Readers/Writers Problem

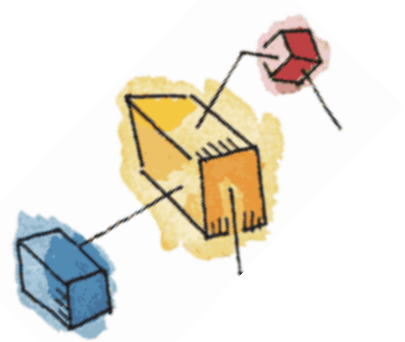
- A data area is shared among many processes
  - Some processes only read the data area, some only write to the area

- Conditions to satisfy:
  1. Multiple readers may read the file at once.
  2. Only one writer at a time may write
  3. If a writer is writing to the file, no reader may read it.

Readers have Priority

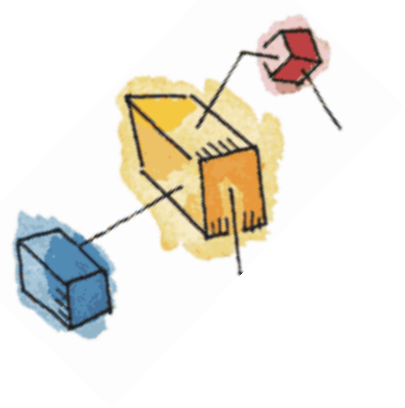
- The writer process is simple. The semaphore wsem is used to enforce mutual exclusion.
- As long as one writer is accessing the shared data area, no other writers and no readers may access it.

- The reader process also makes use of wsem to enforce mutual exclusion.
- However, to allow multiple readers, we require that, when there are no readers reading, the first reader that attempts to read should wait on wsem.



Readers have Priority

- When there is already at least one reader reading, subsequent readers need not wait before entering.
- The global variable readcount is used to keep track of the number of readers.
- The semaphore x is used to assure that readcount is updated properly.



Readers have Priority

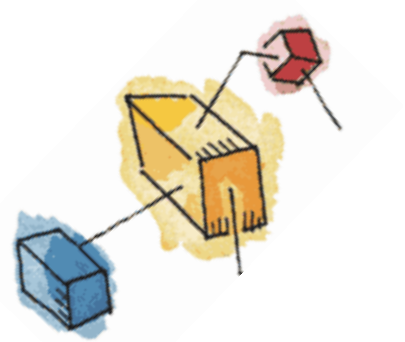
```
/* program readersandwriters */
int readcount;
semaphore x = 1, wsem = 1;
void reader()
{
    while (true) {
      semWait (x);
      readcount++;
      if (readcount == 1) semWait (wsem);
      semSignal (x);
      READUNIT();
      semWait (x);
      readcount--;
      if (readcount == 0) semSignal (wsem);
      semSignal (x);
    }
 }
void writer()
{
    while (true) {
      semWait (wsem);
      WRITEUNIT();
      semSignal (wsem);
    }
}

void main()
{
    readcount = 0;
    parbegin (reader, writer);
}
```

Writers have Priority

- This solution guarantees that no new readers are allowed access to the data area once at least one writer has declared a desire to write.

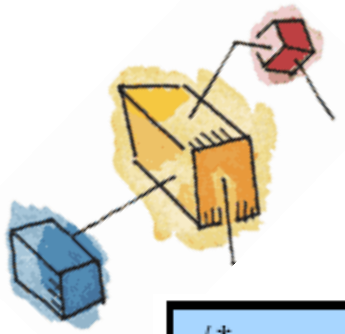
- For writers, the following semaphores and variables are added to the ones already defined:

  – A semaphore rsem that inhibits all readers while there is at least one writer desiring access to the data area
  – A variable writecount that controls the setting of rsem
  – A semaphore y that controls the updating of writecount



Writers have Priority

- For readers, one additional semaphore z is needed.
- A long queue must not be allowed to build up on rsem ; otherwise writers will not be able to jump the queue.
- Therefore, only one reader is allowed to queue on rsem, with any additional readers queuing on semaphore z



Writers have Priority

```
/* program readersandwriters */
int  readcount, writecount;
semaphore x = 1, y = 1, z = 1, wsem = 1, rsem = 1;
void reader()
{
    while (true) {
      semWait (z);
           semWait (rsem);
                semWait (x);
                     readcount++;
                     if (readcount == 1) semWait (wsem);
                semSignal (x);
           semSignal (rsem);
      semSignal (z);
      READUNIT();
      semWait (x);
           readcount--;
           if (readcount == 0) semSignal (wsem);
      semSignal (x);
    }
}
```

Writers have Priority

```
void writer ()
{
    while (true) {
      semWait (y);
           writecount++;
           if (writecount == 1) semWait (rsem);
      semSignal (y);
      semWait (wsem);
      WRITEUNIT();
      semSignal (wsem);
      semWait (y);
           writecount--;
           if (writecount == 0) semSignal (rsem);
      semSignal (y);
    }
}
void main()
{
    readcount = writecount = 0;
    parbegin (reader, writer);
}
```

Writers have Priority

Readers only in the system	• *wsem* set • no queues
Writers only in the system	• *wsem* and *rsem* set • writers queue on *wsem*
Both readers and writers with read first	• *wsem* set by reader • *rsem* set by writer • all writers queue on *wsem* • one reader queues on *rsem* • other readers queue on *z*
Both readers and writers with write first	• *wsem* set by writer • *rsem* set by writer • writers queue on *wsem* • one reader queues on *rsem* • other readers queue on *z*