A Technical Presentation

and Global Data Checking

Presented by

Ian Gilchrist

Software Products Consultant

ian.gilchrist@iplbath.com

IPL

- **In the next 30 minutes you will see:**
  - Short statement on what AdaTEST 95 is and does.
  - Use of AdaTEST 95 to create and run a simple unit test
    - Without Global Data checking
    - With Global Data checking
    - Adding coverage analysis
  - Conclusion and questions

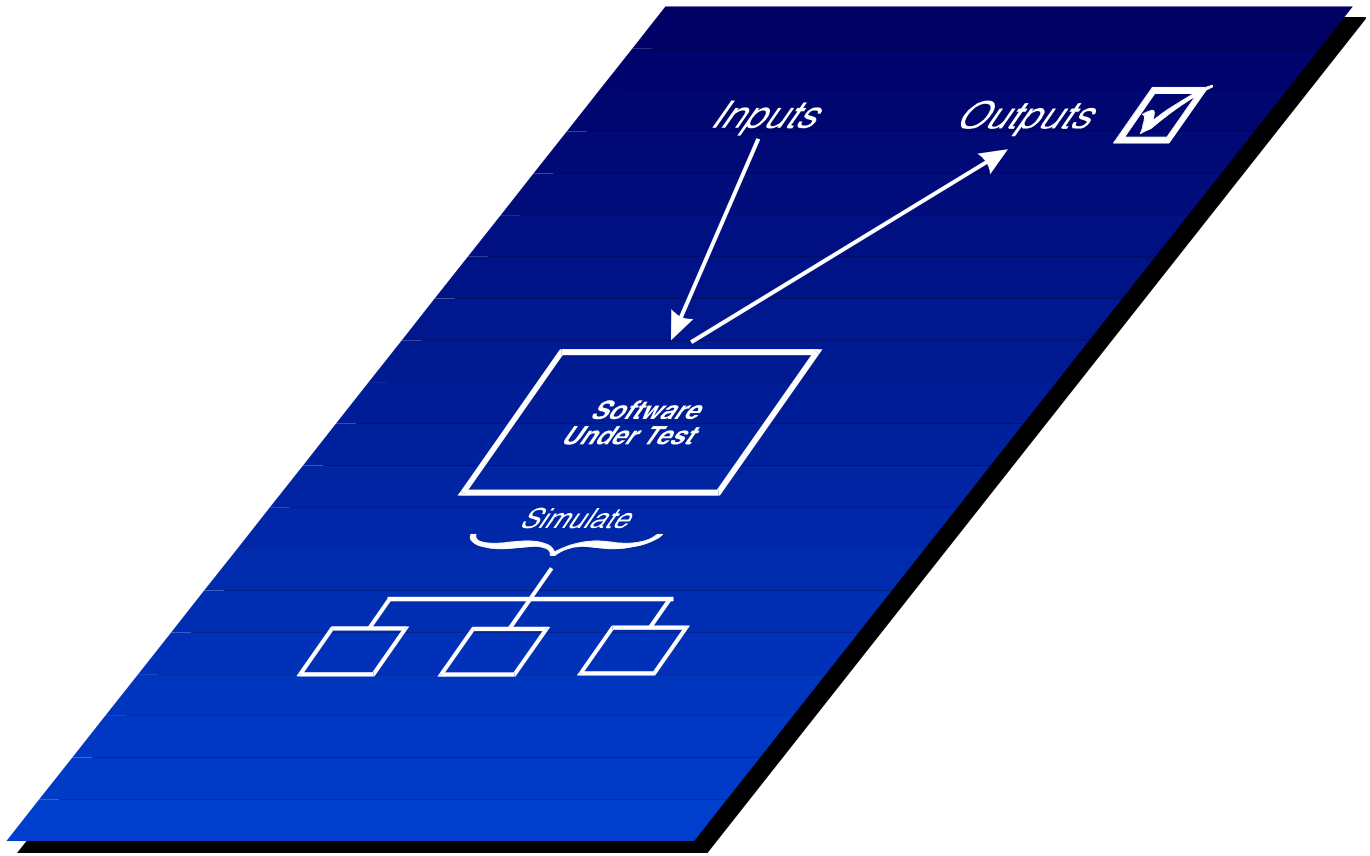
IPL

- ♦(for Ada 83), first released in 1992
- ❄followed in 1996
  - Current version is 2.0
- ❄is a unit/integration testing tool
  - **Dynamic Testing** – see below
  - **Coverage Analysis** – many types
  - **Static Analysis** – code/complexity metrics
  - Many Host and Target platforms supported
  - Certified (many times) to Safety-Crit std (DO-178B Level A)

- **Testing techniques supported include:**
  - Black and White Box testing
  - Exception monitoring
  - Programmable ‘stubs’
  - Automated build and run
  - Timing Analysis
  - Full support for tasking, elaboration code, Ravenscar…

IPL

Dynamic Testing



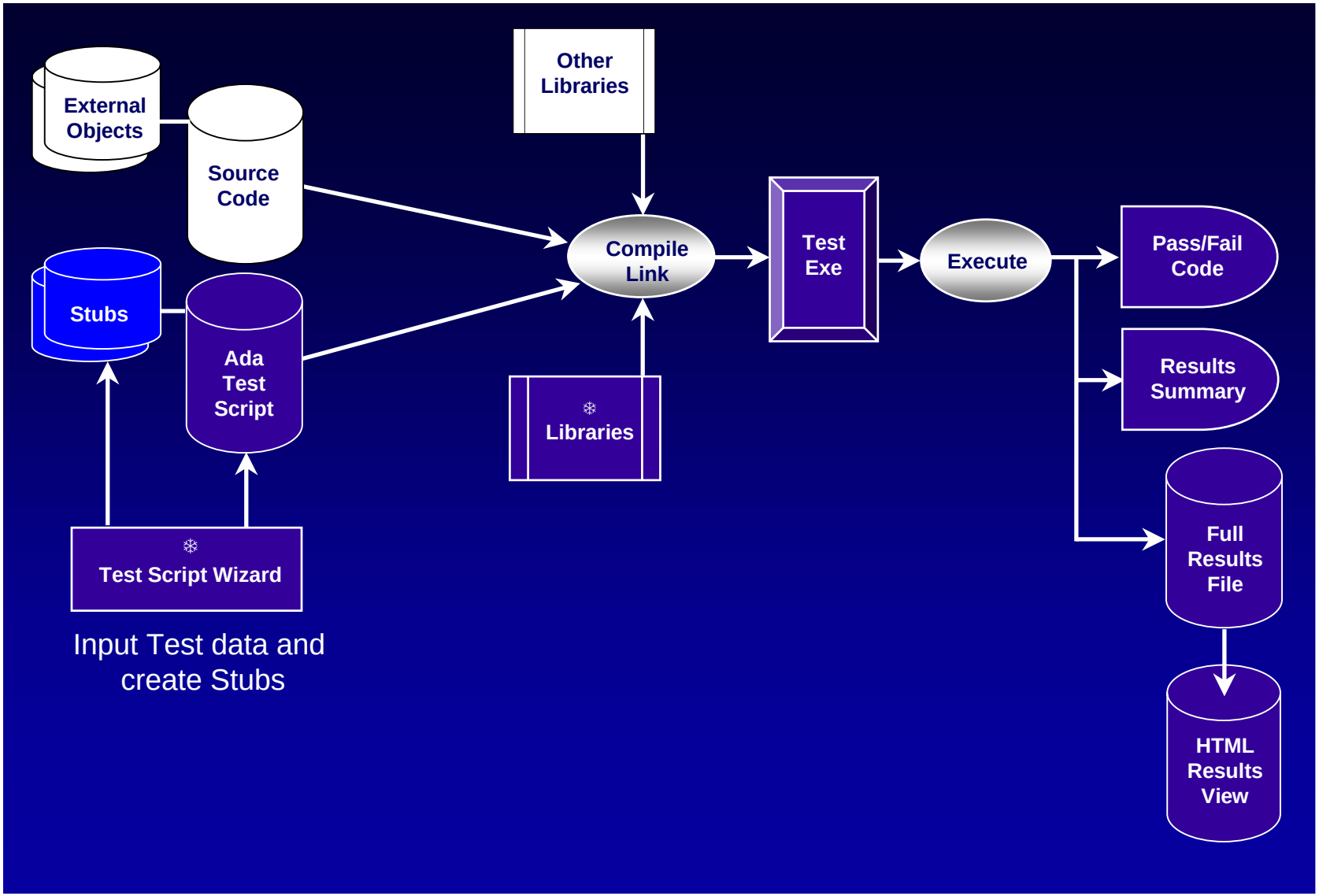
IPL

Dynamic Testing When

- **Dynamic Testing**
  - Executing the software, under known conditions
  - Predict what should happen
  - Verifying results against expected outcomes
- **Unit Test**
  - All software testing should start at the unit or module level isolation test.
  - For Ada the most convenient unit is a file containing one or more packages/classes.
- **Integration Test**
  - Clusters of packages/classes
  - Tasks
  - Sub-systems…

IPL

How ❄Dynamic Testing works



IPL

Demo - Software under Test

Class STACK specification

- A Stack object shall be created in a Null state.
- ‘Push’ shall insert integer input onto top of Stack;
  - ‘Memory_Fault’ exception shall be raised if a new node cannot be created due to memory failure.
- ‘Pop’ shall return an integer from the top of the Stack;
  - Popping the last integer shall return the Stack to a Null State.
  - Popping from an empty stack shall raise the ‘Empty_Pop’ exception.
- ‘Reset’ shall pop all items from the stack.
- Global data item ‘Memory_Status’ shall not be affected by any of the above calls.

IPL

Demo - Software under Test

```
Package Stack is
   type Object is tagged private;
   procedure Push (The: in out Object;
                   Value: in  Integer);
   procedure Pop (The: in out Object;
                  Value : out Integer);
   procedure Reset (The: in out Object);
   Empty_Pop: exception;    -- can be raised by Pop
   Memory_Fault: exception; -- can be raised by Push
   -- Global data
   type Status is (Corrupt, Valid);
   Memory_Status:Status := Valid;
Private
   type Node;
   type Access_Node;
   type Object is tagged record
        Head : Access_Node;
        end record;
   type Node is …
```

IPL

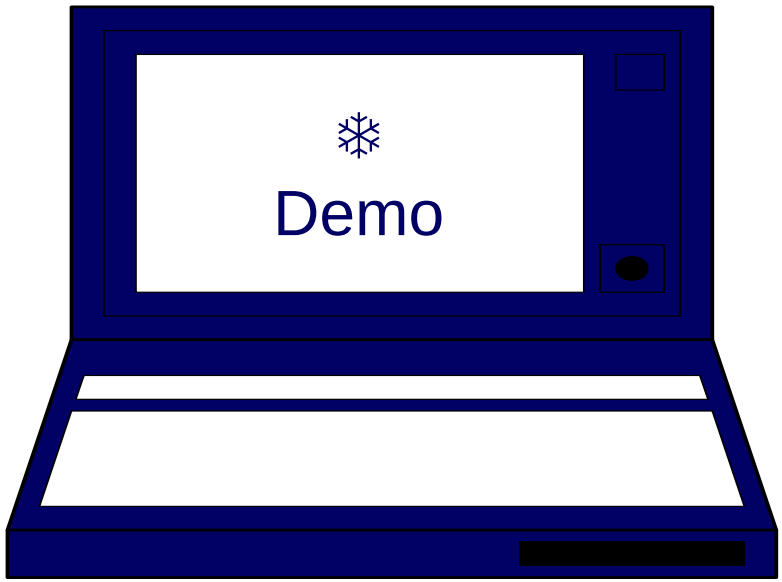
Test Preliminaries for ‘Stack’

- **White Box Testing**
  - Test script is written as ‘child’ procedure, thus giving direct access to Stack Head.
    - Easy to check Null state of Stack
- **Design for Testability**
  - A package body function ‘New_Node’ will be used to allocate memory for new nodes.
    - This can be stubbed, making it easy to simulate ‘out of memory’ situation.

Test Plan for ‘Stack’

- **Three test cases initially:**
  1. **‘New Stack’**
     - Check Head is null
     - Pop, and check Empty_Pop is raised
  2. **‘Push and Pop – nominal usage’**
     - Push and Pop three numbers
     - Check returns are as expected
     - Check Head is null
  3. **‘Reset’**
     - Push three numbers
     - Reset
     - Check Head is null

- **Dynamic Test for package Stack**



Demo - Software under Test

Class STACK specification

- A Stack object shall be created in a Null state.
- ‘Push’ shall insert integer input onto top of Stack;
  - ‘Memory_Fault’ exception shall be raised if a new node cannot be created due to memory failure.
- ‘Pop’ shall return an integer from the top of the Stack;
  - Popping the last integer shall return the Stack to a Null State.
  - Popping from an empty stack shall raise the ‘Empty_Pop’ exception.
- ‘Reset’ shall pop all items from the stack.
- Global data item ‘Memory_Status’ shall not be affected by any of the above calls.

IPL

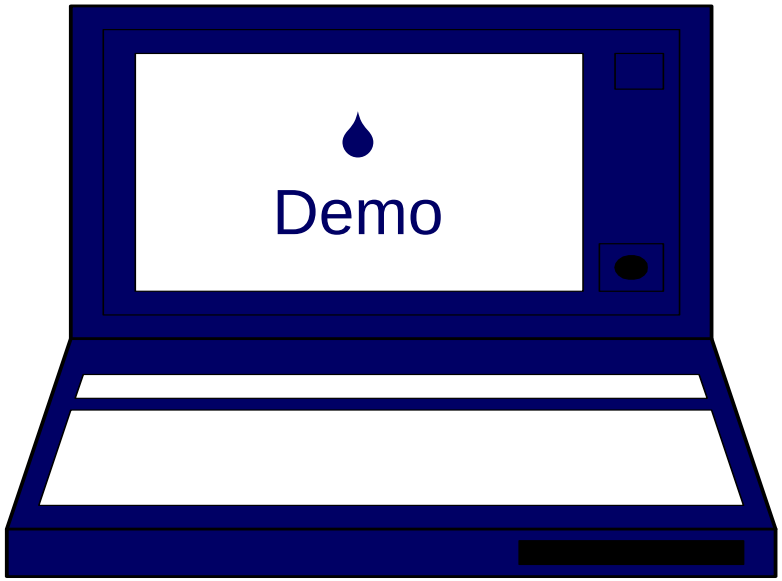
Positive and Negative Testing

- **One definition of Testing is:**
  - "(The verification) that software performs its intended function and does not perform any unintended function" (IEC 61508)
  - These can be called 'positive testing' and 'negative testing' respectively
- **In the context of Ada unit testing here are some aspects of 'negative testing':**
  - Checking unwanted *external calls* are not made
  - Checking unwanted *exceptions* are not raised
  - Checking unwanted memory accesses, including *global data*, are not made
- My aim is to demonstrate the latter point, using new AdaTEST 'Test Support Packages'

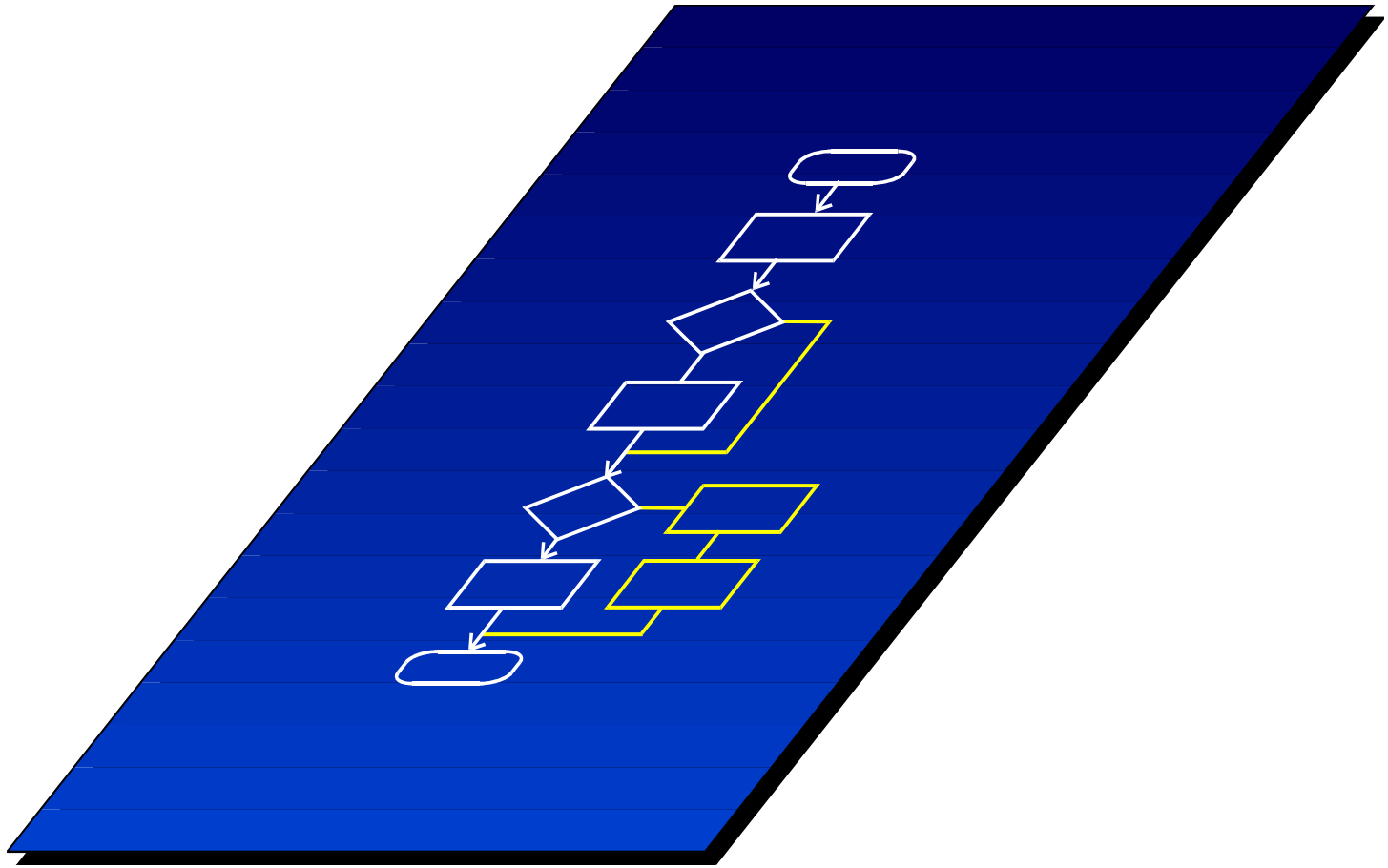
❄ Test Support Packages

- **Test Support Packages (TSPs) provide an option for:**
  - Automating Global Data corruption detection
    - All data in Package Specs
  - Automatic instantiation of Checks for user-defined types
- **Method of use:**
  - Generate TSPs
    - <Package>.TSP or TSP_<Package>
  - Compile in the test lib
  - Use Test Script Generation wizard to incorporate named TSPs in current script

- Dynamic Test for Stack
- With Global Data checking



Coverage Analysis



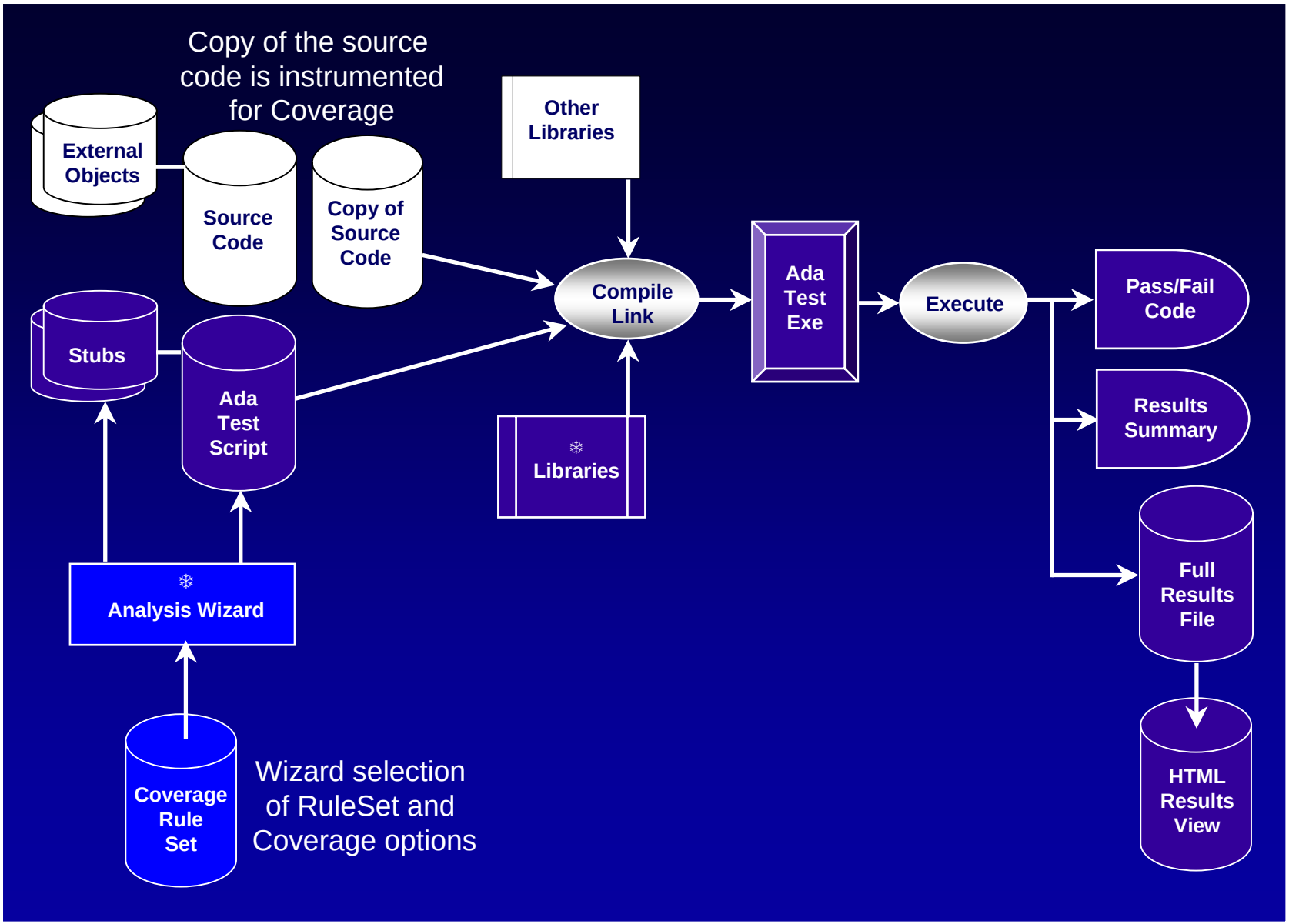
IPL

Coverage Analysis

- ❄provides
  - **Code coverage at levels of**
    - Entry Points
    - Statements (Line)
    - Decisions (Branch)
    - Conditions (Boolean Expression)
    - MC/DC (Masking and Unique-Cause types)
    - Exceptions (handlers & statements)
    - Path Checking
  - **Data Value Coverage**
    - via Assertions

IPL

How Coverage fits in



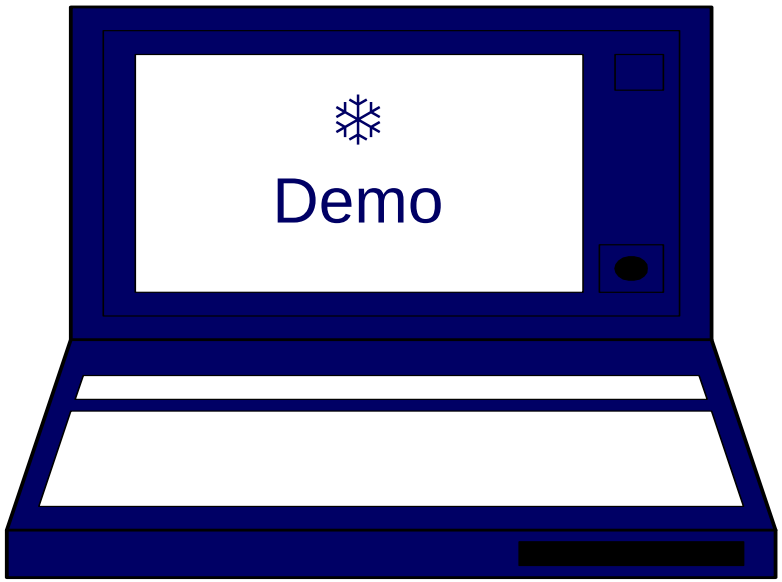
❄ Analysis

- ❄ ‘Rule-sets’ are user-defined.
- Examples:
  - ‘Half of all Statements’
    - Statement_Coverage >= 50%
  - ‘All Branches’
    - Decision_Coverage = 100%
  - ‘DO-178B Level A’
    - Statement_Coverage = 100%
    - Decision_Coverage = 100%
    - Boolean_Operand_Effectiveness = 100%

IPL

Coverage - Demo

- Dynamic Test for package STACK
- With Coverage Analysis
  - 100% Statement Coverage
  - 100% Decision Coverage



IPL

Conclusions

- Standard ‘positive’ testing is always a good thing
  - Find (and remove) bugs
  - Improve reliability etc of code
- ‘Negative’ testing is enhanced testing
  - External calls NOT made if not wanted
  - Exceptions NOT raised if not wanted
  - Global data NOT corrupted
- All of these points supported by ❄
  - Standard usage
  - Test Support Packages

IPL

Want to know more?

- Ask me
  - Ian.gilchrist@iplbath.com
  - Tel: +44-1225-475000
- Speak to our US agents:
  - Mr Scott Thomas, QCS, Portland OR
  - cst@qcsltd.com
  - Tel: 503/646-9991
- Visit IPL website
  - www.iplbath.com

Any questions?

IPL