PERFORMANCE ANALYSIS OF GRADING SANDBOXES FOR REACTIVE TASKS

Bruce Merry, Ph.D.

South Africa Computing Olympiad

Presented by Rob Kolstad, Ph.D.
USA Computing Olympiad

GRADER OVERVIEW

- Training & contests need graders
- Graders must
  - Be easy to use
  - Correctly process their tasks
  - Provide consistent, repeatable results (incl. timing)
- Task types:
  - Batch task (run then evaluate output file)
  - File submission (compare or evaluate file)
  - Program segment (a la Topcoder & IOI'10)
  - Reactive/interactive

LINUX TIMING, I

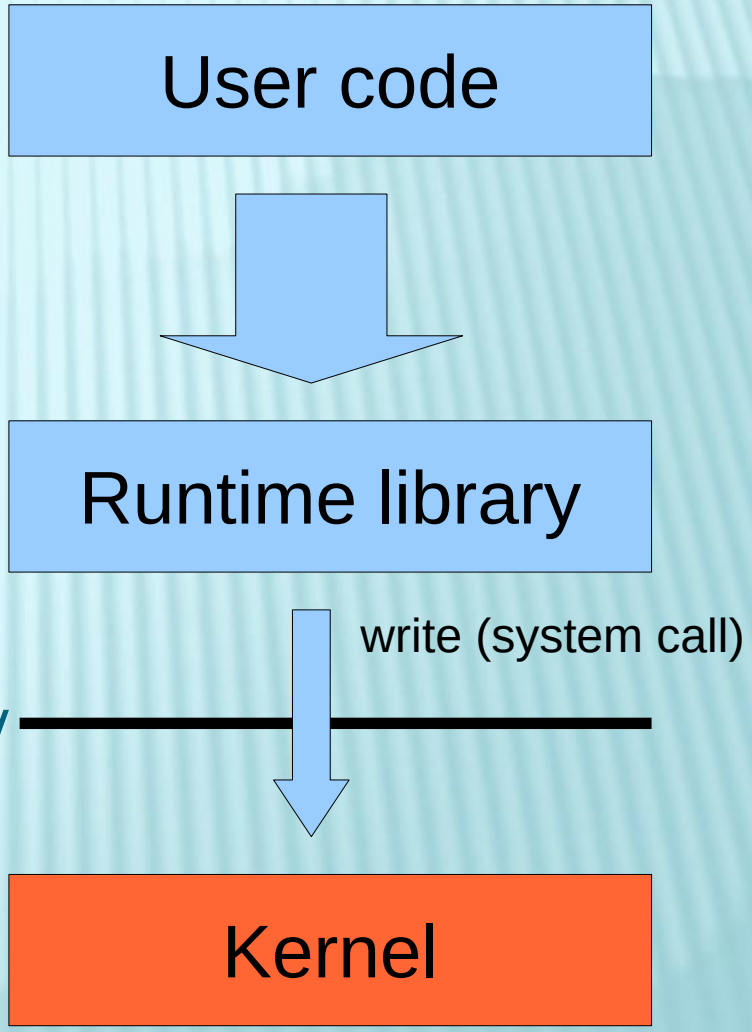
- Traditional (Unix-style) timing:
  - Timer interrupt occurs at 50, 60, or 100 Hz
  - Processing running when interrupt occurs earns entire timeslice for its CPU time
  - Works fine for straight-through, single process tasks (like batch tasks and program-segment submissions)
  - Works terribly when system is rapidly context switching
  - Improve accuracy with longer and repeated runs

LINUX TIMING, II

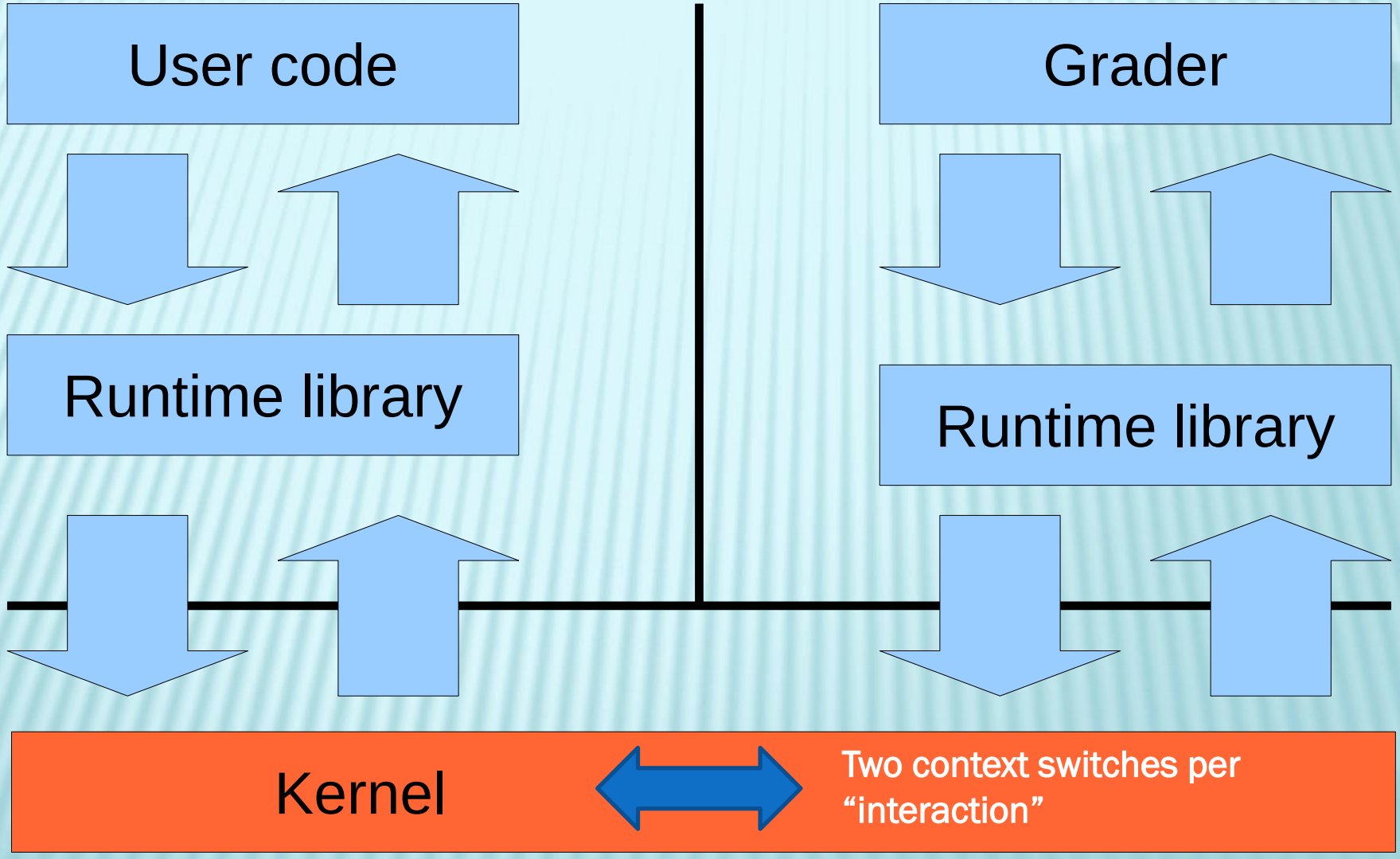
- ‘Microsecond timing’
  - Still not standardized
  - Uses high-resolution clock deltas accounted when control passes to user process
  - Generally high resolution, more accurate, and more repeatable
  - Still slightly inconsistent due to memory caching and disk caching

SYSTEM CALLS: BATCH TASKS

- System calls are generally more CPU-time intensive than user code
- Several fscanf/fprintf invocations (resulting in read/write syscalls)
- Buffered by the runtime library
- Relatively few system calls



SYSTEM CALLS: REACTIVE TASKS



SYSTEM CALLS: REACTIVE TASK

- Every interactive message is flushed (write system call) immediately
- No real buffering possible
- Thus, **lots** of system calls and, of course, lots of context switches
- Syscalls and context switches an easily dominate running time in a task

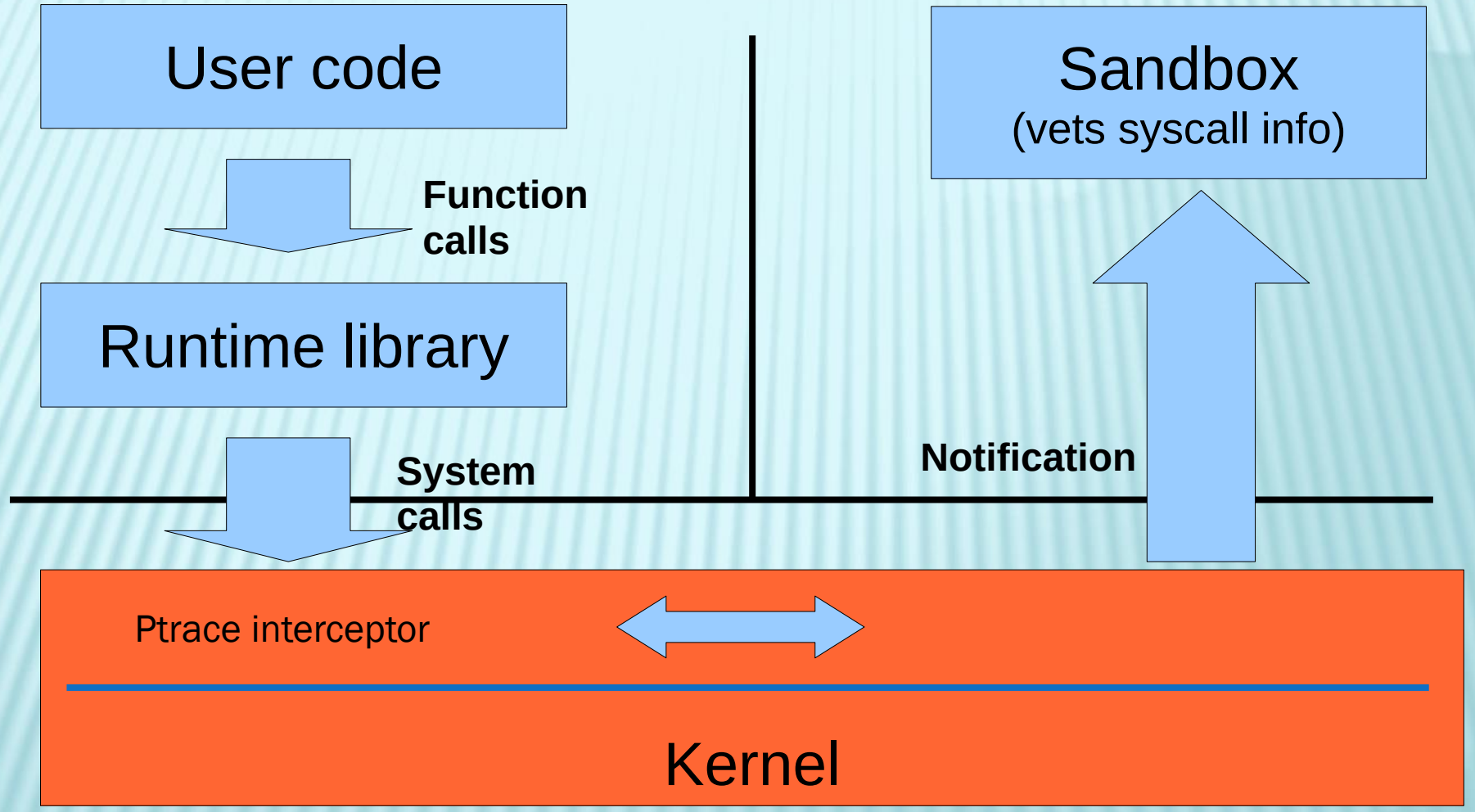
GRADER 'SANDBOXING'

- Modern graders insulate host computer from malicious programs that
  - Try to destroy resources
  - Try to examine system components not meant for public disclosure
  - Execute other processes
  - Start extra threads
  - Open network sockets
  - Kill other processes running under the same user
  - Etc.
- Thus, graders implement a 'sandbox' in which programs can play but not access disallowed resources

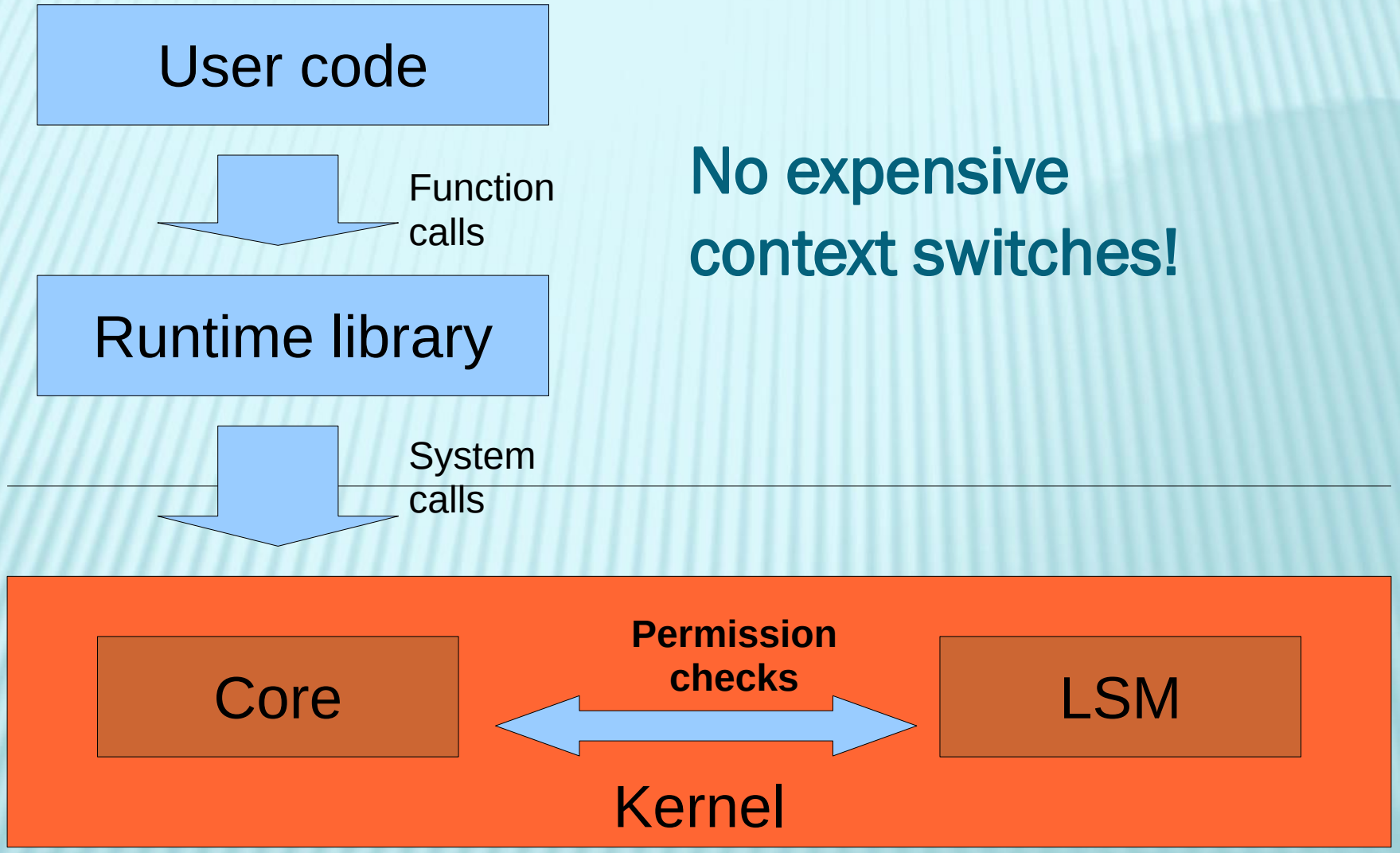
SANDBOX IMPLEMENTATION

- Most popular method for screening is ‘ptrace’ system call
  - Gives **EACH** syscall’s information to security program before syscall execution is started
  - Program vets syscall and returns if all is well
- Easy to see that sandboxed syscalls consume lots of CPU time (checking and two context switches)
- Heavy syscall use (e.g., reactive tasks) exacerbates this problem

TRADITIONAL PTRACE-BASED SANDBOX



NEW LINUX SECURITY MODULE



No expensive context switches!

LSM IMPACT ON PERFORMANCE

- Batch tasks have few syscalls, so little impact
- Reactive tasks perform many syscalls and thus receive highest impact (vs. non-sandbox environment)
- Most interested in impact on measured time
- Also interested in repeatability/variability
- Note: CPU time spent on verification is not a problem if it is correctly accounted

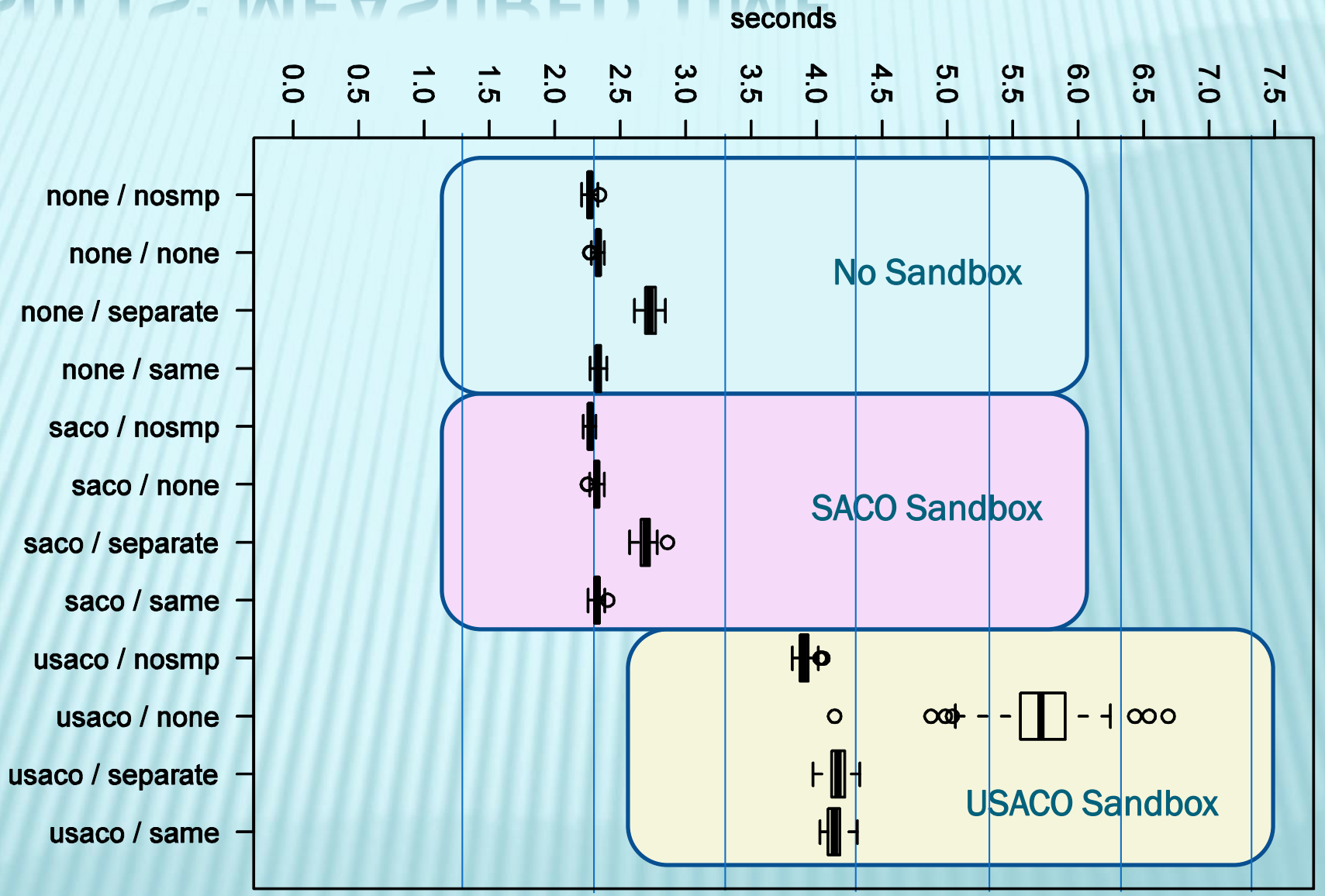
REACTIVE TIMING TEST SETUP

- One task: “Regions” from IOI 2009
- One test case: The largest one
- One machine: Core 2 Duo, dual-core, 2.16GHz
- Two sandboxes:
  - SACO sandbox (LSM-based)
  - USACO sandbox (ptrace-based)

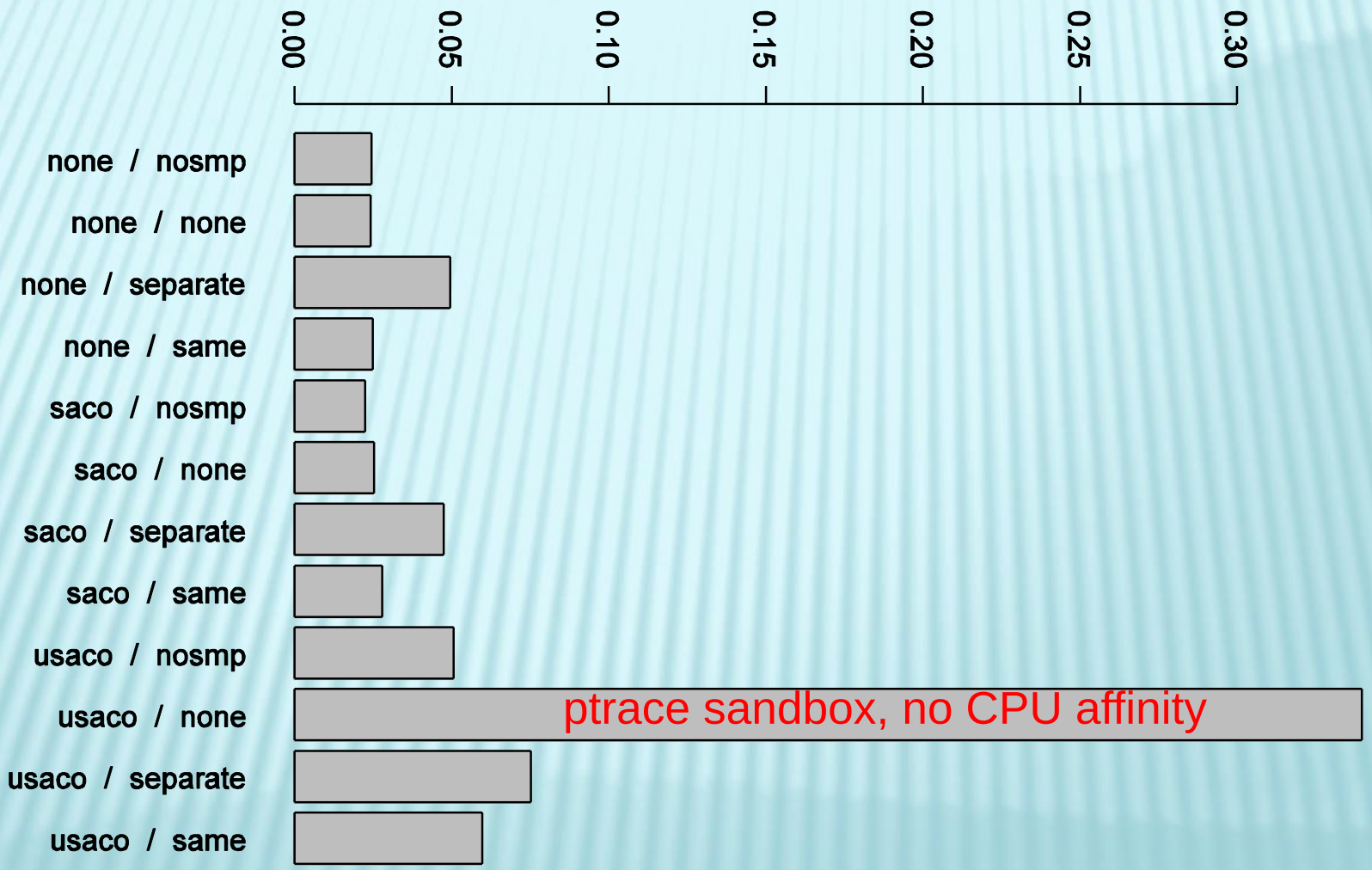
CPU "AFFINITY"

- Multi-core processors can allocate more than one CPU for running programs
- "Affinity" connotes one or more programs preferentially (or exclusively) executing on some single CPU
- Four options for CPU affinity:
  - Disable all but one core
  - Lock grader and user process to same core
  - Lock grader and user process to different cores
  - No affinity (processes free to migrate)

RESULTS: MEASURED TIME



RESULTS: STANDARD DEVIATION



LSM VS. PTRACE

- + LSM abstracts away from syscall interface, which usually has 101 ways to do the same thing, exposing hooks for the fundamental operations (like writing to a file)
- + LSM is Architecture-independent (moving to x86-64 is free).
- - LSM is a less stable interface (kernel internals get moved around on a whim while system calls have to be stable)
- - LSM hangs your machine if you get it wrong
- - LSM is harder to test (until User Mode Linux fixes this)

CONCLUSIONS

- In-kernel security has negligible measured overhead
- ptrace-based sandbox has significant measured overhead
- CPU affinity affects overhead and variability
- ptrace + no affinity is a bad mix
- Caveat: a limited test
- Availibility: Contact Bruce Merry bmerry@gmail.com

QUESTIONS?