TÉCNICO LISBOA

Towards a dependable consensus: Read/write protocols (cont.)

Highly dependable systems – 2024/25

Lecture 5

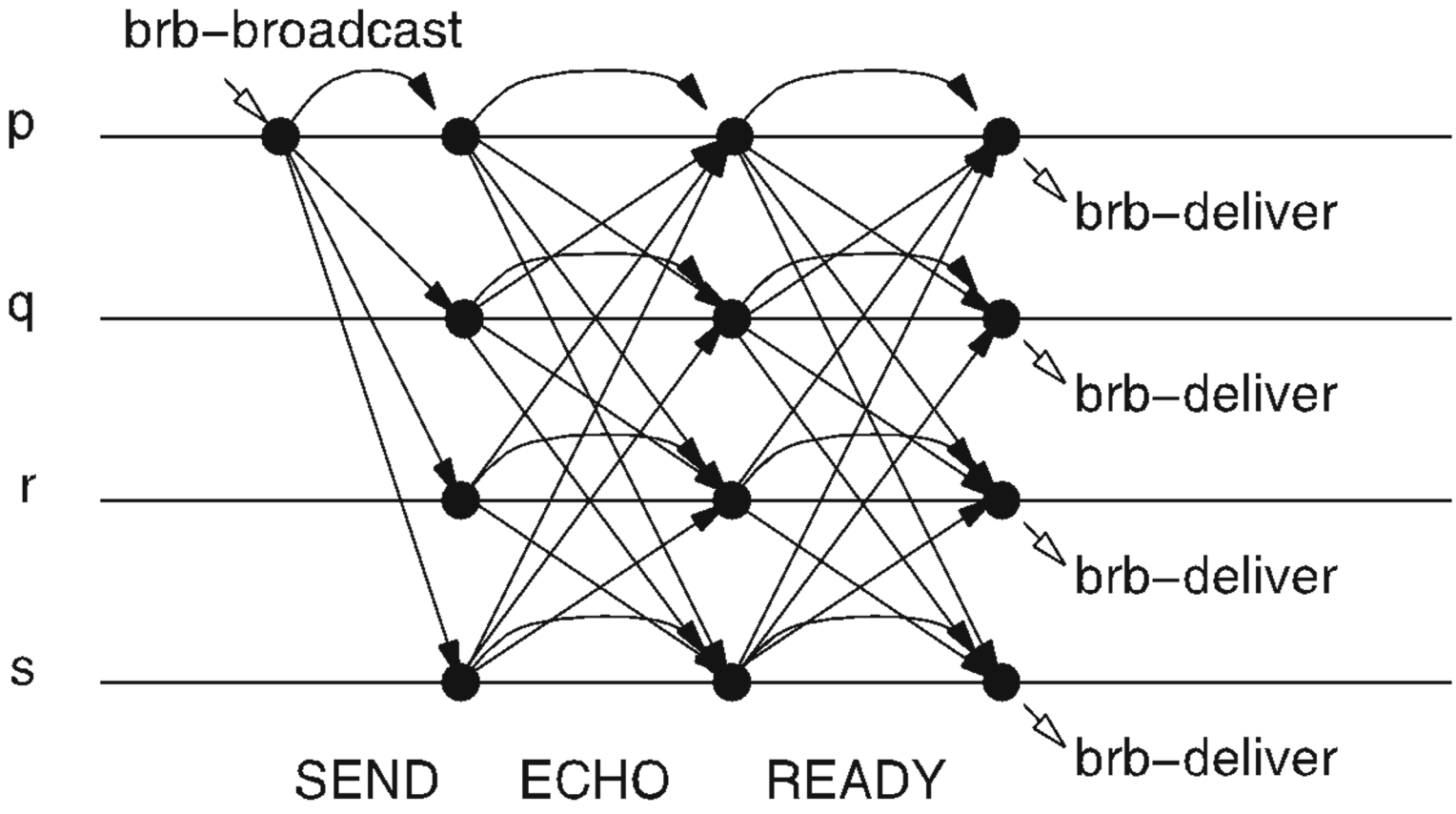
Lecturers: Miguel Matos and Paolo Romano

Last lecture: Byzantine Reliable Broadcast

- BCB1: Validity: If a correct process p broadcasts a msg m, then every correct process eventually delivers m.
- BCB2: No duplication: Every correct process delivers at most one message.
- BCB3: Integrity: If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

  BCB4: Consistency: If some correct process delivers a message m and another correct process delivers a message m′, then m = m′.
- **BRB5: *Totality:* If some message is delivered by any correct process, every correct process eventually delivers a message.**

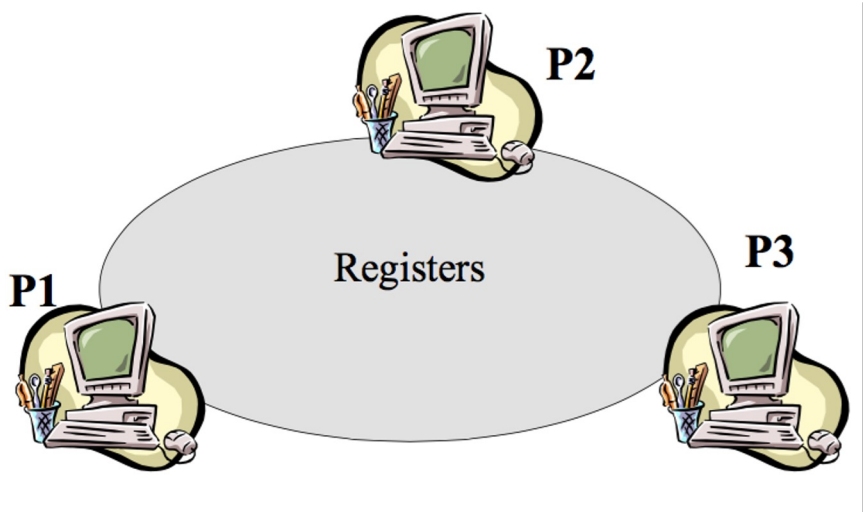
Authenticated Double-Echo Broadcast



Key proof elements

- Consistency: If some correct process delivers a message m and another correct process delivers a message m′, then m = m′.
  - if some correct process broadcasts ready messages for m and another correct process broadcasts ready for m', then m=m'
  - Proof by contradiction, same as in previous proof
  - Directly implies consistency because cannot have more than f ready messages for different messages
- Totality:
  - if a correct process delivers, then it received at least 2f+1 ready messages, at least f+1 from correct process.
  - These f+1 will eventually be delivered to all correct processes, triggering amplification step, and consequently sufficient ready messages for totality

New abstraction: read/write registers



- P1,P2,P3 are both clients and replicas (without loss of generality)
- **`Read()`** → returns **`value`**
- **`Write(value)`** → returns "**`ack`**"

Specification #1: Regular register

- Initially, we will assume a single writer
- [Liveness] If a correct process invokes an operation, then the operation eventually completes.
- [Safety] Read must return:
  - the last value written if there is no concurrent write operation, otherwise
  - either the last value or any value concurrently written
- When a process crashes in the middle of reading/writing, the operation interval does not have an upper limit

Number of writers Number of readers

Implementation of a (1,N)-Regular register

- Uses BestEffortBroadcast + Perfect-p2p-links
- Each process maintains:

  <ts,val> /* Current value and associated timestamp, ts */

  readlist /* List of returned values, for reading */

  rid /* id of current read operation */
- Writer process maintains:

  wts /* Next timestamp to be written */

  acks /* How may writes have been acknowledged */
- Algorithm must tolerate up to *f* crash faults. How?

Implements:

$(1, N)$-RegularRegister, **instance** *onrr*.

Uses:

BestEffortBroadcast, **instance** *beb*;

PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *onrr*, *Init* $\rangle$ **do**

$(ts, val) := (0, \bot)$;

$wts := 0$;

$acks := 0$;

$rid := 0$;

$readlist := [\bot]^N$;

upon event $\langle$ *onrr*, *Write* | v $\rangle$ **do**

$wts := wts + 1$;

$acks := 0$;

trigger $\langle$ *beb*, *Broadcast* | [WRITE, wts, v] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [WRITE, ts', v'] $\rangle$ **do**

if $ts' > ts$ **then**

$(ts, val) := (ts', v')$;

trigger $\langle$ *pl*, *Send* | p, [ACK, ts'] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | q, [ACK, ts'] $\rangle$ **such that** $ts' = wts$ **do**

$acks := acks + 1$;

if $acks > N/2$ **then**

$acks := 0$;

trigger $\langle$ *onrr*, *WriteReturn* $\rangle$;

upon event $\langle$ *onrr*, *Read* $\rangle$ **do**
    $rid := rid + 1$;
    $readlist := [\perp]^N$;
    trigger $\langle$ *beb*, *Broadcast* | [READ, *rid*] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [READ, r] $\rangle$ **do**
    trigger $\langle$ *pl*, *Send* | p, [VALUE, r, ts, val] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | q, [VALUE, r, ts', v'] $\rangle$ **such that** $r = rid$ **do**
    $readlist[q] := (ts', v')$;
    if $\#(readlist) > N/2$ **then**
        $v := highestval(readlist)$;
        $readlist := [\perp]^N$;
        trigger $\langle$ *onrr*, *ReadReturn* | v $\rangle$;

Correctness

- Liveness:
  - Any Read() or Write() eventually returns by the assumption of a majority (*quorum*) of correct processes and the liveness properties of the channels

Correctness

- Safety:
  - Split into two cases:
  1. In the absence of concurrent or failed operation, a Read() returns the last value written:
     - Assume a Write(x) terminates, and no other Write() is invoked. A quorum of processes have x in their local value, and this is associated with the highest timestamp in the system (because it is the last value written). Any subsequent Read() invocation by some process pj returns x, as required by the specification
  2. If writes are issued concurrently, the reader will choose the highest timestamp in the quorum, which can be either the last value in case no one in the quorum had received any of the concurrent writes, or otherwise any value concurrently written

Is this execution valid?

Yes, reads might return the last value written of the value concurrently being written

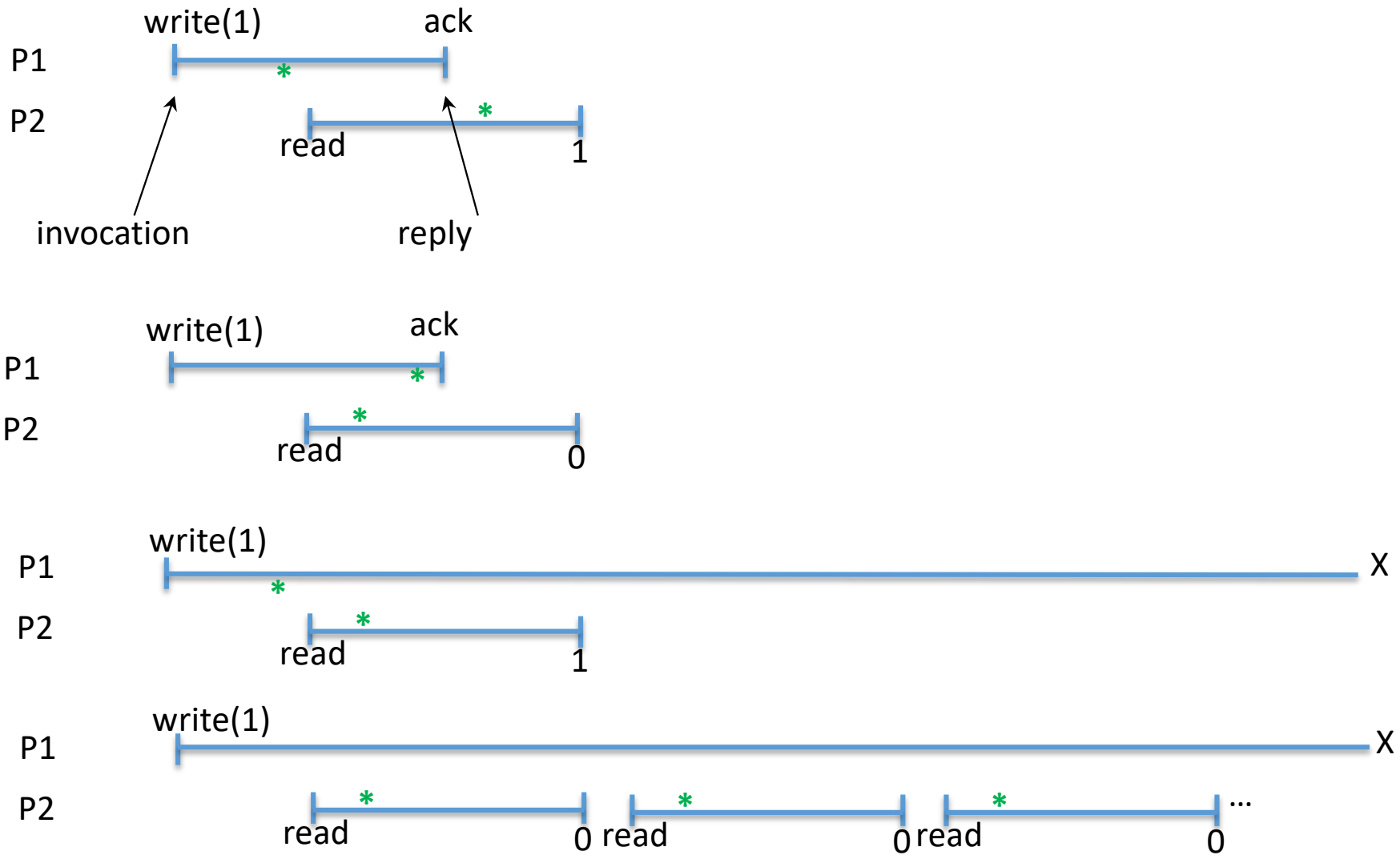
Specification #2: (1,N) Atomic register

- Additional new safety property:
- Ordering: If a read returns a value v and a subsequent read returns a value w, then the write of w does not precede the write of v:
  - write(w) can either be concurrent with or follow v
- Intuition: behavior of the replicated register is the same as that of a non-replicated register
- Also known as linearizability

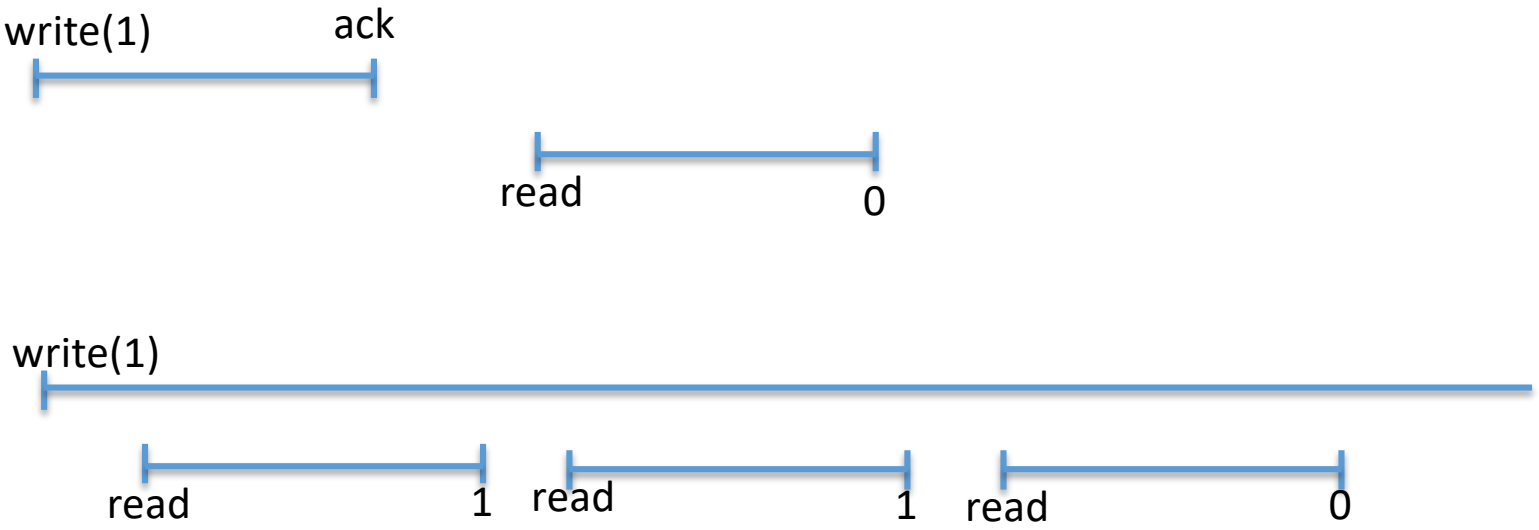
Alternative definition for atomicity (works for (N,N)-atomic registers as well)

- For any operation, there exists a **serialization point**, between the invocation and the reply, such that if we move the invocation and the reply to that point, the resulting execution obeys the sequential specification of a read/write register (operations appear to be executed at some instant between its invocation and reply time)
  - If the last operation does not return, the serialization point may or may not be included
  - (failed writes may or may not complete)

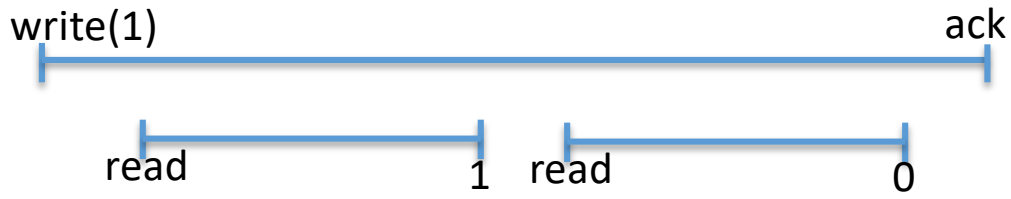
Examples of atomic executions



Non-atomic executions



How can the previous algorithm lead to a non-atomic execution?



- Exercise: draw the timing diagram of an execution that leads to these outputs
- How to fix this problem?

(1,N)-Atomic register implementation

- Write algorithm is the same as regular register
- Add write-back phase after reading
- Intuition: Reader "helps" the concurrent write by completing it in a quorum before returning

Algorithm 4.6: Read-Impose Write-Majority (part 1, read)

Implements:
$(1, N)$-AtomicRegister, **instance** *onar*.

Uses:
BestEffortBroadcast, **instance** *beb*;
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *onar*, *Init* $\rangle$ **do**
$(ts, val) := (0, \bot)$;
$wts := 0$;
acks $:= 0$;
rid $:= 0$;
readlist $:= [\bot]^N$;
readval $:= \bot$;
reading $:=$ FALSE;

upon event $\langle$ *onar*, *Read* $\rangle$ **do**
rid $:=$ *rid* $+ 1$;
acks $:= 0$;
readlist $:= [\bot]^N$;
reading $:=$ TRUE;
trigger $\langle$ *beb*, *Broadcast* | [READ, *rid*] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [READ, r] $\rangle$ **do**
trigger $\langle$ *pl*, *Send* | p, [VALUE, r, ts, val] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | q, [VALUE, r, ts', v'] $\rangle$ **such that** $r =$ *rid* **do**
readlist$[q] := (ts', v')$;
if $\#($*readlist*$) > N/2$ **then**
$($*maxts*, *readval*$) :=$ *highest*(*readlist*);
readlist $:= [\bot]^N$;
trigger $\langle$ *beb*, *Broadcast* | [WRITE, *rid*, *maxts*, *readval*] $\rangle$;

Algorithm 4.7: Read-Impose Write-Majority (part 2, write and write-back)

upon event $\langle$ *onar*, *Write* | v $\rangle$ **do**
    $rid := rid + 1$;
    $wts := wts + 1$;
    $acks := 0$;
    trigger $\langle$ *beb*, *Broadcast* | [WRITE, rid, wts, v] $\rangle$;

upon event $\langle$ *beb*, *Deliver* | p, [WRITE, r, ts', v'] $\rangle$ **do**
    if $ts' > ts$ **then**
        $(ts, val) := (ts', v')$;
    trigger $\langle$ *pl*, *Send* | p, [ACK, r] $\rangle$;

upon event $\langle$ *pl*, *Deliver* | q, [ACK, r] $\rangle$ **such that** $r = rid$ **do**
    $acks := acks + 1$;
    if $acks > N/2$ **then**
        $acks := 0$;
        if *reading* $=$ TRUE **then**
            reading := FALSE;
            trigger $\langle$ *onar*, *ReadReturn* | *readval* $\rangle$;
        else
            trigger $\langle$ *onar*, *WriteReturn* $\rangle$;

From (1,N) to (N,N) atomic register

- With a single writer, writing process can simply increment counter to determine next timestamp
- How to do this with multiple writers?
- Must pick timestamp greater than most recent write
  - writers determine first the timestamp using a majority
- Must be able to break ties
  - timestamp is <seq_number,id> pair

Uses:

BestEffortBroadcast, **instance** *beb*;
PerfectPointToPointLinks, **instance** *pl*.

upon event ⟨ *nnar*, *Init* ⟩ **do**

$(ts, wr, val) := (0, 0, \perp)$;
acks := 0;
writeval := $\perp$;
rid := 0;
readlist := $[\perp]^N$;
readval := $\perp$;
reading := FALSE;

upon event ⟨ *nnar*, *Read* ⟩ **do**

rid := *rid* + 1;
acks := 0;
readlist := $[\perp]^N$;
reading := TRUE;
trigger ⟨ *beb*, *Broadcast* | [READ, *rid*] ⟩;

upon event ⟨ *beb*, *Deliver* | p, [READ, r] ⟩ **do**

trigger ⟨ *pl*, *Send* | p, [VALUE, r, ts, wr, val] ⟩;

upon event ⟨ *pl*, *Deliver* | q, [VALUE, r, ts', wr', v'] ⟩ **such that** $r =$ *rid* **do**

readlist[q] := (ts', wr', v');
if #(*readlist*) > $N/2$ **then**
(*maxts*, *rr*, *readval*) := *highest*(*readlist*);
readlist := $[\perp]^N$;
if *reading* = TRUE **then**
trigger ⟨ *beb*, *Broadcast* | [WRITE, *rid*, *maxts*, *rr*, *readval*] ⟩;
else
trigger ⟨ *beb*, *Broadcast* | [WRITE, *rid*, *maxts* + 1, *rank*(*self*), *writeval*] ⟩;

```
upon event ⟨ nnar, Write | v ⟩ do
    rid := rid + 1;
    writeval := v;
    acks := 0;
    readlist := [⊥]^N;
    trigger ⟨ beb, Broadcast | [READ, rid] ⟩;

upon event ⟨ beb, Deliver | p, [WRITE, r, ts', wr', v'] ⟩ do
    if (ts', wr') is larger than (ts, wr) then
        (ts, wr, val) := (ts', wr', v');
    trigger ⟨ pl, Send | p, [ACK, r] ⟩;

upon event ⟨ pl, Deliver | q, [ACK, r] ⟩ such that r = rid do
    acks := acks + 1;
    if acks > N/2 then
        acks := 0;
        if reading = TRUE then
            reading := FALSE;
            trigger ⟨ nnar, ReadReturn | readval ⟩;
        else
            trigger ⟨ nnar, WriteReturn ⟩;
```

Byzantine (1,N) read/write regular register

- To specify registers in the Byzantine model, we will simplify by considering that clients only suffer crash faults
- With this restriction and adapting the notion of "correct", specification is the same as in crash model
- How to implement?

Strawman attempt

- Start with (1,N) regular register algorithm
- Replace links with perfect authenticated links
- Replace majorities with Byzantine quorums of 2f+1 out of 3f+1
- Does this work?
- Problem: Byzantine process can return an arbitrary <ts,value>

Illustration of problem with strawman

- Exercise: draw a timing diagram to illustrate why the strawman doesn't work

- Solution?
  - Make written values and timestamp unforgeable by byzantine processes… how?

Solution: clients sign written values

- Upon writing, clients piggyback signature of <seq_number,value>
- Replicas store this signature and return it upon read requests
- When reading, clients discard incorrectly signed values
- Need to send identifier associated with read (rid), to avoid replay attacks

Implements:

$(1, N)$-ByzantineRegularRegister, **instance** *bonrr*, with writer w.

Uses:

AuthPerfectPointToPointLinks, **instance** *al*.

upon event $\langle$ *bonrr*, *Init* $\rangle$ **do**

$(ts, val, \sigma) := (0, \bot, \bot)$;
$wts := 0$;
$acklist := [\bot]^N$;
$rid := 0$;
$readlist := [\bot]^N$;

upon event $\langle$ *bonrr*, *Write* | v $\rangle$ **do** // only process w

$wts := wts + 1$;
$acklist := [\bot]^N$;
$\sigma := sign(self, bonrr||self||\text{WRITE}||wts||v)$;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [WRITE, wts, v, σ] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [WRITE, ts', v', σ'] $\rangle$ **such that** $p = w$ **do**

if $ts' > ts$ **then**
$(ts, val, \sigma) := (ts', v', \sigma')$;
trigger $\langle$ *al*, *Send* | p, [ACK, ts'] $\rangle$;

upon event $\langle$ *al*, *Deliver* | q, [ACK, ts'] $\rangle$ **such that** $ts' = wts$ **do**
    $acklist[q] := \text{ACK}$;
    if $\#(acklist) > (N + f)/2$ **then**
        $acklist := [\bot]^N$;
        trigger $\langle$ *bonrr*, *WriteReturn* $\rangle$;

upon event $\langle$ *bonrr*, *Read* $\rangle$ **do**
    $rid := rid + 1$;
    $readlist := [\bot]^N$;
    forall $q \in \Pi$ **do**
        trigger $\langle$ *al*, *Send* | q, [READ, rid] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [READ, r] $\rangle$ **do**
    trigger $\langle$ *al*, *Send* | p, [VALUE, r, ts, val, σ] $\rangle$;

upon event $\langle$ *al*, *Deliver* | q, [VALUE, r, ts', v', σ'] $\rangle$ **such that** $r = rid$ **do**
    if $verifysig(q, bonrr \| w \| \text{WRITE} \| ts' \| v', \sigma')$ **then**
        $readlist[q] := (ts', v')$;
        if $\#(readlist) > \frac{N+f}{2}$ **then**
            $v := highestval(readlist)$;
            $readlist := [\bot]^N$;
            trigger $\langle$ *bonrr*, *ReadReturn* | v $\rangle$;

Byzantine (1,N) read/write atomic register

- Also assume crash-faulty clients
- Also same specification as in the crash model
- Similar adaptation to the crash-fault-tolerant counterparts (from regular to atomic registers)
  - Add write back phase to read algorithm
  - Exercise: extend the algorithm from the previous two slides

Byzantine clients

- Now, assume that clients can be byzantine in digital signature-based BFT algorithm
- Read operations:
  - Do not modify the state of the processes → safe from everyone else's point of view
- Write operations:
  - Byzantine clients can pick wrong timestamps, send write requests for different values to different servers, or only write to a subset
  - Leave it as an exercise to think through consequences and countermeasures

Acknowledgements

- Rachid Guerraoui, EPFL