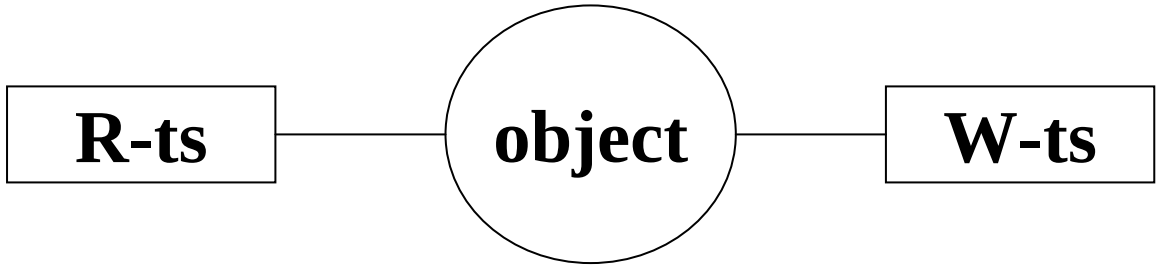
Timestamps in Locking Protocols

- **Timestamps:**
  - used to avoid deadlock.
  - each transaction has a single timestamp.
  - timestamps are used to resolve conflicts between transactions.
- **Possible Actions:**
  - wait: defer until conflicting transaction completes/aborts
  - restart:

    die - begin again but with original timestamp

    wound - attempt to cause the conflicting transaction to die and continue when the conflicting transaction completes / aborts
- **Two algorithms:**
  - wait-die: non-preemptive; a transaction finding a conflict waits if it is older and dies if it is younger.
  - wound-wait: preemptive; a transaction finding a conflict wounds if it is older and waits if it is younger

Basic Timestamp Ordering (BTO)

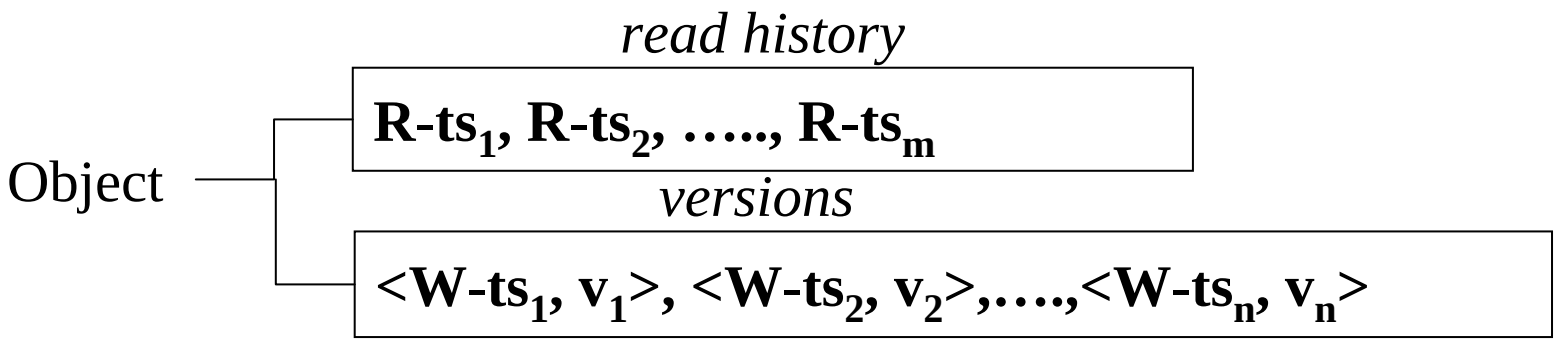


read <object,TS>
    if TS<W-ts
        then reject/abort
        else R-ts = max{R-ts,TS}

write<object, val, TS>
    if TS <R-ts or TS <W-ts
        then reject/abort
        else W-ts = TS

Thomas Write Rule: do not abort conflicting writes, simply ignore them.

Multiversion Timestamp Ordering



read <object,TS>
    read v_j where j = max{i|W-ts_i<TS}
    add <TS> to read history

write<object, val, TS>
    if (there is a k such that
        TS<R-ts_k<W-ts_j where j = min {i|TS<W-ts_i})
    then
        reject operation
    else
        add <TS, vl> to versions

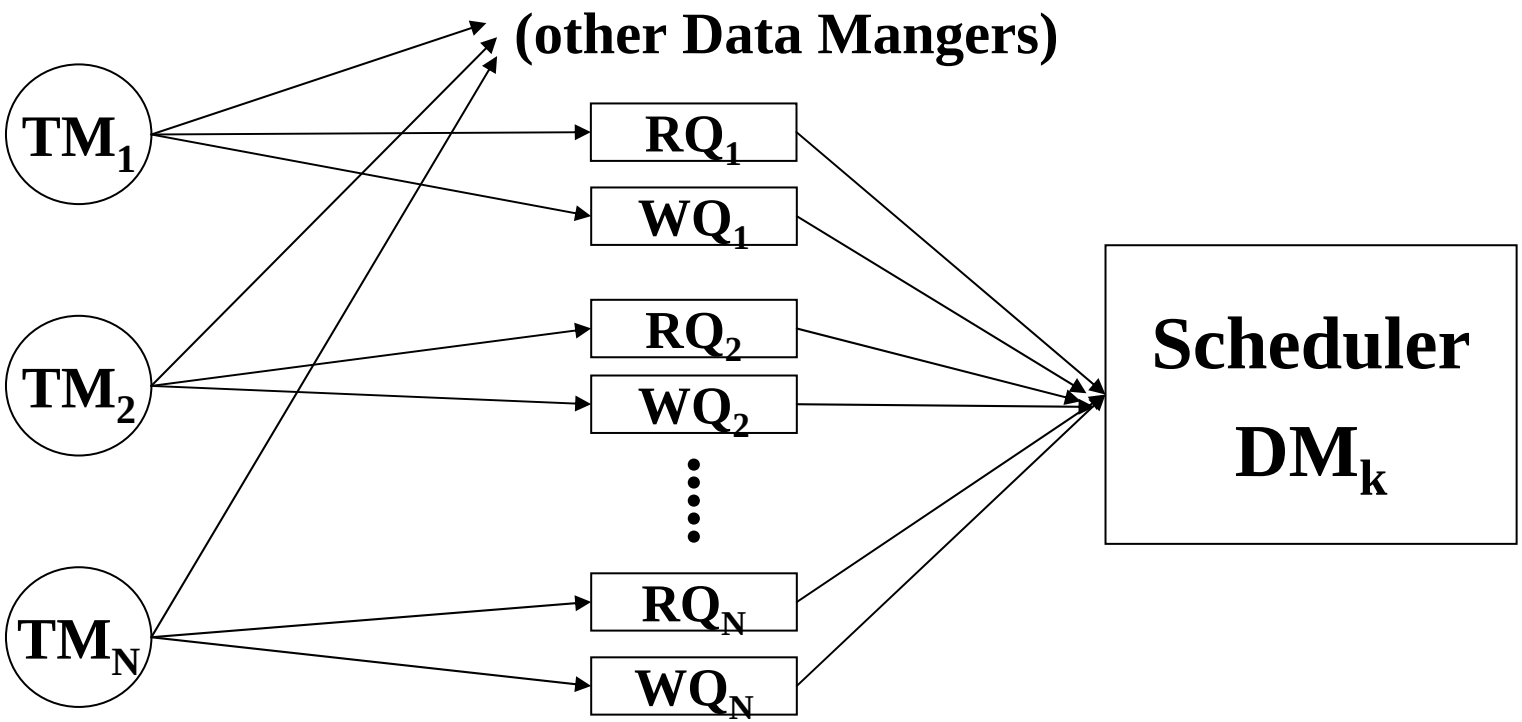
Conservative Timestamp Ordering

Each Data Manager maintains:

- a read queue (RQ_i)
- a write queue (WQ_i)

for each Transaction Manger, TM_i

Let: $TS(Q_i)$ denote the timestamp of the first operation in Q_i



Conservative Timestamp Ordering

Let: $TS(Q_i)$ denote the timestamp of the first operation in Q_i

read <object,TS>

if (non-empty(WQ_i) and $TS(WQ_i) > TS$ for i = 1 ….N)
then execute the read operation
else add the read operation to RQ_i

write<object, val, TS>

if (non-empty (RQ_i) and non-empty (WQ_i) and
$TS(RQ_i) > TS$ and $TS(WQ_i) > TS$ for i = 1….N)
then execute the write operation
else add the write operation to WQ_i

Optimistic Algorithms (Kung-Robinson)

Each transaction, T, has three phases:

- read phase

  read from database and write to temporary storage (log)

- validation phase

  If (T does not conflict with any other executing transaction)

  then

    assign the transaction a unique (monotonically increasing) sequence number and perform the write phase

  else abort T

- write phase

  write log to database

Optimistic Algorithms (Kung-Robinson)

Let:

t_s be the highest sequence number at the start of T

t_f be the highest sequence number at the beginning of T's validation phase

validation algorithm:

```
valid = true;

for t = t_s + 1 to t_f do
   if (writeset[t] intersect readset[T] != ϕ)
         then valid = false;

   if (valid)
   then
         do write phase;
         increment counter;
         assign T a sequence number;
```