UCF Unified Communication Framework

ISC 2021 BoF Session

Gilad Shainer, Pavel Shamis, Steve Poole, Yossi Itigin, Manju G. Venkata, Gil Bloch

ISC 2021

Unified Communication Framework (UCF) Consortium

MISSION: Collaboration between industry, laboratories, and academia to create production grade communication frameworks and open standards for data centric, ML/AI, and high-performance applications

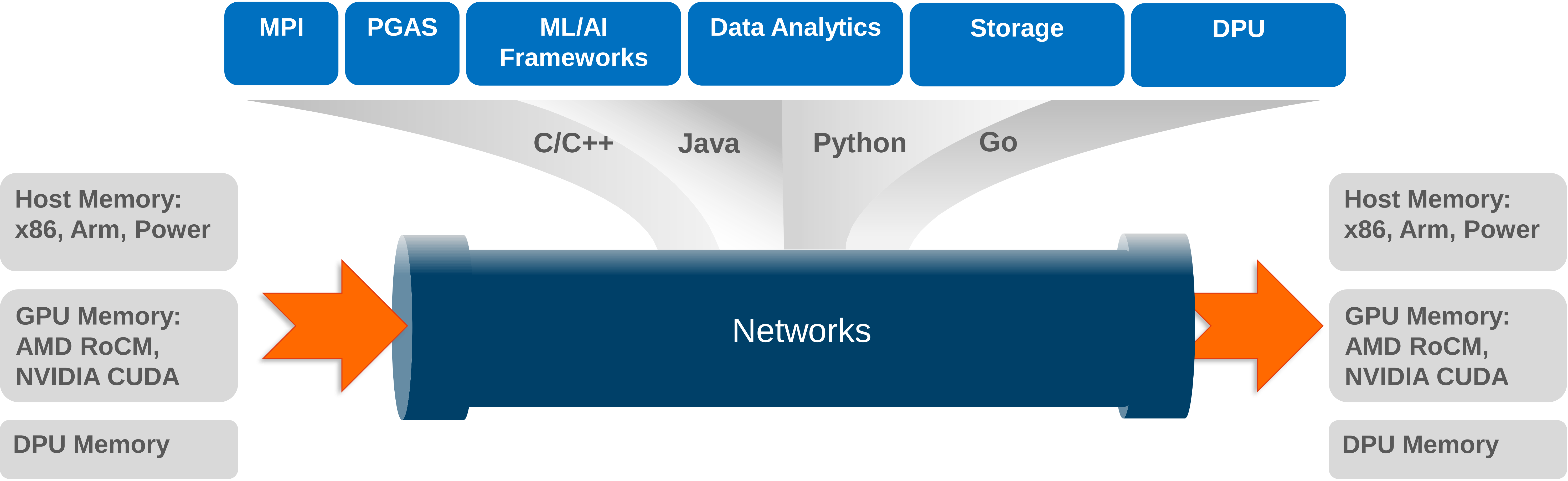
- Projects & Working Groups
  - **UCX – Unified Communication X – www.openucx.org**
  - **UCC – Collective Library**
  - **OpenSNAPI – Smart network Project**
  - SparkUCX – www.sparkucx.org
  - UCD – Advanced Datatype Engine
  - HPCA Benchmark – Benchmarking Effort

- Board members
  - **Jeff Kuehn**, UCF Chairman (Los Alamos National Laboratory)
  - **Gilad Shainer**, UCF President (Nvidia)
  - **Pavel Shamis**, UCF Treasurer (Arm)
  - **Brad Benton**, Board Member (AMD)
  - **Yanfei Guo**, Board Member (Argonne National Laboratory)
  - **Perry Schmidt**, Board Member (IBM)
  - **Dhabaleswar K. (DK) Panda**, Board Member (Ohio State University)
  - **Steve Poole**, Board Member (Open Source Software Solutions)

Join

https://www.ucfconsortium.org or info@ucfconsortium.org

Why UCX ?



High-Performance Universal Data Mover

UCX Users

- MPI implementations: MPICH, Open MPI, NVIDIA HPC-X, Huawei MPI
- PGAS: GasNET
- OpenSHMEM: OSSS SHMEM, Sandia SHMEM, Open MPI SHMEM
- Charm++
- RAPIDS / DASK
- Spark UCX
- NVIDIA NCCL

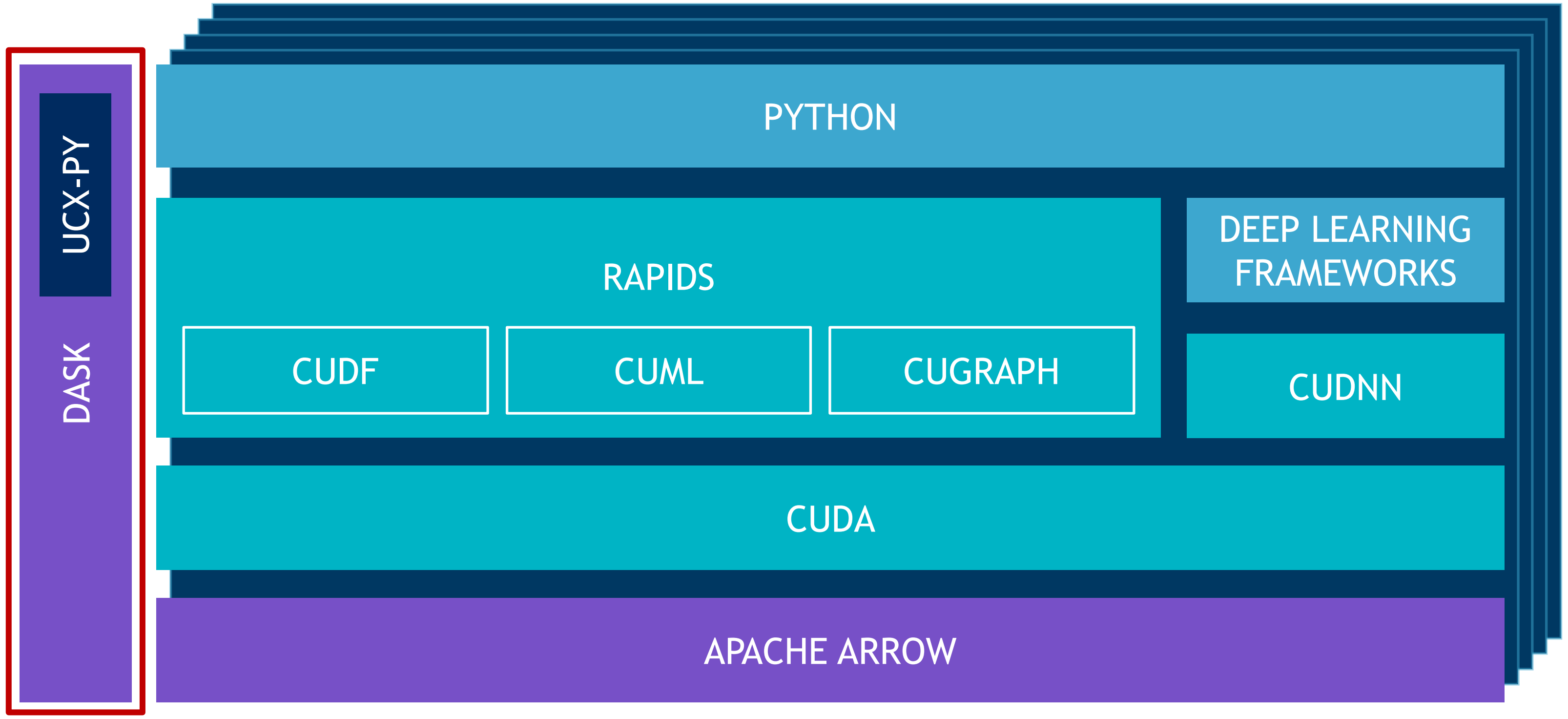


Diagram courtesy of NVIDIA

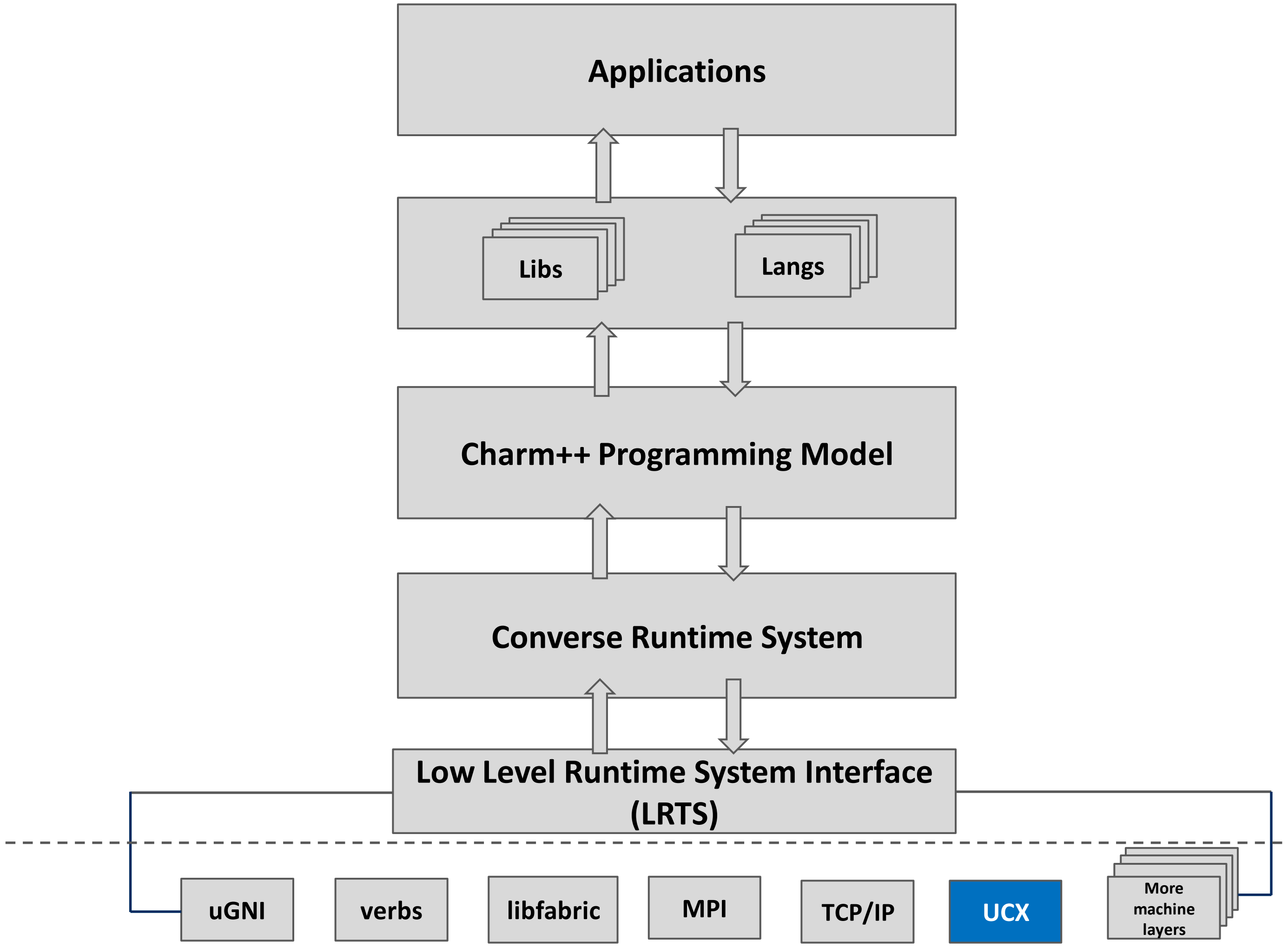


Diagram courtesy of Nitin Bhat @ Charmworks Inc

Content

UCX – Unified Communication X
Web https://www.openucx.org
Git https://github.com/openucx/ucx
Docs https://openucx.readthedocs.io
Mailing list https://elist.ornl.gov/mailman/listinfo/ucx-group

UCC - Collective Communication API
Web https://www.ucfconsortium.org/projects/ucc/
Git https://github.com/openucx/ucc
Git https://github.com/openucx/ucc_spe
Git https://github.com/openucx/torch-ucc

API for Smart Network (DPU) programmability
Web https://www.ucfconsortium.org/projects/opensnapi/
GIT https://github.com/openucx/shmem-opensnapi

info@ucfconsortium.org

Open source framework for high-performance networks

UCX Latest Development

- UCP extensible datapath API – “nbx”
  - Mandatory parameters – passed as regular arguments
  - Optional/extended parameters – passed in a struct “ucp_request_param_t”
  - Allows future extensions without ABI/API breakage, and still enjoy fastcall

- UCP active messages – generic high-level communication primitive
  - Added rendezvous protocol, which uses zero-copy RDMA for bulk transfer
  - Support GPU memory for all operation types

```
ucs_status_ptr_t ucp_am_send_nbx(ucp_ep_h ep, unsigned id,
                                 const void *header, size_t header_length,
                                 const void *buffer, size_t count,
                                 const ucp_request_param_t *param);
```

UCX Latest Development – Cont.

- New client-server connection protocol
  - Quick one-sided disconnect with remote notification (like TCP)
  - Multi-device and multi-path
  - Revamp RDMA_CM and TCP connection managers for better stability
- GPU support improvements
  - Select NIC according to GPU locality on the PCIe bus
  - Support statically-linked Cuda applications
  - Global cache for Cuda IPC remote memory handles
- Error handling improvements
  - Keepalive on UCP layer to detect stale peers
  - Auto-revoke all queued requests when connection is closed

UCX Latest Development – Cont.

- Global configuration file to set UCX parameters
- Shared memory to support asynchronous wakeup
- UD performance optimizations
- Java bindings – full support for UCP API

UCX Roadmap

- Release schedule:
  - v1.11: August 2021
  - v1.12: December 2021
  - v1.13: March 2022 (Tentative)
- Wire protocol compatibility
- SRD support for AWS systems
- Rendezvous protocol with scatter-gather lists
- Golang bindings
- UCP active message improvements
  - Set receive buffer alignment
  - Fragmented receive protocol
- One-sided improvements:
  - Support GPU atomic operations (both on source and target)
  - Multi rail and out-of-order with PUT/FENCE

Unified Collective Communication

Open-source project to provide an API and library implementation of collective (group) communication operations

UCC Design Challenges

- Unified collective stack for HPC and DL/ML workloads
  - Need to support a wide variety of semantics
  - Need to optimize for different performance sensitives - latency, bandwidth, throughput
  - Need for flexible resource scheduling and ordering model
- Unified collective stack for software and hardware transports
  - Need for complex resource management - scheduling, sharing, and exhaustion
  - Need to support multiple semantic differences – reliability, completion
- Unify parallelism and concurrency
  - Concurrency – progress of a collective and the computation
  - Parallelism – progress of many independent collectives
- Unify execution models for CPU, GPU, and DPU collectives
  - Two-way execution model – control operations are tightly integrated
    - Do active progress, returns values, errors, and callbacks with less overhead
  - One-way execution model – control operations are loosely integrated
    - passive progress, and handle return values (GPU/DPUs)

UCC Design Principles: Properties we want

- Scalability and performance for key use-cases
  - Enable efficient implementation for common cases in MPI, OpenSHMEM and AI/ML
- Extensible
  - We cannot possibly cover all the options and features for all use cases
  - We need the API and semantics that is modular
- Opt in-and-out
  - If for a certain path some semantic is not applicable, we need a way to opt-out
- Explicit API and semantics over implicit
  - Explicit -> implicit is easier than implicit -> explicit
- Minimal API surface area
  - Lessen the mental load
  - A few set of abstractions to understand and go into details when required
- Other properties are such as the ability to override functionality, programmability, expressing general and specific functionality are important

UCC's Solution

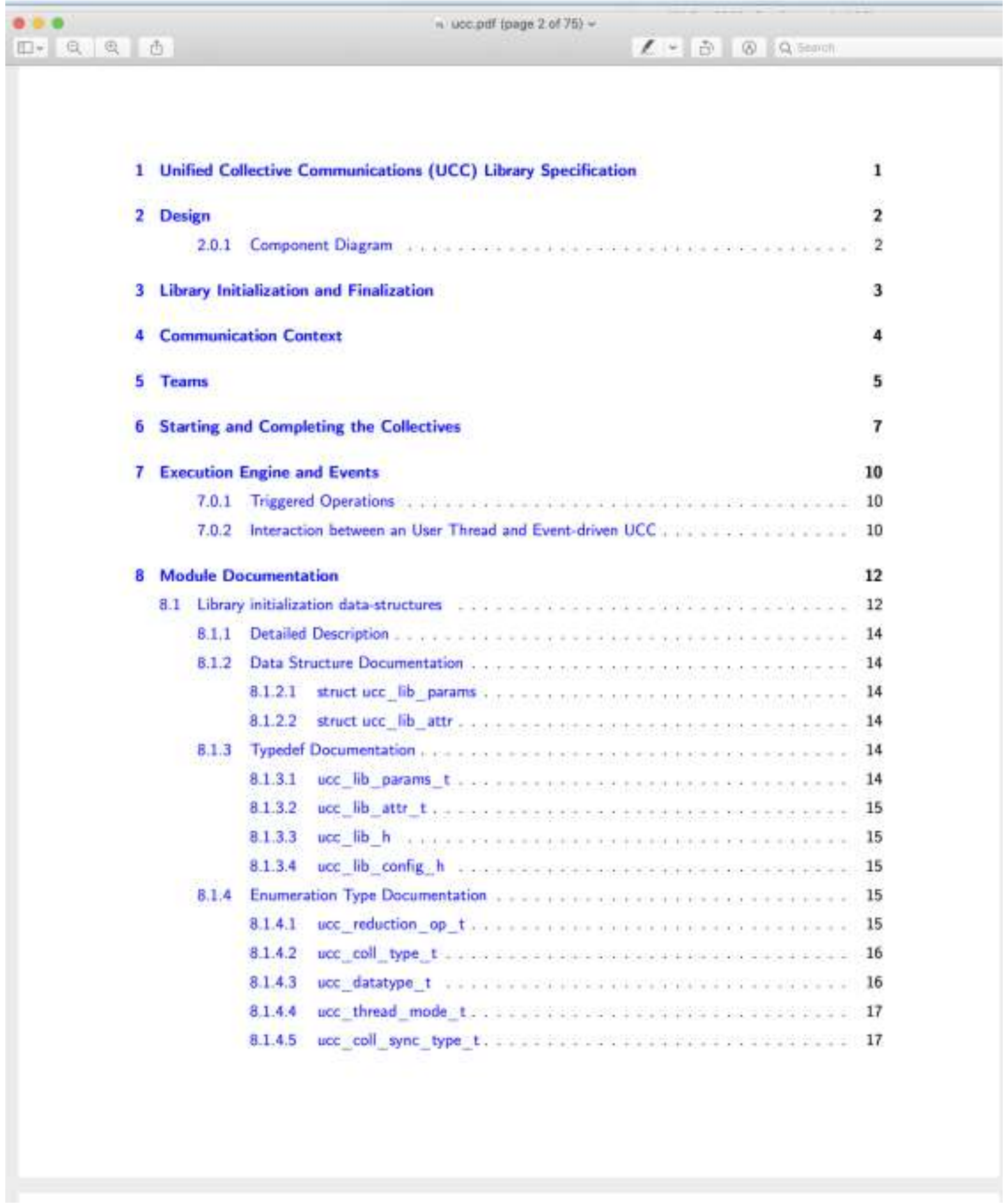
- Abstractions
  - Abstract the resources required for collective operations
  - Local: Library, Context, Endpoints
  - Global: Teams
- Operations
  - Create/modify/destroy the resources
  - Build, launch and finalize collectives
- Properties
  - Explicit way to request for optional features, semantics, and optimizations (opt-in or opt-out model)
  - Provides an ability to express and request many cross-cutting features
  - Properties are preferences expressed by the user of the library and what the library provides is queried

UCC’s Concepts

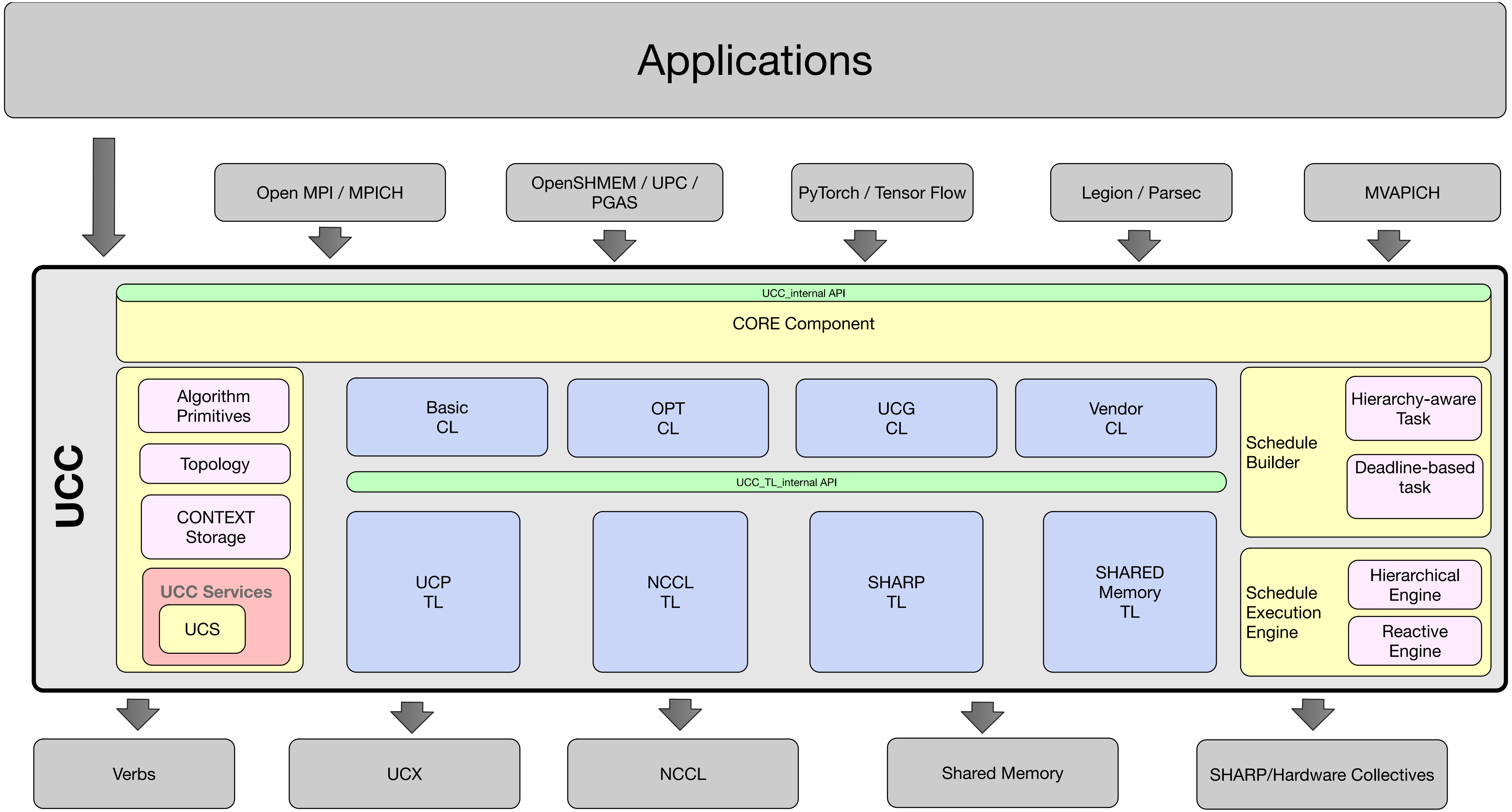
- Abstractions
  - Collective Library
  - Contexts
  - Teams
  - Endpoints
- Operations
  - Create, and destroy the abstractions
  - Post collective operations
  - Triggered post operations
- Details of concepts
  - Code: https://github.com/openucx/ucc
  - Slides: https://github.com/manjugv/ucc_wg_public

UCC Specification: Interfaces and semantics fully specified

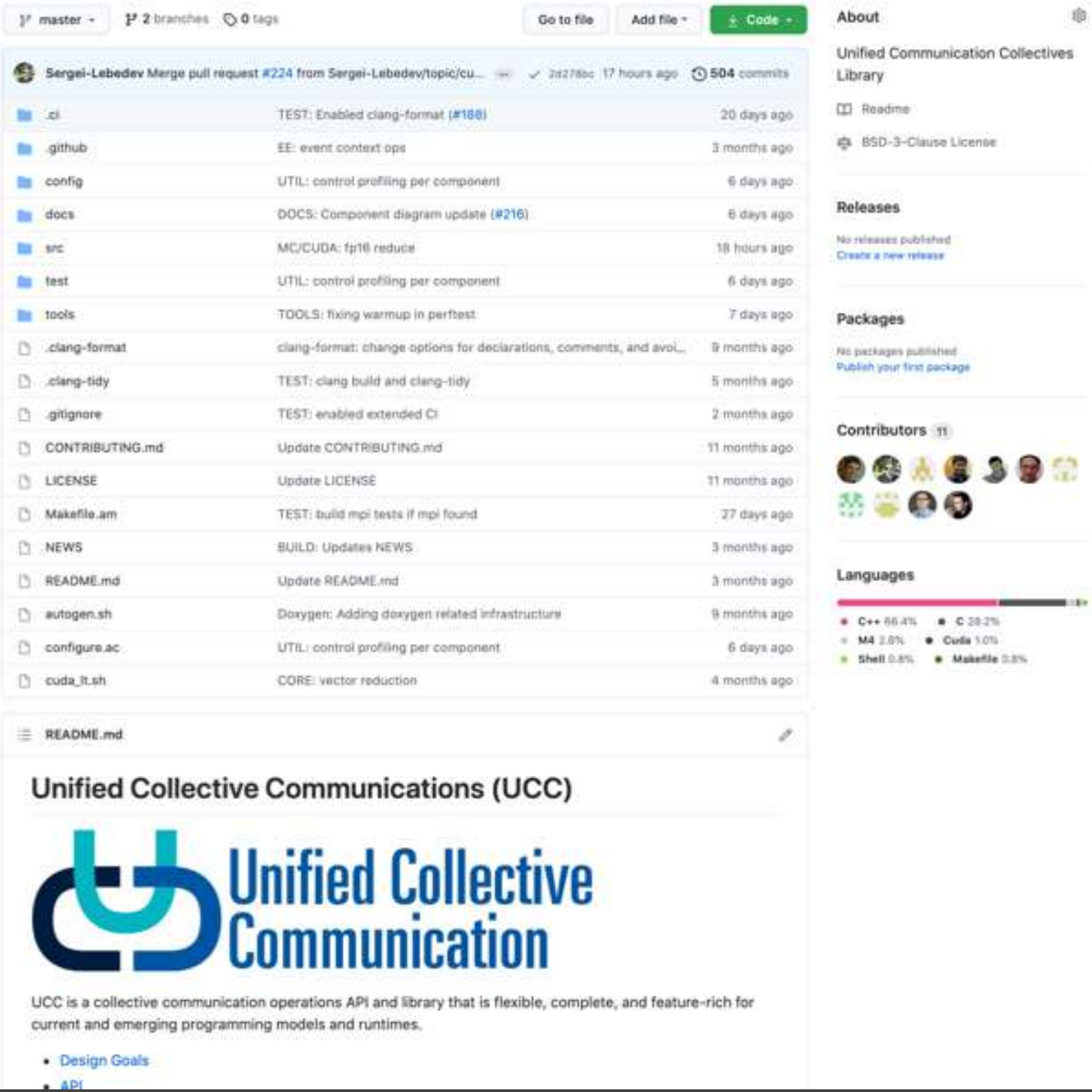
- Specification available on the UCC GH
- Specification is ahead of the code now
- The version 1.0 is agreed by the working group and merged into the master branch
- Over 75 pages of detailed information about the interfaces and semantics
- Doxygen based documentation
- Both pdf and html available



UCC Reference Implementation: Component Diagram



UCC: Reference Implementation Status



UCC v1.0 Expected to Release Q3 2021

- v0.1.0 Early Release (Branched Q1 2021)
  - Support for most collectives required by parallel programming models
  - Many algorithms to support various data sizes, types, and system configurations
  - Support for CPU and GPU collectives
  - Testing infrastructure
    - Unit tests, profiling, and performance tests
  - Support for MPI and PyTorch (via Third-party plugin)
- Expected July 31st.

- v1.0 Stable Release (Expected SC 2021)
  - Incorporate feedback from v0.1.0 release
  - Support for OpenSHMEM with one-sided collectives and active sets
  - Hardware collectives - support for SHARP
  - Support for more optimized collectives (hierarchical/ reactive)
  - Infrastructure for pipelining, task management , and customization (algorithm selection)
  - Persistent collectives

- v1.x Series: Focus on stability, performance and scalability
  - Support for DPUs and DPAs
  - Partitioned collectives
  - OpenSHMEM Teams and nonblocking collectives

Plenty of work : Contributions are Welcome!

- What contributions are welcomed ?
  - Everything from design, documentation, code, testing infrastructure, code reviews …

- How to participate ?
  - WG Meetings : https://github.com/openucx/ucc/wiki/UCF-Collectives-Working-Group
  - GitHUB: https://github.com/openucx/ucc
  - Slack channel: Ask for an invite
  - Mailing list: ucx-group@elist.ornl.gov

Open SNAPI

Open-source, standard application programming interface (API) for accessing compute engines on smart networks

OpenSNAPI

- OpenSNAPI – contributors to this talk
  - Thanks to **Gil Bloch, Gary Grider, Morad Horany, Alex Margolin, Tal Mizrahi, Brad Settlemyer and Brody Williams**
- OpenSNAPI was initiated to define a common, open portable API for smart networks from all vendors and all flavors of smart networks (processor based, FPGA based, ASIC based…)
  - The goal is an API for Computational offload in the network (NIC/DPU/Switch/Storage)
- Benefits of smart network Accelerators
  - Operate on data in place at network edge
    - Minimize data movement
  - Computation of data in-flight
    - Collective/AMO operations
    - Offload kernels into the network
  - Energy efficiency
    - Edge/In-network computing is efficient
    - Arm cores are typically more energy-efficient than x86_64 analogs
    - Reduces operational costs
- Progress
  - Focus on exploring what is possible with smart network-based acceleration
    - Examine feasibility of both computation and communication offloading with scientific applications
  - Utilize smart network devices within SHMEM/MPI as if completely distinct nodes
    - Minimize experimentation overhead
    - Distinct code segments for host and smart network PEs
      - MPMD model

LA-UR-21-26018

OpenSNAPI I/O Offloading Study

- BigSort Benchmark[1] (LLNL)

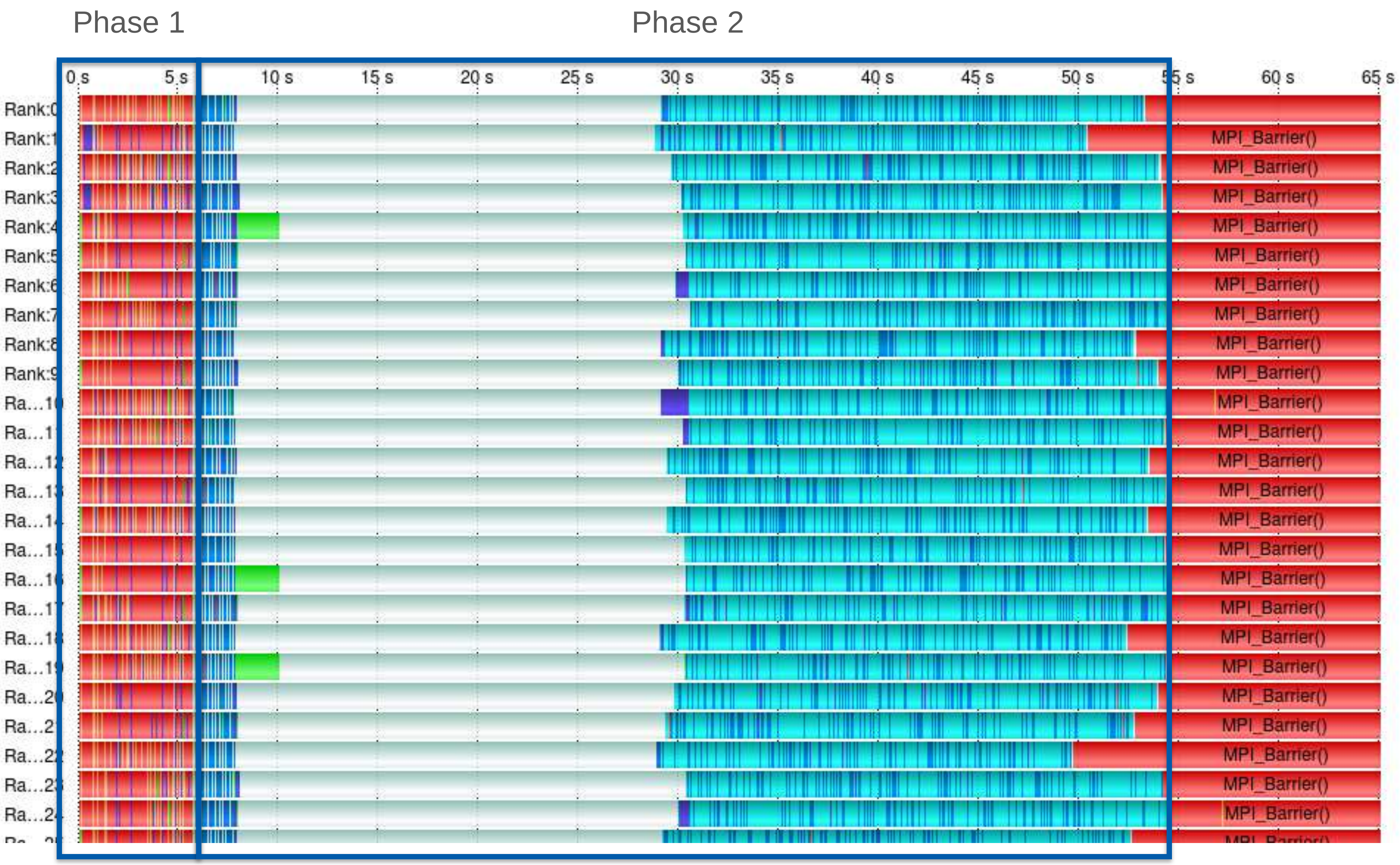
- Parallel sort of 64-bit integer values
  - Total data size may exceed available memory resources
  - Integer operations, all-to-all communication, file I/O
  - MPI + OpenMP

- Two Phases:
  I. Sort values from distinct data segments into num_nodes bins; transfer binned values to appropriate destination via MPI_AlltoAllv()
  II. Perform sort of local data into proper sequence
      - Interleaved computation and file I/O

LA-UR-21-26018

OpenSNAPI Use Case – I/O Offloading

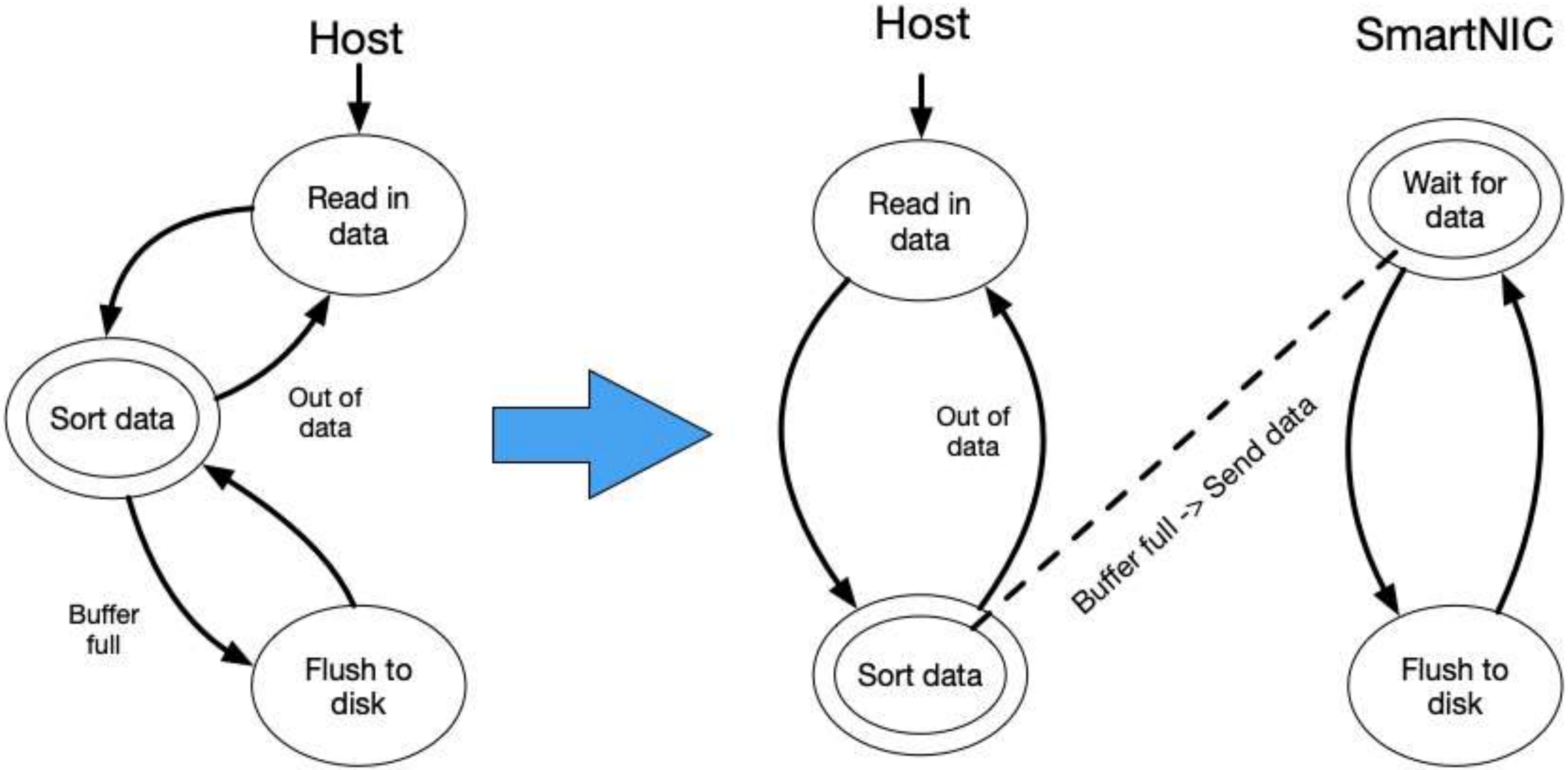
- **MPI Calls**
- **QuickSort**
- **Data Merge**
- **Write()**



LA-UR-21-26018

OpenSNAPI Use Case – I/O Offloading

Phase Two Abstraction

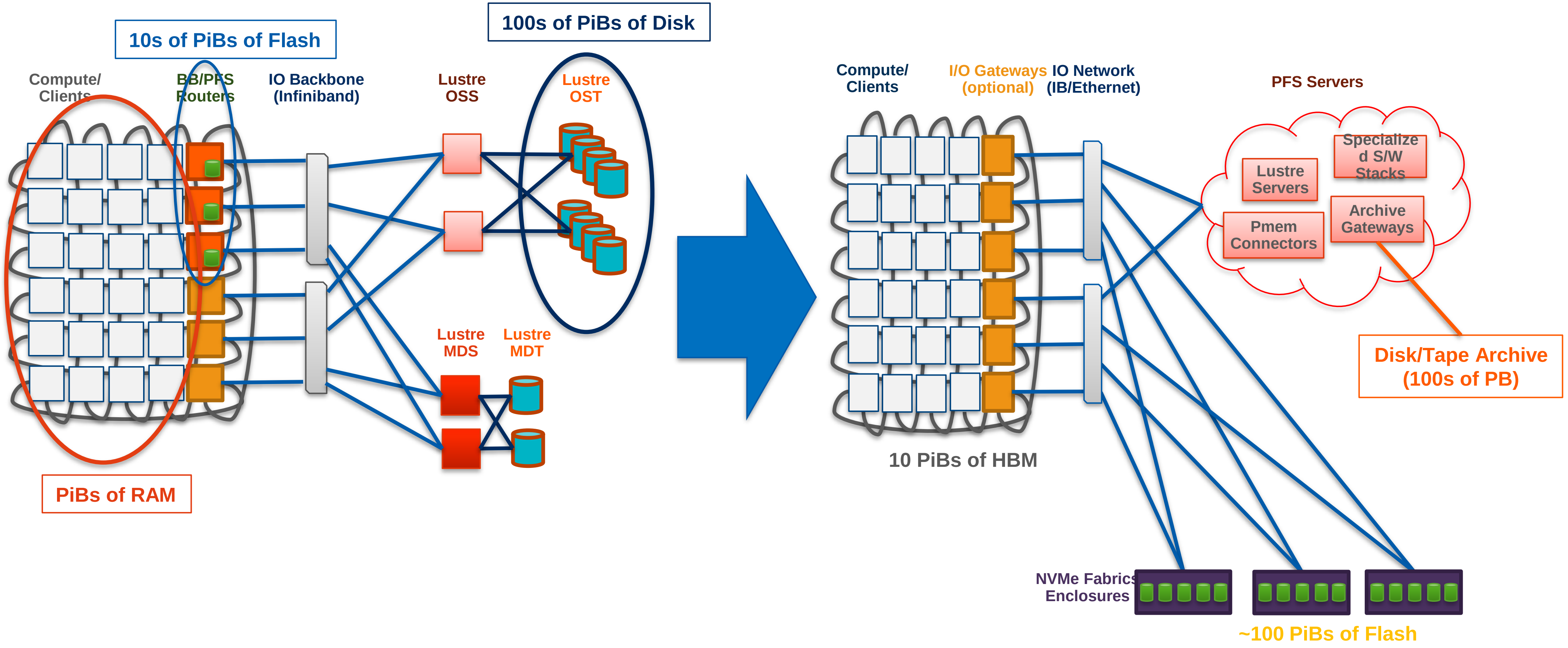


LA-UR-21-26018

How Does LANL Get From

Current

Future



Disaggregated

LA-UR-21-26018

OpenSNAPI - Progress report from Huawei

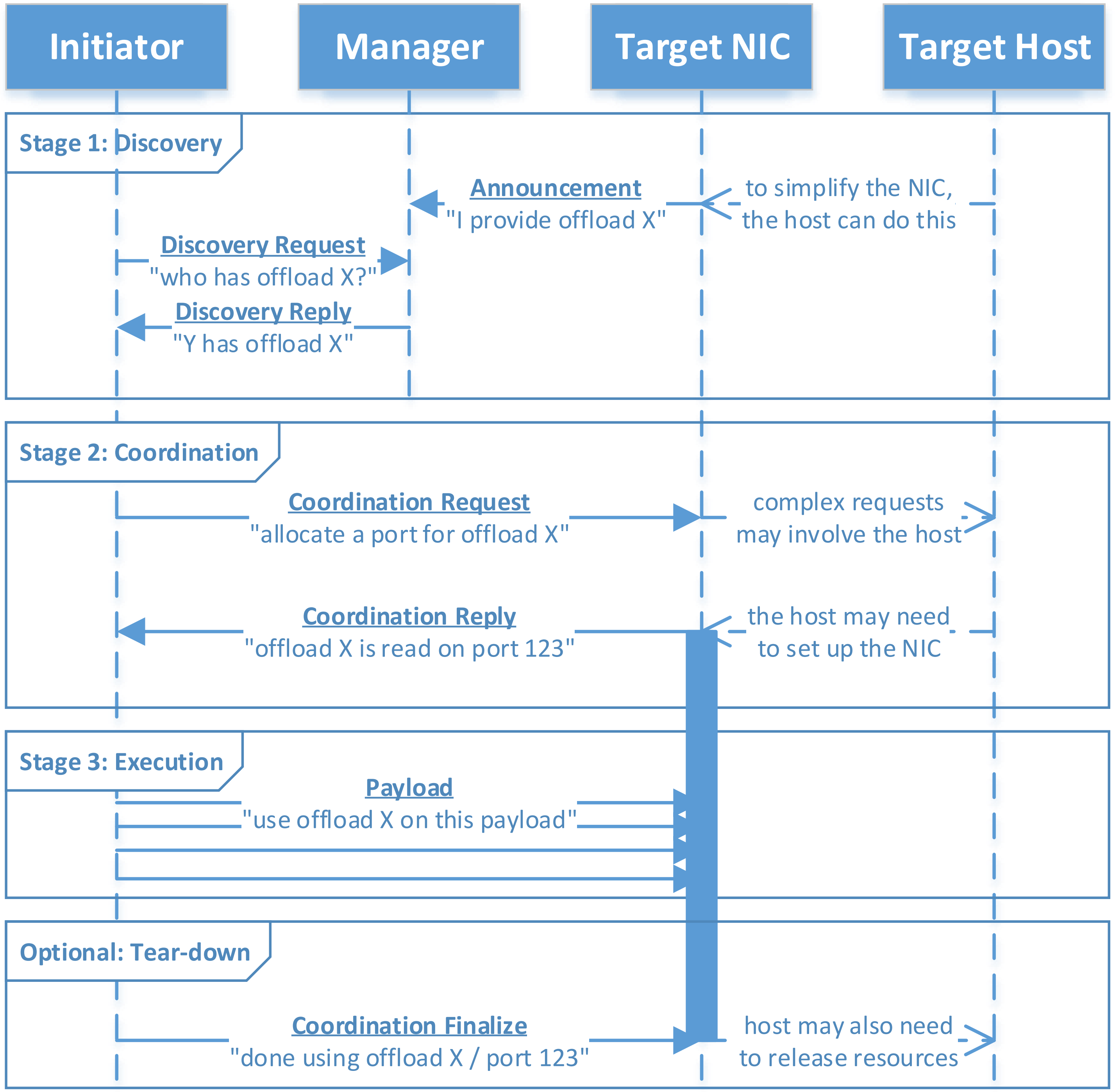
- Our focus: **Scalability**
  - Network-wide Smart-NIC resource discovery and utilization
  - Serving multiple clients from a single Smart-NIC, either local or remote
  - Topology-aware configuration of Smart-NICs

- How this translates to actual features?
  - Emphasis on remote management of Smart-NICs (relying on gNMI and gRPC)
  - Topology-dependent allocation of Smart-NIC resources within the cluster/datacenter
  - API combines existing hardware offloads with custom user programs for the Smart-NIC

Overview of Basic Flows

- The actors:
  - The “target”, offering its Smart-NIC
  - The “initiator”, willing to utilize it
  - Central “Manager” (got to have those ☺)

- General flow:
  1. Initiator asks for some logic/resources,
  2. Target allocates and sends the details,
  3. Initiator sends packets to apply it on,
  4. The transaction is complete.



OpenSNAPI Group Mission

- Enabling open, data-centric computing architectures

- The OpenSNAPI project mission is to create a standard application programming interface (API) for accessing the compute cores on the network, and specifically on the network adapter

- OpenSNAPI allows application developers to leverage the network compute cores in parallel to the host compute cores, for accelerating application run time, and to perform data operations closer to the data.

- Multiple use cases
  - Communication offload
  - Application acceleration
  - Security offload
  - Security enforcement

- Multiple deployment scenarios
  - Factory-based
  - Software update
  - Run-time

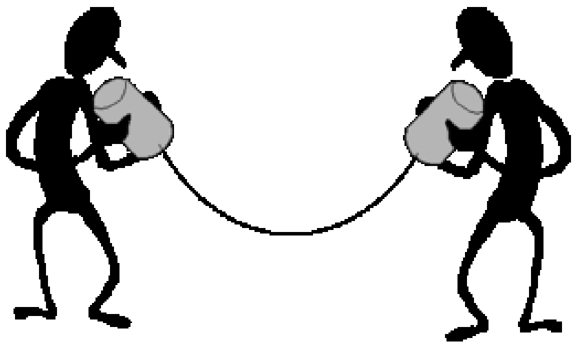
Proposed Basic OpenSNAPI Services

- OpenSNAPI mission is wide and does not limit new ideas!

- Service / offload / acceleration engine provisioning
  - Factory-based / Software update?
  - At runtime? At tenant / user / job provisioning?
  - Any user code? Certified code?

- Service Registration and Discovery
  - What services are available on a local device?
  - What services are available on any remote device?

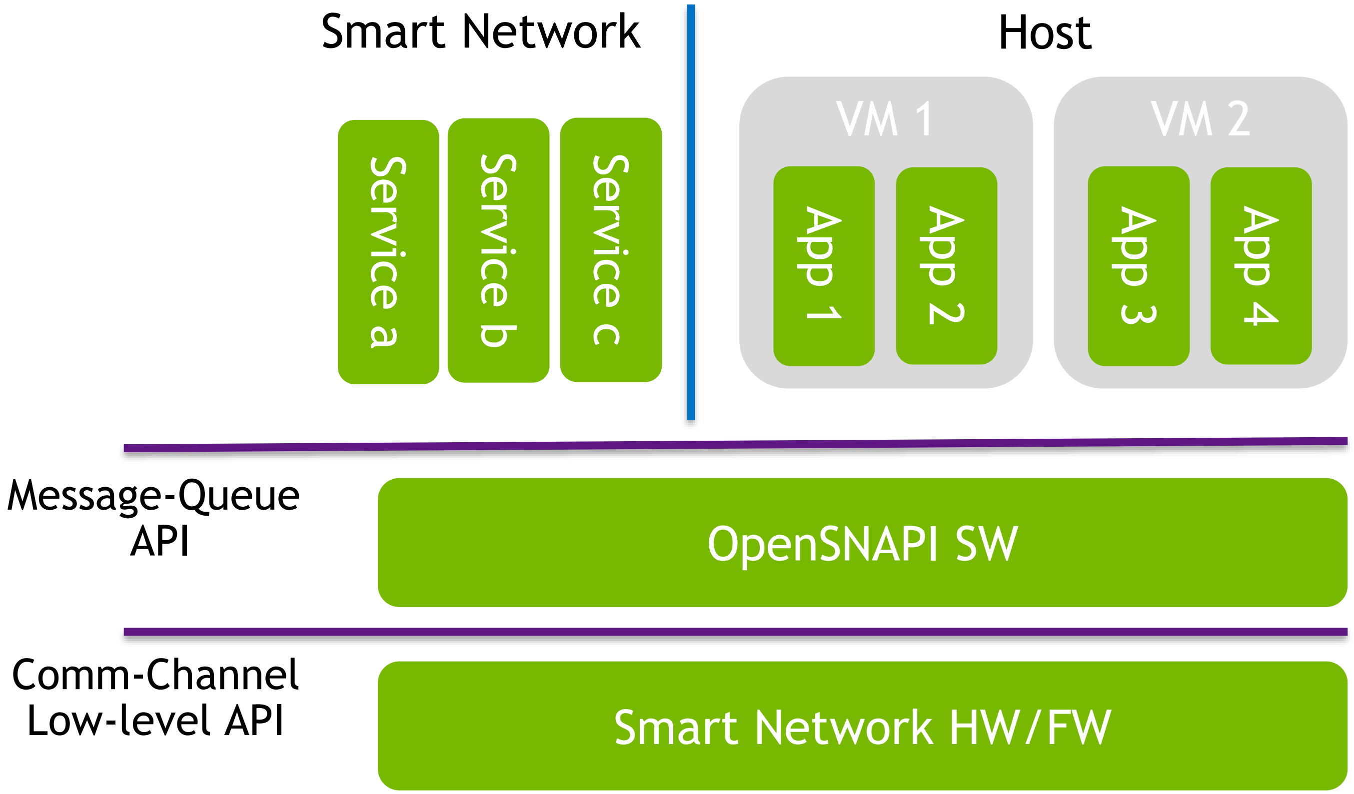
- Service Communication Channel
  - Configuration and control
  - Data-path



Communication Channel – Message Queue APIs

- Make it simple
  - Allow wide usage on different platforms (e.g., FPGA)
- Enable extensions
  - So special devices can exploit enhanced capabilities

Function	Input	Output
snapi_mq_create	devname, mq_params	mq
snapi_mq_connect	mq	-
snapi_mq_listen	mq	-
snapi_mq_send	mq, msg, size	-
snapi_mq_recv	mq	msg, size
snapi_mq_close	mq	-
snapi_mq_reg_mr	mq, addr, size	mr
snapi_mq_put	mq, size, src_mr, dst_mr	handle
snapi_mq_get	mq, size, dst_mr, src_mr	handle
snapi_mq_test	mq, handle	status
snapi_mq_dereg_mr	mr	-



UCF Unified Communication Framework

Thank You