SQL Server on Linux, will it perform?

Slava Oks

Microsoft

Thank You!

Microsoft Research

Windows team

Midori

Our goal is to make SQL Server perform and scale on any platform or hardware of customers choice

Microsoft

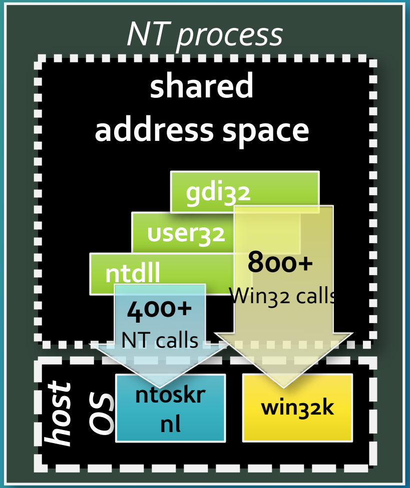
Prolog:
Meet the PALs

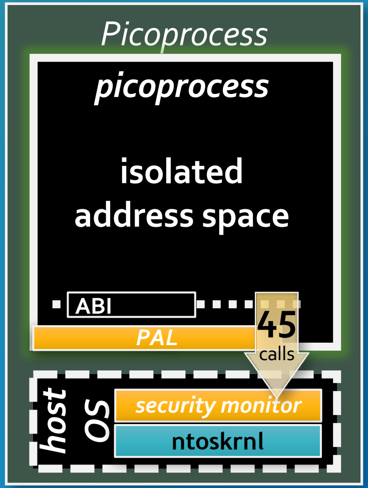
Microsoft

Intro to Drawbridge: A container technology to achieve isolation, security and density in the cloud

- Modified Windows Kernel to run in user mode, aka Library OS or LibOS
- Designed for running on Windows and leverages Pico-process feature
- Pico-process is a NT process with empty address space

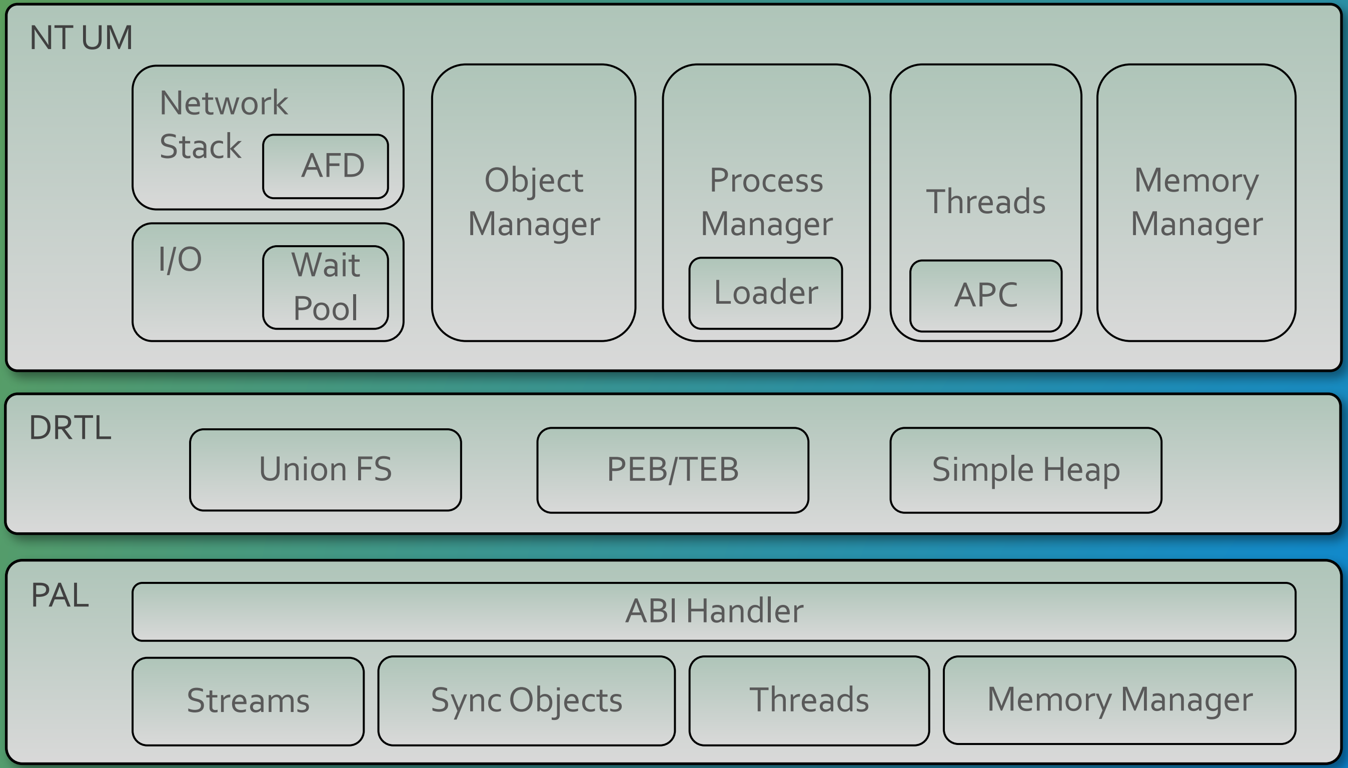
- All 1200+ system calls blocked from user-mode (NTOS and win32k)
    - Enforced by 35-line change to KiSystemServiceHandler
    - No perf impact to other processes — leverages “slow path” used by UMS
- 45 new system calls added to process (Drawbridge system calls)
- Even hard-coded traps can’t break out





Microsoft

LibOS: A user mode runtime library exposing semantics of Windows kernel



LibOS inside

- Storage Manager
  - Asynchronous I/O submitted to the host and registered with WaitPool threads
  - On completion WaitPool threads deliver I/Os to the original thread through APC
  - Original threads deliver I/Os to their final destination
- Network Manager
  - Custom version of AFD (WinSock semantics) with a thread pool
    - AFD Asynchronous I/O submitted to the host and registered with WaitPool threads
    - On completion WaitPool threads deliver I/Os to AFD threads through APC
    - threads deliver network requests to original threads initiated I/O through APC
    - Original threads deliver I/Os to their final destination
- I/O General
  - No proper support for Scatter/Gather

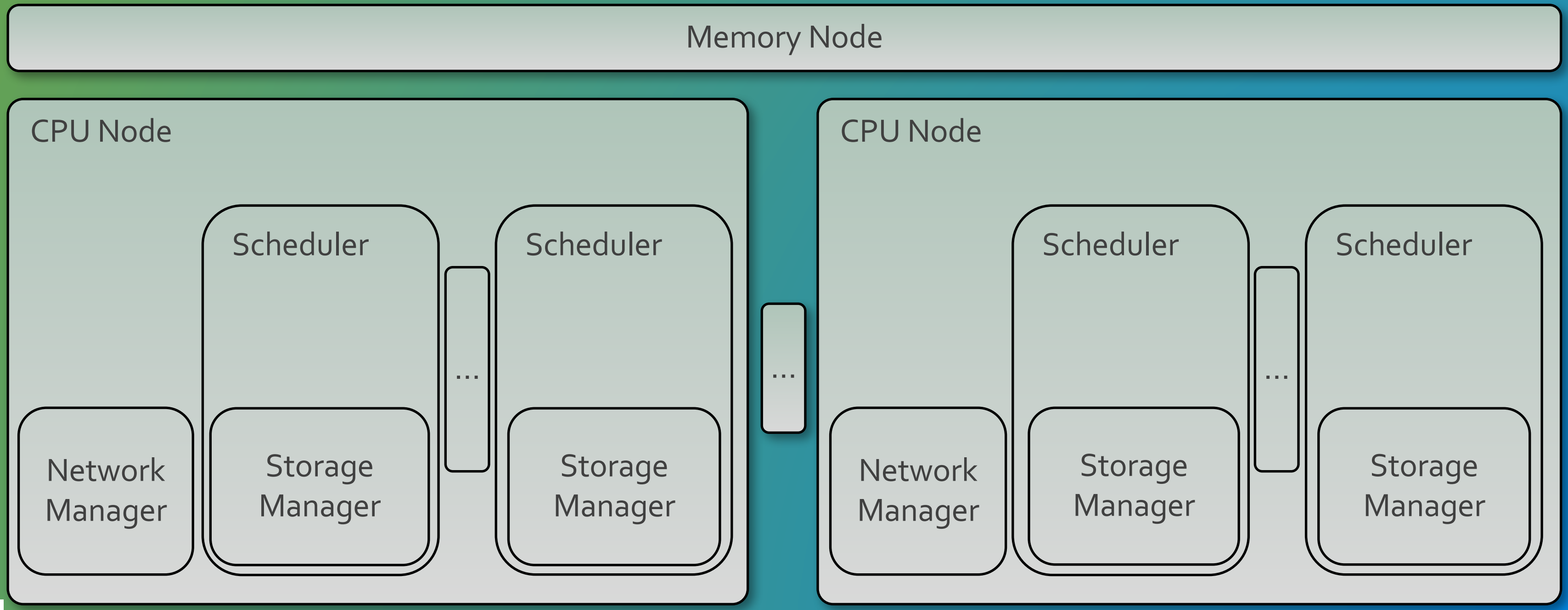
Microsoft

LibOS inside (cont.)

- Memory Manager
  - Global Virtual Address Descriptor (VAD) list
  - Global Heap
- Object Manager
  - Global Directory
- Process Manager
  - Per process runtime libraries – no image sharing
- Threads
  - APCs “injection” through polling

Microsoft

SQL OS (SOS): A user mode runtime library providing performance, scalability and diagnostic foundation for SQL Server



Microsoft

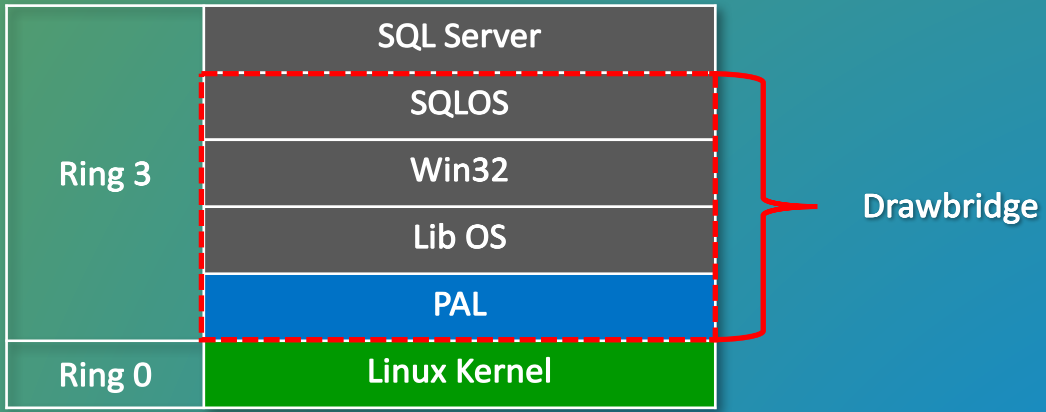
SQLOS inside

- Network Manager
  - I/O completion port/thread per CPU Node
  - Asynchronous delivery
- Storage Manager
  - I/O queue per scheduler
  - Synchronous delivery through periodic polling
- Memory Manager / Object Manager / Scheduling Manager
  - NUMA awareness
  - Partitioned heaps
  - Non-preemptive scheduling & User Mode Threads
  - Synchronization primitives

Chapter 1:
SQL & PALs
The marriage in heaven or…

Microsoft

SQL Server On Top Of PALs



Technologies	SQL	LibOS	Host Extension
Object Management	✔	✔	✔
Memory Management	✔	✔	✔
Threading/Scheduling	✔	✔	✔
Synchronization	✔	✔	✔
I/O (Disk, Network)	✔	✔	✔

Microsoft

Chapter 2:
The sign is on the wall
Introducing Intelligent Hacks

Fast Asynchronous Disk I/O

- Kernel aio
- Pump threads vs WaitPool threads
- Fast I/O

```
// We can do Fast I/O if and only if it follows rules employed by SQL Server
// disk I/O: which is delivered nonpreemptively through polling an overlapped
// data structure
//  - I/O is asynchronous
//  - No user mode APC required
//  - No I/O completion port specified
//  - Contains an event to be signaled (leveraged by SQL Server to wake up idle scheduler
//  - Disk I/O
//

canDoFastIO = WaitForCompletion == FALSE;

canDoFastIO = canDoFastIO && (ApcRoutine == NULL && FileObject != NULL);

canDoFastIO = canDoFastIO && (Args->SkipCompletionPort ||
                              NtpGetCompletionPortObject(FileObject,
                                                         &CompletionKey) == NULL);

canDoFastIO = canDoFastIO && (Args->EventObject != NULL && IoStatusBlock != NULL);

canDoFastIO = canDoFastIO && (NtpGetObjectType(Args->Object) == NTUM_FILE &&
                              NtpIsIoAsynchronous(Args->Object));

canDoFastIO = canDoFastIO && ((FileObject->Type & NtpSeekableFile) &&
                               (Type == NTUM_IO_READ ||
                                Type == NTUM_IO_WRITE ||
                                Type == NTUM_IO_WRITE_GATHER ||
                                Type == NTUM_IO_READ_SCATTER));

// If it is Gather/Scatter I/O then length can't exceed DK_UIO_MAXIOV supported by the Host
//
canDoFastIO = canDoFastIO && (!(Type == NTUM_IO_WRITE_GATHER ||
                                Type == NTUM_IO_READ_SCATTER) ||
                              Length <= DK_UIO_MAXIOV);
```

Microsoft

Fast Asynchronous Network I/O

- Pump threads vs WaitPool
- Fast I/O ~ AFD pass through
- SQLOS completion threads are pump threads ~ no context switch on completion

```
 // Complete I/Os received via the the IOPort are submitted to the I/O
 // completion port queue
 Status = NtpTryToProcessIoCompletion(IoCompletionPort,
                                      IoCompletionInformation);

 // Process any APCs or interruptions for this thread. //
 NtpProcessKernelApc(threadObject);

 Request.IOPort = IoCompletionPort->IOPort;
 Request.PendingIOs = &PendingIOs;

 Status = DrtlReadStreamSync(IoCompletionPort->Stream,
                             0,
                             0,
                             (PVOID)&Request,
                             NULL);

while (PendingIOs != NULL)
{
    //
    // Remember I/O to complete and move to the next I/O before
    // we complete the current one since by the time we return from
    // completion routine the completed I/O will be freed
    //
    CompletedIO = PendingIOs;
    PendingIOs = (PDK_ASYNC_RESULTS_LINKED)PendingIOs->Next;

    //
    // Complete I/O
    //
    NtpCompleteNetworkIoRequest((PNTUM_IO_REQUEST)CompletedIO->Request);
 }
```

Microsoft

Eliminate Global State

- Multiple Heaps
- I/O Request free list per thread
- Per process Virtual Address Space Manager
- NUMA support
- Processor Affinity

```
PVOID
DrtlAllocate(
    __in ULONG  Flags,
    __in SIZE_T Size,
    __in ULONG  Tag
    )
{
    ULONG heapIdx;

    //
    // Early boot we might not have a thread
    //
    heapIdx = DrtlGetCurrentThreadId() % g_DrtlNumberHeaps;

    return DrtlpAllocate(&g_DrtlHeaps[heapIdx], Flags, Size, Tag);
}

NtpAllocateIORequestRaw(
    __in NTUM_IO_TYPE Type)
{

    // Use cache if we have i/O request
    //
    LocalRequest = (PNTUM_IO_REQUEST)ExpInterlockedPopEntrySList(
            &RequestingThread->IORequestsCache);

        // If the cache was empty allocate a new request structure.
        //
        if (LocalRequest == NULL)
        {
            LocalRequest = (PNTUM_IO_REQUEST)ExAllocatePoolWithTag(
                PagedPool,
                sizeof(*LocalRequest),
                ' PRI');
        }
```

Microsoft

Chapter 3:
Pressure is On

Tracking Workloads

<u>Hardware Configuration</u>
Power Settings: OS Control power option, High Performance in OS, HT OFF, Turbo boost OFF
Network: 1x10 GB Network connection per machine
Machine configuration (server and client): Gen3 systems
Model/Processors: Intel Xeon CPU E5-2660 0 @ 2.20 GHz (2S/16C), Memory: 128 GB
Storage: 4x447.13 GB SSDs. All SSDs are striped together and mounted as 1 volume. Both data and log are stored on this volume.

Test Configuration: OLTP Database: 4000 Warehouse Size: 300GB, Total users: 400

				Metric: TransactionsPerSecond					
Case Id	Platform	Run Date	Version	Release Goal	Result	% of Goal	CoV (%) over last 30 days	Diff (%) from Avg	Result Trend
586	RHEL	3/1/2017 6:15:17 PM	14.0.405.33	21500.00	15102.00	70%	27.89	33.80	
556	Windows	3/2/2017 8:58:53 AM	14.0.405.6292	21500.00	17274.00	80%	17.28	-15.32	

Test Configuration: DW Database: 2000 Warehouse DB Size: 700GB, Total user: 1

				Metric: GeometricMean					
Case Id	Platform	Run Date	Version	Release Goal	Result	% of Goal	CoV (%) over last 30 days	Diff (%) from Avg	Result Trend
511	RHEL	3/1/2017 7:11:03 PM	14.0.405.33	12000.00	10985.80	109%	14.12	-1.70	
516	Windows	3/2/2017 11:08:13 AM	14.0.405.6292	12000.00	12023.30	100%	8.30	-2.13	

Test Configuration: InMem Database: Scale Factor 400 Size: 70GB, Total user: 600

				Metric: TransactionsPerSecond					
Case Id	Platform	Run Date	Version	Release Goal	Result	% of Goal	CoV (%) over last 30 days	Diff (%) from Avg	Result Trend
518	RHEL	3/1/2017 7:24:06 PM	14.0.405.33	80000.00	60008.80	75%	12.93	10.50	
517	Windows	3/2/2017 11:39:16 AM	14.0.405.6292	80000.00	82698.10	103%	1.09	-2.17	

Microsoft

Tracking Workloads

Hardware Configuration
Power Settings: OS Control power option, High Performance in OS, HT OFF, Turbo boost OFF
Network: 1x10 GB Network connection per machine
Machine configuration (server and client): 4S systems (for TPCC test)
  Model/Processors: Intel Xeon CPU E7-4850 0 @ 2.00 GHz (4S/40C), Memory: 768 GB
  Data Storage: 2x1.46 TB GB Fusion IO disk. All disks are striped together and mounted as 1 volume.
  Log Storage: 1x5.54 TB HDD

(4S) Test Configuration: OLTP Database: 12000 Warehouse Size: 900GB Total users: 1000									
			Metric: TransactionsPerSecond						
Case Id	Platform	Run Date	Version	Release Goal	Result	% of	CoV (%) over	Diff (%)	Result Trend
10774	**RHEL**	3/2/2017 6:46:27 AM	14.0.405.33	38000.00	**15762.00**	**41%**	**50.79**	**18.65**	
10776	**Windows**	3/2/2017 5:15:09 AM	14.0.405.595 7	38000.00	**34530.00**	**91%**	**6.89**	**-4.36**	

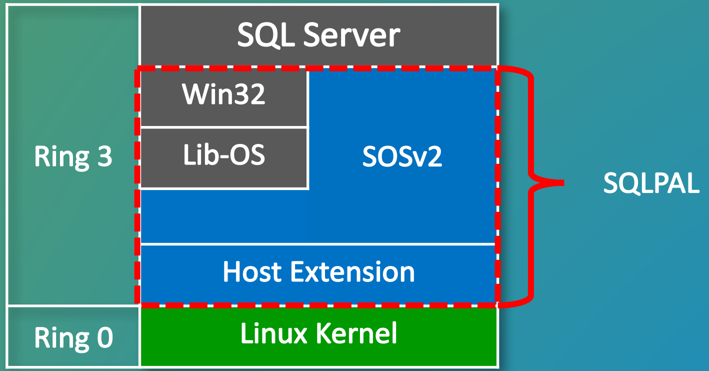
Microsoft

Chapter 4:
The ultimate PAL

Introducing SQLPAL

Principles:

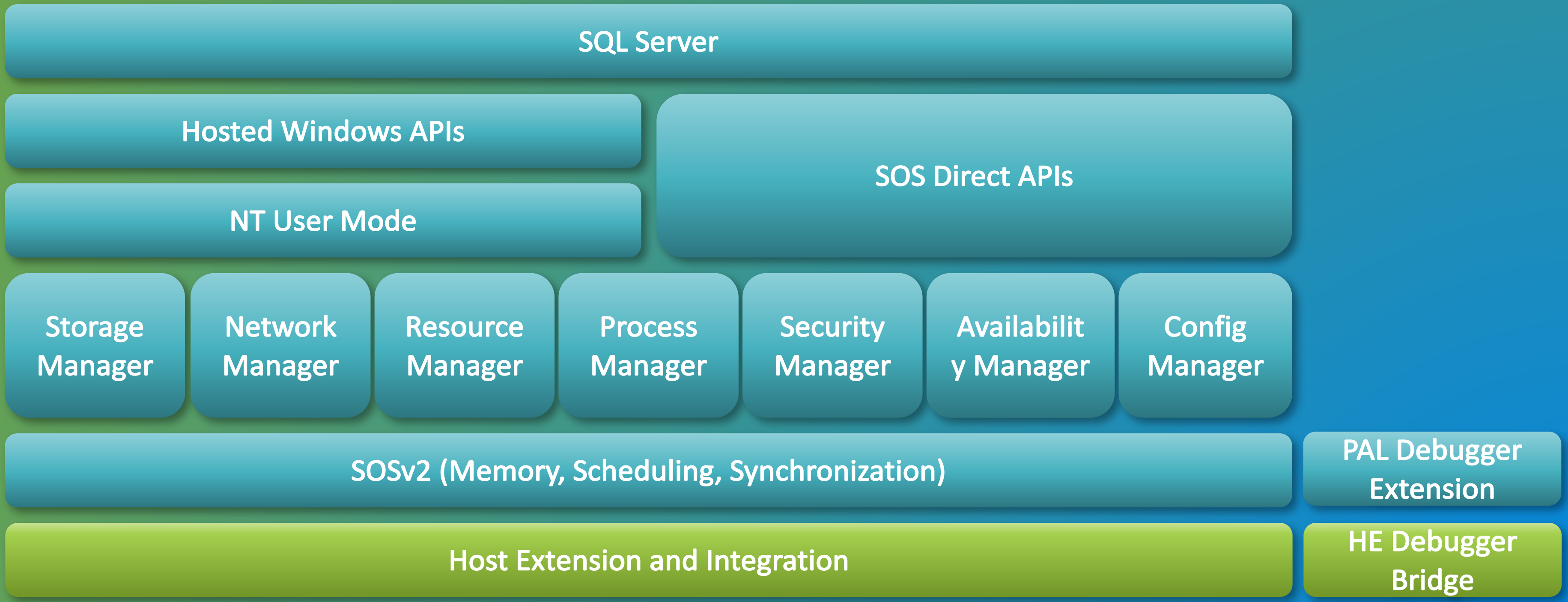
- Remove redundancy
- Optimize Performance critical paths (I/O)
- Shrink code path-length LibOS and Win32



Technologies	SQL	SOSv2	Host Extension
Object Management	✗	✔	✗
Memory Management	✗	✔	✔Host translation (jemalloc)
Threading/Scheduling	✗	✔	✔Host translation (pthreads)
Synchronization	✗	✔	✔Host translation (condition variables)
I/O (Disk, Network)	✗	✔	✔Host translation (kaio)

Microsoft

SQL PAL and SOSv2 Architecture

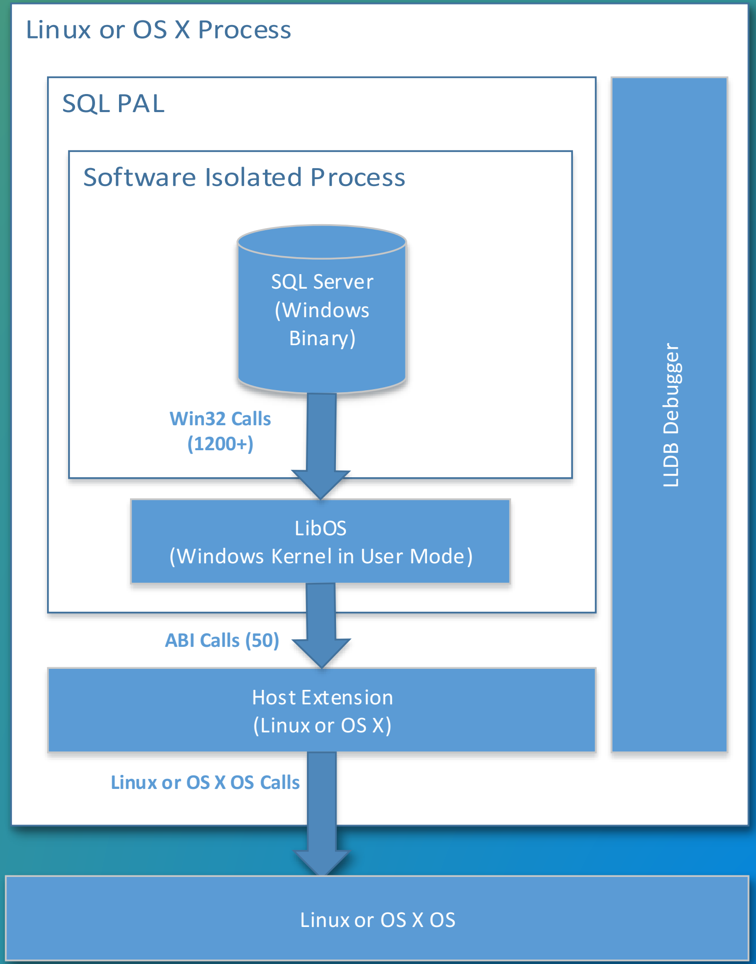


Microsoft

Chapter 5
Natural Habita(n)t

Linux Process Layout

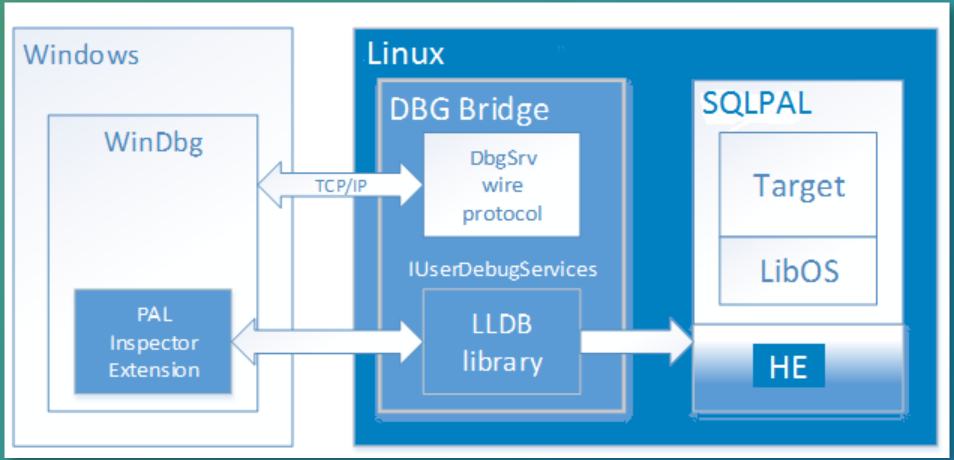
- Host Extension is native Linux process
- The Host Extension loads the SQLPAL native Windows library
- SQLPAL loads SQL Server into a virtual Windows Process.



Microsoft

Debugger

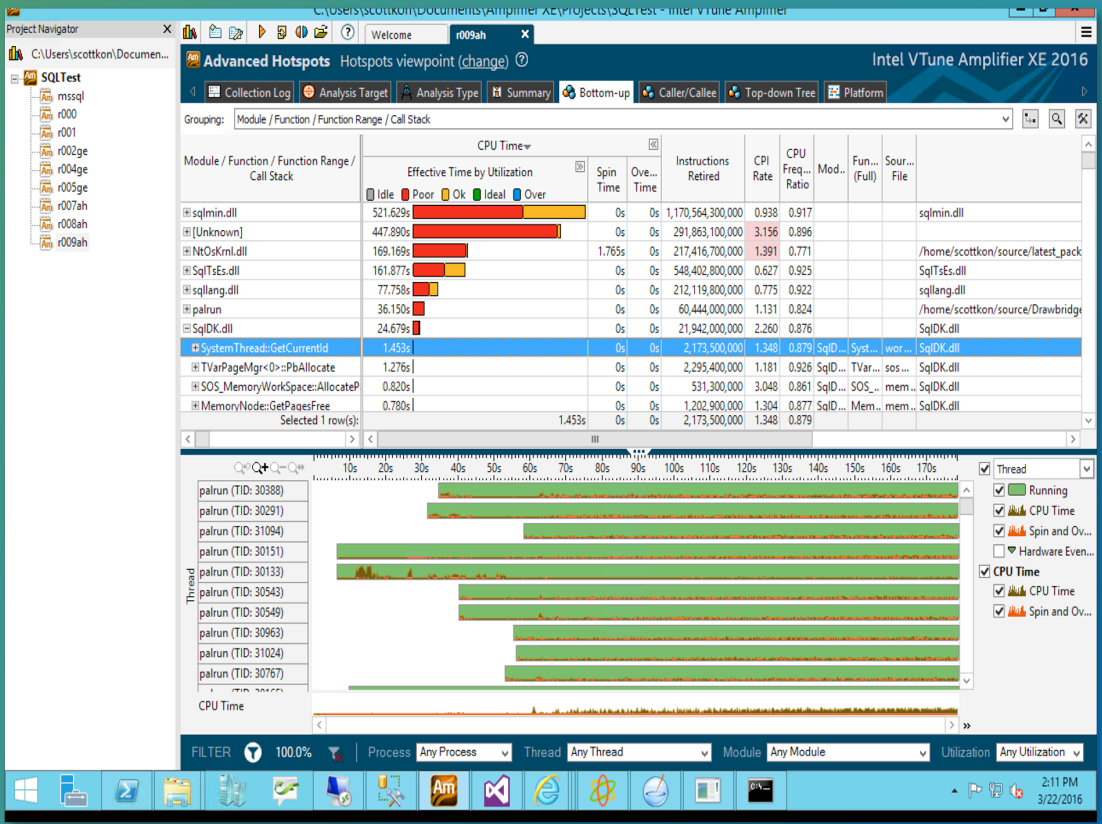
- Debugger bridge for Windbg
- For most scenarios debugging is identical to Windows
- Live Debugging
  - Start SQL on Linux under debugger bridge
  - Attach with Windbg
  - Dscripts etc. work same as against Windows
- Crash Dump
  - Run debugger bridge passing in crash dump file
  - Attach with Windbg and it’s the same as Windows
- Extract Windows dump from Linux Core dump
  - Able to extract a Windows dump from Linux core dump
  - Loses Linux information



- Linux Enlightenment
  - The debugger extension also adds commands to debug Linux parts of the PAL
  - Commands mirror normal Windbg commands
  - Examples:
    - ‘k’ shows Windows stack
    - ‘!k’ shows Linux stack
    - Same for dv (!dv), dt (!dt), etc.
    - Source can be listed and source stepping works

Intel® VTune™

- VTune is a cross platform performance tool
- Process
  - Capture on Linux and resolve on Linux
  - Copy the project to Windows
  - Resolve symbols and rerun analysis
  - This adds the Windows information to the project
- After processing all the code is available for analysis: Linux code, sqlpal.dll, Win32, and SQL



Chapter 6:
The game is ON

Thank You

Microsoft