Middleware for Embedded Systems

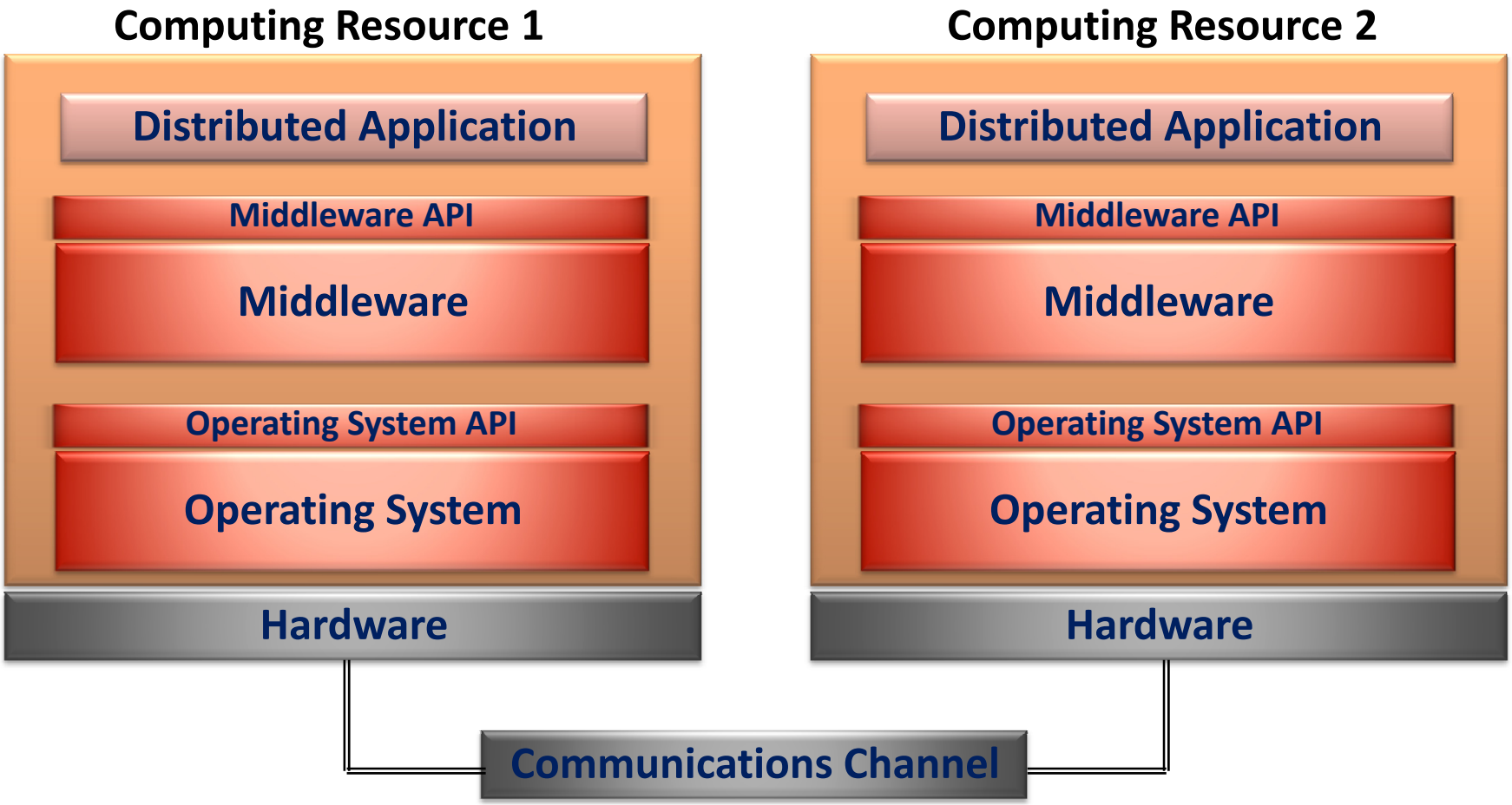
Steven P. Smith

Fall 2009

Middleware

- *Middleware* is a term that refers to a class of software technologies designed to help manage the complexity and heterogeneity inherent in parallel and distributed systems.
- Middleware operates between the application layer software and the operating system.
- Middleware may also be thought of as provisional operating system functionality.
  - The Internet Protocol (IP) was not an integral part of most operating systems until well into the 1980s.

Middleware


Computing Resource 1
Distributed Application
Middleware API
Middleware
Operating System API
Operating System
Hardware
Computing Resource 2
Distributed Application
Middleware API
Middleware
Operating System API
Operating System
Hardware
Communications Channel

Embedded Middleware

- **Most embedded middleware targets cooperative, distributed execution in heterogeneous computing environments.**
  - Early examples include Remote Procedure Calls (RPC)
    - Widely supported across different platforms and operating systems
    - Provides standardized method invocation
    - Does not address differences in data representation across heterogeneous systems
      - Word size, Endianess
- **CORBA: The Common Object Request Broker Architecture**
  - First released in 1991, CORBA sought to standardize method call semantics between application objects, regardless of where those objects reside.
  - Supported across a very large range of operating systems, processor architectures, and programming languages.
  - Developed and maintained by the Object Management Group (OMG).

Common Object Request Broker Architecture

- **Method invocation specifications are captured in a language-independent manner using the CORBA Interface Definition Language (IDL).**
  - IDL provides a means of capturing object interfaces with strong (precise) data types and parameter input/output declarations.
  - IDL specifications of object interfaces are translated into language-specific source code *stubs* for the object invocation and its implementation.
    - The invocation stub is "complete" (but not particularly useful) as generated.
  - Object developers then provide the object implementation software in their language of choice.
  - The object implementation may be platform-specific.
  - The object interface remains platform independent.

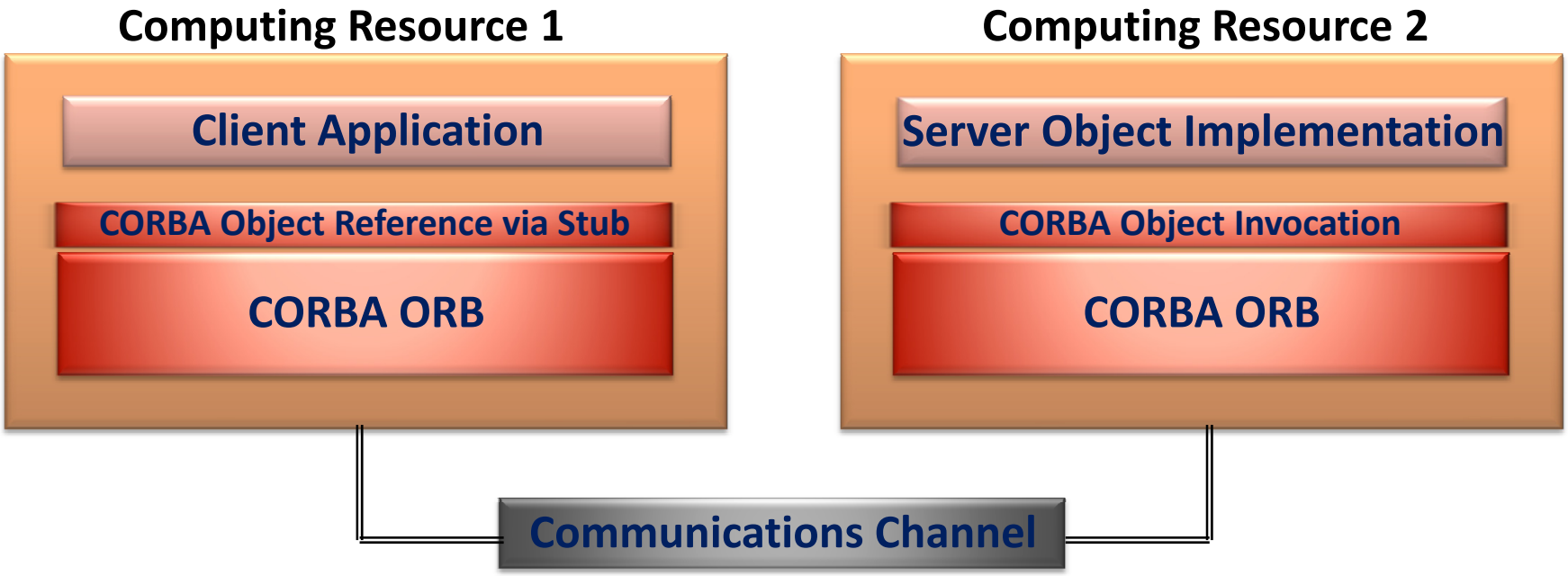
Common Object Request Broker Architecture

- **CORBA uses an Object Request Broker (ORB) to manage objects and accesses to them.**
  - The ORB addresses all compatibility issues.
  - The ORB is basically a platform-independent RPC capability with some additional services, such as an object directory.
- **CORBA employs a client/server Model.**
  - Application software represents the client.
  - Object access requests from the application (client) are made to the local ORB.
  - The ORB looks up the location of the requested object.
  - The ORB that owns the object implementation is contacted by the local ORB.
  - That ORB then invokes the object implementation, which represents the server in the client/server model.

Common Object Request Broker Architecture

- **Method invocation sequence in CORBA.**
  - Upon receipt of an object access request by a client application, the ORB on the requesting application's host:
    - Determines if the requested object is known to it.
    - Determines the location of the object.
    - Marshalls the parameters (converts them to a platform-independent form).
    - Calls the object via the (possibly remote) ORB that "owns" the object, essentially using RPC.
    - Receives method invocation results, again via RPC.
    - Reverses the marshalling process ("demarshalling") for IDL parameters that were tagged as "output" or "inout".
    - Returns the results in the appropriate, platform-dependent form to the client.
  - CORBA objects are accessed by reference, but (almost) all parameters are passed by value.

Common Object Request Broker Architecture



Common Object Request Broker Architecture

- **CORBA is widely used to provide "glue" functionality for enterprise software and complex database systems.**
  - This community was the primary early adopter of CORBA implementations.
- **Interest in CORBA for embedded systems grew as embedded systems became increasingly networked and complex.**
  - Interoperability in heterogeneous systems is an obvious area for the application of standards.
  - But… CORBA's motivating objective was platform, language, and OS independence, with no consideration paid to performance or efficiency.
    - Performance matters were left entirely to ORB implementers.
    - But the full CORBA specification offers few opportunities for meaningful performance optimizations.
  - Result was the emergence of CORBA/e for embedded systems in 2001.

CORBA/e

- **CORBA/e includes most of the capabilities defined by the CORBA-RT specification.**
  - First released in 2001, CORBA-RT extends CORBA to include object invocations with deadlines, task priorities, and hazard prevention (e.g., mutexes).
  - Real-time scheduling policies separated from CORBA and may be user-defined, but CORBA/e (compact profile) supports only static scheduling.
- **CORBA/e introduced the concept of *domain profiles* as a recognition of the limitations of highly resource constrained embedded systems.**
  - Domain profiles specify subsets of the full CORBA specification that must be supported.
  - The subsets were designed to reduce implementation complexity while retaining key CORBA functionality.

CORBA/e Domain Profiles: Compact and Micro

- **The CORBA/e Compact Profile**
  - Targets 32-bit embedded systems with fewer resources than typically found in standard desktop and notebook systems.
  - A real-time operating system (RTOS) is required for compliance with CORBA-RT capabilities.
    - Real-time Linux included, of course.
  - No support for the dynamic aspects of CORBA or the CORBA Component Model (CCM).
- **The CORBA/e Micro Profile**
  - Targets more resource constrained systems, including high-end DSPs.
  - Beyond Compact Profile limitations, drops support for CORBA-RT except for mutexes. Also drops support for some complex data types (ValueType, for object access by value instead of reference, and AnyType).

CORBA/e

- Both CORBA/e domain profiles maintain full compatibility with the Internet Inter-ORB Protocol (IIOP).
  - Protocol defines interface between ORBs, regardless of vendor or platform.
  - All CORBA/e implementations can interact with full CORBA implementations.
- CORBA/e still won't fit on very highly resource-constrained systems (e.g., 8-bit and many 16-bit systems, or systems with limited storage).
- Constant tension between platform-independence and resource constraints.
  - Also fighting cultural aspects of embedded development community.
    - Reusable, platform independent software remains a rarity.
    - Parts of the community are still suspicious of high level programming languages…

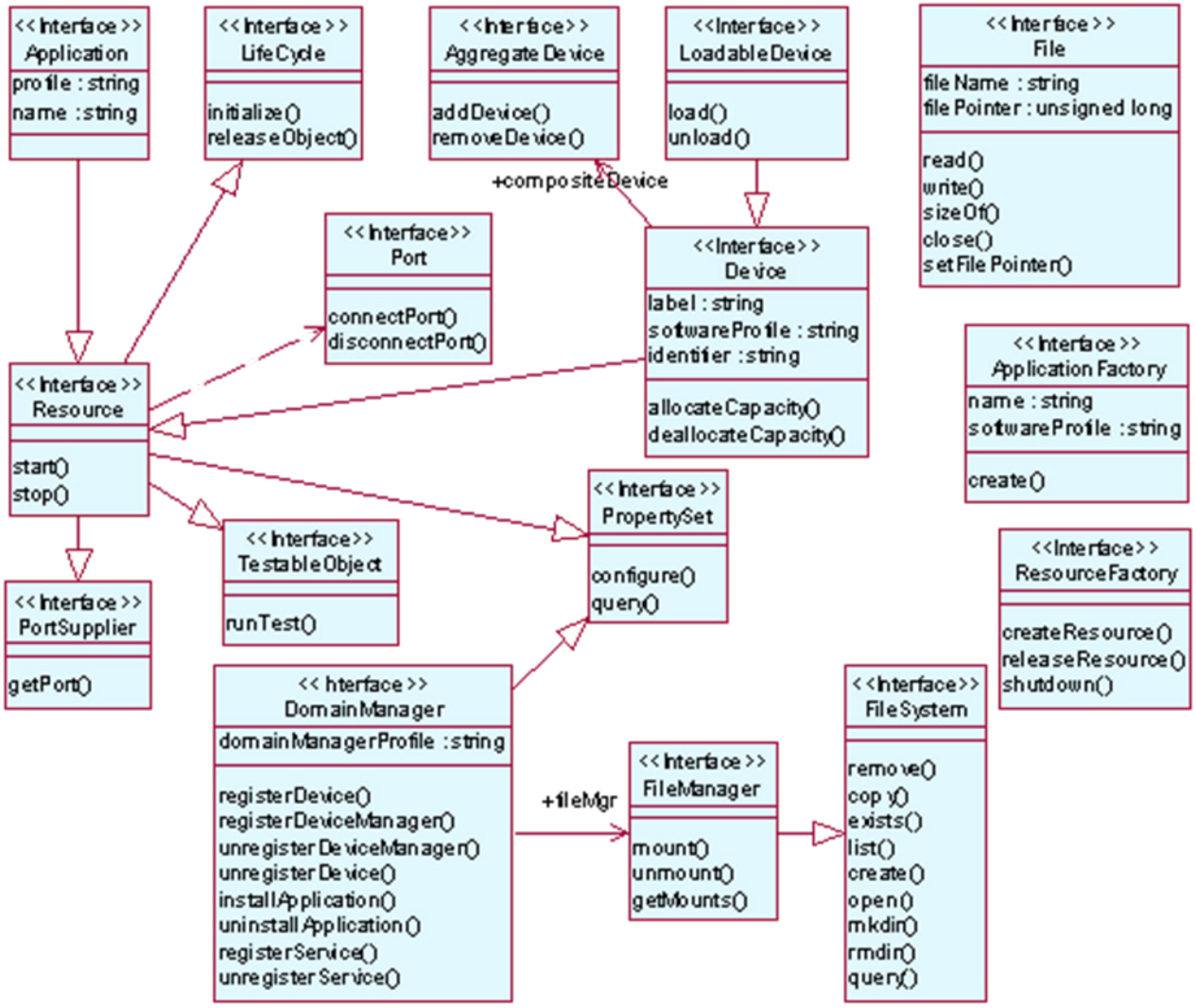
Software Communications Architecture

- **The Software Communications Architecture (SCA) is a middleware standard and specification defined by the Depart of Defense as a part of the Joint Tactical Radio System (JTRS) program.**
  - **The SCA was originally intended to facilitate the design of platform independent "waveforms" for the DoD-wide JTRS software defined radio program that began in the late 1990s.**
    - **In the JTRS context, a *waveform* is the set of end-to-end software and associated design artifacts necessary to realize a specific communications protocol.**
    - **Many companies are developing JTRS-compliant platforms of one form or another. SCA was defined to ensure that a given waveform would need only to be recompiled to support a new target architecture and platform.**

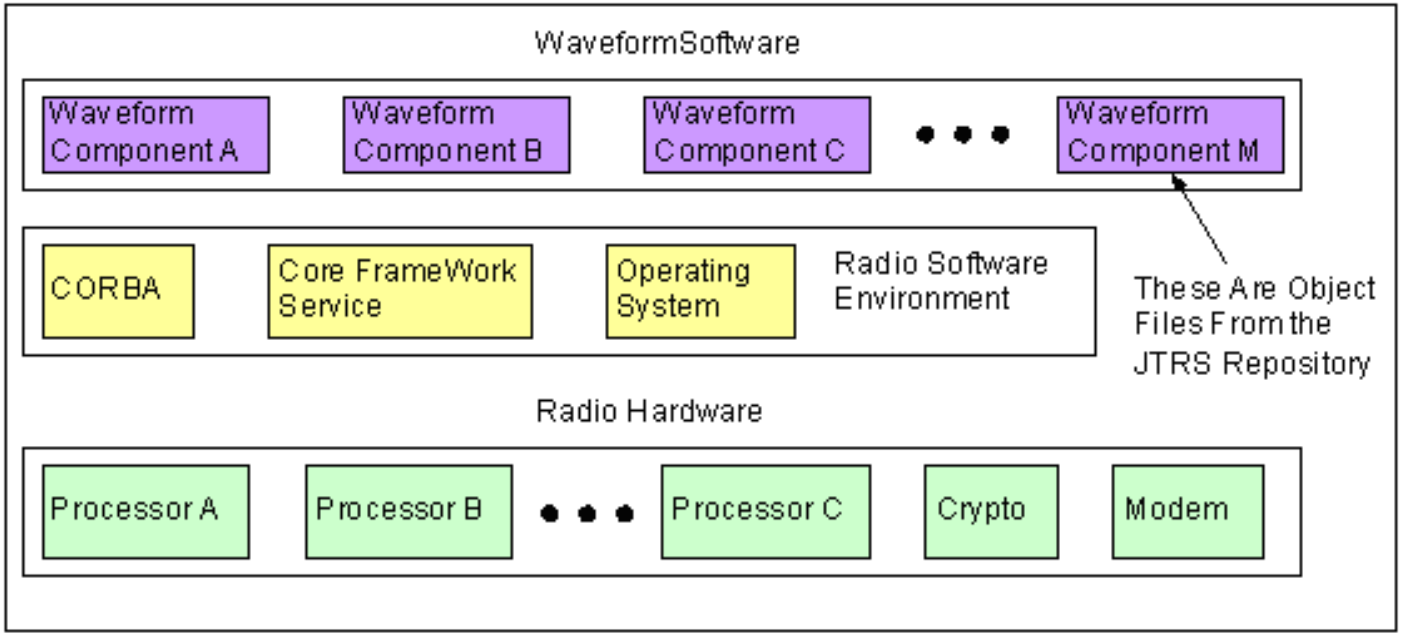
Software Communications Architecture

- **What is the SCA?**
  - Perhaps surprisingly, the SCA specification contains *nothing* directly related to communications or to software defined radio (SDR).
  - Rather, the SCA defines a fairly minimal virtual machine that sits atop an RTOS and defines how real-time embedded applications are created as a collection of interconnected component objects.
    - Data is passed among the components using ports and virtual interconnect.
    - Components, their properties, and their instantiation recipes are captured in XML files that are interpreted to build a waveform application instance.
  - The SCA operates on top of CORBA and CORBA-RT.

SCA Objects and Interfaces



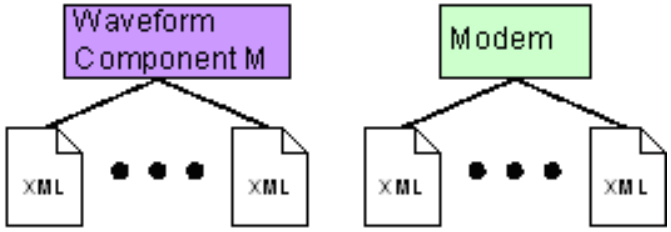
SCA Component Placement



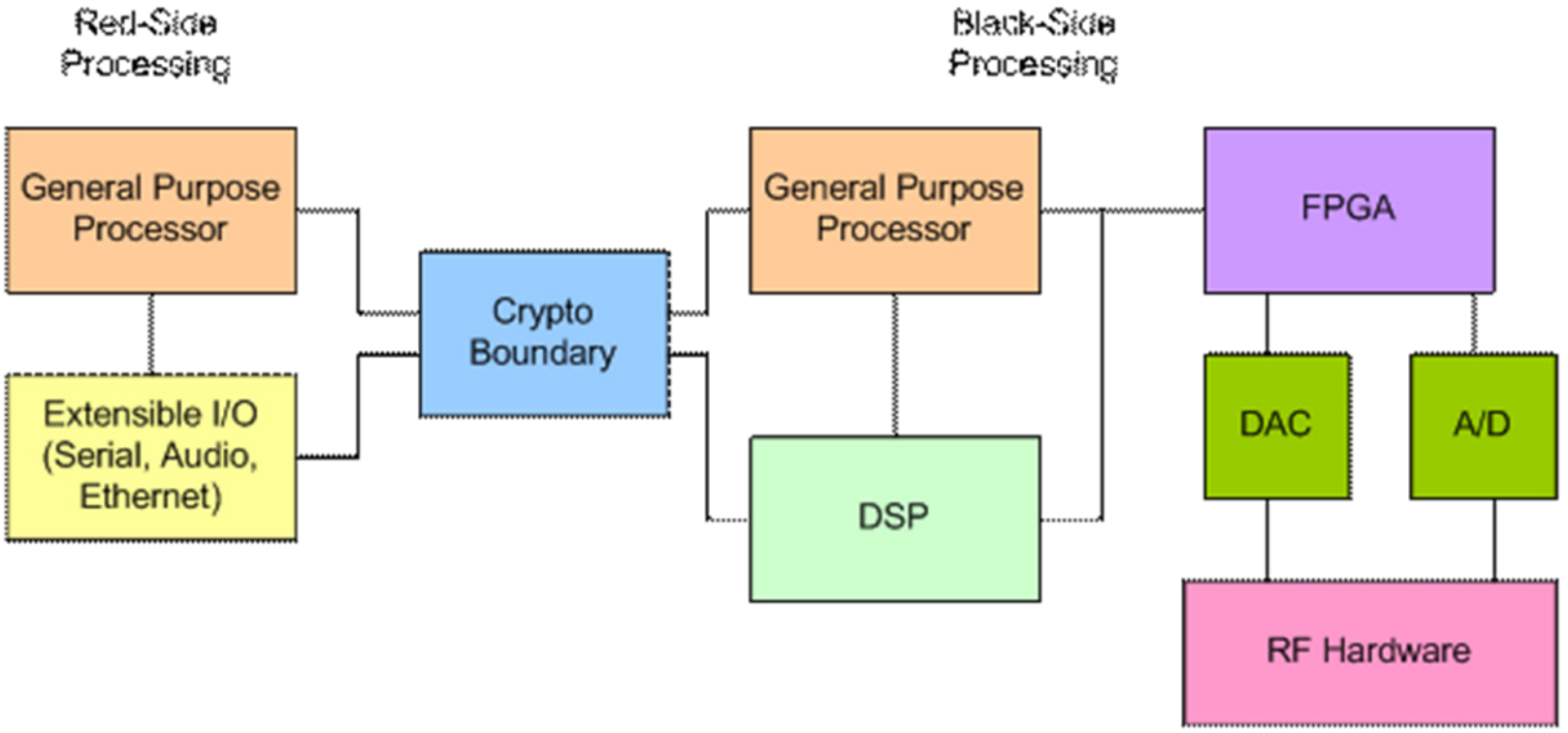
Waveform Component M

This Component Expects Other Components and Hardware Devices, but Doesn't Know Their Location or Capacity

Instead of Having Custom Builds for Individual Radios, We Have XML Files Associated With the Waveform Components and Hardware Devices that Allows the Waveform Startup to Find and Allocate Components Dynamically



One Possible JTRS Hardware Configuration



Software Communications Architecture

- **The DoD has initiated efforts to promote the adoption of the SCA as a broadly useful standard for the development of real-time embedded systems.**
  - **Commercial telecommunications vendors have been urged to adopt the SCA for their SDR efforts.**
  - **Commercial adoption has been nil thus far.**
    - **If CORBA is still viewed with due derision by embedded developers, then adding another layer above it is unlikely to assuage commercial concerns about system performance and cost competitiveness.**
    - **The JTRS program is exploring changes to the SCA to address efficiency concerns for small form factor and battery powered radios. (JTRS defines platform variants that range from "big iron" systems to PDA-sized devices.)**
      - **Current SCA platforms spend as much as 70% of their computational effort inside CORBA.**
      - **So, an SCA-compliant system will require roughly 4x the horsepower of a custom SDR implementation with no CORBA or SCA.**

Other Forms of Middleware for Embedded Systems

- ZigBee is a low-power wireless radio protocol that defines a middleware layer and application structure for interoperability of compliant radios, regardless of vendor.
  - Domain Profiles are being defined for Lighting Control, Home Automation, HVAC, and Building Automation.
- Middleware is also being developed to simplify the use of more exotic computing resources, including:
  - FPGA middleware that facilitates the use of partial dynamic reconfiguration
    - Integration of these capabilities with the SCA specification is also being explored.
  - Emerging parallel and array-based architectures.

The Future of Middleware for Embedded Systems

- **Primary obstacles to more widespread use of middleware in embedded systems stem from two sources.**
  - Cultural resistance to certain more advanced development methodologies in the embedded system development "ecosystem," which is made up largely of small engineering development firms.
    - Limited resources for tools and technology.
    - Short-term project focus that tends forever to encourage the exploration of a better design and development methodology "on the next project."
  - The inescapable engineering trade-offs.
    - Unit system costs must be minimized to make a product successful in the market.
    - Current middleware has focused on portability and platform independence with very little attention paid to performance and resource requirements.
    - Power-aware computing requirements also conflict with current middleware.
  - Embedded middleware must evolve to address these challenges.
    - Once the engineering trade-offs are addressed favorably, then adoption will grow as the cultural barriers fall (i.e., as old engineers keel over).