Dynamic Tracing and Instrumentation

Bryan Cantrill and Mike Shapiro
(bmc, mws@eng.sun.com)
Solaris Kernel Group

Kernel Debugging Today

- **`if`** `(no_advanced_debugging)` **`printf`**`(9f)`
- **`ASSERT`**`(i_am_a_debug_kernel != 0);`
- In-situ kernel debugger (`kadb`)
- In-situ firmware debugger (SPARC obp)
- Run-time tracing (trap trace, kmem allocator, lockstat)
- Advanced post-mortem debugger (`mdb`)

Problem

- Post-mortem analysis is an effective technique for solving problems where the system panics
- Forcing panics is an impractical technique for diagnosing non-fatal problems in the field
- Forcing panics is an ineffective technique for diagnosing transient non-fatal errors and performance problems

General Observations

- ASSERTs and trap trace are powerful facilities, but you only get them in a DEBUG kernel at a high cost
- Kmem allocator debugging facilities are powerful, but require a reboot and cause global performance hit
- Kernel is filled with #ifdef'd tracing/debugging code
- VTRACE facility solves some problems, but has fallen into disrepair and requires a TRACE kernel
- Lockstat is a dynamic facility and has flexible output, but is for a single problem domain and is a closed system

TNF Observations

- Only a small number of probes exist in the kernel
- Probe overhead precludes use in sensitive code paths
- TNF traces can often only be interpreted with the aid of other unrelated tools (e.g. iostat, ls -lL /dev/dsk)
- User interface and programming model are poor

```
TNF_PROBE_3(syscall_end, "syscall thread", /* CSTYLED */,
           tnf_long,  rval1,     rval1,
           tnf_long,  rval2,     rval2,
           tnf_long,  errno,     error);
```

Competition

- IBM has provided extensive general purpose tracing tools as part of MVS and AIX
- GTF facility provides configurable, system-wide tracing facility that can be enabled on a production system
- Data can be consumed live or from crash dump
- Data can be consumed by a variety of tools
- Extensive documentation provided

State of the Union

- Network storage engineers use TNF, but are blocked or frustrated by its limitations
- Mainframe-class sites long for something like GTF
- Field is still stuck with resorting to reboots, custom kernels, or custom patches or drivers for debugging
- Real-time customers lack tools to debug latency bubbles
- Partners (e.g. Fujitsu, Siemens, Motorola) and developers all want better tracing facilities

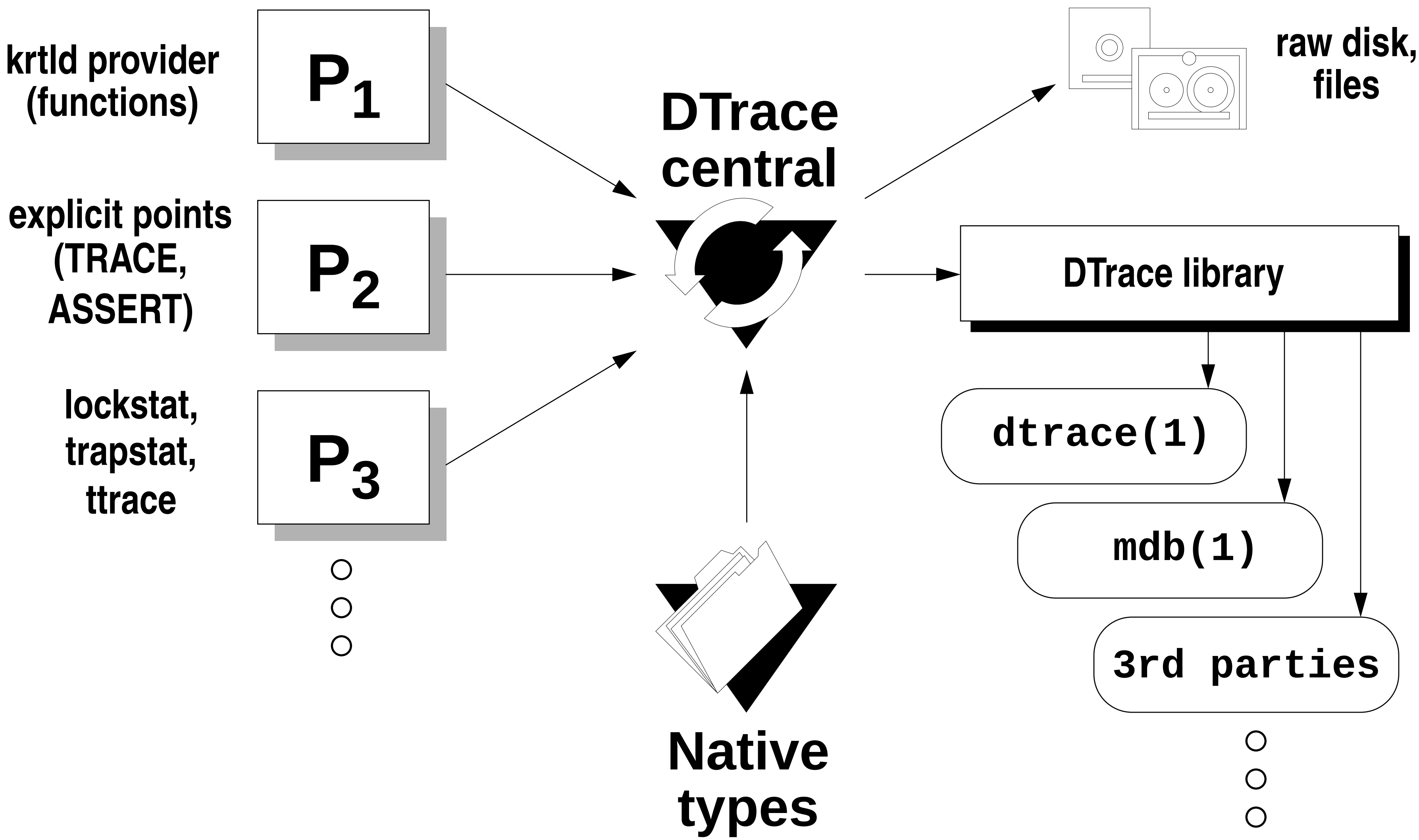
Principles of OS Tracing

- Must be part of production kernels
- Must support thousands of probes or more
- Must support specialized or constrained probe sites
- Must allow for selectively enabling probes
- Must have near-zero overhead when disabled
- Simple programming APIs for producers and consumers
- Effective documentation and namespace management

DTrace Concepts

- Providers — Back-end code that provides trace point locations and properties to the framework
- Points — Known trace locations that can be enabled or disabled selectively (implicit or explicit)
- Actions — Primitives that can be associated with tracepoints (arguments, stack trace, breakpoint, panic)
- Predicates — Conditional statements that can logically prefix trace points

DTrace Architecture



Trace Point Providers

- Essentially all functions can act as trace points
- Need to be able to handle module load and unload
- Specialized providers can *publish* custom trace points, including the set of actions that can be performed:
  - lockstat can essentially become a DTrace provider
  - trapstat (trap table dynamic instrumentation tool)
  - ttrace (tlb dynamic instrumentation tool)
- Explicit TRACE() points require compiler support

Traditional Trace Point

- D-cache hot but not very flexible:

```
if (tracing_on)
        do_trace(arg, ...);
```

- D-cache cold but more flexible:

```
if (this_trace_point_on)
        do_trace(arg, ...);
```

- Neither implementation helping I-cache footprint

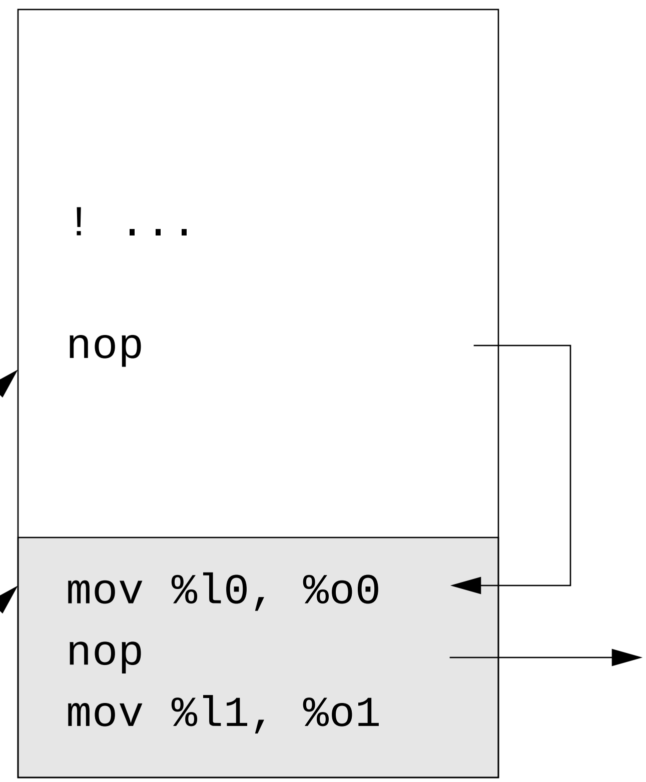
DTrace Trace Point

- Pragma or macro triggers special compiler support

```
int x;
char *y;

/* ... */

DTRACE(arg, ..., x, y);
```



Single no-op identifies code patch point

Trampoline code out of hot i-cache path

DTrace Central

- Providers *publish* trace points, default actions, and supported actions to framework
- Framework requests that provider enable or disable particular points, tracks actions and predicates
- Exports data through device to library or directly to on-disk files or raw devices
- Framework also supplies library with information on location, stability, specificity, and purpose of each point

Buffer Management

- Alternate — provider switches buffers, producing data to one while another is consumed
- Circular — ring buffer wraps around, final contents exported when tracing is explicitly disabled
- Single — tracing disabled when buffer fills
- Library can be used to select buffer policy

Native Types

- Type information is associated with kernel binaries using compiler-generated debugging stabs
- Type information is also associated with point arguments
- Set of conversion functions provides mapping between user types (e.g. PID, filename) and kernel types (e.g. proc_t *, vnode_t *)
- Meaningful predicates can be constructed as long as necessary mapping functions are available

Programming API

- We provide basic tool, C programming API, and possibly Perl scripting support as well
- Easy to write higher-level tools to control tracing facility
- Easy to integrate into existing tools (e.g. Symon, mdb)
- Easy to integrate into 3rd party tools (BMC, TeamQuest)

Long-Term Goals

- truss -k — follow system calls into the kernel and back
- Inject faults into kernel code
- Enable ASSERT() macros in the field
- DEBUG kernels no longer required (really just an /etc/system file or flag that enables a set of traces)
- Robust catalogs of system behavior that we can use to guide decisions, product directions

Sun microsystems