MODULE *ZeusOwnershipMeta*

EXTENDS *Integers*, *FiniteSets*

CONSTANTS *LB_NODES* and *APP_NODES* must not intersect and neither should contain 0.
LB_NODES,
APP_NODES,
O_MAX_VERSION,
O_MAX_FAILURES,
O_MAX_DATA_VERSION

VARIABLES variable prefixes – > *o*: ownership, *r*: request, t: transactional, *m*: membership
VECTORS indexed by *node_id*
oTS,
oState,
oDriver,
oVector, No readers/owner: $.readers = \{\}$ / $.owner = 0$
oRcvACKs,

rTS,
rID,
rType,
rEID, since we do not have message loss timeouts we use this to
track epoch of last issued *INVs* for replays
tState,
tVersion, *tVesion* sufice to represent $tData| = 0 - >$ no data $| > 0$ data (reader / owner)
tRcvACKs,
GLOBAL variables
oMsgs,
mAliveNodes, membership
mEID, membership epoch *id*
committedREQs, only to check invariant that exactly one of concurrent *REQs* is committed
committedRTS only to emulate *FIFO REQ* channels (*i.e.*, do not re execute same client requests)

$vars \triangleq \langle oTS, oState, oDriver, oVector, oRcvACKs, rTS, rID, rType, rEID,$
$tState, tVersion, tRcvACKs, oMsgs, mAliveNodes, mEID, committedREQs, committedRTS\rangle$

Helper operators
$O\_NODES \triangleq LB\_NODES \cup APP\_NODES$
$O\_NODES\_0 \triangleq O\_NODES \cup \{0\}$
$LB\_NODES\_0 \triangleq LB\_NODES \cup \{0\}$
$APP\_NODES\_0 \triangleq APP\_NODES \cup \{0\}$
$LB\_LIVE\_NODES \triangleq LB\_NODES \cap mAliveNodes$
$APP\_LIVE\_NODES \triangleq APP\_NODES \cap mAliveNodes$
$LB\_LIVE\_ARBITERS(driver) \triangleq LB\_LIVE\_NODES \setminus \{driver\}$ all arbiters except driver and owner

ASSUME $LB\_NODES \cap APP\_NODES = \{\}$
ASSUME $\forall\, k \in O\_NODES : k \neq 0$ we use 0 as the default noop

---

Useful Unchanged shortcuts

$unchanged\_M \triangleq$ UNCHANGED $\langle oMsgs \rangle$
$unchanged\_m \triangleq$ UNCHANGED $\langle mEID,\ mAliveNodes \rangle$
$unchanged\_t \triangleq$ UNCHANGED $\langle tState,\ \ tVersion,\ tRcvACKs \rangle$
$unchanged\_r \triangleq$ UNCHANGED $\langle rID,\ rTS,\ rEID,\ rType \rangle$
$unchanged\_c \triangleq$ UNCHANGED $\langle committedREQs,\ committedRTS \rangle$
$unchanged\_o \triangleq$ UNCHANGED $\langle oState,\ oDriver,\ oVector,\ oRcvACKs,\ oTS \rangle$
$unchanged\_mc \triangleq unchanged\_m \land unchanged\_c$
$unchanged\_mtc \triangleq unchanged\_mc \land unchanged\_t$
$unchanged\_mtr \triangleq unchanged\_m \land unchanged\_t \land unchanged\_r$
$unchanged\_Mrc \triangleq unchanged\_r \land unchanged\_c \land unchanged\_M$
$unchanged\_mrco \triangleq unchanged\_mc \land unchanged\_r \land unchanged\_o$
$unchanged\_mtco \triangleq unchanged\_mtc \land unchanged\_o$
$unchanged\_mtrc \triangleq unchanged\_mtc \land unchanged\_r$
$unchanged\_Mtrc \triangleq unchanged\_Mrc \land unchanged\_t$
$unchanged\_Mmrc \triangleq unchanged\_Mrc \land unchanged\_m$
$unchanged\_mtrco \triangleq unchanged\_mtrc \land unchanged\_o$
$unchanged\_Mmrco \triangleq unchanged\_mrco \land unchanged\_M$
$unchanged\_Mmtrc \triangleq unchanged\_mtrc \land unchanged\_M$

Type definitions

$Type\_oTS \triangleq [ver : 0\,..\,O\_MAX\_VERSION,\ tb : LB\_NODES\_0]$
$Type\_rTS \triangleq [ver : 0\,..\,O\_MAX\_VERSION,\ tb : APP\_NODES\_0]$
$Type\_tState \triangleq \{$"valid", "invalid", "write"$\}$ readers can be in valid and invalid and owner in valid and write
$Type\_oState \triangleq \{$"valid", "invalid", "drive", "request"$\}$ all nodes start from valid
$Type\_rType \triangleq \{$"add-owner", "change-owner", "add-reader", "rm-reader", "NOOP"$\}$
$Type\_oVector \triangleq [readers :$ SUBSET $APP\_NODES,\ owner : APP\_NODES\_0]$

$Type\_oMessage \triangleq$ Msgs exchanged by the sharding protocol

$[type : \{$"REQ"$\},\ rTS : Type\_rTS,\ rID : Nat,\ rType : Type\_rType,\ epochID : 0\,..\,O\_MAX\_FAILURES]$

$\cup$

$[type : \{$"NACK"$\},\ rTS : Type\_rTS,\ rID : Nat]$

$\cup$

$[type : \{$"S_INV"$\},\ sender : O\_NODES,\ driver : O\_NODES,\ rTS : Type\_rTS,$

$$
\begin{array}{lll}
 & rID & : Nat, \\
 & oTS & : Type\_oTS, \\
 & oVector & : Type\_oVector, \\
 & rType & : Type\_rType, \\
 & epochID & : 0 \,..\, O\_MAX\_FAILURES] \\
\cup & & \\
[type : \{\text{“S\_ACK”}\}, & sender & : O\_NODES, \\
 & oTS & : Type\_oTS, \\
 & tVersion & : 0 \,..\, O\_MAX\_DATA\_VERSION, \quad \text{emulates data send as well} \\
 & epochID & : 0 \,..\, O\_MAX\_FAILURES] \\
\cup & & \\
[type : \{\text{“RESP”}\}, & oVector & : Type\_oVector, \\
 & oTS & : Type\_oTS, \\
 & rTS & : Type\_rTS, \\
\multicolumn{3}{l}{preOwner \text{ , } \backslash * \text{ pre-request owner is not needed for model check (since we model bcast messages)}} \\
 & tVersion & : 0 \,..\, O\_MAX\_DATA\_VERSION, \\
 & epochID & : 0 \,..\, O\_MAX\_FAILURES] \\
\cup & & \\
[type : \{\text{“S\_VAL”}\}, & oTS & : Type\_oTS, \\
 & epochID & : 0 \,..\, O\_MAX\_FAILURES]
\end{array}
$$

$$
\begin{array}{lll}
Type\_tMessage \triangleq & \text{msgs exchanged by the transactional reliable commit protocol} & \\
[type : \{\text{“T\_INV”}, \text{“T\_ACK”}, \text{“T\_VAL”}\}, & tVersion & : Nat, \\
 & sender & : O\_NODES, \\
 & epochID & : 0 \,..\, O\_MAX\_FAILURES]
\end{array}
$$

Type check and initialization

$$
\begin{array}{lll}
OTypeOK \triangleq & \text{The type correctness invariant} & \\
\quad \wedge\, oTS & \in [O\_NODES & \to Type\_oTS] \\
\quad \wedge\, oState & \in [O\_NODES & \to Type\_oState] \\
\quad \wedge\, oDriver & \in [O\_NODES & \to O\_NODES\_0] \\
\quad \wedge\, oVector & \in [O\_NODES & \to Type\_oVector] \\
\quad \wedge\, \forall\, n \in O\_NODES : oRcvACKs[n] & \subseteq (O\_NODES \setminus \{n\}) & \\
\quad \wedge\, rTS & \in [O\_NODES & \to Type\_rTS] \\
\quad \wedge\, rID & \in [O\_NODES & \to 0 \,..\, O\_MAX\_VERSION] \\
\quad \wedge\, rType & \in [O\_NODES & \to Type\_rType] \\
\quad \wedge\, rEID & \in [O\_NODES & \to 0 \,..\, (Cardinality(O\_NODES) - 1)] \\
\quad \wedge\, tVersion & \in [O\_NODES & \to 0 \,..\, O\_MAX\_DATA\_VERSION] \\
\quad \wedge\, tState & \in [O\_NODES & \to Type\_tState] \\
\quad \wedge\, \forall\, n \in O\_NODES : tRcvACKs[n] & \subseteq (O\_NODES \setminus \{n\}) & \\
\quad \wedge\, committedREQs & \subseteq Type\_oTS & \\
\quad \wedge\, committedRTS & \subseteq Type\_rTS &
\end{array}
$$

$\land\ oMsgs \quad \subseteq (Type\_oMessage \cup Type\_tMessage)$
$\land\ mEID \quad \in 0\,..\,(Cardinality(O\_NODES) - 1)$
$\land\ mAliveNodes \quad \subseteq O\_NODES$

$OInit \triangleq$ The initial predicate
$\land\ oTS = [n \in O\_NODES \mapsto [ver \mapsto 0,\ tb \mapsto 0]]$
$\land\ oState = [n \in O\_NODES \mapsto$ "valid"$]$
$\land\ oDriver = [n \in O\_NODES \mapsto 0]$
$\land\ oVector = [n \in O\_NODES \mapsto [readers \mapsto \{\},\ owner \mapsto 0]]$
$\land\ oRcvACKs = [n \in O\_NODES \mapsto \{\}]$
$\land\ rTS = [n \in O\_NODES \mapsto [ver \mapsto 0,\ tb \mapsto 0]]$
$\land\ rID = [n \in O\_NODES \mapsto 0]$
$\land\ rEID = [n \in O\_NODES \mapsto 0]$
$\land\ rType = [n \in O\_NODES \mapsto$ "NOOP"$]$
$\land\ tVersion = [n \in O\_NODES \mapsto 0]$
$\land\ tState = [n \in O\_NODES \mapsto$ "valid"$]$
$\land\ tRcvACKs = [n \in O\_NODES \mapsto \{\}]$
$\land\ committedRTS = \{\}$
$\land\ committedREQs = \{\}$
$\land\ oMsgs = \{\}$
$\land\ mEID = 0$
$\land\ mAliveNodes = O\_NODES$

$Min(S) \triangleq$ CHOOSE $x \in S : \forall\, y \in S \setminus \{x\} : y > x$
$set\_wo\_min(S) \triangleq S \setminus \{Min(S)\}$

First Command executed once after *OInit* to initialize owner/readers and *oVector* state
$OInit\_min\_owner\_rest\_readers \triangleq$
$\land\ \forall\, x \in O\_NODES : tVersion[x] = 0$
$\land\ tVersion' = [n \in O\_NODES \mapsto$ IF $n \in LB\_NODES$ THEN 0 ELSE 1$]$
$\land\ oVector' = [n \in O\_NODES \mapsto$ IF $n \in set\_wo\_min(APP\_NODES)$
THEN $oVector[n]$
ELSE $[readers \mapsto set\_wo\_min(APP\_NODES),$
$owner \mapsto Min(APP\_NODES)]]$
$\land\ unchanged\_Mmrc$
$\land$ UNCHANGED $\langle tState,\ oState,\ oDriver,\ oRcvACKs,\ oTS,\ tRcvACKs\rangle$

Helper functions
$has\_data(n) \triangleq tVersion[n] > 0$
$has\_valid\_data(n) \triangleq \land\ has\_data(n)$
$\land\ tState[n] =$ "valid"

$is\_owner(n) \triangleq \land\ has\_data(n)$
$\land\ oVector[n].owner = n$

$is\_valid\_owner(n) \triangleq \land\ is\_owner(n)$

$$\wedge\ oState[n] = \text{“valid”}$$

$is\_reader(n) \triangleq \begin{array}{l} \wedge\ has\_data(n) \\ \wedge\ \neg is\_owner(n) \\ \wedge\ n \notin LB\_NODES \end{array}$

$is\_live\_arbiter(n) \triangleq \begin{array}{l} \vee\ n \in LB\_LIVE\_NODES \\ \vee\ is\_owner(n) \end{array}$

$is\_valid\_live\_arbiter(n) \triangleq \begin{array}{l} \wedge\ is\_live\_arbiter(n) \\ \wedge\ oState[n] = \text{“valid”} \end{array}$

$is\_requester(n) \triangleq$
$\quad \wedge\ n \in APP\_LIVE\_NODES$
$\quad \wedge\ \neg is\_owner(n)$

$is\_valid\_requester(n) \triangleq$
$\quad \wedge\ is\_requester(n)$
$\quad \wedge\ oState[n] = \text{“valid”}$

$is\_in\_progress\_requester(n) \triangleq$
$\quad \wedge\ is\_requester(n)$
$\quad \wedge\ oState[n] = \text{“request”}$

$requester\_is\_alive(n) \triangleq rTS[n].tb \in mAliveNodes$

Timestamp Comparison Helper functions

$is\_equalTS(ts1,\ ts2) \triangleq$
$\quad \wedge\ ts1.ver = ts2.ver$
$\quad \wedge\ ts1.tb\ \ = ts2.tb$

$is\_greaterTS(ts1,\ ts2) \triangleq$
$\quad \vee\ ts1.ver > ts2.ver$
$\quad \vee\ \wedge\ \ ts1.ver = ts2.ver$
$\qquad \wedge\ \ ts1.tb\ \ > ts2.tb$

$is\_greatereqTS(ts1,\ ts2) \triangleq$
$\quad \vee\ is\_equalTS(ts1,\ ts2)$
$\quad \vee\ is\_greaterTS(ts1,\ ts2)$

$is\_smallerTS(ts1,\ ts2) \triangleq \neg is\_greatereqTS(ts1,\ ts2)$

Request type Helper functions

$is\_non\_sharing\_req(n) \triangleq (rType[n] = \text{“add-owner”} \vee rType[n] = \text{“add-reader”})$

Post *o_vector* based on request type and r (requester or 0 if requester is not alive)

$post\_oVec(n,\ r,\ pre\_oVec) \triangleq$

$$
\begin{array}{l}
\text{IF } (rType[n] = \text{“add-owner”} \lor rType[n] = \text{“change-owner”}) \\
\text{THEN } [owner \mapsto r, \\
\quad readers \mapsto (pre\_oVec.readers \cup \{pre\_oVec.owner\}) \setminus \{r, 0\}] \\
\text{ELSE } [owner \mapsto pre\_oVec.owner, \\
\quad readers \mapsto \text{IF } rType[n] = \text{“remove-reader”} \\
\qquad\qquad \text{THEN } pre\_oVec.readers \setminus \{r, 0\} \\
\qquad\qquad \text{ELSE } rType[n] = \text{“add-reader”} \\
\qquad\qquad\quad (pre\_oVec.readers \cup \{r\}) \setminus \{0\}]
\end{array}
$$

Message Helper functions

Used only to emulate *FIFO REQ* channels (and not re-execute already completed *REQs*)

$$not\_completed\_rTS(r\_ts) \triangleq \forall\, c\_rTS \in committedRTS : c\_rTS \neq r\_ts$$

Messages in *oMsgs* are only appended to this variable (not removed once delivered) intentionally to check protocols tolerance in dublicates and reorderings

$$send\_omsg(m) \triangleq oMsgs' = oMsgs \cup \{m\}$$

$$
\begin{array}{l}
o\_send\_req(r\_ts, r\_id, r\_type) \triangleq \\
\quad send\_omsg([type \mapsto \text{“REQ”}, \\
\qquad\qquad rTS \mapsto r\_ts, \\
\qquad\qquad rID \mapsto r\_id, \\
\qquad\qquad rType \mapsto r\_type, \\
\qquad\qquad epochID \mapsto mEID])
\end{array}
$$

$$
\begin{array}{l}
o\_send\_nack(r\_ts, r\_id) \triangleq \\
\quad send\_omsg([type \mapsto \text{“NACK”}, \\
\qquad\qquad rTS \mapsto r\_ts, \\
\qquad\qquad rID \mapsto r\_id])
\end{array}
$$

$$
\begin{array}{l}
o\_send\_inv(sender, driver, o\_ts, o\_vec, r\_ts, r\_id, r\_type) \triangleq \\
\quad send\_omsg([type \mapsto \text{“S\_INV”}, \\
\qquad\qquad sender \mapsto sender, \\
\qquad\qquad driver \mapsto driver, \\
\qquad\qquad oTS \mapsto o\_ts, \\
\qquad\qquad oVector \mapsto o\_vec, \\
\qquad\qquad rTS \mapsto r\_ts, \\
\qquad\qquad rID \mapsto r\_id, \\
\qquad\qquad rType \mapsto r\_type, \\
\qquad\qquad epochID \mapsto mEID])
\end{array}
$$

$$
\begin{array}{l}
o\_send\_ack(sender, o\_ts, t\_version) \triangleq \\
\quad send\_omsg([type \mapsto \text{“S\_ACK”}, \\
\qquad\qquad sender \mapsto sender, \\
\qquad\qquad oTS \mapsto o\_ts,
\end{array}
$$

$$
\begin{array}{ll}
tVersion & \mapsto t\_version, \\
epochID & \mapsto mEID])
\end{array}
$$

$o\_send\_resp(r\_ts,\ o\_ts,\ o\_vec,\ t\_version) \triangleq$

$$
\begin{array}{ll}
send\_omsg([type & \mapsto \text{“RESP”}, \\
oVector & \mapsto o\_vec, \\
oTS & \mapsto o\_ts, \\
rTS & \mapsto r\_ts, \\
tVersion & \mapsto t\_version, \\
epochID & \mapsto mEID])
\end{array}
$$

$o\_send\_val(o\_ts) \triangleq$

$$
\begin{array}{ll}
send\_omsg([type & \mapsto \text{“S\_VAL”}, \\
oTS & \mapsto o\_ts, \\
epochID & \mapsto mEID \quad])
\end{array}
$$

Operators to check received messages (m stands for message)

$o\_rcv\_req(m) \triangleq$
- $\wedge\ m.type = \text{“REQ”}$
- $\wedge\ m.epochID = mEID$
- $\wedge\ not\_completed\_rTS(m.rTS)$

$o\_rcv\_nack(m,\ receiver) \triangleq$
- $\wedge\ m.type = \text{“NACK”}$
- $\wedge\ m.rTS\ = rTS[receiver]$
- $\wedge\ m.rID\ = rID[receiver]$

$o\_rcv\_resp(m,\ receiver) \triangleq$
- $\wedge\ m.type\ = \text{“RESP”}$
- $\wedge\ m.epochID = mEID$
- $\wedge\ m.rTS\ = rTS[receiver]$

$o\_rcv\_inv(m,\ receiver) \triangleq$
- $\wedge\ m.type\ = \text{“S\_INV”}$
- $\wedge\ m.epochID\ = mEID$
- $\wedge\ m.sender\ \neq receiver$

$o\_rcv\_inv\_equal\_ts(m,\ receiver) \triangleq$
- $\wedge\ o\_rcv\_inv(m,\ receiver)$
- $\wedge\ is\_equalTS(m.oTS,\ oTS[receiver])$

$o\_rcv\_inv\_smaller\_ts(m,\ receiver) \triangleq$
- $\wedge\ o\_rcv\_inv(m,\ receiver)$
- $\wedge\ is\_smallerTS(m.oTS,\ oTS[receiver])$

$o\_rcv\_inv\_greater\_ts(m,\ receiver) \triangleq$
- $\wedge\ o\_rcv\_inv(m,\ receiver)$
- $\wedge\ is\_greaterTS(m.oTS,\ oTS[receiver])$

$o\_rcv\_inv\_greatereq\_ts(m,\ receiver) \triangleq$
$\quad \land\ o\_rcv\_inv(m,\ receiver)$
$\quad \land\ \neg is\_smallerTS(m.oTS,\ oTS[receiver])$

$o\_rcv\_ack(m,\ receiver) \triangleq$
$\quad \land\ m.type = \text{“S\_ACK”}$
$\quad \land\ m.epochID = mEID$
$\quad \land\ m.sender \neq receiver$
$\quad \land\ oState[receiver] = \text{“drive”}$
$\quad \land\ m.sender \notin oRcvACKs[receiver]$
$\quad \land\ is\_equalTS(m.oTS,\ oTS[receiver])$

$o\_rcv\_val(m,\ receiver) \triangleq$
$\quad \land\ m.type = \text{“S\_VAL”}$
$\quad \land\ m.epochID = mEID$
$\quad \land\ oState[receiver] \neq \text{“valid”}$
$\quad \land\ is\_equalTS(m.oTS,\ oTS[receiver])$

Used to not re-issue messages that already exists (and bound the state space)
$msg\_not\_exists(o\_rcv\_msg(\_,\ \_),\ receiver) \triangleq$
$\quad \neg\exists\, mm \in oMsgs : o\_rcv\_msg(mm,\ receiver)$

$rcved\_acks\_from\_set(n,\ set) \triangleq set \subseteq oRcvACKs[n]$

Check if all acknowledgments from arbiters have been received
$has\_rcved\_all\_ACKs(n) \triangleq$
$\quad \land\ rEID[n] = mEID$
$\quad \land\ \text{IF}\ oVector[n].owner \neq 0$
$\quad\quad \text{THEN}\ rcved\_acks\_from\_set(n,\ \{oVector[n].owner\} \cup LB\_LIVE\_ARBITERS(n))$
$\quad\quad \text{ELSE}\ \lor\ \land \neg requester\_is\_alive(n)$
$\quad\quad\quad\quad\quad\ \land\ rcved\_acks\_from\_set(n,\ LB\_LIVE\_ARBITERS(n))$
$\quad\quad\quad\quad\quad \lor\ \land\ oVector[n].readers \neq \{\}$
$\quad\quad\quad\quad\quad\quad\ \land\ \exists\, x \in oVector[n].readers : rcved\_acks\_from\_set(n,\ \{x\} \cup LB\_LIVE\_ARBITERS(n))$

---

message helper functions related to transactions
$t\_send(n,\ msg\_type,\ t\_ver) \triangleq$
$\quad send\_omsg([type \mapsto msg\_type,$
$\quad\quad\quad\quad tVersion \mapsto t\_ver,$
$\quad\quad\quad\quad sender \mapsto n,$
$\quad\quad\quad\quad epochID \mapsto mEID\ ])$

$t\_rcv\_inv(m,\ receiver) \triangleq$
$\quad \land\ m.type = \text{“T\_INV”}$
$\quad \land\ m.epochID = mEID$
$\quad \land\ m.sender \neq receiver$

$t\_rcv\_ack(m,\ receiver) \triangleq$
$\quad \land\ m.type \quad = \text{“T\_ACK”}$
$\quad \land\ m.epochID \ = mEID$
$\quad \land\ m.sender \quad \neq receiver$
$\quad \land\ tState[receiver] = \text{“write”}$
$\quad \land\ m.sender \quad \notin tRcvACKs[receiver]$
$\quad \land\ m.tVersion = tVersion[receiver]$

$t\_rcv\_val(m,\ receiver) \triangleq$
$\quad \land\ m.type = \text{“T\_VAL”}$
$\quad \land\ m.epochID \ = mEID$
$\quad \land\ tState[receiver] \neq \text{“valid”}$
$\quad \land\ m.tVersion = tVersion[receiver]$

Protocol *Invariants*:

Valid data are consistent
$CONSISTENT\_DATA \triangleq$
$\quad \forall\, k,\ n \in APP\_LIVE\_NODES :\ \lor \neg has\_valid\_data(k)$
$\qquad\qquad\qquad\qquad\qquad\qquad\ \lor \neg has\_valid\_data(n)$
$\qquad\qquad\qquad\qquad\qquad\qquad\ \lor tVersion[n] = tVersion[k]$

Amongst concurrent sharing requests only one succeeds
The invariant that an we cannot have two *REQs* committed with same versions
(*i.e.*, that read and modified the same sharing vector)
$ONLY\_ONE\_CONC\_REQ\_COMMITS \triangleq$
$\quad \forall\, x,\ y \in committedREQs :\ \lor x.ver \neq y.ver$
$\qquad\qquad\qquad\qquad\qquad\qquad\ \lor x.tb = y.tb$

There is always at most one valid owner
$AT\_MOST\_ONE\_OWNER \triangleq$
$\quad \forall\, n,\ m \in mAliveNodes :\ \lor \neg is\_valid\_owner(n)$
$\qquad\qquad\qquad\qquad\qquad\ \lor \neg is\_valid\_owner(m)$
$\qquad\qquad\qquad\qquad\qquad\ \lor m = n$

Valid owner has the most up-to-date data and version among live replicas
$OWNER\_LATEST\_DATA \triangleq$
$\quad \forall\, o,\ k \in mAliveNodes :\ \lor \neg is\_valid\_owner(o)$
$\qquad\qquad\qquad\qquad\qquad\ \lor \neg has\_data(o)$
$\qquad\qquad\qquad\qquad\qquad\ \lor tVersion[o] \geq tVersion[k]$

All valid sharers ($LB + owner$) agree on their sharing vectors (and TS)
$CONSISTENT\_SHARERS \triangleq$
$\quad \forall\, k,\ n \in mAliveNodes :\ \lor \neg is\_valid\_live\_arbiter(n)$
$\qquad\qquad\qquad\qquad\qquad\ \lor \neg is\_valid\_live\_arbiter(k)$
$\qquad\qquad\qquad\qquad\qquad\ \lor\ \land oTS[n] = oTS[k]$

$$\wedge\ oVector[n] = oVector[k]$$

$CONSISTENT\_OVECTORS\_Fwd \triangleq$
$\forall\, n \in mAliveNodes :\ \vee \neg is\_valid\_live\_arbiter(n)$
$\vee\ \wedge \forall\, r \in oVector[n].readers :$
$\wedge\ has\_data(r)$
$\wedge \neg is\_valid\_owner(r)$
$\wedge\ \vee\ oVector[n].owner = 0$
$\vee\ is\_owner(oVector[n].owner)$

$CONSISTENT\_OVECTORS\_Reverse\_owner \triangleq$
$\forall\, o,\ n \in mAliveNodes :\ \vee \neg is\_valid\_owner(o)$
$\vee \neg is\_valid\_live\_arbiter(n)$
$\vee\ oVector[n].owner = o$

$CONSISTENT\_OVECTORS\_Reverse\_readers \triangleq$
$\forall\, r,\ n \in mAliveNodes :\ \vee \neg is\_reader(r)$
$\vee \neg is\_valid\_live\_arbiter(n)$
$\vee\ r \in oVector[n].readers$

The owner and readers are always correctly reflected by any valid sharing vectors
$CONSISTENT\_OVECTORS \triangleq$
$\wedge\ CONSISTENT\_OVECTORS\_Fwd$
$\wedge\ CONSISTENT\_OVECTORS\_Reverse\_owner$
$\wedge\ CONSISTENT\_OVECTORS\_Reverse\_readers$