Distributed Computability and distributed problems

Sergio Rajsbaum
UNAM, Mexico

Joint work with M. Herlihy and others, especially Armando Castañeda and Michel Raynal DISC 2015 extensions in NETYS 2017

Three epochs of computing

- Sequential
- Parallel
- Distributed

In the beginning there was sequential

- modern computer science was born with the discovery of universal computing models
- Especially the Turing machine
- "anything" that can be computed, can be computed by a TM

Computable functions

- Notion of a computing device
- and of a problem

Then there was parallel

- Model of choice– PRAM
- Multiple processes execute steps synchronously
- No process and no communication failures
- In 2007 Kanbalam put UNAM at number 28 among universities, 1,368 processors at a cost of 3 million dollars

Sequential vs. parallel computability

- No challenge to precise definition of “mechanical procedure”
- Wikipedia: TM equivalent to multi-tape Turing machine, is usually interpreted as:
- sequential computing and parallel computing differ in questions of efficiency, but not computability.
- Notion of problem: function

Distributed computing is everywhere!

- Nearly every activity in our society works as a distributed system made up of human and computer processes
- From micro multi-core to wide area systems
- “This revolution requires a fundamental change in how programs are written. Need new principles, algorithms, and tools” [Herlihy Shavit book]
- Challenge to precise definition of “mechanical procedure”
- and of function ?!

What is the distributed equivalent of a function?

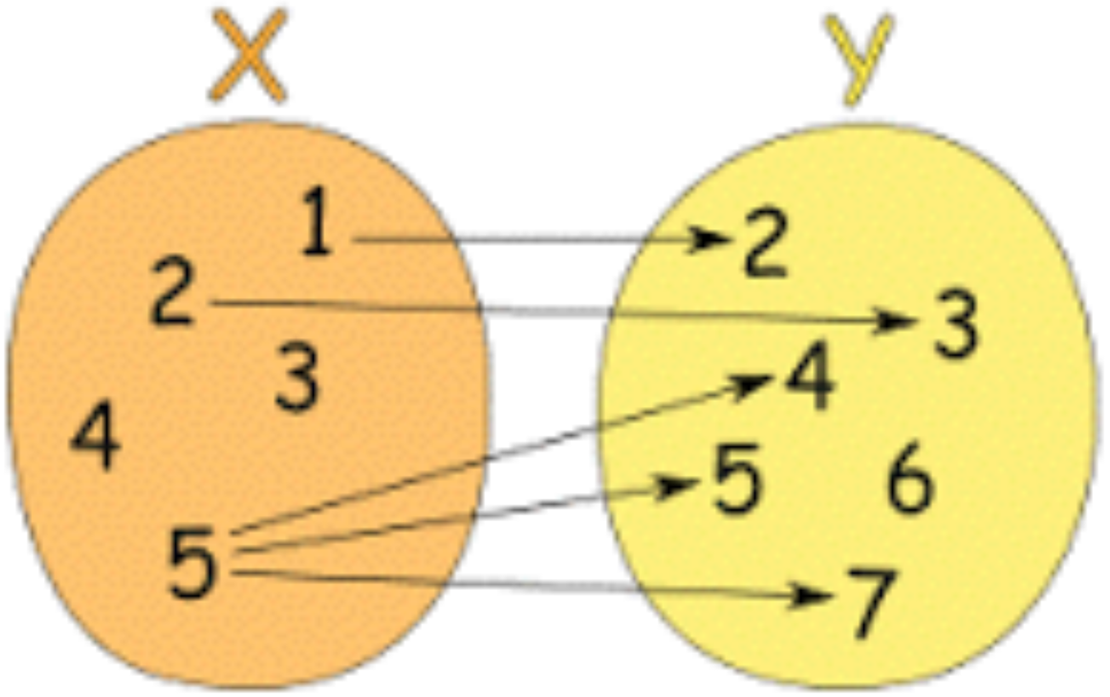
A function!
with a ``little” change

- Inputs and outputs are vectors
- The i-th component belongs to process i
- Also, for each input vector, we allow one or more output vectors

A function!
with a ``little" change

Process Pi knows only its input or output

Relation Delta



Set of Vectors Set of Vectors

Tasks: Consensus

- Inputs vectors define initial values from some set V
- Outputs vectors, where all entries are equal
- Relation: If the input vector has a value v, then the output vector with all entries equal to v is OK
- First task computability characterisation BMZ 90 , J. Algorithms (1 crash failure, message passing)

Tasks are not functions
with a ``little" change !

Point of view- perspective

- Distributed computing is the science of perspectives

Multiperspectivism

- Each process has its own local view
- Nobody can observe the global state

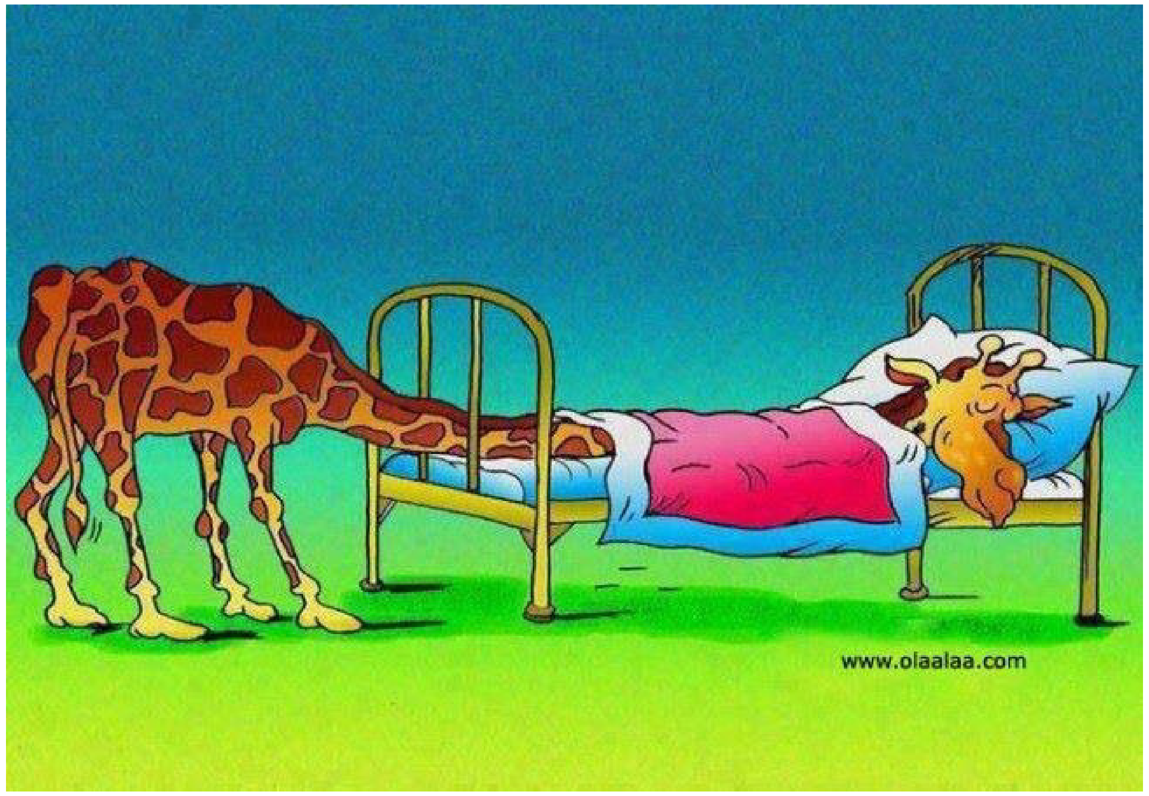
The science of perspectives

- We study how perspectives evolve with time

- and how are distributed decision taken, based on individual perspectives

The science of perspectives

Under unreliable communication and failures!



Distributed computing is different from sequential/parallel

– Processes may never be able to agree on a single perspective, or it would take too much time to agree

— They need to collaborate while having different perspectives

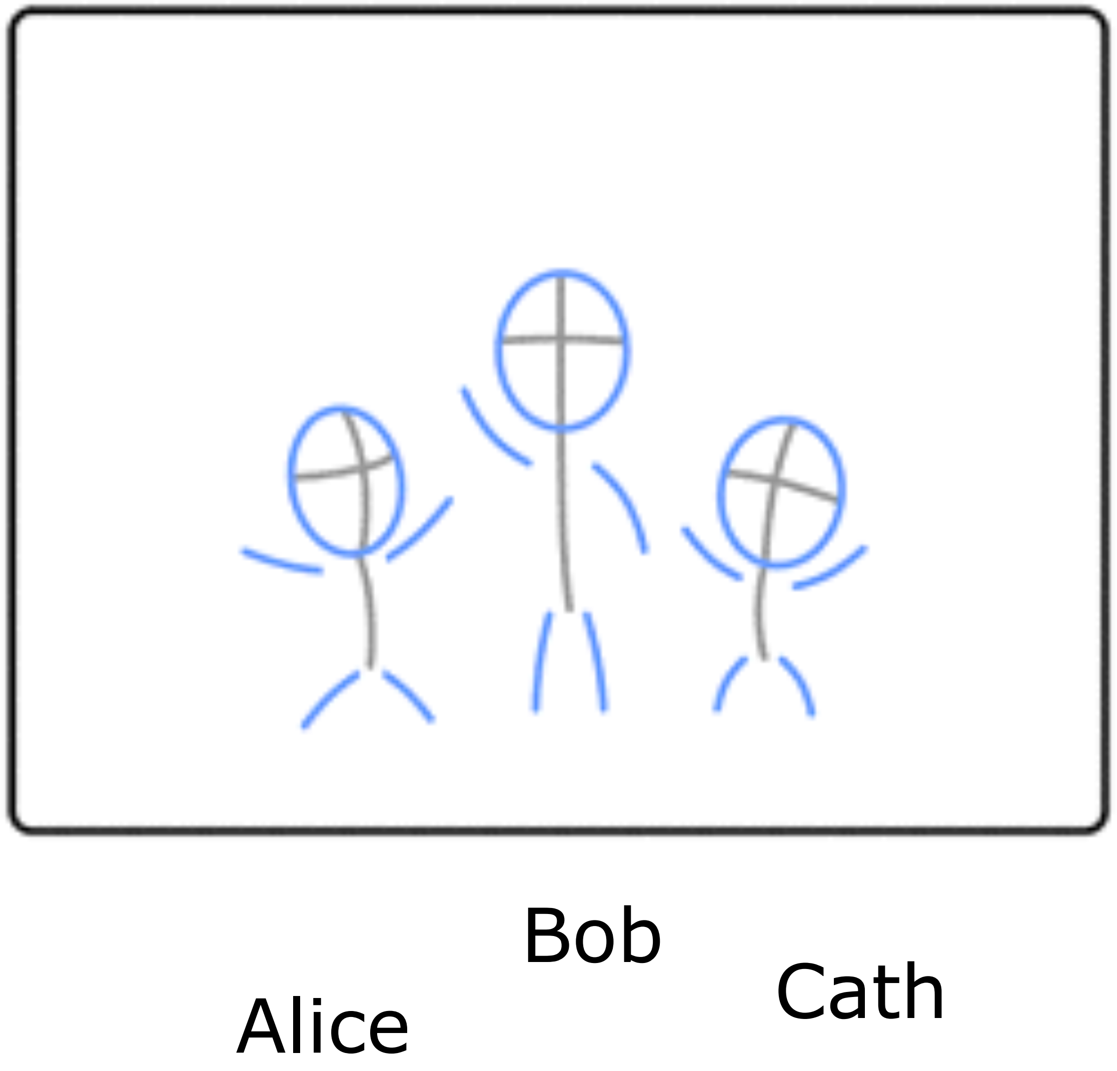
Initial perspectives

input complex

Three players, 3 cards

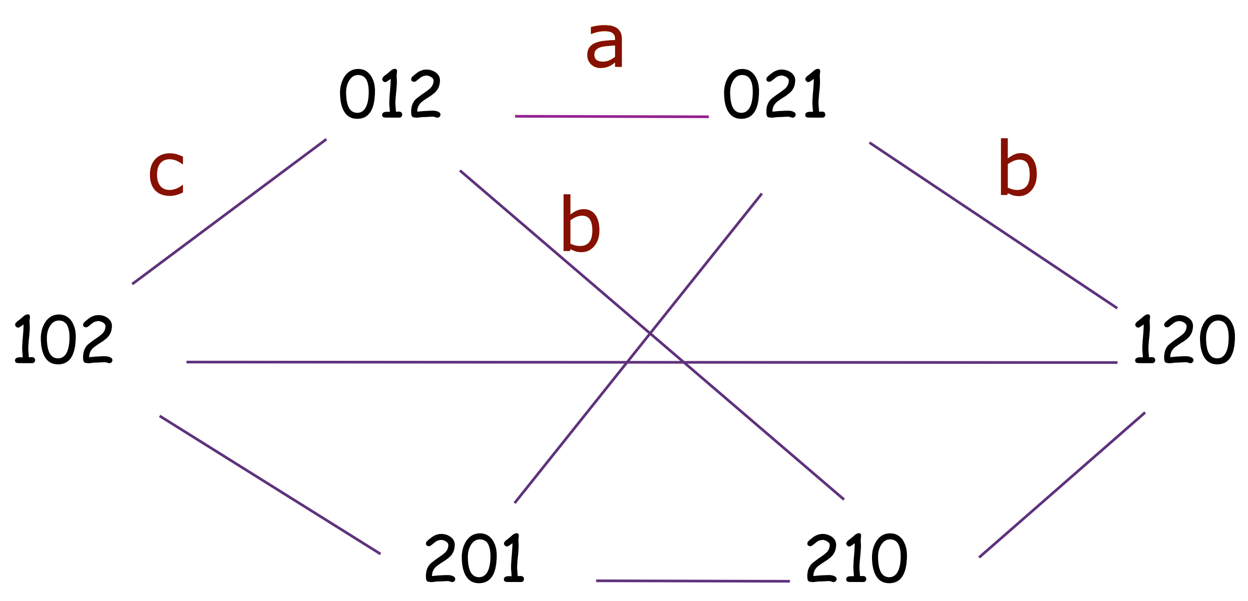
Alice, Bob, and Cath have each drawn one card from a deck of three cards 0, 1, and 2.

Each person can only see his own card



Three players, 3 cards

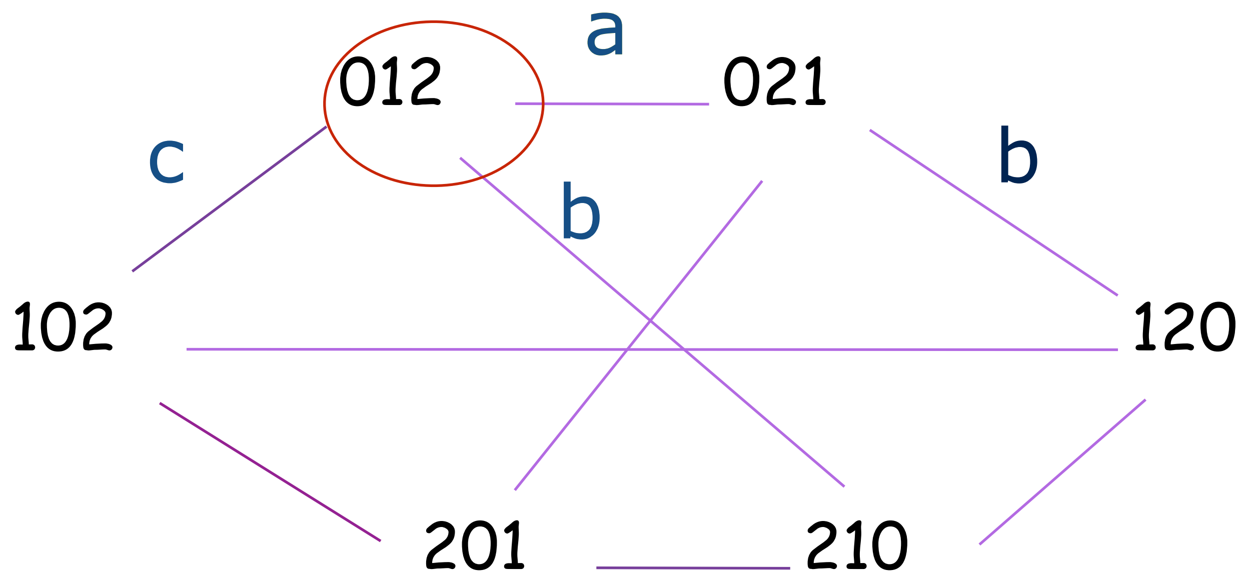
Each initial configuration is a vector, specifying the card that each person got



Graph Representation

- Called Kripke structures in epistemic logic

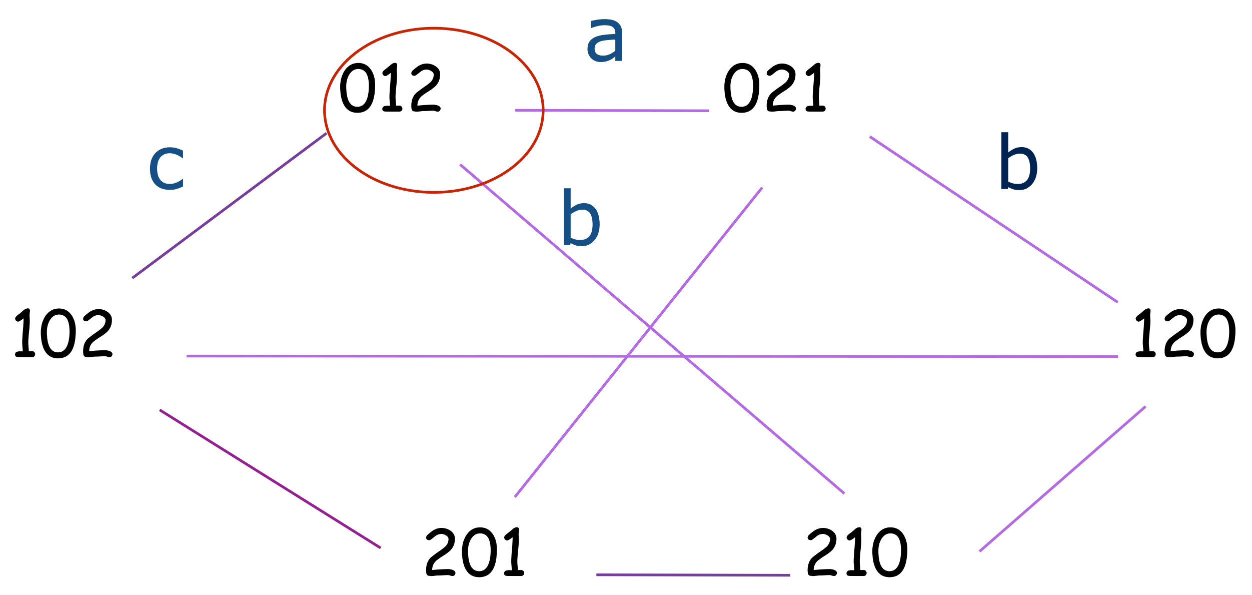
Three players, 3 cards



Alice has drawn card 0, Bob card 1, and Cath card 2

Three players, 3 cards

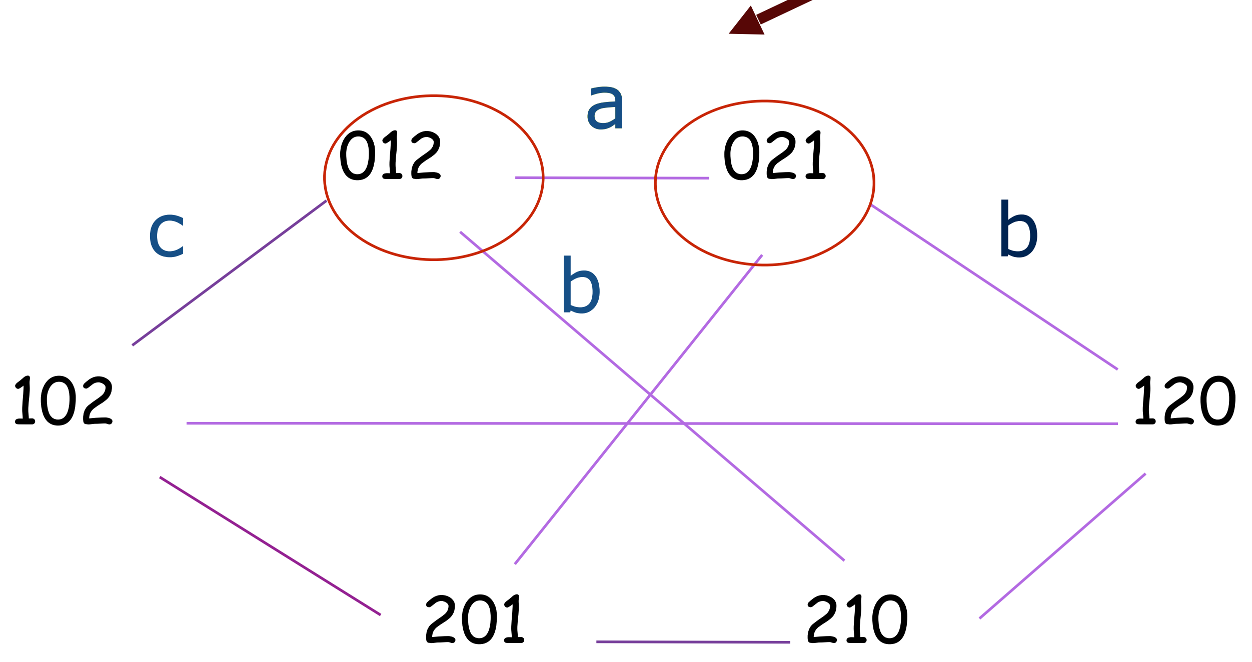
Each edge labeled with the agent that does not distinguish the worlds at its end vertices



Three players, 3 cards

- Alice, Bob, and Cath have each drawn one card from a deck of three cards 0, 1, and 2.

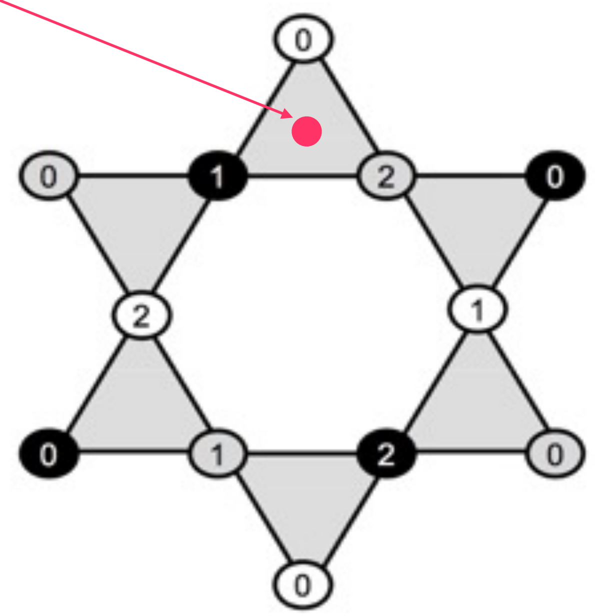
Alice does not know which cards have each of the others



Each edge labeled with the agent that does not distinguish the worlds at its end vertices

Representation by the dual of a graph, a *complex*

Alice has 0, Bob has 1, and Cath has 2.



A=white

B= black

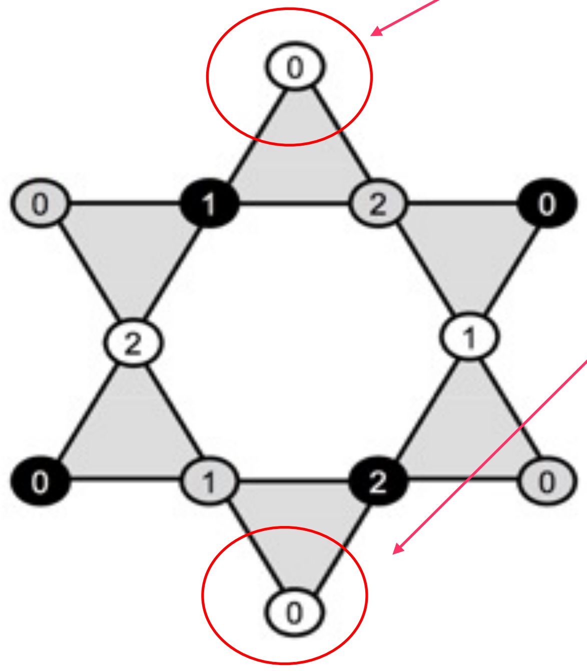
C=grey

Each 2-simplex correspond to a possible world, its vertices are labeled with names and views

Three player, 3 cards

Representation by a *complex*

Alice does not know which cards have the others.



A=white

B= black

C=grey

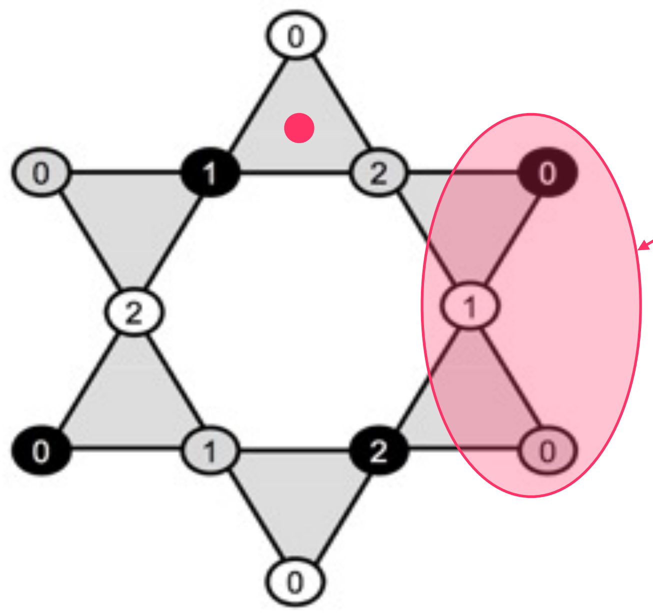
Each 2-simplex correspond to a possible world, its vertices are labeled with names and views

Some vertices depicted as distinct are actually the same. Rook complex.

Perspectives Evolve with communication

Evolution of perspectives

-- Alice now says "I do not have card 1".



Removed

A=white

B= black

C=grey

Evolution of perspectives

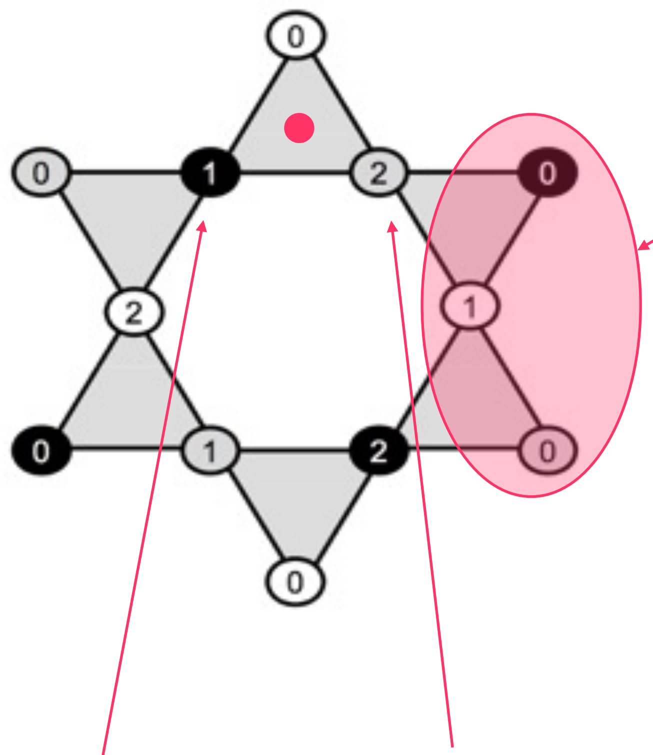
-- Alice now says “I do not have card 1”.

- Communication is by public announcements

A=white

B= black

C=grey



Removed

Bob learned nothing, Cath knows the draw,

Which tasks can we solve under different perspectives?

Informal task specifications

k-set agreement and consensus (k=1)

- **propose(x)**: each process has an input x, returns a value y

1. **Agreement**: at most k different values are returned

2. **Validity**: an output value y was proposed

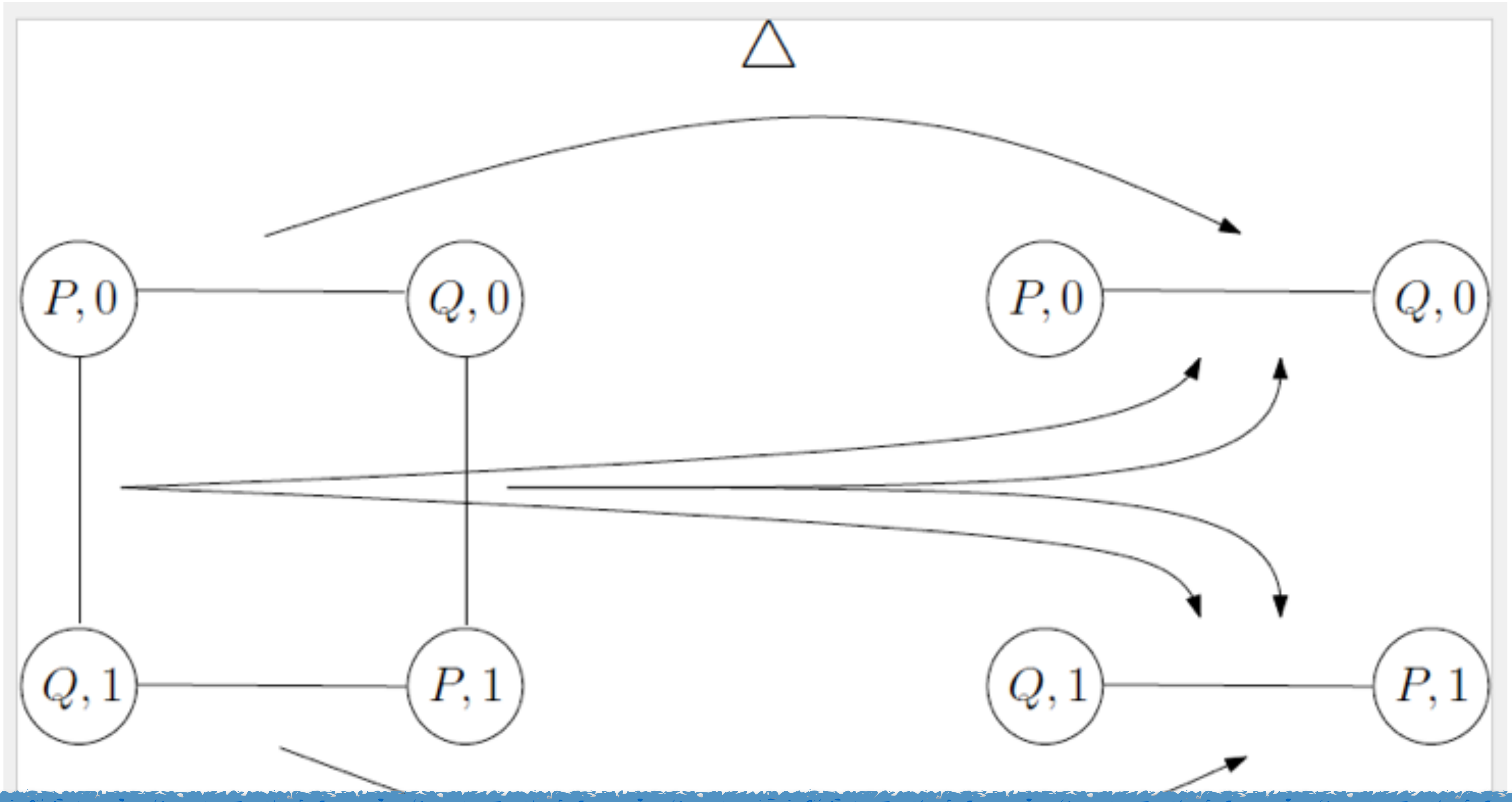
Informal specifications

- **Validity(v)**: propose v, return w
  1. **validity property**:   w is the value proposed by someone
  2. etc (in other problems)

Informal specifications

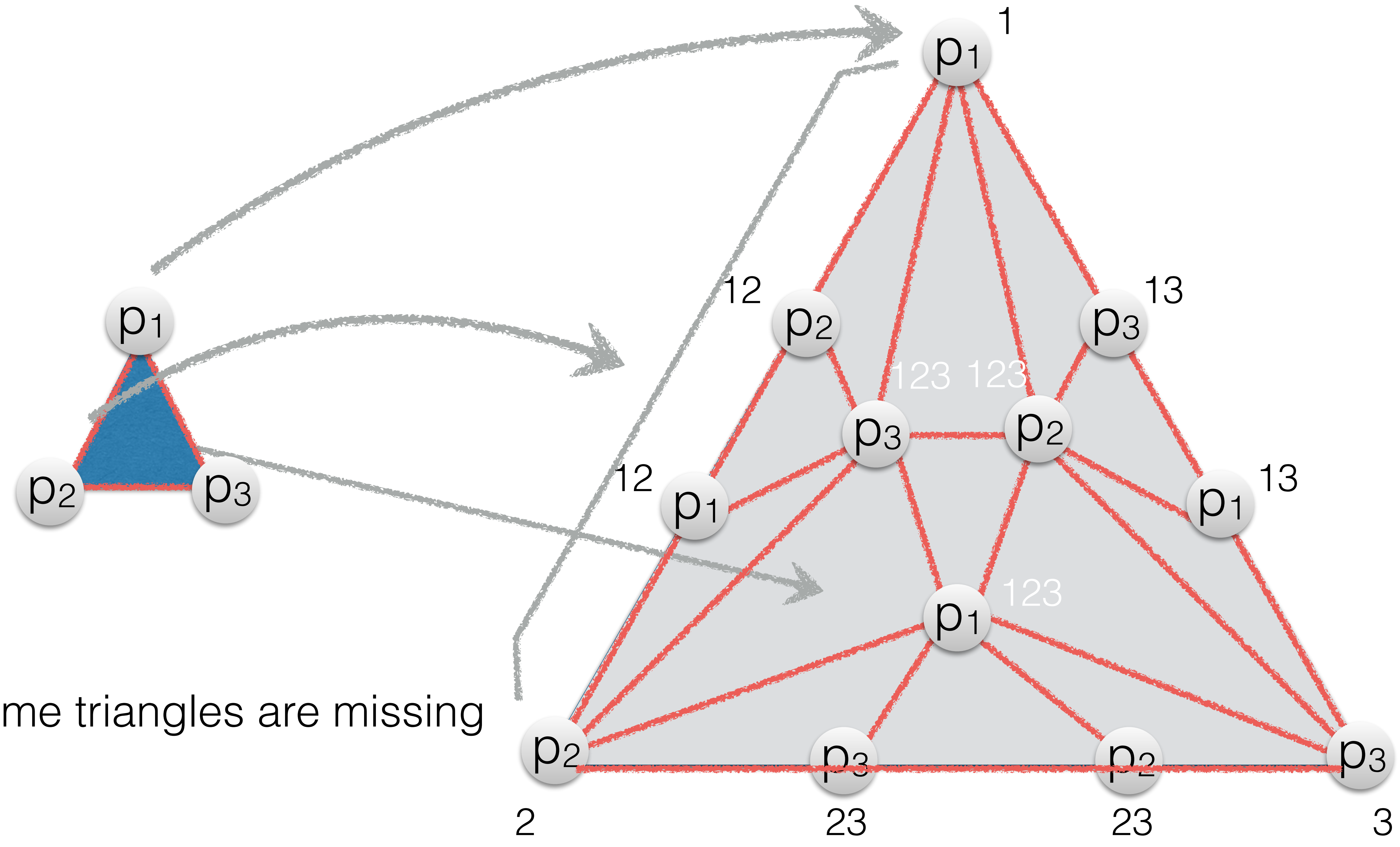
- **Snapshot(v)**: propose v, return set View
  1. **validity property**: w is the value proposed by someone, w in View
  2. Views can be ordered by containment

Formal: Tasks

- 

Tasks tell what might happen in presence of concurrency

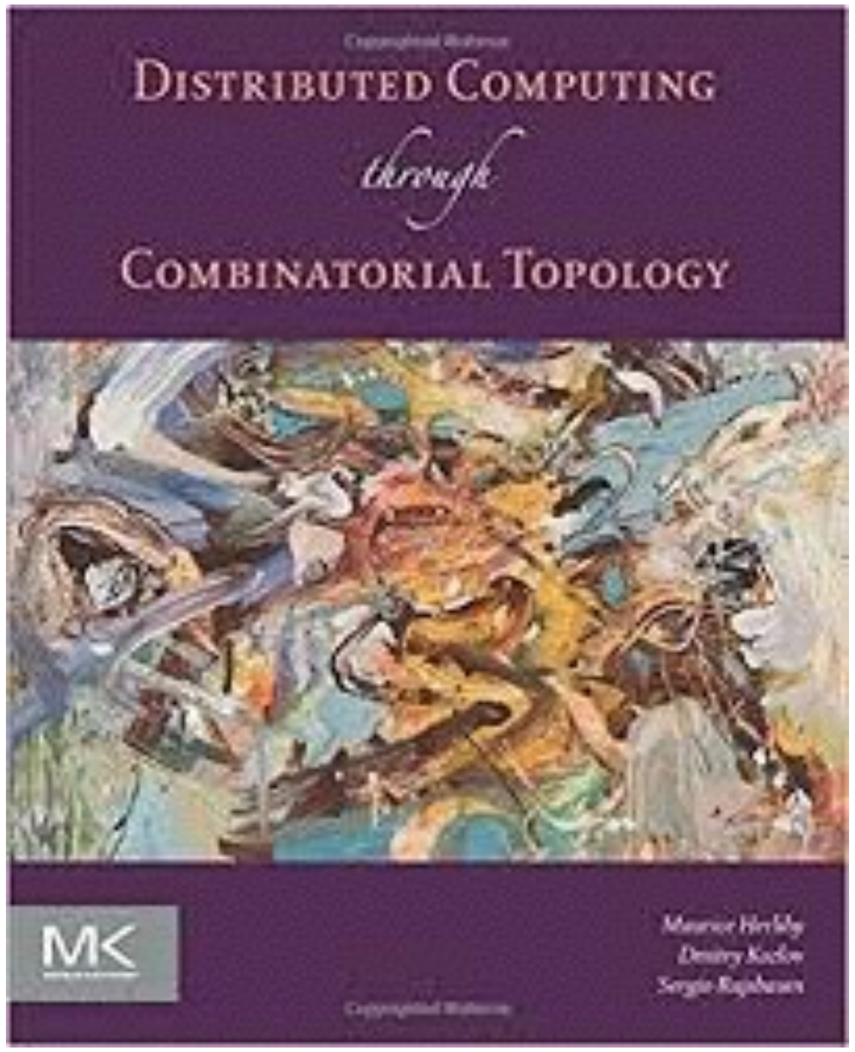
Write-Snapshot Task



*Some triangles are missing

Importance of Tasks

- Basic computability unit
- Study of **set agreement** and **renaming** lead to a connection between distributed computing and **topology**
- Characterisation of solvable tasks is many models
- **Orthogonal to TM computability, each process may be an infinite state machine!**



But…

Distributed computer scientists excel at thinking **concurrently**, and building large distributed systems

Yet, they evade thinking about concurrent problem **specifications**.

Weaver Ants Building Nest from Mango Leaves, Ubon Ratchathani, Thailand

It is infinitely easier and more intuitive for us humans to specify how abstract data structures behave in a sequential setting.

Nir Shavit, CACM 2011

object

In practice often sequential specifications are used

A very different style of problem specification, from tasks

An object- a queue

- Defined by a possible invocations and responses

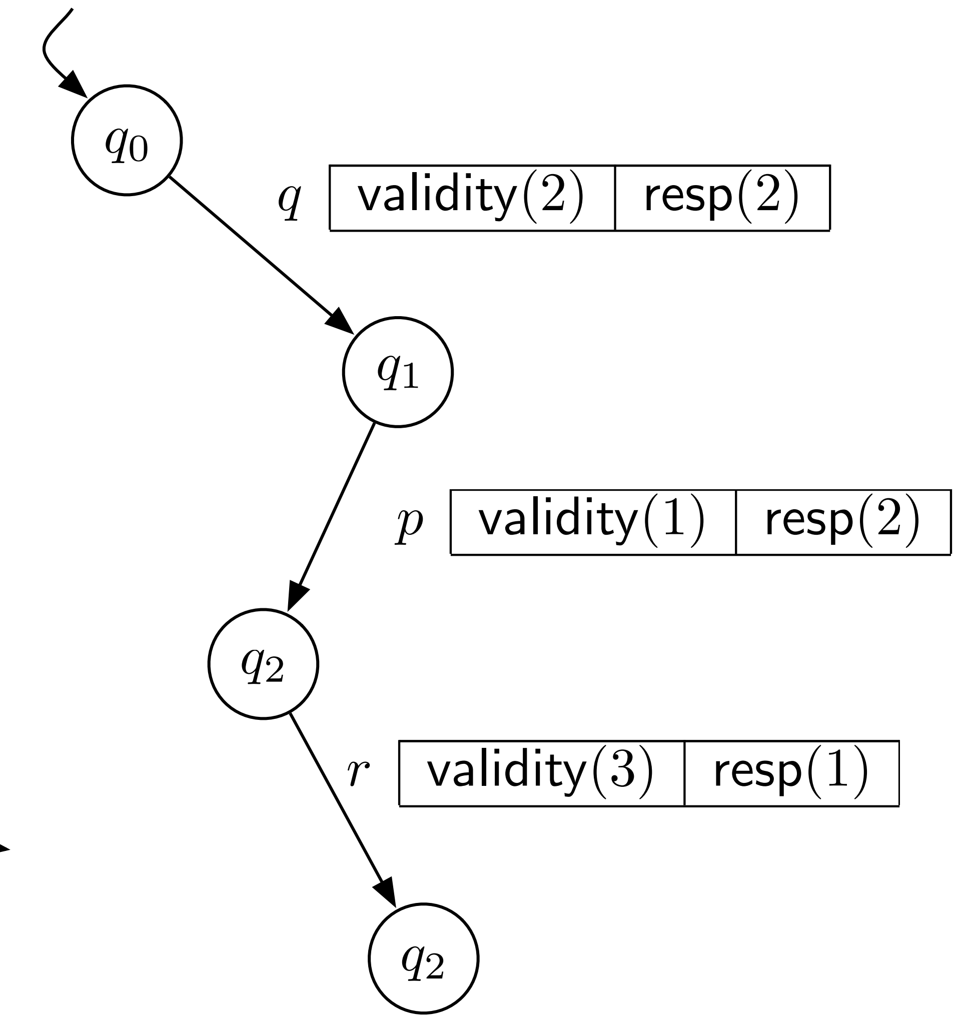
- The processes may invoke concurrently but specified in terms of a sequential specification, namely…

An object

- an **automaton** describing the outputs the object produces when it is accessed sequentially.
    - with a notion of **state**, and transitions of the form

$$\delta(q, in) = (q', r)$$

Example: validity



- Invocations propose input
- responses return values that have been proposed

Sequential specifications are convenient

- Provide the notion of a state
    - Specification manual grows linearly with the number of operations

Is an implementation correct?

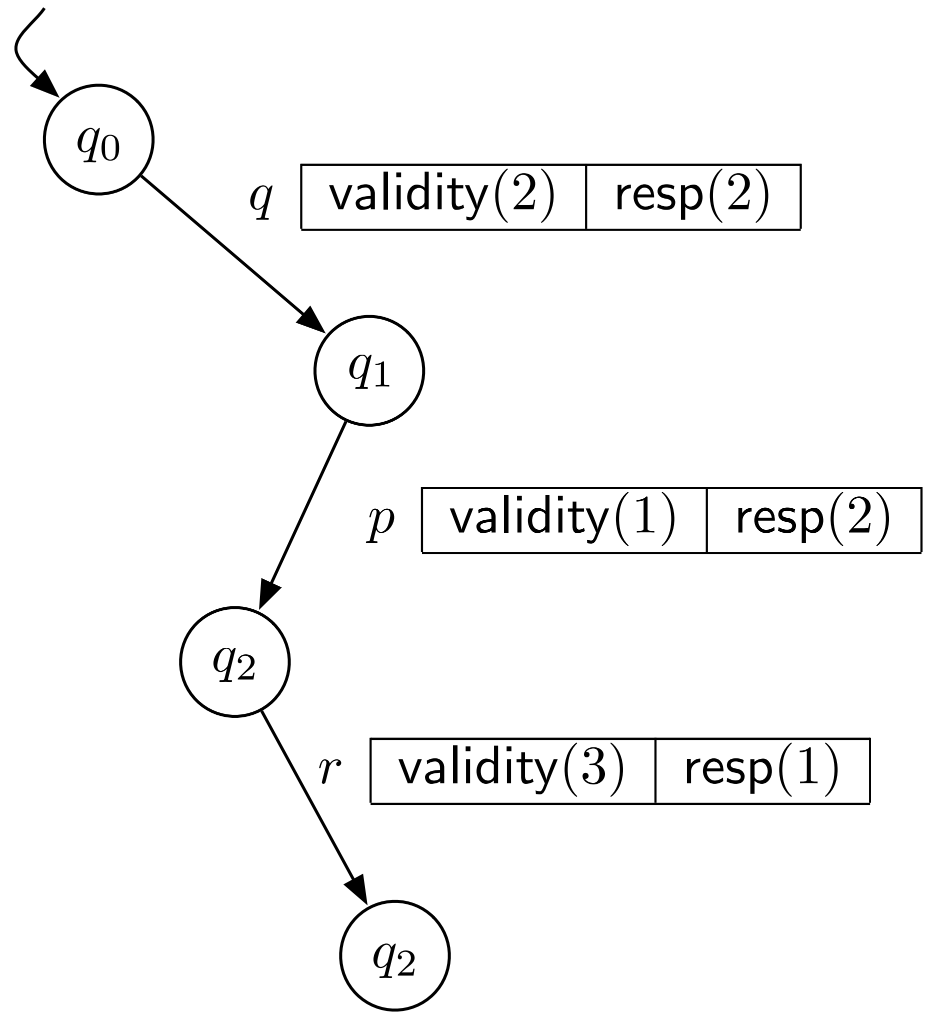
- An object specifies its behaviour in sequential executions, while executions may be concurrent

Is an implementation correct?

- An object specifies its behaviour in sequential executions, while executions may be concurrent

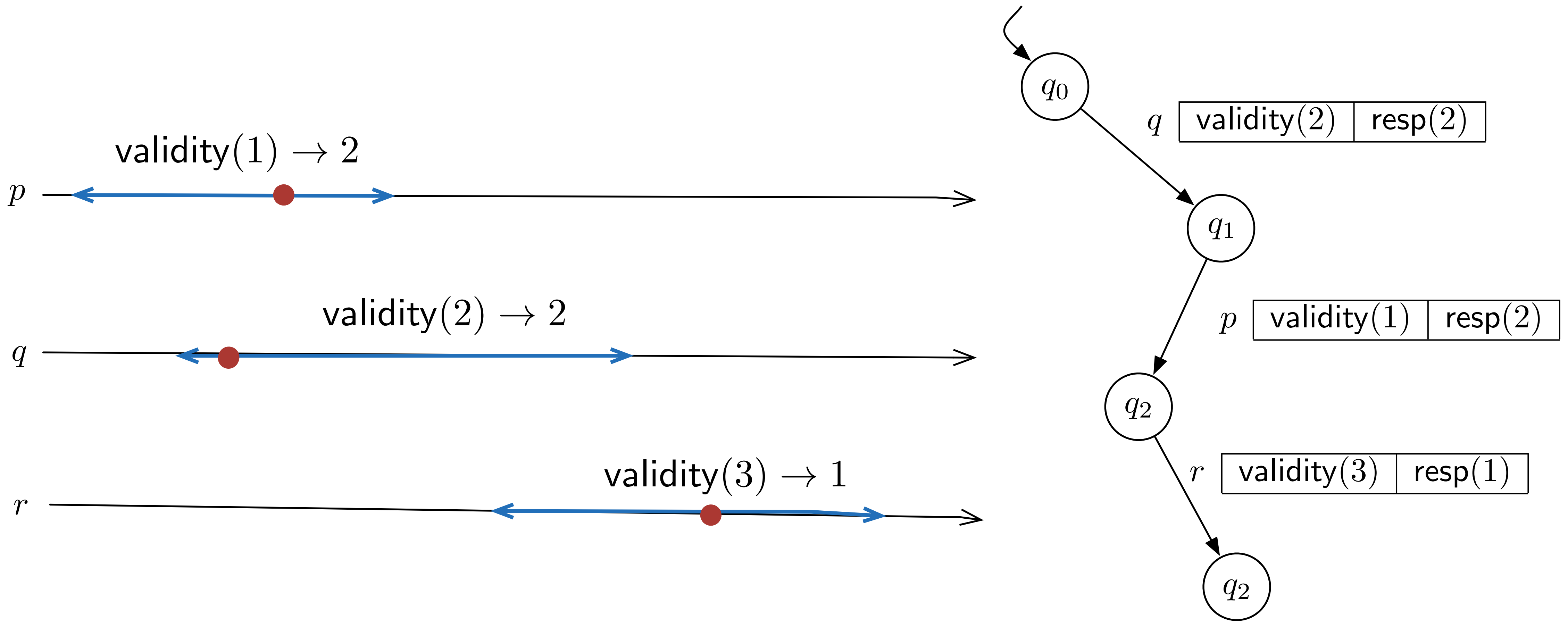
Is an implementation correct?

- A **correctness** implementation notion is needed for concurrent executions



Linearizability

- Operations seem to occur at a point, in between invocation and response,
- i.e., they can be transformed to a valid sequential execution.



Importance of Linearizability

- . **Good properties** for the development of systems:

  Non-blocking: It never forces the system to block

  Locality: Linearizable implementations compose into a linearizable system.

Good pair!

Object: sequential specification

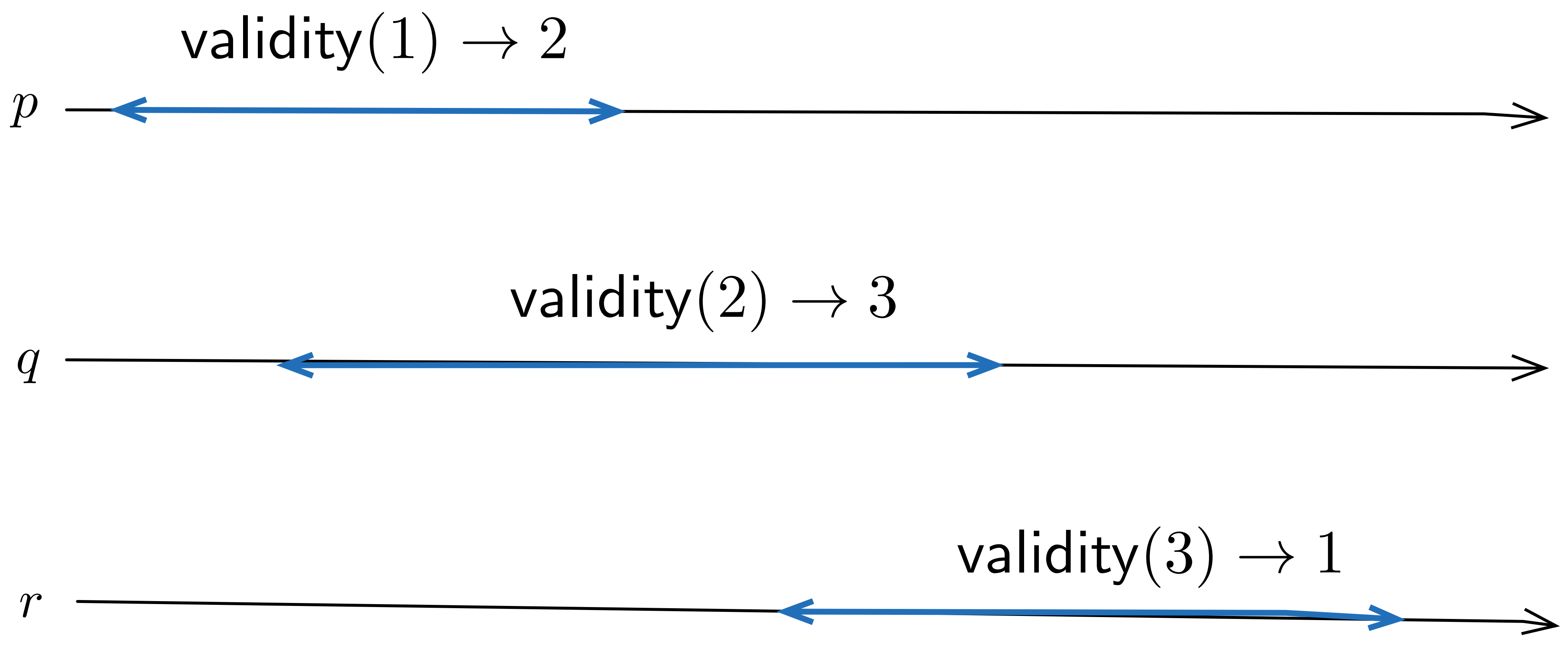
Linearizability: implementation notion

Are all distributed problems sequentially specifiable?

Are all distributed problems sequentially specifiable?

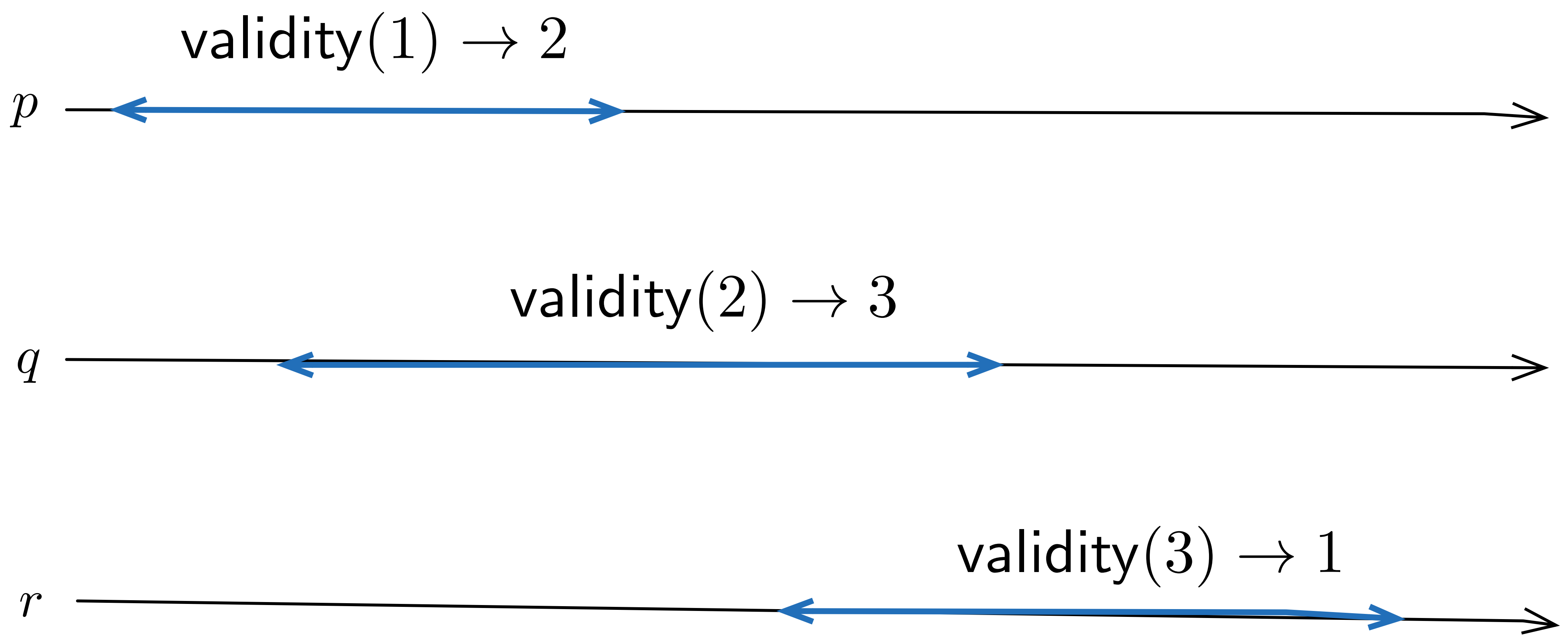
No!

Validity task



- There is a simple implementation based on read/write primitives

Validity task



$\text{validity}(1) \to 2$
p
$\text{validity}(2) \to 3$
q
$\text{validity}(3) \to 1$
r

- There d on read

not linearizable

Validity task

validity$(1) \to 2$

A linearisable implementation would be stronger-

one process would always return its own input

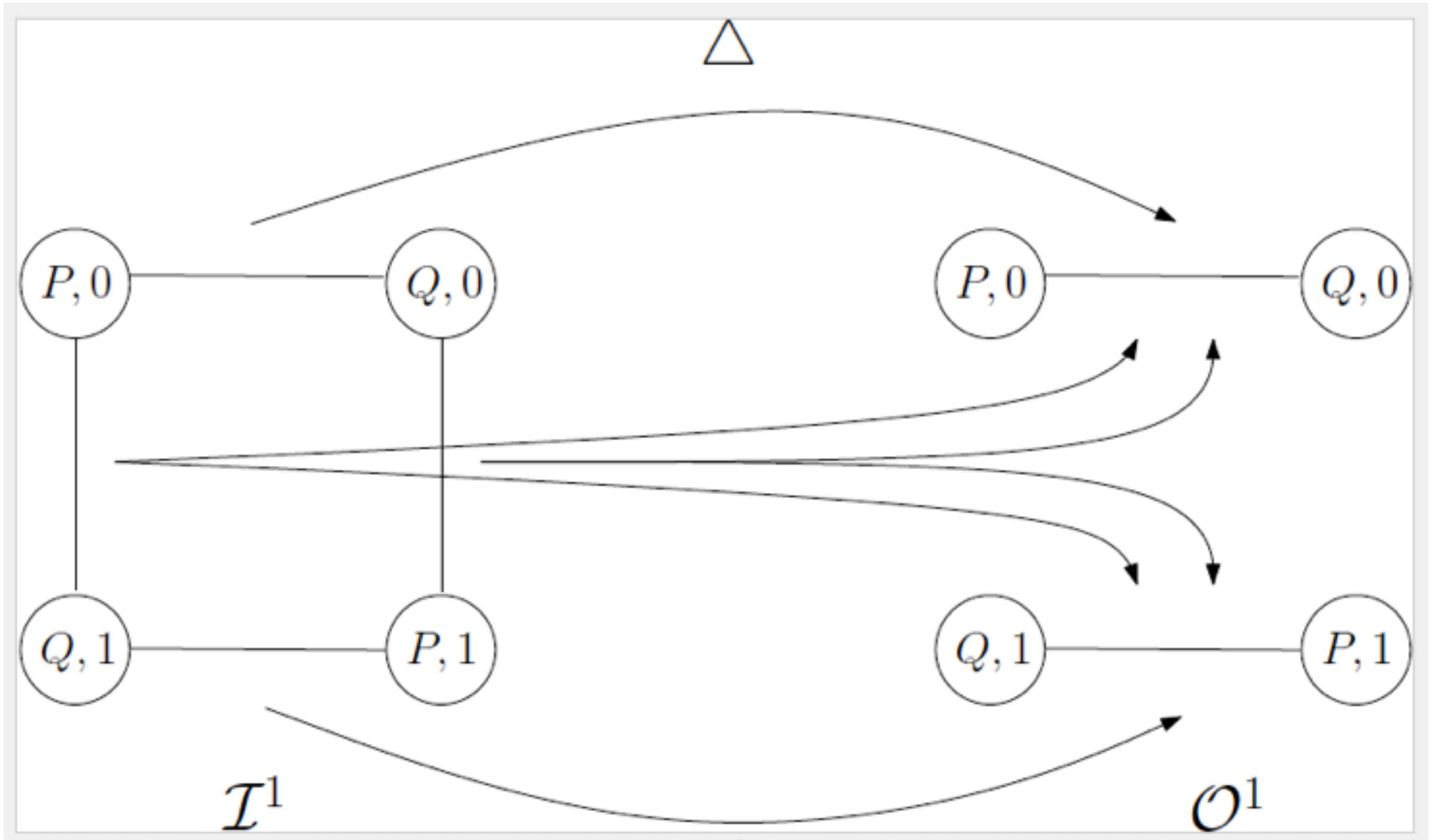
- There is a simple implementation based on read/write primitives

Examples of non-sequentially specifiable:

1. Adopt-commit (used in Paxos for safety)
2. Conflict-detection (Aspnes-Ellen)
3. Safe-consensus (weaker validity of consensus)
4. Write-snapshot and Immediate-snapshot (Asyn. Computability Theorem)
5. k-set agreement (generalization of consensus)
6. Exchanger (Java object)

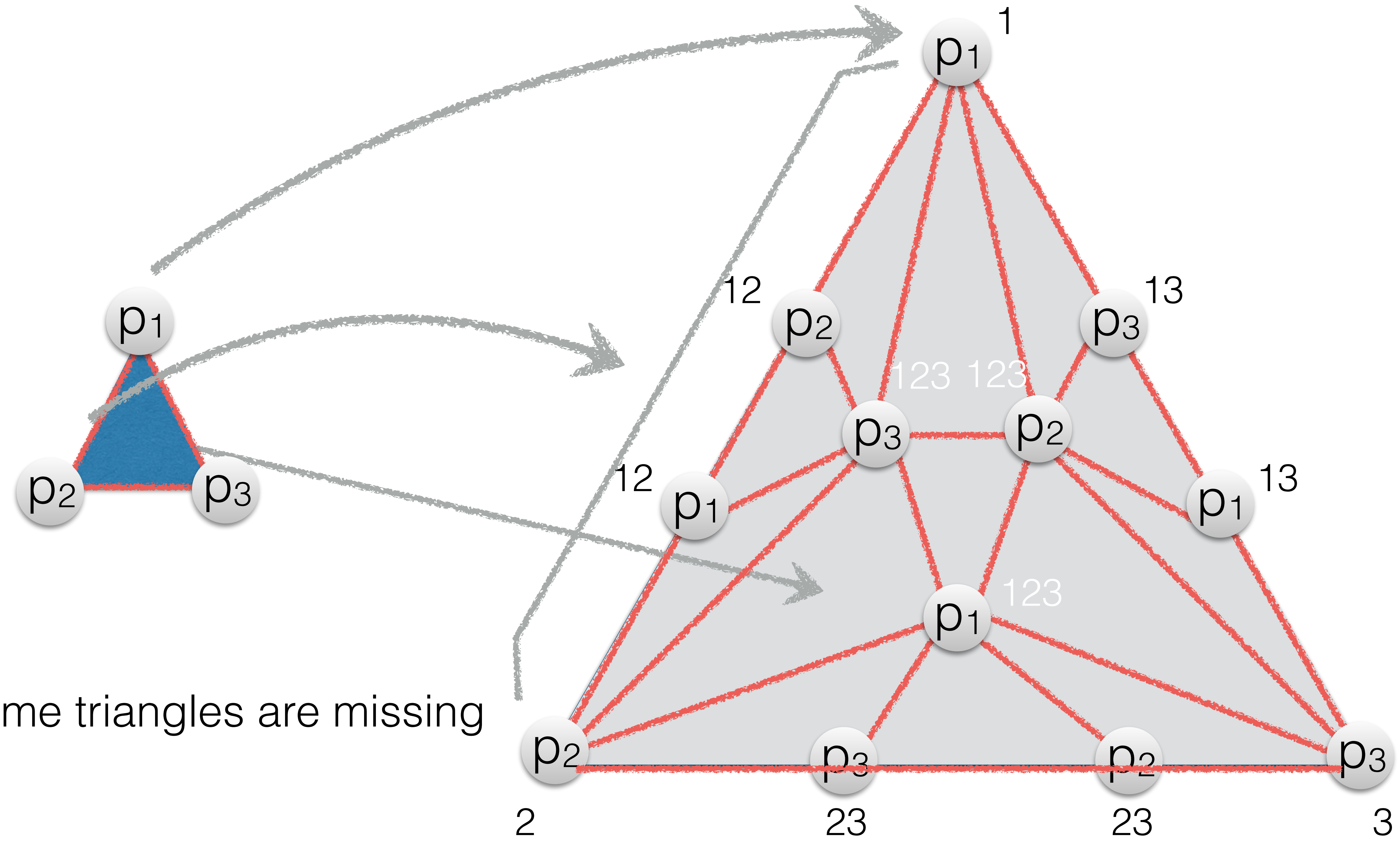
If they are not objects, what are these “distributed problems” ?

Some are tasks



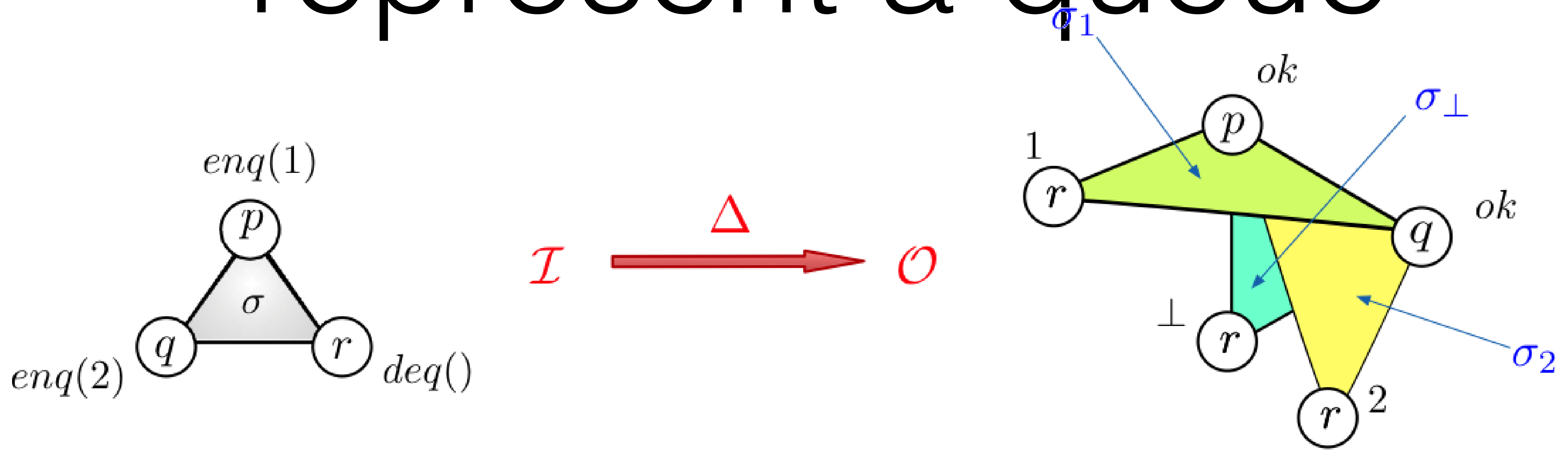
Tasks tell what might happen in presence of concurrency

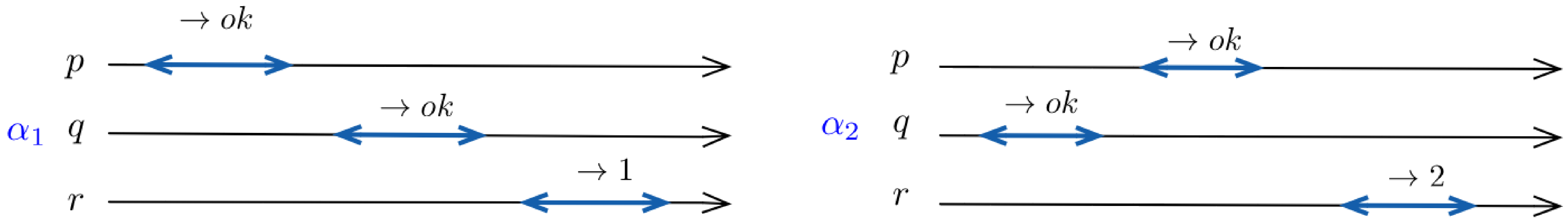
Write-Snapshot Task

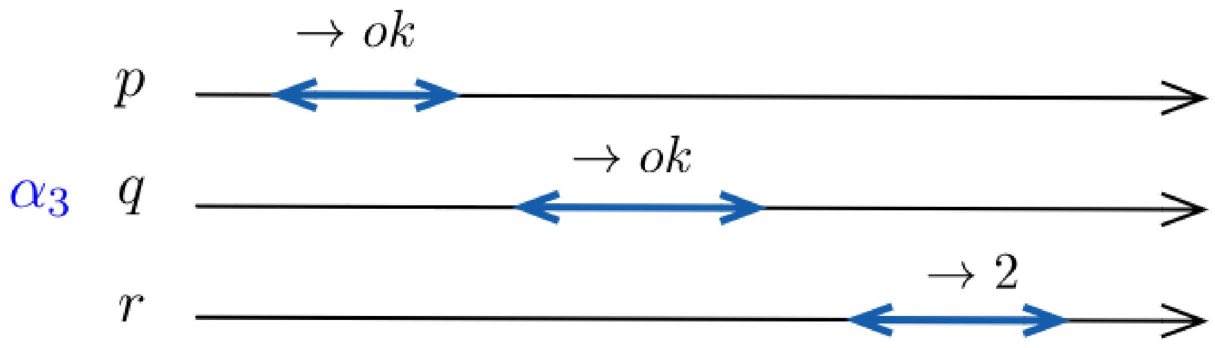


*Some triangles are missing

But not all, a task cannot represent a queue







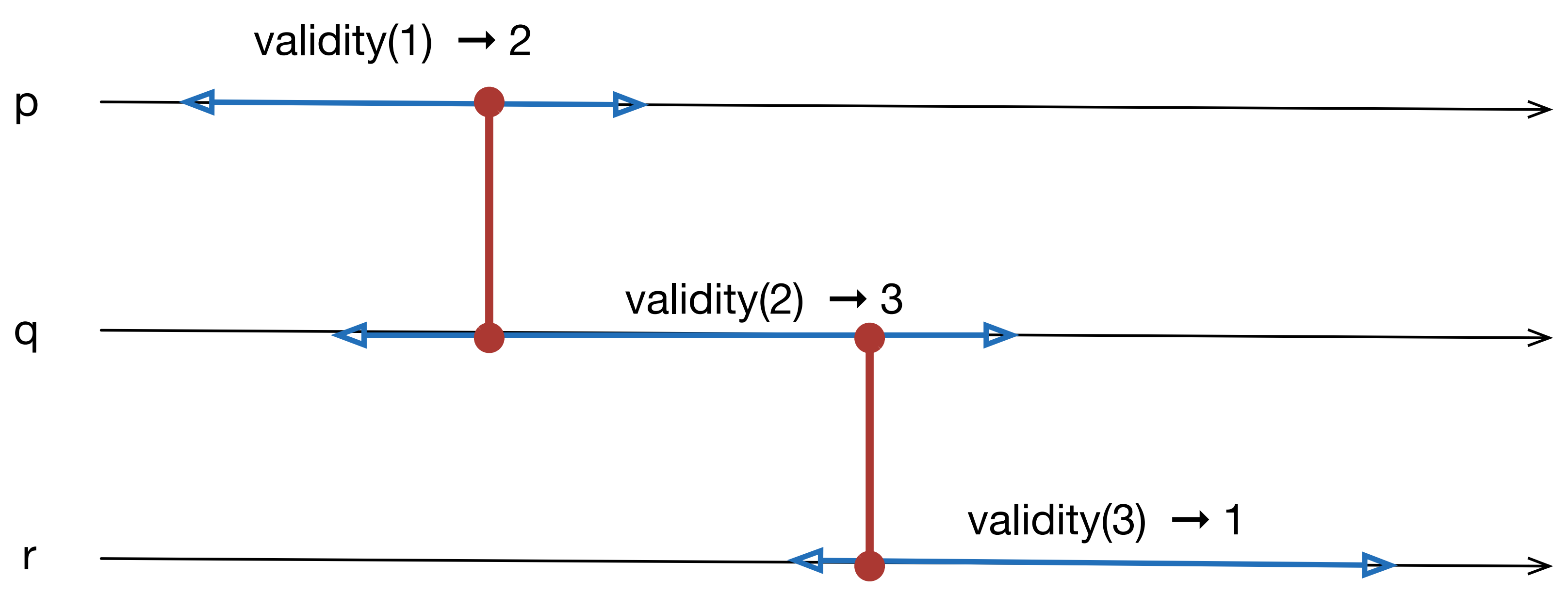
Our proposal

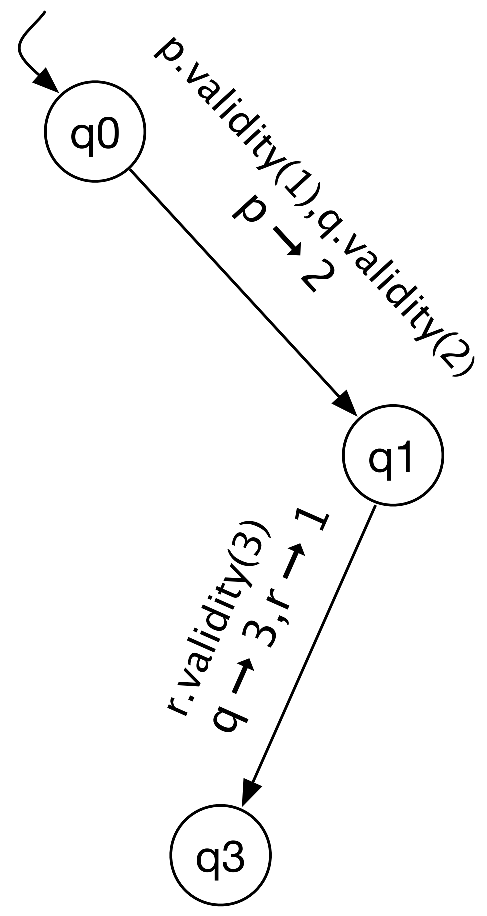
Interval-Sequential automata
and
interval-linearizability

Interval-Sequential automata

- Mealy state machine, based on sets of invocations/ responses
- If in state q and it receives as input a set of invocations I, then
- if $(R,q') \in \delta(q,I)$, may return the non-empty set of responses R and move to state q'.

Interval-Sequential Validity Object





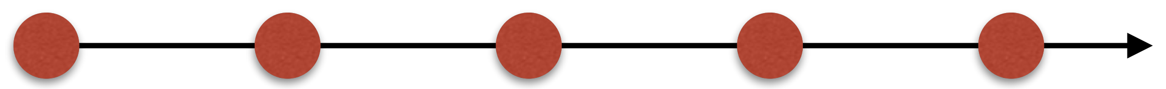
Good pair:

Interval automata and

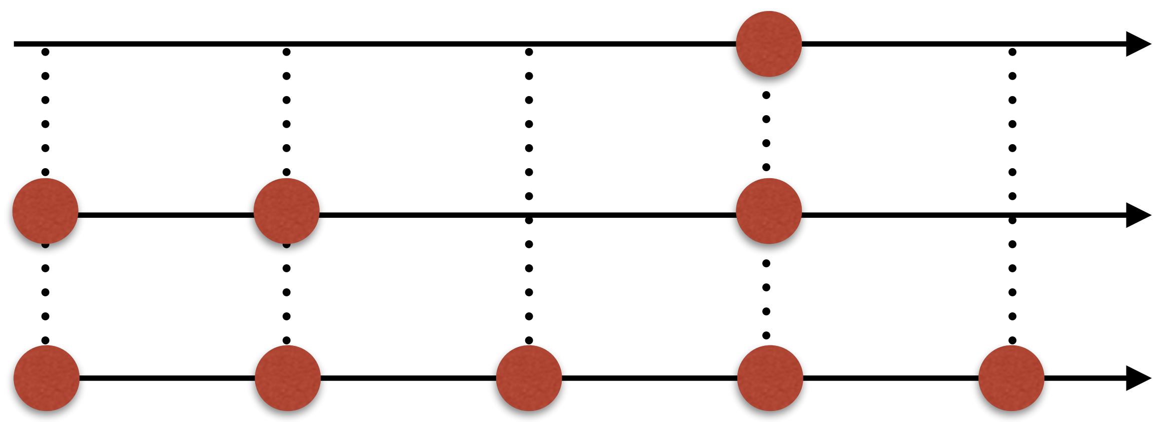
interval linearizability

From linearizability to Interval Linearizability

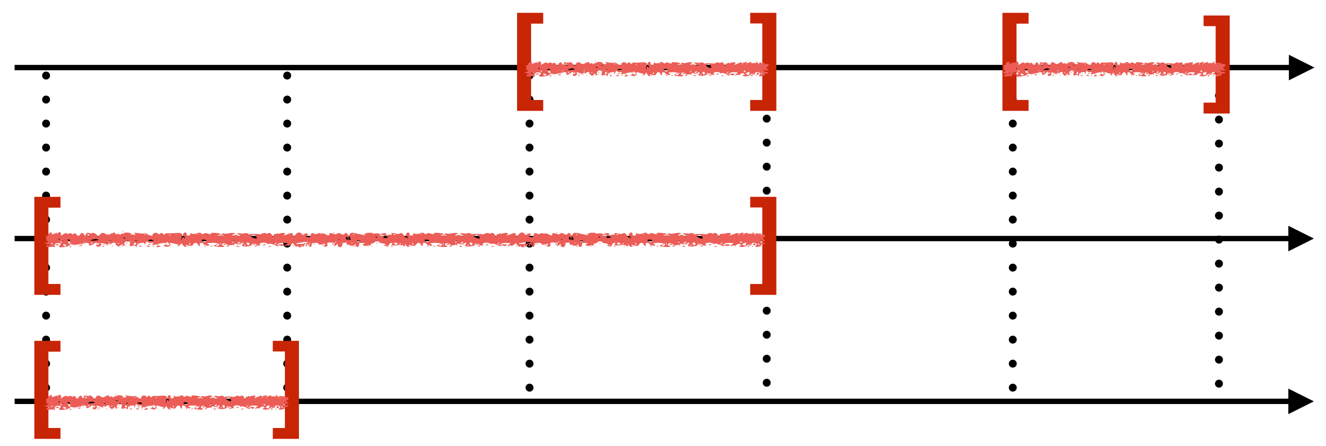
Sequential



Set sequential
(Neiger 94)



Interval sequential
(DISC 15)



Interval Linearizability Properties

- Local property (like linearizability)

An execution E is interval linearizable if and only if each object X, E|$_X$ is interval linearizable

- Non-blocking property (like linearizability)

For every interval linearizable execution E, there is an interval linearization with all ops in E completed

Completness Result

A general definition: Prefix-closed set of executions (with no restrictions, not necessarily one-shot)

Most general definition one can imagine?

For every prefix-closed set of executions, there is a IS object that model the set

Conclusion

$$\begin{tikzcd}
 & \text{Execution} \arrow[dl, bend right, "\text{linearizable}"'] \arrow[d, "\text{set-linearizable}"'] \arrow[dr, "\text{interval-linearizable}"] \arrow[drr, bend left, "\text{satisfies}"] & & \\
\begin{matrix}\text{Sequential}\\\text{Object}\end{matrix} \arrow[r, phantom, "\subset"] & \begin{matrix}\text{Set-Sequential}\\\text{Object}\end{matrix} \arrow[r, phantom, "\subset"] & \begin{matrix}\text{Interval-Sequential}\\\text{Object}\end{matrix} \arrow[r, phantom, "\overset{\textit{one-shot}}{\approx}"] & \text{Task}
\end{tikzcd}$$

- Set-based spec = multi-shot tasks = IS linearizability
- In NETYS17 extend task definition further, to model multi-shot objects
- To apply topological techniques to objects
- and object techniques to tasks, e.g. composability

Thank you !