Figure 16.1 The distributed shared memory abstraction

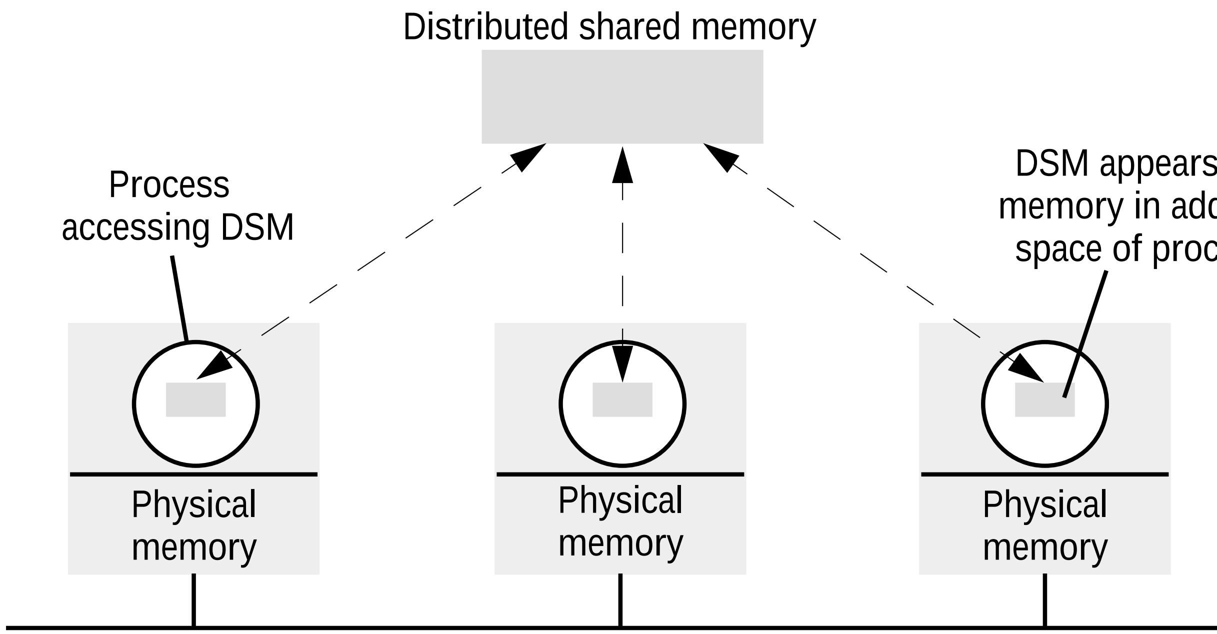


Figure 16.2 Mether system program (cont'd on next slide)

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

Program *Writer*:

```
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*:

```
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

```
br := b;
ar := a;
if(ar ≥ br) then
      print ("OK");
```

Process 2

```
a := a + 1;
b := b + 1;
```

Figure 16.4 Interleaving under sequential consistency

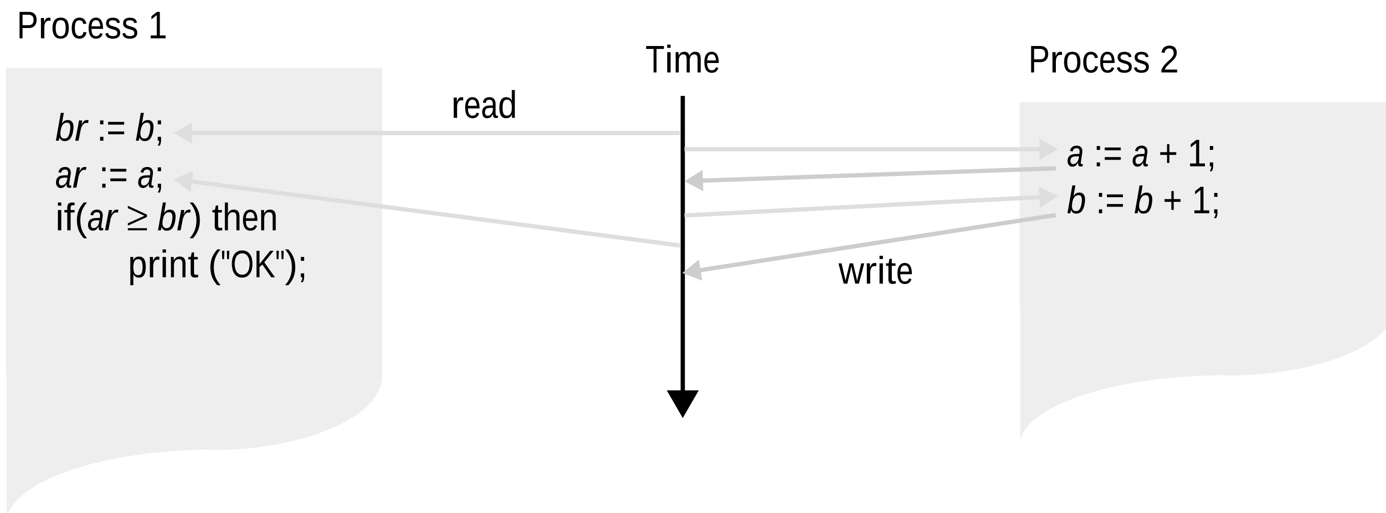


Figure 16.5 DSM using write-update

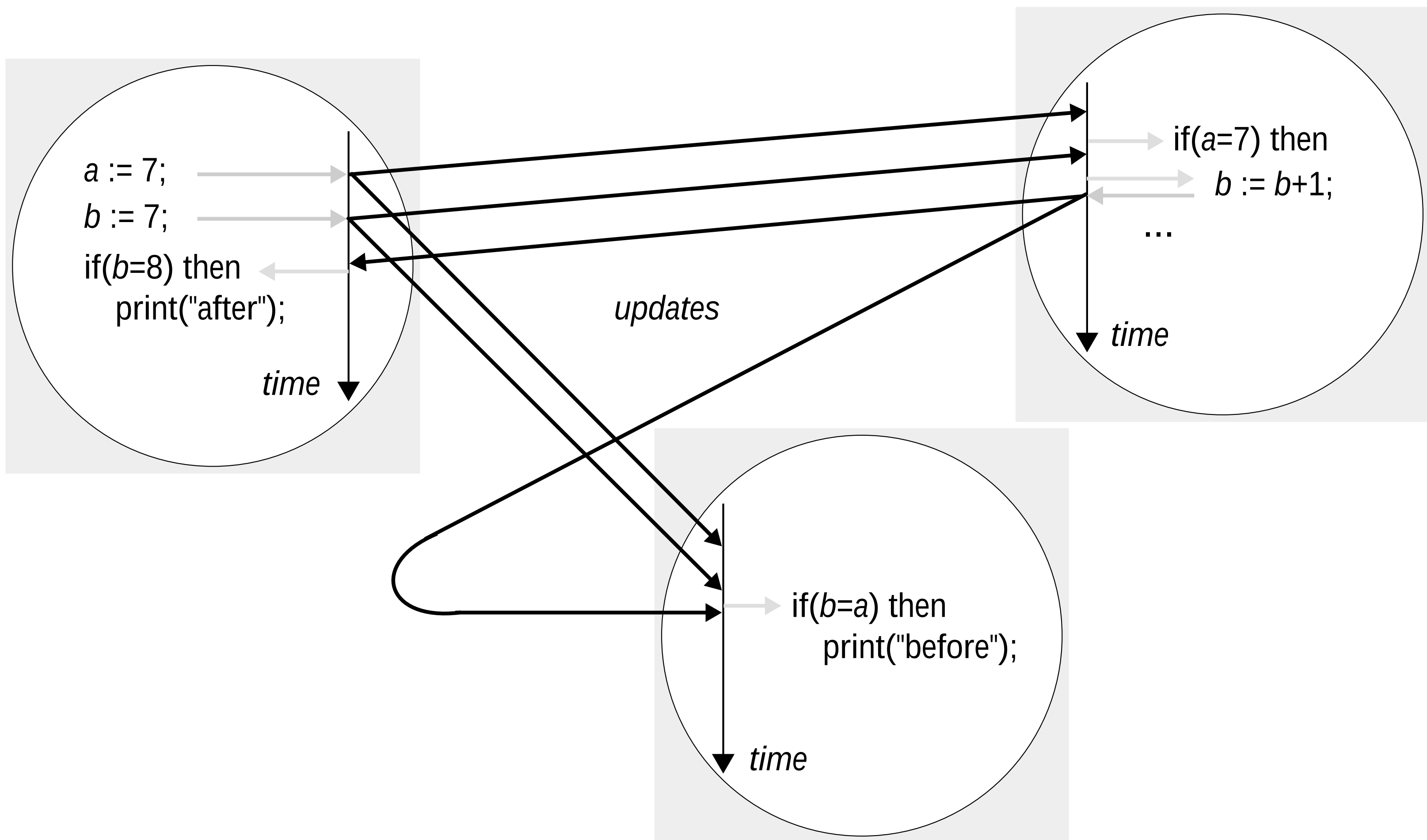


Figure 16.6 Data items laid out over pages

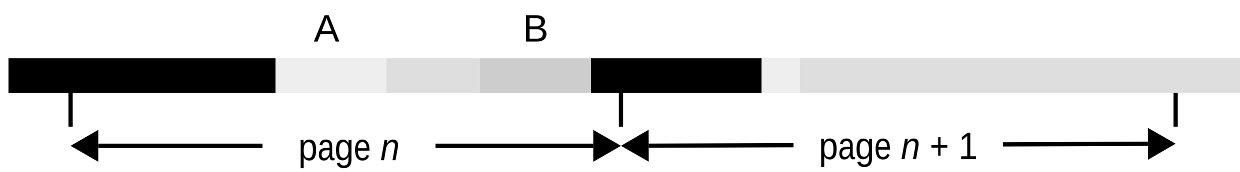


Figure 16.7 System model for page-based DSM

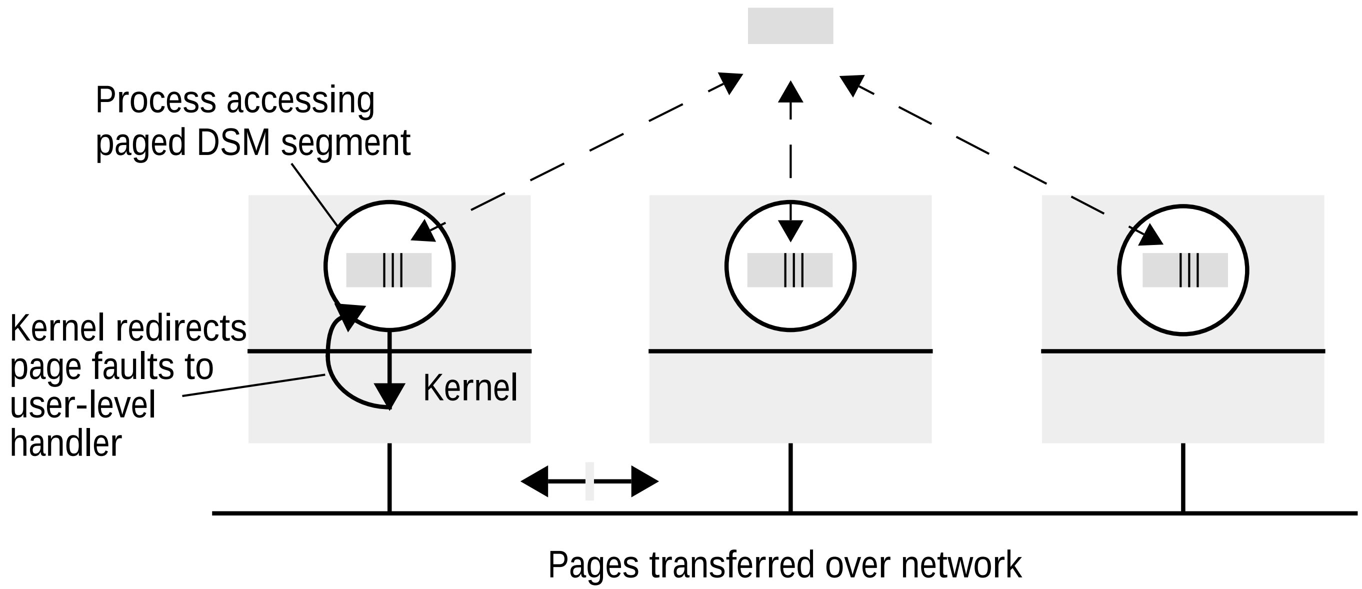
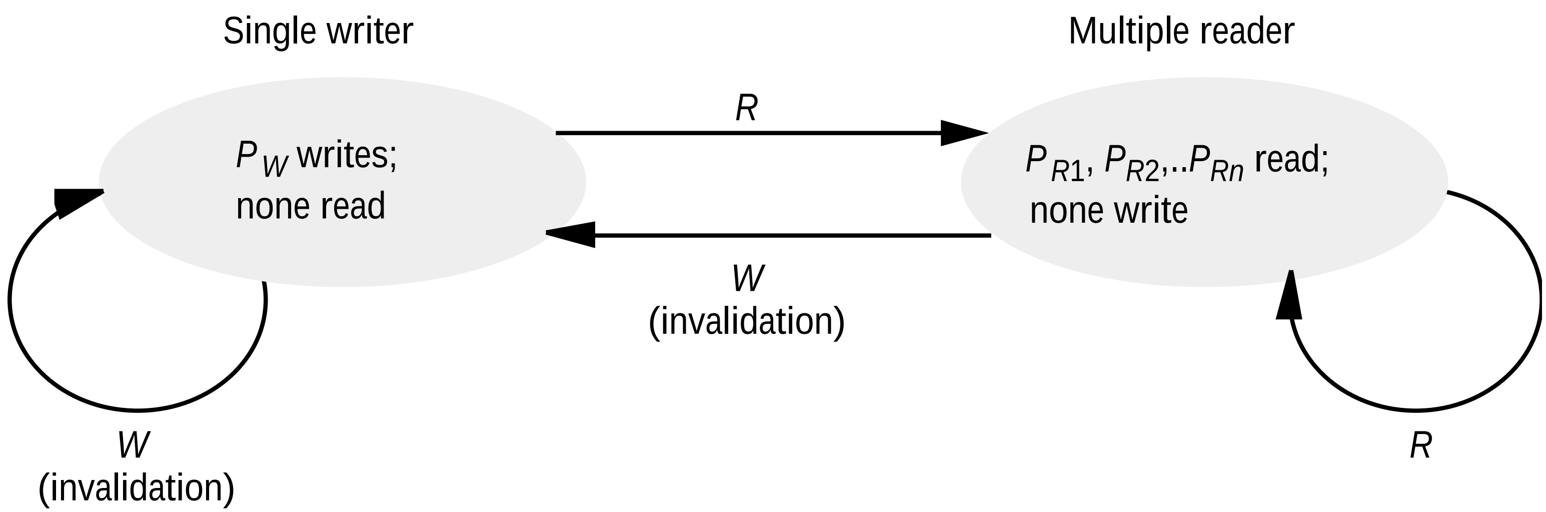


Figure 16.8 State transitions under write-invalidation



Note: R = read fault occurs; *W* = write fault occurs.

Figure 16.9 Central manager and associated messages

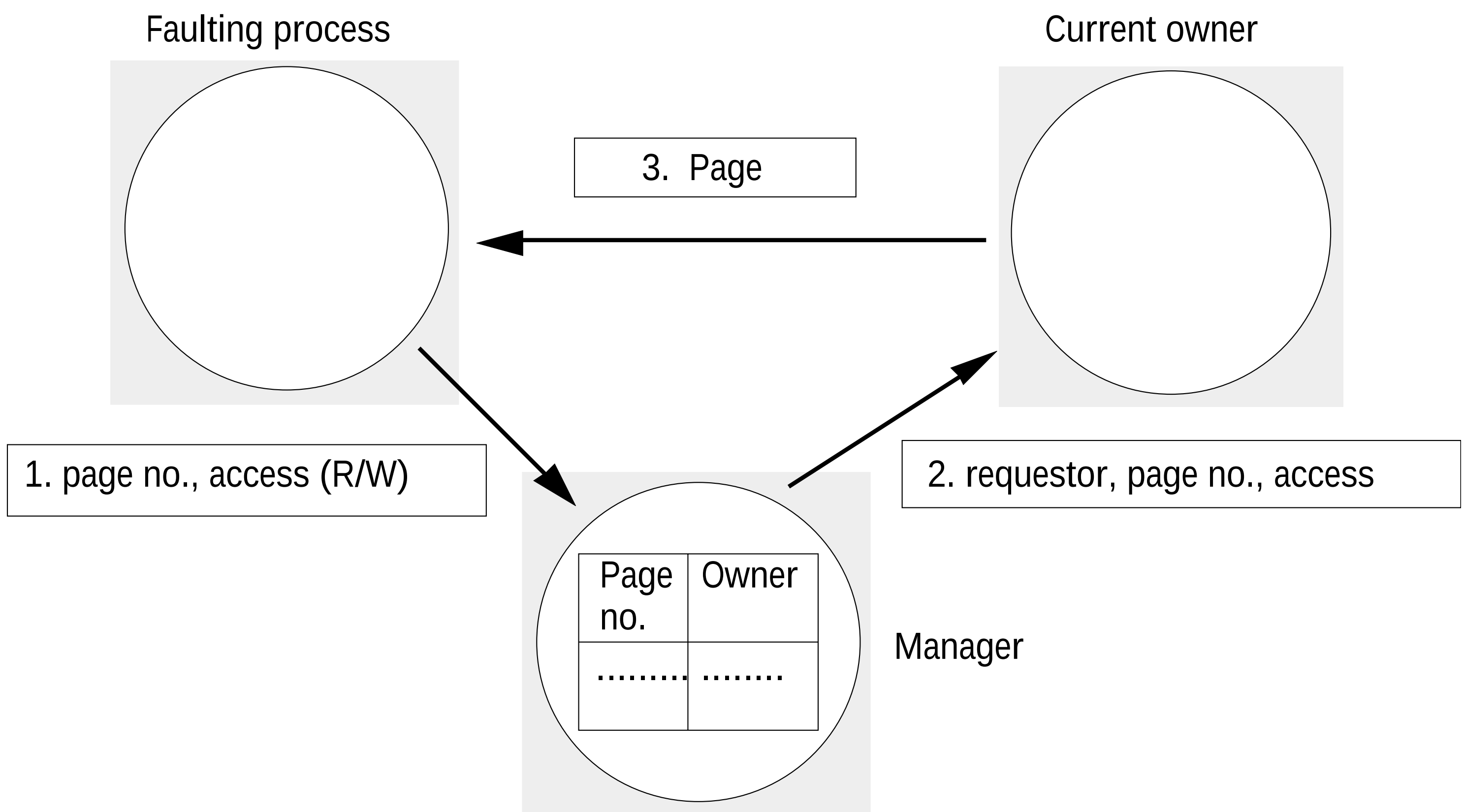
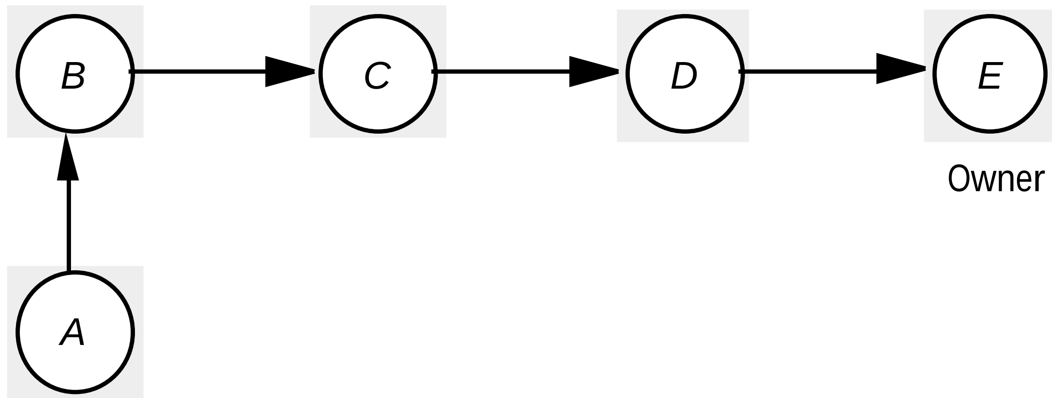
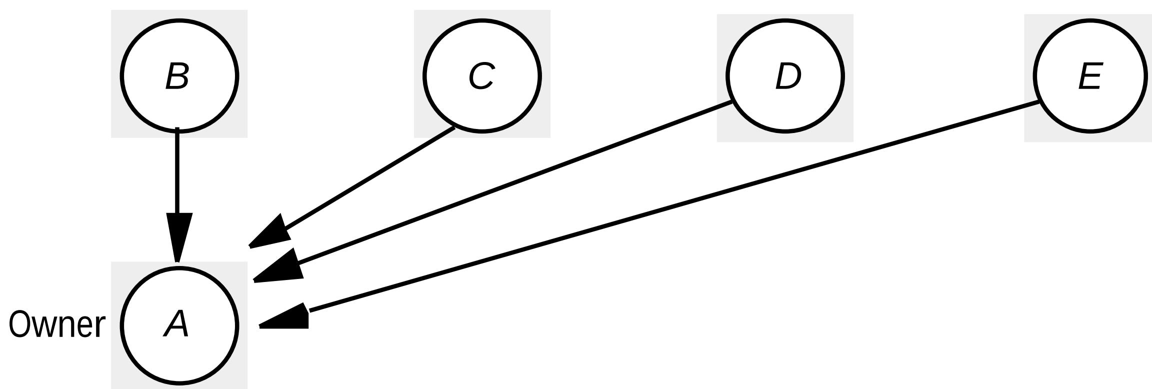


Figure 16.10 Updating *probOwner* pointers (cont'd on next slide)

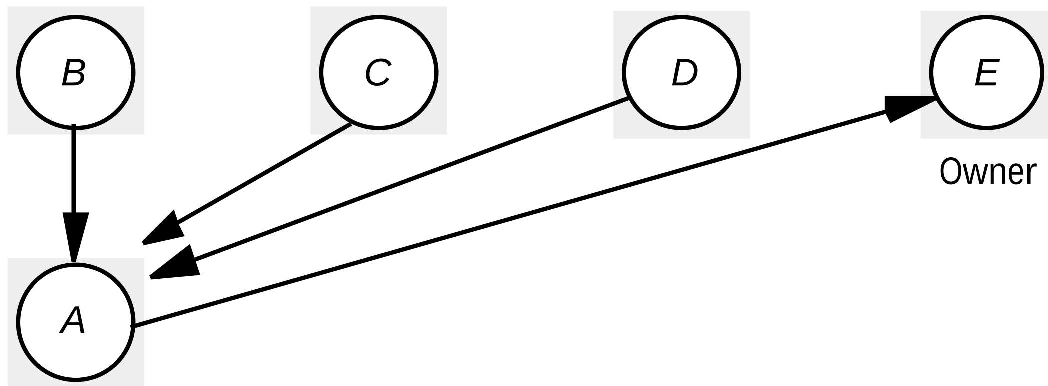


(a) *probOwner* pointers just before process *A* takes a page fault for a page owned by *E*



(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

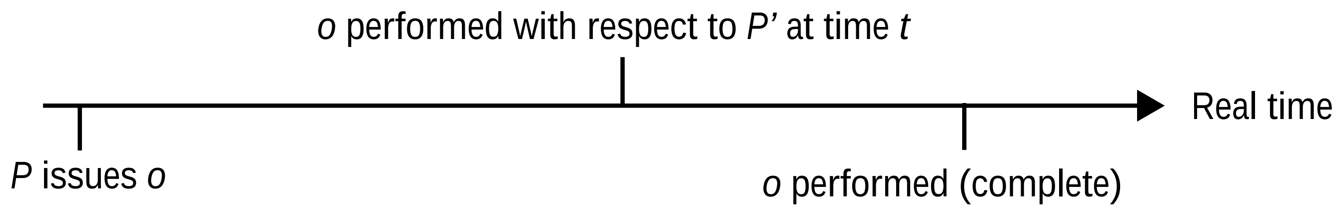


Figure 16.12 Processes executing on a release-consistent DSM

Process 1:

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

Process 2:

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