Improving the Hu-Toueg Construction of a Byzantine Linearizable SWMR Register

Ajay D. Kshemkalyani, Manaswini Piduguralla, **Sathya Peri**, Anshuman Misra

27^{th} International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2025)

10 October 2025

Table of Contents

Table of Contents

Research Objective

Construct a Byzantine linearizable SWMR[1] atomic register using SWSR[2] registers.

The objective of our research is to :

- Linearizability: stronger definition of a Byzantine Linearizability
- Writer and the readers can be Byzantine

[1]SWMR: Single Writer Multi Reader
[2]SWSR: Single Writer Single Reader

Table of Contents

Register Class

Register Class[3]:

A **register** is an object that stores a value of some type and provides simple operations to access or modify it:

- The method `read()` returns the current value stored in the register.
- The method `write(v)` updates the register with a new value `v`.

[3] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780123973375, 9780123977953.

Atomic Register

Atomic Register[4]:

- An atomic register is a linearizable implementation of the sequential register class.
- All reads and writes appear to occur at a single point in time, even when concurrent with other operations.

[4] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780123973375, 9780123977953.

Atomic Register

Atomic Register[4]:

- An atomic register is a linearizable implementation of the sequential register class.
- All reads and writes appear to occur at a single point in time, even when concurrent with other operations.

Types of Atomic Registers:

- SWSR: Single Writer Single Reader
- SWMR: Single Writer Multi Reader
- MWSR: Multi Writer Single Reader
- MWMR: Multi Writer Multi Reader

[4] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780123973375, 9780123977953.

Types of Atomic Registers

Types of Atomic Registers:

- SWSR: Single Writer Single Reader
- SWMR: Single Writer Multi Reader
- MWSR: Multi Writer Single Reader
- MWMR: Multi Writer Multi Reader

Types of Atomic Registers

Types of Atomic Registers:

- SWSR: Single Writer Single Reader
- SWMR: Single Writer Multi Reader
- MWSR: Multi Writer Single Reader
- MWMR: Multi Writer Multi Reader

Construction Hierarchy:

$$\text{SWSR} \Rightarrow \text{SWMR} \Rightarrow \text{MWSR} \Rightarrow \text{MWMR}$$

Atomic Register: SWSR Example

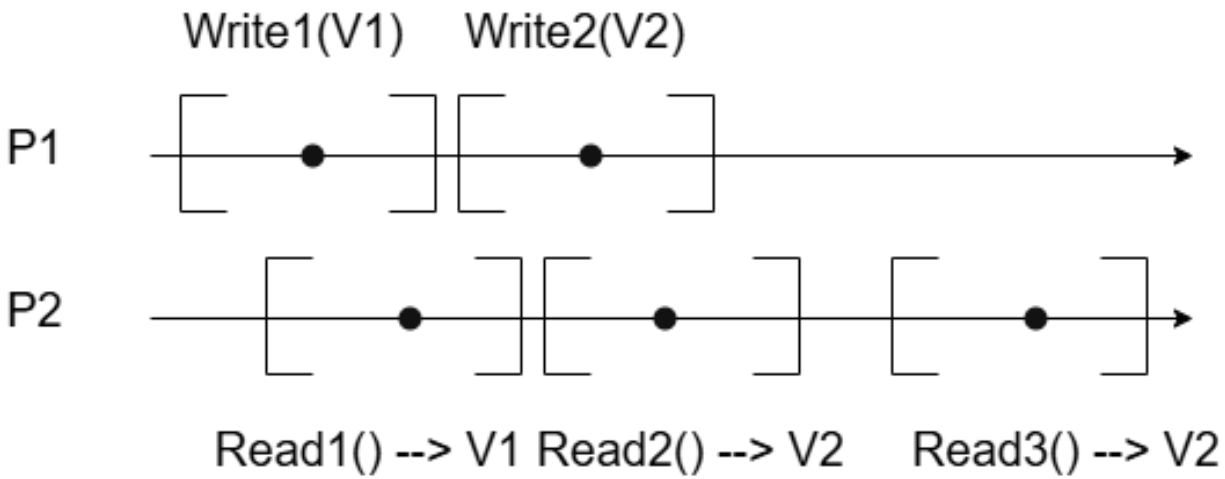

Figure: Concurrent History

Atomic Register: SWSR Example

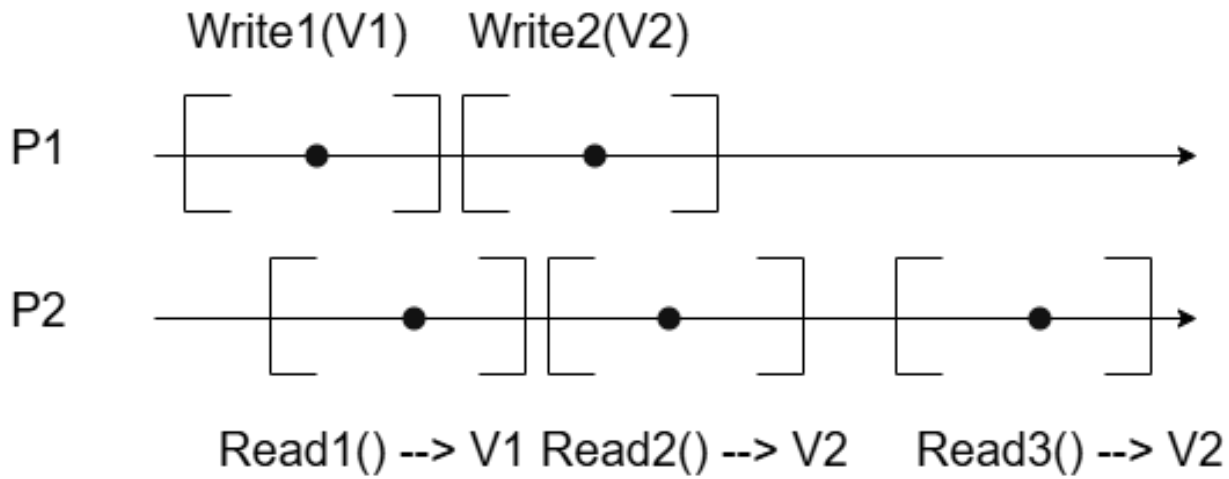

Figure: Concurrent History

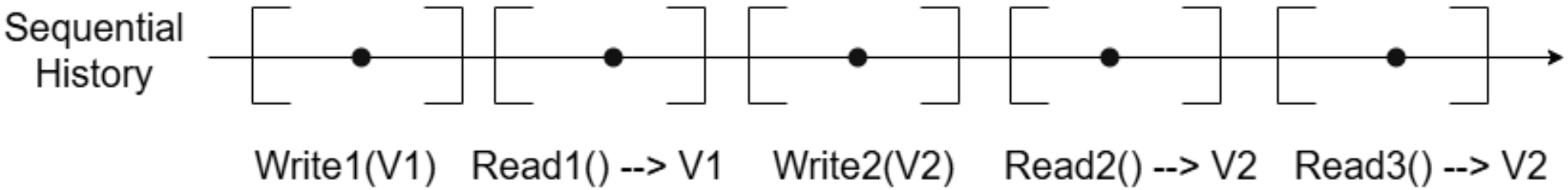

Figure: Sequential History

Faults

Fault Types:

Faults

Fault Types:

- **Crash Fault:** A fault in which a process abruptly stops working and ceases all operations.
  - The process does not recover or perform any additional actions.

Faults

Fault Types:

- **Crash Fault:** A fault in which a process abruptly stops working and ceases all operations.
  - The process does not recover or perform any additional actions.
- **Byzantine Fault:** A fault where a process may exhibit arbitrary behavior,
  - sending conflicting or incorrect information to other processes,
  - due to malicious or unpredictable issues.

Prior Works

- **Cohen & Keidar (2021)** – Byzantine Linearizability definition[5]
- **Mostefaoui et al. (2017)** – SWMR over message passing[6]

[5] Shir Cohen and Idit Keidar. "Tame the Wild with Byzantine Linearizability: Reliable Broadcast, Snapshots, and Asset Transfer". In: *35th International Symposium on Distributed Computing, DISC 2021*. Ed. by Seth Gilbert. Vol. 209. LIPIcs. 2021.

[6] Achour Mostéfaoui et al. "Atomic Read/Write Memory in Signature-Free Byzantine Asynchronous Message-Passing Systems". In: *Theory Comput. Syst.* 60.4 (2017).

Prior Works

- **Hu & Toueg (2022, 2024)** – SWMR from SWSR under Byzantine faults[7,8]

[7] Xing Hu and Sam Toueg. "On Implementing SWMR Registers from SWSR Registers in Systems with Byzantine Failures". In: *36th International Symposium on Distributed Computing, DISC 2022, , Augusta, Georgia, USA*. vol. 246. LIPIcs. 2022.

[8] Xing Hu and Sam Toueg. "On implementing SWMR registers from SWSR registers in systems with Byzantine failures". In: *Distributed Comput.* 37.2 (2024), pp. 145–175. DOI: 10.1007/S00446-024-00465-5.

Histories

History:[9] An execution of a system is modeled by a history, which is a finite sequence of operation (e.g., reads, writes) invocation and response events.

1. A history H defines a partial order $\prec_H$ on operations. $op_1 \prec_H op_2$ if the response event of op_1 precedes the invocation event of op_2 in H.
2. op_1 is concurrent with op_2 if neither precedes the other.

[9] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780123973375, 9780123977953.

History Example

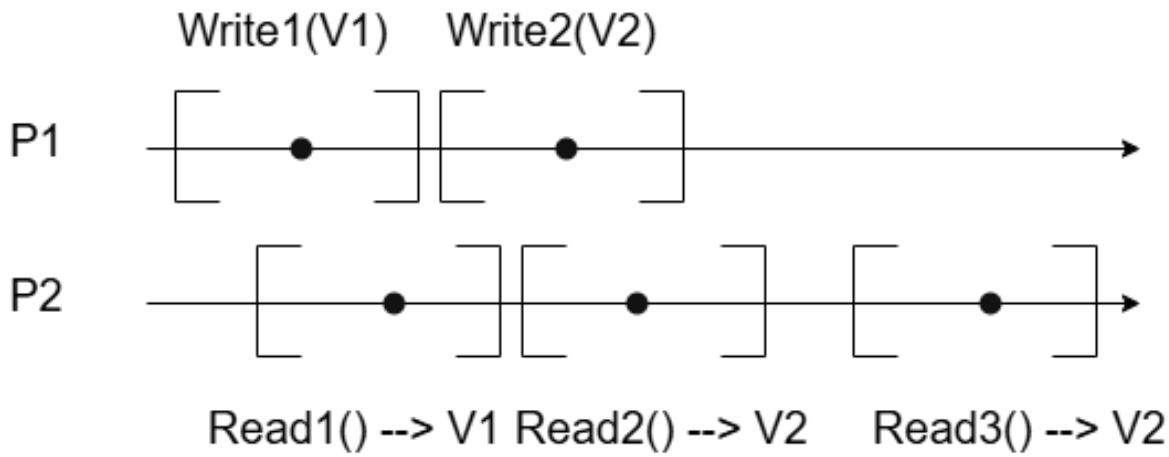

Figure: Register History Example

In the above History:

1. $Write1 \prec_H Write2$
2. $Write1 \prec_H Read2$
3. $Write1 \prec_H Read3$
4. $Write2 \prec_H Read3$

1. $Read1 \prec_H Read2$
2. $Read1 \prec_H Read3$
3. $Read2 \prec_H Read3$

Table of Contents

Linearizability of a Concurrent History

A *linearization* of a concurrent history H of object o is a sequential history H' such that:

1. After removing some pending operations from H and completing others by adding matching responses, it contains the same invocations and responses as H',
2. H' preserves the partial order $\prec_H$, and
3. H' satisfies o's sequential specification.

Byzantine linearizable

Byzantine linearization of a concurrent history[10]**:** A history H is Byzantine linearizable if there exists a history H' such that $H'|_{correct} = H|_{correct}$ and H' is linearizable.

- $H|_{correct}$ denote the projection of history H to all correct processes.
- An object supports Byzantine linearizable executions if all of its executions are Byzantine linearizable.

[10] Shir Cohen and Idit Keidar. "Tame the Wild with Byzantine Linearizability: Reliable Broadcast, Snapshots, and Asset Transfer". In: *35th International Symposium on Distributed Computing, DISC 2021*. Ed. by Seth Gilbert. Vol. 209. LIPIcs. 2021.

Register Linearizability (Simple View)

Definition: In a system with possible Byzantine (malicious) processes, a **Single-Writer Multi-Reader (SWMR)** register is *linearizable* if, when the writer is honest, all non-malicious readers observe behavior consistent with sequential specification.

Key Properties:

1. **Reading a Current Value:** A read returns a value v only if:
   - v was written just before or during the read, **or**
   - $v = v_0$ (the initial value) and no writes occurred before.

Register Linearizability (Simple View)

Definition: In a system with possible Byzantine (malicious) processes, a **Single-Writer Multi-Reader (SWMR)** register is *linearizable* if, when the writer is honest, all non-malicious readers observe behavior consistent with sequential specification.

Key Properties:

1. **Reading a Current Value:** A read returns a value v only if:
   - v was written just before or during the read, **or**
   - $v = v_0$ (the initial value) and no writes occurred before.
2. **No "New–Old" Inversion:** If one read R happens before another R', then R' cannot return an **older** value than R.

Register Linearizability

Register Linearizability[11]**:** In a system with Byzantine process failures, an implementation of a SWMR register is linearizable if and only if the following holds. If the writer is not malicious, then:

[11] Xing Hu and Sam Toueg. "On Implementing SWMR Registers from SWSR Registers in Systems with Byzantine Failures". In: *36th International Symposium on Distributed Computing, DISC 2022, , Augusta, Georgia, USA*. vol. 246. LIPIcs. 2022.

Register Linearizability

Register Linearizability[11]**:** In a system with Byzantine process failures, an implementation of a SWMR register is linearizable if and only if the following holds. If the writer is not malicious, then:

- (Reading a "current" value) If a read operation R by a process that is not malicious returns the value v then:
  - there is a write v operation that immediately precedes R or is concurrent with R, or
  - $v = v_0$ (the initial value) and no write operation precedes R.

[11] Xing Hu and Sam Toueg. "On Implementing SWMR Registers from SWSR Registers in Systems with Byzantine Failures". In: *36th International Symposium on Distributed Computing, DISC 2022, , Augusta, Georgia, USA*. vol. 246. LIPIcs. 2022.

Register Linearizability

Register Linearizability[11]**:** In a system with Byzantine process failures, an implementation of a SWMR register is linearizable if and only if the following holds. If the writer is not malicious, then:

- (Reading a "current" value) If a read operation R by a process that is not malicious returns the value v then:
  - there is a write v operation that immediately precedes R or is concurrent with R, or
  - $v = v_0$ (the initial value) and no write operation precedes R.
- (No "new-old" inversion) If two read operations R and R' by processes that are not malicious return values v_k and $v_{k'}$, respectively, and R precedes R', then $k \leq k'$.

Here k indicates the timestamp of the written value.

[11] Xing Hu and Sam Toueg. "On Implementing SWMR Registers from SWSR Registers in Systems with Byzantine Failures". In: *36th International Symposium on Distributed Computing, DISC 2022, , Augusta, Georgia, USA*. vol. 246. LIPIcs. 2022.

Drawback of Existing Work

1. Drawbacks: If the writer is Byzantine, the register is vacuously linearizable no matter what values the correct readers return.
2. Contribution: In this work, we give some directions to the correct readers when the writer is Byzantine.

Directions for Writers: Pseudo-Correct Write

pseudo-correct operation definition[12]

We use the notion of *Pseudo-Correct Write* to give direction for readers.

[12] Ajay D. Kshemkalyani et al. "On Constructing a Byzantine Linearizable SWMR Atomic Register from SWSR Atomic Registers". In: *Proceedings of the 26th International Conference on Distributed Computing and Networking, ICDCN 2025, Hyderabad, India, January 4-7, 2025*. Ed. by Amos Korman et al. ACM, 2025, pp. 239–243.

Directions for Writers: Pseudo-Correct Write

pseudo-correct operation definition[12]

We use the notion of *Pseudo-Correct Write* to give direction for readers.

A pseudo-correct operation is an operation executed by a process where it results in a value being returned to honest processes, is indistinguishable from the steps of a correct operation.

[12] Ajay D. Kshemkalyani et al. "On Constructing a Byzantine Linearizable SWMR Atomic Register from SWSR Atomic Registers". In: *Proceedings of the 26th International Conference on Distributed Computing and Networking, ICDCN 2025, Hyderabad, India, January 4-7, 2025*. Ed. by Amos Korman et al. ACM, 2025, pp. 239–243.

Example Pseudo-Correct Write linearizable

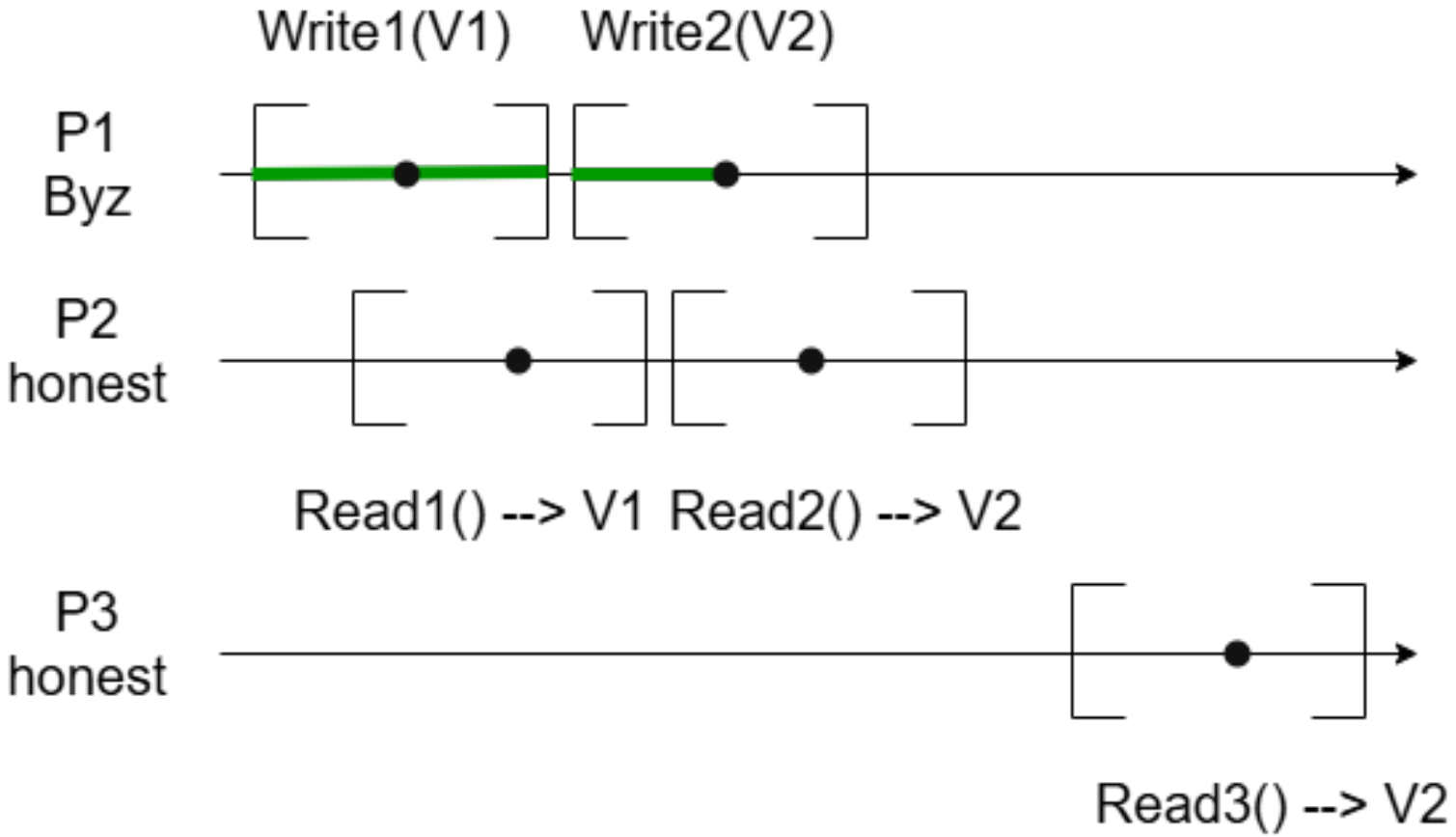

Figure: Byzantine Linearizable Register

Example Pseudo-Correct Write linearizable

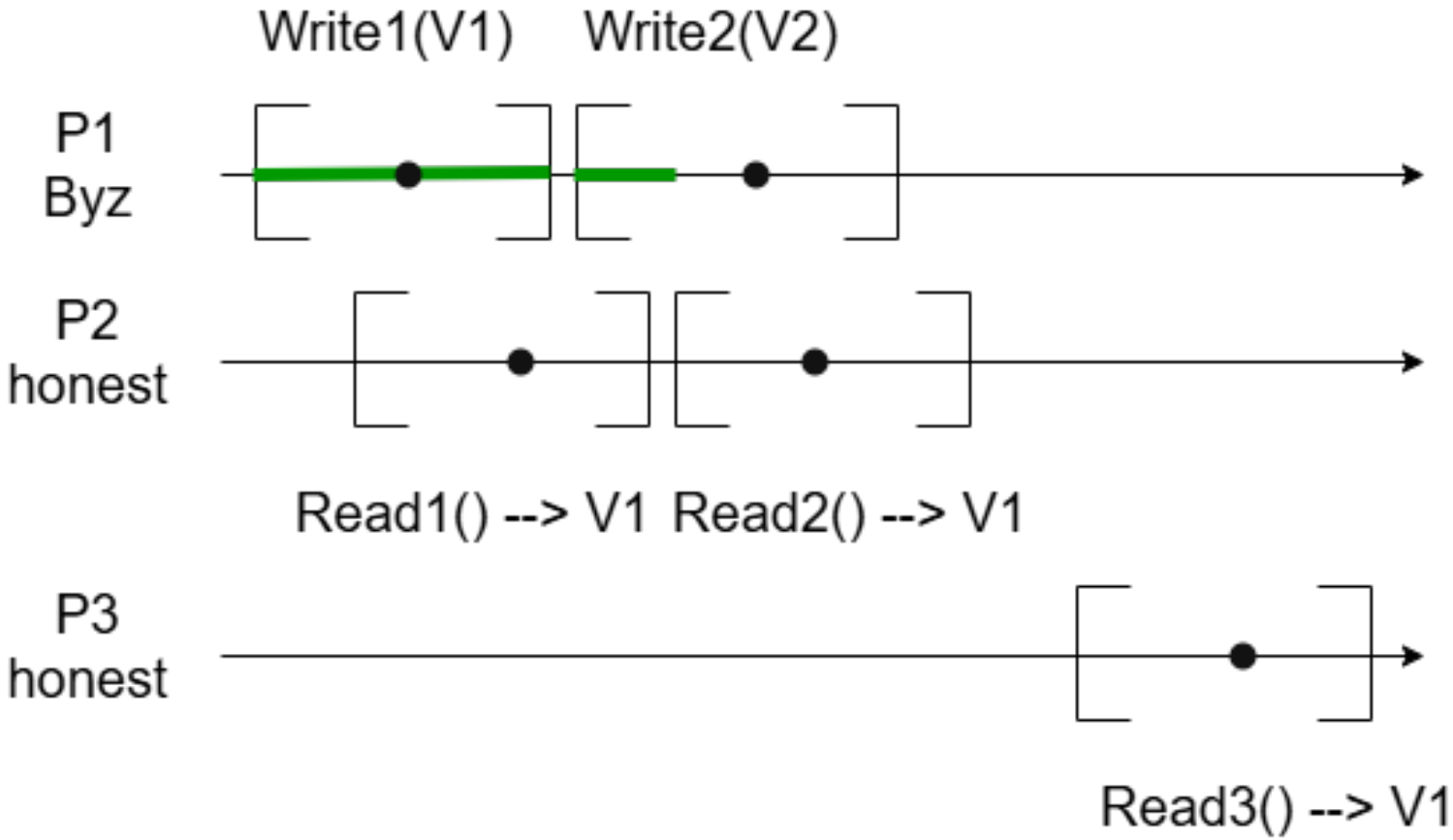

Figure: Byzantine Linearizable Register

Register Linearizability

Register Linearizability[13] In a system with Byzantine process failures, an implementation of a SWMR register is linearizable if and only if the following two properties are satisfied in a Byzantine linearization of a concurrent history.

[13] Ajay D. Kshemkalyani et al. "On Constructing a Byzantine Linearizable SWMR Atomic Register from SWSR Atomic Registers". In: *Proceedings of the 26th International Conference on Distributed Computing and Networking, ICDCN 2025, Hyderabad, India, January 4-7, 2025*. Ed. by Amos Korman et al. ACM, 2025, pp. 239–243.

Register Linearizability

Register Linearizability[13] In a system with Byzantine process failures, an implementation of a SWMR register is linearizable if and only if the following two properties are satisfied in a Byzantine linearization of a concurrent history.

1. **Reading a current value:** When a read operation R by a non-Byzantine process returns the value v:
   1. if $v = v_0$ then no correct or pseudo-correct write operation precedes R
   2. else if $v \neq v_0$ then v was written by the correct or pseudo-correct write operation that immediately precedes R.
2. **No "new-old" inversions:** If read operations R and R' by non-Byzantine processes return values v^i and v^j, respectively, and R precedes R', then $i \leq j$.

[13] Ajay D. Kshemkalyani et al. "On Constructing a Byzantine Linearizable SWMR Atomic Register from SWSR Atomic Registers". In: *Proceedings of the 26th International Conference on Distributed Computing and Networking, ICDCN 2025, Hyderabad, India, January 4-7, 2025*. Ed. by Amos Korman et al. ACM, 2025, pp. 239–243.

Table of Contents

pseudo-correct operation

pseudo-correct operation

A potential pseudo-correct write operation of value (k, v) is a write operation, timestamped k, that may not follow the write protocol but

1. there is quorum of *size* $\geq n - 2t$ indices i of correct readers such that (k, v) was written to R_init_{wi}, and
2. $k > k'$ for all (k', v') already read from these R_init_{wi}

Our Register Setup for the writer Process P_w

Atomic SWSR registers:

1. The writer process P_w maintains n SWSR registers for each reader process P_r for initialization of value and the timestamp.

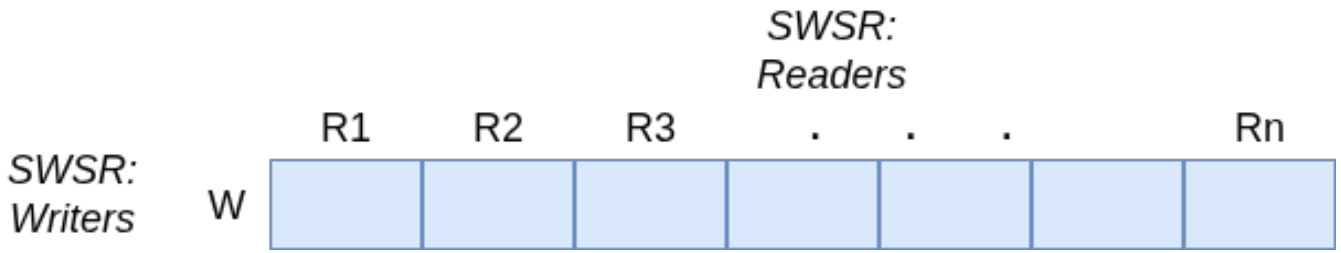

Figure: R_init_{wi} Register

2. Every reader process P_i maintains 1 SWSR register with the writer P_w for acknowledgement.

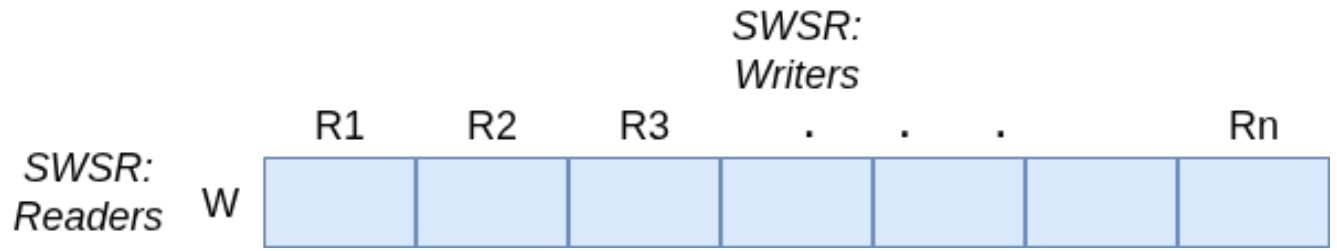

Figure: R_ack_{wi} Register

Our Register Setup for a Process P_i

Atomic SWSR registers:

1. Every reader process P_i maintains 2 SWSR registers with every other reader process P_j for sharing witness and final values.

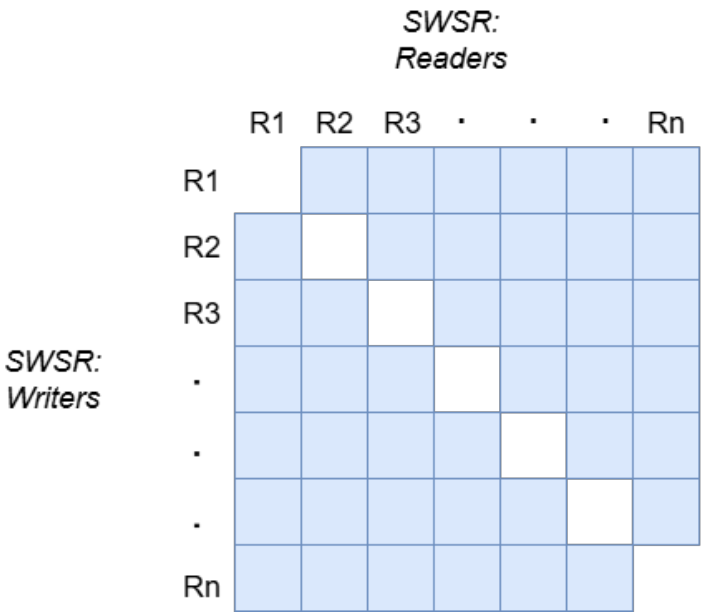

Figure: $R\_witness_{ij}$ Register

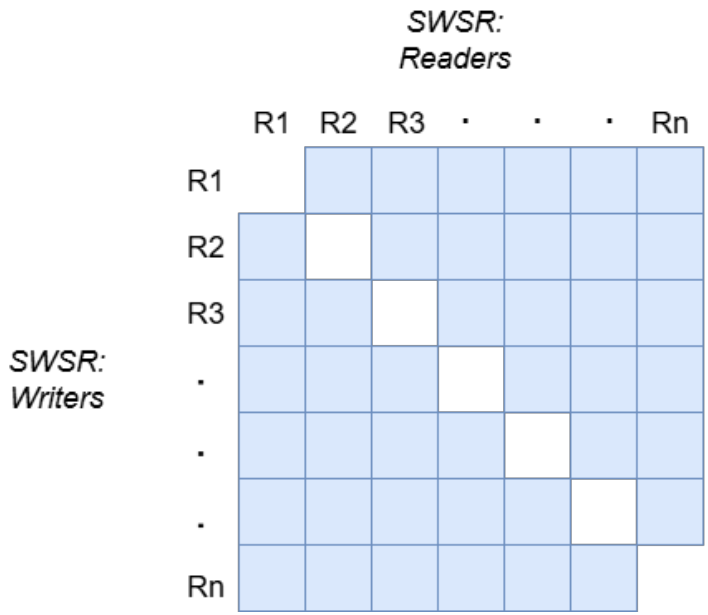

Figure: R_final_{ij} Register

Write Protocol

Algorithm 1: Write Protocol

```
1  W(⟨k, u⟩):
2  for each process i ∈ P do
3      R_init_wi ← ⟨k, u⟩
4  d ← 0
5  while d < n − t do
6      for each new ⟨k, u⟩ in R_ack_iw among R_ack registers do
7          d ← d + 1
   return done
```

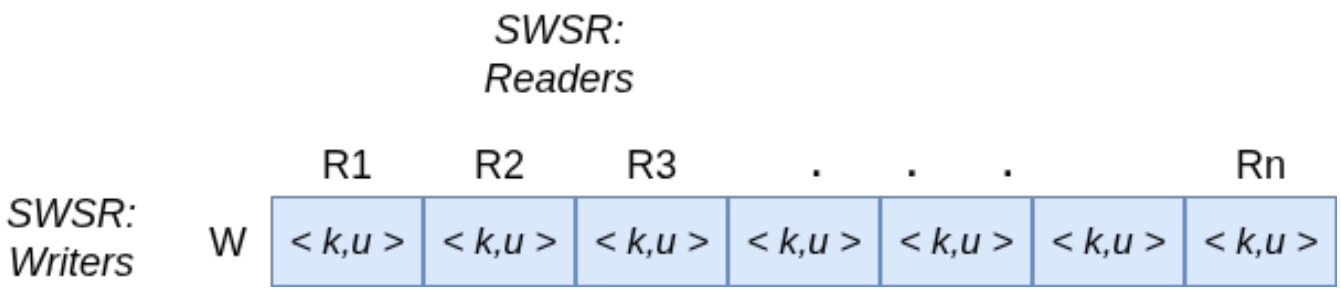

Figure: R_ack_{wi} Register

Write Protocol

Algorithm 2: Write Protocol

1 $\underline{W(\langle k, u\rangle)}$:
2 **for each** *process* $i \in P$ **do**
3 $\quad R\_\text{init}_{wi} \leftarrow \langle k, u\rangle$
4 $d \leftarrow 0$
5 **while** $d < n - t$ **do**
6 $\quad$ **for each new** $\langle k, u\rangle$ **in** R_ack_{iw} **among** R_ack *registers* **do**
7 $\quad\quad d \leftarrow d + 1$

return done

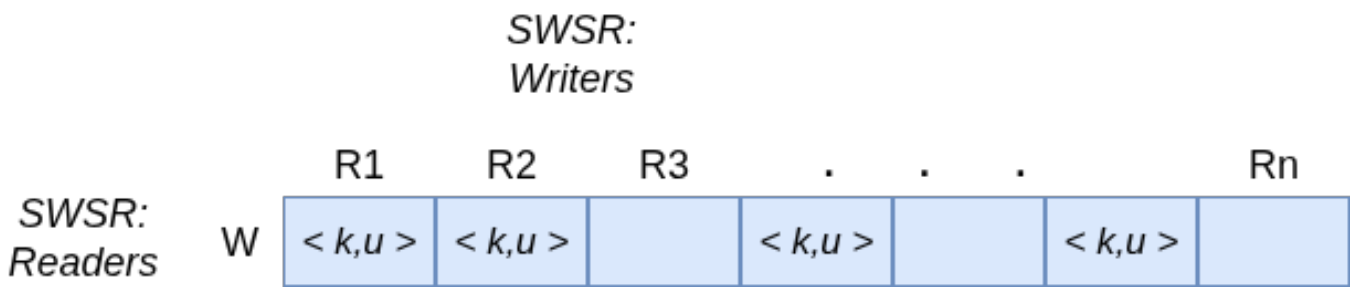

Figure: R_ack_{wi} Register

Reader Helper Thread — Intuition

- Helper Thread run continuously at each **reader process**.
- Help construct a consistent view of the register value.
- Reader Helper Protocol:
  1. R_init_{wi}: Watch for new writes with increasing counter value k in .
  2. $R\_witness_{pi}$: Share and collect *witness* updates from other processes.
  3. $T\_witness_i.k$: Combine *witness* reports $(n - t)$ locally to decide on a value.
  4. R_final_{pi} Update others with the *final* consistent view for the new write.
  5. R_ack_{wi}: Inform the writer that the latest value is *acknowledged*.

Table of Contents

Conclusion

- We proposed an algorithm to construct a Byzantine-tolerant SWMR atomic register using SWSR atomic registers.
- The algorithm satisfies the stronger Byzantine register linearizability definition.

Conclusion

- We proposed an algorithm to construct a Byzantine-tolerant SWMR atomic register using SWSR atomic registers.
- The algorithm satisfies the stronger Byzantine register linearizability definition.
- Future Work:
  - We are designing a new solution that eliminates k ((Byzantine) writer assigned timestamp) and utilizes vector time.
  - Is it possible to achieve this without the helper threads?
  - Explore if SWSR registers are practically possible?

Thank you

Contact: *sathya_p@cse.iith.ac.in*

Read Protocol

Algorithm 3: Detect New Writer Initialization

1 **if** $R\_init_{wp} = \langle k, u \rangle$ *is newly written and* $k > k\_init\_latest$ **then**
2 $\quad k\_init\_latest \leftarrow k$
3 $\quad$ **for each** $i \in P$ **do**
4 $\quad\quad R\_witness_{pi} \leftarrow \langle \langle k, u \rangle, p \rangle_p$

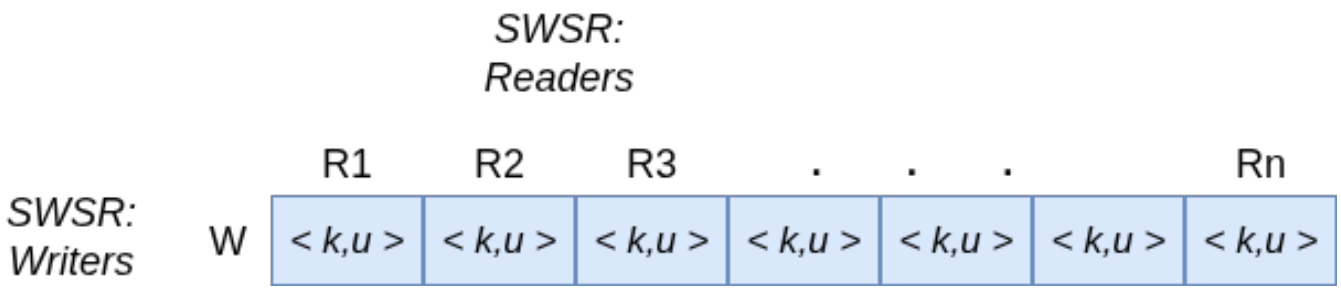

Figure: R_init_{wi} Register

Read Protocol

Algorithm 4: Detect New Writer Initialization

```
1 if R_init_wp = ⟨k, u⟩ is newly written
    and k > k_init_latest then
2     k_init_latest ← k
3     for each i ∈ P do
4         R_witness_pi ← ⟨⟨k, u⟩, p⟩_p
```

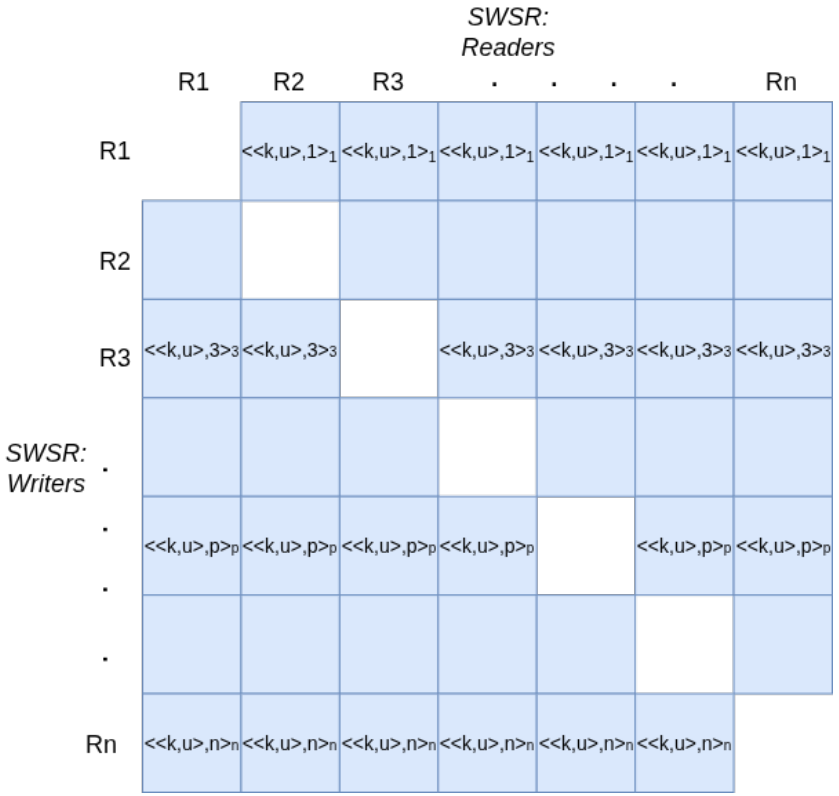

Figure: $R\_witness_{wi}$ Register

Read Protocol

Algorithm 5: Update Local Witness Records

1 **for each** $i \in P$ **do**
2 **if** $R\_witness_{ip} = \langle\langle k, u\rangle, i\rangle_i$ *is newly written and* $k > T\_witness_i.k$ **then**
3 $T\_witness_i \leftarrow \langle\langle k, u\rangle, i\rangle_i$
4 **else if** $(\langle k, u\rangle.u \neq T\_witness_i.\langle k, u\rangle.u) \vee (k \leq T\_witness_i.k)$ **then**
5 mark process i as Byzantine (optional)
6 **if** $\exists \geq n - t$ *latest elements* $\langle\langle k, u\rangle, q\rangle_q$ *in* $T\_witness_q$ **then**
7 $Witness\_Set \leftarrow \{\text{these} \geq n - t \text{ elements}\}$
8 **for each** $i \in P$ **do**
9 $R\_final_{pi} \leftarrow Witness\_Set$
10 $R\_ack_{pw} \leftarrow \langle k, u\rangle$

Read Protocol

Algorithm 6: Detect New Writer Initialization

1 **if** $\exists \geq n - t$ *latest elements* $\langle\langle k, u\rangle, q\rangle_q$ *in* $T\_witness_q$ **then**
2 $Witness_Set \leftarrow \{$these $\geq n - t$ elements$\}$
3 **for each** $i \in P$ **do**
4 $R\_final_{pi} \leftarrow Witness\_Set$
5 $R\_ack_{pw} \leftarrow \langle k, u\rangle$

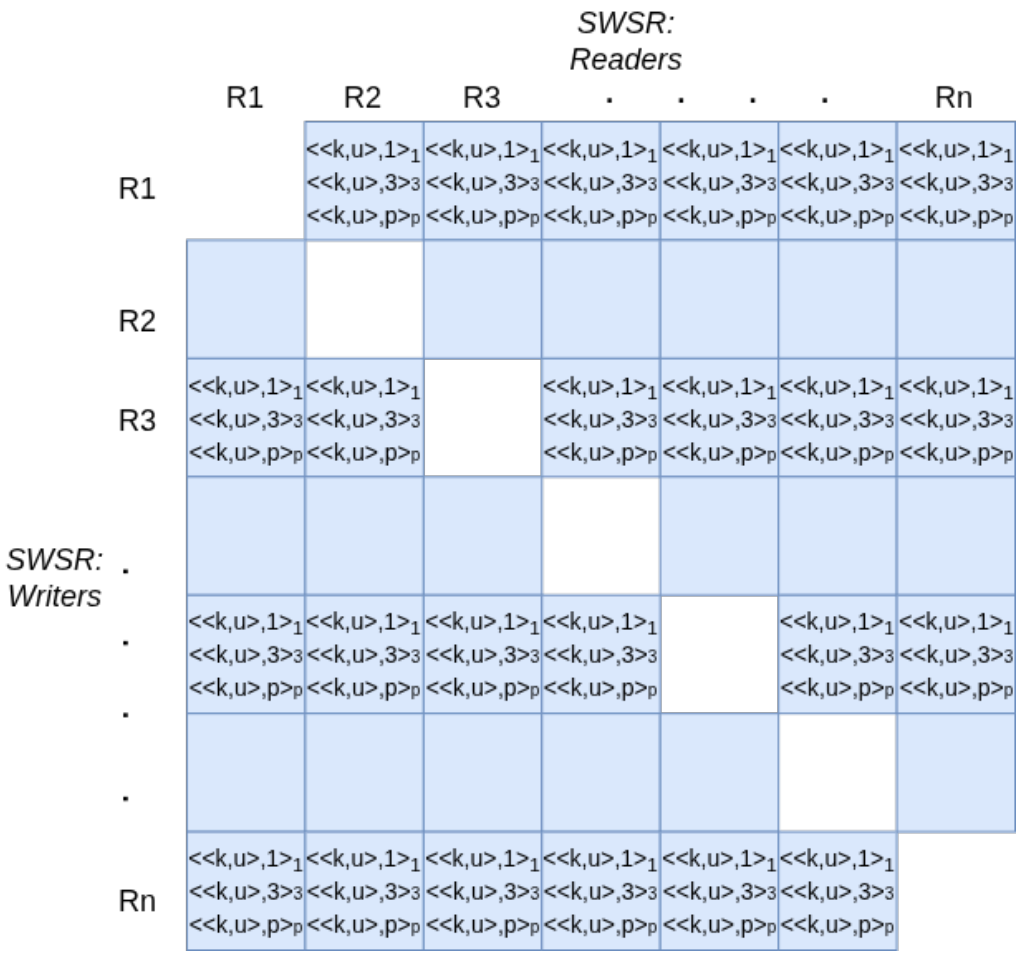

Figure: R_final_{wi} Register

Read Protocol

Algorithm 7: Aggregate Final Sets from Peers

1 $Z \leftarrow \emptyset$

2 **for each** $i \in P$ **do**

3 $\quad Z \leftarrow Z \cup \{R\_final_{ip}\}$

4 $Y \leftarrow Find\_Latest(Z)$ **if** $\langle k, u \rangle$ *in* $Y \neq \langle k, u \rangle$ *in* $Witness\_Set$ **then**

5 $\quad$ **for each** $i \in P$ **do**

6 $\quad\quad R\_final_{pi} \leftarrow Y$

7 $\quad R\_ack_{pw} \leftarrow$ common $\langle k, u \rangle$ in Y

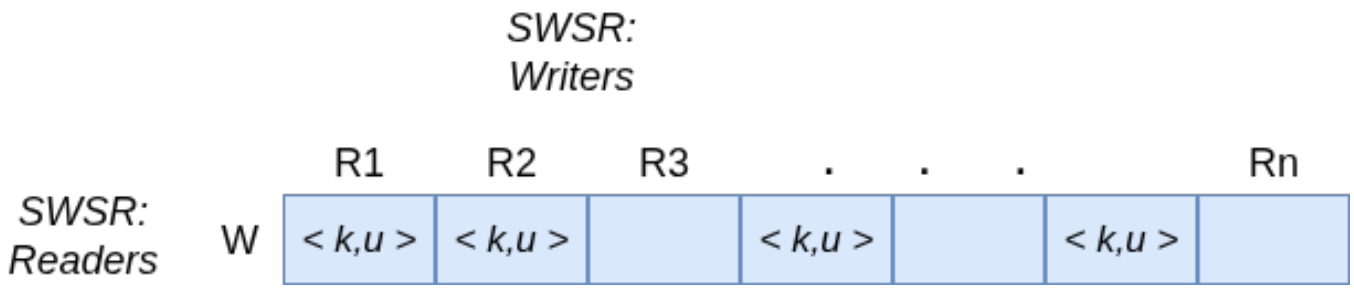

Figure: R_ack_{wi} Register