Figure 16.1 The distributed shared memory abstraction

Distributed shared memory

Process accessing DSM

DSM appears as memory in address space of process

Physical memory

Physical memory

Physical memory
Program Writer:

```c
#include "world.h"
struct shared { int a, b; }

main()
{
    struct shared *p;
    methersetup(); /* Initialize the Mether run-time */
    p = (struct shared *)METHERBASE;
    /* overlay structure on METHER segment */
    p->a = p->b = 0; /* initialize fields to zero */
    while(TRUE){ /* continuously update structure fields */
        p->a = p->a + 1;
        p->b = p->b - 1;
    }
}
```
Mether system program (cont’d)

Program Reader:

```c
main()
{
    struct shared *p;
    methersetup();
    p = (struct shared *)METHERBASE;
    while(TRUE){ /* read the fields once every second */
        printf("a = %d, b = %d\n", p->a, p->b);
        sleep(1);
    }
}
```
Figure 16.3  Two processes accessing shared variables

Process 1

\[ \begin{align*}
  br & := b; \\
  ar & := a, \\
  \text{if}(ar \geq br) \text{ then} & \text{print ("OK");}
\end{align*} \]

Process 2

\[ \begin{align*}
  a & := a + 1; \\
  b & := b + 1;
\end{align*} \]
**Figure 16.4 Interleaving under sequential consistency**

Process 1

\[ br := b; \]
\[ ar := a; \]
\[ \text{if}(ar \geq br) \text{ then} \]
\[ \text{print } ("\text{OK}"); \]

Process 2

\[ a := a + 1; \]
\[ b := b + 1; \]
Figure 16.5 DSM using write-update

\[
\begin{align*}
& a := 7; \\
& b := 7; \\
& \text{if}(b = 8) \text{ then } \\
& \quad \text{print("after");} \\
& \text{if}(a = 7) \text{ then } \\
& \quad b := b + 1; \\
& \text{if}(b = a) \text{ then } \\
& \quad \text{print("before");}
\end{align*}
\]
Figure 16.6   Data items laid out over pages

![Diagram showing data items A and B laid out over pages n and n+1](image)
Figure 16.7  System model for page-based DSM

- Process accessing paged DSM segment
- Kernel redirects page faults to user-level handler
- Pages transferred over network
Figure 16.8 State transitions under write-invalidation

Single writer

\( P_W \) writes; none read

\( W \) (invalidation)

Multiple reader

\( P_{R1}, P_{R2}, \ldots P_{Rn} \) read; none write

\( W \) (invalidation)

\( R \)

Note: \( R = \) read fault occurs; \( W = \) write fault occurs.
Figure 16.9  Central manager and associated messages

1. page no., access (R/W)

2. requestor, page no., access

3. Page

Manager

Page no.  Owner
........  ........
Figure 16.10  Updating *probOwner* pointers (cont’d on next slide)

(a) *probOwner* pointers just before process *A* takes a page fault for a page owned by *E*

(b) Write fault: *probOwner* pointers after *A*’s write request is forwarded
Figure 16.10 (cont’d) Updating *probOwner* pointers

(c) Read fault: *probOwner* pointers after A’s read request is forwarded
Figure 16.11 Timeline for performing a DSM read or write operation

P issues o

o performed with respect to P' at time t

o performed (complete)

Real time
Figure 16.12  Processes executing on a release-consistent DSM

*Process 1:*

```plaintext
acquireLock();  // enter critical section
a := a + 1;
b := b + 1;
releaseLock();  // leave critical section
```

*Process 2:*

```plaintext
acquireLock();  // enter critical section
print ("The values of a and b are: ", a, b);
releaseLock();  // leave critical section
```