SPP (Synchro et Prog Parallèle)

Unit 9: Atomicity

François Taïani

Motivation

- We have use the word "atomic" quite a few times
  - ➔ "this operation is (is not) atomic"
  - ➔ "this sequence of operation needs to be atomic"

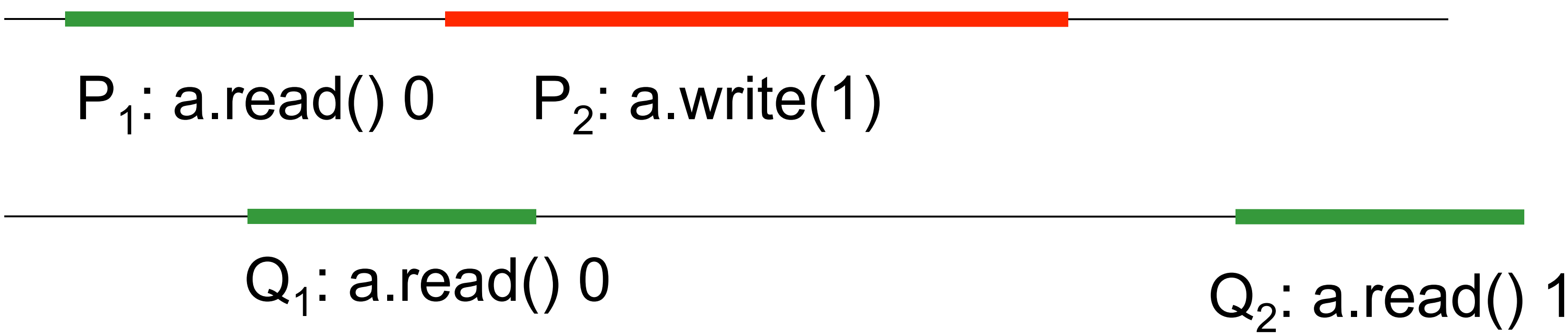
- What does this mean exactly?
  - ➔ Etymology: Ancient Greek ἄ - τομος (cannot be cut)
  - ➔ Can this be captured formally?

Case 1: Read / Write Objects

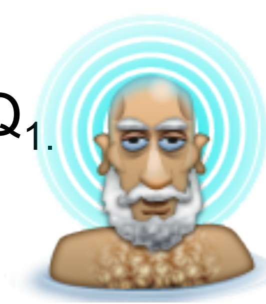
- System model
  - read / write objects: 2 operations read() and write(..)
  - operations invoked by processes, run concurrently
  - an execution captured by an history
- History
  - a **sequence** of events
  - two types of events: invocations, and return events
- invocations labelled with
  - invoker, invoked object, parameter passed
- return events pair-wise associated with an invocation
  - at most one return event per invocation

Example

- One shared object, two processes

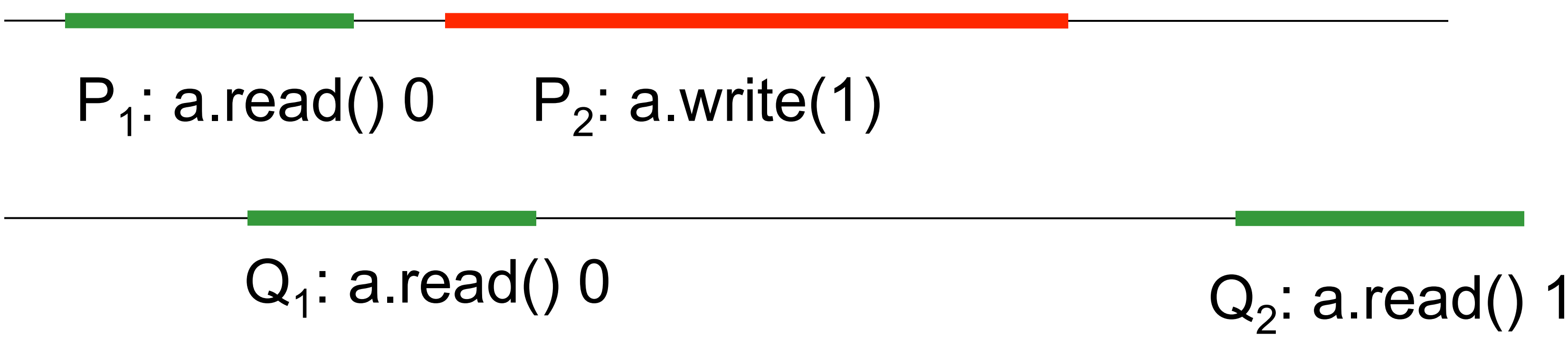


  - ➔ history made of 8 events, 4 operation executions
  - ➔ History: P_1^{start}, Q_1^{start}, P_1^{end}, P_2^{start},Q_1^{end},P_2^{end},Q_2^{start},Q_2^{end}
- Order exists between operations
  - ➔ P_2 happens after P_1. Q_2 happens after P_1, P_2, Q_1.
  - ➔ what about Q_1?
  - ➔ How would you formally define this order?



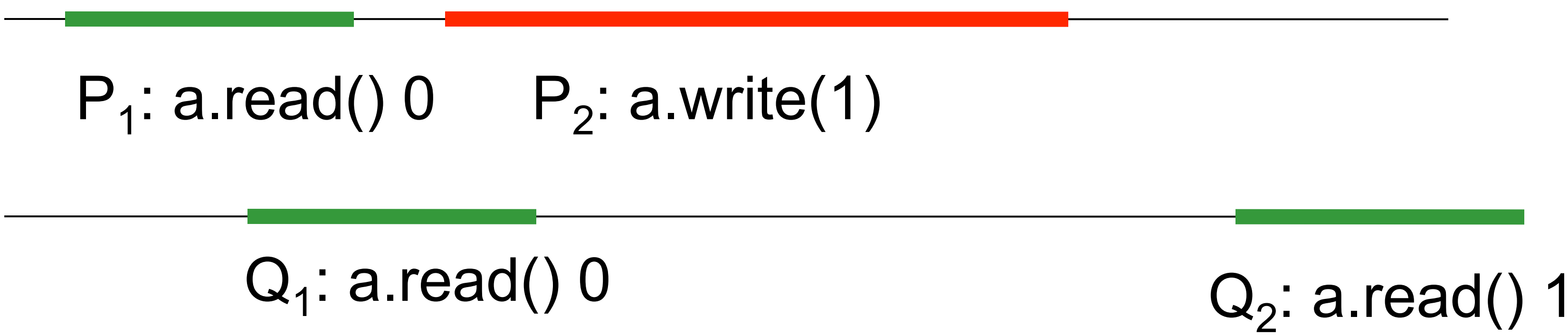
Precedence Order

- A history H induce a partial order $<_H$ on its operations
  - ➔ $op_1 <_H op_2$ iff op_1^{end} appears before op_2^{start} in H
  - ➔ Quiz: Draw the graph of $<_H$ for the previous example



Sequential History

- A history H is sequential iff
  - ➔ every op_x^{start} is immediately followed a mathing op_x^{end}
  - ➔ "every invocation is followed by its result"
- Quiz
  - ➔ Is the following history sequential? Why?



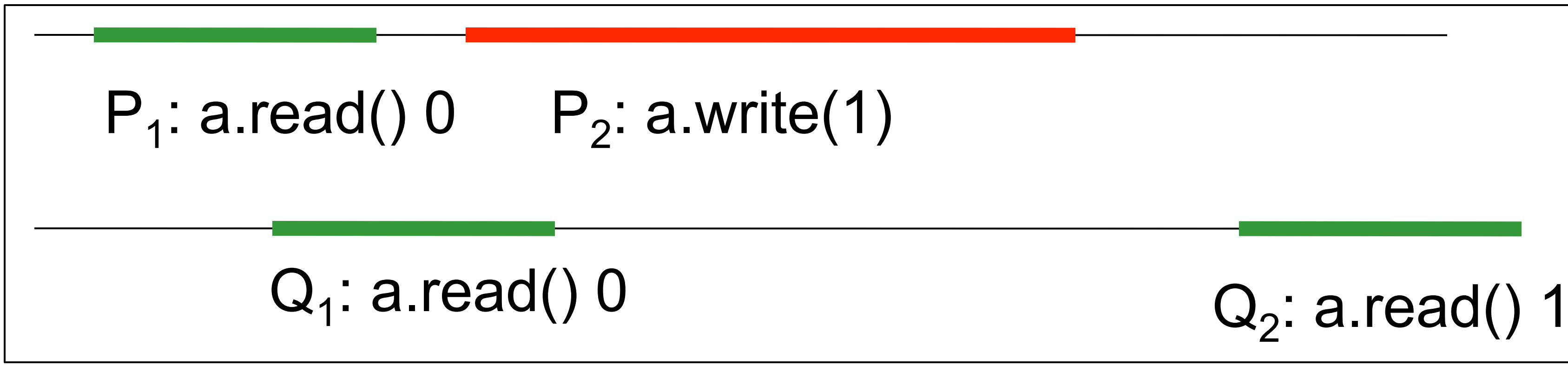
  - ➔ If H is sequential, how is $<_H$?

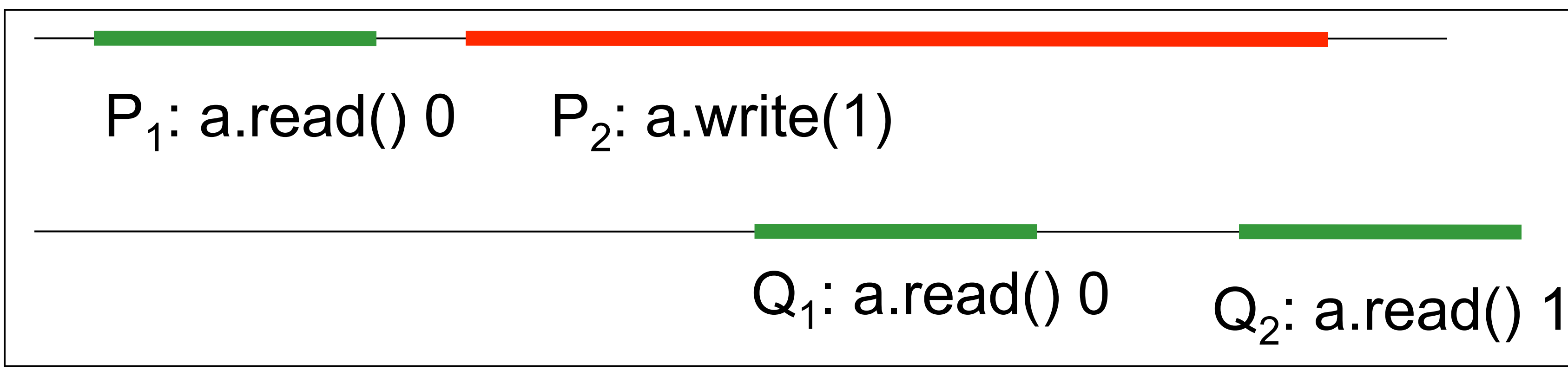
Sub-Histories and Equivalence

- A Sub-History captures the local view of an execution
- Process sub-history H | P
  - → only keep events local to P
- Object sub-history H | a
  - → only keep events happening on a
- Quiz: Write the sub-histories H | P and H | Q for
  - → $H = P_1^{start}, Q_1^{start}, P_1^{end}, P_2^{start}, Q_1^{end}, P_2^{end}, Q_2^{start}, Q_2^{end}$
- 2 histories H_1 and H_2 are equivalent iff
  - → for all processes P: $H_1 | P = H_2 | P$

Equivalence: Quiz

- Are the 2 following histories equivalent?



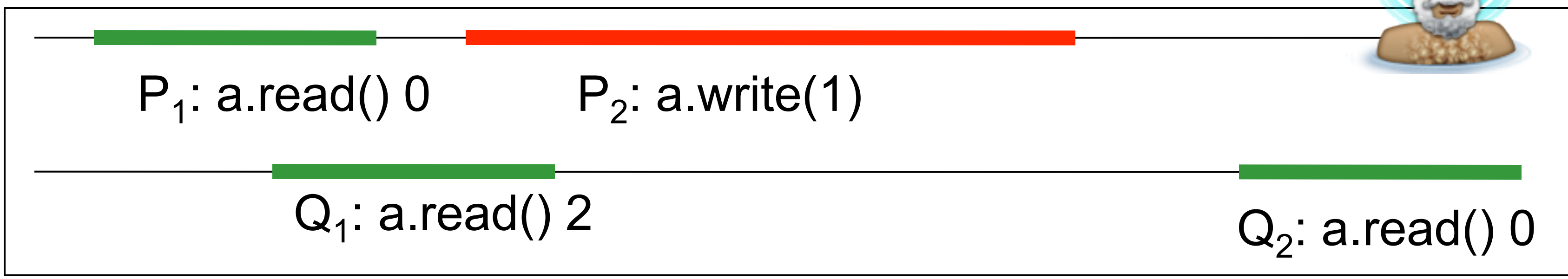


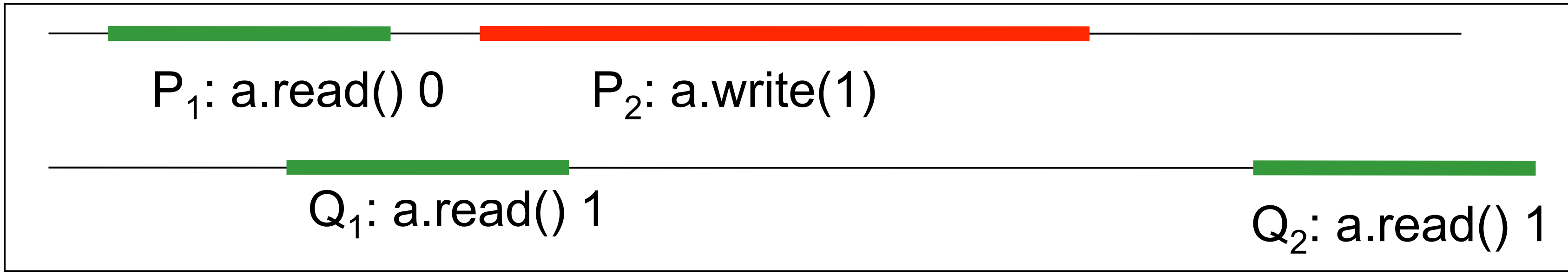
Acceptable Histories

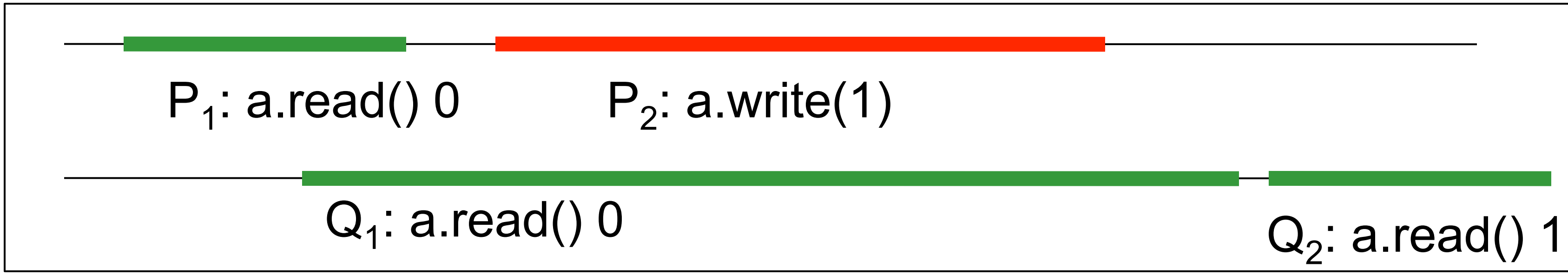
- What we have seen so far
  - ➔ histories
  - ➔ the order they imply on the ops they contain
  - ➔ special case: sequential histories
  - ➔ comparing histories: equivalence (use sub-histories)
- Our aim: capturing atomicity
  - ➔ i.e. defining how an “atomic” object should behave
  - ➔ i.e. defining which behaviour is acceptable, which is not
  - ➔ i.e. defining which histories are acceptable

Acceptable Histories: Intuition

- Are the following histories acceptable?







How to capture acceptability?

- Which histories are acceptable? Which are not?

- First step: focus on **acceptable sequential histories**
  - ➔ no concurrency
  - ➔ aim: capture sequential specification of the object
  - ➔ for R/W O: “reads return value of most recent earlier write”
  - ➔ define set of all “acceptable/legal” sequential histories

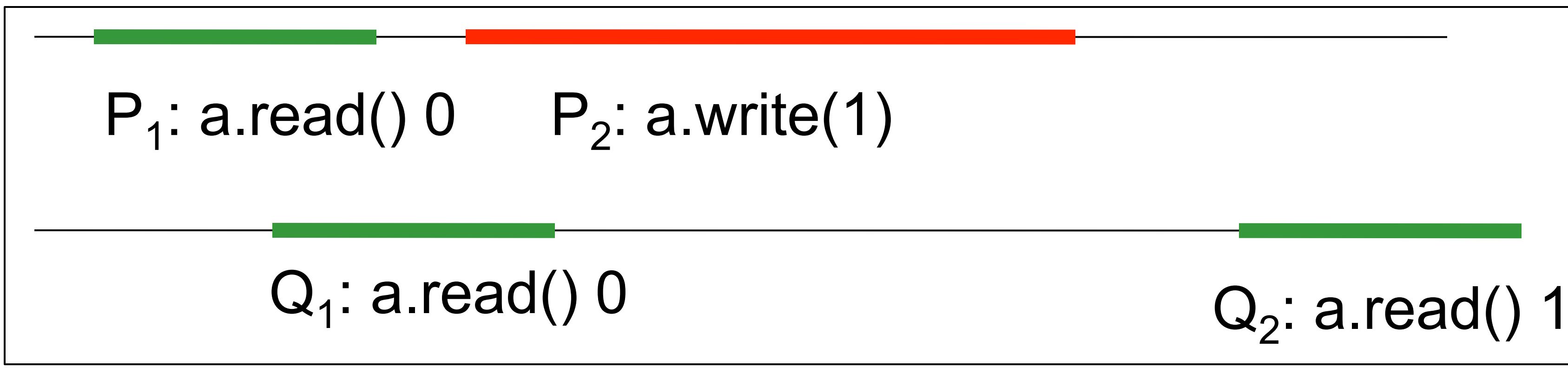
- Second step: define **acceptable concurrent histories**
  - ➔ using acceptable sequential histories as reference
  - ➔ different ways to do this: different semantics of consistency!

Atomicity

- History H is atomic iff
  - ➔ H can be extended into H' by adding (optional) return events
  - ➔ $\exists$ acceptable sequential history S, so that S equivalent H'
  - ➔ $<_H \subseteq <_S$
- Comments
  - ➔ H' needed to handle pending operations
  - ➔ S eq. H' : process cannot distinguish between S and H'
  - ➔ $<_H \subseteq <_S$: S cannot reorder events from H
- **Atomic Object: only accepts atomic histories**
- Also called "Linearizability" (Herlihy & Wing, 1990)

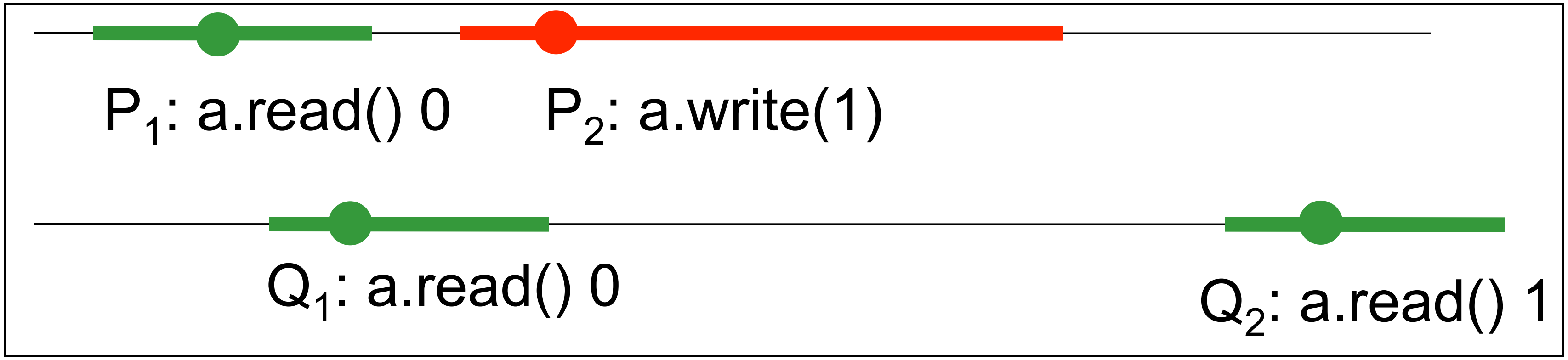
Example

- Find a history S that shows that the following is atomic
  - ➔ reminder 1: S should be sequential and acceptable
  - ➔ reminder 2: S should be equivalent to the history below



Example

- Interpretation
  - ➔ possible to find a point in each interval
  - ➔ so that resulting sequential history is acceptable



- ➔ "Points of Linearizibility"

Case 2: Generalization

- Previous definitions generalizable beyond R/W object
  - ➔ just need to redefine acceptable sequential histories
- Example: queue: sequential specification
  - ➔ #(dequeue) ≤ #(enqueue)
  - ➔ op $dequeue_k$ return value passed by $enqueue_k$
- All other definitions follow

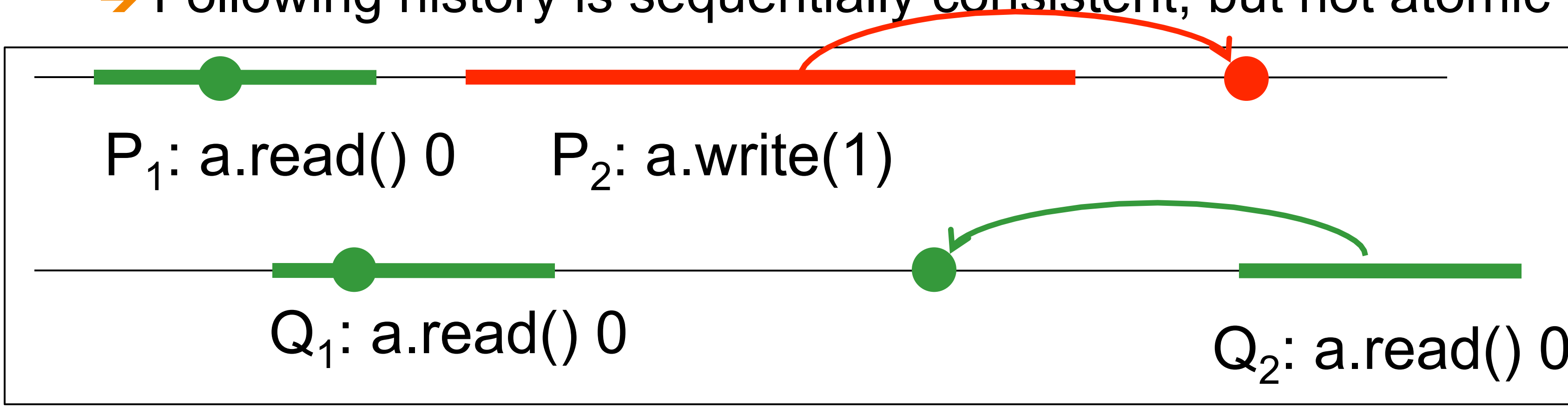
Key Properties of Atomicity

- Locality: For a system made of multiple objects x_i
  - ➔ H is atomic iff $H \mid x_i$ is atomic for all x_i
  - ➔ i.e. composing atomic objects results in an atomic system

- Non-blocking
  - ➔ pending operations can always complete (*)
  - ➔ * = provided they're defined (cf. dequeue an empty queue)

Alternative Consistency Models

- Sequential Consistency (Lamport, 1979)
  - ➔ H sequentially consistent iff
    ∃ acceptable sequential history S, S equivalent to H
- Difference 1: S can reorder operations!
  - ➔ Following history is sequentially consistent, but not atomic



- Difference 2: Not a local property!

Alternative Consistency Model

- (Strict) Serializability (Transactions, Databases)
  - → very close to atomicity as defined here
  - → except order $<_H$ defined on transactions
  - → transactions: multiple operations

- Important Consequences / Differences
  - → strict serializability not a local property
  - → it is a blocking property:
    sometimes transactions must abort

Summary

- This session
  - ➔ formal approach to specifying legal parallel behaviours
- Key notions:
  - ➔ processes, shared objects
  - ➔ history, sub-histories
  - ➔ sequential histories, legal histories, equivalent histories
- Atomicity builds on all these notions
  - ➔ define legal // behaviour based on sequential specification
  - ➔ find an equivalent seq. history that meet specific criteria
- Important to reason about parallel programs
  - ➔ specifications, proofs (manual, automatic), composition

References

- Maurice P. Herlihy and Jeannette M. Wing. 1990. *Linearizability: a correctness condition for concurrent objects*. *ACM Trans. Program. Lang. Syst.* 12, 3 (July 1990), 463-492. DOI=10.1145/78969.78972 http://doi.acm.org/10.1145/78969.78972
- Chapter 4 "Atomicity: Formal Definition and Properties" in Michel Raynal, *Concurrent Programming: Algorithms, Principles, and Foundations*, Springer, Jan 2013 , ISBN-13: 978-3642320262