TraceBack: First Fault Diagnosis by Reconstruction of Distributed Control Flow

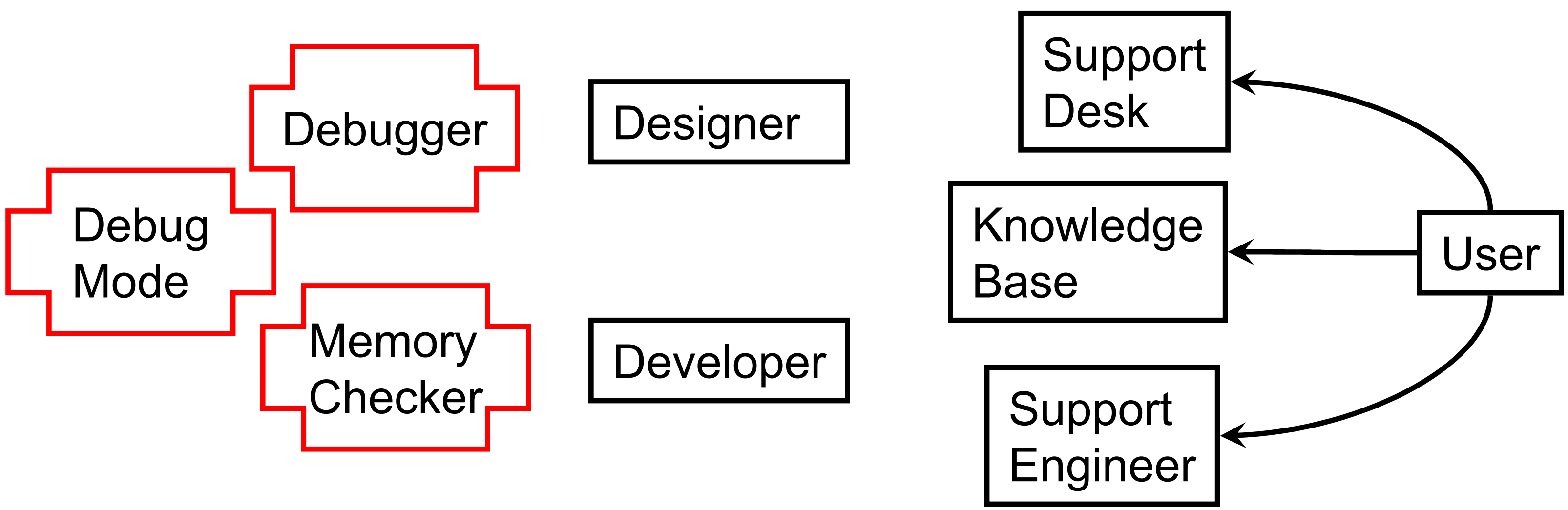
Andrew Ayers
Richard Schooler
Microsoft

Chris Metcalf
VERITAS
Anant Agarwal
MIT

Junghwan Rhee
Emmett Witchel
UT Austin

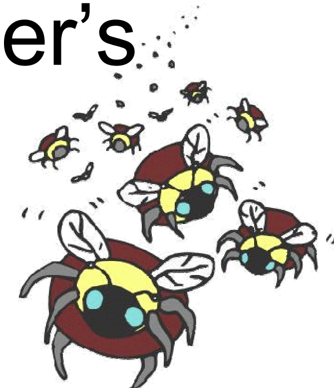
Software Support

- Why aren't users also useful testers?
  - Neither users nor developers benefit from bug
  - Many options for user, all bad
  - Developer tools can't be used in production
  - Sometimes testers aren't useful testers



Bug Reports Lack Information

- Thoroughly documenting a bug is difficult
- Bug re-creation is difficult and expensive
  - Many software components, what version?
  - Might require large, expensive infrastructure
  - Might require high load levels, concurrent activity (not reproducible by user)
  - Might involve proprietary code or data
- Bug re-creation does not leverage user's initial bug experience



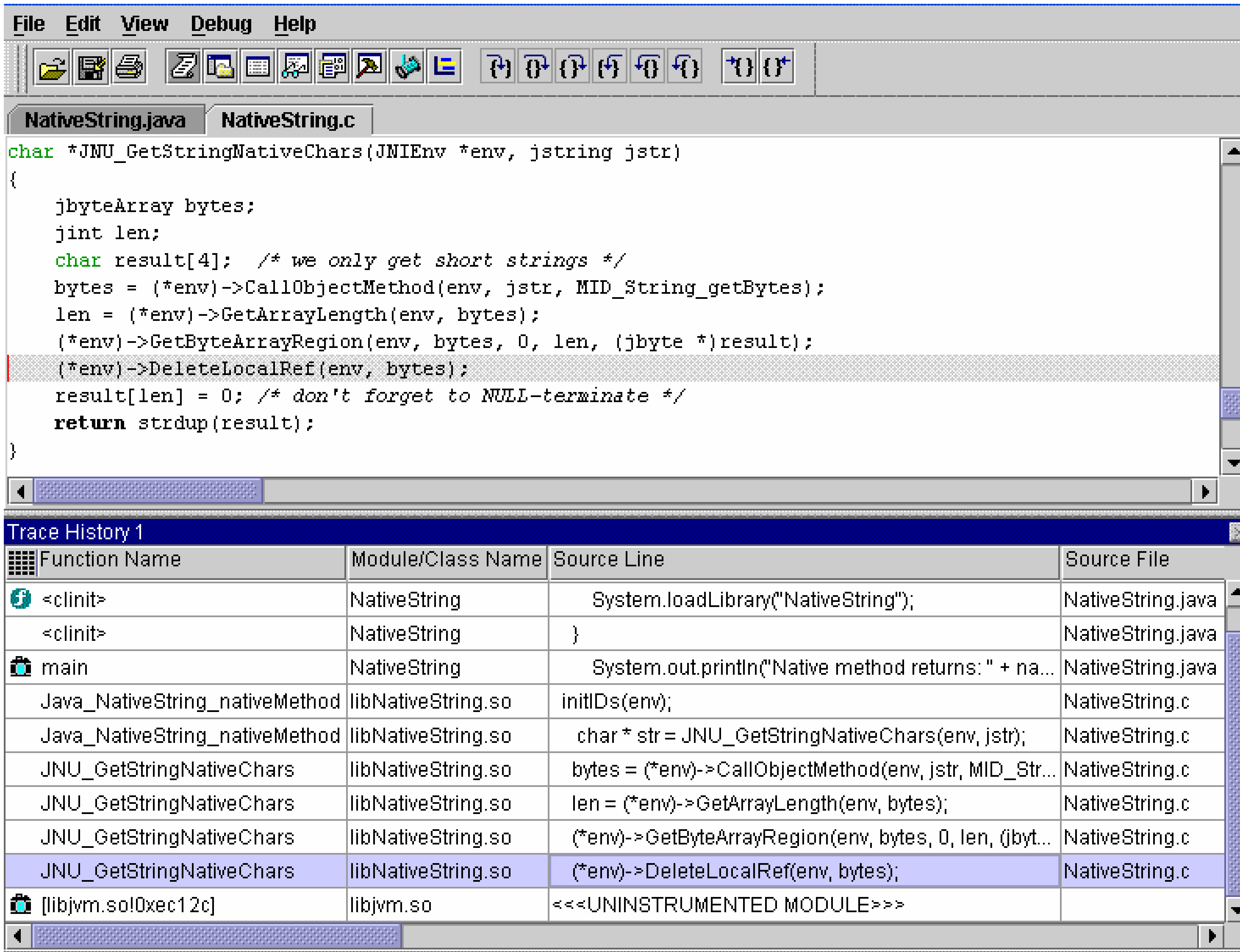
TraceBack: First Fault Diagnosis

- Provide debugger-like experience to developer from user's execution
  - TraceBack helps first time user has a problem
  - Easier model for users and testers
- TraceBack constraints
  - Consume limited resources (always on)
  - Tolerate failure conditions
  - Do not modify source code
- Systems are more expensive to support than they are to purchase

TraceBack In Action

Step back

Step back out



Talk Outline

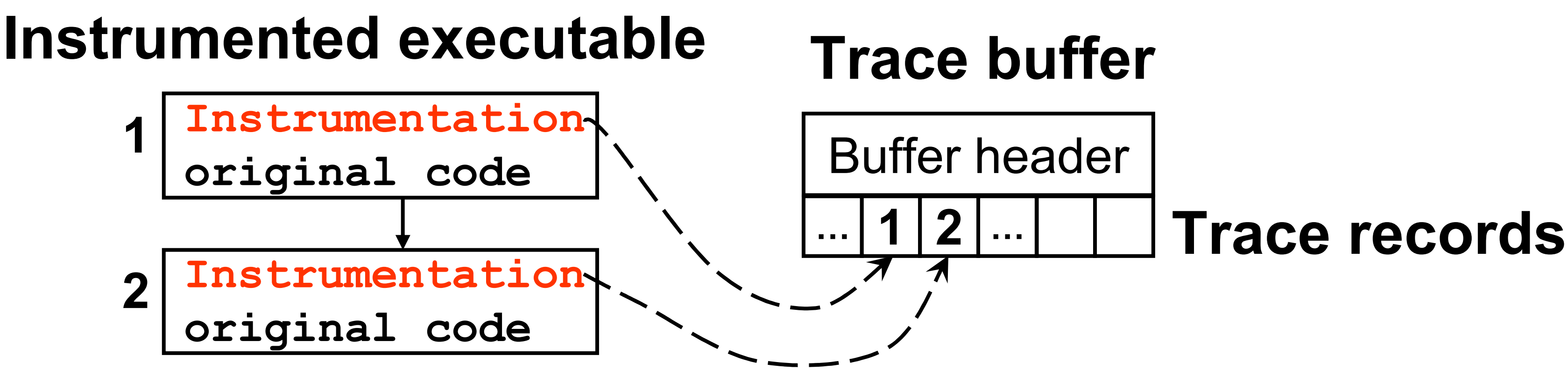
- Design
- Implementation
- Deployment issues
  - Supporting long-running applications
  - Memory allocation issues
  - Generating TraceBack bug reports
- Trace viewing
- Cross-language, cross-machine traces
- Results

TraceBack Design

- Code instrumentation + runtime support
- Do more work before and after execution time to minimize impact on execution time
- Records only control flow
  - Stores environment + optionally core dump
- Captures only recent history
  - Circular buffer of trace records in memory
  - Previous 64K lines
- Vendor statically instruments product using TB, gets TB bug reports from field

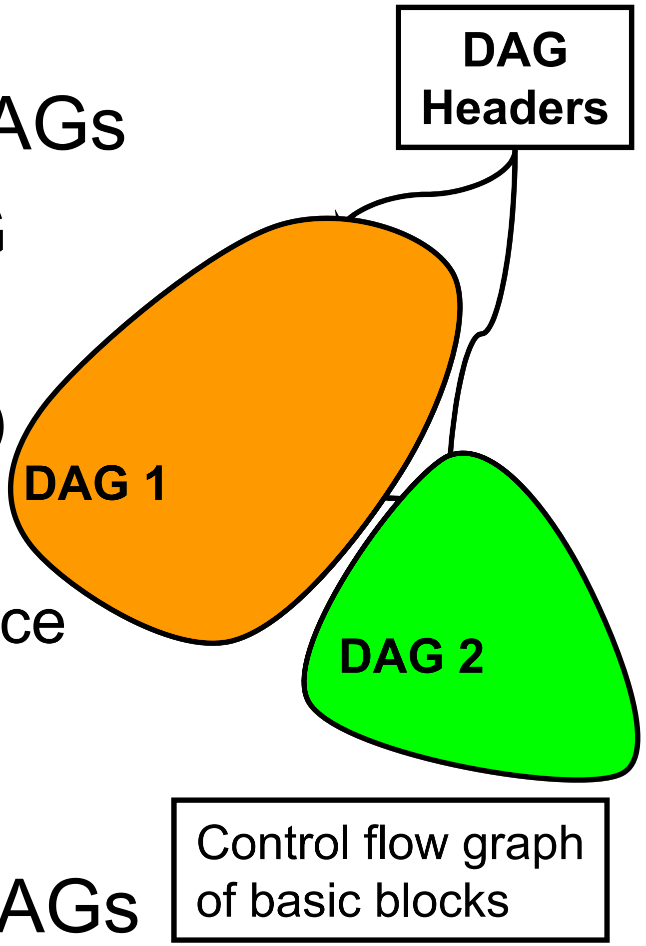
Instrumentation Code

- Instrumentation records execution history
  - Executable instrumented with code probes (statically–minimize impact on execution time)
  - Code probes write trace records in memory
- Common case—flip one bit per basic block
- Each thread has its own trace buffer



Efficiently Encoding Control Flow

- Minimize time & space overhead
- Partition control flow graph by DAGs
- Trace record–one word per DAG
  - DAG header writes DAG number
  - DAG blocks set bits (with single **or**)
- Break DAGs at calls
  - Easiest way for inter-procedural trace
  - Any call can cross modules
  - Performance overhead
- Module becomes sequence of DAGs



Module DAG Renumbering

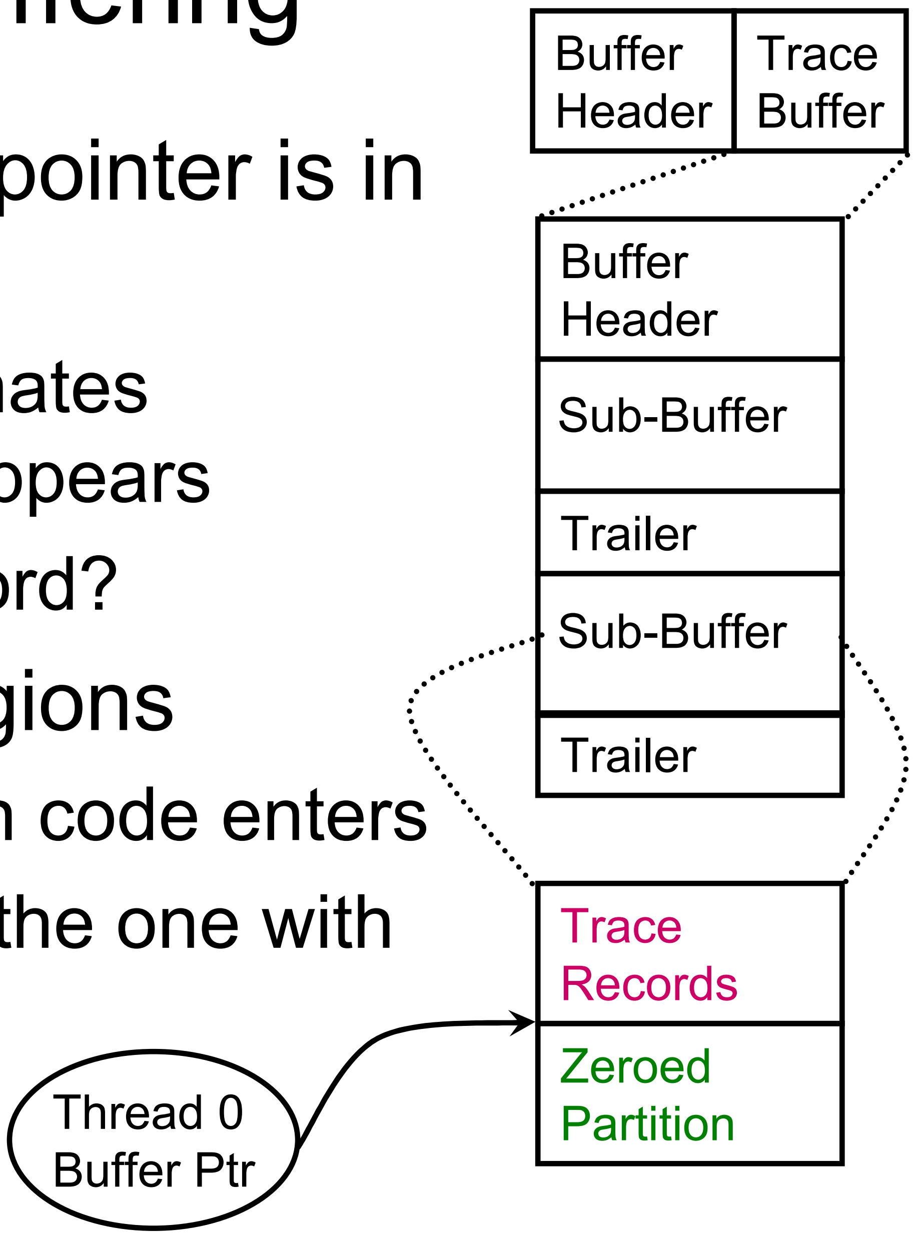
- Real applications made of many modules
- Code modules instrumented independently
  - Which DAG is really DAG number 1?
- Modules heuristically instrumented with disjoint DAG number spaces (dll rebasing)
- TraceBack runtime monitors DAG space
  - If it loads a module with a conflicting space, it renumbers the DAGs
  - If it reloads same module, it uses the same number space (support long running apps)

Allocating Trace Buffers

- What happens if there are more threads than trace buffers?
  - Delegate one buffer as **desperation** buffer
  - Instrumentation must write records somewhere
  - Don’t recover trace data, but don’t crash
  - On buffer wrap, retry buffer allocation
- What if no buffers can be allocated?
  - Use **static** buffer, compiled into runtime
- What if thread runs no instrumented code?
  - Start it in zero-length **probation** buffer

Sub-Buffering

- Current trace record pointer is in thread-local storage
  - When a thread terminates abruptly, pointer disappears
  - Where is the last record?
- Break buffers into regions
  - Zero sub-region when code enters
  - Current sub-buffer is the one with zeroed records



Snapshots

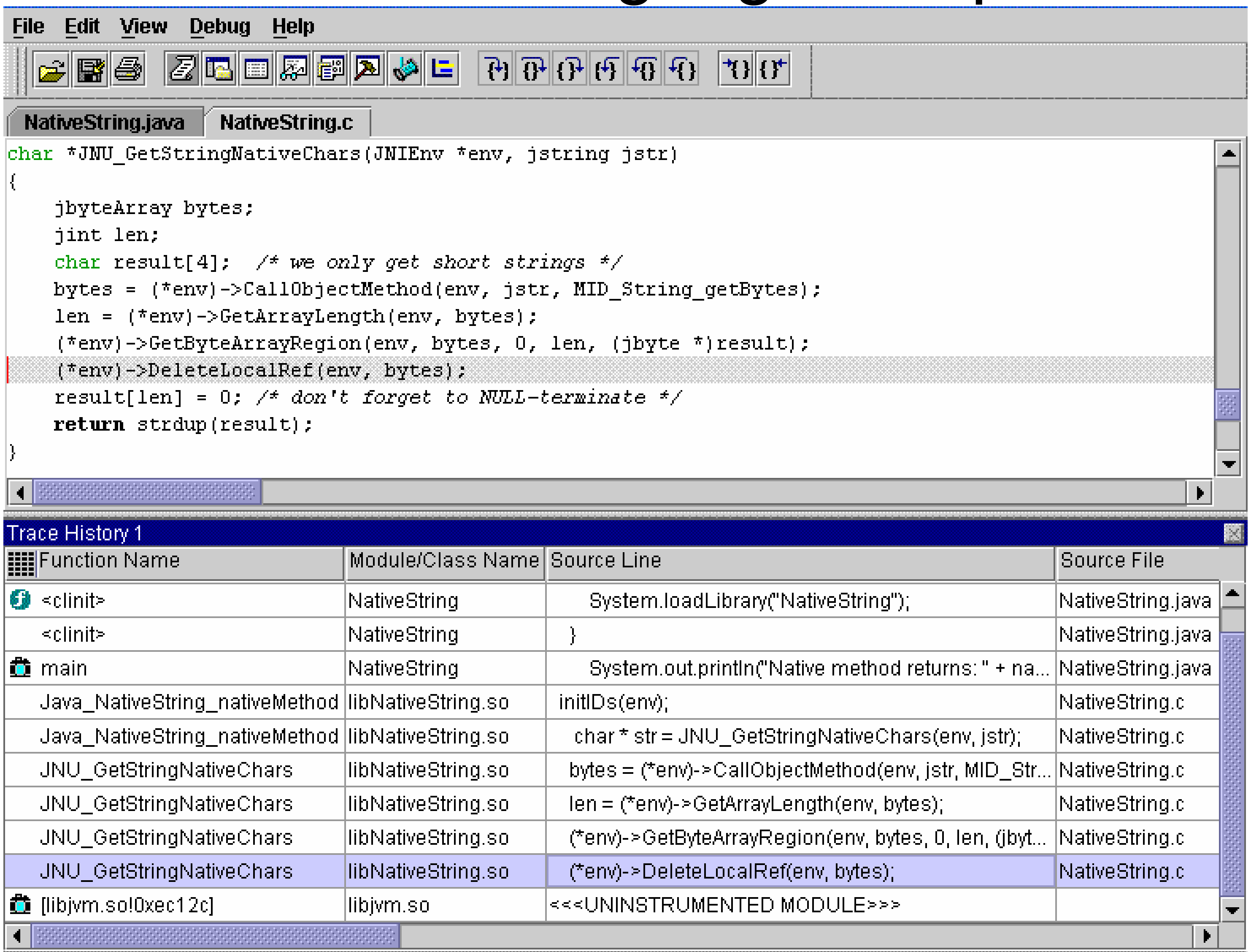
- Trace buffer in memory mapped file
  - Persistent even if application crashes/hangs
- Snapshot is a copy of the trace buffers
  - External program (e.g., on program hang)
  - Program event, like an exception (debugging)
  - Programmatic API (at “unreachable” point)
- Snap suppression is key—users want to store and examine unique snapshots

Trace Reconstruction

- Trace records converted to line trace
- Refine line trace for exceptions
  - Users hate seeing a line executed *after* the line that took an exception
- Call structure recreated
  - Don’t waste time & space at runtime
- Threads interleaved plausibly
  - Realtime timestamps for ordering

Cross language trace

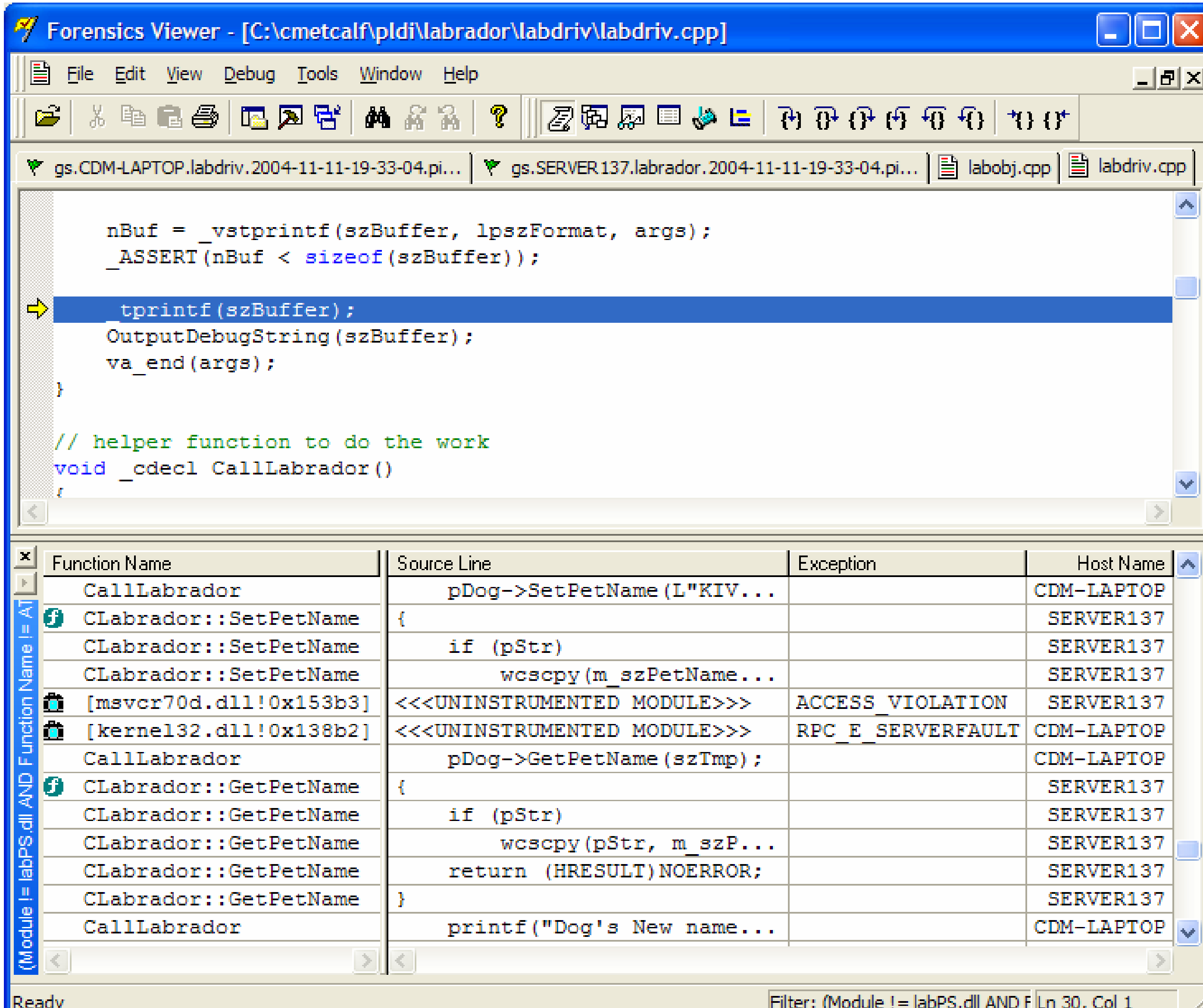
- Trace records are language independent



Distributed Tracing

Logical threads

Real time clocks

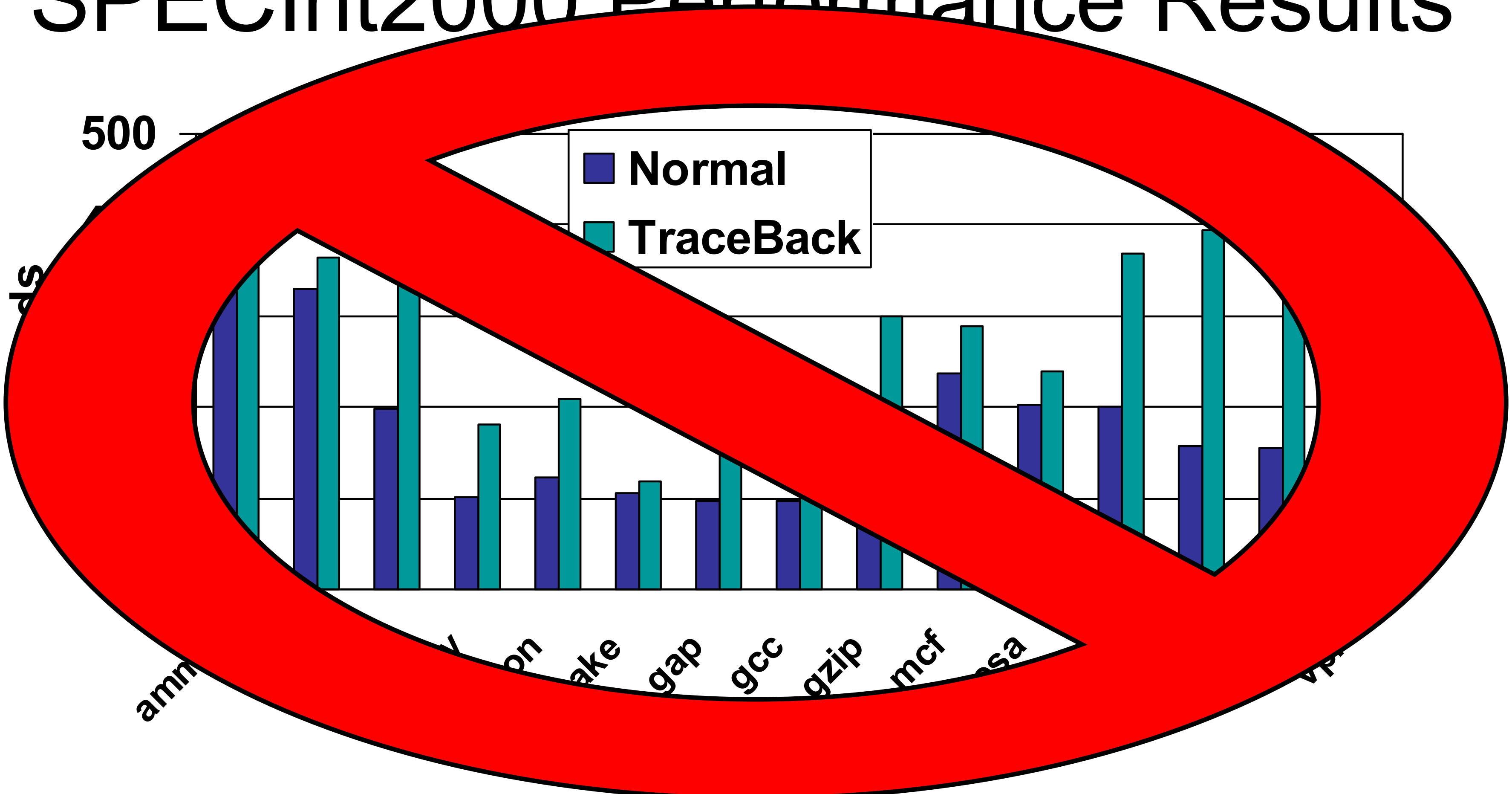


Implementation

- TraceBack on x86—6 engineer-years ('99-'01)
  - 20 engineer-years total for TB functionality
  - Still sold as part of VERITAS Application Saver
- TraceBack product supports many platforms

Language	OS/Architecture
C/C++, VB6, Java, .NET	Windows/x86
C/C++, Java	Linux/x86, Solaris/SPARC
Java only	AIX/PPC, HP-UX/PA
COBOL	OS/390

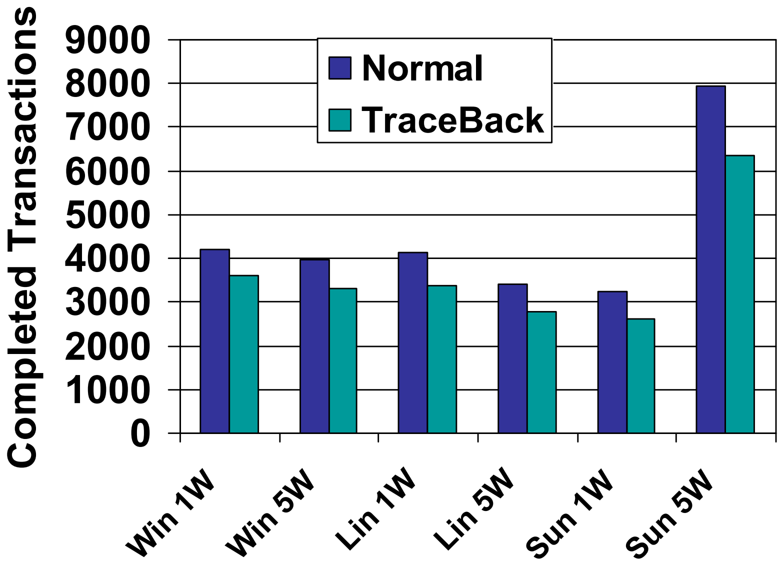
SPECInt2000 Performance Results



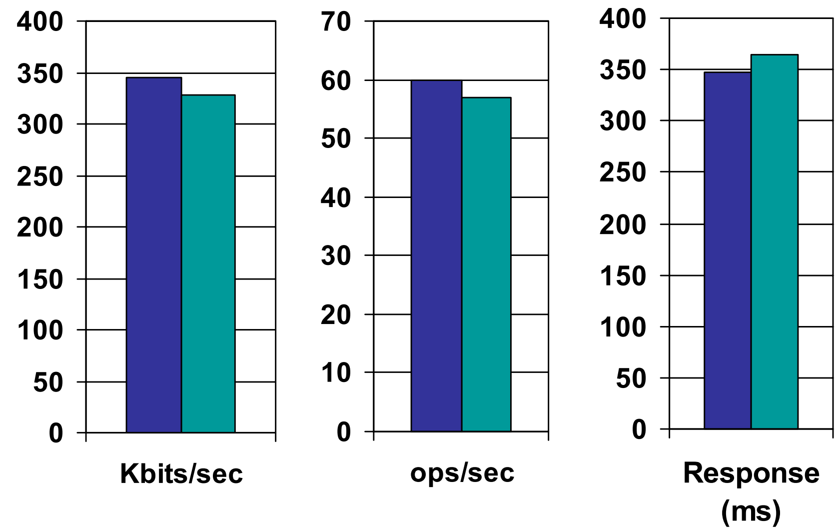
- Geometric mean slowdown 60%
- 3GHz P4, 2GB RAM, VC 7.1, ref inputs

Webserver Performance Results

SPECJbb



SPECWeb99/Apache



- Multi-threaded, long running server apps
- SPECJbb throughput reduced 16%–25%
- SPECWeb throughput & latency reduced < 5%
- Phase Forward slowdown less than 5%

Real World Examples

- Phase Forward's C++ application hung due to a third party database dll
  - Cross-process trace got Phase Forward a fix from database company
- At Oracle, TraceBack found cause of a slow Java/C++ application—too many Java exceptions
  - A call to sleep had been wrapped in a try/catch block
  - Argument to sleep was a random integer, negative half the time
- The TraceBack GUI itself (written in C++) is instrumented with TraceBack
  - At eBay, the GUI became unresponsive (to Ayers)
  - Ayers took a snap, and sent the trace, in real time (to Metcalf)
  - Culprit was an $O(n^2)$ algorithm in the GUI
  - Ayers told the engineers at eBay, right then, about the bug and its fix

Related Work

- Path profiling [Ball/Larus ‘96, Duesterwald ’00, Nandy ’03, Bond ’05]
  - Some interprocedural extensions. [Tallam ’04]
  - Most recent work on making it more efficient (e.g., using sampling which TraceBack can’t).
- Statistical bug hunting [Liblit ’03 & ’05]
- Virtutech’s Hindsight, reverse execution in a machine simulator 2005.
- Omniscient debugger [Lewis ’03]
- Microsoft Watson crashdump analysis.
- Static translation systems (ATOM, etc.)

Future Work

- Navel—current project at UT
- Connect user with workarounds for common bugs
  - Use program trace to search knowledge base
  - Machine learning does the fuzzy matching
- Eliminating duplicate bug reports
- Program behavior is useful data

TraceBack

- Application of binary translation research
  - Efficient enough to be “always on”
- Provides developer with debugger-like information from crash report
  - Multiple threads
  - Multiple languages
  - Multiple machines

Thank you