

## 6.1 ATOMIC Register Execution

Let processes  $\{p, q, r, s, t\}$  have **ranks**  $\{1, 2, 3, 4, 5\}$  respectively.

(a)  $read_r() \rightarrow x$  **and**  $read_s() \rightarrow y$

First we can say that  $x$  has been written immediately after  $write_p(x)$  has started. Therefore  $r$ ,  $s$  and  $t$  have stored  $x$  when the  $read_r()$  operation terminates hence it returns  $x$ . Before  $read_s()$  operation is executed the process  $s$  will store the value of  $y$  because although the timestamps of both  $write$  operation is equal to the **rank** of  $q$  is higher. Therefore process  $s$  can return  $y$ . Because the  $read_t()$  operation is called after the  $write$  operations have finished and process  $q$  has a higher rank than process  $p$  it will also return  $y$ .

(b)  $read_r() \rightarrow y$  **and**  $read_s() \rightarrow x$

First we can say that  $x$  has been written immediately after  $write_p(x)$  has started. Therefore  $r$ ,  $s$  and  $t$  have stored  $x$  when the  $read_s()$  operation terminates hence it returns  $x$ . Before  $read_r()$  operation is executed the process  $r$  will store the value of  $y$  because although the timestamps of both  $write$  operation is equal to the **rank** of  $q$  is higher. Therefore process  $r$  can return  $y$ . Because the  $read_t()$  operation is called after the  $write$  operations have finished and process  $q$  has a higher rank than process  $p$  it will also return  $y$  (same argumentation as above, just switched  $r$  and  $s$ ).

## 6.2 Erasure-Coded Storage

(a) Erasure Code Picking

A suitable erasure code for a distributed storage system with at most  $\frac{n}{2}$  crashed nodes could be any  $(\lceil \frac{n}{2} \rceil, n)$  erasure code. A good abstraction of such an erasure code could be the following *Reed-Solomon* code:

Let  $k := \lceil \frac{n}{2} \rceil$ . The message  $m$  which should be stored is split up into bit-strings  $m_i$  of length  $k$ . For each of these bit-strings  $m_i := (x_1, x_2, \dots, x_k)$  we compute the Lagrange polynomial  $p$  such that  $p(i-1) = x_i$ . These values  $p(i)$  are then stored at node number  $i \mid 0 \leq i \leq n-1$ .

An erasure code implemented like this with  $(\text{int})f < \frac{n}{2}$  failed nodes, it holds that  $n - f \geq \lceil \frac{n}{2} \rceil$ , which is a sufficient number of fragments to reconstruct  $m_i$ .

The storage efficiency of any  $(\lceil \frac{n}{2} \rceil, n)$  erasure code is  $\approx 50\%$ .

**(b) Modify majority voting REGULAR register to an erasure-coded SAFE register**

The Algorithm 4.2 can be adjusted as the following. Not changed event handlings are left out and are similar as in the algorithm. Furthermore we assume that a suitable lagrange interpolation is available from an external source.

Implements:  
 $(1, N)$  SAFE register, *instance* *onsr*

upon  $\langle \text{onsr}, \text{WRITE} \mid v \rangle$  do  
 $v = (v_1, v_2, \dots, v_k)$ , **whereas**  $k = \frac{N}{2}$   
 $\text{wts} := \text{wts} + 1$   
 $\text{acks} := 0$   
 $p := \text{lagrange\_interpolation}((0, v_1), \dots, (k-1, v_k))$   
for  $i = 1$  to  $N$ :  
trigger  $\langle pl, \text{SEND} \mid p_i, [\text{WRITE}, \text{wts}, ((i-1), p(i-1))] \rangle$

upon  $\langle pl, \text{DELIVER} \mid q, [\text{VALUE}, r, ts', v'] \rangle$  s.t.  $r = \text{rid}$  do  
 $\text{readlist}[q] := (ts', v')$   
if  $|\text{readlist}| > N/2$  then  
 $p := \text{reconstruct\_polynom}(\text{readlist})$   
for  $i := 1$  to  $N/2$ :  
 $v_i := p(i-1)$   
trigger  $\langle \text{onsr}, \text{READRETURN} \mid (v_1 \parallel \dots \parallel v_k) \rangle$

**(c) Why is it difficult to extend this protocol to regular semantics?**

**REGULAR:** A read() not concurrent with a write returns the most recently written value. Otherwise read() returns the most recently written value or the concurrently written value.

For the erasure code register the information of any value  $v$  is spread across  $n$  different registers, whereas none of these alone has sufficient information to reconstruct that value  $v$ . Because of the need of information from multiple different registers to reconstruct  $v$ , it is not trivial to ensure that the read is consistent, which would lead to a REGULAR behaviour.