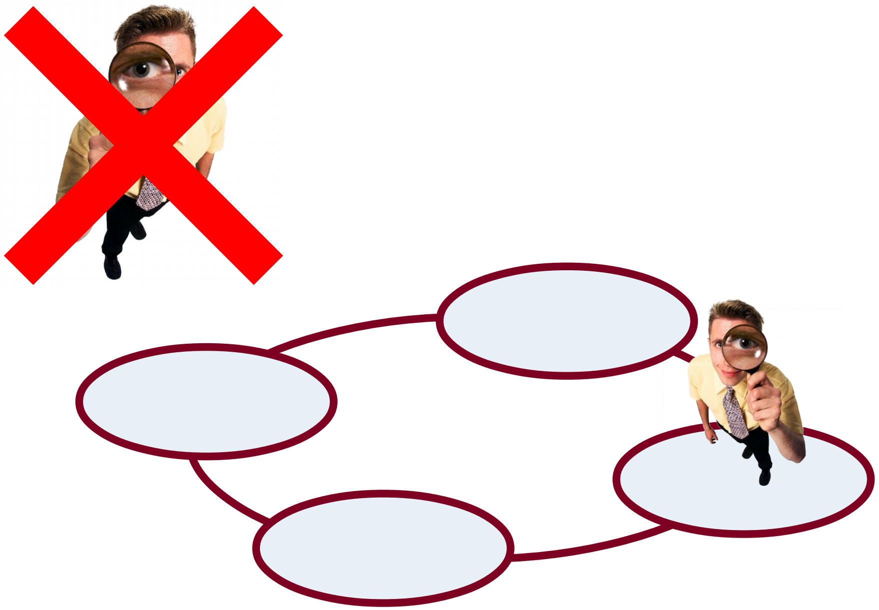
Goals

- No global omniscient observer
- How to observe/reason about the system from the inside?

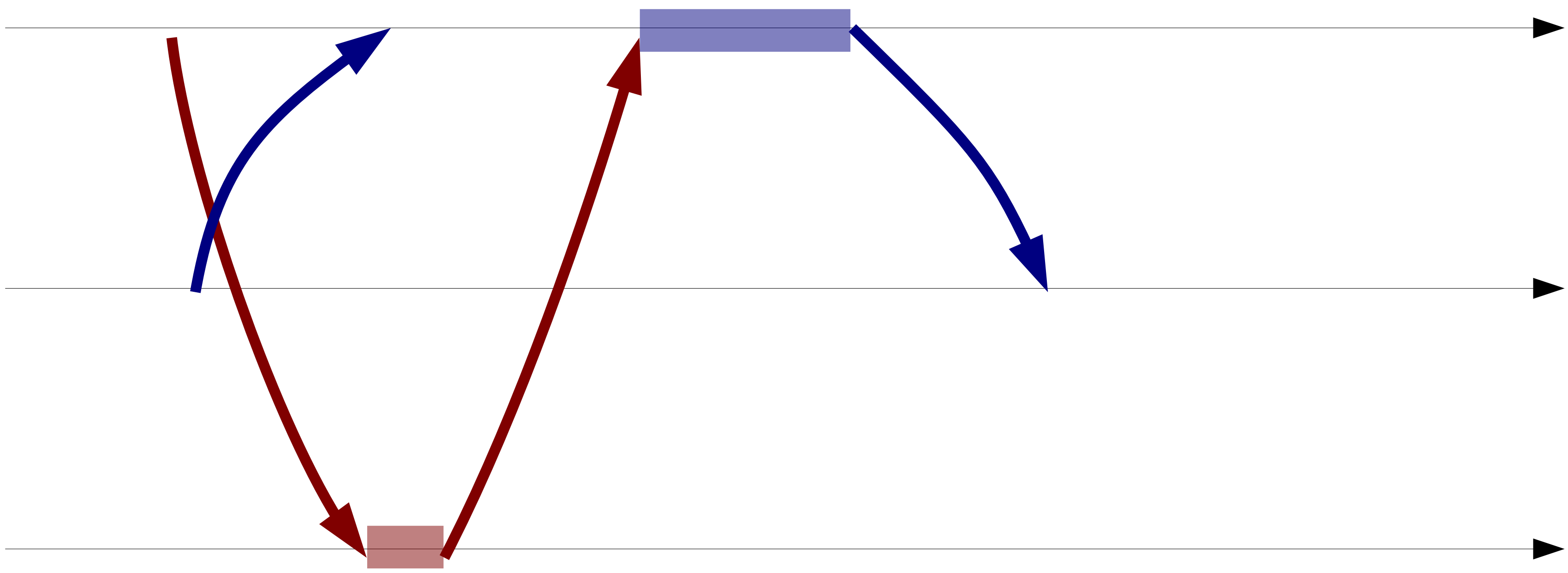


Example: Distributed deadlock

- Remote invocation
- All processes request and reply to invocations
- A mutex is held while invoking remotely or handling remote invocations
- Distributed deadlock possible when multiple processes invoke each other

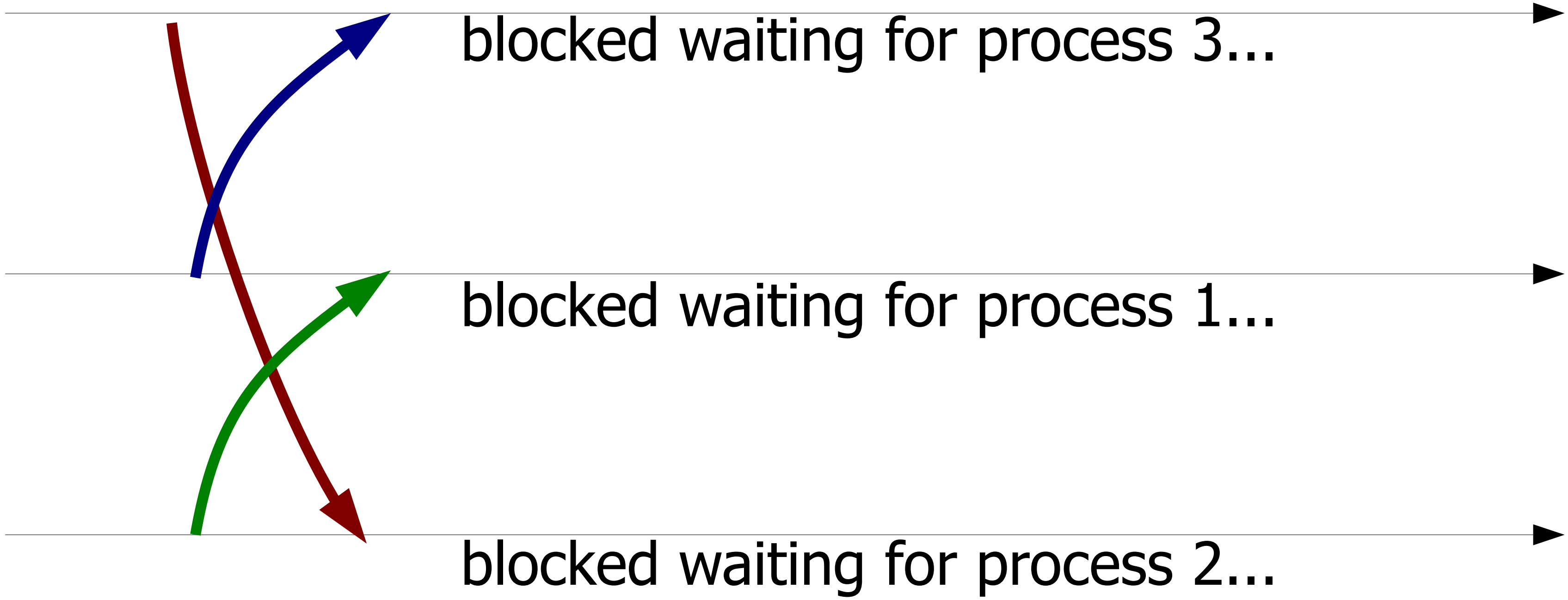
Example: Distributed deadlock

- Deadlock-free run:



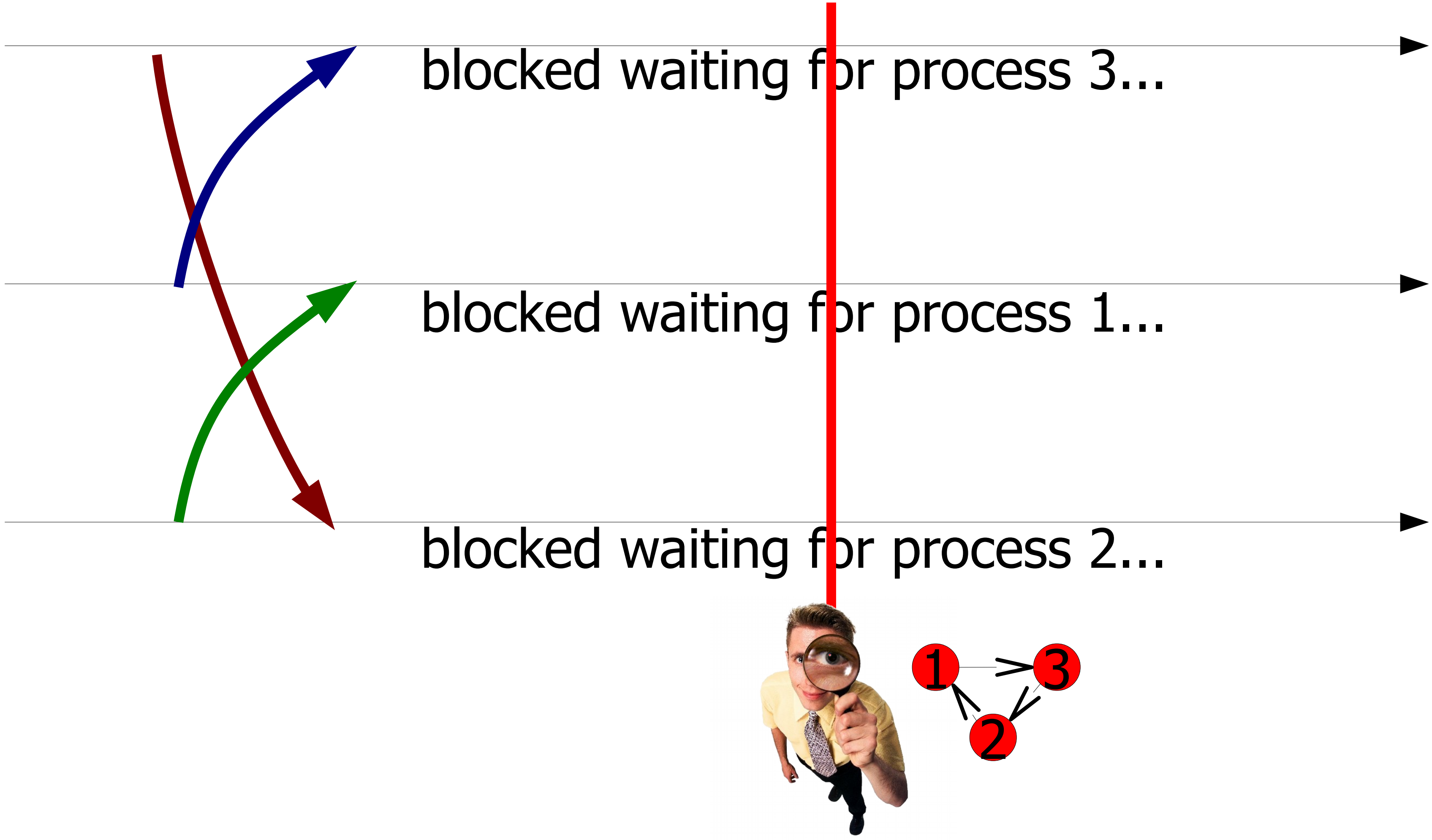
Example: Distributed deadlock

- Distributed deadlock:



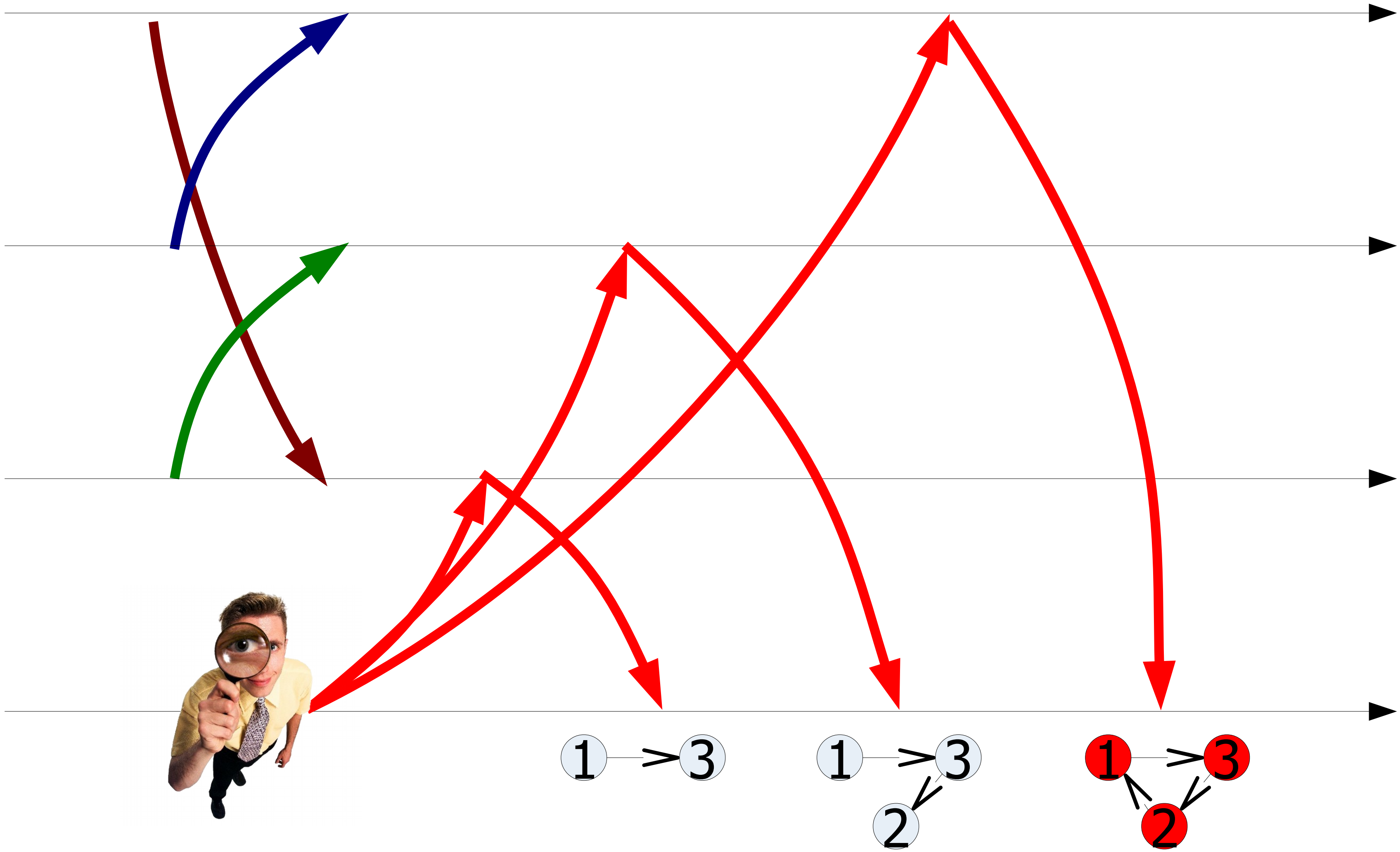
Example: Distributed deadlock

- Instant observation is impossible:



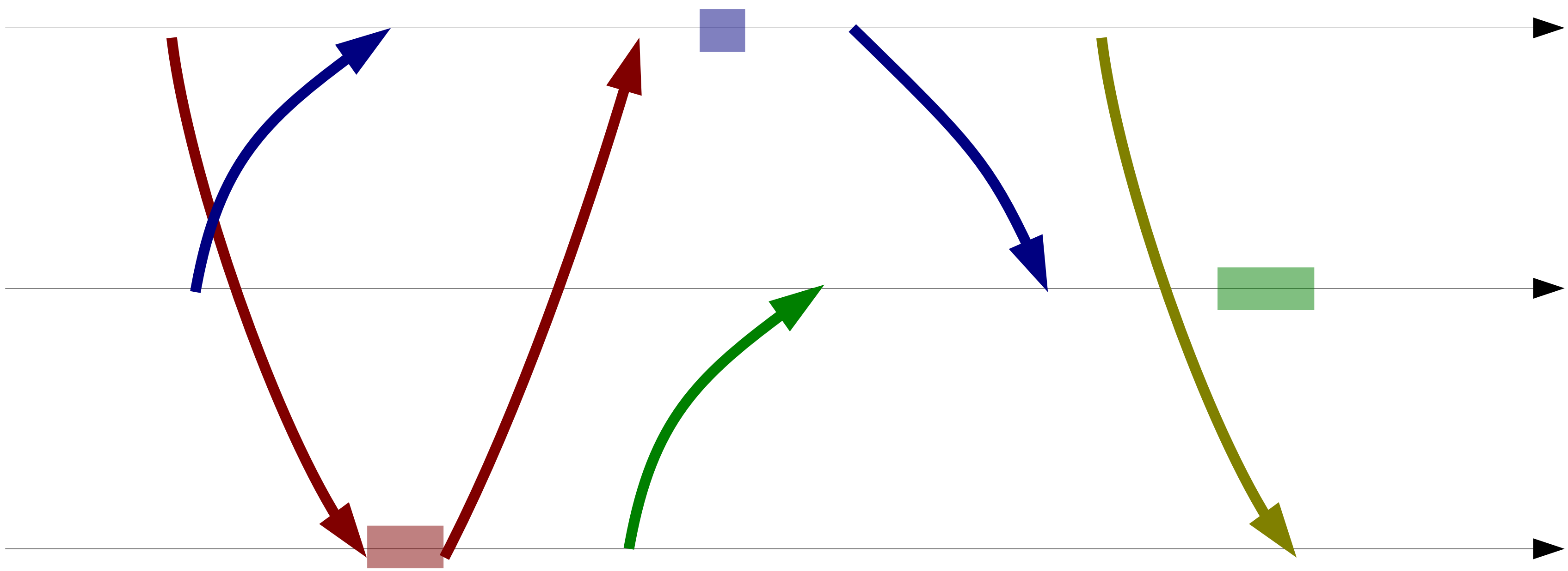
Example: Distributed deadlock

- Deadlock detection with a "wait for" graph:



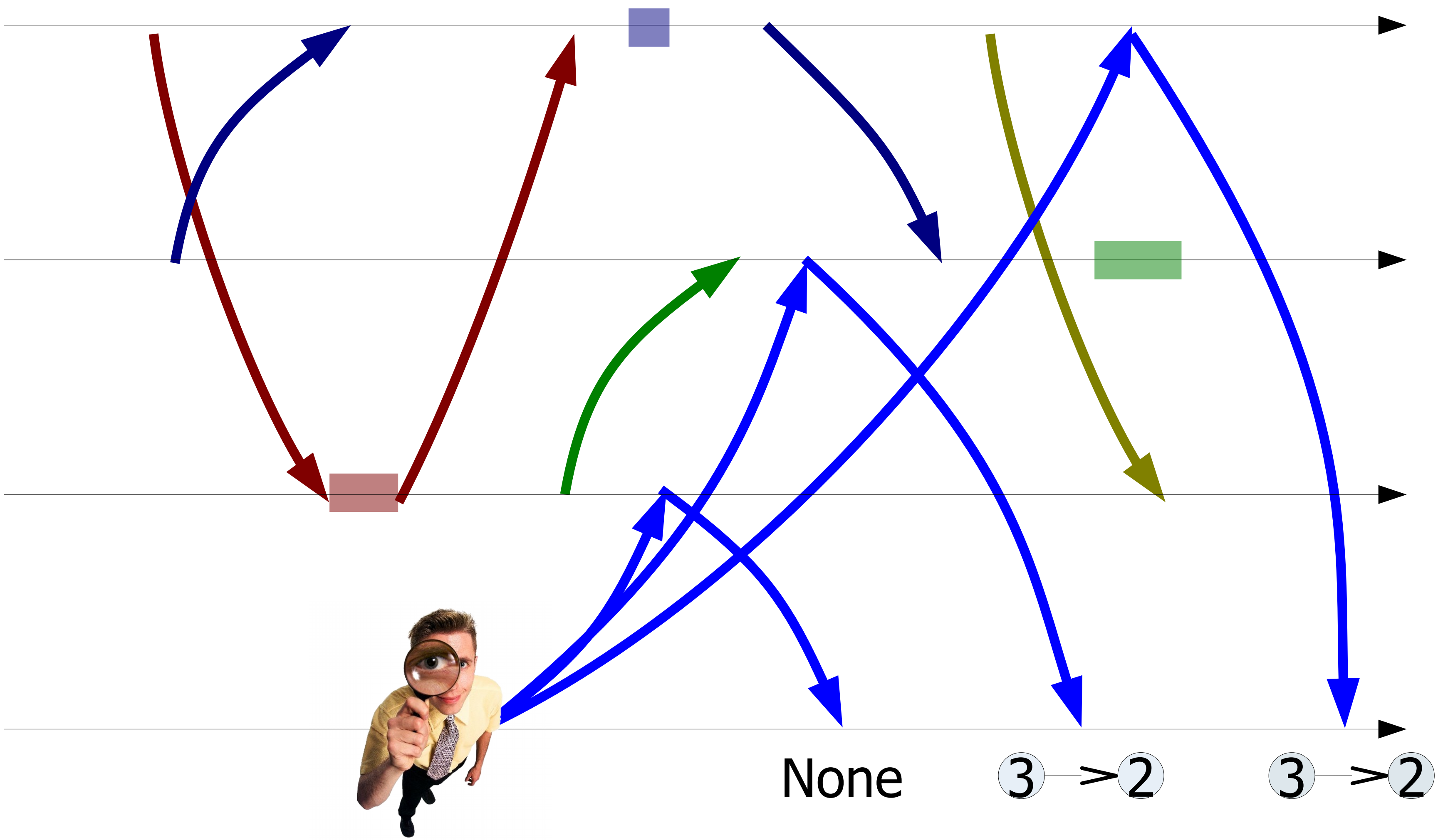
Example: Distributed deadlock

- A more complex deadlock-free run:



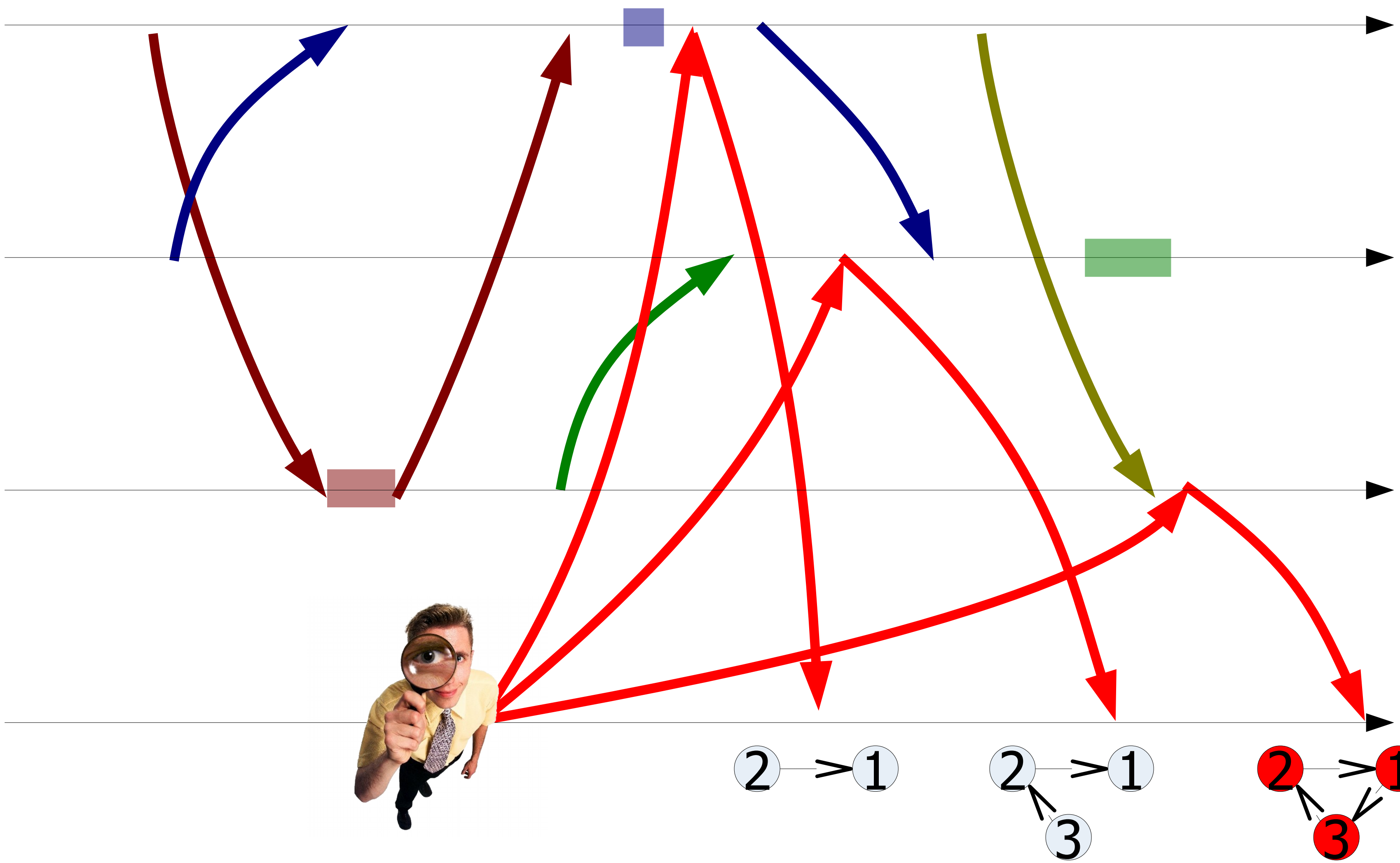
Example: Distributed deadlock

- A deadlock-free WFG:



Example: Distributed deadlock

- A WFG with a ghost deadlock:

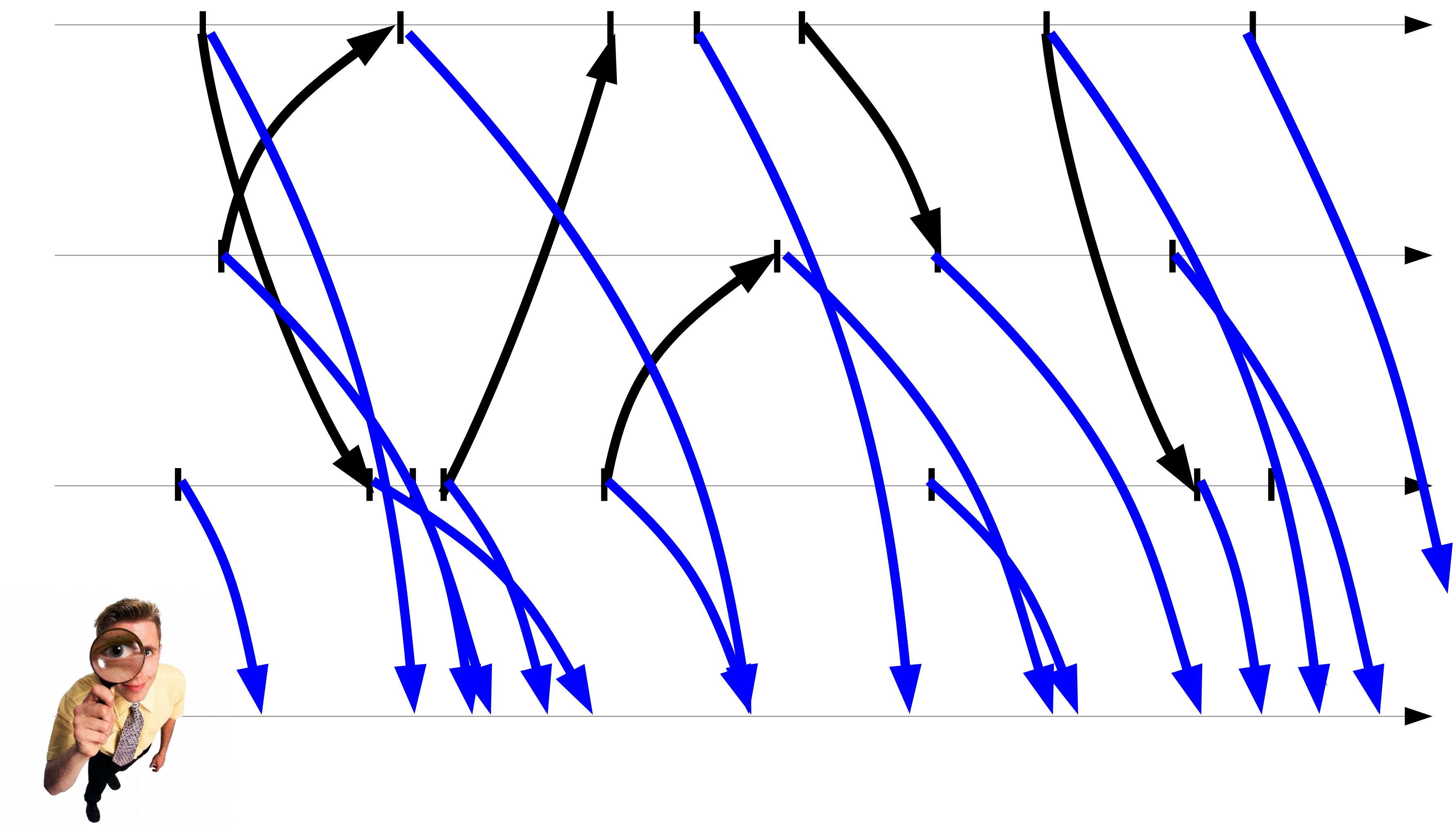


Global Property Evaluation

- Similar problems:
  - Distributed garbage collection
  - Distributed conversation threads
  - ...
- Can it be solved in an asynchronous system?
- Methods that can be used? Relative cost?

Passive monitor process

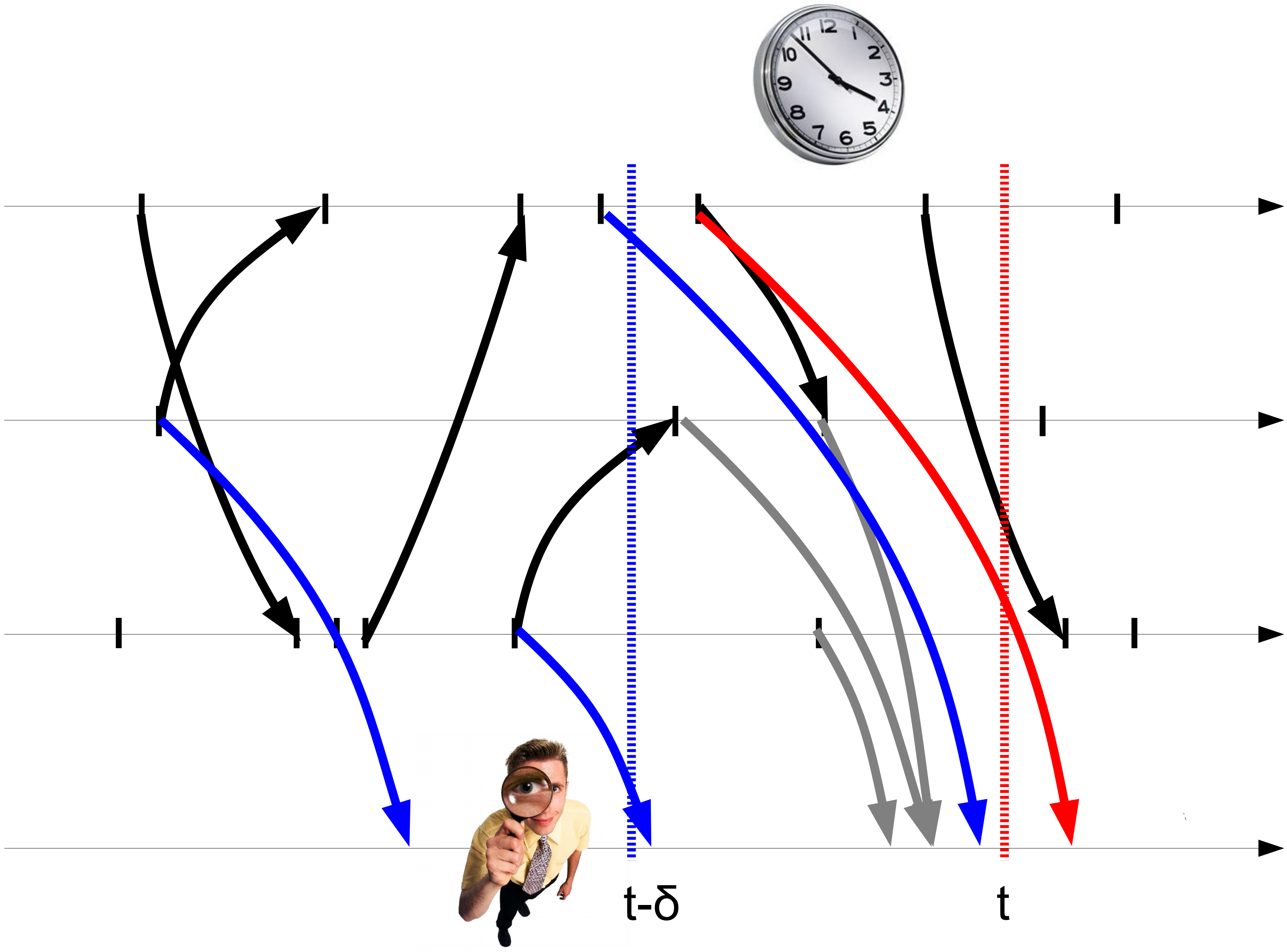
- Report all events to monitor:



First try: Synchronous system

- Global clock, δ upper bound on message delay
- Tag events with real time
- Consider events only up to t-δ
  - With synchronous rounds, this means using messages from the previous round!

First try: Synchronous system



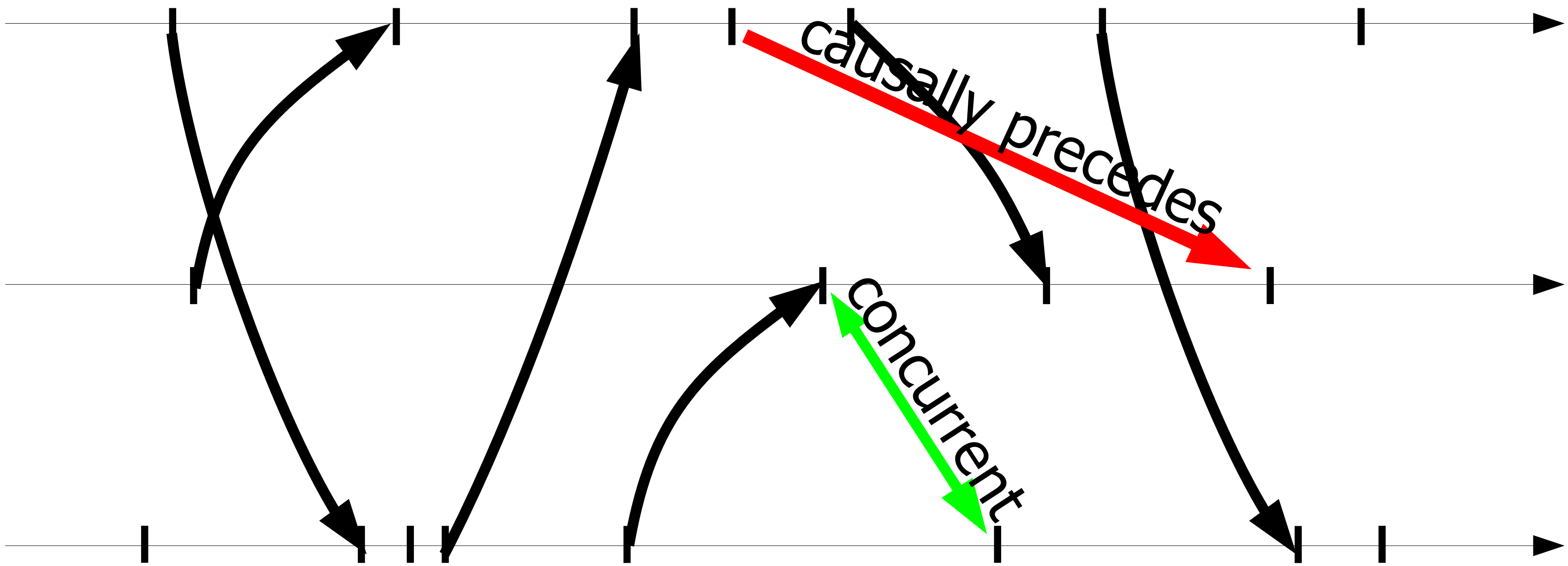
Clock properties

- What properties of a real-time clock make this approach correct?

Definition: Causality

- Events i and j are causally related ($i \rightarrow j$) iff:
  - i precedes j in some process p
  - for some m, i=send(m) and j=receive(m)
  - for some k, $i \rightarrow k$ and $k \rightarrow j$ (transitivity)
- Events i and j are concurrent ($i || j$) iff neither $i \rightarrow j$ or $j \rightarrow i$

Causality

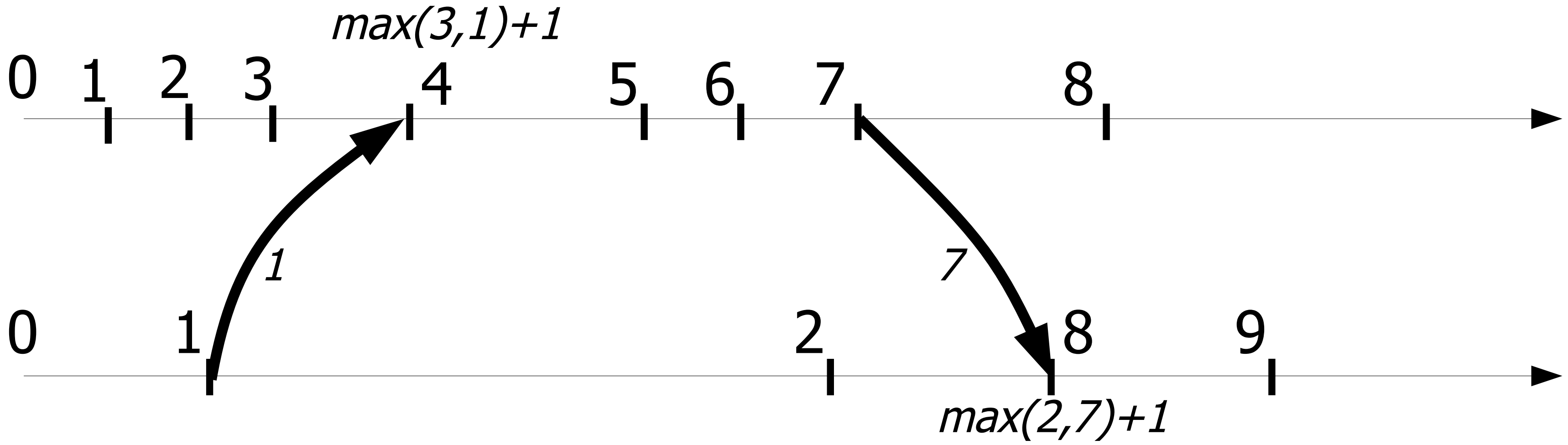


Clock properties

- RC(i) the time at which i happened
- If $i \rightarrow j$ then $RC(i) < RC(j)$
- For some event j:
  - When we are sure that there is no unknown i such that $RC(i) < RC(j)$
  - Then there is no i such that $i \rightarrow j$
- Can we build a logical clock with the same property?

Scalar logical clock

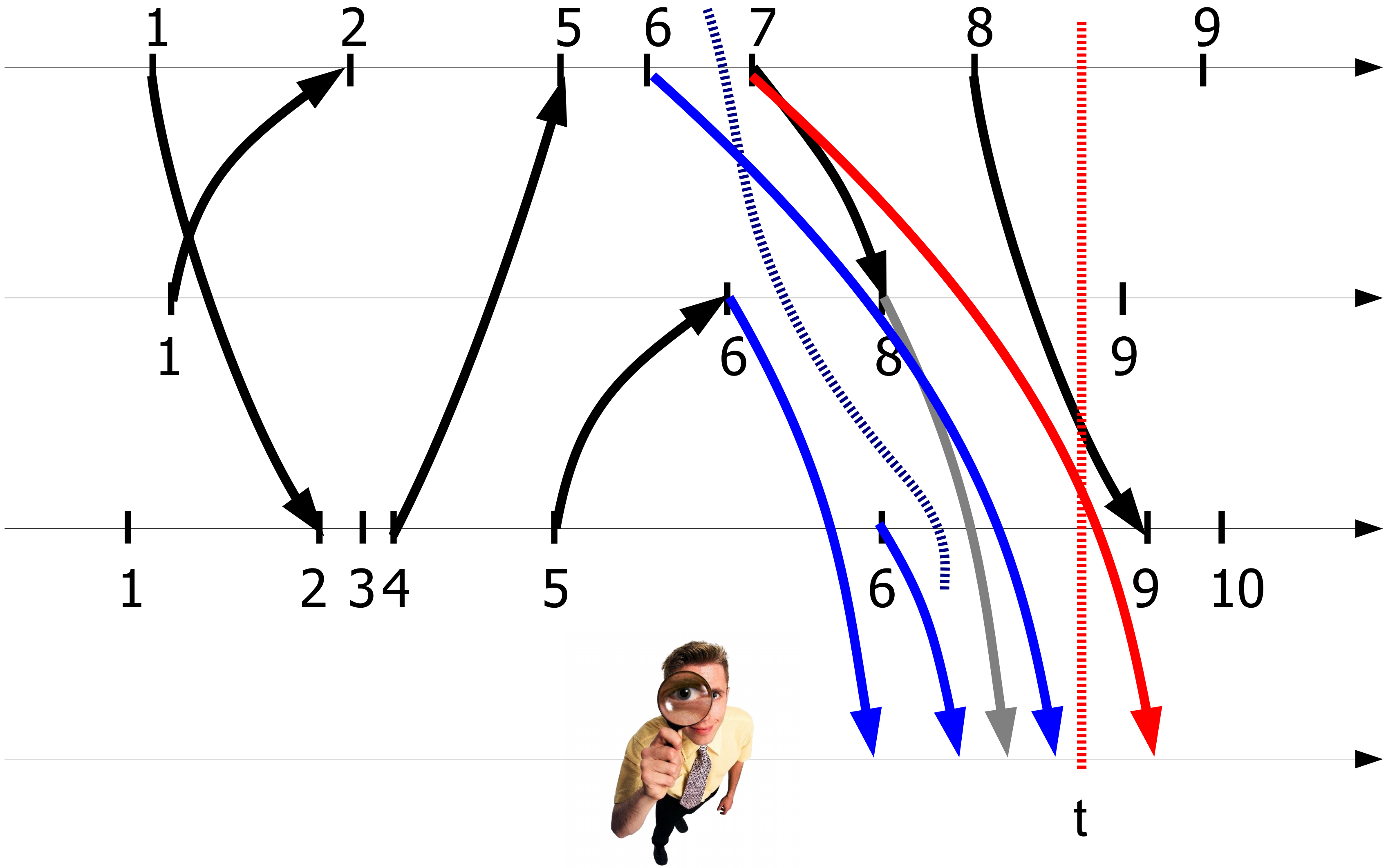
- Local events: increment counter
- Send events: increment and then tag with counter
- Receive events: update local counter to maximum and then increment



Second try: Logical clock

- Use scalar logical clock
- Use FIFO channels
- Consider events only up to the minimum of maximum tags

Second try: Logical clock

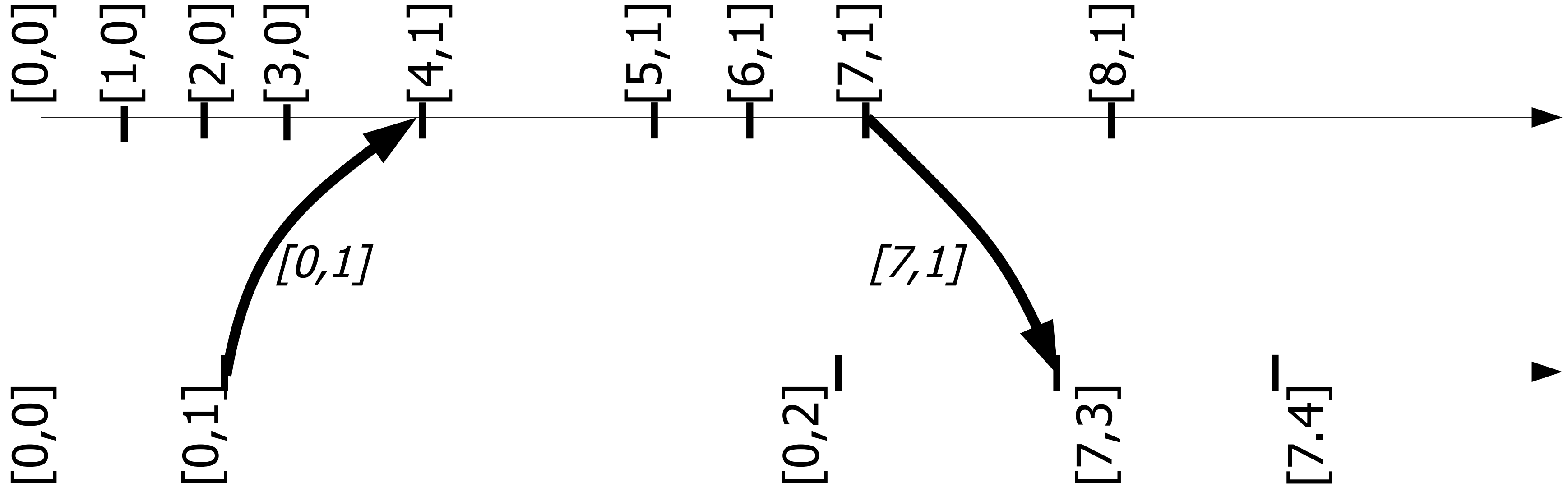


Scalar clocks

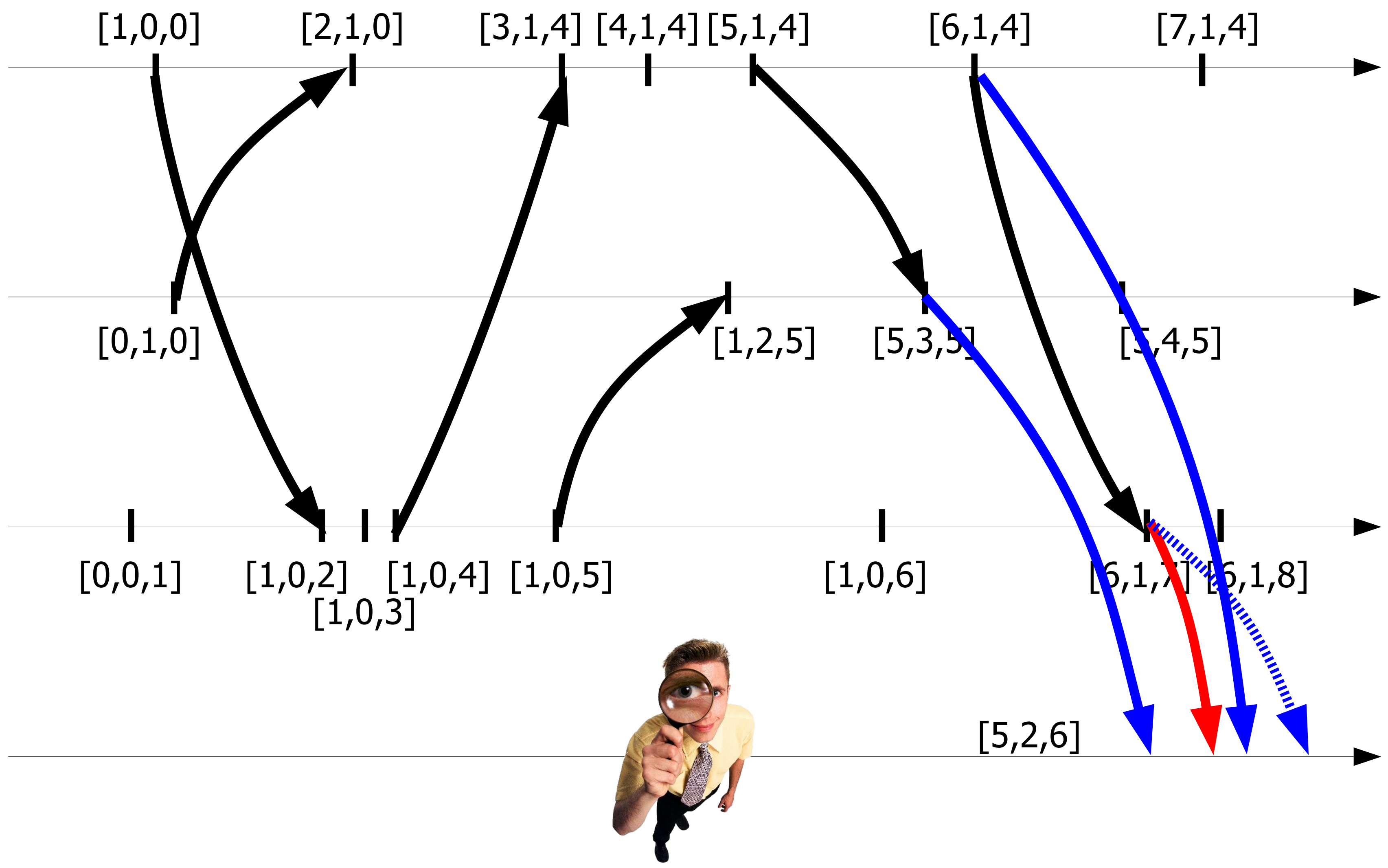
- Synchronous system (RC):
  - Delay δ to consistency
- Asynchronous system (LC):
  - Possible unbounded delay to consistency
  - Blocks if some process stops sending messages

Vector clock

- Local event at i: increment counter i
- Send event at i: increment counter i and tag with vector
- Receive event at i: update each counter to maximum and increment counter i



Third try: Vector clock



Causal delivery to monitor

- The monitor considers events as follows:
  - With local vector l[...]
  - For some event r[...] from i
  - Ignore it until:
    - $l[i]+1=r[i]$
    - For all $j \neq i$: $r[j] \leq l[j]$

Causal delivery in a group

- Broadcast messages in a group
  - Same as “All processes are monitoring send events”
- Increment local counter only on send
  - Not on receive
  - Not on internal events

Summary

- With scalar clocks:
  - Consistent observation by ignoring some messages
  - Blocks if one process stops sending messages
- With vector clocks and causal delivery:
  - Consistent observation whenever a message is delivered
  - Blocking can be avoided by forwarding past messages