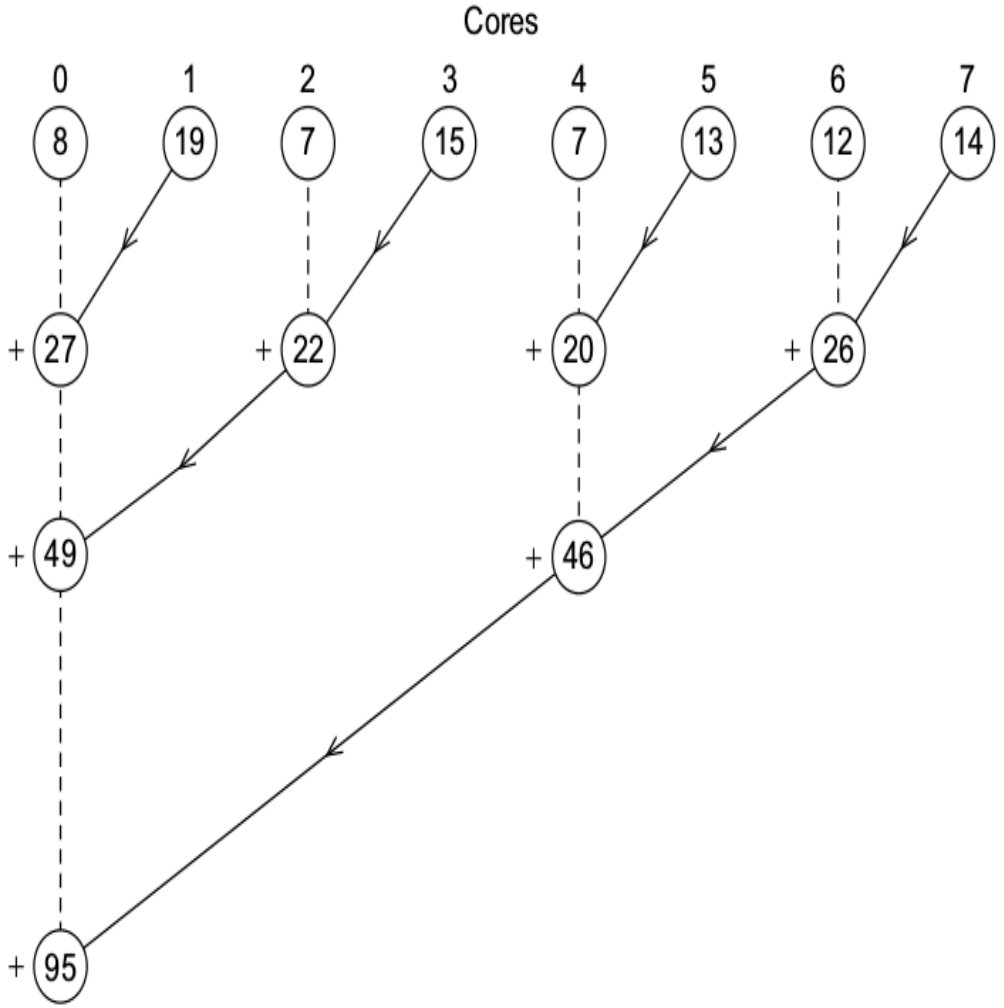


Collective Operations in UCX

Alex Margolin

UCX Hackathon, Dec. 2019

Outline

- whoami
- Current status
- What was built
  - How does it work
  - How does it work with others
- What’s next

whoami

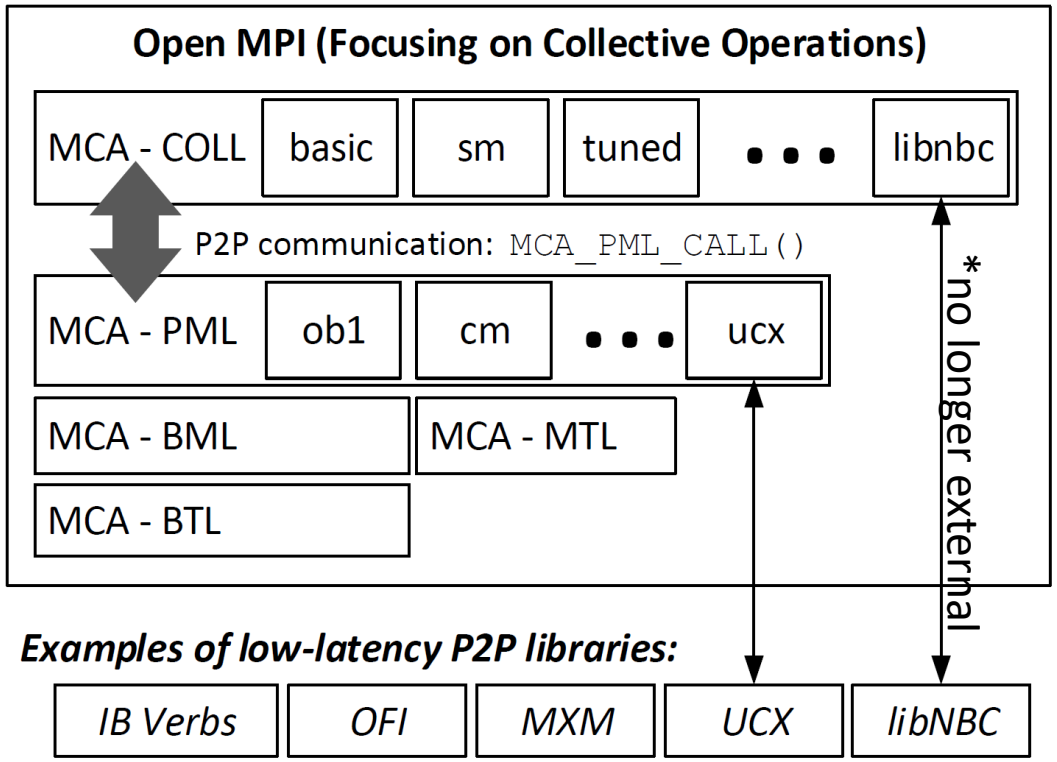
- Alex Margolin, an HPC software architect/developer.

- I regularly wear three hats:
  1. I'm officially an employee of TOGA NETWORKS , an Israeli company
     *No, we're not on anybody's "entity list", so no unusual legal restrictions ☺
  2. I work on Huawei's HPC software (mostly)
  3. I'm also working on my Ph.D. at HUJI
     *Hebrew University of Jerusalem, Israel

Status Update

- The code:
  - Works (tested on 256 nodes, sees regular CI runs, released in Huawei Cloud).
  - API submitted for upstream (~~#3091~~#3602) - posted on Dec. 2018…
  - Full code is available as reference (#4535)
  - First installment of code is ready for review (#4545)
  - OMPI support ready (#7213) - first CR already done, by Jeff Squires

- The team:
  - 5 engineers in Hangzhou (Huawei, CN), 5 in Haifa (Toga, IL) (no only SW…)
  - Contributor agreement – signed.



Recap: Collective Operations in Open MPI

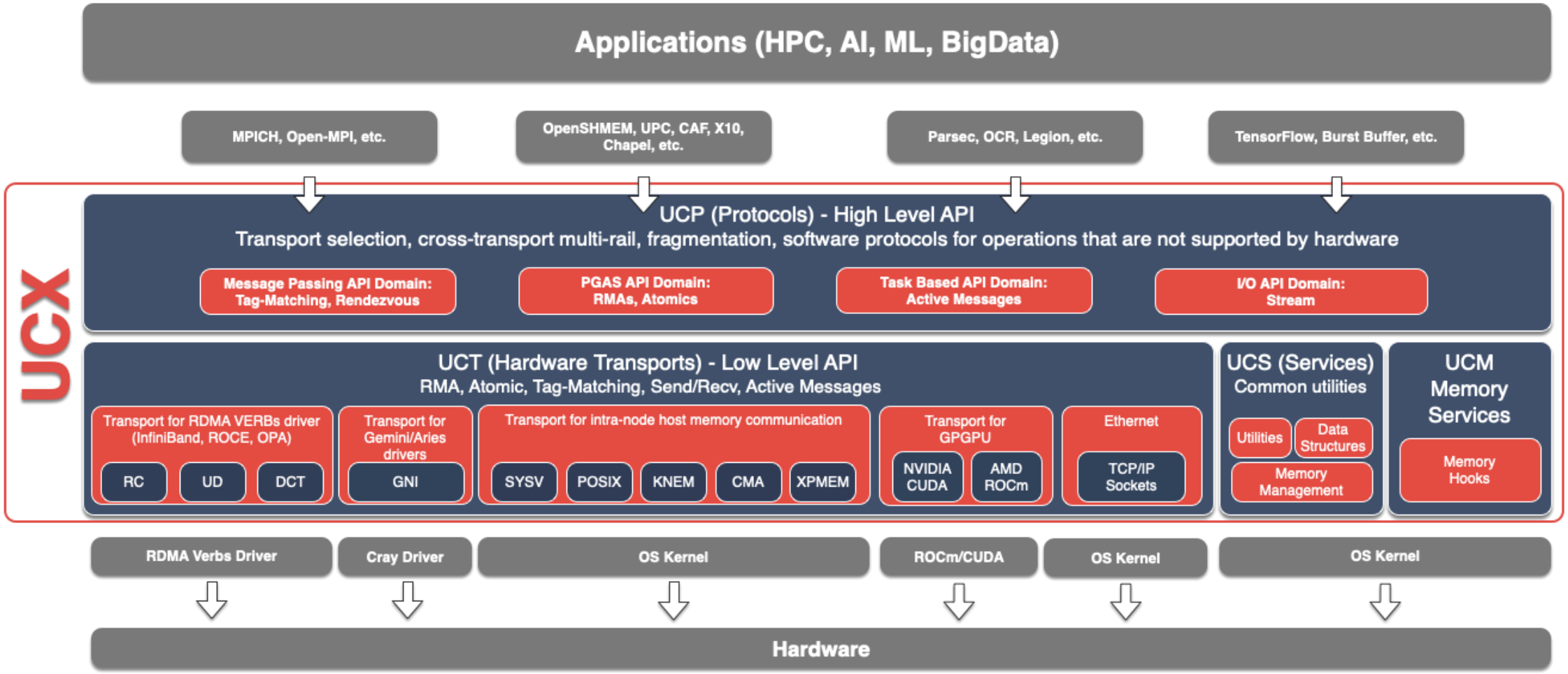
Open MPI is modular, and easily extendable.

Example: P2P module using Open UCX.

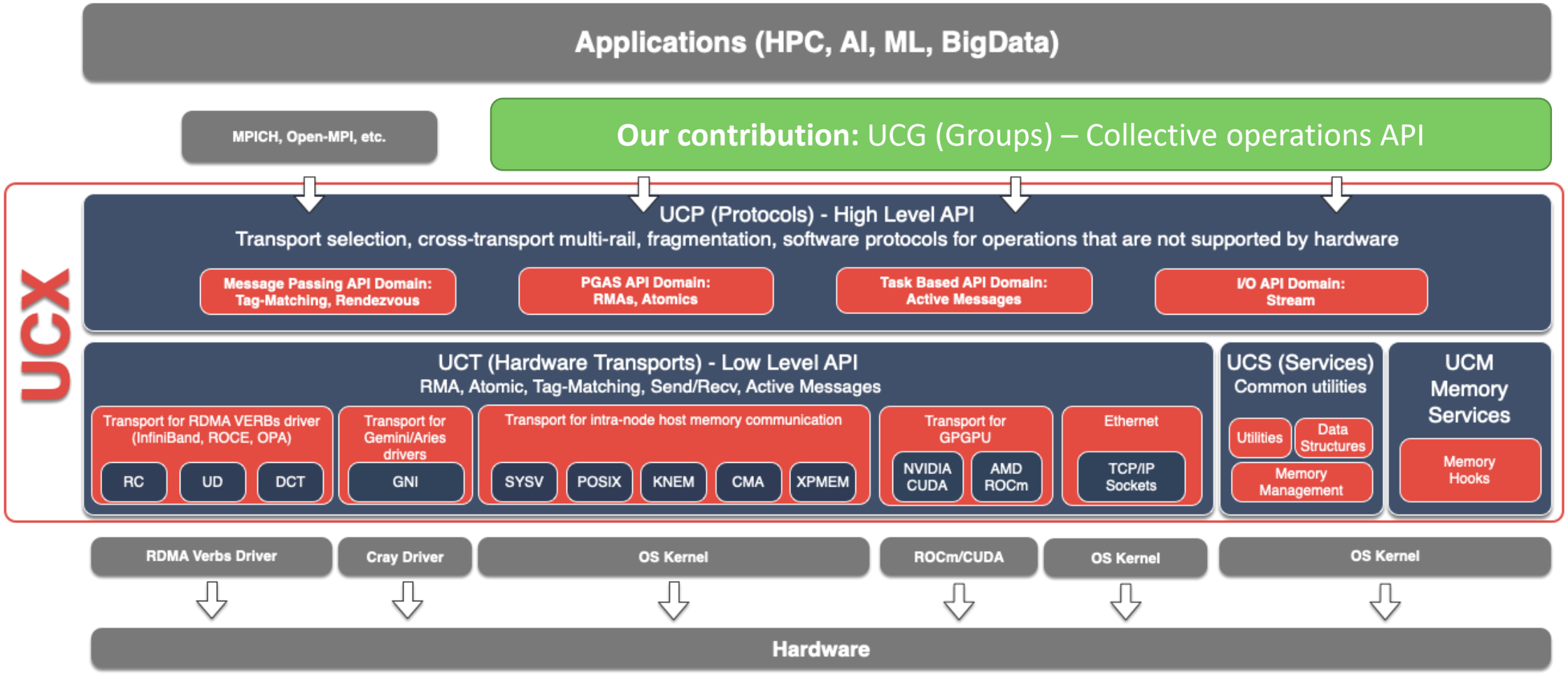
Persistent collective operations (*WIP*):

```
MPI_Bcast_init(buf, ..., &persistent_request);
for (i = 0; i < MAX; i++) {
    /* Some computation */
    MPI_Start(persistent_request);
    MPI_Wait(&persistent_request, &bcast_status);
}
MPI_Request_free(&persistent_request);
```

Open UCX – for P2P and beyond

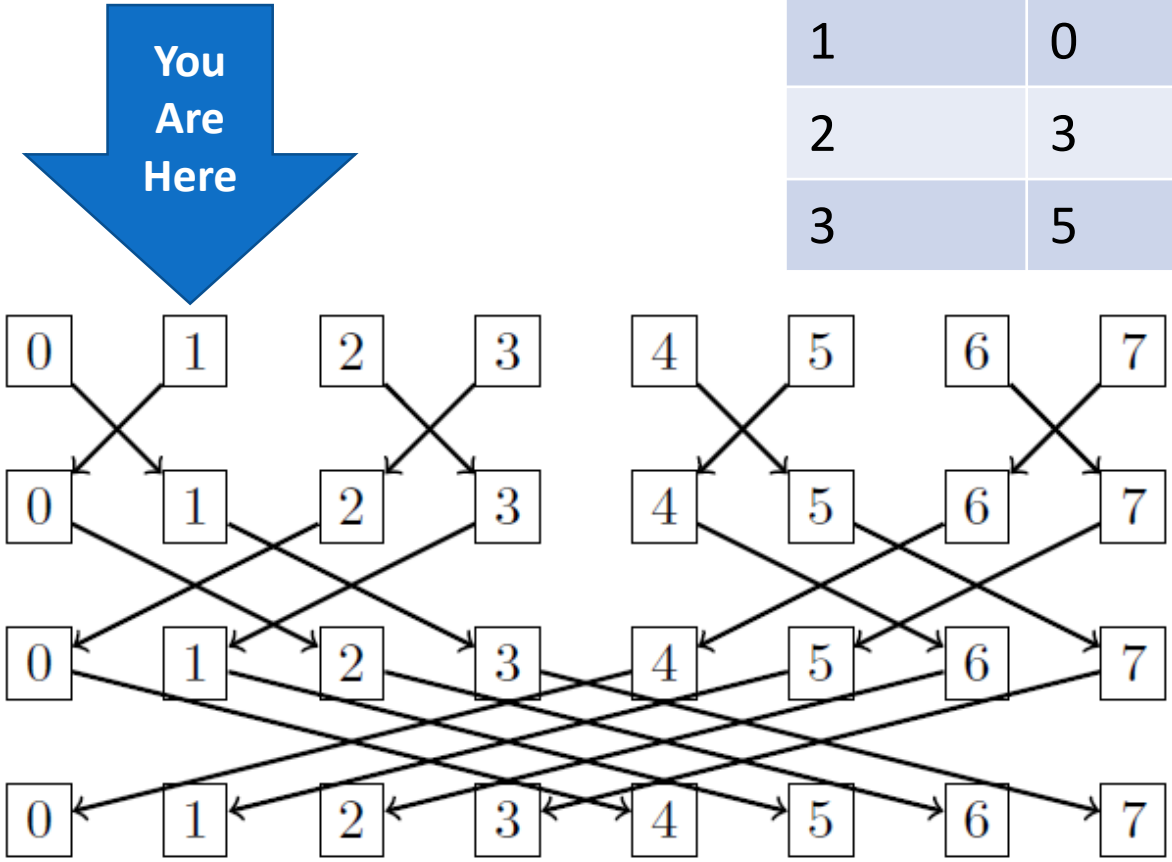


Open UCX – for P2P and beyond



Decoupling Planning and Execution

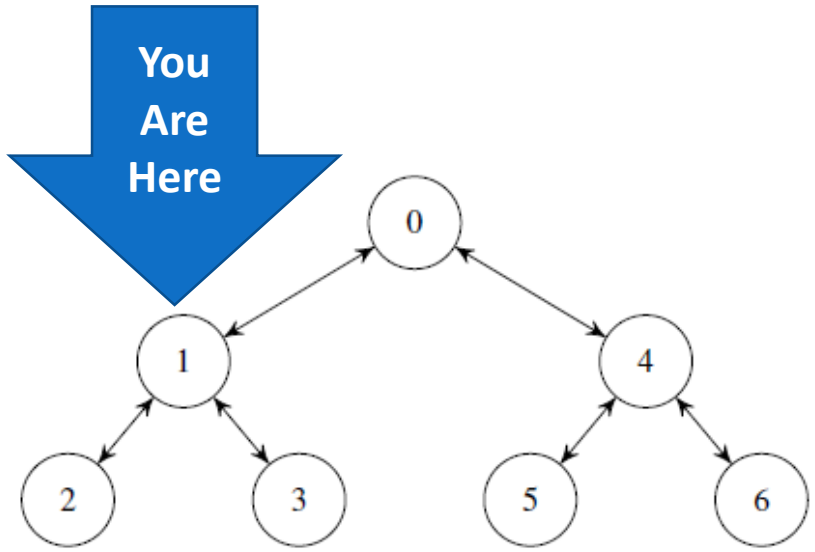
Phase #	Ranks	Action
1	0	Send a message and also wait for one
2	3	Send a message and also wait for one
3	5	Send a message and also wait for one



Decoupling Planning and Execution

Phase #	Ranks	Action
1	2,3	Wait for a message
2	0	Send a message
3	0	Wait for a message
4	2,3	Send a message

Phase #	Ranks	Parameters
1	2,3	**Action:** Wait for a message from each node **Incoming buffer:** temporary buffer of 24 bytes which starts with the user's input **Incoming length:** 16 bytes **Incoming offset:** 8 bytes
2	0	**Action:** Send a message to each node **Outgoing buffer:** the temporary buffer from the previous step **Outgoing length:** 24 bytes **Outgoing offset:** 0 bytes



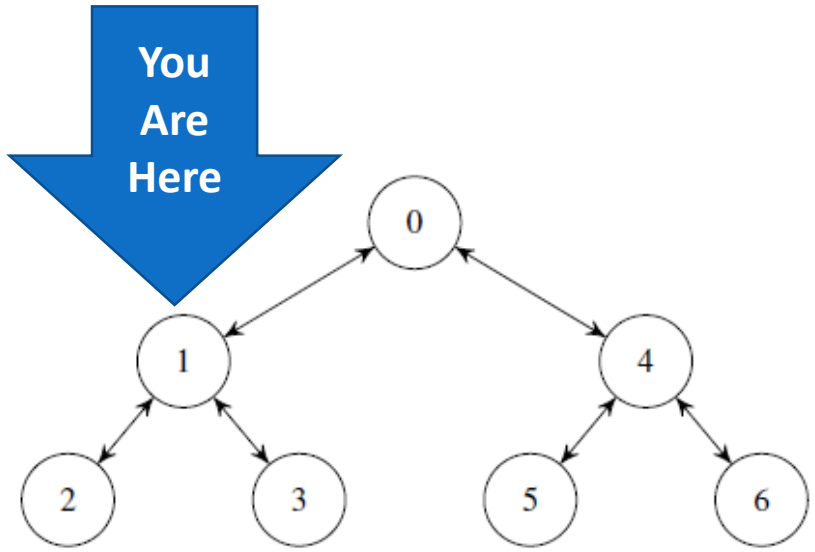
8-byte call to MPI_Gather()

*Is tree the best choice for MPI_Gather? That's another topic…

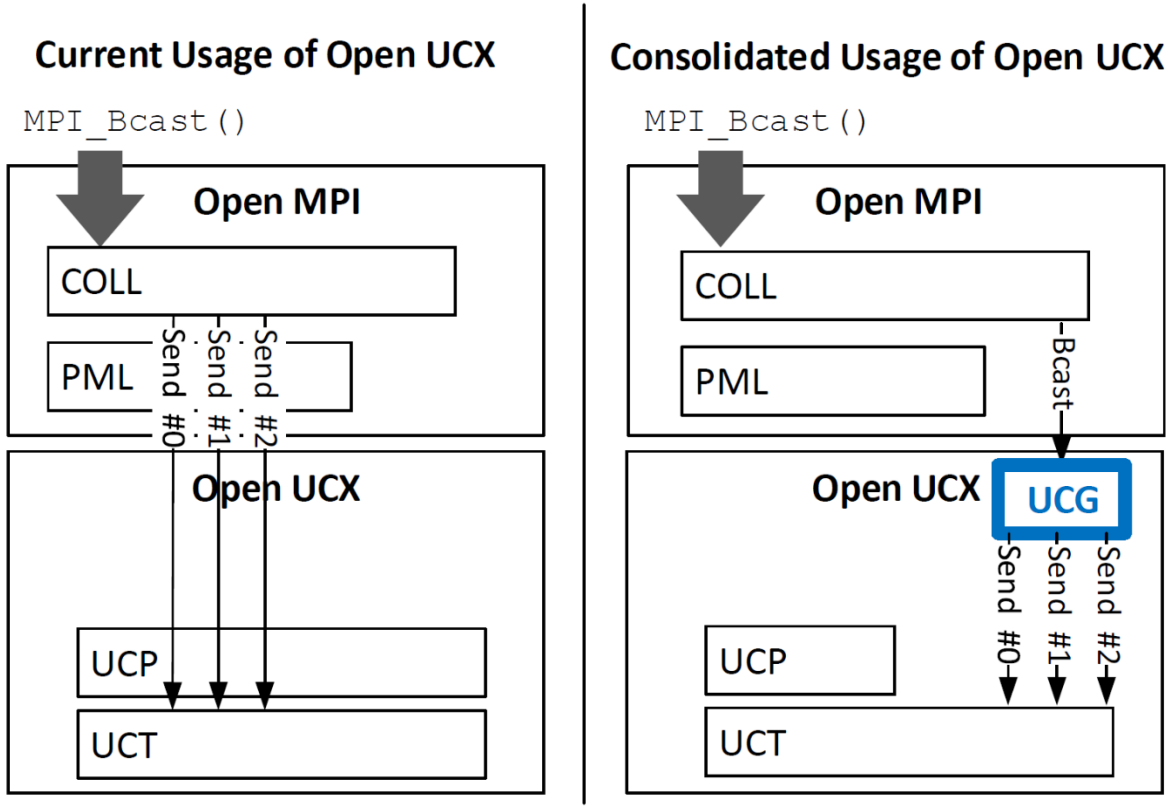
Decoupling Planning and Execution

Phase #	Ranks	Action
1	2,3	Wait for a message
2	0	Send a message
3	0	Wait for a message
4	2,3	Send a message

Phase #	Ranks	Parameters
3	0	**Action:** Wait for a message from each node **Incoming buffer:** destination buffer **Incoming length:** 8 bytes **Incoming offset:** 0 bytes
4	2,3	**Action:** Send a message to each node **Outgoing buffer:** destination buffer **Outgoing length:** 8 bytes **Outgoing offset:** 0 bytes



8-byte call to MPI_Bcast()



Consolidating decisions on P2P communication

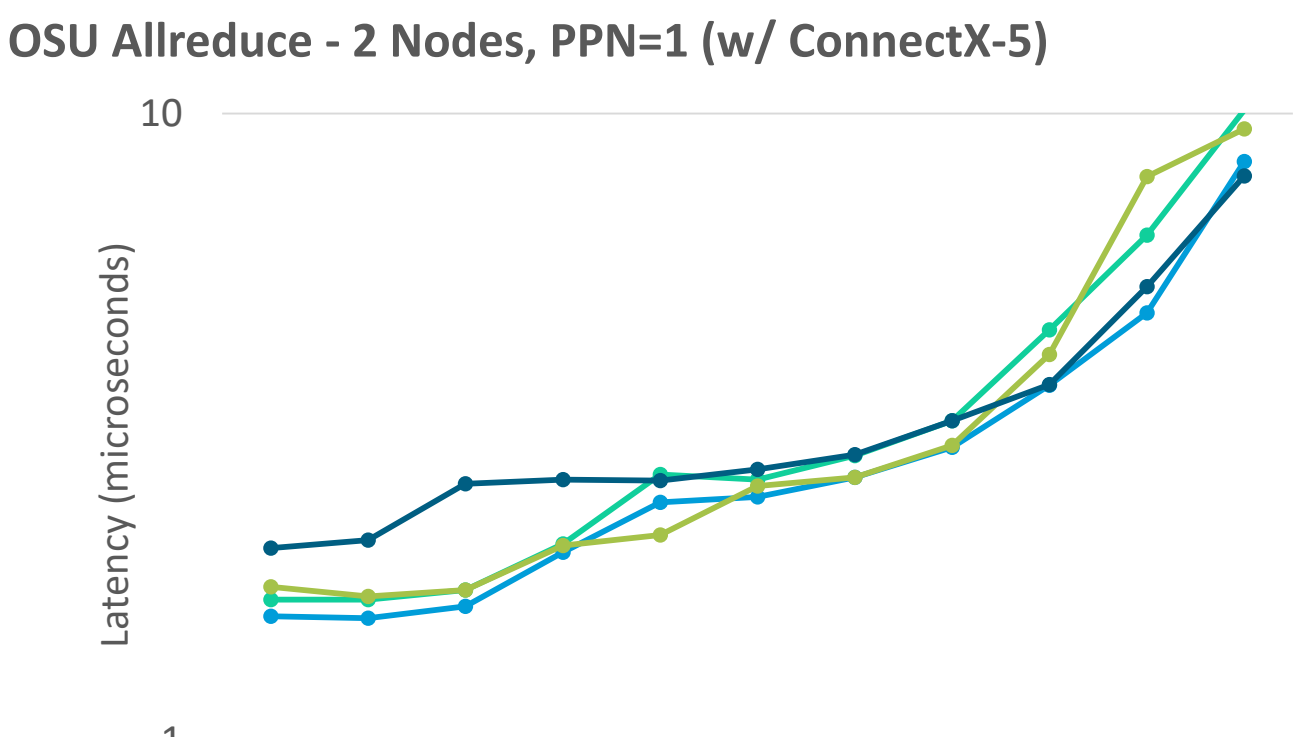
Open UCX, as most P2P libraries, contains considerable logic on how best to send data.

During collective operations, messages of the same size are often sent consecutively.

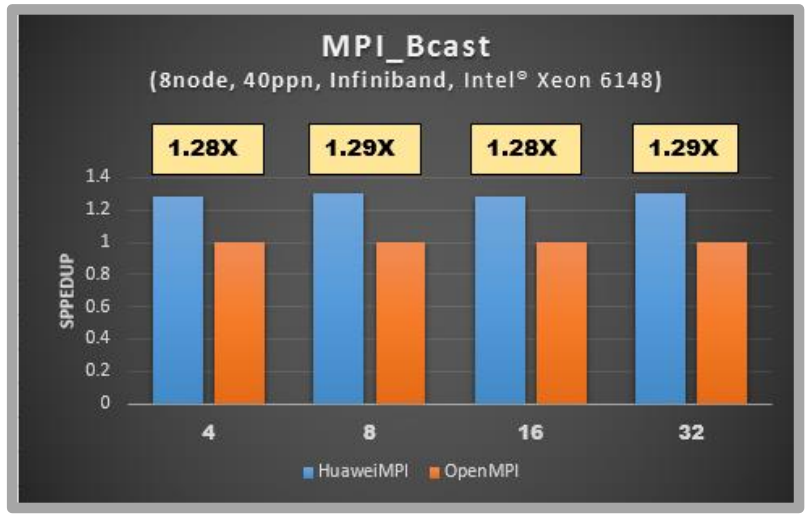
This makes most of the P2P logic redundant.

By consolidating the per-message logic, making it per-collective, we can save latency.

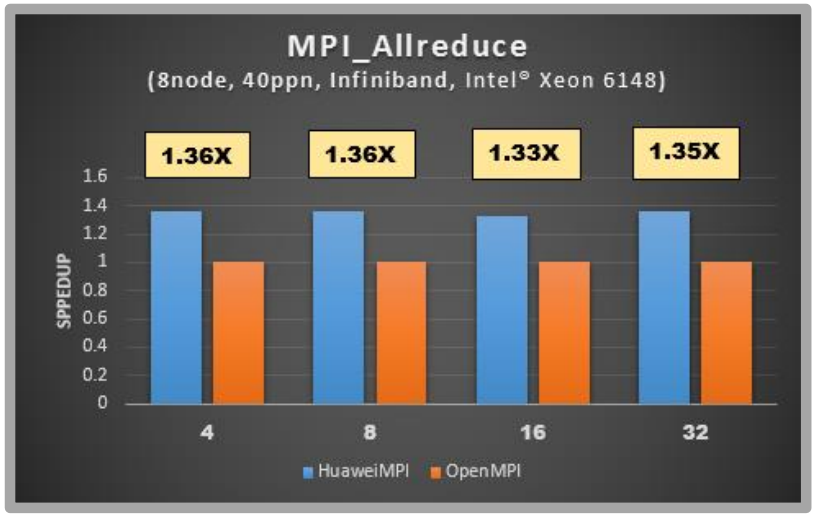
Feature Overview: Low-latency out-of-the-box

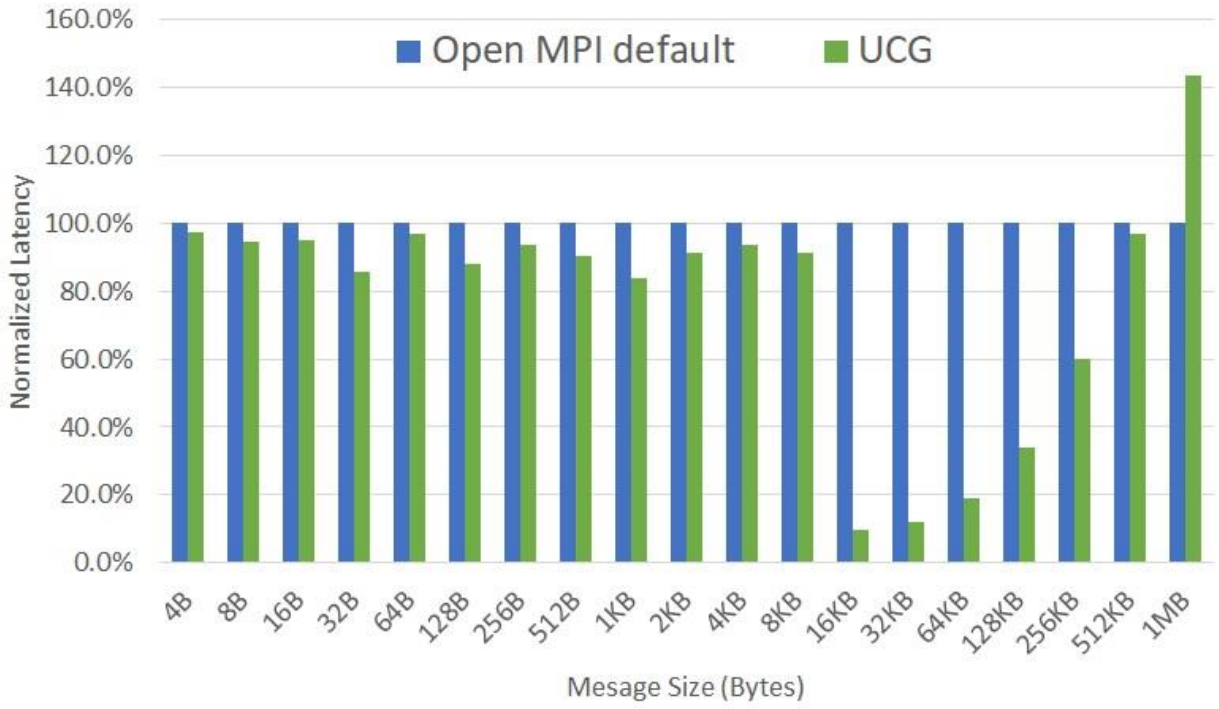


	8	16	32	64	128	256	512	1024	2048	4096	8192
Huawei MPI (GCC)	1.58	1.57	1.64	2	2.4	2.45	2.63	2.94	3.69	4.81	8.39
Open MPI (GCC)	1.68	1.68	1.74	2.06	2.66	2.61	2.85	3.24	4.52	6.4	10.13
HPCX (Mellanox's OMPI)	1.76	1.7	1.74	2.05	2.13	2.55	2.63	2.96	4.13	7.94	9.45
IMPI (Intel's MPI)	2.03	2.09	2.57	2.61	2.6	2.71	2.86	3.24	3.7	5.3	7.96



Huawei MPI vs. OpenMPI (8 x 40ppn)





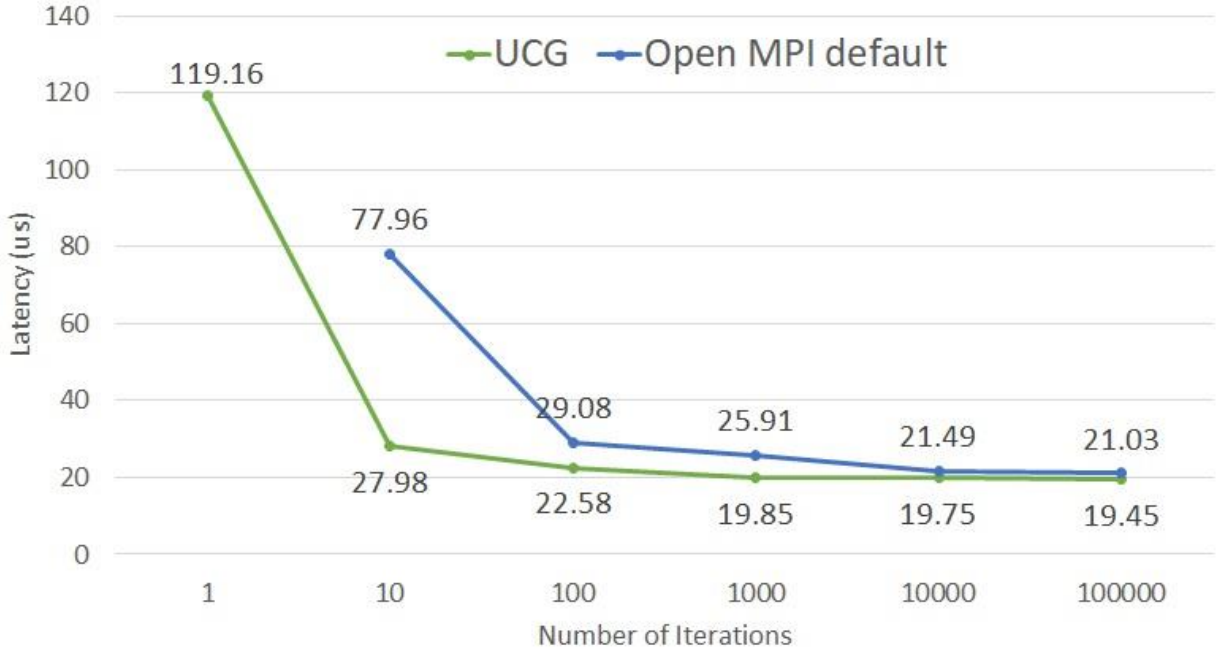
Latency: the average case

For message size <16KB, OSU (Allreduce) shows a steady ~8% latency reduction.

For larger messages – the latency shows linear increase. For some sizes – up to 90%* increase over Open MPI default*.

* Reason: better threshold selection, and a missing *rendezvous* implementation.

** MPI default used "*tuned*" for collectives, over "*ucx*" for P2P.



Latency: convergence

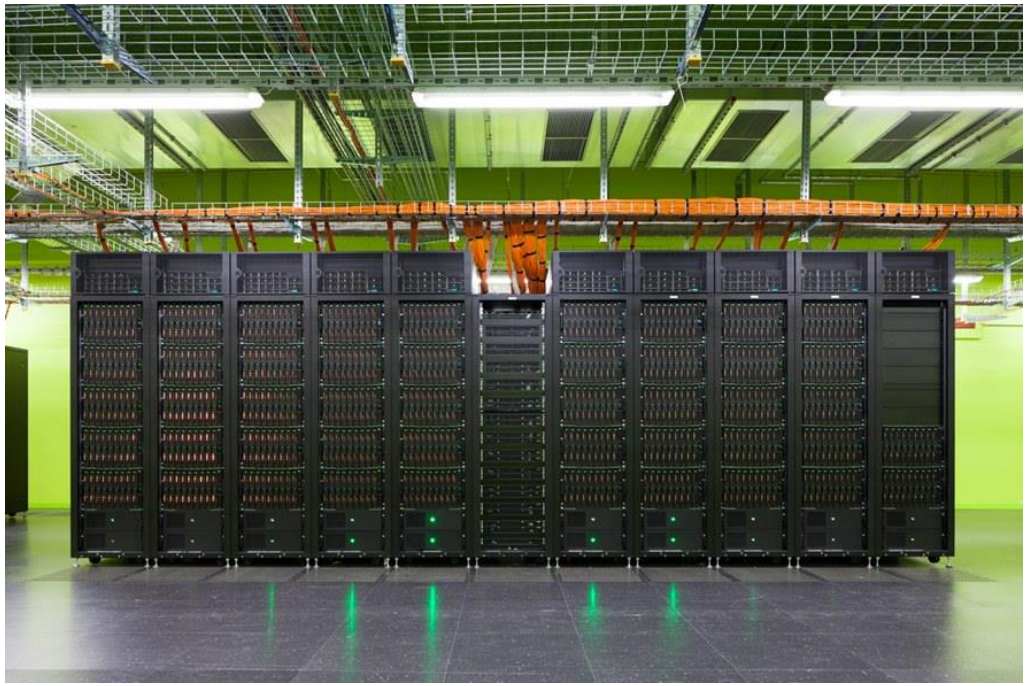
Q: Amortized latency is not the full picture: what happens if a collective is called only once?

A: The latency is considerably higher, but the same is true for most implementations.

Q: Where is the “one iteration” data-point?

A: exceedingly high (9ms), caused by on-demand connection establishment and consequent imbalance...

Huawei MPI vs. Open MPI (256 servers, x86)



The table shows the latency (microseconds) of Allreduce with increasing message sizes (Bytes), on 256 nodes (*Intel Xeon E5-2680* CPUs @2.5GHz, Connect-X3 NIC).

Length (bytes)	**Huawei MPI** - Inline Sends		**Huawei MPI** - Buffer Copy		**Open MPI** Latency (us)
	Latency (us)	Improvement	Latency (us)	Improvement	
16	19.5	6.7%			20.9
32	22.78	1.3%			23.08
64	21.57	8.5%			23.58
128	25.79	8.1%			28.07
256			24.95	6.6%	26.7
512			28	9.9%	31.06
1024			29.66	16.0%	35.33
2048			36.03	8.7%	39.47
4096			47.21	6.4%	50.43
8192			66.68	8.7%	73.02
16384			89.07	90.4%	923.65
32768			135.4	88.2%	1142.66
65536			227.83	81.2%	1210.49
131072			442.31	66.1%	1302.86
262144			905.43	39.8%	1503.8
524288			1823.7	2.9%	1878.85

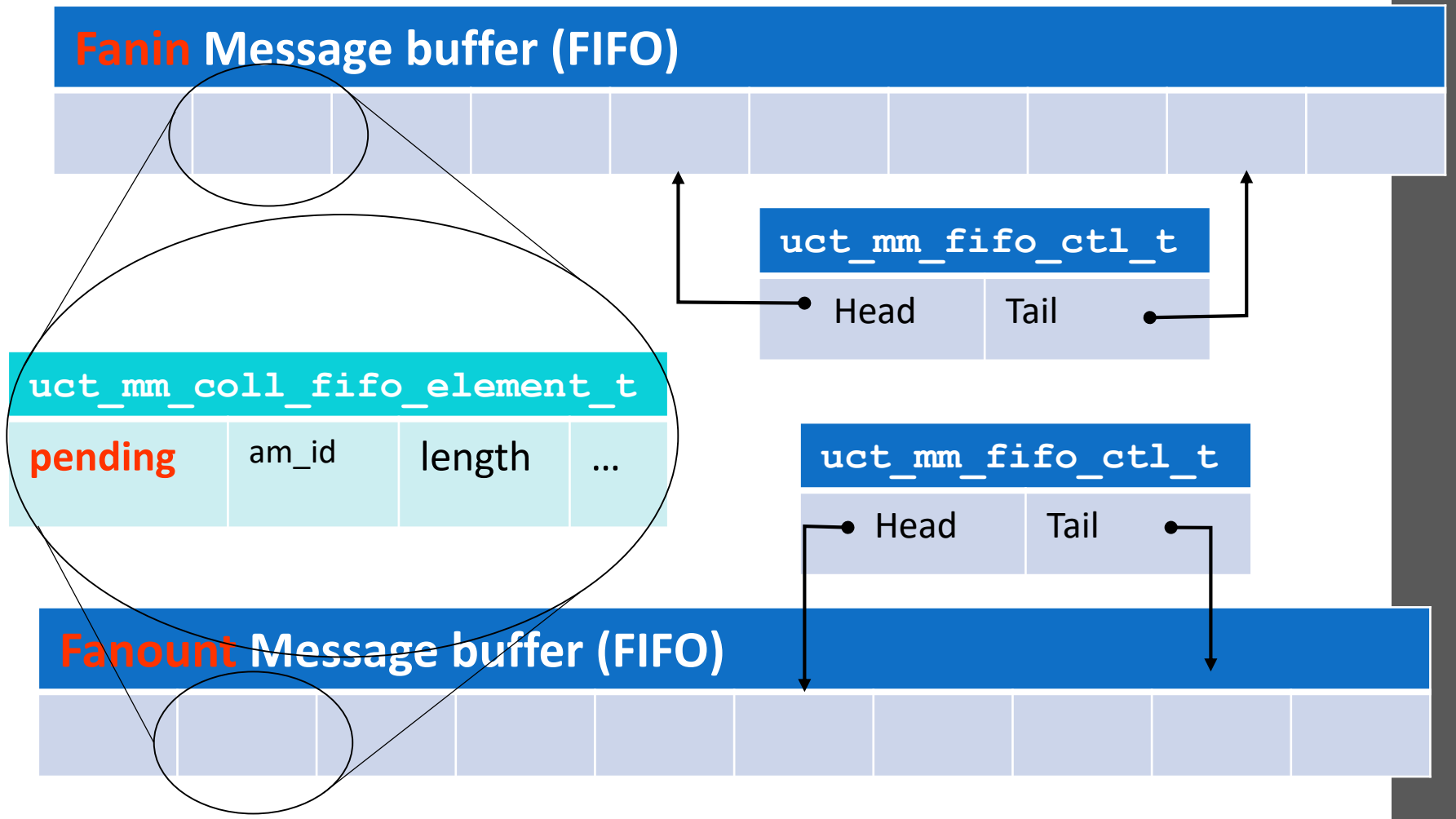
Feature Overview: Shared-memory Collectives

4 queues needed:

1+2. The existing P2P queues, for control messages (e.g. Rendezvous).

3. Fanin, for collectives like reduce or gather.

4. Fanout, for collectives like bcast and scatter.

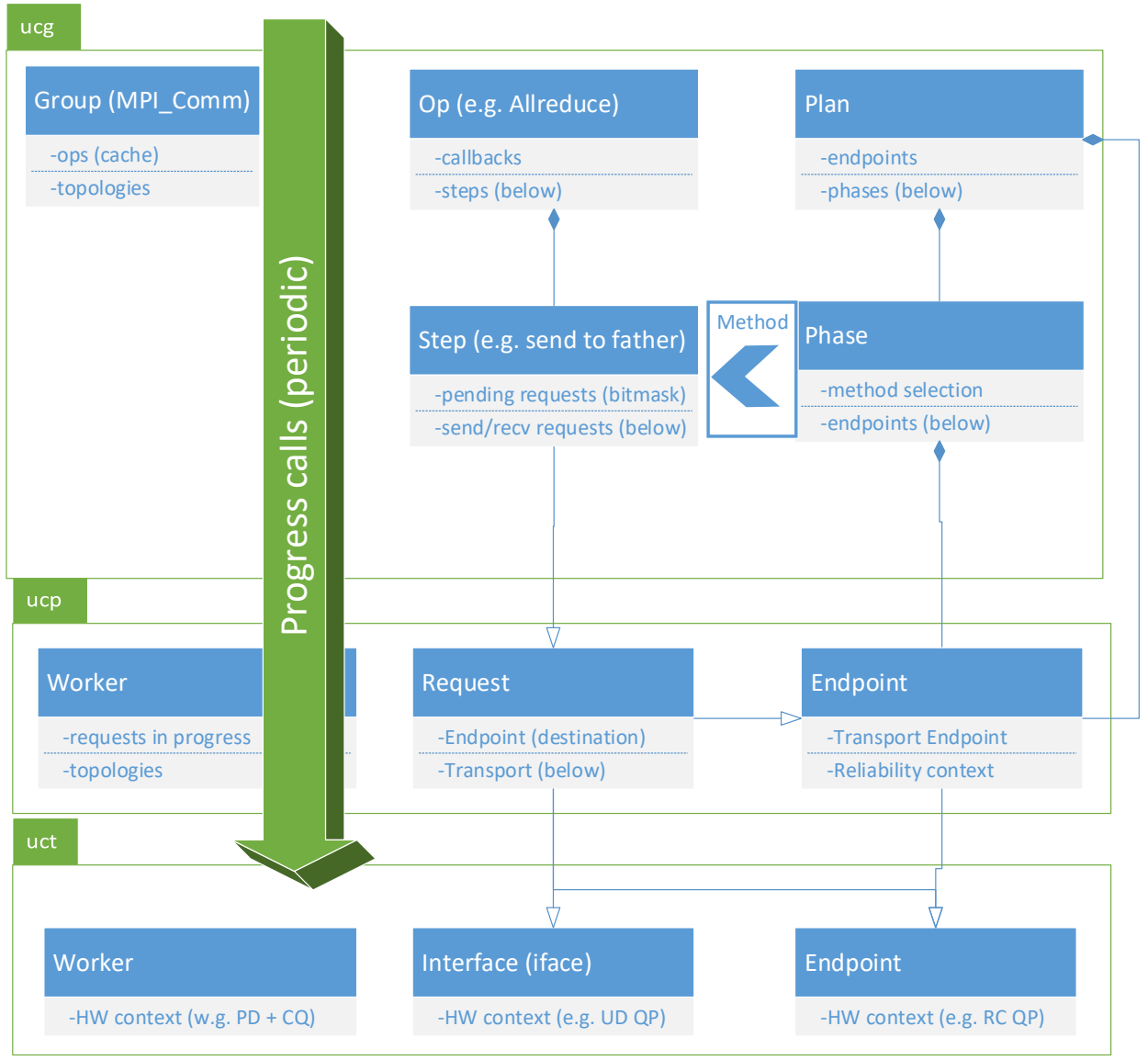


Feature Overview: Persistent Collectives

Algorithm 1: Cached operation lookup algorithm

```
1  op* find_cached_operation(Communicator,
     OperationType, OperationArgumets)
       Input: Type and parameters for this operation.
       Output: an operation object - ready to be executed.
2      if OperationType ∉ Communicator.Plans then
           /* No suitable plan - create one   */
3          Plan ← create_plan(OperationType,
             OperationArgumets);
4      end
5      else
           /* Reuse an existing plan          */
6          Plan ← Communicator.Plans[OperationType];
7      end
8      foreach CachedOp ∈ Plan.OpCache do
9          CachedArguments ← CachedOp.Arguments);
           /* Compare the arguments to those of
              a cached previous operation     */
10         if CachedArguments = OperationArgumets
             then
               /* Match to a cached operation  */
11             remove_from_cache(CachedOp);
12             return CachedOp;
13         end
14     end
       /* Create an operation, given a plan */
15     Op ← create_operation(Plan, OperationArgumets);
16     return Op;
```

[1]*RDMA-based Library for Collective Operations in MPI*, Alex Margolin and Amnon Barak, ExaMPI'19



APIs

- “Northbound”:
  - **Generic** (*src/ucg/api/ucg.h*): `ucg_group_create, ucg_collective_create`
  - **MPI-specific** (*src/ucg/api/ucg_mpi.h*): `MPI_Reduce(), MPI_SUM`
  - **Versioning** (*src/ucg/api/ucg_version.h*): similar to ucp_version.h

- “Southbound”:
  - **Plan Components** (*src/ucg/api/ucg_plan_components.h*):
    - for accommodating multiple implementations (similar to OMPI’s MCA COLL)
  - Expected components:“*builtin*” (day-1), SHARP, hcoll, NCCL, etc.

Northbound API, side-by-side (Blocking)

MPI Application (or MPI-specific API)	UCX Groups (UCG) Equivalent (Generic API)

MPI_Bcast(bufA,...,rowcomm)	paramsA.buf = bufA;
	paramsA.type.modifiers = BCAST_MODIFIERS;
	...
	
	ucg_collective_create(rowGroup, &paramsA, &collA);
	ucg_collective_start_nb(collA, &reqA);
	
	while (UCS_INPROGRESS ==
	ucg_request_check_status(reqA)) {
	ucg_group_progress(rowGroup);
	}
	
	ucg_request_free(reqA);

Northbound API, side-by-side (Non-blocking)

MPI Application (or MPI-specific API)	UCX Groups (UCG) Equivalent (Generic API)

`MPI_Ibcast_init(..., &req);`
 	
`for (i=0; i<MAXITER; i++) {`
`  compute(buf);`
 	
`  MPI_Start(req);`
 	
`  MPI_Wait(req, &status);`
 	
`}`
 	
`MPI_Request_free(req);`
	`ucg_collective_create(Group, ..., &coll);`
	
	`for (i=0; i<MAXITER; i++) {`
	`  compute(buf);`
	
	`  ucg_collective_start_nb(coll, &req);`
	
	`  while (UCS_INPROGRESS ==`
	`  ucg_request_check_status(req)) {`
	`  ucg_group_progress(Group);`
	`  }`
	`}`
	
	`ucg_request_free(reqA);`

What’s next?

1. Upstreaming
2. Work together to make other components available
3. More collectives and algorithms, topology awareness, use PUT/GET
4. Some ARM-specific work (Next talk, if we got time left ☺)