https://hao-ai-lab.github.io/cse234-w25/

CSE 234: Data Systems for Machine Learning
Winter 2025

LLMSys

Optimizations and Parallelization

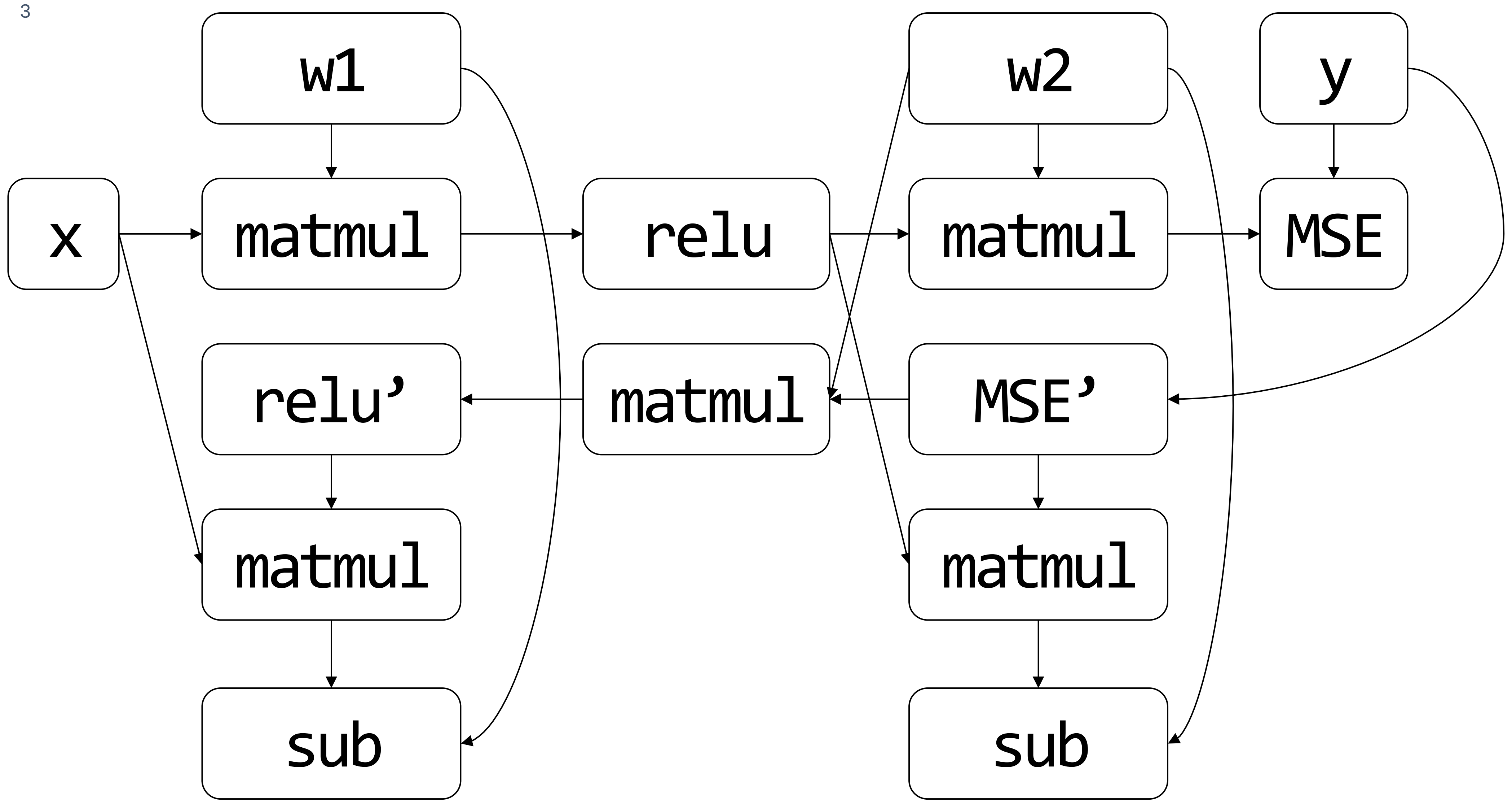
MLSys Basics

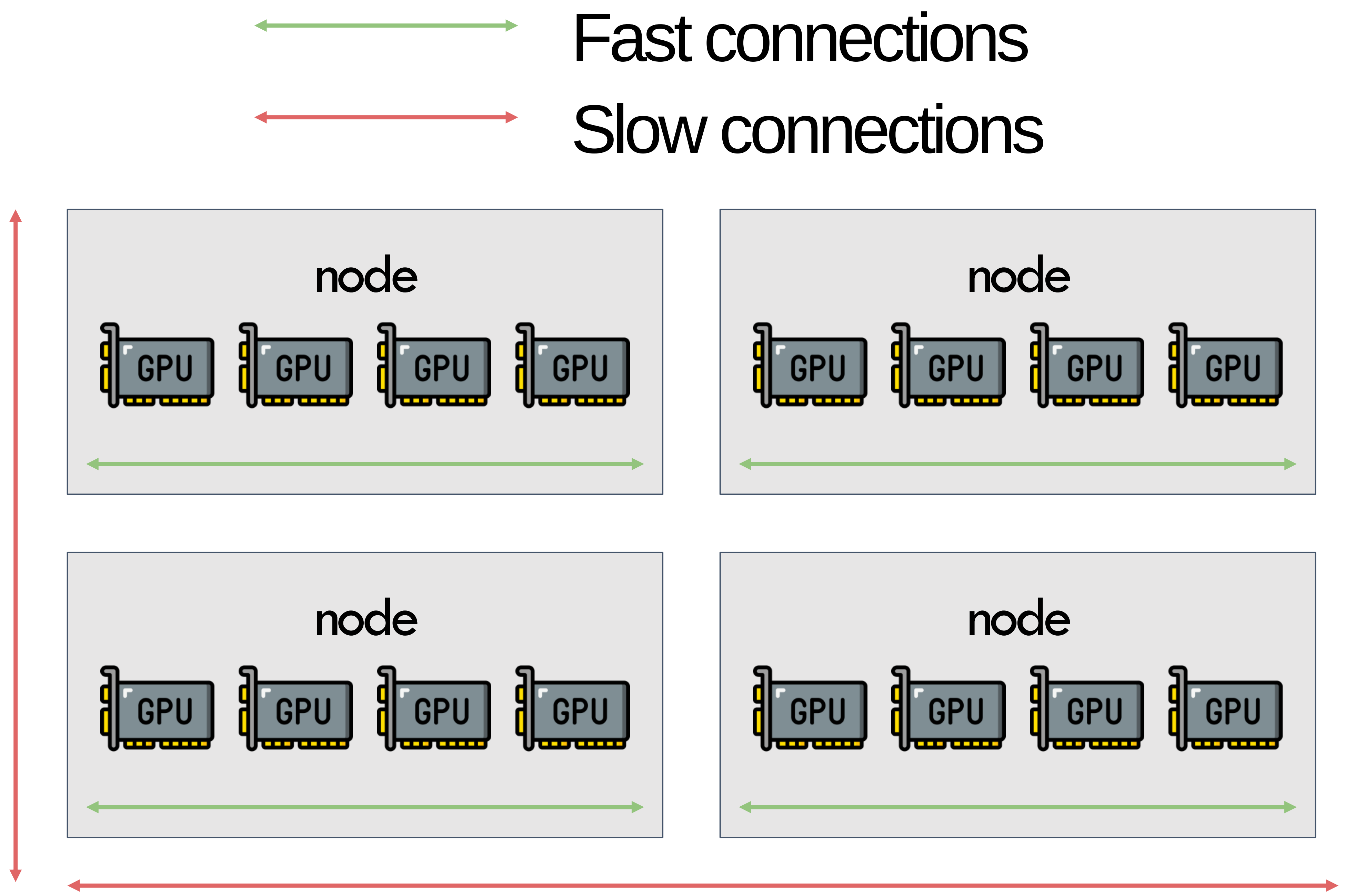
Parallelization

- Why Parallelization: Technology Trend
- **ML Parallelism Overview**
- Collective Communication Review
- Data parallelism
- Model parallelism
  - Inter and intra-op parallelism
- Auto-parallelization

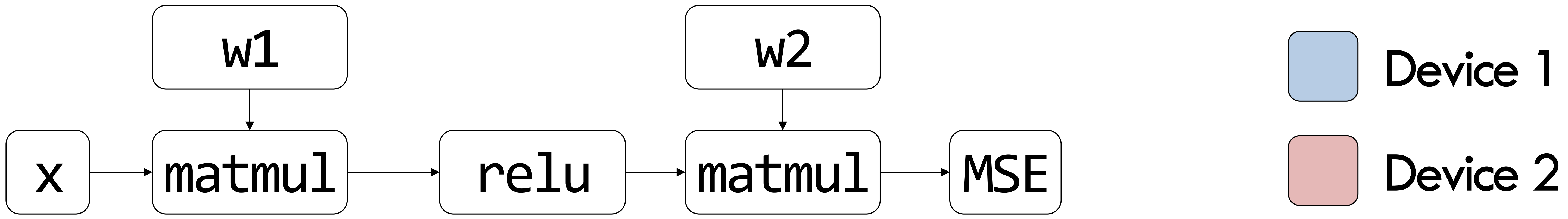
Parallelization = Partitioning Computation Graph on Device Cluster

How to partition the computational graph on the device cluster?

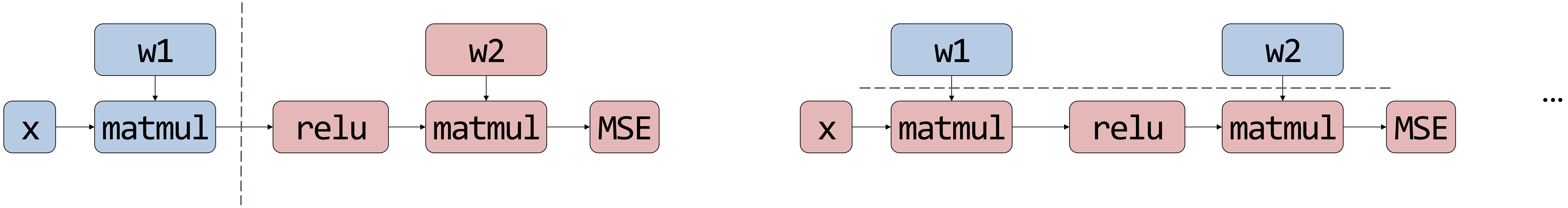




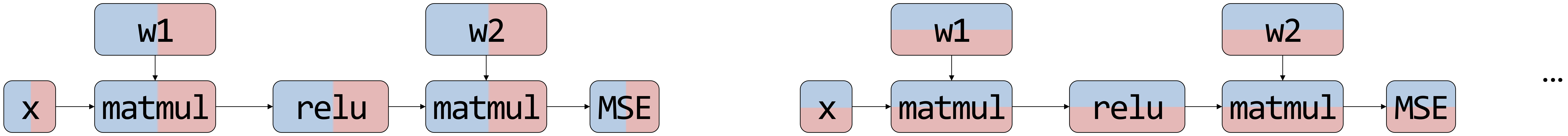
Summary: Inter-op and Intra-op Parallelisms



Inter-op parallelism: Assign different operators to different devices.

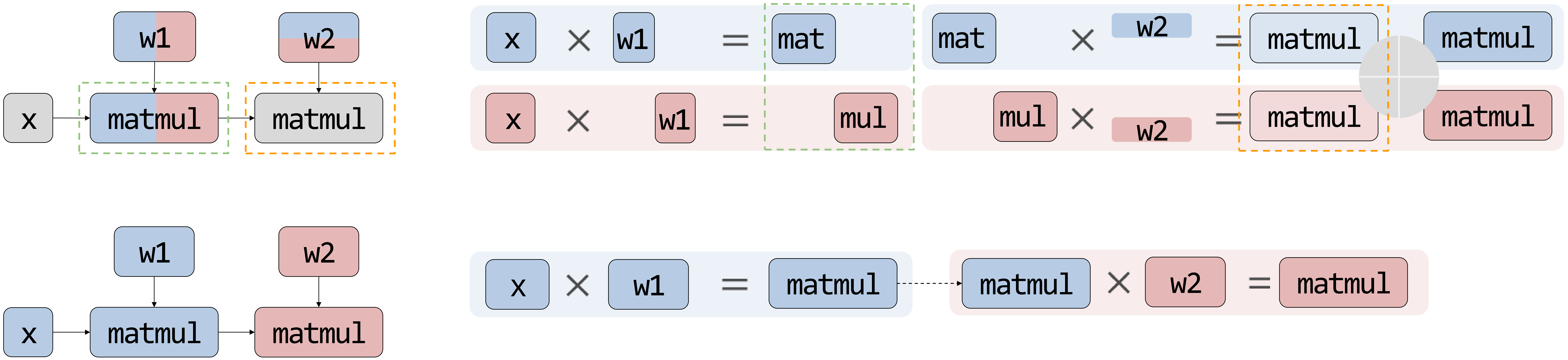


Intra-op parallelism: Assign different regions of a single operator to different devices.

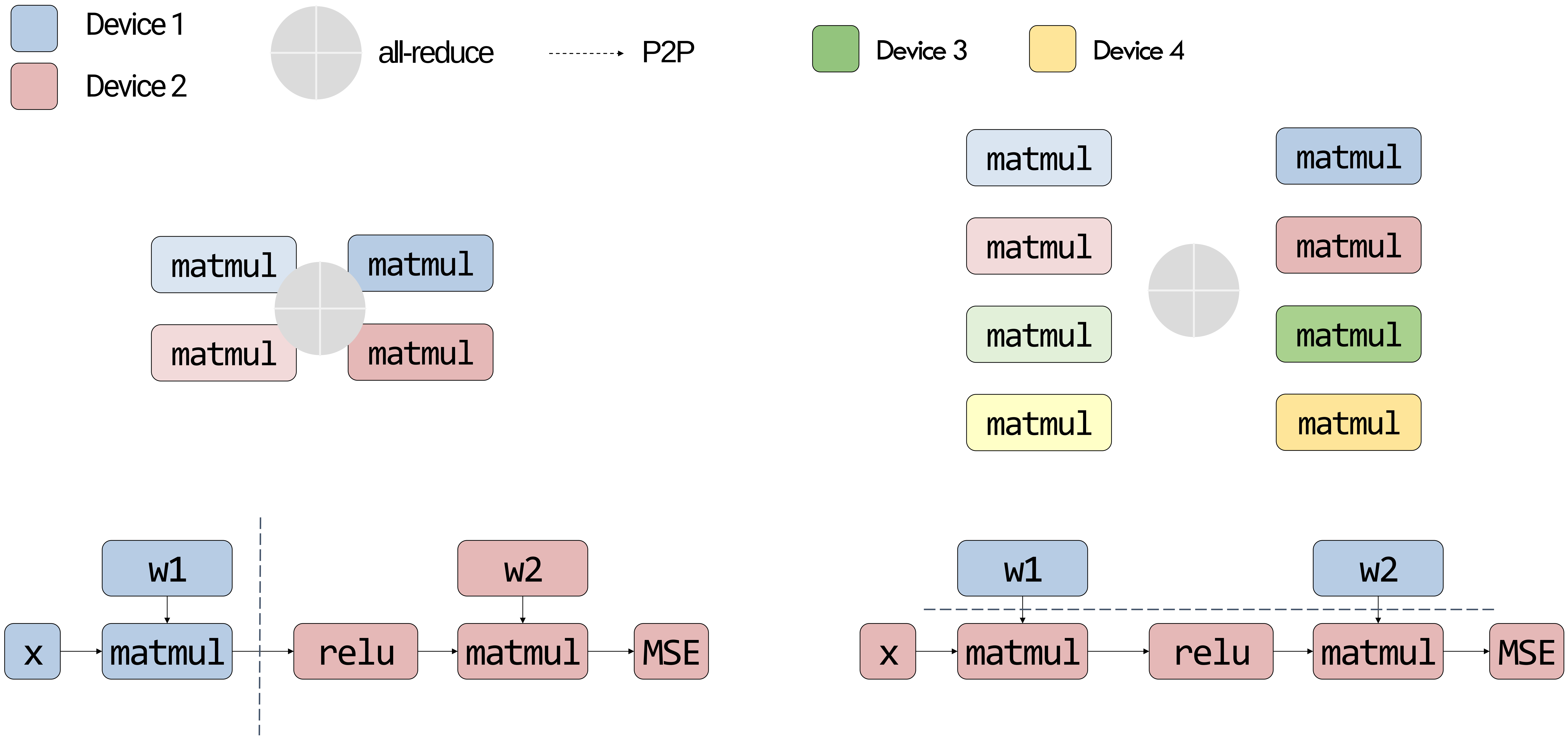


Inside Intra- and Inter-op Parallelism

$$Y = X \cdot W_1 \cdot W_2 = X \cdot \begin{bmatrix} W_1^{d1} & W_1^{d2} \end{bmatrix} \cdot \begin{bmatrix} W_2^{d1} \\ W_2^{d2} \end{bmatrix}$$

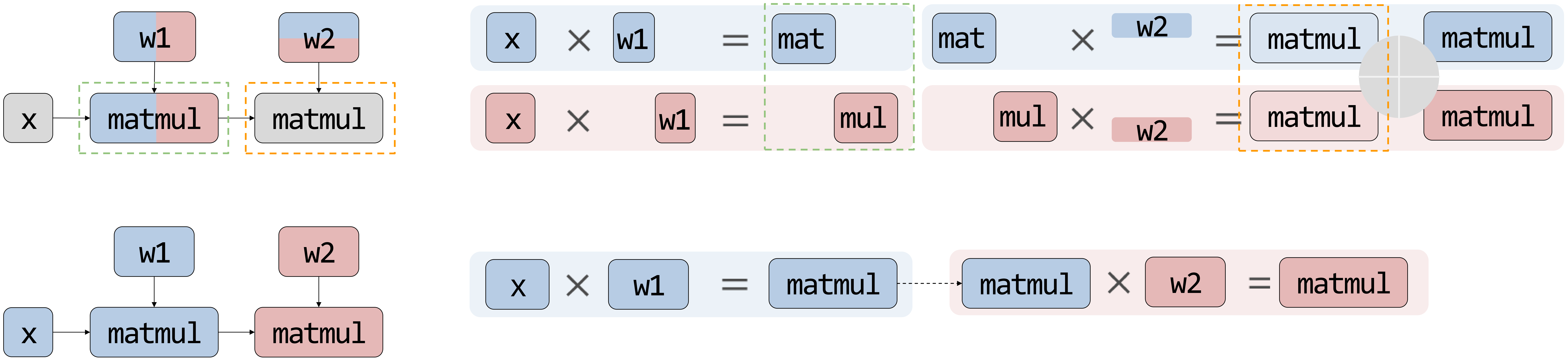


Looking Into the Communication

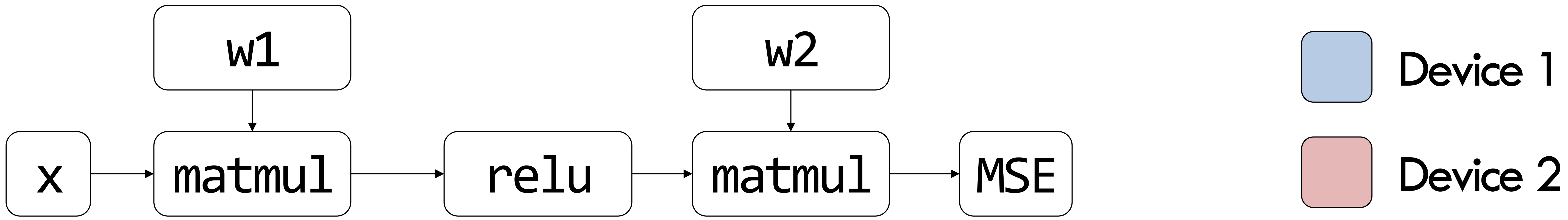


Inside Intra- and Inter-op Parallelism

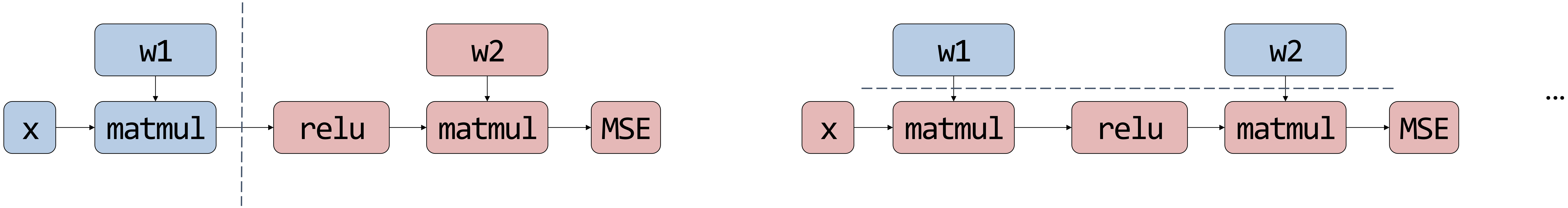
$$Y = X \cdot W_1 \cdot W_2 = X \cdot \begin{bmatrix} W_1^{d1} & W_1^{d2} \end{bmatrix} \cdot \begin{bmatrix} W_2^{d1} \\ W_2^{d2} \end{bmatrix}$$



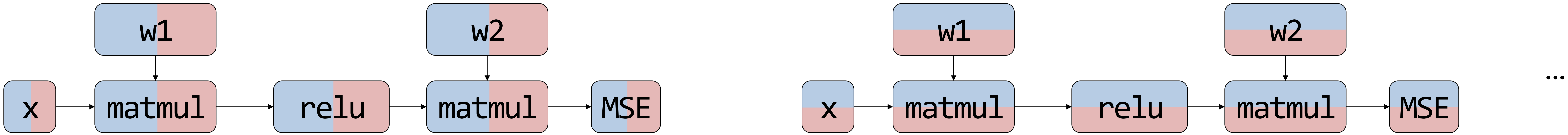
Parallelism: Key Characteristics



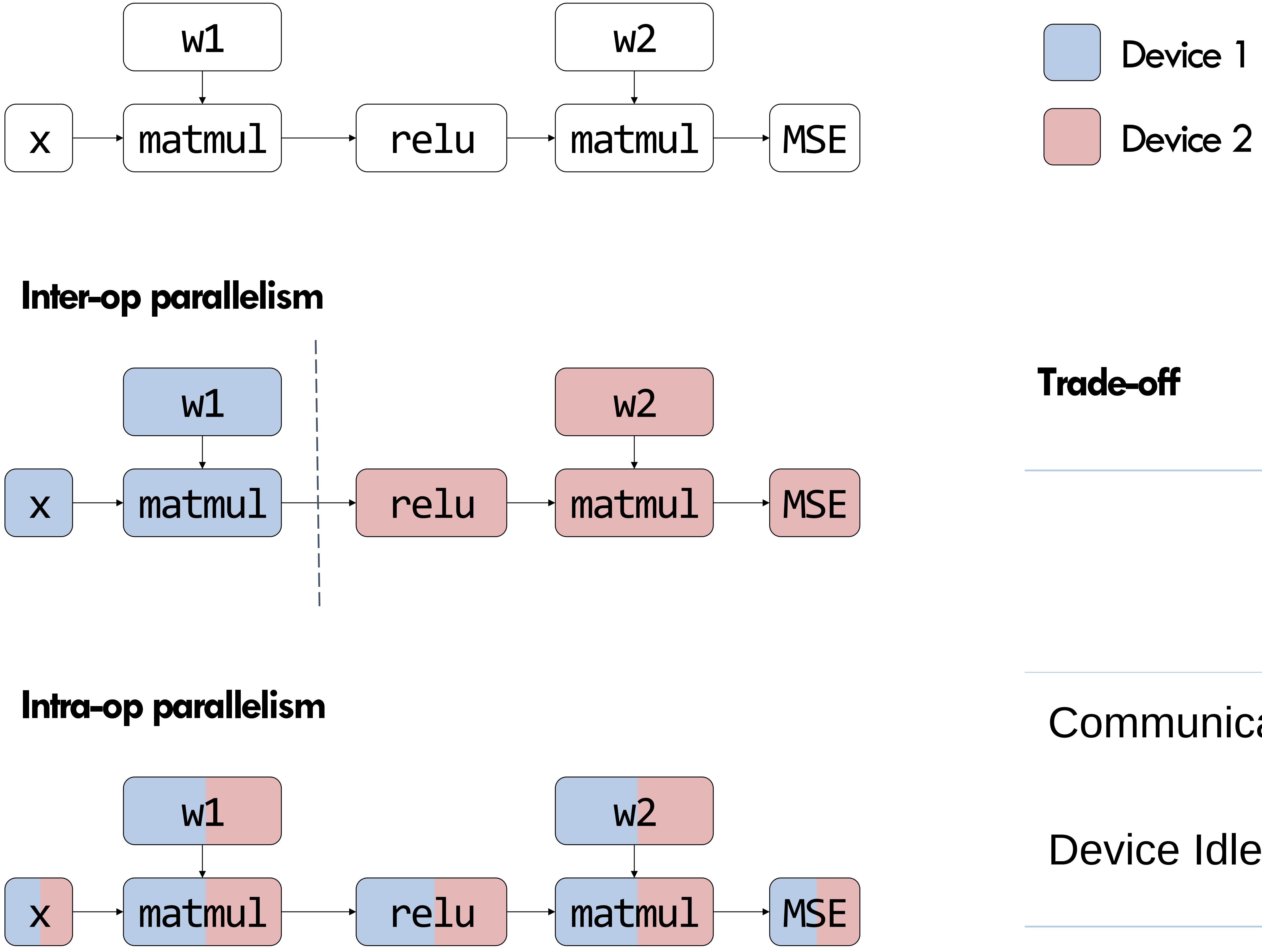
Inter-op parallelism: Requires point-to-point communication but results in device idle



Intra-op parallelism: Devices are busy but requires collective communication



Inter-op and Intra-op Parallelism: Characteristics [Important]



Trade-off

	Inter-operator Parallelism	Intra-operator Parallelism
Communication	Less	More
Device Idle Time	More	Less

Computational View of ML Parallelisms

Classic view

Data parallelism

Model parallelism

New view (this class)

Inter-op parallelism

Intra-op parallelism

Two Views of ML Parallelisms

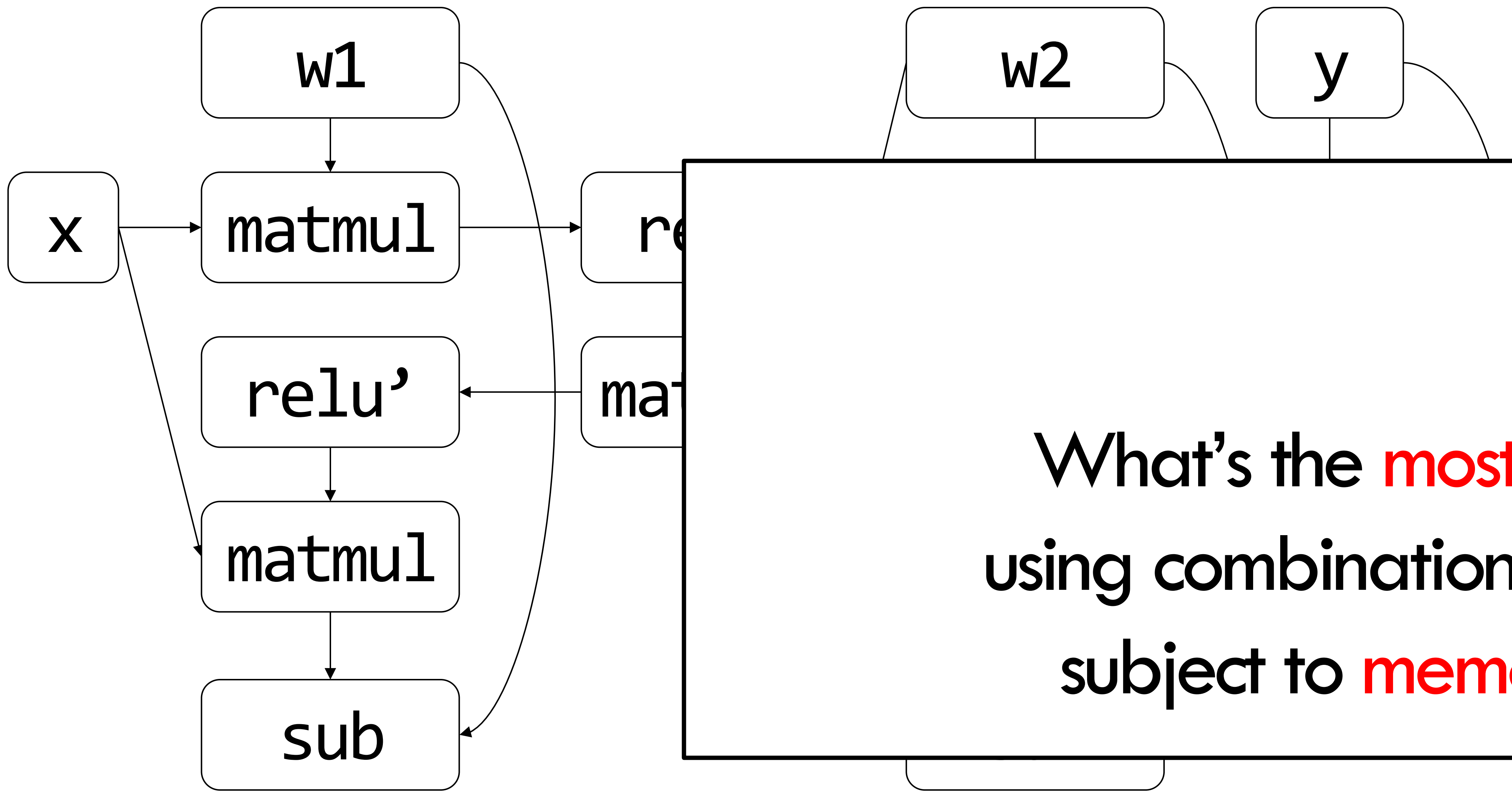
Data and model parallelism

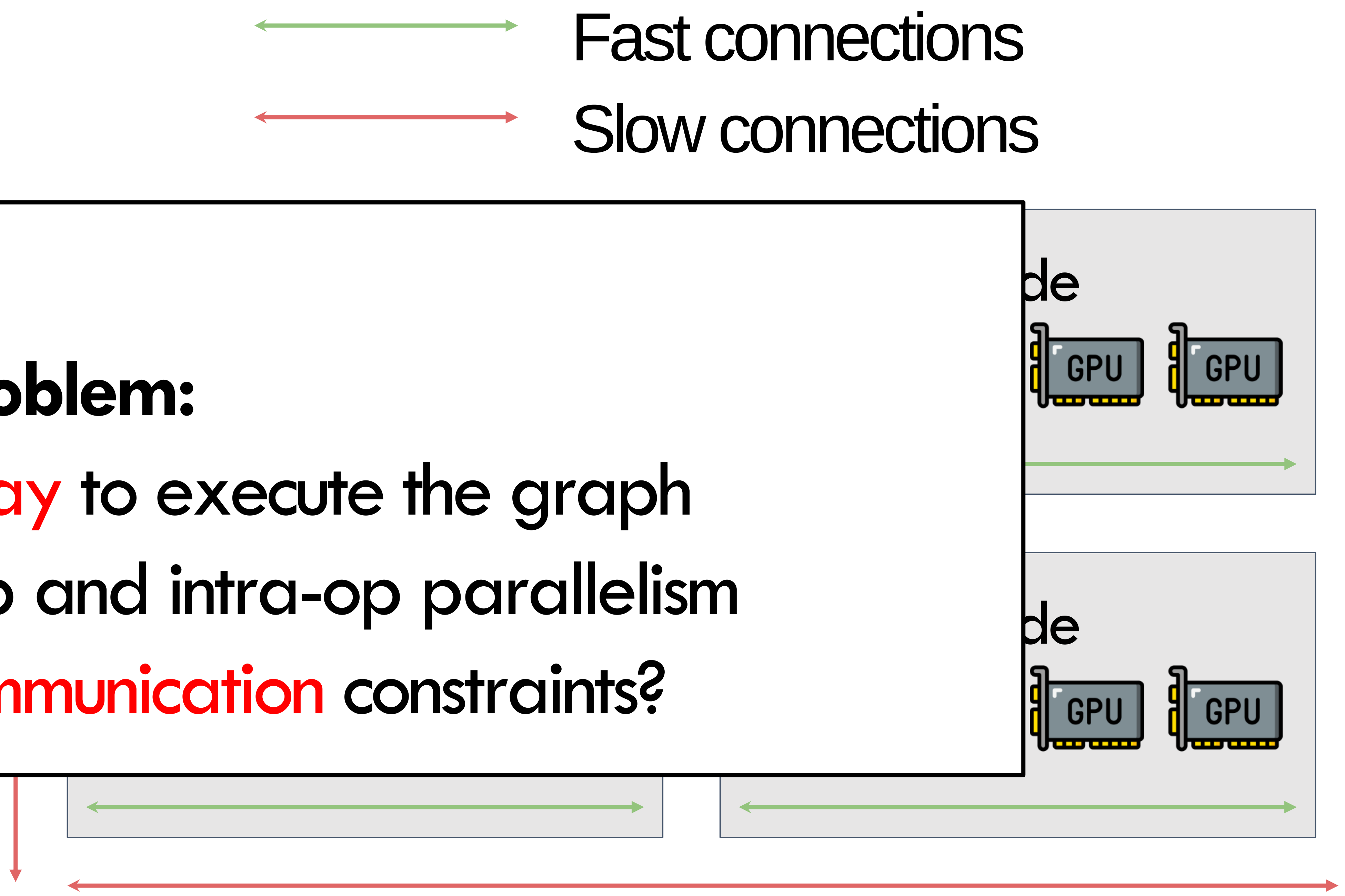
- Two pillars: **data** and **model**.
- ✅ "Data parallelism" is general and precise.
- ❓ "Model parallelism" is vague.
- ❓ The view creates ambiguity for methods that neither partitions data nor the model computation.

New: Inter-op and Intra-op parallelism.

- Two pillars: **computational graph** and **device cluster**
- ✅ This view is based on their computing characteristics.
- ✅ This view facilitates the development of new parallelism methods.

ML Parallelization under New View





Theme problem:

What’s the most efficient way to execute the graph using combinations of inter-op and intra-op parallelism subject to memory and communication constraints?

How to Measure Efficiency of Parallelism?

max AI = #ops / #bytes

A More Holistic (Macro) Measure: MFU)

	H100 SXM
FP64	34 teraFLOPS
FP64 Tensor Core	67 teraFLOPS
FP32	67 teraFLOPS
TF32 Tensor Core*	989 teraFLOPS
BFLOAT16 Tensor Core*	1,979 teraFLOPS
FP16 Tensor Core*	1,979 teraFLOPS
FP8 Tensor Core*	3,958 teraFLOPS
INT8 Tensor Core*	3,958 TOPS

ML program's total flops

$$\boxed{\text{MFU}} = \boxed{\#\text{FLOPs}} \,/\, \boxed{t} \,/\, \boxed{\text{peak FLOPS}}$$

model flops utilization

time to finish the program

Potential Factors Compromising MFU

MFU= #FLOPs / t / peak FLOPS

- Op types in the computational graph (ml model type)
  - What are MFU-friendly op and MFU-unfriendly op?
- Optimization (which we have covered)
- Precision, core, and GPU type
- Communication over network
  - How to reduce/hide communication?

Potential Factors Lowing MFU

MFU= #FLOPs / t / peak FLOPS

- ✅ Op types, op shape (ml model type)
  - What are MFU-friendly op and MFU-unfriendly op?
- ✅ Optimization (which we have covered)
- ✅ Precision, core, and GPU type
- ❓ Communication over network
  - How to reduce/hide communication?

Q: Estimate MFU of your transformers in PA1/PA2?

- Step 1: estimate the total FLOPs (still remember matmul flops?)
- Step 2: benchmark the time t
- Step 3: check GPU spec, type of cores, and their peak FLOPS
- Step 4: calculate the MFU

- This will appear as an assignment in PA3

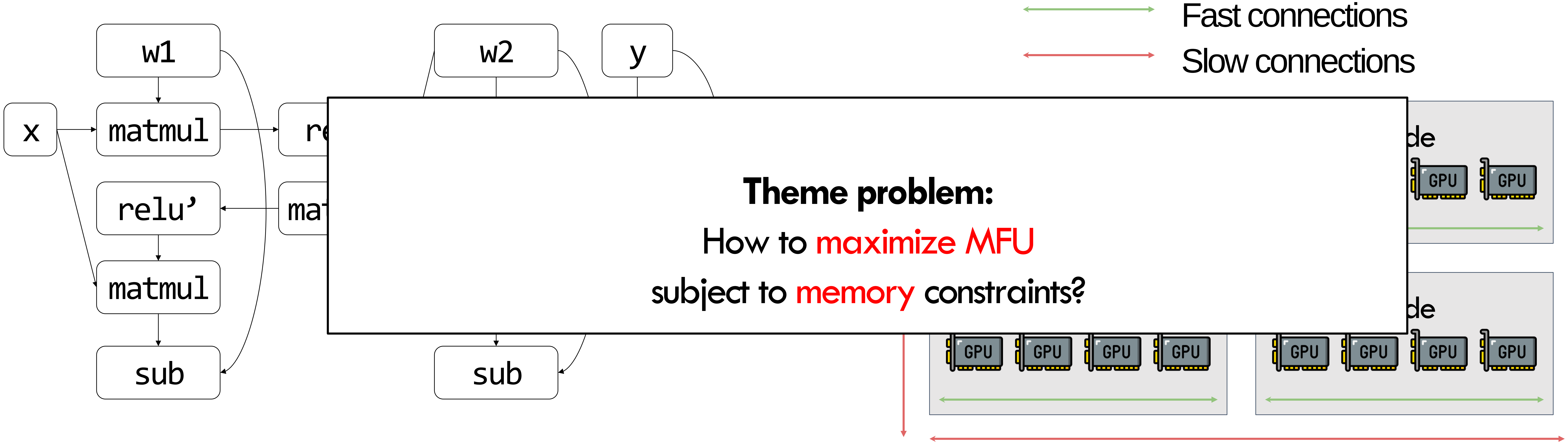
MFU is becoming a metric highly indexed in LLM Industry

- V100: 40 – 50% MFU
  - V100: 112 TFLOPS
- A100: 40%
  - After flash attention: 60%
  - A100: 312 TFLOPS
- H100: 30 – 50% depending on model size
  - H100: 990 TFLOPs
- B100: where will it be?

MFU vs. HFU (Hardware Flops Utilization)

- Hardware FU vs. Model FU
- In what case HFU != MFU?

Simplify the Problem



Theme problem:

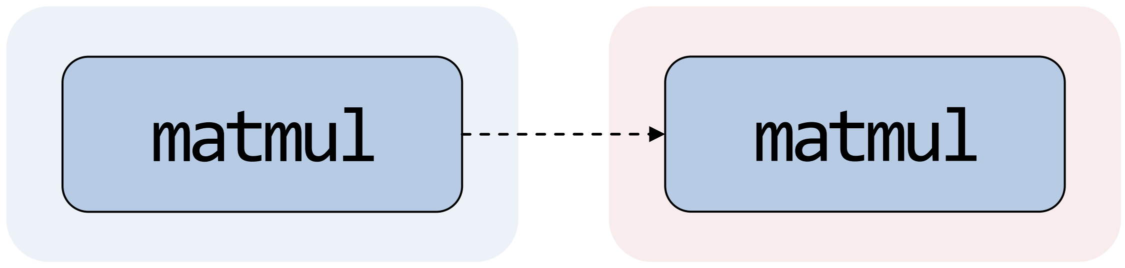
How to maximize MFU

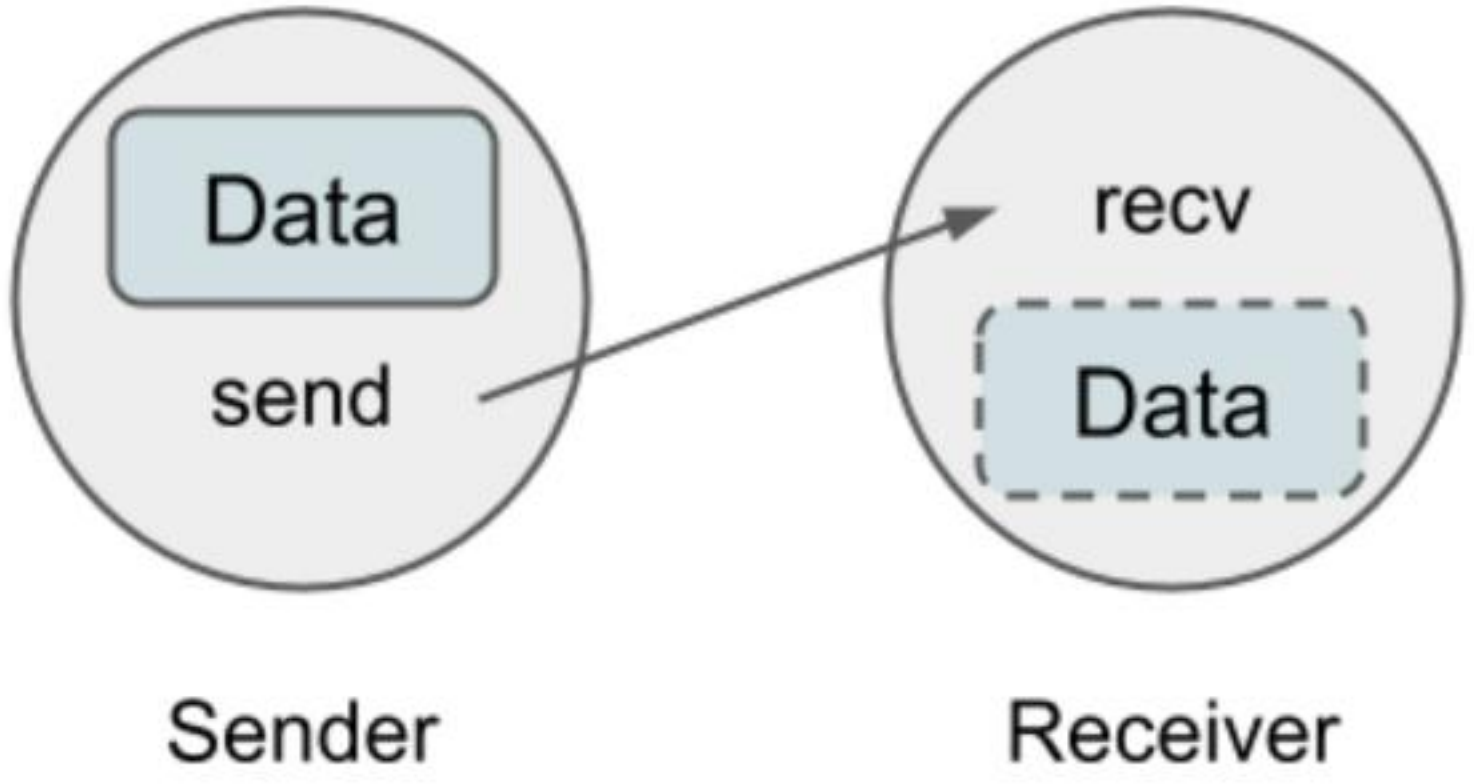
subject to memory constraints?

Parallelization

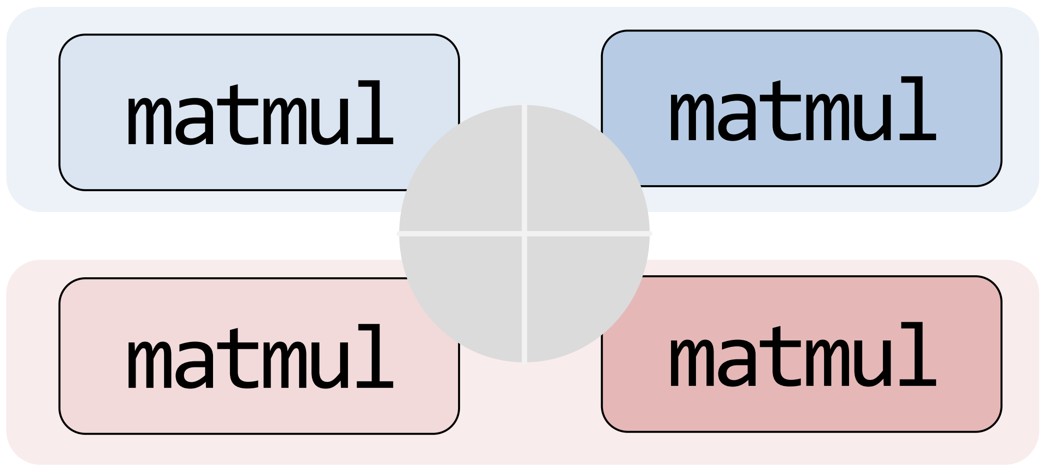
- Why Parallelization: Technology Trend
- ML Parallelism Overview
- **Collective Communication Review**
- Data parallelism
- Model parallelism
  - Inter and intra-op parallelism
- Auto-parallelization

Terminologies: Point-to-point Communication





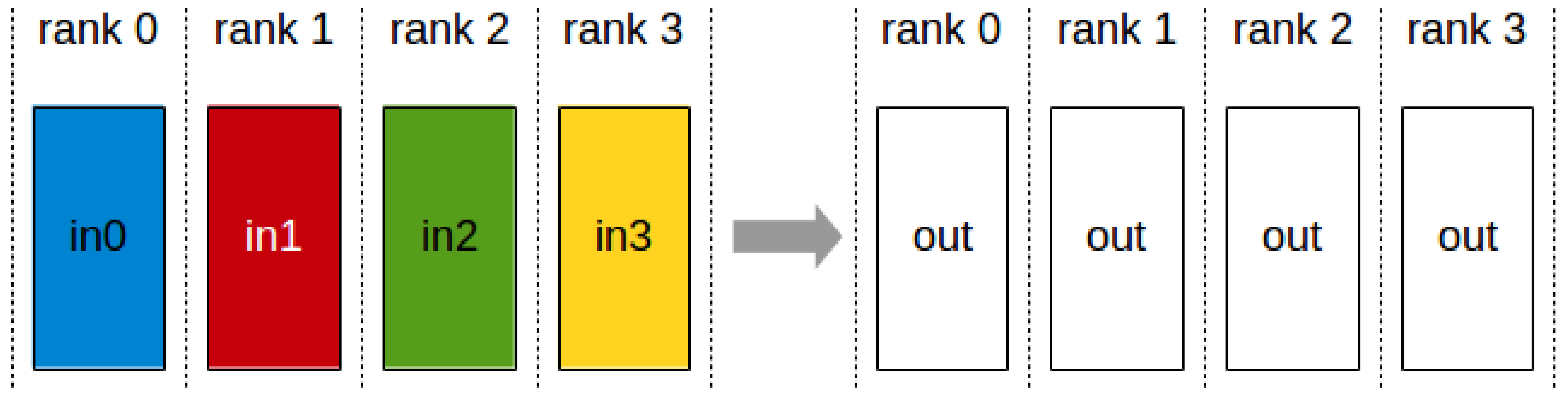
Terminologies: Collective Communication




```
ddp_model = DDP(Model(), device_ids=[rank])
for batch in data_loader:
    loss = train_step(ddp_model, batch)
```

Implicit allreduce here

all-reduce



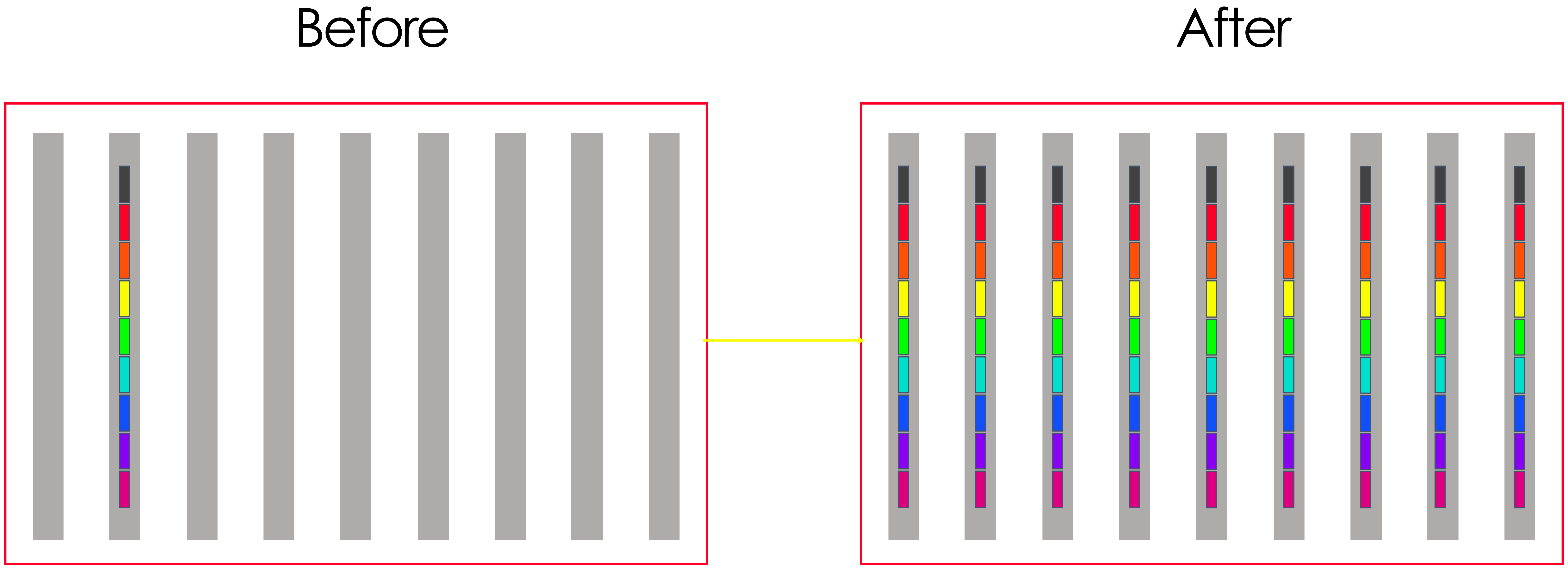
out[i] = sum(inX[i])

Figure from NCCL documentation

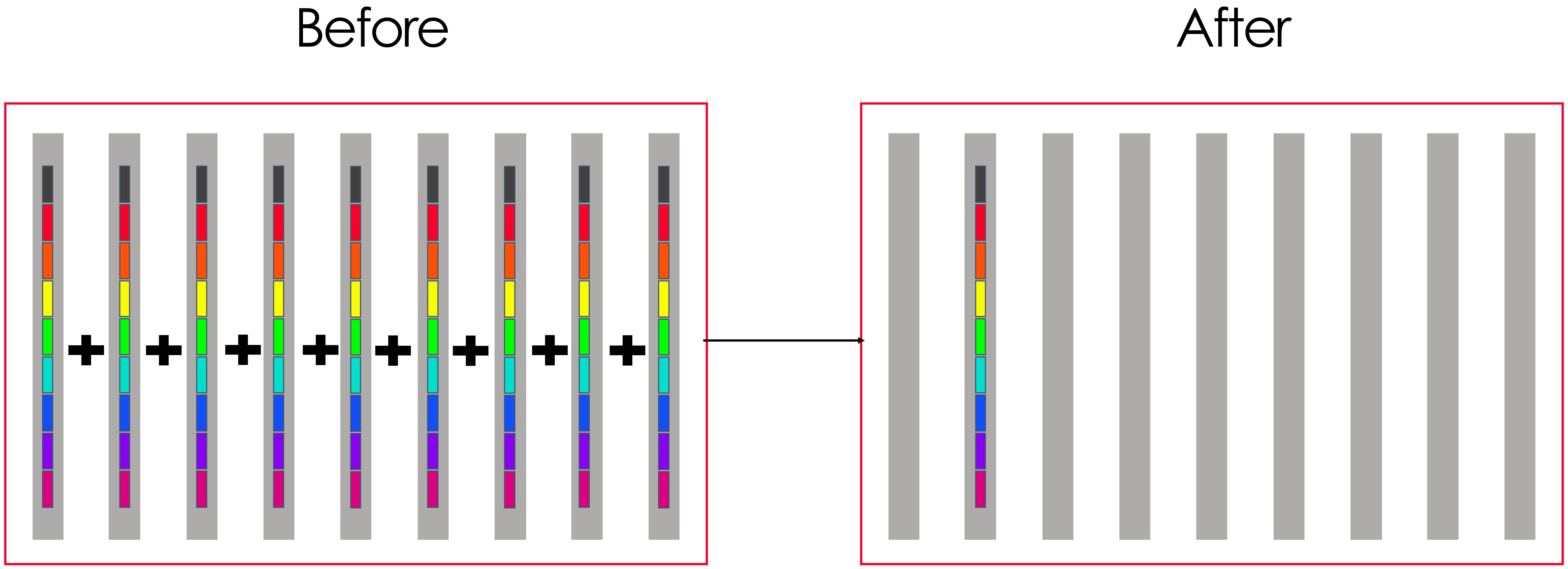
Collective Communications

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

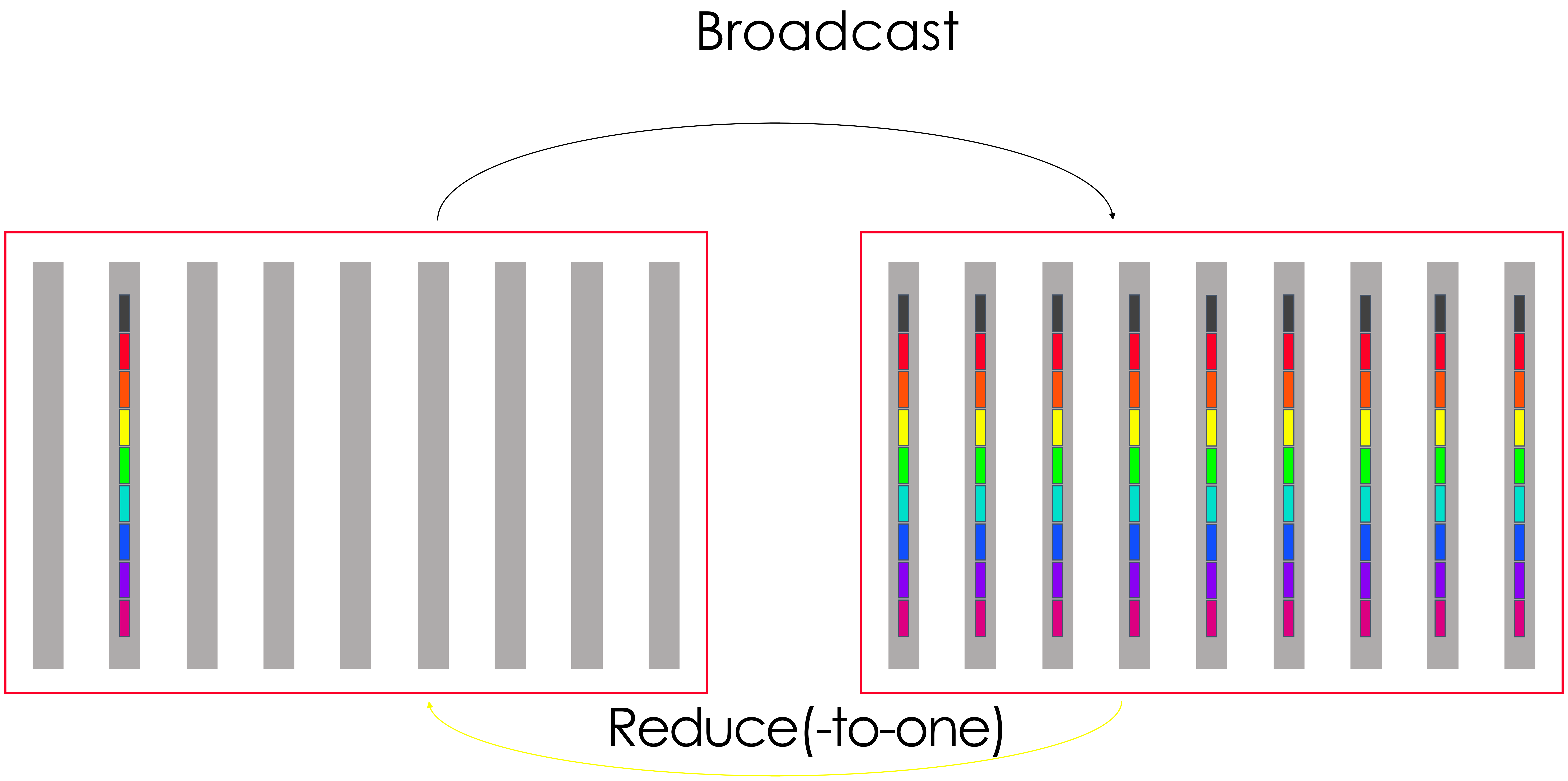
Broadcast



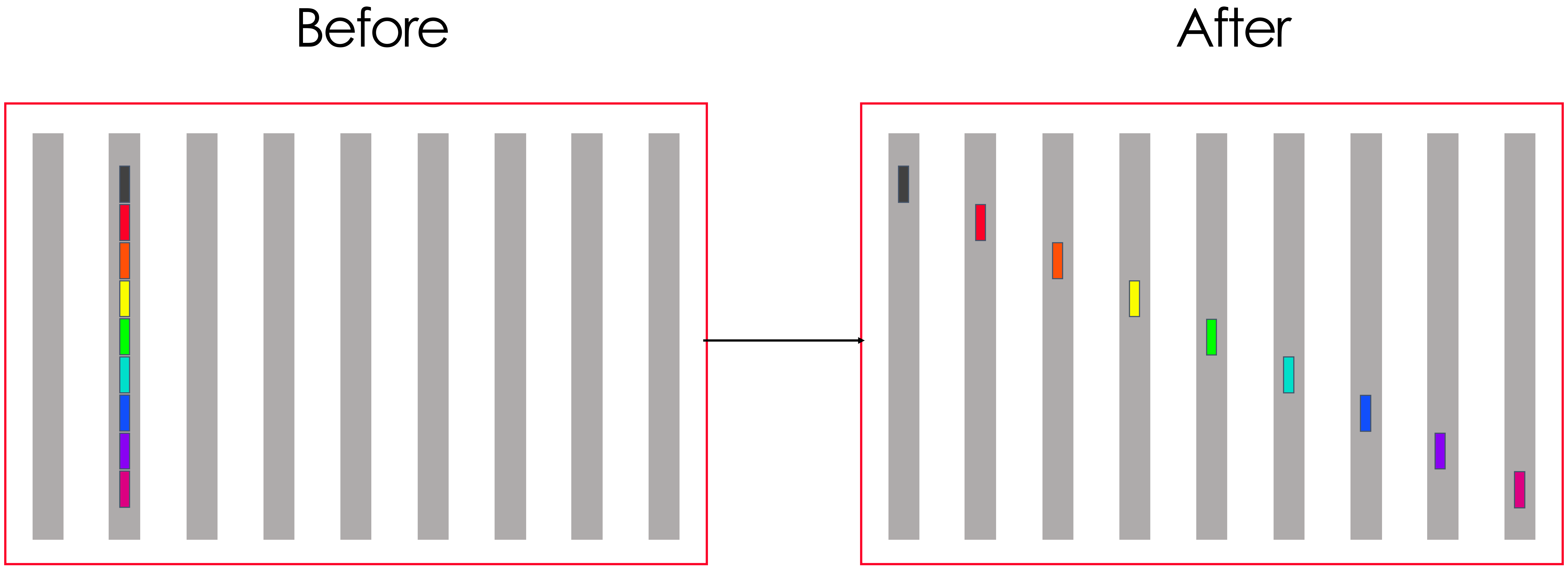
Reduce(-to-one)



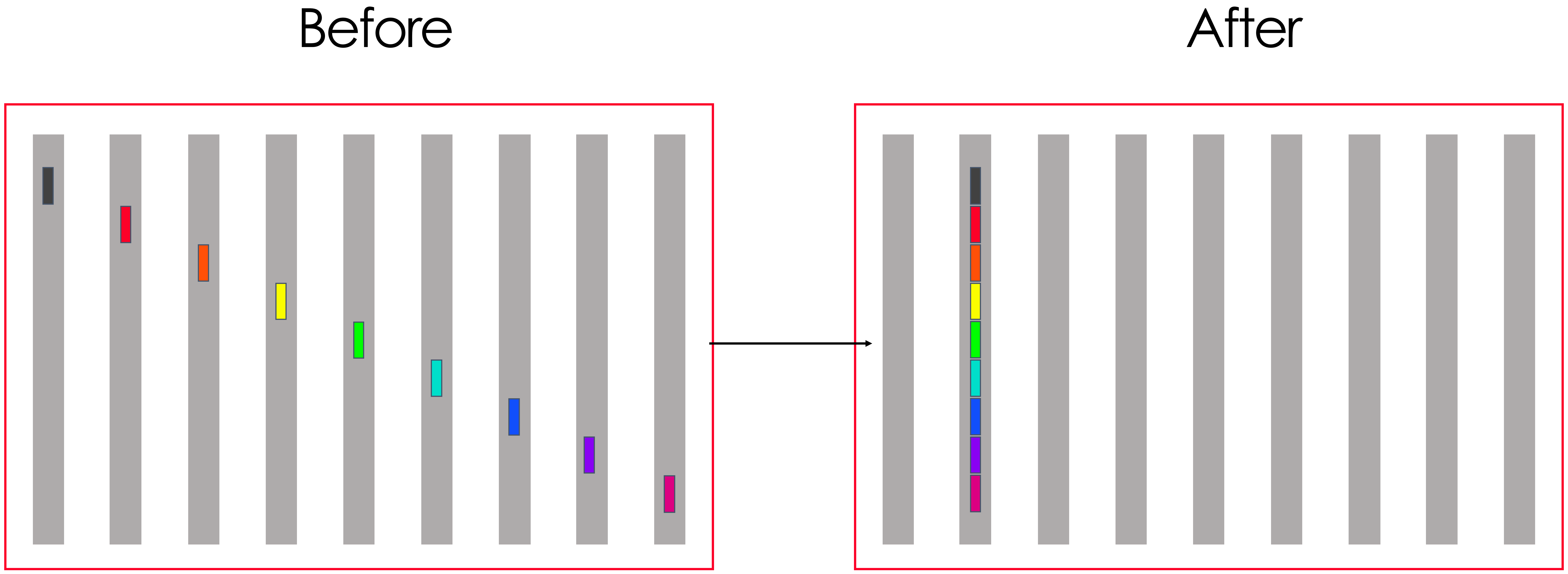
Broadcast/Reduce(-to-one)



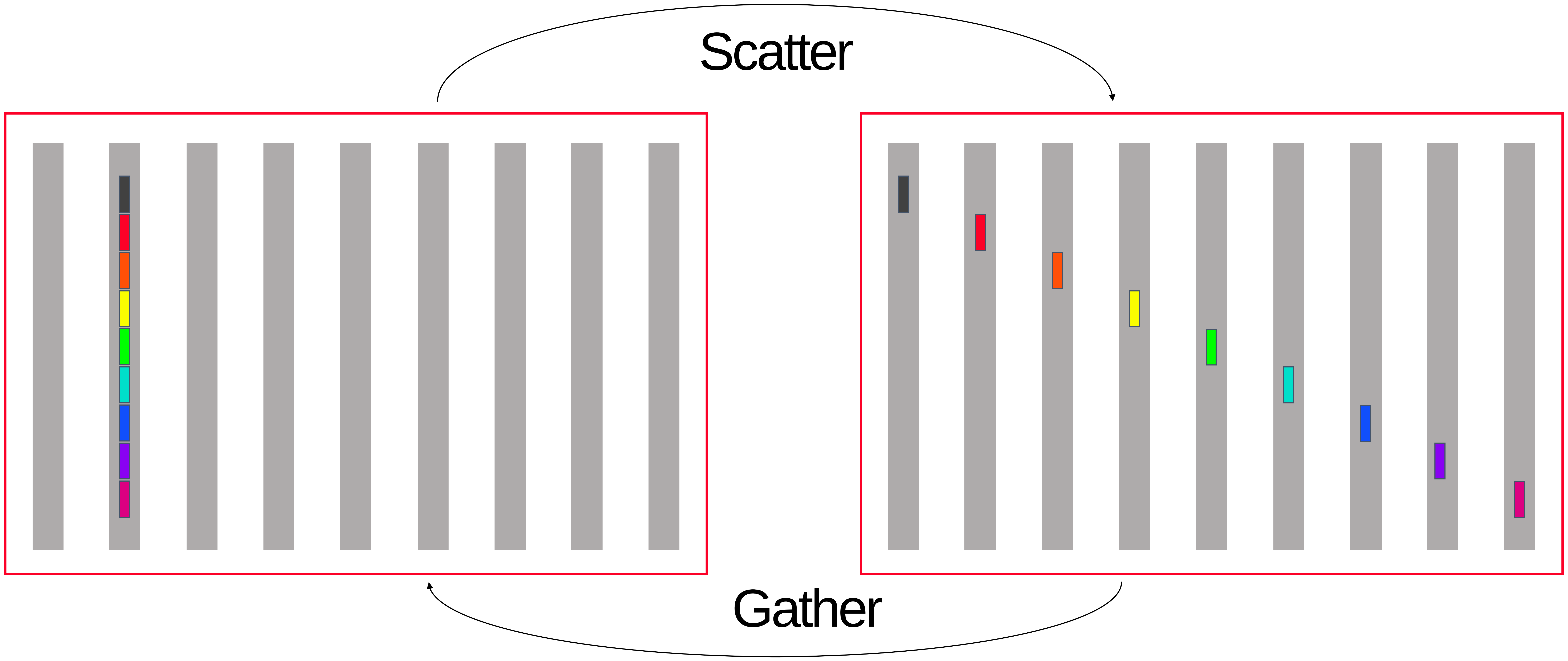
Scatter



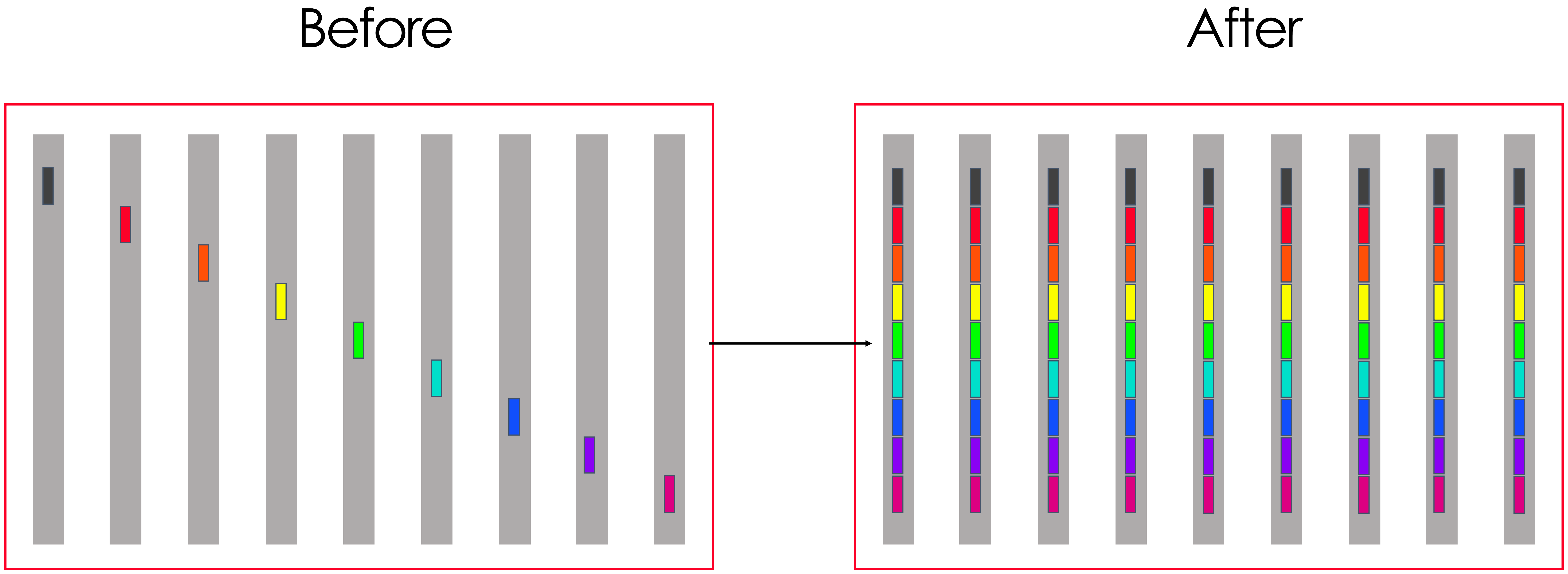
Gather



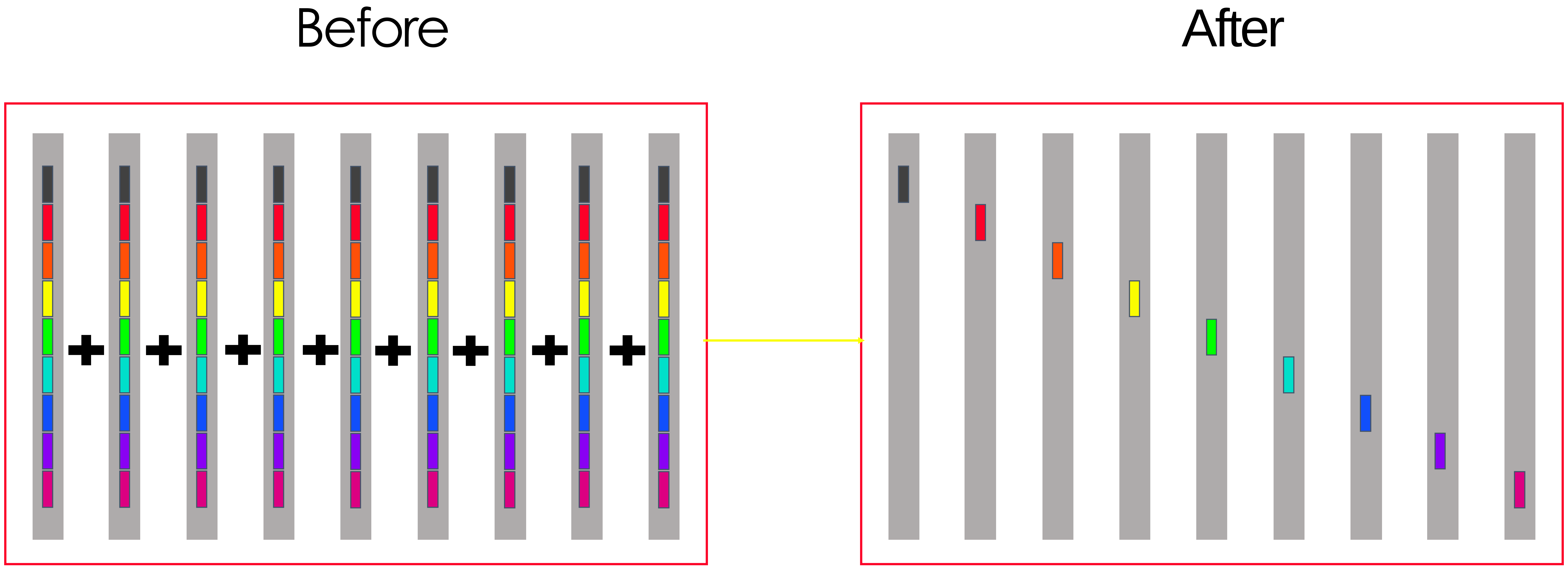
Scatter/Gather



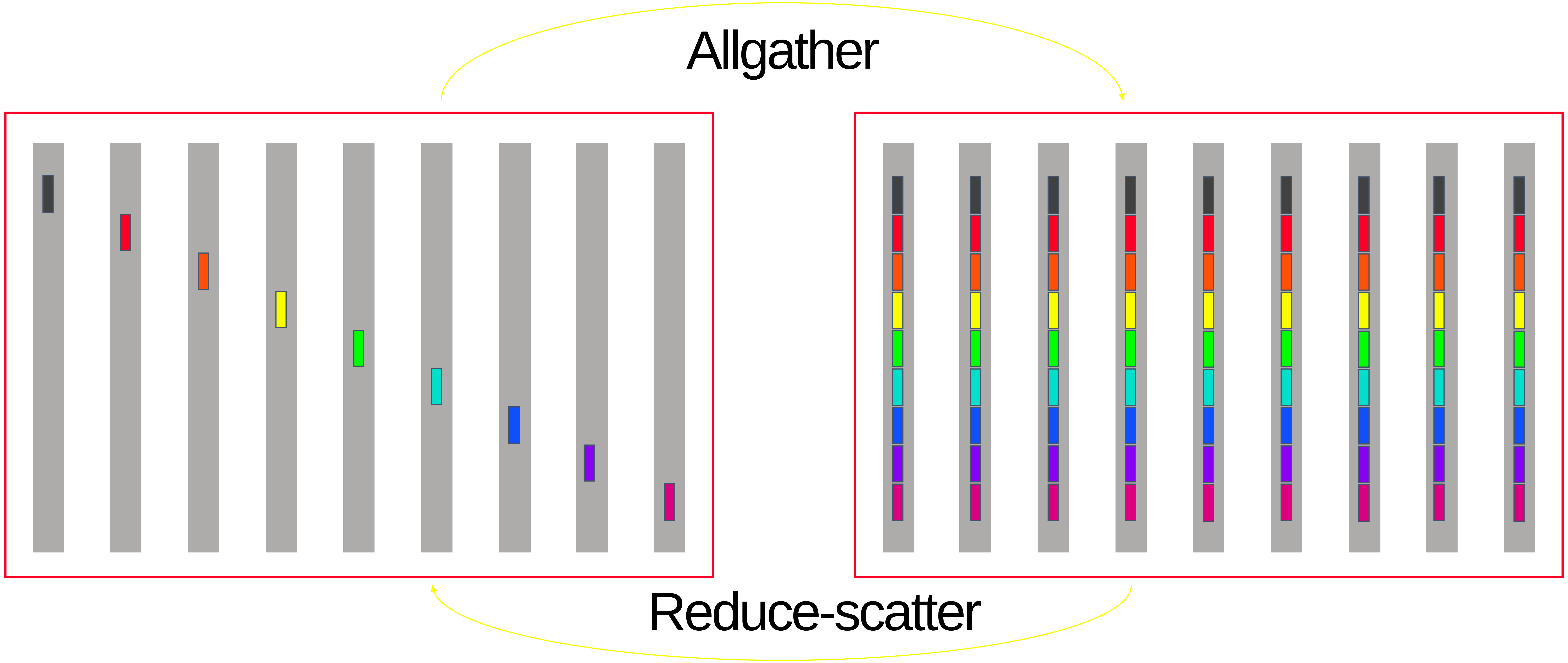
Allgather



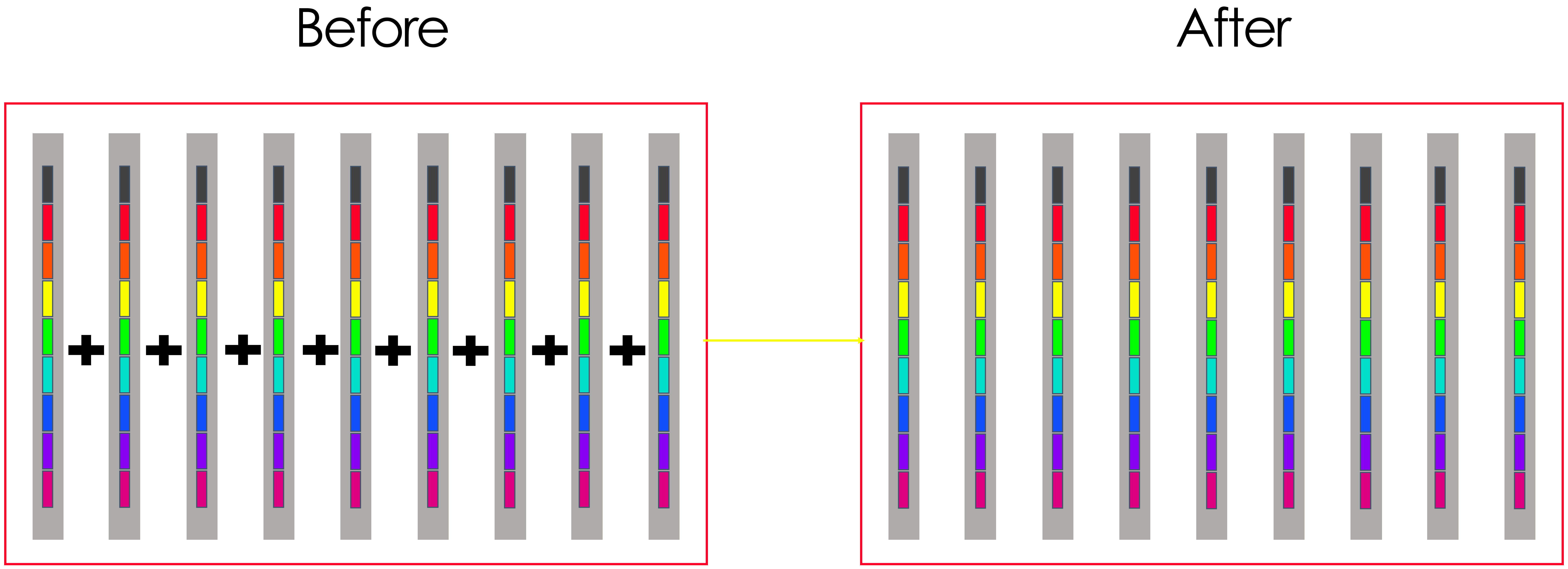
Reduce-scatter



Allgather/Reduce-scatter



Allreduce



Some Facts

- Collective is much more expensive than P2P
  - Collective can be assembled using many P2P
- Collective is highly optimized in the past 20 years
  - Look out for “X”CCL libraries
    - NCCL, MCCL, OneCCL
- Collective is not fault-tolerant

Communication Model: $\alpha\beta$ model

Communication Model: $\alpha + n\,\beta, \beta = \frac{1}{B}$

- Small Message size ($n \to 0$): α dominates, emphasize latency
- Large Message Size ($n \to +\infty$): $n\,\beta$ dominate, emphasize bandwidth utilization

Two Family of Mainstream Algorithms/Implementations

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

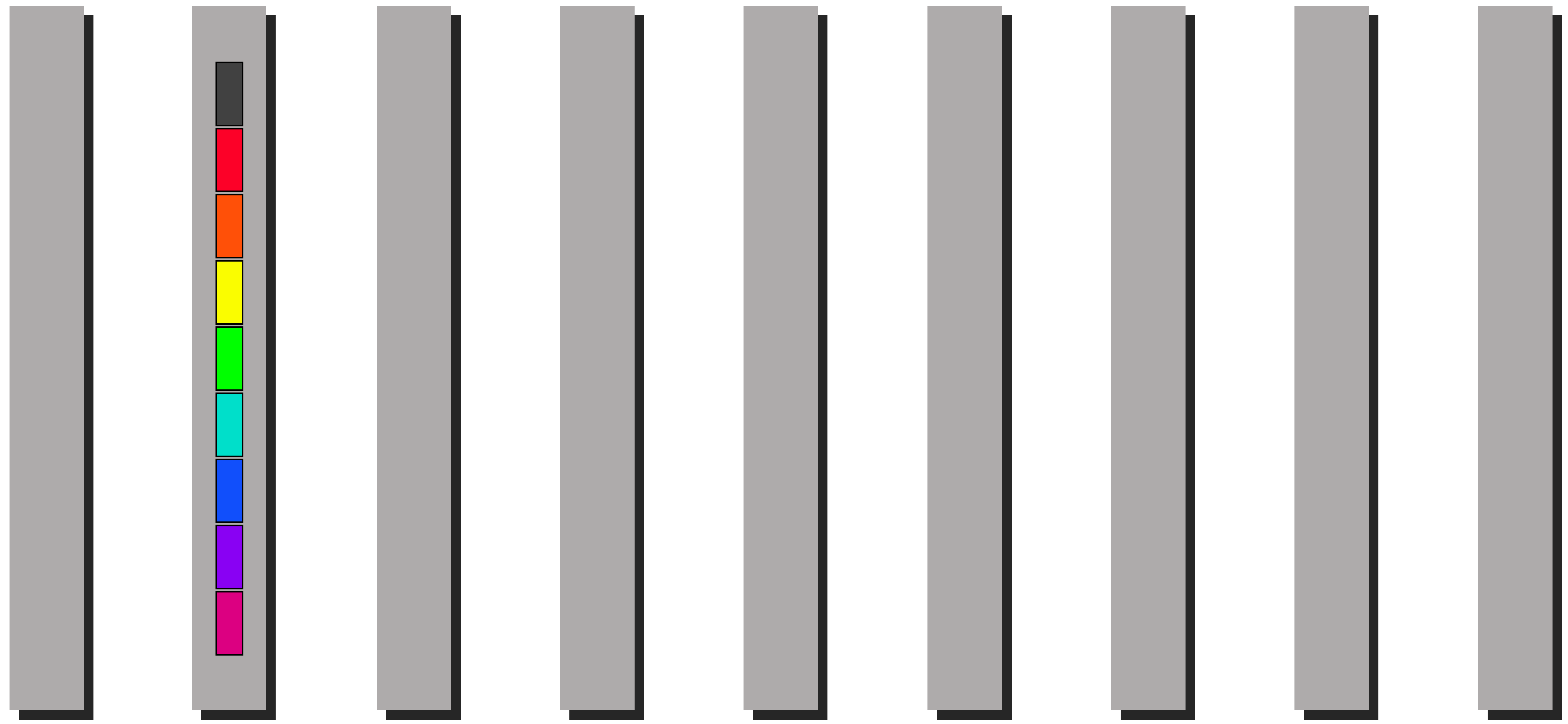
- Large Message: Ring algorithm
  - Emphasize **bandwidth utilization**

- There are 50+ different algorithms developed in the past 50 years by a community called "High-performance computing"
  - 2023 Turing award

General principles: Low Latency

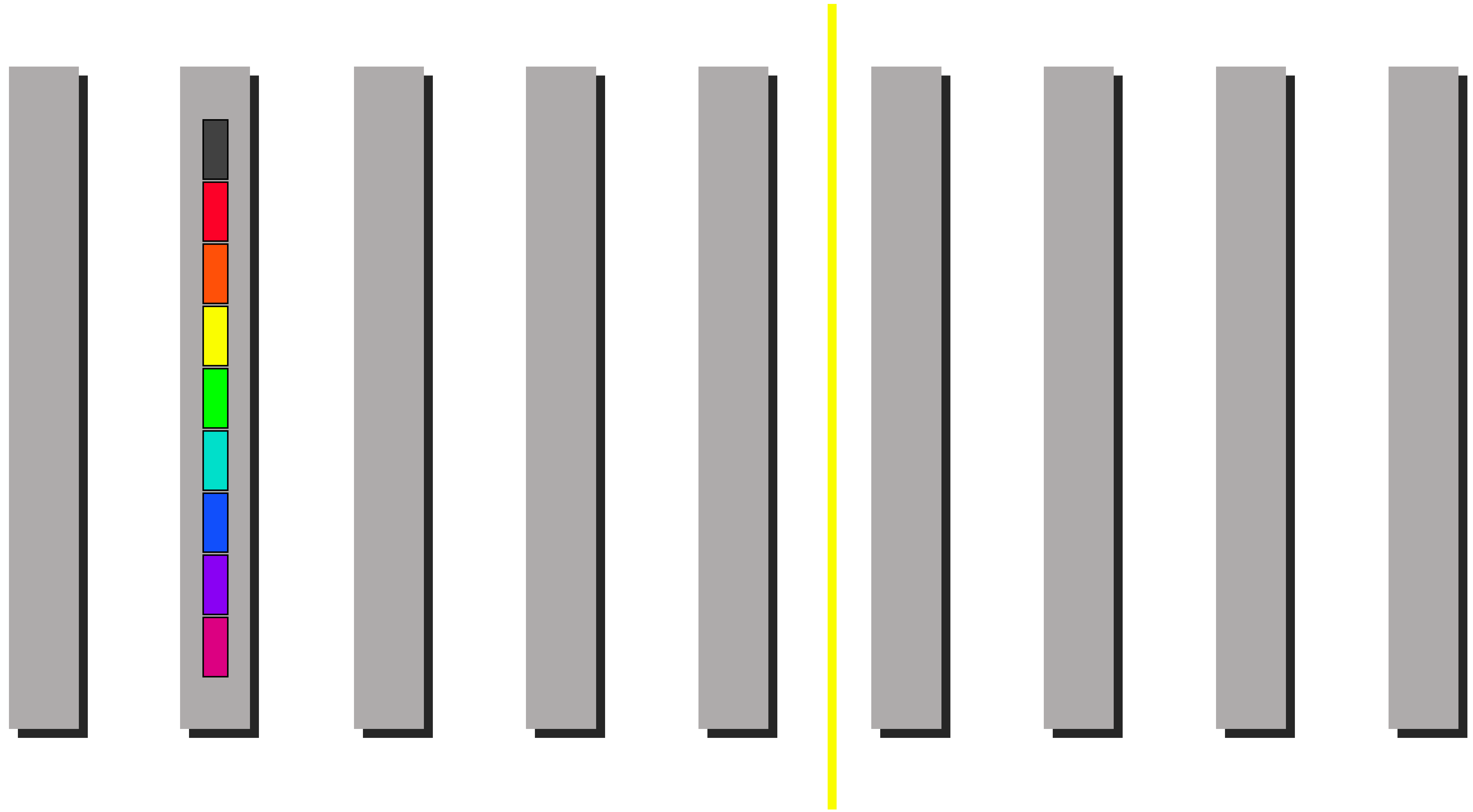
- Minimize the number of rounds needed for communication
- Minimal-spanning tree algorithm

General principles: Low-latency



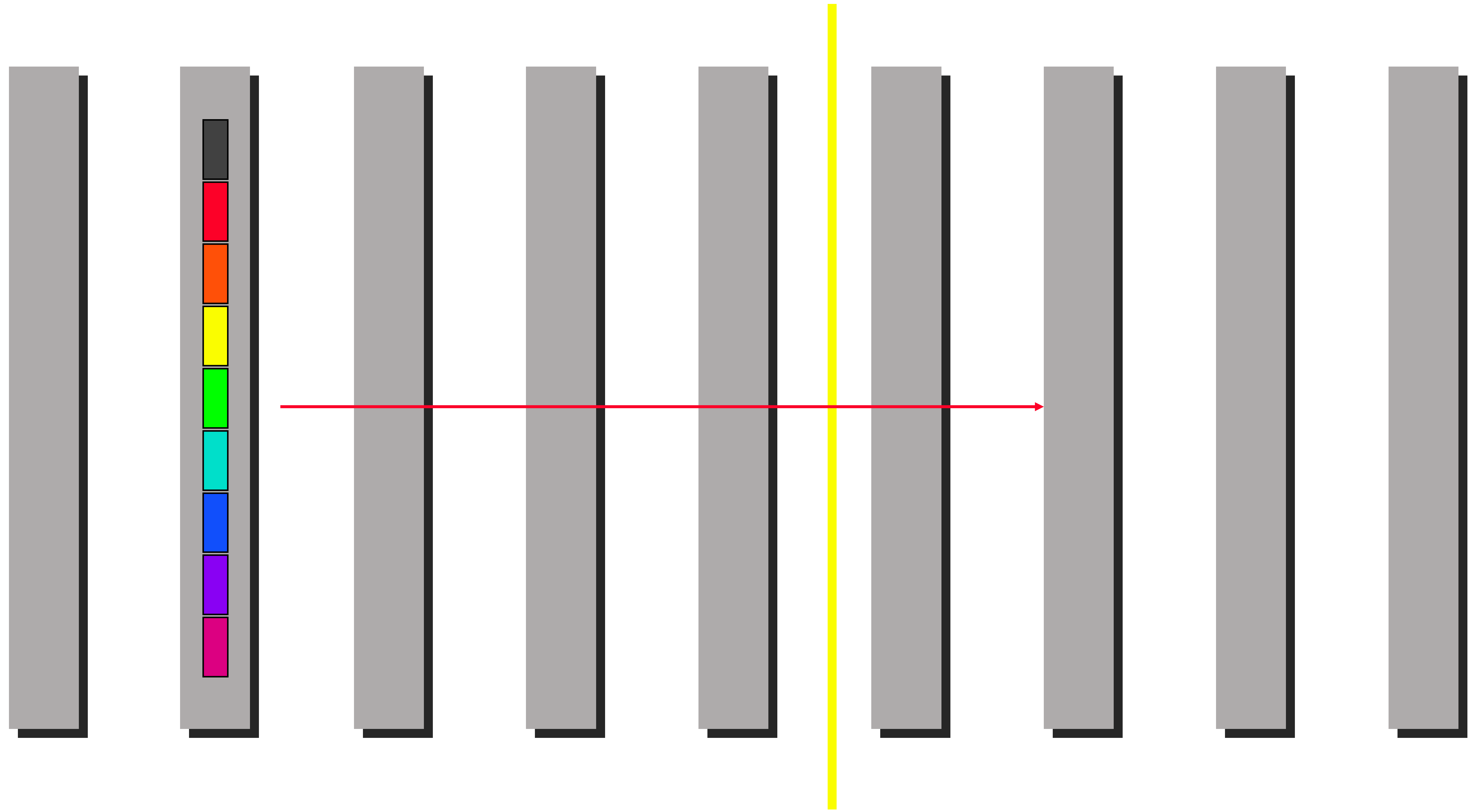
- message starts on one processor

General principles



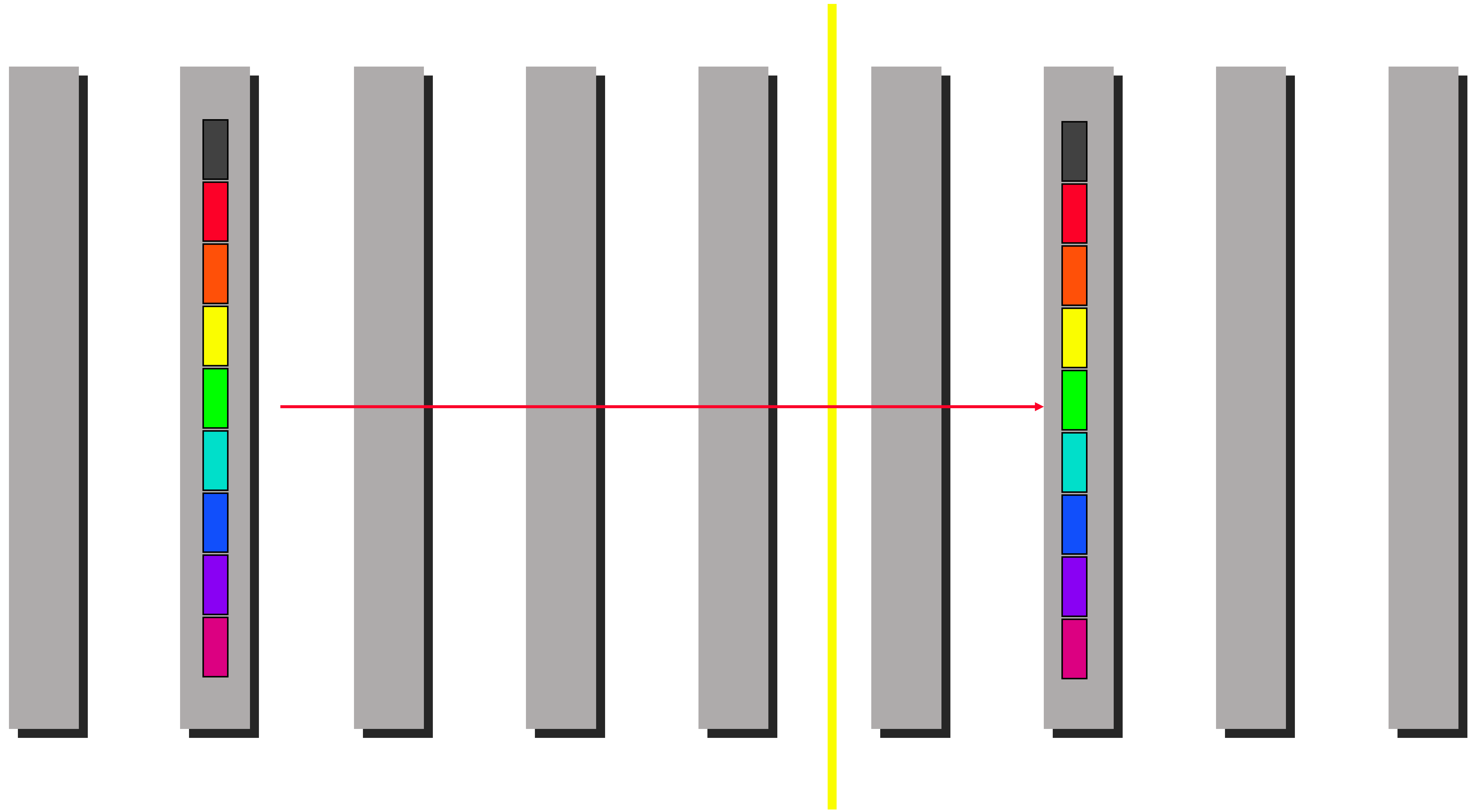
- divide logical linear array in half

General principles



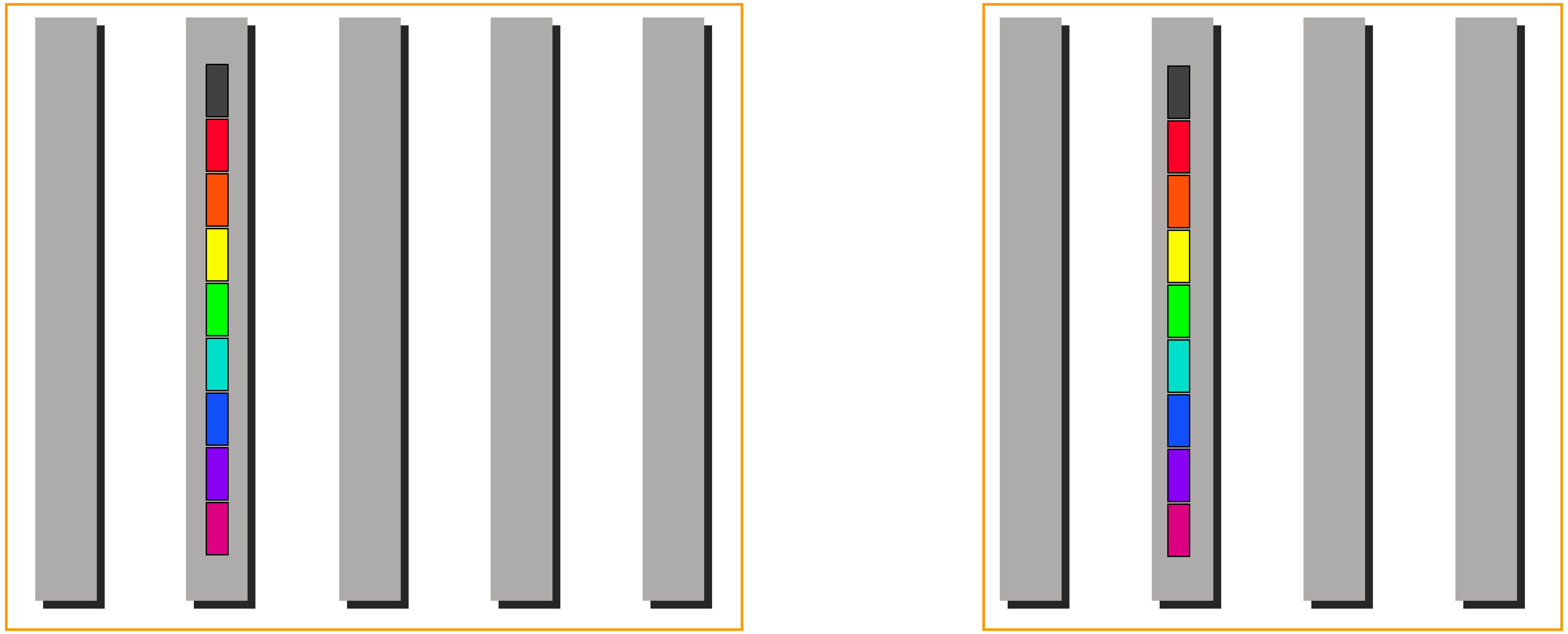
- send message to the half of the network that does not contain the current node (root) that holds the message

General principles



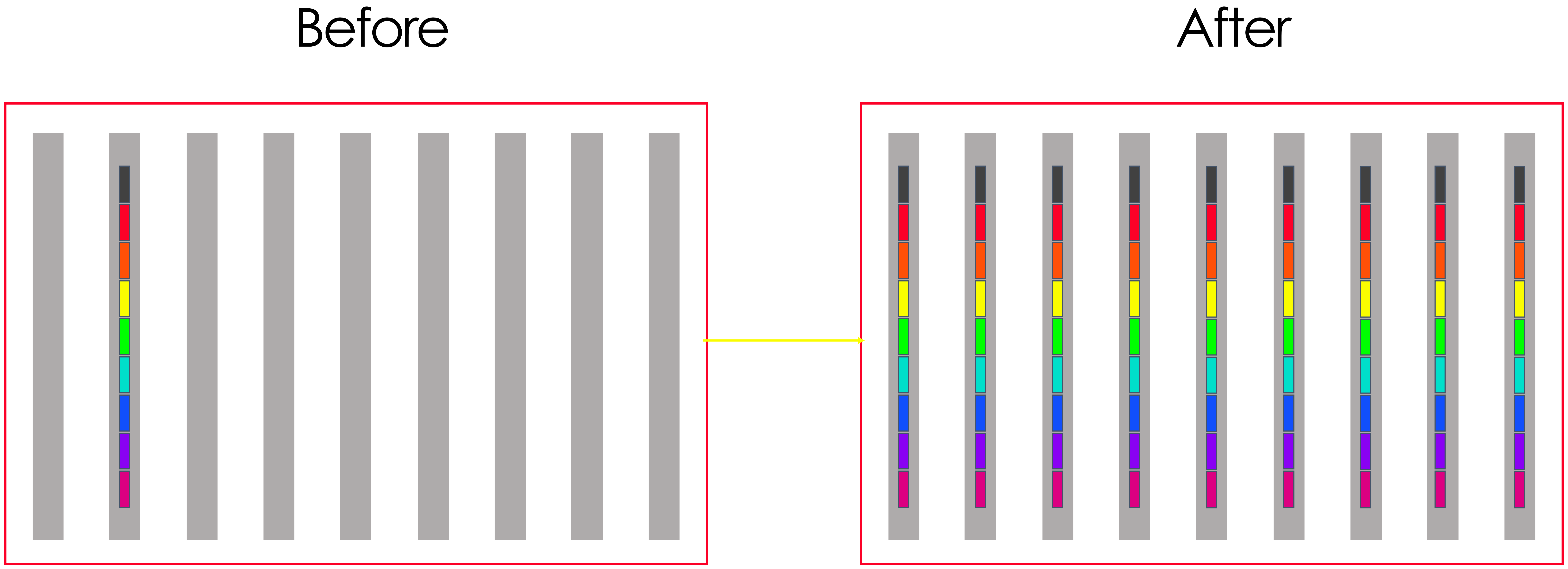
- send message to the half of the network that does not contain the current node (root) that holds the message

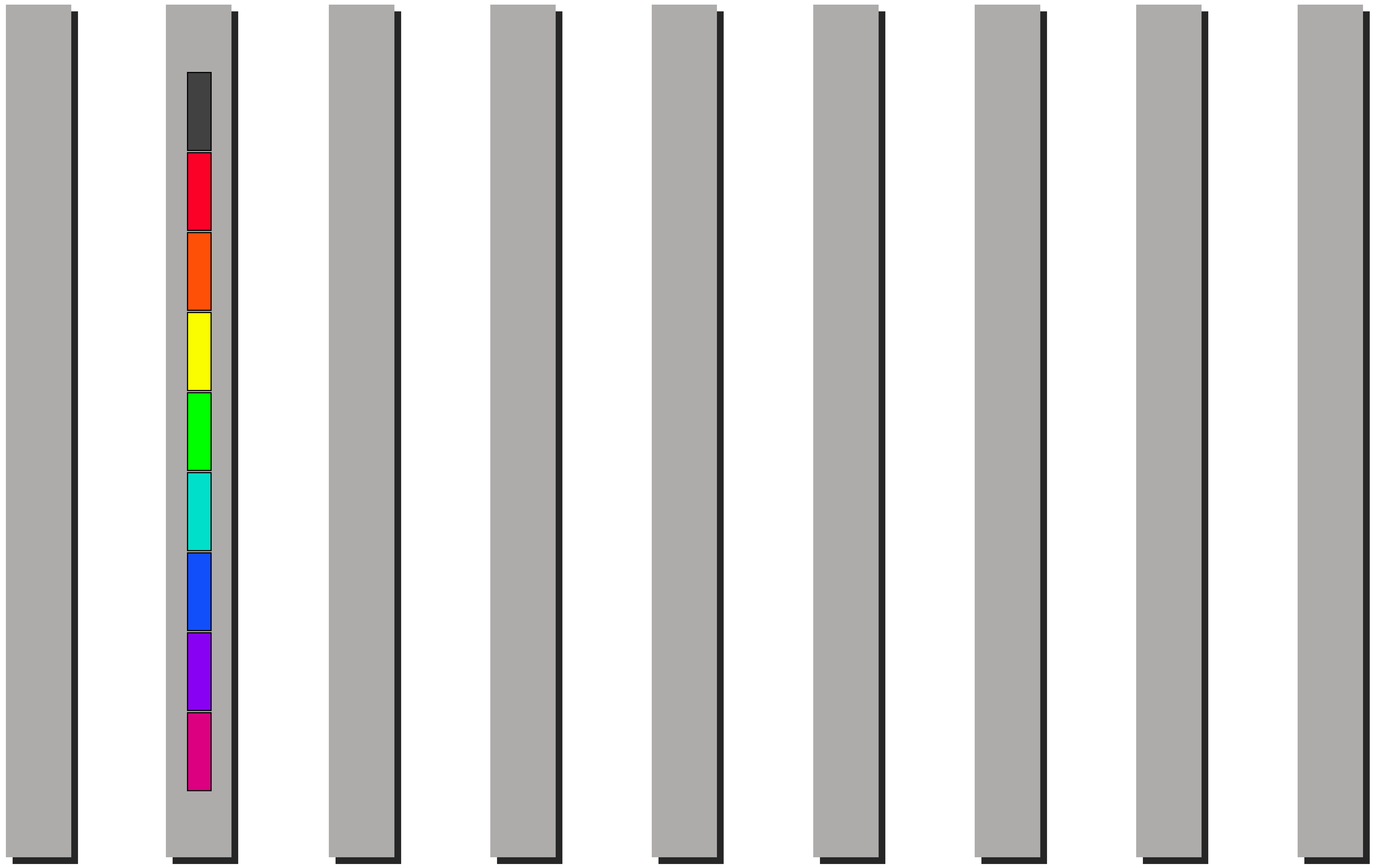
General principles

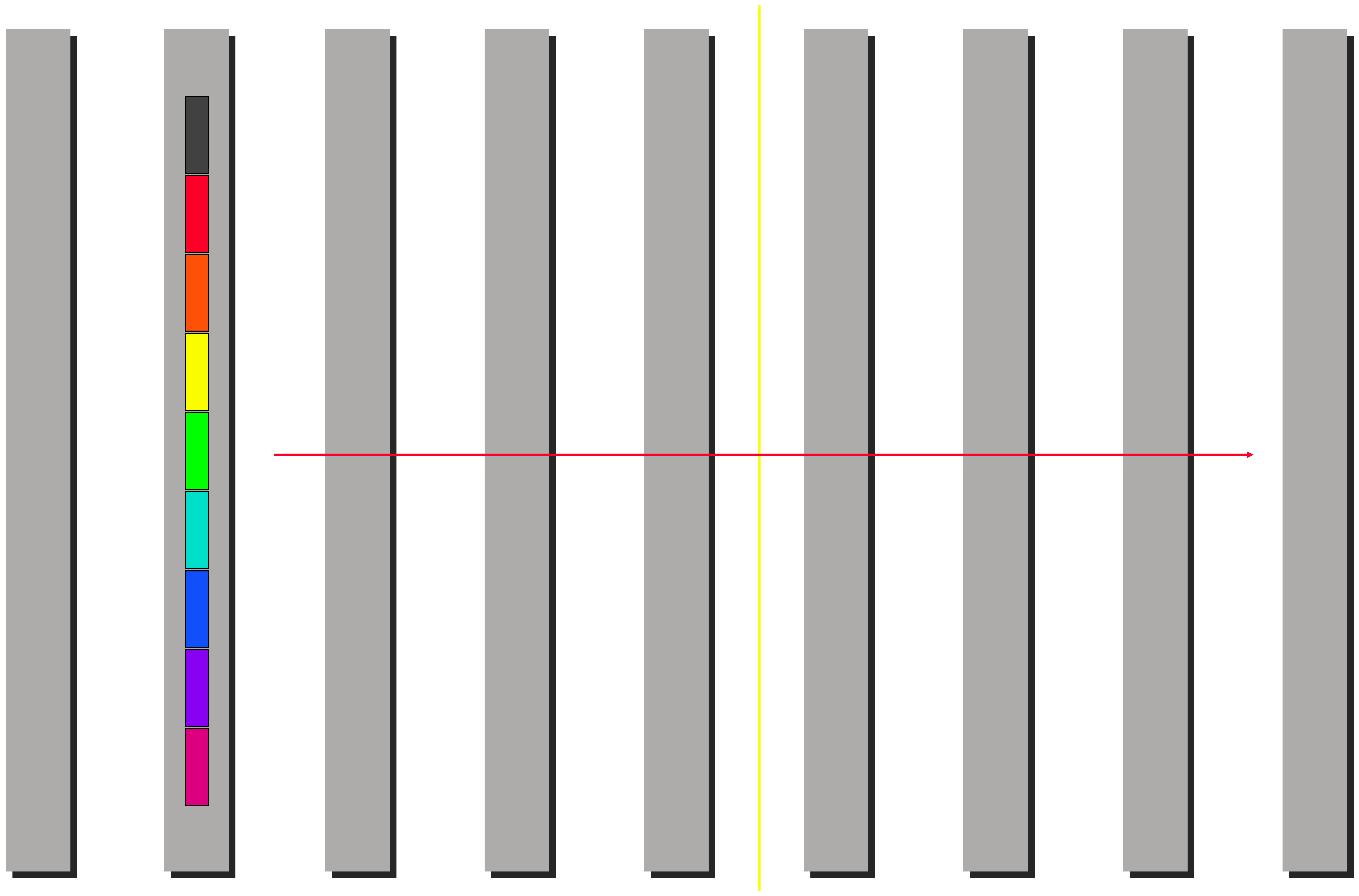


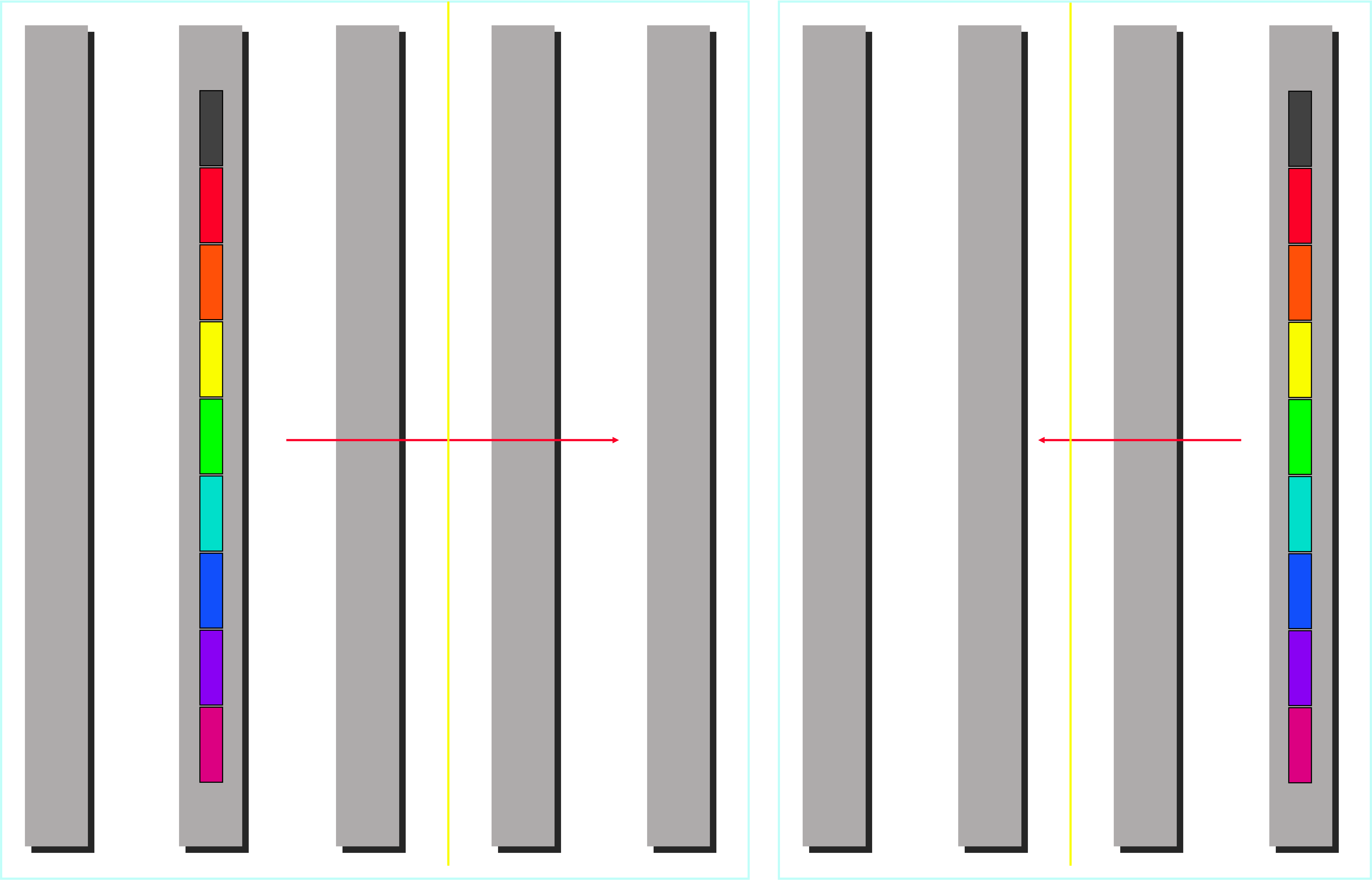
- continue recursively in each of the two halves

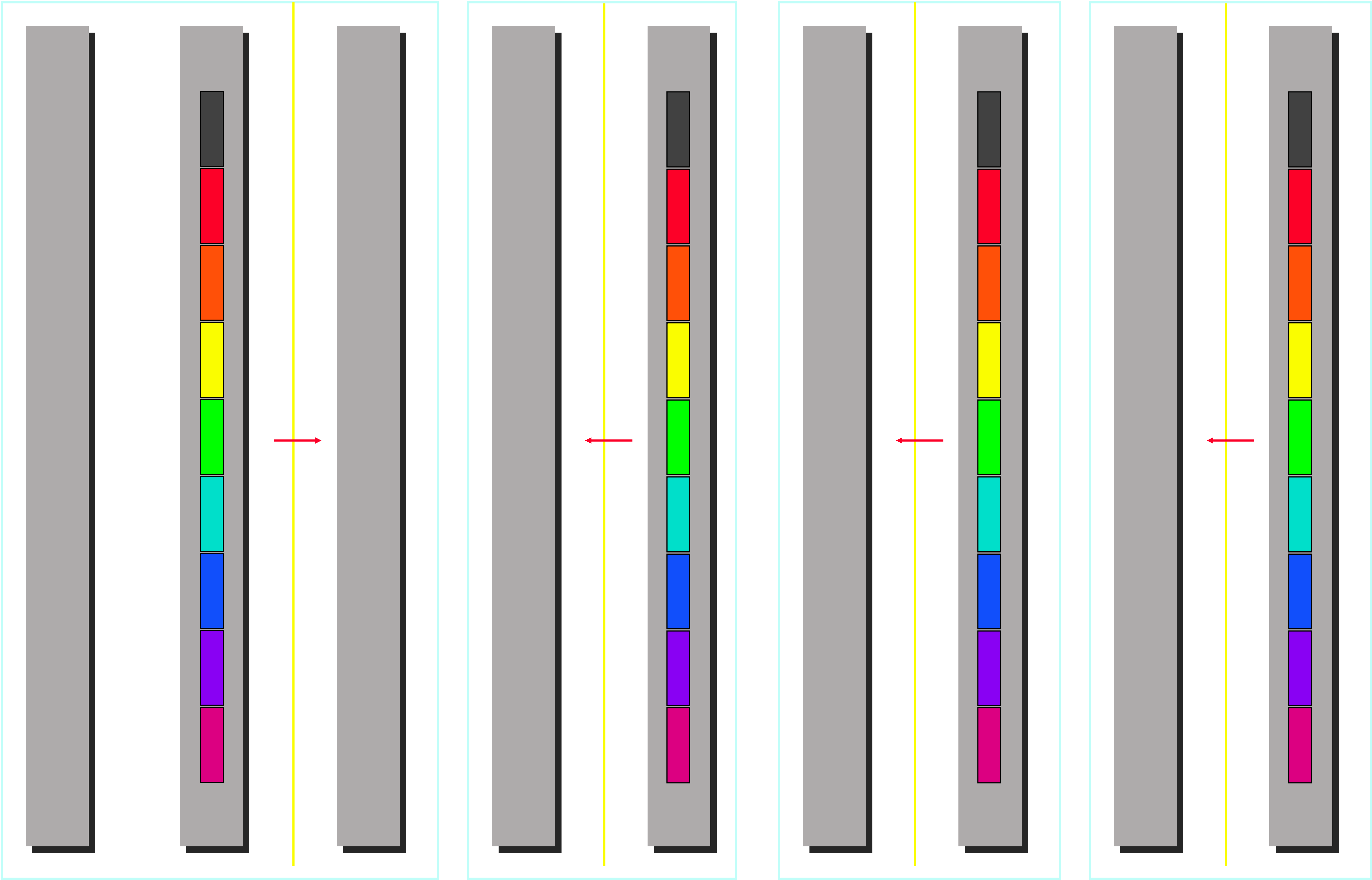
Broadcast

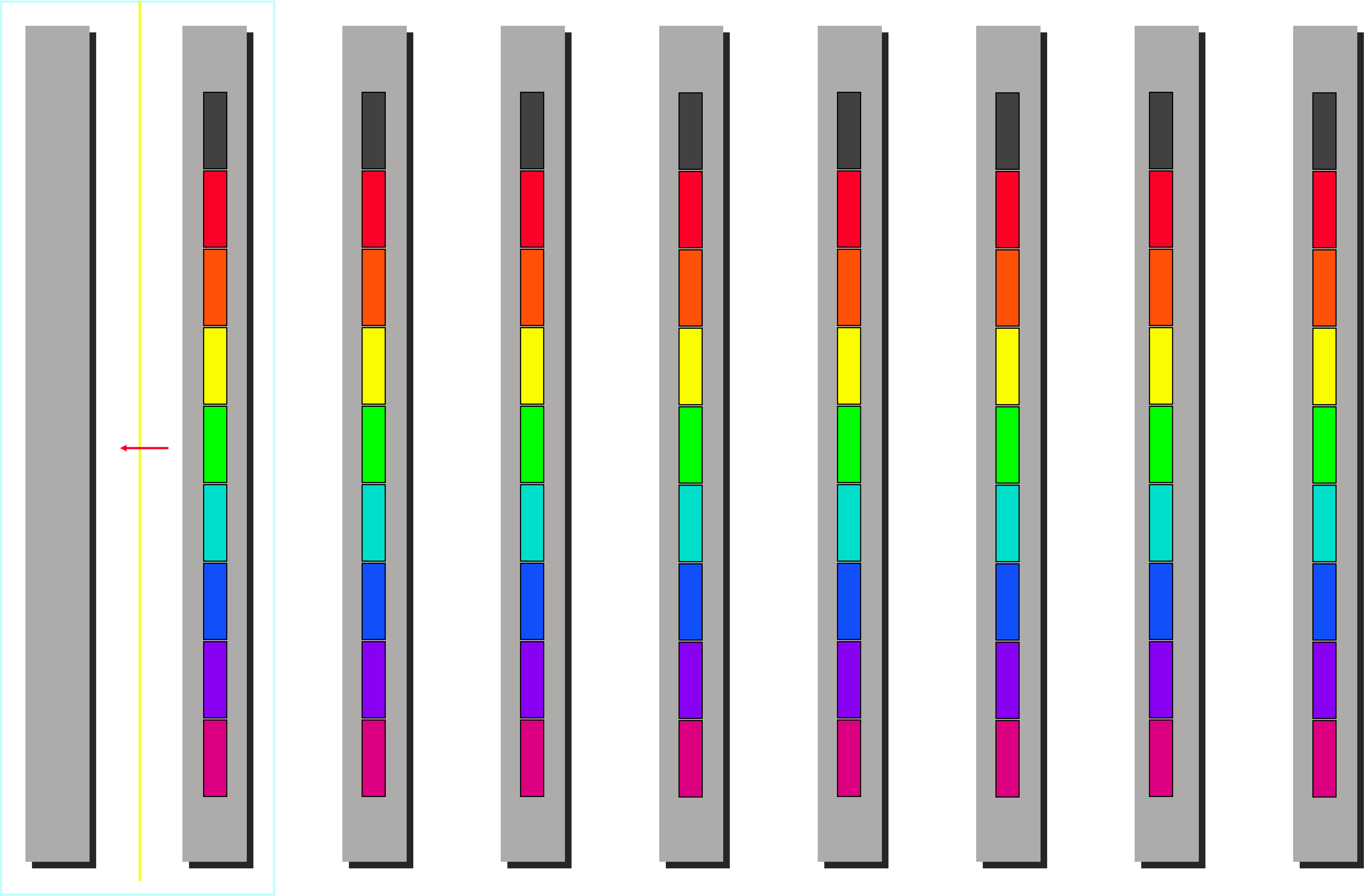


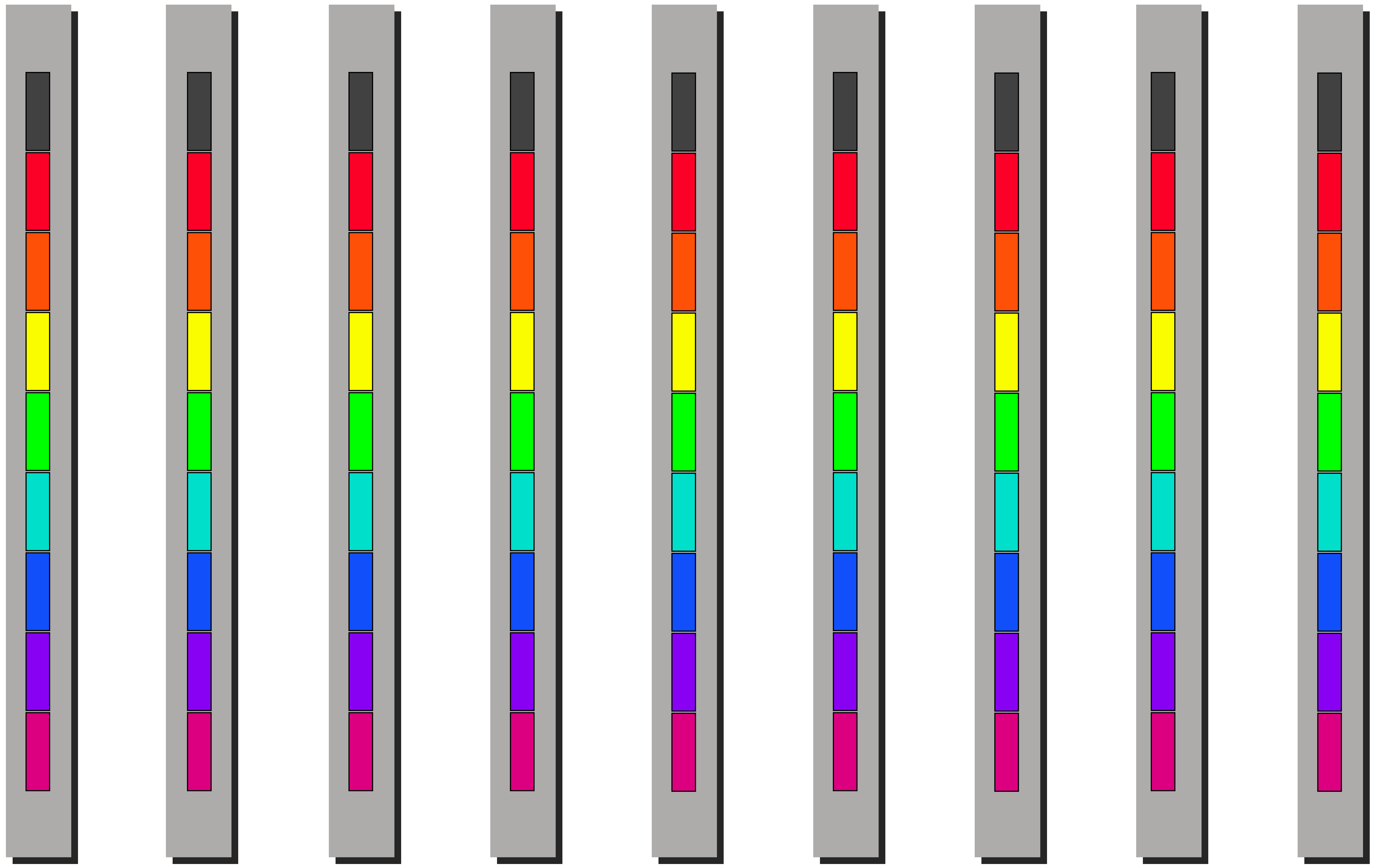












Recap

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

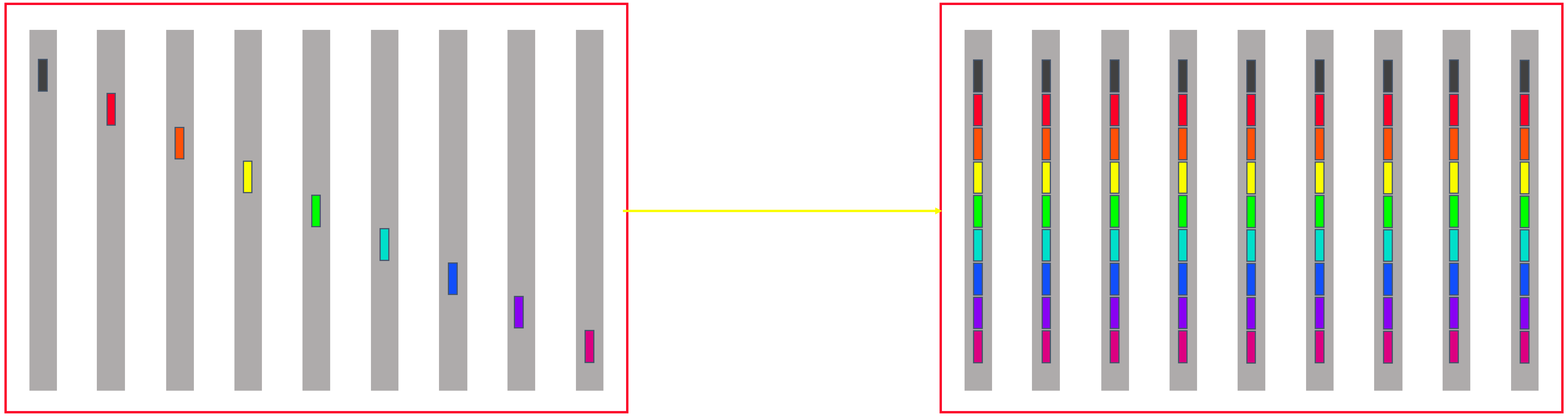
$$log(p)(\alpha + n\beta)$$

Reduce-scatter

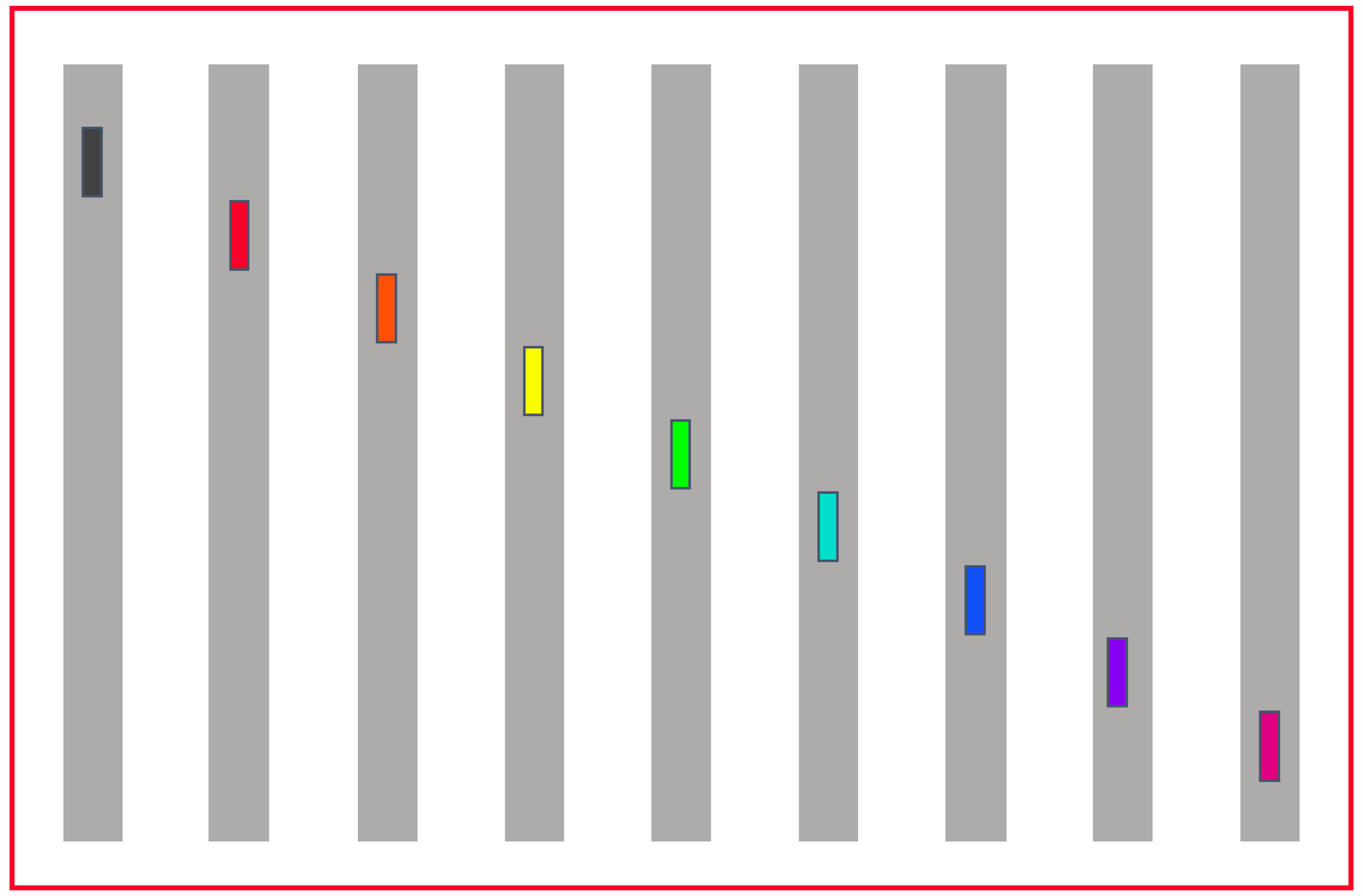
Allreduce

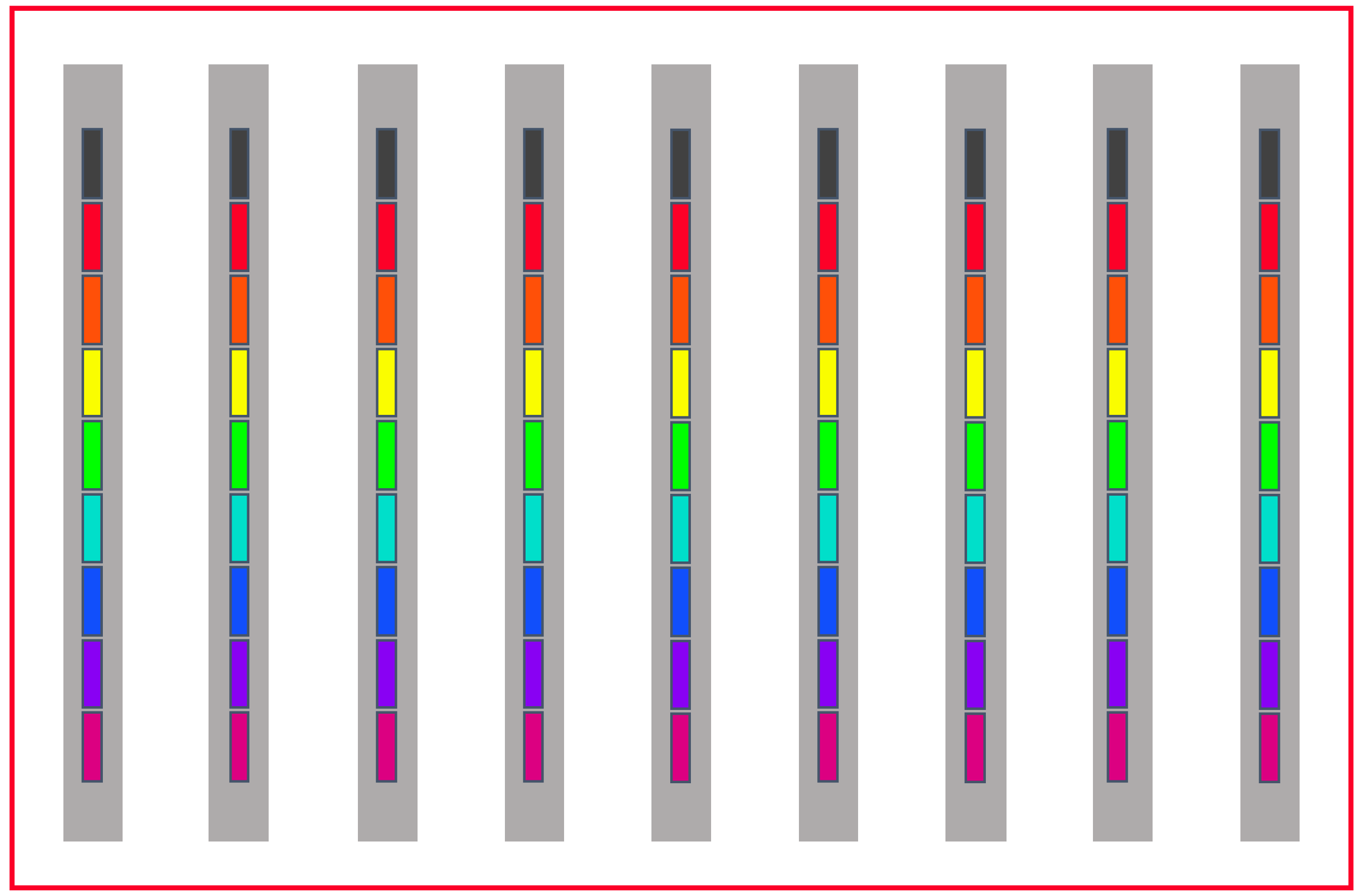
Allgather

Allgather

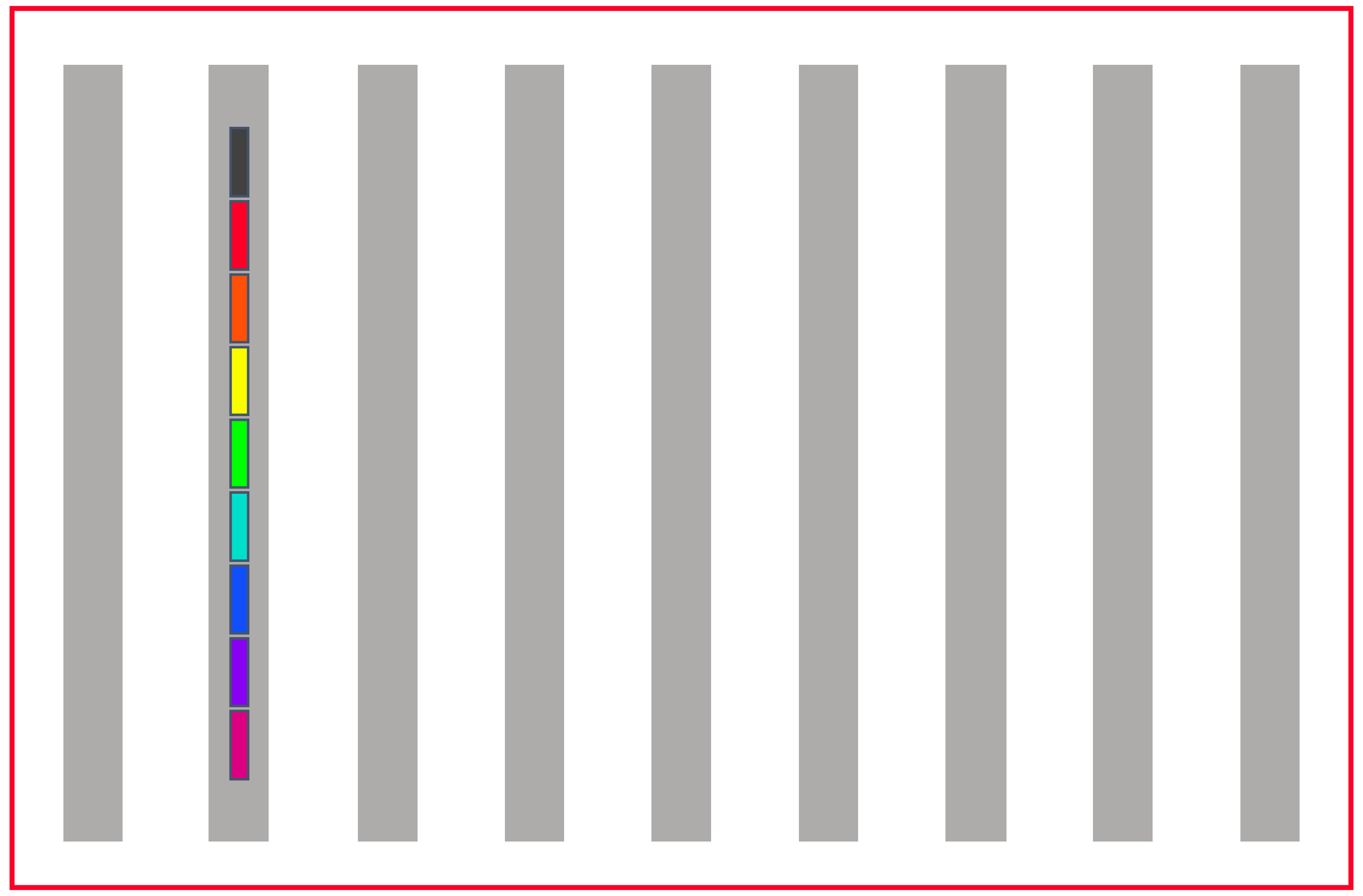


Allgather

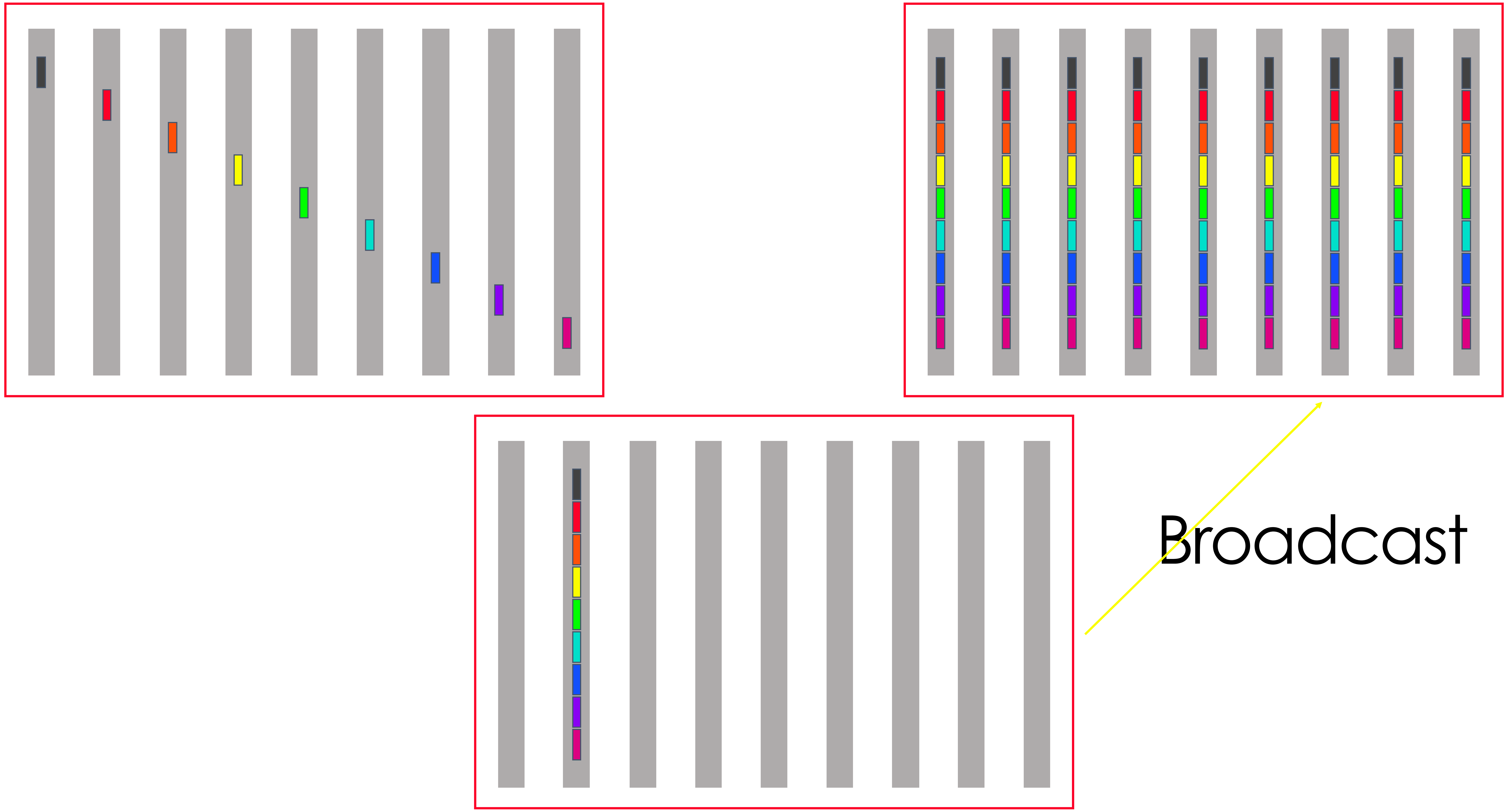




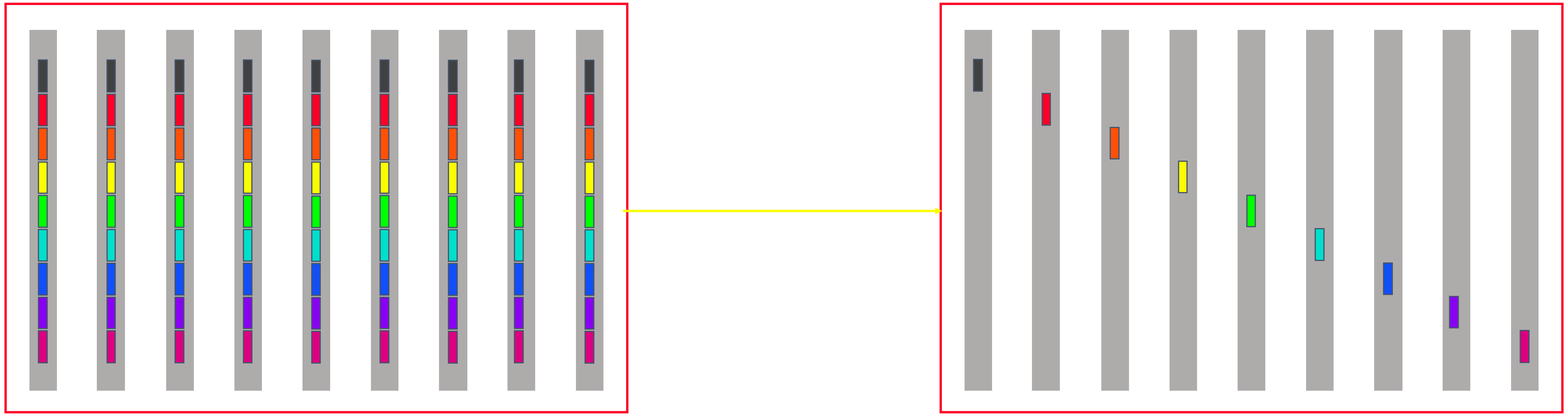
Gather



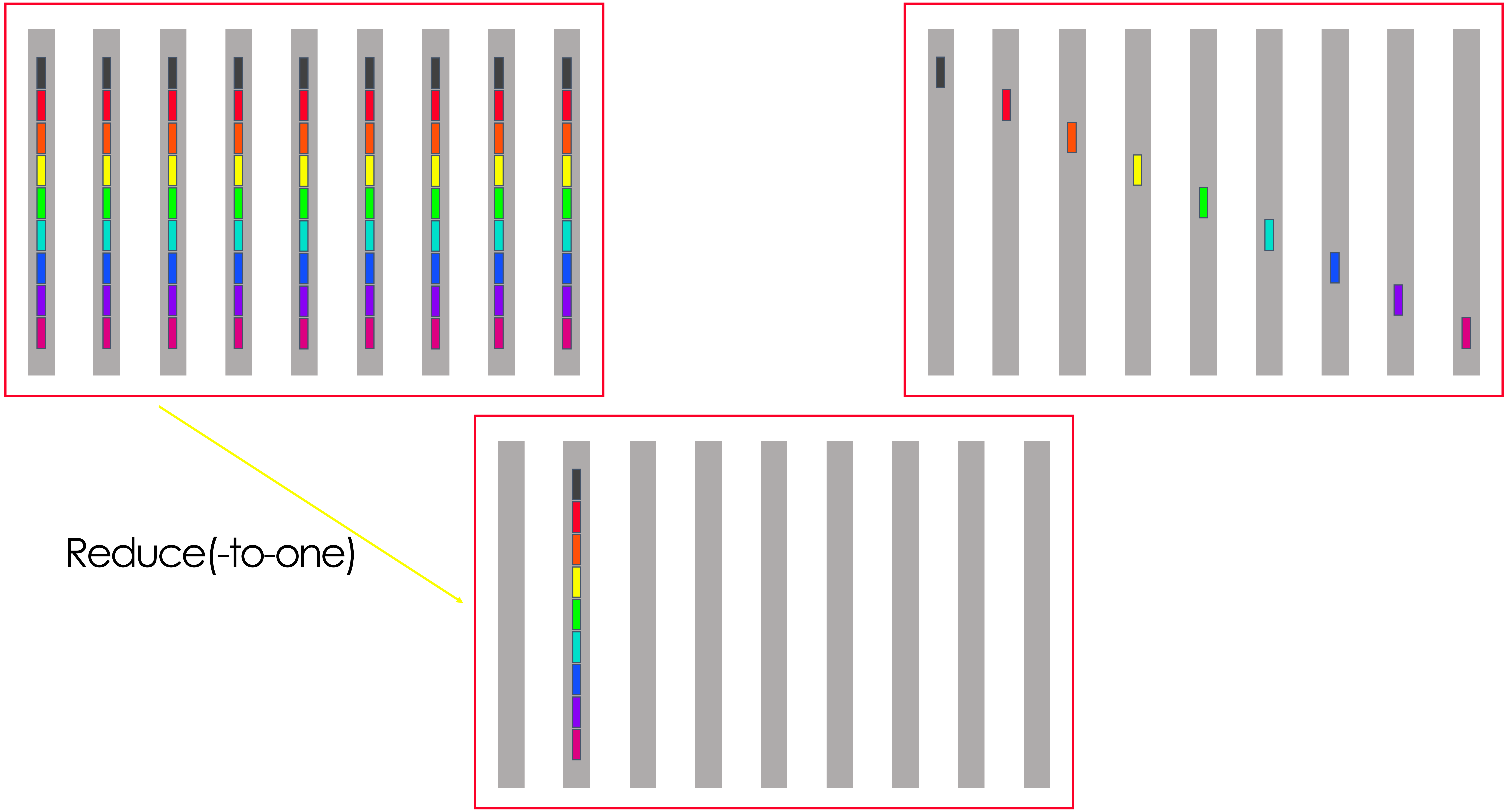
Allgather (short vector)



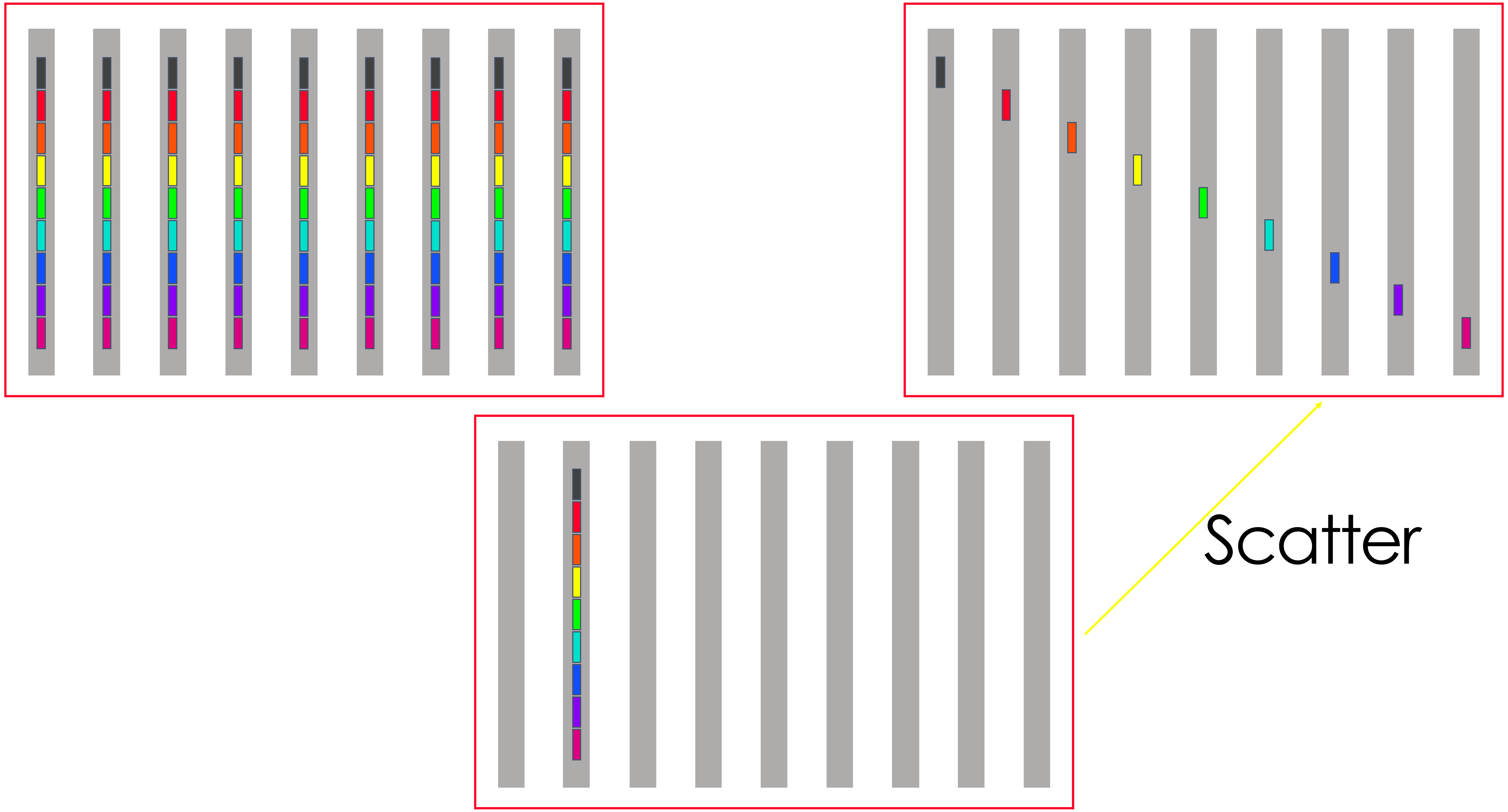
Reduce-scatter (small message)



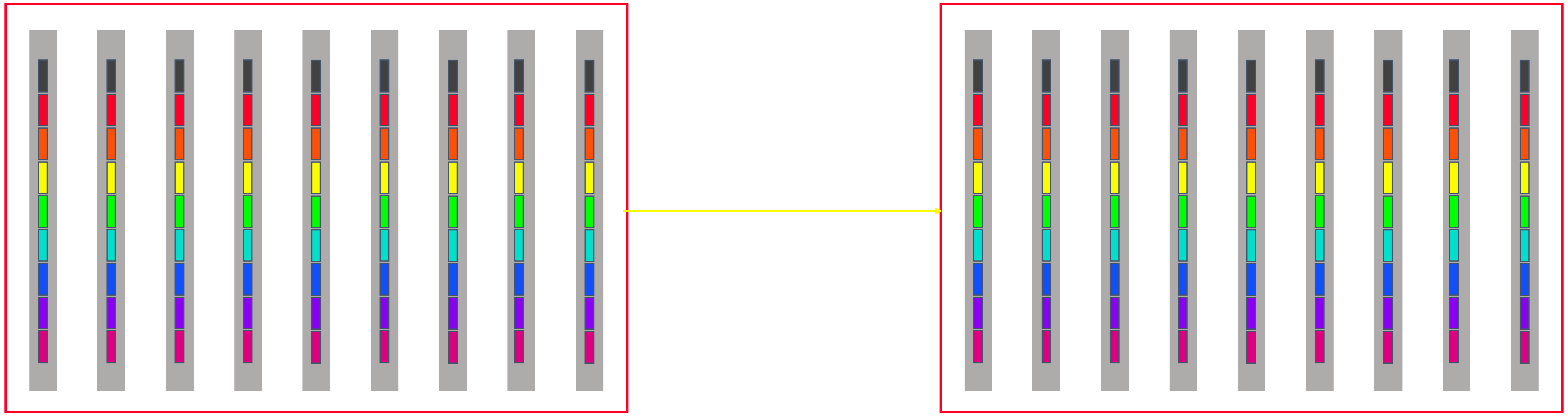
Reduce-scatter (short vector)



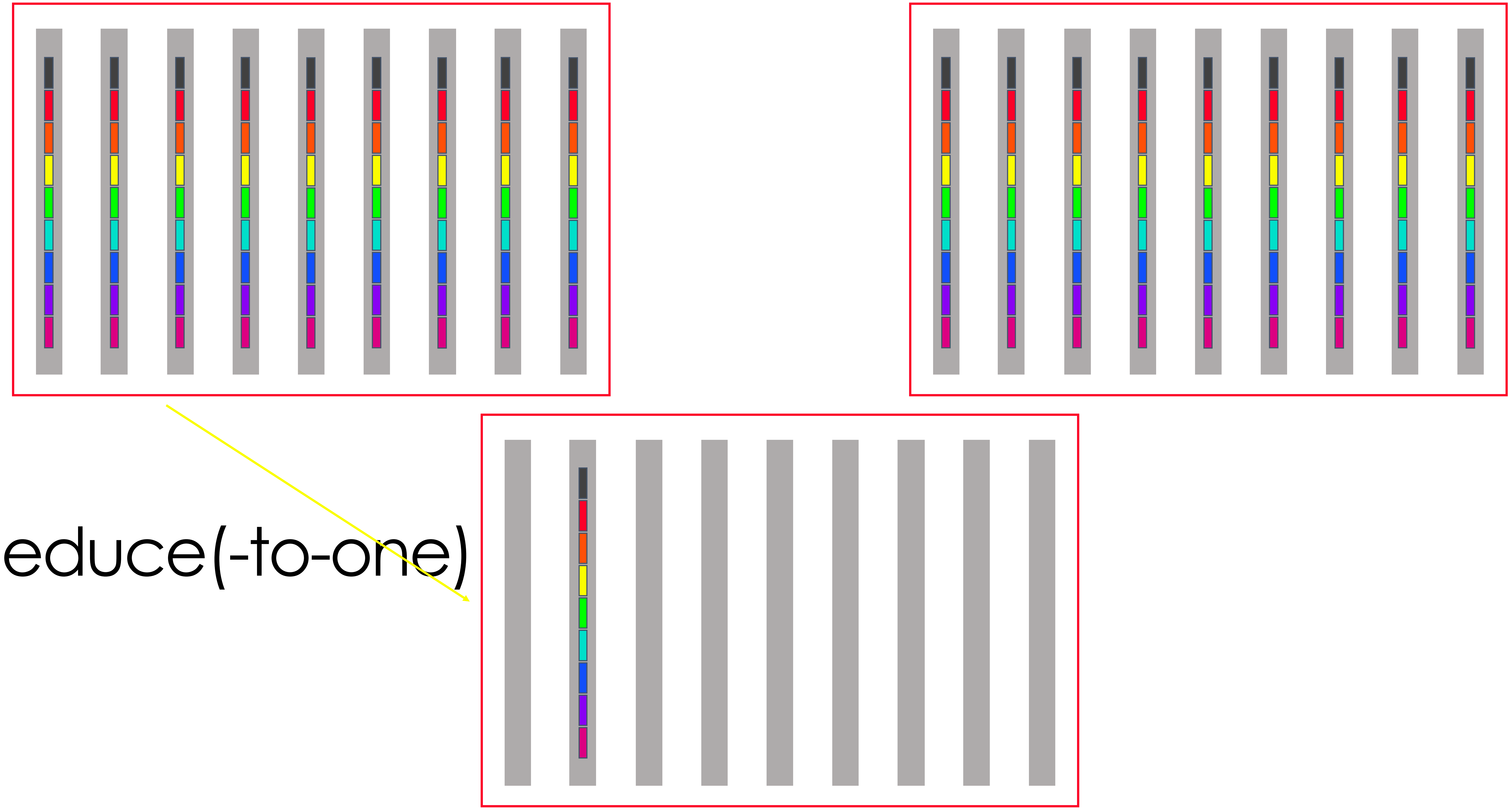
Reduce-scatter (short vector)



Allreduce (Latency-optimized)

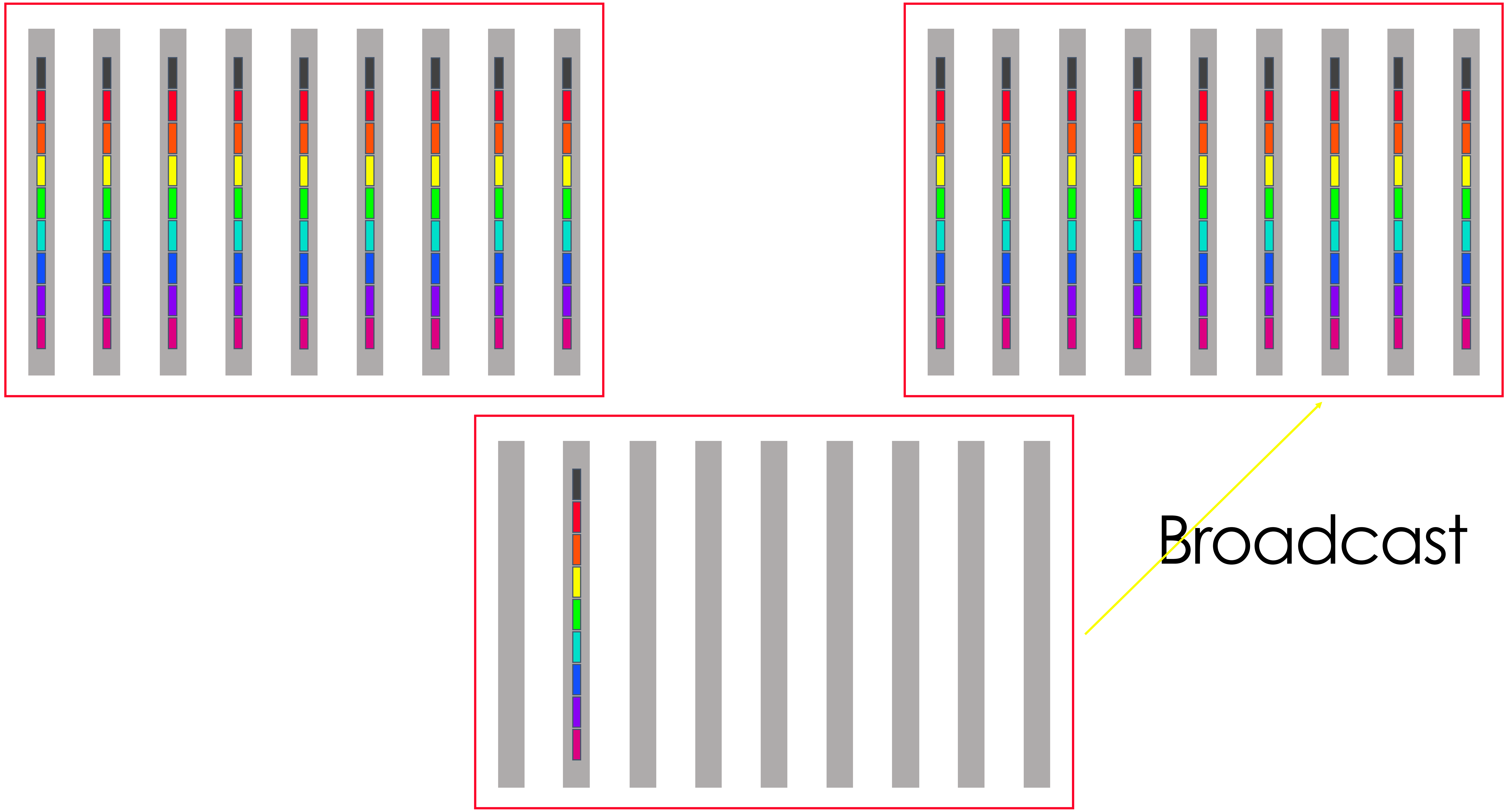


Allreduce (Latency-optimized)



Reduce(-to-one)

Allreduce
(short vector)



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

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

Summary of MST algorithms

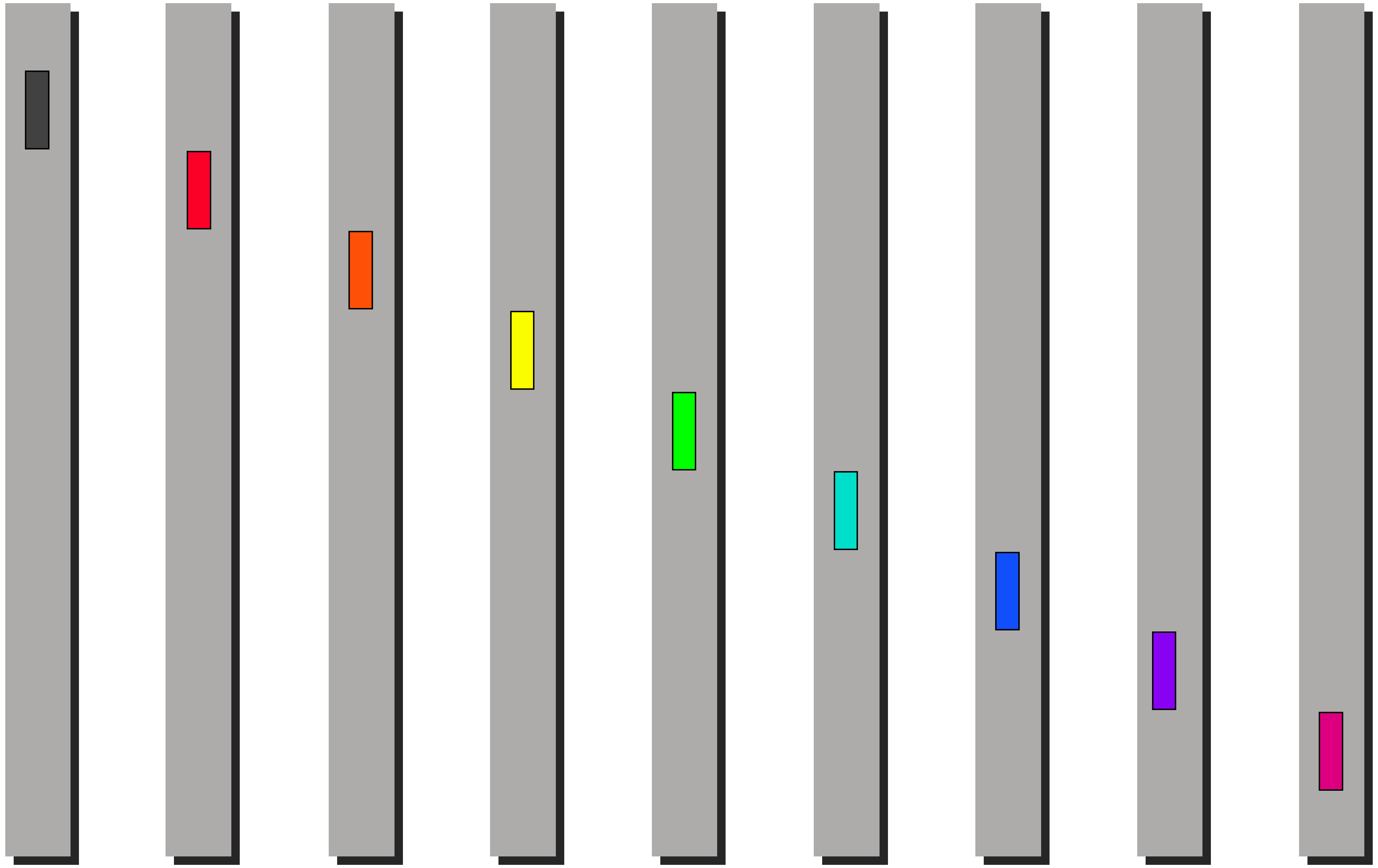
- Small message: Minimum Spanning Tree algorithm
  - Emphasize **low latency**

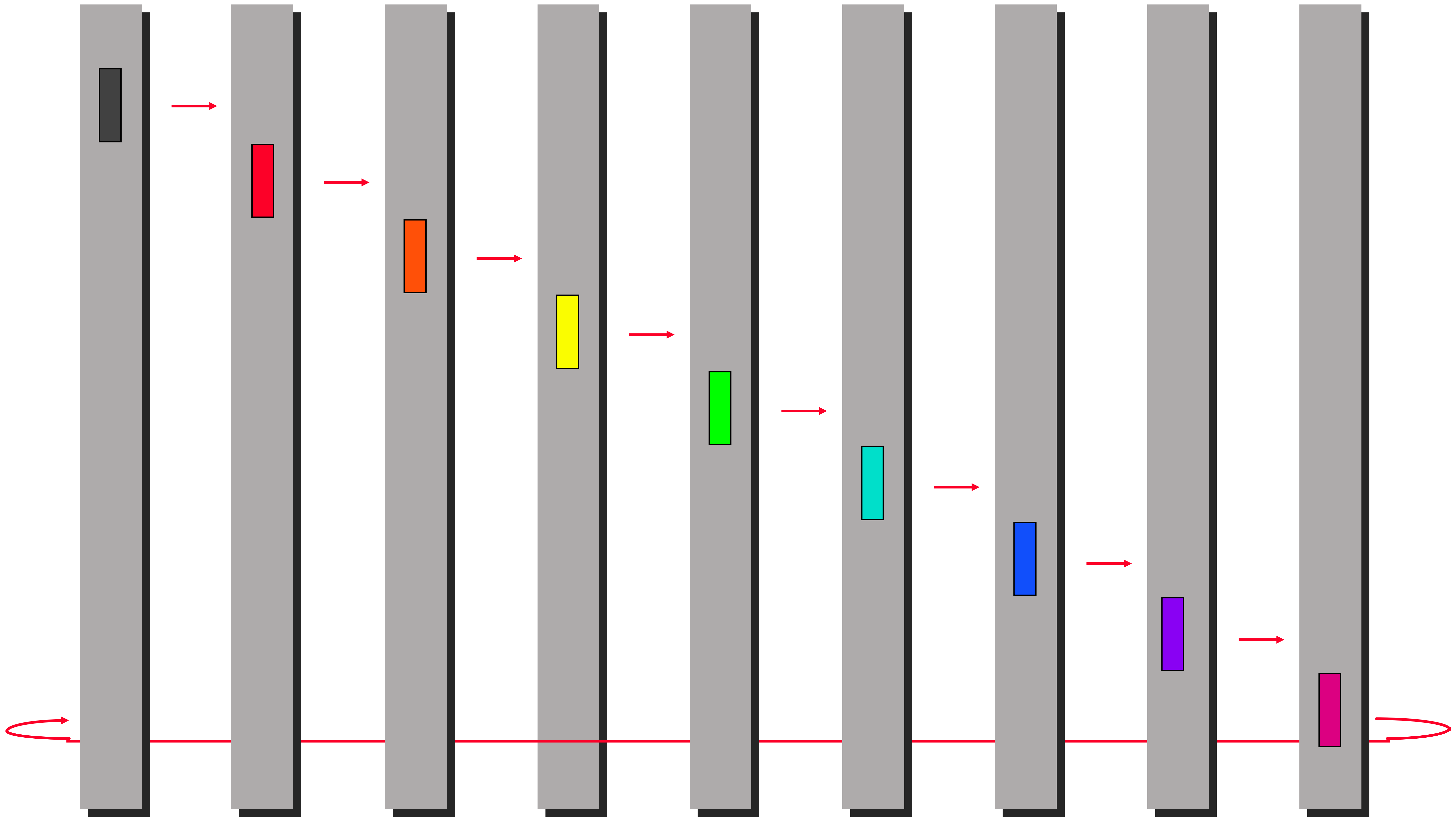
- Problem of Minimum Spanning Tree Algorithm?
  - It prioritize latency rather than bandwidth
  - Hence: Some links are idle

- Next: Large message size algorithm

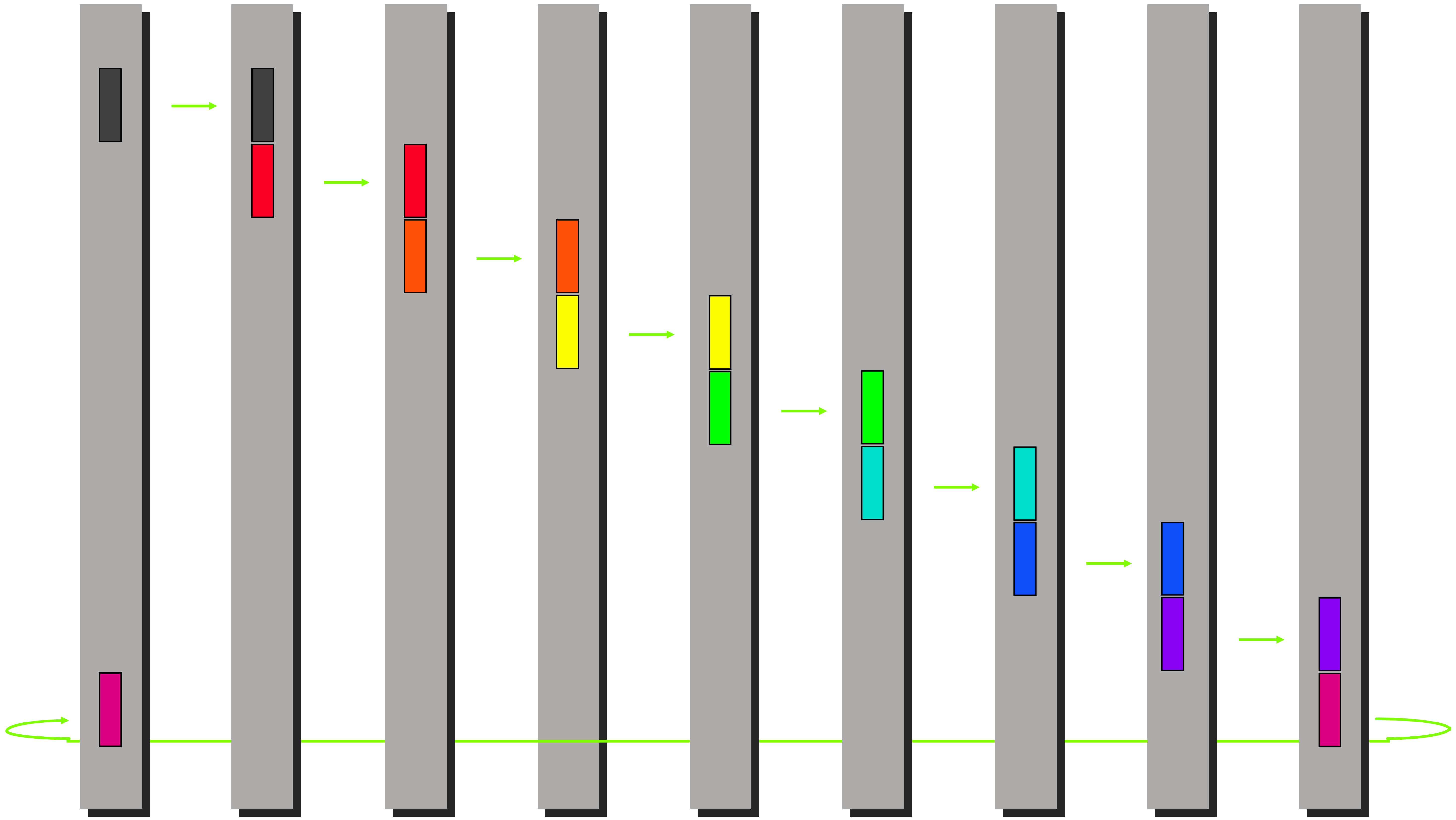
General principles: High Bandwidth

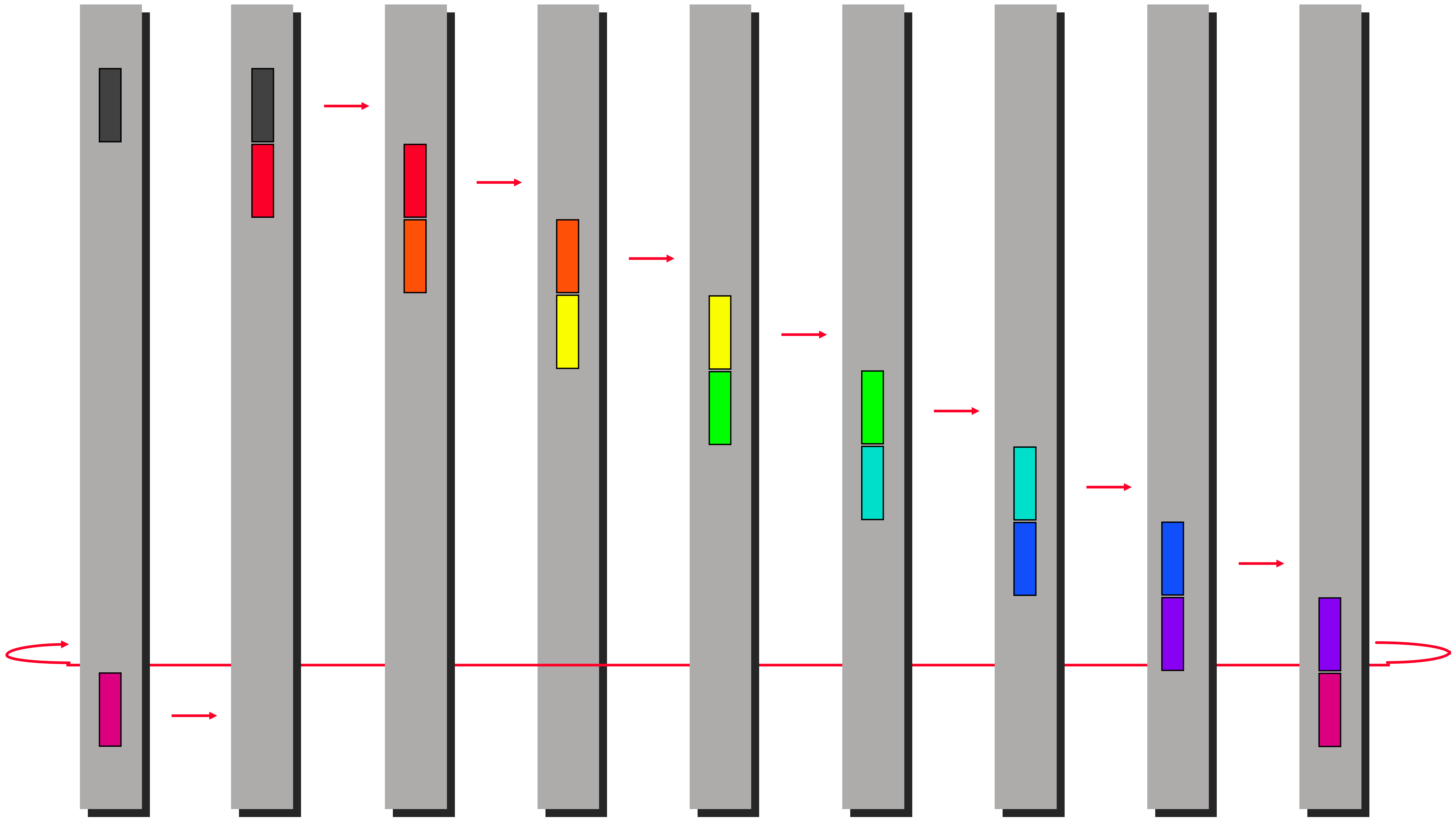
- Use all the links between every two nodes
- How many rounds of communication does not matter
- Ring algorithm: A logical ring can be embedded in a physical linear array with worm-hole routing, since the “wrap-around” message doesn’t conflict

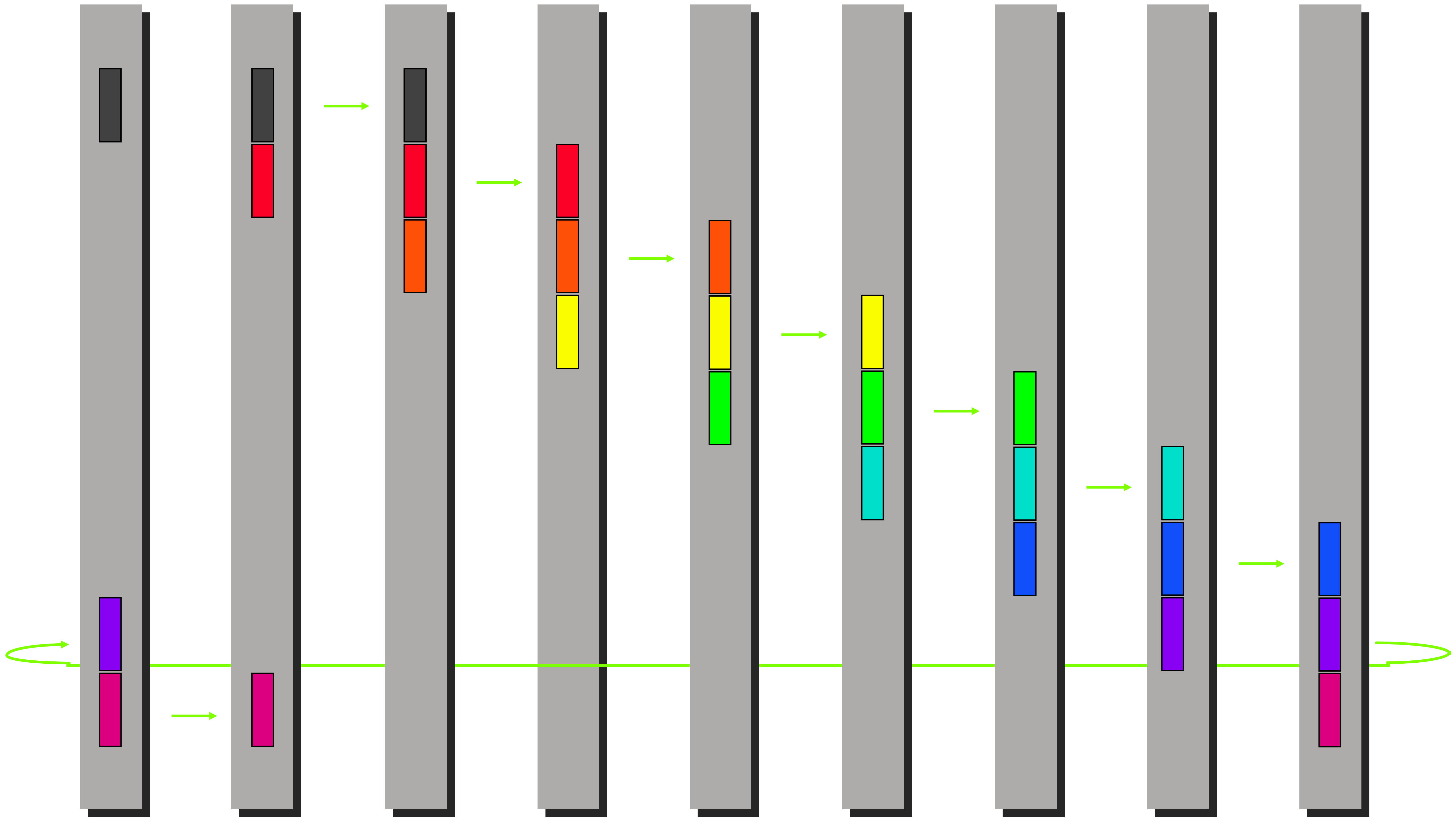




- A logical ring can be embedded in a physical linear array with worm-hole routing, since the “wrap-around” message doesn’t conflict





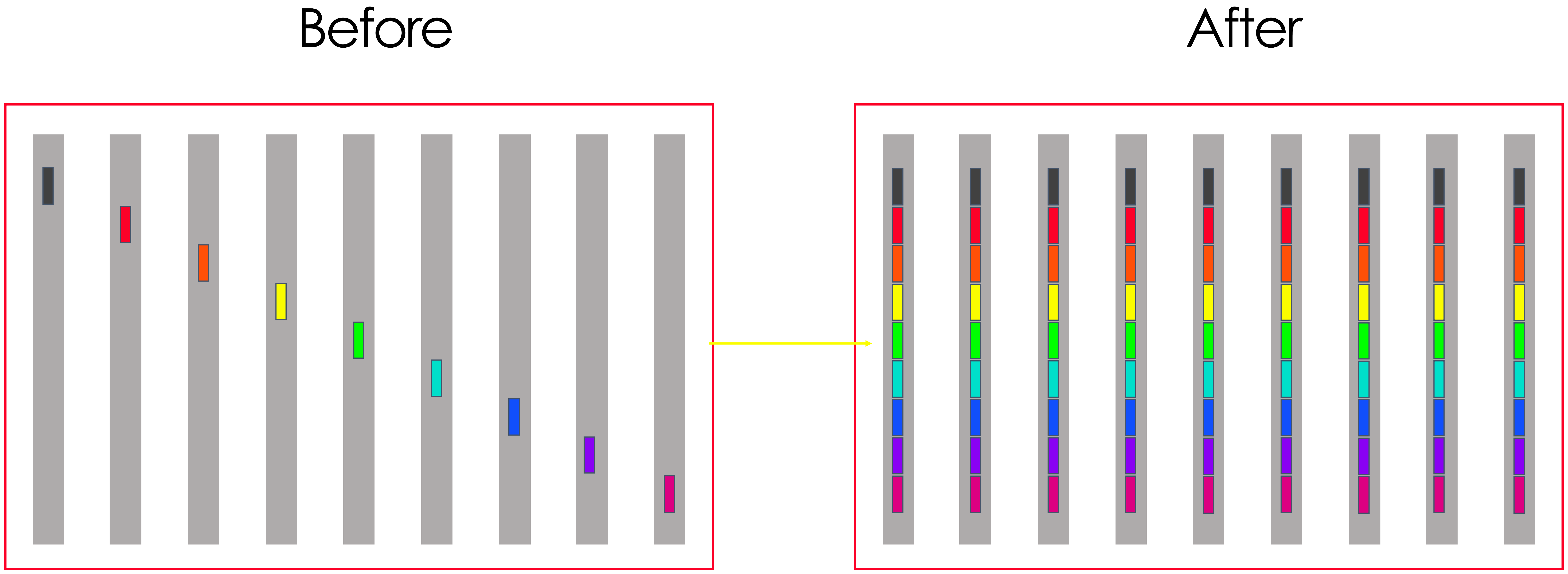


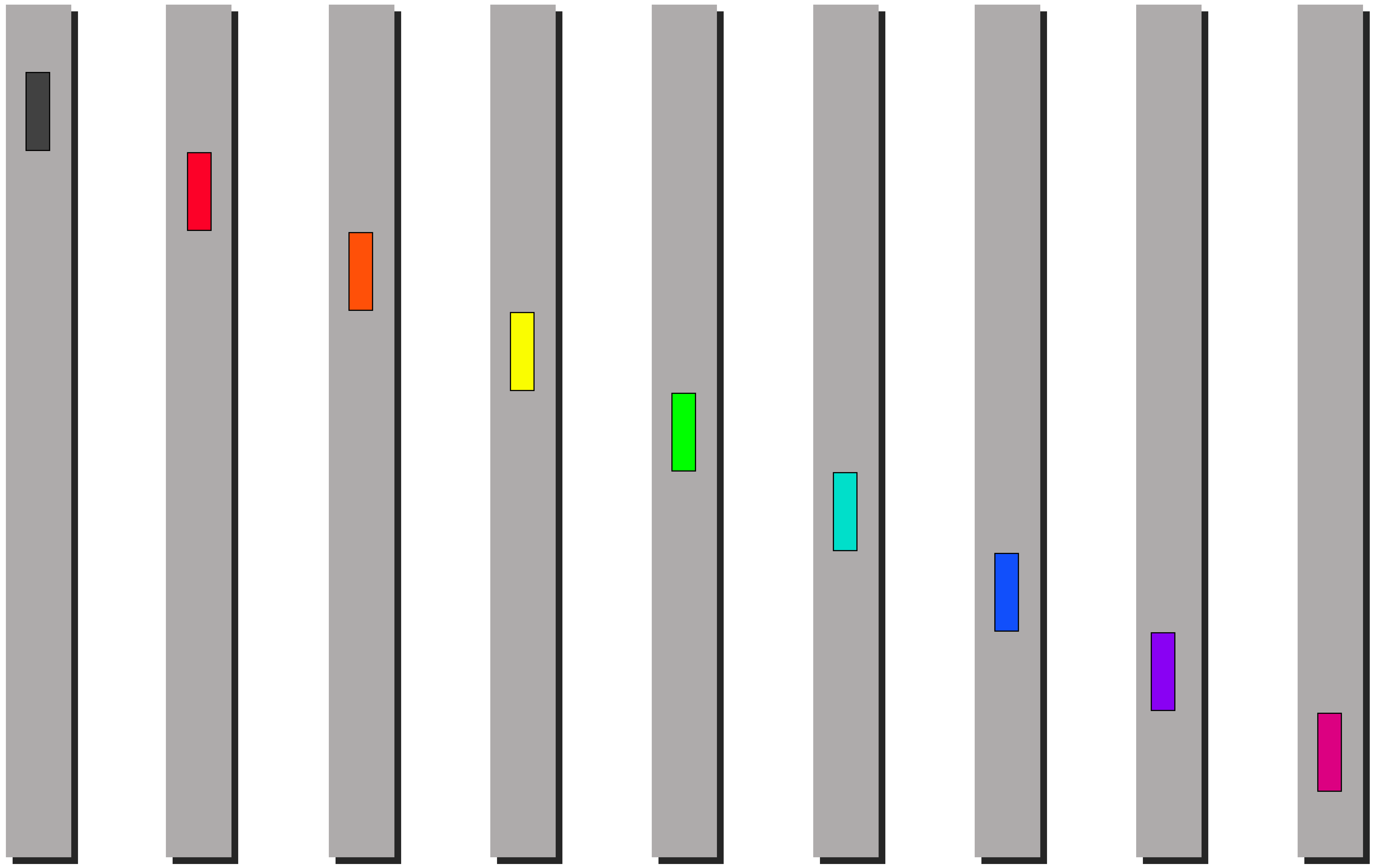
General principles

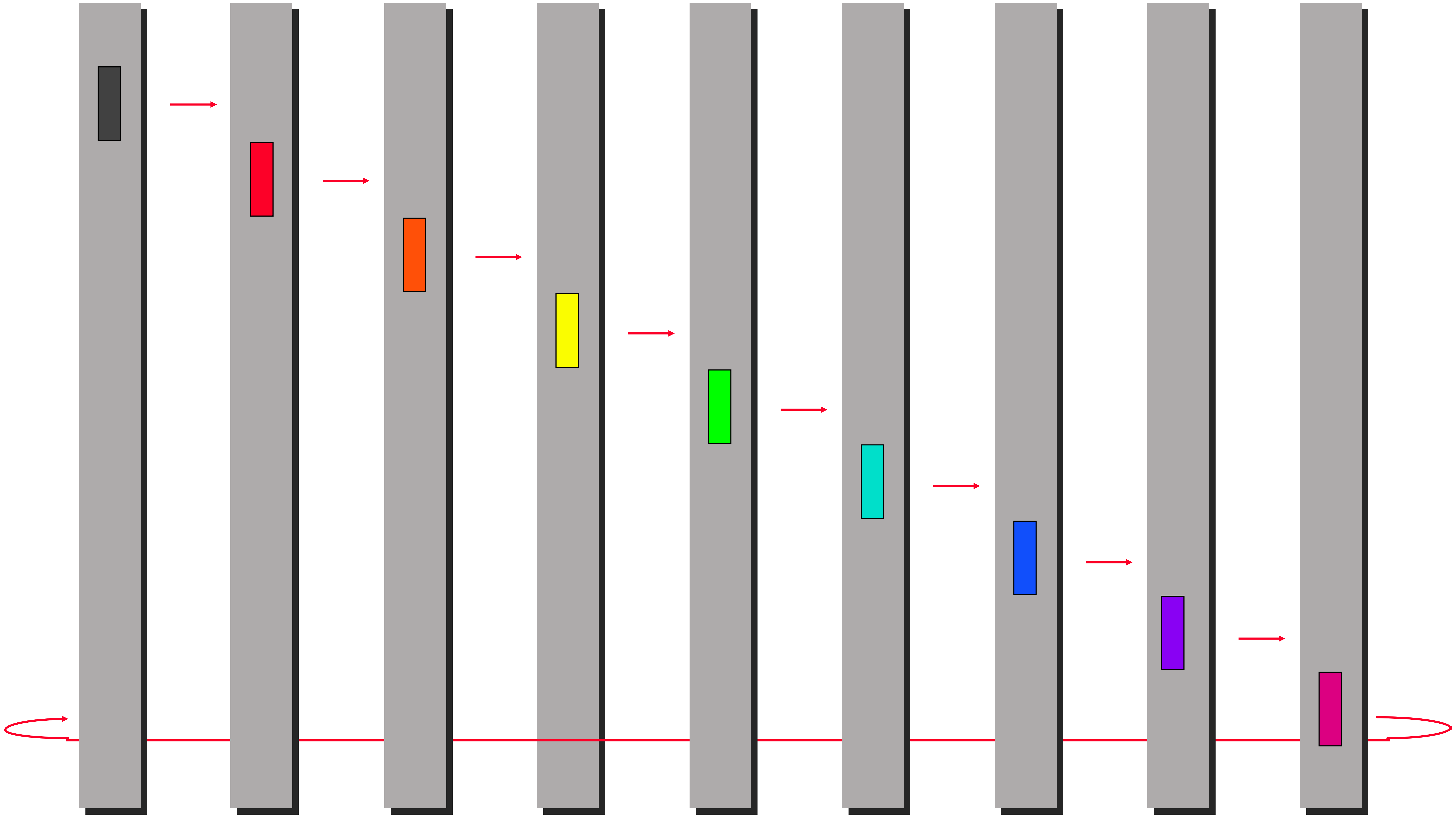
Ring algorithm has the following advantages

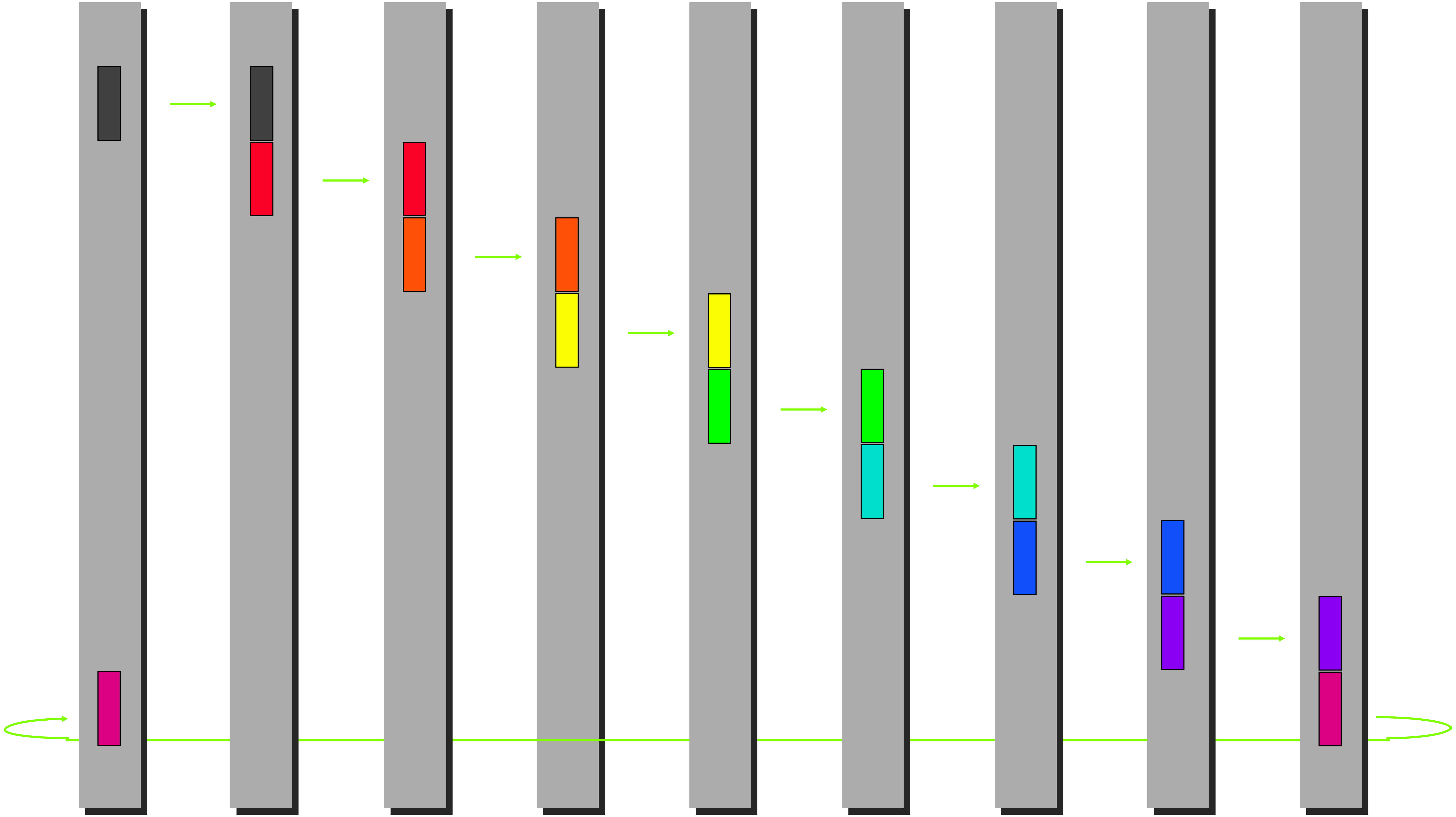
- Fully utilize the bandwidth (bandwidth optimal)
- implementation for arbitrary numbers of node

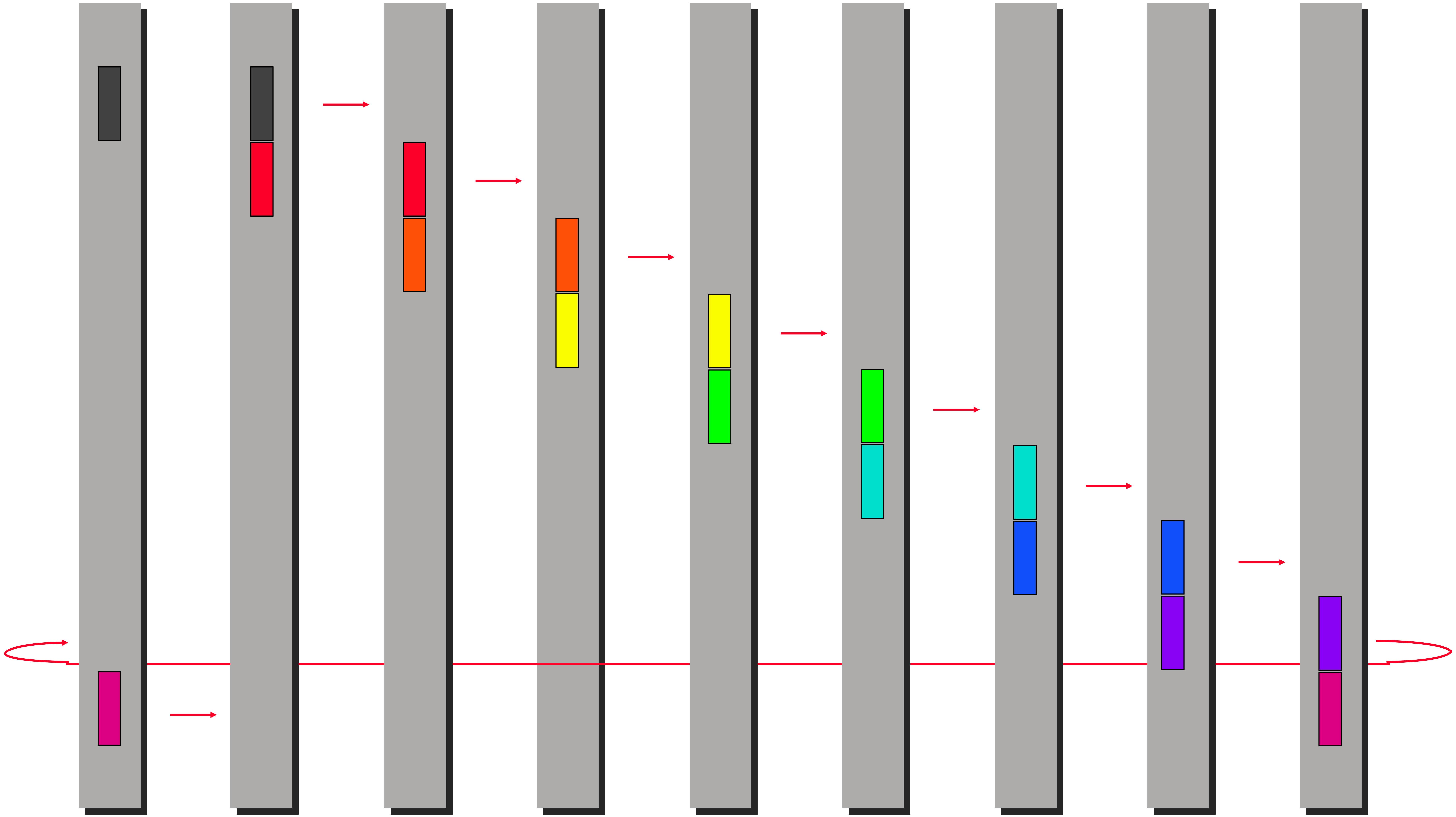
Allgather

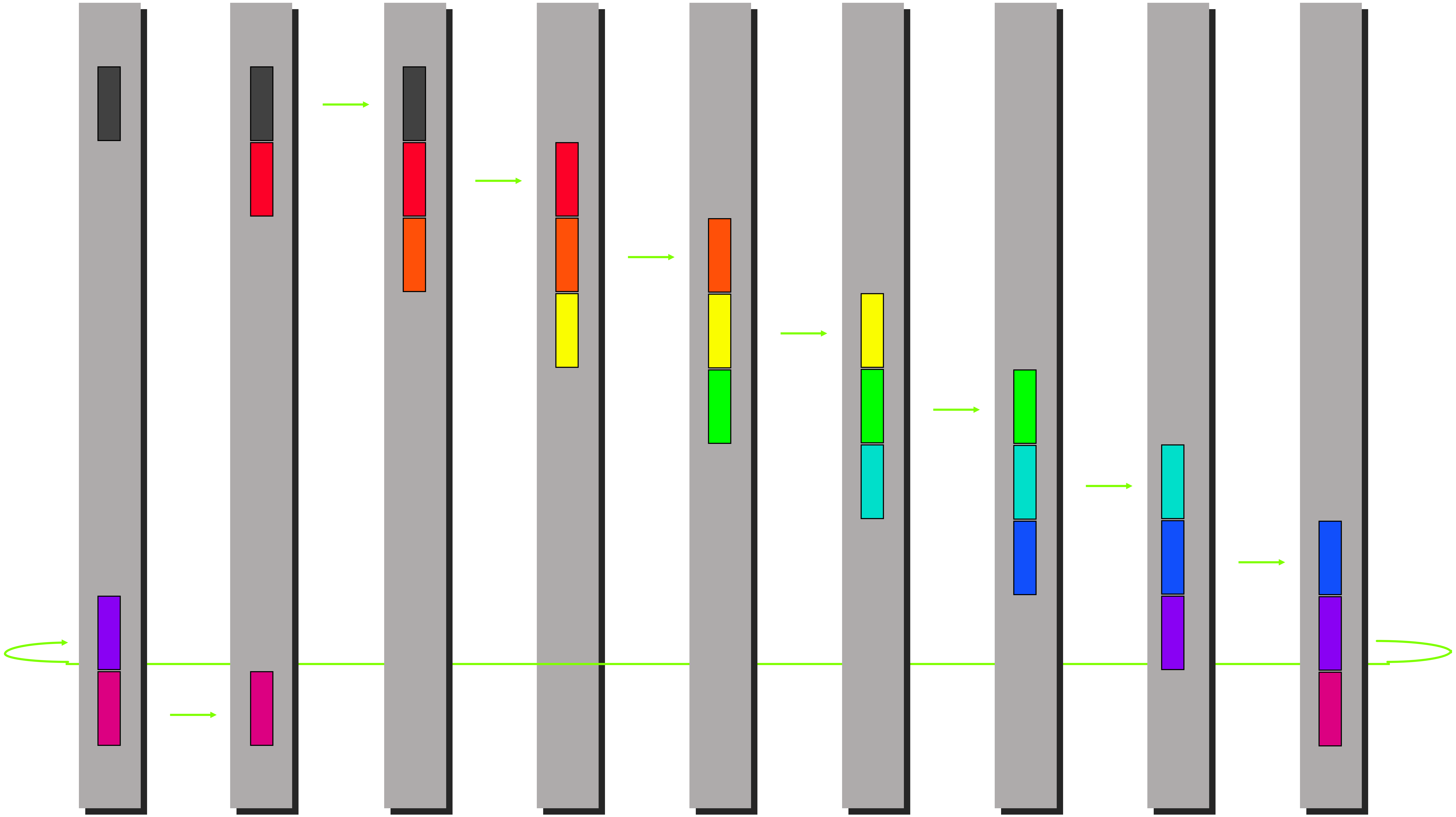


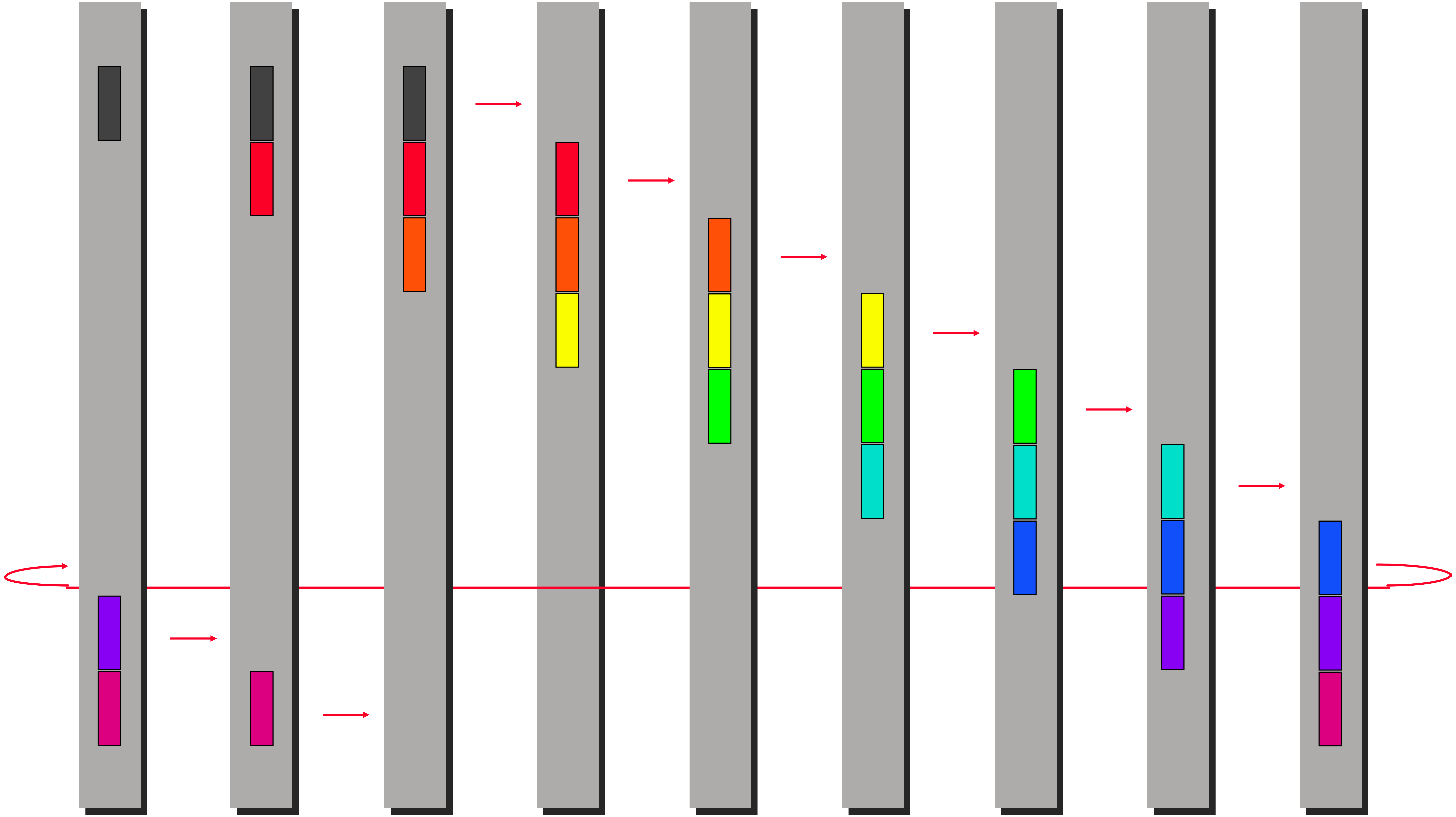


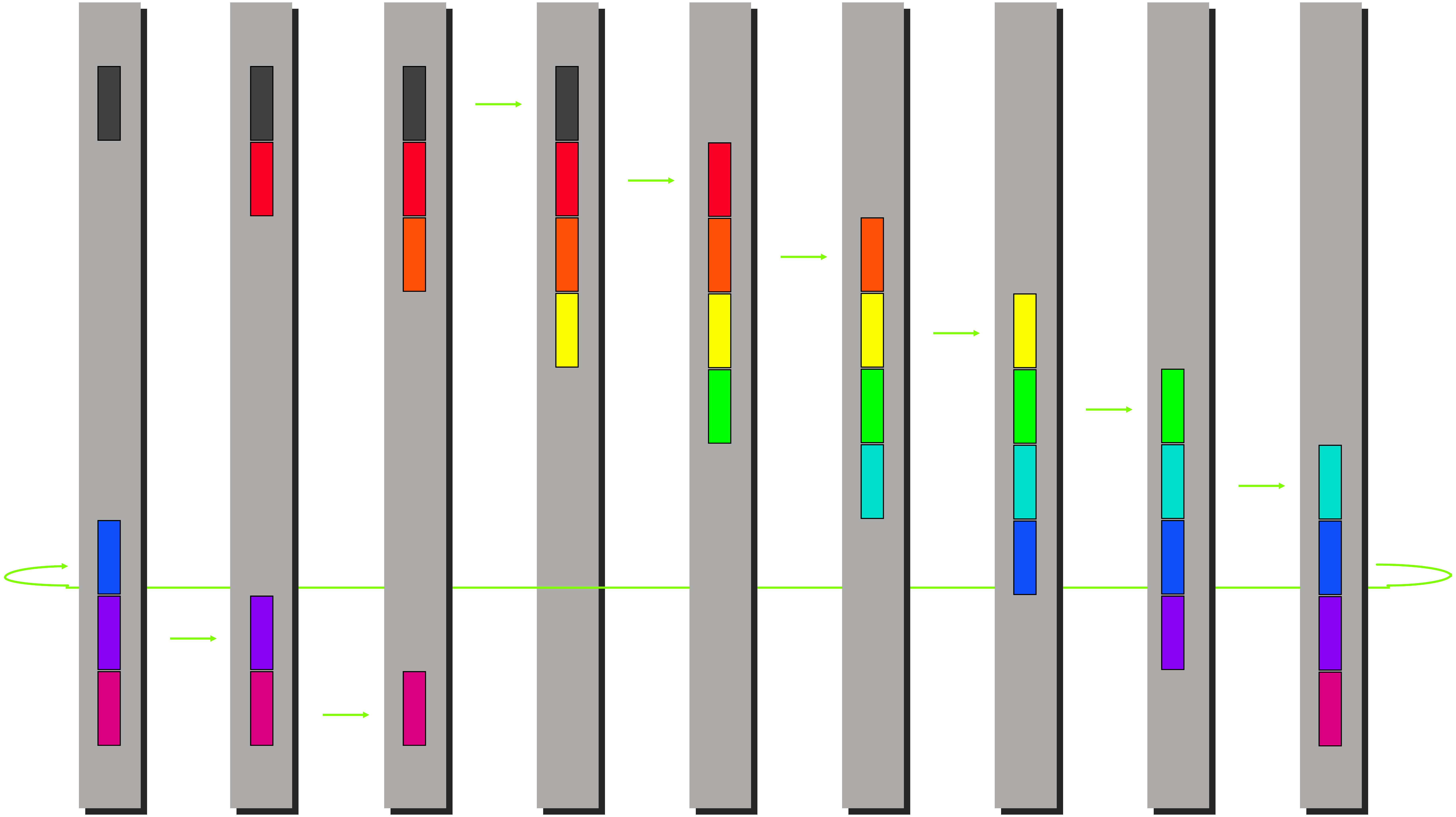


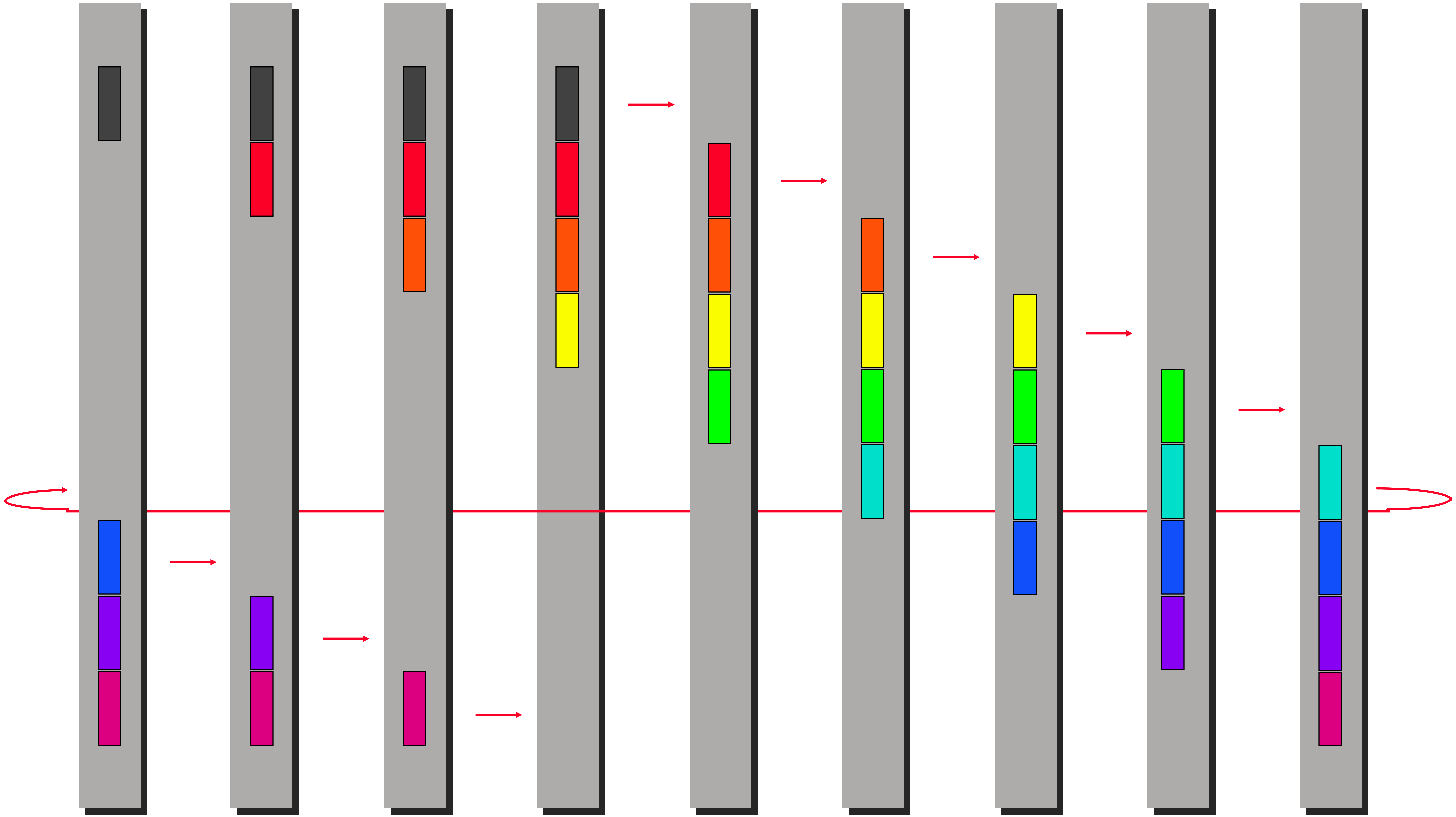


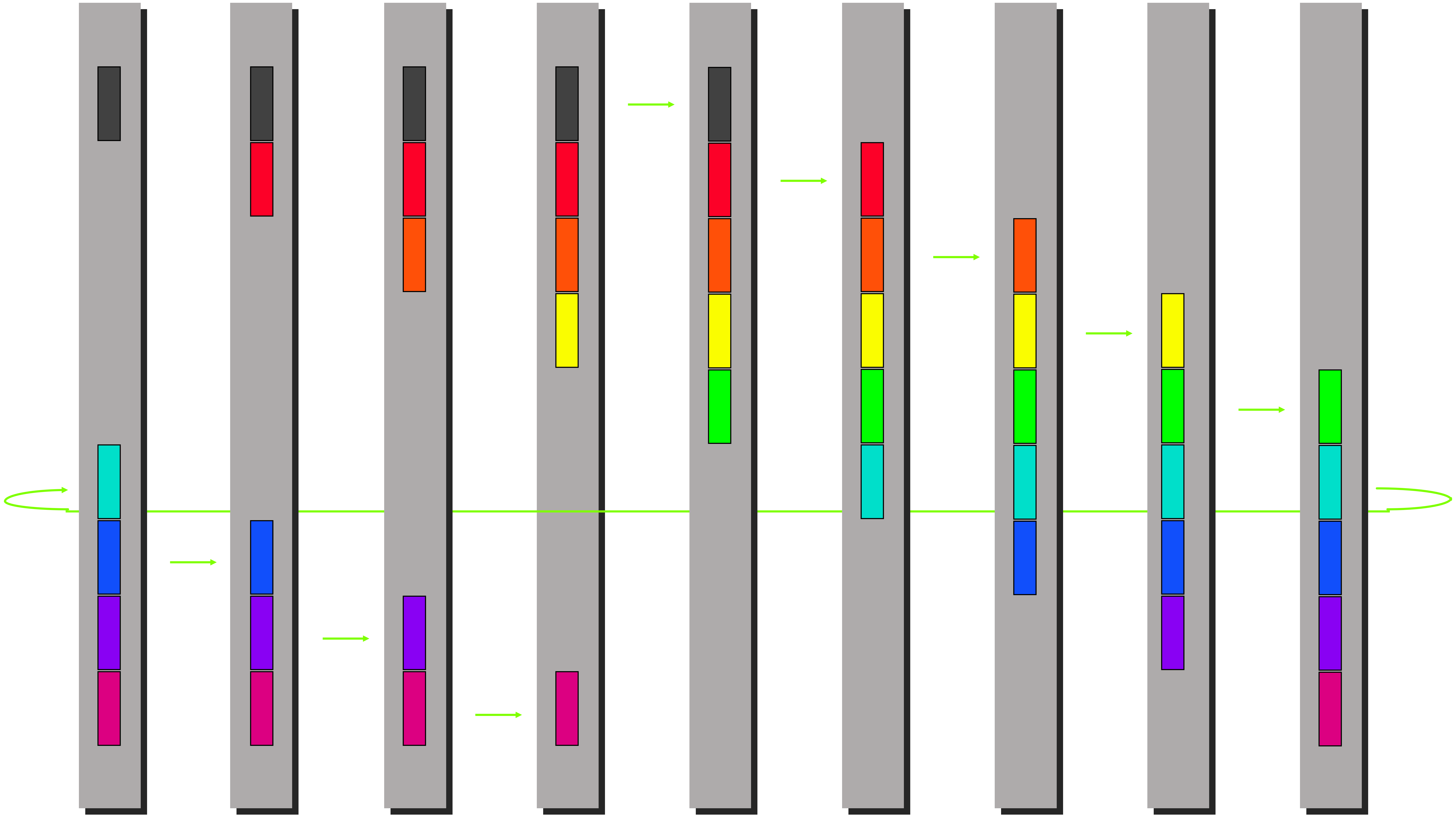


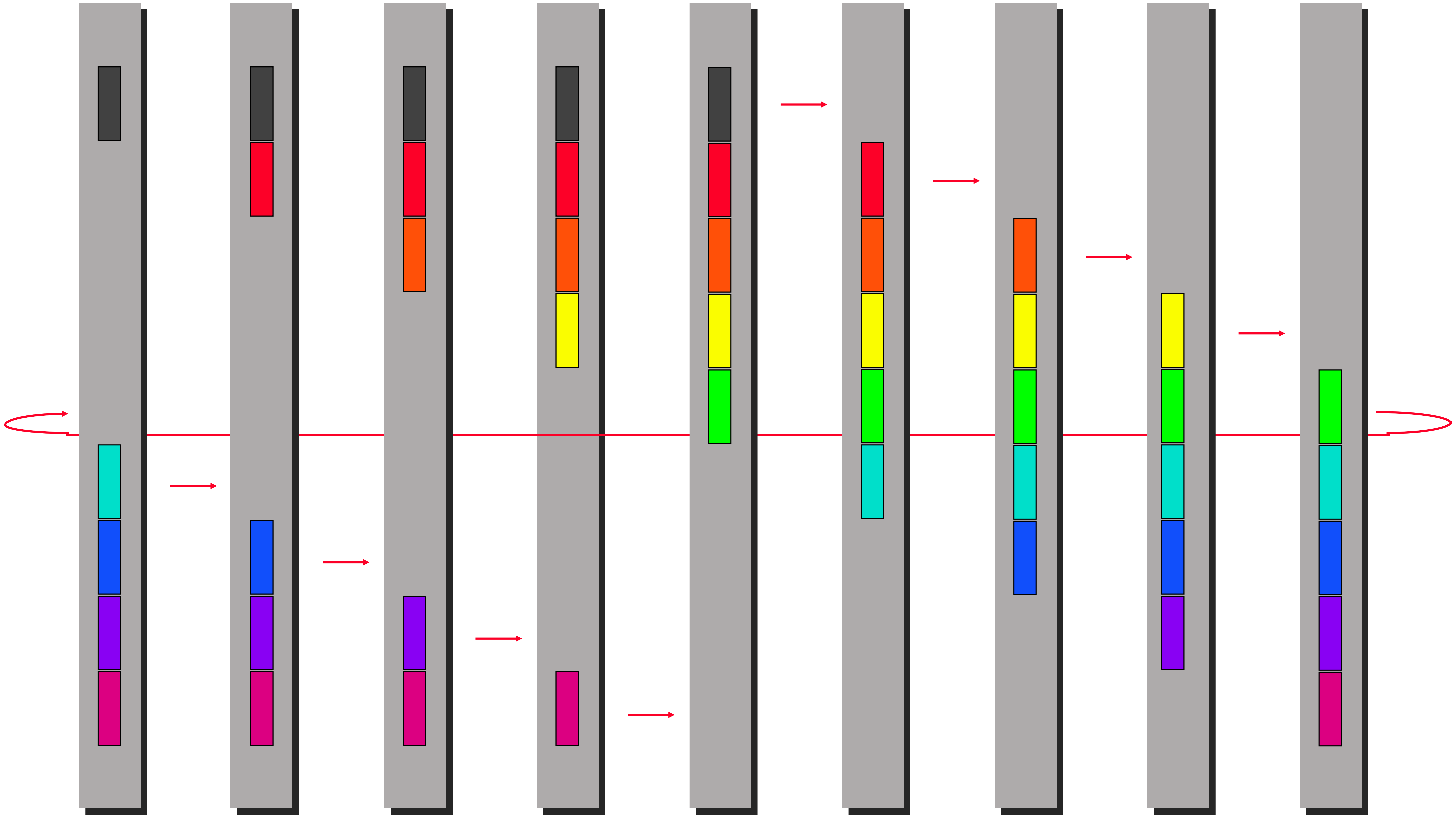


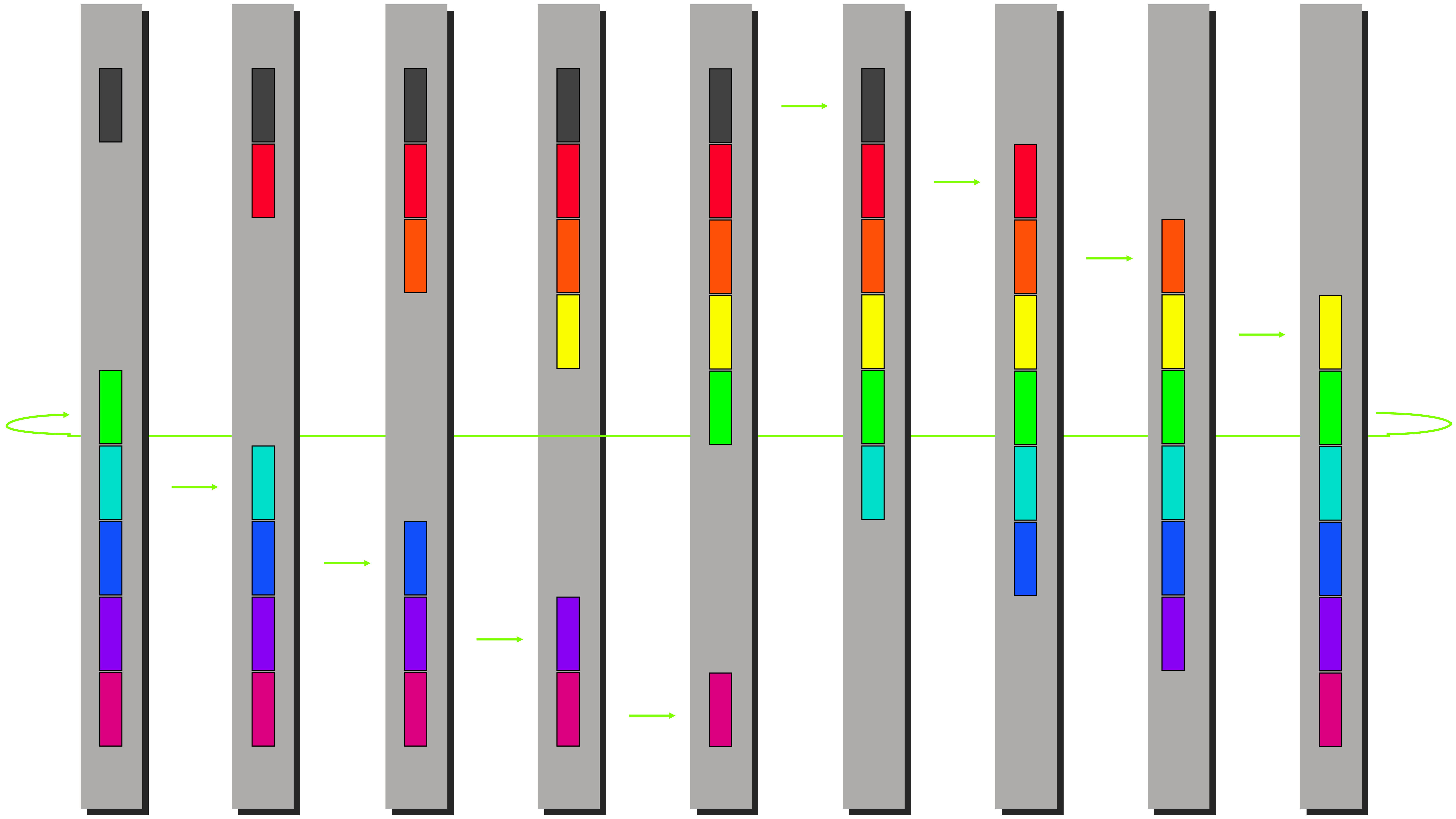


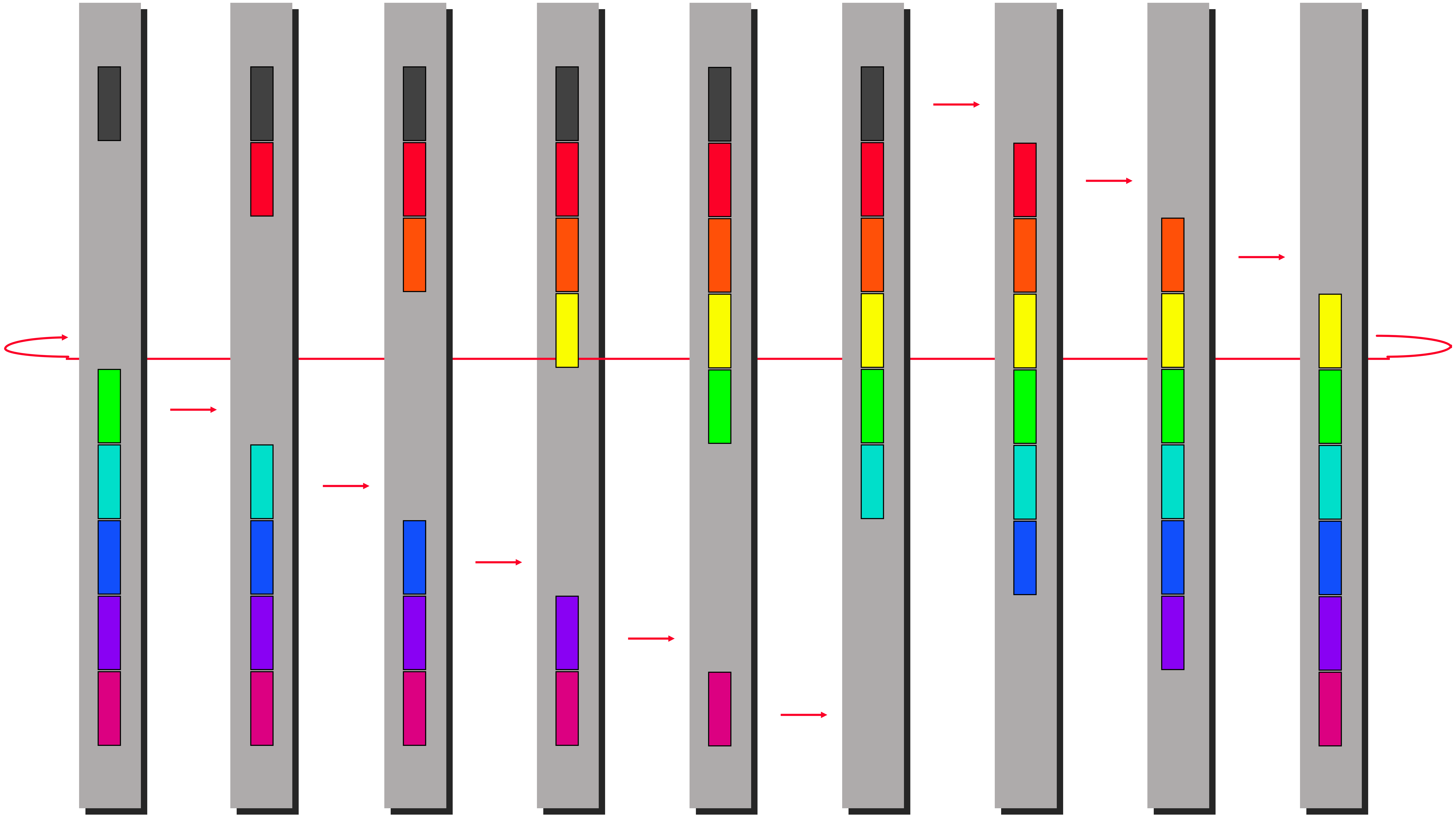


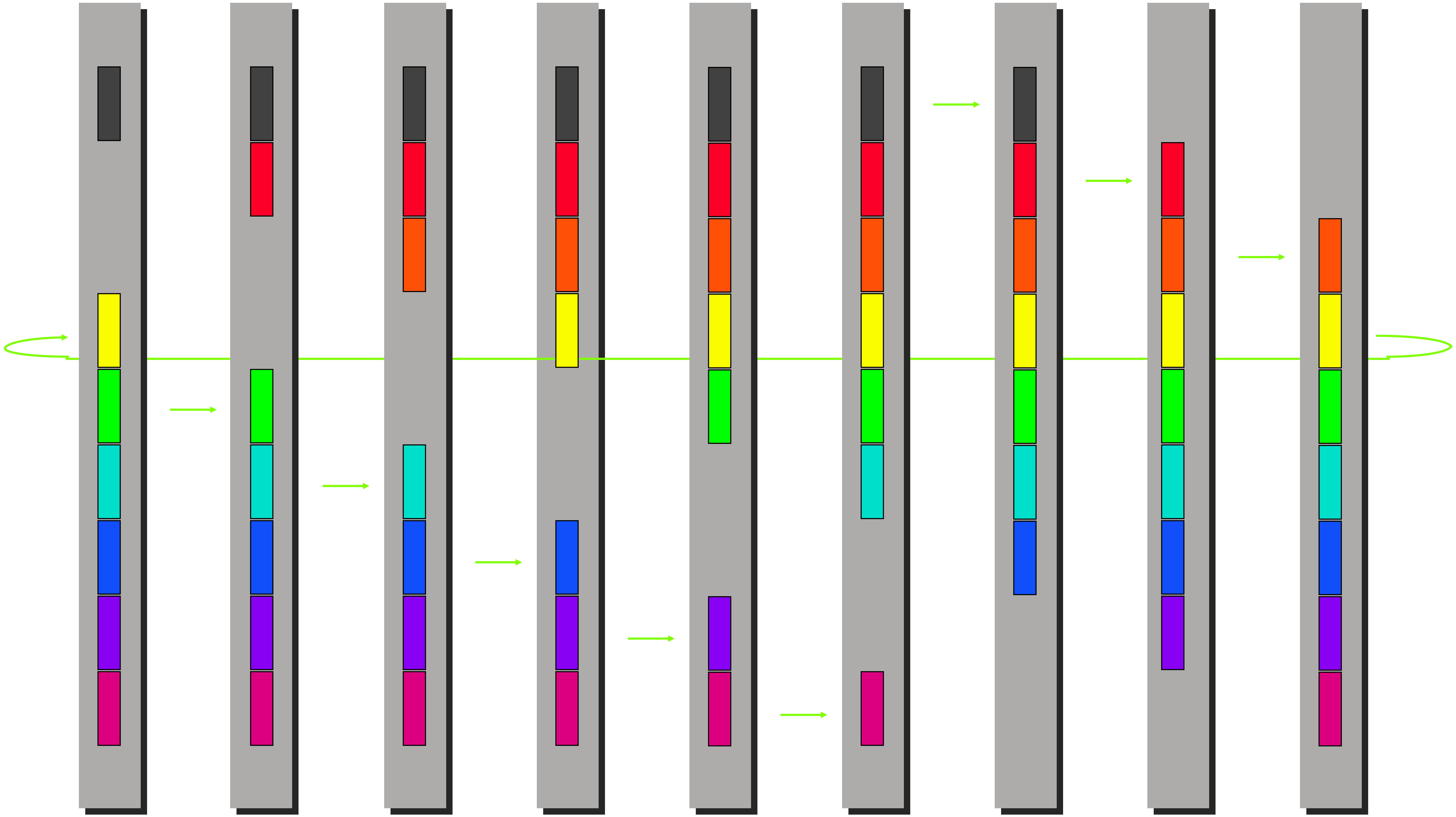


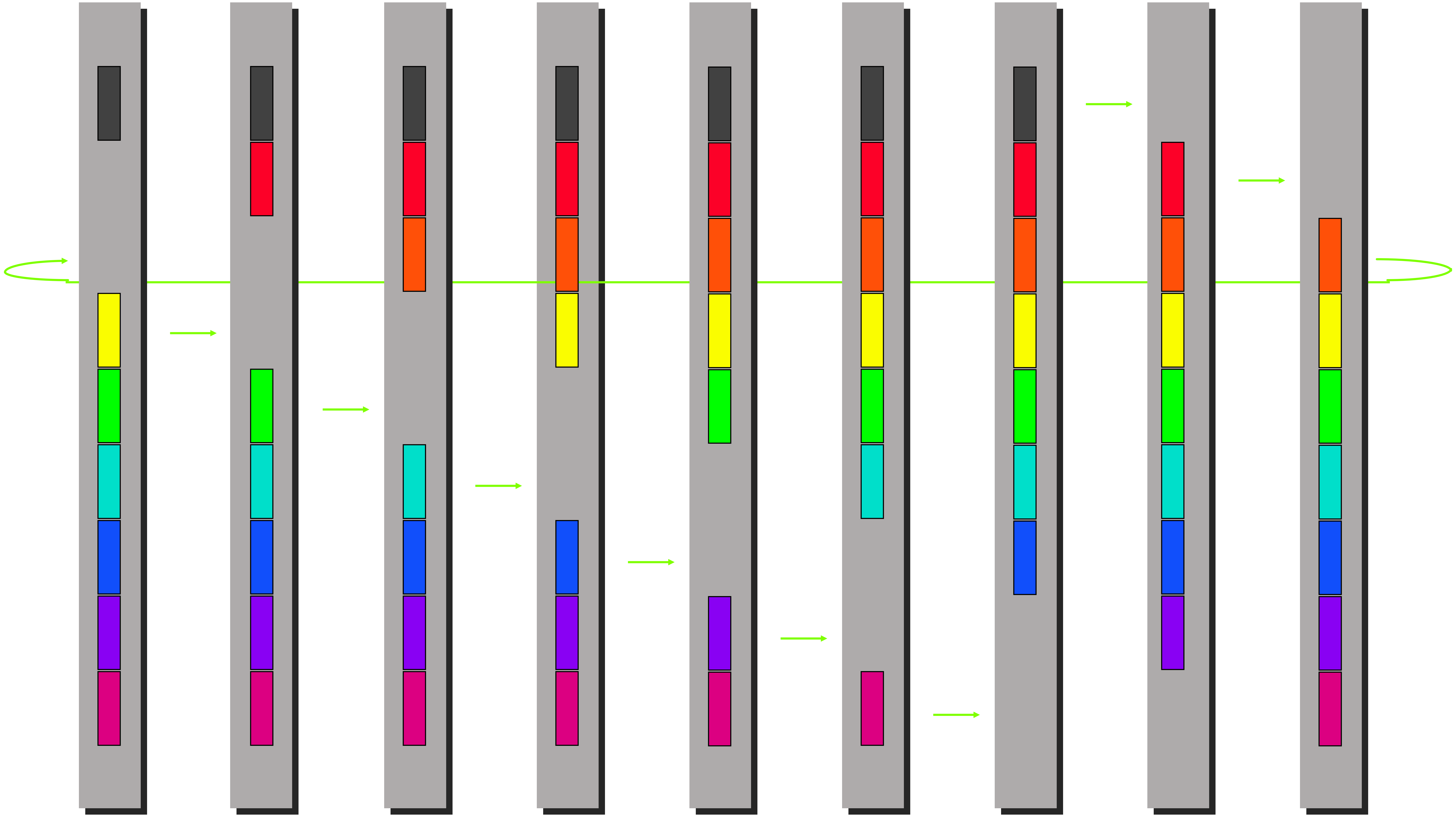


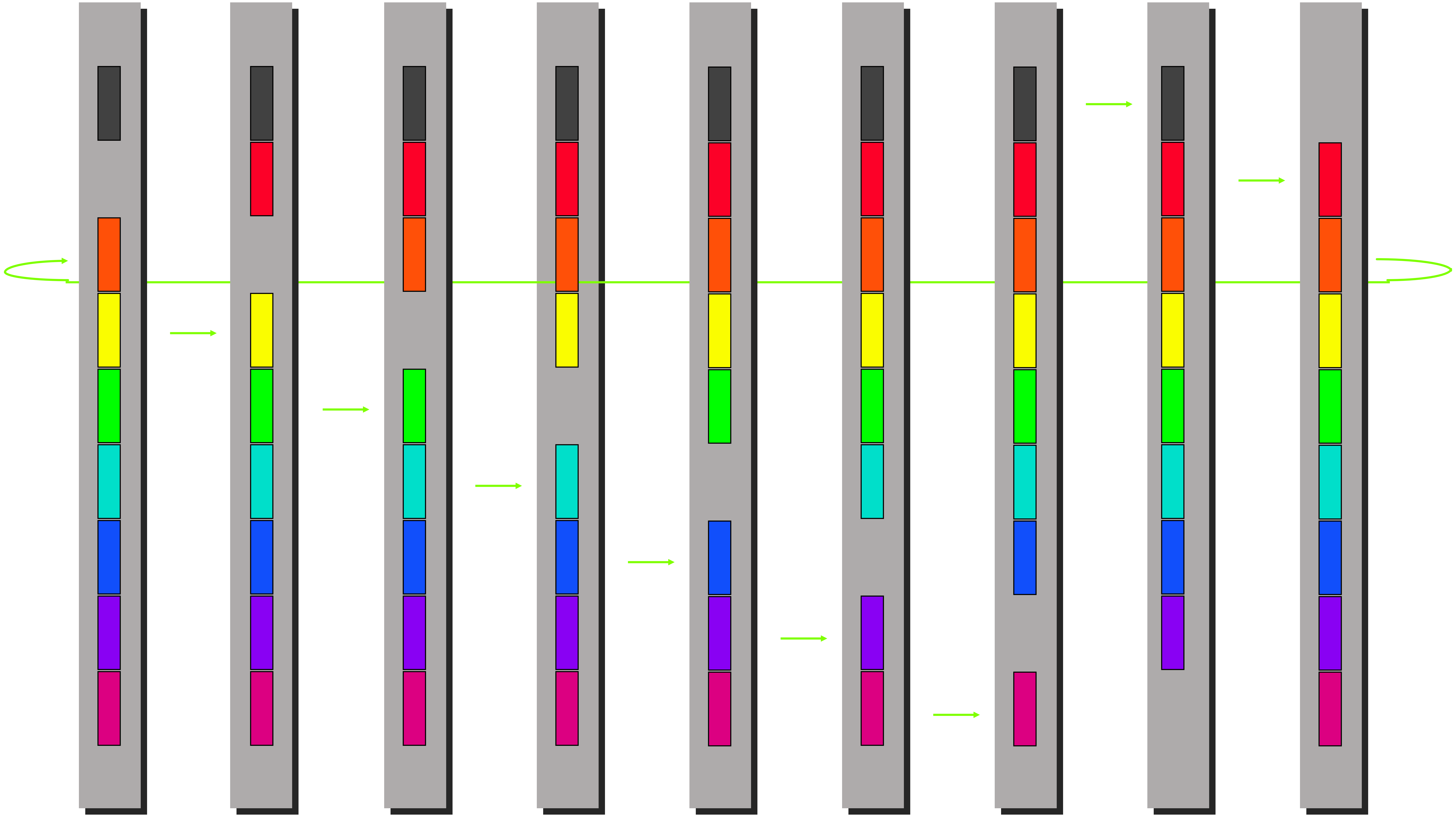


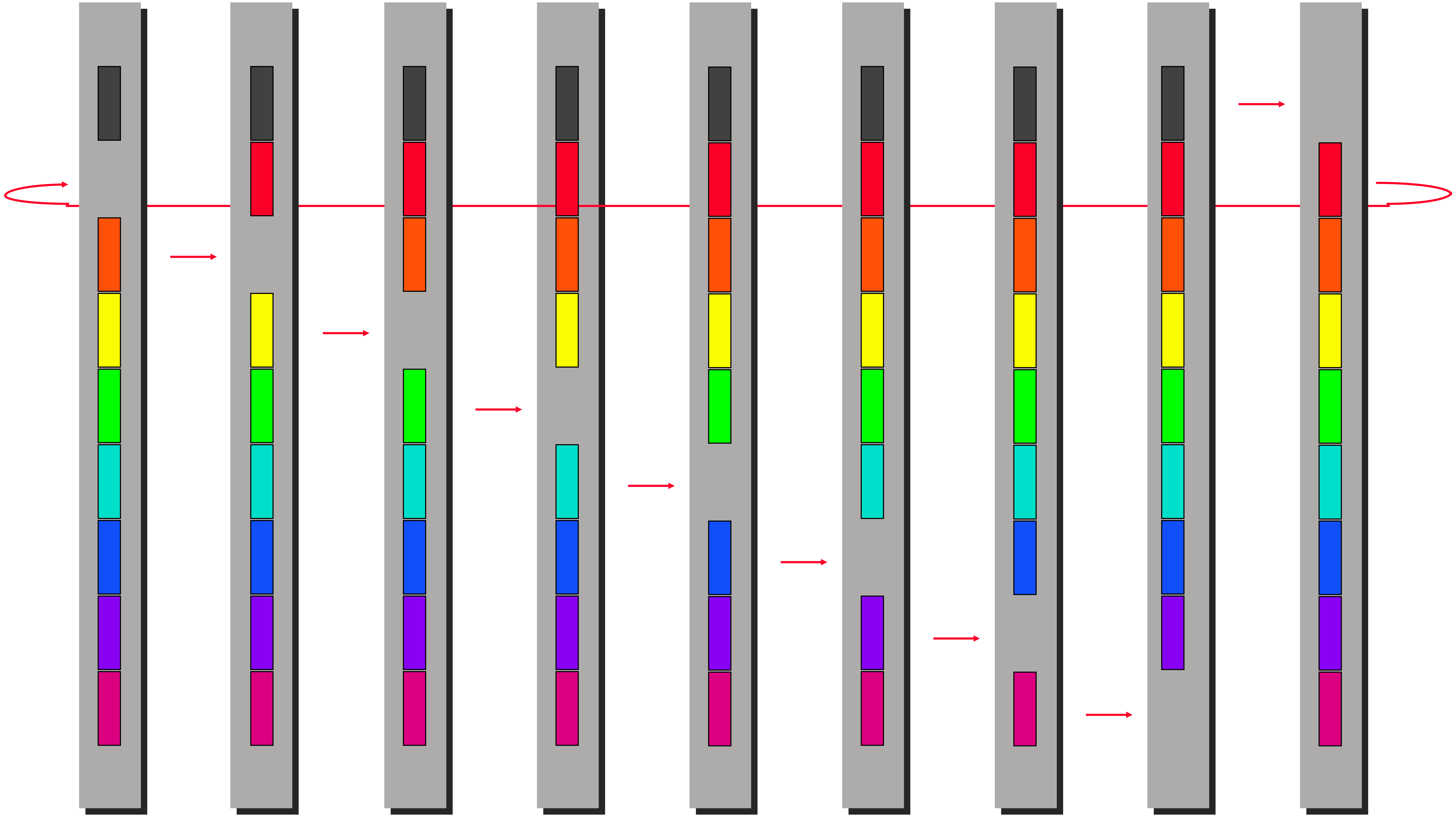


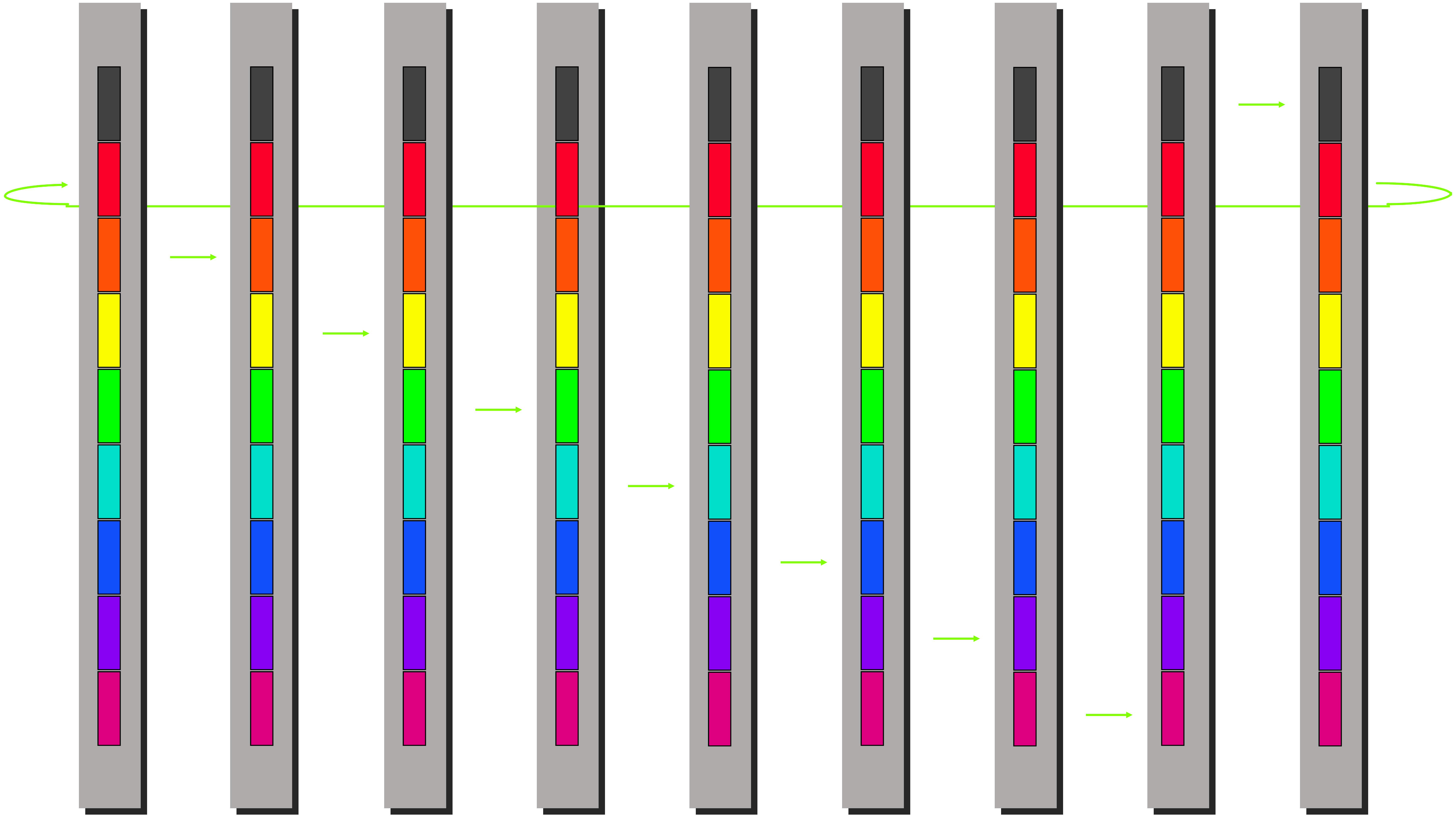


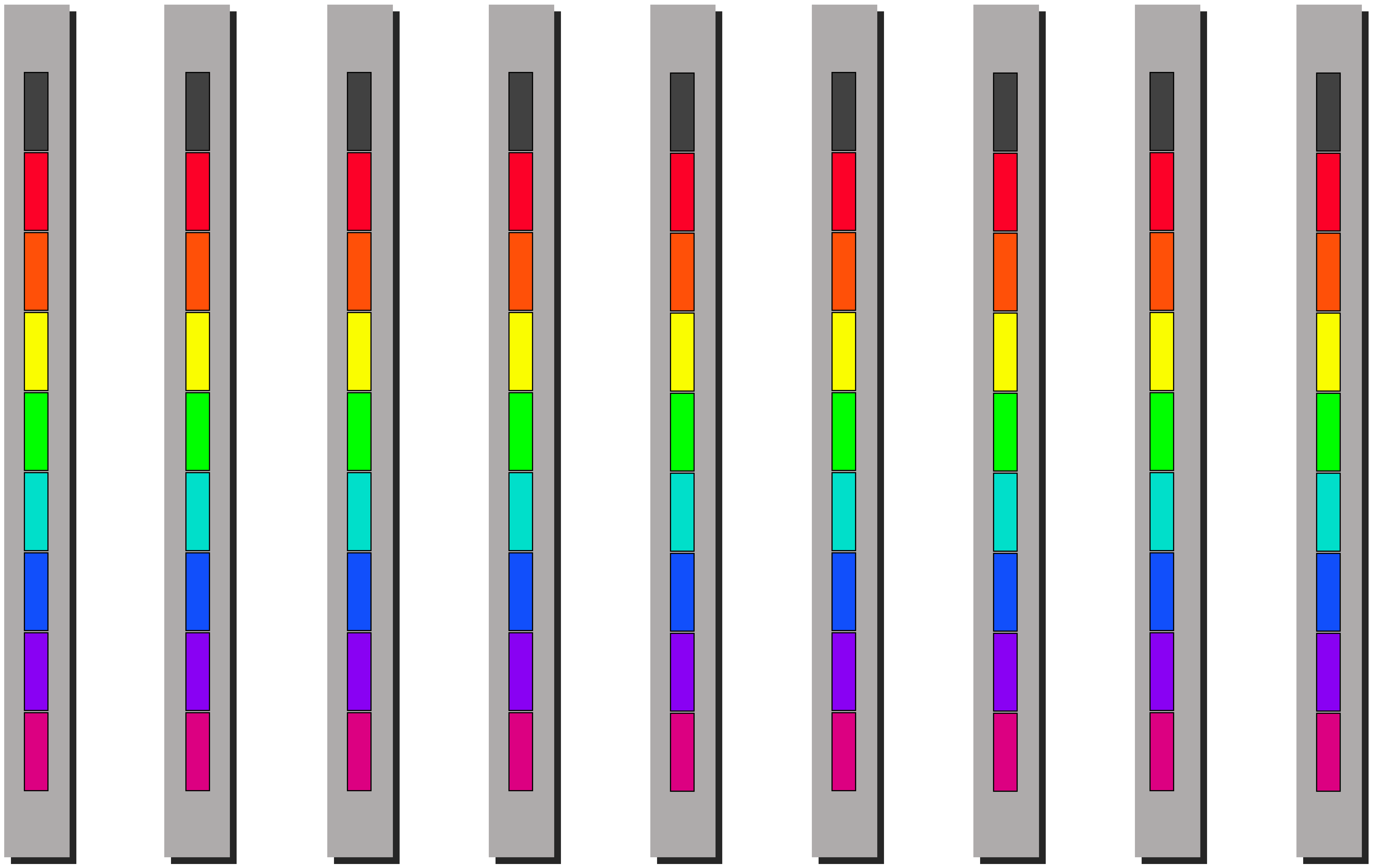




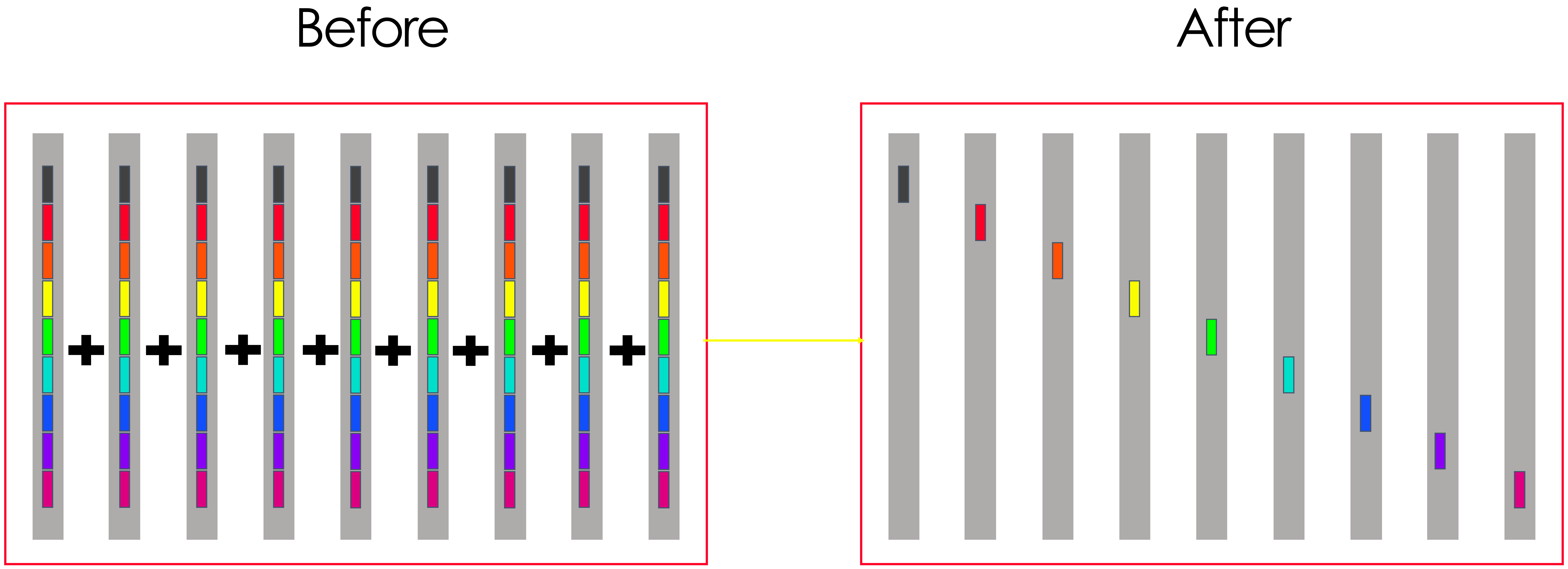


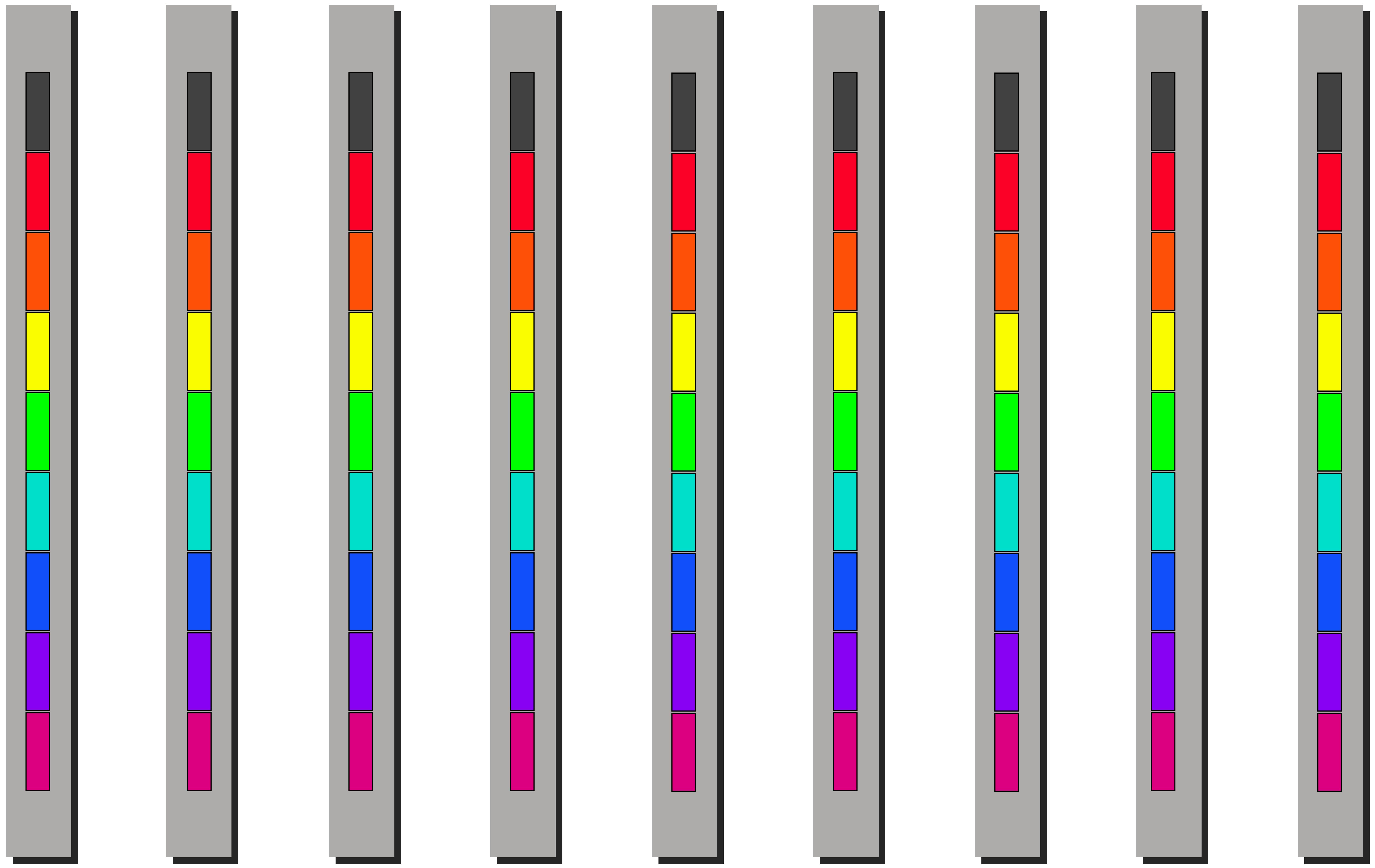


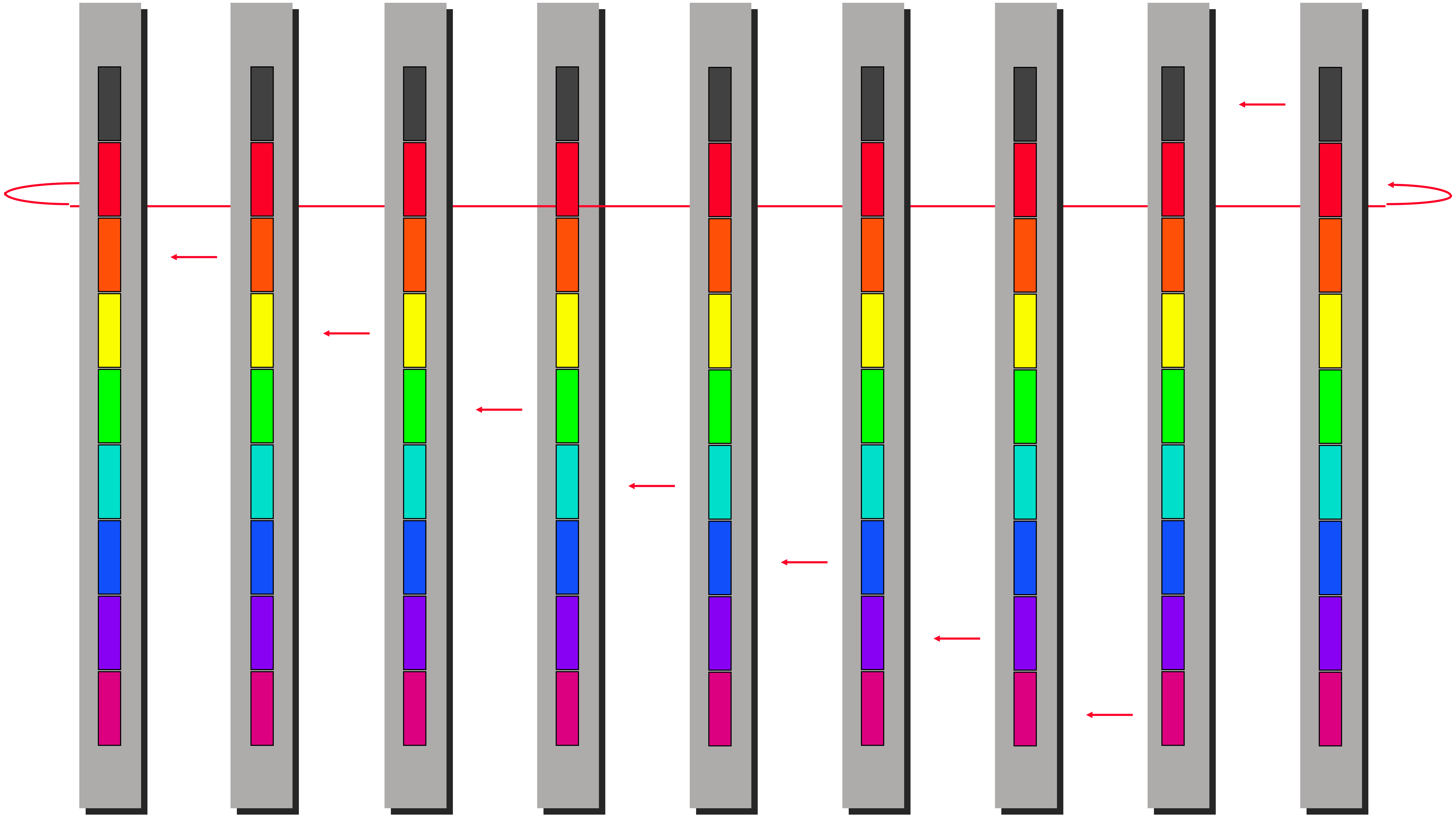


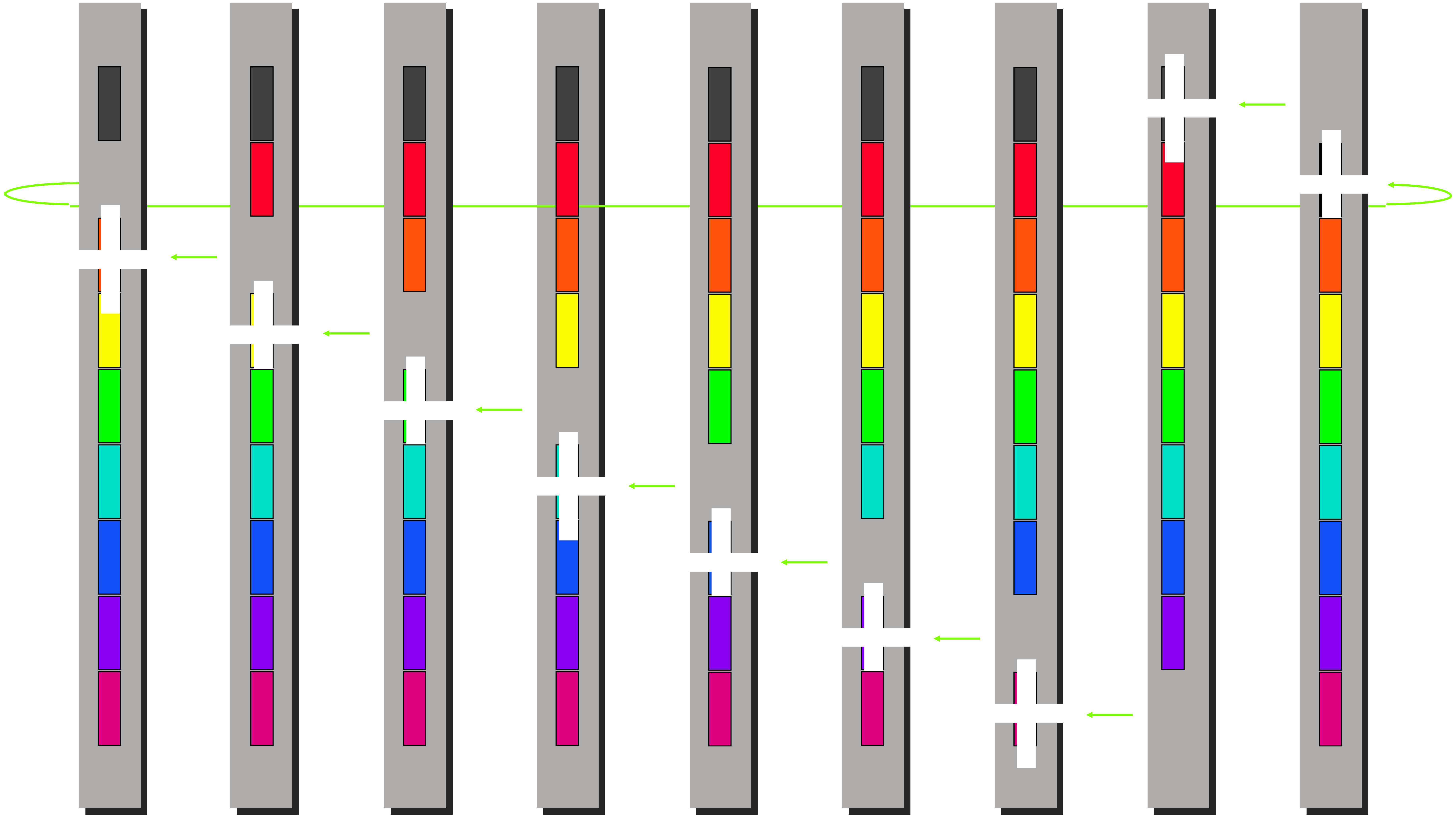


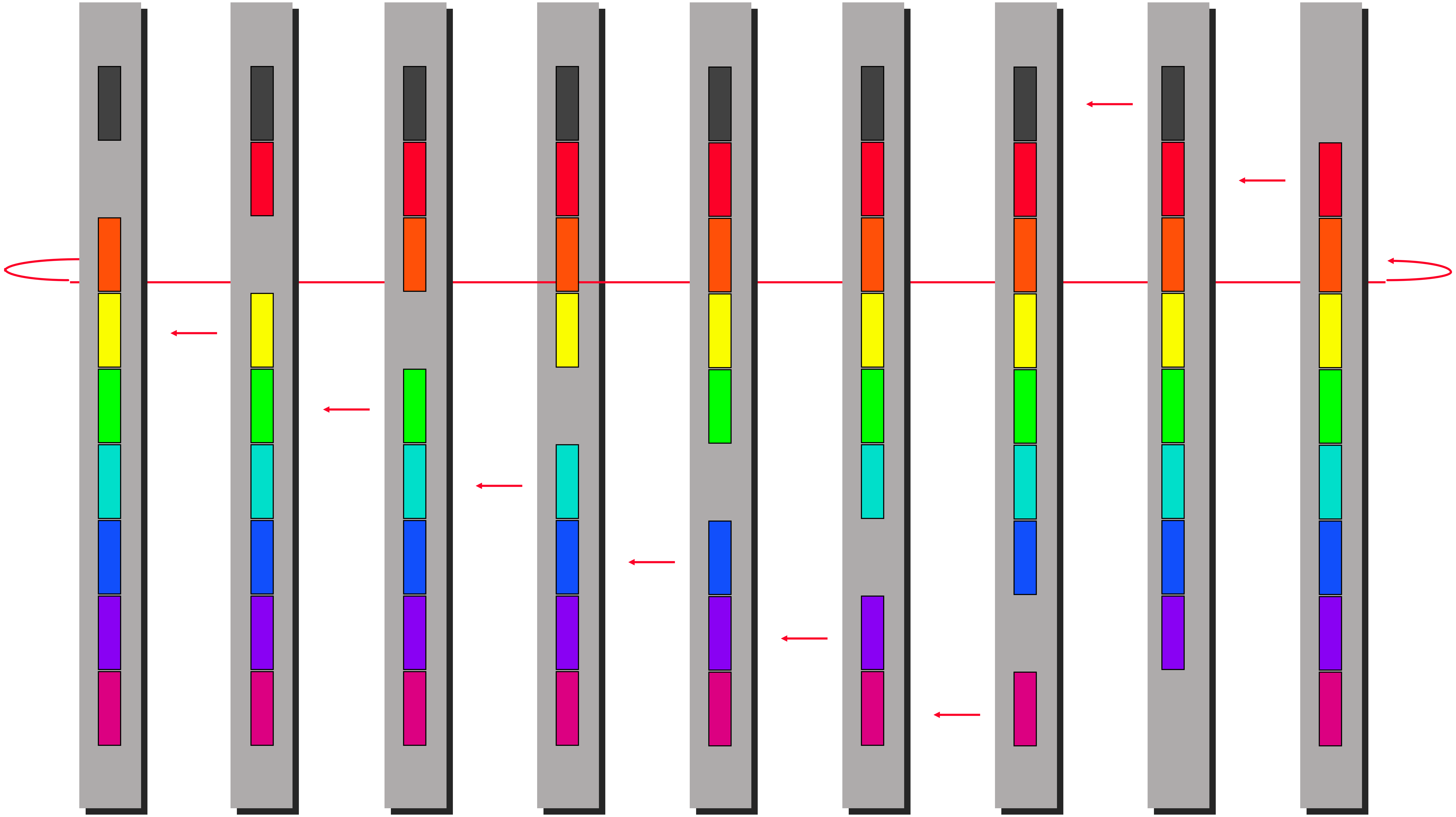
Reduce-scatter

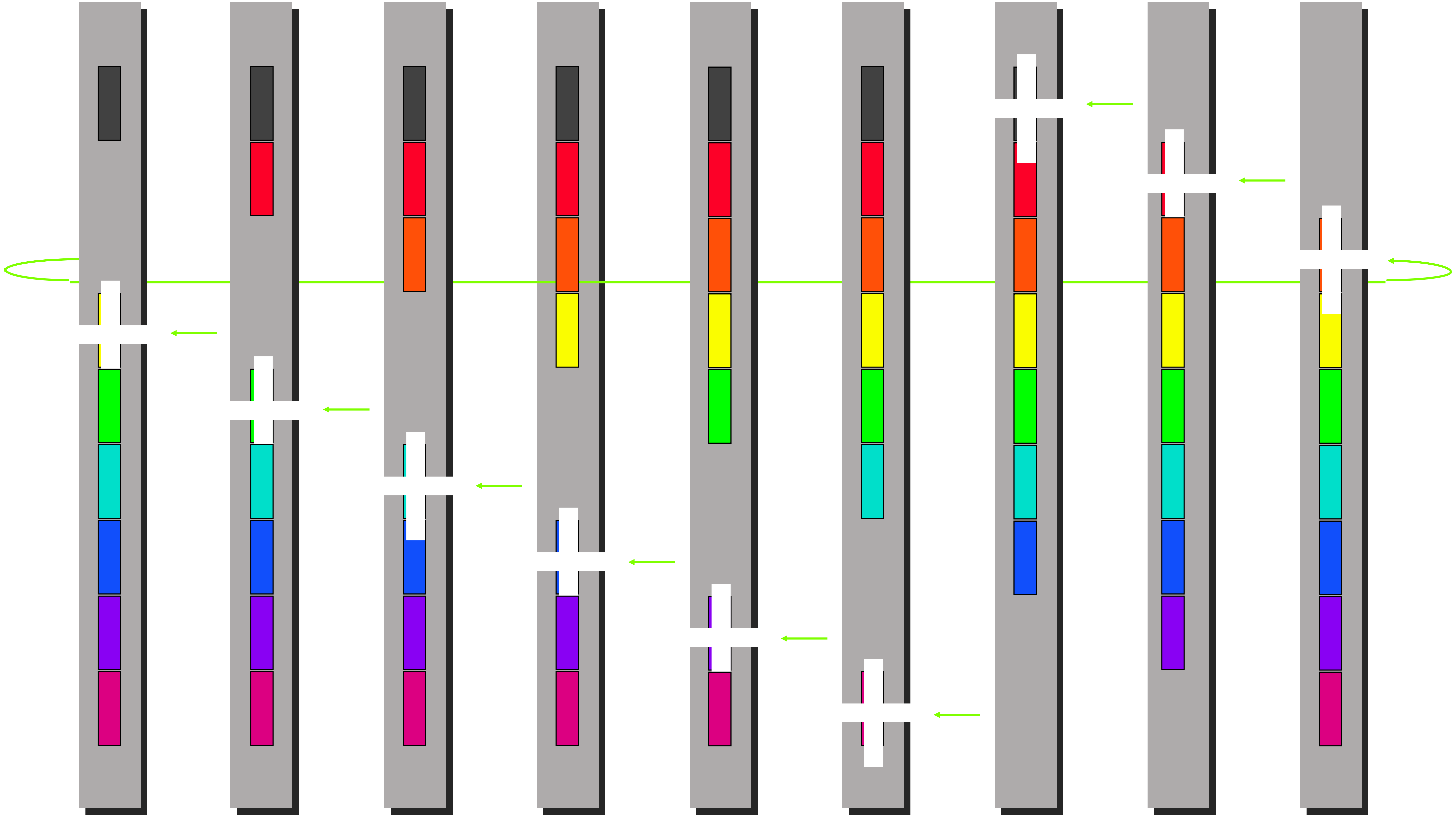


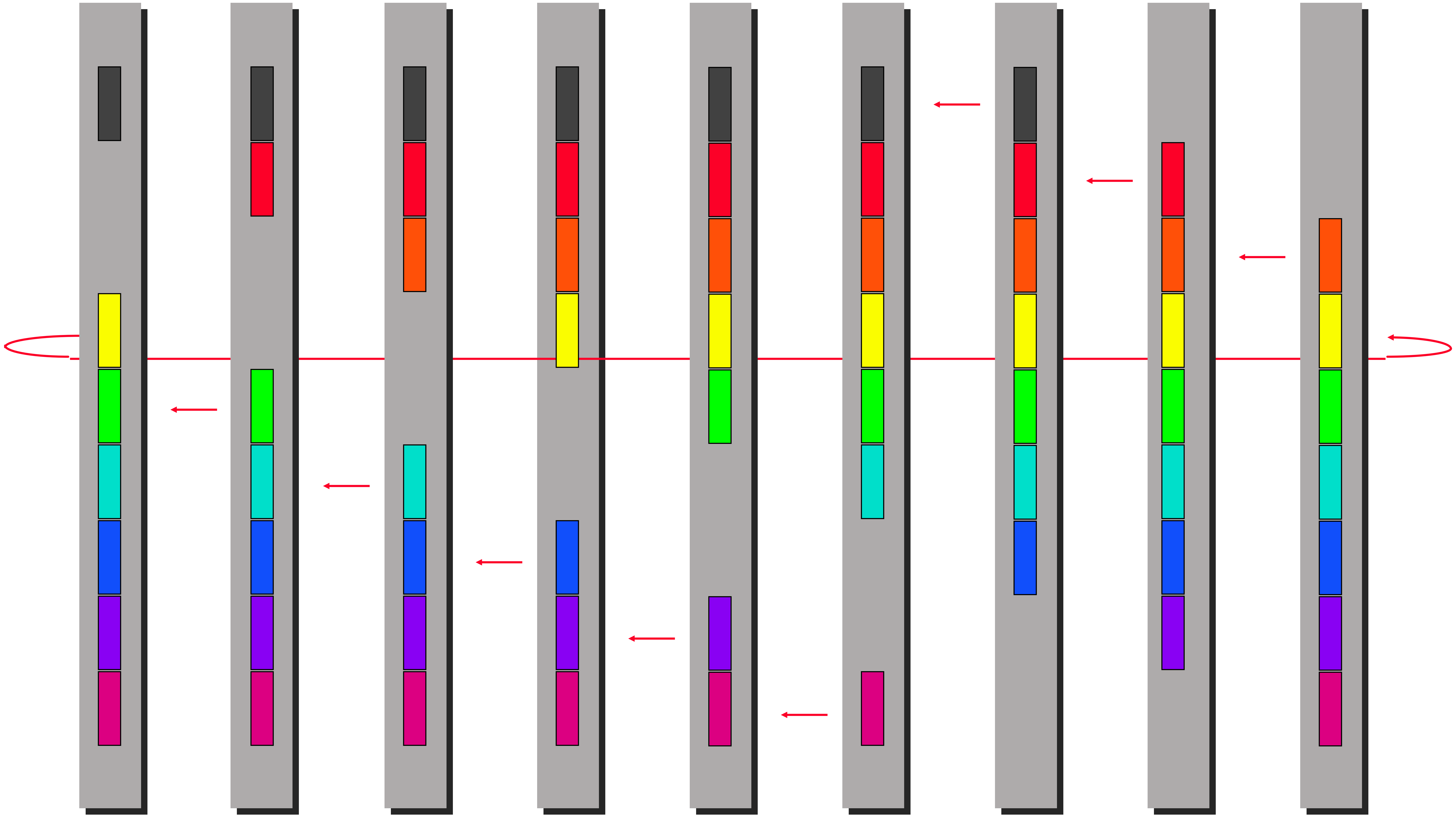


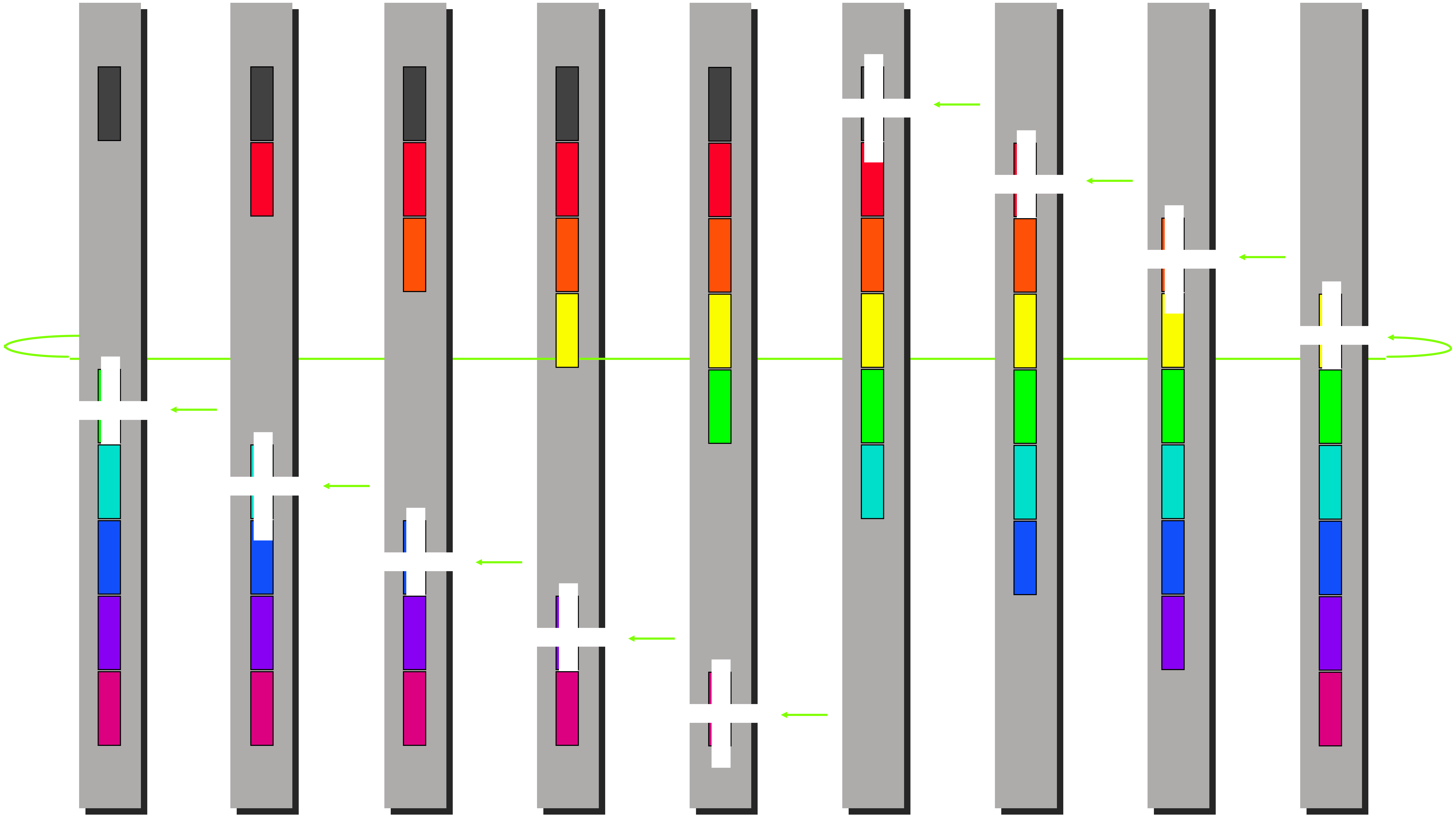


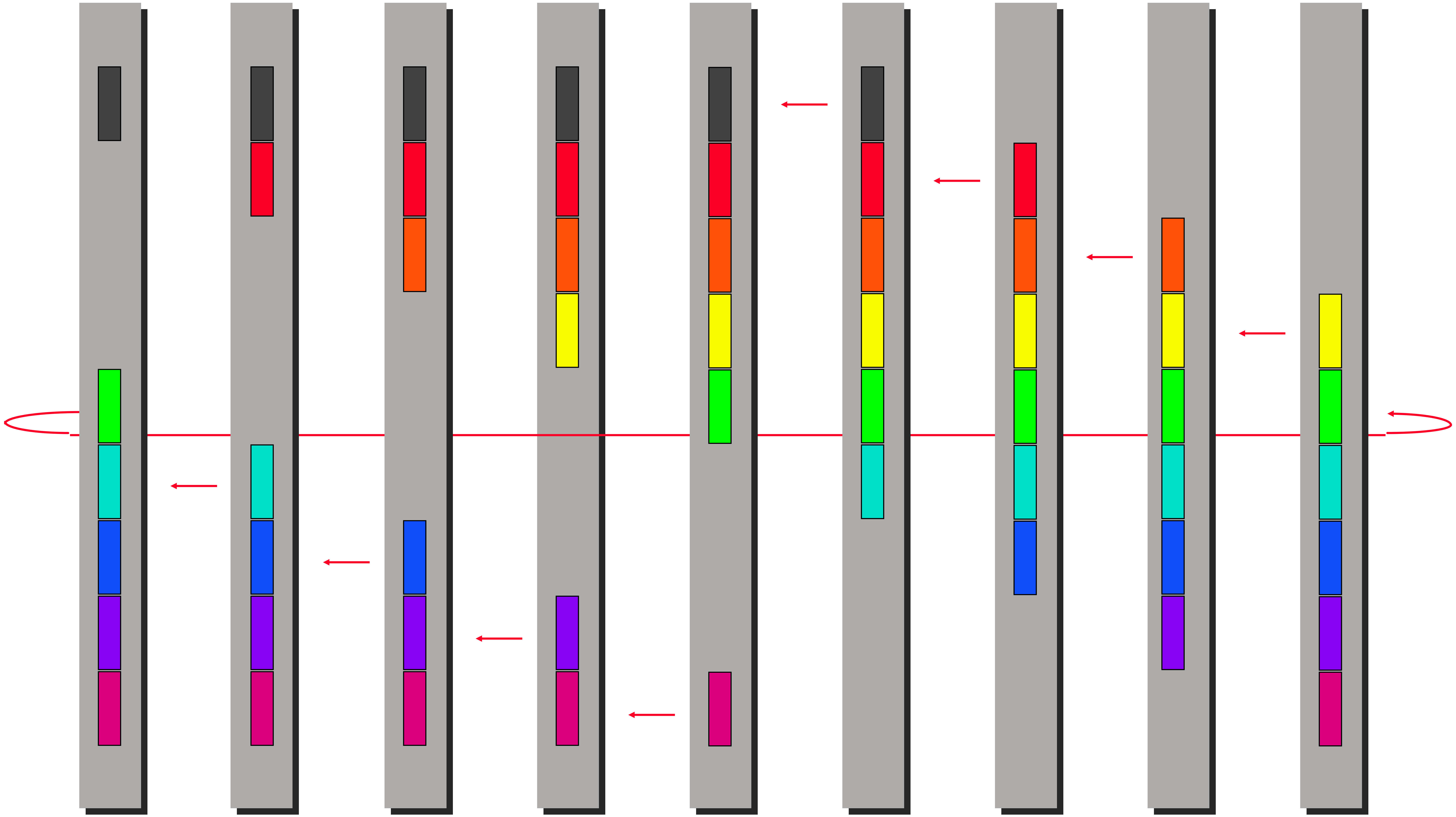


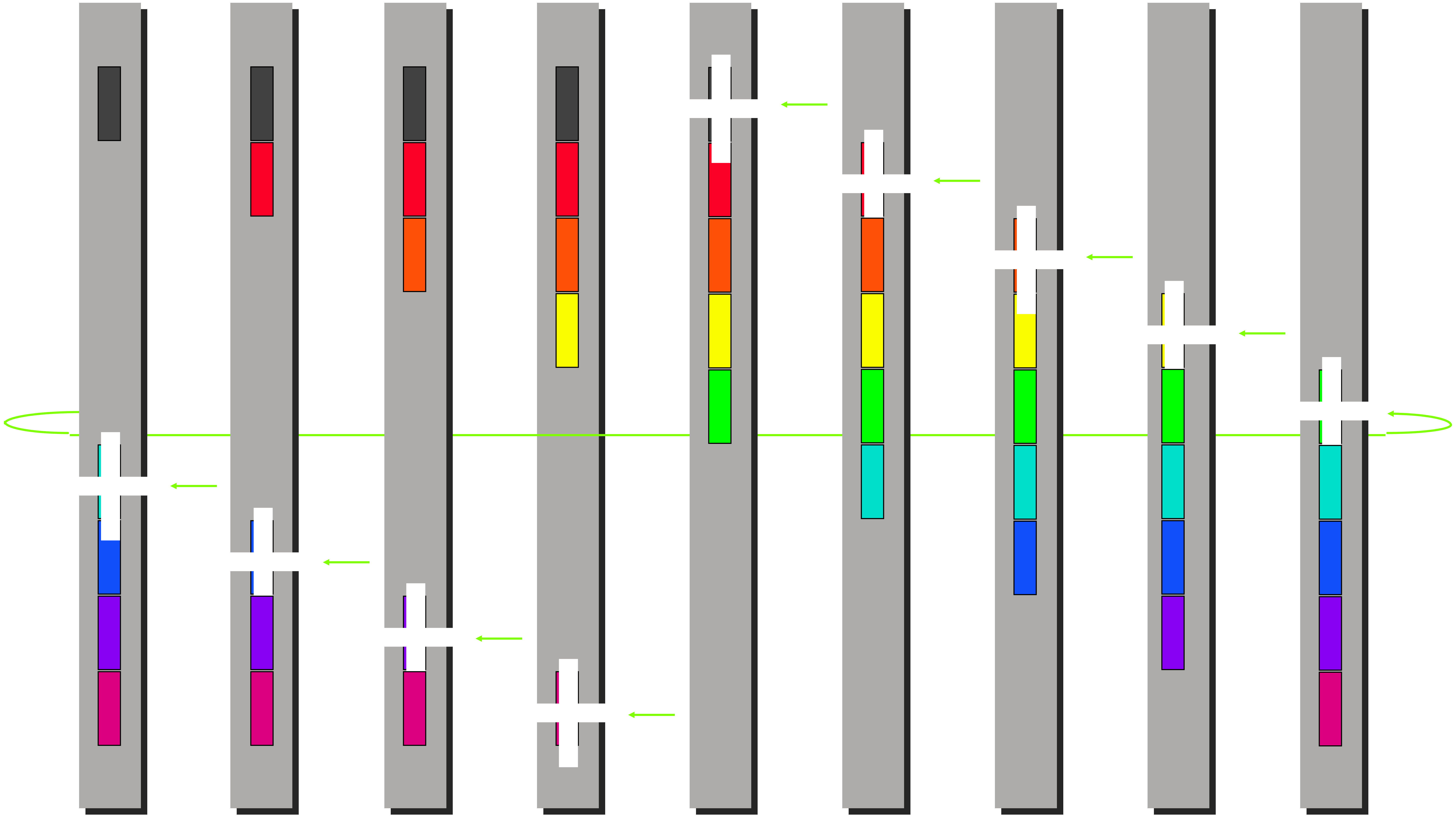


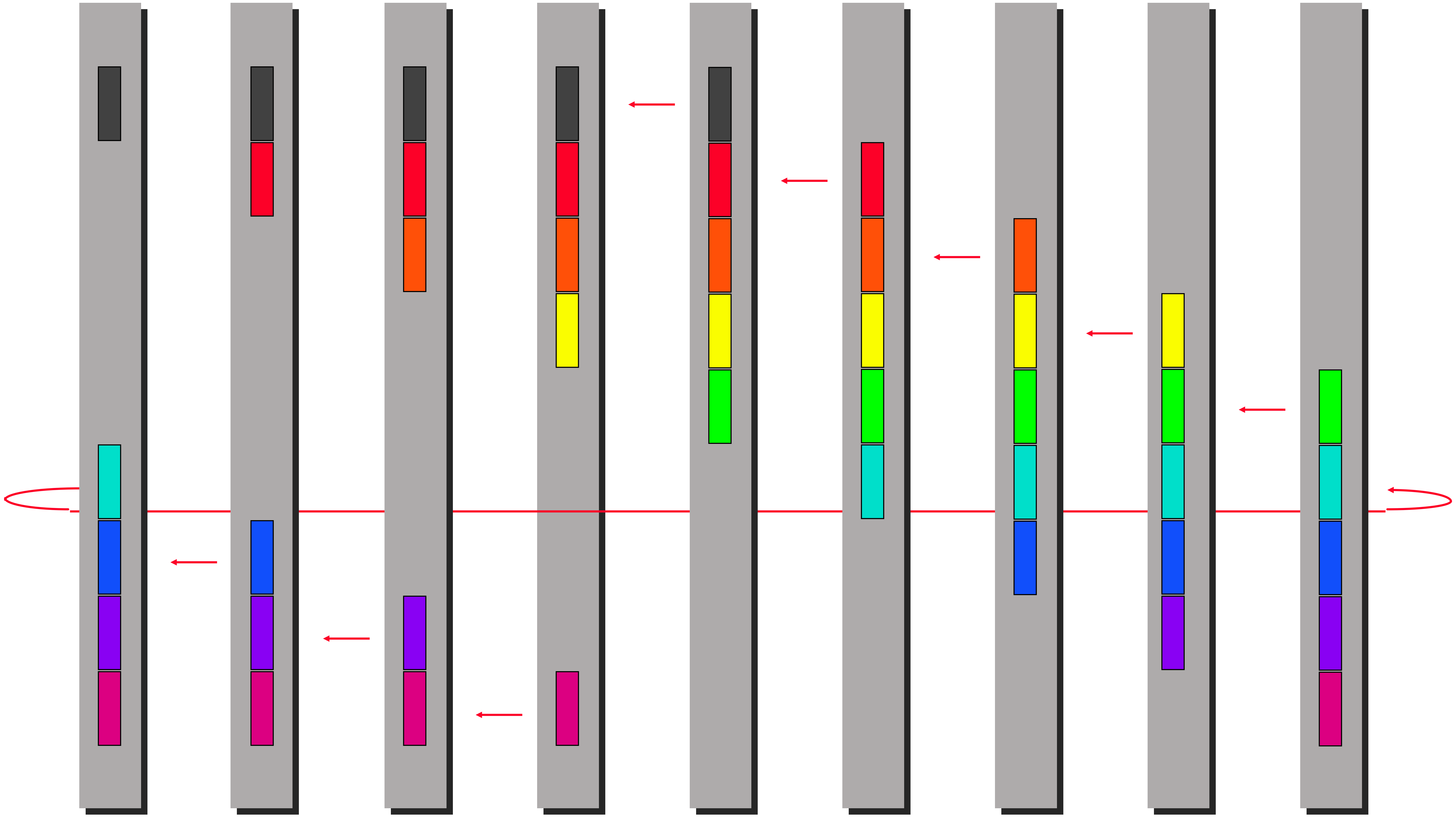


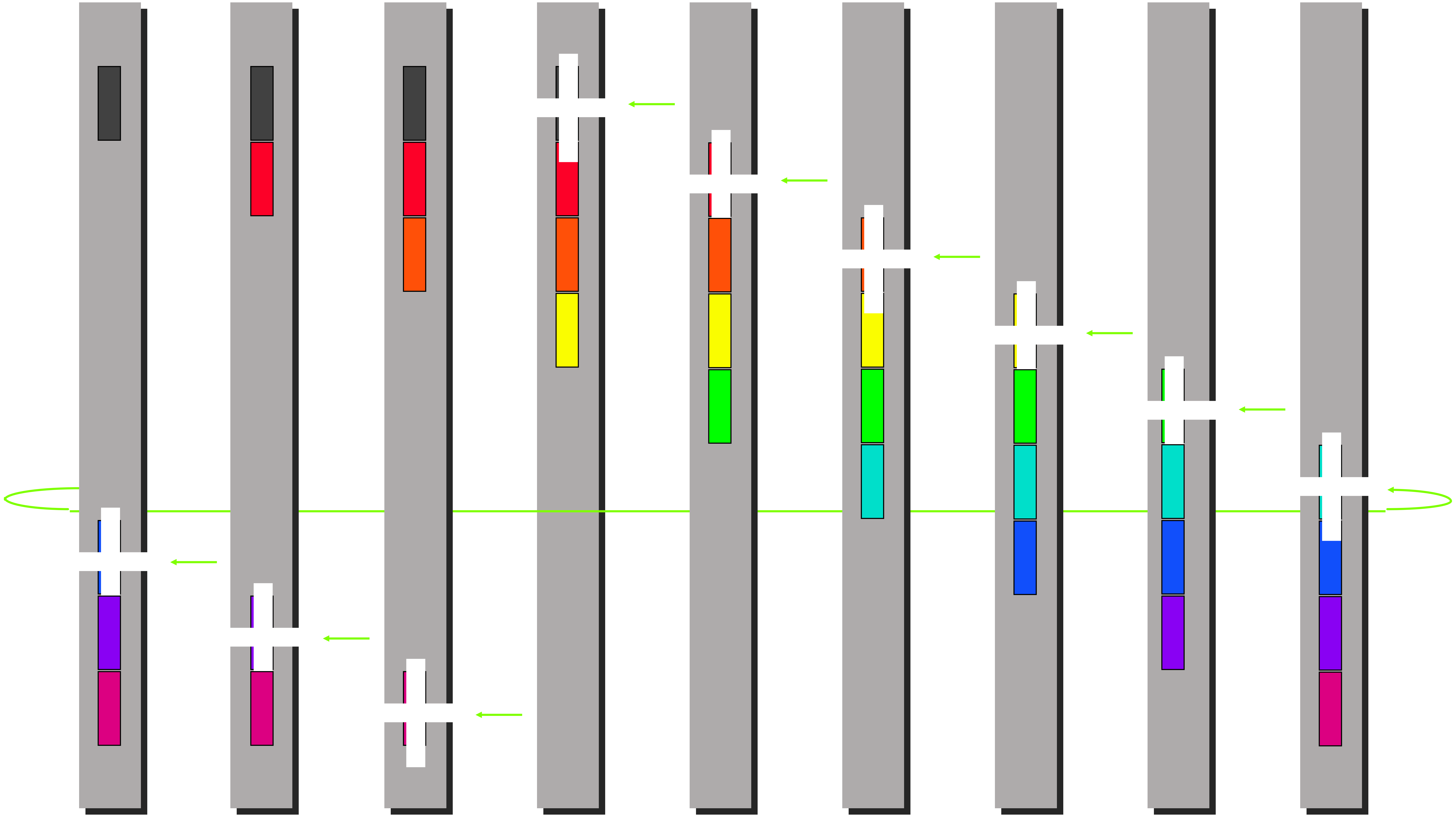


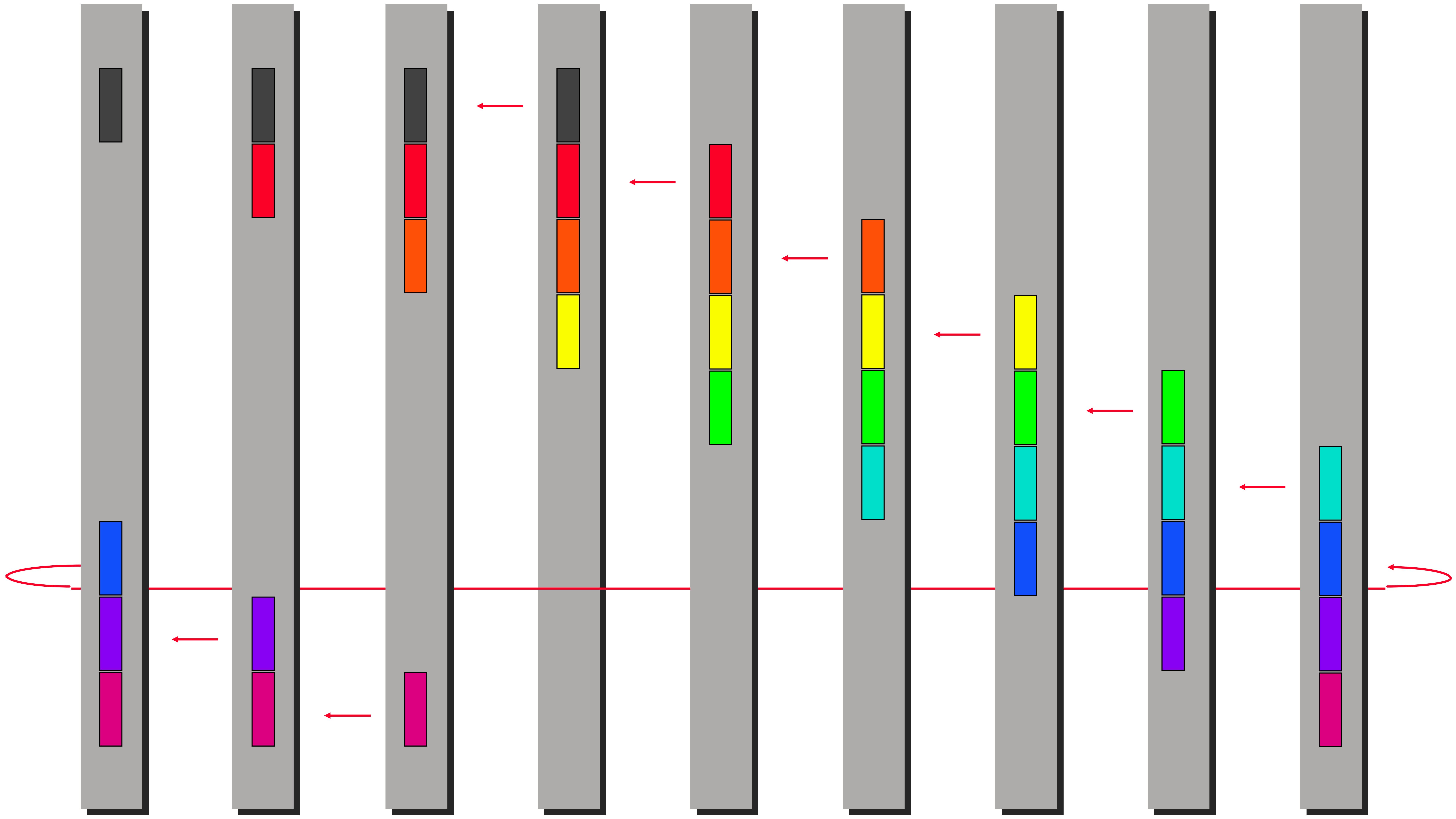


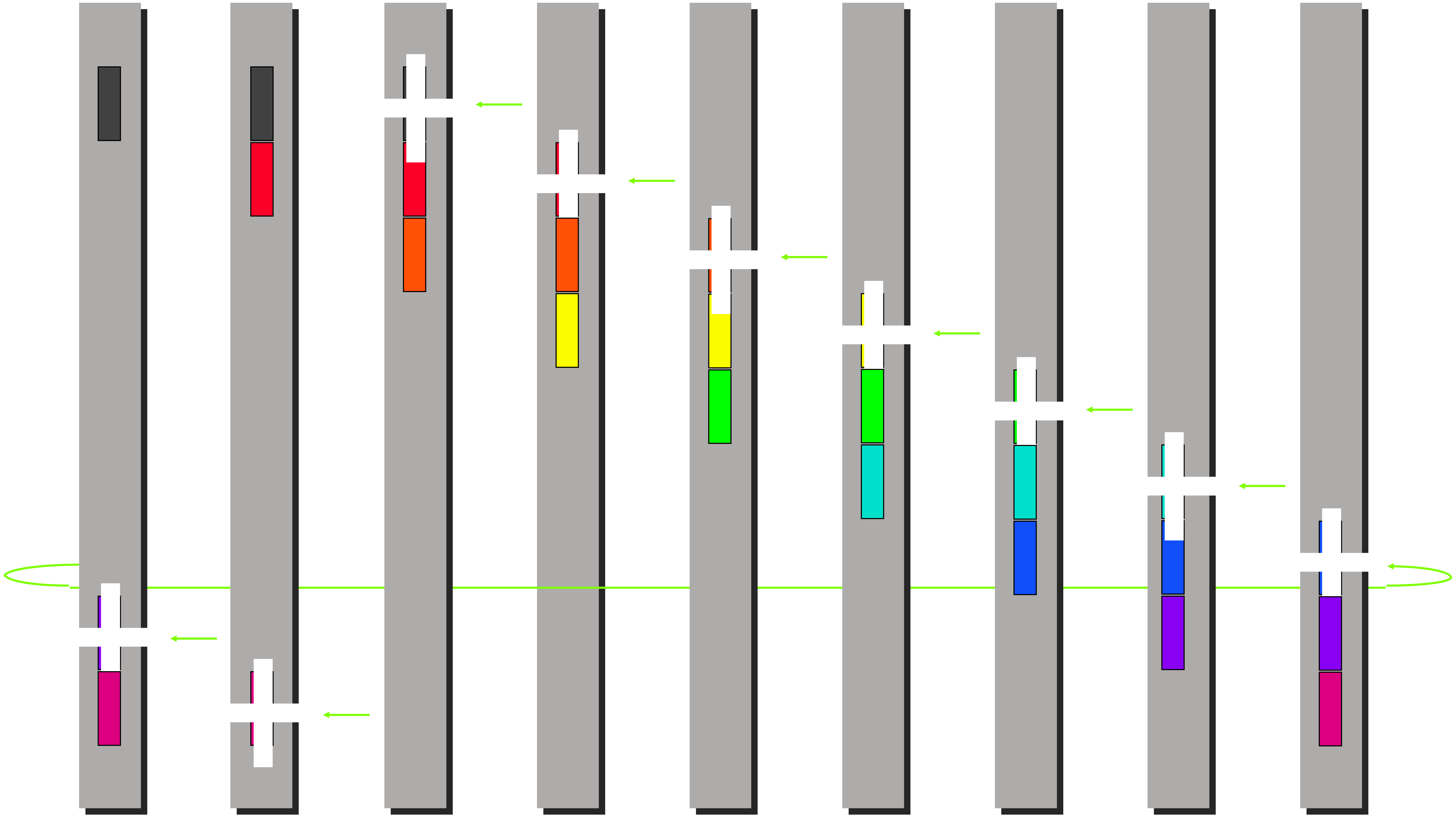


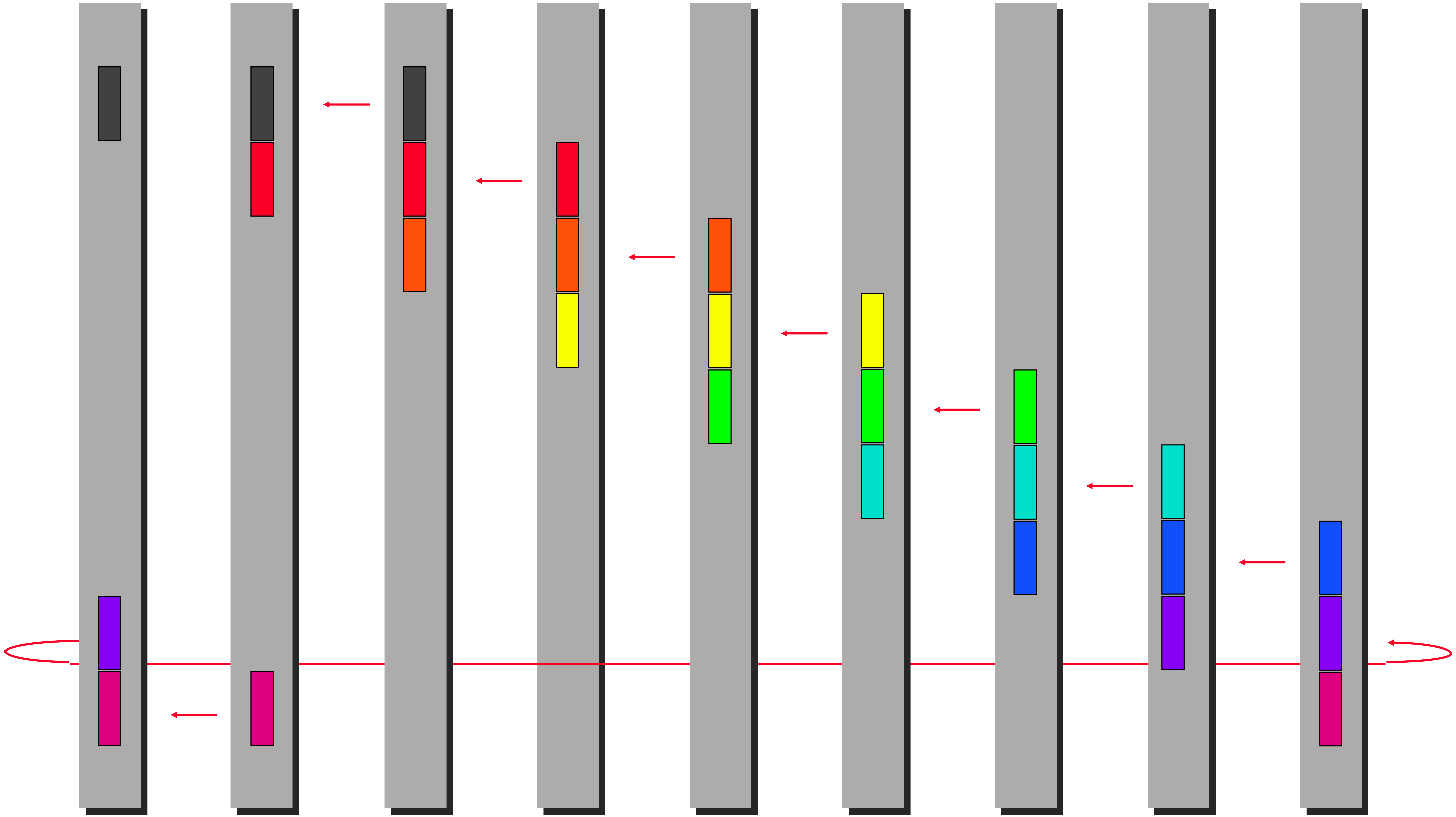


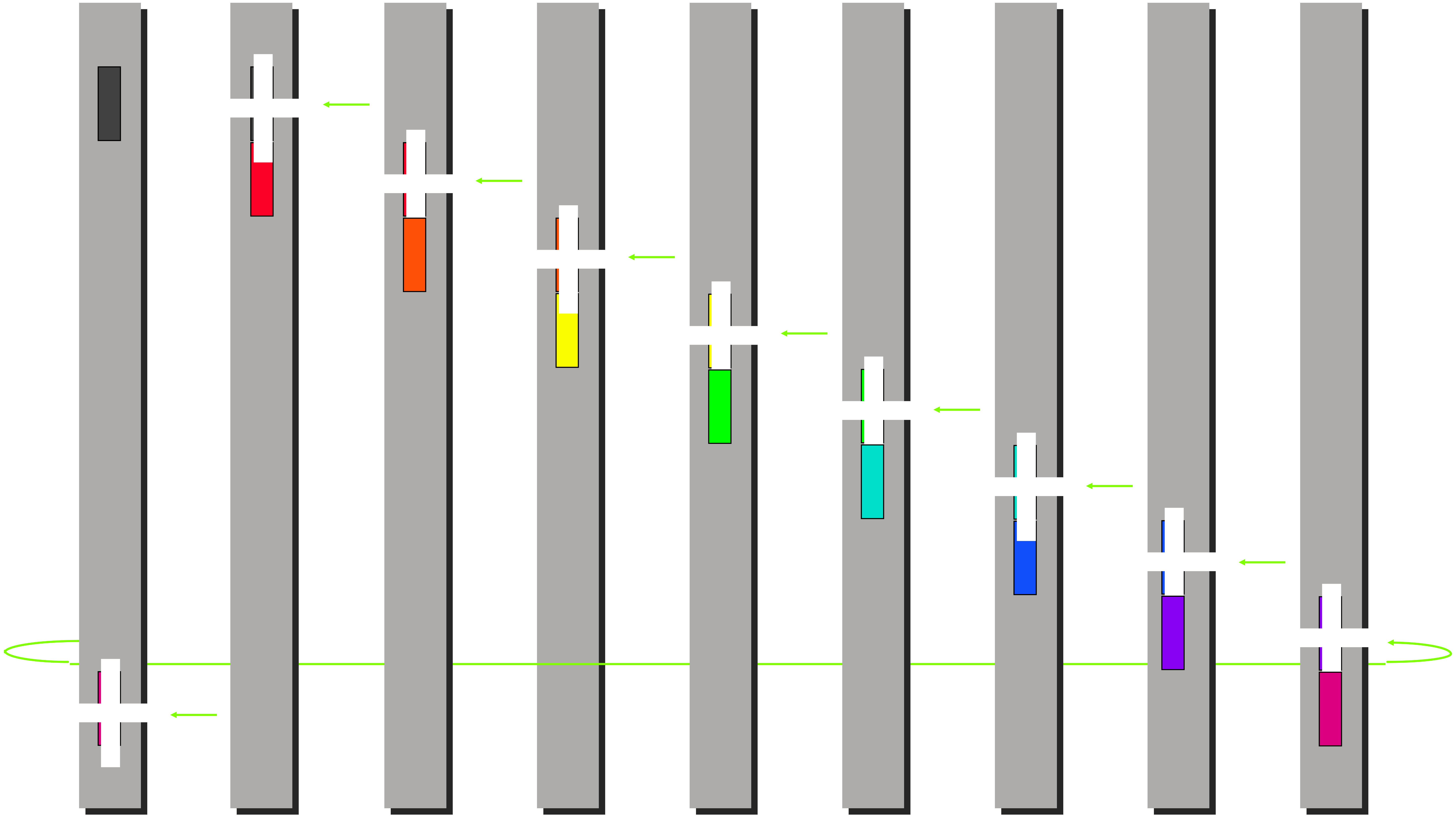


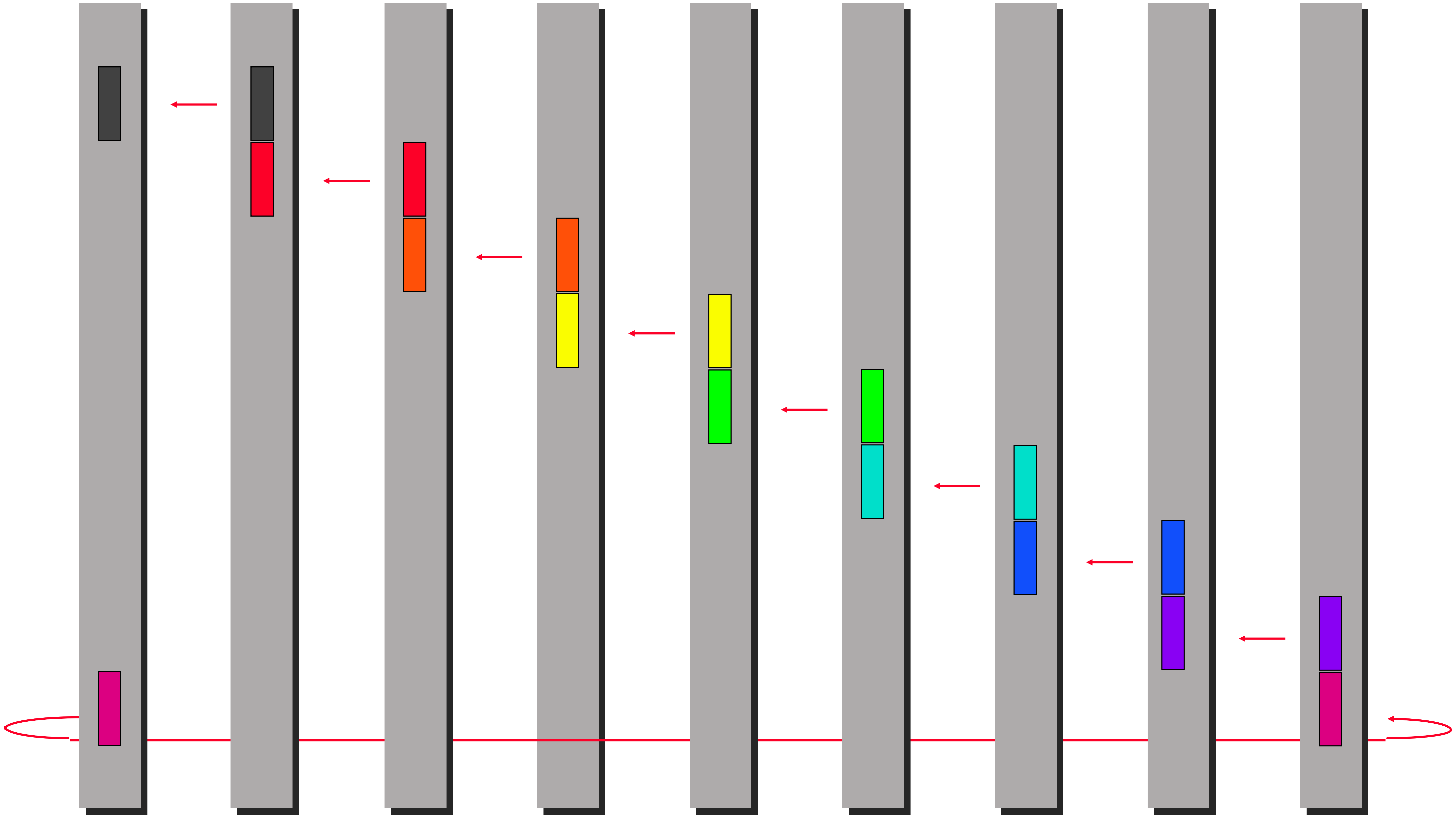


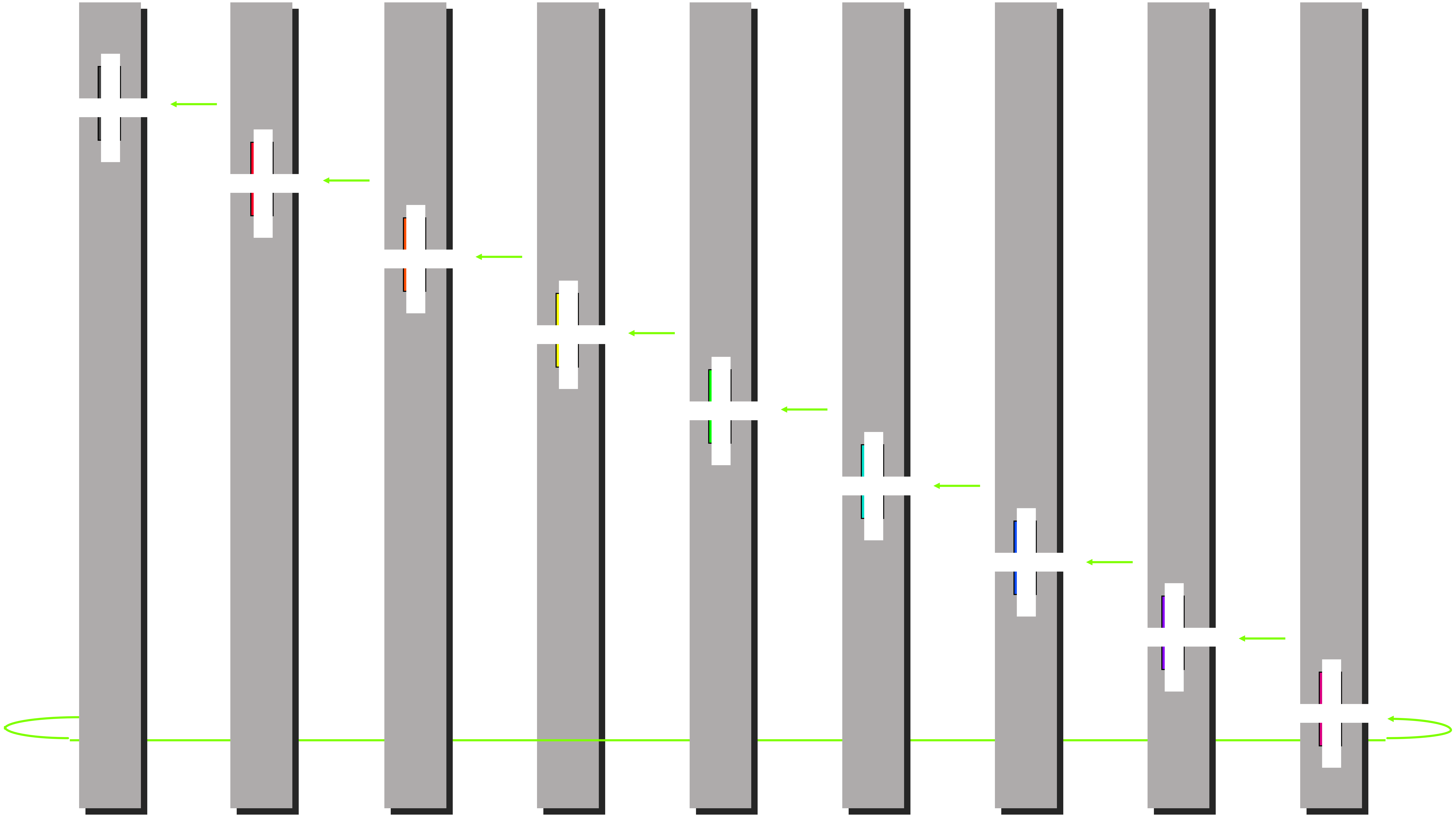


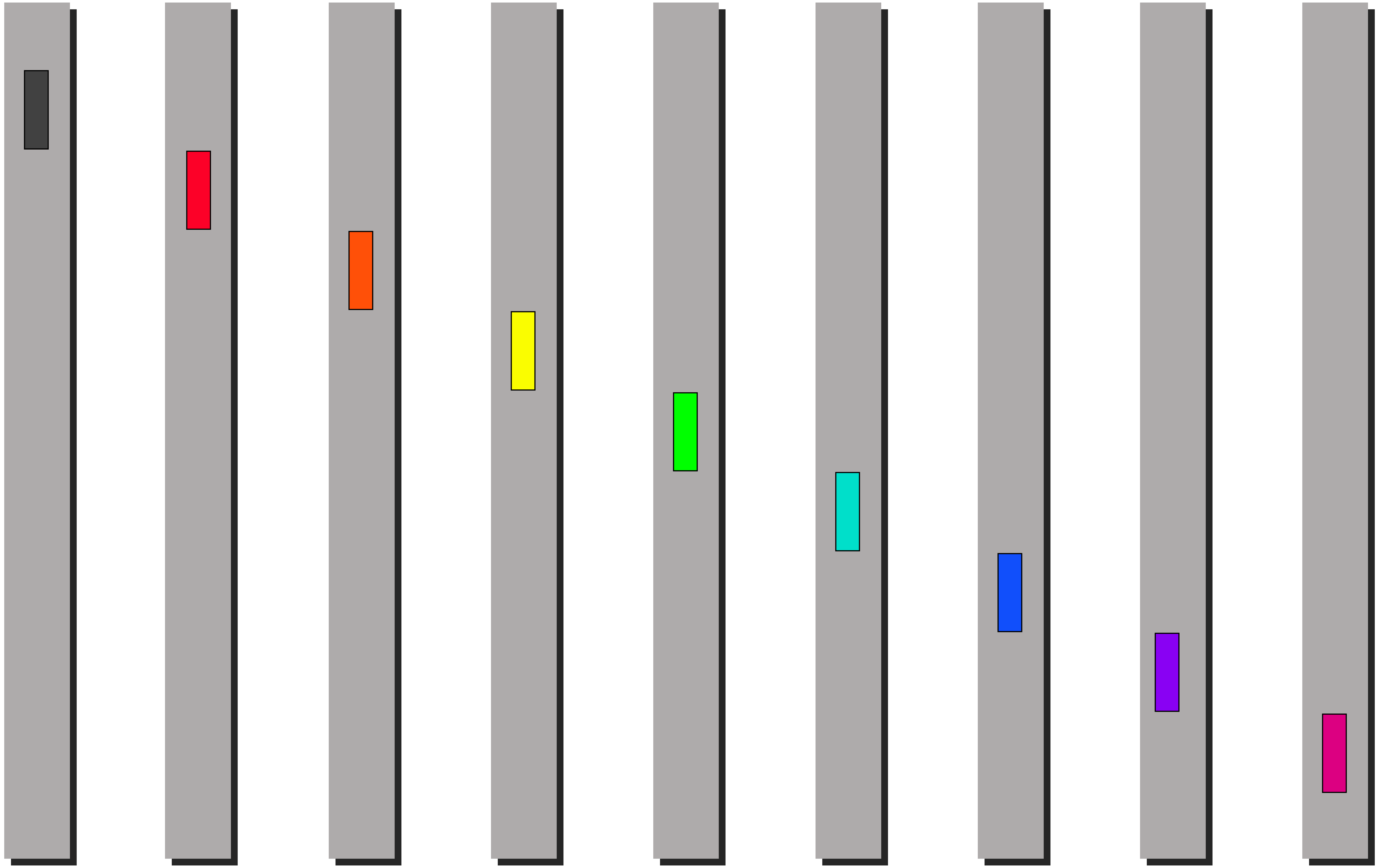












Some Transformations

Reduce-scatter

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

Scatter

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

Gather

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

Allgather

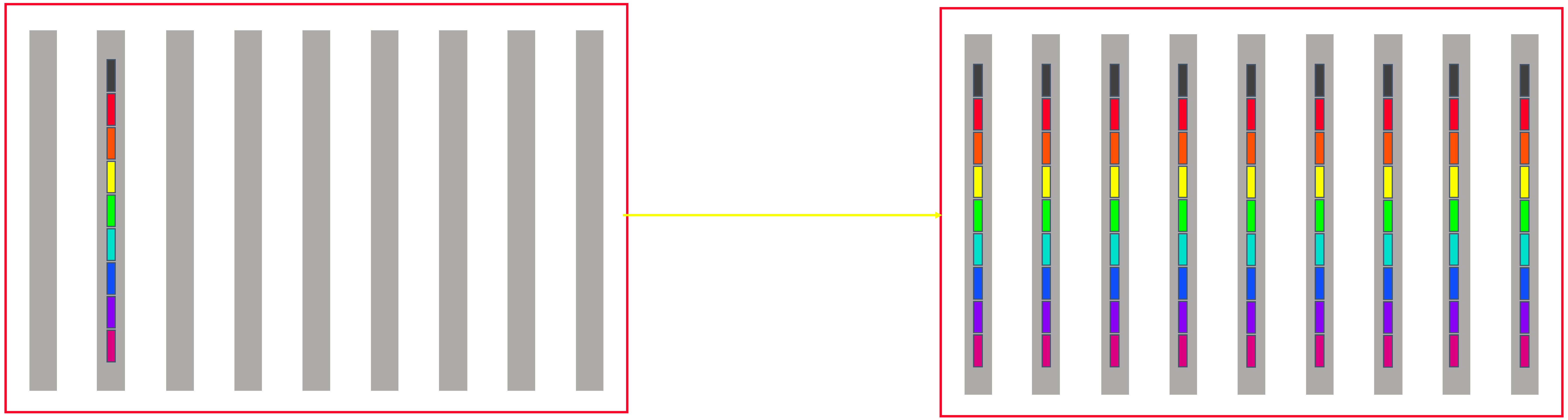
$$(p-1)\alpha+\frac{p-1}{p}n\beta$$

Reduce(-to-one)

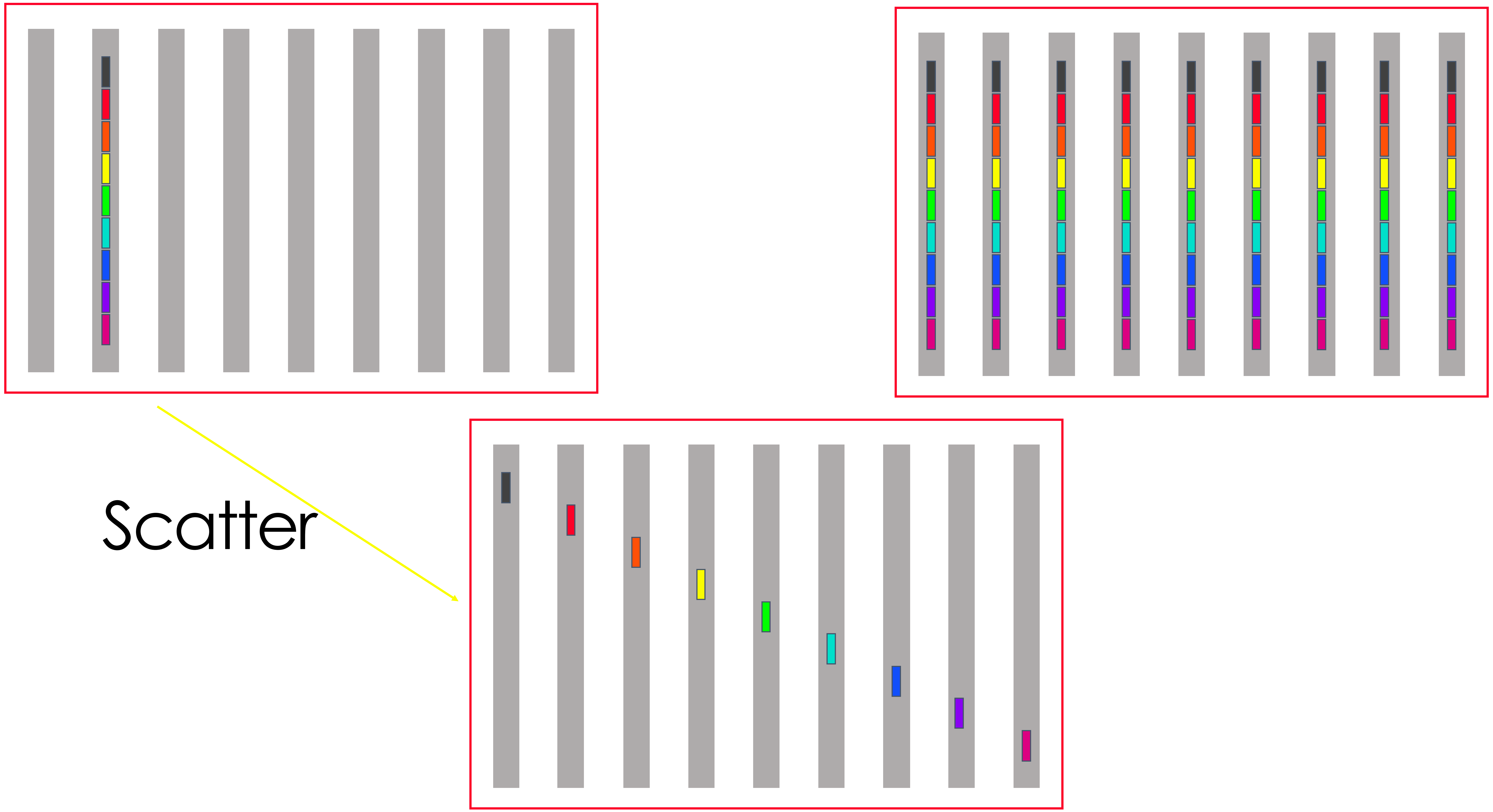
Allreduce

Broadcast

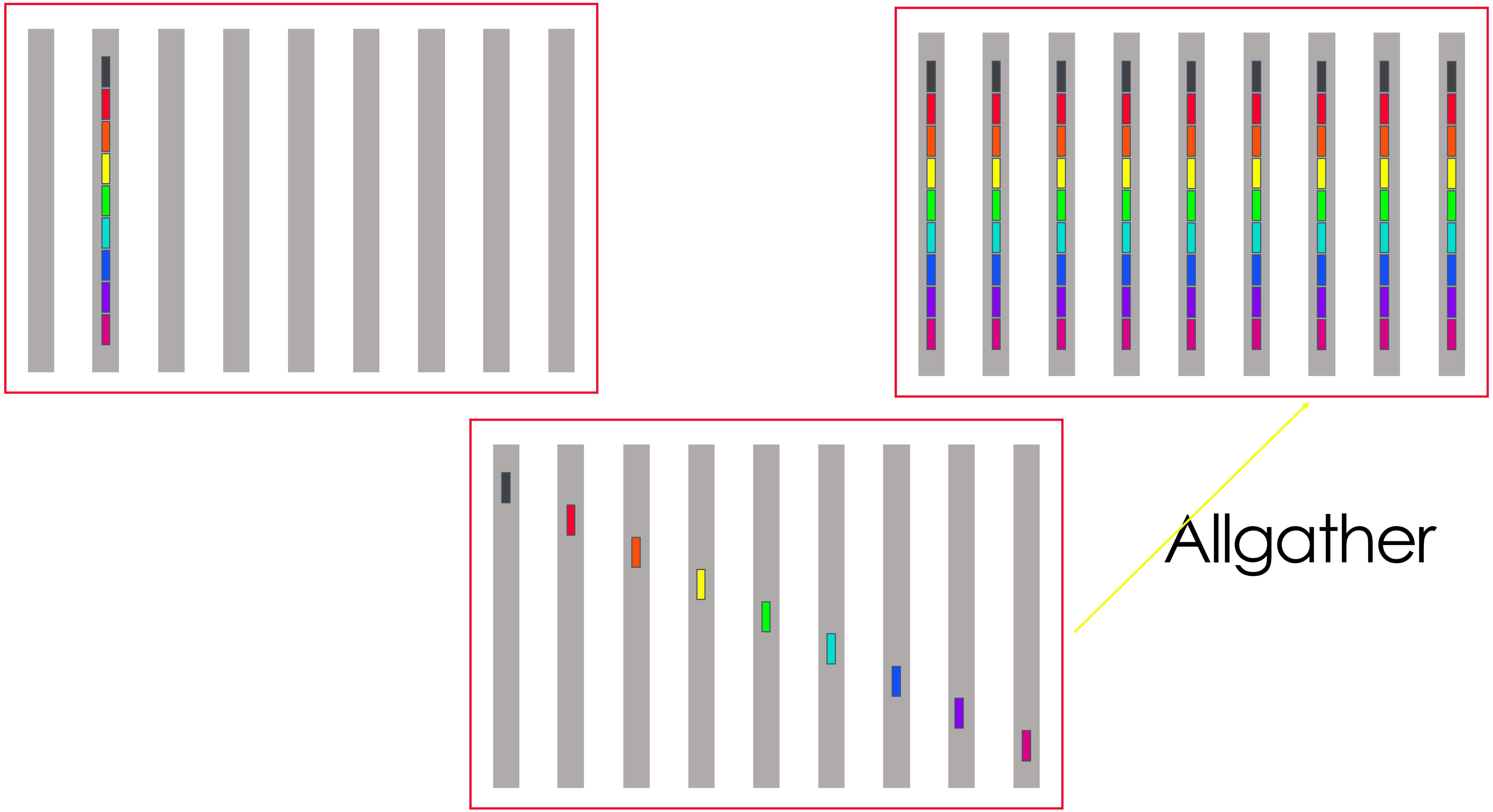
Broadcast (Large Message)



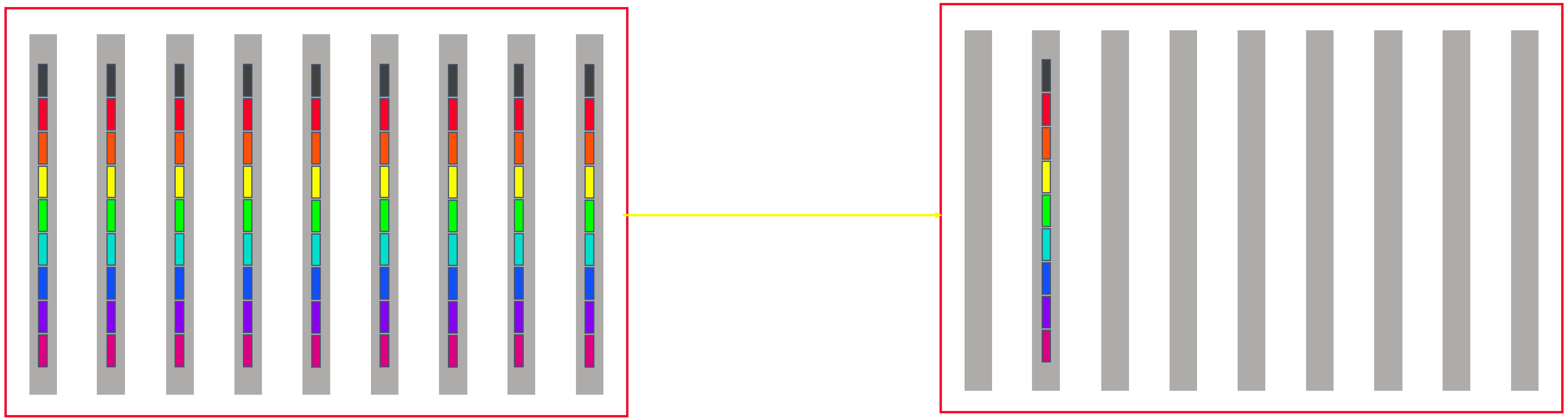
Broadcast (long vector)



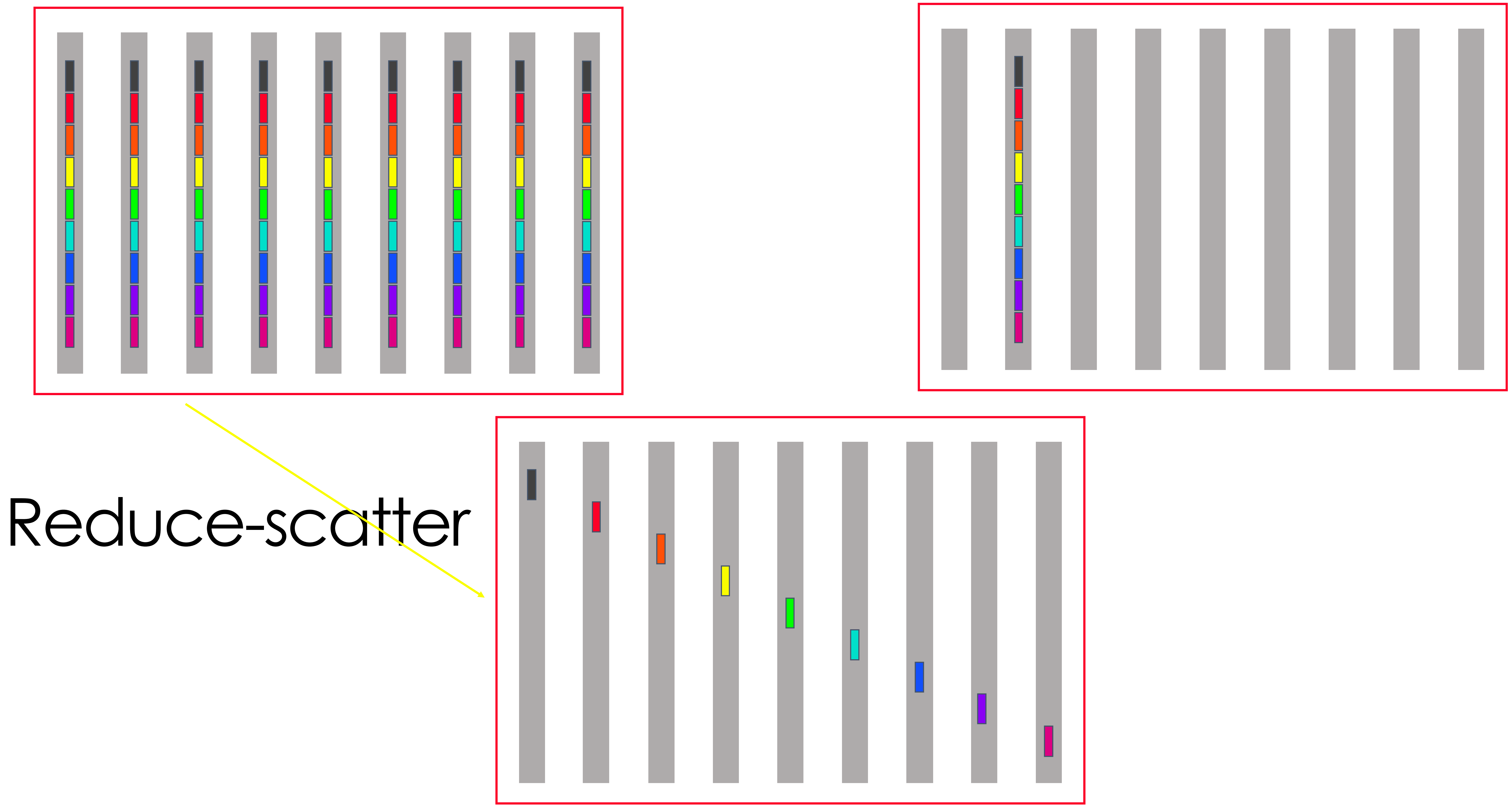
Broadcast (long vector)



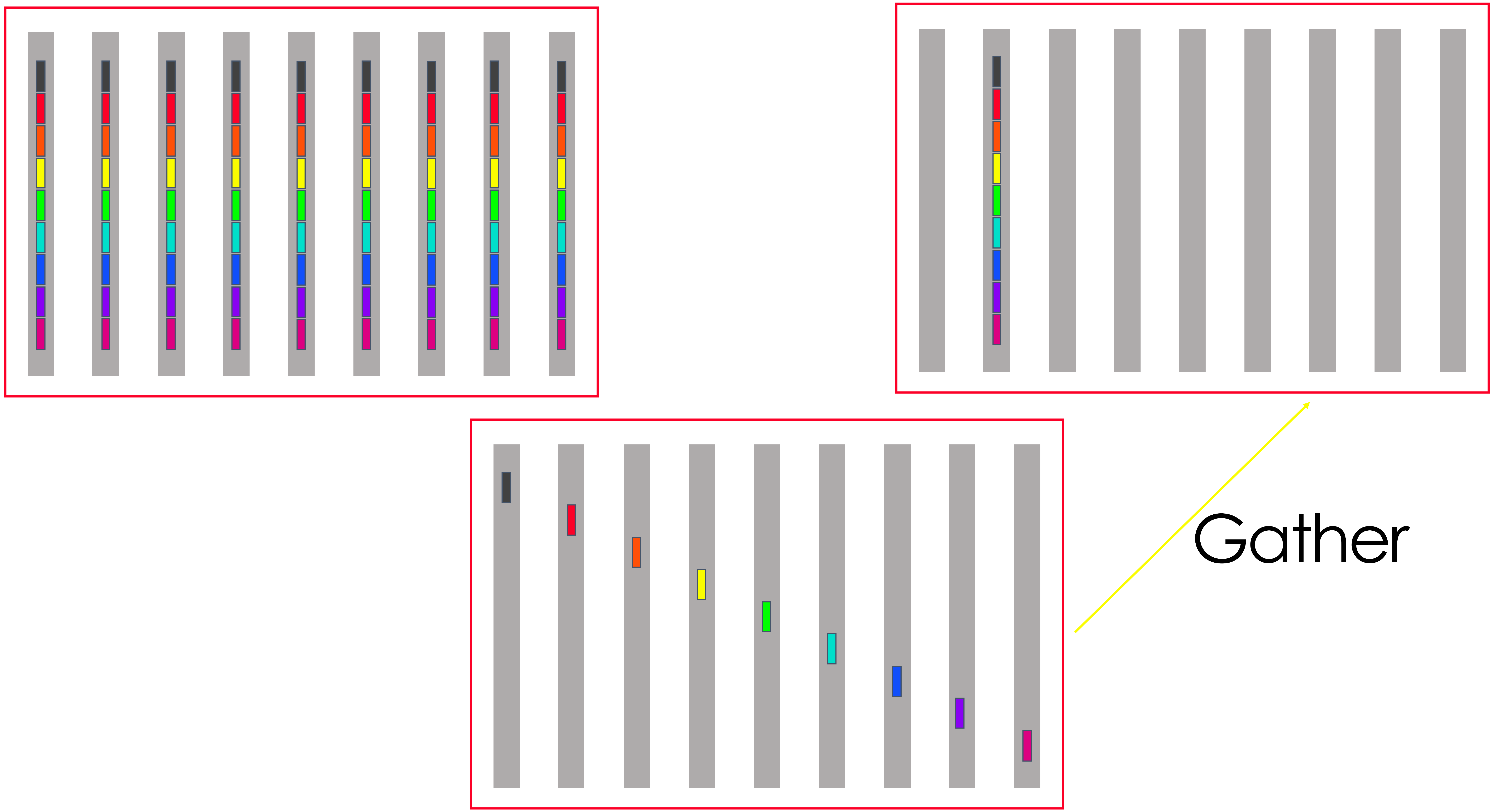
Reduce(-to-one) (long vector)



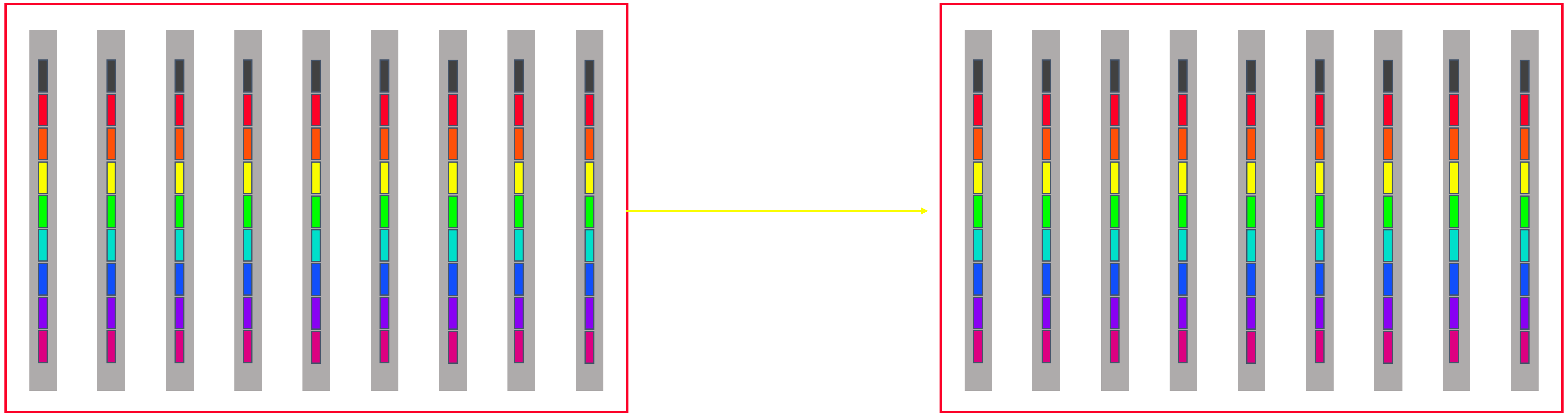
Reduce (long vector)



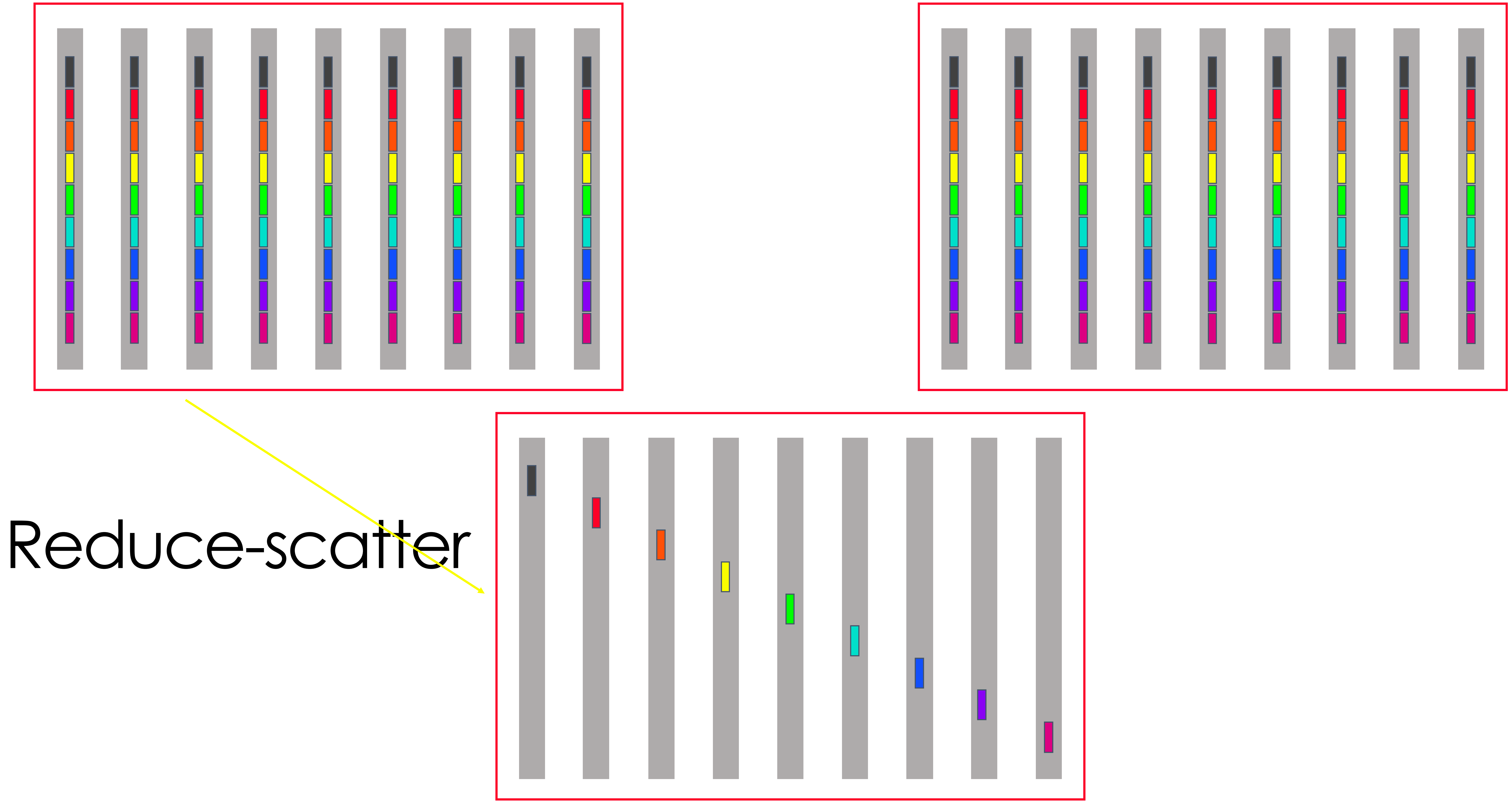
Combine-to-one (long vector)



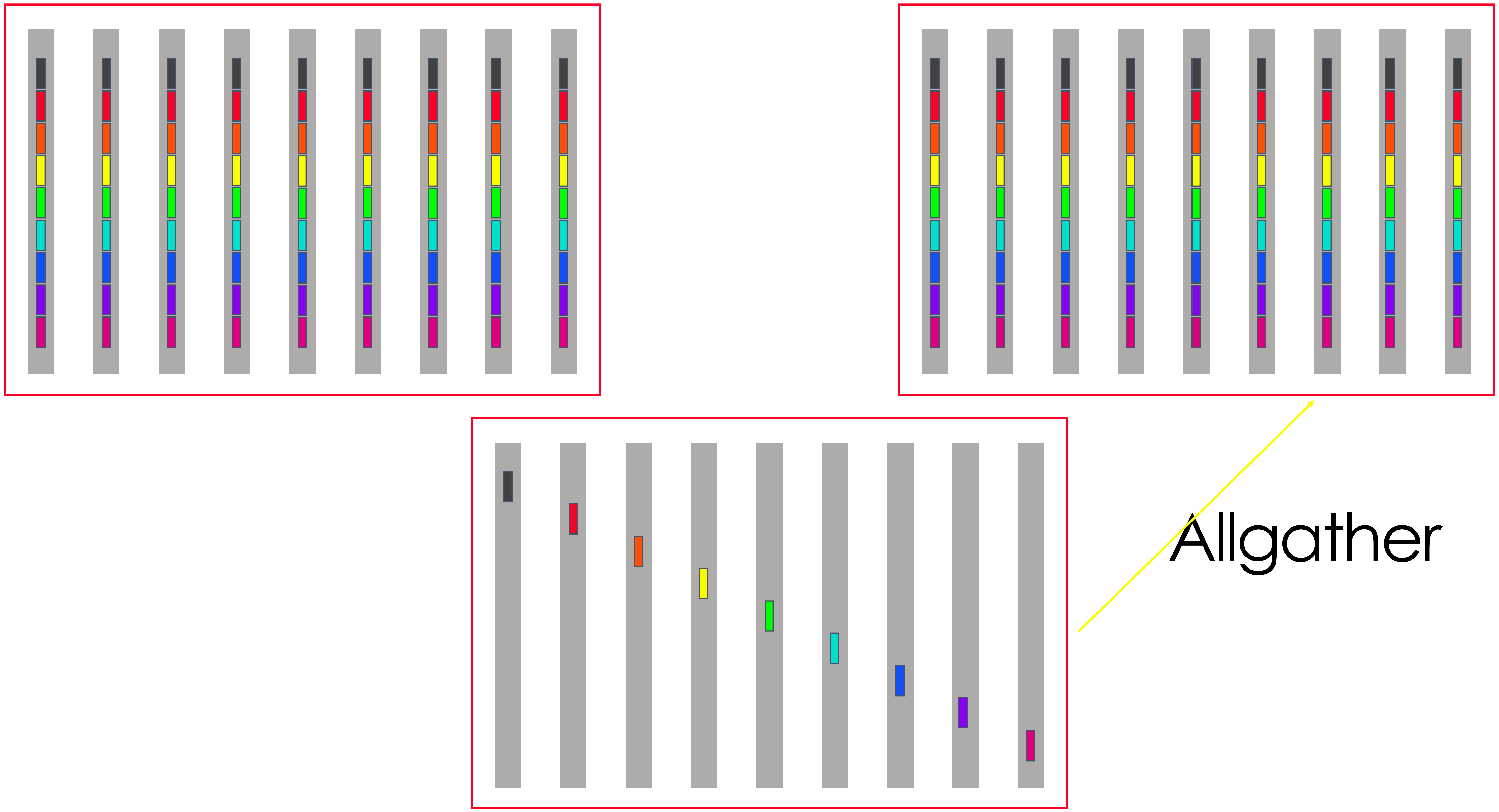
Allreduce (Large Message)



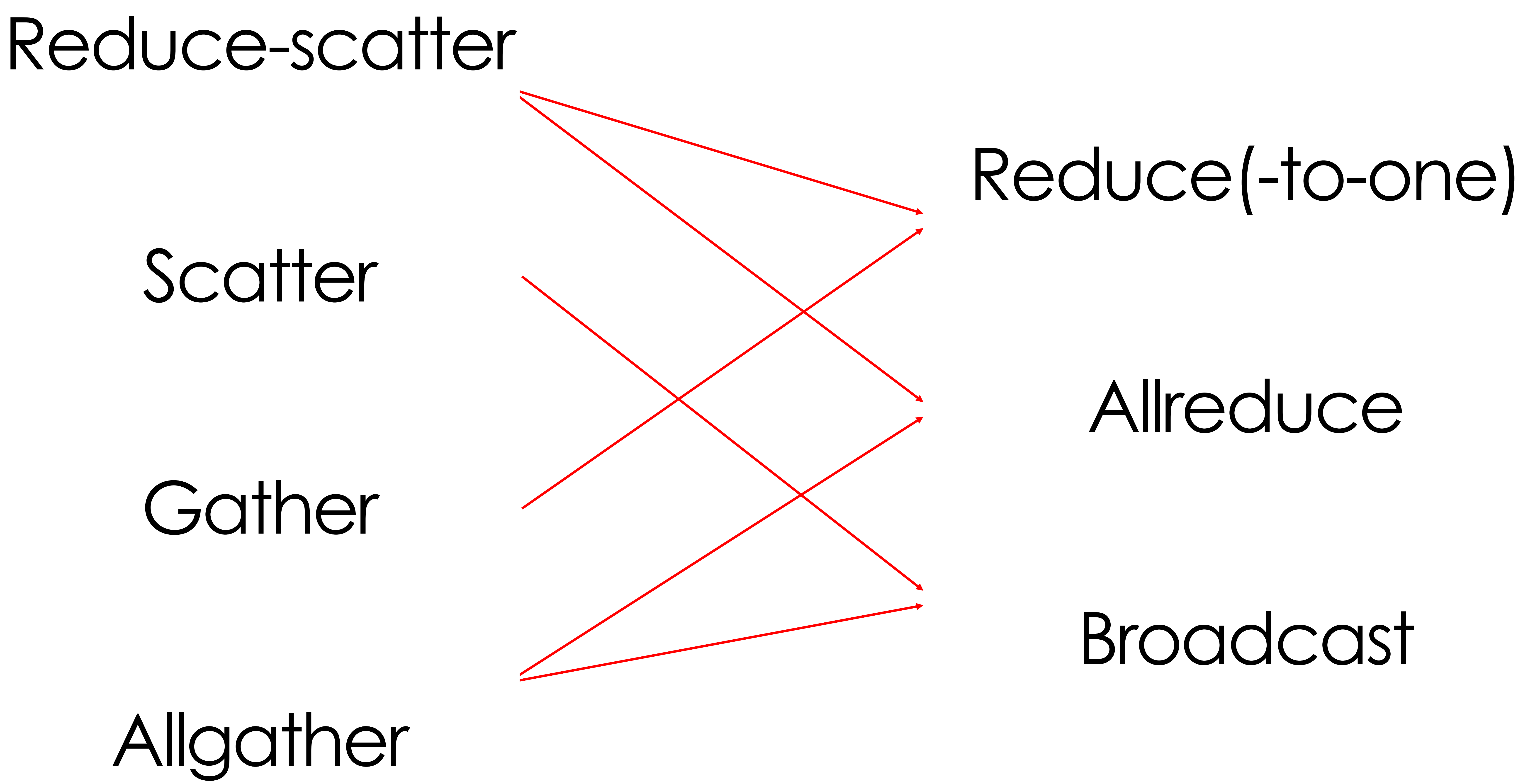
Allreduce (Large Message)



Allreduce
(long vector)



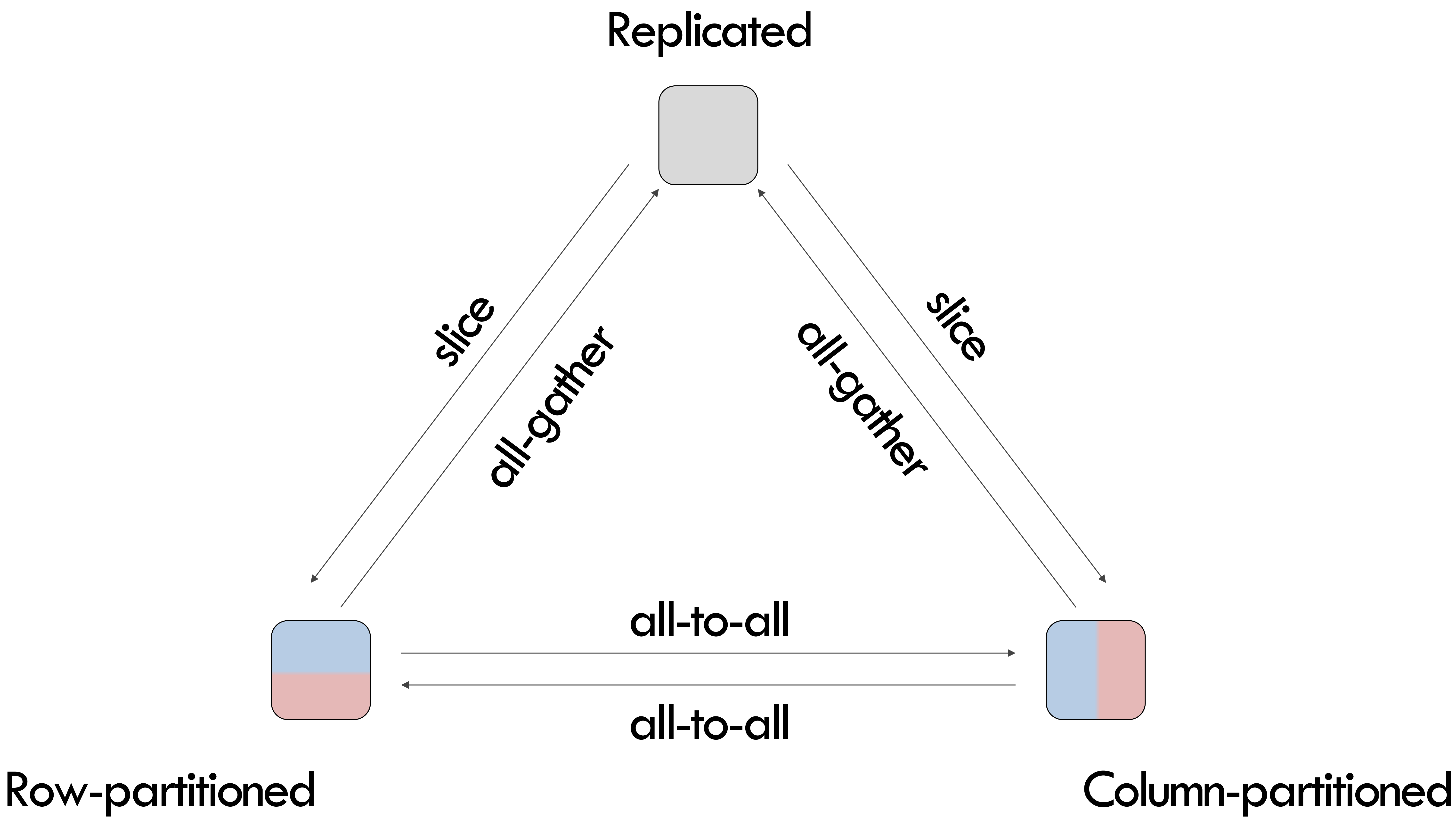
Recap


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

ML Parallelism and Communication

- Inter-op always results in P2P communication
  - This is quite obvious
- Intra-op always results in collective communication
  - Why?

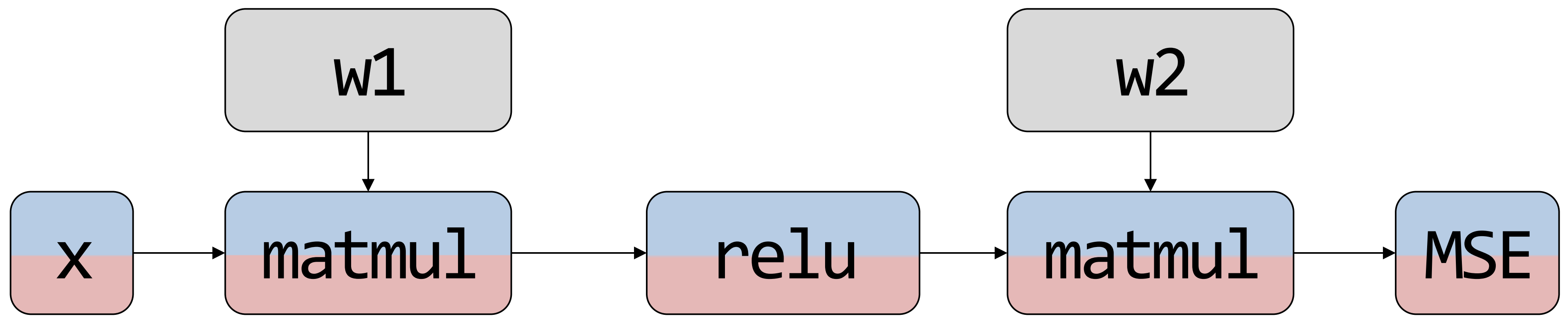
“Re-partition” Communication Cost in 2D



Where We Are

- Motivation
- History
- Parallelism Overview
- **Data parallelism**
- Model parallelism
  - Inter and intra-op parallelism
- Auto-parallelization

Data Parallelism



Two Solutions

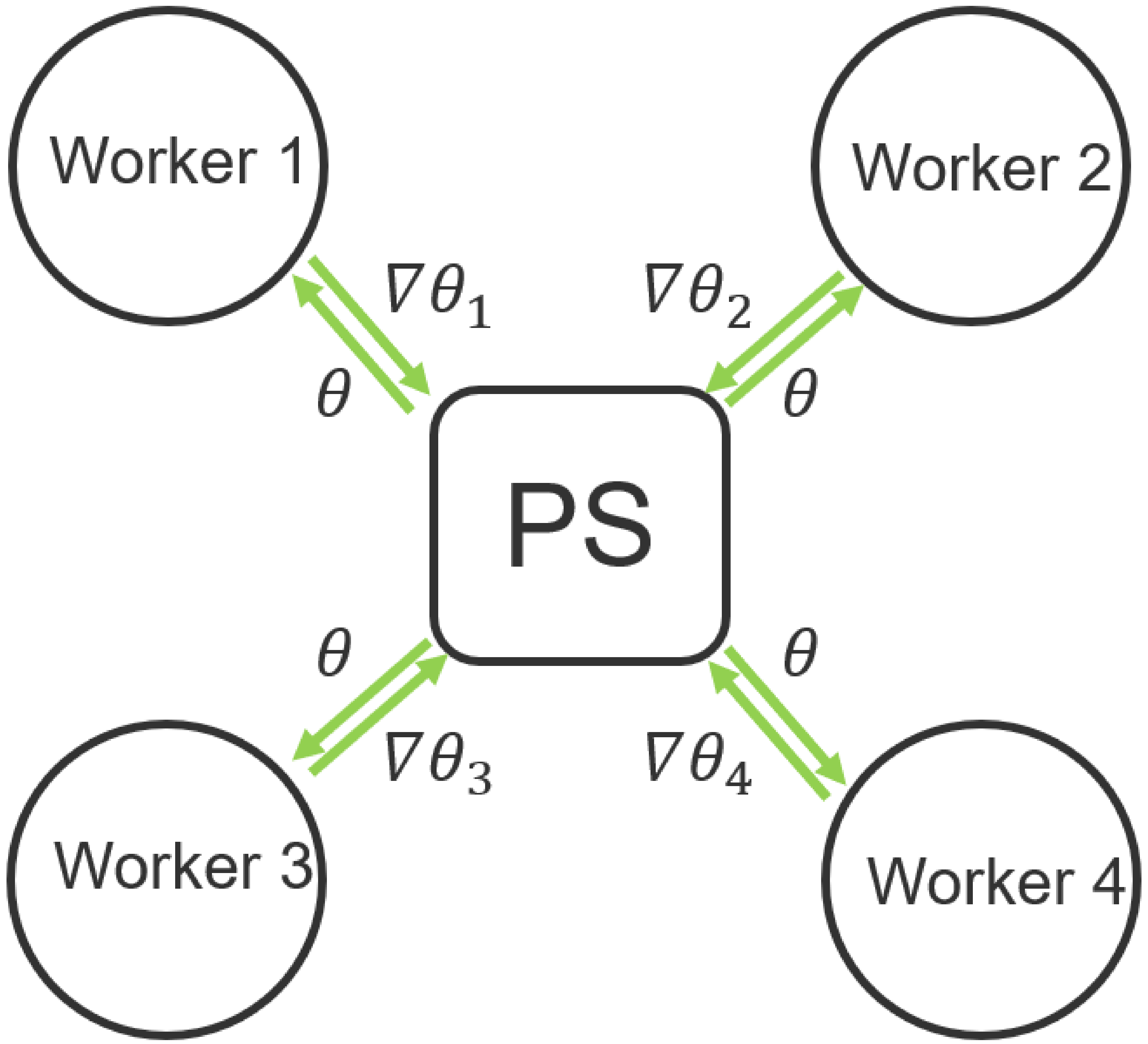
- Parameter Server
- AllReduce
- Key assumption:
  - The model can fit into an (GPU) worker memory hence we can create many replica

Parameter Server Assumption

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

- Very heavy communication per iteration
- Compute : communication = 1:10 in the era of 2012

Parameter Server Naturally emerges

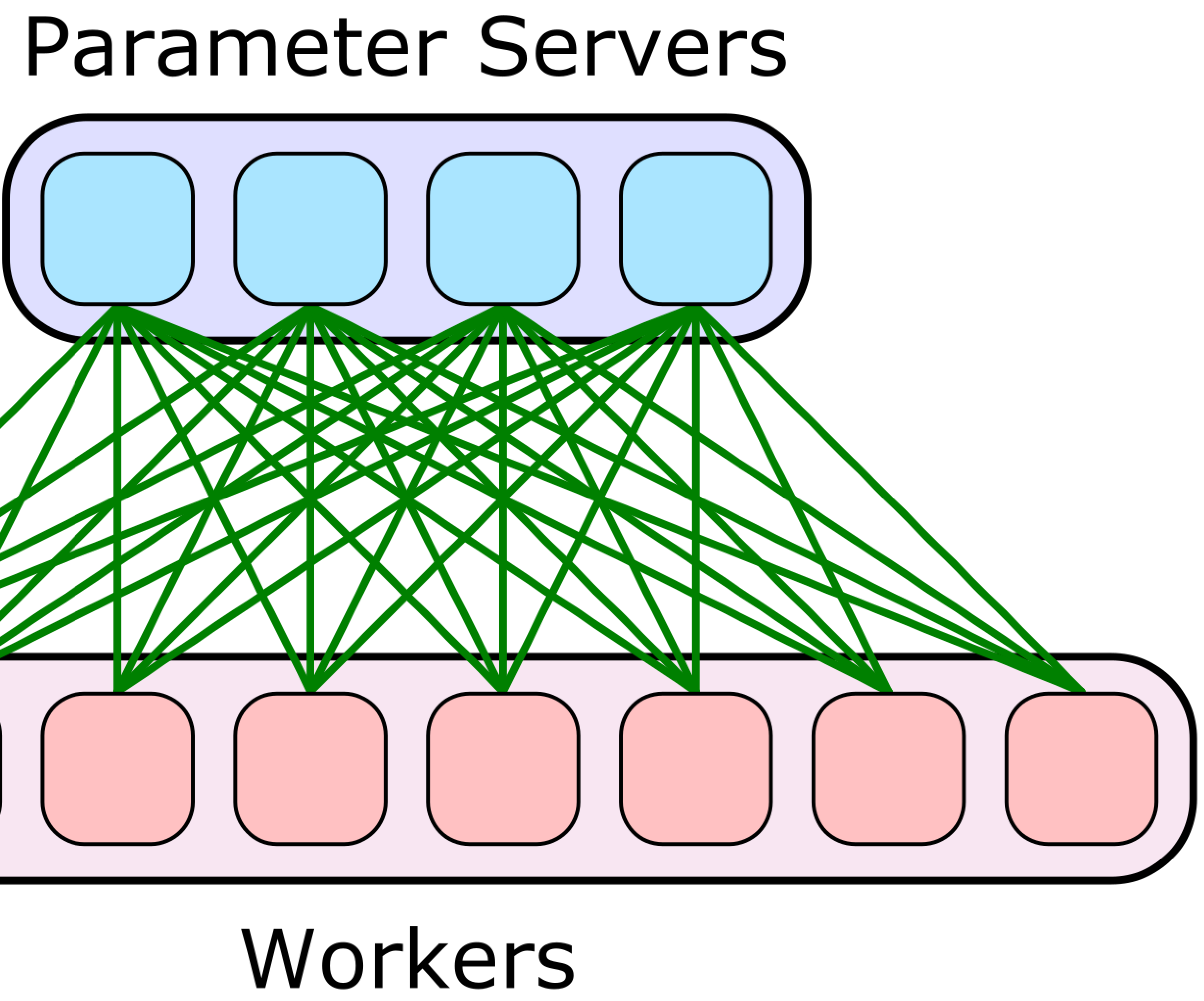


How to Implement Parameter Server?

- Key considerations:
  - Server: Communication bottleneck
  - Many (CPU) workers: hence fault tolerance

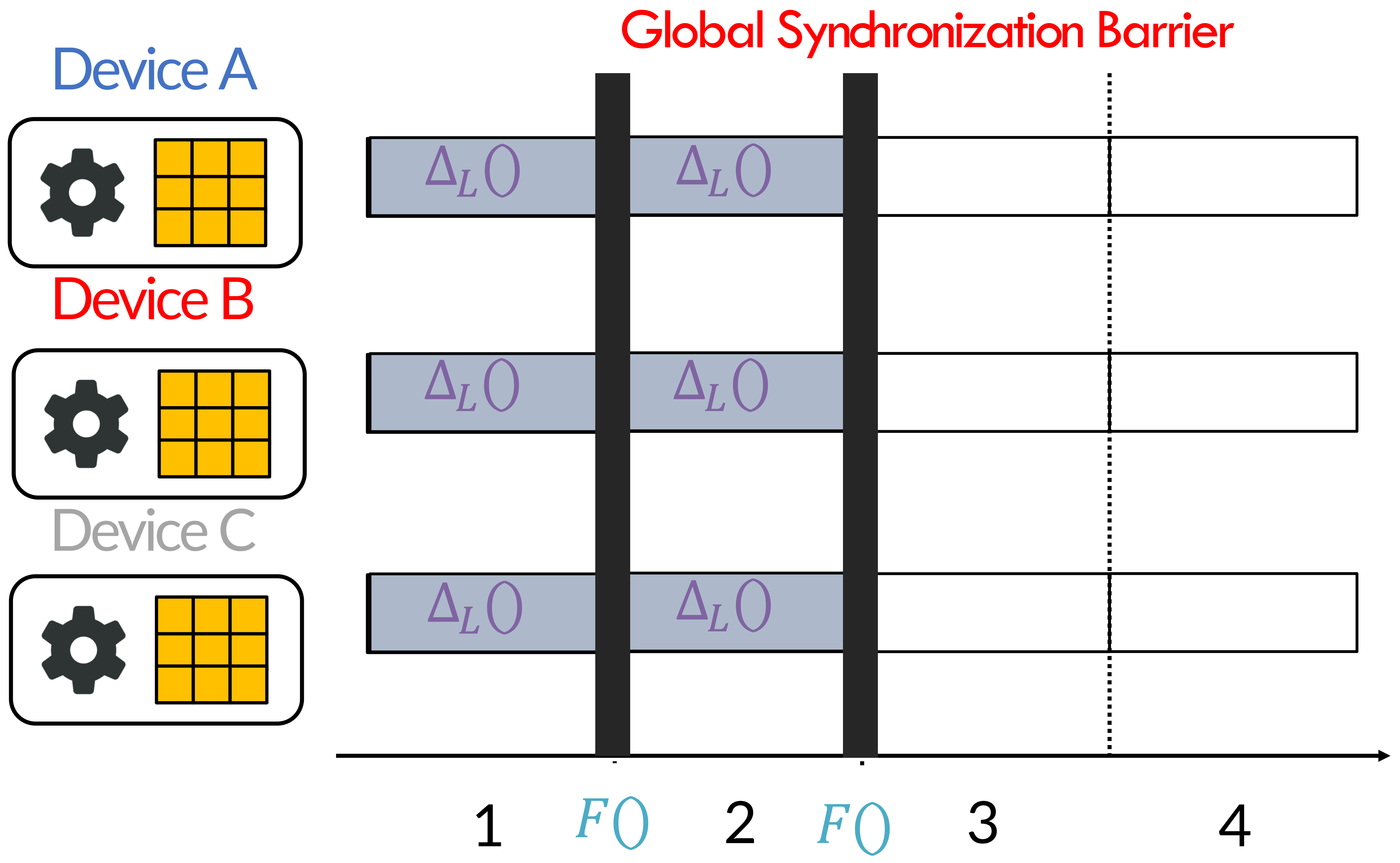
Parameter Server Implementation

- Sharded parameter server: sharded KV stores
  - Avoid communication bottleneck
  - Redundancy across different PS shards



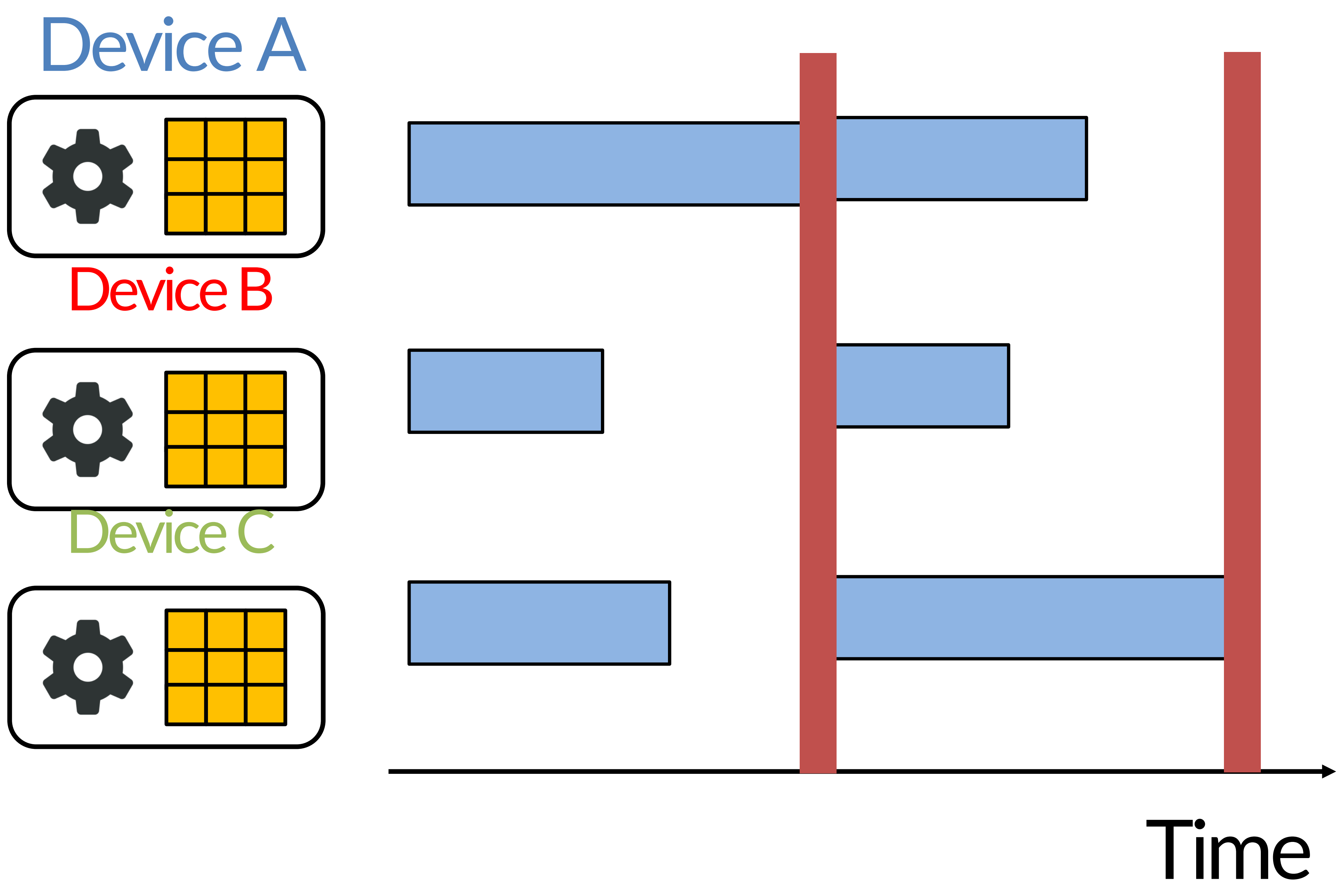
Consistency

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



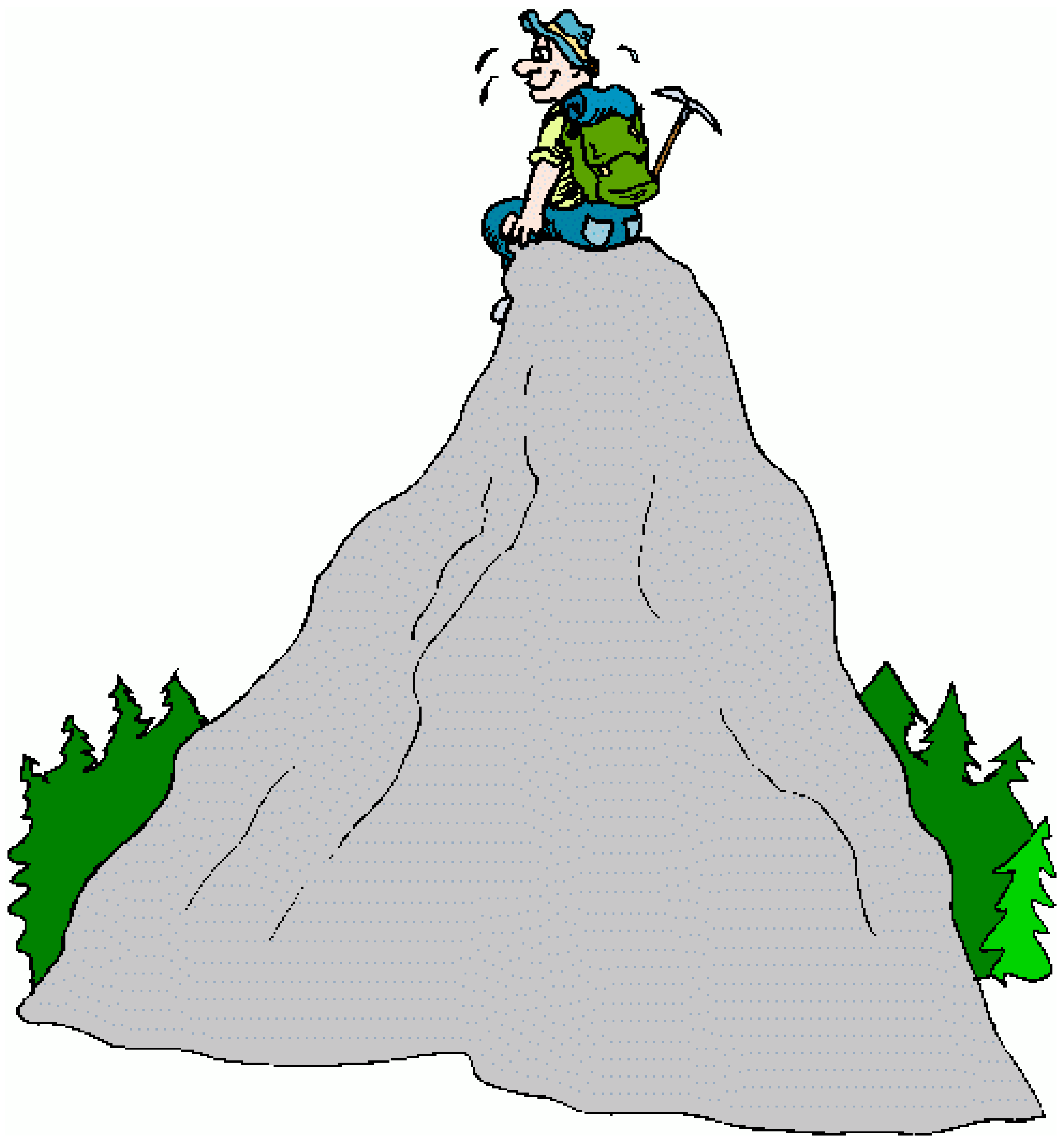
BSP's Weakness: Stragglers

- **BSP suffers from stragglers**
  - Slow devices (stragglers) force all devices to wait
  - More devices → higher chance of having a straggler

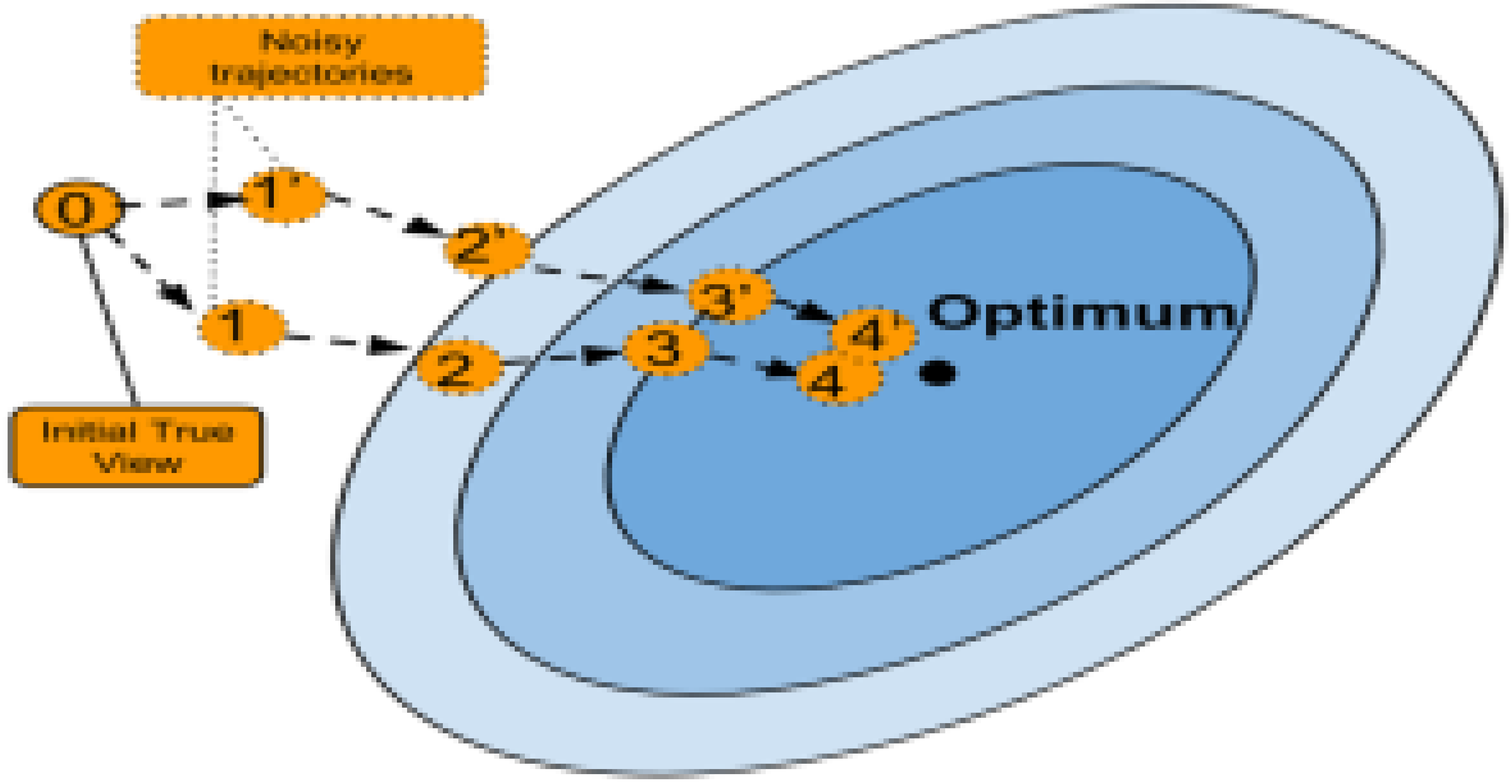


An interesting property of Gradient Descent (ascent)

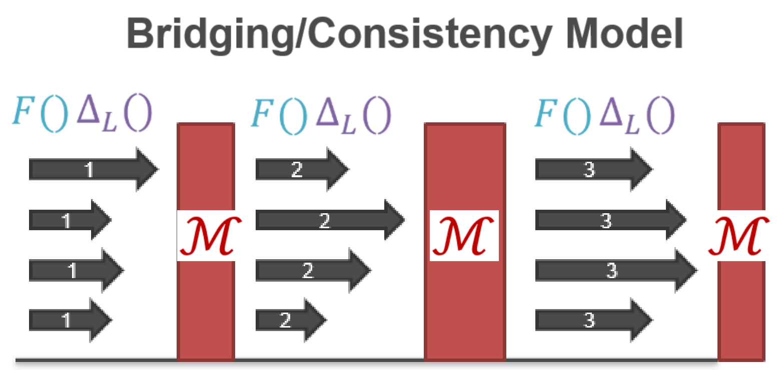
$$\theta^{(t+1)} = \theta^{(t)} + \varepsilon \sum_{p=1}^{P} \nabla_{\mathcal{L}}(\theta^{(t)}, D_p^{(t)})$$



Machine Learning is Error-tolerant (under certain conditions)

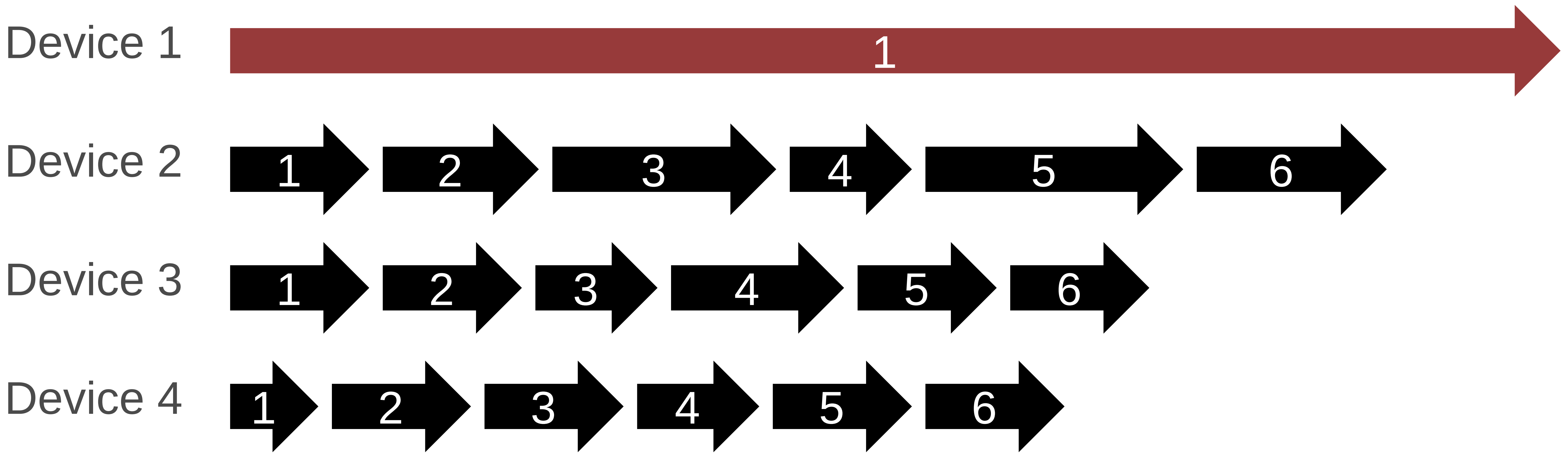


Background: Asynchronous Communication (No Consistency)

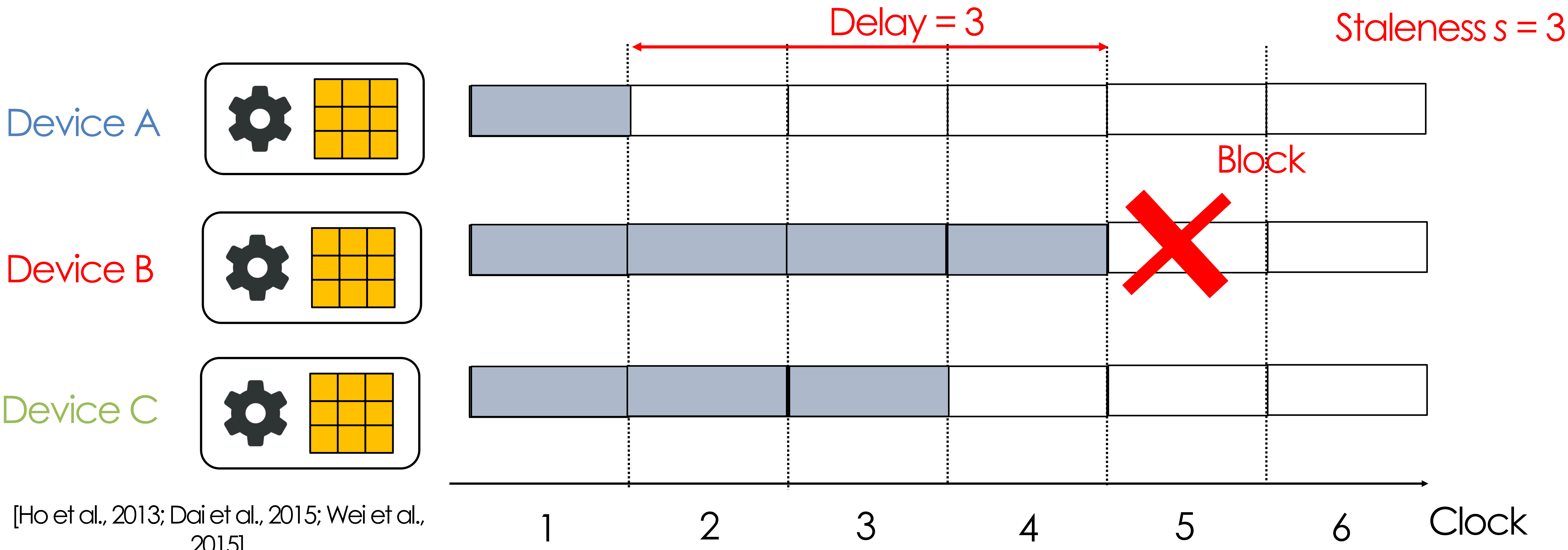


- **Asynchronous (Async):** removes all communication barriers
  - Maximizes computing time
  - Transient stragglers will cause messages to be extremely stale
    - Ex: Device 2 is at $t = 6$, but Device 1 has only sent message for $t = 1$

- **Some Async software:** <u>messages can be applied while computing</u> $F()$, $\Delta_L()$
  - Unpredictable behavior, can hurt statistical efficiency!

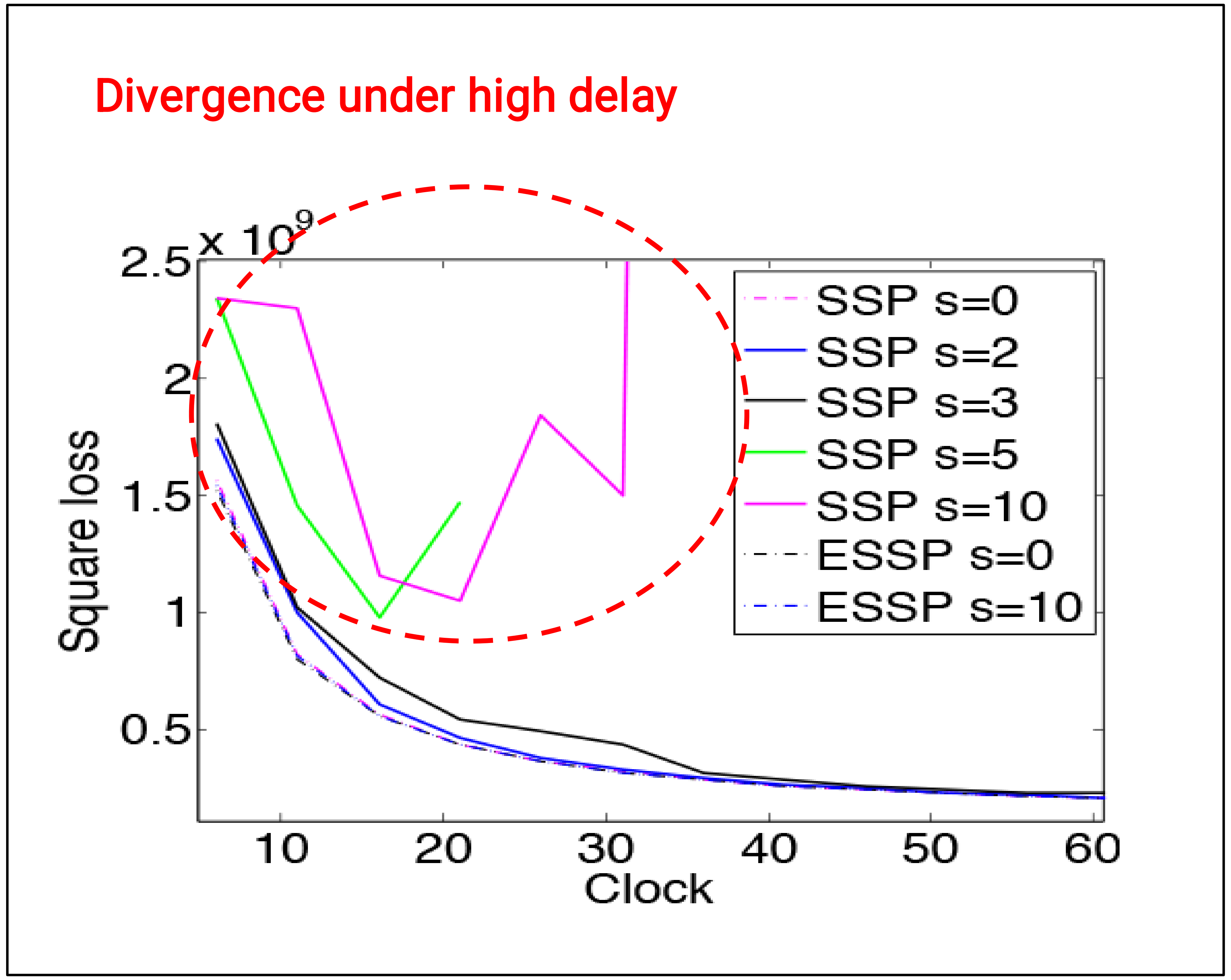


Background: Bounded Consistency

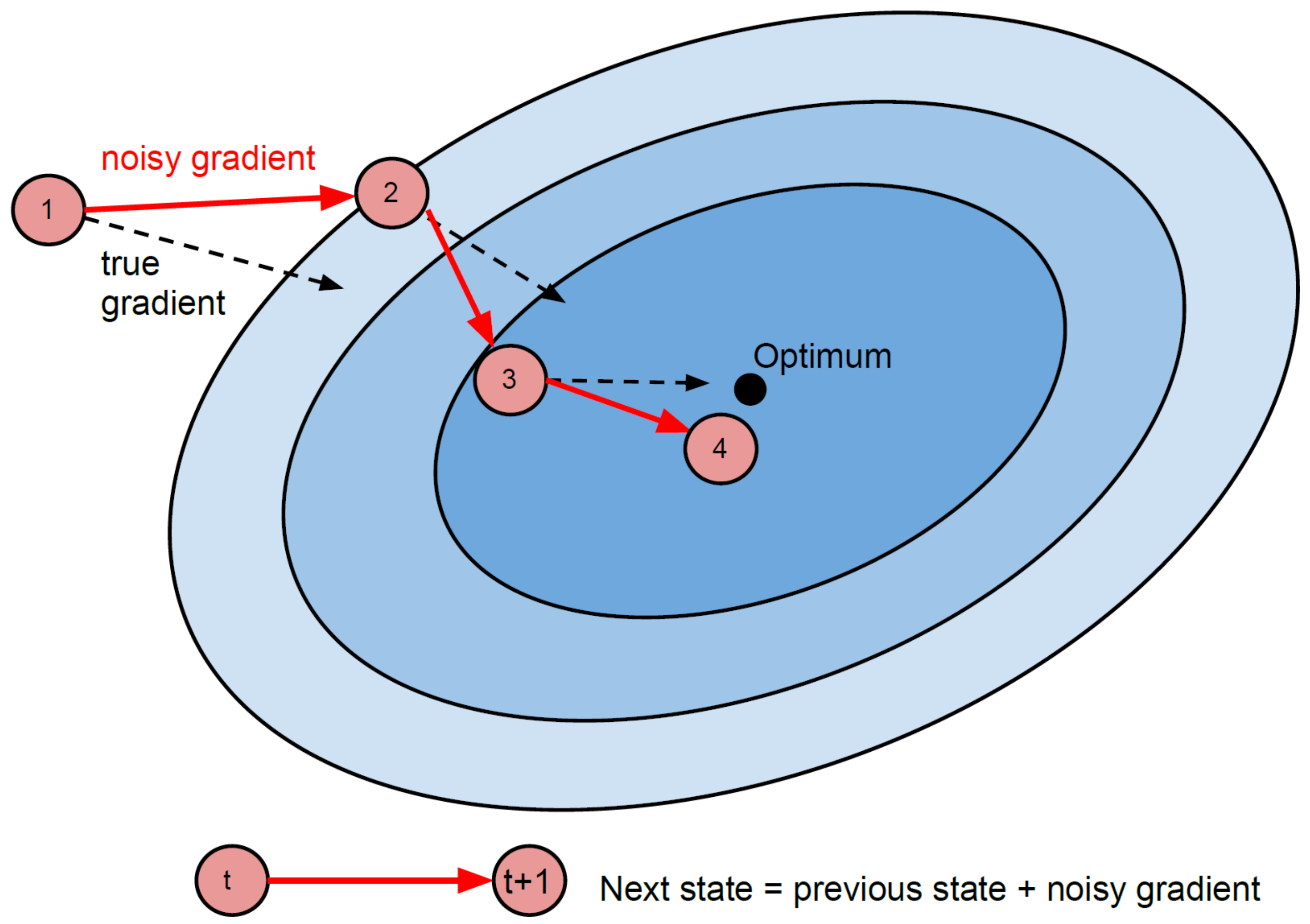


[Ho et al., 2013; Dai et al., 2015; Wei et al., 2015]

Impacts of Consistency/Staleness: Unbounded Staleness



Theory: SSP Expectation Bound

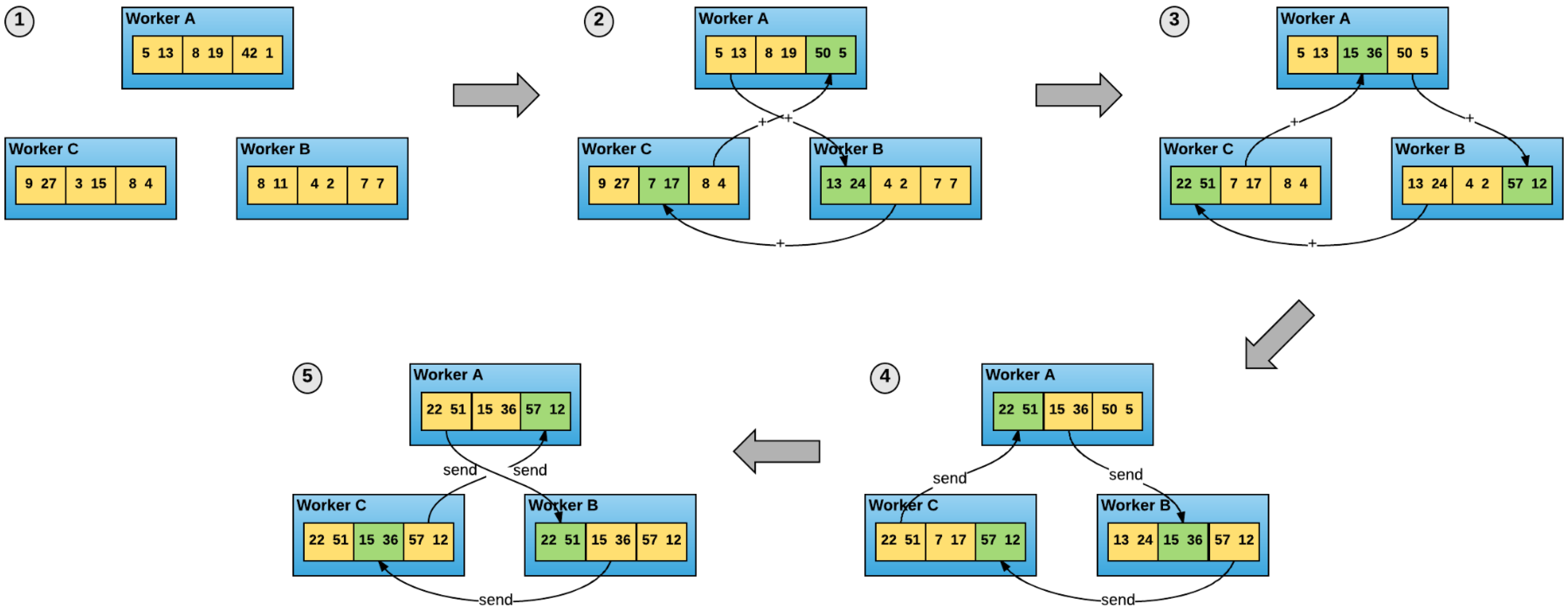


$$R[\mathbf{X}] := \overbrace{\left[\frac{1}{T}\sum_{t=1}^{T} f_t(\tilde{\mathbf{x}}_t)\right] - f(\mathbf{x}^*)}^{\text{Difference between SSP estimate and true optimum}} \leq 4FL\sqrt{\frac{2(s+1)P}{T}}$$

Summary: Parameter Server

- Why did it emerge?
- Why did it become irrelevant?

AllReduce



1
Worker A
5 13
8 19
42 1
Worker C
9 27
3 15
8 4
Worker B
8 11
4 2
7 7
2
Worker A
5 13
8 19
50 5
Worker C
9 27
7 17
8 4
Worker B
13 24
4 2
7 7
3
Worker A
5 13
15 36
50 5
Worker C
22 51
7 17
8 4
Worker B
13 24
4 2
57 12
4
Worker A
22 51
15 36
50 5
Worker C
22 51
7 17
57 12
Worker B
13 24
15 36
57 12
send
send
send
5
Worker A
22 51
15 36
57 12
Worker C
22 51
15 36
57 12
Worker B
22 51
15 36
57 12
send
send
send

Data Parallelism with All-reduce

```python
import torch.nn.parallel as dist
from torch.nn.parallel import DistributedDataParallel as DDP

dist.init_process_group("nccl", rank=rank, world_size=world_size)
ddp_model = DDP(Model(), device_ids=[rank])

for batch in data_loader:
    loss = train_step(ddp_model, batch)
```

Sergeev et al., "Horovod: fast and easy distributed deep learning in TensorFlow". *Preprint 2018.*
Li et al., "PyTorch Distributed: Experiences on Accelerating Data Parallel Training". VLDB 2020.

Allreduce

- Initially implemented in Horovod
- Being Optimized by nvidia (hw/sw cooptimizaiton)
- Being adopted in PyTorch DDP
- Not Fault tolerant

Q: Why Allreduce dominates parameter server today?

Next Lecture

- Motivation
- History
- Parallelism Overview
- Data parallelism
- **Model parallelism**
  - Inter and intra-op parallelism
- Auto-parallelization