How To Use Event Tracing For Windows For Performance Analysis

Outline

- Why use Event Tracing?
- How to use Event Tracing
- Event Tracing vs. PerfCounters
- What events should be logged
- An example
- The kernel logger
- The Future of ETW

Goals

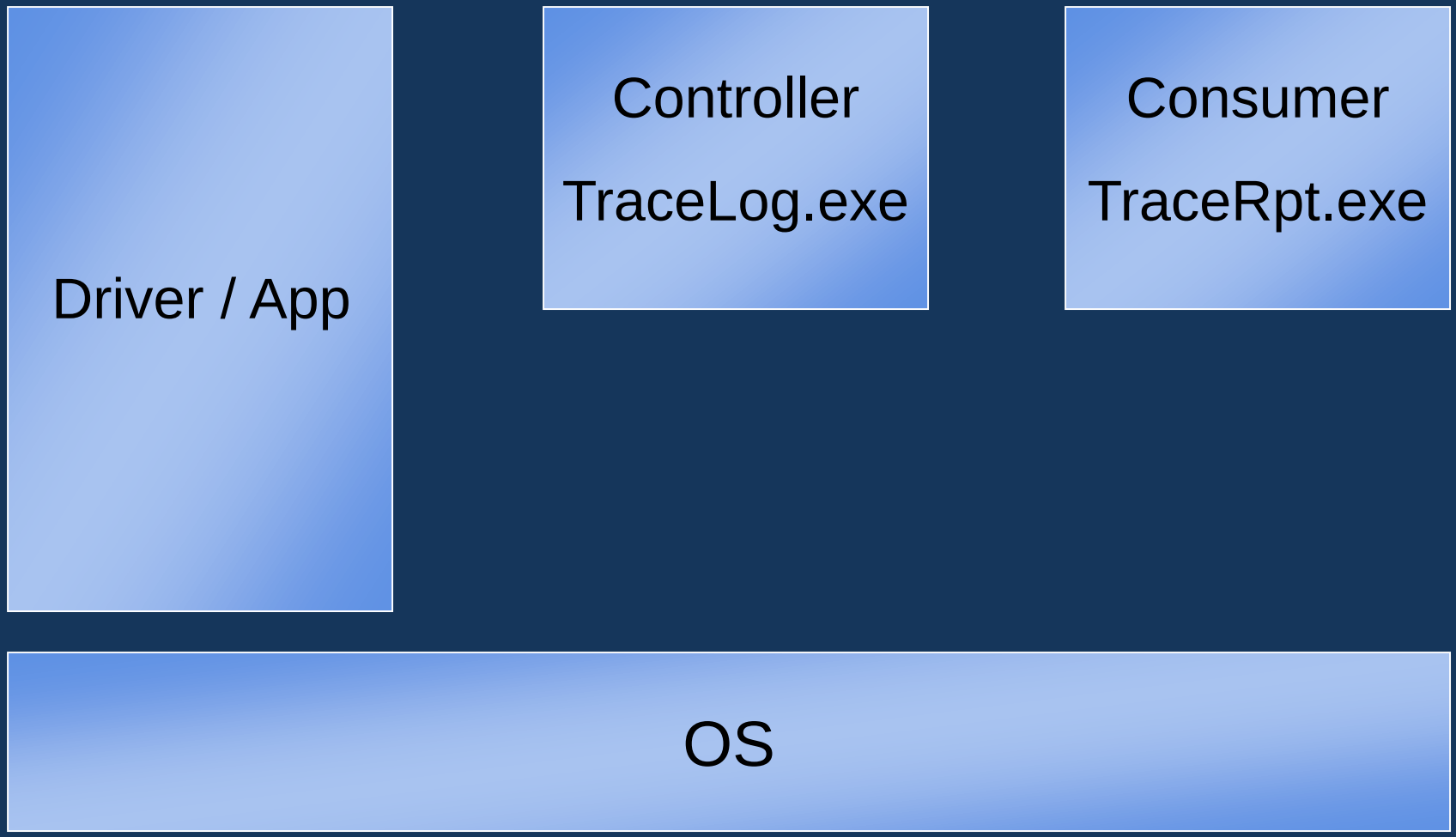
- A better understanding Event Tracing for Windows
- How to use Event Tracing to increase driver quality
- How to help your customers use your software efficiently

Why ETW?

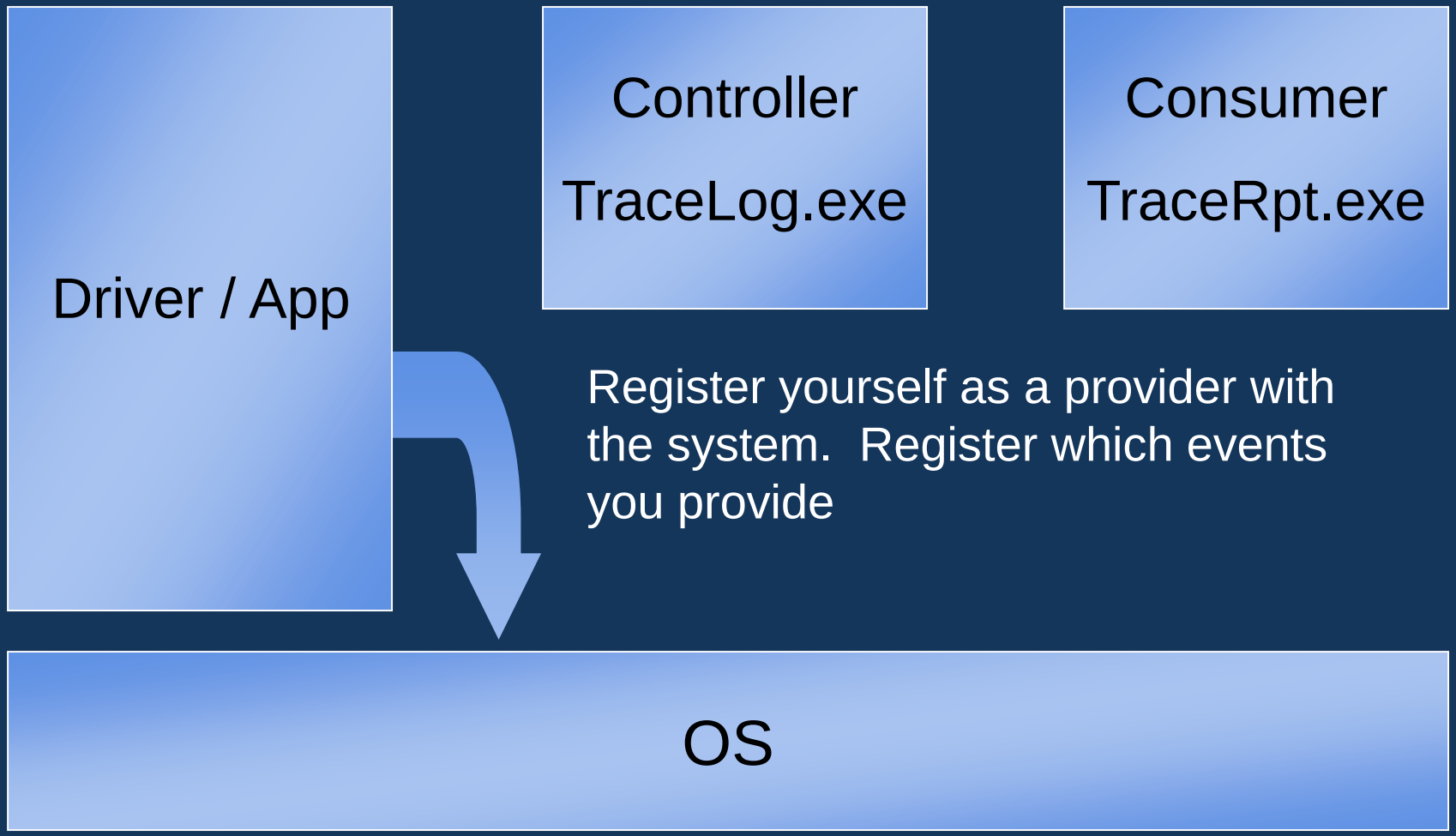
- Unified logging facility provided by the OS
  - Provides holistic view of the system
- High speed
  - 1200 to 2000 cycles per logging event
- Low overhead
  - Less than 5% of the total CPU cycles for 20,000 events/sec
- Works for both user mode applications and drivers
- Tracing sessions and event provider separated
- Dynamically enabled or disabled
  - Designed to allow tracing of production code

How Event Tracing Works

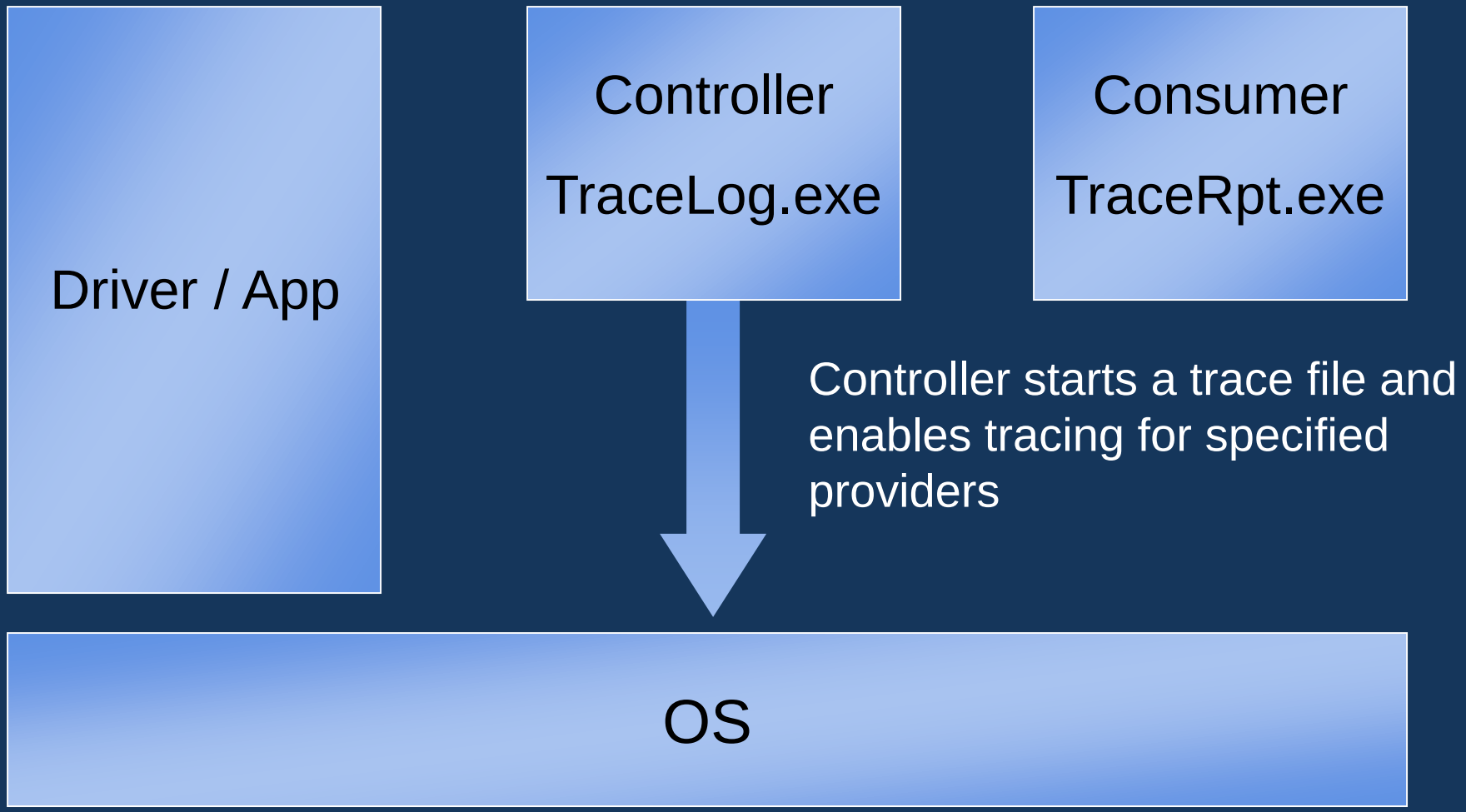
Event Tracing Layout



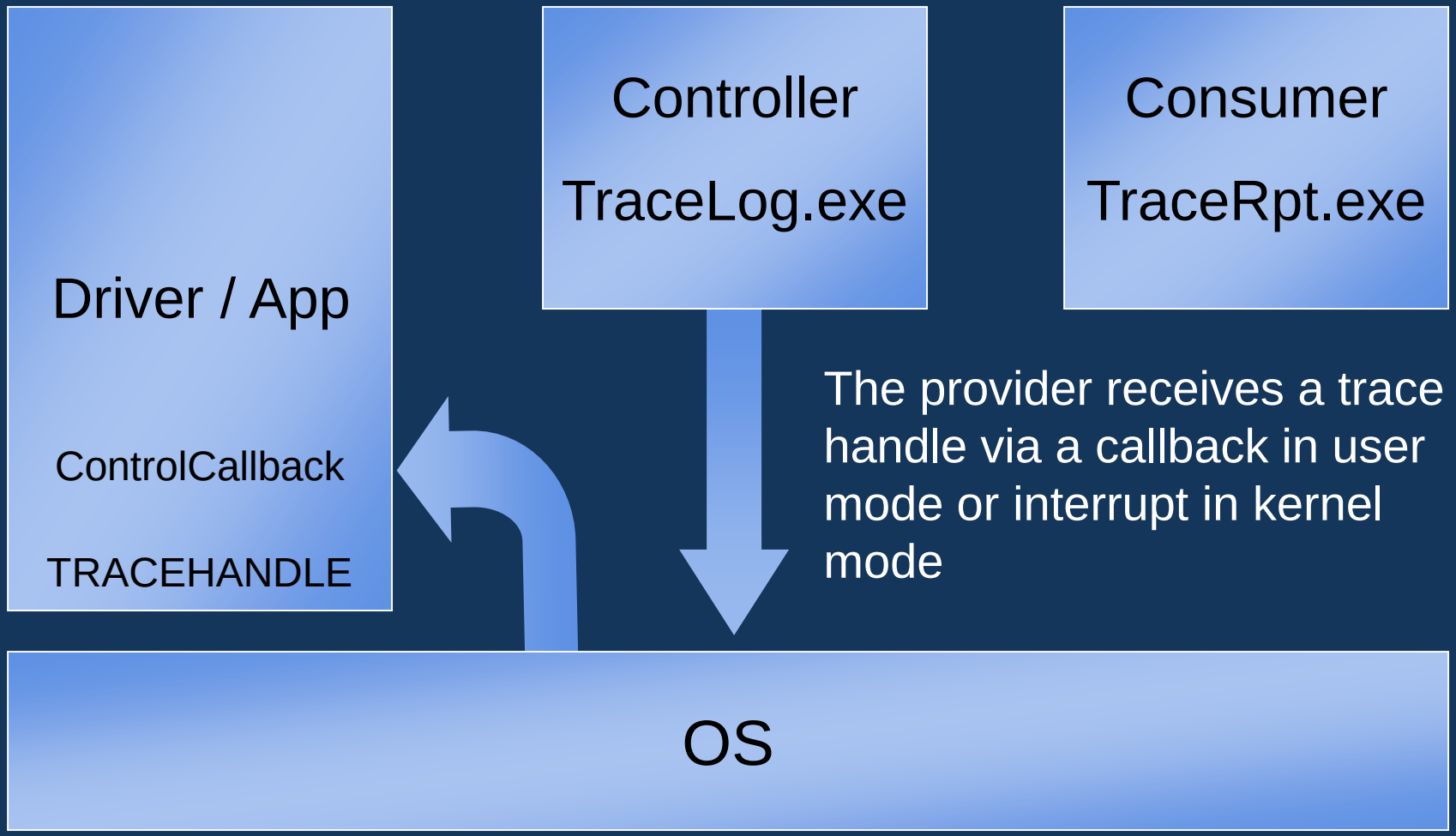
Provider Registration



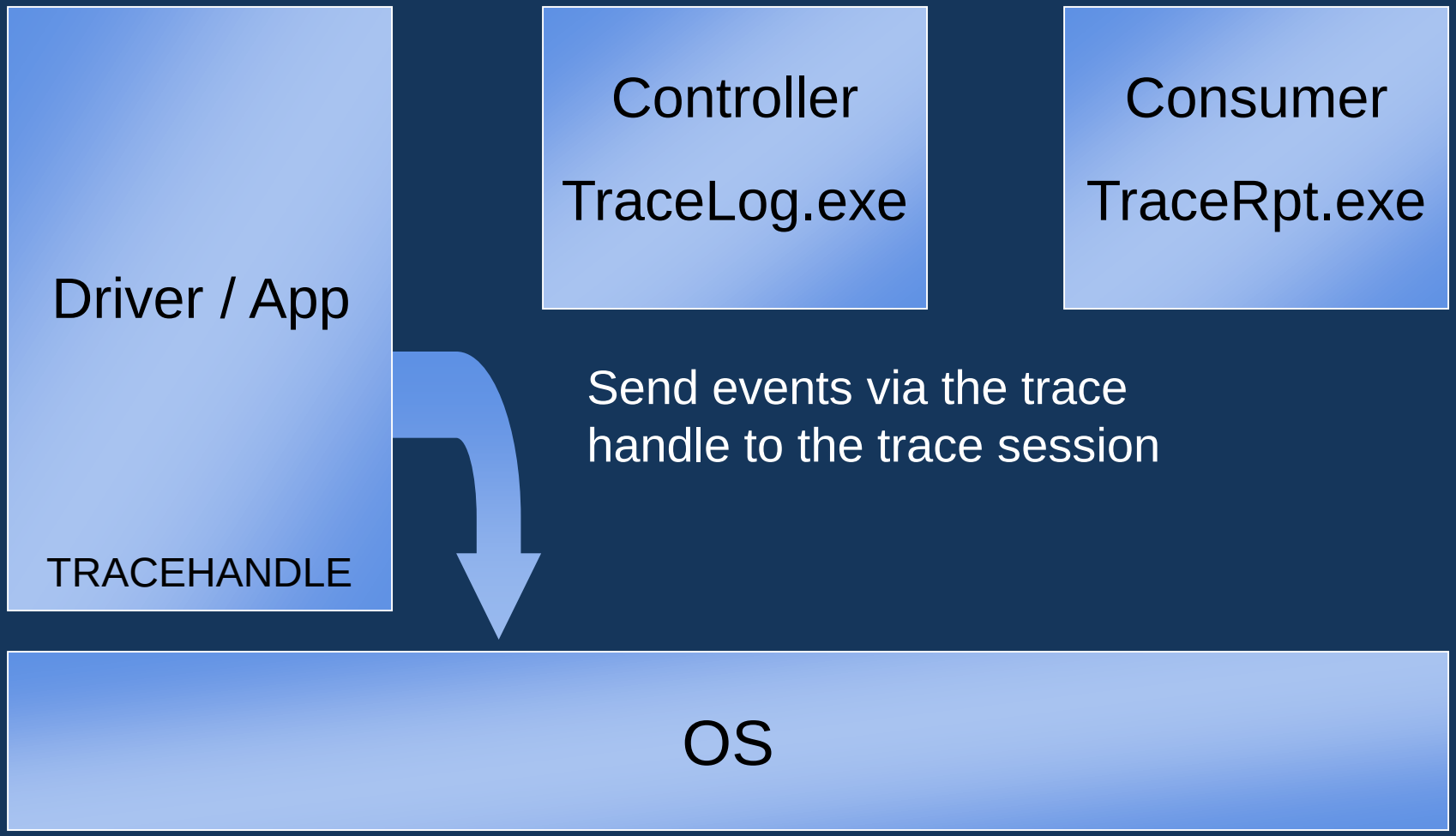
Enabling Tracing



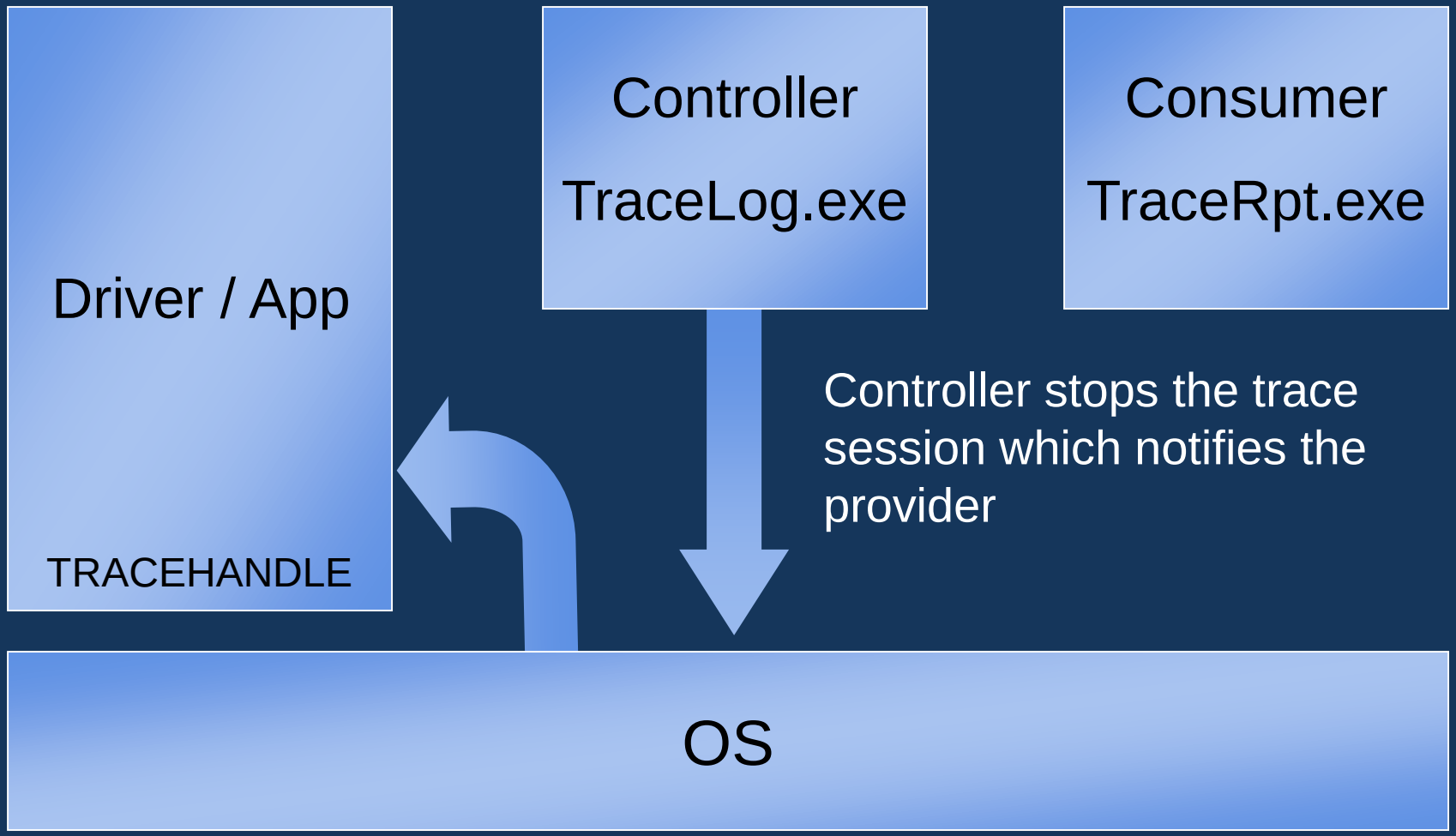
Enabling Tracing



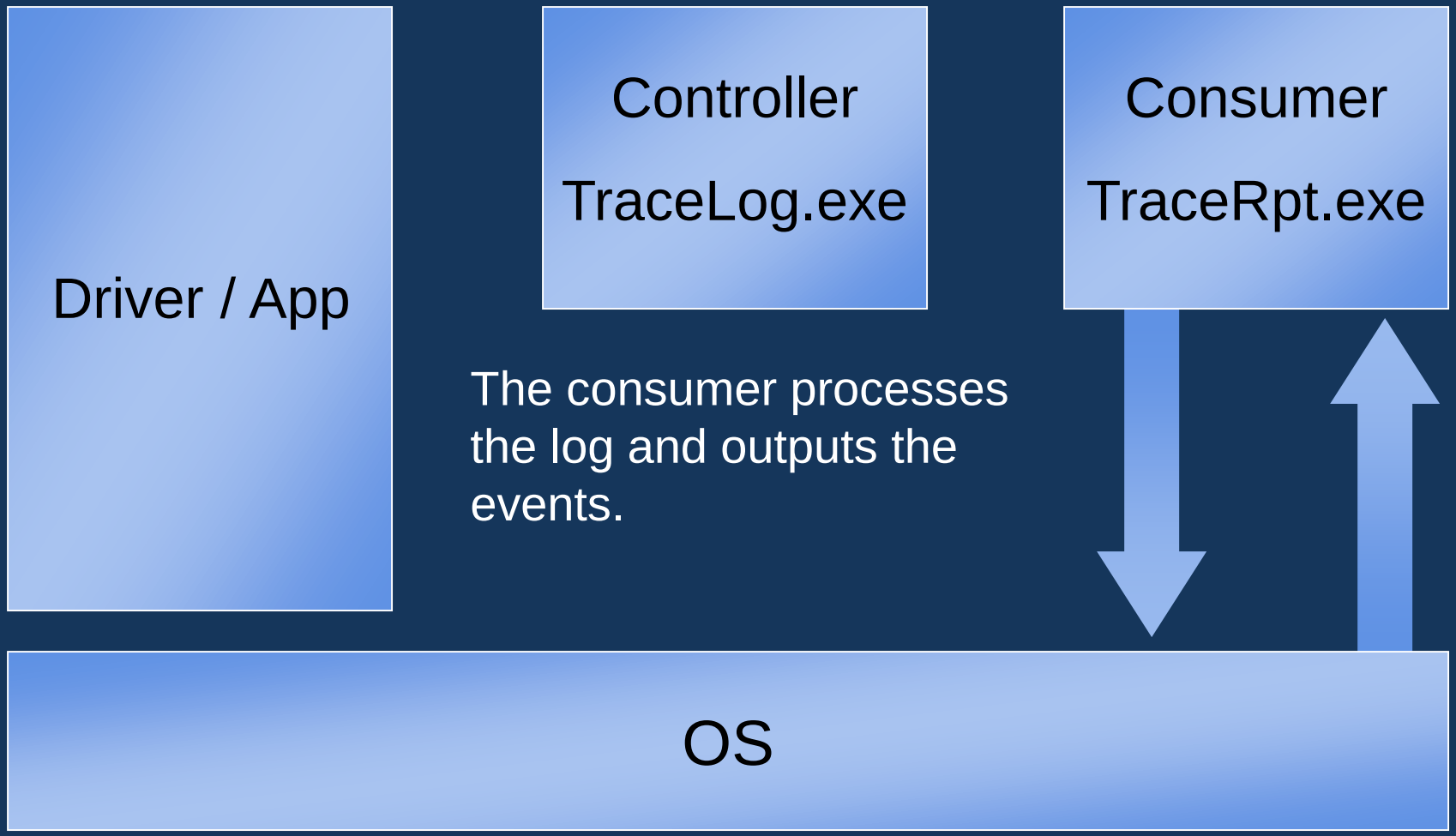
Sending Events



Stop Tracing



Processing the Trace



Events vs. PerfCounters

Events vs. PerfCounters

- Events
  - Discrete Events
  - Accurate CPU utilization
  - Freeform data
  - Detailed system info
  - ISR/DPC info
- PerfCounters
  - 100ms sampled
  - Aligned to system timer
  - Restricted by API
  - High-level diagnostic
  - Less overhead for continuous event

Logging Events

Logging Events

- State Changes
- Begin/End of significant operations
- Resource creation/deletion
- Other events related to performance or reliability
- Debug events

Event Header Layout

- Event header is required for all ETW events
- GUID for the Event Class
- UCHAR for the Event Type
- USHORT for Version

Flags and Levels

- Developer-defined values to control event generation of the provider
- The Provider gets the current flag and level from the Controller
- Flags are logical groupings of events
- Levels are gradations of severity
  - Ex Debug events would be high level. Used only in exceptional cases.

An Example ETW Application

Creating Events

- Assign a GUID to each provider
  - referred to as the ControlGUID
- Create an Event structure
- Assign a GUID to each event class
- Assign a UCHAR to each event type
- Create a MOF for each event type
  - On Windows XP and above must be compiler with mofcomp.exe
- Create a ControlCallback function for each provider

My Events

```
DEFINE_GUID( MyEventsGUID, x );

const UCHAR Event1Start = 0;
const UCHAR Event1End   = 1;
const UCHAR Event2Start = 2;
const UCHAR Event2End   = 3;

typedef struct _MyEvent1 {
    EVENT_TRACE_HEADER m_Header;
    UINT m_uMyData;
    WCHAR m_wsMyString[ 256 ];
} MyEvent1;

typedef struct _MyEvent2 {
    EVENT_TRACE_HEADER m_Header;
    UINT_PTR m_cMyPointer;
} MyEvent2;
```

MOF Description

Provider Class MOF

```
#pragma classflags( "forceupdate" )
#pragma namespace( "\\\\.\\Root\\WMI" )

[Dynamic,
 Description("ETW Example Provider") : amended,
 Guid("{FDAF6C10-8530-4e23-9D28-715CB763768E}"),
 locale("MS\\0x409")
]

class ExampleProvider:EventTrace
{
};
```

Event Class

```
[Dynamic,
 Description("ETW Example Events") : amended,
 Guid("{D3DD533F-9B62-4e78-8747-AAC84E75F5D0}"),
 DisplayName("ETW Example Events") : amended,
 locale("MS\\0x409")
]
class ExampleEventsClass:ExampleProvider
{
};
```

Example Event 1 MOF

```
[Dynamic,
  Description("Example Event 1") : amended,
  EventType{0, 1},
  EventTypeName{"Event1 Start", "Event1 End"} : amended,
  DisplayName("Example Event 1") : amended,
  Version(0), locale("MS\\0x409")]
class ExampleEvent1:ExampleEventClass
{
    [WmiDataId(1), Description("LoopCount") : amended,
     read]
     uint32  LoopCount;
    [WmiDataId(2), Description("MyString") : amended,
     StringTermination("NullTerminated"), format("w"),
     read]
     string MyString;
};
```

Example Event 2 MOF

```
[Dynamic,
  Description("Example Event 2") : amended,
  EventType{2, 3},
  EventTypeName{"Event2 Start", "Event2 End"} : amended,
  DisplayName("Example Event 2") : amended,
  Version(0),
  locale("MS\\0x409")]
class ExampleEvent2:ExampleEventClass
{
    [WmiDataId(1), Description("MyPointer") : amended,
     pointer,
     format("x"), read]
     uint32 MyPointer;
};
```

Registering MOF Descriptions

- Compile your MOF on installation of your component
- Mofcomp.exe MyEvents.mof

Control Callback

```
TRACEHANDLE    g_hTrace                    = NULL;
BOOL           g_bTracingEnabled           = FALSE;

ULONG WINAPI MyControlCallback(
    WMIDPREQUESTCODE RequestCode, PVOID Context,
    ULONG* Reserved, PVOID Buffer )
{
    if( RequestCode == WMI_ENABLE_EVENTS )  {
        g_hTrace = GetTraceLoggerHandle( Buffer );
        g_bTracingEnabled = TRUE;
        SetEvent( g_hStartEvent );
    } else if ( RequestCode == WMI_DISABLE_EVENTS )  {
        g_bTracingEnabled = FALSE;
    }
    return 1;
}
```

Registering and Sending Events

```
Status = RegisterTraceGuids(
          MyControlCallbck,
          NULL,
          MyControlGuid,
          0, NULL, NULL,
          NULL,
          &RegistrationHandle);


if ( g_bTraceEnabled ) {
  Status = TraceEvent(
            TraceHandle,
            MyEvent );
}

UnregisterTraceGuids(
           RegistrationHandle );
```

- Register with ETW
  - Provide ControlGuid and Callback function
- On callback set/clear global flag *TraceOn*
- Instrument code at appropriate places
  - Check Trace Flag
  - Call TraceEvent
- Unregister

Example Run

- Start ETWProvider.exe
- TraceLog.exe -f MyLog.etl -guid TraceGuids.txt -start MyLog
- Run for a while
- TraceLog.exe -stop MyLog
- Tracerpt.exe MyLog.etl

Results

Summary.txt

```
Files Processed:
    MyLog.etl
+-----------------------------------------------------------+
|Event Count   Event Name            Event Type   Guid      |
+-----------------------------------------------------------+
|          1   EventTrace            Header       {68fdd900…}
|         92   ETW Example Events    Event1 Start {d3dd533f…}
|         91   ETW Example Events    Event1 End   {d3dd533f…}
|         91   ETW Example Events    Event2 Start {d3dd533f…}
|         91   ETW Example Events    Event2 End   {d3dd533f…}
+-----------------------------------------------------------+
```

Dumpfile.csv

```
ETW Example Events, Event1 Start, 6190806, 0, "This is my Event1 String"
ETW Example Events, Event2 Start, 6658869, 0x0012FF3C
ETW Example Events, Event1 End,   7594995, 1, "This is my Event1 String”
ETW Example Events, Event2 End,   8219079, 0x0012FF40
```

Kernel Mode / Managed Code Providers

Kernel Mode Providers

```
Status =
 IoWMIRegistrationControl(
        pDeviceObject,
        WMI_ACTION_REGISTER );


if ( g_bTraceEnabled ) {
    Status = IoWmiWriteEvent(
                WmiDataBlock );
}

Status =
 IoWMIRegistrationContol(
       pDeviceObject,
       WMI_ACTION_DEREGISTER );
```

- Register driver with WMI
- Process WMI IRP
  - IRP_MJ_SYSTEM_CONTROL
- Instrument code at appropriate places
  - Check Trace Flag
  - Call IoWmiWriteEvent
- Unregister

Managed Code

```
Guid SampleGuid = new Guid("…");
TraceProvider myProvider =
    new TraceProvider (
                "Sample",
                SampleGuid );
```

- TraceProvider class handles Registration and callback with ETW

```
if ( myProvider.enabled ) {
       MyProvider.TraceEvent(
              TransactionGuid,
              EventType.Start,
              arg1, arg2 );
}
```

- Instrument code at appropriate places
  - Check enabled
  - TraceEvent

The Kernel Logger

The Kernel Logger

- Special logger for kernel events
- Exclusively logged to by the OS
- Can be merged with other logs
- Global resource

Kernel Events

- Process and Thread creation/deletion
- Disk and File IO and Loader
- Memory faults
- Network Stack
- Registry Access
- Context Switch
  - For Windows XP, must register CSwitch.mof from CD
- ISR/DPC
  - Use –dpcisr switch to TraceLog.exe
- …

Enable the Kernel Logger

- Start ETWProvider.exe
- TraceLog.exe –f MyLog.etl –guid TraceGuids.txt –start MyLog
- TraceLog.exe –f KernelLog.etl –start
- Run for a while
- TraceLog.exe –stop
- TraceLog.exe –stop MyLog
- Tracerpt.exe MyLog.etl KernelLog.etl

Merged ETW Results

```
Files Processed:
    MyLog.etl
    KernelLog.etl
+-----------------------------------------------------------+
|Event Count   Event Name            Event Type   Guid      |
+-----------------------------------------------------------+
|         65   DiskIo                Write        {3d6fa8d4…}|
|         46   UdpIp                 Recv         {bf3a50c5…}|
|          2   HWConfig              Default      {01853a65…}|
|          1   HWConfig              CPU          {01853a65…}|
|          1   HWConfig              PhyDisk      {01853a65…}|
|          2   EventTrace           Header       {68fdd900…}|
|         92   ETW Example Events    Event1 Start {d3dd533f…}|
|         91   ETW Example Events    Event 1End   {d3dd533f…}|
|         91   ETW Example Events    Event2 Start {d3dd533f…}|
|         91   ETW Example Events    Event2 End   {d3dd533f…}|
|         10   Thread                Start        {3d6fa8d1…}|
|         12   Thread                End          {3d6fa8d1…}|
+-----------------------------------------------------------+
```

Merged ETW Results

```
…
DiskIo,                  Write,          0x001C, 598311281 …
UdpIp,                   Recv,       0xFFFFFFFF, 598467302 …
ETW Example Events, Event1 End,          0x0D90, 598779344 …
ETW Example Events, Event2 End,          0x0D90, 599247407 …
ETW Example Events, Event1 Start,        0x0D90, 600339554 …
ETW Example Events, Event2 Start,        0x0D90, 600807617 …
ETW Example Events, Event1 End,          0x0D90, 601743743 …
DiskIo,                  Write,          0x0868, 601743743 …
…
```

ETW Consumers

- Event Log (Windows codenamed “Longhorn”)
- RATT
- PIX for Windows (Q4)
- SysInternals TCPView and DiskMon
- VS Whidbey

ETW on Longhorn Overview

- Key piece of Longhorn Instrumentation Infrastructure
- New Enhanced and Simplified API
  - Enhanced discovery of instrumented components
  - One set of APIs for tracing (ETW) and eventing (Longhorn EventLog Service)
- New features and improvements for Longhorn
  - Provider security to protect sensitive data
  - Multiplexing of events to multiple consumers
  - Activity ID support for correlating events
  - TDH library for a unified way to decode events
- More Events from Windows components
- More Third-Party ETW produces and consumers

Call To Action

- Use tracing to understand what is happening on the system
- Instrument your software for performance analysis and debugging
- Ship ETW enabled software
- Consider exposing selected events for your customers and partners

Additional Resources

- Web Resources
  - Event Tracing Reference
    - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/about_event_tracing.asp
  - MOF Reference
    - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/data_types_for_event_data.asp
- WDK
  - Tools: TraceLog.exe
    - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ddtools/hh/ddtools/tracelog_b6beb1b9-7356-4975-8f53-2f2338ae1927.xml.asp
  - TraceRpt.exe available in windows\system32

Microsoft®

Your potential. Our passion.™