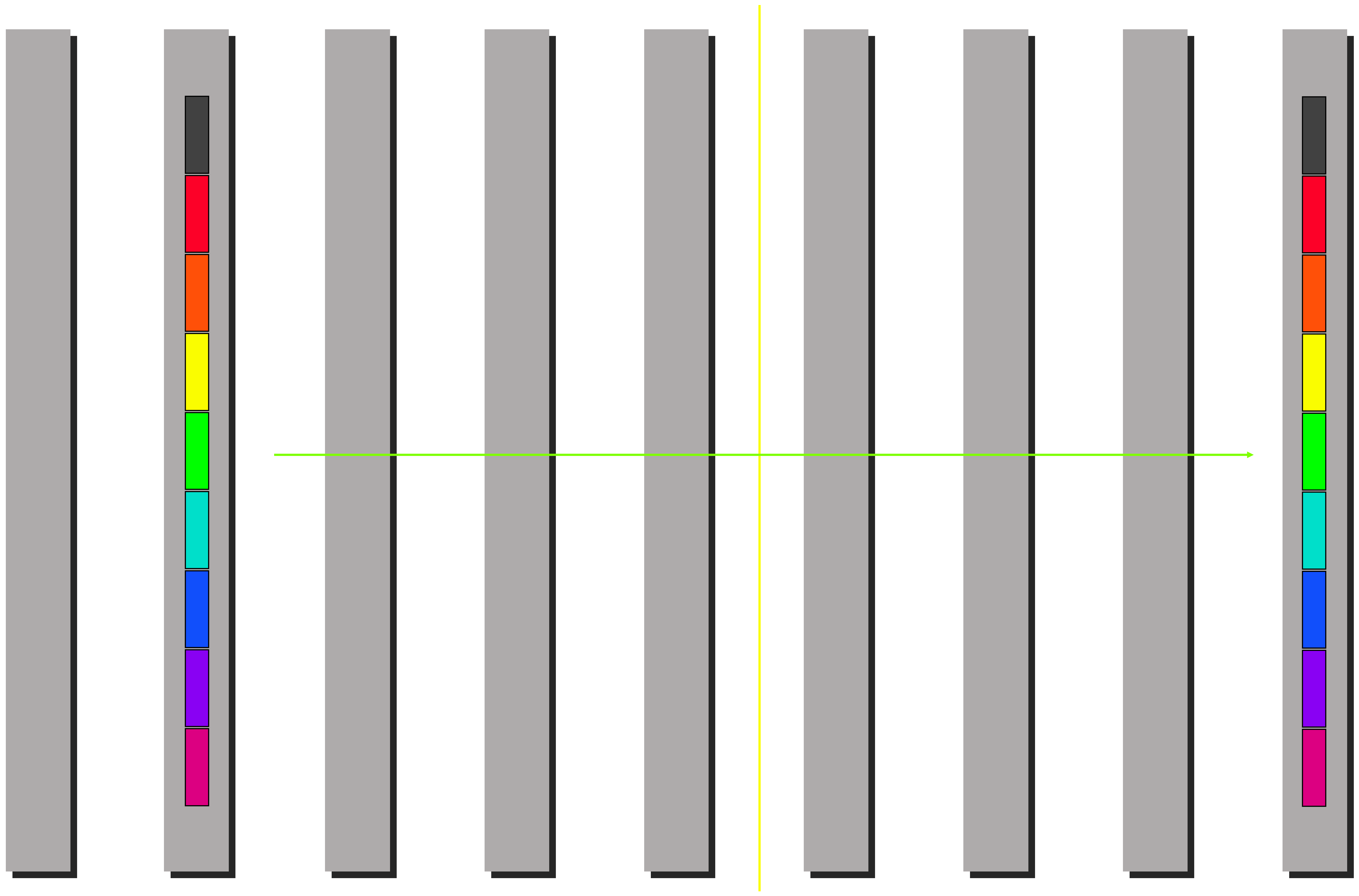


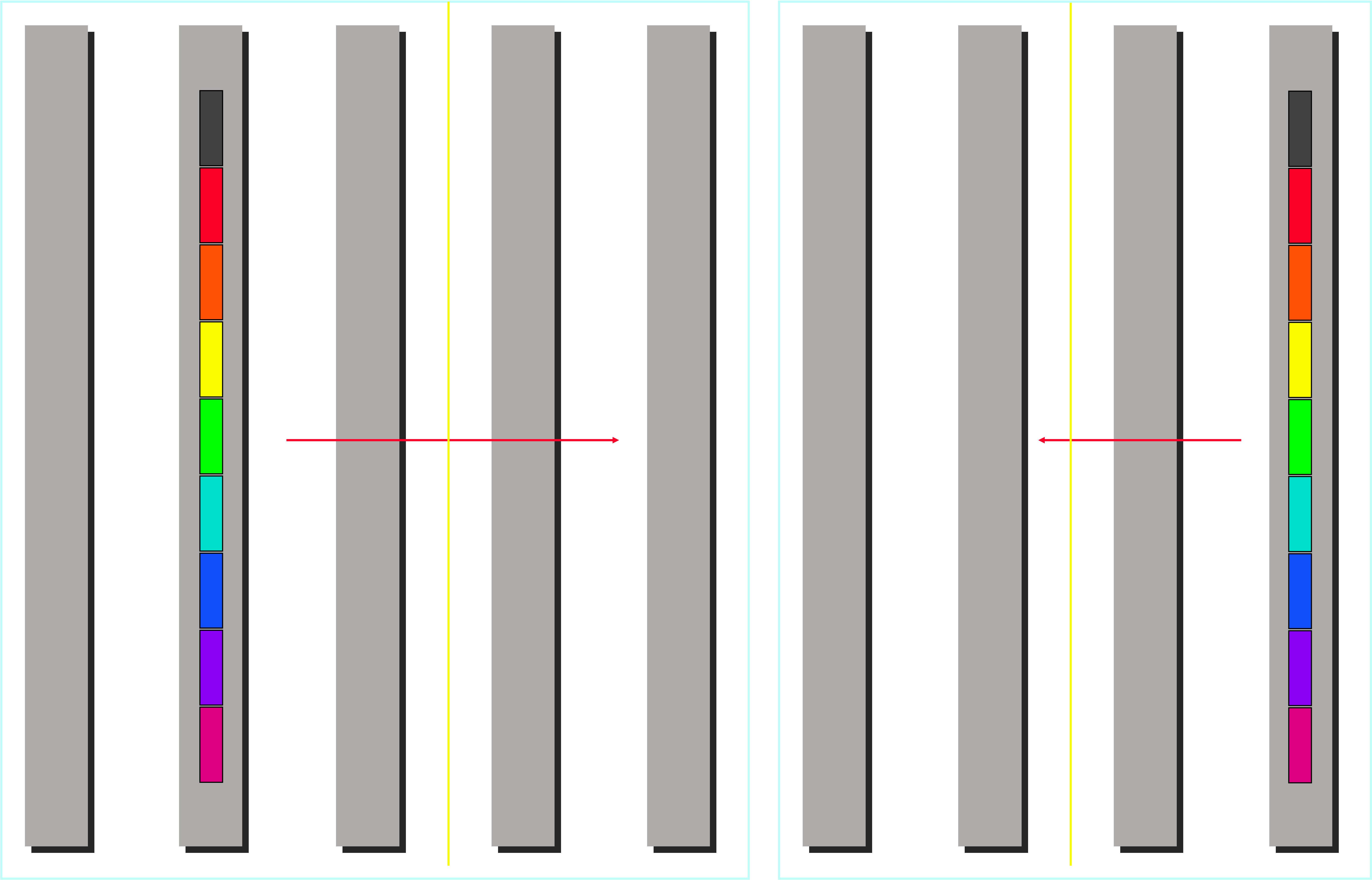


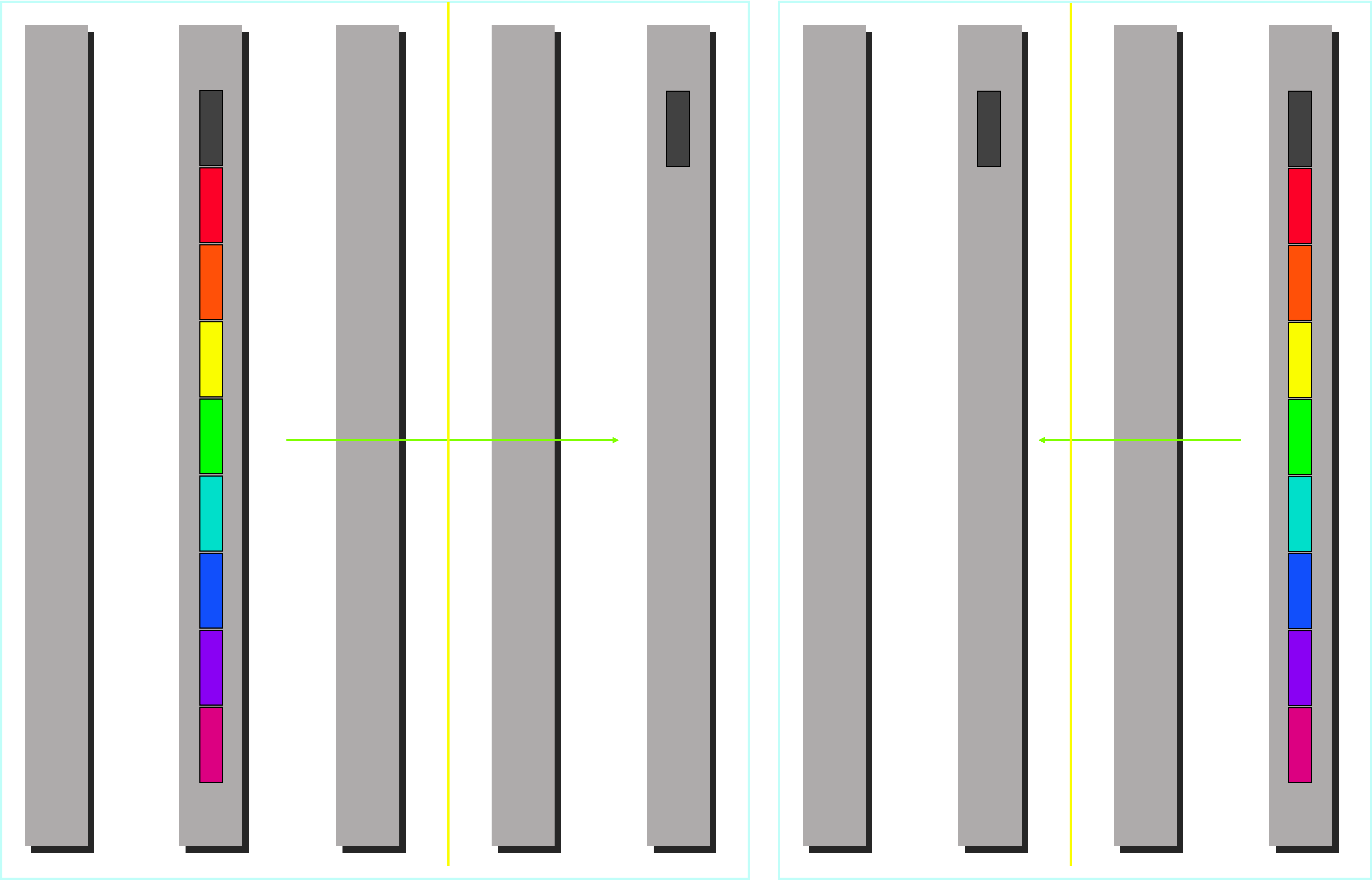


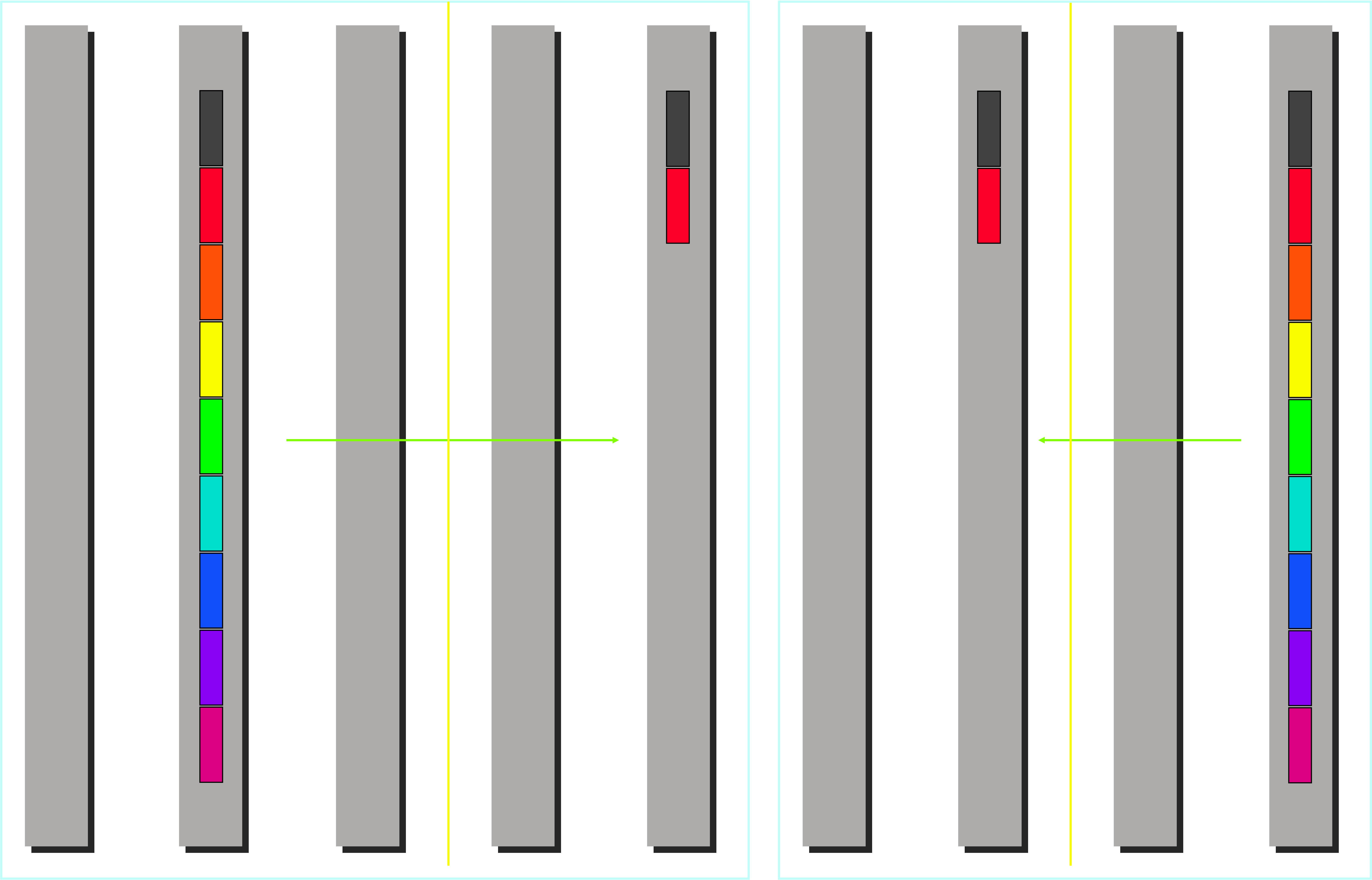


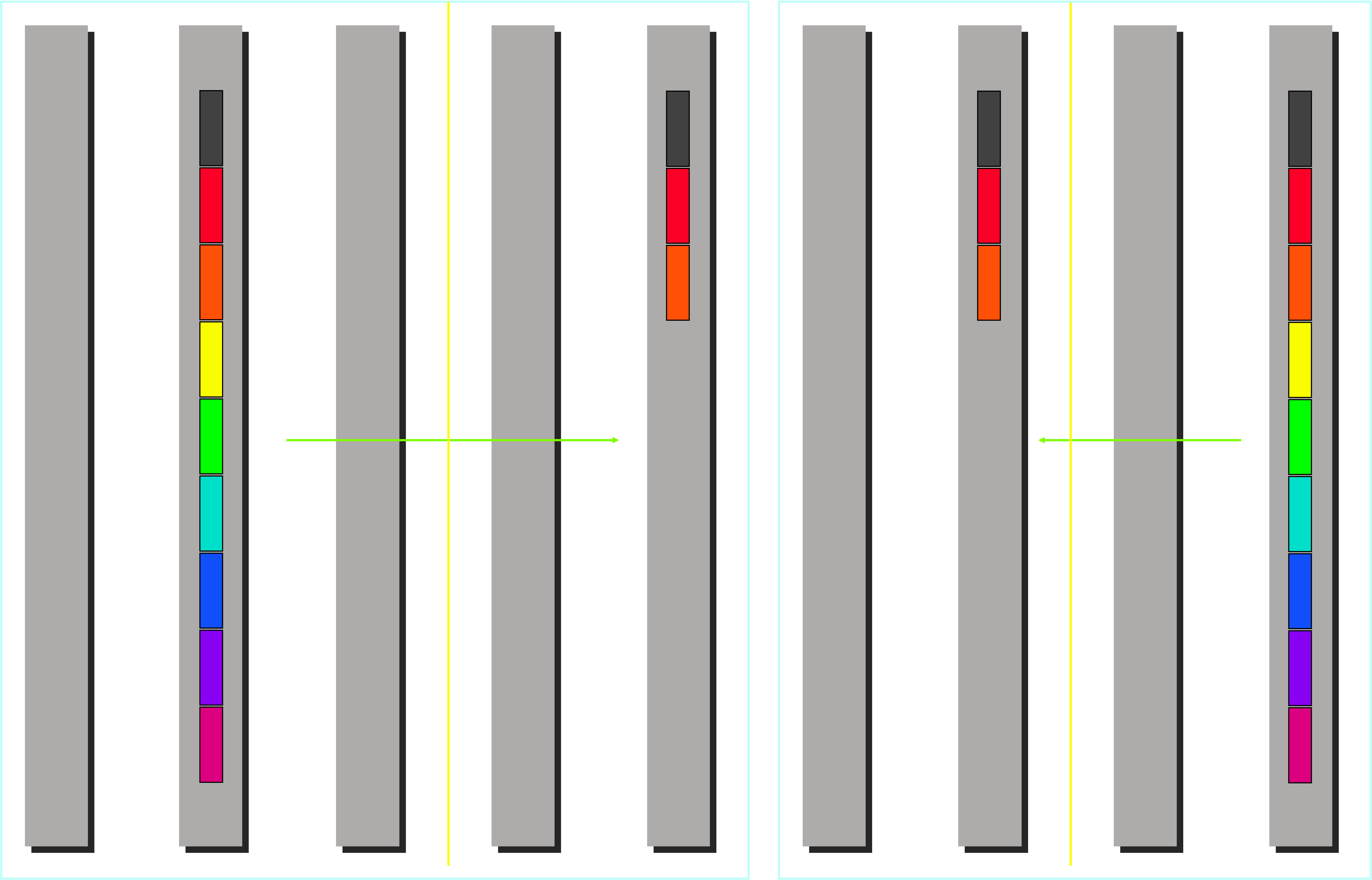


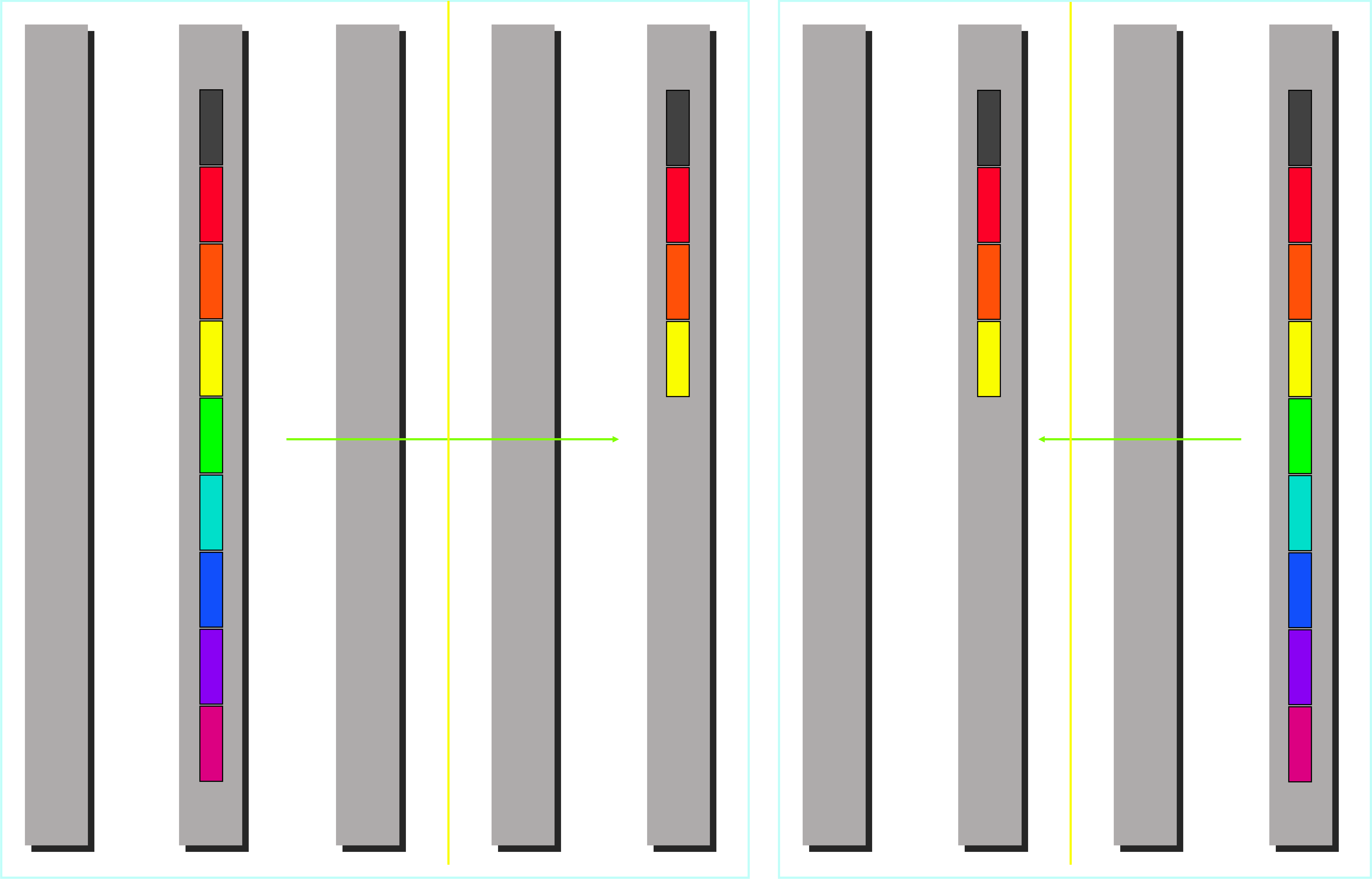


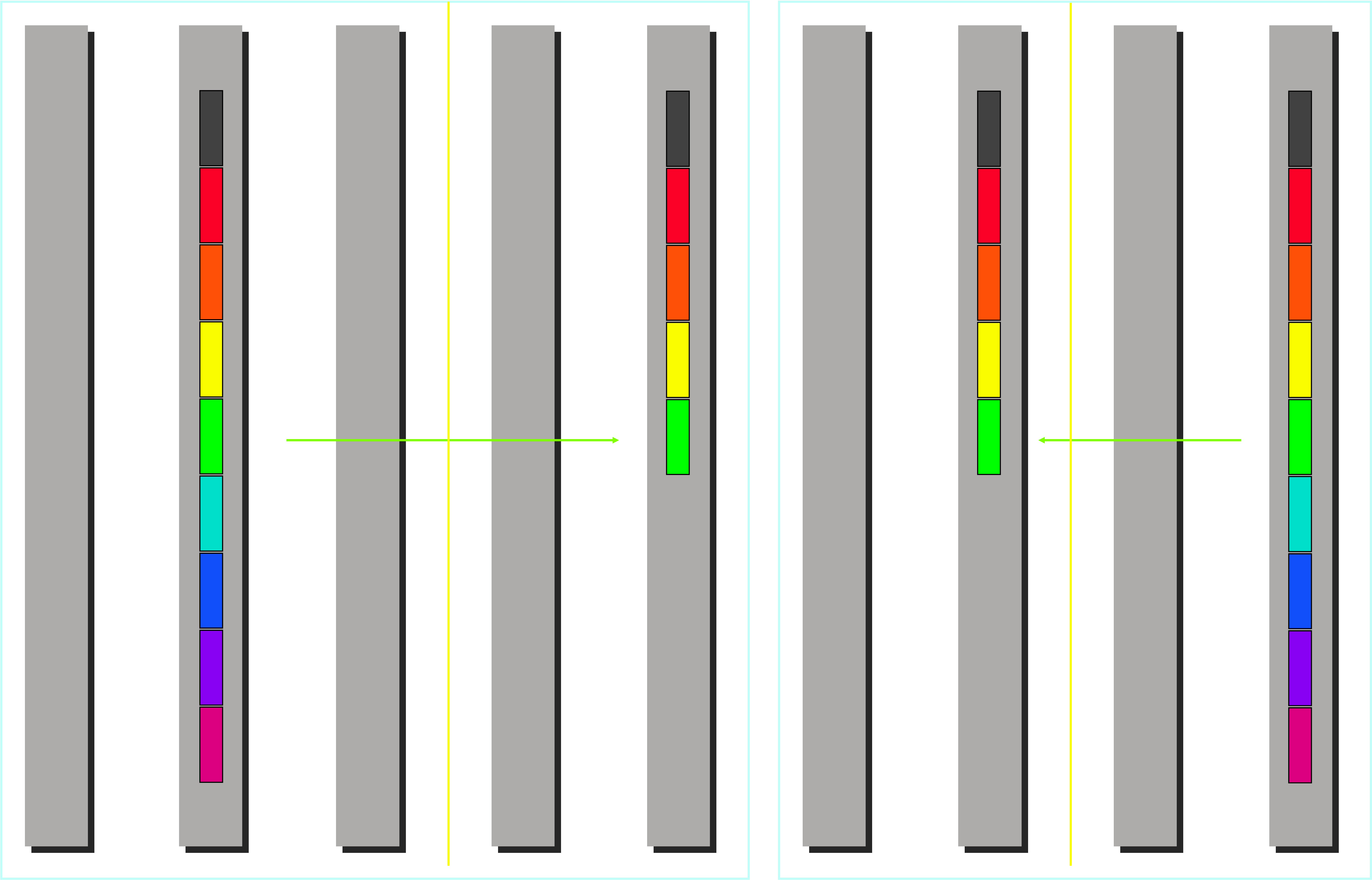


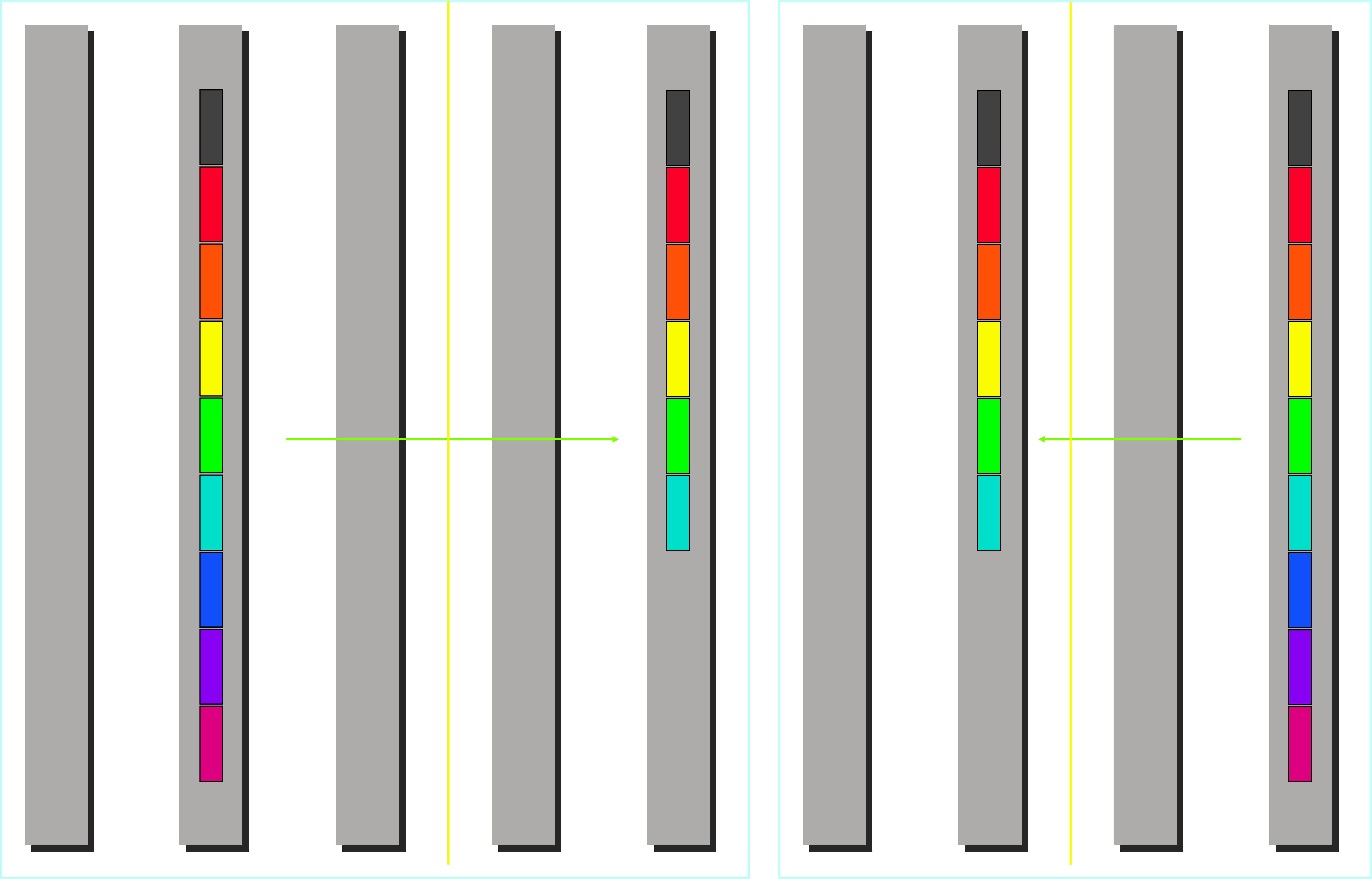


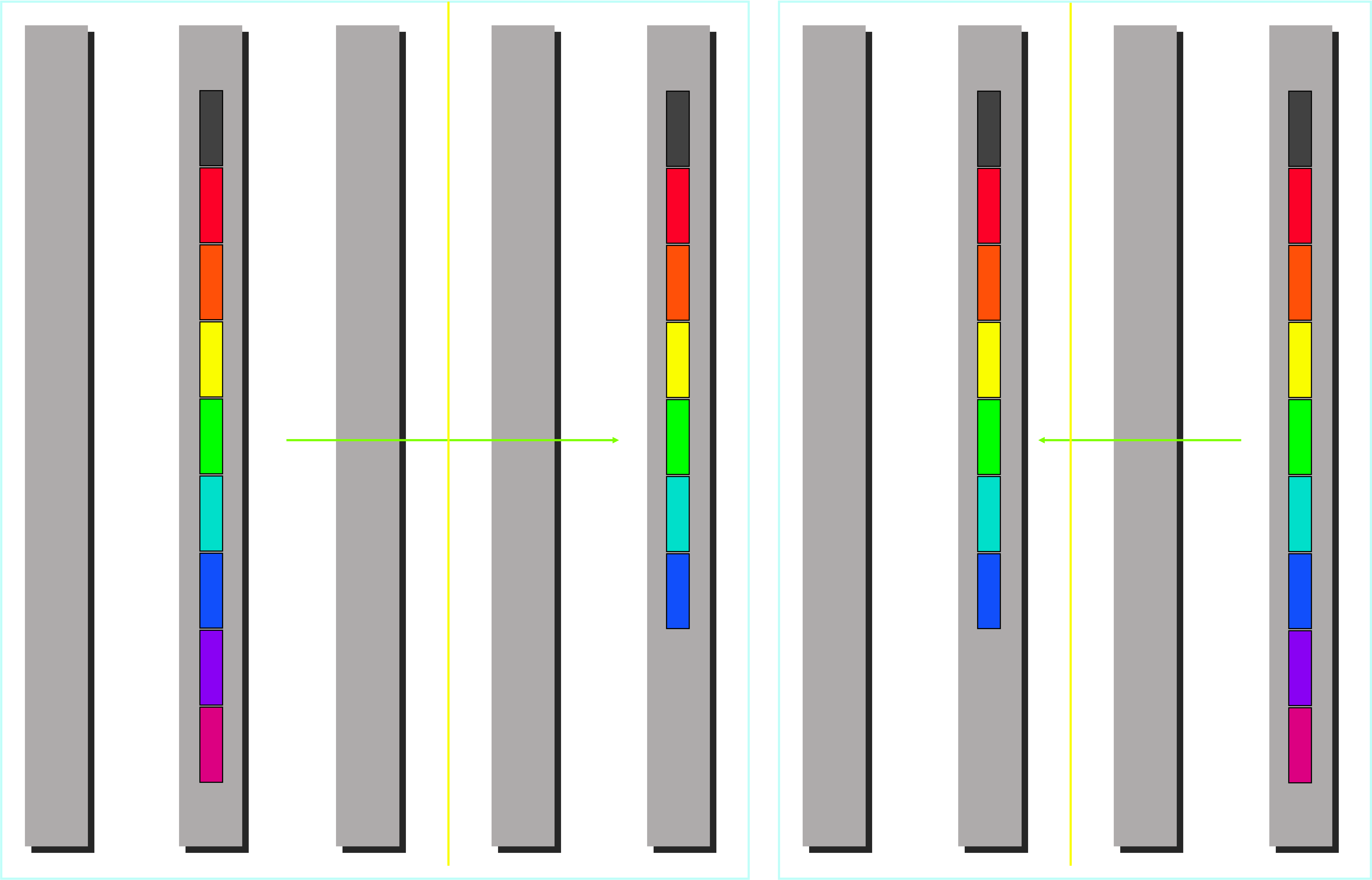


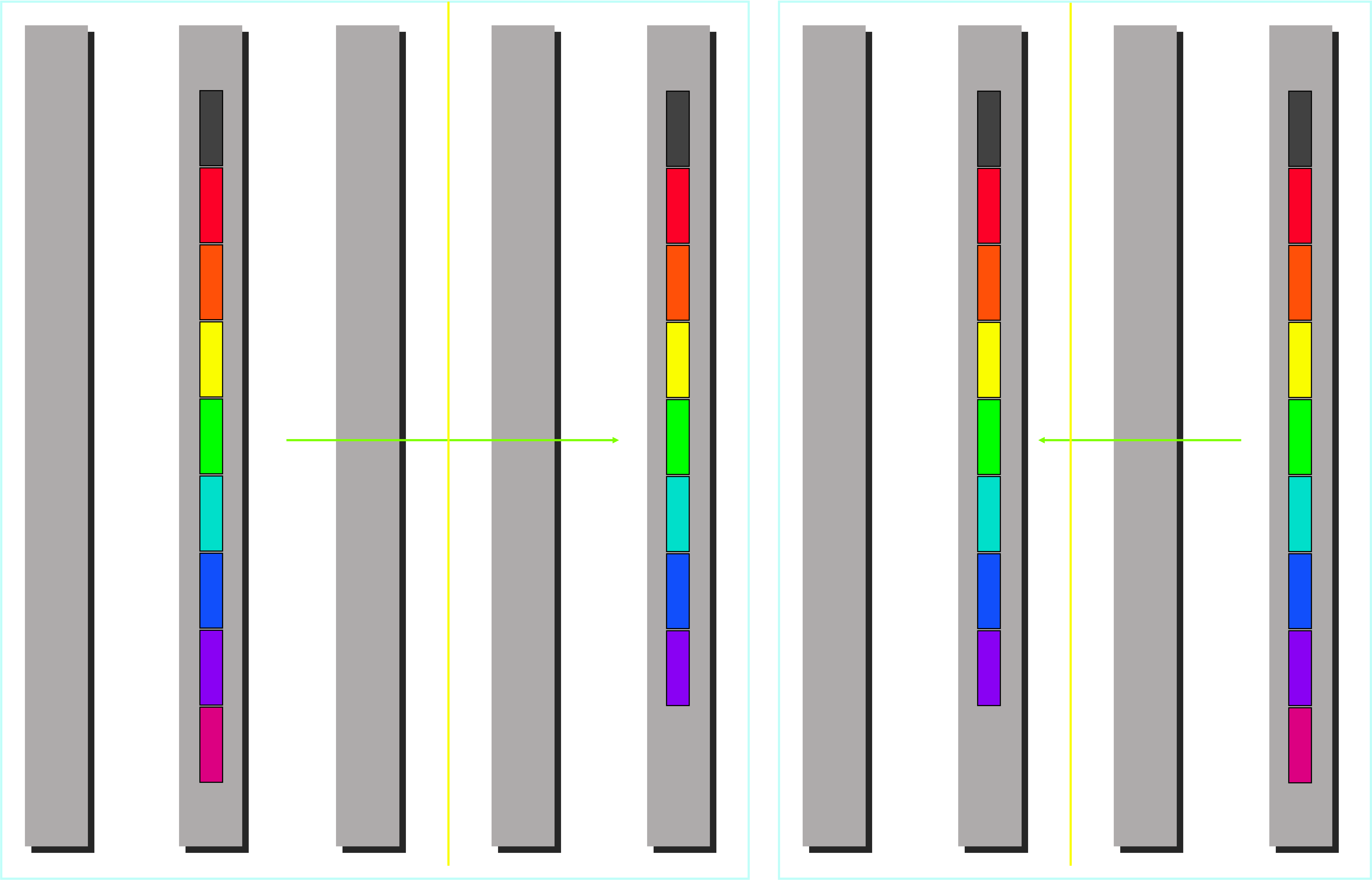


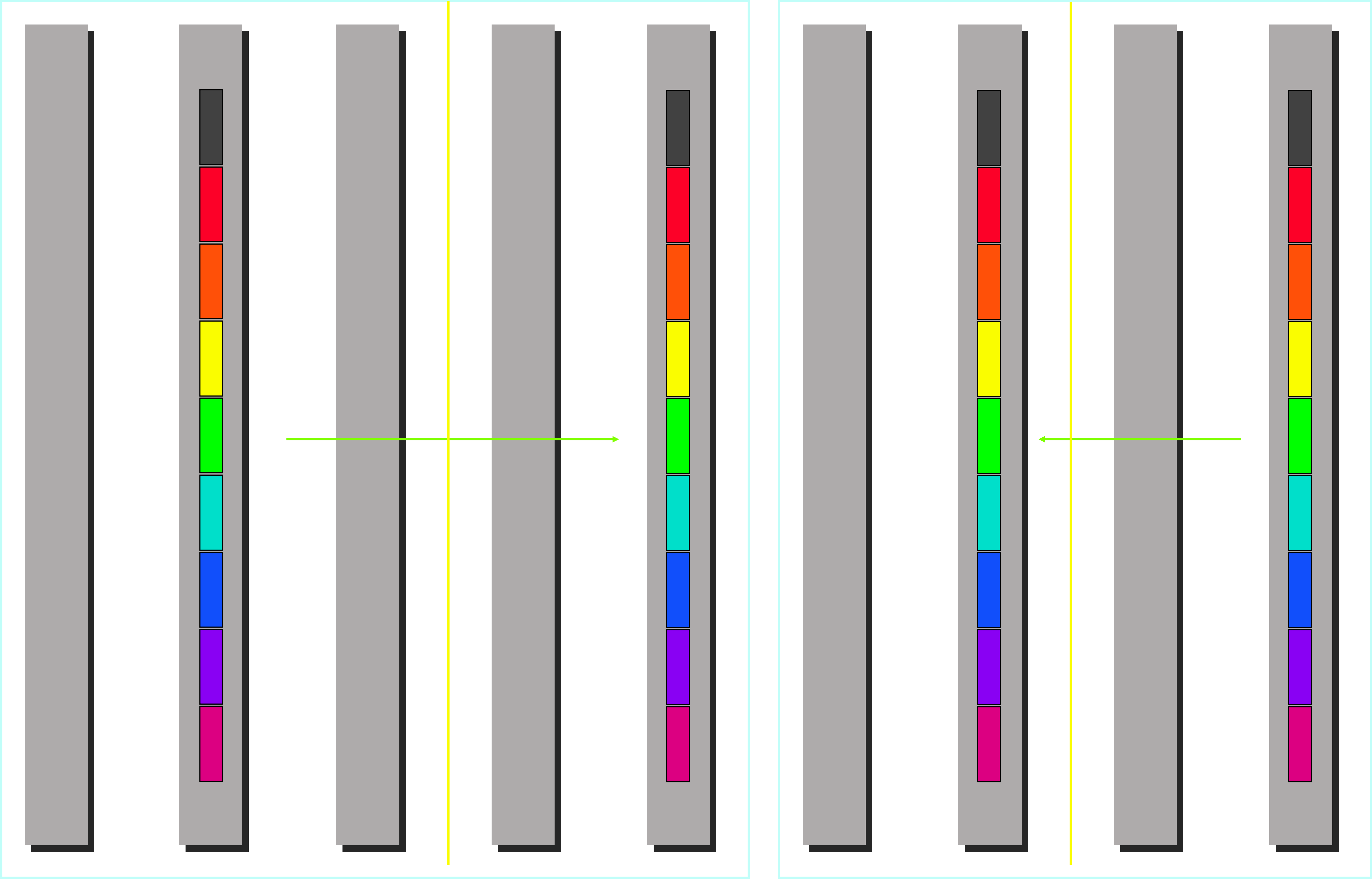


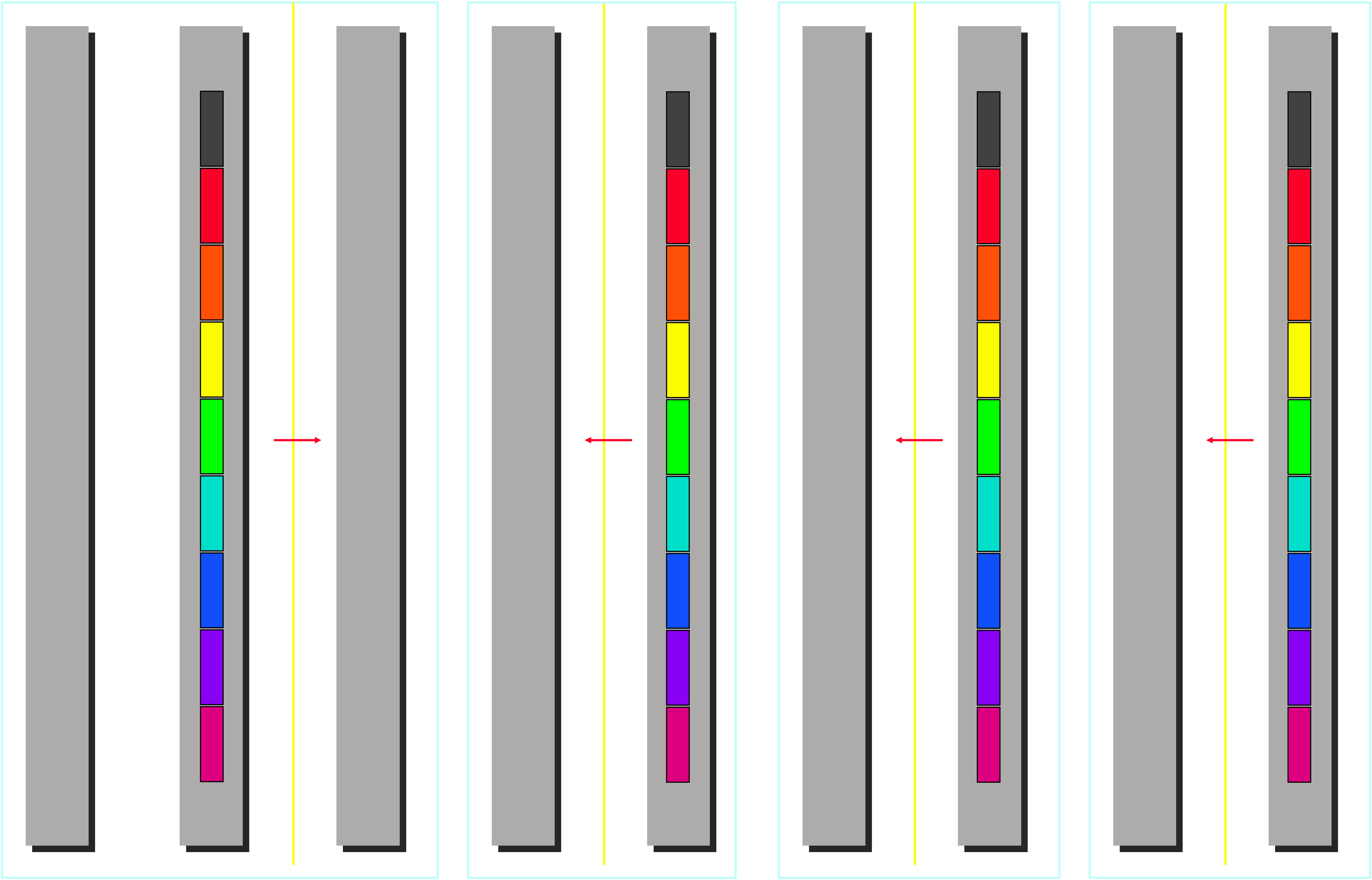


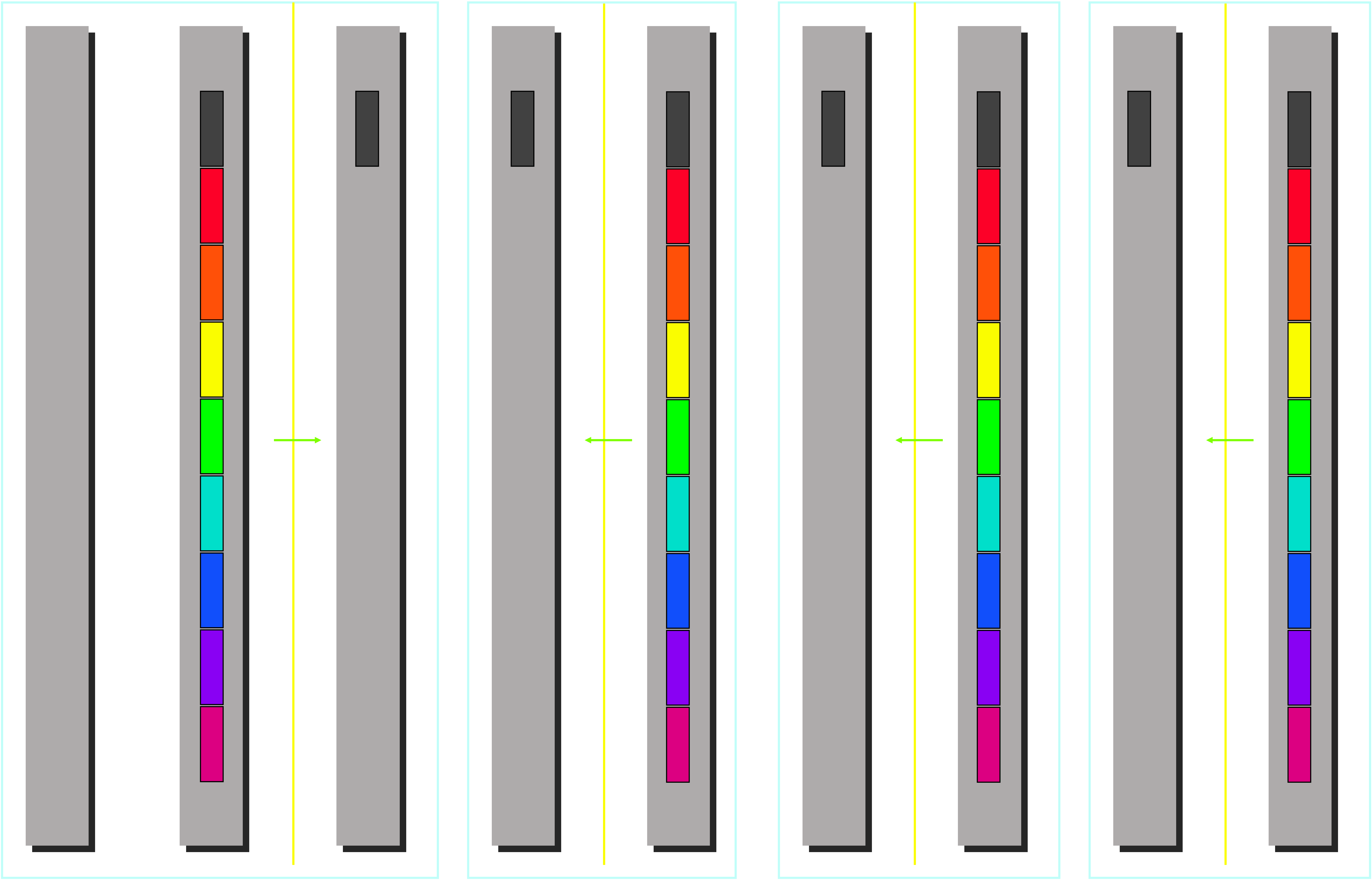


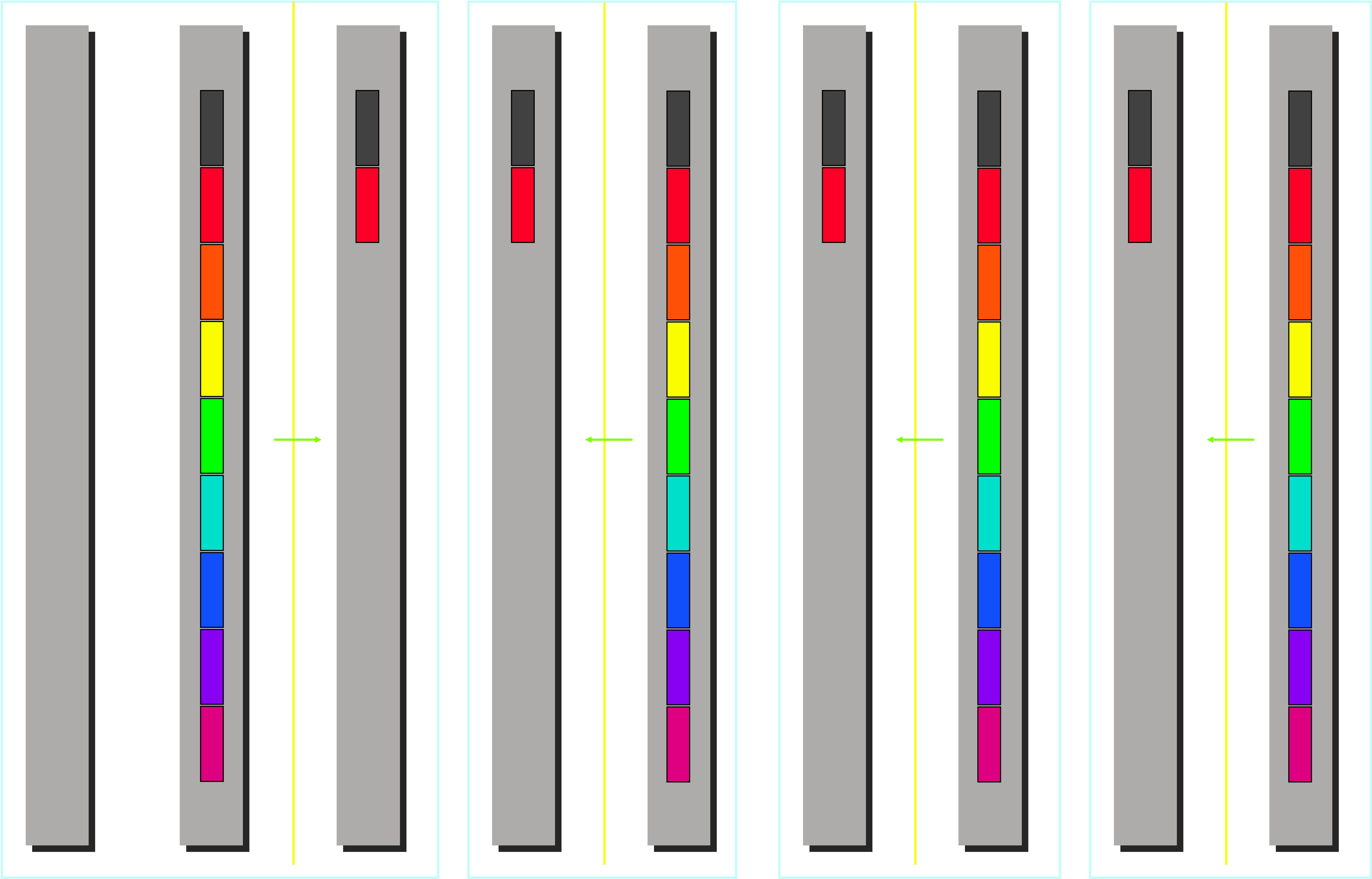


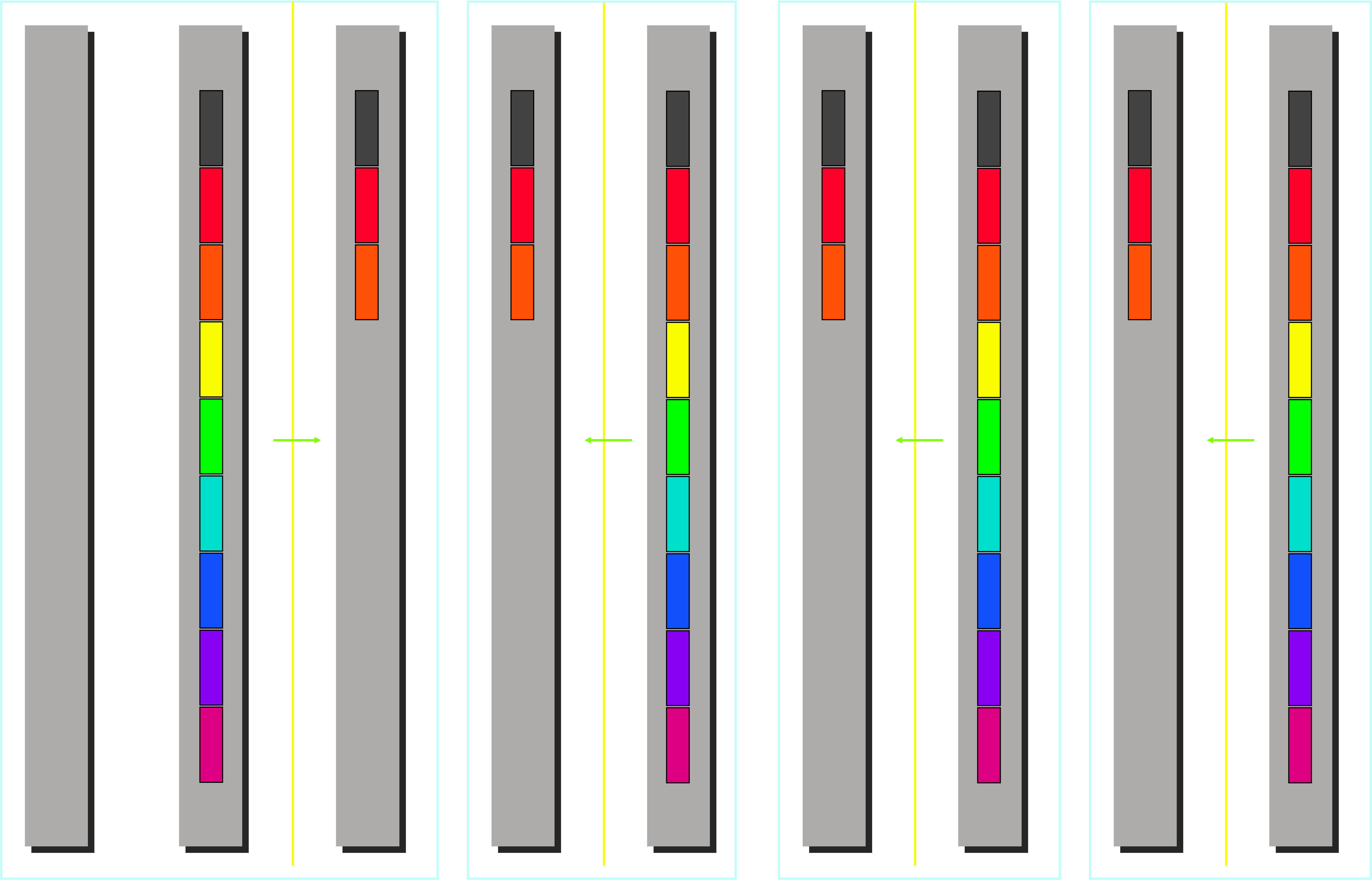


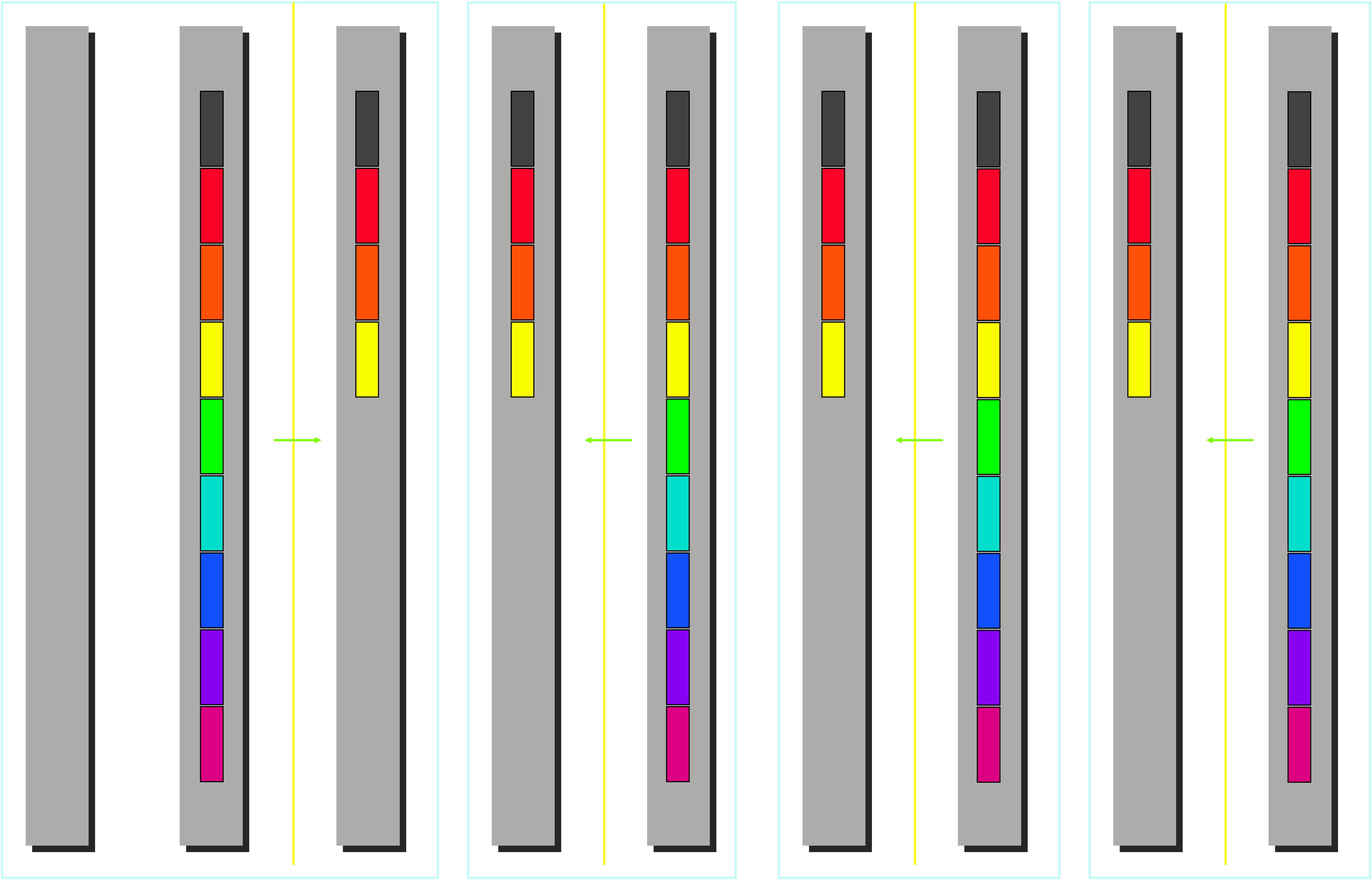


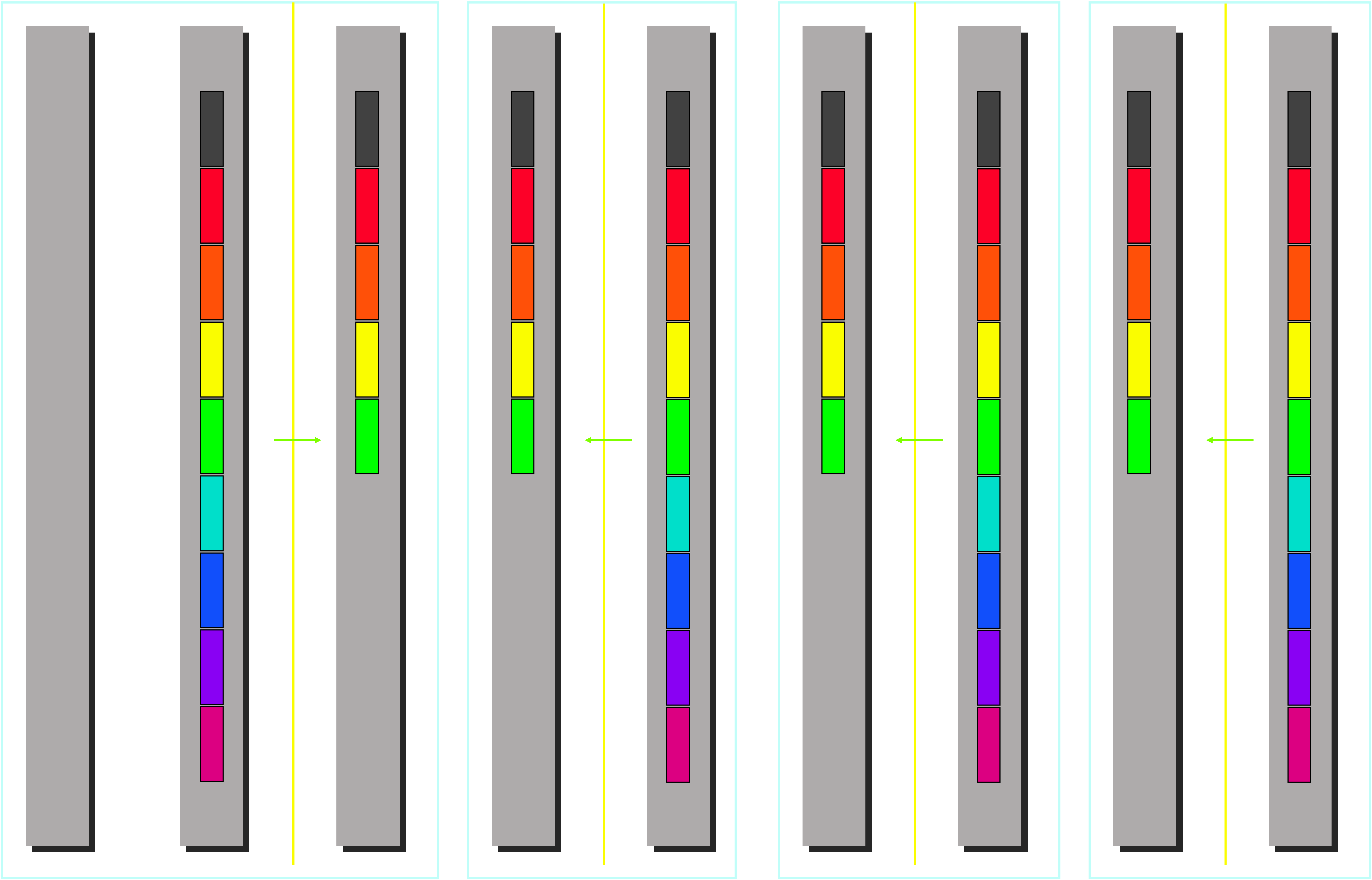


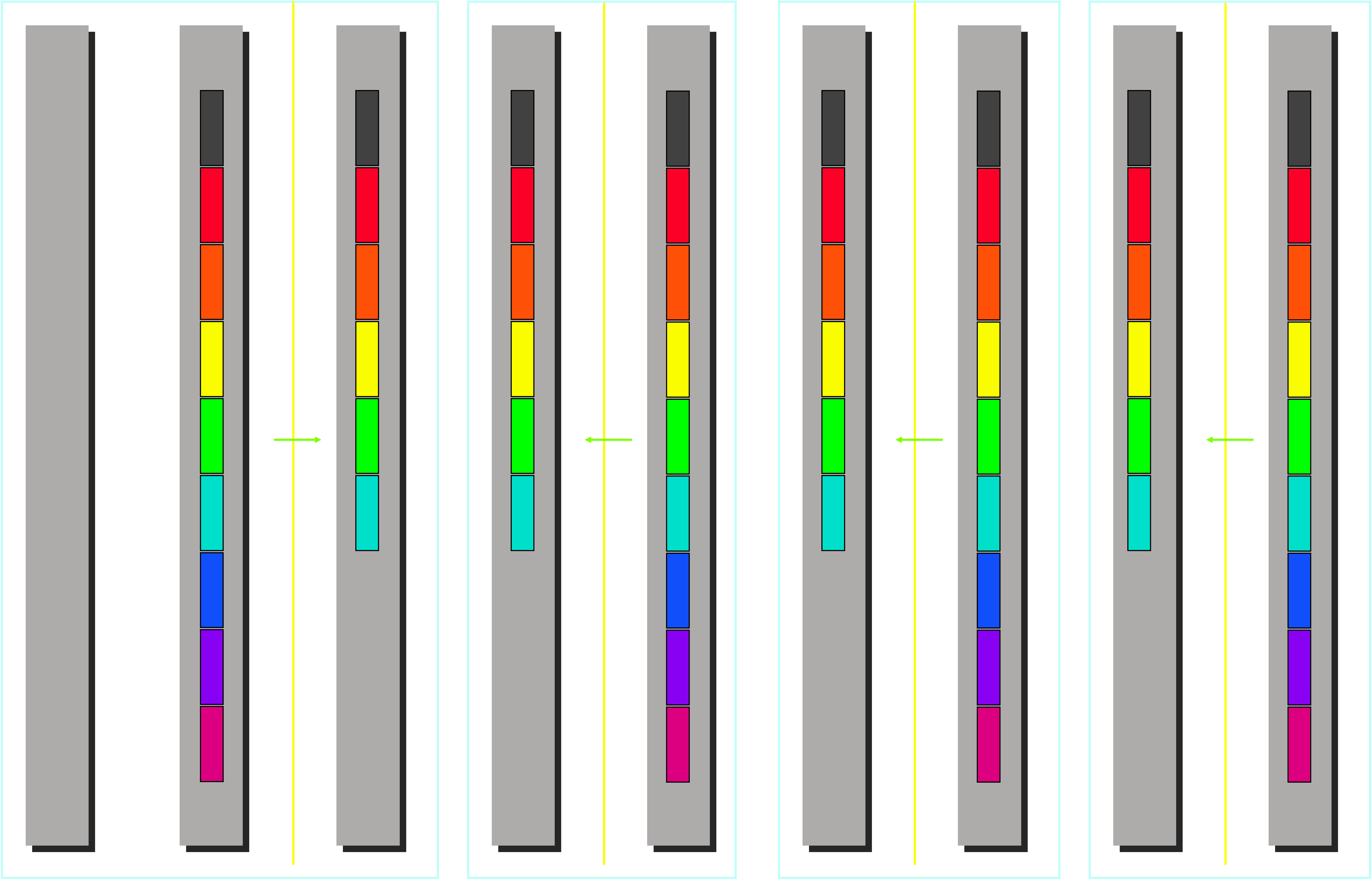


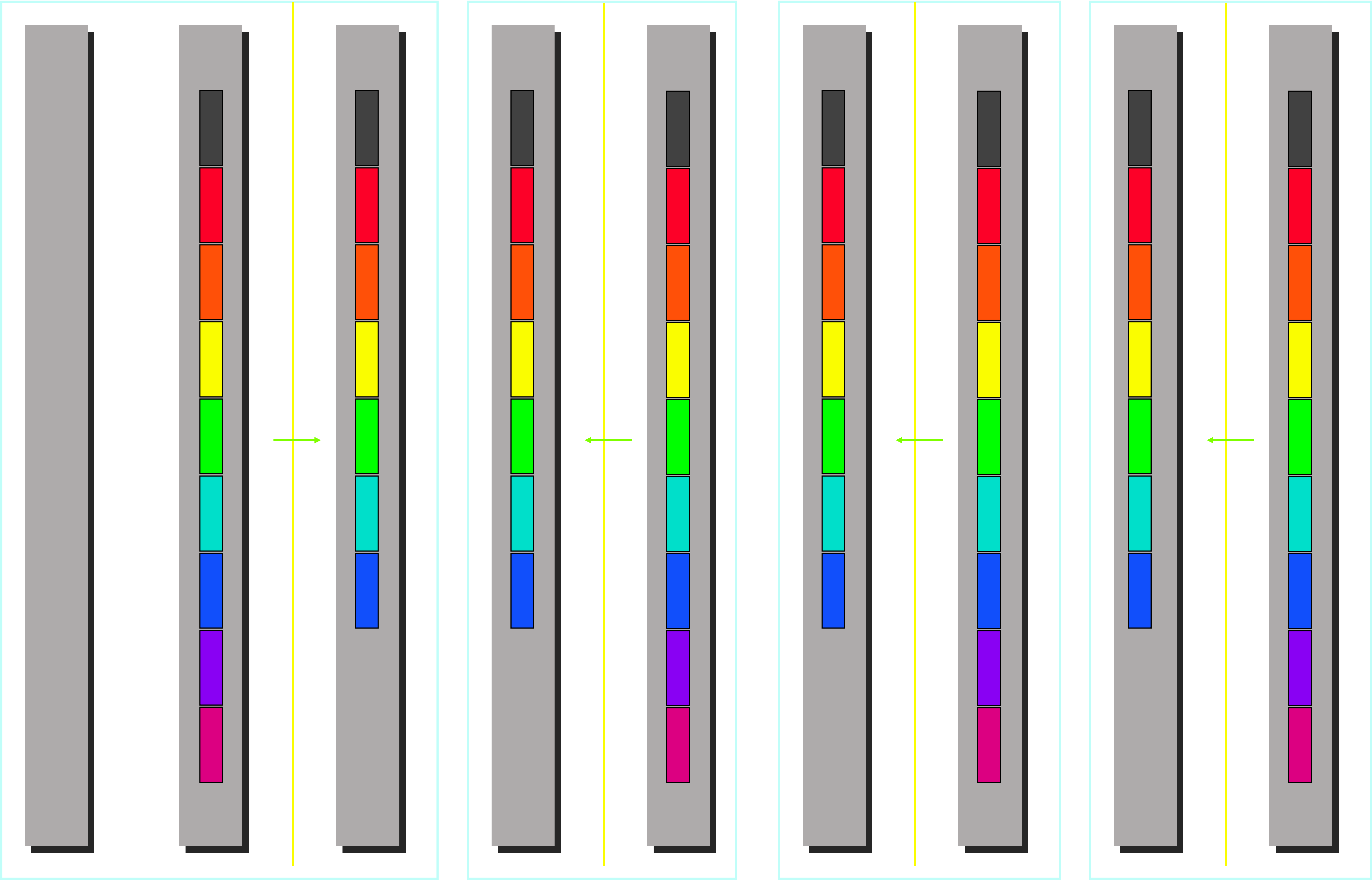


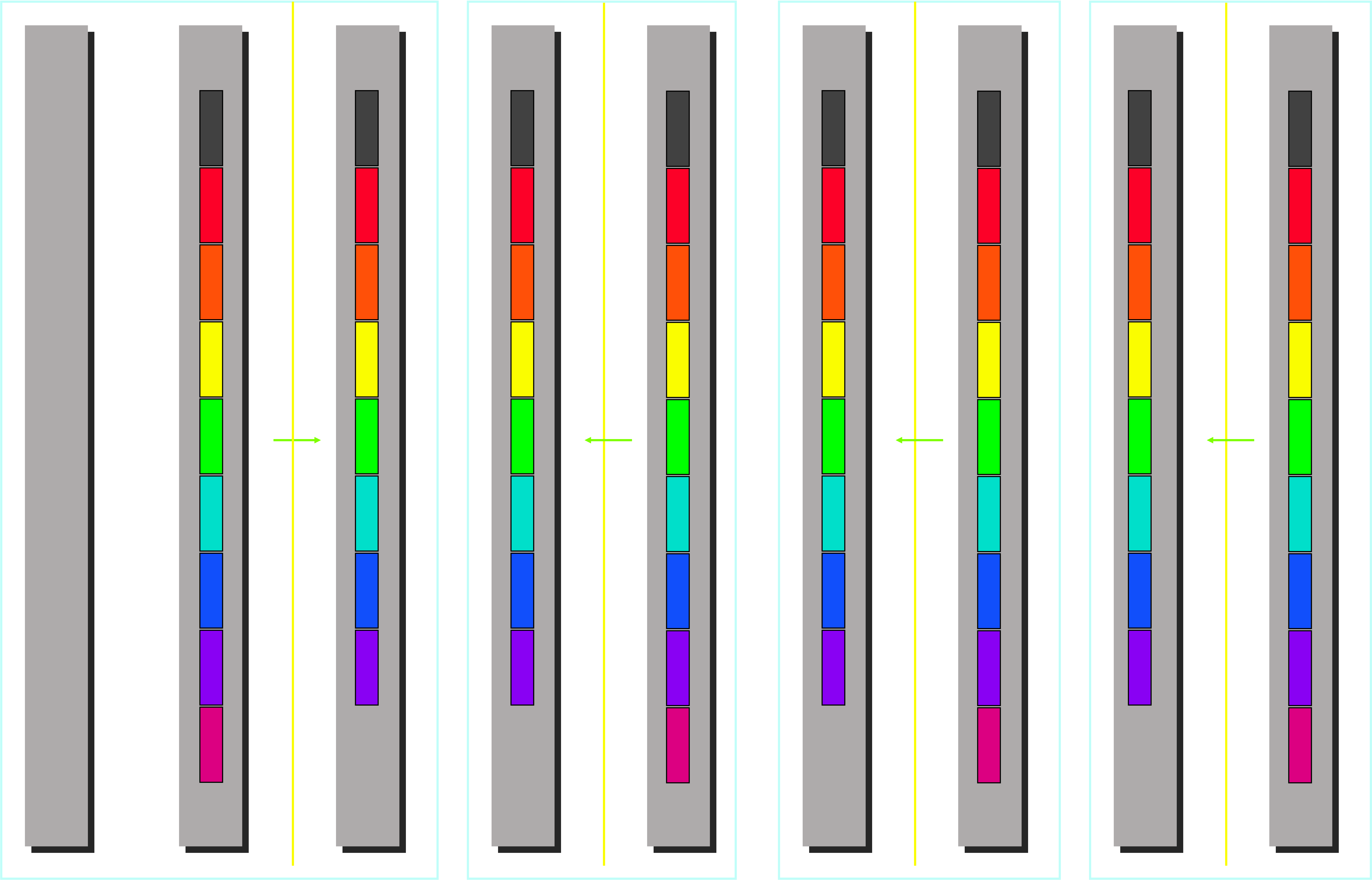


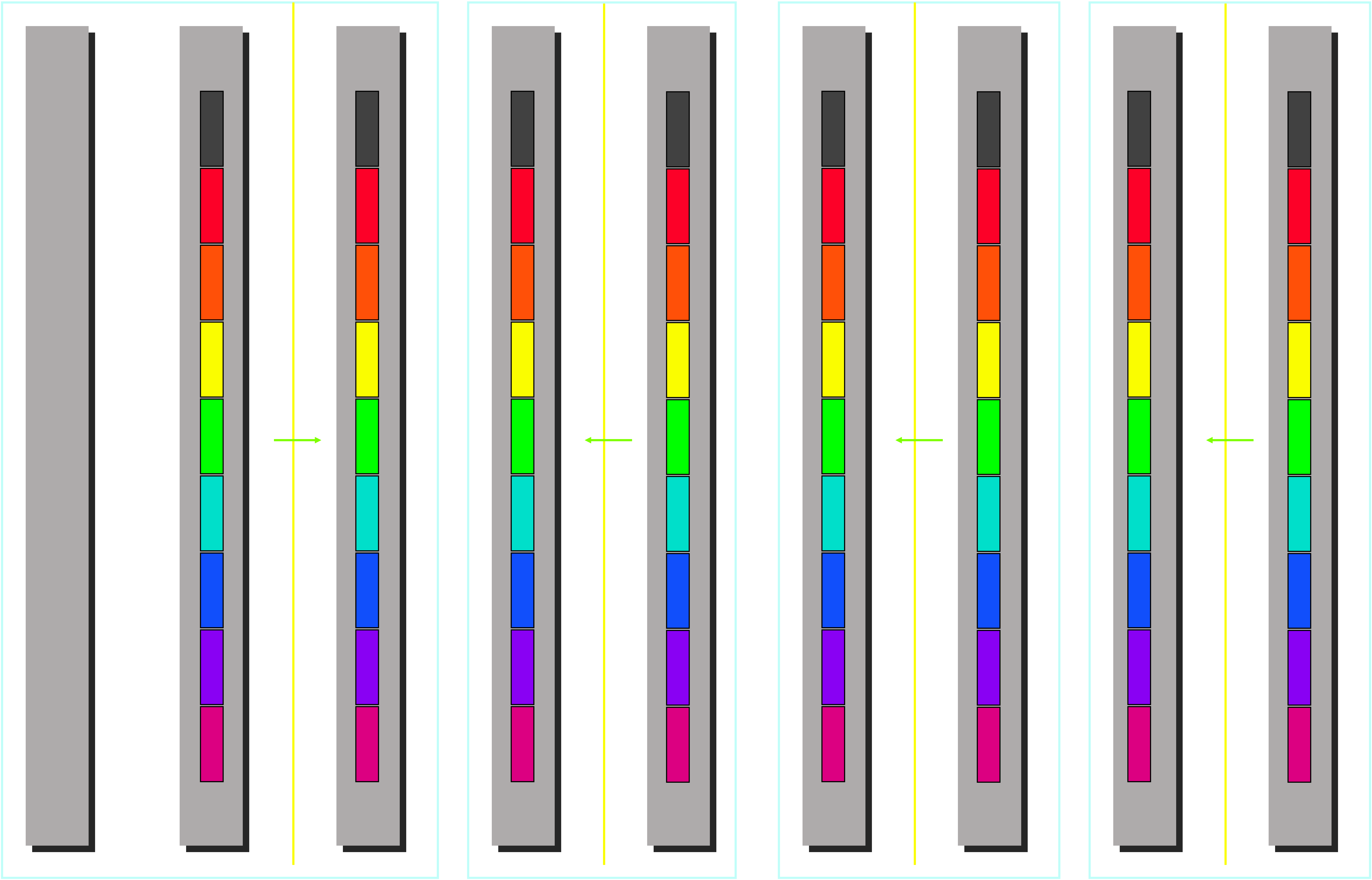


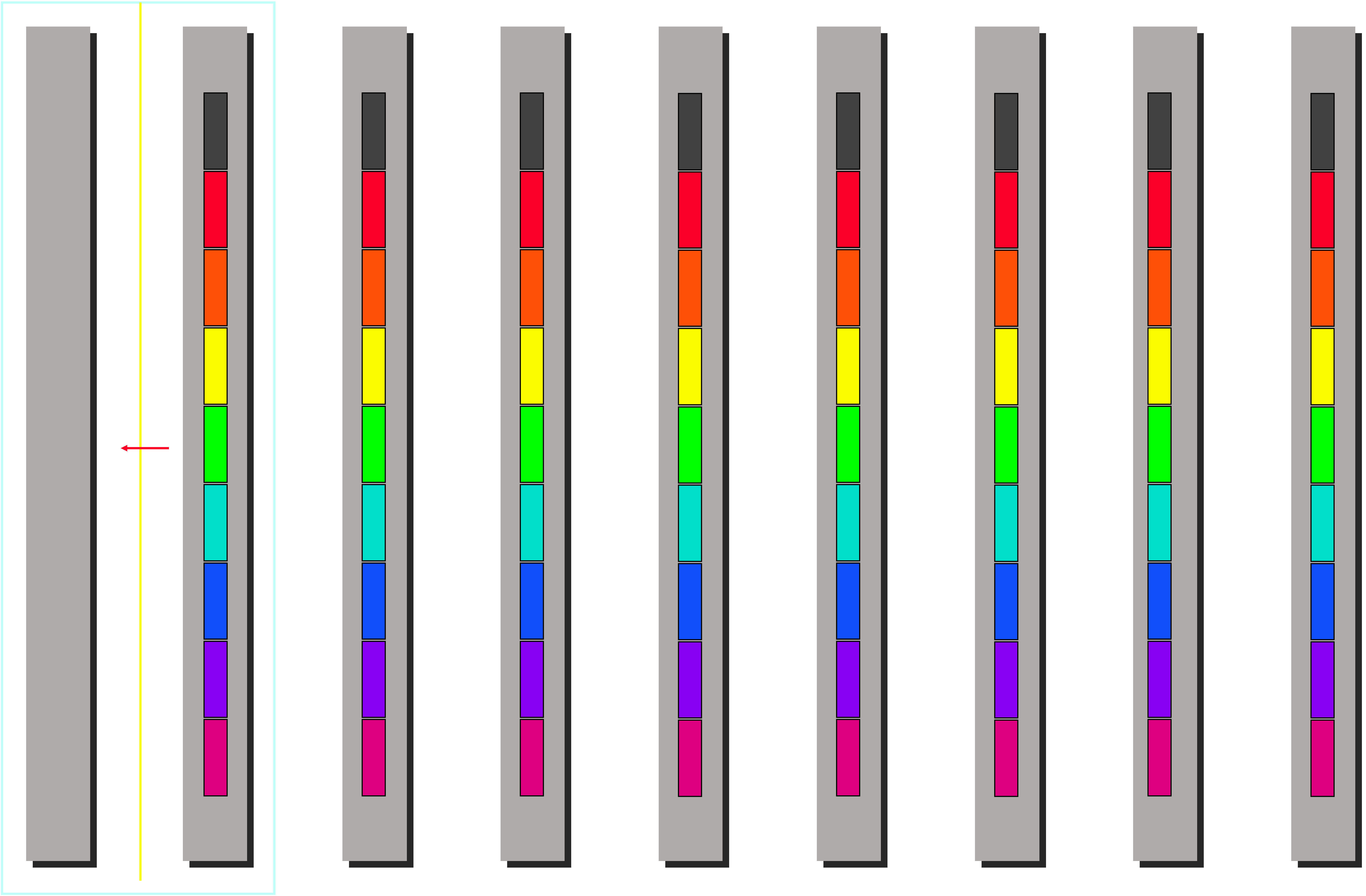


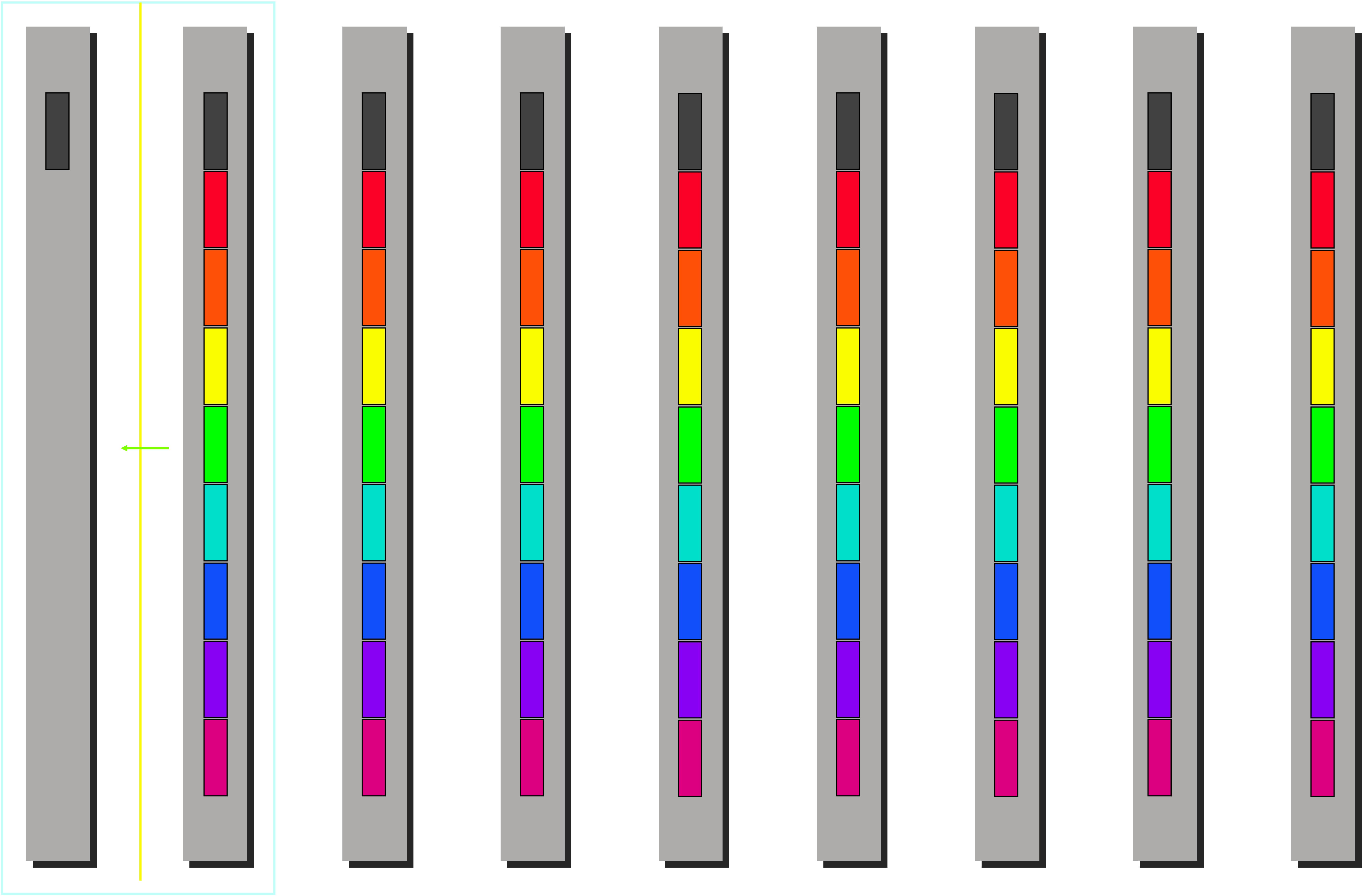


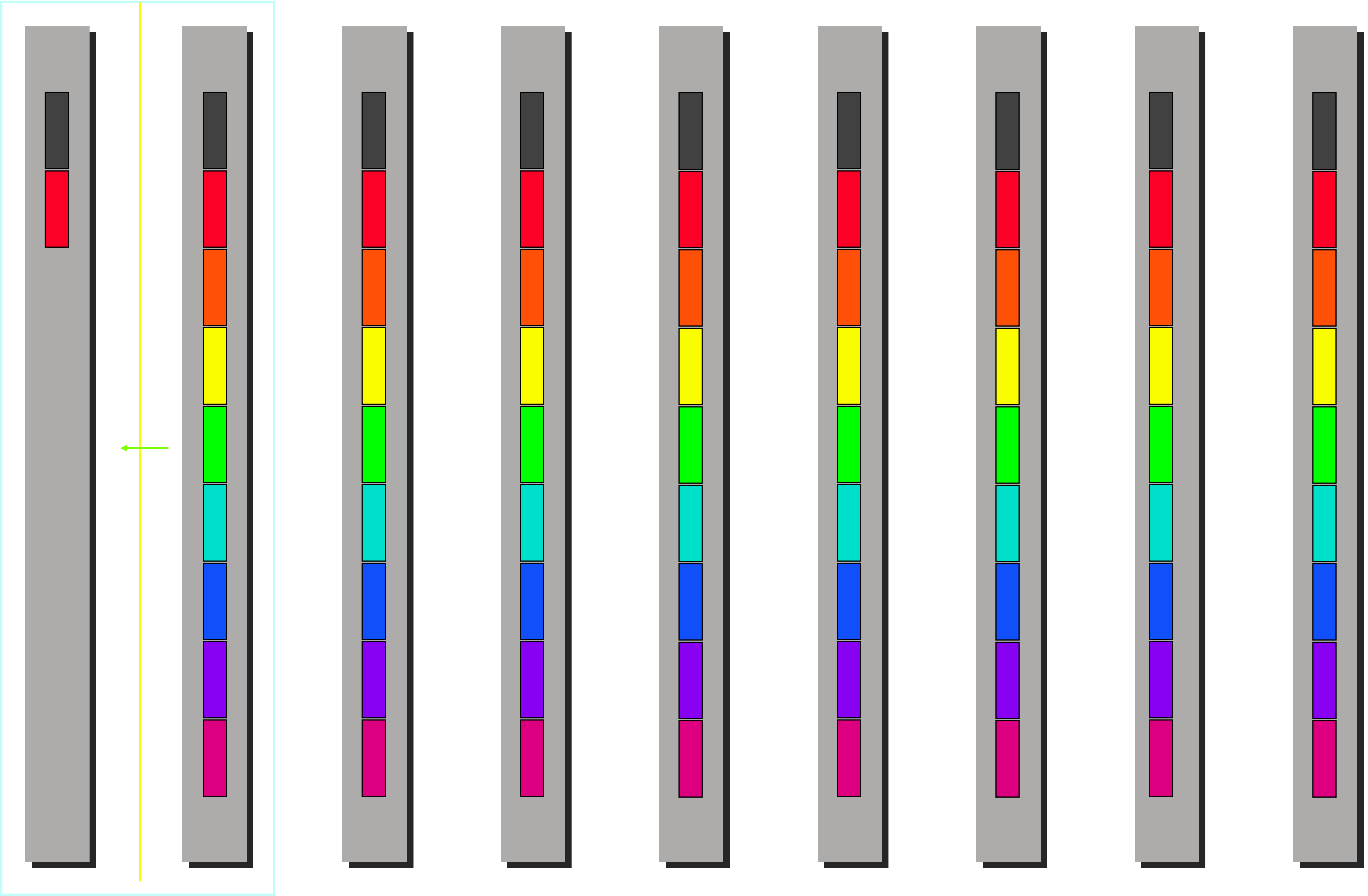


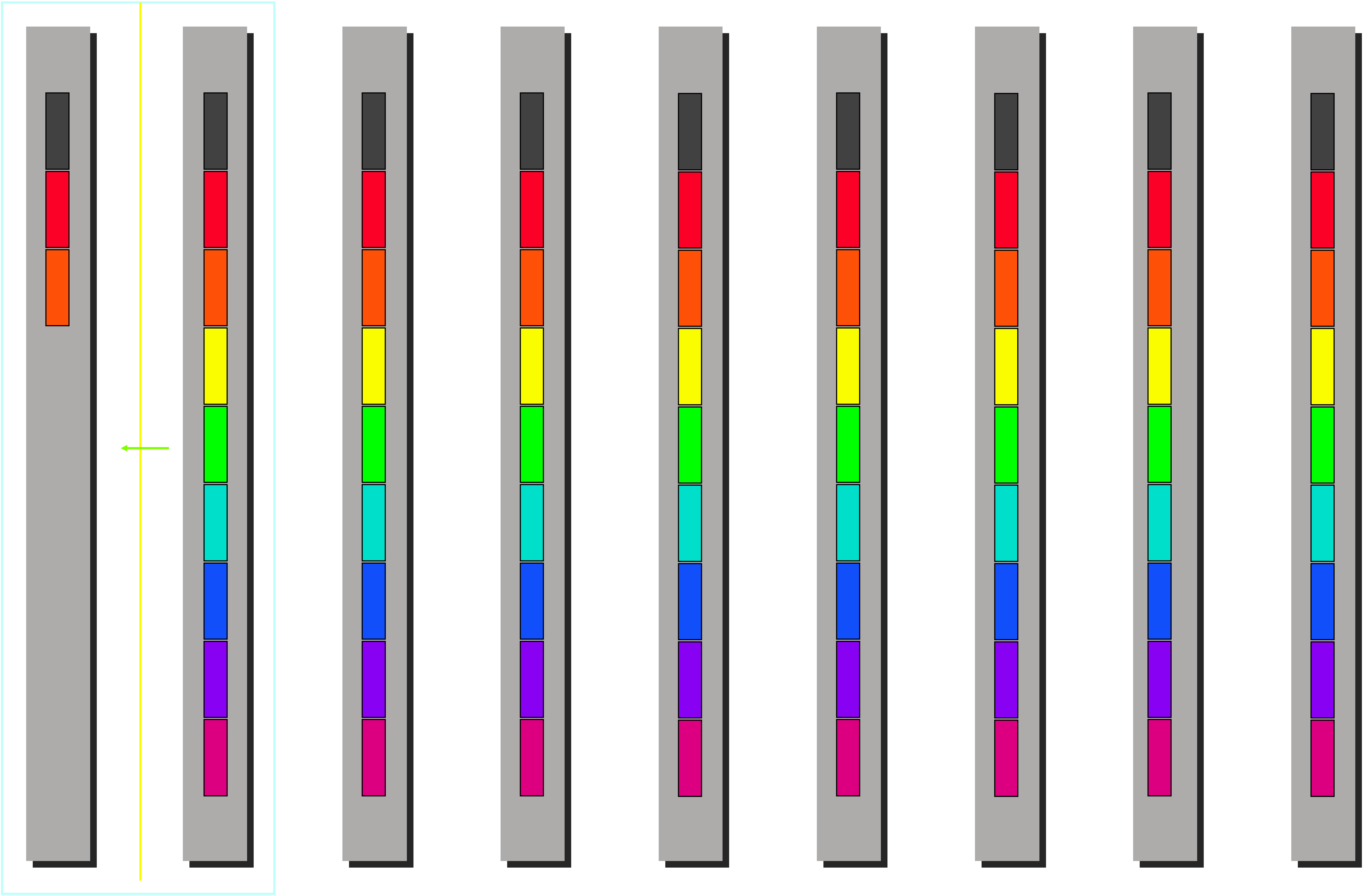


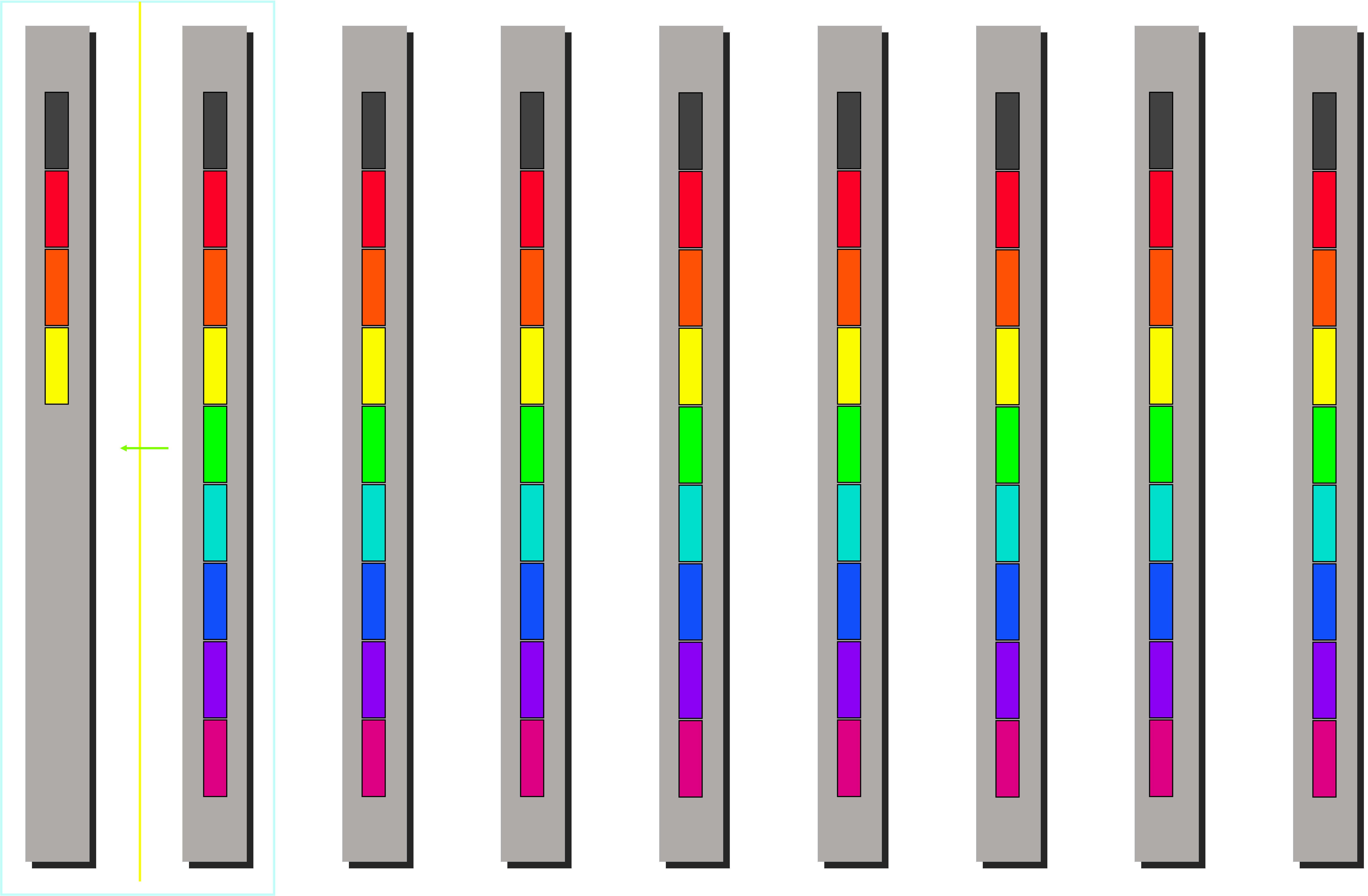


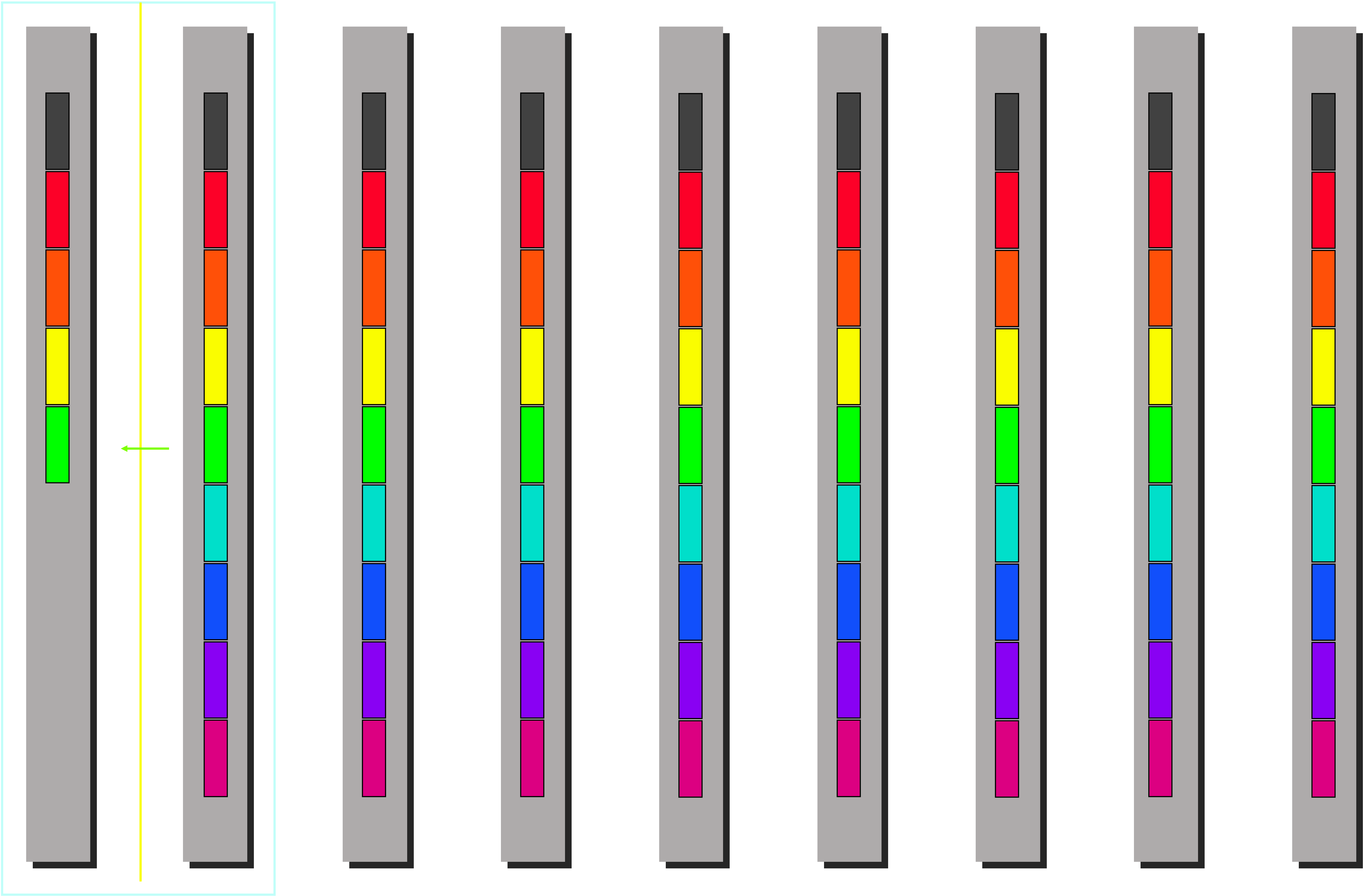


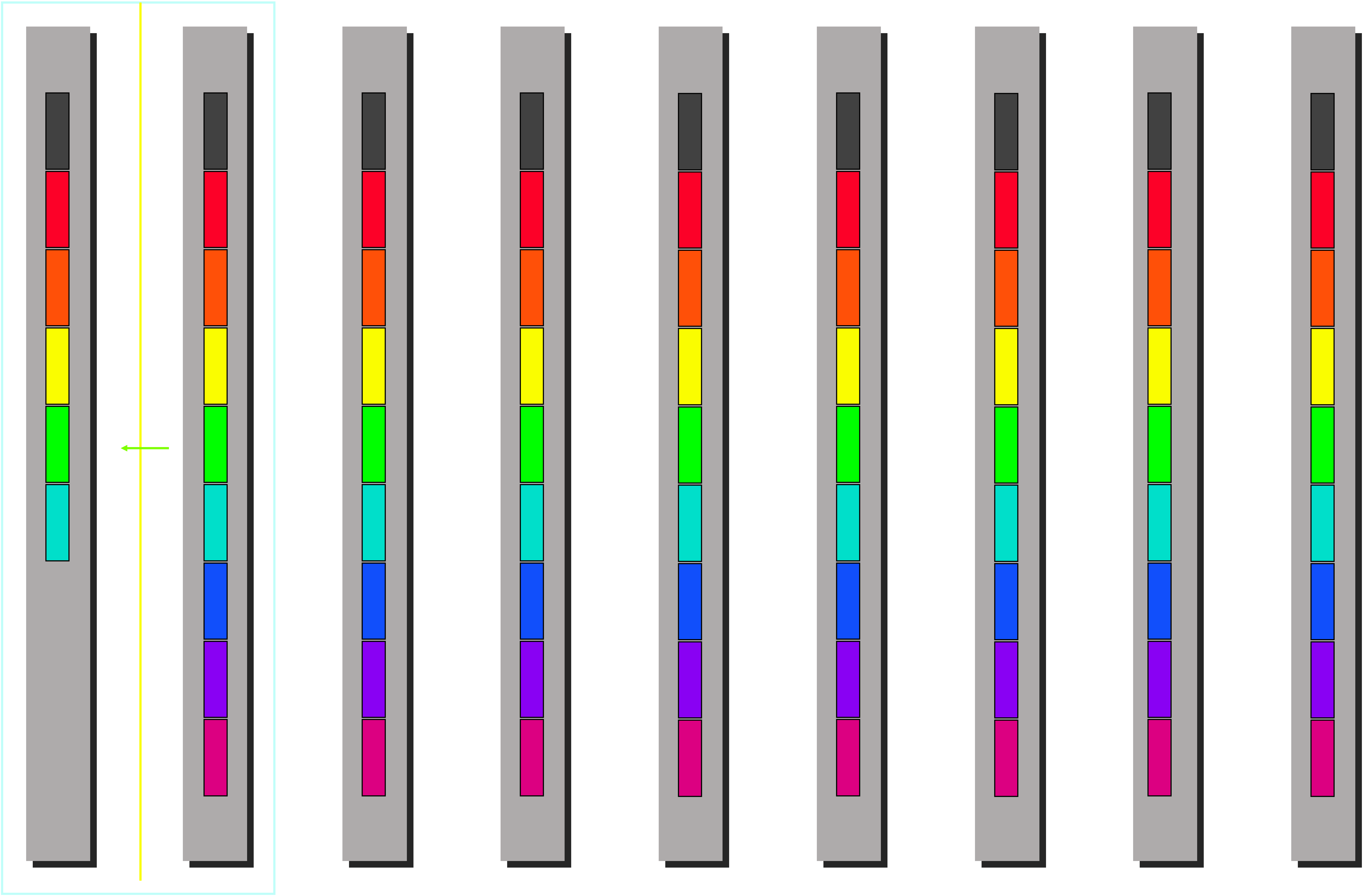


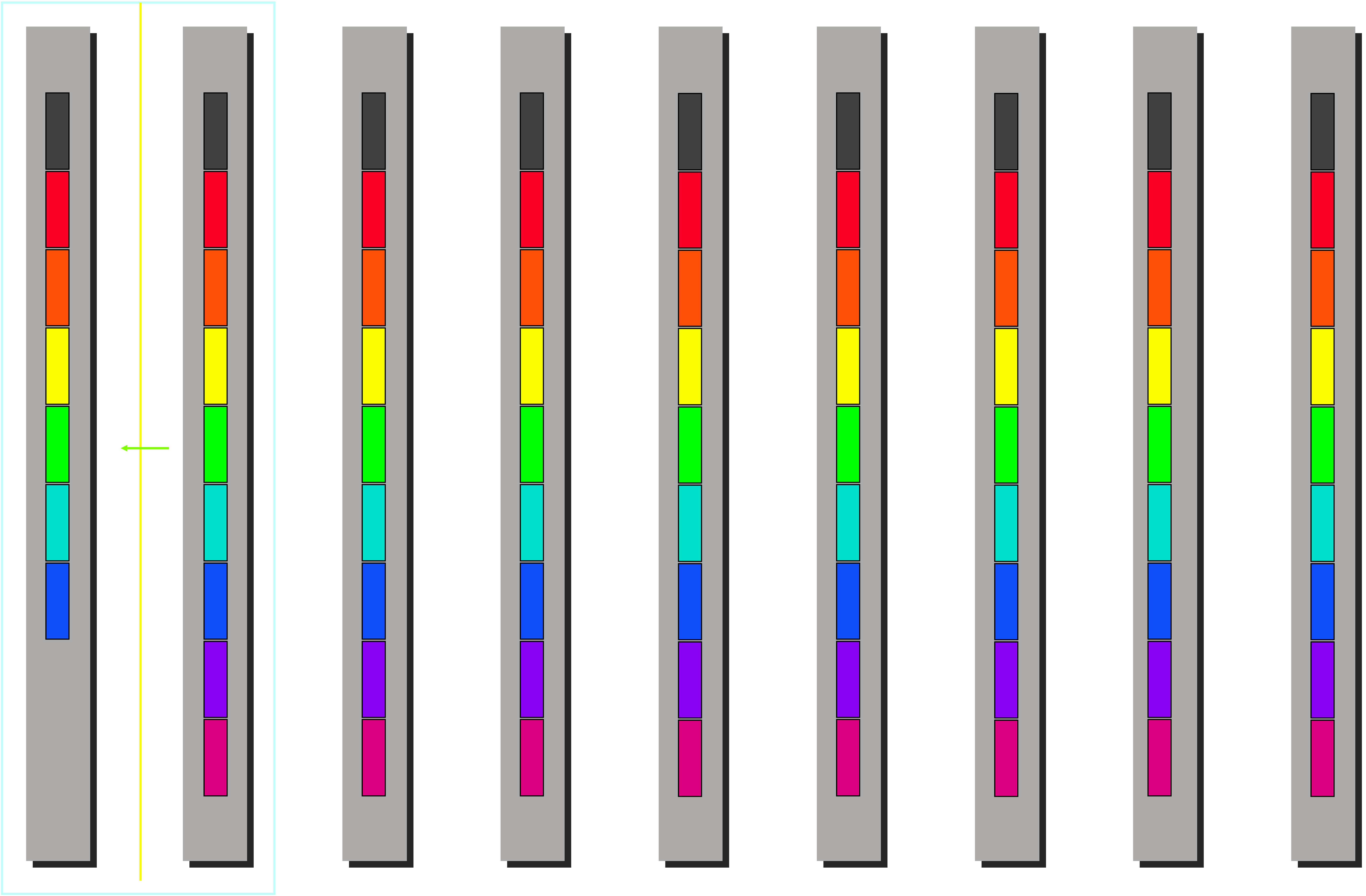


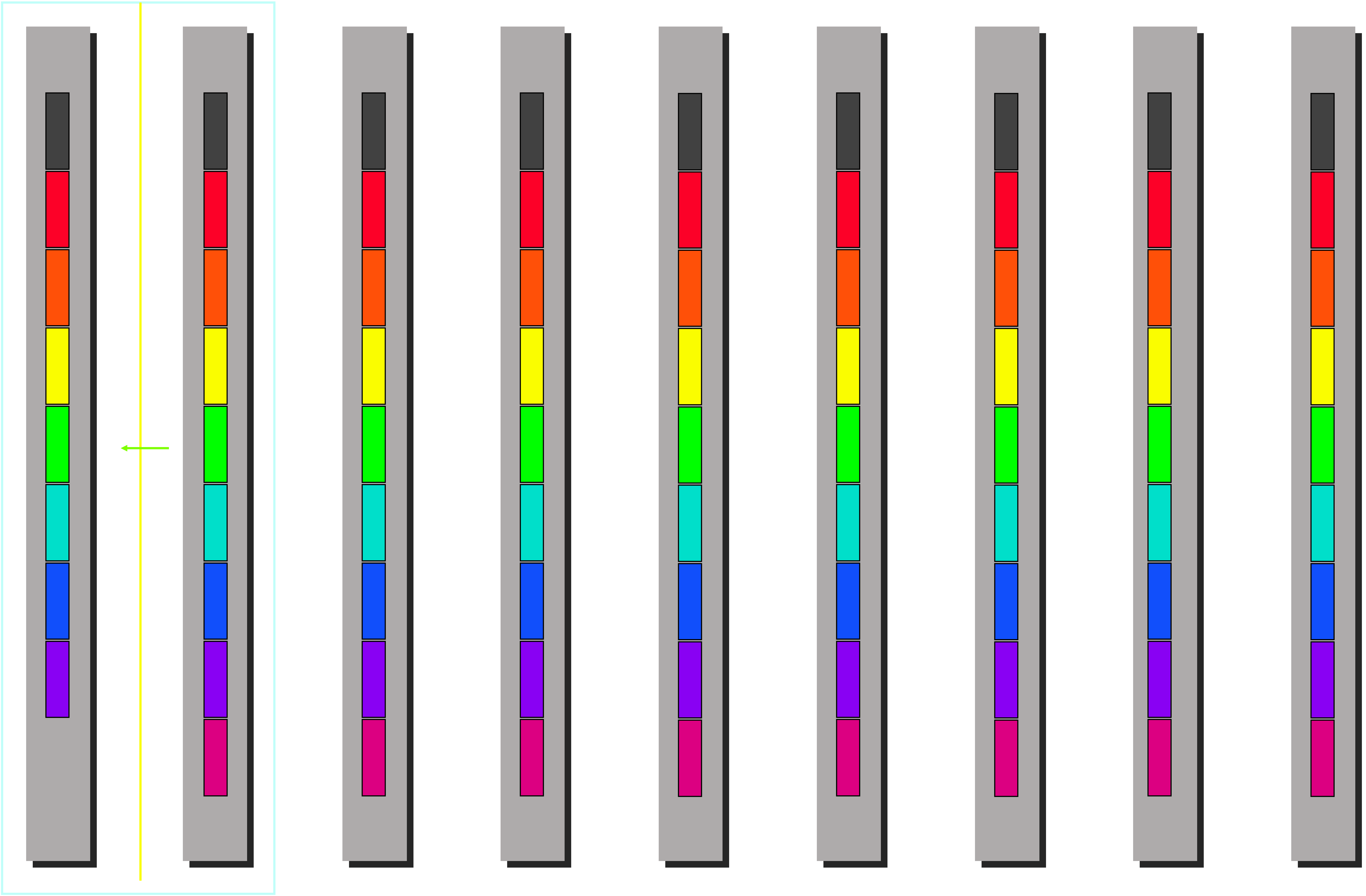


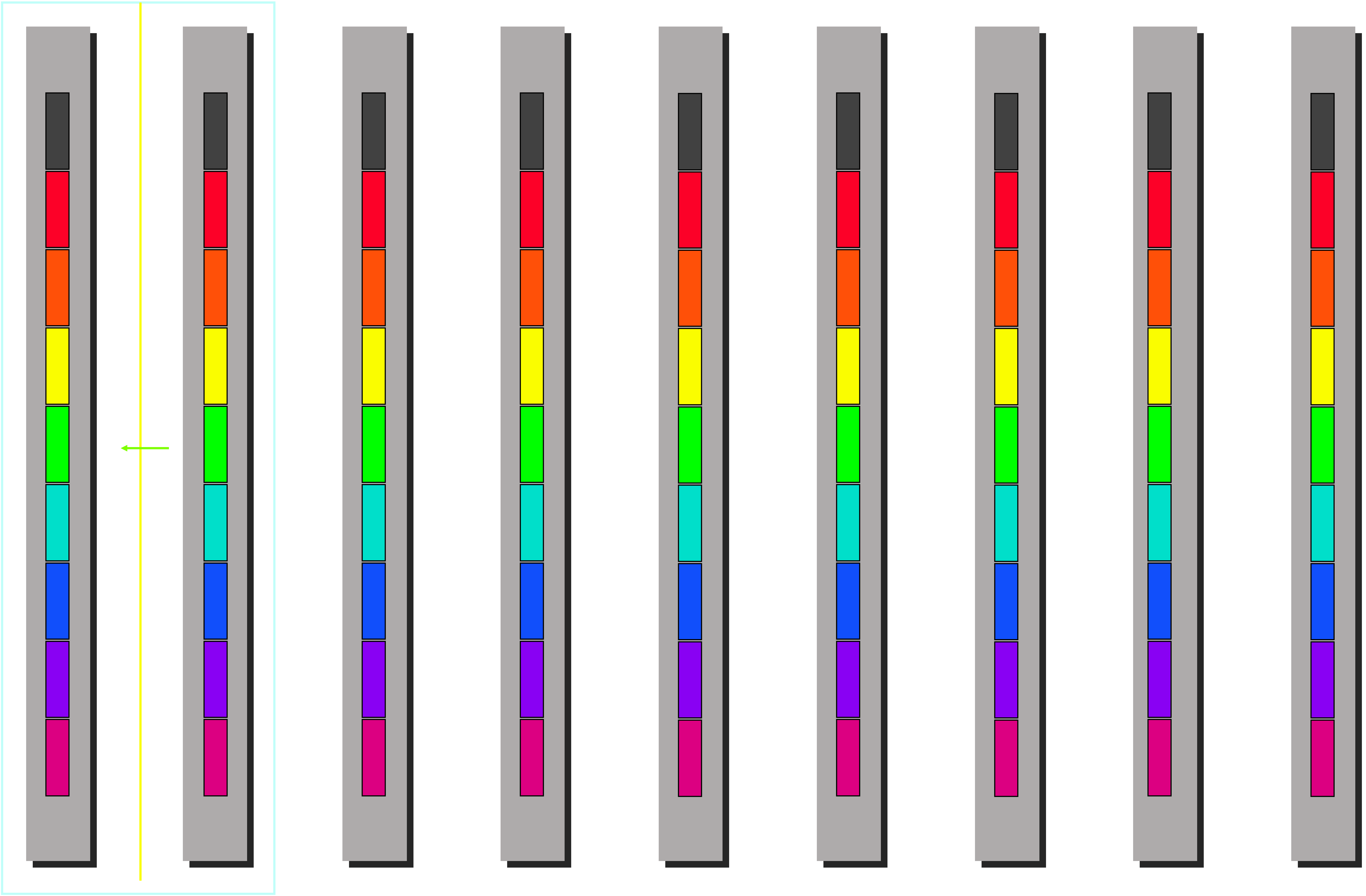


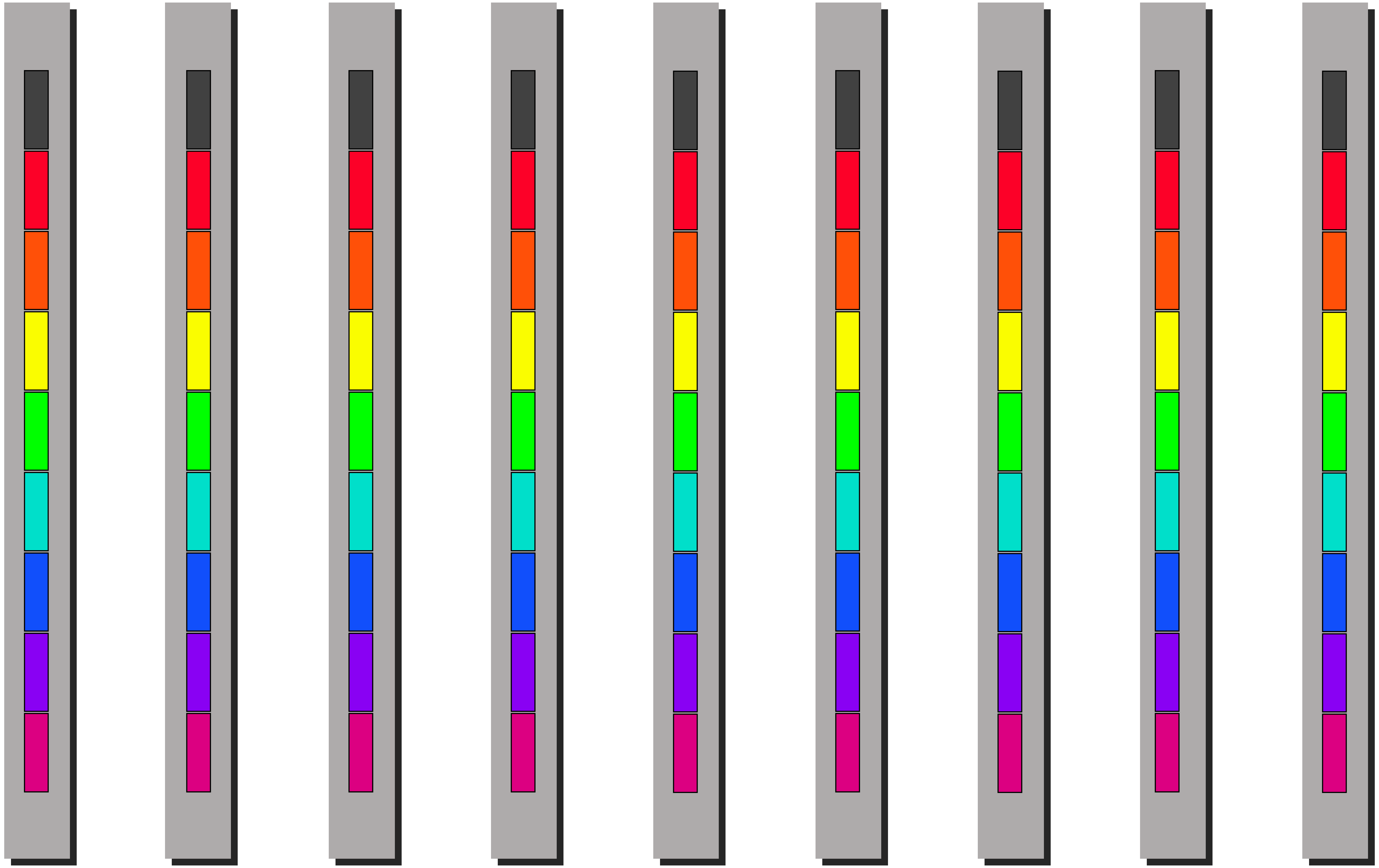












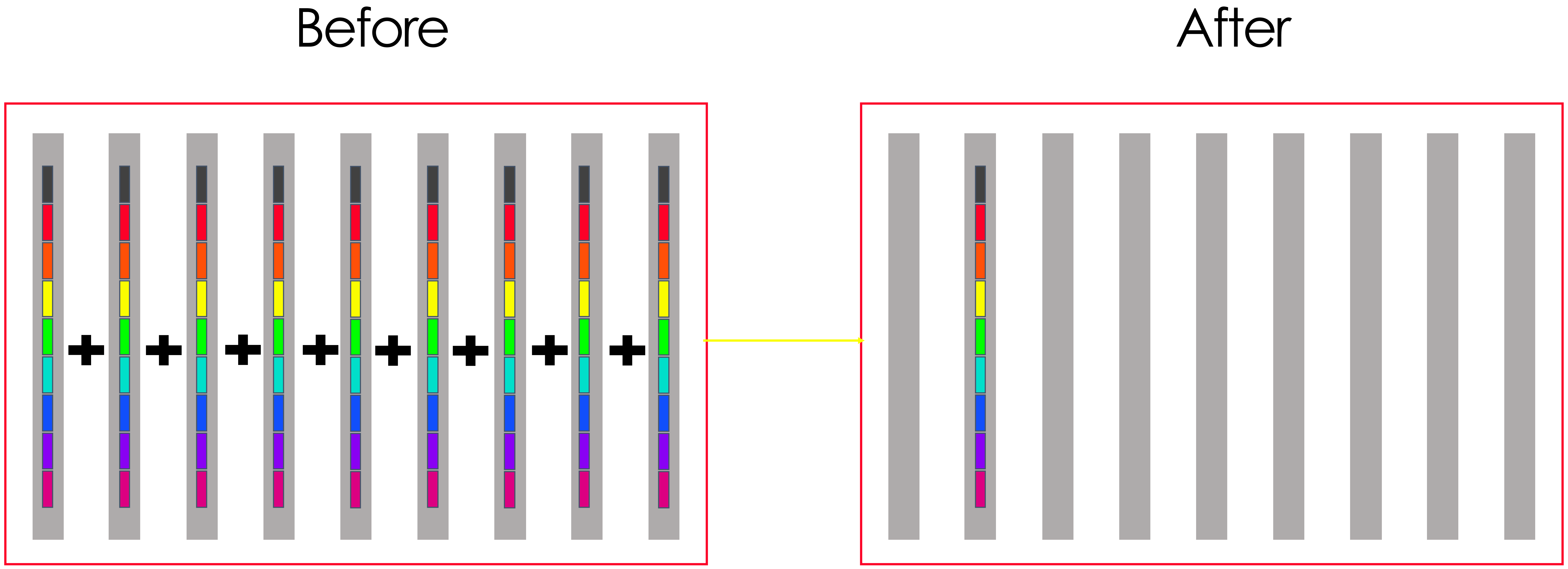
Cost of minimum spanning tree broadcast

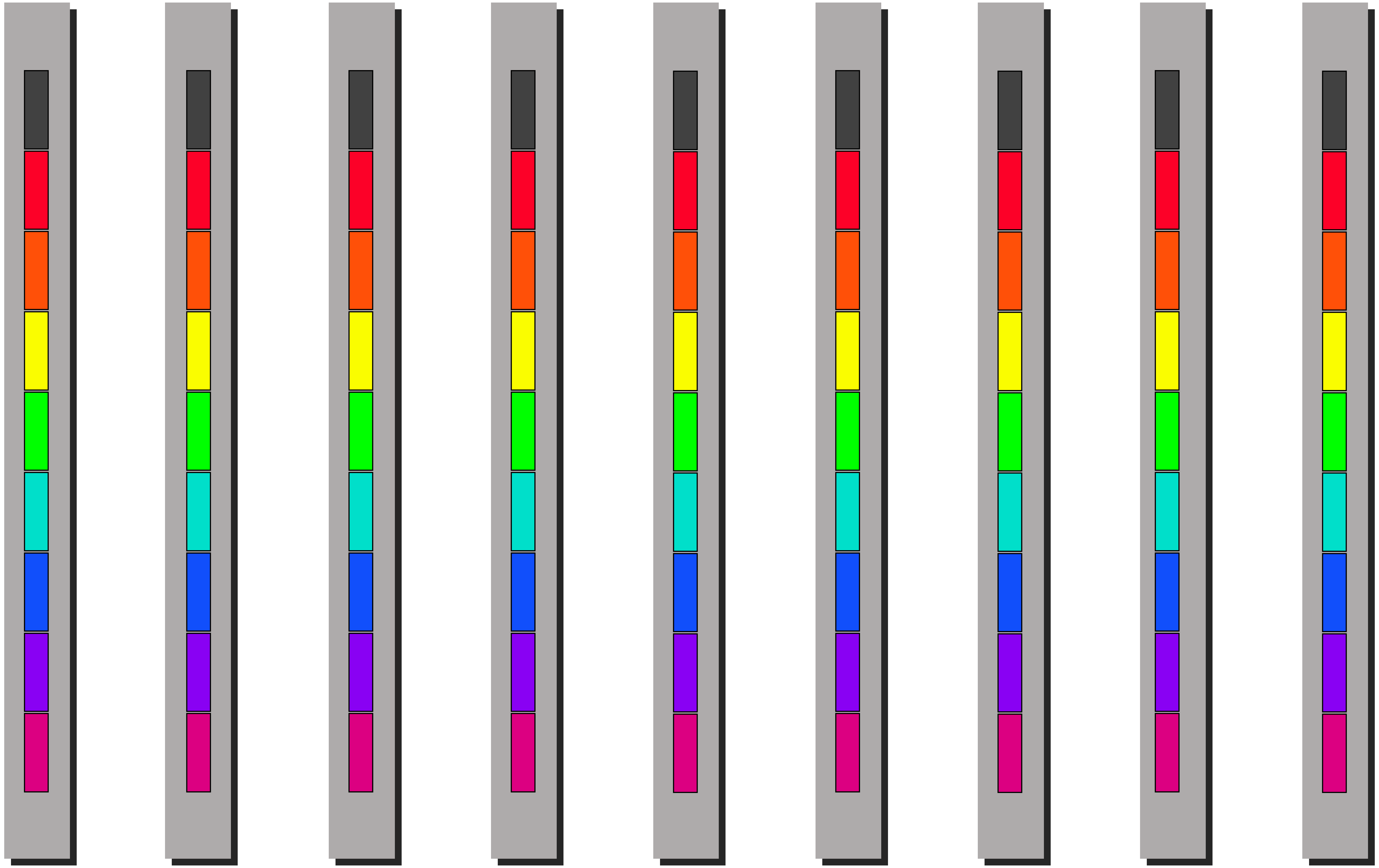
$$\lceil log(p) \rceil \, (\alpha + n\beta)$$

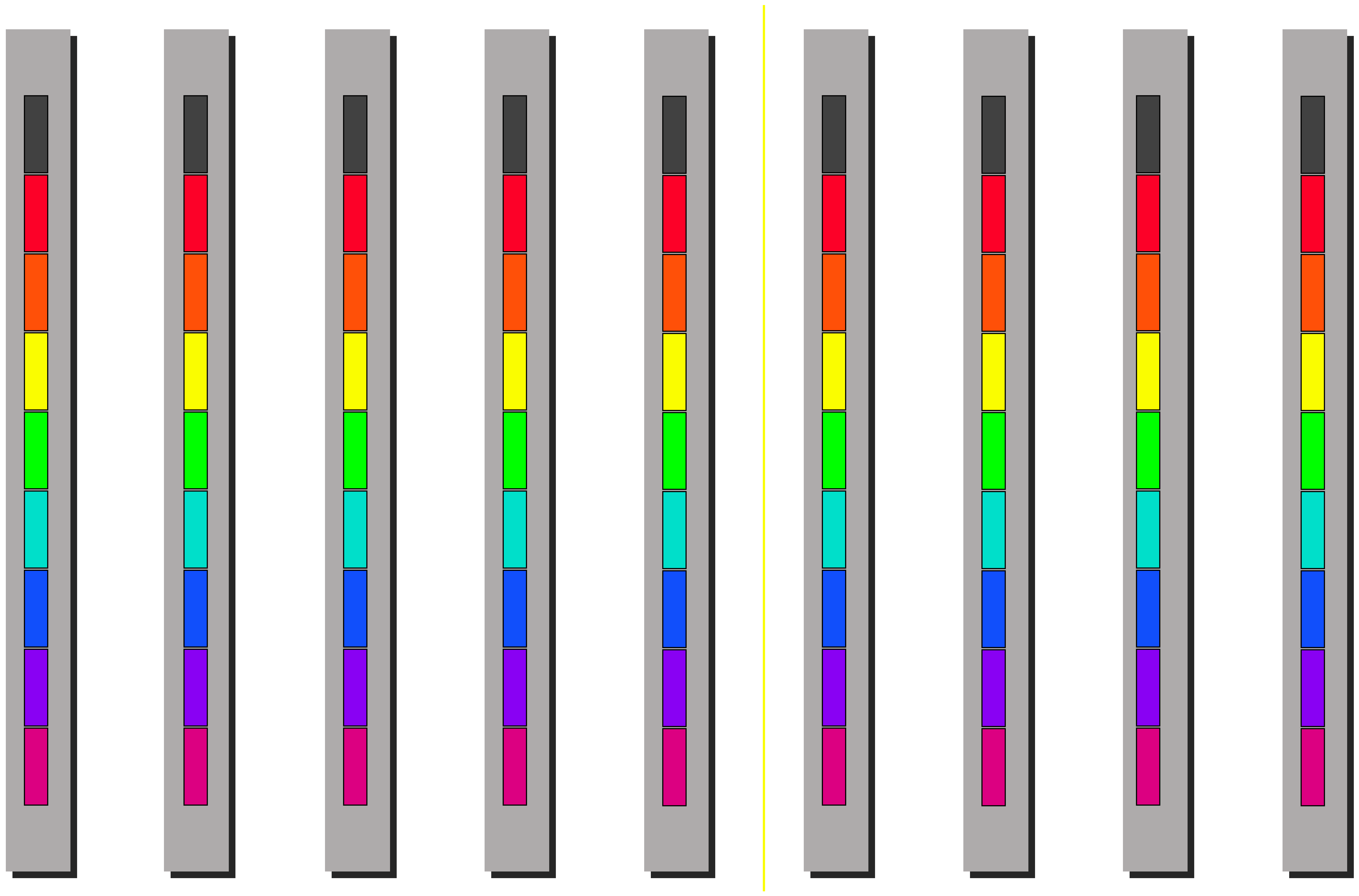
number of steps

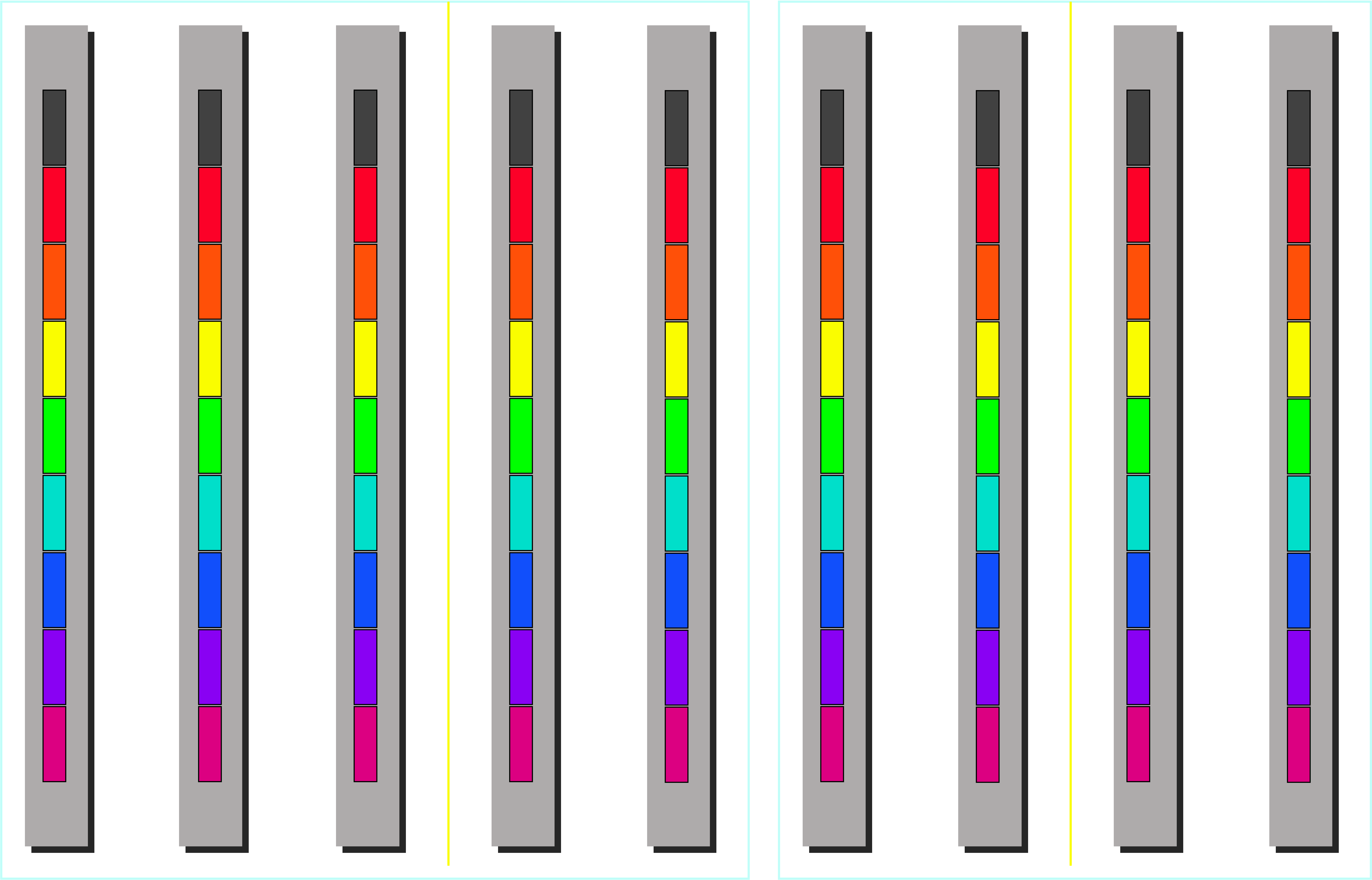
cost per steps

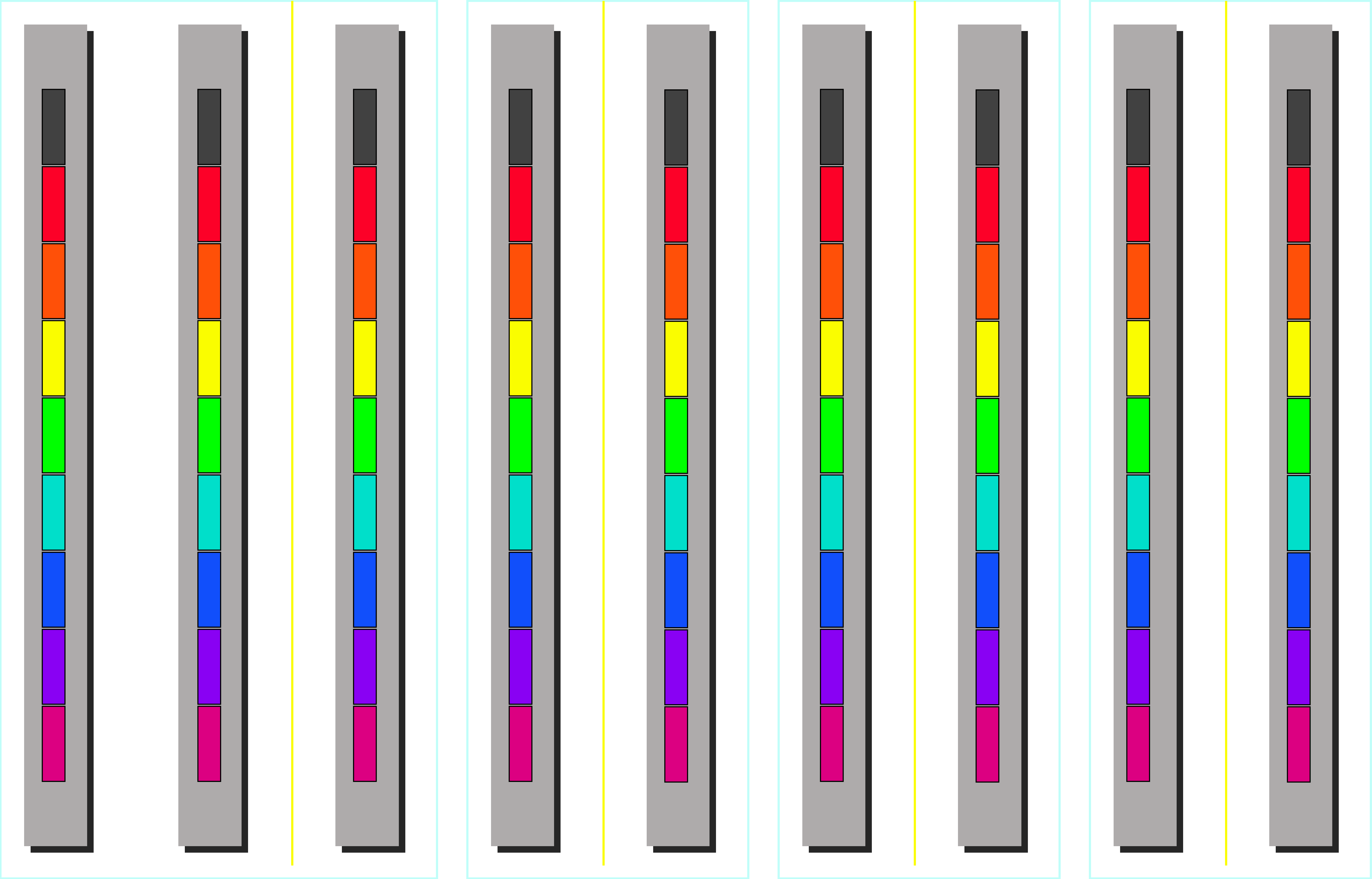
Reduce(-to-one)

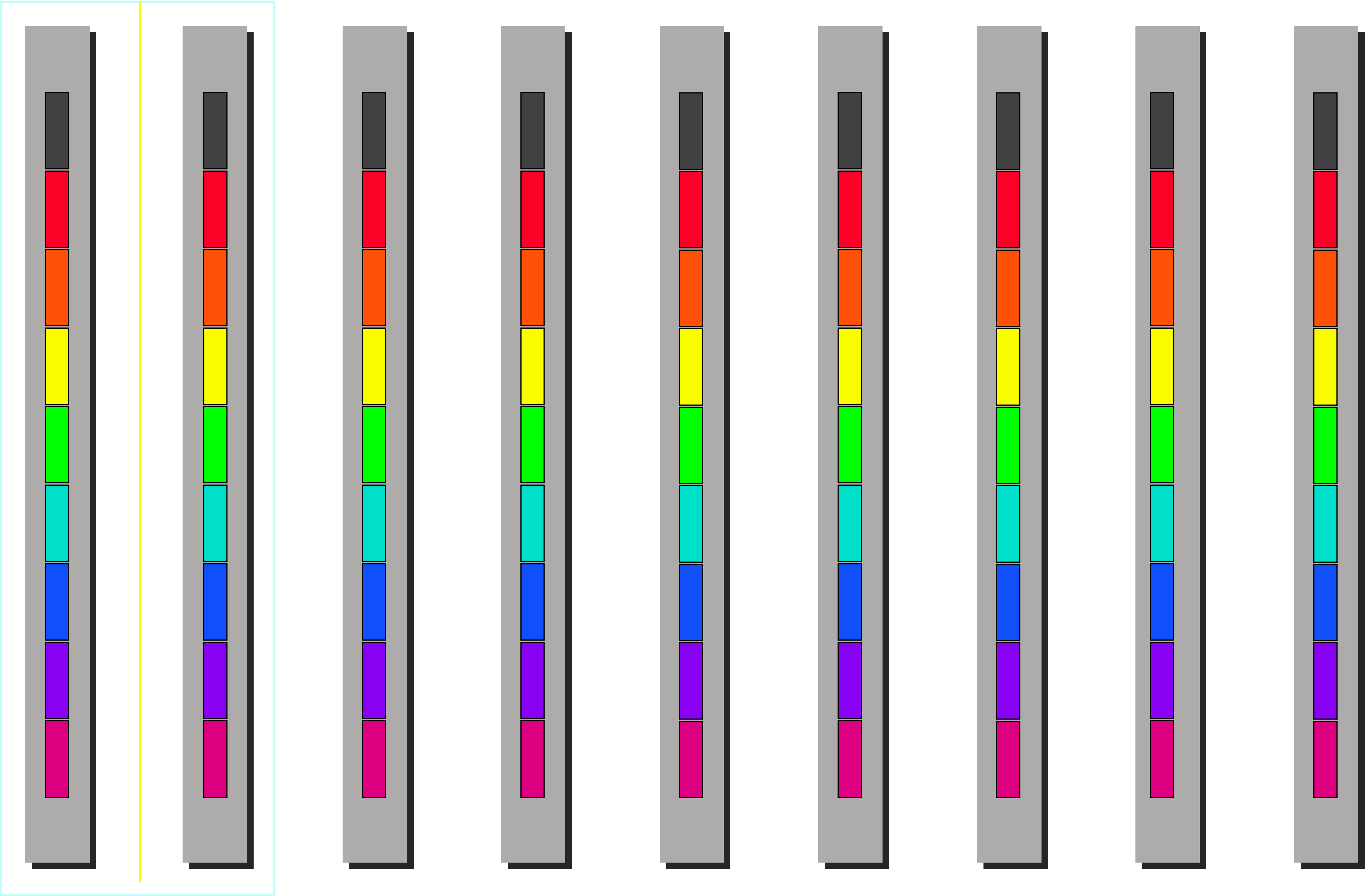


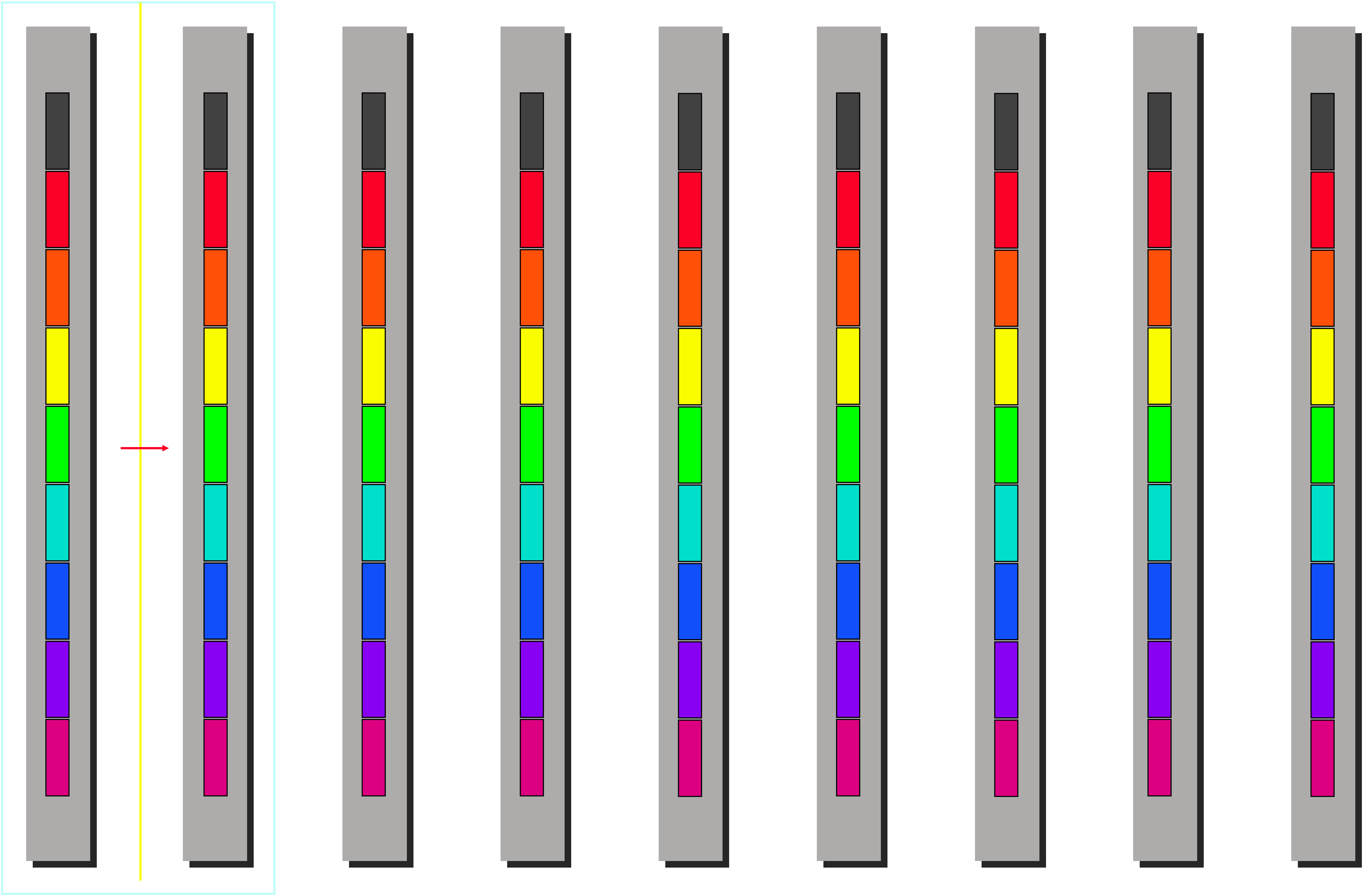


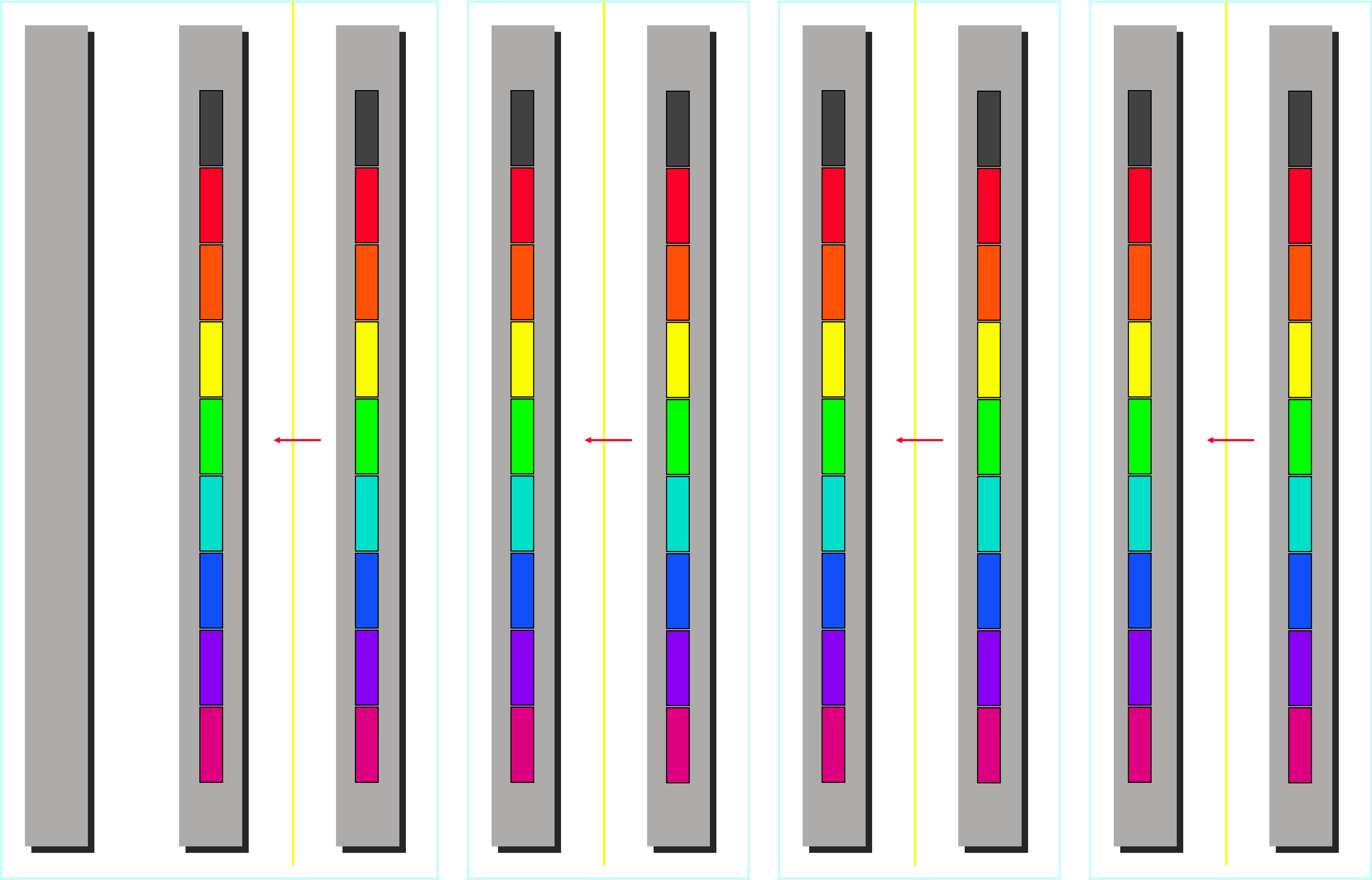


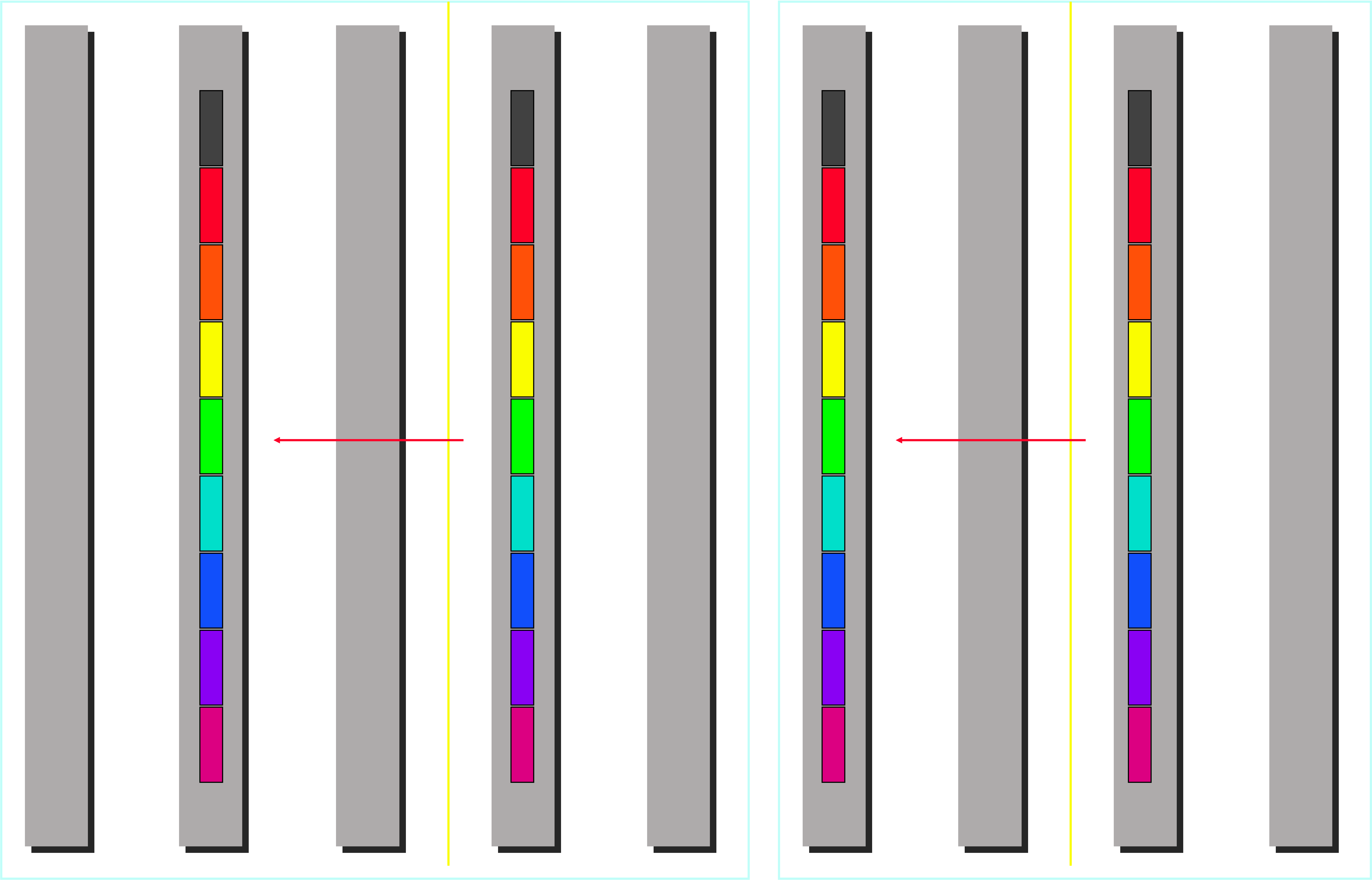


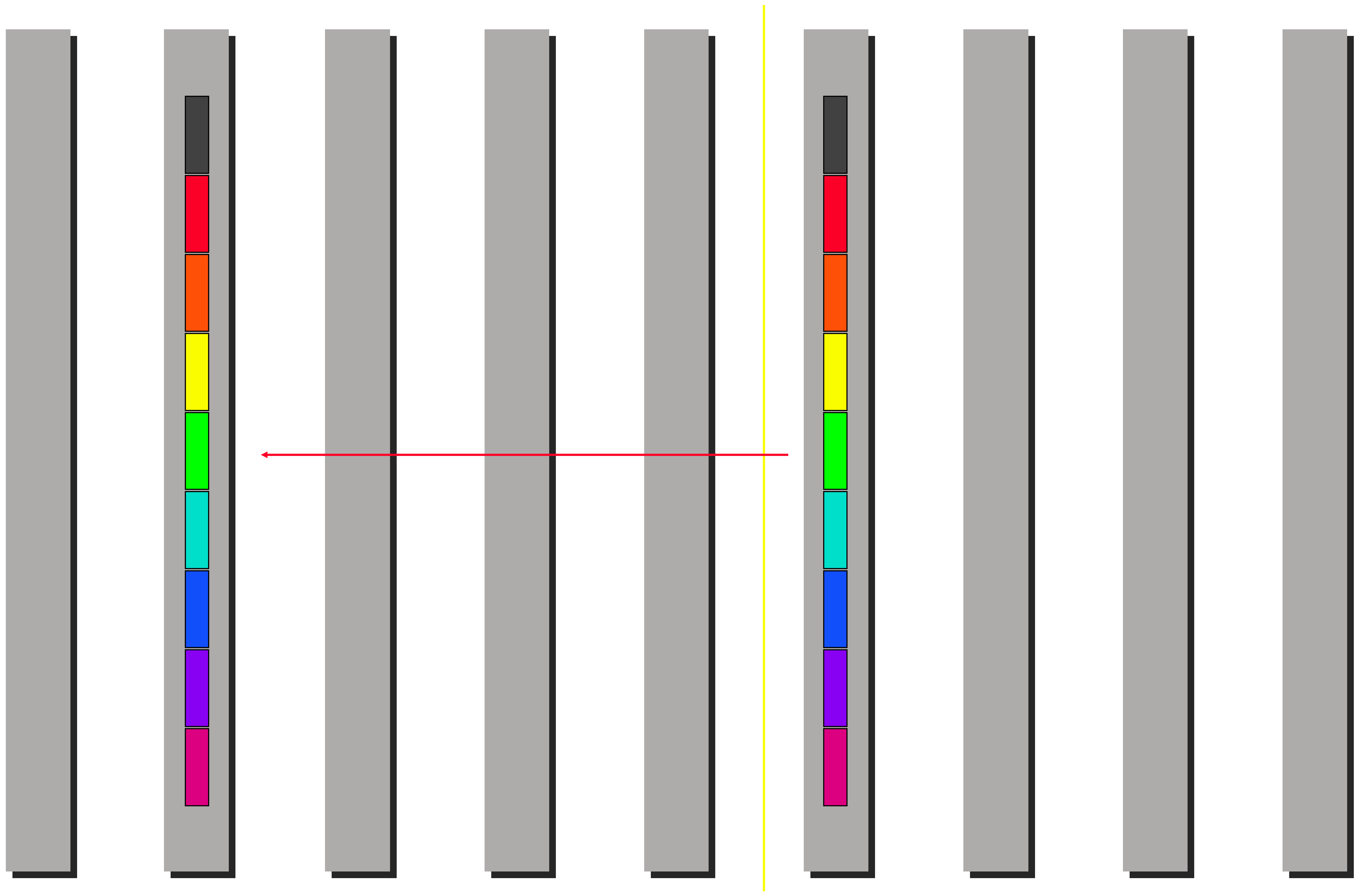








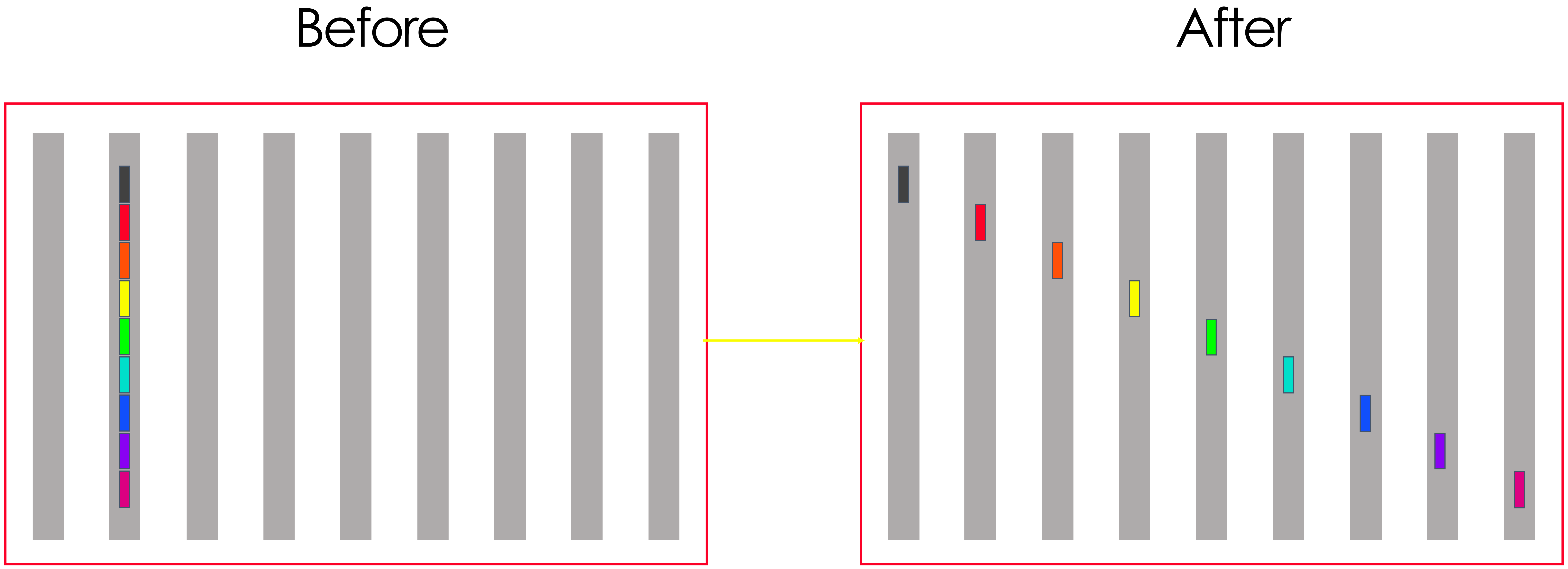


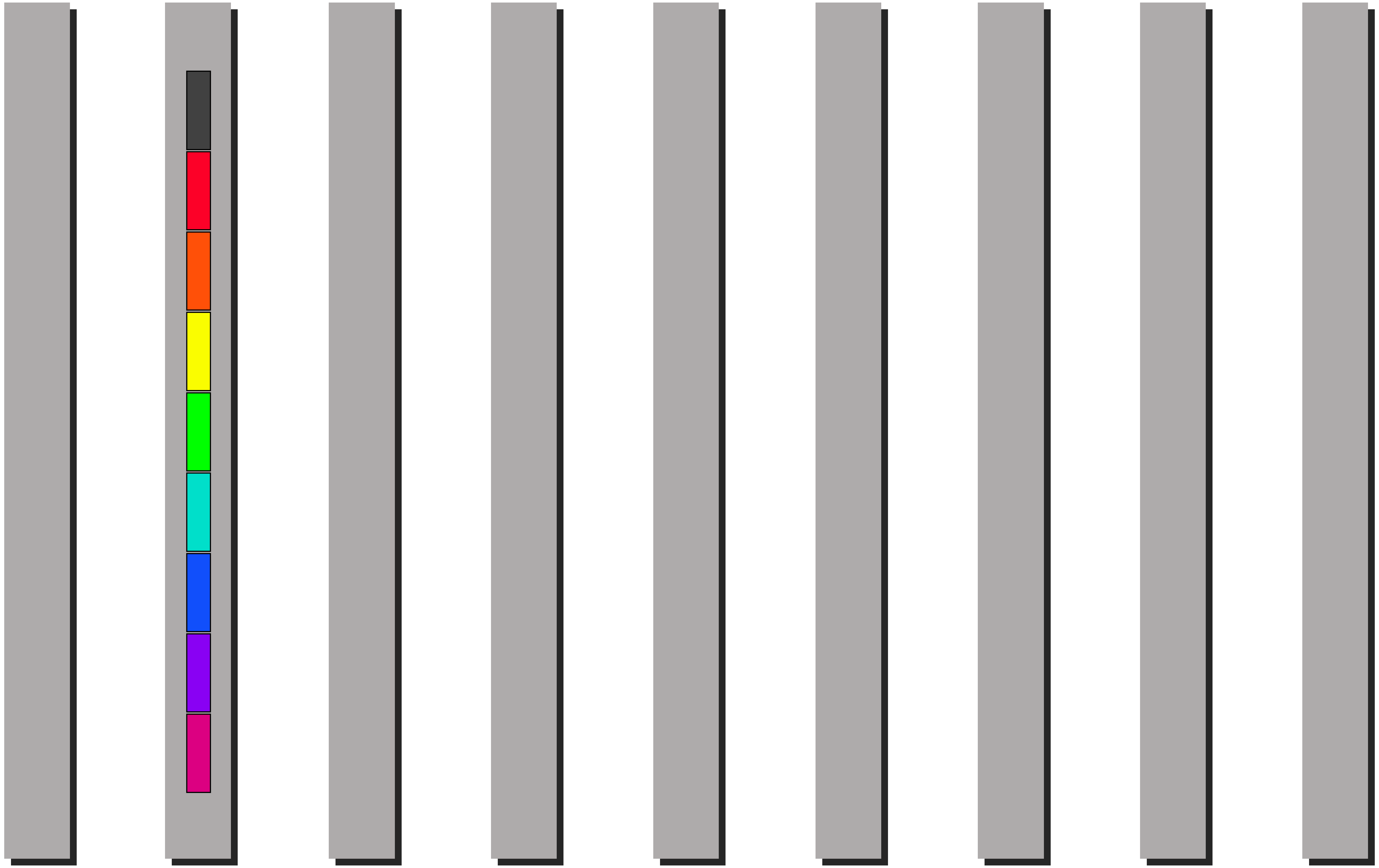


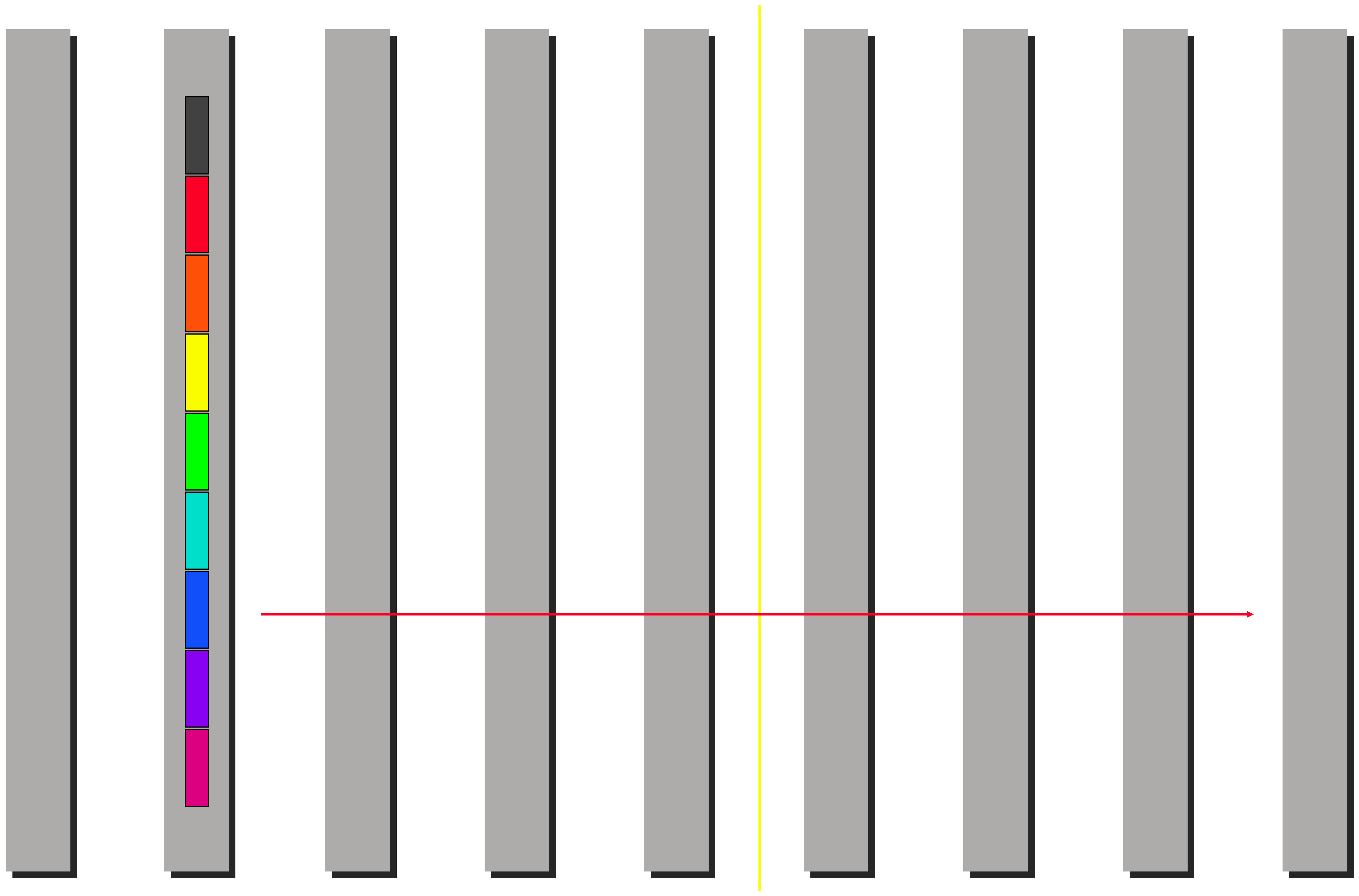
Cost of minimum spanning tree reduce(-to-one)

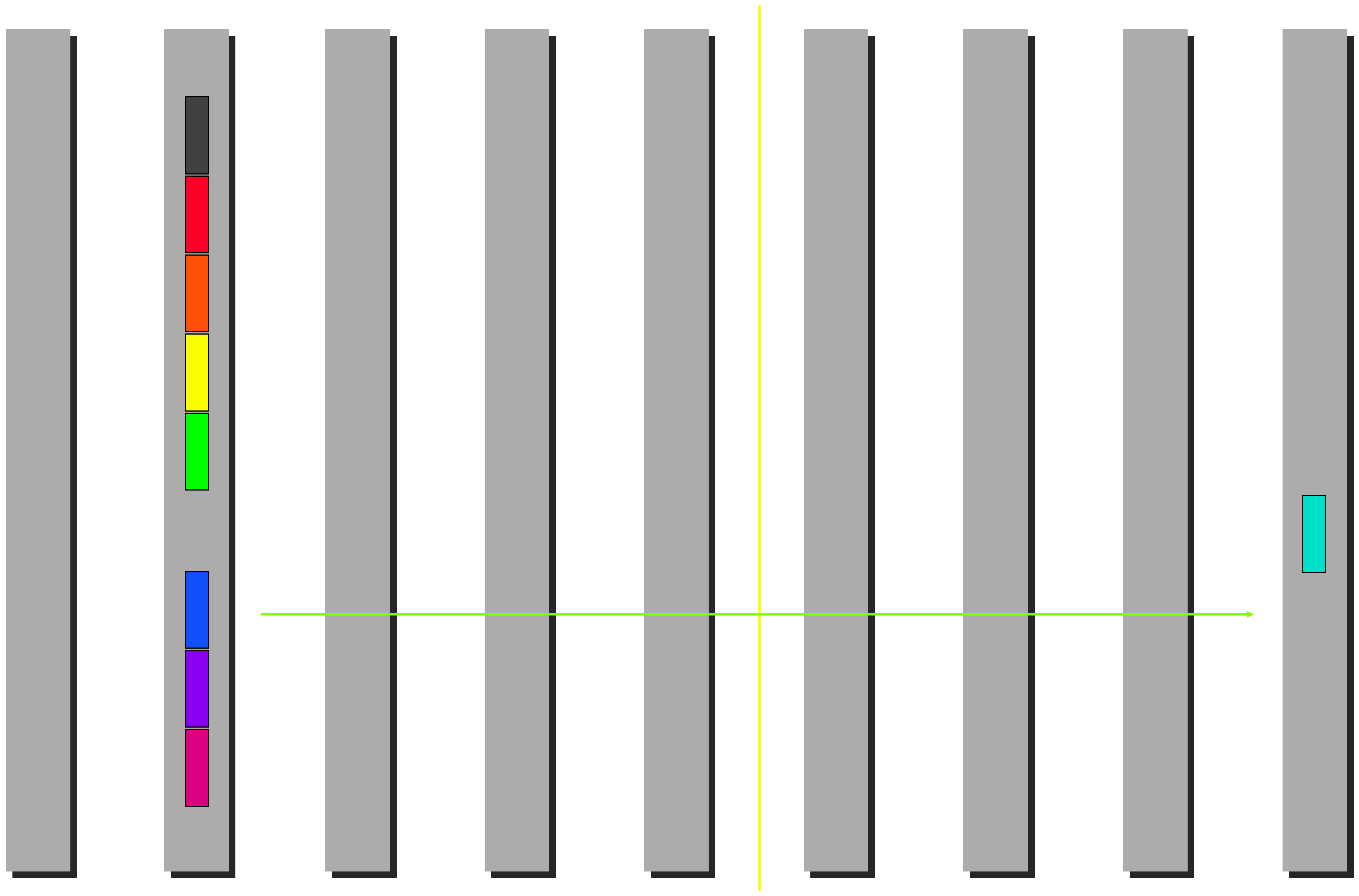
$$\lceil log(p) \rceil \ (\alpha + n\beta + n\gamma)$$

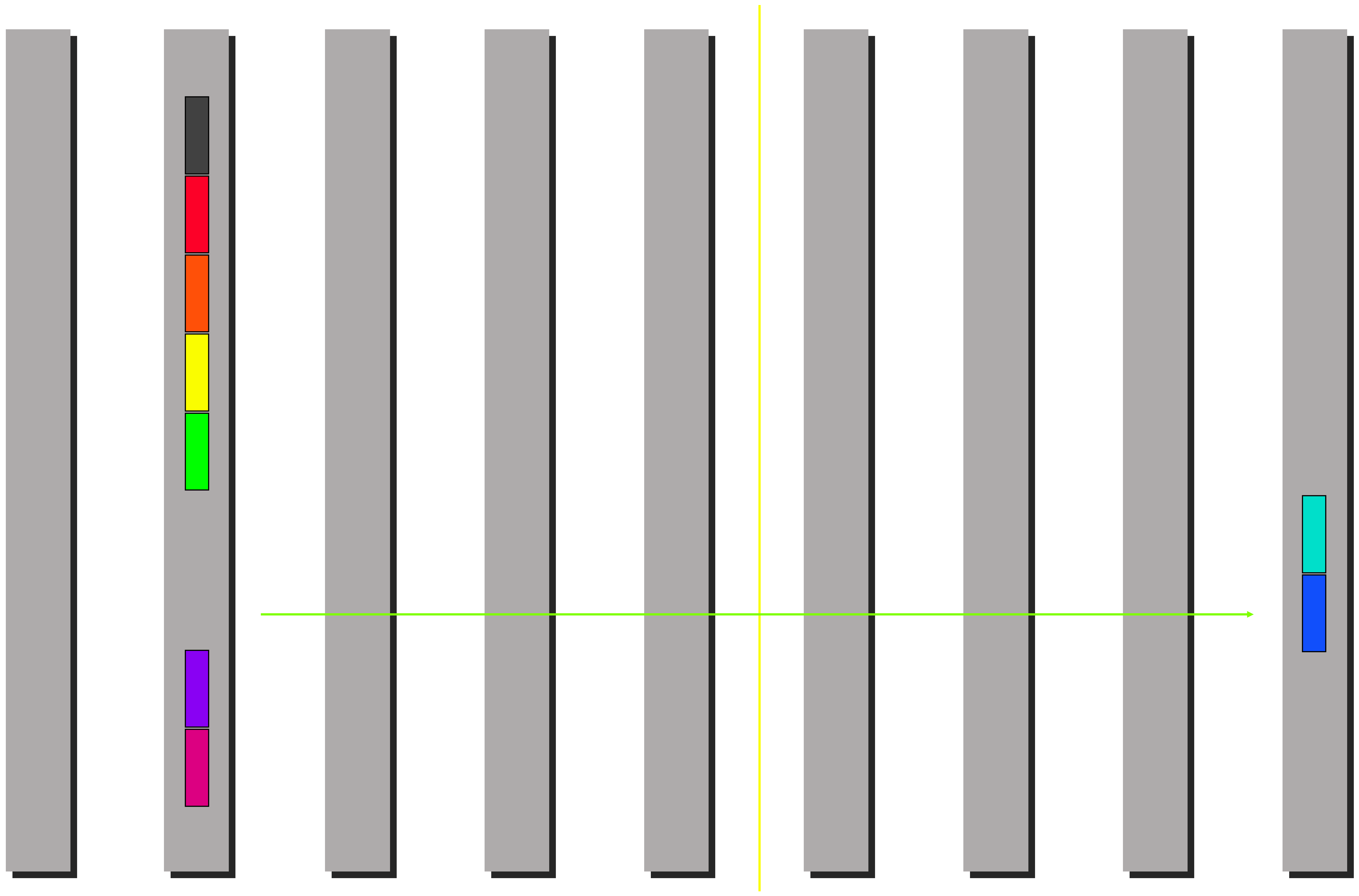
Scatter

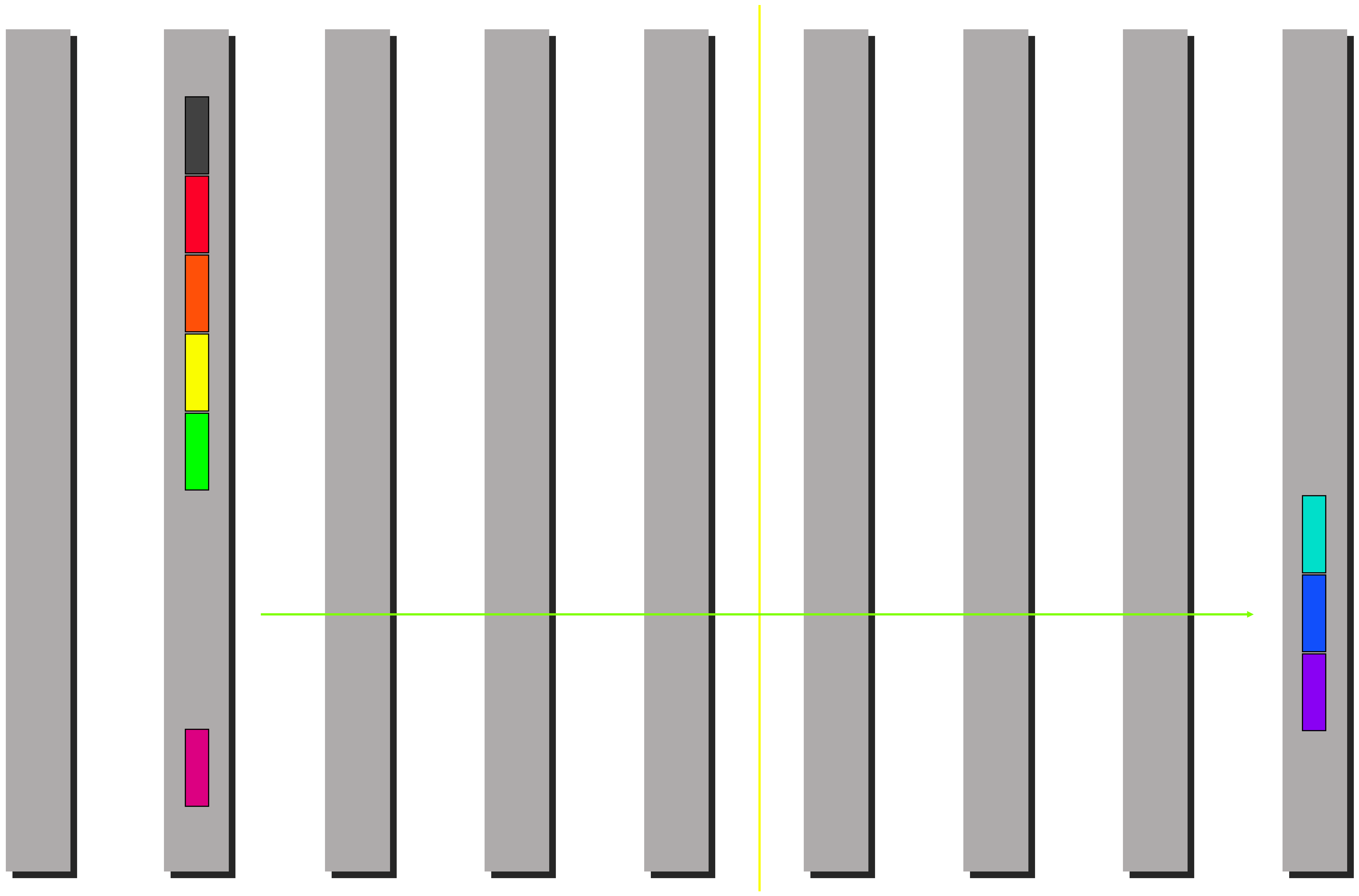


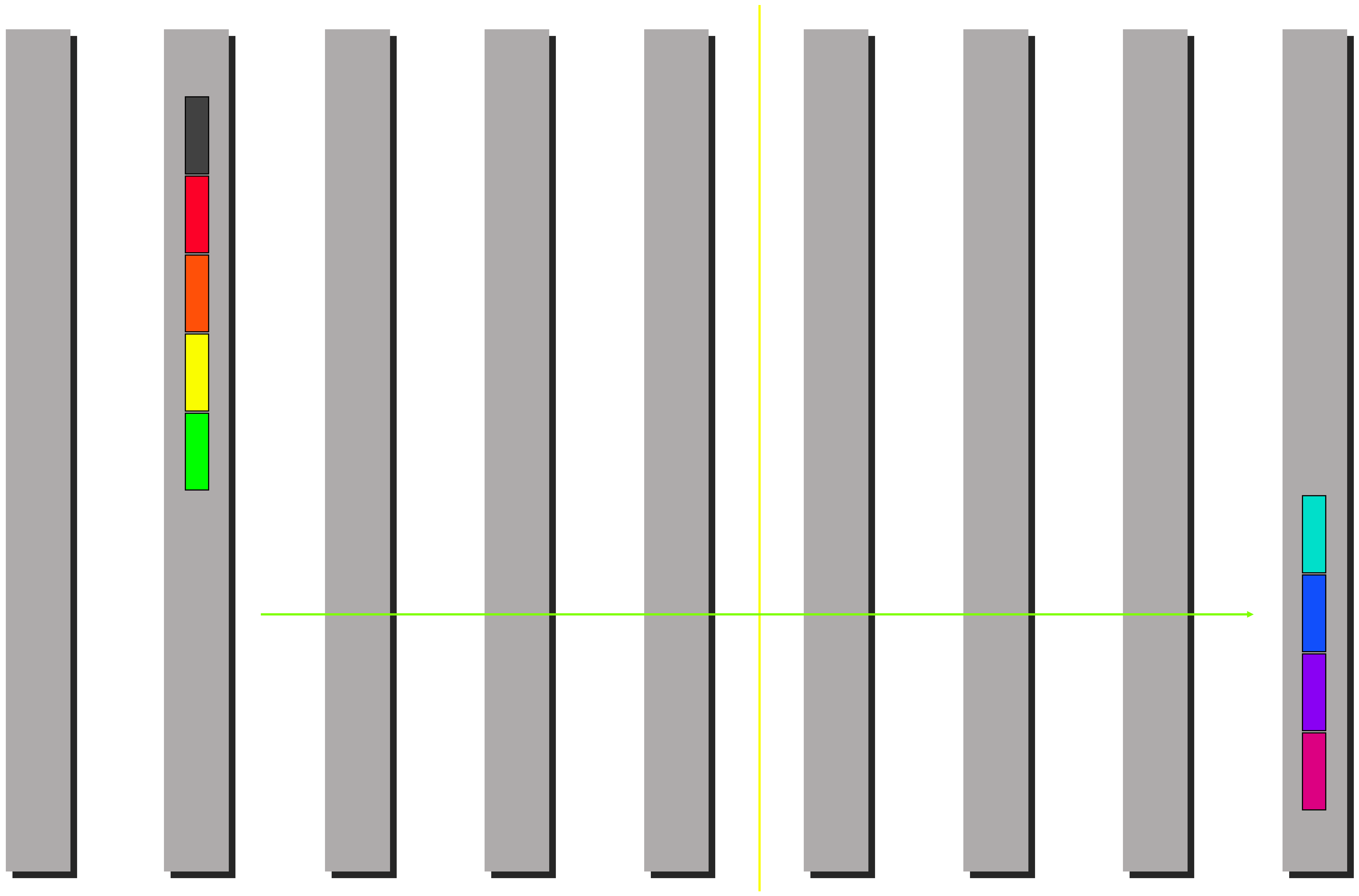


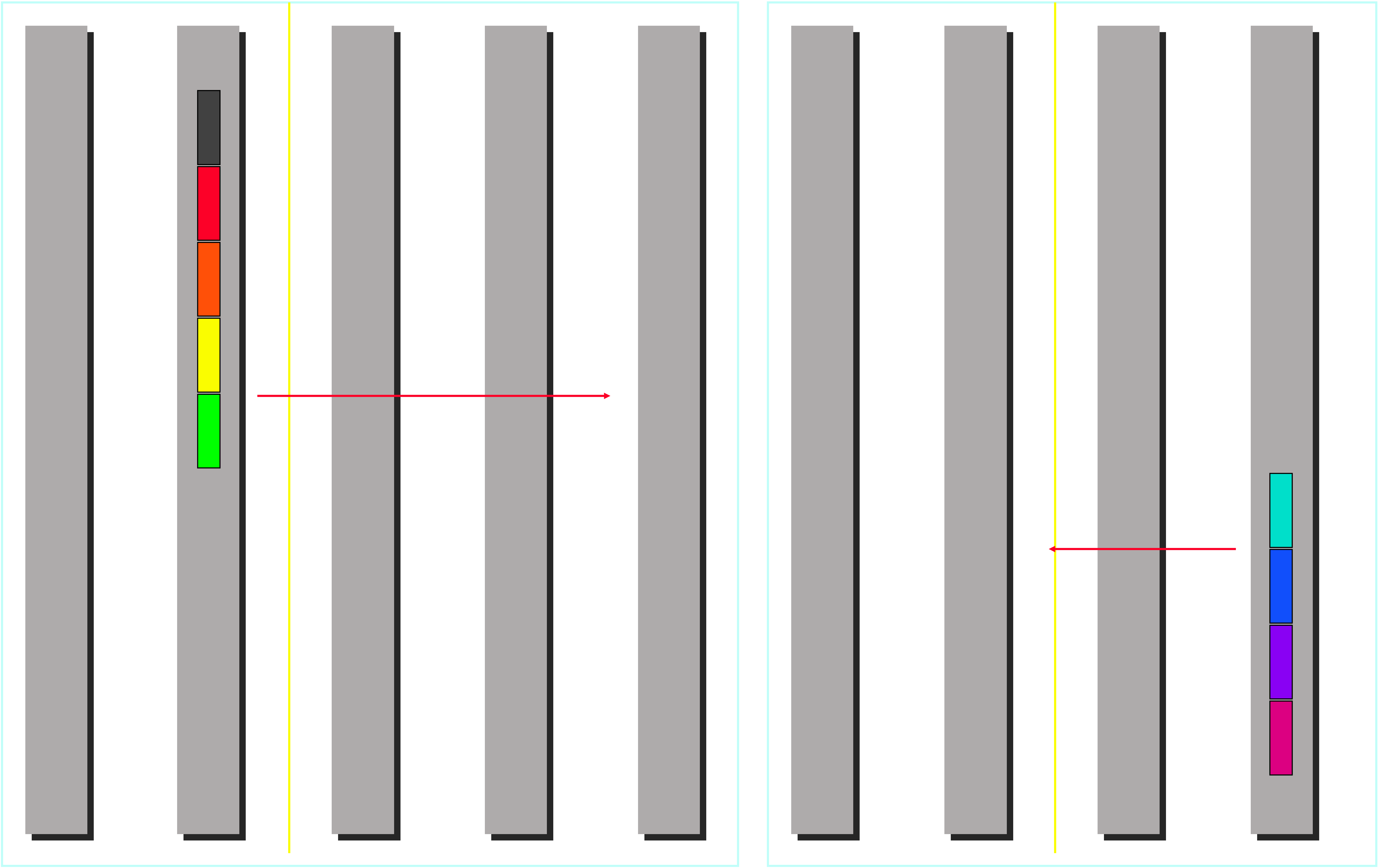


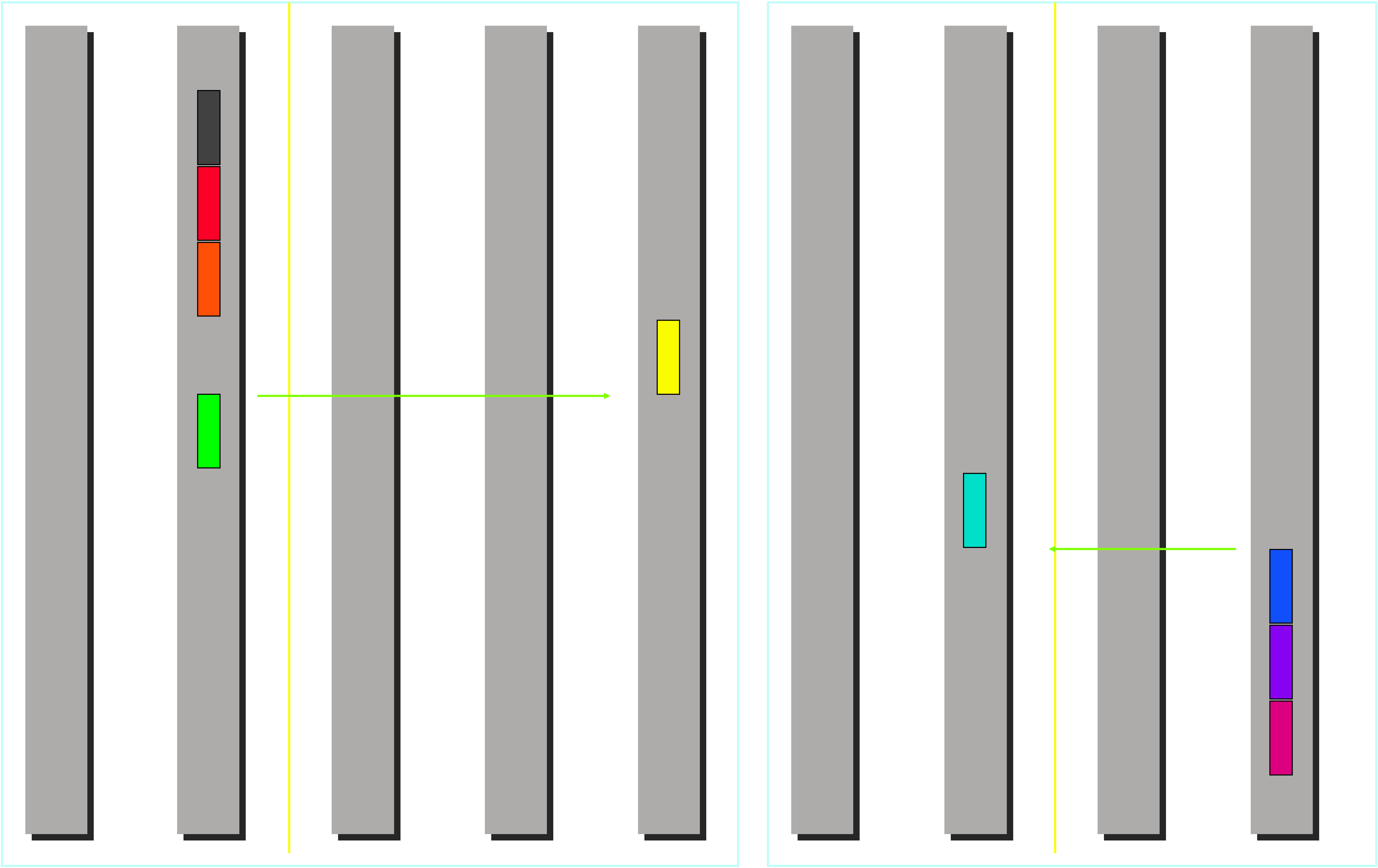


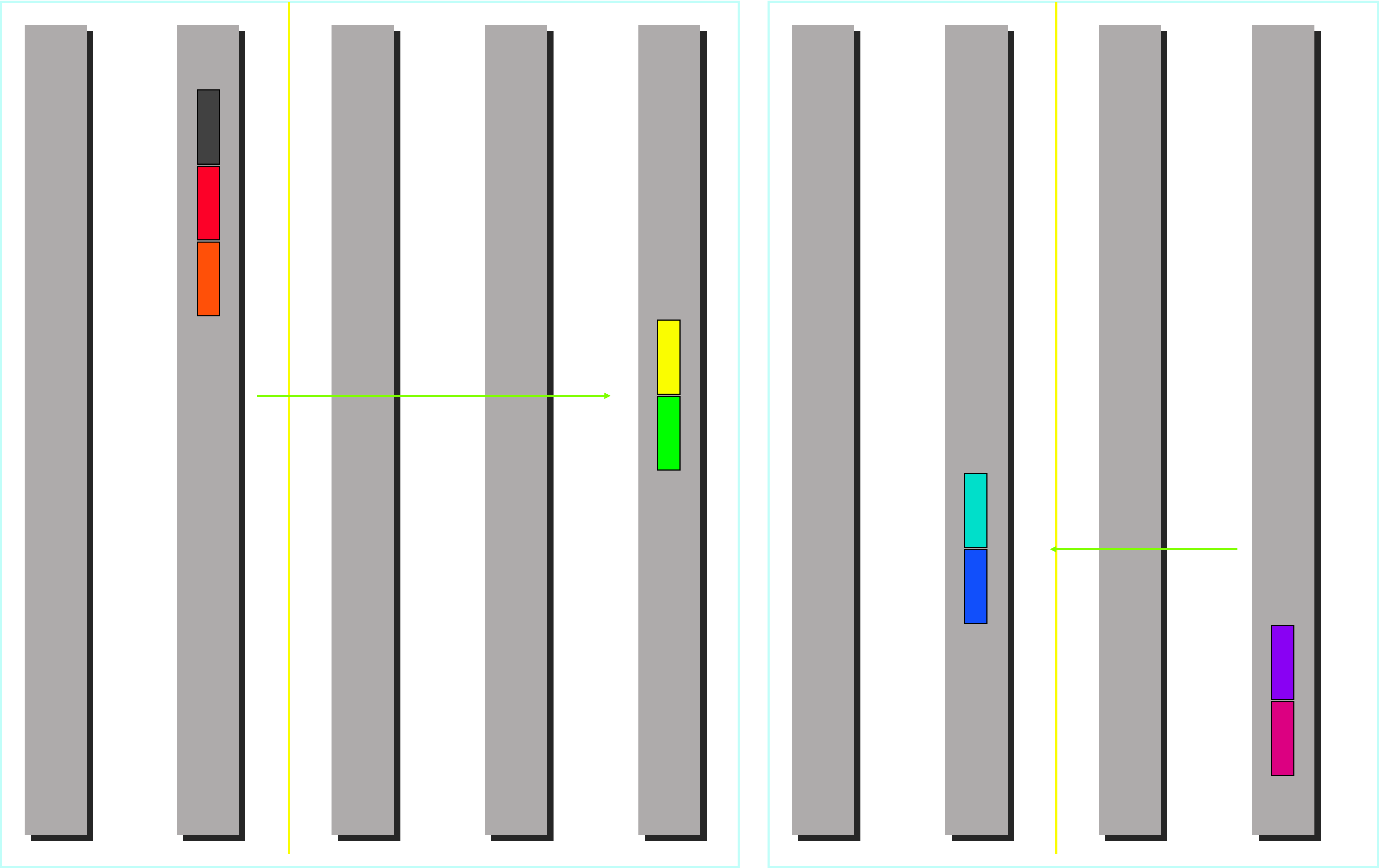


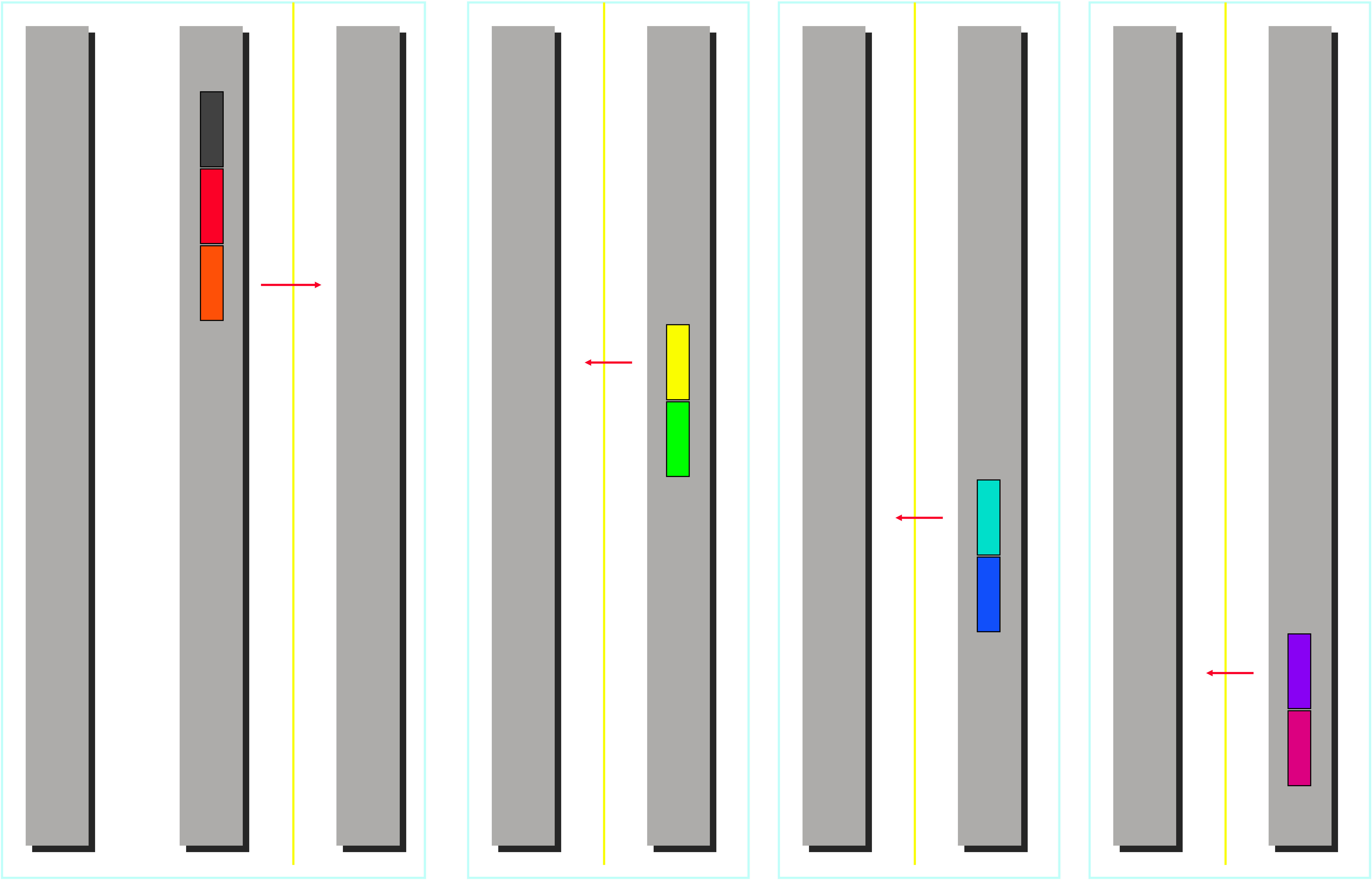


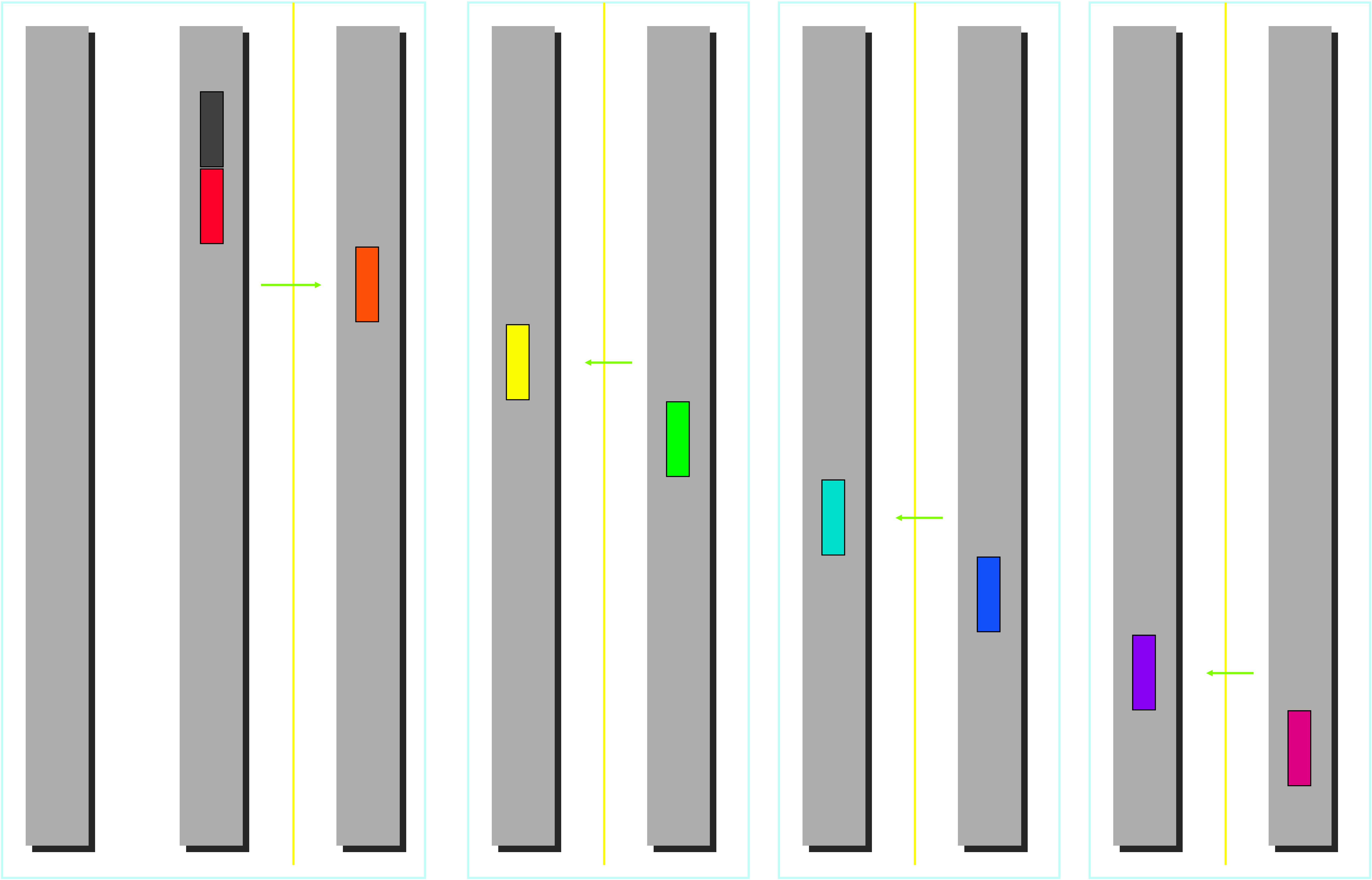


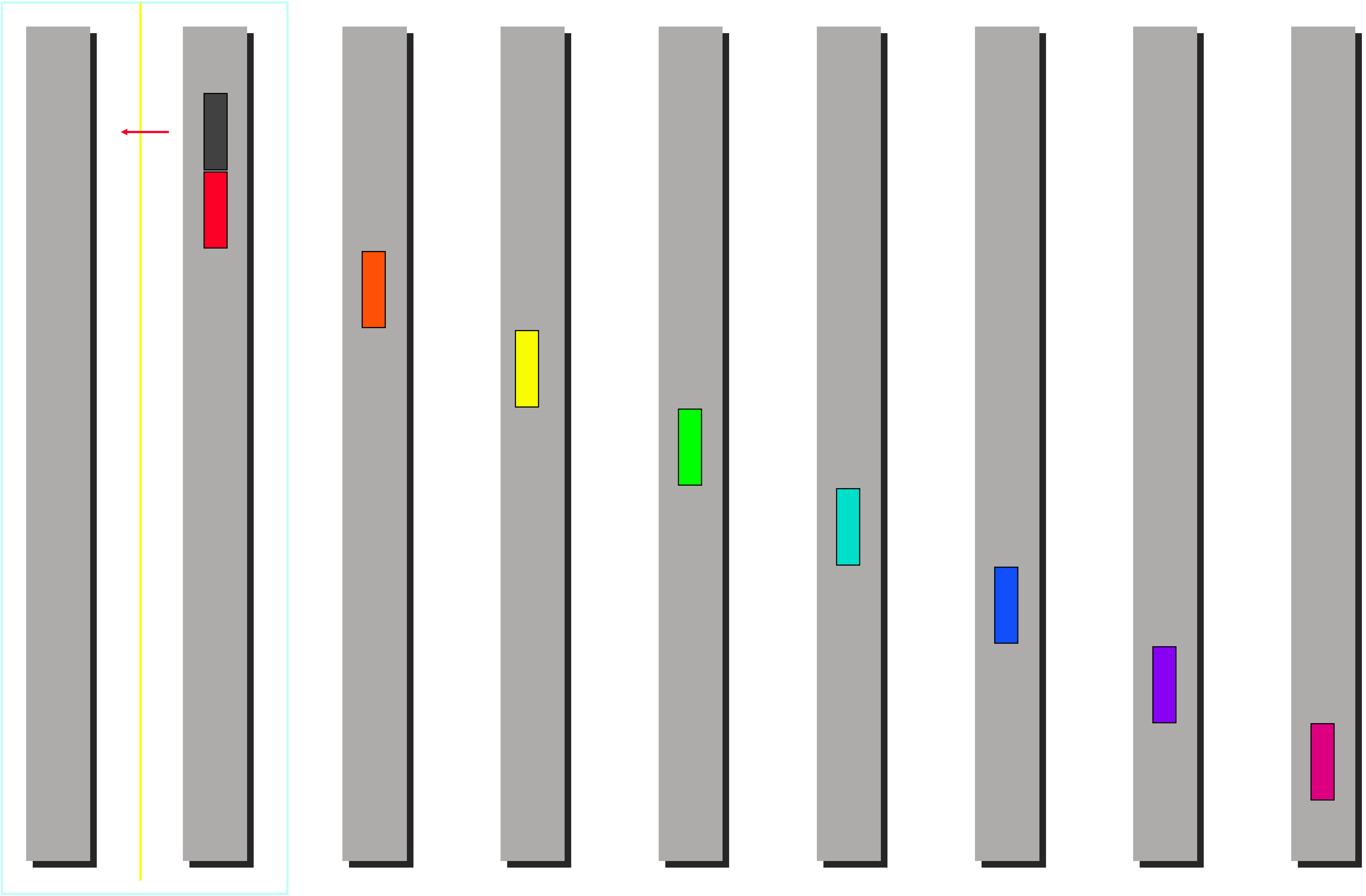


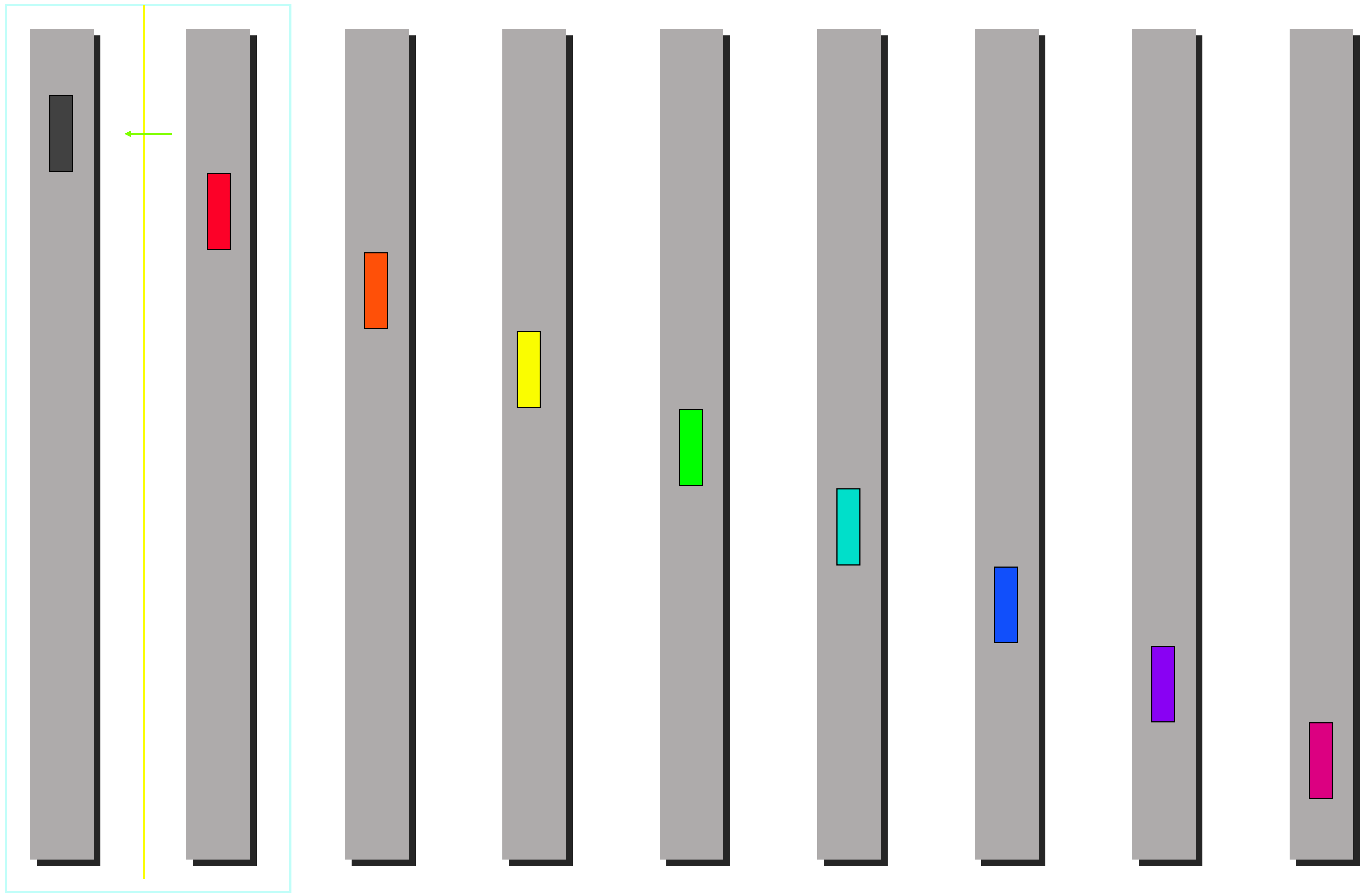


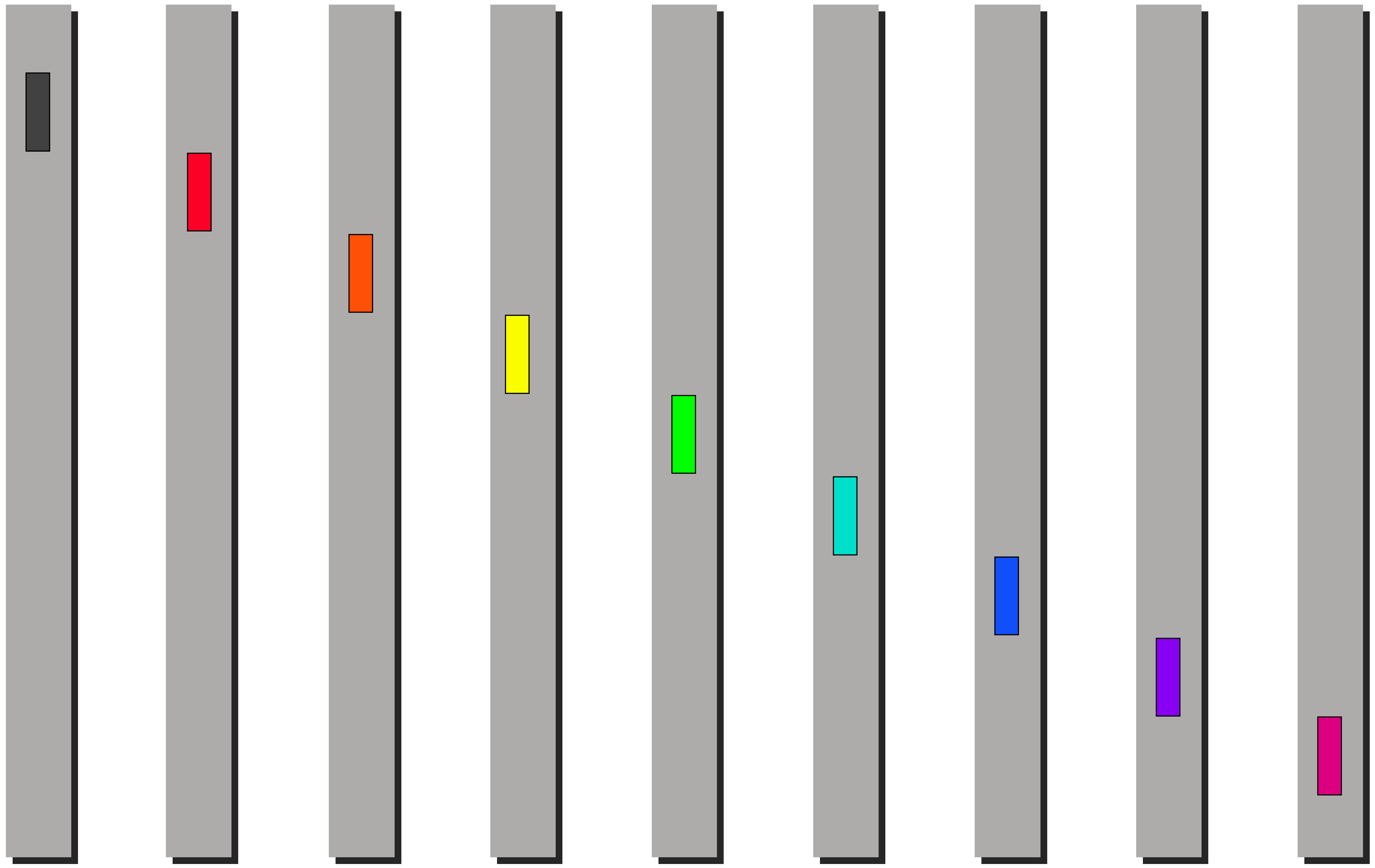












Cost of minimum spanning tree scatter

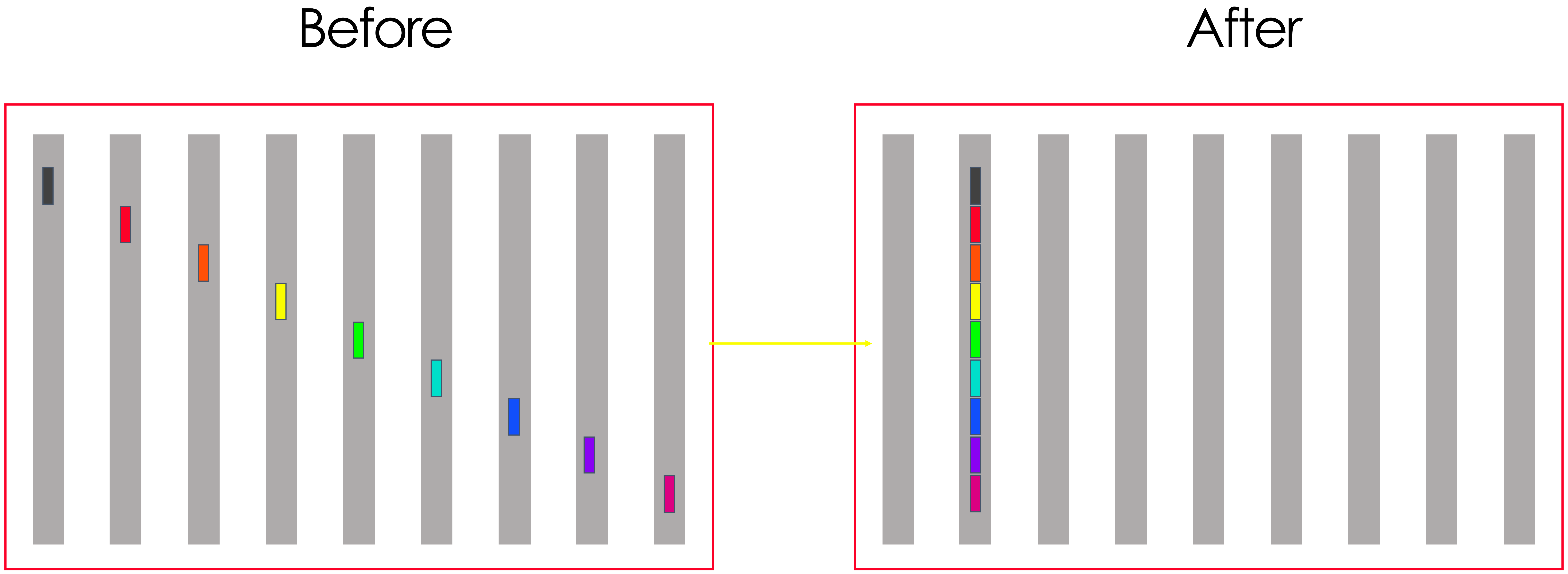
- Assumption: power of two number of nodes

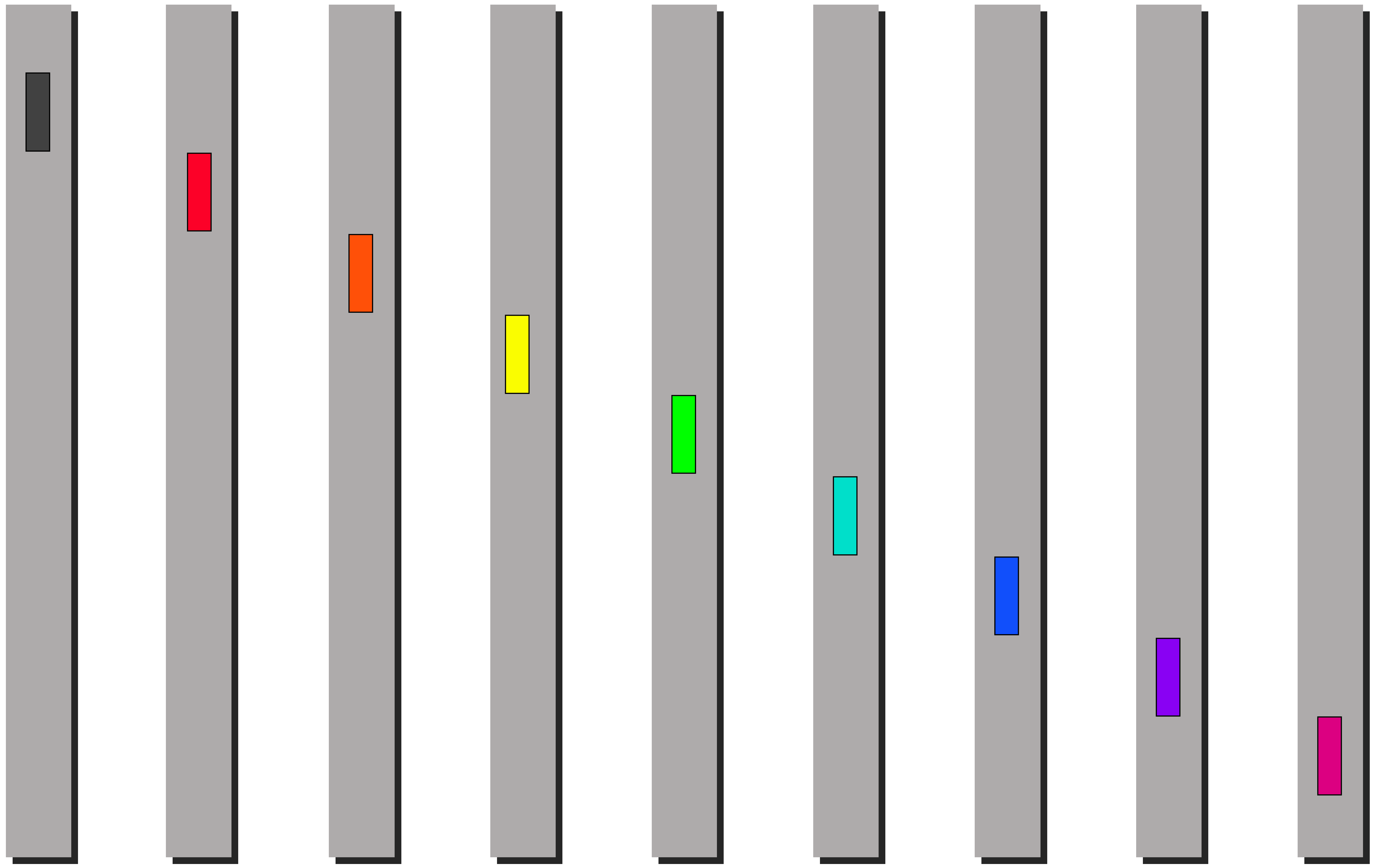
$$\sum_{k=1}^{log(p)} \left(\alpha + \frac{n}{2^k} \beta \right)$$

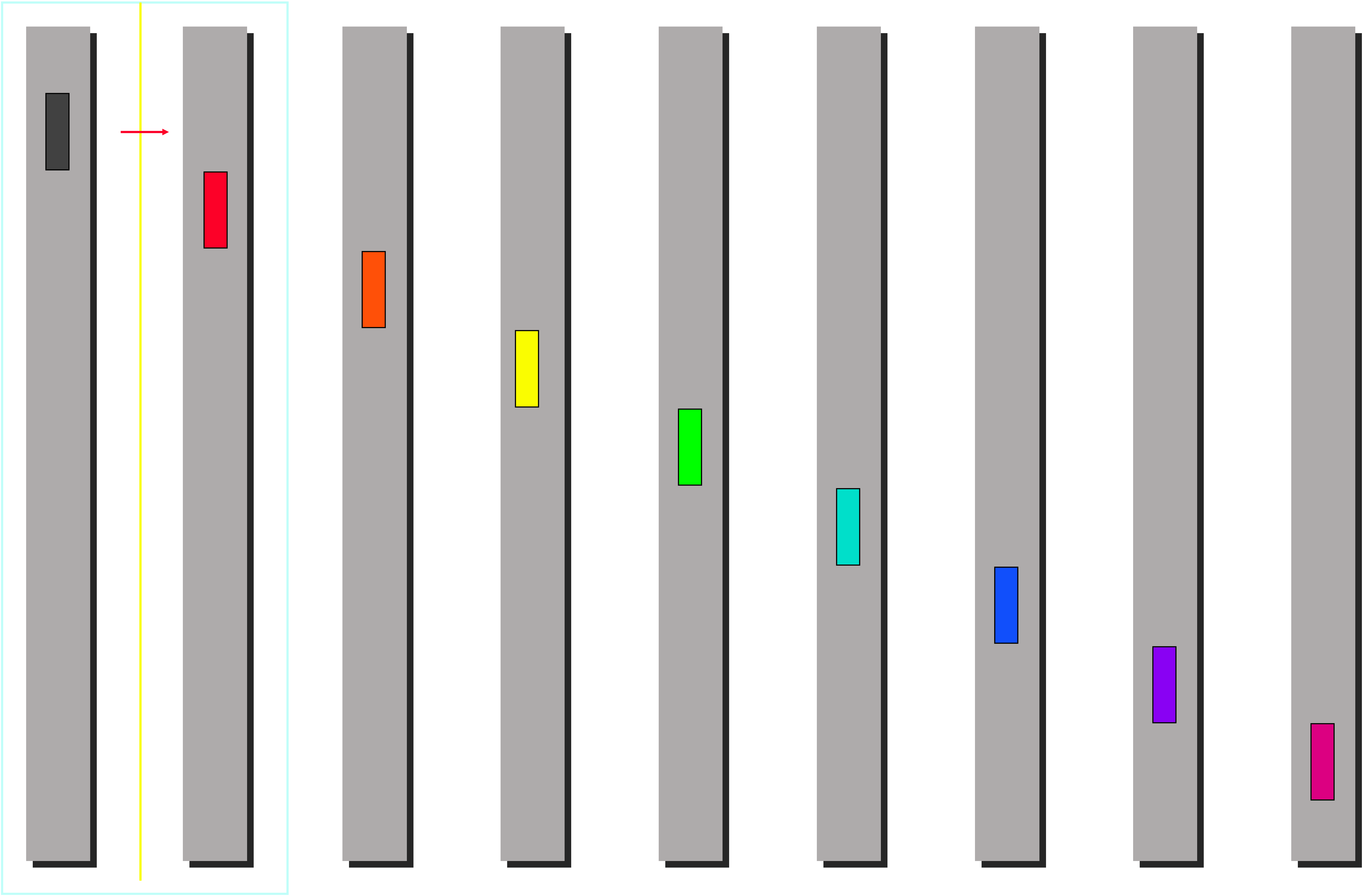
$$=$$

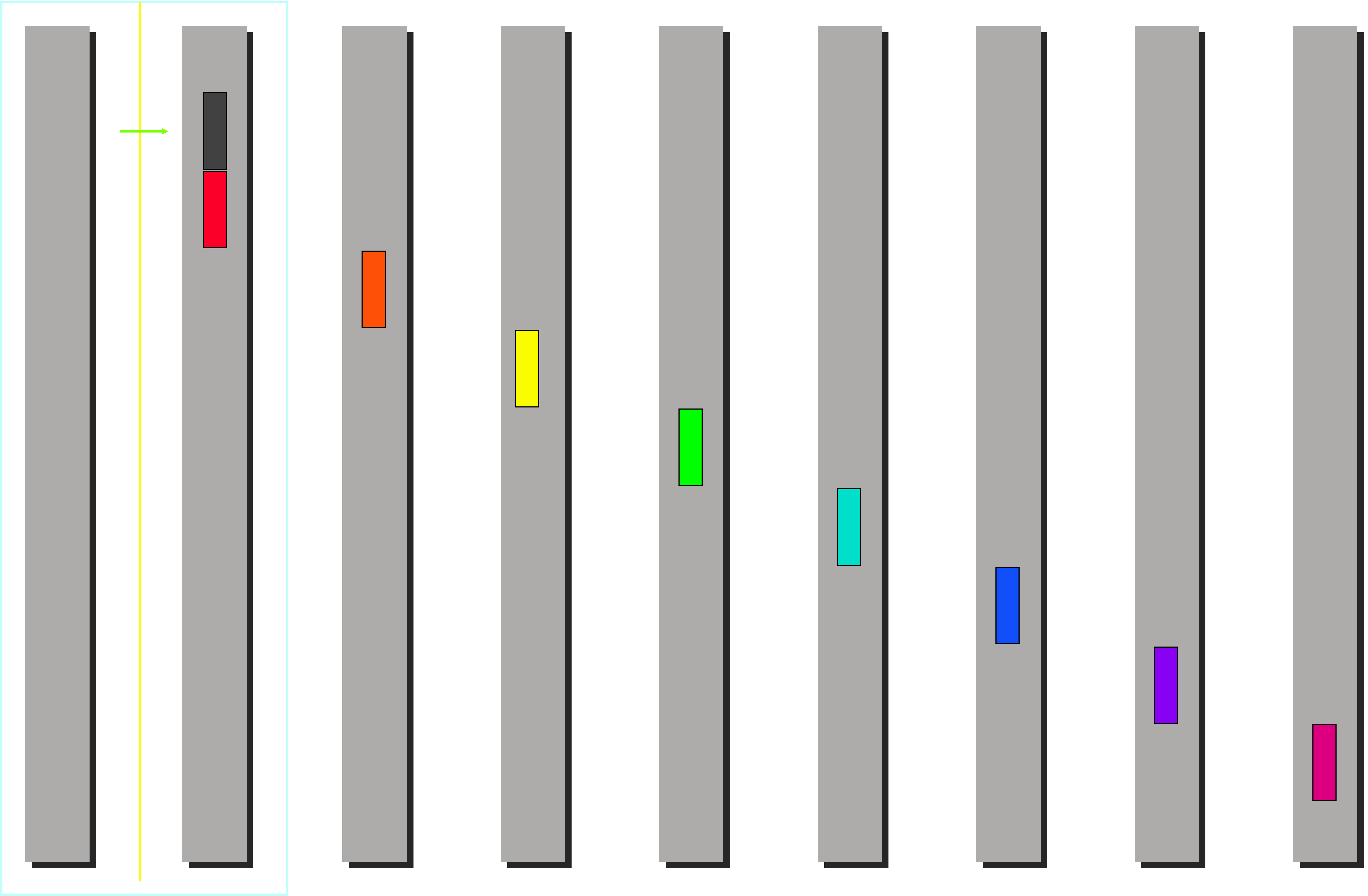
$$log(p) \quad \alpha + \frac{p-1}{p} n\beta$$

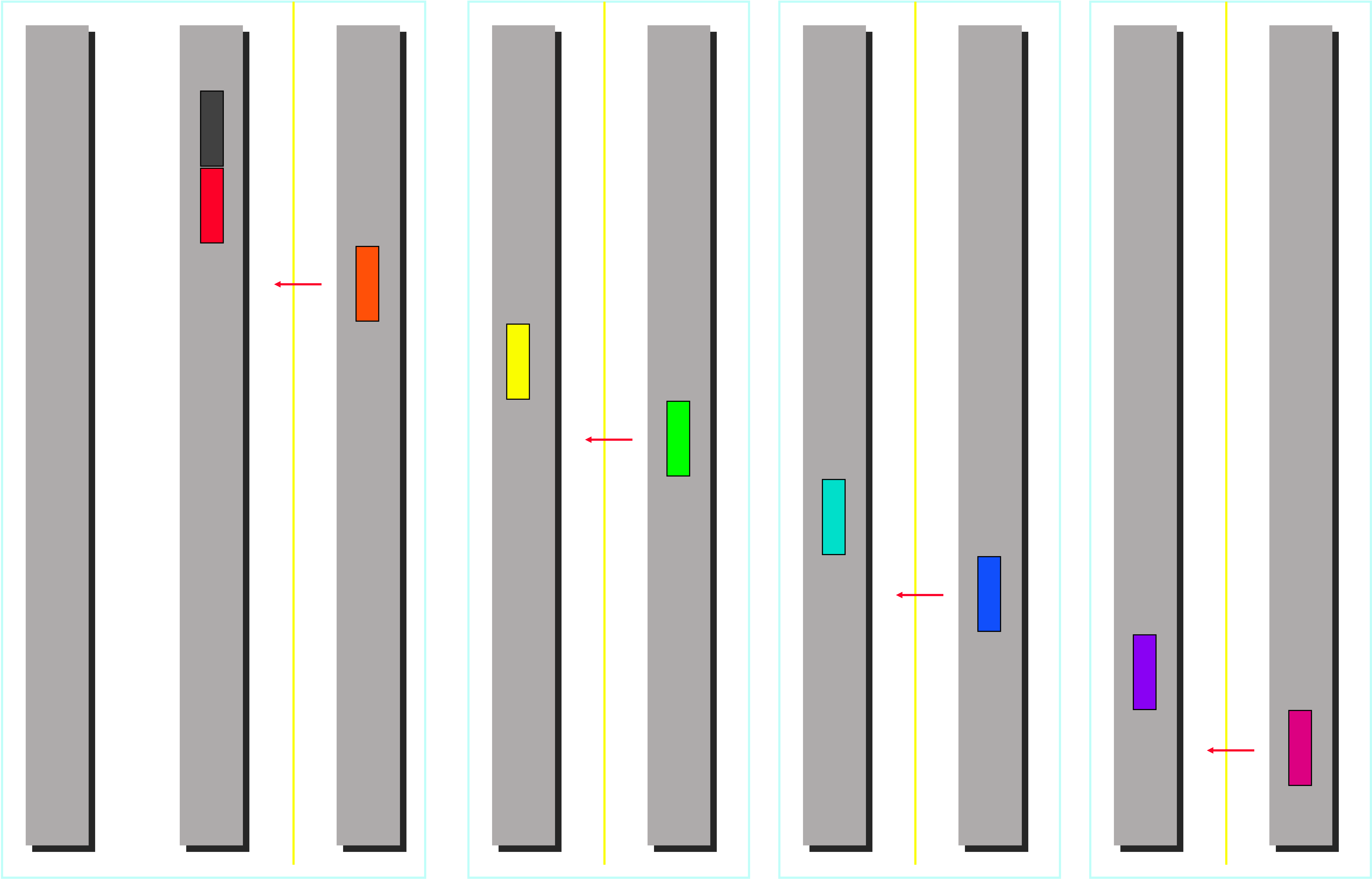
Gather

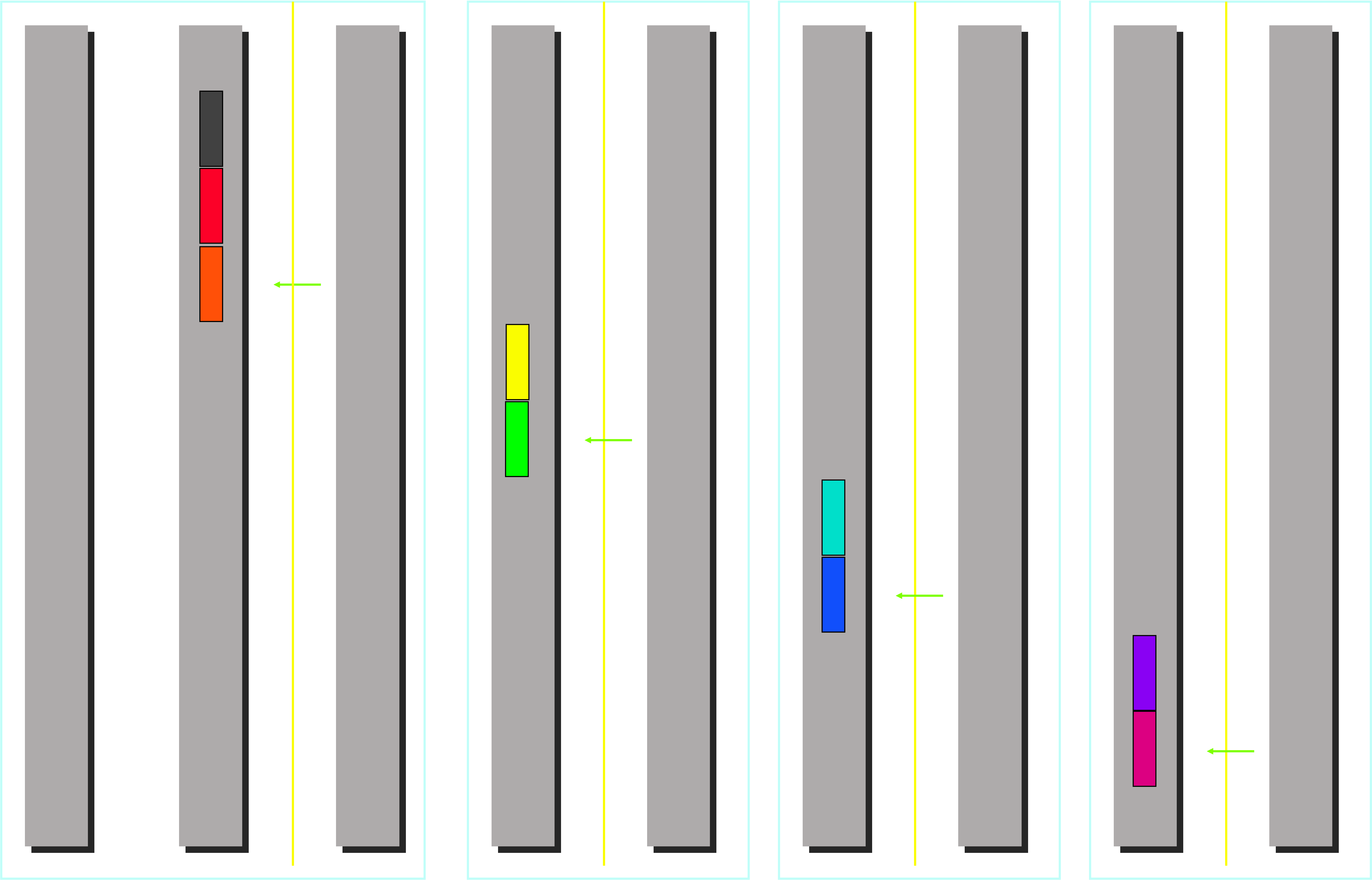


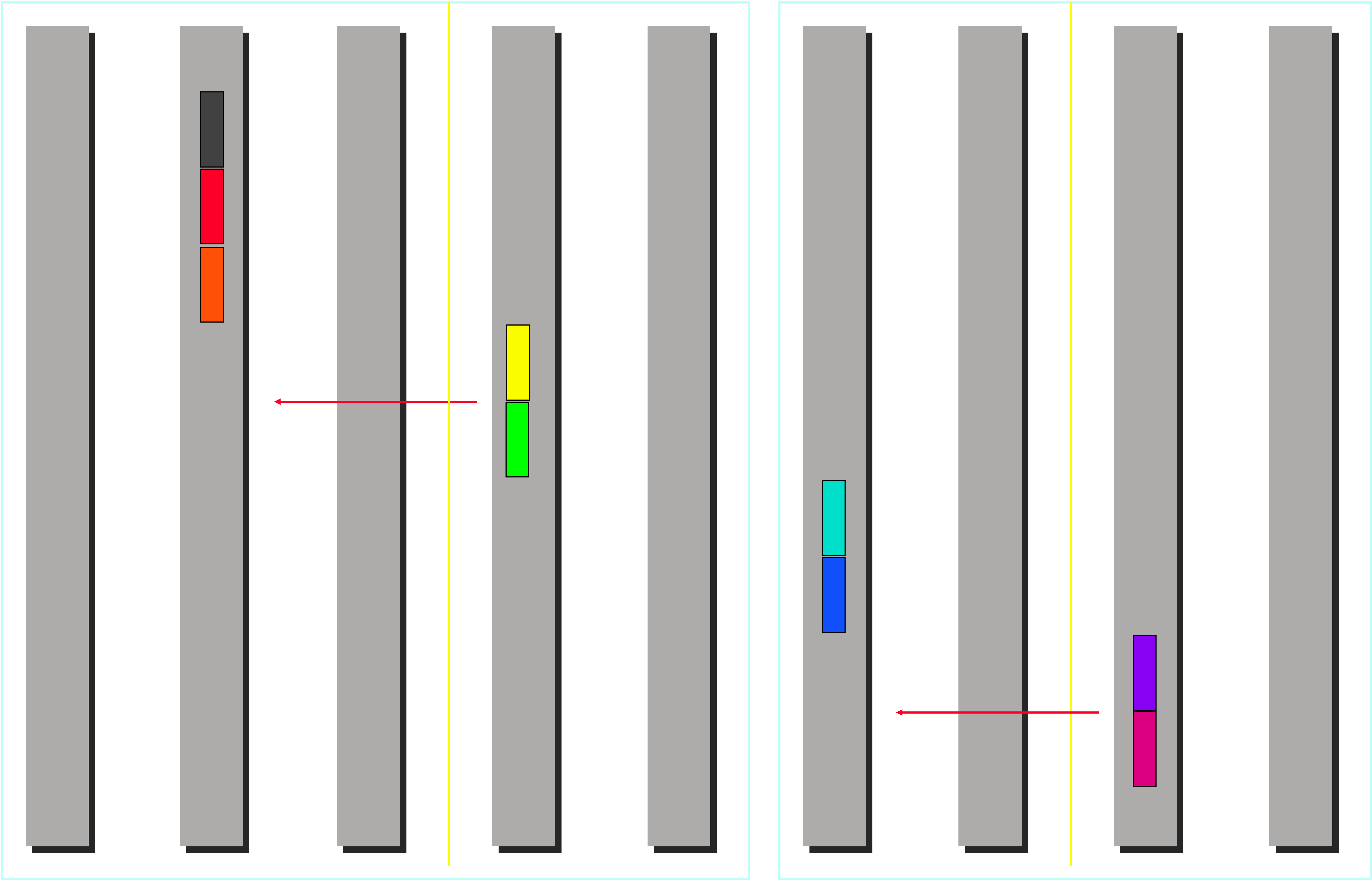


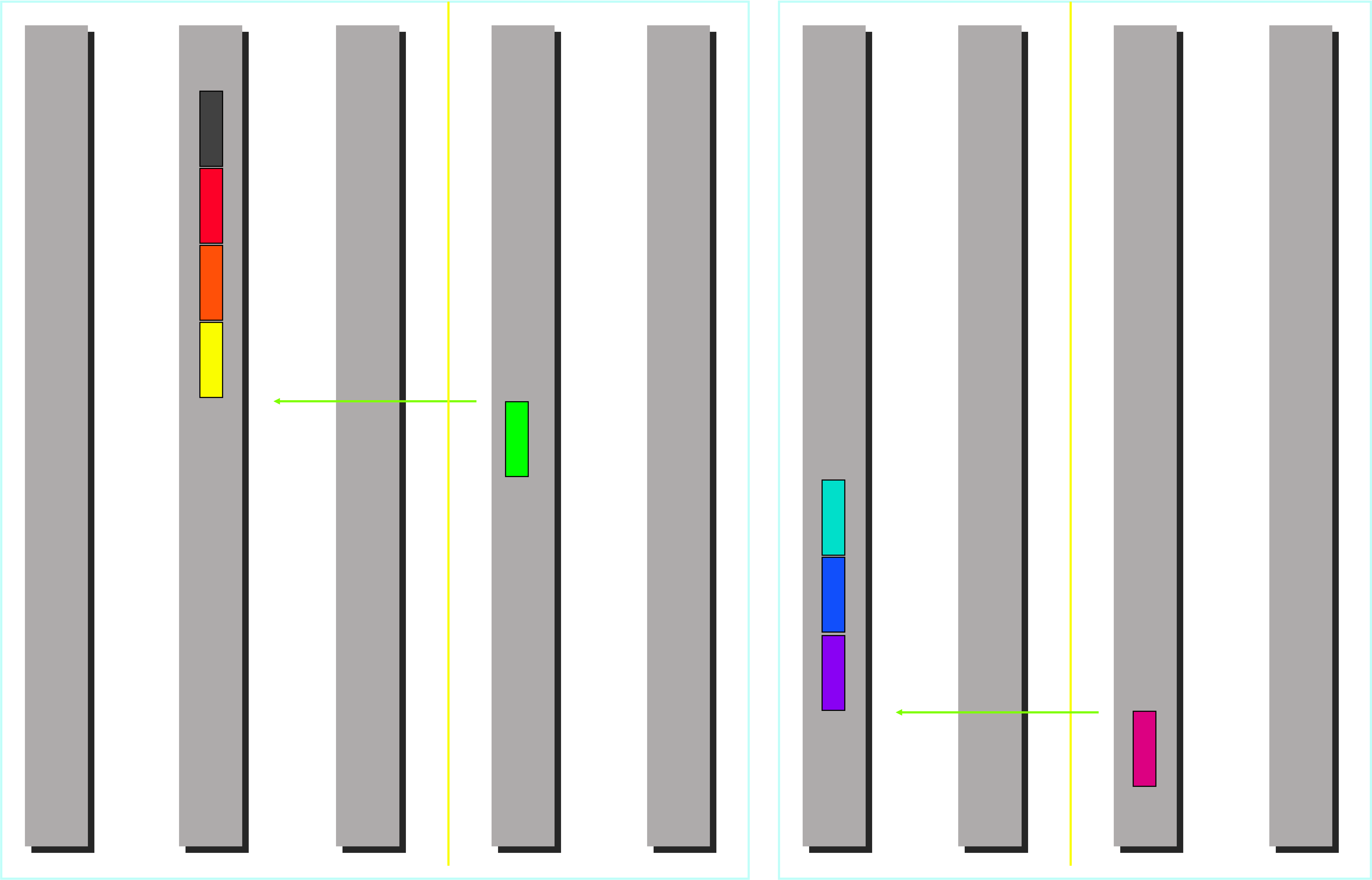


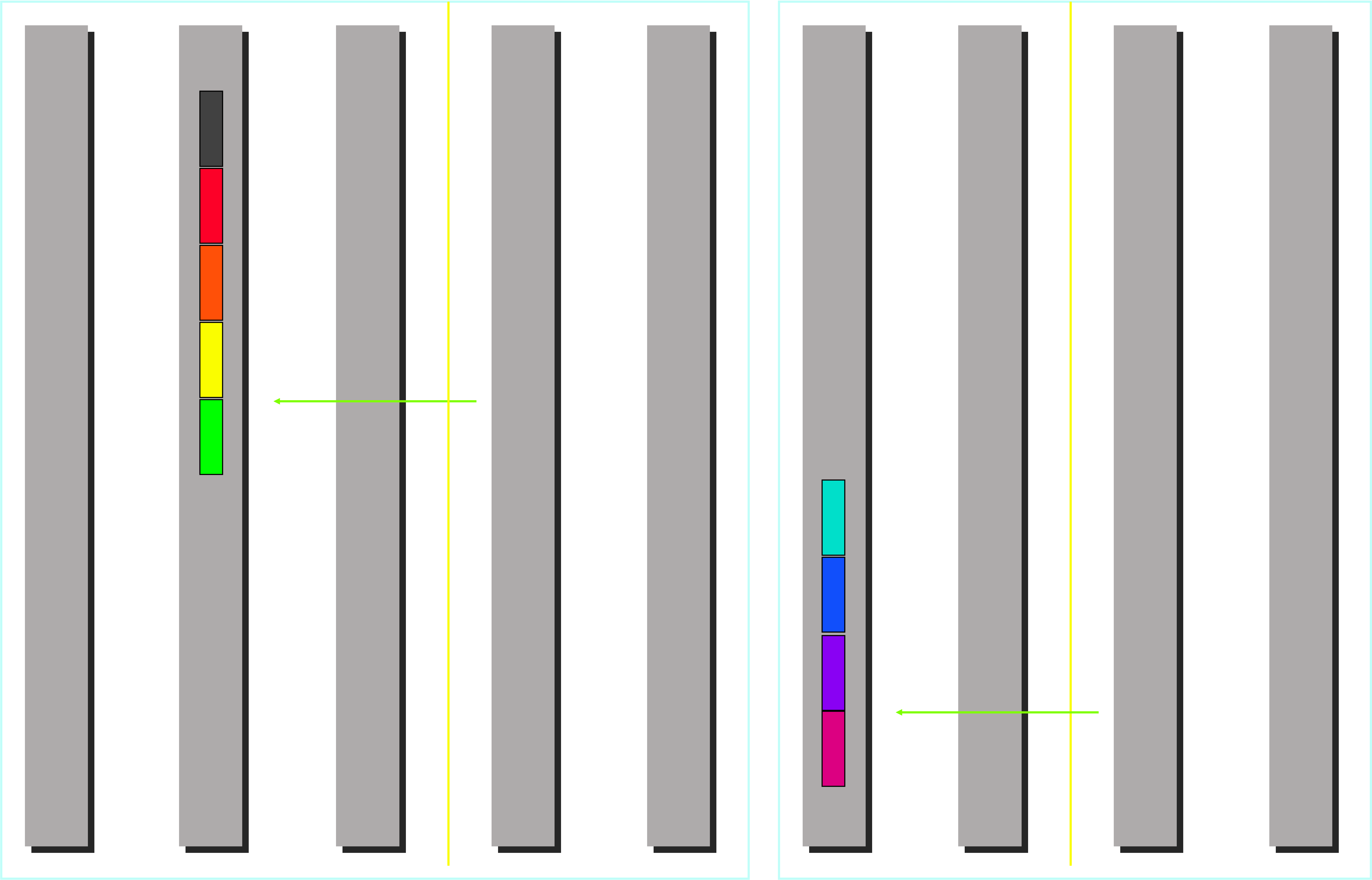


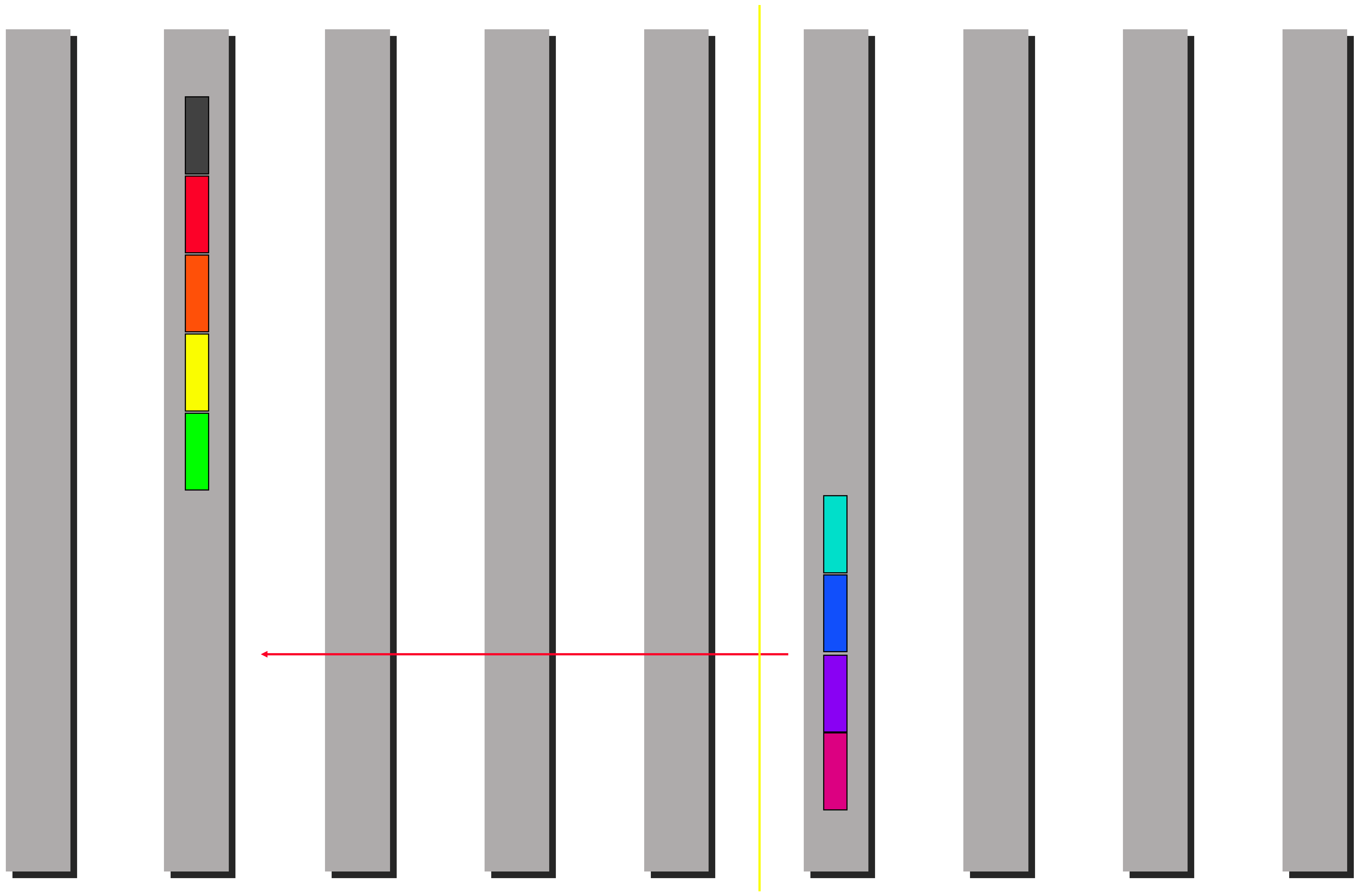


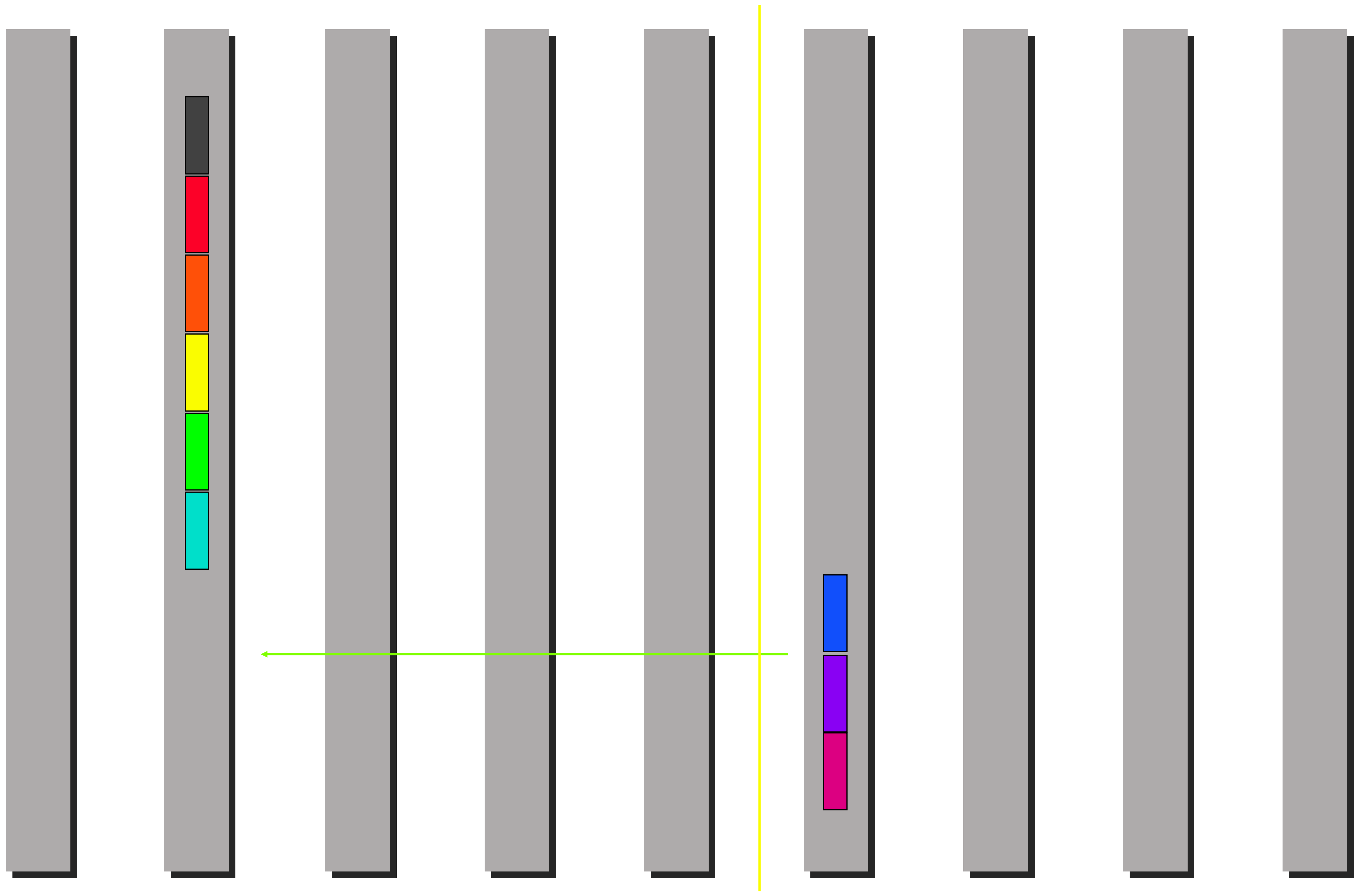


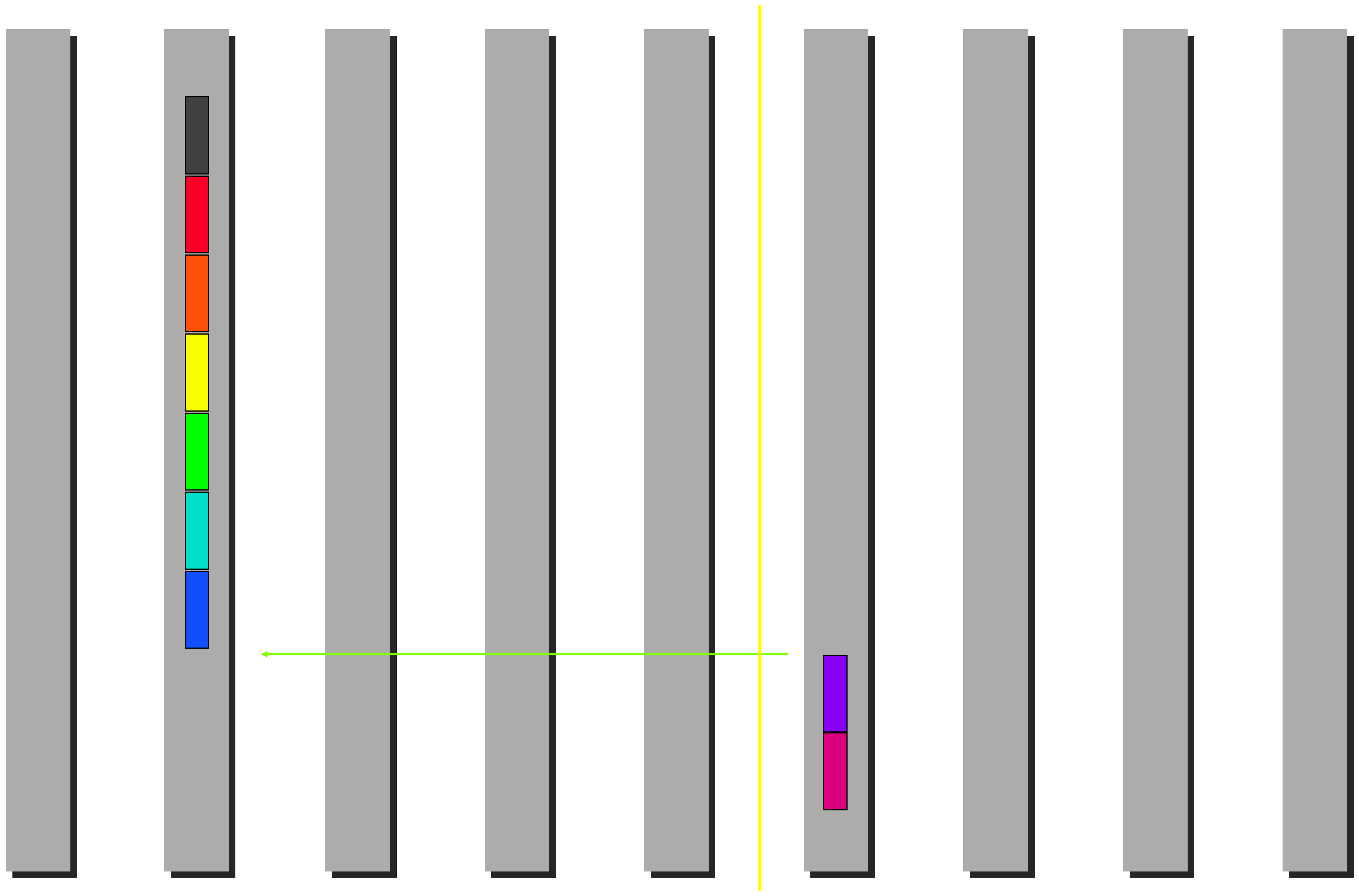


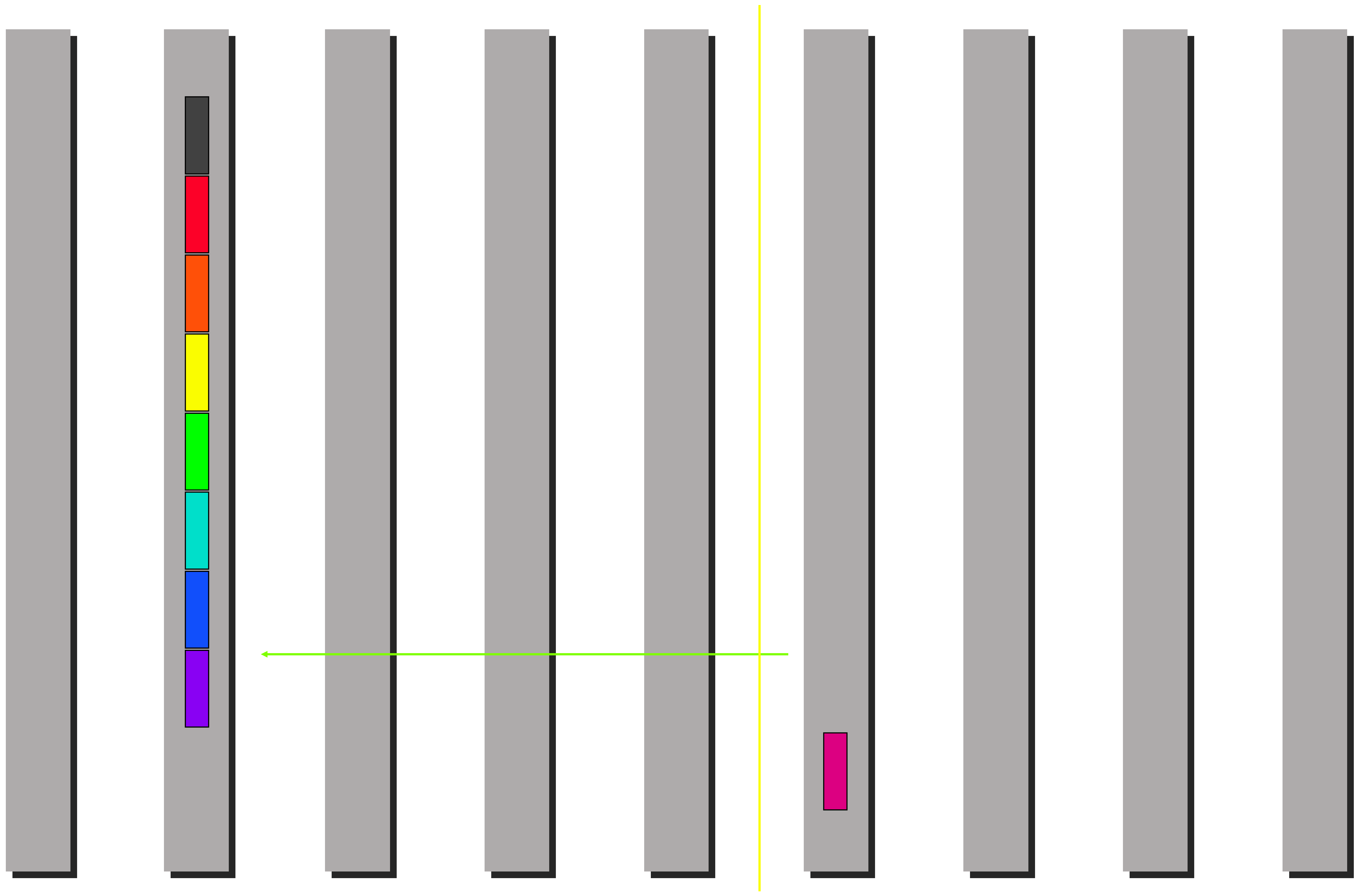


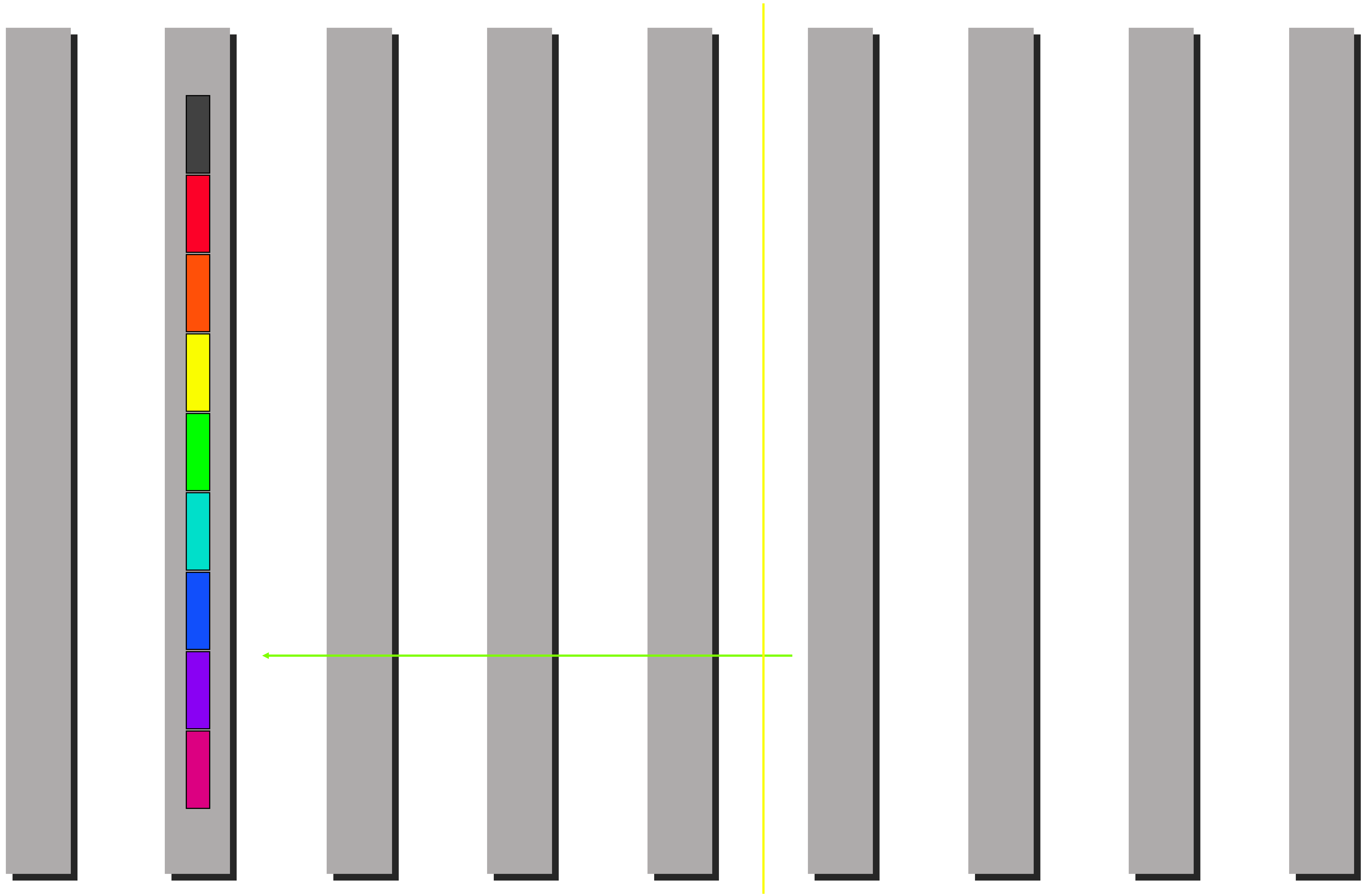


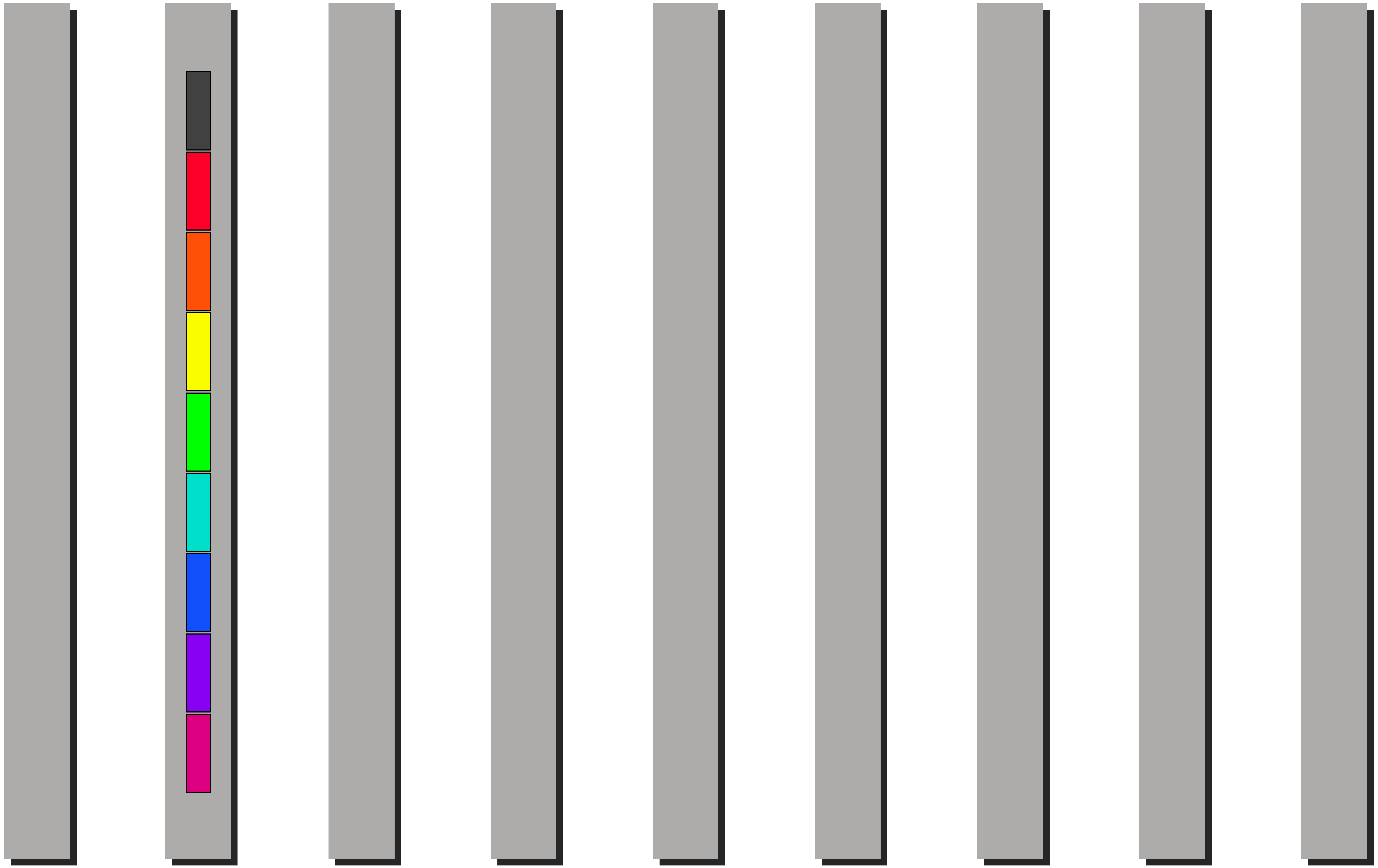












Cost of minimum spanning tree gather

- Assumption: power of two number of nodes

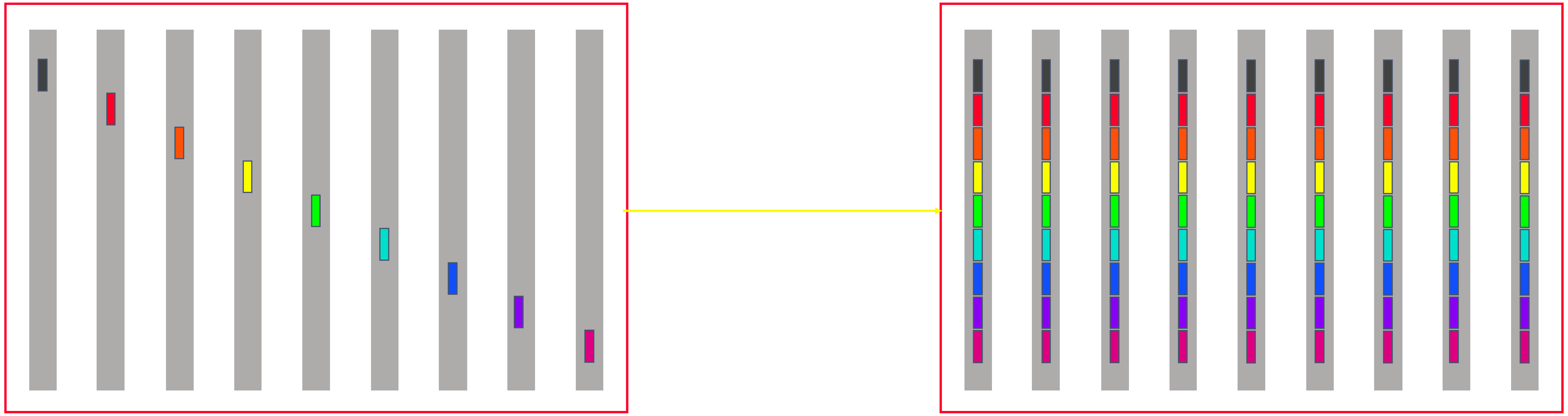
$$\sum_{k=1}^{log(p)} \left(\alpha + \frac{n}{2^k} \beta \right)$$

$$=$$

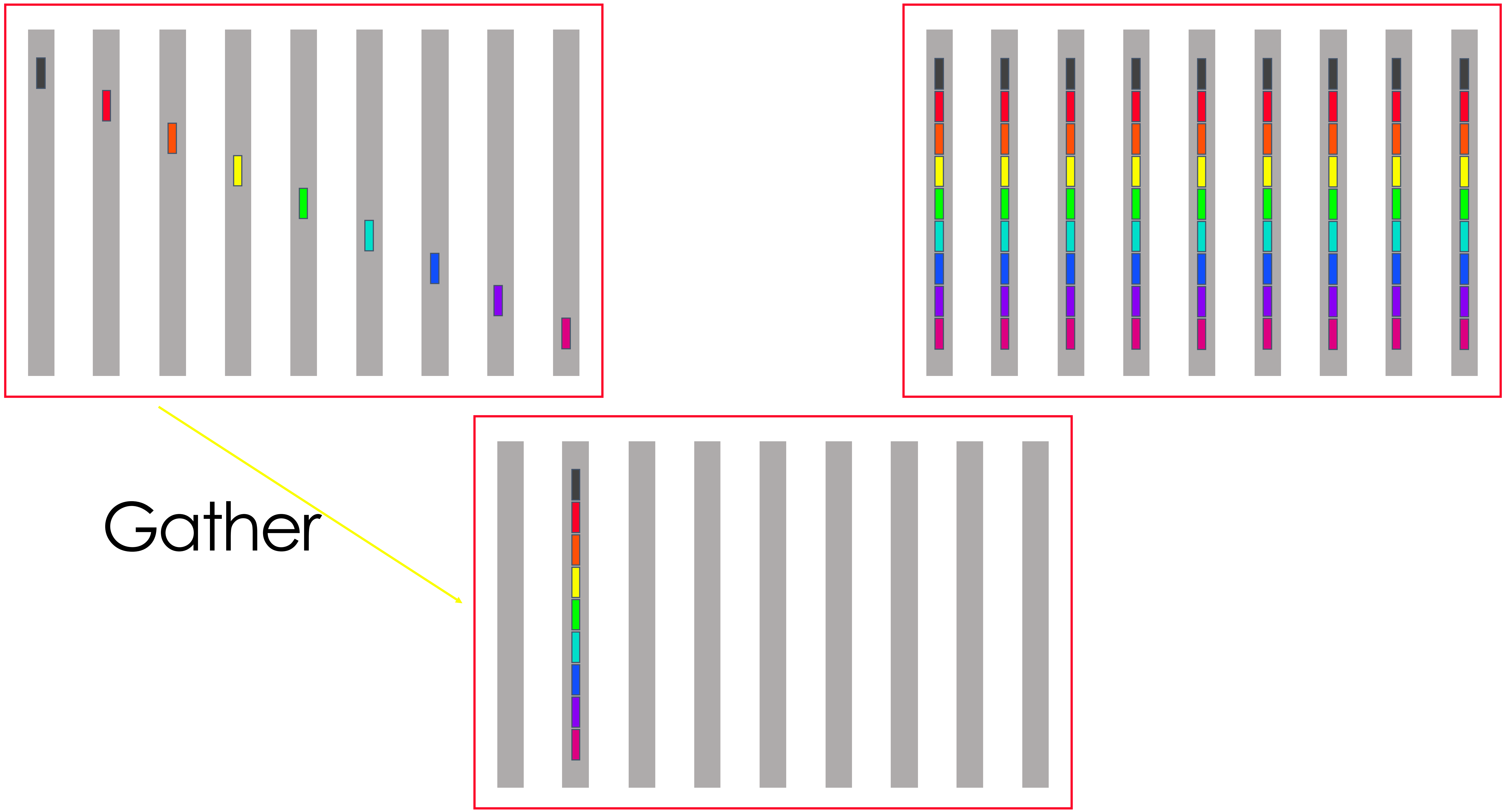
$$log(p) \quad \alpha + \frac{p-1}{p} n\beta$$

Using the building blocks

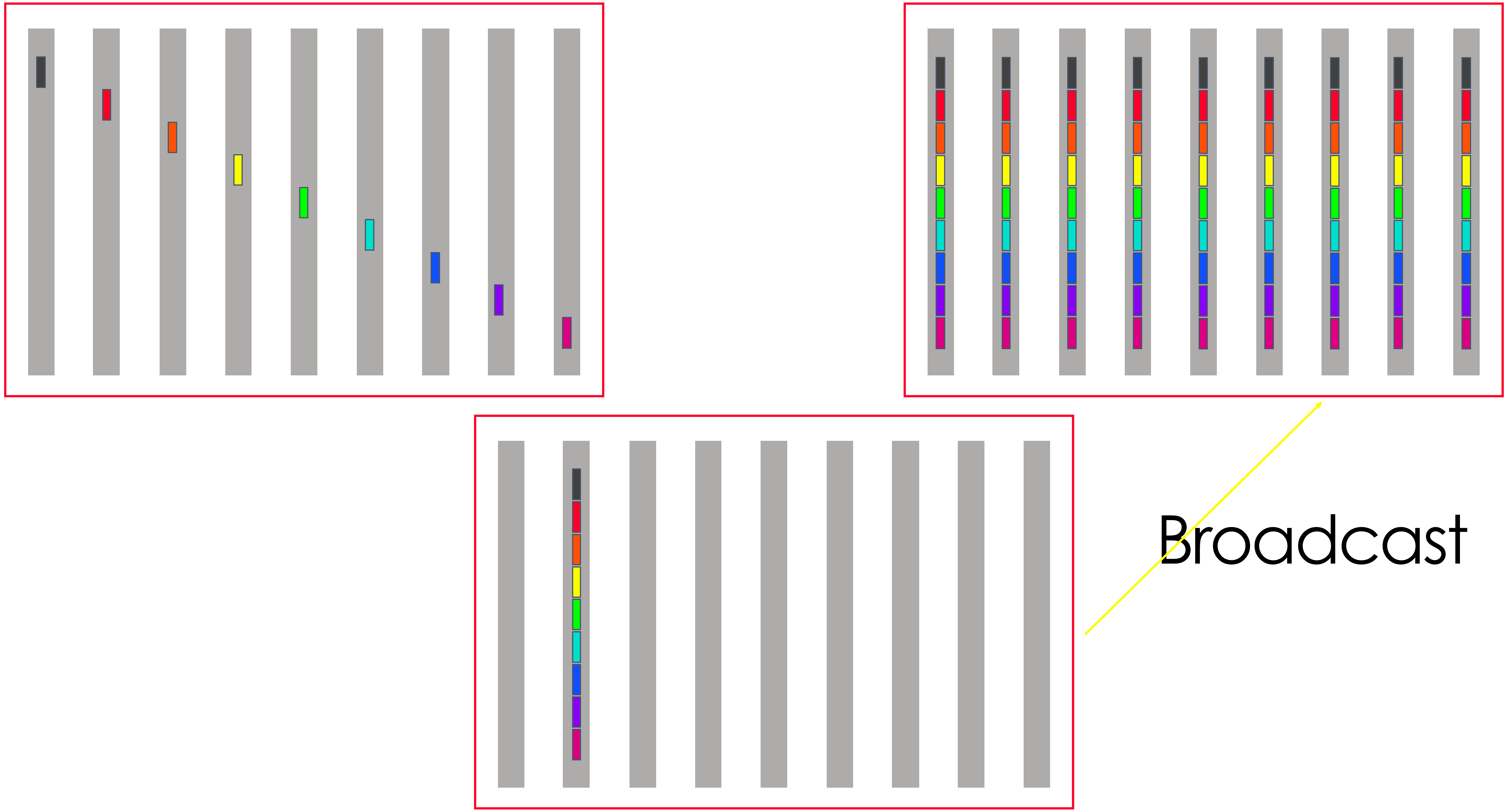
Allgather



Allgather



Allgather (short vector)



Cost of gather/broadcast allgather

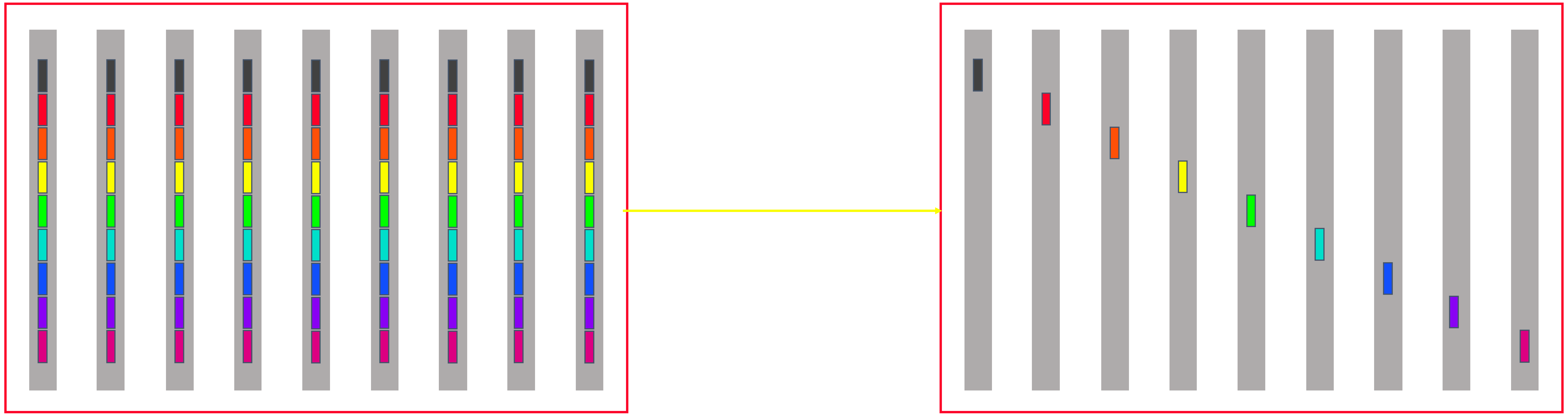
- Assumption: power of two number of nodes

gather $$log(p)\alpha + \frac{p-1}{p}n\beta$$

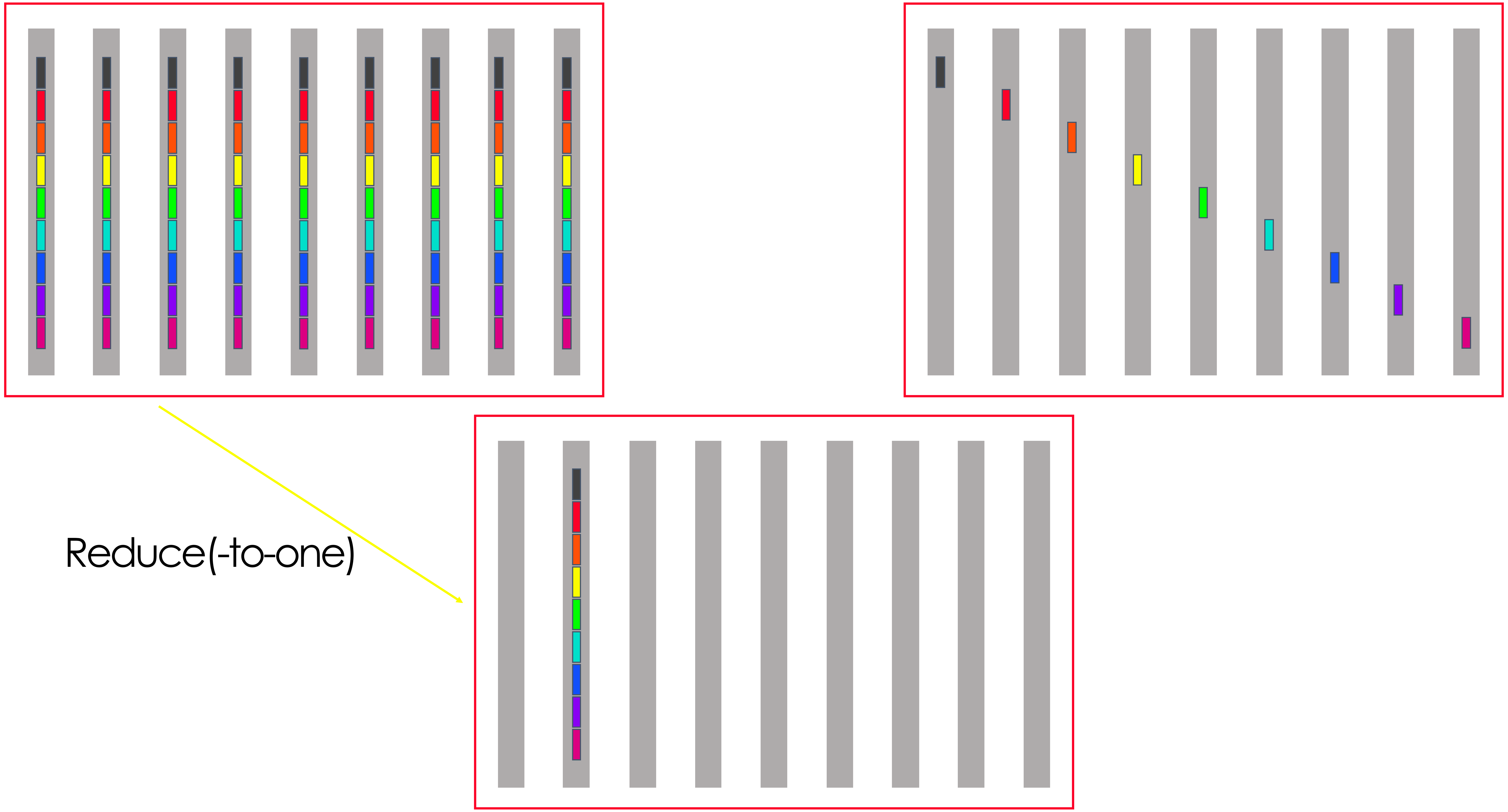
broadcast $$log(p)(\alpha + n\beta)$$

$$2log(p)\alpha + \left(\frac{p-1}{p} + log(p)\right)n\beta$$

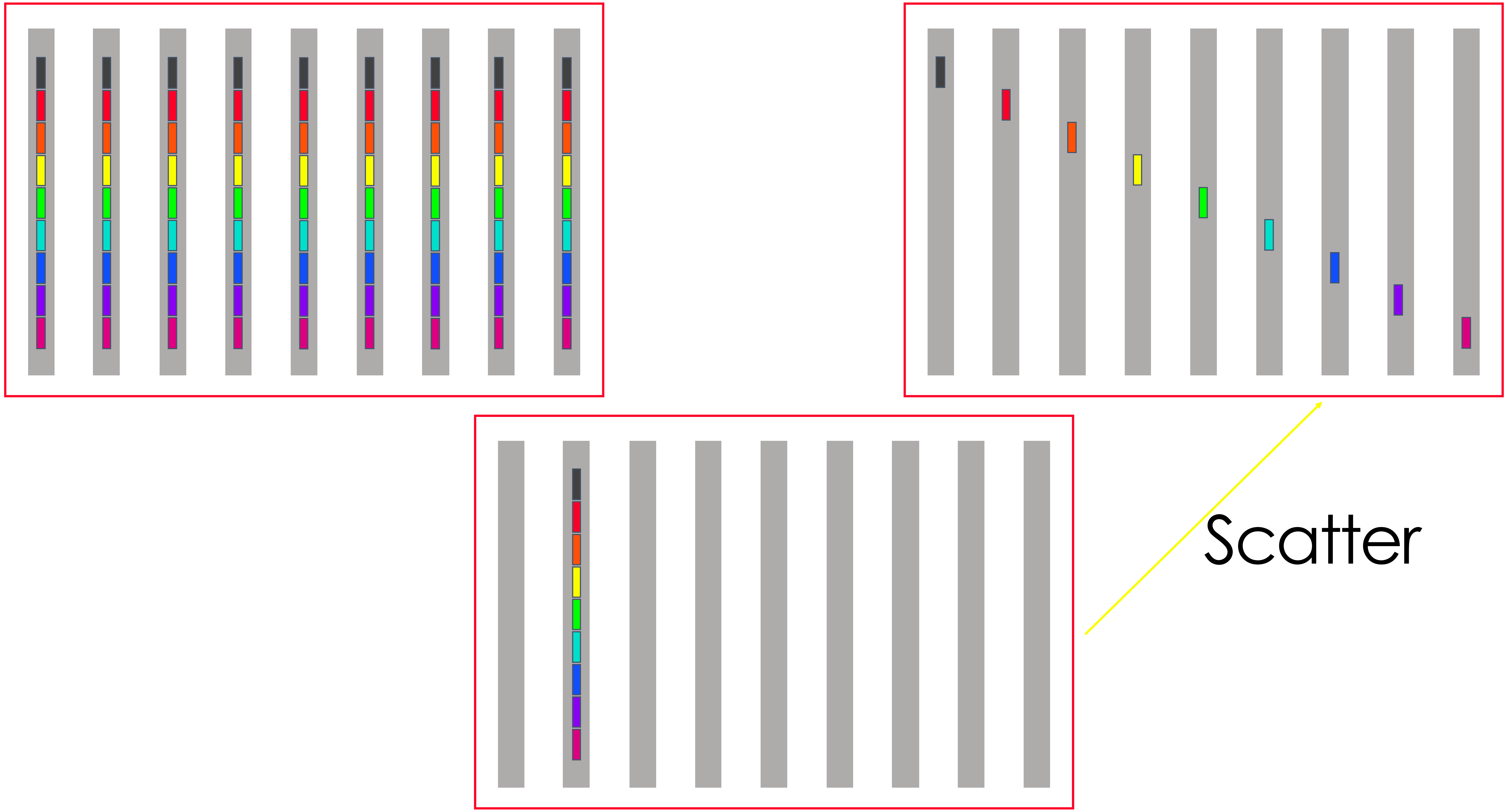
Reduce-scatter (small message)



Reduce-scatter (short vector)



Reduce-scatter (short vector)



Cost of Reduce(-to-one)/scatter Reduce-scatter

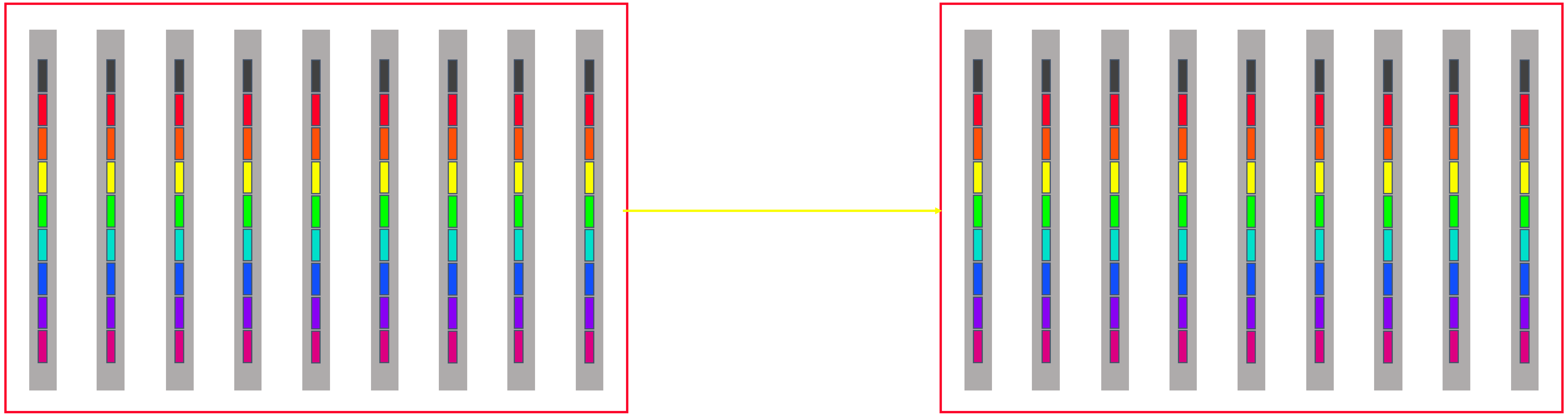
- Assumption: power of two number of nodes

Reduce(-to-one) $log(p)(\alpha + n\beta + n\gamma)$

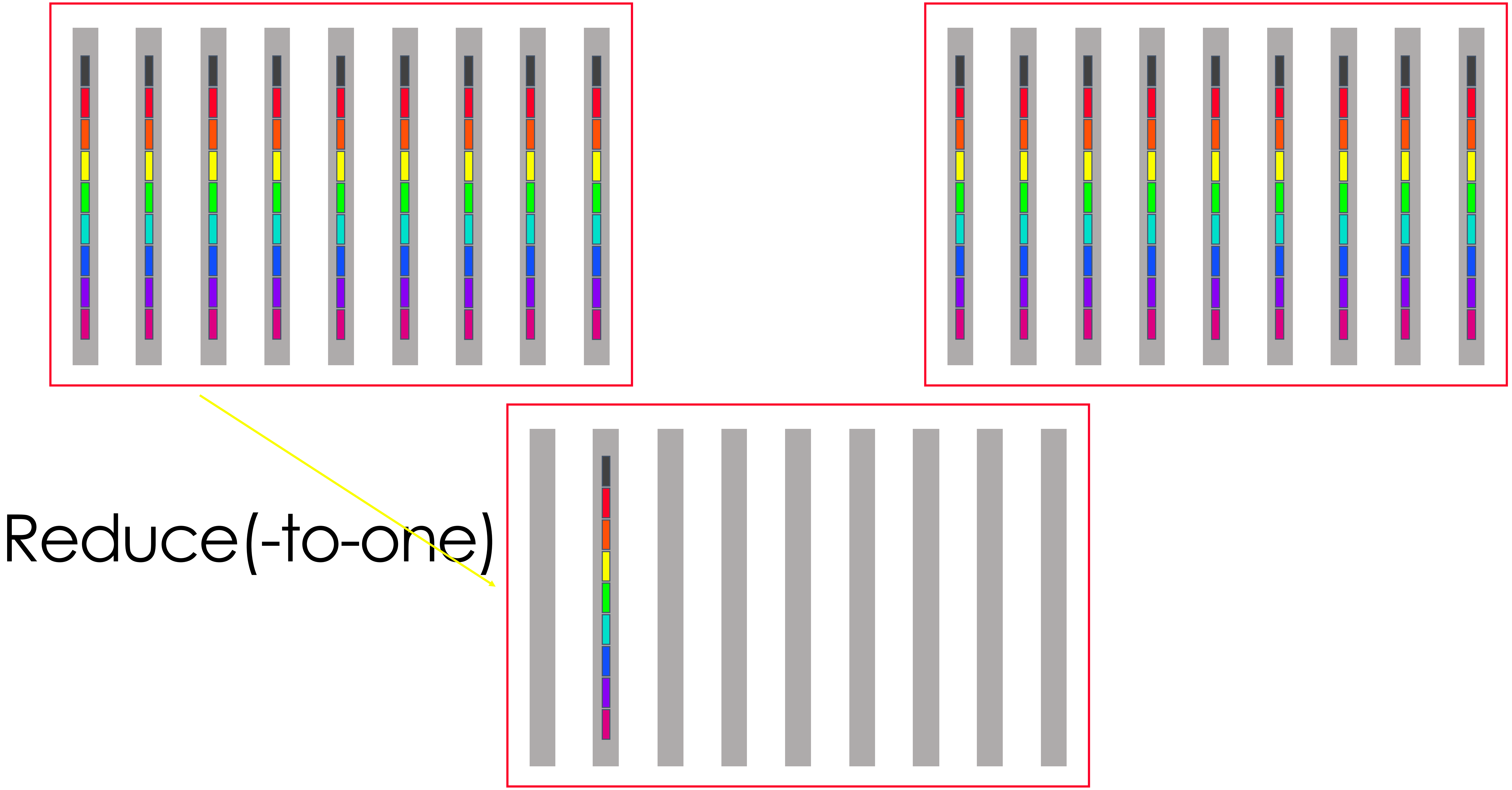
scatter $log(p)\alpha + \frac{p-1}{p}n\beta$

$$2log(p)\alpha + \left(\frac{p-1}{p} + log(p)\right)n\beta + log(p)n\gamma$$

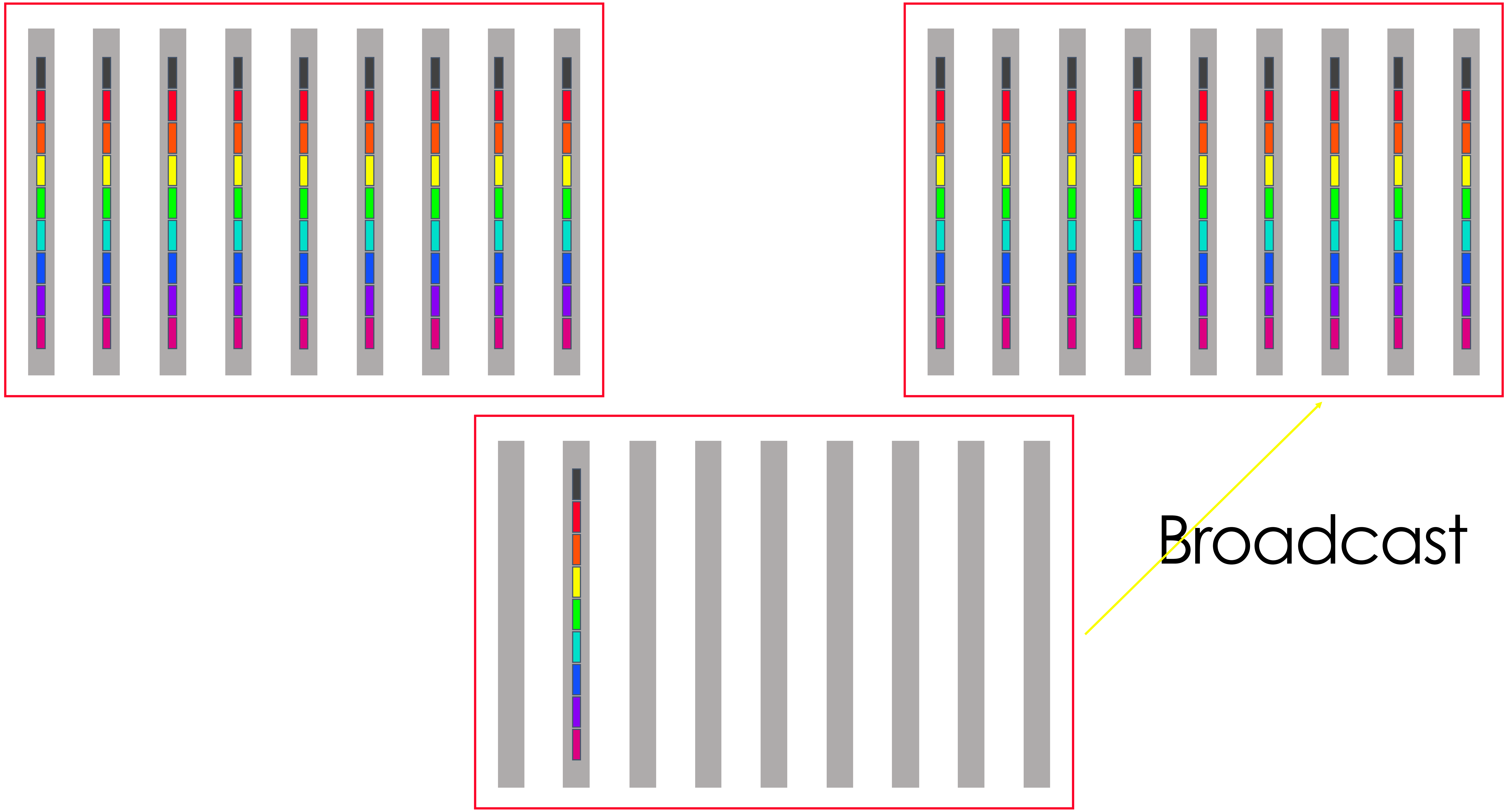
Allreduce (Latency-optimized)



Allreduce (Latency-optimized)



Allreduce
(short vector)



Cost of reduce(-to-one)/broadcast Allreduce

- Assumption: power of two number of nodes

Reduce(-to-one) $log(p)(\alpha + n\beta + n\gamma)$

broadcast $log(p)(\alpha + n\beta)$

$$2log(p)\alpha + 2log(p)n\beta + log(p)n\gamma$$

Recap

Reduce(-to-one)

$log(p)(\alpha + n\beta + n\gamma)$

Scatter

$log(p)\alpha + \frac{p-1}{p}n\beta$

Gather

$log(p)\alpha + \frac{p-1}{p}n\beta$

Broadcast

$log(p)(\alpha + n\beta)$

Reduce-scatter

Allreduce

Allgather

Recap

Reduce(-to-one)

$log(p)(\alpha + n\beta + n\gamma)$

Scatter

$log(p)\alpha + \frac{p-1}{p} n\beta$

Gather

$log(p)\alpha + \frac{p-1}{p} n\beta$

Broadcast

$log(p)(\alpha + n\beta)$

Reduce-scatter

$2log(p)\alpha + log(p)n(\beta + \gamma) + \frac{p-1}{p} n\beta$

Allreduce

Allgather

Recap

Reduce(-to-one)

$log(p)(\alpha + n\beta + n\gamma)$

Scatter

$log(p)\alpha + \frac{p-1}{p} n\beta$

Gather

$log(p)\alpha + \frac{p-1}{p} n\beta$

Broadcast

$log(p)(\alpha + n\beta)$

Reduce-scatter

$2log(p)\alpha + log(p)n(\beta + \gamma) + \frac{p-1}{p} n\beta$

Allreduce

$2log(p)\alpha + log(p)n(2\beta + \gamma)$

Allgather

$2log(p)\alpha + log(p)n\beta + \frac{p-1}{p} n\beta$

Recap

Reduce(-to-one)

$$\log(p)(\alpha + n\beta + n\gamma)$$

Scatter

$$\log(p)\alpha + \frac{p-1}{p} n\beta$$

Gather

$$\log(p)\alpha + \frac{p-1}{p} n\beta$$

Broadcast

$$\log(p)(\alpha + n\beta)$$

Reduce-scatter

$$2\log(p)\alpha + \log(p)n(\beta + \gamma) + \frac{p-1}{p} n\beta$$

Allreduce

$$2\log(p)\alpha + \log(p)n(2\beta + \gamma)$$

Allgather

Recap

Reduce(-to-one)

$log(p)(\alpha + n\beta + n\gamma)$

Scatter

$log(p)\alpha + \frac{p-1}{p} n\beta$

Gather

$log(p)\alpha + \frac{p-1}{p} n\beta$

Broadcast

$log(p)(\alpha + n\beta)$

Reduce-scatter

$2log(p)\alpha + log(p)n(\beta + \gamma) + \frac{p-1}{p} n\beta$

Allreduce

$2log(p)\alpha + log(p)n(2\beta + \gamma)$

Allgather

$2log(p)\alpha + log(p)n\beta + \frac{p-1}{p} n\beta$

Summary of MST algorithms

- Small message: Minimum Spanning Tree algorithm
  - Emphasize **low latency**

- **Can we do better**

- Problem of Minimum Spanning Tree Algorithm?
  - It prioritize latency rather than bandwidth
  - Hence: Some links are idle

- Next class: Large message size algorithm