

TraceBack: First Fault Diagnosis by Reconstruction of Distributed Control Flow

Andrew Ayers

Richard Schooler

Microsoft

Chris Metcalf

VERITAS

Anant Agarwal

MIT

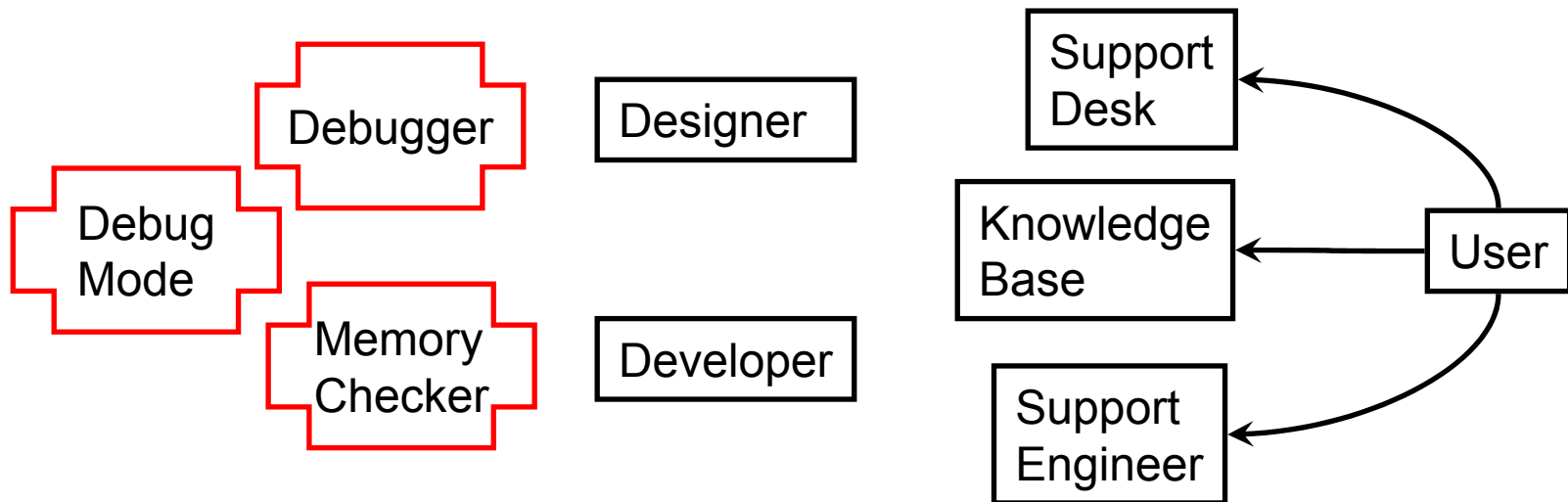
Junghwan Rhee

Emmett Witchel

UT Austin

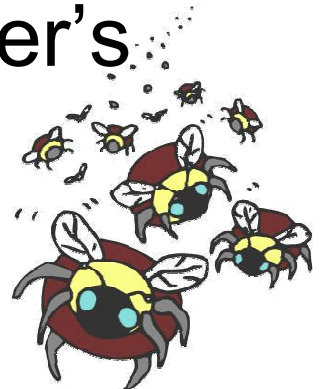
Software Support

- 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

The screenshot shows an IDE window with two tabs: 'NativeString.java' and 'NativeString.c'. The 'NativeString.c' tab is active, displaying the following C code:

```
char *JNU_GetStringNativeChars(JNIEnv *env, jstring jstr)
{
    jbyteArray bytes;
    jint len;
    char result[4]; /* we only get short strings */
    bytes = (*env)->CallObjectMethod(env, jstr, MID_String_getBytes);
    len = (*env)->GetArrayLength(env, bytes);
    (*env)->GetByteArrayRegion(env, bytes, 0, len, (jbyte *)result);
    (*env)->DeleteLocalRef(env, bytes);
    result[len] = 0; /* don't forget to NULL-terminate */
    return strdup(result);
}
```

Below the code editor is a 'Trace History 1' window. It contains a table with the following data:

Function Name	Module/Class Name	Source Line	Source File
<clinit>	NativeString	System.loadLibrary("NativeString");	NativeString.java
<clinit>	NativeString	}	NativeString.java
main	NativeString	System.out.println("Native method returns:" + na...	NativeString.java
Java_NativeString_nativeMethod	libNativeString.so	initIDs(env);	NativeString.c
Java_NativeString_nativeMethod	libNativeString.so	char * str = JNU_GetStringNativeChars(env, jstr);	NativeString.c
JNU_GetStringNativeChars	libNativeString.so	bytes = (*env)->CallObjectMethod(env, jstr, MID_Str...	NativeString.c
JNU_GetStringNativeChars	libNativeString.so	len = (*env)->GetArrayLength(env, bytes);	NativeString.c
JNU_GetStringNativeChars	libNativeString.so	(*env)->GetByteArrayRegion(env, bytes, 0, len, (jbyt...	NativeString.c
JNU_GetStringNativeChars	libNativeString.so	(*env)->DeleteLocalRef(env, bytes);	NativeString.c
[libjvm.sol0xec1 2c]	libjvm.so	<<<UNINSTRUMENTED MODULE>>>	

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

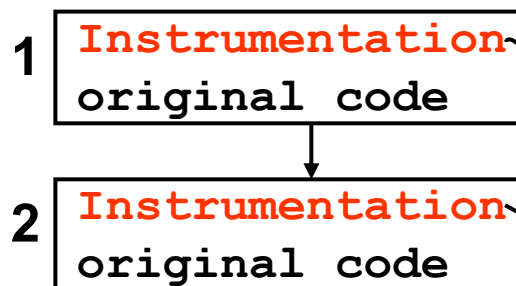
TraceBack Design

- 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

Instrumented executable

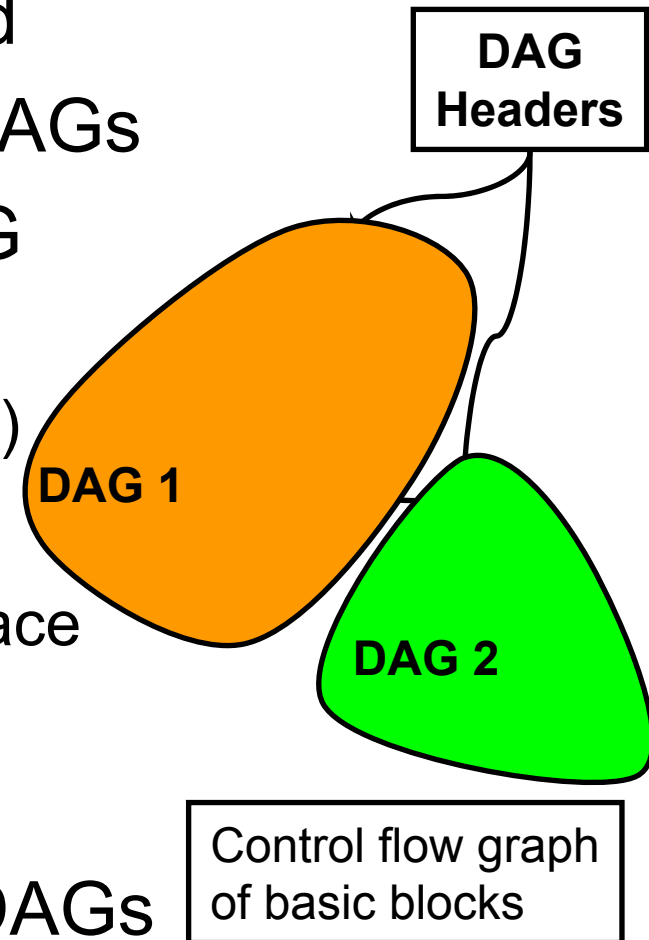


Trace buffer



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

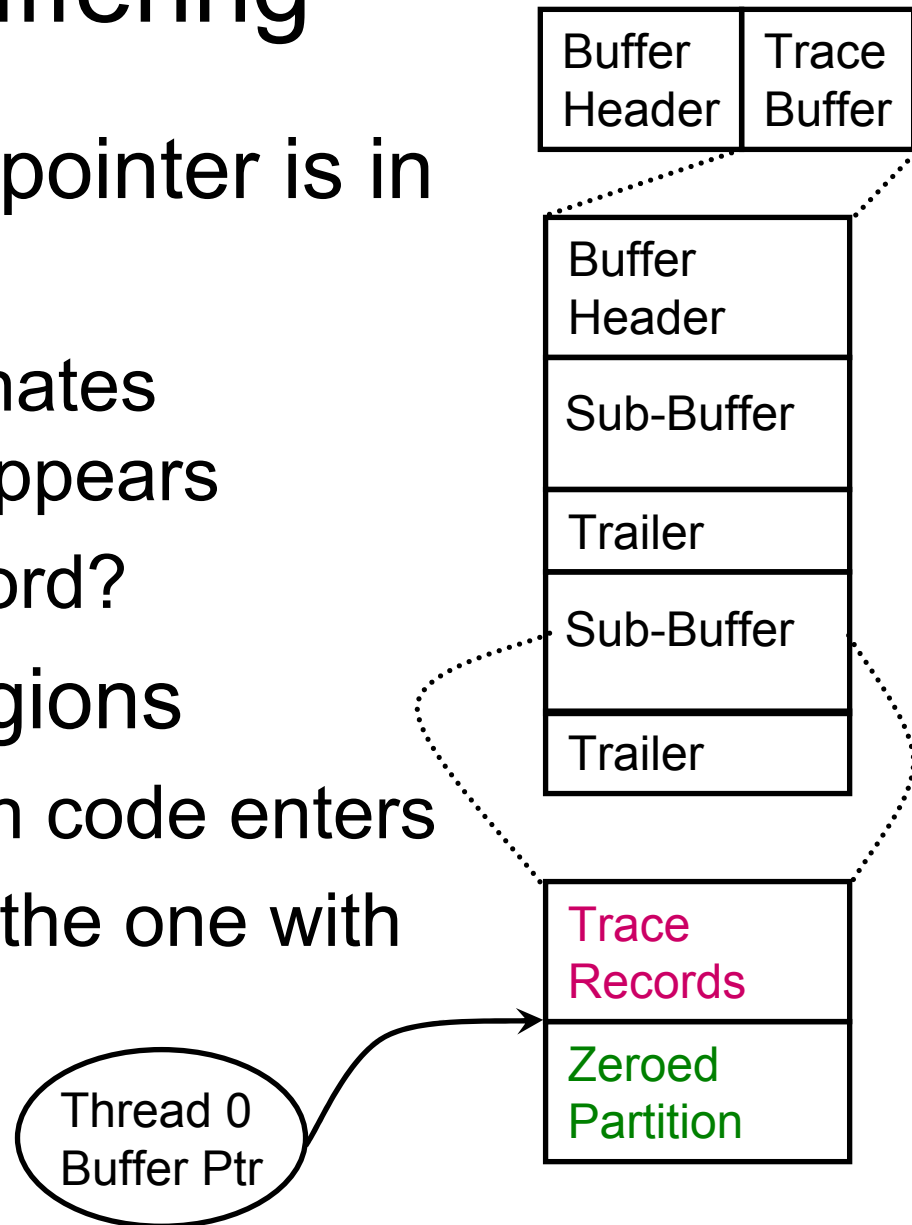
- Real applications made of many modules
- Code modules instrumented independently
 - Which DAG is really DAG number 1?
- Modules heuristically instrumented with disjoint DAG number spaces (dll rebasing)
- TraceBack runtime monitors DAG space
 - If it loads a module with a conflicting space, it renumbers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

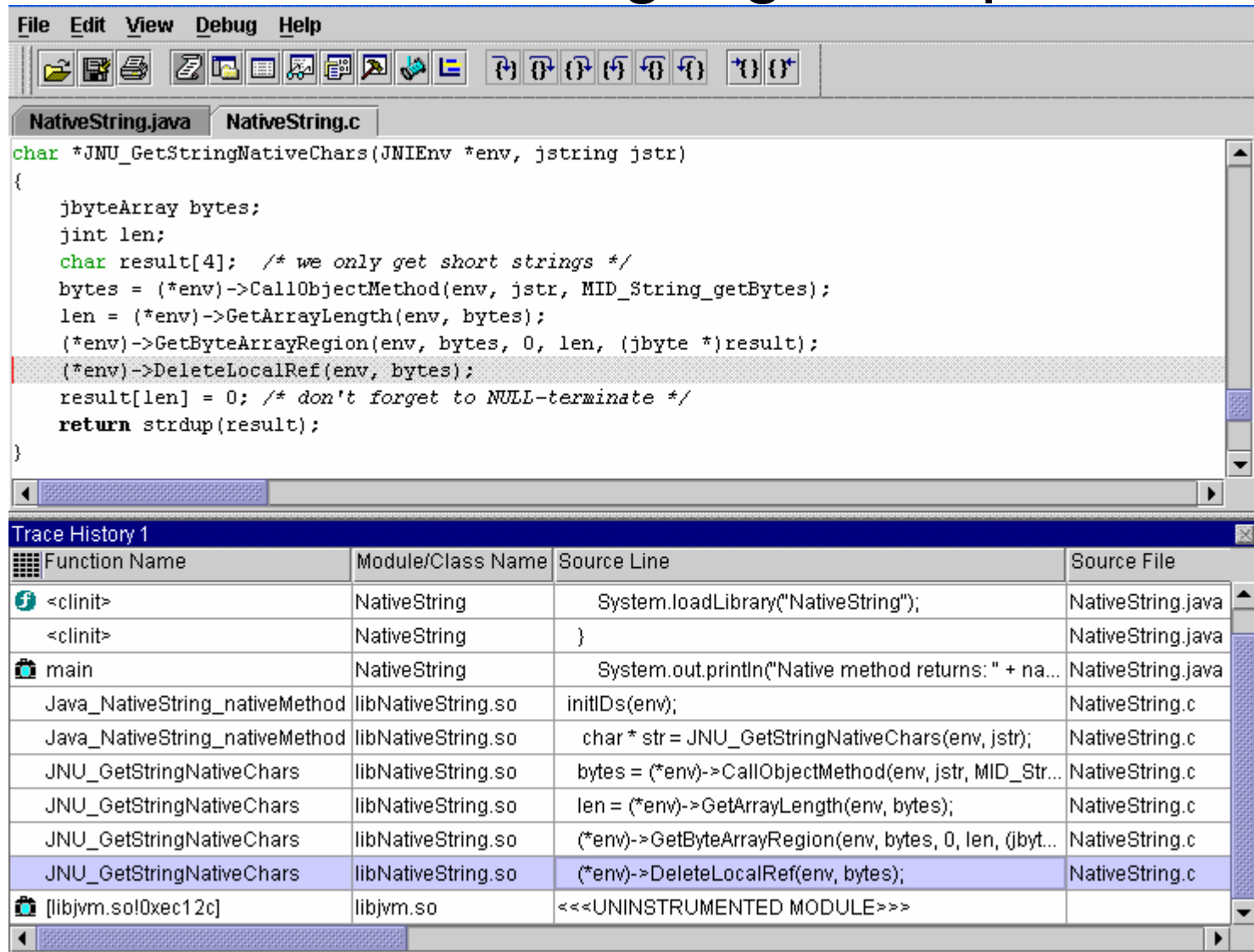
- Trace buffer in memory mapped file
 - Persistent even if application crashes/hangs
- Snapshot is a copy of the trace buffers
 - External program (e.g., on program hang)
 - Program event, like an exception (debugging)
 - Programmatic API (at “unreachable” point)
- Snap suppression is key—users want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



The screenshot displays an IDE window with two panes. The top pane shows the source code for `NativeString.c`, which is a JNI wrapper for a Java string method. The bottom pane shows the 'Trace History 1' window, which is a table of execution records.

```
char *JNU_GetStringNativeChars(JNIEnv *env, jstring jstr)
{
    jbyteArray bytes;
    jint len;
    char result[4]; /* we only get short strings */
    bytes = (*env)->CallObjectMethod(env, jstr, MID_String_getBytes);
    len = (*env)->GetArrayLength(env, bytes);
    (*env)->GetByteArrayRegion(env, bytes, 0, len, (jbyte *)result);
    (*env)->DeleteLocalRef(env, bytes);
    result[len] = 0; /* don't forget to NULL-terminate */
    return strdup(result);
}
```

Function Name	Module/Class Name	Source Line	Source File
<clinit>	NativeString	System.loadLibrary("NativeString");	NativeString.java
<clinit>	NativeString	}	NativeString.java
main	NativeString	System.out.println("Native method returns: " + na...	NativeString.java
Java_NativeString_nativeMethod	libNativeString.so	initIDs(env);	NativeString.c
Java_NativeString_nativeMethod	libNativeString.so	char * str = JNU_GetStringNativeChars(env, jstr);	NativeString.c
JNU_GetStringNativeChars	libNativeString.so	bytes = (*env)->CallObjectMethod(env, jstr, MID_Str...	NativeString.c
JNU_GetStringNativeChars	libNativeString.so	len = (*env)->GetArrayLength(env, bytes);	NativeString.c
JNU_GetStringNativeChars	libNativeString.so	(*env)->GetByteArrayRegion(env, bytes, 0, len, (jbyt...	NativeString.c
JNU_GetStringNativeChars	libNativeString.so	(*env)->DeleteLocalRef(env, bytes);	NativeString.c
[libjvm.sol0xec12c]	libjvm.so	<<<UNINSTRUMENTED MODULE>>>	

Distributed Tracing

Logical threads

Real time clocks

The screenshot shows the Forensics Viewer application displaying a distributed tracing log. The main window shows source code for `labdrv.cpp` with a yellow arrow pointing to the `tprintf` call. Below the code is a table of function calls and exceptions.

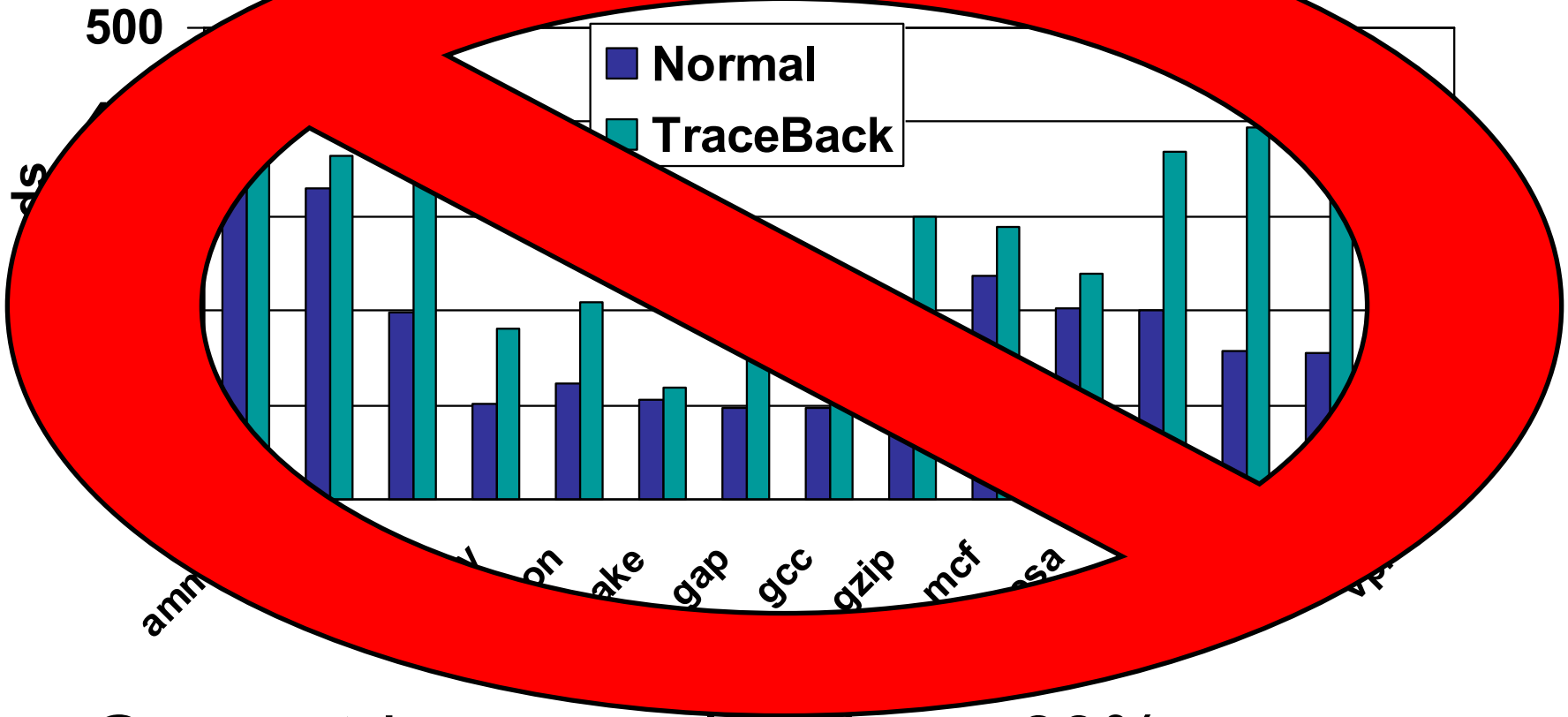
Function Name	Source Line	Exception	Host Name
CallLabrador	<code>pDog->SetPetName (L"KIV...</code>		CDM-LAPTOP
CLabrador::SetPetName	<code>{</code>		SERVER137
CLabrador::SetPetName	<code>if (pStr)</code>		SERVER137
CLabrador::SetPetName	<code>wcscpy(m_szPetName...</code>		SERVER137
[msvcr70d.dll!0x153b3]	<<<UNINSTRUMENTED MODULE>>>	ACCESS_VIOLATION	SERVER137
[kernel32.dll!0x138b2]	<<<UNINSTRUMENTED MODULE>>>	RPC_E_SERVERFAULT	CDM-LAPTOP
CallLabrador	<code>pDog->GetPetName (szTmp);</code>		CDM-LAPTOP
CLabrador::GetPetName	<code>{</code>		SERVER137
CLabrador::GetPetName	<code>if (pStr)</code>		SERVER137
CLabrador::GetPetName	<code>wcscpy(pStr, m_szP...</code>		SERVER137
CLabrador::GetPetName	<code>return (HRESULT)NOERROR;</code>		SERVER137
CLabrador::GetPetName	<code>}</code>		SERVER137
CallLabrador	<code>printf("Dog's New name...</code>		CDM-LAPTOP

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

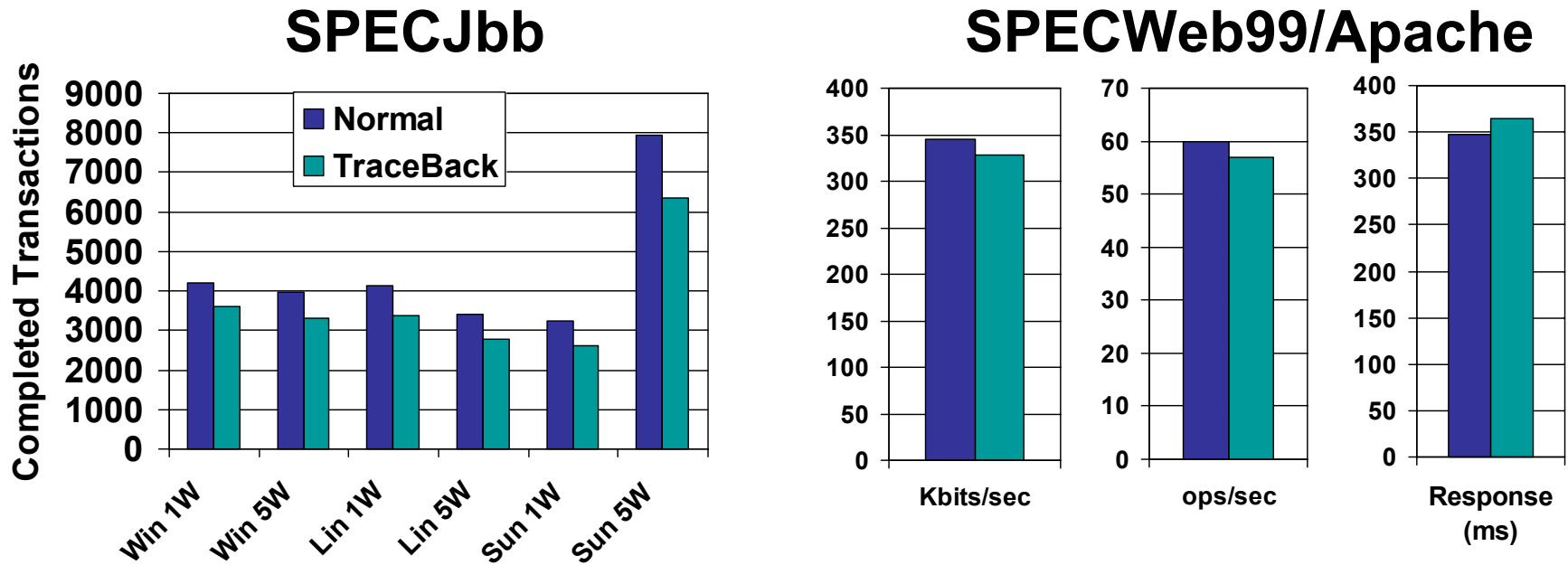
Language	OS/Architecture
C/C++, VB6, Java, .NET	Windows/x86
C/C++, Java	Linux/x86, Solaris/SPARC
Java only	AIX/PPC, HP-UX/PA
COBOL	OS/390

SPECInt2000 Performance Results



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

Webserver Performance Results



- 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