The SAE Architecture Analysis & Design Language (AADL)

www.aadl.info

Joyce L Tokar, PhD
Pyrrhus Software
Phoenix, AZ
tokar@pyrrhusoft.com
480-951-1019

Tutorial Objectives

- *Provide* an overview of the SAE AADL Standard.
- *Introduce* architecture-based development concepts.
- *Provide* a summary of AADL capabilities.
- *Demonstrate* the benefits of AADL in real-time systems design.
- *Provide* an overview of the AADL development environment.

Tutorial Outline

- **Background & Introduction**
  - Motivation
  - Model Based Engineering
  - Standardization Process
- AADL – The Language
- AADL Development Environment
- AADL in Use
- Summary and Conclusions

Systems Development Concerns

- Incomplete capture of specification and design.
- Little insight into behavioral system properties until system integration & test
  - Performance (e.g., Throughput, Quality of Service)
  - Safety
  - Reliability
  - Time Critical
  - Security
  - Schedulability
  - Fault Tolerance
- System Integration – high risk
- Evolvability – very expensive
- Life Cycle Support – very expensive
- Leads to rapidly outdated components

Systems Engineering

- Requires integrated hardware and software modeling
- There are many dimensions to consider
- Requires an integrated process to understand effects
- Changes occur throughout the development lifecycle
- Changes occur in the fielded systems
- Integration problems are the largest single technical cause of program failure.

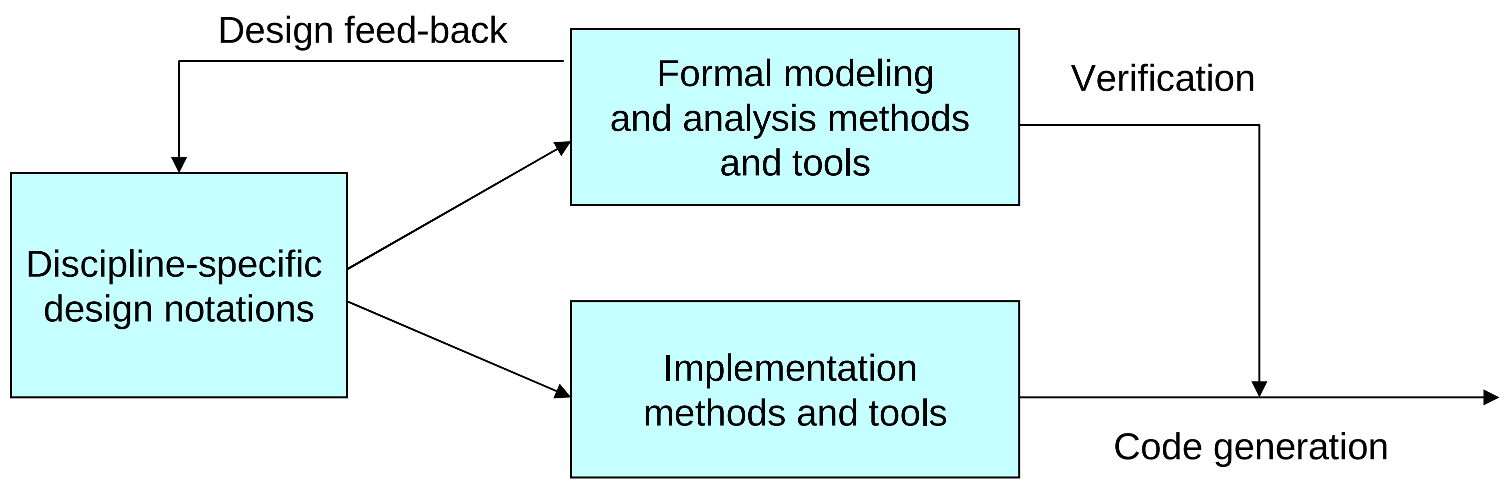
AADL is a single language that captures the multiple dimensions of systems engineering.

AADL Support Systems Engineering

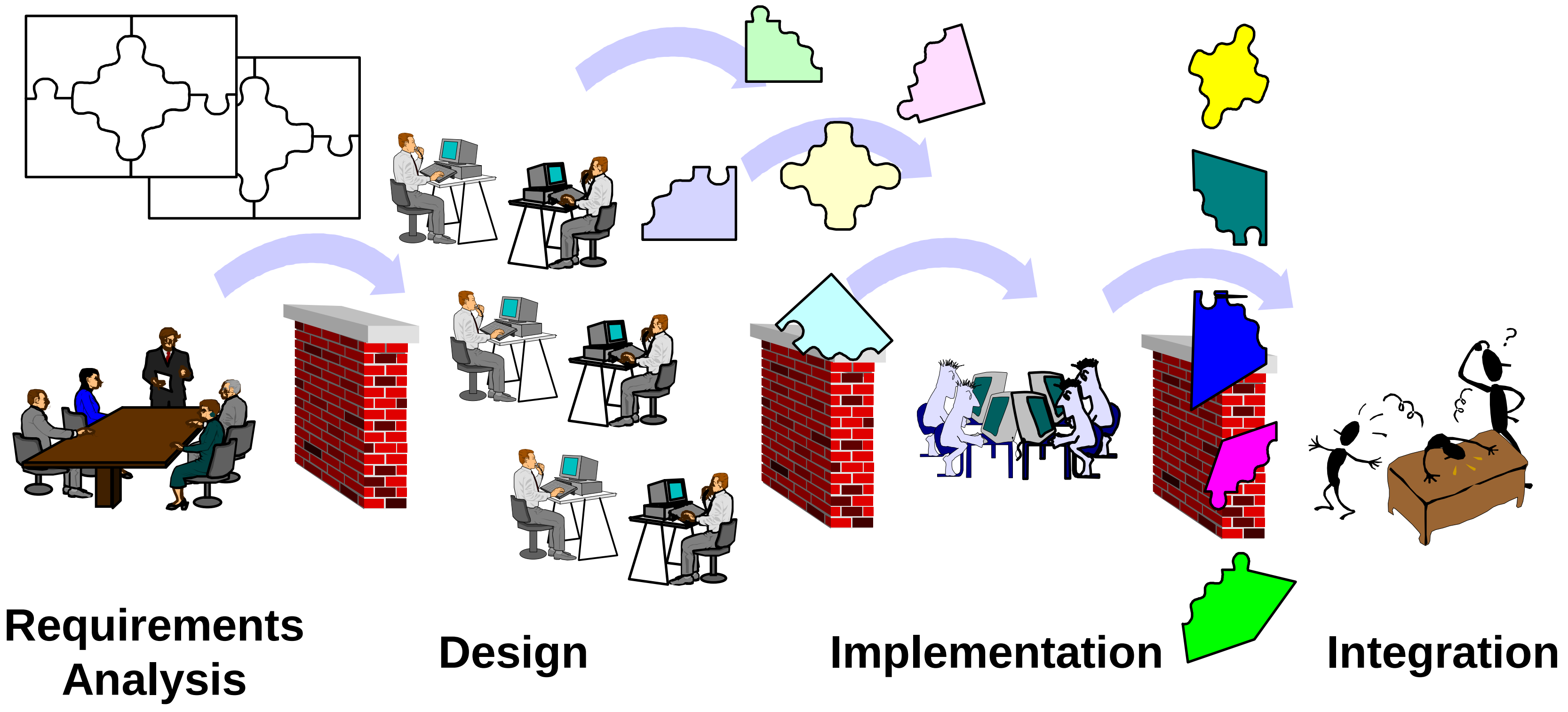
- AADL is a single language that captures the many dimensions of systems engineering leading to successful integration.
- AADL provides
  - A language for capturing the architecture
  - A capability to analyze critical system parameters
  - An infrastructure for automating the integration process

Model Based Engineering

- Repeated system analyses track development and evolution
- Auto generation, component integration to specification/analysis
- Rapid evolution through the refinement of specification/components


Design feed-back
Formal modeling and analysis methods and tools
Verification
Discipline-specific design notations
Implementation methods and tools
Code generation

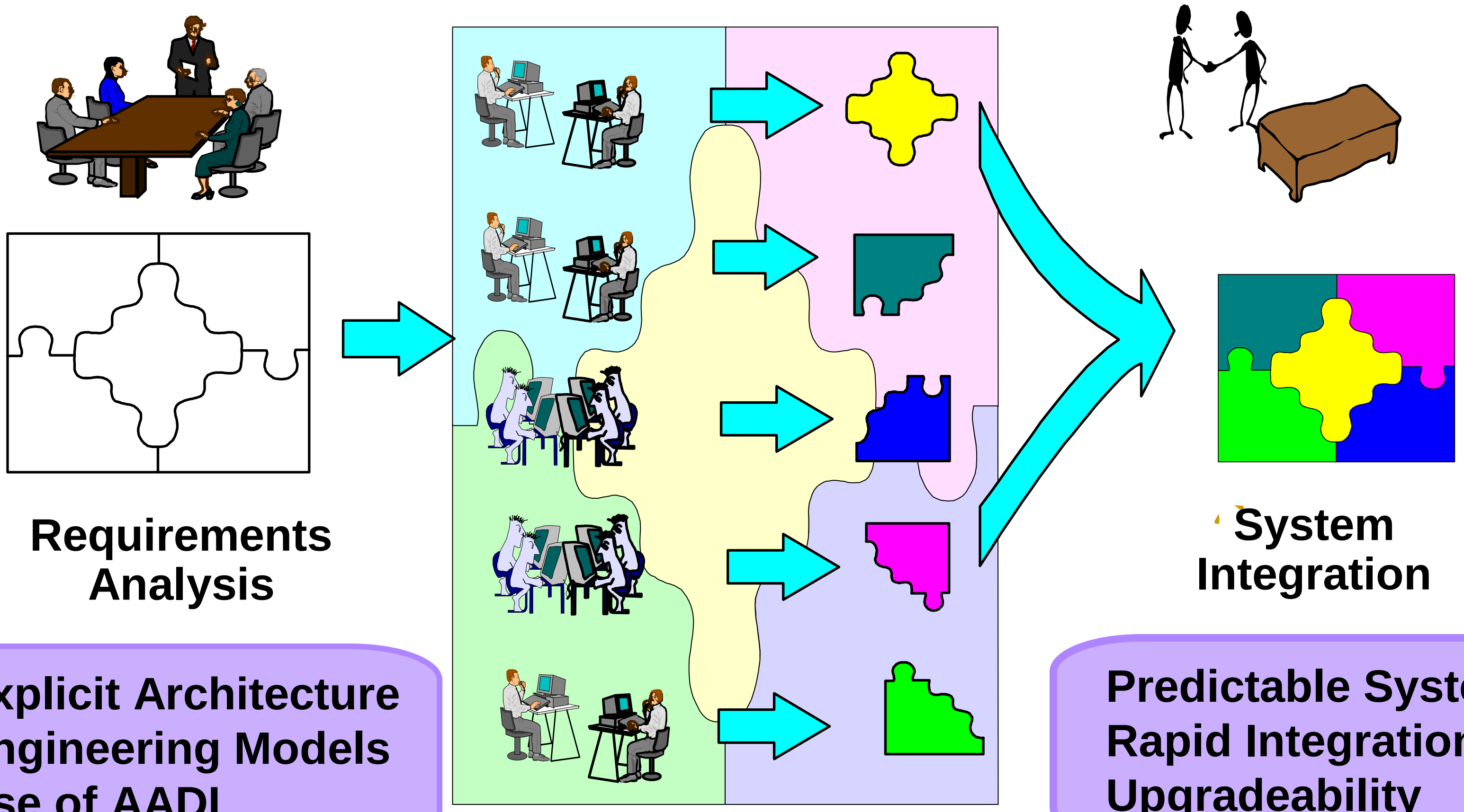
Typical Software Development Process



manual, paper intensive, error prone, resistant to change

Model-Based System Engineering

Model-Based & Architecture-Driven



Requirements Analysis

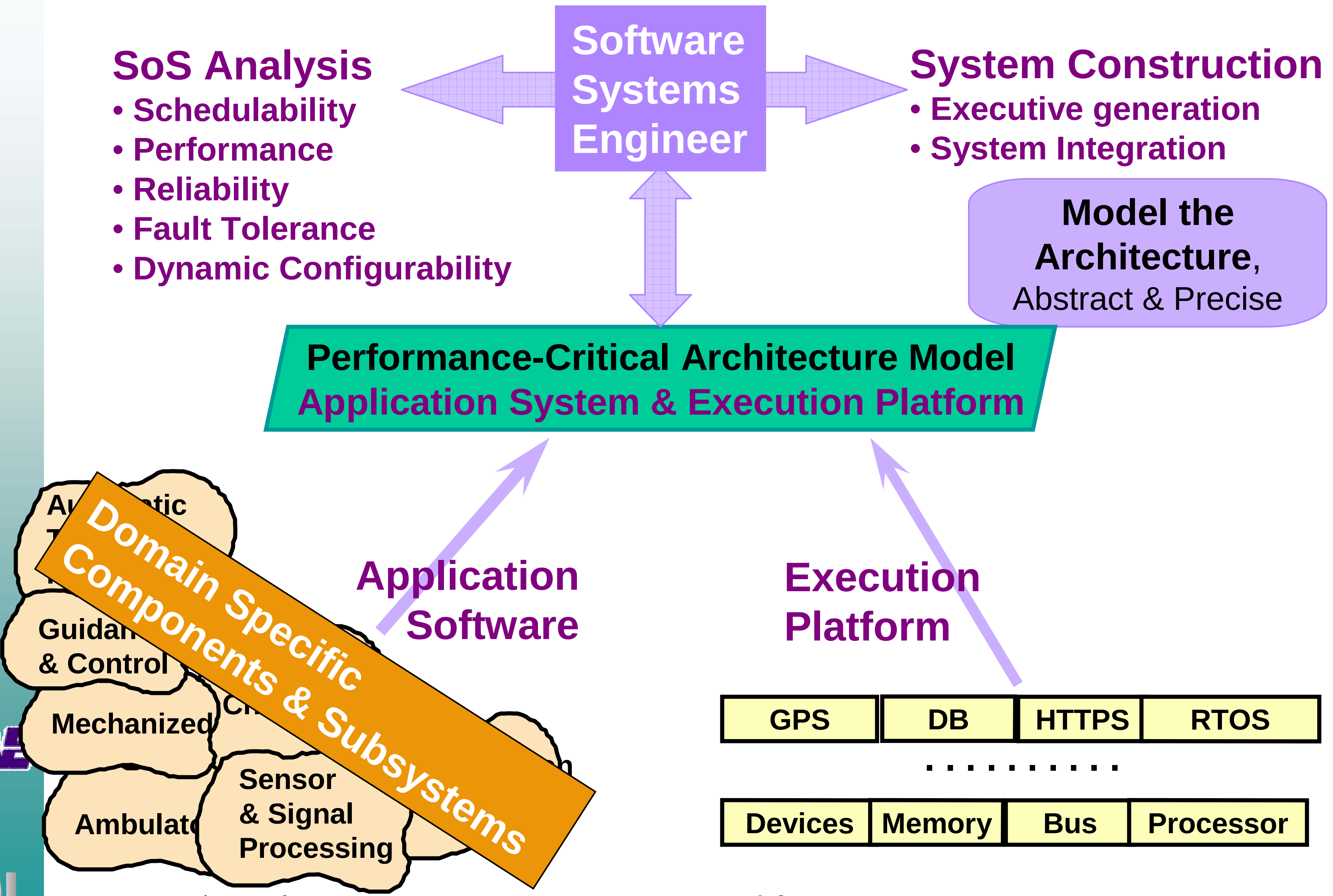
Design, Analysis and Implementation

System Integration

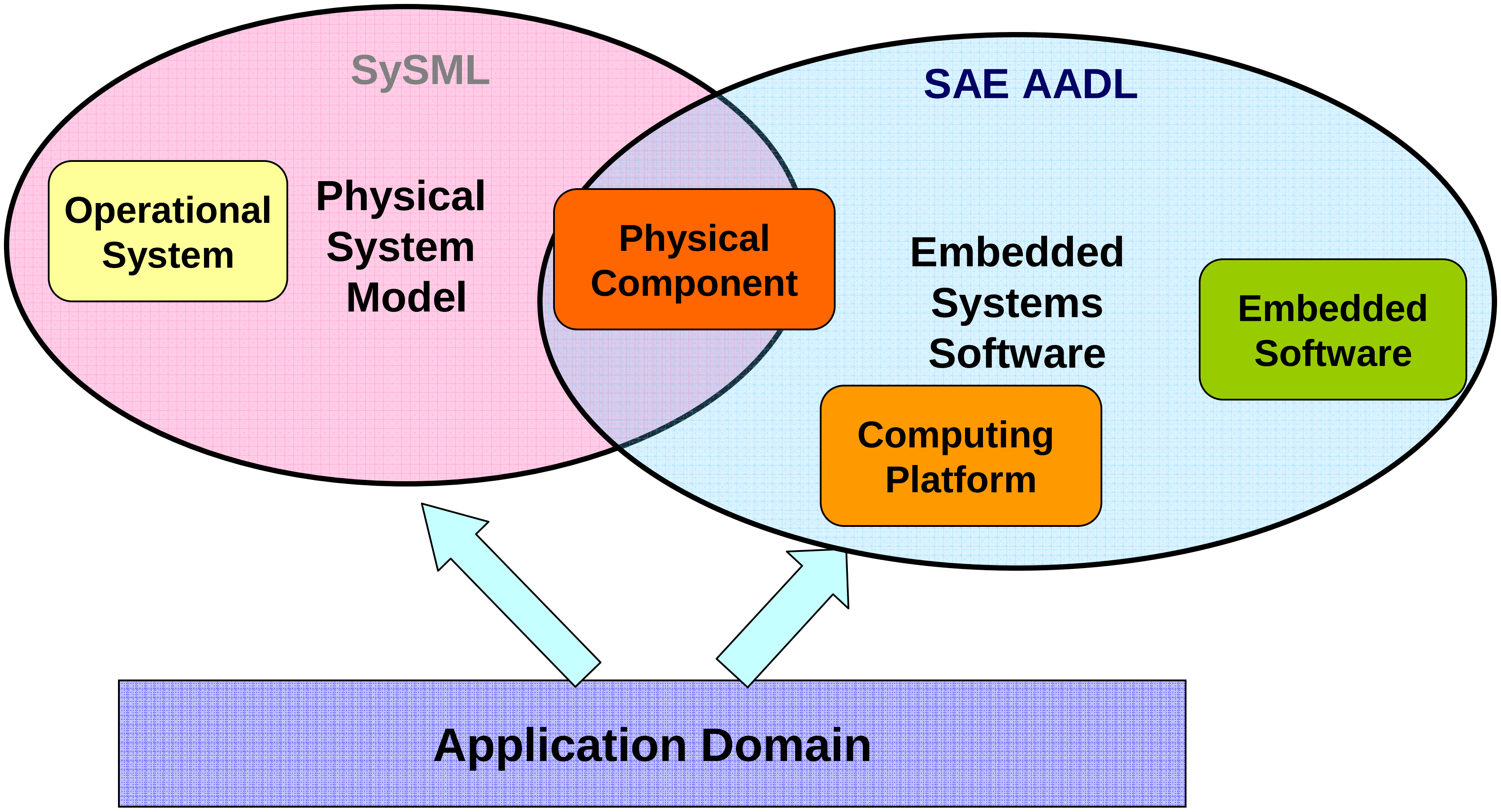
Explicit Architecture Engineering Models Use of AADL

Predictable System Rapid Integration Upgradeability

Model-Based System Engineering



Software and Systems Engineering



The SAE AADL Standard

- **Sponsored by**
  - Society of Automotive Engineers (SAE)
    - Avionics Systems Division (ASD)
      - Embedded Systems (AS2)
        - Avionics Architecture Description Language Subcommittee (AS2C)
        - Bruce Lewis – Chairman bruce.a.lewis@us.army.mil
- **Status**
  - Requirements document SAE ARD 5296 – balloted & approved 2000
  - Standard document SAE AS 5506 – balloted & approved 2004.
  - Annex documents
    - Balloted & Approved July 2005
      - Graphical Annex
      - XML Annex
      - Programming Language Annex
    - Balloted & Approved Nov 2005
      - Error Annex
    - To be balloted
      - UML Annex
- **Contact**

  http://www.aadl.info email: info@aadl.info

SAE AS-2C ADL Subcommittee

- **Key Players:**
  - Bruce Lewis (AMCOM): Chair, technology user
  - Steve Vestal (Honeywell): MetaH originator, co-author
  - Peter Feiler (SEI): Technical lead, author, co-editor, technology user
  - Ed Colbert (USC): AADL & UML Mapping
  - Joyce Tokar (Pyrrhus Software): Programming Language Annex, co-editor; secretary
- **Members:**
  - Boeing, Rockwell, Honeywell, Lockheed Martin, Raytheon, Smith Industries, Airbus, Axlog, Dassault, EADS, High Integrity Solutions
  - NAVAir, Open Systems JTF, British MOD, US Army
  - European Space Agency
- **Coordination with:**
  - NATO, ESA, COTRE, OMG-UML&SysML, SAE AS-1 Rapid Weapon Integration

SAE Architecture & Analysis Design Language AADL

- The AADL is a language for the specification of systems that are:
  - Real-time
  - Embedded
  - Fault-tolerant
  - Secure
  - Safety-critical
  - Modal & dynamically configurable
- The AADL is a language that enables the specification of software task and communication architectures
- Which may be bound to
  - Distributed multiple processor hardware architectures
- Fields of application include
  - Avionics, Aerospace, Automotive, Autonomous systems, …

The SAE AADL Standard

- Provides a standard & precise way to describe the architecture of embedded computer systems.
- Provides a standard way to describe components, assemblies of components, and interfaces to components.
- Describes how components are composed together to form complete system architectures.
- Describes the runtime semantics and thread scheduling protocols.
- Describes the mechanisms to exchange control and data between components.
- Describes dynamic run-time configurations.

Tutorial Outline

- Background & Introduction
- **AADL – The Language**
  - Overview of ADLs
  - AADL Syntax & Semantics
- AADL Development Environment
- AADL in Use
- Summary and Conclusions

What is Systems Architecture

- Architecture is the fundamental organization of a system as embodied in
  - its components,
  - their relationships to each other and the environment,
  - the principles governing its design and evolution.

- The architecture of a program or computing system is
  - the structure or structural arrangements of its composite elements, both hardware and software,
  - the externally visible properties of those elements,
  - the relationships among them.

Architecture is the foundation of good software & systems engineering

What is an Architecture Description Language (ADL)

- The ***architecture*** of a system defines its high-level structure and exposes its gross organization as a collection of interacting components.
- An ***Architecture Description Language (ADL)*** focuses on the high-level structure of the overall application rather than on the implementation details of any specific component.
- ADLs and their accompanying toolsets support architecture-based development, formal modeling, and analysis of architectural specifications.
- The ***AADL*** is an architecture description language that includes support for the inclusion of both the software components and the execution platform components in the system architectural specification.

Architecture Description Languages

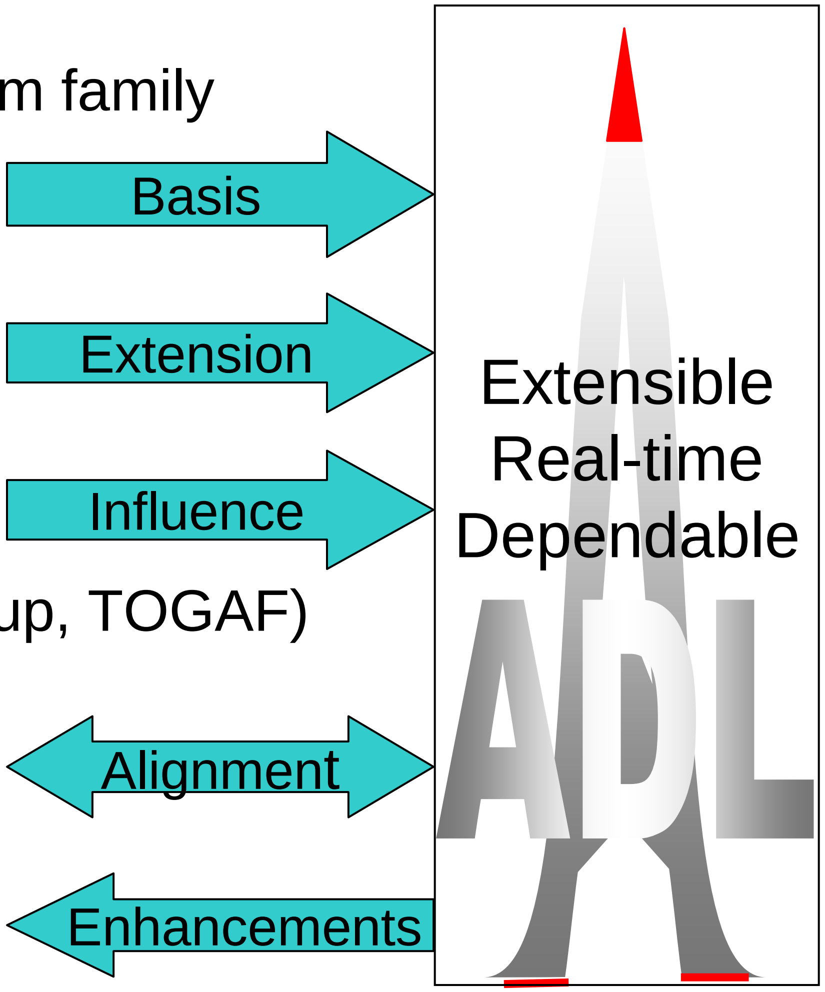
From Research to Standards

Research ADLs

- MetaH
  - Real-time, modal, system family
  - Analysis & generation
  - RMA based scheduling
- Rapide, Wright, ..
  - Behavioral validation
- ADL Interchange
  - ACME, xADL
  - ADML (MCC/Open Group, TOGAF)

Industrial Strength

- HOOD/STOOD
- SDL
- UML 2.0, UML-RT



Common Foundation of ADLs

- ***Components*** represent the primary (computational) elements and data stores of a system.
- ***AADL Components****:*
  - *Software Components:*
    - Data, subprogram, thread, thread groups process.
  - *Execution Environment Components:*
    - Memory, bus, device, processor
  - *Composite Components:*
    - System
- ***AADL Component Interfaces****:*
  - *Shared Data*
  - *Ports*
    - Data ports, event ports, event data ports.
  - *Subprogram Parameters*

"Acme: Architectural Description of Component-Based Systems" by Garlan, Monroe, and Wile

Common Foundation of ADLs

- *Connectors* represent interactions among components. Connectors mediate the communication and coordination activities among components
- *Connectors in AADL:*
  - *Subprogram Call Sequence*
  - *Connections – physical linkage*
    - Port connections, access connections
  - *Flows – logical flow*

Common Foundation of ADLs

- *Systems* represent configurations of components and connectors.
- *Systems in AADL*:
  - *Packages*
  - *Systems*

Common Foundation of ADLs

- *Properties* represent semantic information about a system and its components that goes beyond structure.
- *AADL supplies*:
  - *Predefined Properties*
  - *User-defined Property Sets*

Common Foundation of ADLs

- *Constraints* represent claims about an architectural design that should remain true even as it evolves over time.
- *Constraints in AADL:*
  - *Hybrid automata assertions*
  - *Property value constraints*
  - *Annexes*

Common Foundation of ADLs

- *Styles* represent families of related systems.
- *Styles in AADL:*
  - *Multiple implementations – families*
  - *Refinement*
  - *Packages*
  - *Property sets*
  - *Annexes*

Common Foundation of ADLs

- **Components**

  AADL components represent the elements of a system including software elements and execution platform elements.
- **Connectors**

  Shared data, ports, and parameters represent the interfaces between components. Components are linked together using subprogram call sequences and connections. Logical data connections are represented using flows.
- **Systems**

  Systems may be represented as collections of components, hierarchies of components, or systems of systems.
- **Properties**

  Properties represent both functional and non-functional characteristics of an architecture. Property values are intrinsically important to model analysis tools as well as code generation and system generation.
- **Constraints**

  Properties may be used to represent some of the design constraints of an architecture. Constant property values may be used to represent invariants on a design. Users may build property sets to define properties to be used to constrain a particular system or family of systems even further.
- **Styles**

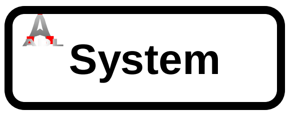
  A component may have multiple implementations to supply support for families of components.

AADL Component-Based Architecture

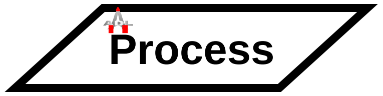
- Specifies a well-formed interface.
- All external interaction points defined as features.
- Multiple implementations per component type.
- Properties as specified component characteristics.
- Components organized in nested hierarchy.
- Component interaction declarations must follow system hierarchy.

AADL Components

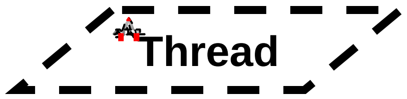
- Composite Components
  - System



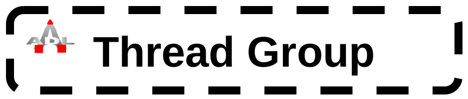
- Software Components
  - Process



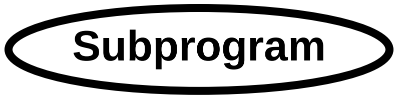
  - Thread



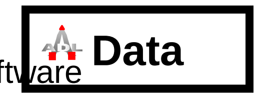
  - Thread Group



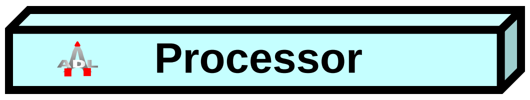
  - Subprogram



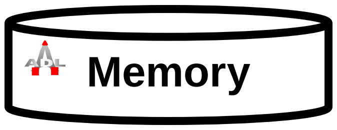
  - Data



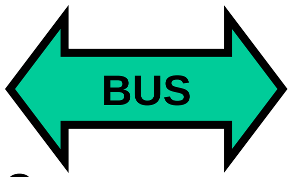
- Execution Platform Components
  - Processor



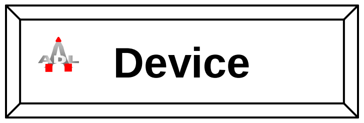
  - Memory



  - Bus



  - Device

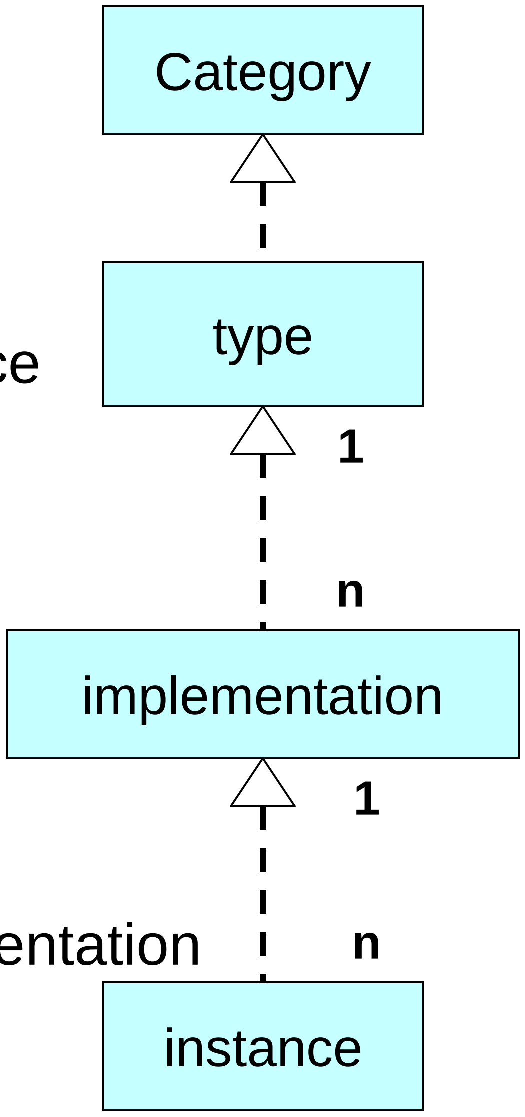


AADL Concepts Overview

- Hierarchically composed & interconnected components: ***system.***
- Application system components: ***thread***, ***thread group***, ***process***, ***data***, and ***subprogram***.
- Execution platform components: ***processor***, ***memory***, ***bus***, and ***device.***
- Interaction in terms of directional flow via *ports*, call/return, and data sharing.
- Flow of signals/state (data), control (event & time triggered), and messages (event data) across multiple components.
- Dynamic behavior in terms of statically known alternate task & communication configurations (modes).
- Core AADL and Annex extensions.

AADL Levels of Description

- Category (predefined)
- Type
  - Specification of the external interface
  - *Promised set of properties*
- Implementation
  - Specification of the content
  - *type*
- Instance
  - Instantiation of a type or an implementation
  - *variable*

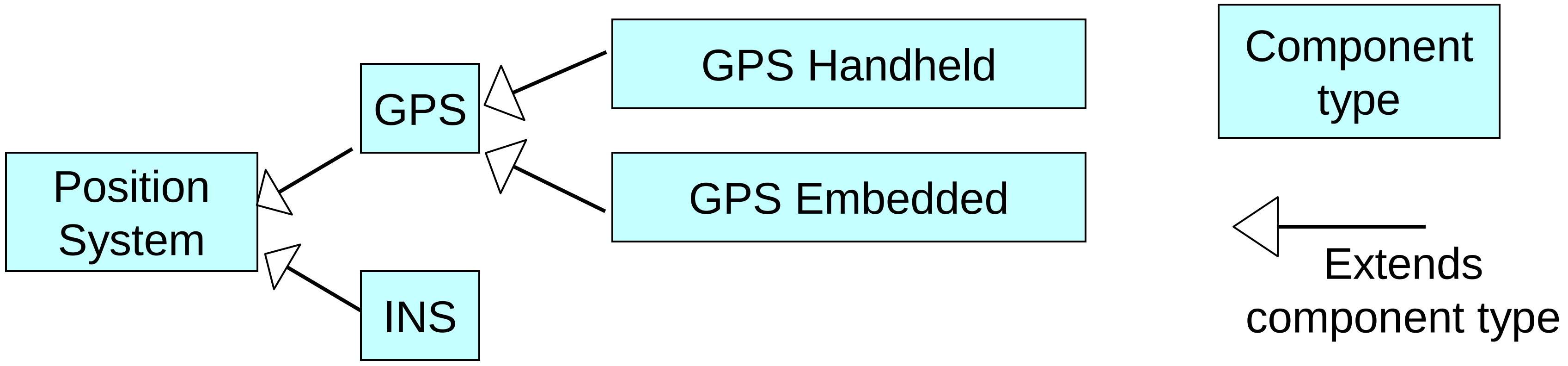


AADL Components

- Component Type -- specifies the interface to the component.
  - Features
  - Flow specifications
  - Property associations
- Component Implementation -- zero or more specifications of the component’s internal representation.

AADL Component Type Extension Hierarchy

- Component types can be declared in terms of other component types, → a component type can *extend* another component type, inheriting its declarations and property associations.
- When a component type extends another component type, then features, flows, and property associations can be added to those already inherited.
- A component type extending another component type can also refine the declaration of inherited feature and flow declarations by more completely specifying partially declared component classifiers and by associating new values with properties.



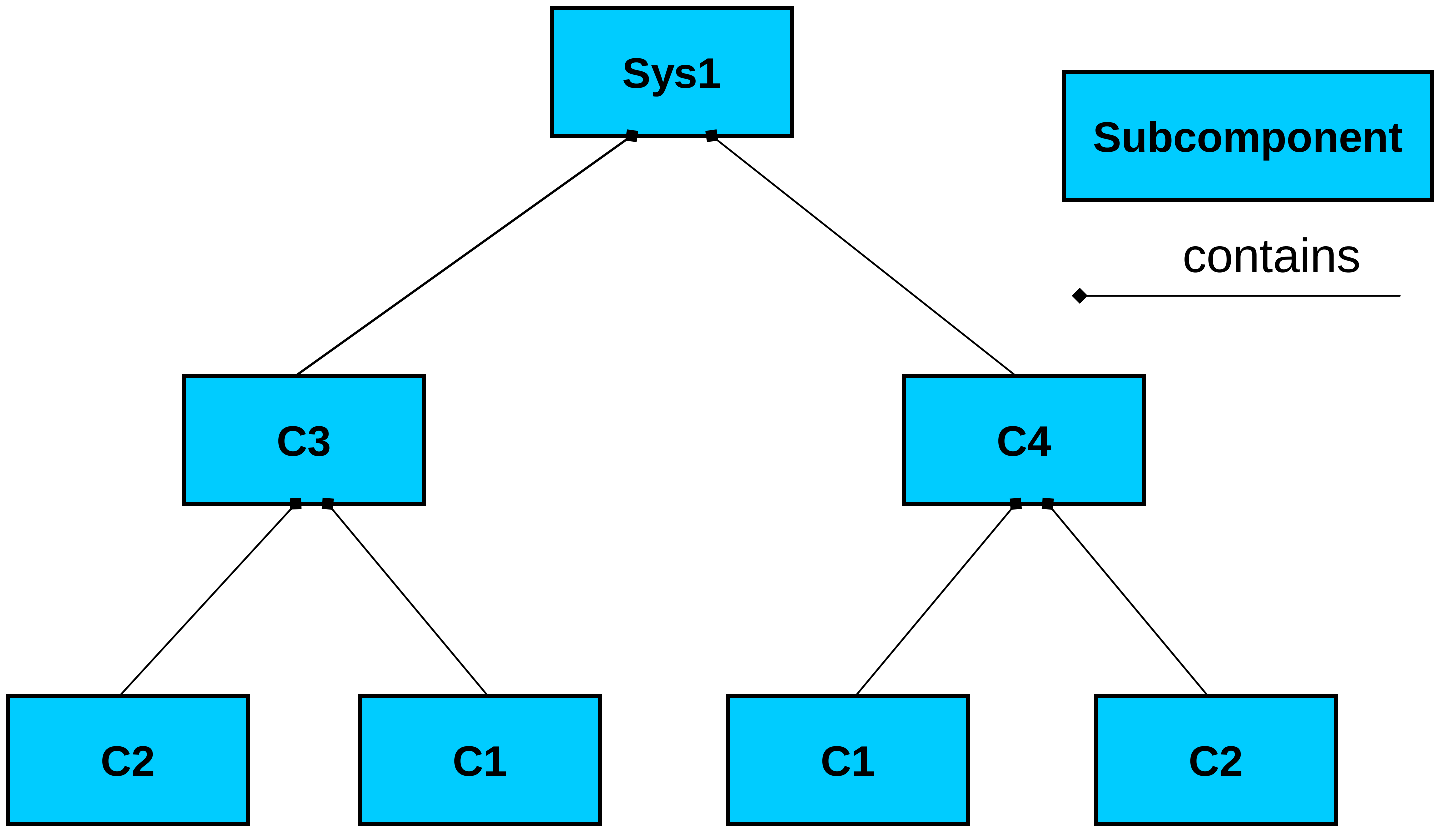
AADL Components

- Component Implementation
  - Must conform to the interface definition in the corresponding component type
  - Refines and completes the component type definition with
    - **refines type** – refines the properties of features declared in the component type
    - **subcomponents** – declaration of component parts
    - **calls** – subprogram calls
    - **connections** – declaration of physical connections between features
    - **flows** – declaration of logical flow through components
    - **modes** – declaration of operational modes
    - **properties** – property associations for the component

AADL Subcomponent

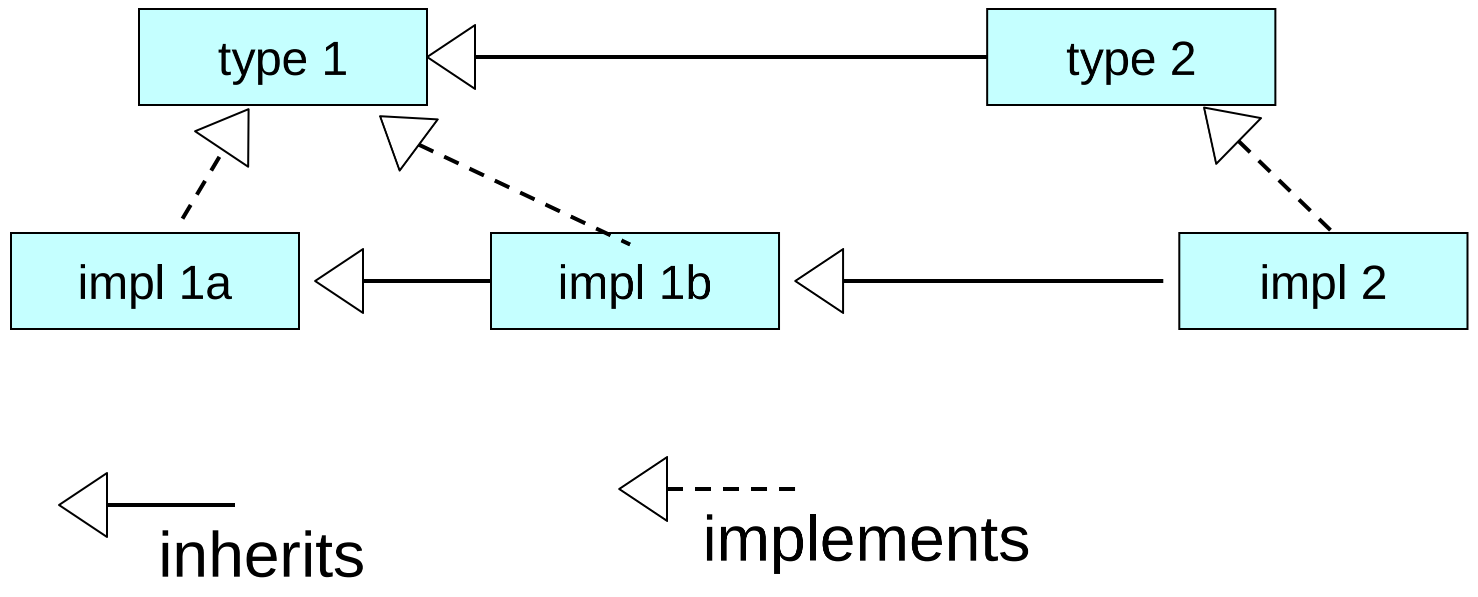
- A *subcomponent* represents a component contained within another component
- A *subcomponent declaration* may resolve required *subcomponent access* declared in the component type of the subcomponent.
- A *subcomponent* may be declared to apply to specific modes (rather than all modes) defined within the component implementation.
- *Subcomponents* can be refined as part of component implementation extensions.

AADL Component Containment Hierarchy

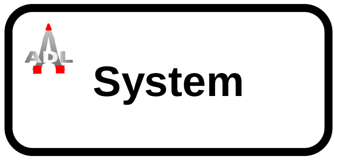


AADL Inheritance

- An AADL component type may inherit from another AADL component type.
- An AADL component implementation may inherit from another AADL component implementation.



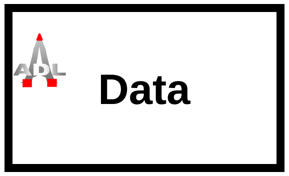
System



- Hierarchical organization of components.
- Execution platform and application software components.
- Data and bus sharable across system hierarchy.
- A *system instance* models an instance of an application system and its bindings execution platform components

Features: Port, Subprogram Provides: data access, bus access Requires: bus access, data access	Subcomponents: process, system, memory, processor, bus, device, and data Connections: yes Modes: yes

Data

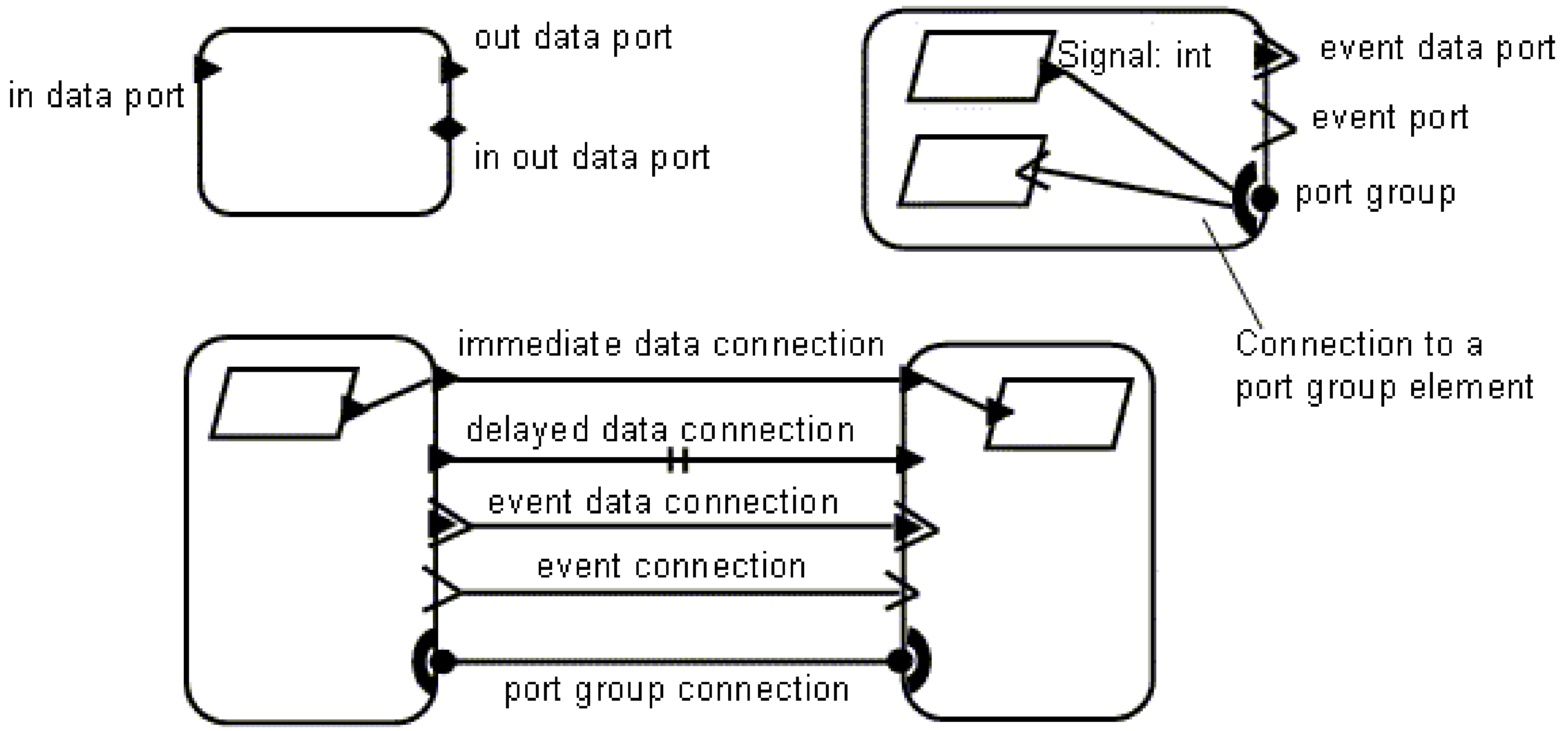


- Data component type represents data type
  - Used for typing ports
  - Optional modeling of operations
- Data component implementation
  - Substructure modeling
- Data component
  - Sharable between threads through data access connections
  - Access properties
  - Concurrency control protocol property

Features: Subprogram	Connections: access
Provides: data access	Modes: yes

AADL Interfaces & Connections

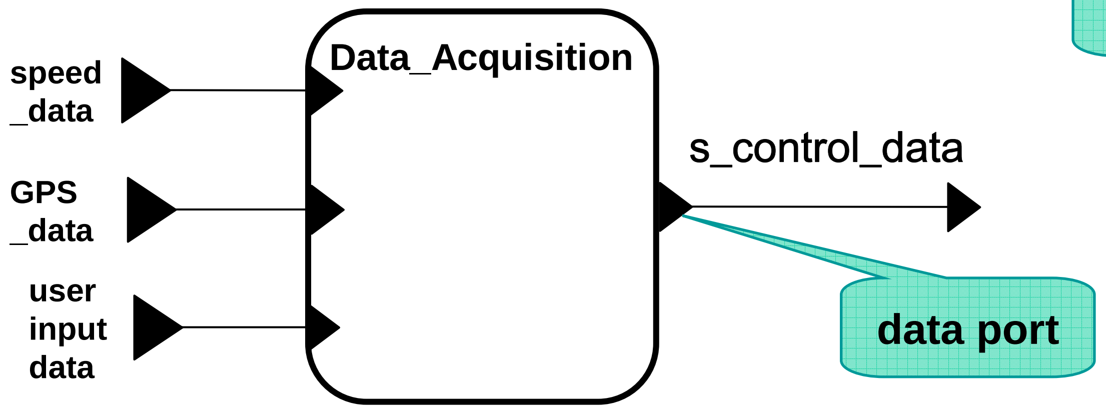
- Ports
  - Data ports
  - Event Data ports
  - Event ports
- Connections
  - Immediate
  - Delayed



Graphical & Textual Notation

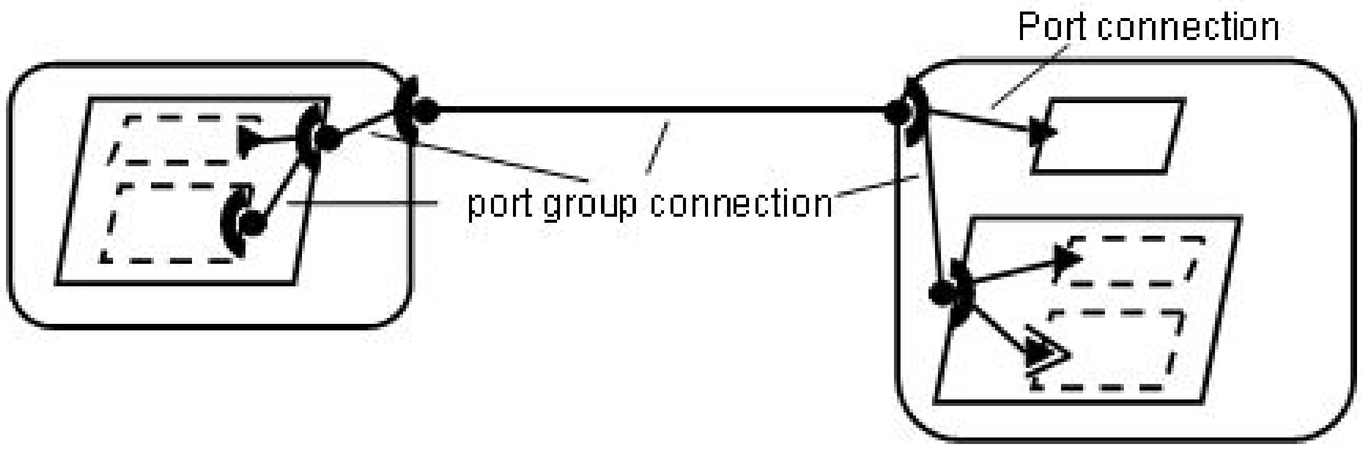
```
system Data_Acquisition
features
   speed_data: in data port metric_speed;
   GPS_data:   in data port position_carthesian;
   user_input_data: in data port user_input;
   s_control_data:  out data port state_control;
end Data_Acquisition;
```

data type of port



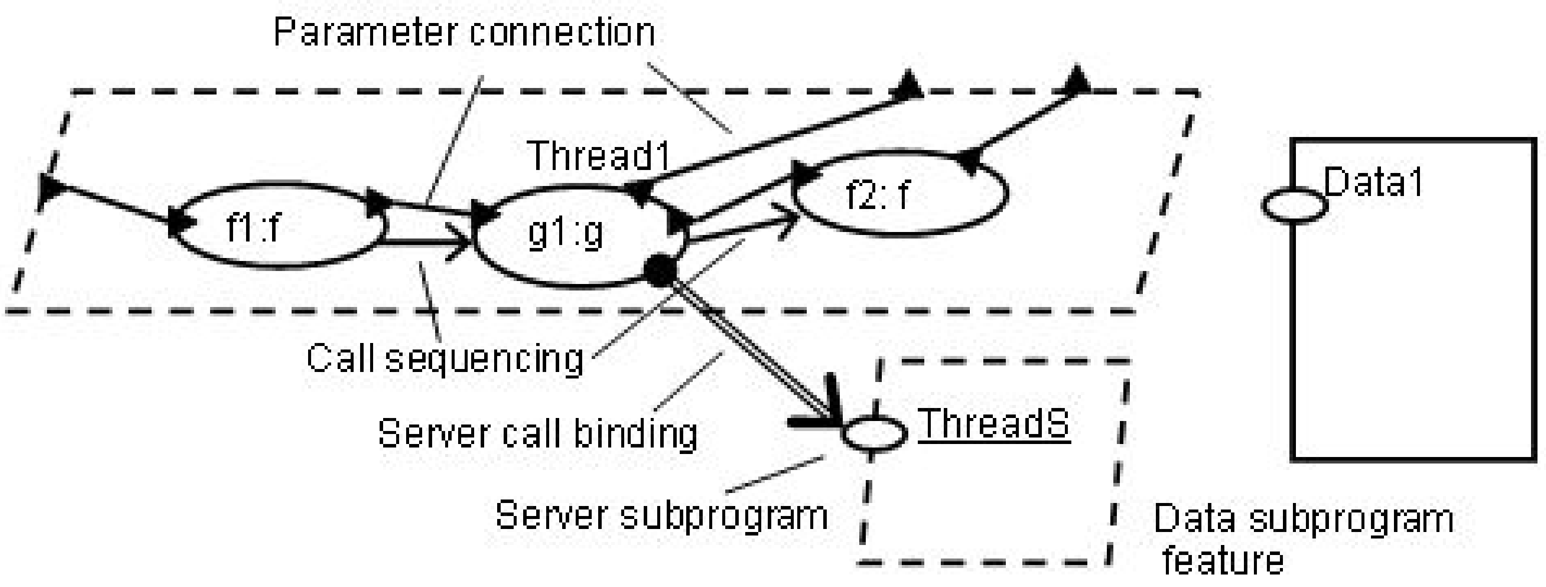
AADL Interfaces & Connections

- Port Groups
- Connections

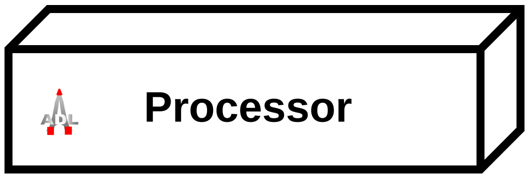


AADL Interfaces & Connections

- Subprograms
  - Local
  - Server
- Subprogram calls
  - Local
  - Remote



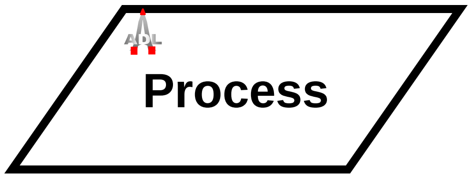
AADL Processors



- An abstraction of hardware and software that is responsible for scheduling and executing threads.
- Execute threads declared in application software systems.
- Execute threads declared in devices that can be accessed from the processor.
- May contain memories and may access memories and devices via buses.

Features: server subprogram port	Subcomponents: memory
Requires bus access	Connections: no Modes: yes

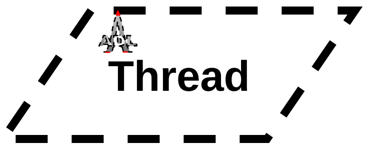
AADL Processes



- Represents a virtual address space.
- A complete implementation of a process must contain at least one thread or thread group subcomponent.

Features:	Subcomponents:
server subprogram	data
port	thread
port groups	thread group
Provides data access	Connections: yes
Requires data access	Modes: yes

AADL Threads



- Represents a sequential flow of control.
- Models a schedulable unit that transitions between various scheduling states.
- Always executes within the virtual address space of a process.
- Interacts with other threads through port connections, remote subprogram calls, and shared data access.
- Can be logically organized into thread groups.

Features: server subprogram port port groups	Subcomponents: data
Provides data access	Connections: yes
Requires data access	Modes: yes

AADL and Scheduling

- AADL provides precise dispatch & communication semantics via hybrid automata.
- AADL task & communication abstraction does not prescribe scheduling protocols
  - Cyclic executive can be supported.
- Specific scheduling protocols may require additional properties.
- Predefined properties support rate-monotonic fixed priority preemptive scheduling.

This scheduling protocol is analyzable,
requires small runtime footprint,
provides flexible runtime architecture.

AADL Thread Dispatch Protocols

5ms Periodic -- execute at given time intervals.

Aperiodic -- are triggered by an event or remote procedure call.

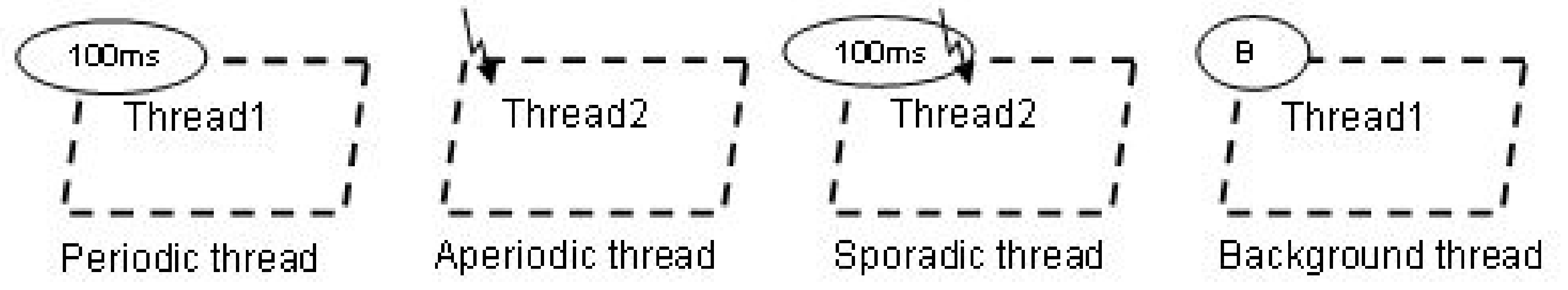
5ms Sporadic -- paced to limit execution rate.

B Background -- run when there is available processor time.

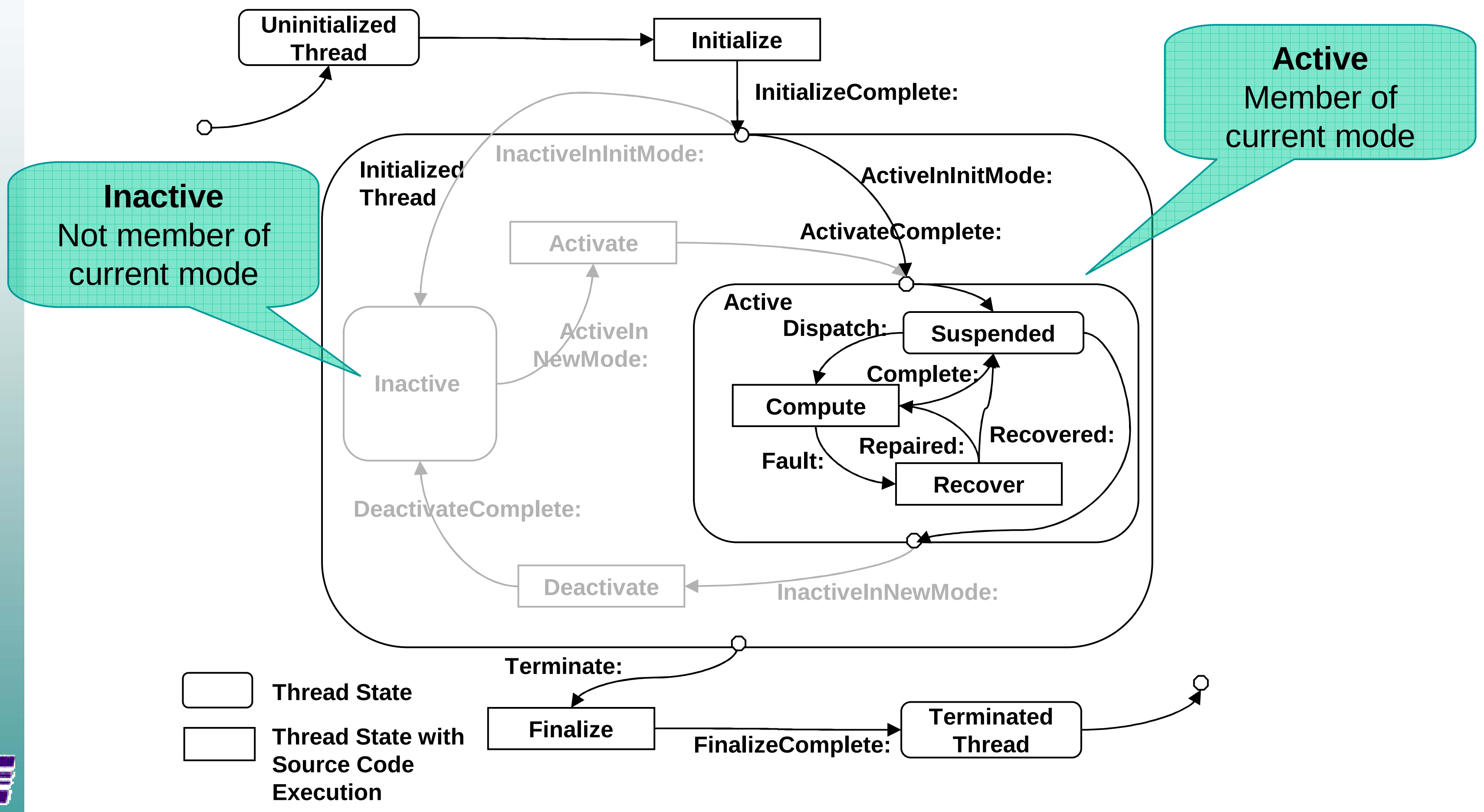
Additional protocols may be added by a user or tool vendor.

AADL Properties

- Predefined property sets
- User defined property sets



Thread States



Some Thread Properties

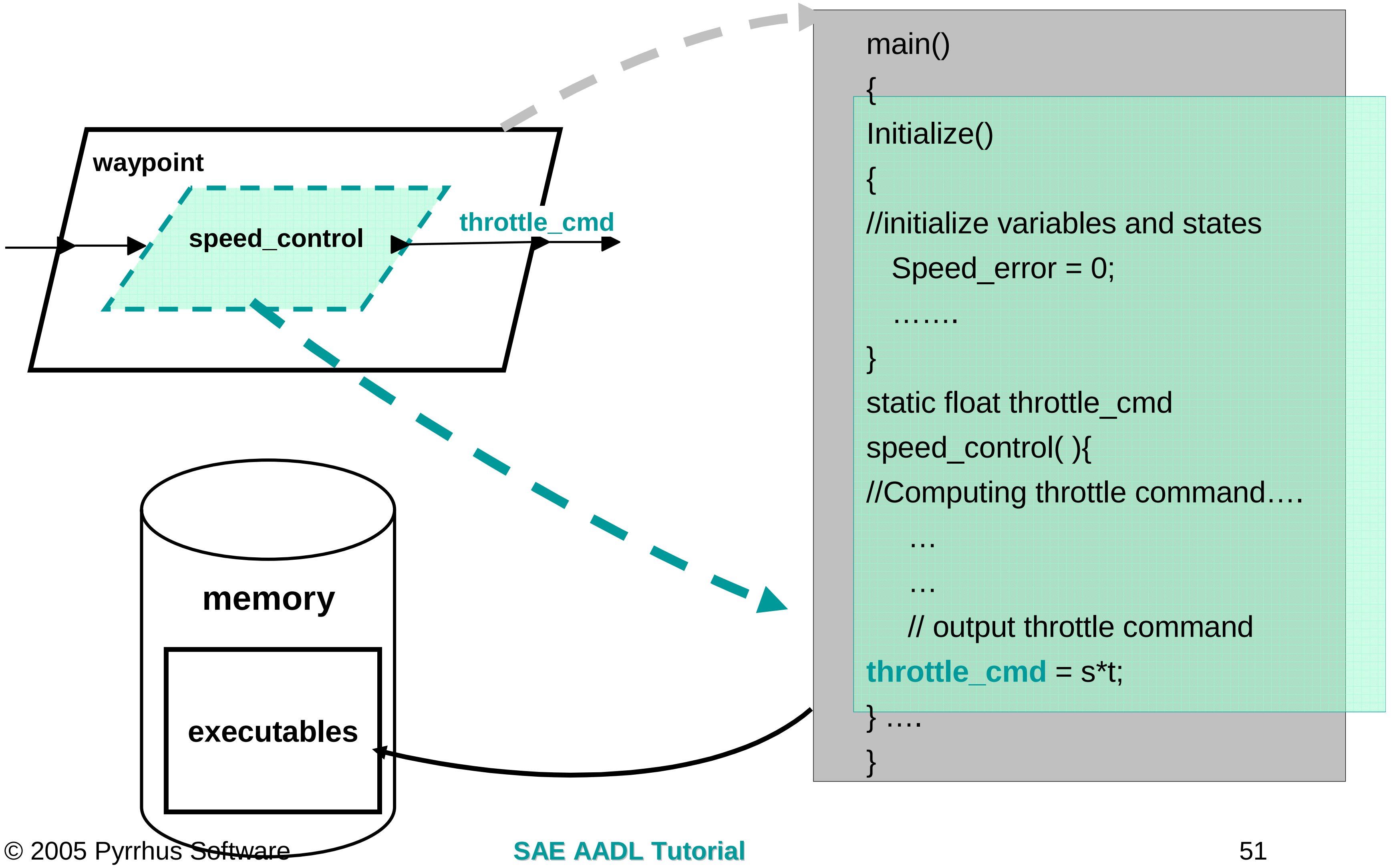
- Dispatch_Protocol => Periodic;
- Period => 100 ms;
- Compute_Deadline => value(Period);
- Compute_Execution_Time => 20 ms;
- Initialize_Entrypoint => “initialize”;
- Initialize_Deadline => 10 ms;
- Initialize_Execution_Time => 1 ms;
- Compute_Entrypoint => “speed_control”;
- Source_Text => “waypoint.java”;
- Source_Code_Size => 1.2 KB;
- Source_Data_Size => .5 KB;

Code to be executed on initialization

Code function to be executed on dispatch

File containing the application code

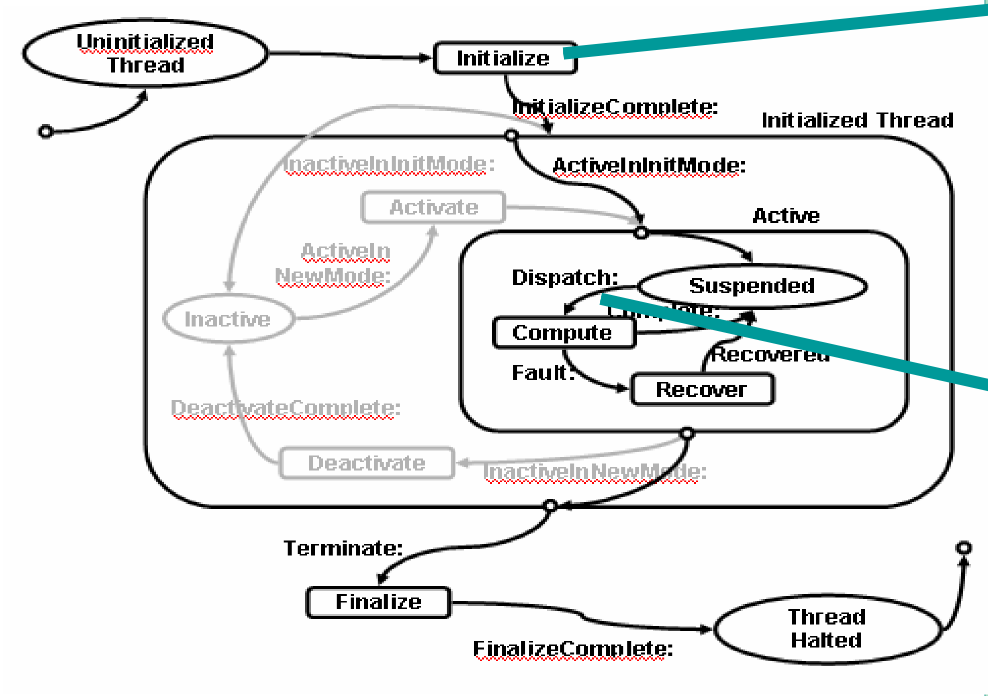
Threads to Source Text and Executables



```
main()
{
Initialize()
{
//initialize variables and states
   Speed_error = 0;
   …….
}
static float throttle_cmd
speed_control( ){
//Computing throttle command….
     …
     …
     // output throttle command
throttle_cmd = s*t;
} ….
}
```

Thread Entrypoints & Source Text

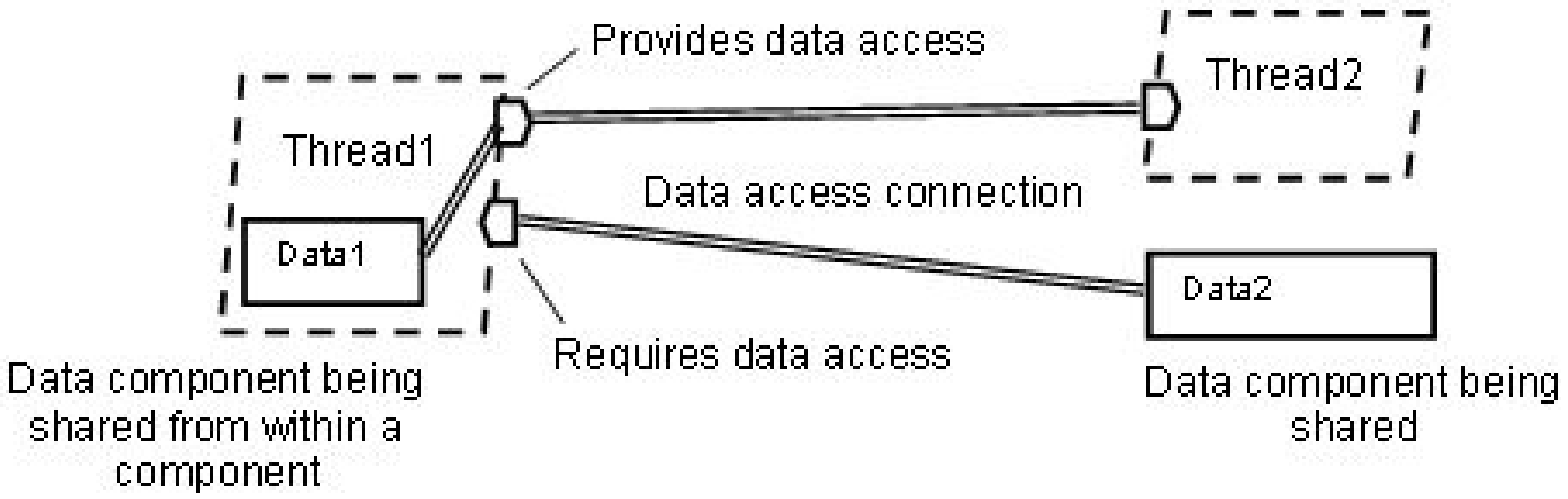
Specified by Entrypoint properties

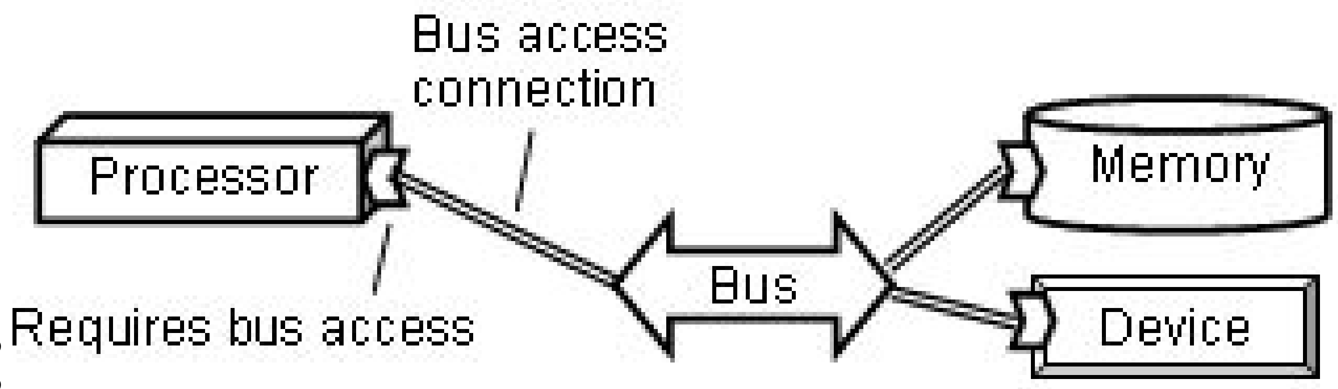


```
main()
{
Initialize()
{
//initialize variables and states
   Speed_error = 0;
   …….
}
static float throttle_cmd;
speed_control( ) {
//Computing throttle command….
     …
     …
     // output throttle command
throttle_cmd = s*t;
}
}
```

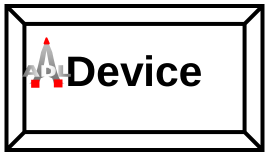
Shared Data & Bus Access

- Component requires access
- Component provides access





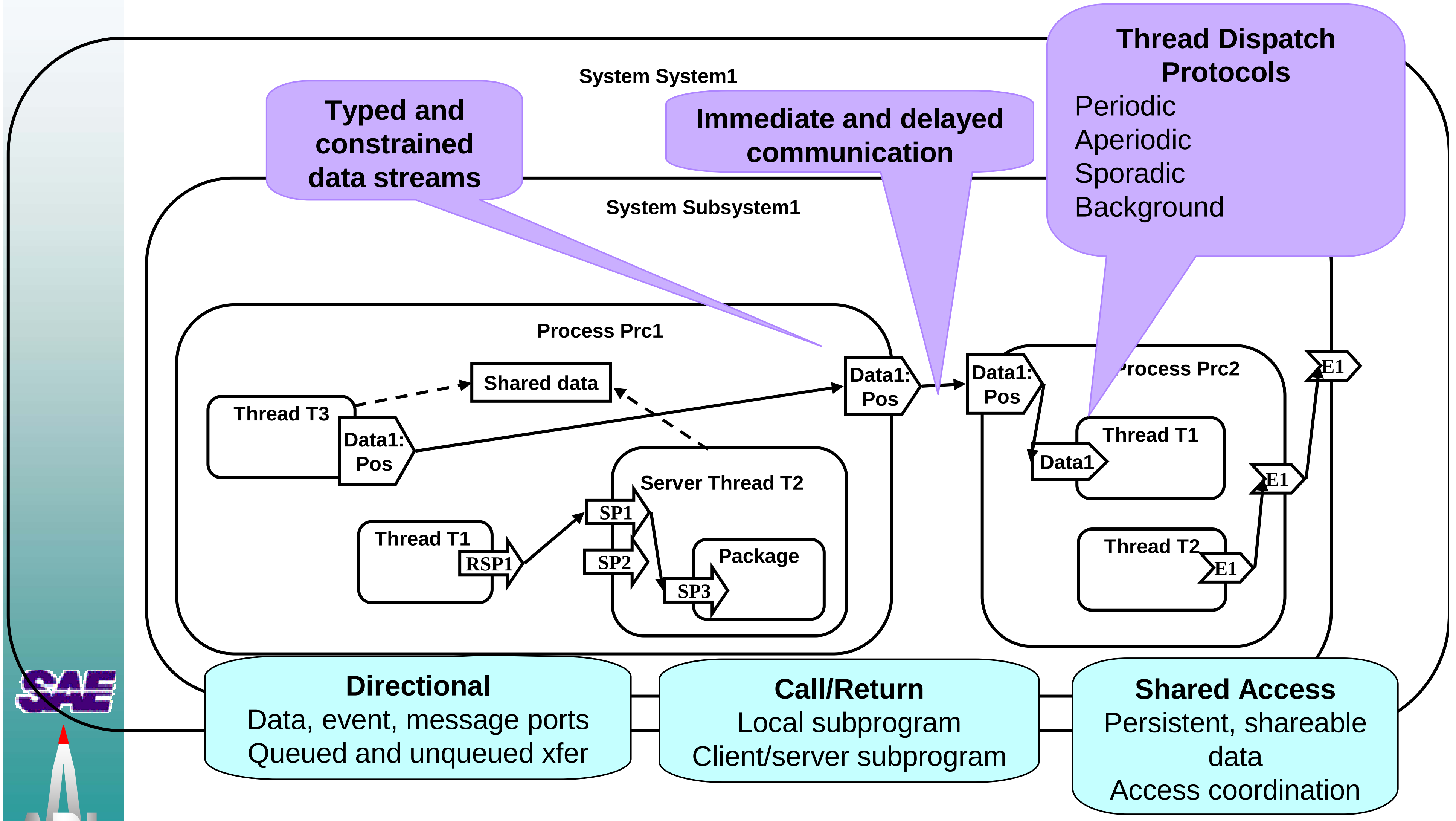
Device



- Physical component interfacing with environment.
- Interacts with application components via port connections and subprograms.
- Connects physically to processors via bus.
- May have associated software executes on connected processor.
- Examples
  - sensors and actuators
  - standalone systems such as a GPS

Features: port, subprogram	Connections: no
Requires: bus access	Modes: yes

Task & Interaction Architecture



Typed and constrained data streams

Immediate and delayed communication

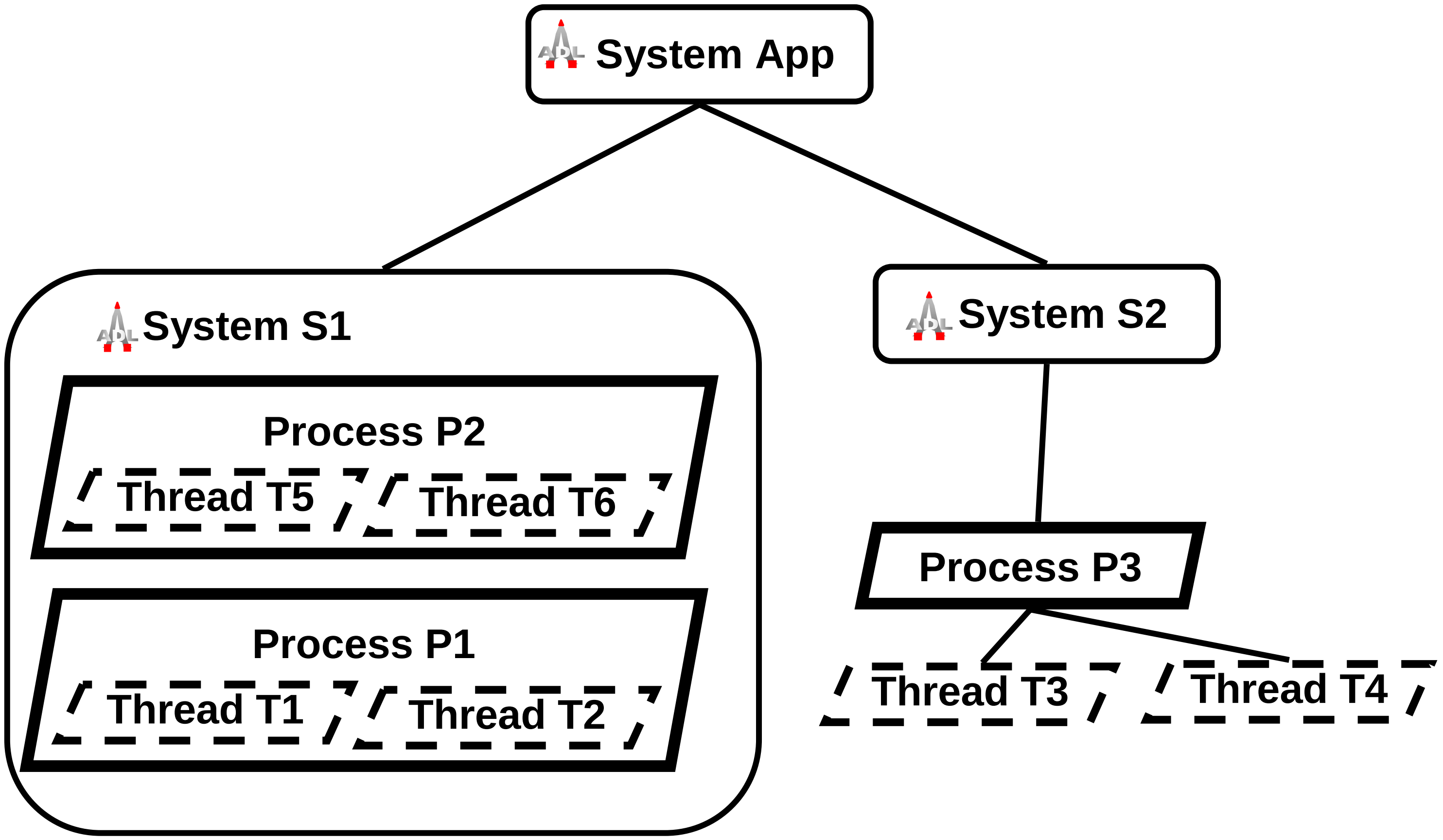
Thread Dispatch Protocols
Periodic
Aperiodic
Sporadic
Background

Directional
Data, event, message ports
Queued and unqueued xfer

Call/Return
Local subprogram
Client/server subprogram

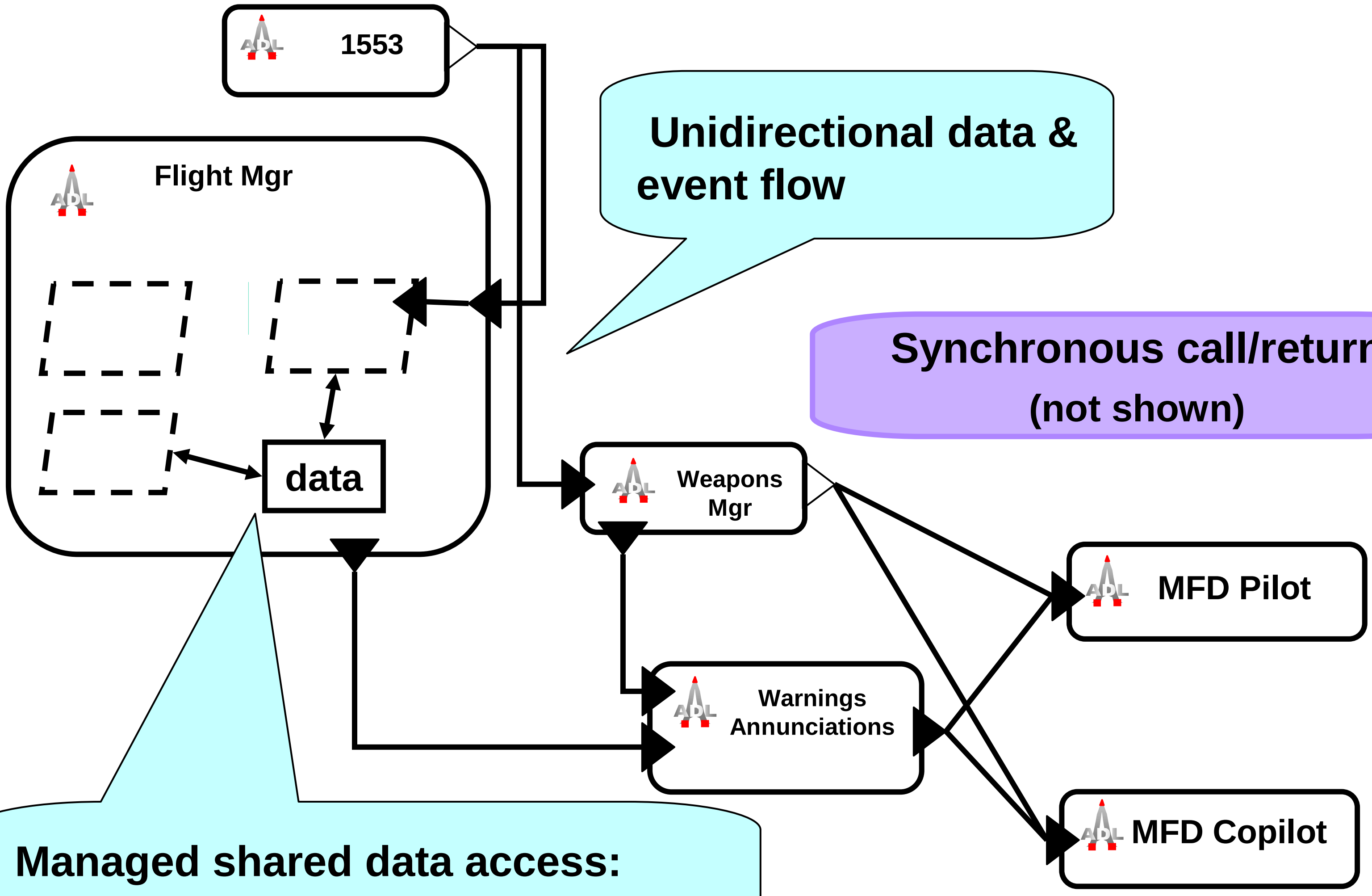
Shared Access
Persistent, shareable data
Access coordination

Application Component Hierarchy



Two ways of graphically visualizing component hierarchy

AADL Component Interaction

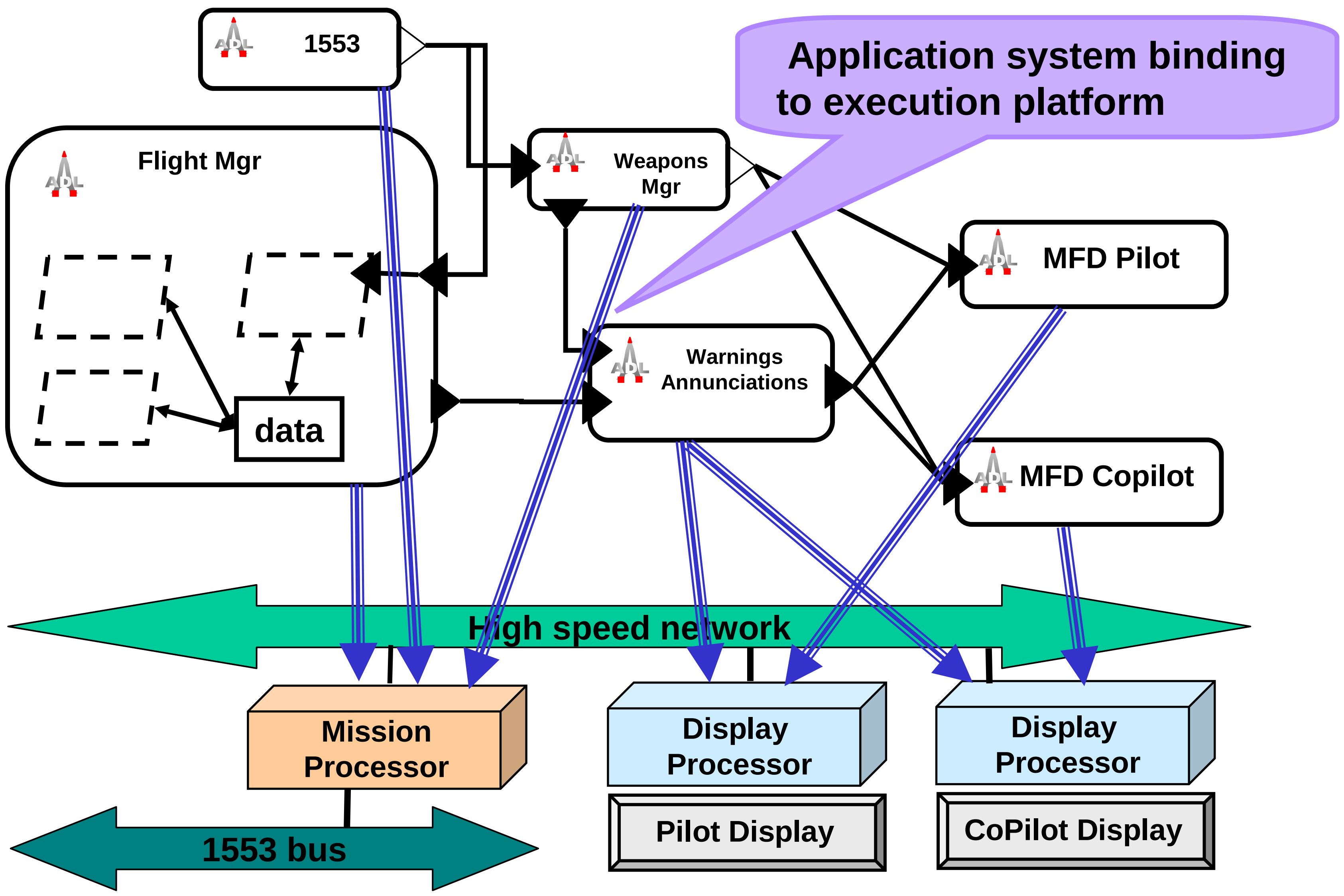


Unidirectional data & event flow

Synchronous call/return (not shown)

Managed shared data access: only two threads have access

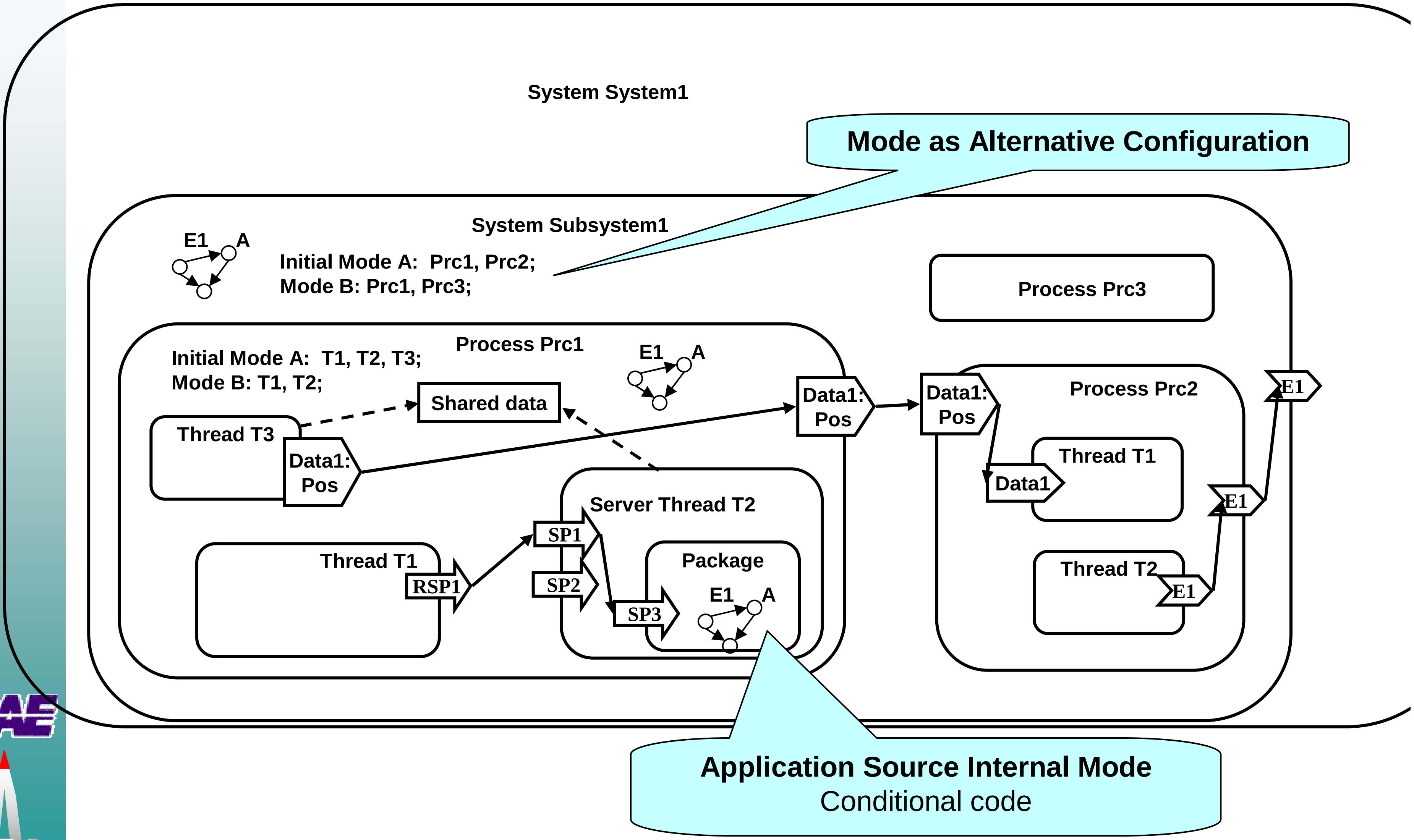
Application System & Execution Platform



Faults and Modes

- AADL provides a fault handling framework with precisely defined actions.
- AADL supports runtime changes to task & communication configurations.
- AADL defines timing semantics for task coordination on mode switching.
- AADL supports specification of mode transition actions.
- System initialization & termination are explicitly modeled.

Hierarchical Modes



A Mode Example

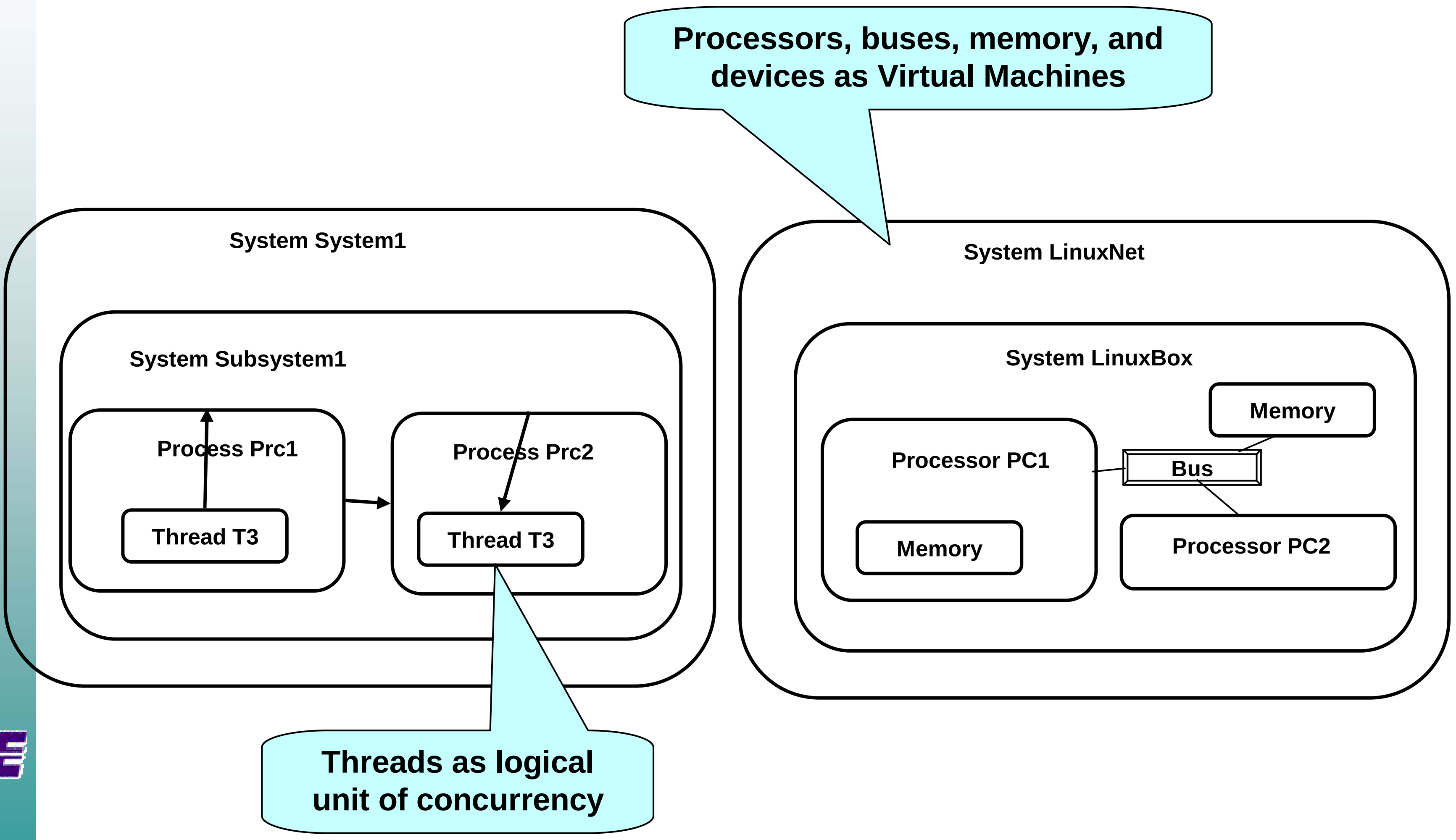
```
system PrimaryBackupPattern
features
   insignal: data port;
   outsignal: data port;
end PrimaryBackupPattern;

system implementation PrimaryBackupPattern.impl
subcomponents
   Primary: system sys;
   Backup: system sys;
connections
   inPrimary: data port insignal -> Primary.insignal;
   inBackup: data port insignal -> Backup.insignal;
   outPrimary: data port Primary.outsignal -> outsignal;
   outBackup: data port Backup.outsignal -> outsignal;
modes
   Primarymode: initial mode;
   Backupmode: mode;
   Reinitmode: mode;
   Backupmode –[restart]-> Reinitmode;
   Reinitmode –[Primary.Complete]-> Primarymode;
end PrimaryBackupPattern.impl;
```

Defines a dual redundant pattern

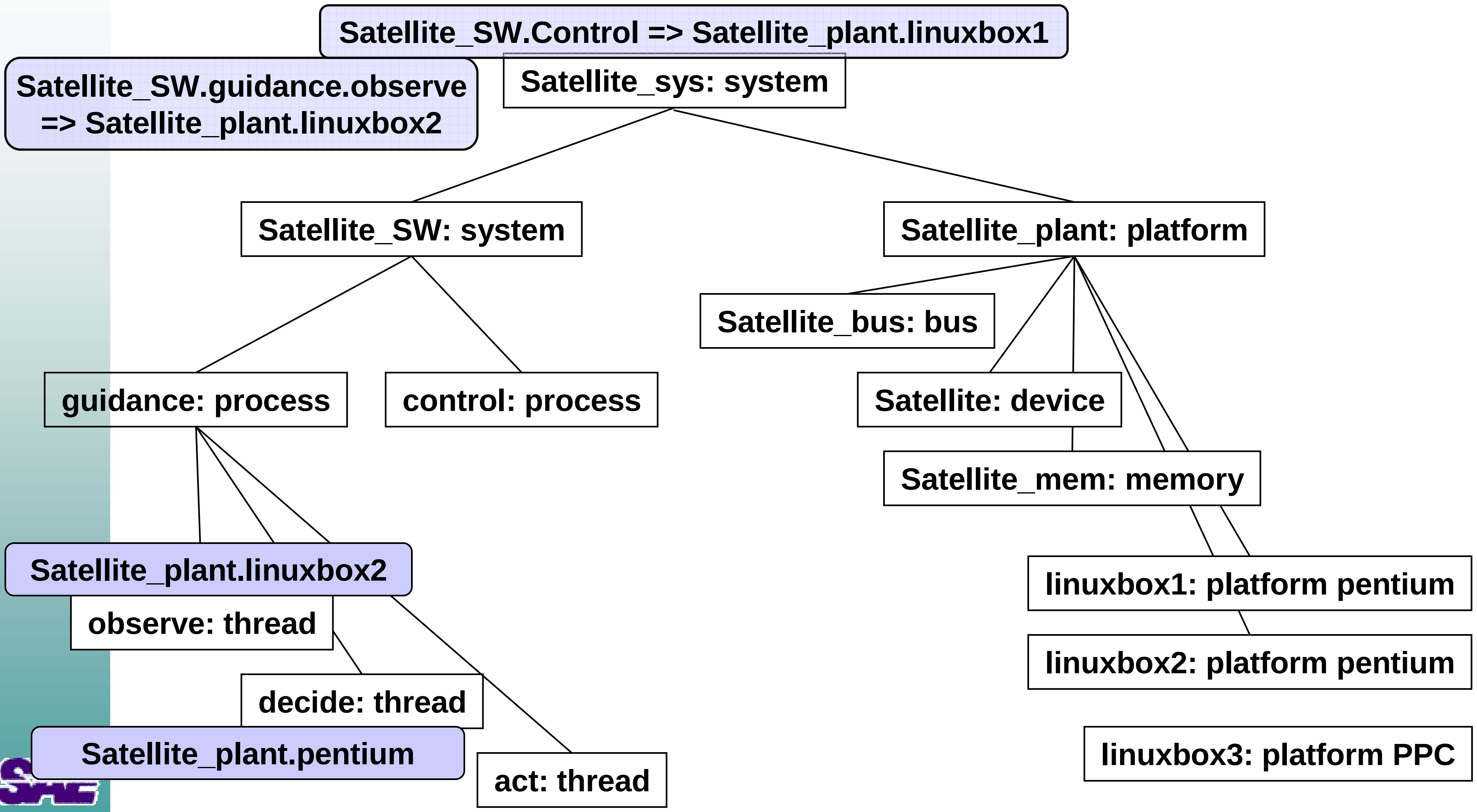
Modal Systems

- Operational modes
  - Reflecting system operation.
- Modal subsystems
  - Independent & coordinated mode switching.
- Alternate system configurations
  - Reachability of mode combinations.
- Reduced analysis space
  - Worst-case scheduling analysis.
- Dynamic configuration management
  - Inconsistency identification through analysis.
  - Inconsistency repair through selective reconfiguration.

System & Execution Platforms



Binding Systems to Execution Platforms



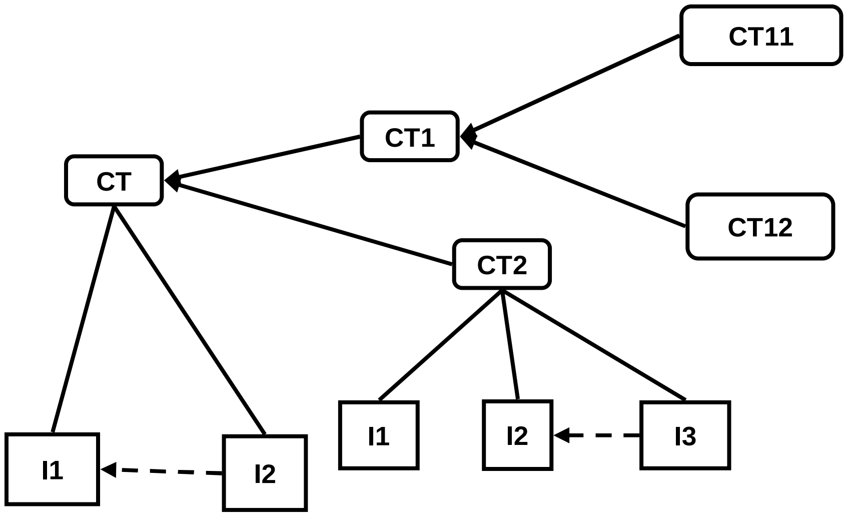
Extending AADL

- Component Types have multiple implementations $\longrightarrow$ families
- Component extension and refinement
- Packages
- Property Sets
- Annex Subclauses
- AADL Standard Annexes

Extensible Components

Component type (interface)
Component implementations
Subcomponents (hierarchy)
Component instance

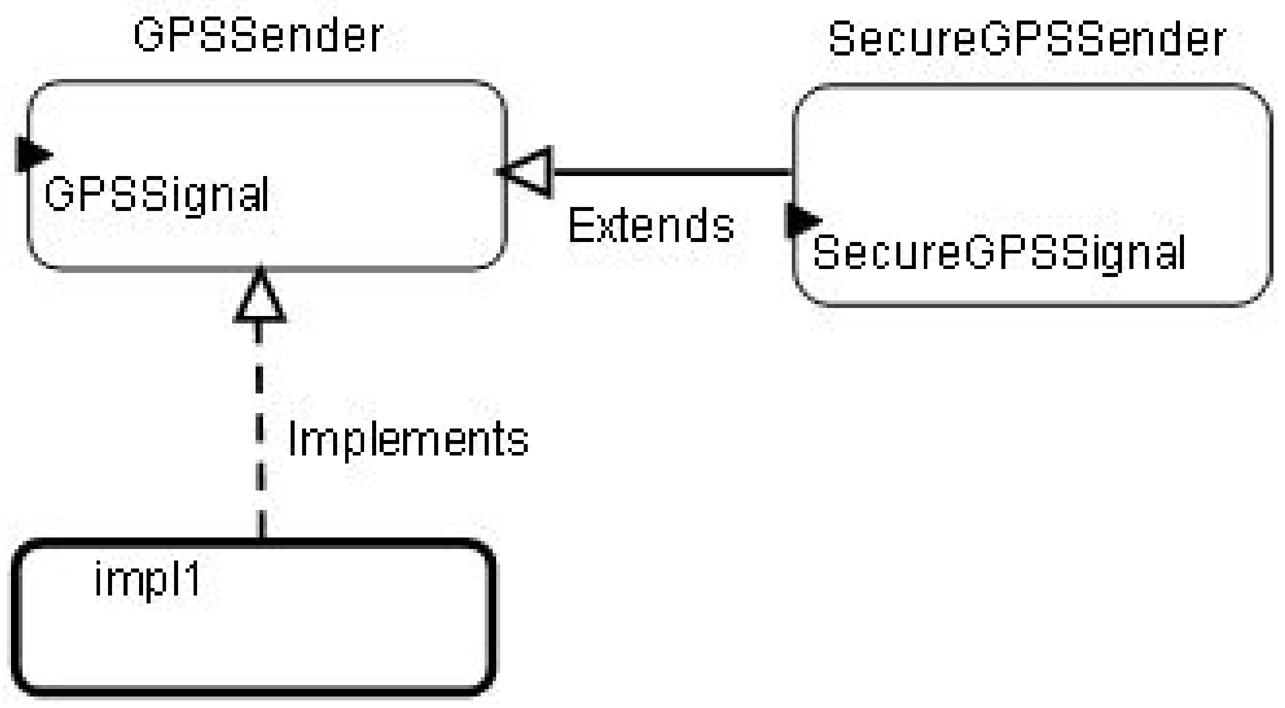
Ports
Connections
Modes
Properties
Behavior



Component Classifier Refinement

▢	**Component type**
←	**Component type extension**
□	**Component implementation**
◄- -	**Component implementation extension**

Extending an AADL Component



Refined Dual Redundancy Pattern

```
system PassivePrimaryBackup extends PrimaryBackupPattern
features
   restart: in event port;
end PassivePrimaryBackup;
```

Provides externally restart control

```
system implementation PassivePrimaryBackup.impl extends
   PrimaryBackupPattern.impl
subcomponents
   Primary: refined to system in modes ( Primarymode );
   Backup: refined to system in modes ( Backupmode );
   Reinit: system reloadsys in modes ( Reinitmode );
connections
   inPrimary: refined to data port in modes ( Primarymode );
   inBackup: refined to data port in modes ( Backupmode );
   outPrimary: refined to data port in modes ( Primarymode );
   outBackup: refined to data port in modes ( Backupmode );
modes
   Reinitmode: mode;
   Backupmode –[restart]-> Reinitmode;
   Reinitmode –[Reinit.Complete]-> Primarymode;
end PassivePrimaryBackup.impl;
```

Defines who is active when

Defines restart logic

Component Evolution

- Partially complete component type and implementation.
- Multiple implementations for a component type.
- Extension & refinement
  - Component templates to be completed.
  - Variations and extensions in interface (component type).
  - Variations and extensions in implementations.

Large-Scale Development

- Component type and implementation declarations in *packages*
    - Name scope for component types.
    - Grouping into manageable units.
    - Nested package naming.
    - Qualified naming to manage name conflicts.
- Supports independent development of subsystems.
- Supports large-scale system of system development.

AADL Language Extensions

- Core standard plus optional annexes.
- Examples
  - Error Model
  - ARINC 653
  - Behavior
  - Constraint sublanguage
- Annex as document
  - New properties through property sets
  - Annex-specific subclauses expressed in an annex-specific sublanguage

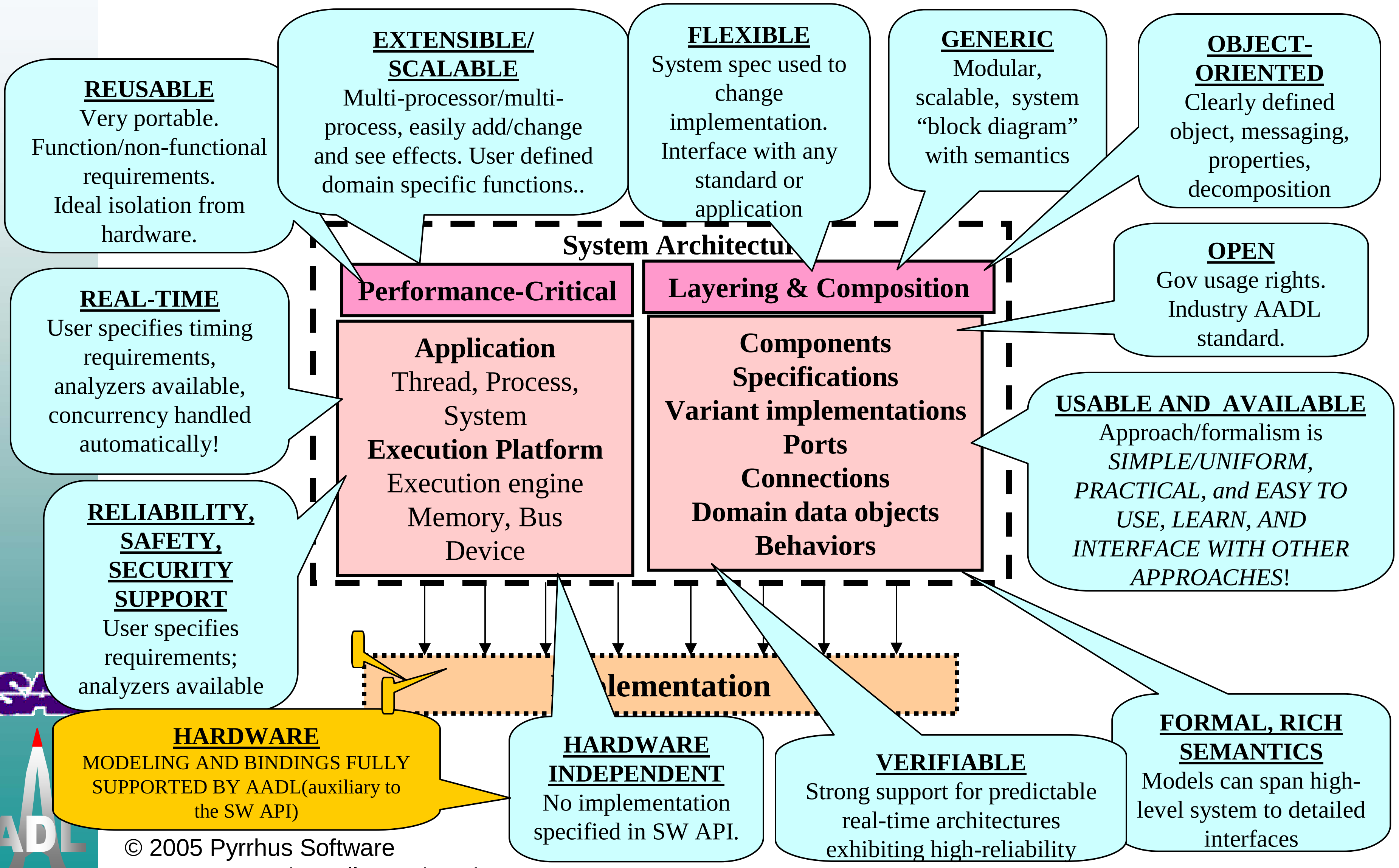
Summary of AADL Capabilities

- AADL abstractions separate application domain concerns from runtime architecture concerns.
- AADL combines predictable task execution with deterministic communication.
- AADL is effective for embedded, real-time, high-dependability, software-intensive application systems.
- AADL supports predictable system analysis and deployment through model-based system engineering.
- AADL component & communication semantics facilitate the dialogue between application and software experts.
- AADL provides an extensible basis for a wide range of embedded systems analyses.
- AADL builds on 15 years of DARPA investment + experiments.

Tutorial Outline

- Background & Introduction
- AADL – The Language
- **AADL Development Environment**
  - Tools
  - Guidelines
- AADL in Use
- Summary and Conclusions

The AADL in a Nutshell



Application
Thread, Process, System
Execution Platform
Execution engine
Memory, Bus
Device

Components
Specifications
Variant implementations
Ports
Connections
Domain data objects
Behaviors

REUSABLE
Very portable. Function/non-functional requirements. Ideal isolation from hardware.

EXTENSIBLE/ SCALABLE
Multi-processor/multi-process, easily add/change and see effects. User defined domain specific functions..

FLEXIBLE
System spec used to change implementation. Interface with any standard or application

GENERIC
Modular, scalable, system "block diagram" with semantics

OBJECT-ORIENTED
Clearly defined object, messaging, properties, decomposition

OPEN
Gov usage rights. Industry AADL standard.

REAL-TIME
User specifies timing requirements, analyzers available, concurrency handled automatically!

USABLE AND AVAILABLE
Approach/formalism is *SIMPLE/UNIFORM, PRACTICAL, and EASY TO USE, LEARN, AND INTERFACE WITH OTHER APPROACHES*!

RELIABILITY, SAFETY, SECURITY SUPPORT
User specifies requirements; analyzers available

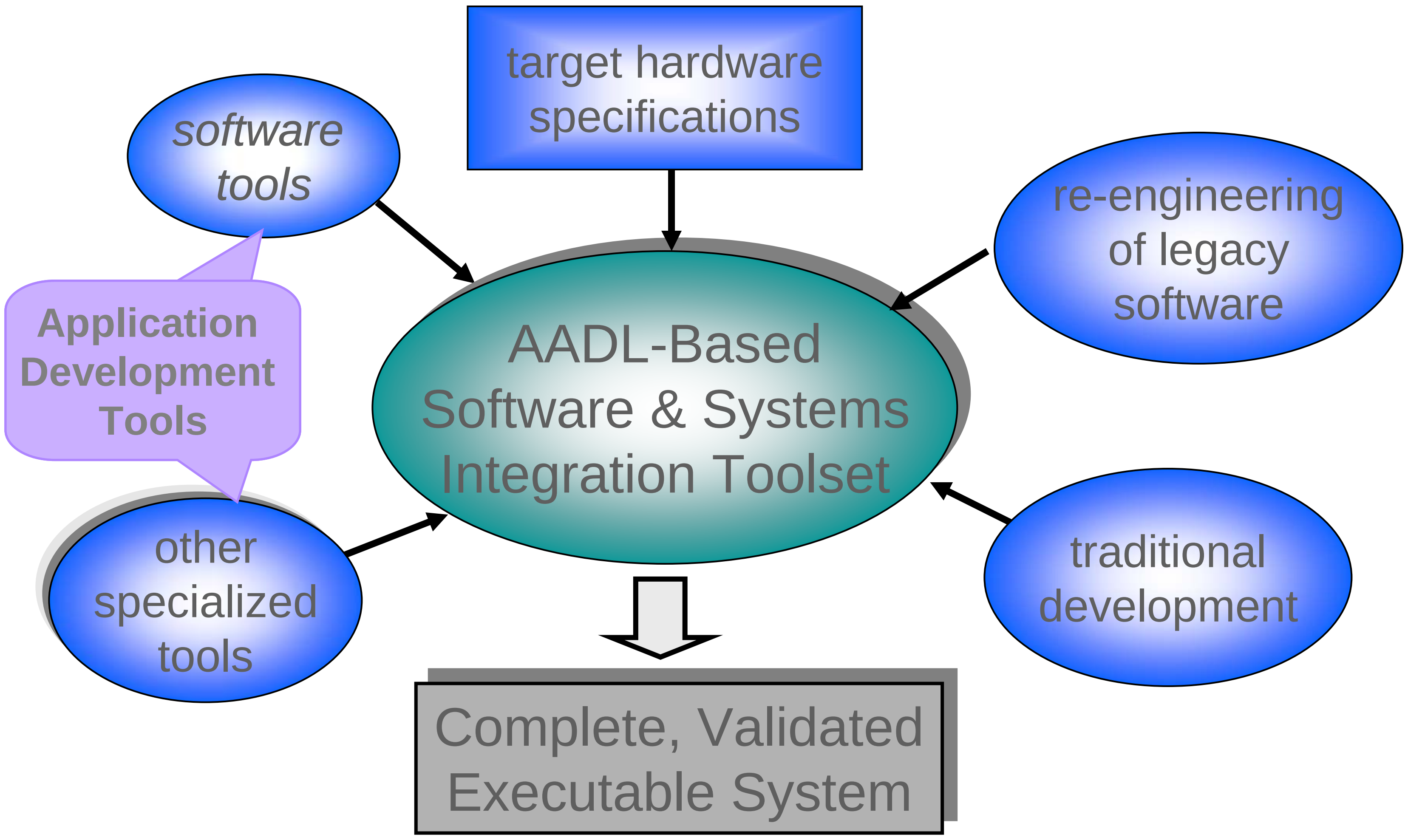
HARDWARE
MODELING AND BINDINGS FULLY SUPPORTED BY AADL(auxiliary to the SW API)

HARDWARE INDEPENDENT
No implementation specified in SW API.

VERIFIABLE
Strong support for predictable real-time architectures exhibiting high-reliability

FORMAL, RICH SEMANTICS
Models can span high-level system to detailed interfaces

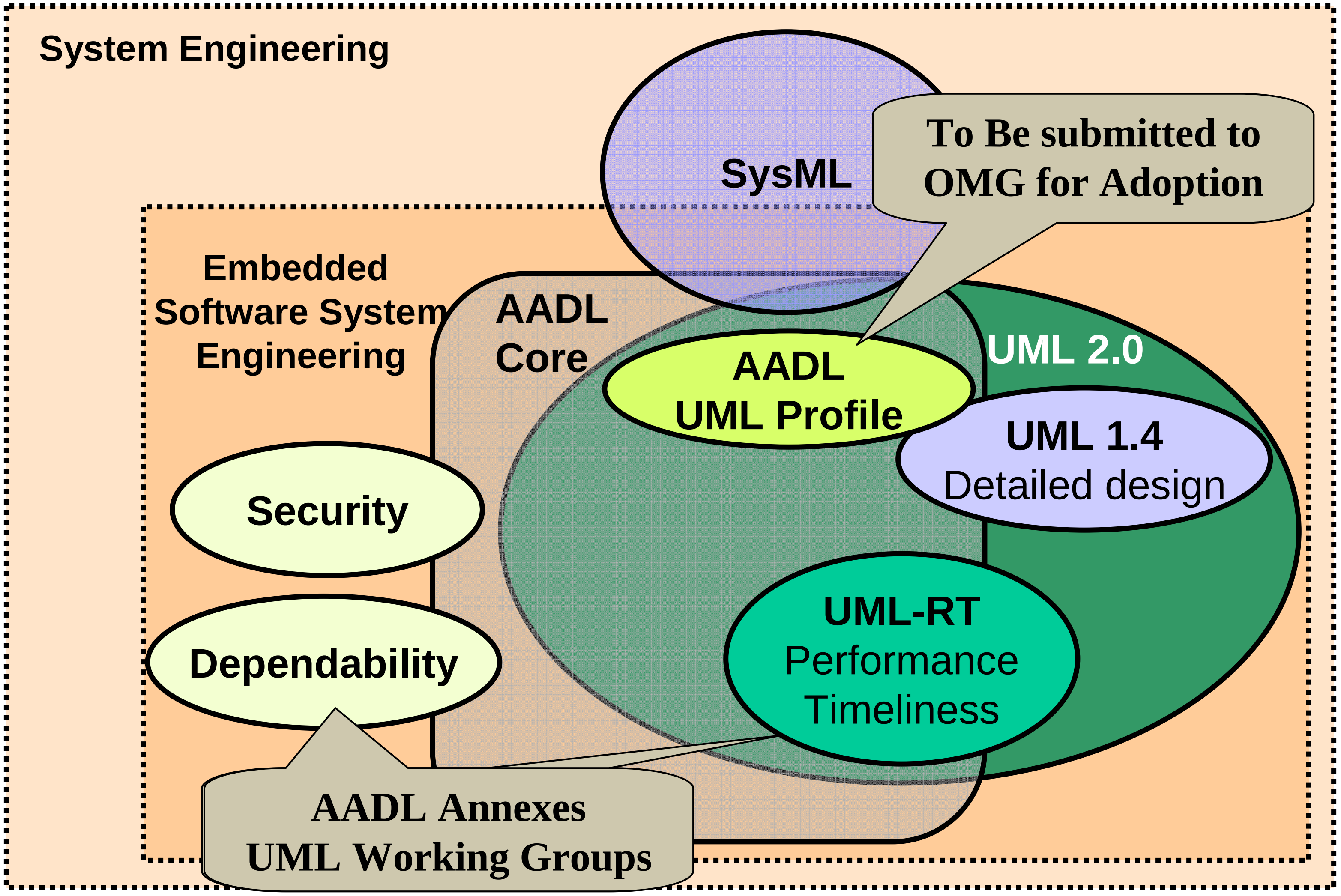
AADL Development Environment


software tools
target hardware specifications
re-engineering of legacy software
Application Development Tools
AADL-Based Software & Systems Integration Toolset
other specialized tools
traditional development
Complete, Validated Executable System

Two-Tier Tool Strategy

- Open Source AADL Tool Environment (OSATE)
  - Developed by SEI
  - Low entry cost solution (no cost GPL)
  - Multi-platform based on Eclipse
  - Prototyping environment for project-specific analysis
  - Architecture research platform
- Commercial Tool Support
  - UML tool environment extension based on UML profile
  - Extension to existing modeling environment with AADL export/import
  - Analysis tools interfacing via XML or XML to native filter
  - Runtime system generation tools

AADL/UML Relationship

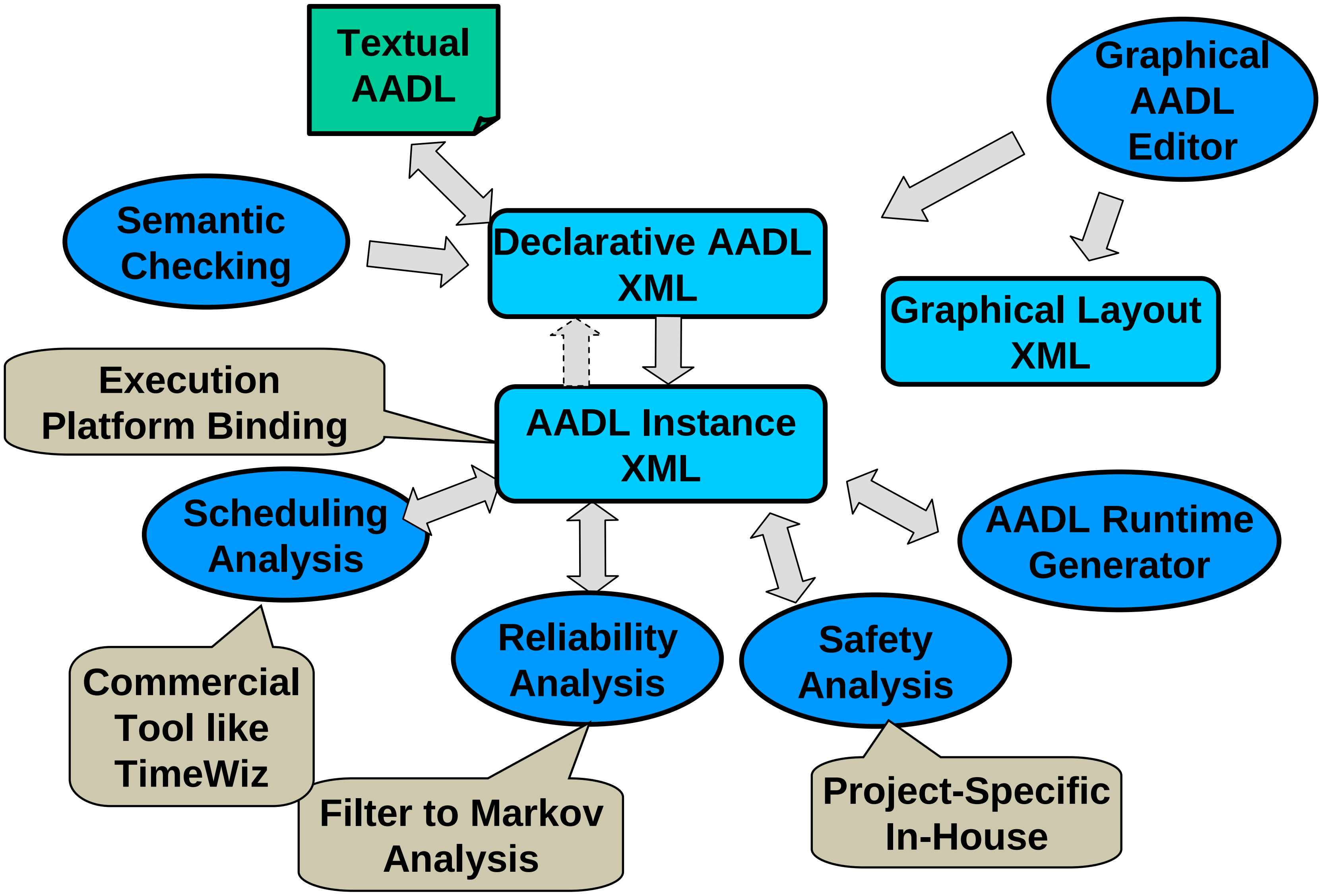

System Engineering
SysML
To Be submitted to OMG for Adoption
Embedded Software System Engineering
AADL Core
AADL UML Profile
UML 2.0
UML 1.4 Detailed design
Security
Dependability
UML-RT Performance Timeliness
AADL Annexes UML Working Groups

Multiple Viewpoints of SAE AADL

- ***Component View***
  - Model of system composition & hierarchy.
  - Well-defined component interfaces.
- ***Concurrency & Interaction View***
  - Time ordering of data, messages, and events.
  - Dynamic operational behavior.
  - Explicit interaction paths & protocols.
- ***Execution View***
  - Execution platform as resources.
  - Specification & analysis of runtime properties
    - timeliness, throughput, reliability, graceful degradation, ...
  - Binding of application software.
- ***User-defined View***
  - Analysis-oriented.

Primary targets are the concepts and viewpoints associated with an operational system.

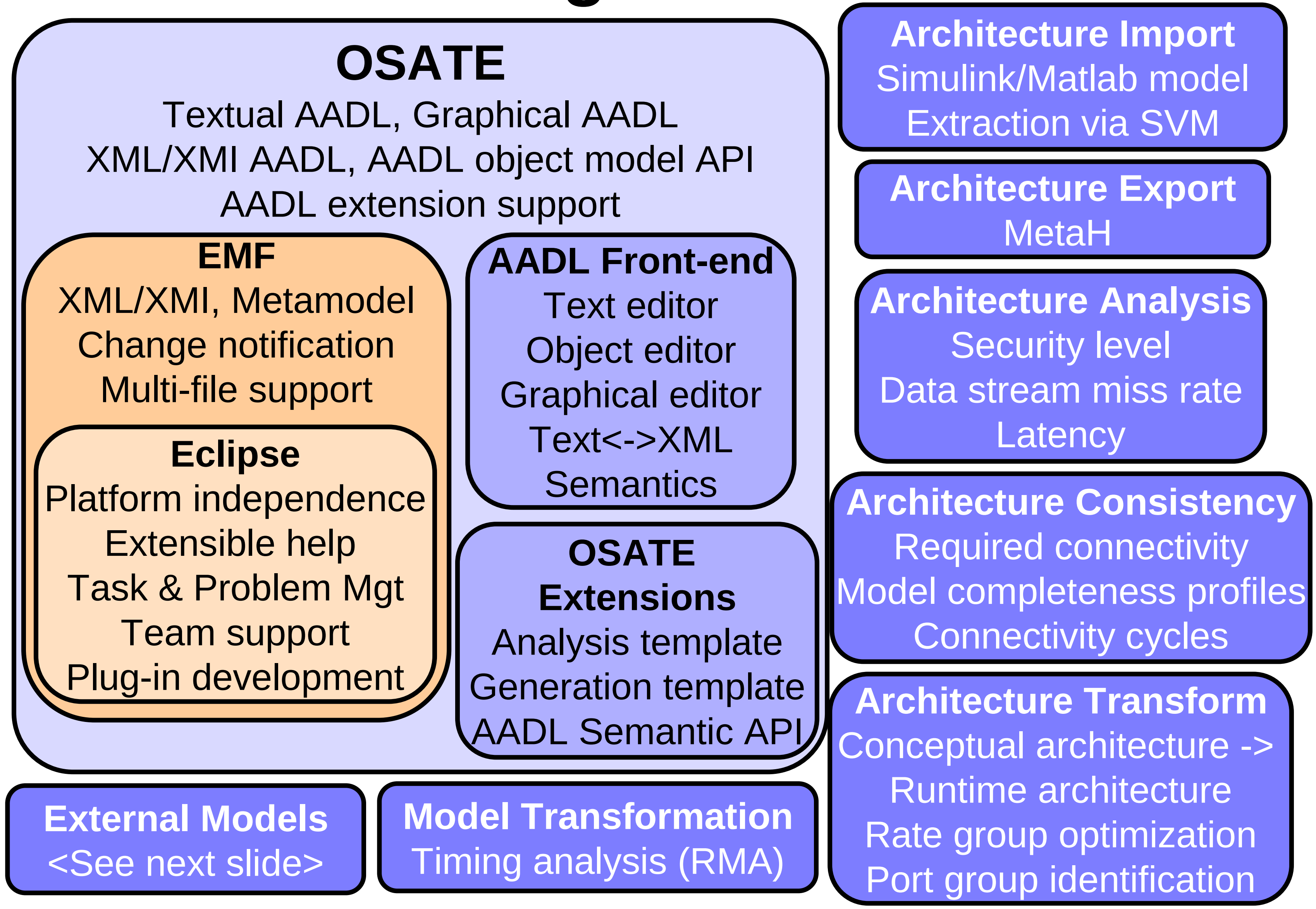
An XML-Based AADL Tool Strategy



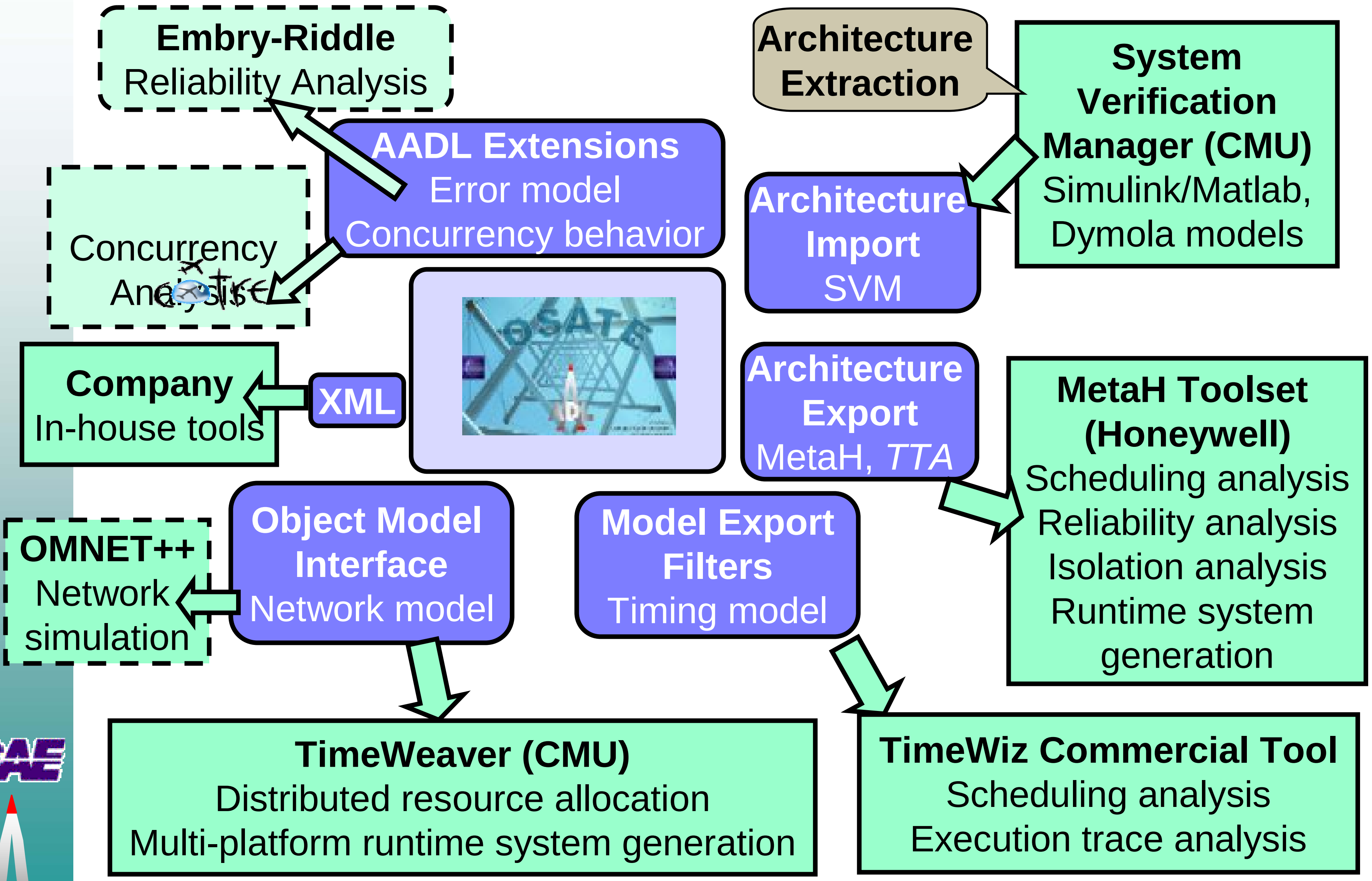
Open Source AADL Tool Environment (OSATE)

- OSATE is
  - Developed by the Software Engineering Institute
  - Available at under a no cost Common Public License (CPL)
  - Implemented on top of Eclipse Release 3 (www.eclipse.org)
  - Generated from an AADL meta model
  - A textual & graphical AADL front-end with semantic & XML/XMI support
  - Extensible through architecture analysis & generation plug-ins
- OSATE offers
  - Low cost entrypoint to the use of SAE AADL
  - Platform for in-house prototyping of project specific architecture analysis
  - Platform for architecture research with access to industrial models & industry exposure to research results

OSATE Plug-in Extensions



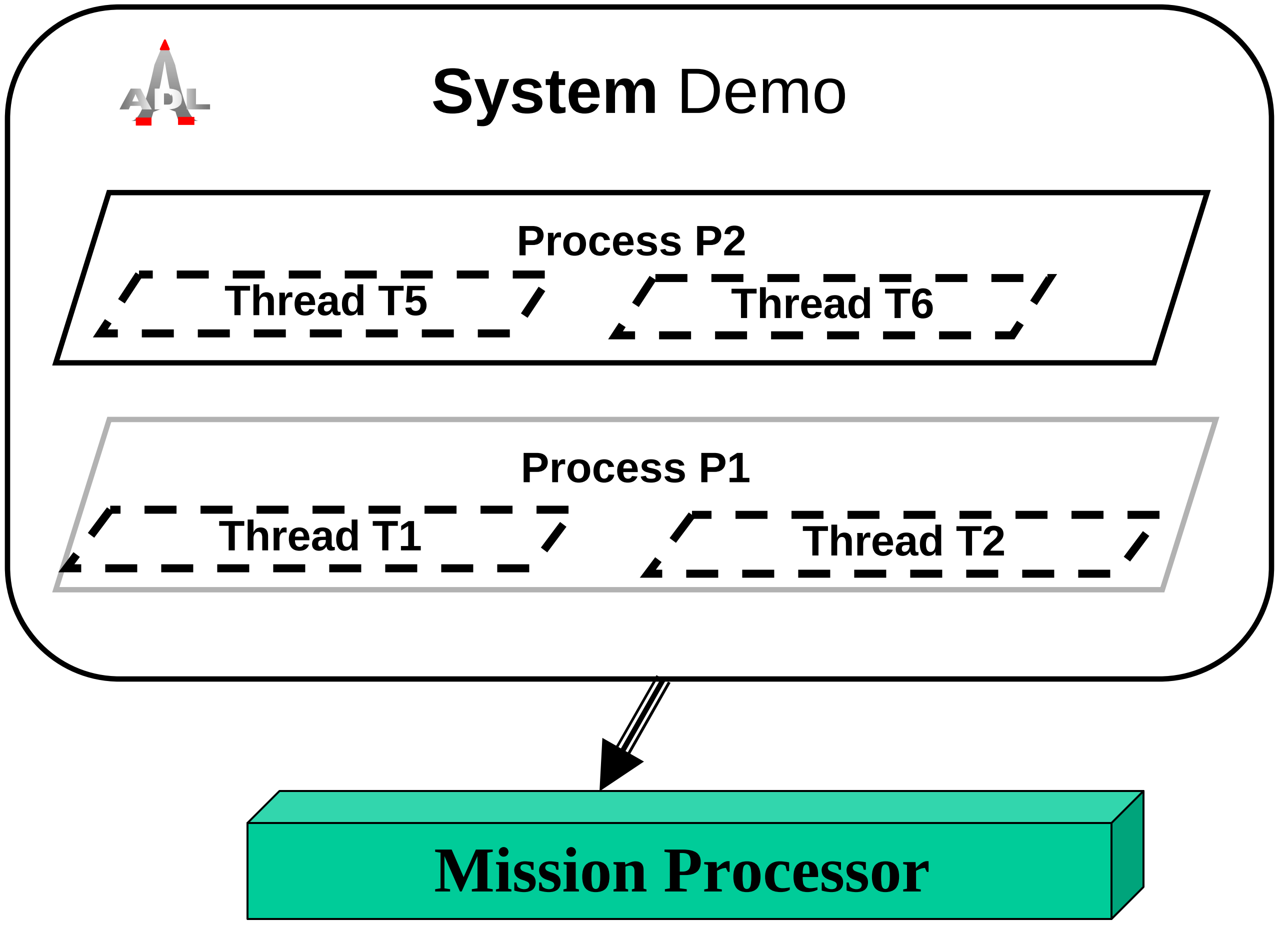
OSATE and External Tools



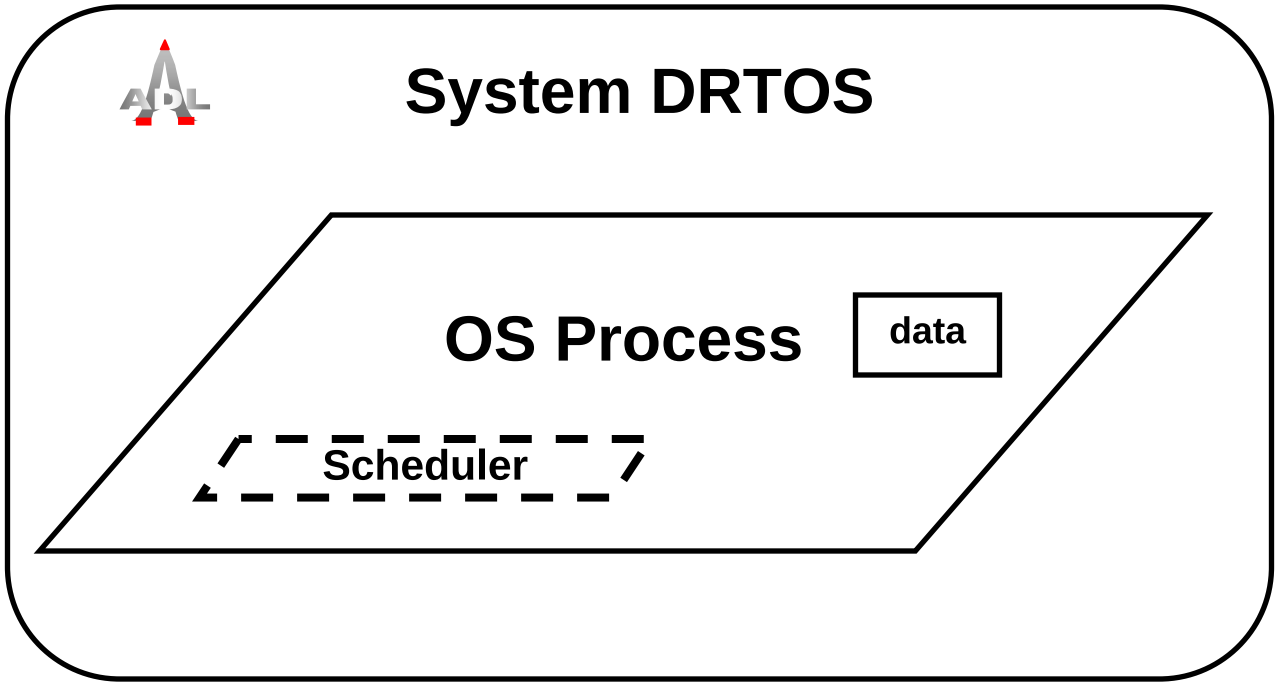
Representing Operating Systems in an AADL Specification

- OS is part of a processor specification.
- OS can be modeled using software and execution platform components.
  - OS memory may be modeled as a process with access to the actual memory.
  - OS may be represented as a thread which is a subcomponent of the OS process.

Processor Provides OS



Using AADL to Model an OS

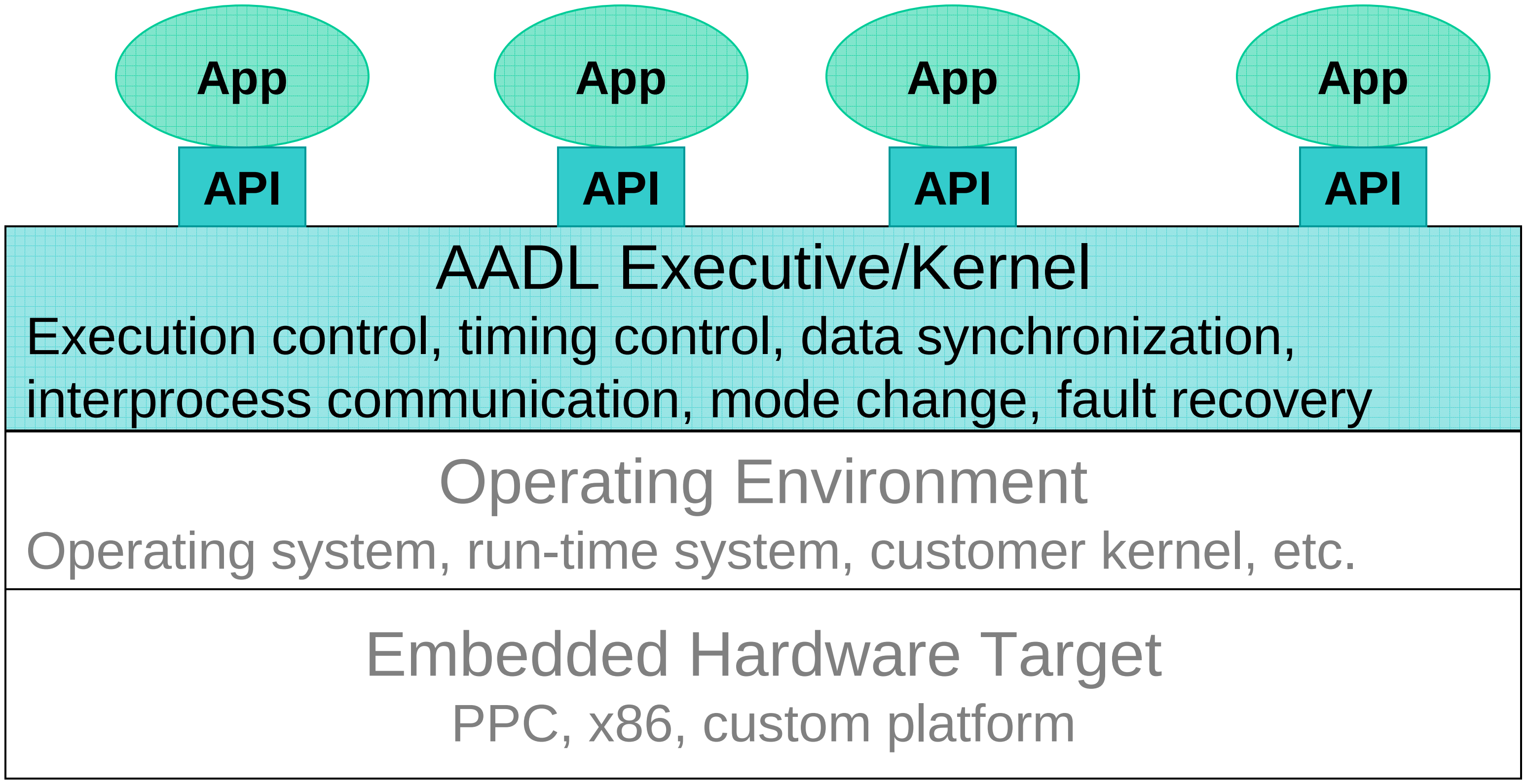


Programming Language Guidelines

- For vendors to create tools to support the development and analysis of source code and AADL models.
- For software application engineers developing source code and corresponding AADL specifications.
- For program managers to obtain a consistent and uniform mapping between source code and AADL specifications.
- For system integrators to specify the definition of the form of software components that are acceptable for integration.

Programming Language Guidelines

AADL Execution Environment


App
App
App
App
API
API
API
API
AADL Executive/Kernel
Execution control, timing control, data synchronization, interprocess communication, mode change, fault recovery
Operating Environment
Operating system, run-time system, customer kernel, etc.
Embedded Hardware Target
PPC, x86, custom platform

Programming Language Guidelines

- An ***application source code*** *module* is any separately compilable unit written in a traditional programming language such as Ada or C.
- ***AADL specifications*** are used to describe important characteristics of application source code modules including the application source code objects that are shared between modules and the interfaces between application source code modules.
- ***AADL properties*** are used to define characteristics of software components, such as timing, safety level, execution behavior, and memory size as well as the actual application source text that implements application software components.

Programming Language Guidelines

AADL Software Components

AADL	Ada	C
Data	Data	Data
Subprogram	Procedure, Function	Function
Thread	Procedure	Function
Thread Group	Procedure(s) & data in a package	Function(s) & data in an include file
Process	Application	Application
System	Applications+	Applications+

Programming Language Guidelines

Communication and Control

AADL	Ada	C
Data Port	Static Variables in Packages	Static variables in include files
Event Port	Parameter to Raise_Event	Parameter to Raise_Event
Event Data Port	Record with data pointer; Raise_Event	Struct with data pointer;Raise_Event
Subprogram Parameter	Formal Parameters	Formal Parameters
Requires Provides	Shared data / protected object	Shared data
Feature Subprogram	Subprogram	Function

Programming Language Guidelines

Packages and Libraries

AADL	Ada	C
Package	Package	Include file
Private Part	Private Part	“Private” Macro

Programming Language Guidelines

Language Support Property Set

```
property set Language_Support is
   Integer_Range       : range of aadlinteger applies to ( data );
   Real_Range          : range of aadlreal applies to ( data );
   String_List_Value   : list of aadlstring applies to ( data );
   Positive            : type aadlinteger 0..value(Max_Aadlinteger);
   Constant_Integer    : aadlinteger applies to
                              ( data, thread, process, system );
   Constant_Real       : aadlreal applies to ( data, thread, process, system );
   Constant_String     : aadlstring applies to ( data, thread, process, system );
   Priority            : aadlinteger applies to ( thread );
   Data_Format         : enumeration ( Integer, int, Float, flt, Boolean,
                                       String, str, Character, char,
                                       Wide_String, Natural, Positive, Enum,
                                       long, long_long) applies to ( Data );
   Data_Structure      : enumeration ( Record, Array, Protected, Tagged,
                                       Abstract, Interface, struct)
                              applies to ( Data );
end Language_Support;
```

Programming Language Guidelines

AADL Specification

```
package Sampling
public
    data Sample
        properties
            Source_Data_Size => 16 B;
            Language_Support::Data_Format => Character;
            Language_Support::Data_Structure => Record;
    end Sample;
    data Sample_Set extends Sample
        properties
            Source_Data_Size => 1 MB;
            Language_Support::Data_Structure => Array;
    end Sample_Set;
    data Dynamic_Sample_Set extends Sample_Set
    end Dynamic_Sample_Set;
end Sampling;
```

Programming Language Guidelines

Ada

```
package Sampling is
   type Sample is record
      Byte0 : character;   Byte1 : character;   Byte2 : character;   Byte3 : character;
      Byte4 : character;   Byte5 : character;   Byte6 : character;   Byte7 : character;
      Byte8 : character;   Byte9 : character;   ByteA : character;   ByteB : character;
      ByteC : character;   ByteD : character;   ByteE : character;   ByteF : character;
   end record;
    for Sample'Size use 16 * 8;
   -- Note that Ada 95 Size is represented in bits.  The AADL property was for
   -- a size of 16 bytes, hence the * 8.

   type Sample_Set is array ( 1..65536 ) of Sample;
   for Sample_Set'Size use 1_048_576 * 8; -- 1MB == 1048576 bytes

   type Dynamic_Sample_Set is access all Sample_Set;
   type Access_Sample_Set is access all Sample_Set;
end Sampling;
```

Programming Language Guidelines

C

```
/* sampling.h */

typedef struct {
  char Byte0, Byte1, Byte2, Byte3, Byte4, Byte5, Byte6, Byte7,
       Byte8, Byte9, ByteA, ByteB, ByteC, ByteD, ByteE, ByteF;
} Sample;

typedef Sample Sample_Set[65535];
typedef Sample_Set *Dynamic_Sample_Set;
typedef Sample_Set *Access_Sample_Set;
```

Programming Language Guidelines

AADL Specification

```
thread DriverModeLogic
features
   BreakPedalPressed      : in data port Standard::Boolean;
   ClutchPedalPressed     : in data port Standard::Boolean;
   Activate               : in data port Standard::Boolean;
   Cancel                 : in data port Standard::Boolean;
   OnNotOff               : in data port Standard::Boolean;
   CruiseActive           : out data port Standard::Boolean;
end DriverModeLogic;

thread implementation DriverModeLogic.Ada
properties
  Dispatch_Protocol=>Periodic;
  Period=> 10 ms;
end DriverModeLogic.Ada;
```

Programming Language Guidelines

AADL Specification

```
process CruiseControl
features
   BreakPedalPressed     : in data port Standard::Boolean;
   ClutchPedalPressed    : in data port Standard::Boolean;
   Activate              : in data port Standard::Boolean;
   Cancel                : in data port Standard::Boolean;
   OnNotOff              : in data port Standard::Boolean;
   CruiseActive          : out data port Standard::Boolean;
end CruiseControl;
```

Programming Language Guidelines

AADL Specification

```
process implementation CruiseControl.Ada
subcomponents
    DriverMode : thread DriverModeLogic.Ada;
connections
   data port BreakPedalPressed -> DriverMode.BreakPedalPressed;
   data port ClutchPedalPressed -> DriverMode.ClutchPedalPressed;
   data port Activate -> DriverMode.Activate;
   data port Cancel -> DriverMode.Cancel;
   data port OnNotOff -> DriverMode.OnNotOff;
   data port DriverMode.CruiseActive -> CruiseActive;
end CruiseControl.Ada;
```

Ada

```
with AADL; use AADL;
with Ada.Real_Time;  use Ada;
with CruiseControl_Ports;
procedure CruiseControl is
        procedure DriverMode;
        type DriverMode_Parameter_Block is
                record
                        BreakPedalPressed   : Boolean;
                        ClutchPedalPressed  : Boolean;
                        Activate            : Boolean;
                        Cancel              : Boolean;
                        OnNotOff            : Boolean;
                        CruiseActive        : Boolean;
                end record;

        DriverModeParameters : DriverMode_Parameter_Block :=
                (       CruiseControl_Ports.BreakPedalPressed,
                        CruiseControl_Ports.ClutchPedalPressed,
                        CruiseControl_Ports.Activate,
                        CruiseControl_Ports.Cancel,
                        CruiseControl_Ports.OnNotOff,
                        CruiseControl_Ports.CruiseActive );
```

Ada

```
        procedure DriverMode  is
        begin
                loop
                        AADL.Await_Dispatch;
                        -- do work e.g. return a value for CruiseActive.
                        DriverModeParameters.CruiseActive := some return value;
                end loop;
        end DriverMode;
begin
        -- initialize thread ports
        -- initialize thread
        AADL.Create_Thread( DriverMode'Address, DriverModeParameters'Address
                                        Periodic,  Ada.Real_Time.Milliseconds( 10 )  );
        loop
                -- hand over control to threads once system is initialized;
                -- may not need to call Await_Dispatch as processes do not
                -- necessarily get scheduled.
                AADL.Await_Dispatch;
                -- any local processing is done here
        end loop;
end CruiseControl;
```

Ada – Generic Solution (1/3)

```
generic
procedure DriverModeLogic (  BreakPedalPressed :   in        Boolean;
                             ClutchPedalPressed :  in        Boolean;
                             Activate           :  in        Boolean;
                             Cancel             :  in        Boolean;
                             OnNotOff           :  in        Boolean;
                             CruiseActive       :      out   Boolean );
```

Ada – Generic Solution (2/3)

```
with AADL; with CruiseControl_Ports;
procedure DriverModeLogic ( BreakPedalPressed  : in      Boolean;
                            ClutchPedalPressed : in      Boolean;
                            Activate           : in      Boolean;
                            Cancel             : in      Boolean;
                            OnNotOff           : in      Boolean;
                            CruiseActive       :     out Boolean )
is

   type DriverModeLogic_Parameter_Block is record
        BreakPedalPressed    :    Boolean;
        ClutchPedalPressed   :    Boolean;
        Activate             :    Boolean;
        Cancel               :    Boolean;
        OnNotOff             :    Boolean;
         CruiseActive        :    Boolean;
   end record;
```

Ada – Generic Solution (3/3)

```
         DriverModeLogic_Paraemeters : DriverModeLogic_Parameter_Block
                                       := (  BreakPedalPressed,
                                             ClutchPedalPressed,
                                             Activate,
                                             Cancel,
                                             OnNotOff,
                                             CruiseActive );
      procedure DriverModeLogic_Thread is
      begin
         loop
            AADL.Await_Dispatch;
            -- do work, e.g., generate a return value for CruiseActive.
         end loop;
      end DriverModeLogic_Thread;
begin – DriverModeLogic
      AADL.Create_Thread(
         DriverModeLogic_Thread'Address
         DriverModeLogic_Parameters'Address,
         Periodic,
         Ada.Real_Time.Milliseconds( 10 ) );
end; -- DriverModeLogic
```

Ada – Generic Instantiation

```
with AADL; with CruiseControl_Ports; with DriverModeLogic;
procedure CruiseControl is
    procedure DriverMode is new DriverModeLogic;
begin
        -- initialize thread component by calling the corresponding procedure
    DriverMode(  CruiseControl_Ports.BreakPedalPressed,
                 CruiseControl_Ports.ClutchPedalPressed,
                 CruiseControl_Ports.Activate,
                 CruiseControl_Ports.Cancel,
                 CruiseControl_Ports.OnNotOff,
                 CruiseControl_Ports.CruiseActive );
    loop
        -- hand over control to threads once system is initialized;
        -- may not need to call Await_Dispatch as processes do not
        -- necessarily get scheduled.
        AADL.Await_Dispatch; --- any local processing is done here
    end loop;
end CruiseControl;
```

Programming Language Guidelines

Ada to AADL

```
procedure Date_Book is
    type Month_Name is ( Jan, Feb, Mar, … );
    subtype Day_Range is Integer range 1 .. 31;
    subtype Year_Range is Integer range 1900 .. 2326;
    type Date is
        record
            Day   : Day_Range;
            Month : Month_Name;
            Year  :  Year_Range;
        end record;
    …
    Tomorrow, Yesterday : Date;
    …
end Date_Book;
```

AADL Package Standard - 1

```
package STANDARD
        public
                data Boolean
                end Boolean;

                data Integer
                end Integer;

                data Natural
                properties
                        Language_Support::Integer_Range => 0 .. value(Max_Aadlinteger);
                end Natural;

                data Positive
                properties
                        Language_Support::Integer_Range => 1 .. value(Max_Aadlinteger);
                end Positive;

                data Float
                end Float;

                data Character
                end Character;

                data Wide_Character
                end Wide_Character;
```

AADL Package Standard - 2

```
    data String
    end String;

    data Wide_String
    end Wide_String;

    data Duration
    end Duration;

    data Exception
    end Exception;

    data implementation Exception.Constraint_Error
    end Exception.Constraint_Error;

    data implementation Exception.Program_Error
    end Exception.Program_Error;

    data implementation Exception.Storage_Error
    end Exception.Storage_Error;

    data implementation Exception.Tasking_Error
    end Exception.Tasking_Error;
end STANDARD;
```

Programming Language Guidelines

Ada to AADL

```
data Month_Range
properties
    Language_Support:: String_List_Value => ("Jan", "Feb", "Mar", …);
end Month_Range;

data Date
properties
   Type_Source_Name => "Date_Book.Date";
end Date;
```

Programming Language Guidelines

Ada to AADL

```
data implementation Date.others
subcomponents
    Day    : STANDARD::Integer
                 { Language_Support::Integer_Range => 1 ..31 };
    Month : Month_Range;
    Year   : STANDARD::Integer
                 { Language_Support::Integer_Range => 1900 .. 2326 };
end Date.others;

system Date_Book
end Date_Book;

system implementation Date_Book.imp1
subcomponents
    Tomorrow : data Date.others;
    Today       :  data Date.others;
end Date_Book.imp1;
```

Package AADL.ads

```
package AADL is
   type Event is access String;
   function "+" (S : String) return Event;

   type Error is access String;
   function "+" ( S : String ) return Error;

   procedure Create_Thread( Thread : System.Address; Thread_Parameters : System.Address;
                            Dispatch_Protocol : Thread_Dispatch_Protocol );
   procedure Create_Thread( Thread : System.Address; Thread_Parameters : System.Address;
                            Dispatch_Protocol : Thread_Dispatch_Protocol; Period : Ada.Real_Time.Time_Span

   procedure Await_Dispatch;

   procedure Get_Resource;
   procedure Release_Resource;

   procedure Raise_Event ( EVID : in Event );
   procedure Raise_Error ( ERID : in Error );

   procedure Thread_Dispatch_Status ( Thread : System.Address );
   procedure Port_Dispatch_Status ( Port : System.Address);
   procedure Connection_Status ( Port : System.Address);

   procedure Stop_Process;
   procedure Abort_Process;

   procedure Stop_Processor;
   procedure Abort_Processor;
   procedure Stop_System;
   procedure Abort_System;
end AADL;
```

AADL.h

```
/* AADL.h */

typedef char *Event;
typedef struct { void *P1; void *P2; } Parameters;

void Create_Thread( void Thread(void *Arg), void *Parms );
void Await_Dispatch( void );

void Get_Resource( void );
void Release_Resource( void );

void Raise_Event( Event EVID );
void Raise_Error( Event IRIS );

void Dispatch_Status( void );
void Connection_Status( void );

void Stop_Processor( void );
void Abort_Processor( void );

void Stop_System( void );
void Abort_System( void );
```

Programming Language Guidelines

Ada Ravenscar & AADL

- The *AADL* ***loaded(Process)*** transition corresponds to the loading of the Ada program onto the processor.
- The AADL performing thread *initialization* state corresponds to the Ada program elaboration, system initialization, and task activation. All of the activities associated with organizing Ada tasks shall be done during this state of execution. Thus, those tasks that are part of different modes of operation will be moved into the Ada suspended state; this state corresponds to the AADL suspended awaiting mode state.
- The tasks that comprise the set of tasks to begin execution as part of the initial execution mode would move into the Ada ready-to-run state. This state corresponds to the AADL suspended awaiting dispatch state.
- When a mode change occurs, the currently executing task set will be suspended onto appropriate suspension objects corresponding to their mode of execution. This activity corresponds to the AADL performing thread *deactivation* state. Similarly, the triggering events that are needed to activate the task set corresponding to the new execution mode shall be executed. This activity corresponds to the AADL performing thread *activation* state.
- The AADL performing thread *finalization* state is an implementation defined state for an Ada Ravenscar application that handles abnormal task termination.

Tutorial Outline

- Background & Introduction
- Introduction to AADL
- AADL Development Environment
- **AADL in Use**
  - Modeling Paradigms
  - Early Users
- Summary and Conclusions

What Is Involved In Using The AADL?

- Specify software & hardware system architectures.
- Specify component interfaces and implementation properties.
- Analyze system timing, reliability, partition isolation.
- Tool-supported system integration.
- Verify source code compliance & middleware behavior.

Model and analyze early and throughout the product lifecycle.

AADL Analysis & Design Methodology

- Here we use the AADL as analysis & design methodology on an existing system
- AADL employs
  - Components with precisely defined semantics.
  - Explicit interactions.
  - Decomposition.
  - Separation of concerns.
- Pattern-based architecture analysis approach
  - Uses design patterns in analysis.
  - Identifies systemic problems early.
  - Enables the right choices with confidence.
  - Provides analysis-based decisions.

Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering

Avionics Systems

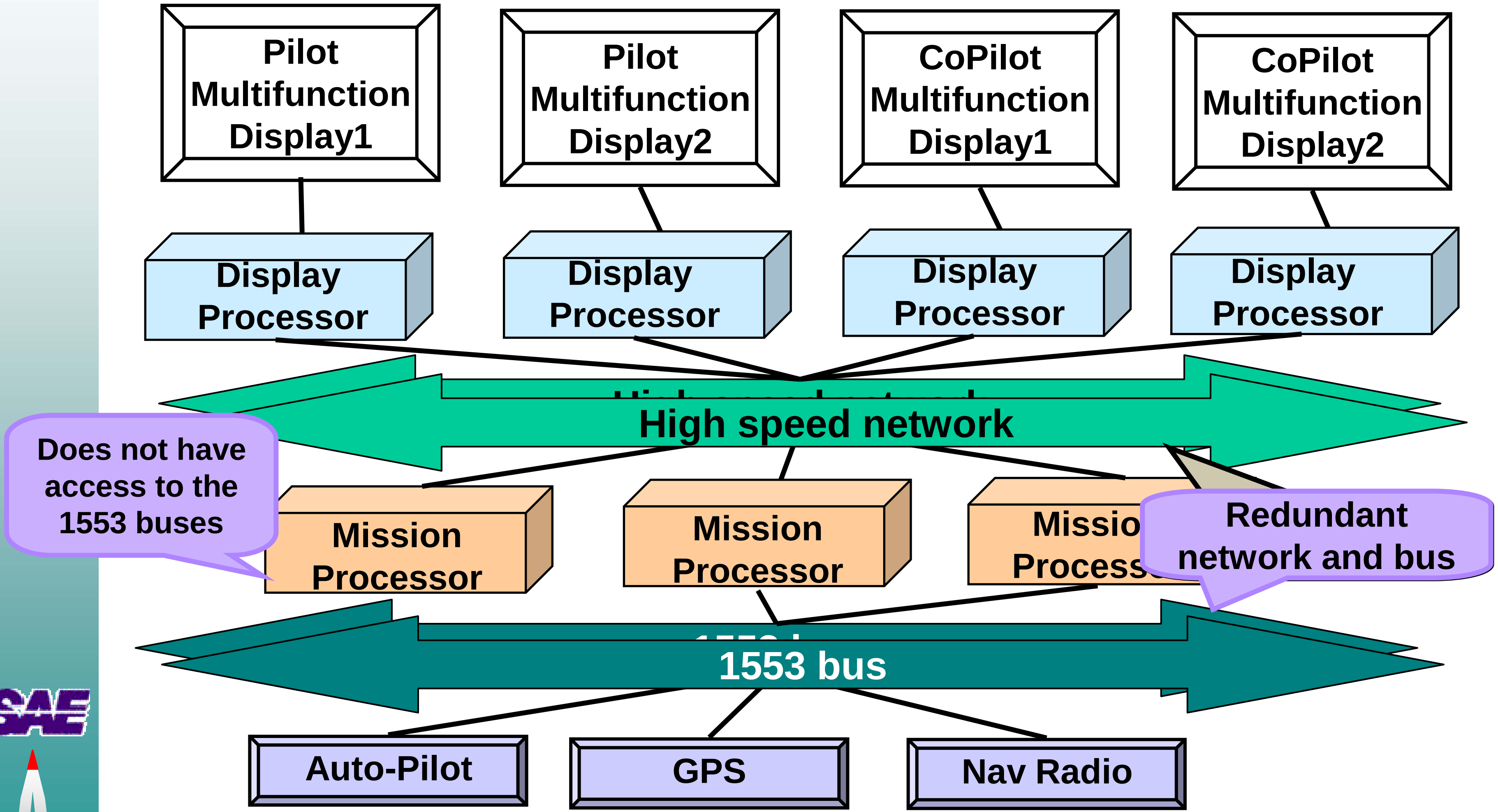
- Embedded avionics system designs are evolving to
  - Integrated systems from federated ones.
  - Predictable preemptive scheduling.
  - Extensible system architectures.
- There are distinct perspectives in the design
  - Control and domain engineers.
  - Application software engineers.
  - System software engineers.
- In the remainder of this tutorial we consider
  - A representative avionics system.
  - Design within the context of the AADL.
  - The distinct perspectives involved.
  - Issues associated with the evolution of avionics systems.

Avionics System Example

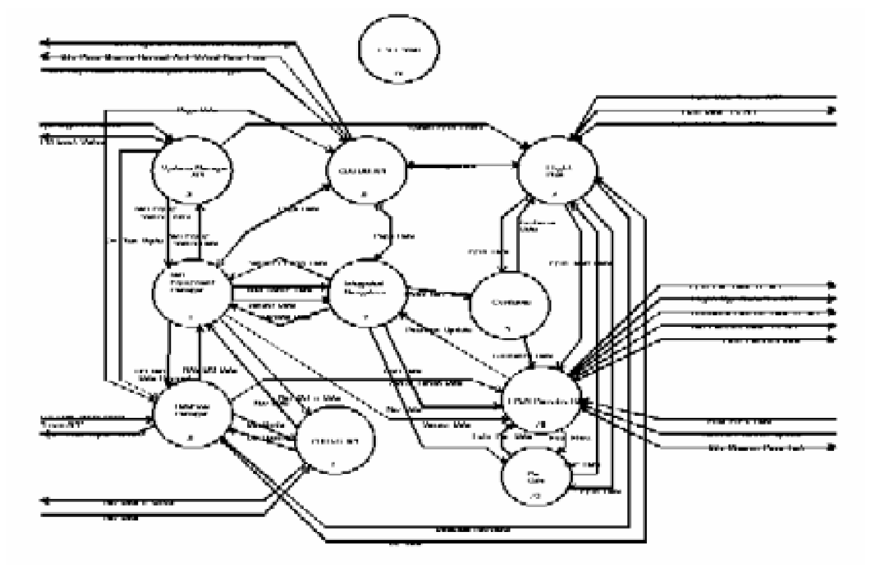
- Reflects current design approaches including
  - Time and space partitioning
  - Shared memory & port communication
  - Cyclic executive & preemptive scheduling
  - Deterministic communication
  - Distributed system with legacy hardware
  - Fault tolerance and reconfiguration
  - Efficient execution and footprint

Focus on performance-critical system properties

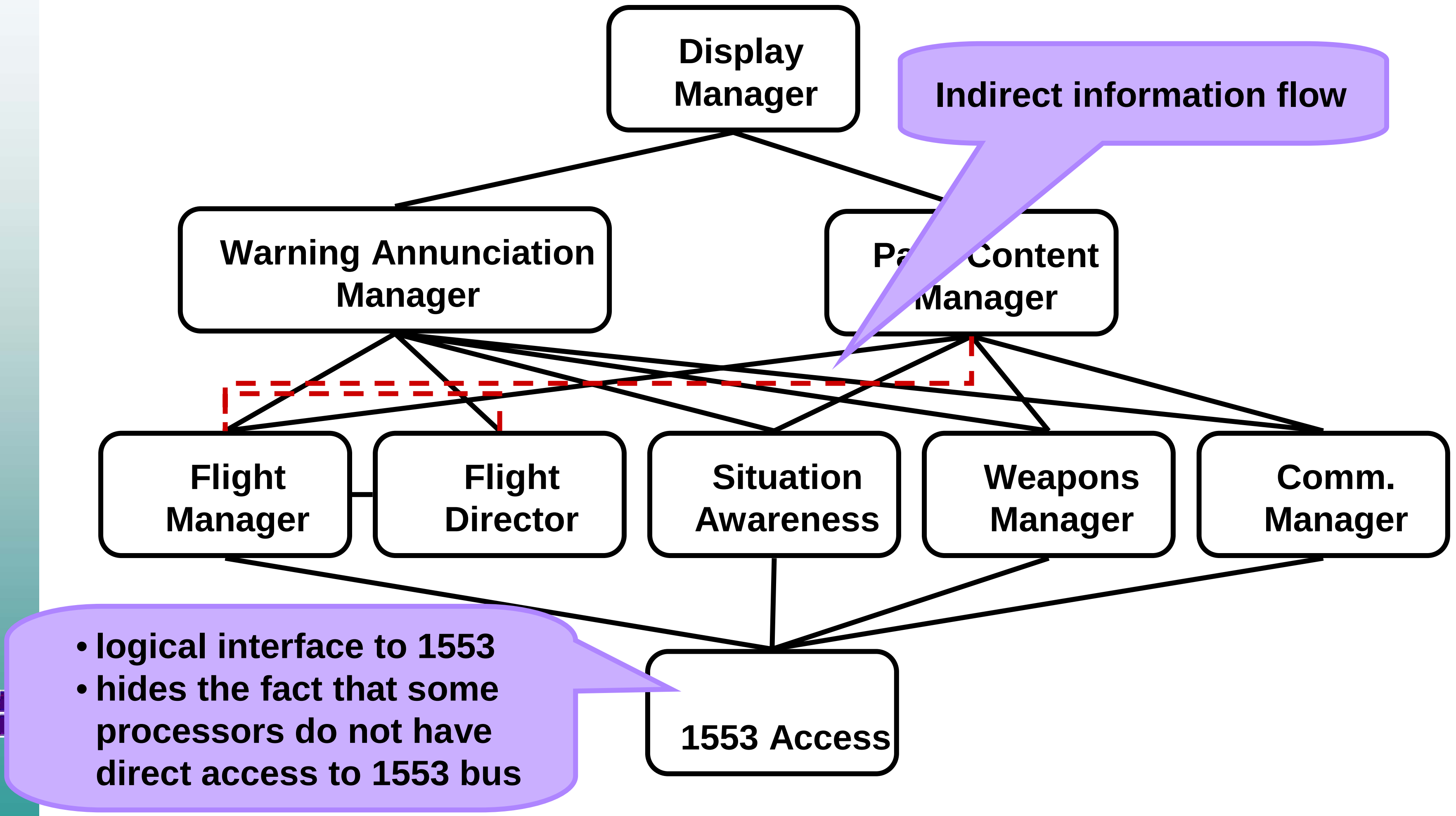
Sample Avionics System Hardware Configuration


Pilot Multifunction Display1
Pilot Multifunction Display2
CoPilot Multifunction Display1
CoPilot Multifunction Display2
Display Processor
Display Processor
Display Processor
Display Processor
High speed network
Does not have access to the 1553 buses
Mission Processor
Mission Processor
Mission Processor
Redundant network and bus
1553 bus
Auto-Pilot
GPS
Nav Radio

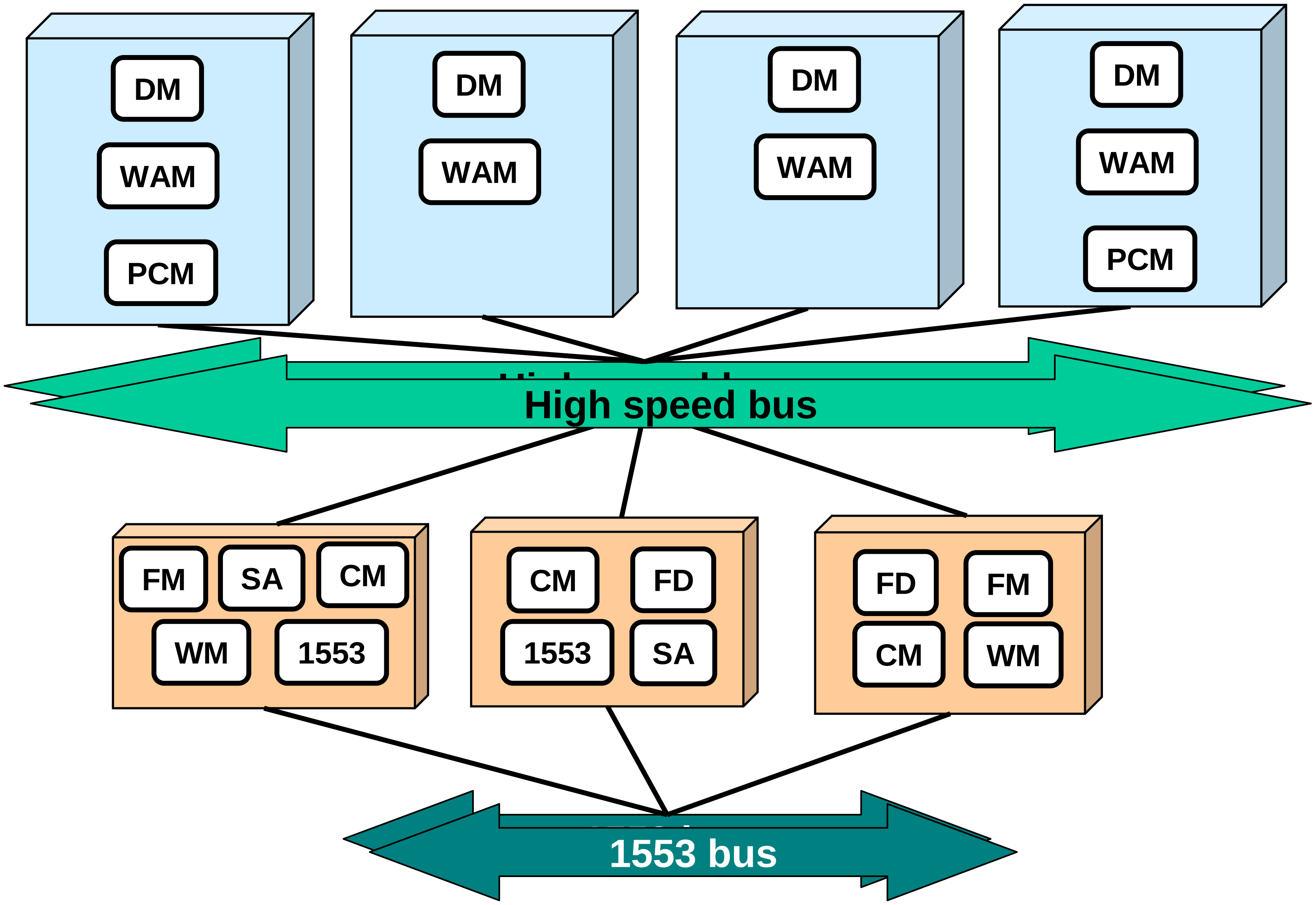
A Typical Context Diagram



Avionics Software Component Layers


Display Manager
Indirect information flow
Warning Annunciation Manager
Pa Content Manager
Flight Manager
Flight Director
Situation Awareness
Weapons Manager
Comm. Manager
1553 Access
• logical interface to 1553
• hides the fact that some processors do not have direct access to 1553 bus

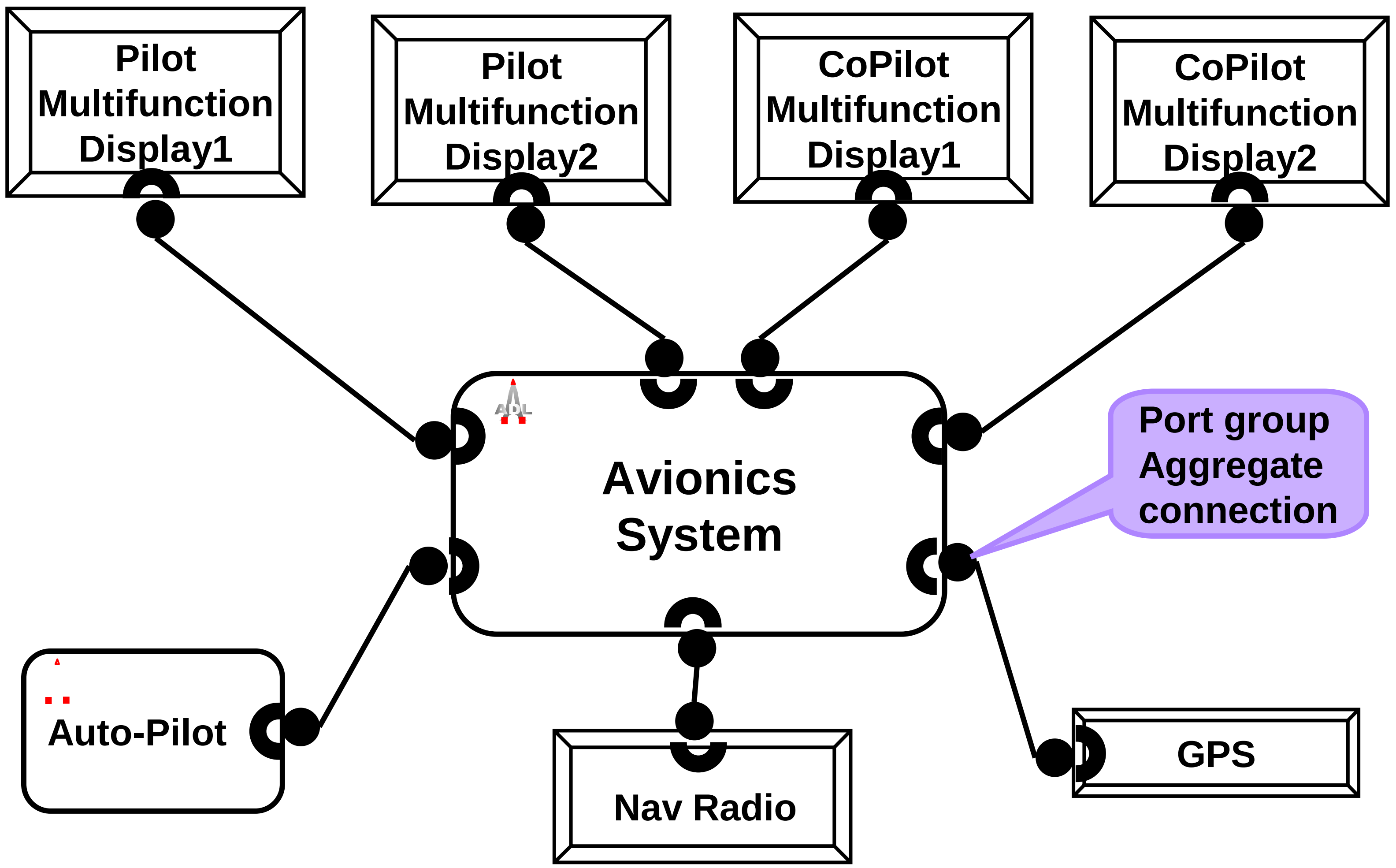
Typical Software to Hardware Mapping



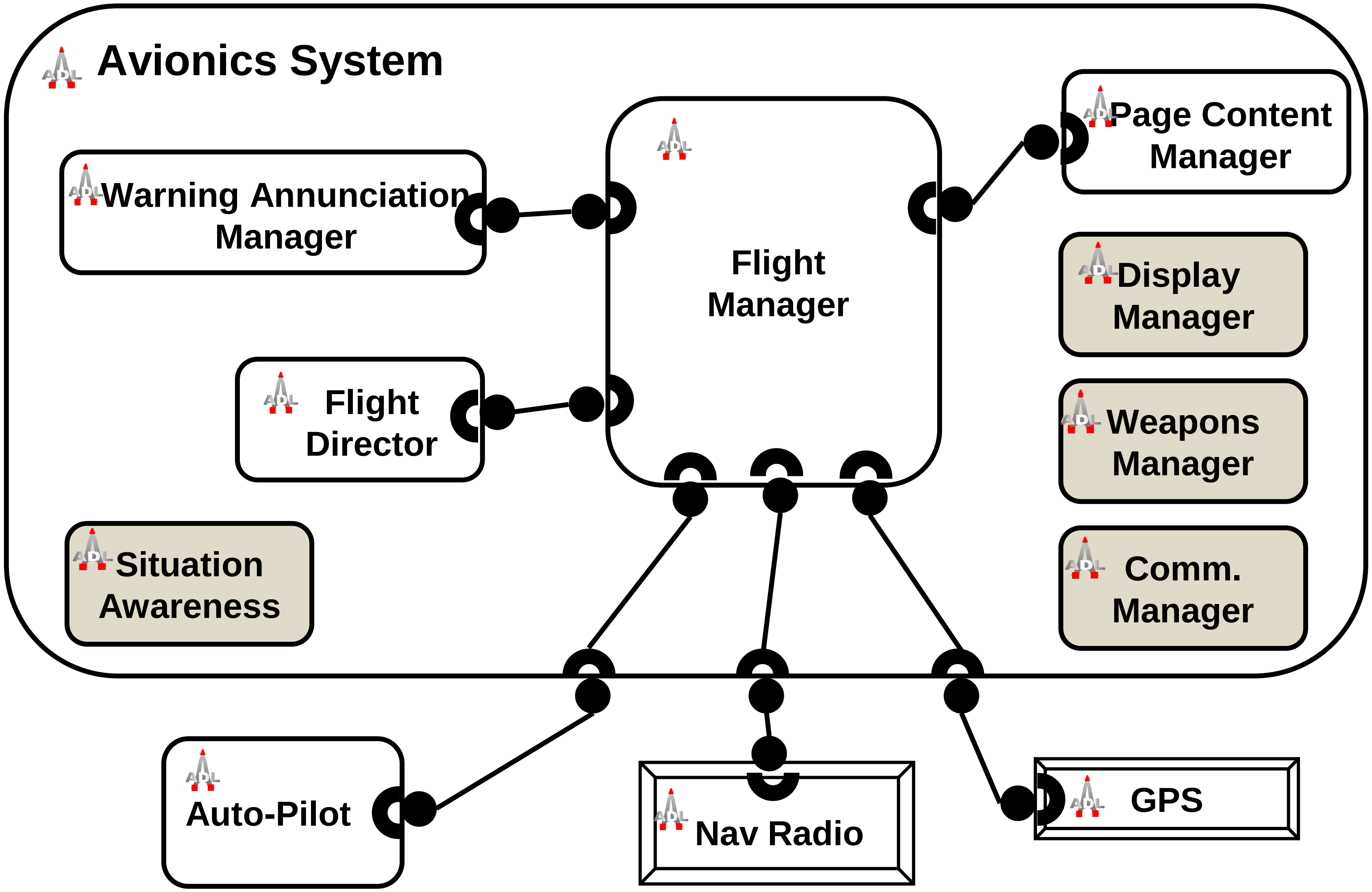
Observations: Hidden Information

- Multiple instances as separate components.
- Both dual and quad redundancy.
- Grouping of redundant instances.
- Documented in text.
- Difficult to understand and analyze.

AADL Avionics System Context Diagram

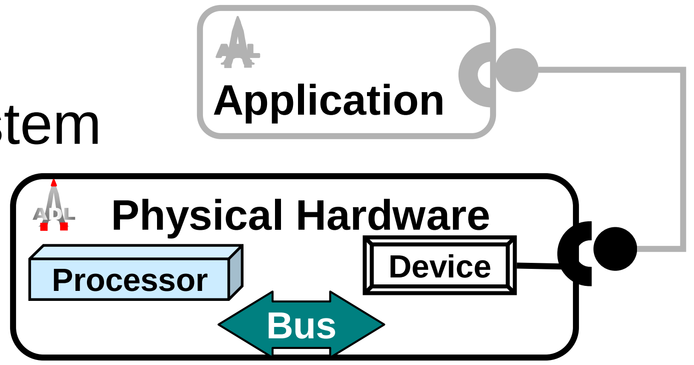

Pilot Multifunction Display1
Pilot Multifunction Display2
CoPilot Multifunction Display1
CoPilot Multifunction Display2
Avionics System
Port group Aggregate connection
Auto-Pilot
Nav Radio
GPS

AADL Flight Manager Context Diagram

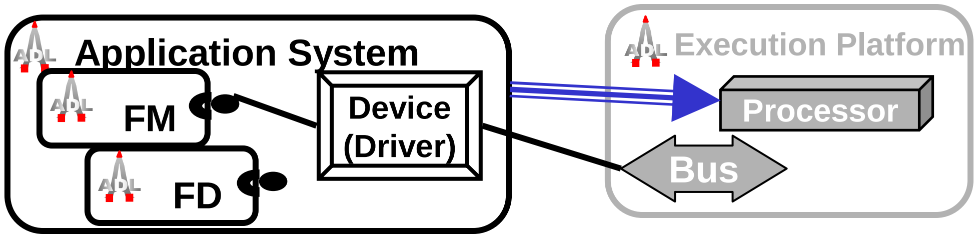


Perspectives on Devices

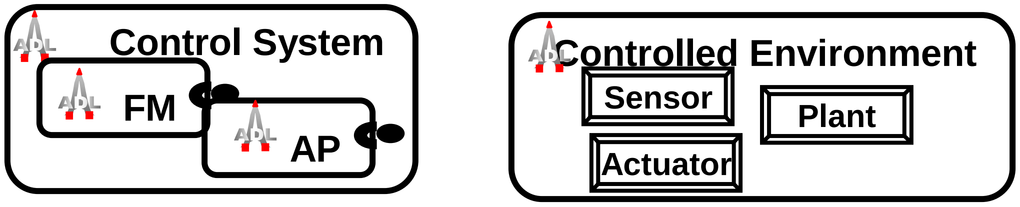
- Hardware Engineer
  - Device is part of physical system



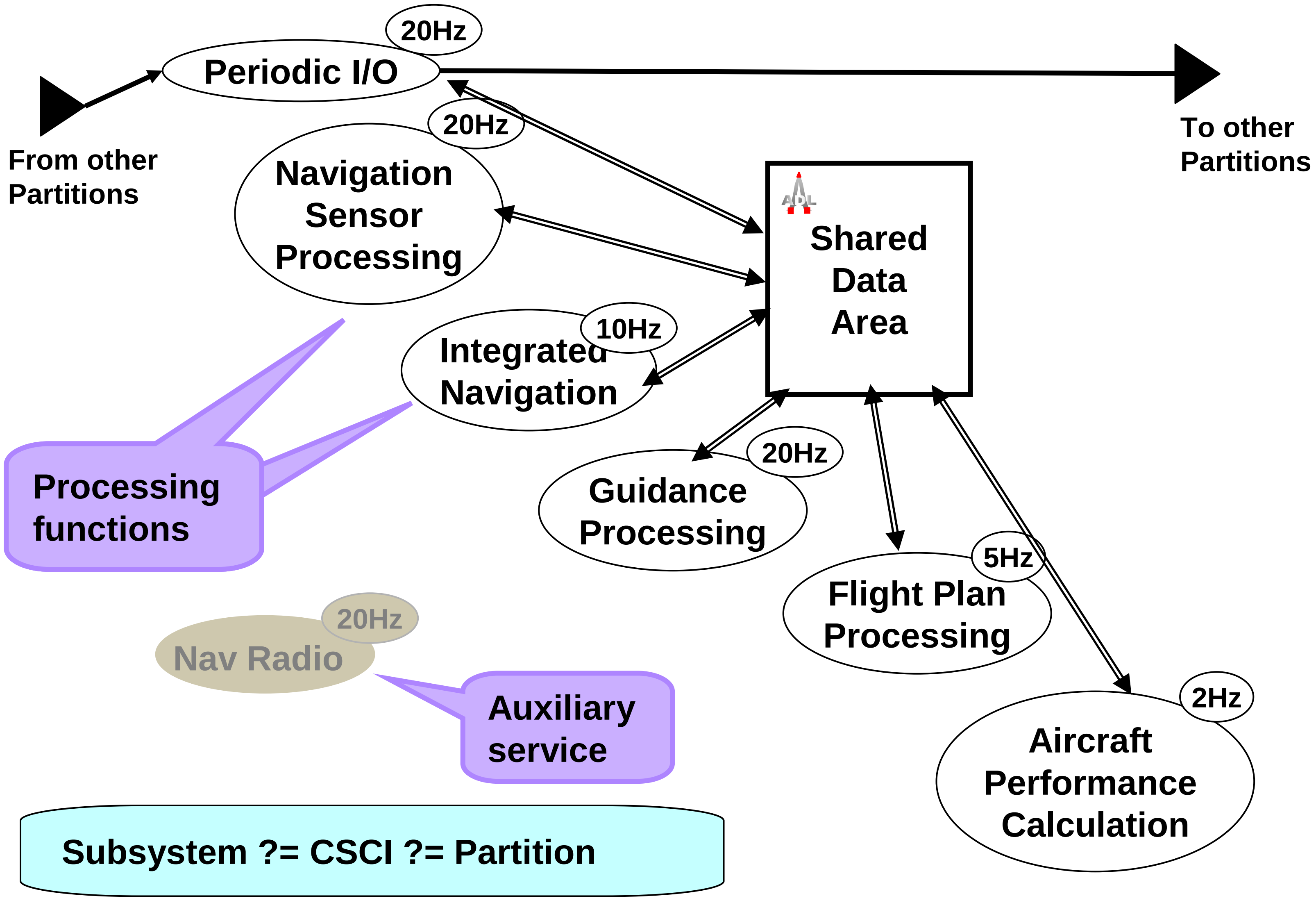
- Application developer
  - Device functionality is part of the application software



- Control Engineer
  - Device represents the plant being controlled



Flight Manager: Principal Functionality

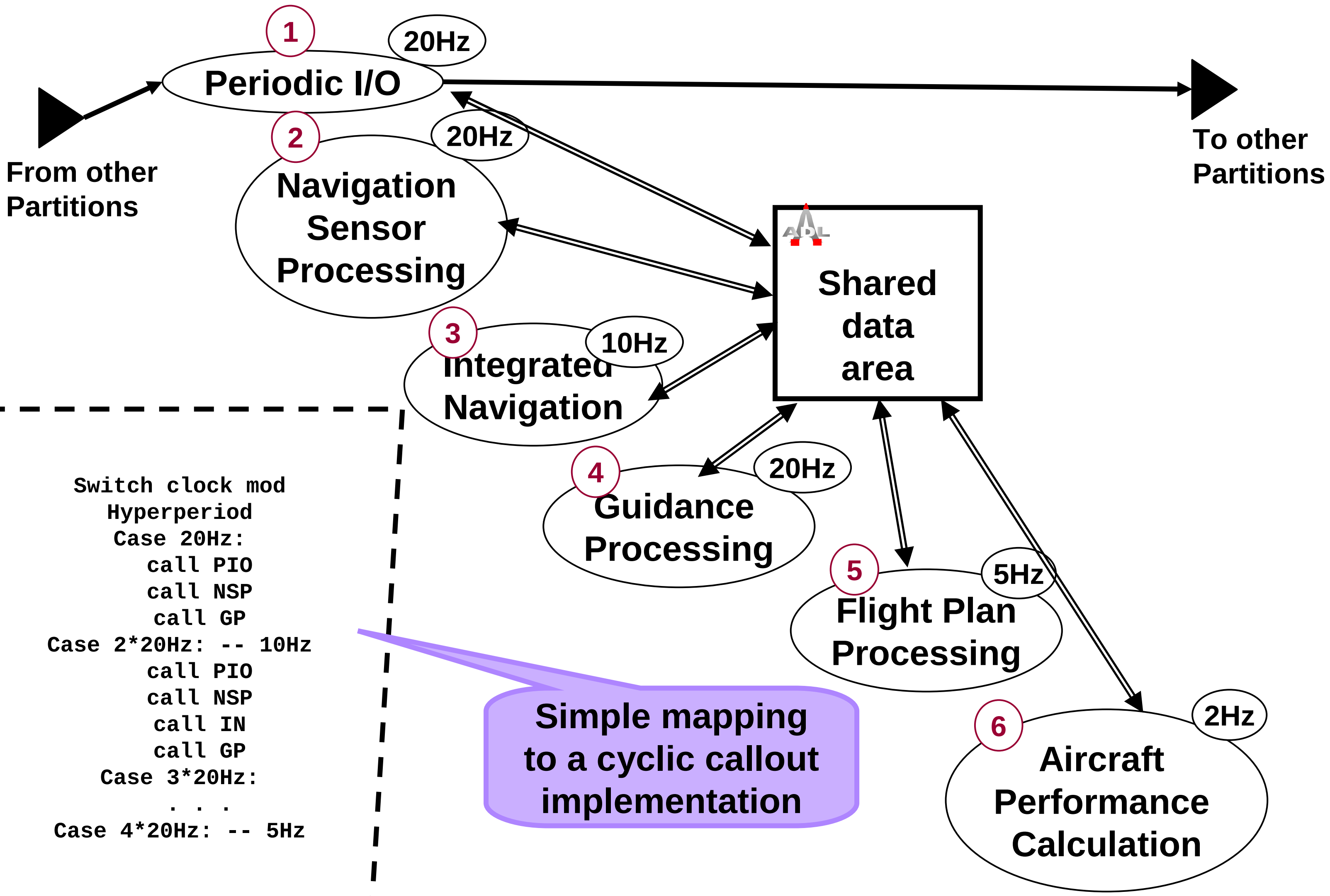


Preemptive Scheduling & Data Flow

- Preemptive scheduling & shared variables
- Data flow patterns
- Threads and port connections in AADL
- Efficient and correct implementations

Modernizing an embedded real-time system
From cyclic executive to preemptive scheduling
From shared variables to port communication

A Cyclic Executive Implementation



```
Switch clock mod
Hyperperiod
Case 20Hz:
    call PIO
    call NSP
    call GP
Case 2*20Hz: -- 10Hz
    call PIO
    call NSP
    call IN
    call GP
Case 3*20Hz:
    . . .
Case 4*20Hz: -- 5Hz
```

Preemptive Scheduling and Port Communication

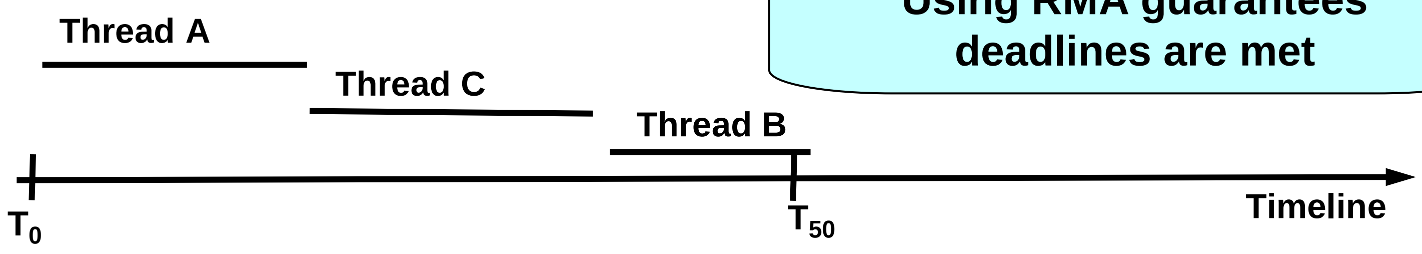
- Towards Rate-Monotonic Scheduling
  - Fixed priority preemptive scheduling based on rate-monotonic analysis (RMA)
  - Better resource utilization, more extensible architecture
- Towards port connections
  - Many shared variables can be converted to data port connections.
  - Port connections can be optimized using shared variables.
  - AADL connections provide explicit communication behavior.
- Scheduling and communication in a partition

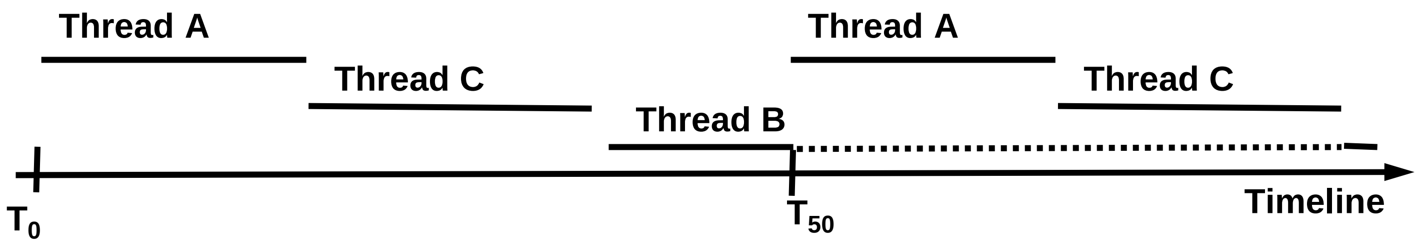
Benefit of Preemptive Scheduling

- Thread A: 20Hz 20ms
- Thread B: 10Hz 11ms
- Thread C: 20Hz 20ms
- Not schedulable with cyclic executive

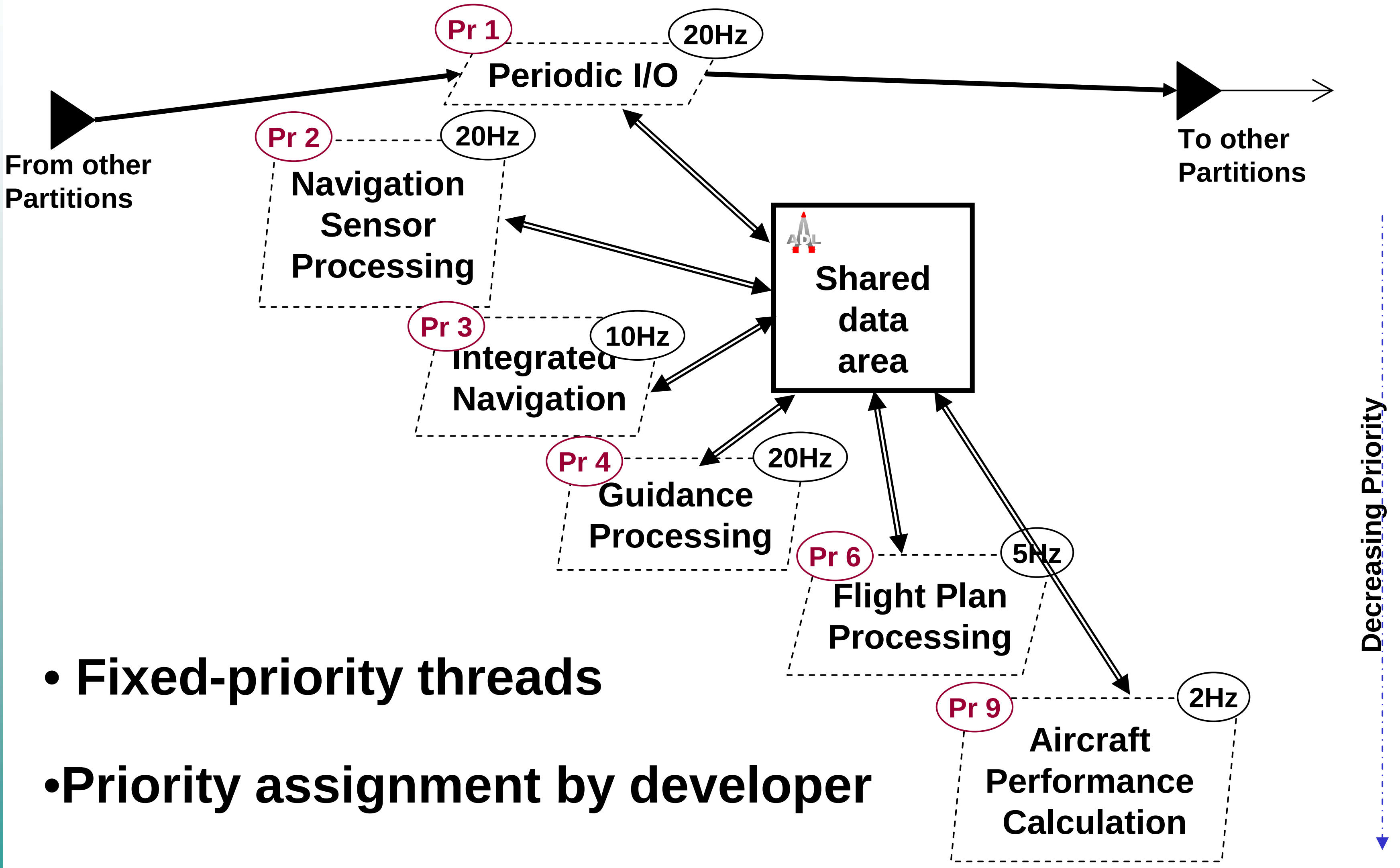
Schedulable with
- **preemption**
- **correct priority (EDF, RMA)**

Using RMA guarantees deadlines are met





A Naïve Thread-based Design



- Fixed-priority threads
- Priority assignment by developer

Design Decisions Taken

- Shared variable communication within partition
  - Achieve efficient resource utilization
  - Accommodate legacy code
- Preemptive fixed-priority thread scheduling
  - Used Schedulability analysis (RMA) to confirm schedulability
  - Benefit of more flexible system and efficient resource usage
- Priority assignment for precedence ordering to achieve desired flow
  - Needed because of shared data area
- Periodic I/O
  - Port data to shared data area (support legacy code)
  - Deterministic communication
  - Output packaging of time consistent data set

Impact of Decisions

Promoted by AADL

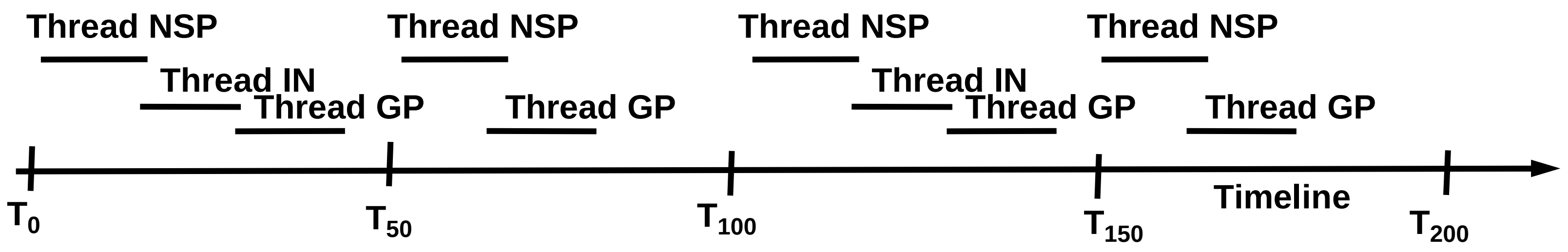
- Rate-Monotonic scheduling analysis (RMA)
  - Priority inversion by 10Hz thread
- 10Hz thread (IN) may prevent lower priority 20Hz from running if IN's deadline > .05 sec
  - Assume IN pre-period deadline of .05 sec or less
- 10Hz thread preemption: shared data inconsistency and non-deterministic sampling
  - Atomic shared data read/write operations assures data consistency
  - pre-period deadline of .05sec assures determinism
- .05 sec pre-period deadline for 10Hz
  - Threads must complete at .05 sec
  - Deadline monotonic scheduling analysis

Effectively cyclic executive

Expressed in AADL

Observations on Scheduling

- Situation: Preemptive scheduling and shared data
- Communication via shared data for efficiency
- Precedence ordering to achieve desired flow
- Priority assignment to achieve precedence ordering
- Causes priority inversion
- Requires pre-period deadline
- Leads to cyclic executive like scheduling behavior at expense of preemptive scheduling overhead

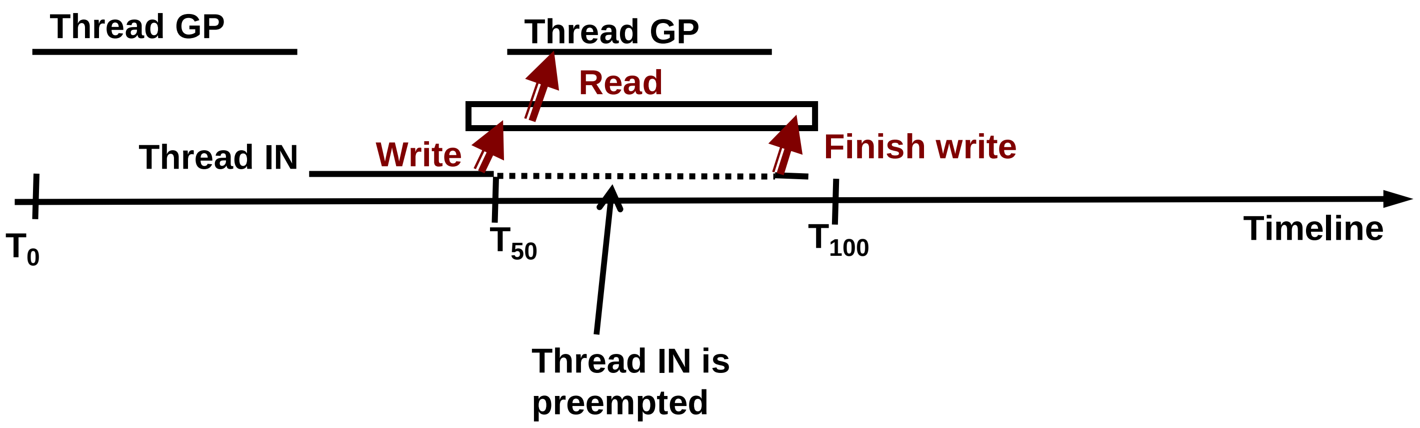


Consistent Communication of State

- Situation: Communication by shared variables
- Issue: preemptive scheduling introduces concurrency
- Read/write of consistent data values
- Deterministic read/write order

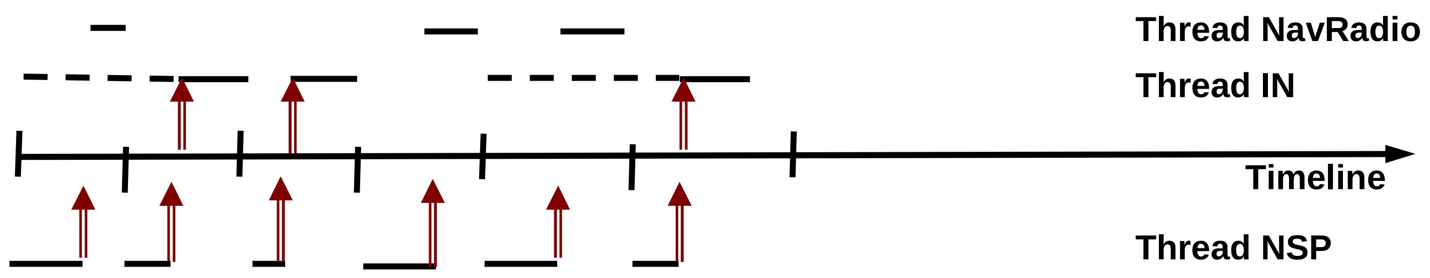
Need for Atomic Read/Write

- Thread C updates state in multiple write operations
- Thread A preempts thread C while write is in progress
- Techniques for atomic read/write
  - Hardware supported
  - Locking
  - Temporary non-preemption



Deterministic State Communication

- Write & read (send & receive) performed by application
- Variation in actual write & read time
- Preemption & completion time affect receiver sampling
- Example:
  - Thread A -> thread B
  - Thread A twice rate of thread B
  - Thread C preemption adds to variation
- Desired sampling pattern 2X: n, n+2, n+4 (2,2,2,…)
- Worst-case sampling pattern: n, n+1, n+4 (1,3,…)



State Communication in AADL

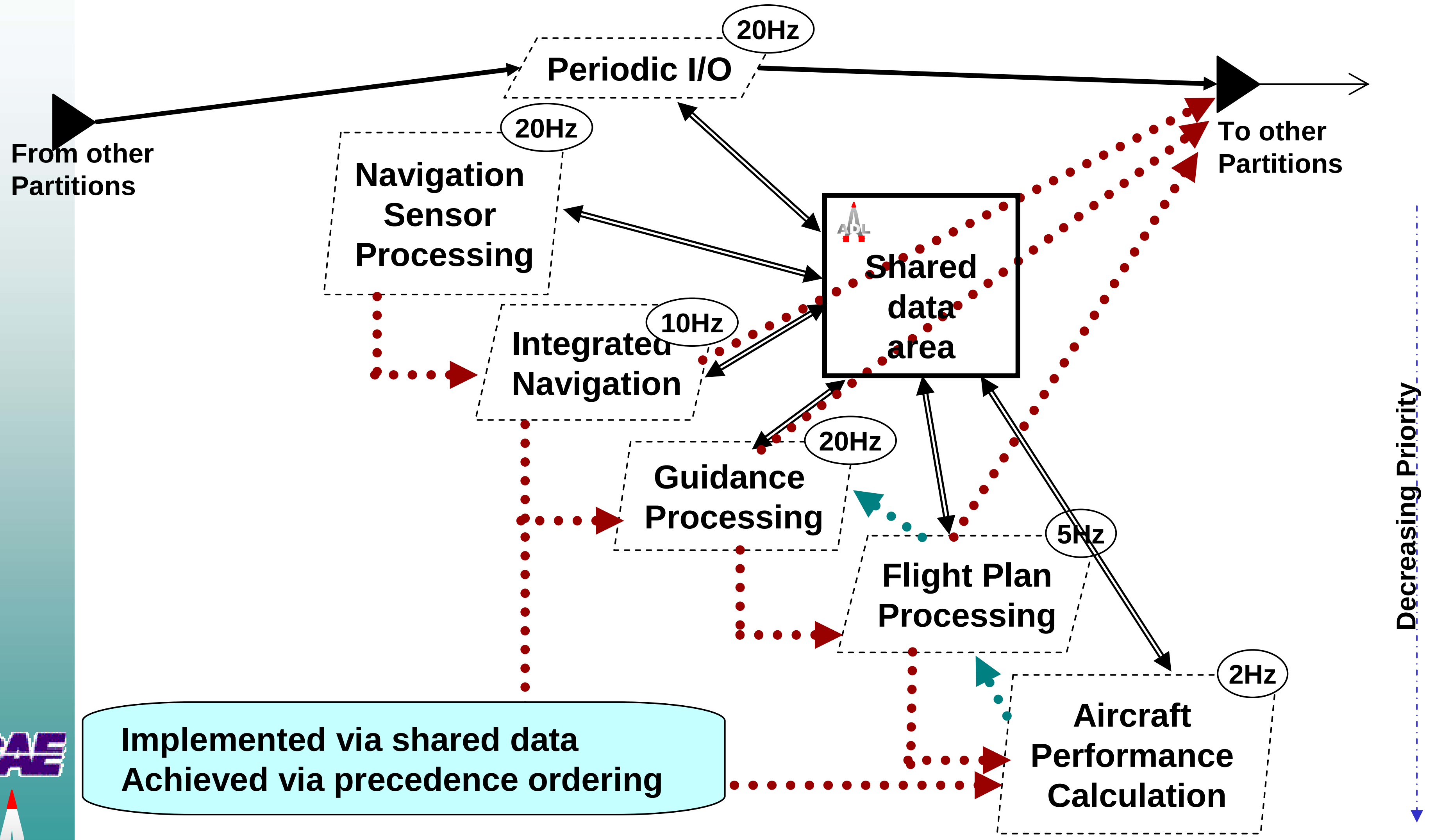
- AADL precisely specifies the time of data transfer
- AADL semantics assure consistent and deterministic state communication
- Implementation may require double buffering
- Requires runtime system support

Preemptive Scheduling & Data Flow

- Preemptive scheduling & shared variables
- Data flow patterns
- Threads and port connections in AADL
- Efficient and correct implementations

Modernizing an embedded real-time system
From cyclic executive to preemptive scheduling
From shared variables to port communication

Intended Data Flow

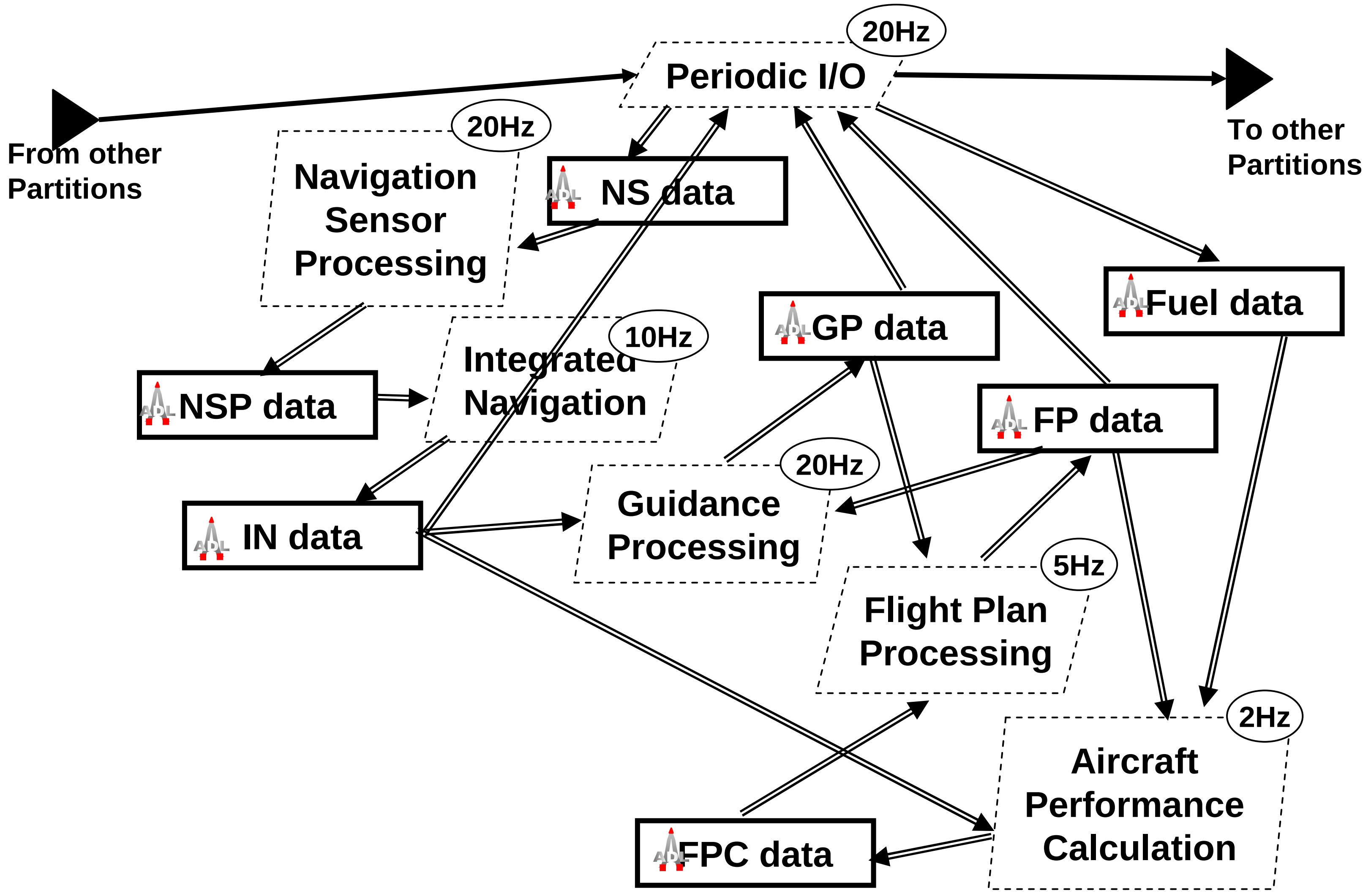


Implemented via shared data
Achieved via precedence ordering

Observations on Data Flow

- Data flow described in text and tables.
- Phase delayed data flow hidden in text.
- Output phase delay embedded in periodic I/O thread.
- Application component code contains communication mechanism decision.

Shared Data Area Detailing



Observations on Data Detailing

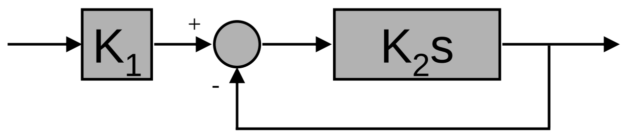
- More explicit documentation of information flow is needed in this example
- Periodic I/O as focal point
  - Loss of cross-partition flow information
- It is important to explicitly document the flow of data and control information.
- Towards thread interface specification
- Explicit documentation of all interaction points
- Data vs. Message Communication Choice
  - State vs. state change
  - Control system signal stream -> data ports

Preemptive Scheduling & Data Flow

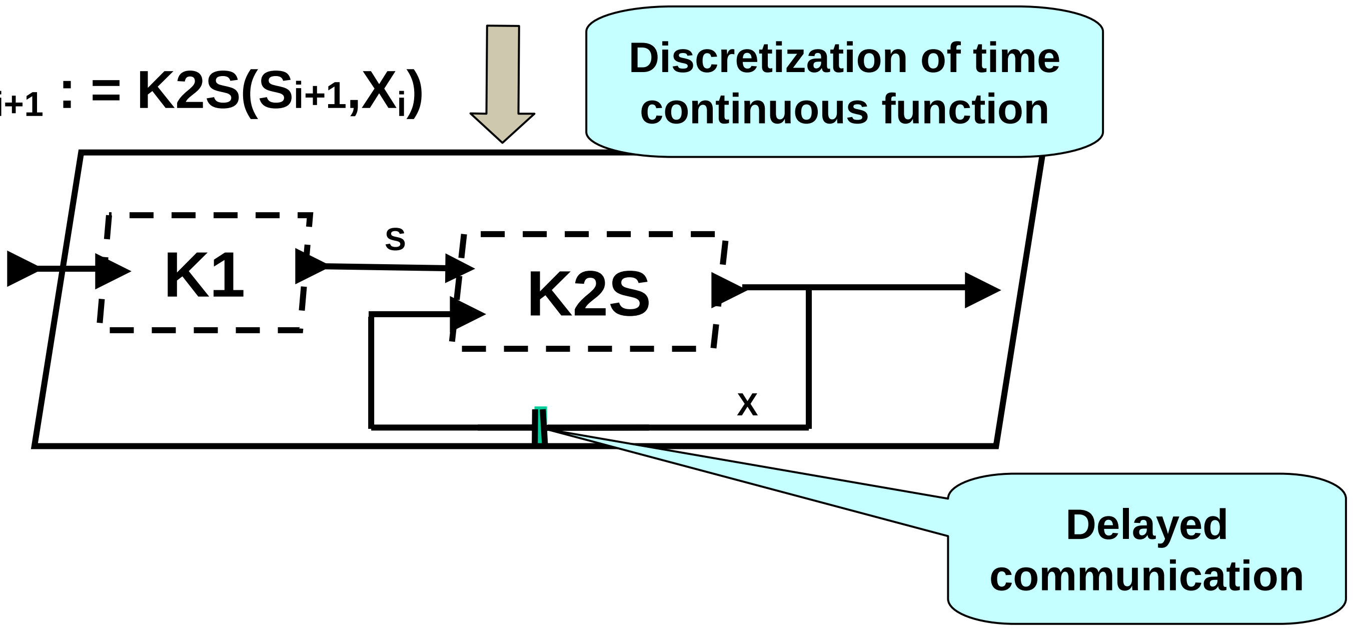
- Preemptive scheduling & shared variables
- Data flow patterns
- Threads and port connections in AADL
- Efficient and correct implementations

Modernizing an embedded real-time system
From cyclic executive to preemptive scheduling
From shared variables to port communication

A Control System Example



$X_{i+1} := K2S(S_{i+1}, X_i)$



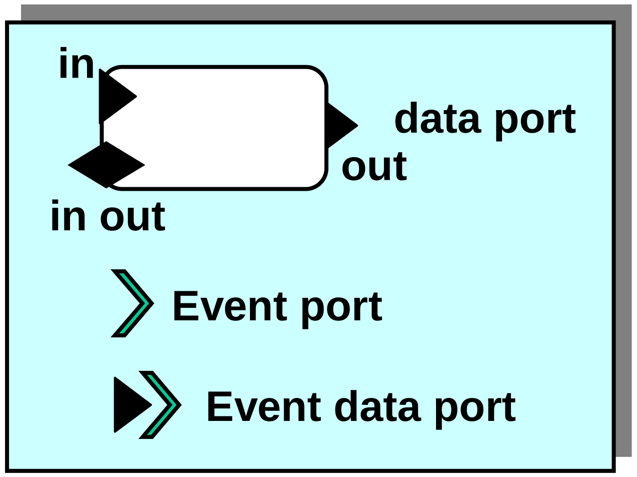
Used for iterative/recurrence functions (e.g. integrators, filters)

Observations: Control Processing

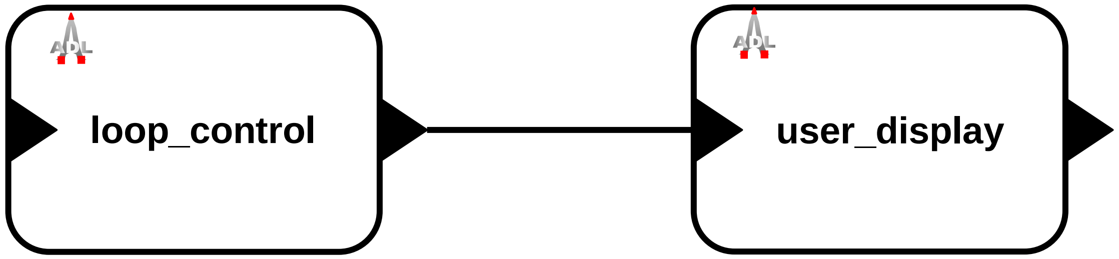
- AADL supports mid-frame communication & single sample delay..
- Allows explicit specification of communication timing which correlates to the real-time control domain.
- AADL analysis identifies mid-frame communication cycles
- AADL determines schedulability of implementation.
- Opens dialogue between application developers (control and systems engineers) and software system engineers.

Ports and Connections

- Port (flow of data and control)
  - data (state)
  - event-data (queued)
  - event (queuing possible)



Data type checking on connected ports.
Checking of data value & stream characteristics.



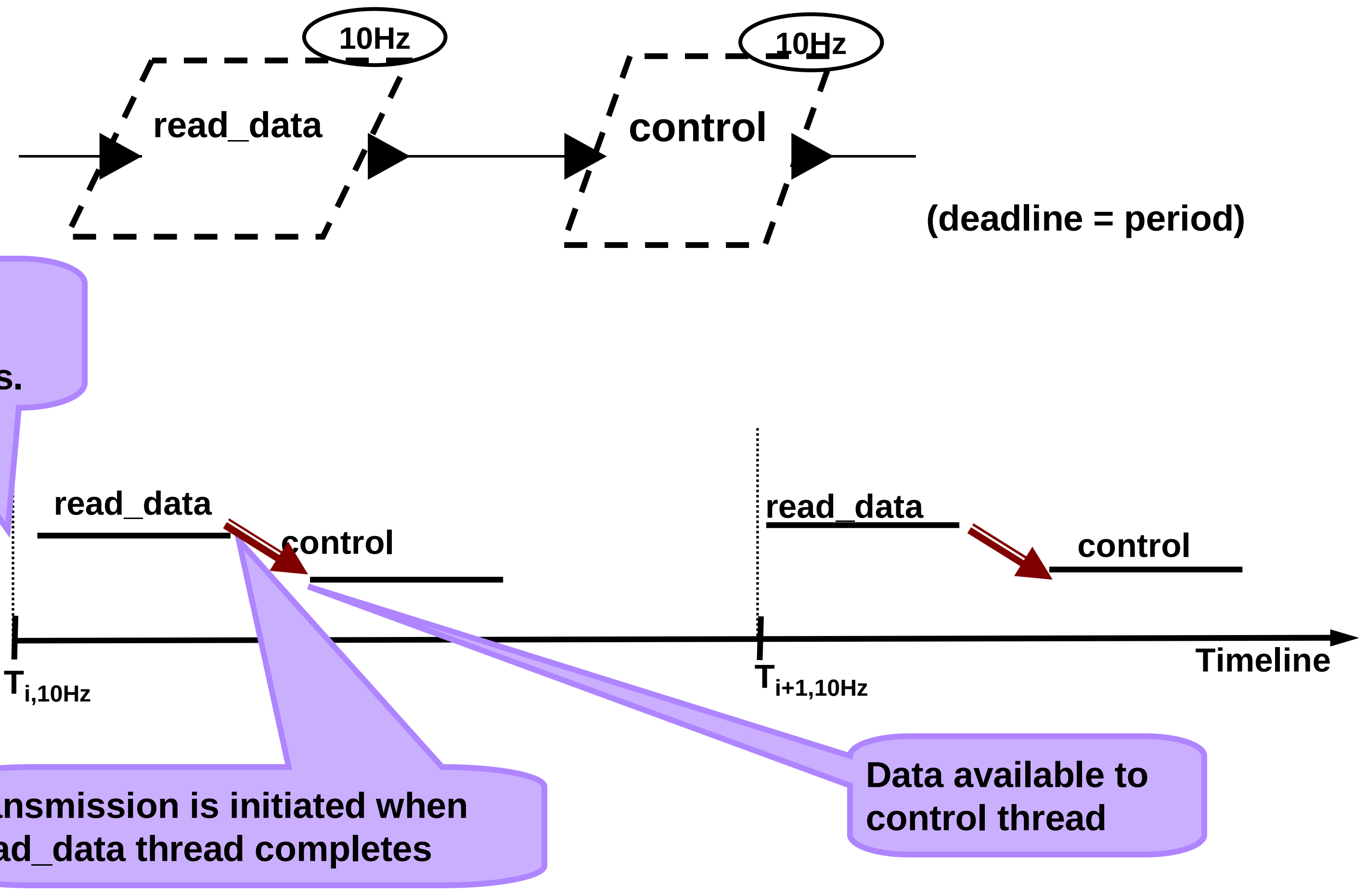
: **out data** boat_speed : **in data** boat_speed

Immediate Data Connections

Connection (immediate)

- Immediate data connections
  - initiated when source thread completes
  - start of receiving thread execution is delayed until completion of the sending thread
  - data transfer occurs only when the dispatch for both the sending and the receiving threads are logically simultaneous
- Effect of immediate communication
  - all threads see input from same sample period
  - constrain order of execution and communication
- Restrictions: No cycles

Immediate Communication

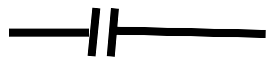


Vertical leveling does not imply different priorities.

transmission is initiated when read_data thread completes

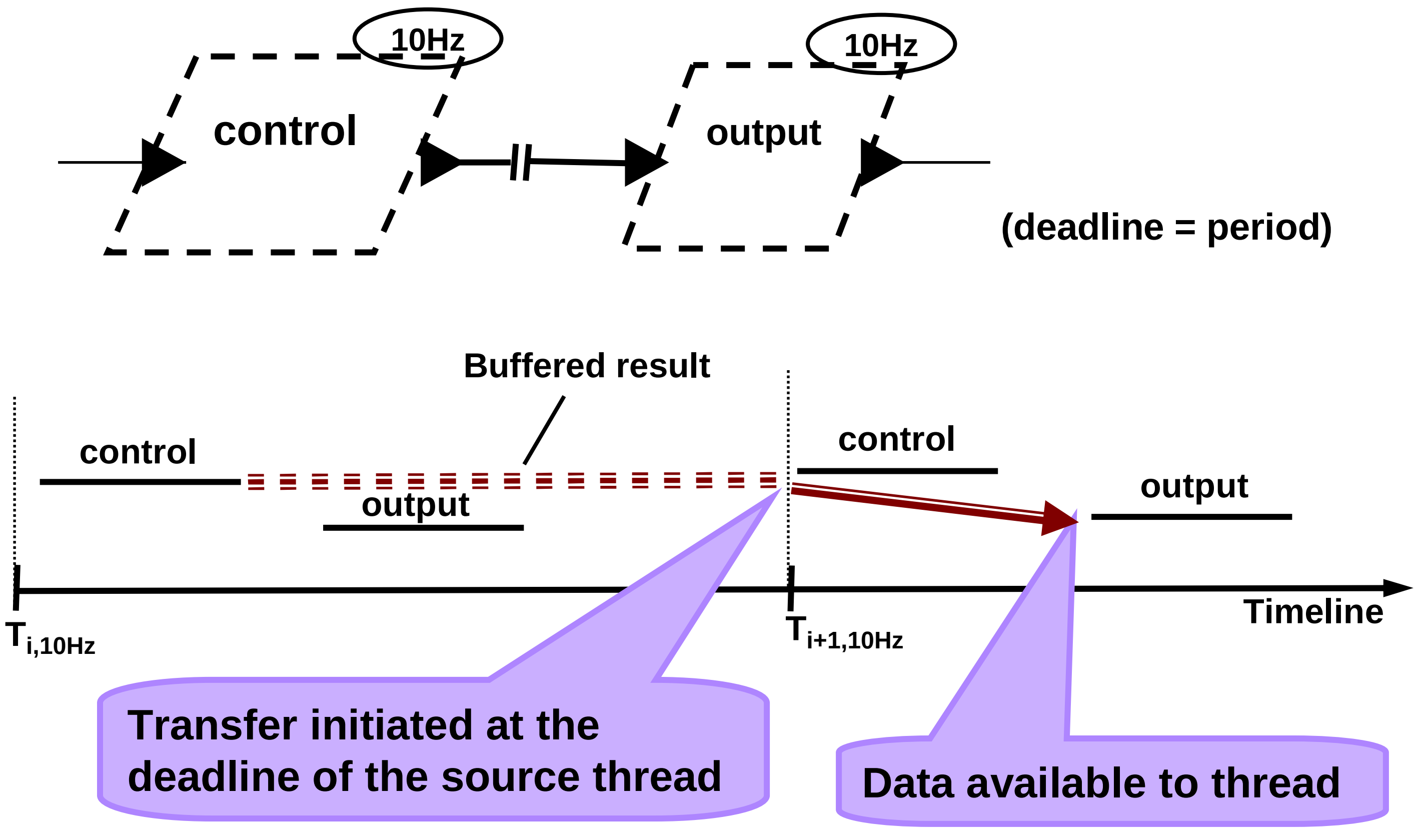
Data available to control thread

Delayed Connections-Periodic Threads

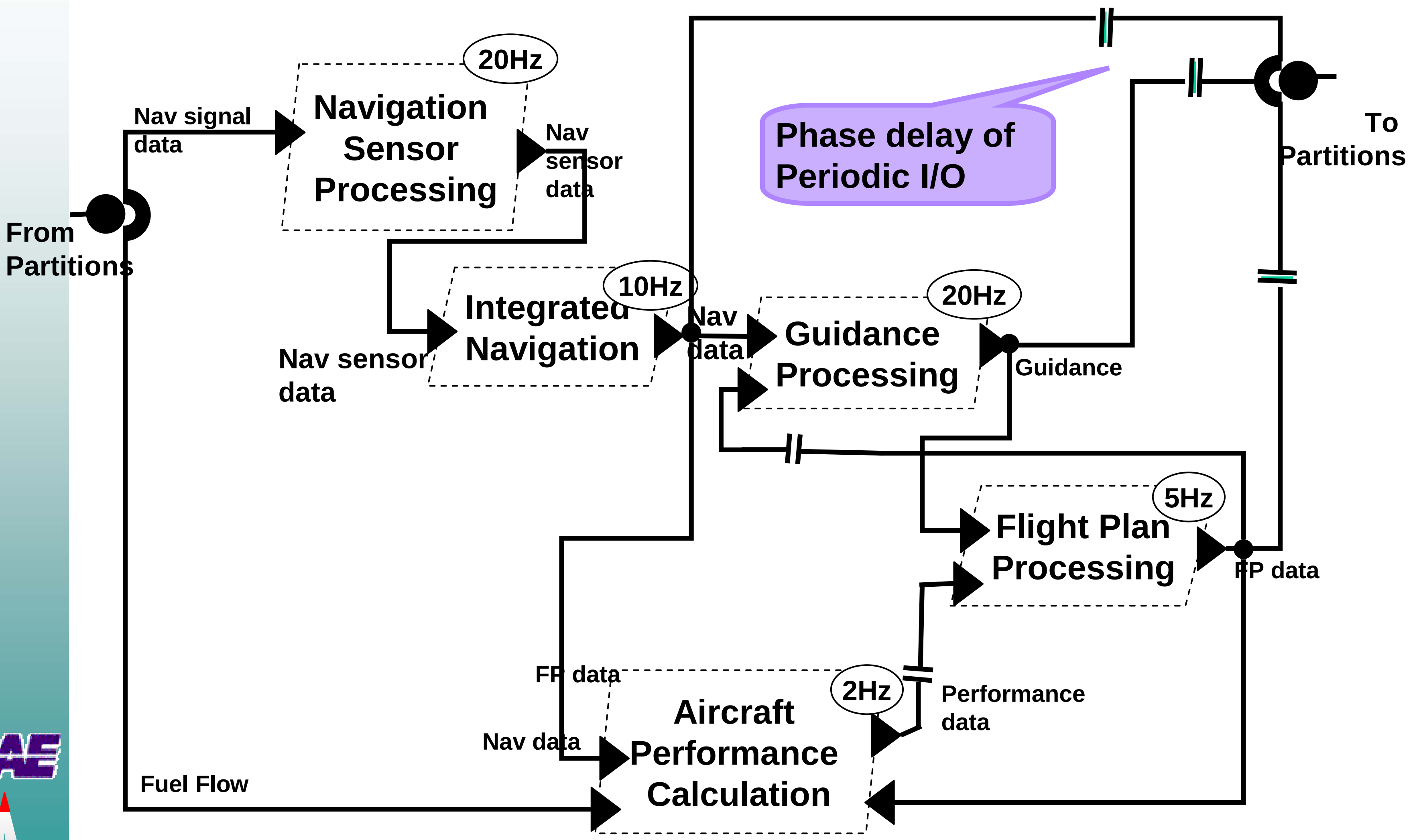

Connection (delayed)

- For delayed data connections
  - data transmission is initiated at the deadline of the connection source thread
  - data is available at the destination port at the next dispatch of the destination thread
- Value:
  - controlled communication delays
  - allow feed-back loops
  - break connection cycle
  - recipient samples at its dispatch rate and time

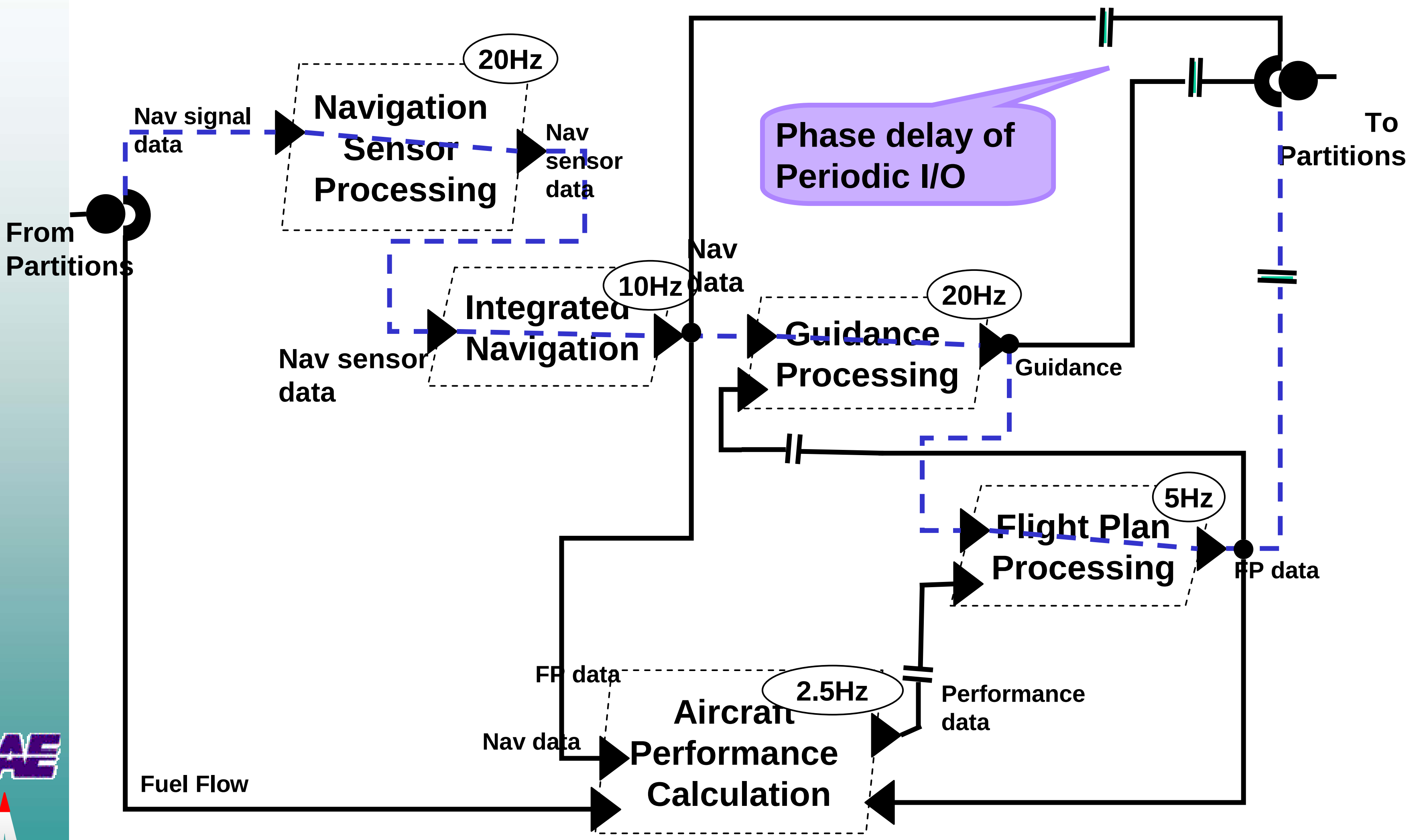
Delayed Communication



Flight Manager in AADL



End-To-End Flow Specification



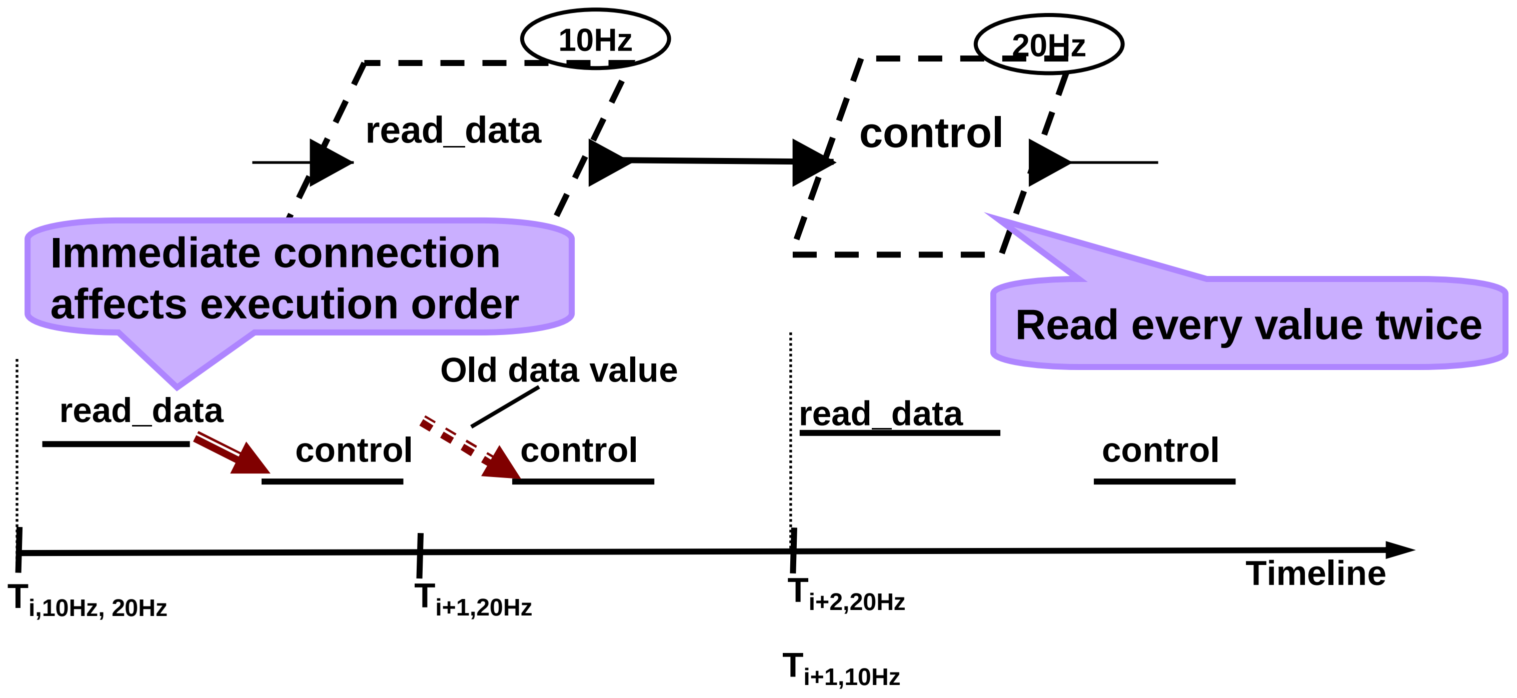
Observations on Use of AADL

- Explicit interface specification for all threads.
- Named and typed ports with flow constraints supports flow consistency checking.
- Focus on data flow, not task priority assignment:
  - Flow expressed through data ports & connections,
  - No shared data object used.
- Explicit specification of communication timing through delayed connection.
- Specification of thread dispatch characteristics.
- Understandability of AADL models.
- Multiple viewpoints available using AADL.

Deterministic Communication in AADL

- AADL semantics assure communication determinism
  - In terms of dispatch & deadline.
- Implementation considerations
  - Mutual exclusive port variable access,
  - Double buffering as appropriate,
  - AADL runtime system responsible for dispatch & communication.

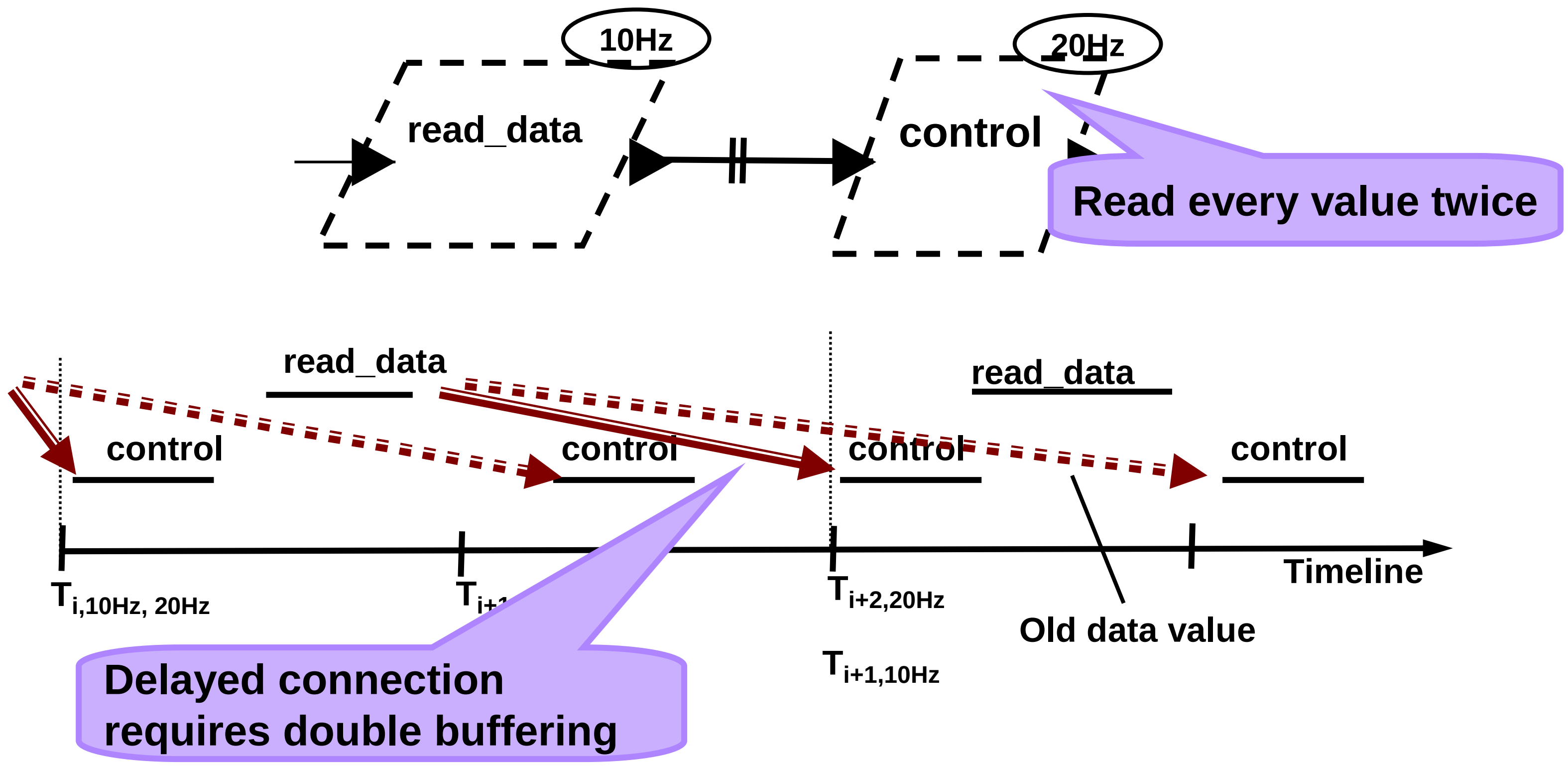
Sampling & Immediate Connections



AADL assures

- Deterministic sampling
- Single buffer solution

Sampling & Delayed Connections

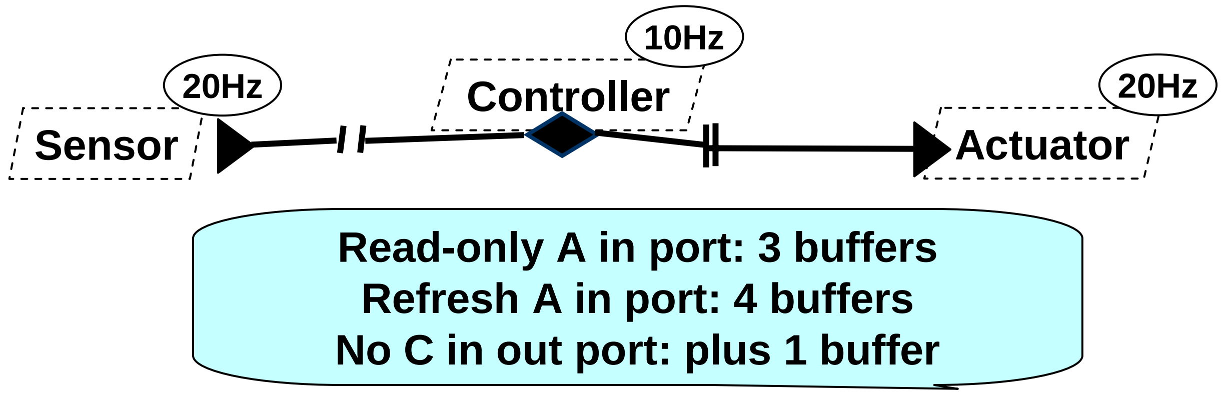


Delayed vs. immediate

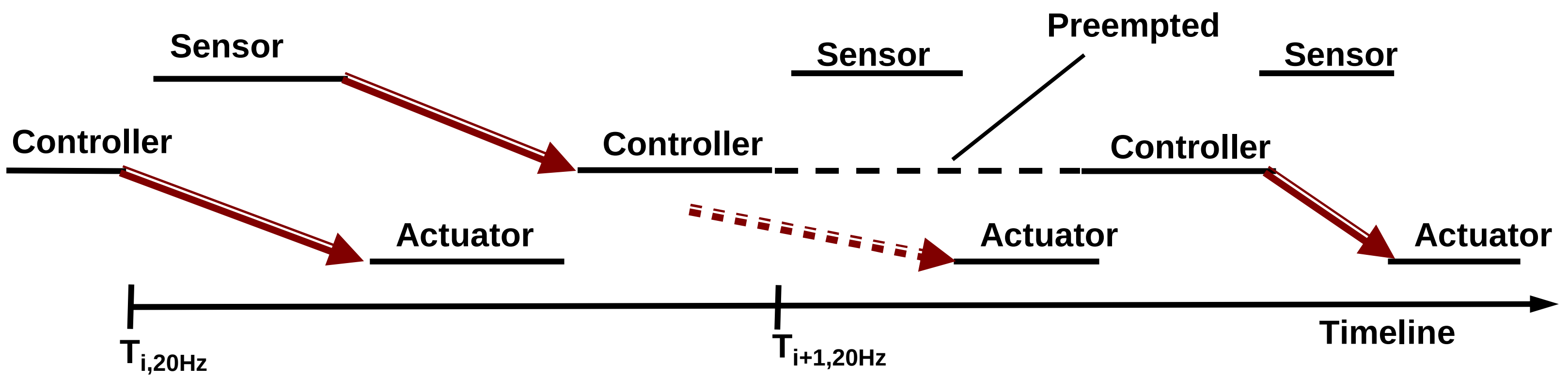
- Both are deterministic
- They differ in amount of latency

Up and Down Sampling

- S-C: overlapping lifespan – xfer at Controller rate
- C-A: overlapping lifespan – xfer at Actuator rate



10Hz
20Hz
Controller
Sensor
20Hz
Actuator
Read-only A in port: 3 buffers
Refresh A in port: 4 buffers
No C in out port: plus 1 buffer



Connection Sequences

- What is the effect of combining immediate & delayed connections?

@ separates 20 Hz timeframes

Implied C deadline

- S_{20Hz} -> C_{10Hz} -> A_{20Hz}

  S;C;A @ S;A: C_{D20Hz} 0 pd

- S_{20Hz} ->> C_{10Hz} -> A_{20Hz}

  S|A @ S|(C;A): C_{D20Hz} 1 pd

- S_{20Hz} -> C_{10Hz} ->> A_{20Hz}

  (S;C) | A @ S|A : C_{D10Hz} 2 pd

- S_{20Hz} ->> C_{10Hz} ->> A_{20Hz}

  (S|C|A) @ (S|C|A):C_{D10Hz} 3 pd

| indicates concurrency for multi processing

Amount of phase delay

Application Design Tradeoff

- When do I need immediate data transfer?
- When do I need phase delayed transfer?
- When can I afford phase delayed transfer?

Reduction in resource constraints results in higher resource utilization

Observations: AADL Communication

- Focus on what communication is desired, not how it is implemented.
- Assures deterministic communication when desired
- Shows application rates & desired phase delay explicitly.
- Bridging the time gap:
  - Basis for tradeoff between application and system engineer;
  - Sensitivity analysis on variation in sampling rates & dispatch rates;
  - Connection timing affects latency;
  - Connection timing choice affects resource utilization & controller stability.

Preemptive Scheduling & Data Flow

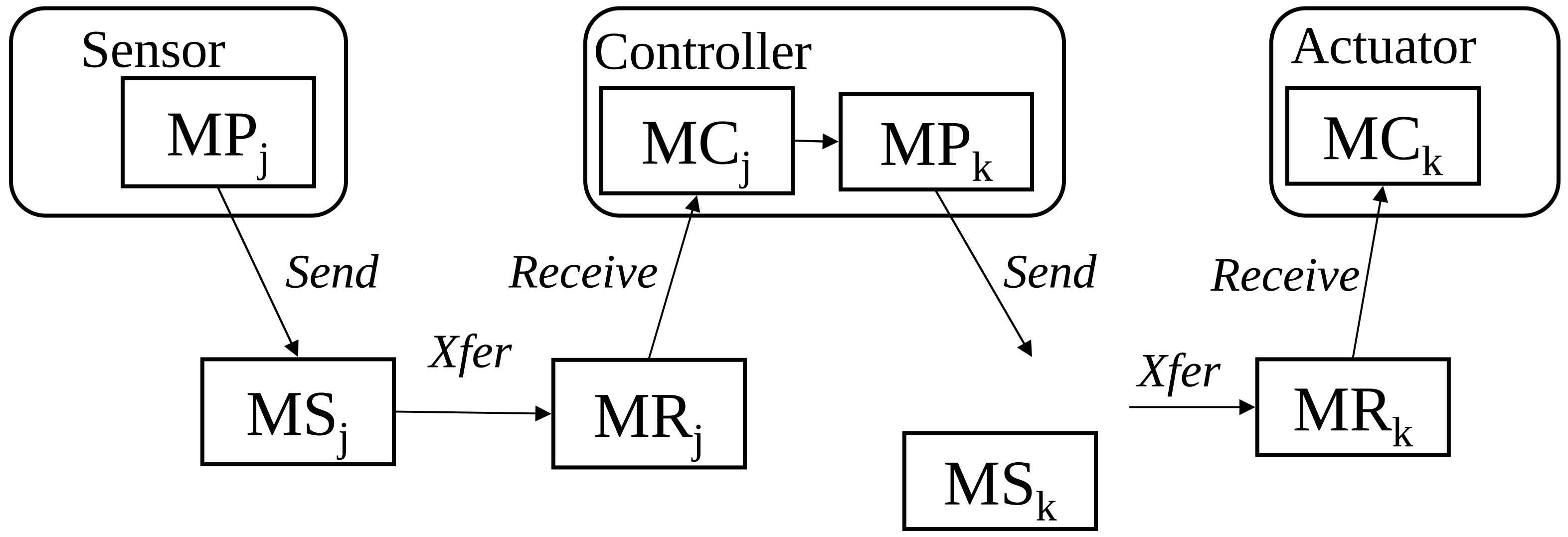
- Preemptive scheduling & shared variables
- Data flow patterns
- Threads and port connections in AADL
- Efficient and correct implementations

Modernizing an embedded real-time system
From cyclic executive to preemptive scheduling
From shared variables to port communication

Efficient Communication

- Should not be application developer responsibility
- Optimization problem for software system engineer
- Built into a runtime generation tool

Port Communication & Buffers



Buffer reduction techniques

- Send/receive with or without copy or no send/receive
- Transfer with or without copy or no transfer
- Processing with or without copy

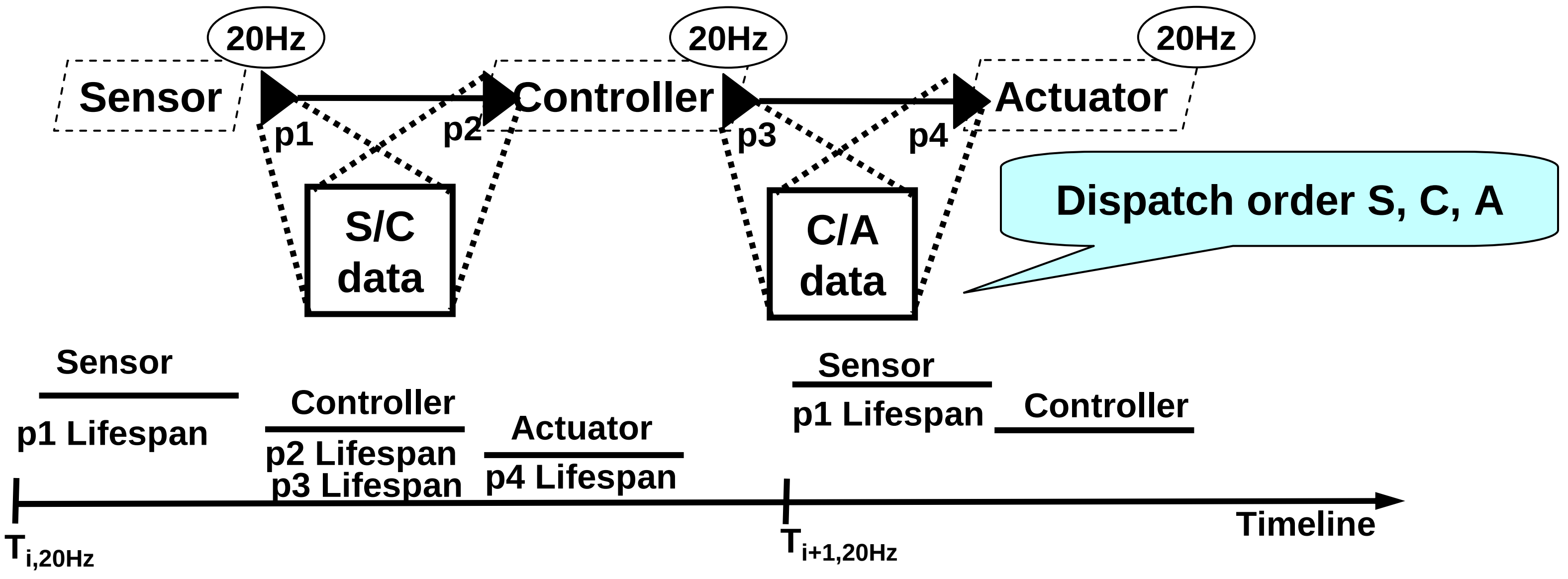
copy
MR: receive copy
MC: consumer copy

ARINC 653, OSEK & other standards intend to support such optimizations

Efficient Communication Implementation

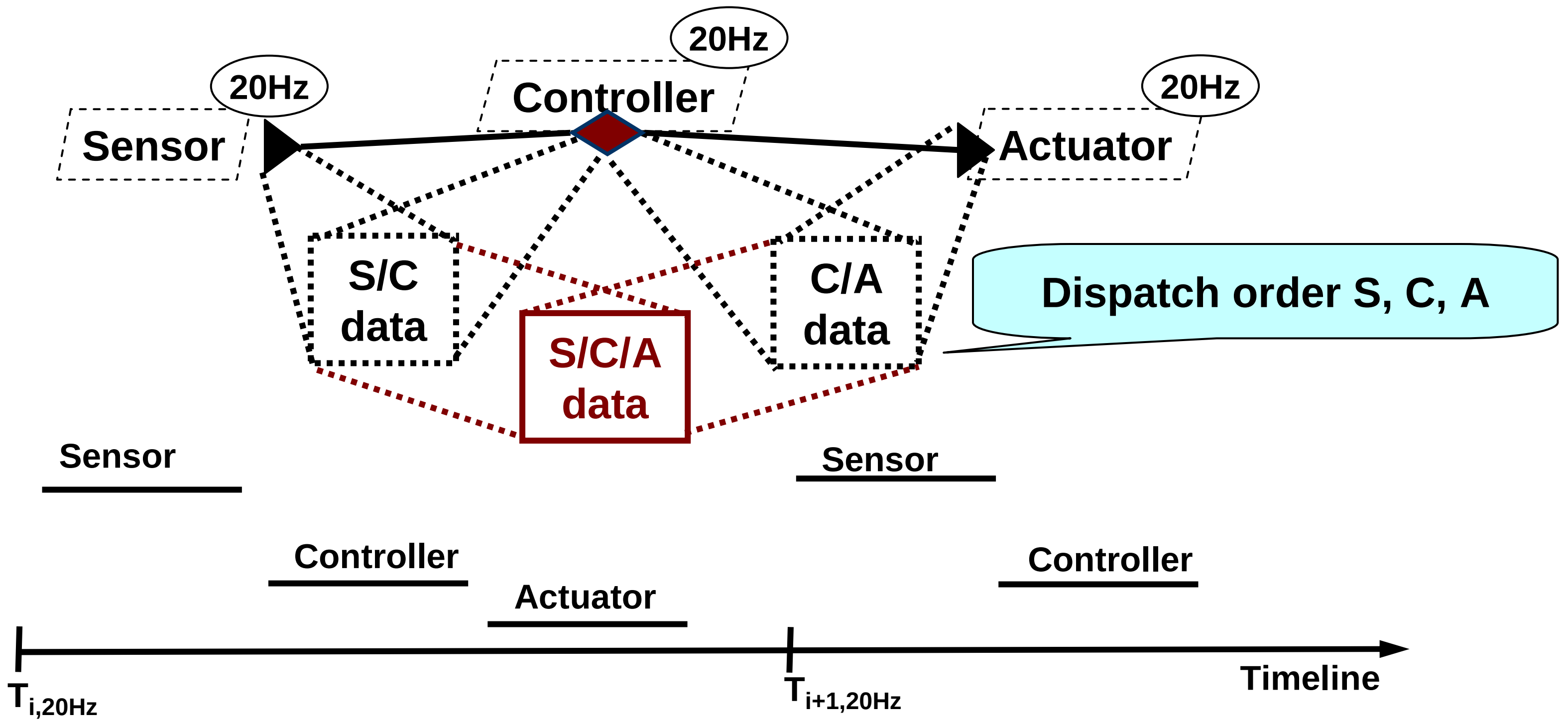
- No preemption within priority level is assumed
- Dispatch order determines precedence order within priority level
- Consider port & buffer variable lifespan to determine sharing of buffer variables

Similar to register allocation problem for compilers



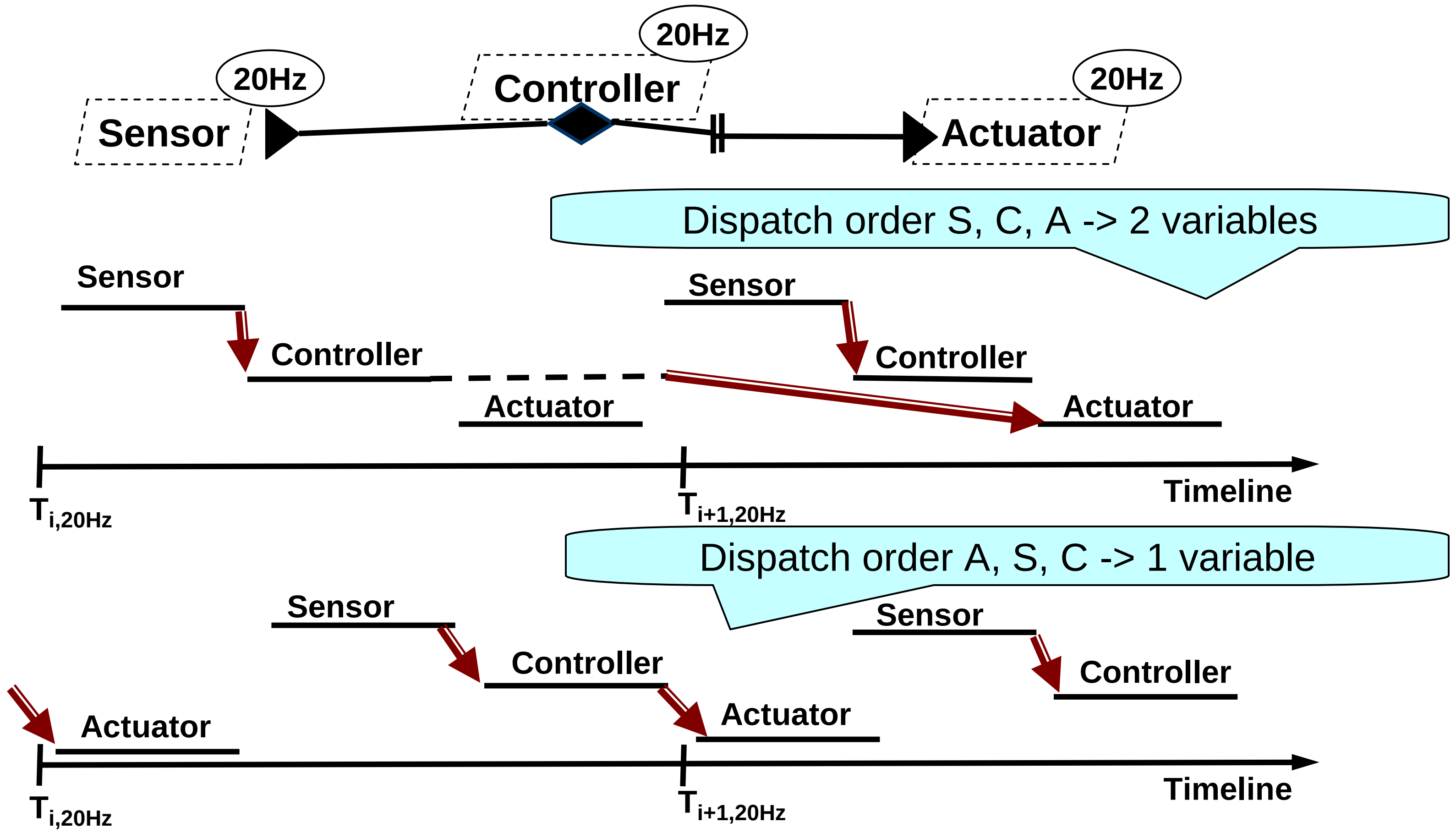
Going All The Way

- Specify common in & out port within component using **in out** port in AADL
- Same data type for both ports
- Thread B updates port data



Need For Double Buffering

- Delayed actuator connection: controller time extrapolation
- Dispatch order affects # of buffer variables



Periodic Task Communication Summary

Periodic Same period	ASR IMT | PMT		DSR		CSR	
τ_C ; τ_P	PD/1B S/X_{NC} R_{NC}	PD/1B S_{NC}, X_{NC} R_{NC}	S/X_{NC} R_{NC}	S_{NC} X/R_{NC}	$S/X/R_{NC}$	NA
τ_P ; τ_C	MF/1B S/X_{NC} R_{NC}	PD/2B $(S \vee X \vee R)_C$	PD/2B S/X_{NC} R_C	PD/2B $(S \vee X/R)_C$	MF/1B $S/X/R_{NC}$	NA
$\tau_P \neq \tau_C$	ND/1B S/X_{NC} R_{NC}	PD/2B S_{NC}, X_C R_{NC}	PD/2B S/X_{NC} R_C	PD/2B S_{NC} X/R_C	ND/1B $S/X/R_{NC}$	NA
$\tau_P \mid \tau_C$	ND/3B S/X_C R_C	PD/2B S_{NC}, X_C R_{NC}	PD/2B S/X_{NC} R_C	PD/2B S_{NC} X/R_C	DR/2B $S/X/R_C$	NA

Application-level send & receive compounds problem

MF: Mid-Frame
PD: Period Delay
ND: Non-Deterministic
DR: Destructive Receive
NA: Not Applicable

1B: Single buffer
2B: Two buffers
3B: Three buffers
4B: Four buffers

Op_C : S, X, R data copy
Op_{NC} : S, X, R no data copy
S/X : IMT combined send/xfer
S/X/R : CSR combined S, X, R
$(o1 \vee o2)_C$: One operation copy

Harmonic Task Communication Summary

Harmonic Slow - Fast	ASR				DSR			DMT
	IMT |	PMT_P |	PMT_C		IMT |	MT_P	| PMT_C	
τ_C ; τ_P ; τ_C	PD_C 1B	PD_P 2B: X	PD_{DC} 2B: X	ND_{DP} 3B: S X	PD_D 1B	PD_P 2B:X	PD_D 3B: S X/R	PD_C 1B
τ_P ; τ_C ;; τ_C	MF 1B	PD_P 2B: X	PD_{DC} 2B: X	ND_{DP} 3B: S X	PD_C 1B	PD_P 2B:X	PD_D 3B: S X/R	MF 1B
$(\tau_P \neq \tau_C)$; τ_C τ_C ; $(\tau_P \neq \tau_C)$	ND 1B	PD_P 2B: X	PD_{DC} 2B: X	ND_{DP} 3B: S X	PD_D 1B	PD_P 2B:X	PD_D 3B: S X/R	ND 1B
$(\tau_P \mid \tau_C)$; τ_C τ_C ; $(\tau_P \mid \tau_C)$ $\tau_P \mid \tau_C \mid \tau_C$	ND 3B: S/X R	PD_P 2B: X	ND 3B: S X		PD_D 3B: S R	PD_P 2B:X	PD_D 3B: S X/R	NDI 2B: S/X/R

PD_C : Consumer p

PD_P : Producer

PD_D : $D_P \leq T_C$ th

This can be analyzed!!!

Should be responsibility of AADL analyzer & executive generator

No AADL Generator?

- AADL supports application flow specification & preemptive scheduling.
- Complexity left to AADL tools.

- What if no AADL tool?
  - Plug-in as application component development style.
  - Application code unaffected by implementation decisions.
  - Runtime system implementation patterns used by system implementer.

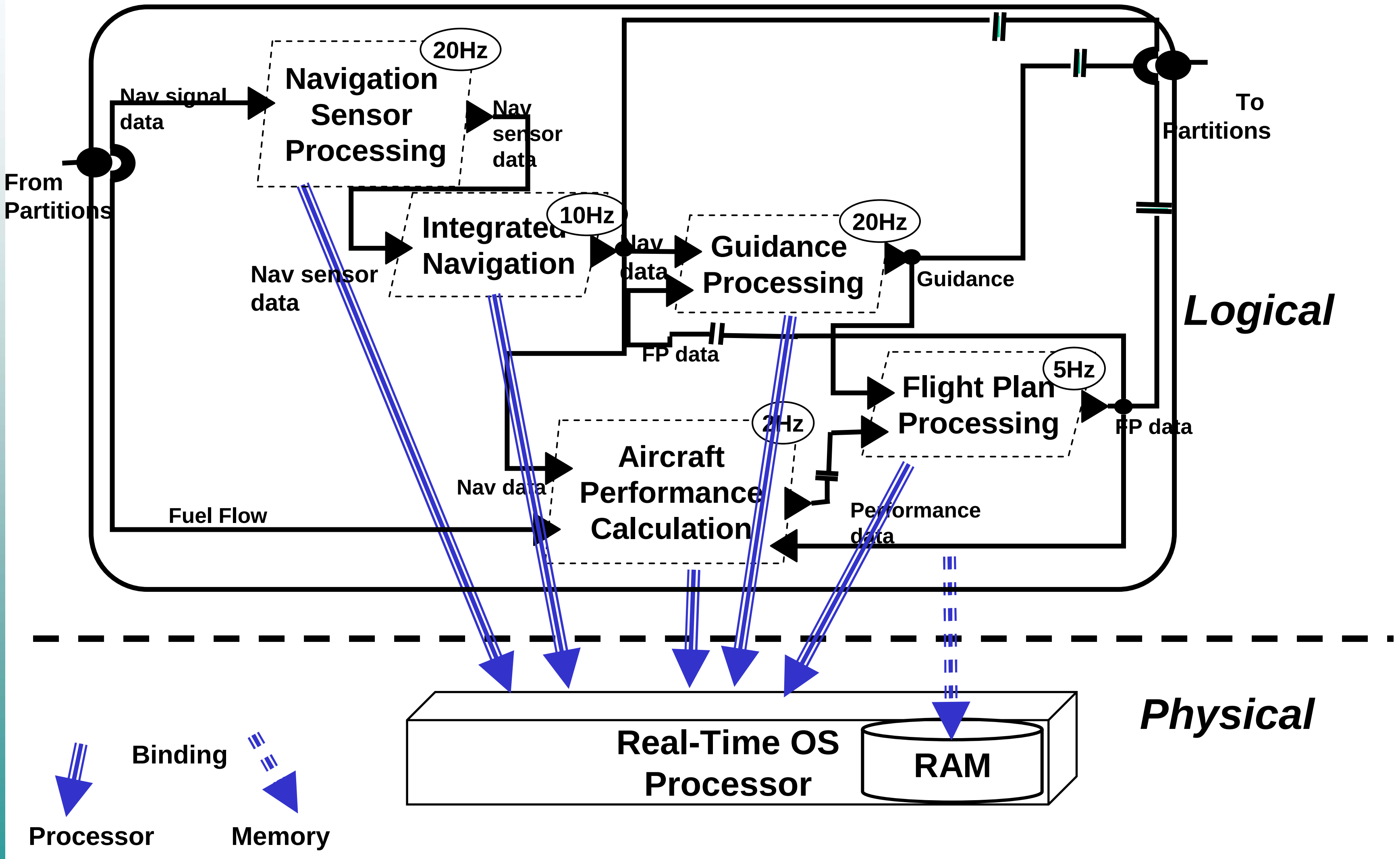
Some Observations

- Legacy Option
  - Static timeline scheduling.
  - Deterministic execution & communication.
  - Underutilized resources & brittle timing architecture.
- Modernization Option
  - Predictable preemptive fixed-priority scheduling.
  - Better resource utilization & flexible timing architecture.
  - Explicit data flow modeling & communication timing.
  - Buffer management is an optimization problem.

Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering

Execution View



Scheduling Analysis

- Scheduling protocol determines analysis
  - Processor budget for static time line (cyclic executive)
  - Rate-monotonic Analysis (RMA) for preemptively scheduled fixed-priority tasks
  - 100% utilization for Earliest Deadline First (EDF)
- What if analysis of
  - Schedulability under different dispatch & scheduling schemes
  - Miss-estimated worst case execution times (WCET)

Commercial real-time system analysis tools provide such support

WCET & Sensitivity Analysis

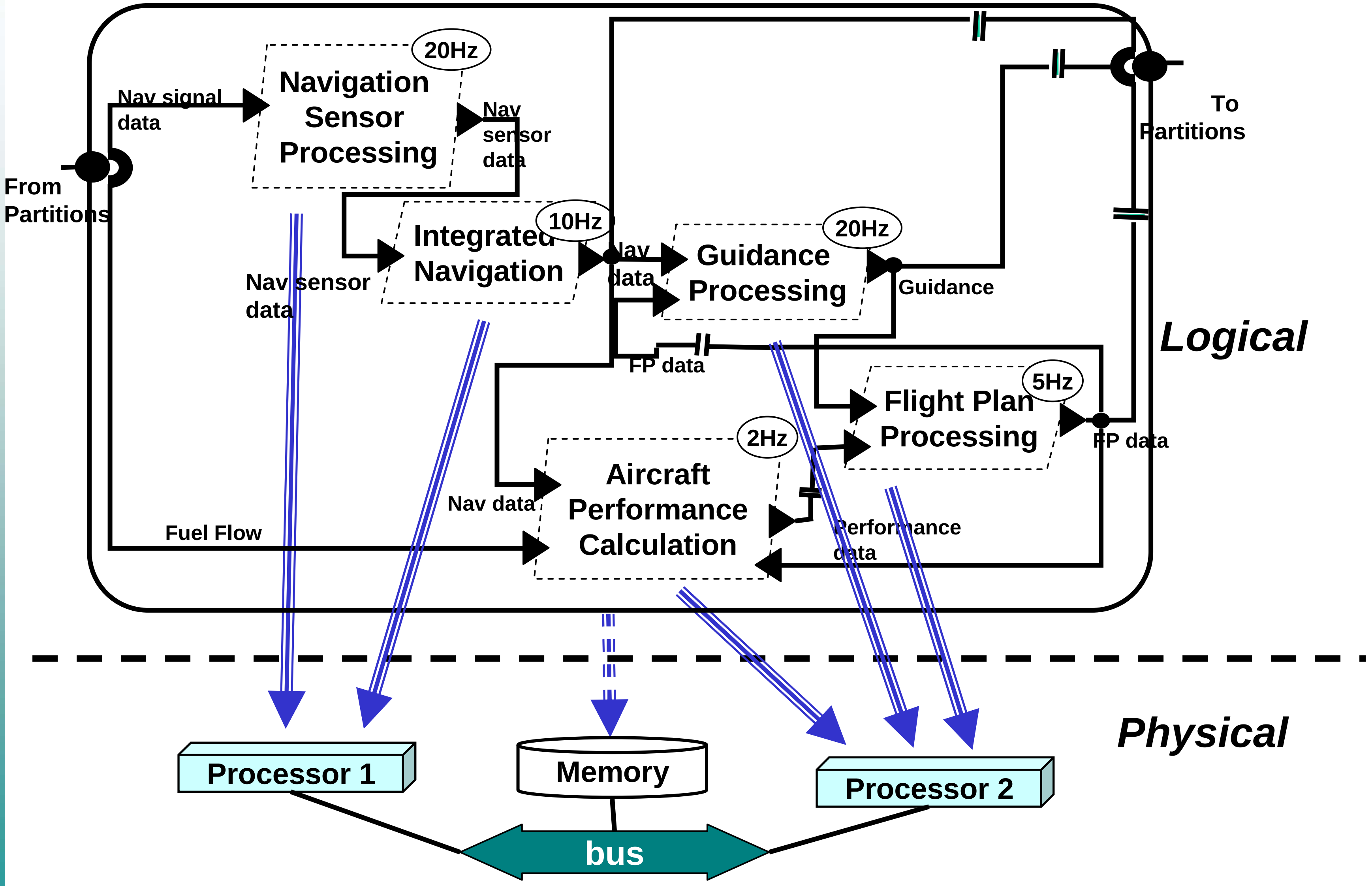
```
    Processor timing, application Periodic_IO, mode Periodic_IO, processor Ada95

Module                  Period    Budget  Critical   Maximum  % Margin  % Util   Max
    Pri

Periodic_IO_system      10000       100       588      8399      98.8     1.0

Fast                    10000      1050      6174      9349      88.8    10.5
  <self>                           1000      5880      9299      89.2    10.0   Fast
  <system>                           50       294      4425      98.9     0.5   Fast

Slow                    20000      1051      6178     17649      94.0     5.3
  <self>                           1000      5880     17599      94.3     5.0   Slow
  <system>                           51       298      8350      99.4     0.3   Slow

Total utilization =  16.8%
Breakdown utilization =  98.5%
Critical scaling factor =  5.88
Processor schedule is feasible

Times are reported in microseconds
Nominal compute times were used in the compute paths.
```

How much all tasks can increase their WCET

Real-time Performance

- If a single processor system is not schedulable
- Might consider
  - rewrite code to make it faster
  - Consider lower signal processing rate for controller (e.g. 40Hz)
  - Use faster processor
  - Add second processor
- Explore these options using AADL and support tools
  - Reduce worst-case execution time
  - Identify schedulable rate from sensitivity analysis results
  - Execution time properties specific to processor speed
  - Reconfigure to different execution platform

Thread Distribution



Observations: AADL and Performance

- Enables exploration of design alternatives without rewriting code
- Makes explicit critical design considerations
- Facilitates early decisions on real-time design issues
- Redistribution of components
  - Potentially even threads
  - Completely specified interfaces for threads

More on performance later.

Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering

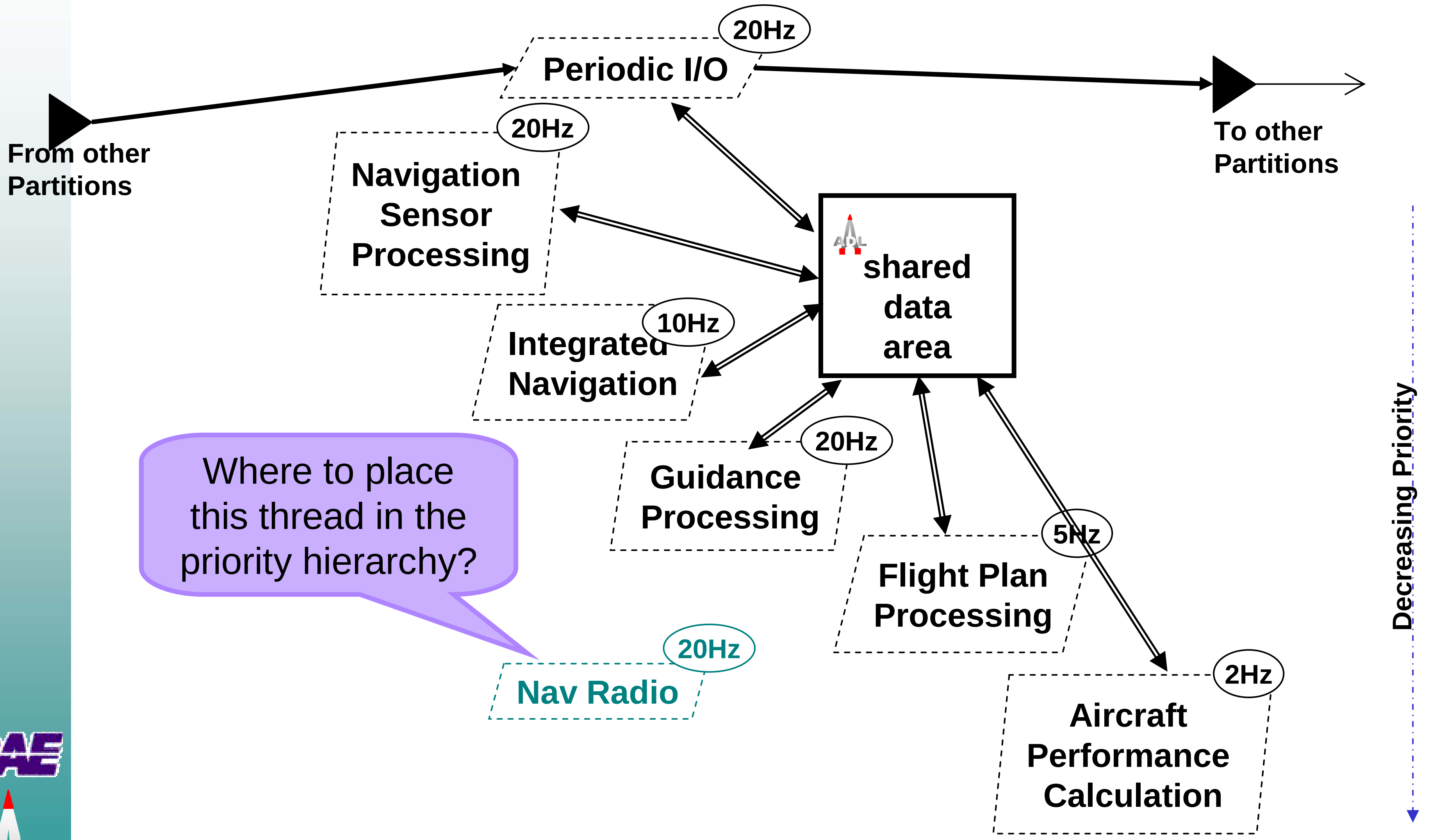
Event Handling By Polling

- Establishes deterministic system execution patterns
  - Fits easily into a cyclic executive.
  - Helps avoid saturation (e.g. alarm overload).
- Minimizes response times
  - Establishes a well-defined reaction latency.
  - Possible to readily minimize reaction latency.
  - Polling rate driven by minimum state change interval.
- Supports hardware environments that are not interrupt driven.

Avionics System Polling Examples

- 10 Hz keypad polling
  - 10 Hz execution rate reserved
  - Interaction transactions
  - Keypad input -> page content update -> keypad input
  - Multiple partitions reserve resources
  - Realistic event interarrival time
  - Keypad interrupt to get system attention
- 20 Hz Navigation radio channel setting
  - Always-on periodic polling
  - Placed with flight manager partition
  - Occasional pilot activity
  - Reserved resource

Nav Radio In Flight Manager



Where to place this thread in the priority hierarchy?

Polling and Scheduling Analysis

- Consider a polling system with a 5x over sampling of a10 Hz stream (scheduling rate of 50 Hz)
  - High rate is to reduce reaction latency
  - Reserve processing time at 50Hz: max. 20% resource utilization
  - Assume processing at 10Hz: impact of variation in data arrival
- Many devices (e.g., switches) are polled
  - Resource reservation for all
  - Assume a subset of switches are can be manipulated at any one time
- Towards an event-driven system
  - process only when new data arrives
  - bounded minimum interarrival time
  - results in aperiodic/sporadic thread sequence

Event-Based Processing

- Recognize bounded event arrival rates
- Manage event processing resource needs
- Leverage operational modes

Bounds on Interaction Rate

- Examine end-to-end flow of interaction
  - Utilize AADL flow specifications
  - Avionics example: Nav Radio

    DM -> PCM/domain -> FM -> 1553 IO -> 1553 device ->

    1553 IO -> FM -> PCM -> DM
  - Each inter-partition step contributes phase delay
  - Interaction as single-threaded synchronous call/return
  - Realistic inter-arrival rate much lower

Events with minimum interarrival time
Bounded by physical world
Bounded by application logic

Manage Event Processing Resources

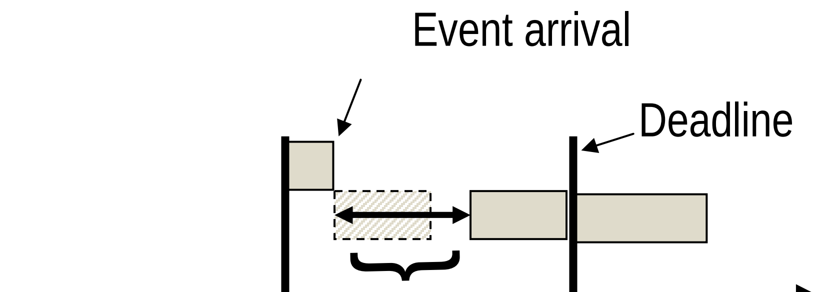
- Sporadic Threads
  - Event dispatching of threads with hard deadlines and minimum dispatch separation
  - Spread the processing load over time
- Sporadic Server
  - Integrated into scheduler
  - Manage response time through queuing discipline
  - Application level implementations exist
- Event sampling
  - Periodic sampling of event queues
  - Bundled event processing
  - Example: cascading alarm processing

Approach used by
- Process control systems
- Time-Triggered Architecture (TTA)

All three techniques are supported by AADL

Slack Scheduling Approach

- Guarantees deadline of scheduled threads.
- Static slack
  - Unscheduled time
- Dynamic slack
  - Unused scheduled time



Slack(t) is maximum time that scheduled tasks can be delayed at time t without missing their deadlines.

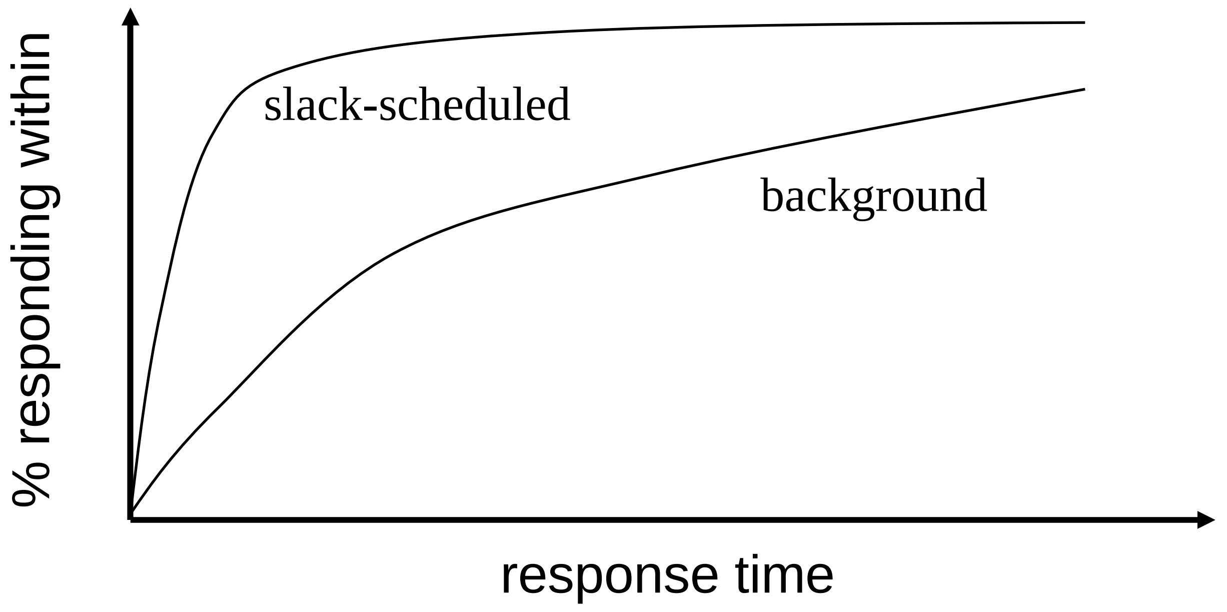
Slack Scheduling Performance

Partitioned aperiodic, incremental, and dynamic threads

- DEOS (Primus Epic RTOS, replaced deferred server)

COTS FTP/TCP/IP stack hosted in DEOS

- > 3X improvement in throughput
- 7X reduction in reserved processor utilization (70% down to 10%)



Leverage Operational Modes

- Limit on number of sporadic streams.
- Limit due to physical environment:
  - Pilot only has two hands.
- Operational mode limits pilot activity:
  - Landing gear switch disabled during dogfight.

Mode-Based Solutions

- Improvements for polled systems.
- Consider optional services
  - Active only in certain operational modes.
  - Smaller set of polling threads.
- Consider reachable mode combinations
  - Reduction in worst-case execution time requirements.

Mode Sensitive WCET

- Worst-case mode combinations
  - Example: Terrain Following (TF) & Reconnaissance (Recon)
  - Recon Hi & TF Hi operationally or logically infeasible

Worst-Case Execution Time	TF Hi	TF Lo
Recon Hi	~~50ms+50ms~~	50ms+10ms
Recon Lo	20ms+50ms	20ms+10ms

Observations on Polling and Event Driven Systems

- AADL supports:
  - Event driven system view.
  - Co-existence with periodic processing.
  - Realistic event rates for given flows.
  - Techniques for managing resource demands.
  - Real-time performance analysis sensitive to operational modes.
  - Ability to compare poling, event, or periodic approaches.

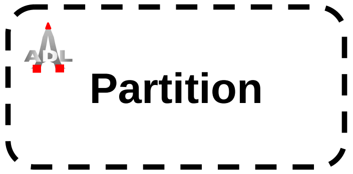
Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering

The Partition Concept

- Found in ARINC 653
- Runtime protected address space
- A virtual processor scheduled on a static timeline
- Contained threads (ARINC processes) are scheduled within the bounds of a scheduled partition
- Different partitions can use different thread scheduling protocols
- Communication of queued and unqueued data
- Inter vs. intra partition communication

Partitioning in AADL

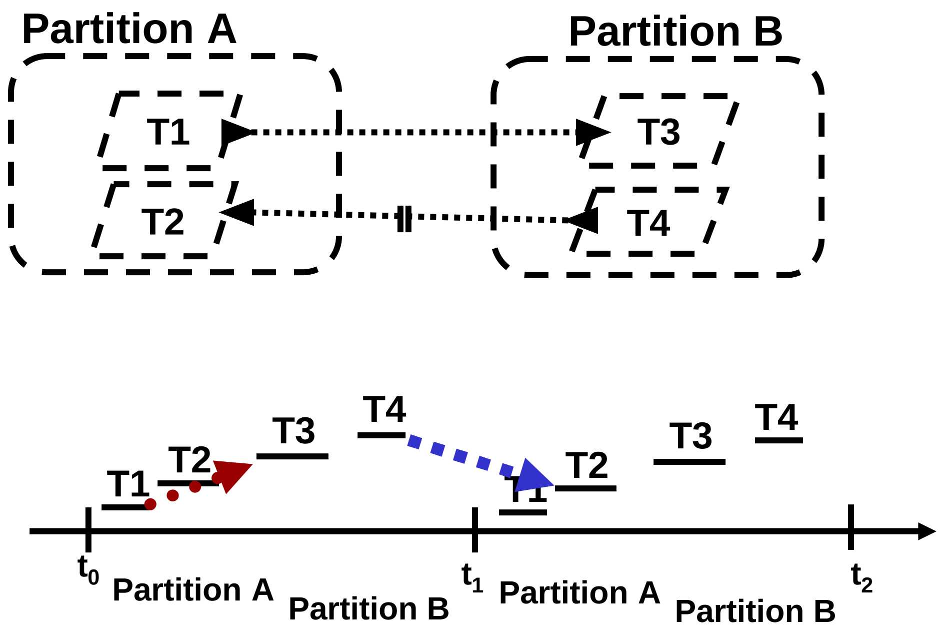


- Core AADL has
  - Systems as logical units of composition.
  - Threads as schedulable units of execution.
  - Processes as protected address spaces.

- Extend process concept of AADL
  - New partition scheduling protocol.
  - New scheduler properties for process.
  - Clarification of communication timing.

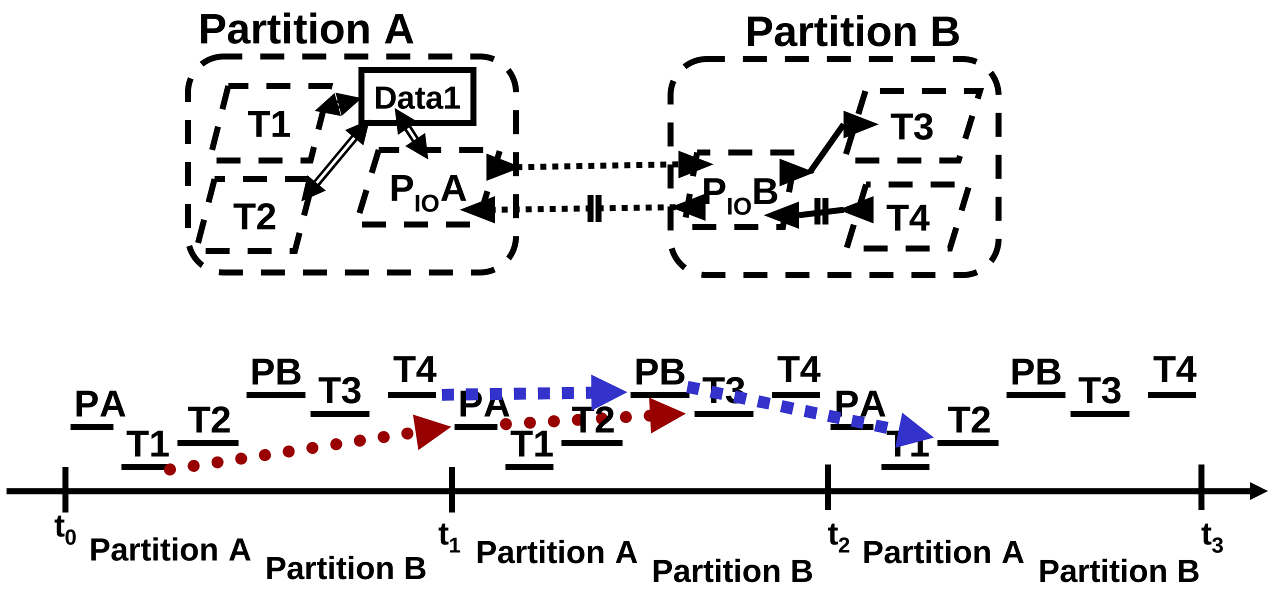
Partition Order Side Effects

Partition communication via send/receive



Partition Order Side Effects: Periodic I/O

Periodic I/O adds phase delay
Periodic I/O is sensitive to partition order



Why Periodic I/O Does Not Work

- Partitions on single processor
  - Periodic I/O has partition dispatch as time reference.
  - Statically known communication timing characteristics.
  - Partition order affects communication timing.

- Partitions distributed on multiple processors
  - Concurrent partition execution.
  - Non-deterministic communication timing.

Partitions execute simultaneously

Network Loopback Approach

- Partition communication within processor via network.
- Latency unaffected by partition placement.

No latency reduction through partition relocation

- Network load unaffected by partition placement

No network load reduction through partition relocation

Partition Communication in AADL

- Communication via ports and connections.
- Common reference time for data transfer across partitions.

AADL Annex on partitions must address this.

- AADL runtime system manages
  - Partition execution.
  - Inter-partition communication with respect to processor time.
  - Assumes synchronous system.

Periodic system bus plays such a role in Time-Triggered Architecture (TTA)

Partitioned System Design in AADL

- Focus on partition order isolation
  - Delayed connections insensitive to partition order.
  - Delayed connections insensitive to partition concurrency for synchronous systems.
  - Delayed connections contribute to latency.
- Focus on latency
  - Immediate connections reduce latency.
  - Immediate connections constrain partition order.
  - Immediate connection cycles

    Detectable by analysis

    - Direct cycle: $P_A.T1 \rightarrow P_B.T2 \rightarrow P_A.T3$
    - Pair-wise cyclic: $P_A.T1 \rightarrow P_B.T2$ & $P_B.T4 \rightarrow P_A.T3$
- Focus on flexibility
  - Acceptable variation in phase delay.

    Document as property

Asynchronous Systems

- Periodic sampling by independent clocks.
- Potential clock drift
- Sliding [0-period] sampling delay variance for inter-processor communication.
- Partition placement results in latency variation.
- Latency and latency variation can be reduced by running *x* times faster (over-sampling).

Aperiodic thread sequences are independent of clock drift.

Analysis & Design of an Avionics System

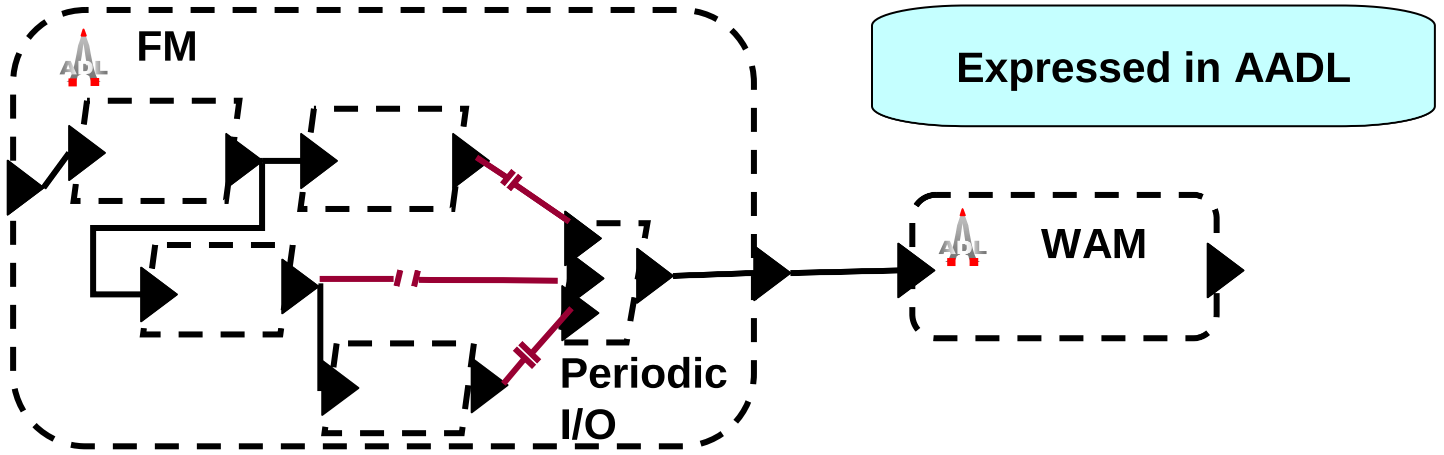
- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering

Acceptable AADL Connection Patterns

- Connection sequences
  - Pipeline, flow
- Connection tree
  - Branching flow
  - Different endpoint latencies
- Directed acyclic graph (DAG)
  - Flow with merge points
  - Phase delay difference of branches at merge point
  - Effects of phase delay oscillation in non-deterministic case
- Cyclic connections
  - Feedback control, action/observation
  - Phase delay breaks cycle

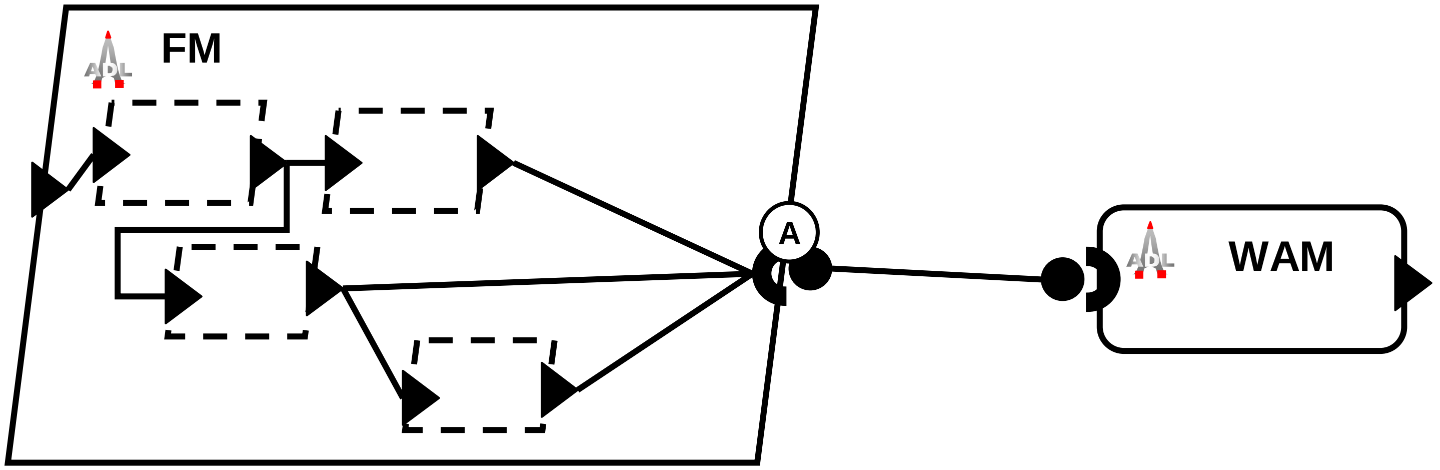
Analyzable in AADL

How Does Periodic IO Stack Up?



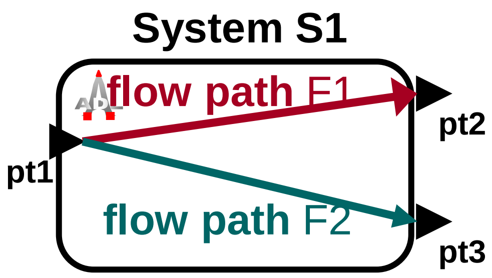
- Periodic IO as application thread
  - Phase delay within source partition.
  - Unnecessary merge-point phase delay.

Time Consistent Data Sets



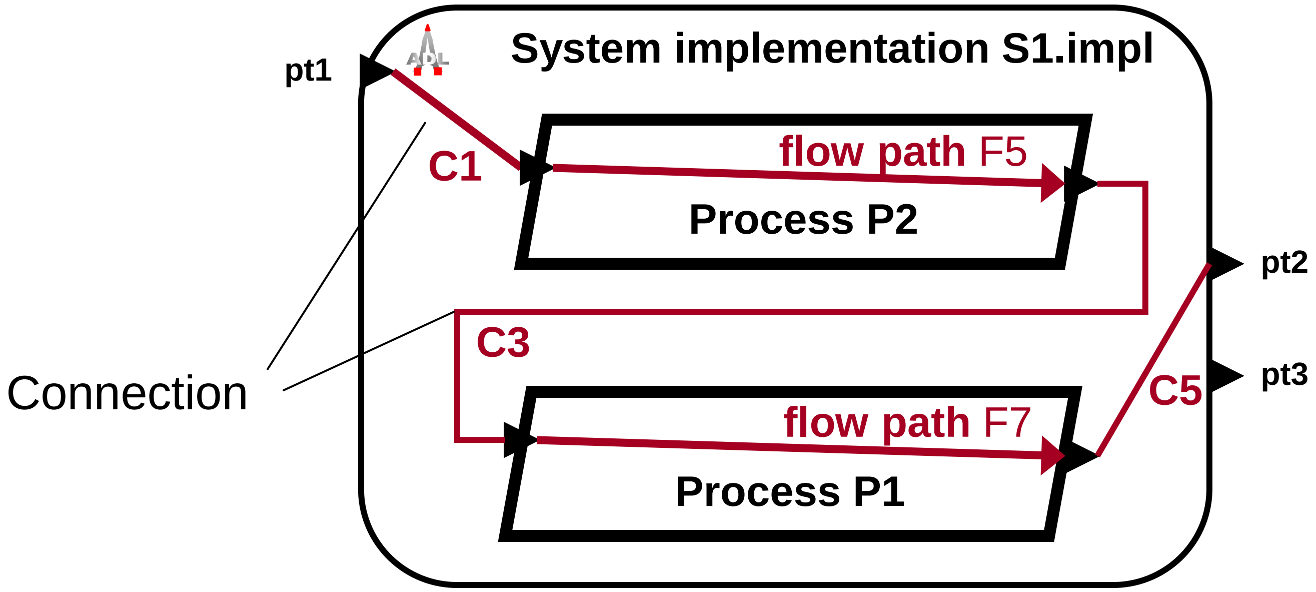
- Aggregated data transfer
  - Use of port group concept.
  - Property to indicate aggregation semantics.
  - Availability of last data value triggers transfer as a single unit.

Flow Specification in AADL



Flow Specification

flow path F1: pt1 -> pt2
flow path F2: pt1 -> pt3



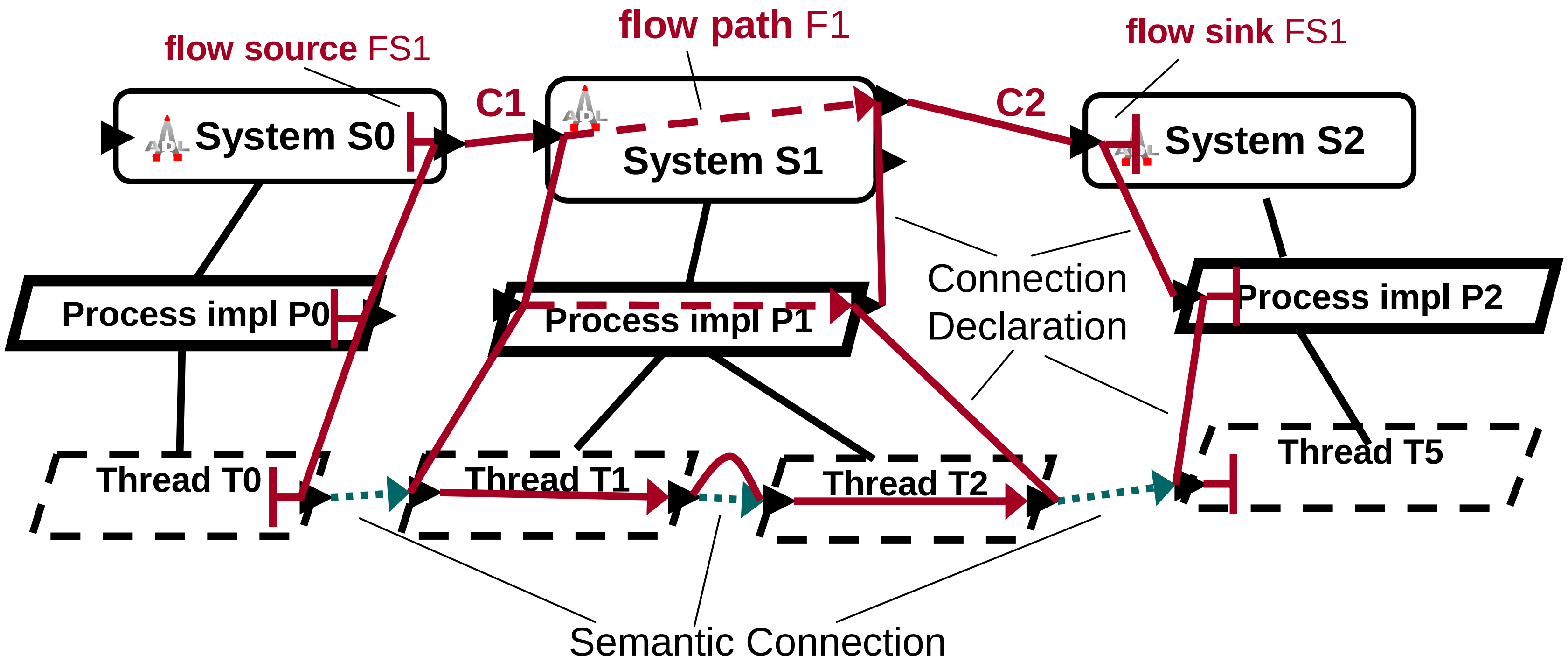
Flow Implementation

flow path F1: pt1 -> C1 -> P2.F5 -> C3 -> P1.F7 -> C5 -> pt2

End-To-End Flow Semantics

End-To-End Flow Declaration

flow SenseControlActuate: S0.FS1 -> C1 -> S1.F1 -> C2 -> S2.FS1



End-To-End Flow Semantics of SenseControlActuate

T0.FS1 -> <SemCon1> -> S1.P1.T1.FX1 -> <SemCon3> -> S1.P1.T2.FX2 -> <SemCon3> -> S2.P2.T5.FS1

Data Stream Latency Analysis

- Flow specifications in AADL
  - Properties on flows: expected & actual end-to-end latency
  - Properties on ports: expected incoming & end latency
- End-to-end latency contributors
  - Delayed connections result in sampling latency
  - Immediate periodic & aperiodic sequences result in cumulative execution time latency
- Phase delay shift & oscillation
  - Noticeable at flow merge points
  - Variation interpreted as noisy signal to controller
  - Analyzable in AADL

Potential hazard

Latency calculation & jitter accumulation

Reduce End-To-End Latency

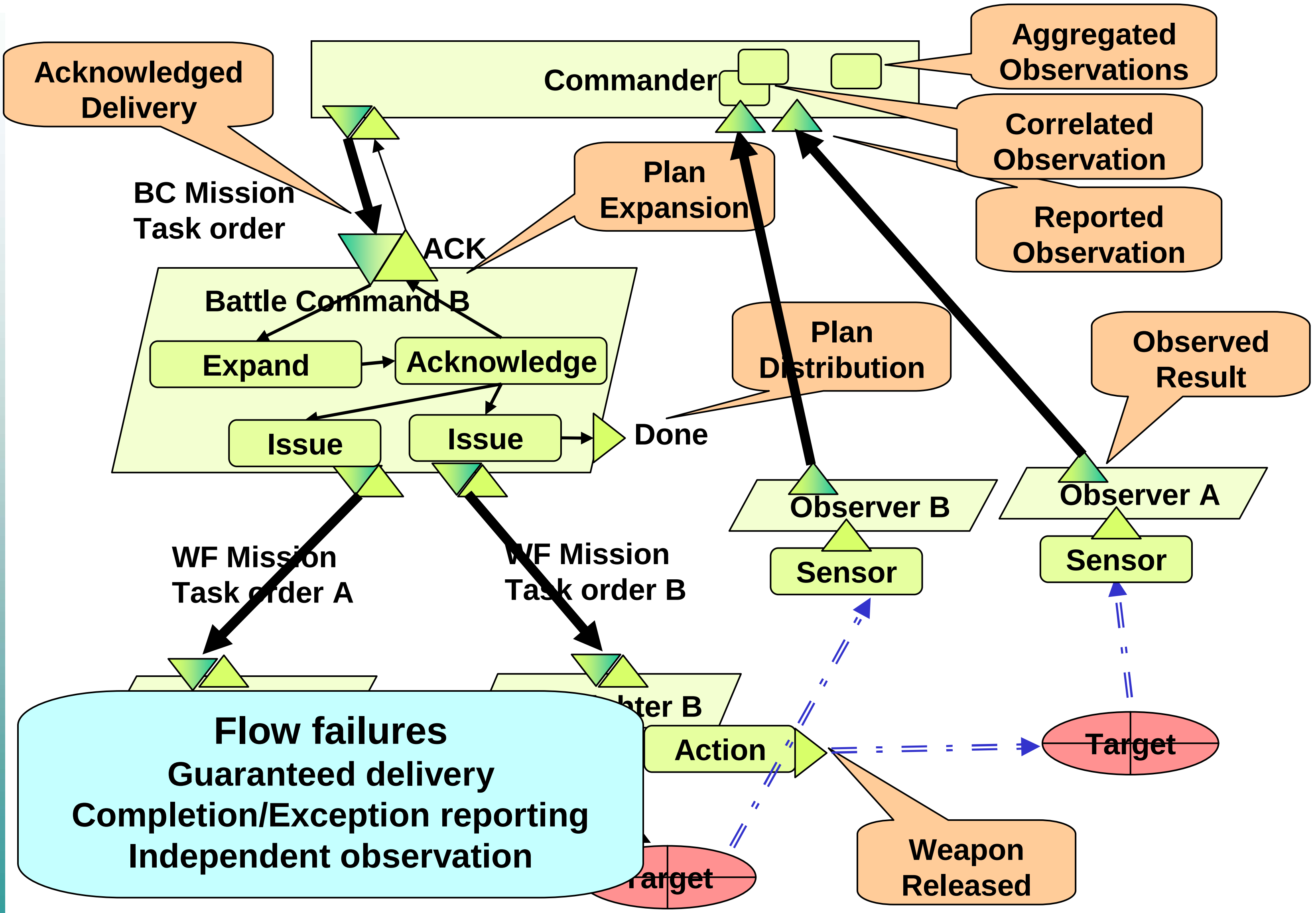
Consider engineering alternatives

- Reduce execution time
  - Limited impact in periodic sampling systems.
- Eliminate unnecessary sampling steps
  - Change to immediate connections for critical flows.
  - Results in partition order constraint.
- Merge partitions
  - Eliminates partition order constraint.
- Aperiodic sequencing of periodic streams
  - Eliminates sampling phase delay.
  - Quasi-periodic stream processing by individual threads.
  - Worst-case latency: sum of deadlines.

Insights into Flow Characteristics

- Miss rate of data stream
  - Accommodates incomplete sensor readings.
  - Allows for controlled deadline misses.
- State vs. state delta communication
  - Data reduction technique.
  - Events as state transitions.
  - Requirement for guaranteed delivery.
- Data accuracy
  - Reading accuracy.
  - Computational error accumulation.
- Information flow as state transition

End-To-End Response

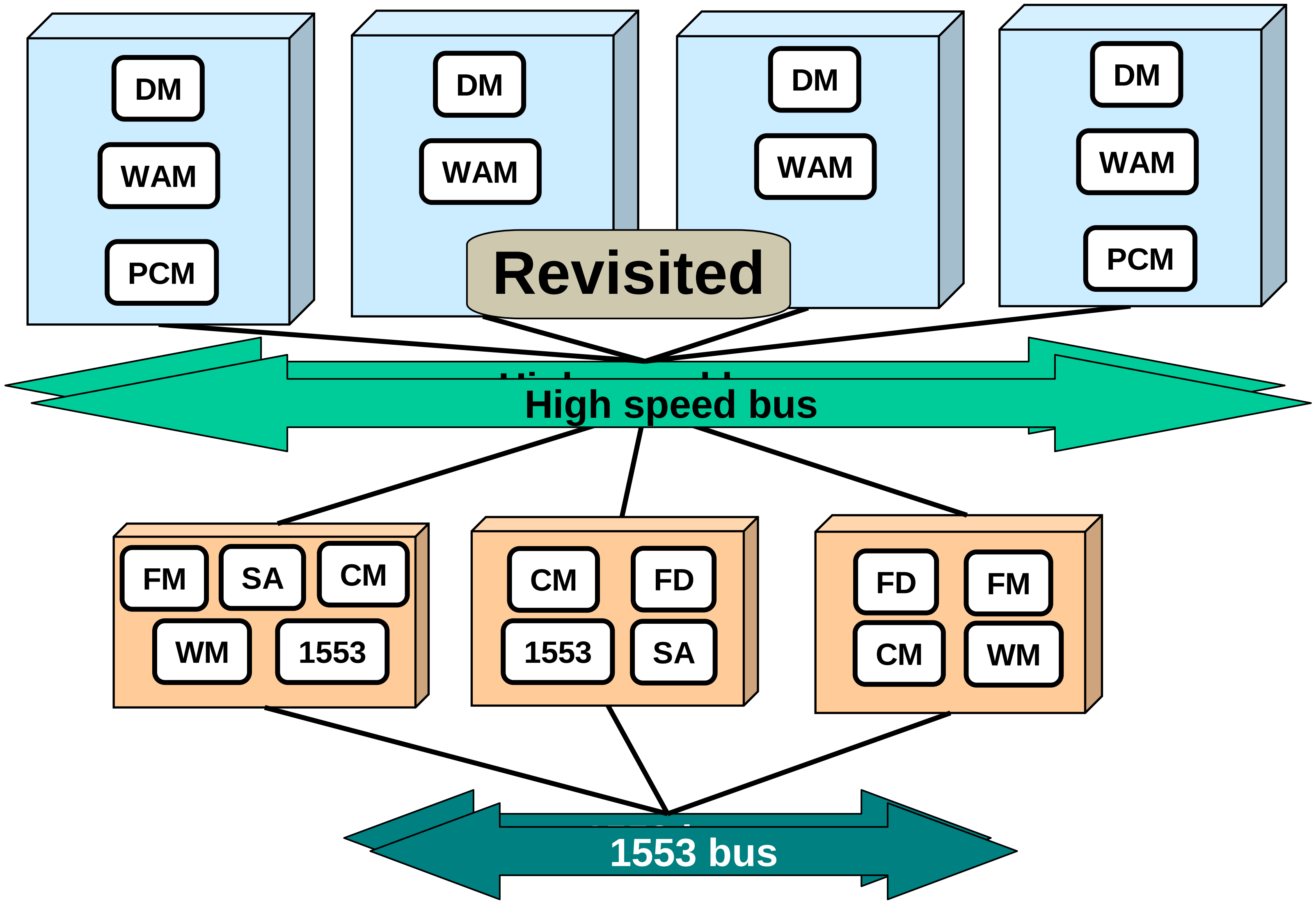


Flow failures
Guaranteed delivery
Completion/Exception reporting
Independent observation

Analysis & Design of an Avionics System

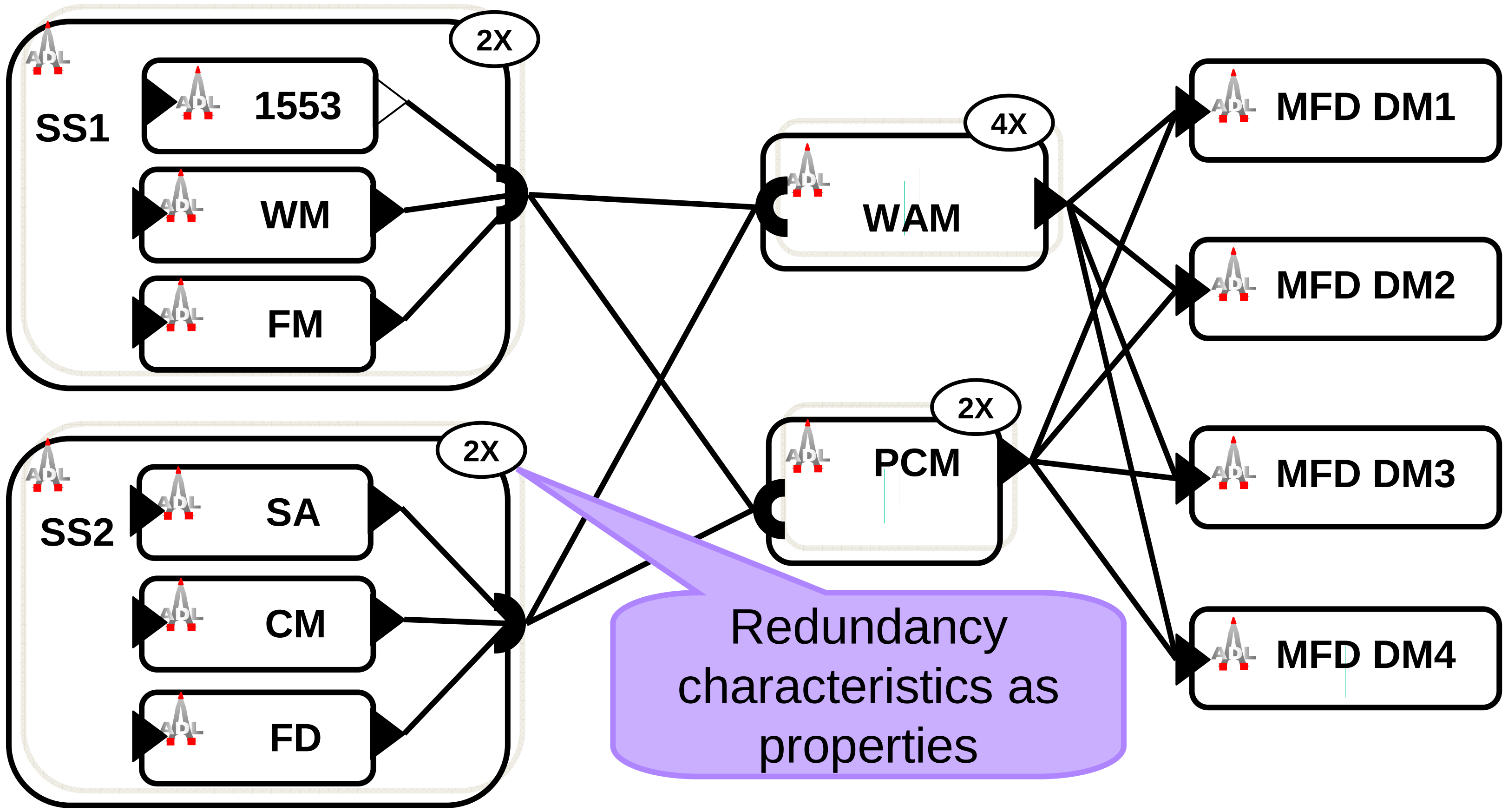
- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering

System Redundancy

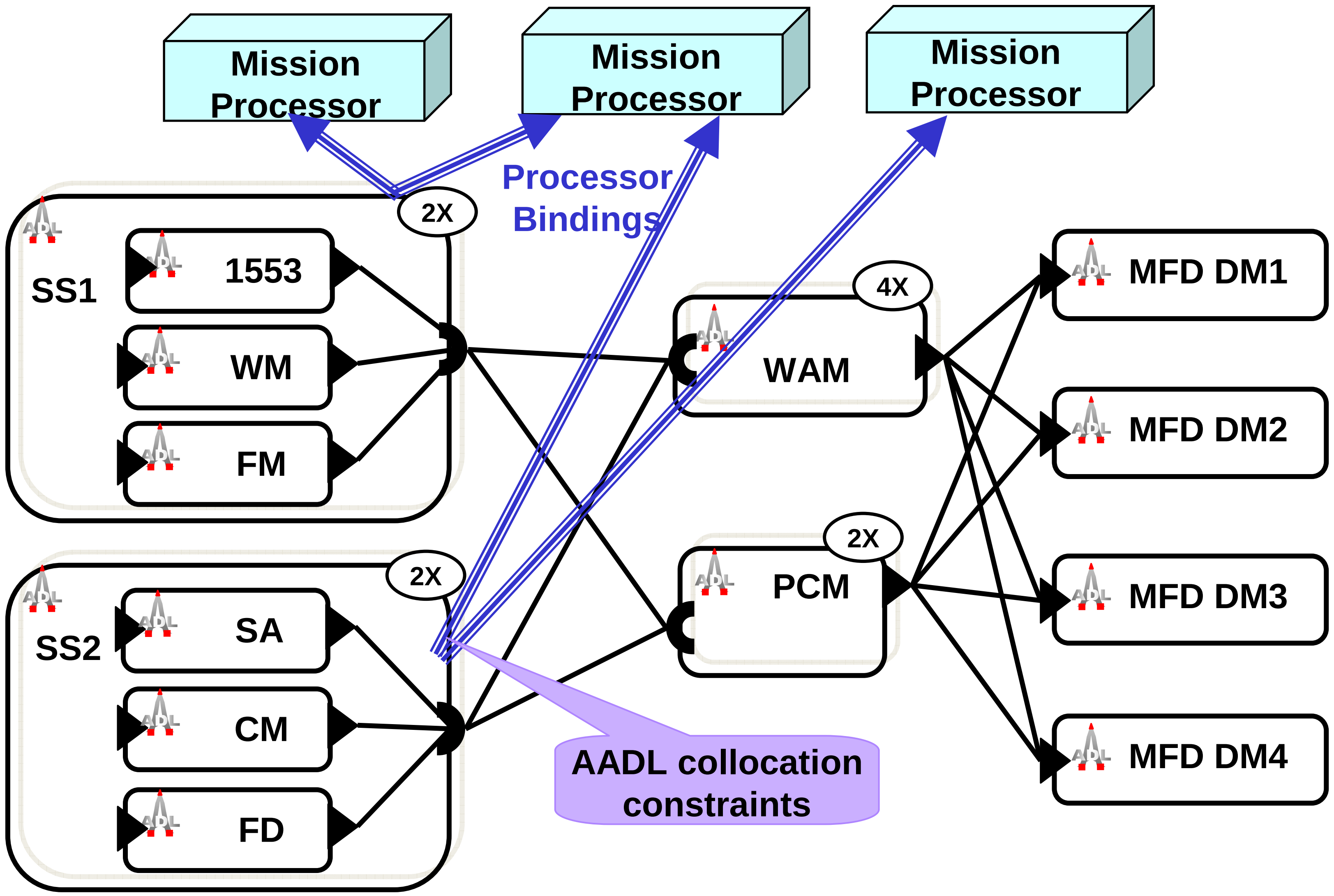


Redundancy Specification

- Grouping into subsystems for configuration
- Non-redundant component replication

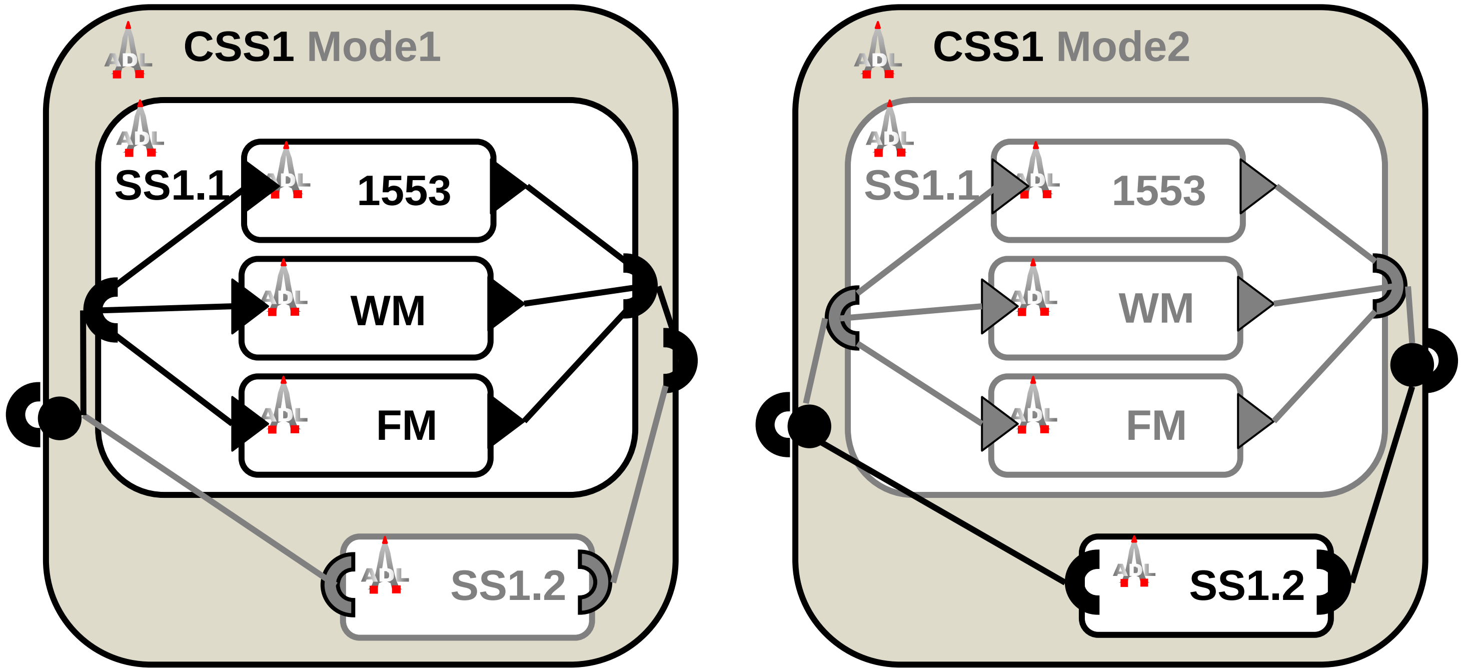


Runtime Co-location Constraints

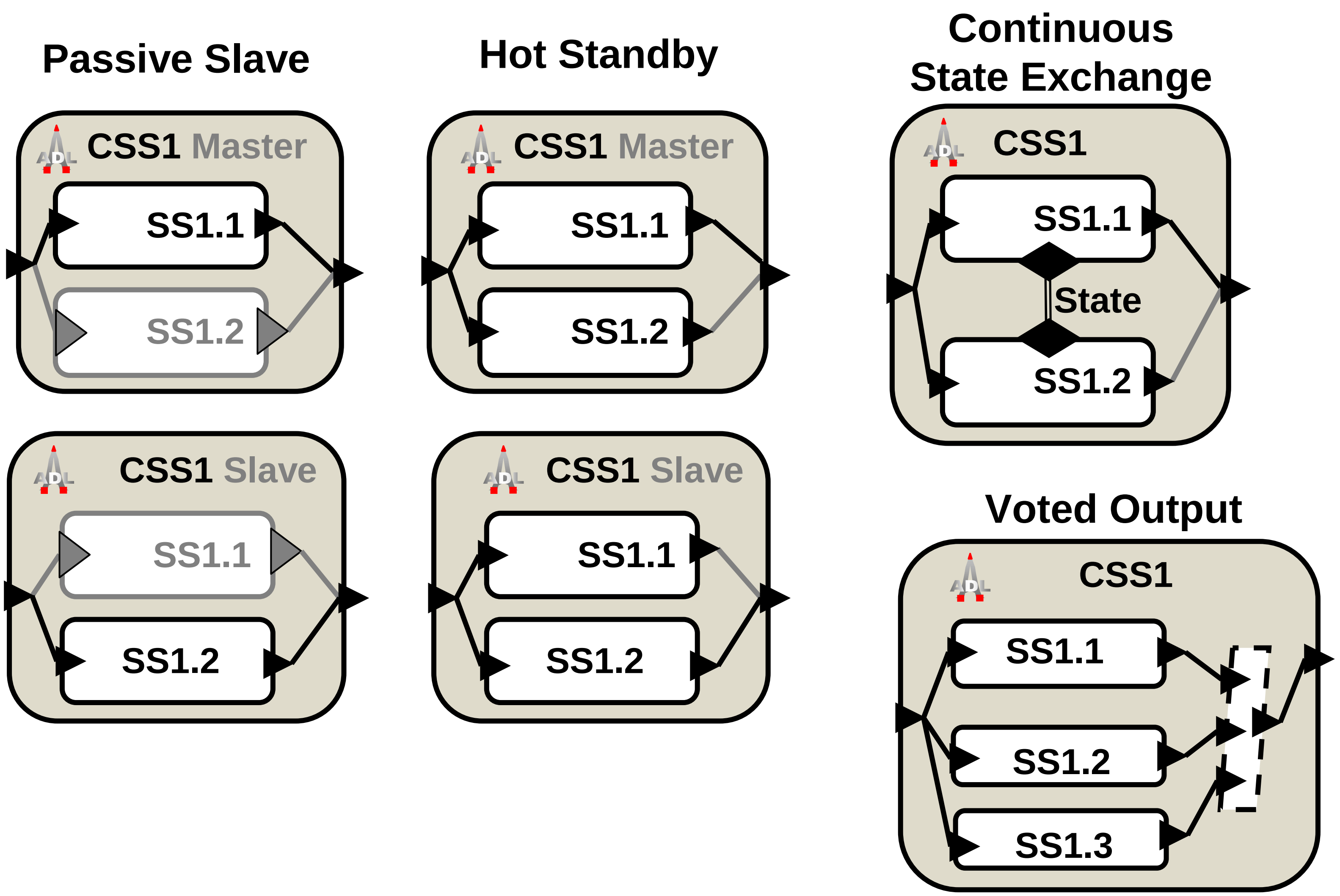


Replication as AADL Pattern

- Encapsulate redundancy replication in system
- Use mode to specify alternative configurations
- Specify processor and memory binding constraints

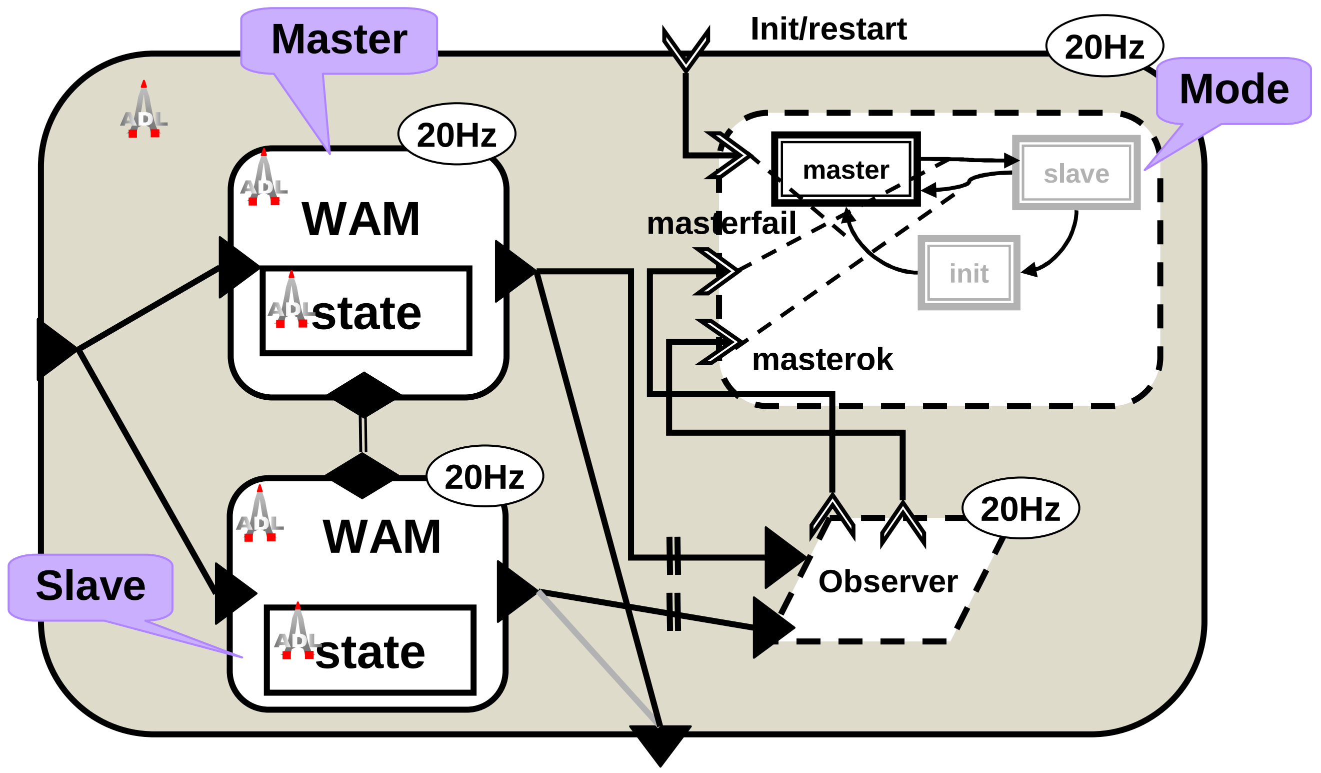


Master/Slave Configurations



Master Slave Synchronization

- External and internal mode control
- Errors reported as events
- Supports reasoning about master/slave logic



Observations On System Redundancy

- Redundancy as an abstraction
  - Multiple redundant instances
  - Grouping of redundant instances
  - Redundancy protocol selection
  - Deployment constraints
- Redundancy mechanism as pattern
  - An orthogonal architecture view
  - Nominal & anomalous behavior
  - Modeling of redundancy logic

Understandable and analyzable

AADL Fault Handling

- Application language exception handling
  - not represented in AADL.
- Thread-level recovery
  - recovery entrypoint execution to prepare for next dispatch.
- Thread unrecoverable
  - Reported as error event.

Fault Management

- Fault containment
  - Process as a runtime protected address space.
- Fault recovery
  - Within application code & thread local.
  - Error propagation.
- Propagated error management
  - Propagation through event connections.
  - Trigger reconfiguration through mode switching.
  - Monitoring & decision making through health monitor.

Event queue processing by aperiodic & periodic threads

Temporary Load Shedding

- Scheduling priority-based shedding
  - Fixed priority: lowest rate first; urgency property controls who.
  - EDF: spread the shedding across threads.
- Occasional deadline misses
  - Data sample miss already handled by co
  - Multiple misses per time window.

Affected by scheduling protocol

- AADL: Flow/connection property
  - Expected and provided degree of stream completeness.

Explicit Load Management

- Detection mechanism modeled in AADL
  - Deadline miss as processor event.
- Service level reduction via AADL mode
  - Mode-specific property values.
  - Thread internal modes
    - Different service levels within application component.
    - Different precision, different algorithm, …
  - Alternative active thread configurations
    - Load balancing via AADL mode.
    - Preconfigured alternate bindings.

Analysis & Design of an Avionics System

- Preemptive Scheduling and Data Flow
- Scheduling and Real-time Performance
- To Poll or Not to Poll
- Partitions & Communication Timing
- End-to-end Flows
- Many Uses of Modes
- System Safety Engineering

System Dependability and Safety

- Enables integration of system dependability and safety analyses with architectural design
  - Capture the results of
    - Hazard analysis.
    - Component failure modes & effects analysis.
  - Specify and analyze
    - Partition isolation/event independence.
    - Fault trees.
    - Markov models.
- Advantages
  - Ensures consistency between dependability and safety models and architectural design.
  - Enables cross-checking among analysis models.
  - Reduces specification and verification effort.

Fault Avoidance With AADL

- Support to identify and prevent design faults and to ensure correct runtime characteristics.
- Language features for fault avoidance that
  - Encourage and enforce safe software engineering practices.
  - Provide explicit space and time partitioning.
  - Clearly define system fault and error behaviors.
  - Provide clear specification of safety and certification properties.
  - Ready incorporates specialized safety/fault tolerance hardware.

Fault-Tolerance and Safety Features

- Processes may be time and space partitioned
- Safety/design assurance level can be specified for any component
- Hazardous run-time capabilities enabled on a per-process basis
- Executive consensus protocol is plug-replaceable
- Message data errors detected and reported
- Well-defined process error handling semantics

An Error Model Extension

- Supports reliability, availability, maintainability, safety, and related specification and analysis activities.
- Useful in system safety certification and qualification.
- Optional set of declarations and associated semantics.
- Facilitates a variety of analyses
  - Top-down hazard analysis.
  - A bottom-up failure modes and effects analysis.
  - Fault tree and stochastic process analyses.
  - Safety, reliability, availability, maintainability.
  - Integrated analyses.
- Provides a representation of
  - System hazards.
  - Component failure modes.

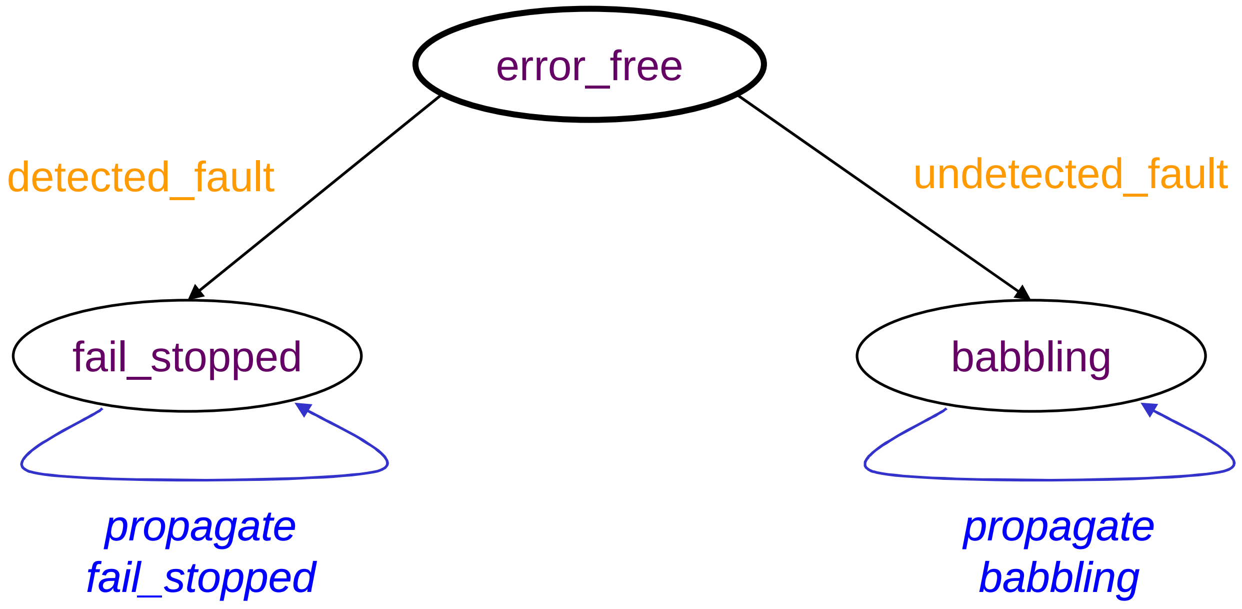
Error Annex Foundations

- Based on the concepts and terminology defined by IFIP WG10.4.
- Compatible with the core AADL standard.
- Coupled with AADL core capabilities provides
  - Fault avoidance,
  - Fault tolerance,
  - Integrated dependability analyses.

Error Model Description

Error model descriptions may declare

- Fault events (in subcomponents section)
- Additional error models (in modes section)
- Error mode transitions (in modes section)



Reliability Modeling

- Model generators output human-readable & structured models.
- Capture data from top-down hazard analysis.
- Capture data from bottom-up failure modes and effects analysis.
- Enables multiple integrated system safety checks and analyses.

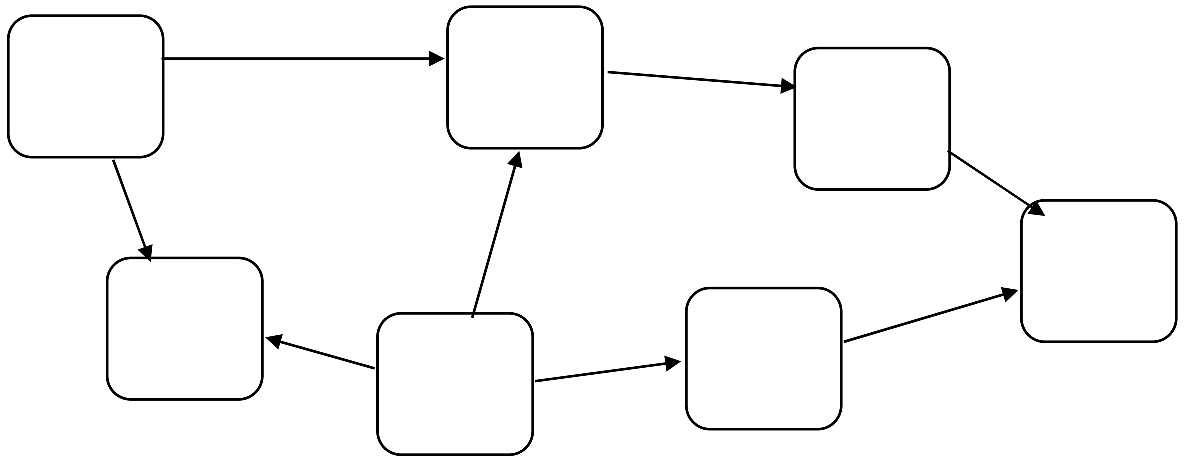
Integrated Analysis Capabilities

- Expandable set of analyses of a system specification
  - Basic error model states and transitions.
  - Partition isolation analysis.
  - Stochastic concurrent process model
    - Interfacing with a Markov chain as the reachable non-death states of stochastic concurrent process model.
    - Interfacing with Markov chain analysis tools (e.g. SURE from NASA Langley, UltraSAN from University of Illinois, SHARPE from Duke).

Partition Isolation Analysis

- Partitions support software error containment.
- Safety level properties for analysis and certification
  - RTCA/DO-178B defines 5 failure categories
    - Catastrophic
    - Hazardous
    - Major
    - Minor
    - No effect
  - Lower category defect must not affect higher category component.
- Significant reduction of re-certification costs.

Isolation Verification Algorithm

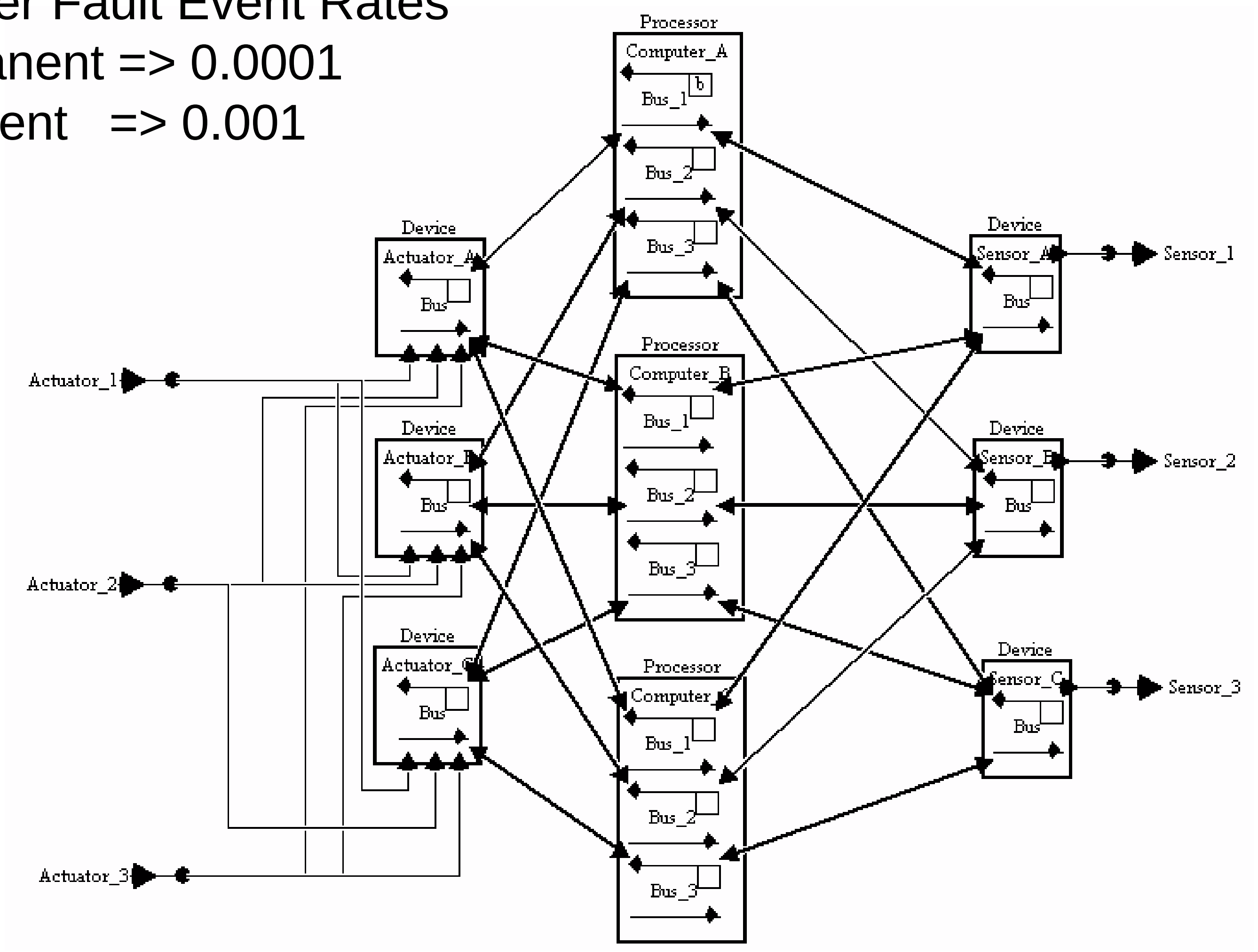


Analysis tool constructs “can affect” relationships:

- Port-to-port data connection (unless specified to the contrary).
- Software component hosted-on hardware component.
- Shared data structures.
- Scheduling attributes impacts on potential timing interference.
- Software component affects on processor or system.

Example Redundant System

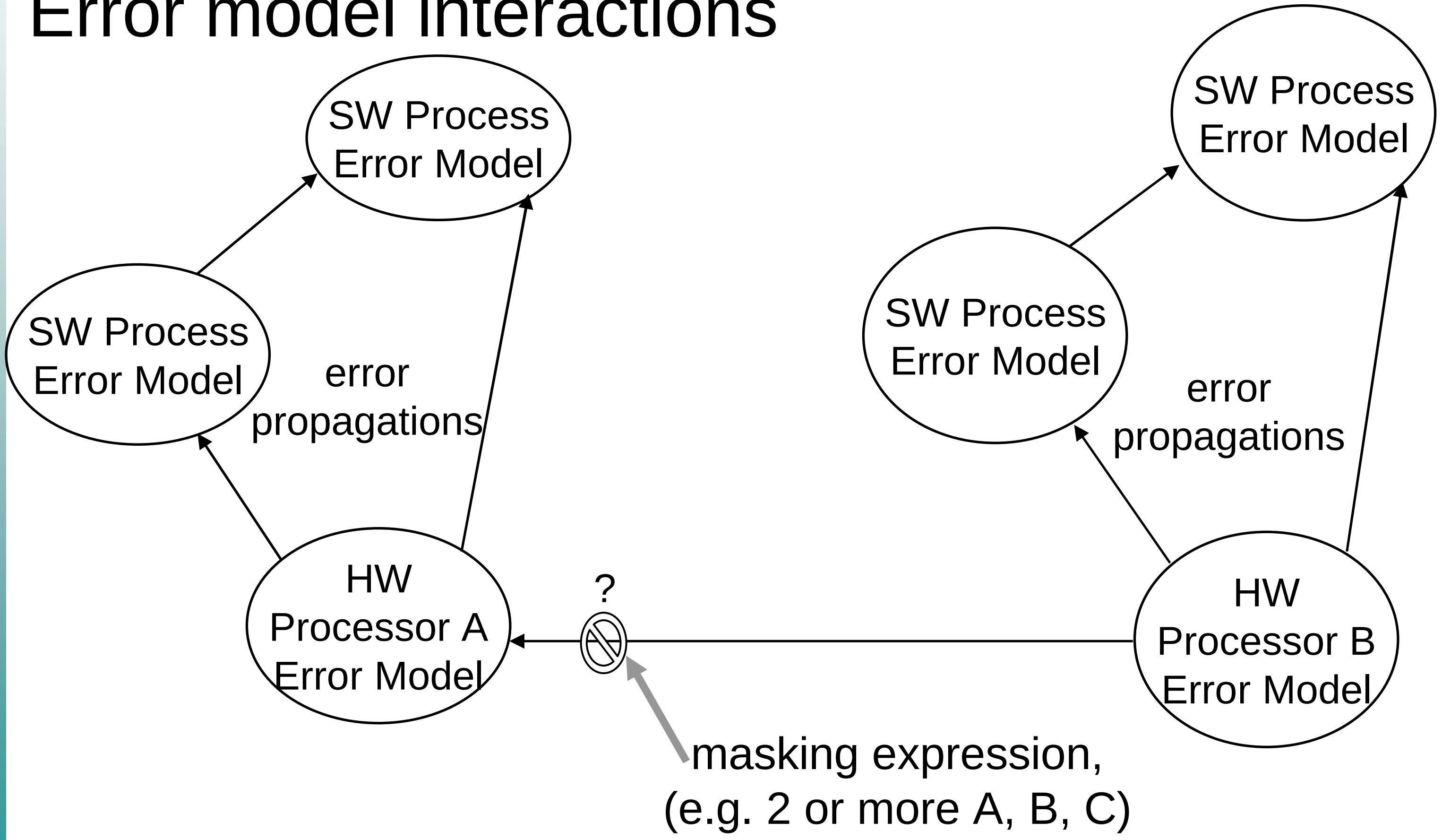
Computer Fault Event Rates
- Permanent => 0.0001
- Transient => 0.001



Stochastic Concurrent Process Reliability Model

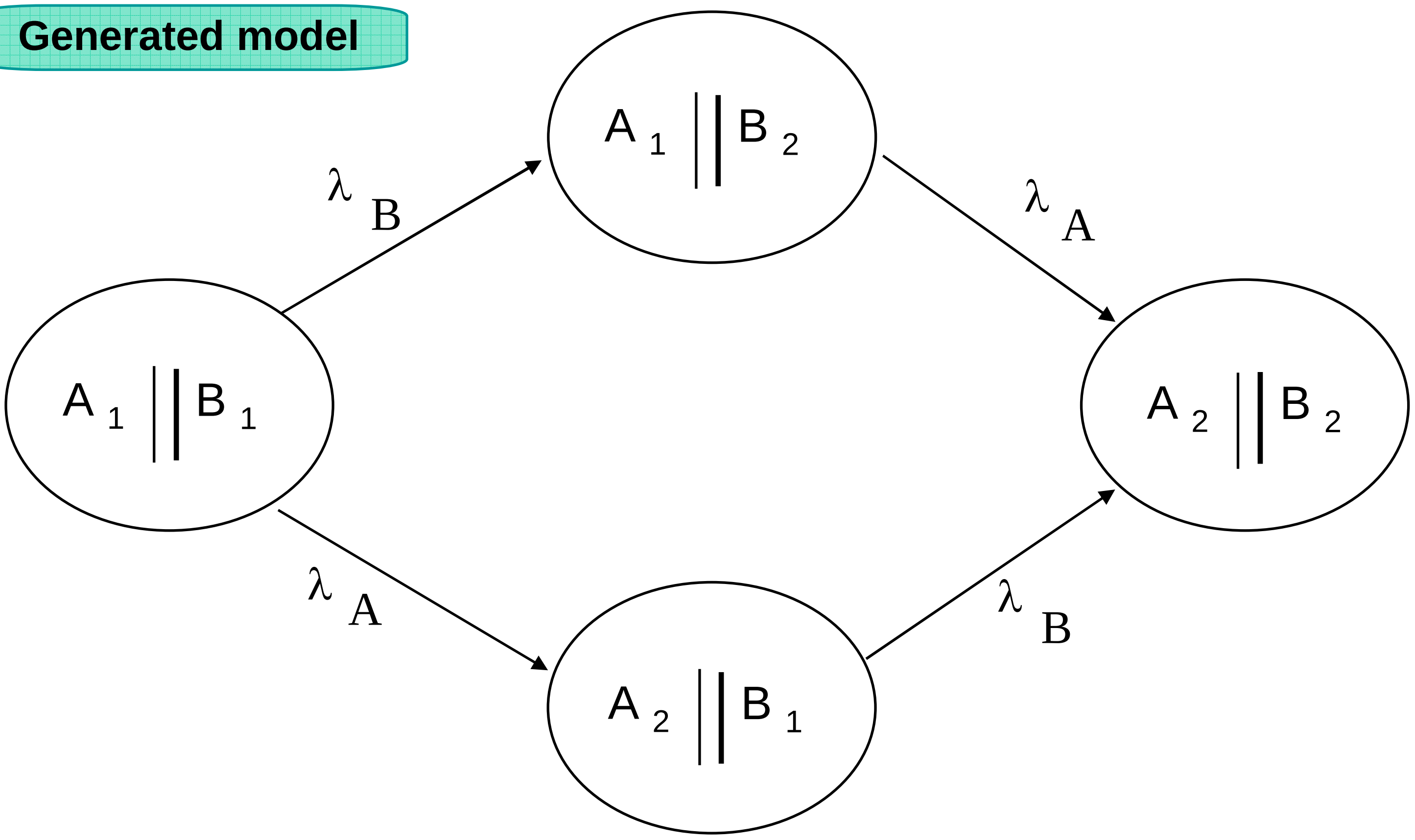
Error models mapped to system components

Error model interactions



Reachable States: A Markov Chain

Generated model



Example Analysis Results

```
SURE V7.9.8   NASA Langley Research Center

           LOWERBOUND   UPPERBOUND   COMMENTS         RUN #1
---------- -----------  -----------  --------------------------
           3.05514e-06  3.05513e-06  <prune 3.5e-13> <ExpMat>


103055 PATH(S) TO DEATH STATES 11939 PATH(S) PRUNED
HIGHEST PRUNE LEVEL = 1.29776e-15
Q(T) ACCURACY >= 7 DIGITS
480.733 SECS. CPU TIME UTILIZED
```

Analyzable and Reconfigurable AADL Specifications for IMA System Integration

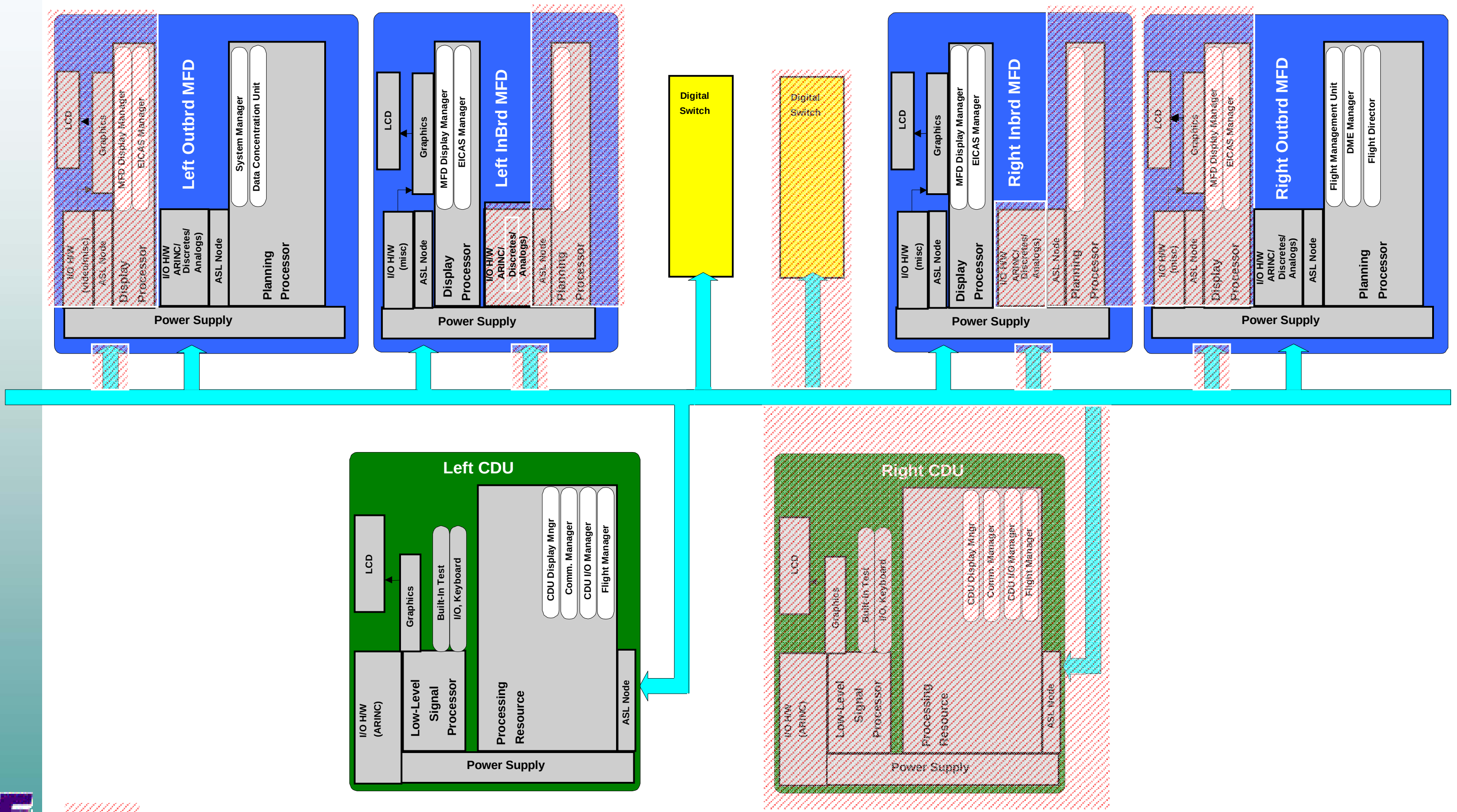
Outline

- Description of Model
- Description of Analysis

Proof of Concept Example

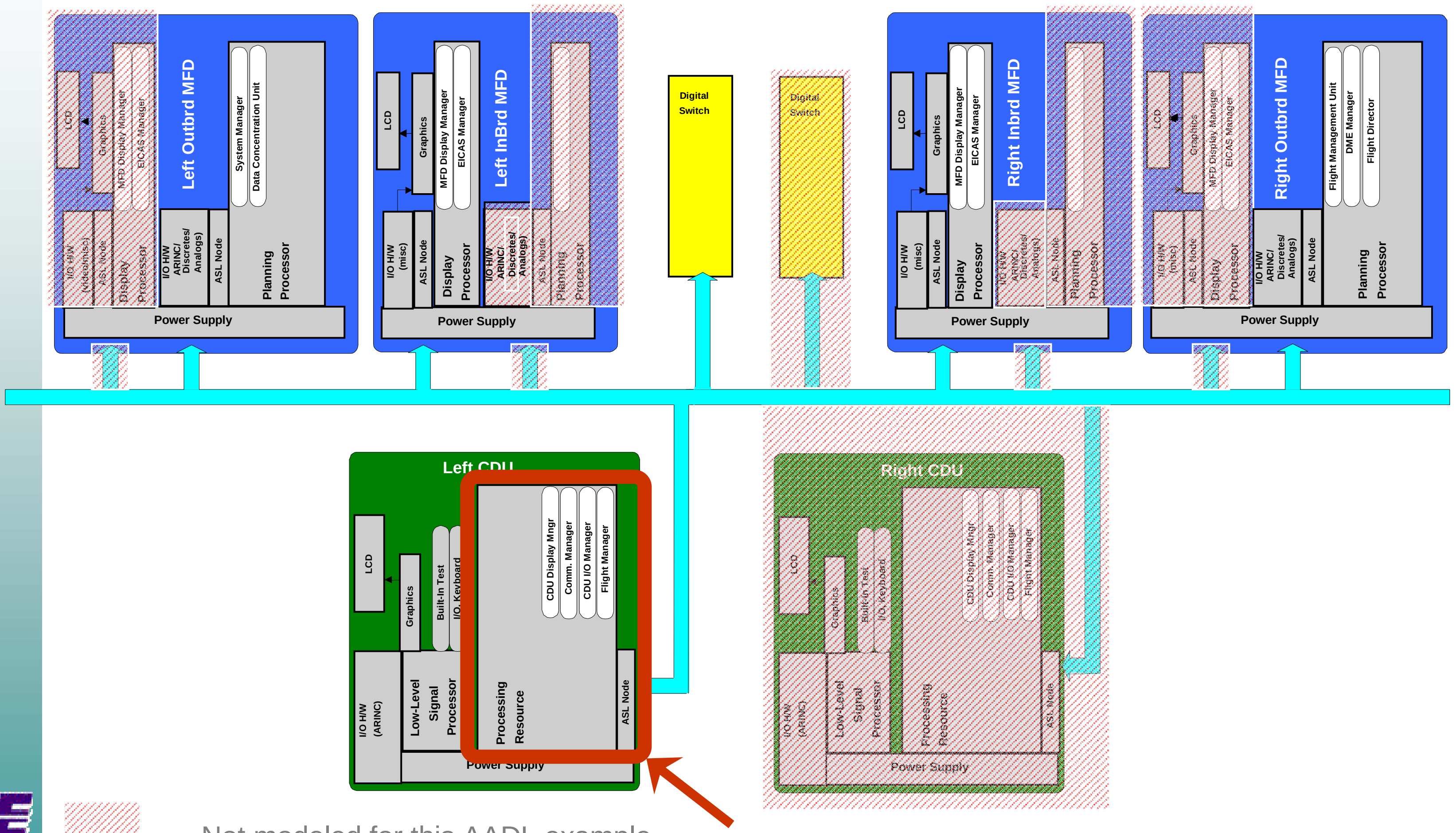
- Software task and communication architectures
- How they are bound to hardware in
  - Integrated Modular Architectures (IMA)
  - Federated Hardware Architectures
- Generic Display System with Rockwell Collin's Switched Ethernet LAN
  - Only LAN-related entities modeled
  - Model generated from Input/Output & Thread data stored in Database
- Model Size
  - 5 Common Processing Modules
  - 13 Virtual Machines
  - 90 Threads
  - 165 End-to-end Data Flows

Display System Architecture



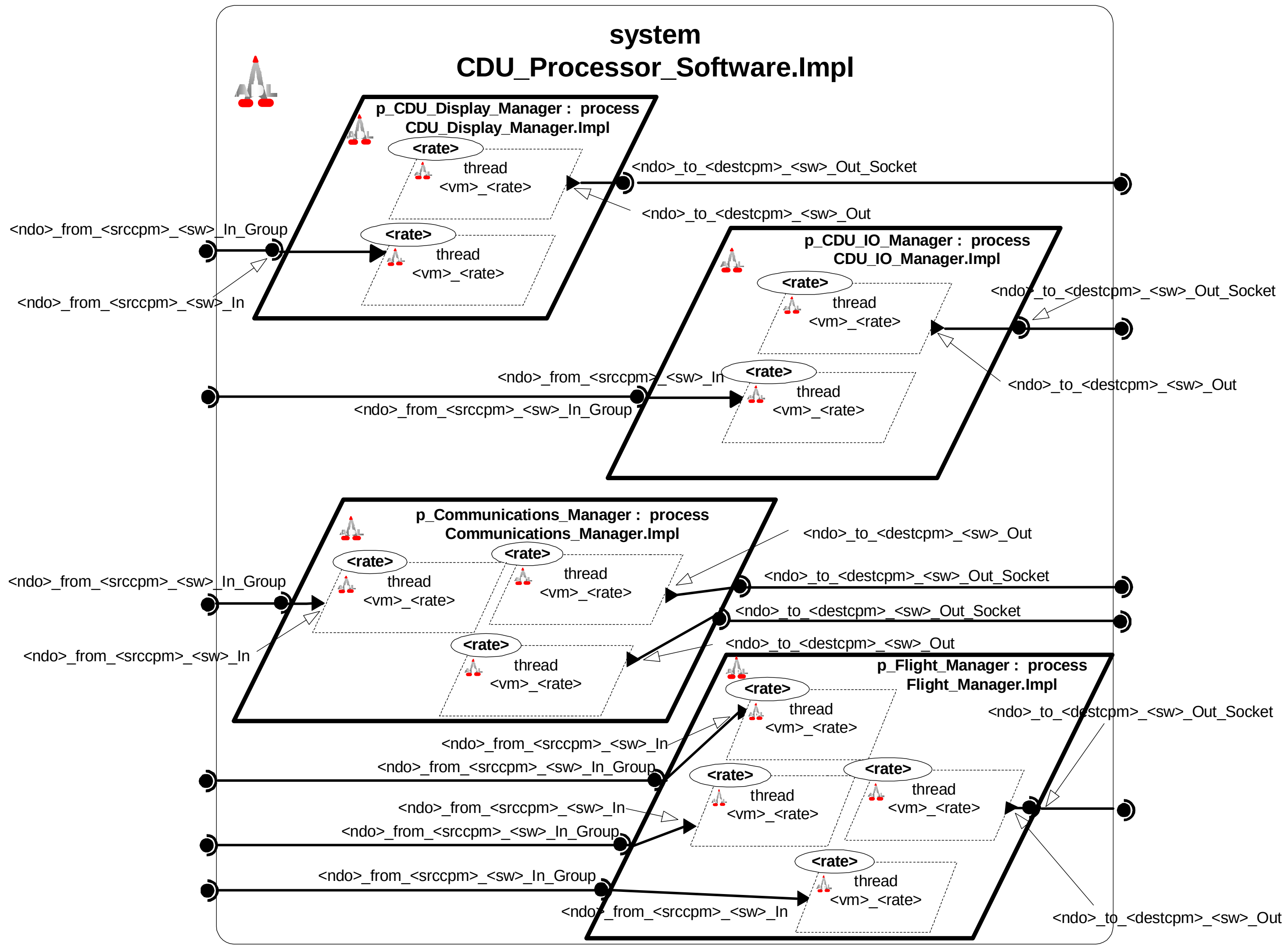
▨ - Not modeled for this AADL example

CDU Subsystem Architecture



▨ - Not modeled for this AADL example

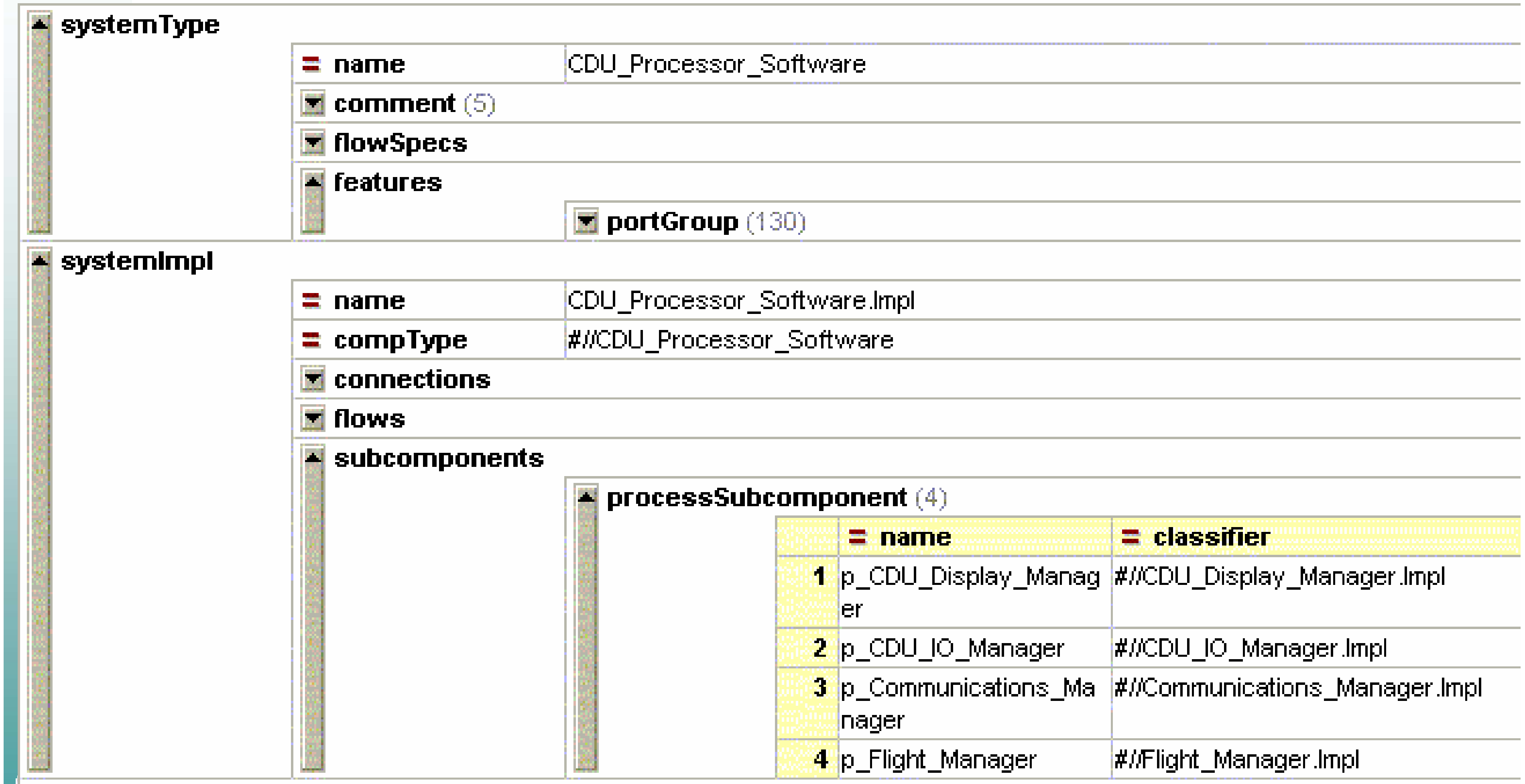
Graphical Software (Logical) View


system
CDU_Processor_Software.Impl
p_CDU_Display_Manager : process
CDU_Display_Manager.Impl
<rate>
thread
<vm>_<rate>
<ndo>_to_<destcpm>_<sw>_Out_Socket
<ndo>_to_<destcpm>_<sw>_Out
<ndo>_from_<srccpm>_<sw>_In_Group
<ndo>_from_<srccpm>_<sw>_In
p_CDU_IO_Manager : process
CDU_IO_Manager.Impl
<ndo>_to_<destcpm>_<sw>_Out_Socket
<ndo>_to_<destcpm>_<sw>_Out
<ndo>_from_<srccpm>_<sw>_In
<ndo>_from_<srccpm>_<sw>_In_Group
p_Communications_Manager : process
Communications_Manager.Impl
<ndo>_to_<destcpm>_<sw>_Out
<ndo>_to_<destcpm>_<sw>_Out_Socket
<ndo>_to_<destcpm>_<sw>_Out_Socket
<ndo>_to_<destcpm>_<sw>_Out
<ndo>_from_<srccpm>_<sw>_In_Group
<ndo>_from_<srccpm>_<sw>_In
p_Flight_Manager : process
Flight_Manager.Impl
<ndo>_to_<destcpm>_<sw>_Out_Socket
<ndo>_from_<srccpm>_<sw>_In
<ndo>_from_<srccpm>_<sw>_In_Group
<ndo>_from_<srccpm>_<sw>_In
<ndo>_from_<srccpm>_<sw>_In_Group
<ndo>_from_<srccpm>_<sw>_In_Group
<ndo>_from_<srccpm>_<sw>_In
<ndo>_to_<destcpm>_<sw>_Out

Textual Software View

```
system CDU_Processor_Software
 features
   CDU_Disp_EICAS_Cmds_to_LI_MFD_SW_L_Out_Socket : port group
    PG_CDU_Disp_EICAS_Cmds_Out;
   CDU_DM_Display_Buffer_NDO_from_CDU_L_SW_L_In_Group : port group
    PG_CDU_DM_Display_Buffer_NDO_In;
...
end CDU_Processor_Software;

system implementation CDU_Processor_Software.Impl
 subcomponents
  p_CDU_Display_Manager : process CDU_Display_Manager.Impl;
  p_CDU_IO_Manager : process CDU_IO_Manager.Impl;
  p_Communications_Manager : process Communications_Manager.Impl;
  p_Flight_Manager : process Flight_Manager.Impl;
 connections
     ...
 flows
     ...
end CDU_Processor_Software.Impl;
```

XML Software View



systemType

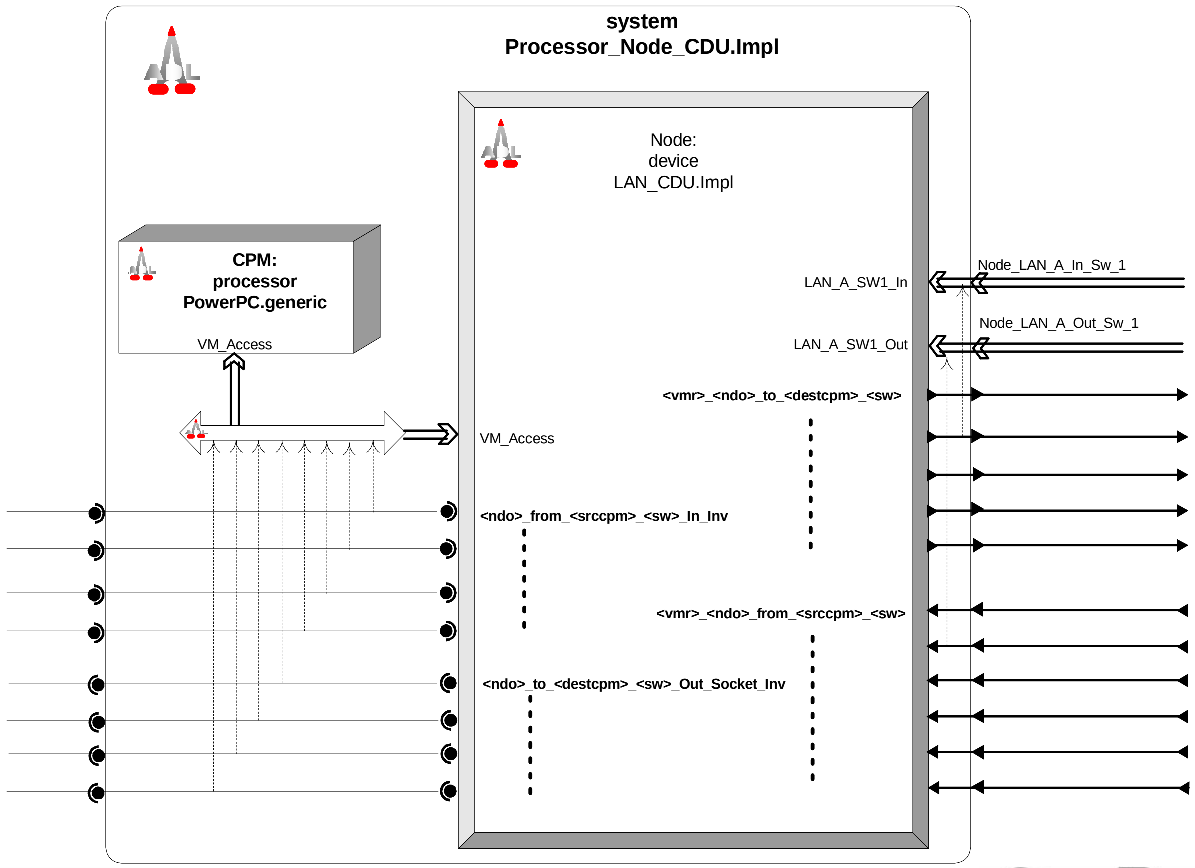
name	CDU_Processor_Software
comment (5)	
flowSpecs	
features	portGroup (130)

systemImpl

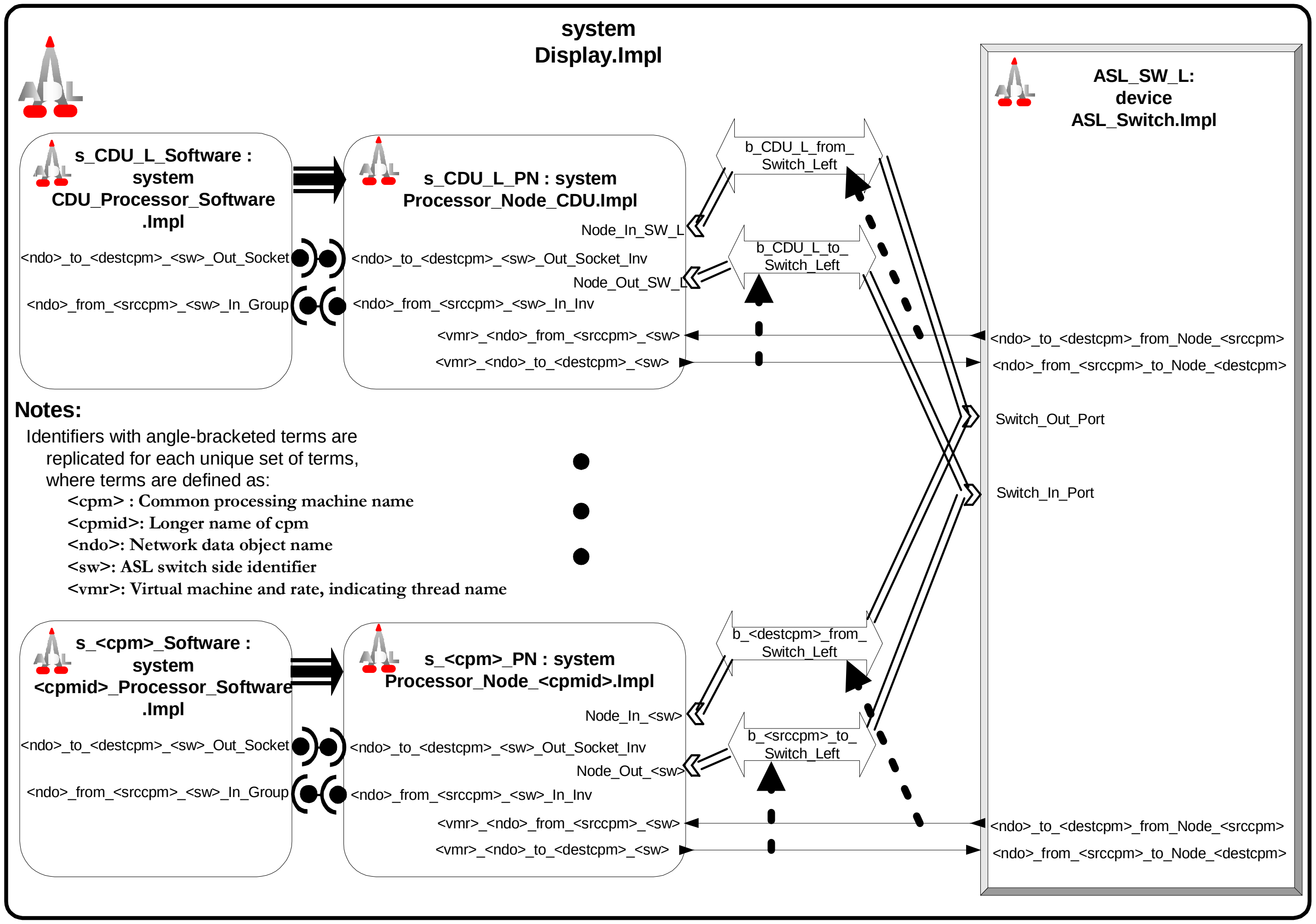
name	CDU_Processor_Software.Impl
compType	#//CDU_Processor_Software
connections	
flows	
subcomponents	processSubcomponent (4)

	name	classifier
1	p_CDU_Display_Manager	#//CDU_Display_Manager.Impl
2	p_CDU_IO_Manager	#//CDU_IO_Manager.Impl
3	p_Communications_Manager	#//Communications_Manager.Impl
4	p_Flight_Manager	#//Flight_Manager.Impl

Hardware (Physical) View with Mappings



Overall System Integration



system Display.Impl
s_CDU_L_Software : system CDU_Processor_Software .Impl
<ndo>_to_<destcpm>_<sw>_Out_Socket
<ndo>_from_<srccpm>_<sw>_In_Group
s_CDU_L_PN : system Processor_Node_CDU.Impl
Node_In_SW_L
<ndo>_to_<destcpm>_<sw>_Out_Socket_Inv
Node_Out_SW_L
<ndo>_from_<srccpm>_<sw>_In_Inv
<vmr>_<ndo>_from_<srccpm>_<sw>
<vmr>_<ndo>_to_<destcpm>_<sw>
b_CDU_L_from_ Switch_Left
b_CDU_L_to_ Switch_Left
ASL_SW_L: device ASL_Switch.Impl
<ndo>_to_<destcpm>_from_Node_<srccpm>
<ndo>_from_<srccpm>_to_Node_<destcpm>
Switch_Out_Port
Switch_In_Port
s_<cpm>_Software : system <cpmid>_Processor_Software .Impl
<ndo>_to_<destcpm>_<sw>_Out_Socket
<ndo>_from_<srccpm>_<sw>_In_Group
s_<cpm>_PN : system Processor_Node_<cpmid>.Impl
Node_In_<sw>
<ndo>_to_<destcpm>_<sw>_Out_Socket_Inv
Node_Out_<sw>
<ndo>_from_<srccpm>_<sw>_In_Inv
<vmr>_<ndo>_from_<srccpm>_<sw>
<vmr>_<ndo>_to_<destcpm>_<sw>
b_<destcpm>_from_ Switch_Left
b_<srccpm>_to_ Switch_Left
<ndo>_to_<destcpm>_from_Node_<srccpm>
<ndo>_from_<srccpm>_to_Node_<destcpm>

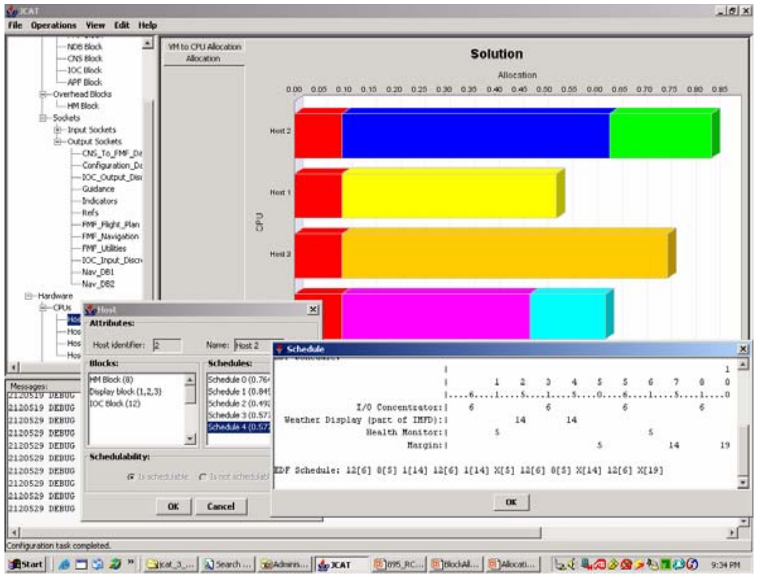
Notes:

Identifiers with angle-bracketed terms are replicated for each unique set of terms, where terms are defined as:

- <cpm> : Common processing machine name
- <cpmid>: Longer name of cpm
- <ndo>: Network data object name
- <sw>: ASL switch side identifier
- <vmr>: Virtual machine and rate, indicating thread name

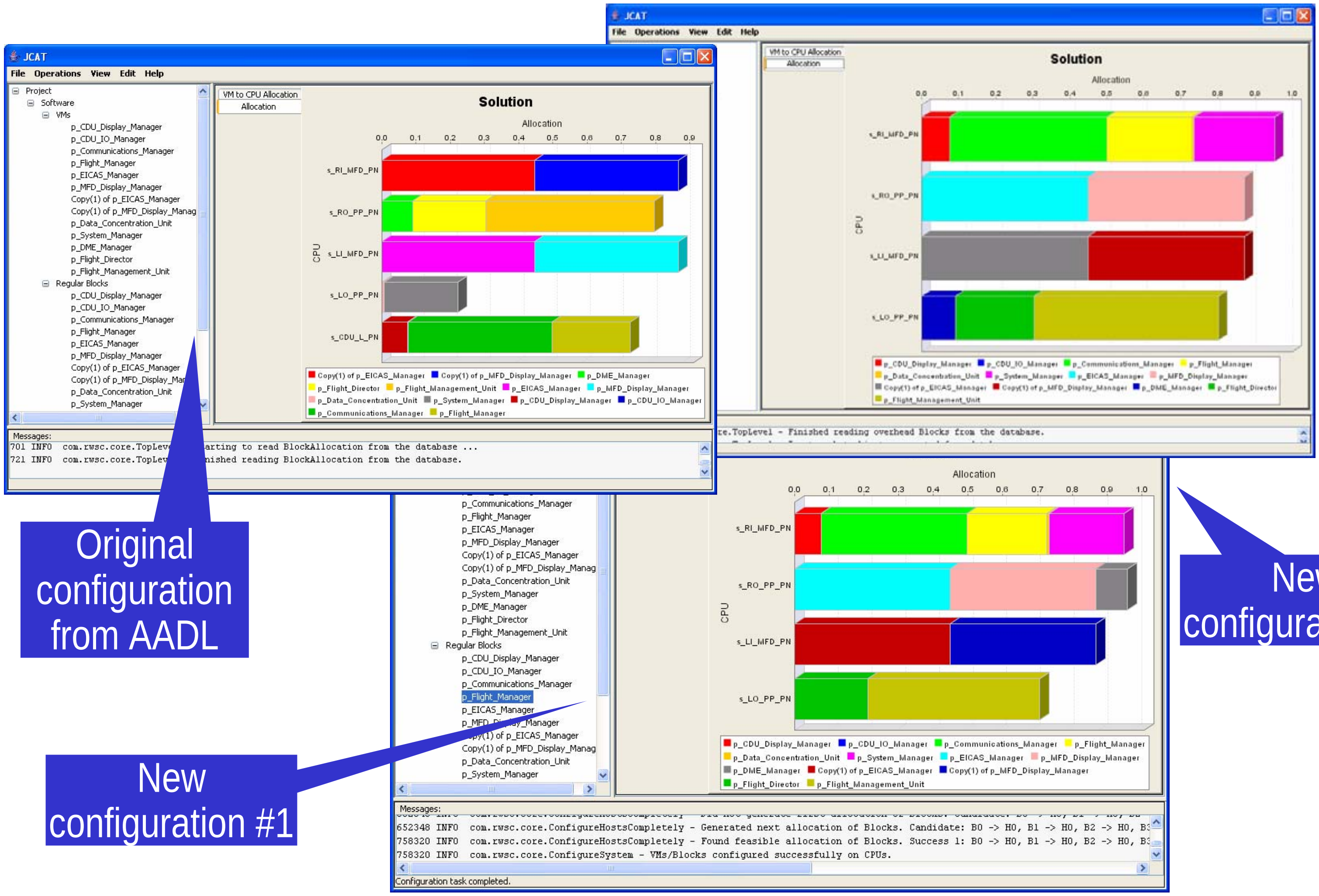
Analysis and Reconfiguration Tool

- System generation from Translated XML AADL
  - Automatic schedule generation
  - Allocation of VMs to hosts
- System analysis
  - Schedulability, rate-monotonic analysis
  - Network analysis
- Editing and visualization
  - Direct manipulation, tree view
  - Graphs, tables, trade studies



```
                                  |                                                    1
                                  |       1    2    3    4    5    5    6    7    8    0
                                  |....6....1....5....1....5....0....6....1....5....1....0
             I/O Concentrator:|    6              6              6              6
 Weather Display (part of IMFD):|         14        14
               Health Monitor:|       5                             5
                       Margin:|                        5               14         19

EDF Schedule: 12[6] 8[5] 1[14] 12[6] 1[14] X[5] 12[6] 8[5] X[14] 12[6] X[19]
```

Multiple Configurations for Trade-Off Studies



```
Messages:
701 INFO   com.rwsc.core.TopLev    arting to read BlockAllocation from the database ...
721 INFO   com.rwsc.core.TopLev    nished reading BlockAllocation from the database.
```

```
re.TopLevel - Finished reading overhead Blocks from the database.
```

```
Messages:
652348 INFO   com.rwsc.core.ConfigureHostsCompletely - Generated next allocation of Blocks. Candidate: B0 -> H0, B1 -> H0, B2 -> H0, B3
758320 INFO   com.rwsc.core.ConfigureHostsCompletely - Found feasible allocation of Blocks. Success 1: B0 -> H0, B1 -> H0, B2 -> H0, B3
758320 INFO   com.rwsc.core.ConfigureSystem - VMs/Blocks configured successfully on CPUs.
Configuration task completed.
```

Original configuration from AADL

New configuration #1

New configuration #3

Project Accomplishments

- Creation of Display System in AADL (Textual compiled to XML format)
- Translation to Analysis /Reconfiguration Tool
- Analysis of Initial Configuration for Fit
  - CPU Schedulability/Schedule
  - Network Latency
- Generation of Alternate Configurations

AADL In Use

EADS Military Aircraft

ASAAC Modelling with AAD

André Windisch
SAE AS-2 Meeting on AAD
Edinburgh, July 2004

NATO Fighter Reference

Architecture Specification
and
Automated Timing and Safety Analysis
for a
Large Avionics System

Steve Vestal
Larry Stickler
Dennis Foo Kune
Pam Binns
Nitin Lamba

Play Reference Architecture

DYNAMICS
mation Systems

AIRBUS France
TNI-Valiosys
E.N.S.T. Bretagne
I.R.I.T.
L.A.A.S.
O.N.E.R.A.-C.E.R.T.

AADL
Edinburgh
meeting
July 12-15,
2004

COTRE as an AA

- Funded by the French research departr
m.m), from 2002 to 2004
- Goal : Real Time architecture verificatio
behavioral point of view)
- Exploration project aiming to develop a
- Partners : AIRBUS, TNI, IRIT, LAAS, O

Analyzable and Reconfigurable AADL Specifications for IMA System Integration

David Statezni
Advanced Technology Center
Rockwell Collins, Inc.

SAE Aerospace
An SAE International Group

Rockwell Collins
Rockwell Scientific

esa

ASSERT

d proof based System and Software
ineering for Real Time systems

Eric Conquet
ESA/ESTEC
Software Engineering and Standardization
Noordwijk, The Netherlands

Tutorial Outline

- Background & Introduction
- Introduction to AADL
- AADL in Use
- **Summary and Conclusions**
  - Summary of AADL
  - Benefits of Using AADL
  - Final Observations

Summary of AADL Capabilities

- AADL abstractions separate application architecture concerns from runtime architecture concerns
- AADL incorporates a run-time architecture perspective through application system and execution platform
- AADL is effective for specialized views of embedded, real-time, high-dependability, software-intensive application systems
- AADL supports predictable system integration and deployment through model-based system engineering
- AADL component semantics facilitate the dialogue between application and software experts

Summary of AADL

- AADL provides a formal notation to capture the architecture specification of system in a single notation.
  - Formal → provides well defined abstractions to mathematical analysis
  - ADL → mathematical analysis of architecture
  - AADL → mathematical analysis of real-time, embedded, multiprocessor, safety-critical, fault tolerant systems – including both hardware and software components.
- AADL is a standard
  - Can be used with multiple projects & multiple toolsets
  - Enables a common and standard language to be used by multiple subcontractors and prime contractors
- AADL supports various levels of abstraction utilizing a precise notation
  - Analyze various levels of the system at various stages of development
  - Incrementally extend detail, depth, and analysis models

Dependability and Safety Summary

- Integration of dependability and system safety with architectural design
  - Ensures consistency of error models and design architecture.
  - Extensible to include additional models.
  - Enables cross-checking between models.
  - Reduces specification and verification effort.

Value of AADL-Based Development

- Early Prediction and Verification Tools
  - performance
  - reliability
  - system safety
- Component Compliance Verification Tools
  - functional interface
  - resource requirements
  - system safety
- System Integration and Verification Tools
  - workstation testing
  - system performance
  - system safety verification

Benefits

- Model-based system engineering benefits
  - Analyzable architecture models drive development
  - Predictable runtime characteristics at different modeling fidelity
  - Model evolution & tool-based processing
  - Prediction early and throughout lifecycle
  - Reduced integration & maintenance effort

- Benefits of AADL as SAE standard
  - Common component definitions across teams, documents
  - Single architecture model augmented with analysis properties
  - Interchange & integration of architecture models
  - Tool interoperability & extensible engineering environments
  - Aligned with UML engineering, potential adoption by UML

AADL Benefits

- Standard analyzable architecture description language → well defined documentation
- Supports integrated specification for application, system, and software engineering architectural requirements → avoid miscommunication
- Supports analysis of compositional effects and cross cutting impacts → manage complexity
- Supports early analysis, incremental development and model driven automated integration → reduce rework, rapid change, reduce risk and costs.

Final Observations

- Abstract but precise task & communication architecture
  - A bridge between control engineer & software system engineer
- Early insight into para-functional properties
- Runtime system generation leads to model-consistent implementation
- Leads to architecture patterns with predictable characteristics
- An extensible basis for a wide range of embedded systems analyses

Thank You

www.aadl.info

Joyce L Tokar, PhD
Pyrrhus Software
Phoenix, AZ
tokar@pyrrhusoft.com
480-951-1019