Open Fabrics Interfaces Architecture Introduction

Sean Hefty

Intel Corporation

Current State of Affairs

OFED software

- Widely adopted low-level RDMA API
- Ships with upstream Linux

but…

- OFED SW was not designed around HPC
- Hardware and fabric features are changing
  - Divergence is driving competing APIs
- Interfaces are being extended, and new APIs introduced
  - Long delay in adoption
- Size of clusters and single node core counts greatly increasing
- More applications are wanting to take advantage of high-performance fabrics

Solution

Evolve OpenFabrics

Design software interfaces that are aligned with application requirements

- Target needs of HPC
- Support multiple interface semantics
- Fabric and vendor agnostic
- Supportable in upstream Linux

Enabling through OpenFabrics

- Leveraging existing open source community
- Broad ecosystem
  - Application developers and vendors
  - Active community engagement
- Drive high-performance software APIs
  - Take advantage of available hardware features
  - Support multiple product generations

Open Fabrics Interfaces
Working Group

OFIWG Charter

- Develop an ***extensible***, open source framework and ***interfaces aligned with ULP and application*** needs for ***high-performance*** fabric services
- Software leading hardware
  - Enable future hardware features
  - Minimal impact to applications
- Minimize impedance match between ULPs and network APIs
- Craft optimal APIs
  - Detailed analysis on MPI, SHMEM, and other PGAS languages
  - Focus on other applications – storage, databases, …

Call for Participation

- OFI WG is open participation
  - Contact the ofiwg mail list for meeting details
  - ofiwg@lists.openfabrics.org
- Source code available through github
  - github.com/ofiwg
- Presentations / meeting minutes available from OFA download directory

Help OFI WG understand workload requirements and drive software design

Enable..

Scalability

Reduced cache and memory footprint

- Scalable address resolution and storage
- Tight data structures

High performance

Optimized software path to hardware

- Independent of hardware interface, version, features

App-centric

Analyze application needs

- Implement them in a coherent, concise, high-performance manner

Extensible

More agile development

- Time-boxed, iterative development
- Application focused APIs
- Adaptable

Verbs Semantic Mismatch

Current RDMA APIs

50-60 lines of C-code

```
Allocate WR
Allocate SGE
Format SGE – 3 writes
Format WR – 6 writes
```

generic send call

```
Loop 1
    Checks – 9 branches
    Loop 2
            Check
            Loop 3
    Checks – 3 branches
Checks – 3 branches
```

Reduce setup cost
- Tighter data

Evolved Fabric Interfaces

25-30 lines of C-code

```
Direct call – 3 writes
```

optimized send call

```
Checks – 2 branches
```

Eliminate loops and branches
- Remaining branches predictable

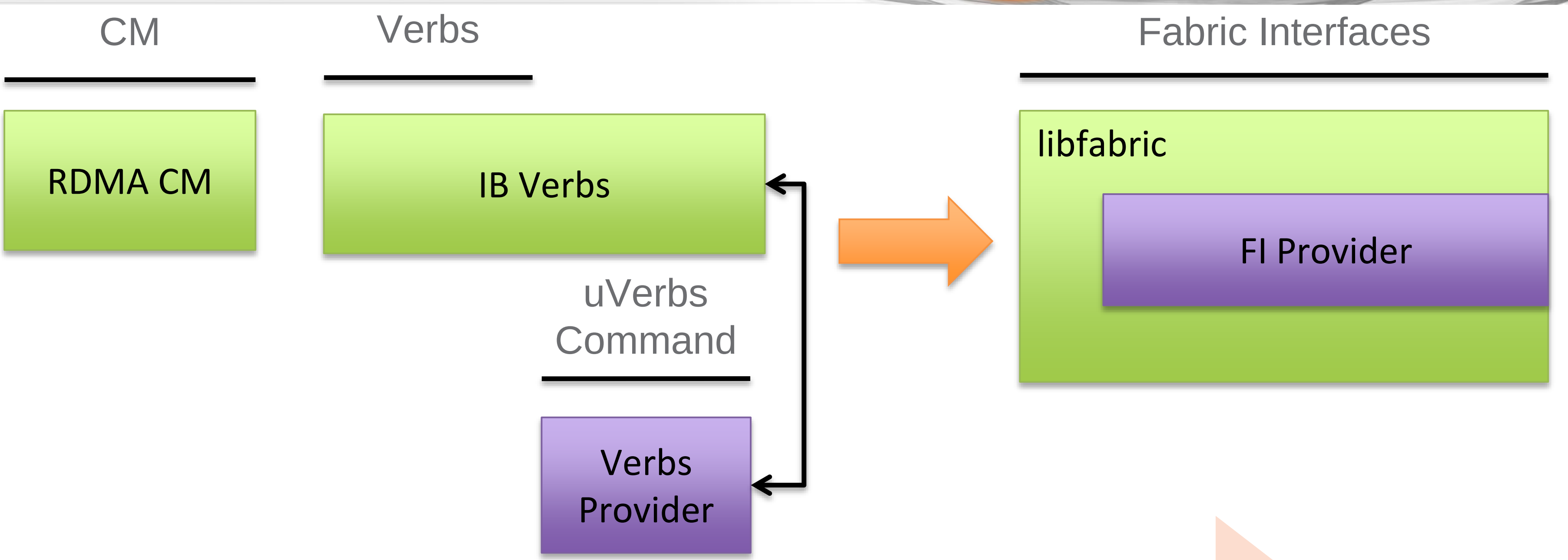
Selective optimization paths to HW
- Manual function expansion

Application-Centric Interfaces

Reducing instruction count *requires* a better application impedance match

- Collect application requirements
- Identify common, fast path usage models
  - Too many use cases to optimize them all
- Build primitives around *fabric services*
  - Not device specific interface

OFA Software Evolution

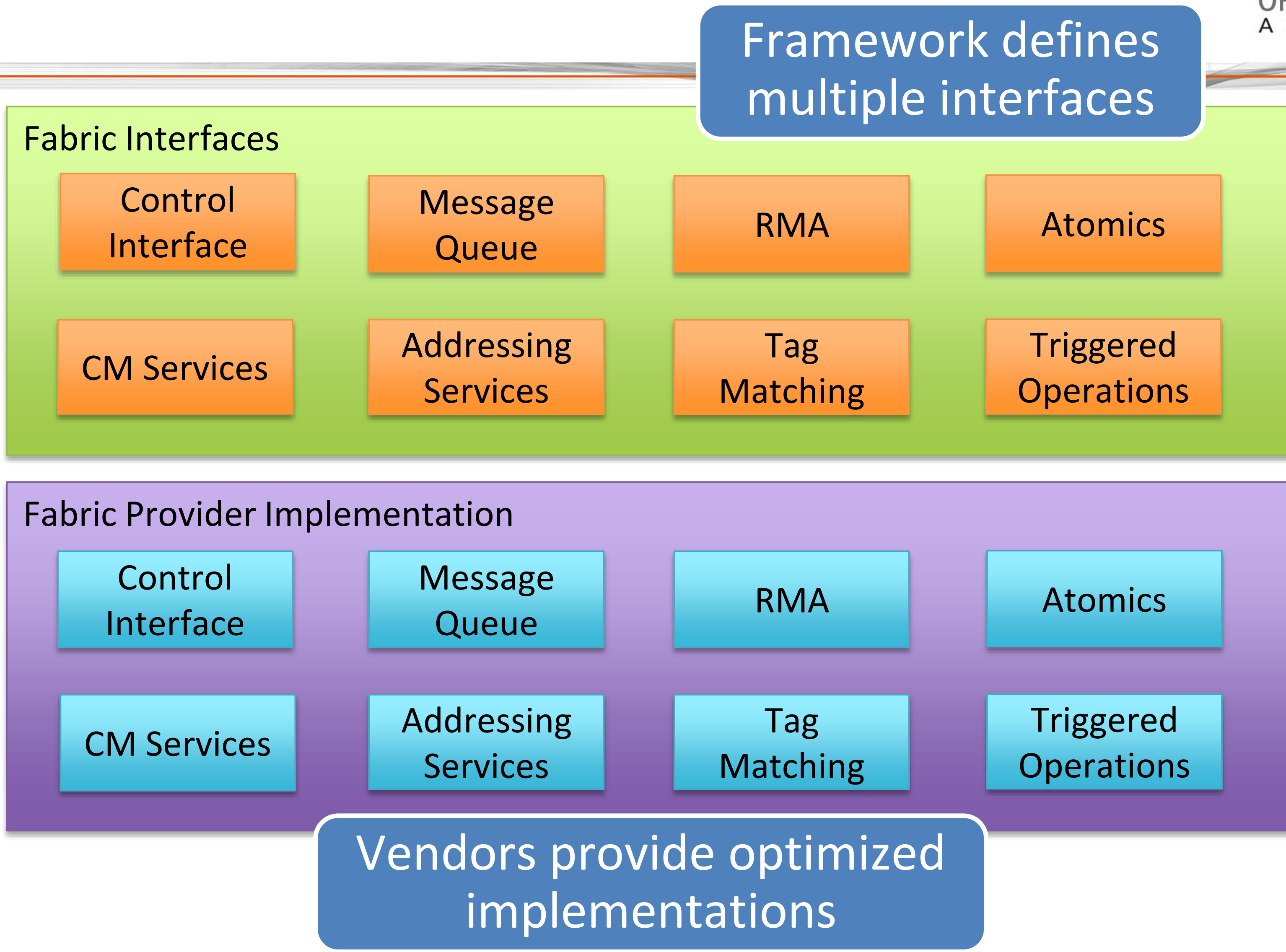


Fabric Interfaces Framework

Focus on longer-lived interfaces – software leading hardware

- Take growth into consideration
- Reduce effort to incorporate new *application* features
  - Addition of new interfaces, structures, or fields
  - Modification of existing functions
- Allow time to design new interfaces correctly
  - Support prototyping interfaces prior to integration

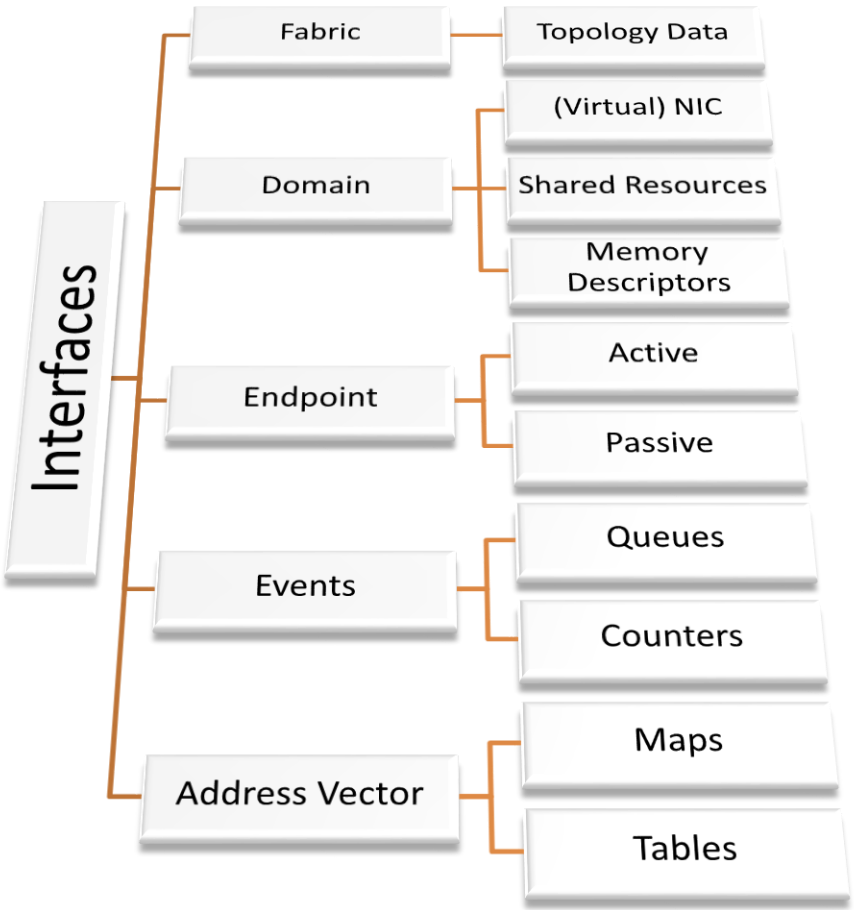
Fabric Interfaces



Fabric Interfaces

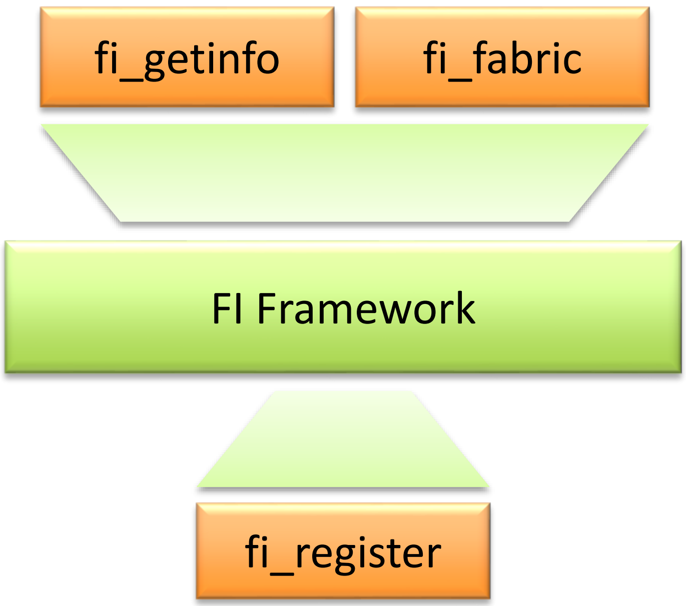
- Defines philosophy for interfaces and extensions
  - Focus interfaces on the *semantics* and *services* offered by the hardware and not the hardware implementation
- *Exports* a minimal API
  - Control interface
- Defines *fabric interfaces*
  - API sets for specific functionality
- Defines core object model
  - Object-oriented design, but C-interfaces

Fabric Interfaces Architecture



- Based on object-oriented programming concepts
- Derived objects define interfaces
  - New interfaces exposed
  - Define behavior of inherited interfaces
  - Optimize implementation

Control Interfaces



fi_getinfo

- Application specifies desired functionality
- Discover fabric providers and services
- Identify resources and addressing

fi_fabric

- Open a set of fabric interfaces and resources

fi_register

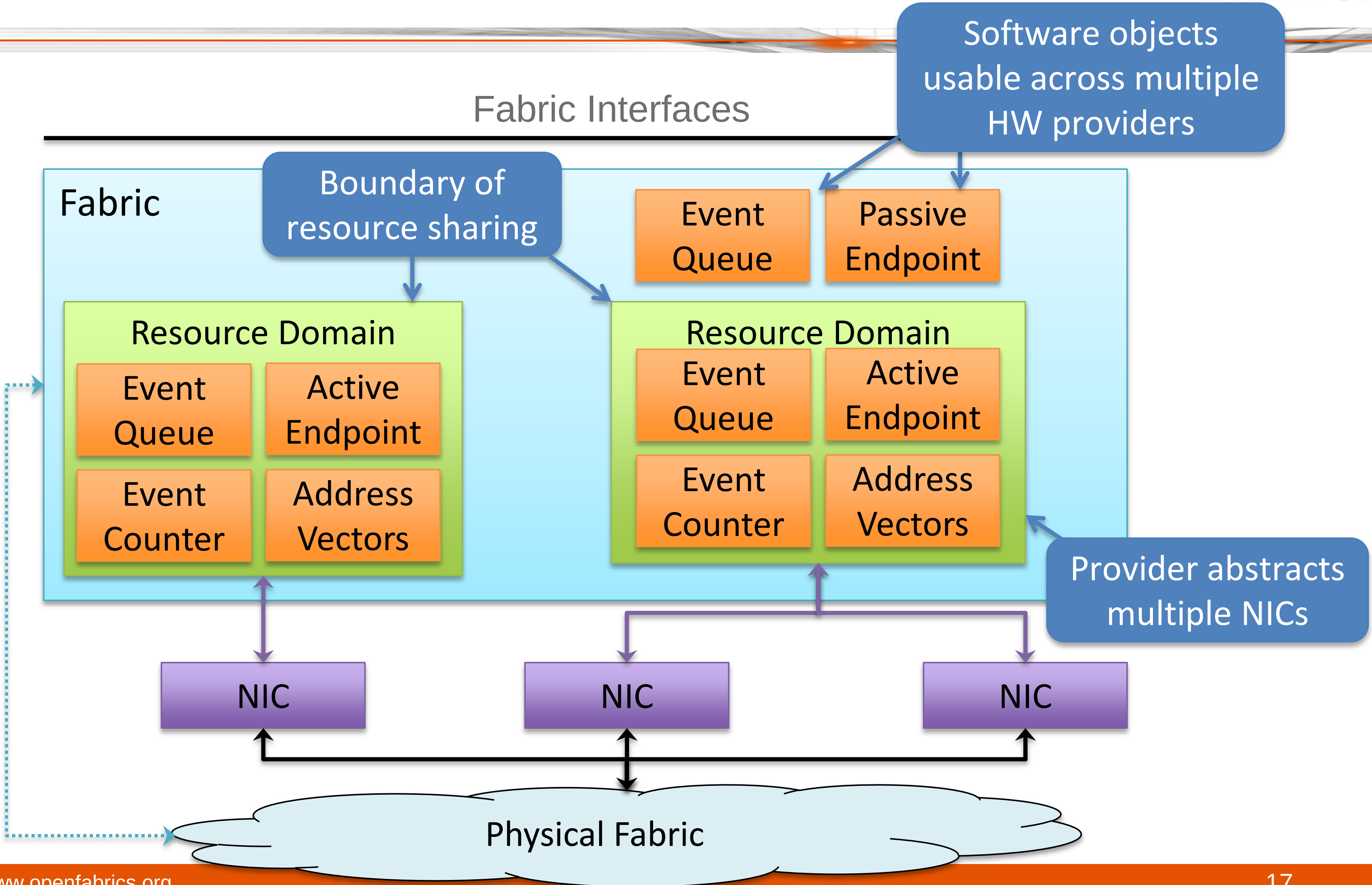
- Dynamic providers publish control interfaces

Application Semantics

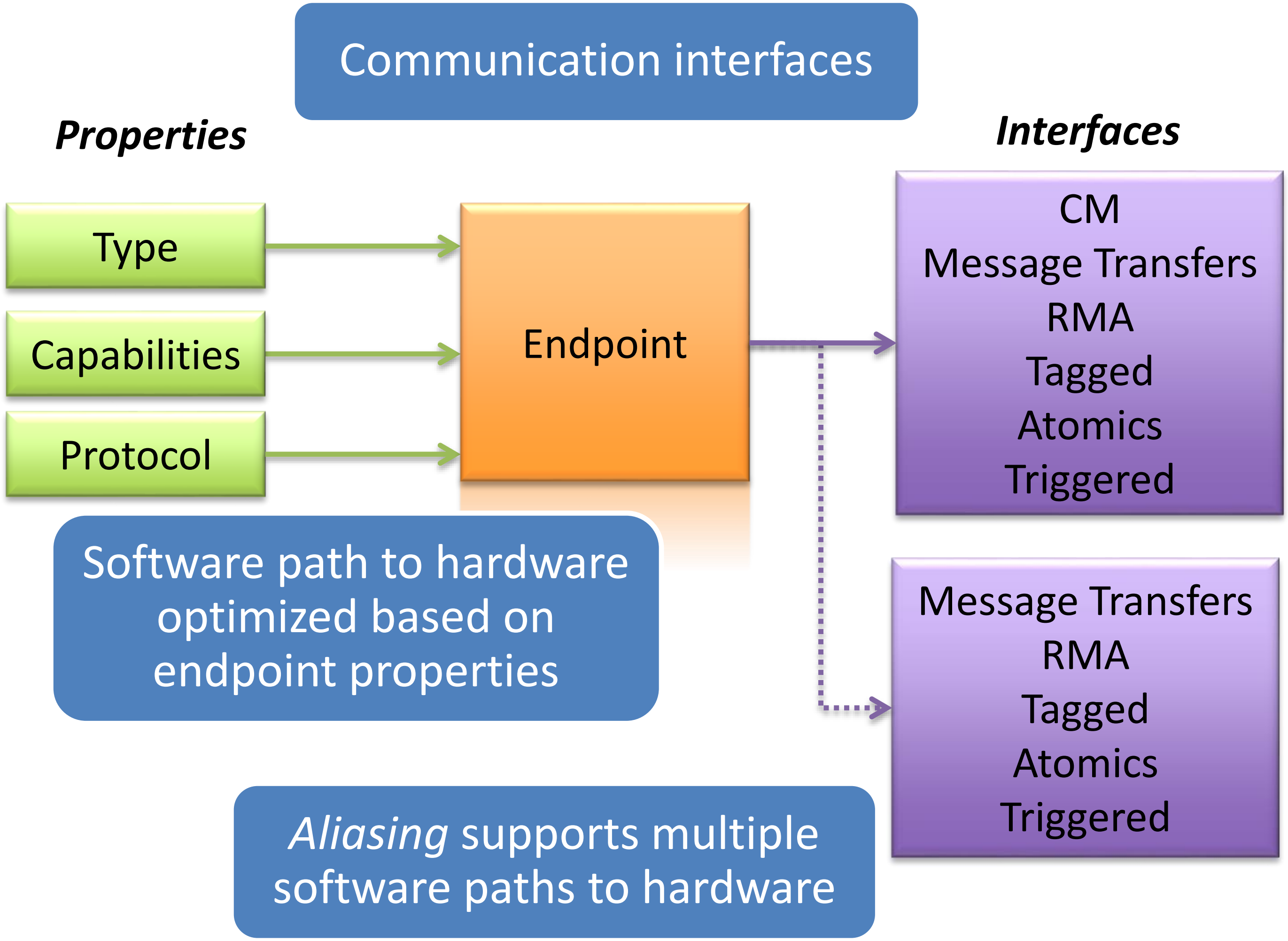
Get / set using control interfaces

- Progress
  - Application or hardware driven
  - Data versus control interfaces
- Ordering
  - Message ordering
  - Data delivery order
- Multi-threading and locking model
  - Compile and run-time options

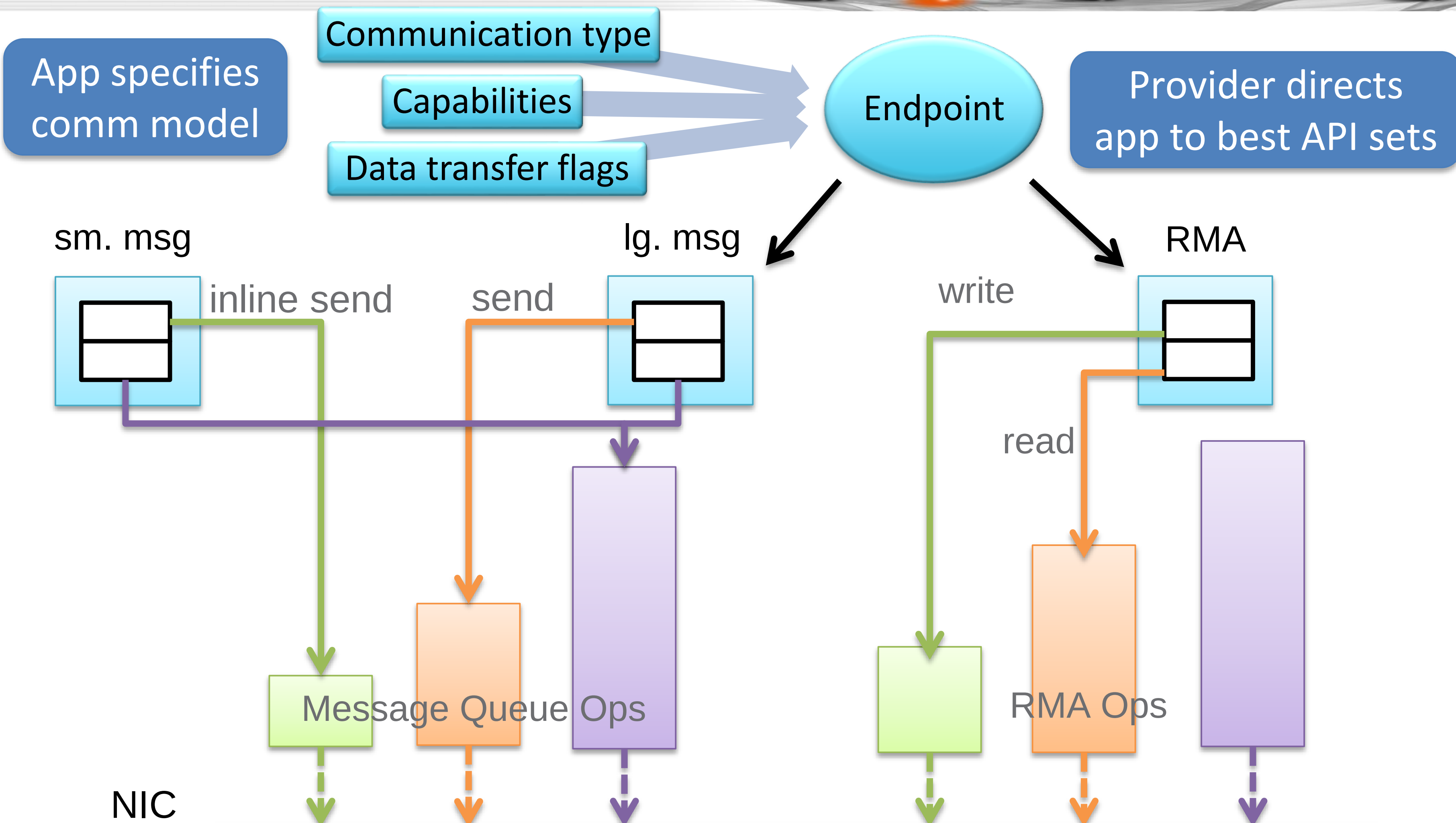
Fabric Object Model



Endpoint Interfaces

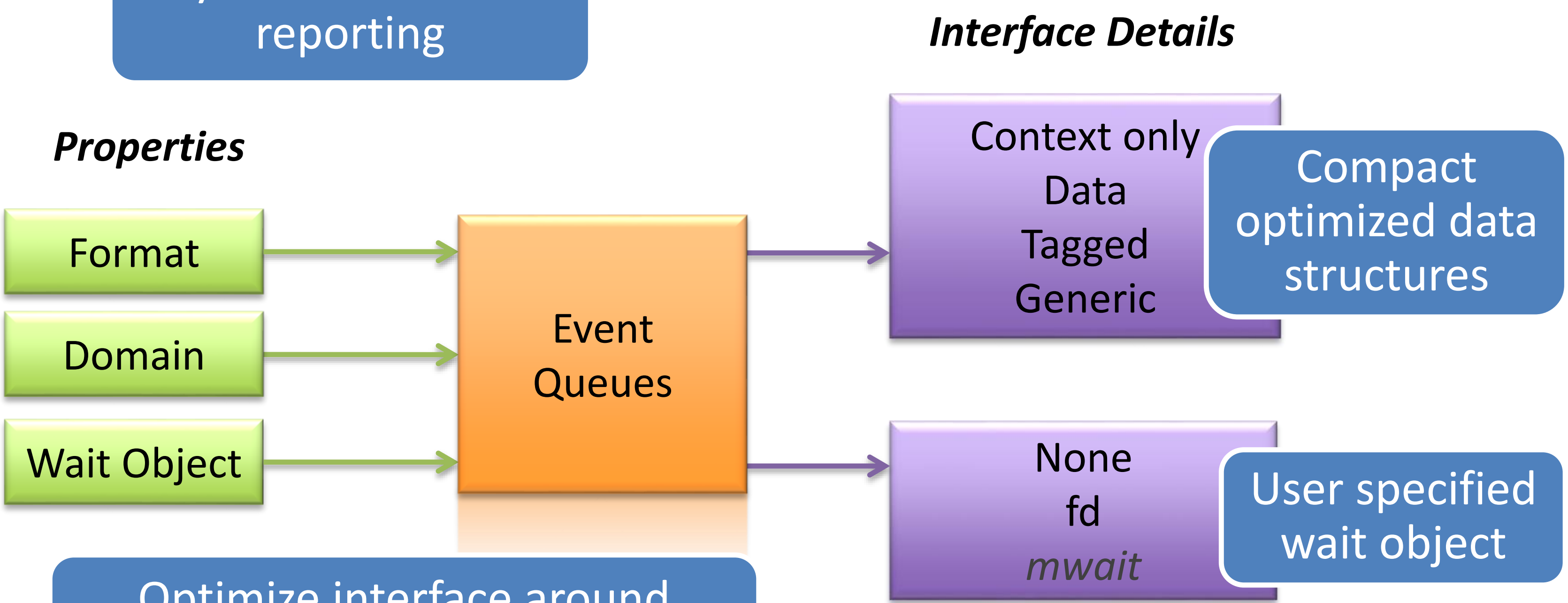


Application Configured Interfaces



Event Queues

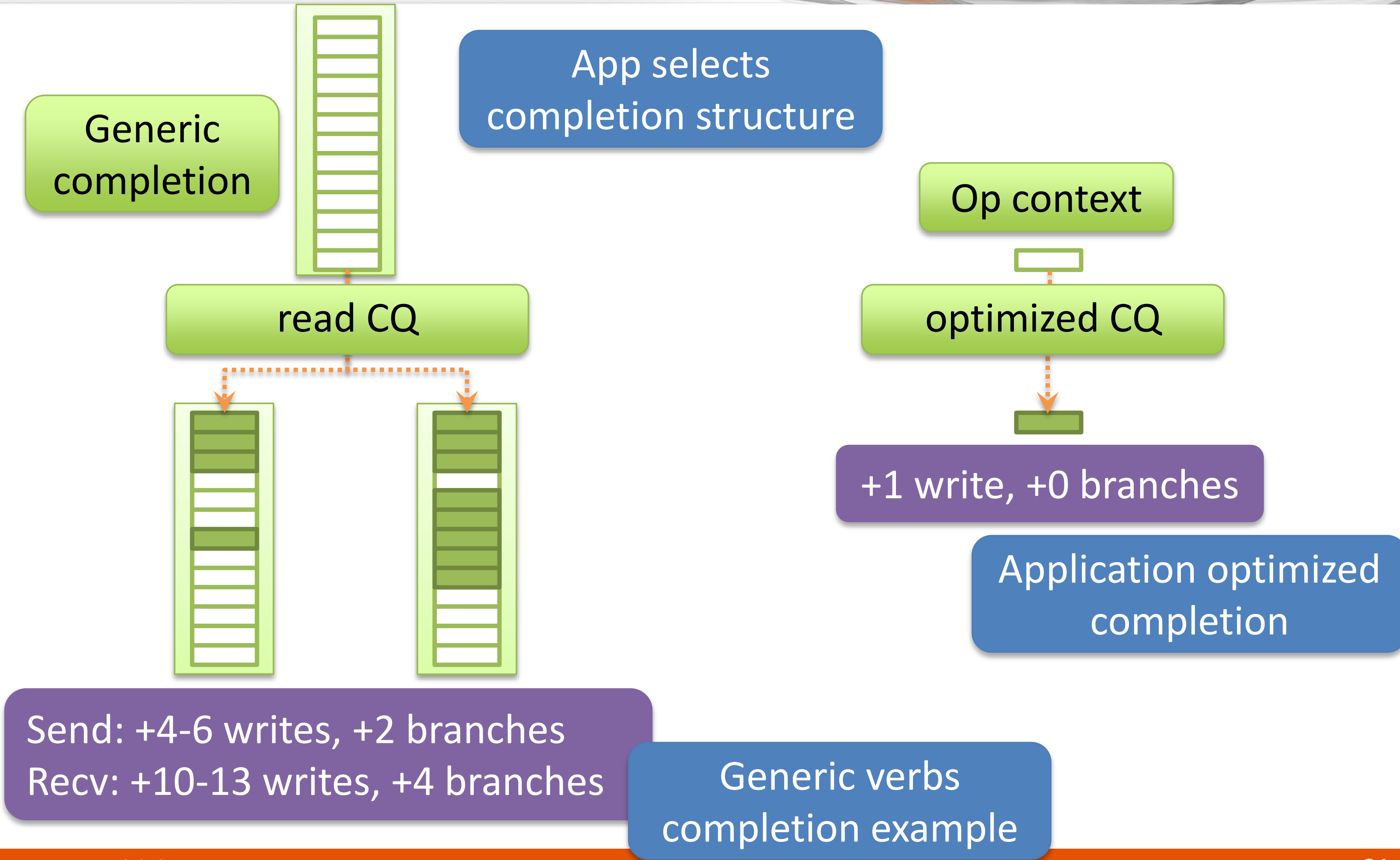
Asynchronous event reporting



Optimize interface around reporting successful operations

Event counters support lightweight event reporting

Event Queues



Address Vectors

Fabric specific addressing requirements

Store addresses/host names
- Insert range of addresses with single call

Share between processes

Reference entries by handle or index
- Handle may be encoded fabric address

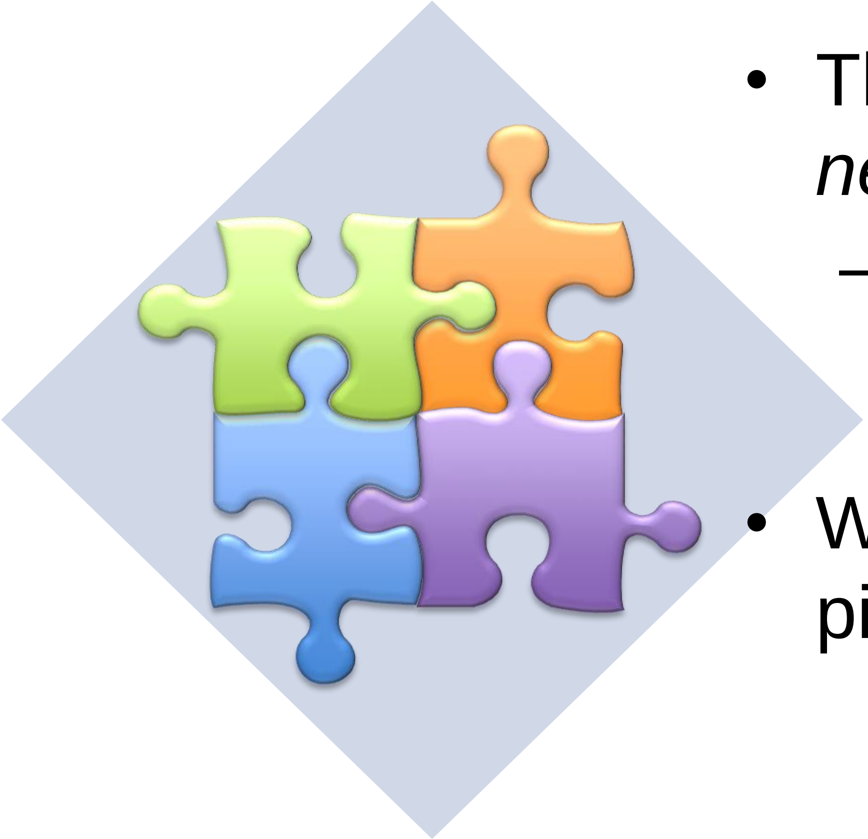
Reference vector for group communication

Example only

Start Range	End Range	Base LID	SL
host10	host1000	50	1
host1001	host4999	2000	2

Enable provider optimization techniques
- Greatly reduce storage requirements

Summary



- These concepts are *necessary*, not revolutionary
  - Communication addressing, optimized data transfers, app-centric interfaces, future looking
- Want a solution where the pieces fit tightly together

Repeated Call for Participation

- Co-chair (sean.hefty@intel.com)
  - Meets Tuesdays from 9-10 PST / 12-1 EST
- Links
  - Mailing list subscription
    - http://lists.openfabrics.org/mailman/listinfo/ofiwg
  - Document downloads
    - https://www.openfabrics.org/downloads/OFIWG/
  - libfabric source tree
    - www.github.com/ofiwg/libfabric
  - labfabric sample programs
    - www.github.com/ofiwg/fabtests

Backup

Verbs API Mismatch

Significant SW overhead

Application request

<buffer, length, context>

```
struct ibv_sge {
    uint64_t    addr;
    uint32_t    length;
    uint32_t    lkey;
};

struct ibv_send_wr {
    uint64_t            wr_id;
    struct ibv_send_wr *next;
    struct ibv_sge      *sg_list;
    int                 num_sge;
    enum ibv_wr_opcode  opcode;
    int                 send_flags;
    uint32_t            imm_data;
    ...
};
```

3 x 8 = 24 bytes of data needed

SGE + WR = 88 bytes allocated

Requests may be linked - next must be set to NULL

Must link to separate SGL and initialize count

App must set and provider must switch on opcode

Must clear flags

28 additional bytes initialized

Verbs Provider Mismatch

```
For each work request
    Check for available queue space
    Check SGL size
    Check valid opcode
    Check flags x 2
    Check specific opcode
    Switch on QP type
        Switch on opcode
    Check flags
        For each SGE
            Check size
            Loop over length
    Check flags
    Check
    Check for last request
Other checks x 3
```

- For each work request ← Most often 1 (overlap operations)
- Check SGL size ← Often 1 or 2 (fixed in source)
- Check valid opcode ← Artifact of API
- Switch on QP type ← QP type usually fixed in source
- Check flags ← Flags may be fixed or app may have taken branches

19+ branches including loops

100+ lines of C code
50-60 lines of code to HW

Verbs Completions Mismatch

Application accessed fields

```
struct ibv_wc {
    uint64_t      wr_id;
    enum ibv_wc_status   status;
    enum ibv_wc_opcode   opcode;
    uint32_t      vendor_err;
    uint32_t      byte_len;
    uint32_t      imm_data;
    uint32_t      qp_num;
    uint32_t      src_qp;
    int                  wc_flags;
    uint16_t      pkey_index;
    uint16_t      slid;
    uint8_t              sl;
    uint8_t              dlid_path_bits;
};
```

App must check both return code and status to determine if a request completed successfully

Provider must fill out all fields, even those ignored by the app

Provider must handle all types of completions from any QP

Developer must determine if fields apply to their QP

Single structure is 48 bytes likely to cross cacheline boundary

RDMA CM Mismatch

Want: reliable data transfers, zero copies to thousands of processes

RDMA interfaces expose:

```
struct rdma_route {
    struct rdma_addr          addr;
    struct ibv_sa_path_rec *path_rec;
    ...
};

struct rdma_cm_id {...};

rdma_create_id()
rdma_resolve_addr()
rdma_resolve_route()
rdma_connect()
```

- Src/dst addresses stored per endpoint
- 456 bytes per endpoint
- Path record per endpoint
- Resolve single address and path at a time
- All to all connected model for best performance

Progress

- Ability of the underlying implementation to complete processing of an asynchronous request
- Need to consider **ALL** asynchronous requests
  - Connections, address resolution, data transfers, event processing, completions, etc.
- HW/SW mix

All(?) current solutions require significant software components

Progress

- Support two progress models
  - Automatic and implicit
- Separate operations as belonging to one of two progress domains
  - Data or control
  - Report progress model for each domain

SAMPLE	Implicit	Automatic
Data	Software	Hardware offload
Control	Software	Kernel services

Automatic Progress

- Implies hardware offload model
  - Or standard kernel services / threads for control operations
- Once an operation is initiated, it will complete without further user intervention or calls into the API
- Automatic progress meets implicit model by definition

Implicit Progress

- Implies significant software component
- Occurs when reading or waiting *on EQ*(s)
- Application can use separate EQs for control and data
- Progress limited to objects associated with selected EQ(s)
- App can request automatic progress
  - E.g. app wants to wait on native wait object
  - Implies provider allocated threading

Ordering

- Applies to a single initiator endpoint performing data transfers to one target endpoint over the same data flow
  - Data flow may be a conceptual QoS level or path through the network
- Separate ordering domains
  - Completions, message, data
- Fenced ordering may be obtained using fi_sync operation

Completion Ordering

- Order in which operation completions are reported relative to their submission
- Unordered or ordered
  - No defined requirement for ordered completions
- Default: unordered

Message Ordering

- Order in which message (transport) headers are processed
  – I.e. whether transport message are received in or out of order
- Determined by selection of ordering bits
  – [Read | Write | Send]   After   [Read | Write | Send]
  – RAR, RAW, RAS, WAR, WAW, WAS, SAR, SAW, SAS
- Example:
  – fi_order = 0 // unordered
  – fi_order = RAR | RAW | RAS | WAW | WAS |
  SAW | SAS   // IB/iWarp like ordering

Data Ordering

- Delivery order of transport data into target memory
  - Ordering per byte-addressable location
  - I.e. access to the same byte in memory
- Ordering constrained by message ordering rules
  - Must at least have message ordering first

Data Ordering

- Ordering limited to message order size
  - E.g. MTU
  - In order data delivery if transfer <= message order size
  - WAW, RAW, WAR sizes?
- Message order size = 0
  - No data ordering
- Message order size = -1
  - All data ordered

Other Ordering Rules

- Ordering to different target endpoints not defined
- Per message ordering semantics implemented using different data flows
  - Data flows may be less flexible, but easier to optimize for
  - Endpoint aliases may be configured to use different data flows

Multi-threading and Locking

- Support both thread safe and lockless models
  – Compile time and run time support
  – Run-time limited to compiled support
- Lockless (based on MPI model)
  – Single – single-threaded app
  – Funneled – only 1 thread calls into interfaces
  – Serialized – only 1 thread at a time calls into interfaces
- Thread safe
  – Multiple – multi-threaded app, with no restrictions

Buffering

- Support both application and network buffering
  - Zero-copy for high-performance
  - Network buffering for ease of use
    - Buffering in local memory or NIC
  - In some case, buffered transfers may be higher-performing (e.g. “inline”)
- Registration option for local NIC access
  - Migration to fabric managed registration
- Required registration for remote access
  - Specify permissions

Scalable Transfer Interfaces

- *Application* optimized code paths based on usage model
- Optimize call(s) for single work request
  - Single data buffer
  - Still support more complex WR lists/SGL
- Per endpoint send/receive operations
  - Separate RMA function calls
- Pre-configure data transfer flags
  - Known before post request
  - Select software path through provider