TI | Developer Conference

February 28-March 2, 2006 • Dallas, TX

Waveform Component Portability for DSP Processing Platforms

A. Tansu Demirbilek

Systems Engineer
Mercury Computer Systems
tansu@mc.com

MERCURY Computer Systems, Inc.
Challenges Drive Innovation™

Introduction

- **Portable waveforms that can work on any hardware**
- **Re-use existing components by using different parameters**
- **Maximum portability ➔ standardize every aspect of the architecture**
  - Waveform Control
  - Data movement
  - Software Architecture
  - Security
  - etc..
- **Standards tend to create “Performance Bottlenecks”**
- **Not all of the standards are commoditized**
  - E.g. There is no “unique” way of communicating with non-GPP elements (DSPs, FPGAs)

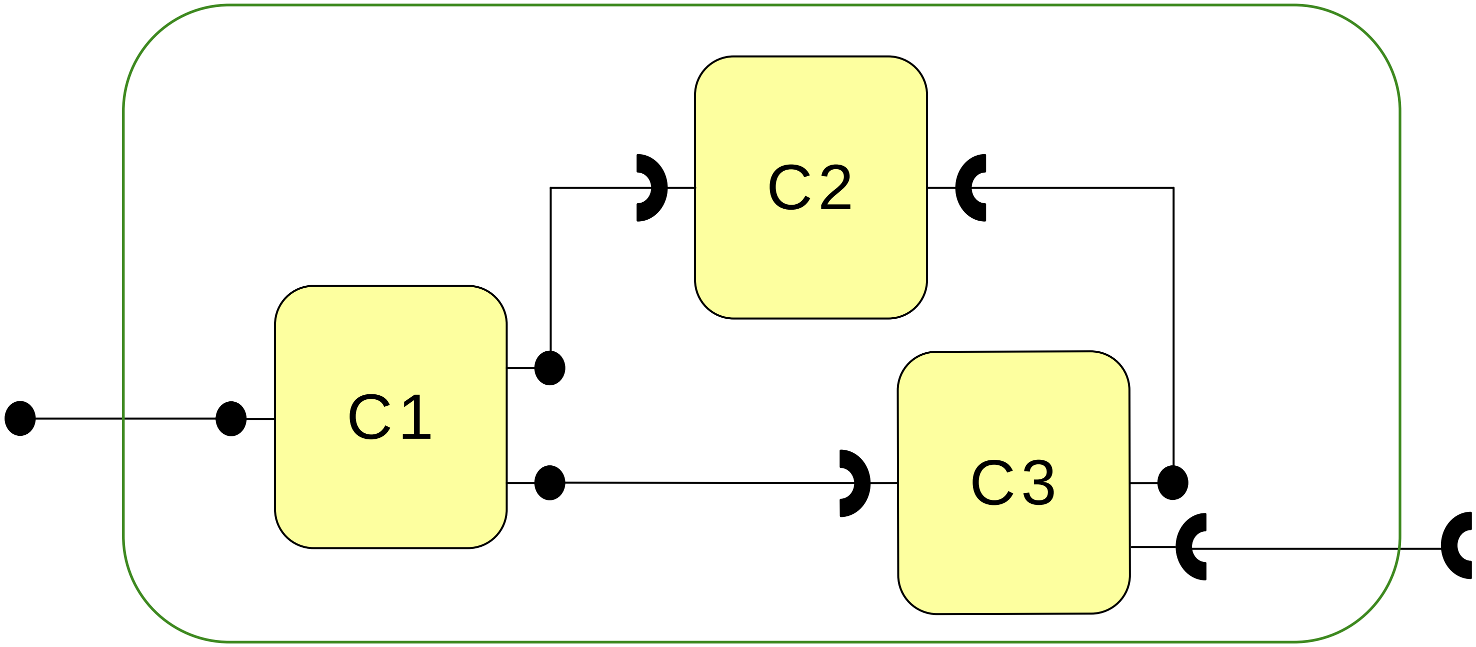
Agenda

- ➡ Component Based Programming
- Development Environment
- Deployment Engine
- Run-time
- Component model

Component Based Programming

Why use components?

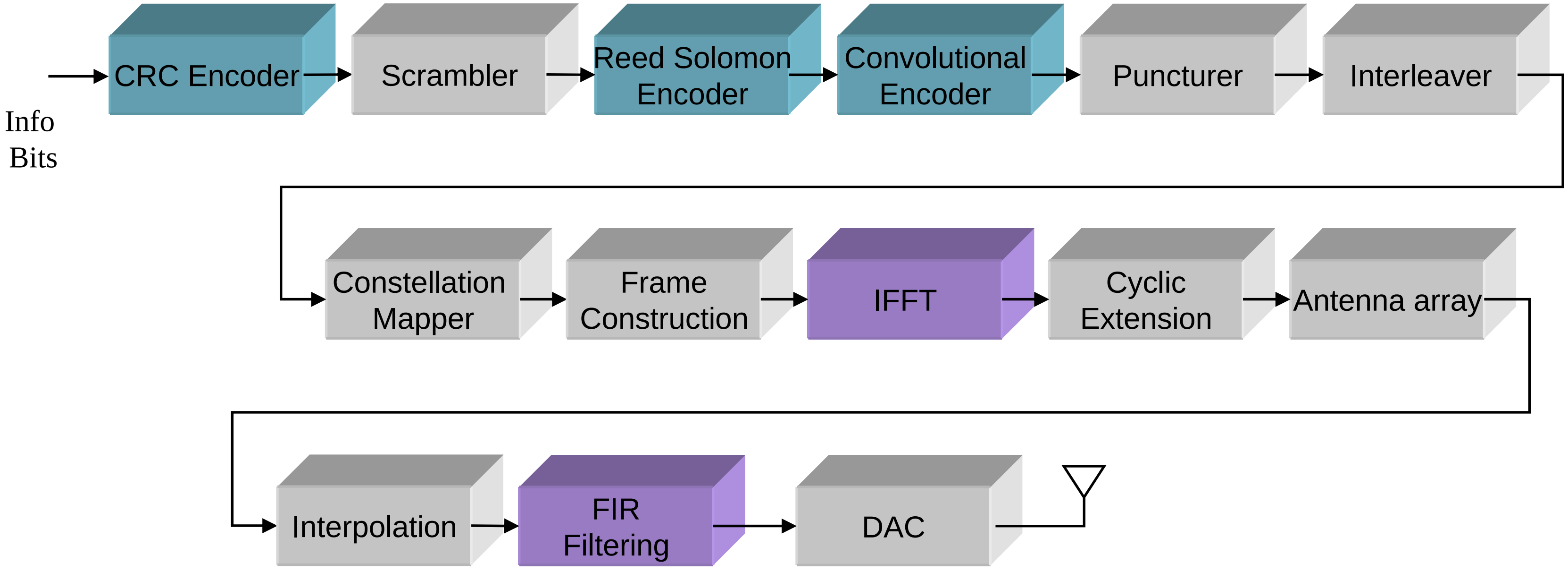
- If we build the parts of our system to well-defined specifications, then it should be easy to assemble these pieces to create a system
- It should also be relatively easy to replace one part of a system with another part that meets the same specification, *after the system is built*



TI | Developer Conference

Third Party Development

- Create an ecosystem
- Enables third party development of components
- Components can be tested and verified before integration



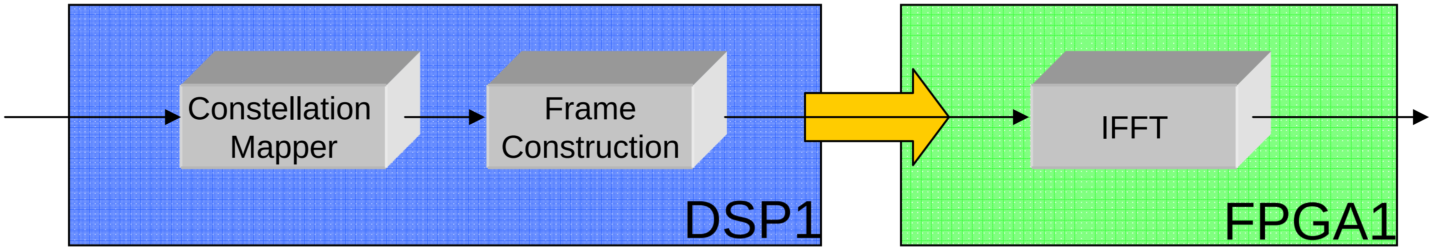
🟦 Supplied by Company A

🟪 Supplied by Company B ⬛ Internal Development

Example Use Case

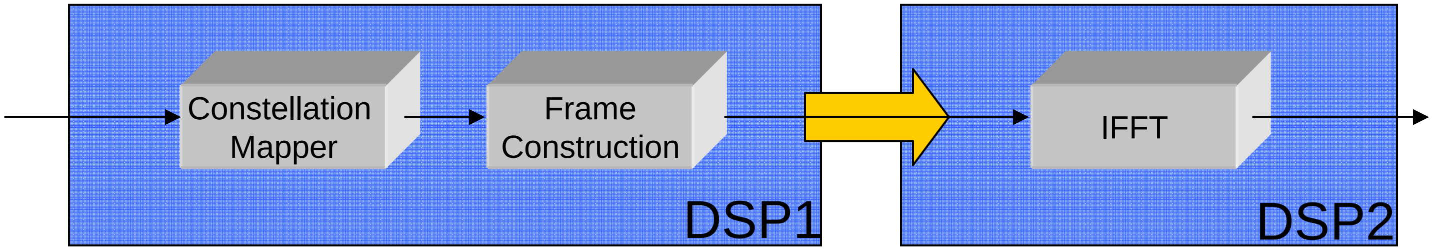
C1: Constellation Mapper
C2: Frame Construction
C3: IFFT

Deployment 1: C1, C2 to DSP1; C3 to FPGA1



- FPGA malfunction → switch to deployment plan 2…

Deployment 2: C1, C2 to DSP1; C3 to DSP2*



*Two different implementations required for C3 (I3a and I3b)

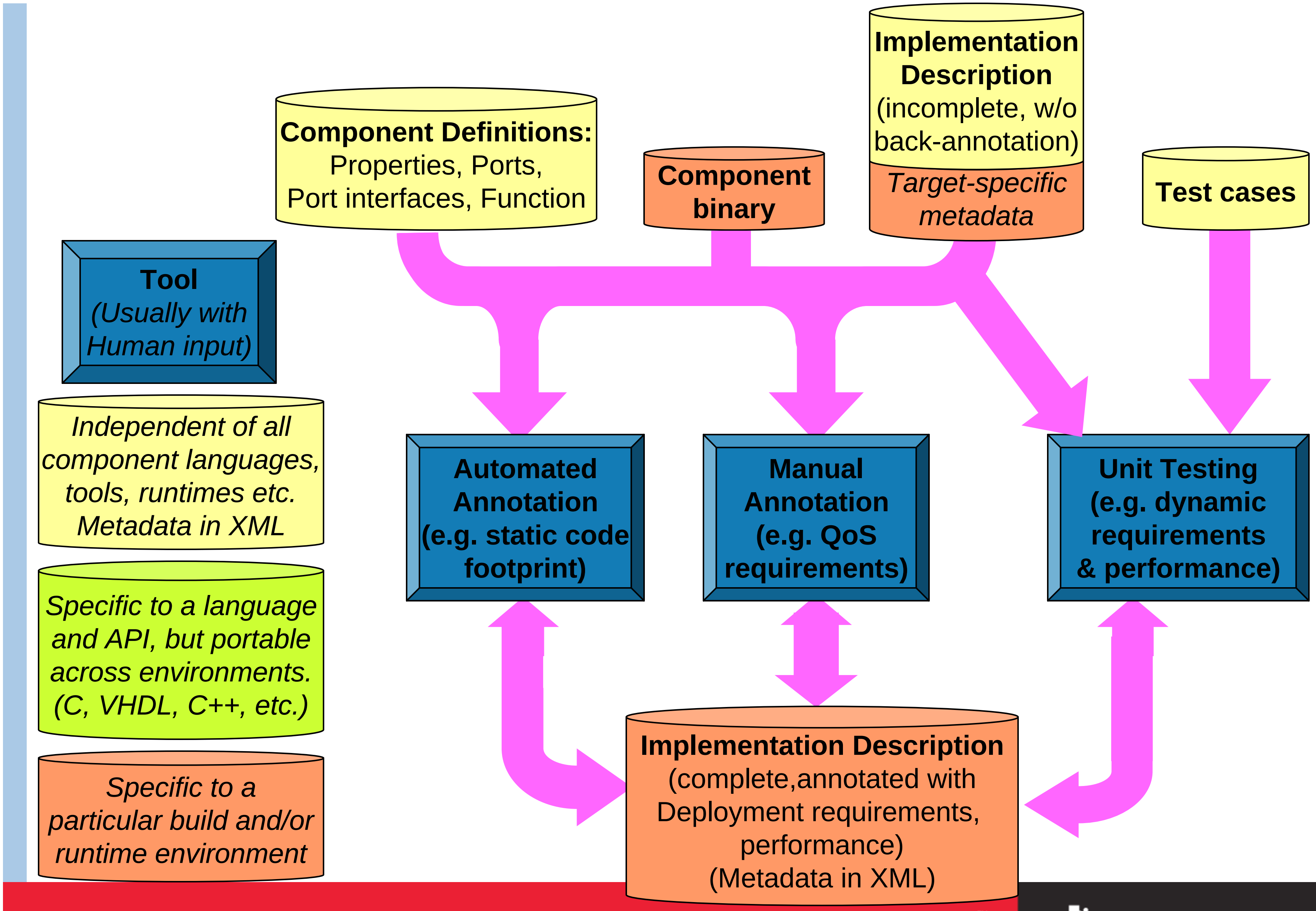
Agenda

- Component Based Programming
- ➡ Development Environment
- Deployment Engine
- Run-time
- Component model

Development Environment

1. Specify components and assemblies using a tool
2. Generate metadata (XML) and portable code (headers , declarations, and skeleton for executable code)
3. Enter target specific information and generate target specific code (wrapper code, project files, build scripts, makefiles etc)
4. Implement signal processing logic

Component Characterization: annotating implementations (describe how to deploy components)

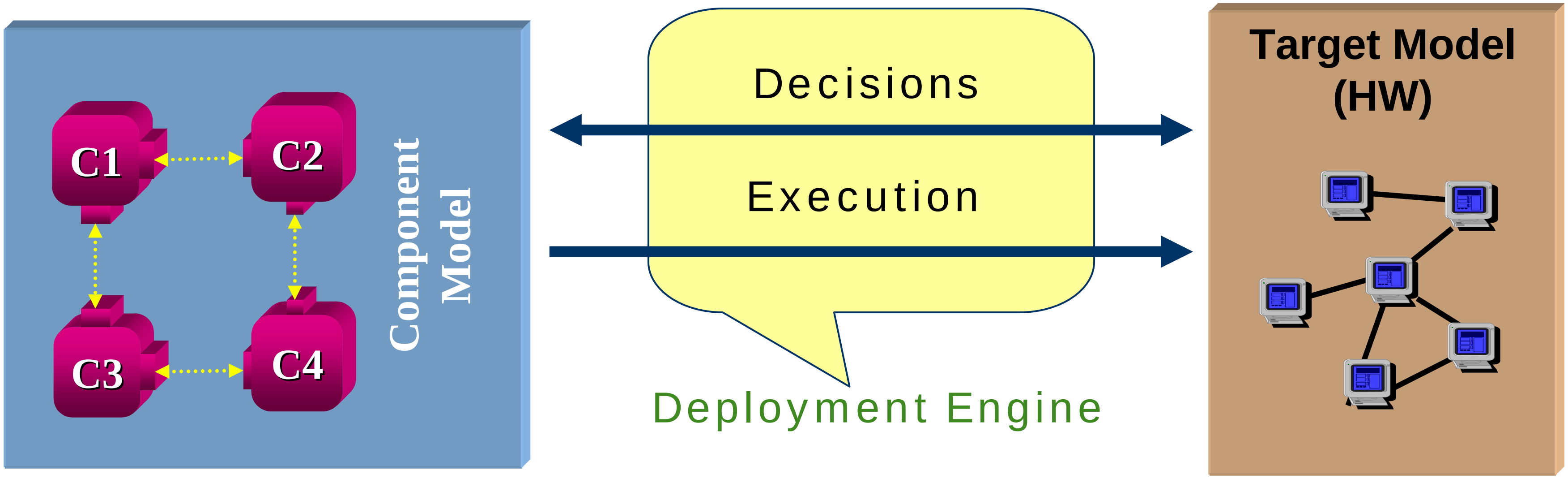

Implementation Description
(incomplete, w/o back-annotation)
Target-specific metadata
Component Definitions:
Properties, Ports,
Port interfaces, Function
Component binary
Test cases
Tool
(Usually with Human input)
Independent of all component languages, tools, runtimes etc. Metadata in XML
Specific to a language and API, but portable across environments. (C, VHDL, C++, etc.)
Specific to a particular build and/or runtime environment
Automated Annotation
(e.g. static code footprint)
Manual Annotation
(e.g. QoS requirements)
Unit Testing
(e.g. dynamic requirements & performance)
Implementation Description
(complete,annotated with Deployment requirements, performance)
(Metadata in XML)

Agenda

- Component Based Programming
- Development Environment
- Deployment Engine
- Run-time
- Component model

Deployment Engine

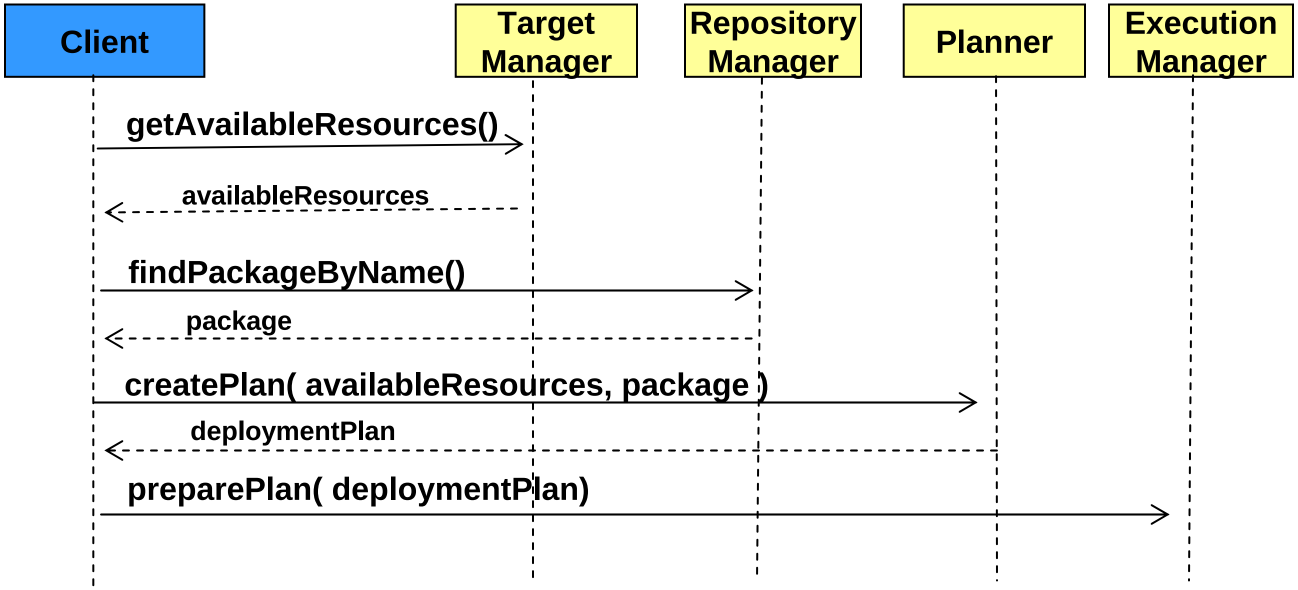
- Deployment engine automatically deploys the software modules to the available hardware, and runs the application
- Input is an XML file (specified by OMG)
- Monolithic implementation does not use CORBA
- Distributed implementation also available



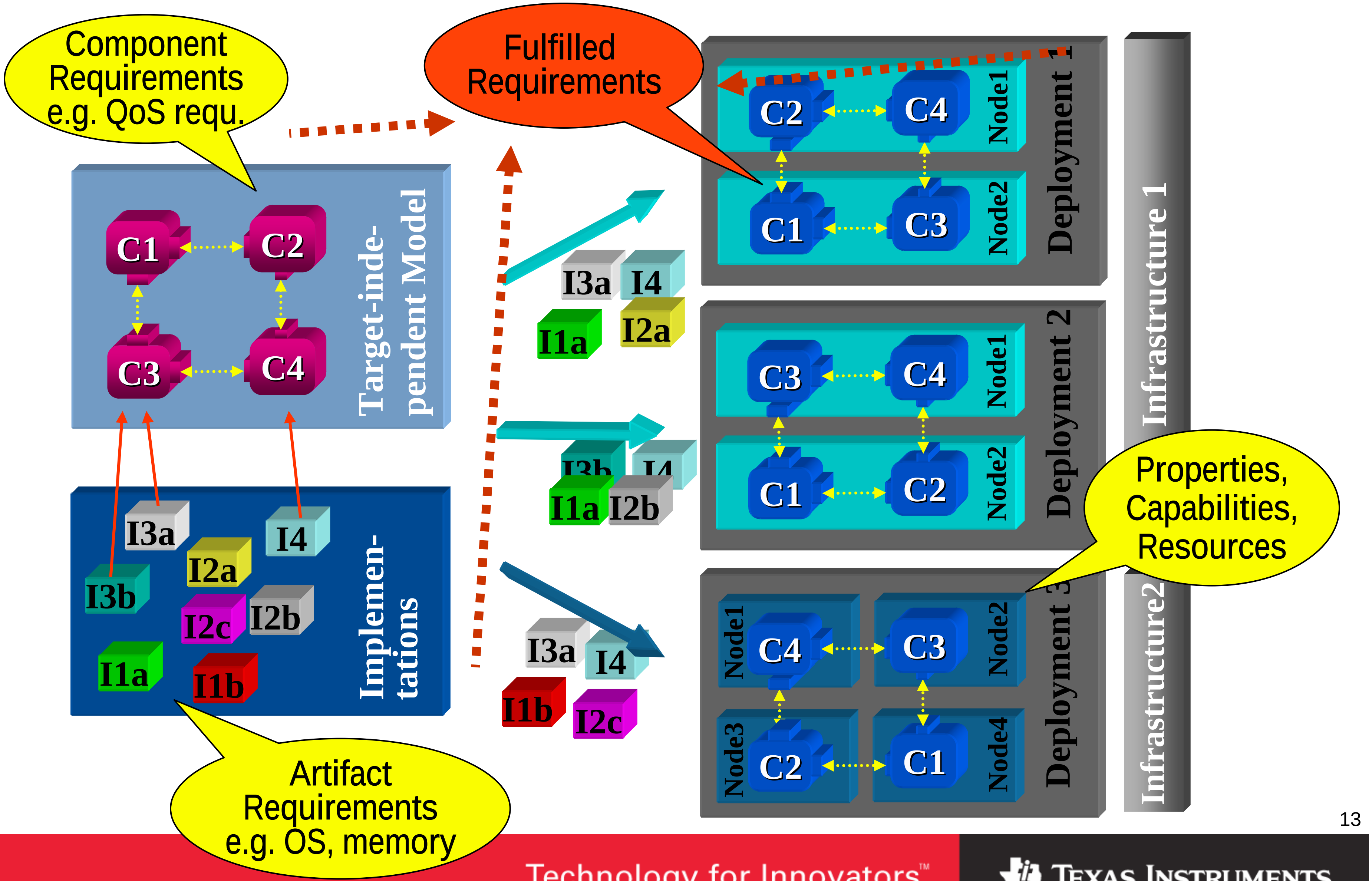
TI Developer Conference

Planning the Deployment

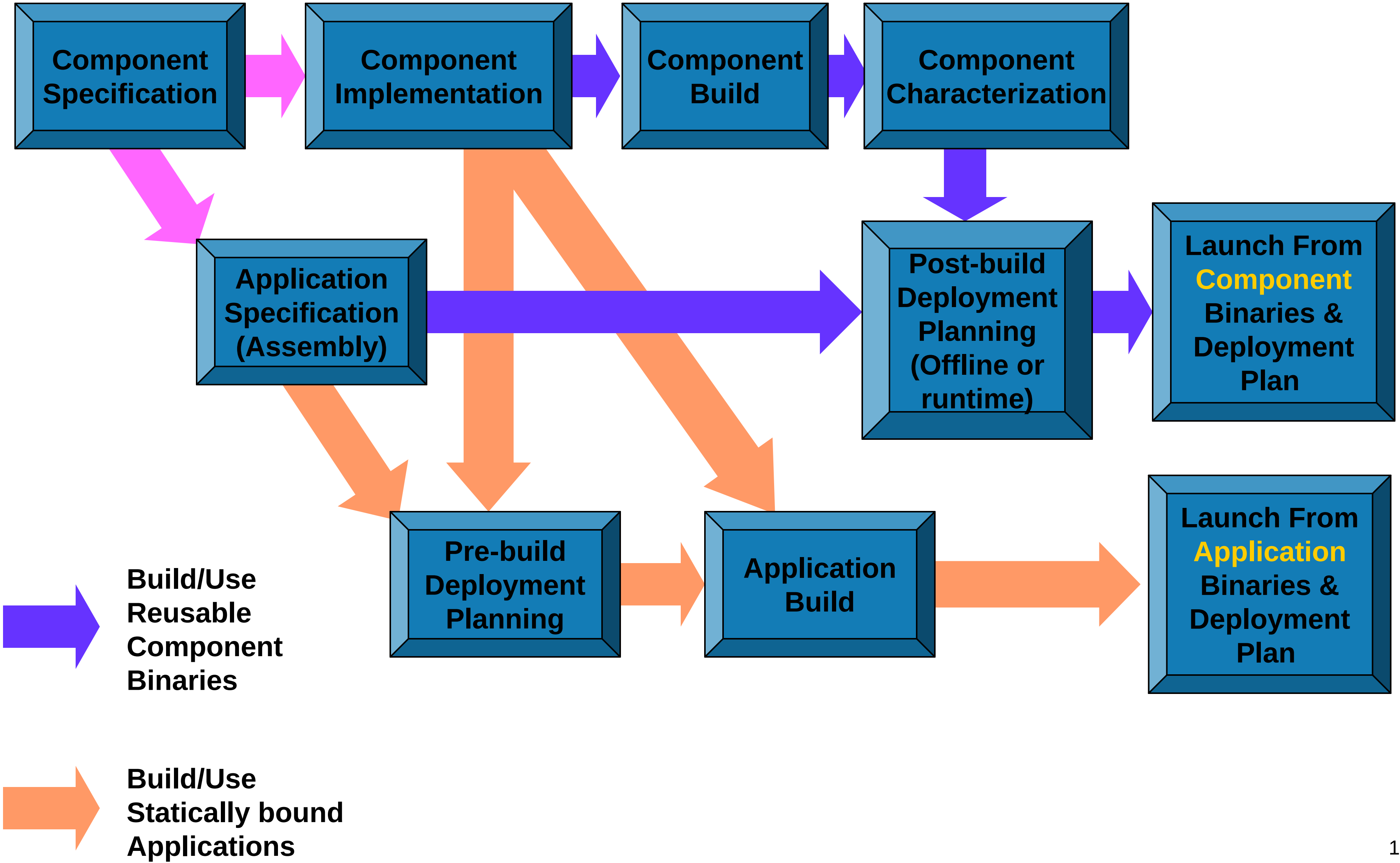
- “Planner” creates the Execution Model (Deployment Plan)
- Can be Online or Offline (Dynamic or Static)
- Generated Automatically
- Pluggable Deployment Algorithms
- Various Scenarios for Different Configurations



Fulfilling Requirements



Component-based application flows



Component Specification

Component Implementation

Component Build

Component Characterization

Application Specification (Assembly)

Post-build Deployment Planning (Offline or runtime)

Launch From Component Binaries & Deployment Plan

Pre-build Deployment Planning

Application Build

Launch From Application Binaries & Deployment Plan

Build/Use Reusable Component Binaries

Build/Use Statically bound Applications

Agenda

- Component Based Programming
- Development Environment
- Deployment Engine
- Run-time
- Component model

Run-time Monitoring

- Provided by the deployment engine
- Specific to implementation
- Start/Stop components using the control plane
- No interruption to the data plane

Agenda

- Component based programming
- Development environment
- Deployment Engine
- Run-time
- Component model

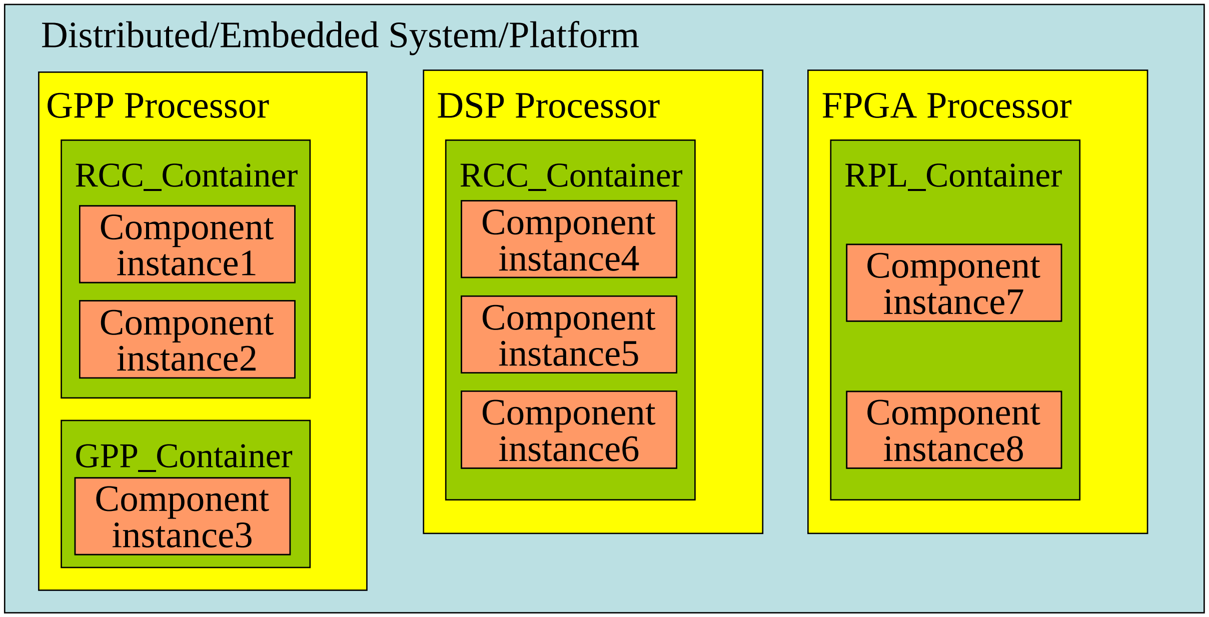
Goals

- **Overall:**
  - Waveform portability by integrating GPP, DSP and FPGA technologies into component models for embedded/distributed systems
- **Specific goals:**
  - Portability of components at source level
  - Replace-ability across technologies
  - Separation of concerns between platform provider and component author
  - Resource efficiency and performance
  - Minimal impact/changes required on existing component models

Component Portability Specification

- Works with Deployment Engine, without needing an SCA CF
- Does not have CORBA overhead
- Developed by Mercury under contract from JTRS Program Office
- Officially submitted as Change Proposal 289 to SCA
- Minimal effect to existing SCA implementations

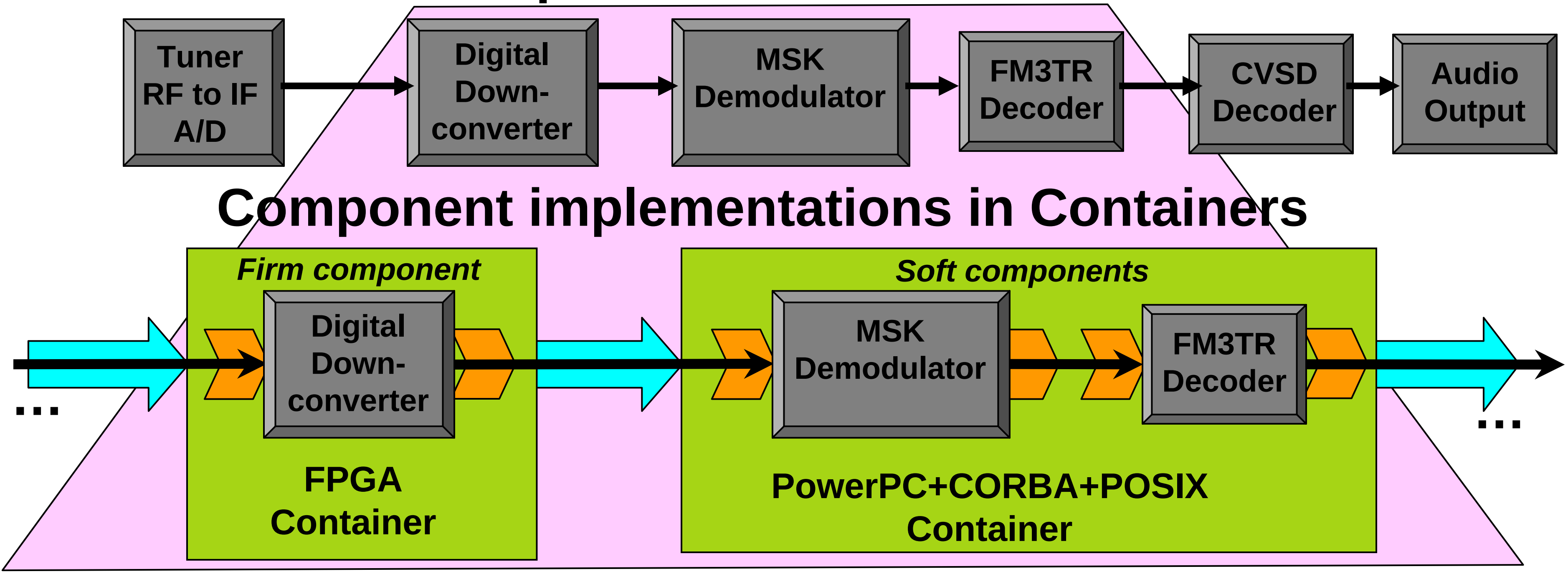
Component Model Concepts



- **Worker: (Component implementation)**
  - Interacts with containers for data transmission
  - Developed for containers which provide portability of worker code
- **Container:**
  - the *immediate* runtime environment in which a component instance executes
  - Provided by platform provider
- **Class (a.k.a. which Component Implementation Framework):**
  - A particular language/API model to which components are written

FM3TR Voice Receiver

Simple FM3TR Receiver



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

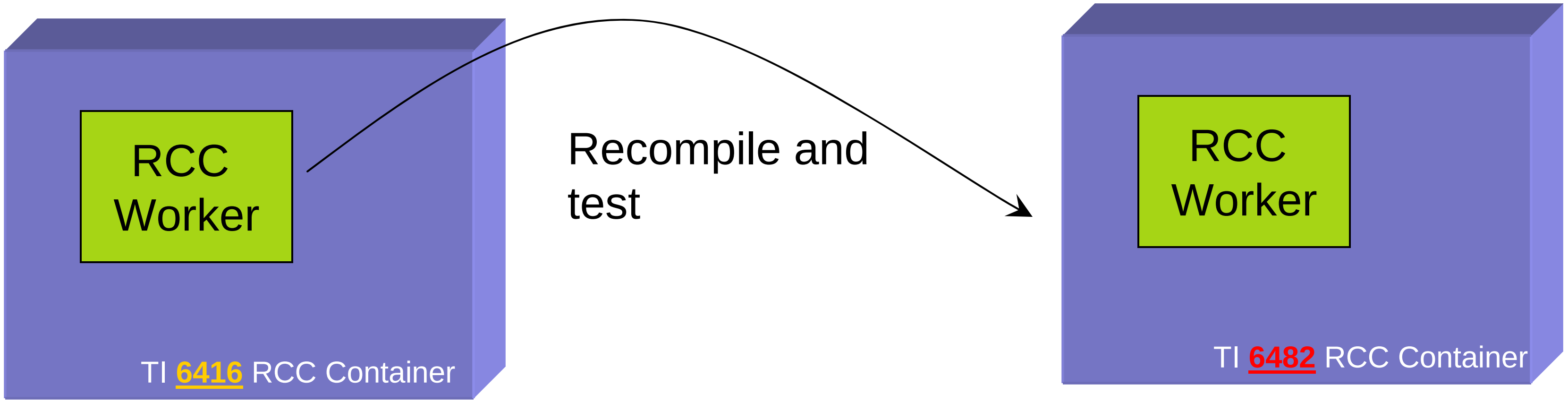
⇨ Containers talk to containers. Important interface for interoperability/plug&play of containers (e.g. boards). Protocols/networks/busses.

→ Communication between components, conveyed by their containers

RCC: Resource-Constrained C Environments

- **Component Model for DSPs**
- **Defines meta-data for the component**
- **All interfaces are ANSI C**
- **Single-threaded and Multi-threaded Profiles**
- **Management interface**
  - Simple component model
  - *Initialize/start/stop/release/configure/test/run*
- **Intercomponent interface**
  - Get/put buffers
  - Can block on combinations of ports
- **Local interfaces**
  - Roughly ANSI C without I/O

Portability

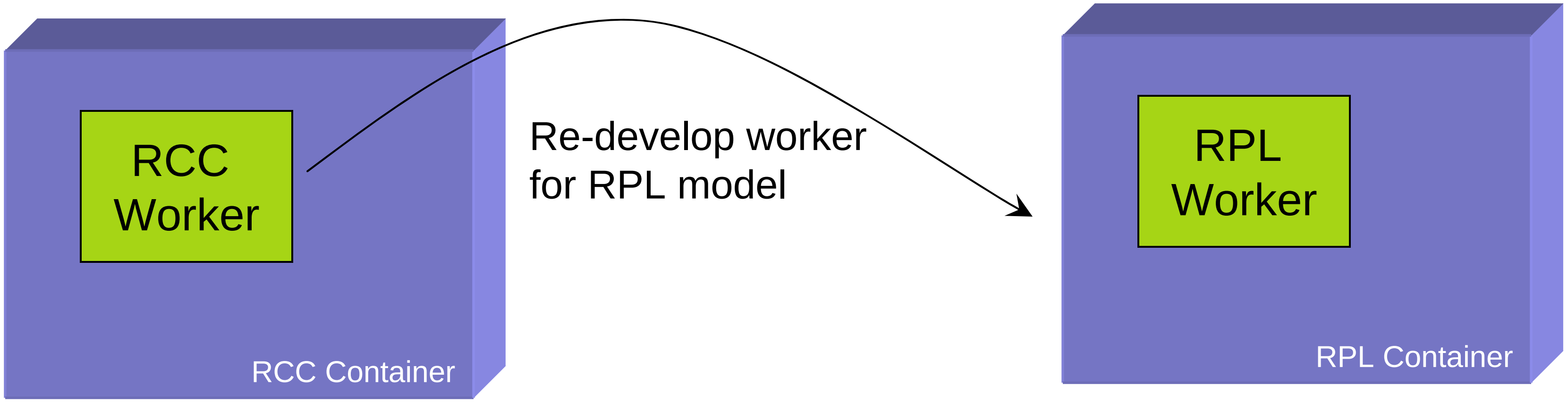


•A component implementation can move to same class of container (“like for like”), recompiling source

•Portable “reference implementations” can be tweaked to use special features (e.g. Viterbi accelerator on DSP)

•RCC components easily port and wrap into GPP environments.

Replace-ability



- **Enables changes in technology/processor class with no impact on the rest of the application**
- **Enables addition of component implementations to existing components**
  - Multiple implementations in a component package are possible
  - Allow adding FPGA implementation to component with GPP implementation without impacting application
- **Implies opaque interoperability between all classes of component implementations**

Resource Efficiency and Performance

- Minimize "tax" for portability and interoperability
- Low memory 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

Status

- **Deployment Engine:**
  - Deployment implementation is complete for online and offline planning
  - Run-time control and monitoring infrastructure can launch applications
- **Component model:**
  - Container and components are implemented for TMS320C6482, TMS320C6416 DSPs, as well as Intel Pentium and PowerPC GPPs
  - Container implementation in progress for FPGAs
- **Tools:**
  - Metadata, project files, offline deployment planning and skeleton code for DSP components can be generated by Zeligsoft Component Enabler^{TM}

Conclusion

- A component environment with support for multiple processor classes
- Highly automated development environment
- Waveforms can be easily ported from one platform or processor to another maximizing re-use
- No added CORBA overhead
- All real, exists today!

Waveform Component Portability for DSP Processing Platforms

Mercury Computer Systems, Inc.
Challenges Drive Innovation™

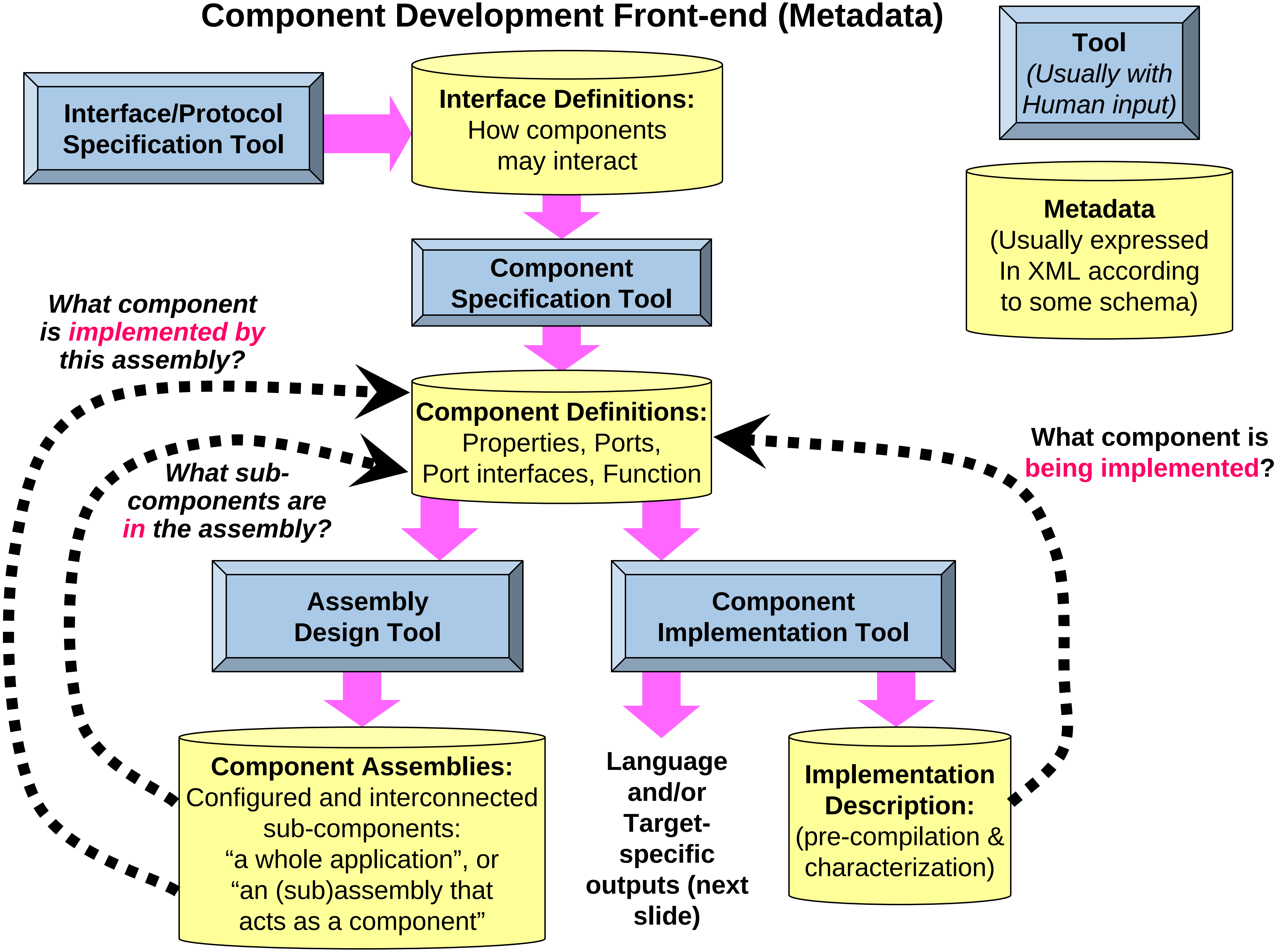
Tansu Demirbilek

Systems Engineer
Mercury Computer Systems
tansu@mc.com

SEE THE FUTURE

CREATE YOUR OWN

Component Development Front-end (Metadata)



Tool
(Usually with Human input)

Metadata
(Usually expressed In XML according to some schema)

Interface/Protocol Specification Tool

Interface Definitions:
How components may interact

Component Specification Tool

Component Definitions:
Properties, Ports, Port interfaces, Function

What component is implemented by this assembly?

What sub-components are in the assembly?

What component is being implemented?

Assembly Design Tool

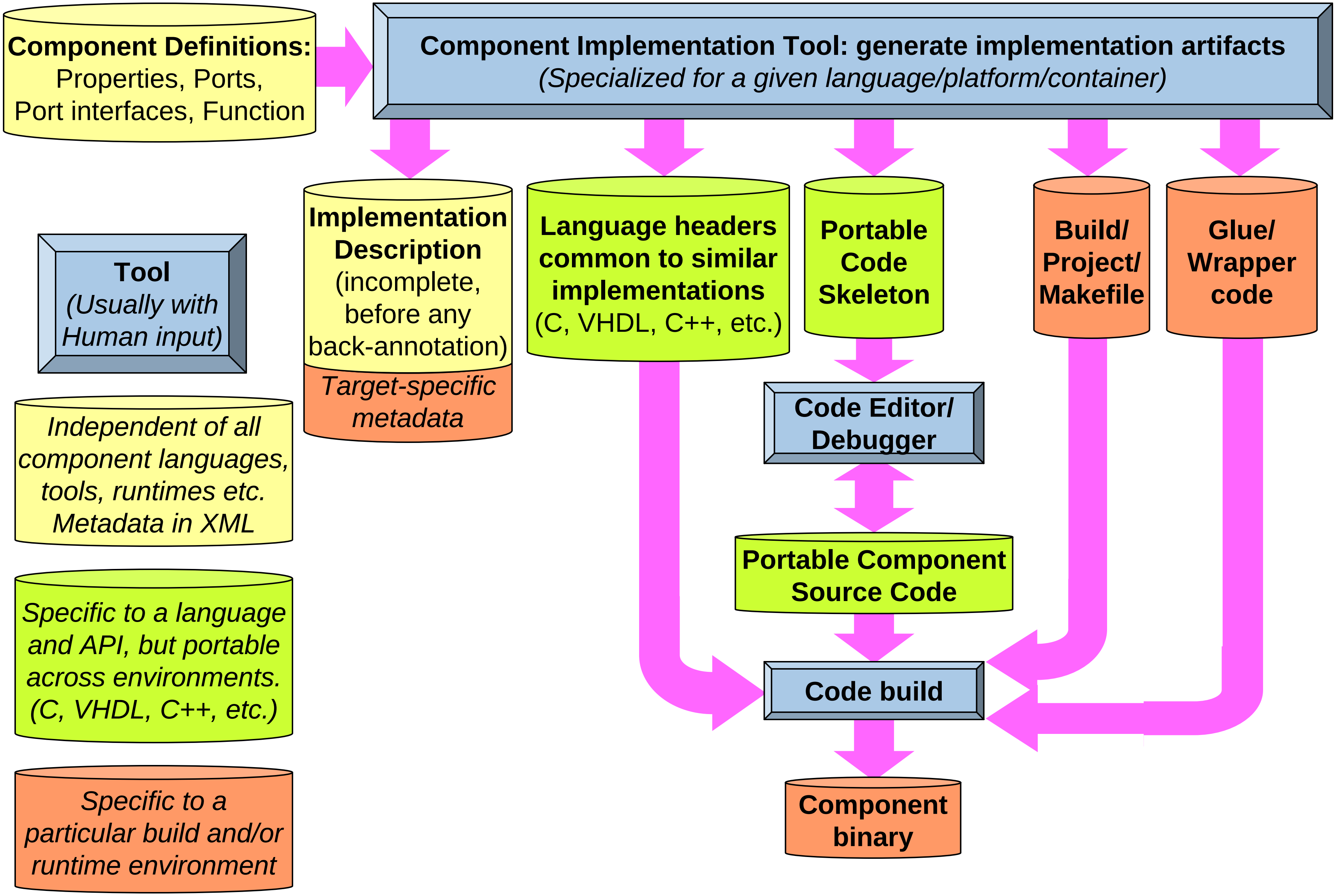
Component Implementation Tool

Component Assemblies:
Configured and interconnected sub-components:
“a whole application”, or
“an (sub)assembly that acts as a component”

Language and/or Target-specific outputs (next slide)

Implementation Description:
(pre-compilation & characterization)

Component Development Back-end: Creating Implementations (deployable binaries for each individual component)



Component Definitions: Properties, Ports, Port interfaces, Function

Component Implementation Tool: generate implementation artifacts *(Specialized for a given language/platform/container)*

Implementation Description (incomplete, before any back-annotation)

Target-specific metadata

Language headers common to similar implementations (C, VHDL, C++, etc.)

Portable Code Skeleton

Build/ Project/ Makefile

Glue/ Wrapper code

Code Editor/ Debugger

Portable Component Source Code

Code build

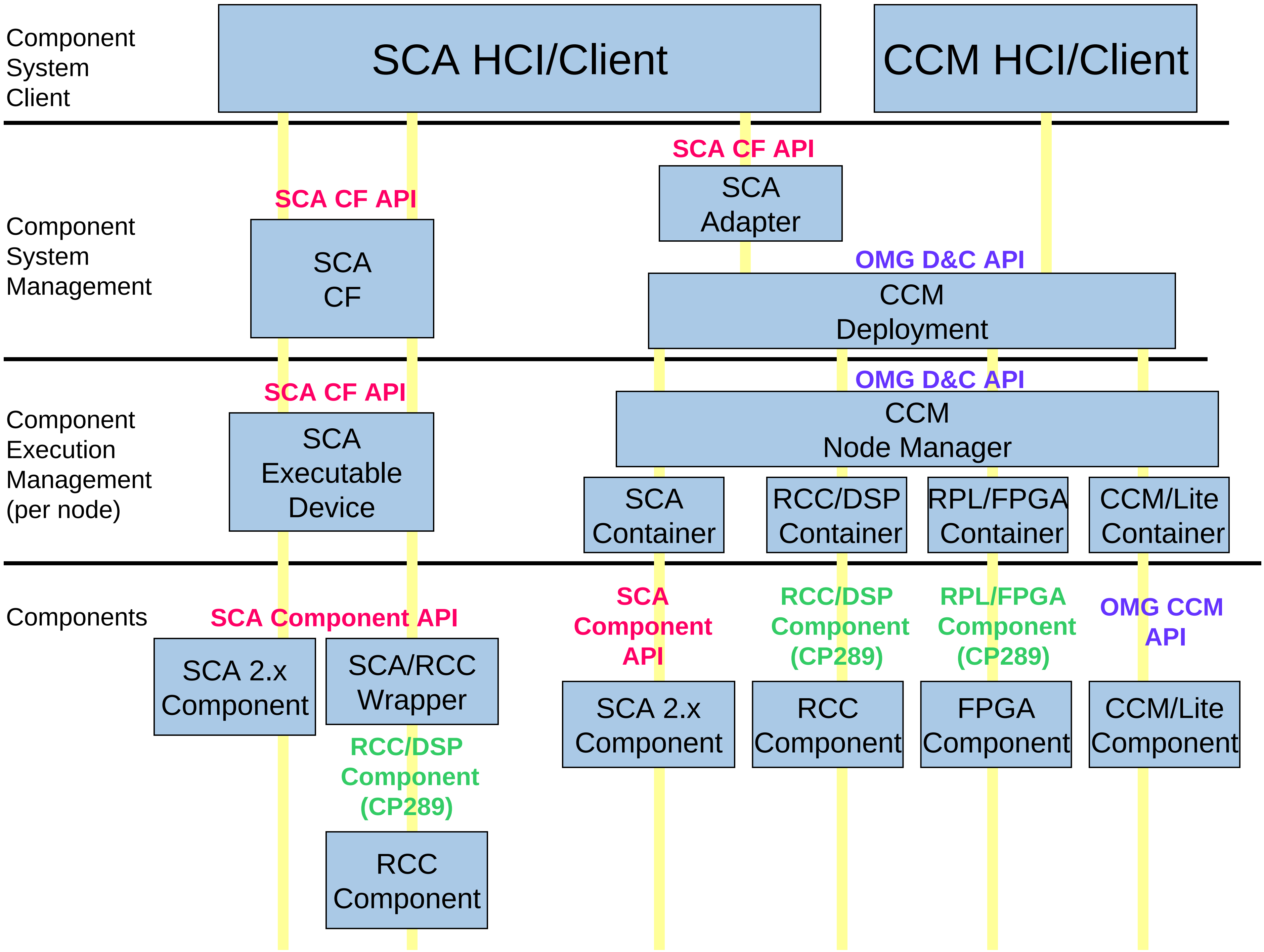
Component binary

Tool *(Usually with Human input)*

Independent of all component languages, tools, runtimes etc. Metadata in XML

Specific to a language and API, but portable across environments. (C, VHDL, C++, etc.)

Specific to a particular build and/or runtime environment



Component System Client

SCA HCI/Client

CCM HCI/Client

Component System Management

SCA CF API

SCA CF

SCA CF API

SCA Adapter

OMG D&C API

CCM Deployment

Component Execution Management (per node)

SCA CF API

SCA Executable Device

OMG D&C API

CCM Node Manager

SCA Container

RCC/DSP Container

RPL/FPGA Container

CCM/Lite Container

Components

SCA Component API

SCA 2.x Component

SCA/RCC Wrapper

RCC/DSP Component (CP289)

RCC Component

SCA Component API

SCA 2.x Component

RCC/DSP Component (CP289)

RCC Component

RPL/FPGA Component (CP289)

FPGA Component

OMG CCM API

CCM/Lite Component