

OpenVMS Update

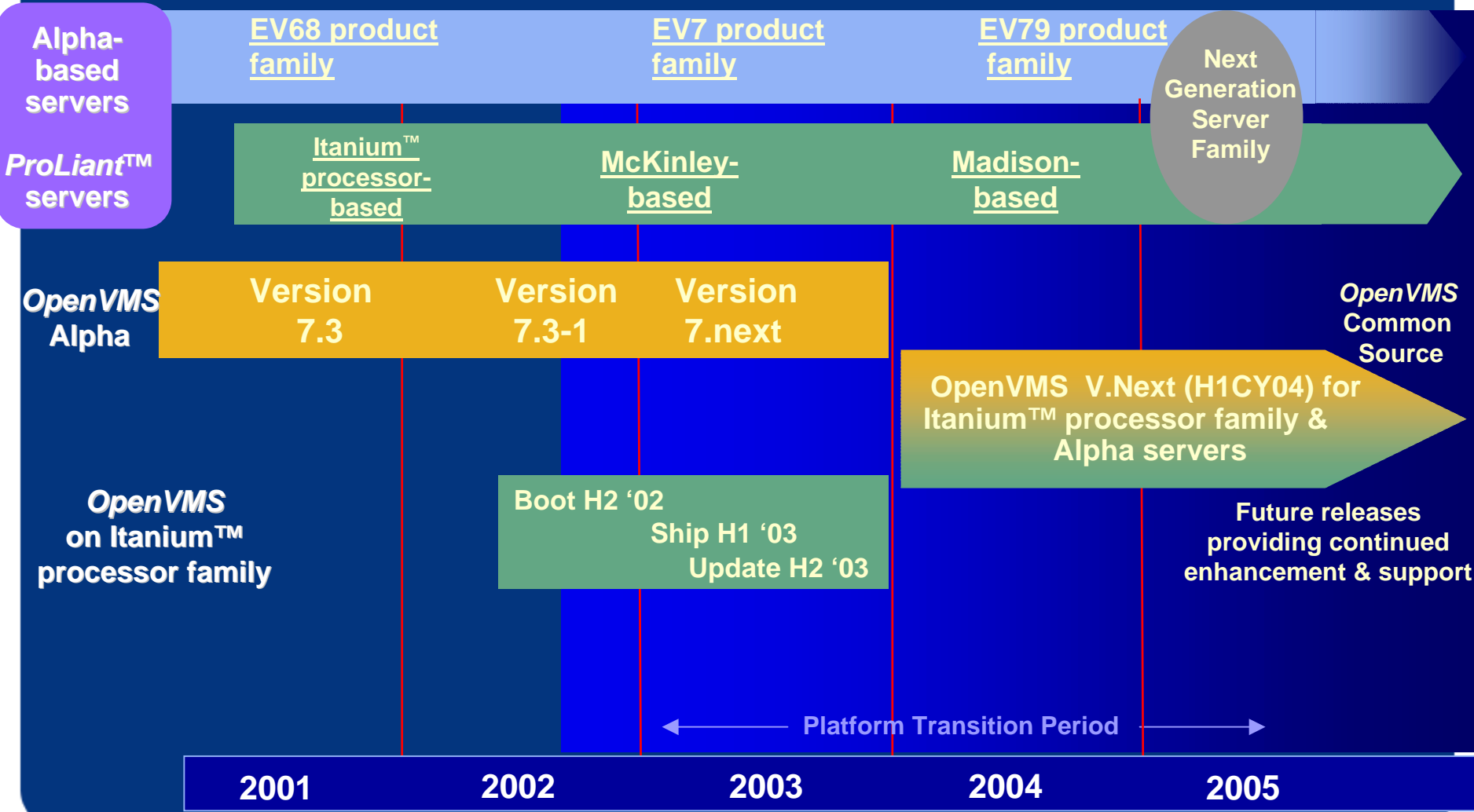
Porting to Itanium™ System Performance (Marvel) Future Plans

Christian Moser
OpenVMS Engineering
HP

christian.moser@compaq.com
moser@hp.com

June 2002

OpenVMS Roadmap for Alpha and Itanium™ Processor Family*



Itanium Development Schedule*

- Q4 2002 - Internal Layered Product Kit
- H1 2003 – 1st release – key ISVs, other partners, early adopters
- H2 2003 – 2nd release: key ISVs, other partners, early adopters
- H1 2004 – 3rd release: production quality

*Previously announced; subject to change

What version of *OpenVMS* is being ported?

- We are adding support to the *OpenVMS AlphaServer* code base for the Itanium™ processor family.
- We will build releases from the same sources for both AlphaServer and the Itanium™ processor family.
- A future release will support the Itanium™ processor family. Its version number cannot be predicted at this time.
- The first Itanium™ processor family release will reflect on-going *OpenVMS* development work.

VAX-to-Alpha vs. Alpha-to-Itanium™ processor family

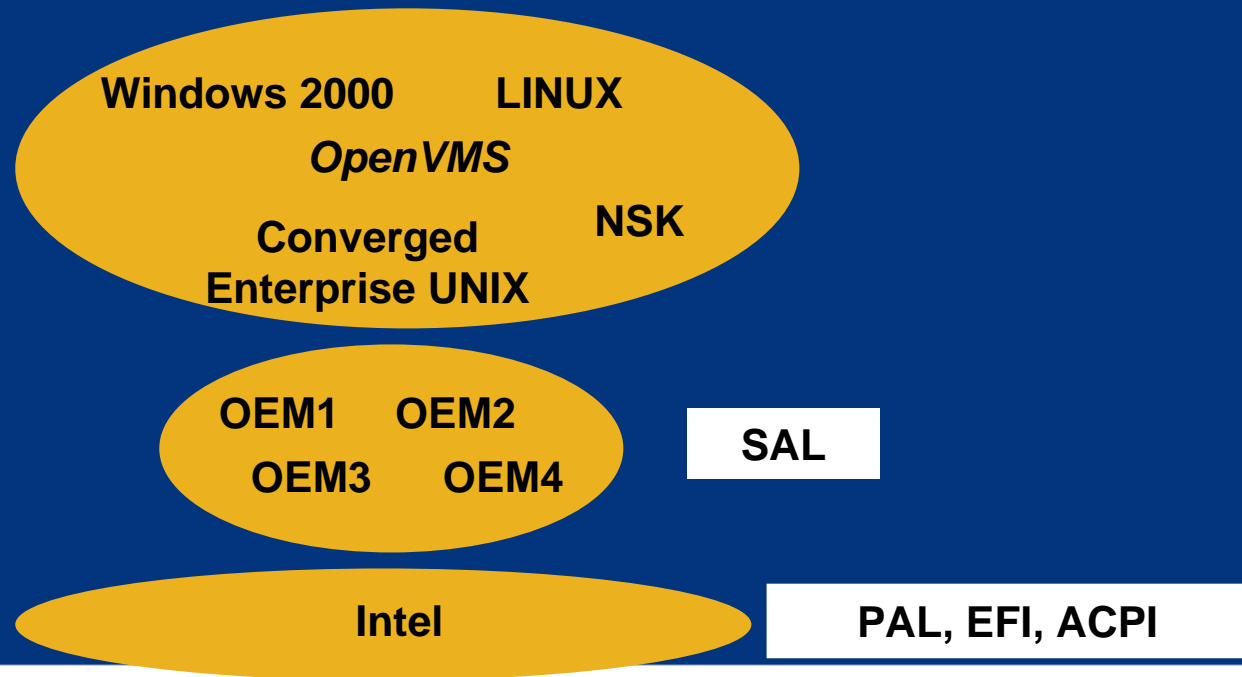
- VAX-to-Alpha: huge volume of coding work
 - AMACRO and 1100+ VAX MACRO-32 modules
 - 32b to 64b
 - data alignment
 - atomicity
 - multiple, out-of-order execution streams
- Alpha-to-Itanium™ processor family
 - much less coding but more complex

Big Challenges for the Base OS...

- No Alpha Console
- No Alpha PALcode
- Different primitives in the CPU
 - Register Conventions
 - Exception Handling
 - Interrupts
- Different Calling Standard
- ...but 4 processor modes like on VAX and Alpha
 - Kernel, Exec, Supervisor and User
- 3 C's

Console Architecture

- Processor Abstraction Layer (PAL)
- System Abstraction Layer (SAL)
- Extensible Firmware Interface (EFI)
- Advanced Configuration and Power Interface (ACPI)



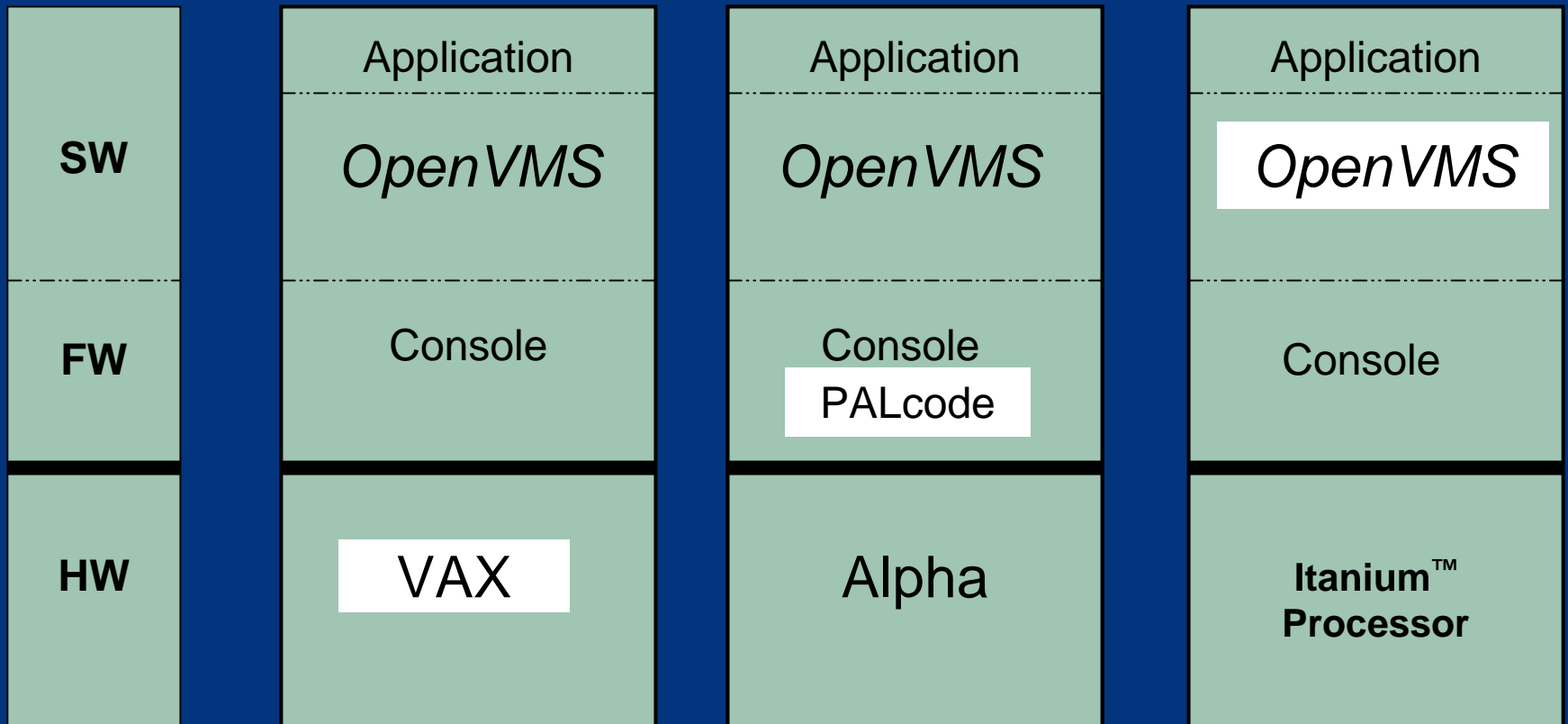
Console: Booting on EFI

- Requires “OS loader” to be in a FAT32 file partition
- *OpenVMS* implementation
 - A PC-style Master Boot Record overlays the ODS-2 “boot block”
 - The MBR contains a pointer to an ODS-2 container file which acts as the FAT32 partition
 - Our “OS loader” loads IPB
 - VMS_LOADER and IPB do on the Itanium™ processor family what the Alpha console and APB do on Alpha in preparing the system for SYSBOOT

Console: Special Run Time Services

- Operator Terminal I/O - *OpenVMS* will do it
- CPU Management, e.g. \$ STOP CPU
- Partitioning
 - Need “ownership of devices” info
 - Need “switch of ownership” capabilities
 - Need “notification of configuration changes”
 - Soft partitions (Galaxy) will be very challenging
- Crash Dump
 - Driver
 - Secondary CPUs halted to ‘console mode’

It's All in the Software



Privileged Architecture Library (PALcode)

- Alpha PALcode execution environment
 - Complete control of machine state
 - Interrupts disabled
 - I-stream mapping disabled
- A CALL_PAL is very expensive
- Not all functions need such complete control

PALcode Functions

- Instructions
 - Complex sequencing and atomic operation
 - VAX interlocked instructions
 - Privileged instructions
- Translation buffer management
- Interrupt and exception setup and dispatching
- Synchronization primitives
- CALL_PAL examples:
 - CHMK *, REI *, SWPCTX, CFLUSH, MFPR/MTPR *, interlocked queue instructions *
(* = VAX instructions)

Remove from head of queue, interlocked

- **VAX:** microcoded instruction REMQHI
- **Alpha:** CALL_PAL REMQHIL
- **Itanium™ processor family:** *OpenVMS* system service SYS\$PAL_REMQHIL

IPL / ASTs / Software Interrupts

- 0 - 31 IPLs (but we only define 14 of them...
(2,3,4,5,6,7,8,9,10,11,15,21,22,31)
 - Map 16-31 directly onto a 16-bit interrupt register
 - Levels 0-15 are generated by software
- *OpenVMS* controls IPL and mode changes and delivers ASTs and software interrupts
- Alpha “registers” (e.g. ASTSR, IPL, ...) become
 - Itanium™ processor registers, or
 - CPU database cells or HWPCB cells

Page Size

- Page size will be 8KB, initially
- 3 levels of page tables
- Granularity hints (GH)
 - concept is the same as Alpha
 - size options are slightly different
 - GH regions (i.e. huge pages) will be handled by the *OpenVMS* TLB miss handler
- Address translation
 - Page tables
 - Virtual Hash Page Table

Page Protection

- Itanium™ architecture access rights and privilege levels can do all common VAX and Alpha page protections except
 - User Read Exec Write (UREW) - considering using User Read Super Write (URSW)
 - User Read Super Write (URSW) and Super Read Exec Write (SREW) won't allow execute access. (Privileged code must change: write code, then set page read only.)
- New possibilities
 - 'no execute' pages
 - protection key registers for additional protection

Virtual Address Space

- Address space will be 8TB in size initially
- 32-bit System Page Table (SPT) window will still be created in S1 space for 32-bit device driver code
- Each Itanium™ processor family region will have its own page table space
- P0, P1, S0, S1 will be 32-bit; P2 and S2 will be 64-bit

RID

7

S0, S1, S2

6

5

4

3

2

1

0

P0, P1, P2

PTE Format

- Since *OpenVMS* will implement the TLB miss handler it can retain its current PTE formats.
- The Address Space Match (ASM) bit will not be set for system space. ASM will be a function of the virtual address, not a PTE bit. All addresses in Itanium™ processor family region 7 will “have ASM set”.
- Other than protection bits and ASM, all PTE fields will be the same as on Alpha

Synchronization Techniques

- Requirement: to read/write a shared location in a single atomic operation
- Example *OpenVMS* Uses:
 - Spinlock
 - MUTEX
 - Semaphore
 - Queue instructions
- Alpha: CALL_PAL, LDxL / STxC and MB
- Itanium™ architecture: FETCHADDx, CMPXCHGx, XCHGx, MF, and acquire/release semantics on loads and stores

Process Context Switching

- More registers - 128 general, 128 floating... but
 - 2 FEN bits distinguish 32 registers vs. up to 128 registers in use
 - Considering “lazy restore” of FP registers
 - Register stack engine knows the general registers to save
- 2 Stacks
 - Memory stack - move a pointer
 - RSE backing store
 - Fill/spill happens in the background

Hello World

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("hello world\n");
```

```
    return 0;
```

```
}
```

Hello World (Alpha)

```
00A0          MAIN: ; 000874
23DEFFE0     00A0          LDA      SP, -32(SP)
221B0030     00A4          LDA      R16, 48(R27) ; 000876
B75E0008     00A8          STQ     R26, 8(SP) ; 000874
47E03419     00AC          MOV     1, R25 ; 000876
A75B0020     00B0          LDQ    R26, 32(R27)
B77E0000     00B4          STQ    R27, (SP) ; 000874
B7BE0010     00B8          STQ    FP, 16(SP)
47FE041D     00BC          MOV    SP, FP
A77B0028     00C0          LDQ    R27, 40(R27) ; 000876
6B5A4000     00C4          JSR    R26, DECC$GXPRINTF ; R26, R26
47FD041E     00C8          MOV    FP, SP ; 000878
A75D0008     00CC          LDQ    R26, 8(FP)
A7BD0010     00D0          LDQ    FP, 16(FP)
47E03400     00D4          MOV    1, R0 ; 000877
23DE0020     00D8          LDA    SP, 32(SP) ; 000878
6BFA8001     00DC          RET    R26
```

Routine Size: 64 bytes, Routine Base: \$CODE\$ + 00A0

Hello World (IA64)

```

                                .proc   MAIN
                                .align 32
                                .global  MAIN

                                MAIN:
                                { .mmi
012000100200    0000                add     r8 = $LITERAL$, r1 ;;           // r8 = 0, r1 // 000876
0080C0800200    0001                ld8     r8 = $LITERAL$                // r8 = [r8]
000008000000    0002                nop.i   0 ;;

                                }
                                { .mii
002C00308840    0010                alloc   r33 = rspfs, 0, 3, 1, 0      // 000874
010800100880    0011                mov     r34 = r1
000188000800    0012                mov     r32 = br0 ;;

                                }
                                { .mib
000008000000    0020                nop.m   0
0000B08008C0    0021                sxt4    r35 = r8                    // 000876
00A000001000    0022                br.call.sptk.many br0 = DECC$TXPRINTF ;; // br0 = DECC$TXPRINTF

                                }
                                { .mii
012000002200    0030                mov     r8 = 1                      // 000877
010802200040    0031                mov     r1 = r34                    // 000876
000154042000    0032                mov.i   rspfs = r33 ;;              // 000877

                                }
                                { .mib
000008000000    0040                nop.m   0
000E00140000    0041                mov     br0 = r32
000108001100    0042                br.ret.sptk.many br0 ;;

                                }

                                .endp   MAIN

```

Routine Size: 80 bytes,

Routine Base: \$CODE\$ + 00E0

System Performance

- Evolution of system architecture
- What performance?
 - CPU, Memory and IO subsystem
- Why isn't more always better?
- Can I use money to buy better performance?
- What is NUMA?
- SMP scaling?
- Does OpenVMS care?
- Is engineering doing anything to improve system performance?

Evolution of System Architecture



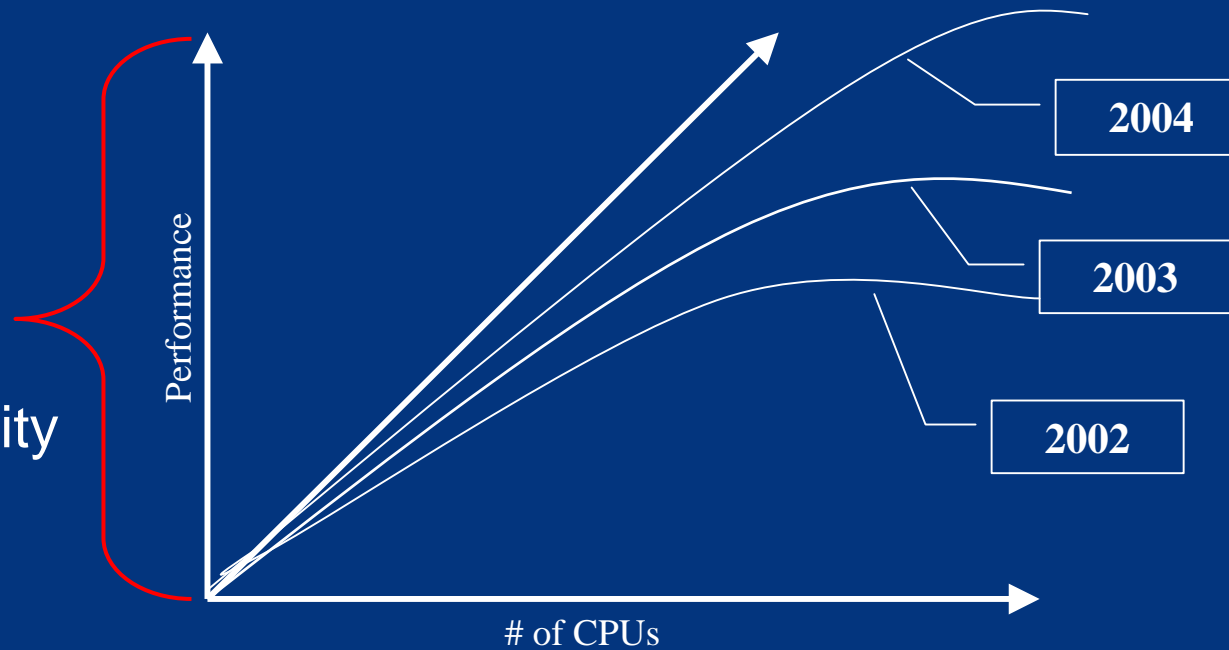
- Past
 - Single shared system bus
- Current
 - EV6, Crossbar technology
 - NUMA, QBB, partition
- Future
 - EV7, Rambus technology
 - CPU interconnect on-chip, Torus, switch fabric to connect CPUs to IO bus, PCI-X, partition, scalable

OpenVMS Performance Futures

Continued Year over Year improvement

Improved:

- SMP Scaling
- System Scalability



Projects to improve performance & scaling:

2002:

Finer Grain Locking
Distributed Interrupts
Mailbox Scaling

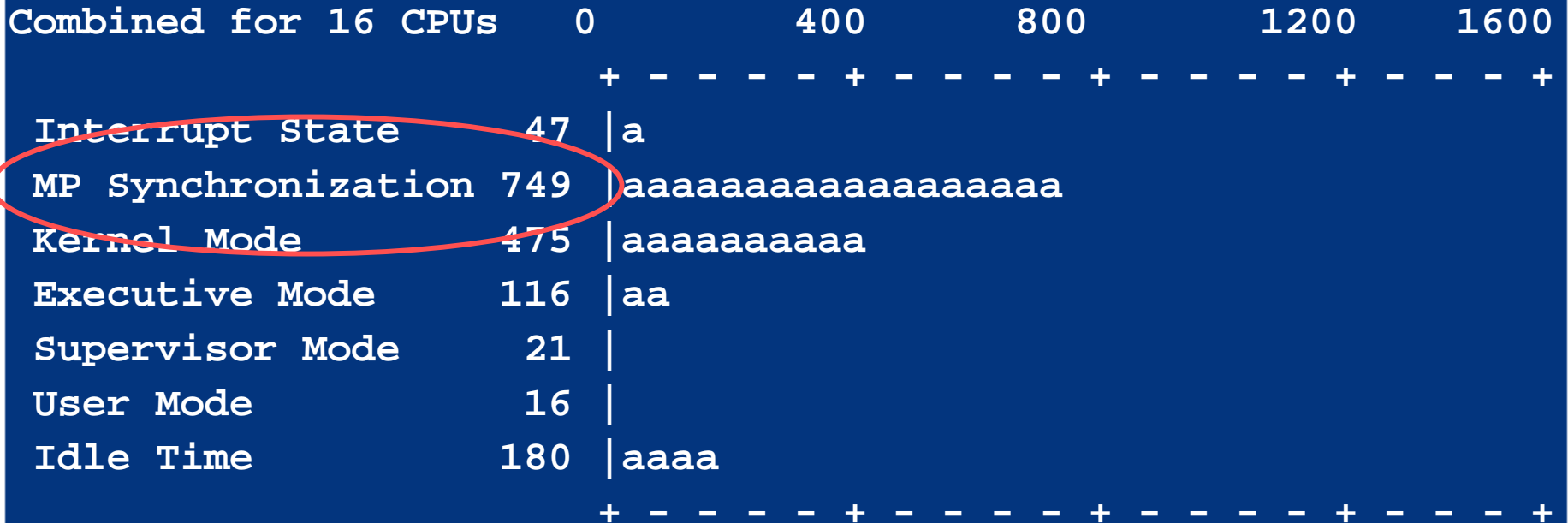
2003:

LAN Fastpath
Large working sets
Improve locking for TCP/IP

MONITOR MODE

```
+-----+  
| CUR |  
+-----+
```

```
TIME IN PROCESSOR MODES  
on node DECRDB  
12-JAN-2001 09:00:06.64
```

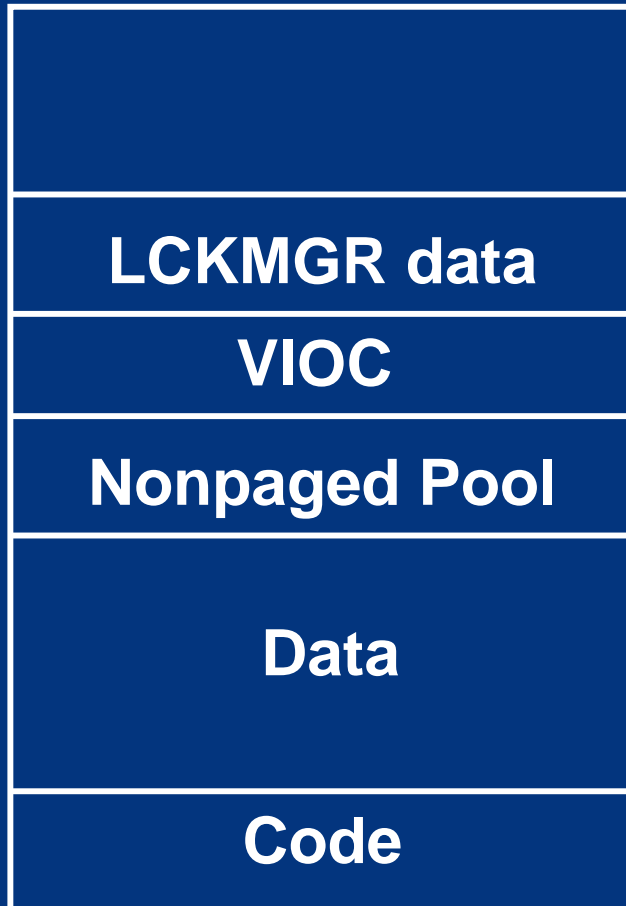


Where are the problem areas?

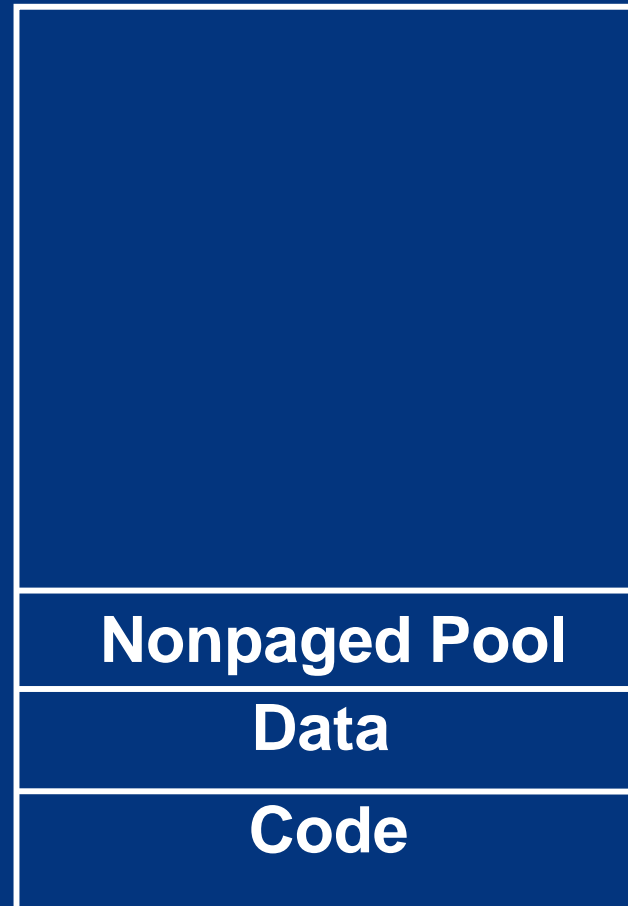
- SMP contention - High MPsynch
 - A new spinlock trace tool can provide very detailed data about MP synch time
 - \$ @sys\$examples:spl.com
 - trace hooks in V7.2-1H1, but procedure only shipping with V7.2-2, V7.3 and newer
- Primary CPU contention - High I-Stack Time
 - I/O interrupts?
 - Don't schedule any processes on CPU0?
 - `SYSGEN> SET SCH_CTLFLAGS 3`

Exec Layout on NUMA Systems

RAD0



RAD1



Performance Improvements in V7.2-2 and V7.3

■ General

- Dedicated-CPU lock manager
- Process scheduling, idle loop
- MUTEX without SCHED spinlock
- SYS\$RESCHED (used by DECthreads and Oracle)
- SYS\$GETJPI
- MailBox driver

■ I/O

- Fibre fastpath (V7.3)
- SCSI fastpath (V7.3)
- XFC (V7.3)

Dedicated CPU Lock Manager

- Greatly reduced lock manager spinlock contention
- \$ENQ/ \$DEQ
 - creates request packet in a queue
 - process spins at IPL2 for completion status
- LCKMGR_SERVER process
 - hard CPU affinity to a single CPU
 - excellent use of CPU cache
 - services requests and returns status
- LCKMGR_MODE dynamic system parameter

Performance Improvements in V7.3-1

- NUMA Changes
 - distributed interrupts for fastpath drivers
 - replication of read-only data
 - layout of exec global data
- Scaling Changes
 - balance slot size
 - timer queue processing
- SMP changes
 - AST delivery
 - event flag posting
 - I/O

NUMA Changes for V7.3-1

- Replication of some Read Only OS Data
 - All static SYSGEN Parameters
 - Some Static OS Data, examples:
 - MMG\$GL_PAGE_SIZE, etc...
 - CPU\$GB_ARCH_TYPE
 - EXE\$GPQ_HWRPB
- KPBs are RAD specific
 - Kernel process for fastpath IOs
- PCBs are RAD specific
 - During process creation, the PCB is allocated from memory from the home RAD

Scaling Changes - More S0S1 space

- FREDs hold context for Kernel threads
 - Prior to V7.2, each process had 16 FREDs pre allocated - this increased to 256 in V7.2
 - FREDs lived in S0S1 space, and were allocated as part of the Balance Set slots
 - At 512 bytes each, a SYSTEM with BALSETCNT set to 2000 had 256MB of S0S1 virtual address space allocated to 512,000 FREDs which in most cases, few would ever be used
- FREDs are no longer pre-allocated, they have also moved to S2 Space

SMP Changes - Mailbox Spinlocks

- We are now allocating a dynamic spinlock per mailbox
 - Mailbox operations were previously synchronized by the static MAILBOX spinlock
 - Allows parallel operations to occur on different mailboxes
 - A few permanently allocated mailboxes still use the MAILBOX spinlock

SMP Changes - Fork Lock Interface

- The Lock Manager has a Fork Lock Interface for use by high IPL threads
 - This interface has required callers to hold IOLOCK8 when calling and all callbacks were also made with IOLOCK8 held
- The Fork Lock Interface now supports a flag indicating that IOLOCK8 is not required
- RMS uses this flag for the locking operations on Global Buffers

SMP Changes - Timer Queue

- The Timer Queue on VMS has been a linked list of TQEs in order of due time
- Synchronization is with the TIMER spinlock
- Systems with large numbers of entries and heavy additions and removals would experience high MP_Synch
- V7.3-1 changes the structure of the Timer Queue to a tree structure

SMP Changes - PCB Specific Spinlock

- A Dynamic Spinlock is now associated with each PCB
- The spinlock synchronizes the AST Queues in the PCB and KTBs of the process
- AST Delivery no longer requires the SCHED spinlock
- Queuing an AST still utilizes the SCHED spinlock to change the process state (ex. LEF to COM)

SMP Changes – I/O Post-Processing

- Finish I/O on current CPU as opposed to let primary CPU do the post-processing
 - global IOC\$GQ_POSTIQ queue
 - per-CPU CPU\$L_PSFL/BL post queue
- Driver must set UCB\$V_IOPOST_LOCAL bit in UCB\$L_STS
- I/O completion routines know about this
 - IOC\$REQCOM, IOC\$ALTREQCOM, IOC\$POST_IRP, COM\$POST and COM\$POST_NOCNT
- DECram, Mailbox and Shadow driver initially

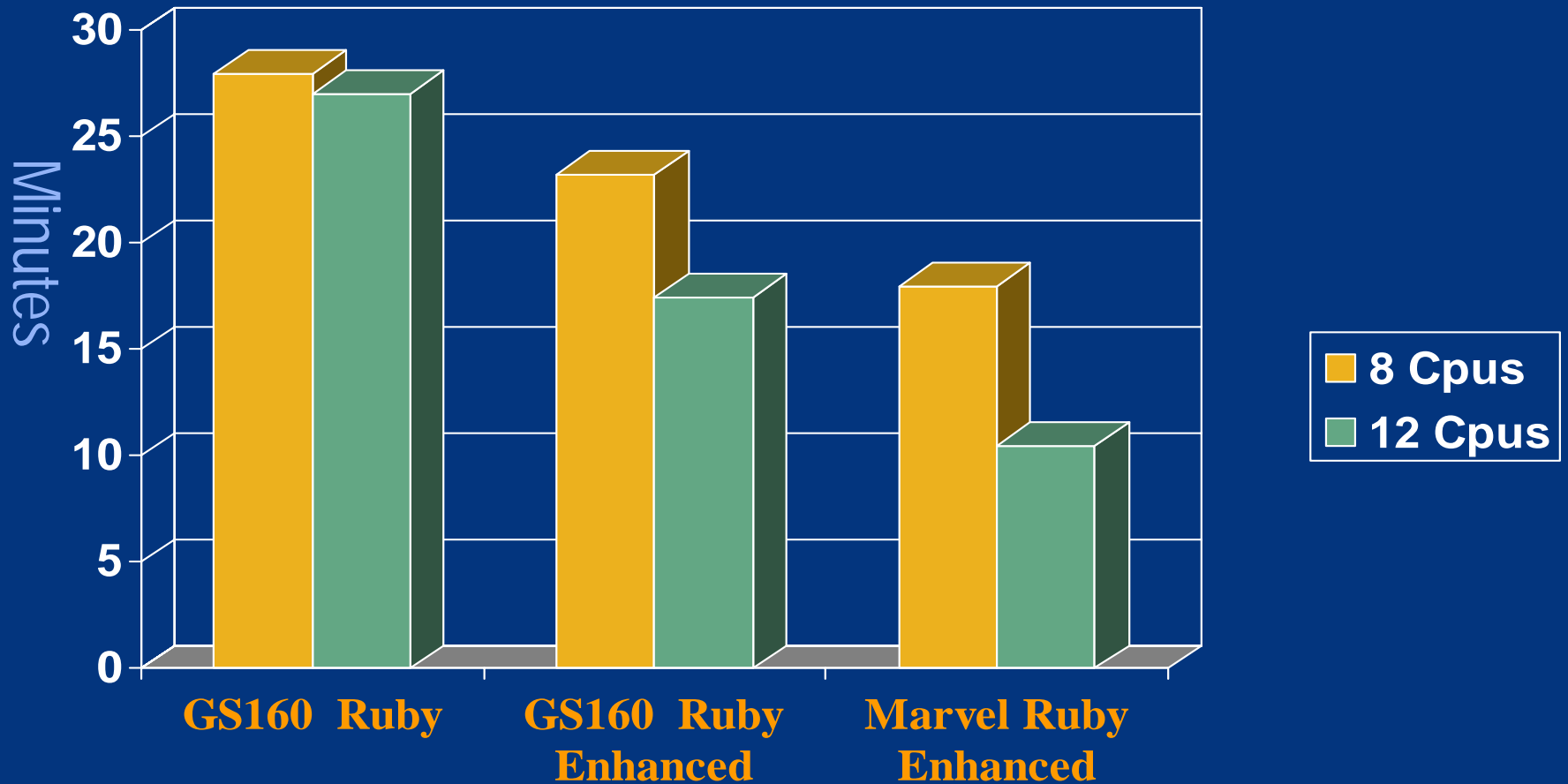
Real Life Customer Example

- Financial customer has a nightly 1-2 hour window to produce analysis reports
- The run is made up of a variable number of analysis processes and a reporter process
- Data is numerous RMS indexed files, some read/write, some read-only, some created read/write
- Currently GS140, 12 CPUs, 8 GB memory
- All data is on a RAMdisk (about 3 GB)
- Current run takes about 45..50 minutes

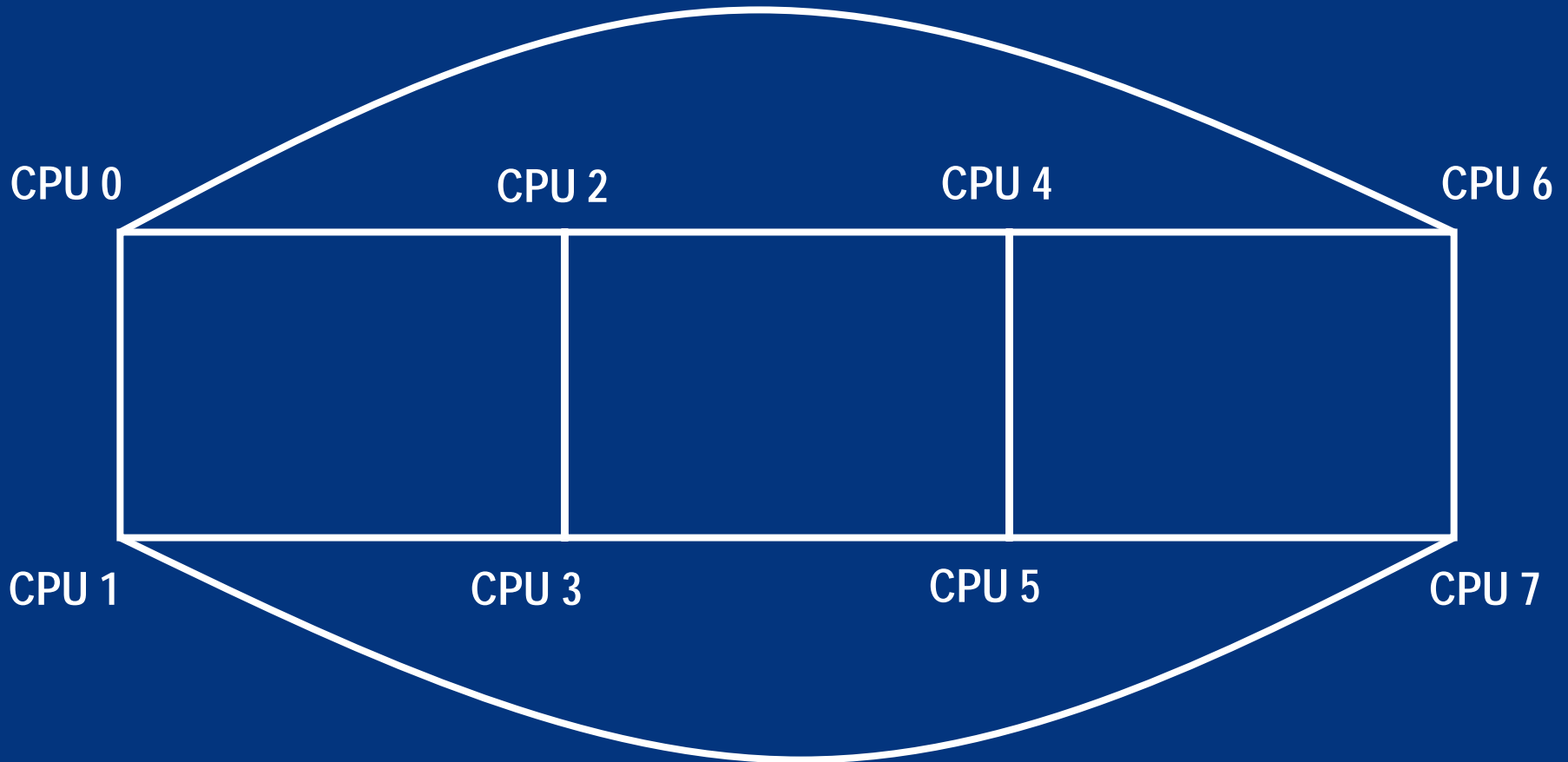
Financial Test Runs

GS160 - 931mhz EV68

Marvel - 800mhz EV7



Future AlphaServer - Marvel 8p



Memory Latency

| System: | Chip/Speed: | Rad/Hops: | Latency: |
|----------------|-------------|-----------|-----------|
| Marvel nsec | EV7/800 | 0 hops | 150 |
| Marvel nsec | EV7/800 | 1 hops | 250 |
| Marvel nsec | EV7/800 | 2 hops | 300 |
| Marvel nsec | EV7/800 | 3 hops | 350 |
| WildFire | EV68/1000 | on-Rad | 310 nsec |
| WildFire | EV68/1000 | off-Rad | 1110 nsec |
| WildFire | EV67/730 | on-Rad | 390 nsec |
| WildFire | EV67/730 | off-Rad | 1020 nsec |

Future SMP Performance Work

- Performance work can be difficult to plan
 - Many areas we know need improvement
 - Many areas just appear before us at any time
- Breakup of IOLOCK8
 - remove usage of IOLOCK8 spinlock in driver paths
- Additional SCHED reduction
 - synchronize more process specific data with the PCB specific spinlock
- XFC
 - improve spinlock usage and concurrency to insure scaling

Performance Improvements beyond V7.3-1

- SMP changes
 - NAM_TO_PCB without SCHED spinlock
 - more to come...
- Scaling changes
 - Move working sets out of S0S1 and into S2 space
- I/O
 - LAN fastpath
 - Breakup of IOLOCK8

SMP Changes – \$HIBER / \$WAKE

- No need for SCHED spinlock to wake current process
- PCB\$V_WAKEPEN bit in PCB\$L_STS synchronized under SCHED spinlock
- new PCB\$V_WAKEPEN bit moved to PCB\$L_STS3
- waking up current process from within AST saves 2 SCHED acquire/release
- waking up target process not current on any CPU saves 1 SCHED acquire/release
- Helps multi-threaded processes (i.e. Java apps and friends)

Scaling Changes – Move WSL into S2

- Currently balance slot live in S0S1 space
 - consist of PHD, BAK array, working set list (WSL) and process section table (PST)
 - Double-mapped into P1 space
 - Trade-off large number of resident processes (BALSETCNT) vs large working set (WSMAX)
- New working set slots in S2 space
 - WSL
 - BALSETCNT obsolete (forced to be MAXPROCESSCNT-2)
 - Bslots still in S0S1 (PHD and PST) always allocated
- Example:
 - booted GS160 (19GB phys mem) with 10K processes and 18GB WSMAX !!!
 - 64K process quota limit removed, run with 8GB working set

TCPIP Performance

Current Synchronization Mechanisms

- Single Threaded
 - One user/operation in execution at any instance
 - Needed to guarantee synchronization of internal kernel data structures
 - This is true regardless of the number of CPUs or users
- Synchronization achieved using global single Spinlock: IOLOCK8
 - Contention with other IOLOCK8 users
 - DECnet, LAN drivers, SCS, etc..... Everybody!

TCPIP Performance

Future Synchronization Mechanisms

- Multiple dynamic spinlocks
 - No more IOLOCK8
- Queue KRP (kernel request packet)
 - Handled by fork thread on non-primary CPU
 - Similar to dedicated lock manager
- Improve concurrency
 - Multiple concurrent network I/O
 - Multiple processes can allocate mbufs, fill in data and queue requests