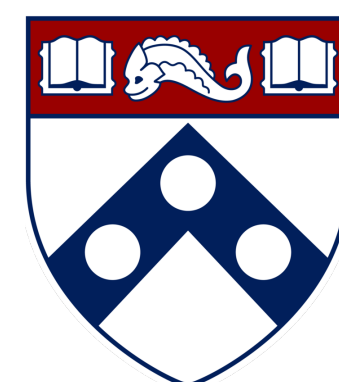
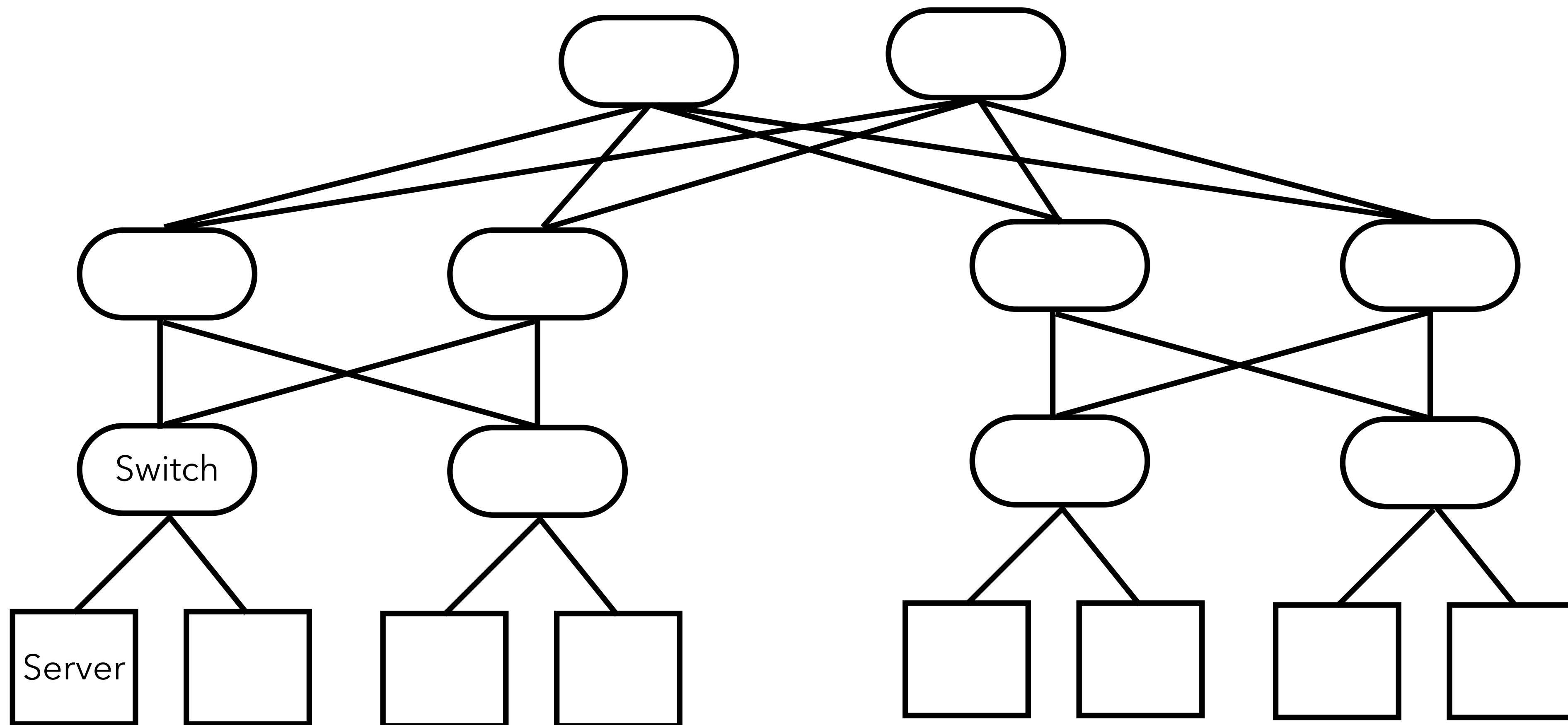


# FAST: An Efficient Scheduler for All-to-All GPU Communication

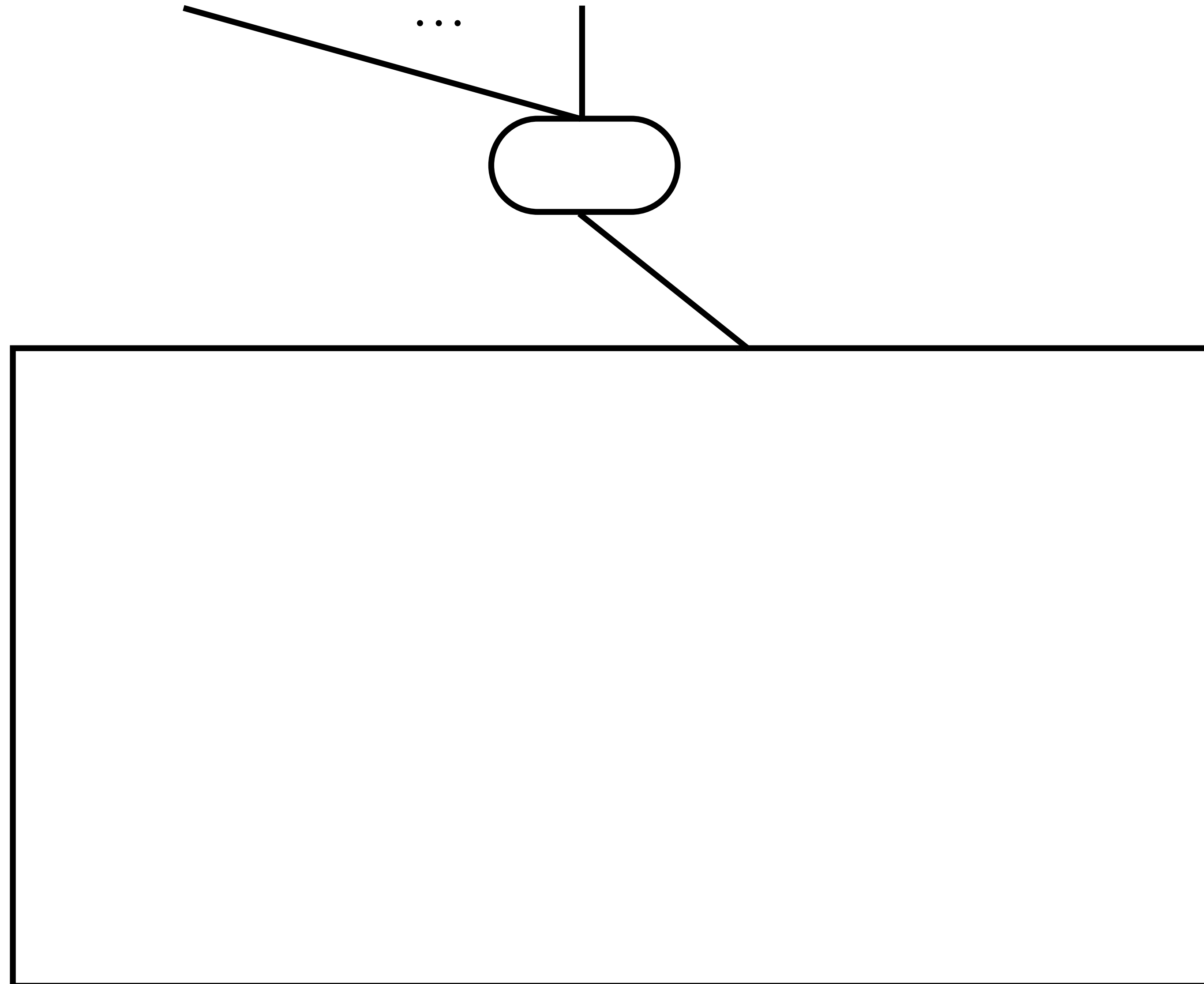
Yiran Lei, Dongjoo Lee, Liangyu Zhao, Daniar Kurniawan, Chanmyeong Kim, Heetaek Jeong, Changsu Kim, Hyeonseong Choi, Liangcheng Yu, Arvind Krishnamurthy, Justine Sherry, Eriko Nurvitadhi



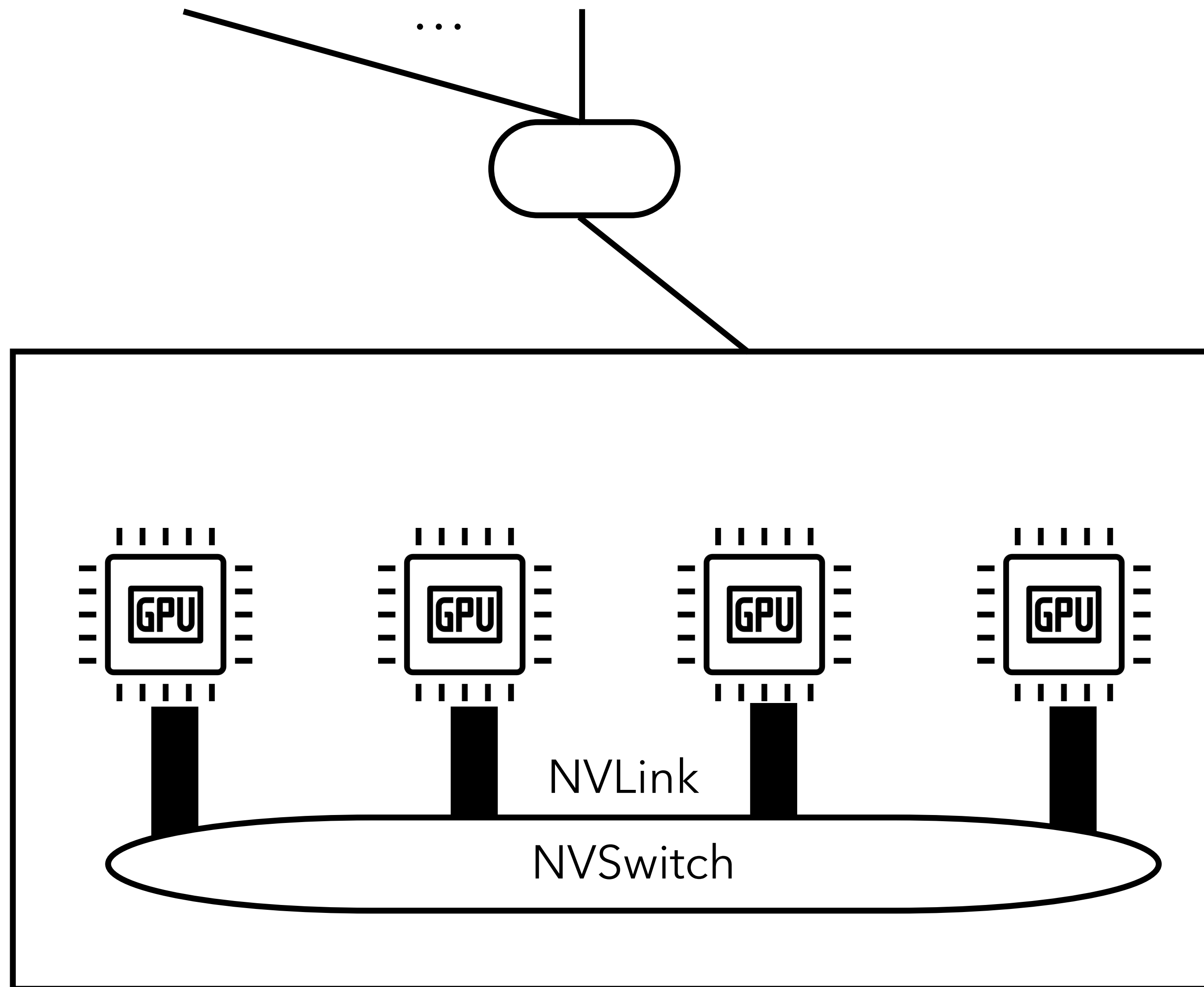
# Traditional data center networks



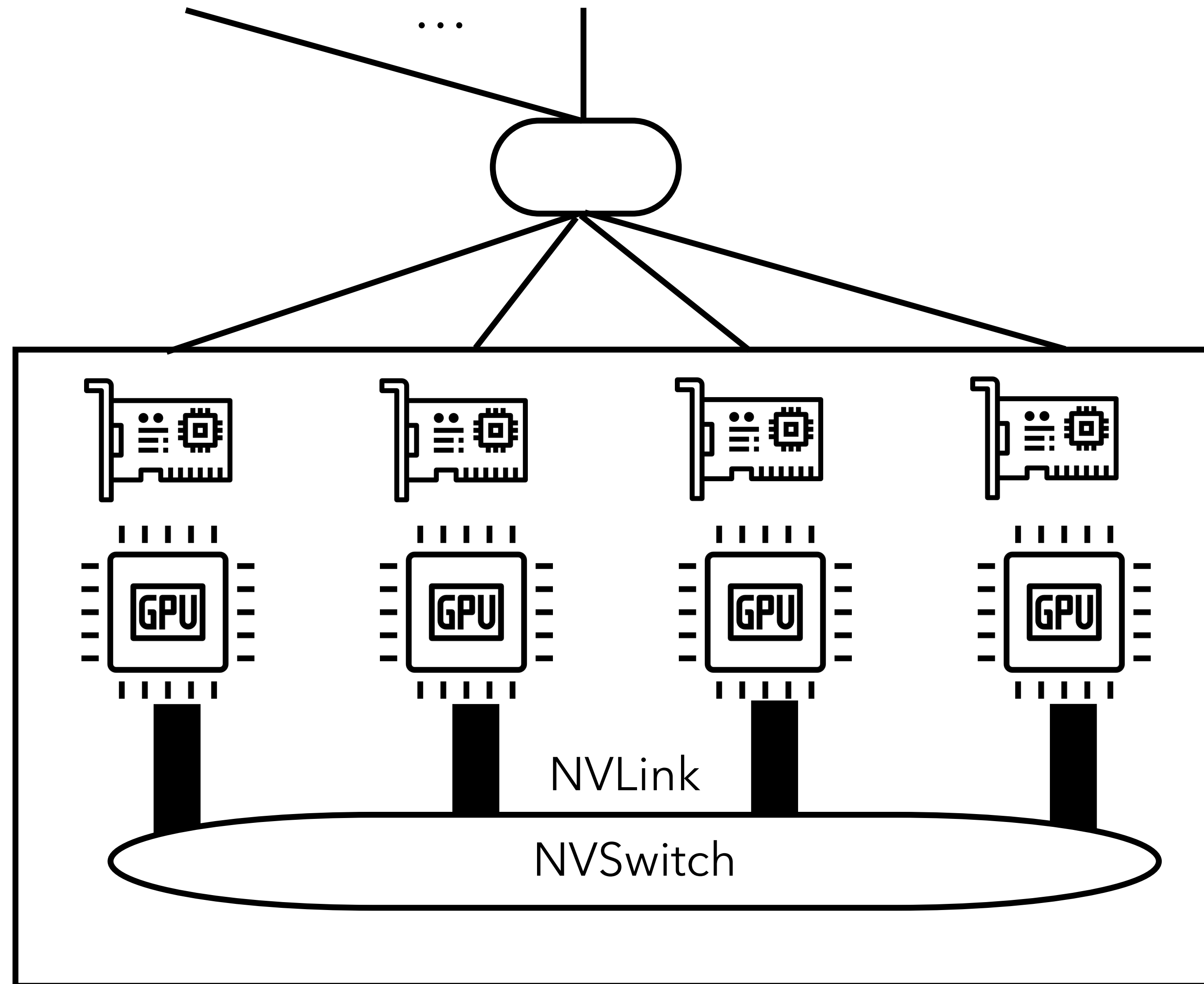
# Networks inside networks



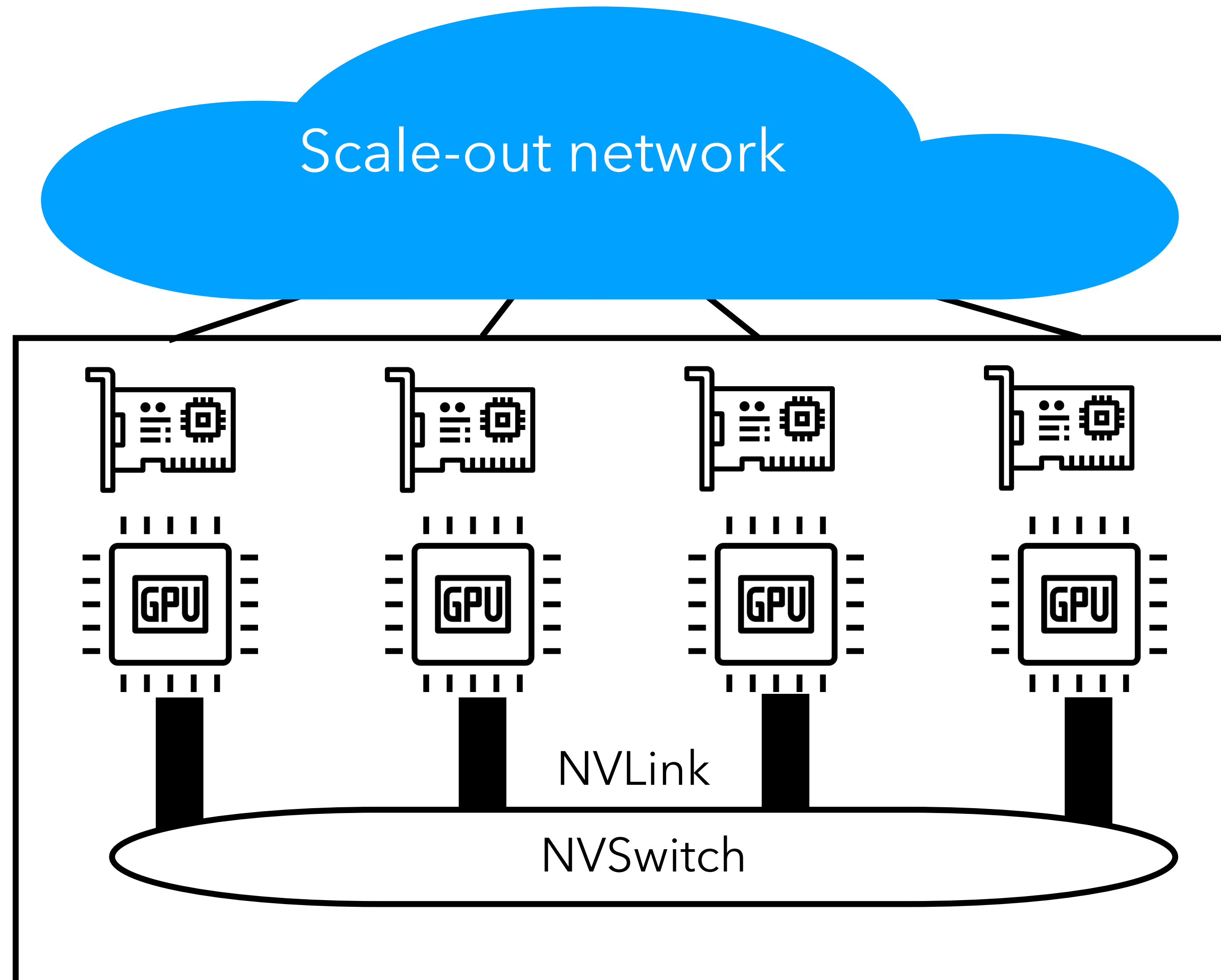
# Networks inside networks



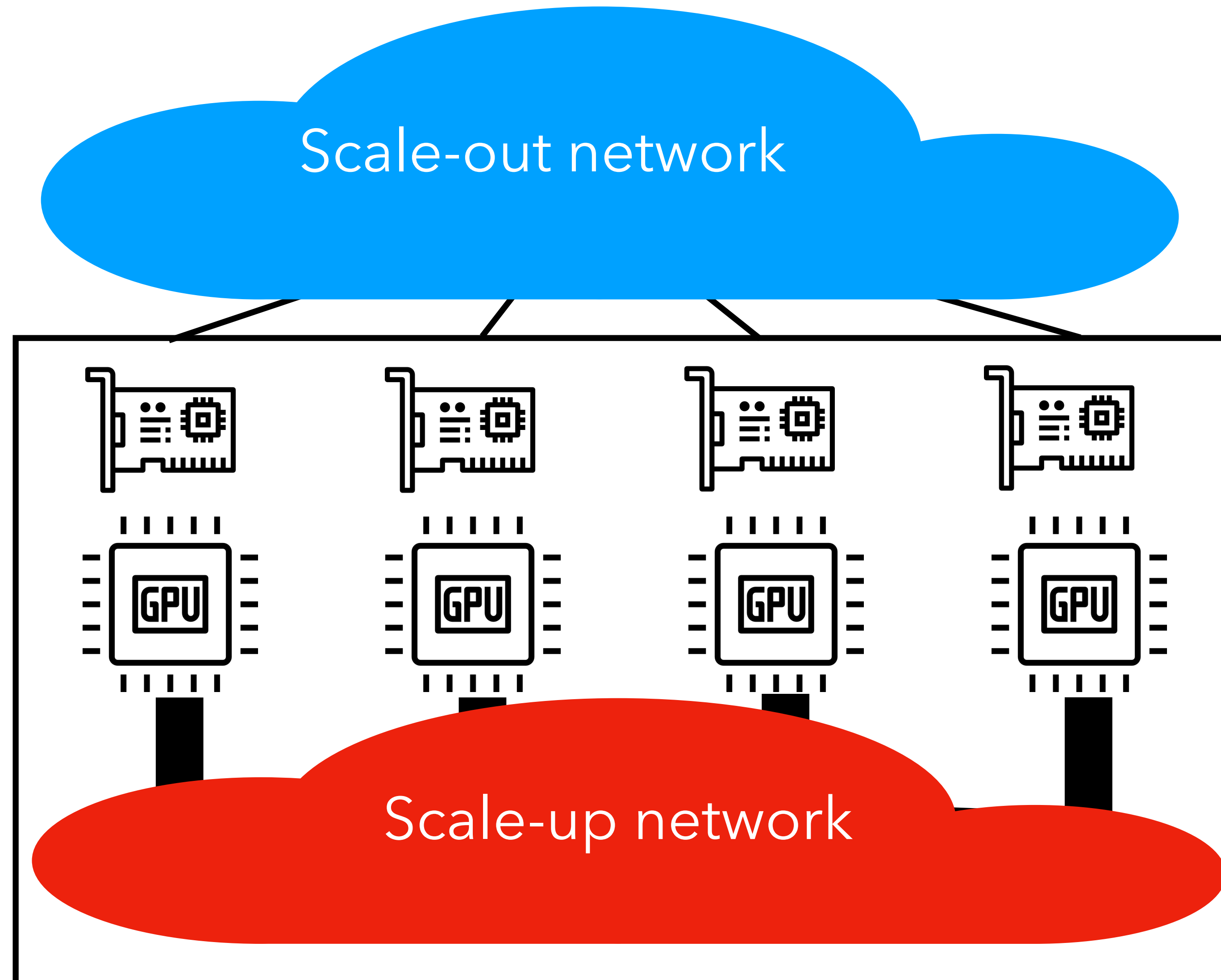
# Networks inside networks



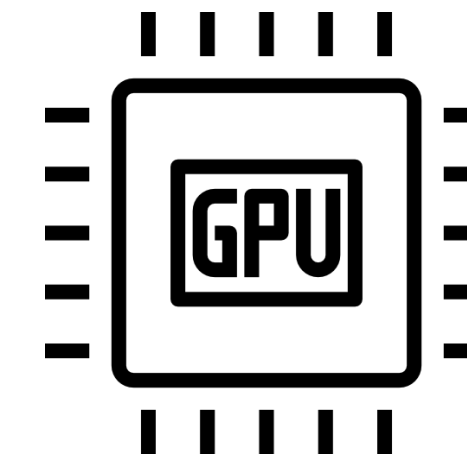
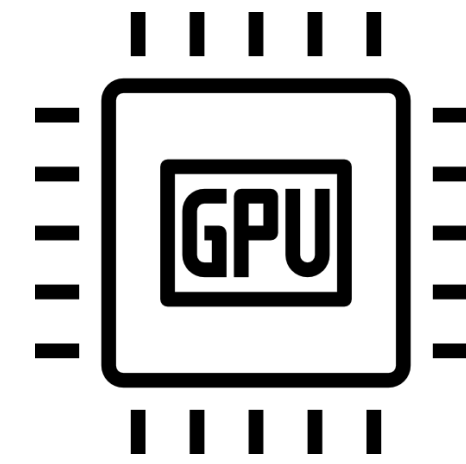
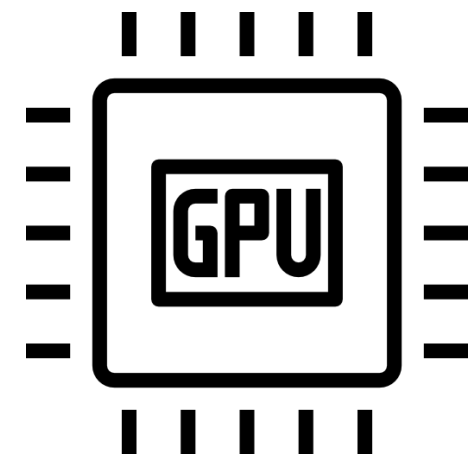
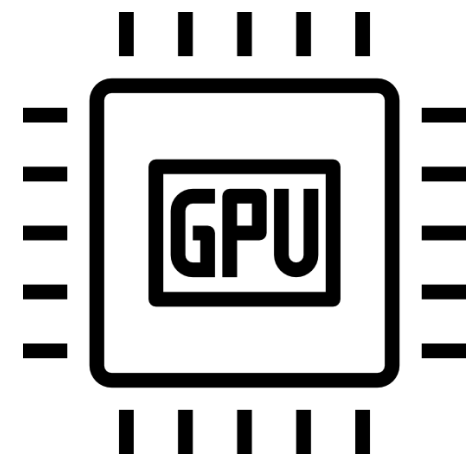
# Networks inside networks



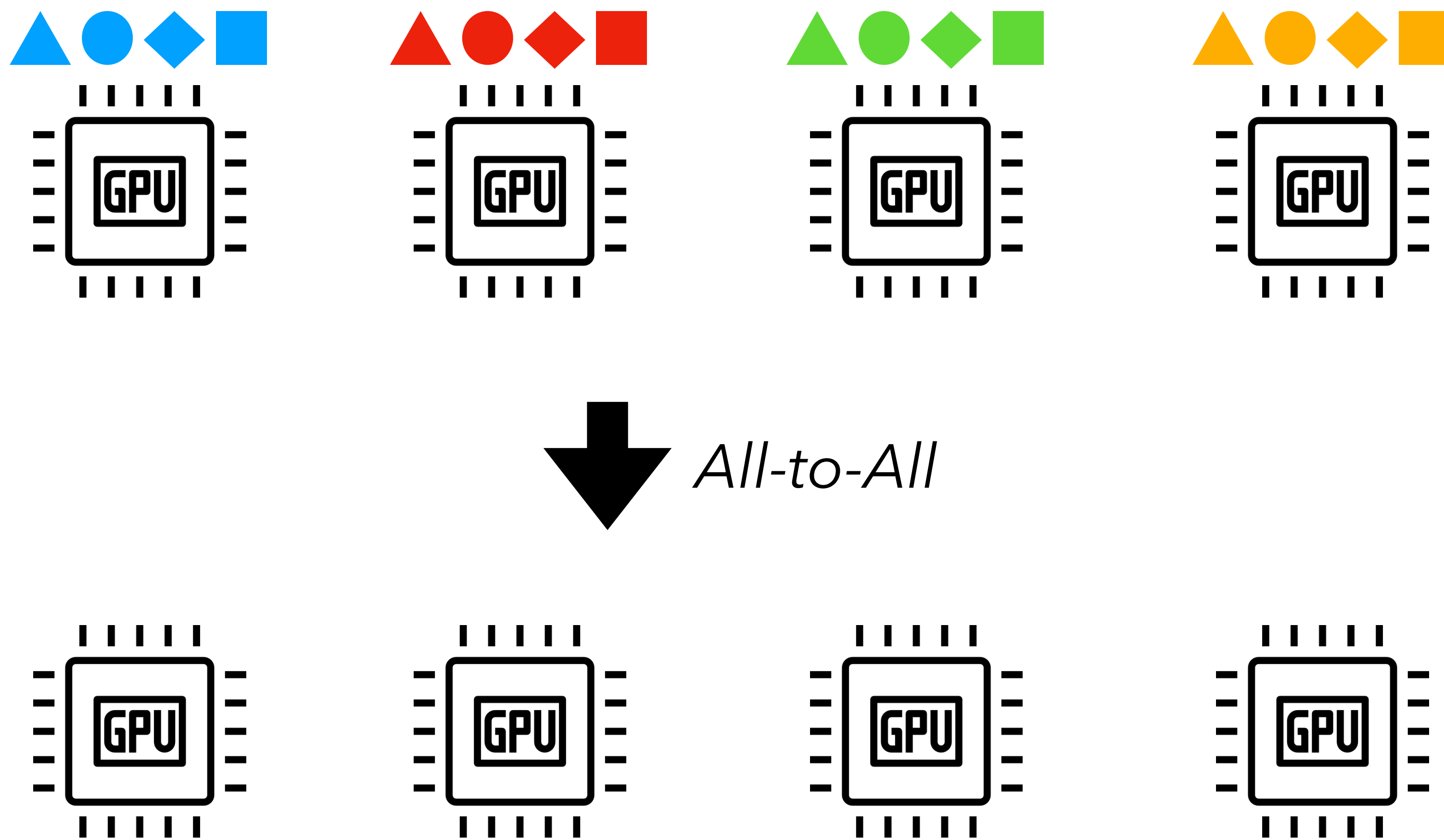
# Networks inside networks



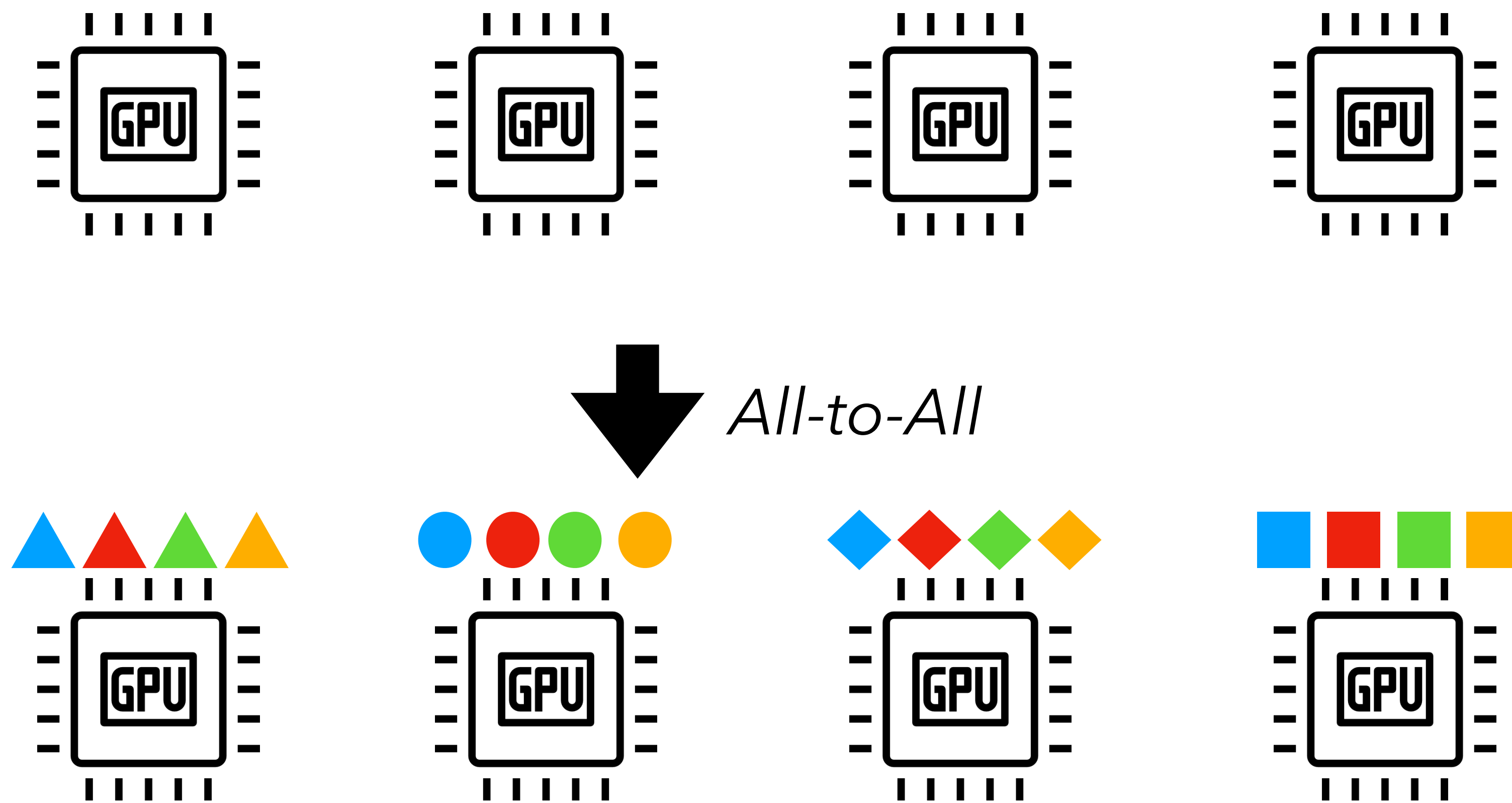
# All-to-All arises on top of these networks



# All-to-All arises on top of these networks



# All-to-All arises on top of these networks





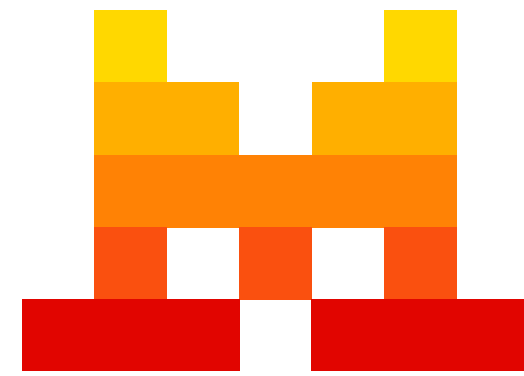
All-to-All is expensive in many systems

# All-to-All is expensive in many systems

## Mixture-of-Expert



DeepSeek



Mixtral

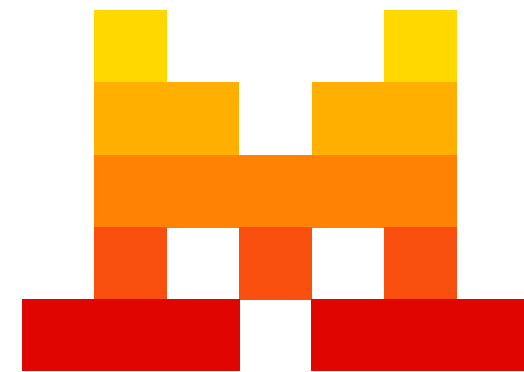
30 - 56%  
of training time

# All-to-All is expensive in many systems

## Mixture-of-Expert



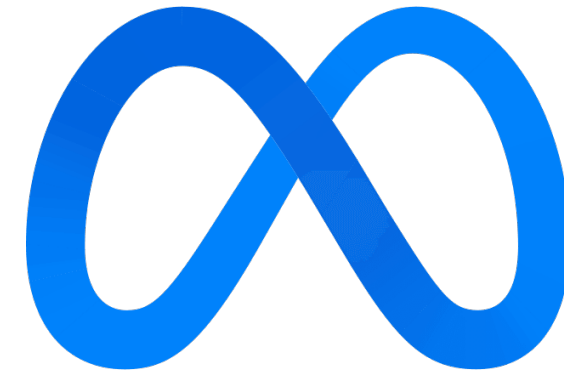
DeepSeek



Mixtral

30 - 56%  
of training time

## Ranking



Recommendation  
models in Meta

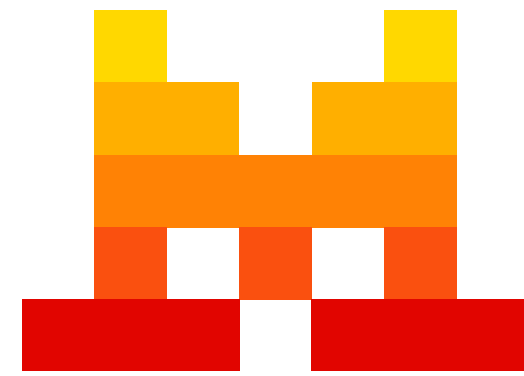
55 - 82%  
of total collectives

# All-to-All is expensive in many systems

## Mixture-of-Expert



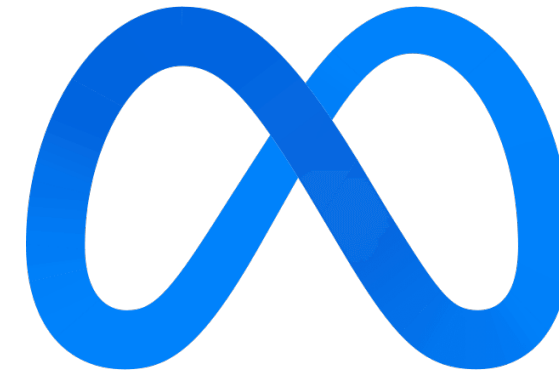
DeepSeek



Mixtral

30 - 56%  
of training time

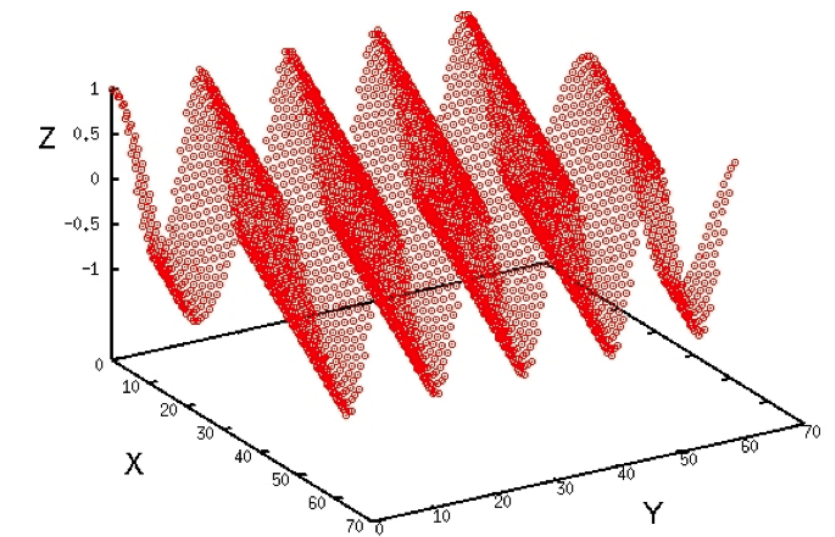
## Ranking



Recommendation  
models in Meta

55 - 82%  
of total collectives

## HPC workloads



3D FFT

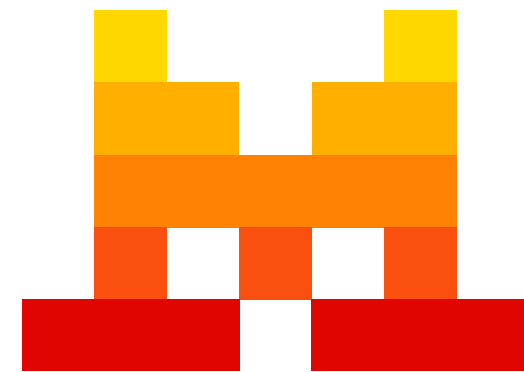
Up to 97.3%  
of runtime

# All-to-All is expensive in many systems

## Mixture-of-Expert



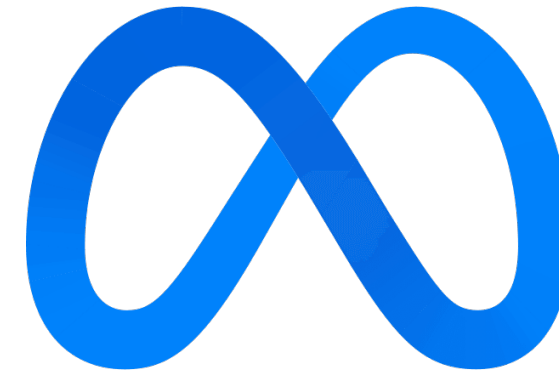
DeepSeek



Mixtral

30 - 56%  
of training time

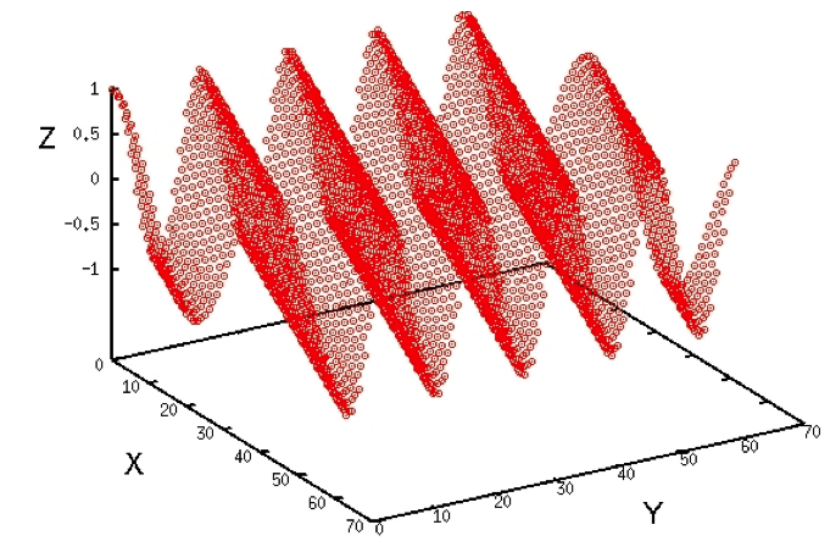
## Ranking



Recommendation  
models in Meta

55 - 82%  
of total collectives

## HPC workloads



3D FFT

Up to 97.3%  
of runtime

**We need efficient All-to-All!**

Two costs: moving data and planning move

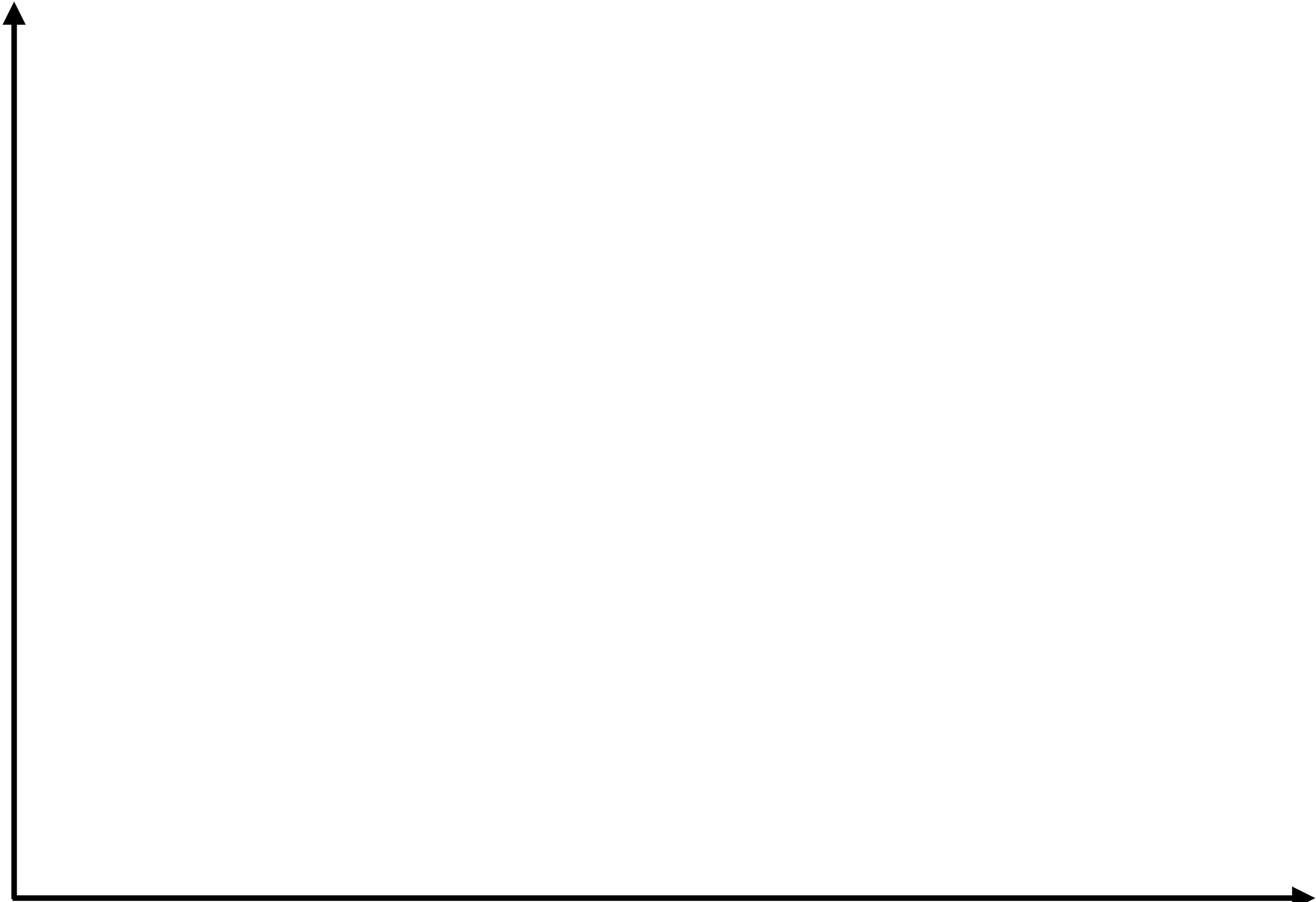
# Two costs: moving data and planning move



Transfer performance  
(moving data)

# Two costs: moving data and planning move

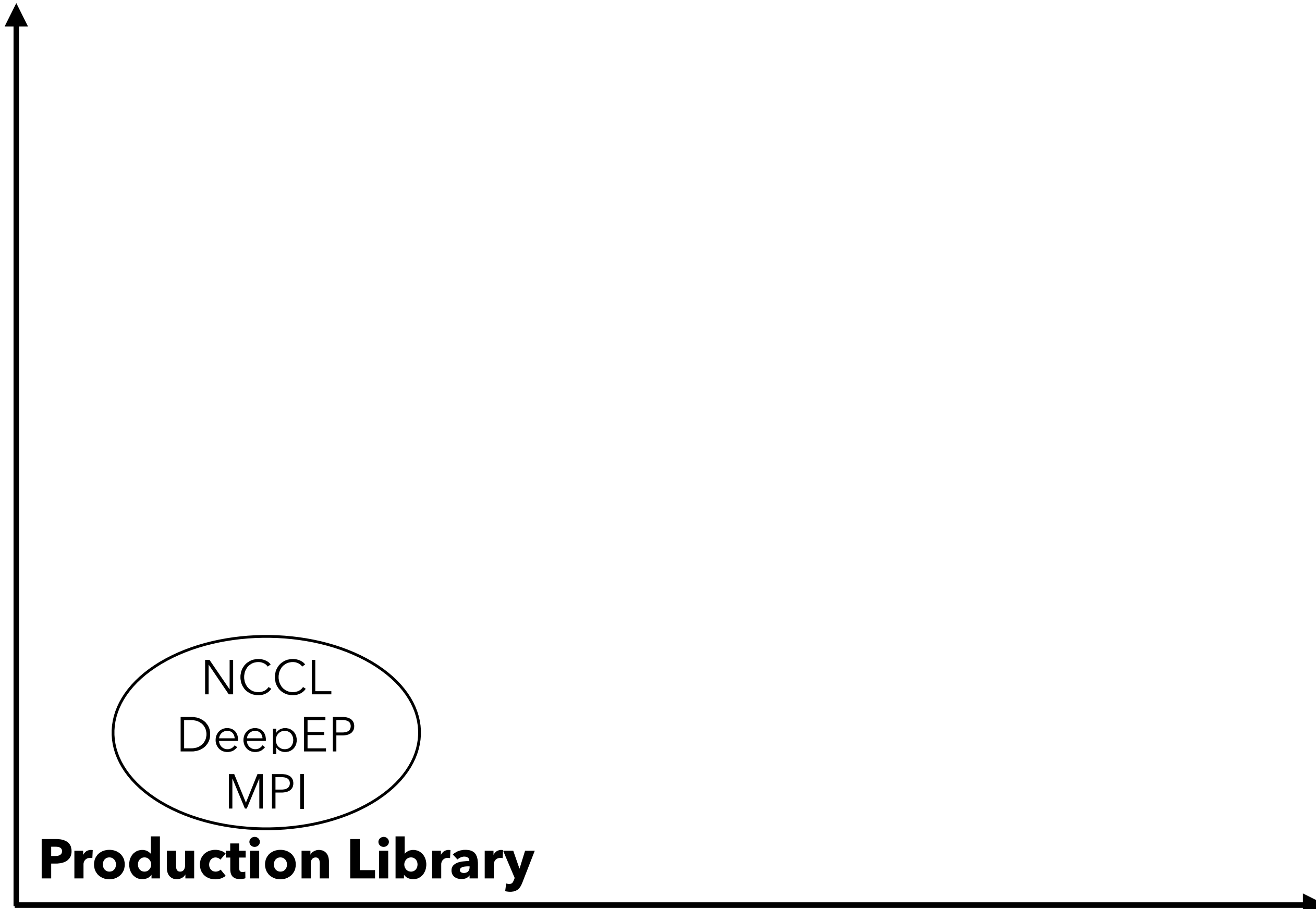
Scheduling time  
(planning move)



Transfer performance  
(moving data)

# Two costs: moving data and planning move

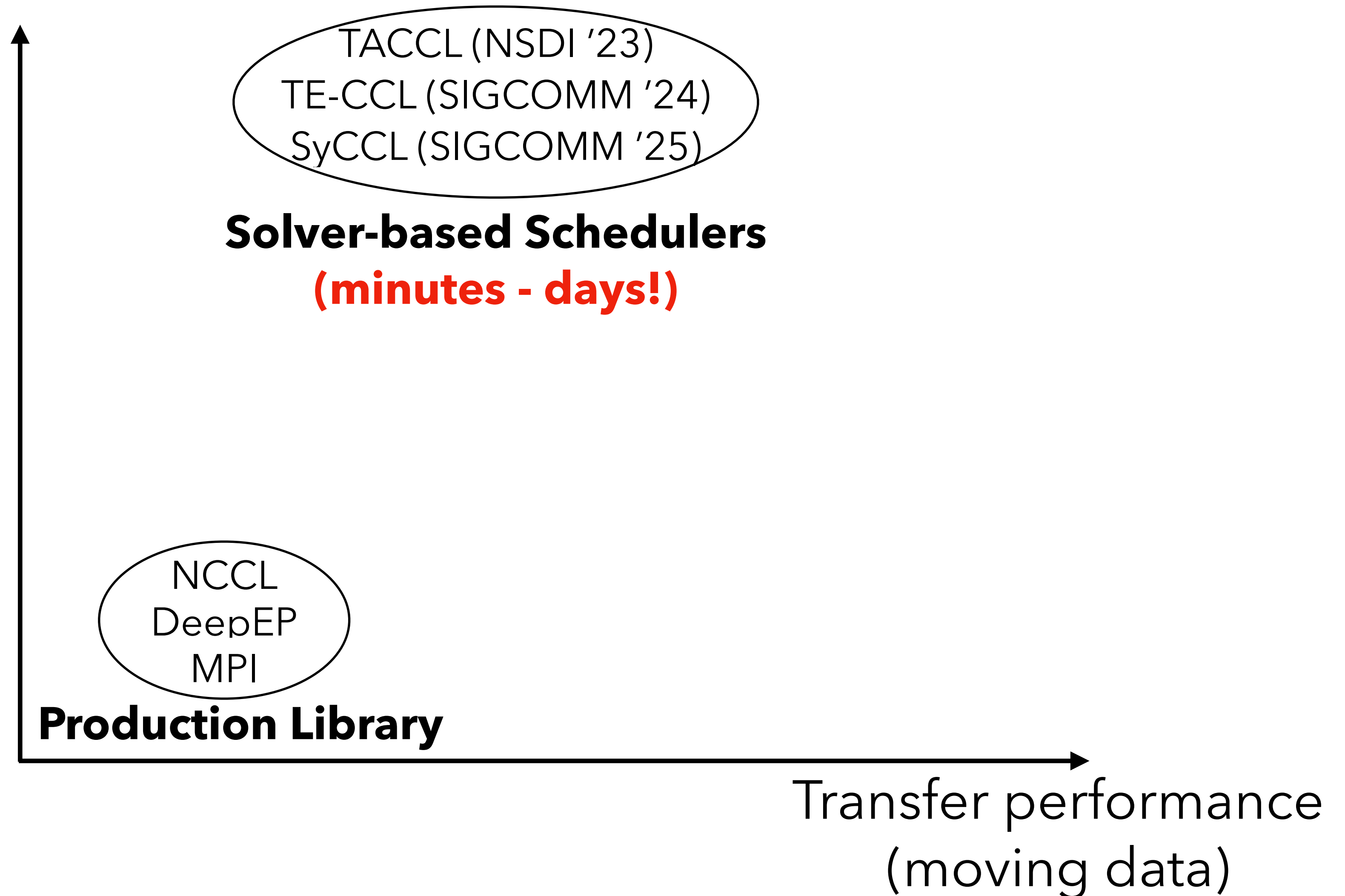
Scheduling time  
(planning move)



Transfer performance  
(moving data)

# Two costs: moving data and planning move

Scheduling time  
(planning move)



# Two costs: moving data and planning move

Scheduling time  
(planning move)

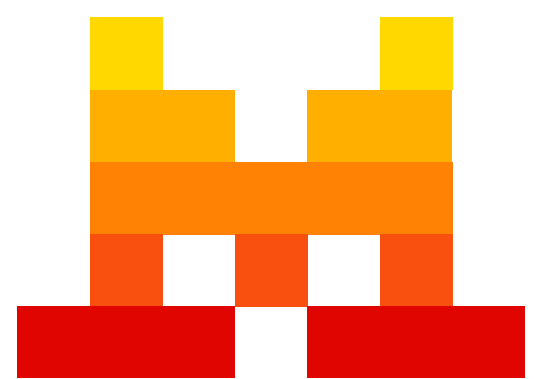
TACCL (NSDI '23)  
TE-CCL (SIGCOMM '24)  
SyCCL (SIGCOMM '25)

**Solver-based Schedulers**  
**(minutes - days!)**

NCCL  
DeepEP  
MPI

**Production Library**

Transfer performance  
(moving data)



In MoE, **skewed** traffic,  
refreshed every few **100 ms**

# Two costs: moving data and planning move

Scheduling time  
(planning move)

TACCL (NSDI '23)  
TE-CCL (SIGCOMM '24)  
SyCCL (SIGCOMM '25)

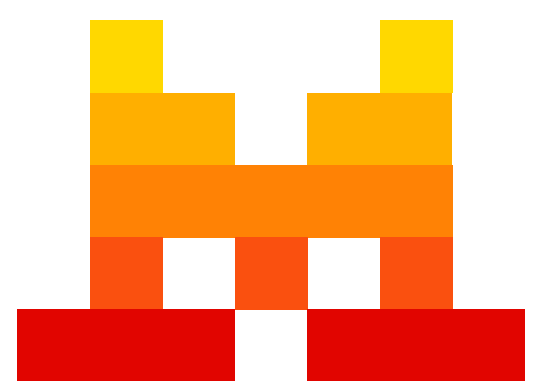
**Solver-based Schedulers**  
**(minutes - days!)**

NCCL  
DeepEP  
MPI

**Production Library**

★  
**FAST**  
( $\mu s$  -  $ms$ )

Transfer performance  
(moving data)



In MoE, **skewed** traffic,  
refreshed every few **100 ms**

# What is a scheduler solving?

$$\text{Minimize } time \quad (1)$$

$$time \geq start[c, r] \quad \forall (c, r) \in coll.postcondition \quad (2)$$

$$start[c, r] = 0 \quad \forall (c, r) \in coll.precondition \quad (3)$$

$$send[c, src, r] \geq start[c, src] \quad \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (4)$$

$$is\_sent[c, src, r] \rightarrow start[c, r] = send[c, src, r] + lat(src, r) \\ \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (5)$$

$$time \geq \sum_{c \in C} (lat(src, r) * is\_sent[c, src, r]) \quad \forall (src, r) \in \mathcal{L} \quad (6)$$

$$time \geq \sum_{c \in C} \sum_{dst \in \mathcal{S}_r^{send}} (lat(r, dst) * is\_sent[c, r, dst]) \quad \forall r \in \mathcal{S}_{send} \quad (7)$$

$$time \geq \sum_{c \in C} \sum_{src \in \mathcal{S}_r^{recv}} (lat(src, r) * is\_sent[c, src, r]) \quad \forall r \in \mathcal{S}_{recv} \quad (8)$$

$$is\_util[src, r] \geq is\_sent[c, src, r] \quad \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (9)$$

$$is\_util[src, r] \leq \sum_{c \in C} is\_sent[c, src, r] \quad \forall (src, r) \in \mathcal{L} \quad (10)$$

$$\text{Minimize } time + \gamma \times \left( \sum_{(src, r): \text{switched links}} is\_util[src, r] \right) \quad (11)$$

$$start[c, r] = start[\hat{c}, \hat{r}] \quad (12)$$

$$send[c, src, r] = send[\hat{c}, \hat{src}, \hat{r}] \quad (13)$$

$$is\_sent[c, src, r] = is\_sent[\hat{c}, \hat{src}, \hat{r}] \quad (14)$$

$$\sum_{(r_1, r_2) \in \mathcal{L}: r_1 \in node_1, r_2 \in node_2} is\_sent[c, r_1, r_2] \geq 1 \quad (15)$$

# What is a scheduler solving?

$$\begin{aligned}
 & \text{Minimize } time & (1) \\
 & time \geq start[c, r] \quad \forall (c, r) \in coll.postcondition & (2) \\
 & start[c, r] = 0 \quad \forall (c, r) \in coll.precondition & (3) \\
 & send[c, src, r] \geq start[c, src] \quad \forall c \in C \quad \forall (src, r) \in \mathcal{L} & (4) \\
 & is\_sent[c, src, r] \rightarrow start[c, r] = send[c, src, r] + lat(src, r) & \\
 & \quad \forall c \in C \quad \forall (src, r) \in \mathcal{L} & (5) \\
 & time \geq \sum_{c \in C} (lat(src, r) * is\_sent[c, src, r]) \quad \forall (src, r) \in \mathcal{L} & (6) \\
 & time \geq \sum_{c \in C} \sum_{dst \in \mathcal{S}_r^{send}} (lat(r, dst) * is\_sent[c, r, dst]) \quad \forall r \in \mathcal{S}_{send} & (7) \\
 & time \geq \sum_{c \in C} \sum_{src \in \mathcal{S}_r^{recv}} (lat(src, r) * is\_sent[c, src, r]) \quad \forall r \in \mathcal{S}_{recv} & (8) \\
 & is\_util[src, r] \geq is\_sent[c, src, r] \quad \forall c \in C \forall (src, r) \in \mathcal{L} & (9) \\
 & is\_util[src, r] \leq \sum_{c \in C} is\_sent[c, src, r] \quad \forall (src, r) \in \mathcal{L} & (10) \\
 & \text{Minimize } time + \gamma \times \left( \sum_{(src, r): \text{switched links}} is\_util[src, r] \right) & (11) \\
 & \quad start[c, r] = start[\hat{c}, \hat{r}] & (12) \\
 & \quad send[c, src, r] = send[\hat{c}, \hat{src}, \hat{r}] & (13) \\
 & \quad is\_sent[c, src, r] = is\_sent[\hat{c}, \hat{src}, \hat{r}] & (14) \\
 & \quad \sum_{(r_1, r_2) \in \mathcal{L}: r_1 \in node_1, r_2 \in node_2} is\_sent[c, r_1, r_2] \geq 1 & (15)
 \end{aligned}$$

**Who sends what, to whom, through who, and when?**



Why hard#1: last flow determines completion

# Why hard#1: last flow determines completion



# Why hard#1: last flow determines completion



Completion = when **longest** flow finish

# Why hard#1: last flow determines completion

$$\text{Minimize } time \quad (1)$$

$$time \geq start[c, r] \quad \forall (c, r) \in coll.postcondition \quad (2)$$

$$start[c, r] = 0 \quad \forall (c, r) \in coll.precondition \quad (3)$$

$$send[c, src, r] \geq start[c, src] \quad \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (4)$$

$$is\_sent[c, src, r] \rightarrow start[c, r] = send[c, src, r] + lat(src, r) \\ \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (5)$$

$$time \geq \sum_{c \in C} (lat(src, r) * is\_sent[c, src, r]) \quad \forall (src, r) \in \mathcal{L} \quad (6)$$

$$time \geq \sum_{c \in C} \sum_{dst \in \mathcal{S}_r^{send}} (lat(r, dst) * is\_sent[c, r, dst]) \quad \forall r \in \mathcal{S}_{send} \quad (7)$$

$$time \geq \sum_{c \in C} \sum_{src \in \mathcal{S}_r^{recv}} (lat(src, r) * is\_sent[c, src, r]) \quad \forall r \in \mathcal{S}_{recv} \quad (8)$$

$$is\_util[src, r] \geq is\_sent[c, src, r] \quad \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (9)$$

$$is\_util[src, r] \leq \sum_{c \in C} is\_sent[c, src, r] \quad \forall (src, r) \in \mathcal{L} \quad (10)$$

$$\text{Minimize } time + \gamma \times \left( \sum_{(src, r): \text{switched links}} is\_util[src, r] \right) \quad (11)$$

$$start[c, r] = start[\hat{c}, \hat{r}] \quad (12)$$

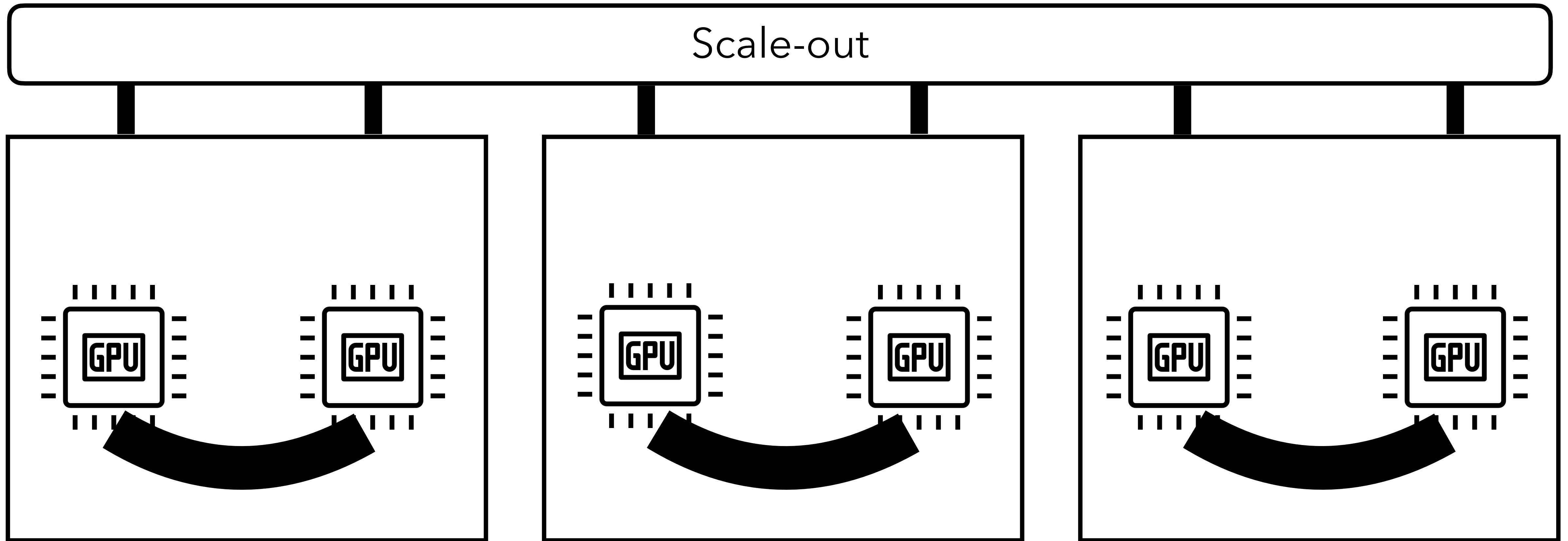
$$send[c, src, r] = send[\hat{c}, \hat{src}, \hat{r}] \quad (13)$$

$$is\_sent[c, src, r] = is\_sent[\hat{c}, \hat{src}, \hat{r}] \quad (14)$$

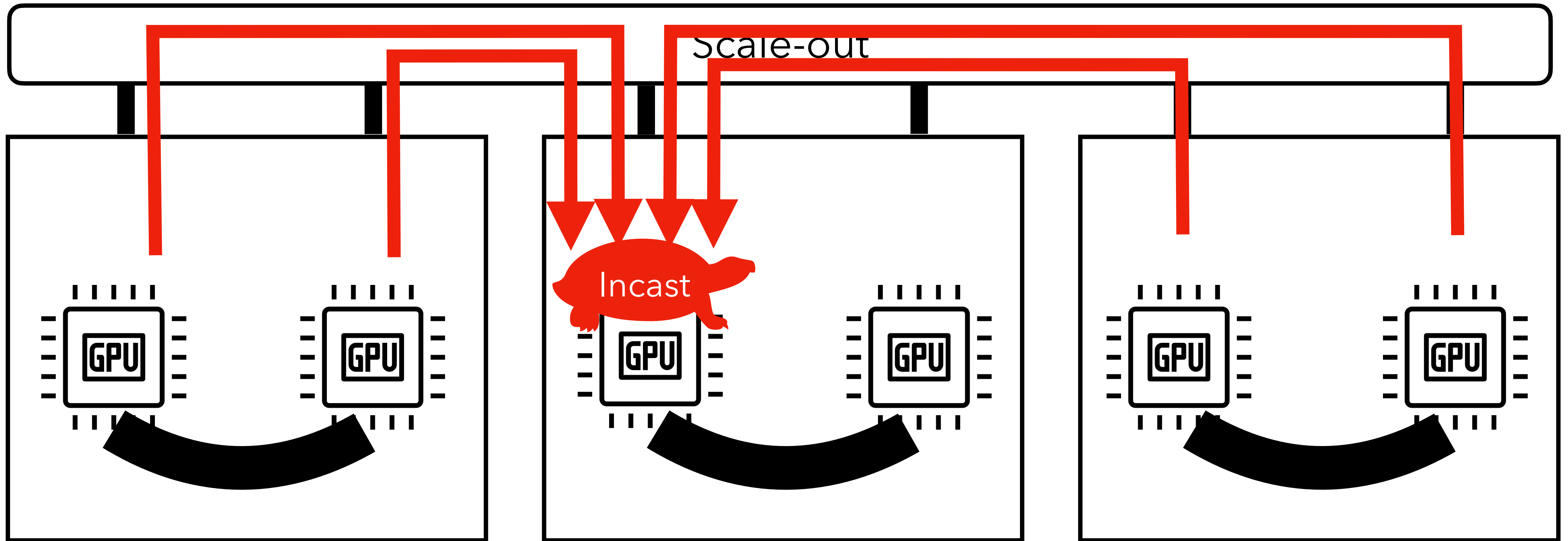
$$\sum_{(r_1, r_2) \in \mathcal{L}: r_1 \in node_1, r_2 \in node_2} is\_sent[c, r_1, r_2] \geq 1 \quad (15)$$

gglr

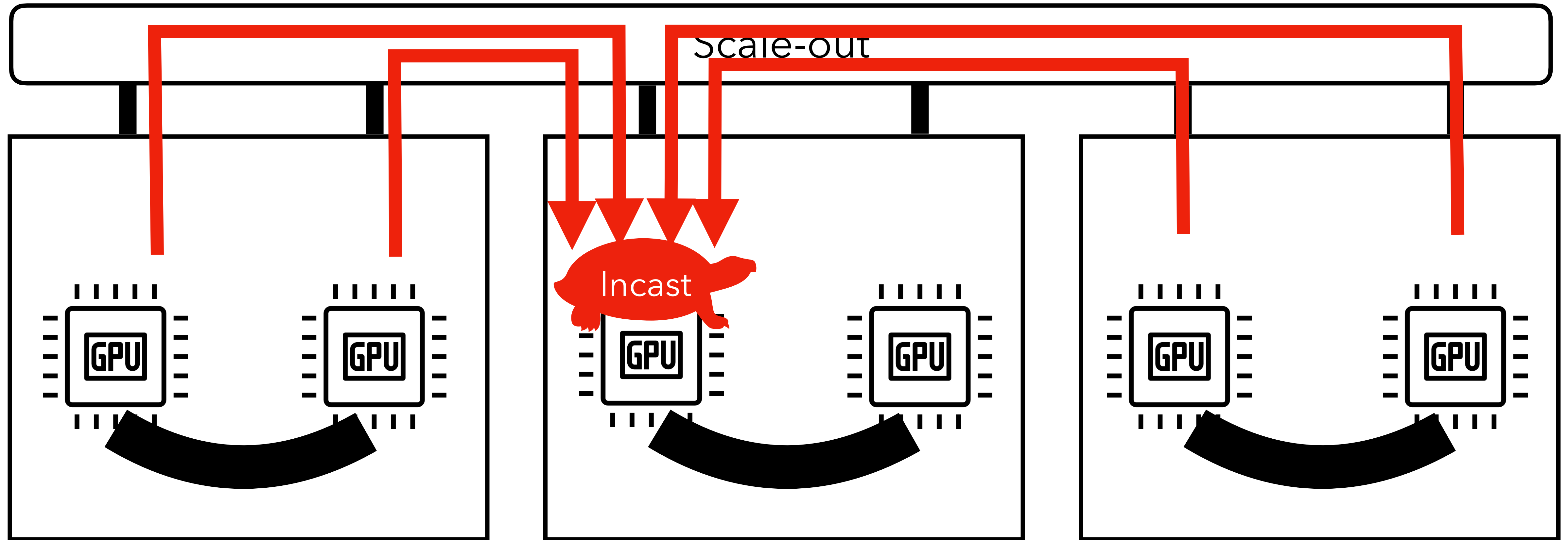
# Why hard#2: incast creates stragglers



# Why hard#2: incast creates stragglers



# Why hard#2: incast creates stragglers



Avoid incast: **at most one flow** per sender, per receiver

# Why hard#2: incast creates stragglers

$$\text{Minimize } time \quad (1)$$

$$time \geq start[c, r] \quad \forall (c, r) \in coll.postcondition \quad (2)$$

$$start[c, r] = 0 \quad \forall (c, r) \in coll.precondition \quad (3)$$

$$send[c, src, r] \geq start[c, src] \quad \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (4)$$

$$is\_sent[c, src, r] \rightarrow start[c, r] = send[c, src, r] + lat(src, r) \\ \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (5)$$

$$time \geq \sum_{c \in C} (lat(src, r) * is\_sent[c, src, r]) \quad \forall (src, r) \in \mathcal{L} \quad (6)$$

$$time \geq \sum_{c \in C} \sum_{dst \in S_r^{send}} (lat(r, dst) * is\_sent[c, r, dst]) \quad \forall r \in S_{send} \quad (7)$$

$$time \geq \sum_{c \in C} \sum_{src \in S_r^{recv}} (lat(src, r) * is\_sent[c, src, r]) \quad \forall r \in S_{recv} \quad (8)$$

$$is\_util[src, r] \geq is\_sent[c, src, r] \quad \forall c \in C \quad \forall (src, r) \in \mathcal{L} \quad (9)$$

$$is\_util[src, r] \leq \sum_{c \in C} is\_sent[c, src, r] \quad \forall (src, r) \in \mathcal{L} \quad (10)$$

$$\text{Minimize } time + \gamma \times \left( \sum_{(src, r): \text{switched links}} is\_util[src, r] \right) \quad (11)$$

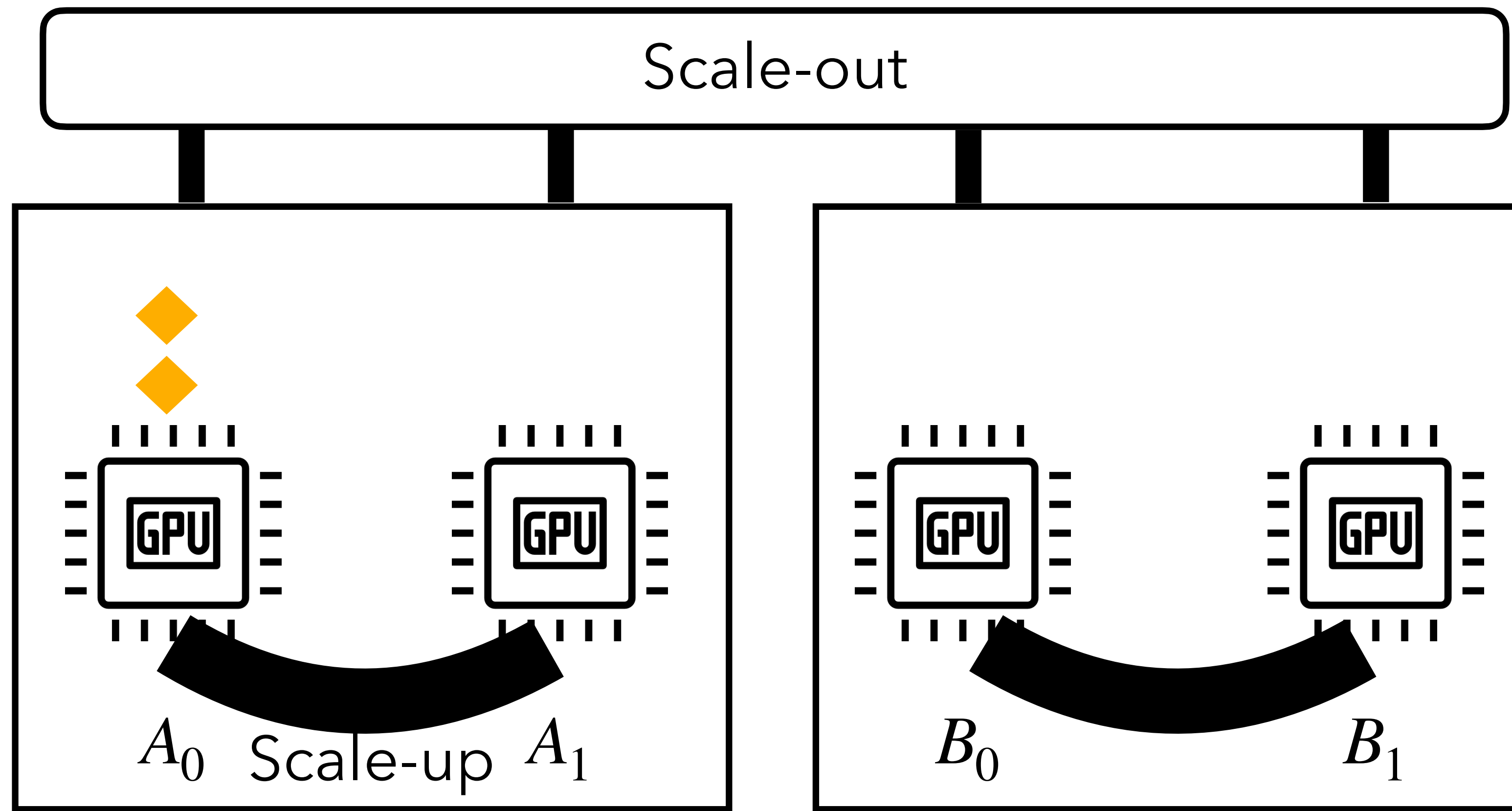
$$start[c, r] = start[\hat{c}, \hat{r}] \quad (12)$$

$$send[c, src, r] = send[\hat{c}, \hat{src}, \hat{r}] \quad (13)$$

$$is\_sent[c, src, r] = is\_sent[\hat{c}, \hat{src}, \hat{r}] \quad (14)$$

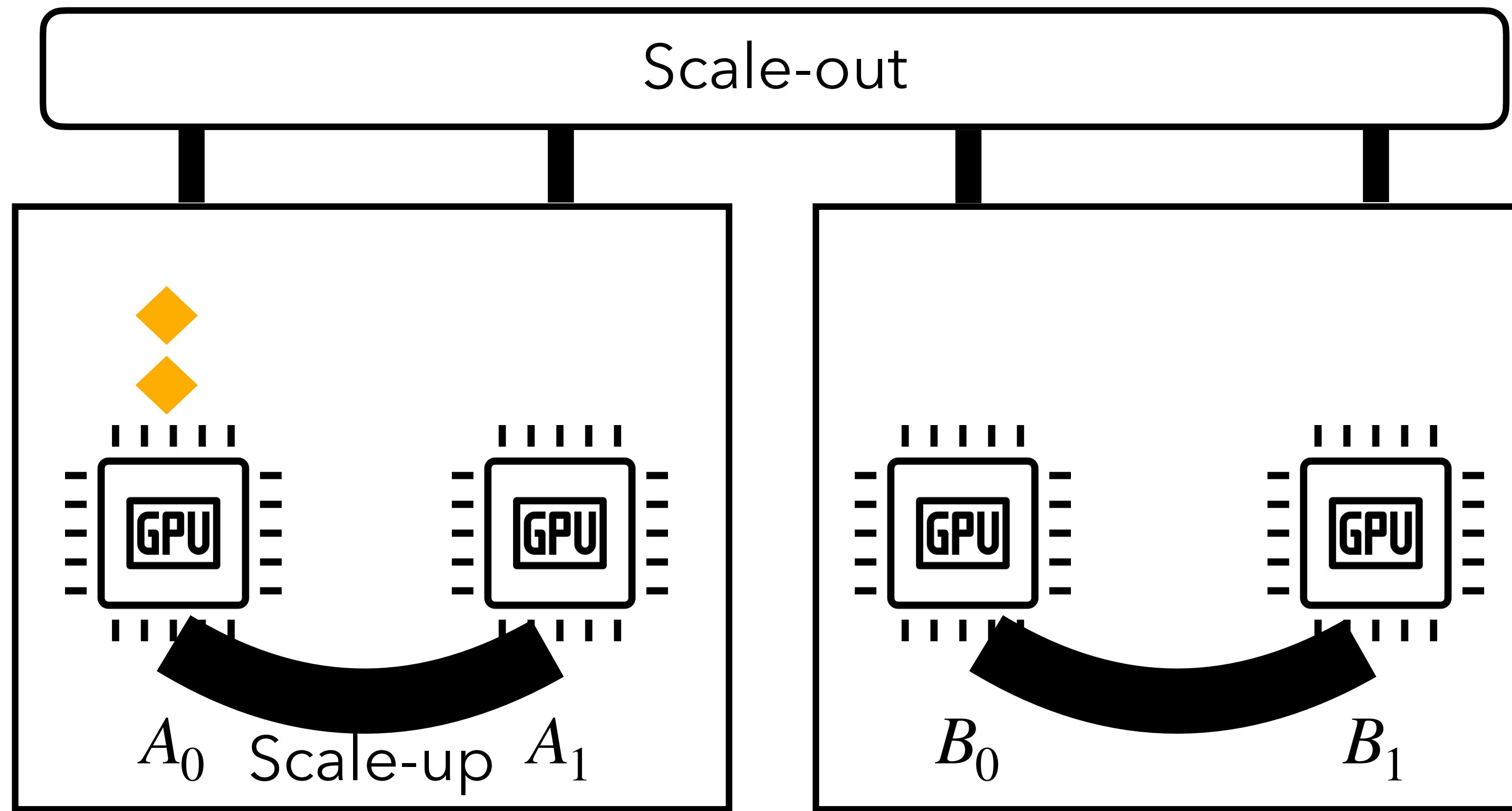
$$\sum_{(r_1, r_2) \in \mathcal{L}: r_1 \in node_1, r_2 \in node_2} is\_sent[c, r_1, r_2] \geq 1 \quad (15)$$

# Why hard#3: waypointing explodes the decision space



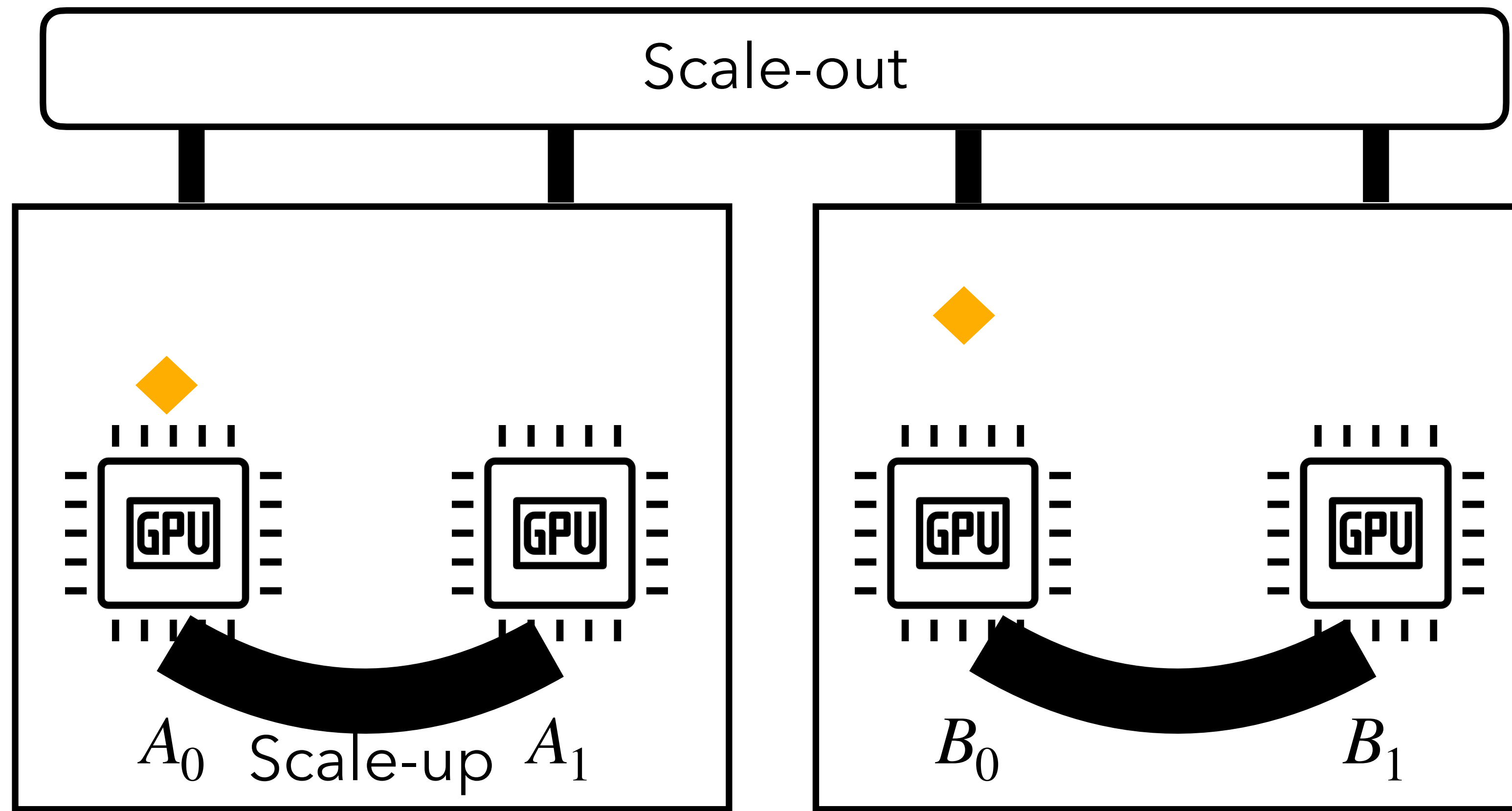
# Why hard#3: waypointing explodes the decision space

$A_0$  send to  $B_0$

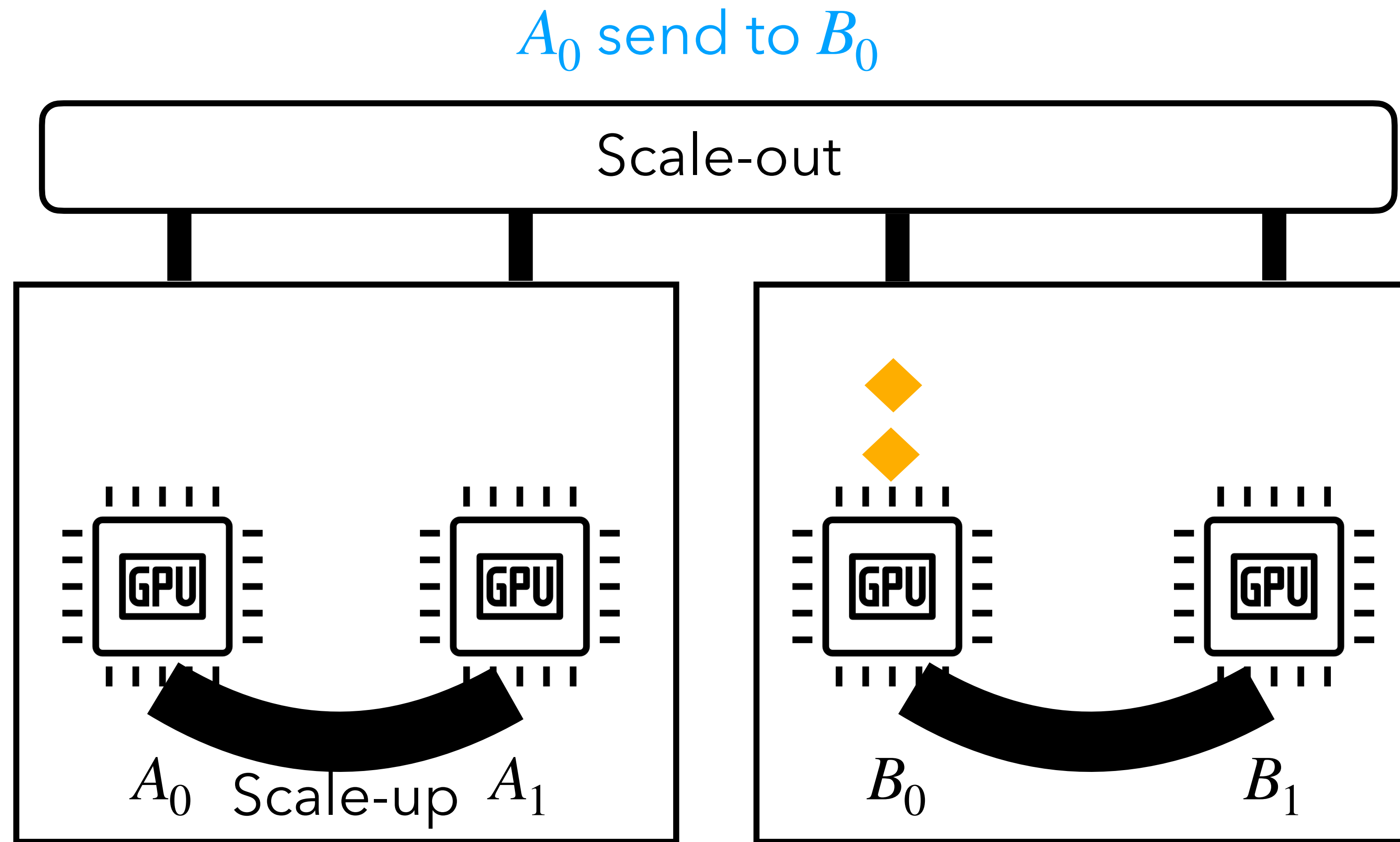


# Why hard#3: waypointing explodes the decision space

$A_0$  send to  $B_0$

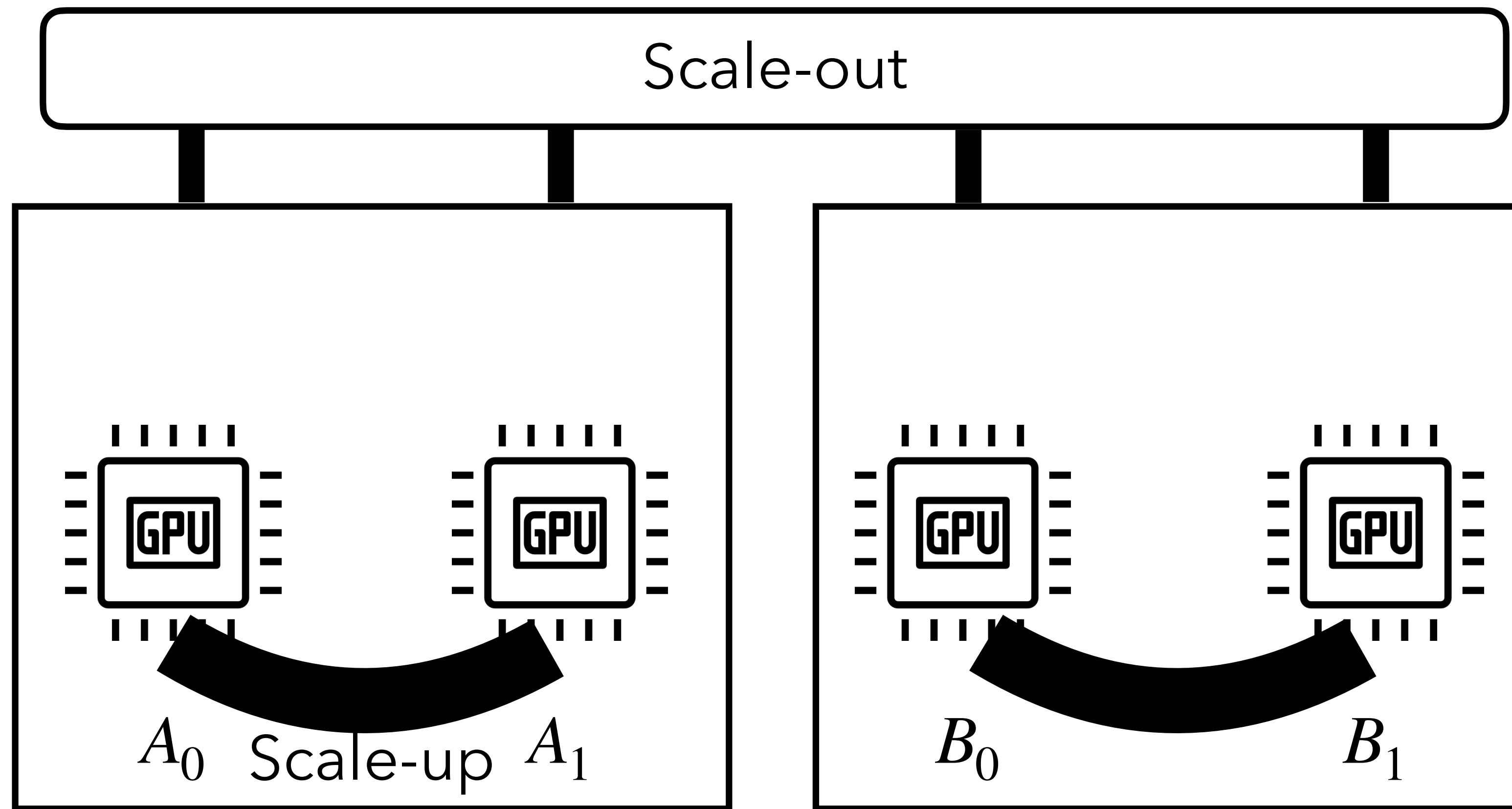


# Why hard#3: waypointing explodes the decision space



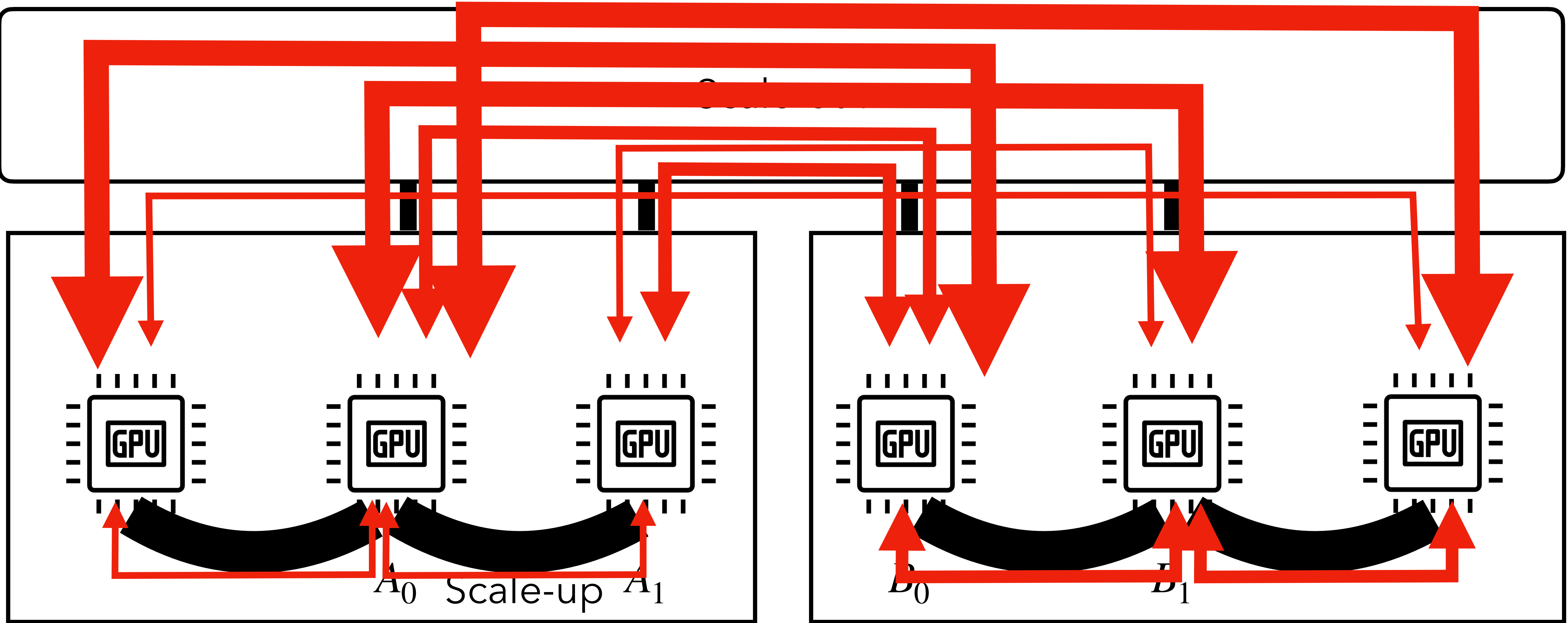
# Why hard#3: waypointing explodes the decision space

$A_0$  send to  $B_0$

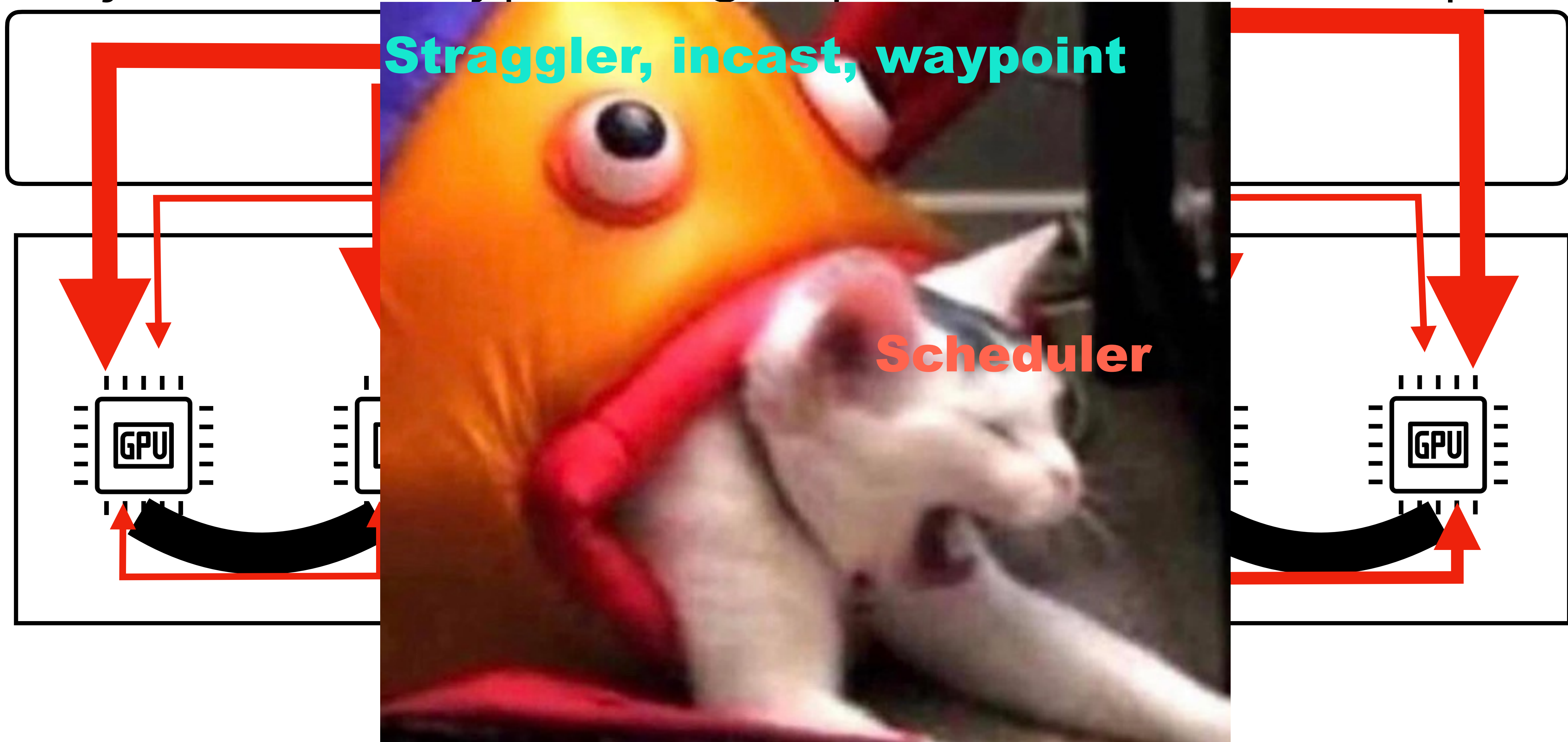


$$2 \times \frac{\text{amount of data waypointed}}{\text{scale-up BW}} + \frac{\text{remaining data not waypointed}}{\text{scale-out BW}}$$

# Why hard#3: waypointing explodes the decision space

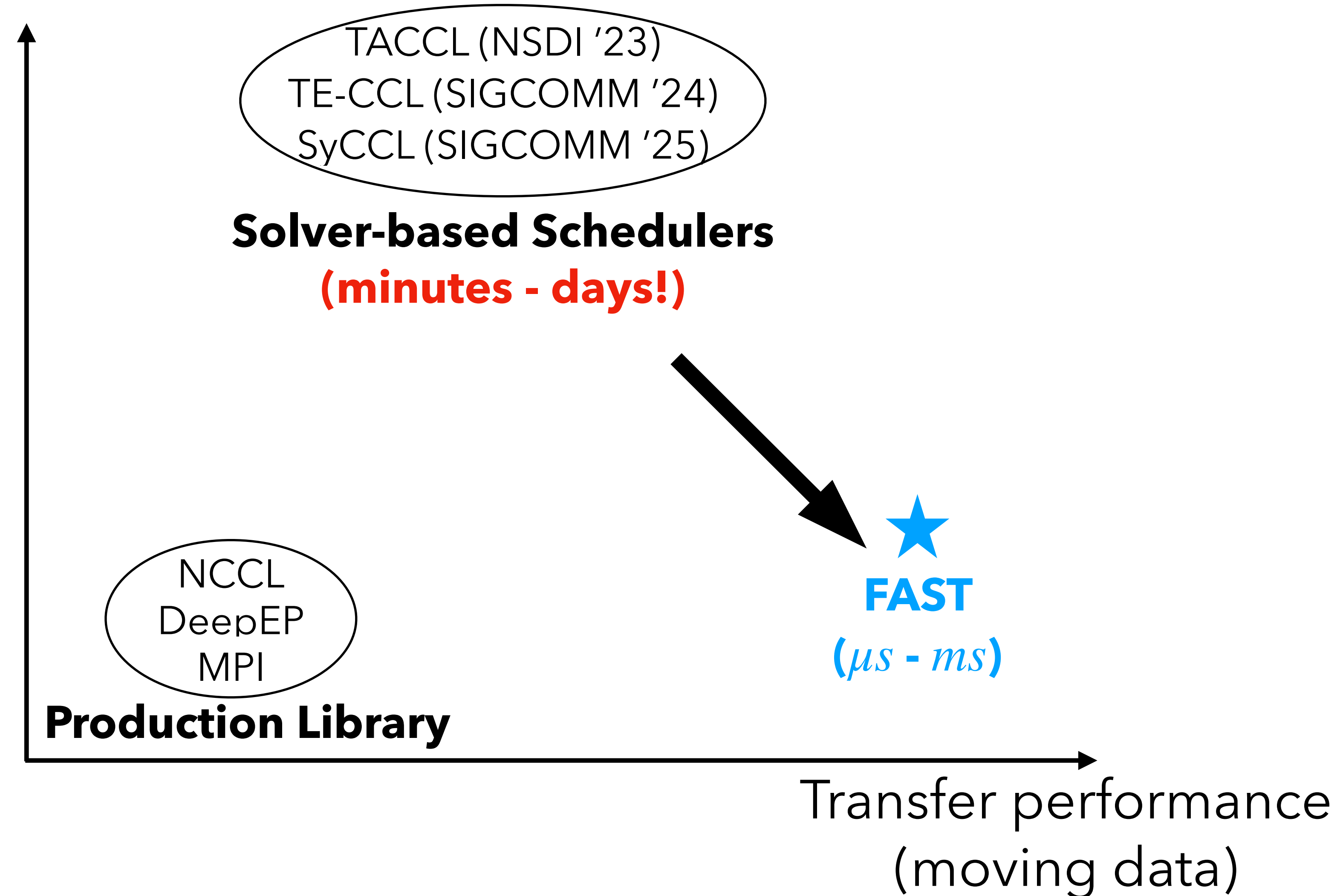


# Why hard#3: waypointing explodes the decision space



# How do we get there?

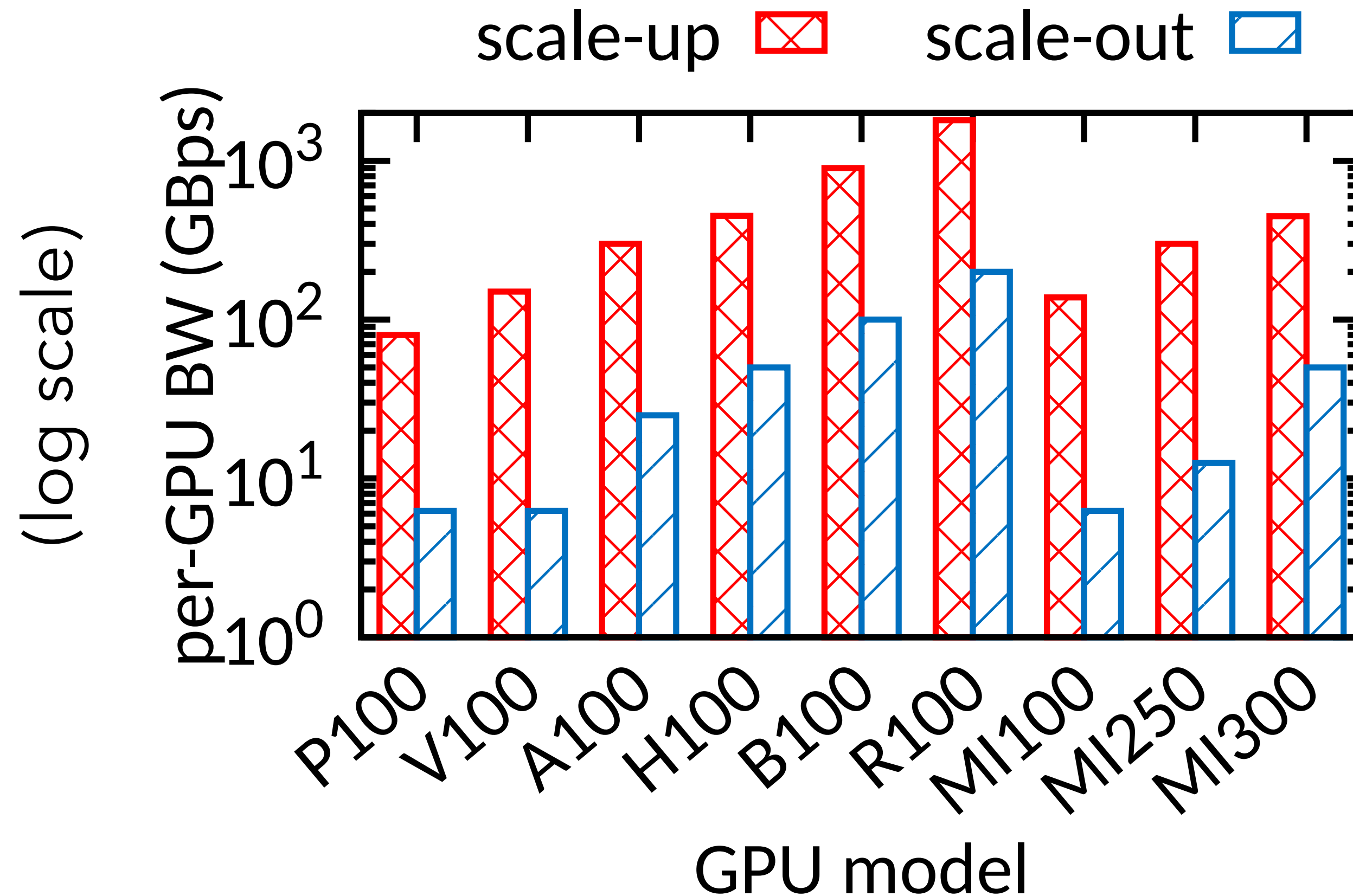
Scheduling time  
(planning move)



Pretend scale-up is free

# Scale-up is “almost free”

Scale-up bandwidth  $\approx$  9x scale-out



Scale-out network is bottleneck  
when scale-up is infinitely fast

Scale-out network is bottleneck  
when scale-up is infinitely fast

(Optimal scale-out schedule)

**Global Scheduling**

Scale-out network is bottleneck  
when scale-up is infinitely fast

(Optimal scale-out schedule)

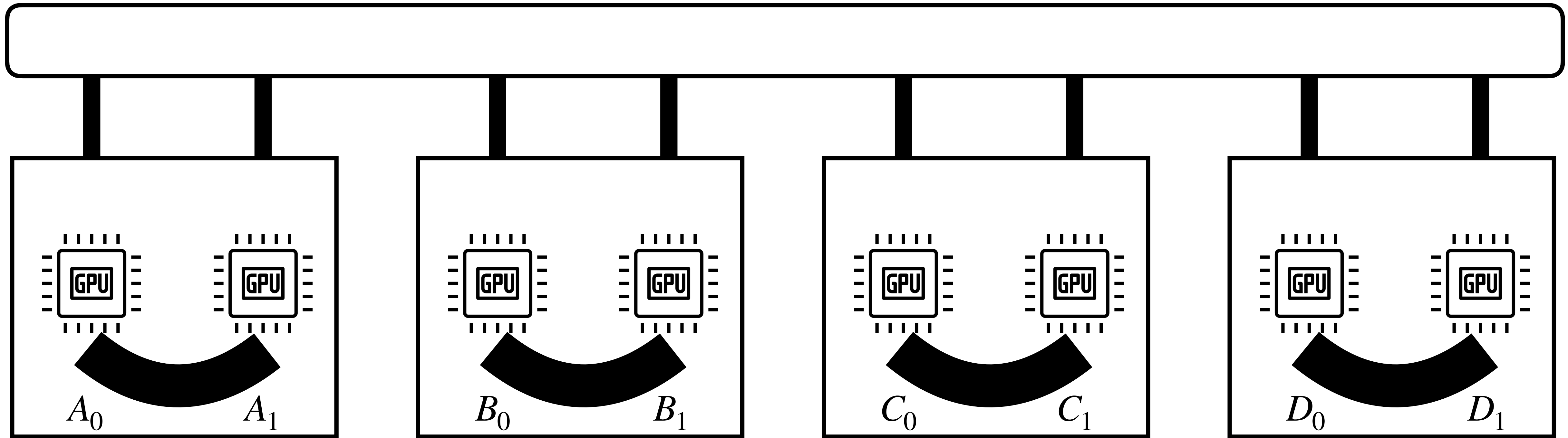
**Global Scheduling**



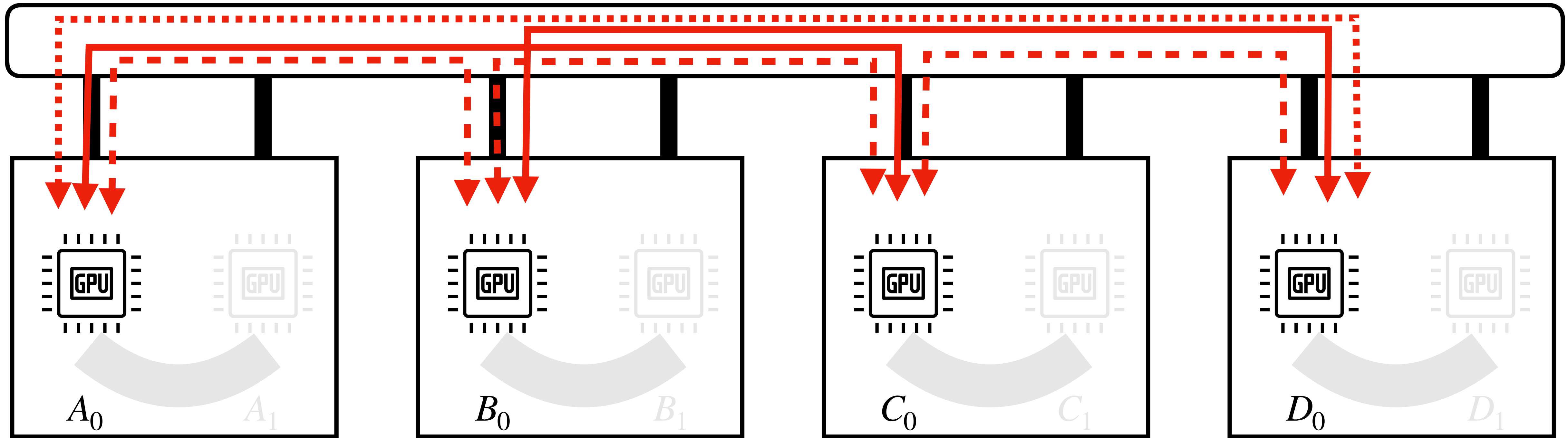
Local Scheduling

(Further speedup using scale-up)

# Optimal scale-out transfers with no incast

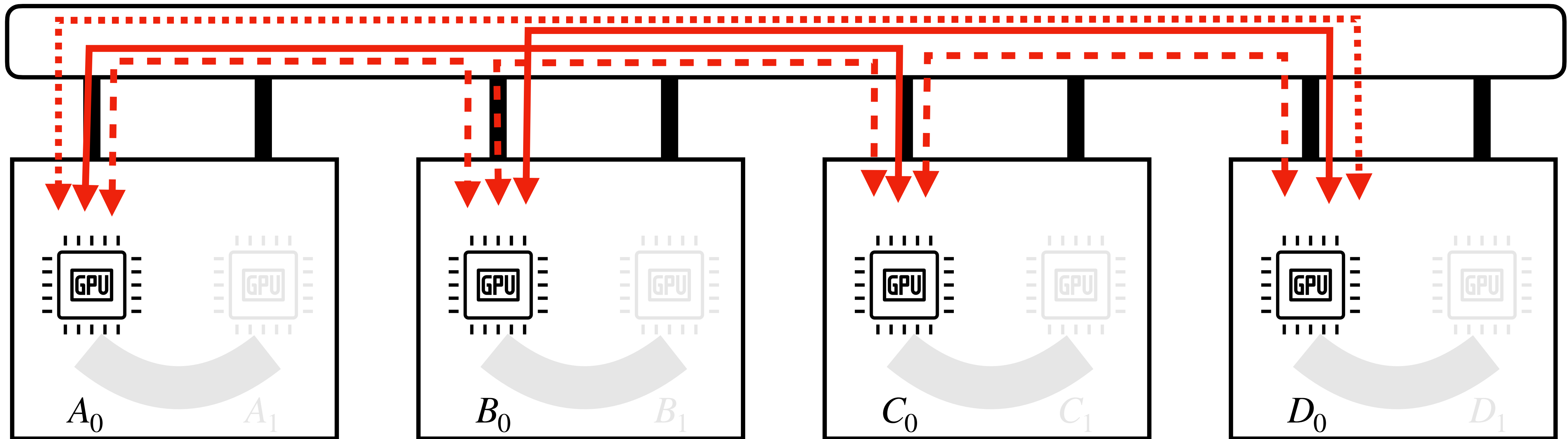


# Optimal scale-out transfers with no incast



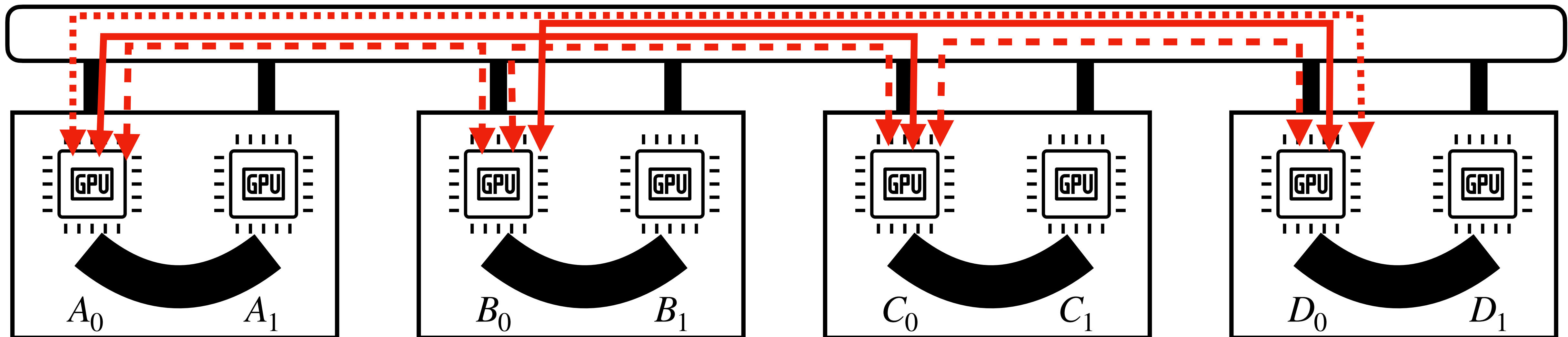
# Optimal scale-out transfers with no incast

Birkhoff-Von Neumann Decomposition, 1946



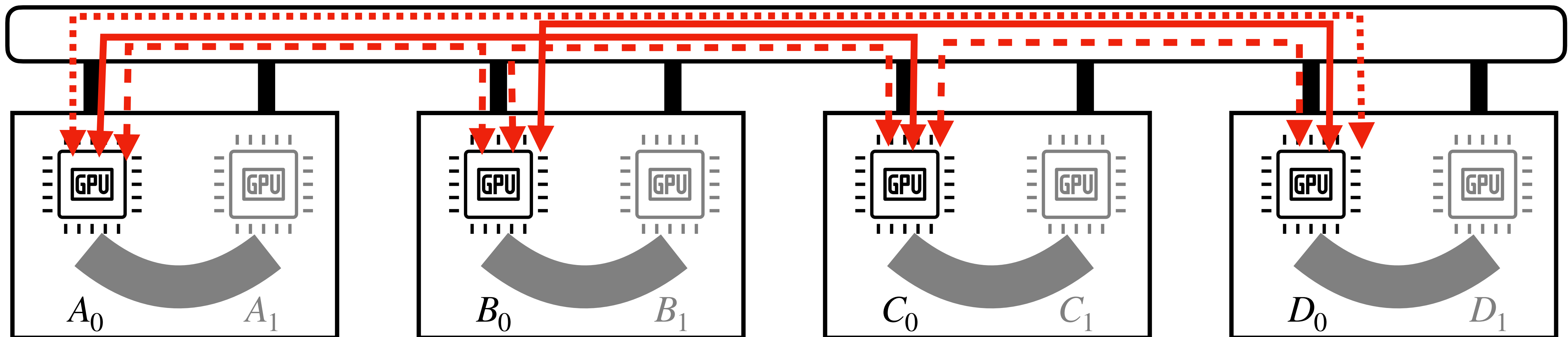
# Example of incast-free transfer stages

		Receive			
		$A_0$	$B_0$	$C_0$	$D_0$
Send	$A_0$	0	0	3	2
	$B_0$	5	0	0	4
	$C_0$	4	2	0	3
	$D_0$	0	7	2	0



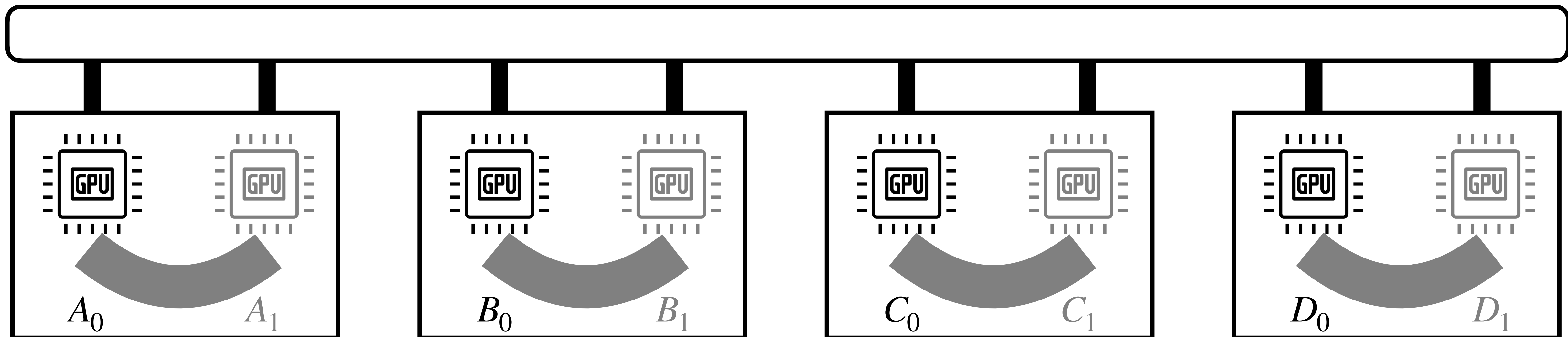
# Example of incast-free transfer stages

		Receive			
		$A_0$	$B_0$	$C_0$	$D_0$
Send	$A_0$	0	0	3	2
	$B_0$	5	0	0	4
	$C_0$	4	2	0	3
	$D_0$	0	7	2	0



# Example of incast-free transfer stages

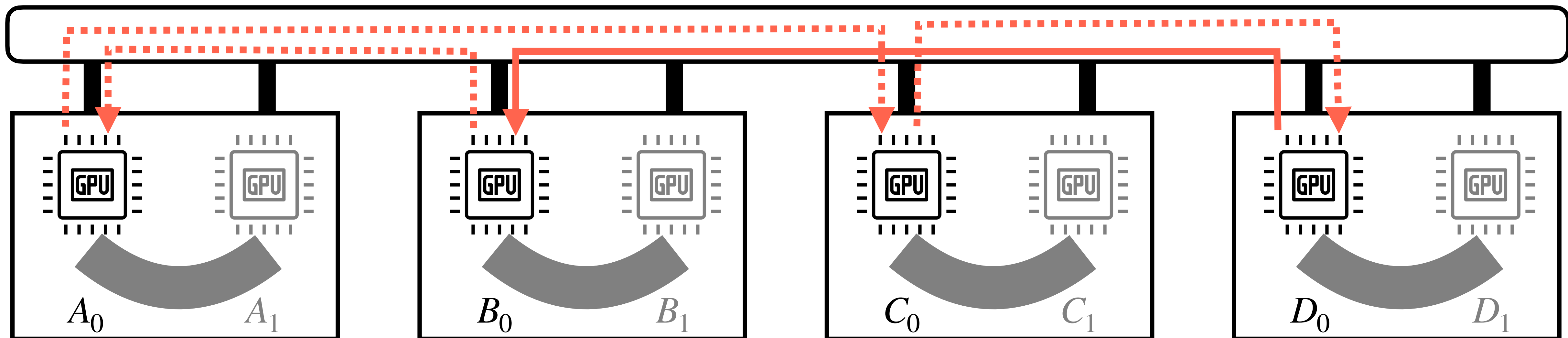
		<i>Receive</i>											
		$A_0$	$B_0$	$C_0$	$D_0$								
<i>Send</i>	$A_0$	0	0	3	2	=							
	$B_0$	5	0	0	4								
	$C_0$	4	2	0	3								
	$D_0$	0	7	2	0								
						①							
						0	0	3	0	+			
						3	0	0	0				
						0	0	0	3				
						0	3	0	0				
						②							
						0	0	0	2	+			
						2	0	0	0				
						0	2	0	0				
						0	0	2	0				
						③							
						0	0	0	0	+			
						0	0	0	4				
						4	0	0	0				
						0	4	0	0				



# Example of incast-free transfer stages

		Receive																																																																		
		$A_0$	$B_0$	$C_0$	$D_0$																																																															
Send	$A_0$	0	0	3	2	=	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>0</td><td>0</td><td style="background-color: #f8d7da;">3</td><td>0</td></tr> <tr><td style="background-color: #f8d7da;">3</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td style="background-color: #f8d7da;">3</td></tr> <tr><td>0</td><td style="background-color: #f8d7da;">3</td><td>0</td><td>0</td></tr> </table>				0	0	3	0	3	0	0	0	0	0	0	3	0	3	0	0	+	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>0</td><td>0</td><td>0</td><td>2</td></tr> <tr><td>2</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>2</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>2</td><td>0</td></tr> </table>				0	0	0	2	2	0	0	0	0	2	0	0	0	0	2	0	+	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>4</td></tr> <tr><td>4</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>4</td><td>0</td><td>0</td></tr> </table>				0	0	0	0	0	0	0	4	4	0	0	0	0	4	0	0
	0	0	3	0																																																																
	3	0	0	0																																																																
	0	0	0	3																																																																
0	3	0	0																																																																	
0	0	0	2																																																																	
2	0	0	0																																																																	
0	2	0	0																																																																	
0	0	2	0																																																																	
0	0	0	0																																																																	
0	0	0	4																																																																	
4	0	0	0																																																																	
0	4	0	0																																																																	
$B_0$	5	0	0	4																																																																
$C_0$	4	2	0	3																																																																
$D_0$	0	7	2	0																																																																

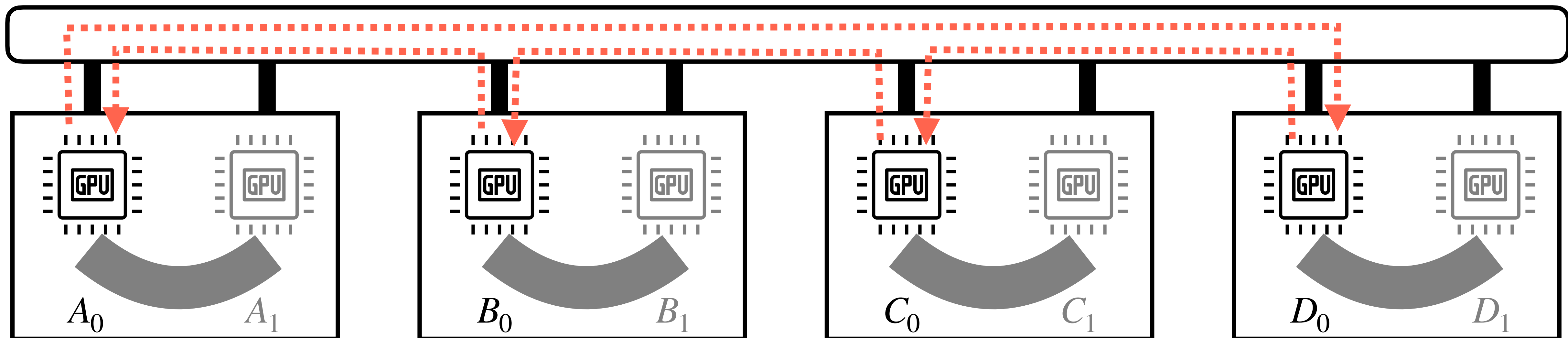
Each stage: one-to-one (incast-free), equal sized



# Example of incast-free transfer stages

		Receive											
		$A_0$	$B_0$	$C_0$	$D_0$								
Send	$A_0$	0	0	3	2	=							
	$B_0$	5	0	0	4								
	$C_0$	4	2	0	3								
	$D_0$	0	7	2	0								
						①							
						0	0	3	0	+			
						3	0	0	0				
						0	0	0	3				
						0	3	0	0				
						②							
						0	0	0	2	+			
						2	0	0	0				
						0	2	0	0				
						0	0	2	0				
						③							
						0	0	0	0	+			
						0	0	0	4				
						4	0	0	0				
						0	4	0	0				

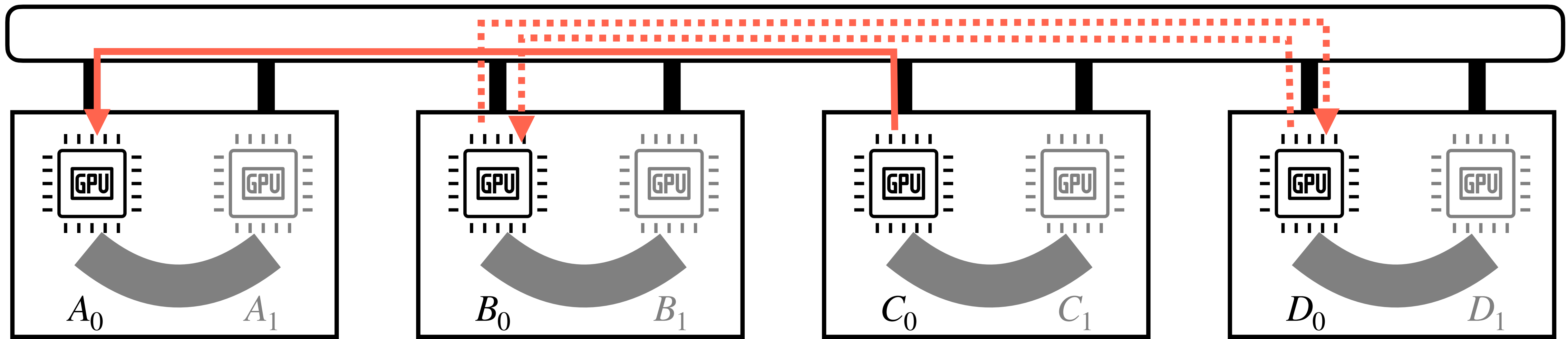
Each stage: one-to-one (incast-free), equal sized



# Example of incast-free transfer stages

		Receive														
		$A_0$	$B_0$	$C_0$	$D_0$											
Send	$A_0$	0	0	3	2	=										
	$B_0$	5	0	0	4											
	$C_0$	4	2	0	3											
	$D_0$	0	7	2	0											
						①	②				③					
						0	0	3	0	0	0	2	0	0	0	0
						3	0	0	0	2	0	0	0	0	4	
						0	0	0	3	0	2	0	0	4	0	0
						0	3	0	0	0	0	2	0	0	4	0

Each stage: one-to-one (incast-free), equal sized



# Example of incast-free transfer stages

		<i>Receive</i>							
		$A_0$	$B_0$	$C_0$	$D_0$				
<i>Send</i>	$A_0$	0	0	3	2	=			
	$B_0$	5	0	0	4				
	$C_0$	4	2	0	3				
	$D_0$	0	7	2	0				

①				②				③			
0	0	3	0	0	0	0	2	0	0	0	0
3	0	0	0	2	0	0	0	0	0	0	4
0	0	0	3	0	2	0	0	4	0	0	0
0	3	0	0	0	0	2	0	0	4	0	0

9s lower bound  
(max row/col sum)

**match**  
←

Stage times =  $3s + 2s + 4s = 9s$

The scale-out bound is still set by the largest senders/receivers

	$A_0$	$B_0$	$C_0$	$D_0$
$A_0$	0	0	3	2
$B_0$	5	0	0	4
$C_0$	4	2	0	3
$D_0$	0	7	2	0

bound = **max row/col sum**

The scale-out bound is still set by the largest senders/receivers

Global Scheduling on Scale-out

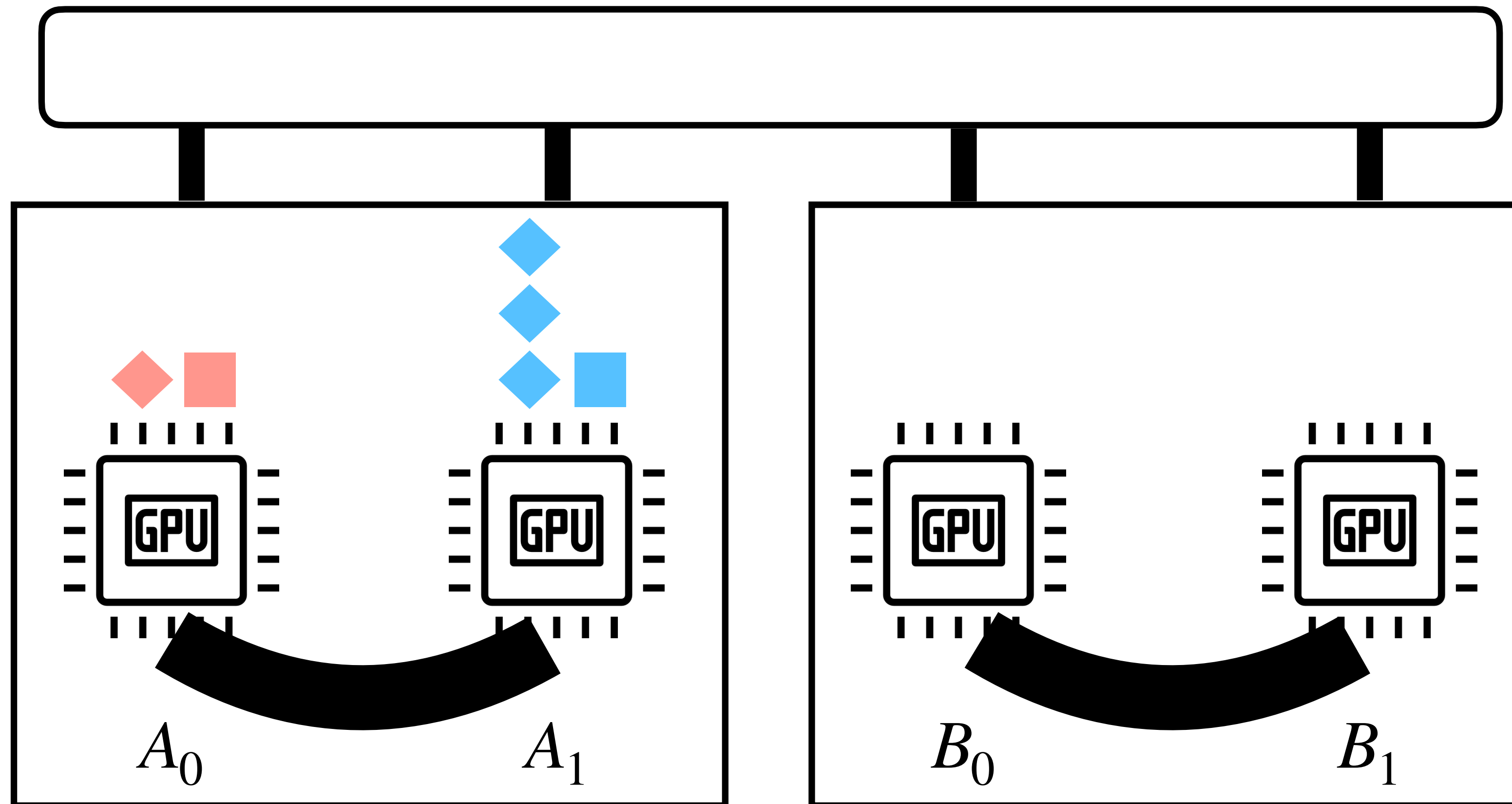


(Reduce largest row/column sum)

**Local Scheduling on Scale-up**

# With free scale-up, waypointing becomes simple

Target#1: reduce the heaviest scale-out **sending** load

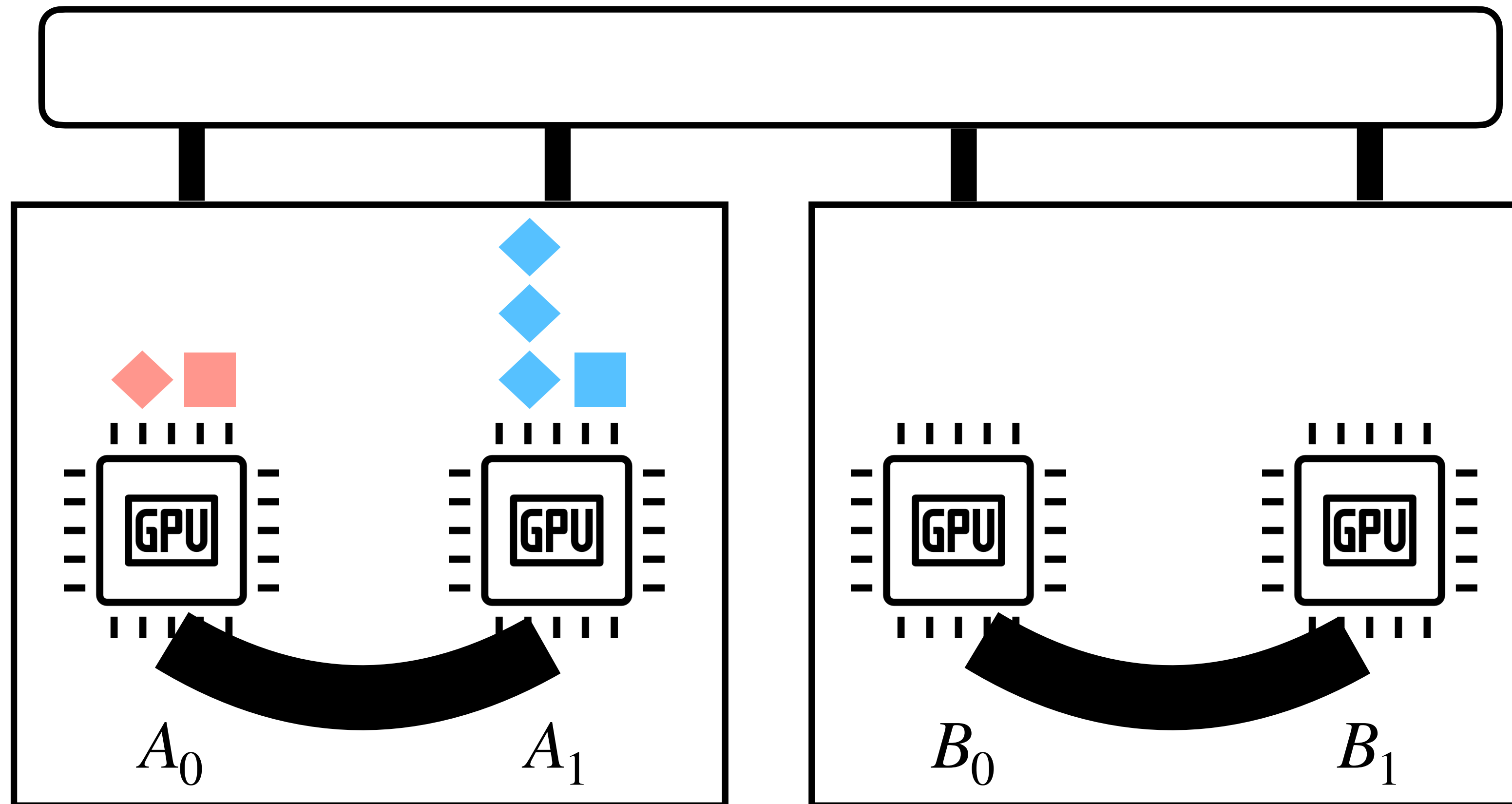


Send

	<i>Receive</i> $B_0$	$B_1$
$A_0$	1	1
$A_1$	3	1

# With free scale-up, waypointing becomes simple

Target#1: reduce the heaviest scale-out **sending** load

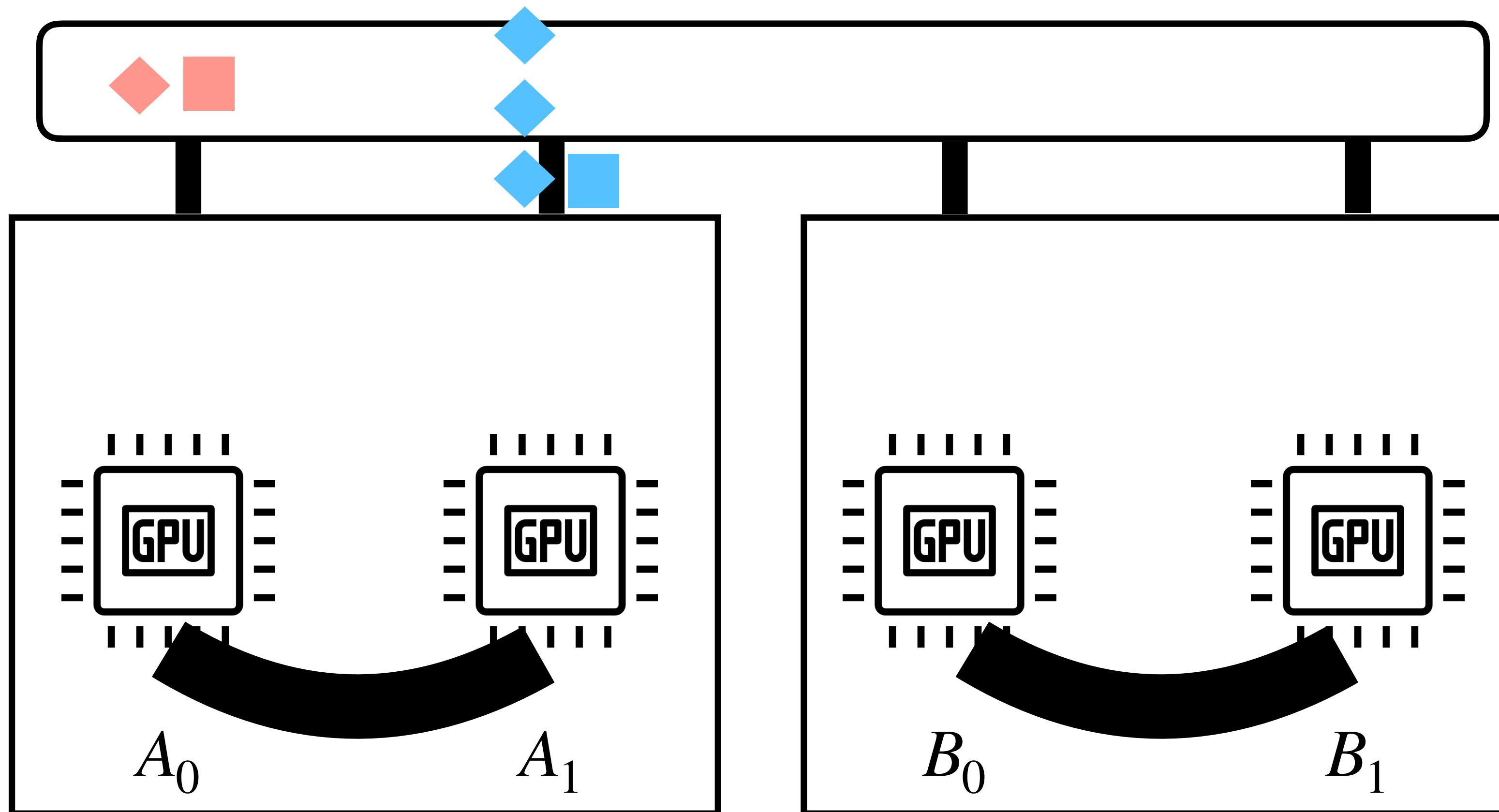


Send

	<i>Receive</i> $B_0$	$B_1$
$A_0$	1	1
$A_1$	3	1

# With free scale-up, waypointing becomes simple

Target#1: reduce the heaviest scale-out **sending** load

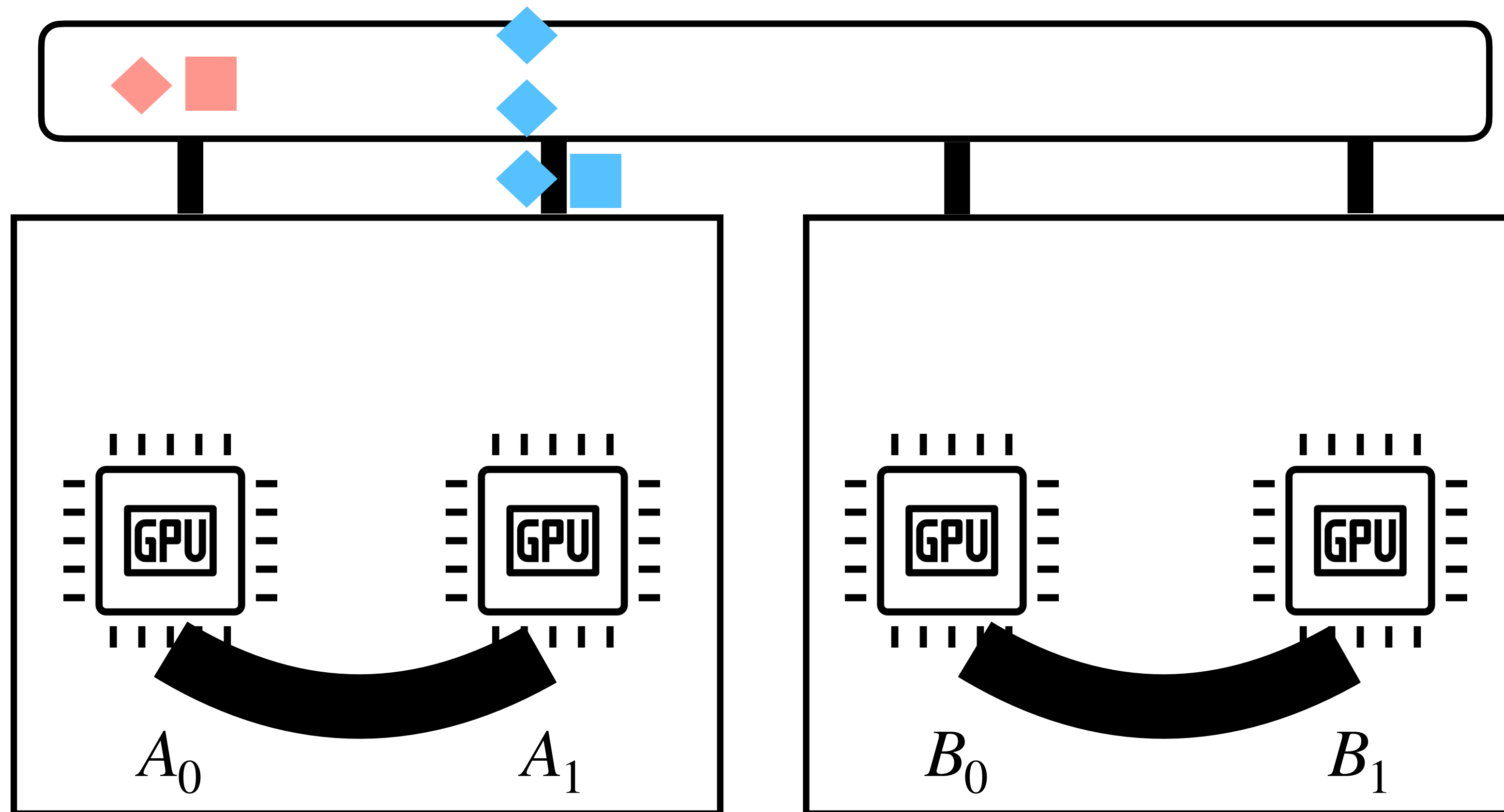


Send

	Receive	
	$B_0$	$B_1$
$A_0$	1	1
$A_1$	3	1

# With free scale-up, waypointing becomes simple

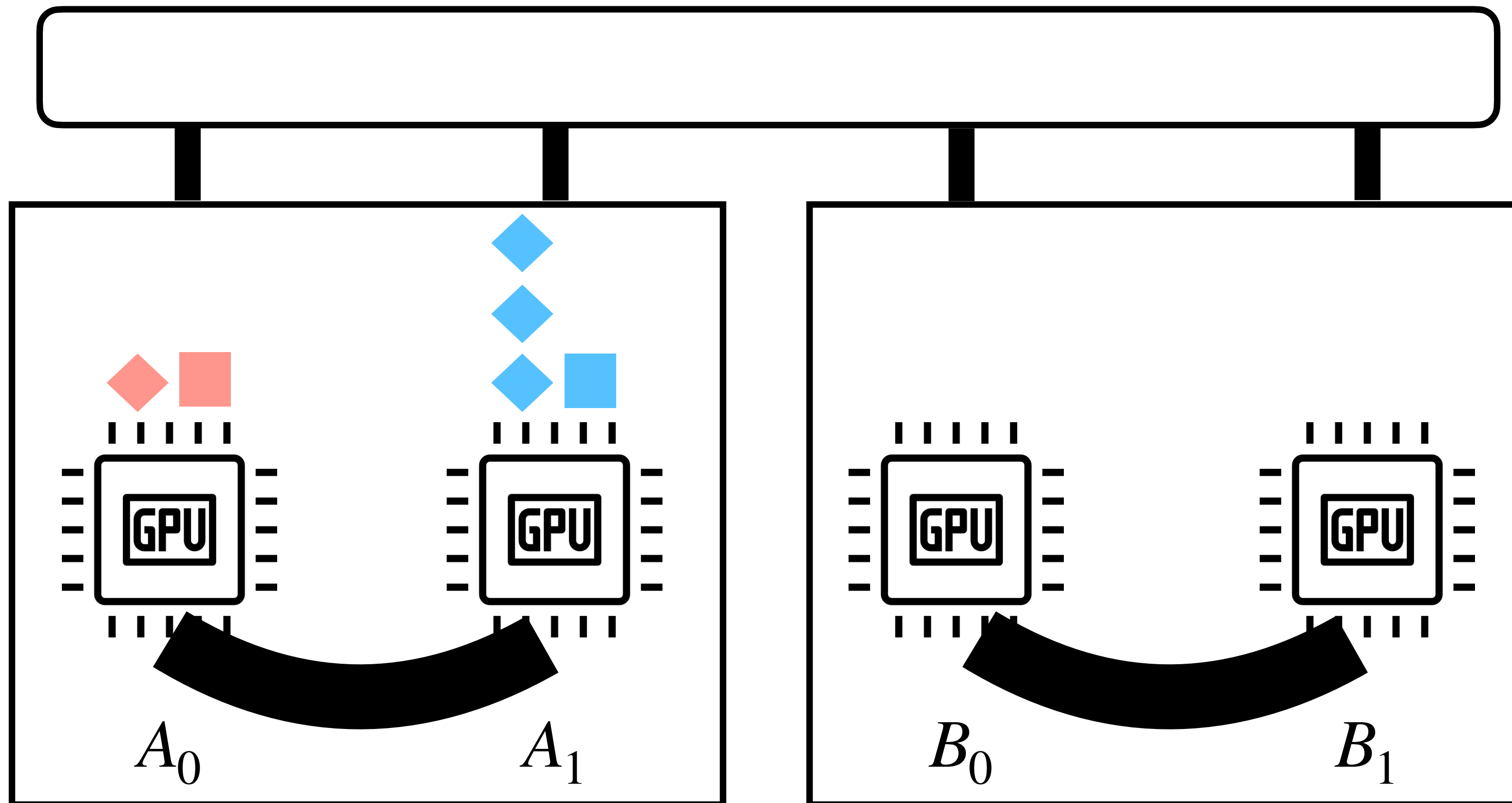
Target#1: reduce the heaviest scale-out **sending** load



		Receive		
		$B_0$	$B_1$	
Send	$A_0$	1	1	$\Sigma = 2$
	$A_1$	3	1	$\Sigma = 4$

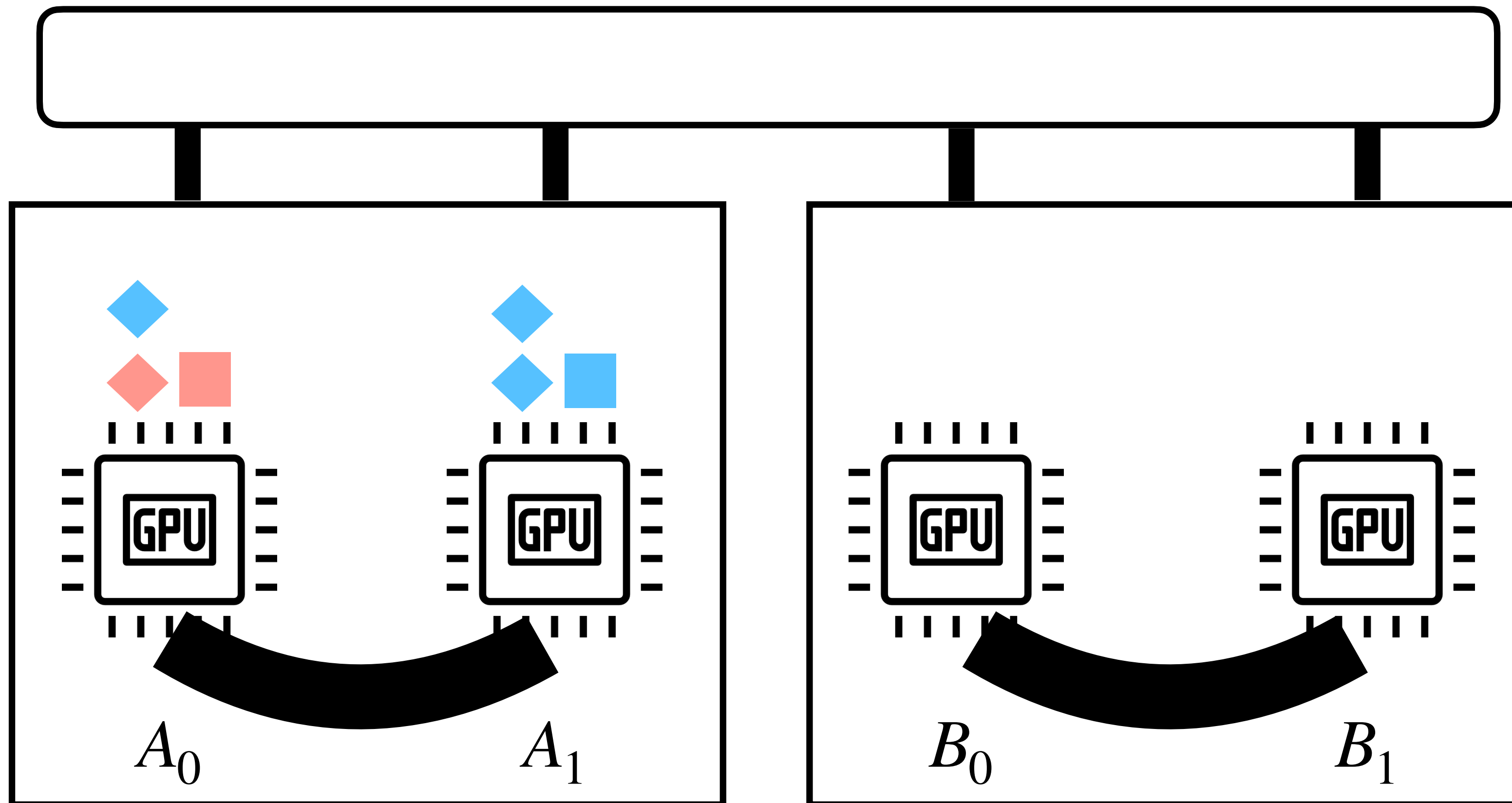
Sender straggler  $A_1$

Solution#1: rebalance sender load locally



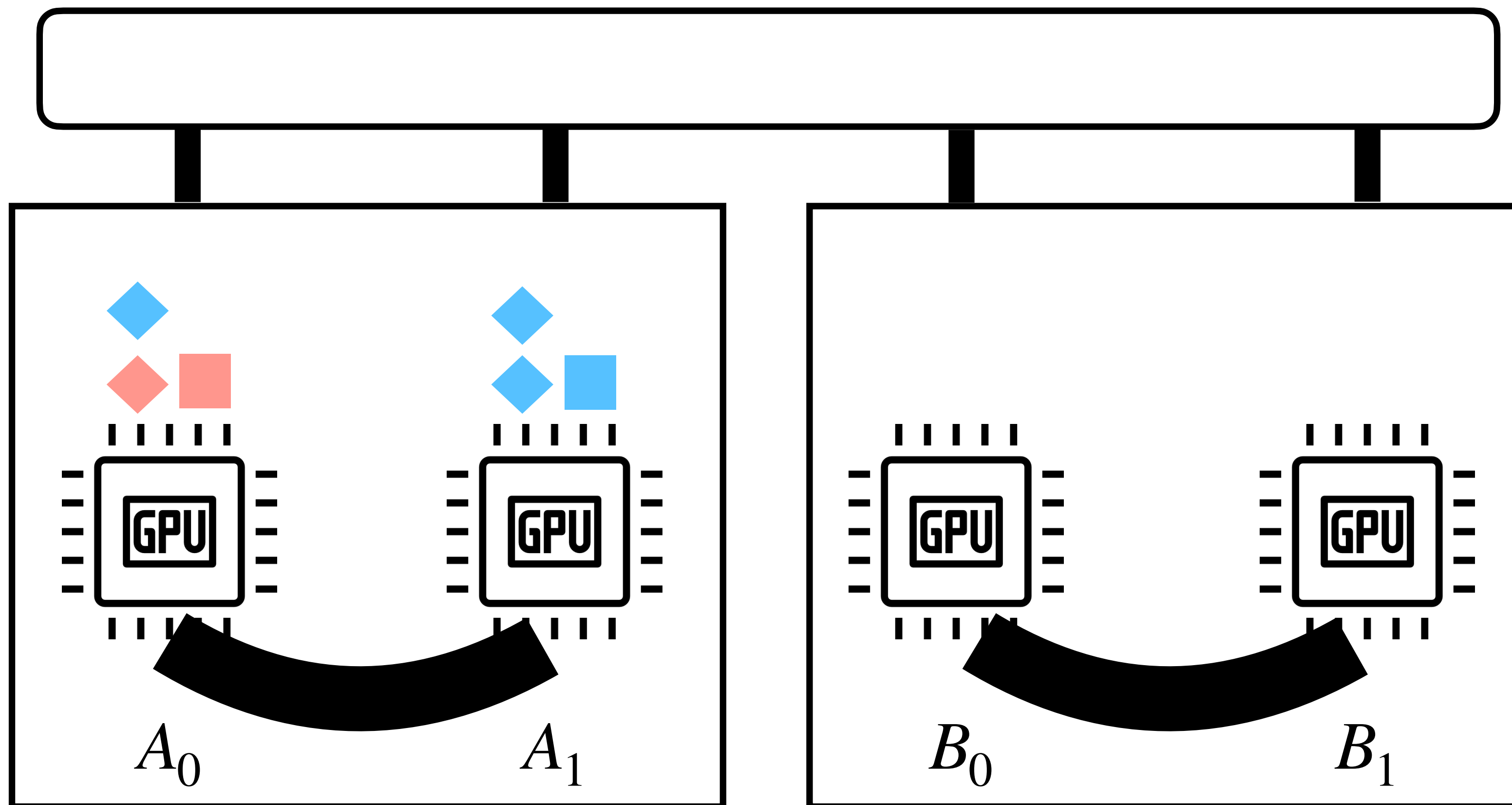
	$B_0$	$B_1$
$A_0$	1	1
$A_1$	3	1

Solution#1: rebalance sender load locally



	$B_0$	$B_1$
$A_0$	1	1
$A_1$	3	1

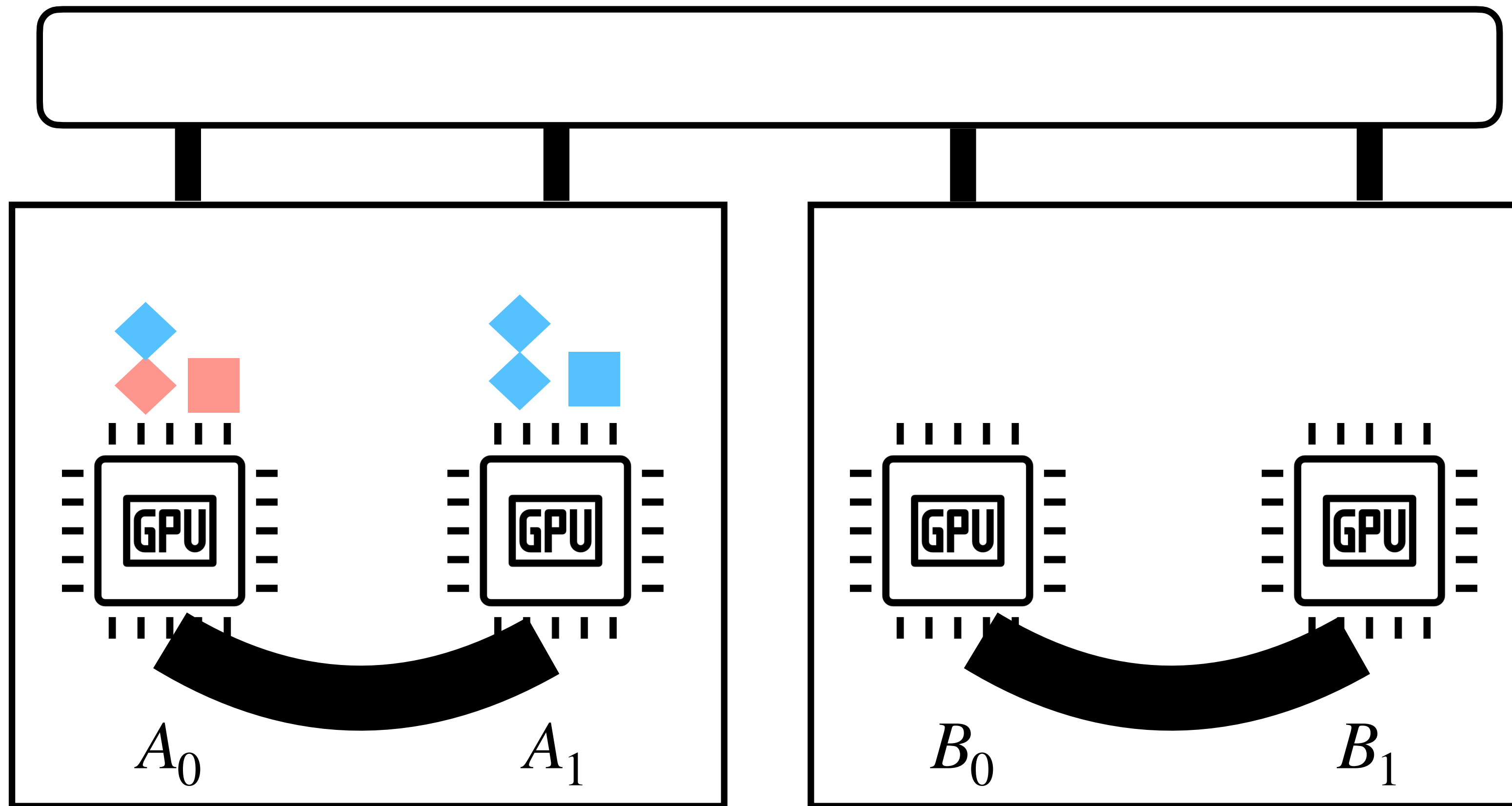
Solution#1: rebalance sender load locally



	$B_0$	$B_1$	
$A_0$	2	1	$\Sigma = 3$
$A_1$	2	1	$\Sigma = 3$

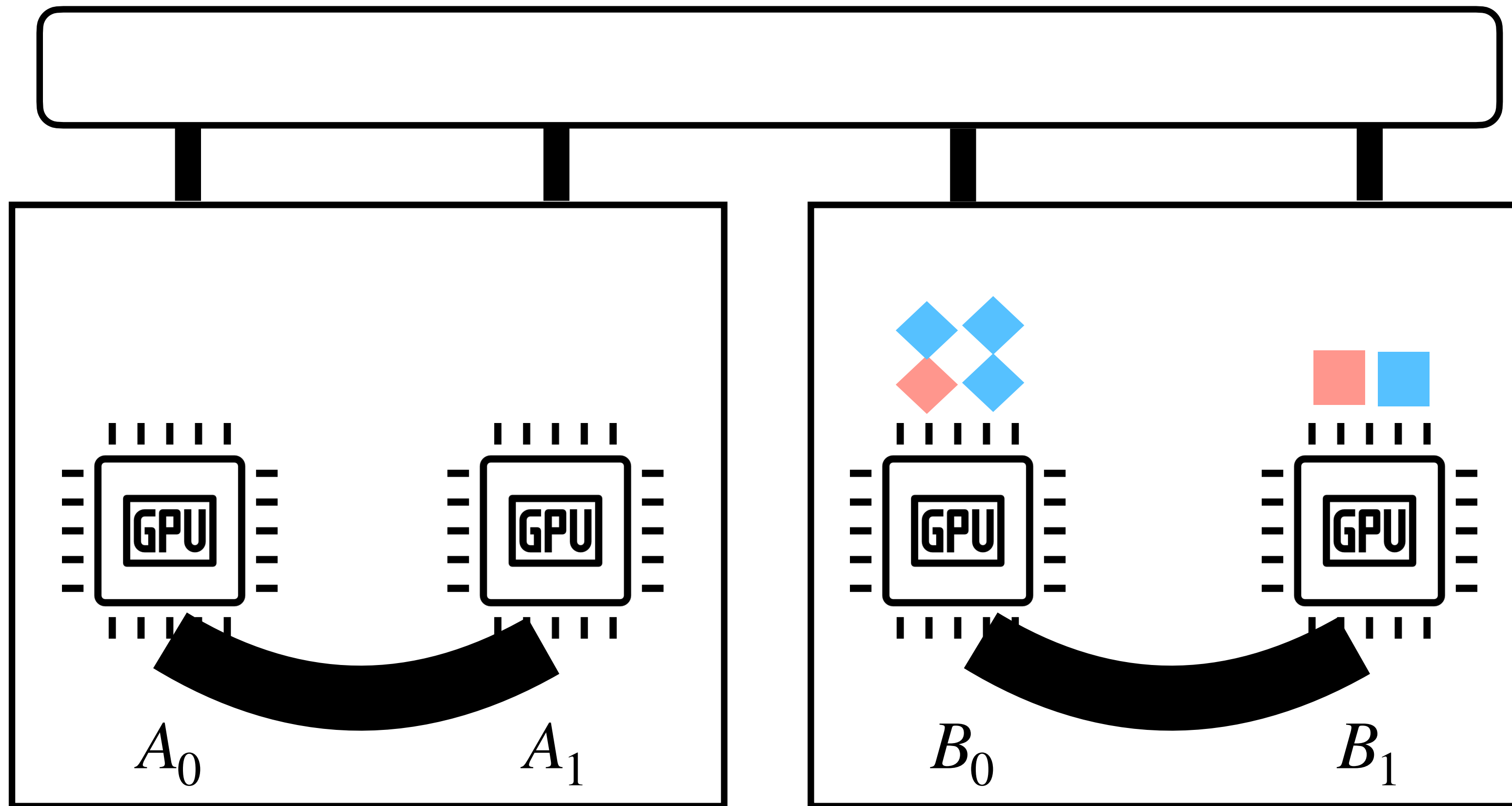
No sender stragglers!

Target#2: reduce the heaviest scale-out **receiving** load



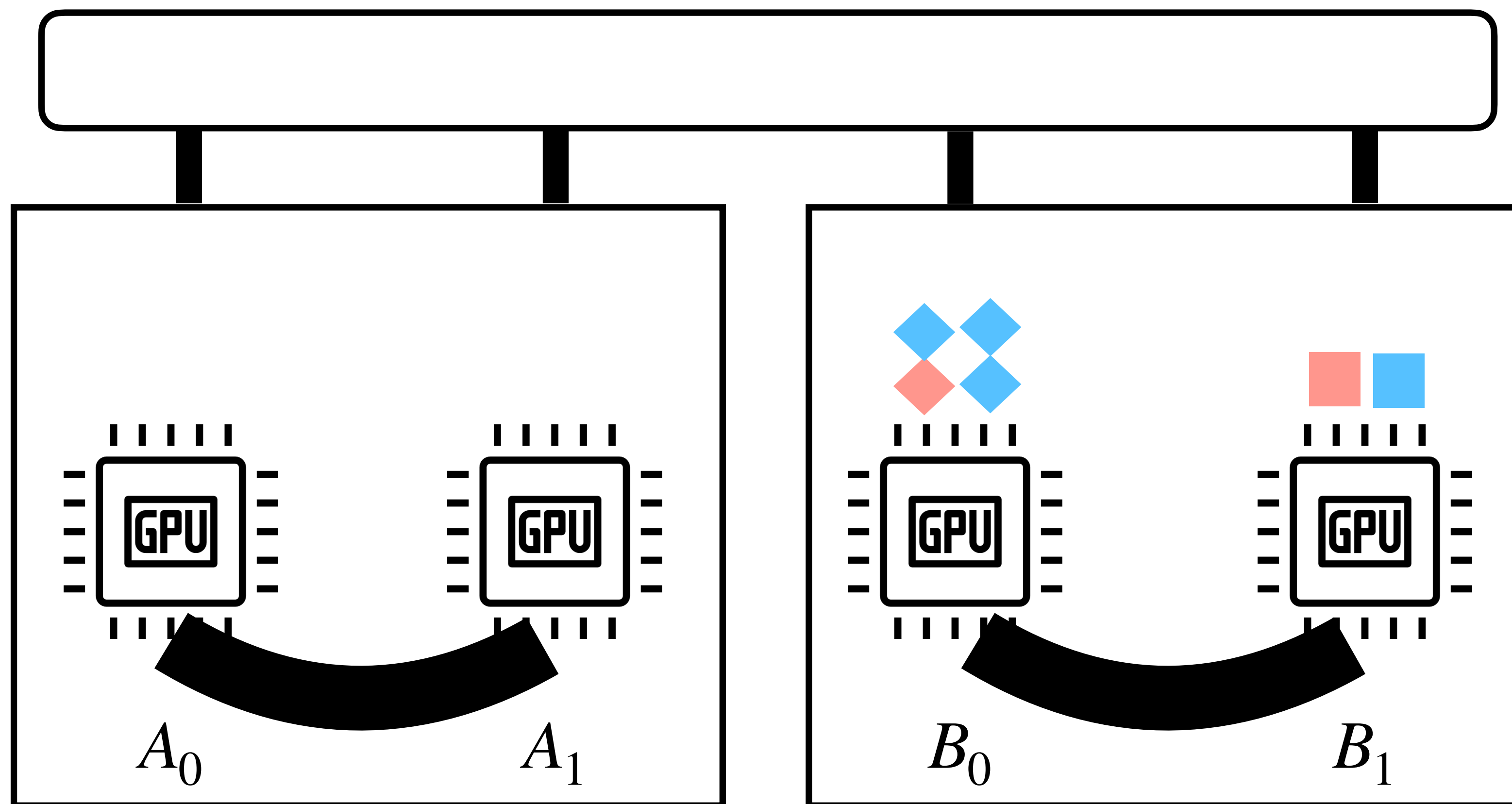
	$B_0$	$B_1$
$A_0$	2	1
$A_1$	2	1

Target#2: reduce the heaviest scale-out **receiving** load



	$B_0$	$B_1$
$A_0$	2	1
$A_1$	2	1

Target#2: reduce the heaviest scale-out **receiving** load

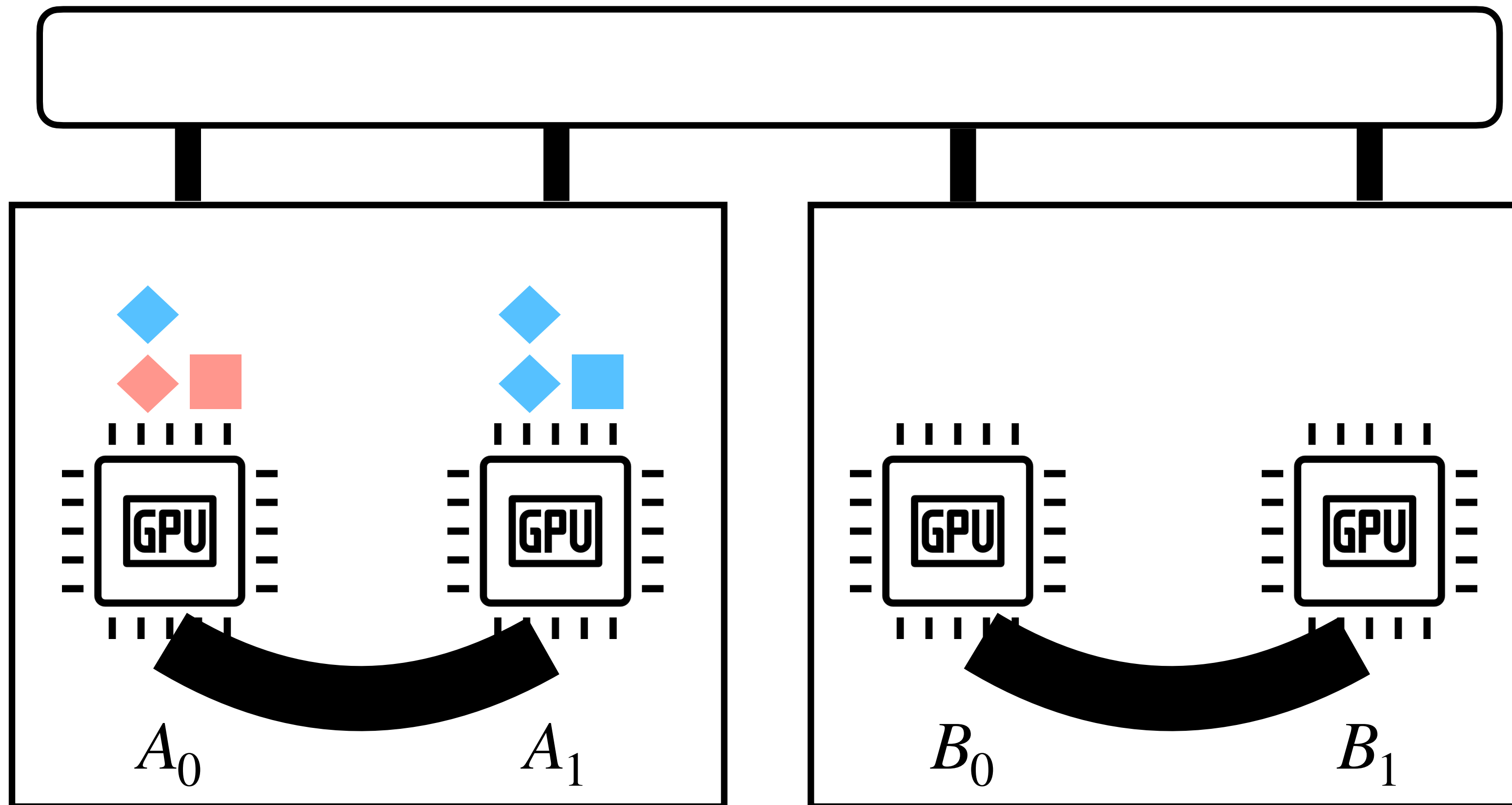


	$B_0$	$B_1$
$A_0$	2	1
$A_1$	2	1

$\Sigma = 4$   $\Sigma = 2$

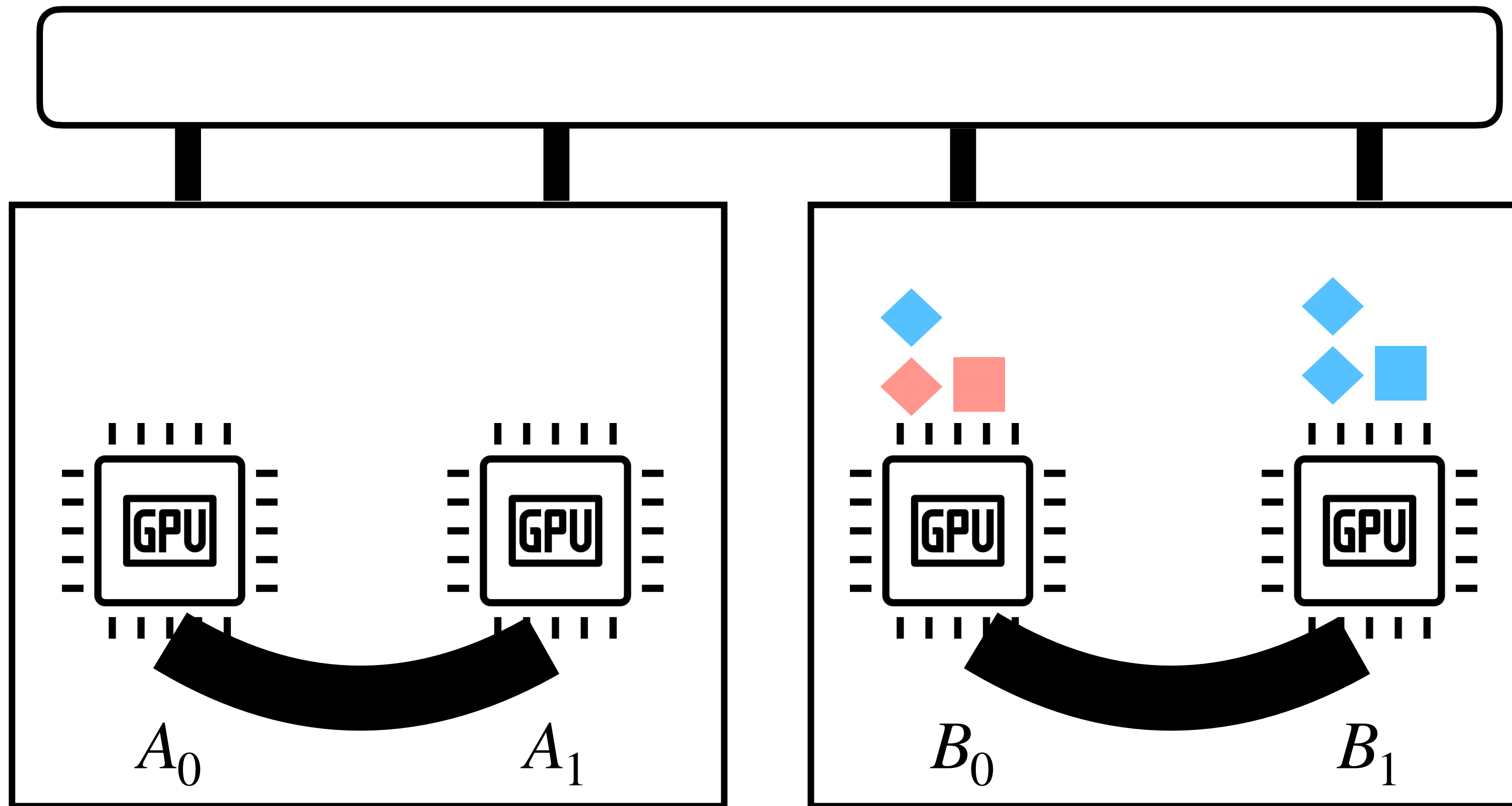
Receiver straggler  $B_0$

Solution#2: pack by destination server



	$B_0$	$B_1$
$A_0$	2	1
$A_1$	2	1

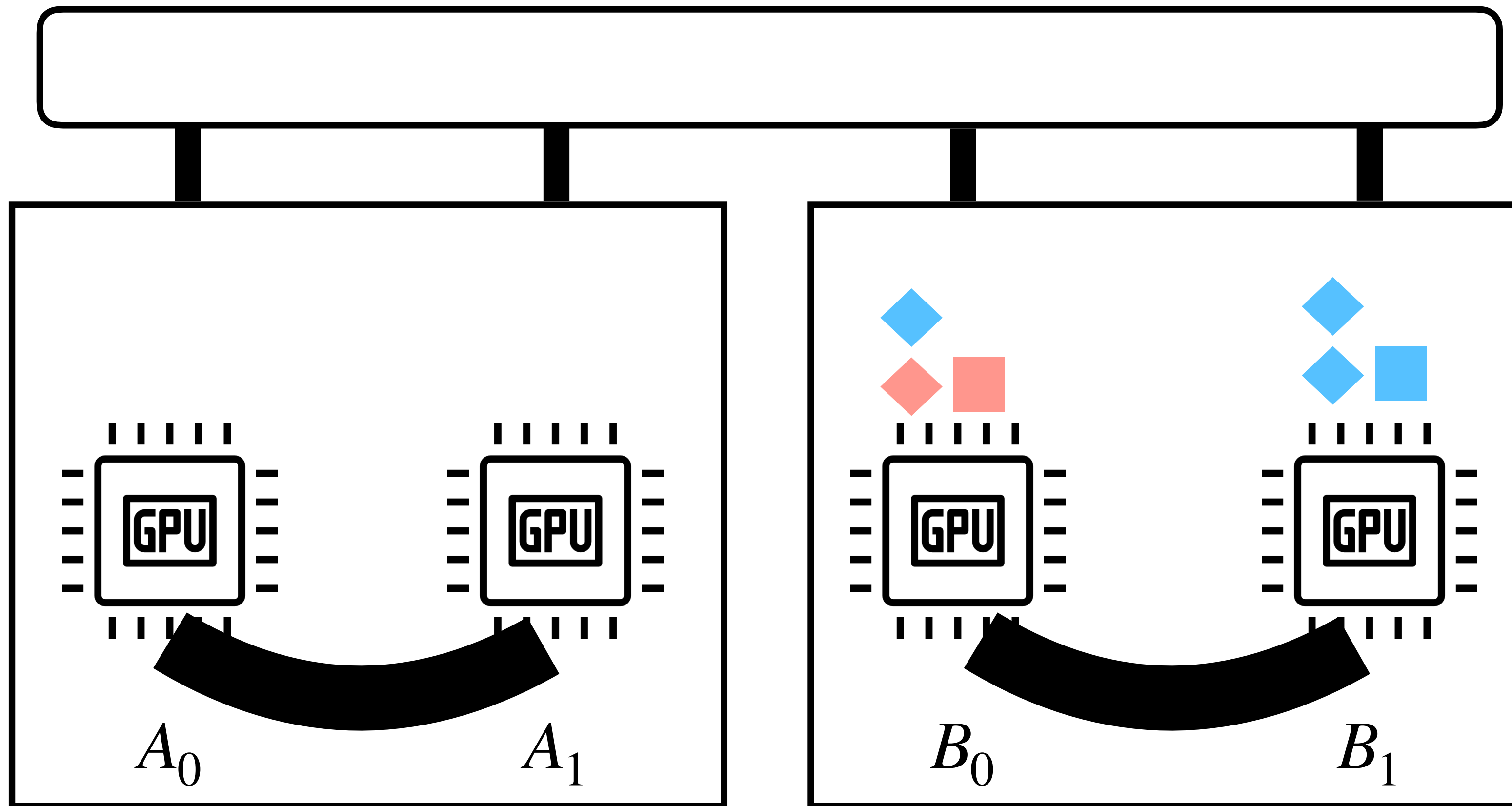
Solution#2: pack by destination server



Balanced scale-out!

	$B_0$	$B_1$
$A_0$	2	1
$A_1$	2	1

Solution#2: pack by destination server

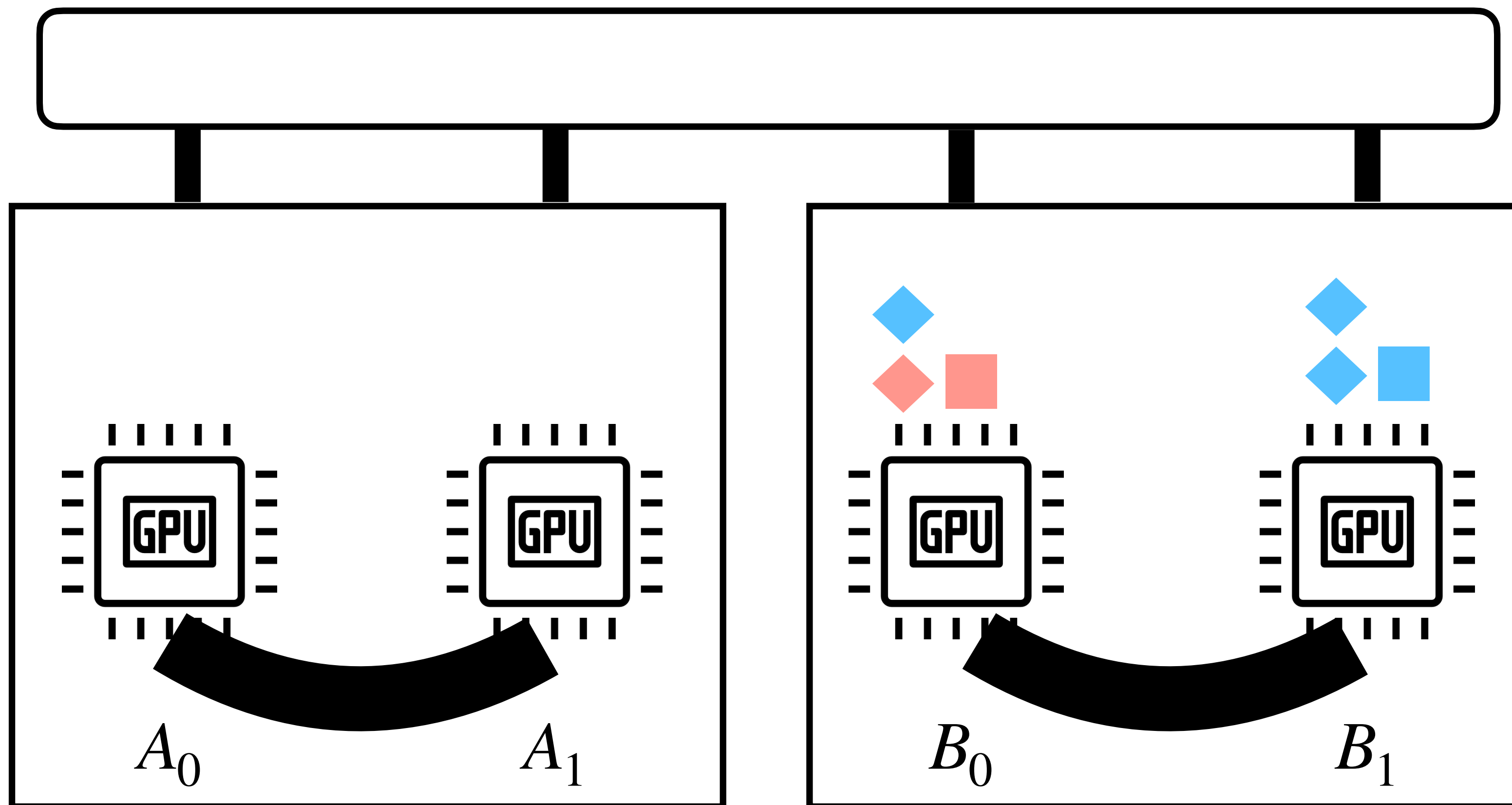


Balanced scale-out!

	$B_0$	$B_1$
$A_0$	3	0
$A_1$	0	3

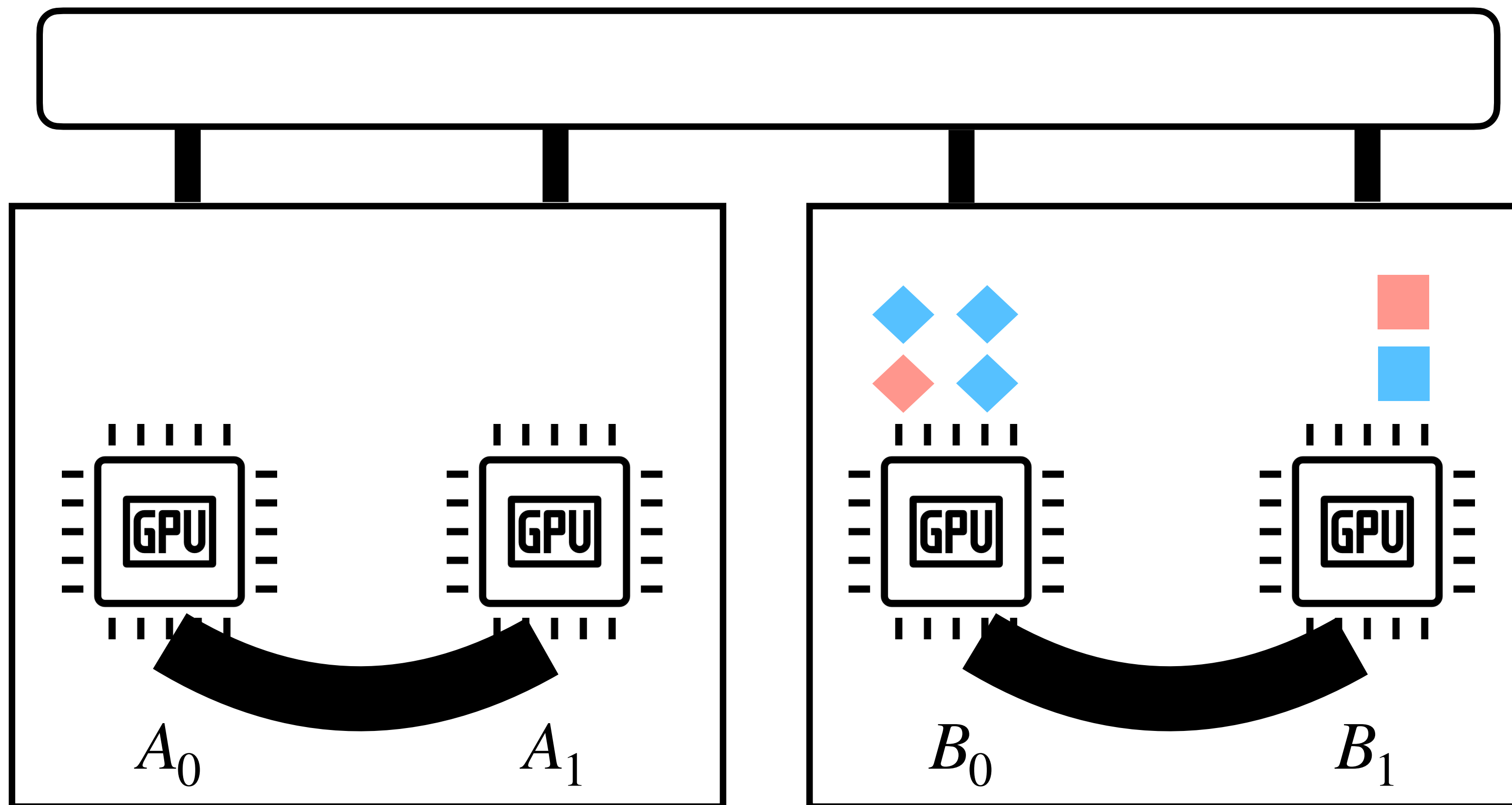
No stragglers!

Fix final placement locally



	$B_0$	$B_1$
$A_0$	3	0
$A_1$	0	3

Fix final placement locally



	$B_0$	$B_1$
$A_0$	3	0
$A_1$	0	3

# Faster scale-out by balancing every server pair

	$A_0$	$A_1$	$B_0$	$B_1$	$C_0$	$C_1$
$A_0$			6	1	1	0
$A_1$			1	4	1	2
$B_0$	0	1			2	1
$B_1$	0	3			5	2
$C_0$	4	2	2	0		
$C_1$	3	3	1	1		

# Faster scale-out by balancing every server pair

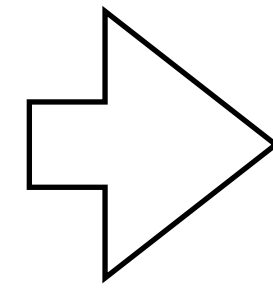
	$A_0$	$A_1$	$B_0$	$B_1$	$C_0$	$C_1$
$A_0$			6	1	1	0
$A_1$			1	4	1	2
$B_0$	0	1			2	1
$B_1$	0	3			5	2
$C_0$	4	2	2	0		
$C_1$	3	3	1	1		

$$\Sigma = 10$$

# Faster scale-out by balancing every server pair

	$A_0$	$A_1$	$B_0$	$B_1$	$C_0$	$C_1$
$A_0$			6	1	1	0
$A_1$			1	4	1	2
$B_0$	0	1			2	1
$B_1$	0	3			5	2
$C_0$	4	2	2	0		
$C_1$	3	3	1	1		

$$\Sigma = 10$$

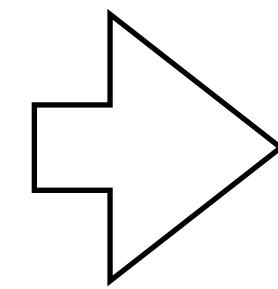


		6	0	2	0
		0	6	0	2
2	0			5	0
0	2			0	5
6	0	2	0		
0	6	0	2		

# Faster scale-out by balancing every server pair

	$A_0$	$A_1$	$B_0$	$B_1$	$C_0$	$C_1$
$A_0$			6	1	1	0
$A_1$			1	4	1	2
$B_0$	0	1			2	1
$B_1$	0	3			5	2
$C_0$	4	2	2	0		
$C_1$	3	3	1	1		

$\Sigma = 10$



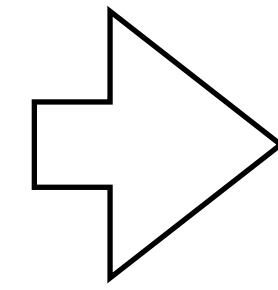
		6	0	2	0
		0	6	0	2
2	0			5	0
0	2			0	5
6	0	2	0		
0	6	0	2		

$\Sigma = 8$

# Faster scale-out by balancing every server pair

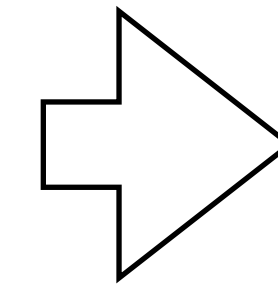
	$A_0$	$A_1$	$B_0$	$B_1$	$C_0$	$C_1$
$A_0$			6	1	1	0
$A_1$			1	4	1	2
$B_0$	0	1			2	1
$B_1$	0	3			5	2
$C_0$	4	2	2	0		
$C_1$	3	3	1	1		

$\Sigma = 10$



		6	0	2	0
		0	6	0	2
2	0			5	0
0	2			0	5
6	0	2	0		
0	6	0	2		

$\Sigma = 8$



Transfer stages

But scale-up is not free, so FAST pipelines it

# But scale-up is not free, so FAST pipelines it

- Scale-up is merely 9x faster

# But scale-up is not free, so FAST pipelines it

- Scale-up is merely 9x faster
- Pipeline scale-up with scale-out to hide most extra scale-up transfers

# But scale-up is not free, so FAST pipelines it

- Scale-up is merely 9x faster
- Pipeline scale-up with scale-out to hide most extra scale-up transfers
- Within 2.1x of an oracle bound in worst case
  - Empirically, 1.2x of that bound

Scheduling time

TACCL (NSDI '23)  
TE-CCL (SIGCOMM '24)  
SyCCL (SIGCOMM '25)

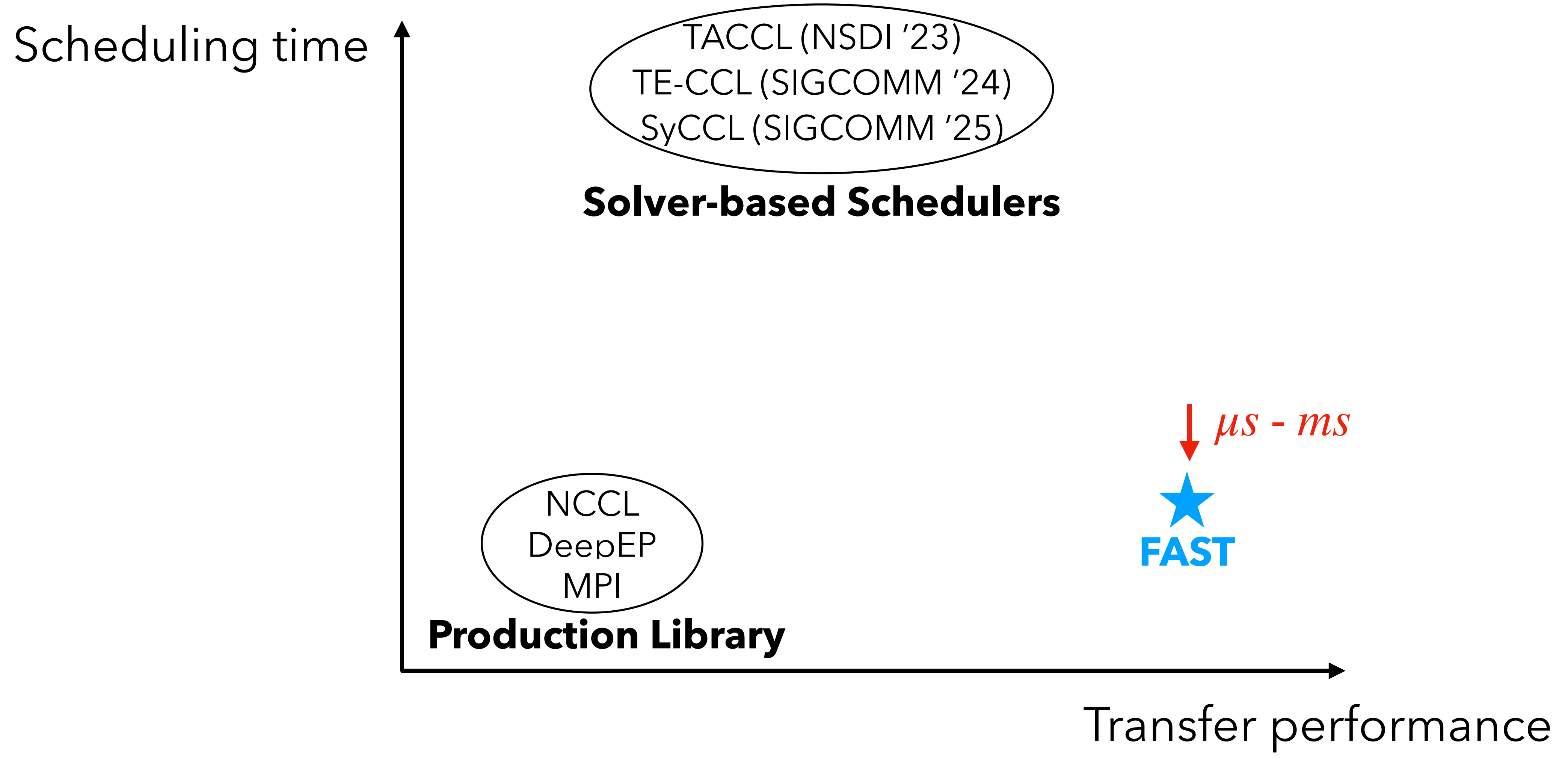
**Solver-based Schedulers**

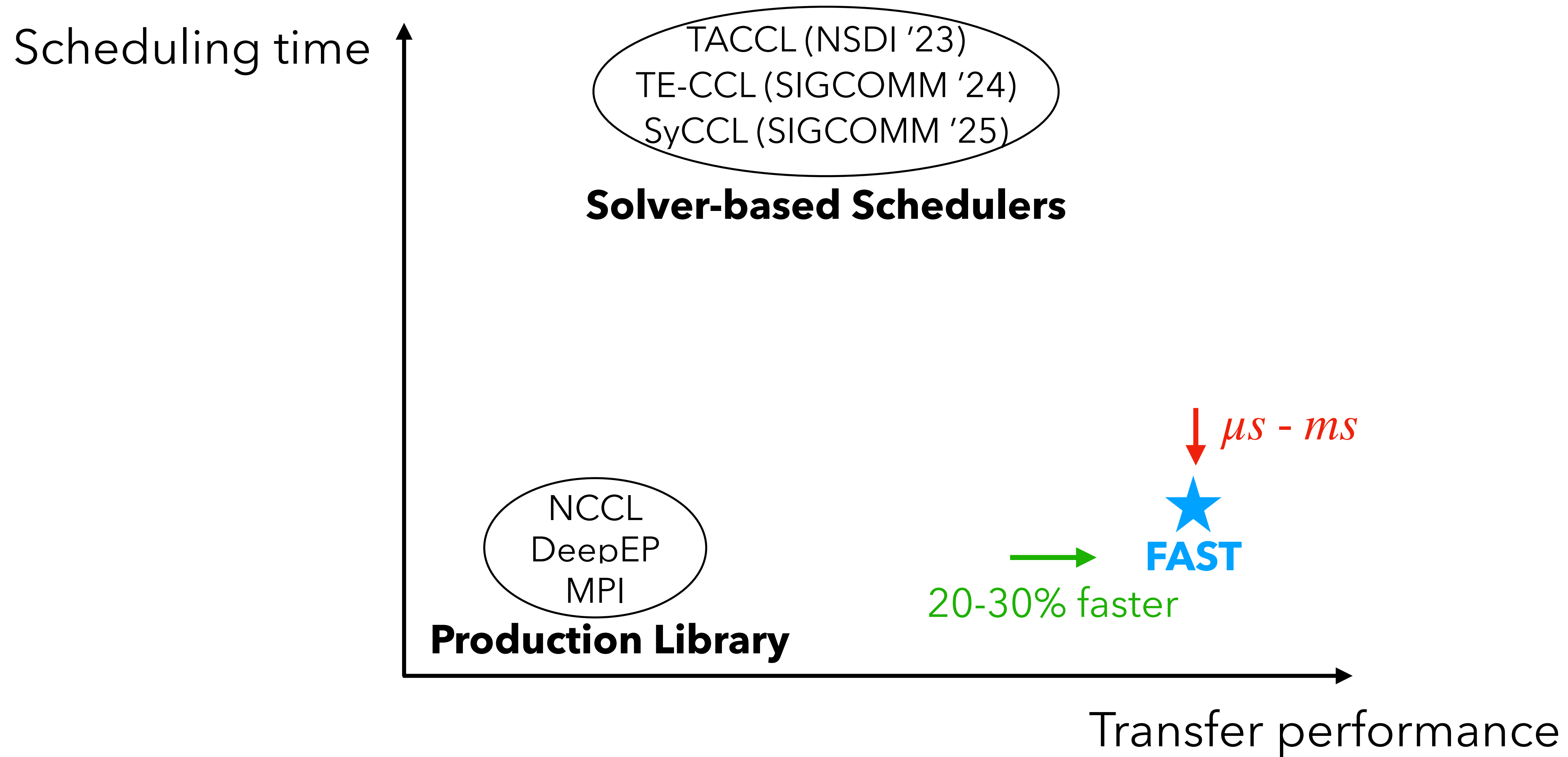
NCCL  
DeepEP  
MPI

**Production Library**



Transfer performance





# Implementation & evaluation setup

## NVIDIA

## AMD

**Software stack**

NVSHMEM

RCCL

**GPU model**

H200

MI300X

**Scale-up vs scale-out**

450 vs 50 GBps  
(x9 gap)

448 vs 12.5 GBps  
(x36 gap)

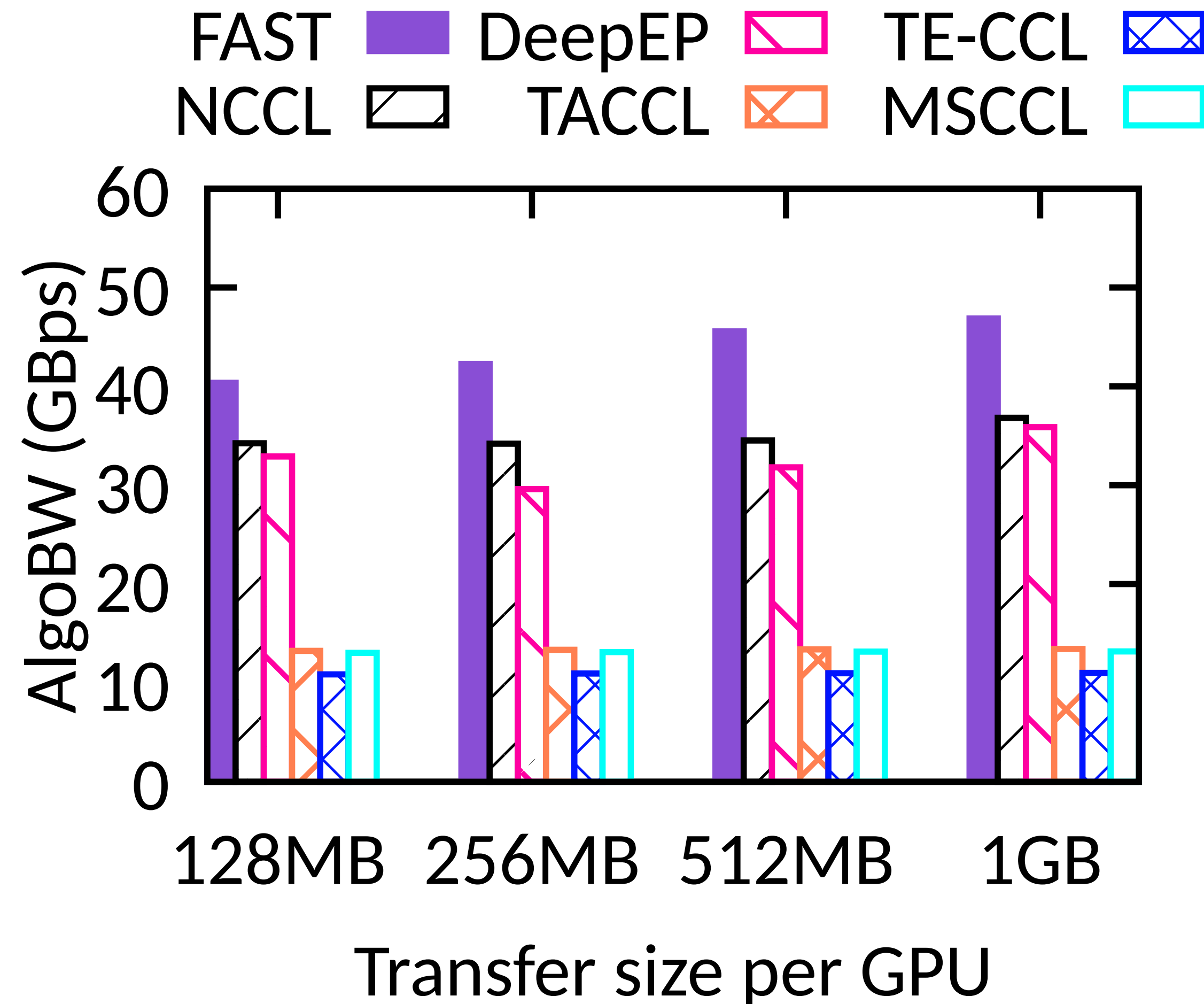
**Workload**

Skewed traffic with [Zipf](#) distribution

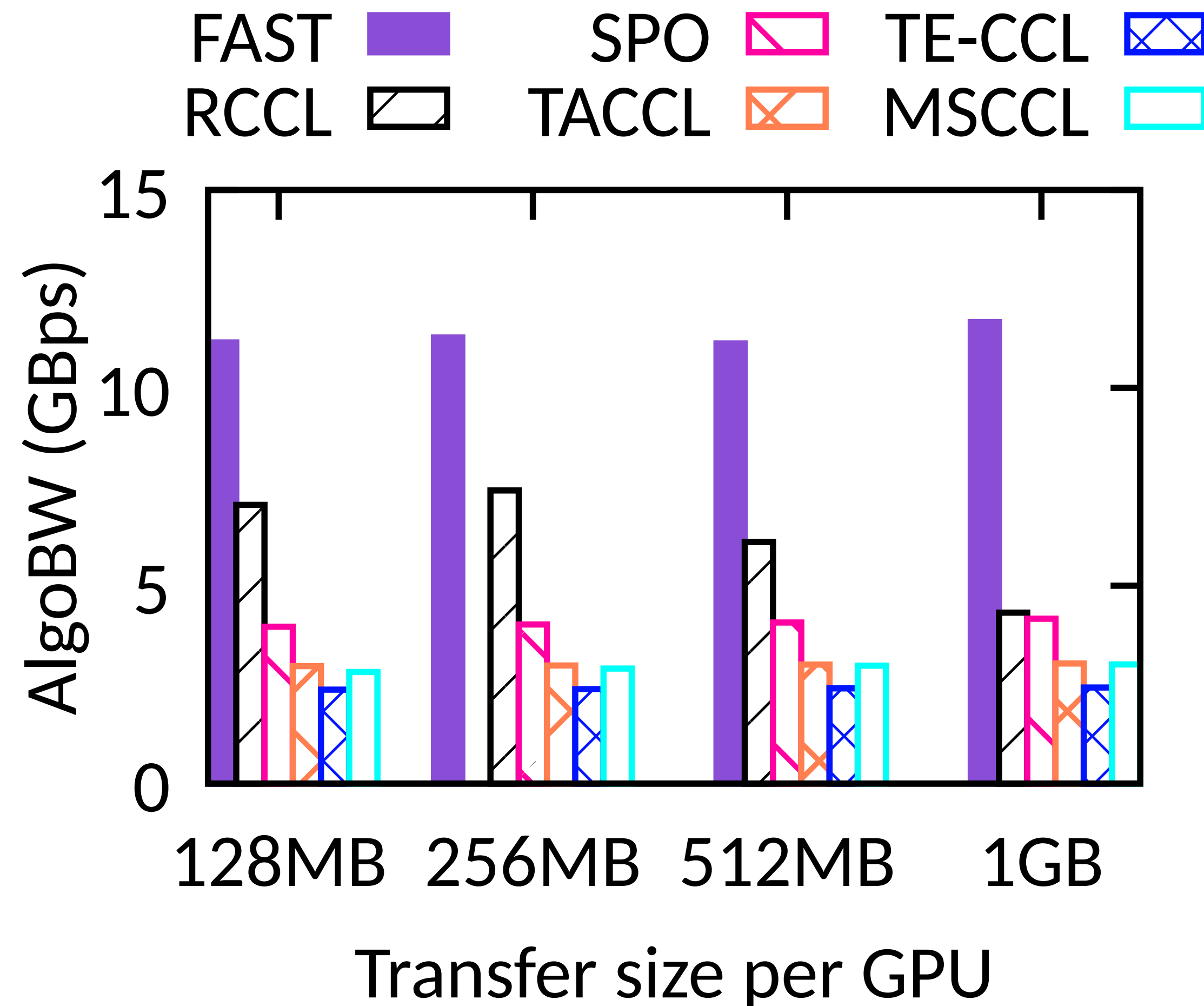
**Scale**

32 GPUs, 8 GPU-per-server

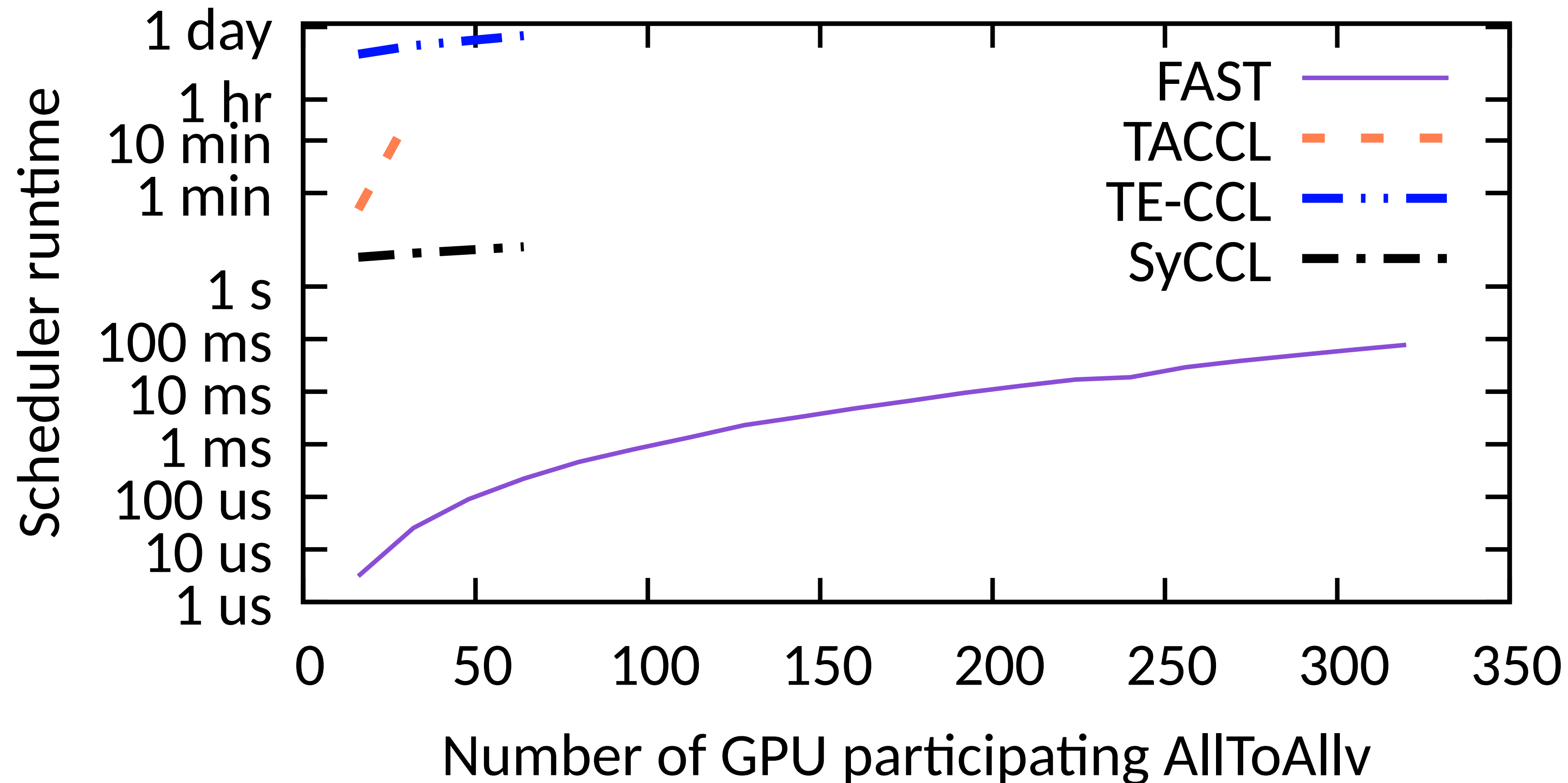
FAST achieves **1.2-1.3x** higher algorithmic bandwidth than the best **NVIDIA** baseline



FAST achieves **1.3-2.6x** higher bandwidth than the best **AMD** baseline



FAST completes in  $\mu s - ms$ ,  
orders of magnitude faster than solvers



# Takeaway

# Takeaway

- All-to-All is hard: skew, incast, and fast-changing traffic

# Takeaway

- All-to-All is hard: skew, incast, and fast-changing traffic
- FAST: reshape traffic locally to simplify global transfers

# Takeaway

- All-to-All is hard: skew, incast, and fast-changing traffic
- FAST: reshape traffic locally to simplify global transfers
- 20-30% higher bandwidth,  $\mu s$  –  $ms$  schedules

# Takeaway

- All-to-All is hard: skew, incast, and fast-changing traffic
- FAST: reshape traffic locally to simplify global transfers
- 20-30% higher bandwidth,  $\mu s$  –  $ms$  schedules

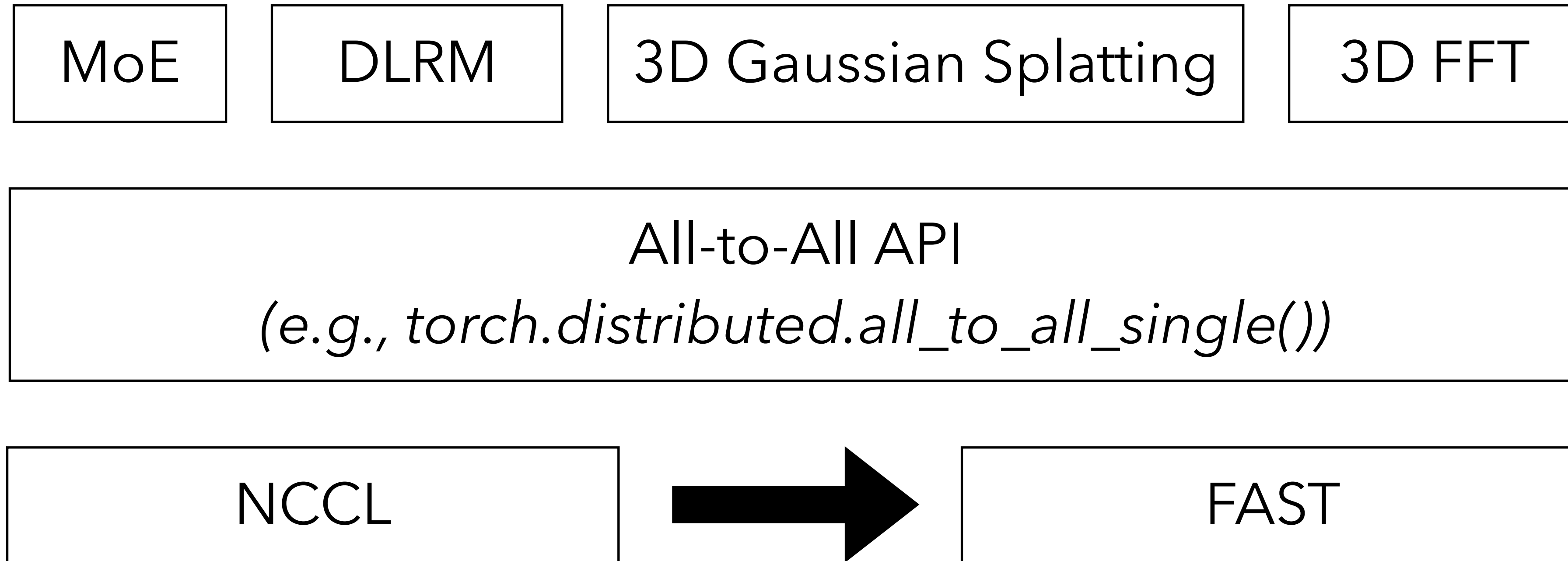
Contact: [yiranlei@cs.cmu.edu](mailto:yiranlei@cs.cmu.edu)

FAST is open sourced

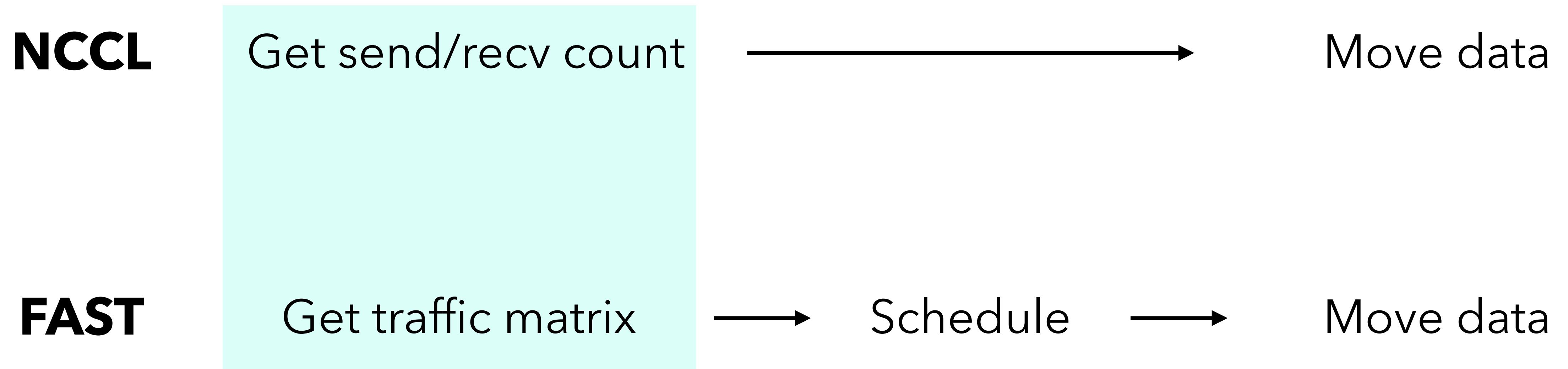


# Backup slides

# FAST drops in for NCCL All-to-All



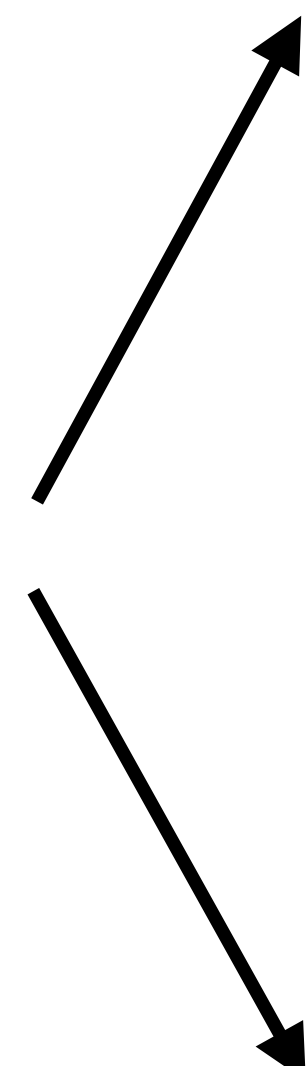
# NCCL vs. FAST: where the scheduler fits



**All-Gather** in Megatron-LM

# Birkhoff's Decomposition vs Spreadout

	$A_0$	$B_0$	$C_0$	$D_0$
$A_0$	0	0	3	2
$B_0$	5	0	0	4
$C_0$	4	2	0	3
$D_0$	0	7	2	0



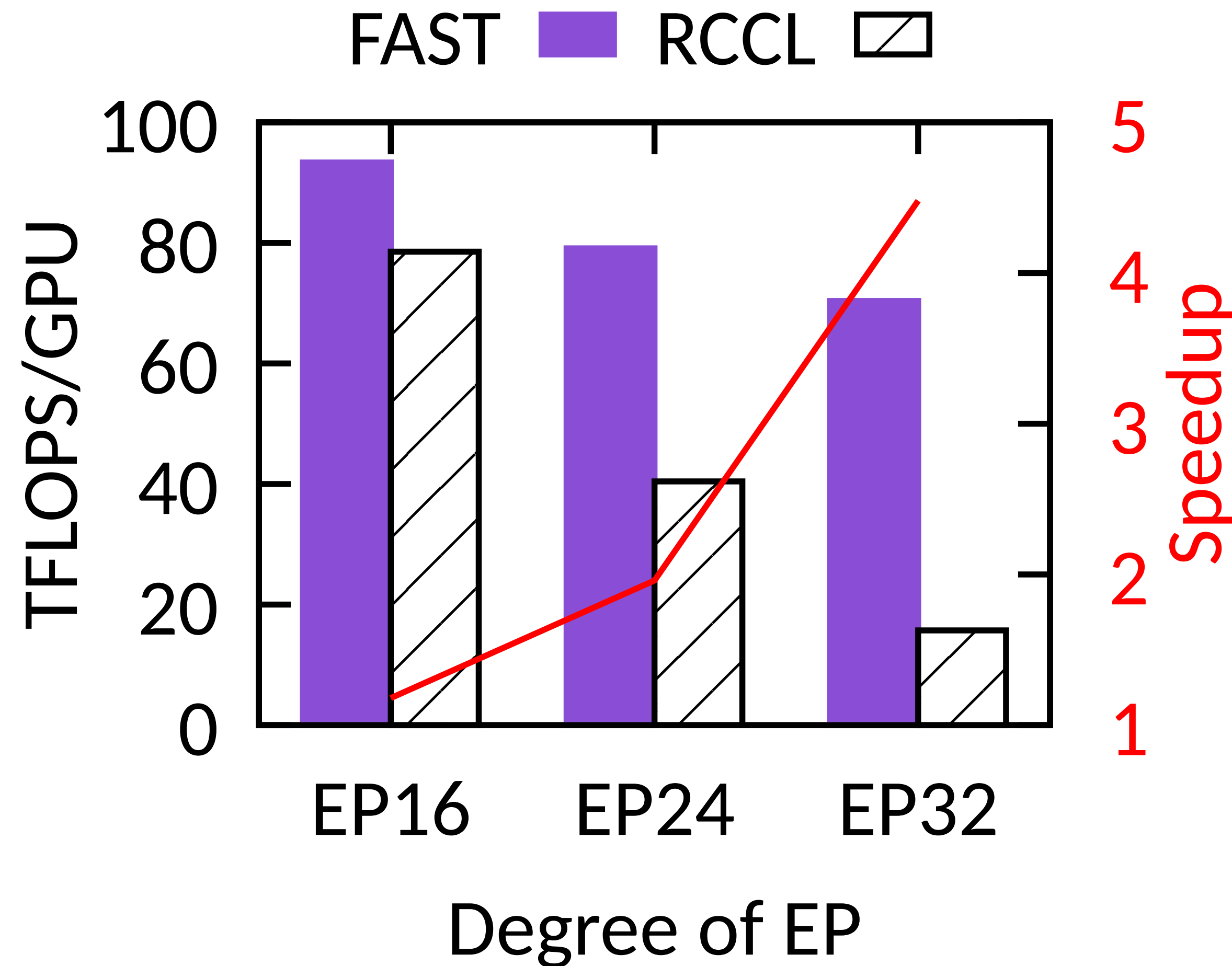
Birkhoff's decomposition

0	0	3	0	+	0	0	0	2	+	0	0	0	0
3	0	0	0		2	0	0	0		0	0	0	4
0	0	0	3		0	2	0	0		4	0	0	0
0	3	0	0		0	0	2	0		0	4	0	0
$3s$					$2s$					$4s$			

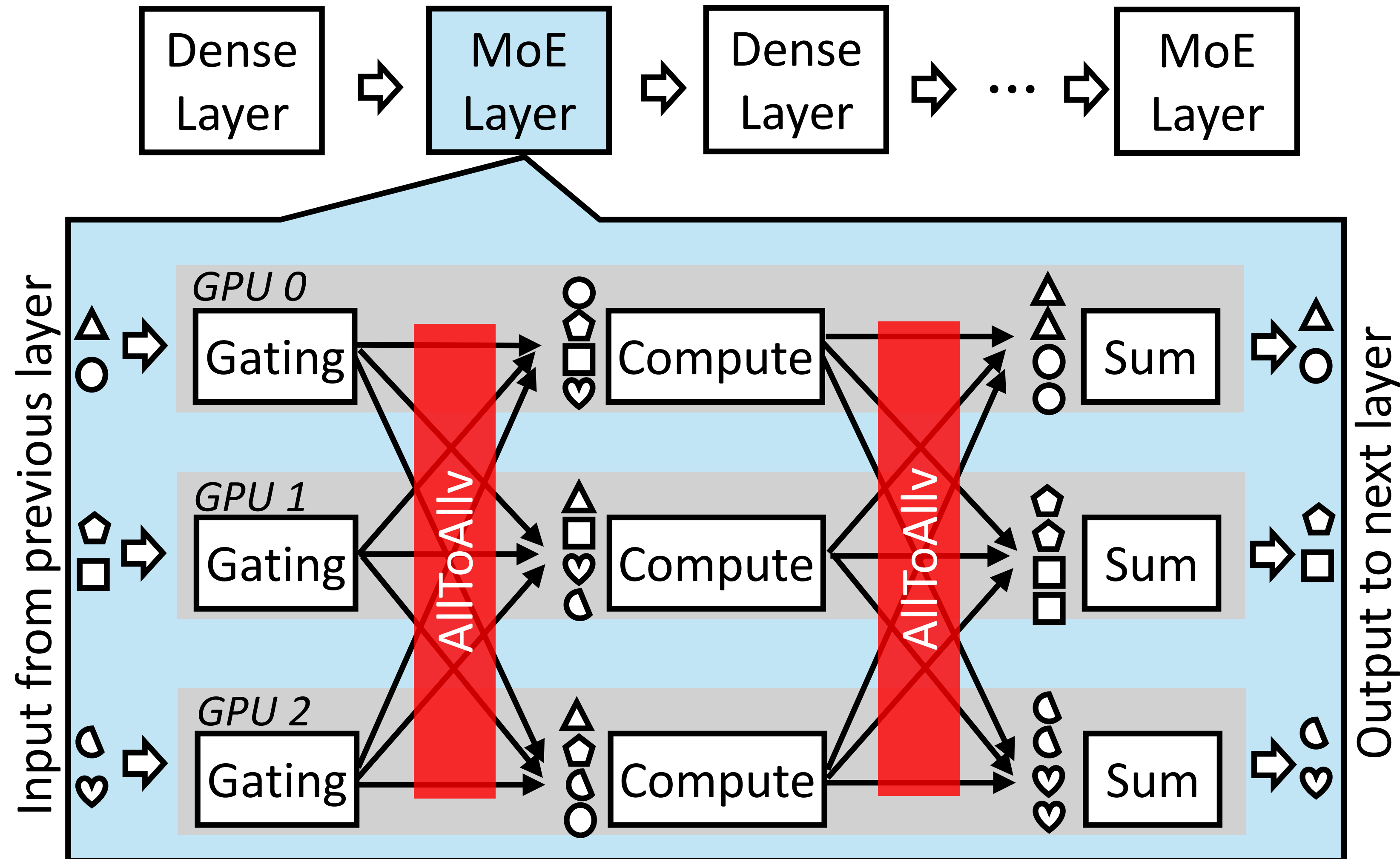
Spreadout

0	0	3	0	+	0	0	0	2	+	0	0	0	0
0	0	0	4		5	0	0	0		0	0	0	0
4	0	0	0		0	2	0	0		0	0	0	3
0	7	0	0		0	0	2	0		0	0	0	0
$7s$					$5s$					$3s$			

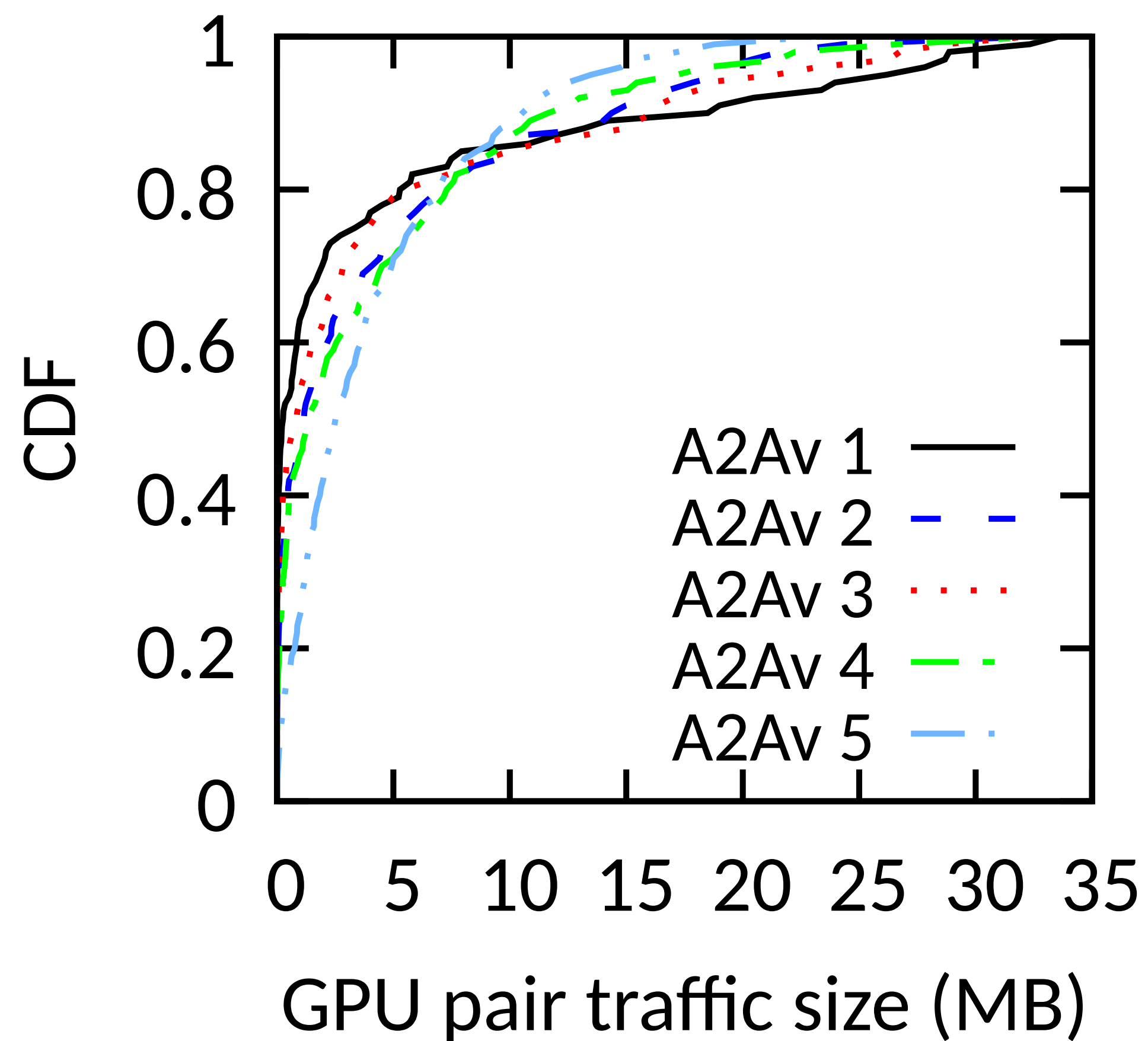
FAST achieves **1.2-4.5x** higher performance when using Megatron-LM to train a MoE model on AMD testbed



# All-to-All is heavily used in MoE models



# Skew from pretraining a MoE model using Megatron-LM



# Dynamism from pretraining a MoE model using Megatron-LM

