OpenCPI - Open Component Portability Infrastructure

An Open Source Infrastructure for Heterogeneous Component-Based Systems and Applications

Community Briefing
January 31, 2011

- **Jim Kulp** jek@parera.com
- **Michael Pepe** mpepe@mercfed.com

Agenda

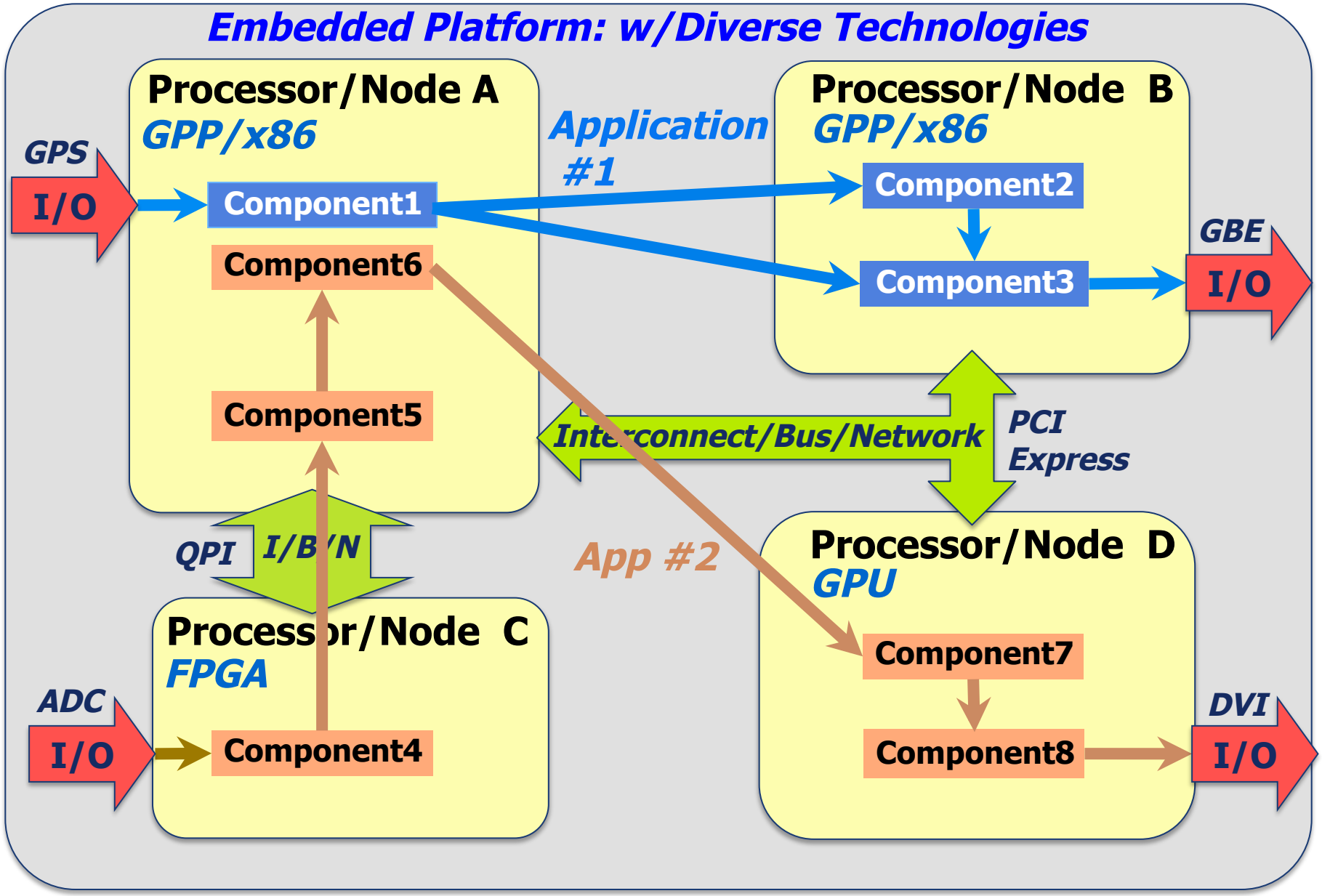
- **OpenCPI Introduction, Goals, History** **9:00**
- OpenCPI Briefing 9:15
- Recent Developments 10:00
- Break 10:30
- Roadmap Ideas, Discussion 10:45
- Workflow, code examples 11:15
- Q&A, Wrap-up 11:45

Open-Source Component Portability Infrastructure

Is the:

1. **Open Source Community Solution for:**
2. **Component-Based Development (CBD) in**
3. **Heterogeneous Embedded Systems**

Component-Based Embedded Systems & Applications



What is OpenCPI *really*?

- In the box/kit (actually in a public open source repository):
  - For runtime: executing components and moving data
    - Runtime libraries (software and FPGA IP)
    - Drivers
  - For development time: developing components and applications
    - Tools and Scripts (command-line)
    - Specifications and Documentation
    - Examples and Tests
- A framework and methodology for:
  - Component-based Development
  - Mixing processor types (GPP, GPU, FPGA, DSP, etc.)
  - Diverse interconnect technologies
  - *Targeting embedded defense applications*
- OpenCPI sits below modeling tools and IDEs
- Cooperating/layered with a variety of other middlewares

Why bother? Why is OpenCPI needed?

- Especially in embedded systems, *diversity is reality*.
  - Meeting SWAP constraints *demands* diversity.
  - Some technology and vendors *advocate* diversity.
  - Changes in technology or mission *require diversity over time.*

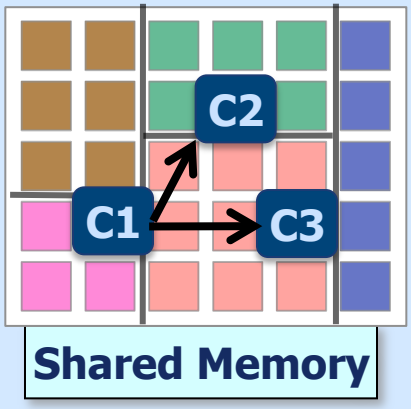
- Diversity/heterogeneity of:
  - Processing: GPP/Multicore, GPU, FPGA, DSP, Tilera/many-core
    - Separate chips or SoCs (OMAP, Zync, etc.)
  - Interconnects: 1/10GE, PCIe, SRIO, point-to-point, Infiniband,…
    - And on-chip between collocated components
    - And external connections to required middleware: DDS, CORBA
  - Scale: Multicore SoC to distributed Large Data clusters.
    - Allow embedded/tactical configurations that also work scaled up.

- Diversity brings unnecessary baggage:
  - *Things are different that don't have to be*

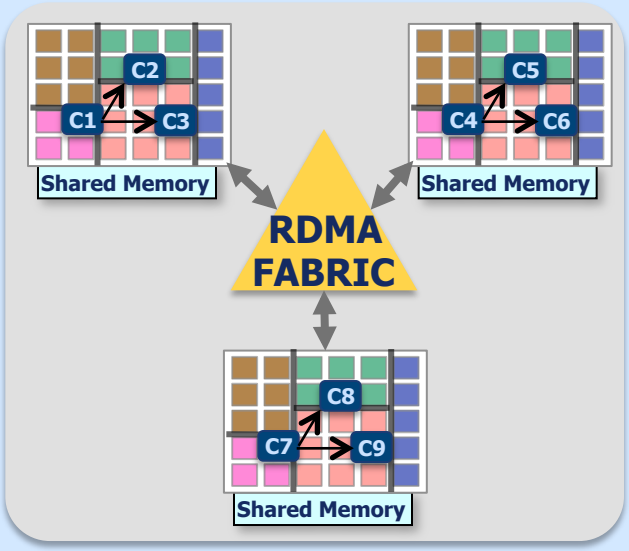
Interconnect Diversity

- **Many options available**
  - ***with different programming abstractions and APIs***

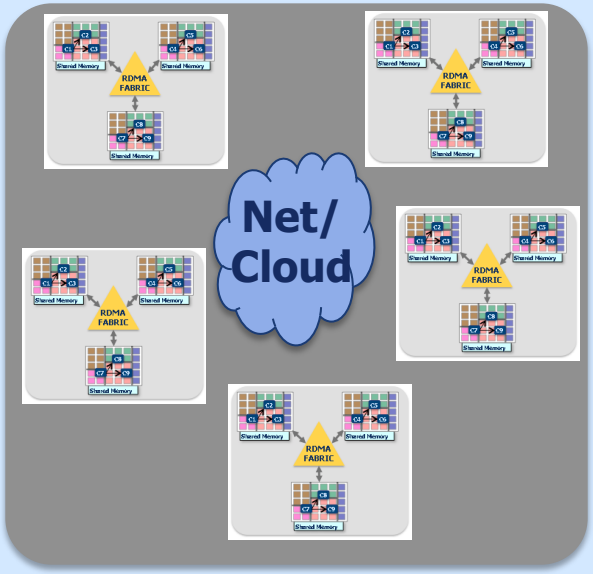
Intra-Chip/On-Chip
Multi-core/FPGA/GPU/DSP



Inter-Chip/Board/In-Box
Fabric/Bus/Backplane



Inter-Box/Platform
Network (IP/TCP/UDP)



Multicore Processing

Multi-node Processing

Grid/Cluster/Distributed Processing

OpenCPI alleviates and attacks diversity

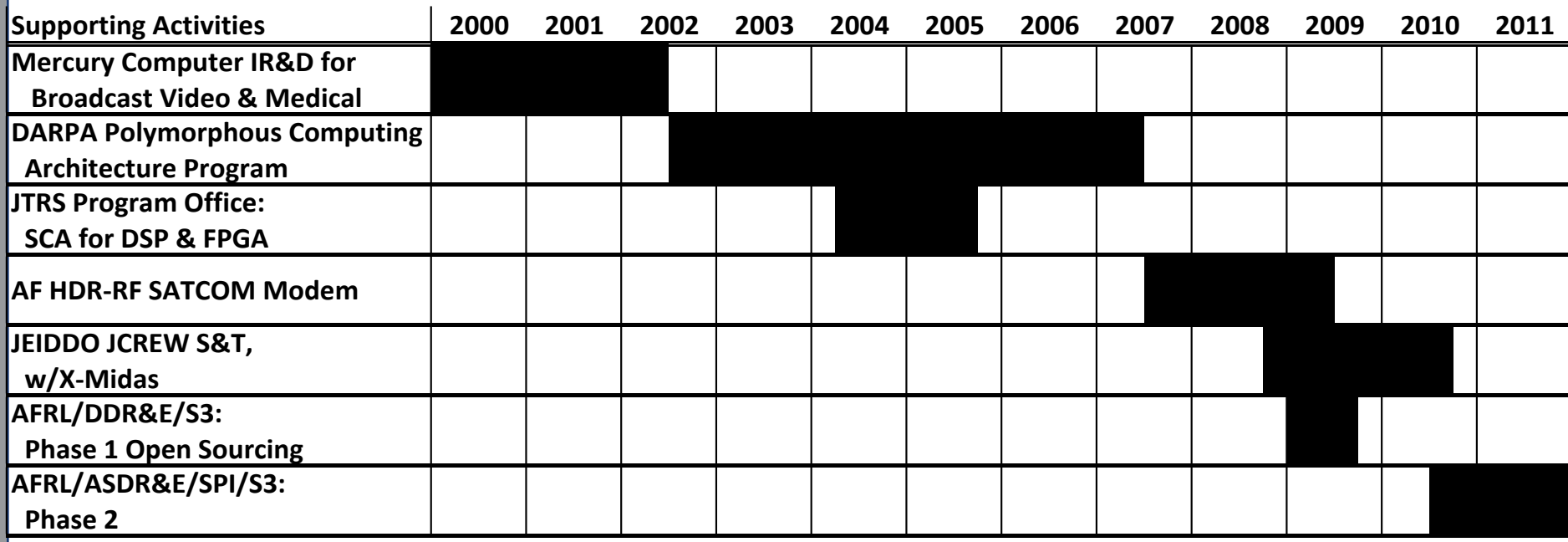
- Commonality at the right price
  - Well-informed, vendor and technology-neutral choices
  - Don't reinvent how algorithmic code is written
  - Suppress diversity without performance tax

- Open source is key:
  - Industry, products, tools are in silos
  - Community and openness is necessary to cross all the boundaries.

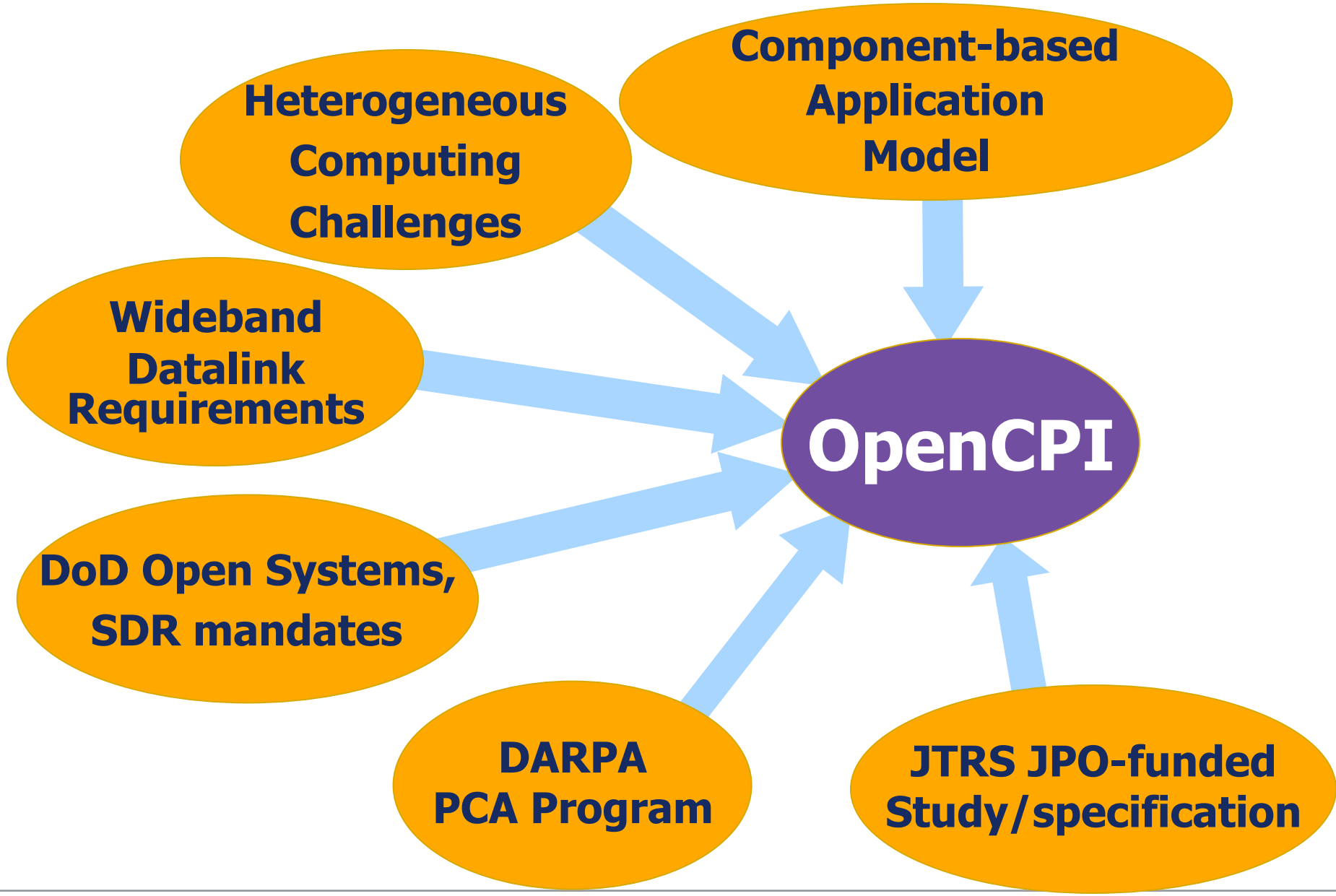
- OpenCPI *attacks* this diversity as its primary focus:
  - Creating more consistency during development and at runtime
  - Paying attention to embedded efficiency requirements
  - Crossing boundaries not addressed by other approaches
  - Common component-based design & runtime across the boundaries

Where did OpenCPI come from?

- It evolved via a succession of IR&D and DoD funded projects
- Core motivation was: *apply CBD to real embedded applications.*
- Various application areas were assessed and needs addressed
- Many lessons learned

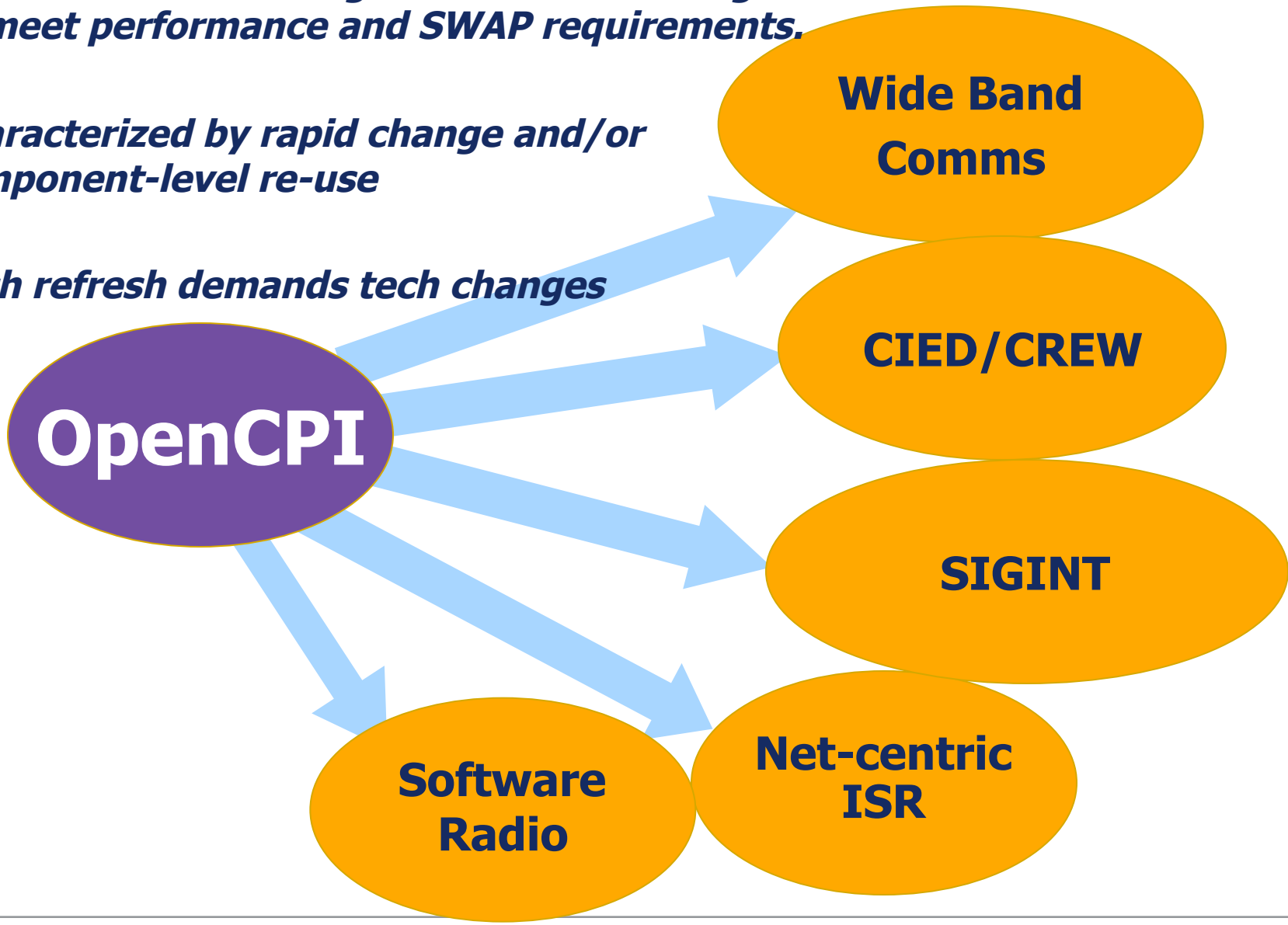


Where OpenCPI comes from



Where OpenCPI can be applied:

- ***Application areas using advanced technologies to meet performance and SWAP requirements.***
- ***Characterized by rapid change and/or component-level re-use***
- ***Tech refresh demands tech changes***



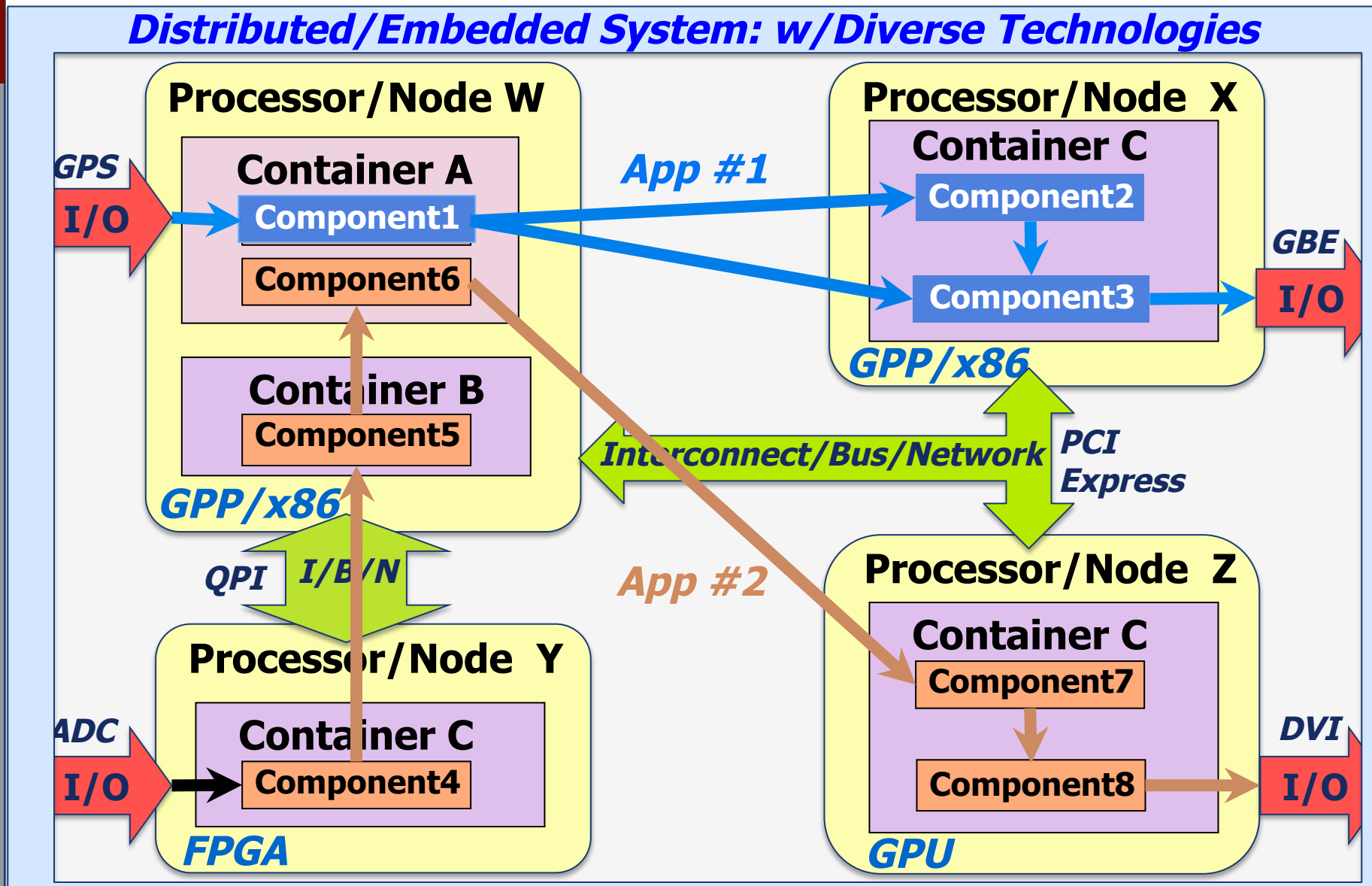
Agenda

- OpenCPI Introduction, Goals, History 9:00
- **OpenCPI Briefing** **9:15**
- Recent Developments 10:00
- Break 10:30
- Roadmap Ideas, Discussion 10:45
- Workflow, code examples 11:15
- Q&A, Wrap-up 11:45

OpenCPI Briefing

- Model/terminology of component-based systems
  - Consistent with other component-based systems
- Components
  - Describing, writing, building
- Applications
  - Specifying, controlling, executing
- Authoring Models for different technologies
  - GPP, DSP, GPU, FPGA…
- Platform support for component-based applications
  - Infrastructure for running component-based applications

Component-Based Embedded Systems & Applications



Component Model

Name:

- The functionality

Properties:

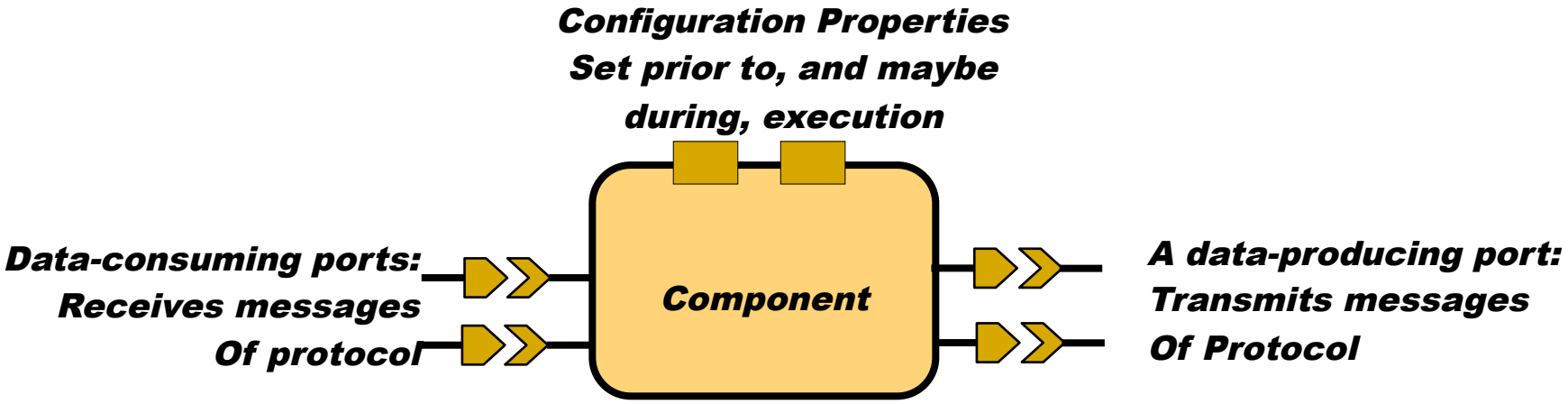
- With data type, including arrays etc.
- Initially configured or available to read/write at runtime

Ports:

- With *protocols* that say what messages come in or go out

Execution:

- Autonomous, simplified "actor" model, inherently concurrent, interacting via messages at connected ports



OpenCPI Terms: for components and applications

How we define components

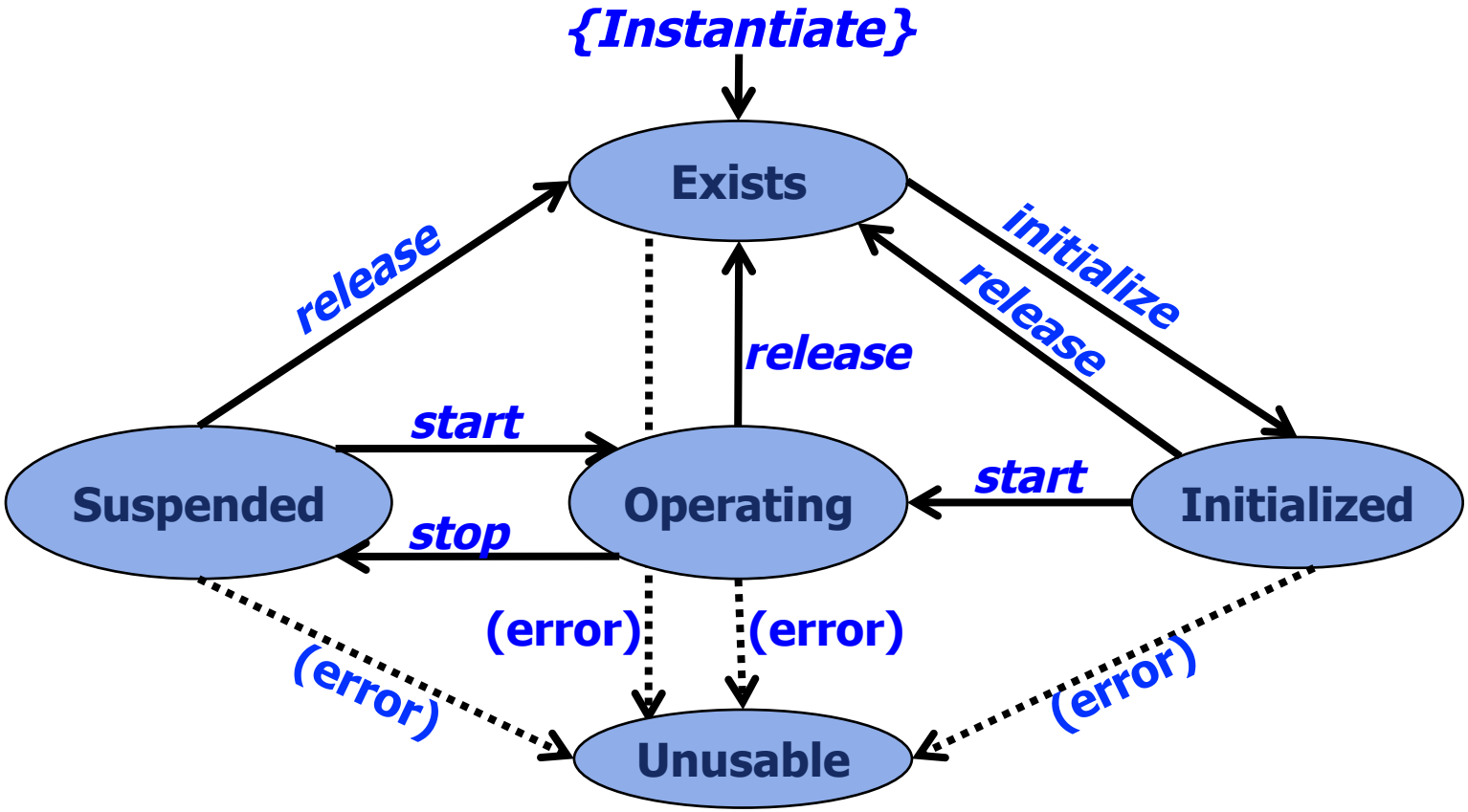
- Component: building blocks for applications
  - Defined in XML as a "spec" for alternative implementations
- Property: component configuration and control parameters
  - Defined as part of component "spec", usable at runtime
- Port: component interfaces to talk to others
  - Defined as part of component "spec" to send/receive messages
- Protocol: the set of supported messages at a port
  - Defined in XML and associated with component ports
- Worker: one of several implementations of a component
  - Defined in XML and associated with a component

How we define applications

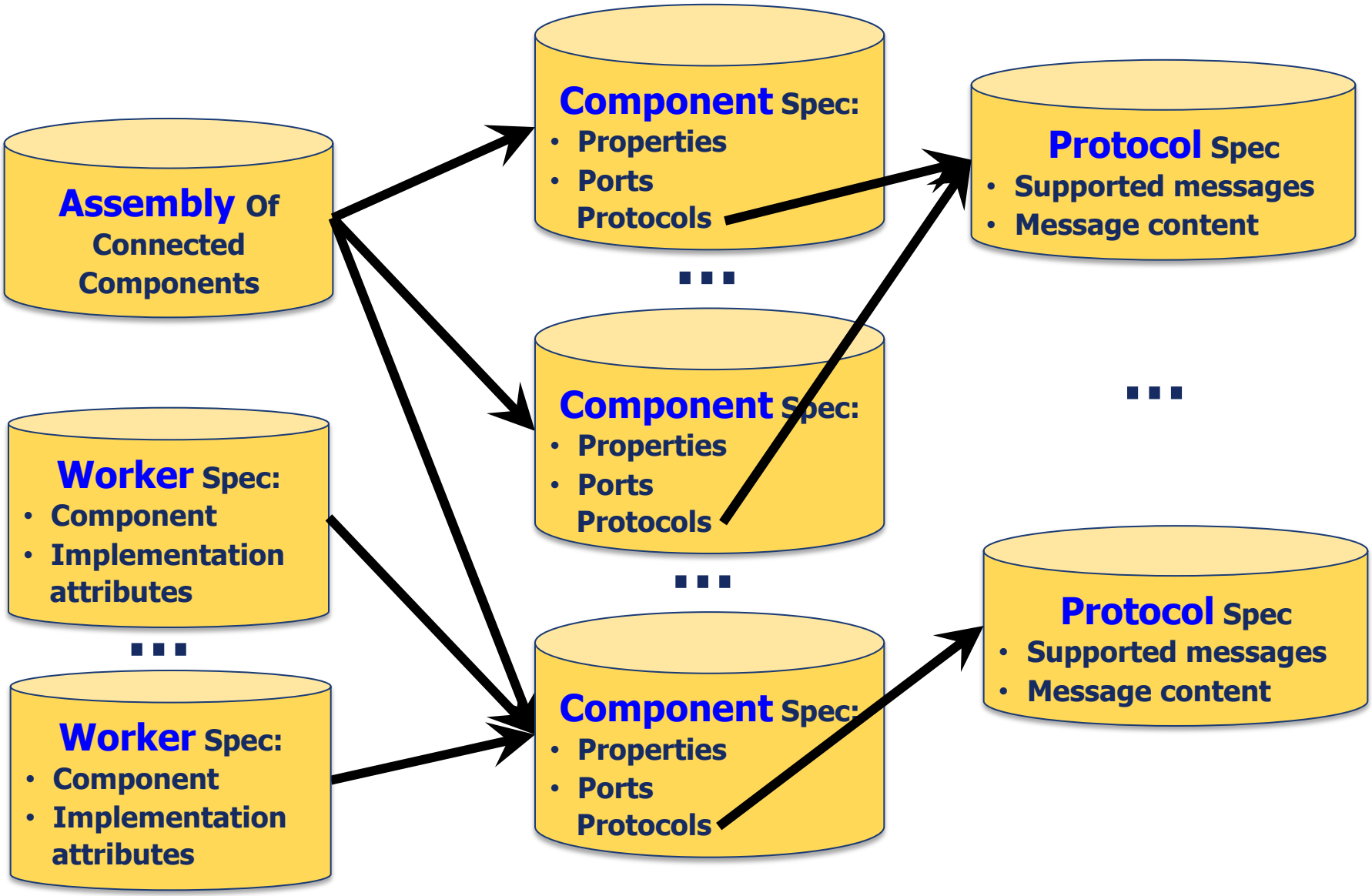
- Assembly: a set of components with interconnected ports
  - Defined in XML as used to launch applications

Component Lifecycle (Control State) Model

- Simplified/compatible control model (vs. CCM or SCA)
  - initialize – called by container upon instantiation
  - start – called by control application to become operational
  - stop – when control application suspends the application or component
  - release – before a worker is destroyed, or reused



OpenCPI Metadata XML Files



OpenCPI Terms: Actual component *implementations*

- Worker: implementation of a component
  - Declared in XML (OWD) and based on compilable source code
  - Refers to the component spec (OCS) it is implementing
- Artifact: compiled binary package of workers
  - Result of building/compiling some workers
  - Includes embedded metadata needed for runtime
  - *Physical unit that can be patched, updated, emailed, like a DLL/so.*
- Library: collections of artifacts for runtime
  - Typical "library" concept
  - Runtime has a typical "library path" to find and use workers
- Authoring Model: one of several ways to write worker code
  - For different technologies, in different languages
  - APIs for worker authors are similar, but specific, for each model

OpenCPI Terms: Infrastructure Software

The runtime software that runs component-based apps

- Container: the execution environment for workers
  - Managing the execution of workers on a processor
  - Arranging data plane communications with other containers
- Transport: a driver for data path capabilities
  - Convey messages between containers, for workers
  - Supports some particular underlying mechanism/interconnect
- Control API: support launch and control of applications
  - Allow "control applications" to run component-based apps
  - Two styles: dynamic, fine grained control, or XML assemblies
- Application: a software object at runtime
  - The single point of control for the component-based application

OpenCPI Terms: The system platform

A system running component-based applications

- Processor: the execution engines in the system
  - All logically acting as peers, places to execute workers
- Interconnect: data pathways between processors
  - How workers in different containers talk to each other
  - PCIe, SRIO, GBE etc.
- I/O: hardware points of entry and exit for data
  - Usually associated with or managed by specific processors
- External Services: soft sources and destinations of data
  - Endpoints reachable via other middleware

OpenCPI Briefing

- Model/terminology of component-based systems
  - Consistent with other component-based systems
- **Components**
  - Describing, writing, building
- Applications
  - Specifying, controlling, executing
- Authoring Models for different technologies
  - GPP, DSP, GPU, FPGA…
- Platform support for component-based applications
  - Infrastructure for running component-based applications

Creating components and workers: Step 0/5

- Each port of a component (input or output) port has a protocol
- A protocol is a set of allowable messages
- Each message has a defined payload
- A compatible simplification of "IDL" from DDS/CORBA/SCA
- Define (or refer to existing) message protocols for ports:

```
<Protocol>
  <Operation Name="Samples">
    <!-- fixed size frames with 16 bit values -->
    <Argument Name="samples" Type="short" ArrayLength="4k"/>
  </Operation>
  <Operation Name="Sync"/>
    <!- phase discontinuity - no arguments -->
  <Operation Name="Time">
    <!-- timestamp for next sample - unsigned 64 bits-->
    <Argument Name="time" Type="ULongLong"/>
  </Operation>
</Protocol>
```

Creating components and workers: Step 1/5

- Describe the component in XML: the "spec file"
  - The OCS: OpenCPI Component Specification
  - Name, properties, and ports (and message protocol per port)
  - The basis for all implementations (workers)

```
<ComponentSpec Name="FFT1D">
  <Property Name="direction"/>
  <Property Name="scale" type="float"/>
  <DataInterfaceSpec Name="in">
    <xi:include href="stream_protocol.xml"/>
  </DataInterfaceSpec>
  <DataInterfaceSpec Name="out" Producer="true">
    <xi:include href="stream_protocol.xml"/>
  </DataInterfaceSpec>
</ComponentSpec>
```

Creating components and workers: Step 2/5

For each implementation (worker):

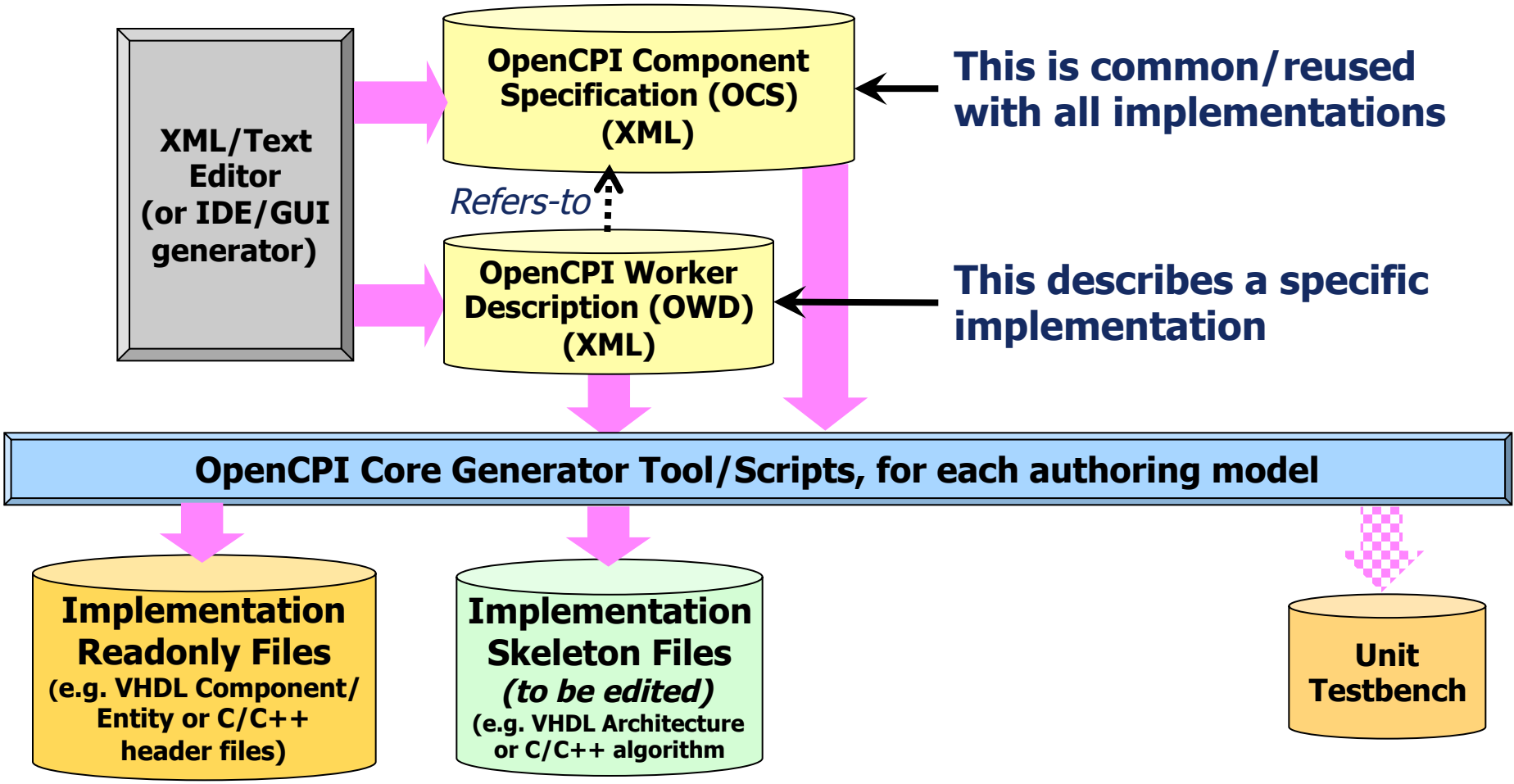
- Describe the worker's *implementation* attributes in XML
  - The OWD: OpenCPI Worker Description
  - Extra implementation-specific attributes of ports
  - Extra implementation-specific properties
  - Typically not much to say

```
<OclImplementation Name="FFT"">
  <xi:include href="fft_spec.xml"/>
</OclImplementation>
```

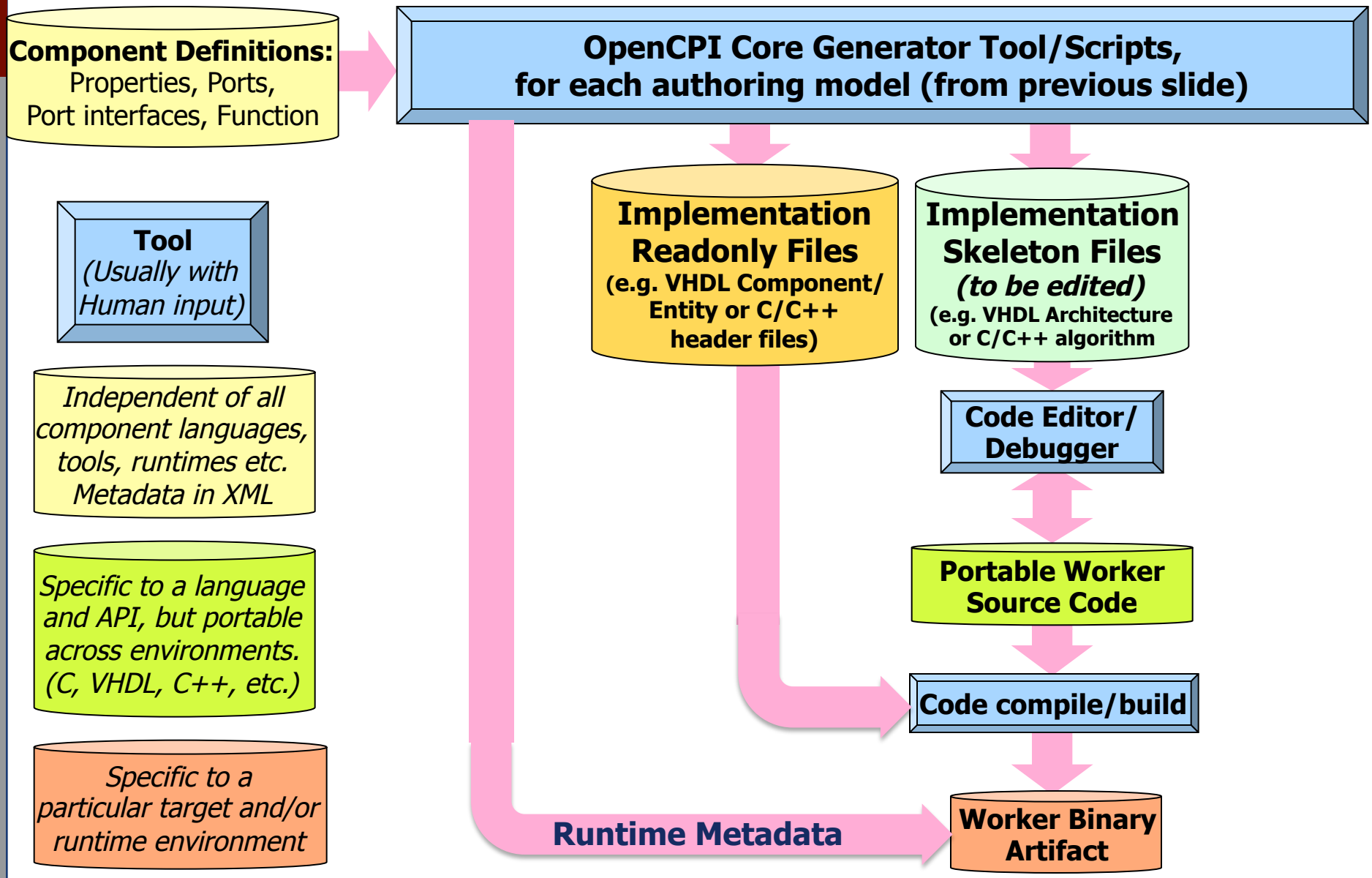
- This example is for GPUs, and describes an OCL (GPU/ OpenCL-based) implementation.

Creating components and workers: Step 3/5

- Component and worker XML descriptions are done
- A simple Makefile/script runs a tool to create the source code starting point ("skeleton").



Creating components and workers: Step 4/5



Component Binary Libraries: Step 5/5

- A component binary library is a directory tree filled with worker binary "artifact" files which:
  - Are individually loadable/executable
  - Have embedded descriptive metadata
  - Contain one or more workers in a given binary file.
- Applications are executed against a path of libraries
  - The same "library path" concept on all Windows and UNIX, e.g.:
    - export OCPI_LIBRARY_PATH=/home/fred/myocpilib:/home/proj/projlib
  - Except they contain self-describing ***heterogeneous*** binaries.
  - Each component mentioned in the assembly is found in a library

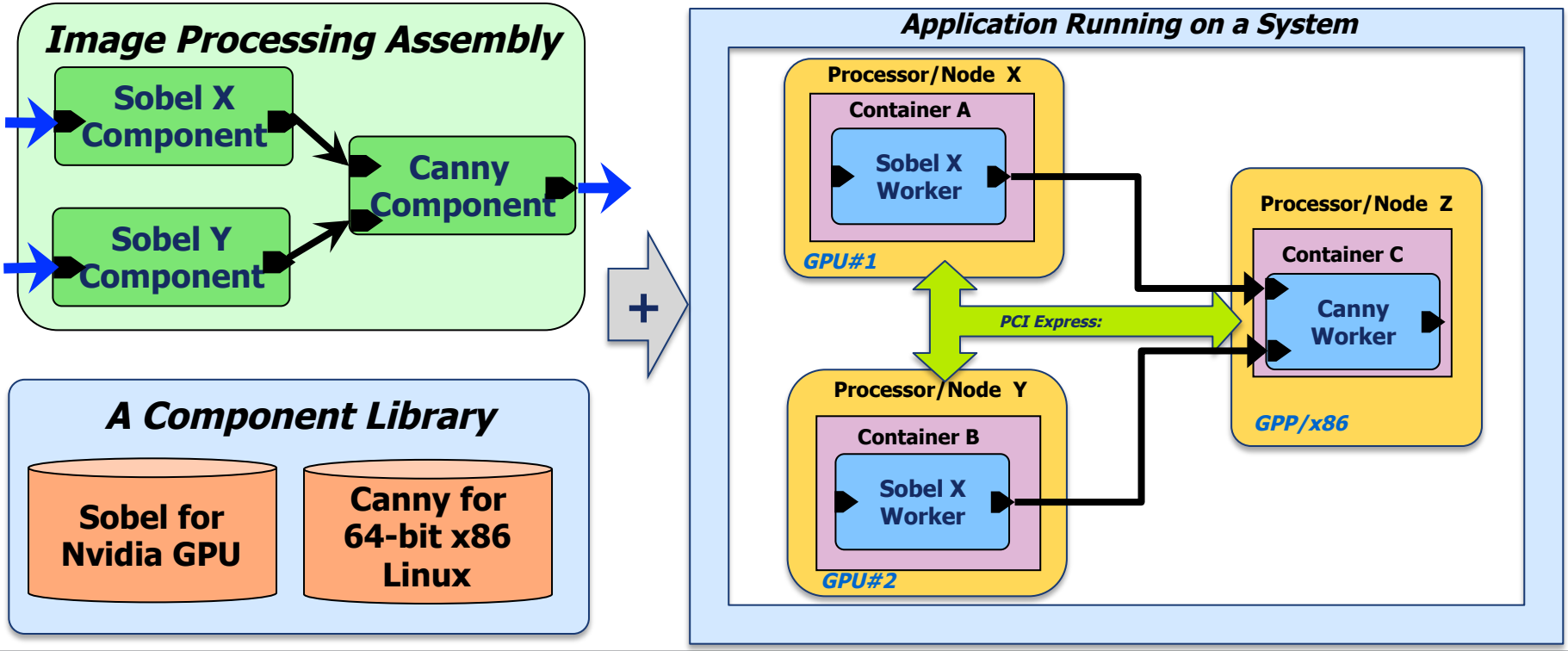
A Component Binary Library (directory tree), full of Artifact files

OpenCPI Briefing

- Model/terminology of component-based systems
  - Consistent with other component-based systems
- Components
  - Describing, writing, building
- **Applications**
  - Specifying, controlling, executing
- Authoring Models for different technologies
  - GPP, DSP, GPU, FPGA…
- Platform support for component-based applications
  - Infrastructure for running component-based applications

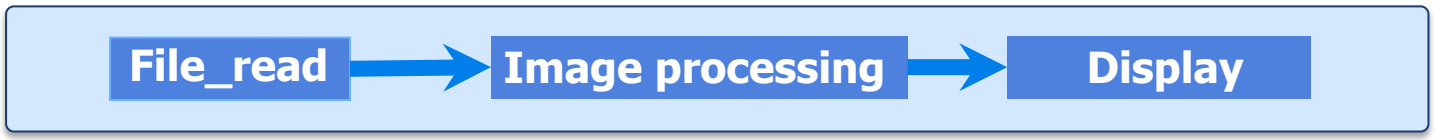
OpenCPI Applications (using components!)

- An interconnected set of components (an assembly)
- Executed in containers (runtime environments)
- On a collection of processing nodes
- Using component implementations (workers)
- Found in heterogeneous component binary libraries
- Resolving external ports of the whole assembly (later)



Resolving External Connections: outside the assembly

- I/O (device/sensor/file) is usually handled by special I/O components
  - That read/write files
  - That put/get data from I/O ports (sensors, DAC/DAC)
  - That may well be heterogeneous (e.g. direct I/O in FPGA, video in GPU)
  - Thus this is really inside the assembly, done by components



- External ports of the assembly have (currently) 3 choices:
  - Messaged to/from the controlling/launching application via an API
  - Connecting to CORBA running elsewhere (e.g. SCA software radio)
  - Connection to DDS running elsewhere (pub/sub, wide area dissemination, sensor alerts etc.).

An OpenCPI application runs one of two ways

1. Using an Assembly Description in XML
   - Most applications are easier/simpler to express this way
   - Connections, allocation policy, external connections all possible
   - XML schema is designed for humans: simple and readable

2. Using the OpenCPI Application Control API.
   - For complex, dynamically or parametrically structured applications
   - For use by other "application management" layers such as:
     - SCA (Application Factory)
     - CCM (Deployment and Configuration Node Manager)
   - "main" creates and connects workers via API calls

OpenCPI XML-based applications

- An assembly of connected components
  - With optional external connections
- A file copy application with two components:


```
<application>
  <instance worker="file_read">
    <property name="infile"       value="hello.file"/>
    <property name="messageSize" value="4"/>
  </instance>
  <instance worker="file_write">
    <property name="outfile"      value="out.file"/>
  </instance>
  <connection transport="ofed">
    <port instance="file_read"    name="out"/>
    <port instance="file_write"   name='in'/>
  </connection>
</application>
```

OpenCPI Control Application for vector add (*generic*)

```
Container* ocl = OA::ContainerManager::find ("ocl”);
ContainerApplication *app = ocl->createApplicatio(”myapp”);
Worker& w = app.createWorker("vector_add", "vector_add");
ExternalPort
  &port_a = w.getPort ( "a" ).connectExternal ( ),
  &port_b = w.getPort ( "b" ).connectExternal ( ),
  &port_c = w.getPort ( "c" ).connectExternal ( );
ExternalBuffer
  *a = port_a.getBuffer ( data_a, length_a),
  *b = port_b.getBuffer ( data_b, length_b );
// Fill buffers A & B w/ inputs to the vector ADD component
….
// Now send the A and B inputs on their way
a->put(length_a);
b->put(length_b);
// Now get the answer in buffer C
port_c.getBuffer(data_c, length_c, opcode_c, end);
// Take data from “data_c”.
```

OpenCPI Briefing

- Model/terminology of component-based systems
  - Consistent with other component-based systems
- Components
  - Describing, writing, building
- Applications
  - Specifying, controlling, executing
- **Authoring Models for different technologies**
  - GPP, DSP, GPU, FPGA…
- Platform support for component-based applications
  - Infrastructure for running component-based applications

Authoring Models: How to write worker code

- An Authoring Model:
  - Defines one of several alternative ways to write a worker
  - To fit well with:
    - ***How practitioners actually write code for some technology***
    - ***Tools/compilers that enable effective use of the technology***
  - To coexist and interoperate with workers from other models
  - To efficiently interact with collocated similar workers

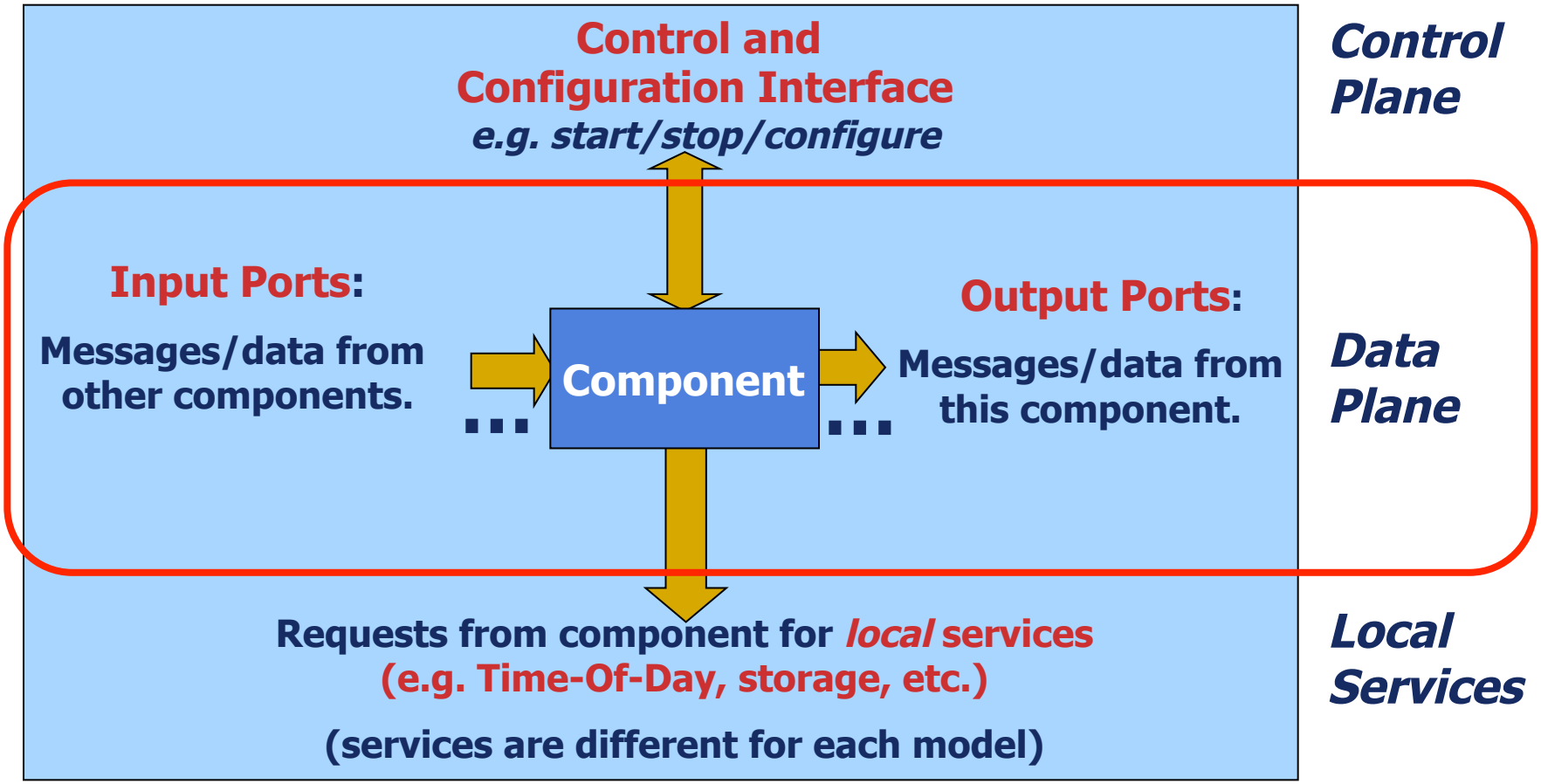
- For each authoring model, we define:
  1. The XML for implementation-specific details, if any
  2. The control interface/API for properties and lifecycle management.
  3. The data interface/API for talking to other workers (data plane)
  4. Interfaces to "local services" like local memory allocation etc.
  5. The execution model: how the container executes the workers

- OpenCPI has authoring models for GPP, DSP, FPGA, GPU

OpenCPI Authoring Models

- RCC (Resource-constrained C-language)
  - For GPPs and DSPs, including very limited single-threaded systems
- HDL (Hardware Description Language)
  - For FPGAs/ASICS, Verilog (or VHDL* or Bluespec*)
- OCL (OpenCL worker model)
  - For GPUs and some multicore (even Altera FPGA announced by them)
- XM (X-Midas*)
  - To use algorithms/primitives from X-Midas libraries
- SCA (Software Radio)
  - When combined with an SCA Core Framework (more later)

SW/Gateware Authoring Model *Pattern*



- Models for all technologies must *interoperate* in a system
- *Simple enough for low-level implementations*

"RCC" Authoring Model for Resource-Constrained C Language environments

- All interfaces are C, can run in all C environments
  - Could easily have a simple C++ version too
  - Not all DSPs/Microcontrollers have good C++ support
- Lifecycle control and property interface entry points
  - Shared memory for very efficient runtime configuration properties
- Inter-component data-plane interfaces
  - Get/put buffers formatted as message structures derived from protocol
  - *Zero copy everywhere.*
- Local service interfaces
  - Roughly ANSI C runtime library without I/O
- *Data-driven execution model does not require an OS, or threads.*
  - *Thus usable for DSPs, Microcontrollers etc.*

Simple RCC worker code example: add a bias value

```
static RCCResult
run(RCCWorker *self, RCCBoolean timedOut, RCCBoolean *nrc) {
    RCCPort
        *in = &self->ports[BIAS_IN],
        *out = &self->ports[BIAS_OUT];
    BiasProperties *props = self->properties;
    uint32_t
        *inData = in->current.data,
        *outData = out->current.data;

    unsigned n;
    for (n = in->input.length / sizeof(uint32_t); n; n--)
        *outData++ = *inData++ + props->biasValue;
    out->output.length = in->input.length;
    return RCC_ADVANCE;
}
```

"OCL": authoring model for GPUs based on OpenCL

- OpenCL is an open standard for writing task- and data-parallel programs that execute on GPUs/Multicores.

- OpenCL is actually two distinct things:
  1. A C99 dialect (subset and superset) and APIs for writing data/task parallel compute "kernels" that execute on GPUs
  2. A C API for compiling, controlling and exchanging data with "kernels" running on GPUs (accelerators), from the "host" CPU.

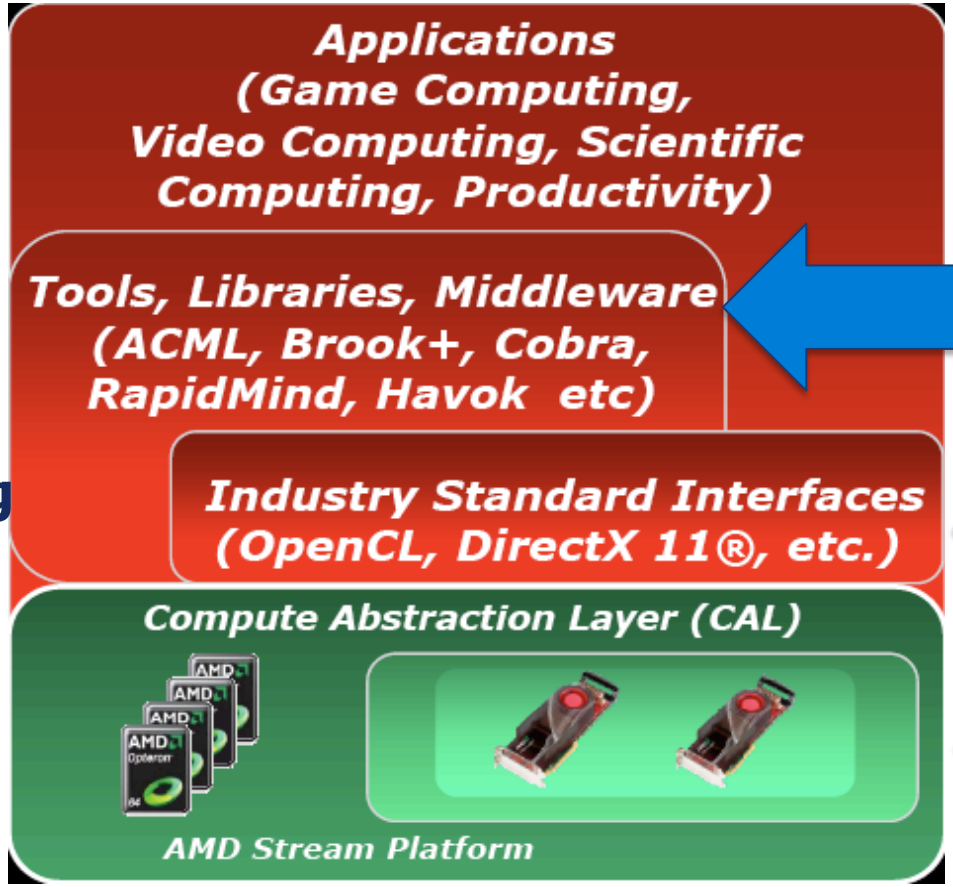
- Broad Vendor support for OpenCL
  - AMD/ATI offers both OpenCL for their CPUs and GPUs
  - IBM offers OpenCL for the Cell Broadband Engine
  - Intel offers OpenCL for their multicore processors.
  - NVIDIA offers OpenCL and their proprietary "CUDA" on their GPUs

- Initial OpenCL offerings were weak (2009), now are better

OpenCL for GPUs

- Abstracts vendor-specific details but no more
- Usable for various multicore architectures
- Required to fully exploit GPU power with portable APIs

AMD's Graphic: Positioning OpenCL



OpenCPI is in this category

Appropriate Interface Underneath OpenCPI

Vendor-Specific

The OpenCPI "OCL" Authoring Model for GPUs

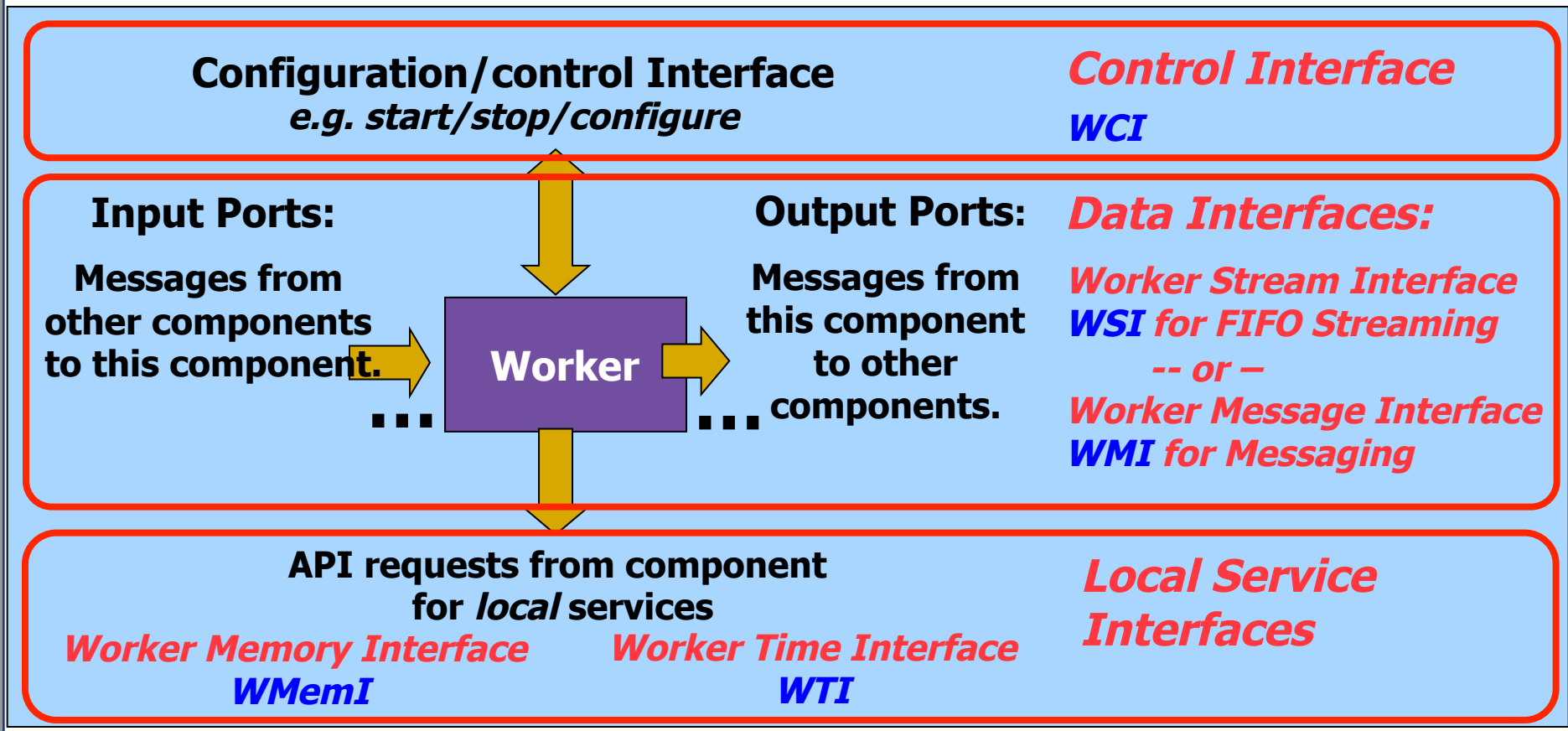
- Uses a subset of OpenCL
  - Uses the OpenCL dialect of C99 as language for workers
  - Supports the "kernel built-in functions" as in OpenCL
  - Does ***not*** require or expose the OpenCL ***host*** API
  - OpenCL knowledge needed is limited to kernel coding
  - Overall, a simpler process than "native OpenCL"

- Writing an OCL worker is (*just*) an OpenCL kernel
  - Almost any OpenCL "kernel" is easily re-cast as an OCL worker.

- Conceptually similar to RCC authoring model

HDL (HW Description Language) Authoring Model

- For FPGAs/ASICs using HDLs like Verilog, VHDL*, Bluespec*
- "Velvet handcuffs" to achieve commonality, portability, interoperability, and tools leverage
- Defines 5 lightly parameterized interfaces, based on OCP



All Interfaces achieved via OCP profiles

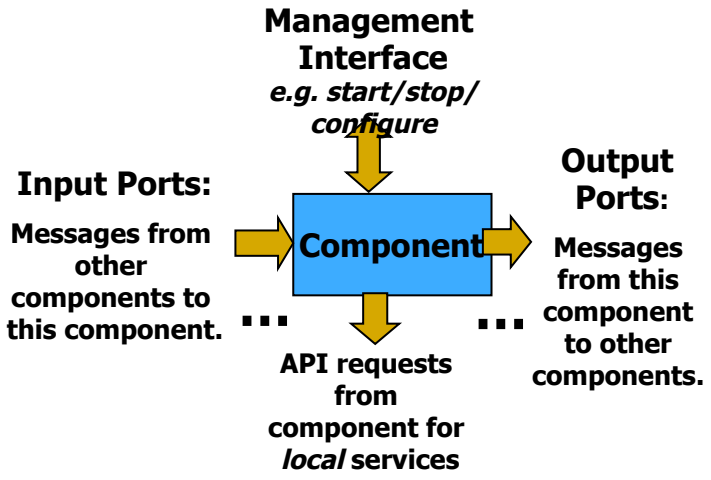
- Open Core Protocol:
  - An open standard for defining how "IP Cores" are connected
  - A broad range of clearly defined interface choices
  - *Profiling is key to appropriate use of OCP*
  - Well defined signaling protocols/timing between "cores".
  - From SoC/ASIC world: very robust and mature
- Language and Vendor independent
  - VHDL vs. Verilog vs. BluespecSV etc., doesn't matter
- FPGA "Workers" are natural "IP Cores".
- OCP mandated by several DoD programs

Interface Profiles for the HDL Authoring Model

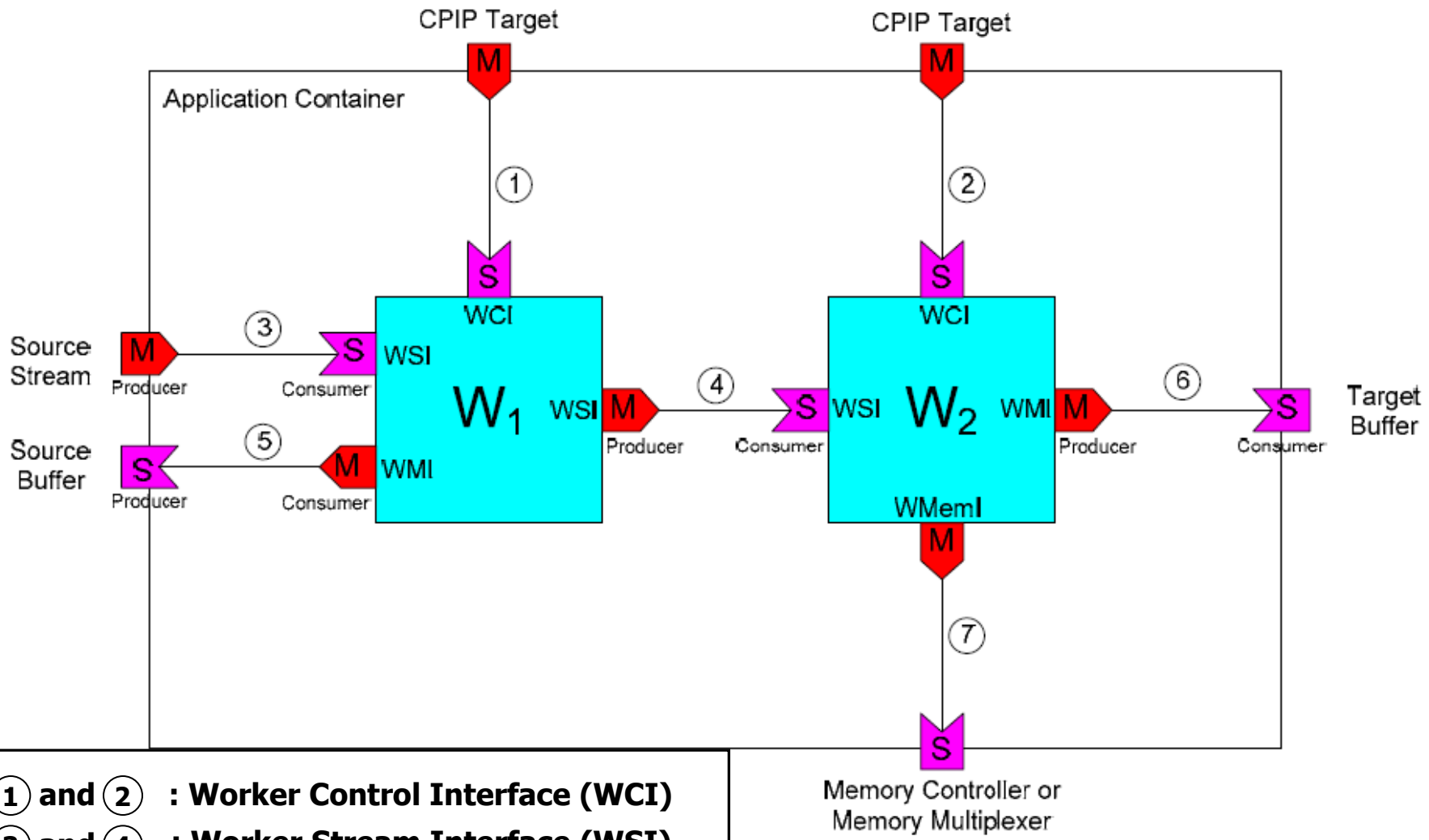
- Worker Control Interface (WCI):
  - Provides control and configuration: the control plane
- Worker Stream Interface (WSI):
  - Provides unidirectional FIFO streaming with flow control.
- Worker Message Interface (WMI):
  - Enables producing/consuming data from addressable buffers.

2 choices for data ports

- Worker Memory Interface (WMemI):
  - Provides access to local memory:
- Worker Time Interface (WTI):
  - Provides access to precise time of day

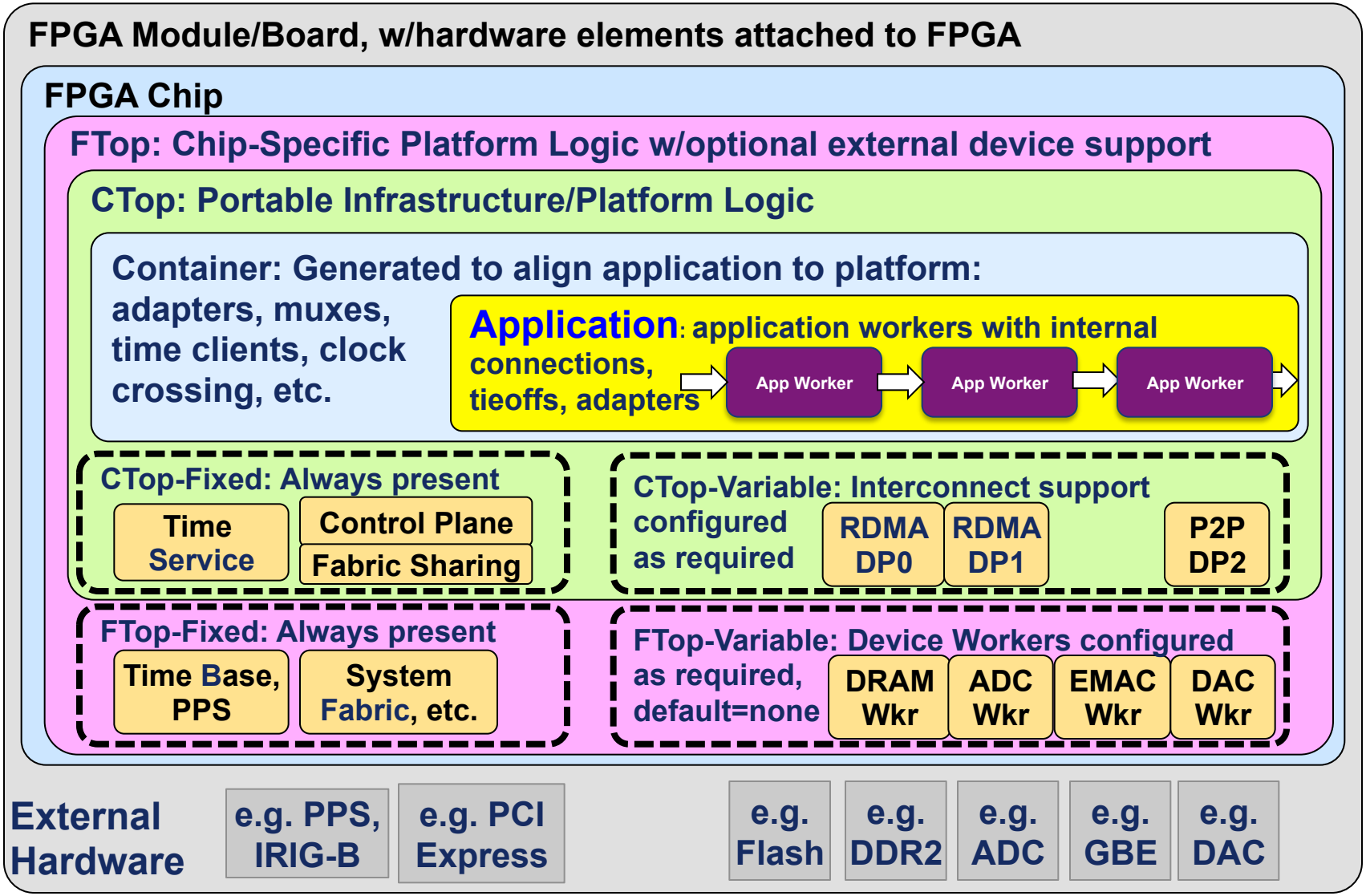


Example: Two FPGA Components in a Container

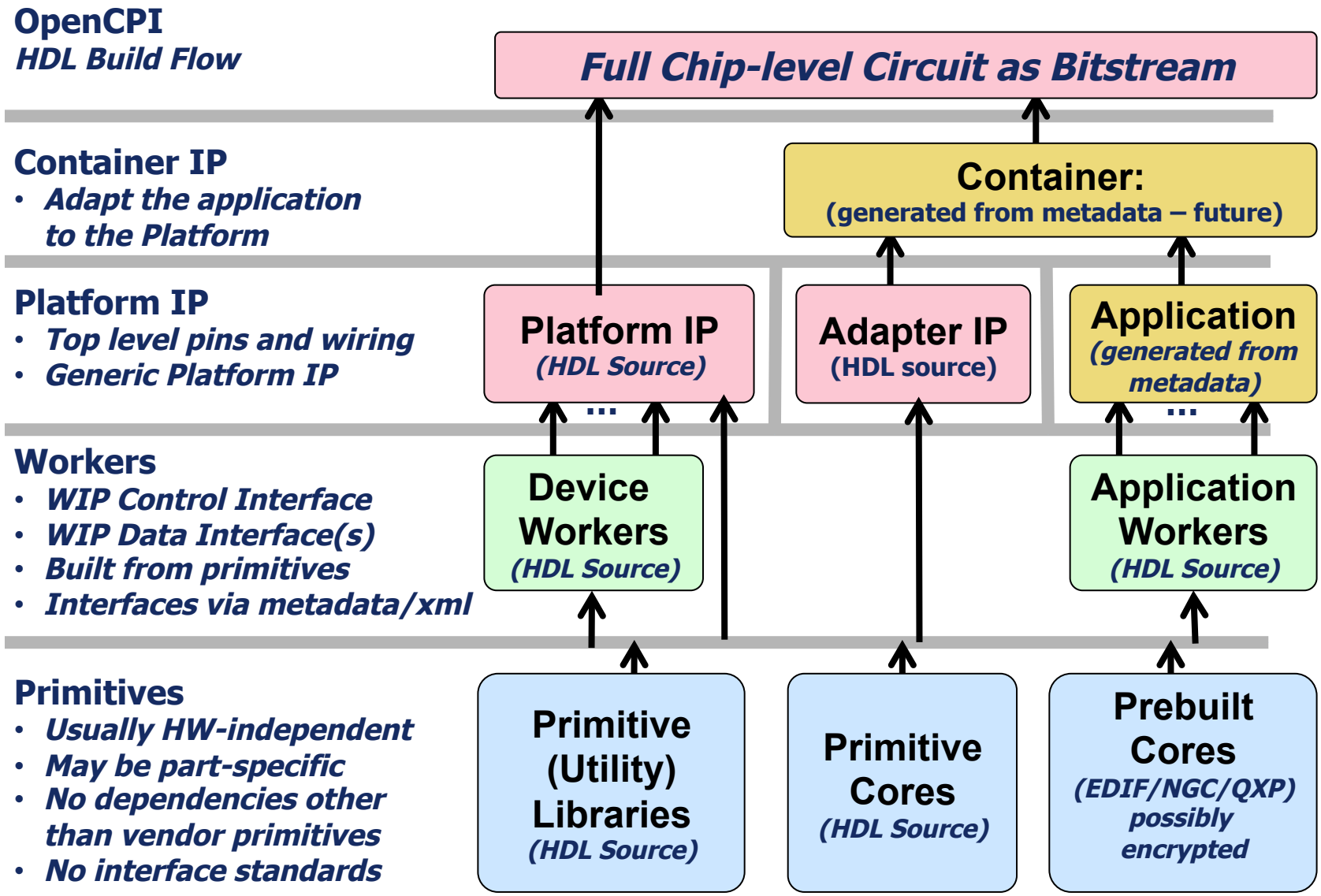


(1) and (2) : Worker Control Interface (WCI)
(3) and (4) : Worker Stream Interface (WSI)
(5) and (6) : Worker Message Interface (WMI)
(7) : Worker Memory Interface (WMemI)

OpenCPI Common FPGA On-Chip Architecture "stack"



OpenCPI HDL/FPGA Build Flow

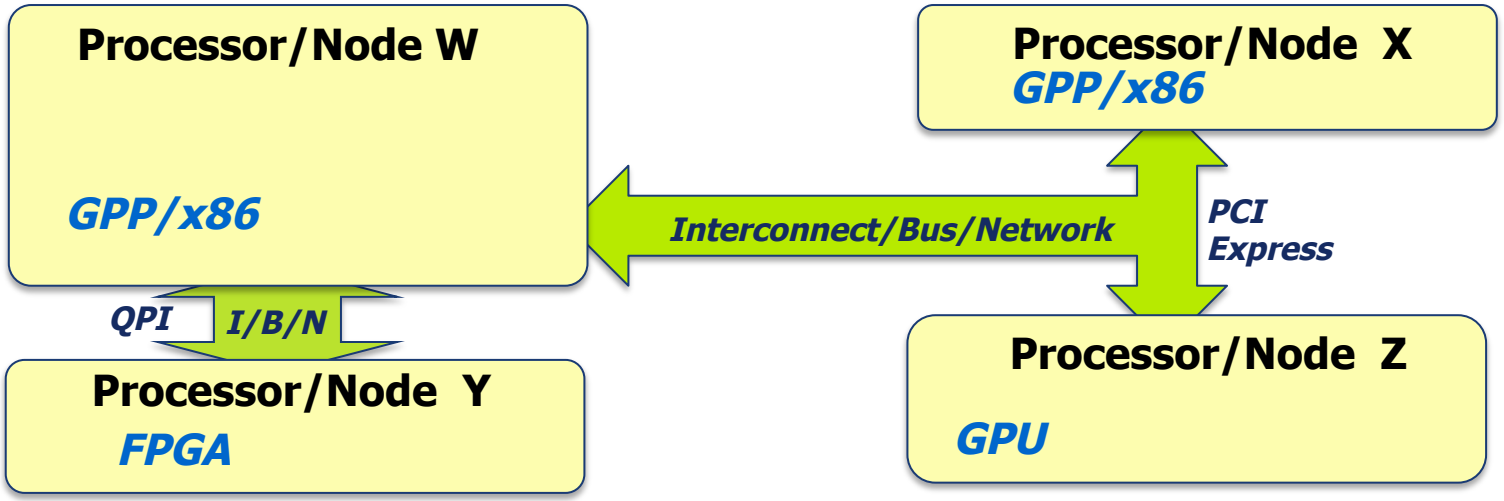


OpenCPI Briefing

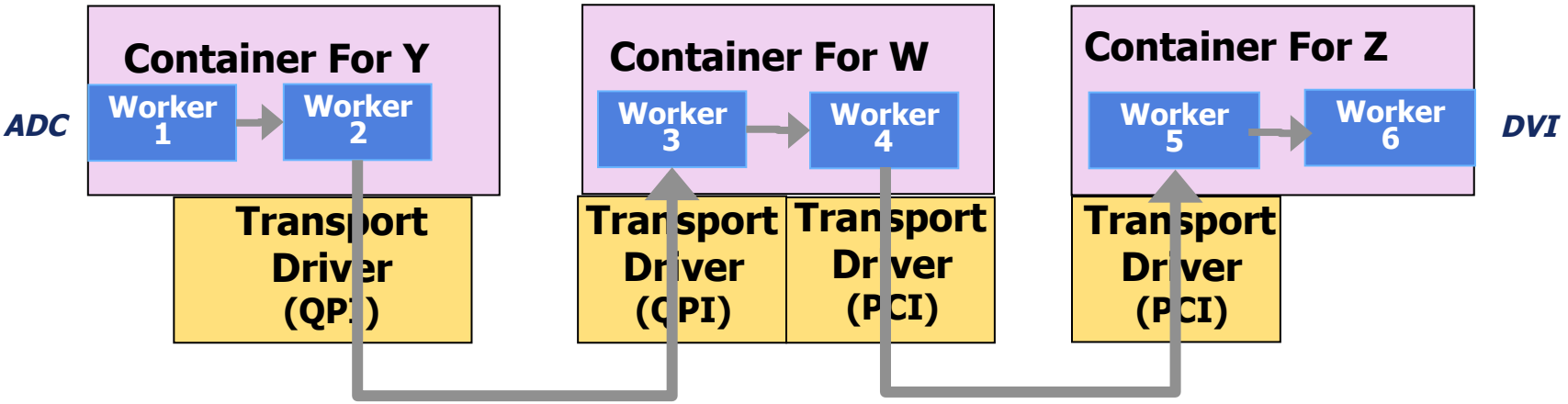
- Model/terminology of component-based systems
  - Consistent with other component-based systems
- Components
  - Describing, writing, building
- Applications
  - Specifying, controlling, executing
- Authoring Models for different technologies
  - GPP, DSP, GPU, FPGA…
- **Platform support for component-based applications**
  - Infrastructure for running component-based applications

What is an OpenCPI System/Platform?

- Hardware: processors and interconnects

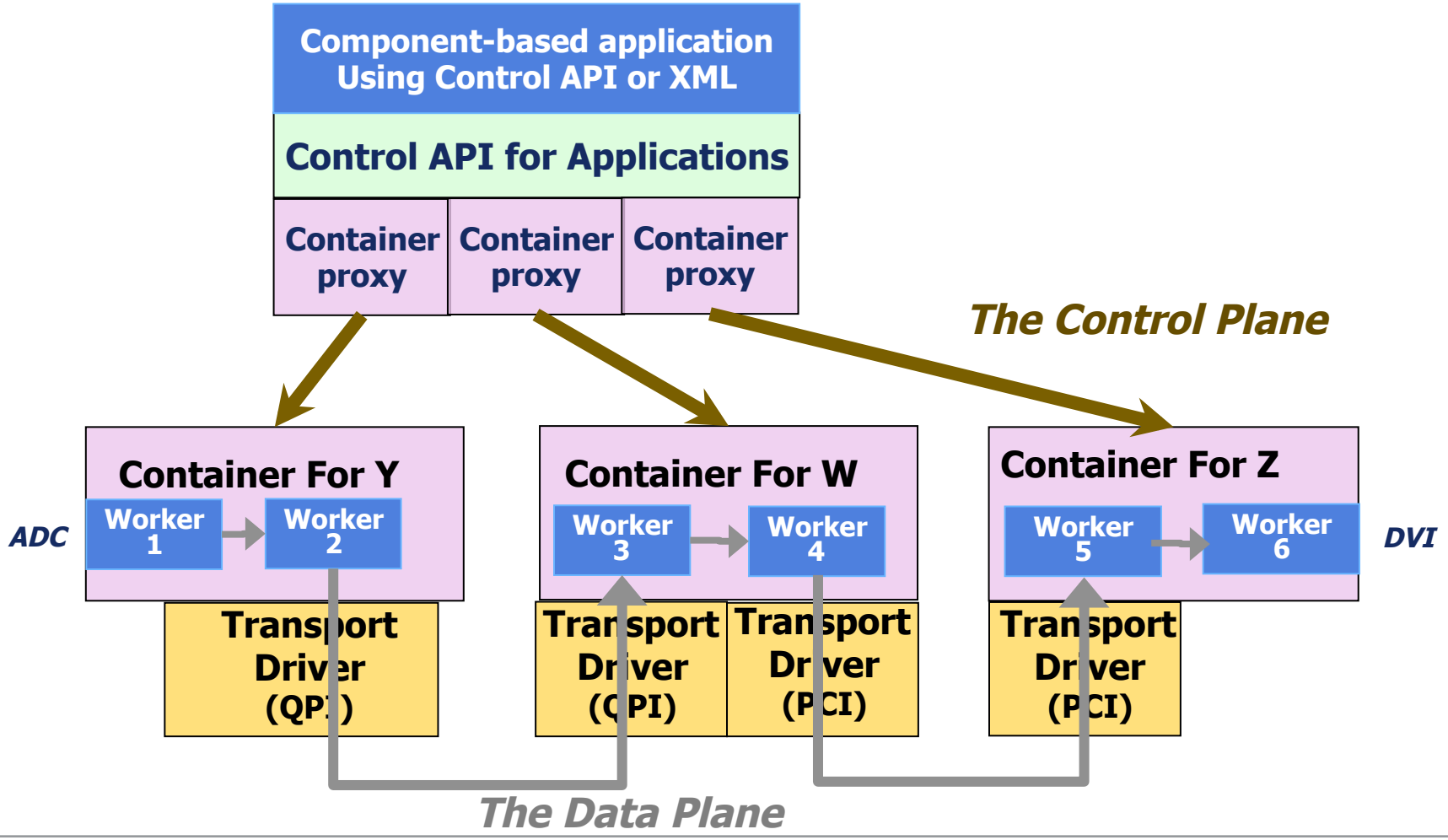


- Software: containers and transport drivers



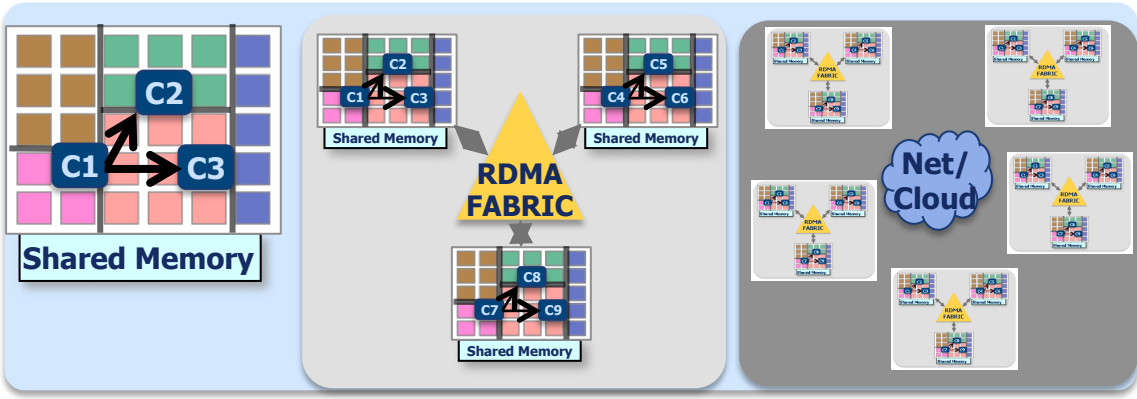
OpenCPI Infrastructure to Run/Launch applications

- The top level control application
  - Either using the OpenCPI Control API or XML app
  - Running on some "host" processor



Supported containers and transports

- Containers
  - RCC: Linux x86_64
  - RCC: Linux ppc_32*
  - RCC: Vxworks on ppc*
  - RCC: Windows*
  - RCC: C64 DSP BIOS*
  - RCC: Darwin/MacOS
  - OCL: OpenCL NVidia
  - OCL: OpenCL AMD*
  - OCL: OpenCL CPU*
  - HDL: Altera Stratix4
  - HDL: Xilinx Virtex5/Virtex6
- Transports (RDMA model)
  - OS Shared Memory
  - RDMA over sockets
  - RDMA over OFED/Infiniband/iWarp
  - PCI Express DMA (FPGA + GPP)



- External Services connectivity
  - CORBA
  - DDS

Agenda

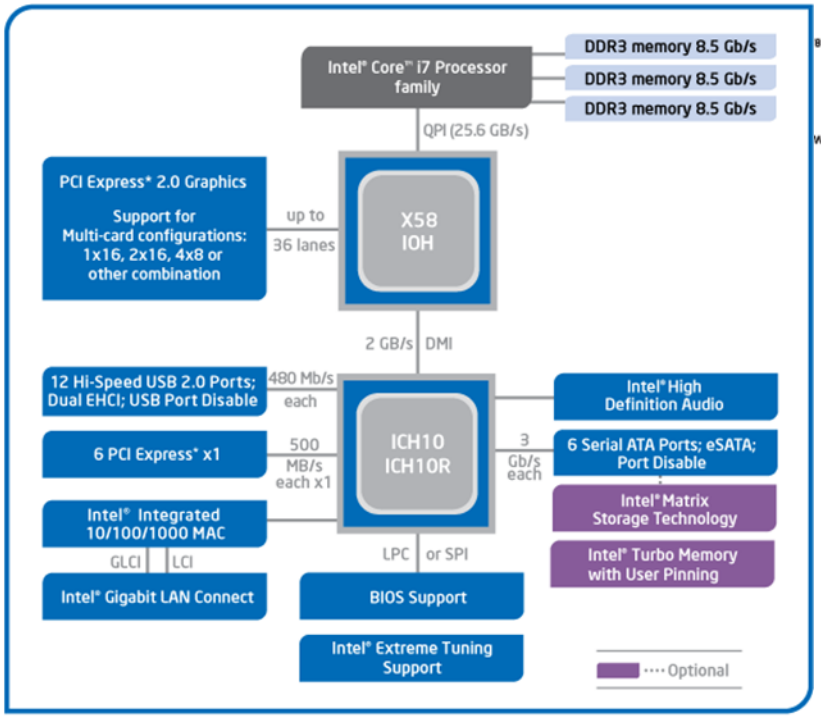
- OpenCPI Introduction, Goals, History 9:00
- OpenCPI Briefing 9:15
- **Recent Developments under SPI/S3 Program 10:00**
- Break 10:30
- Roadmap Ideas, Discussion 10:45
- Workflow, code examples 11:15
- Q&A, Wrap-up 11:45

SPI/S3 Phase 1, in 2009 (BAA 08-03-RIKA)

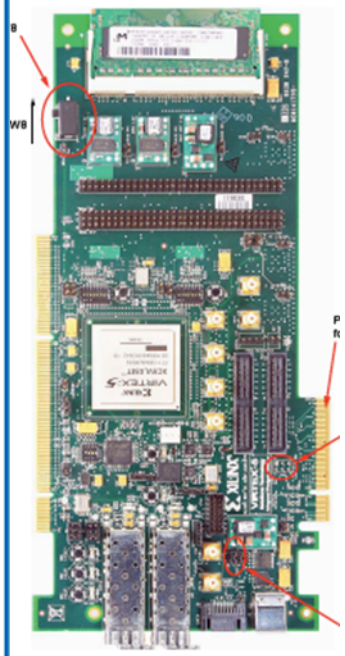
- Primarily: transition to open source
  - Clean-room replace all proprietary aspects
  - Investigate licensing issues
  - Establish open repository/site
- Study application of OpenCPI to GPU computing
  - A 5th authoring model, after RCC, HDL, XM, SCA
  - OpenCL investigation/assessment
- Define and support development platform
  - Purchase and assembly heterogeneous development platform & tools
  - Exact BOM for all parts and tools, ~$5K, including all FPGA tools.
  - Establish roadmap
- OpenCPI project won the down-select for Phase 2

A modern, low-cost, powerful reference system

- Easy to buy, easy to try, GPP+FPGA+GPU,
- Latest technology with high speed slots for other technologies
- 3 high speed slots for other processors: GPU, FPGA, Cell
- Inexpensive: Box w/FPGA+GPU boards + all tools = ~$5K



Intel® X58 Express Chipset Block Diagram

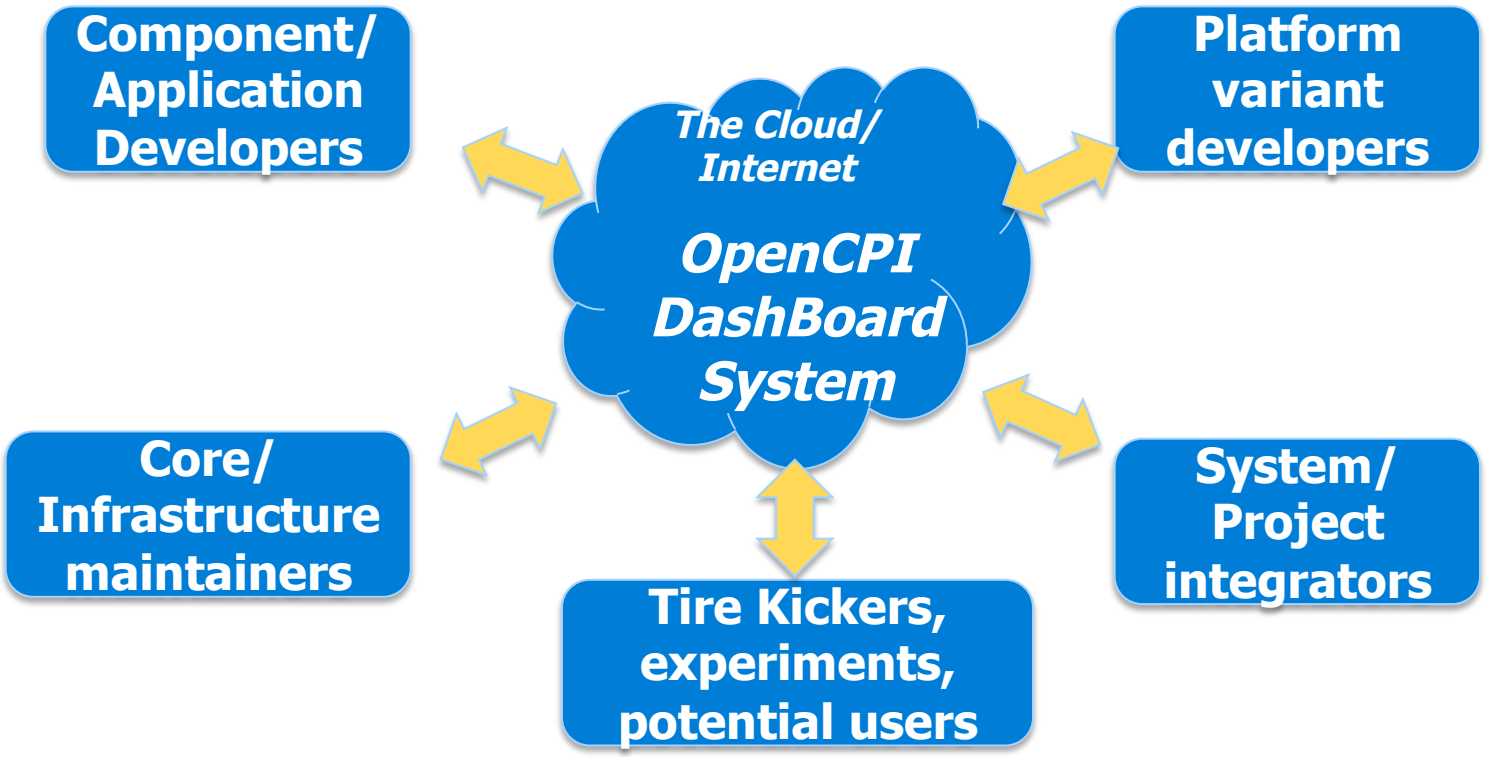


SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-3/2012)

- **Testing infrastructure**
- New authoring model: GPU Support
- New inter-component transport: Infiniband/iWarp/RDMA
- OpenCV image processing integration
- Add Altera support for Xilinx/Altera neutrality
- DDS middleware interoperation for external connectivity
- Benchmarking/measurement capabilities.
- Processor and implementation selection policies
- SDR/SCA (re)integration

Testing Infrastructure

- Selected/implemented "Google Test" as project test framework.
- Implemented synthesizable FPGA Protocol Monitors
- Implemented OpenCPI DashBoard web-based build/test center
- Continuous integration, orchestrating distributed build and test
  - Providing different benefits to different parties.



SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-2/2012)

- Testing infrastructure
- **New authoring model: GPU Support**
- New inter-component transport: Infiniband/iWarp/RDMA
- OpenCV image processing integration
- Add Altera support for Xilinx/Altera neutrality
- DDS middleware interoperation for external connectivity
- Benchmarking/measurement capabilities.
- Processor and implementation selection policies
- SDR/SCA (re)integration

Some Metrics of OCL Authoring Model

- Lines of code (OpenCL verse OpenCPI/OCL)
  - Host code
  - Kernel code
  - Other – XML
- Performance (elapsed time)
  - Is there an OCL performance penalty?

Metrics: Lines of Code

	Vector Add		FFT1D		Matrix Multiply	
	OpenCL	OCL	OpenCL	OCL	OpenCL	OCL
Host	277/49	204/19	301/37	200/15	498/51	239/19
Kernel	9	22	905	918	91	104
XML	0	11	0	15	0	11

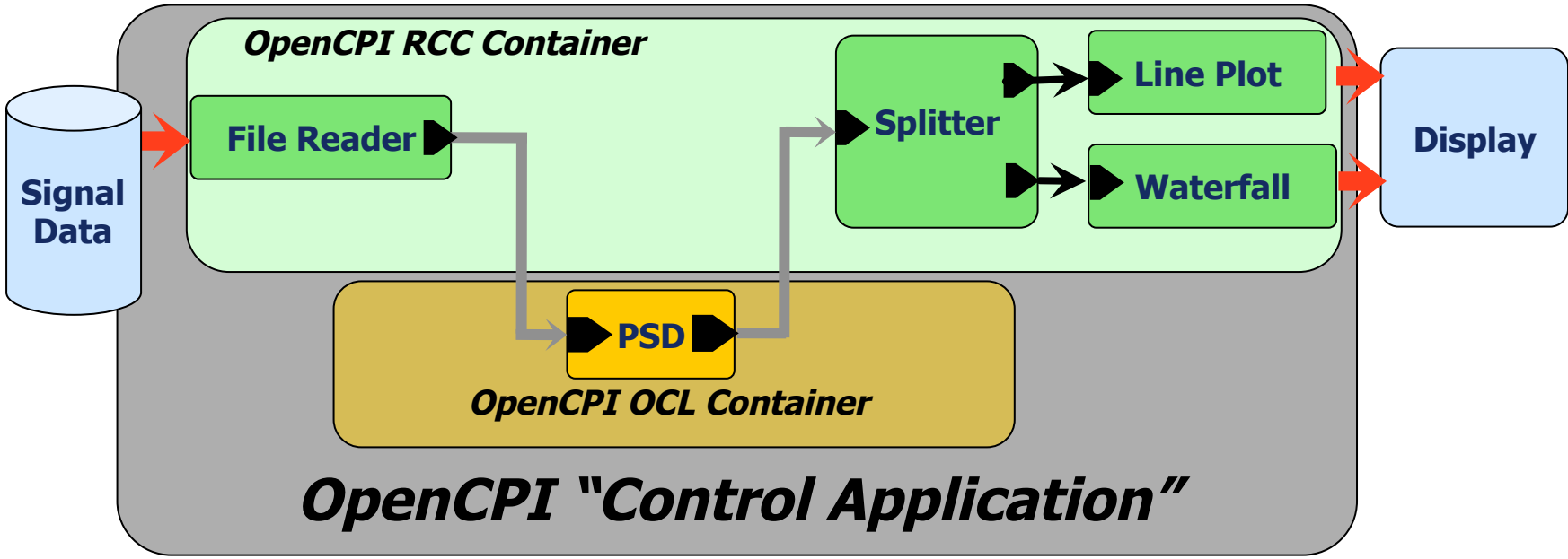
total SLOC/relevant SLOC

Metrics: Performance

- Performance data collected using:
  - NVIDIA OpenCL 1.1 version 259.19 driver
  - NVIDIA GTX 480 GPU (peak 1.35 **T**FLOPS)
  - Red Hat 5.7 with six Core i7 X 980 processors running at 3.33 GHz.

	Vector Add		**FFT1D**		**Matrix Multiply**	
	OpenCL	OCL	OpenCL	OCL	OpenCL	OCL
Milliseconds	28.66	28.66	880.64	880.65	8.21	8.21
Notes	• Vector length 11.5 million 32-bit floats • I/O bound • NVIDIA kernel		• 1024 1K 1D FFTs • 400 GFLOPS ($5*NLog_2(N)$ • I/O bound • Kernel from Apple SDK		• Dimensions A(800 x 1600), B(800 x 800), C(800 x 1600) • I/O Bound (250 GFLOPS) •NVIDIA kernel	

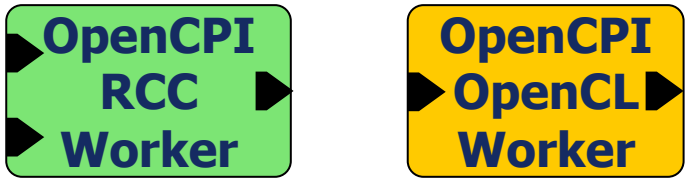
Power Spectrum Density Application example



Connections inside the OpenCPI Container (zero copy)

Connections between Containers

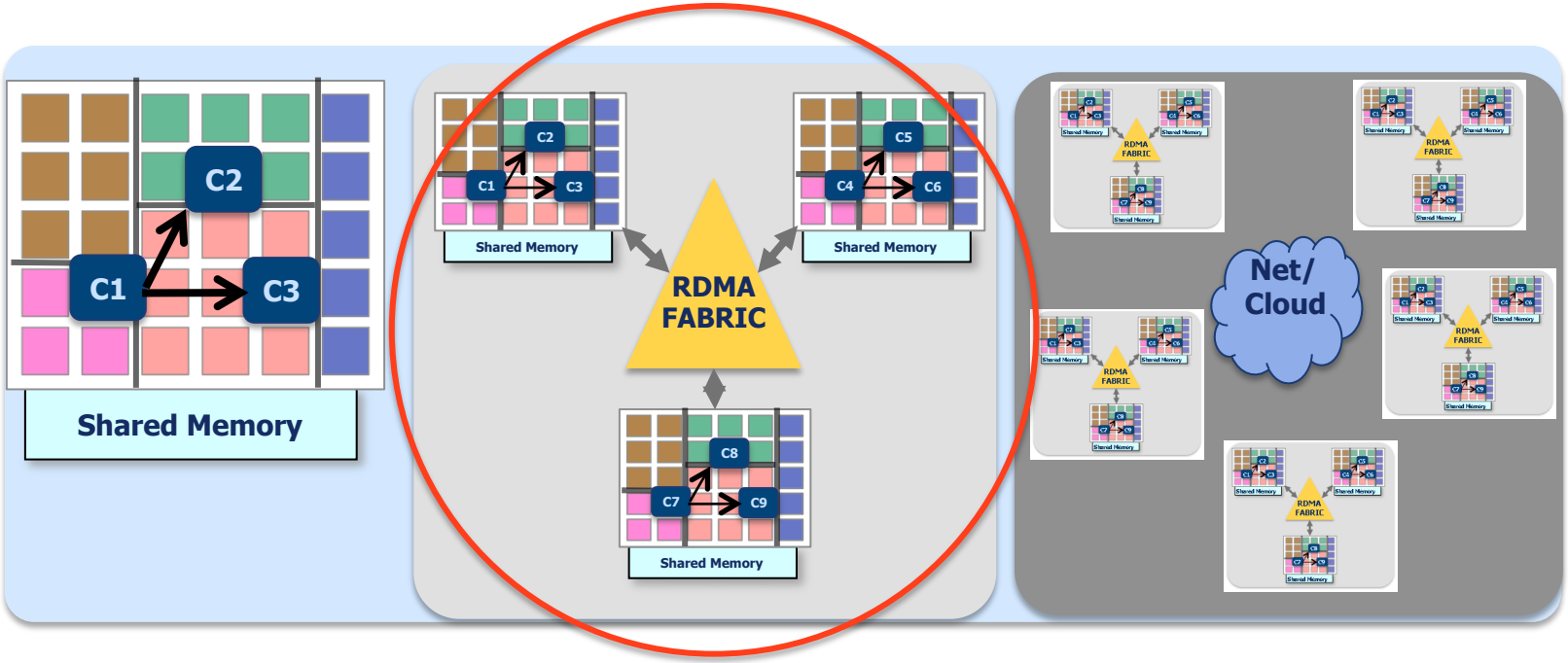
OpenCPI Internal I/O

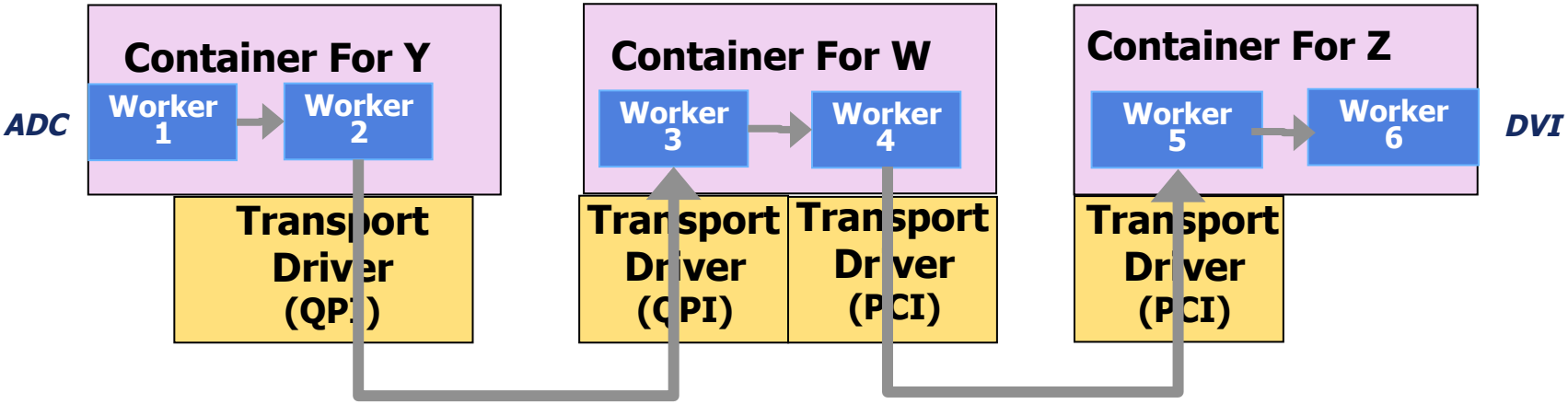


SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-2/2012)

- Testing infrastructure
- New authoring model: GPU Support
- **New transport: OFED (Infiniband/iWarp/RDMA)**
- OpenCV image processing integration
- Add Altera support for Xilinx/Altera neutrality
- DDS middleware interoperation for external connectivity
- Benchmarking/measurement capabilities.
- Processor and implementation selection policies
- SDR/SCA (re)integration

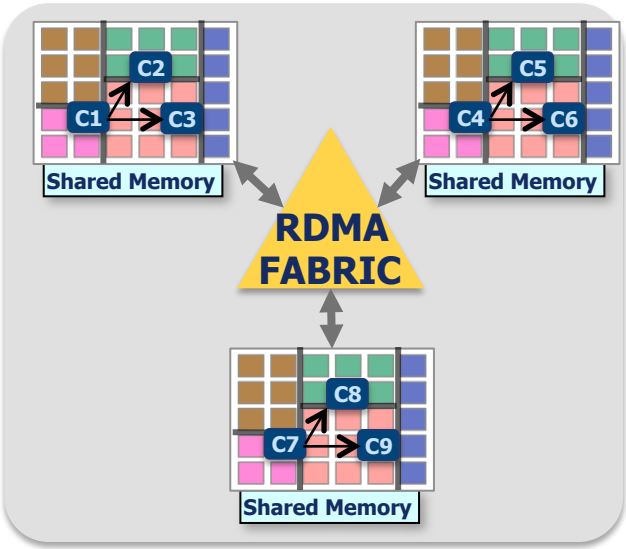
Add OFED to Data Plane Support in OpenCPI





Support for OFED RDMA as OpenCPI dataplane

- *Another way for components to talk to each other.*
- OFED: (Open Fabrics Enterprise Distribution)
  - Software stack for RDMA (remote DMA) technology
  - Most common usage is Infiniband-based clusters (Mellanox/Qlogic)
  - Low latency networks with HW protocol support, up to 40Gb/sec.
  - Common in computing clusters
  - Open-source, Linux-based comm SW stack



- Work was completed based on software emulation stack
  - "SoftRoCE" emulation of RDMA using sockets
  - Avoid requirement for actual RDMA hardware

- Extends range of OpenCPI deployments
  - High performance (cloud) clusters to small embedded systems.
  - Supports emerging "tactical cloud" deployments

OFED (Open Fabrics Enterprise Distribution)

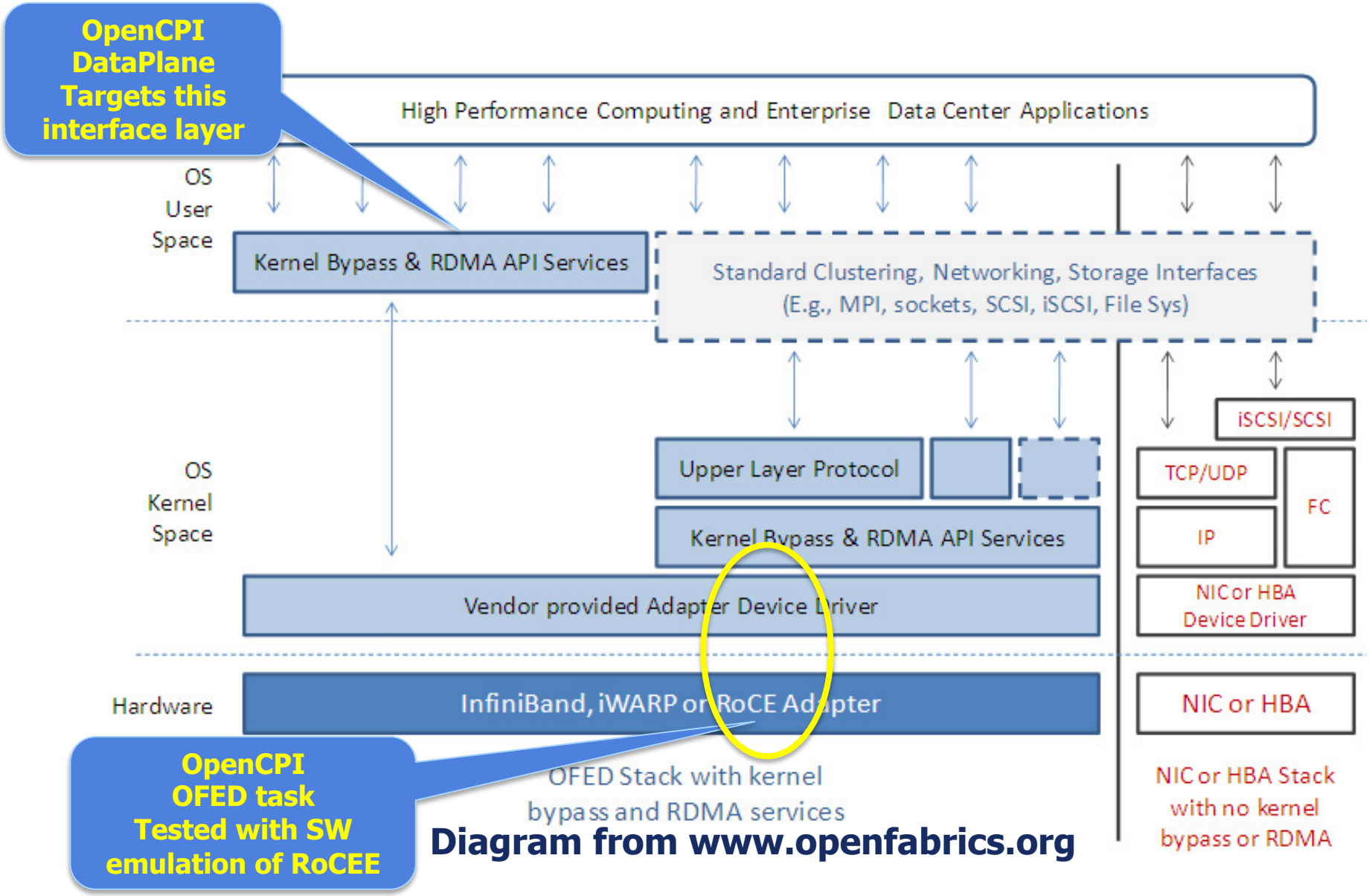


Diagram from www.openfabrics.org

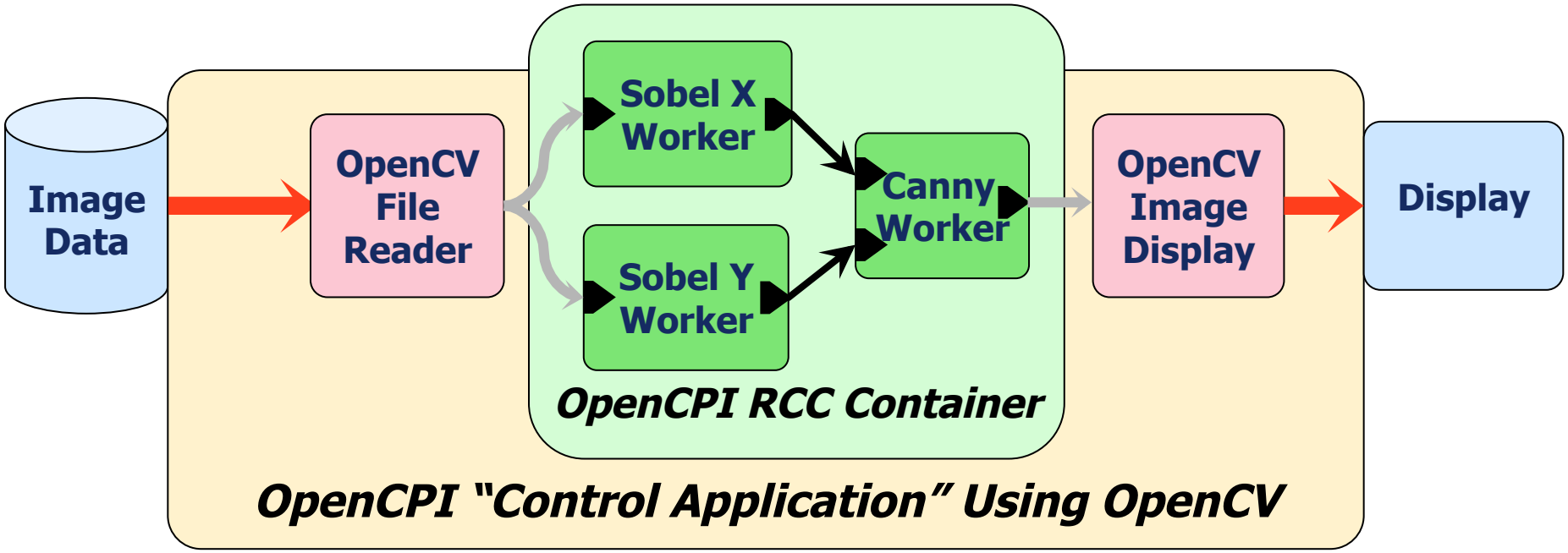
SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-2/2012)

- Testing infrastructure
- New authoring model: GPU Support
- New inter-component transport: Infiniband/iWarp/RDMA
- **OpenCV image processing integration**
- Add Altera support for Xilinx/Altera neutrality
- DDS middleware interoperation for external connectivity
- Benchmarking/measurement capabilities.
- Processor and implementation selection policies
- SDR/SCA (re)integration

MIT OpenCV Work (graduate student sponsorship)

- **Established library of 10+ algorithms:**
  - Basic: sobel/scharr/laplace/dilate/erode/blur/median/gaussian
  - Edge detection: sobel, canny
  - Optical Flow: gradient, box filter, eigenvalues, Lucas-Kanade
  - Feature ID/tracking: corner detection/motion detection/resolution-pyramid,
- **Used in three demo applications.**

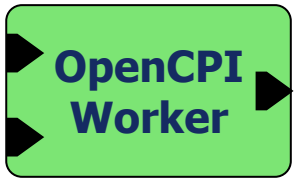
Canny Edge Detection Application:

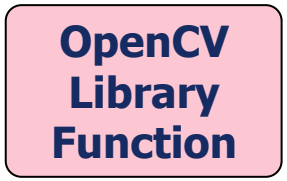


Connections inside the OpenCPI Container (zero copy)

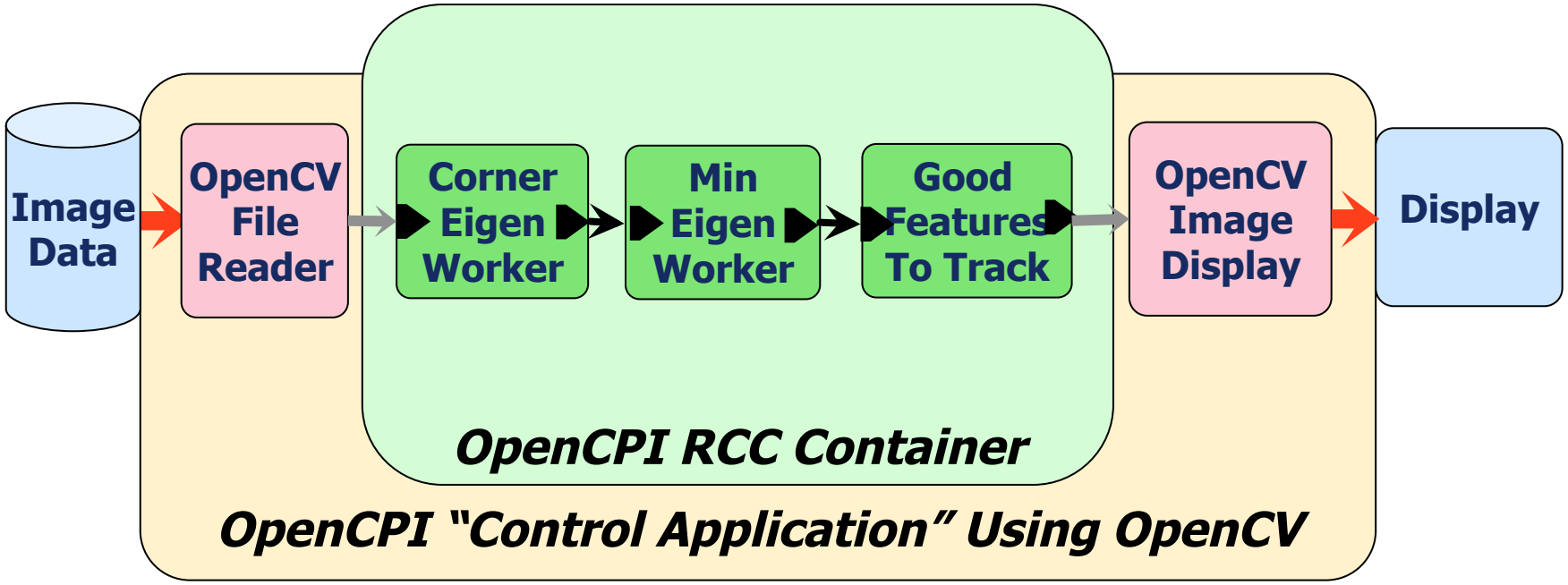
Control Application Connection between OpenCV and OpenCPI Container

OpenCV Internal I/O





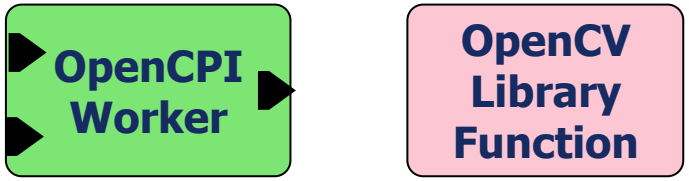
Feature Detection Application



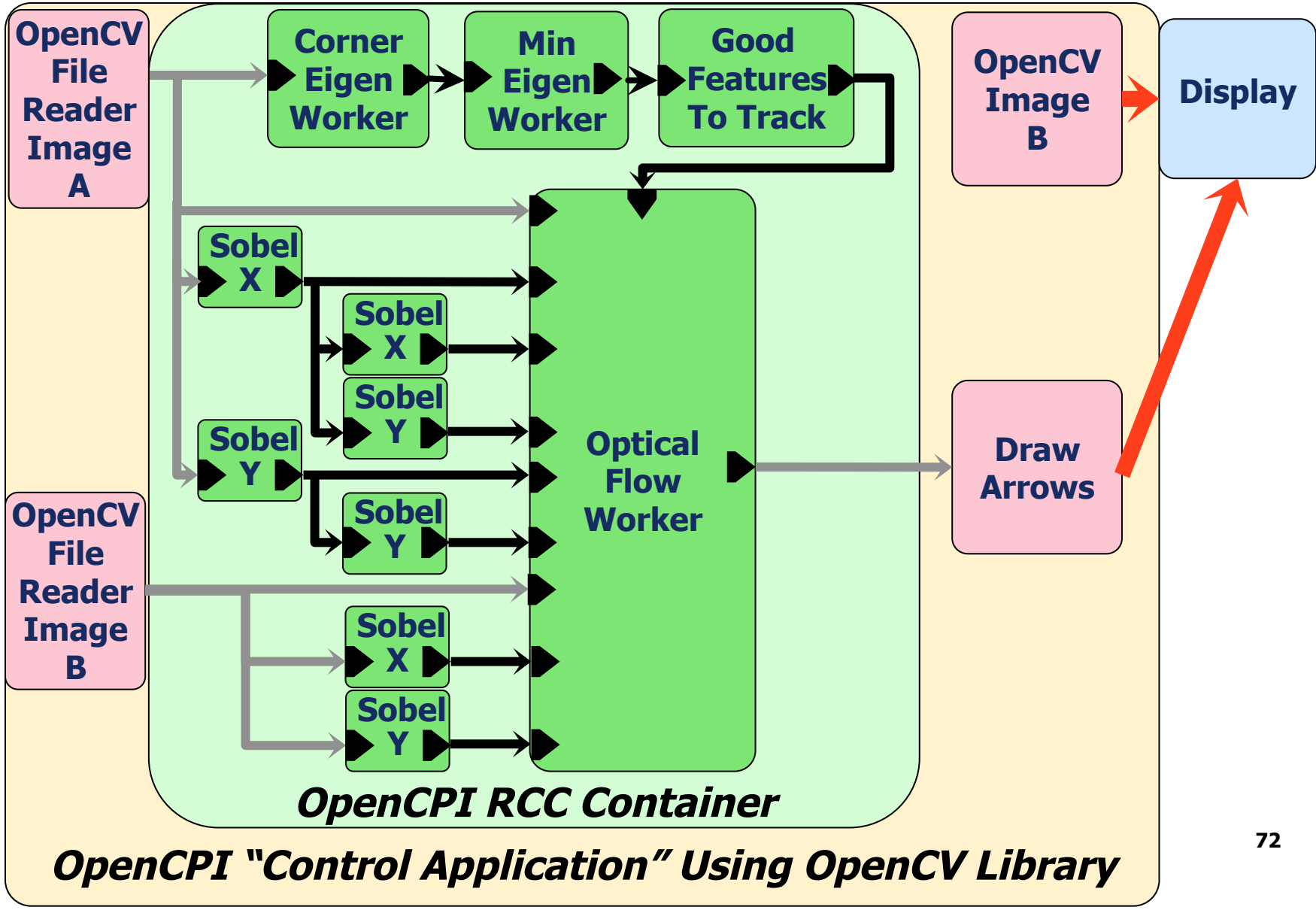
→ Connections inside the OpenCPI Container (zero copy)

→ Control Application Connection between OpenCV and OpenCPI Container/Workers

→ OpenCV Internal I/O



Optical Flow Application:



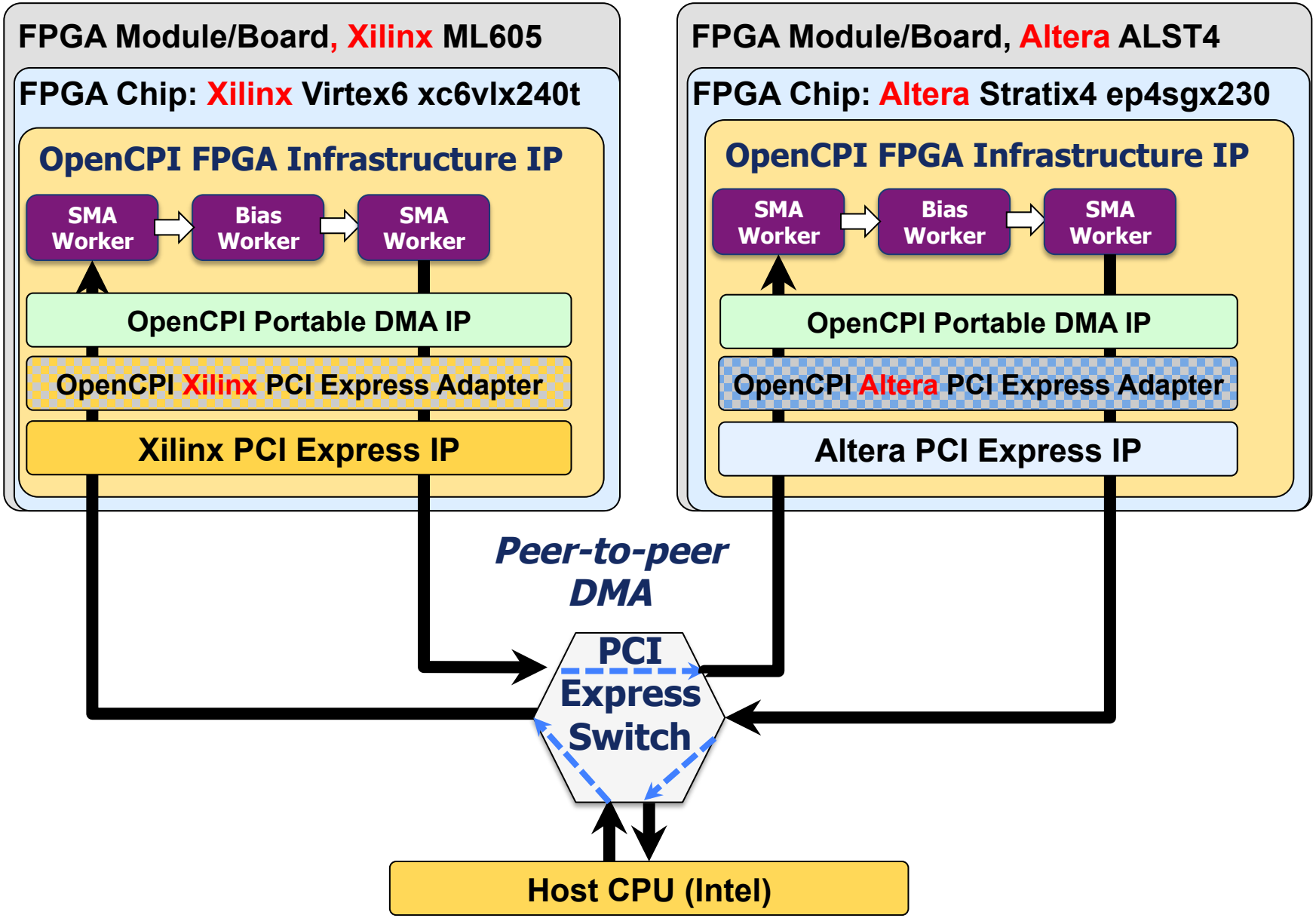
SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-2/2012)

- Testing infrastructure
- New authoring model: GPU Support
- New inter-component transport: Infiniband/iWarp/RDMA
- OpenCV image processing integration
- **Add Altera support for Xilinx/Altera neutrality**
- DDS middleware interoperation for external connectivity
- Benchmarking/measurement capabilities.
- Processor and implementation selection policies
- SDR/SCA (re)integration

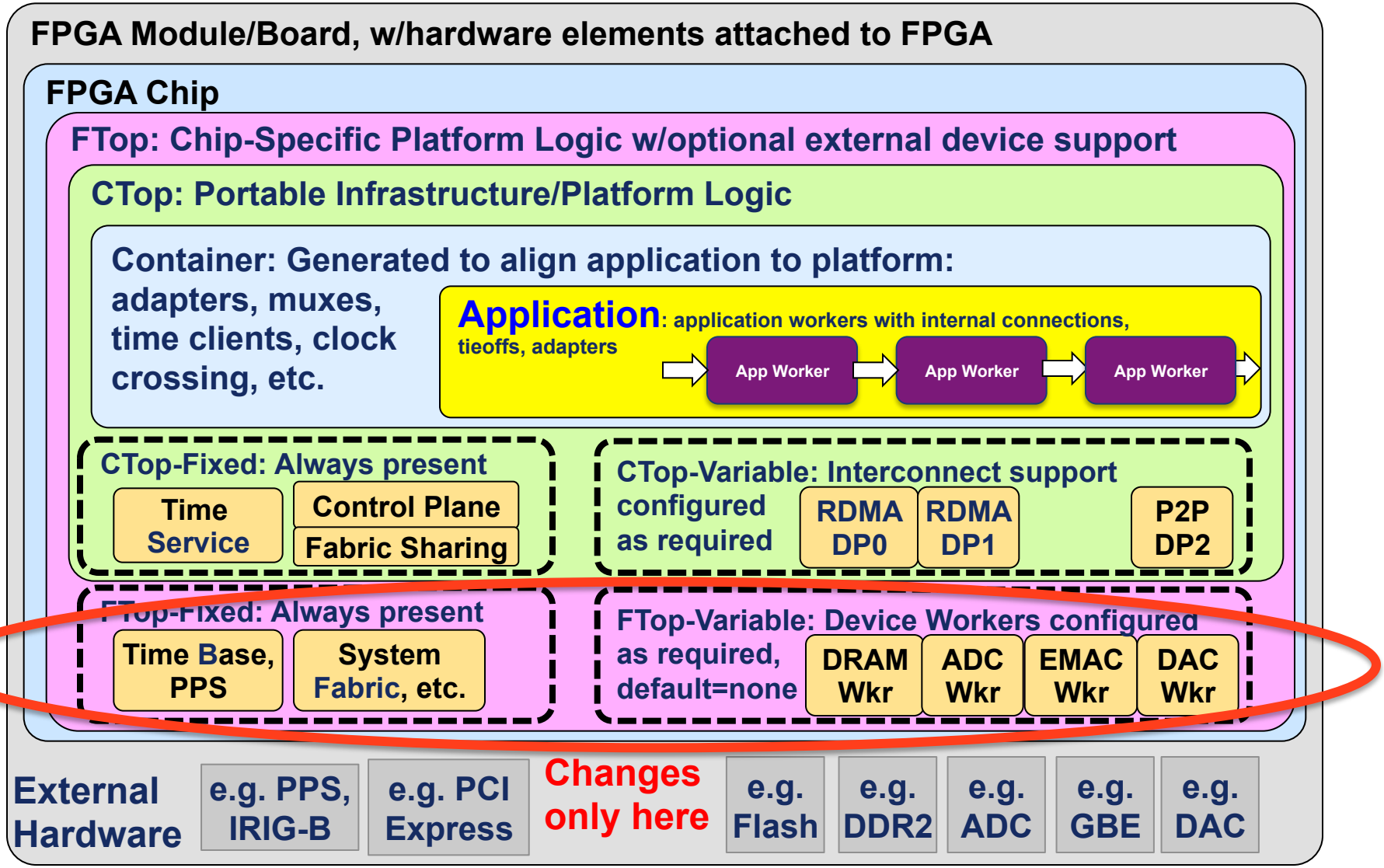
FPGA from Xilinx-centered to vendor neutral/Altera

- All the initial OpenCPI FPGA support was Xilinx-centered
  - This program wanted vendor neutrality, proven.
  - The "other" big fish in the pond is Altera.
- Coding and architecture more alike than different.
  - All the diversity is in the details, especially tools/workflow.
- Different Tools
  - Synthesis Backend: Quartus vs. ISE
  - Functional Simulation: ModelSim vs. Isim
  - In-Circuit Debug: SignalTap vs. ChipScope
- Wrapping Vendor-Specific Cores
  - PCIe, DDR3 DRAM, others
  - Hiding low-level differences
  - Exposing common "wrapped" abstraction

Demo Data Flow, Xilinx and Altera Interoperating



OpenCPI Common FPGA On-Chip Architecture "stack"

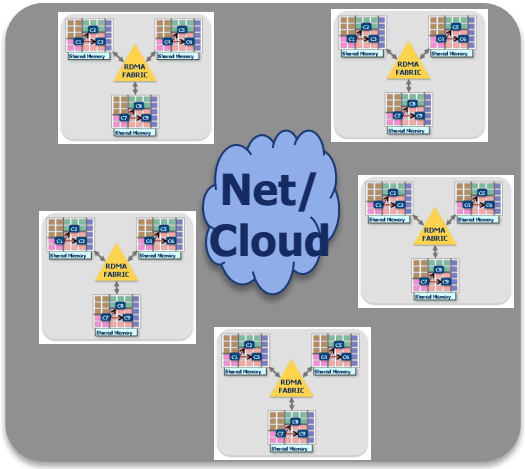


SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-2/2012)

- Testing infrastructure
- New authoring model: GPU Support
- New inter-component transport: Infiniband/iWarp/RDMA
- OpenCV image processing integration
- Add Altera support for Xilinx/Altera neutrality
- **DDS middleware interoperation for external data-plane connectivity**
- Benchmarking/measurement capabilities.
- Processor and implementation selection policies
- SDR/SCA (re)integration

DDS (Data Distribution Service) Integration

- DDS 101:
  - Middleware for pub/sub data distribution
  - Based on OMG standard, like CORBA
  - Heavy emphasis on QoS and subscribers with varying needs for "freshness of data"
  - Used across DoD
  - Good fit for dissemination to users/subscribers



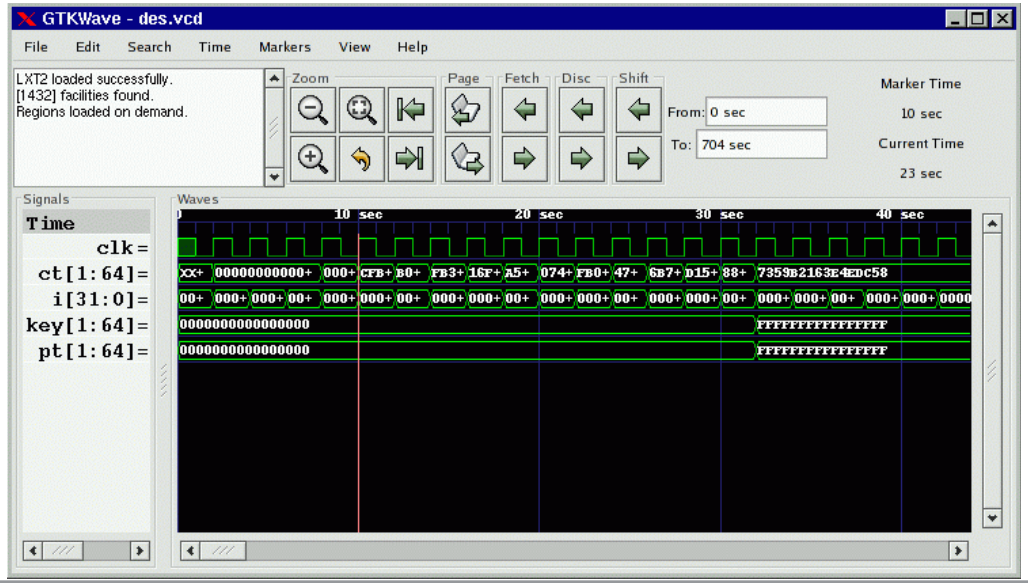
- DDS Integration with OpenCPI: three modalities
  - Any component's output can be made a DDS publisher
    - To disseminate to external DDS-based subscriber apps
  - Any component's input can be made a DDS subscriber
    - To receive from external DDS-based publisher apps
  - DDS can do data transport inside a distributed OpenCPI application.
    - Especially for distributed fan-out/multicast
    - Another way for components to talk to each other.

SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-2/2012)

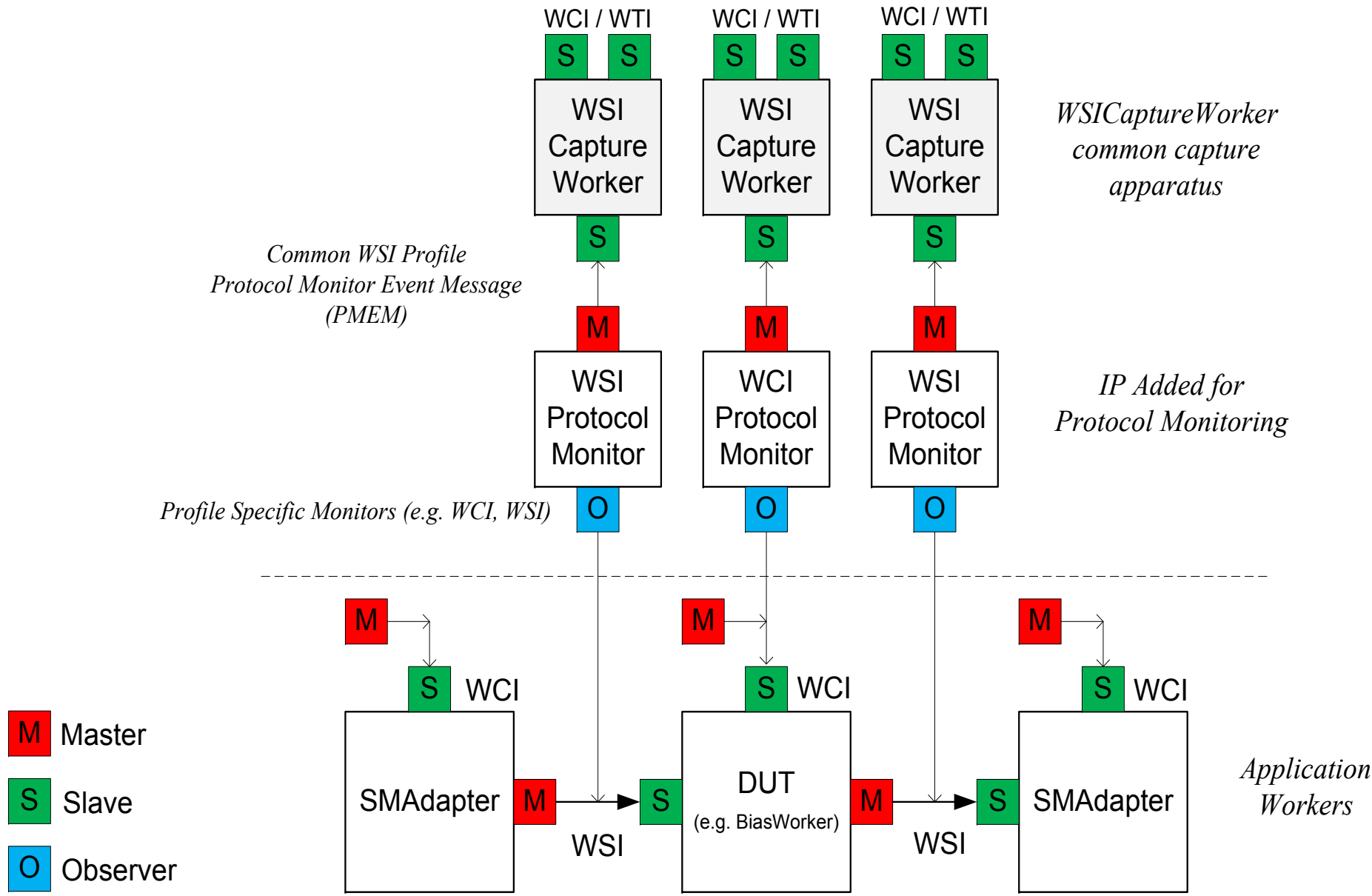
- Testing infrastructure
- New authoring model: GPU Support
- New inter-component transport: Infiniband/iWarp/RDMA
- OpenCV image processing integration
- Add Altera support for Xilinx/Altera neutrality
- DDS middleware interoperation for external connectivity
- **Benchmarking/measurement capabilities.**
- Processor and implementation selection policies
- SDR/SCA (re)integration

Benchmarking/instrumentation enhancements

- FPGA and CPU synchronization to 100s ns
- Capture events across software object hierarchy
  - As hardware-like events, also combined with hardware events
  - Separate capture poilcy/queueing
  - Viewable using GTKWave (usually used for HW simulation).



Capture FPGA protocol monitor events



SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-2/2012)

- Testing infrastructure
- New authoring model: GPU Support
- New inter-component transport: Infiniband/iWarp/RDMA
- OpenCV image processing integration
- Add Altera support for Xilinx/Altera neutrality
- DDS middleware interoperation for external connectivity
- Benchmarking/measurement capabilities.
- **Processor and implementation selection policies**
- SDR/SCA (re)integration

Processor and Implementation Selection Policies

- Dynamic Ability to Choose Best Implementation
  - Add "selection expressions" to implicitly "score" the available implementations based on user criteria.

```
<application>
  <instance worker="psd" selection="accuracy > 3">
…
```

- Dynamic Ability to Expand/Reduce Processors
  - Add worker-to-container allocation policies to allow control applications and/or XML users to bias the distribution of workers to processors: use minimum, use maximum, use limited numbers.

```
<application maxProcessors="3">
…
```

SPI/S3 Phase 2 (AFRL/ASDR&E 10/2010-2/2012)

- Testing infrastructure
- New authoring model: GPU Support
- New inter-component transport: Infiniband/iWarp/RDMA
- OpenCV image processing integration
- Add Altera support for Xilinx/Altera neutrality
- DDS middleware interoperation for external connectivity
- Benchmarking/measurement capabilities.
- Processor and implementation selection policies
- **SDR/SCA (re)integration**

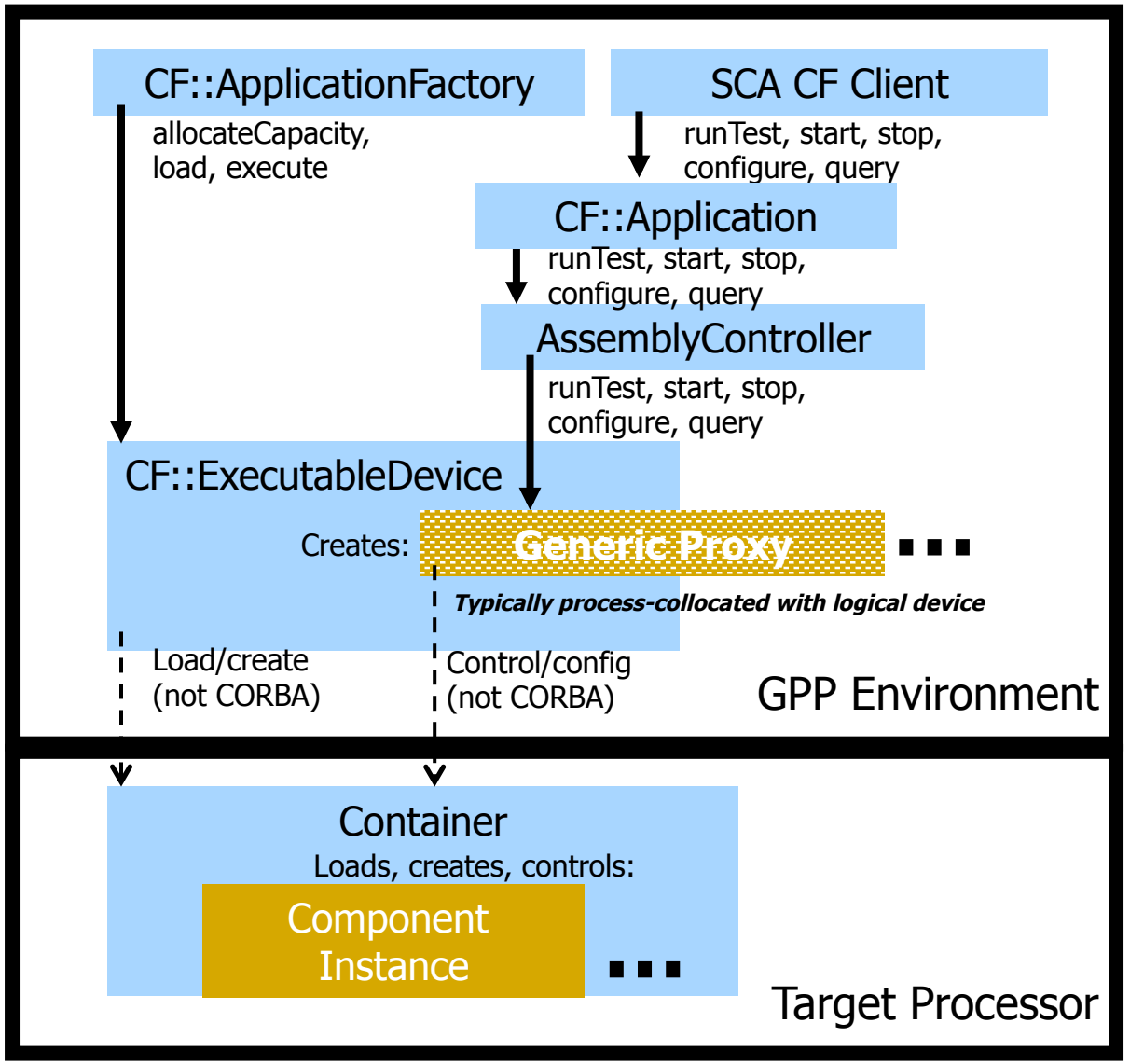
Key facts on How OpenCPI relates to the DoD SCA.

- OpenCPI is *not* a replacement for an SCA core framework
  - It supports a CF::ExecutableDevice interface for each container

- OpenCPI workers (DSP/FPGA/GPU etc.) are seen as SCA "Resources":
  - Coexist with runtime interoperability (data plane)
  - Are transparent to CF, assembly controllers, SCA waveform components
  - Can use same metadata (SPD/SCD/PRF + IDL)
  - Can use same modeling tools (e.g. SCARI and Zeligsoft/PrismTech)
  - Use IDL restrictions from OMG/ISO CORBA/E micro profile
  - (Preconnected workers require a patch in App Factory).

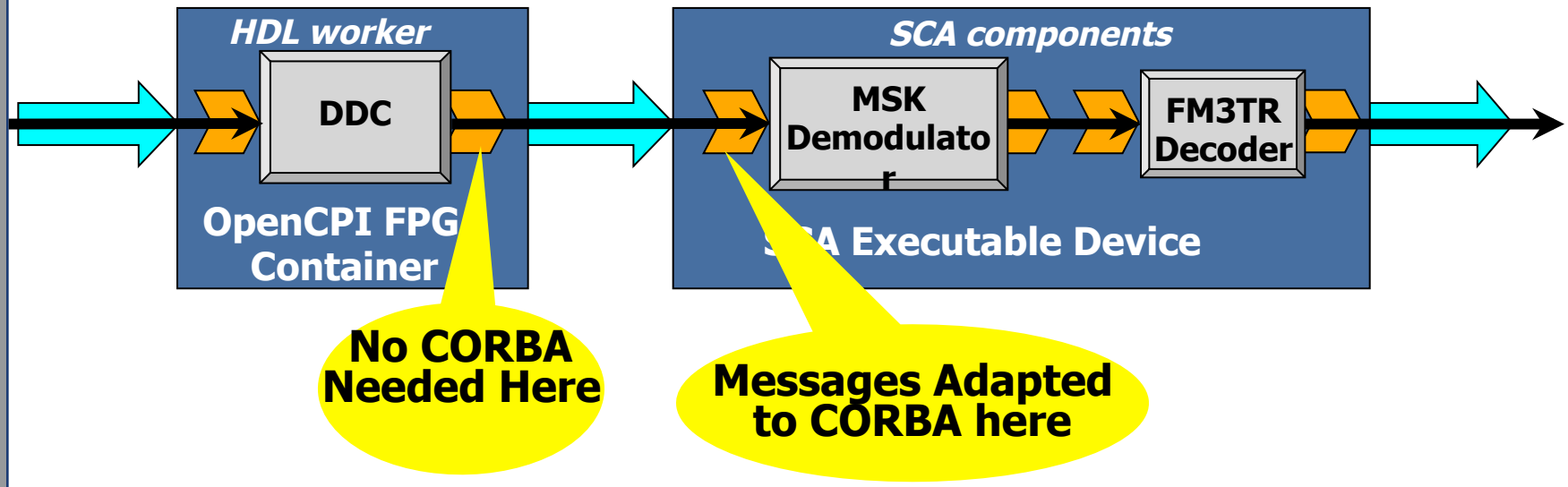
- Currently targeting open-source OSSIE CF, OMNI ORB, Linux
  - Previously used Harris/SCARI CF, OIS/TAO ORB, VxWorks-DKM/Linux

How SCA sees OpenCPI workers

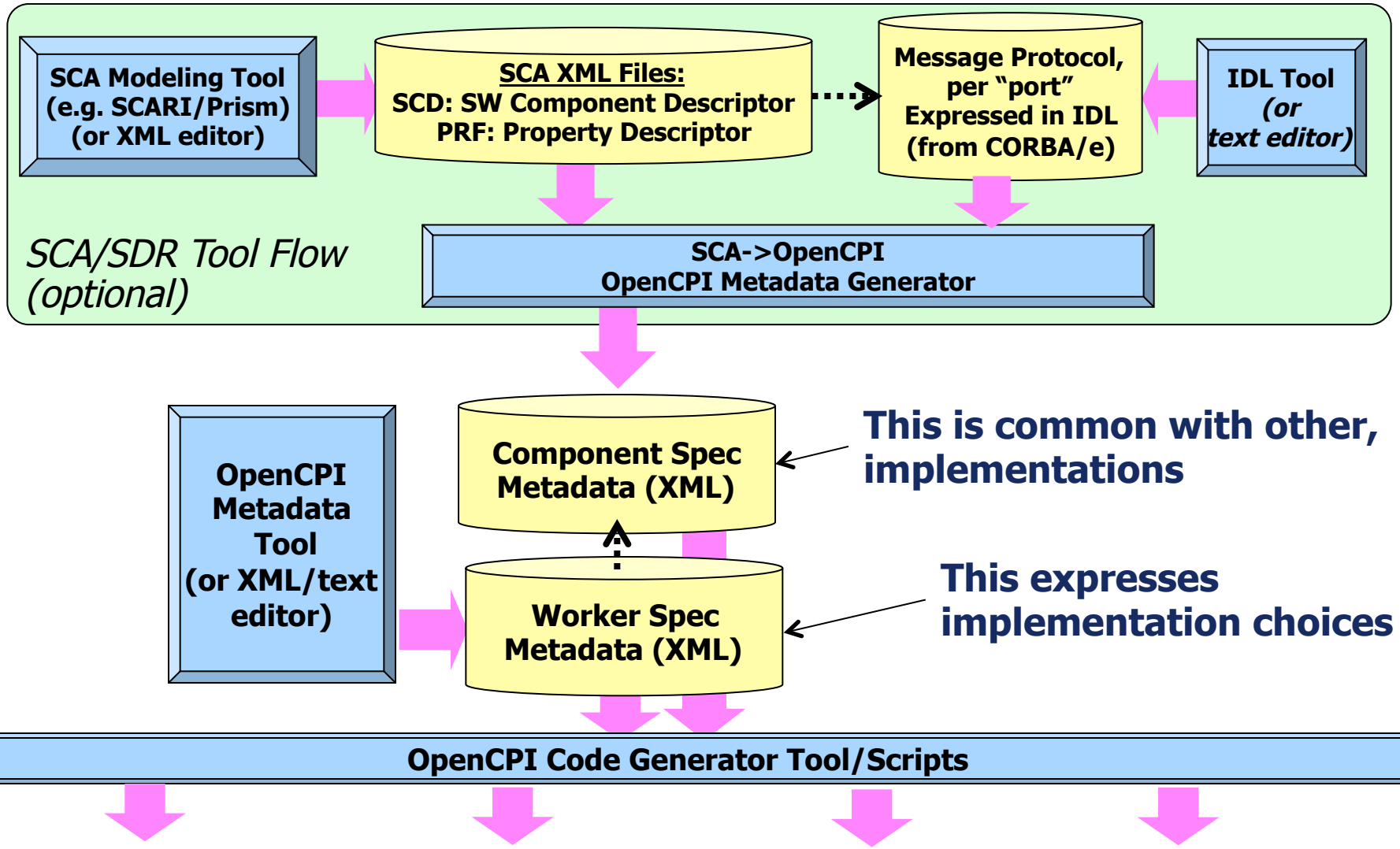


FPGA->GPP SCA data connectivity

- In SAD, an FPGA worker's port connects to an SCA/GPP component's data port.
- The GPP SCA component is written using CORBA normally, no adapter needed
- The messages are delivered so that:
  - **FPGA worker sees a simple message/burst at its data port, doesn't know/care what it is talking to**
  - **SCA component sees a CORBA "method invocation" at its data port.**
- OpenCPI provides an ORB transport that reformats messages as CORBA messages, on the GPP, where it is needed (e.g. inserting a GIOP header).



SCA-oriented unit dev flow, when SCA is required



Agenda

- OpenCPI Introduction, Goals, History 9:00
- OpenCPI Briefing 9:15
- Recent Developments under SPI/S3 Program 10:00
- Break 10:30
- **Roadmap Ideas, Discussion** **10:45**
- Workflow, code examples 11:15
- Q&A, Wrap-up 11:45

Roadmap Ideas

- Application-specific experiments
- Specific enhancements
- Community enhancements

Specific Enhancements

- Security
  - MILS OS: Components in partitions, etc.
  - Secure transports/barriers
- GIG integration
  - "sensors as a service", SOA+DDS etc.
- Processing technology, authoring model
  - Many-core, e.g. Tilera
  - Divide/conquer authoring models (e.g. CILK)
- More FPGA automation
- GPU Data Movement (GPU peer DMA without NDA).

Specific Enhancements

- Language support
  - More complete VHDL, Bluespec
  - RCC/C++, Java etc.
- Platform support: transport, OS, processor etc.
  - Examples: Zync (Xilinx SoC with ARM core + gates)
  - OMAP (ARM + DSP)
  - Integrity, Lynx, etc.
- Higher level parallelism (above processors)
  - System-level heterogeneous GPU-like decomposition
  - (From earlier DARPA work)
- Robustness:
  - More/better testing and verification
  - Code analysis
  - More/better documentation

Agenda

- OpenCPI Introduction, Goals, Background, History 9:00
- OpenCPI Briefing 9:15
- Recent Developments 10:00
- Break 10:30
- Roadmap Ideas, Discussion 10:45
- **Workflow, code examples** **11:15**
- Q&A, Wrap-up 11:45

Simple RCC worker code example: add a bias value

```
static RCCResult
run(RCCWorker *self, RCCBoolean timedOut, RCCBoolean *nrc) {
    RCCPort
        *in = &self->ports[BIAS_IN],
        *out = &self->ports[BIAS_OUT];
    BiasProperties *props = self->properties;
    uint32_t
        *inData = in->current.data,
        *outData = out->current.data;
    unsigned n;
    for (n = in->input.length / sizeof(uint32_t); n; n--)
        *outData++ = *inData++ + props->biasValue;
    out->output.length = in->input.length;
    return RCC_ADVANCE;
}
```

Baseline OpenCL Kernel/GPU side of vector add

```
__kernel void VectorAdd ( __global const float* a,
                          __global const float* b,
                          __global float* c,
                          unsigned iNumElements )
{
    // get index into global data array
    unsigned iGID = get_global_id(0);
    // bound check
    // (like limit on 'for' loops for serial C code)
    if (iGID < iNumElements)
      // add the vector elements
      c[iGID] = a[iGID] + b[iGID];
}
```

OpenCPI OCL Code Example: vector add worker

```
// The OpenCPI run function for the OCL model
OCLResult vector_add_run (__local OCLWorkerVectorAdd* self,
                          OCLBoolean timedOut) {
   VectorAdd(self->a.current.data,
             self->b.current.data,
             self->c.current.data,
             self->a.length / sizeof ( float ));
  self->c.length = self->a.length;
  return OCL_ADVANCE; // we consumed input & produced output
}
// The preexisting untouched OpenCL kernel for vector add
__kernel void VectorAdd(__global const float* a,
                        __global const float* b,
                        __global float* c,
                        unsigned iNumElements) {
  unsigned iGID = get_global_id(0);
  if (iGID < iNumElements)
    c[iGID] = a[iGID] + b[iGID];
}
```

Component Specification (from before)

- Describe the component in XML: the "spec file"
  - The OCS: OpenCPI Component Specification
  - Name, properties, and ports (and message protocol per port)
  - The basis for all implementations

```
<ComponentSpec Name="FFT1D">
  <Property Name="direction"/>
  <Property Name="scale" type="float"/>
  <DataInterfaceSpec Name="in">
    <xi:include href="stream_protocol.xml"/>
  </DataInterfaceSpec>
  <DataInterfaceSpec Name="out" Producer="true">
    <xi:include href="stream_protocol.xml"/>
  </DataInterfaceSpec>
</ComponentSpec>
```

Authoring an OCL Worker: generated code

- Use OpenCPI CDK tools to generate the worker skeleton files
- Body of generated: `fft1d_Worker.h`

```
// Properties
typedef struct {
  uint32_t direction;

  float scale;
} Fft1dProperties;
```

```
// Worker Context: "self"
typedef struct {
  __global Fft1dProperties*
     properties;
  OCLPort in, out;
} OCLWorkerFft1d;
```

- Body of editable: `ftt1d_skel.cl`

```
OCLResult fft1d_run ( __local OCLWorkerFft1d * self,
                      OCLBoolean timedOut) {
  // put real work here
  return OCL_ADVANCE; // input consumed, output produced
}
```

OpenCL FFT worker source code

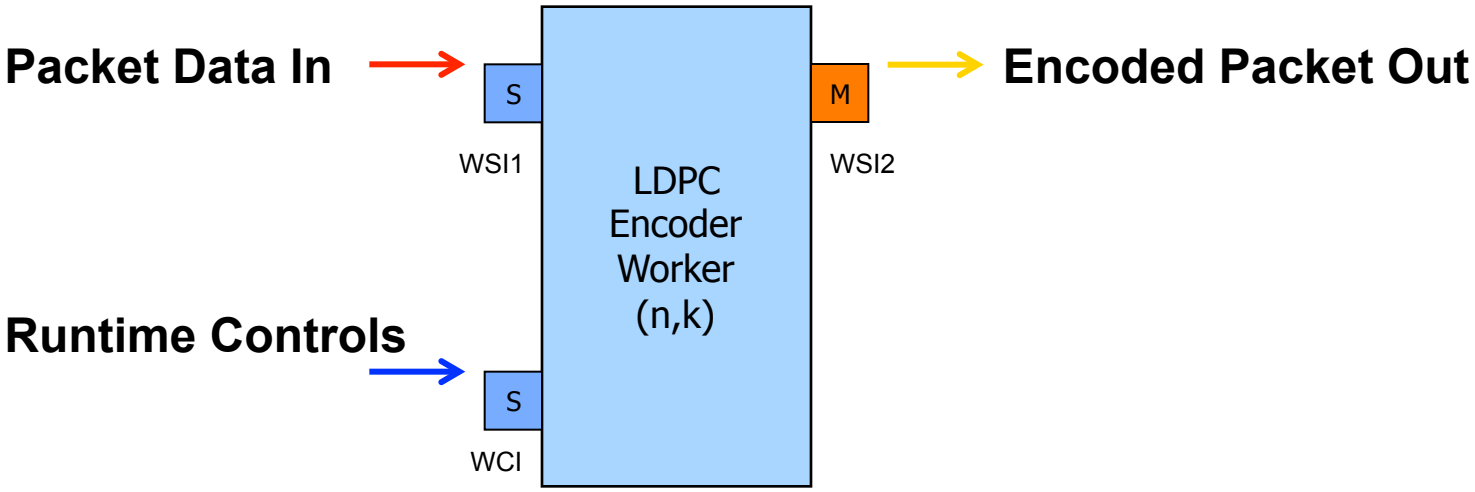
- Using an existing OpenCL FFT, this is the whole "worker"

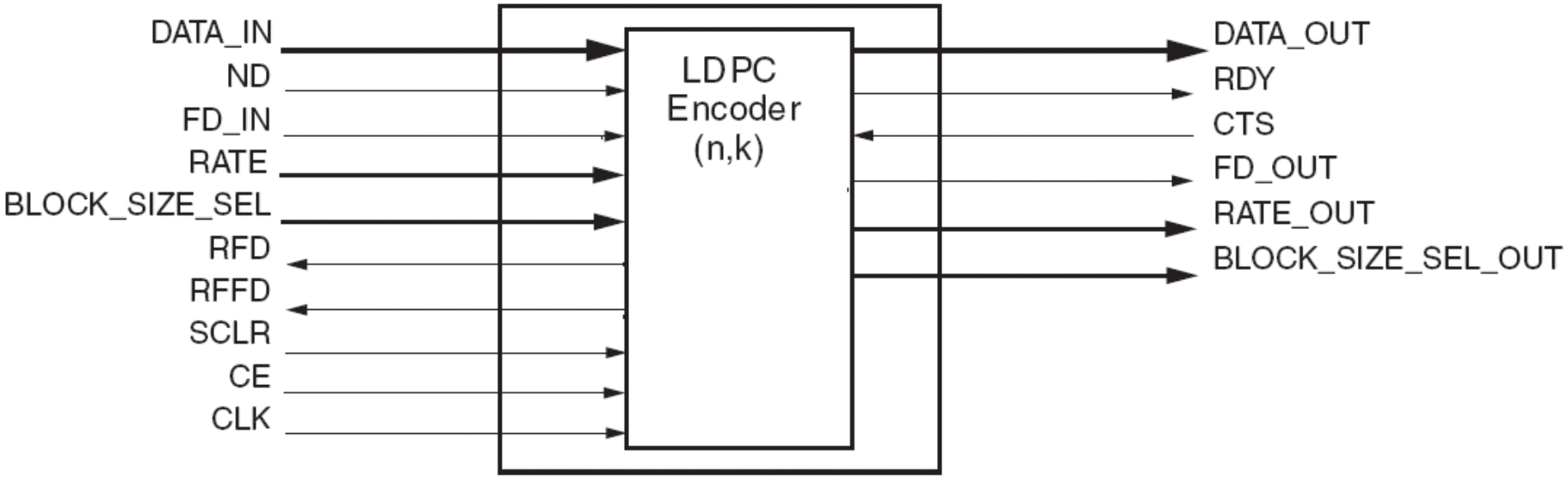
```
OCLResult fft1d_run(__local OCLWorkerFft * self,
                    OCLBoolean timedOut )
{
  fft ( self->in.current.data,
        self->out.current.data,
        self->in.length / sizeof ( float ),
        self->properties->scale,
        self->properties->direction );

  self->out.length = self->in.length;
  return OCL_ADVANCE;
}
```

LDPC Example: creating an HDL worker that wraps

- Take off-the-shelf 802.16e LDPC core from Xilinx website
  - eval license
- Select OCP-based WIPs (OCP profiles)
  - To properly export inner proprietary/ad-hoc interfaces
  - Control plane for control and configuration
    - WCI profile interface, with defined configuration properties
  - Data plane
    - WSI profile interfaces
- Adapt/stitch/wrap COTS core signals to WIP interfaces.
- Measure resource impact of standardizing interfaces

Top Level Existing Signal Definition





Structural VHDL from Core Gen

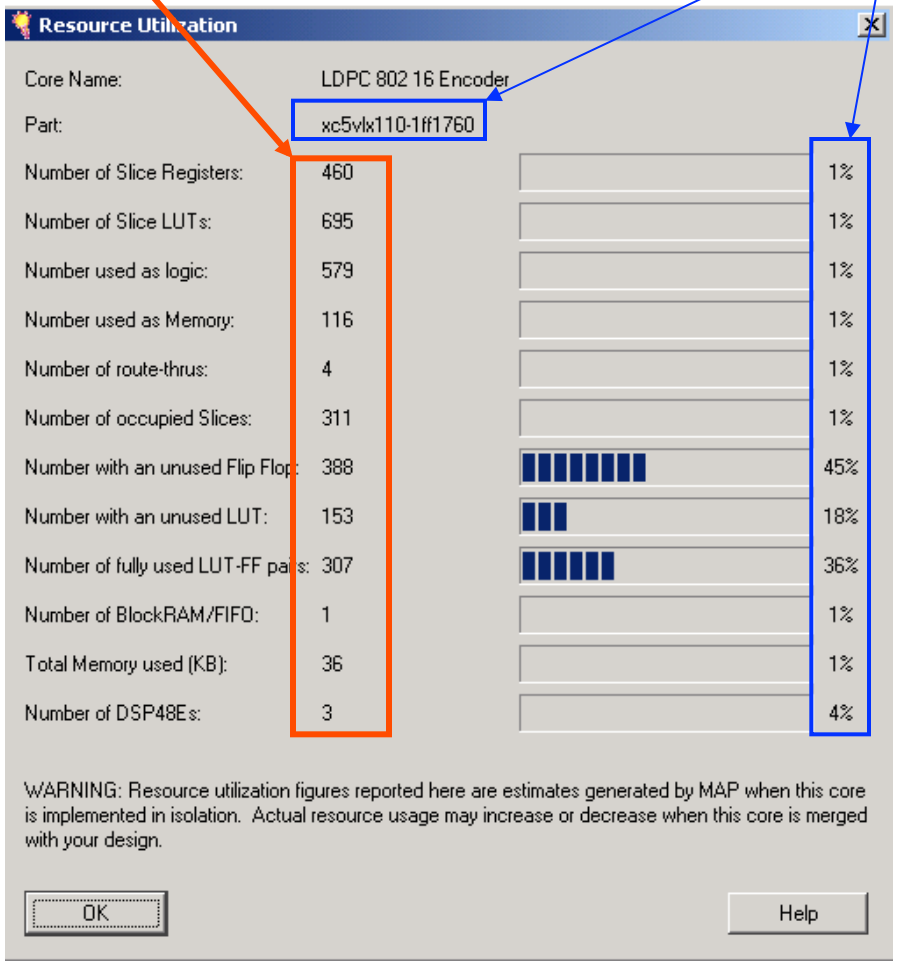
-
```
--------------------------------------------------------------------------------
-- Copyright (c) 1995-2007 Xilinx, Inc.  All rights reserved.
--------------------------------------------------------------------------------
--
--
--                  Vendor: Xilinx
--                  Version: J.33
--                  Application: netgen
--                  Filename: ldpc_802_16_enc_v1_0.vhd
--                  Timestamp: Tue Jun 26 10:53:37 2007
--
--
--
-- Command : -intstyle ise -w -sim -ofmt vhdl C:\projects\cpi_coregen\cpi_coren\tm
-- Device  : 5vlx110ff1760-1
-- Input file : C:/projects/cpi_coregen/cpi_coren/tmp/_cg/ldpc_802_16_enc_v1_0.ngc
-- Output file   : C:/projects/cpi_coregen/cpi_coren/tmp/_cg/ldpc_802_16_enc_v1_0.
-- # of Entities : 1
-- Design Name   : ldpc_802_16_enc_v1_0
-- Xilinx  : C:\Xilinx91i
--
-- Purpose:
--     This VHDL netlist is a verification model and uses simulation
--     primitives which may not represent the true implementation of the
--     device, however the netlist is functionally correct and should not
--     be modified. This file cannot be synthesized and should only be used
--     with supported simulation tools.
--
-- Reference:
--     Development System Reference Guide, Chapter 23
--     Synthesis and Simulation Design Guide, Chapter 6
--
--------------------------------------------------------------------------------


-- synopsys translate_off
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
use UNISIM.VPKG.ALL;

entity ldpc_802_16_enc_v1_0 is
  port (
    sclr : in STD_LOGIC := 'X';
    data_in : in STD_LOGIC := 'X';
    fd_in : in STD_LOGIC := 'X';
    rfd : out STD_LOGIC;
    rdy : out STD_LOGIC;
    fd_out : out STD_LOGIC;
    nd : in STD_LOGIC := 'X';
    clk : in STD_LOGIC := 'X';
    data_out : out STD_LOGIC;
    rffd : out STD_LOGIC;
    cts : in STD_LOGIC := 'X';
    block_size_sel_out : out STD_LOGIC_VECTOR ( 4 downto 0 );
    rate_out : out STD_LOGIC_VECTOR ( 2 downto 0 );
    block_size_sel : in STD_LOGIC_VECTOR ( 4 downto 0 );
    rate : in STD_LOGIC_VECTOR ( 2 downto 0 )
  );
end ldpc_802_16_enc_v1_0;
```

Resource Report of raw, "unwrapped" IP Core

These values pertain to one, individual IP core

These values express the percentage used by one core within the 5VLX110T device

- **We will revisit these values after the wrapper is complete**



Construction of ldpc_worker

- Entity generated based on WIP choices (i.e. Options for OpenCPI OCP profiles)

```
entity ldpc_worker is
  port (
    -- Component-Wide Signals
    clk                 : in     std_logic;
    rstn                : in     std_logic;
    -- Interface WCI
    wci_MAddr           : in     std_logic_vector(5 downto 0);
    wci_MAddrSpace      : in     std_logic;
    wci_MCmd            : in     ocp_CmdT;
    wci_MData           : in     ocp_Data32T;
    wci_MFlag           : in     std_logic;
    wci_SData           : out    ocp_Data32T;
    wci_SFlag           : out    std_logic;
    wci_SResp           : out    ocp_RespT;
    wci_SThreadBusy     : out    std_logic;
    -- Interface WSI1
     -- DataWidth : 1
     -- NumberOfOpcodes : 1
     -- MessageLengthIsExplicit : TRUE
     -- PreciseBurstsOnly : TRUE
     -- Producer : FALSE
    wsi1_MBurstLength : in     std_logic_vector(1 downto 0);
    wsi1_MCmd           : in     ocp_CmdT;
    wsi1_MData          : in     std_logic;
    wsi1_MDataLast      : in     std_logic;
    wsi1_MDataValid     : in     std_logic;
    wsi1_SThreadBusy    : out    std_logic;
    -- Interface WSI2
     -- DataWidth : 1
     -- NumberOfOpcodes : 1
     -- MessageLengthIsExplicit : TRUE
     -- PreciseBurstsOnly : TRUE
     -- Producer : TRUE
    wsi2_MBurstLength : out    std_logic_vector(1 downto 0);
    wsi2_MCmd           : out    ocp_CmdT;
    wsi2_MData          : out    std_logic;
    wsi2_MDataLast      : out    std_logic;
    wsi2_MDataValid     : out    std_logic;
    wsi2_SThreadBusy    : in     std_logic
  );
end ldpc_worker;
```

Instance LDPC core and adapt to WIP OCP

```
-- Instantiate and connect the LDPC core...
ldpc_core : entity work.ldpc_802_16_enc_v1_0
  port map    (
  sclr               => rst,              -- active high reset
  data_in            => wsi1_MData_d,     -- packet data to be encoded
  fd_in              => wsi1_MDataFirst,  -- first data in input packet pulse
  rfd                => ldpc_rfd,         -- ready for data
  rdy                => wsi2_MDataValid,  -- qualfies active data on data_out
  fd_out             => ldpc_fd_out,      -- first data out
  nd                 => wsi1_MDataValid,  -- new data, qualifies data on data_in
  clk                => clk,              -- clock
  data_out           => wsi2_MData,       -- output data: unaltered systematic bits followed by code bits
  rffd               => foo,              -- ready for first data
  cts                => ldpc_cts,         -- clear to send; allows downstream consumer to apply backpressure
  block_size_sel_out => open,             -- block size pass-through
  rate_out           => open,             -- rate pass-through
  block_size_sel     => ldpc_block_size_sel,  -- ldpc block size
  rate               => ldpc_rate         -- ldpc rate
);


-- Combi Assignments...
rst                <= not rst_n;              -- invert reset
wsi1_SThreadBusy   <= not ldpc_rfd;           -- invert ldpc "ready for data" to make upstream wsi1_SThreadBusy
ldpc_cts           <= not wsi2_SThreadBusy;   -- invert downstream wsi2_SThreadBusy to make ldpc clear to send
ldpc_block_size_sel <= conf_props(0)(4 downto 0) -- 0x0: Block Size bits [4:0]
ldpc_rate          <= conf_props(1)(2 downto 0) -- 0x4: Rate       bits [2:0]
```

- This slide contains the essential code needed to instantiate and stitch the LDPC core as an OpenCPI LDPC worker
- Some housekeeping details shown elsewhere

Area

OpenCPI "wrapper":

Partition "/ldpc_worker"			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	17		
Number of Slice LUTs	5		
Number used as logic	5		
Slice Logic Distribution			
Number of occupied Slices	10		
Number of LUT Flip Flop pairs used	22		
Number with an unused Flip Flop	5		
Number with an unused LUT	17		
Number of fully used LUT-FF pairs	0		

LDPC Core:

Partition "/ldpc_worker/ldpc_core"			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	460		
Number of Slice LUTs	810		
Number used as logic	694		
Number used as Memory	116		
Slice Logic Distribution			
Number of occupied Slices	262		
Number of LUT Flip Flop pairs used	806		
Number with an unused Flip Flop	346		
Number with an unused LUT	111		
Number of fully used LUT-FF pairs	349		
Number of Block RAM/FIFOs	1		
Number of DSP48Es	3		

Weight of Implementation (2)

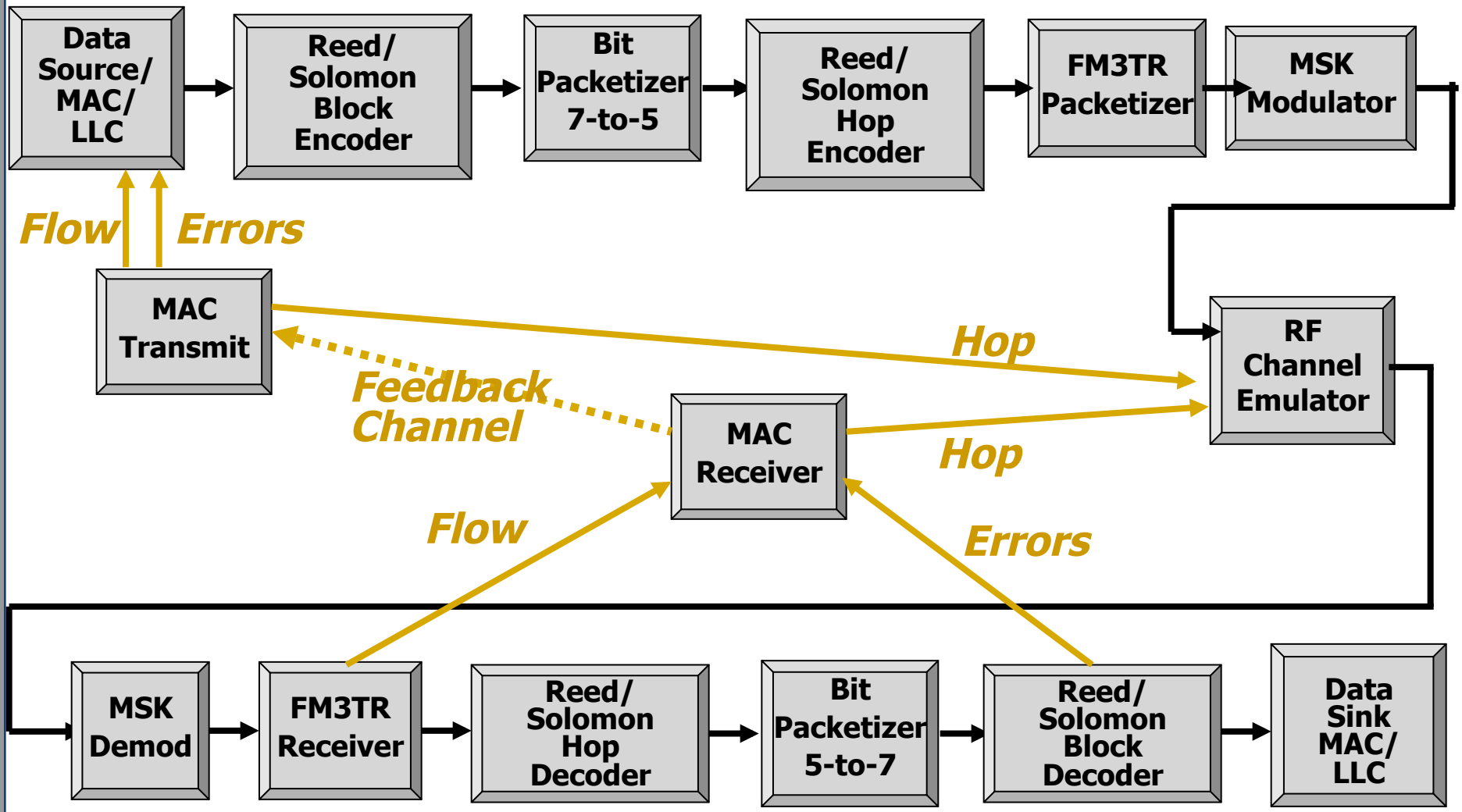
- Area Cost of LDPC Wrapper:
  - 5 LUTs, 17 FFs
- Latency Cost of implementation:
  - Zero Cycles
- F_{MAX} Degradation:
  - None observed (200 MHz design had slack)
- Code Complexity:
  - Wrapper took 192 lines of code
    - Mostly “wire”; only 5 LUTs and 17 FFs instanced

Reference Application: FM3TR (common SDR example)

- Proposed Coalition Tactical Radio Waveform
  - Open specification
  - Parameters: band coverage, hop rate, modulation rate, etc.
  - 30-400MHz range, channel spacing 25kHz, frequency accuracy 10^{-6}
  - Modulation: CPFSK, index 0.5, rate 25kbps
  - Hop: 250-2000/s, dwell 40-80 bits, tune 5-10 bits, 3 hopping sub-bands, set=128
  - Voice mode coding (CVSD 16kbits), Data messages up to 48K bytes.
- C++/SCA/CORBA version of all components
  - submitted as reference SCA WF to SDRForum
- C/RCC/DSP version of all components
  - created under DARPA PCA for Multi-RISC
- Voice mode waveform as well as data mode waveform
- No hopping/tuning in digital WF.

Reference Application: FM3TR Data

- Signal chain, waveform components, control path are:



Agenda

- OpenCPI Introduction, Goals, Background, History 9:00
- OpenCPI Briefing 9:15
- Recent Developments 10:00
- Break 10:30
- Roadmap Ideas, Discussion 10:45
- Workflow, code examples 11:15
- **Q&A, Wrap-up** **11:45**

Open-Source Component Portability Infrastructure

Is the:

1. **Open Source Community Solution for:**
2. **Component-Based Development (CBD) in**
3. **Heterogeneous Embedded Systems**