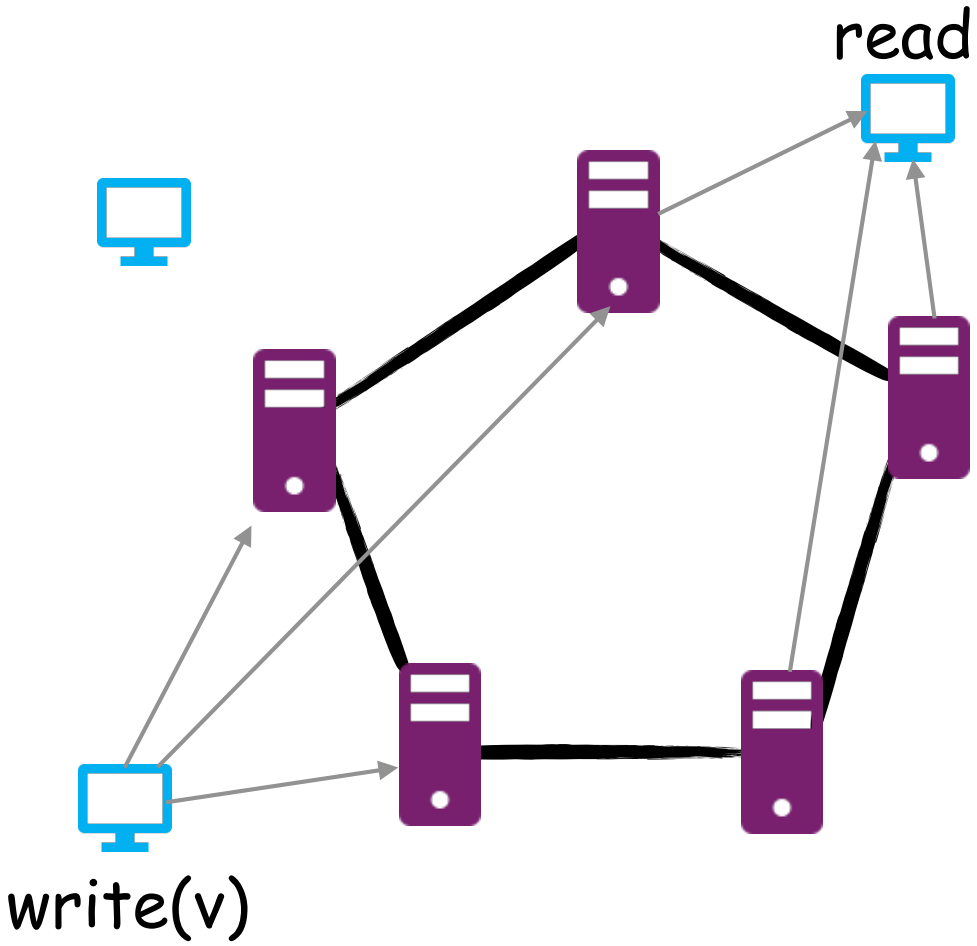
Proving Linearizability of Fault-Tolerant Register Protocols

Dependency Graph Approach

Gregory Chockler
University of Surrey

Alexey Gotsman (IMDEA), Sadegh Keshavarzi (U of Surrey),
Alejandro Naser-Pastoriza (IMDEA)

Fault-Tolerant Register Protocols

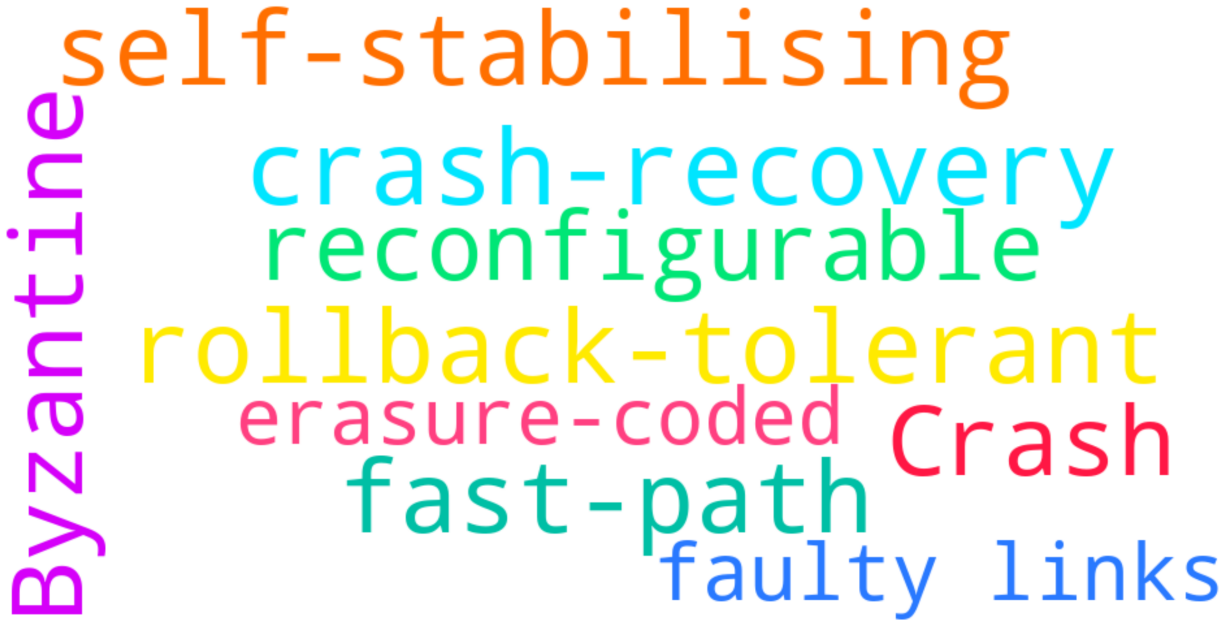


- Asynchronous message-passing system
- Fail-prone servers (replicas)
- Fail-prone clients interact with replicas to implement a R/W register abstraction

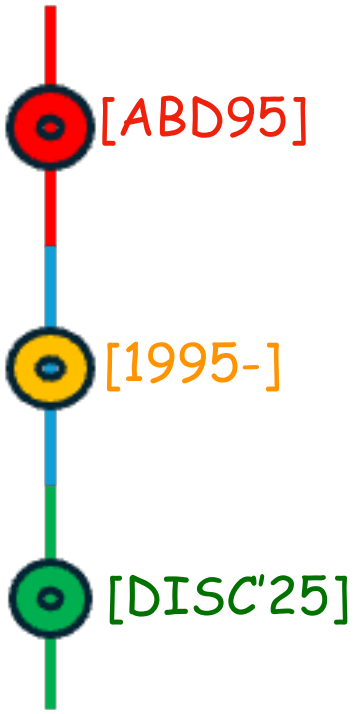
Important Abstraction

- Registers model key storage functionality

- *Sharing memory robustly in message-passing systems*, Attiya, Bar-Noy, Dolev, JACM'95



1000+ papers



This Talk

- New methodology for proving linearizability of multi-writer/multi-reader (MWMR) register implementations

  - Register implementation is linearizable IFF
    ∀ histories σ of read/write invocations and responses,
    ∃ permutation of σ (linearization) that complies with:

    (1) Real-time order of non-overlapping invocations in σ, and

    (2) Every read returns a value written by last preceding write

Motivation

- We have recently been working on MW register implementations extending ABD for new failure models
  - [Naser-Pastoriza, C, Gotsman, OPODIS'23, PODC'25], [Keshavarzi, C, Gotsman, DISC'25]

- We were looking for proof techniques to establish their linearizability

- Unexpectedly, this turned out to be a stumbling block…

It ain't easy…

- Textbook techniques do not work
  - Linearization point arguments
  - Forward simulations towards an atomic object
- "…there are no strongly-linearizable fault-tolerant message-passing implementations of multi-writer registers, max-registers, snapshots or counters" [Attiya, Enea, Welch, DISC'21]

It ain't easy…

- Find a partial order on the invocations and prove it satisfies certain properties
  - [Lemma 13.16, Lynch'96], [Lynch & Shvartsman, DISC'02]
- Still not easy to use
  - Some properties cannot be proven before partial order is found
  - Assumes all invocations are complete

It ain't easy…

- Capture partial order of [Lemma 13.16, Lynch'96] as an abstract automaton (PO Machine)
  - [C, Lynch, Mitra, Tauber, DISC'05]
- Prove MWMR ABD by forward simulation towards PO Machine
- Simulation relation turned out to be difficult to customise for more advanced register implementations

Our Approach

Happens-before, coherence, reads-from, from-read

- Flip the partial order approach on its head!

  (1) Express operation dependencies in terms of four standard binary relations from weak memory literature

  (2) Prove that the union of these relations (dependency graph) is a acyclic

- A simple and elegant linearizability proof of multi-writer/ multi-reader (MWMR) ABD protocol: “proof pearl”

Our Approach

Inspired by prior work on

- Early work on weak memory
  - [Shasha & Snir, ACM TOPLAS 1988]
- Transaction isolation specifications
  - [Adya's PhD thesis, 1999]
- Aspect-oriented linearizability proofs in shared memory
  - [Henzinger et al., CONCUR'13], [Dodds et al., POPL'15], [Domínguez & Nanevski, CONCUR'23]
- Transactional memory
  - [Khyzha, Attiya, Gotsman, Rinetzky, PPoPP'18]

Dependency Graph

Union of four relations for a given execution σ

(Assume first event in σ is write(v_0))

- Real-Time (rt)

  $\text{rt}(a, b) \iff a$ completes before b is invoked in σ

- Write-Write (ww)

  ww: is a total order on the writes in σ

- Write-Read (wr) ("reads-from")

  $\text{wr}(w, r) \iff$ read r returns the value written by write w

  $\text{wr}(w, r) \wedge \text{wr}(w', r) \implies w = w'$

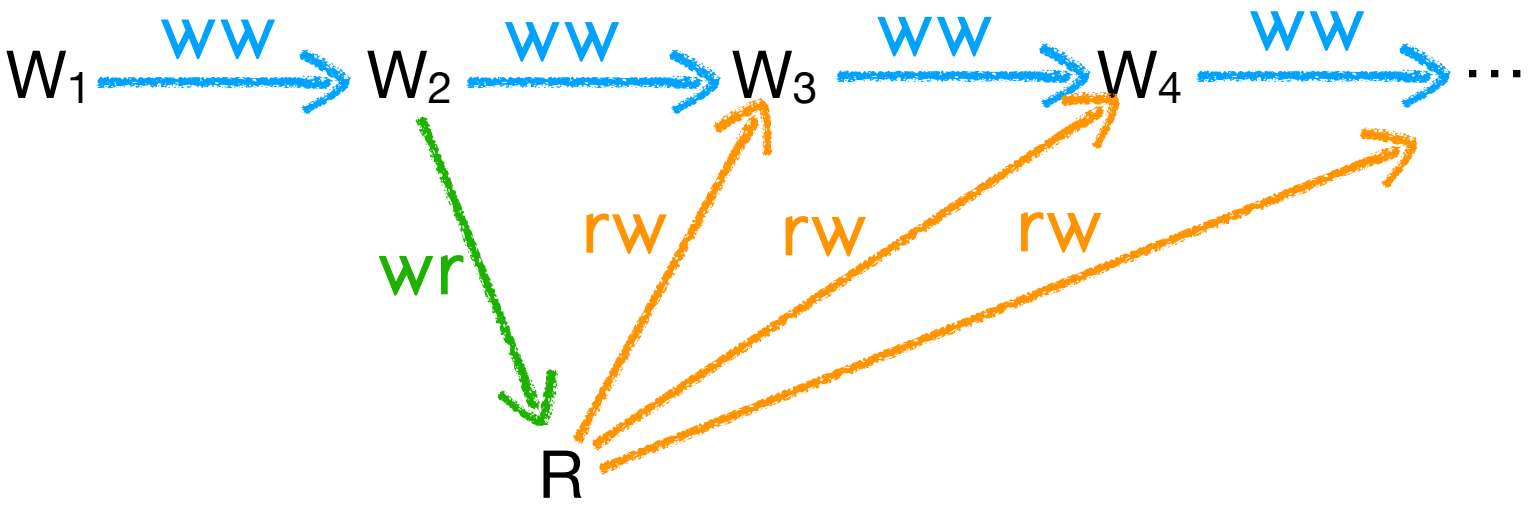
Dependency Graph

Union of four relations for a given execution σ

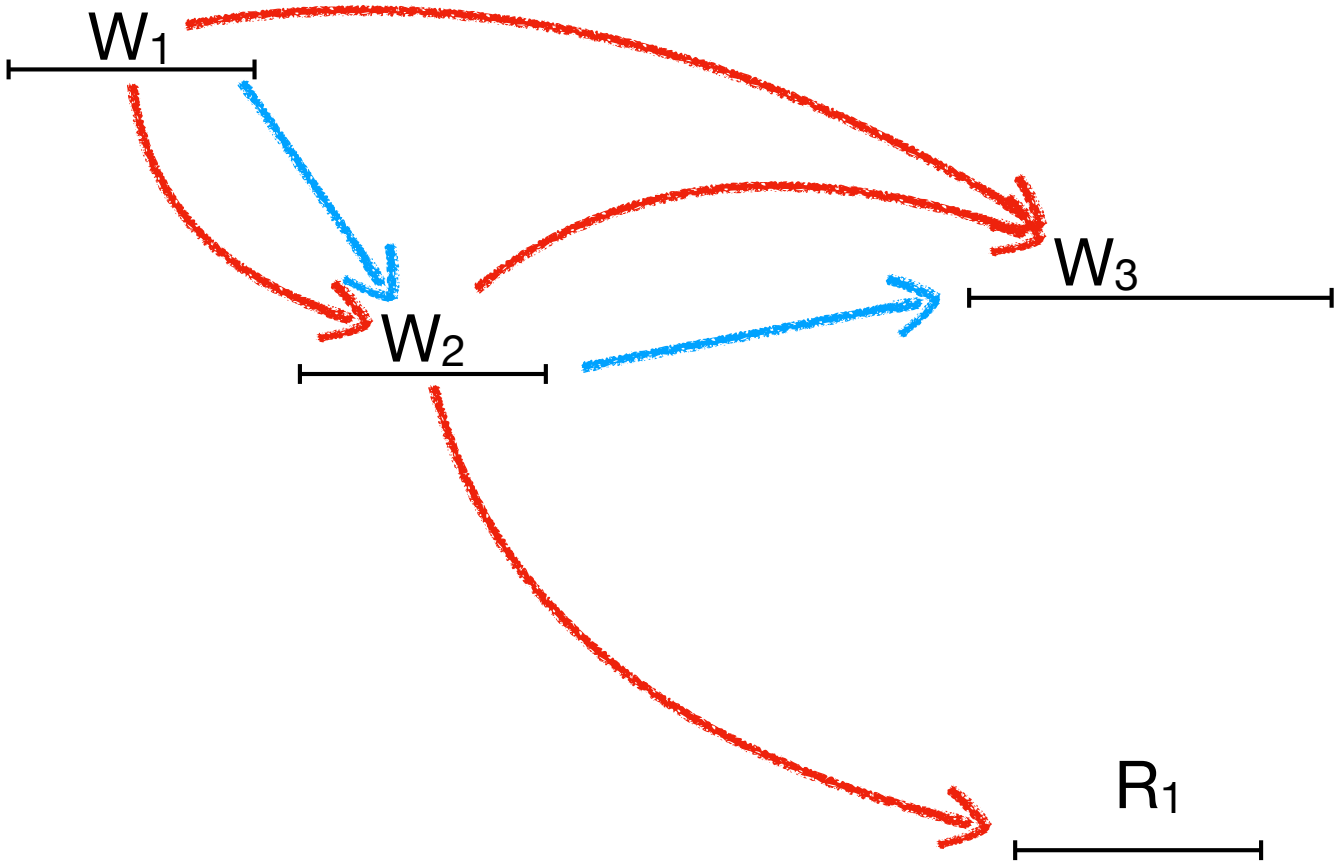
(Assume first event in σ is write(v_0))

- Read-Write (rw) ("from-read")

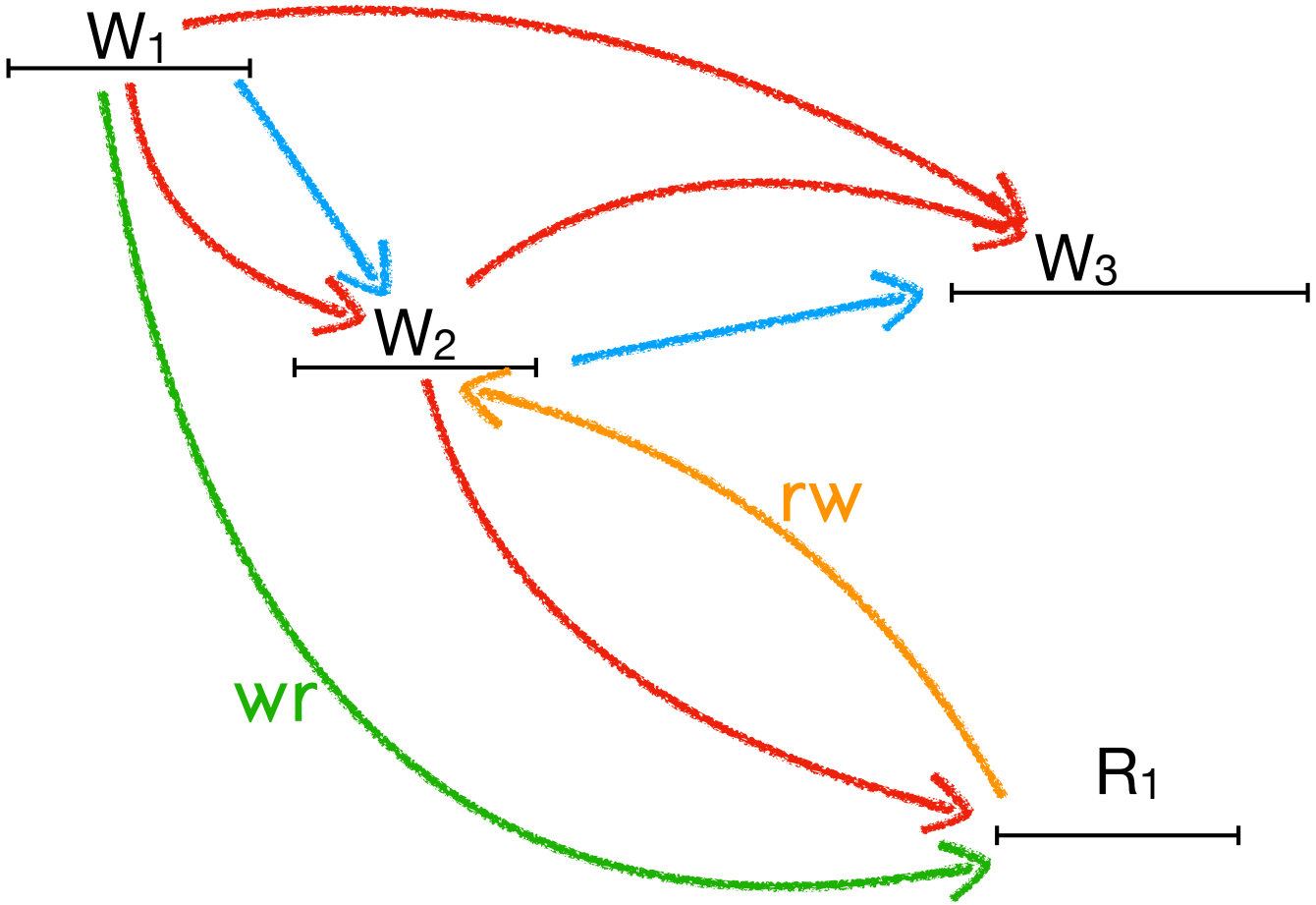
$\text{rw}(r, w) \iff$ read r reads-from a write preceding w in ww



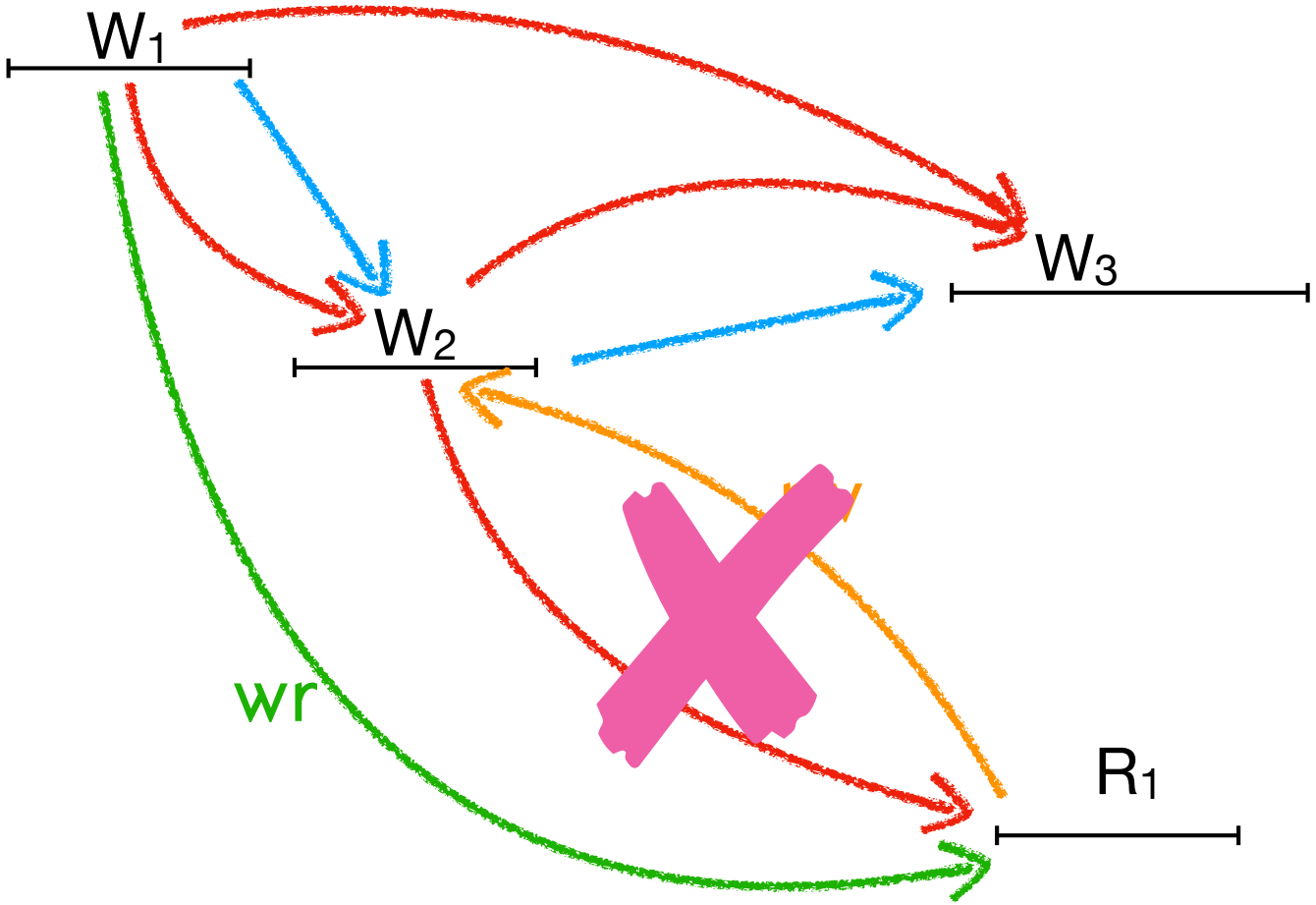
Dependency Graph



Dependency Graph

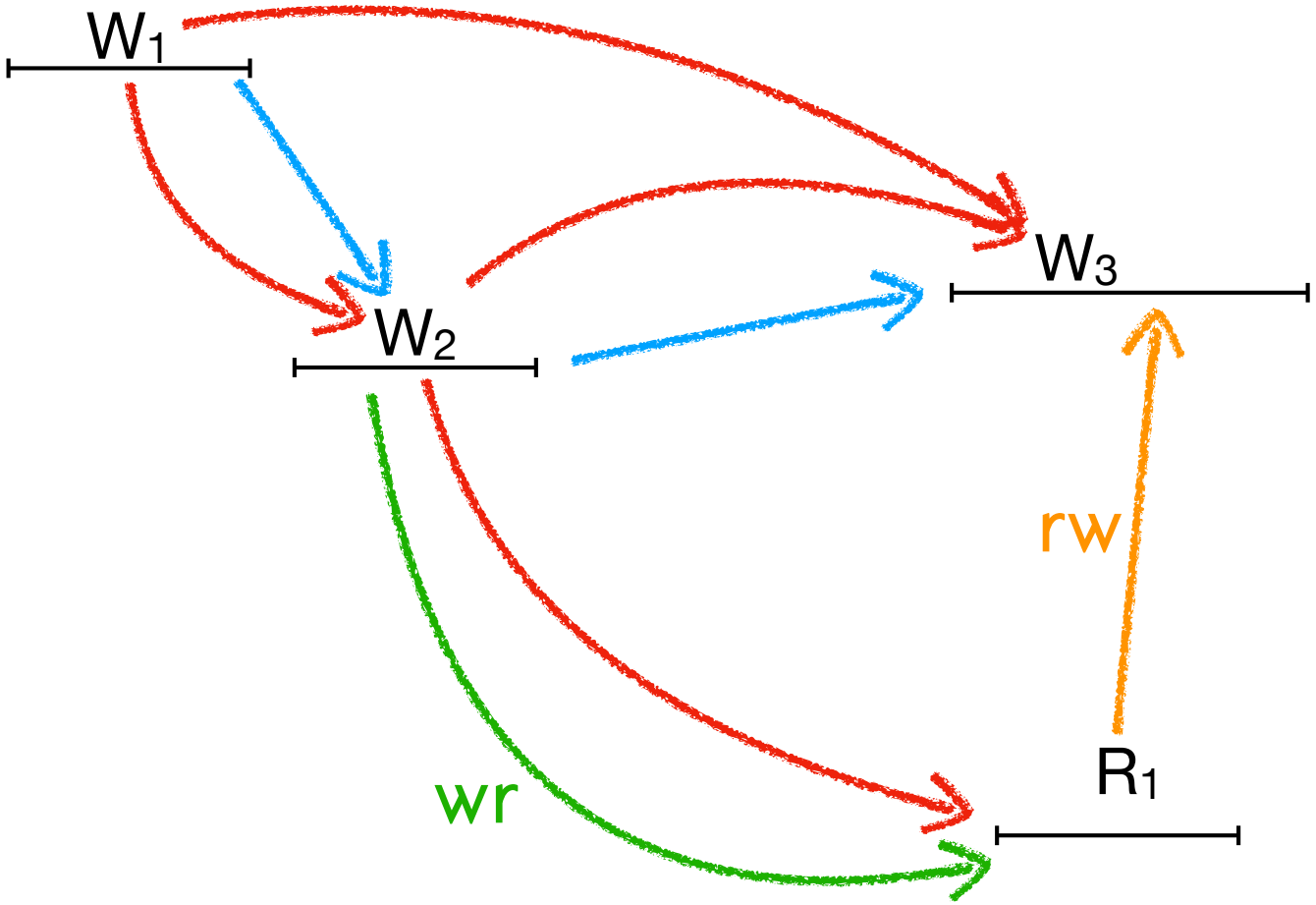


Dependency Graph



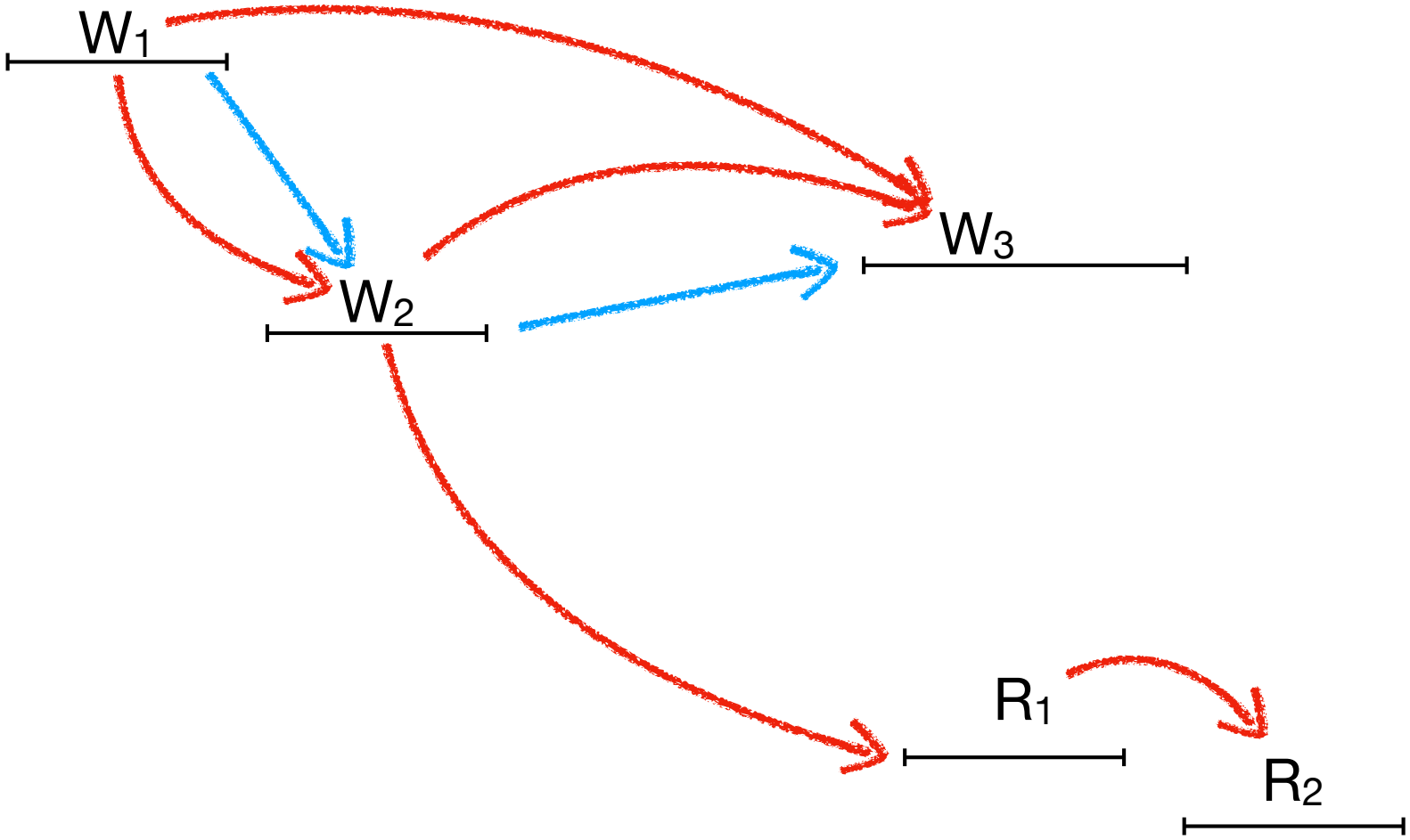
Cannot fix the cycle — not linearizable!

Dependency Graph

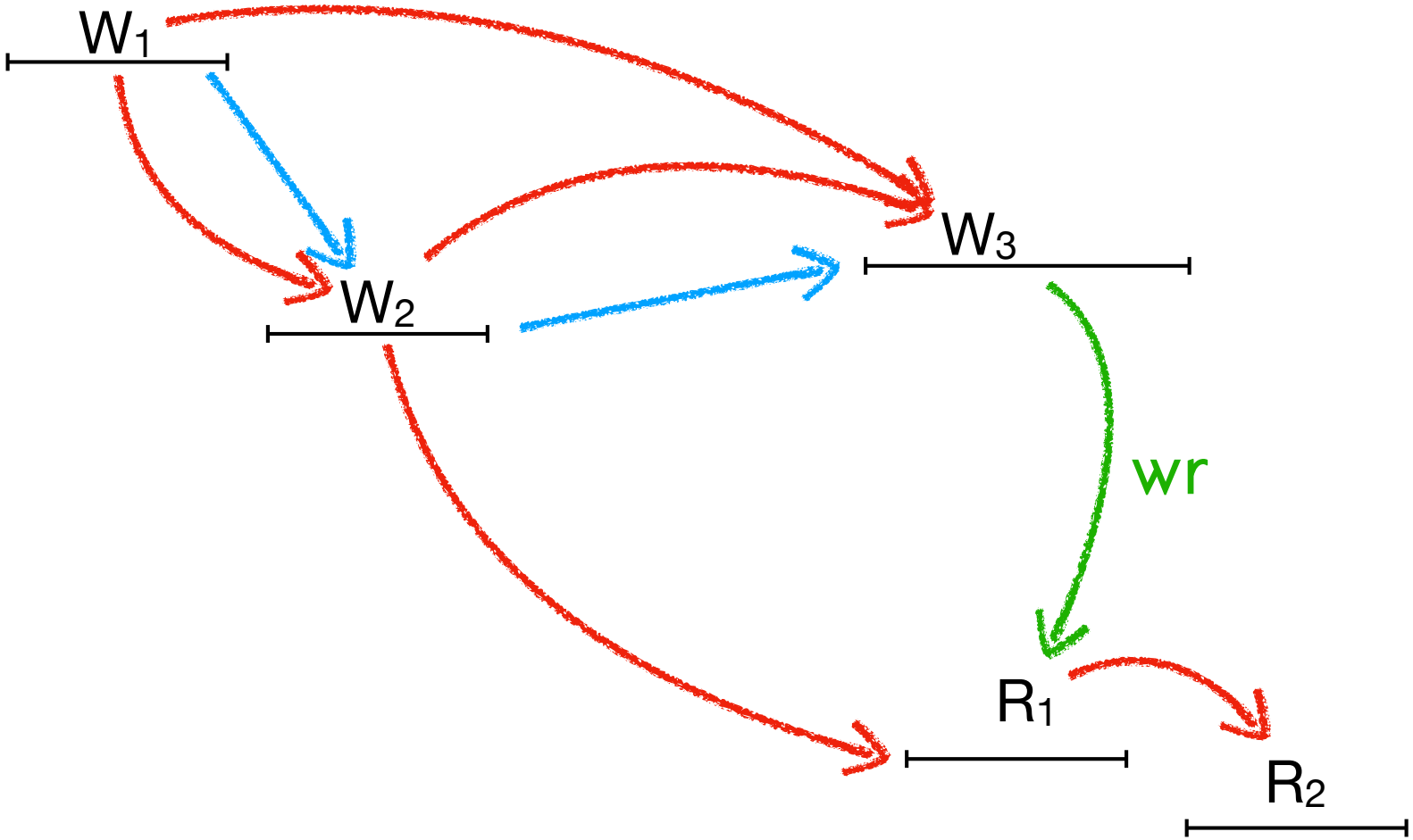


No cycles — linearizable!

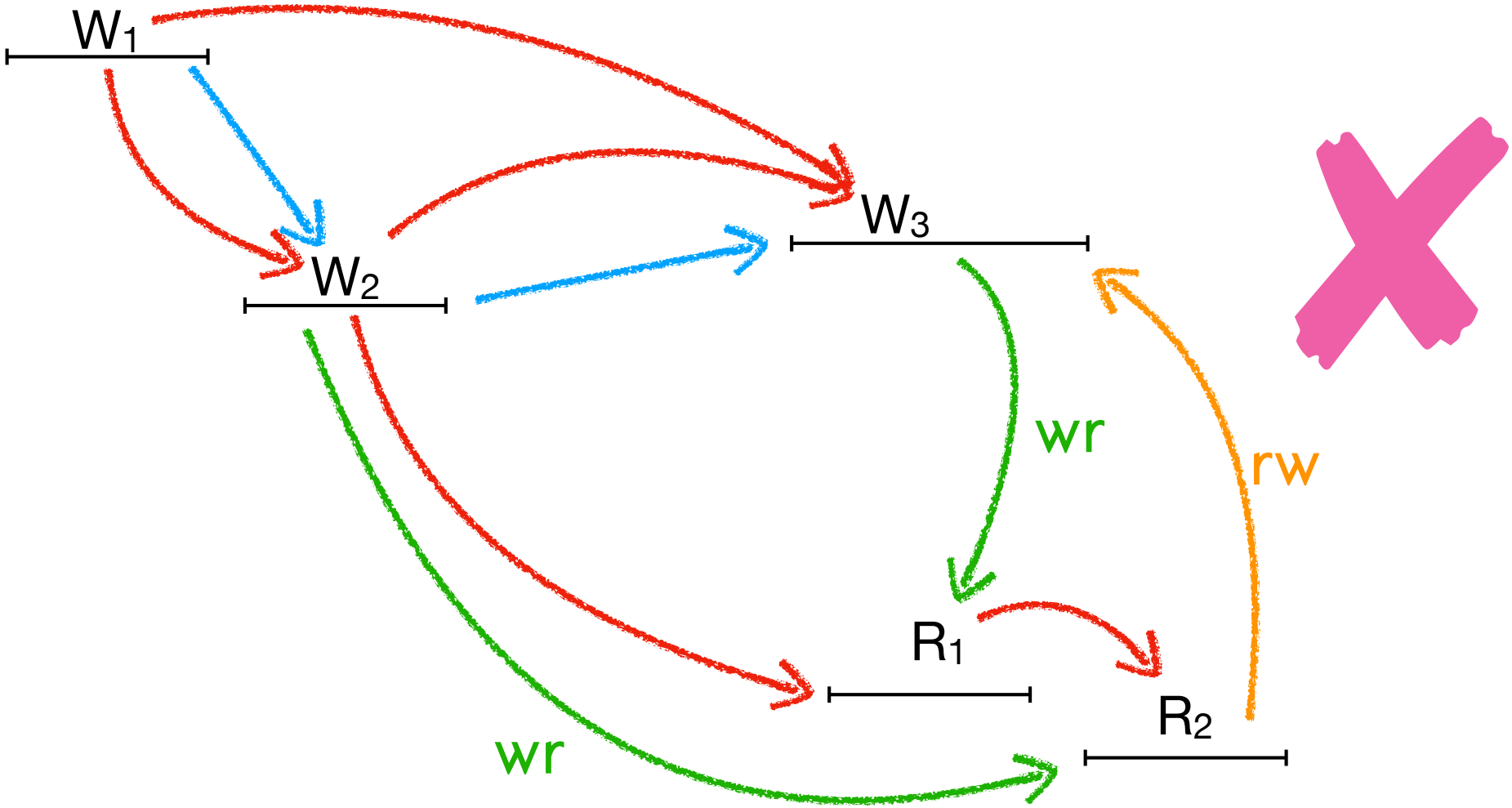
Dependency Graph



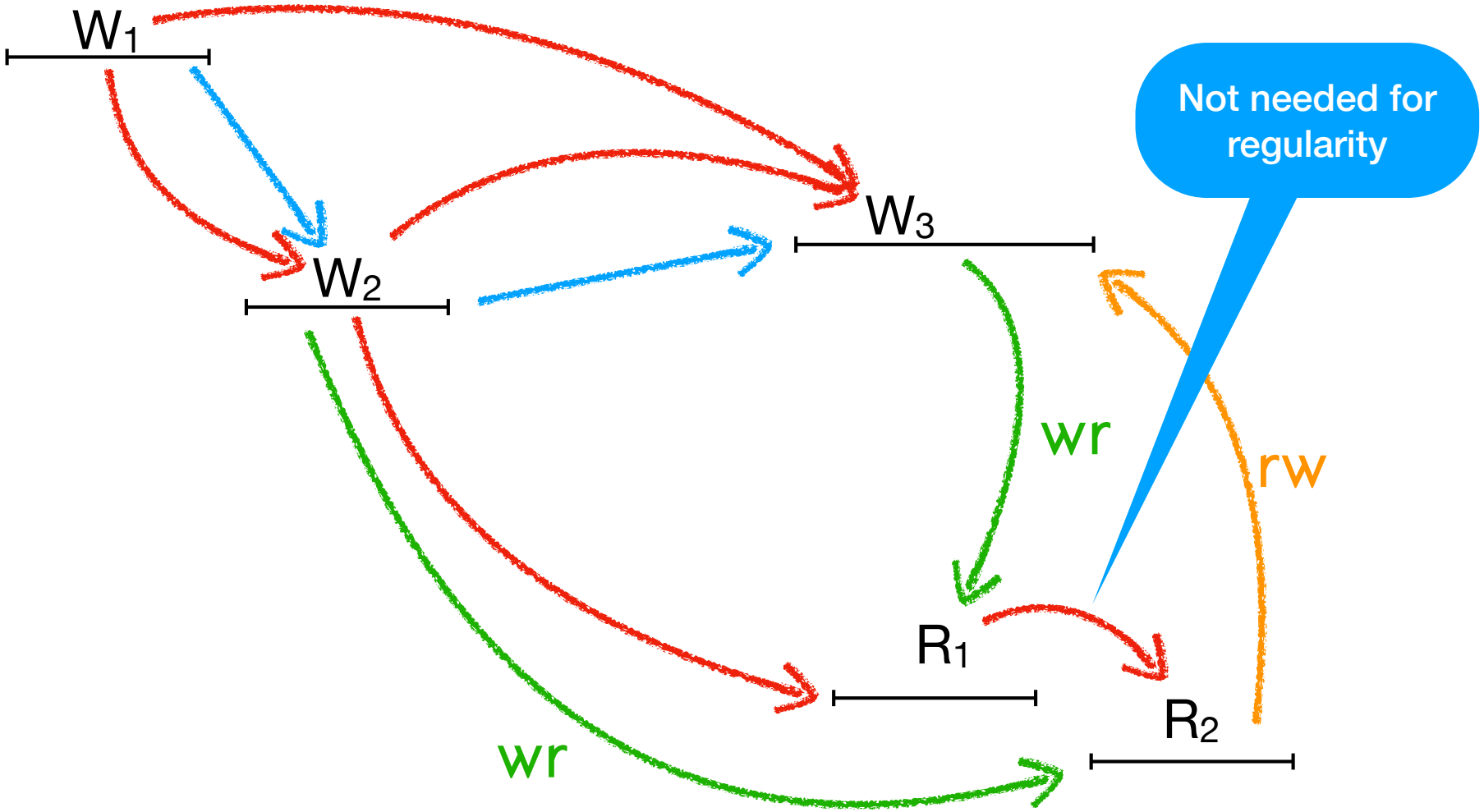
Dependency Graph



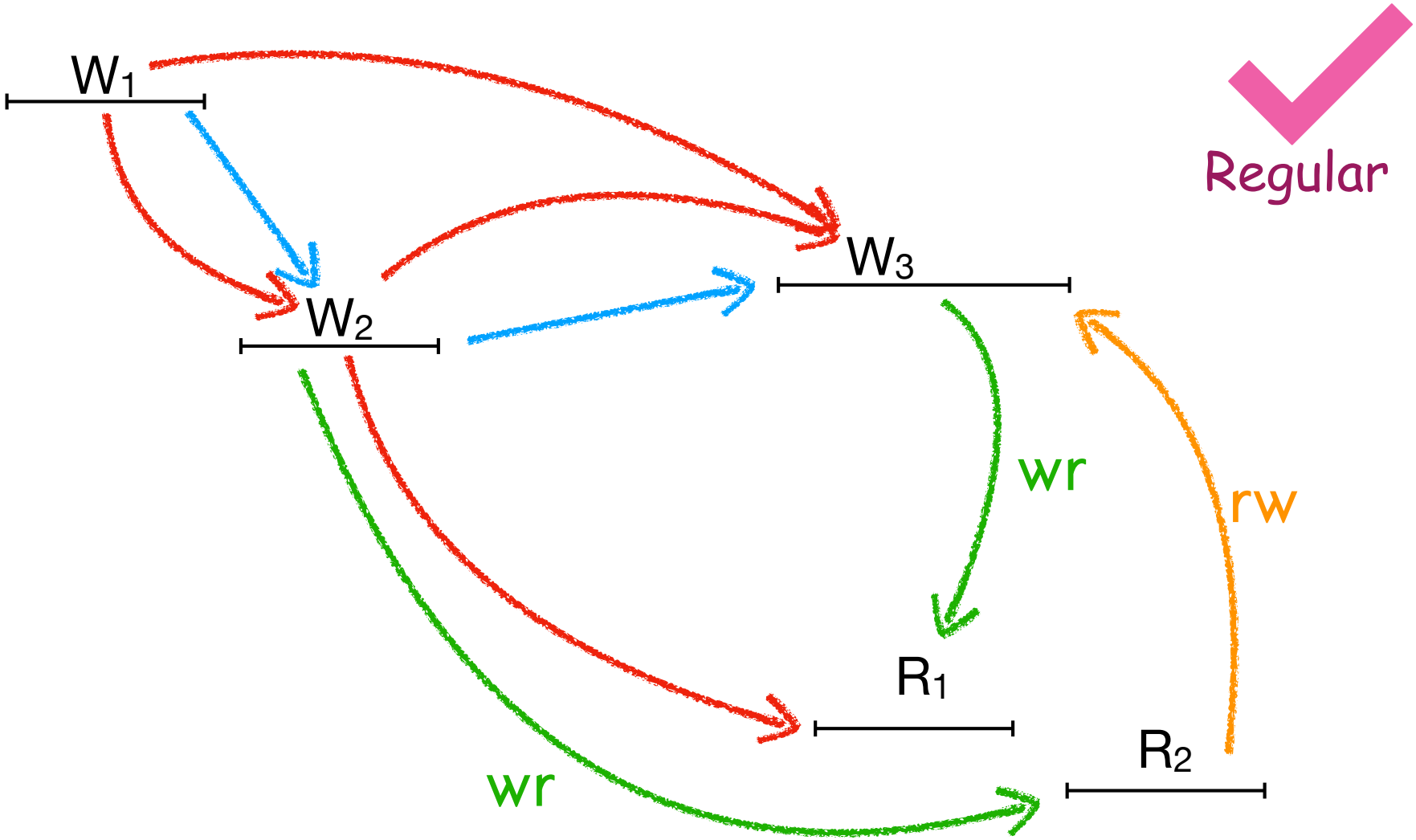
Dependency Graph



Dependency Graph



Dependency Graph



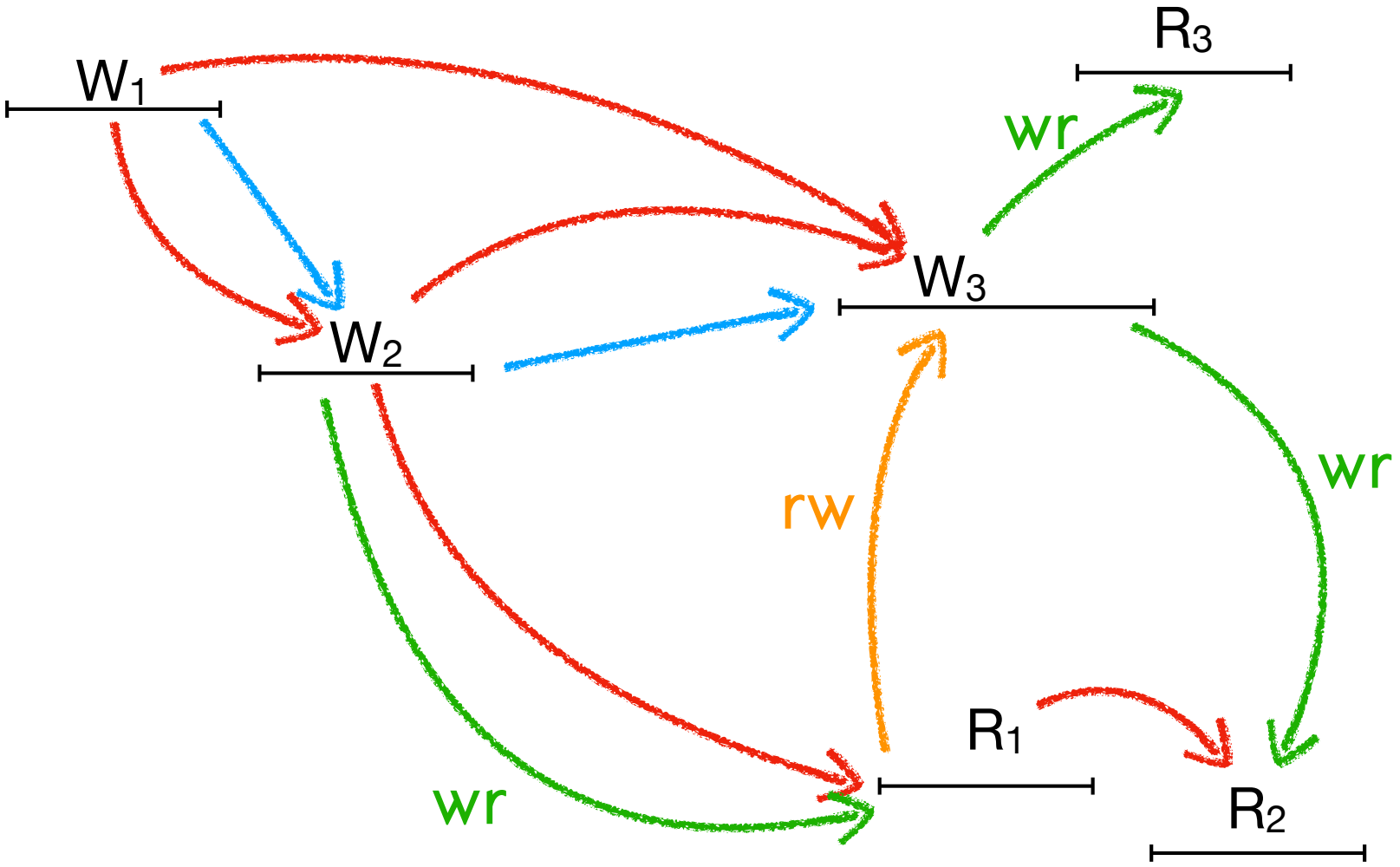
Linearizability Theorem

- An execution is linearizable **IFF** ∃ rt, ww, wr, rw such that the graph (rt ∪ ww ∪ wr ∪ rw) is acyclic

- An execution is regular **IFF** ∃ rt^{-}, ww, wr, rw such that the graph (rt ∪ ww ∪ wr ∪ rw) is acyclic

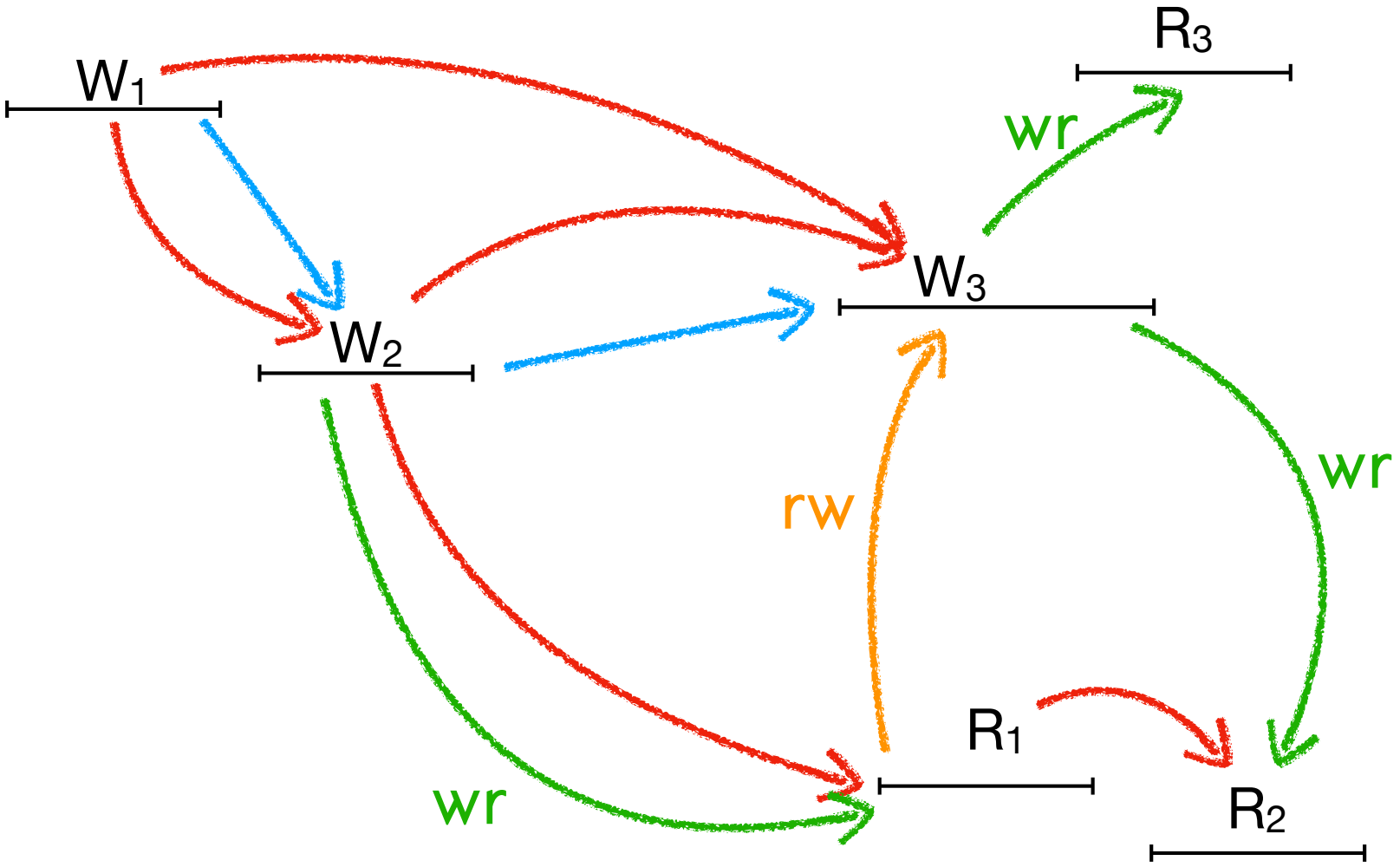
  - rt^{-}: is a restriction of rt excluding read-read pairs

Proof: If



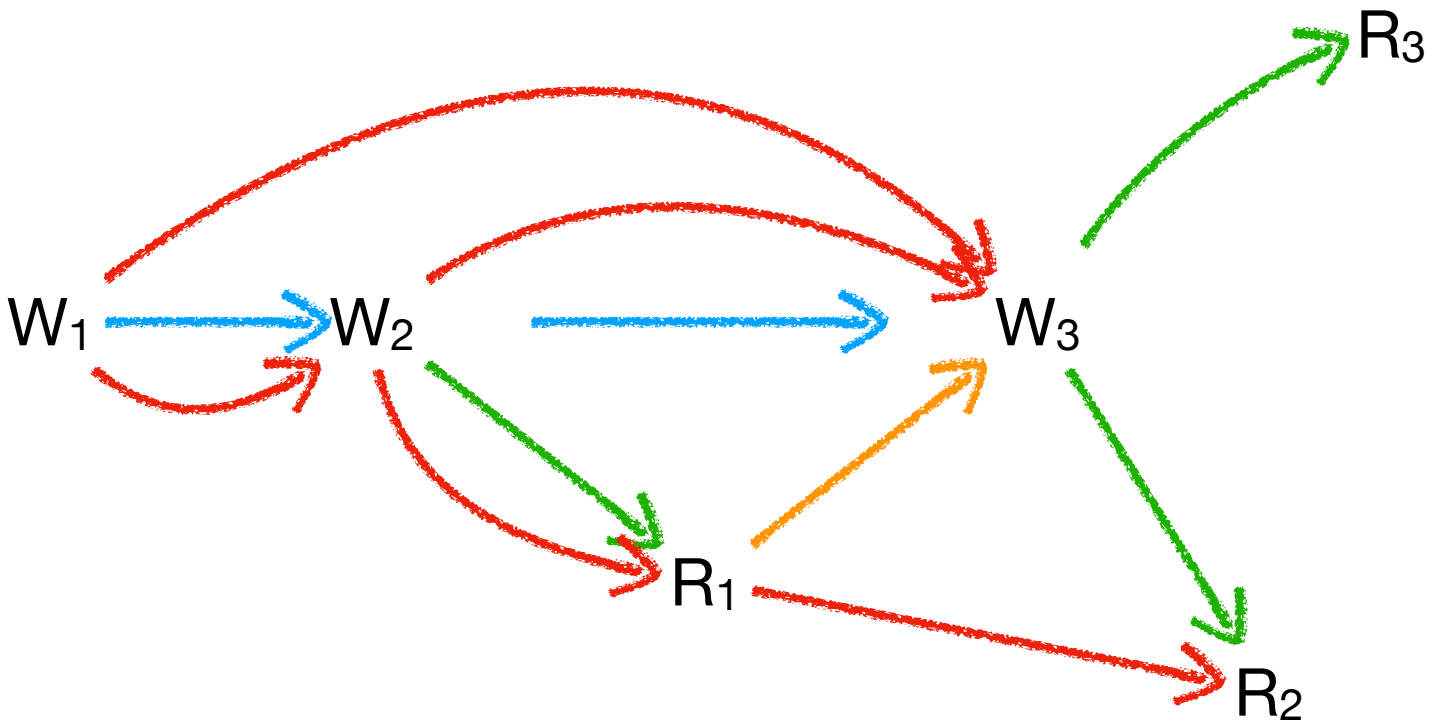
Acyclic dependency graph

Proof: If



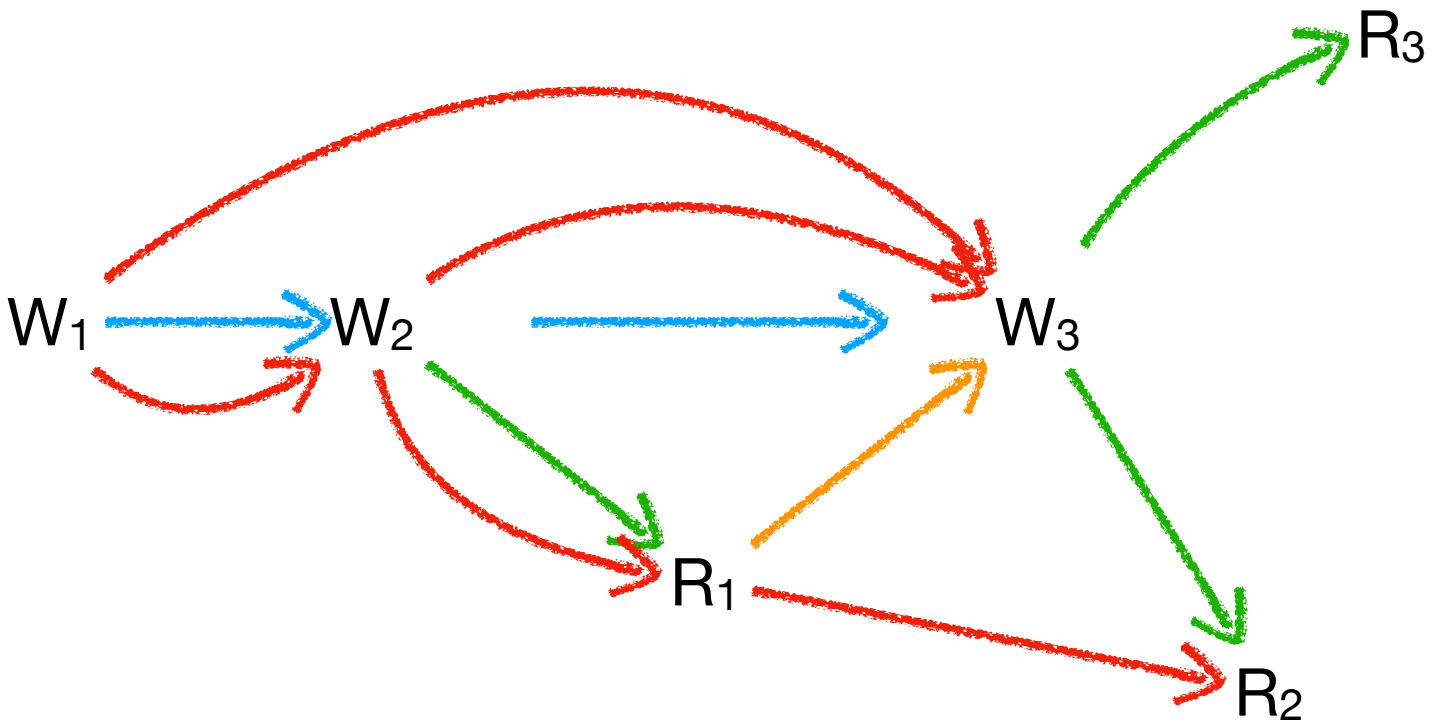
Acyclic dependency graph induces a partial order

Proof: If



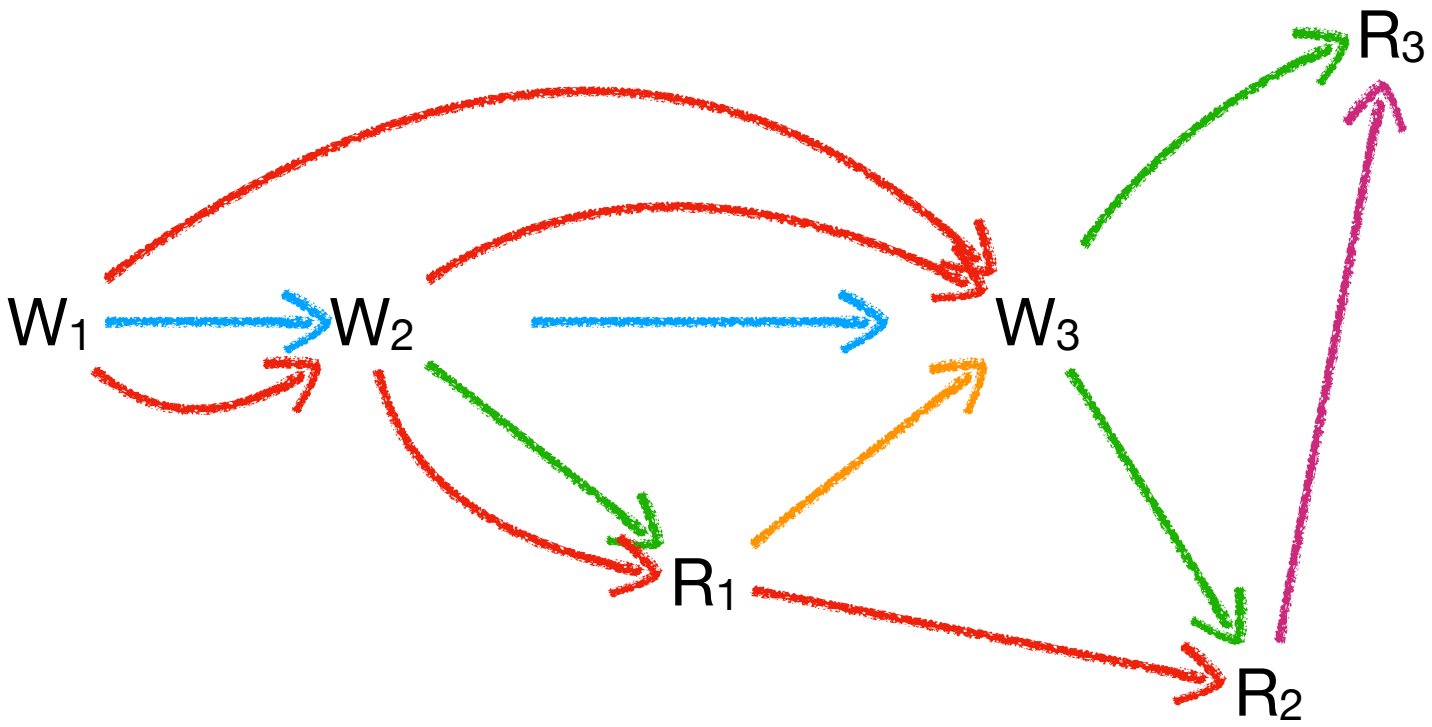
Acyclic dependency graph induces a partial order

Proof: If



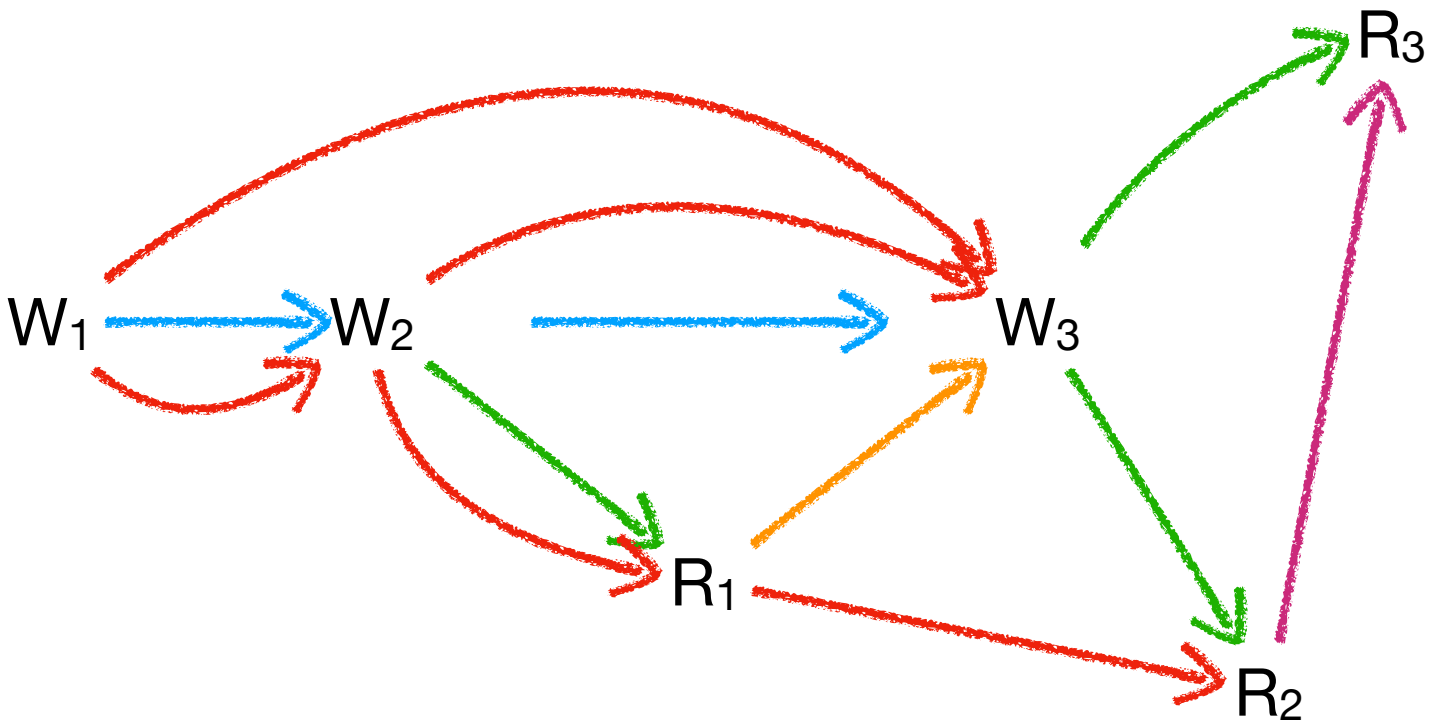
…that can be extended into a total order

Proof: If



…that can be extended into a total order

Proof: If



…which is a linearization

MWMR ABD

- $n \geq 2f + 1$ replicas, f can crash
- Read/write quorum system
  - $\mathscr{R} = \{Q \subseteq P \ : \ |Q| = f + 1\}, \mathscr{W} = \{Q \subseteq P \ : \ |Q| = n - f\}$
  - $\forall Q_r \in \mathscr{R} \,.\, \forall Q_w \in \mathscr{W} \,.\, Q_r \cap Q_W \neq \emptyset$
  - Some $Q_r \in \mathscr{R}$, $Q_w \in \mathscr{W}$ are available in every execution

MWMR ABD: Client q_i

Write(v):

$S \leftarrow$ read_quorum(TS)

$c \leftarrow \max\{c'_i \mid (c'_i, \_) \in S\}$

$ts \leftarrow (c + 1,\ i)$

$S \leftarrow$ write_quorum(TSVal(ts, v))

Read:

$S \leftarrow$ read_quorum(TSVal)

Let (ts, v) be such that $(ts, v) \in S \wedge$
$ts = \max\{ts' \mid (ts', \_) \in S)\}$

$S \leftarrow$ write_quorum(TSVal(ts, v))

return v

write_quorum(req):

send WRITE (req) to P

Wait until received

$\{$WRITE_ACK(j) $\mid p_j \in Q\} \wedge Q \in \mathscr{W}$

read_quorum(req):

send READ (req) to P

Wait until received

$\{$READ_ACK(r_j, j) $\mid p_j \in Q\} \wedge Q \in \mathscr{R}$

return $\{r_j \mid p_j \in Q\}$

MWMR ABD: Replica p_i

On WRITE ($\texttt{TSVal}(ts, v)$) from q_j

if $ts >$ ts then

$(\text{ts}, \text{val}) \leftarrow (ts, v)$

send WRITE_ACK(i) to q_j

On READ (req) from q_j

$r \leftarrow$ case req do

$\texttt{TS}$: ts

$\texttt{TSVal}$: (ts, val)

send READ_ACK(r, i) to q_j

MWMR ABD is Linearizable

Write(v):

$S \leftarrow$ read_quorum(TS)

$c \leftarrow \max\{c'_i \mid (c'_i, \_) \in S\}$

$ts \leftarrow (c + 1,\ i)$

$S \leftarrow$ write_quorum(TSVal(ts, v))

Timestamp of a write invocation

Read:

$S \leftarrow$ read_quorum(TSVal)

Let (ts, v) be such that $(ts, v) \in S \wedge$
$ts = \max\{ts' \mid (ts', \_) \in S)\}$

$S \leftarrow$ write_quorum(TSVal(ts, v))

return v

Timestamp of a read invocation

TS: (read/write invocations) $\rightarrow \mathbb{N}^{\geq 0}$

MWMR ABD is Linearizable

- $(W, W') \in \text{ww} \iff TS(W) < TS(W')$
  - Timestamp uniqueness $\implies$ ww is a total order
- $(W, R) \in \text{wr} \iff TS(R) = TS(W)$
  - Every read R returns the value with timestamp $TS(R)$ written by a write W such that $TS(W) = TS(R)$

MWMR ABD is Linearizable

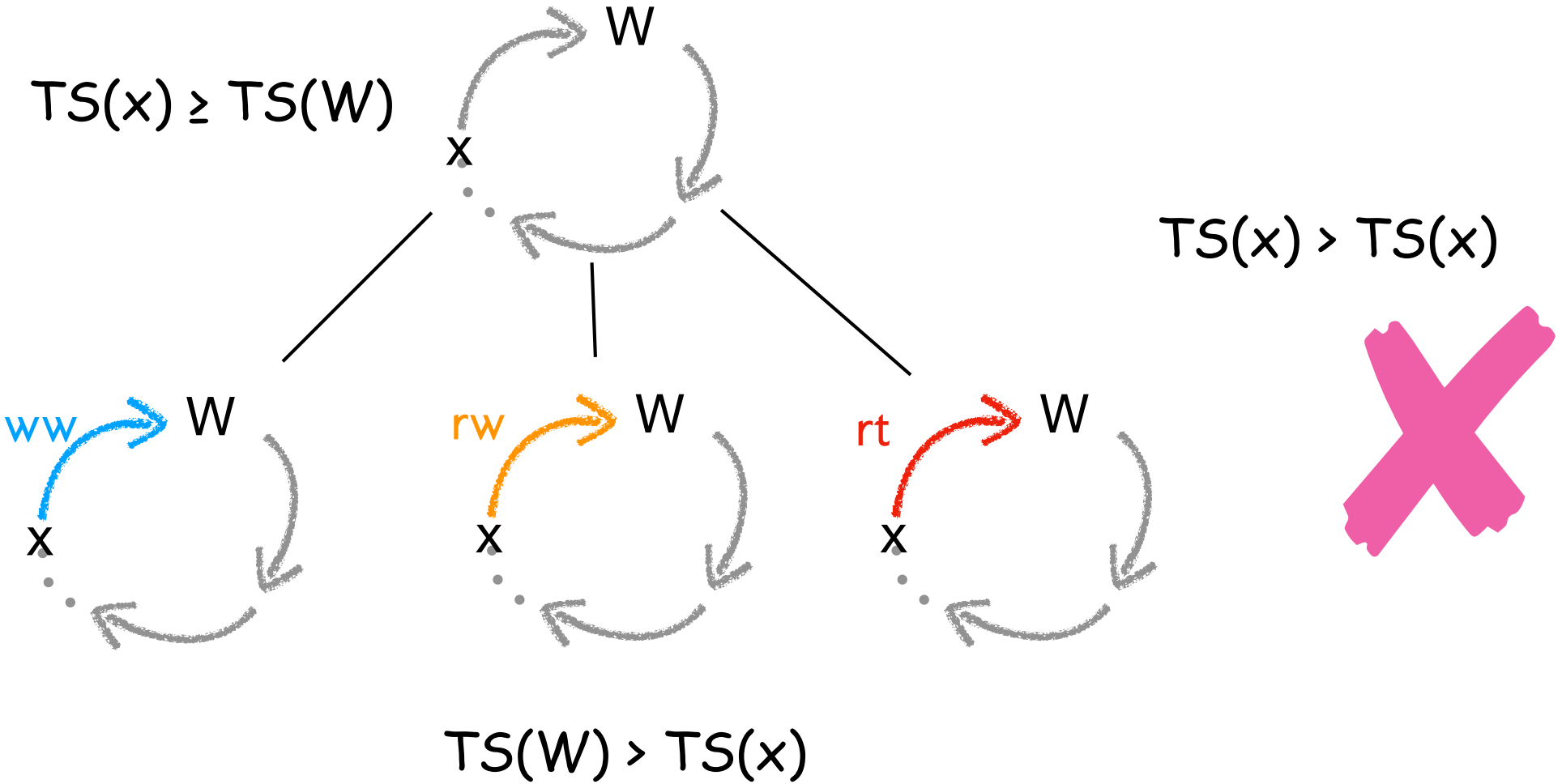
- $(R, W) \in \mathsf{rw} \iff TS(R) < TS(W)$
  - $(R, W) \in \mathsf{rw} \iff \exists W' . (W', R) \in \mathsf{wr} \wedge (W', W) \in \mathsf{ww}$
    $\iff TS(R) = TS(W') < TS(W)$
- By quorum intersection, update rule, write-back:
  - $(x, W) \in \mathsf{rt} \iff TS(x) < TS(W)$
  - $(x, R) \in \mathsf{rt} \iff TS(x) \leq TS(R)$

MWMR ABD is Linearizable

- σ: an execution of MWMR ABD
- G: a dependency graph for σ constructed as above
- Suppose that G has a cycle C
- Then, C only includes reads

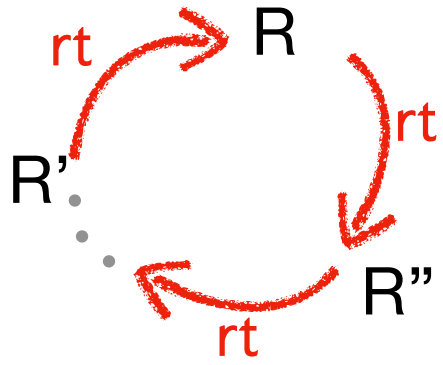
MWMR ABD is Linearizable

- Then, C only includes reads



MWMR ABD is Linearizable

- Then, C only includes reads
- Then, all edges in C are rt edges



R has started after R has finished

MWMR ABD is Linearizable

- σ: an execution of MWMR ABD
- G: a dependency graph for σ constructed as above
- Suppose that G has a cycle C
- Then, C only includes read vertices and rt edges
- Then, some read starts after it finishes. A contradiction.
- G is acyclic
- σ is linearizable

Conclusions

- Proof pearl of linearizability for message-passing implementations of MWMR registers
- Easy generalisation for incomplete operations via visibility predicate
  - [Khyzha et al., PPoPP'18], [Keshavarzi, C, Gotsman, DISC'25]
- Linearizability theorem was proven for infinite executions
- Challenge for automated verification
  - Need to reason about whole traces, rather than inductively