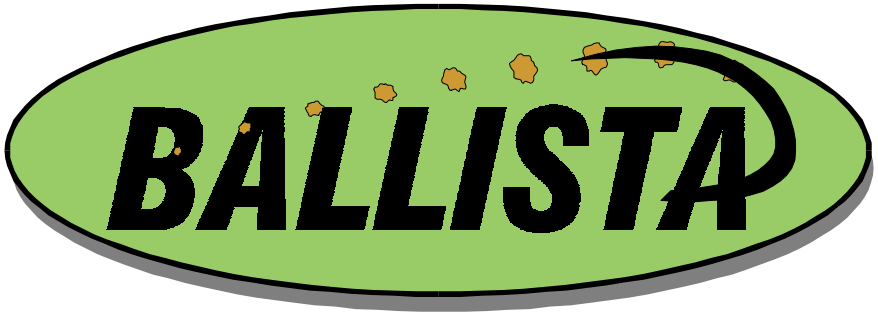
Comparing the Robustness of POSIX Operating Systems

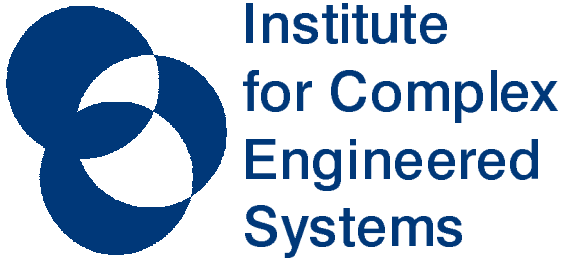


http://www.ices.cmu.edu/ballista

Philip Koopman & John DeVale

ECE Department

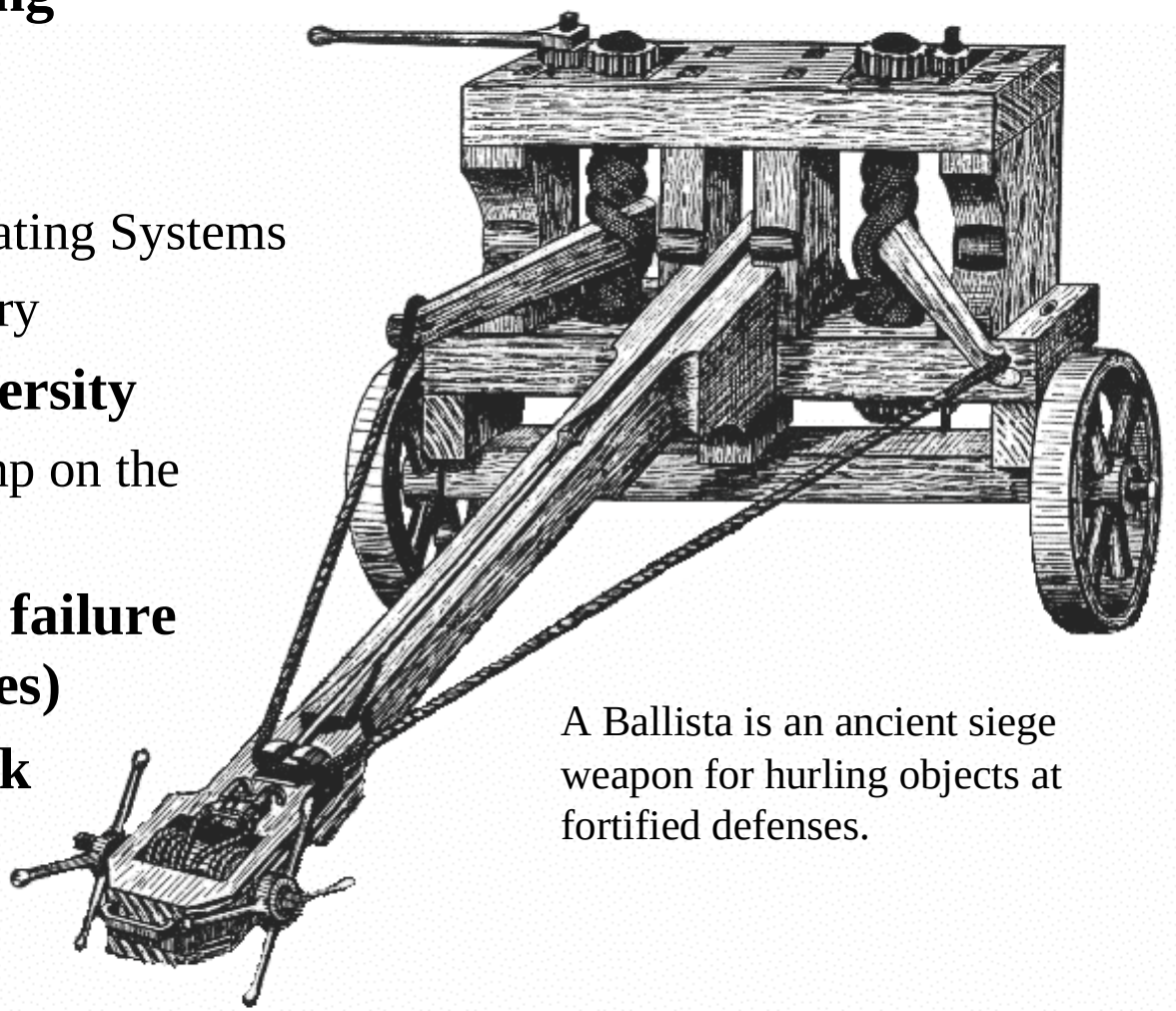
koopman@cmu.edu - (412) 268-5225 - http://www.ices.cmu.edu/koopman



Carnegie Mellon

Overview: Ballista Automated Robustness Testing

- **Generic robustness testing**
  - Based on data types
- **OS Testing results**
  - Raw results for 15 Operating Systems
  - System calls vs. C Library
- **Exception Handling Diversity**
  - Does everyone core dump on the *same* exceptions? (no)
- **Approximating "Silent" failure rates (missing error codes)**
- **Conclusions/Future work**



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

BALLISTA

Ballista: Software Testing + Fault Injection Ideas

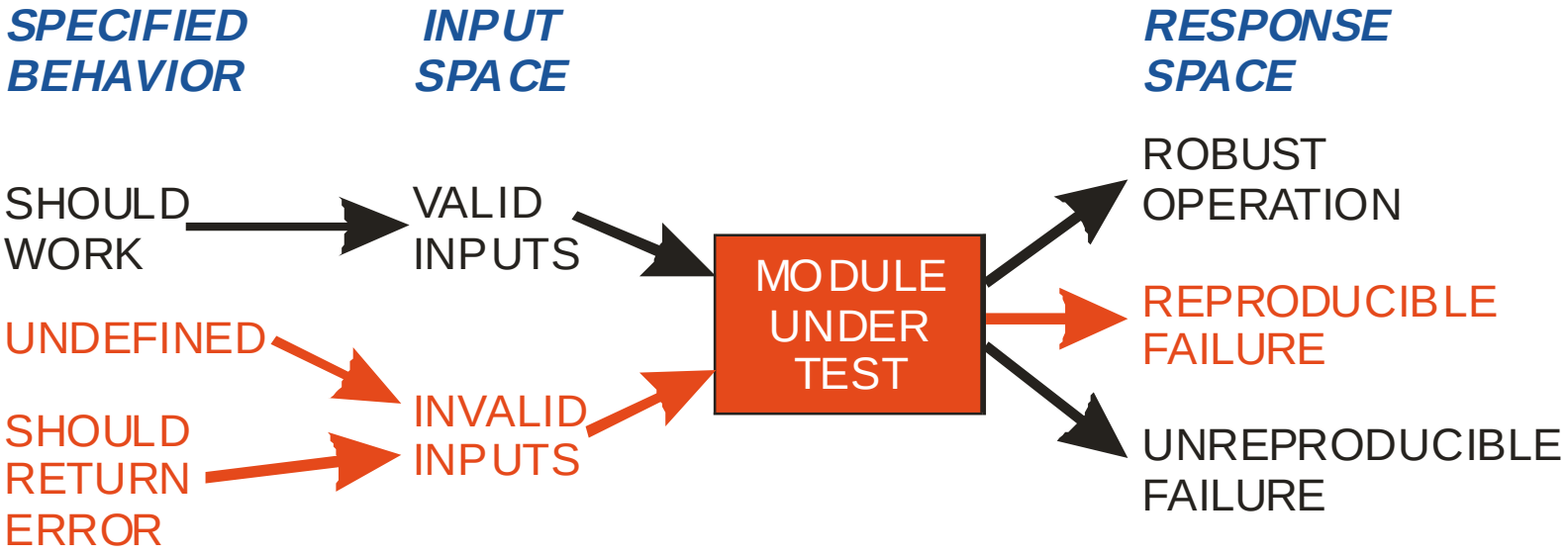
- **SW Testing requires:**
  - Test case
  - Module under test
  - *Oracle* (a "specification")

Ballista uses:

"Bad" value combinations

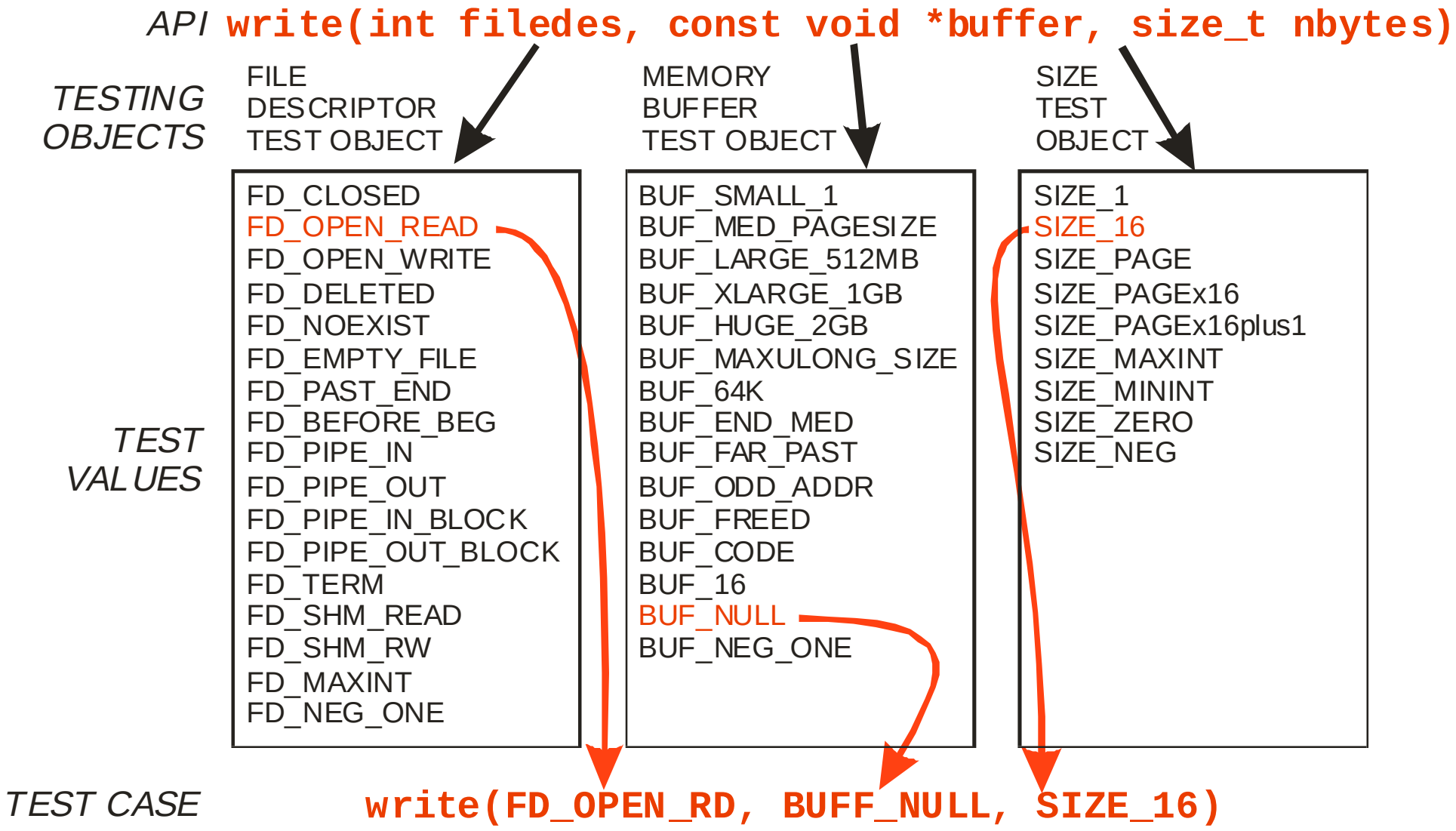
Module under Test

Watchdog timer/core dumps



- **Ballista combines ideas from:**
  - Domain testing ideas / Syntax testing ideas
  - Fault injection at the API level

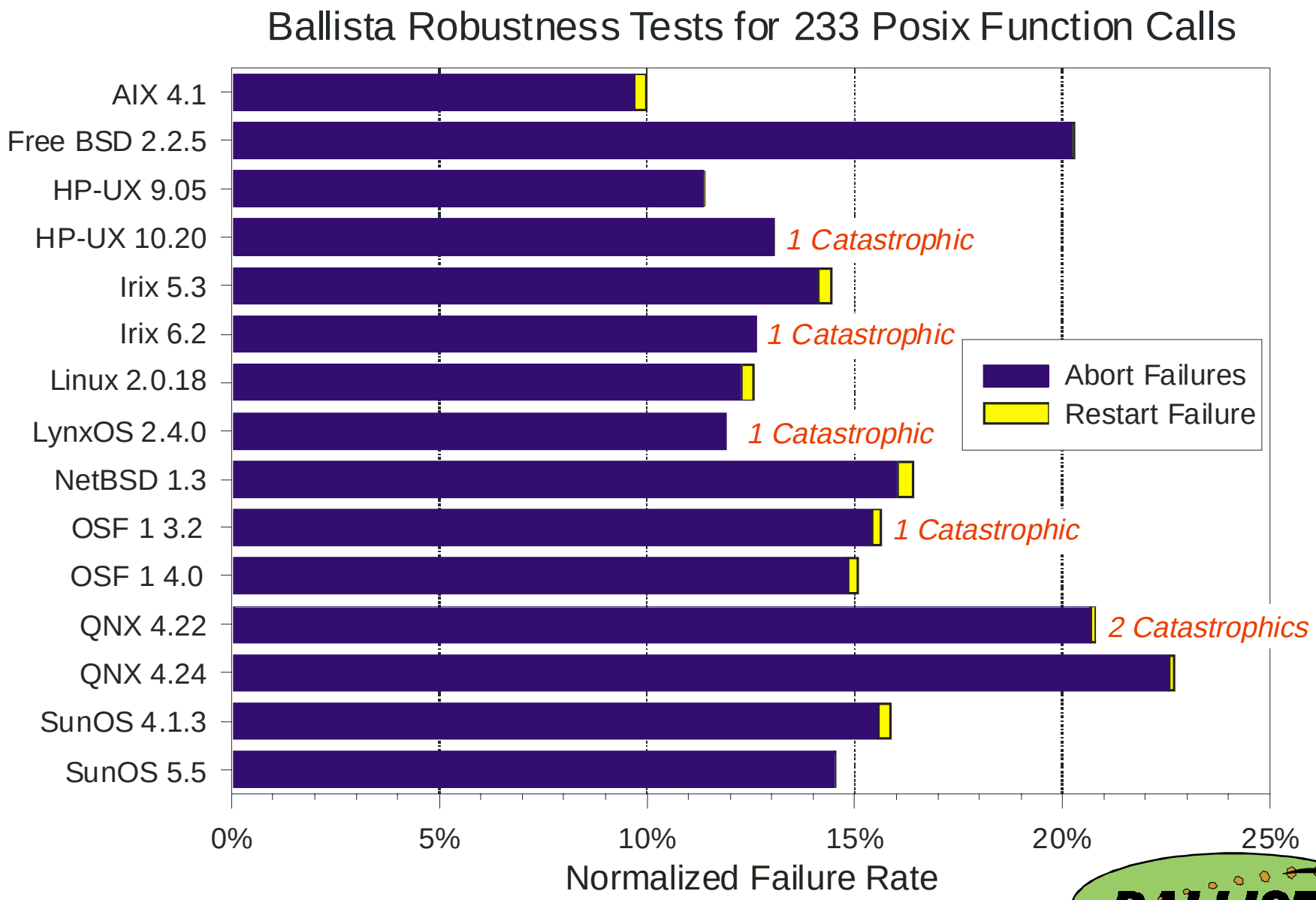
Scalable Test Generation



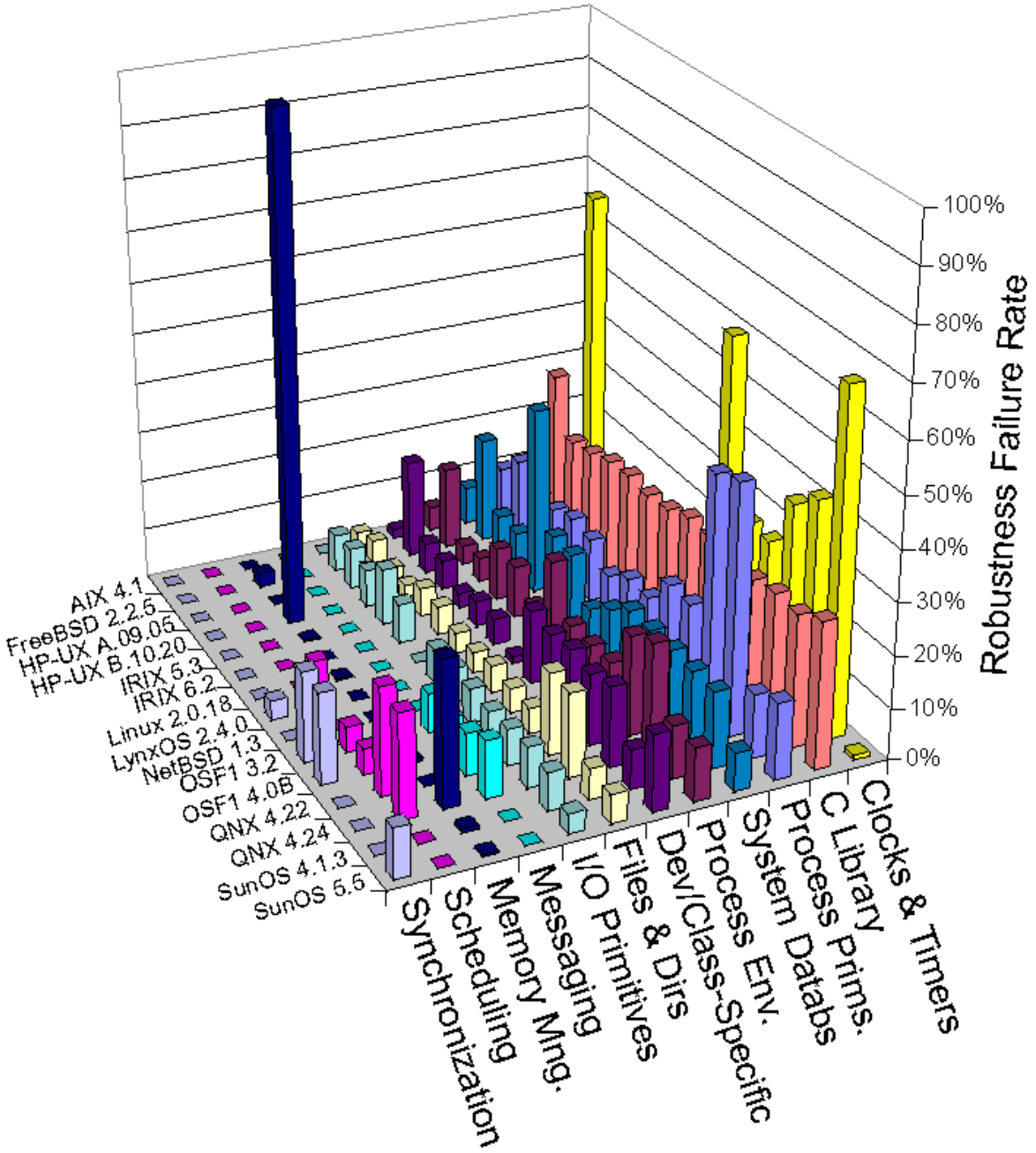
CRASH Severity Scale

- **Catastrophic**
  - Test computer crashes (both Benchmark and Starter abort or hang)
  - Irix 6.2: `munmap( malloc((1<<30)+1), ((1<<31)-1)) );`
- **Restart**
  - Benchmark process hangs, requiring restart
- **Abort**
  - Benchmark process aborts (*e.g.*, “core dump”)
- **Silent**
  - No error code generated, when one should have been (*e.g.*, de-referencing null pointer produces no error)
- **Hindering**
  - Incorrect error code generated

Comparing Fifteen Operating Systems

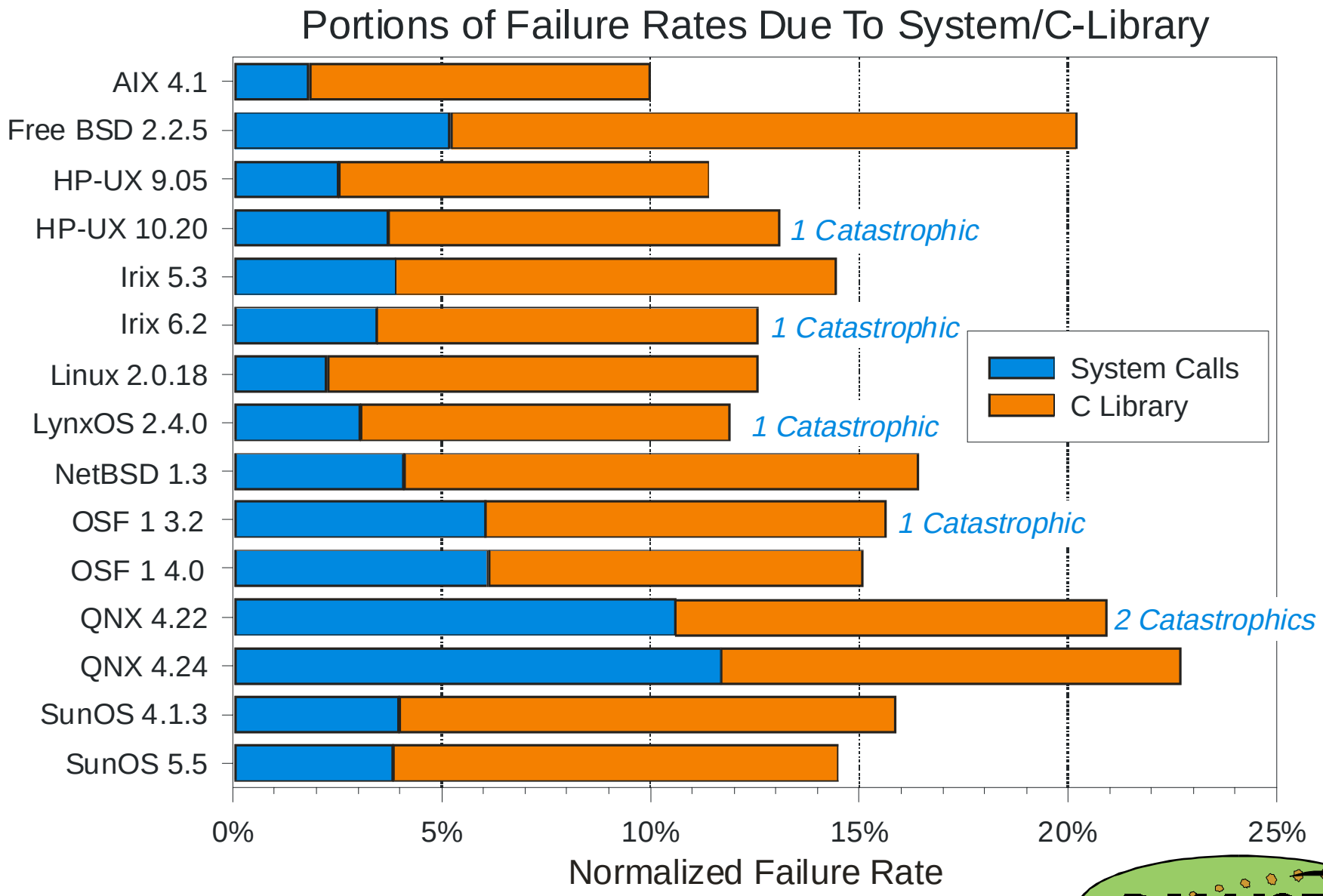


Failure Rates By POSIX Fn/Call Category



BALLISTA

C Library Is A Potential "Robustness Bottleneck"



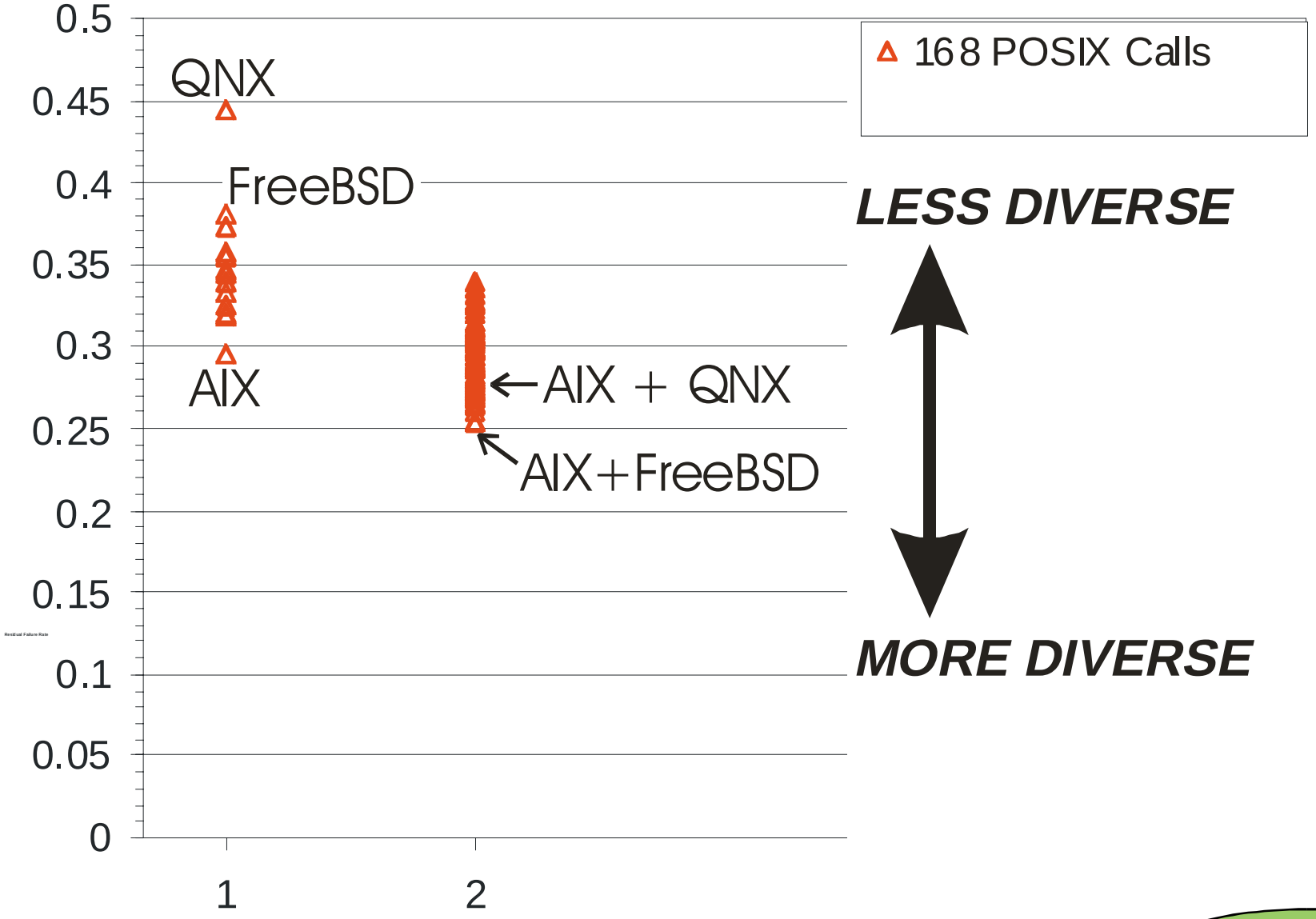
Common Failure Sources

- **Based on correlation of failures to data values, not traced to causality in code**

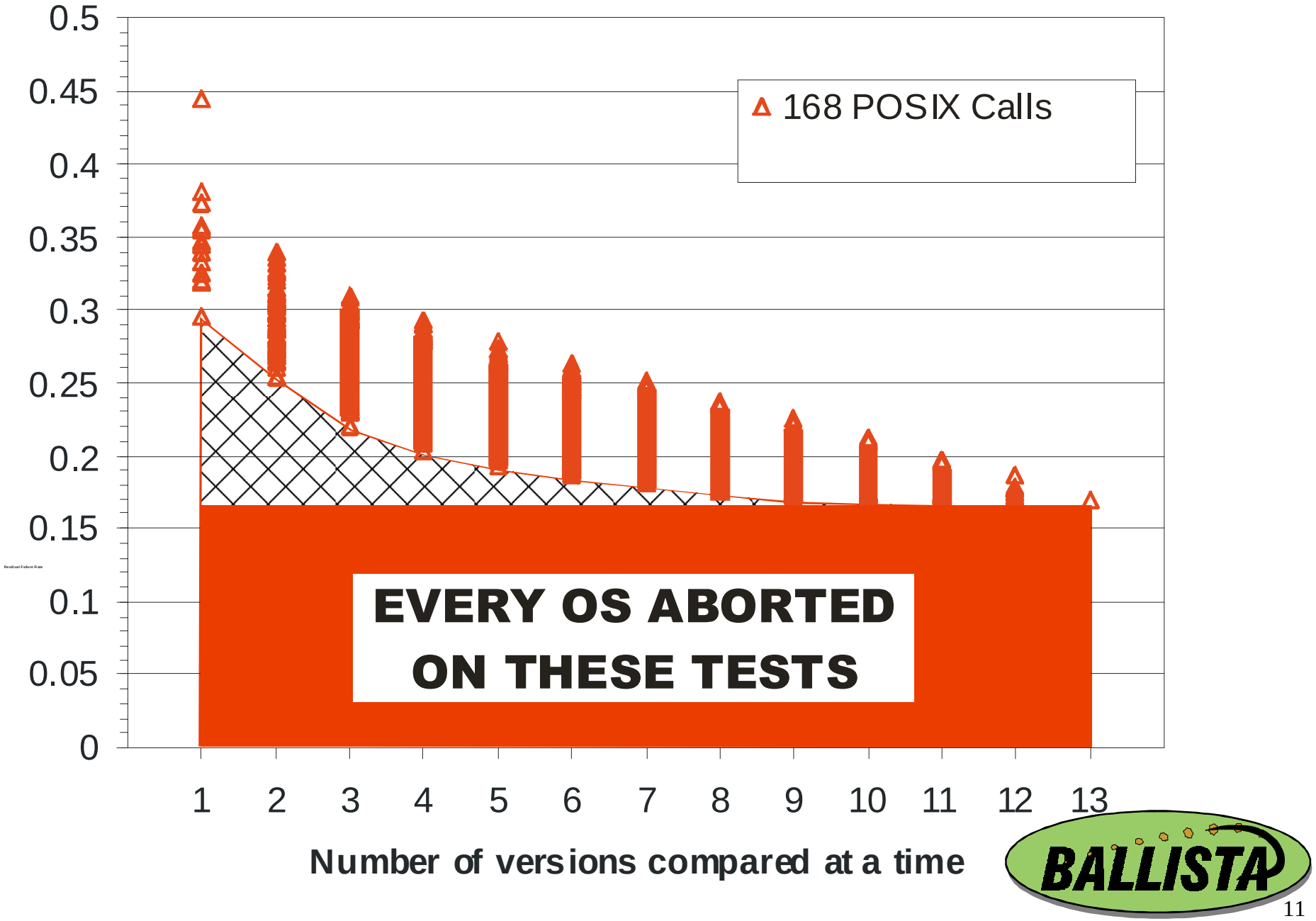
- **Associated with a robustness failure were:**
  - 94.0% of invalid file pointers (excluding NULL)
  - 82.5% of NULL file pointers
  - 49.8% of invalid buffer pointers (excluding NULL)
  - 46.0% of NULL buffer pointers
  - 44.3% of MININT integer values
  - 36.3% of MAXINT integer values

- **Operational profile results vary depending on workload**
  - IBS benchmarks: 19% to 29% weighted average failure rate
  - SPEC floating point less than 1% weighted average failure rate

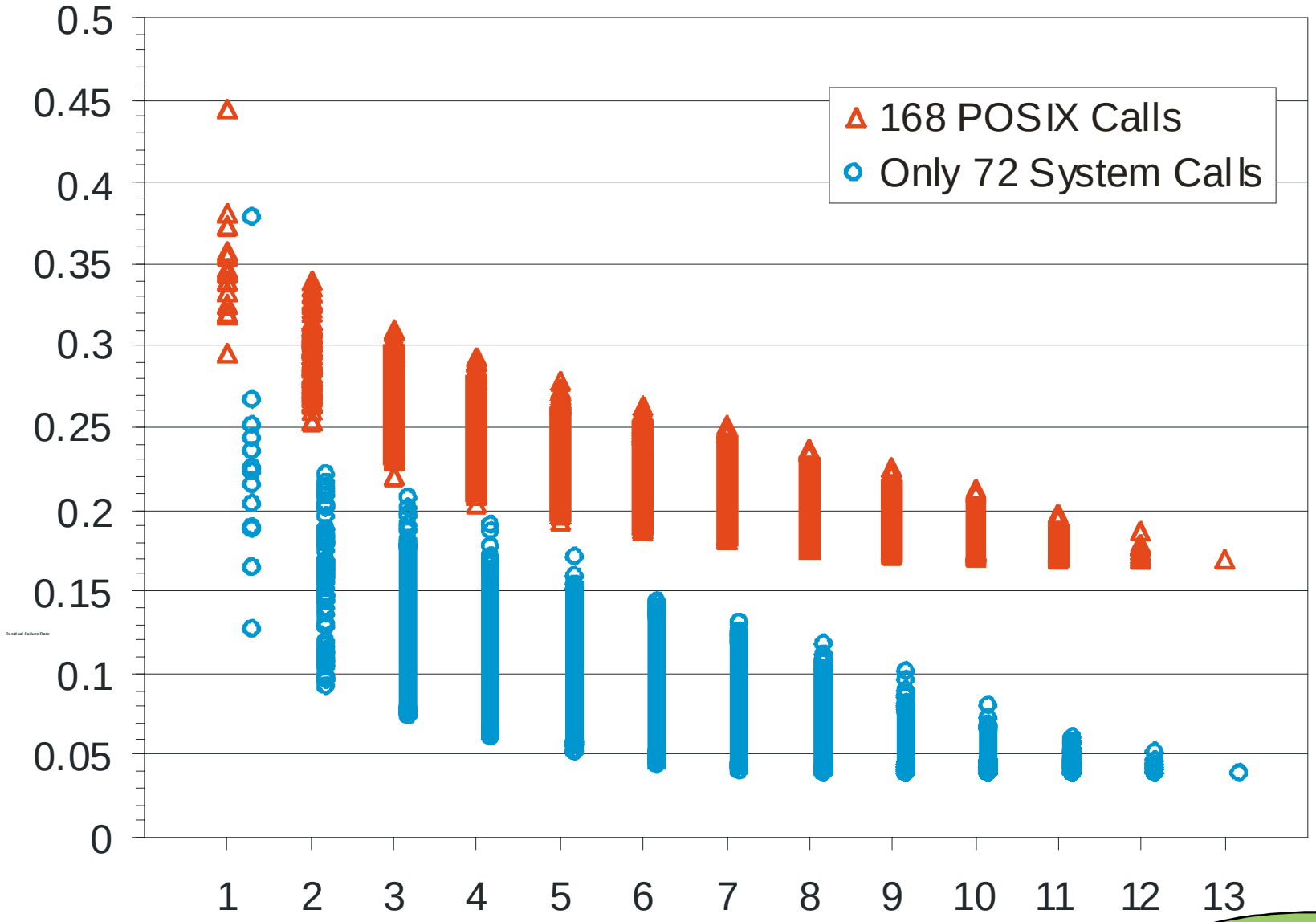
Does Everyone Abort on the Same Things?



17% (Normalized) Common Mode Aborts



Most System Call Aborts Potentially Avoidable

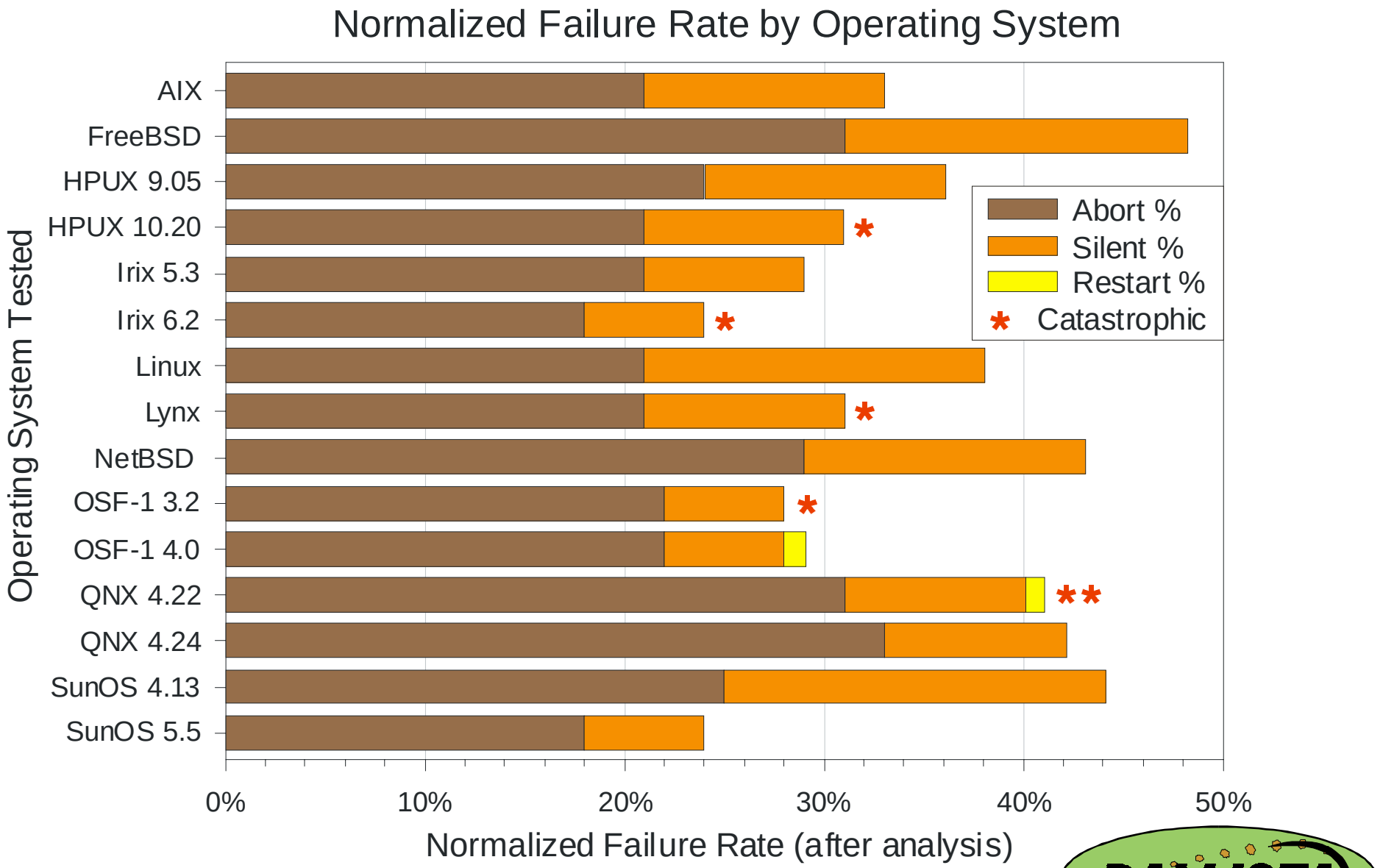


Data Analysis Using N-Version Detection

- **Use N-version software voting to refine data (and use manual sampling to check effectiveness)**
  - Eliminate non-exceptional tests -- **12% of data**; method ~100% accurate
    - *e.g.*, reading from read-only file
  - Identify Silent failures

- **Silent failures -- 6% to 17% additional robustness failure rate**
  - 80% accurate when one OS reports "OK" while at least one other OS reports an error code
    - ~2% were bugs involving failure to write past end of file
    - 28% of remainder due when POSIX permits either case
    - 30% of remainder due to false alarm error codes (many in QNX)
    - ~40% of remainder just out of scope of POSIX standard
  - 50% accurate when one OS reports "OK" but another OS dumps core
    - Half of remainder due to order in which parameters are checked
    - Half of remainder due to FreeBSD floating point library Abort failures (*e.g.*, fabs(DBL_MAX))

Estimated Failure Rates After Analysis

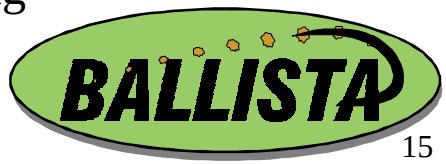


Is Dumping Core The "Right Thing?"

- **AIX has only 10% raw Abort failure rate -- on purpose**
  - Wish to avoid Abort failures in production code
  - Ignores some NULL pointer reads by setting page 0 to read permission
  - *BUT -- 21% adjusted Abort failure rate*; 12% Silent failure rate

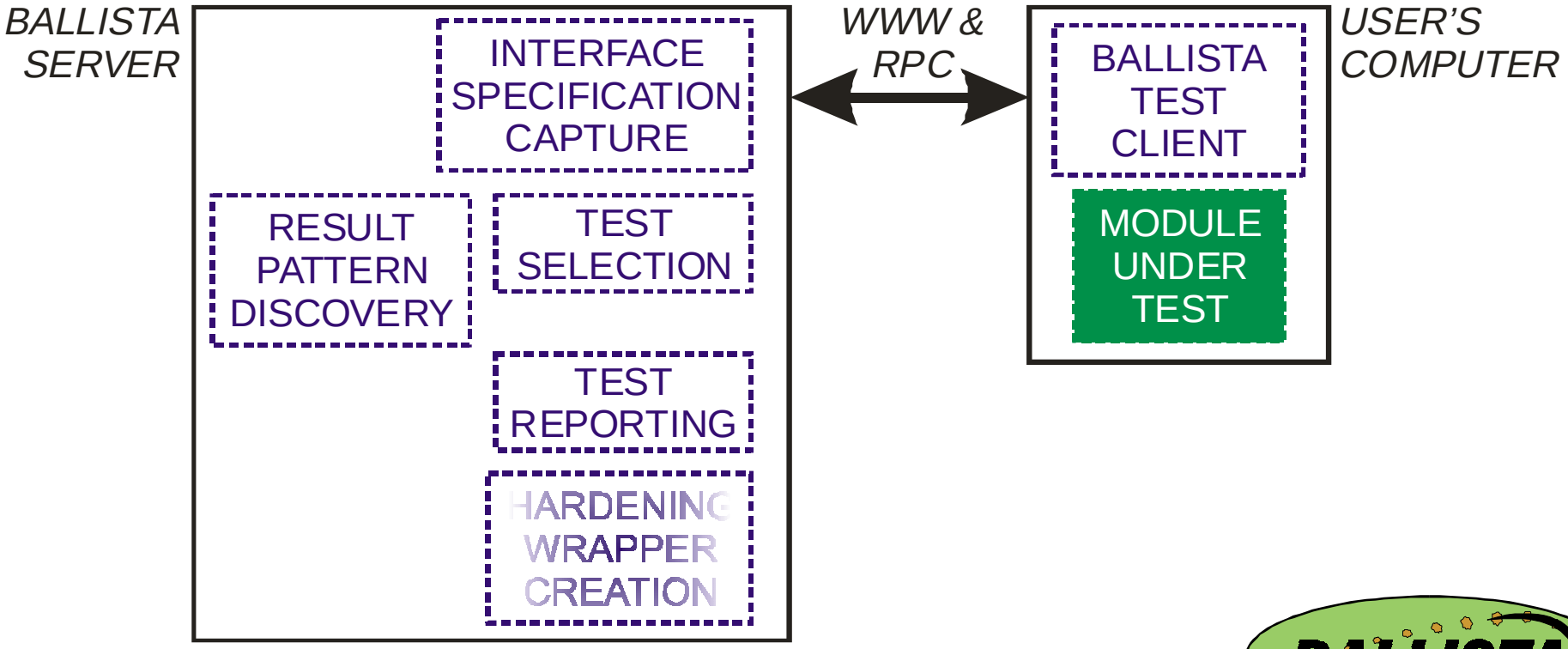
- **FreeBSD has 20% raw Abort failure rate -- on purpose**
  - Intentionally aborts to flag bugs during development cycle
  - 31% adjusted Abort failure rate; *BUT -- 17% adjusted Silent failure rate*

- **Future challenges:**
  - Flag defects during development
    - Boundschecker-like systems need a workload to find problems
  - And still tolerate robustness problems once system is fielded
    - Truly Portable exception handling for POSIX API
    - Perhaps wrappers to manage complexity of exception handling (*e.g.*, Bell Labs XEPT work)



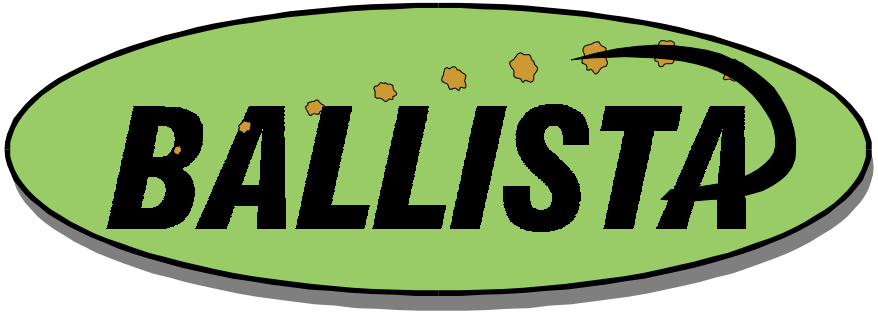
Next Step: Robustness Testing Service

- **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)



Wrap-up

- **"Lofty Goal:" harden legacy and COTS software components**
  - For mission-critical systems
    Without extensive re-engineering to improve robustness
- **Robustness metric for Operating Systems**
  - Failure rates look high; true impact depends on operational profile
  - Controversy as to whether Abort failures are OK
  - Metrics help stimulate demand for improvement
- **Ballista robustness testing approach**
  - Scalable, portable, reproducible
  - C library has higher failure rate, less diverse than OS system calls
  - Currently available as web server; applying to several domains
- **Future: Windows NT, more system state, heavy system loads**



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