AI Applications Inter-Process Communications (MPI4Py and RCCL)

Presenter: Jose Noudohouenou

Oct 14, 2025

AMD Datacenter Solutions Group

AMD @ CASTIEL

AMD

together we advance_

Example of Running AI Applications on AMD GPUs

#	Phases	Steps	HW Resources	Communication Libraries (Reducing data preprocessing and processing times*)
1	Data preprocessing phase Data loaded in memory and transformed on CPU	Dataset discovering Sampler distribution Data loading	CPU	MPI (inter-CPU commications)
2	Data transfers between (mostly from) CPU and (to) GPU		CPU  GPU	
3	Data processing on GPU	Distributed DataParallel (Training)	GPU	RCCL (AMD inter-GPU communications)

* Full optimization requires additional tunings that are not listed on this slide
* Processes communicate with each other by sending messages via the interconnect network
* Communications are managed by libraries such as MPI (CPU), MPI4Py (CPU/GPU), RCCL (AMD GPUs)

MPI4Py

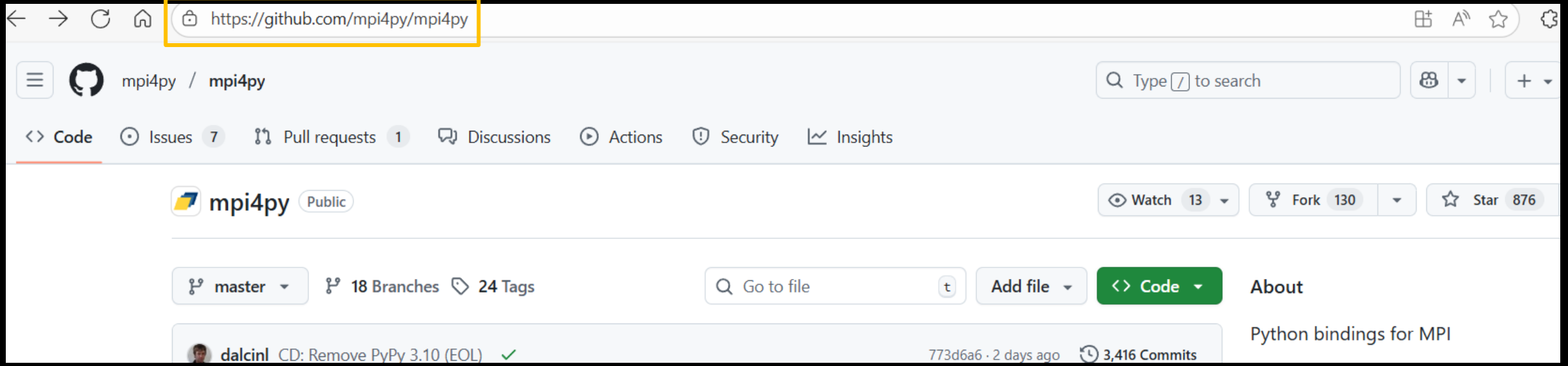
What is MPI4Py

- The Message Passing Interface (MPI) is a standardized and portable message-passing system designed to function on a wide variety of parallel computers
- The MPI standard defines the syntax and semantics of library routines and allows users to write portable programs in the main scientific programming languages (Fortran, C, or C++).
- MPI for Python™ provides (MPI4Py) MPI bindings for the Python™ programming language, allowing any Python program to exploit multiple processors across multiple nodes.
- MPI4Py can send data directly from one GPU to another GPU by using GPU-aware MPI.
- MPI4Py can be configured to use any MPI implementation

source: mpi4py documentation

Where to get MPI4Py

main project repo



There are simple ways to install mpi4py using pre-built wheels for instance, but we want to leverage a GPU-aware installation of MPI done with **this** script

Hence, we will be installing mpi4py from source using the above GitHub repo

MPI4Py Installation

- Script to install mpi4py from source: https://github.com/amd/HPCTrainingDock/blob/main/comm/scripts/mpi4py_setup.sh
- Installation script needs an existing MPI installation, specified in the environment variable `MPI_PATH`
- We are using the OpenMPI GPU-Aware MPI present in the system: `module show openmpi`
  - Notice that `MPI_PATH` is defined in the OpenMPI module

```
module load ${MPI_MODULE} rocm/${ROCM_VERSION}
git clone --branch 4.1.0 https://github.com/mpi4py/mpi4py.git
cd mpi4py
echo "[model]              = ${MPI_PATH}" >> mpi.cfg
echo "mpi_dir              = ${MPI_PATH}" >> mpi.cfg
echo "mpicc                = ${MPI_PATH}"/bin/mpicc >> mpi.cfg
echo "mpic++               = ${MPI_PATH}"/bin/mpic++ >> mpi.cfg
echo "library_dirs         = %(mpi_dir)s/lib" >> mpi.cfg
echo "include_dirs         = %(mpi_dir)s/include" >> mpi.cfg
CC=${ROCM_PATH}/bin/amdclang CXX=${ROCM_PATH}/bin/amdclang++ python3 setup.py build --mpi=model
CC=${ROCM_PATH}/bin/amdclang CXX=${ROCM_PATH}/bin/amdclang++ python3 setup.py bdist_wheel
pip3 install -v --target=${MPI4PY_PATH} dist/mpi4py-*.whl
```

MPI4Py vs OpenMPI API Comparison

MPI4Py

Allreduce(*sendbuf, recvbuf, op=SUM*)

Reduce to All.

Parameters:
- **sendbuf** (*BufSpec* | *InPlace*)
- **recvbuf** (*BufSpec*)
- **op** (*Op*)

Return type: None

Bcast(*buf, root=0*)

Broadcast data from one process to all other processes.

Parameters:
- **buf** (*BufSpec*)
- **root** (*int*)

Return type: None

Send(*buf, dest, tag=0*)

Blocking send.

Note

This function may block until the message is received. Whether `Send` blocks or not depends on several factors and is implementation dependent.

Parameters:
- **buf** (*BufSpec*)
- **dest** (*int*)
- **tag** (*int*)

Return type: None

OpenMPI

```
int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count,
                  MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
```

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype,
    int root, MPI_Comm comm)
```

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,
     int tag, MPI_Comm comm)
```

Notes on MPI4Py API

- Use methods with all-lowercase name for generic Python™ objects (example: `Comm.send`)
- Use methods with an upper-case letter for buffer-like objects (example: `Comm.Send`)
- Source: mpi4py tutorial

Note about GPU Aware MPI and MPI4Py

- If mpi4py is built against a GPU-aware MPI implementation, GPU arrays can be passed to upper-case methods as long as they have either the `__dlpack__` and `__dlpack_device__` methods or the `__cuda_array_interface__` attribute that are compliant with the respective standard specifications.
- Only C-contiguous or Fortran-contiguous GPU arrays are supported.
- GPU buffers must be fully ready before any MPI routines operate on them to avoid race conditions. This can be ensured by using the synchronization API of your array library (as we'll see in the next example). mpi4py does not have access to any GPU-specific functionality and thus cannot perform this operation automatically for users.

source: mpi4py tutorial

MPI4Py and CuPy example: Allreduce and Bcast

Find the example in our exercises repo:
https://github.com/amd/HPCTrainingExamples/blob/main/Python/mpi4py/mpi4py_cupy.py

```python
def mpi4py_cupy_test():

    comm = MPI.COMM_WORLD
    size = comm.Get_size()
    rank = comm.Get_rank()

    # Allreduce
    if rank == 0:
      print("Starting allreduce test...")
    sendbuf = cupy.arange(10, dtype='i')
    recvbuf = cupy.empty_like(sendbuf)
    # always make sure the GPU buffer is ready before any MPI operation
    cupy.cuda.get_current_stream().synchronize()
    comm.Allreduce(sendbuf, recvbuf)
    assert cupy.allclose(recvbuf, sendbuf*size)

    # Bcast
    if rank == 0:
      print("Starting bcast test...")
    if rank == 0:
        buf = cupy.arange(100, dtype=cupy.complex64)
    else:
        buf = cupy.empty(100, dtype=cupy.complex64)
    cupy.cuda.get_current_stream().synchronize()
    comm.Bcast(buf)
    assert cupy.allclose(buf, cupy.arange(100, dtype=cupy.complex64))
```

Returns an array with evenly spaced values within a given interval:
in this case it will be 10,11,…,19
Returns a new array with same shape and dtype of `sendbuf`.

Similar to the corresponding numpy calls but happening on the GPU

Note that the call `cupy.cuda.get_current_stream()` returns an object of type `cupy.cuda.Stream`, see the documentation for the full list of methods, including `synchronize()`

alias for `numpy.complex64` which is a `float complex` in C

Returns `True` if the two arrays are element-wise equal within a tolerance,
Using this formula:

$$|a - b| \leq a * tol + |b| * rtol$$

where `a` is `recvbuf`, `b` is `sendbuf*size`, and by default `tol=1.e-08` and `rtol=1.e-05`

MPI4Py and CuPy example: Send-Recv

Find the example in our exercises repository:
https://github.com/amd/HPCTrainingExamples/blob/main/Python/mpi4py/mpi4py_cupy.py

```
# Send-Recv
if rank == 0:
  print("Starting send-recv test...")

if rank == 0:
    buf = cupy.arange(20, dtype=cupy.float64)
    cupy.cuda.get_current_stream().synchronize()
    for j in range(1,size):
       comm.Send(buf, dest=j, tag=88+j)
else:
    buf = cupy.empty(20, dtype=cupy.float64)
    cupy.cuda.get_current_stream().synchronize()
    comm.Recv(buf, source=0, tag=88+rank)
    assert cupy.allclose(buf, cupy.arange(20, dtype=cupy.float64))

if rank == 0:
  print("Success")
```

Add:
`print("Rank is:", rank)`
to show that multiple processes are executing

Then run with:
```
module load mpi4py cupy
mpirun -n 4 python3 mpi4py_cupy.py
```
and see this output:

```
Rank is: 2
Rank is: 1
Rank is: 3
Rank is: 0
Starting allreduce test...
Starting bcast test...
Starting send-recv test...
Success
```

Verifying that MPI4Py and CuPy example runs on the GPU

Set the AMD_LOG_LEVEL

```
export AMD_LOG_LEVEL=3
```

Then run again

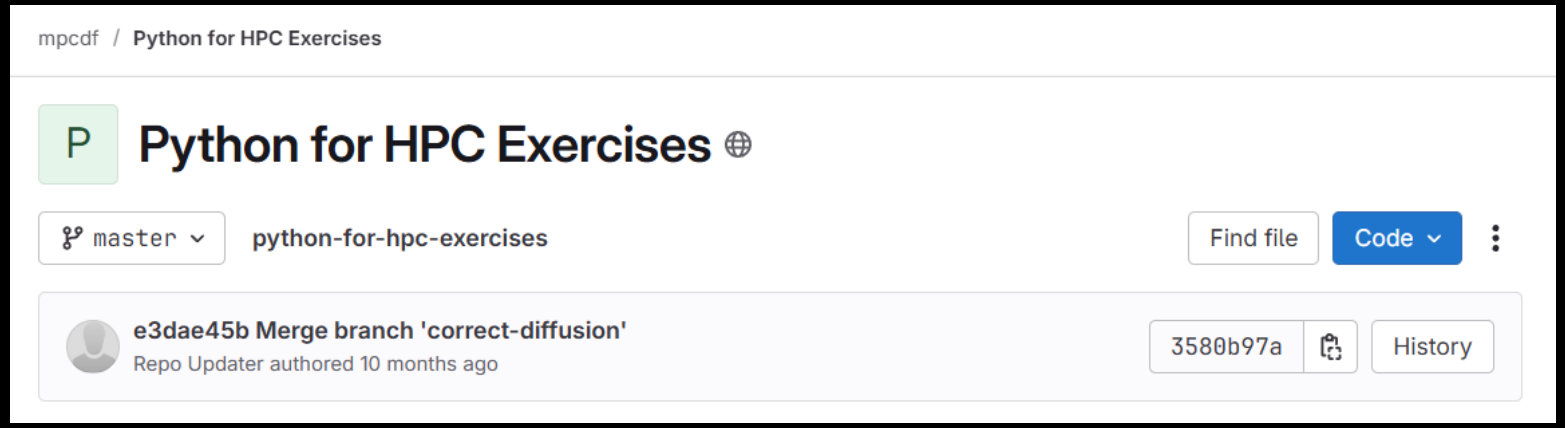
```
mpirun -n 4 python3 mpi4py_cupy.py
```

and see a lot more output including:

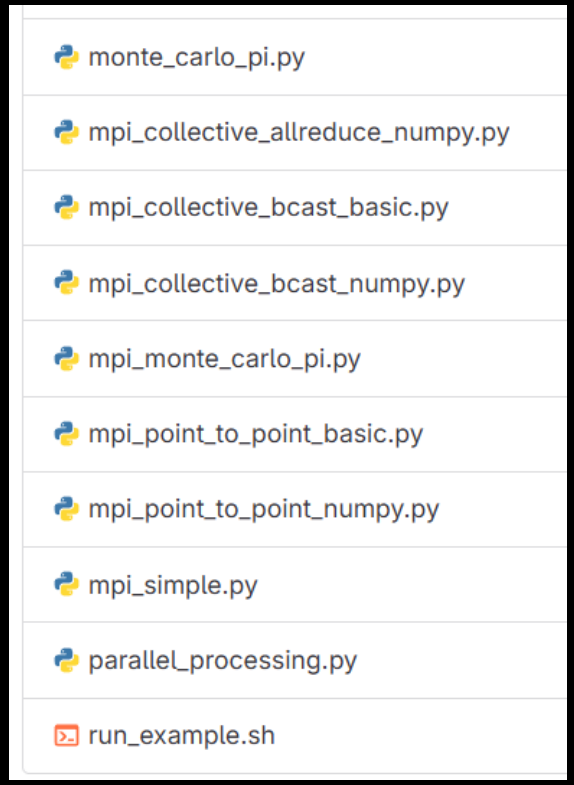
```
hiprtcCreateProgram ( 0x7fffa382ee28, #include <cupy/complex.cuh>
#include <cupy/carray.cuh>
#include <cupy/atomics.cuh>
#include <cupy/math_constants.h>
#include <cupy/hip_workaround.cuh>

typedef bool type_in0_raw;
typedef bool type_out0_raw;
typedef int IndexT;

#define REDUCE(a, b) (a & b)
#define POST_MAP(a) (out0 = a)
#define _REDUCE(_offset) if (_tid < _offset) {   _type_reduce _a = _sdata[_tid], _b = _sdata[(_tid + _offset)];   _sdata[_tid] =
REDUCE(_a, _b); }

typedef bool _type_reduce;
extern "C" __global__ void cupy_all(const CArray<bool, 1, 1, 1> _raw_in0, CArray<bool, 0, 1, 1> _raw_out0, CIndexer<1, 1> _in_ind,
CIndexer<0, 1> _out_ind, const int _block_stride) {
  __shared__ char _sdata_raw[256 * sizeof(_type_reduce)];
  _type_reduce *_sdata = reinterpret_cast<_type_reduce*>(_sdata_raw);
  unsigned int _tid = threadIdx.x;
```

Additional Resources

- MPI presentation (touching C,Fortran and Python™) from Rolf Rabenseifner at HLRS: https://fs.hlrs.de/projects/par/par_prog_ws/pdf/mpi_3.1_rab.pdf
- Python for HPC Exercises by MPCDF



mpi4py exercises →



RCCL

Introduction to RCCL

- RCCL: **R**OCm **C**ollective **C**ommunication **L**ibrary
  - Pronounced “rickle” (rhymes with nickel)
- Open-source host-initiated library enabling collective communications executed via GPU as well as direct send/receive operations
- Supports collective algorithms across multiple processes / nodes via networking using Infiniband Verbs or TCP/IP sockets
- Actively worked on to optimize performance for AMD hardware
- Forked from NCCL (NVIDIA Collective Communication Library)
  - Maintains identical API (drop-in replacement)

ROCm Collective Communication Library (RCCL)

- Library of standard communication routines for GPUs
- Implementing collectives e.g., all-reduce, all-gather, reduce, broadcast, reduce-scatter, gather, scatter, and all-to-all
- Initial support for direct GPU-to-GPU send and receive operations
- It is used as backend for collective communication for AI applications e.g., in PyTorch
- RCCL-test is a benchmark suite to evaluate the performance and correctness of RCCL operations
- Typical use cases:
  - AI workloads
  - No HPC Workloads
  - Distributed Training/Inferencing

RCCL Usage

- RCCL is provided as library and is not a standalone executable
  - Unit tests executables exist, but generally are not packaged and must be built from source
- Performance benchmarking can be done via `rccl-tests`
  - Fork of NCCL's nccl-tests
- RCCL is currently integrated in many machine learning frameworks, such as PyTorch, JAX, etc.
- RCCL is controlled primarily via environment variables

Example - RCCL test on MI300A

git clone https://github.com/ROCm/rccl-tests.git
cd rccl-tests/
make MPI=1 MPI_HOME=/opt/rocmplus-6.1.0/openmpi/ HIP_HOME=/opt/rocm/

#After successful build, you should be able to see the executables in ./build directory. You can run the collectives with:

./build/all_reduce_perf -b 4M -e 128M -f 2 -g 4

Run with 4 GPUs

multiplication factor between sizes

Run for 4M to 128M messages

```
sghazimi@d9dbb2d52d84:~/rccl/rccl-tests$ ./build/all_reduce_perf -b 4M -e 128M -f 2 -g 4
# nThread 1 nGpus 4 minBytes 4194304 maxBytes 134217728 step: 2(factor) warmup iters: 5 iters: 20 agg iters: 1 validation: 1 graph: 0
#
rccl-tests: Version develop:990f88c
# Using devices
#   Rank  0 Pid 1004519 on d9dbb2d52d84 device  0 [0000:01:00.0] AMD Instinct MI300A
#   Rank  1 Pid 1004519 on d9dbb2d52d84 device  1 [0001:01:00.0] AMD Instinct MI300A
#   Rank  2 Pid 1004519 on d9dbb2d52d84 device  2 [0002:01:00.0] AMD Instinct MI300A
#   Rank  3 Pid 1004519 on d9dbb2d52d84 device  3 [0003:01:00.0] AMD Instinct MI300A
#
#                                                     out-of-place                       in-place
#       size         count      type   redop    root     time   algbw   busbw #wrong     time   algbw   busbw #wrong
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)
     4194304       1048576     float     sum      -1    91.43   45.88   68.81      0    85.67   48.96   73.44      0
     8388608       2097152     float     sum      -1    128.0   65.53   98.30      0    135.4   61.97   92.95      0
    16777216       4194304     float     sum      -1    226.2   74.16  111.25      0    246.0   68.19  102.29      0
    33554432       8388608     float     sum      -1    409.7   81.91  122.86      0    441.4   76.01  114.01      0
    67108864      16777216     float     sum      -1    789.2   85.04  127.56      0    819.6   81.88  122.83      0
   134217728      33554432     float     sum      -1   1567.0   85.65  128.48      0   1612.5   83.24  124.85      0
```

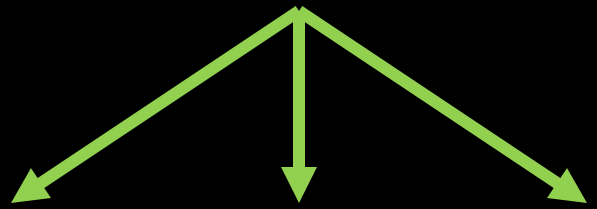

These tests have been executed as a demo (not performance claim) on AAC nodes with MI300A APUs , rocm 6.1.0 and OpenMPI 5.0.3/UCX 1.16.0 on April/22/2024

Visualization of RCCL Collectives

Collectives Supported by RCCL

Point to Point Collectives

Send/Recv



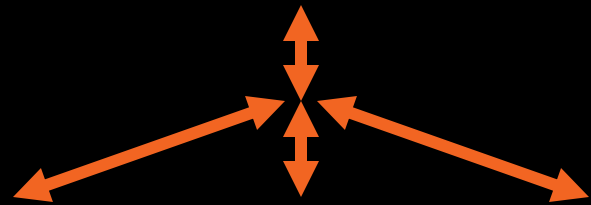
One to Many Collectives

Broadcast

Gather

Reduce

Scatter



Many to Many Collectives

AllGather

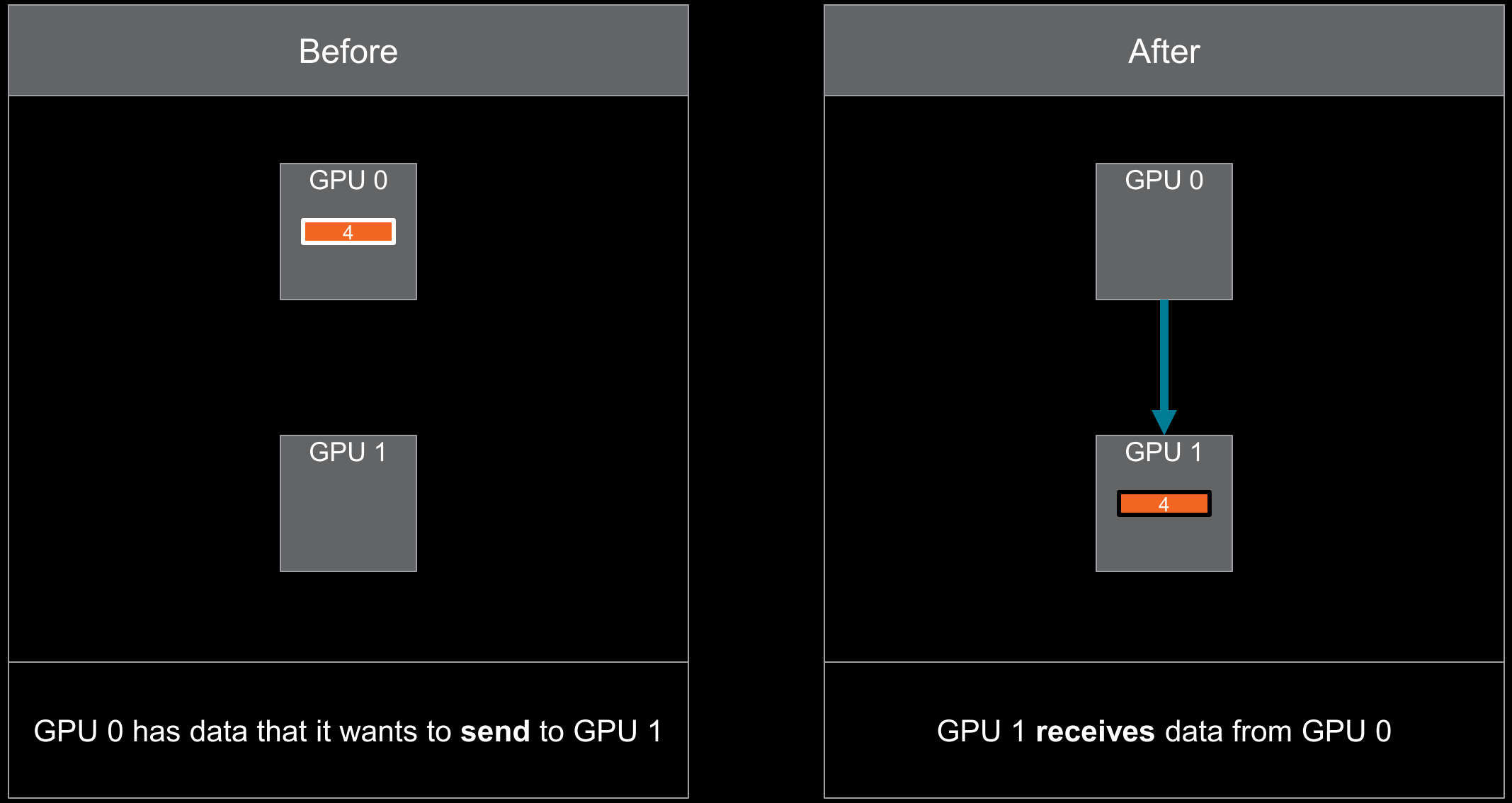
AllReduce

AllToAll

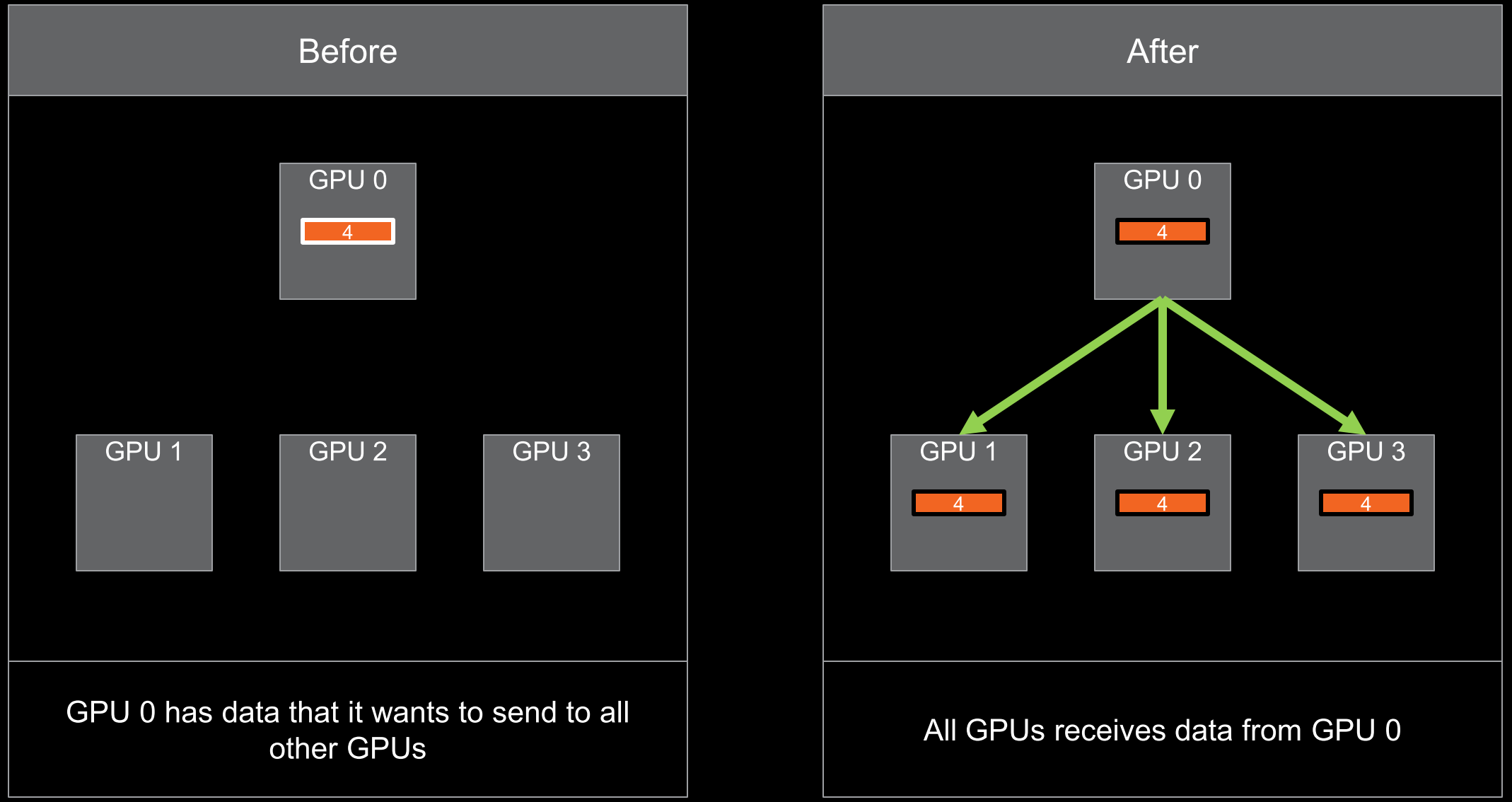
AllToAllV

ReduceScatter

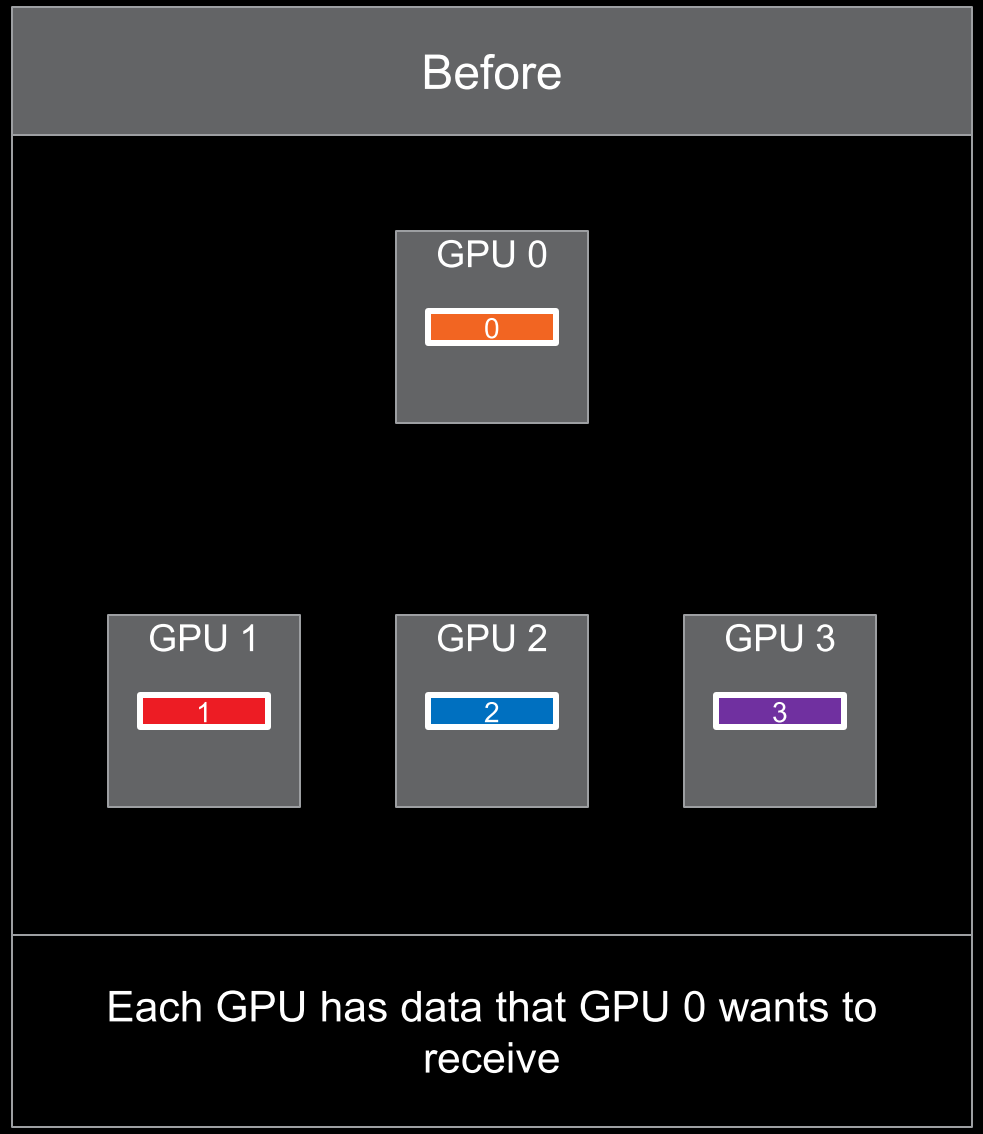
SEND/RECV

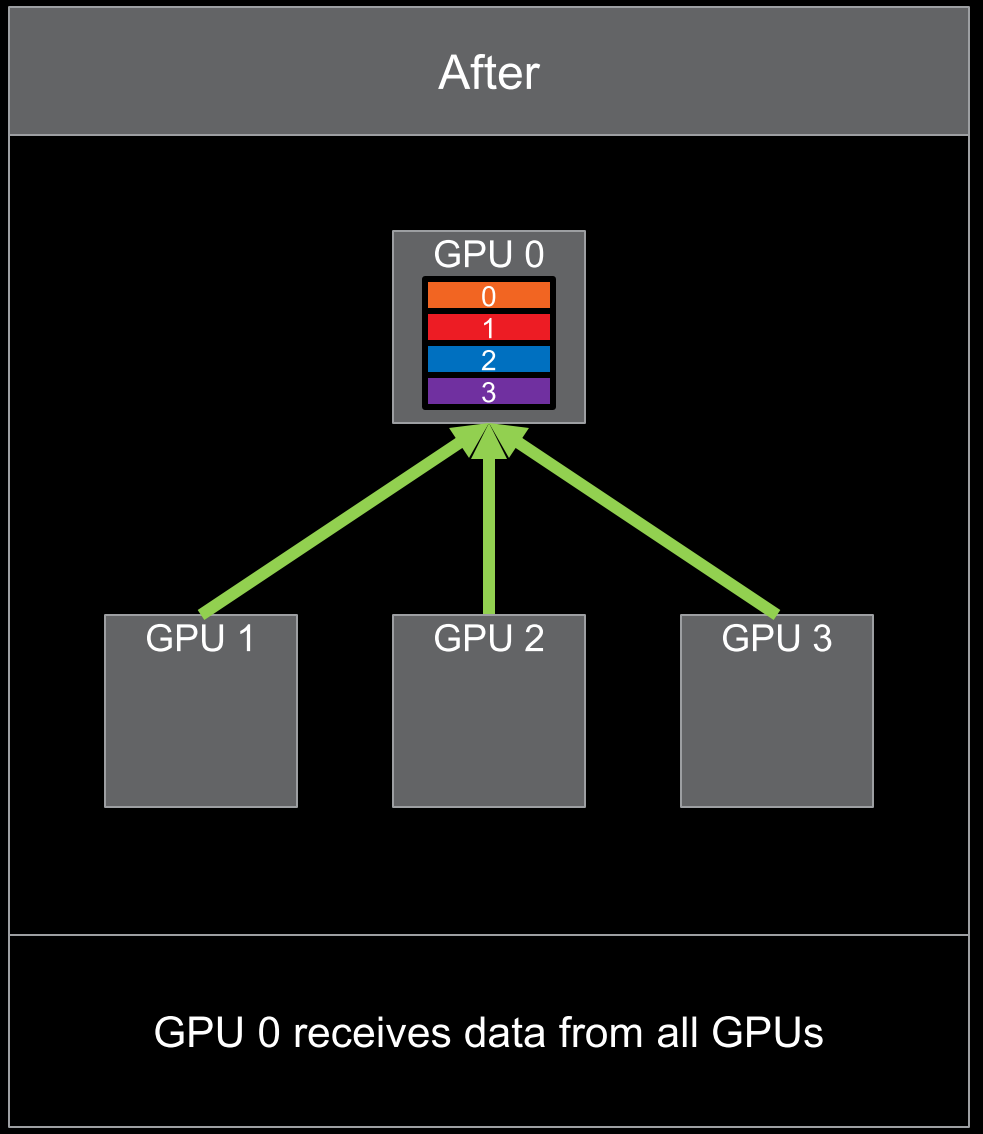


BROADCAST

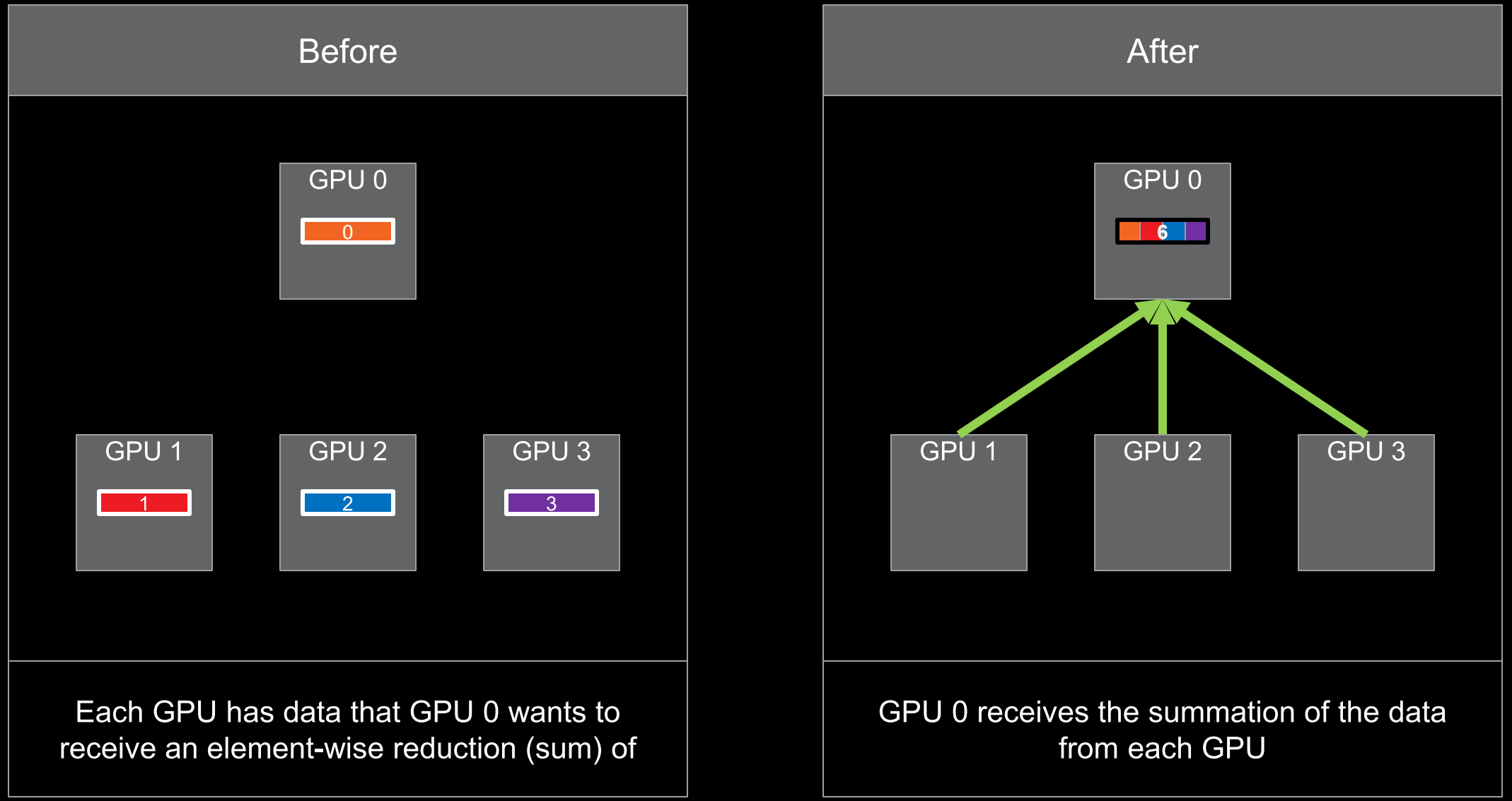


GATHER

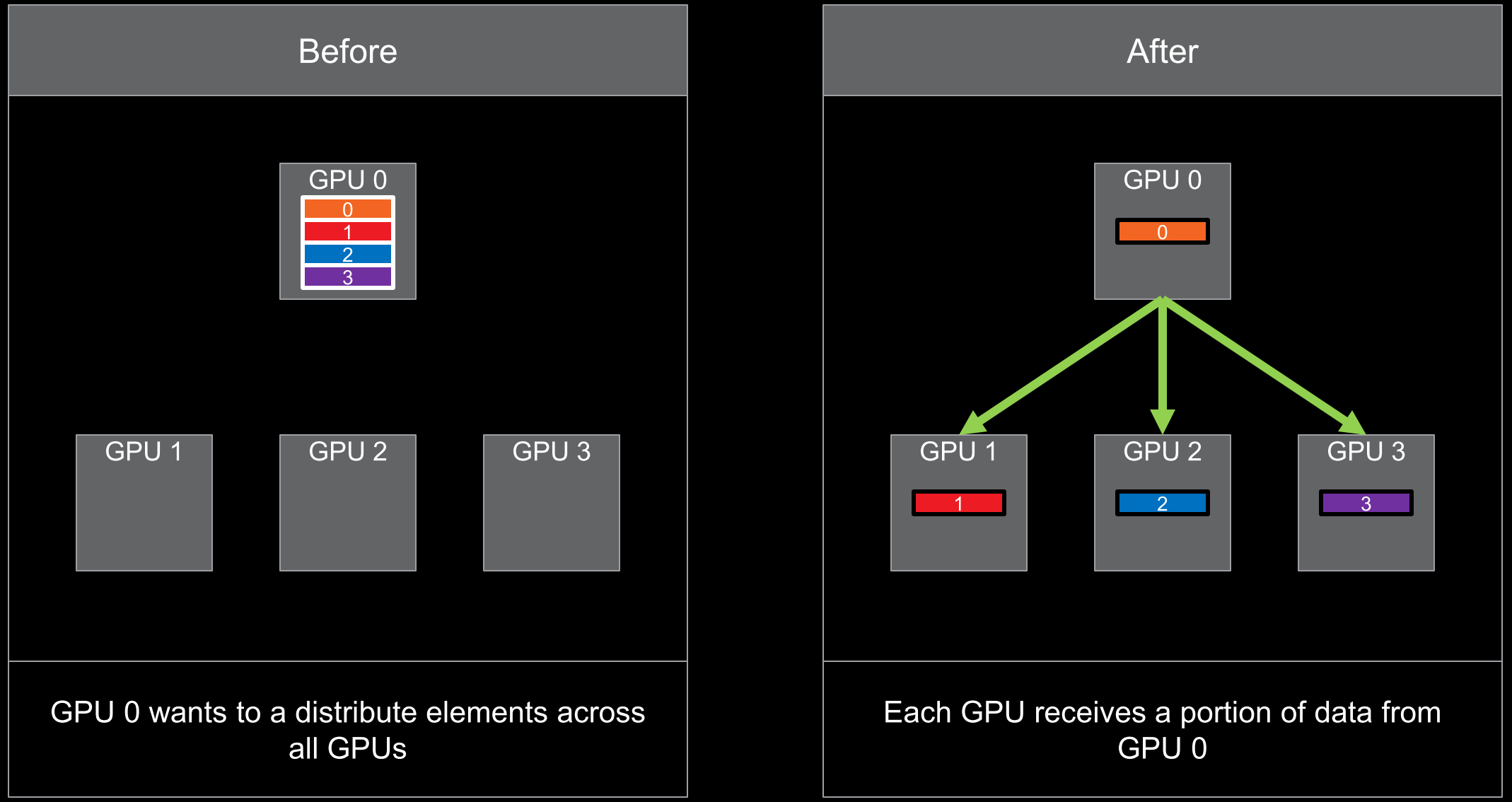




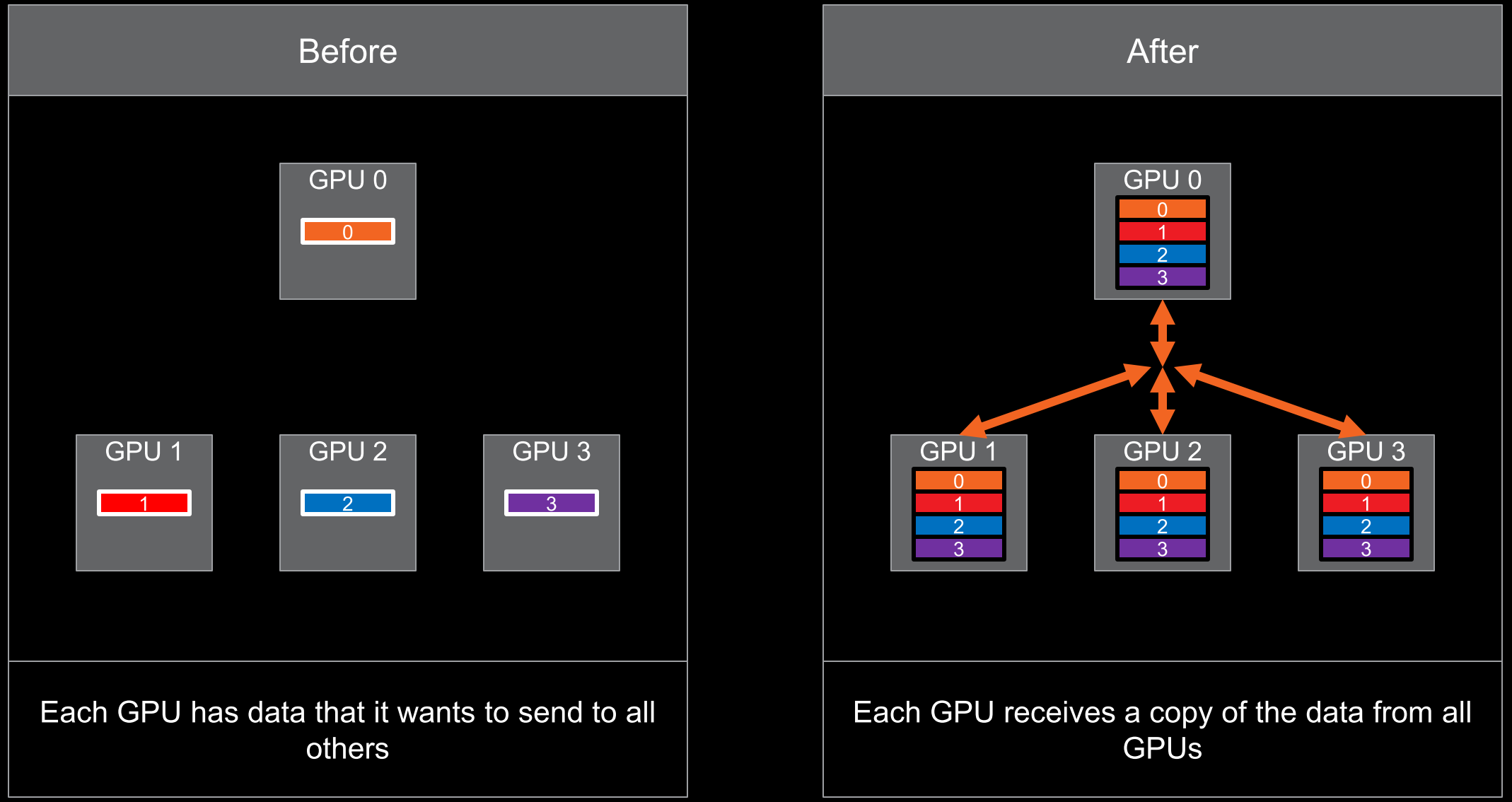
REDUCE



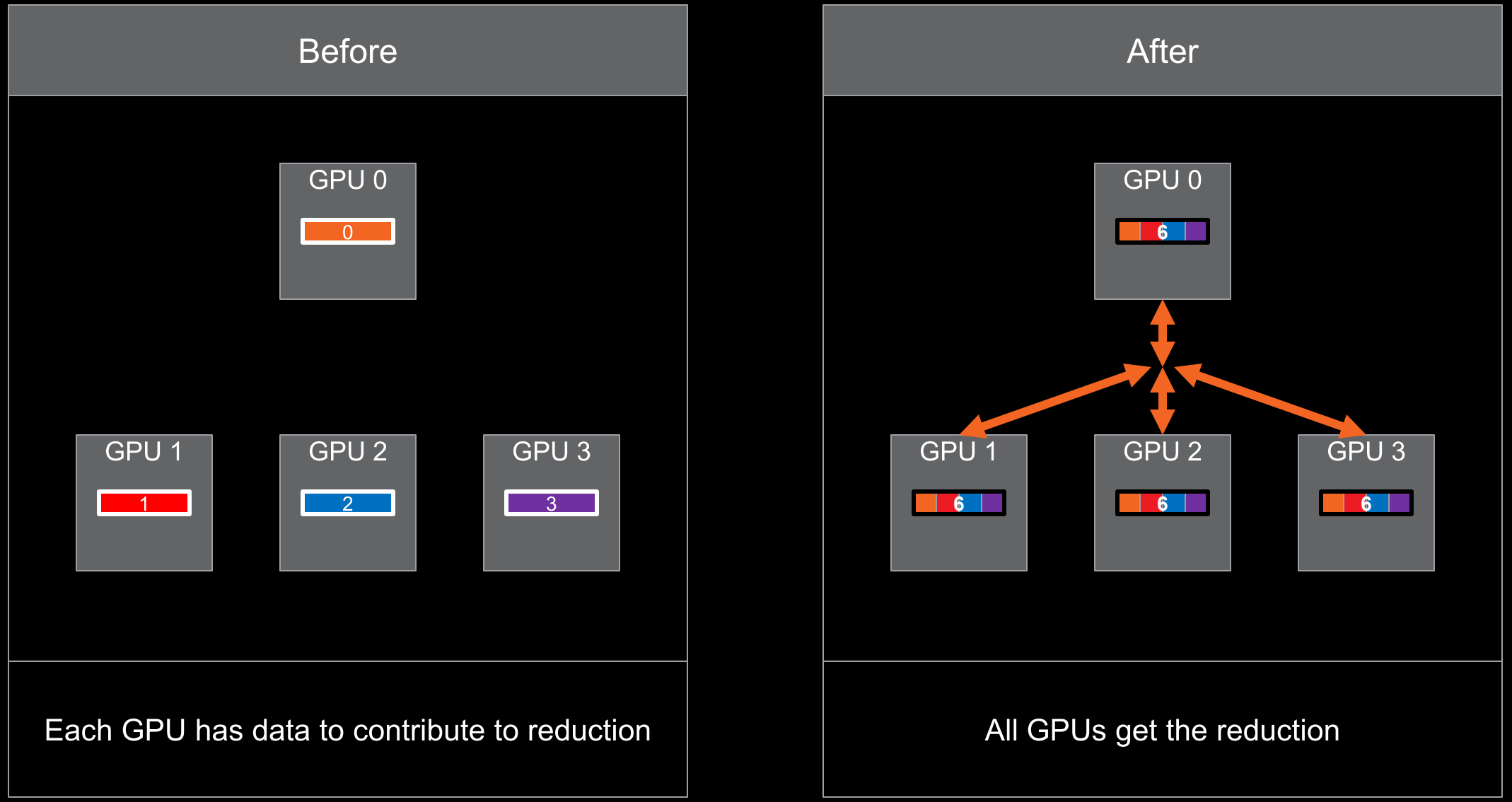
SCATTER



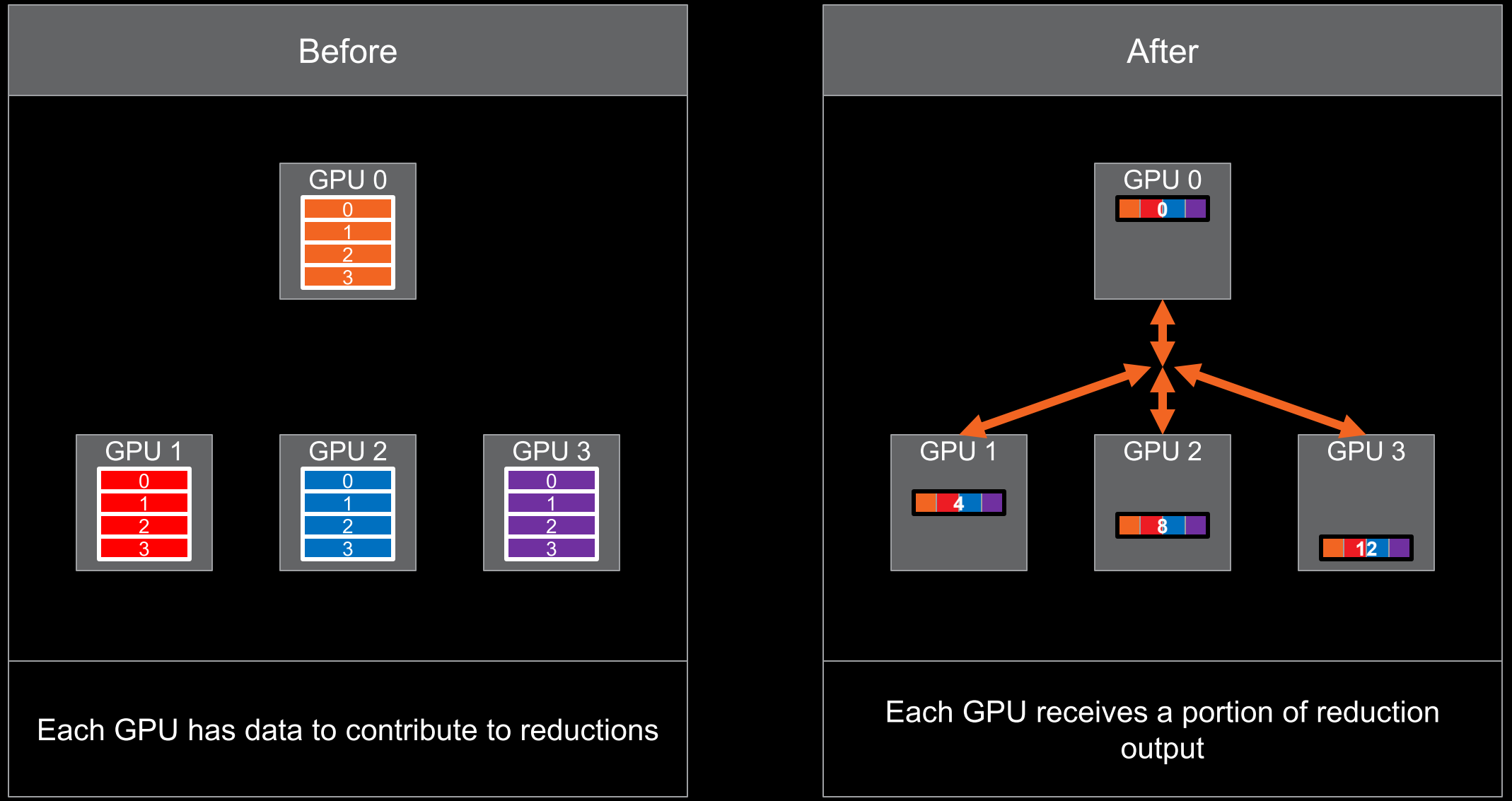
ALLGATHER



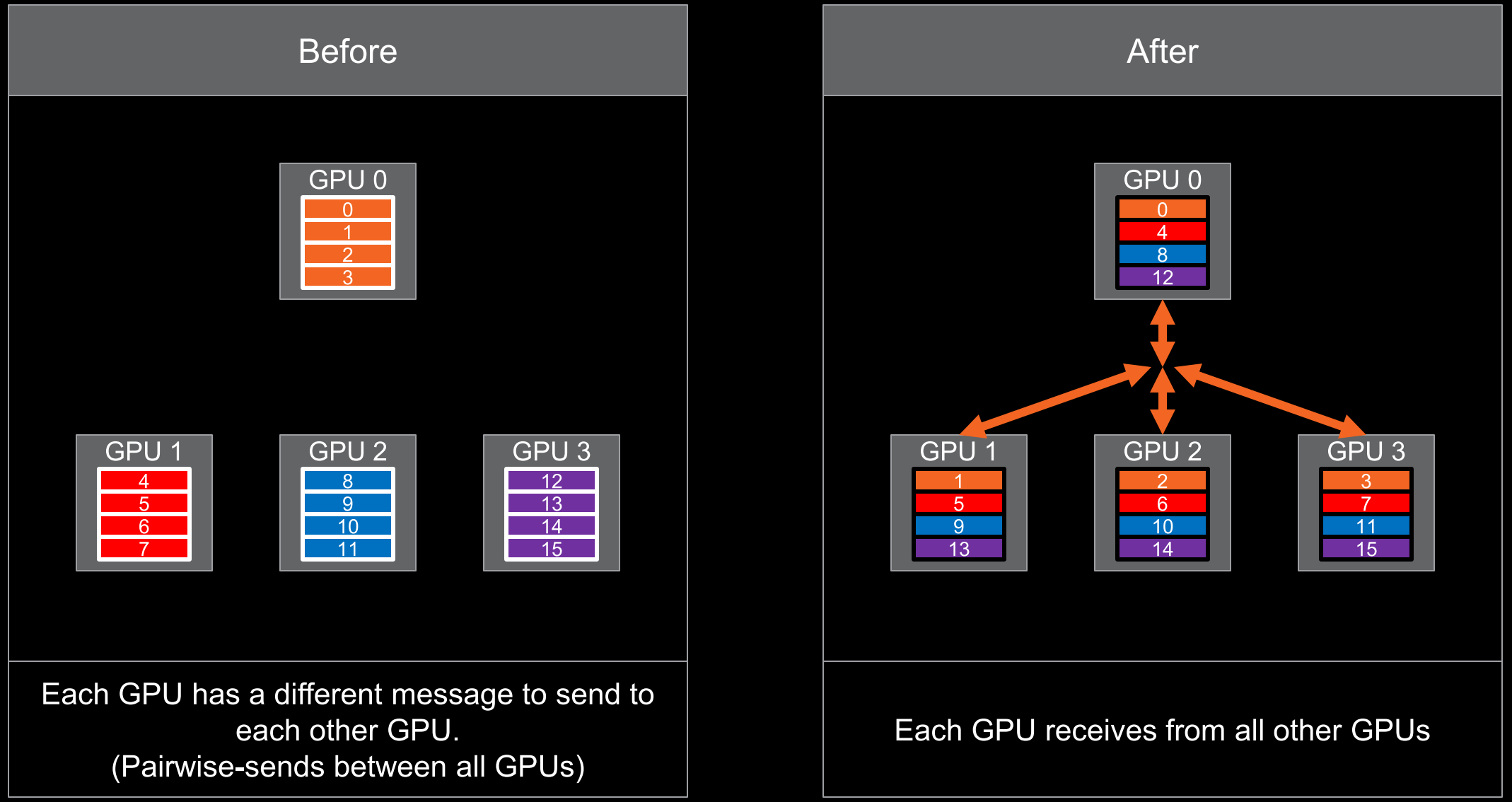
ALLREDUCE



REDUCE SCATTER



ALL TO ALL / ALL TO ALL V



With AllToAll each message is the same size.

With AllToAllV, each message size can be **V**ariable

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED ‘AS IS.” AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED “AS IS” WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.

Git and the Git logo are either registered trademarks or trademarks of Software Freedom Conservancy, Inc., corporate home of the Git Project, in the United States and/or other countries


AMD