P.165

Distributed Systems and Ada

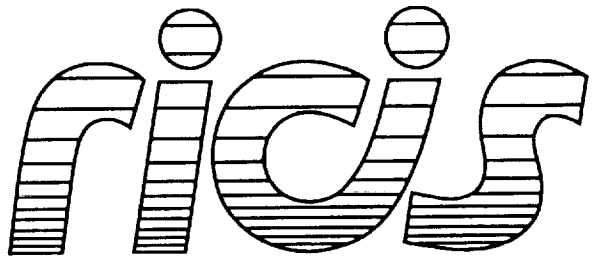
Final Report

Intermetrics, Inc.

June 30, 1989

Cooperative Agreement NCC 9-16
Research Activity No. SE.28

NASA Space Station Freedom Program Office
Level II
NASA Headquarters



Research Institute for Computing and Information Systems
University of Houston-Clear Lake

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a continuing cooperative agreement with UHCL beginning in May 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research and integrates technical results into the goals of UHCL, NASA/JSC and industry.

Distributed Systems and Ada

Final Report

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Intermetrics, Inc. Dr. Charles McKay served as RICIS research representative.

Funding has been provided by the NASA Space Station Freedom Program Office, Level II, NASA Headquarters, through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Marge Kirchosf of NASA Headquarters.

Final Report under University of Houston Research Activity No. SE-28

Distributed Systems and Ada
Part 1

June 30, 1989

Presenters: Dr. Bruce Burton
Dr. William Bail
Mr. David Barton
Intermetrics, Inc.

Note: NASA Cooperative Agreement NCC9-16

Intermetrics

OUTLINE

- **Introduction and Overview**
  - Development and Integration of Ada systems, Part 1
  - Real-time programming in Ada, Part 1
  - Tasks and Communications
  - Relation of Ada to the Underlying OS / UNIX / POSIX
  - Design and Verification of Distributed Systems
  - Conclusions, Part 1

OVERVIEW OF PRESENTATIONS

Purpose

These two briefings are designed to provide information to the Software I&V Study Group to help complete the I&V study task.

This information is taken from the areas of Ada real-time processing support, Ada runtime environments, Ada program construction, object-oriented design, and Ada/UNIX/POSIX interfaces.

CONTENTS OF BRIEFING #1

- Overview
- Development and integration of Ada systems, Part 1
- Real-time programming in Ada, Part 1
- Inter-Task Communications
- Relation of Ada to the underlying OS / UNIX / POSIX
- Design and Verification of Distributed Systems
- Conclusions, Part 1

CONTENTS OF BRIEFING #2

- Real-time programming in Ada, Part 2
- Development, integration , and verification of Ada systems, Part 2
- Fault tolerance and Ada
- Programming standards, techniques, and tools for Ada
- Conclusions, Part 2

THE SPEAKERS

- Dr. Bruce Burton, Director of Washington Division of Intermetrics
- Mr. David Barton, Senior Computer Scientist
- Dr. William Bail, Senior Computer Scientist

Intermetrics

OUTLINE

Introduction and Overview

→ **Development and Integration of Ada systems, Part 1**

Real-time programming in Ada, Part 1

Tasks and Communications

Relation of Ada to the Underlying OS / UNIX / POSIX

Design and Verification of Distributed Systems

Conclusions, Part 1

WHAT WILL BE PRESENTED

- **In Briefing 1**
  - -- An overview of how Ada systems are put together
  - -- An overview of how system components communicate
  - -- An overview of what these systems look like when deployed
- **In Briefing 2**
  - -- Verification considerations
  - -- Pragmatics (compiler selections, performance, system architectures)
  - -- Ada language considerations and details

BUILDING ADA SYSTEMS

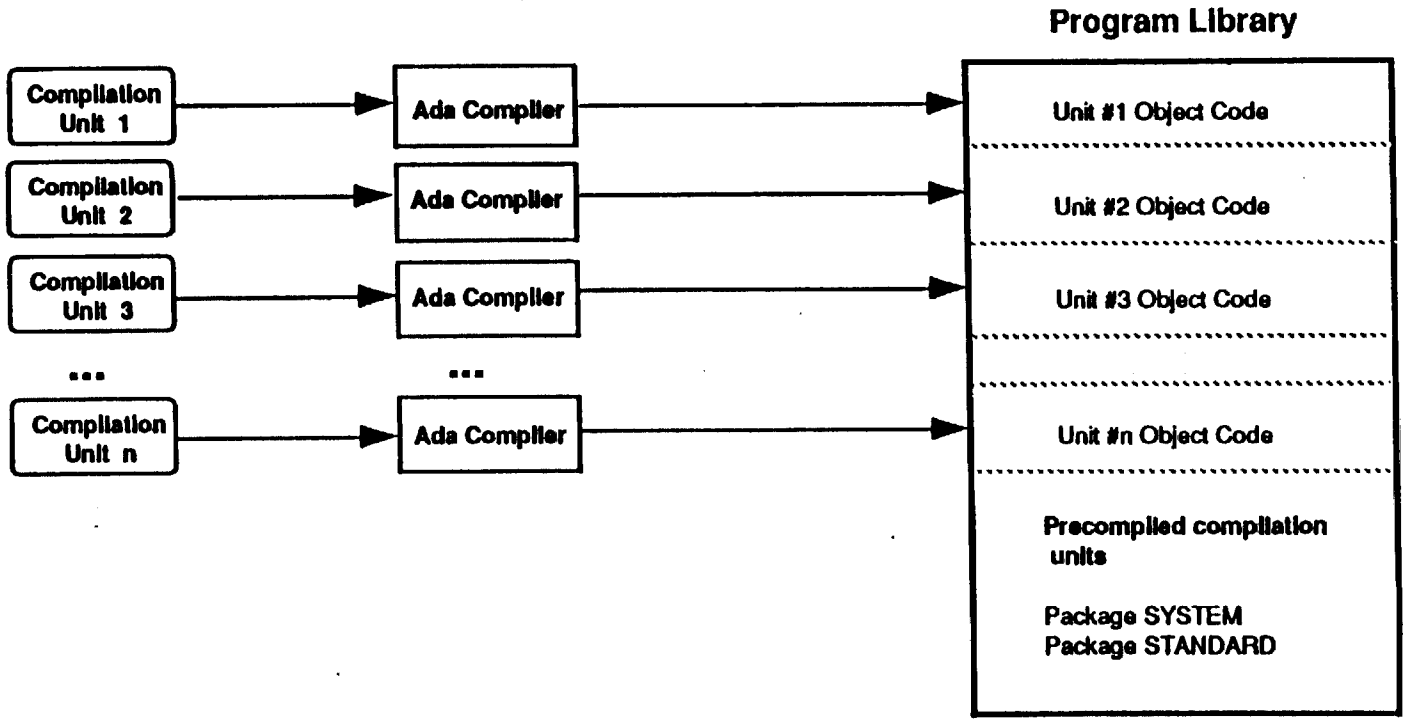
- **Ada was designed to facilitate the building of large systems in modules**
  - It supports separate compilation
  - It supports packages of resources
  - It supports the separation of **interface** definitions from **implementation** definitions

- **Central to this support is the Ada program library**

ADA PROGRAM LIBRARIES

- **Each library helps to create a single Ada program**
- **The library accumulates the modules as they are compiled**
  - -- These modules are called ***compilation units***
- **The library contains all program resources needed to build the program, e.g.**
  - -- All program modules
  - -- Standard type declarations, I/O routines, etc.
- **Some of these have been precompiled into the library,e.g.**
  - -- PACKAGE STANDARD
  - -- PACKAGE SYSTEM

THE COMPILATION PROCESS



Intermetrics

ADA PROGRAM BUILDING BLOCKS

What are these building blocks ?

- **Ada Compilation units**

  In Ada, a compilation unit can be a

  -- procedure declaration

  -- procedure body

  -- function declaration

  -- function body

  -- package declaration

  -- package body

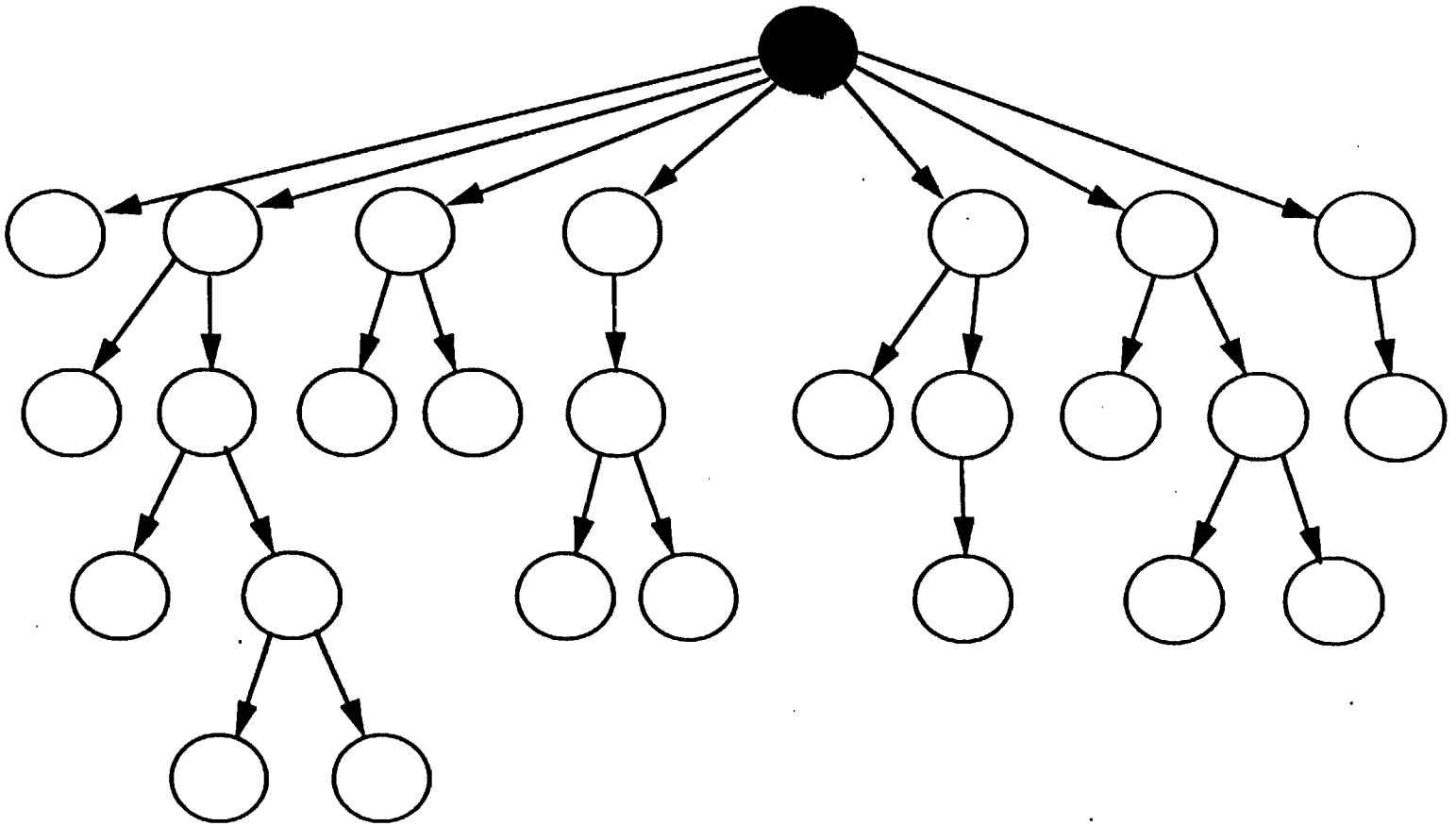
Intermetrics

SUBPROGRAMS

- **Procedures and Functions are collectively called Subprograms**
  - -- The main routine of every Ada program is a subprogram.
  - -- Subprograms contain declarations and executable Ada code.
  - -- Subprograms can contain declarations of subprograms, packages, and tasks
  - -- These nested subprograms, packages, and tasks are **not** compilation units.
    - They cannot be independently compiled as long as they are part of the surrounding module.

NESTING OF MODULES

Static Structure Chart for Compilation Unit



Intermetrics

PACKAGES

- **A Package is a collection of objects that can be used as resources by other compilation units.**
  - -- Similar in purpose to the FORTRAN subroutine libraries
- **These objects can include:**
  - -- Types
  - -- Data
  - -- Subprograms
  - -- Tasks
  - -- other Packages
- **Packages are not called**
  - -- Subprograms are called
  - -- Packages provide their visible contents for use by other program modules.
    - If a package contains subprograms, then these subprograms may be called.
- **e.g. a package for CRT graphics might contain**
  - -- callable routines to display information on a screen
  - -- callable routines to access information from a CRT
  - -- types appropriate to screen data, such as Position
  - -- tasks to refresh the screen

INTERFACES AND IMPLEMENTATIONS

- **To support modular construction, Ada allows developers to define module interfaces separately from module implementation details.**
  - -- Interfaces provide a stable framework for system development
  - -- Facilitate verification process.
  - -- Implementations can change without affecting interfaces.

- **Module declarations are used to define interfaces, while Module bodies are used to define implementations.**

- **Interfaces can be defined for**
  - -- procedures
  - -- functions
  - -- packages
  - -- tasks

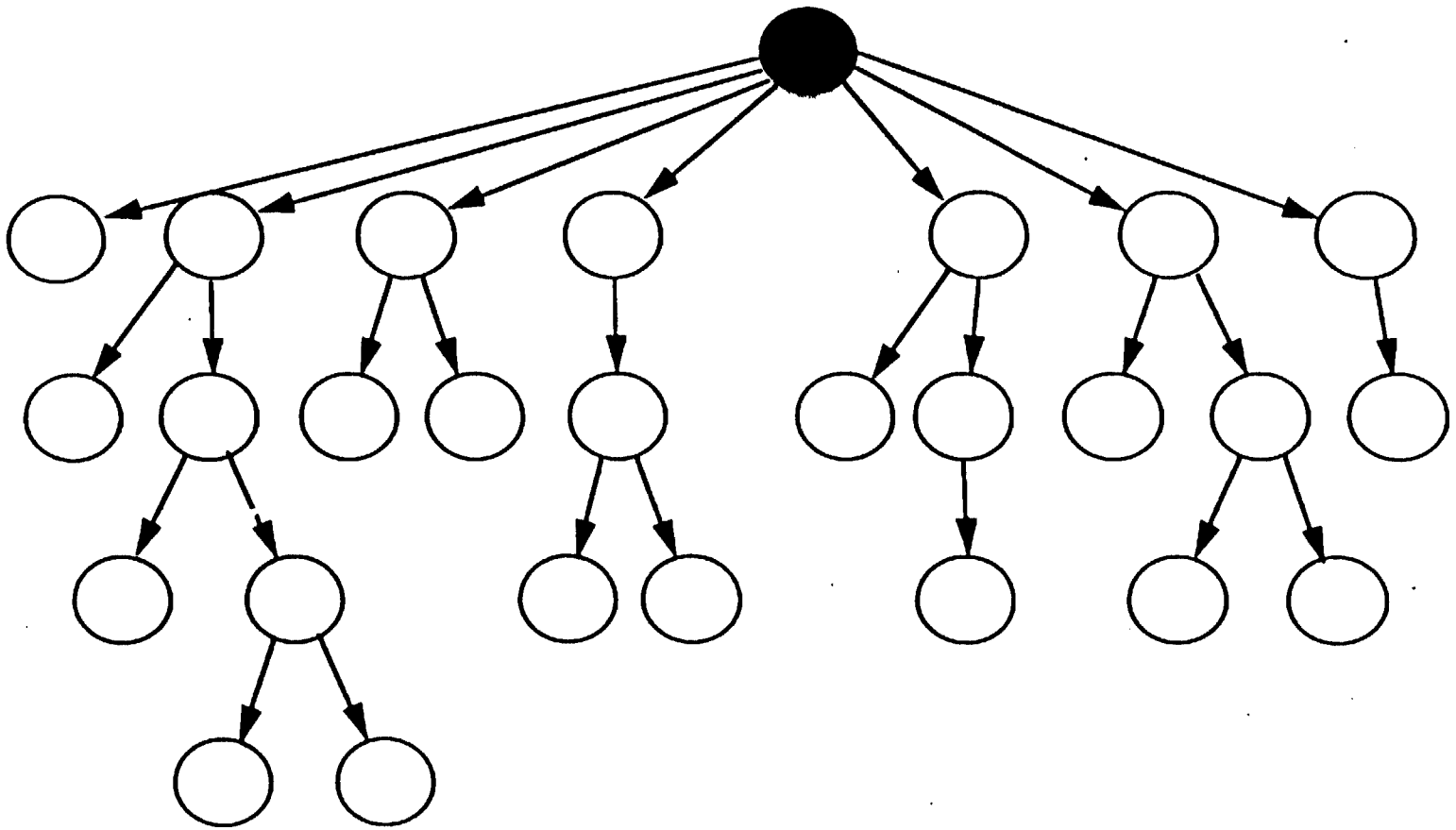
SUBUNITS

- **It is sometimes desirable to compile large compilation units in pieces.**
  - -- Top-down program development requires upper level routines to be written first.
  - -- Large compilation units are unwieldy, and time-consuming to compile
  - -- Testing and integration strategies may be made more efficient

- **Certain types of modules can be "un-nested" and compiled separately from their containing units**
  - -- Procedures,
  - -- functions,
  - -- task bodies,
  - -- package bodies

COMPILATION UNIT BEFORE SUBUNIT REMOVAL

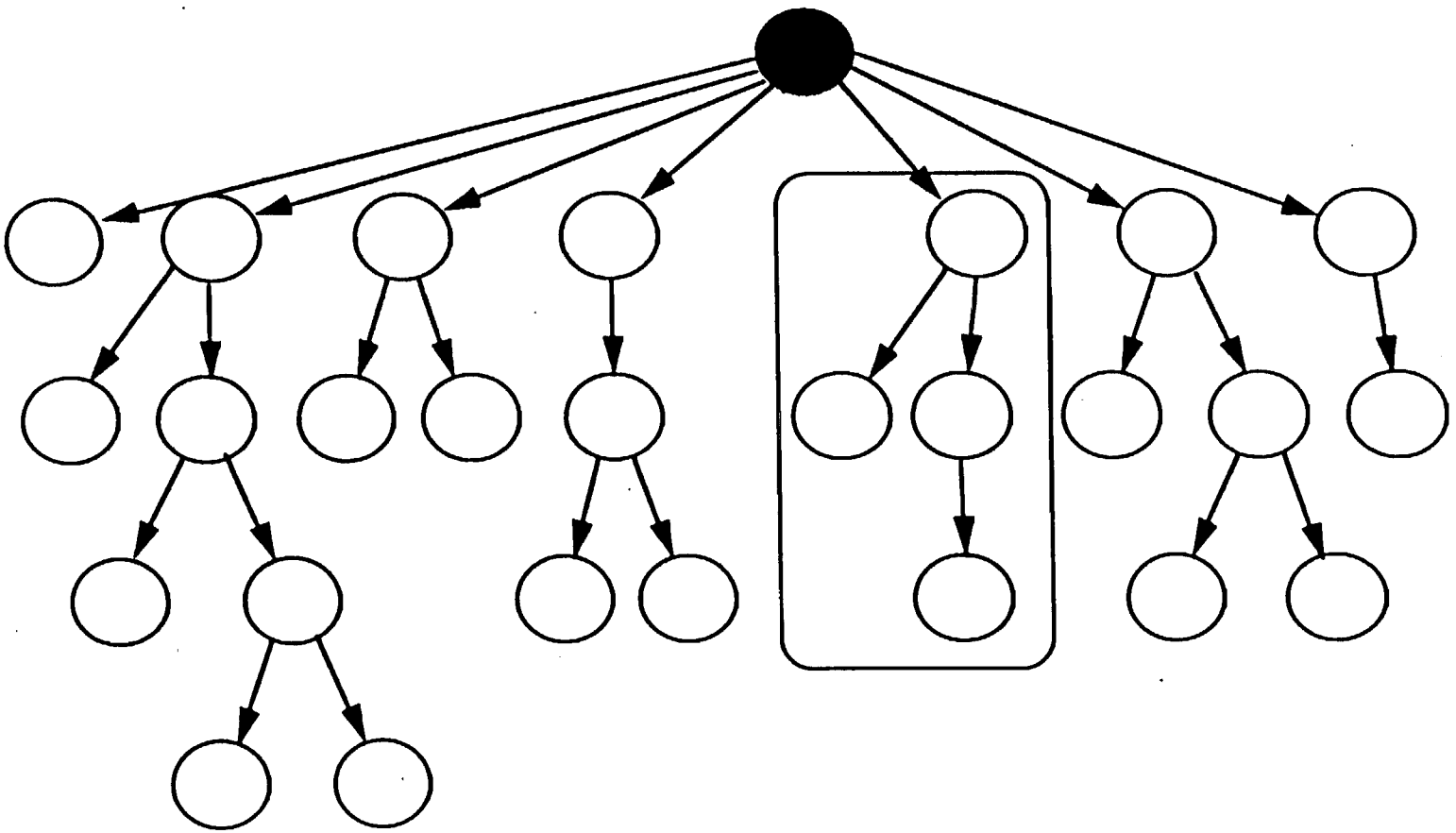
Static Structure Chart for Compilation Unit



Intermetrics

COMPILATION UNIT DURING SUBUNIT SELECTION

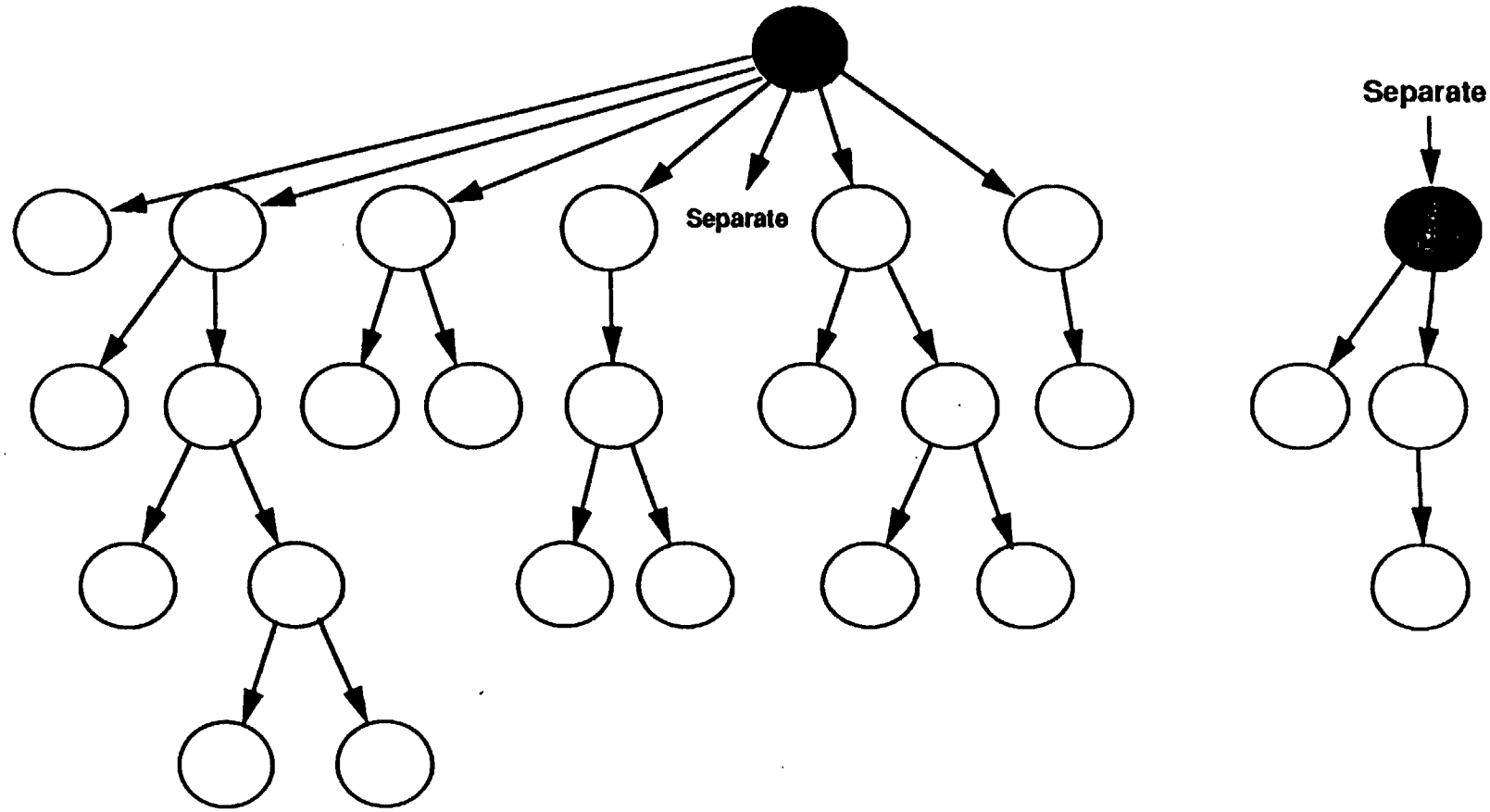
Static Structure Chart for Compilation Unit



Intermetrics

SEPARATE COMPILATION OF SUBUNIT

Static Structure Chart for Compilation Units

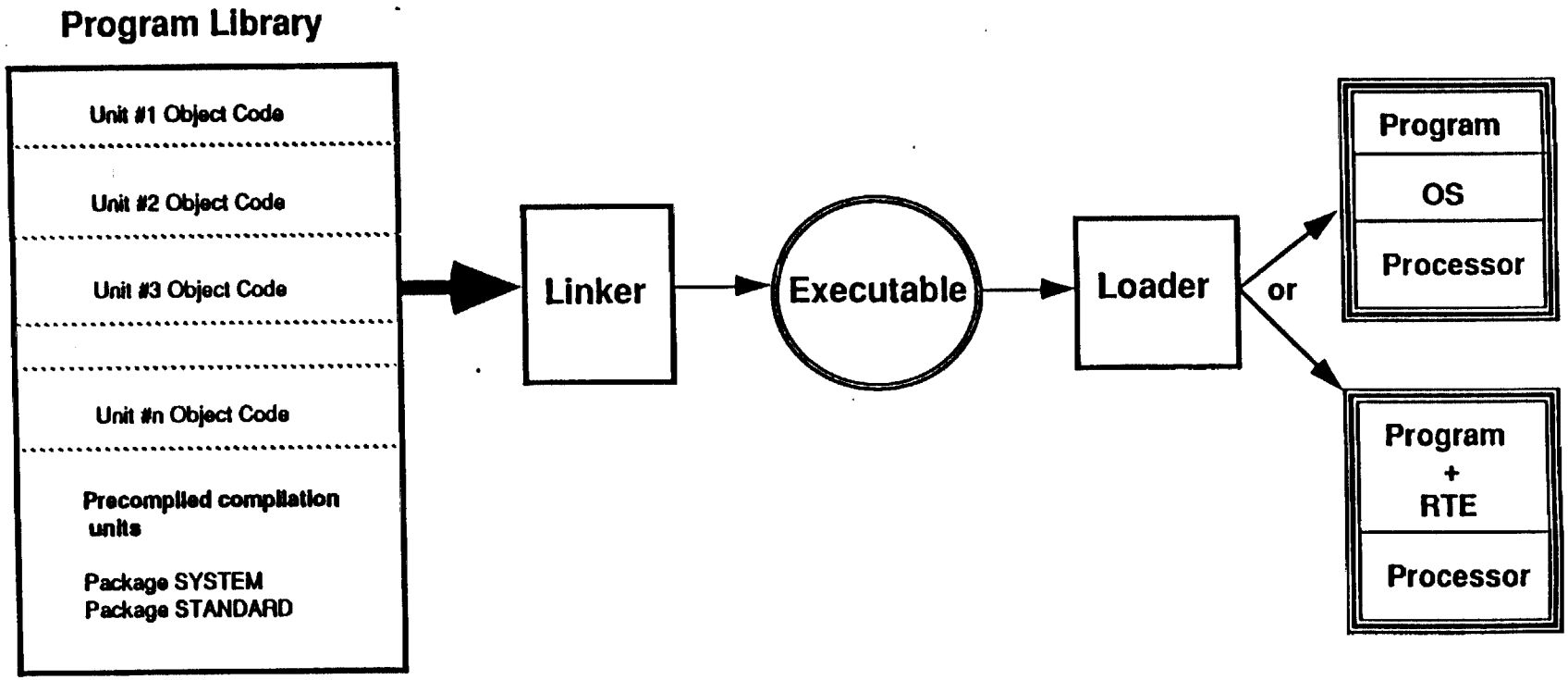


Intermetrics

COMPILATION UNIT DEPENDENCIES

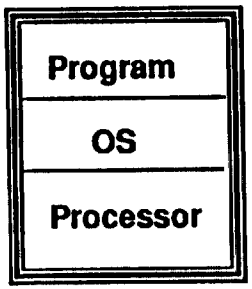
- **Within a program, the compilation units depend on each other in certain ways.**
  - -- Any unit that uses a package is said to ***depend*** on that package.
  - -- Any unit that uses a procedure is said to ***depend*** on that procedure.
  - -- The body of a subprogram, package, or task depends on the specification of that unit.
  - -- Subunits depend on the enclosing units from which they are separate.
- **Before any unit can be compiled, all units on which it depends, either directly or indirectly, must have already been compiled.**
- **In particular, all units must have been compiled with the latest versions present in the program library.**
  - -- This requirement ensures that if an interface is changed from one day to the next, then all modules that depend on that interface will have been verified to compile against the new version.
- **This requirement imposes a heavy testing challenge**
  - -- Test the units are heavily referenced by other units first.
  - -- Late changes to these units will result in expensive recompilations.

PUTTING IT ALL TOGETHER

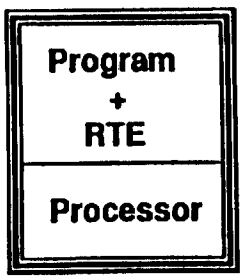


Intermetrics

TWO OPPOSITE TYPES OF ADA RUN-TIME ENVIRONMENT



- Ada programs are under control of an operating system **and its scheduler**
- The Ada programs are **independent** in that they do not know about each other in the Ada world.



- Only **one** (Ada) program is running, and it has control.
- The Ada program is a single load image, and knows about the whole world of the processor.

Intermetrics

OUTLINE

Introduction and Overview

Development and Integration of Ada systems, Part 1

→ **Real-time programming in Ada, Part 1**

Tasks and Communications

Relation of Ada to the Underlying OS / UNIX / POSIX

Design and Verification of Distributed Systems

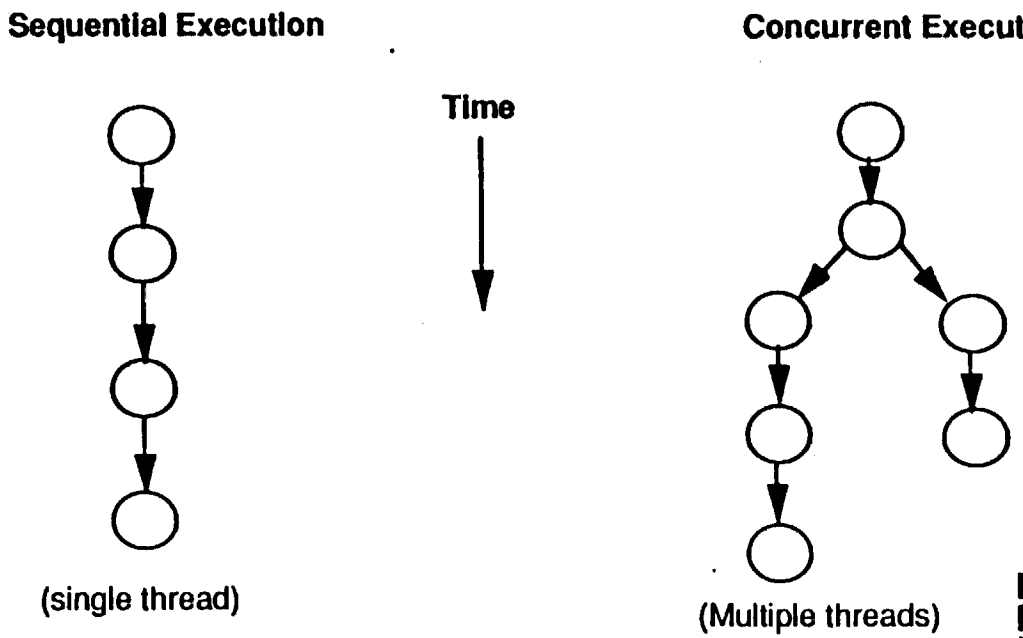
Conclusions, Part 1

REAL-TIME PROCESSING REQUIREMENTS

- **A real-time processing system must be able to interact with the demands of its environment.**
- **A system is real-time if time-of-response criteria are a part of the system requirements.**
  - -- and consequently must be a part of the system verification process.
  - -- The "real" of real-time is only relative to the environment
    - Not necessarily to WWV time.
- **Single-thread-of-activity processes can be supported by sequential Ada code.**
  - -- assuming that the performance is adequate
- **Multiple-threads-of-activity processes can be supported by concurrent Ada code**
  - -- *i.e.* Ada tasking
  - -- They can also be supported by sequential Ada code with a real-time operating system to perform the scheduling.

CONCURRENT COMPUTATION IN ADA

- In Ada, concurrent computation is supported by tasking.
- Without tasks, Ada programs execute sequentially.
- Within tasks, Ada programs execute sequentially.
- But, in an Ada program, several tasks can execute concurrently.



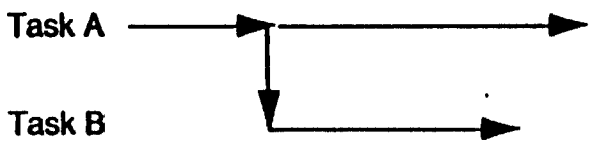
Intermetrics

TASKS

- **Tasks are <u>not</u> compilation units**
  - -- They are declared as part of a subprogram or a package.
  - -- Perhaps nested in a Block statement or in another Task.
- **Tasks are declared in two parts:**
  - -- **Task interface,** which provides
    - the **name** of the task and
    - a list of its **entries**
      - Tasks perform synchronization using these entries
  - -- **Task body,** which describes how the task is to be implemented.
- **A task body can be a subunit**
  - -- Hence it can be compiled separately
  - -- Supports modular development

TASK CAPABILITIES

- **Tasks can be activated**
  - -- A module can activate a task, causing it to execute concurrently.



- **Tasks communicate using**
  - -- Shared data
  - -- Entry call parameter lists
  - -- Communication with other intermediary tasks

WAIT STATEMENT

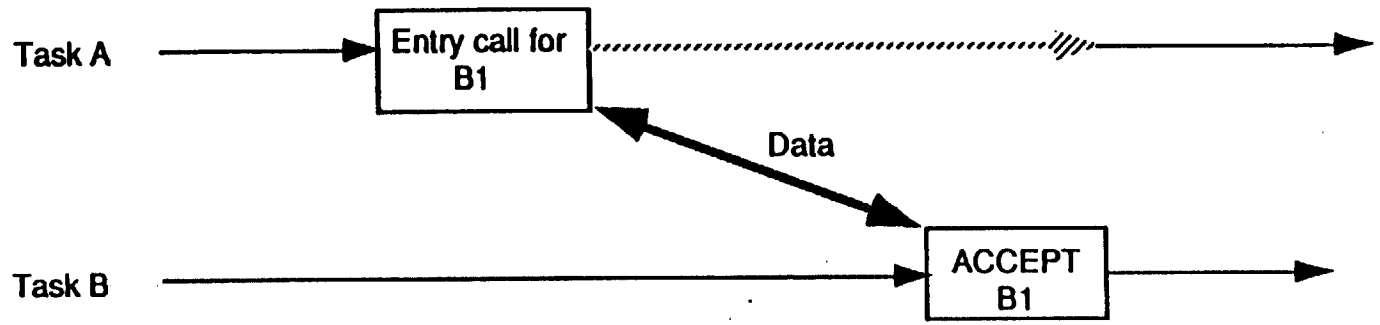
- **Ada supports the WAIT statement**
  - -- WAIT n
- **The WAIT statement suspends the execution of the task containing the statement for at least the specified amount of time.**
  - -- Ada does not guarantee how much longer the task will be suspended.
  - -- You cannot expect an *exact* amount of time delay
  - -- The WAIT statement can be used only for the module in which the statement occurs.
  - -- You cannot use the WAIT statement to affect the behavior of another task.

SIMPLE RENDEZVOUS

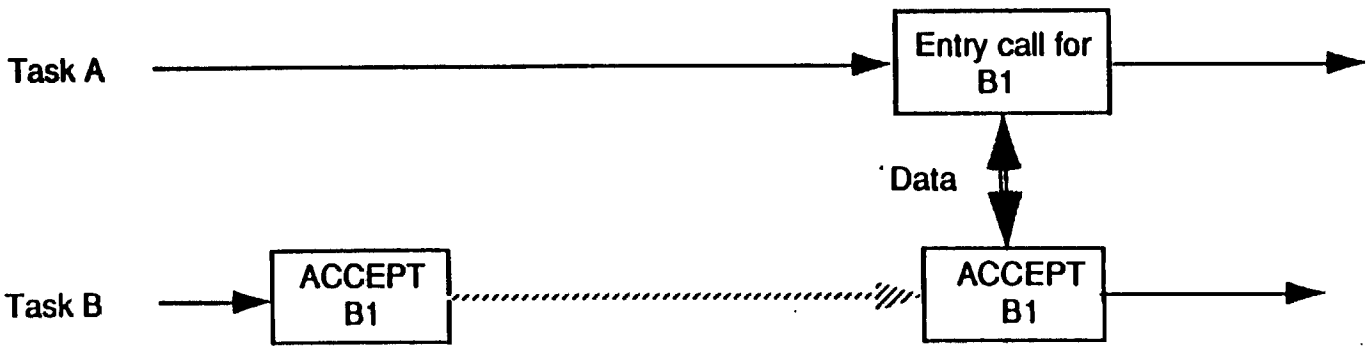
- **The Ada rendezvous statement supports inter-task synchronization.**
  - -- A rendezvous is requested by calling an entry name of the task with which the rendezvous is desired.
  - -- The calling task is then suspended until the called task **ACCEPTS** and completes the rendezvous request.
  - -- The entry call looks like a procedure call
    - It may contain a parameter list of information to be passed back and forth.
  - -- The called task completes the rendezvous with an ACCEPT statement.
    - For each task entry, there is at least one ACCEPT statement.
    - The task may also execute some additional code while the calling task is suspended.
    - *Critical region*
  - -- When the ACCEPT code is complete, both tasks resume concurrent execution.
  - -- If the called task reaches the ACCEPT before the calling task actually calls the entry, then it is suspended until the call occurs.

SIMPLE RENDEZVOUS (EXAMPLE)

Entry call occurs before ACCEPT statement



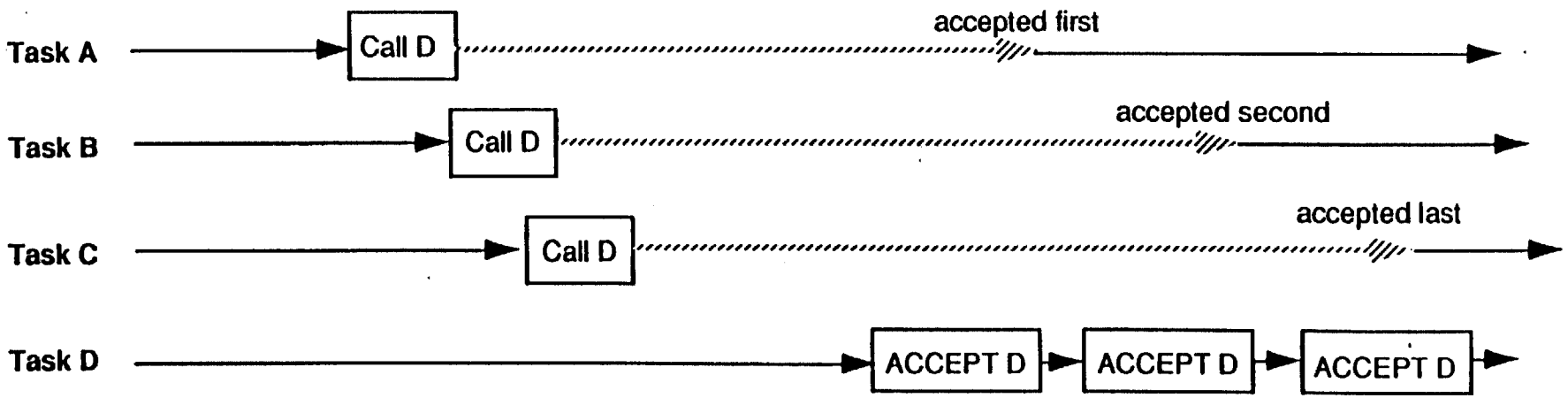
ACCEPT statement occurs before Entry call



Intermetrics

MULTIPLE ENTRY CALLS (FIFO QUEUE)

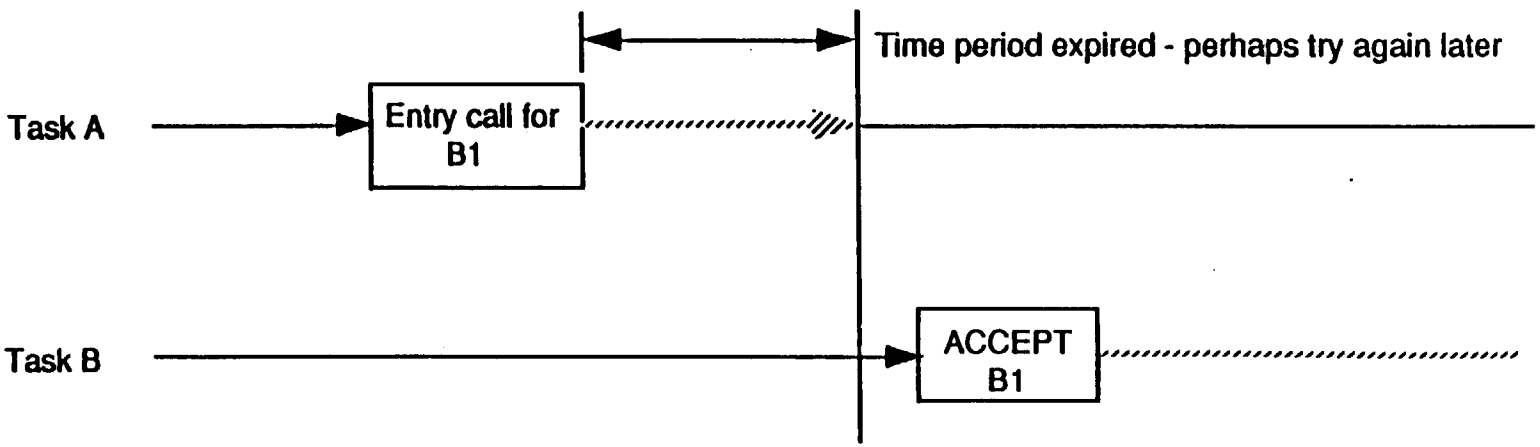
- **If more than one task call the same entry for an accepting task, these entry calls are placed into a queue, and serviced first-in / first-out.**



Intermetrics

TIMED ENTRY CALL

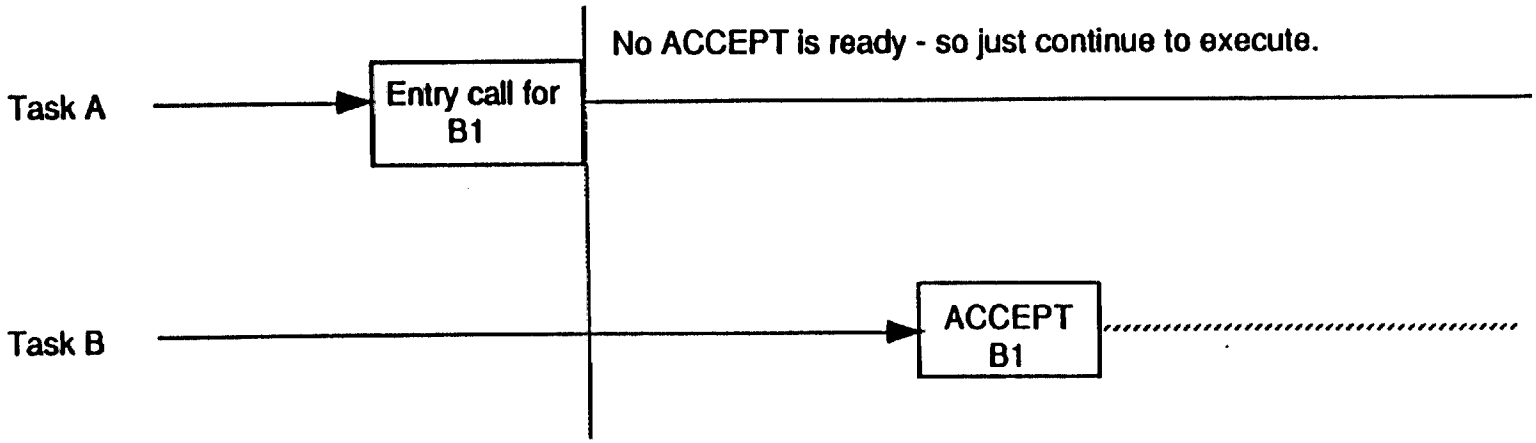
- **Sometimes, when calling a task entry, you just do not want to wait around until the call is accepted.**
- **Can use a *Timed Entry Call***
- **Calling task specifies how long it is willing to wait.**
  - -- If the entry call is not accepted before the time runs out, then the entry call is cancelled.
  - -- If the entry call is accepted in time, then the entry proceeds normally.



Intermetrics

CONDITIONAL ENTRY CALL

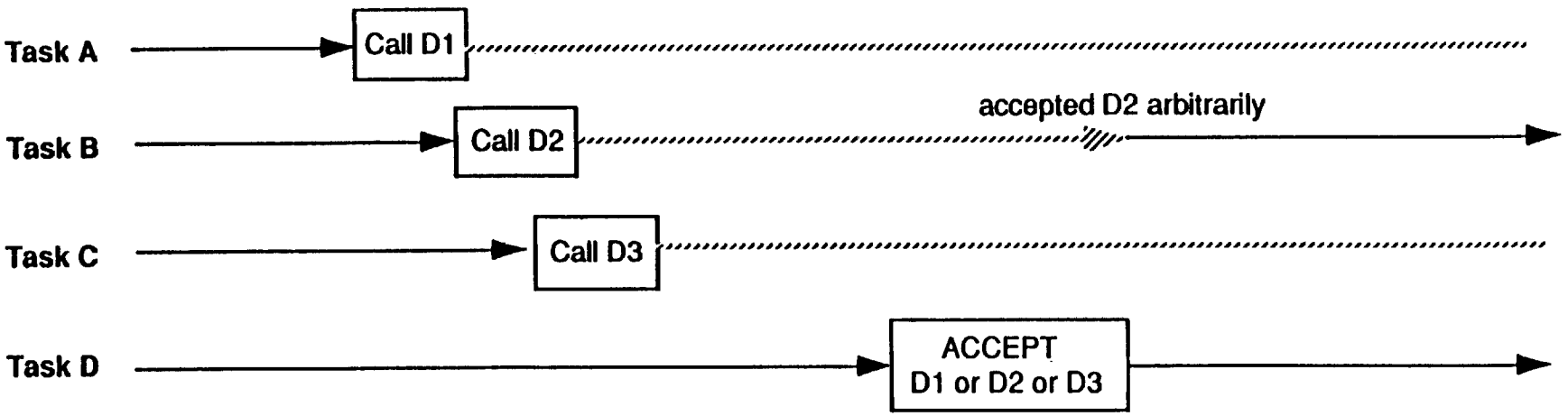
- **Sometimes, when calling a task entry, you just do not want to wait around until the call is accepted or for any time delay at all.**
- **Can use a *Conditional Entry Call***
- **Calling task specifies what to do if the entry call is not immediately accepted.**
  - -- If the entry call is not immediately accepted, the entry call is cancelled and a specific sequence of statements are executed.
  - -- If the entry call is accepted in time, then the entry proceeds normally.



Intermetrics

SELECTIVE WAIT STATEMENT

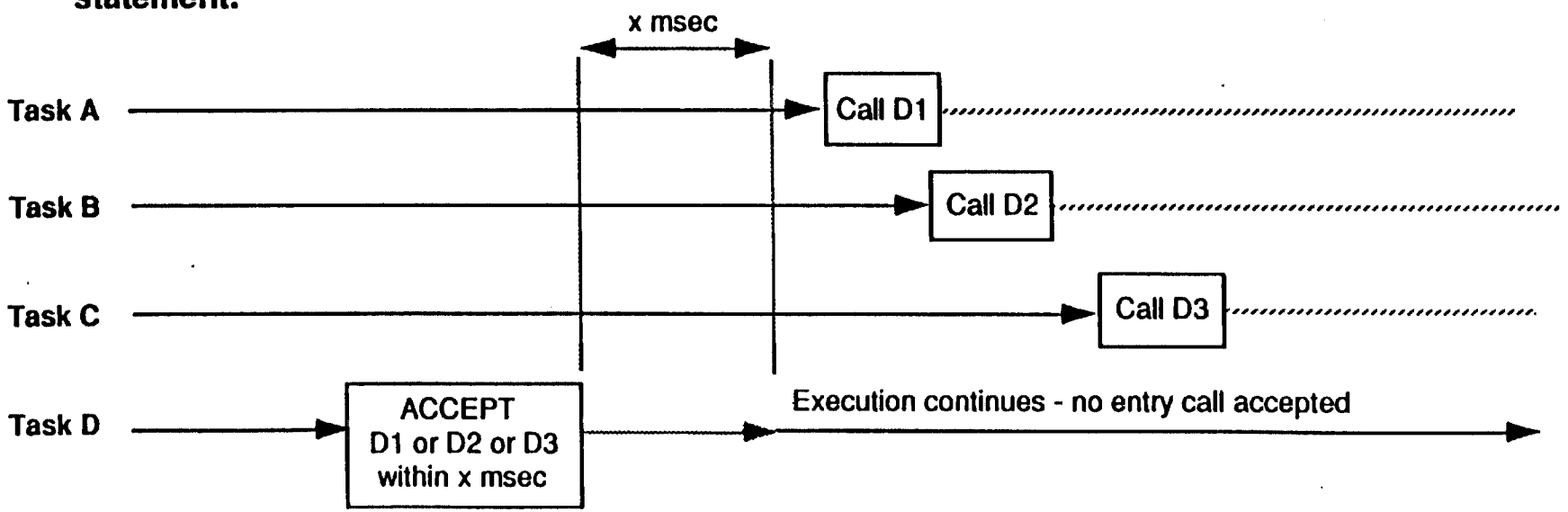
- **A Task may contain several different entries**
  - -- These entries may be called by different tasks.
- **The SELECTIVE WAIT statement allows a task to look at several entries at once.**
  - -- If none of the entries have been called, then the task waits.
  - -- If one or more have been called, then the task arbitrarily selects one for acceptance, and continues.
  - -- A maximum of one entry call can be accepted for each execution of the SELECT statement.



Intermetrics

SELECT-DELAY

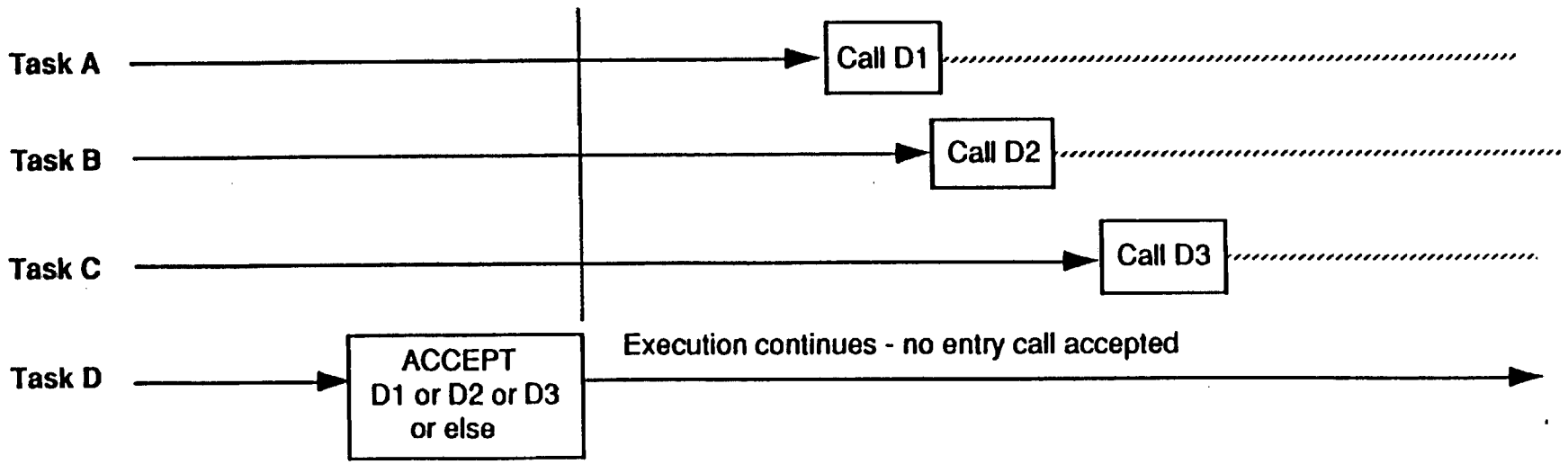
- The SELECT-DELAY statement also allows a task to look at several entries at once.
- However, if none of the entries have been called, then the task waits for a specified period of time.
- If that time period expires without any of the entries being called, then the ACCEPT statement terminates, and the task continues execution.
- If one or more have been called, then the task arbitrarily selects one for acceptance, and continues.
- A maximum of one entry call can be accepted for each execution of the SELECT statement.



Intermetrics

SELECT ELSE

- The SELECT-ELSE statement also allows a task to look at several entries at once.
- However, if none of the entries have been called, then the task terminates the ACCEPT statement immediately, and continues with its own execution.
- If one or more have been called, then the task arbitrarily selects one for acceptance, and continues.
- A maximum of one entry call can be accepted for each execution of the SELECT statement.



Intermetrics

TASK PRIORITIES

- Ada allows the optional assignment of a priority to a task.
- All such priorities are static - they cannot change at execution time.
- The Ada LRM says:

> *If two tasks with different priorities are both eligible for execution and could sensibly be executed using the same physical processors and the same other processing resources, then it cannot be the case that the task with the lower priority is executing while the task with the higher priority is not.*

- ACCEPT queues are not affected by priorities
- Entry SELECTions are not necessarily affected by priorities.
- Seen as a limitation of Ada

OUTLINE

Introduction and Overview

Development and Integration of Ada systems, Part 1

Real-time programming in Ada, Part 1

Tasks and Communications

Relation of Ada to the Underlying OS / UNIX / POSIX

Design and Verification of Distributed Systems

Conclusions, Part 1

COMMUNICATIONS

- The General Problem -- Given the Facilities of the SSF DMS, What Other Facilities are Needed?
- A Proposed Solution -- the RODB
- Definition of RODB Undefined for Now
- The Following Concentrates on Alternatives
- Alternative Selected Has Serious Implications for Verification
- Some Alternatives Require More Visibility Into Individual Work Package Design and Implementation

Intermetrics

DATA BASE OR MESSAGE SYSTEM

- **The Primary Decision -- Is the RODB a True Distributed Data Base or a Message Switch?**
- **Message Switch -- Simpler to Design and Implement**
  - -- Allows Messages to be Sent by Name or Location
  - -- May Be Simple or Complex In Its Own Right
  - -- Limitation -- Always Deals with Applications and Data as Separate Entities
- **Distributed Data Base -- More Complex, More Capability**
  - -- May Impose Constraints on Multiple Data Items
  - -- Allows Locking and Unlocking of Records
  - -- May Involve Unsolved Problems
  - -- Difficult to Implement in General
- **Whether in the RODB or Not, "Data Base Type" Problems are Difficult to Verify**

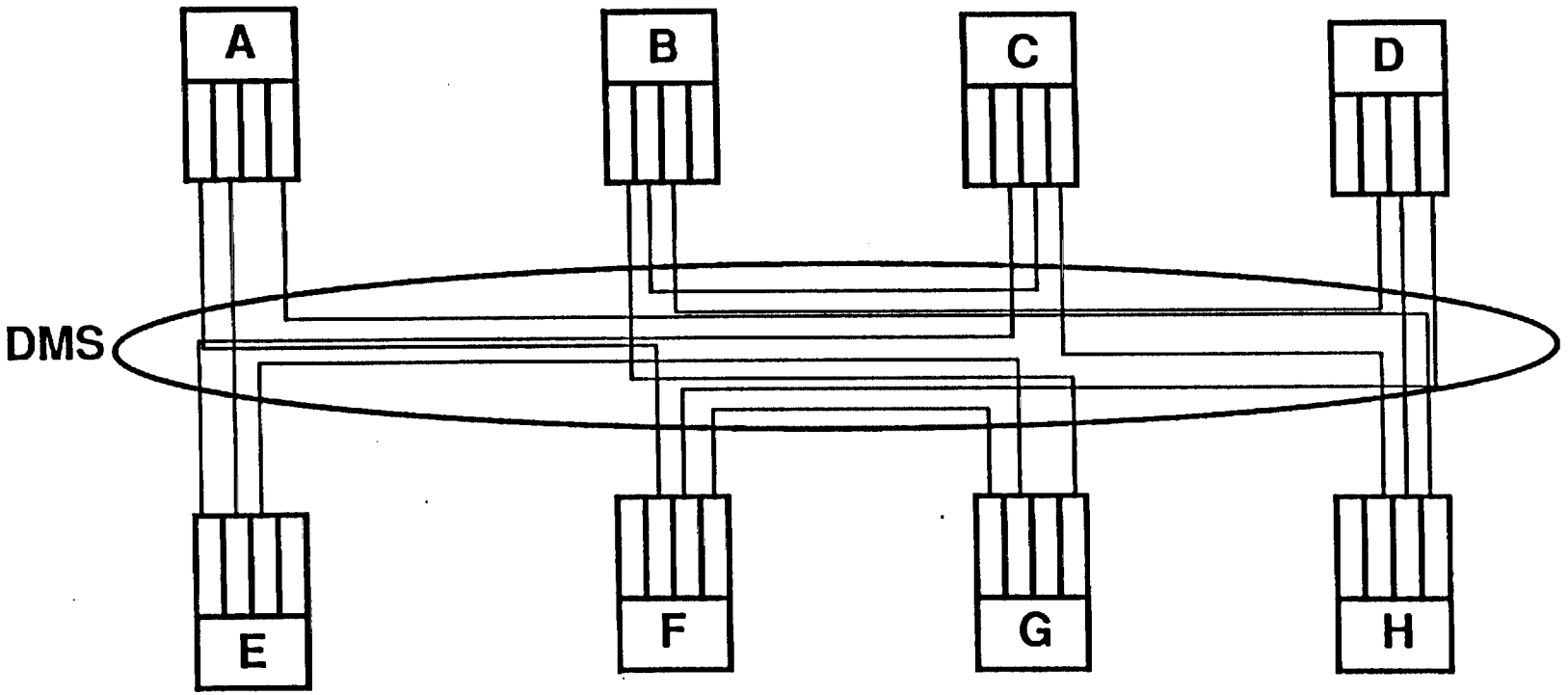
DMS CAPABILITIES

- **Outside of RODB Definition -- Provided by DMS**
- **Part of OSI Standard Definitions**
- **Still Requires Some Attention**
  - -- OSI Does Not Specify Everything
  - -- Fidelity of Communication -- Packet Loss, Duplication, and Re-order
  - -- May Affect RODB Design -- Ack/Nack Protocols, etc.

INDIVIDUAL COMMUNICATIONS

- **Simplest Solution (From RODB Viewpoint) -- Individual Communications to Actual Network Addresses**
- **RODB Effectively Null**
- **Any Change in Application Location Requires Changes to All Communicating Applications -- Re-Location is Complex**
- **Verification Consists of Verifying Each Combination of Communicating Applications**
  - -- Verification Points Explicitly Located by Communication Path
  - -- Overall DMS Verification Consists of Assuring Sufficient Message Capacity
  - -- Only Actual Message Traffic Imposed on Bus
  - -- Inter-Work Package Dependencies May Still Exist as Implicit Constraints Between Applications

DIRECT COMMUNICATION BETWEEN APPLICATIONS

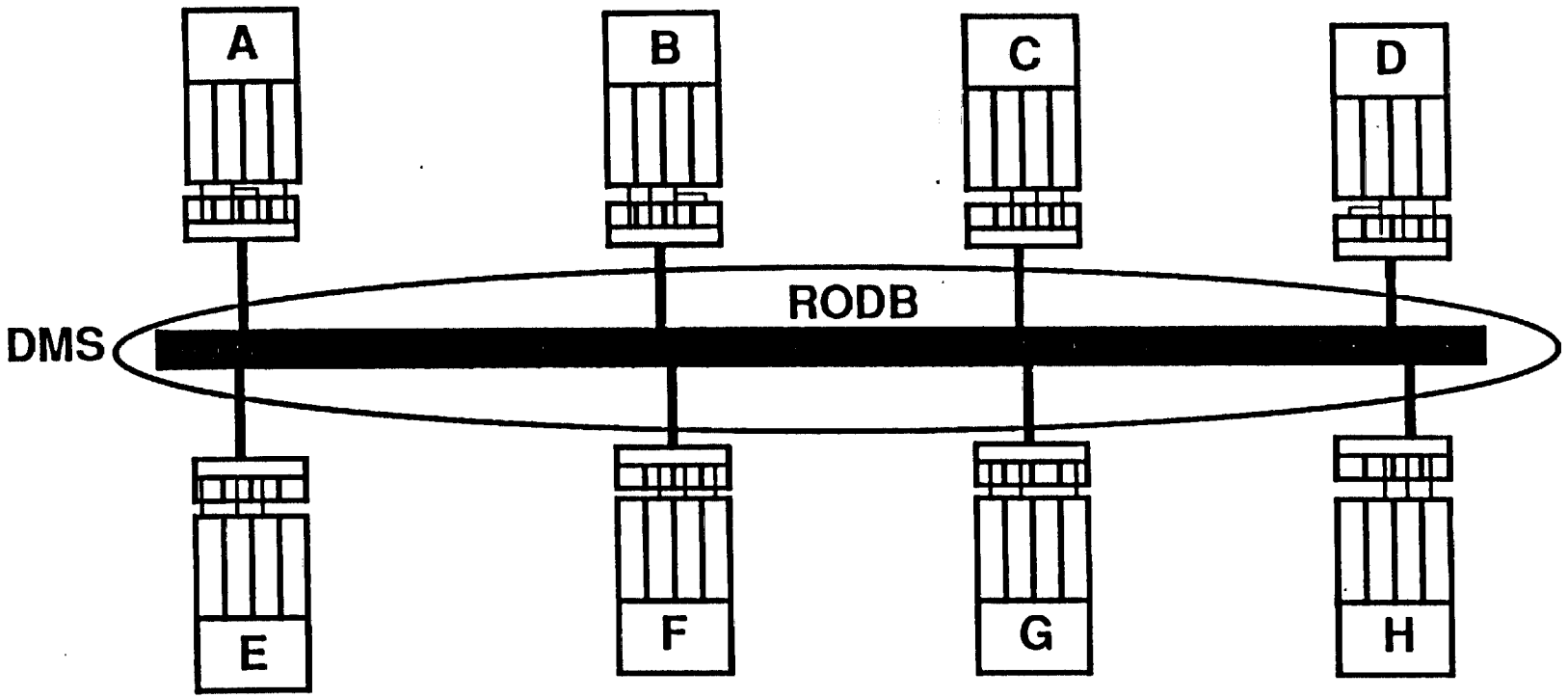


Intermetrics

MAILBOX COMMUNICATIONS

- **The RODB May Be Viewed as a Communications Package**
- **Mailbox Communications -- Applications Communicate by Local Named Mailboxes**
- **May or May Not Involve Queueing of Multiple Messages between the Same Applications**
- **Easier for Application Programming -- Communications are By Name**
- **Re-Location of Application Involves Changing Mailbox Locations (Tables)**
  - -- May Involve Re-Building All Tables
  - -- Alternative -- Master Table on MSU, Local Tables on Each Processor

MAILBOX COMMUNICATION BETWEEN APPLICATIONS



Intermetrics

STATIC TRANSLATION

- **Simplest Mailbox Solution -- Mailbox Consists of Local Tables**
- **RODB Consists of Local Table Access**
- **Any Change in Application Location Requires Rebuilding Tables of All Affected Processors**
- **Verification Consists of Verifying Tables and Each Combination of Communicating Applications**
  - -- Verification Points Identified by Calls to Mailbox System by Name
  - -- Mailbox Table Translation Must Be Verified
  - -- Only Actual Message Traffic Imposed on Bus
  - -- Inter-Work Package Dependencies May Still Exist as Implicit Constraints Between Applications

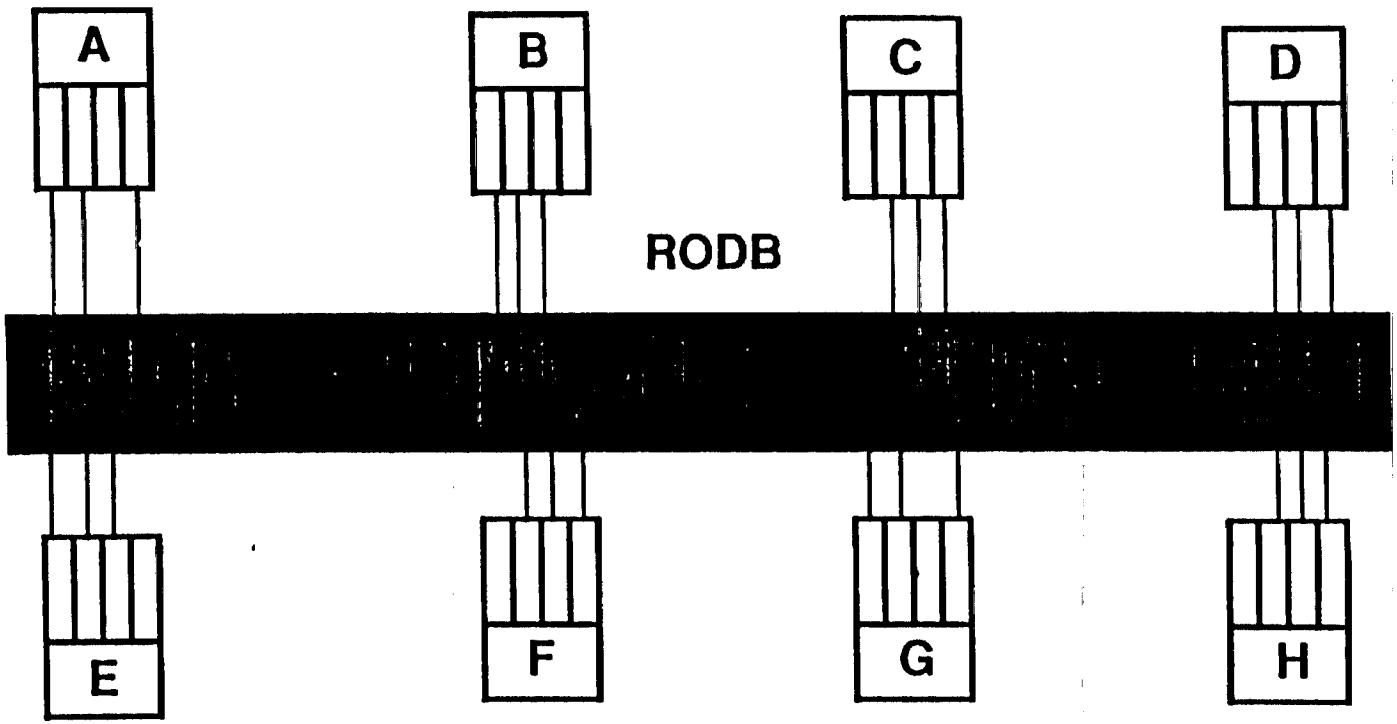
DYNAMIC TRANSLATION

- **Most Flexible Mailbox Solution -- All Communication by Name**
- **RODB Must Maintain Global and Local Mailbox**
  - -- Master List Kept on MSU
  - -- First Message Requires Request to MSU for Mailbox Translation
  - -- Subsequent Messages Use Local Translation
- **Application Location Change Requires Changing Master Table in MSU**
  - -- Affected Application Changes Own Location
  - -- Local Locations Contain Identifier -- Changing Location Changes Identifier, Requiring Translation Request to MSU
- **Verification Now Requires Extensive Testing of Complex Mailbox System**
  - -- Individual Exchanges Still Subject to Verification
  - -- Bus Now Subject to Extra Traffic for Translation Requests

DISTRIBUTED DATA BASE

- The RODB May Be Viewed as a Distributed Data Base
- No Communication Per Se -- Data is Read and Written as to a Data Base Management System
- Extremely Flexible -- A Different Way of Thinking
- All Station Data Contained in Data Base -- Access to MSU or to Sensors Looks the Same
- Re-Location of Application Now Trivial and Contained in Data Base
- Distributed Data Bases Still a Research Topic -- Potential Unsolved Problems
- Design, Implementation, and Verification Extremely Complex

MAILBOX COMMUNICATION BETWEEN APPLICATIONS



Intermetrics

DANGEROUS FEATURES OF DISTRIBUTED DATA BASES

- **Record Locking**
  - -- Any Application Needing to Lock Multiple Records
  - -- Danger of Deadlock
  - -- Base of Other Problems
- **Transaction Read -- Reading Many Records at Once**
- **Transaction Write -- Writing Many Records at Once**
- **Generalized Form -- Maintaining Distributed Constraints**
  - -- Powerful Way of Expressing Safety Constraints
  - -- General Problem Unsolved Without Extensive Locking that Eliminates Advantages of Distributed Data Base
- **Implementing Data Base Features by Message Passing Still Leads to Extensive Verification Obligations**

OUTLINE

Introduction and Overview

Development and Integration of Ada systems, Part 1

Real-time programming in Ada, Part 1

Tasks and Communications

→ **Relation of Ada to the Underlying OS / UNIX / POSIX**

Design and Verification of Distributed Systems

Conclusions, Part 1

Intermetrics

ADA AND OTHER OPERATING SYSTEMS

- **The Ada Tasking Semantics Assumes Control Over and Visibility Over the Entire "World"**
  - -- Definition of "World" is Situation Dependant
  - -- "World" Comprises Limit of Ada Semantic Checking
- **The Operating System Also Assumes It Controls the "World"**
  - -- Definition of "World" Still Situation Dependant, but Different
  - -- Definition Also Somewhat Elastic -- Processor versus Network
- **The Resolution of this Conflict Defines the Two Different "Worlds"**
  - -- Grant the Operating System its "World"
  - -- The Ada "World" Limited to a Single Operating System "Process"
- **This Assignment Requires Grouping Ada Tasks into Main Programs as Part of the Design Process**

MAINTAINING FLEXIBILITY

- **Grouping Tasks into Main Programs Limits Flexibility and Semantic Checking**
  - -- Grouping is Often Arbitrary, Especially if Object Oriented Design Techniques are Used
  - -- Semantic Checking Stops at Process Boundary
- **Operating System Bottlenecks and Ada Task Bottlenecks May Be Different**
  - -- What Happens to the Process When a Task Blocks on I/O?
  - -- Process Time Slicing May Interrupt Tasks with Useful Work
- **To Maintain Flexibility and Eliminate Bottlenecks, Re-Allocation of Ada Tasks to OS Processes Must Not Require Re-Coding**

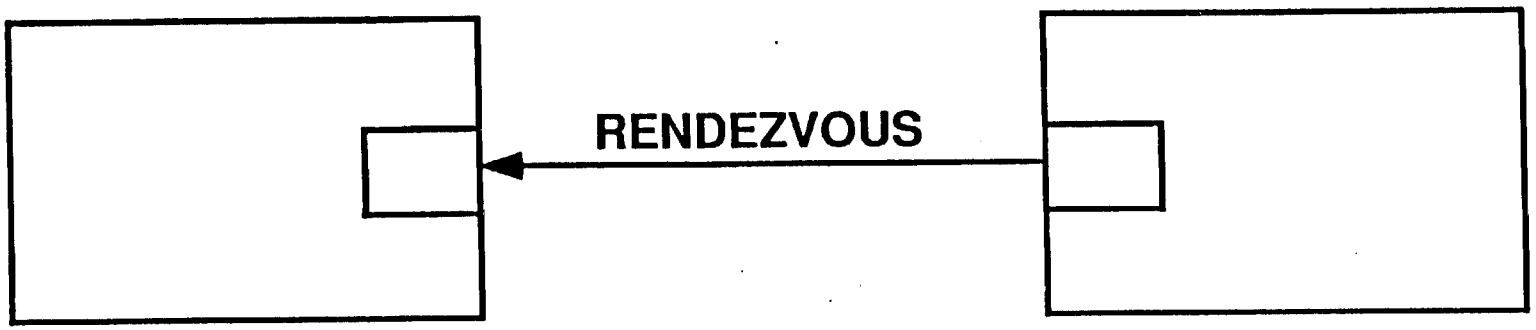
ADA AND I/O

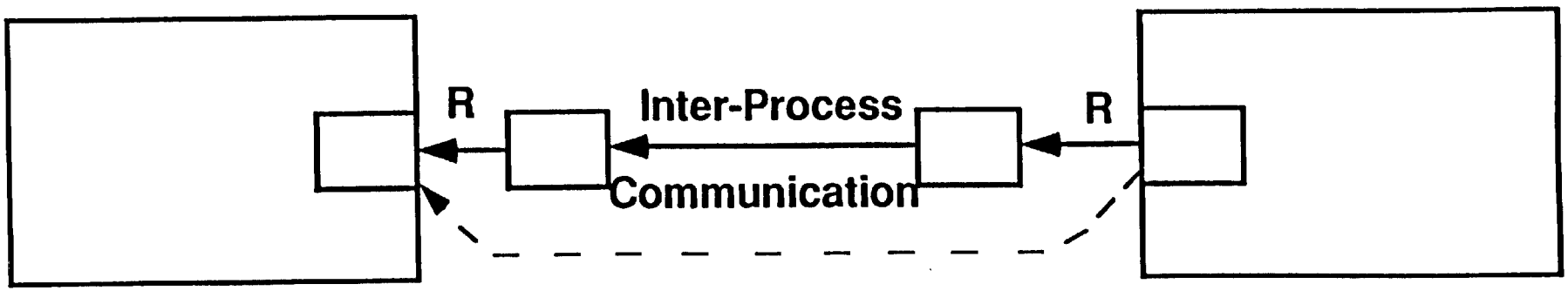
- Is Input/Output Performed Using Ada Standard Procedures or by Operating System Calls?
- In Either Case, What Blocks (Task or Process) When I/O Occurs?
- Different Compilers May Produce Different Implementations -- I/O Should Be Encapsulated in Special Tasks or Subprograms
- If Process Blocks, Flexibility in Task Grouping is Essential

SEPARATING TASKS AND PROCESSES

- Task Rendezvous and Inter-Process Communication Should Look as Similar as Possible
- For Each Rendezvous, a "Phantom Task" May Be Defined
- The "Phantom Task" Disguises the Operating System Communications from the Ada Tasks
- Distributing Tasks across Different Processes Becomes the Insertion of "Phantom Tasks"
- Given Appropriate Naming Conventions, No Recompilation is Necessary
- Assuming Identical Types and Parameter Names, Semantic Checks Preserved to Some Extent

PHANTOM TASKS



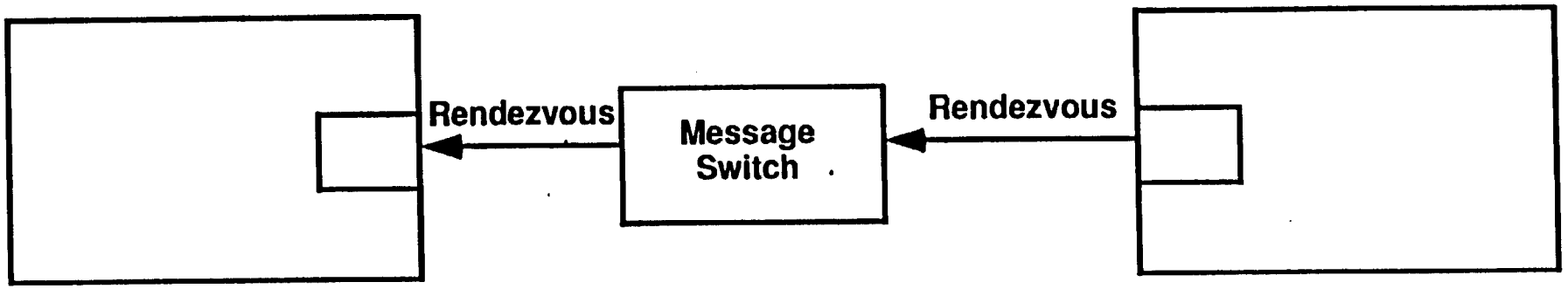


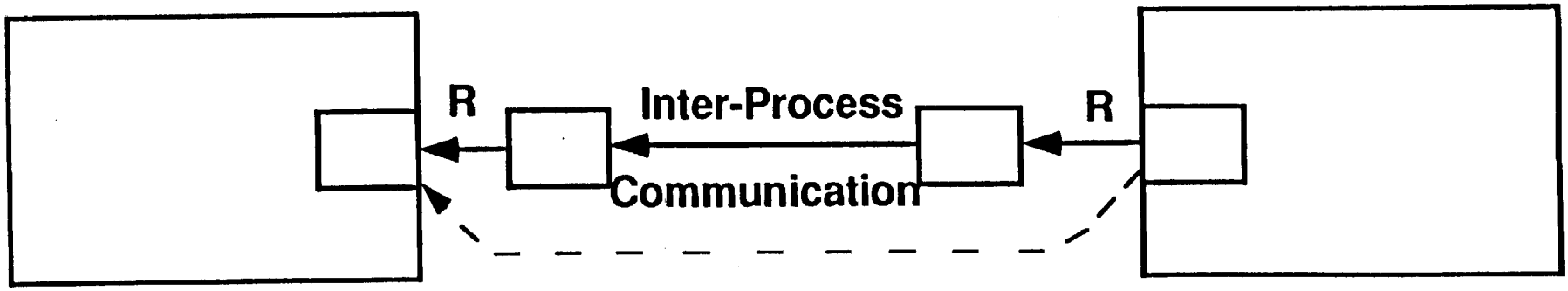
Intermetrics

DECOUPLING TASKS

- **This Solution Does Not Preserve Ada Tasking Semantics**
  - -- In Ada, Tasks Execute Rendezvous Together
  - -- Operating System May Queue Messages Rather Than Synchronizing Processes
- **Semantics May Be Preserved by "Decoupling Tasks"**
- **Requires an "Intermediate Task" Between Cooperating Tasks**
- **Has Other Benefits**
  - -- Looser Synchronization
  - -- Message Queues
  - -- All Other Benefits of Loose Message Coupling
- **Can Be Eliminated by Careful Validation**
- **Ada Semantics Can Be Re-Produced by Acknowledgement Protocol**
- **Implies a Design Decision**

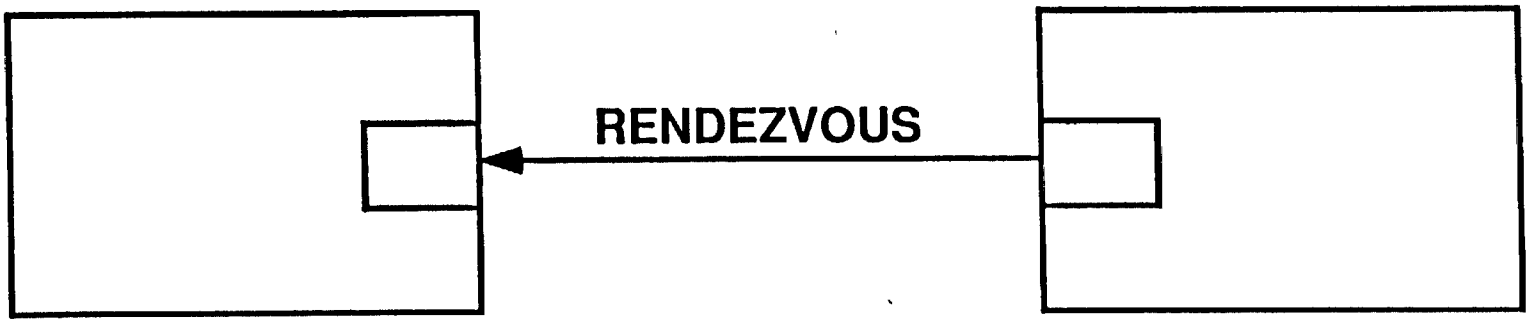
DECOUPLED TASKS

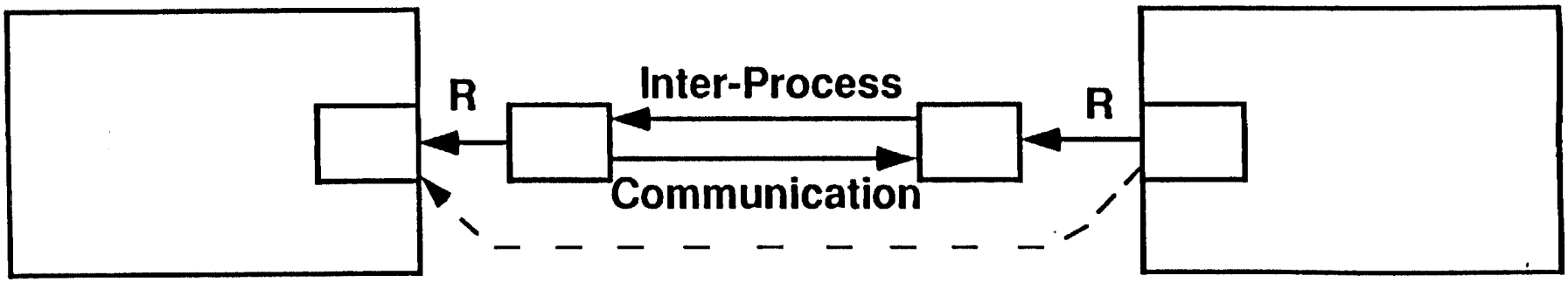




Intermetrics

COMMUNICATION ACKNOWLEDGEMENT





Intermetrics

ADA AND UNIX/POSIX

- **UNIX Presents some Special Problems**
  - -- Real Time Concerns Exist for both Ada and Unix
  - -- Recommended Solutions are Different for Both Systems
- **Present Unix Implementations Block Process on All I/O**
- **POSIX Committees are Studying These Problems**
  - -- 1003.3 Studying Real Time Issues
  - -- 1003.5 Studying POSIX and Ada
- **Charles Draper Laboratory Study Foresees Problems**
- **Question - How to Combine Benefits of Ada with POSIX?**

ADA AS A DESIGN LANGUAGE

- **Ada May Be Used as Language of Choice, with Controlled Use of C Where Necessary**
- **Rules:**
  - -- All Software Will Be Written and Tested in Ada
  - -- Selected Routines May Be Re-Written in C
  - -- When C Language Changes, Ada Must Change As Well
  - -- Requires Compiler That Can Co-Exist With C Language
  - -- Implied Decision: Use POSIX or DMS Real Time Features
- **Creates Significant Configuration Management Problem**

Intermetrics

ADA 9X

- Ada Language Standard is Under Revision
- For Thirty Year Program, Later Software May Be Written in New Language
- Committee Will Consider Compatibility
- Most Suggested Ada Changes in Area of Task Semantics
  -- ARTEWG Report
- Flexible Task Coupling Becomes Even More Important

Intermetrics

OUTLINE

Introduction and Overview

Development and Integration of Ada systems, Part 1

Real-time programming in Ada, Part 1

Tasks and Communications

Relation of Ada to the Underlying OS / UNIX / POSIX

→ **Design and Verification of Distributed Systems**

Conclusions, Part 1

VERIFICATION OF DISTRIBUTED SYSTEMS

- Component Applications of Distributed Systems Must Be Verified as in Normal Applications
- Communications Protocols and Cooperations Adds Complexity to Verification Task
- Distributed Control Creates Danger of Deadlock, Bottlenecks, Inconsistent Data, etc.
- Simplicity of Design and Inter-process Communication Eases the Design, Implementation, and Verification Task

DELAY INSENSITIVITY

- **A Network of Processes whose Correctness Does Not Depend Upon Relative Execution Times or Communication Delays is Called "Delay Insensitive"**

* **Verification of Delay Insensitive Process Networks Is Limited to Verifying the Correctness of Each Process**
  - -- Overall Efficiency Constraints, such as Network Speed, Must Also Be Verified

- **Delay Insensitive Networks Can Be Implemented Using a Message Passing System (such as Mailboxes)**

Intermetrics

MUTUAL EXCLUSION

- **Mutual Exclusion Provides Safe Access to Shared Resources Over Time**
- **Mutually Exclusive Access to Shared Resources May Be Inefficient**
- **Algorithms for Implementing Mutual Exclusion Must Be Verified**
- **Mutually Exclusive Access to More Than One Resource Can Result in Deadlock**
- **Infinite Waiting Is Also a Possibility**
- **Rules to Avoid Mutual Exclusion:**
  - -- Each Piece of Data is Written By a Single Application
  - -- Writing is an Atomic Action
  - -- Avoid Consistency Constraints Between Data

COMPLICATIONS IN DISTRIBUTED SYSTEMS

- **Delay Sensitivity -- One Process Happening "Faster" or Needing to "Keep Up"**
  - -- Limited Sense of the Words -- Rough Relative Speeds are Not Excluded By This
  - -- Can Be Eliminated by Ack/Nack Protocols
- **Synchronous Execution -- Applications in Lock Step**
- **Time Sensitive Data**
  - -- Limited Lifetime
  - -- Data Samples Time Related
- **Multiple Writers of Data**
- **Consistency Constraints Across Data Items**

ELIMINATING DELAY SENSITIVITY

- **"Latch" Data Together -- Present Data As a Unit**
  - -- May Require Multiple Reads or Writes
- **Acknowledge Data Transmission -- Do Not Transmit New Data Before Acknowledgement**
  - -- Implies a Form of Synchronous Execution
- **Design Algorithms to Use "Latest and Greatest"**
  - -- "Do the Best You Can"
  - -- Moves Concern to Overall Response Time
  - -- Makes Acceptance Criteria Somewhat More Ambiguous
- **Message Queueing -- Can Eliminate "Bulges"**
- **Stochastic, "Read Then Write" Applications**

ELIMINATING SYNCHRONOUS EXECUTION

- Message Queueing to Avoid "Bulges"
- Global Clock
- Smaller, "Stochastic" Applications
- Some Situations Inherently Synchronous -- Isolate These

TIME SENSITIVE DATA

- **Locate Time Related Data Samples in Same Processor**
- **Establish Special Data Link Outside DMS and RODB between Time Related Samplers**
- **Use Data Sampling Rate Much Slower than System Response**
  - -- Should Be At Least an Order of Magnitude
  - -- Risky -- May Fail under Heavy Loads

MULTIPLE WRITERS OF DATA

- **Separate One Data Item Into Many**
- **Establish Single "Collator" That Reads from Multiple Sources and Merges Data**
- **Duplicate Data Item**
  - -- Implies Decision Concerning Which Copy to Read

CONSISTENCY CONSTRAINTS

- More General Form of Previous Complications
- May Be Helped by Duplicating Data
- Constraints Exist -- Should They Be Explicit?
- Small Number of Separate Constraints May Be Implemented by Separate Application
- Generalization of These Cases Is a Complex Engineering Decision
- Flexibility and Clarity of Expression May Justify Some Consistency Checking

VISIBILITY NECESSARY FOR VERIFICATION

- **Delay Insensitive, Non-Mutually-Exclusive Networks Require Very Little Visibility into Work Package Designs**
- **Seemingly Innocuous Individual Decisions May Affect Other Work Packages**
  -- Time Relationships, Multiple Data Item Reads, etc
- **Guidelines Should Be Established and Clearly Documented**
- **Decisions Involving Delay Sensitivity, Mutual Exclusion, and Communications Should Be Exposed and Reviewed Even If Contained Within One Work Package**

Final Report under University of Houston Research Activity No. SE-28

Distributed Systems and Ada
Part 2

June 30, 1989

Presenters: Dr. Bruce Burton
Dr. William Bail
Mr. David Barton
Intermetrics, Inc.

Note: NASA Cooperative Agreement NCC9-16

Intermetrics

OUTLINE

Real-time programming in Ada, Part 2

Development, integration , and verification of Ada systems, Part 2

Fault tolerance and Ada

Programming standards, techniques, and tools for Ada

Conclusions, Part 2

OUTLINE

- **Real-time programming in Ada, Part 2**
- **Development, integration , and verification of Ada systems, Part 2**
- **Fault tolerance and Ada**
- **Programming standards, techniques, and tools for Ada**
- **Conclusions, Part 2**

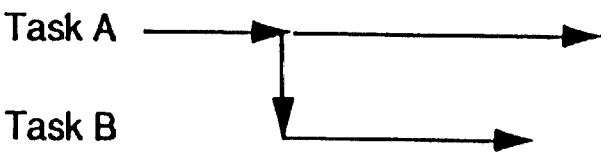
OVERVIEW OF ADA REAL-TIME FEATURES

- Two parts will be presented
- Part 1 contains a detailed summary of Ada tasking features
  - -- Illustrates capabilities with actual Ada code segments
  - -- Provides overview of the tools that Ada provides to create real-time programs
  - -- In particular, reveals potential limitations.
- Part 2 contains a discussion of these Ada tasking features
  - -- Problems in Ada tasking model
  - -- Efficiency of Ada implementations

PART 1
DETAILED SUMMARY OF ADA TASKING FEATURES

TASK ACTIVATION

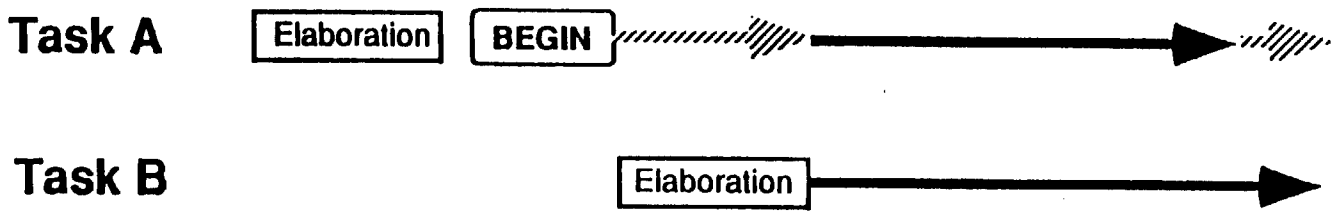
- **There are several ways to activate tasks**
  - -- to achieve concurrent execution.



Note: In the following diagrams,  denotes a waiting state for a task, while denotes an execution state.

Intermetrics

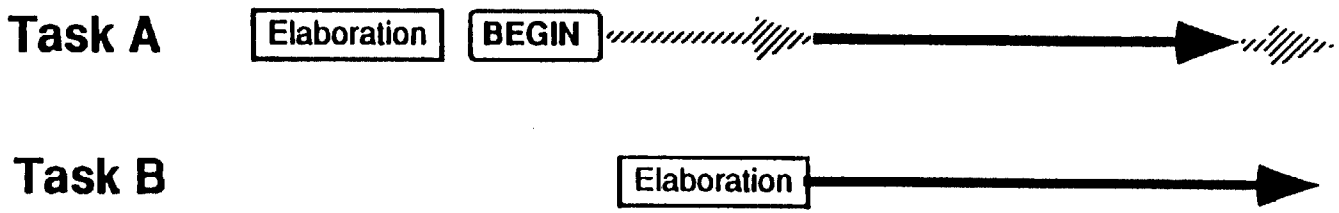
SIMPLE TASK ACTIVATION




```
PROCEDURE A IS
      TASK B;                          -- Task B specification
      TASK BODY B IS                   -- Task B Body
      BEGIN
            -- Do something
      END B;
BEGIN                                  -- Task B is elaborated here
      -- Do something                  -- After which A & B start to execute
END A;                                 -- A cannot terminate until B has
```

Intermetrics

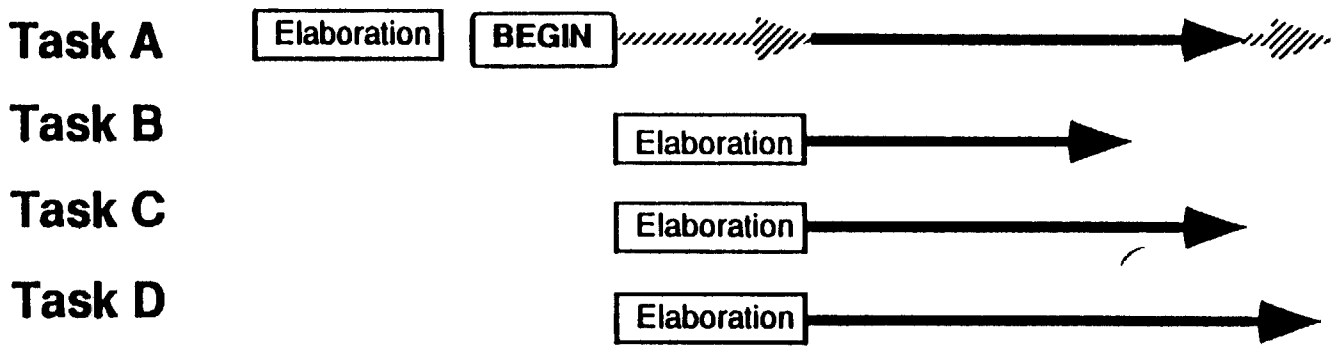
TASK TYPES WITH SINGLE TASK




```
PROCEDURE A IS
      TASK TYPE Sample_Task_Type;          -- Task Type specification
      TASK B: Sample_Task_Type;            -- Task B declaration
      TASK BODY Sample_Task_Type IS        -- Task Type Sample_Task_Type Body
      BEGIN
            -- Do something
      END B;
BEGIN                                      -- Task B is elaborated here
      -- Do something                      -- After which A & B start to execute
END A;                                     -- A cannot terminate until B has
```

Intermetrics

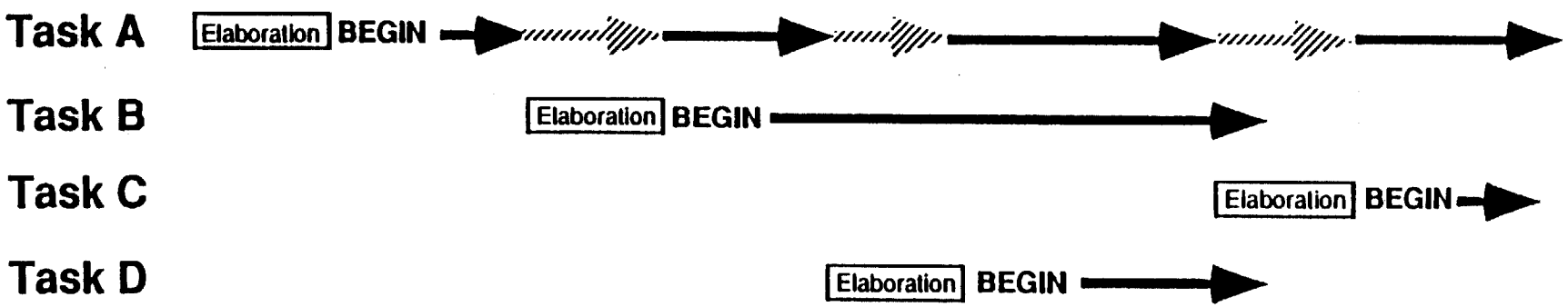
TASK TYPES WITH MULTIPLE TASKS




```
PROCEDURE A IS
      TASK TYPE Sample_Task_Type;          -- Task Type specification
      TASK B, C, D: Sample_Task_Type;      -- Declaration for tasks B, C, D
      TASK BODY Sample_Task_Type IS        -- Task Type Sample_Task_Type Body
      BEGIN
            -- Do something
      END B;
BEGIN                                      -- Tasks B, C, and D are elaborated here
      -- Do something                      -- After which A, B , C, and D start to execute
END A;                                     -- A cannot terminate until B, C, and D have
```

Intermetrics

ALLOCATED TASKS



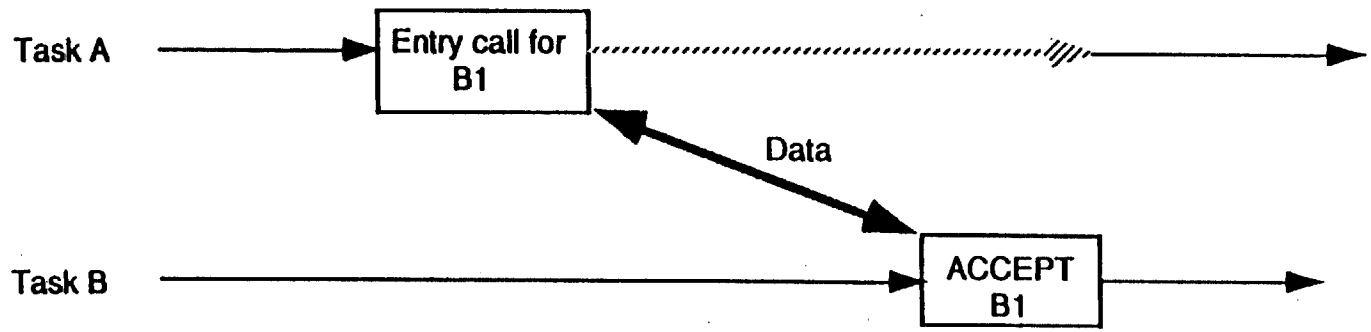
Intermetrics

ALLOCATED TASKS CODE SAMPLE

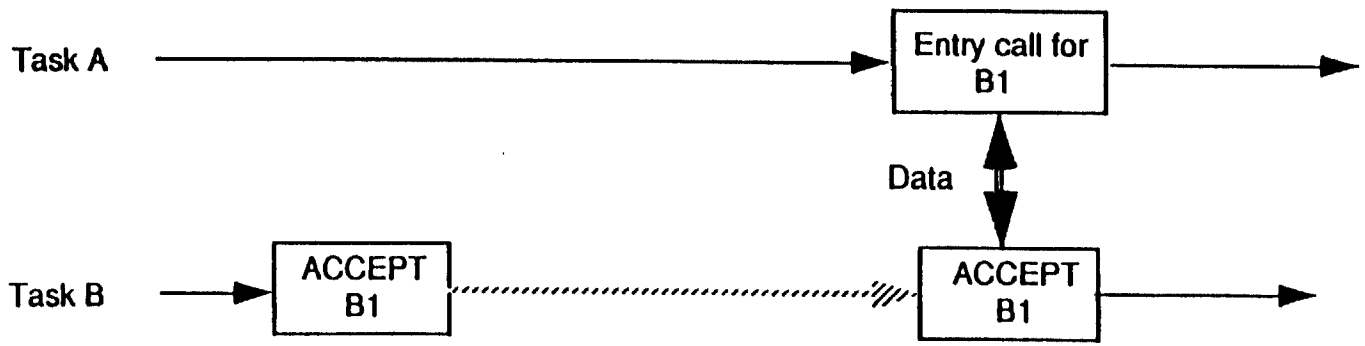
```
PROCEDURE A IS
      TASK TYPE Sample_Task_Type;                    -- Task Type specification
      TYPE Task_Ptr  IS ACCESS Sample_Task_Type;
      TASK B, C, D: Task_Ptr ;                       -- Declaration for task ptrs B, C, D
      TASK BODY Sample_Task_Type IS                  -- Task Type Sample_Task_Type Body
      BEGIN
            -- Do something
      END B;
BEGIN
      -- Do something
      B := NEW Sample_Task_Type;                     -- Task B is activated here.  A resumes
      -- Do more stuff                               -- only after B is elaborated.

      C := NEW Sample_Task_Type;                     -- Ditto for task C
      -- Do something else
      D := NEW Sample_Task_Type;                     -- Ditto for task D
      -- Finish up
END A;                              -- A cannot terminate until B, C, and D have
```

SIMPLE RENDEZVOUS (EXAMPLE 1)

Entry call occurs before ACCEPT statement



ACCEPT statement occurs before Entry call



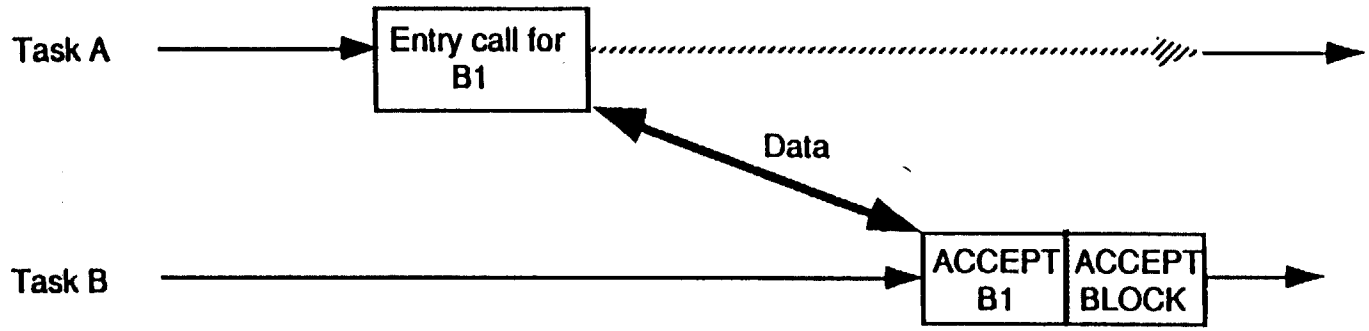
Intermetrics

SIMPLE RENDEZVOUS CODE

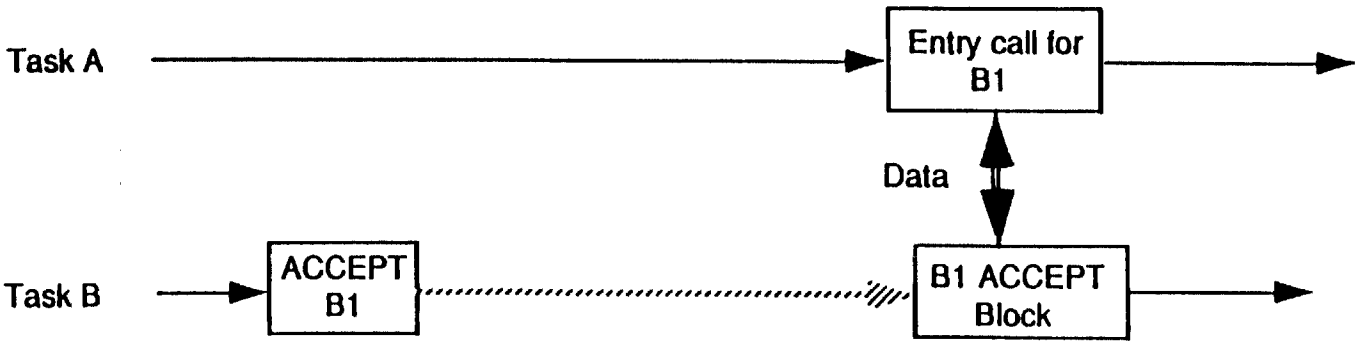
```
PROCEDURE A IS
    TASK B IS                          -- Declare task interface with a single entry
        ENTRY Sync (x:Integer);
    END;
    TASK BODY B IS
    BEGIN
        -- Do something
        ACCEPT Sync (i:Integer);       -- This is where the task looks for entry calls
        -- Do something else
    END;
BEGIN
    -- Do something
    B.Sync (5);                        -- Entry call for Sync in Task B
    -- Do something else
END A;
```

SIMPLE RENDEZVOUS WITH ACCEPT BLOCK (DIAGRAM)

Entry call occurs before ACCEPT statement



ACCEPT statement occurs before Entry call



Intermetrics

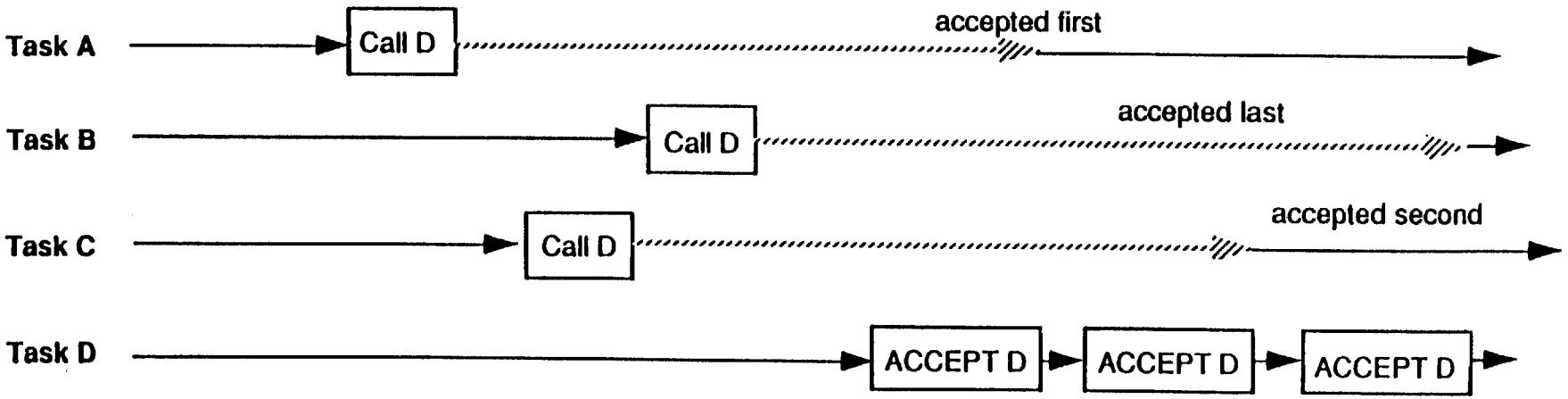
SIMPLE RENDEZVOUS WITH ACCEPT BLOCK (CODE SAMPLE)

```
PROCEDURE A IS
    TASK B IS
        ENTRY Sync;
    END;
    TASK BODY B IS
    BEGIN
        -- Do something
        ACCEPT Sync DO                  -- This is where the task looks for entry calls
        -- Do something                 -- Calling task is blocked for this code
        END;                            --     Critical region
        -- Do something else            -- Calling task is released before this code
    END;
BEGIN
    -- Do something
    B.Sync;
    -- Do something else
END A;
```

Intermetrics

MULTIPLE ENTRY CALLS (FIFO QUEUE)

- **If more than one task call the same entry for an accepting task, these entry calls are placed into a queue, and serviced first-in / first-out.**



Intermetrics

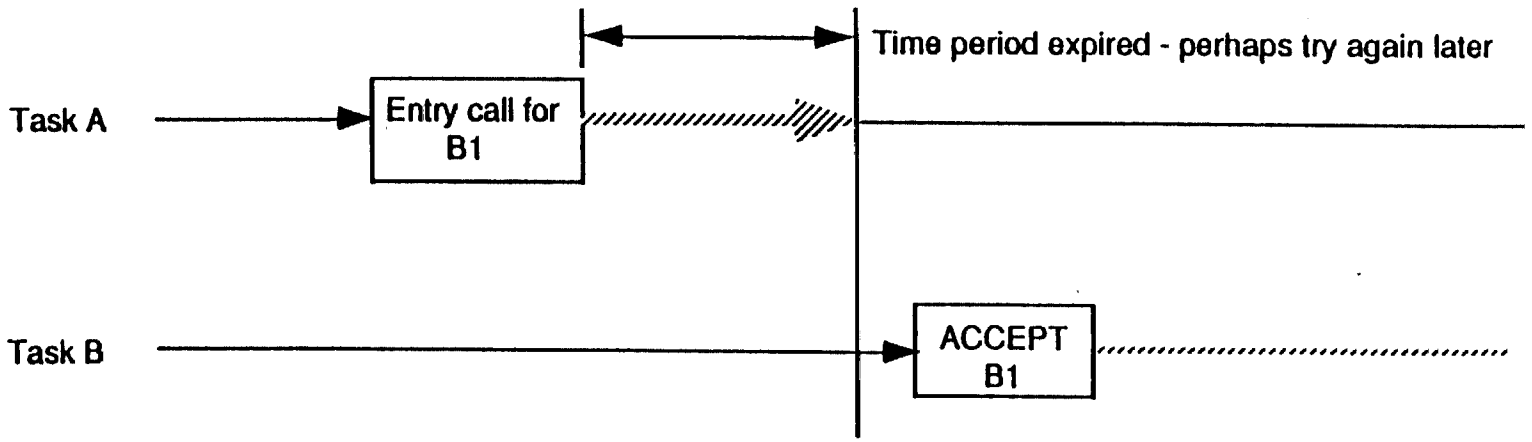
MULTIPLE ENTRY CALLS
CODE SAMPLE

```
PROCEDURE A IS
    TASK B;
    TASK C;
    TASK D IS
        ENTRY Sync;
    END;

    TASK BODY B IS
    BEGIN
        -- Do something
        D.Sync;
        -- Do something
    END B;

    BEGIN
        -- Do something
        D.Sync;
    END;
```

```
TASK BODY D IS
BEGIN
    -- Do something
    ACCEPT Sync;
    -- Do something
END D;

BEGIN
    -- Do someting
    D.Sync;
    -- Do something
END A;
```

TIMED ENTRY CALL

- **Calling task specifies how long it is willing to wait.**
  - -- If the entry call is not accepted before the time runs out, then the entry call is cancelled.
  - -- If the entry call is accepted in time, then the entry proceeds normally.

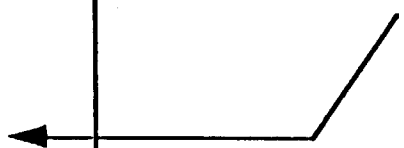


Intermetrics

TIMED ENTRY CALL SAMPLE CODE

```
PROCEDURE A IS
      TASK B IS
            ENTRY Sync;
      END B;
      TASK BODY B IS
      BEGIN
            -- Do something
            ACCEPT Sync;
            -- Do something
      END B;
BEGIN
      -- Do something
      SELECT
            B.Sync;
      OR
            DELAY 10.0;
            -- Do something
               appropriate
      END SELECT;
      -- Do something
END A;
```

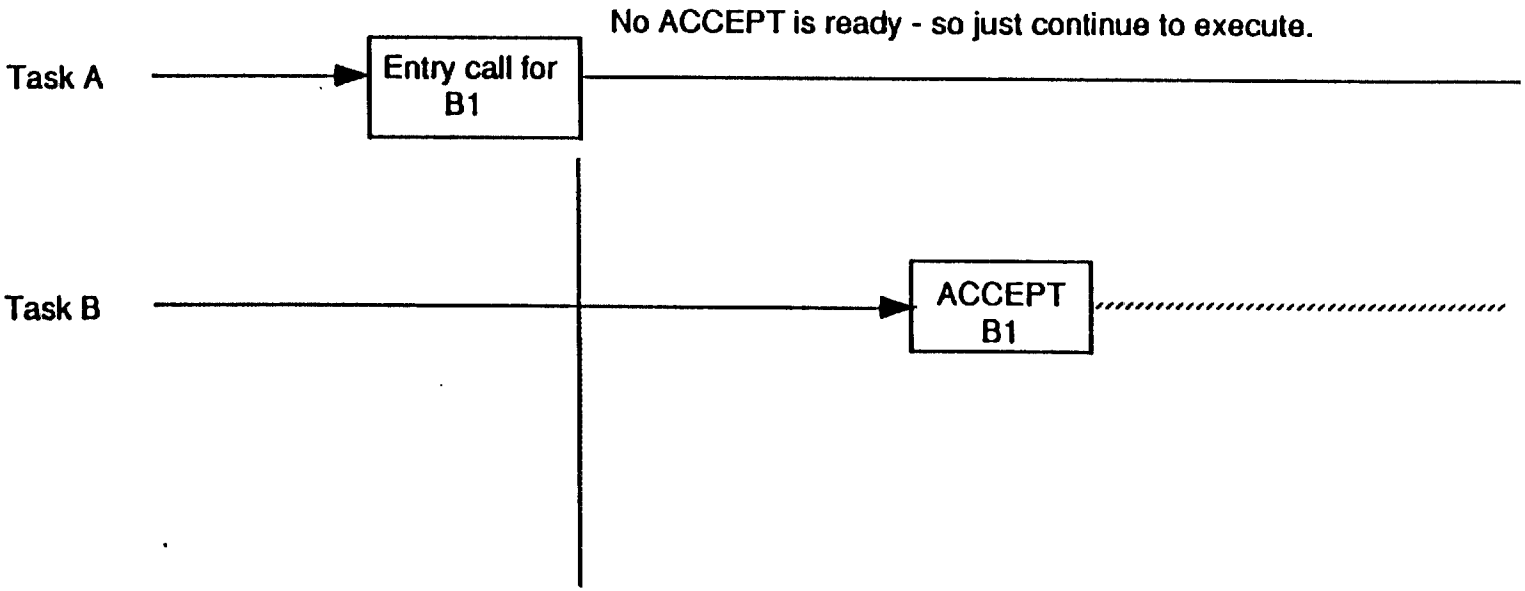
If B.Sync is not accepted wtihin 10.0 seconds, then some alternative actions will be taken.



Intermetrics

CONDITIONAL ENTRY CALL

- **Calling task specifies what to do if the entry call is not immediately accepted.**
  - -- If the entry call is not immediately accepted, the entry call is cancelled and a specific sequence of statements are executed.
  - -- If the entry call is accepted in time, then the entry proceeds normally.



Intermetrics

CONDITIONAL ENTRY CALL SAMPLE CODE

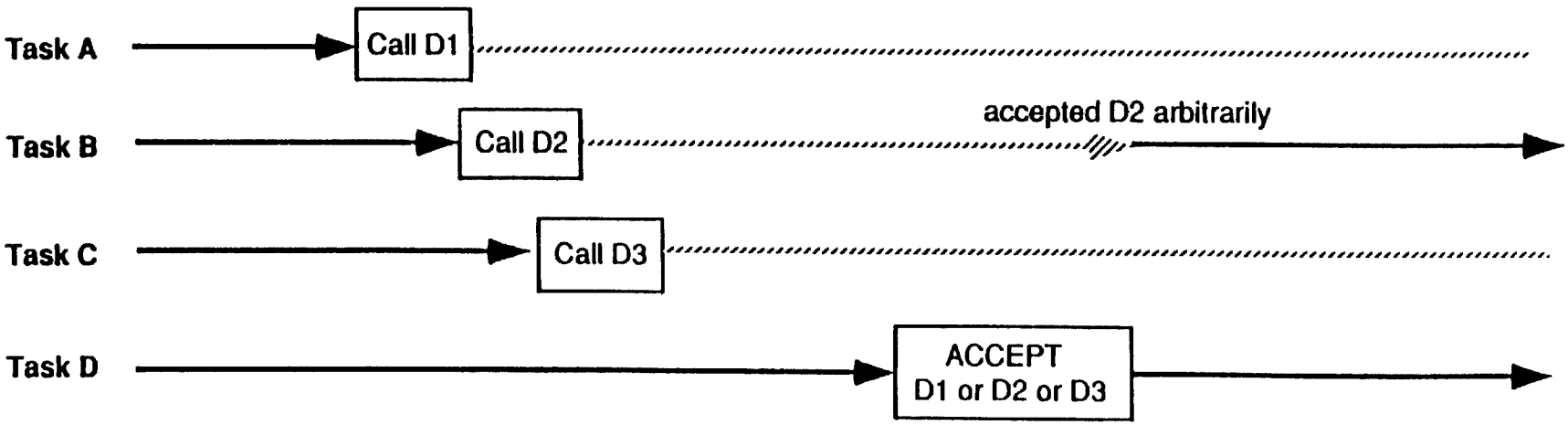
```
PROCEDURE A IS
    TASK B IS
        ENTRY Sync;
    END B;
    TASK BODY B IS
    BEGIN
        -- Do something
        ACCEPT Sync;
        -- Do something
    END B;
BEGIN
    -- Do something
    SELECT
        B.Sync;
    ELSE
        -- Do something
           appropriate
    END SELECT;
    -- Do something
END A;
```

If B.Sync is not accepted immediately then some alternative actions will be taken.

Intermetrics

SELECTIVE WAIT STATEMENT

- **The SELECTIVE WAIT statement allows a server task to look at several entries at once.**
  - -- If none of the entries have been called, then the task waits.
  - -- If one or more have been called, then the task arbitrarily selects one for acceptance, and continues.
  - -- A maximum of one entry call can be accepted for each execution of the SELECT statement.



Intermetrics

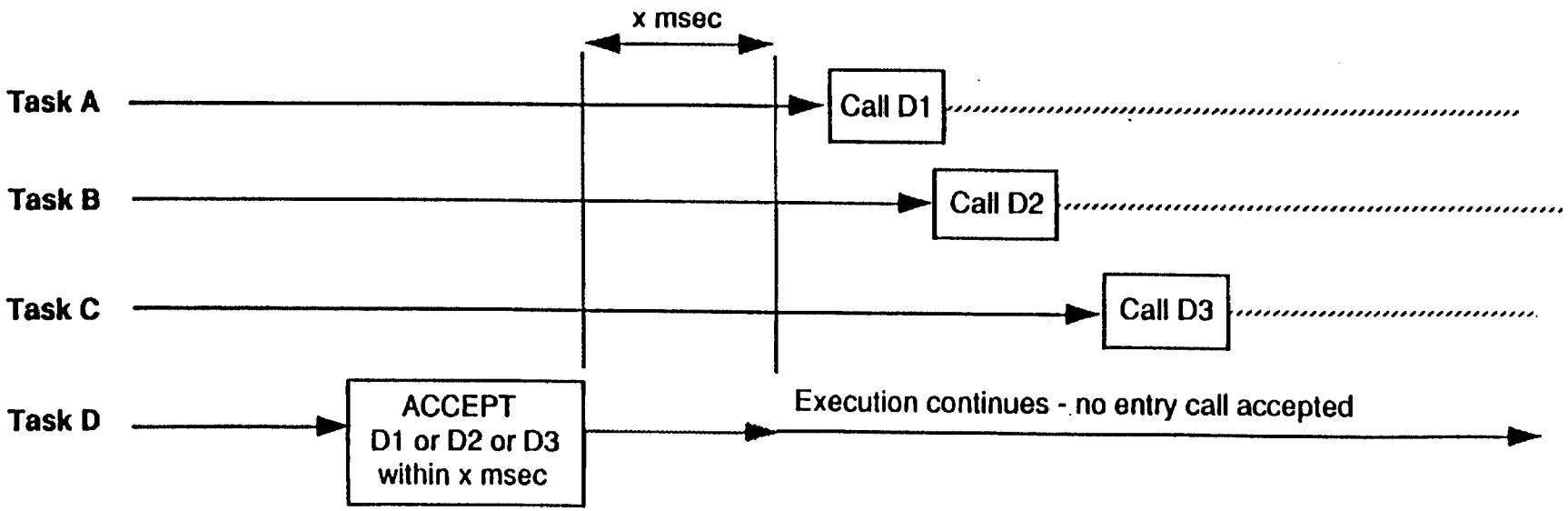
SELECTIVE WAIT STATEMENT SAMPLE CODE

```
PROCEDURE A IS
    TASK B;
    TASK C;
    TASK D IS
        ENTRY D1;
        ENTRY D2;
        ENTRY D3;
    END D;
    TASK BODY B IS
    BEGIN
        ---
        D.D1;
        ---
    END B;
    TASK BODY C IS
    BEGIN
        ---
        D.D2;
        ---
    END C;
    TASK BODY D IS
    BEGIN
        ---
        SELECT
            ACCEPT D1;
            ---
        OR
            ACCEPT D2;
            ---
        OR
            ACCEPT D3;
            ---
        END SELECT;
        ---
    END D;
BEGIN
    ---
    D.D3;
    ---
END A;
```

Intermetrics

SELECT-DELAY

- The SELECT-DELAY statement also allows a task to look at several entries at once.
- However, if none of the entries have been called, then the task waits for a specified period of time.
- If that time period expires without any of the entries being called, then the ACCEPT statement terminates, and the task continues execution.
- If one or more have been called, then the task arbitrarily selects one for acceptance, and continues.
- A maximum of one entry call can be accepted for each execution of the SELECT statement.



Intermetrics

SELECT-DELAY SAMPLE CODE

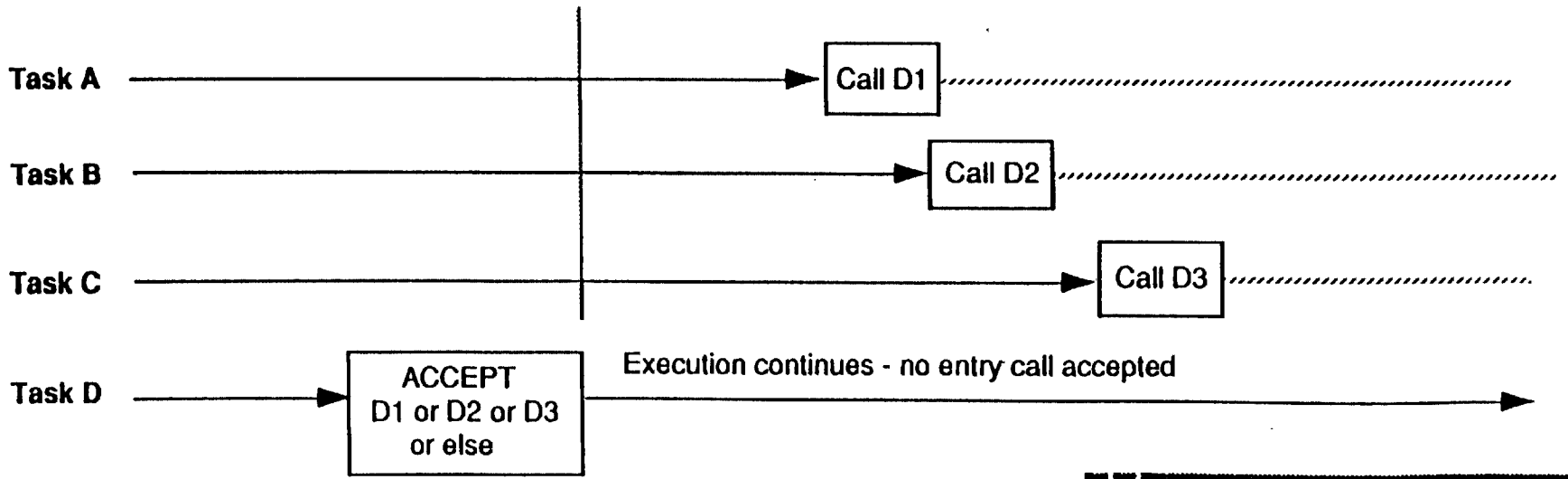
```
PROCEDURE A IS
    TASK B;
    TASK C;
    TASK D IS
        ENTRY D1;
        ENTRY D2;
        ENTRY D3;
    END D;
    TASK BODY B IS
    BEGIN
        ---
        D.D1;
        ---
    END B;
    TASK BODY C IS
    BEGIN
        ---
        D.D2;
        ---
    END C;
```

```
TASK BODY D IS
BEGIN
    ---
    SELECT
        ACCEPT D1;
        ---
    OR
        ACCEPT D2;
        ---
    OR
        ACCEPT D3;
        ---
    OR
        DELAY 5.0;
    END SELECT;
    ---
END D;
BEGIN
    ---
    D.D3;
    ---
END A;
```

Intermetrics

SELECT ELSE

- The SELECT-ELSE statement also allows a task to look at several entries at once.
- However, if none of the entries have been called, then the task terminates the ACCEPT statement immediately, and continues with its own execution.
- If one or more have been called, then the task arbitrarily selects one for acceptance, and continues.
- A maximum of one entry call can be accepted for each execution of the SELECT statement.



Intermetrics

SELECT ELSE SAMPLE CODE

```
PROCEDURE A IS
      TASK B;
      TASK C;
      TASK D IS
            ENTRY D1;
            ENTRY D2;
            ENTRY D3;
      END D;
      TASK BODY B IS
      BEGIN
            ---
            D.D1;
            ---
      END B;
      TASK BODY C IS
      BEGIN
            ---
            D.D2;
            ---
      END C;
```

```
TASK BODY D IS
BEGIN
      ---
      SELECT
            ACCEPT D1;
            ---
      OR
            ACCEPT D2;
            ---
      OR
            ACCEPT D3;
            ---
      ~~OR~~ ELSE
            DELAY 5.0;
      END SELECT;
      ---
END D;
BEGIN
      ---
      D.D3;
      ---
END A;
```

Intermetrics

SELECTIVE WAIT WITH GUARDS

- The selective wait statement allows the placement of a conditional before ACCEPT statements in a SELECT statement.
- This conditional, called a ***guard*** , permits the exercise of a greater degree of control on the part of the server task before committing to a rendezvous.
- When the SELECT statement is encountered,
  - -- All guards are evaluated first (and **once** at the beginning of the SELECT statement).
  - -- For all ACCEPT statements that have either True guards, or no guards at all, the state of their ENTRY call queues are examined.
    - -- If no entry calls are waiting, then this task goes into waiting until an entry call is issued for one of these specific entries.
    - -- If there is an entry call waiting, then an arbitrary selection of one of these is made.
  - -- If no ACCEPT statements qualify, then the SELECT statement is exited.

SELECTIVE WAIT WITH GUARDS
SAMPLE CODE

```
PROCEDURE A IS
    TASK B;
    TASK C;
    TASK D IS
        ENTRY D1;
        ENTRY D2;
        ENTRY D3;
    END D;
    TASK BODY B IS
    BEGIN
        ---
        D.D1;
        ---
    END B;
    TASK BODY C IS
    BEGIN
        ---
        D.D2;
        ---
    END C;

    TASK BODY D IS
    BEGIN
        ---
        SELECT
            WHEN Count > 0 =>
                ACCEPT D1;
            ---
        OR
            WHEN Count >= 0 =>
                ACCEPT D2;
            ---
        OR
            ACCEPT D3;
            ---
        END SELECT;
        ---
    END D;

BEGIN
    ---
    D.D3;
    ---

END A;
```

Intermetrics

TERMINATING TASKS

- **A task can terminate itself, or abort others**
- **SELECT-TERMINATE statement**

```
-- SELECT
      ACCEPT xxxx DO
            ...
      END;
   OR
      ACCEPT yyyy DO
            ...
      END;
   OR
      TERMINATE;
   END SELECT;
```

If the task containing this statement has determined that

1. It is finished
2. All dependent tasks are finished, or are at Select-terminate statements.

then the termiante alternative will be chosen.

ABORT STATEMENT

- ABORT Task_A, Task_X ;
- Unconditionally terminates the listed tasks.
- According to the ARM, it makes them *abnormal.*
- This is a very dangerous statement, since the effects are complicated.
  - -- For example, the aborted task could continue to execute for an undetermined period of time.
  - -- The effects if in the middle of a rendezvous are unpredictable.

ADA AND HARDWARE INTERRUPTS

- Ada permits an implementation to map interrupts into task entry calls.
- `FOR HW_Interrupt USE AT 16#FF02#`
- This statement would appear in a task specification.
- When the interrupt occurs, an entry call is made for that task entry
  - -- In this case, for entry HW_Interrupt

Intermetrics

PART 2
ADA TASKING SUMMARY

ADA TASKING SUMMARY

- **Ada synchronizes tasks using rendezvous.**
  - -- Inherently asynchronous
- **Ada uses FIFO queues for scheduling multiple entry calls on a single entry**
  - -- avoids starvation of lower-priority tasks.
- **Ada makes an arbitrary selection to select from among several available entry calls in a SELECT statement.**
  - -- avoids starvation of lower-priority tasks.
- **Ada uses a fixed task priority system**
  - -- in which tasks are optionally assigned priorities at compile time.
  - -- **During rendezvous, the two tasks assume the higher priority of the two**
    - If priorities are assigned to the tasks.

PROBLEMS IN ADA TASKING MODEL

- **No clear concept of time**
  - -- Delays expressed in terms of minimum time periods.
  - -- Users are interested in **maximum** time periods.
- **No easy way to support periodic tasks**
  - -- Synchronous executives
  - -- Requiring prompt responses
- **Allows *priority inversion***
  - -- in which a task with low priority can **block** a higher-priority task.
- **Allows deadlock situations**
  - -- A circular dependence of tasks in dead states.
- **Limited provisions to affect other tasks.**
  - -- Abort is dangerous

CONCURRENCY

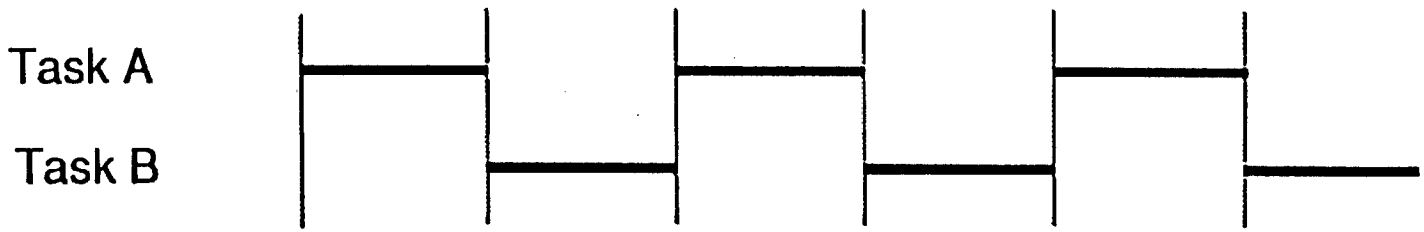
- **Concurrency can be**
  - -- Actual, when implemented on more than one processor.
  - -- Apparent, when implemented on a single processor.
- **To achieve concurrency, various forms of scheduling techniques are utilized to share processor time**
  - -- *e.g.*, time slicing, task preemption, etc.
  - -- Best technique depends on purpose of system
    - Maximum throughput (payroll processing)
    - Minimum average time-to-completion (bank queues)
    - Maximum processor utilization (fast food restaurants)
  - -- In other words, the **design of the scheduler** impacts how well the implemented system performs relative to its goals.
    - It decides what tasks will get the processor, and when.

Intermetrics

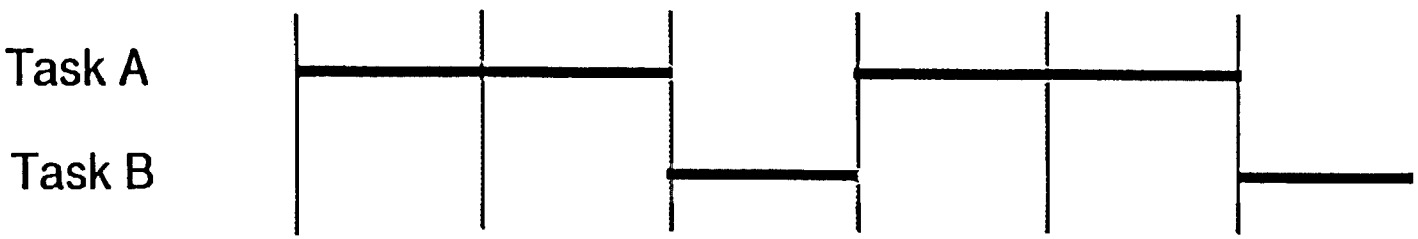
CONCURRENCY

- **Sample approaches to achieve concurrency**
- **Assume that both tasks are eligible for execution.**

Time slicing



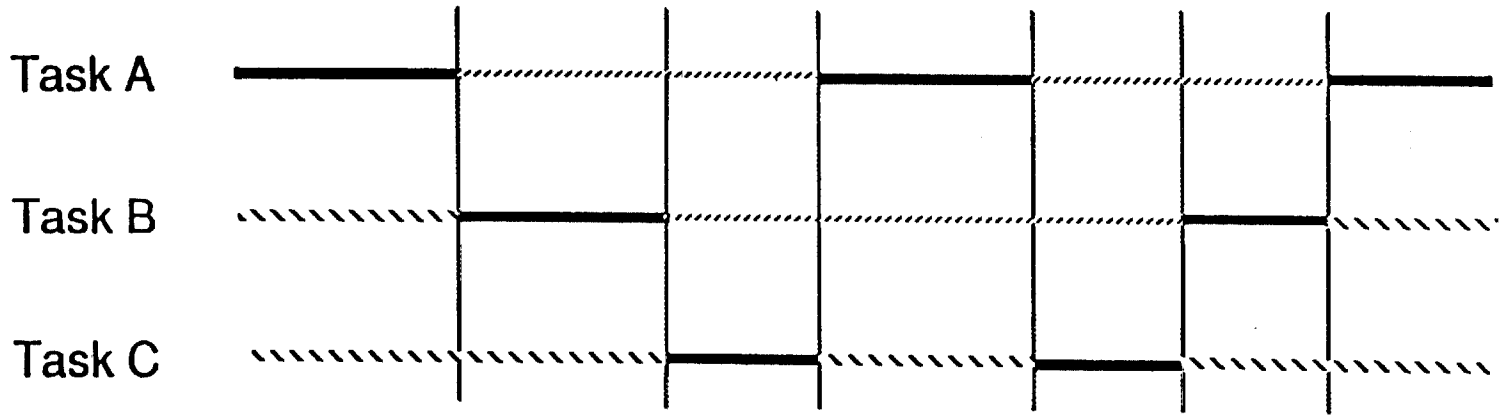
Time slicing with priorities



Intermetrics

CONCURRENCY

Preemption (with priorities)



Key

Currently executing

Blocked; waiting for something

Eligible, waiting for execution

Intermetrics

EXECUTION ELIGIBILITY IN ADA

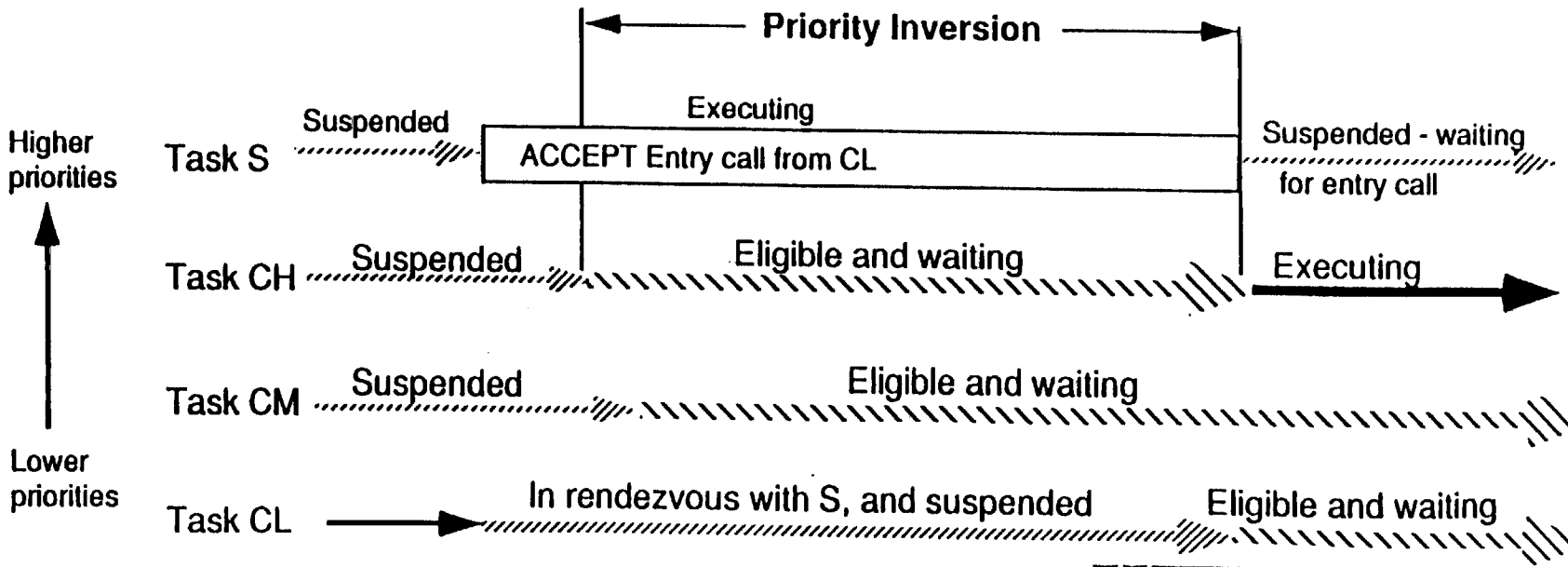
> ***If two tasks with different priorities are both eligible for execution and could sensibly be executed using the same physical processors and the same other processing resources, then it cannot be the case that the task with the lower priority is executing while the task with the higher priority is not.***

- **Ada requires that whenever its scheduler is ready to decide which task is to execute next, it must select that task from those that are eligible for execution and that have the highest priority, exclusively.**
- **Except, Ada does not specify what to do with tasks that have no priority.**

PRIORITY INVERSION

- **One problem with the Ada technique is that it allows *priority inversion.***
- **A low priority activity can take over the processor.**
- **Example 1:**
  - -- A server task has the highest priority.
  - -- Three other client tasks are also present, with lower priorities.
  - -- In this example, Task S, the server task, is doing low-priority work (for Task CL)

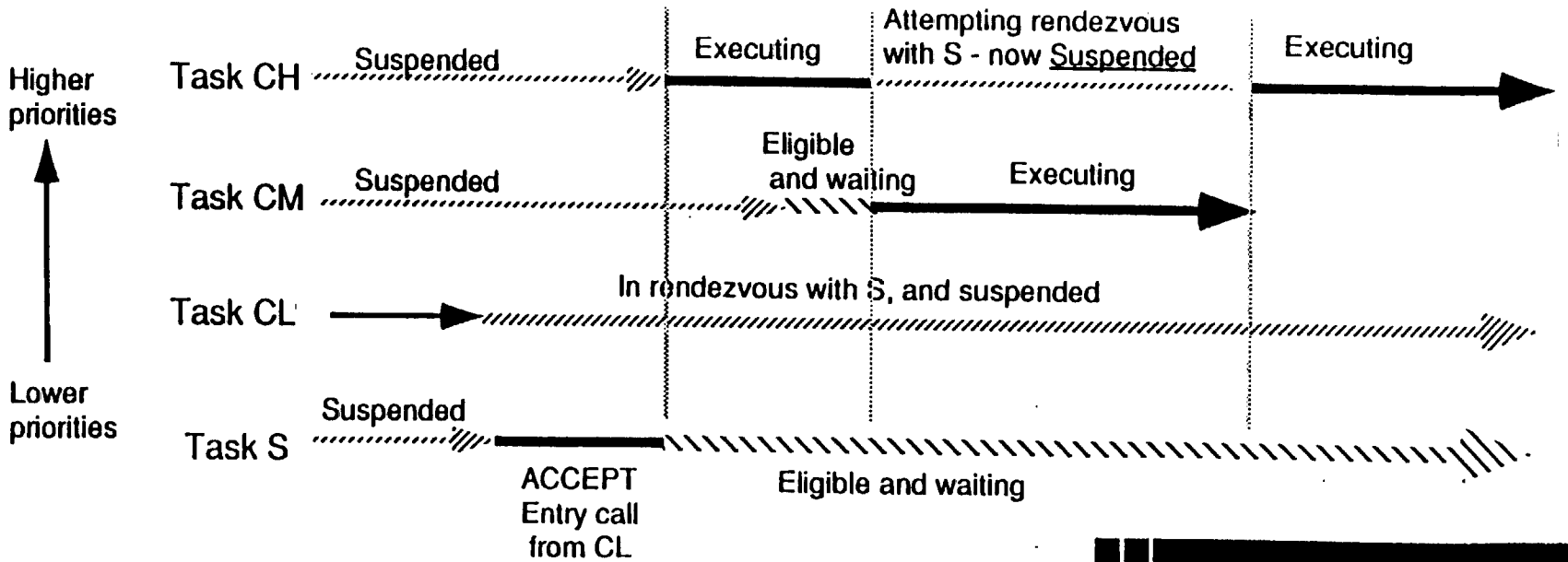
  BUT
  - -- Tasks CH and CM cannot get CPU cycles since Task S has higher priority.



Intermetrics

PRIORITY INVERSION EXAMPLE 2

-- A server task S has the lowest priority.
-- Three other client tasks are also present, with higher priorities.
-- Lowest priority task, CL, is in rendezvous with S.
-- When the highest priority task, CH, becomes eligible, it begins to execute, interrupting the rendezvous.
-- CL now tries to rendezvous with S. S is waiting (not executing, and not in a position to accept the entry call since the previous rendezvous has not been completed), and so CL is suspended.
-- The highest priority eligible task is CM, according to Ada rules.
-- So CM executes while CH is suspended.



Intermetrics

ADA TASKING DEFICIENCIES

- **Real-time systems have another goal besides to avoid starvation -**

  ⟶ **to avoid missed deadlines**

- **A real-time scheduling system should provide:**

  -- **Priority inherentence** - in which a server task assumes the priority equal to the maximum of its priority and the priorities of all of the client tasks in the queue.

  -- **Variable priorities,** via either

  - **Dynamic priority assignment,** in which priorities can be computed and assigned at each schduling decision, or
  - **Modifiable priority assignment,** in which priorities are computed at system configuration time.

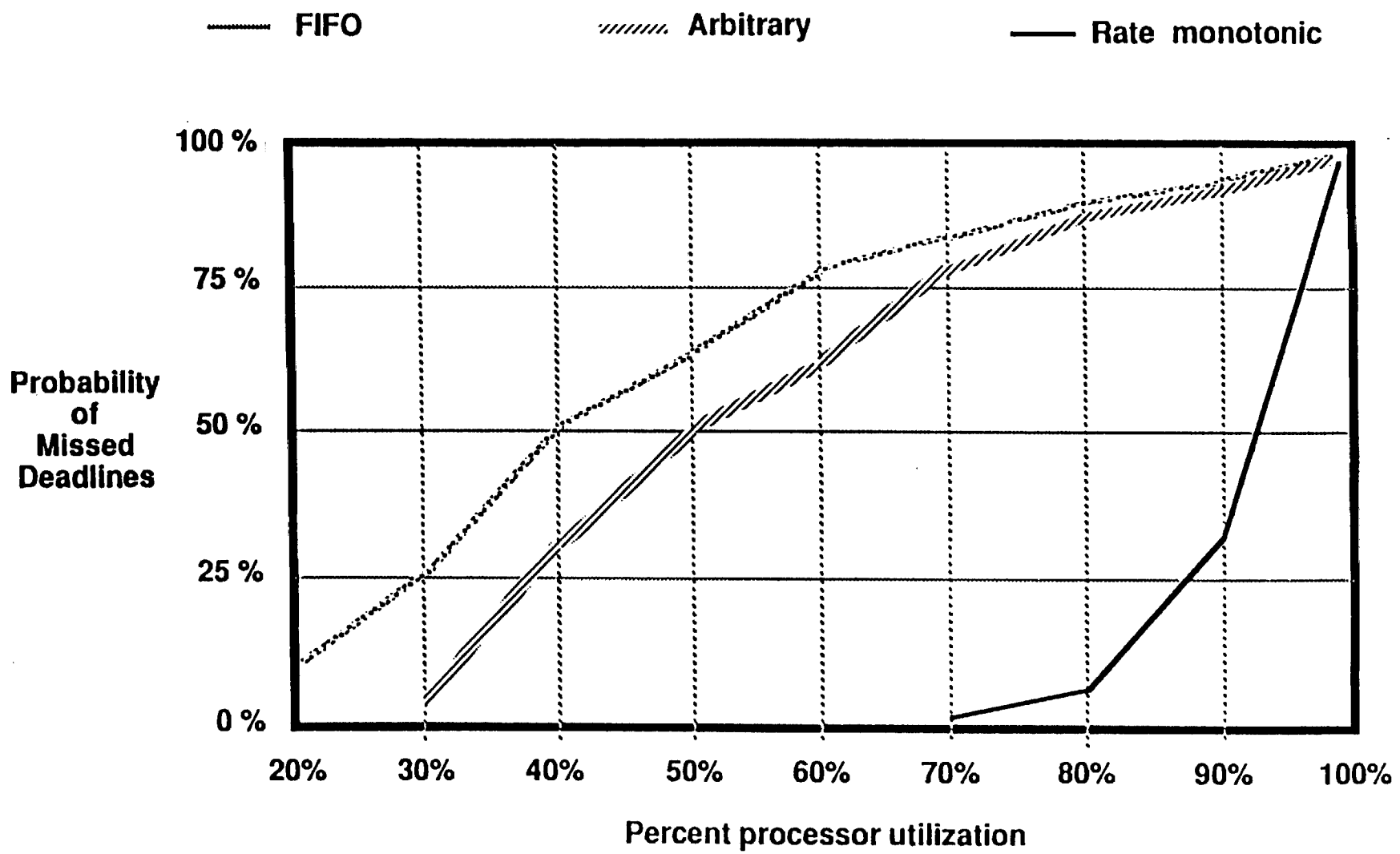
Intermetrics

EFFECT OF SCHEDULING TECHNIQUE ON MISSED DEADLINES

- **Ref: Cornhill et al. "Limitations of Ada for Real-Time Scheduling" International Workshop on Real-Time Ada Issues, 13-15 May 1987.**
- **Three scheduling algorithms were analyzed using simulation to determine the effect on missed deadlines.**
  - -- Periodic task sets with periodicity and execution time drawn from a uniform distribution
  - -- FIFO
  - -- Arbitrary selection
  - -- stabilized rate monotonic
    - fixed priority, hard-real time scheduling algorithm
    - time slices according to priority
    - makes scheduling decisions based on job importance, periodicity, and average and worst case execution times.
- **The FIFO had a 50% chance of missing deadlines with only a 40% processor loading.**

Intermetrics

PROBABILITY OF MISSED DEADLINES



Intermetrics

TASKING EFFICIENCY OF ADA COMPILERS

- Burger and Nielsen. "An Assessment of the Overhead Associated with Tasking Facilities and Task Paradigms in Ada". AdaLetters vii.1-49.
- DEC VAX 8600/VMS with DEC Ada V1.2

Description	microsec	normalized*
Task activation and termination	1960	178
Task created via an allocator	150	14
Procedure call	11	1
Producer-Consumer	503	46
Producer-Buffer-Consumer	1220	111
Producer-Buffer-Transporter-Consumer	1694	154
Producer-Transporter-Buffer-Transporter-Consumer	2248	204
Relay	906	82
Conditional entry		
-- no rendezvous	170	15
-- with rendezvous	29	3
Selective wait with terminate	127	12
Exception in a block	222	20
Exception in a procedure	217	20
Exception during a rendezvous	962	87

* Normalized to procedure call

Intermetrics

OUTLINE

Real-time programming in Ada, Part 2

→ **Development, integration , and verification of Ada systems, Part 2**

Fault tolerance and Ada

Programming standards, techniques, and tools for Ada

Conclusions, Part 2

Intermetrics

EXAMPLE OF A PACKAGE

- **A package that supports operations with Stacks**

```
PACKAGE Stack_Support IS
      TYPE Stack_Type IS PRIVATE;
      PROCEDURE Push (Element: Integer; Stack_Object : IN OUT Stack_Type);
      FUNCTION Pop (Stack_Object  Stack_Type) RETURN Integer;
      FUNCTION Size (Stack_Object: Stack_Type) RETURN Integer;

PRIVATE
      TYPE Elements_Type IS ARRAY (1..100) OF Integer;
      TYPE Stack_Type IS RECORD
            Top_Element : Integer RANGE 0..100:= 0;
            Elements : Elements_Type;
      END RECORD;

END Stack_Support;
```

Intermetrics

EXAMPLE OF A PACKAGE

```
PACKAGE  BODY Stack_Support IS

   PROCEDURE Push (Element: Integer; Stack_Object : IN OUT Stack_Type) IS
   BEGIN
       Stack_Object.Top_Element :=  Stack_Object.Top_Element + 1;
       Stack_Object.Elements(Stack_Object.Top_Element) := Element;
   END Push;

   FUNCTION Pop (Stack_Object : Stack_Type) RETURN Integer IS
       Element : Integer;
   BEGIN
       Element := Stack_Object.Elements(Stack_Object.Top_Element);
       Stack_Object.Top_Element := Stack_Object.Top_Element - 1;
       RETURN  Element;
   END Pop;

   FUNCTION Size (Stack_Object: Stack_Type) RETURN Integer IS
   BEGIN
        RETURN Stack_Object.Top_Element;
   END Size;

END Stack_Support;
```

NOTES ON PACKAGES

- The interface to Package Stack_Support is defined on the previous slide.
- Its body, which defines how the interface is to be implemented, is defined elsewhere.
- Programs which use this package have no access to any information in the body.
  - -- In particular, the data structures, any variables, or utility routines.
- The PRIVATE portion is also hidden from outside clients.
- Client programs must use only the information contained in the package specification.
- Hence, clients cannot use tricks, short cuts, etc. based on inside information to implement functions.
- Result is a well-defined, enforceable interface.
- This enforcement helps the verification process.

DISTRIBUTED ADA PROGRAMS

- **The Ada ARM specifically talks about distributed systems.**
  - -- But does not say enough
- **Current language approach is to leave distribution up to the language implementation.**
  - -- Transparent to the programmer
- **Current development approach is to create different Ada programs for each processor.**
  - -- But lose the advantages of Ada compiler semantic checking between units.
- **Also this approach is not practical for all problems.**
  - -- Need to be able to define distribution units.
  - -- Need to be able to assign units and tasks to processors.
  - -- Need to be able to control reconfiguration in case of a failure of one of the component systems.
  - -- Perhaps a separate ***configuration language*** could be defined.
    - But why have two languages when one should do?
- **Heterogeneous systems place special demands on compilers for distributed systems.**
  - -- Need different code generators for each type of system.
  - -- Need smart linkers
- **An open problem area...**

DEBUGGING AND VERIFICATION

- **Ada compile-time semantic checks provide valuable, cost-effective system verification support.**
  - -- Catch many errors which are left undetected in other languages.
- **Program interfaces defined by Ada facilities provide stable, concise, and single-point verification targets.**
  - -- Single interfaces used by many clients are easier to verify than multiple interfaces used by these same clients.
  - -- Separating interfaces from implementations allow the changing of algorithms without the changing of their respective interfaces.
    - But still must verify operational semantics.
- **Ada programs that use tasking, and that are distributed, have special verification needs.**
  - -- Verification requires tool support.
  - -- Such as Petri nets for design simulation and analysis
    - -- Wait for tools and techniques briefing.
  - -- Such as interactive debuggers for sequential code verification.
  - -- Such as simulation environments with data logging capabilities
    - When combined with browsing tools
- **It is critical that the programs be designed with verification in mind**

COMPILER ISSUES

- **Ada compilers are immature**
  - -- Recently developed
  - -- Not heavily used, particular for tasking.
- **Compiler validation is only the ticket of admission**
  - -- The ACVC tests are not exhaustive.
  - -- Compilers are validated yet still have errors.
  - -- Compilers can be validated and still have differences in interpretation of the ARM.
  - -- The ACVC does not check performance.
    - Certain features could be inefficient (tasking, compilation,..)
    - Quality of generated code is also of concern.
- **Different compilers can be very different**
  - -- Several areas of the ARM are interpreted differently (or more liberally) by different compiler vendors.
  - -- Different compilers have dramatically different capacity limits.
  - -- Just because a program compiles on one compiler does not imply that it will compile on another compiler
    - Although for small programs your chances are better than for FORTRAN.

USING MULTIPLE COMPILERS

- Using more than one compiler on a single project must be done with care.
- Two situations:
  - One compiler for code development and unit testing, the other for integration and unit testing.
  - Different compilers for different processors.
- Problems arise from the compilers having
  - -- different capacities of code that they can compile.
  - -- sometimes subtle differences in semantics (particularly in tasking).
  - -- different quality levels of error messages.
  - -- different interpretations in the ARM.
  - -- different maturity levels
- It can be very hard to isolate errors between compilers, run-time systems, application programs, and operating systems.

INTERFACE TO OTHER LANGUAGES

- **Ada permits its programs to call programs written in other languages.**
- **Completely dependent on the compiler implementation**
- **PRAGMA INTERFACE (FORTRAN, LISP, Pascal, MACRO);**
- **The Ada program library must have a procedure interface written and compiled into the library for each external procedure invoked.**
- **The compiler must adjust for all procedure linkage protocols.**
- **Short machine procedures may be written using the predefined library package MACHINE_CODE**
  if provided by the implementation.
- **The Ada LRM says nothing about other programs in other languages calling Ada programs.**
  -- Outside of the Ada universe!

Intermetrics

EDUCATION ISSUES

- **Ada is not a difficult language to learn and to program.**
  - -- While Ada has a lot of detail, it is all relatively consistent.
  - -- A programmer need not learn all about Ada to use it effectively.
- **To use Ada effectively, the student must learn to respect the problem.**
  - -- The problem defines the structure of the solution.
  - -- If you think about the problem and its solution before you code, the program will be more natural and effective.
  - -- If you want to start coding before you have invested sufficient thought, your catch-up expenses will overwhelm what you think you have saved.
- **It is important for managers to understand Ada craftsmanship**
  - -- You have to walk a mile in their shoes.
    - -- Experience the fastidiousness of Ada
    - -- It is very different in philosophy from traditional languages.
  - -- Understand where the bottenecks are.
- **It is easy to write very bad programs in Ada**
  - -- Verification is still critically important.

OUTLINE

Real-time programming in Ada, Part 2

Development, integration , and verification of Ada systems, Part 2

→ **Fault tolerance and Ada**

Programming standards, techniques, and tools for Ada

Conclusions, Part 2

FAULT TOLERANCE

- **Fault tolerance involves:**
  - -- **Detection** - to observe that a fault has caused an error.
  - -- **Isolation** - to ensure that the error does not propogate.
  - -- **Recovery** - to attempt to restore either normal or a degraded mode of operation
- **Fault tolerance techniques have been used for many years with hardware.**
  - -- e.g. equipment redundancy
- **Fault tolerance for software is an immature discipline.**

FAULT TYPES

- **Hardware faults detected by hardware**
  - -- Overheating
  - -- Flooding
  - -- Loss of power
- **Hardware faults detected by software**
  - -- Failure of a node in a distributed system.
  - -- Noise encountered in a communications link.
  - -- Overheating.
  - -- Memory parity errors.
- **Software faults detected by hardware**
  - -- Illegal memory address
  - -- Divide by zero
- **Software faults detected by software**
  - -- Illegal values
  - -- Out of range computation
  - -- Incompatible types
  - -- constraint error

Intermetrics

HARDWARE FAULTS

- **A hardware fault can result in either a degraded performance or a loss of function.**
- **Detection can be through**
  - -- the loss of a heartbeat signal,
  - -- the failure to respond to a periodic poll, or
  - -- a device monitor (such as Built-in test)
- **Recovery can include**
  - **-- the use of standby redundant units which can be switched on to take over**
    - e.g. dual CPUs
  - **-- reconfiguration of the (distributed) system to reallocate responsibility**
    - e.g. bubble memory system
    - suspenders and a belt
  - **-- the use of software**

Intermetrics

SOFTWARE FAULTS

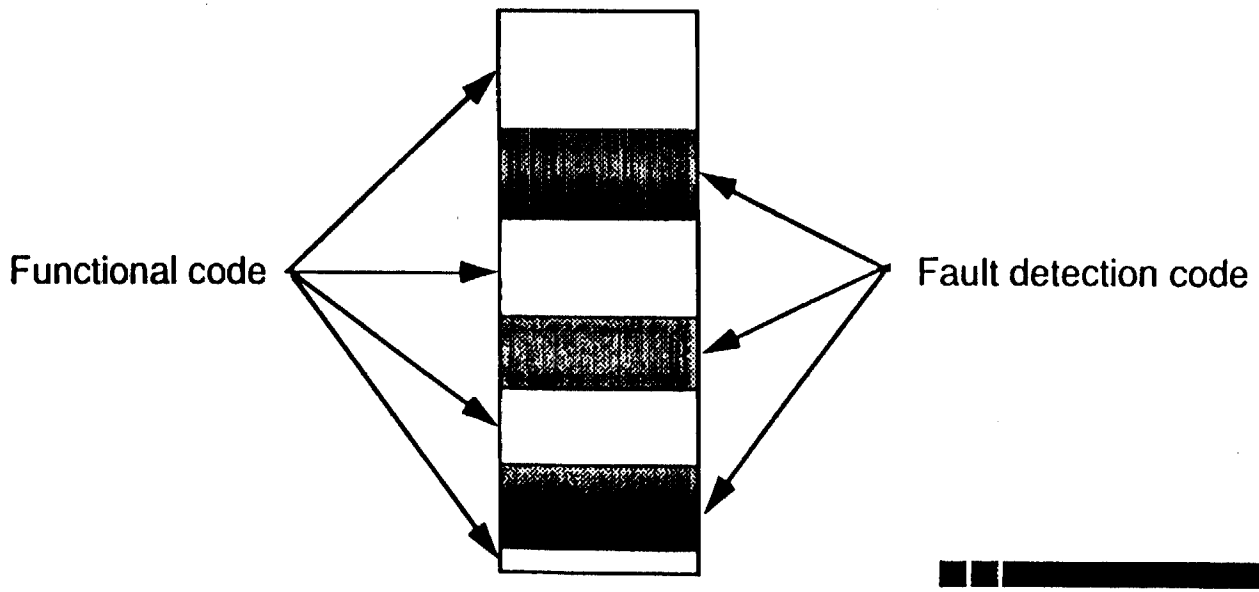
- **Software faults are very different from hardware faults**
  - -- Hardware faults can be due to
    - equipment degradation due to use
    - damage
    - design errors
  - -- Software faults are always due to design errors
- **Software redundancy is not as valuable as hardware redundancy**
  - -- If the hardware fault is due to degradation (wear-out) then a redundant piece of equipment should restore full functionality, since it does not contain the fault.
  - -- Software does not wear out.
    - So a redundant copy will contain the same fault
    - and evidence the same erroneous behavior.

SOFTWARE REDUNDANCY

- **Software redundancy can be useful.**
- **If the fault occurred as a result of an intermittent state, then a simple reset can often provide recovery.**
  - -- If the redundant software was written to the same functional specifications but by a different group of developers than the original component, then sometimes switching to it can resolve the difficulty.
  - - Unless the error is in the specifications.
- **If a hardware unit becomes unavailable, then a redundant copy of the software may become necessary in order to achieve a system reconfiguration.**
- **Ada says nothing about these issues.**

SOFTWARE FAULT TOLERANCE

- **Software faults can be detected by both hardware and software techniques.**
- **Usually, software fault recovery must be performed by software.**
- **It is very difficult to identify code that is designed to detect the occurance of errors caused by faults:**



Intermetrics

SOFTWARE FAULT TOLERANCE

- **Code that looks for faults can look like functional code.**
  - -- IF n > 0 AND THEN SumX/n < MaxValue THEN

    ...

    ELSE

    ...

    END IF;
- **Programmers regularly insert such self-checking code into their products.**
- **The reliability of such units should be higher than the unit without the "fault detection" code.**
- **What is the resulting fault rate of the software unit?**
  - -- Is the integrated unit measured
  - -- or only the "functional code"?

Intermetrics

SOFTWARE FAULT TOLERANCE

- **Fault detection code can detect:**
  - **-- Unexpected values**
    - such as out-of-range
  - **-- Unexpected states**
    - such as invalid handshaking sequences
    - delays longer than reasonable
  - **-- Incorrect computations**
    - such as slow convergence rates for approximation routines
    - unreasonable results yet still valid values.
- **Designers must integrate functional code with fault tolerant code throughout the system.**
- **To verify the fault detection and recovery code, faults must be inserted into the program during testing.**
- **Ada provides exceptions to assist in fault detection.**
- **Ada does not help in developing recovery techniques.**

Intermetrics

ADA EXCEPTIONS

- Ada supports the detection of certain types of unexpected conditions
  - -- *exceptions*
- To be used for errors or exceptional situations.
  - -- Not for commonly occuring situations
  - -- Not as a substitute for IF or CASE statements.
- Can be used for fault detection
- The Ada language has a set of predefined exceptions.
  - -- Constraint_Error
  - - Numeric_Error
  - - Program_Error
  - - Storage_Error
  - - Tasking_Error
- Users can define their own exceptions in addition to these.

ADA EXCEPTIONS

- **Handling a predefined exception:**

```
FUNCTION ReturnValue ( i : Integer ) RETURN Integer IS
   ValueArray : ARRAY (1..100) OF Integer := (OTHERS=>0);
BEGIN
   RETURN ValueArray(i);
EXCEPTION
   WHEN Constraint_Error =>
      -- Do something
END;
```

- **If this function receives a value of 101, the attempt to reference the array ValueArray(101) will result in a constraint_error.**
- **The exception handler will be given control if that happens in order to take recovery action.**

ADA EXCEPTIONS

- **User defined exceptions**

```
PACKAGE Stack_Support IS
    TYPE Stack_Type IS PRIVATE;
    Stack_Overflow, Stack_Underflow : EXCEPTION;
    PROCEDURE Push (Element: Integer; Stack_Object : IN OUT Stack_Type);
    FUNCTION Pop (Stack_Object  Stack_Type) RETURN Integer;
    FUNCTION Size (Stack_Object: Stack_Type) RETURN Integer;

PRIVATE
    TYPE Elements_Type IS ARRAY (1..100) OF Integer;
    TYPE Stack_Type IS RECORD
        Top_Element : Integer RANGE 0..100:= 0;
        Elements : Elements_Type;
    END RECORD;

END Stack_Support;
```

USER DEFINED EXCEPTIONS

```
PACKAGE  BODY Stack_Support IS

   PROCEDURE Push (Element: Integer; Stack_Object : IN OUT Stack_Type) IS
   BEGIN
      IF Stack_Object.Top_Element = 100 THEN
         RAISE Stack_Overflow_Error;
      END IF;
      Stack_Object.Top_Element :=  Stack_Object.Top_Element + 1;
      Stack_Object.Elements(Stack_Object.Top_Element) := Element;
   END Push;

   FUNCTION Pop (Stack_Object : Stack_Type) RETURN Integer IS
      Element : Integer;
   BEGIN
      IF Stack_Object.Top_Element = 0 THEN
         RAISE Stack_Underflow_Error;
      END IF;
      Element := Stack_Object.Elements(Stack_Object.Top_Element);
      Stack_Object.Top_Element := Stack_Object.Top_Element - 1;
      RETURN  Element;
   END Pop;

   FUNCTION Size (Stack_Object: Stack_Type) RETURN Integer IS
   BEGIN
       RETURN Stack_Object.Top_Element;
   END Size;

END Stack_Support;
```

OUTLINE

Real-time programming in Ada, Part 2

Development, integration , and verification of Ada systems, Part 2

Fault tolerance and Ada

Programming standards, techniques, and tools for Ada

Conclusions, Part 2

Intermetrics

CASE TOOLS MUST SUPPORT DEVELOPMENT OF DIFFERENT TYPES OF SYSTEMS

INFORMATION SYSTEMS

- Data driven
- Complex data structures
- I/O intensive
- Machine independent
- Probably use Ada-bindings to commercial standards - X-Windows, SQL

REAL-TIME SYSTEMS

- Event driven
- Simple data structures
- Computational intensive
- Machine dependent
- Probably use Ada tasking or alternative tasking kernel
- Probably use rep specs

SOFTWARE ENGINEERING

Basic premises-

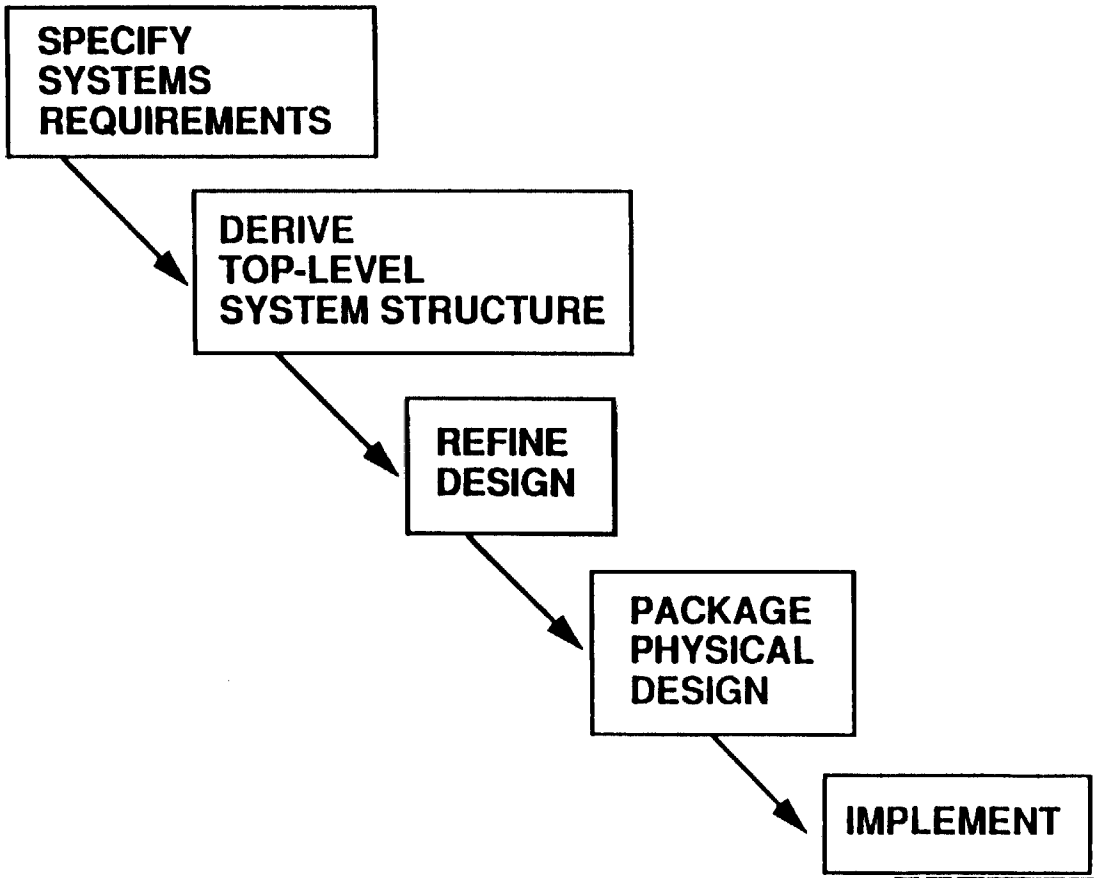
A Top-Down, Phased-Implementation approach requires that development proceed incrementally from a top-level control and data definition downward to functional modules and data structures.

Insistence on firm requirements.

Data is designed to preserve system's functionality and the independence of program modules.

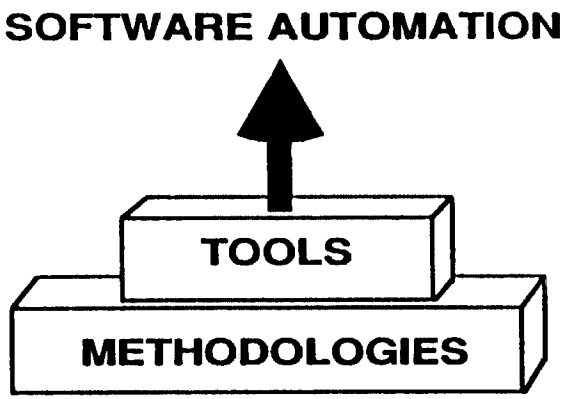
SOFTWARE ENGINEERING

- Top-Down, Procedure-Driven approach based on analysis of data flow
- Methodologies for Management of Programming



Intermetrics

CASE TOOLS, METHODOLOGIES, AND SOFTWARE AUTOMATION: WHAT IS THE CONNECTION?



AGENDA

- Differences Between Targeted System Areas
- SA/SD, ERA, OOD - The Alphabet Soup of CASE Methodologies
- Teamwork vs Excelerator
- CASE Tools, Methodologies, and Software Automation - Implication for Integration and Verification

Intermetrics

SOFTWARE ENGINEERING DIAGRAMS

- Data Flow Diagram
- Hierarchical Tree Structure Diagram
- Detailed Procedural Logic
- Screen and Report Layout

INFORMATION ENGINEERING

Basic premises-

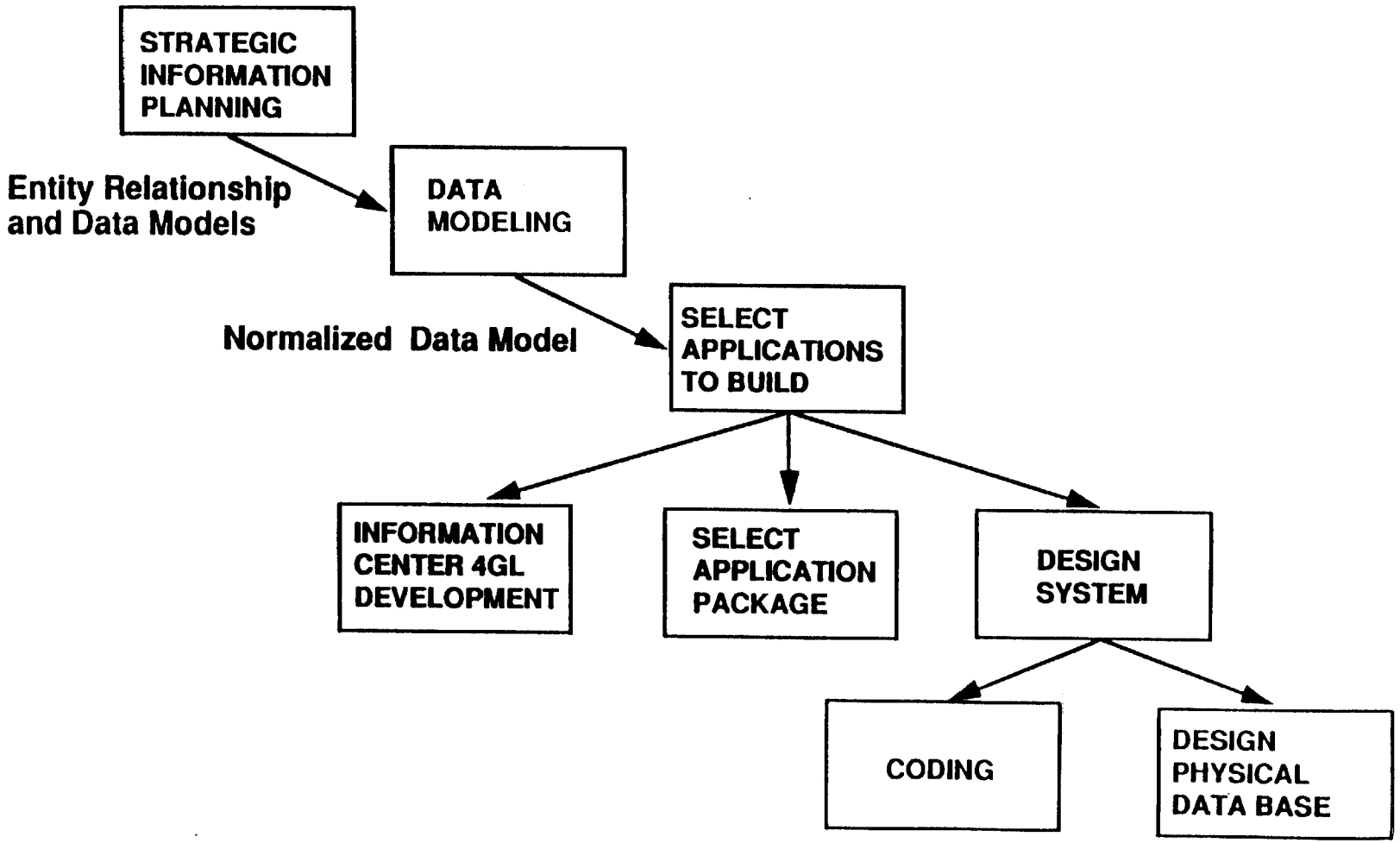
An overall system development strategy that focuses on strategic planning and business goals.

Information systems can be better integrated if data which are shared are controlled centrally by being part of the same logical data model.

The Logical Data Model, which reflects what the organization is, not how it currently operates, should be the base for system development.

INFORMATION ENGINEERING

- Top-Down, Data-Driven approach to system development



Intermetrics

CASE TOOLS, METHODOLOGIES, AND SOFTWARE AUTOMATION

Bruce Burton

Intermetrics

INFORMATION ENGINEERING DIAGRAMS

- Entity Relationship Diagram
- Data Structure Diagram
- Hierarchical Tree Structure Diagram
- Data Flow Diagram
- Screen and Report Layouts
- Detailed Procedural Logic

OBJECT ORIENTED DEVELOPMENT

Basic premises-

An overall system development strategy that focuses on objects and object operations

Emphasis is on objects - a system is a set of objects, procedures belong to objects

Objects can inherit behavior from other objects

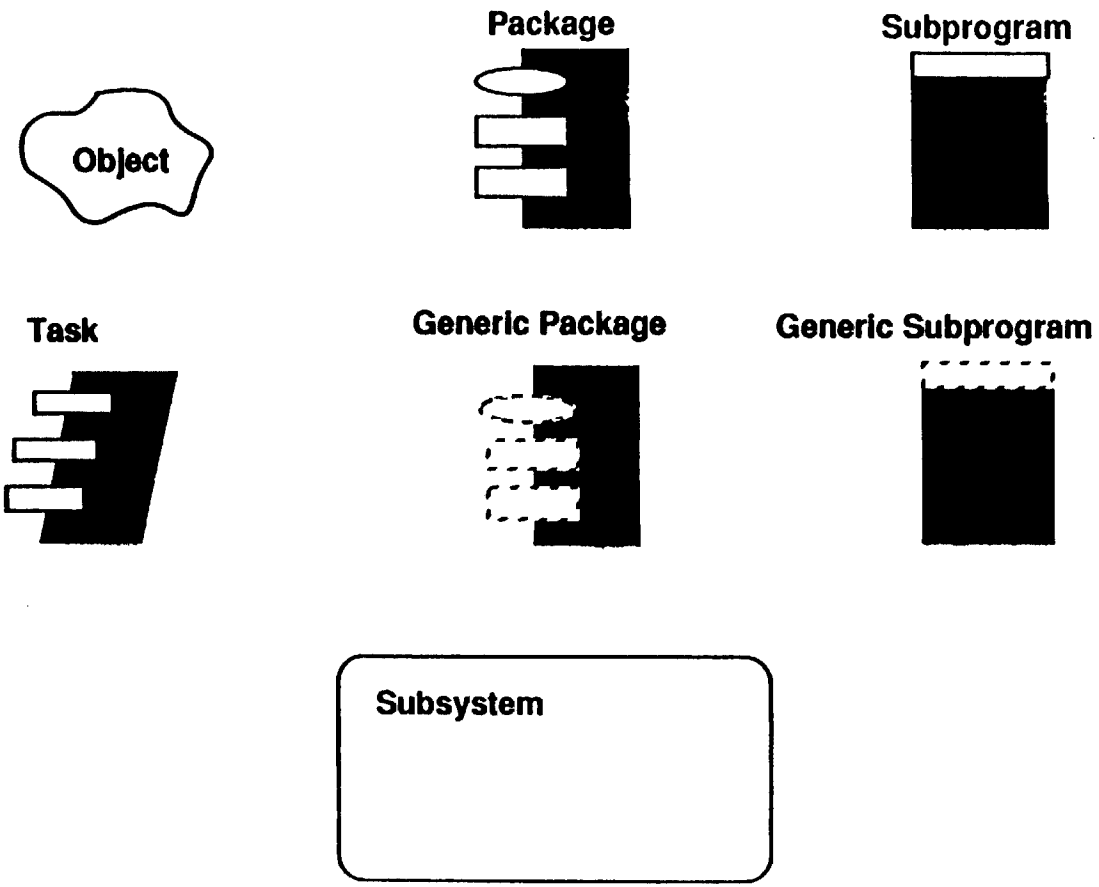
Booch/Buhr diagrams provide unique tie-in to Ada

OBJECT-ORIENTED DEVELOPMENT

- Object - indivisible unit of data and operations
- System - collection of related objects
- Inheritance - new objects are created from existing objects (method for reusability)
- Design by identifying objects and operations from the real-world

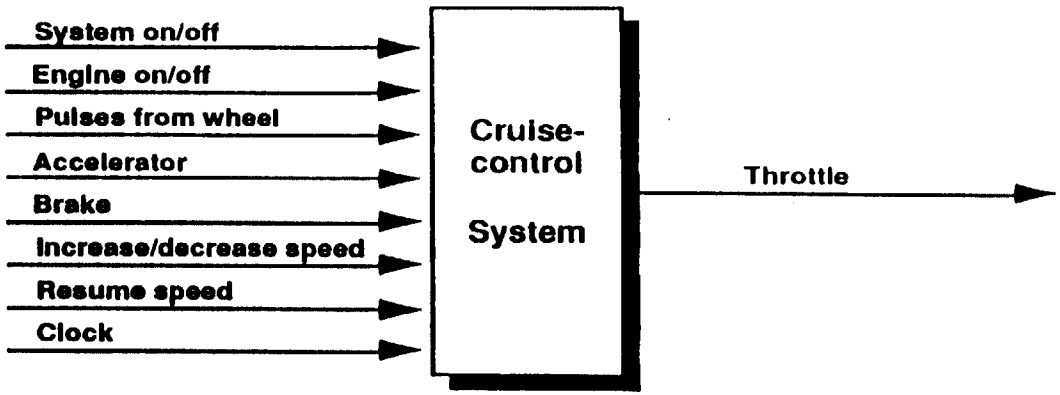
Intermetrics

SYMBOLS FOR OBJECT-ORIENTED DESIGN CAN BE DIRECTLY RELATED TO ADA COUNTERPARTS

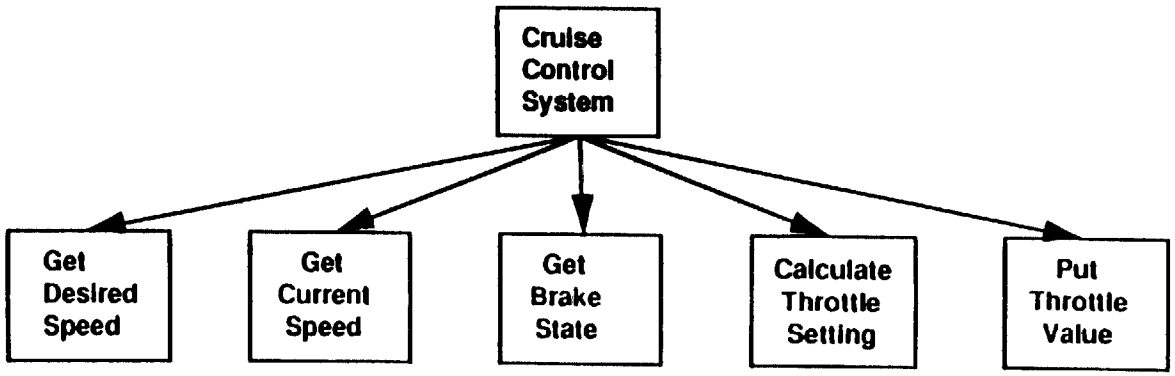


Intermetrics

CRUISE CONTROL EXAMPLE



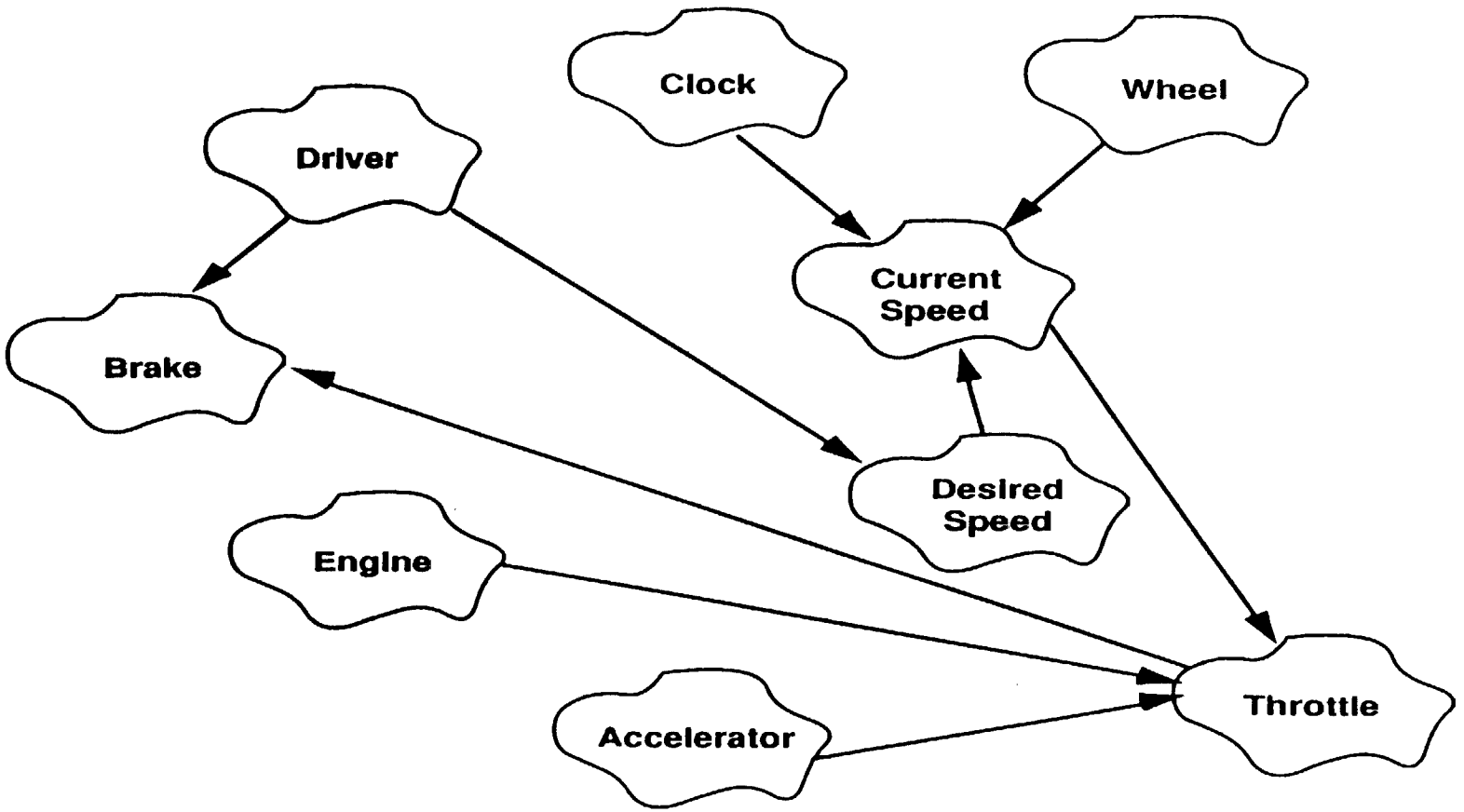
Hardware View



Typical Functional Decomposition

Intermetrics

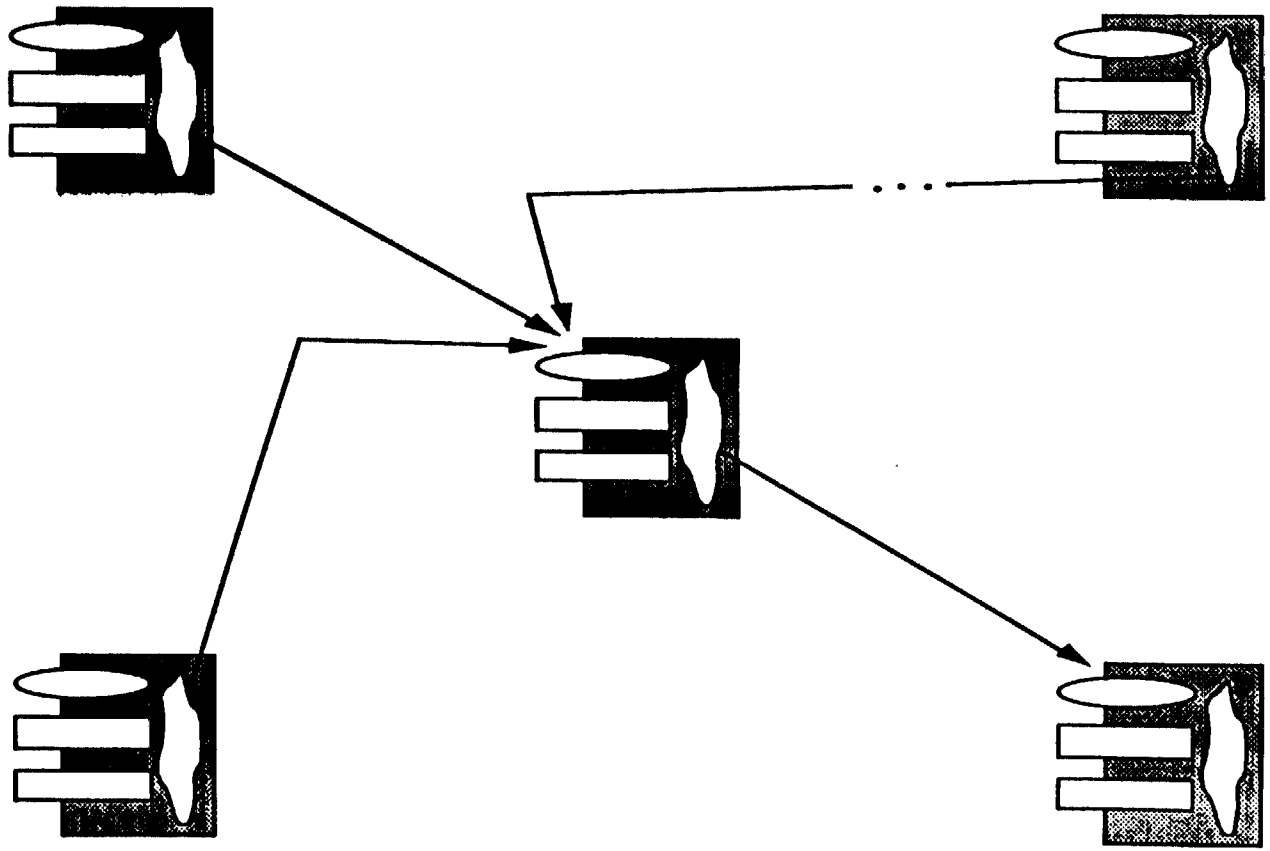
Cruise Control Example Continued



Object -oriented Decomposition

Intermetrics

CRUISE CONTROL EXAMPLE CONTINUED



Object-Oriented Decomposition

Intermetrics

BASIC STEPS OF SOFTWARE DEVELOPMENT

OBJECT-ORIENTED BASIC DEVELOPMENT STEPS	SOFTWARE ENGINEERING BASIC DEVELOPMENT STEPS	INFORMATION ENGINEERING BASIC DEVELOPMENT STEPS
1. Define objects and classes of objects	1. Define process model	1. Define data entity types
2. Define behavior of each object by specifying its operations	2. Transform process model into a program structure represented by a hierarchy of functions	2. Create data entity model showing relationships between entities
3. Define relationships (dependencies) between objects	3. Refine and expand hierarchy of functions and their interfaces	3. Expand entity model to describe attributes and to normalize model
4. Define interface of each object	4. Implement design	4. Design functions that use the dat on top of the data model
5. Implement objects		5. Implement design

TEAMWORK VS EXCELERATOR
BATTLE OF THE CASE TOOLS

VENDOR	PRODUCT	PLATFORMS	ANALYSIS/DESIGN TECHNIQUES	SML	ER	DFD	CFD	STD	TM	PS	OOD	TSK	PDL	NETWORK
CADRE	Teamwork	IBM RT, Apollo3000, Sun 3, HP9000, DEC Vaxstation	Yourdon, Chen, Hatley, others		√	√	√	√	√	√	√?		√?	√
Index	Excelerator	IBM PC, Vaxstation, Sun 3, Apollo	Yourdon, Chen, Hatley, Ward-Mellor		√	√	√	√	√	√				

General comments - **Both products are good**

Teamwork has edge in power, networking, Ada

Excelerator has edge in low end platforms, information systems

KEY:

SML - System Modeling Language

ER - Entity-Relationship

DFD - Data Flow Diagram

CFD - Control Flow Diagram

STD - State Transition Diagram

TM - Tables & Matrices

PS - Program Structure

OOD - Object-oriened Design

TSK - Tasking

PDL - Program Description Language

IMPORTANT ISSUES OF CASE AND ADVANCED METHODOLOGIES TO INTEGRATION AND VERIFICATION (I&V)

- Numerous methodologies and case tools will be used in development of SSS
- Controlled use of CASE & Ada could improve productivity and be manageable from I&V point of view
- Tailoring of design database can be used to facilitate I&V
- Integration of Ada program library with CASE design library will facilitate I&V

OUTLINE

Real-time programming in Ada, Part 2

Development, integration , and verification of Ada systems, Part 2

Fault tolerance and Ada

Programming standards, techniques, and tools for Ada

→ **Conclusions, Part 2**

Summary of NASA Experience

In support of the Software Integration and Verification Concept and Flow Analysis Task Team :

- Reviewed and commented on the Software Requirements definition and design approach of the Data Management System (DMS) and the OMA software of the SSFP.
  - Generated RIDs for the Software Requirements Specifications for each of the DMS software components.
- Performed an analysis of the DMS Runtime Object Database (RODB), and made recommendations about possible procedures to validate it.
- Supported SSFPO Level II in performing a technical audit of the Support Software Environment.
- Performed an evaluation of the APCE, and made recommendations for potential improvements.
- Gave a two day briefing on the SIB to NASA and Contractor personnel.
- Analyzed the DMS for performance issues relating to Ada, and made recommendations.
- Gave two half-day briefings to NASA and Contractor personnel on
  - Development and integration of Ada systems
  - Real-time programming in Ada
  - Tasks and communication
  - Relation of Ada to underlying OS/UNIX/POSIX
  - Design and verification of distributed systems
  - Fault tolerance and Ada
  - Programming techniques, standards, and tools for Ada.