Software Robustness Testing Service

BALLISTA

http://www.ices.cmu.edu/ballista

Philip Koopman

ECE Department

koopman@cmu.edu - (412) 268-5225 - http://www.ices.cmu.edu/koopman

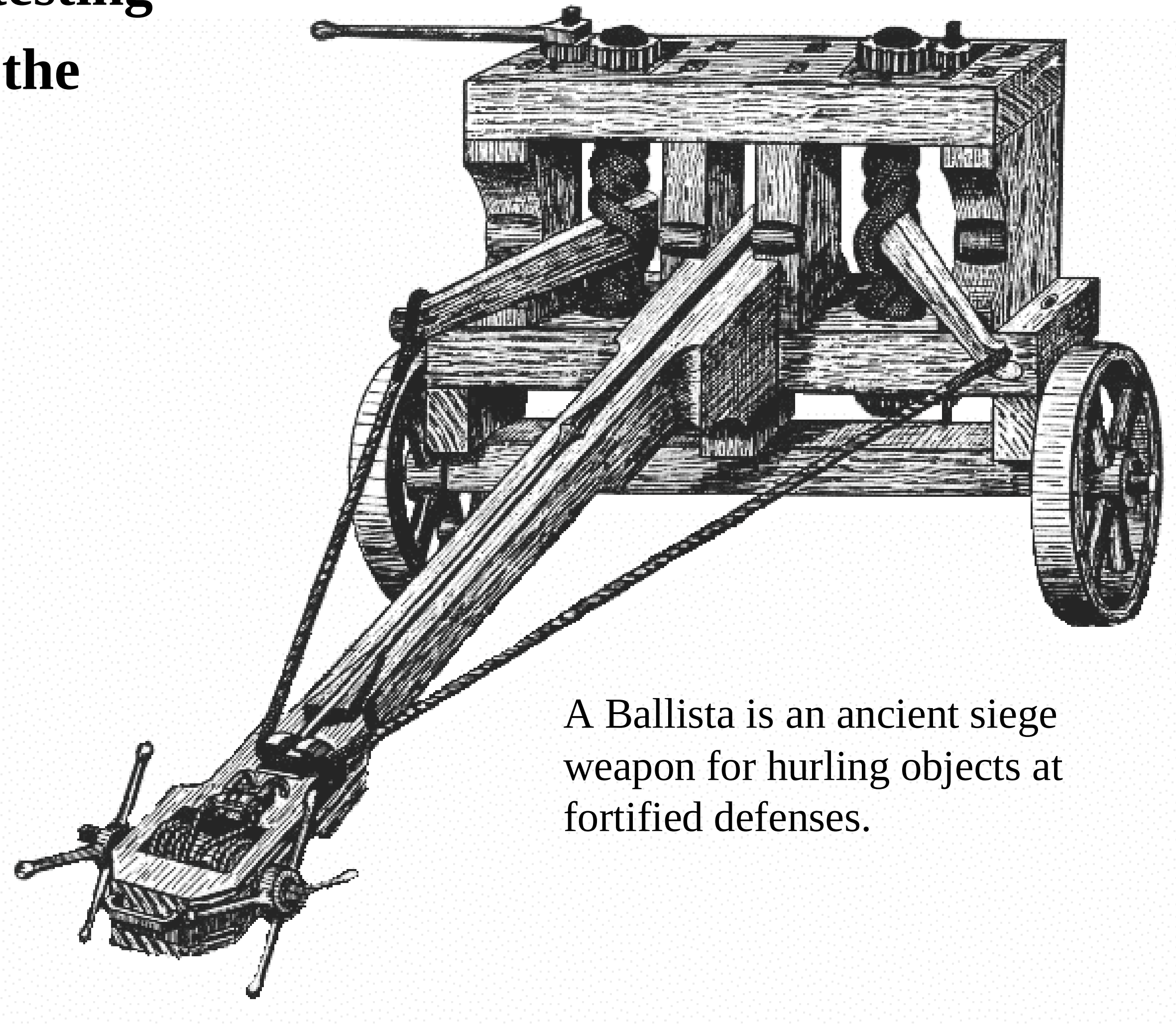
Institute
for Complex
Engineered
Systems

DARPA

Carnegie
Mellon

Overview: Practical Issues in a Testing Service

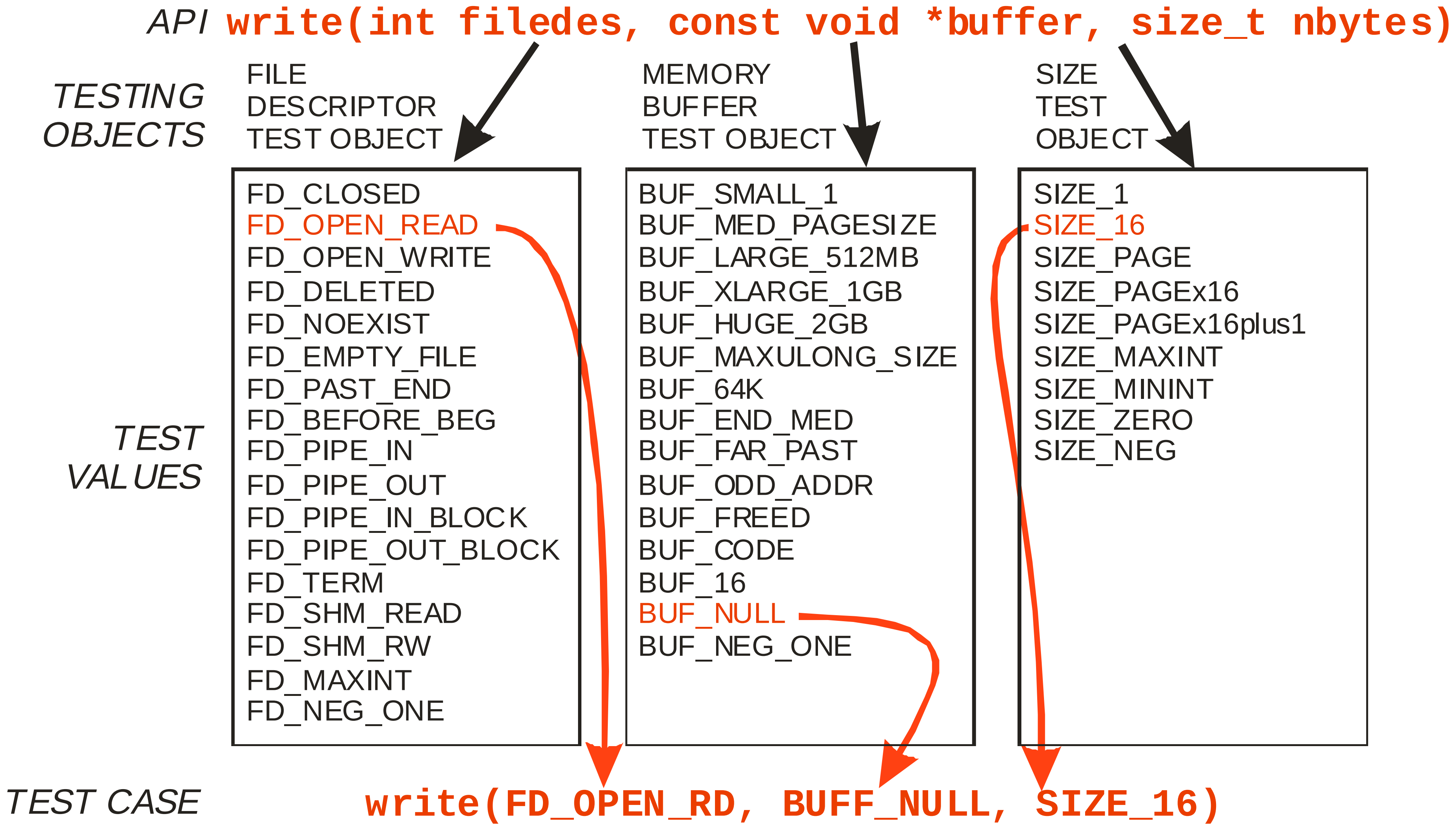
- **Brief review of Ballista testing**
- **Robustness testing over the Internet**
- **Supporting features:**
  - Setting global state
  - Fine-grain test coverage
  - Test scaffolding
  - Legitimate exceptions
- **Future work**
  - What we can do
  - What we can't do



A Ballista is an ancient siege weapon for hurling objects at fortified defenses.

BALLISTA

Object-Oriented Test Generation



API write(int filedes, const void *buffer, size_t nbytes)

TESTING OBJECTS

FILE DESCRIPTOR TEST OBJECT

MEMORY BUFFER TEST OBJECT

SIZE TEST OBJECT

TEST VALUES

FILE DESCRIPTOR TEST OBJECT:

- FD_CLOSED
- FD_OPEN_READ
- FD_OPEN_WRITE
- FD_DELETED
- FD_NOEXIST
- FD_EMPTY_FILE
- FD_PAST_END
- FD_BEFORE_BEG
- FD_PIPE_IN
- FD_PIPE_OUT
- FD_PIPE_IN_BLOCK
- FD_PIPE_OUT_BLOCK
- FD_TERM
- FD_SHM_READ
- FD_SHM_RW
- FD_MAXINT
- FD_NEG_ONE

MEMORY BUFFER TEST OBJECT:

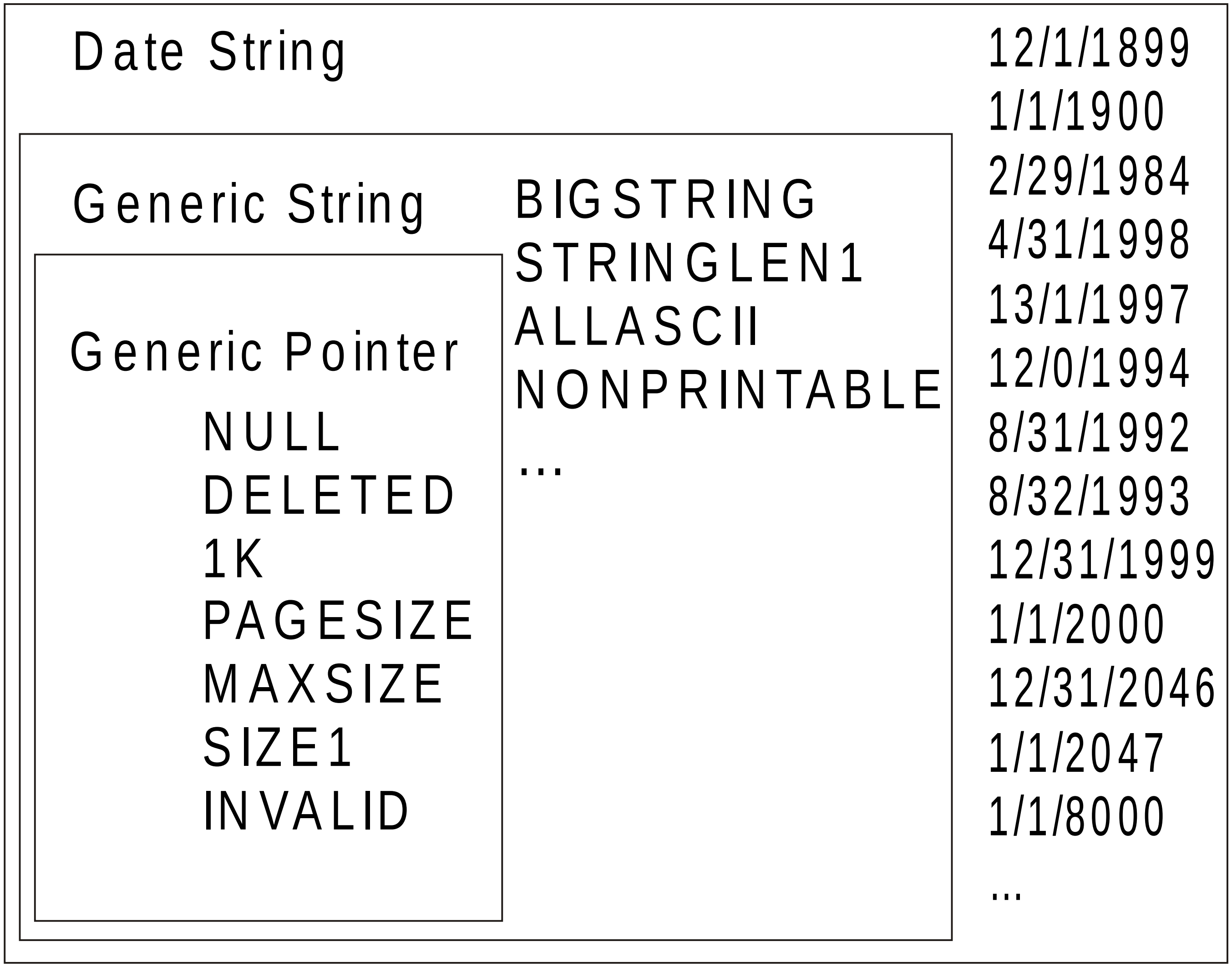
- BUF_SMALL_1
- BUF_MED_PAGESIZE
- BUF_LARGE_512MB
- BUF_XLARGE_1GB
- BUF_HUGE_2GB
- BUF_MAXULONG_SIZE
- BUF_64K
- BUF_END_MED
- BUF_FAR_PAST
- BUF_ODD_ADDR
- BUF_FREED
- BUF_CODE
- BUF_16
- BUF_NULL
- BUF_NEG_ONE

SIZE TEST OBJECT:

- SIZE_1
- SIZE_16
- SIZE_PAGE
- SIZE_PAGEx16
- SIZE_PAGEx16plus1
- SIZE_MAXINT
- SIZE_MININT
- SIZE_ZERO
- SIZE_NEG

TEST CASE write(FD_OPEN_RD, BUFF_NULL, SIZE_16)

Test Value Inheritance



Date String

12/1/1899
1/1/1900
2/29/1984
4/31/1998
13/1/1997
12/0/1994
8/31/1992
8/32/1993
12/31/1999
1/1/2000
12/31/2046
1/1/2047
1/1/8000
...

Generic String

BIGSTRING
STRINGLEN1
ALLASCII
NONPRINTABLE
...

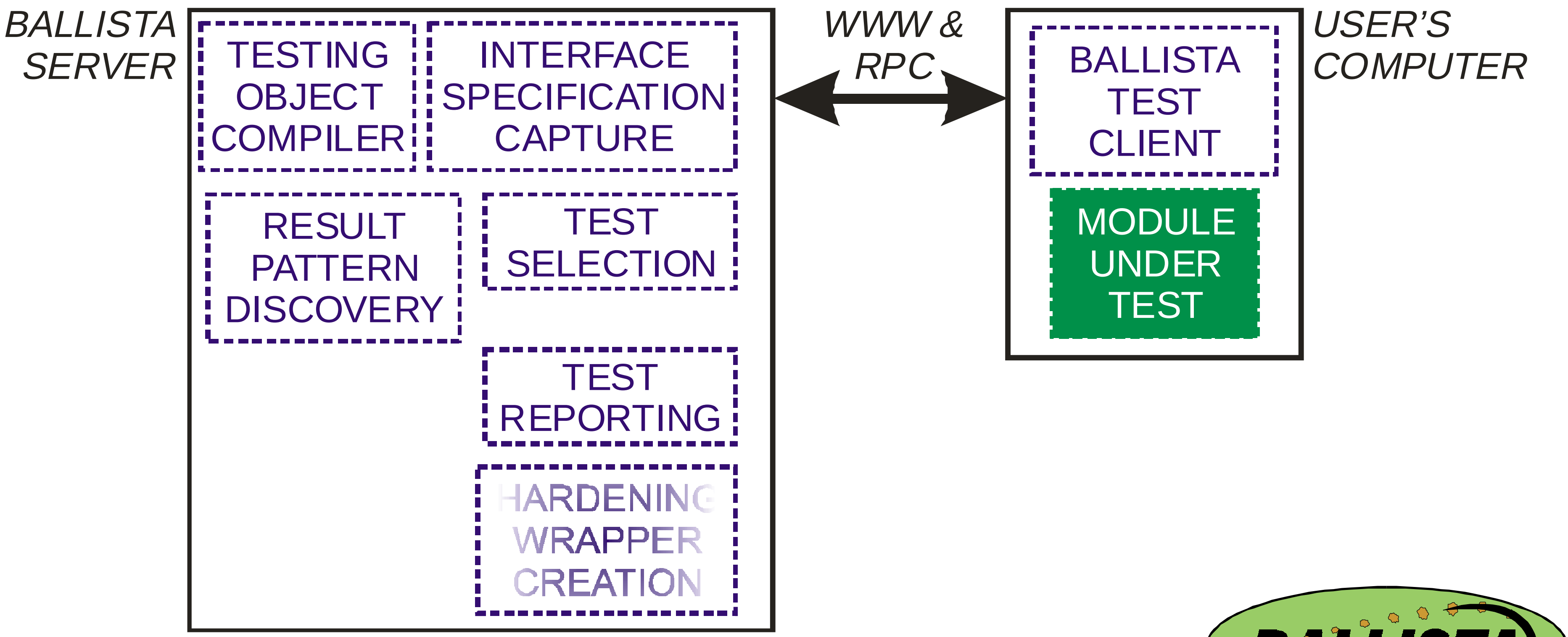
Generic Pointer

NULL
DELETED
1K
PAGESIZE
MAXSIZE
SIZE1
INVALID

Date string inherits test cases from all parents

Robustness Testing Service

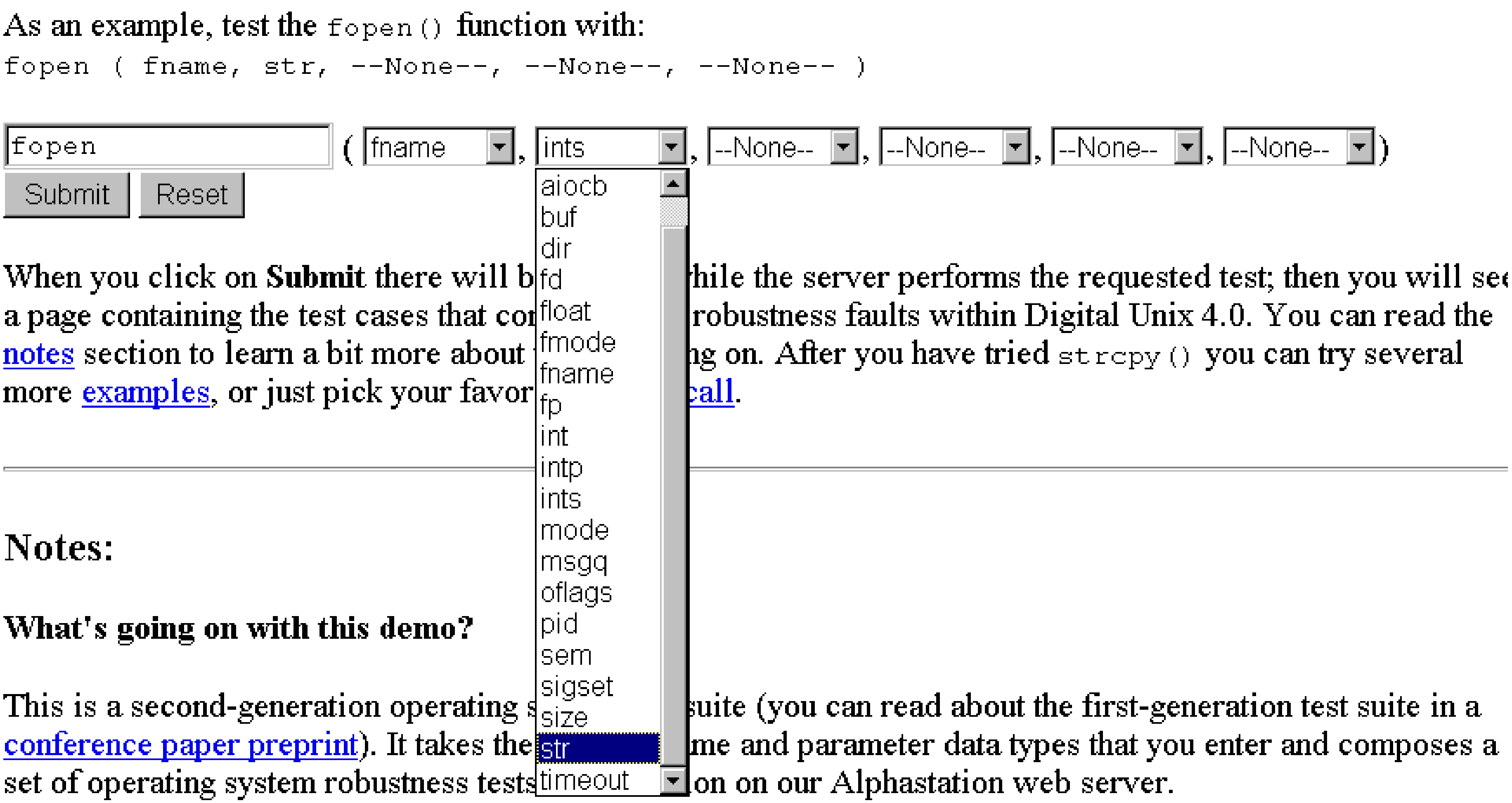
- **Ballista Server**
  - Selects tests
  - Performs pattern Analysis
  - Generates “bug reports”
  - Never sees user’s code
- **Ballista Client**
  - Links to user’s SW under test
  - Can “teach” new data types to server (definition language)



BALLISTA

Specifying the Test

- **Simple demo interface; real interface has a few more steps...**



As an example, test the `fopen()` function with:

```
fopen ( fname, str, --None--, --None--, --None-- )
```

When you click on **Submit** there will b hile the server performs the requested test; then you will se a page containing the test cases that co robustness faults within Digital Unix 4.0. You can read the notes section to learn a bit more about ng on. After you have tried `strcpy()` you can try several more examples, or just pick your favor call.

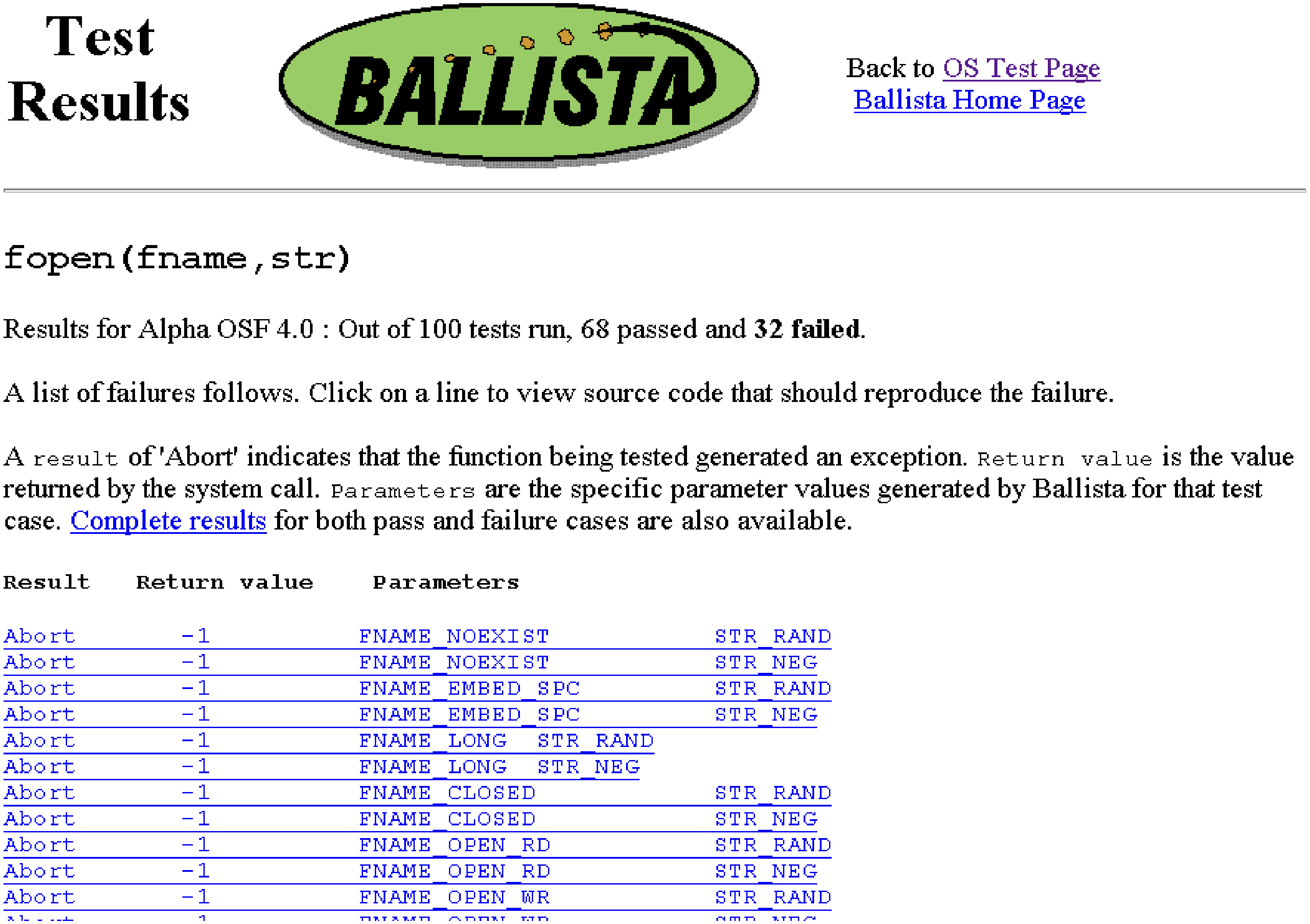
Notes:

What's going on with this demo?

This is a second-generation operating s suite (you can read about the first-generation test suite in a conference paper preprint). It takes the me and parameter data types that you enter and composes a set of operating system robustness tests on on our Alphastation web server.

Viewing Results

- Each robustness failure is one test case (one set of parameters)



Test Results

Back to OS Test Page
Ballista Home Page

fopen(fname,str)

Results for Alpha OSF 4.0 : Out of 100 tests run, 68 passed and **32 failed**.

A list of failures follows. Click on a line to view source code that should reproduce the failure.

A `result` of 'Abort' indicates that the function being tested generated an exception. `Return value` is the value returned by the system call. `Parameters` are the specific parameter values generated by Ballista for that test case. Complete results for both pass and failure cases are also available.

Result	Return value	Parameters	
Abort	-1	FNAME_NOEXIST	STR_RAND
Abort	-1	FNAME_NOEXIST	STR_NEG
Abort	-1	FNAME_EMBED_SPC	STR_RAND
Abort	-1	FNAME_EMBED_SPC	STR_NEG
Abort	-1	FNAME_LONG STR_RAND	
Abort	-1	FNAME_LONG STR_NEG	
Abort	-1	FNAME_CLOSED	STR_RAND
Abort	-1	FNAME_CLOSED	STR_NEG
Abort	-1	FNAME_OPEN_RD	STR_RAND
Abort	-1	FNAME_OPEN_RD	STR_NEG
Abort	-1	FNAME_OPEN_WR	STR_RAND
Abort	-1	FNAME_OPEN_WR	STR_NEG

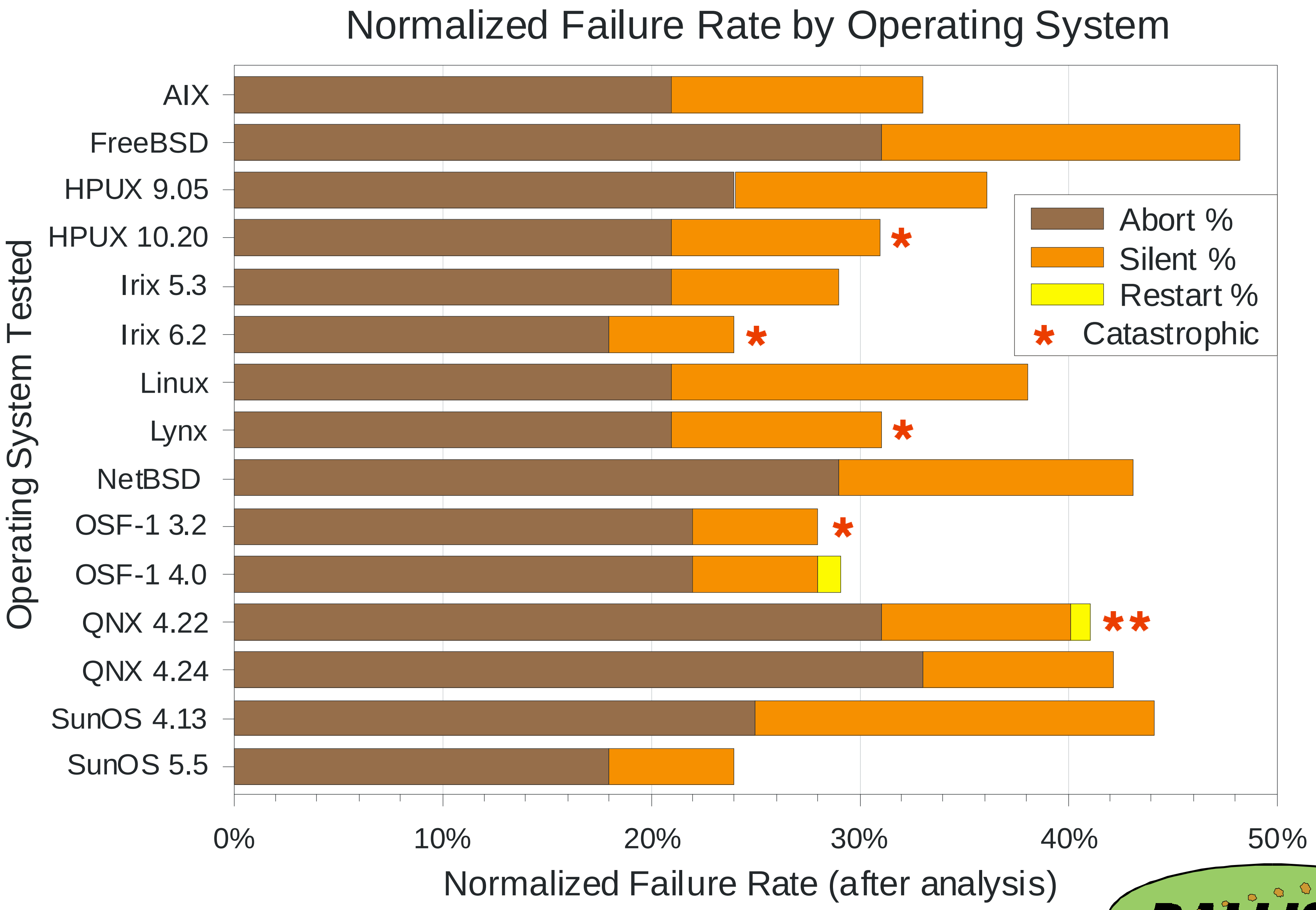
“Bug Report” program creation

- Reproduces failure in isolation (>99% effective)

```
/* Ballista single test case Sun Jun 13 14:11:06 1999
 * fopen(FNAME_NEG, STR_EMPTY) */
...
  const char *str_empty = "";
...
  param0 = (char *) -1;

  str_ptr = (char *) malloc (strlen (str_empty) + 1);
  strcpy (str_ptr, str_empty);
  param1 = str_ptr;
...
  fopen (param0, param1);
```

Estimated Failure Rates After Analysis

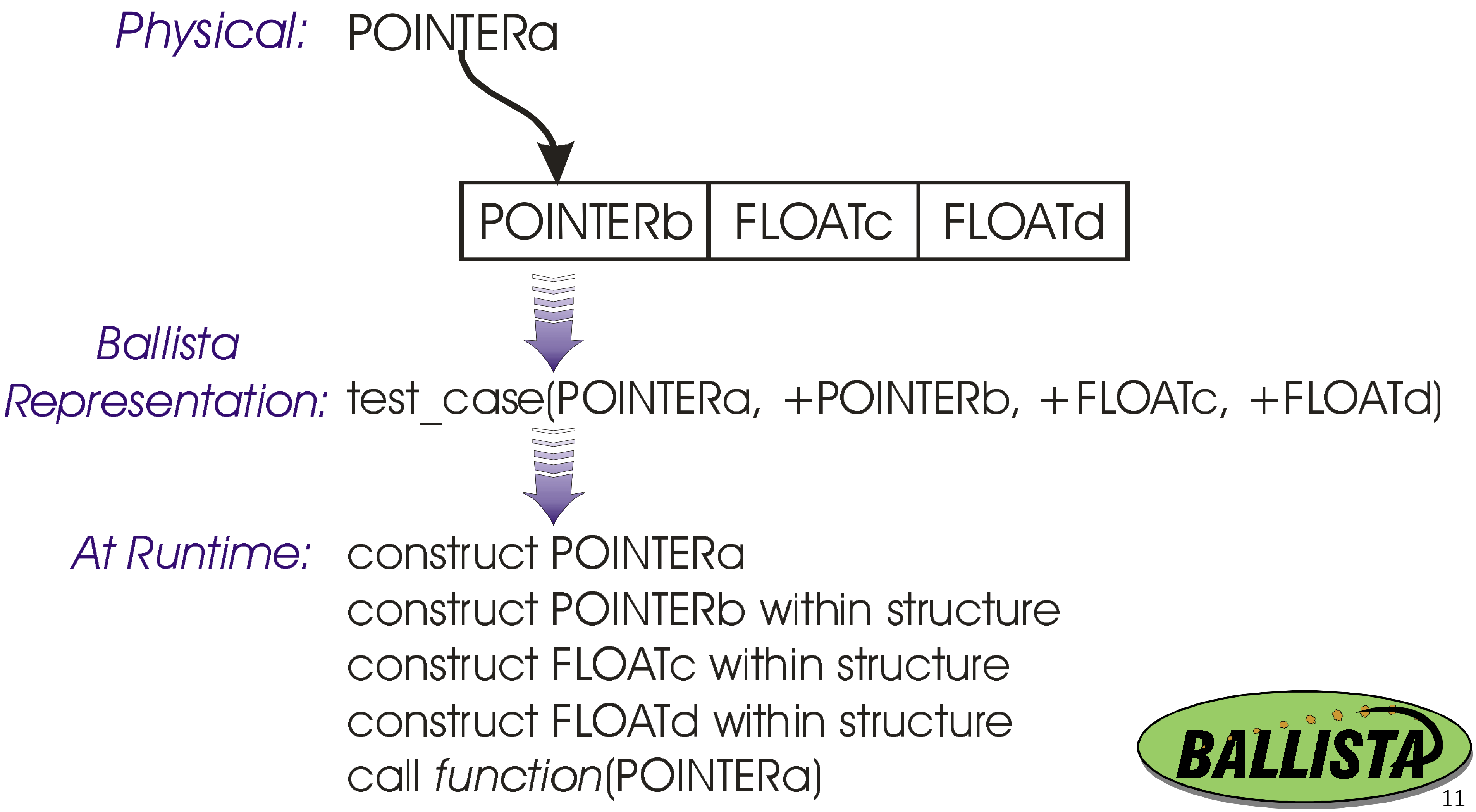


Support Features

- **Test selection / pattern discovery**
  - Randomly selected subset of tests for large testing spaces
  - In future, smarter testing to identify failure-free regions
  - Need fine-grain tests to achieve notion of “adjacent” test cases
- **Data type compiler**
  - Define new testing objects for new data types
  - Want finer grain testing for better testing coverage
  - Want automatic composition of data structures from existing primitives
- **Hardening wrappers**
  - Easy wrappers are easy (*e.g.*, NULL pointer hardening)
  - Hard wrappers get harder the more we think about them

Physical Structures (work in progress)

- **Flatten structure and use existing primitive constructors**
  - Example of single element; linked list of complex numbers



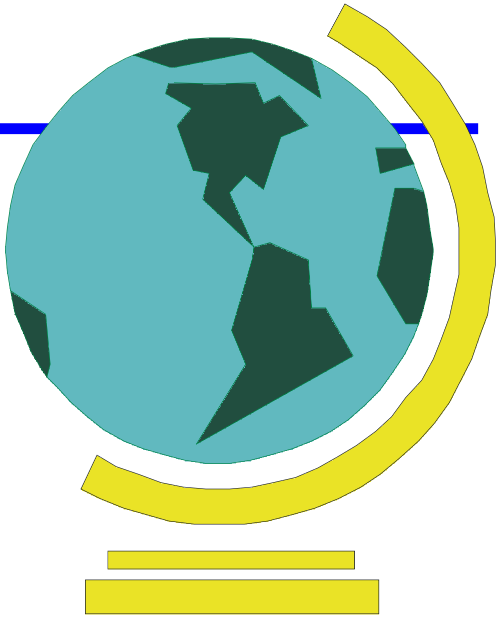
Physical: POINTERa

POINTERb	FLOATc	FLOATd

Ballista Representation: test_case(POINTERa, +POINTERb, +FLOATc, +FLOATd)

At Runtime: construct POINTERa
construct POINTERb within structure
construct FLOATc within structure
construct FLOATd within structure
call *function*(POINTERa)

Setting Global State



- **Use *phantom parameter* idea to set global state**
  - User specifies:

    function(+param0, param1, …)
  - System executes all constructors
  - But, system only passes physical parameters:

    function(param1)

Example:

```
random(+seed_value)
```

establishes a random number seed via a constructor, then calls `random()`

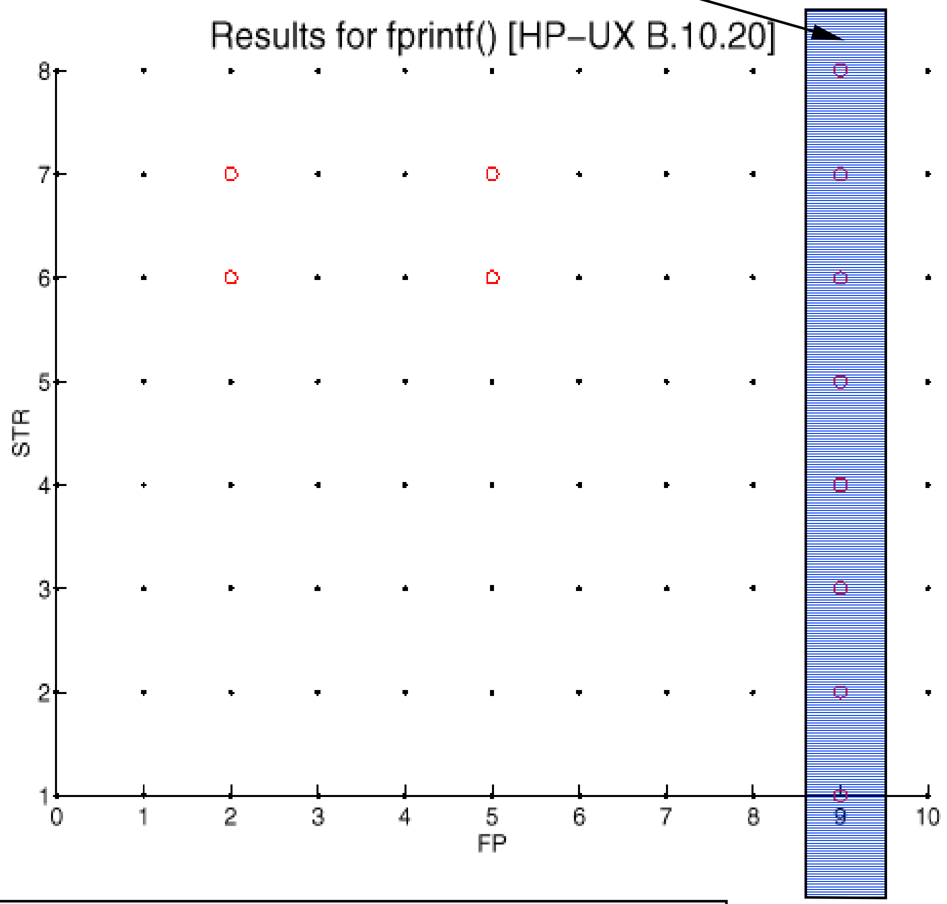
- **Permits setting substantial amount of state using testing objects**
  - Execute test scaffolding (*e.g.*, create federation; join federation)
  - Set global state (*e.g.*, fill up hard disk before file I/O)
  - Set hidden state: (*e.g.*, testing random number generator)

Patterns of Testing Result *(Jiantao Pan's work)*

- **`fprintf(File_Pointer, STRing)` in HP-UX**

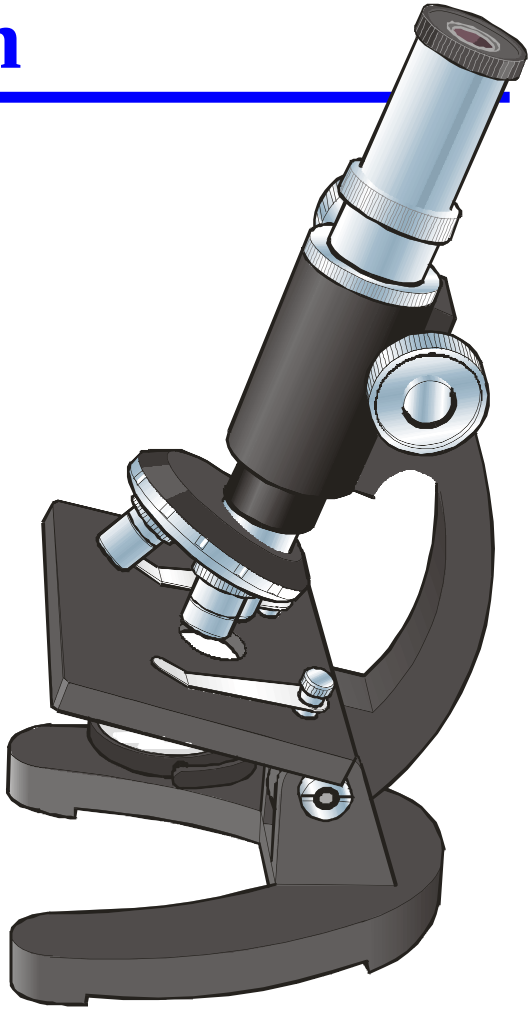
- **1-D failures:**
  - They form a line in a 2-D function (function that parameter dimensionality=2)
  - They form a hyperplane in a n-D function

All 1-D failures this line



● Pass or error code
○ Robustness Failure (Abort/Restart)

Toward Fine-Grain Characterization

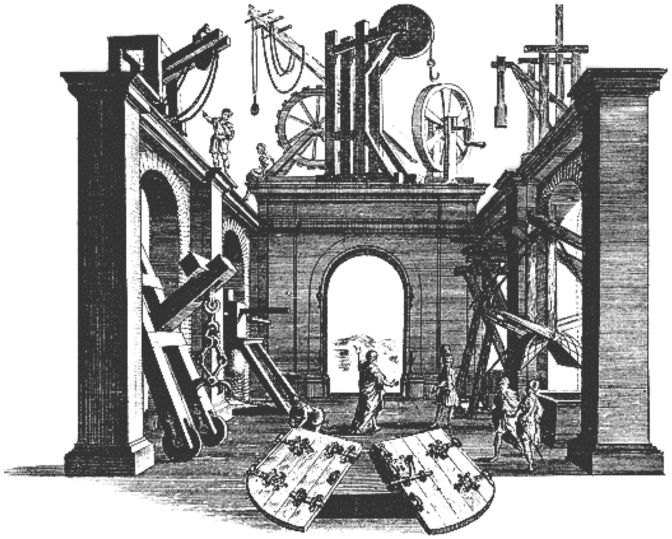


- **Problem: detailed coverage of rich data types (*e.g.*, file handle)**
  - Current tests have large grain size
  - Want tests with high degree of flexibility
  - Want useful notion of "adjacency" in test results

- **Solution: Logical Structs**
  - Decompose data type into *logical* struct of orthogonal sub-types
  - Example for file handle:
    1) File exists, does not exist, deleted after creation
    2) Open for: read, write, r/w, closed
    3) File system permissions for: read, write, r/w, none
    4) File positioned at: beginning, middle, end, past end
    5) ...

BALLISTA

What About Required Scaffolding?



- **Operating system code:**
  - No scaffolding required
  - All durable system state set in constructors / restored by destructors
    - File creation/deletion
    - Process creation/deletion

- **HLA RTI distributed simulation framework:**
  - Requires scaffolding
    - *e.g.*, create Federation, create Federate, join Federation
  - But, not that many distinct scaffolding sets
    - 10 sets of scaffolding for 86 modules
    - Only a few lines of code each
  - Expect to see a similar outcome on many other applications

BALLISTA

What About Different Exception Models?

- **Not all programs use error return codes**
  - What is a “robustness failure” in context of thrown exceptions?
  - But, assume that interface spec. defines all valid exceptions

- **We consider these failures (based on HLA RTI results):**
  - System crashes/hangs = Catastrophic
  - Task hangs = Restart
  - Exception system panic = Abort+
  - “Unknown/default” exception = Abort
  - SIGSEGV (uncaught system exception) = Abort
  - No exception thrown = Silent (difficult to test for)
  - Undocumented exception = Hindering

Future Work

- **Heavy load testing**
  - Resource exhaustion
  - Timing-dependent failures

- **Varied applications**
  - HLA RTI simulation backplane
    - Paper submitted to ISSRE
    - Plans to make Ballista testing part of RTI certification suite
  - Windows (Win32 API)
  - State-intensive object repository for train control (ABB)
  - Factory process control (Emerson)

BALLISTA

What Ballista Does (and Doesn’t Do)

- **Quantification of exception handling robustness**
  - Scalable, inexpensive compared to traditional testing approaches
  - Makes a contribution toward the ~80% of code for exception handling
  - In the future, will include heavy-load testing
  - But, any such metric is difficult to relate to an operational profile

- **Currently, uses heuristic tests**
  - Fine grain searching will enable use of adaptive testing + search methods

- **Easier than it appears to test some system state**
  - Small amounts of system state in parameter-based tests
  - Larger system state possible using phantom parameters
  - But, will it work on a database-like system? (we’ll find out…)

Other Potential Uses

- **Best used as a QA technique**
  - *Quality must be designed in, not tested in*

- **Perhaps extend to light-weight correctness testing**
  - Dynamic tension between scalability and specificity
  - Can other behaviors be represented with a simple oracle?
    - Memory consumption
    - Touching (or not touching) safety critical objects

- **High-level security check**
  - Buffer over-run testing
  - Detect touching non-permissible items (*e.g.*, security logs)

- **Potentially useful as a metric for diversity**