OpenVMS Process Internals

Wayne Sauer

President, PARSEC Group

sauer@parsec.com

www.parsec.com

888-4-PARSEC

get connected PEOPLE. TECHNOLOGY. SOLUTIONS.

HP Technology Forum & Expo 2008

Topics

- OpenVMS internal symbol layout
- SDA commands
- Linked lists and hashing tables
- Virtual address space layout
- Process data structures
- Kernel threads
- SDA Lab

OpenVMS Symbol Type and Layout

- There are basically three types of symbols that you will encounter in OpenVMS
  - Symbolic data structure offset – which is used by adding the symbols value to the base of the data structure to get to a field in the data structure. For example
    - PCB$L_PID
  - Symbolic address of an OpenVMS system routine – which is an address of a routine within OpenVMS. For example:
    - EXE$TIMEOUT
  - Symbolic address of an OpenVMS executive location. For example:
    - SCH$GQ_COMQS

Introduction to SDA

- There are two ways of getting into SDA. One is to examine the live system, and the other is to analyzed a crash dump file.
- To analyze a live system, issue the following:
  - $ analyze/system
    - Need the CMKRNL privilege
    - Since it is a live system, things change – REMEMBER that
- To analyze a crash dump, issue the following:
  - $ analyze/crash dump-filespec
    - Need read access to the dump file
    - Everything is static

Introduction to SDA (continued)

- Remember most ALL references are in hexadecimal
- To specify decimal or octal, use the following:
  - ^d or ^o
- Other operators are
  - @ before a reference is a level of indirection
  - +, -, *, / are arithmetic operators
- Order of precedence is the same as in basic mathematics, including changing precedence by enclosing the expression in parenthesis ()
- The period (.) is the current location pointer

SDA Commands - processes

- SDA> **show summary**
- SDA> **show summary/image**
- SDA> **set process/index=1e**
- SDA> **set process parsec**
- SDA> **show process/index=1e**

SDA Commands - examine

- SDA> **examine 20000**
- SDA> **examine 20000;20**
- SDA> **examine exe$timeout**
- SDA> **examine @sch$gl_pcbvec;(^d32*4)**
- SDA> **show stack/long @sch$gl_pcbvec;(^d32*4)**

SDA Commands - evaluate

- SDA> **evaluate sch$gl_pcbvec**
- SDA> **evaluate 64*2-44**
- SDA> **evaluate ^d72**

- SDA> **evaluate/time @exe$timeout**
- SDA> **examine/time exe$timeout**

SDA Commands - symbols

- SDA> **show sym pcb$l_pid**
- SDA> **show sym *pcbvec***
- SDA> **show sym/all pcb$l_**
- SDA> **define mypcb 80EE0300**
- SDA> **undefine mypcb**

SDA Commands – Automatically created symbols

- There are a number of symbols automatically created when you are looking at a process or device
- For processes some of the symbols are:
  - PCB, JIB, PHD
- For devices, some of the symbols are:
  - UCB, DDT

SDA Commands - misc

- SDA> **read sys$loadable_images:sysdef**
- SDA> **format pcb**

- SDA> **read/executive**
- SDA> **show executive**
- SDA> **map 810B8050**

- SDA> **show device**
- SDA> **show cluster**
- SDA> **show lan**

SDA Extensions

- SDA has a number of extensions that can be used
- To find out what SDA extensions exist, issue the following command:

```
CLASS3> dir sys$library:*sda*

Directory SYS$COMMON:[SYSLIB]

CLUE$SDA.EXE;1      CNX$SDA.EXE;1       DECDTM$SDA.EXE;1    DKLOG$SDA.EXE;1
FC$SDA.EXE;1        IO$SDA.EXE;1        IPC$SDA.EXE;1       LAN$SDA.EXE;1
LCK$SDA.EXE;1       LNM$SDA.EXE;1       MTX$SDA.EXE;1       OCLA$SDA.EXE;1
PCS$SDA.EXE;1       PE$SDA.EXE;1        PTHREAD$SDA.EXE;1   PWIP$SDA.EXE;1
SDA$SHARE.EXE;1     SDA$SHARE.EXE_OLD;1 SDARMS$SHARE.EXE;1  SPL$SDA.EXE;1
TCPIP$SDA.EXE;1     TQE$SDA.EXE;1       TR$SDA.EXE;1        USB$SDA.EXE;1
XFC$SDA.EXE;
```

SDA Extensions (continued)

- To find out how to use them, issue the first part of the SDA extension name at the SDA prompt, for example to learn what commands are available for the TQE$SDA.EXE SDA extension, issue the TQE command at the SDA prompt:

```
SDA> tqe
Timer Tracing Utility TQE commands:

        TQE LOAD
        TQE UNLOAD

        TQE START TRACE [/BUFFER=pages]
        TQE STOP  TRACE

        TQE SHOW  TRACE [/SUMMARY]
                        [/IDENTIFICATION=pid]
                        [/ADDRESS=address]
```

SDA Extensions (continued)

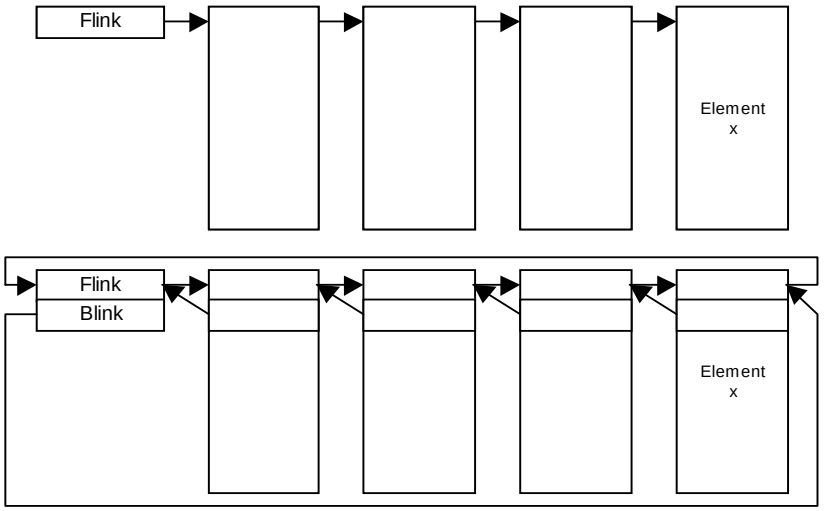
- Probably the most used (and oldest) SDA extension is CLUE. It has a separate help library as follows:

```
SDA> clue
CLUE Alpha   -   Type CLUE HELP for further Information
CLUE commands: CALL_FRAME, CANASTA, CLEANUP, CONFIG, CRASH, DEBUG, ERRLOG, FRU,
               HELP, HISTORY, KPB, MCHK, MEMORY, PROCESS, REGISTER, SCSI, SG,
               STACK, SYSTEM, VCC, XQP
SDA> help clue

CLUE


     Invokes the Crash Log Utility Extractor


  Additional information available:

  CALL_FRAME CLEANUP    CONFIG     CRASH      ERRLOG     FRU        HISTORY
  MCHK       MEMORY     PROCESS    REGISTER   SG         STACK      SYSTEM
  VCC        XQP

CLUE Subtopic?
```

Linked Lists

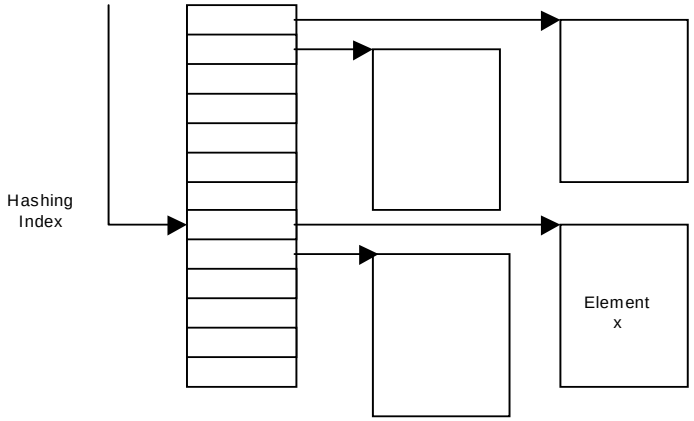
- OpenVMS stores data structures using one of two ways: linked lists and hashing tables.
- Linked lists consist of either a forward pointer, or a combination of forward and backward pointers.
- They are easy to implement since finding an element in the list is as simple as following the pointers until you find the element that you are searching for.
- For example:

Single and Double Link List Layout

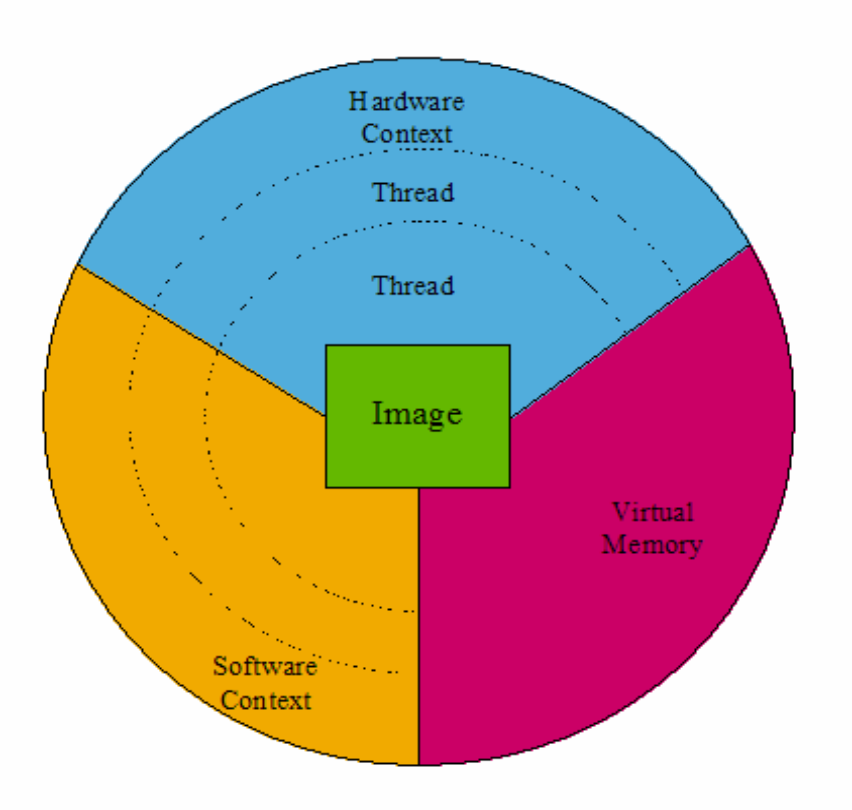


SDA> validate queue sch$gq_hibwq
SDA> format @sch$gq_hibwq
SDA> format @.

Hashing Tables and Hashing Algorithm



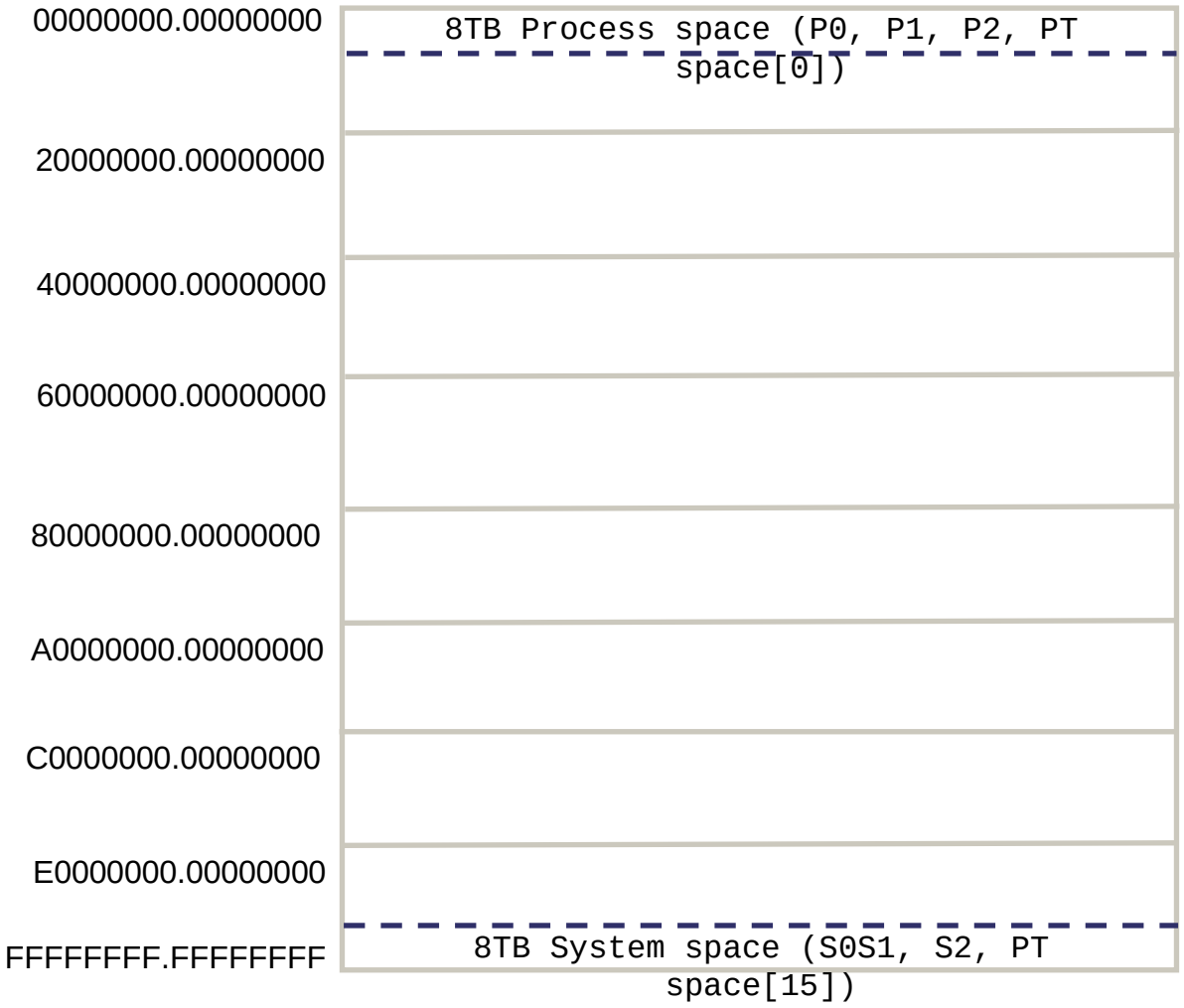
Pictorial Representation of a Process



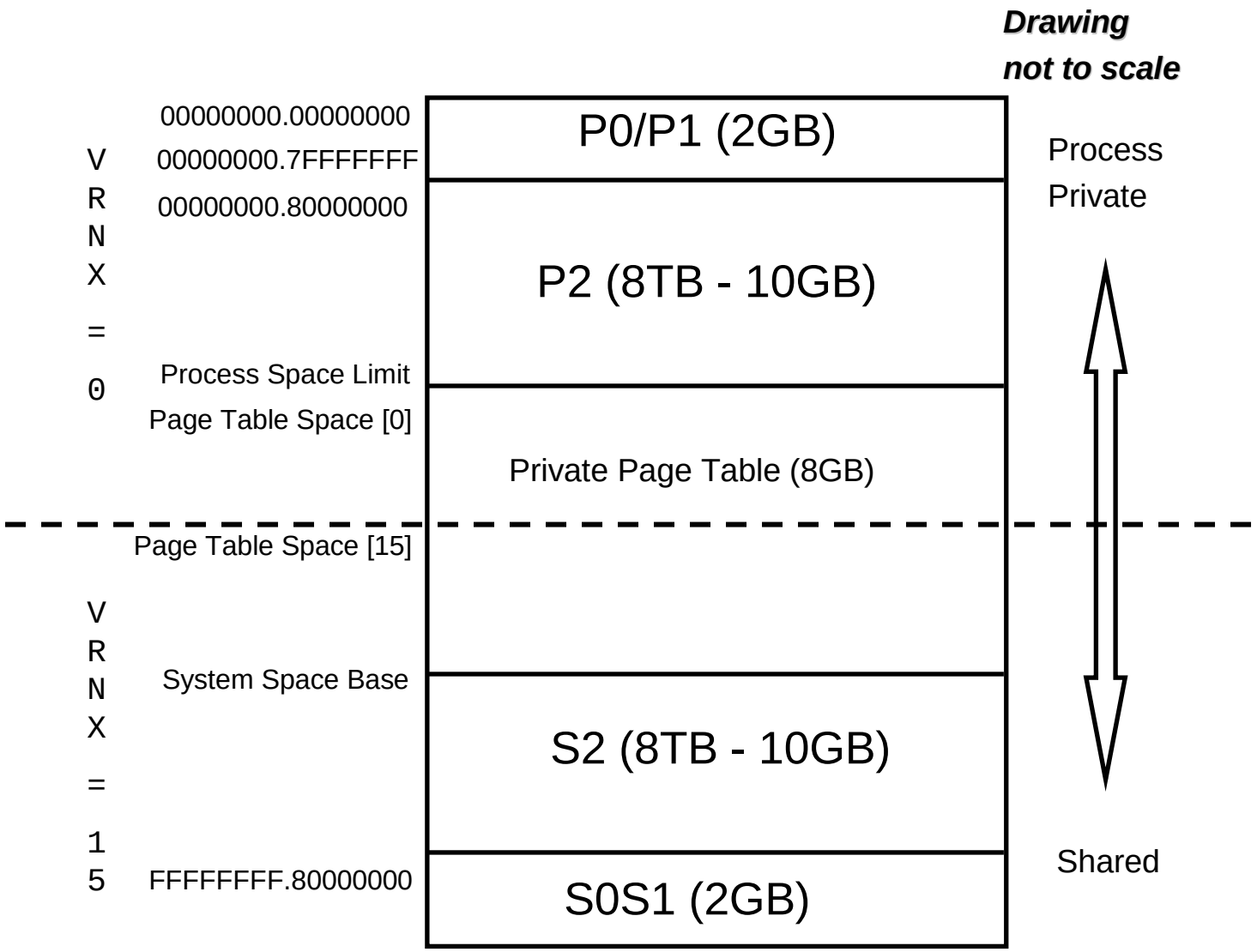
Alpha Virtual Address Space

00000000.00000000 to 00000000.3FFFFFFF	P0 Space
00000000.40000000 to 00000000.7FFFFFFF	P1 Space
00000000.80000000 to 000003FF.FFFFFFFF	P2 Space
FFFFFC00.00000000 to FFFFFFFB.FFFFFFFF	P2 Space
FFFFFFFC.00000000 to FFFFFFFD.7FFFFFFF	Page Table Space
FFFFFFFE.00000000 to FFFFFFFF.7FFFFFFF	S2 Space
FFFFFFFF.80000000 to FFFFFFFF.FFFFFFFF	S0/S1 Space

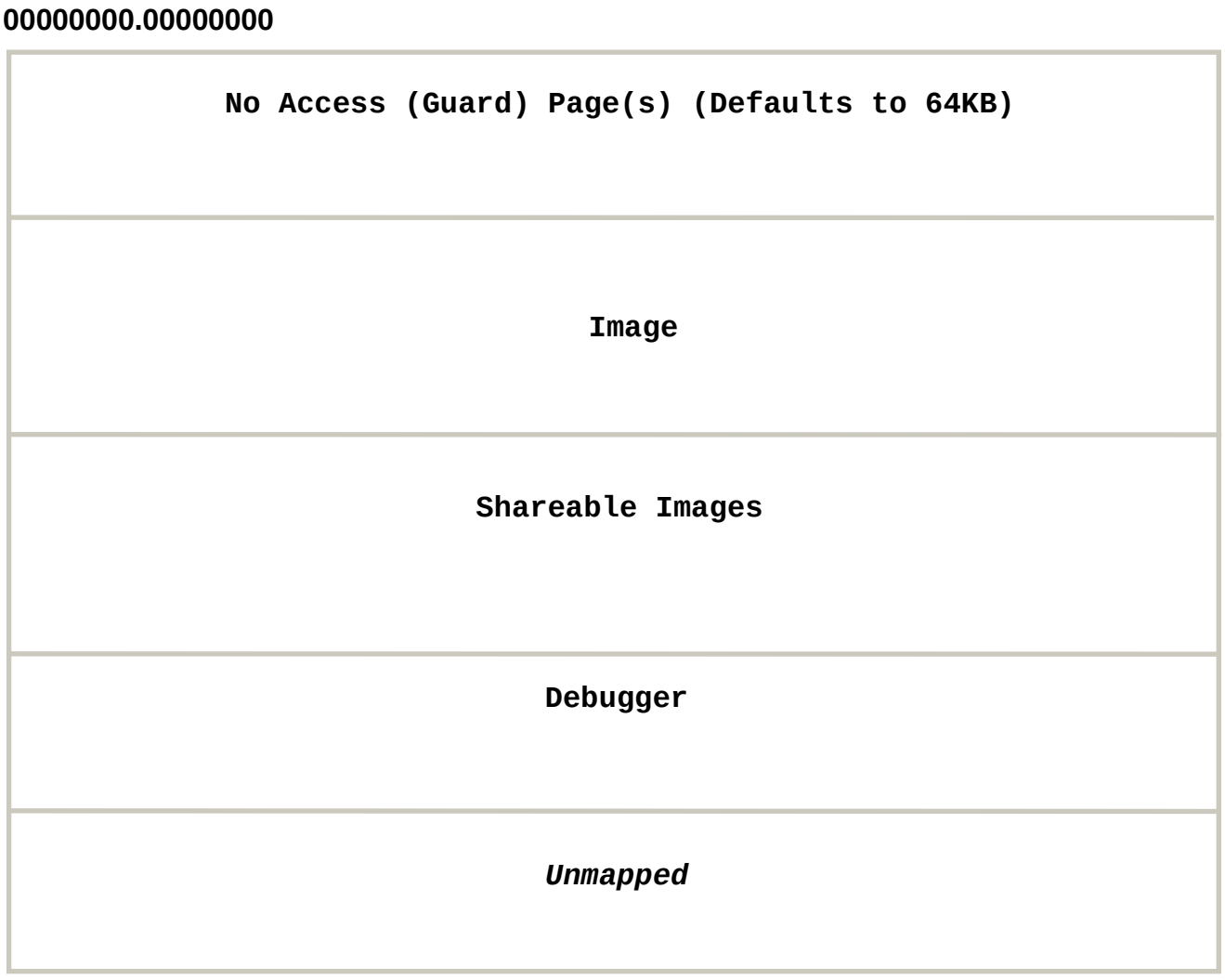
64-bit IVMS Address Space



64-bit IVMS Address Space



P0 Space Layout



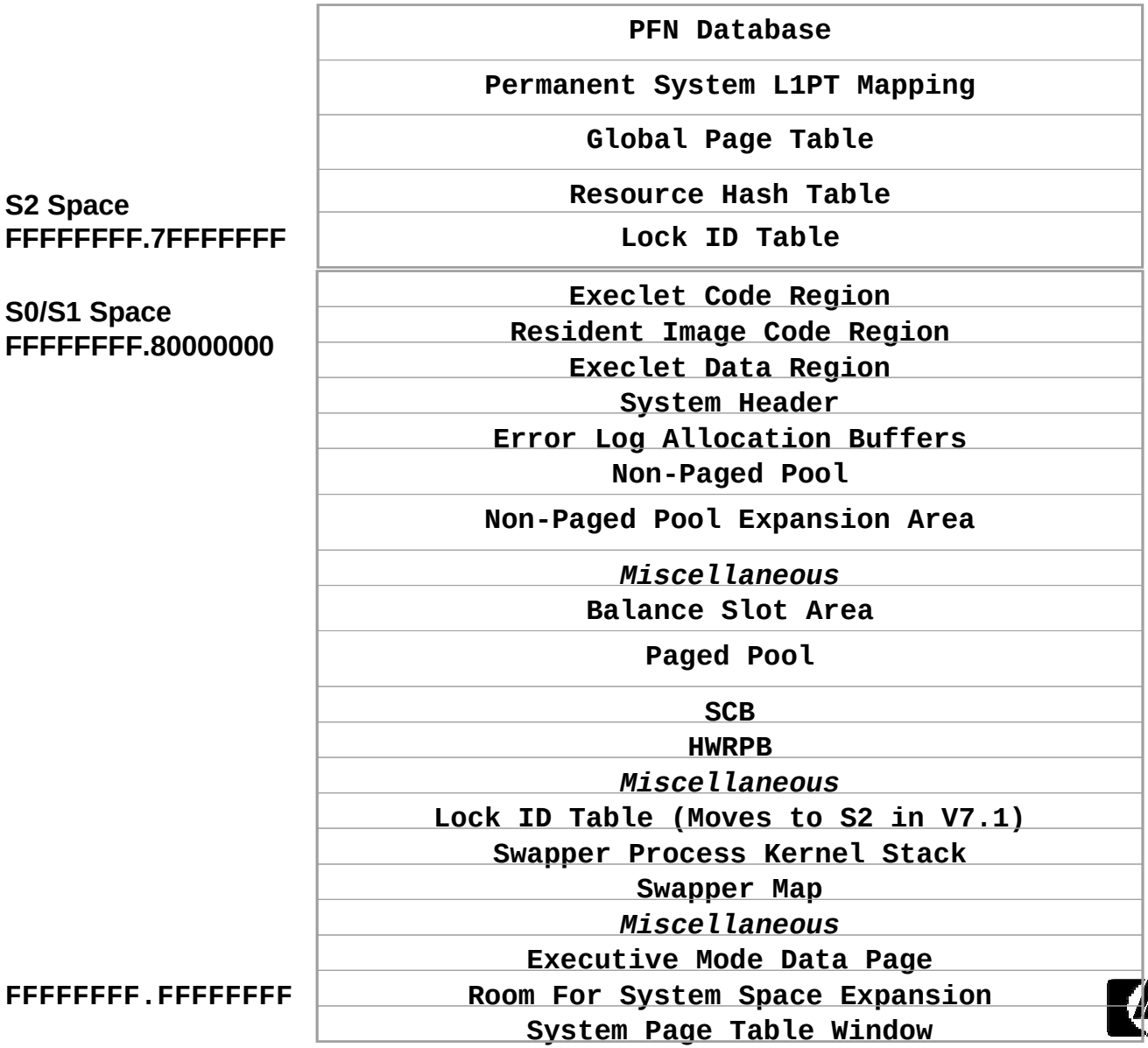
P1 Space

00000000.400000000

Unmapped
User Stack
CLI Data (symbol tables)
CLI Command Tables
CLI Image
File System Impure Area
Image I/O Segment (IMGIOCNT) (UREW)
Image I/O Segment (Exec)
Process I/O Segment (PIOPAGES)
Shareable Image Linkage Area (IMGREG_PAGES)
Channel Control Block Table (CHANNELCNT)
Window to PHD
Kernel Stack (initial thread) (KSTACKPAGES)
Executive Stack (initial thread/2 pages)
Supervisor Stack (initial thread/4 pages)
Exec Mode Data Area
NSA Audit Table
Privileged Library Dispatch Table
User Mode Event Data Area
KRP Lookaside List
Debug Context
Debug Data Area
Generic CLI Data Pages
Image Header Buffer
RMS Process Context Area
RMS Directory Cache
RMS IFAB/IRAB Tables
Image Activator Context
Image Activator Scratch
Per-Process Common Area
OpenVMS User Mode Data Page
Process Initial Thread Area
PKTA Vector
P1 Pointer Area
Kernel Mode Data Area

00000000.7FFFFFFF

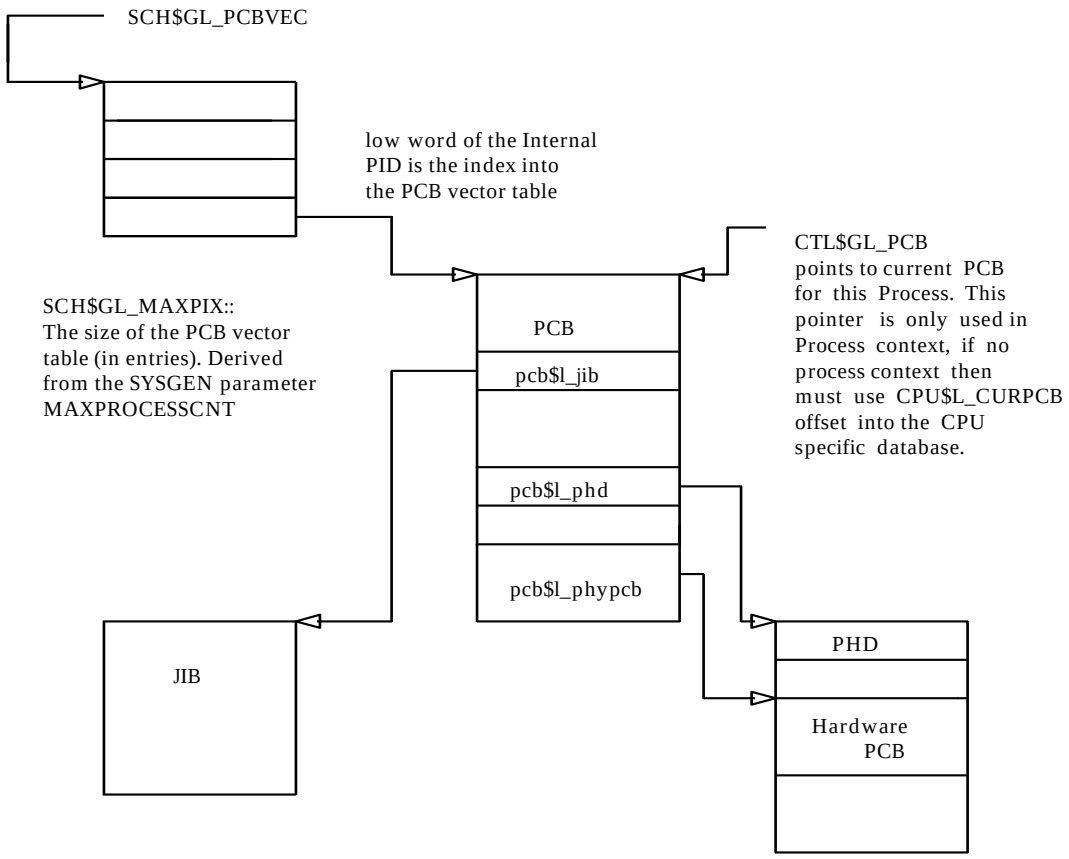
S0/S1 and S2 Space



SDA Commands

- SDA> **show proc/proc/page**
- SDA> **show proc/page/p0**
- SDA> **show proc/page/p1**
- SDA> **show proc/page/p2**
- SDA> **show page**
- SDA> **show page/gpt**
- SDA> **show page/free**
- SDA> **show page/global**
- SDA> **clue memory/layout**
- SDA> **clue process/layout**
- SDA> **clue memory/lookaside**

Process Data Structure Layout

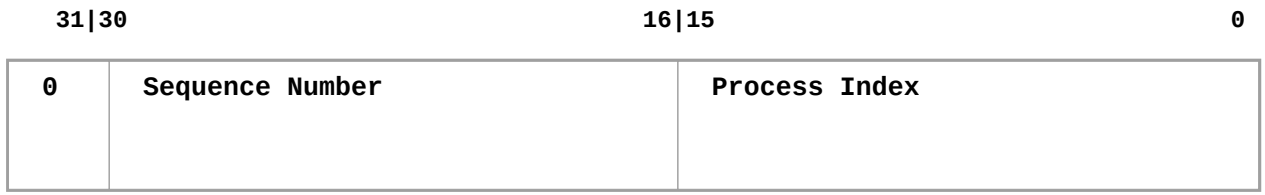


Locating the PCB

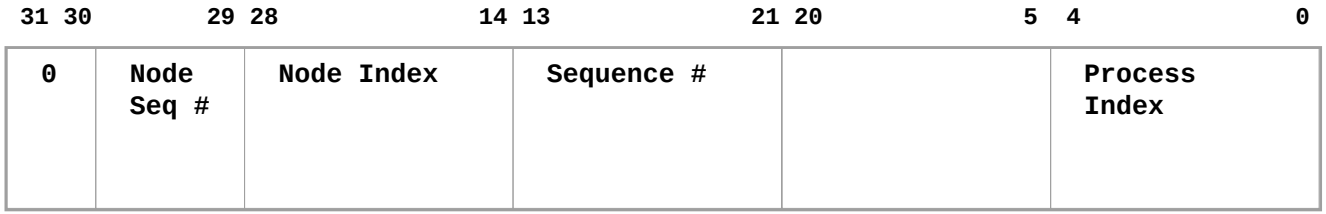
- There are three ways of locating a PCB
  - You can locate your PCB via the symbol CTL$GL_PCB, which is a symbol in your P1 space
  - You can locate any processes PCB via the PID by using the index portion of the internal PID to index into the PCB vector table
  - You can locate the current PCB on any CPU via the CPU database
- Finding your PCB is easy. We will look at how to find the other PCBs

PID/EPID Layouts

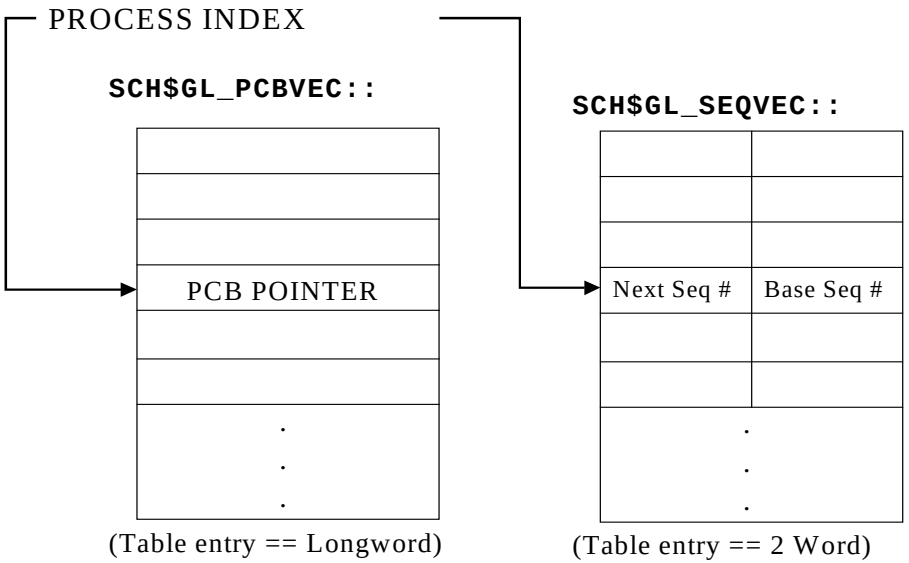
PID LAYOUT



EPID LAYOUT

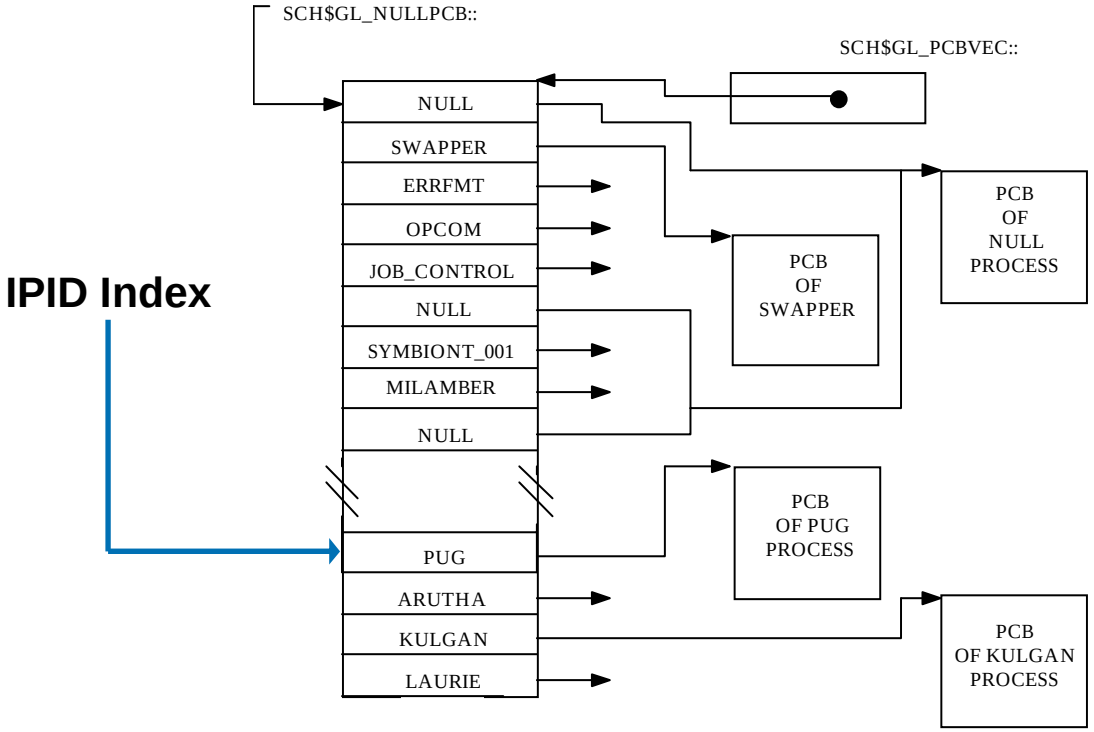


Sequence Vector Table



- The internal PID is created by putting the index of the PCB vector and the sequence number together.
- The sequence number is incremented each time the vector slot is reused
- **SCH$GL_SEQVEC** contains the address of the PCB vector table
- The Extended PID is created by adding the node's cluster system ID to the PID.

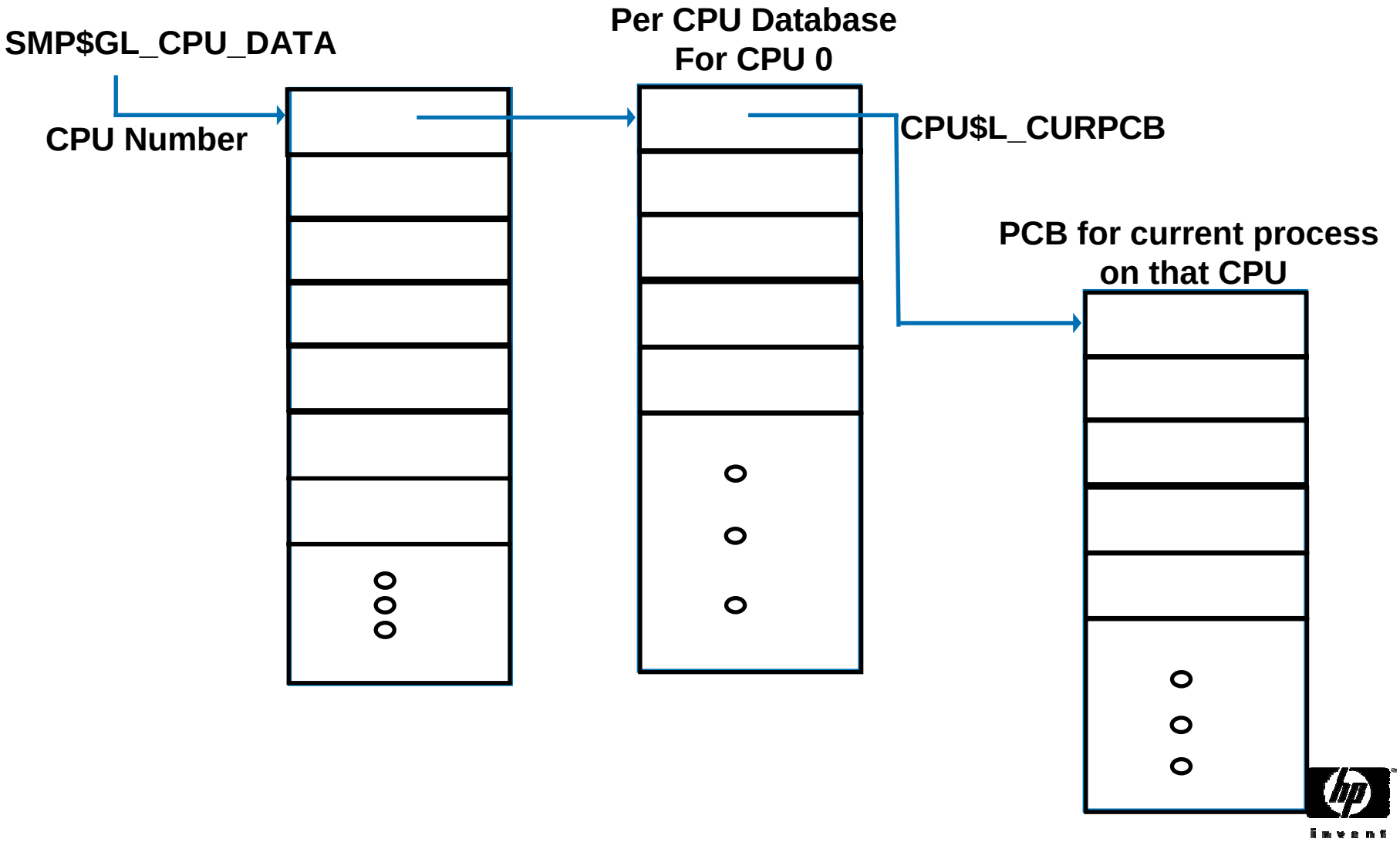
Locating a PCB Via the PCB Vector Table



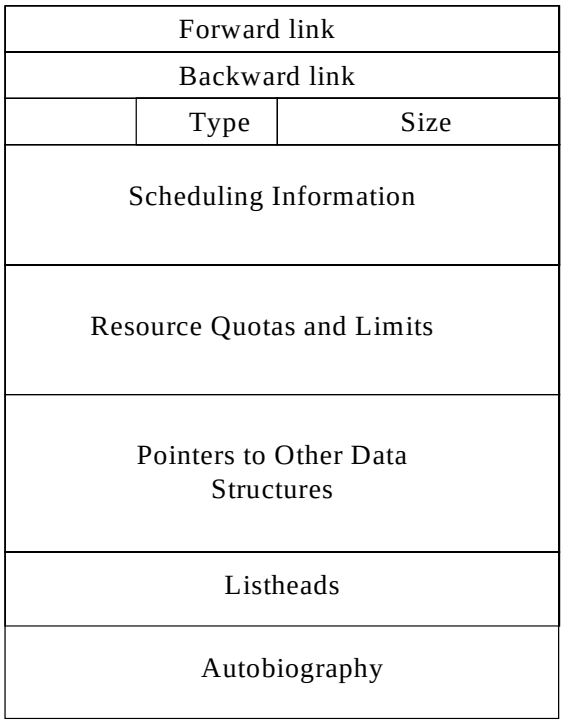
SDA Commands:

SDA> **ex sch$gl_maxpix**
SDA> **evaluate sch$gl_pcbvec**
SDA> **examine sch$gl_pcbvec**
SDA> **examine @sch$gl_pcbvec;200**
SDA> **show stack/long @sch$gl_pcbvec;7f*4**
SDA> **examine @sch$gl_pcbvec+(ea*4)**

Locating a PCB Via the CPU Database

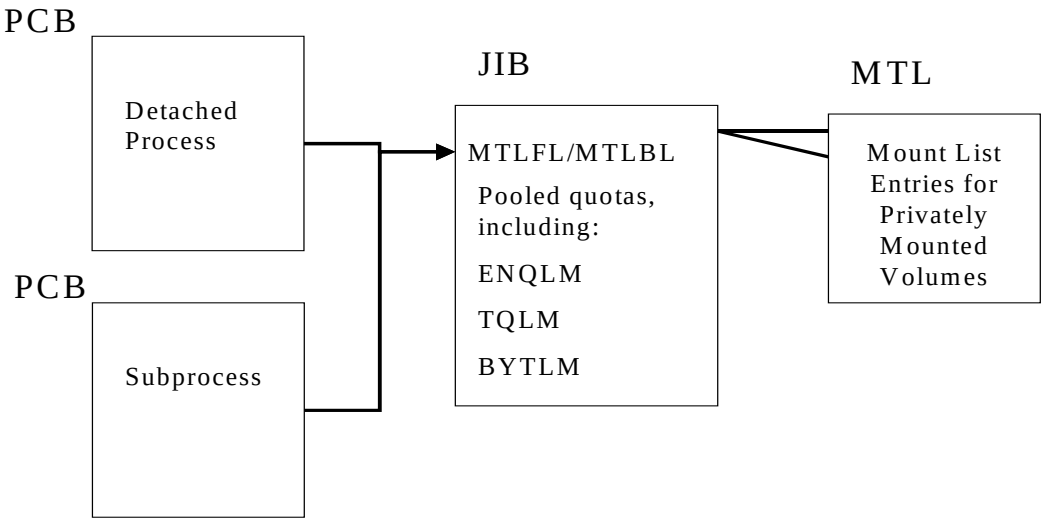


The Process Control Block



- There is one PCB per Process and it is allocated out of Non-Paged pool
- The current PCB is pointed to by CTL$GL_PCB if in process context. If you are in system context, you must locate the current process by using the CPU$L_CURPCB offset into the CPU specific database.
- All other PCB's can be located by using the low word of the internal PID as an index into the PCB vector table.
- The number of PCB's you can have on the system is determined by the SYSGEN parameter MAXPROCESSCNT

Job Information Block (JIB)



- A Job consists of a detached process and all of its subprocesses
- The JIB tracks all shared resources allotted to a job
- There is one JIB per job, ie. multiple subprocesses will share one JIB
- The JIB is allocated from Non-paged pool
- The JIB is located via the PCB
- The JIB is the job specific data structure

Process Header (PHD)

Offsets to Working Set List and Process Section Table
Accounting Statistics
HWPCB
More Accounting Statistics
P0/P1/P2 Page Table Descriptions
Working Set List (Located in S2 space in Alpha and I64 OpenVMS V8.2)
FREDs
Process Section Table

- The PHD is located in the balance slot area and there is one PHD per process
- The number of balance slots now is 2 minus MAXPROCESSCNT
- The PHD is the image specific data structure and may be outswapped
- Although the PHD has a TYPE field, it is not used. In order to format this data structure in SDA you must include the /type=PHD

Hardware Privileged Context Block

Stack Pointers
L1 Page Table PFN
Miscellaneous Registers
Floatpoint Registers

Canonical Kernel Stack

SP->

R8-R15
R29
(fill)
R0
R1
R16
R17-R28
(fill)
R2-R7
PC
PS

CALL_PAL SWPCTX
CALL_PAL REI

SDA Commands

- SDA> **show proc/channel**
- SDA> **show proc/work**
- SDA> **show proc/proc**
- SDA> **show proc/lock**
- SDA> **show resource/lock=0D00033A**
- SDA> **show lock 24000140**

Kernel Threads

- OpenVMS supplies the DECthreads run-time library, to support the multithreading of an application. The DECthreads library is implemented as user mode services.

  $ define pthread_config “vp-count=4”

  Where 4 is the number of kernel threads that you want.

- In order for the application to be multithreaded it must be linked with the /THREAD_ENABLE qualifier.

- The MULTITHREAD SYSGEN parameter controls the availability of kernel threads functions. With this parameter the following values can be specified:

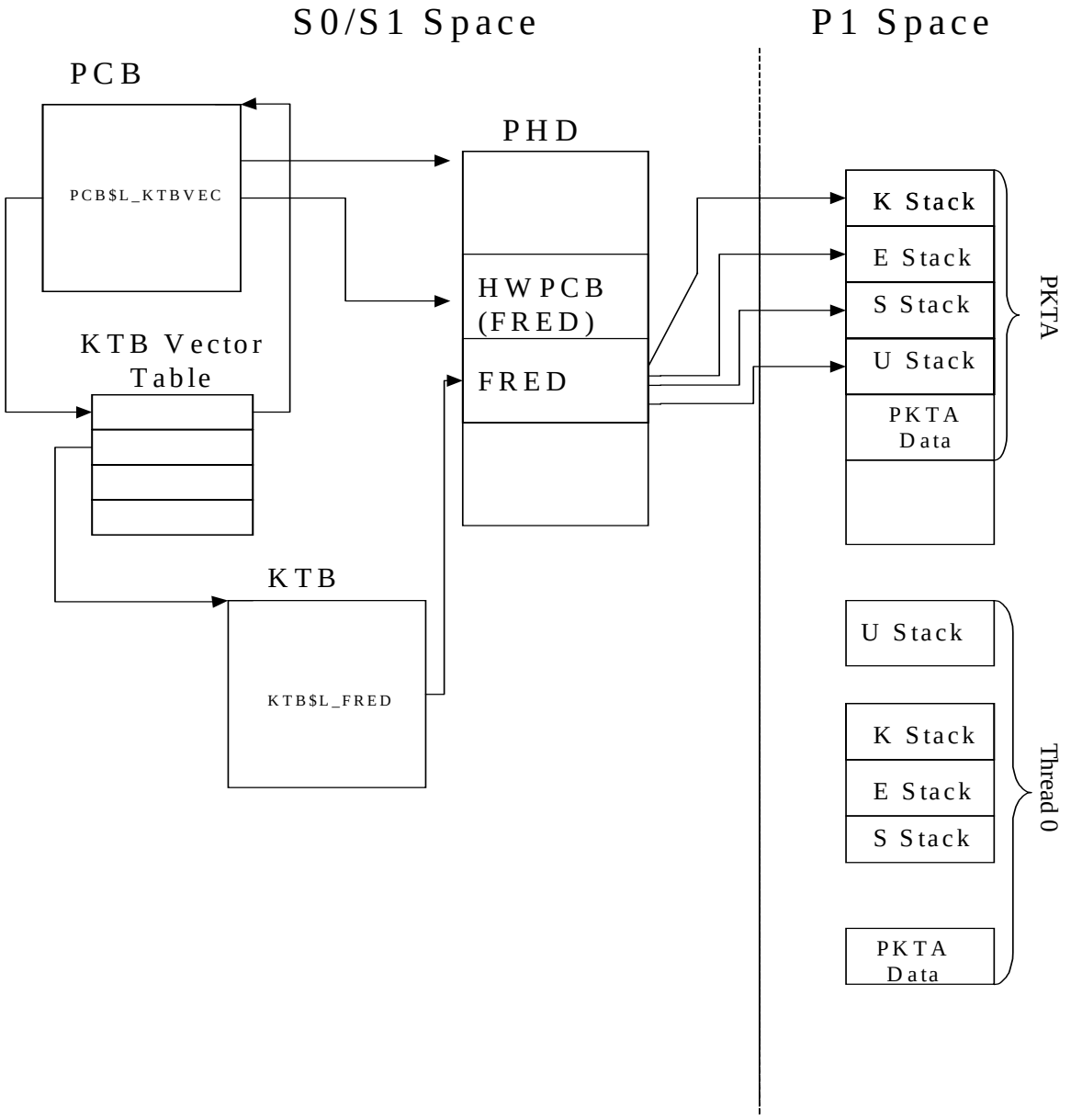
Value	Description
0	Both Thread Manager upcalls and the creation of multiple kernel threads are disabled.
1	Thread Manager upcalls are enabled; the creation of multiple kernel threads is disabled.
2-16	Both Thread Manager upcalls and the creation of multiple (Alpha kernel threads are enabled. The number specified only) represents the maximum number of kernel threads that can be created for a single process.

Kernel Thread Data Structures

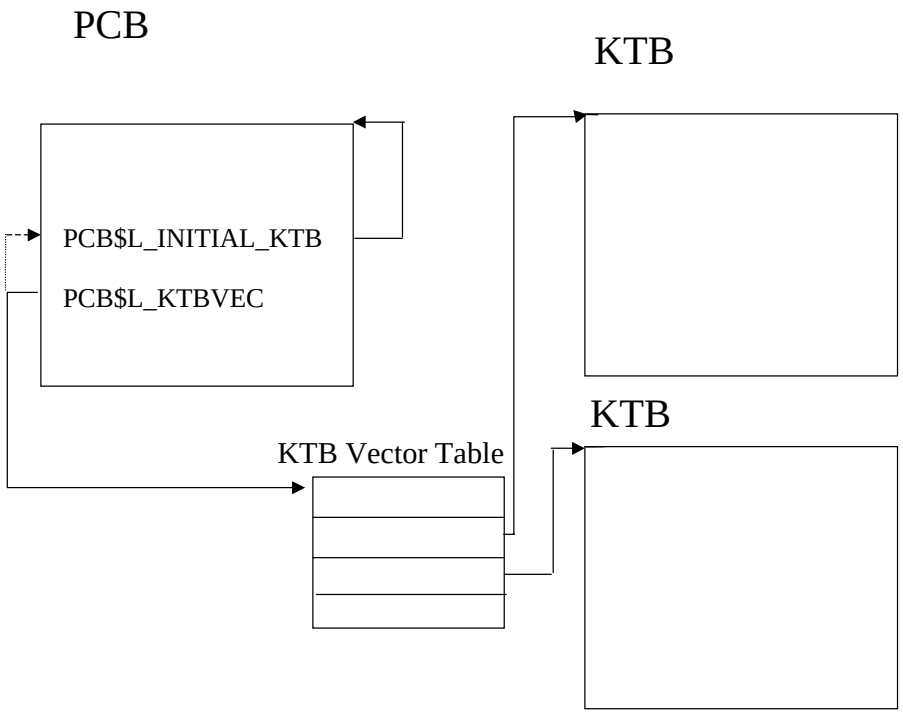
To support Kernel threading, the following data structures were added:

- Kernel Thread Block (KTB)
- Floating point Register Execution Data (FRED)
- Per Kernel Thread Area (PKTA)

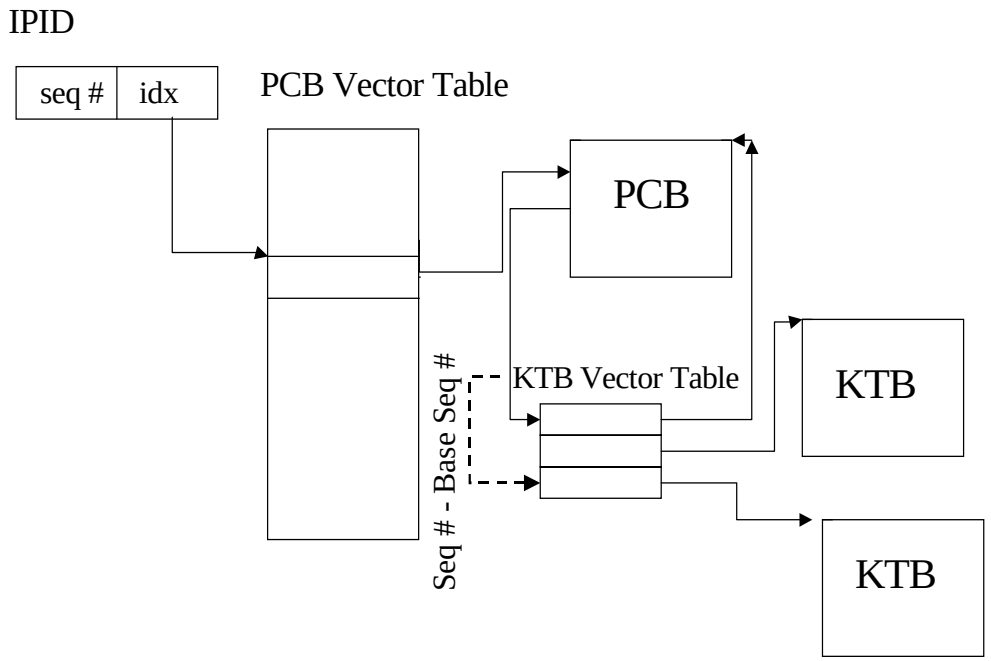
Kernel Thread Data Structures



KTB Vector Table



Locating the KTB Using the PID



Questions?

Wayne Sauer
President, PARSEC Group
sauer@parsec.com
www.parsec.com
888-4-PARSEC

HP Technology Forum & Expo

Get Connected!

Training | Knowledge | Solutions