READERS-WRITERS

Module Number 2. Section 13
COP4600 – Operating Systems
Richard Newman

CLASSIC SYNCHRONIZATION PROBLEMS

Critical Section Problem: Mutual exclusion – only one process can be in Critical Region at a time.

Producer-Consumer Problem: Producers produce items and write them into buffers; Consumers remove items from buffers and consume them. A buffer can only be accessed by one process at a time.

Bounded Buffer Problem: P-C with finite # buffers.

Dining Philosophers: Philosophers arranged in a circle share a fork between each pair of neighbors. Both forks are needed for a philosopher to eat, and only one philosopher can use a fork at a time.

Readers-Writers: Any number of readers can read from a DB simultaneously, but writers must access it alone

THE READERS AND WRITERS PROBLEM

Situation arises often in accessing shared information bases.

Read-write and write-write conflicts cause race conditions
Must avoid them!
Reads do NOT conflict with other reads, so concurrent reads are allowed.

In a nutshell: Readers can access database at same time,
but writers must have exclusive access to it.

Hence, there are distinct types of locks: *read locks* and *write locks*.
Read locks are compatible with read locks, but not write locks
Write locks are not compatible with read or write locks.

Policy issues surround whether readers can enter if a writer is waiting,
and who gets in next when a writer leaves.

THE READERS AND WRITERS SPECIFICATION

Two types of process: Readers and Writers

There may be many of each type, and they run in parallel.

Readers read but do not change the contents of a database.

Writers may read and will change the contents of the database.

Concurrency: (**C1**) Multiple readers may read the database at the same time;

(**C2**) No assumptions may be made about the speed of any process.

Safety: It is NOT allowed for (**S1**) Two Writers to write to the DB at the same time; or

(**S2**) A Reader to read from the DB at the same time that a Writer is writing to it.

Liveness: (**L1**) No process that is not currently using the DB must prevent another process from accessing the DB;

(**L2**) No process must wait indefinitely to access the DB

LOCK COMPATIBLITY MATRIX

Lock type requested	Lock type held		
	None	Read	Write
Read	Yes	Yes	No
Write	Yes	No	No

Lock Compatibility Matrix hows how lock decisions should be made
May be done with lock manager
or by the synchronization code itself

THE READERS AND WRITERS PROBLEM (1)

```
typedef int semaphore;              /* use your imagination */
semaphore mutex = 1;                /* controls access to 'rc' */
semaphore db = 1;                   /* controls access to the database */
int rc = 0;                         /* # of processes reading or wanting to */

void reader(void)
{
      while (TRUE) {                /* repeat forever */
            down(&mutex);           /* get exclusive access to 'rc' */
            rc = rc + 1;            /* one reader more now */
            if (rc == 1) down(&db); /* if this is the first reader ... */
            up(&mutex);             /* release exclusive access to 'rc' */
            read_data_base();       /* access the data */
            down(&mutex);           /* get exclusive access to 'rc' */
            rc = rc − 1;            /* one reader fewer now */
            if (rc == 0) up(&db);   /* if this is the last reader ... */
            up(&mutex);             /* release exclusive access to 'rc' */
            use_data_read();        /* noncritical region */
      }
}


void writer(void)
```

Figure 2-48. A solution to the readers and writers problem.

THE READERS AND WRITERS PROBLEM (2)

```
        use_data_read();            /* noncritical region */
    }
}

void writer(void)
{
    while (TRUE) {                  /* repeat forever */
        think_up_data();            /* noncritical region */
        down(&db);                  /* get exclusive access */
        write_data_base();          /* update the data */
        up(&db);                    /* release exclusive access */
    }
}
```

Figure 2-48. A solution to the readers and writers problem.

READERS AND WRITERS POLICIES

Possible policies for R/W problem		Reader in DB and Writer waiting	
		Readers Enter	Readers Wait
		Reader Preference	Writer Preference
Writer Leaving DB, Readers and Writers Waiting	Reader Enters	Strong Reader Preference	Makes no Sense
	??? Enters	Weak Reader Preference	Fair policies possible
	Writer Enters	Weaker Reader Preference	Writer Preference

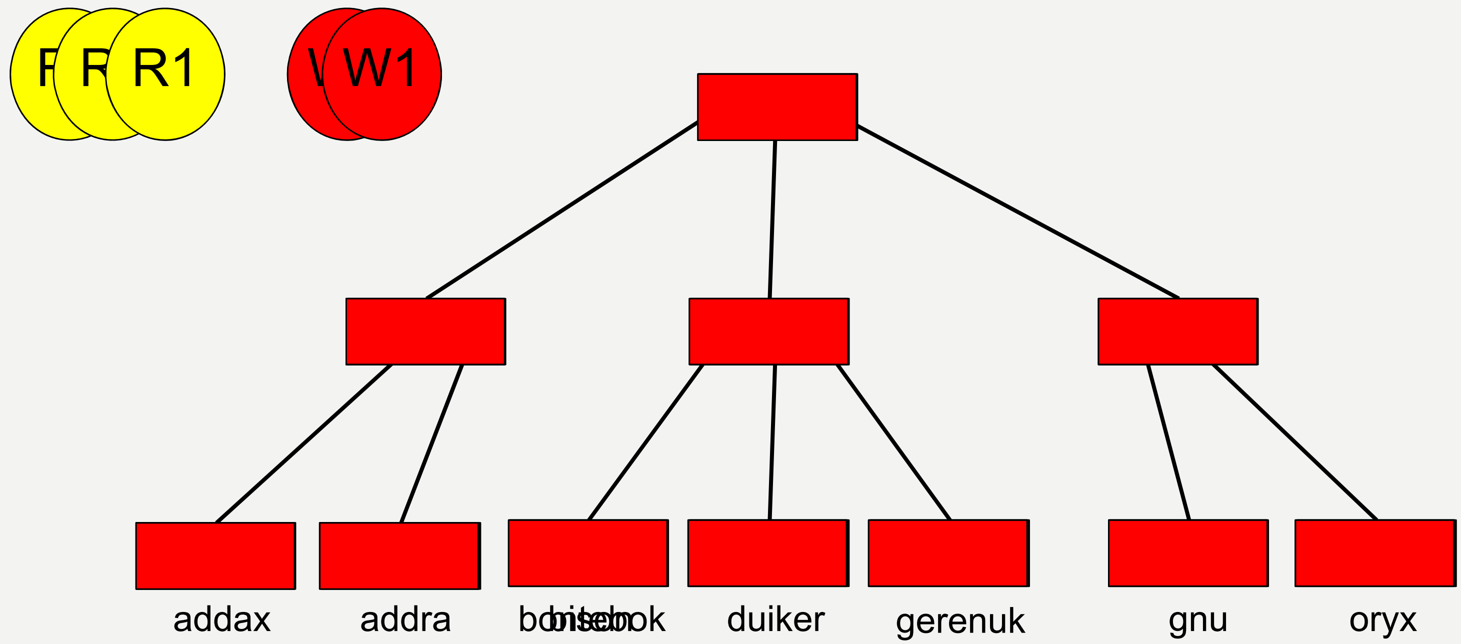
TENTATIVE WRITE LOCKS

Alpha-lock: *tentative write lock* used to increase concurrency for larger search structures

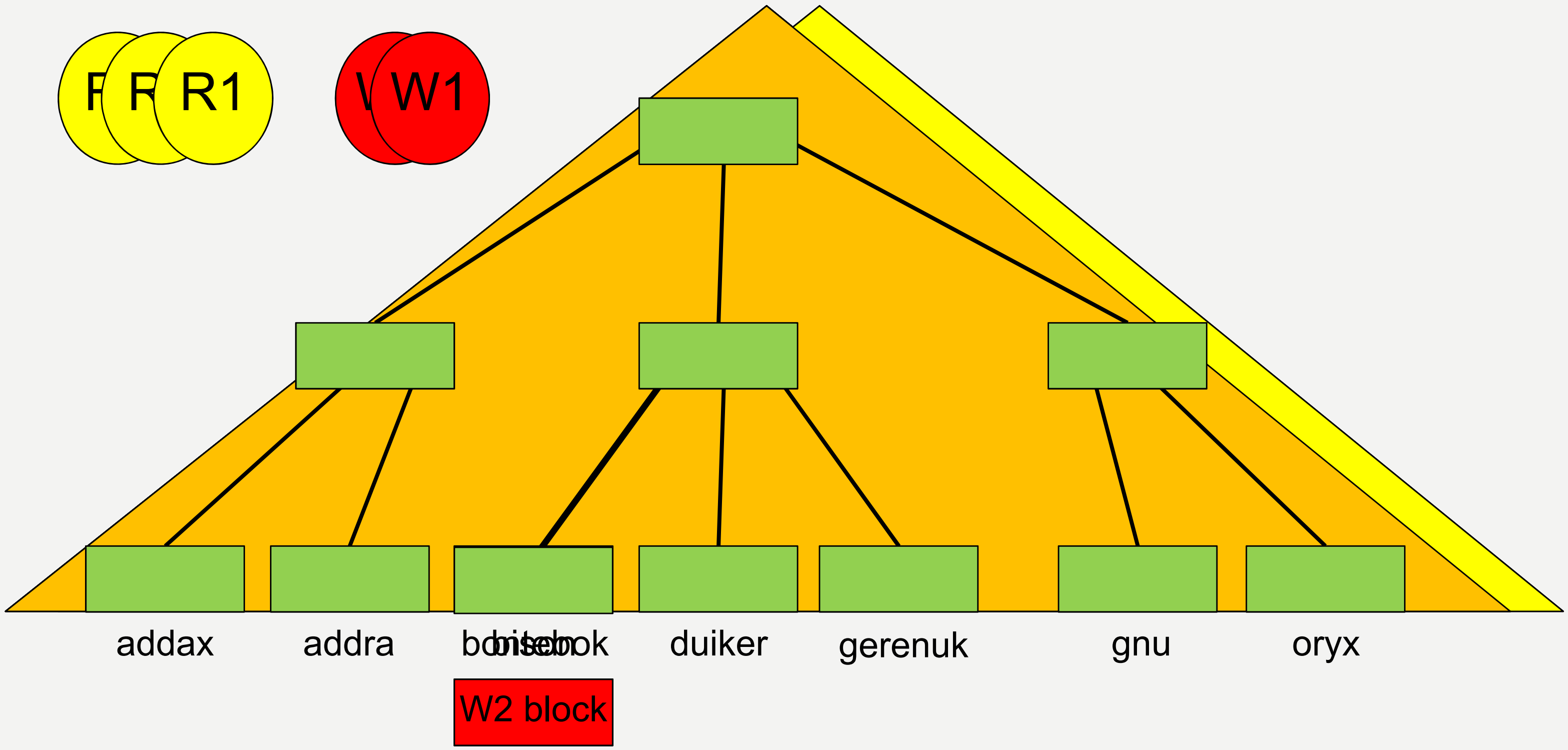
	Lock type held			
Lock type requested	None	Read	Alpha	Write
Read	Yes	Yes	Yes	No
Alpha	Yes	Yes	No	No
Write	Yes	No	No	No

Tentative Write Lock Compatibility Matrix

LOCKING WHOLE DB



LOCKING DB USING TW LOCKS



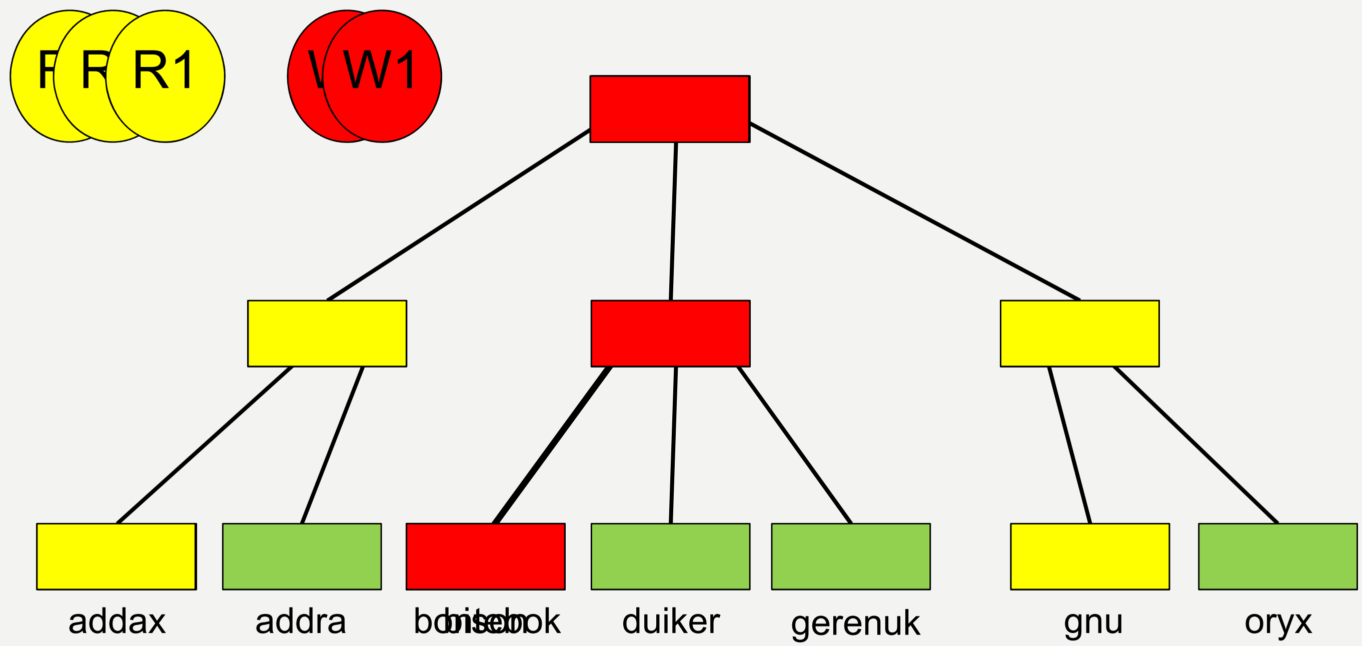
TENTATIVE WRITE LOCKS

- Tentative write locks on whole DB
  - Allow readers to read while tentative writer determines whether it needs to write
  - Tentative writer gets “dibs” on DB ahead of other writers
- Can do better on Structured Database

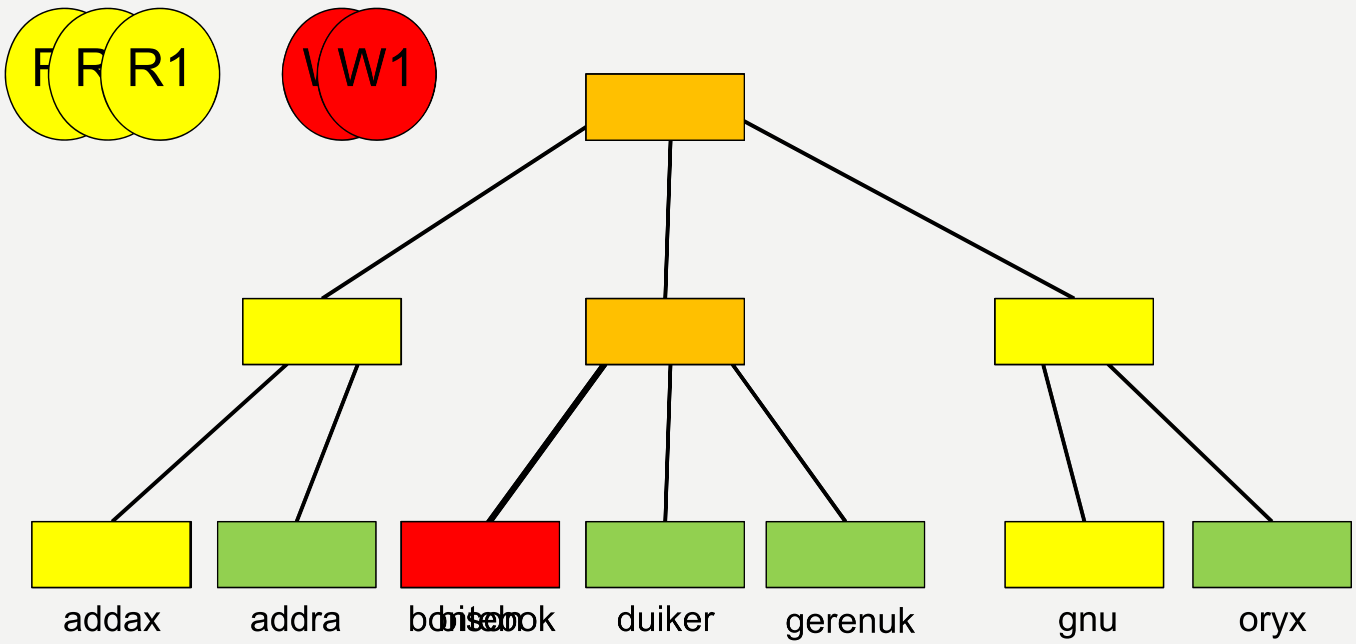
FINER-GRAINED USE OF LOCKS

- Structured Database
  - Tree, B-Tree, or graph structures
  - Writer needs to find *where* update *may* occur
  - Writer places tentative write locks in those places
  - This allows readers to continue to read
  - Writer removes these if write not needed
  - This allows other writers to start their searches
  - Writer promotes alpha lock to write lock where write needed
- Permits much higher concurrency
  - Even if plain read locks and write locks are used!

FINER GRAINED LOCKING WITH STANDARD LOCKS



FINER GRAINED TENTATIVE WRITE LOCKING



AVOIDING LOCKS: READ-COPY-UPDATE (1)

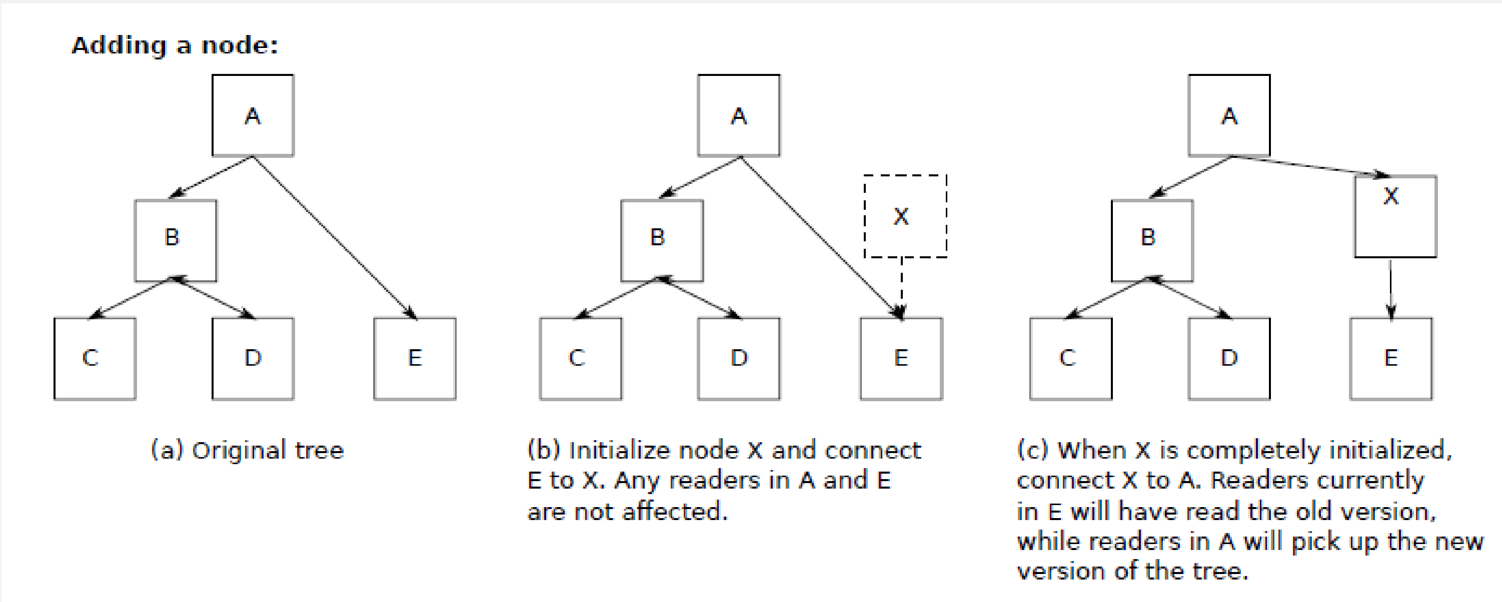


Figure 2-38. Read-Copy-Update: inserting a node in the tree and then removing a branch—all without locks

AVOIDING LOCKS: READ-COPY-UPDATE (2)

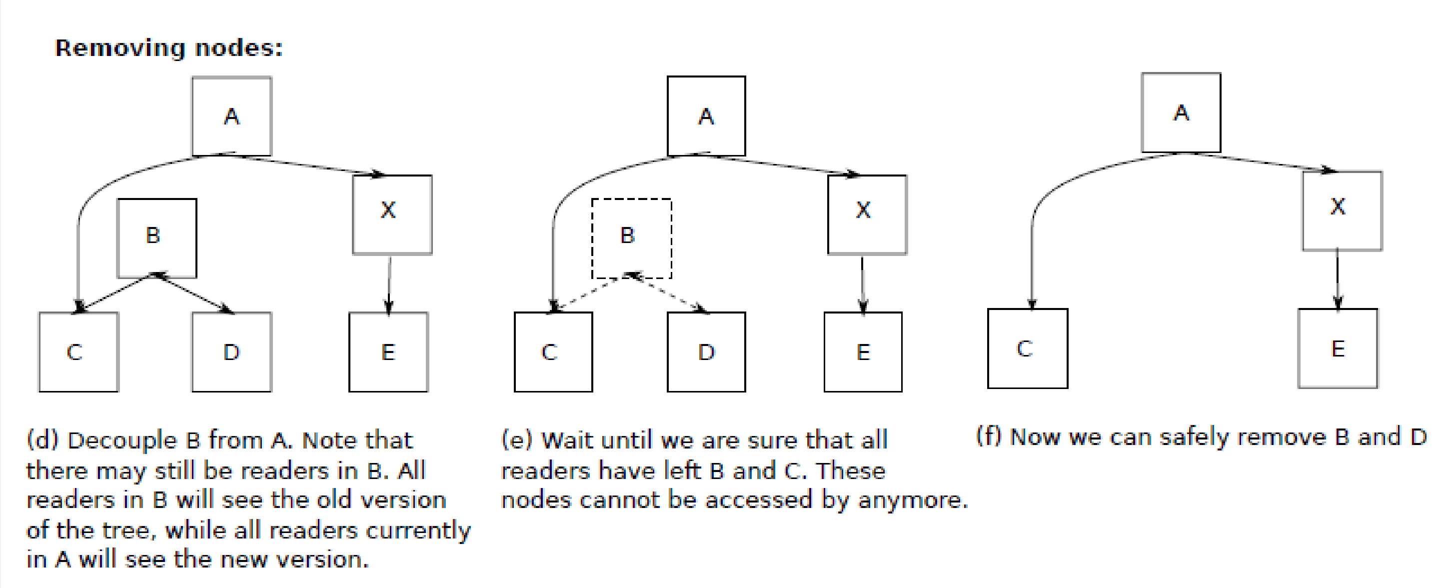


Figure 2-38. Read-Copy-Update: inserting a node in the tree and then removing a branch—all without locks

SUMMARY OF READERS-WRITERS

- Specification
  - Safety, Liveness, Concurrency
- Simple solution
  - Lock compatibility matrix
- Variety of policies
  - Effects on liveness
- Tentative write locks
- Shadow copies

SUMMARY OF CLASSIC SYNCH PROBLEMS

Critical Section Problem

- Symmetric – simplest problem – mutual exclusion

Producer-Consumer Problem

- Asymmetric – wait on event from other type of process – also have mutual exclusion for access to buffers, control vars

Dining Philosophers Problem

- Symmetric – must wait for access to two resources – five is smallest interesting number of philosophers

Reader-Writers Problem

- Asymmetric – readers can share with each other, but not with writers, writers must have exclusive access – want maximum concurrency – many possible policies