Data Distribution Strategies with Fault-Tolerant Components

OMG's Workshop on Distributed Object Computing for Real-time and Embedded Systems

Richard T. McClain
richard.mcclain@lmco.com

Joseph N. Licameli
joseph.licameli@lmco.com

Lockheed Martin
Naval Electronic & Surveillance Systems
Syracuse, NY
Advanced Systems Architecture

Session Outline

- Intro
- Component Design
- Challenge
- Fault Tolerant Strategies
  - Master/Standby Component
  - Master/Standby Data Distribution
  - Master/Standby Storage with Distribution Capability
- Lessons Learned
- Wrap-Up
- Q & A

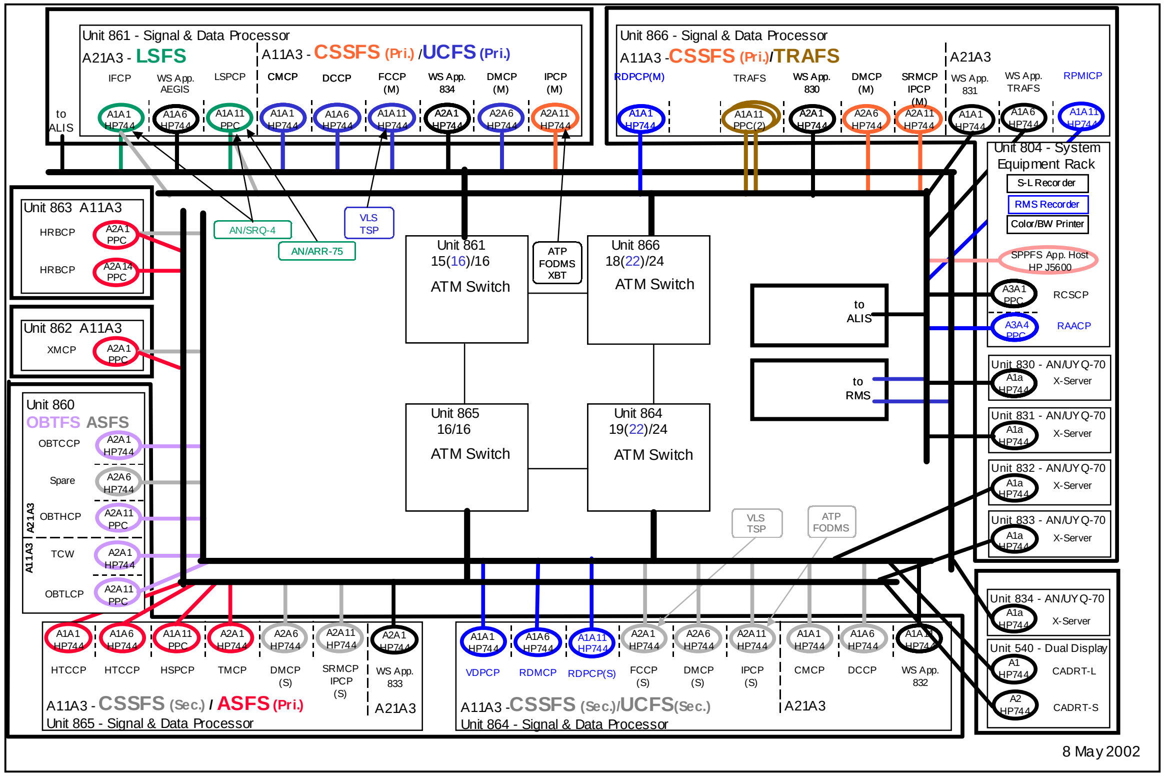
Intro

- AN/SQQ-89 is an undersea warfare (USW) system for Aegis destroyers
- Multiple variants exist
- COTS Incorporation
  - Development began in 1990's
  - Design Constraints
    - Open System Architecture
    - Distributed, real-time and embedded (DRE)
    - VxWorks & HP-UX
    - C++, Ada-95 and Java
    - Combinations of legacy and new software

OMG CORBA chosen as middleware for communications among different software entities

AN/SQQ-89 USW Combat System

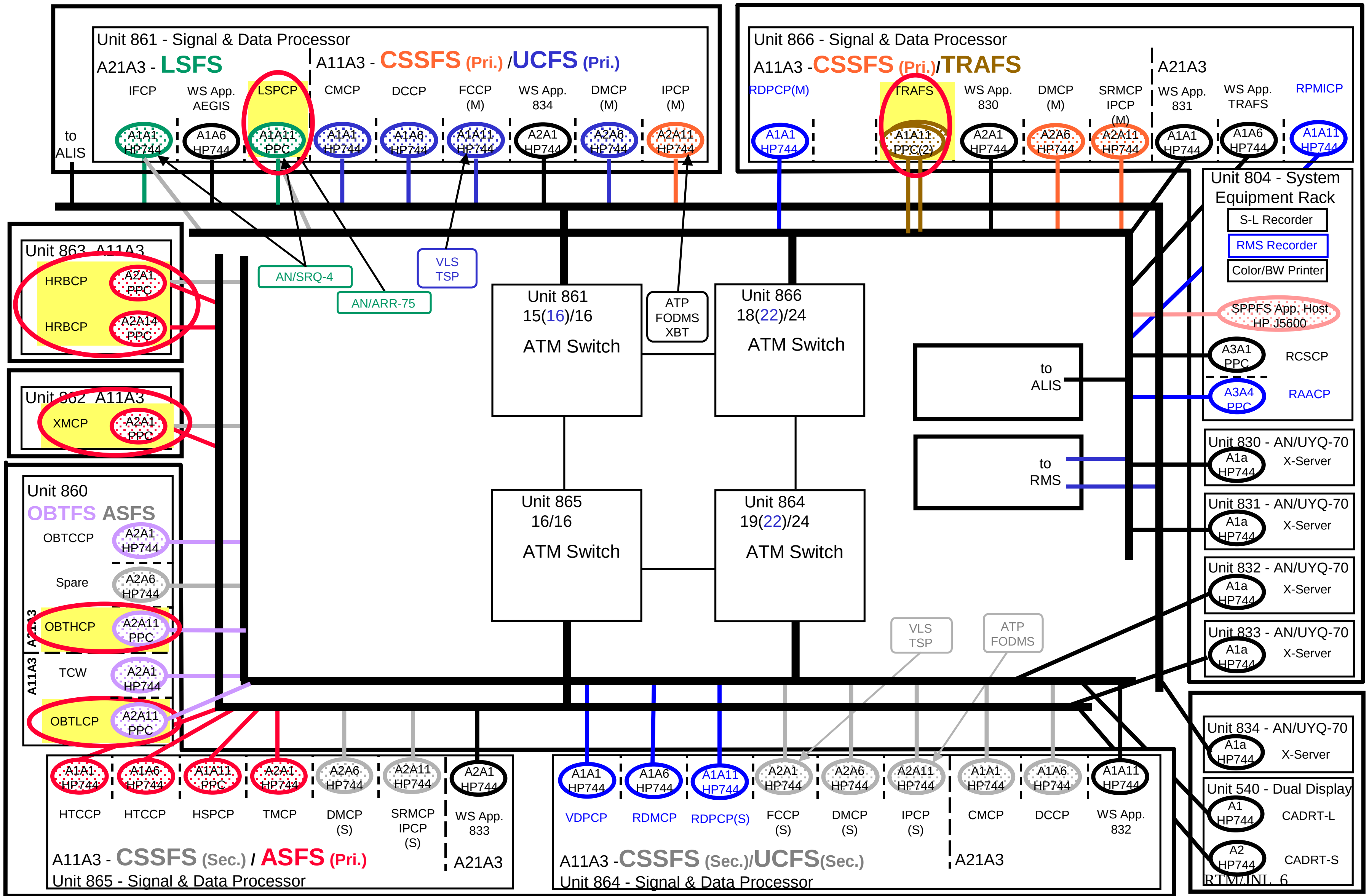
- Full COTS Implementation
- Network centric utilizing Internet Protocol suite
- OMG CORBA as primary communication protocol
- 2 Million Software Lines of Code - C++, Ada-95 and Java.



Real-Time - Hard vs Soft

- Hard Real-Time
  - Used Motorola PPC running VxWorks
    - Mercury Quad PPC for Digital Signal Processing (DSP)
  - Processing localized to node; avoided network interference
  - Inter-process communications via UDP
    - Connectionless transmission and low latency
    - Used CORBA as a means for senders to setup channels to receivers
- Soft Real-Time
  - Used HP744 running HP-UX 11.0
  - Used Motorola PPC running VxWorks
    - Non real-time string
  - Inter-process communications via CORBA

Hard Real-Time Processing Nodes

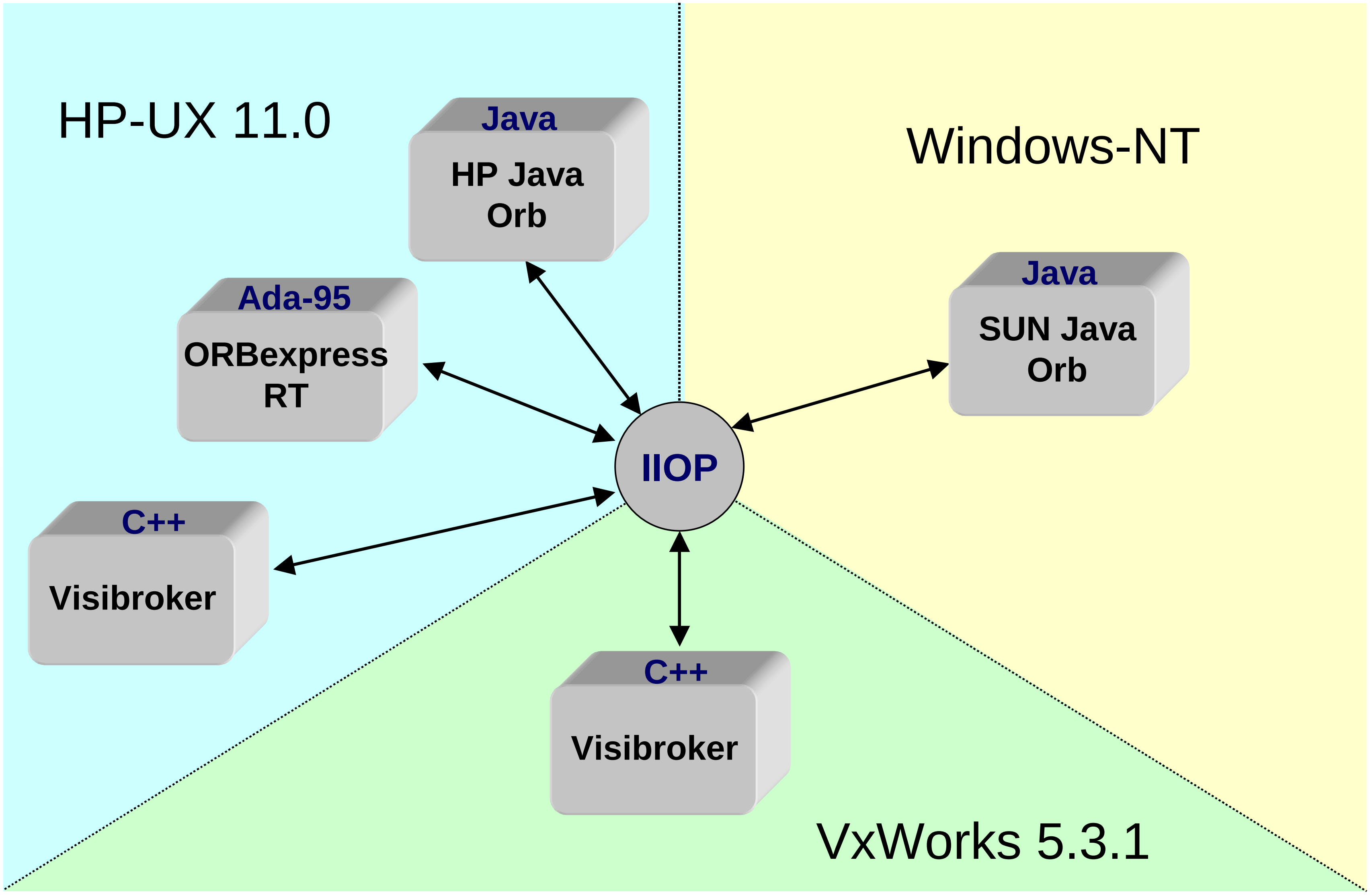


Tolerant Components

Applied Elements of CORBA

- Naming Service
  - Developed Fault Tolerant Scheme
- Static Invocations
  - Did not utilize Dynamic CORBA
- Basic Object Adapter (BOA)
  - POA not yet adopted
- Publish-Subscribe Semantics for Data Distribution
  - Event Service Not Readily Available
    - Immature Implementations
    - Some products relied on Multicast

Interoperability

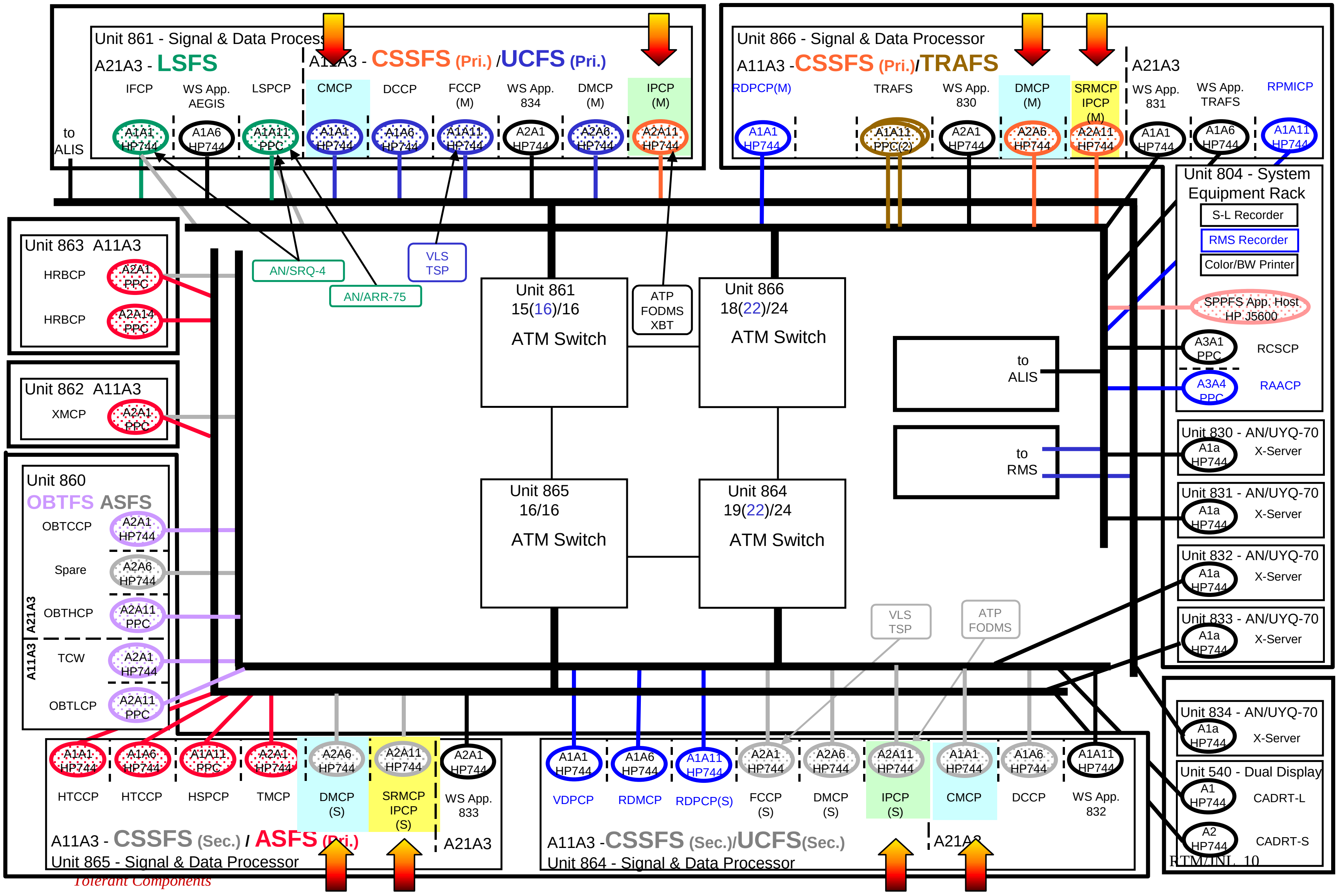


Component Development

- Why Components?
  - IDL does not define all of the requirements placed upon an interface.
    - Fault Tolerant Requirements
    - Communication Maintenance
    - Failure Events
  - Multiple IDL users leads to multiple solutions to meet requirements
    - Leads to multiple problems that are unique
- Examples of Common System Services (CSS) Components with Fault Tolerant Requirements
  - System Resource Management
  - Ownship Data Management
  - Database Management

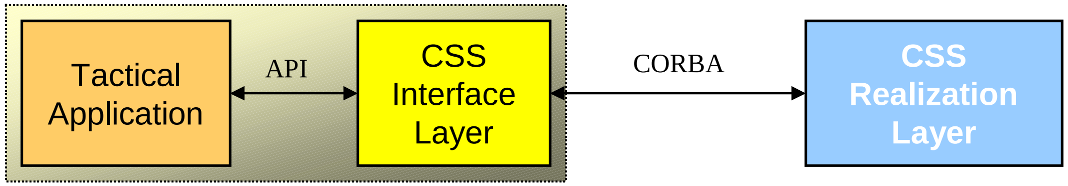
Components allow for a common mechanism to meet system requirements

CSS Components



CSS Component Design

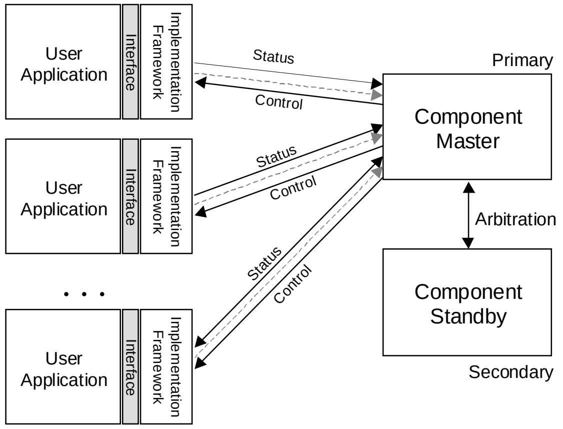
- CSS consists of an interface layer and a realization layer
- Interface layer provides a set of services commonly required by tactical applications in their native language
  - For example: Alerts, Time, Ownship and Database
- Realization layer contains a set of executables that collaborate to provide an implementation to the services offered at the interface layer
  - For example: Ownship Management provides the implementation to the Ownship interface.



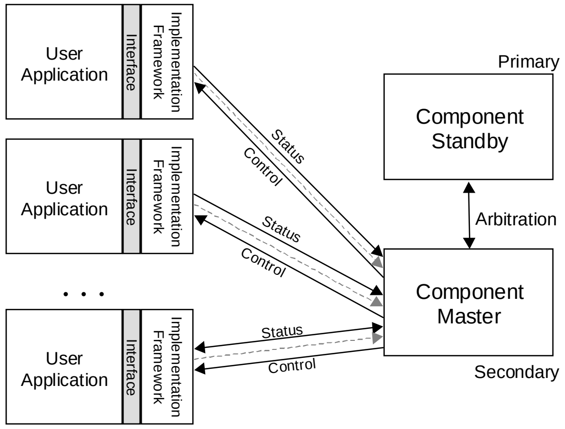
Fault Tolerant Challenge & Strategies

- Challenge
  - Provide a common strategy for meeting systemic fault tolerant requirements
  - Isolate the strategies from the applications utilizing the fault tolerant components
  - Handle different levels of fault tolerant quality of service
- Strategies
  - Master/Standby
    - System Resource Management Component
  - Master/Standby Data Distribution
    - Ownship Management Component
  - Master/Standby Storage with Distribution Capability
    - Database Management Component

Master/Standby – System Resource Management



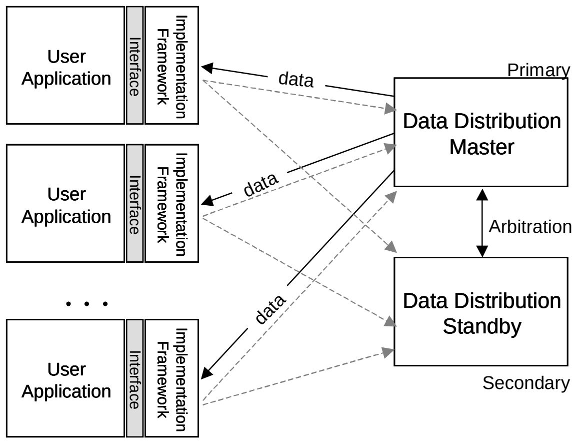
Implementation Framework connected to Primary Component running as Master



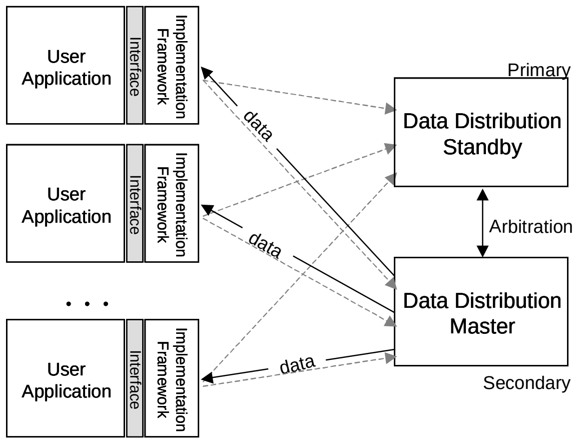
Implementation Framework connected to Secondary Component after a transition to Master

- Mechanism
  - Interface Layer maintains connection to one component site (Master)
  - Only master site publishes objects into the Naming Service
  - Upon failure standby site overwrites references in the Naming Service
    - Interface Layer reestablishes connection to new master site
  - Arbitration Exists between Primary and Secondary Sites
- Suitability
  - Bi-directional Data Flow
  - Temporary loss of data is allowed during transition
  - Data Exchanged – Posting Alerts & System State Updates

Master/Standby Data Distribution – *Ownship Management*



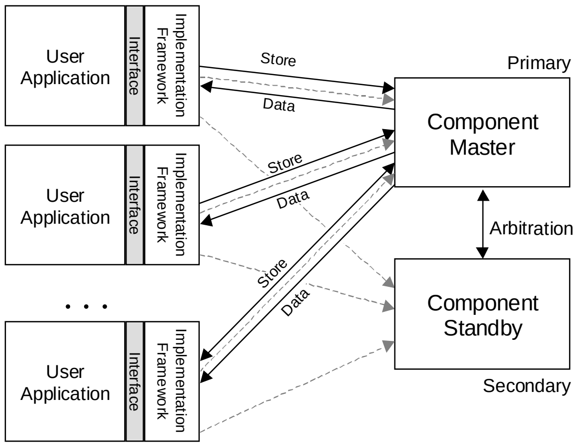
Implementation Framework connected to Primary & Secondary Component. Data distributed from Master only.



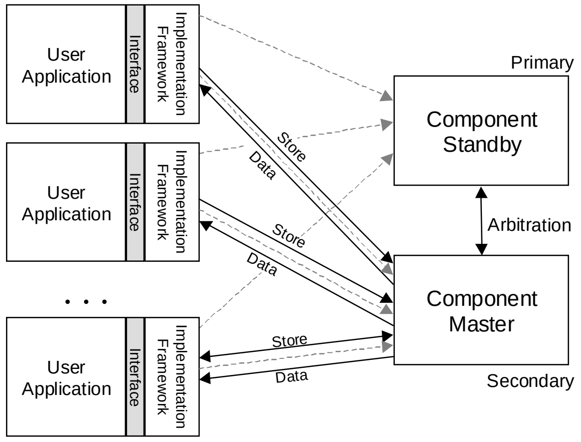
Implementation Framework connected to Primary & Secondary Component. Data distributed from Master only.

- Mechanism
  - Interface Layer maintains connection to both sites
  - Publish-Subscribe pattern utilized
  - Arbitration exists between sites
    - Only master site distributes data
  - Hides multiple distribution sites from applications and provides uninterrupted flow of data
- Suitability
  - Unidirectional Data Flow (Pure Data Sources)
  - Data Exchanged – Ship Positional Data

Master/Standby Storage with Distribution Capability – Database Management



Implementation Framework connected to Primary & Secondary Component. Bi-directional interface with Master only.



Implementation Framework connected to Secondary & Primary Component. During transition, framework updated to communicate with new Master.

- Mechanism
  - Interface Layer maintains connection to both sites
  - Arbitration exists between sites
    - Only master site distributes data – Publish-Subscribe pattern
    - Applications only allowed to send and retrieve from master site
    - Standby site notifies Interface Layer when transitioning to master
  - Hides multiple storage/distribution sites from applications and provides uninterrupted flow of data
- Suitability
  - Bi-directional Data Flow where information is stored, retrieved and distributed
  - Loss of Data Not Tolerated
  - Data Exchanged – System Information, Persistent Information, …

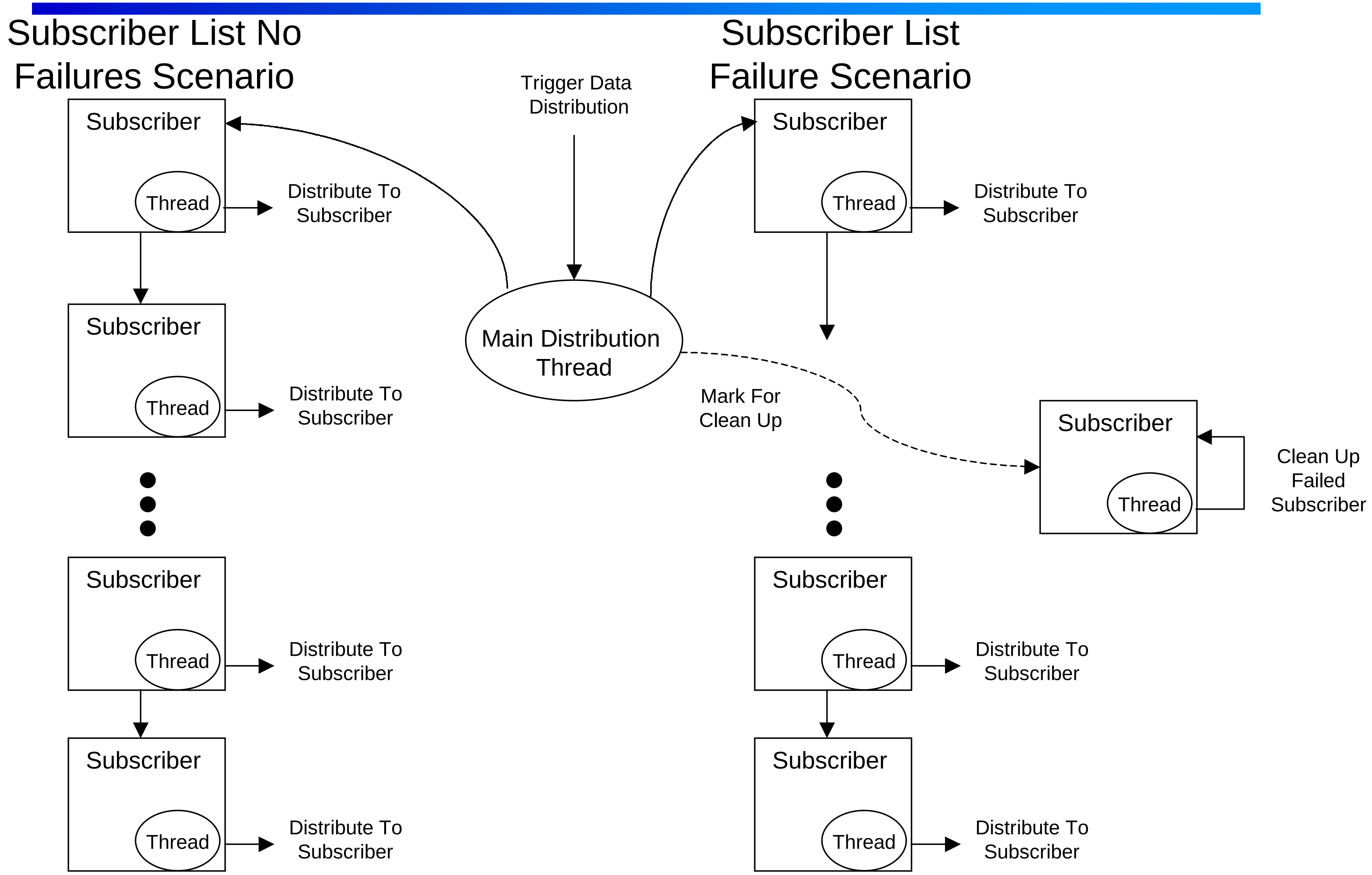
Lessons Learned

- Implementation of Publish-Subscribe patterns
  - Obtaining a Key when Registering with a Data Server
    - Subscribers need to pass a unique key to the Data Servers
    - Data Servers providing the key to the subscriber can cause performance issues
  - Clean Up of Failed Subscriber References
    - Destroy Failed References in a separate thread
    - QoS of underlying TCP socket closure can block thread of execution
  - Data Servers Provide Tolerance on Subscriber Reference Failures
    - Intermittent failures can cause the break down of otherwise valid communication channel
    - Do not generically process exceptions

Lessons Learned -- Observations with Subscription Failures

- CORBA Setup
  - Oneway invocations for distributing subscription data
  - Original Assumption – Oneway Invocations will not block
- Observations on TCP socket in flow control
  - Oneway invocations blocked thread
  - Eventually No Response exception thrown
- Observations on Kernel Crashes and TCP sockets
  - Mainly Observed with VxWorks nodes which needed a reset to restart applications
  - Oneway invocations blocked thread
  - Eventually No Response exception thrown
- Summary
  - One subscriber effected distribution to all other subscribers
- Solution
  - Distribution with Active Objects

Lessons Learned -- Distribution with Active Objects


Subscriber List No Failures Scenario
Subscriber List Failure Scenario
Trigger Data Distribution
Main Distribution Thread
Subscriber
Thread
Distribute To Subscriber
Mark For Clean Up
Clean Up Failed Subscriber

Lessons Learned

- Real-Time Implications
  - Handling Failures
    - Minimize effect of a failed subscriber on the data flow to other system subscribers
    - Build in a tolerance for intermittent failures on subscribers
      - Prevents time consuming recovery actions
  - Performance
    - Simplify the subscription process to minimize the processing involved with removal and additions to the distribution list.

Wrap-Up

- IDL alone is not sufficient for defining all of the requirements of an interface
  - Fault tolerant requirements are not apparent from the IDL
- Component Development provides a common mechanism for meeting these interface requirements
- Isolate communication failure events from the main thread of execution
  - Do not allow the handling of a failure on a subscriber reference to interrupt the flow of data to other system subscribers
- SQQ-89 successfully deployed real-time embedded system that used CORBA as the system middleware

Questions & Answers