Software Design Strategies for OpenVMS

Bruce Ellis

President

BRUDEN Corporation

HP WORLD 2003

Solutions and Technology Conference & Expo

Session overview

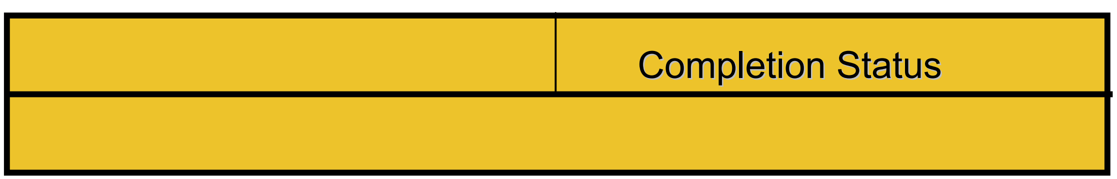
- Synchronization
  - Event flags
  - I/O status blocks
  - Asynchronous system traps (ASTs)
- Mailboxes
- Global sections
- Locks
- Doorbell Locks
- Design considerations
- Case studies and/or prototype programs

Asynchronous system services

- OpenVMS provides many system services (system APIs) that operate asynchronously
- For I/O
  – SYS$QIO[W], Intra-Cluster Communications (ICC) services, SYS$BRKTHRU[W], SYS$UPDSEC[W]
- For time-based events
  – SYS$SETIMR
- For obtaining information
  – SYS$GETJPI[W], SYS$GETDVI[W], SYS$GETLKI[W]
- For requesting locks
  – SYS$ENQ[W]

Design issues with asynchronous services

- When using asynchronous services it is important to:
  - Guarantee completion of the event
  - Determine when the event has completed
  - Determine the completion status of the event
- OpenVMS supports the following methods for detecting completion of asynchronous events
  - Event Flags
  - "I/O" status block



  - Asynchronous System Traps (ASTs)

Event flags

- Event flags are associated with events when calling asynchronous system services
  - 2 local event flag clusters (plus EFN$C_ENF)
  - 32 flags per cluster
  - Can wait on flags in one cluster at a time
  - Can wait for:
    - a specific flag (SYS$WAITFR) to be set
    - any flags in cluster (SYS$WFLOR) to be set
    - or all (SYS$WFLAND) to be set
  - OpenVMS sets the flags
  - Process placed in LEF scheduling state until event flag is set

"I/O" status block

- The success status of an asynchronous event is reported through the "I/O" status block (IOSB)
  - The I/O status block has a 32-bit and 64-bit form
  - Although, advertised as optional argument to system services, it should **not** be considered optional
  - Should have one unique IOSB per concurrent asynchronous event
- Can be used with SYS$SYNCH system service
  - Especially useful when possible concurrent duplication of event flags could occur in application

Design case study - Event with time out

- Customer is porting an application from PDP-11 to OpenVMS Alpha
  - Communicates with PDP through terminal line
  - Sends out a polling message using SYS$QIO (asynchronous form)
  - Waits one second
  - If PDP does not respond, sends another message and repeats
- When PDP does not respond, process eventually hangs in RWAST state due to quota exhaustion (BIOLM)
- Customer proposes monitoring process quotas

Design case study - Event with time out

- The following solution is proposed
  - Set a polling timer (3 seconds)
  - Poll (write to) PDP line
  - Read the line asynchronously, using event flag 33
  - Set a time to expire in 1 second that will set flag 34
  - Wait for either flag 33 or 34 to be set (SYS$WFLOR)
  - When one of the flags is set, read the event flag cluster (SYS$READEF) to determine which flag was set
  - If read completed, wait for poll interval to expire and issue next read
  - If time out, cancel I/O, report time out, and wait for next poll interval

Design case study - Event with time out

Sample run of poller

```
$ define terminal opa0:
%DCL-I-SUPERSEDE, previous value of TERMINAL has been superseded
$ r term_reader
PDP data: yes
Timestamp:  2-JUL-2003 00:09:16.98
Timeout!
Timestamp:  2-JUL-2003 00:09:31.98
PDP data: m
Timestamp:  2-JUL-2003 00:09:46.98
PDP data:
Timestamp:  2-JUL-2003 00:10:01.98
PDP data:
Timestamp:  2-JUL-2003 00:10:16.98
PDP data: Yes
Timestamp:  2-JUL-2003 00:10:31.98
PDP data: Yes
Timestamp:  2-JUL-2003 00:10:46.98
PDP data: Yes
Timestamp:  2-JUL-2003 00:11:01.98
PDP data: No
Timestamp:  2-JUL-2003 00:11:16.98
$
```

Design case study - Event with time out

Sample run from remote terminal

```
Are you there? yes
Are you there? yes ia
Are you there? m
Are you there?
Are you there?
Are you there? Yes
Are you there? Yes
Are you there? Yes
```

Took too long

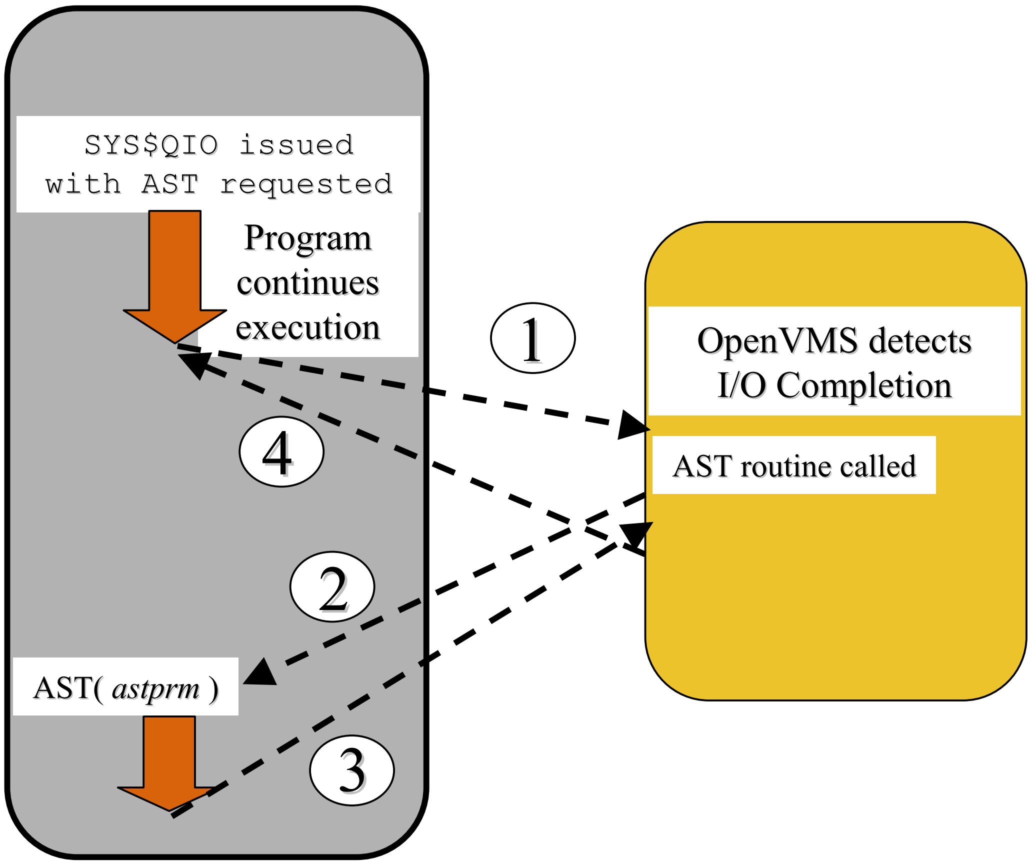
TERM_READER.C

Asynchronous system traps

- Asynchronous system traps (ASTs) are subroutines called by OpenVMS, asynchronous to the flow of execution, when an event completes
- System services allow you to identify one parameter (ASTPRM) that will be passed to the AST routine when it is called
- AST routine operating in a single threaded process cannot be preempted by another AST delivered in the same access mode (can be preempted by an AST in elevated access mode, e.g. Executive mode AST can preempt user mode AST)

Asynchronous system traps

AST flow



AST design considerations

- For debugging purposes AST routines should avoid accessing external data
  - Additionally, for synchronization data updated by both AST routine and mainline code requires "Load Lock/Store Conditional" sequence in mainline code
    - Can be implemented using __add_atomic_long( )
- Data can be encapsulated by passing a structure to the AST routine

Interprocess communication and synchronization

- When designing a system where multiple processes will support different functions within the system, processes need a method to communicate with each other
- OpenVMS provides several methods of interprocess communication, including:
  - Mailboxes
  - Global sections
  - Logical names
  - Shared files
  - Intracluster Communication (ICC) services
  - Lock value blocks

Mailboxes

- Mailboxes are pseudo-devices, similar to UNIX pipes, that support bi-directional communication
  - Can force unidirectional communication with argument to SYS$CREMBX
- Mailboxes must be created prior to use (SYS$CREMBX)
- Mailbox device names are of the form MBA*n*, where the unit number is generated dynamically
- Mailboxes are usually identified by logical names
- Reads complete when a corresponding write to the mailbox is issued and vice versa

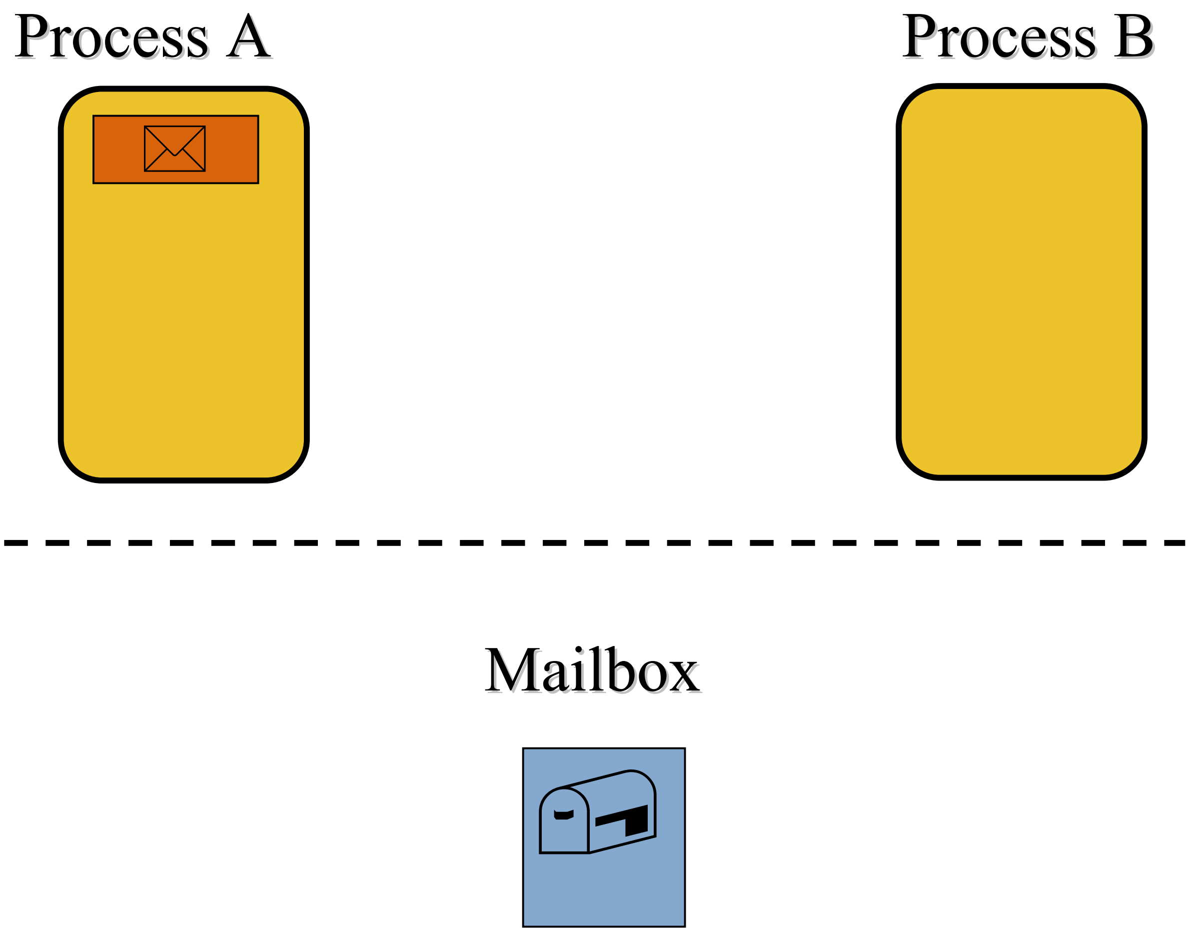
Mailboxes

- Mailboxes may be temporary or permanent
  - Temporary mailboxes:
    - Are deleted when the last channel is deassigned
    - Require TMPMBX privilege to create
    - By default, logical names go in job logical name table, can change with:

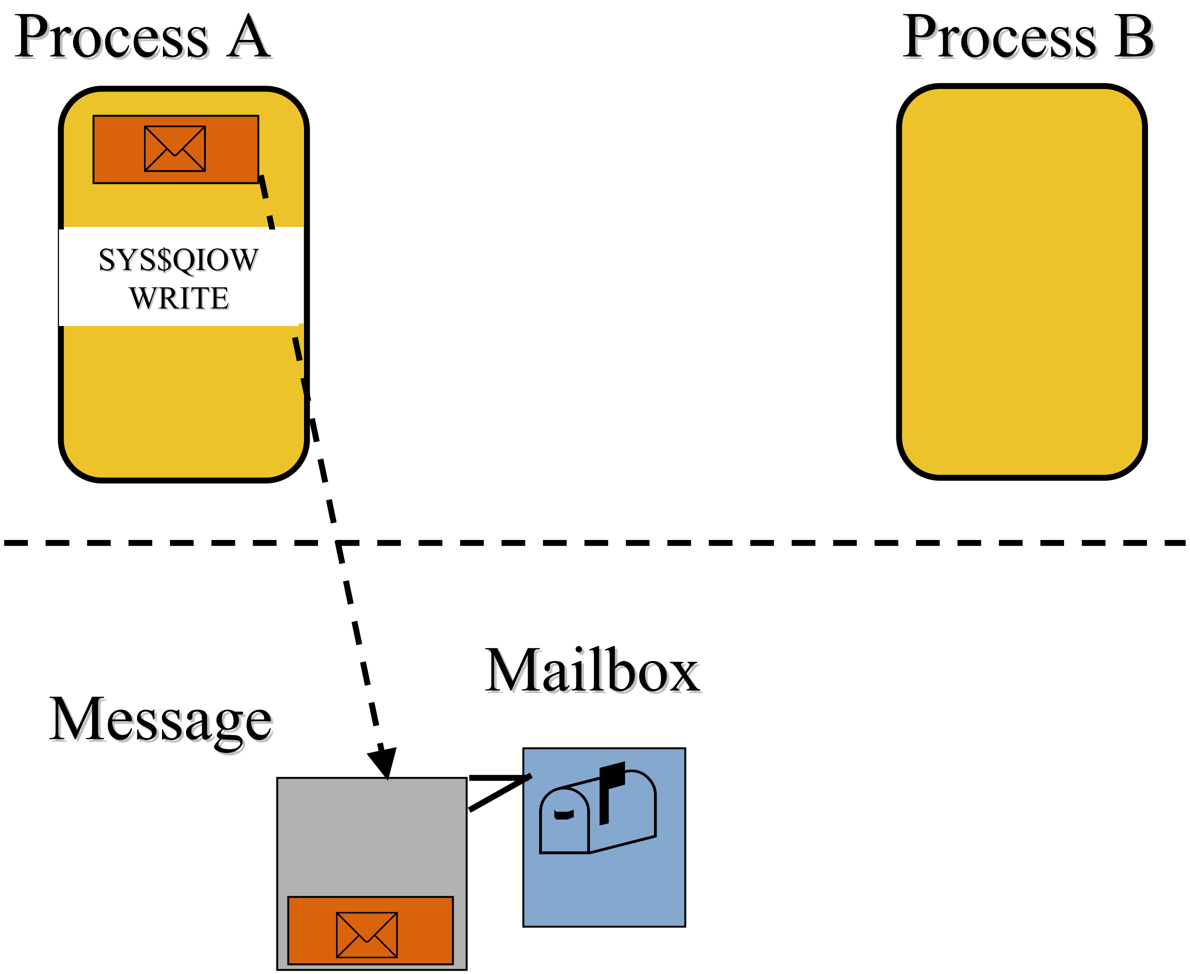
```
$ define/table=lnm$process_directory lnm$temporary_mailbox lnm$group
```

  - Permanent mailboxes:
    - Must be explicitly deleted (SYS$DELMBX)
    - Require PRMMBX privilege to create
    - By default, logical names go in system logical name table

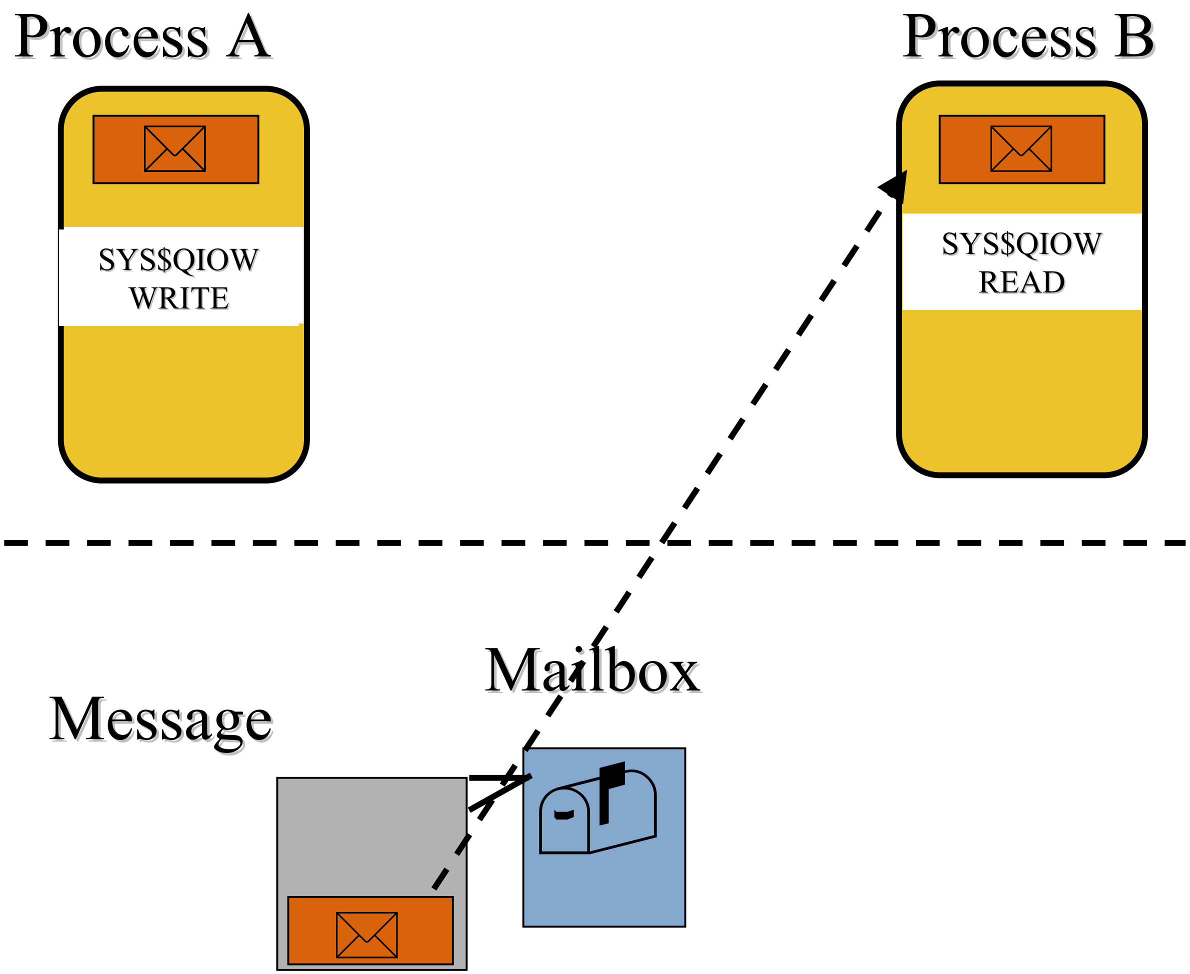
Mailbox mechanics



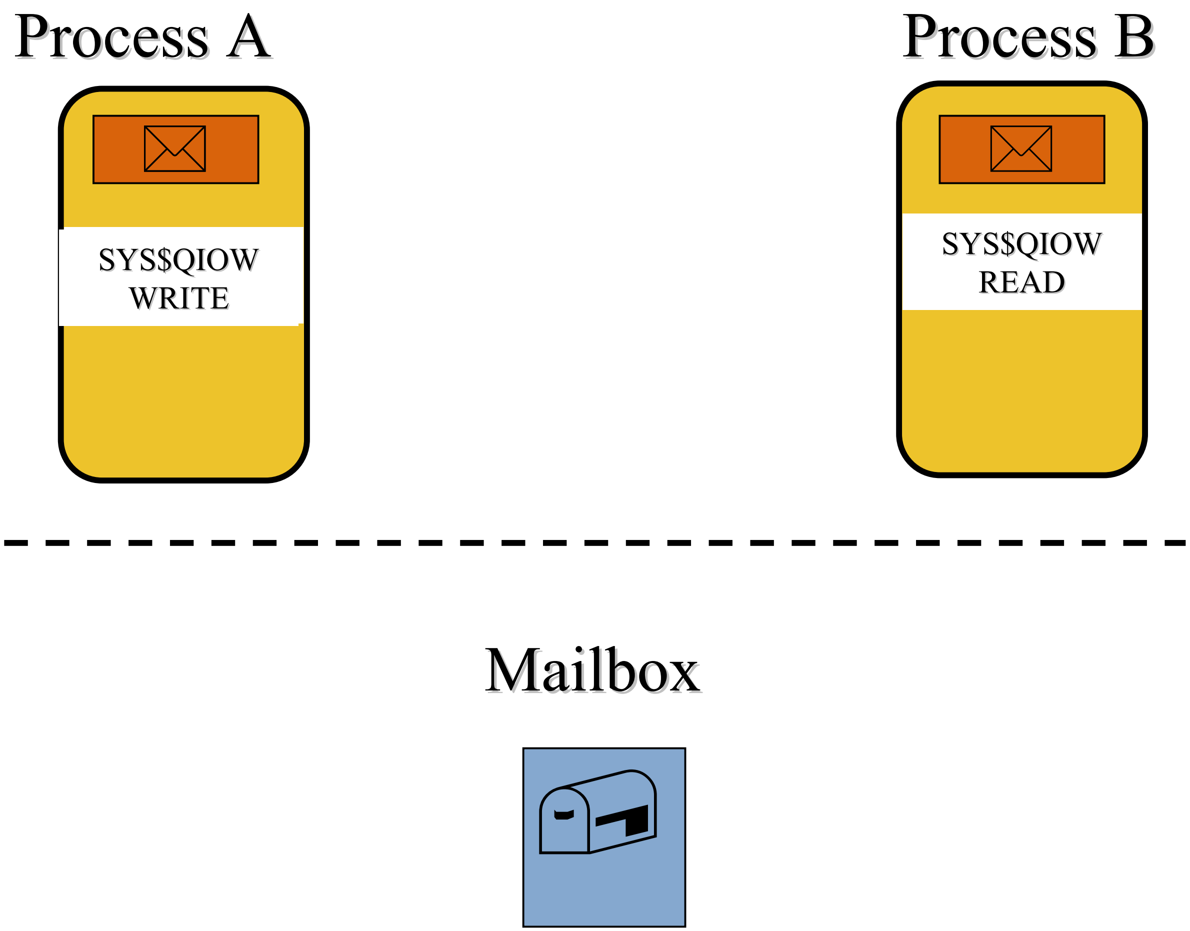
Mailbox mechanics



Mailbox mechanics

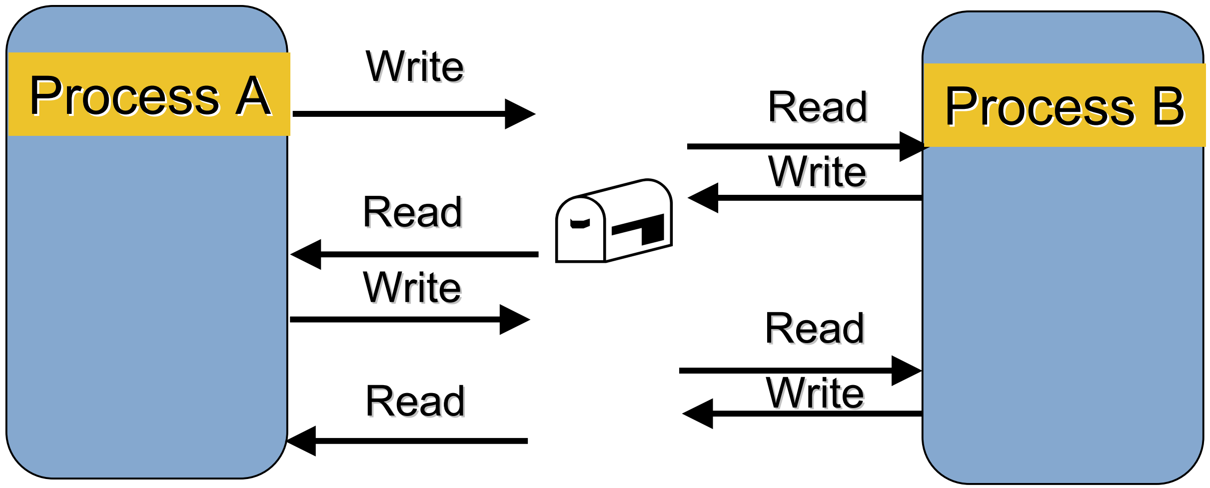


Mailbox mechanics



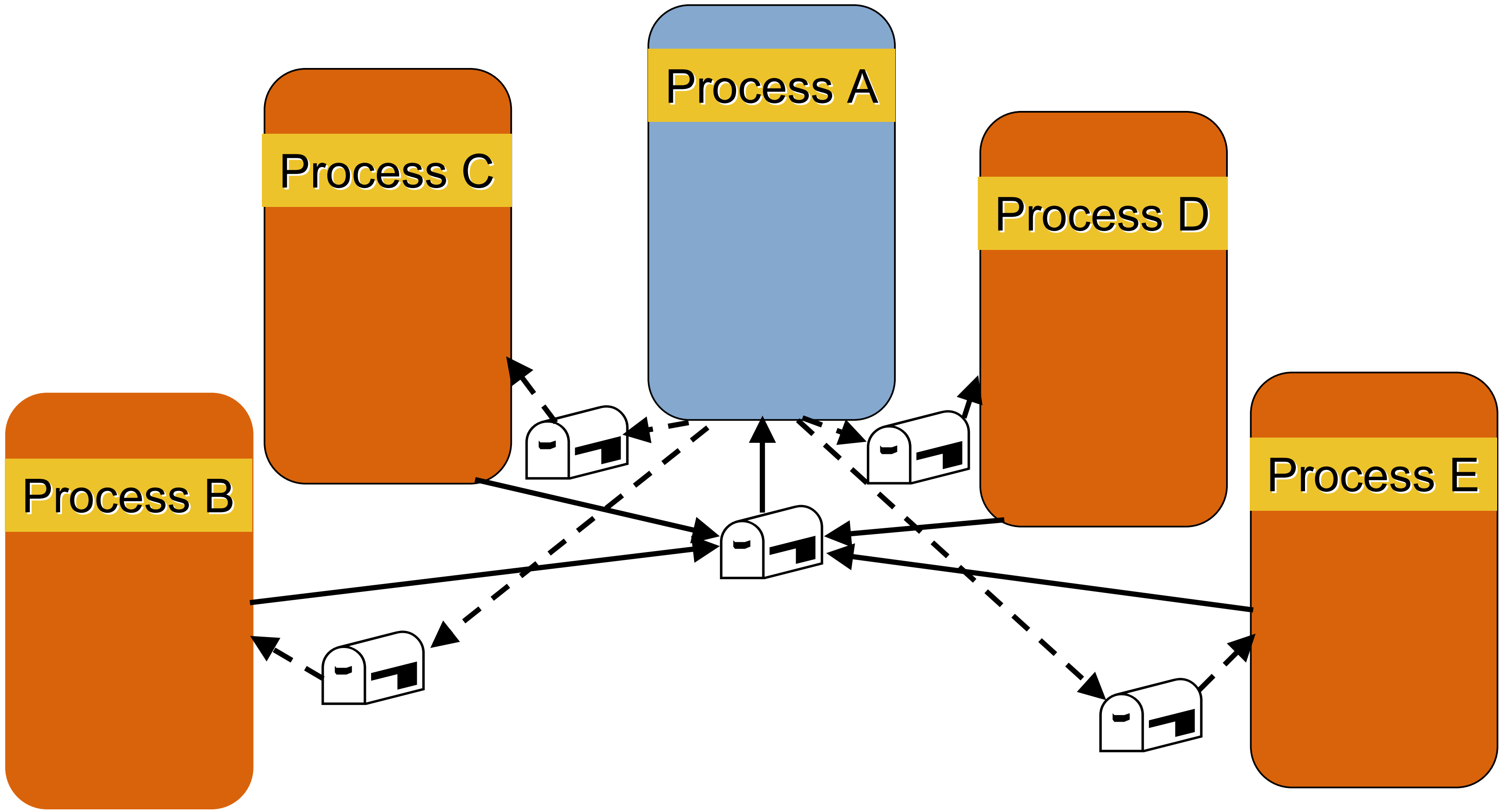
Mailbox implementations

One possible mailbox implementation



Mailbox implementations

More common mailbox implementation



Mailbox design considerations

- Mailboxes provide relatively fast communication
  - ~9,500 read/write pairs of 128 bytes per second on ES40 with 2 500Mhz CPUs
- Mailboxes support event notification of writes/reads through SYS$QIO interface
- When the reader thread is slow to respond mailboxes can back up
  - If mailbox fills up, writers can stall in RWMBX state
- Great care should be taken when designing a server process
- Use caution with asynchronous reads/writes, can read own messages

Design case study - Mailbox implementation

Customer requirements

- Create processes (pvcs) to handle X.25 communications with remote sites
- Process names defined by site, input from TERMDATA.DAT file
- Messages from processes to be logged in a daily log file
- Synch messages to file on regular basis
- Log file reopened daily
- Detect process failure, if detected log and restart processes

Design case study - Mailbox implementation

Design considerations

- Central process will handle log files and read data passed from site processes
- To handle failure detection, we use termination mailbox messages
  - Create process system service (SYS$CREPRC) supports a mbxunt parameter
  - On deletion/failure of a process an accounting message is written to the mailbox unit
  - PID is in accounting message, process name is not
  - To map PIDs to process names, an array of structures is filled in that maps PIDs to process names

Design case study - Mailbox implementation

Design considerations

- Central process is entirely event driven
- It sets up ASTs to handle:
  - Reads from process (pvc) mailboxes
  - Reads from termination mailboxes
  - Synchs for files
  - Reopening log files
- After setup, the process hibernates (SYS$HIBER)
  - A wake call (SYS$WAKE) will cause process to run down

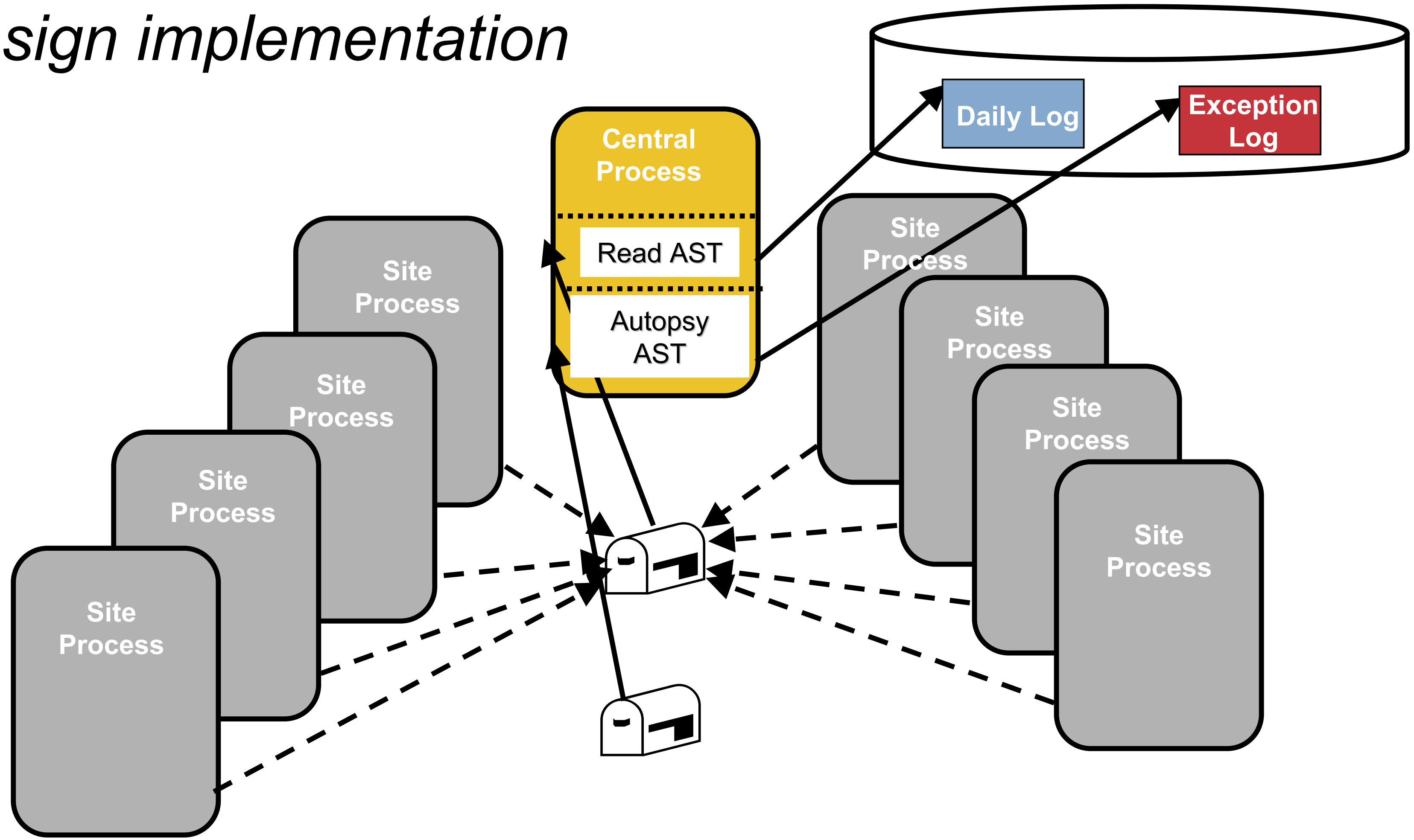
Design case study - Mailbox implementation

Design considerations

- AST routines are passed pointers to structures (through ASTPRM) that allow them to access:
  - File pointers to daily log and exception files
  - IOSB associated with read
  - Read buffer (accounting buffer pointer is maintained separately to avoid casting pointer)
  - Mailbox unit
- Upon completion of processing, another asynchronous read is issued on the mailbox

Design case study - Mailbox implementation

Design implementation



Design case study - Mailbox implementation

Design implementation

- Master process flow
  - Create daily and exception log files
  - Create permanent mailbox for general communications
  - Create temporary mailbox for termination messages
  - Create processes with termination mailbox
  - Set timers to:
    - Flush log files
    - Close and reopen log files daily
  - Post asynchronous read to mailbox with AST routine to be called on completion
  - Hibernate

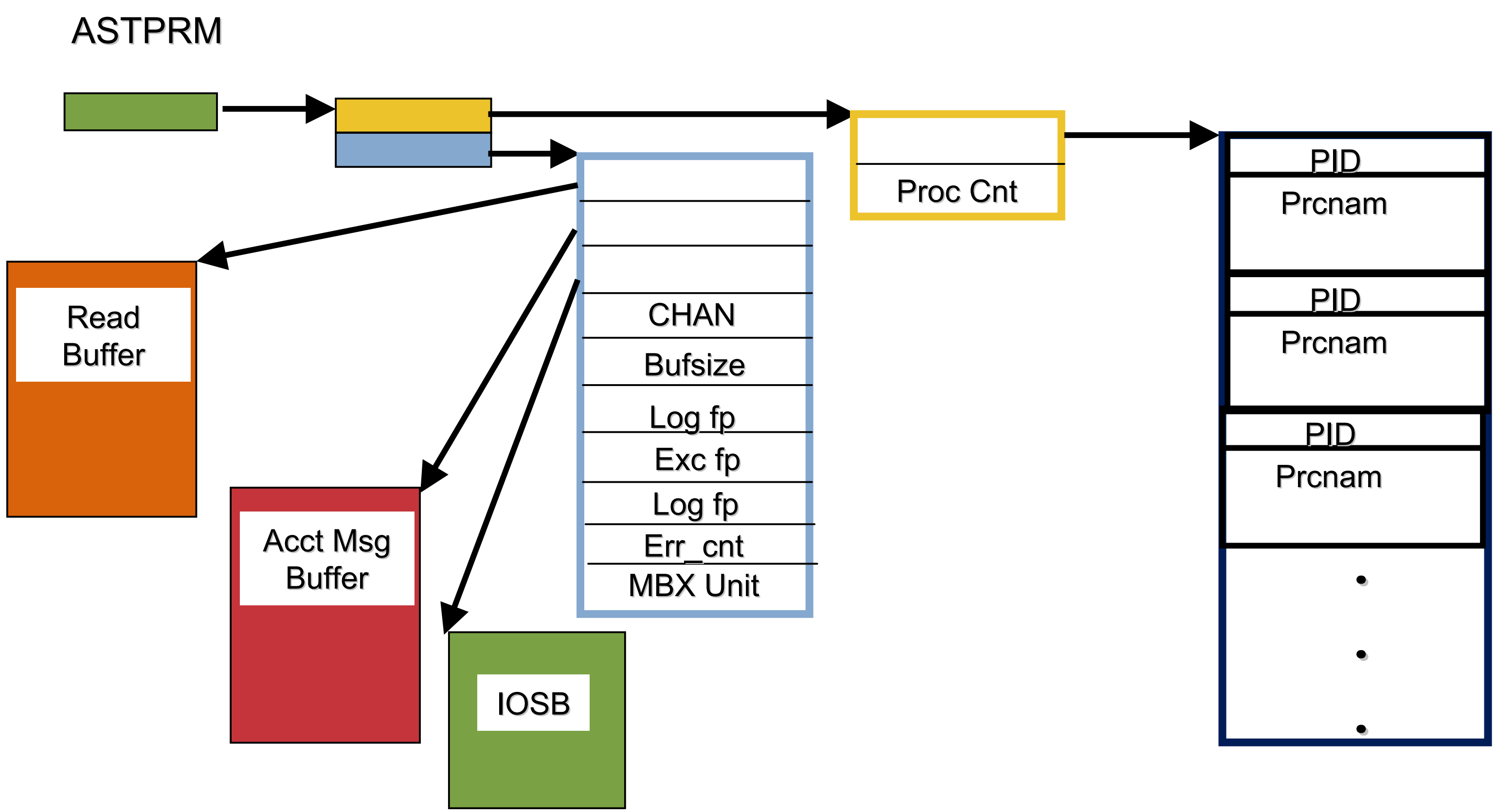
Design case study - Mailbox implementation

Design implementation

- Process creation flow
  - Read process names from TERMDATA.DAT file
  - Loop and create processes with same termination mailbox unit
  - Post asynchronous read on mailbox unit, specifying *autopsy* AST routine
  - AST parameter is a structure with pointers to structure for mailbox and structure for process table
- MBX_READER.C

Design case study - Mailbox implementation

Design implementation



Design case study - Mailbox implementation

- A prototype of the mailbox writer (pvc processes) was written that fabricated a 128 record and wrote it at random intervals (each site process used thismechaism
  - Intervals of 1 to 30 seconds were chosen to emulate the user environment
  - Intervals of 10 to 300 milliseconds were chosen to emulate stress conditions
- MBX_WRITER.C (Both reader and writer use MBX_REC.H)

Design case study - Mailbox implementation examples

System startup

Termdata file used to define process names.

```
$ type termdata.dat
KLWN01KELOWNA/HANY01          83600131000000176219992503721
KAML01KAMLOOPS/HANY06         83501166000000237269992503726
CTWD01CHETWYND VHF #1         83700381000000378809992507846
VANC02Vancouver VHF #2        83100292000000466299996046897
VERN01VERNON/HANEY04          83600021000000577060992503729
FSJN02FT ST JOHN VHF #2       83700228000000678769992507847
PGRG01PRINCE GEORGE VHF #1    65100102000000756594992505659
VICT01VICTORIA/ HANY02        67102932000000838954992502864
NNIM01NANAIMO/ HANY03         83102148000000975504992502863
CBRV01CAMPBELL RIVER VHF #1   68350011000001028624992502860
DWCK01DAWSON CREEK VHF #1     65200010000001178494992507849
...
```

Command to kick off reader, which in turn starts site processes.

```
$ run/detach/privileges=all/process=ttrama mbx_reader
%RUN-S-PROC_ID, identification of created process is 20400491
$
```

Design case study - Mailbox implementation examples

System processes

```
$ show system
OpenVMS V7.3-1  on node ALPH40   7-JUL-2003 21:05:56.54  Uptime  0 10:11:07
  Pid     Process Name     State  Pri      I/O       CPU          Page flts  Pages
...
20400491 TTRAMA           HIB      5     2572   0 00:00:00.03        113    130
20400492 KLWN01           LEF      6       95   0 00:00:00.00         73     86
20400493 KAML01           LEF      4       96   0 00:00:00.02         73     86
20400494 CTWD01           LEF      6       86   0 00:00:00.01         73     86
20400495 VANC02           LEF      5       86   0 00:00:00.00         73     86
20400496 VERN01           LEF      6       84   0 00:00:00.01         73     86
20400497 FSJN02           LEF      6       91   0 00:00:00.01         73     86
20400498 PGRG01           LEF      6       82   0 00:00:00.01         73     86
20400499 VICT01           LEF      6       92   0 00:00:00.00         73     86
2040049A NNIM01           LEF      6       93   0 00:00:00.01         73     86
2040049B CBRV01           LEF      6       96   0 00:00:00.01         73     86
2040049C DWCK01           LEF      6       92   0 00:00:00.01         73     86
2040049D CRBK01           LEF      6       83   0 00:00:00.01         73     86
2040049E NLSN01           LEF      6       90   0 00:00:00.02         73     86
2040049F TRRC01           LEF      6       84   0 00:00:00.00         73     86
204004A0 WLLK01           LEF      6       76   0 00:00:00.03         73     86
204004A1 PTHY01           LEF      6       94   0 00:00:00.01         73     86
204004A2 PNTN01           LEF      6       94   0 00:00:00.04         73     86
204004A3 QSNL01           LEF      6      100   0 00:00:00.00         73     86
204004A4 SMTR01           LEF      6       74   0 00:00:00.01         73     86
204004A5 FSJN01           LEF      6       83   0 00:00:00.00         73     86
204004A6 MAST01           LEF      6       83   0 00:00:00.01         73     86
204004A7 PGRG02           LEF      6       91   0 00:00:00.00         73     86
204004A8 VANC03           LEF      6       85   0 00:00:00.01         73     86
204004A9 PTHY02           LEF      6       90   0 00:00:00.00         73     86
204004AA FTNL01           LEF      6       84   0 00:00:00.01         73     86
204004AB PTAI01           LEF      6       91   0 00:00:00.01         73     86
204004AC PRRT01           LEF      6       86   0 00:00:00.01         73     86
204004AD KAML02           LEF      6       91   0 00:00:00.01         73     86
204004AE BULK01           LEF      6       86   0 00:00:00.01         73     86
204004AF VANEVA           LEF      6       86   0 00:00:00.02         73     86
$
```

Design case study - Mailbox system load under stress

Load of 1 write per process at random intervals from 10 to 300 milliseconds (anticipated activity is 1 write/15 seconds)

```
$ monitor modes,process/topcpu,io,disk

                              OpenVMS Monitor Utility
                               TOP CPU TIME PROCESSES
                                   on node ALPH40
                               7-JUL-2003 21:07:09.21

                                       0         25        50        75       100
                                       + - - - - + - - - - + - - - - + - - - - -+
 20400491  TTRAMA
 20400490  ELLIS
...
                              OpenVMS Monitor Utility
           +------+           TIME IN PROCESSOR MODES
           | CUR  |                on node ALPH40
           +------+            7-JUL-2003 21:07:12.21

Combined for  2 CPUs                   0         50        100       150      200
                                       + - - - - + - - - - + - - - - + - - - - -+
 Interrupt State                       |
 MP Synchronization                    |
 Kernel Mode                           |
 Executive Mode                        |
 Supervisor Mode                       |
 User Mode                             |
 Compatibility Mode                    |
 Idle Time                         199 |***************************************
                                       + - - - - + - - - - + - - - - + - - - - -+
```

Design case study - Mailbox system load under stress

With ~200 mailbox writes/second system overhead is negligible

```
                              OpenVMS Monitor Utility
                               I/O SYSTEM STATISTICS
                                 on node ALPH40
                              7-JUL-2003 21:07:15.21

                                     CUR        AVE        MIN        MAX

    Direct I/O Rate                  3.66       2.99       2.66       3.66
    Buffered I/O Rate              402.66     400.57     384.00     419.00
    Mailbox Write Rate             196.33     195.88     187.33     205.33
...
                              OpenVMS Monitor Utility
                                DISK I/O STATISTICS
                                 on node ALPH40
                              7-JUL-2003 21:07:18.21

I/O Operation Rate                         CUR        AVE        MIN        MAX

$1$DGA42:     (ALPH40) ES40                4.00       4.13       3.66       4.66
$1$DGA142:    (ALPH40) RAID                0.00       0.00       0.00       0.00
...
```

Design case study - Mailbox system load under stress

Operations to mailbox is double the write rate.

```
$ @mb_mon
******************
MBA1       operations:      119  ops/sec:       11.9
MBA2       operations:       65  ops/sec:        6.5
MBA3       operations:        2  ops/sec:        0.2
...
MBA56      operations:       10  ops/sec:        1.0
MBA57      operations:        2  ops/sec:        0.2
MBA68      operations:   513848  ops/sec:    51384.8
Enter Control Y to exit.
******************
MBA68      operations:   517726  ops/sec:      387.8
Enter Control Y to exit.
******************
MBA68      operations:   521620  ops/sec:      389.4
Enter Control Y to exit.
******************
MBA68      operations:   525482  ops/sec:      386.2
Enter Control Y to exit.
 Interrupt
$ show logical Bruce_mbx
   "Bruce_mbx" = "MBA68:" (LNM$SYSTEM_TABLE)
$ show device mba68:/full

Device MBA68:, device type local memory mailbox, is online, record-oriented
    device, shareable, mailbox device.

    Error count                     0    Operations completed             420786
    Owner process                  ""    Owner UIC                       [ELLIS]
    Owner process ID         00000000    Dev Prot    S:RWPL,O:RWPL,G:RWPL,W:RWPL
    Reference count                31    Default buffer size                 128

$
```

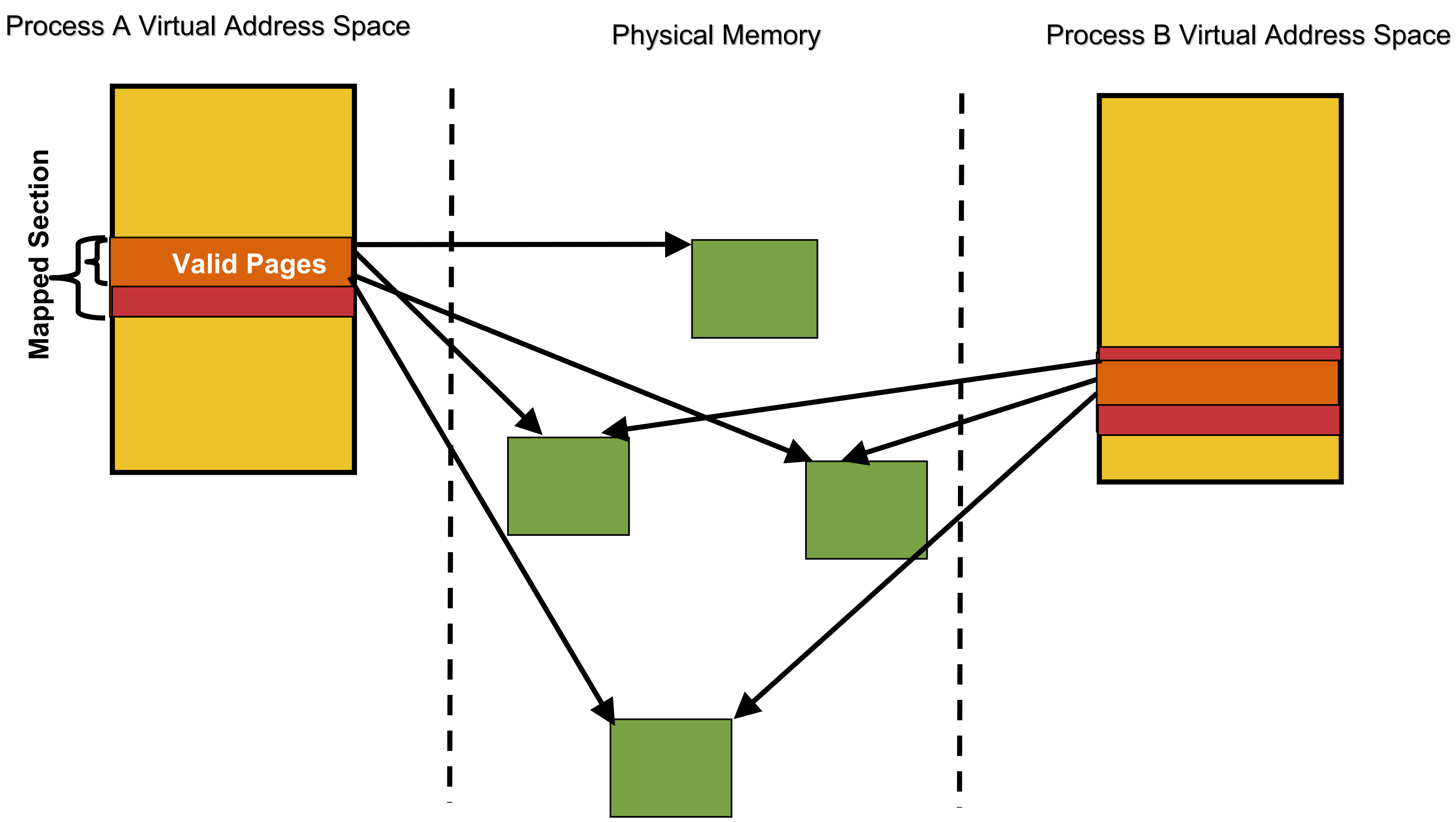
Global sections

- Global sections are memory sections that can be commonly "mapped" by multiple processes
- Global sections are created using the create and map section system service (SYS$CRMPSC/SYS$CRPMSC_64)(64-bit services allow a section to be created for later mapping by processes)
  - Creating a section constructs system data structures (global section descriptors, global section table entries, and global page table entries), that describe the section to the system and other processes

Global sections

- Mapping to a global section is performed through a map global section system service (SYS$MGBLSC/SYS$MGBLSC_64)
  - Mapping to a global section constructs process page table entries that can be used to locate the corresponding global page table entries
- Address space for global sections is commonly allocated dynamically, using the expand region (SEC$M_EXPREG) flag
- Global sections may be mapped to different addresses for different processes, in fact may be mapped to different virtual address space regions

Global section mechanics



Global section design considerations

- Once the section has been mapped, processes access the section (normally through pointers) as they would access local memory
- Some form of synchronization must be employed while accessing the section to guarantee data integrity
- Synchronization techniques include
  - Load lock/store conditional instruction sequences
  - Interlocked queues
  - Locks

Global section design considerations

- Global sections are preferable over mailboxes in high concurrency situations
- On an ES40 (with 2 500Mhz CPUs) a process can write ~15,000 128-byte messages/second to a section, with no synchronization
  - With a simple locking scheme, ~12,000 128-byte messages can be written per second
- Drawbacks to global sections include:
  - No built-in signaling mechanism for write/read notification (can be implemented with doorbell locks)
  - Limited to a single node (you can use galaxywide shared memory sections to share data between galaxy instances)

Global section design considerations

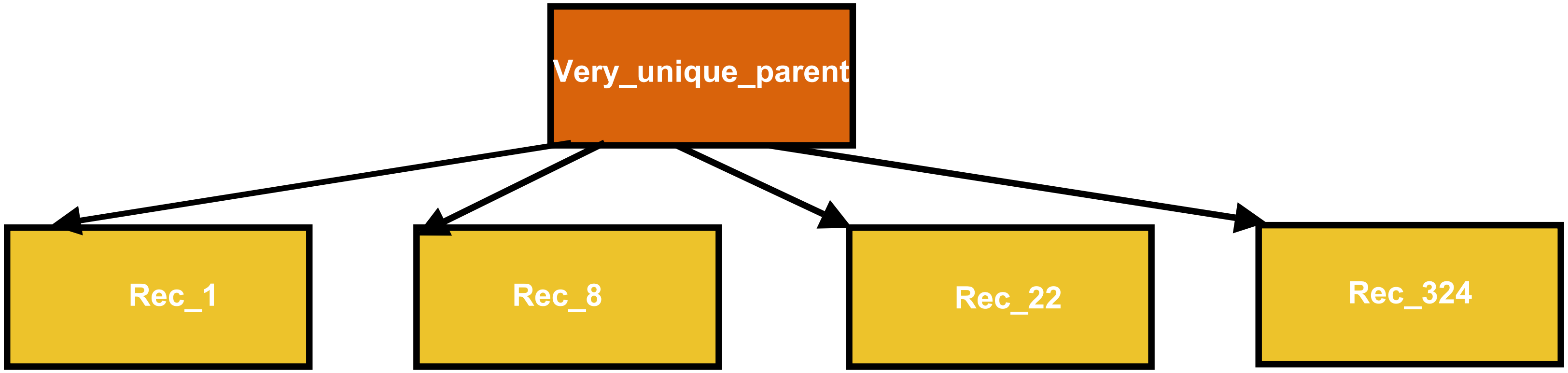
- Global sections can be:
  - Backed to a section file
    - Persistent data
    - Can be updated (SYS$UPDSEC)
  - Backed to the page file
  - Memory resident
    - Not charged against the process' working set list
    - Not backed to disk

Locks

- OpenVMS lock management provides a synchronization method
- You lock on a uniquely named resource (could be a file, record, structure in a global section, or even a mythical resource), using the SYS$ENQ system service
  - Name space is system-wide (LCK$M_SYSTEM), group-wide, or within a resource domain
- Locks are requested in a given access mode (NL, CR, CW, PR, PW, or EX)
  - Compatible locks are granted, incompatible locks are placed in a queue until compatible

Locks

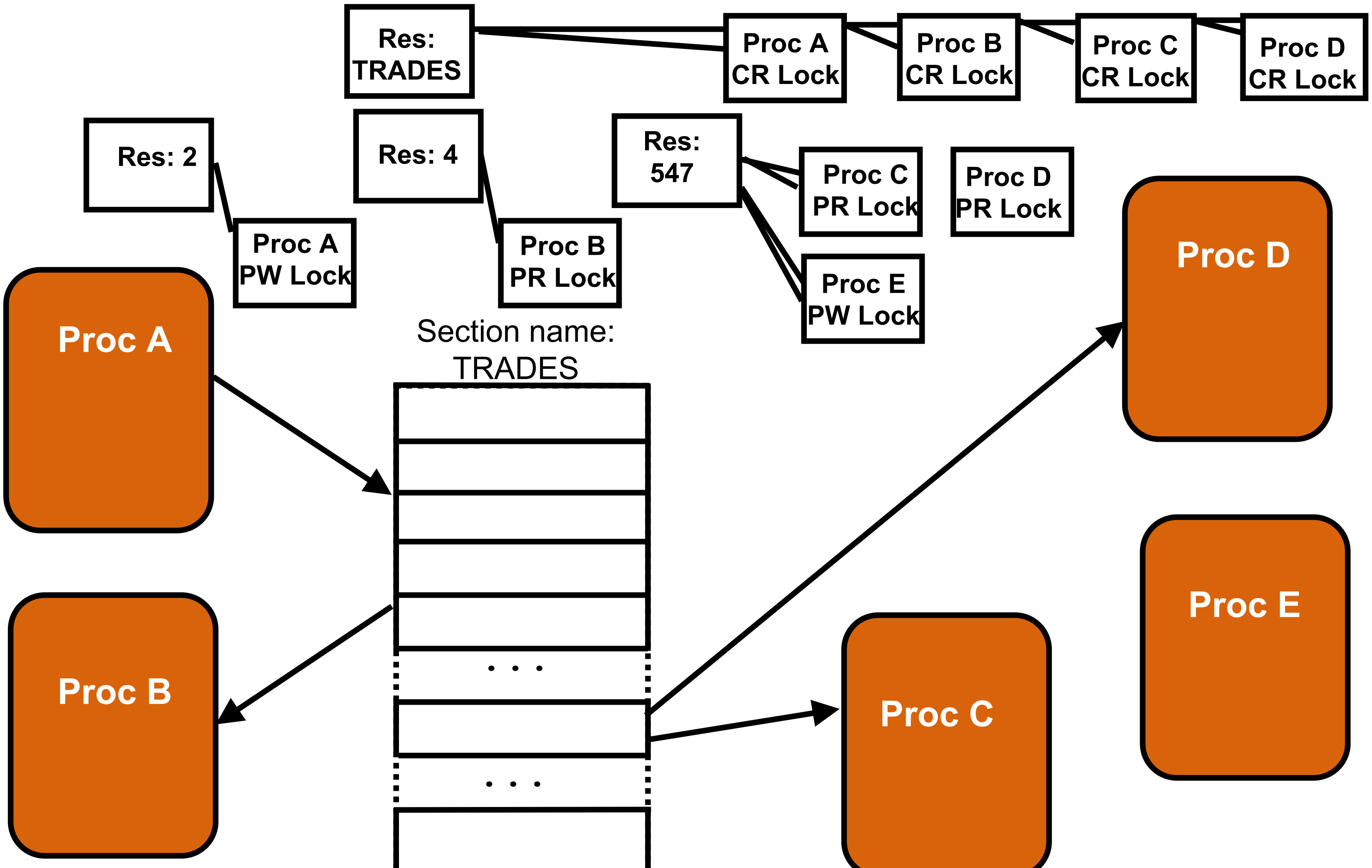
- A unique lock identification is returned to the caller of SYS$ENQ
- Lock management supports sublocks
  - Useful for providing a unique naming convention, sub-resources only require unique names within the sub-resource hierarchy
    - Example: file lock and record lock



Lock features

- The lock manager supports many features
  - Resource value blocks
    - Can be used to identify changes to resources (16-bytes of data)
  - Lock conversions
    - Preserve the value block
    - Slightly faster than SYSENQ/SYSDEQ/SYS$ENQ
  - Blocking ASTs
    - Notification that a process is being blocked

Sample lock and global section implementation



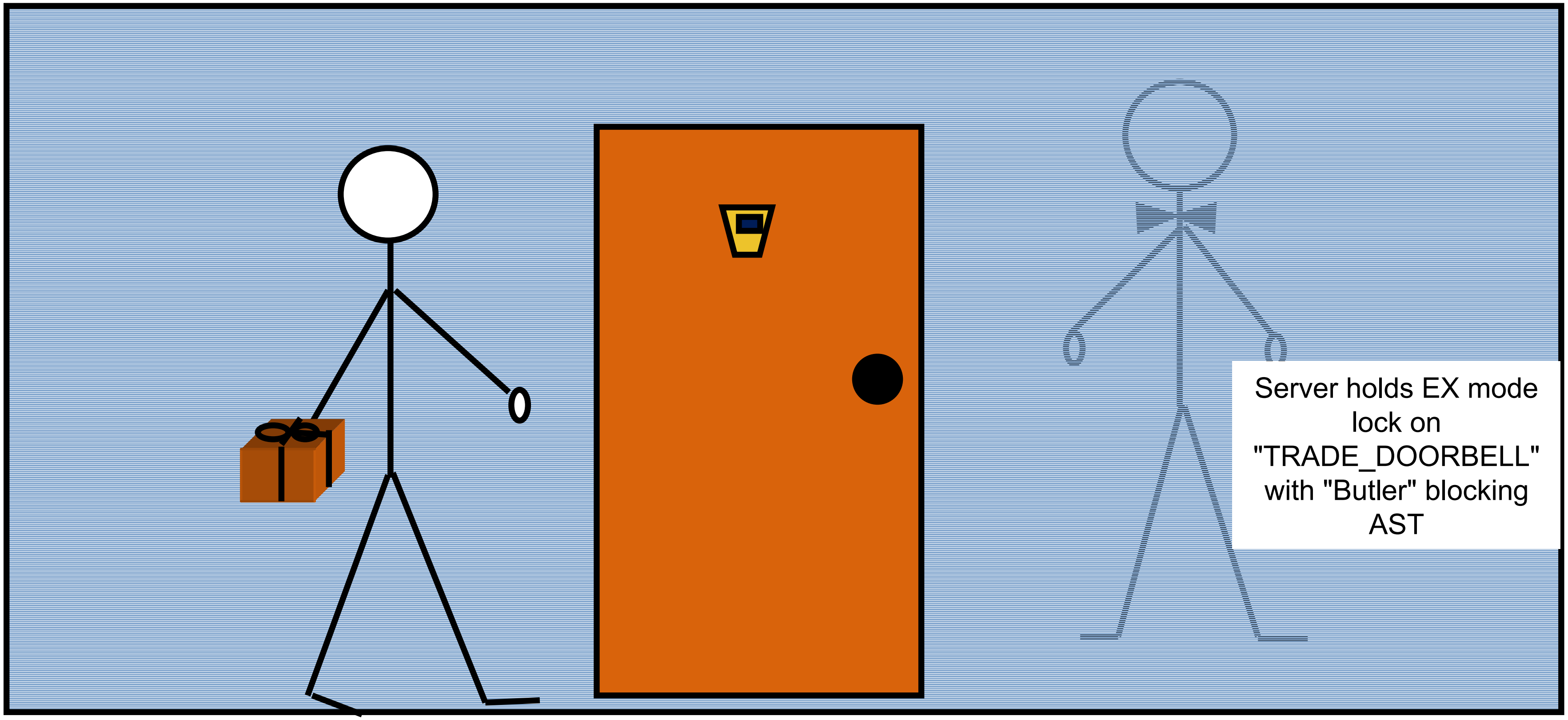
Sample lock and global section programs

- This implementation is illustrated by the programs:
  - CREATE_SEC.C
    - Creates a permanent global section named "trades"
  - SEC_WRITER.C
    - Writes one million random records to random indices in the section
    - Displays the number of messages written per second
  - SEC_READER.C
    - Reads one million random messages and displays the first 20 characters of each message
  - All programs use locks

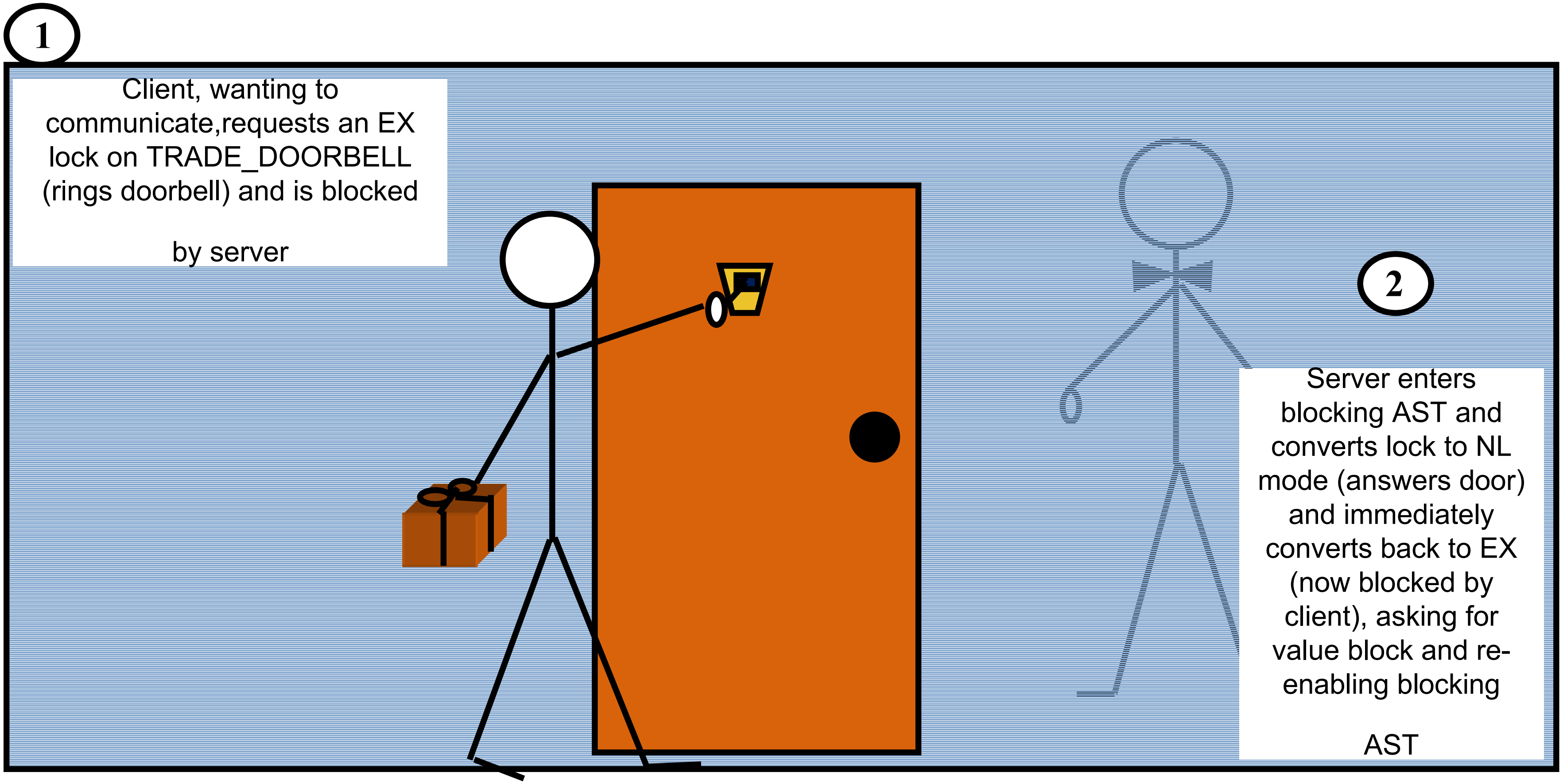
Doorbell locks

- Technically there is no such thing as a "doorbell lock"
  - There is no argument on the SYS$ENQ system service that asks for a doorbell lock. There is a doorbell communication method that takes advantage of the following characteristics of the locking mechanism:
    - Blocking ASTs
    - Value blocks
    - Lock conversions
- The name "doorbell lock" is a slang reference to locks that use these mechanisms to support inter-process communication on a single node or clusterwide

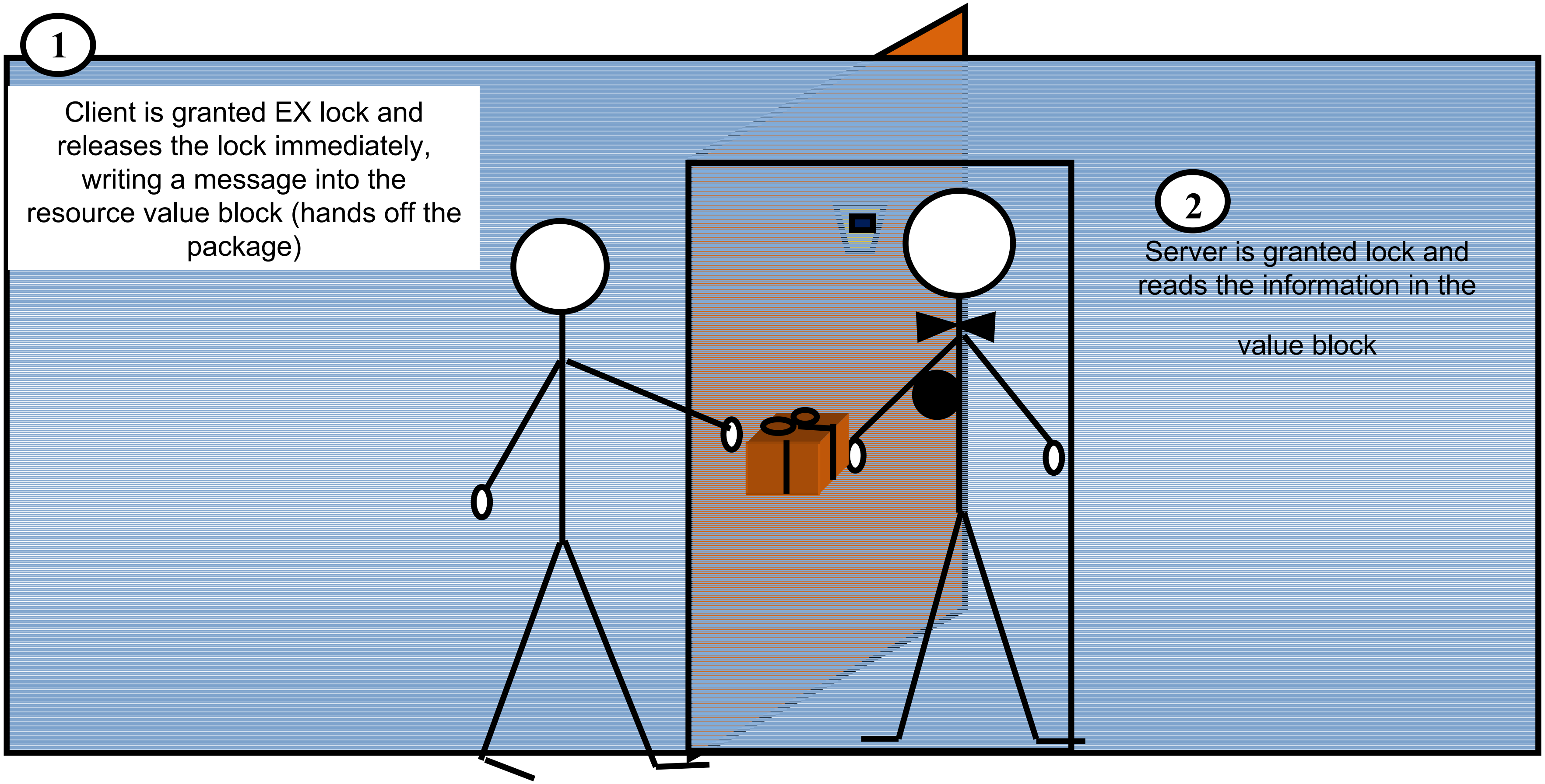
Doorbell lock concepts



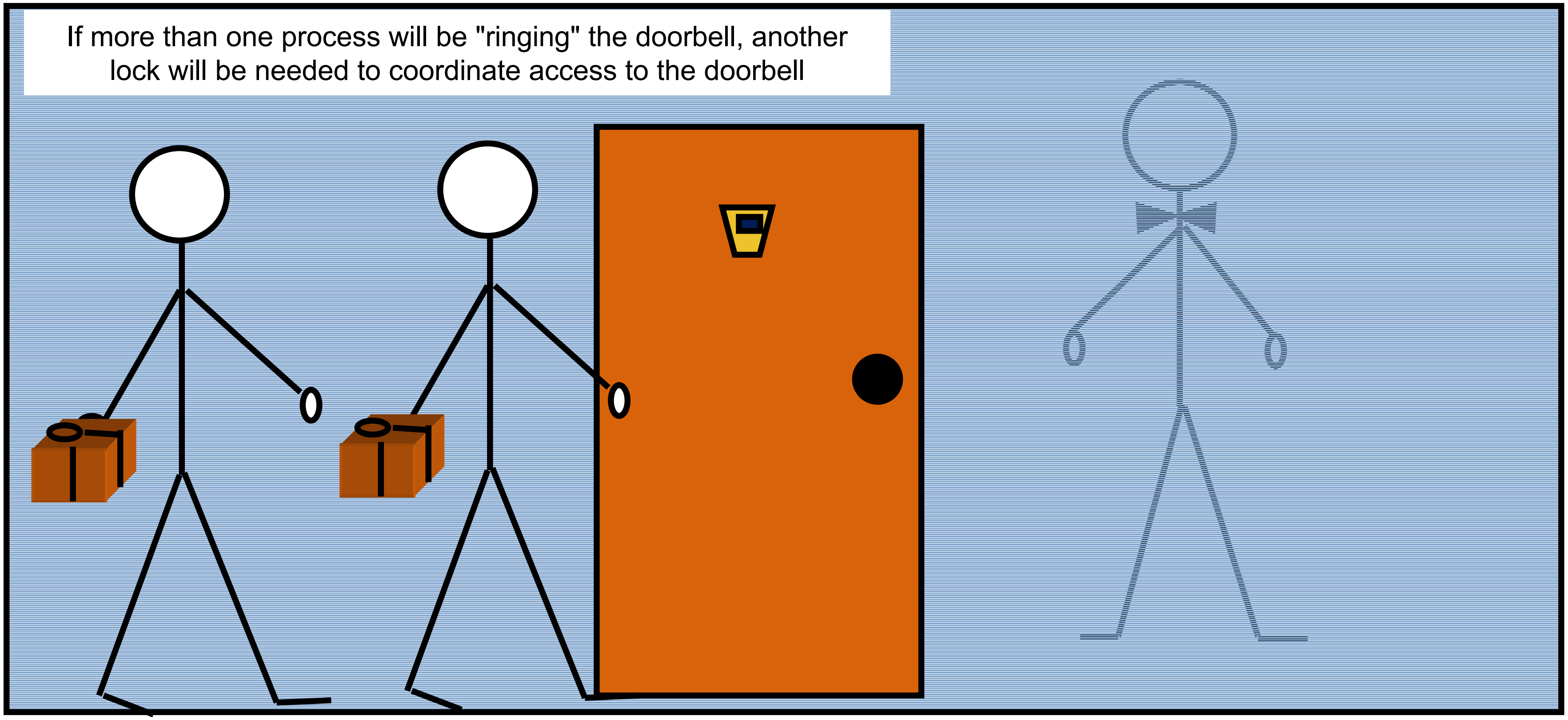
Doorbell lock concepts



Doorbell lock concepts



Doorbell lock concepts



Sample doorbell implementation

- The sample doorbell implementation performs the following steps:
  - A separate program creates a global section called "TRADES" (CREATE_SEC.C)
    - Global section consists of one million 128-byte structures

Sample doorbell implementation

- A server process (SERVER_SEC.C) maps the global section
  - A CR mode lock is taken out by the server on the section
  - The server sets up a doorbell lock request and goes to sleep (SYS$HIBER)
  - When waken, the server takes out a sublock of the "TRADES" lock, on the structure (using index as name)
  - The server displays the record and goes back to sleep
    - In a real application, the record might be updated in a database

Sample doorbell implementation

- A client process (CLIENT_SEC.C) maps the global section
  - Takes out a CR mode lock on the TRADES section
  - It prompts for user input (an ID number and a string)
  - The ID is used as an index into the section
  - The ID also forms the sub-resource name for a lock taken out in PW mode
  - The data is copied into the section
  - The doorbell is rung

Doorbell implementation considerations

- The doorbell scheme used here allows ~5,000 messages per second on an ES40 with 2 500Mhz CPUs

Doorbell implementation sample run

Server Session

```
$ run server_sec
Item 1700 updated
Data is Max Ellis
Item 500000 updated
Data is Wim Wenders
$
```

Client Session

```
$ run client_sec
Enter id: 1700
Enter name: Max Ellis
$
$ run client_sec
Enter id: 500000
Enter name: Wim Wenders
$
$
$ run client_sec
Enter id: 1000000
$
```

For additional information

- For more information on these techniques, see:
  - OpenVMS Programming Concepts Manual
- Additionally, HP Educational Services offers the following courses:
  - OpenVMS Alpha Programming Features I
  - OpenVMS Alpha Programming Features II
- Electronic forms of the sample programs can be downloaded from www.BRUDEN.com
- Questions?
  - Bruce.Ellis@BRUDEN.com

HP WORLD 2003
Solutions and Technology Conference & Expo

Interex, Encompass and HP bring you a powerful new HP World.

interex
shared knowledge • shared power

encompass
AN HP USER GROUP

hp
invent