Implementing the CCA Event Service for HPC

Ian Gorton, Daniel Chavarria, Manoj Krishnan, Jarek Nieplocha

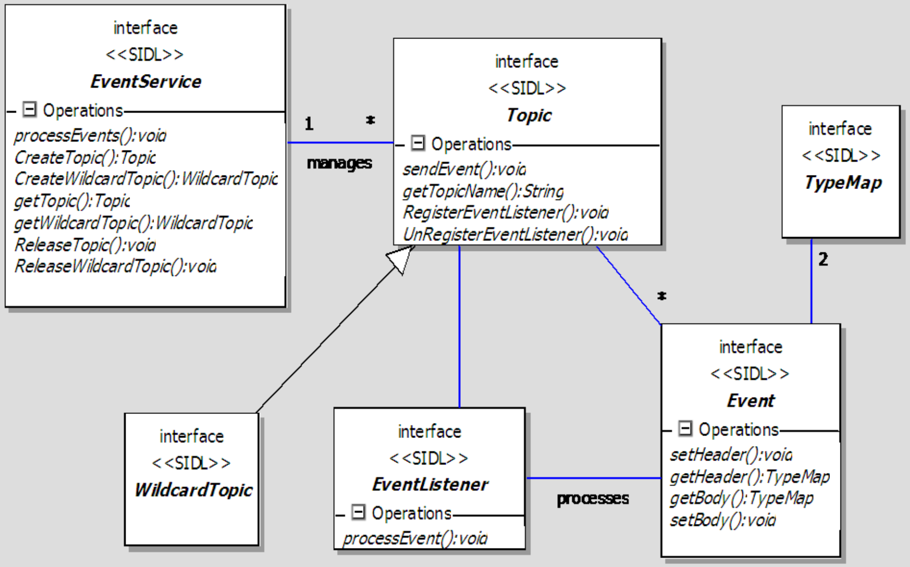
Pacific Northwest National Lab

CCA 101

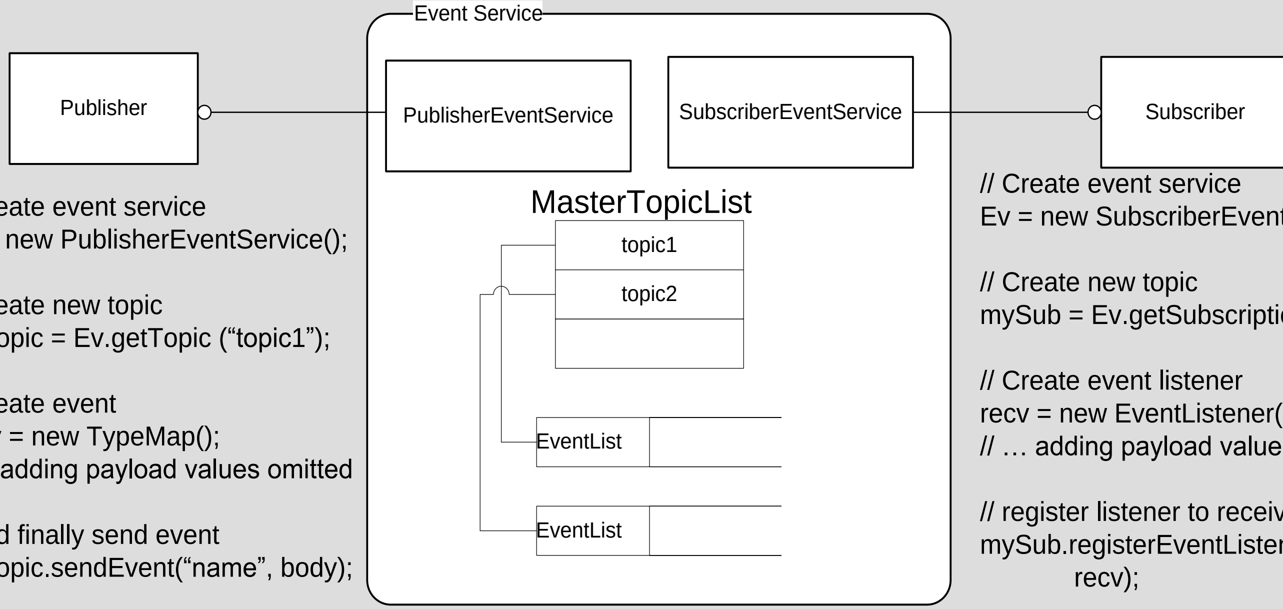
- Component architecture for HPC
- Components have *provides* and *requires* ports
- A CCA compliant framework configures component connections and launches computation
- Component model current supports SCMD approach

CCA Event Service 101

- Publish-subscribe
  - 1-n, n-m, n-1
- Specification is similar to:
  - Java Messaging Service
  - Many distributed event/messaging services



Conceptual Architecture



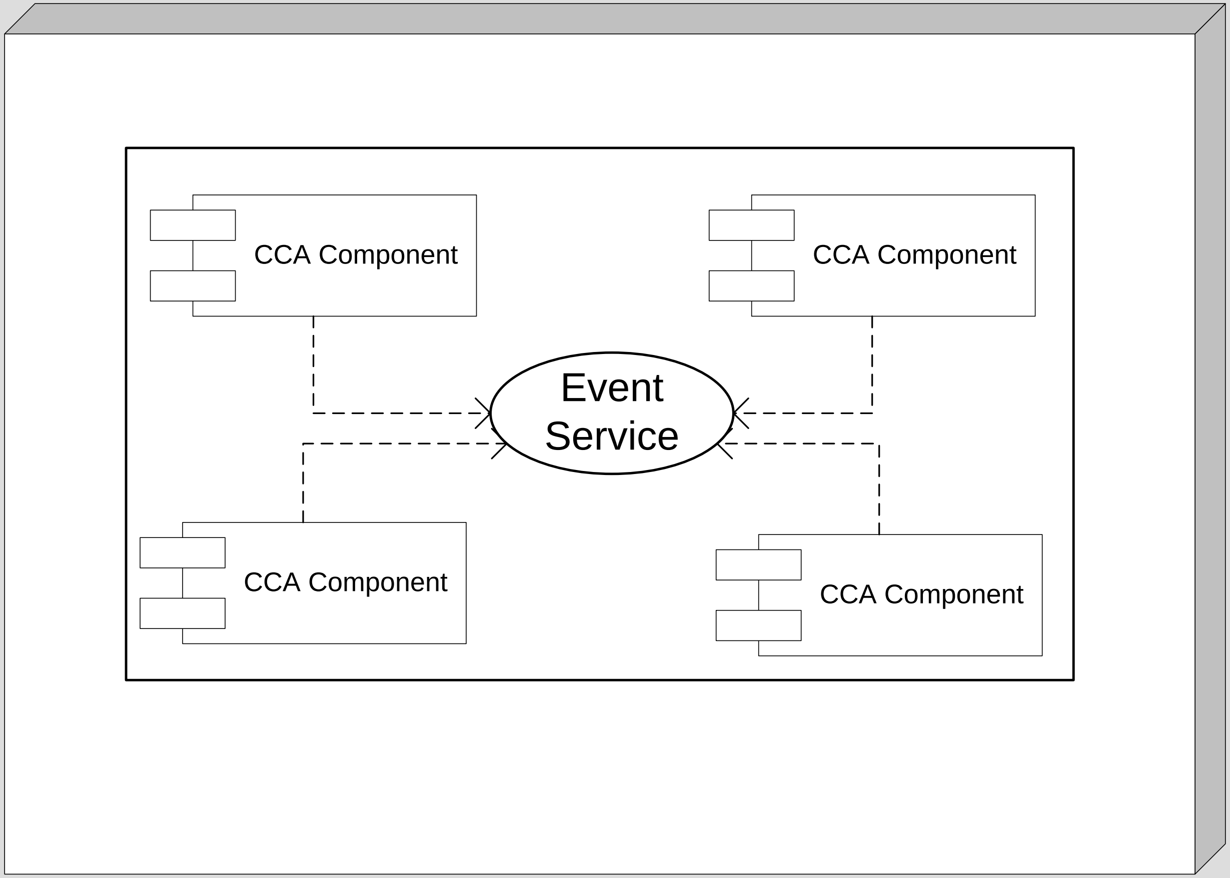

```
// Create event service
Ev = new PublisherEventService();

// Create new topic
myTopic = Ev.getTopic ("topic1");

// Create event
body = new TypeMap();
// … adding payload values omitted

// and finally send event
myTopic.sendEvent("name", body);
```

```
// Create event service
Ev = new SubscriberEventService();

// Create new topic
mySub = Ev.getSubscription ("topic1");

// Create event listener
recv = new EventListener();
// … adding payload values omitted

// register listener to receive events
mySub.registerEventListener ("key",
          recv);

// and enter event loop to deliver events
While (running)
    ev.processEvents()
```

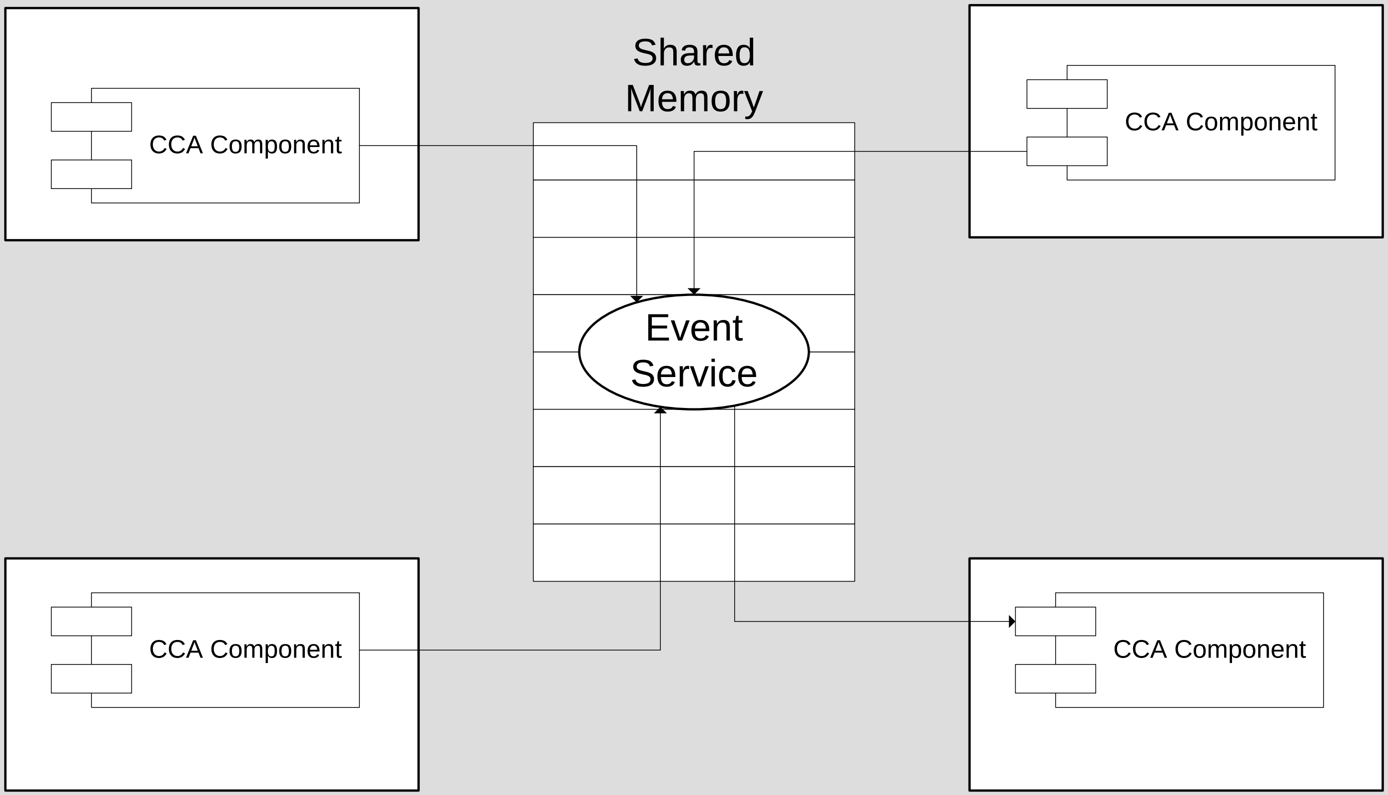
Possible use cases

- Potential for a standard API for events/messaging
  - Same address space
  - Across address spaces
  - Needs to be fast
  - Handle a range of potential payload sizes
    - Event/messaging service schizophrenia!!
- Other work exists …
  - ECho
  - Grid event service
  - Many others

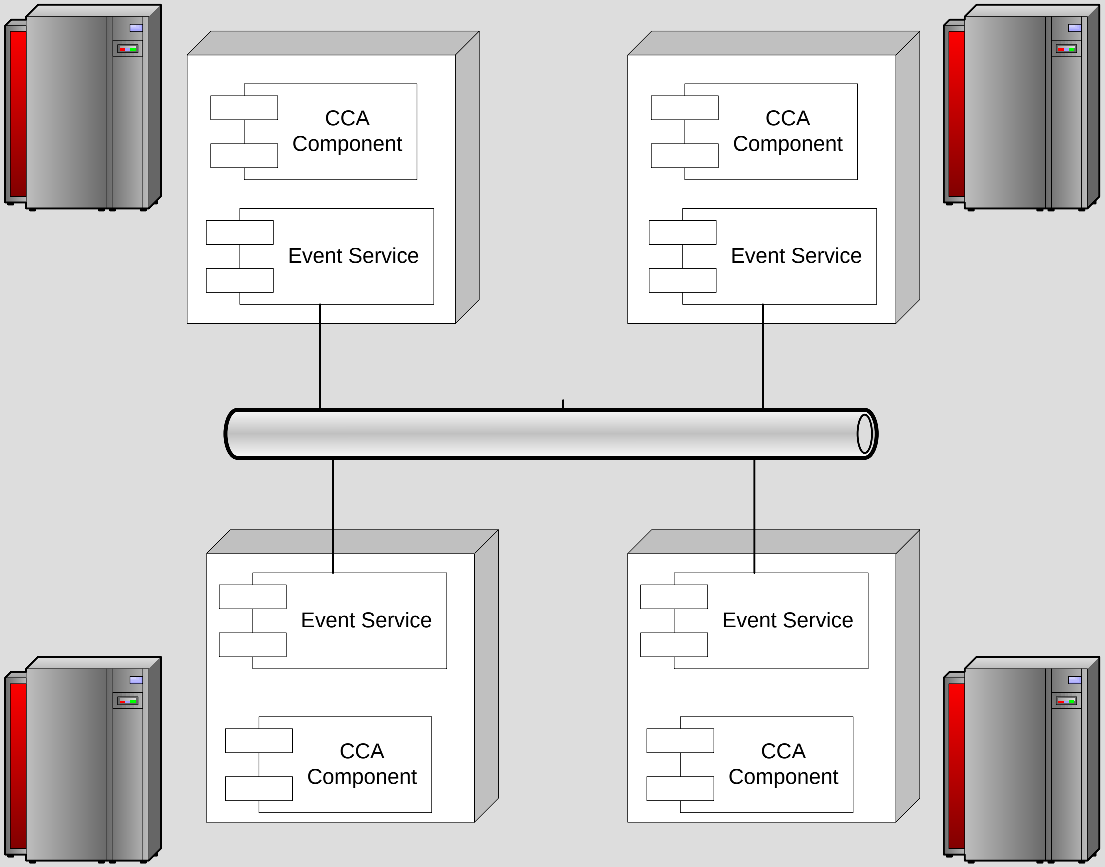
Same Address Space/Process



Multiple processes, same platform



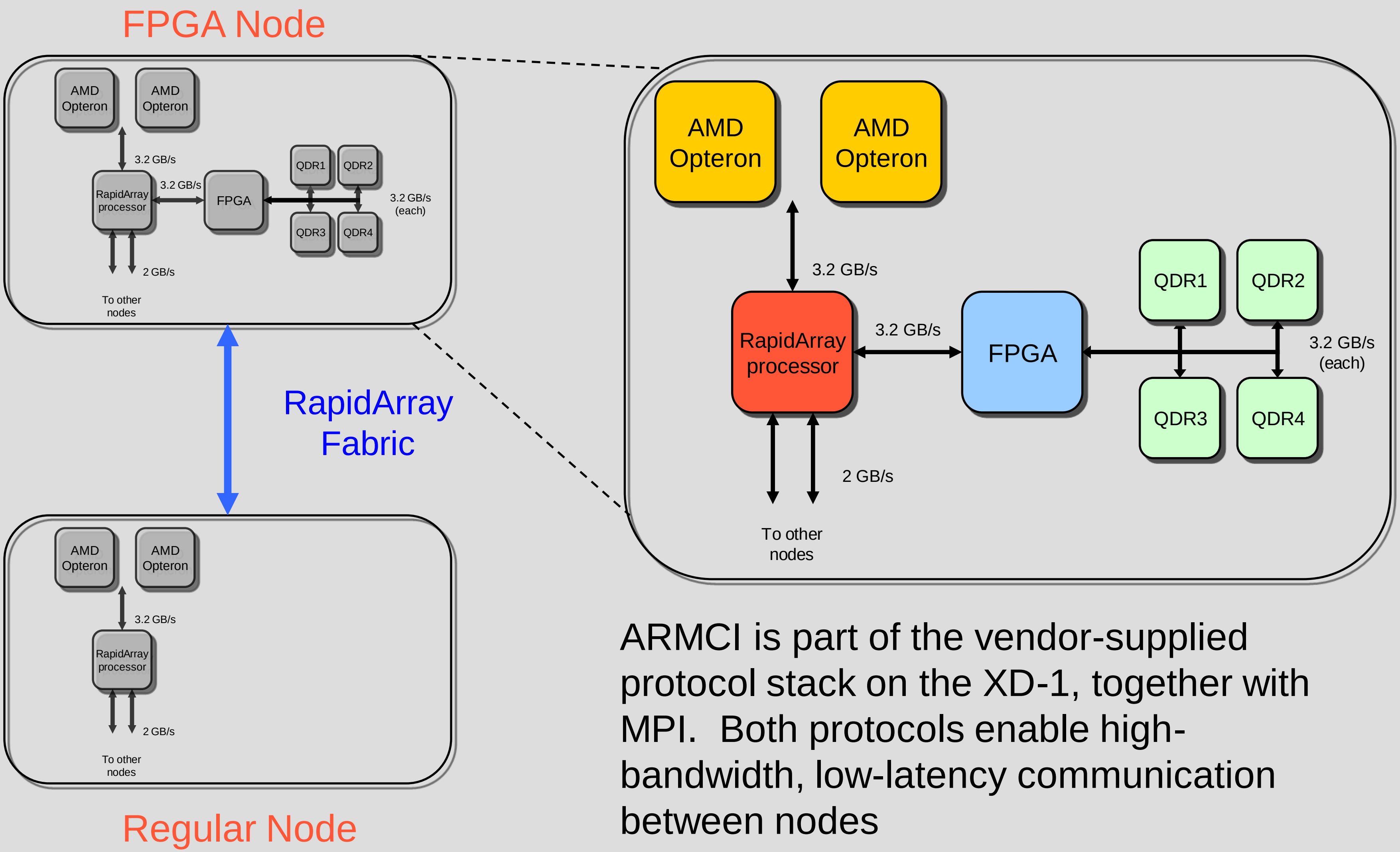
Nothing shared …



What we've been working on

- Started with Utah CCA/SciRun event service implementation
  - As of August 2006
- Created two standalone prototypes (no SIDL, no framework):
  - Reliable: events transferred via files
  - Fast: events transferred over ARMCI on Cray XD1
    - Single-sided memory transfers

Cray XD-1

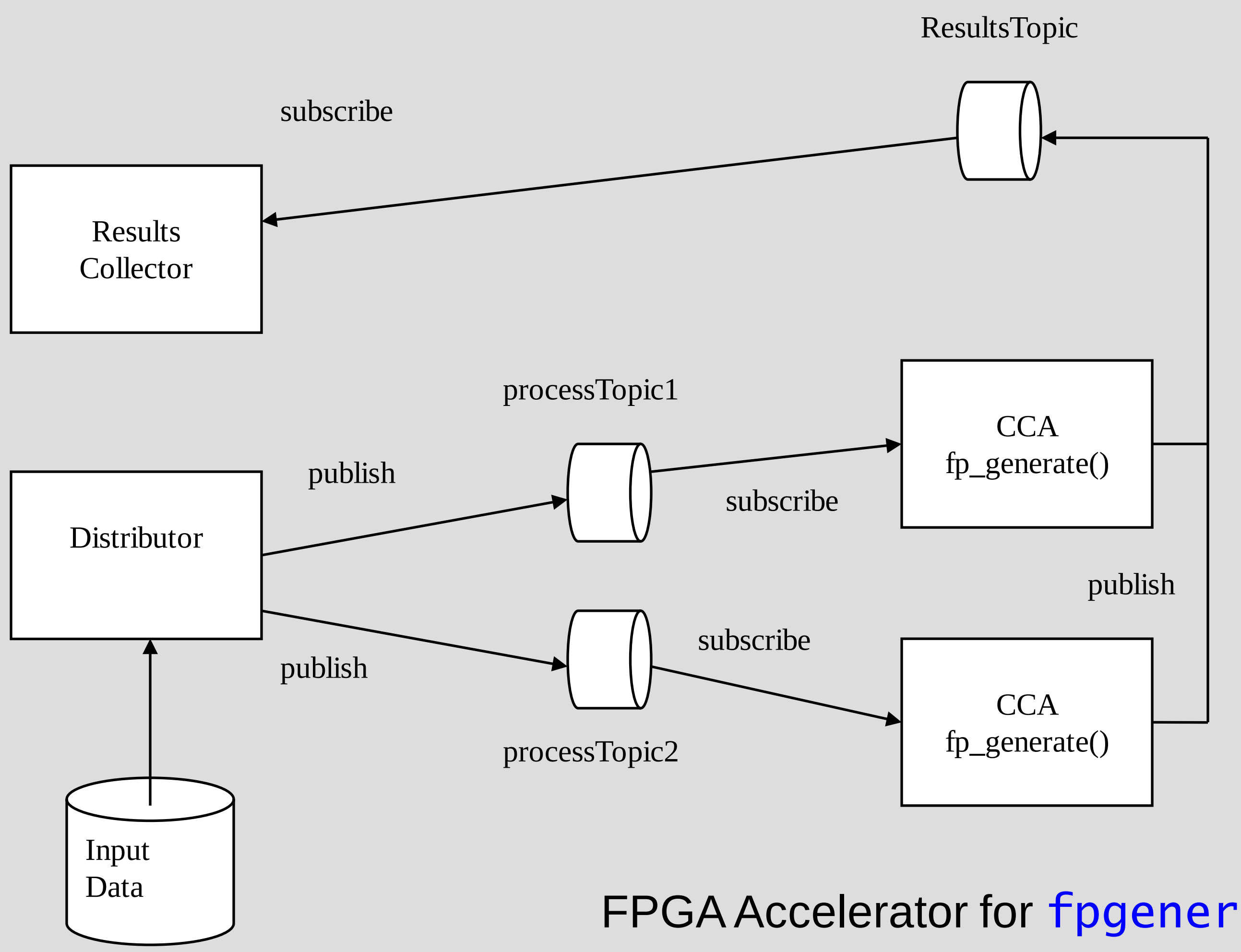


ARMCI is part of the vendor-supplied protocol stack on the XD-1, together with MPI. Both protocols enable high-bandwidth, low-latency communication between nodes

Polygraph

- *Polygraph* is a proteomics application developed at PNNL
  - Analyzes protein spectra obtained from mass spectrometry experiments
  - Each spectrum consists of position and intensity arrays (100 - 400 entries)
- For each input, *Polygraph* scans a reference database of several million proteins (FASTA, multi-GB size)
  - Generates a list of matching peptides based on weight (thousands to millions of candidates)
  - Match list is refined further by computing a projected spectrum for the reference data point and assigns it a score based on statistically generated datasets & matching "peaks"
  - Top matches are identified for each spectrum
- Profile of the application indicates that 3 routines take 51% of the exec. Time
  - `fpgenerate(), fp_set_hypoth(), fpextract()`

Our Target – PolyGraph/FPGAs



FPGA Accelerator for `fpgenerate()`

ARMCI Prototype

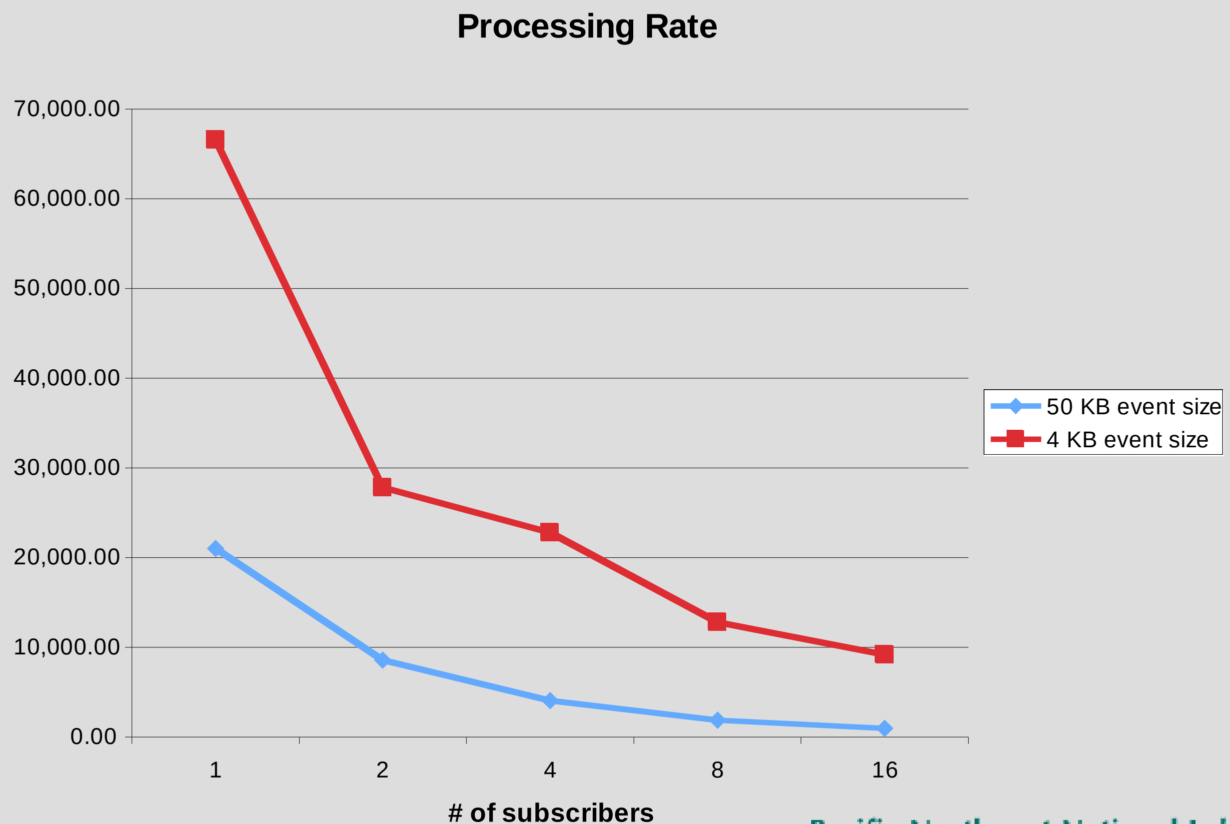
- Goals:
  - maintain interface/semantics of the event service model
  - achieve high performance in a distributed memory HPC system
- Used combination of MPI & ARMCI
- MPI - Process 0 operates as a *Topic Directory* process
  - Maintains a Topic List with the locations of the publishers
  - Uses an MPI messaging protocol to serve topic creation requests and queries
- ARMCI - Publishers create events locally in their own address space
  - Subscribers read remote events from the publishers using one-sided ARMCI_Get() operations
    - no need for coordination with the publisher

ARMCI Prototype (cont.)

- Used a combination of MPI & ARMCI to create the event service
  - Transfer C++ class instances directly over ARMCI without the need for type serialization
  - Events comprise two TypeMaps: header and body
- Created a special heap manager for the ARMCI address space
  - objects can be allocated directly through standard new() and delete() operators
  - synchronous garbage collection by the publisher
- For high performance, all objects in the ARMCI heap are flattened
  - no pointers or references to external objects
  - member variables embedded
  - fixed size

Initial Performance Results

- We measured event processing rates:
  - 66K events/second with one publisher/one subscriber (small event 4KB)
  - 950 events/second with one publisher/16 subscribers (large event 50KB)
  - Minimal overhead to reconstruct the object on the subscriber after the transfer

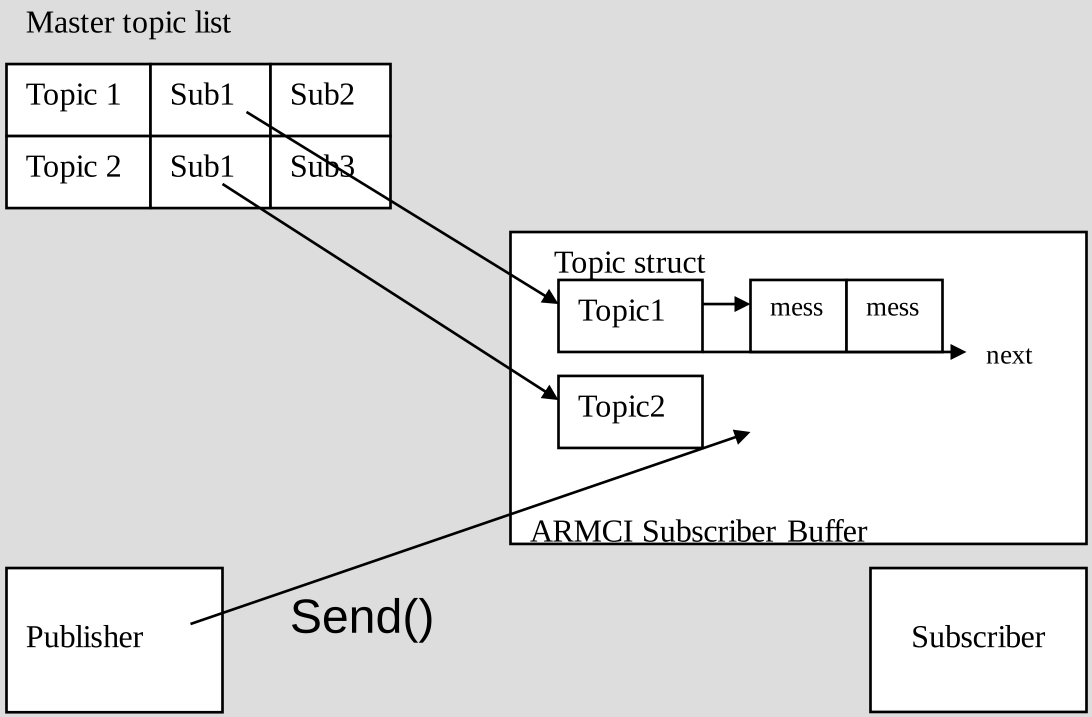


Analysis

- Performance drops as number of subscribers increases
  - Not unsurprisingly :-}
  - Contention for events at publisher ARMCI memory
- Alternatives implementations are possible:
  - Maintain topics for subscribers only in local ARMCI memory
  - Publishers write to subscriber memory directly for each event published

Alternative Design

Maintain topic list in process 0 (using MPI) or ARMCI shared memory?



Strengths?

Likely reduced contention
Simplifies 'publish semantics' and event retention issues

Weaknesses?

Publish can fail if subscriber memory full
Some subscribers slower than others - events delivered unpredictably depending on consumption rate

Polygraph Issues: Delivery Semantics

- Basic pub-sub good for N-to-N event distribution
  - Need to keep events until all subscribers consume them
  - Optional ‘time-to-live’ in header can help
- Workload distribution use cases require ‘load-balancing’ topics
  - Same programmatic interface
  - Each event consumed by only one subscriber
  - No complex event retention issues
  - Could define load-balancing policies for publishers
    - Declaratively?
  - A ‘one-to-one’ queue-like mechanism may also be useful?

Issues: Topic Memory Management

- Managing memory for a topic is tricky:
  - Need to know how many subscribers for each specific event
  - Events are variable size, hence allocating/reclaiming memory for events is complex
- One possibility: typed topics
  - Associate an event type with a topic
  - Specify maximum size for any event
  - Simplifies memory management for each topic

Issues - Miscellaneous

- What are semantics when a new subscriber subscribes to a topic?
  - What exactly do they see?
  - All messages in topic queue at subscription time?
  - Only new ones?
- In ARMCI implementation, memory for topic queues is finite
  - Should it be user-configurable?
  - What happens when topic memory full?
  - Standard publish error defined by Event Service?

Other Implementation Issues

- Should events have a ‘standard’ header
  - Used by all event service implementations
  - Not settable programmatically
  - E.g. Time-to-live, timestamp, correlation-id, likely others …
- Push versus pull implementation model
- Threading
- Topic wildcarding
- Message priorities

Since we wrote the paper …

```
//
// Event Service Specification (Draft as of February 6th 2007)
//
interface EventServiceException extends CCAException {
}
interface PublisherEventService extends cca.Port {
cca.Topic getTopic(in string topicName)
  throws EventServiceException;
bool existsTopic(in string topicName);
}
interface SubscriberEventService extends cca.Port {
cca.Subscription getSubscription(in string subscriptionName)
  throws EventServiceException;
void processEvents() throws EventServiceException;
}
interface Event extends sidl.io.Serializable {
cca.TypeMap getHeader();
cca.TypeMap getBody();
}
interface EventListener {
void processEvent(in string topicName, copy in Event theEvent);
}
interface Topic {
string getTopicName();
void sendEvent(in string eventName, in cca.TypeMap eventBody)
  throws EventServiceException;
void release();
}
interface Subscription {
void registerEventListener(in string listenerKey,
  in EventListener theListener) throws EventServiceException;
void unregisterEventListener(in string listenerKey);
string getSubscriptionName();
void release();
}
```

Next steps …

- Implemented alternative ‘subscriber side’ ARMCI implementation
- Detailed performance analysis
  - As we speak …
- Use Event Service to implement several use cases
  - Polygraph
  - Asynchronous IO
  - Proteomics processing pipeline
  - Hiding complexity of hybrid architectures
- Would like to discuss others …
  - Potential for collaboration?