Parallel Computing in TensorFlow

Shusen Wang

TensorFlow Strategies

- `MirroredStrategy`
- `TPUStrategy`
- `MultiWorkerMirroredStrategy`
- `CentralStorageStrategy`
- `ParameterServerStrategy`
- `OneDeviceStrategy`

Parallel Training CNN on MNIST

Find Devices

```
from tensorflow.python.client import device_lib

device_lib.list_local_devices()
```

For example, my server has 1 CPU and 4 GPUs:

- `/device:CPU:0`
- `/device:GPU:0`
- `/device:GPU:1`
- `/device:GPU:2`
- `/device:GPU:3`

Mirrored Strategy

```python
from tensorflow import distribute

strategy = distribute.MirroredStrategy()
```

Mirrored Strategy

```python
from tensorflow import distribute

strategy = distribute.MirroredStrategy()
```

```python
# number of processors
m = strategy.num_replicas_in_sync

print('Number of devices: {}'.format(m))
```

```
Number of devices: 4
```

Load and Process MNIST Data

```python
import tensorflow as tf

def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255
    return image, label
```

Load and Process MNIST Data

```python
import tensorflow as tf

def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255
    return image, label
```

```python
import tensorflow_datasets as tfds

datasets, info = tfds.load(name='mnist',
                           with_info=True,
                           as_supervised=True)

mnist_train = datasets['train'].map(scale).cache()
mnist_test = datasets['test'].map(scale)
```

Load and Process MNIST Data

```
BUFFER_SIZE = 10000
BATCH_SIZE_PER_REPLICA = 128
BATCH_SIZE = BATCH_SIZE_PER_REPLICA * m

data_train = mnist_train.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

Load and Process MNIST Data

```
BUFFER_SIZE = 10000
BATCH_SIZE_PER_REPLICA = 128
BATCH_SIZE = BATCH_SIZE_PER_REPLICA * m

data_train = mnist_train.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
data_test = mnist_test.batch(BATCH_SIZE)
```

Build Neural Network

```
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

with strategy.scope():
    model = keras.Sequential()
    model.add(Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D())
    model.add(Conv2D(64, 3, activation='relu'))
    model.add(MaxPooling2D())
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(10, activation='softmax'))
```

Build Neural Network

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 64)	102464
dense_1 (Dense)	(None, 10)	650

Total params: 121,930
Trainable params: 121,930
Non-trainable params: 0

Compile the Model

```
with strategy.scope():
    model.compile(loss='sparse_categorical_crossentropy',
                  optimizer=keras.optimizers.RMSprop(learning_rate=1E-3),
                  metrics=['accuracy'])
```

Train the Model

```
model.fit(data_train, epochs=10)
```

```
Epoch 1/10
118/118 [==============================] - 18s 151ms/step - loss: 0.4800 - accuracy: 0.8558
Epoch 2/10
118/118 [==============================] - 1s 9ms/step - loss: 0.1229 - accuracy: 0.9632
Epoch 3/10
118/118 [==============================] - 1s 9ms/step - loss: 0.0772 - accuracy: 0.9764
Epoch 4/10
118/118 [==============================] - 1s 9ms/step - loss: 0.0583 - accuracy: 0.9821
Epoch 5/10
118/118 [==============================] - 1s 9ms/step - loss: 0.0448 - accuracy: 0.9859
Epoch 6/10
118/118 [==============================] - 1s 9ms/step - loss: 0.0364 - accuracy: 0.9888
Epoch 7/10
118/118 [==============================] - 1s 9ms/step - loss: 0.0311 - accuracy: 0.9905
Epoch 8/10
118/118 [==============================] - 1s 9ms/step - loss: 0.0258 - accuracy: 0.9919
Epoch 9/10
118/118 [==============================] - 1s 9ms/step - loss: 0.0220 - accuracy: 0.9931
Epoch 10/10
118/118 [==============================] - 1s 9ms/step - loss: 0.0187 - accuracy: 0.9940
```

Evaluate the Model on Test Set

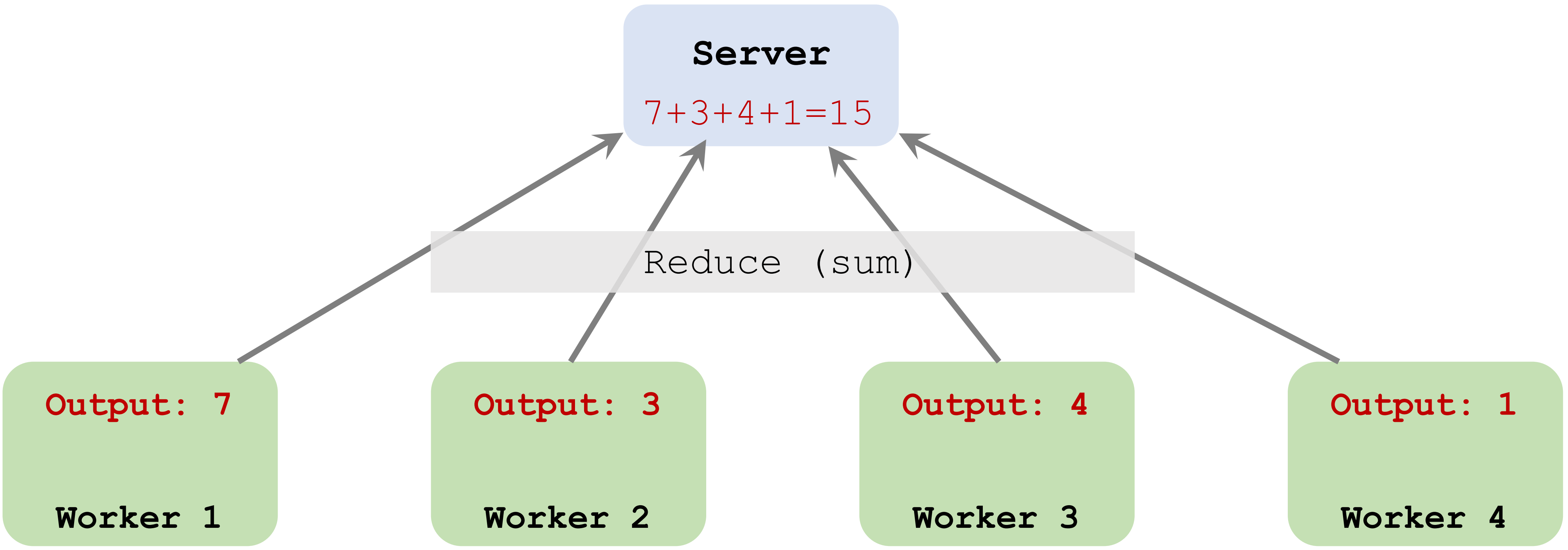
```
eval_loss, eval_acc = model.evaluate(data_test)
```

```
20/20 [==============================] - 2s 92ms/step - loss: 0.0255 - accuracy: 0.9896
```

Ring All-Reduce

Reduce VS All-Reduce

- Reduce: The server gets the result of reduce (e.g., sum, mean, count.)

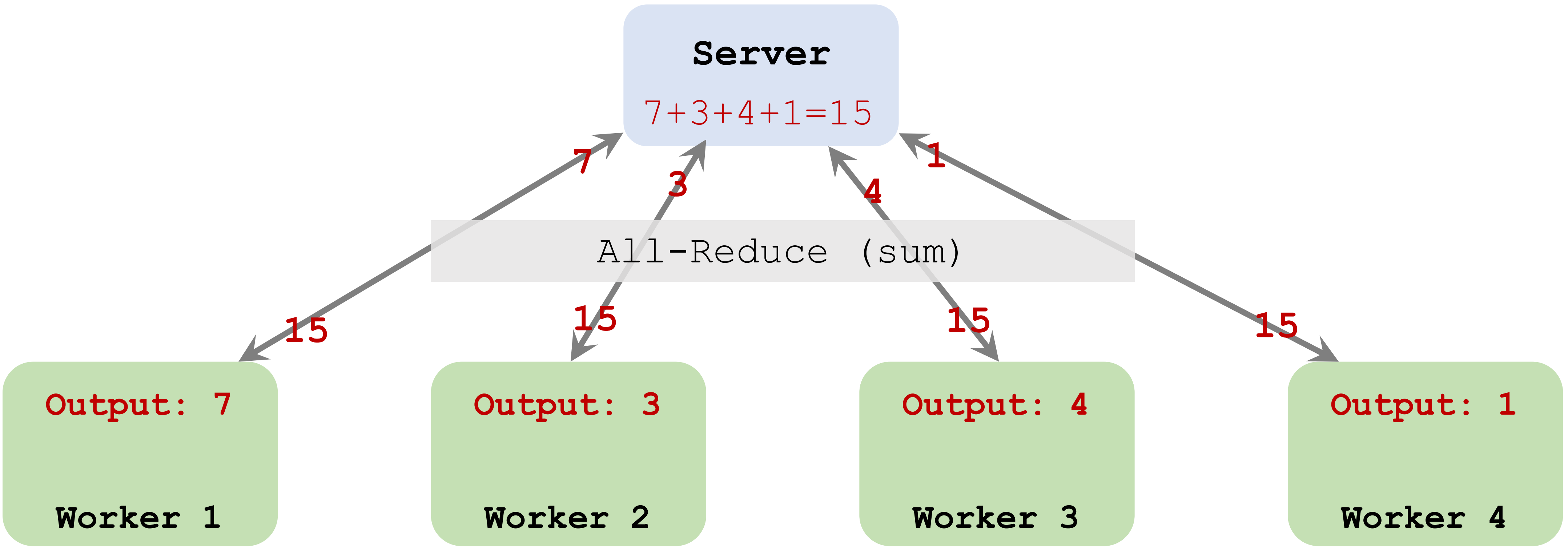


Reduce VS All-Reduce

- Reduce: The server gets the result of reduce (e.g., sum, mean, count.)
- All-Reduce: Every node gets a copies of the result of reduce.

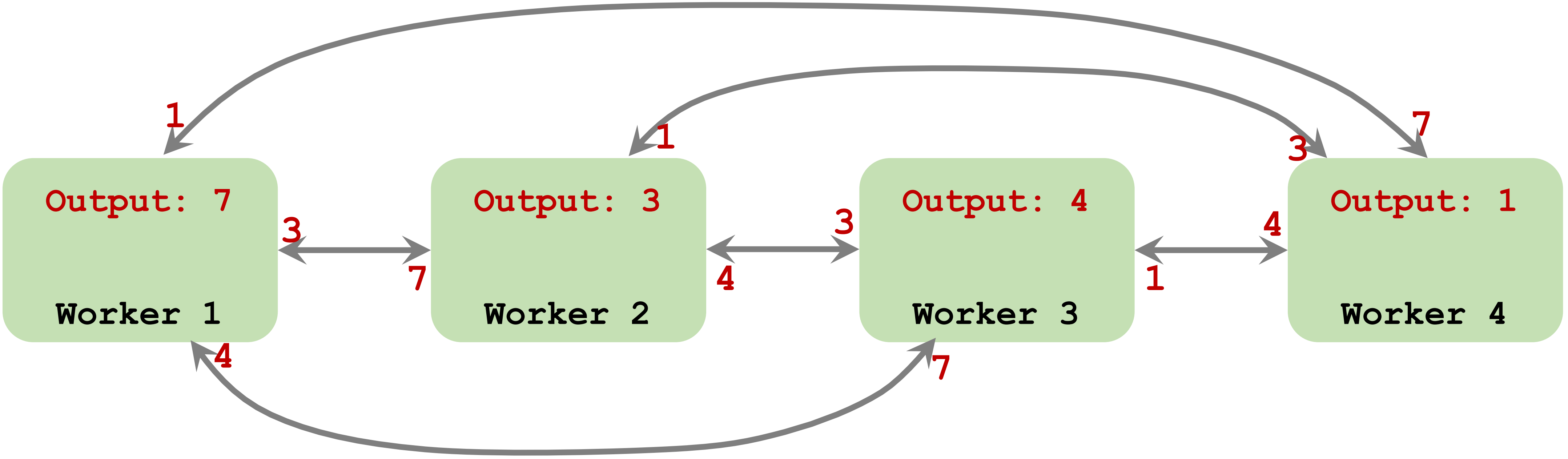
Reduce VS All-Reduce

- Reduce: The server gets the result of reduce (e.g., sum, mean, count.)
- All-Reduce: Every node gets a copies of the result of reduce.
  - E.g., all-reduce via reduce+broadcast.



Reduce VS All-Reduce

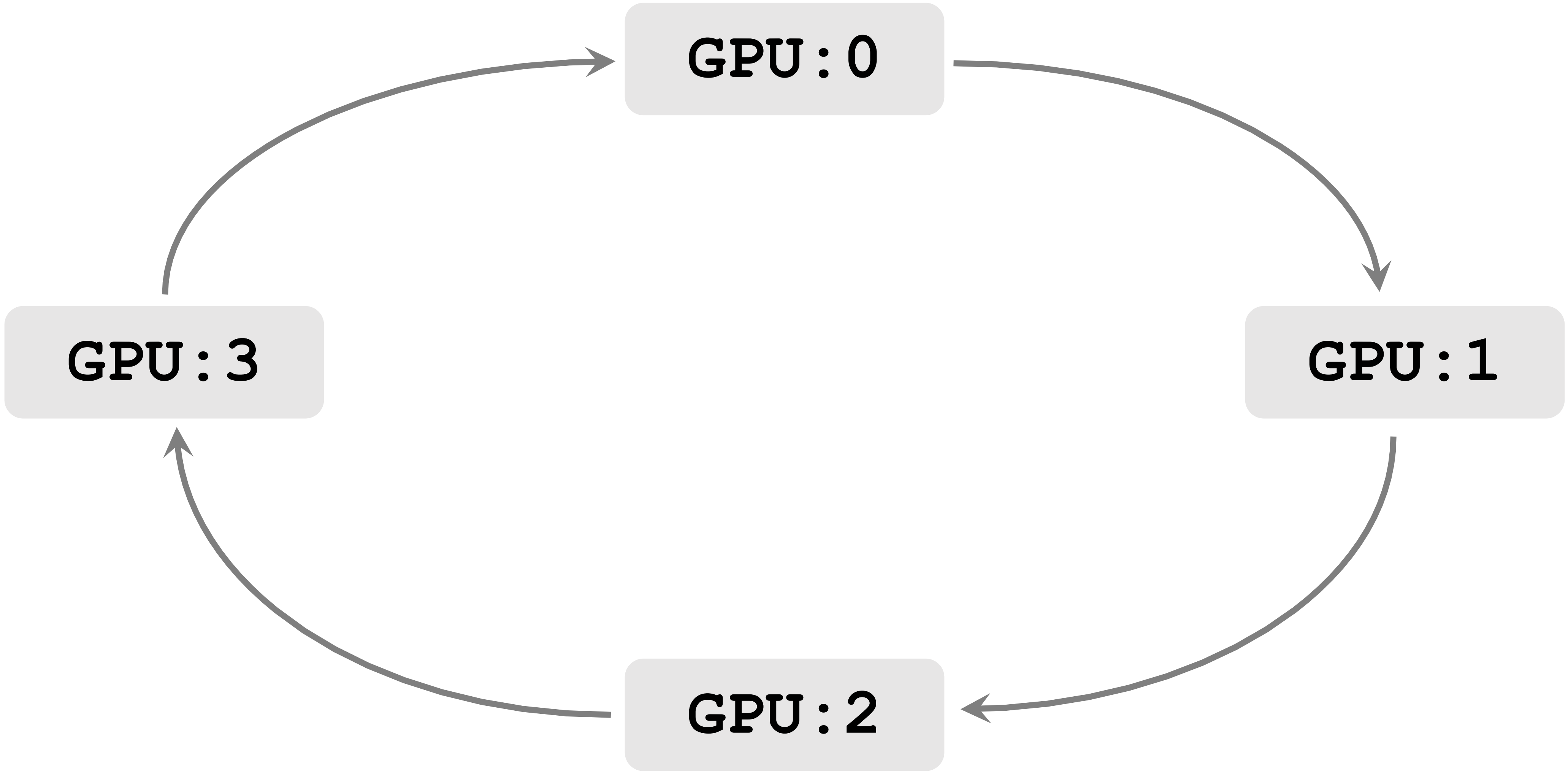
- Reduce: The server gets the result of reduce (e.g., sum, mean, count.)
- All-Reduce: Every node gets a copies of the result of reduce.
  - E.g., all-reduce via reduce+broadcast.
  - E.g., all-reduce via all-to-all communication.



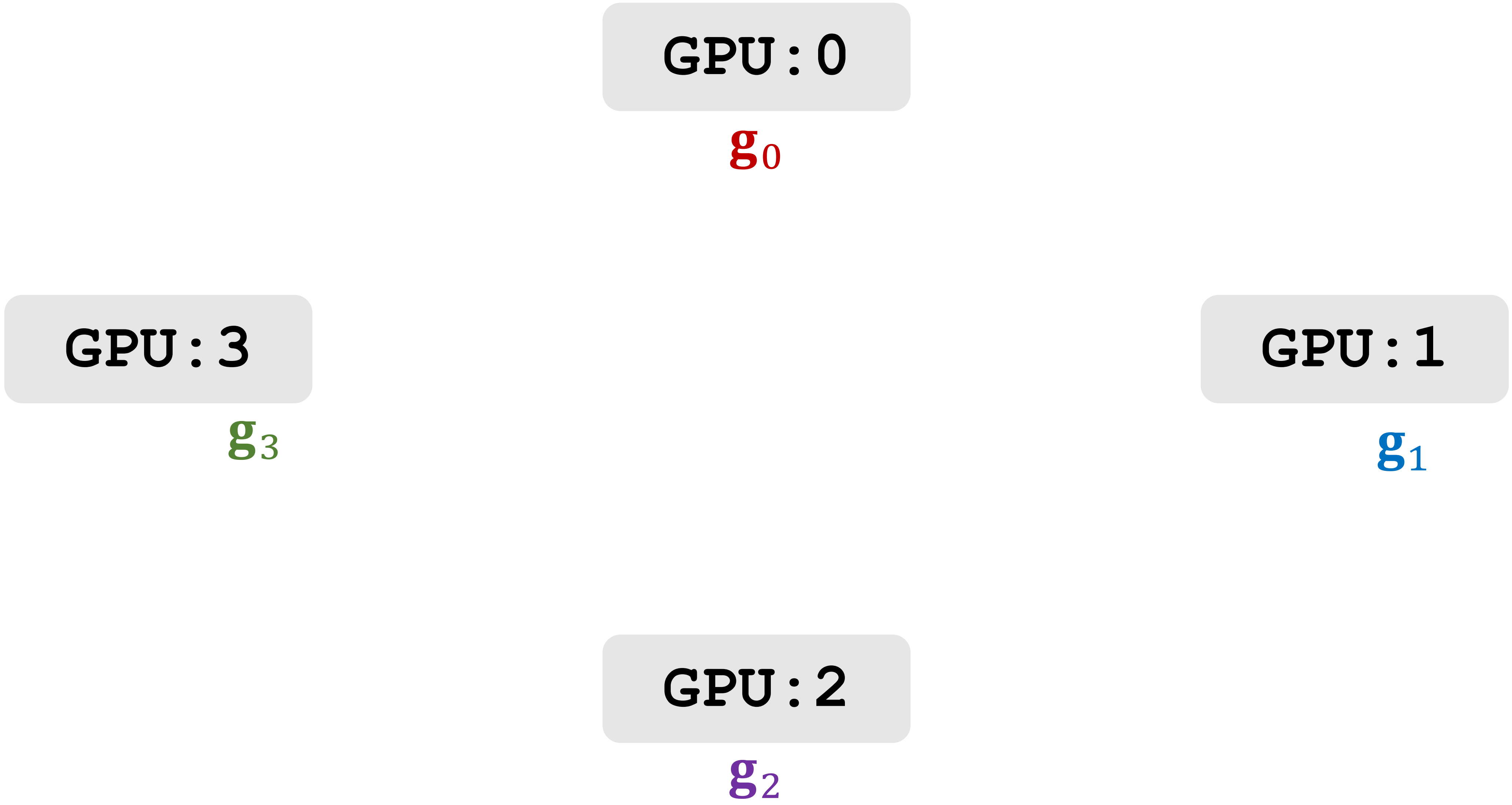
Reduce VS All-Reduce

- Reduce: The server gets the result of reduce (e.g., sum, mean, count.)
- All-Reduce: Every node gets a copies of the result of reduce.
  - E.g., all-reduce via reduce+broadcast.
  - E.g., all-reduce via all-to-all communication.
  - E.g., ring all-reduce (this lecture.)

Ring All-Reduce



Ring All-Reduce



Ring All-Reduce

GPU : 0

$\mathbf{g}_0$

GPU : 3

$\mathbf{g}_3$

GPU : 1

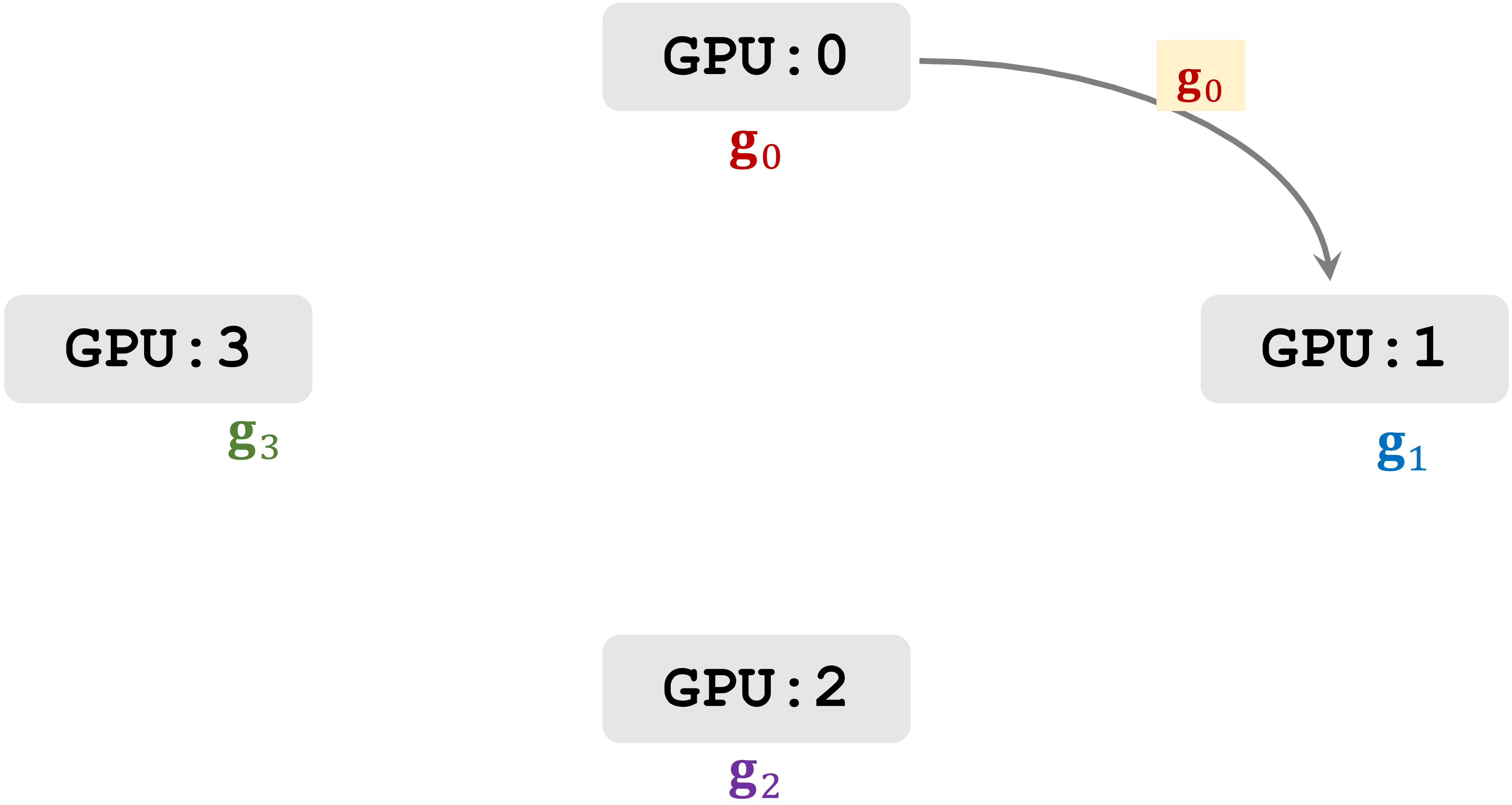
$\mathbf{g}_1$

Goal: Compute $\mathbf{g} = \mathbf{g}_0 + \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$ and send it to all the GPUs.

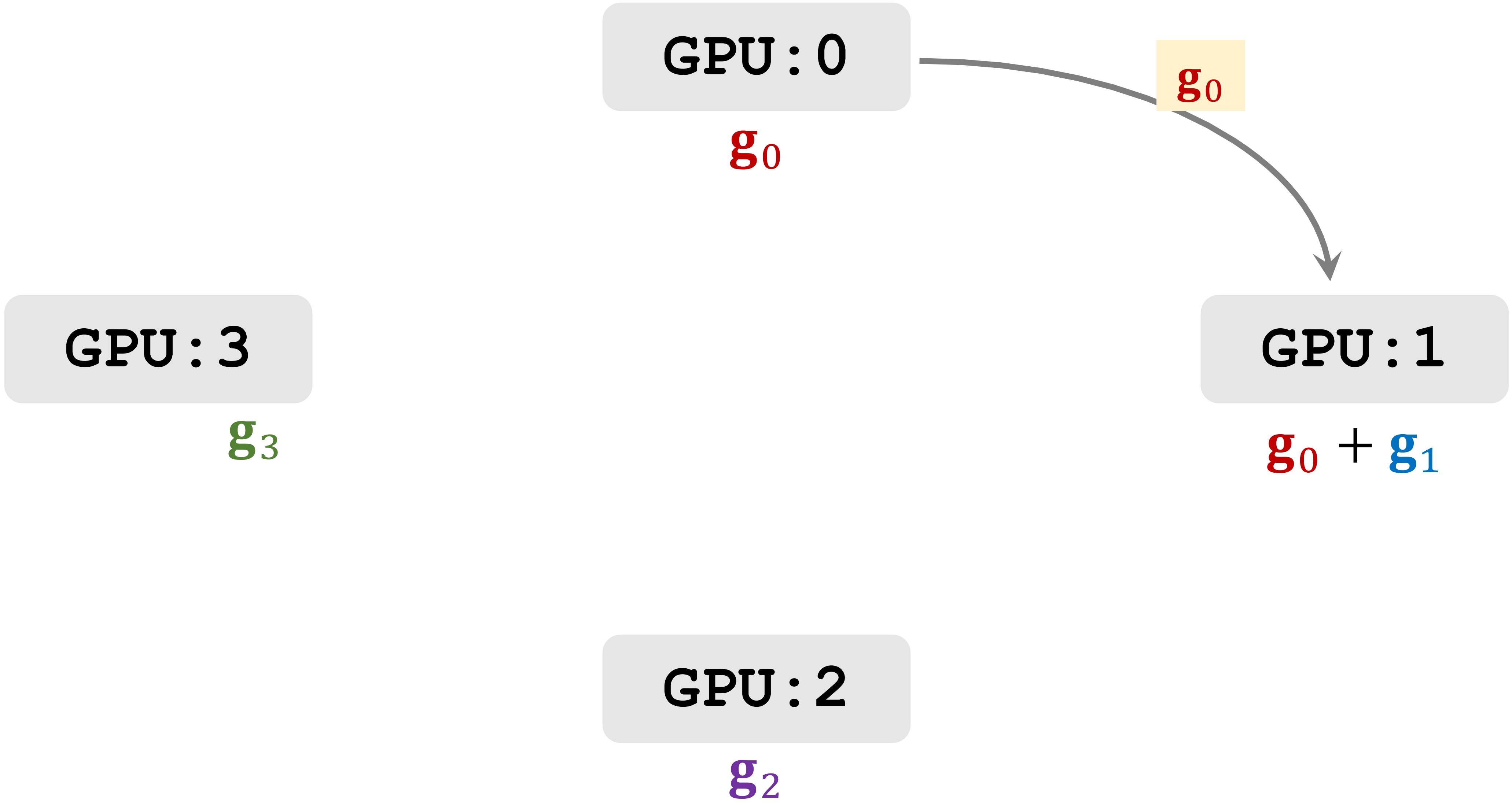
GPU : 2

$\mathbf{g}_2$

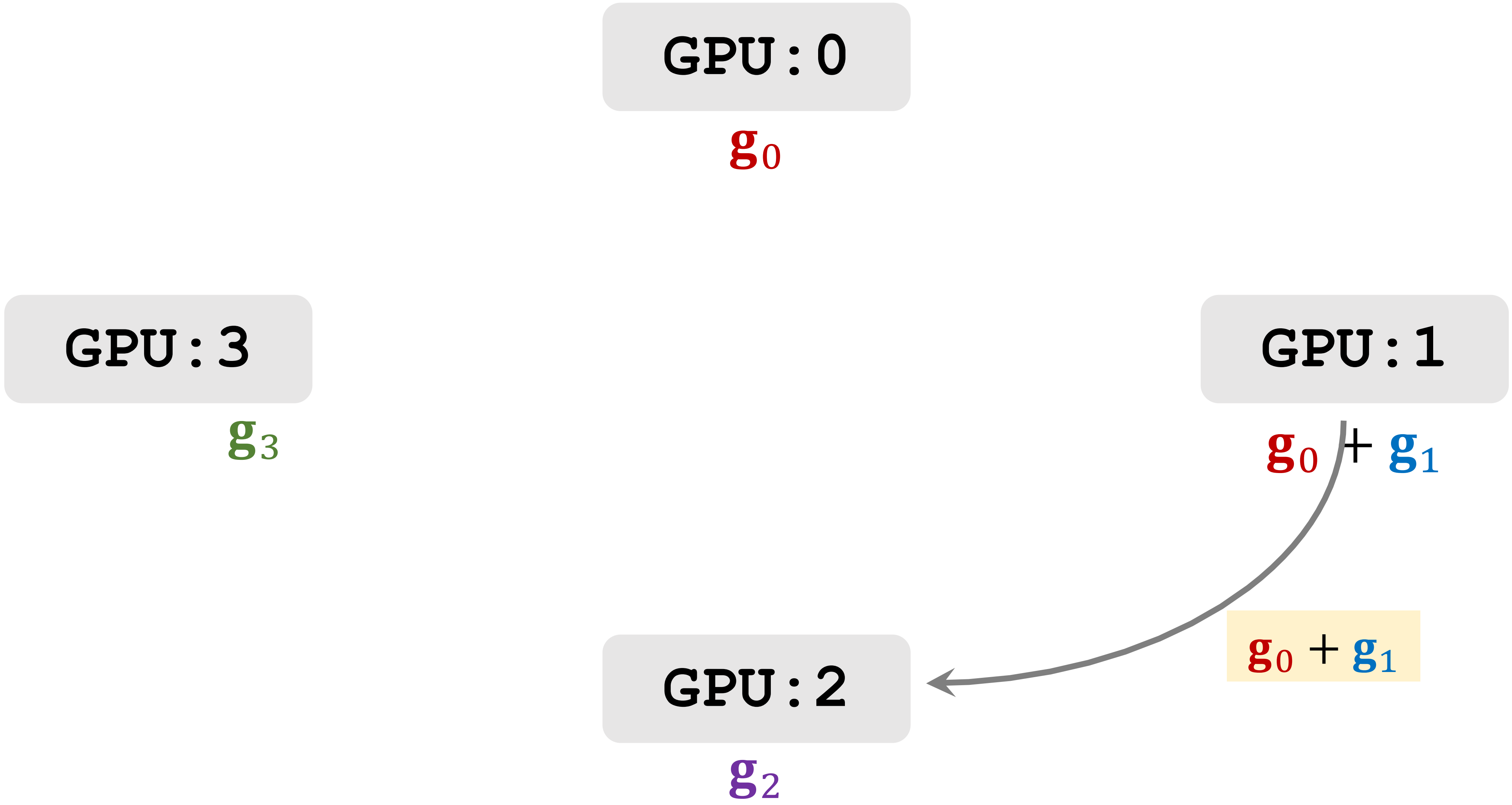
Ring All-Reduce (Naïve Approach)



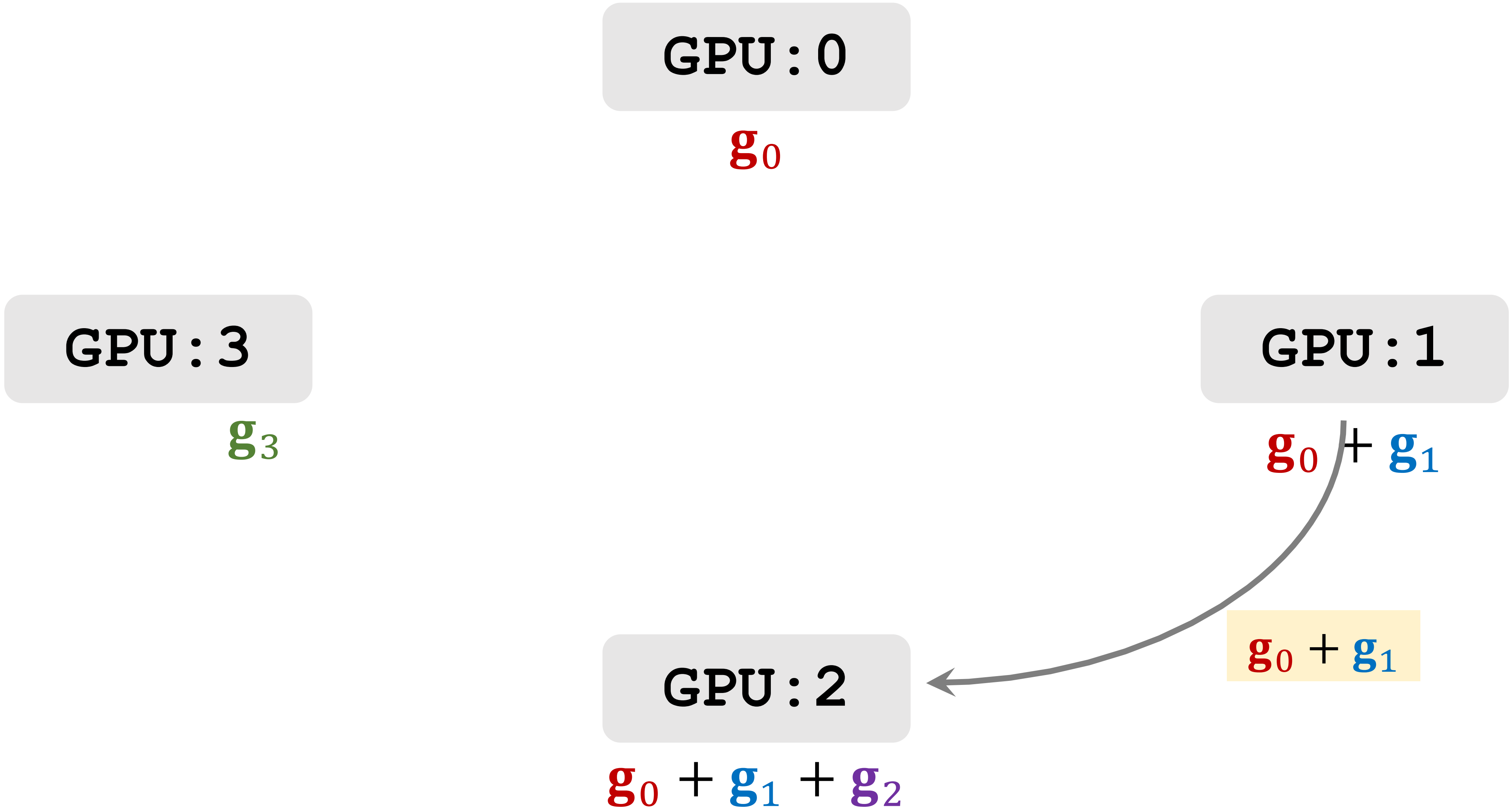
Ring All-Reduce (Naïve Approach)



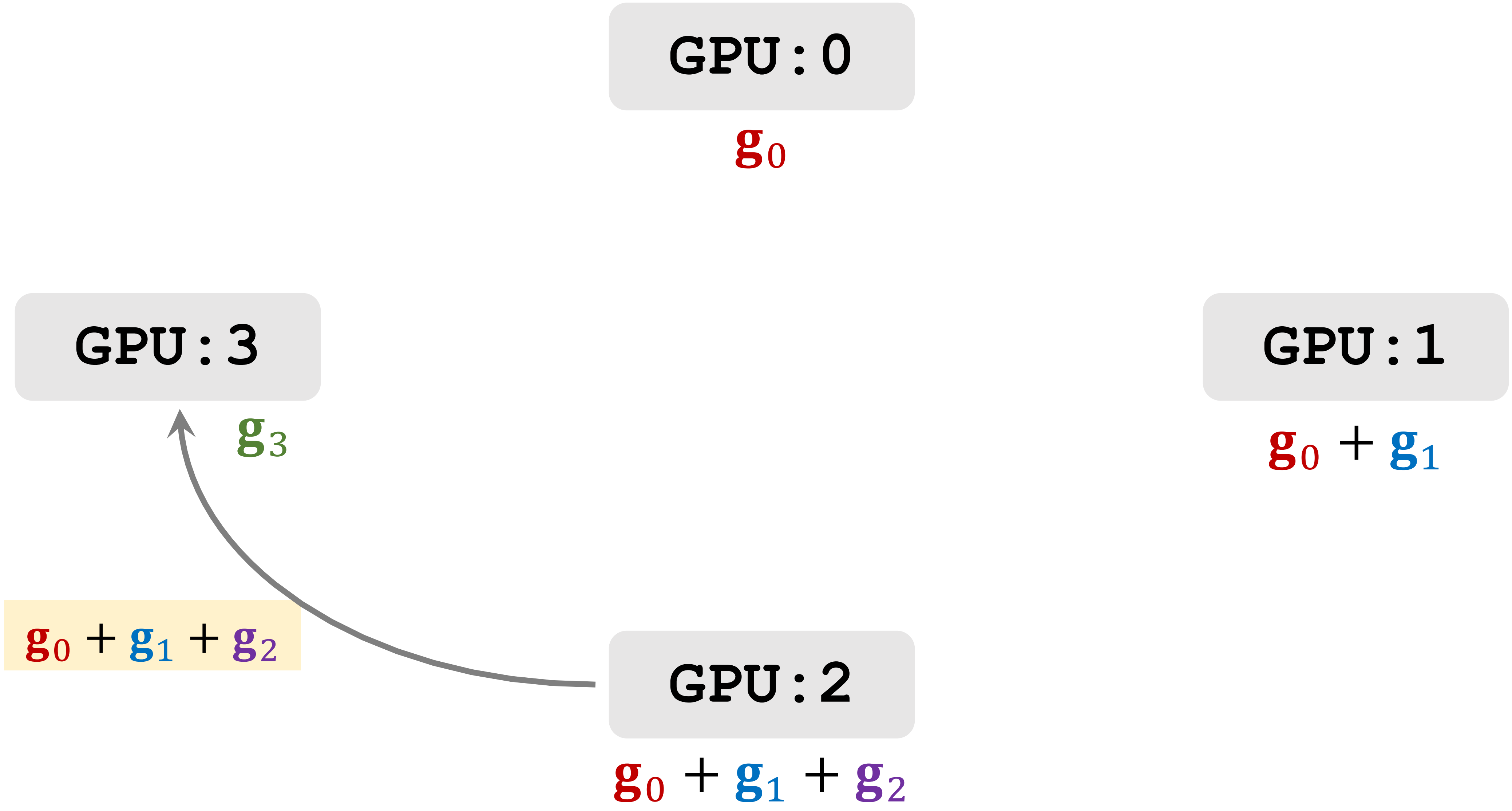
Ring All-Reduce (Naïve Approach)



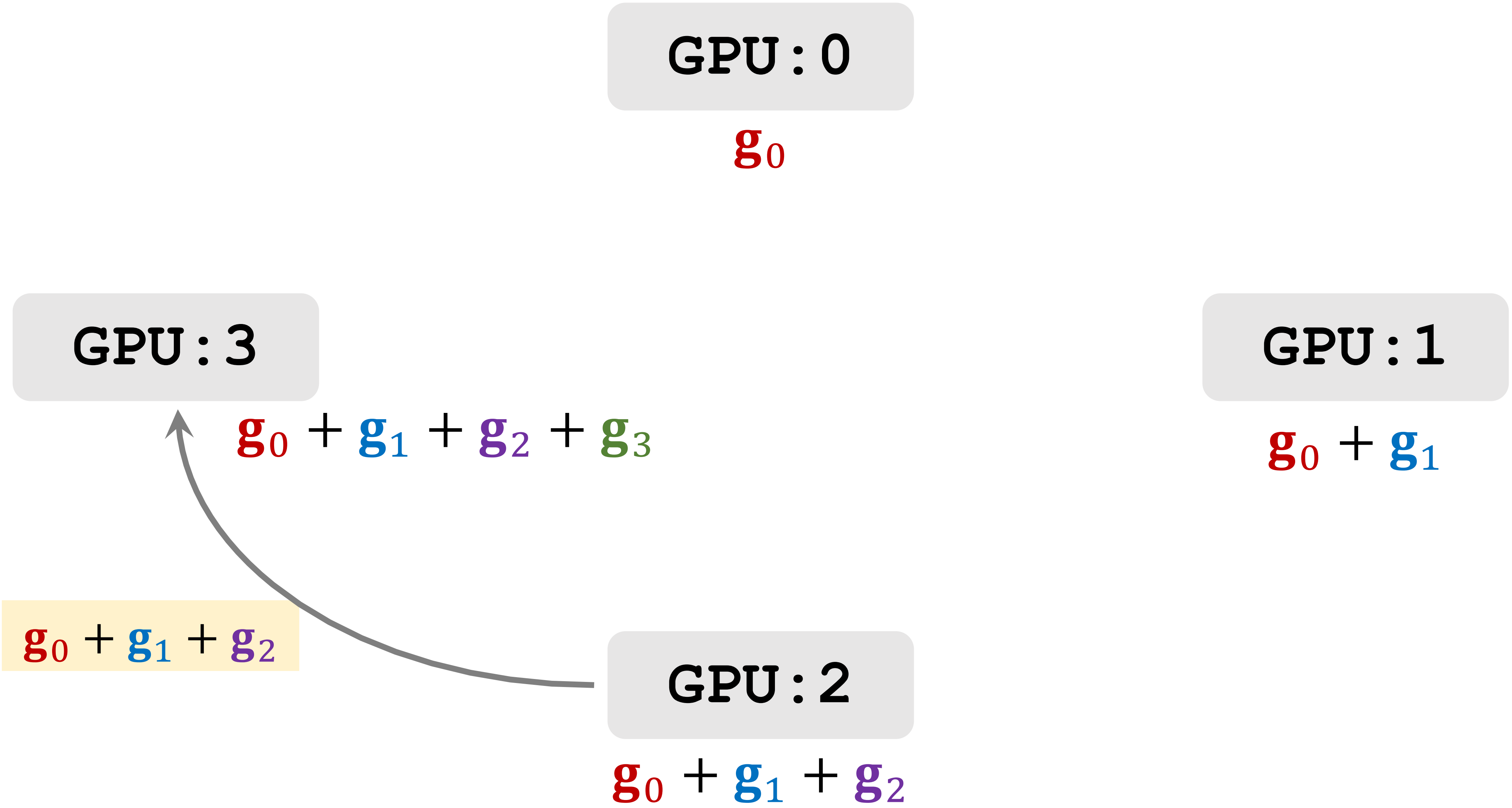
Ring All-Reduce (Naïve Approach)



Ring All-Reduce (Naïve Approach)



Ring All-Reduce (Naïve Approach)



Ring All-Reduce (Naïve Approach)

GPU : 0

$\mathbf{g}_0$

GPU : 3

$\mathbf{g} = \mathbf{g}_0 + \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$

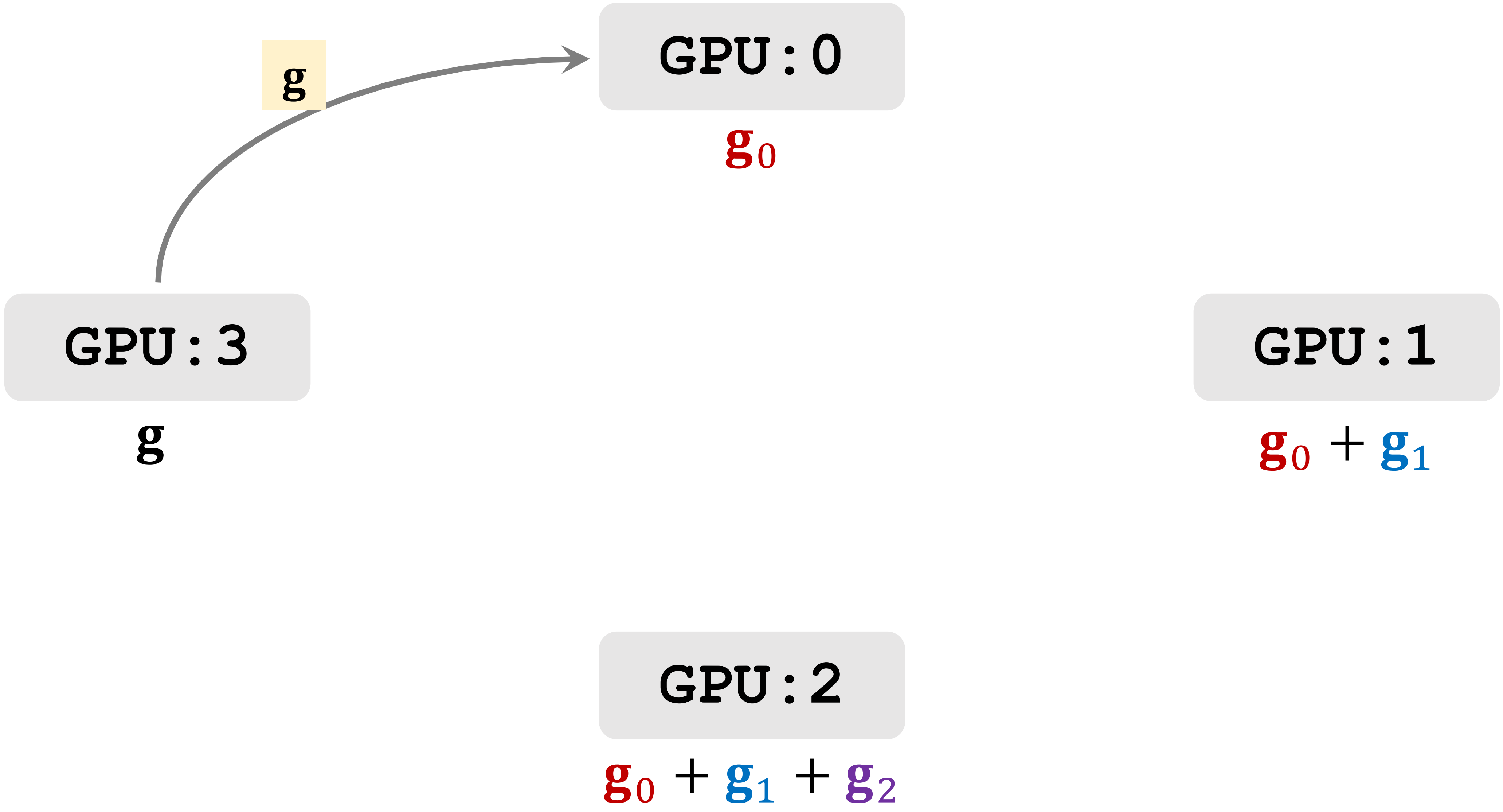
GPU : 1

$\mathbf{g}_0 + \mathbf{g}_1$

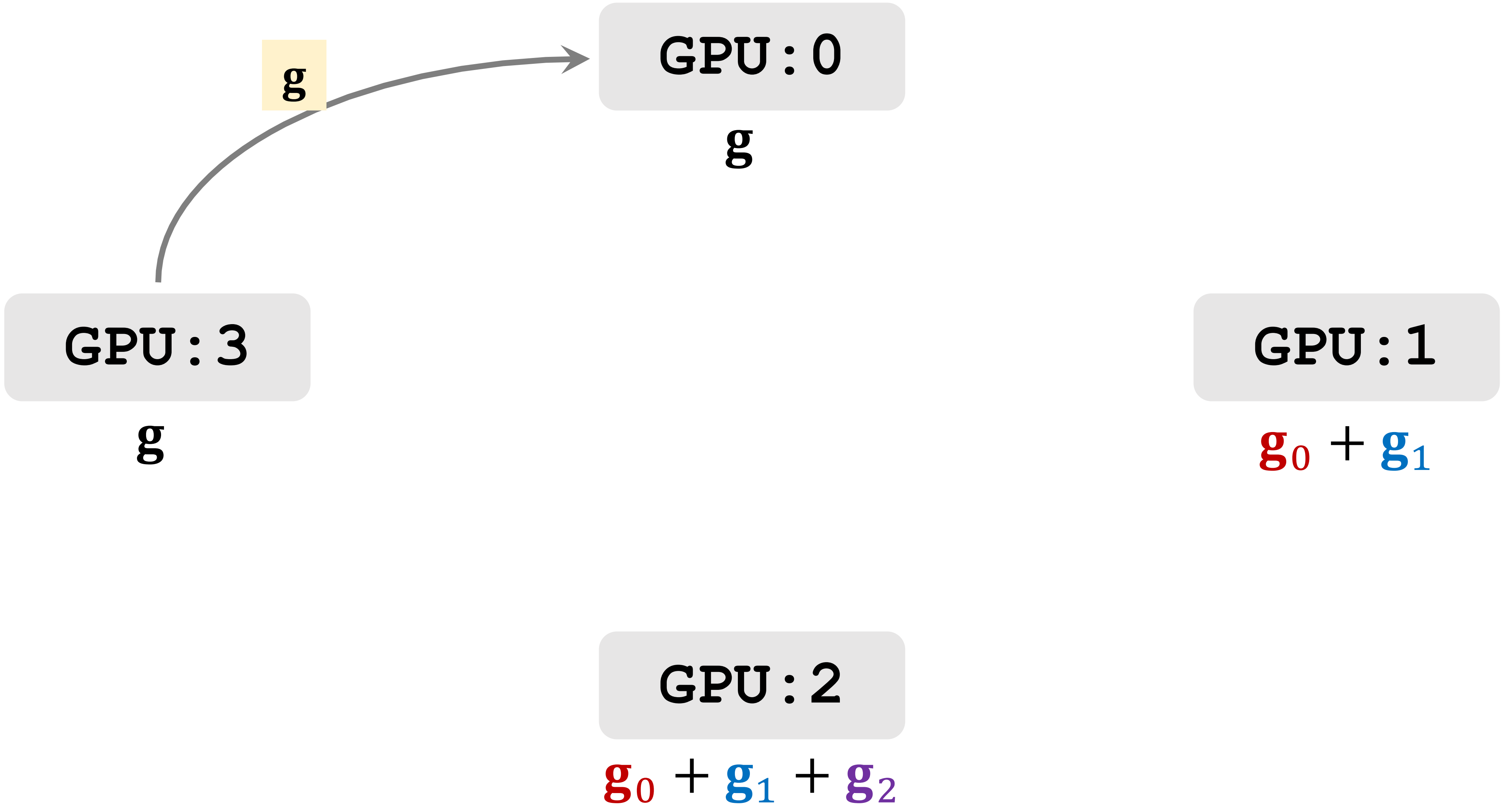
GPU : 2

$\mathbf{g}_0 + \mathbf{g}_1 + \mathbf{g}_2$

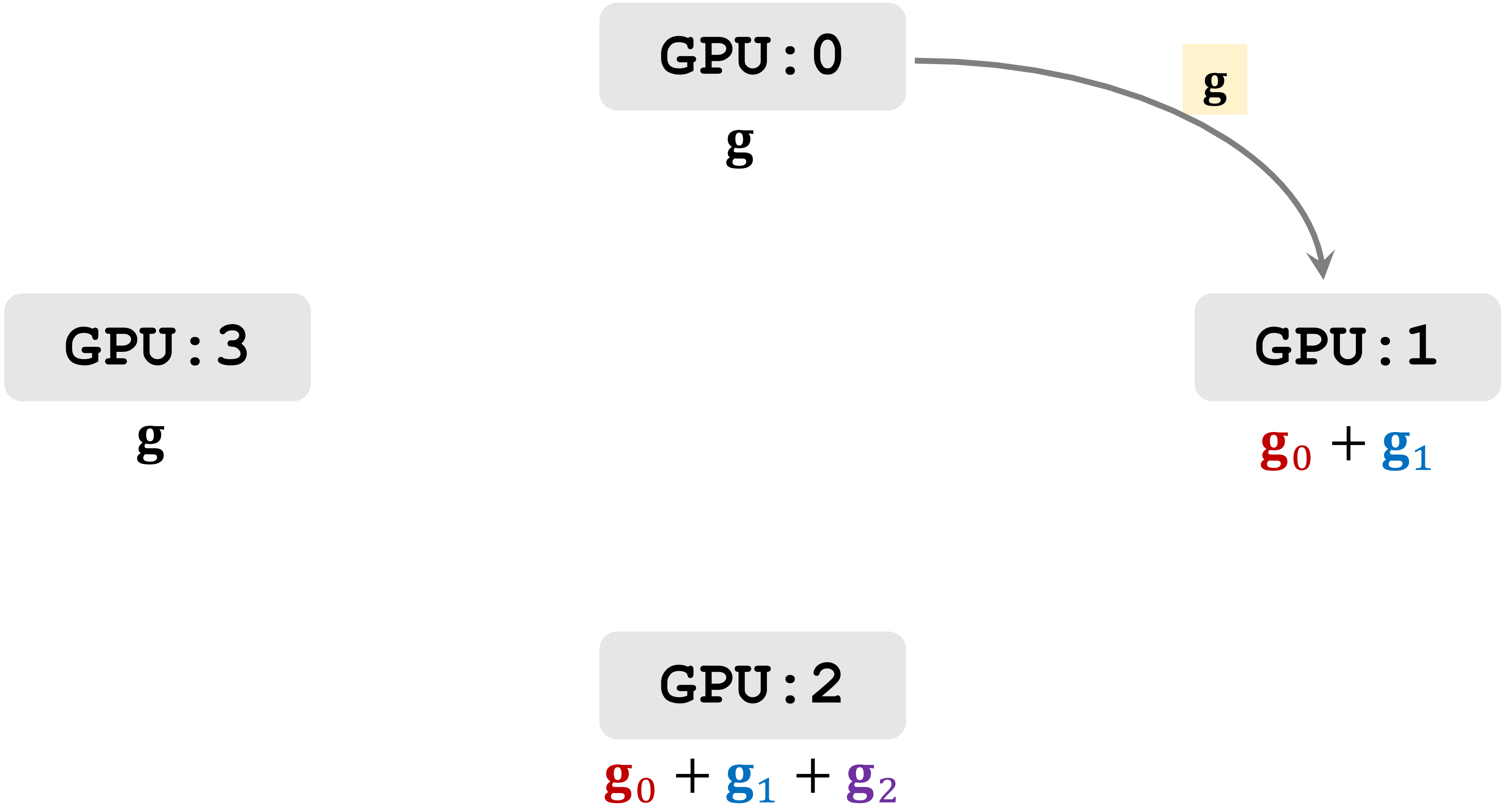
Ring All-Reduce (Naïve Approach)



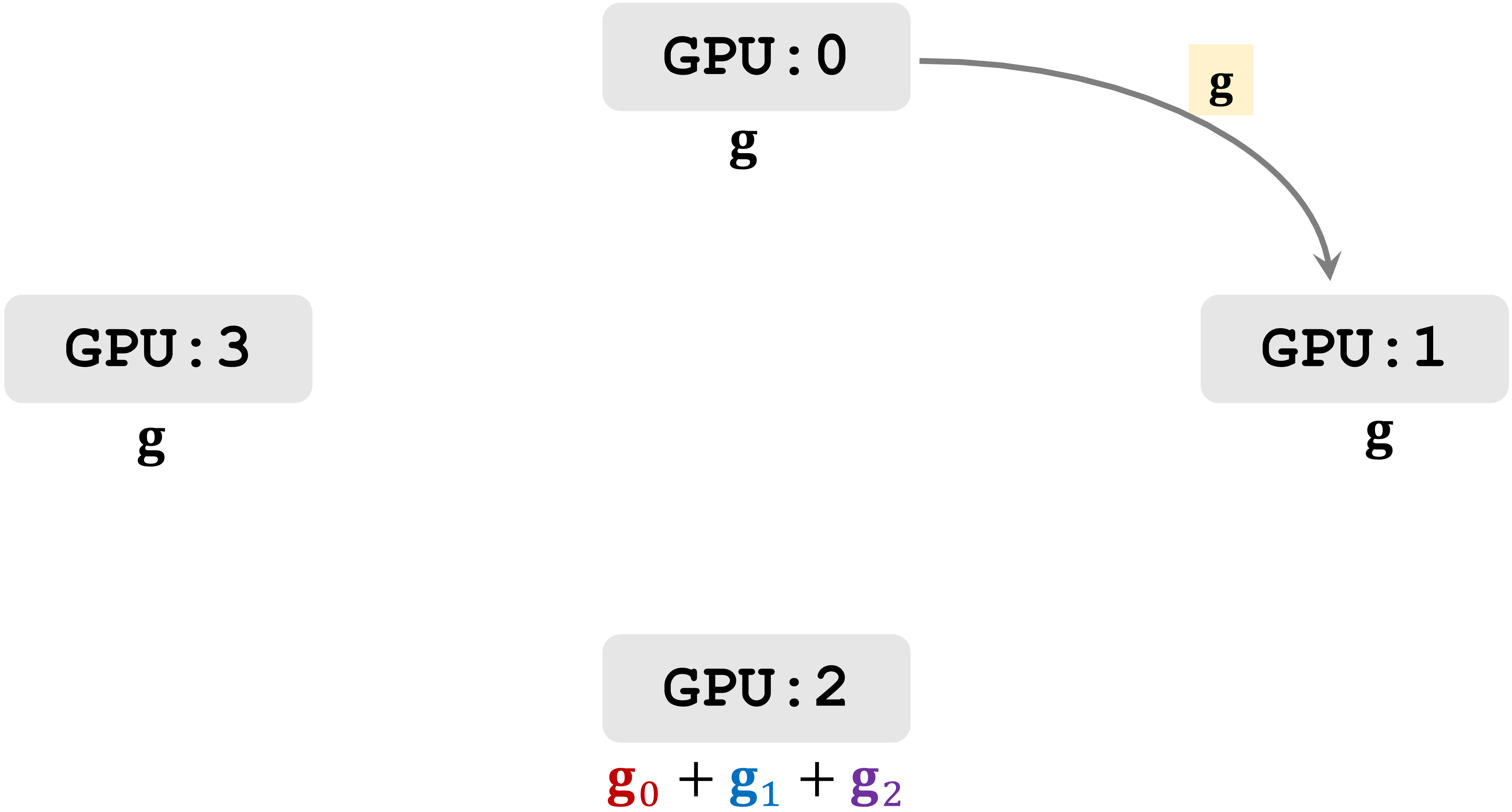
Ring All-Reduce (Naïve Approach)



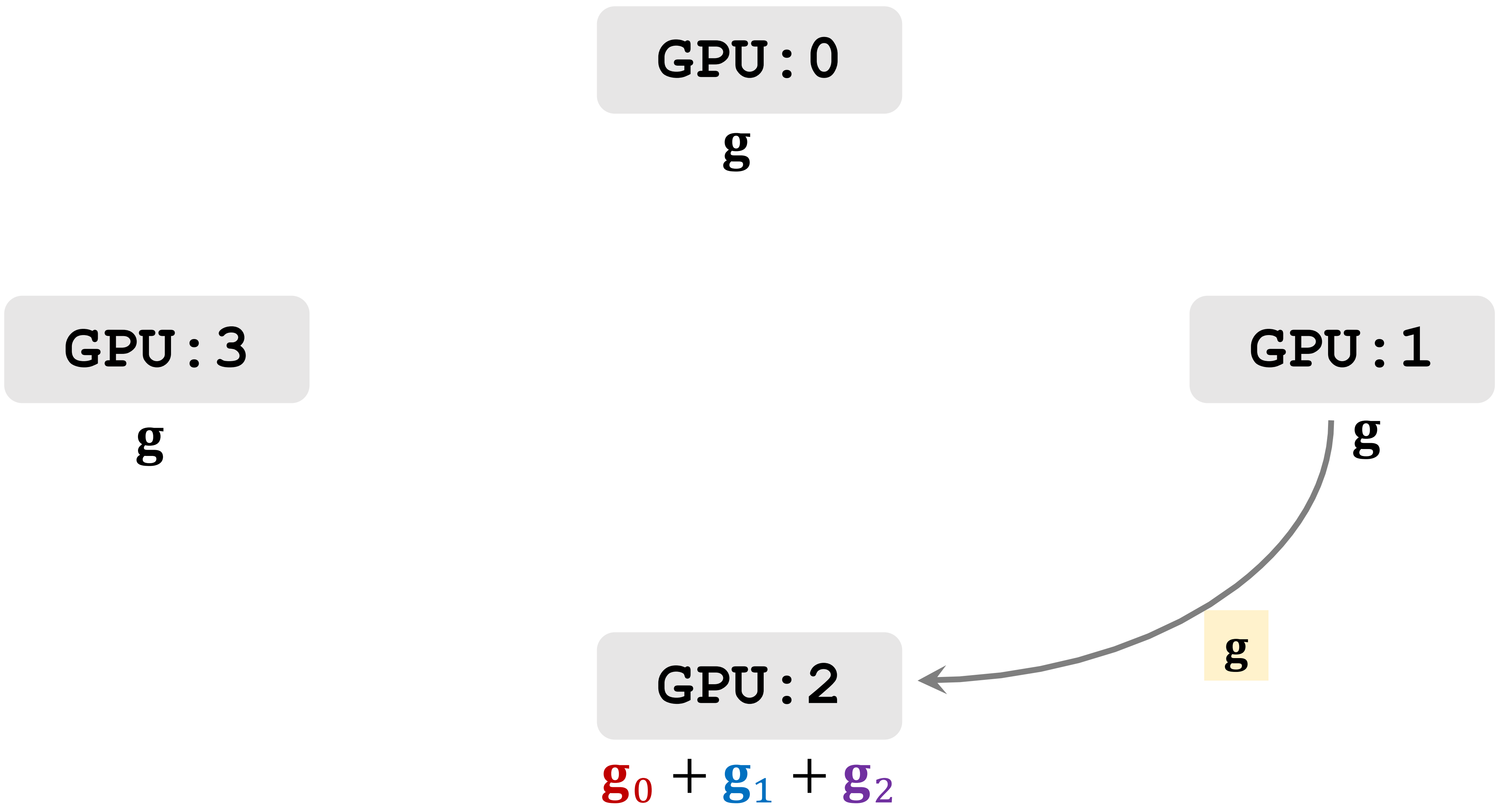
Ring All-Reduce (Naïve Approach)



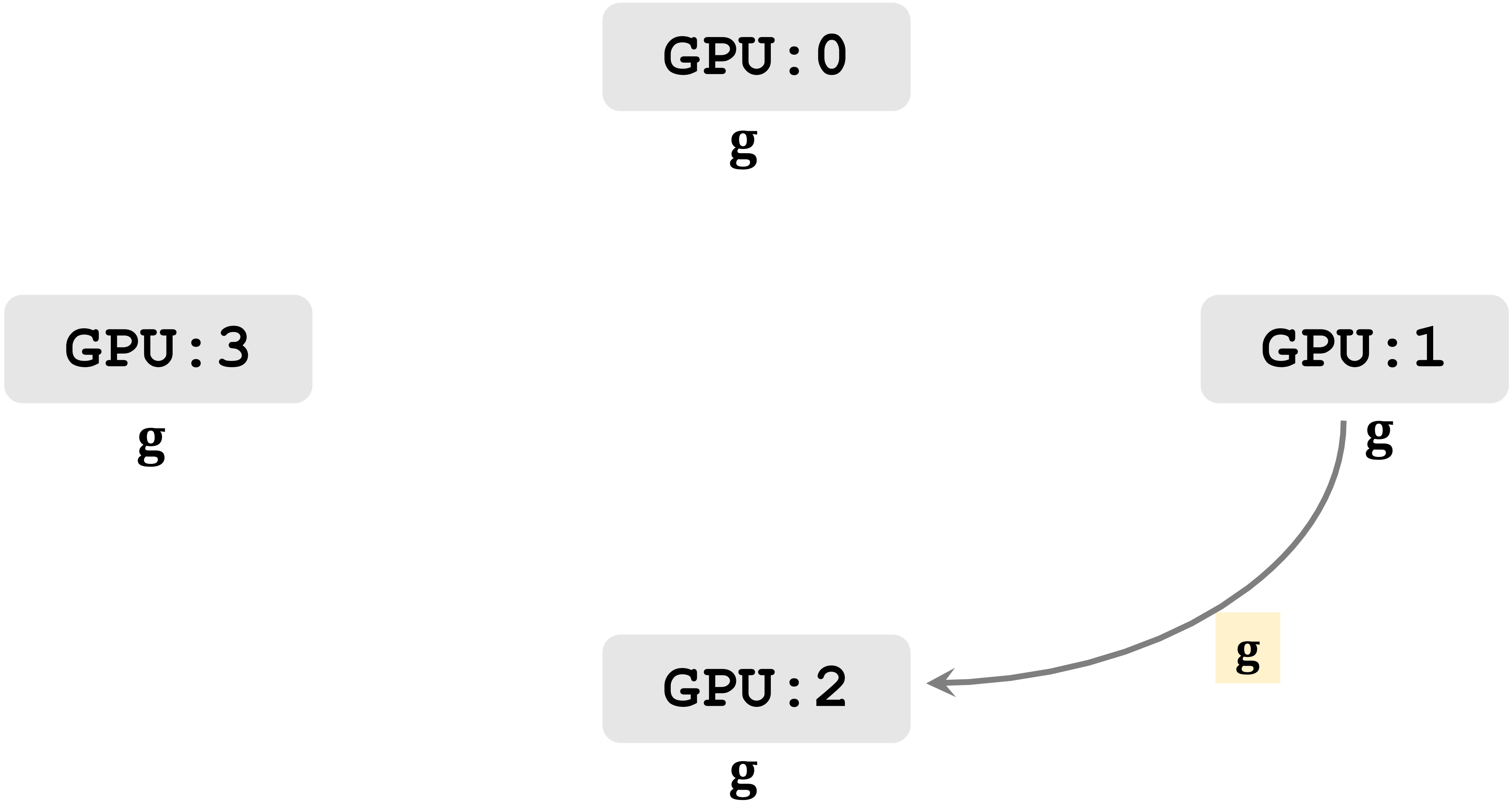
Ring All-Reduce (Naïve Approach)



Ring All-Reduce (Naïve Approach)



Ring All-Reduce (Naïve Approach)



Ring All-Reduce (Naïve Approach)

`GPU:0`

$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \mathbf{g}.$

`GPU:3`

$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \mathbf{g}.$

`GPU:1`

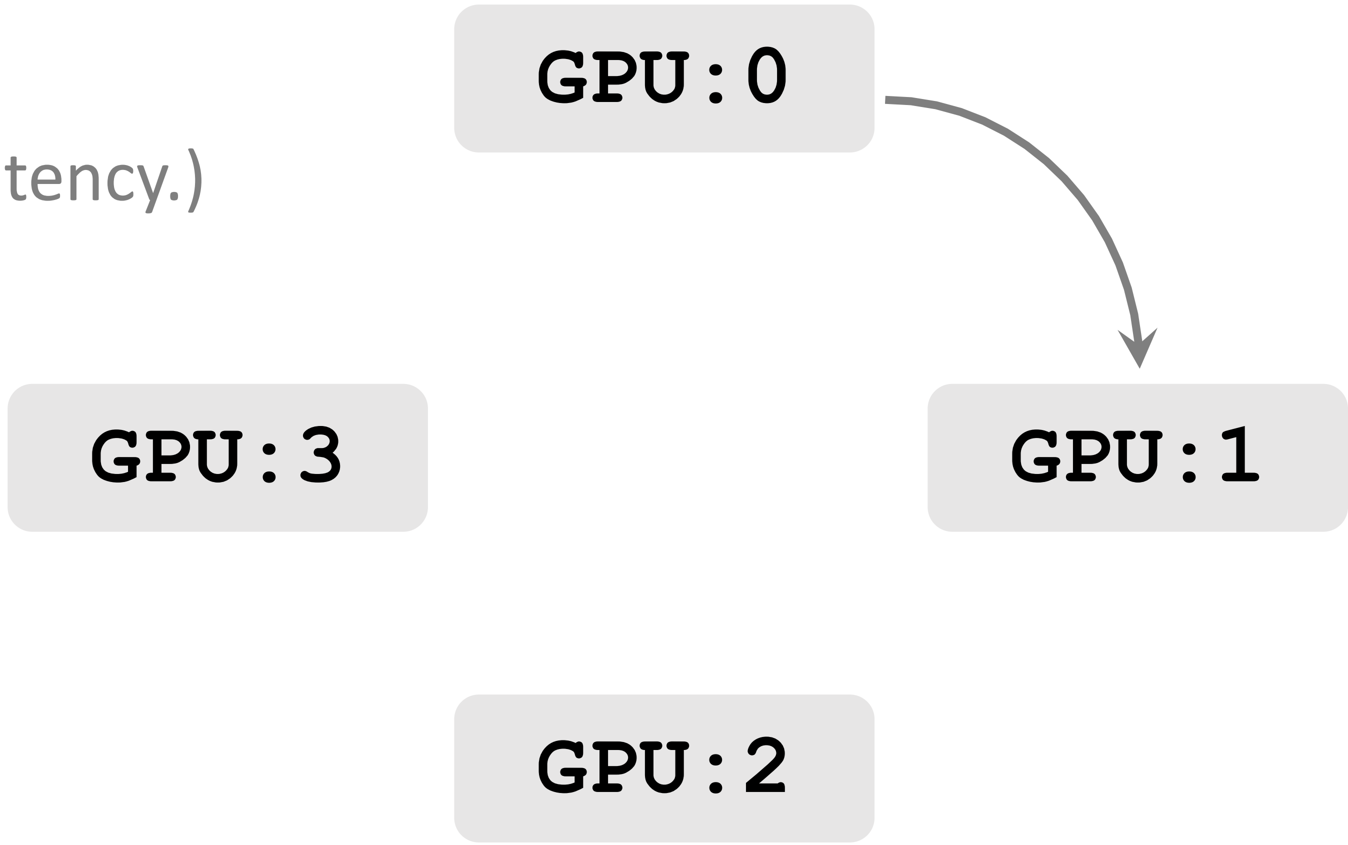
$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \mathbf{g}.$

`GPU:2`

$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \mathbf{g}.$

What is wrong with the naïve approach?

- Most computer networks are idle.
- Communication time: $\frac{md}{b}$. (Ignore latency.)
  - m: number of GPUs.
  - d: number of parameters.
  - b: network bandwidth.



Ring All-Reduce (Efficient Approach)

GPU:0

$\mathbf{g}_0 = [\mathbf{a}_0;\ \mathbf{b}_0;\ \mathbf{c}_0;\ \mathbf{d}_0]$

GPU:1

$\mathbf{g}_1 = [\mathbf{a}_1;\ \mathbf{b}_1;\ \mathbf{c}_1;\ \mathbf{d}_1]$

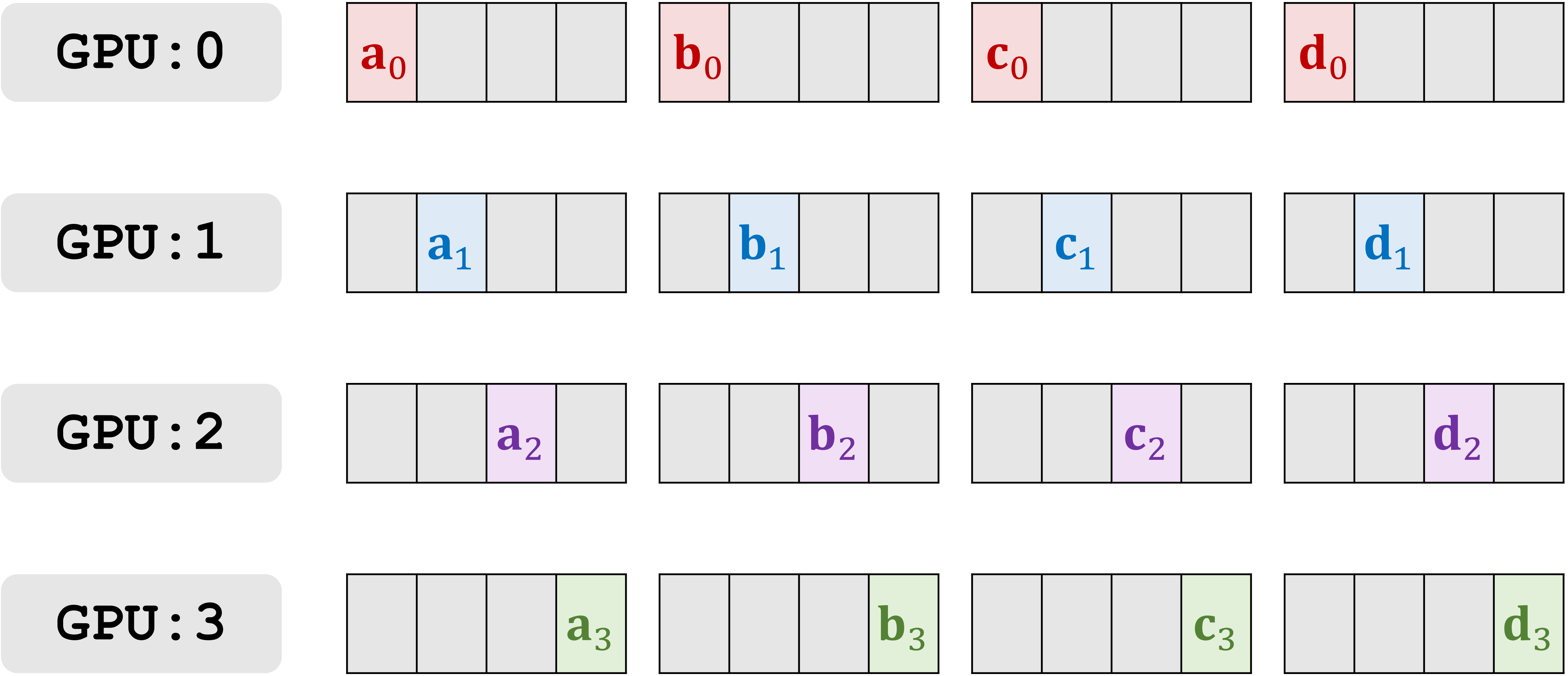
GPU:2

$\mathbf{g}_2 = [\mathbf{a}_2;\ \mathbf{b}_2;\ \mathbf{c}_2;\ \mathbf{d}_2]$

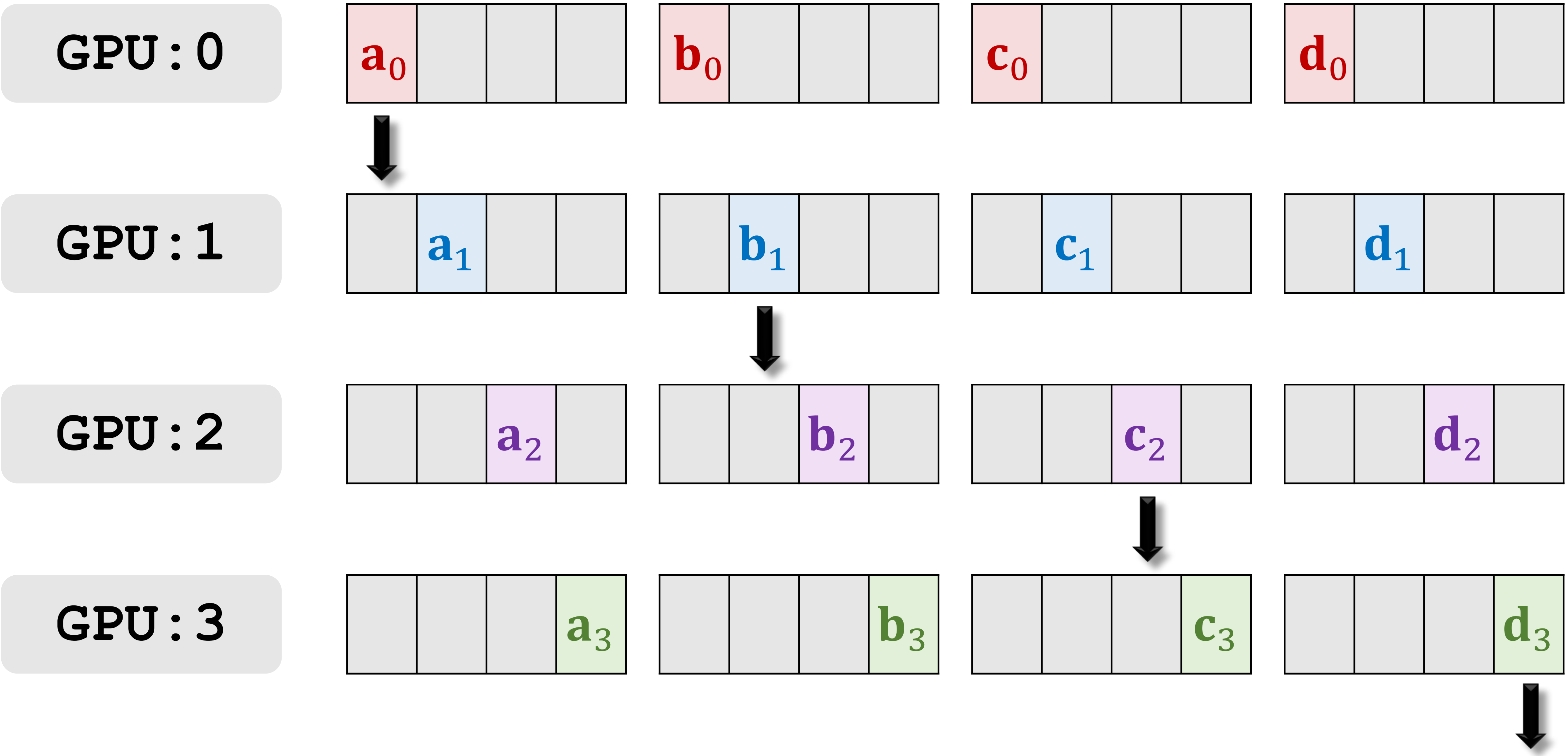
GPU:3

$\mathbf{g}_3 = [\mathbf{a}_3;\ \mathbf{b}_3;\ \mathbf{c}_3;\ \mathbf{d}_3]$

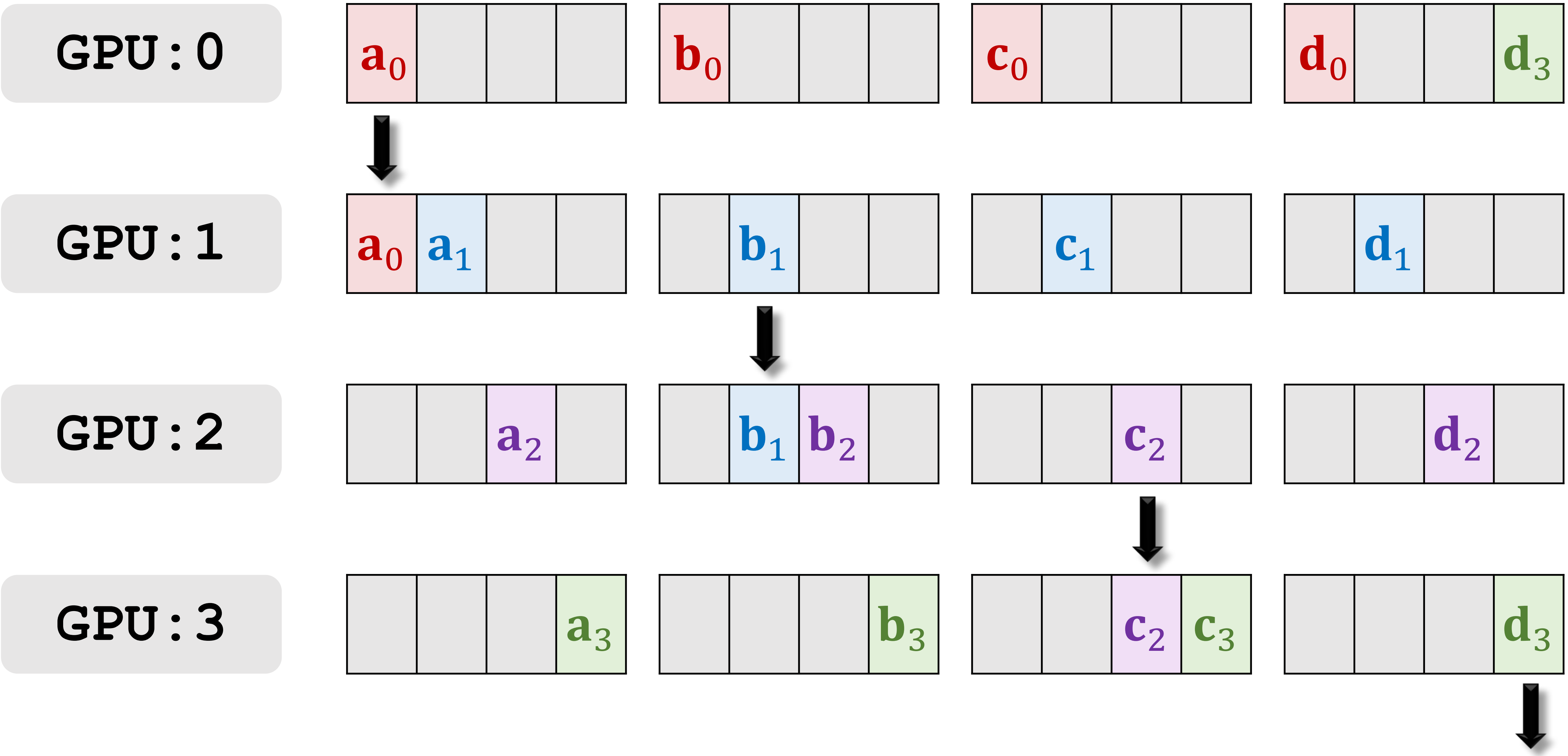
Ring All-Reduce (Efficient Approach)



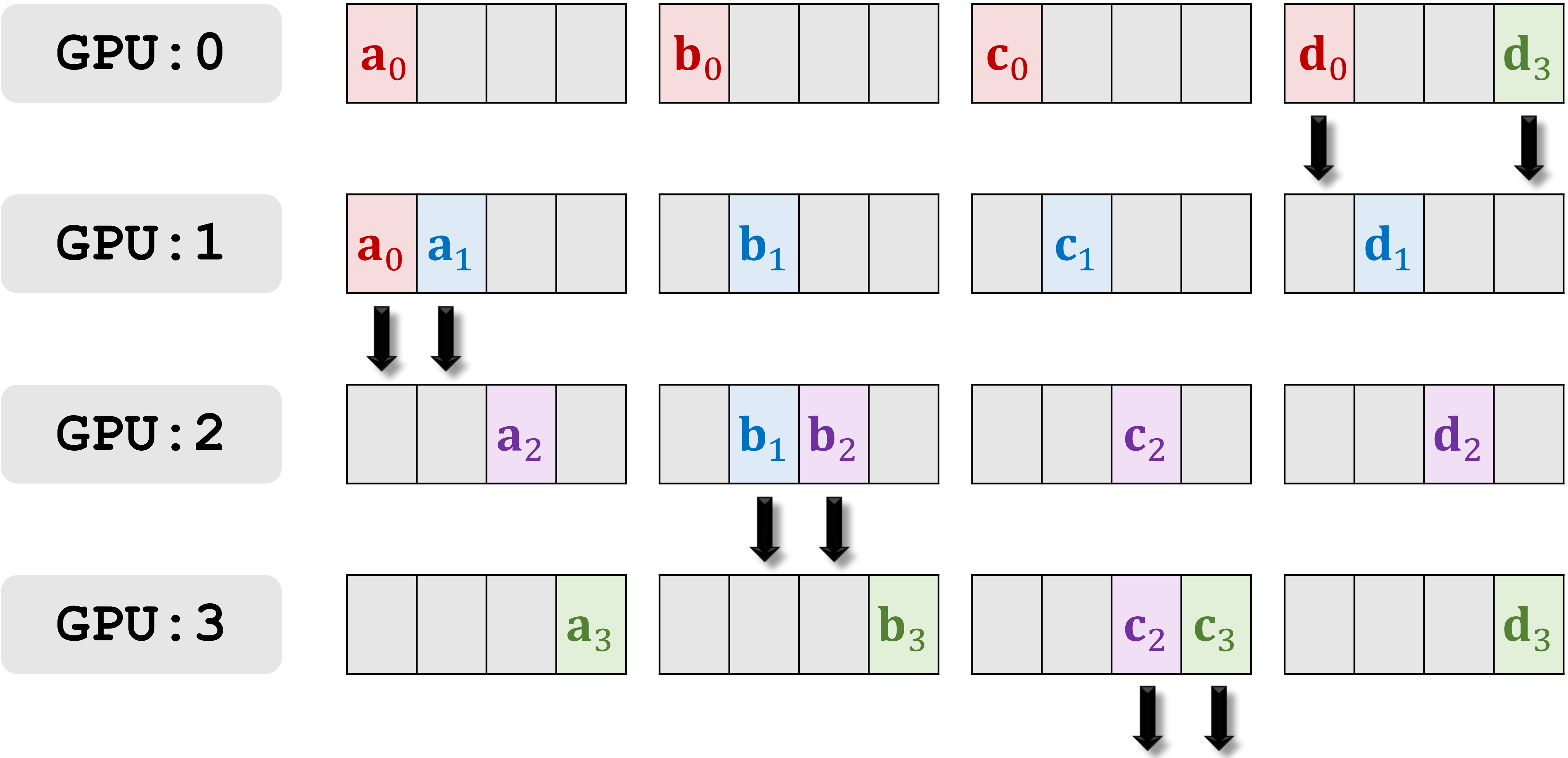
Ring All-Reduce (Efficient Approach)



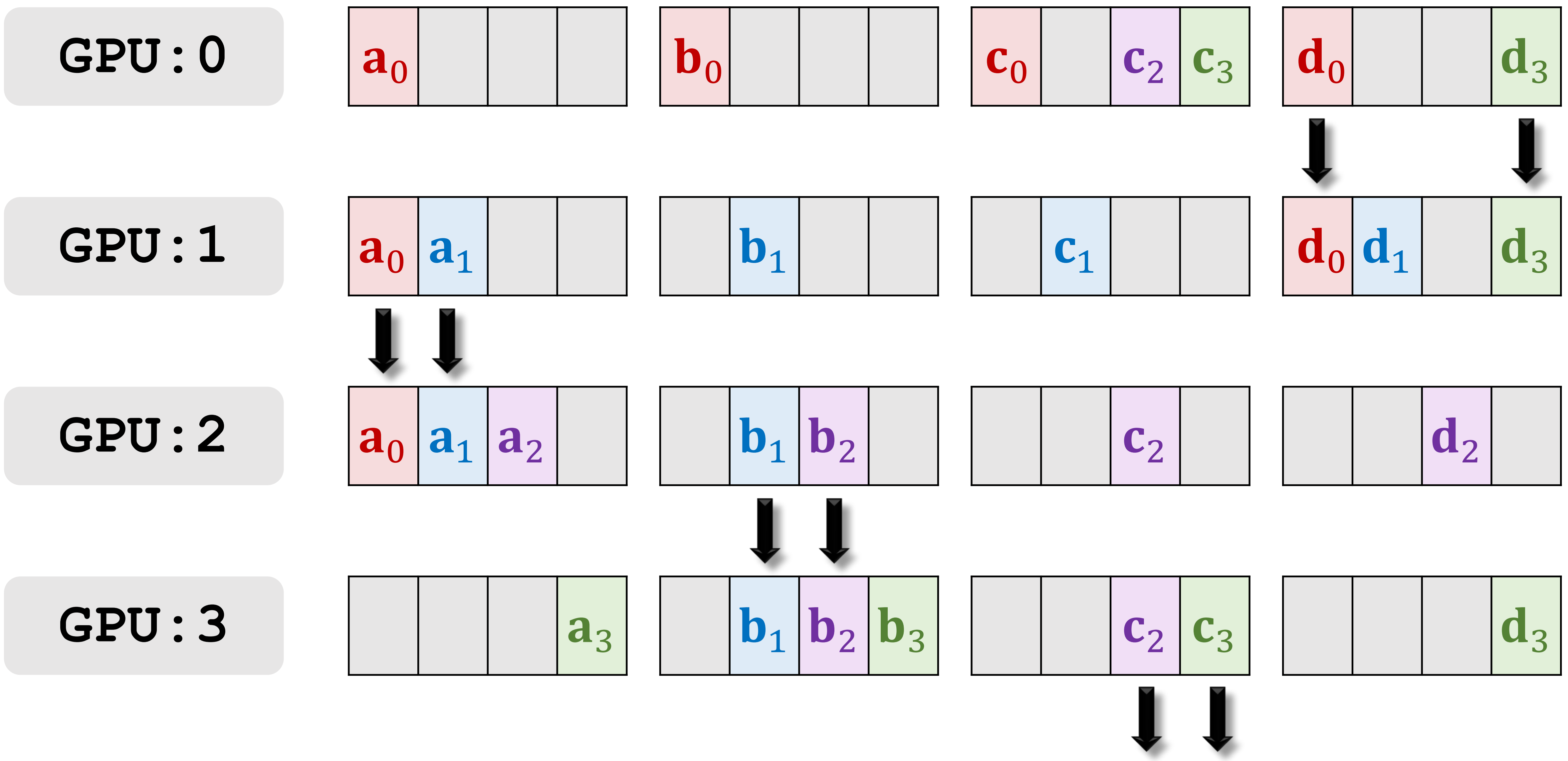
Ring All-Reduce (Efficient Approach)



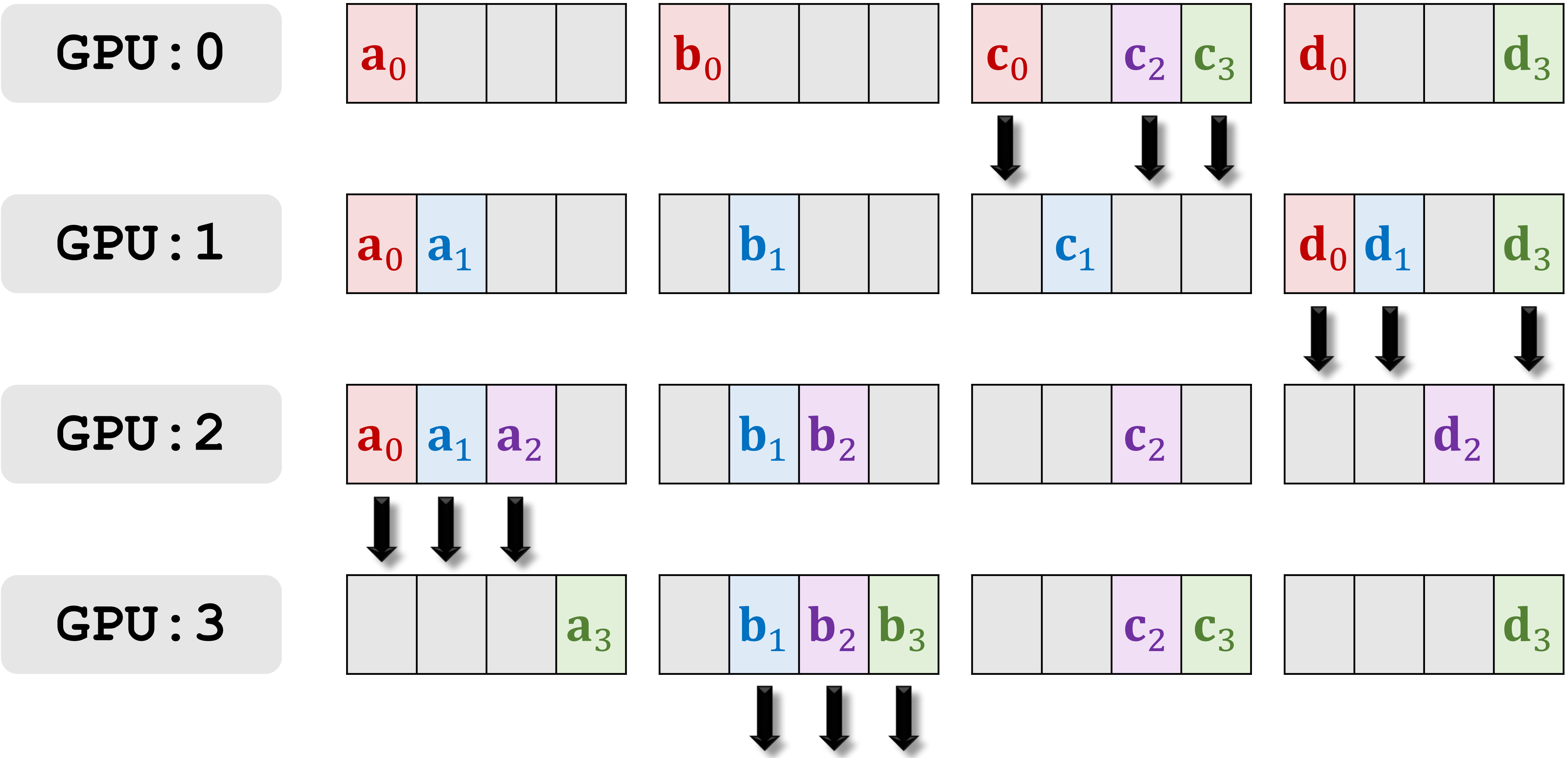
Ring All-Reduce (Efficient Approach)



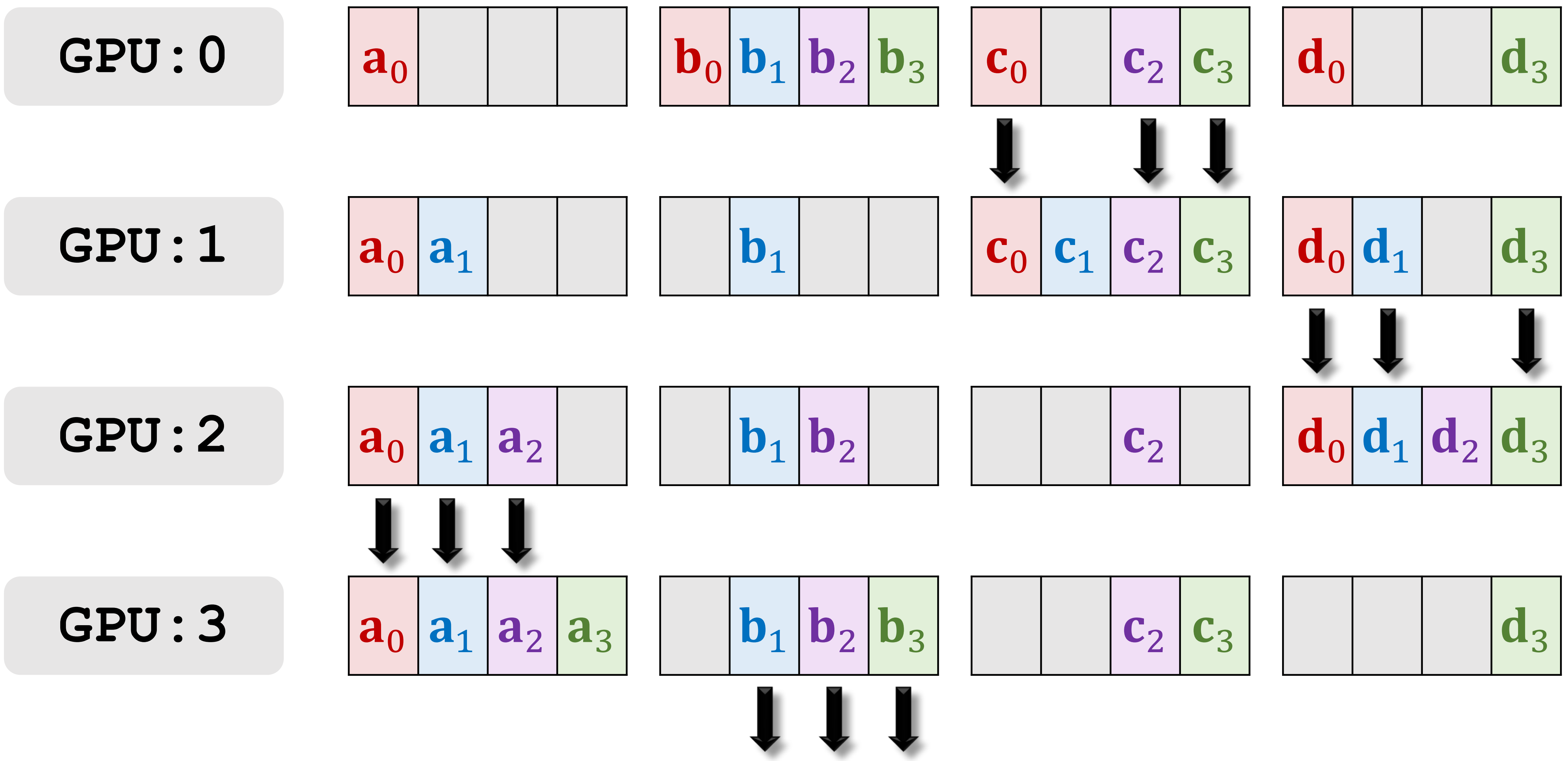
Ring All-Reduce (Efficient Approach)



Ring All-Reduce (Efficient Approach)



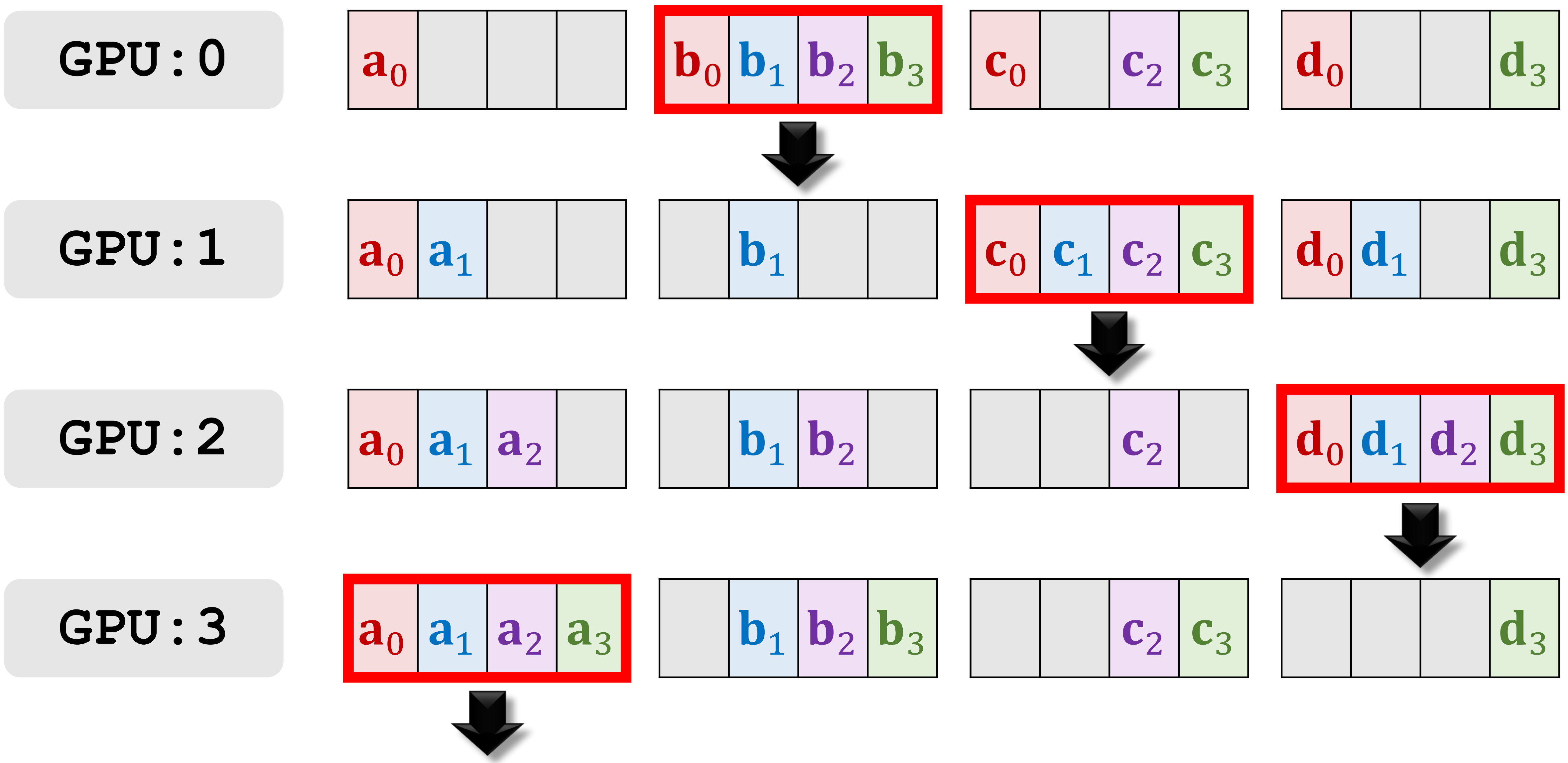
Ring All-Reduce (Efficient Approach)



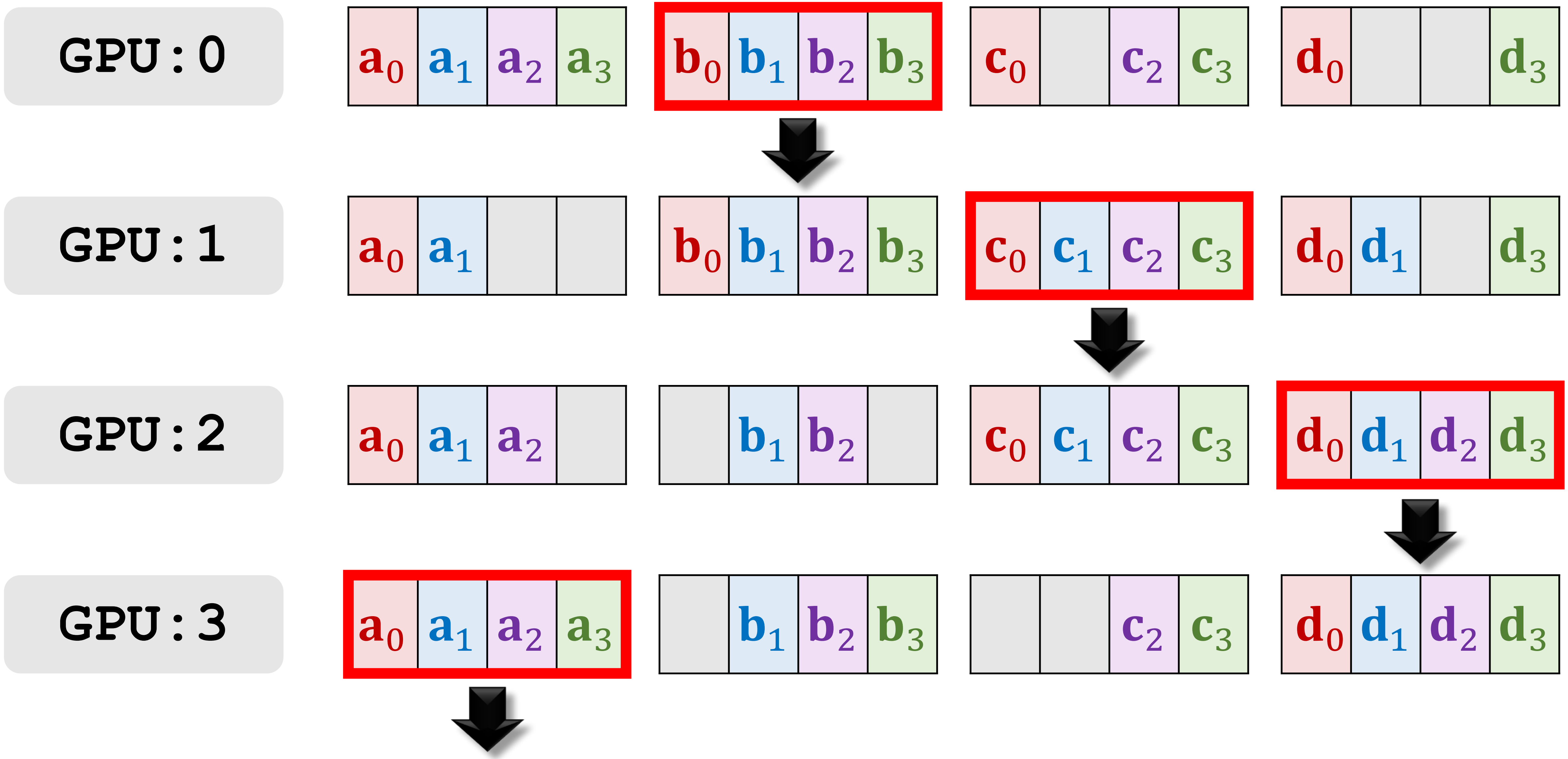
Ring All-Reduce (Efficient Approach)

GPU:0	a_0				b_0	b_1	b_2	b_3	c_0		c_2	c_3	d_0			d_3
GPU:1	a_0	a_1				b_1			c_0	c_1	c_2	c_3	d_0	d_1		d_3
GPU:2	a_0	a_1	a_2			b_1	b_2				c_2		d_0	d_1	d_2	d_3
GPU:3	a_0	a_1	a_2	a_3		b_1	b_2	b_3			c_2	c_3				d_3

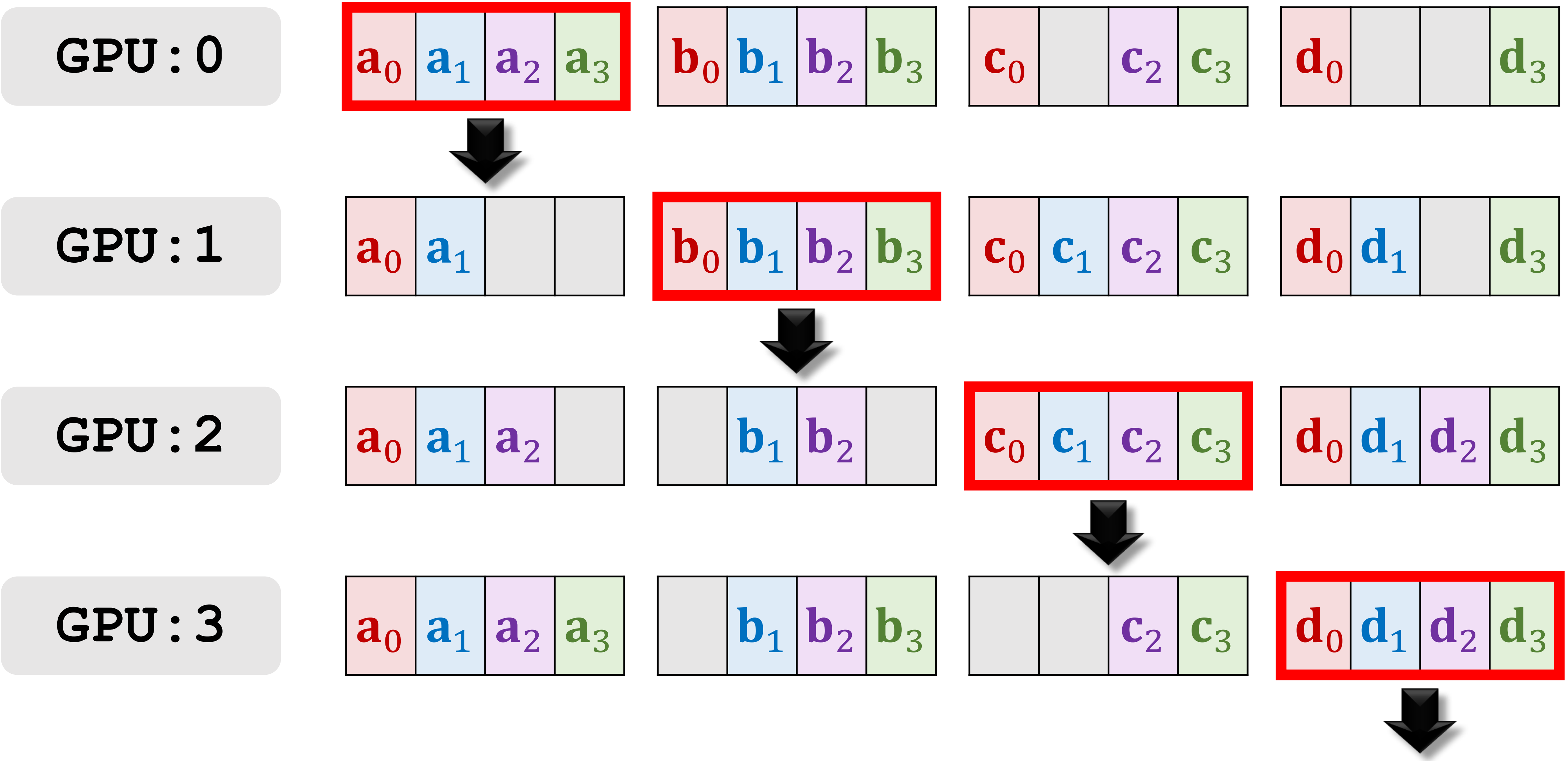
Ring All-Reduce (Efficient Approach)



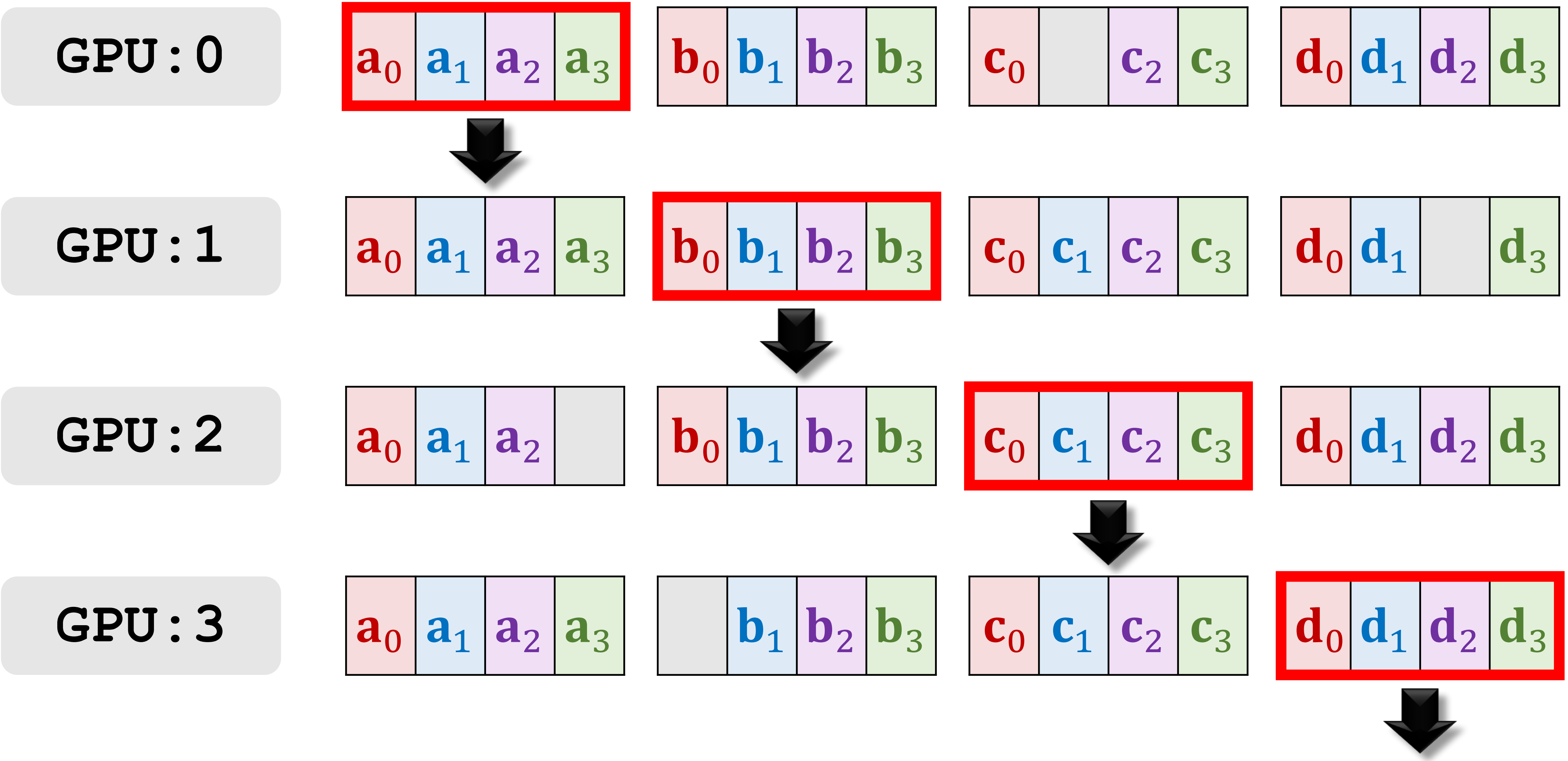
Ring All-Reduce (Efficient Approach)



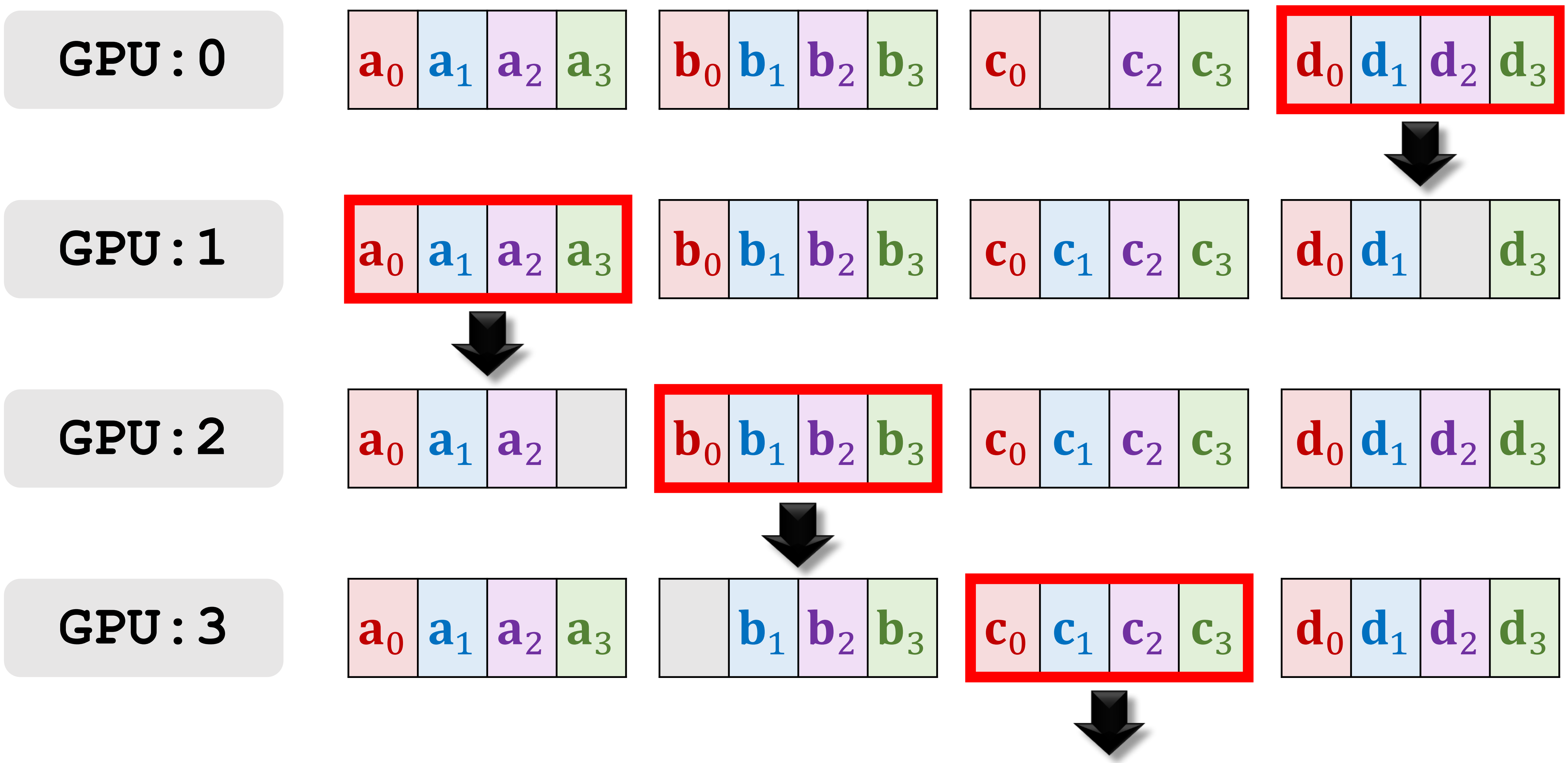
Ring All-Reduce (Efficient Approach)



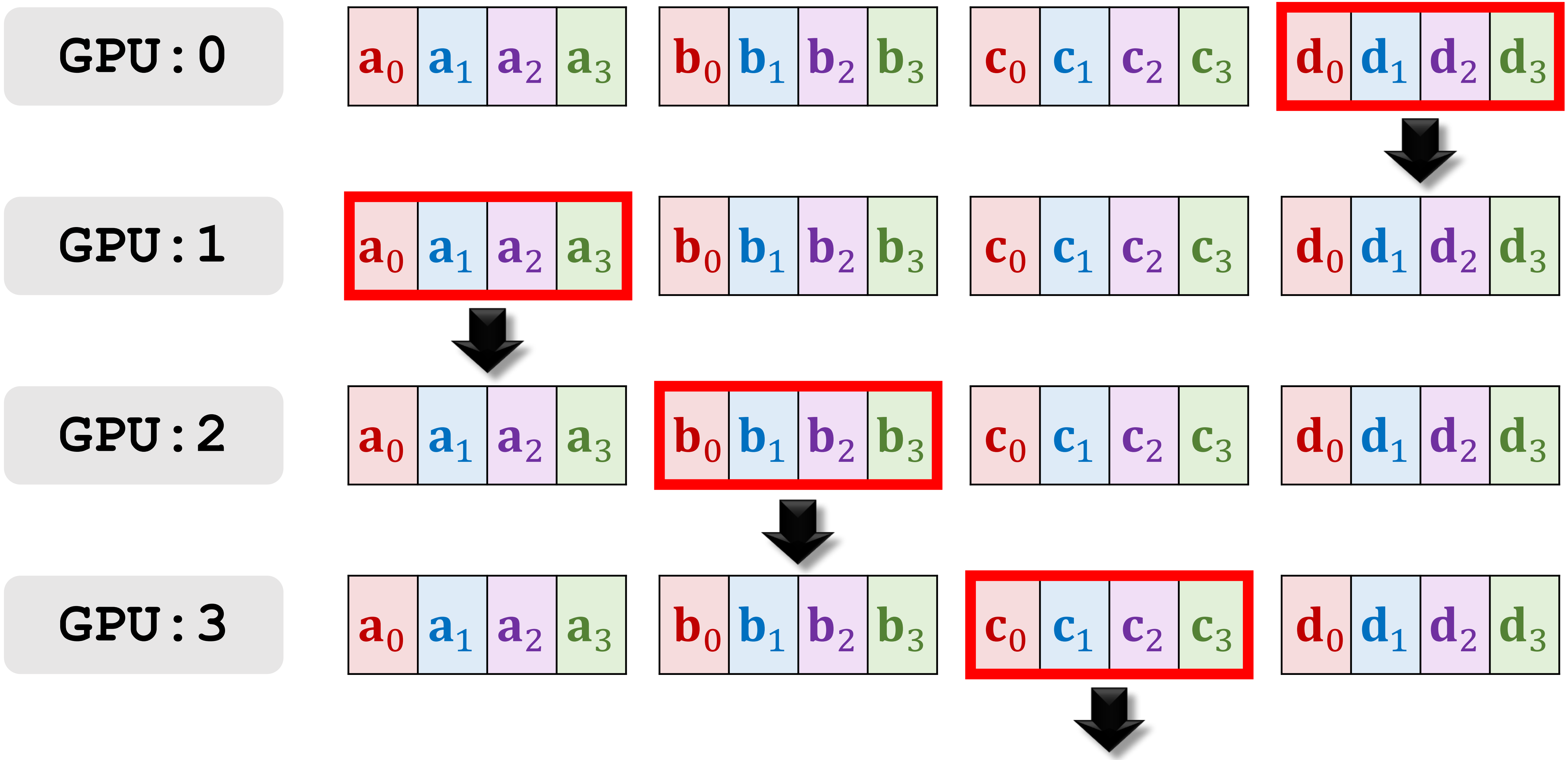
Ring All-Reduce (Efficient Approach)



Ring All-Reduce (Efficient Approach)



Ring All-Reduce (Efficient Approach)



Comparisons

Naïve Algorithm

- Most computer networks are idle.
- Communication time: $\frac{md}{b}$.
  - m: number of GPUs.
  - d: number of parameters.
  - b: network bandwidth.

Efficient Algorithm

- No idle computer network.
- Communication time: $\frac{d}{b}$.
  - d: number of parameters.
  - b: network bandwidth.

Thank you!