2020 OFA Virtual Workshop

ENHANCING OFI FOR INVOKING ACCELERATION CAPABILITIES ON AN INTEGRATED NETWORKING/ACCELERATOR PLATFORM

Venkata Krishnan, Olivier Serres, Mike Blocksome

Intel Corporation

June 19, 2020

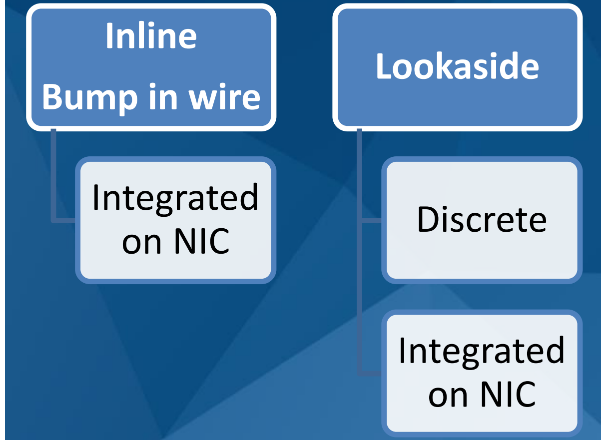
ACK: Andriy Kot, Patrick McCormick, Sean Hefty

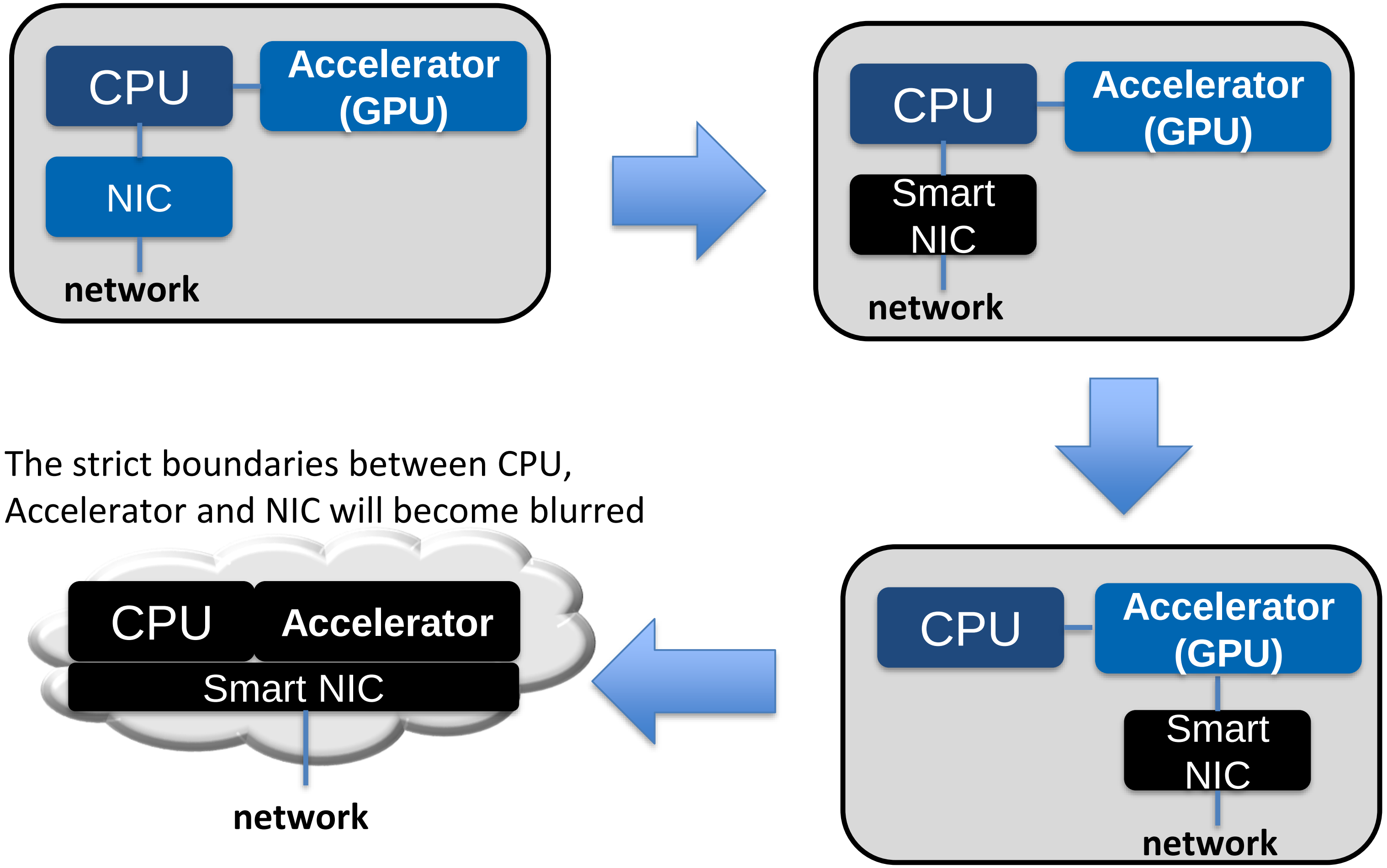
PROBLEM STATEMENT

© OpenFabrics Alliance

NETWORK IS THE COMPUTER®
(AND THIS INCLUDES ACCELERATORS)

To achieve scalable performance, acceleration will become an integral part of network communication.





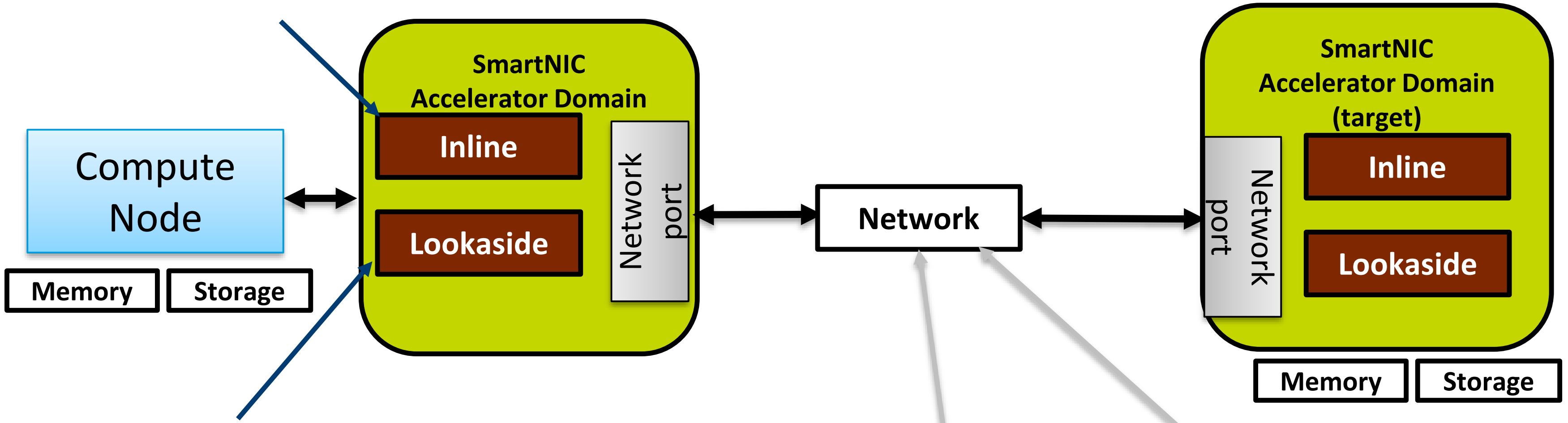
® John Gage & Cloudera

© OpenFabrics Alliance

ACCELERATOR MODELS (INTEGRATED WITH NETWORKING)

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

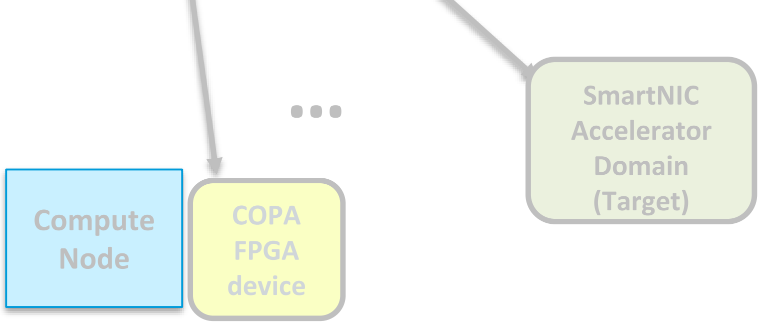
Remote Mode Inline/Lookaside accelerators can be triggered by incoming packet. **No host/OS involvement**



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

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

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



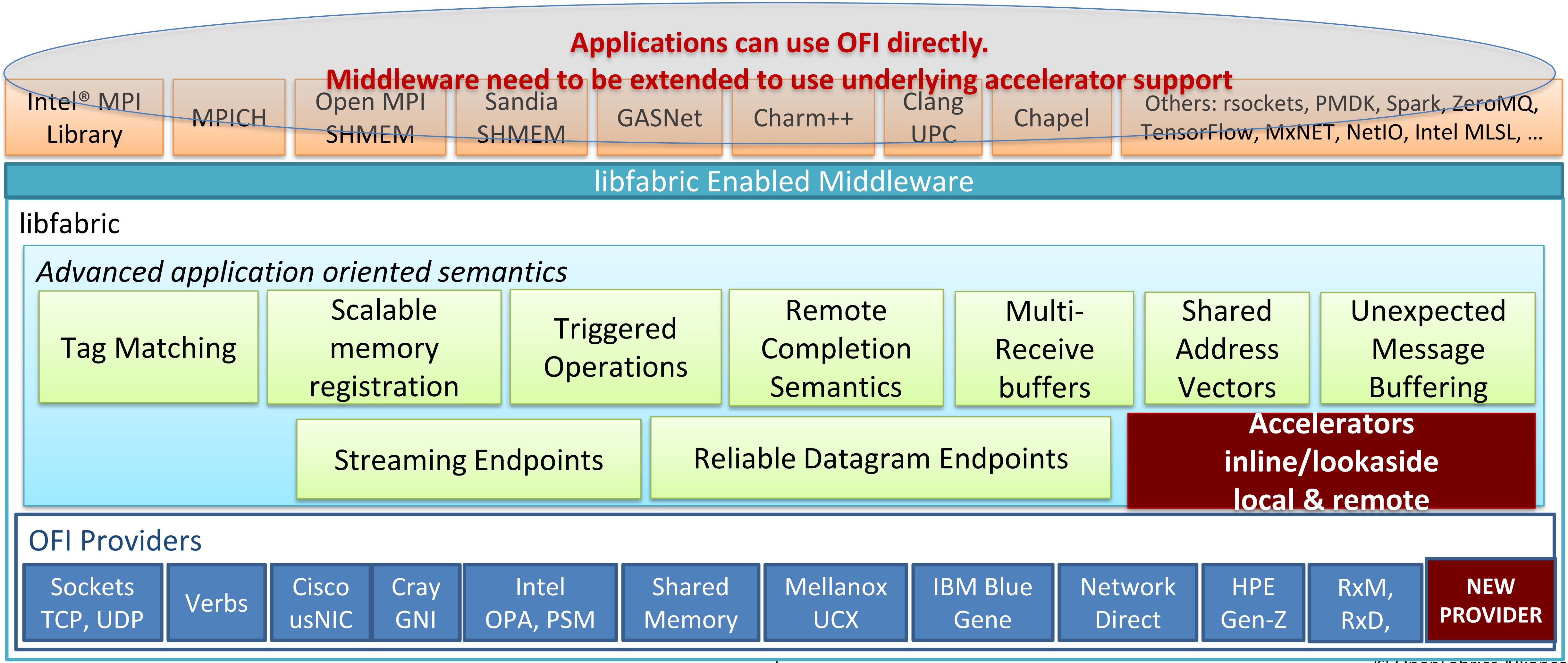
Need for a standard API to expose acceleration modes to middleware & applications

© OpenFabrics Alliance

APPROACH – USE OFI (WITH EXTENSIONS)

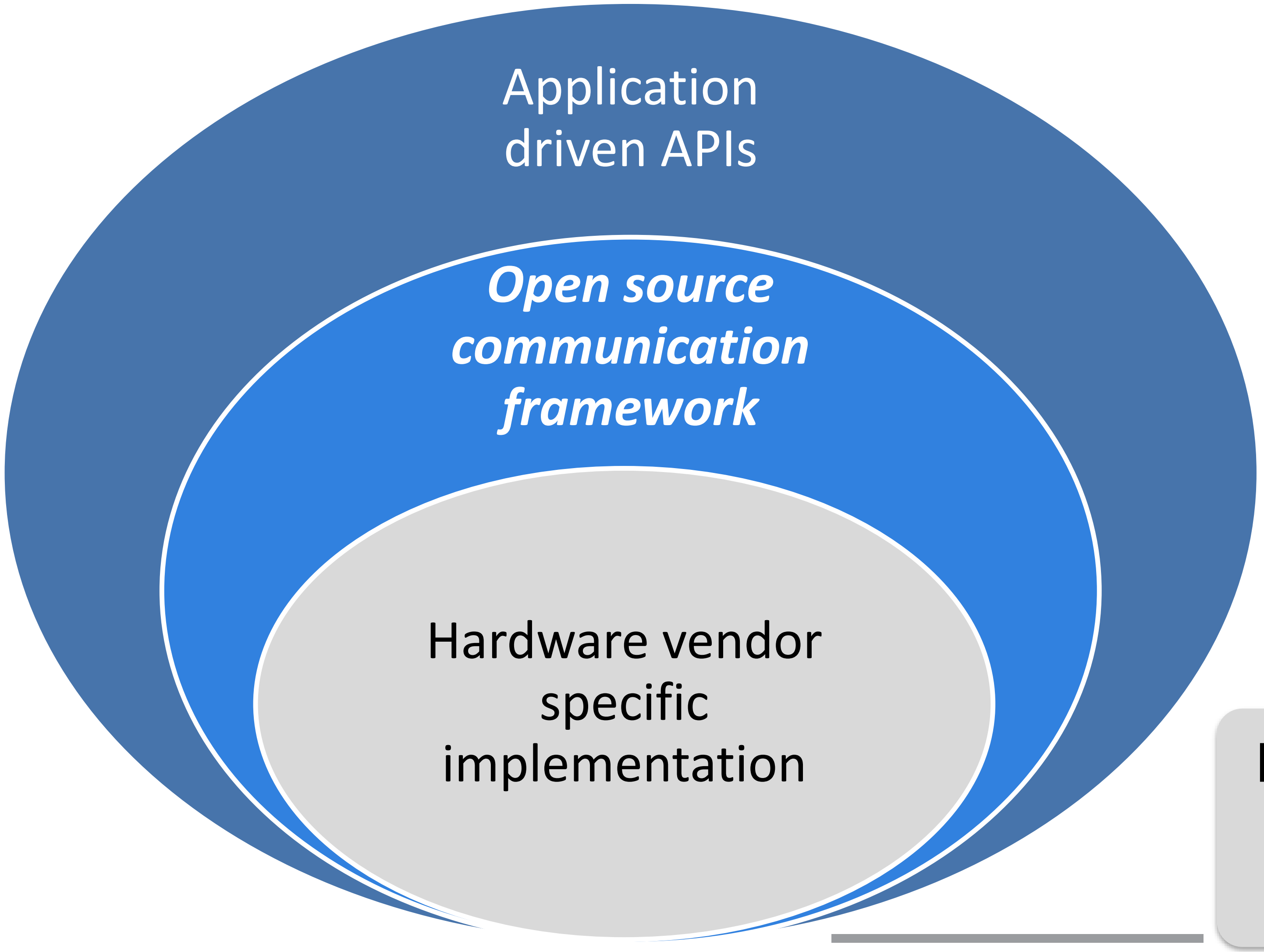
Extend a network API to include acceleration support to support a truly scalable model

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



© OpenFabrics Alliance

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

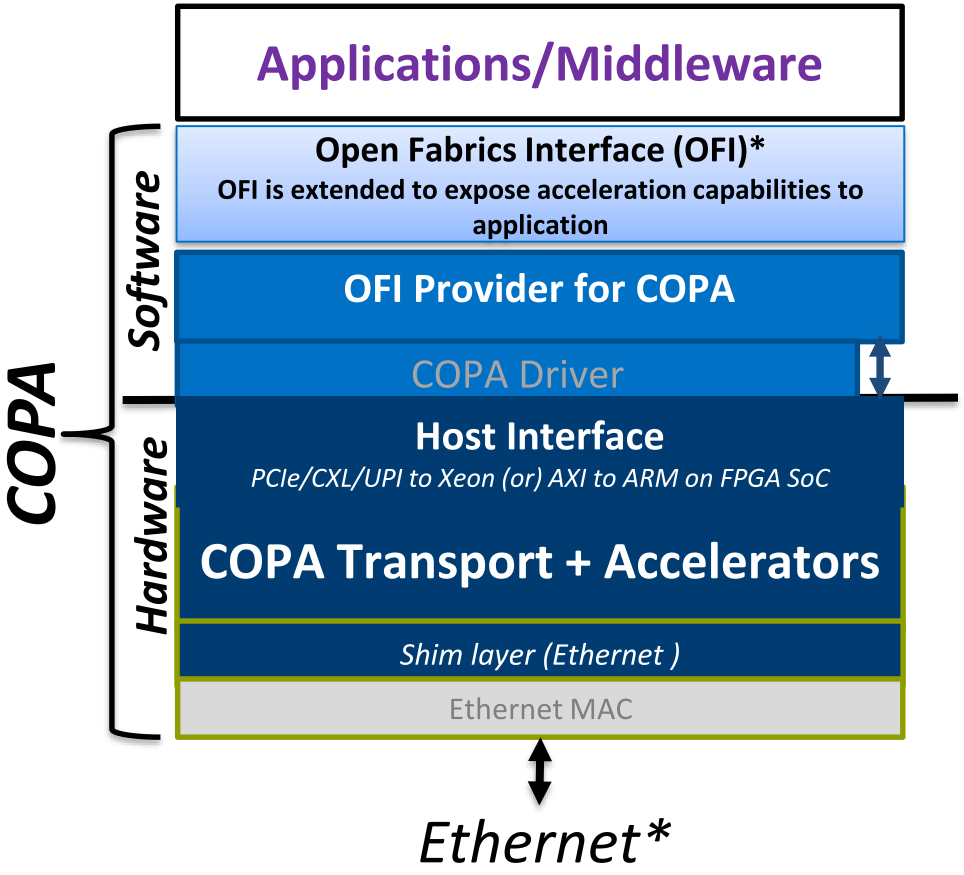
PROTOTYPE FOR OFI EXTENSIONS – SMARTNIC/ACCELERATOR FPGA FRAMEWORK - COPA †

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

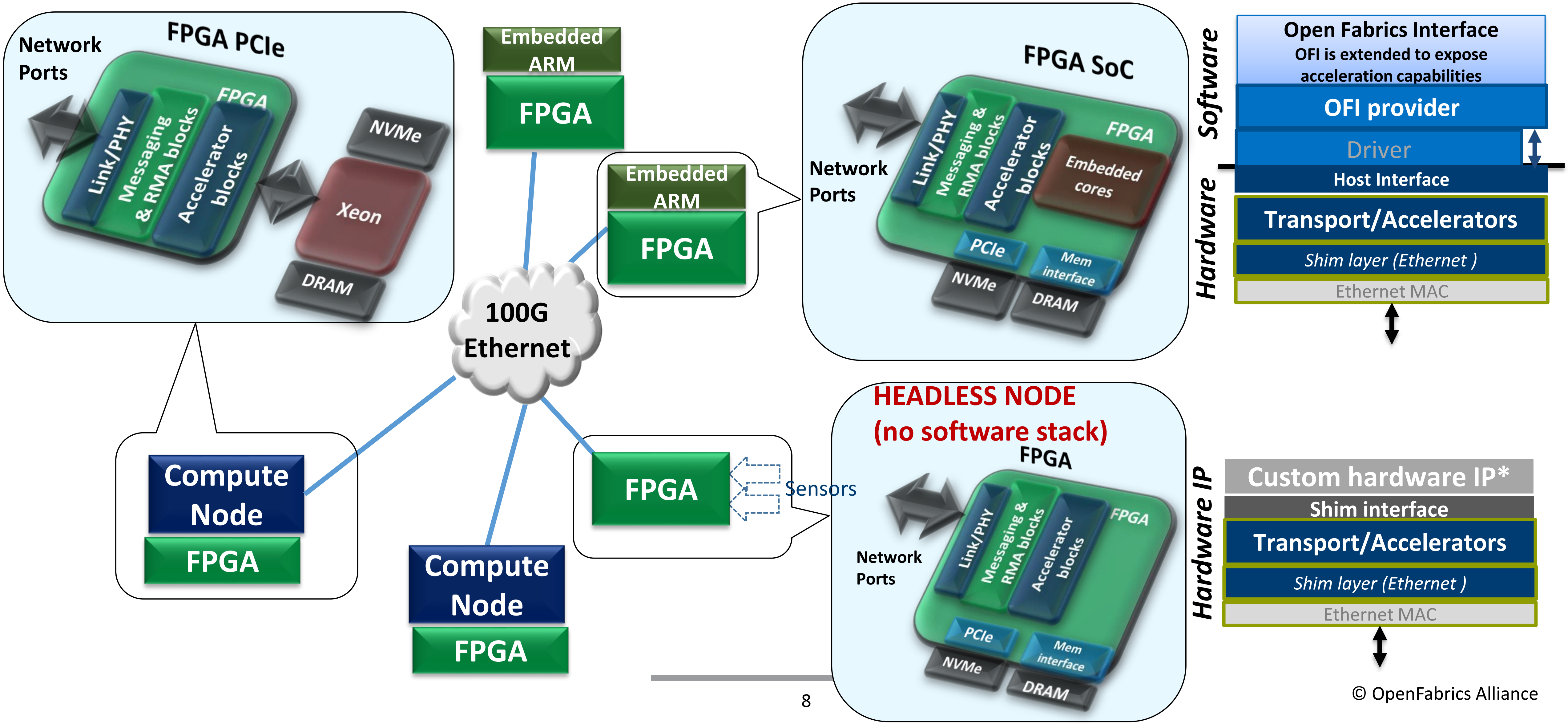
- Provides a modular architecture - can add necessary IP (accelerator) blocks and new features for a customized solution



† **CO**PA stands for **CO**nfigurable network **P**rotocol **A**ccelerator. It is a POC and not a product.

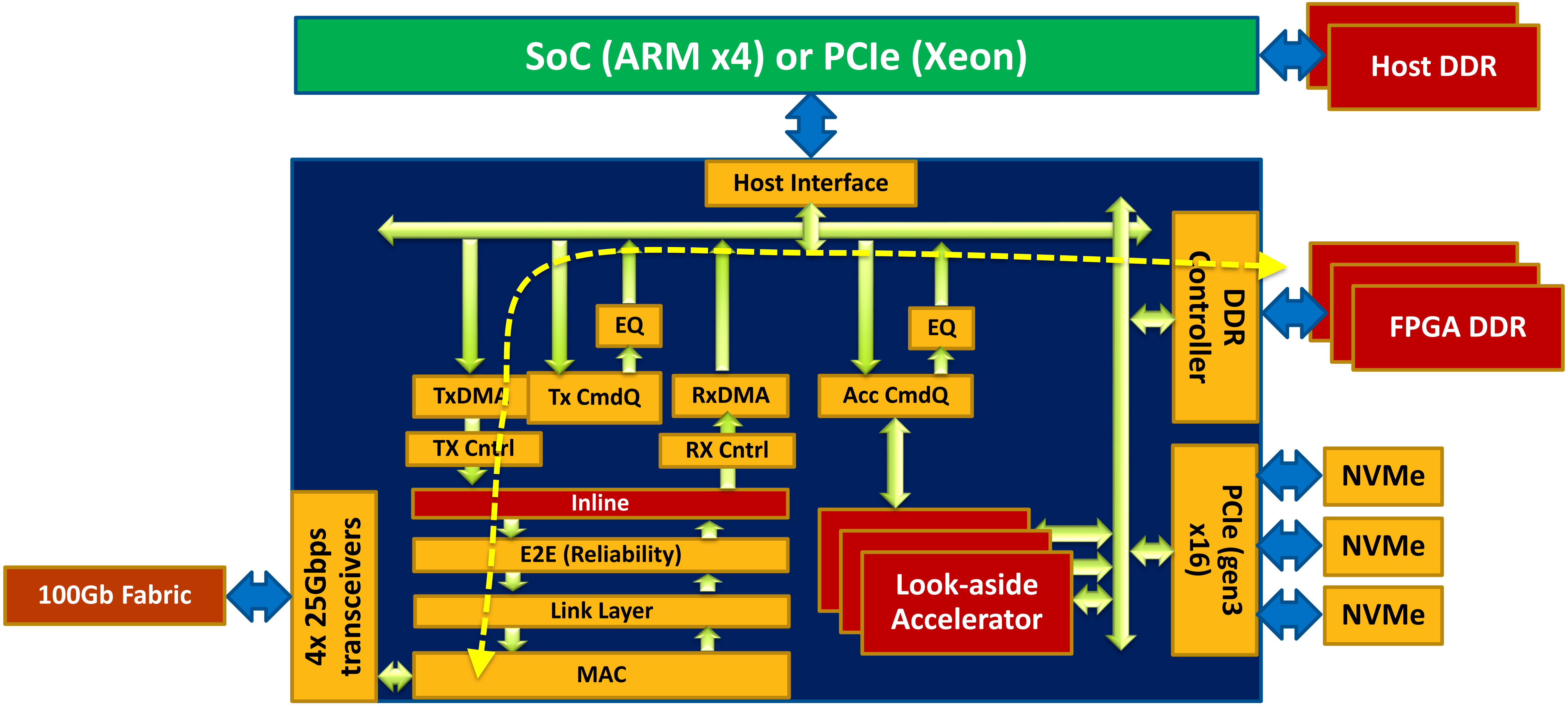
SYSTEM COMPONENTS

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



© OpenFabrics Alliance

HARDWARE ARCHITECTURE – NIC/ACCELERATOR DOMAIN



© OpenFabrics Alliance

COPA SOFTWARE STACK

Applications / Middleware

Open Fabrics Interface (OFI)

OFI provider for COPA (libfabric)

- 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

COPA hardware library (libchw)

- 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

COPA device driver

- 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

COPA FPGA hardware

© OpenFabrics Alliance

OFI COPA PROVIDER

- **Full featured OFI provider**
- **Only small changes needed to add acceleration to existing OFI-enabled middleware and applications**
- **Temporary 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

© OpenFabrics Alliance

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()

© OpenFabrics Alliance

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 */
};
```


© OpenFabrics Alliance

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

© OpenFabrics Alliance

CLOSING REMARKS

Traditional Accelerator APIs are meant for single node and not really architected to support networking

Network APIs are more suitable for this purpose

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

- **COPA provider exposes acceleration capabilities via OFI**
  - With minimal extensions or extending semantics of certain operations or structures. ***NOT OFFICIAL OFI***

- Both inline & lookaside acceleration invocations (including remote triggered mode) **validated on a fully functional COPA FPGA cluster**

- Work has shown that OFI/libfabric can indeed provide a **<u>unified interface for invoking acceleration/networking</u>**

© OpenFabrics Alliance

BACKUP - EXAMPLES

© OpenFabrics Alliance

ENDPOINT FLAGS

Set endpoint flags

```
struct fid_ep *ep = …;
uint64_t tx_flags = FI_TRANSMIT;
fi_control(ep, FI_GETOPSFLAG, (void*)&tx_flags);
uint64_t new_tx_flags = tx_flags | FI_TRANSMIT | FI_ACCELERATION;
fi_control(ep, FI_SETOPSFLAG, (void*)&new_tx_flags);
…
/* every data transfer on the endpoint is accelerated */
fi_write(ep, buf, len, desc, dest_addr, addr, key, context);
```

Create endpoint alias

```
struct fid_ep *ep = …;
struct fid_ep *acc_ep = NULL;
fi_ep_alias(ep, &acc_ep, FI_TRANSMIT | FI_ACCELERATION | … );
…
/* normal, non-acceleration, data transfer */
fi_write(ep, buf, len, desc, dest_addr, addr, key, context);

/* acceleration data transfer */
fi_write(acc_ep, buf, len, desc, dest_addr, addr, key, context);
```


© OpenFabrics Alliance

ACCELERATION INPUTS

```
union fi_acceleration {
  struct fi_context2 context;
  struct {
    union {
      uint32_t  input_u32[7];
      uint16_t  input_u16[14];
      uint8_t   input_u8[28];
    };
    uint32_t    reserved[7];
    uint64_t    flags;
  } __attribute__((packed));
};
```

```
#define FI_INLINE_ACCELERATOR_0       (1ULL << 32)
…
#define FI_INLINE_ACCELERATOR_7       (1ULL << 39)
#define FI_LOOKASIDE_ACCELERATOR_0    (1ULL << 40)
…
#define FI_LOOKASIDE_ACCELERATOR_7    (1ULL << 47)
…
union fi_acceleration context;
context.flags = FI_INLINE_ACCELERATOR_0;
context.input_u32[0] = 0x11223344;

fi_write(ep, buf, len, desc, dest_addr, addr, key, &context);
```


© OpenFabrics Alliance

ACCELERATION OUTPUTS

```
/* FI_CQ_FORMAT_DATA */
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 */
};
```

```
union fi_acceleration context;
context.flags = FI_INLINE_ACCELERATOR_0;       /* e.g. crc */
context.input_u32[0] = 0x11223344;

fi_write(ep, buf, len, desc, dest_addr, addr, key, &context);
…
do {
  rc = fi_cq_read(cq, &entry, 1);
} while (rc != -FI_EAGAIN);

uint32_t crc = 0;
if (entry.flags & FI_INLINE_ACCELERATOR_0) {
  crc = entry.data;
}
```


© OpenFabrics Alliance

INVOKE ACCELERATOR ON RDMA WRITE (PUT)

```
struct fid_ep *ep = ...; /* endpoint with FI_ACCLERATION flag
*/
fi_addr_t self = ...;  /* CRC example is "send-to-self" */

/* create src and dst memory regions */
uint8_t src_buffer[1024*1024];
struct fid_mr *src_mr = NULL;
fi_mr_reg(domain, src_buffer, sizeof(src_buffer),
    0, 0, 0, flags, &src_mr, NULL);
void *desc = fi_mr_desc(src_mr);  /* local mr descriptor */

uint8_t dst_buffer[1024*1024];
struct fid_mr *dst_mr = NULL;
fi_mr_reg(domain, dst_buffer, sizeof(dst_buffer),
    0, 0, 0, flags, &dst_mr, NULL);
uint64_t key = fi_mr_key(dst_mr); /* remote mr key */

union fi_accelerator
{
    struct fi_context2  context;
    struct {
        uint16_t        id;
        uint16_t        options;
        uint32_t        input[7];
        uint64_t        reserved[4];
    };
};
```

```
union fi_accelerator context;

memset((void*)&context, sizeof(union fi_accelerator), 0);
context.id = FI_ACCELERATOR_CRC;
context.input[0] = 0x01234567; /* initial CRC value */


/*
 * invoke the CRC accelerator
 */
fi_write(ep,
    src_data,          /* source buffer virtual address */
    sizeof(src_data),  /* size of the source buffer */
    desc,              /* op descriptor; see FI_MR_LOCAL */
    self,              /* endpoint address */
    0,                 /* remote buffer offset */
    key,               /* access key */
    (void*)&context);  /* user-defined context */

/* ========================================================
 * Because the FI_ACCELERATION flag is set at the endpoint
 * for all data transfer operations, the COPA provider will
 * inspect the memory of the context parameter for the
 * additional accelerator input parameters.
 *
 * The special accelerator id FI_ACCELERATOR_NONE can be
 * specified to invoke the regular, non-accelerated, data
 * movement operation.
 * ======================================================== */
```


© OpenFabrics Alliance

2020 OFA Virtual Workshop

THANK YOU