2021 OFA Virtual Workshop

ACCELERATING HPC RUNTIMES SUCH AS OPENSHMEM WITH COPA

Dave Ozog, Andriy Kot, Venkata Krishnan

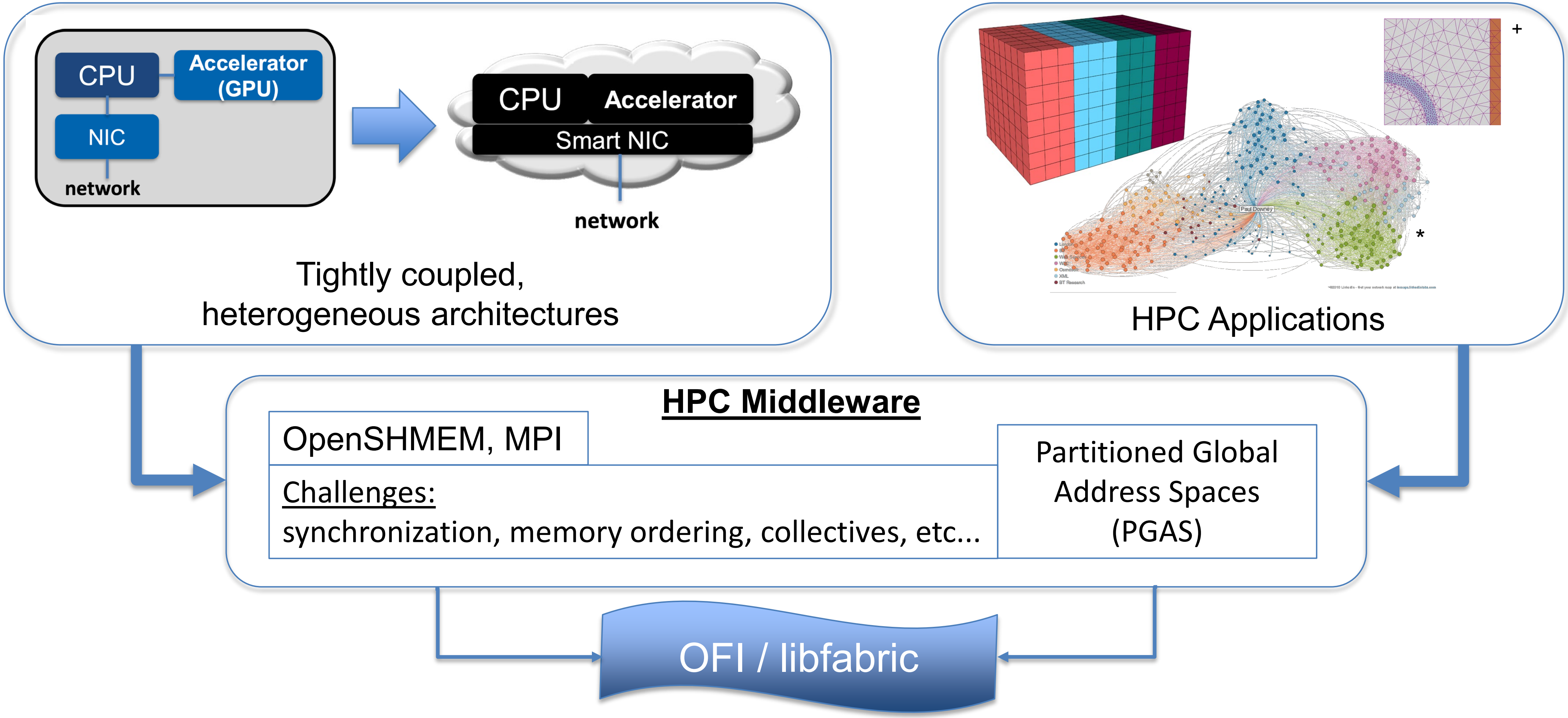
Intel Corporation

March 16, 2021

ACK: Mike Blocksome, Oliver Serres, Pallabi Chatterjee, Poorna Shivalingappa, Brian Holland

PROBLEM STATEMENT

FUTURE HPC ARCHITECTURES, APPLICATIONS, MIDDLEWARE



* Attr: Paul Downey; https://creativecommons.org/licenses/by/2.0/
+ Attr: Zureks; https://creativecommons.org/licenses/by-sa/3.0/deed.en

HOW DO WE ACCELERATE APPLICATIONS?

Option 1:

Accelerate / improve middleware interface standards

For example:

OpenSHMEM v1.6:

- Non-blocking collectives
- Per-PE fence

Option 2:

Application-aware accelerator optimizations

Extend middleware interfaces

For example:

- FI_ACCELERATION*
- SHMEMX interfaces

Offload custom app-specific patterns

For example:

- Custom collective ops
- Data transformation (e.g. compression, filtering)

* Enhancing OFI for Invoking Acceleration Capabilities on an Integrated Networking/Accelerator Platform (COPA) (OFAWS 2020)

CAN COPA ACCELERATE MIDDLEWARE?

- Recently defined an acceleration model for COPA✯
- Can OpenSHMEM and MPI interfaces benefit from COPA's acceleration capabilities? *Likely – e.g. collectives, per-PE fence/quiet, put w/ signal, etc.*
- What about application-level optimizations? *Yes.*
- Co-design effort OpenSHMEM + COPA SW/HW teams.
- Our Primary Goal: Run full OpenSHMEM on COPA HW.
- Future Plan: Prototype and analyze our ideas for OpenSHMEM/MPI/collectives accelerations

✯ Configurable Network Protocol Accelerator (COPA)
For details, see Hot Interconnects (HOTI) Aug 2020 & IEEE Micro Jan 2021

COPA BACKGROUND

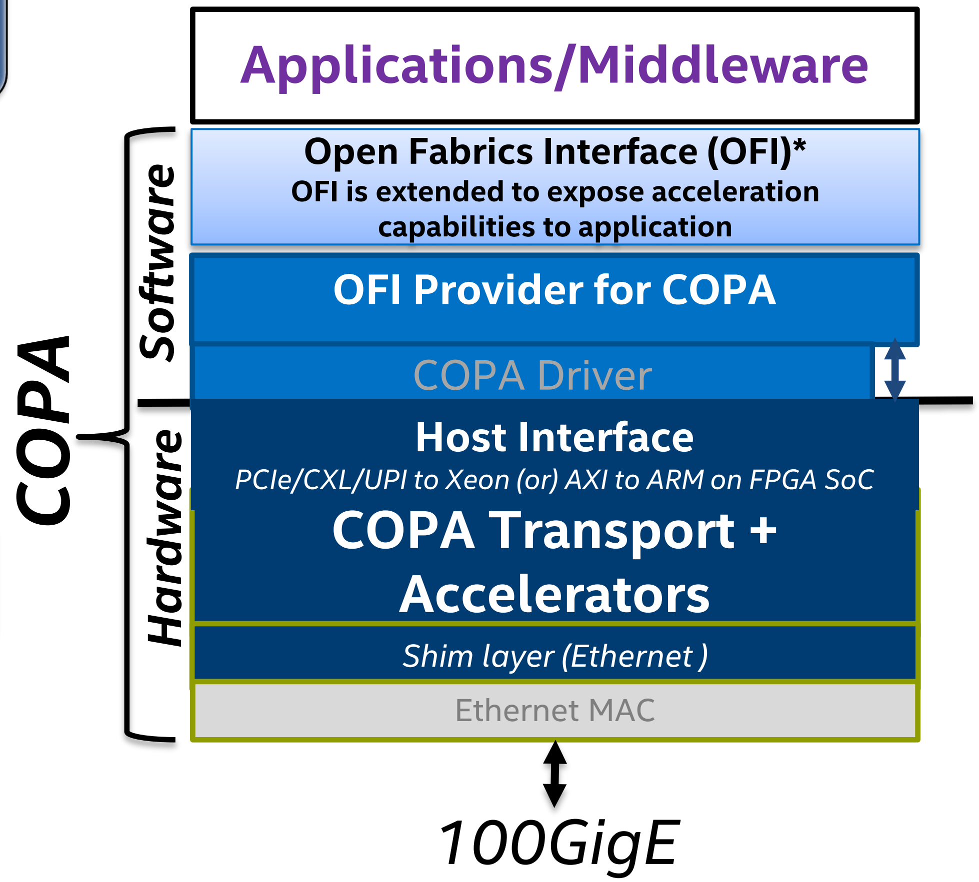
COPA† IS THE POC PLATFORM FOR OFI EXTENSIONS
(A SOFTWARE/HARDWARE FRAMEWORK FOR DISTRIBUTED FPGA COMPUTING)

Provides an integrated networking and accelerator framework with programming simplicity

- Supports RDMA (PUT/GET) based communication over commodity networks.
- Accelerators invoked as part of communication.
- Familiar environment developed around open standards (e.g. libfabric/OFI)

Customizable framework for specific deployments

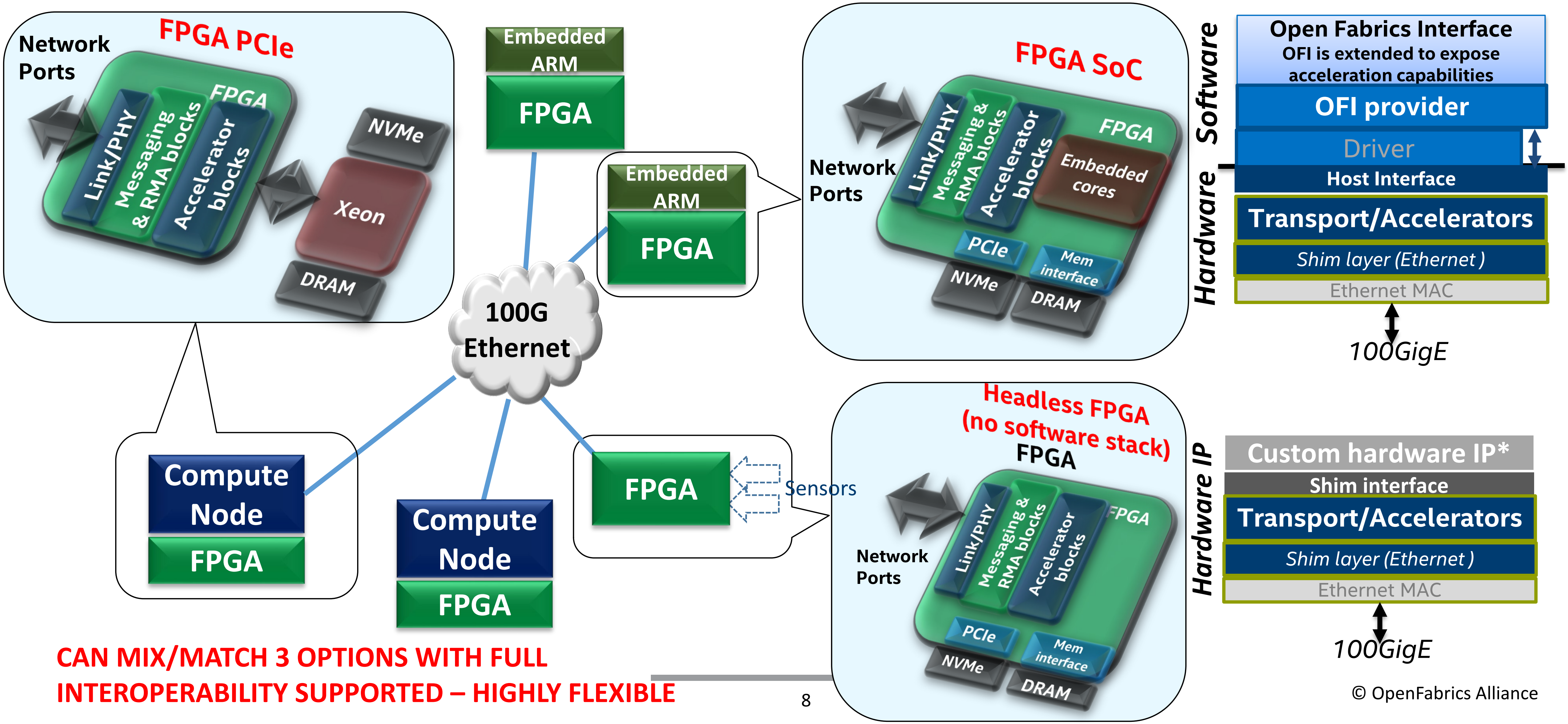
- Provides a modular architecture - can add necessary IP (accelerator) blocks and new features for a customized solution



†**COPA** = **CO**nfigurable network **P**rotocol **A**ccelerator

SYSTEM COMPONENTS

OFI stack on FPGA SOC and FPGA PCIe (currently on Stratix 10)



CAN MIX/MATCH 3 OPTIONS WITH FULL INTEROPERABILITY SUPPORTED – HIGHLY FLEXIBLE

COPA SOFTWARE STACK

Applications / Middleware

Open Fabrics Interface (OFI)

OFI provider for COPA (libfabric)

COPA hardware library (libchw)

COPA device driver

COPA FPGA hardware

- Open Fabrics Interface (OFI) implementation
- Main application programming interface to COPA functionality
- Extended for acceleration
- All access to COPA FPGA through COPA hardware library (libchw) for kernel bypass

- Main system programming interface to COPA FPGA hardware (ioctl syscalls)
- Opens device and mmap's hardware command queues and event queues into user process address space
- Optionally hardware emulation mode - COPA functional model

- Memory map CSRs and command queues at initialization time
- Initialize E2E connection table
- Provides memory pinning and virtual-to-physical address translation
- Not accessed on critical path - Userspace application software interacts with hardware using mmaped command queues

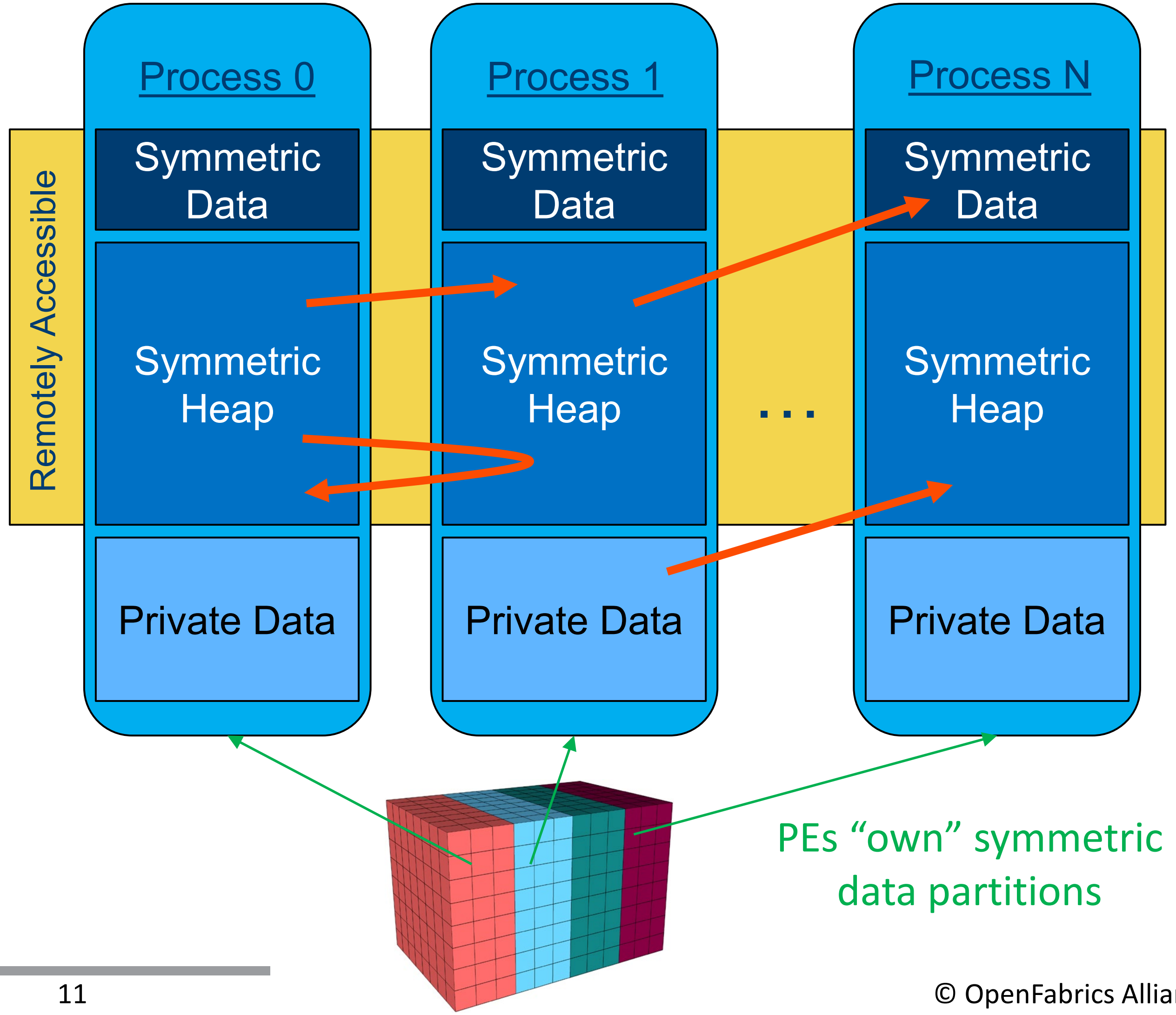
OPENSHMEM ACCELERATION

OPENSHMEM CRASH COURSE

OpenSHMEM:
- Open standard PGAS model: emphasizes one-sided operations (put/get/atomics w/ RDMA)
- Symmetric memory exposed for remote access across processing elements (PEs)
- This work uses Sandia OpenSHMEM (SOS) v1.5.0

Operations:
- *RMA/AMO*: put/get, remote atomics
- *Collectives*: barrier, broadcast, reductions, etc.
- *Memory ordering*: fence and quiet

Specification:
- Currently version 1.5
- Upcoming: spaces, NB collectives, per-PE fence/quiet, and more



* OpenSHMEM and OFI: Better Together (OFAWS 2018)

PRACTICAL MATTERS / WORKAROUNDS

- **Some practical matters need to be addressed to run SOS on COPA:**
  - **FI_CONTEXT2:**
    - COPA leverages FI_CONTEXT2 (64B), but SOS doesn't currently pass contexts to communication ops.
    - Solution: COPA now supports an internal context2 pool so apps don't need to bother with contexts.
  - **FI_PROGRESS_MANUAL:**
    - COPA does not yet support full automatic progress; SOS (the stable release) doesn't fully support manual.
    - Solution: hacked an unofficial version of SOS with a timer-based progress thread (caveat: no private contexts).
  - **FI_MR_LOCAL:**
    - HW requires *all* buffers to be registered, but OpenSHMEM supports src buffers outside symmetric heap.
  - Registered memory must be physically contiguous – this means we might have to strip out global/static variables.
  - Remote atomics support is currently in software only.
  - Optional, not yet supported: FI_THREAD_DOMAIN, FI_RMA_EVENT, FI_FENCE, FI_DELIVERY_COMPLETE
  - **<u>Fence/quiet operations leverage libfabric counters</u>**

- **Let's dive into the <u>counters</u> – they're interesting and important.**

SANDIA OPENSHMEM PUT/GET COUNTERS

Example "trace" of put/get counters througout a simple OpenSHMEM program:

SHMEM Program	pending put	pending get	put	get
`put(..., PE1)`	1	0	0	0
`put(..., PE2)`	2	0	0	0
`get(..., PE1)`	2	1	0	0
`get(..., PE2)`	2	2	1	0
`amo_inc(..., PE1)`	3	2	1	0
`amo_fetch(..., PE2)`	3	3	1	2
`fence*/quiet()`	3	3	3	3

Local atomics (pending put, pending get)

libfabric counters (put, get)

1 put/get counter pair per SHMEM ctx

Target (RX) counter support, if available

`* fence != quiet` on ordered networks in SOS, get_wait() only...

COPA HARDWARE COUNTERS

- Implemented as Control Status Registers (CSRs)
- CSRs are setup to atomically operate on counters (read/write, inc., trigger)
- COPA supports 512 counters now, but this number is flexible.
- All CSRs are mapped to individual PEs for counter isolation.
- Reading CSRs is expensive, so HW periodically performs a write-back of the counter values to host memory.
- Write-backs occur only during wait() operations:
  - no timer-based write-backs yet, but could be added later.
  - wait() has a desired trigger value. When met, HW does a write-back.
- Incrementing by 0 performs write-back to host memory.

COPA HARDWARE COUNTERS

COPA HW counters enable:

- An optimized per-PE fence/quiet interface (e.g., as proposed for OpenSHMEM v1.6):

  Current (v1.5) interface:

`shmem_fence()` `shmem_quiet()`	`shmem_ctx_fence(shmem_ctx_t ctx)` `shmem_ctx_quiet(shmem_ctx_t ctx)`

  Proposed (v1.6) interface:

`shmem_pe_fence(ctx, target_pe)` `shmem_pe_quiet(ctx, target_pe)`	`shmem_ctx_pe_fence(ctx, target_pe)` `shmem_ctx_pe_quiet(ctx, target_pe)`

- Per-operation completion tracking in the OpenSHMEM runtime
  - Applications *tend* to quiet many puts, gets, or AMOs in isolation (i.e. not mixed)
  - Possible advantage in tracking operation-specific completions

CURRENT STATUS

Current Status:

- We are very close to full SOS support with the COPA libfabric provider.
- Several temporary workarounds are in place (see slide 12), next steps are to modify them.

Future Work:

- MPI validation – potentially fewer workarounds (manual progress support, FI_CTX2, etc.)
- Implement proposed OpenSHMEM optimizations (in HW where possible).
- Performance analysis of microbenchmarks and key applications.

CLOSING REMARKS

OFI's open standard & definition of communication API agnostic to protocols/hardware makes it ideal to include acceleration extensions

Summary:

- COPA provider exposes acceleration capabilities via OFI.
- Several opportunities to accelerate OpenSHMEM with COPA+OFI.
- Co-design HW/SW effort is underway and inspiring promising ideas.
- COPA counters are flexible, performant, OFI/SOS compatible, and enable optimizations like per-PE fence/quiet and per-op tracking.

Future:

- Long-term prospects to extend OpenSHMEM, MPI, and OFI interfaces and capabilities.
- Compute is moving into the fabric, and COPA will customize for HPC application and middleware accelerations.

2021 OFA Virtual Workshop

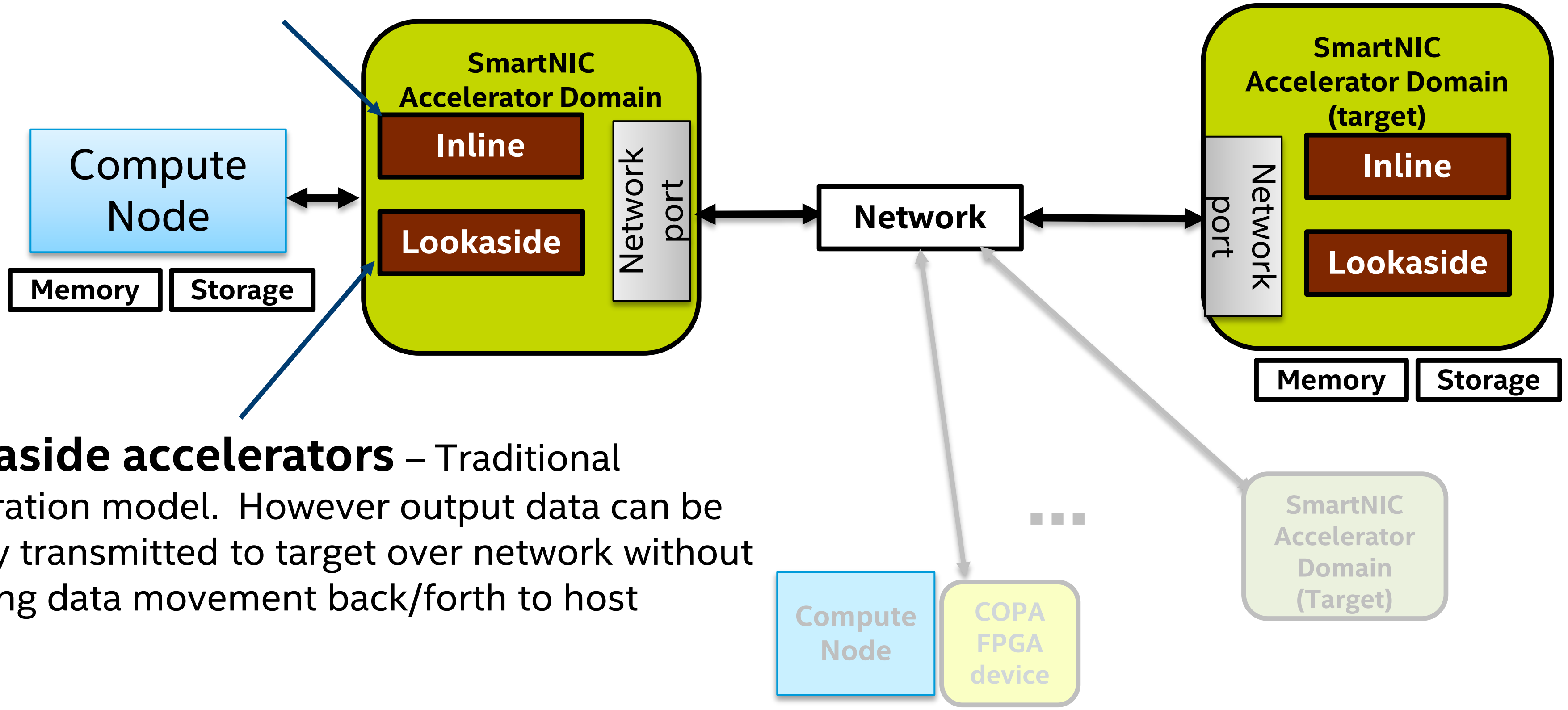
THANK YOU

BACKUP

ACCELERATOR MODELS (INTEGRATED WITH NETWORKING)

Inline accelerators perform compute on data during transmit/receive operation (streaming model)

Remote Mode Inline/Lookaside accelerators can be triggered by incoming packet. **No host/OS involvement**



Naturally extends to offloading collectives, reduction, atomics, distributed hash lookup etc.

Lookaside accelerators – Traditional acceleration model. However output data can be directly transmitted to target over network without requiring data movement back/forth to host

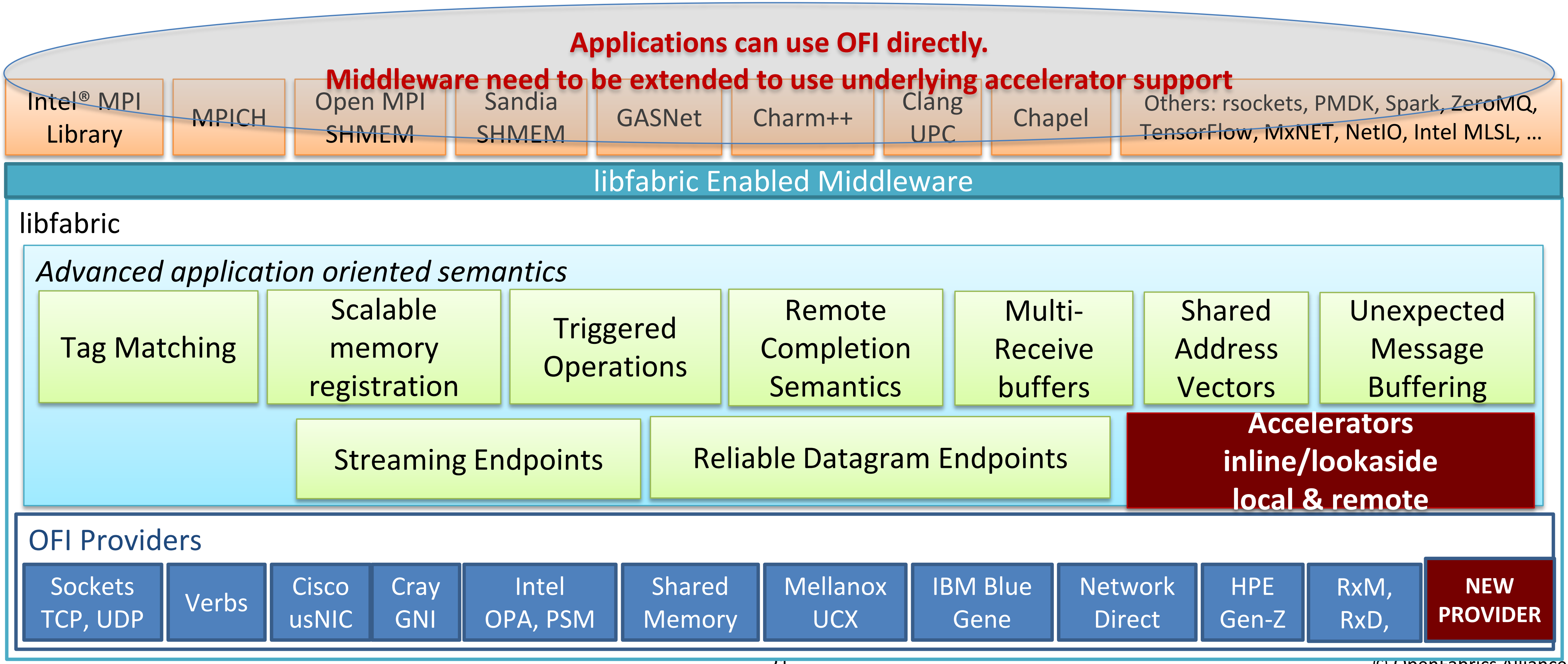
Could hang off a switch port (headless) or be an integral part of the switch

Need for a standard API to expose acceleration modes to middleware & applications

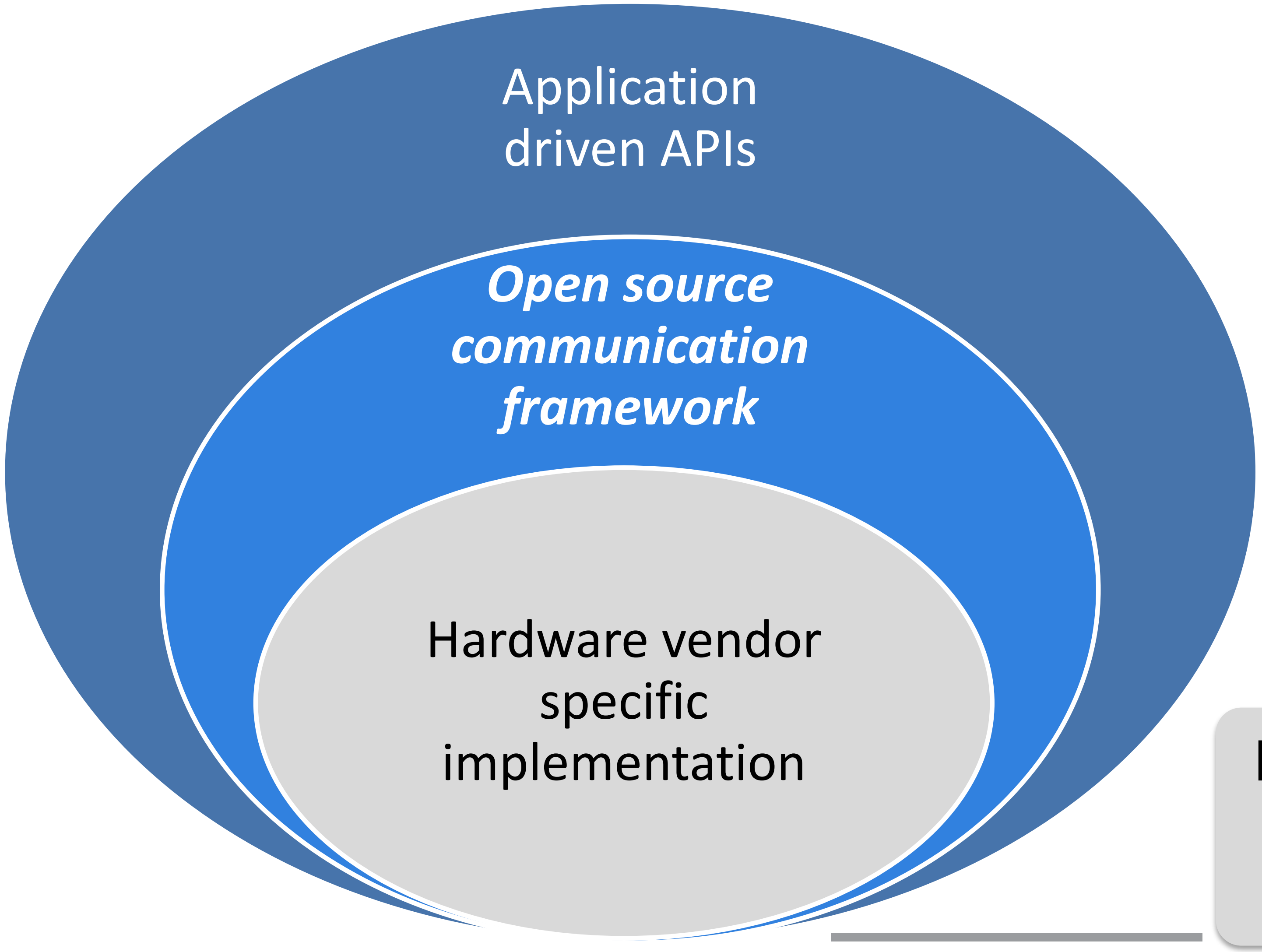
APPROACH – USE OFI (WITH EXTENSIONS)

Extend a network API to include acceleration support to support a truly scalable model

- **Extending an accelerator API (e.g. OpenCL) to support networking is not scalable**



CURRENT VISION OF SOLUTION



Based on internal hardware prototyping – FPGA-based

APIs targeting application use of specific accelerations

Extend existing communication framework to support acceleration functions

Define mechanism to pass input/output parameters and invoke acceleration

OFI COPA PROVIDER

- **Full featured OFI provider**
- **Only small changes needed to add acceleration to existing OFI-enabled middleware and applications**
- **<u>Temporary</u> until official OFI support**
- **Minimal OFI extensions to enable "inline" and "lookaside" COPA acceleration**
  - Extend semantics of data structures and operations
  - Define new FLAGS for acceleration
- **Implements a wide variety of interfaces to support many kinds of HPC middleware**
  - FI_MSG, FI_TAGGED, FI_RMA
  - FI_PROGRESS_MANUAL, FI_THREAD_COMPLETION, FI_AV_MAP
  - FI_EP_RDM

ENABLE ACCELERATION

- **New FI_ACCELERATION flag informs provider application wants inline accelerator to be invoked during a data movement operations**

- **FI_ACCELERATION flag can be set on the endpoint object to invoke acceleration on all endpoint data movement operations**
  - fi_control() with FI_SETOPTS

- **Alternatively, FI_ACCELERATION flag can be specified for individual data movement operations**
  - fi_write_msg()
  - fi_read_msg()

ACCELERATOR OUTPUTS

- **Output data may be provided as a result of acceleration**
- **Available for endpoints bound to a completion queue initialized with data format**
  - FI_CQ_FORMAT_DATA
  - FI_CQ_FORMAT_TAGGED
- **FI_ACCELERATION flags, etc., are set in the flags field**
  - FI_CQ_FORMAT_MSG

- **Normally the completion entry data field is for remote metadata**
- **Extend the data field semantics for initiator acceleration output**

```
struct fi_cq_data_entry {
  void     *op_context; /* operation context */
  uint64_t flags;       /* completion flags */
  size_t   len;         /* size of received data */
  void     *buf;        /* receive data buffer */
  uint64_t data;        /* completion data */
};
```

LOOKASIDE ACCELERATION

- **Local operation – no fabric communication involved**
- **Complex accelerators that do not fit in the packet pipeline (inline acceleration)**
- **Same mechanism as inline to invoke lookaside acceleration**
  - fi_read(), fi_write(), etc.
  - FI_ACCELERATION
- **Lookaside accelerator flags**
  - FI_LOOKASIDE_ACCELERATION_*
- **Current restrictions**
  - physically contiguous memory for all inputs and outputs