

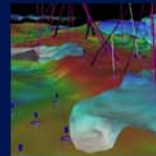
# Component Portability for Specialized Hardware in Software Defined Radios

**8/3/06**

- **SCA Overview**
- **SHS – Brief History**
- **CP289 Concepts and Goals**
- **Specifics of Different Models:**
  - SHP: Specialized Hardware Processor components
  - RCC: Resource-Constrained C components
  - RPL: RTL-Programmable Logic components
- **Summary and Status**

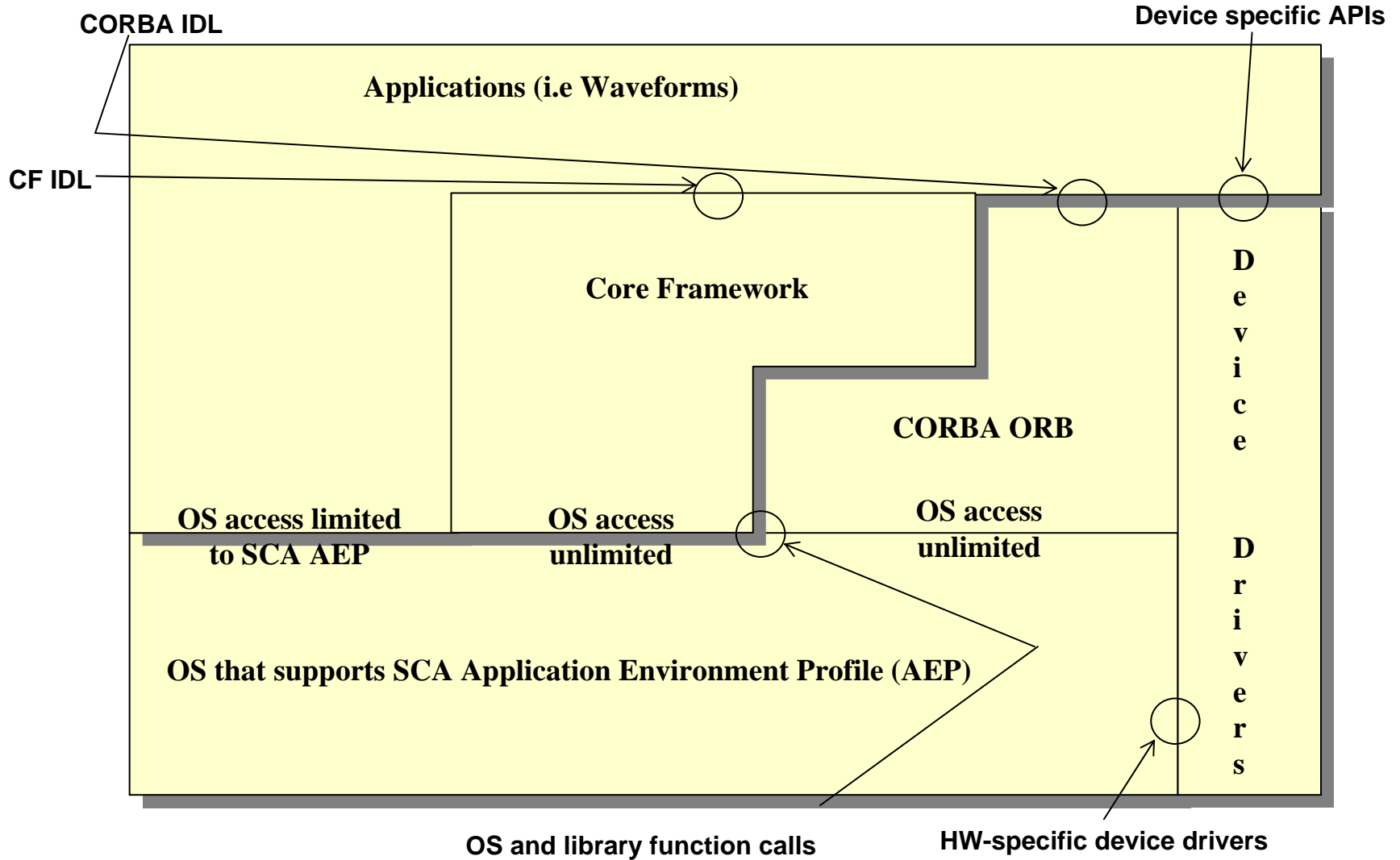
Computer Systems, Inc.  
**MERCURY**

*Challenges Drive Innovation™*



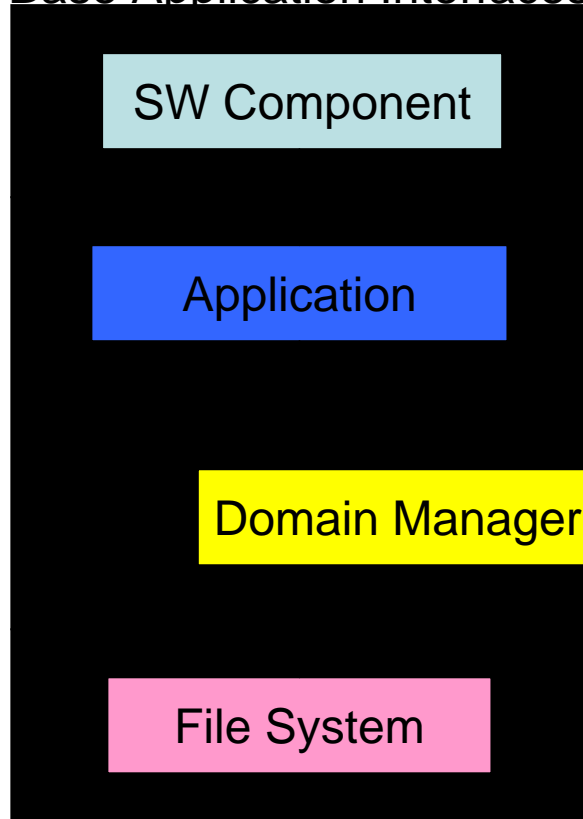
## Software Communications Architecture (SCA)

- **Interoperability of different SDR systems**
- **Portability of applications software between different SDR implementations**
- **Upgradeability in terms of easy technology insertion**
- **Reduced development time of new waveforms through the ability to reuse design modules**
- **Reduced system acquisition, operation and supportability costs**

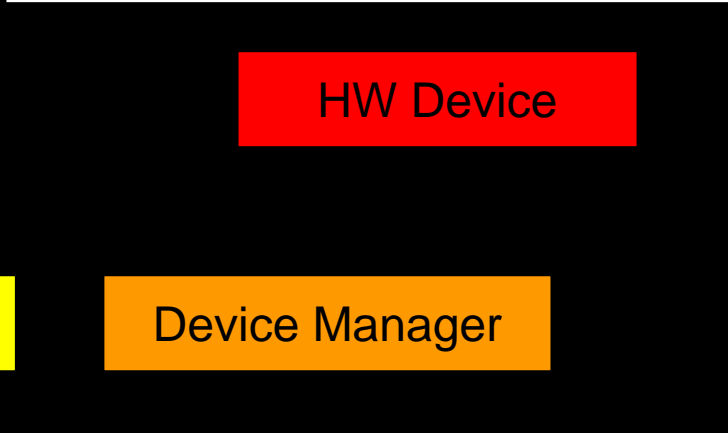


# Core Framework Interface Categories

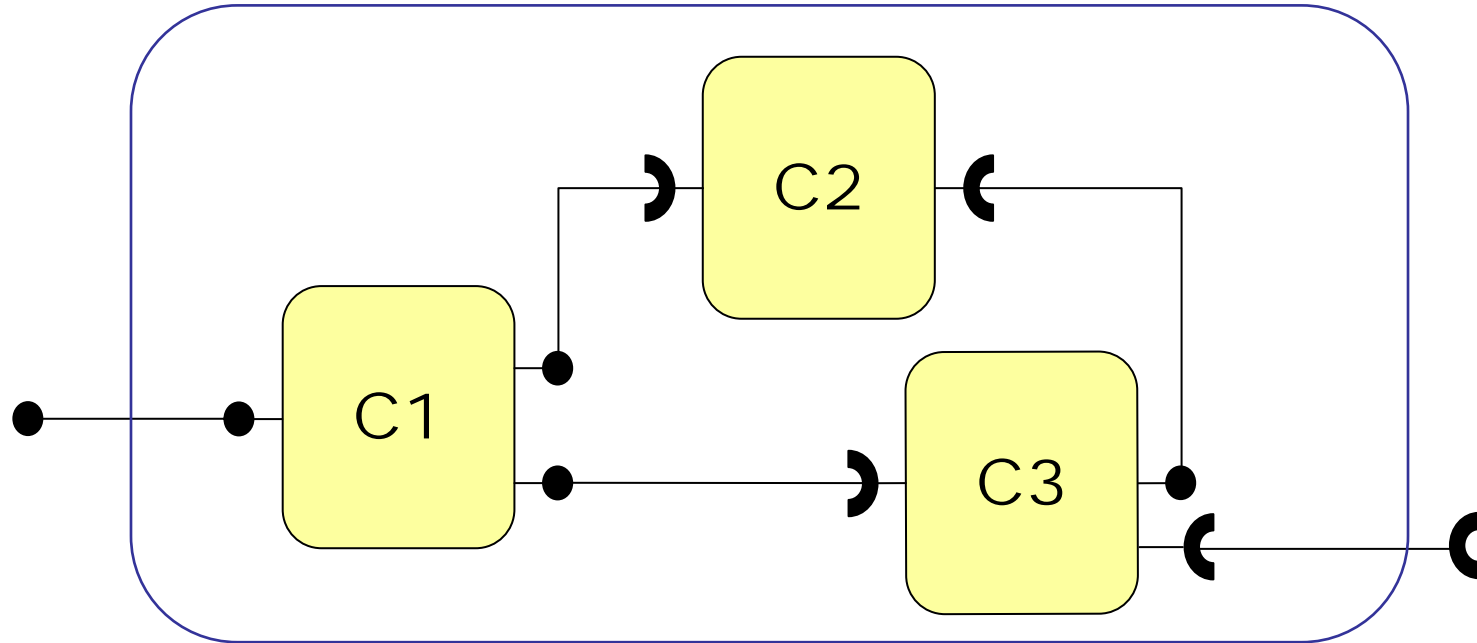
## Base Application Interfaces



## Framework Control Interfaces

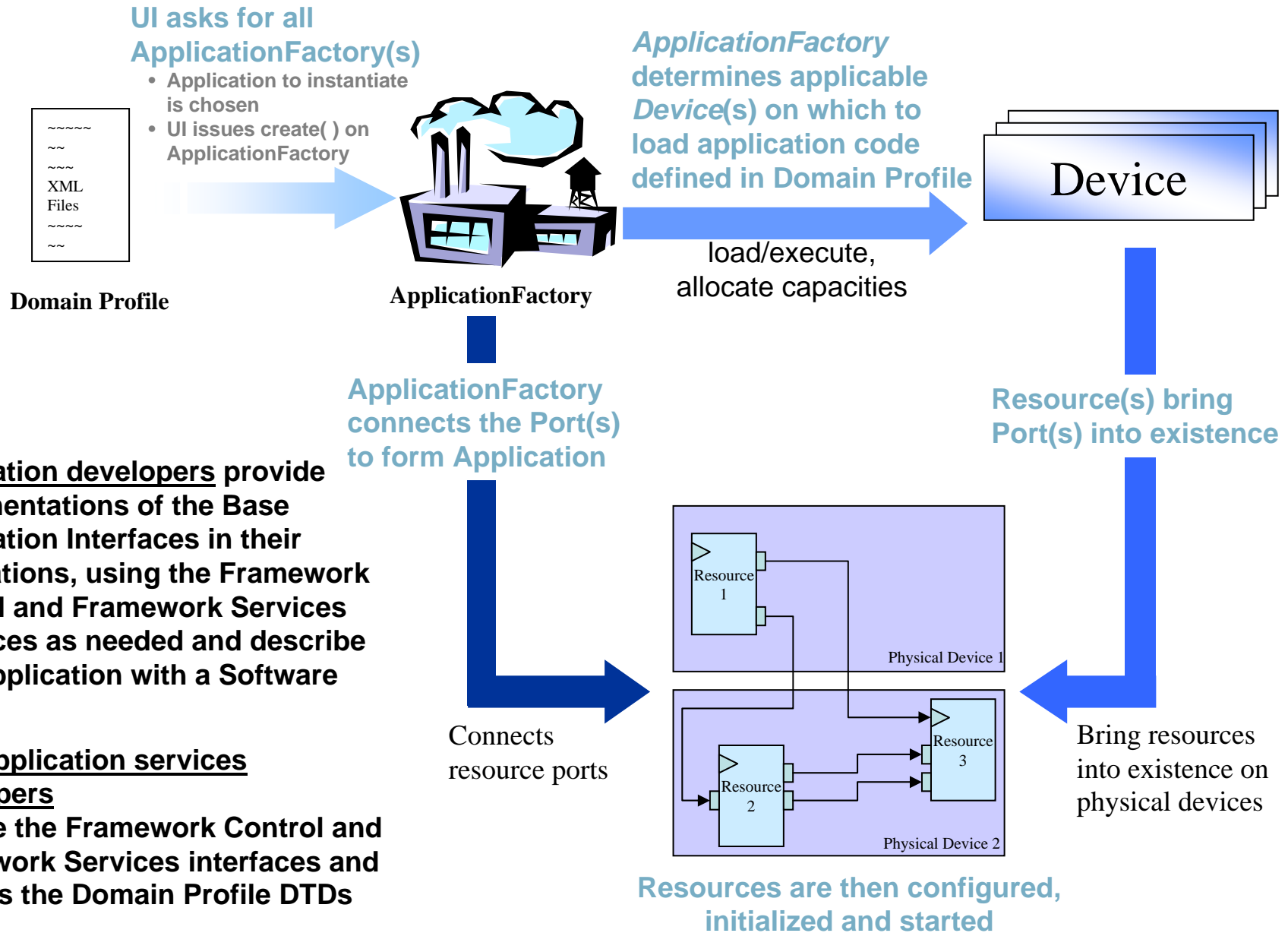


## Framework Services Interfaces



- Waveforms are represented by *SCA Applications*
- An application is an assembly of software components connected to each other through their *Ports*
- These software components are called *Resources*
- Each Resources is a functional block and is treated as a **black box**

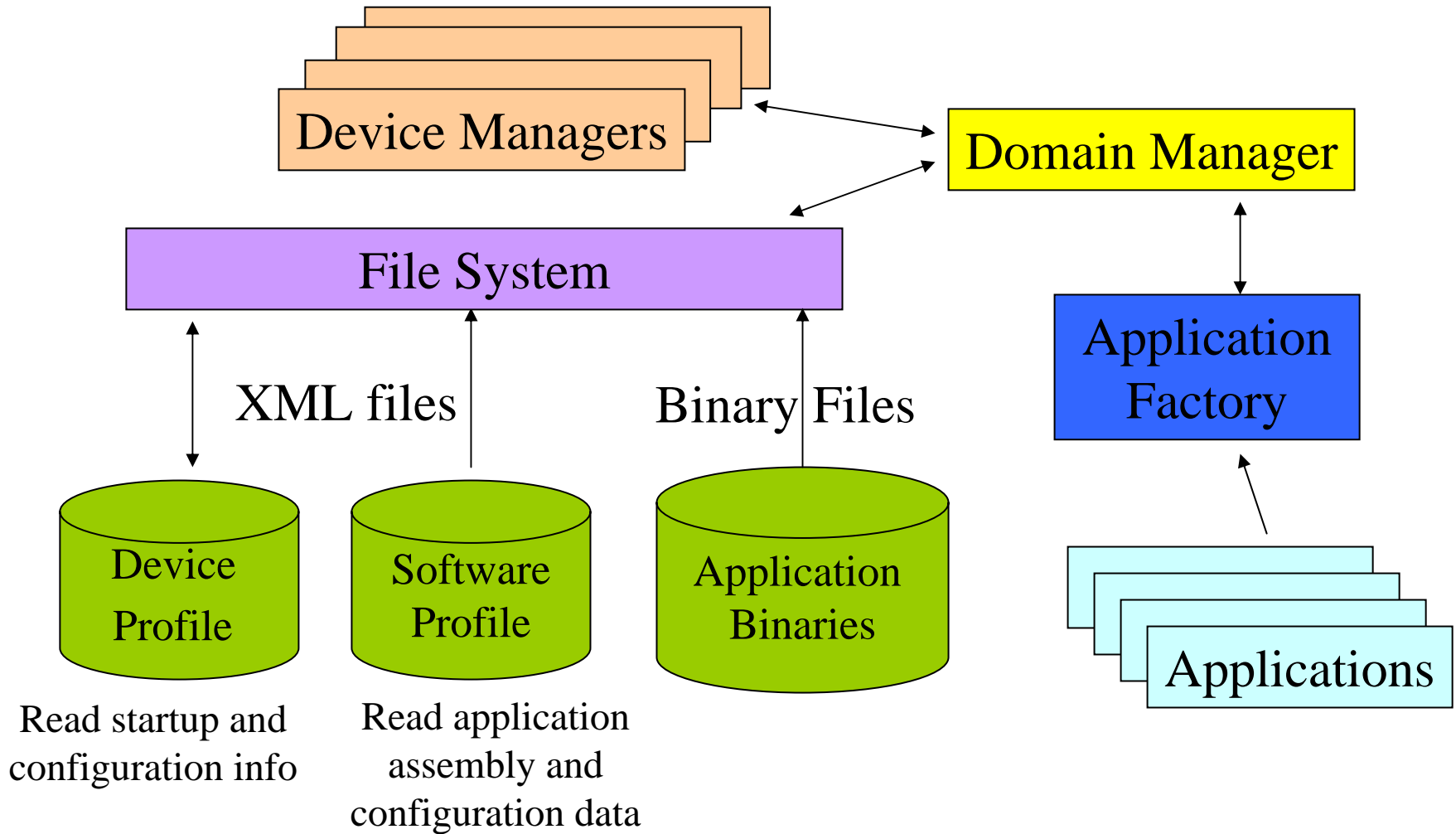
# Creating Applications

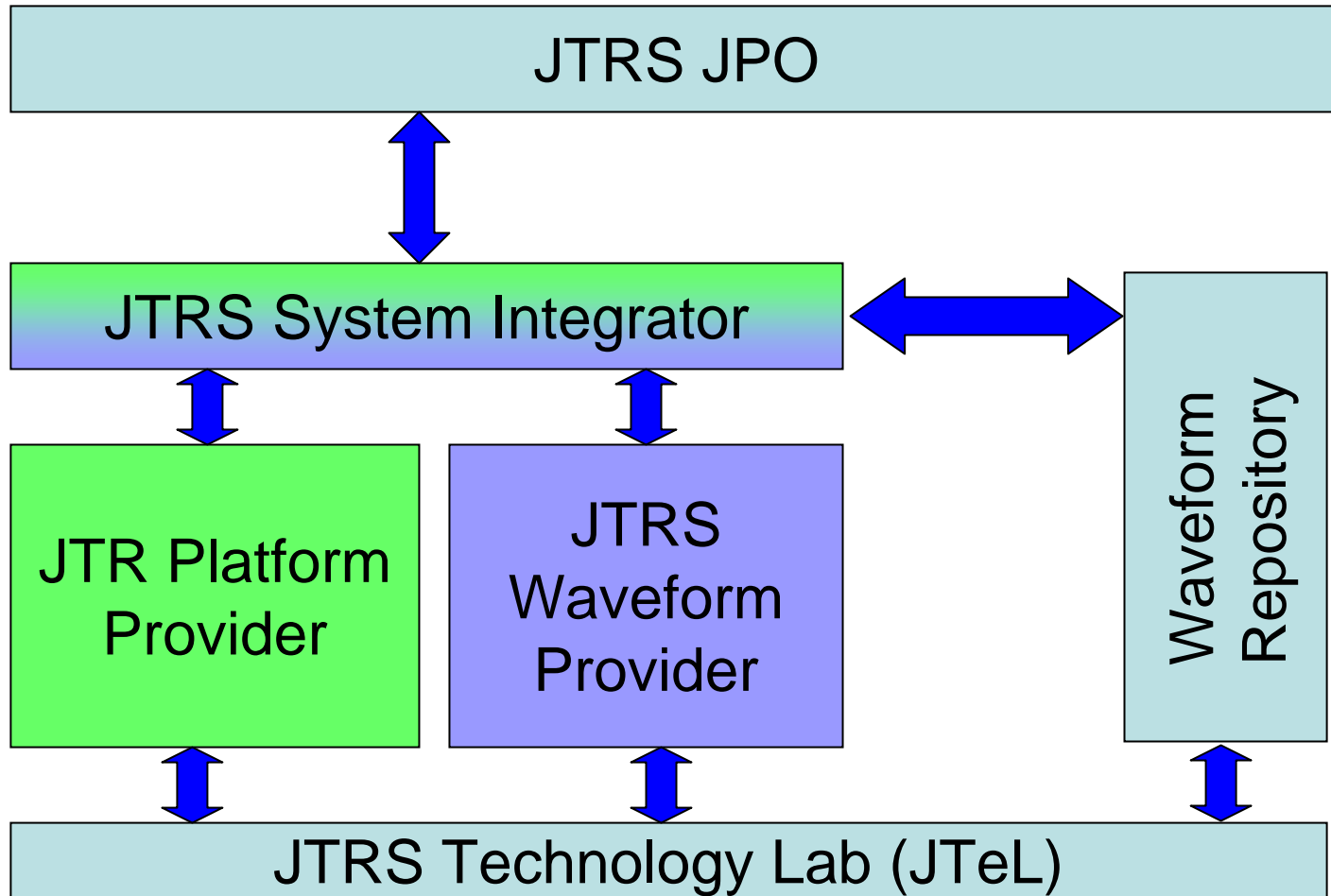


**Application developers provide implementations of the Base Application Interfaces in their applications, using the Framework Control and Framework Services Interfaces as needed and describe their application with a Software Profile**

**Core application services developers provide the Framework Control and Framework Services interfaces and process the Domain Profile DTDs**

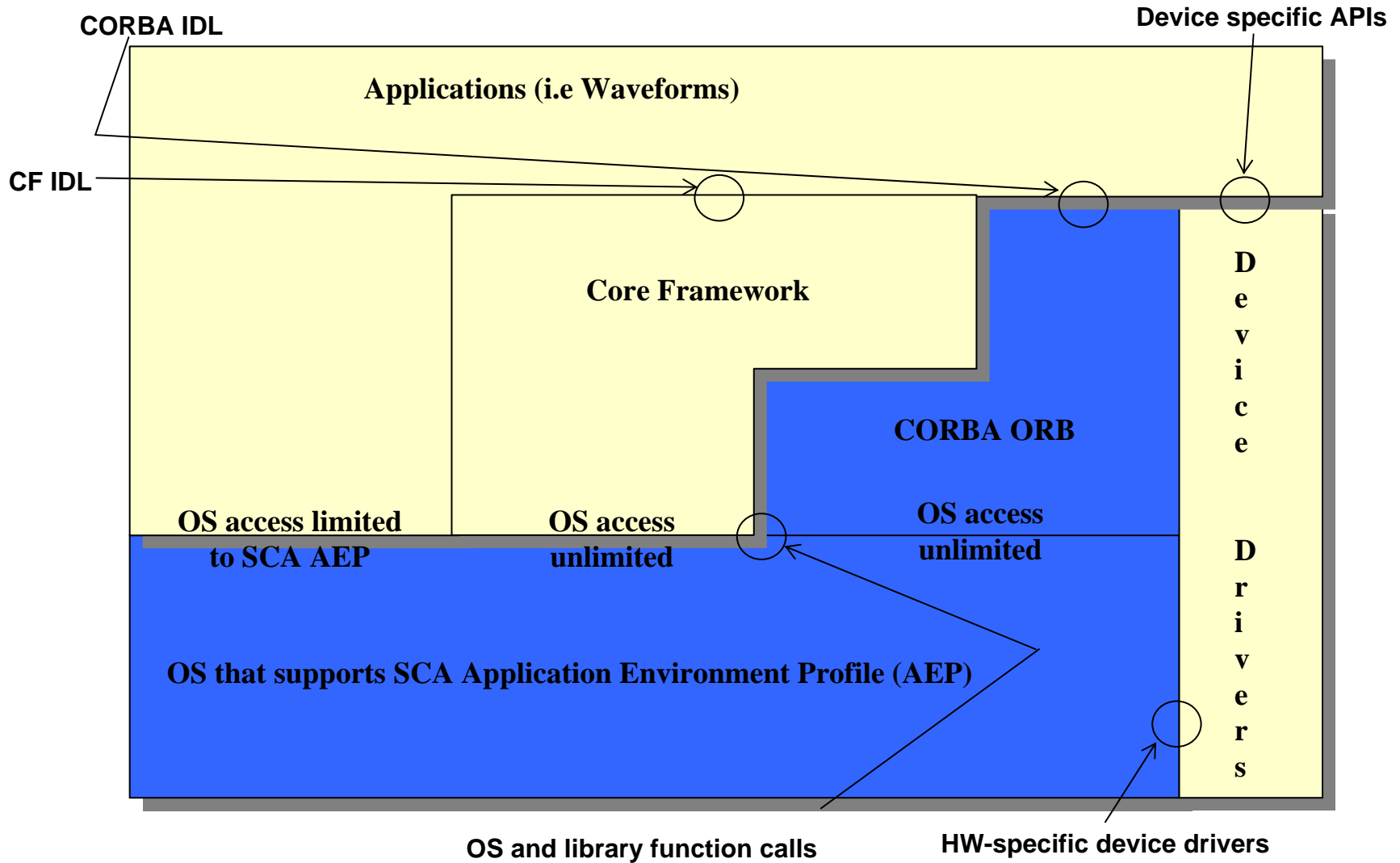
# SCA Core Framework Entities





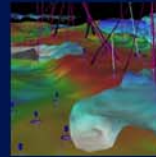
# Vendors – Example Deliverables

	Core Framework Provider	Platform Provider	Waveform Provider
<b>SCA Components</b>	<ul style="list-style-type: none"> <li>• Domain Manager</li> <li>• Application Factory</li> <li>• Application</li> <li>• File Manager</li> </ul>	<ul style="list-style-type: none"> <li>• Device Manager</li> <li>• Device</li> <li>• Loadable Device</li> <li>• Executable Device</li> <li>• Aggregate Device</li> <li>• Platform-Specific Devices</li> <li>• File System</li> </ul>	<ul style="list-style-type: none"> <li>• Life Cycle</li> <li>• Port Supplier</li> <li>• Testable Object</li> <li>• Property Set</li> <li>• Port</li> <li>• Resource</li> <li>• Resource Factory</li> </ul>
<b>Support Tools</b>	<ul style="list-style-type: none"> <li>• CF Monitoring Tools</li> </ul>	<ul style="list-style-type: none"> <li>• User Interfaces (HCI)</li> </ul>	<ul style="list-style-type: none"> <li>• Simulation Models</li> </ul>



Computer Systems, Inc.  
**MERCURY**

*Challenges Drive Innovation™*



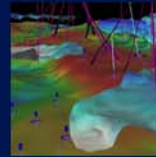
## Specialized Hardware Supplement *A Brief History*

- **JTRS JPO initiated effort for portability of “Specialized Hardware” parts of SDR waveforms (DSP, FPGA, ASIC etc.) in April 2004**
  - SDR designs include DSPs and FPGAs: the SCA 2.x did not apply
    - RTOS, CORBA ORB
  - High-band/SATCOM/Datalink designs dominated by ASICs/FPGAs
  - Need to extend SCA’s model and portability to such hardware
  - Initial workshop April 2004, SCA 3.0 issued a few months later
- **SCA 3.0 had “drafts” for non-GPP portability solutions, called Specialized Hardware Supplement, as well as a few minor SCA updates**

- **When SCA 3.0 was issued, JPO also funded maturing the “Component Portability Specification” (CPS) proposal for SCA 3.1**
- **Second JPO workshop held to create proposals for SCA 3.1, January 2005**
- **CPS, after extensive discussions with industry and government participants, improved and submitted officially as SCA CP289 in March 05**
- **Another proposal, mostly a transport abstraction, also submitted as CP237**
- **JPEO suspended work/funding on SCA 3.1: “A Strategic Pause” as of June 2005**

Computer Systems, Inc.  
**MERCURY**

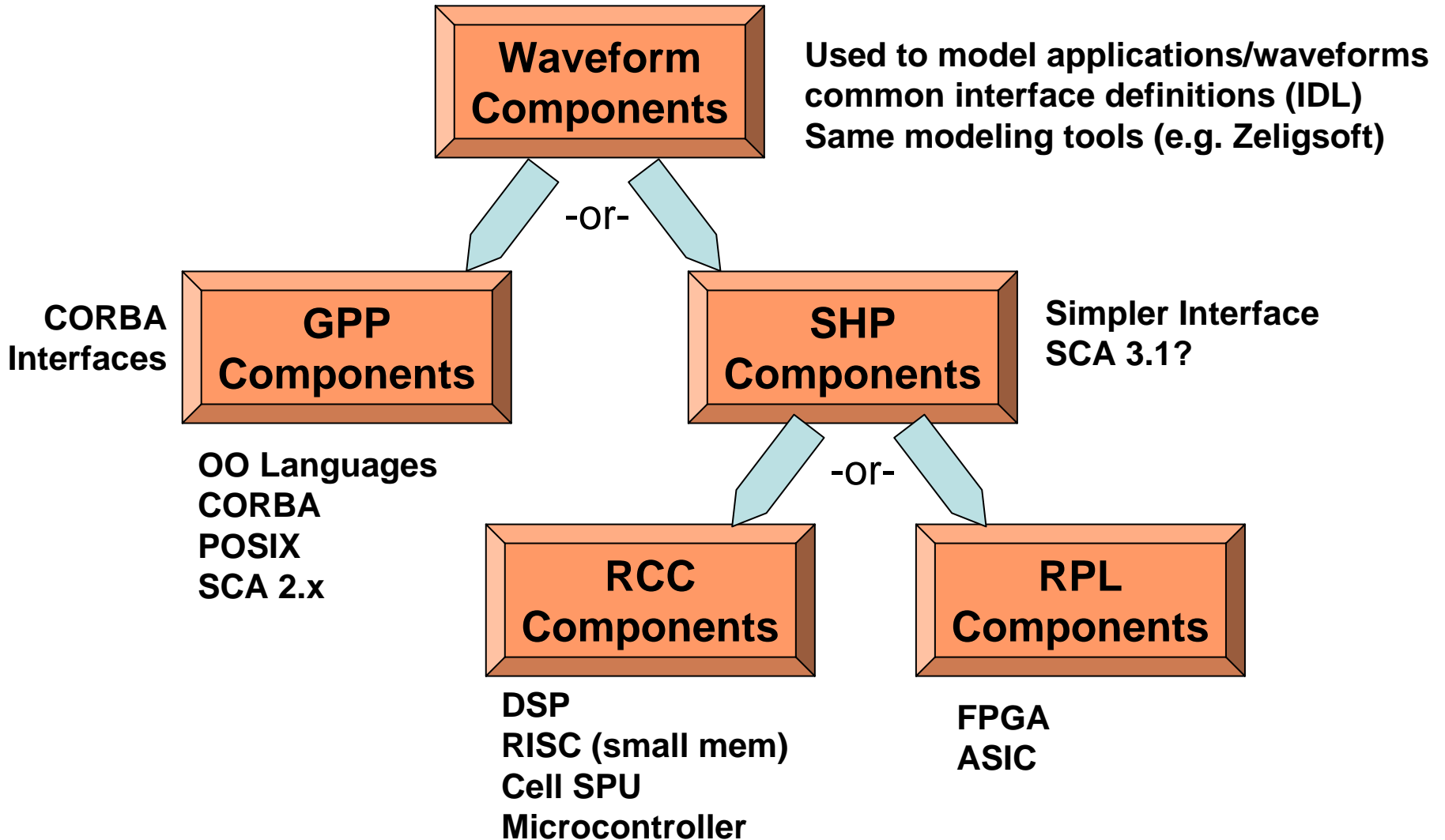
*Challenges Drive Innovation™*

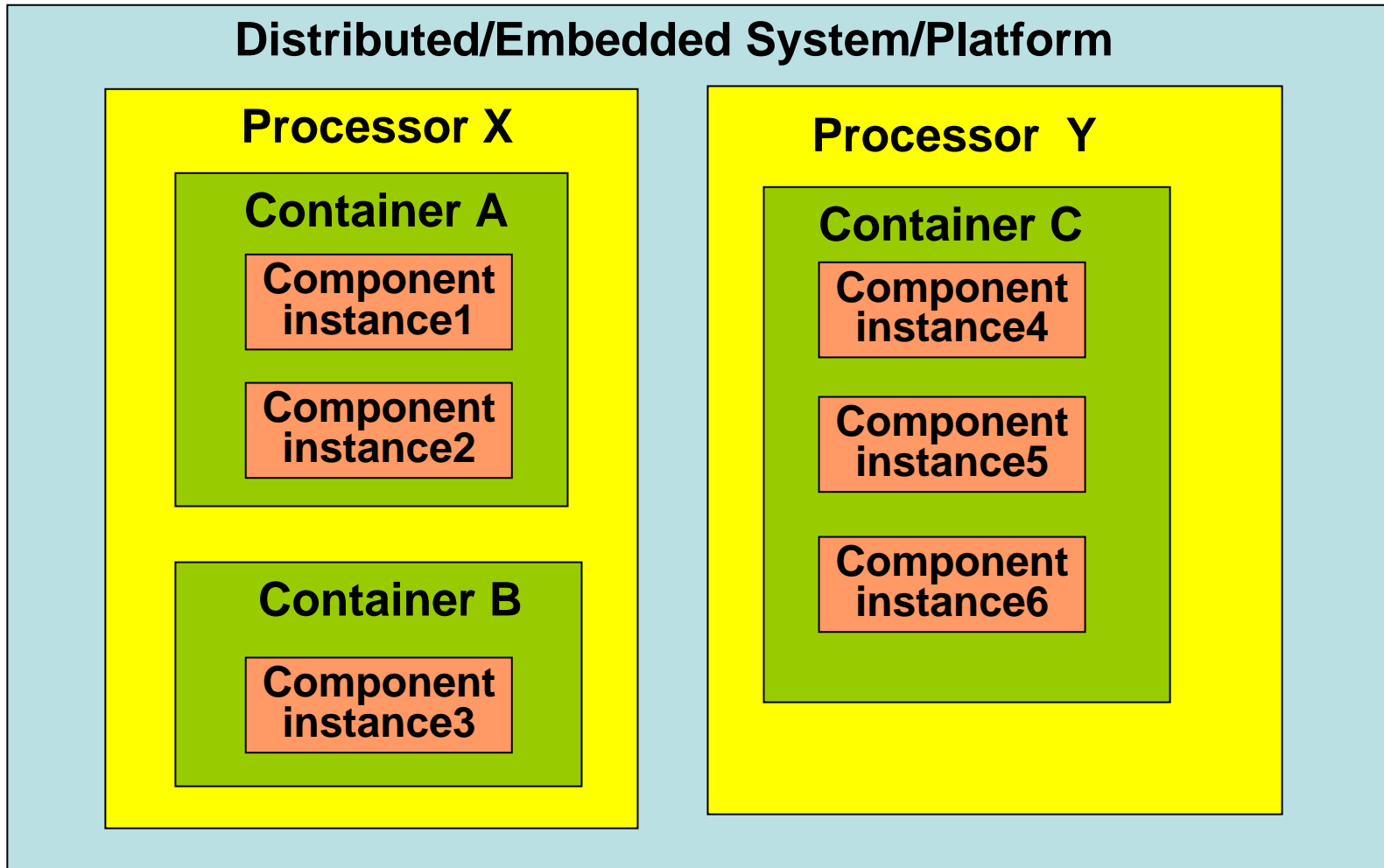


# Component Portability Specification *Change Proposal (CP) 289*

- **Component (as in SCA):**
  - Unit of deployable, updatable, packaged functionality
  - Independently defined unit of functionality of an application
  - Combines code and descriptive metadata
  - Multiple implementations can exist for different processors
- **Application/waveform (as in SCA):**
  - One or more components deployed as a unit to perform interesting work for the user/client
  - A configured and interconnected set of components
- **Container:**
  - The *immediate* runtime environment where a component executes
  - The provider of any *local* runtime services or APIs to components
  - The *local* invoker/controller/manager of the component
- **Class**
  - A particular language/API model to which components are written

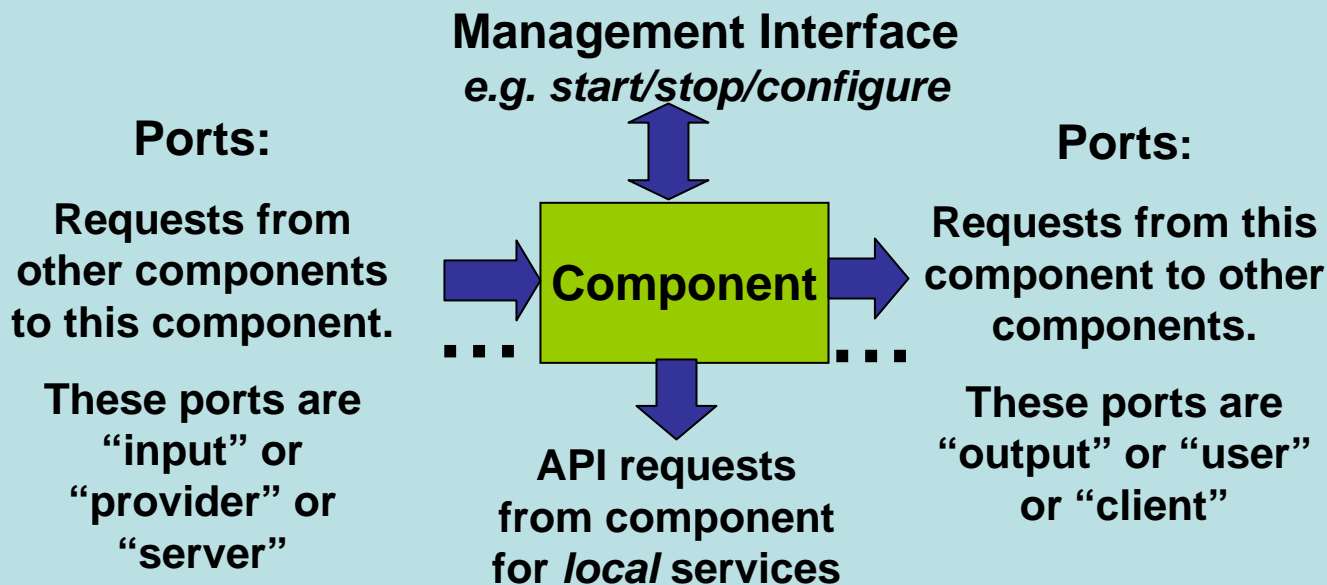
- **Different processor types need different languages and APIs**
  - No “one language/API fits all,” especially for performance
- **GPP Class: General-Purpose Processor environments (SCA 2.x)**
  - CORBA enabled environments, with C++/Java/Ada etc.
  - Existing component specifications are suitable, require CORBA
- **RCC Class: Resource-Constrained C language environments**
  - When C is available, and CORBA is not available/suitable/acceptable
  - E.g. DSPs, Microcontrollers, Cell SPUs, RISC cores w/limited RAM
- **RPL Class: RTL-Programmable-Logic environments**
  - When RTL language is available (VHDL/Verilog)
  - When C is not available/suitable/acceptable
  - E.g. FPGAs and ASICs
- **Components for all classes (languages/APIs) fit a common model for design and tools**





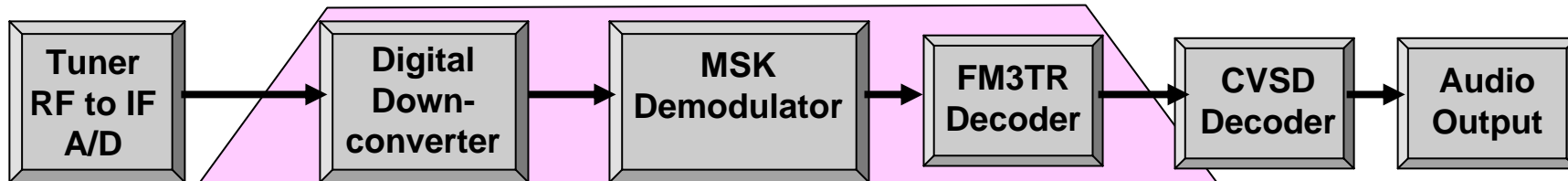
- **Component functionality, and applications are modeled/specified/defined *independent of class***
- **Components written for different classes *will interoperate***
- **A component implementation (source code) is written for a class (GPP/RCC/RPL)**
- **A container may support one or more classes**
  - Example: GPP container may support both RCC and GPP components
- **A processor may support one or more containers**
  - Example: GPP processor may support different process address spaces for security partitioning

## Executing inside container

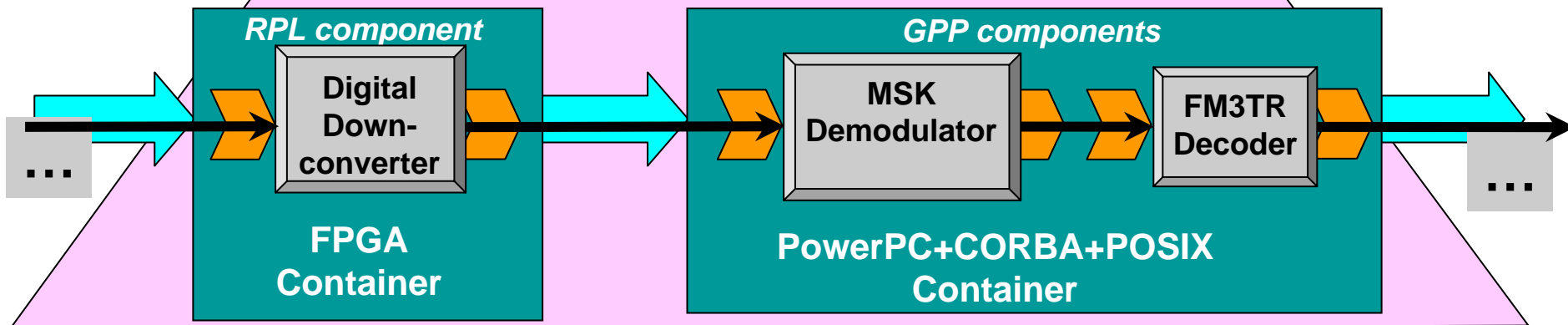


- Each class defines API for these interactions
- Interactions via ports can be:
  - Request/response RPC/RMI style
  - One way messaging
  - Streaming
- SCA 2.x defines these for GPP components, using CORBA

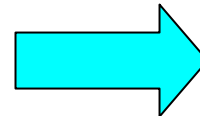
## Simple FM3TR Receiver



## Component implementations in Containers



Components talk to their containers. Important interface for portability of components. APIs used by component authors.



Containers talk to containers. Important interface for interoperability/plug&play of containers (e.g. boards). Protocols/networks/busses.



Communication between components, conveyed by their containers

- **General goal:**

- Integration of DSP, FPGA and similar technologies into *component software models/systems* for embedded/distributed systems
  - (aka SHP components: “Specialized Hardware Processors”)
  - SCA: the component model for software radio (from DoD/JTRS)
  - CCM (CORBA Component Model): a more general model for SW components (from OMG)

- **Specific goals:**

- **Portability** of SHP waveform components at source level
- **Replace-ability** across technologies within applications/waveforms
- **Resource efficiency and performance**
- **Separation of concerns** between platform provider and component/waveform author
- **Minimal impact** /changes required on the existing component models and tools

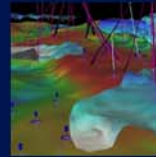
- **A component implementation can port to same class of container (“like for like”), recompiling source code**
  - RPL component written in VHDL ports between FPGA families or to an ASIC
  - Same portability goal/assertion as for GPP (SCA 2.x)
- **Use of platform/processor/container-specific features impedes portability**
- **Portable “reference implementations” can be tweaked to use special features (e.g. Viterbi accelerator on DSP), but original should be delivered as reference for further porting**

- **Enable changes in technology/processor class with no impact on the rest of the application (other components)**
  - Change a filter from FPGA to (new faster) DSP
  - Change a modem from DSP to (new faster) GPP
  - Increase data rate requiring switch to (new faster) FPGA
- **Enable simple addition of component *implementations* to existing components, without definition**
  - Both CCM & SCA support multiple implementations in a component package
  - Allow adding FPGA implementation to component with GPP implementation without impacting application
- **Implies opaque interoperability between all classes of component implementations**

- **Minimize “tax” for portability**
- **Minimize “tax” for interoperability**
- **Enable appropriately small footprint**
- **Enable full performance usage of inter-processor hardware interconnections**
  - Busses, networks, fabrics, NICs
- **Enable full performance for collocated component instances**
- **Enable statically pre-combinations of component implementations**
- **Enable zero copy operation**
  - To inter-processor interconnects
  - Between collocated components
  - Between input and output of a component

Computer Systems, Inc.  
**MERCURY**

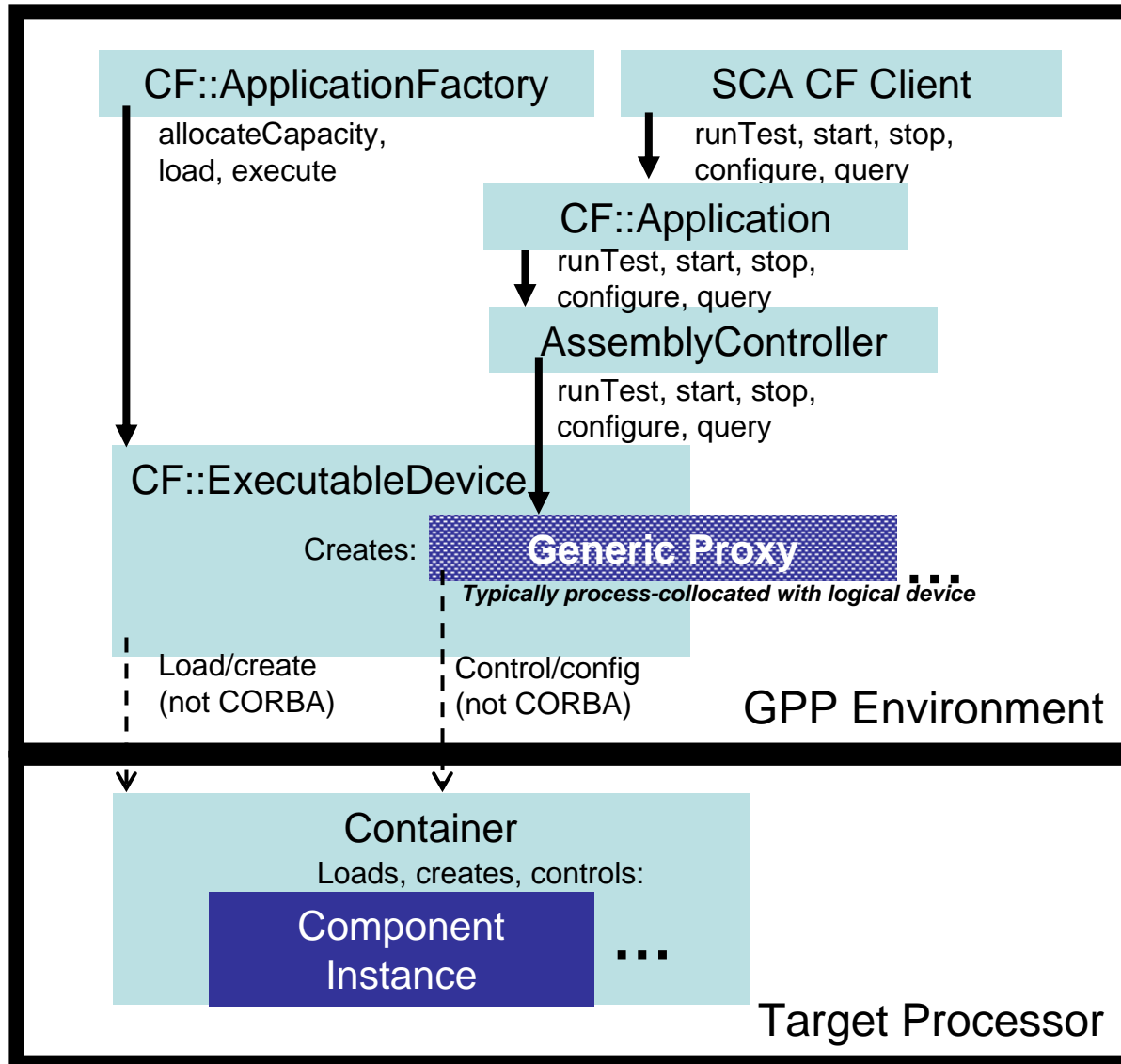
*Challenges Drive Innovation™*



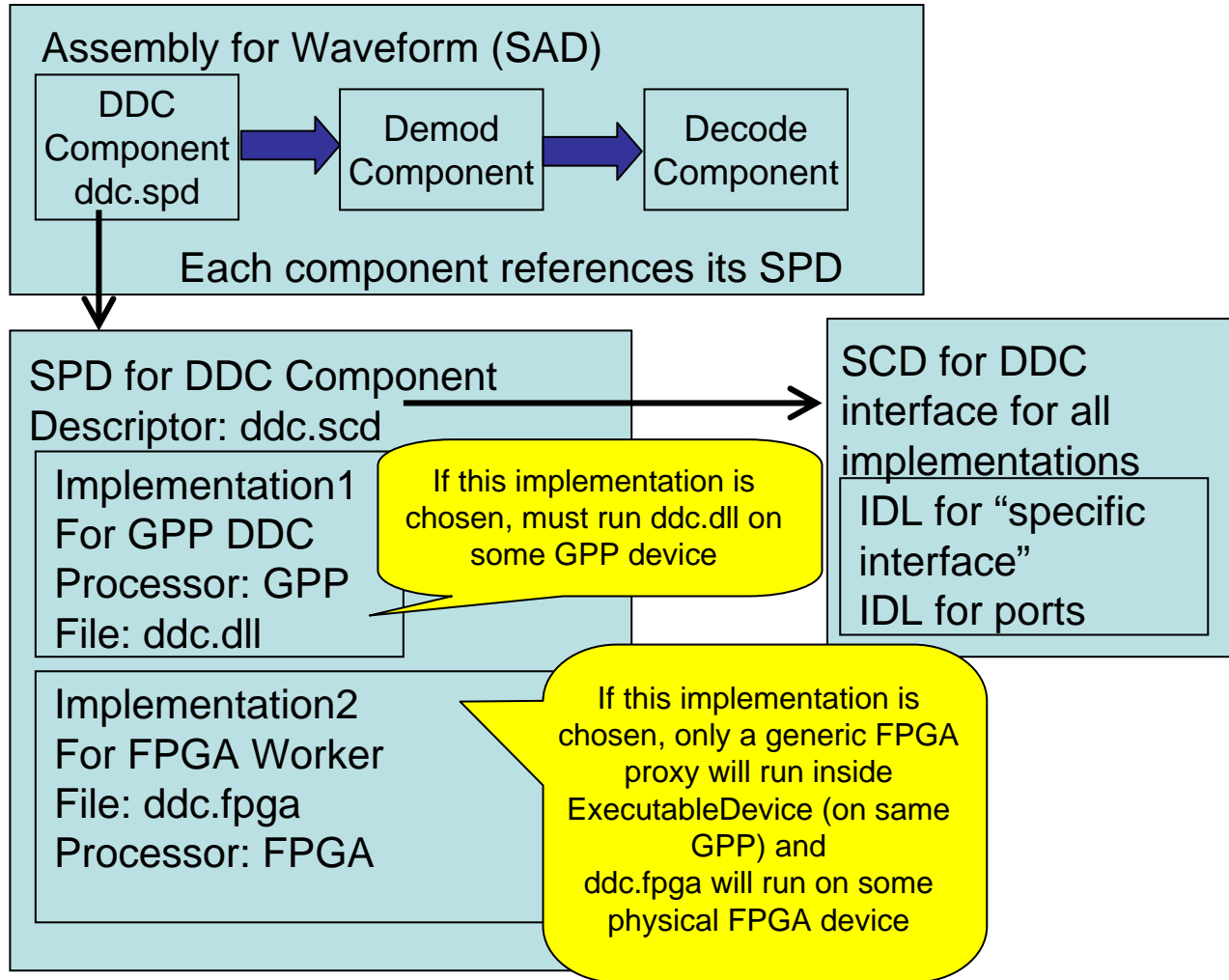
## CP289 Component Types

- **CF:: Resource interface is fixed, not inheritable**
- **Port interfaces use constrained IDL**
- **Request/response messages use derived “struct” from IDL (subset), using OMG HP CORBA Marshalling Specification**
  - Component source code fills/uses struct, sends/received messages.
  - No “stub/skeleton” generated code required or desired at this level
  - Simple arrays work fine (fixed or variable)
- **Configuration properties use a derived “struct” from property definition, using same layout rules**

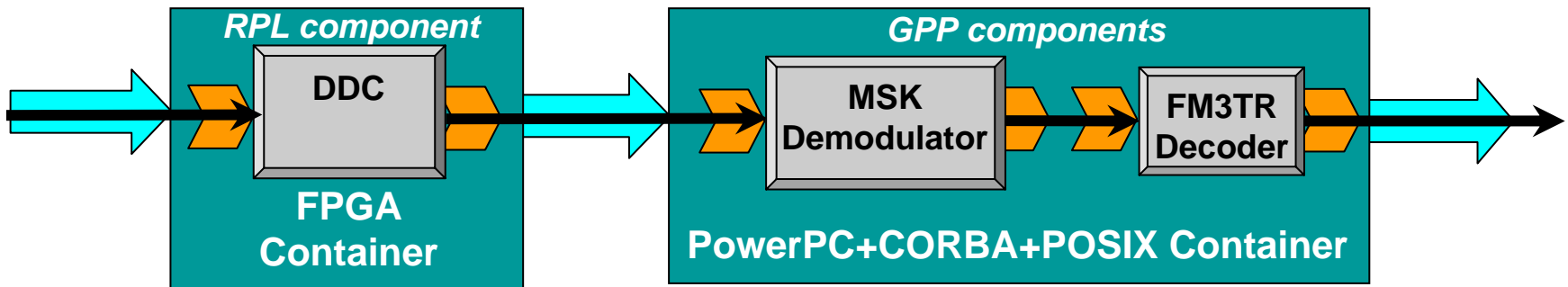
## SCA View



## SCA Metadata View



- **Adaptation to GPP ORBs is at the ORB transport level**
  - ORB transport knows what to do directly at GPP side  
*--- or ---*
  - Container may create CORBA/GIOP messages directly
  - *Choice is up to the platform integrator*
  - *No such adaptation is used or needed unless SHP and GPP component implementations are connected.*



- **Generic proxies, synthesized by SCA Logical Devices, represent the CF::Resource interface to the world**
- **Proxies acting as “adapters” can do customized translation if “interception overhead” is acceptable (legacy approach)**

- **All interfaces are C**
  - Could have a C++ version too
  - Not all DSPs have good C++ support
- **Management interface**
  - Simplified from (CCM or SCA) component model
  - Initialize/start/stop/release/configure/test/run
- **Inter-component interfaces, no generated code (no CORBA)**
  - Get/put buffers formatted from IDL-derived message structs
  - *Zero copy everywhere*
  - Can block on combinations of ports, FSM style
- **Local interfaces**
  - Roughly ANSI C without I/O
  - Optional use of threaded APIs for porting legacy DSP code
- **RCC components easily port and wrap into GPP environments**

- **DSPs with limited memory**
  - TI C5510 with 360KB on chip
  - ADI Sharc with 512KB on chip
- **RISC cores on FPGA**
  - Xilinx with PPC core using on-chip BRAM as microcontroller
- **IBM/Sony Cell processor**
  - 8 on-chip “SPUs,” like DSPs, 256KB each
- **DARPA PCA program processor architectures**
  - Multiple RISC cores with limited on-chip memories

- **All interfaces are defined using OCP: Open Core Protocol**
  - An open standard for how “IP Cores” are connected
  - Independent of VHDL vs. Verilog, interface mappings for both
  - A range of performance options
  - Increasing acceptance by users and producers of FPGAs, SoC
- **Management interface**
  - Initialize/start/stop/release/test on one OCP “thread”
  - Configure read/write on second OCP “thread”
- **Inter-component interface**
  - Burst read/write transactions on OCP-port
    - One OCP port per IDL port per direction
  - Implementation chooses master or slave role
  - Implementation chooses FIFO or random access style
- **Local interfaces**
  - Clocks and local memory access (several styles)

- **Implementation recently prototyped for RCC to validate specification**
  - DSP w/DSP BIOS, Windows, Linux
  - Footprint analysis targeting < 100KB
- **Feasibility analysis for IBM Cell SPU**
- **FPGA (RPL) implementation in progress**
- **SCA 3.1 process unclear after “strategic pause”**
- **Doesn't apply retroactively to existing programs (e.g. cluster1/MHAL)**

- **A proposed SCA 3.1 approach for integrating DSP/FPGA/Cell technology into SCA**
  - Portability, replace-ability, interoperability
  - Strong focus on resource efficiency and performance
  - Same component architecture/modeling/tools as SCA 2.x
  - RCC: C-based: DSP/Cell SPU, no CORBA required
  - RPL: OCP/VHDL/Verilog-based: FPGA/ASIC
  - Simple C-based model can be used on GPPs too
  - Simple C-based model can be automatically wrapped for CCM/SCA compliance
  - Works with non-SDR component systems (OMG/CCM)