Fabric Interfaces Architecture

Sean Hefty - Intel Corporation

Changes

- v2
  - Remove interface object
  - Add open interface as base object
  - Add SRQ object
  - Add EQ group object
- v3
  - Modified SRQ
  - Enhanced architecture semantics

Overview

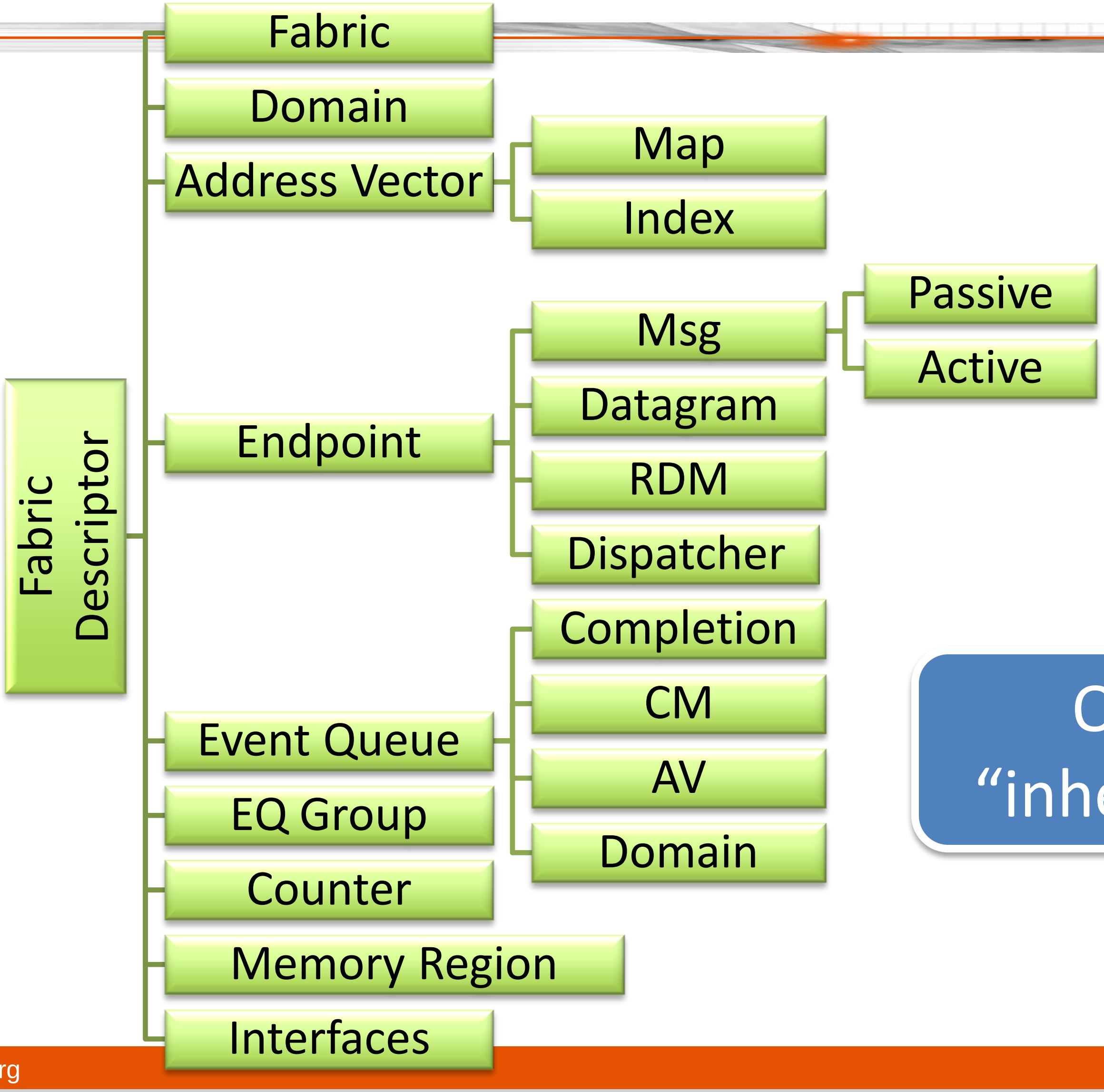
- Object Model
  - Do we have the right type of objects defines?
  - Do we have the correct object relationships?
- Interface Synopsis
  - High-level description of object operations
  - Is functionality missing?
  - Are interfaces associated with the right object?
- Architectural Semantics
  - Do the semantics match well with the apps?
  - What semantics are missing?

Object “Class” Model

- Objects represent collection of attributes and interfaces
  - I.e. object-oriented programming model
- Consider architectural model only at this point

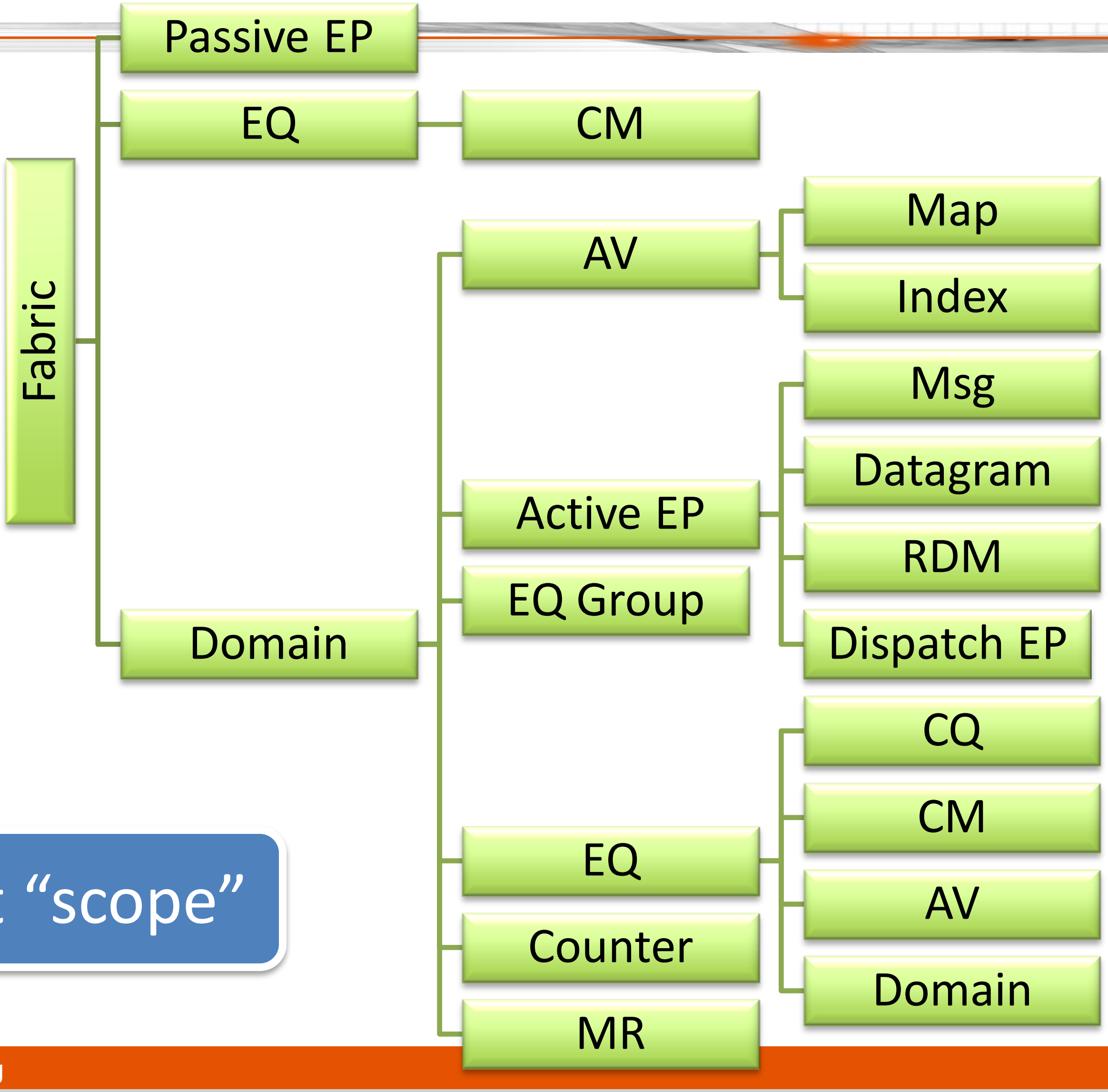
Objects do not necessarily map directly to hardware or software objects

Conceptual Object Hierarchy



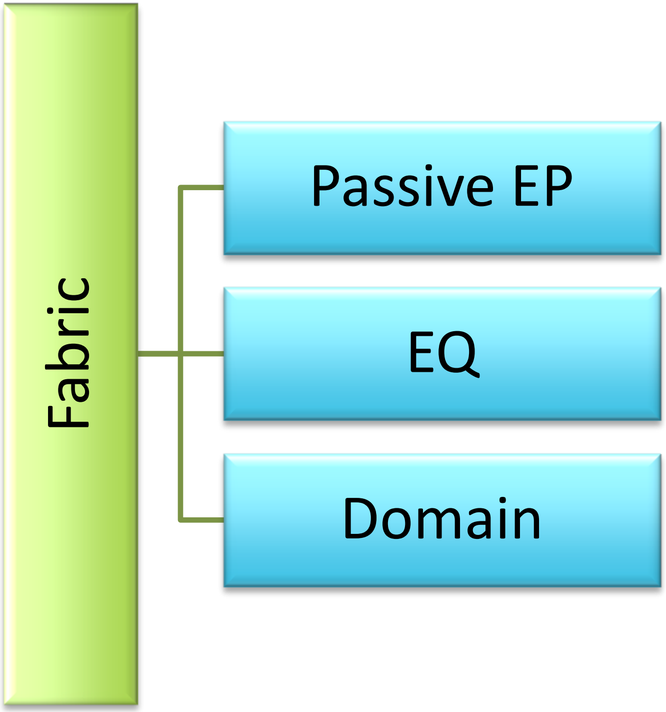
Object “inheritance”

Object Relationships



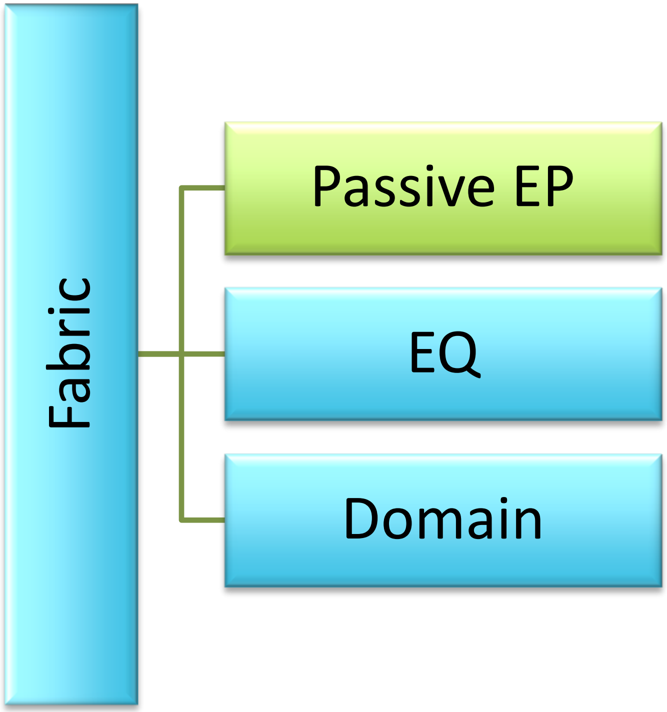
Object “scope”

Fabric



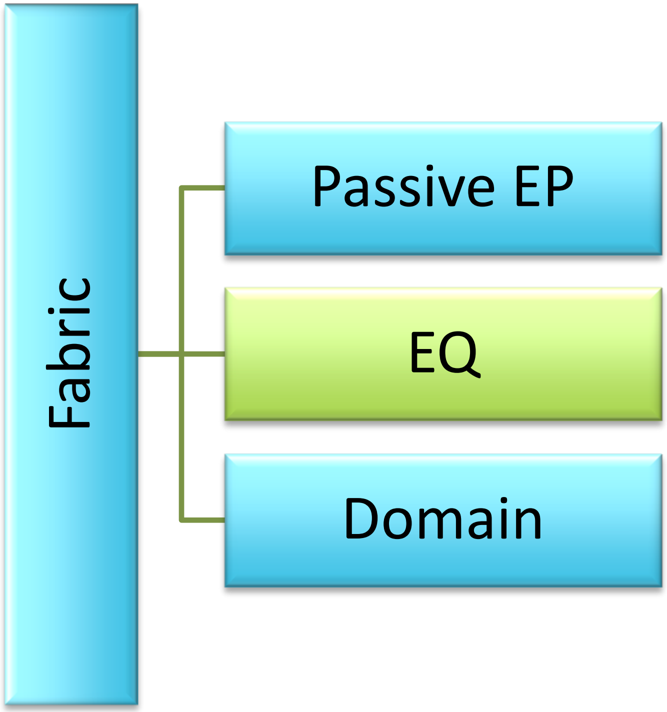
- Represents a communication domain or boundary
  - Single IB or RoCE subnet, IP (iWarp) network, Ethernet subnet
- Multiple local NICs / ports
- Topology data, network time stamps
- Determines native addressing
  - Mapped addressing possible
  - GID/LID versus IP

Passive (Fabric) EP



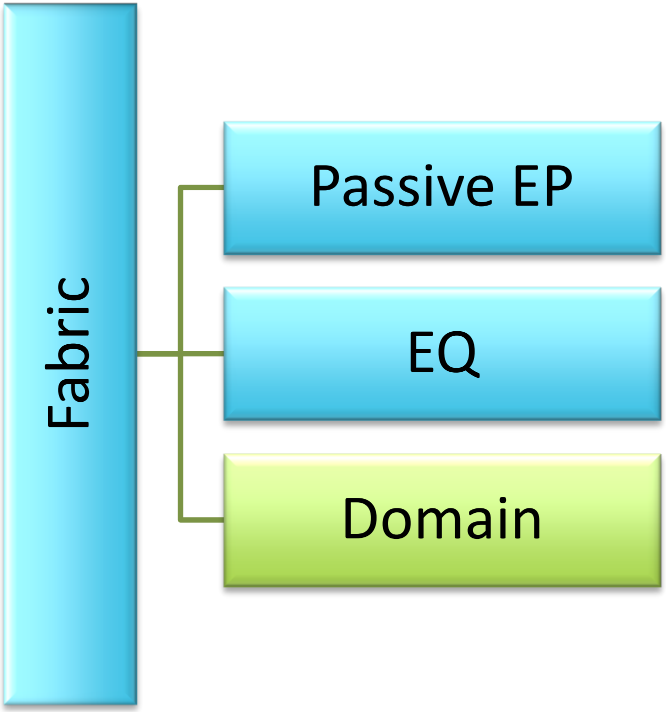
- Listening endpoint
  - Connection-oriented protocols
- Wildcard listen across multiple NICs / ports
- Bind to address to restrict listen
  - Listen may migrate with address

Fabric EQ



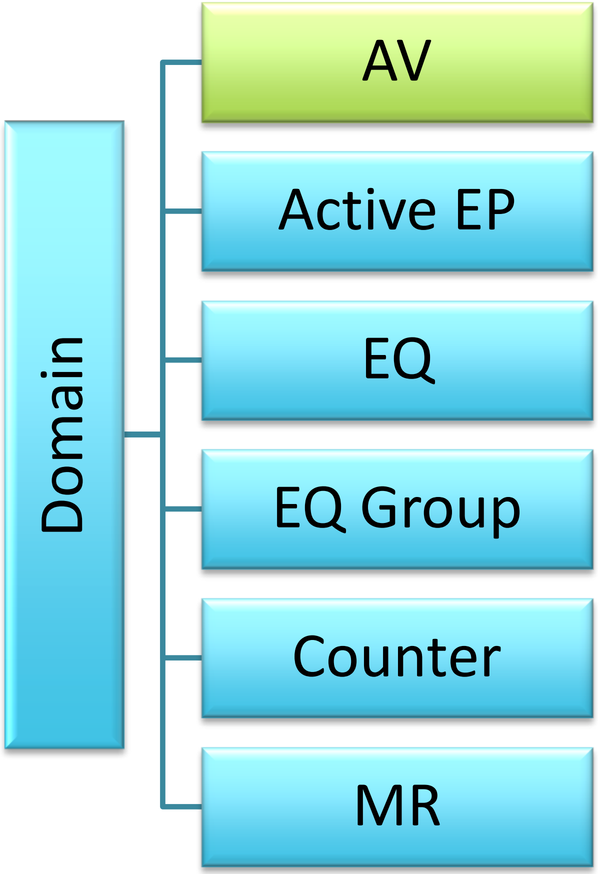
- Associated with passive endpoint(s)
- Reports connection requests
- Could be used to report fabric events

Resource Domain



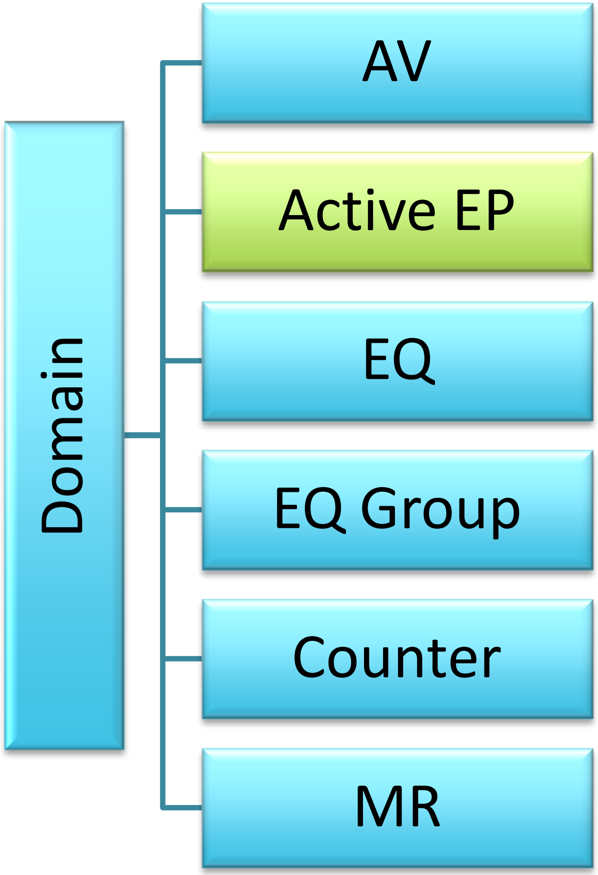
- Boundary for resource sharing
  - Physical or logical NIC
  - Command queue
- Container for data transfer resources
- A provider may define multiple domains for a single NIC
  - Dependent on resource sharing

Domain Address Vectors



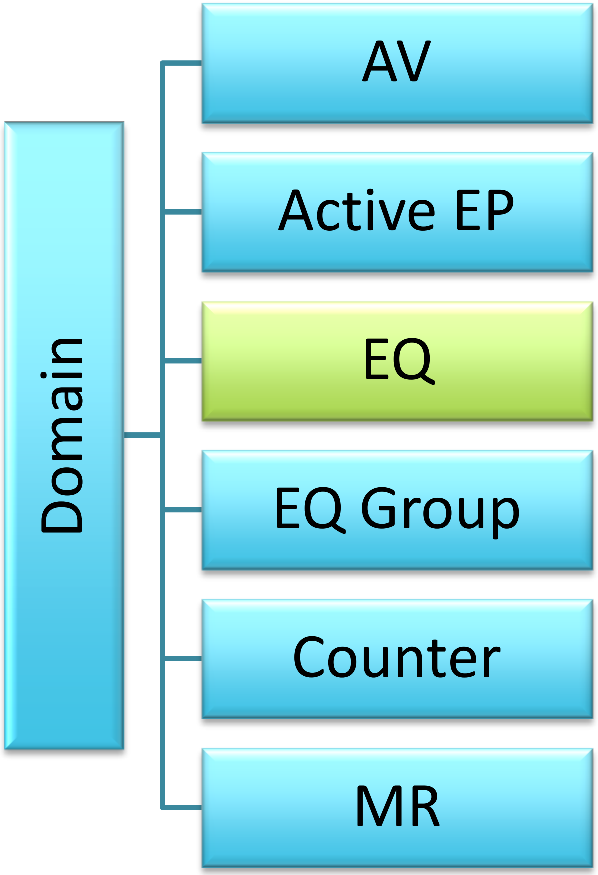
- Maintains list of remote endpoint addresses
  - Map – native addressing
  - Index – ‘rank’-based addressing
- Resolves higher-level addresses into fabric addresses
  - Native addressing abstracted from user
- Handles address and route changes

Domain Endpoints



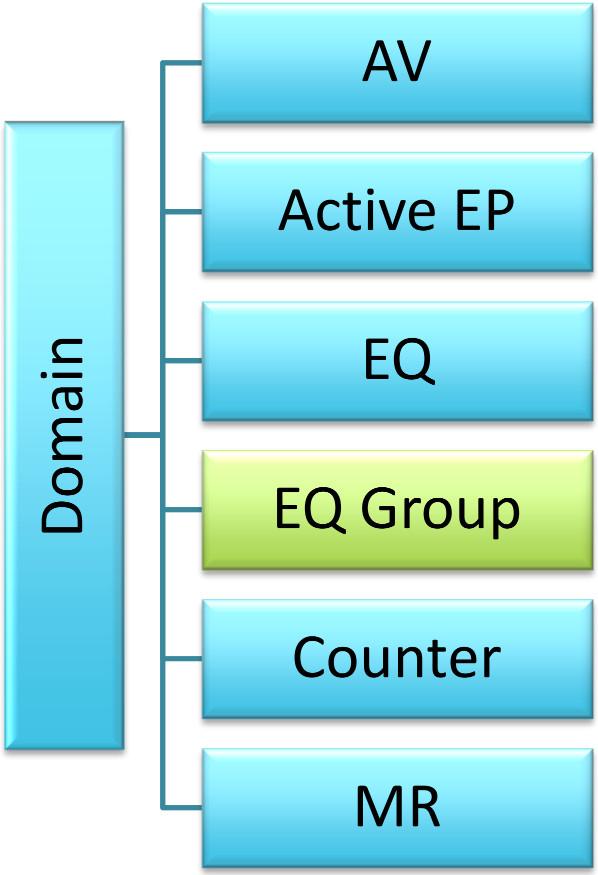
- Data transfer portal
  - Send / receive queues
  - Command queues
  - Ring buffers
  - Buffer dispatching
- Multiple types defined
  - Connection-oriented / connectionless
  - Reliable / unreliable
  - Message / stream

Domain Event Queues



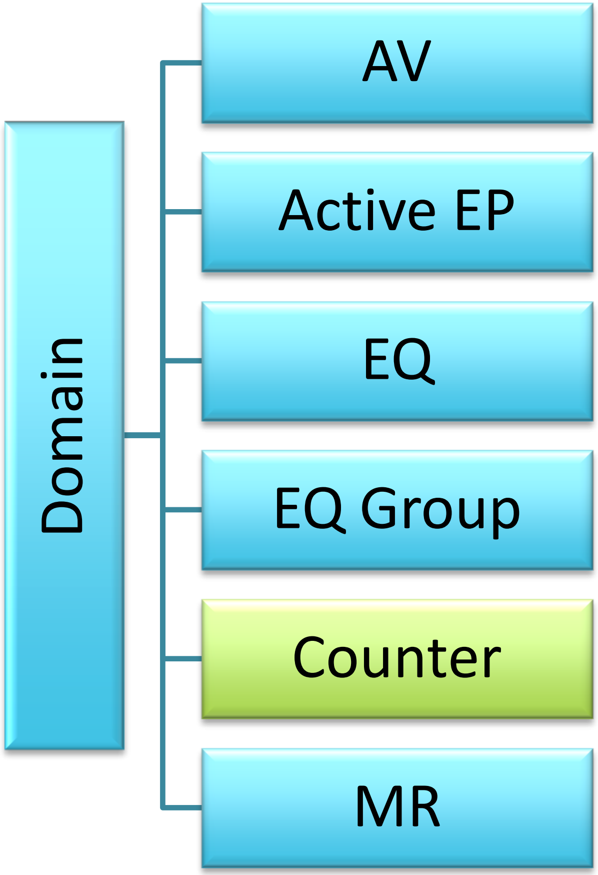
- Reports asynchronous events
- Unexpected errors reported ‘out of band’
- Events separated into ‘EQ domains’
  - CM, AV, completions
  - 1 EQ domain per EQ
  - Future support for merged EQ domains

EQ Groups



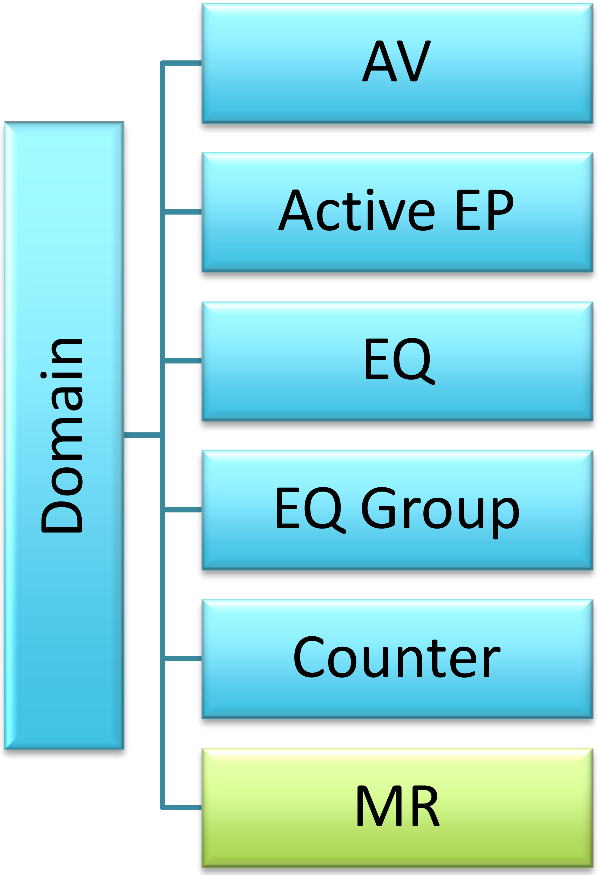
- Collection of EQs
- Conceptually shares same wait object
- Grouping for progress and wait operations

Domain Counters



- Provides a count of successful completions of asynchronous operations
  - Conceptual HW counter
- Count is independent from an actual event reported to the user through an EQ

Domain Memory Regions



- Memory ranges accessible by fabric resources
  - Local and/or remote access
- Defines permissions for remote access

Interface Synopsis

- Operations associated with identified ‘classes’
- General functionality, versus detailed methods
  - The full set of methods are not defined here
  - Detailed behavior (e.g. blocking) is not defined
- Identify missing and unneeded functionality
  - Mapping of functionality to objects

Use timeboxing to limit scope of interfaces to refine by a target date

Base Class	
Close	Destroy / free object
Bind	Create an association between two object instances
Sync	Fencing operation that completes only after previously issued asynchronous operations have completed
Control	(~fcntl) set/get low-level object behavior
I/F Open	Open provider extended interfaces

Fabric	
Domain	Open a resource domain
Endpoint	Create a listening EP for connection-oriented protocols
EQ Open	Open an event queue for listening EP or reporting fabric events

Resource Domain	
Query	Obtain domain specific attributes
Open AV, EQ, EP, SRQ, EQ Group	Create an address vector, event or completion counter, event queue, endpoint, shared receive queue, or EQ group
MR Ops	Register data buffers for access by fabric resources

Address Vector	
Insert	Insert one or more addresses into the vector
Remove	Remote one or more addresses from the vector
Lookup	Return a stored address
Straddr	Convert an address into a printable string

Base EP	
Enable	Enables an active EP for data transfers
Cancel	Cancel a pending asynchronous operation
Getopt	(~getsockopt) get protocol specific EP options
Setopt	(~setsockopt) set protocol specific EP options

Passive EP	
Getname	(~getsockname) return EP address
Listen	Start listening for connection requests
Reject	Reject a connection request

Active EP	
CM	Connection establishment ops, usable by connection-oriented and connectionless endpoints
MSG	2-sided message queue ops, to send and receive messages
RMA	1-sided RDMA read and write ops
Tagged	2-sided matched message ops, to send and receive messages (conceptual merge of messages and RMA writes)
Atomic	1-sided atomic ops
Triggered	Deferred operations initiated on a condition being met

Event Queue	
Read	Retrieve a completion event, and optional source endpoint address data for received data transfers
Read Err	Retrieve event data about an operation that completed with an unexpected error
Write	Insert an event into the queue
Reset	Directs the EQ to signal its wait object when a specified condition is met
Strerror	Converts error data associated with a completion into a printable string

EQ Group	
Poll	Check EQs for events
Wait	Wait for an event on the EQ group

Completion Counter	
Read	Retrieve a counter’s value
Add	Increment a counter
Set	Set / clear a counter’s value
Wait	Wait until a counter reaches a desired threshold

Memory Region	
Desc	(~lkey) Optional local memory descriptor associated with a data buffer
Key	(~rkey) Protection key against access from remote data transfers

Architectural Semantics

Need refining

- Progress
- Ordering - completions and data delivery
- Multi-threading and locking model
- Buffering
- Function signatures and semantics

Once defined, object and interface semantics cannot change – semantic changes require new objects and interfaces

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 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
- 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