https://synergy.ece.gatech.edu

ASTRA-sim Tutorial
@HotI 2024
Aug 23, 2024

ASTRA-sim and Chakra Tutorial: *System Layer*

Will Won
Ph.D. Student
School CS, Georgia Institute of Technology
william.won@gatech.edu

Slide courtesy: Saeed Rashidi
<rashidi1saeid@gmail.com>

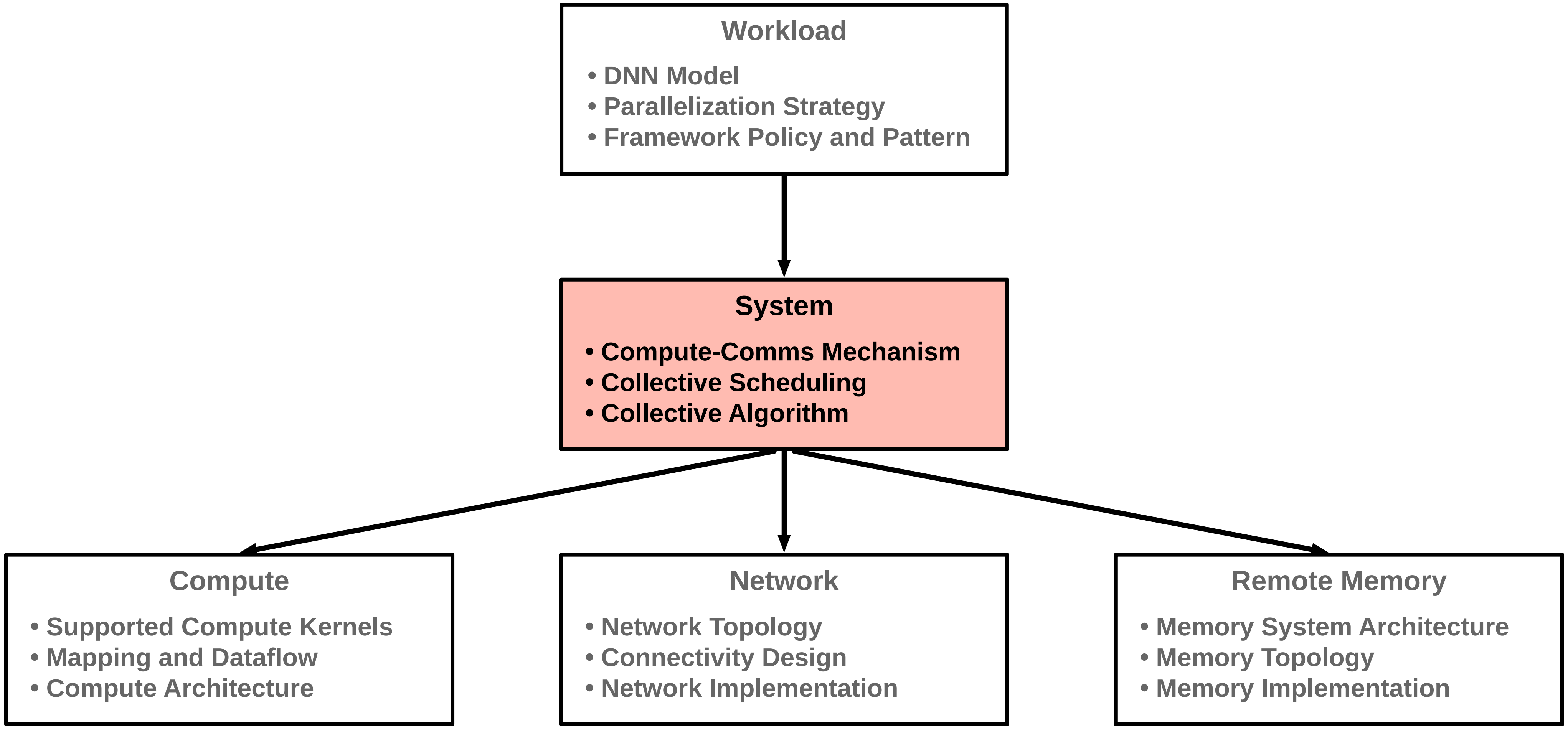
ASTRA-sim Tutorial - Agenda

Time (PDT)	Topic	Presenter
3:00 – 3:30 pm	**Introduction to Distributed ML**	Tushar Krishna
3:30 – 3:45 pm	**Overview of Chakra and ASTRA-sim**	Tushar Krishna
3:45 – 4:35 pm	**Deeper Dive into Chakra and ASTRA-sim**	Will Won
	Workload, System, and Network Layers	
4:35 – 4:45 pm	**Demo**	Will Won
4:45 – 5:00 pm	**Closing Remarks**	Tushar Krishna

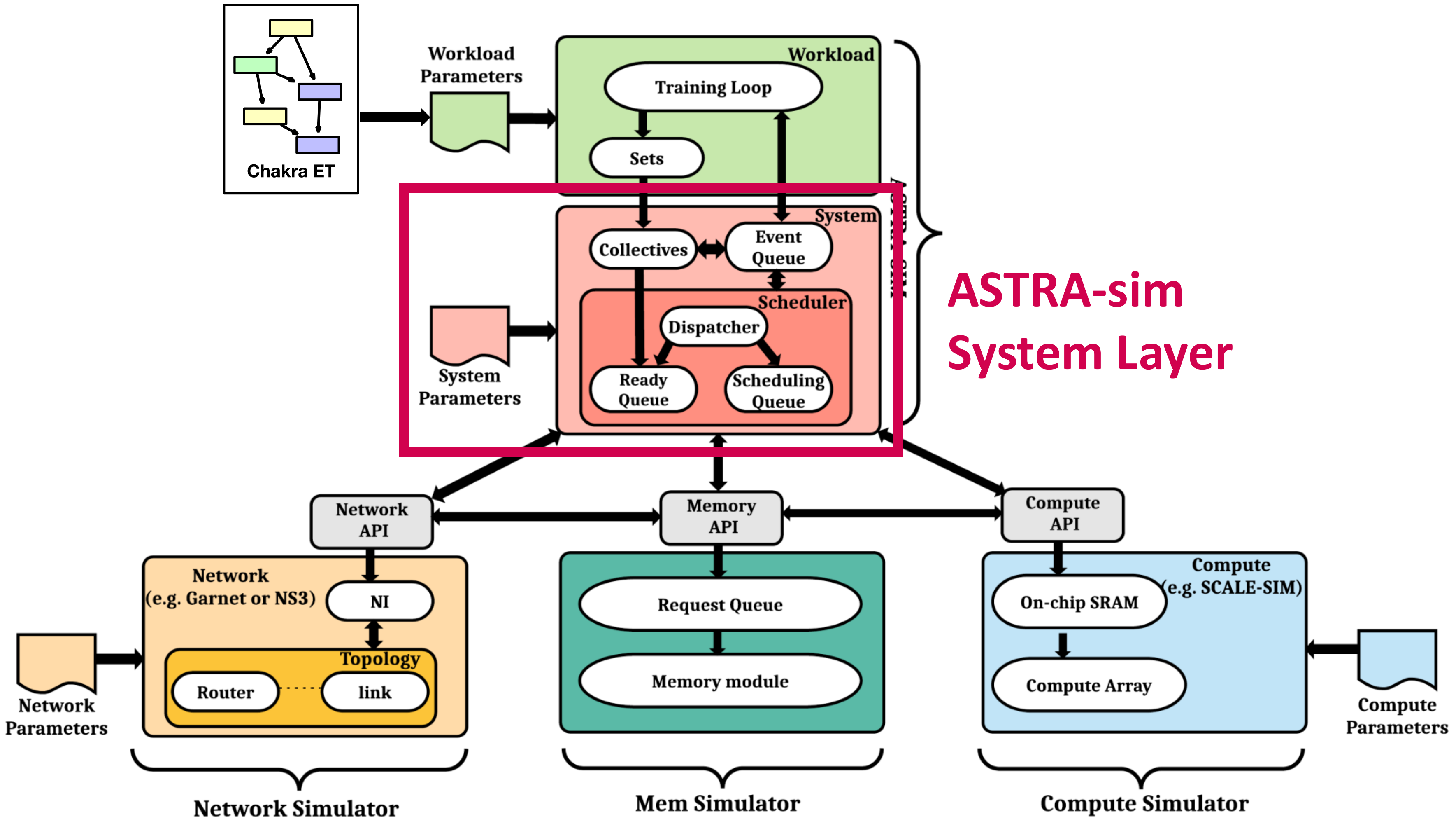
Tutorial Website
includes agenda, slides, ASTRA-sim installation instructions (via source + docker image)
https://astra-sim.github.io/tutorials/hoti-2024

Attention: Tutorial is being recorded

Design Space: System



ASTRA-sim: System Layer



Recall: Workload Layer

```
void Workload::issue_comm(node) {
        hw_resource->occupy(node);

        if (node->comm_type() == ChakraCollectiveCommType::ALL_REDUCE) {
                DataSet* fp = sys->generate_all_reduce(node->comm_size(), ...)

                fp->set_notifier(EventType::CollectiveCommunicationFinished);

        }

        (...)
}
```

Managed by the System Layer

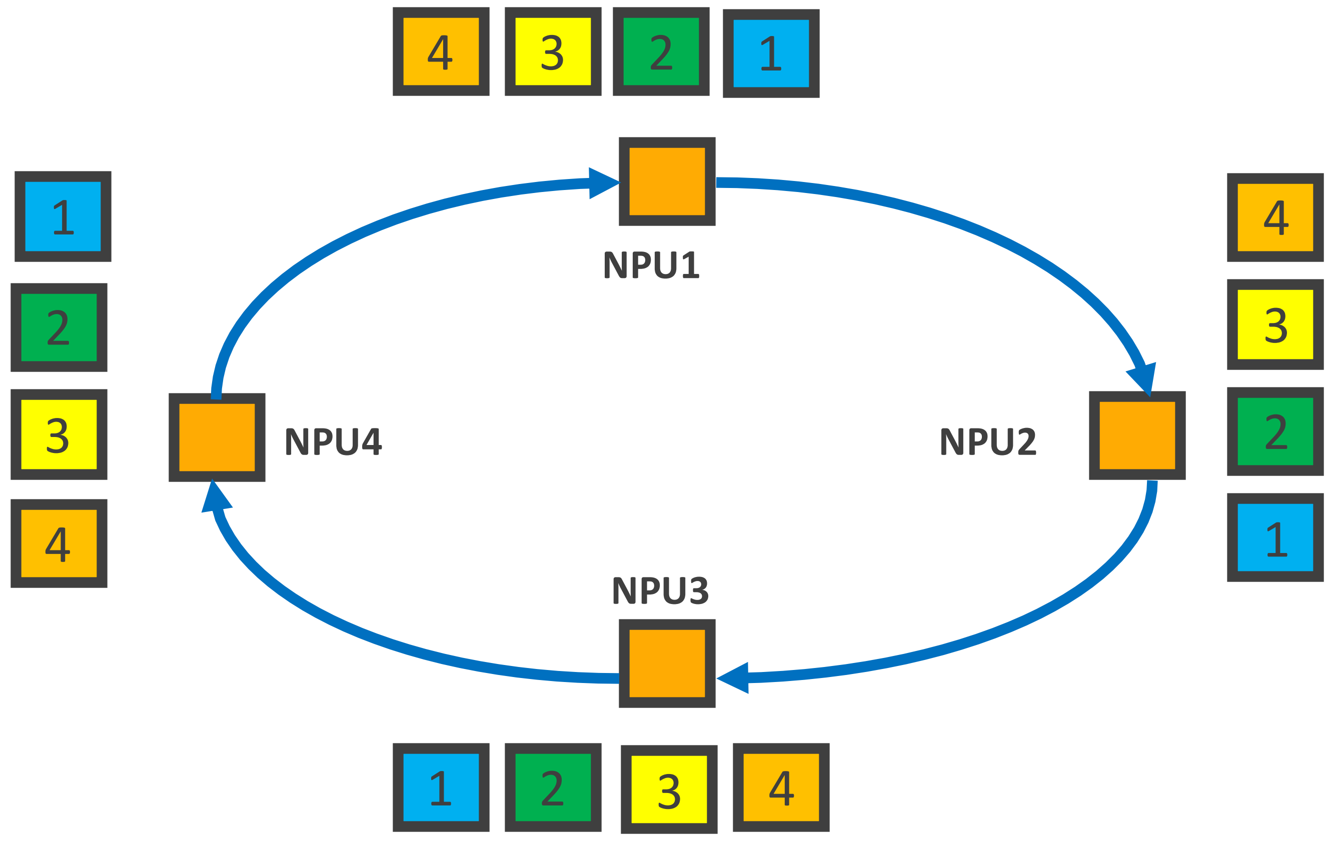
System Layer

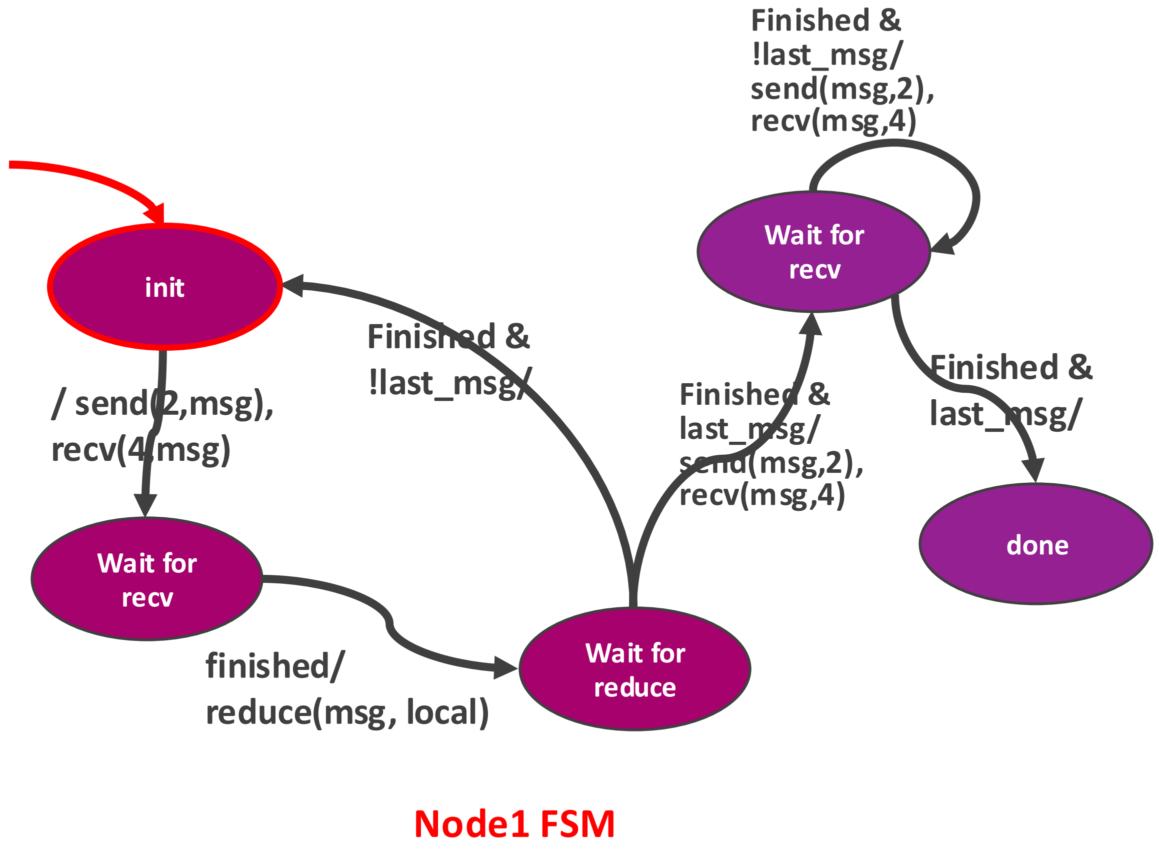
- **Collective algorithms** are implemented
  - Collectives are broken down into individual **chunk send-recv**

- Collective algorithms are managed via:
  - **Finite State Machine**

System Layer Collective Implementation

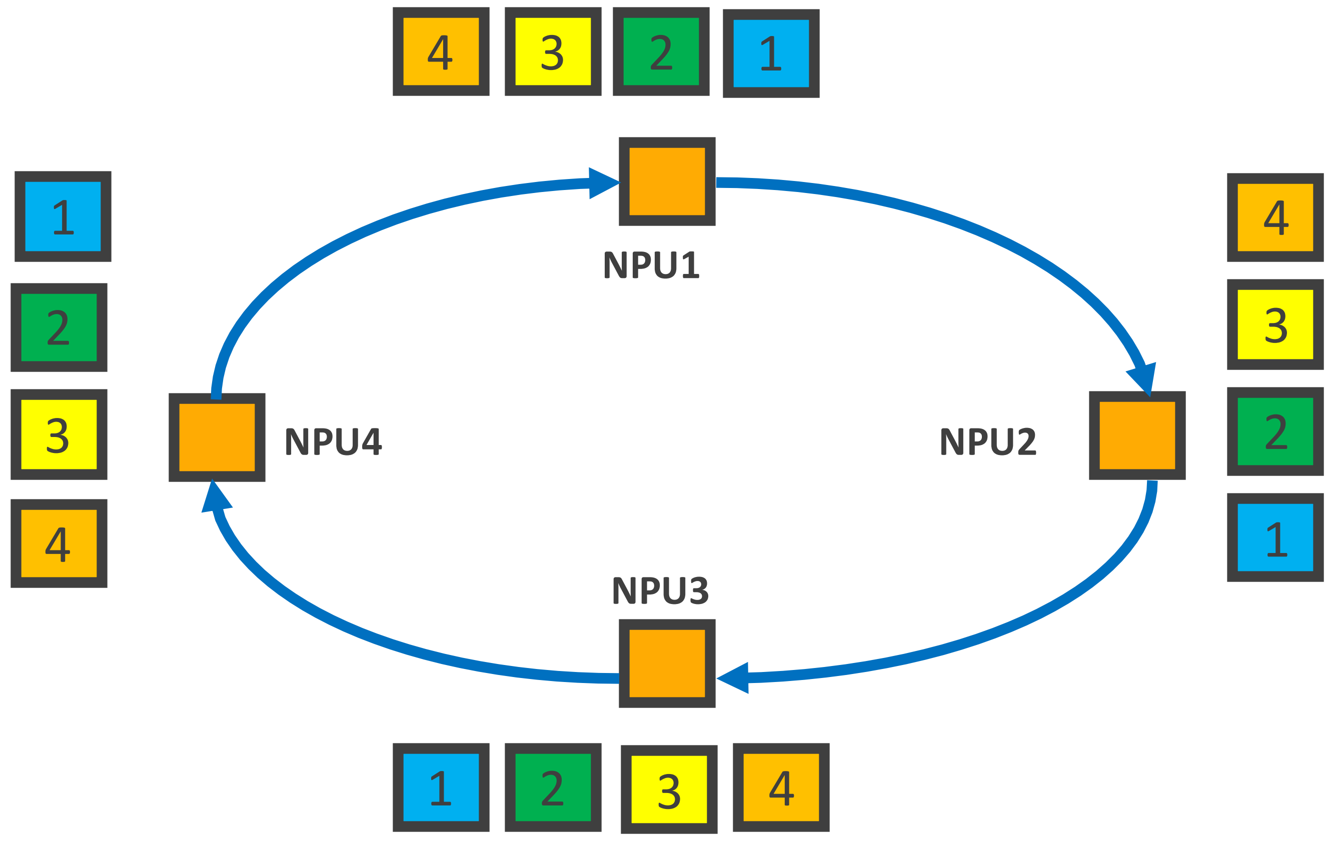
- Collective algorithms can be implemented using **state machines**.

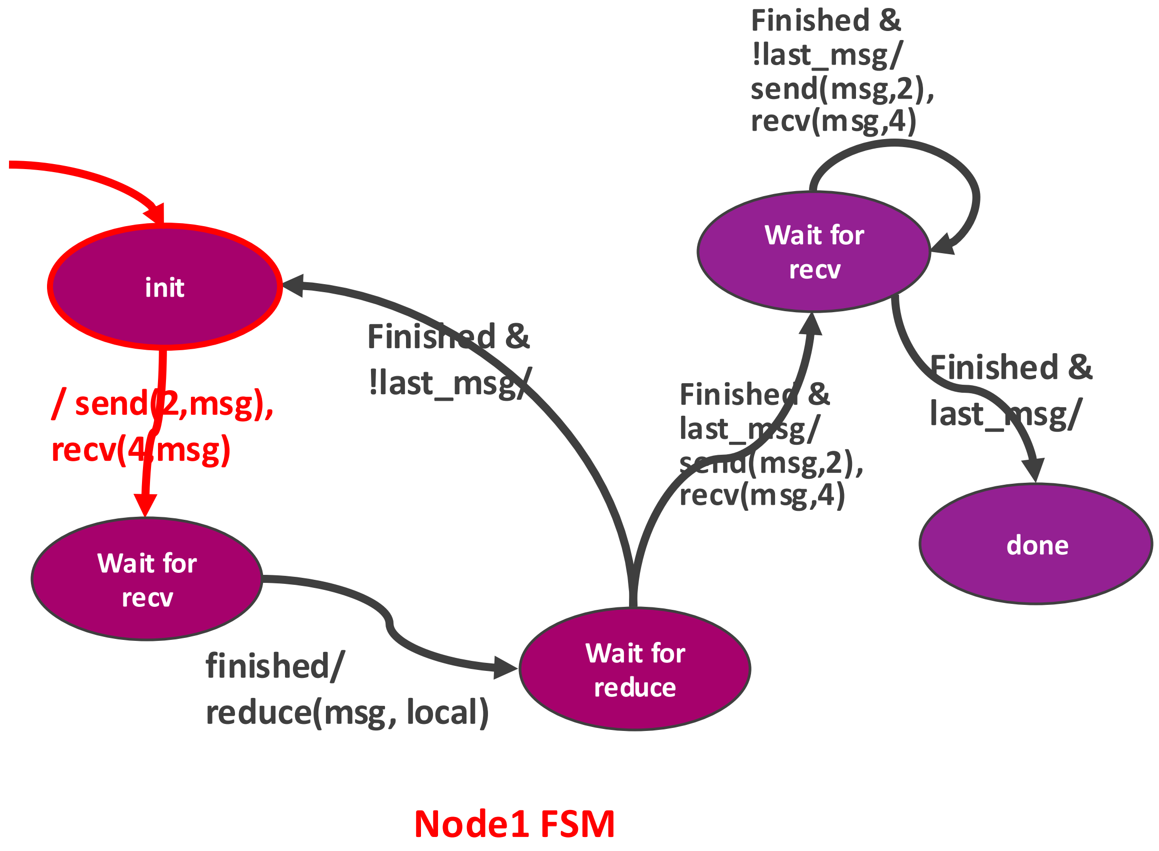



init
/ send(2,msg), recv(4,msg)
Wait for recv
finished/ reduce(msg, local)
Wait for reduce
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Wait for recv
Finished & !last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done
Node1 FSM

System Layer Collective Implementation

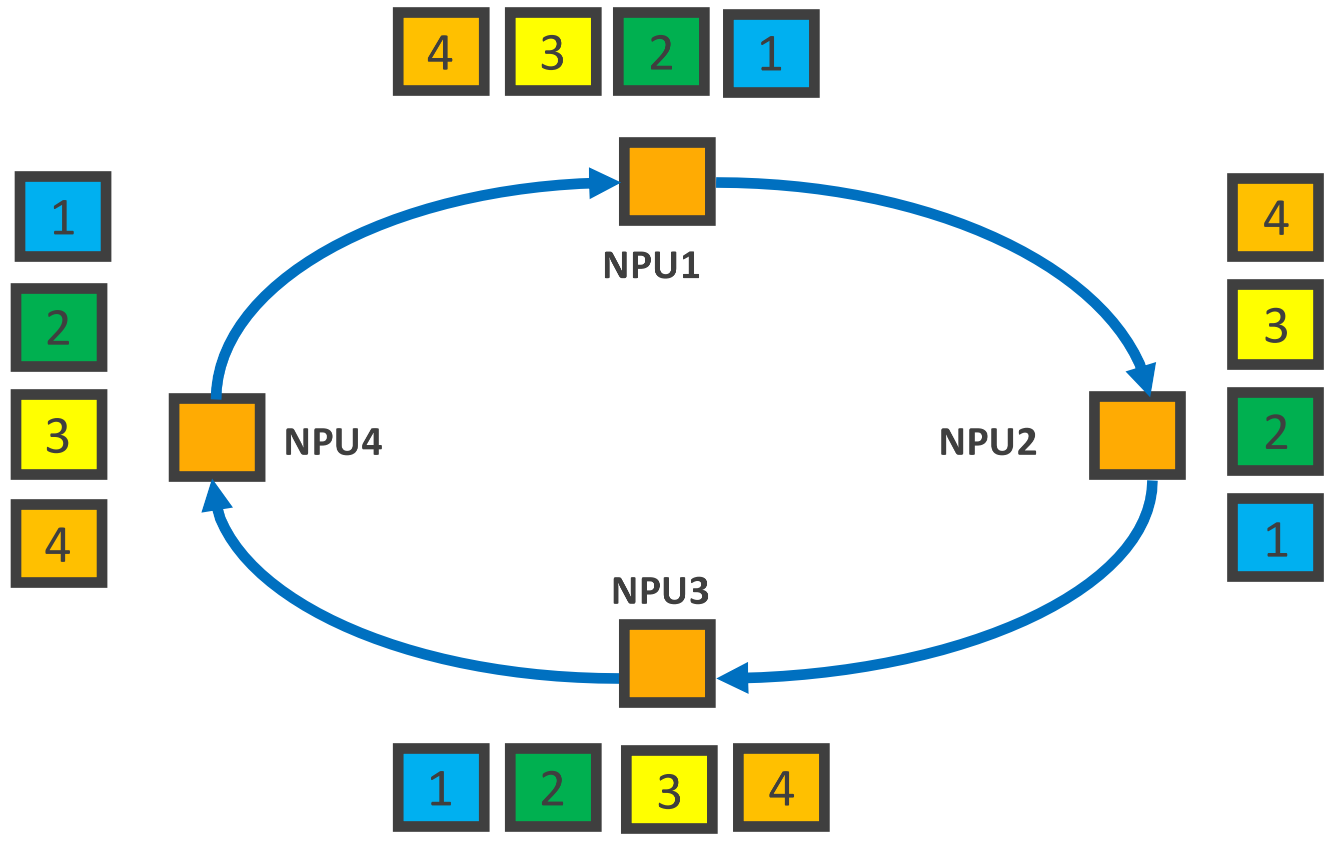
- Collective algorithms can be implemented using **state machines**.

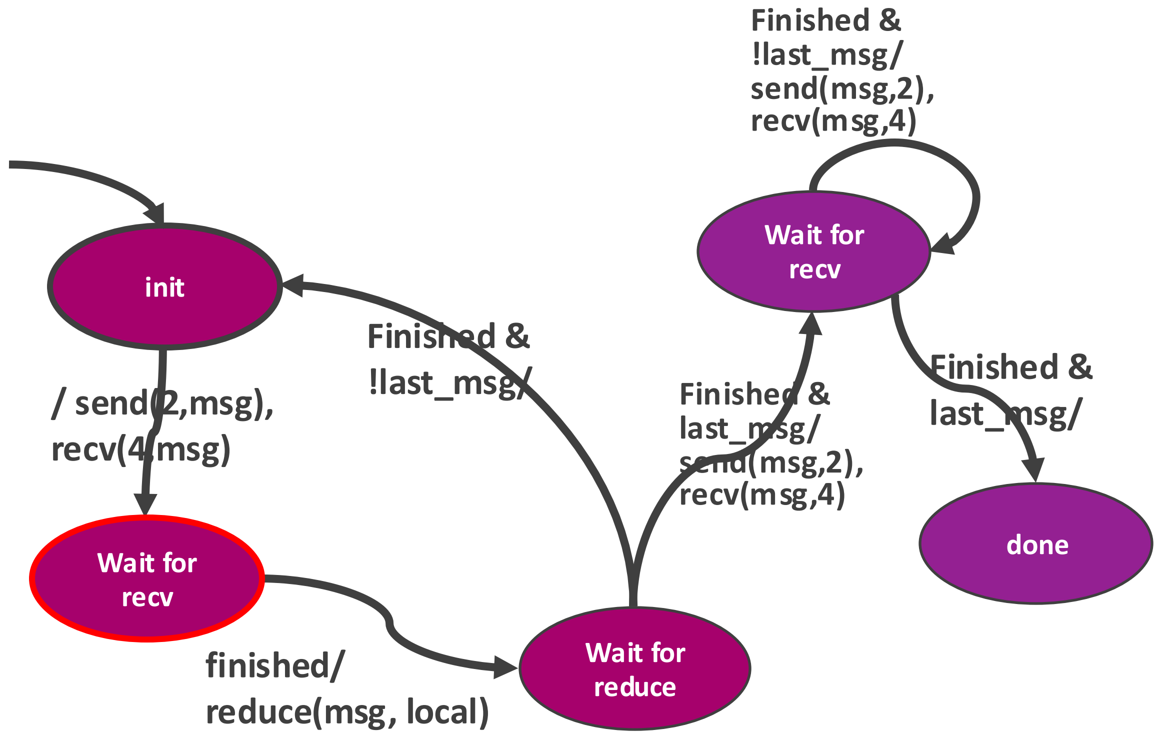



init
/ send(2,msg), recv(4,msg)
Wait for recv
finished/ reduce(msg, local)
Wait for reduce
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Wait for recv
Finished & !last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done
Node1 FSM

System Layer Collective Implementation

- Collective algorithms can be implemented using **state machines**.

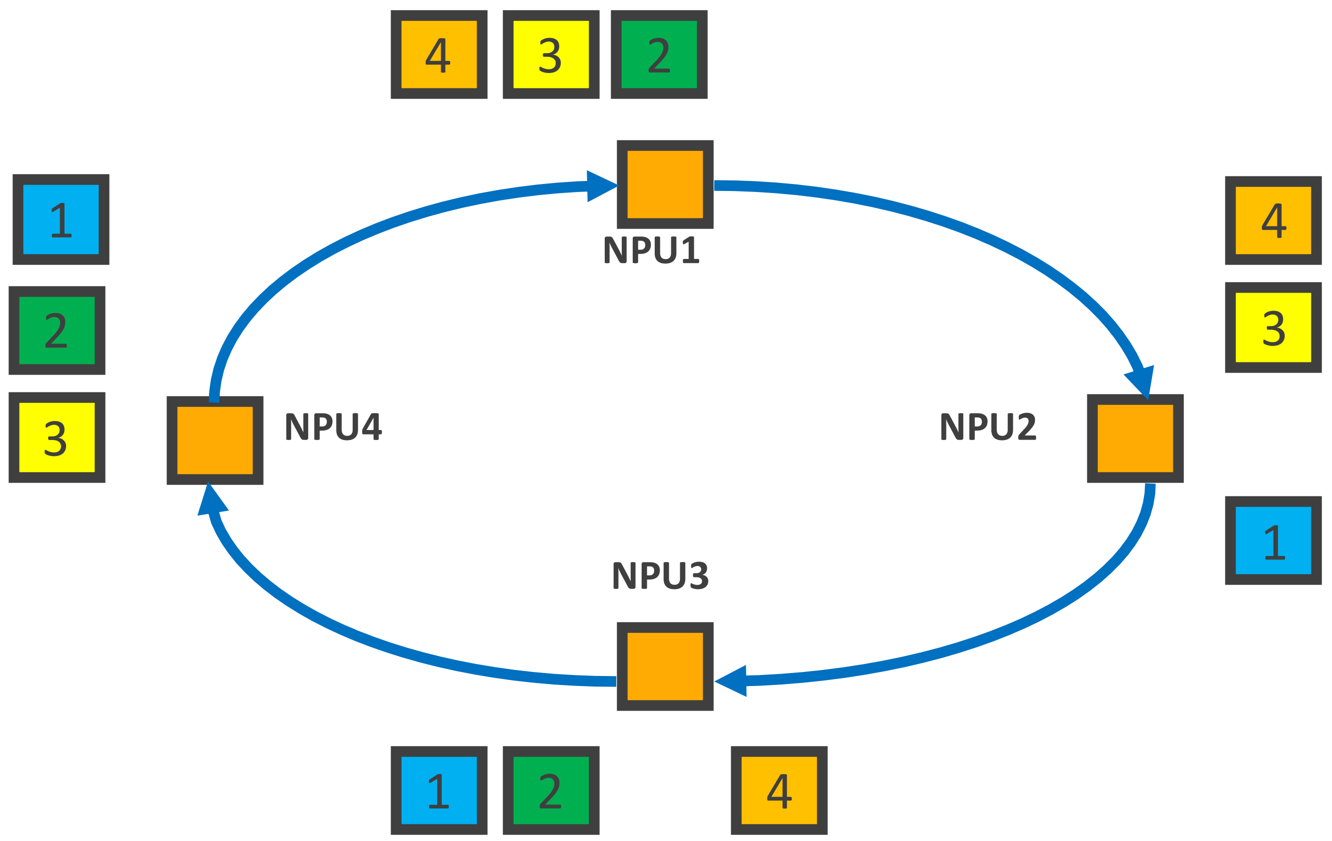


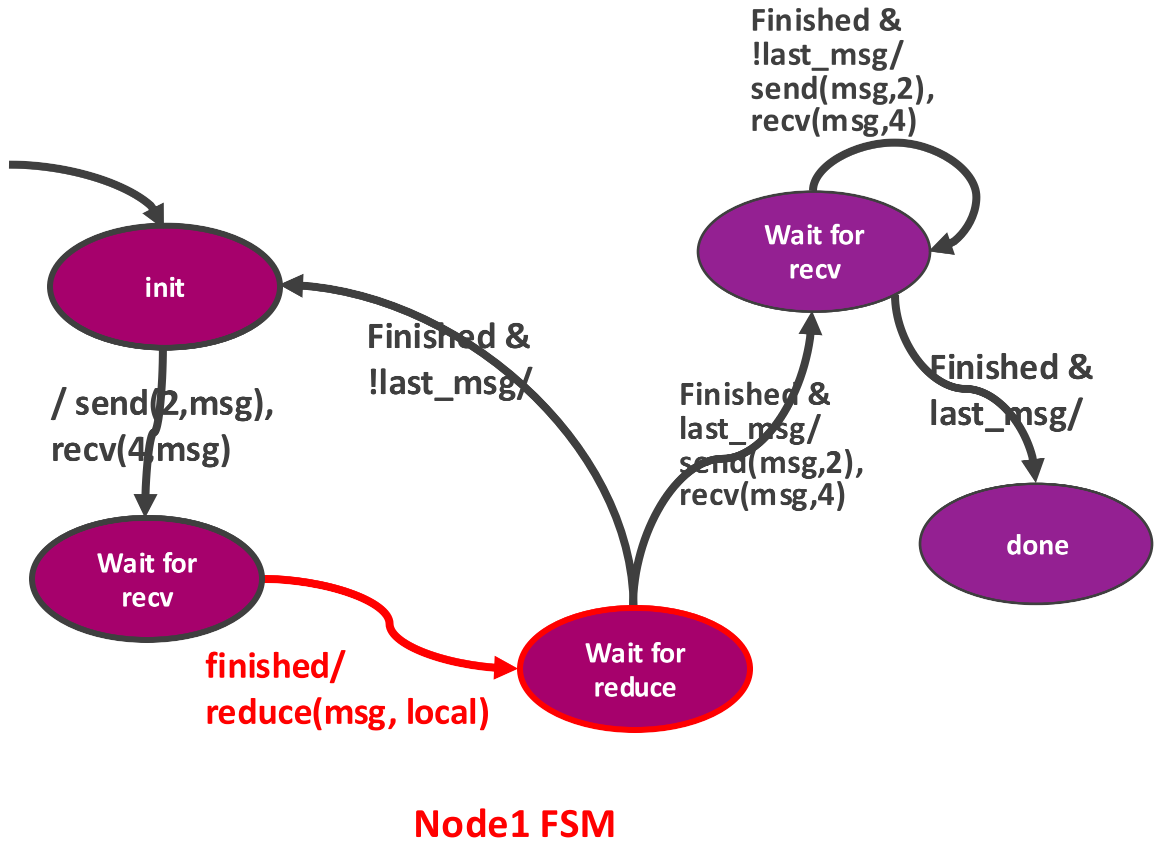

init
/ send(2,msg), recv(4,msg)
Wait for recv
finished/ reduce(msg, local)
Wait for reduce
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Wait for recv
Finished & !last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done

Node1 FSM

System Layer Collective Implementation

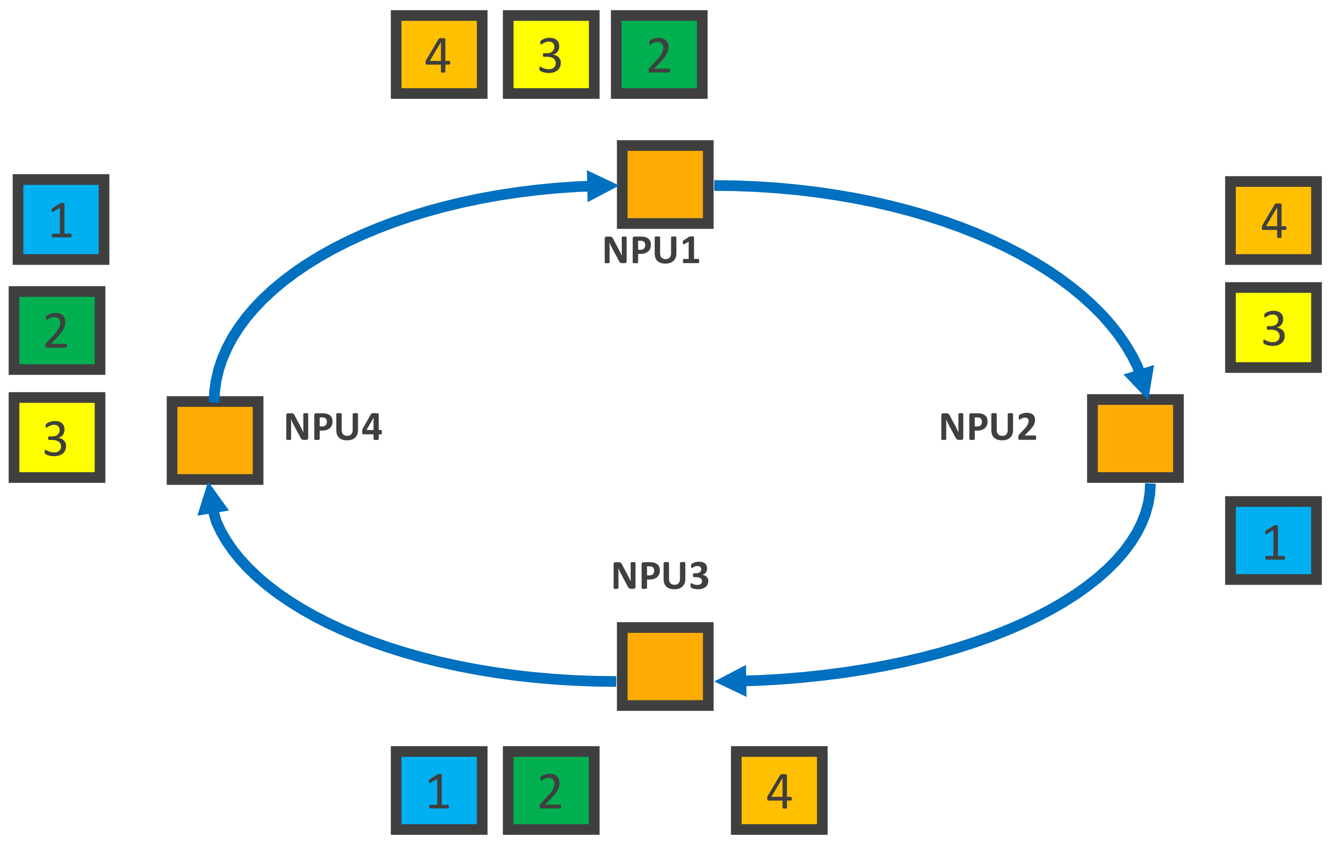
- Collective algorithms can be implemented using **state machines**.

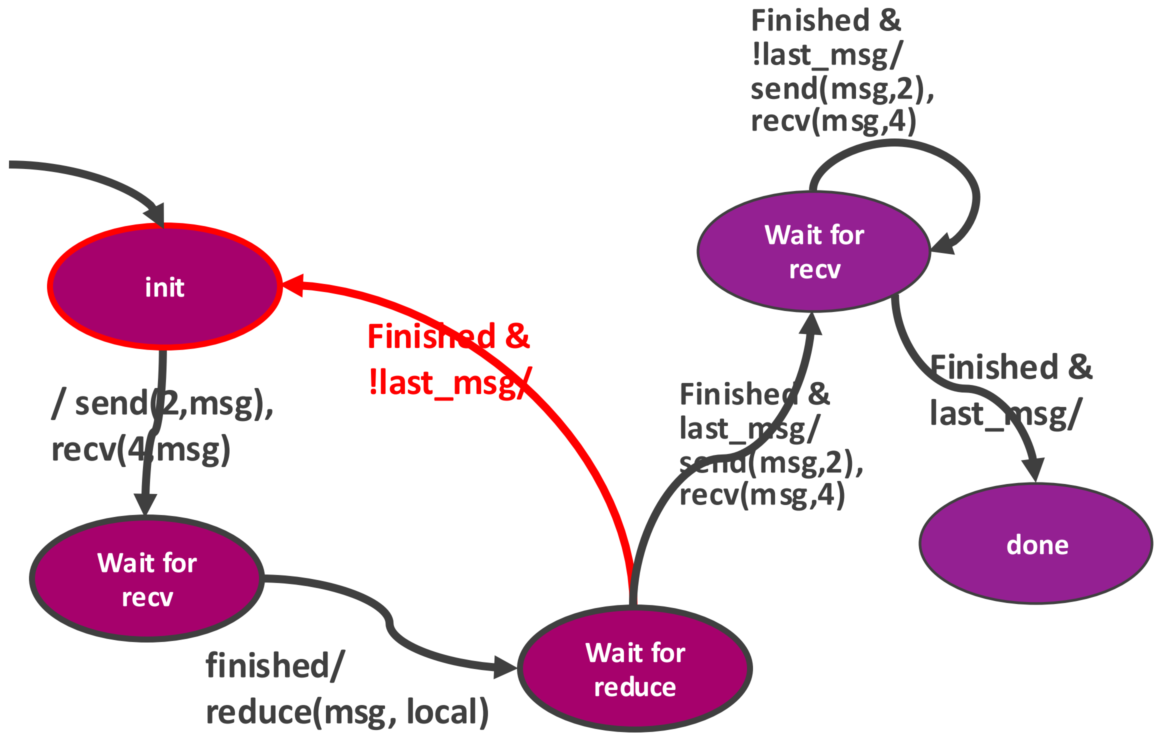



Finished & !last_msg/ send(msg,2), recv(msg,4)
init
Wait for recv
/ send(2,msg), recv(4,msg)
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
Wait for recv
done
finished/ reduce(msg, local)
Wait for reduce
Node1 FSM

System Layer Collective Implementation

- Collective algorithms can be implemented using **state machines**.

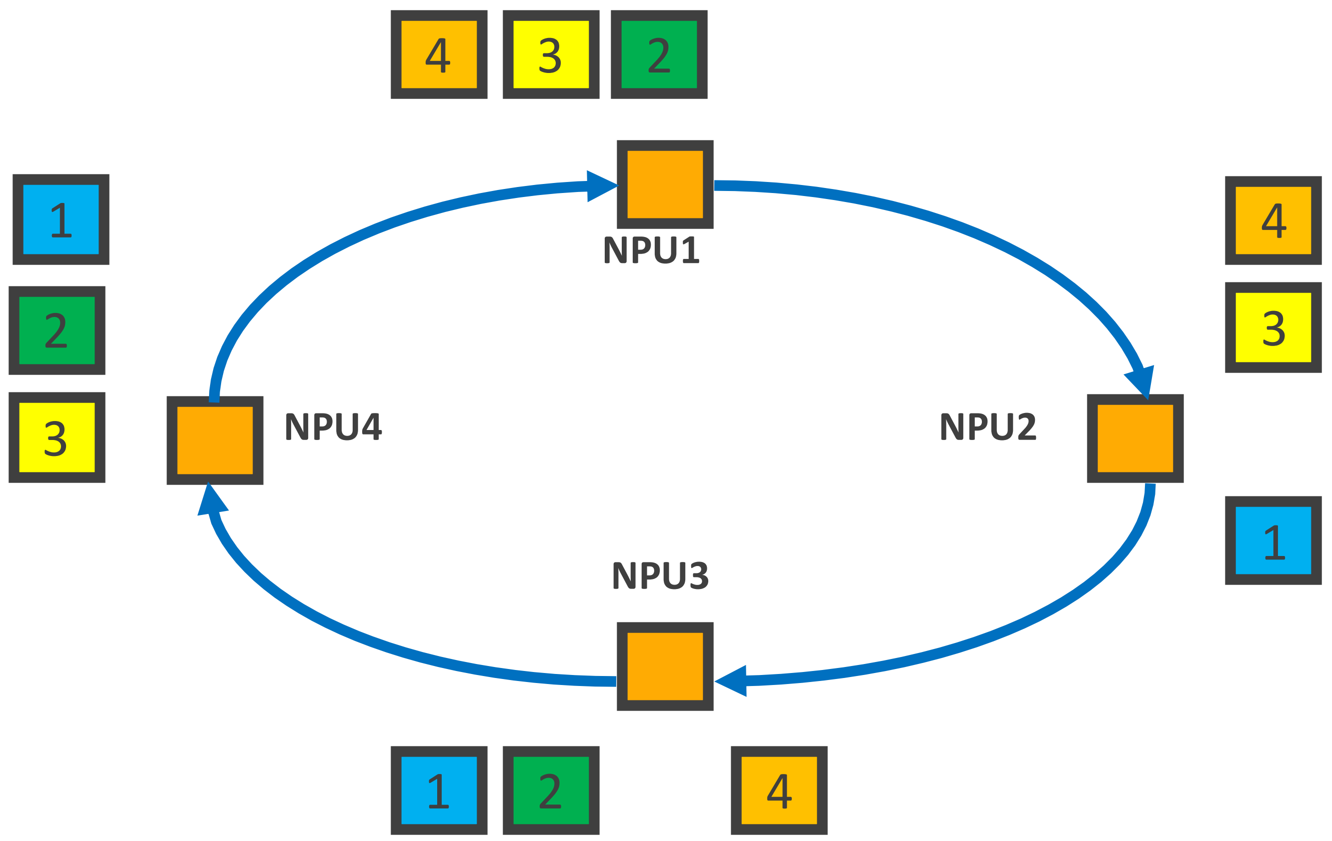


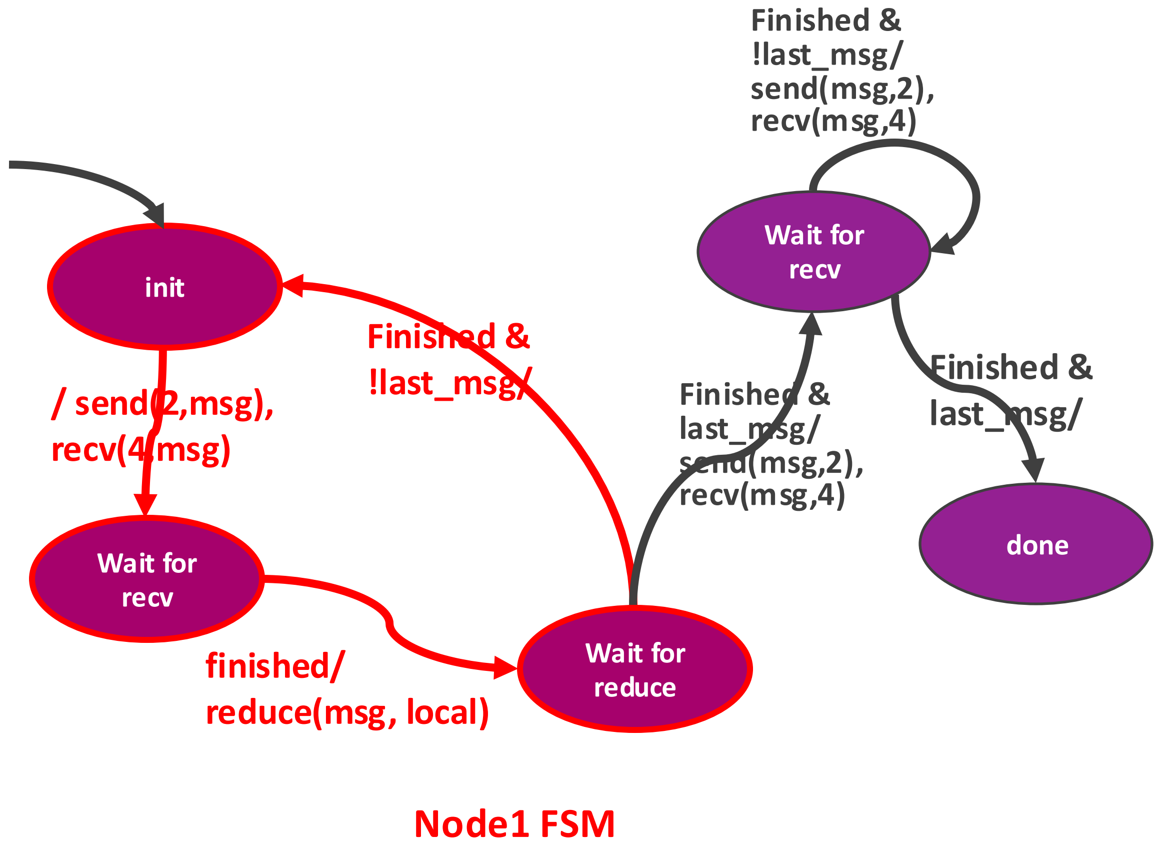

init
/ send(2,msg), recv(4,msg)
Wait for recv
finished/ reduce(msg, local)
Wait for reduce
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Wait for recv
Finished & !last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done

Node1 FSM

System Layer Collective Implementation

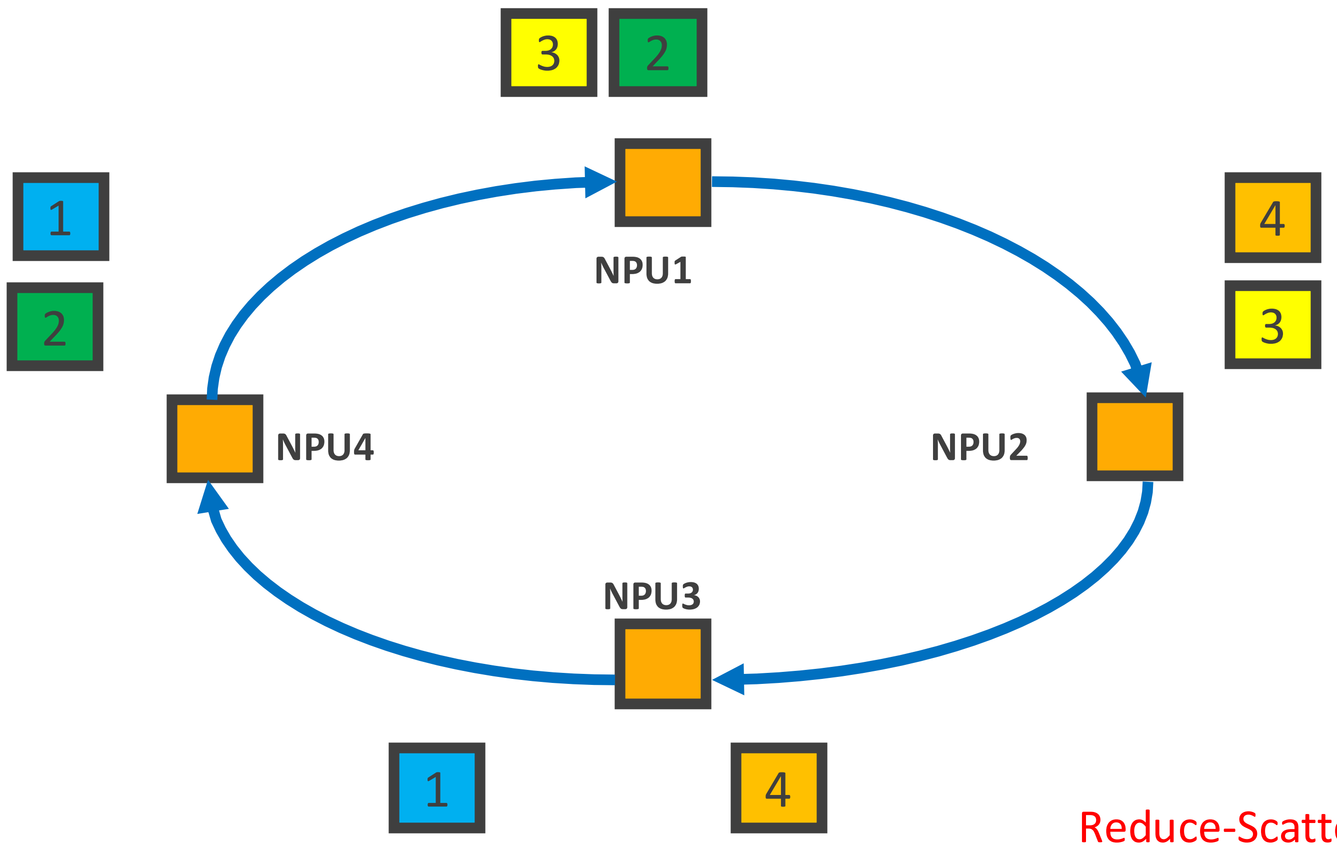
- Collective algorithms can be implemented using **state machines**.

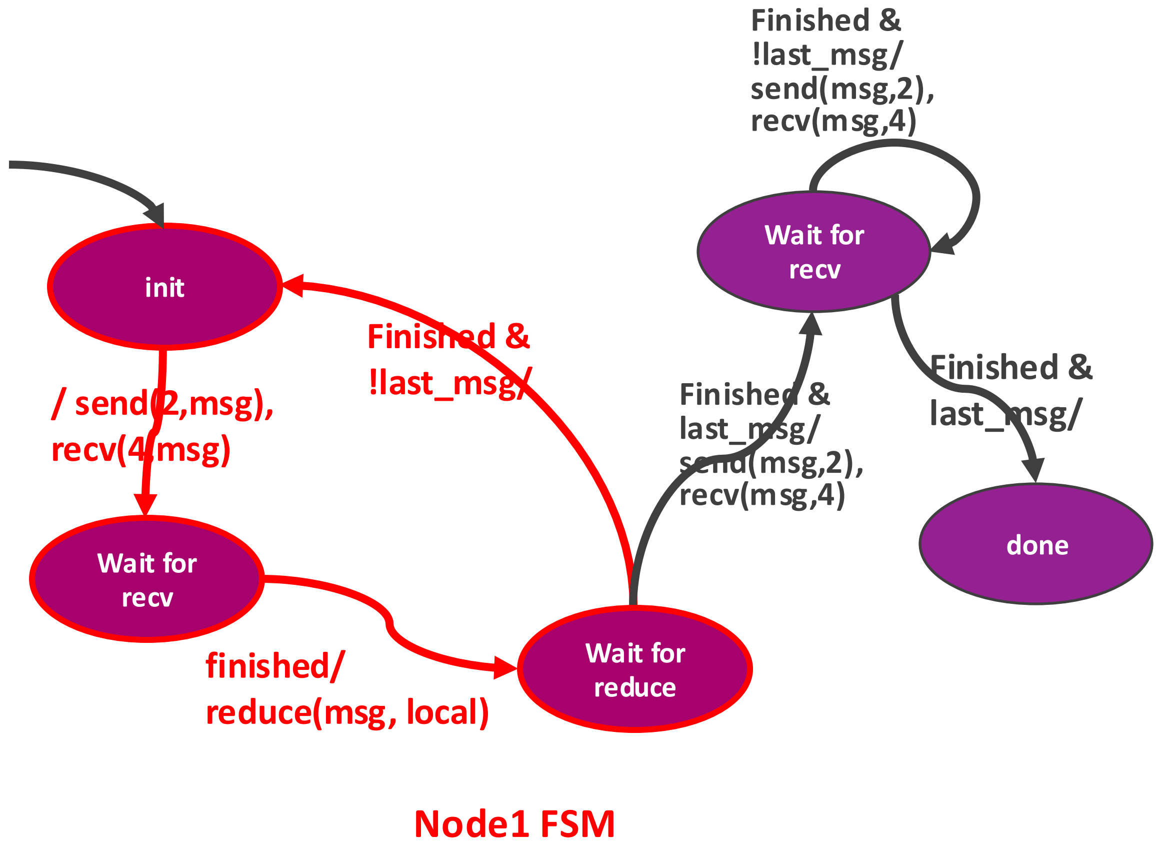



init
/ send(2,msg), recv(4,msg)
Wait for recv
finished/ reduce(msg, local)
Wait for reduce
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Wait for recv
Finished & !last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done
Node1 FSM

System Layer Collective Implementation

- Collective algorithms can be implemented using **state machines**.

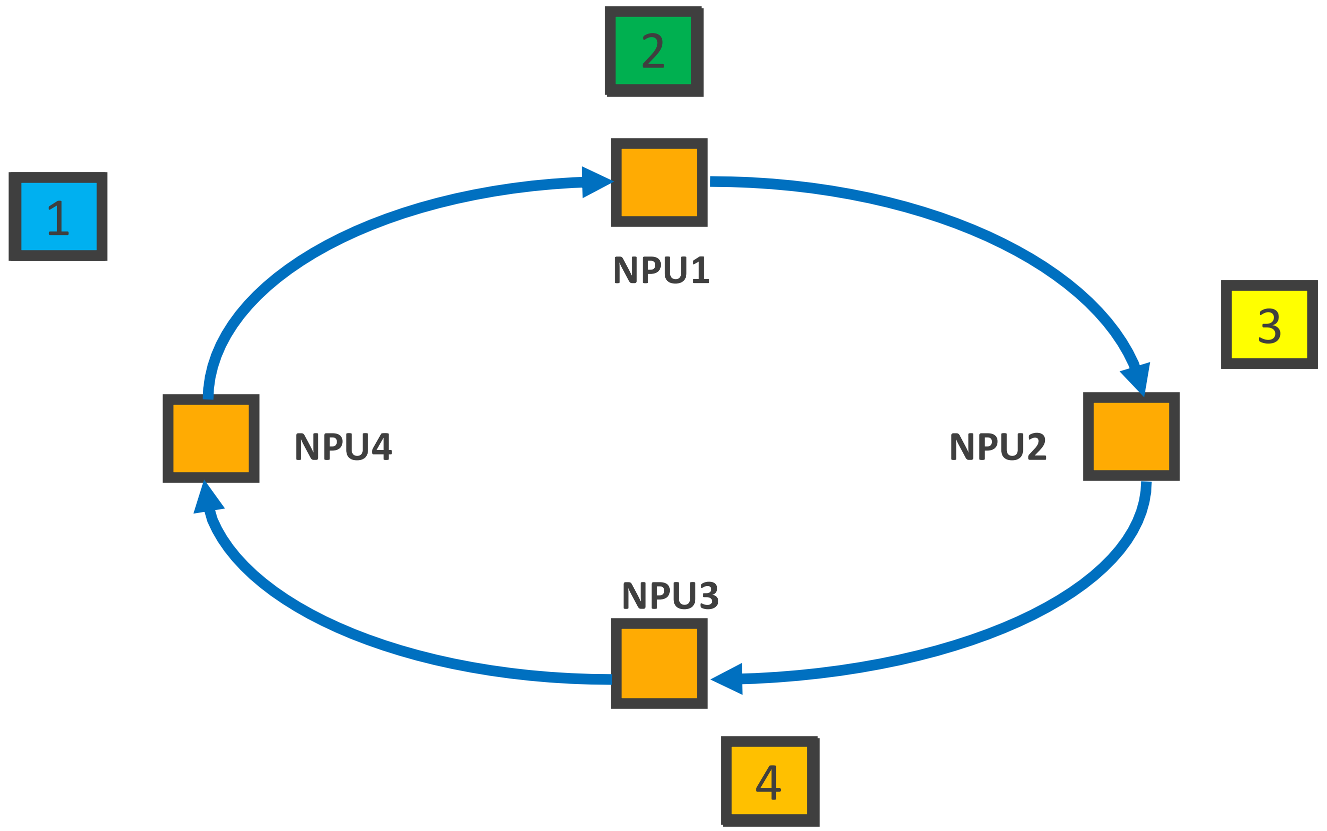


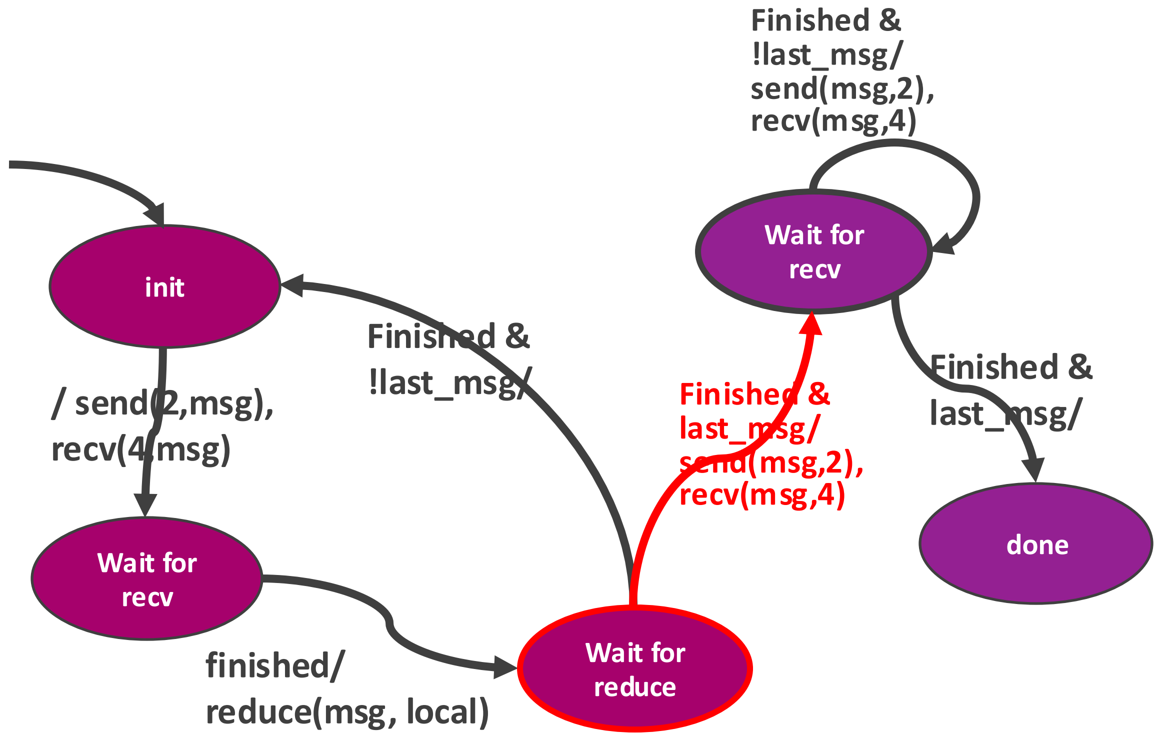


Reduce-Scatter done!

System Layer Collective Implementation

- Collective algorithms can be implemented using **state machines**.

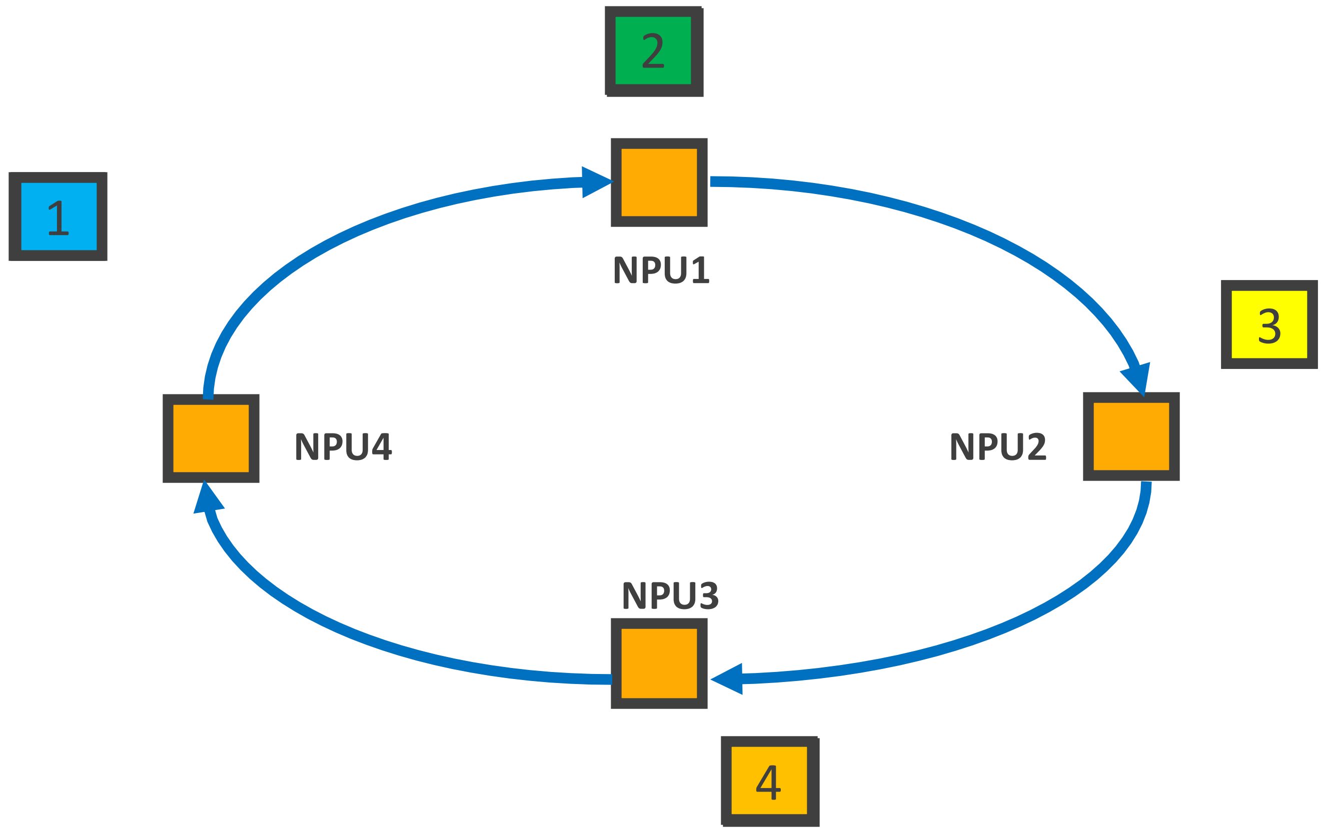


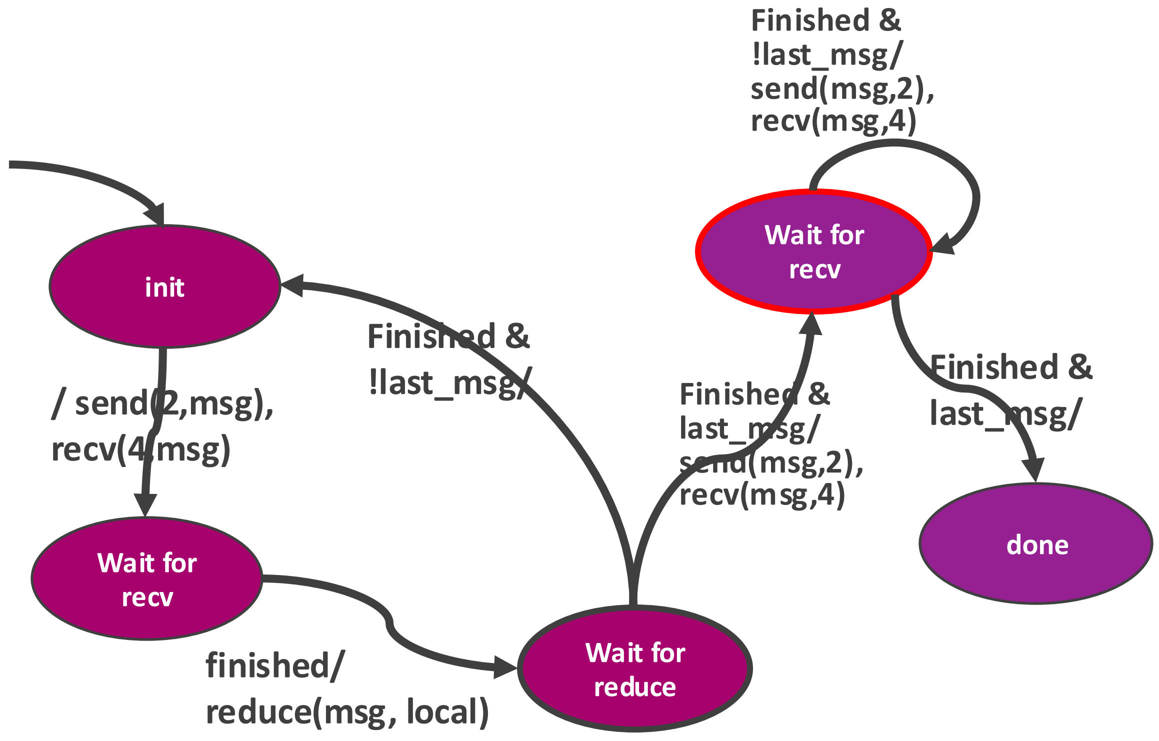

init
/ send(2,msg), recv(4,msg)
Wait for recv
finished/ reduce(msg, local)
Wait for reduce
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Wait for recv
Finished & !last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done

Node1 FSM

System Layer Collective Implementation

- Collective algorithms can be implemented using **state machines**.

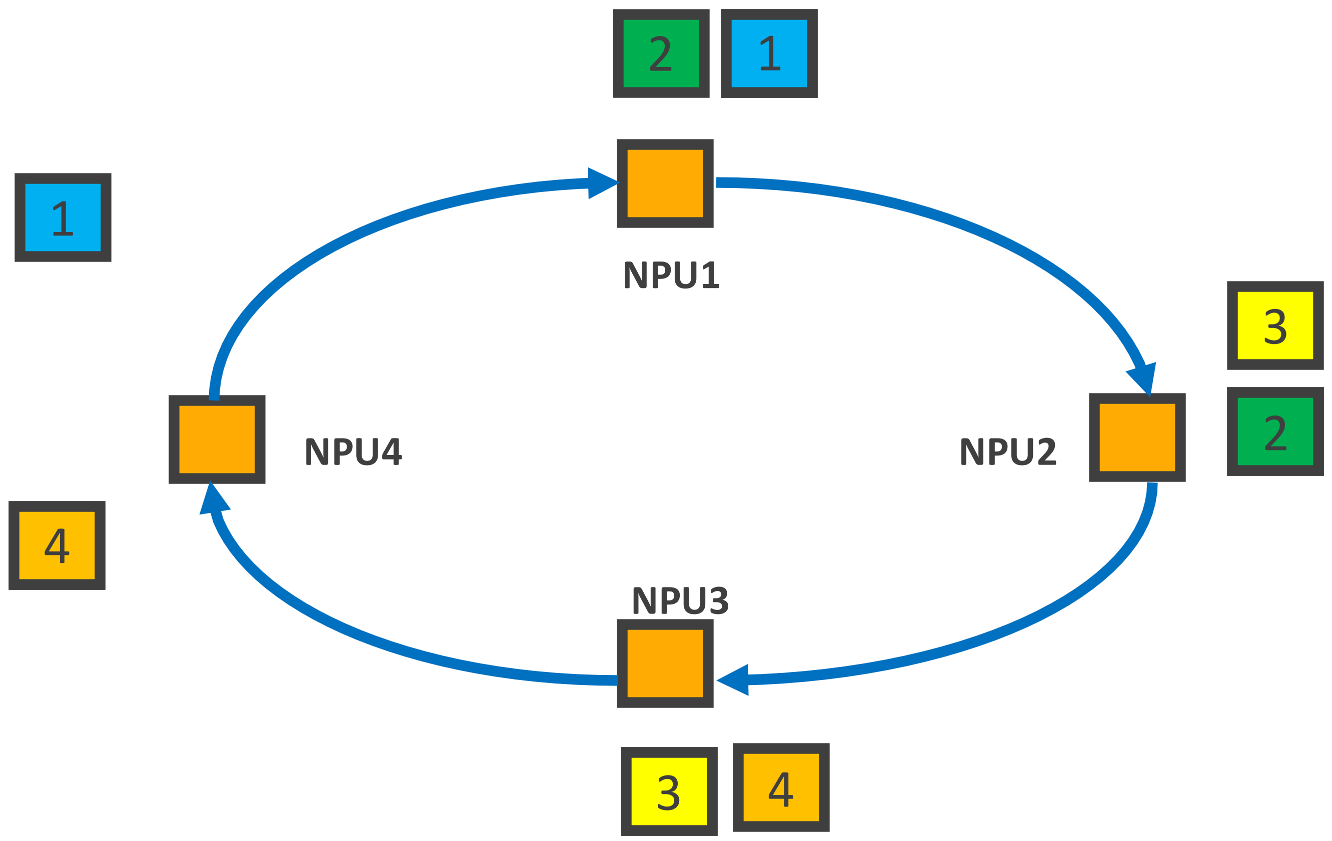


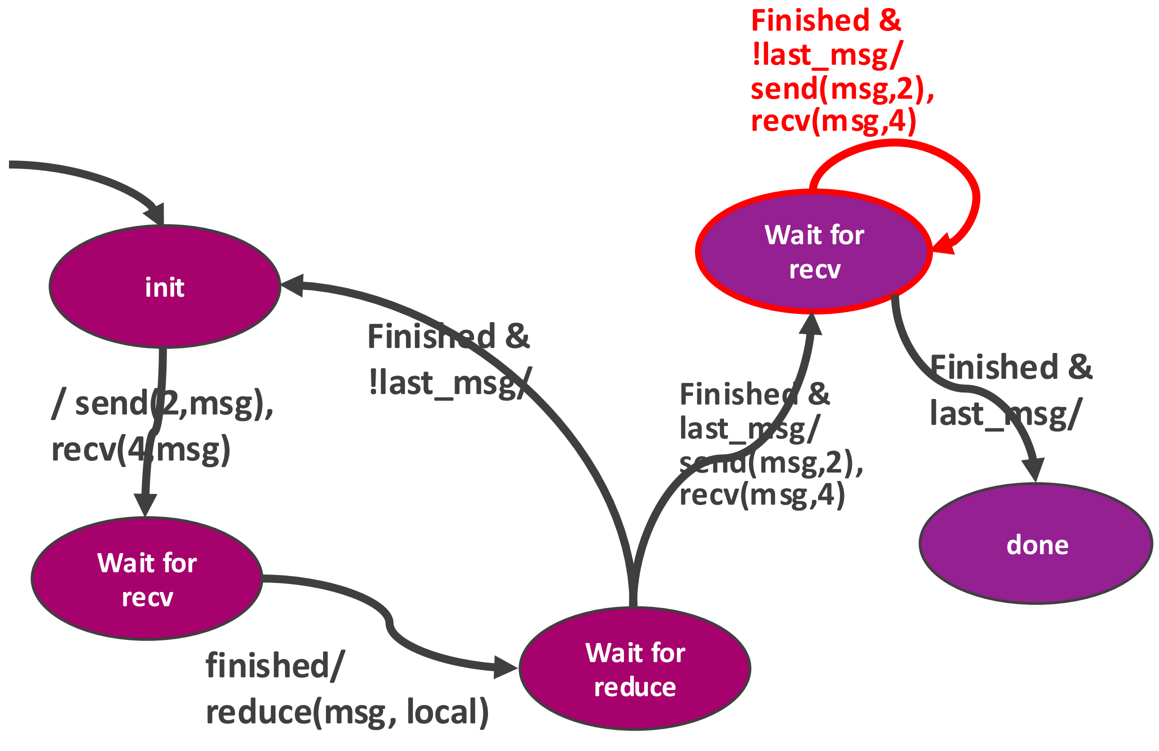

init
/ send(2,msg), recv(4,msg)
Wait for recv
finished/ reduce(msg, local)
Wait for reduce
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Wait for recv
Finished & !last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done

Node1 FSM

System Layer Collective Implementation

- Collective algorithms can be implemented using **state machines**.

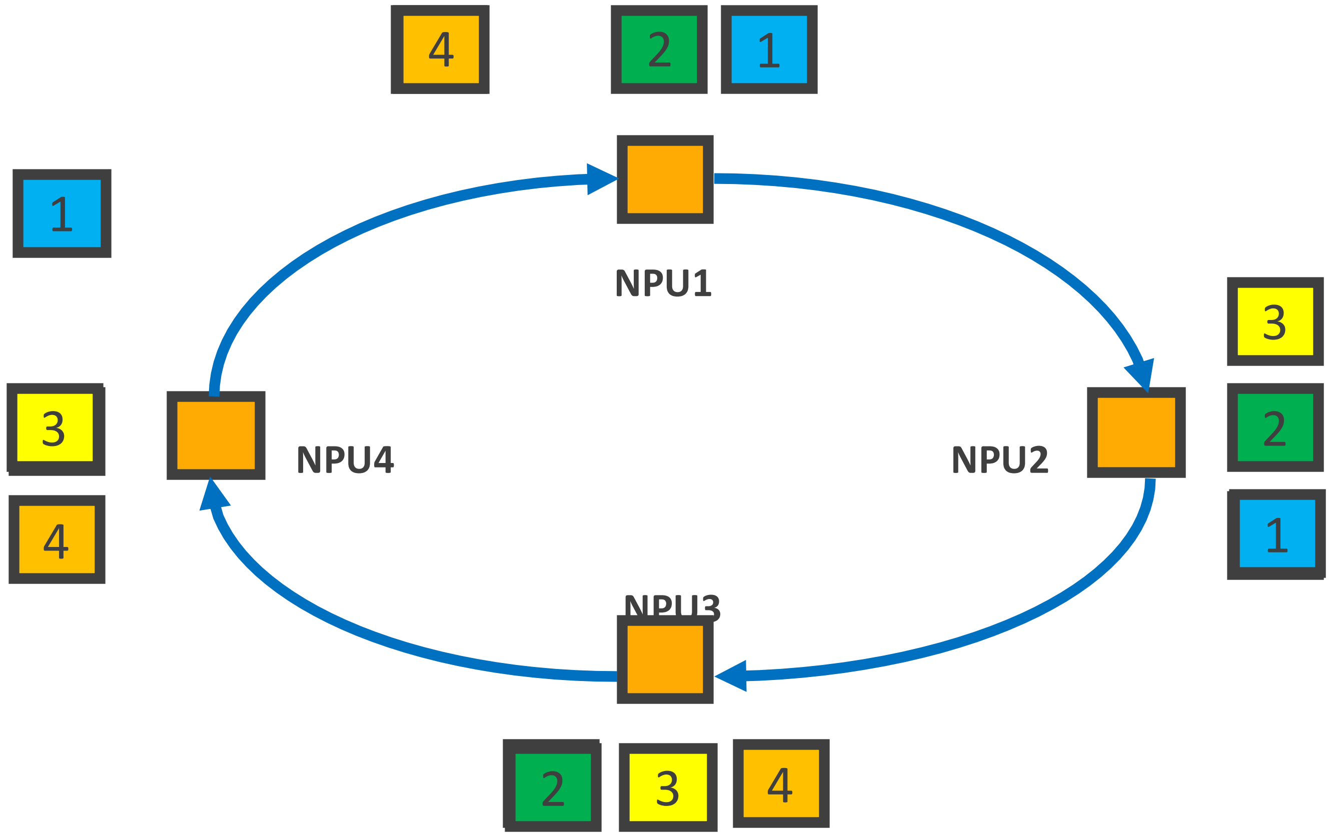


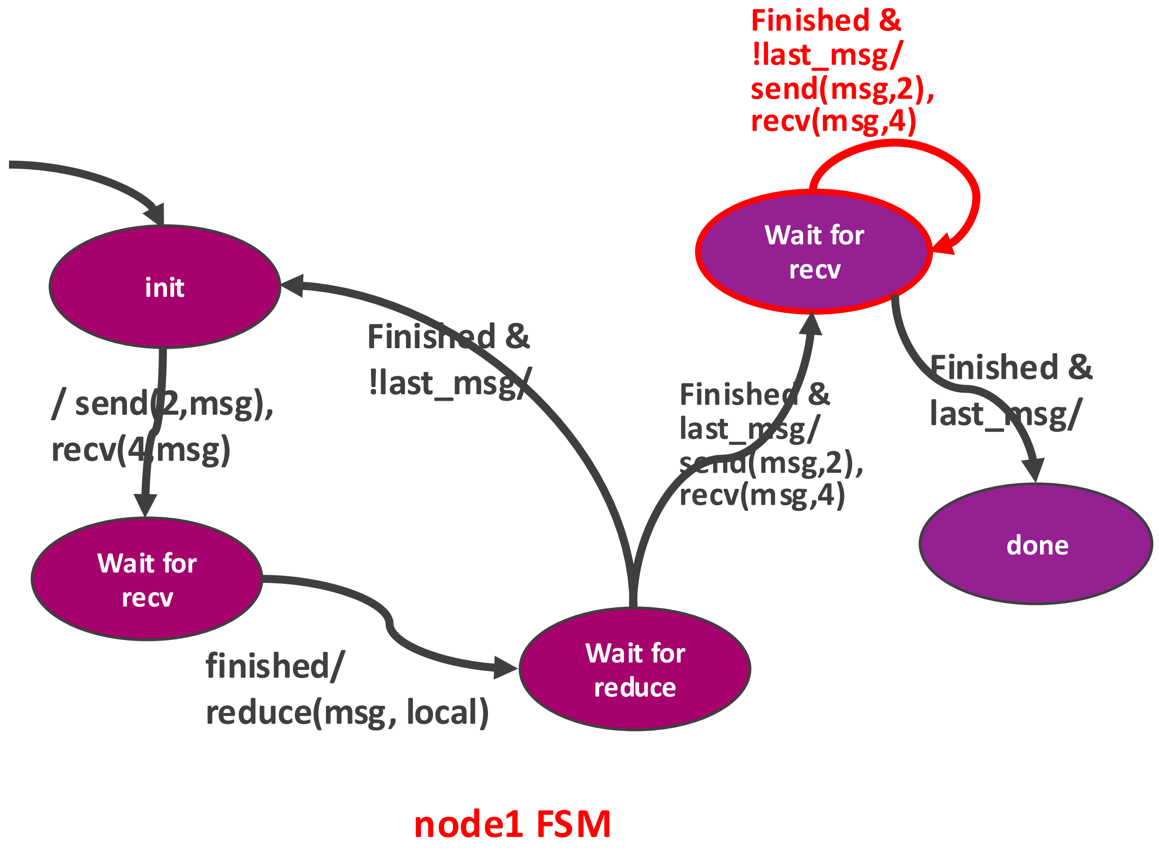

Finished & !last_msg/ send(msg,2), recv(msg,4)
init
Wait for recv
/ send(2,msg), recv(4,msg)
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done
Wait for recv
finished/ reduce(msg, local)
Wait for reduce

Node1 FSM

System Layer Collective Implementation

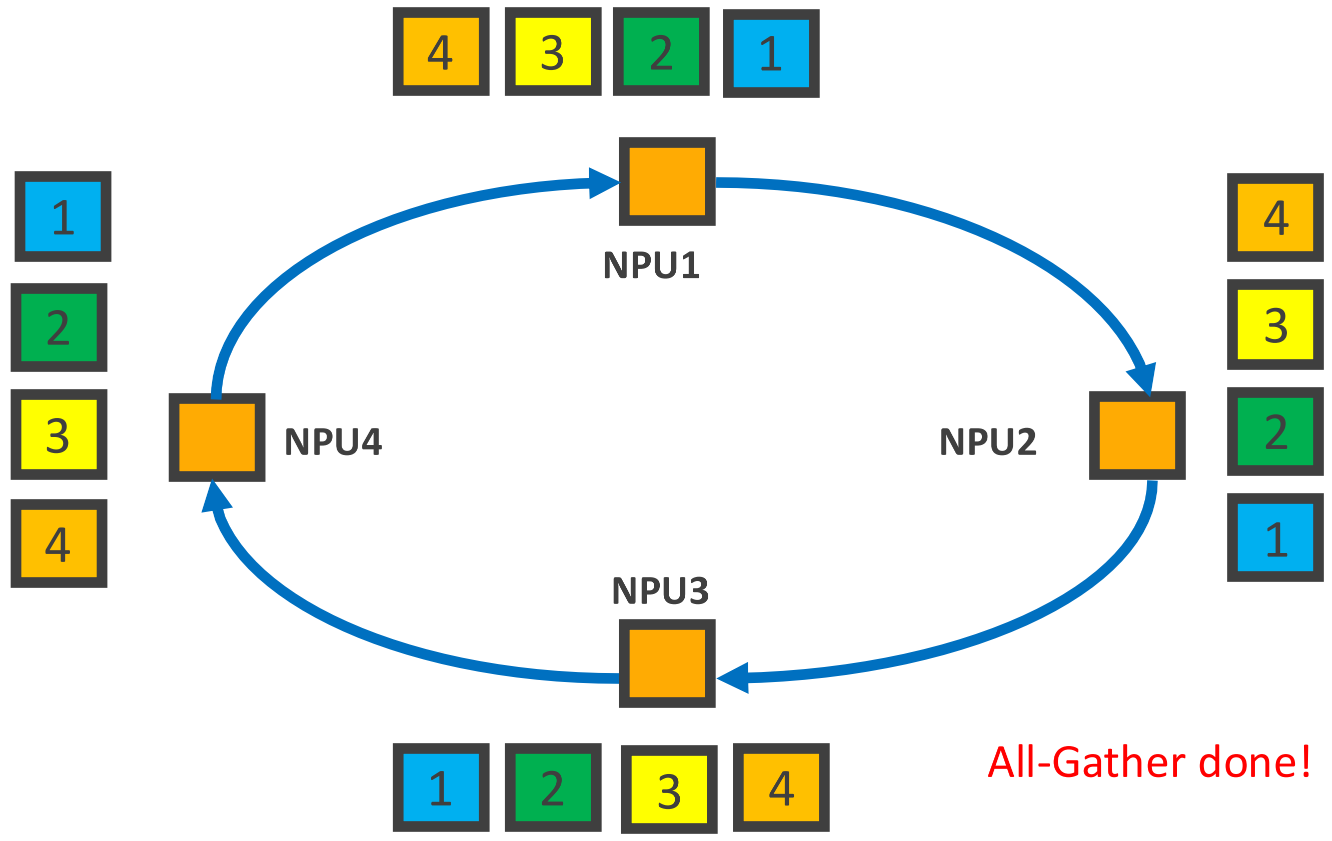
- Collective algorithms can be implemented using **state machines**.



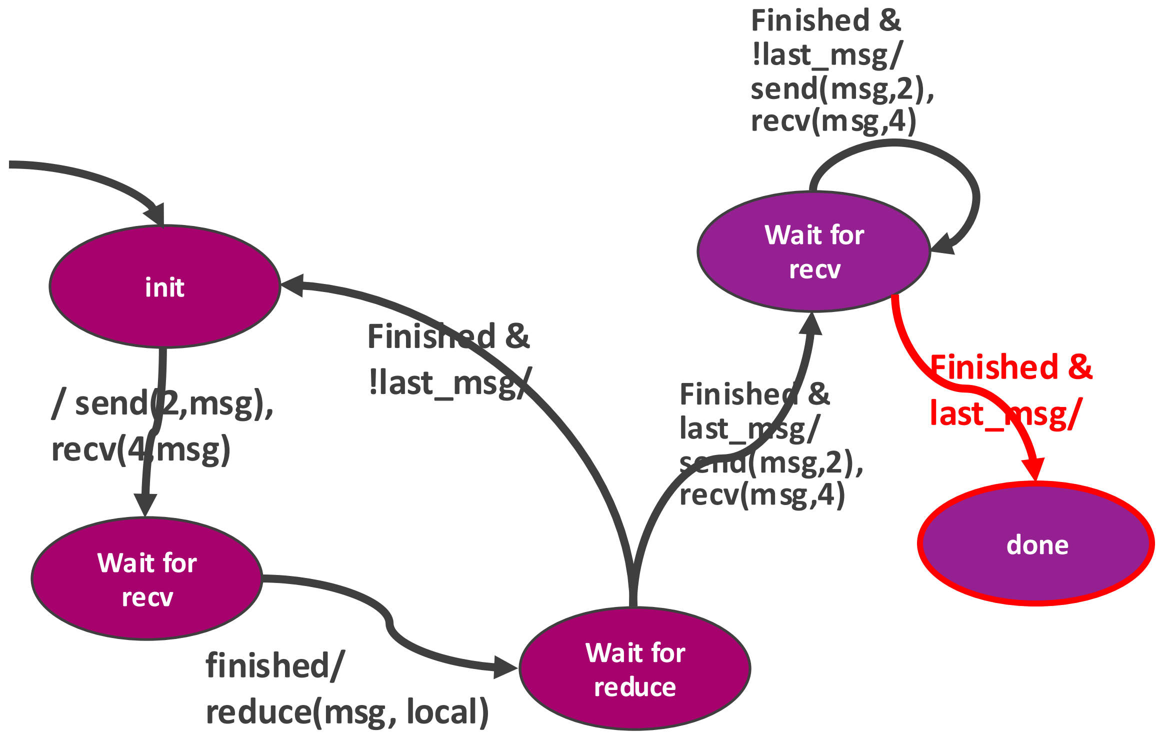

Finished & !last_msg/ send(msg,2), recv(msg,4)
init
Wait for recv
Finished & !last_msg/
/ send(2,msg), recv(4,msg)
Finished & last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
Wait for recv
done
finished/ reduce(msg, local)
Wait for reduce
node1 FSM

System Layer Collective Implementation

- Collective algorithms can be implemented using **state machines**.



All-Gather done!

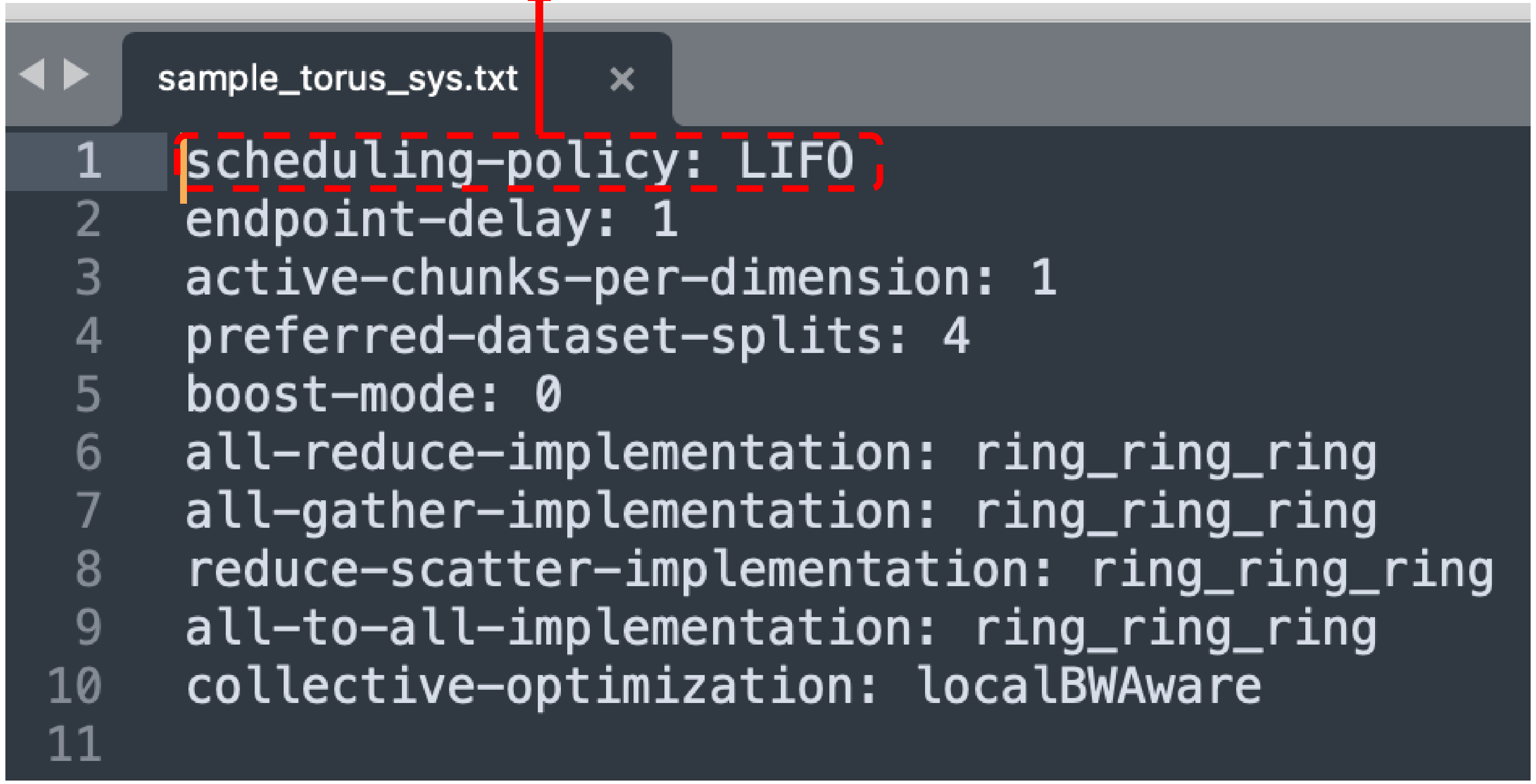


init
/ send(2,msg), recv(4,msg)
Wait for recv
finished/ reduce(msg, local)
Wait for reduce
Finished & !last_msg/
Finished & last_msg/ send(msg,2), recv(msg,4)
Wait for recv
Finished & !last_msg/ send(msg,2), recv(msg,4)
Finished & last_msg/
done

node1 FSM

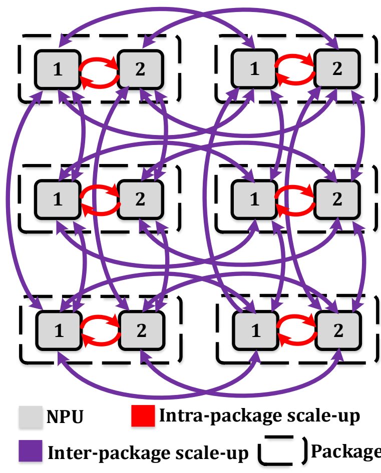
System Input

Collective Policy for



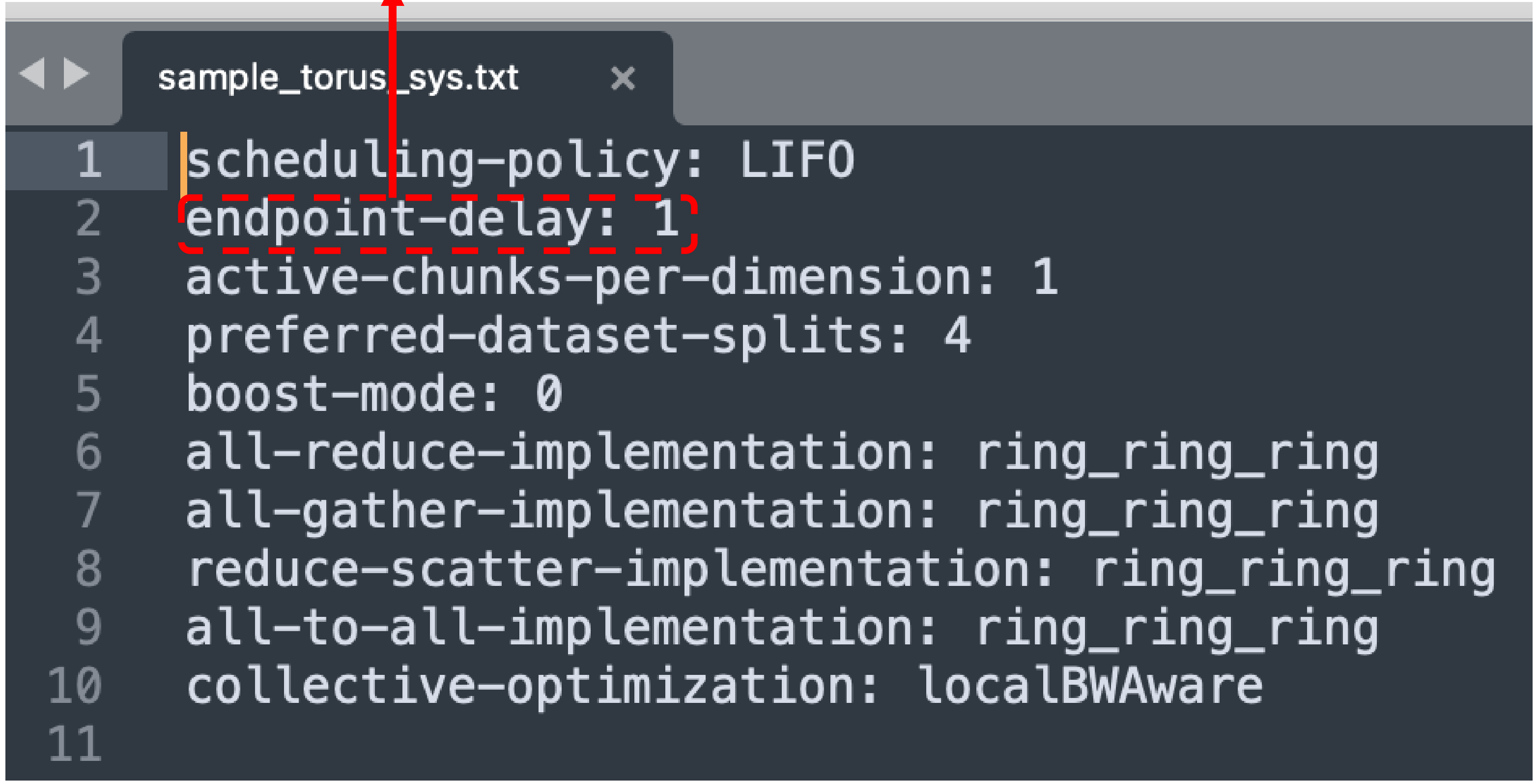
sample_torus_sys.txt

```
scheduling-policy: LIFO
endpoint-delay: 1
active-chunks-per-dimension: 1
preferred-dataset-splits: 4
boost-mode: 0
all-reduce-implementation: ring_ring_ring
all-gather-implementation: ring_ring_ring
reduce-scatter-implementation: ring_ring_ring
all-to-all-implementation: ring_ring_ring
collective-optimization: localBWAware
```

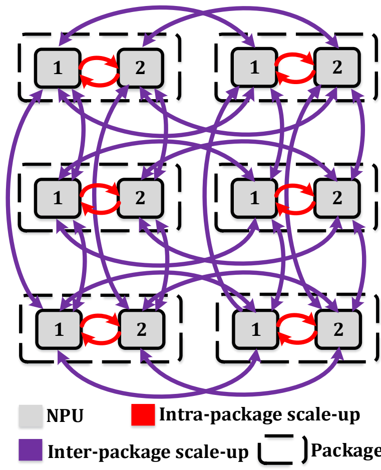


System Input

Constant delay before NPU sending a message

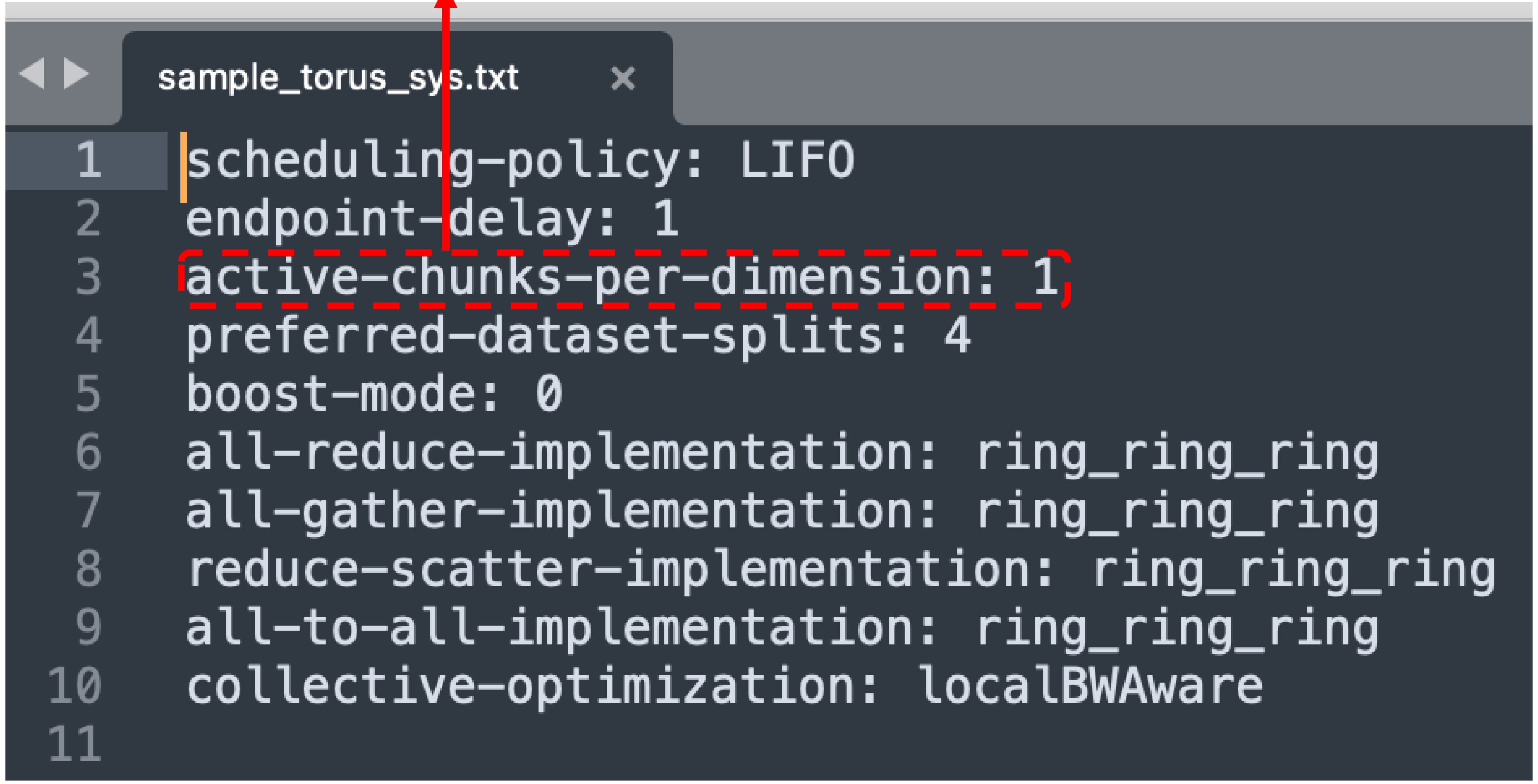

sample_torus_sys.txt

```
scheduling-policy: LIFO
endpoint-delay: 1
active-chunks-per-dimension: 1
preferred-dataset-splits: 4
boost-mode: 0
all-reduce-implementation: ring_ring_ring
all-gather-implementation: ring_ring_ring
reduce-scatter-implementation: ring_ring_ring
all-to-all-implementation: ring_ring_ring
collective-optimization: localBWAware
```



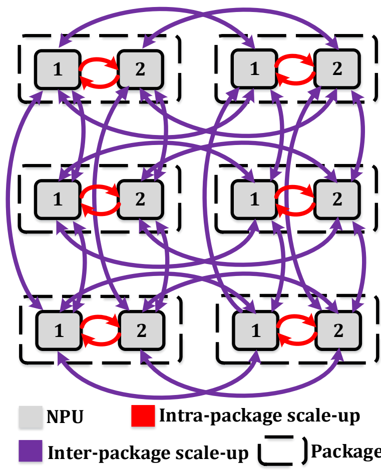
System Input

Max running chunks per each physical network dimension



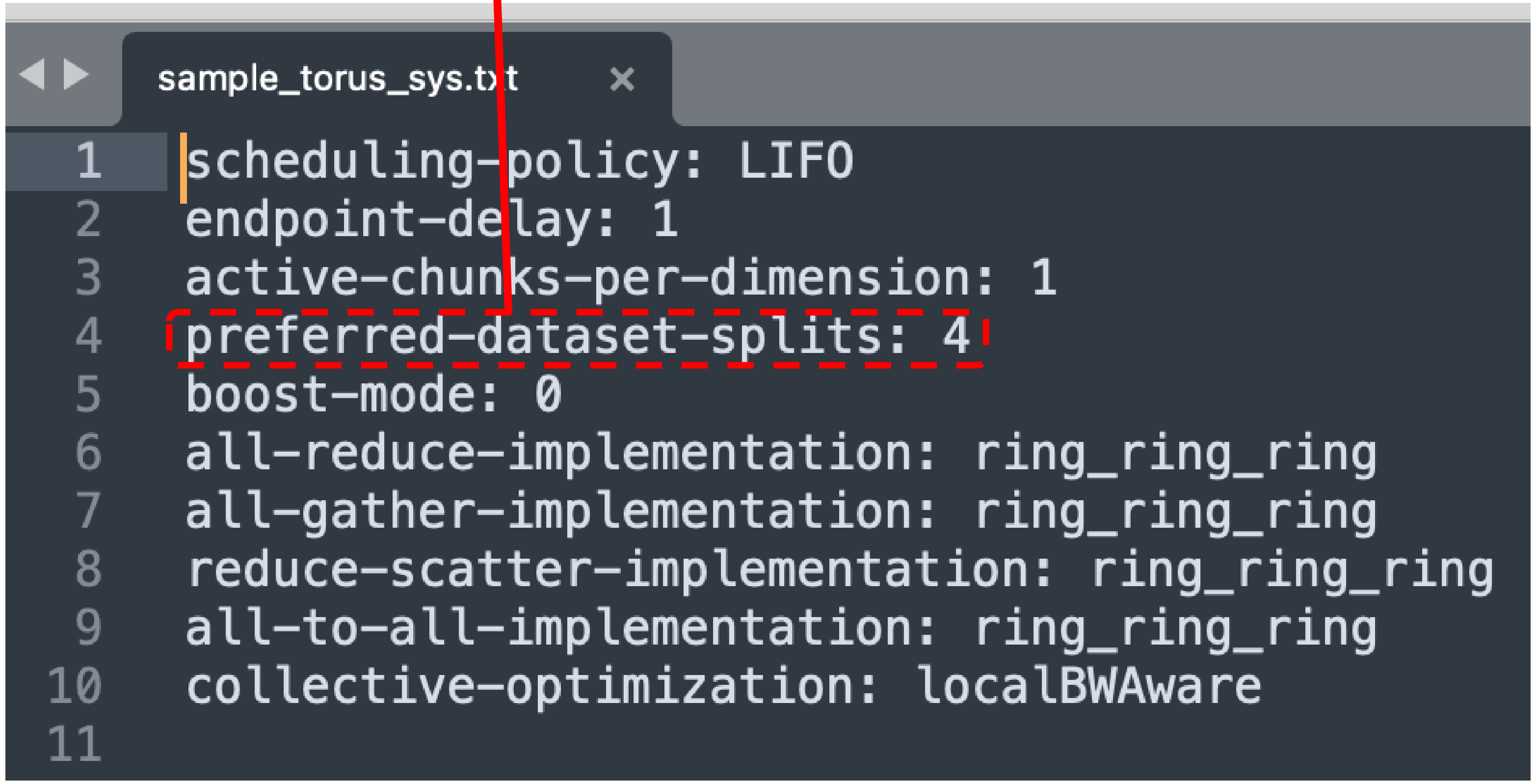
sample_torus_sys.txt

```
scheduling-policy: LIFO
endpoint-delay: 1
active-chunks-per-dimension: 1
preferred-dataset-splits: 4
boost-mode: 0
all-reduce-implementation: ring_ring_ring
all-gather-implementation: ring_ring_ring
reduce-scatter-implementation: ring_ring_ring
all-to-all-implementation: ring_ring_ring
collective-optimization: localBWAware

```



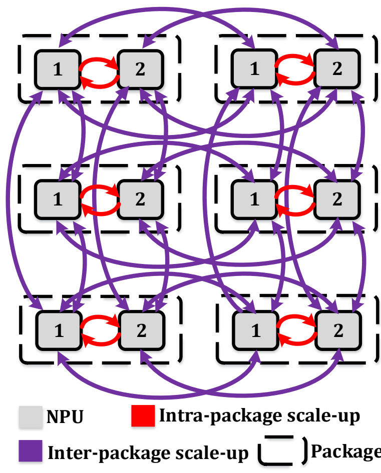
System Input

# of chunks to split each collective into

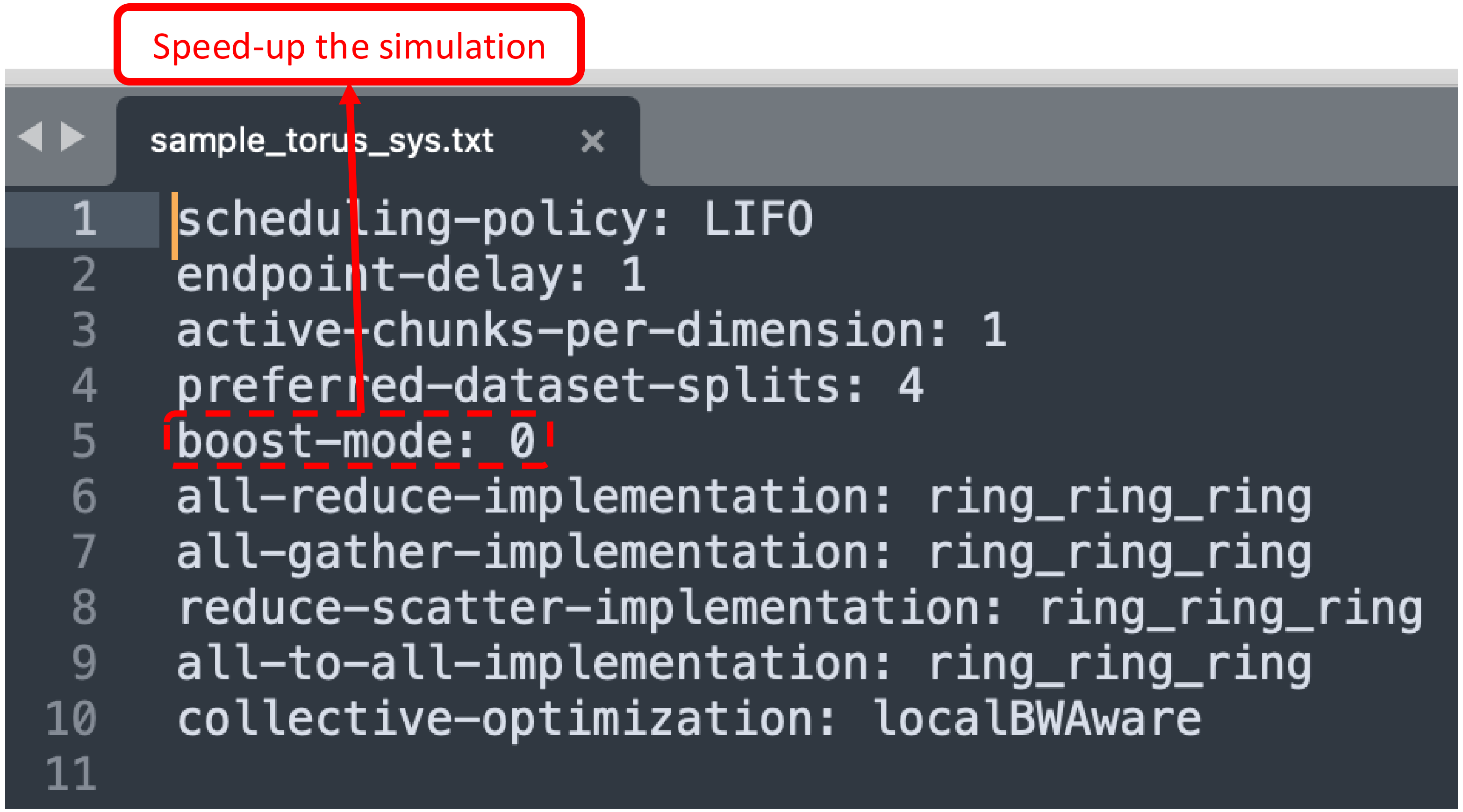


sample_torus_sys.txt

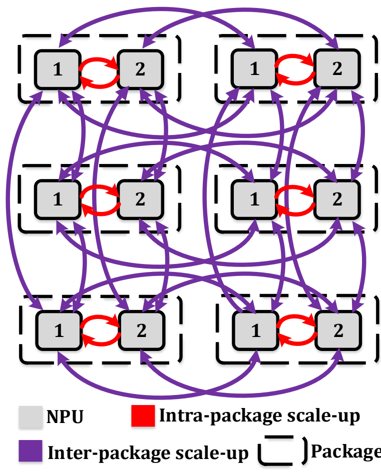
```
scheduling-policy: LIFO
endpoint-delay: 1
active-chunks-per-dimension: 1
preferred-dataset-splits: 4
boost-mode: 0
all-reduce-implementation: ring_ring_ring
all-gather-implementation: ring_ring_ring
reduce-scatter-implementation: ring_ring_ring
all-to-all-implementation: ring_ring_ring
collective-optimization: localBWAware
```



System Input

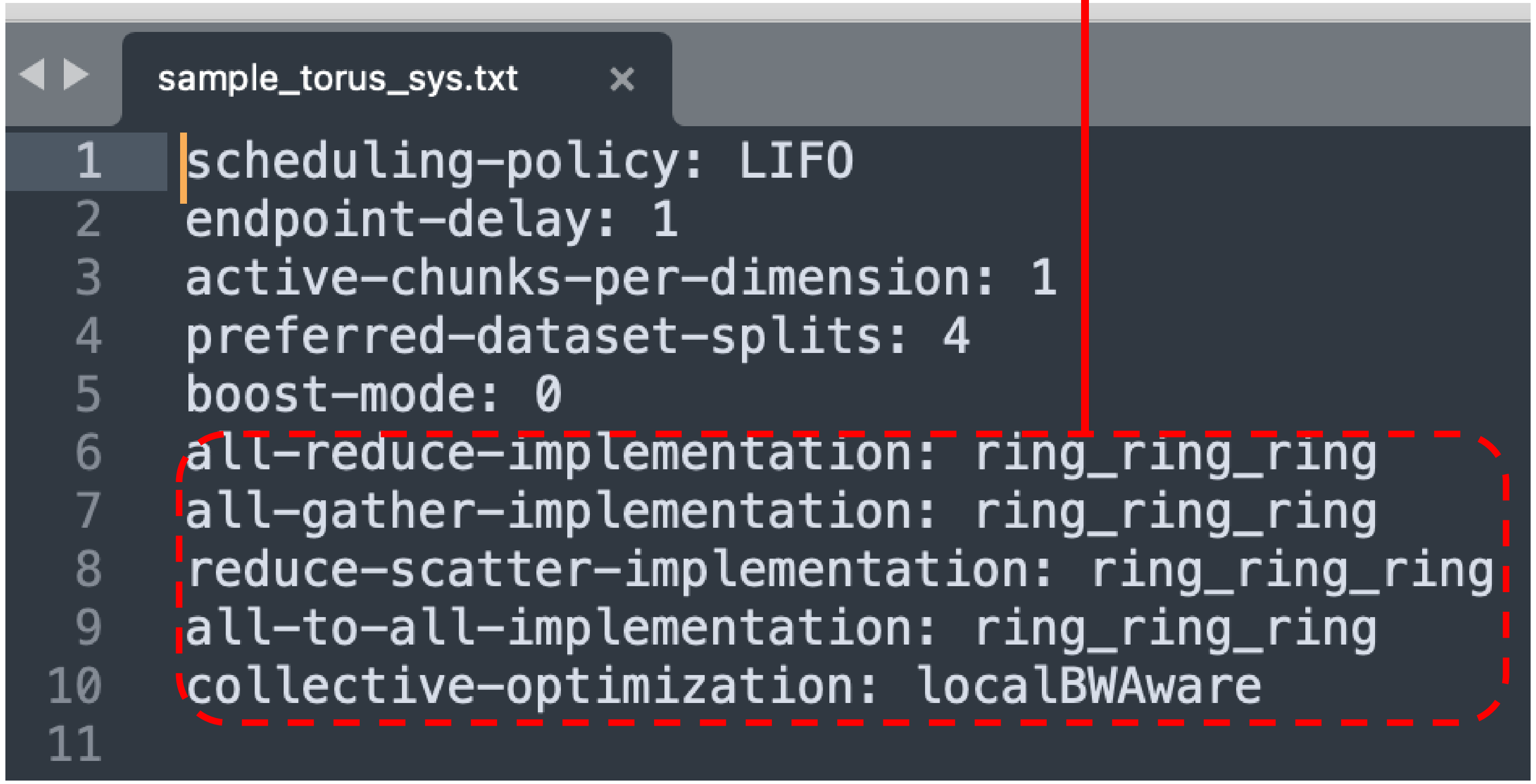


```
scheduling-policy: LIFO
endpoint-delay: 1
active-chunks-per-dimension: 1
preferred-dataset-splits: 4
boost-mode: 0
all-reduce-implementation: ring_ring_ring
all-gather-implementation: ring_ring_ring
reduce-scatter-implementation: ring_ring_ring
all-to-all-implementation: ring_ring_ring
collective-optimization: localBWAware
```

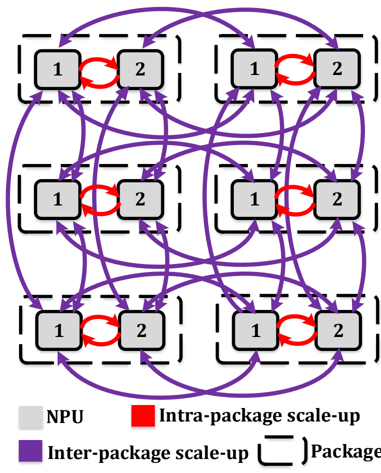


System Input

Hierarchical collective algorithm implementation



```
scheduling-policy: LIFO
endpoint-delay: 1
active-chunks-per-dimension: 1
preferred-dataset-splits: 4
boost-mode: 0
all-reduce-implementation: ring_ring_ring
all-gather-implementation: ring_ring_ring
reduce-scatter-implementation: ring_ring_ring
all-to-all-implementation: ring_ring_ring
collective-optimization: localBWAware
```



Sneak Peek: Collective API

- MSCCLang high-level DSE to represent collective algorithm

```
def allgather_ring(size, channels, instances, protocol):
    (...)
    for step in range(0, size-1):
        for index in range(0, size):
                    rank = (index + step) % size
                    next_rank = (index + step + 1) % size
                    c = chunk(rank, Buffer.output, index)
                    c = c.copy(next_rank, Buffer.output, index, sendtb=channel, recvtb=channel, ch=channel)

    XML()
```

Represent arbitrary collective algorithm and store it in standardized XML format

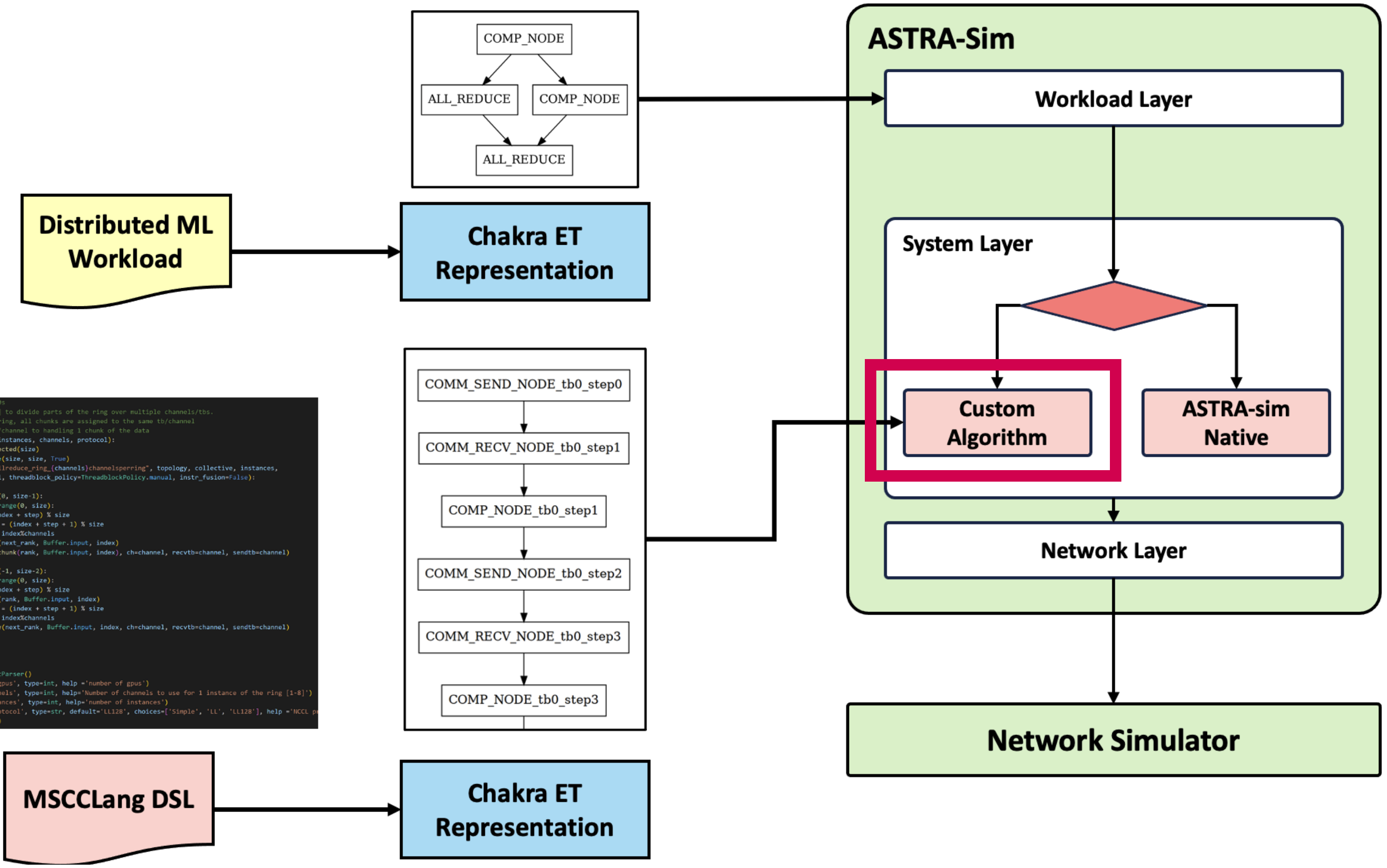
- Custom collective algorithms represented by MSCCL-xml

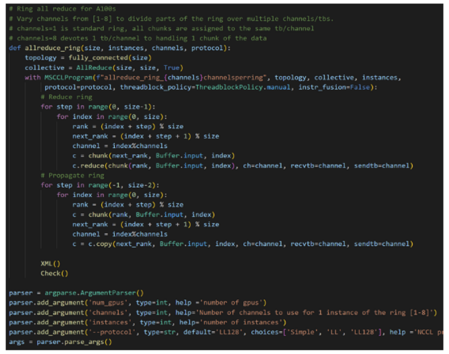
```
<algo name="allreduce_ring_1channelsperring" (...) >
<gpu id="0" i_chunks="4" o_chunks="0" s_chunks="0">

<tb id="0" send="1" recv="3" chan="0">
<step s="0" type="s" srcbuf="i" srcoff="0" dstbuf="i" dstoff="0" cnt="1" depid="-1" deps="-1" hasdep="0"/>
<step s="1" type="rrc" srcbuf="i" srcoff="3" dstbuf="i" dstoff="3" cnt="1" depid="-1" deps="-1" hasdep="0"/>
<step s="2" type="s" srcbuf="i" srcoff="3" dstbuf="i" dstoff="3" cnt="1" depid="-1" deps="-1" hasdep="0"/>
<step s="3" type="rrc" srcbuf="i" srcoff="2" dstbuf="i" dstoff="2" cnt="1" depid="-1" deps="-1" hasdep="0"/>

(...)
```

Sneak Peek: Collective API

Work in Progress!

- HotI paper: *https://arxiv.org/abs/2408.11008*





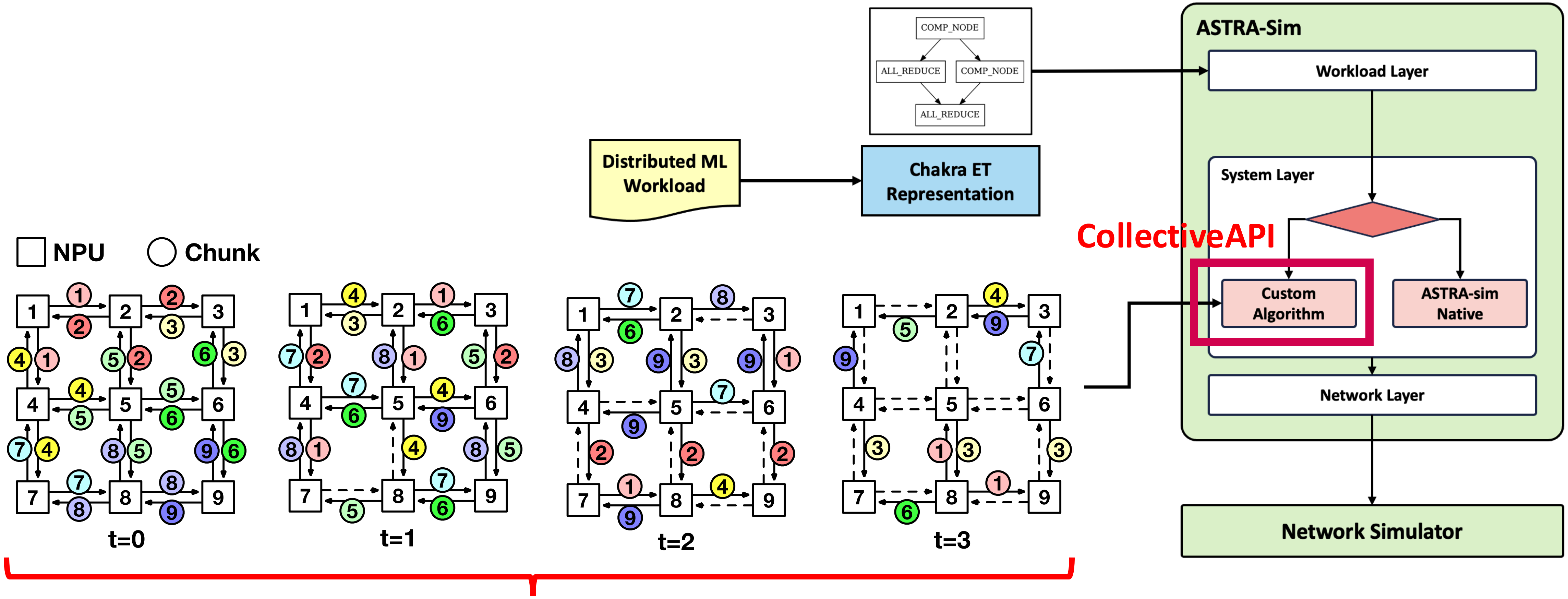
```
# Ring all reduce for A100s
# Vary channels from [1-8] to divide parts of the ring over multiple channels/tbs.
# channels=1 is standard ring, all chunks are assigned to the same tb/channel
# channels=8 devotes 1 tb/channel to handling 1 chunk of the data
def allreduce_ring(size, instances, channels, protocol):
    topology = fully_connected(size)
    collective = AllReduce(size, size, True)
    with MSCCLProgram(f"allreduce_ring_{channels}channelsperring", topology, collective, instances,
            protocol=protocol, threadblock_policy=ThreadblockPolicy.manual, instr_fusion=False):
        # Reduce ring
        for step in range(0, size-1):
            for index in range(0, size):
                rank = (index + step) % size
                next_rank = (index + step + 1) % size
                channel = index%channels
                c = chunk(next_rank, Buffer.input, index)
                c.reduce(chunk(rank, Buffer.input, index), ch=channel, recvtb=channel, sendtb=channel)
        # Propagate ring
        for step in range(-1, size-2):
            for index in range(0, size):
                rank = (index + step) % size
                c = chunk(rank, Buffer.input, index)
                next_rank = (index + step + 1) % size
                channel = index%channels
                c = c.copy(next_rank, Buffer.input, index, ch=channel, recvtb=channel, sendtb=channel)

        XML()
        Check()

parser = argparse.ArgumentParser()
parser.add_argument('num_gpus', type=int, help ='number of gpus')
parser.add_argument('channels', type=int, help='Number of channels to use for 1 instance of the ring [1-8]')
parser.add_argument('instances', type=int, help='number of instances')
parser.add_argument('--protocol', type=str, default='LL128', choices=['Simple', 'LL', 'LL128'], help ='NCCL p
args = parser.parse_args()
```

Slide courtesy: Jinsun Yoo <jinsun@gatech.edu>

Sneak Peek: TACOS

To appear in MICRO '24!
- https://arxiv.org/abs/2304.05301

- A mechanism to generate **topology-aware custom collective plan**



TACOS custom collective plan

Slide courtesy: Jinsun Yoo <jinsun@gatech.edu>