Ada 2005

Putting It All Together

S. Tucker Taft, Chairman and CTO

SofCheck, Inc.

Ada Germany Conference

Stuttgart, Germany – October 2004

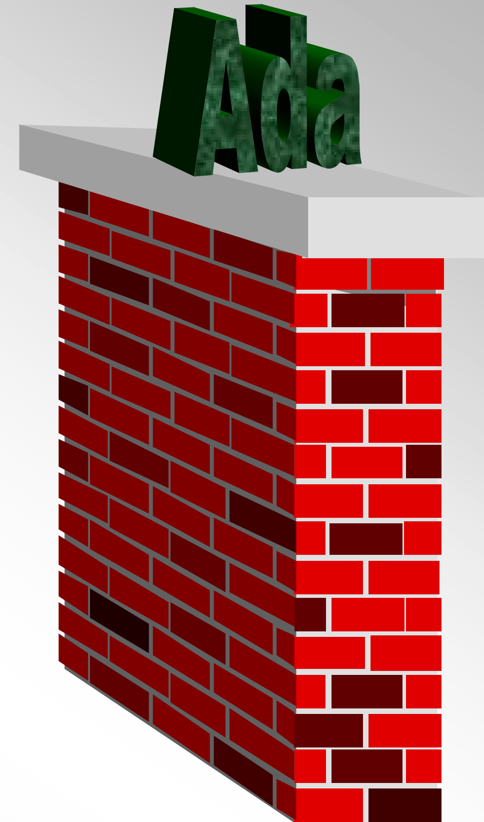
Ada is Alive and Evolving

- Ada 83 Mantra: “No Subsets, No Supersets”
- Ada 95 Mantra: “Portable Power to the Programmer”
- Ada 2005 Mantra: “Putting It All Together”...
  - Safety and Portability of Java
  - Efficiency and Flexibility of C/C++
  - Unrivaled Standardized Support for Real-Time and High-Integrity Systems
- Open-Source GNAT and Internet have fostered...
  - Active interplay between users, vendors, and language lawyers
  - Grass roots interest in Ada
  - Additional open-source contributions to compiler and library
  - Experiments with new syntax and semantics

© 2004 SofCheck, Inc.

Ada is Well Supported

- Four Major Ada Compiler Vendors:
  - ACT (GNAT Pro)
  - Aonix (ObjectAda)
  - Green Hills (AdaMulti)
  - IBM Rational (Apex)
- Several Smaller Ada Compiler Vendors
  - DDC-I
  - Irvine Compiler
  - OC Systems
  - RR Software
  - SofCheck
- Many Tool Vendors Supporting Ada
  - IPL, Vector, LDRA, PolySpace, Grammatech, Praxis...



© 2004 SofCheck, Inc.

ISO WG9 and Ada Rapporteur Group

- Stewards of Ada's Standardization and Evolution
- Includes users, vendors, and language lawyers
  - Supported by AdaEurope and SIGAda
- First "Official" Corrigendum Released 9/2000
- First Language "Amendment" Set for Fall 2005
- WG9 Established Overall Direction for Amendment...

SofCheck

© 2004 SofCheck, Inc.

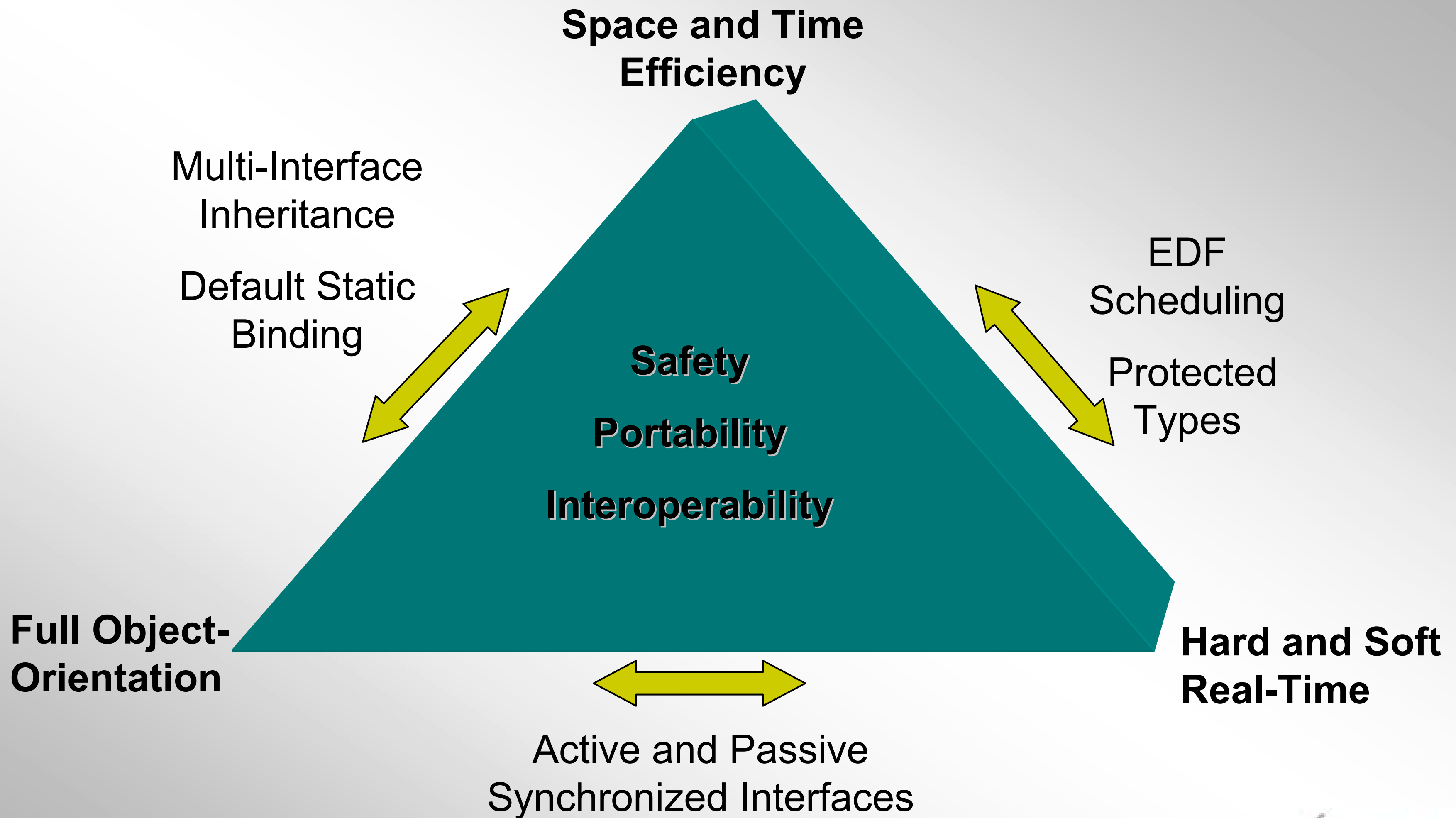
Overall Goals for Ada 2005 Amendment

- Enhance Ada's Position as a:
  - Safe
  - High Performance
  - Flexible
  - Portable
  - Interoperable
  - Distributed, Concurrent, Real-Time, Object-Oriented Programming Language
- Further Integrate and Enhance the Object-Oriented Capabilities of Ada

SofCheck

© 2004 SofCheck, Inc.

Ada 2005 – Putting It All Together



SofCheck

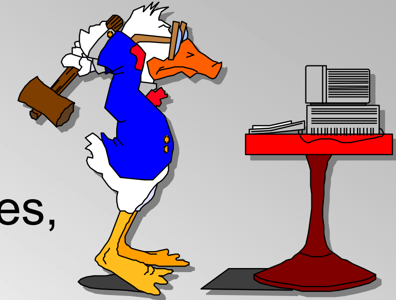
© 2004 SofCheck, Inc.

Safety is Our Most Important Product

- Ada is the premier language for safety critical software
- Ada's safety features are critical to making Ada such a high-productivity language in all domains
- Amendments to Ada carefully designed so as to not open any new safety holes
- Several amendments provide even more safety, more opportunities for catching mistakes at compile-time

© 2004 SofCheck, Inc.

Ada 2005 Safety-Related Amendments



- Syntax to prevent unintentional overriding or non-overriding of primitive operations
  - Catch spelling errors, parameter profile mismatches, maintenance confusion
- Standardized Assert Pragma
  - Assertion_Policy pragma determines how Assert is handled by implementation (Check, Ignore, …)
- Availability of “not null” and “access constant” qualifiers for access parameters
- Standardized High-Integrity “Ravenscar” Profile
- More Flexible Information Hiding Structure (“private with”)
- Standardized No_Return Pragma
  - Identifies routines guaranteed to never return to point of call
- Handlers for Unexpected Task Termination

SofCheck

© 2004 SofCheck, Inc.

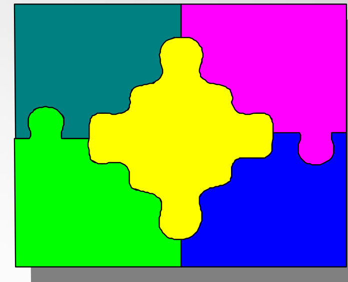
Ada 2005 Portability

- Ada is a very “strong” standard
- Vigorous publicly available validation suite of 3000+ tests
  - 2000+ Self-Checking Executable Tests
  - Also Includes Large Number of Compile-Time and Link-Time Error Detection Tests
- Active ISO Rapporteur Groups Handling Interpretation Issues
- Ada 2005 Enhancements to Existing Ada 95 Library:
  - Standard Packages for Files and Directories
  - Standard Packages for Calendar Math, Timezones, and I/O
  - Standard Package for Environment Variables
  - Standard “Container” and Sorting (Generic) Packages
- Ada 2005 Enhancements for Real-Time and High-Integrity
  - Earliest-Deadline First (EDF) and Round-Robin Scheduling
  - Ravenscar High-Integrity Run-Time Profile

© 2004 SofCheck, Inc.

Ada 2005 Interoperability

- Today's Reality:
  - The rise in importance of the Java Virtual machine and .Net common runtime
  - Increasingly complex APIs; API Wars
  - Component based systems
  - Multilingual Systems
  - Dynamically Bound Systems



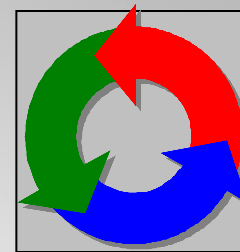
- Cyclic Dependence between types is the norm in complex O-O systems
- Emergence of Notion of "Interface" that can have multiple implementations (CORBA, Java, C#, COM)
- Amendments to Ada help address this reality

SofCheck

© 2004 SofCheck, Inc.

Enhancing Interoperability with Today's Reality

- Support Cyclic Dependence Between Types in Different Packages
  - "Limited with" context clause



- Support Notion of "Interface" as used in Java, CORBA, C#, etc.
  - "interface" types
  - Active and Passive "synchronized" interface types integrate
    O-O programming with real-time programming

- Familiar Object.Operation notation supported
  - Uniformity between synchronized and unsynchronized types

- Pragma "Unchecked_Union" for interoperating with C/C++ subsystems

SofCheck

© 2004 SofCheck, Inc.

Example of “limited with” context clause

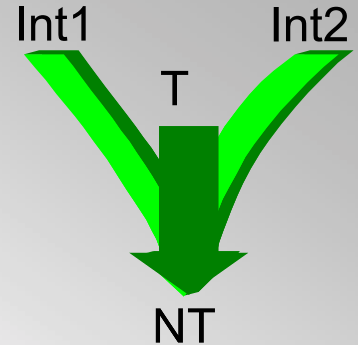
```
limited with Departments;                          -- << --
package Employees is
    type Employee is private;
    procedure Assign_Employee(E : in out Employee;
      D : access Departments.Department);
    ...
    function Current_Department(
      D : Employee) return
      access Departments.Department;
end Employees;

limited with Employees;                            -- << --
package Departments is
    type Department is private;
    procedure Choose_Manager(D : in out Department;
      Manager : access Employees.Employee);
    ...
end Departments;
```

SofCheck

© 2004 SofCheck, Inc.

Multiple Inheritance via Interface Types




- ```
  type NT is new T
      and Int1 and Int2 with
            record … end record;
  ```
- Int1 and Int2 are "Interfaces"
  - Declared as: `type Int1 is interface;`
  - Similar to **abstract tagged null record** (no data)
  - All primitives must be **abstract** or **null**
- NT must provide primitives that match all primitives of Int1 and Int2
  - In other words, NT *implements* Int1 and Int2.
- NT is implicitly convertible to Int1'Class and Int2'Class, and explicitly convertible back
  - and as part of dispatching, of course
- Membership test can be used to check before converting back (narrowing)

SofCheck

© 2004 SofCheck, Inc.
  ```

Example of Interface Types

```
limited with Observed_Objects;
package Observers is  -- "Observer" pattern
    type Observer is interface;
    type Observer_Ptr is access all Observer'Class;

    procedure Notify(O : in out Observer;
      Obj : access Observed_Objects.Observed_Obj'Class)
      is abstract;

    procedure Set_Next(O : in out Observer; Next : Observer_Ptr)
      is abstract;
    function Next(O : Observer) return Observer_Ptr is abstract;

    type Observer_List is private;
    procedure Add_Observer(List : in out Observer_List;
      O : Observer_Ptr);
    procedure Remove_Observer(List : in out Observer_List;
      O : Observer_Ptr);
    function First_Observer(List : in Observer_List)
      return Observer_Ptr;
```

© 2004 SofCheck, Inc.

Synchronized Interfaces

- Interface concept generalized to apply to Protected and Task types
  - "Limited" Interface can be implemented by:
    - Non-limited (tagged) interface
    - Synchronized interface
  - "Synchronized" Interface can be implemented by:
    - Task interfaces or types ("active")
    - Protected interfaces or types ("passive"), e.g.:

```
type Semaphore is synchronized interface;
procedure Acquire(Sem: in out Semaphore);
procedure Release(Sem: in out Semaphore);

protected type Sem_With_Caution_Period is Semaphore with
    entry Acquire;
    procedure Release;
    function Is_In_Caution_Period return Boolean;
    procedure Release_With_Caution;
private
    Sem_State: ...
end Sem_With_Caution_Period;
```

© 2004 SofCheck, Inc.

Object.Operation Syntax

- More familiar to users of other object-oriented languages
- Reduces need for extensive utilization of "use" clause
- Allows for uniform reference to dispatching operations and class-wide operations, on pointers or objects; e.g.:

```
package Windows is
    type Root_Window is abstract tagged private;
    procedure Notify_Observers(Win : Root_Window'Class);
    procedure Display(Win : Root_Window) is abstract;
    ...
end Windows;
package Borders is
    type Bordered_Window is new Windows.Root_Window with private;
    procedure Display(Win : Bordered_Window);
    ...

procedure P(BW: access Bordered_Window'Class) is
begin
    BW.Display;                    -- both of
    BW.Notify_Observers;           -- these work
```

© 2004 SofCheck, Inc.

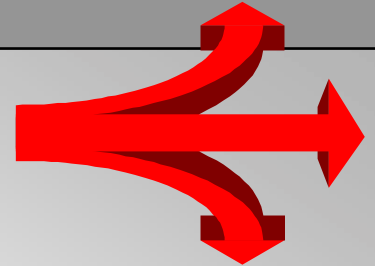
Other Enhancements…

- Generalized Use of Anonymous Access Types
- Generalized Parameterization of Formal Packages
- Make Limited Types Less Limited
- Pragma Pure_Function (from GNAT)
- “private with A.B;” – A.B only visible in private part
- Downward closures – local subprograms can be passed as parameters to other subprograms
  - Uses anonymous access-to-subprogram types for parameters.
- Task termination handlers
  - especially for termination due to unhandled exceptions

SofCheck

© 2004 SofCheck, Inc.

Generalized Use of Anonymous Access Types



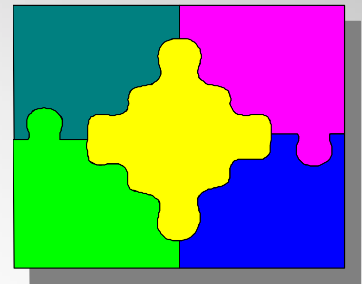
- Two kinds of generalization
  - Allow access “parameters” for access-to-constant and access-to-subprogram cases
  - Allow use of anonymous access types in components, object renamings, and function results
- Should help reduce “noise” associated with unnecessary explicit conversions of access values
- Also allow optional specification of “not null” constraint on access subtypes, and anonymous access type specifications
  - E.g.: type String_Ref is access all String not null;
  - Improves safety, efficiency, and documentation by pushing check for null to caller or assigner rather than ultimate point of use.

SofCheck

© 2004 SofCheck, Inc.

Generalized Formal Package Parameters

- Allow partial specification of actual parameters
  - In Ada 95 it is all or nothing
  - Important when there are two formal package parameters that need to be "linked" partially through their actual parameters
- Example

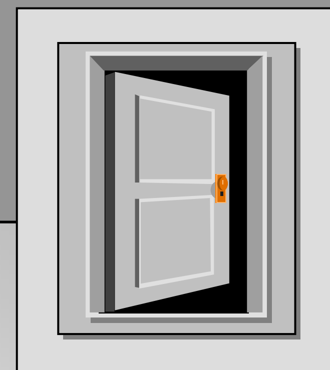


```
generic
      with package I1 is new G1(<>);
      with package I2 is new G2(
        Element => I1.Element, others => <>);
package New_Abstraction is ...
```

SofCheck

© 2004 SofCheck, Inc.

Make Limited Types Less Limited



- Allow use of explicitly initialized limited objects, where initial value is an aggregate.
  - Aggregate is built in place (as it is now for controlled types)
  - New syntax to represent “implement by default”
    - Using “<>” for this, corresponds to notion of “unspecified”
  - Still no copying allowed, and no assignment statements
  - Aggregates can be used as initial expression for declaration, as expression for initialized allocator, and as actual parameter value
- Allow functions to return limited objects
  - Simple return statement must return aggregate or function call
  - Extended return statement may build up result and return
  - Function call can be used where aggregate is allowed above
  - Replaces “return-by-reference” of Ada 95
    - Use anonymous access type results instead

SofCheck

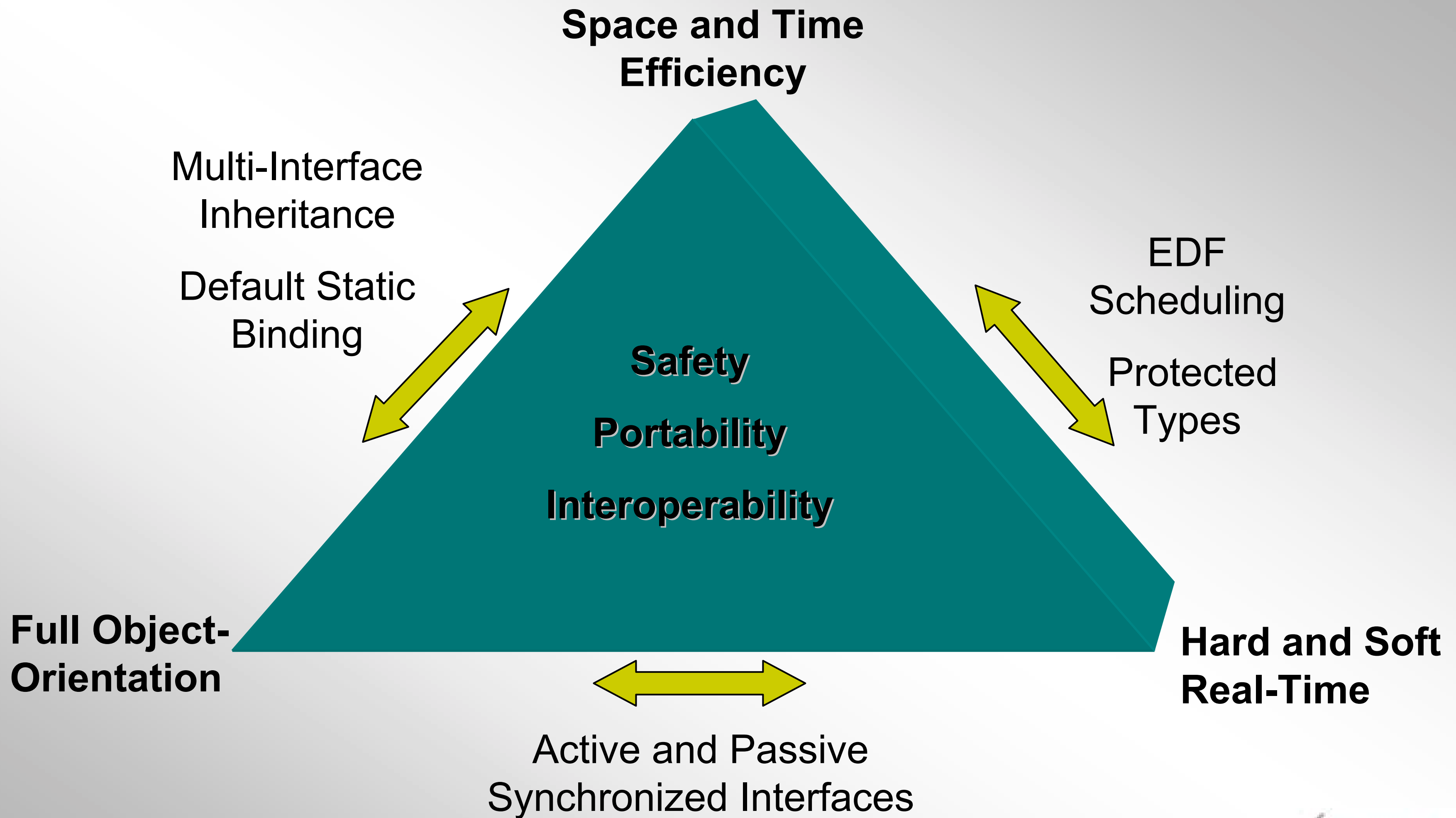
© 2004 SofCheck, Inc.

Ada 2005 Summary

- Complete the object-oriented capabilities
  - Multiple Inheritance via Interfaces
  - Cyclic Dependence between Abstractions
  - Object.Operation Notation Supported
- Enhance the standardized library
  - Containers
  - Directories, Calendar, Environment Variables
  - Linear Algebra
- Extend Ada's Unmatched Real-Time and High-Integrity Support:
  - Synchronized Interfaces to integrate O-O and Real-Time
  - High-Integrity Ravenscar Run-Time Profile
  - Enhanced Scheduling and Time Control
    - Earliest Deadline First (EDF)
    - Mixed Scheduling Across Priorities (Priority, EDF, Round-Robin)
    - Budget-based Scheduling

© 2004 SofCheck, Inc.

Ada 2005 – Putting It All Together



SofCheck

© 2004 SofCheck, Inc.