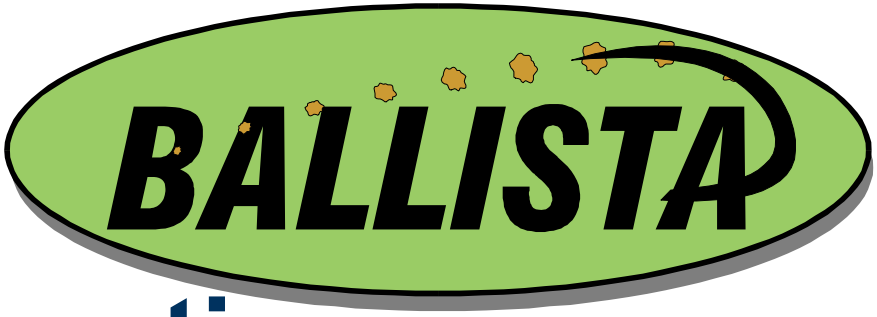
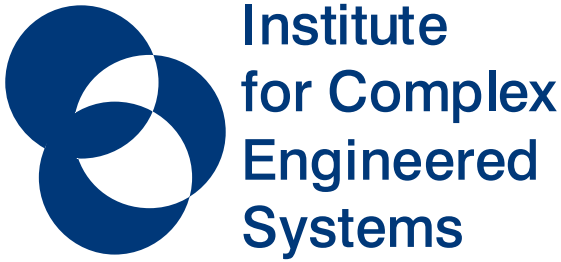


Performance evaluation of exception handling in I/O libraries

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

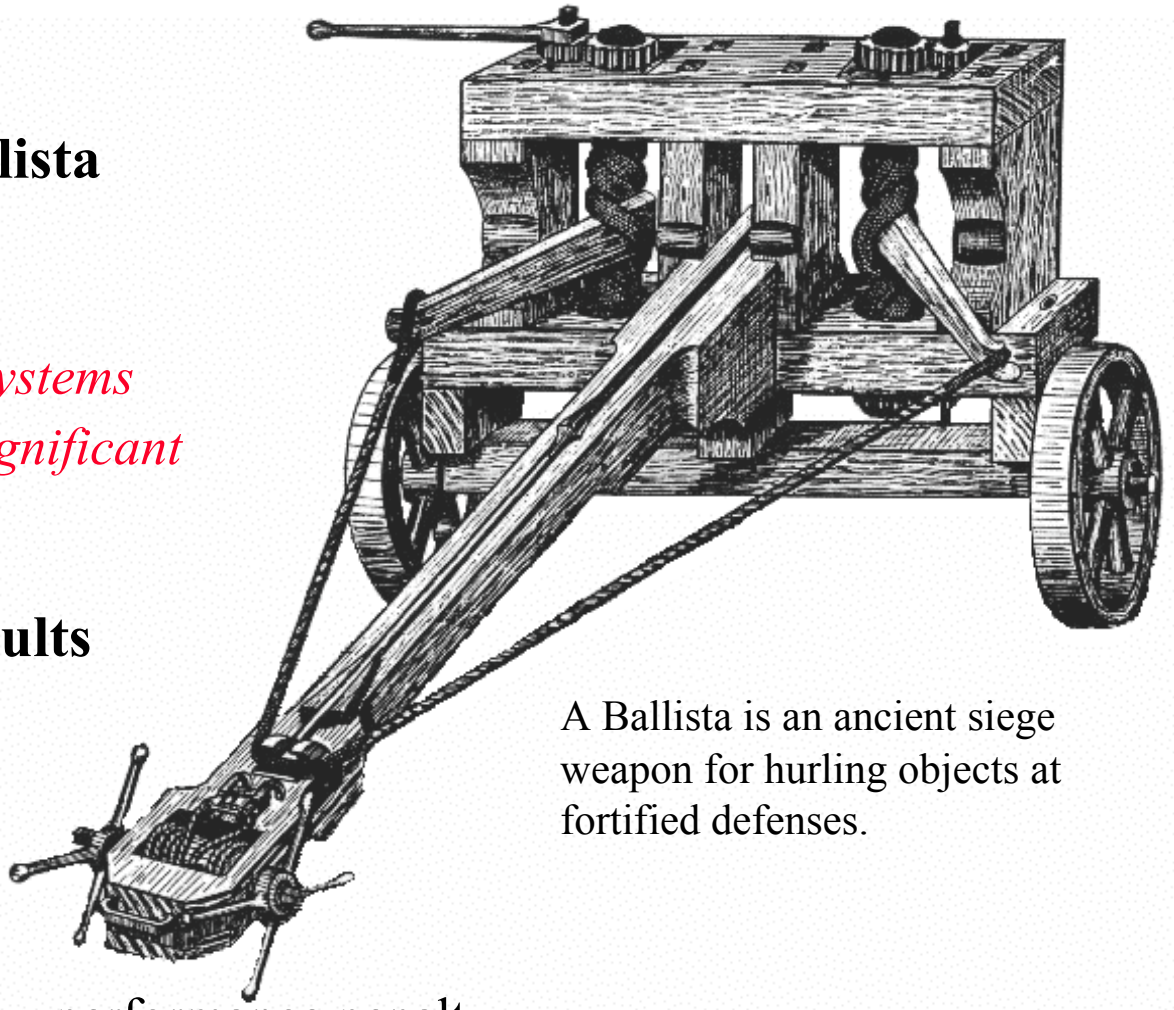
John P. DeVale

devale@cmu.edu - (412) 268-4264 - http://www.ece.cmu.edu/~jdevale



Overview

- **General overview of Ballista**
- **Hypothesis**
  - *We can make software systems extremely robust with no significant performance penalty*
- **Experimental Setup/Results**
  - SFIO
- **Conclusions**
  - High robustness with Low performance penalty



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

BALLISTA

Overview

System Robustness -- Improves Dependability

- **Graceful behavior in the presence of exceptional conditions**
  - Unexpected operating conditions
  - Activation of latent design defects

- **Research Goal**
  - *Metric for comparative evaluation of software robustness*
  - *Ability to apply metric results in a consistent fashion to improve robustness*
  - *Structure exception handling code to specifically leverage hardware performance features and minimize performance impact*

Ballista Software Testing Heritage

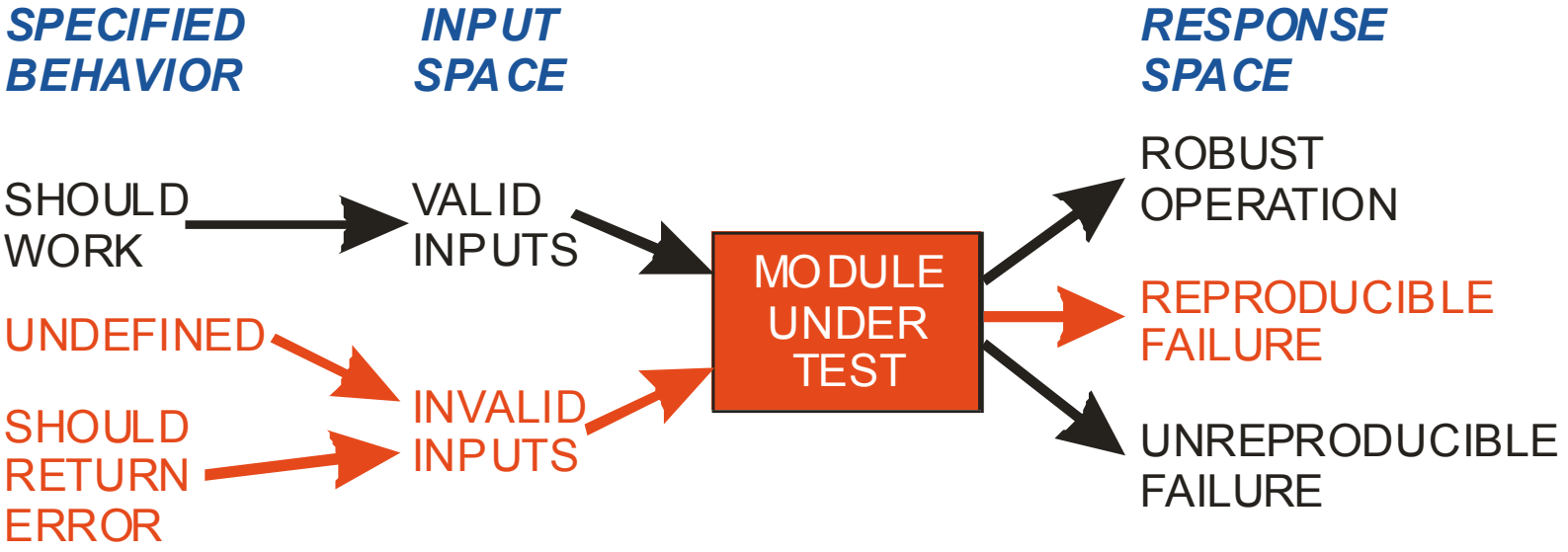
- **SW Testing requires:**
  - Test case
  - Module under test
  - *Oracle* (a "specification")

Ballista uses:

"Bad" value combinations

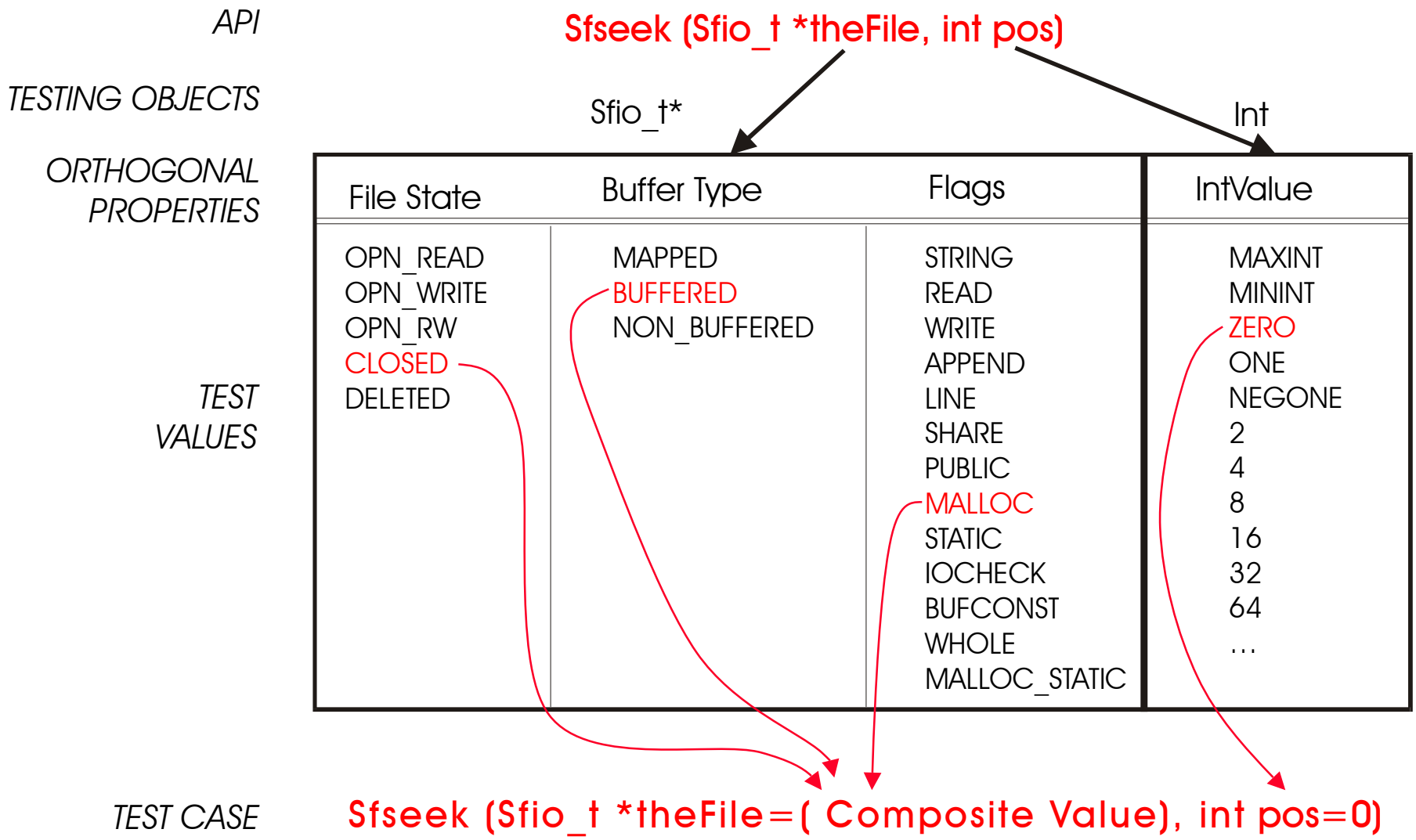
Module under Test

Watchdog timer/core dumps



- **Ballista combines:**
  - Domain testing ideas / Syntax testing ideas
  - In general, "dirty" testing

Ballista: 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

Where we currently are

- **Applied methodology across a wide range of software systems**
  - Operating Systems
  - User level libraries
  - DOD distributed simulation framework
  - Commercial Java Beans
  - Corporate COM/DCOM distributed control framework
  - Critical Military Systems

- **Improved testing granularity by decomposing data types into orthogonal properties**

Experimental Question

Can we get excellent robustness without sacrificing performance?

Goal: *Improved Robustness*

- **In general, robustness of commercial systems is low**
  - OS core system call failure rates from 2-12% across a range of systems
  - User level code varies greatly, on average not as good as OSes

- **Anecdotal evidence indicated that more robust systems are more reliable**

Goal: *Low execution time performance penalty*

- **Original Ballista data resulted in much interaction with commercial OS and middleware developers**

- **Major reasons given for not including better exception handling in systems to increase robustness:**

NEAR PERFECT COVERAGE

&

PERFORMANCE PENALTY

Experimental Setup

SFIO[korn91] – a brief introduction

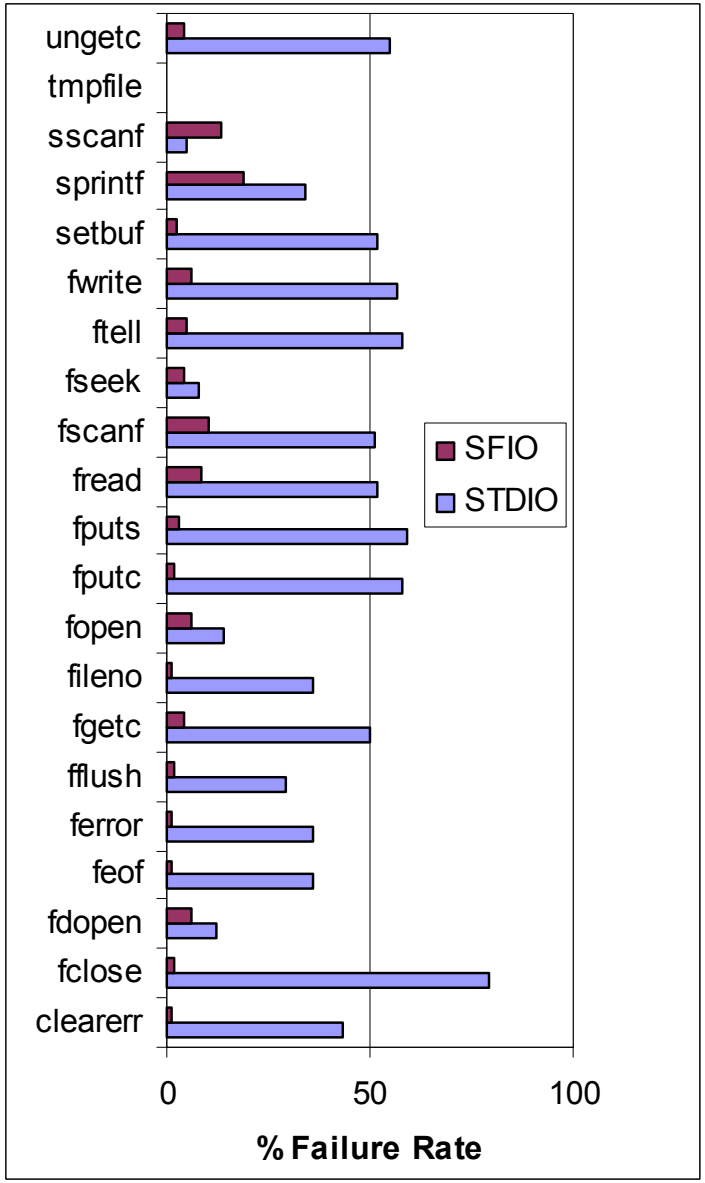
- **Idea:**
  - Measure something that is supposed to be bulletproof
  - See if being really "bulletproof" of necessity costs performance

- **The Safe, Fast, I/O library**
  - Written by Korn and Vo at AT&T research, 1991
  - Addresses the many safety/robustness/reliability issues found in the Standard IO libraries

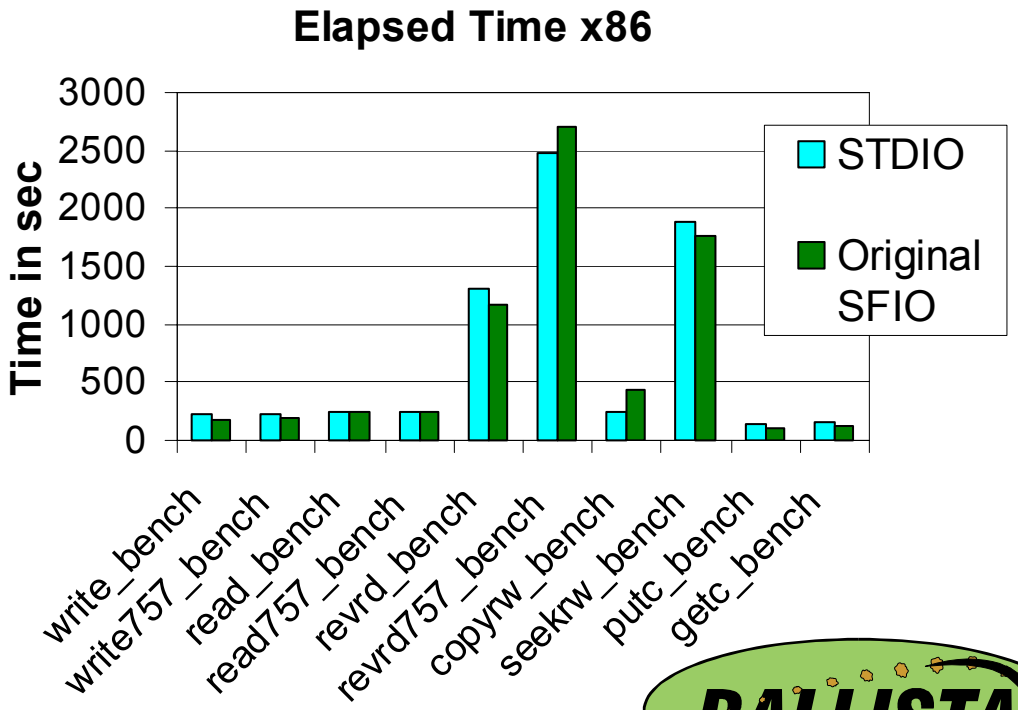
- **Their goal: safe operation with robust exception handling without paying a performance premium**

[Korn91] Korn, D.G.; Vo, K.-P., "SFIO: safe/fast string/file IO," Proceedings of the Summer 1991 USENIX Conference

SFIO, the original version (1990)



- They couldn't measure; but we can
- Up to 10x Improvements in robustness
- Low performance impact



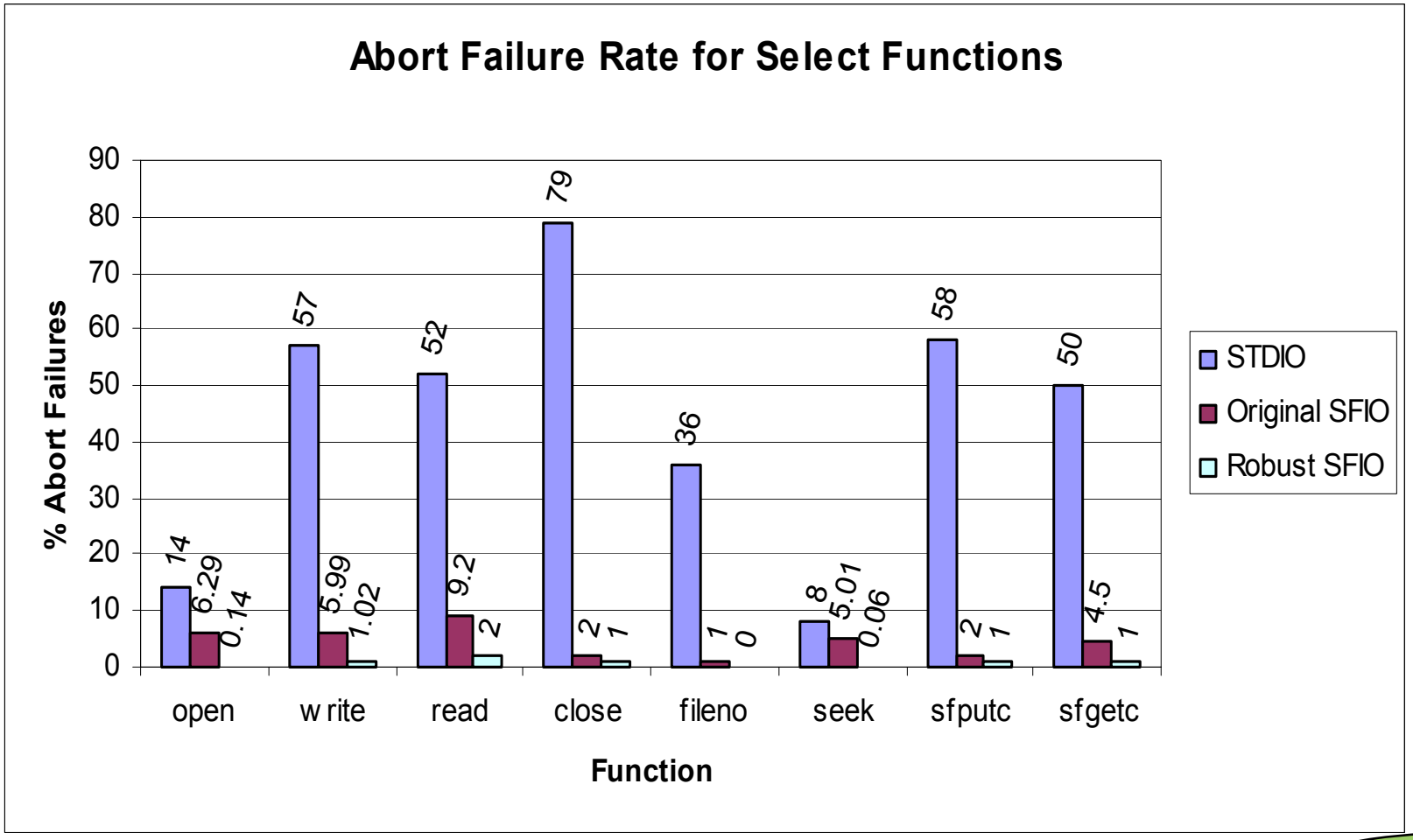
BALLISTA

So what can we observe?

- **The authors of SFIO had no metric**
- **They fixed a large number of problems**
  - BUT, they didn't find them all!
- *** The lack of quantitative feedback made it difficult to know how well they had done, and cost vs. benefit**
- **Performance impact was low**
  - If they fixed everything possible what more could we do?
  - If we could fix anything else, what would the cost be?

Our version is 5-7x more robust

- The use of a metric – in our case Ballista – allowed us to improve performance with respect to exception handling an additional 5-7x



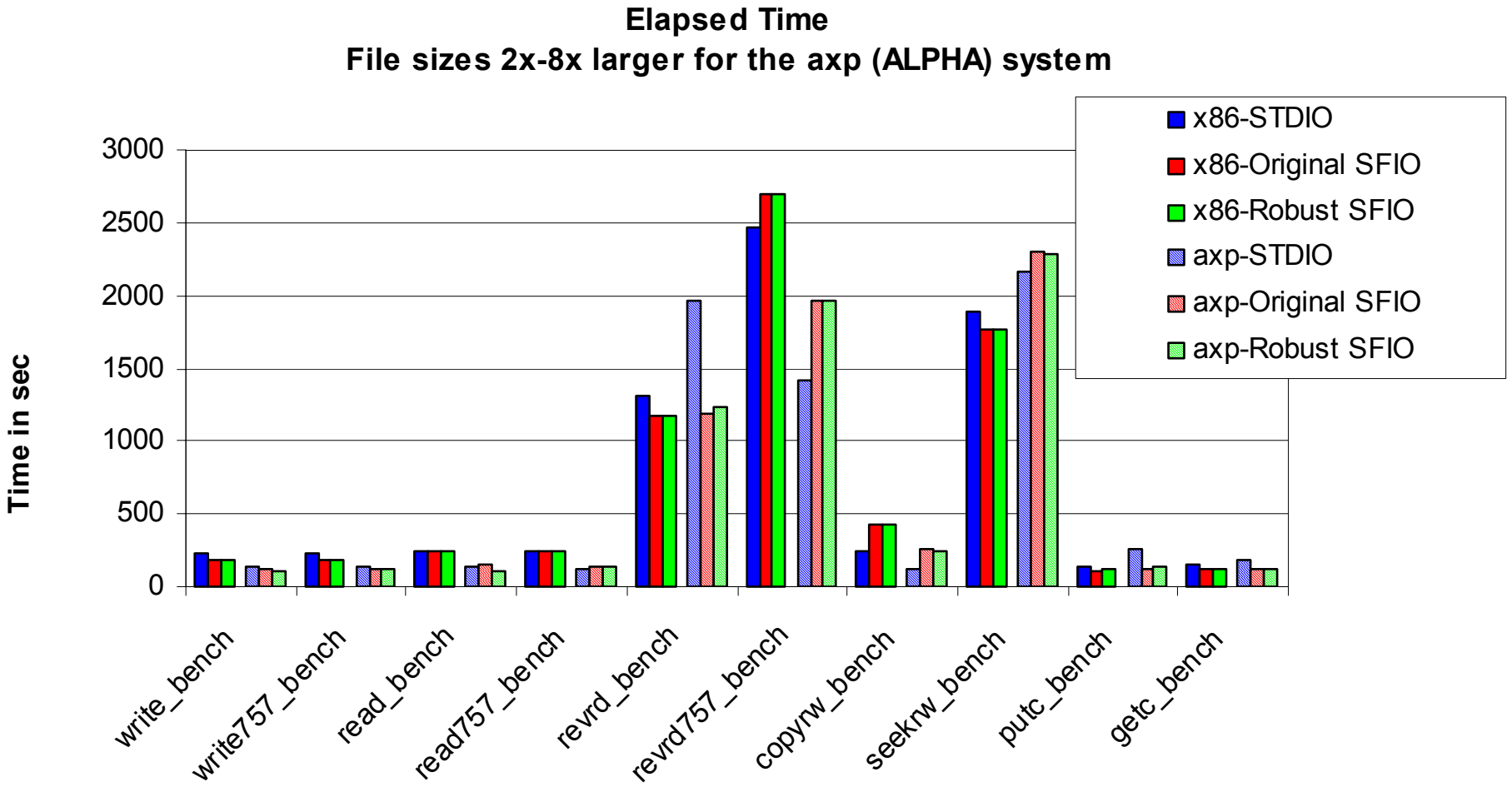
Using a Metric leads to better robustness

- **So without a good metric…**
  - They missed opportunities for easy robustness gains
  - They honestly thought they had found all the easy stuff

- **The types of failures exhibited can be broadly classified as:**
  - File permissions
  - Memory validation

CPU Cycles – wither thou goest?

- **Better exception handling, but at what cost? – Not much <1%**



What changed?

- **It was likely true that robust software suffered a large performance penalty in the past**
  - *In fact, our first attempt suffered huge performance penalties*
- **But it is not true today (penalties can be small)**
- **Penalties will continue to shrink in the future**
- ***Advances in µArchitecture allow us to hide the cost of the added instructions***

Resource Heavy Super-scalar

- **Glut of unused processor resources allow us to insert independent code without starving the program thread**
  - The Intel Pentium-4 processor has 5 integer execution units, 4 address calculation units, and 2 floating units
- **P4 IPC(instructions per cycle) is only 20-40% more than the P-Pro (source intel: http://developer.intel.com/design/pentium4/papers/249438.htm)**
  - Likely only rarely exceeds 2, when in tightly optimized inner loops using netburst
- **This leaves plenty of resources free**

Fetch Bandwidth

- **Unused resources are only part of the answer**
- **What about Branches that tend to waste fetch bandwidth, contributing to pipeline stalls?**

- **The Trace/Block cache**
  - Allows fetch of multiple basic blocks at once
- **Multiple Branch Predictions**
  - Allows speculative execution to begin on *several* basic blocks
- **Easy to predict**
  - Usually only 1

Summary

- **The performance cost of building robust systems need not be large (less than 1%)**
  - New hardware will reduce the penalty further
- **Without a good metric, even the best effort is just a stab in the dark**
  - In this case, the metric was used as feedback to improve SW
- **With a good metric we can do a better job with robustness, and know where to expend effort and what that effort buys us**