Distributed Computing through Topology

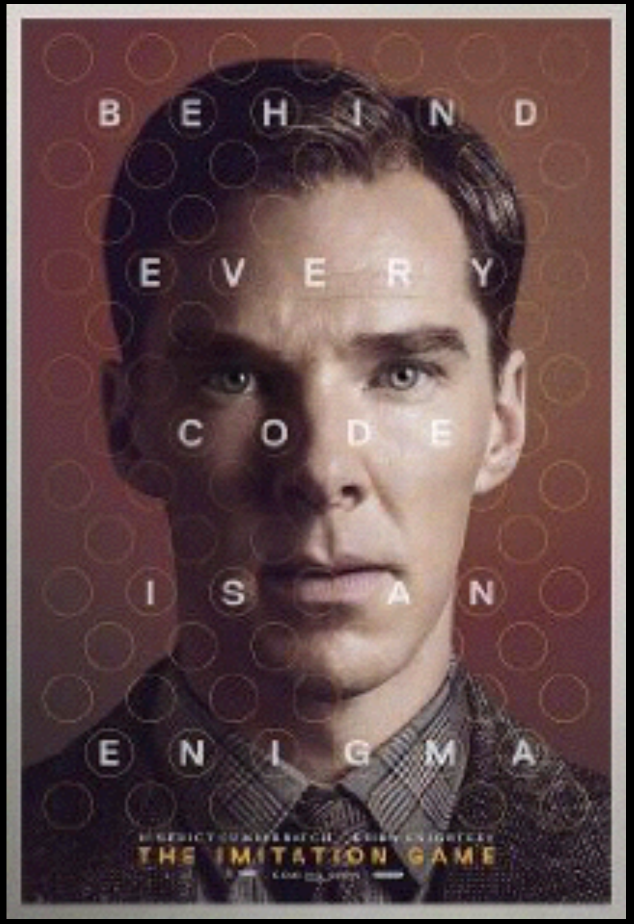
an introduction

Sergio Rajsbaum
Instituto de Matemáticas
UNAM

Sequential Computing

- Turing Machine
- model of choice for <u>theory of computation</u>

- provides a precise definition of a "mechanical procedure"



Turing Year 2012
centenary of his birth

What about concurrency?

Concurrency is everywhere

Nearly every activity in our society works as a <u>distributed system</u> made up of human and sequential computer processes

Very different from sequential computing

This revolution requires a fundamental change in how programs are written. Need new principles, algorithms, and tools

- The Art of Multiprocessor Programming
Herlihy & Shavit book

Would not seem so according to traditional views

- single-tape ≃ multi-tape TM
- interpreted as sequential computing and distributed computing differ in questions of efficiency, but not computability.
- The TM wikipedia page mentions limitations: unbounded computation (OS) and concurrent processes starting others

Why concurrency is different ?

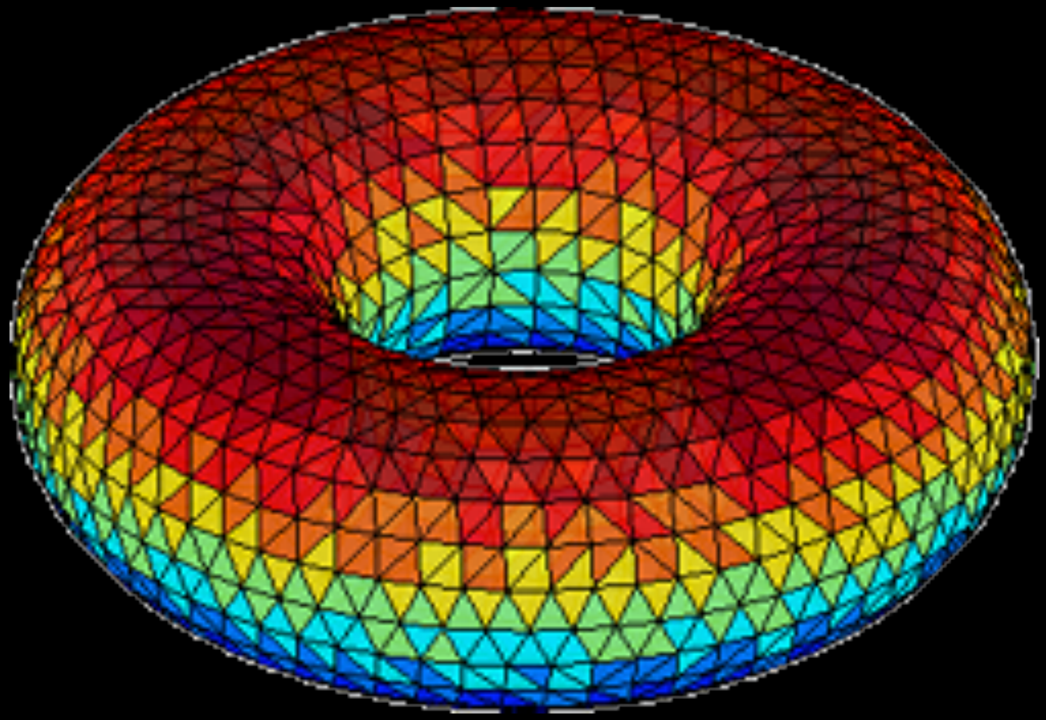
Distributed systems are subject to failures and timing uncertainties, properties not captured by classical multi-tape models.

Processes have partial information about the system state

- Even if each process is more powerful than a Turing machine
- and abstracting away the communication network (processes can directly talk to each other)

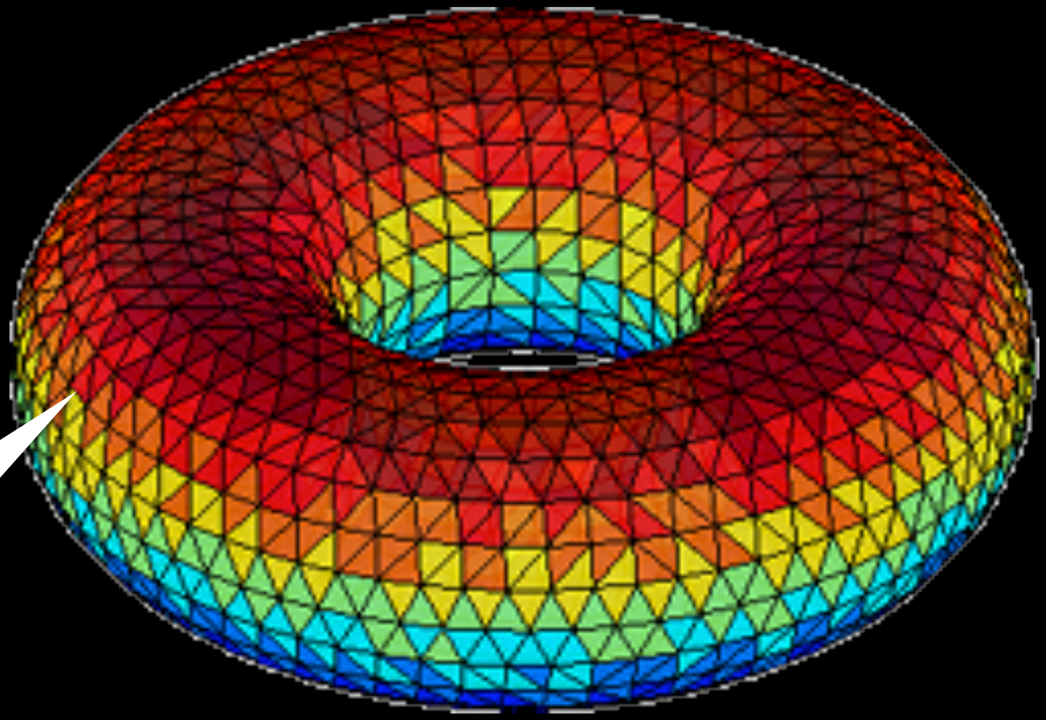
Topology

Placing together all these views yields a simplicial complex



"Frozen" representation all possible interleavings and failure scenarios into a single, static, simplicial complex

Topology



Each simplex is
an interleaving

views label vertices
of a simplex

Topological invariants

- Preserved as computation unfolds
- Come from the nature of the faults and asynchrony in the system
- They determine what can be computed, and the complexity of the solutions

Short History

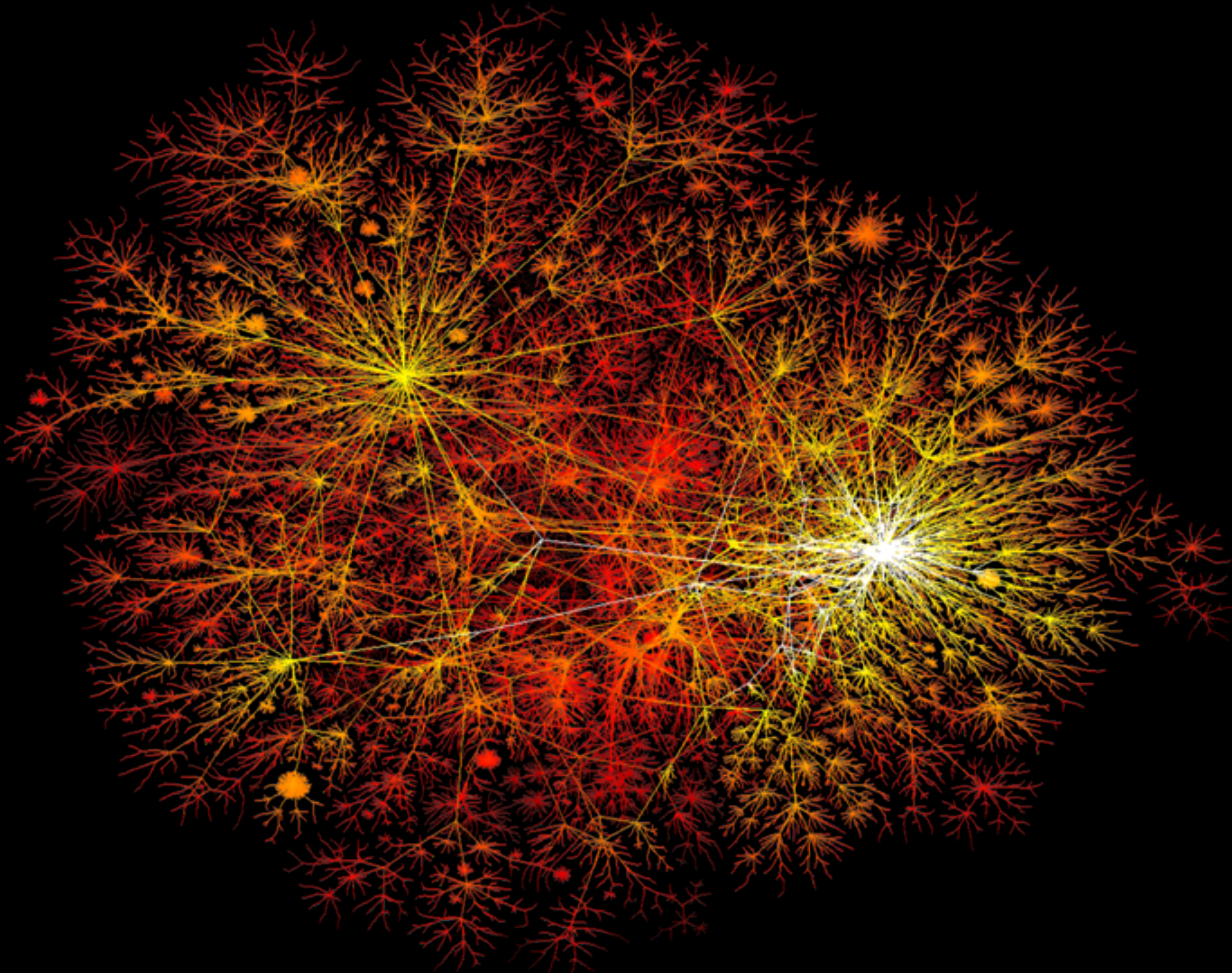
Discovered in PODC 1988 when only 1 process may crash (dimension=1) by Biran, Moran and Zaks, after consensus FLP impossibility of PODS 1983

Generalized in 1993:

- Three STOC papers by Herlihy, Shavit, Borowski, Gafni, Saks, Zaharoughlu
    - and dual approach by Eric Goubault in 1993!

Distributed Computing through Combinatorial Topology, Herlihy, Kozlov, Rajsbaum, Elsevier 2014

What would a theory of *distributed* computing be?



Distributed systems...

- Individual sequential processes
- Cooperate to solve some problem
- By message passing, shared memory, or any other mechanism

Many kinds

- Multicore, various shared-memory systems
- Internet
- Interplanetary internet
- Wireless and mobile
- cloud computing, etc.

… and topology

Many models, appear to have little in common besides the common concern with complexity, failures and timing.

Combinatorial topology provides a common framework that unifies these models.

Theory of distributed computing research

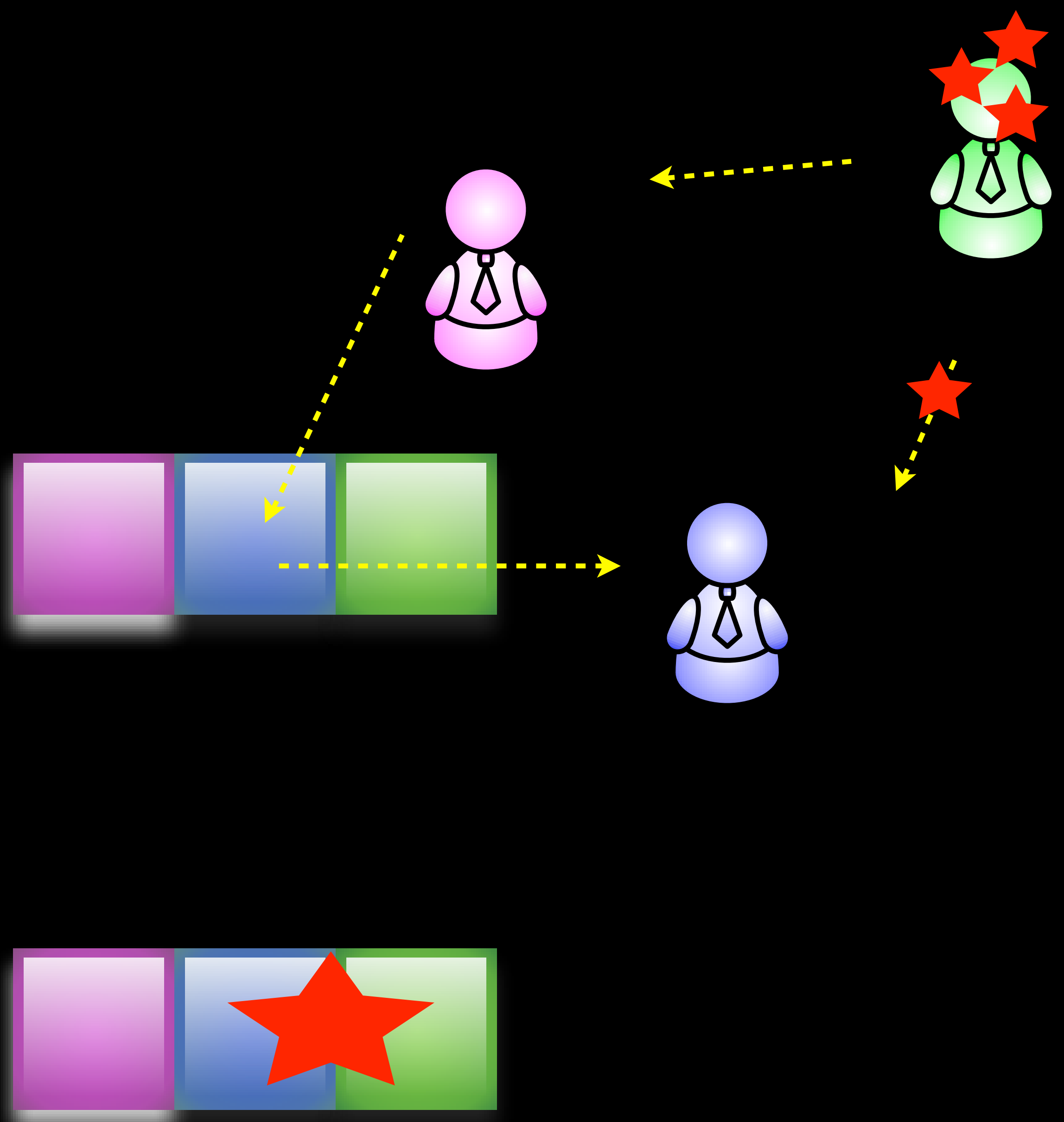
- Models of distributed computing systems: communication, timing, failures, which models are central?
- Distributed Problems:
one-shot task, long-lived tasks, verification, graph problems, anonymous,…
- Computability, complexity, decidability
- Topological invariants:
(a) how are related to failures, asynchrony, communication, and (b) techniques to prove them
- Simulations and reductions

A “universal” distributed computing model

(a Turing Machine for DC)

Ingredients of a model

- processes
- communication
- failures



Once we have a “universal” model, how to study it?

single-reader/single-writer

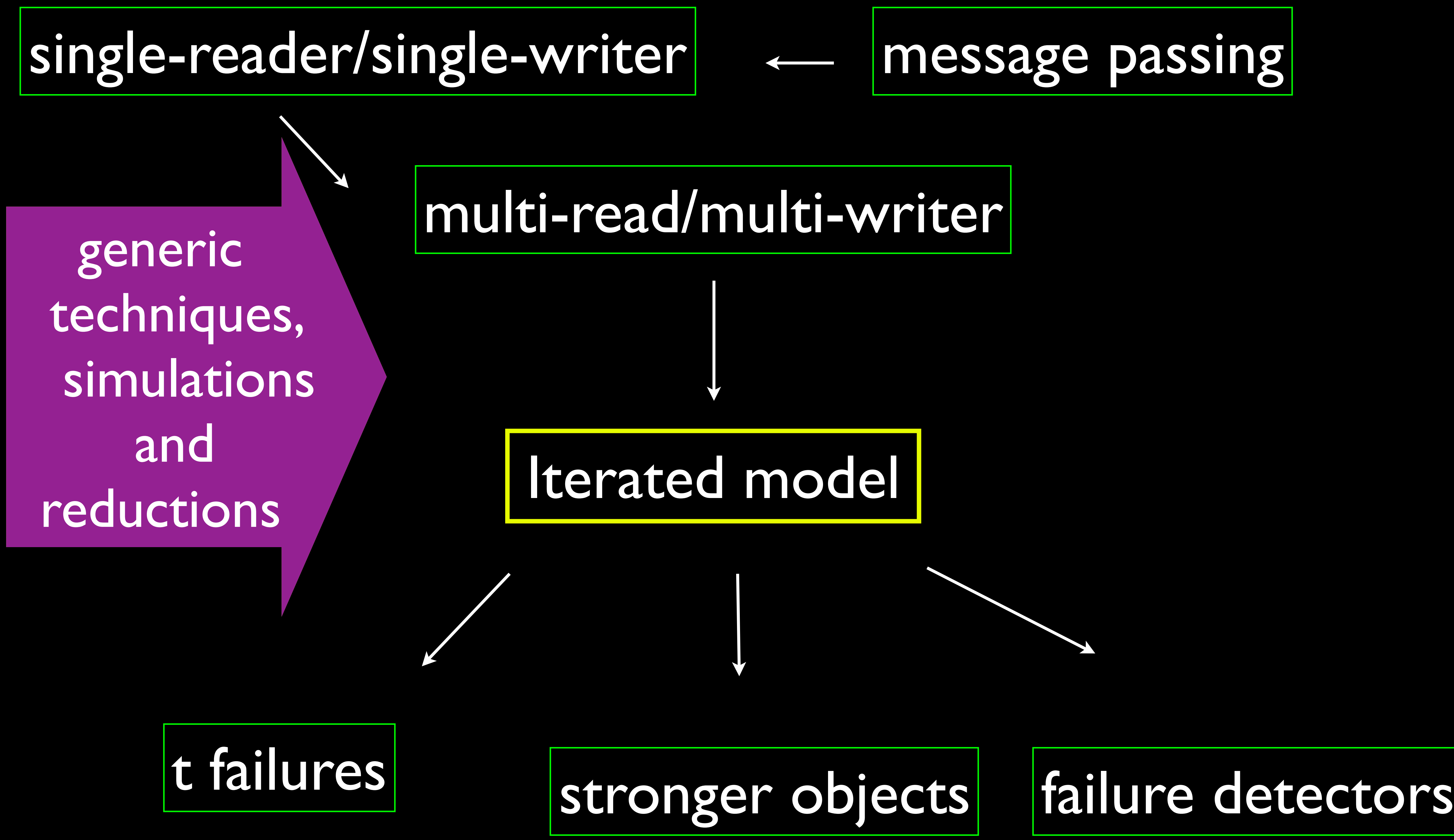
message passing

multi-read/multi-writer

t failures

stronger objects

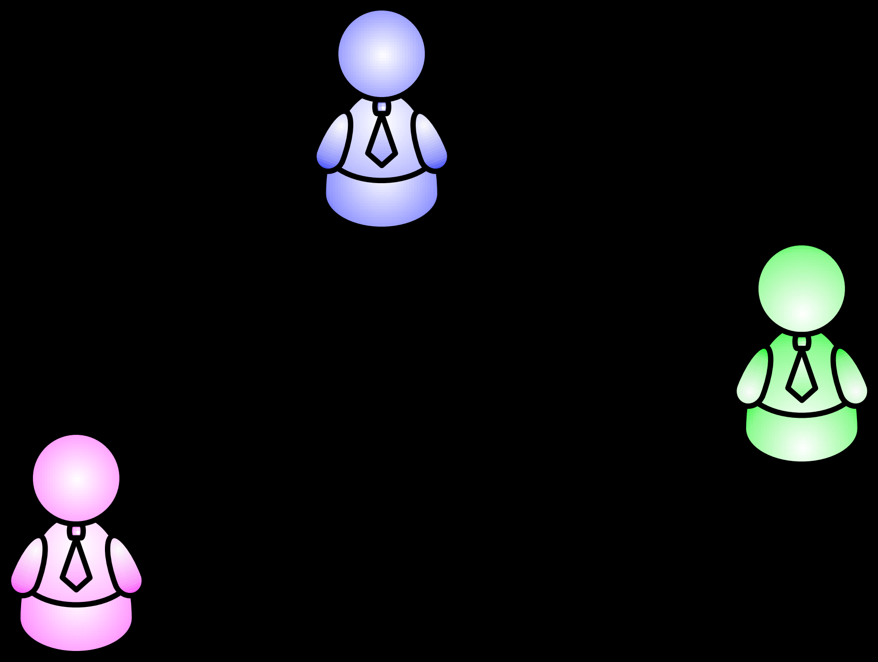
failure detectors


single-reader/single-writer
message passing
generic techniques, simulations and reductions
multi-read/multi-writer
Iterated model
t failures
stronger objects
failure detectors

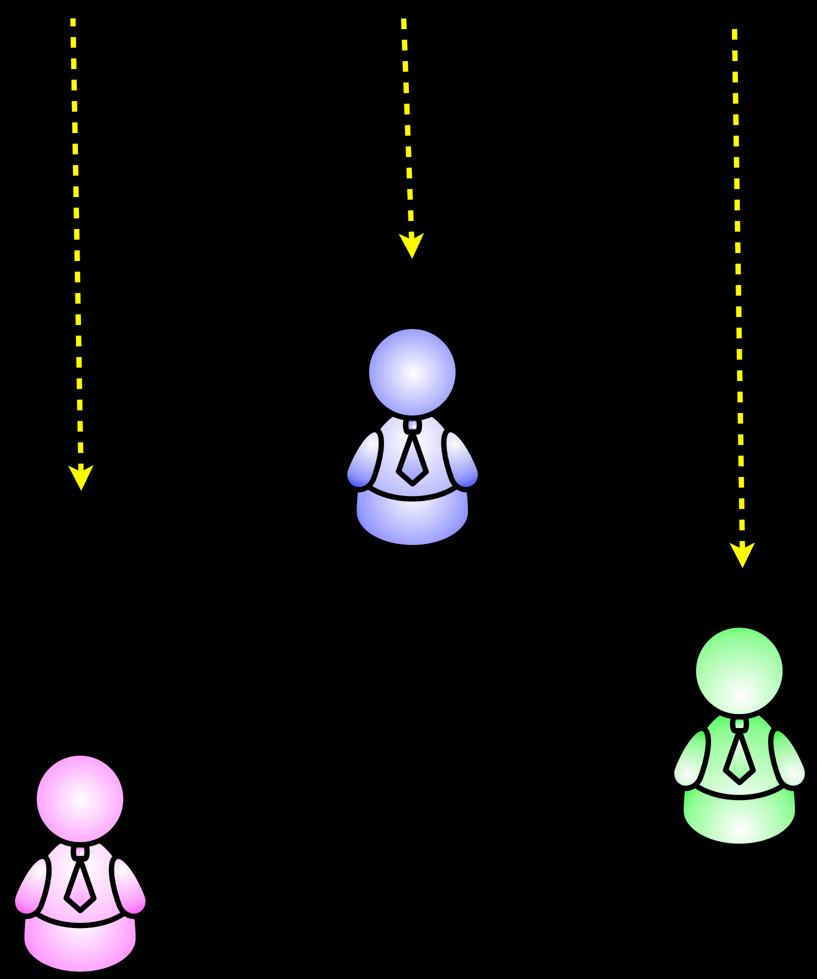
Iterated shared memory

(a Turing Machine for DC ?)

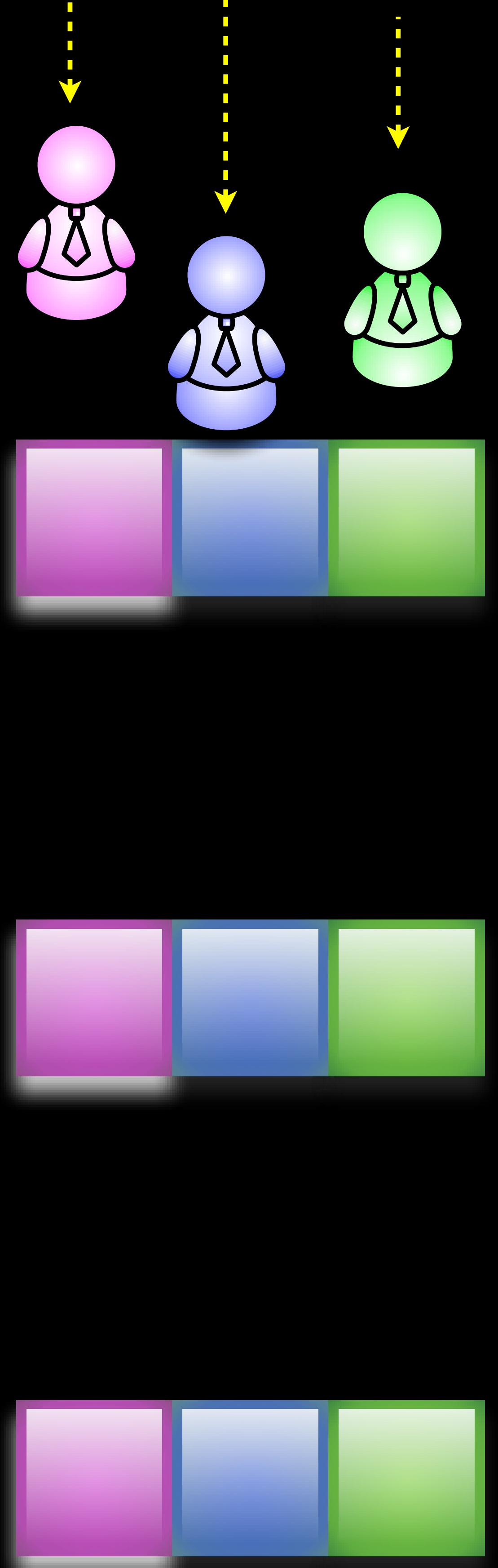
n Processes



asynchronous, wait-free

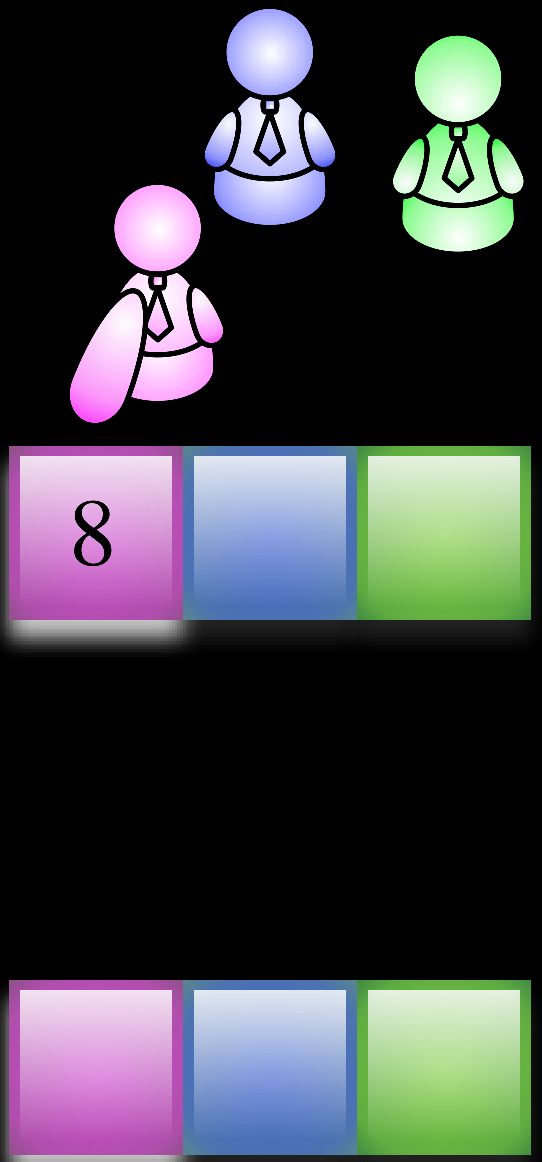


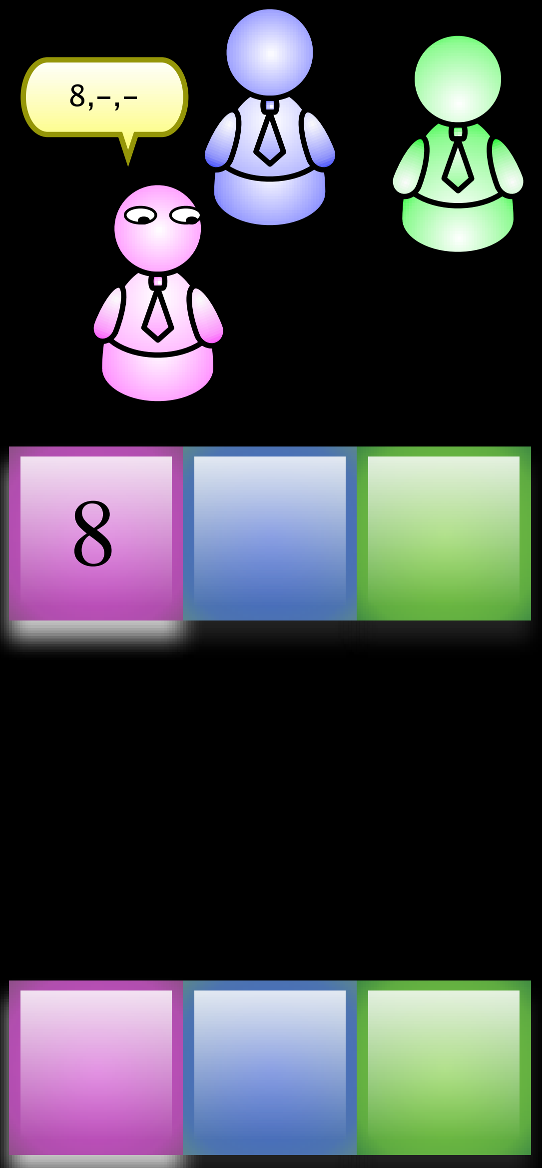
Unbounded sequence of read/write shared arrays

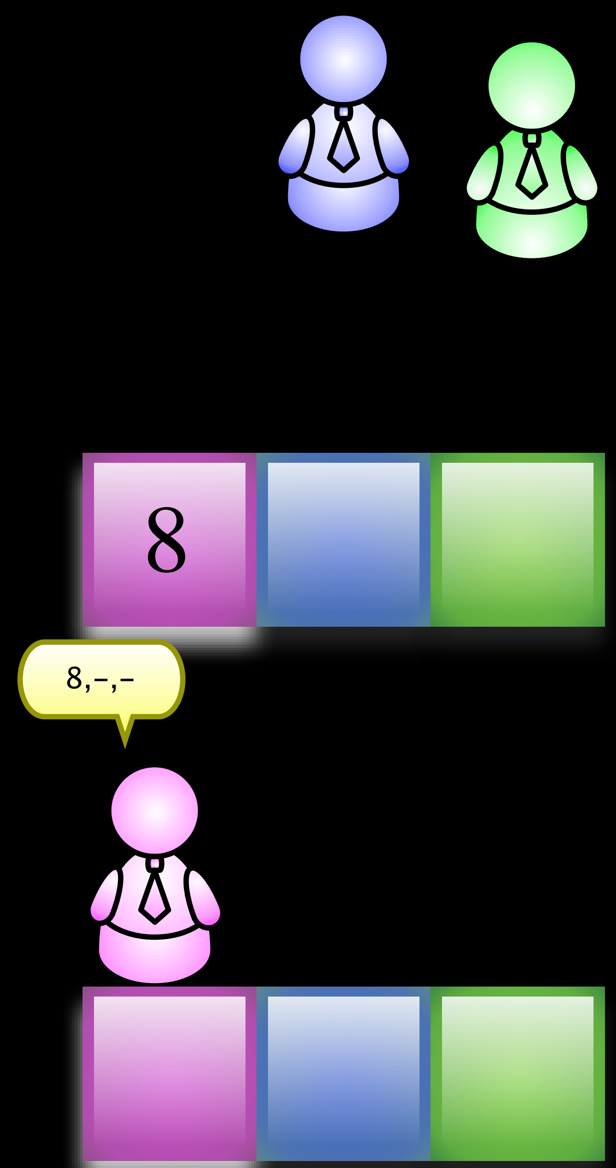


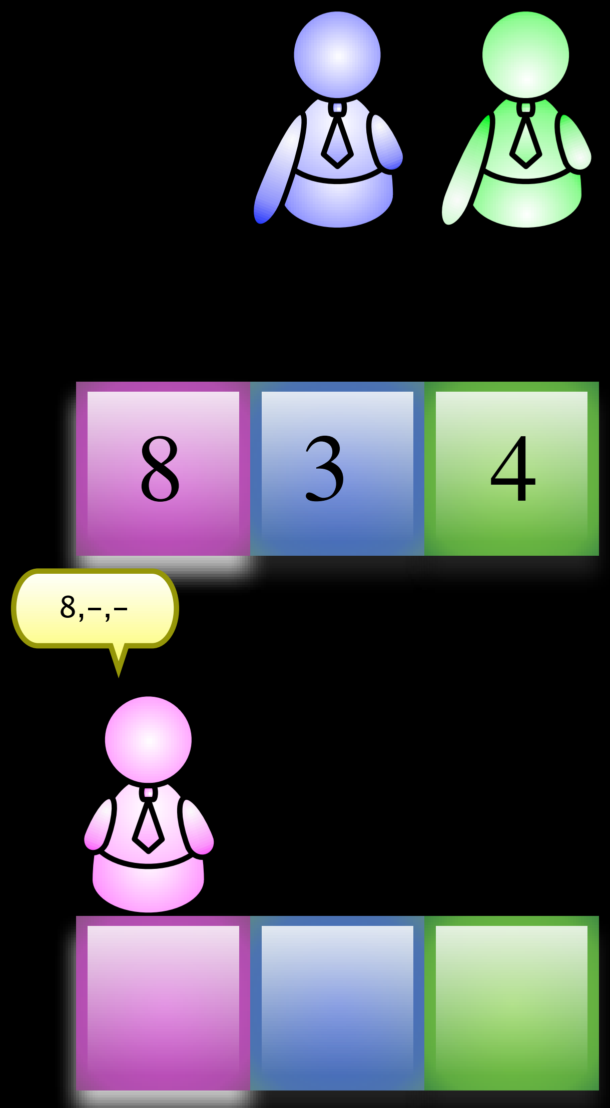
- use each one once
- in order

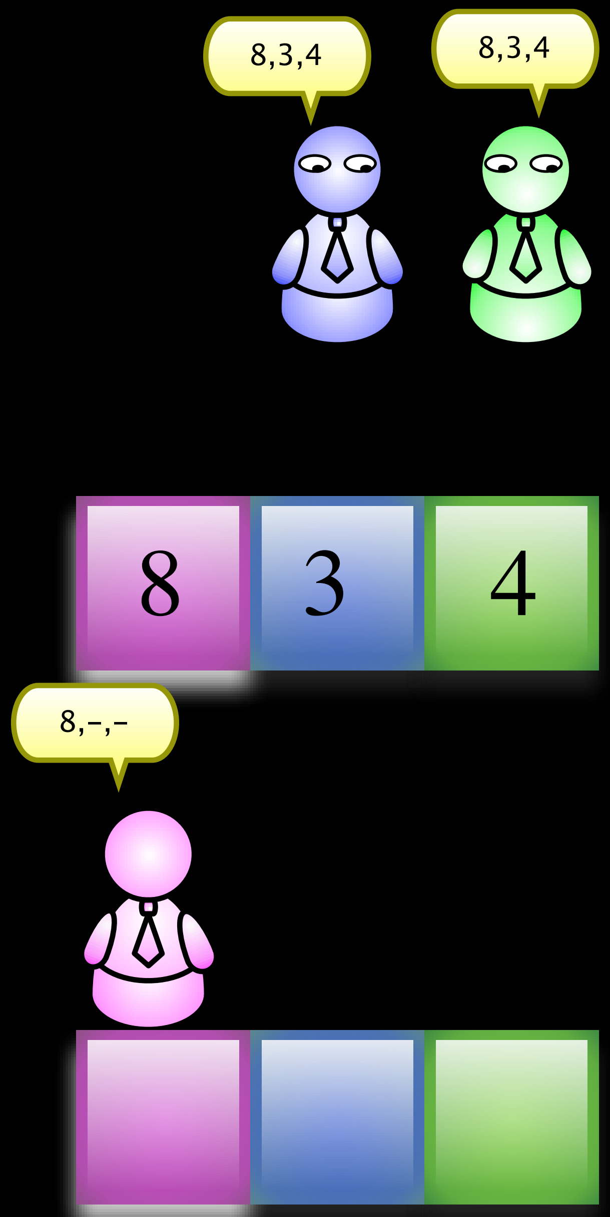
write, then read




8,–,–
8

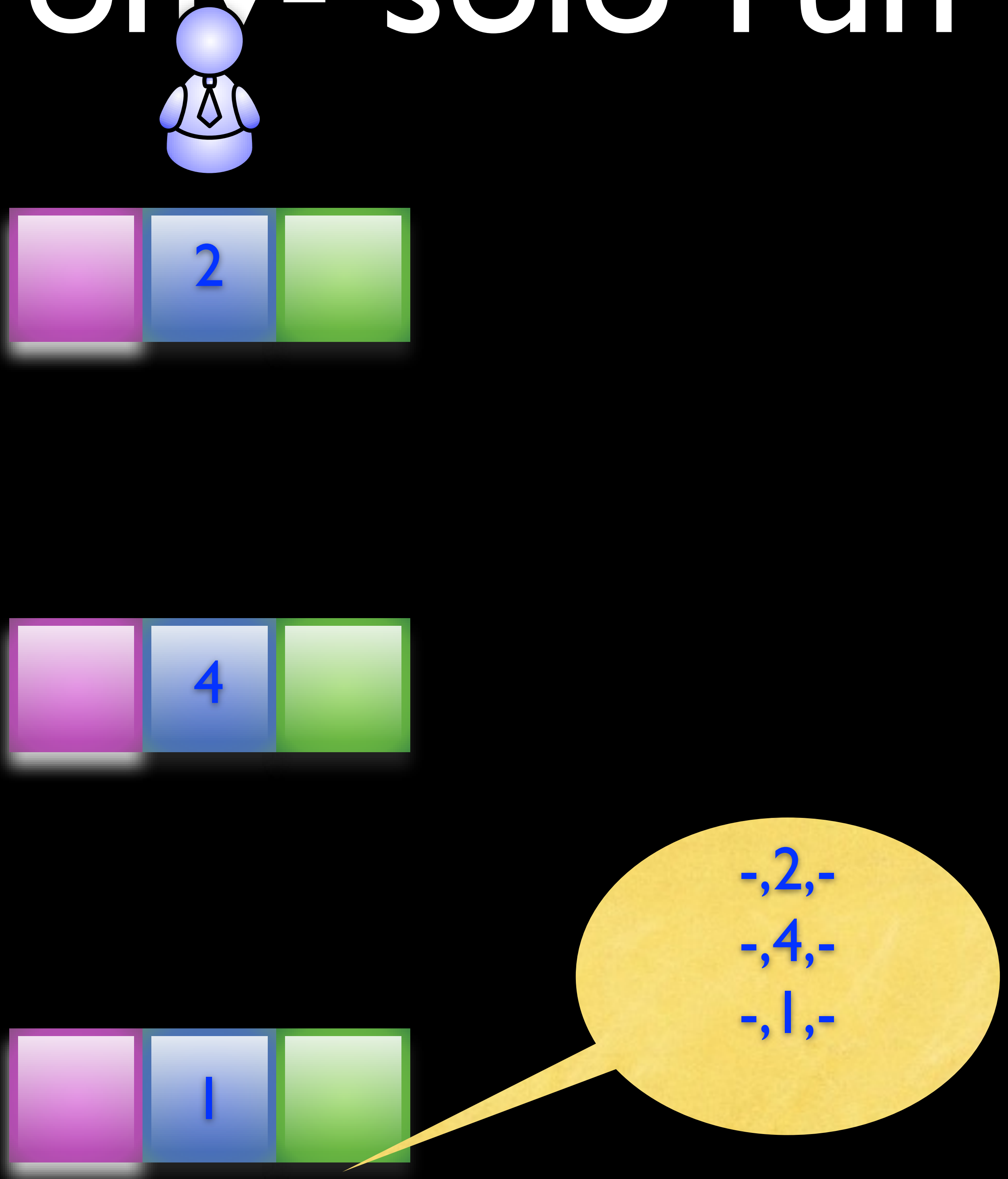

8
8,–,–


8
3
4
8,–,–


8,3,4
8,3,4
8
3
4
8,–,–

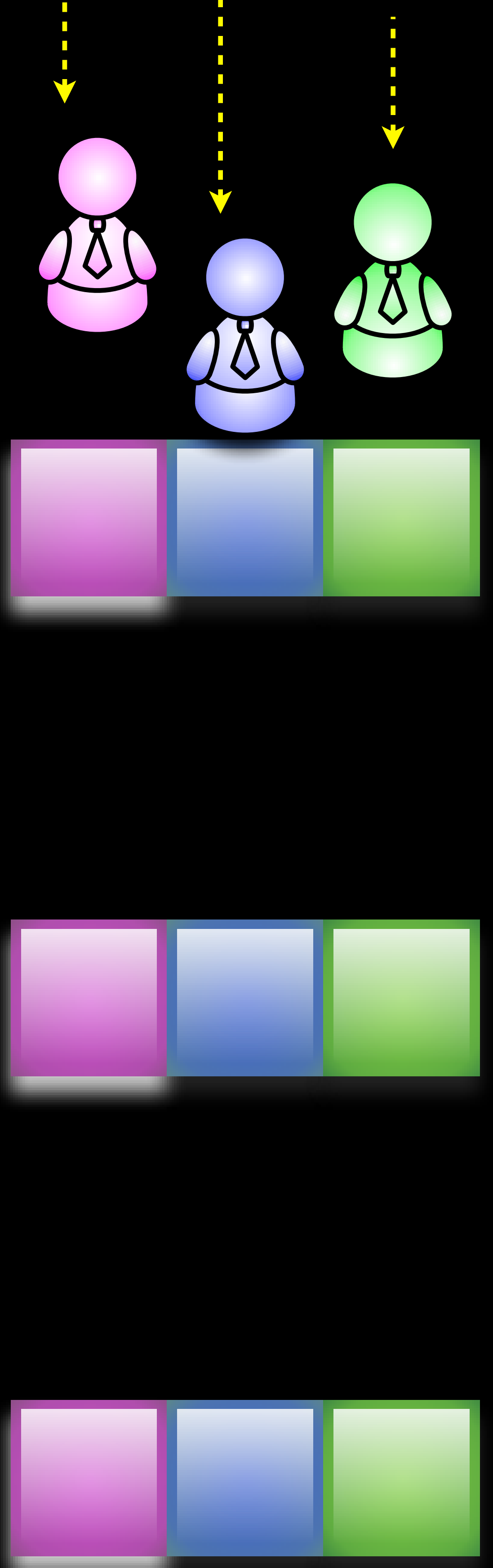

8
3
4
8,–,–
8,3,4
8,3,4

Asynchrony- solo run

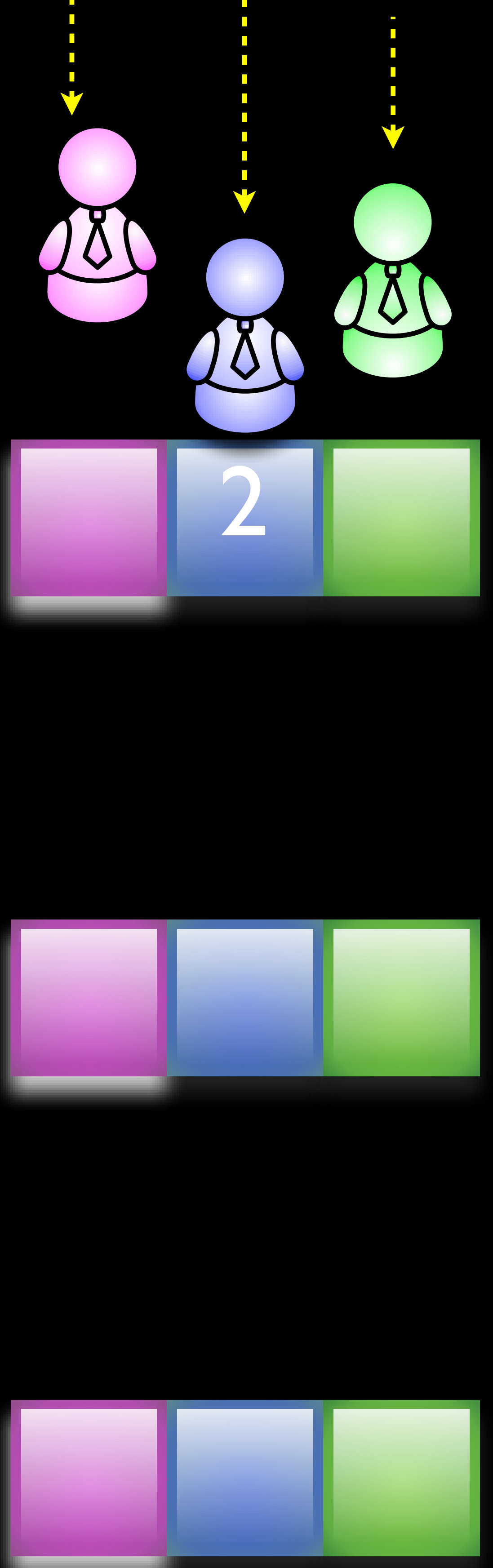


every copy is new

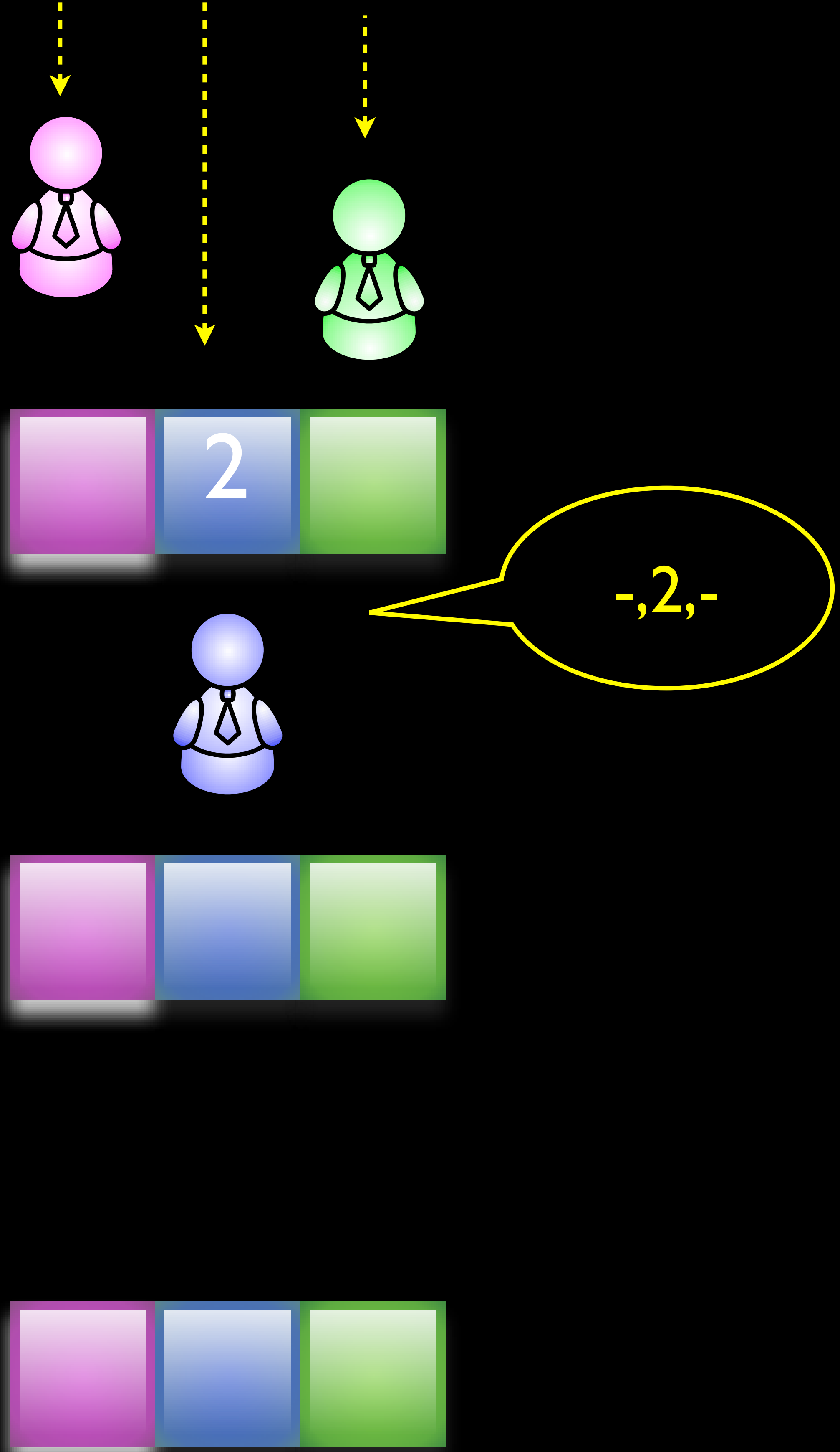
- arrive in arbitrary order
- last one sees all



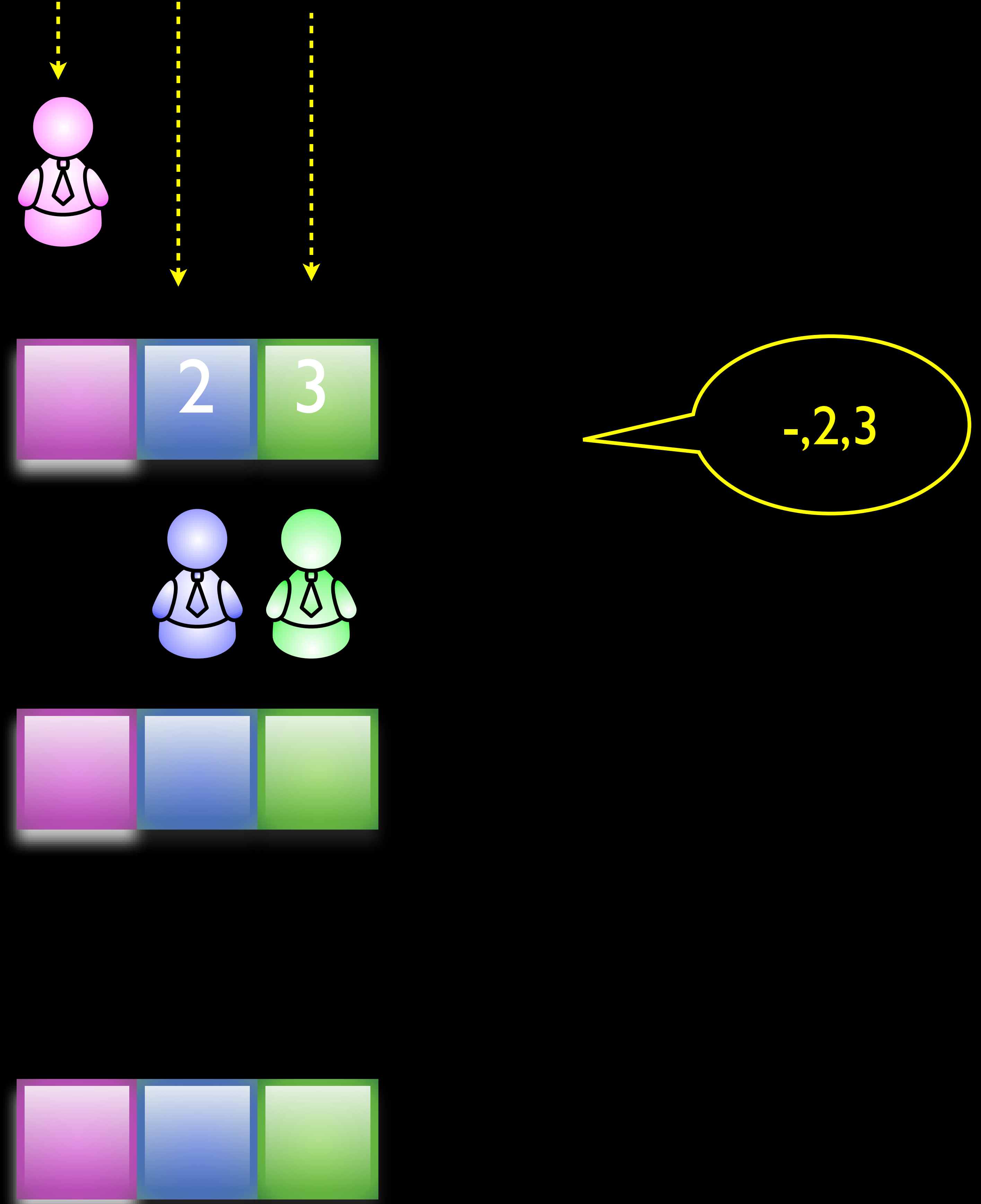
- arrive in arbitrary order
- last one sees all



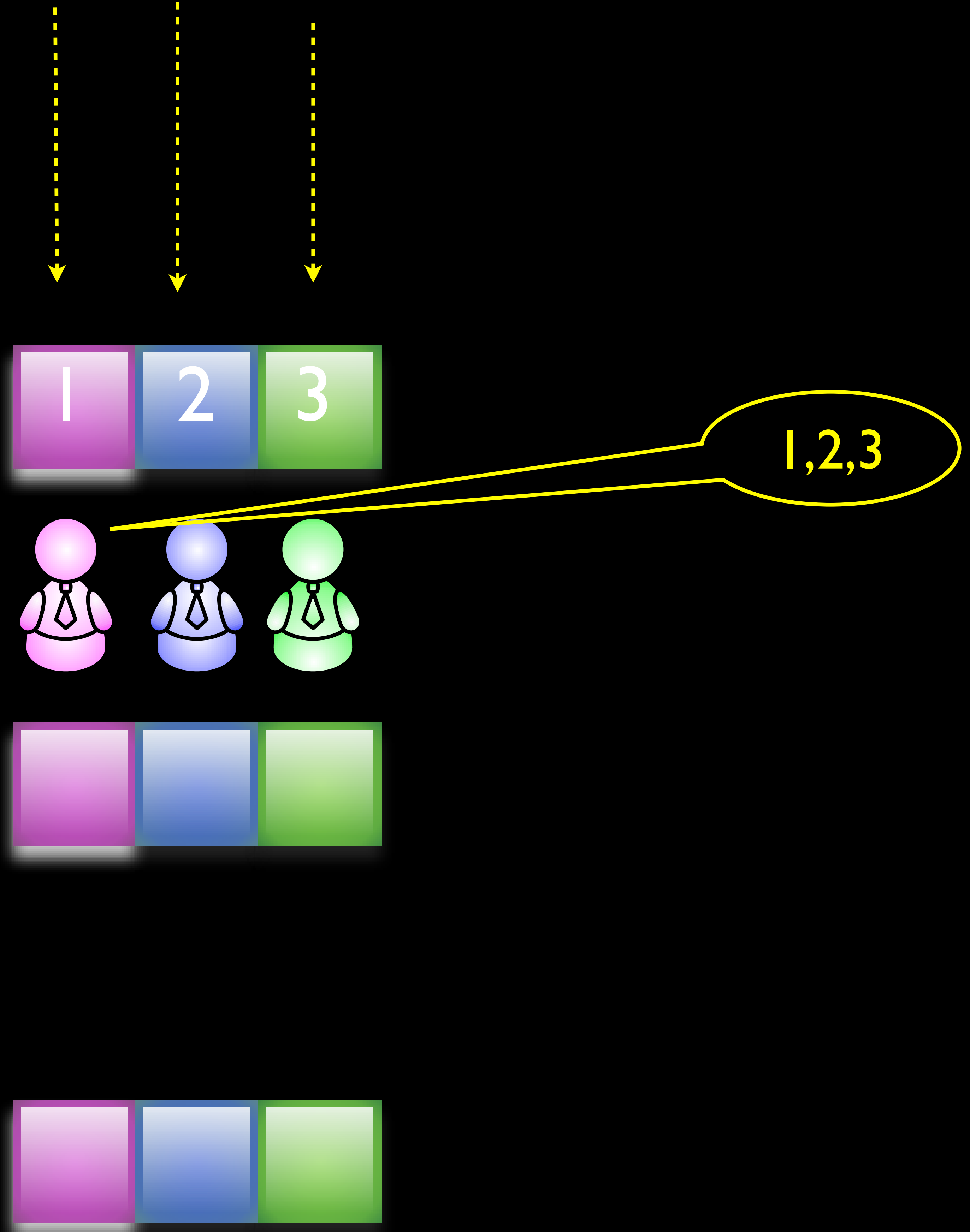
- arrive in arbitrary order
- last one sees all



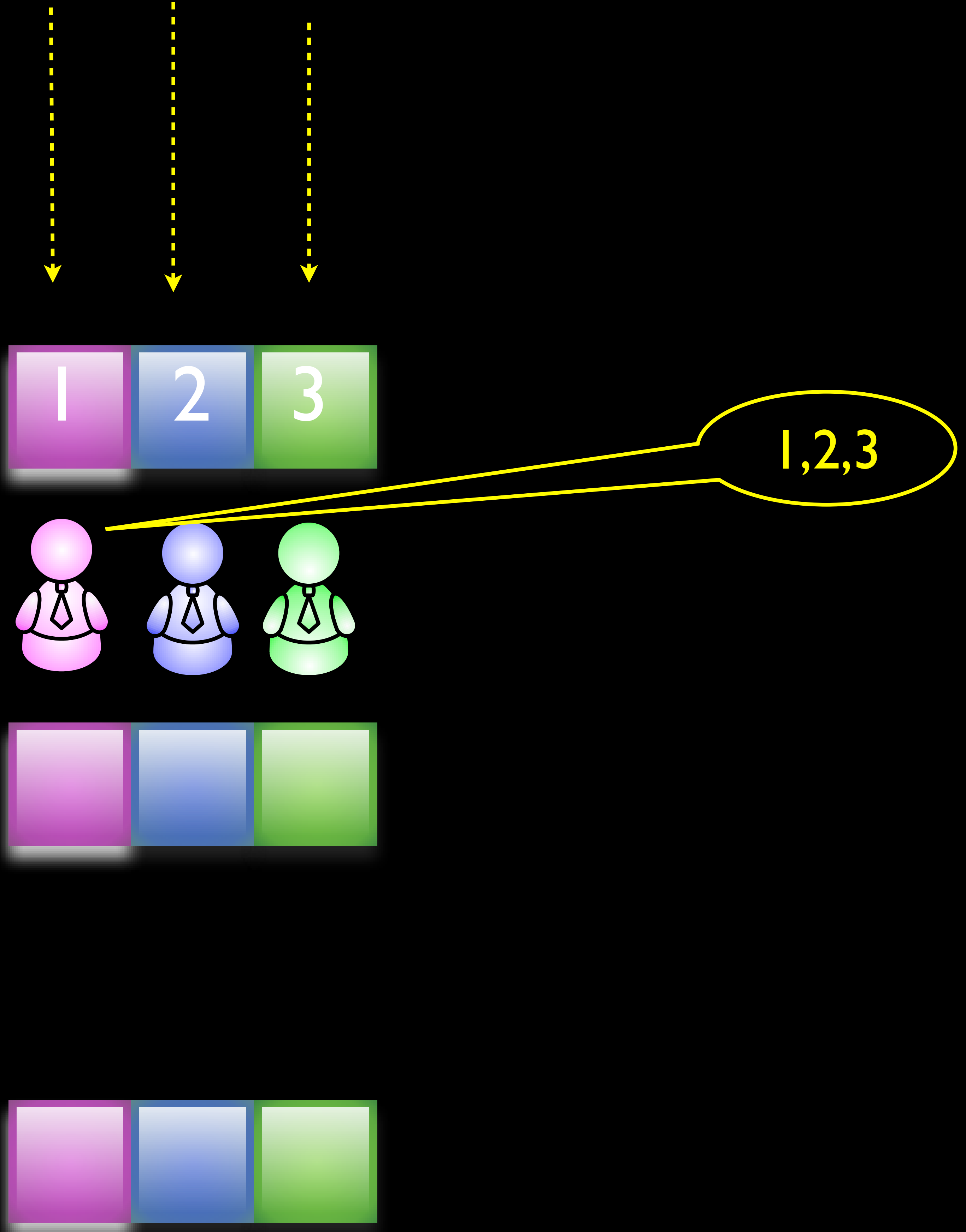
- arrive in arbitrary order
- last one sees all



- arrive in arbitrary order
- last one sees all

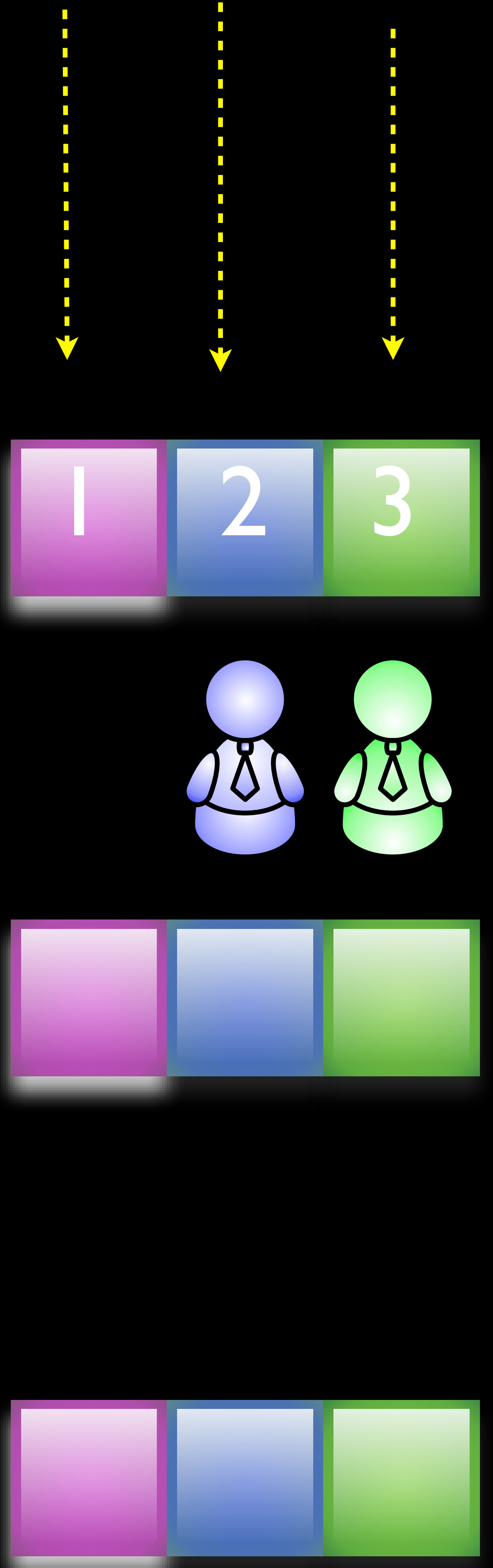


•arrive in arbitrary order
•last one sees all

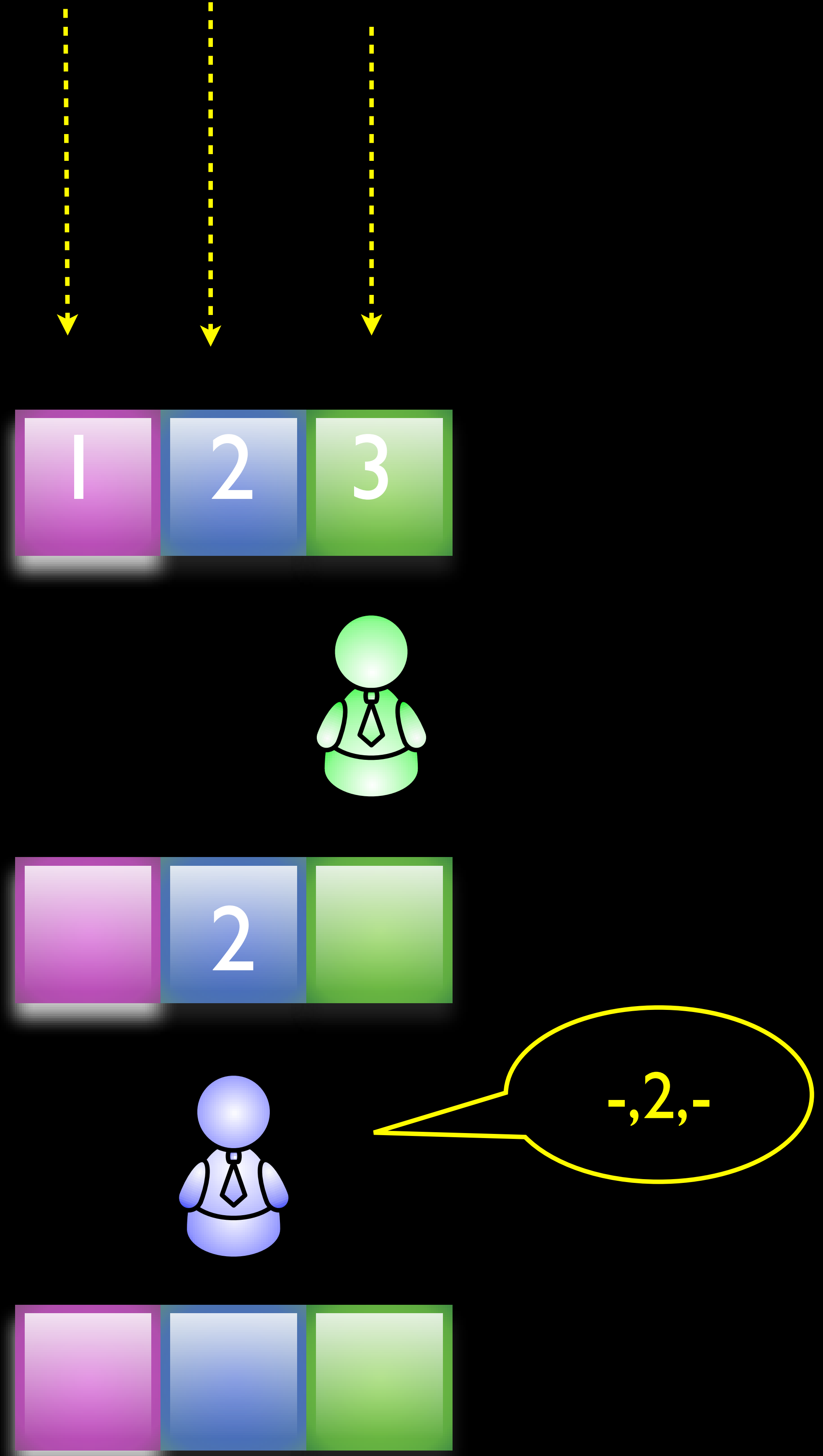


returns 1,2,3

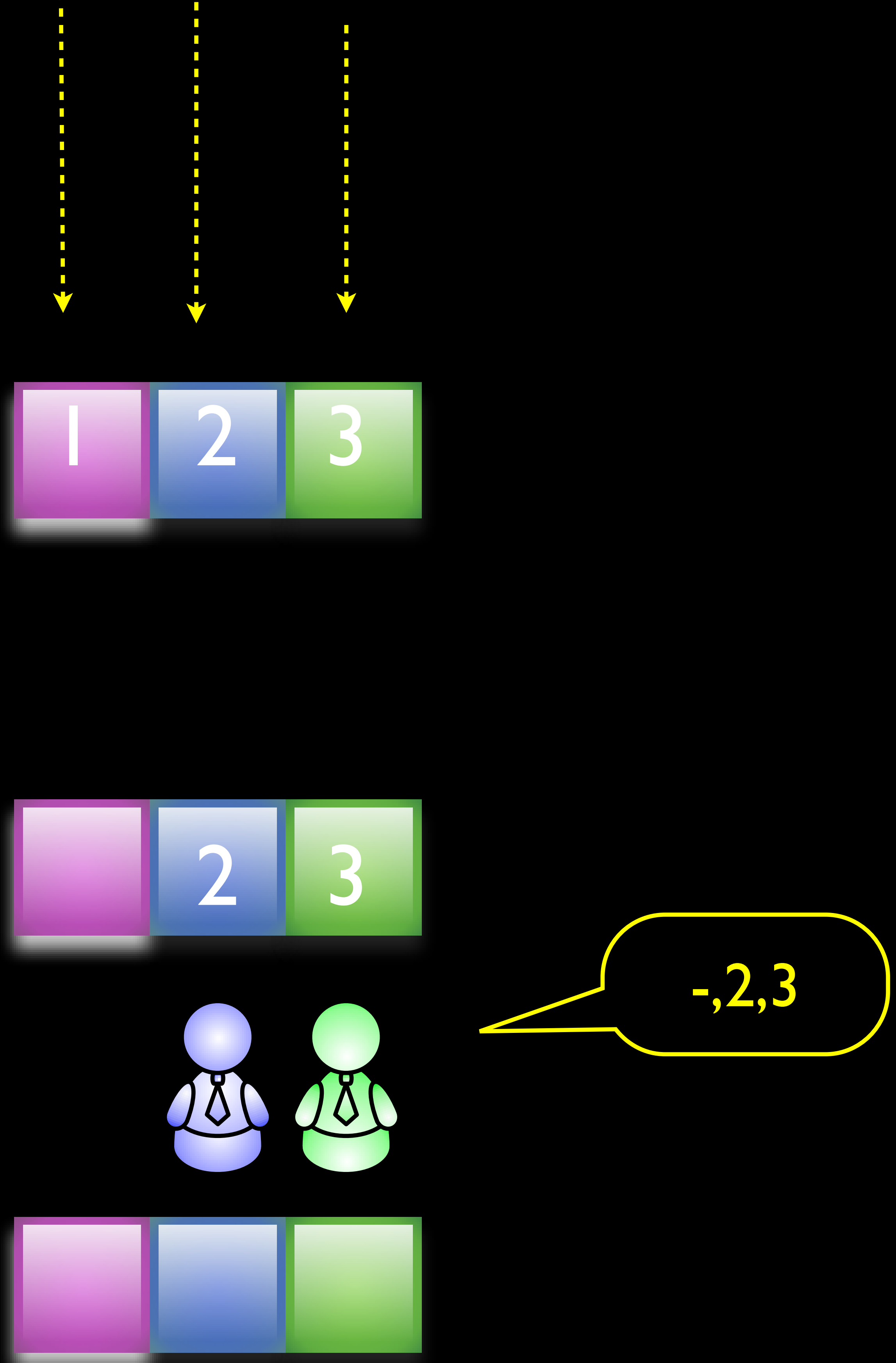
• remaining 2 go to next memory



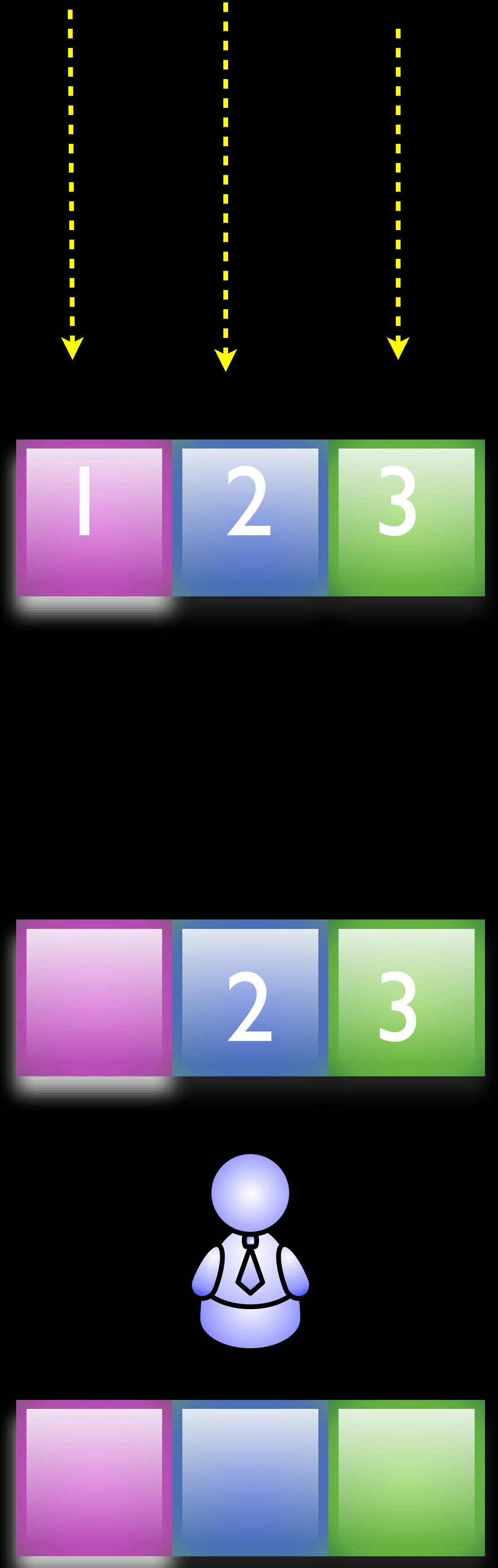
• remaining 2 go to next memory



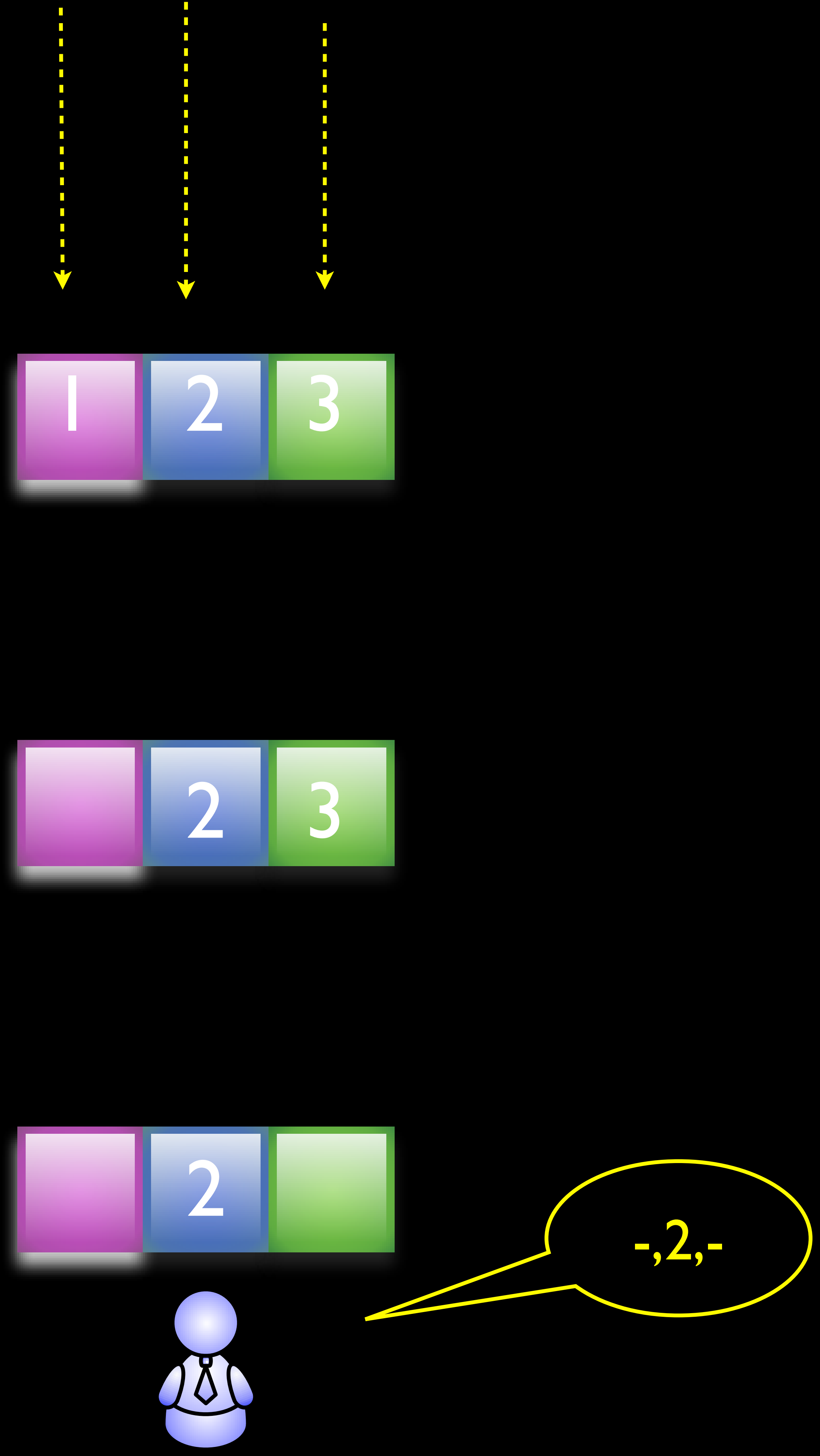
- 3rd one returns -,2,3



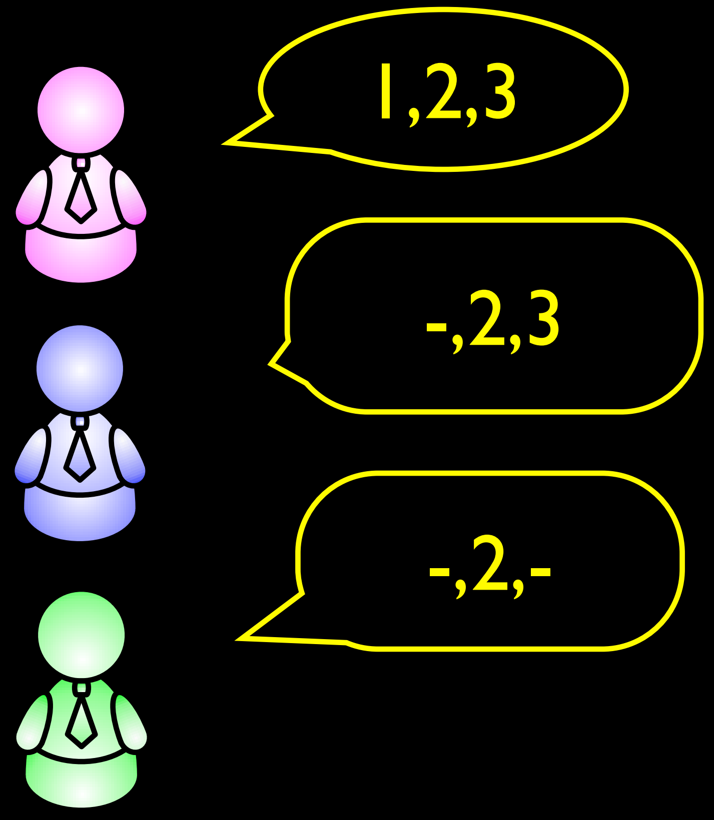
• 2nd one goes alone



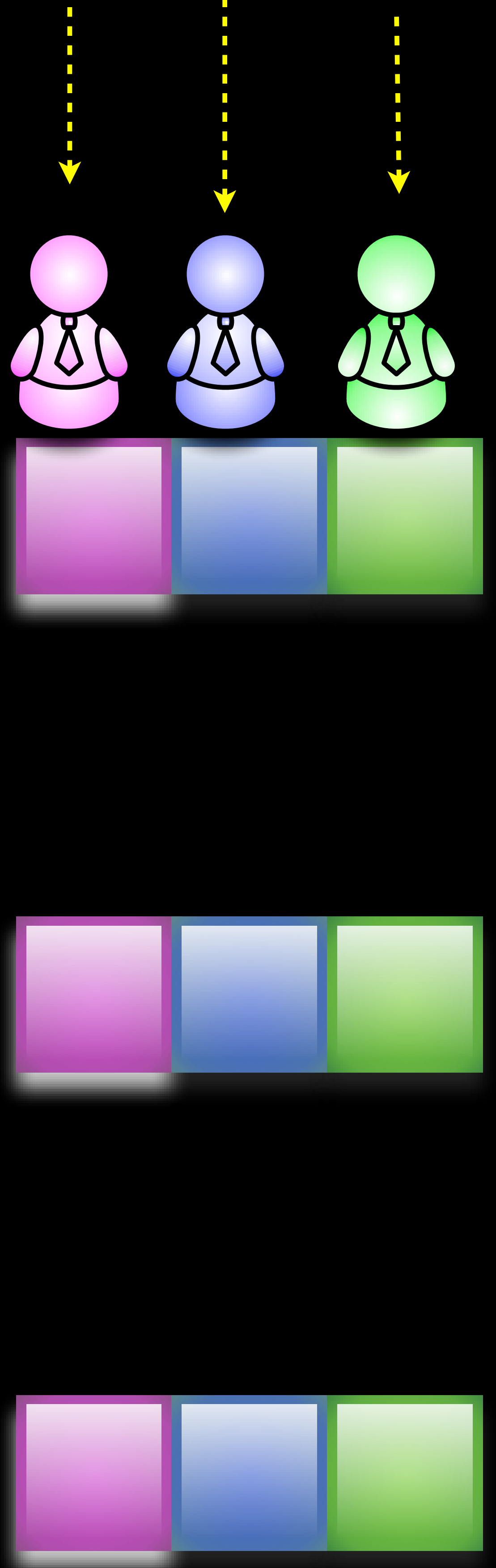
- returns -,2,-



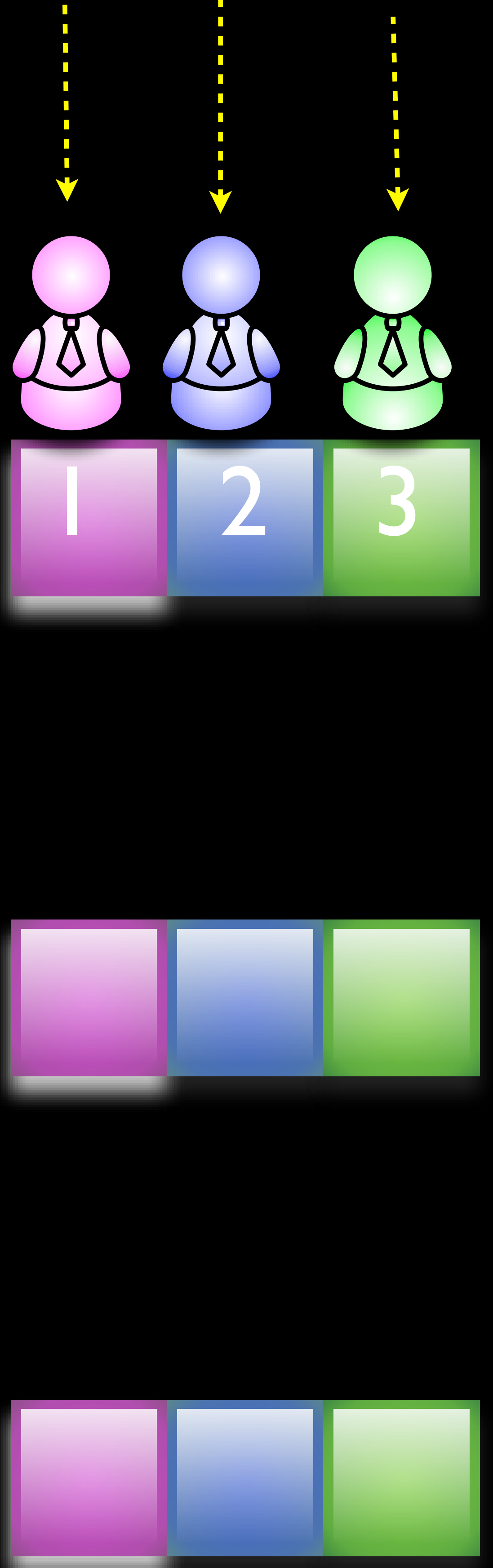
so in this run,
the views are



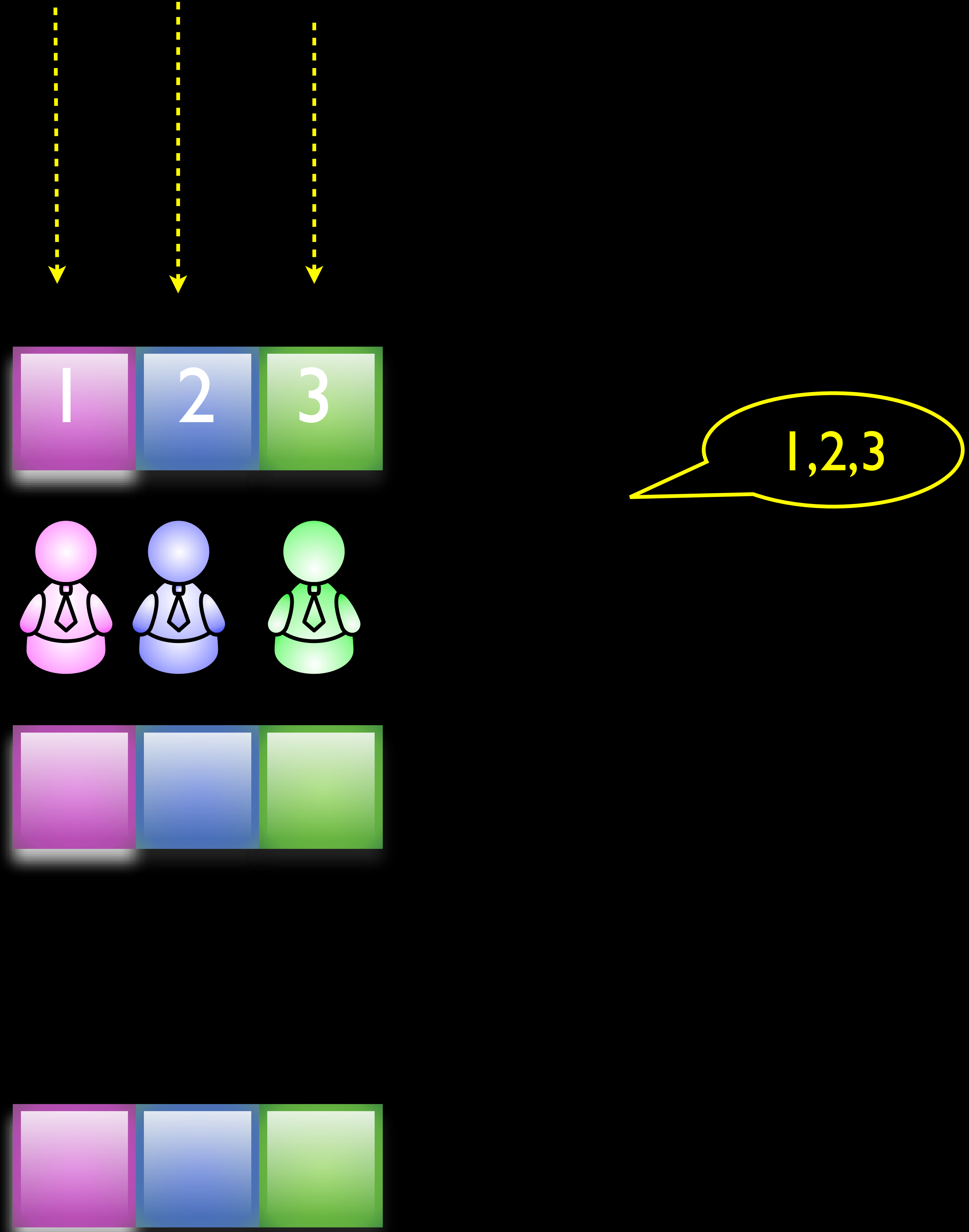
another run



•arrive in arbitrary order



- all see all

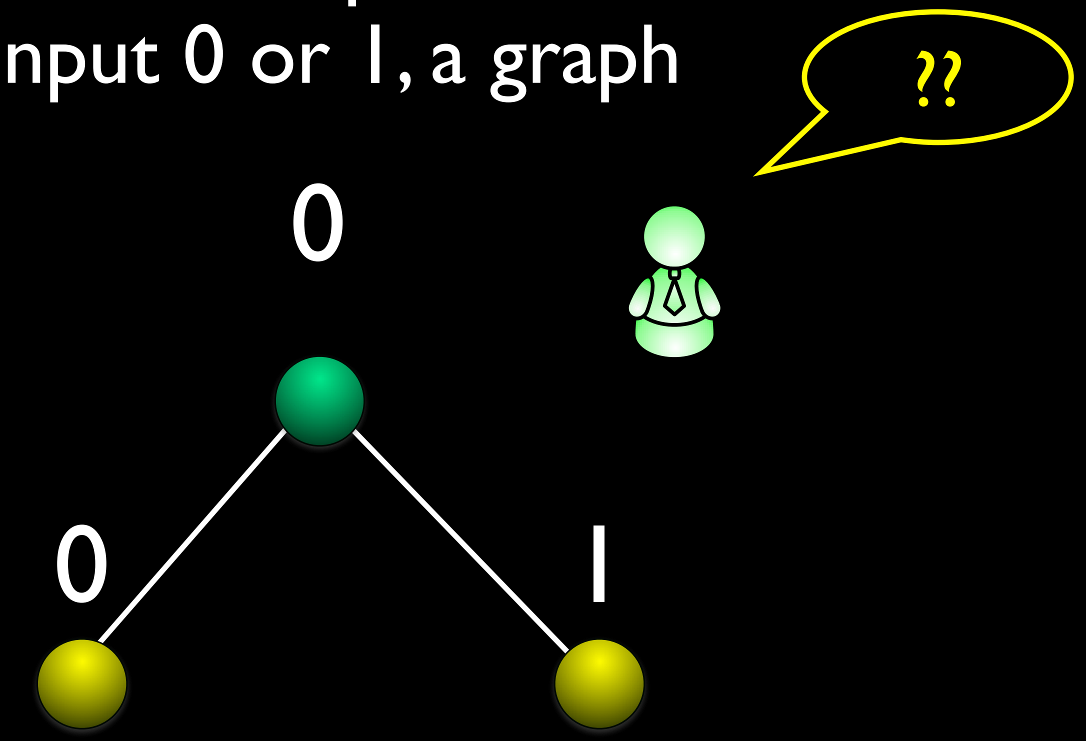


View graph

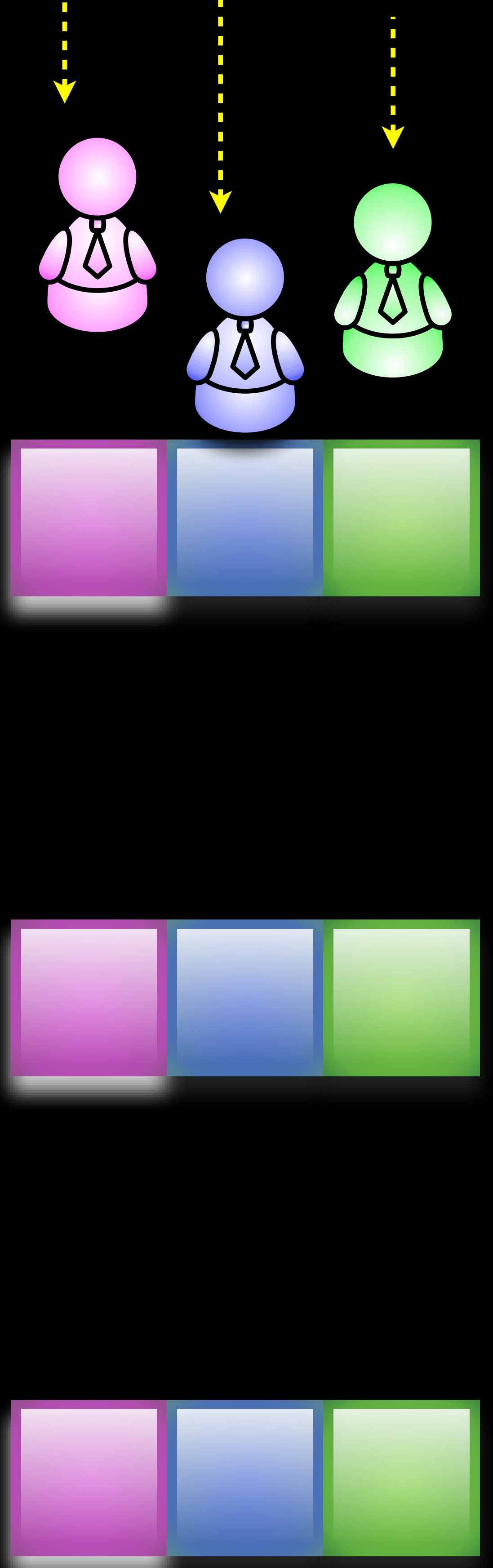
indistinguishability

- The most essential distributed computing issue is that a process has only a local perspective of the world
- Represent with a vertex labeled with id (green) and a local state this perspective
- E.g., its input is 0

- Process does not know if another process has input 0 or 1, a graph

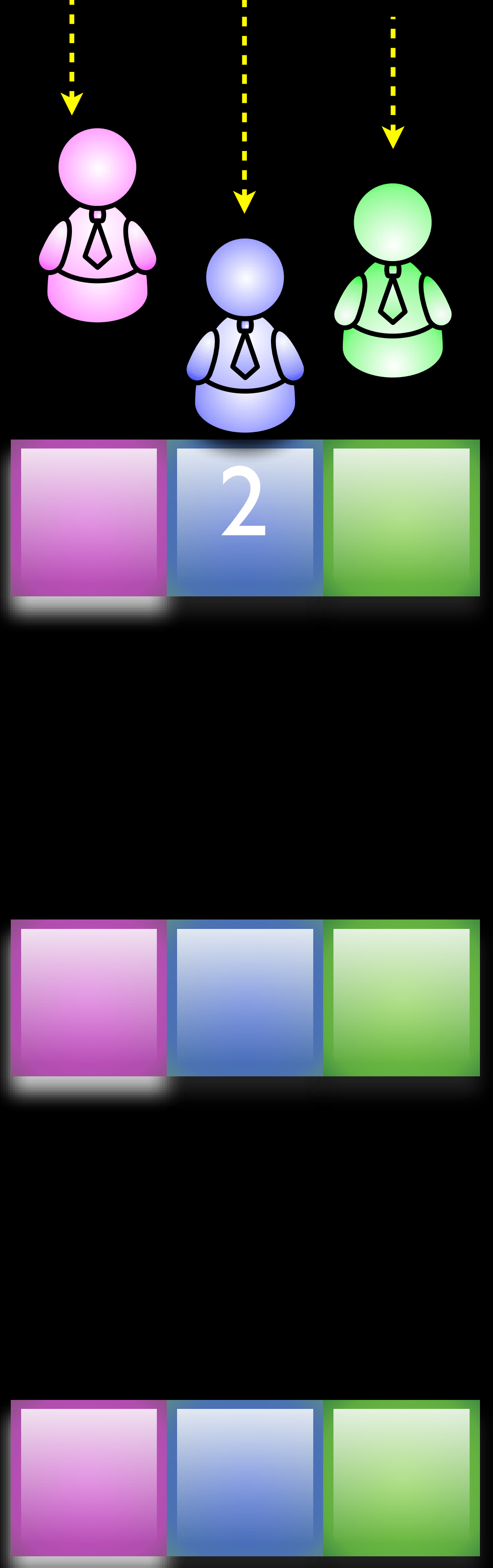


Indistinguishability graph for 2 processes

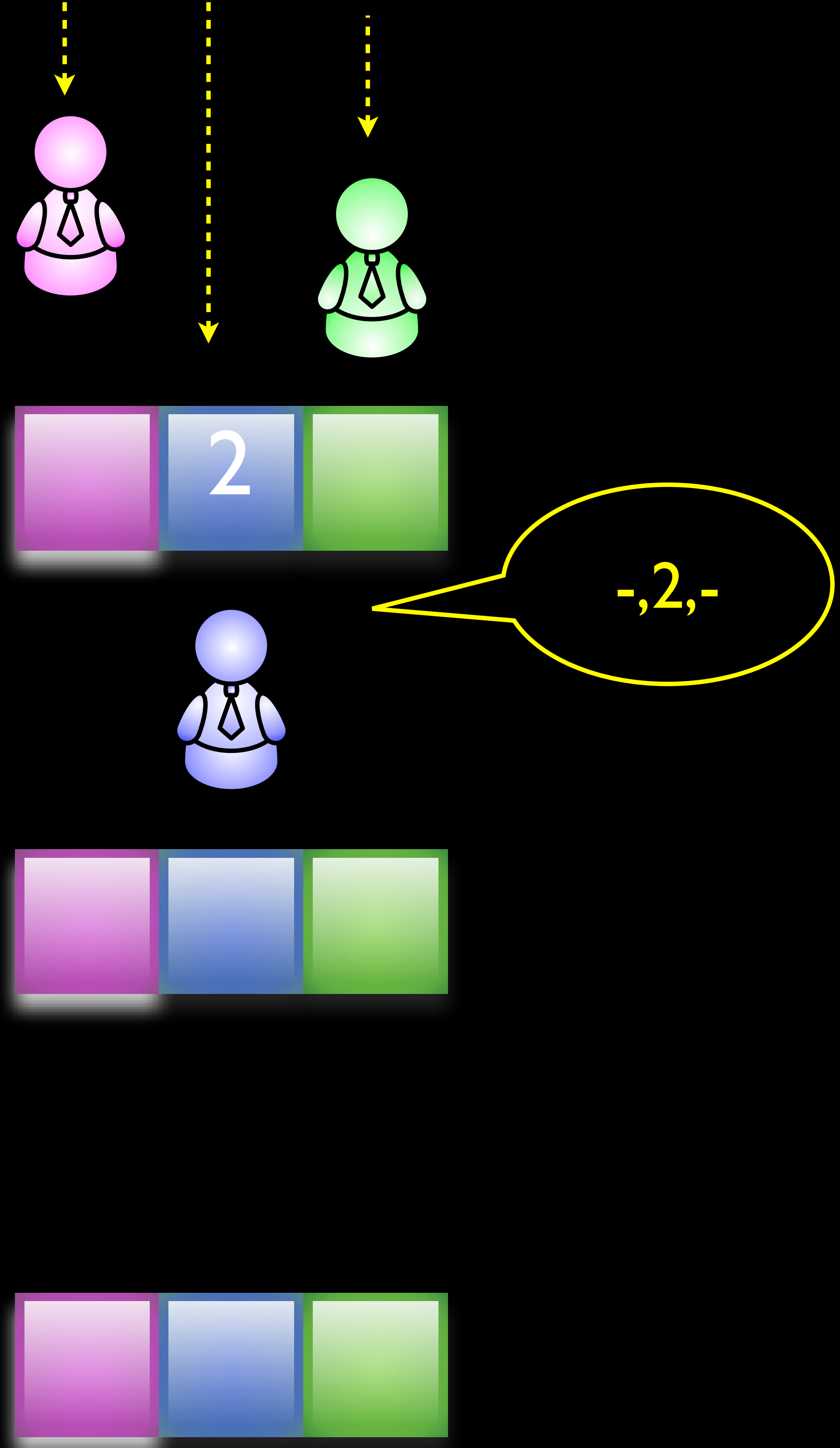


- focus on 2 processes

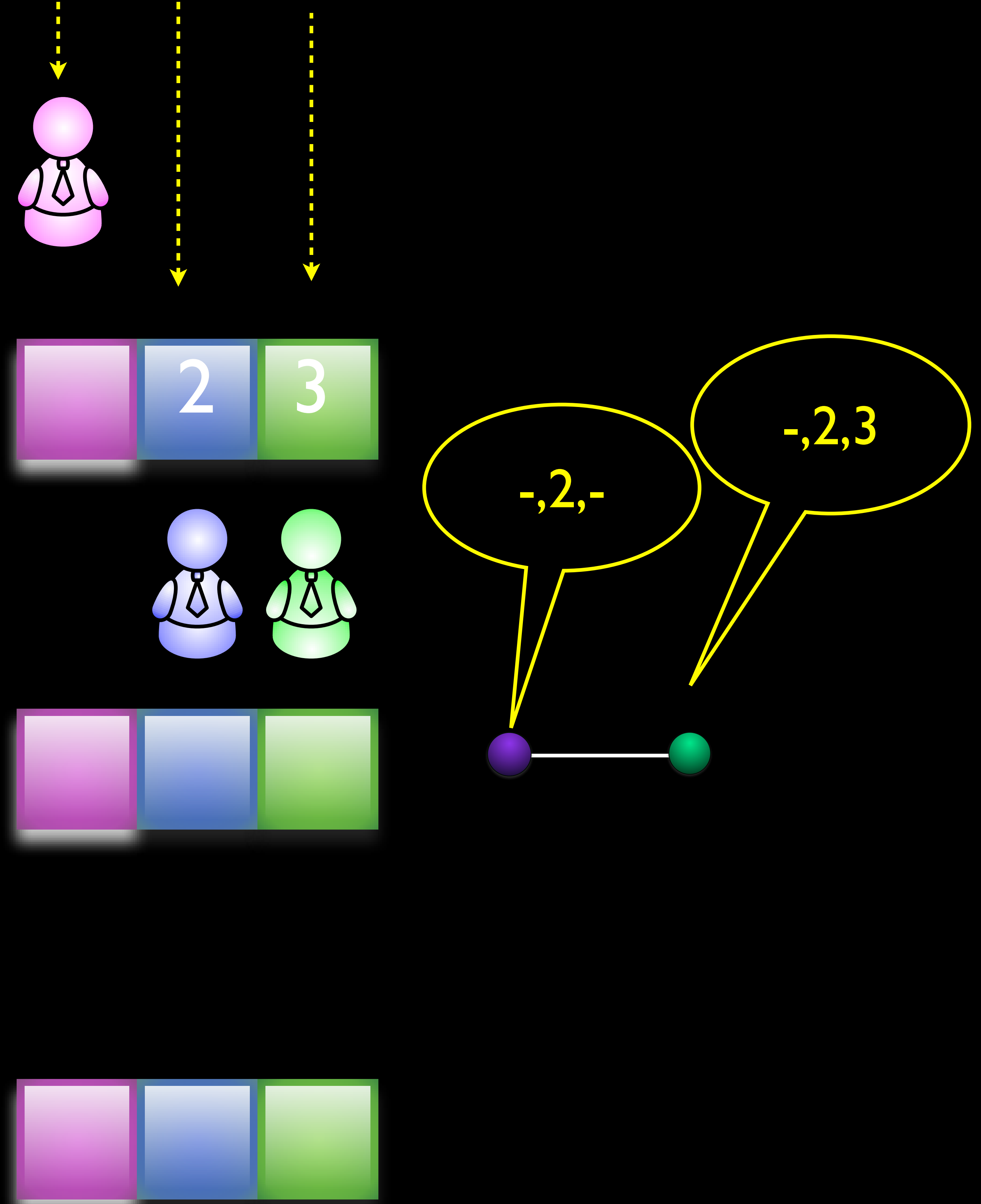
- there may be more that arrive after



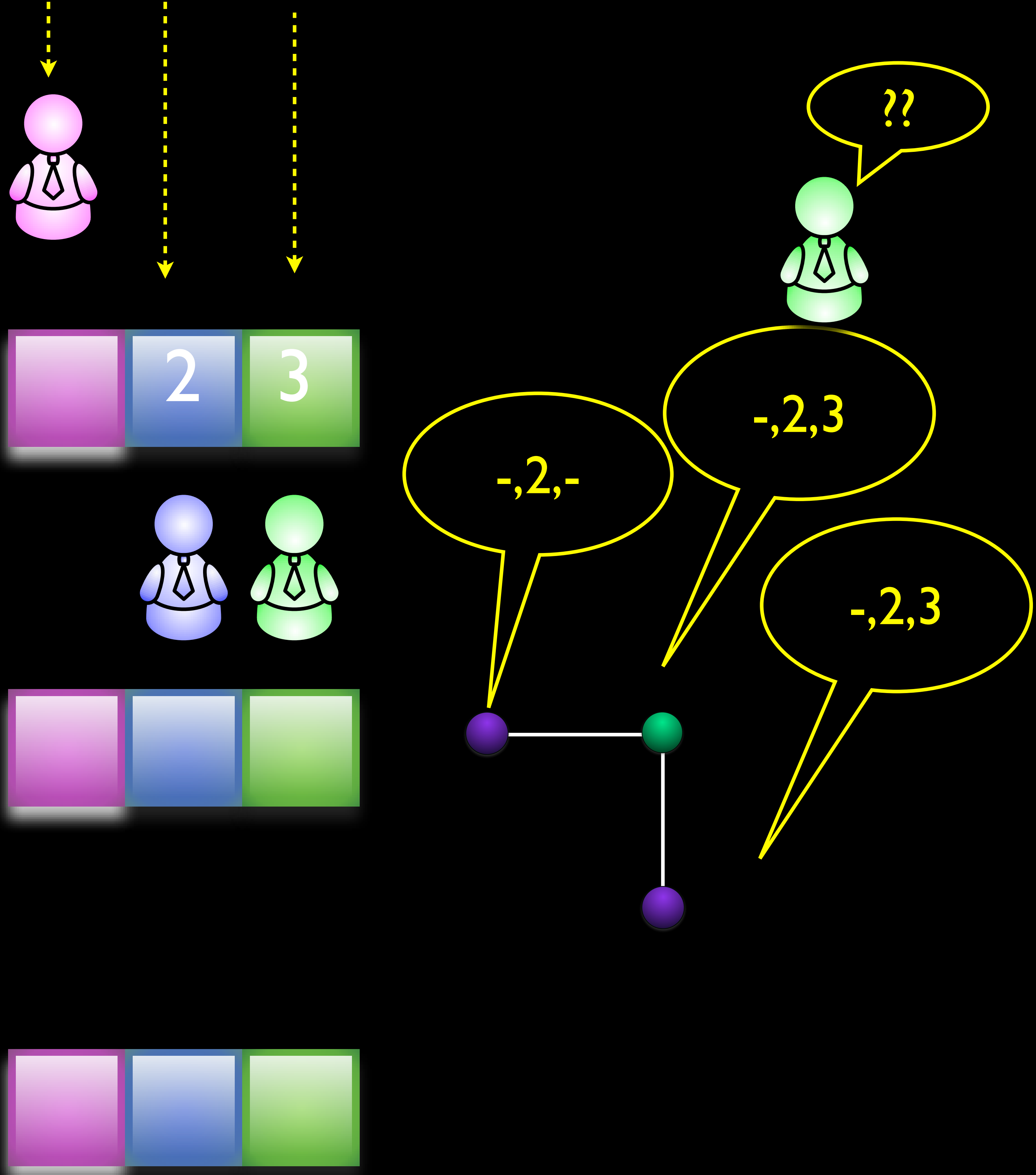
sees only itself



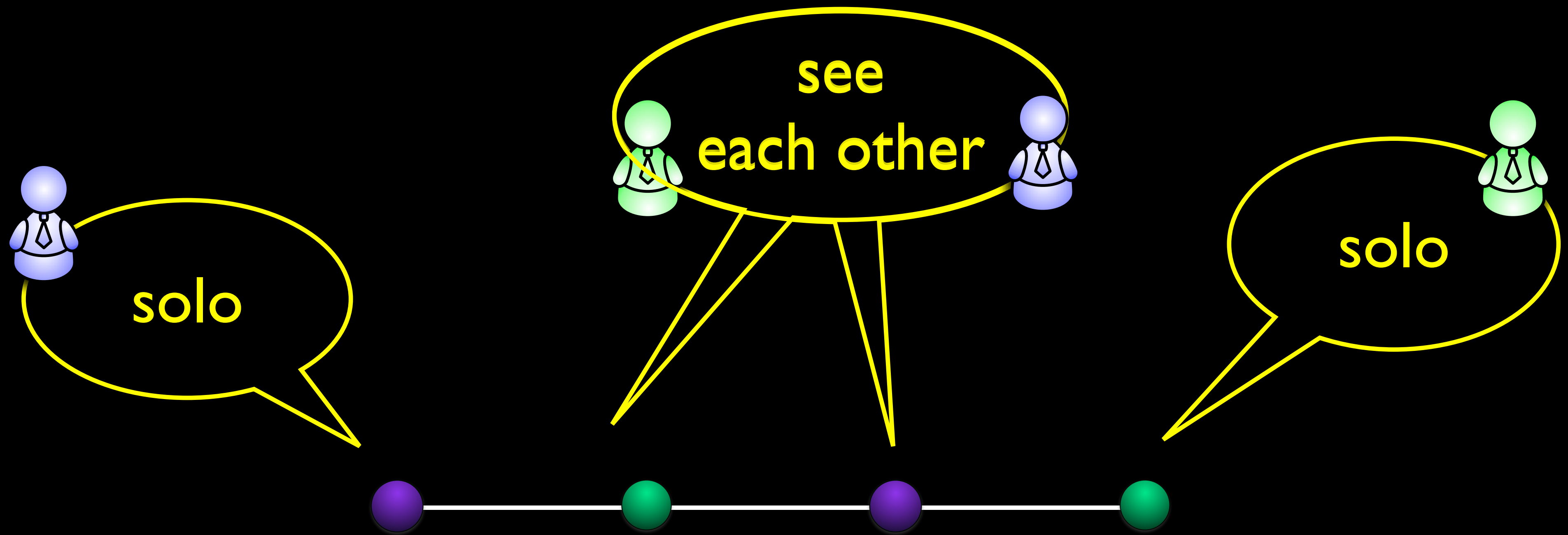
- green sees both
- but ...



- green sees both
- but, doesn't know if seen by the other

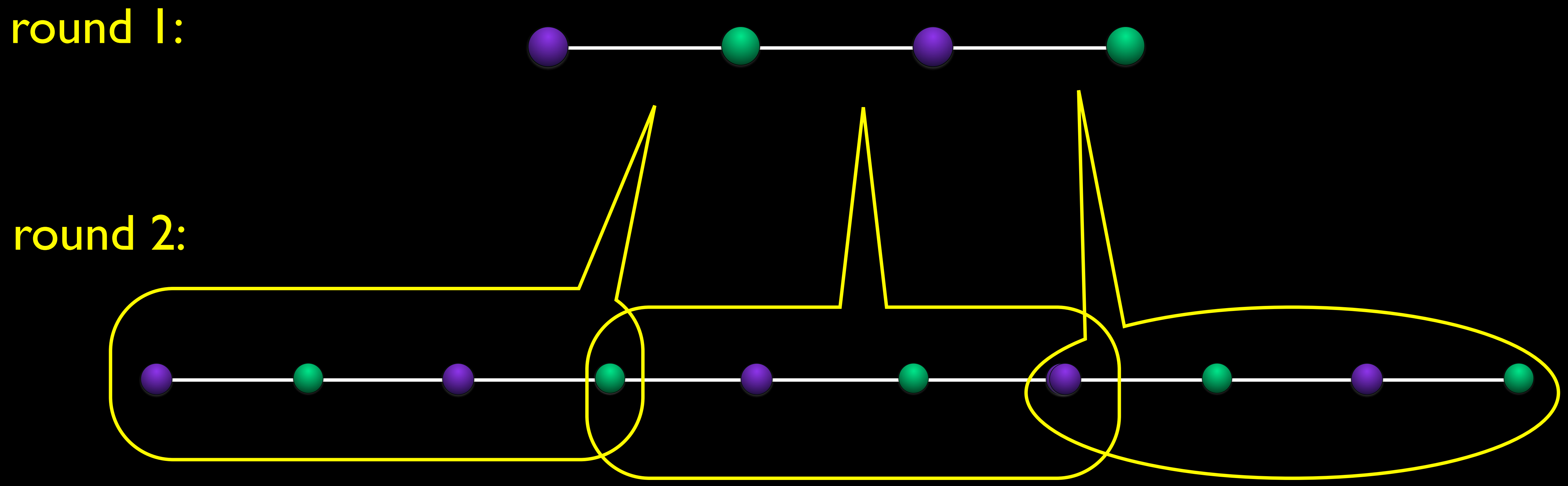


one round graph for 2 processes

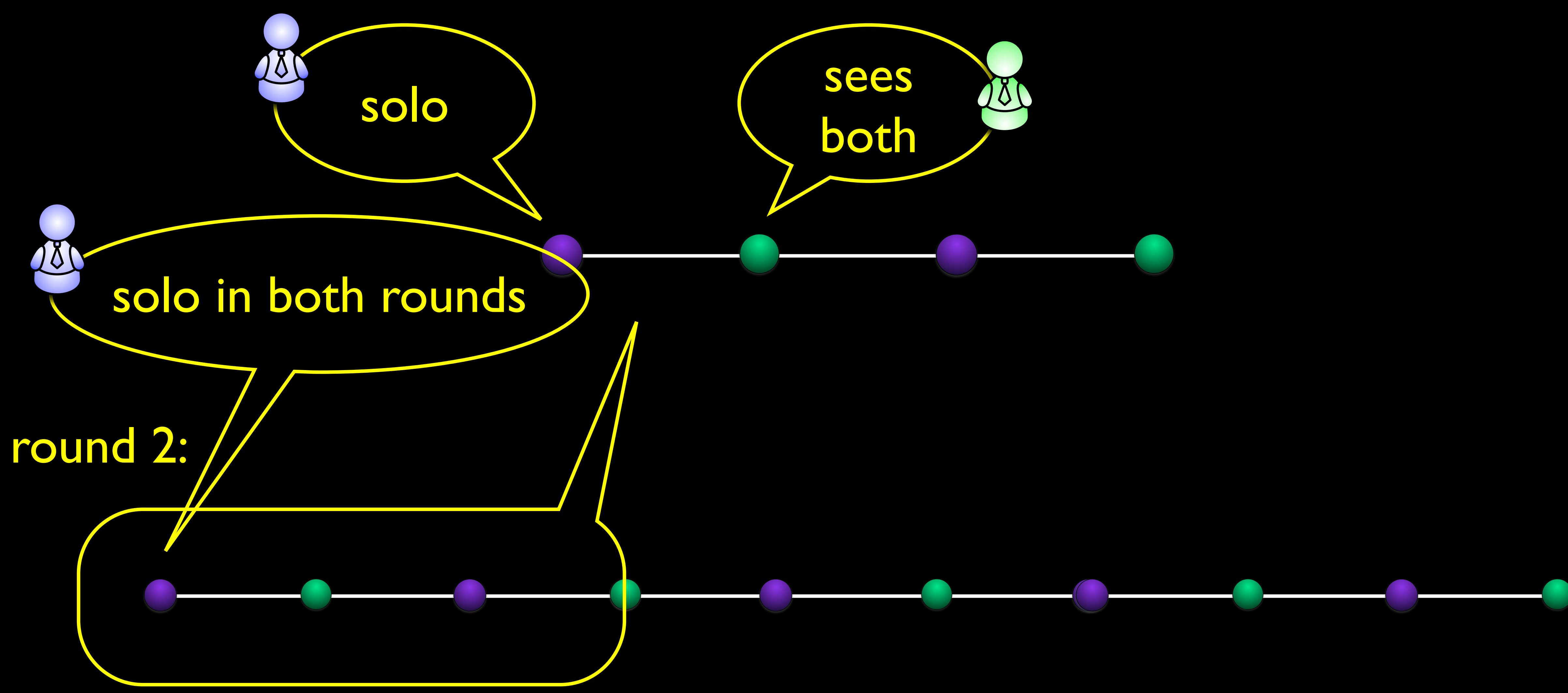


iterated runs

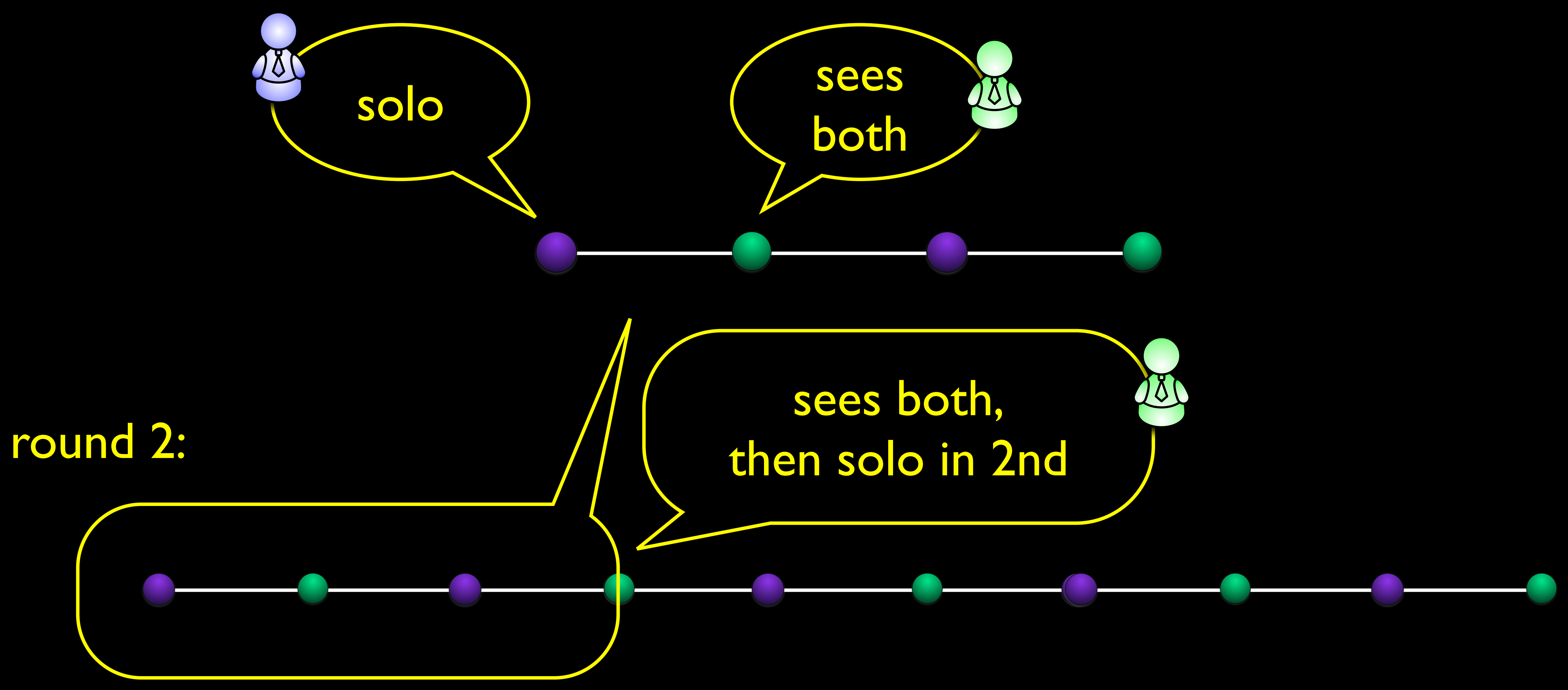
for each run in round 1 there are the same 3 runs in the next round



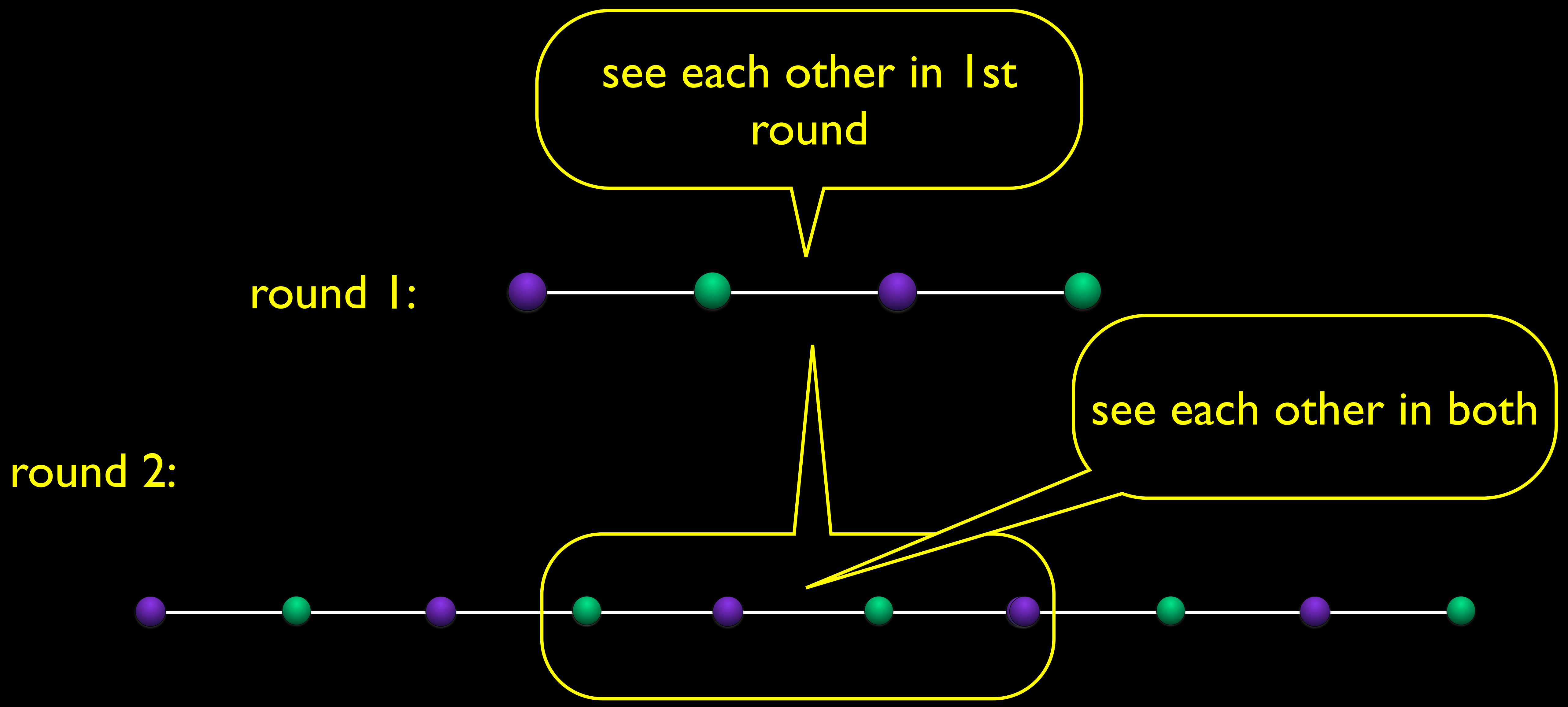
iterated runs



iterated runs



iterated runs

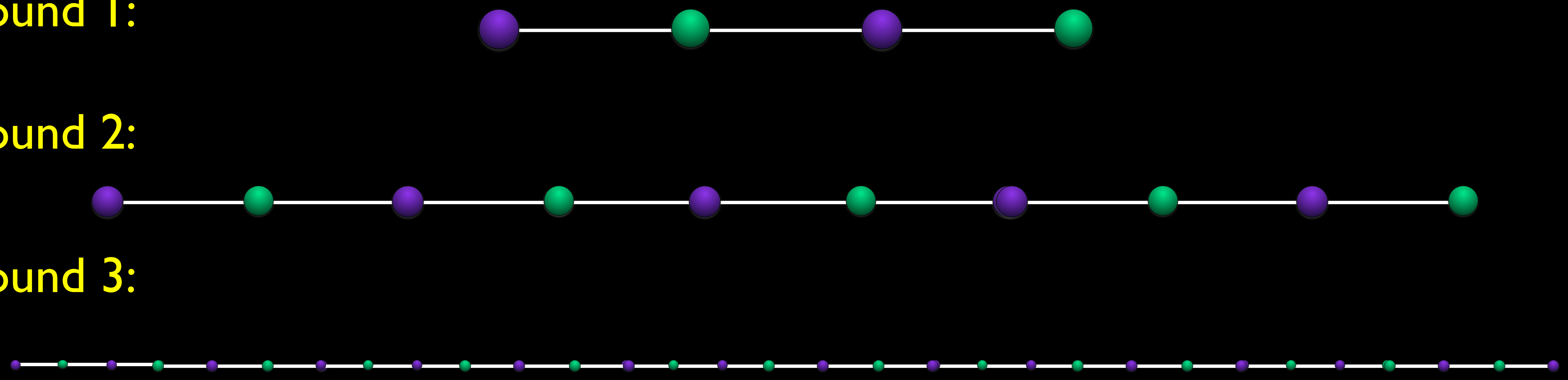


More rounds

round 1:

round 2:

round 3:



Topological invariant: protocol graph after *k* rounds

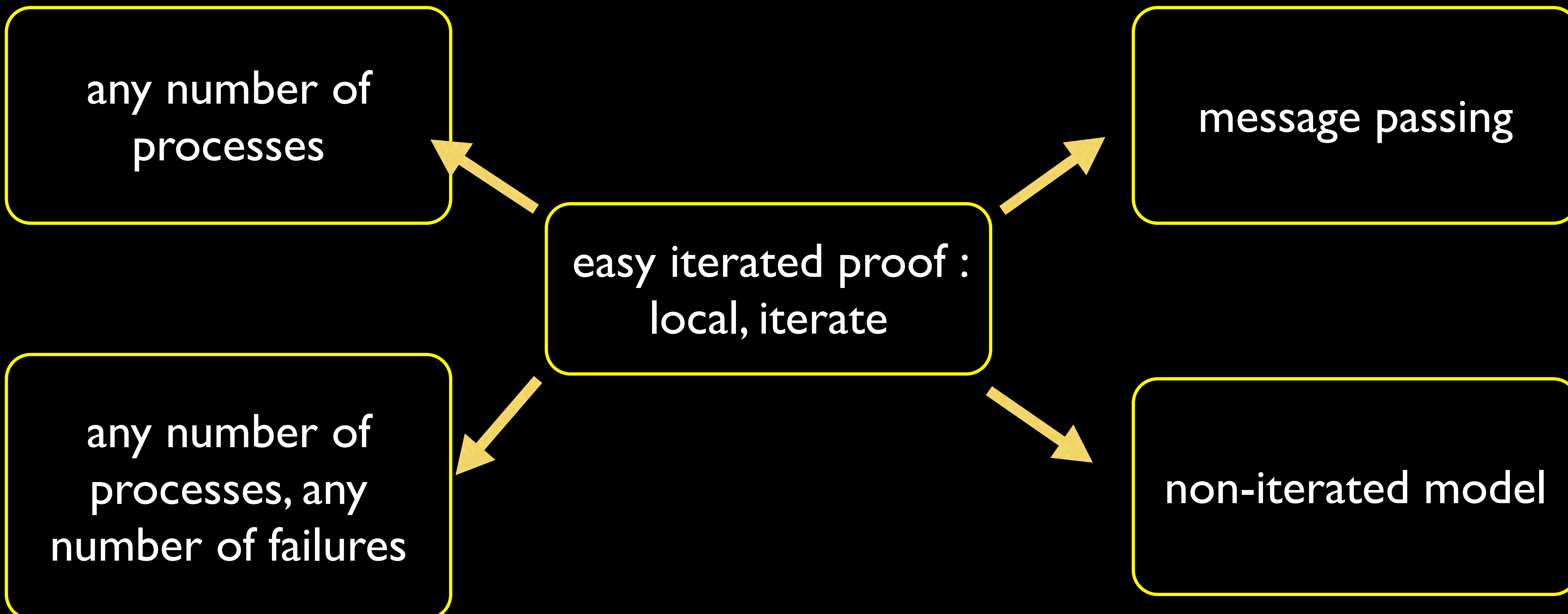
-longer
-but always connected

Wait-free theorem for 2 processes

For any protocol in the iterated model, its graph after *k* rounds is

- -longer
- -but always connected

Iterated approach: theorem holds in other models



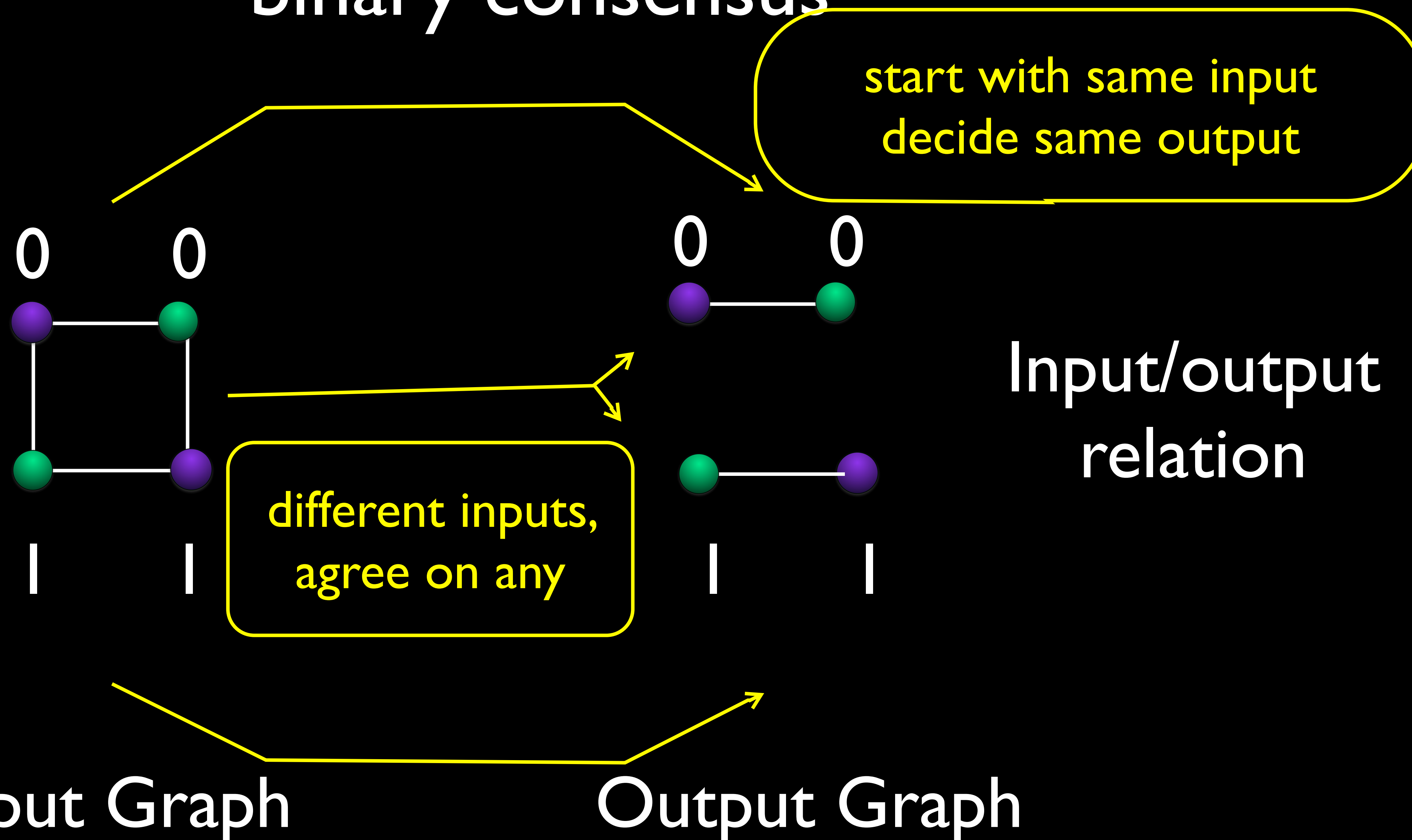
- Via known, generic simulation
- Instead of ad hoc proofs (some known) for each case

implications in terms of

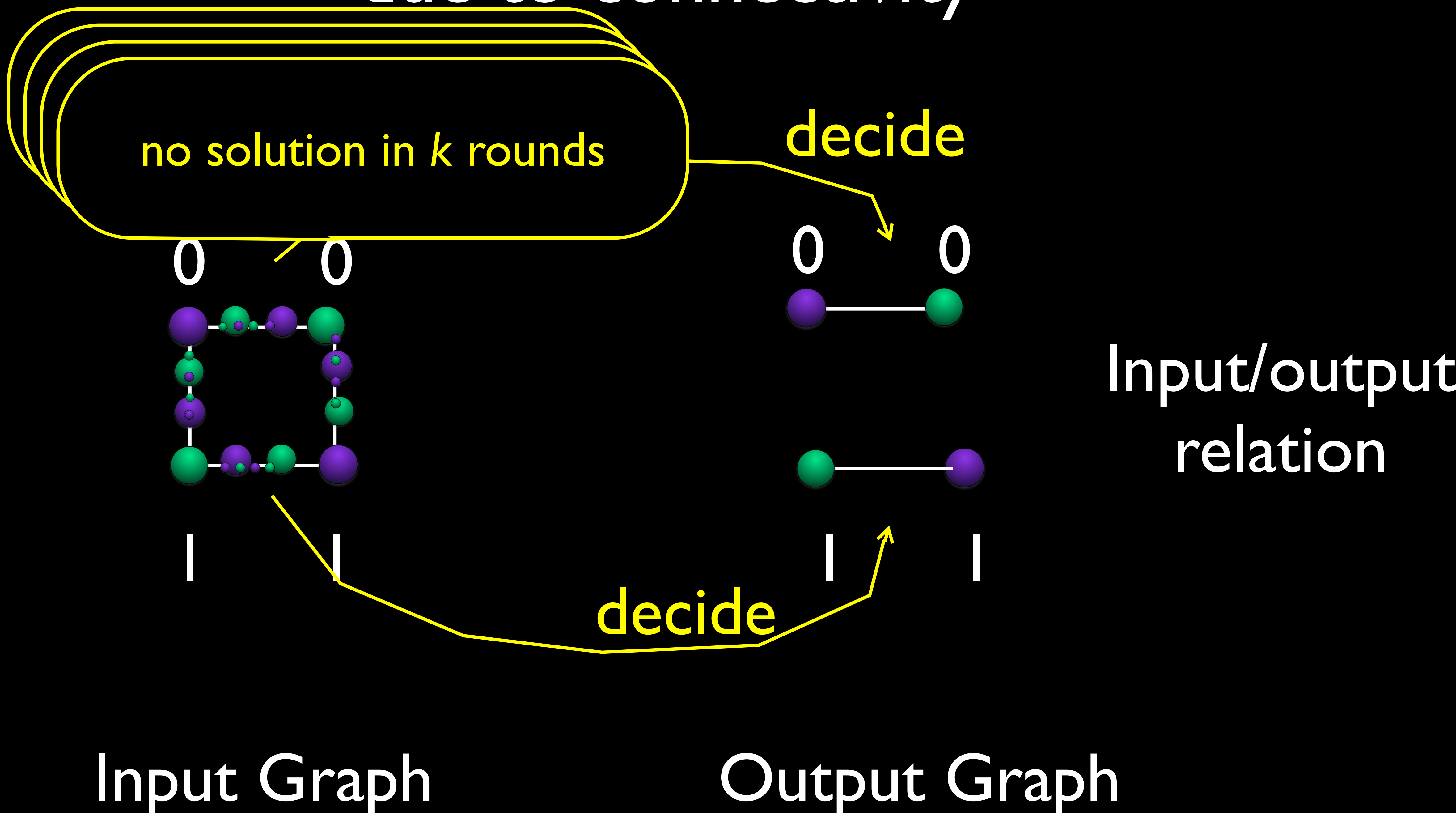
- solvability
- complexity
- computability

Distributed problems

binary consensus



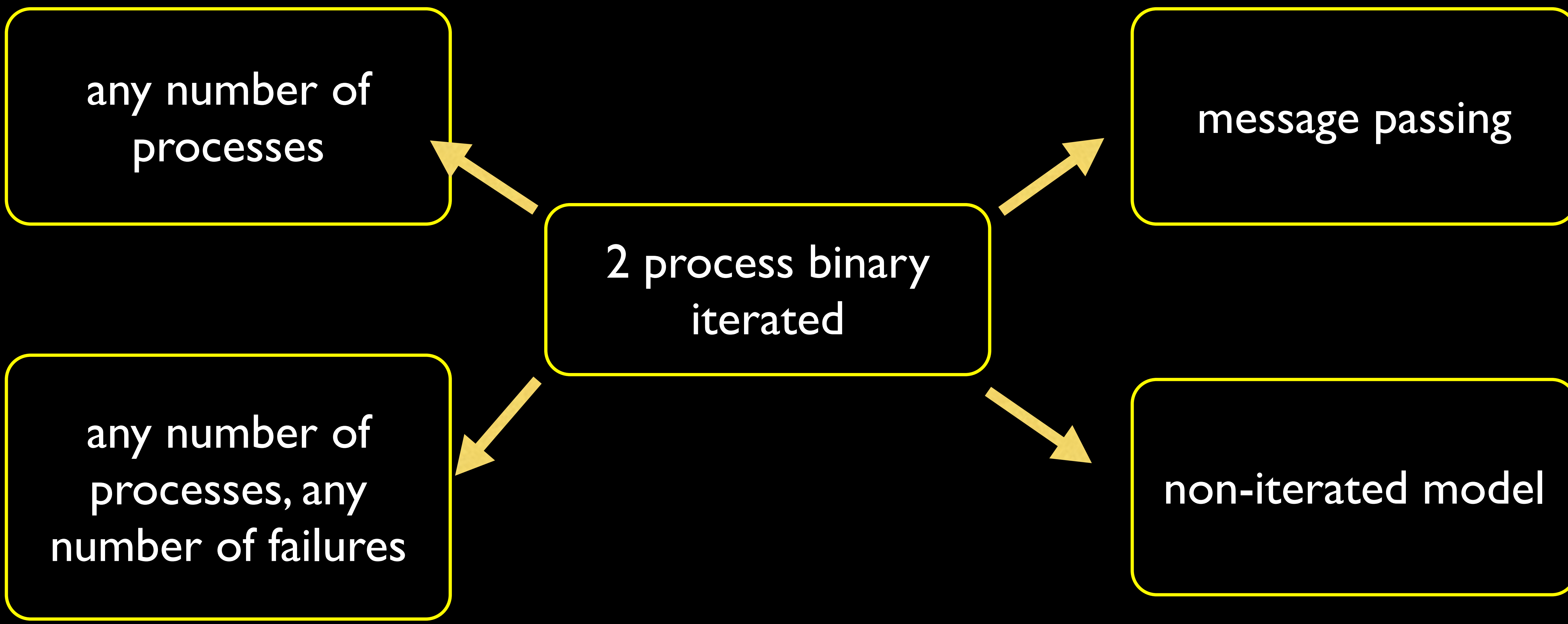
Binary consensus is not solvable due to connectivity



corollaries:

consensus impossible in the iterated model

consensus impossibility holds in other models



- Via known, generic simulation
- Instead of ad hoc proofs for each case

Decidability

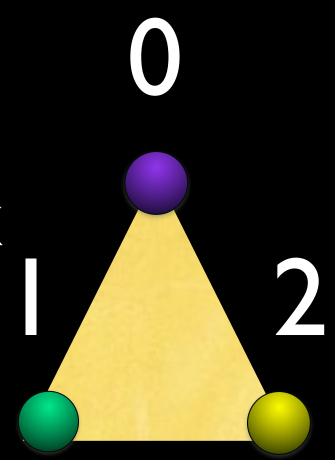
- Given a task for 2 processes, is it solvable in the iterated model?
- Yes, there is an algorithm to decide: a graph connectivity problem
- Then extend result to other models , via generic simulations, instead of ad hoc proofs

Beyond 2 processes

from *1*-dimensional graphs to *n*-dimensional complexes

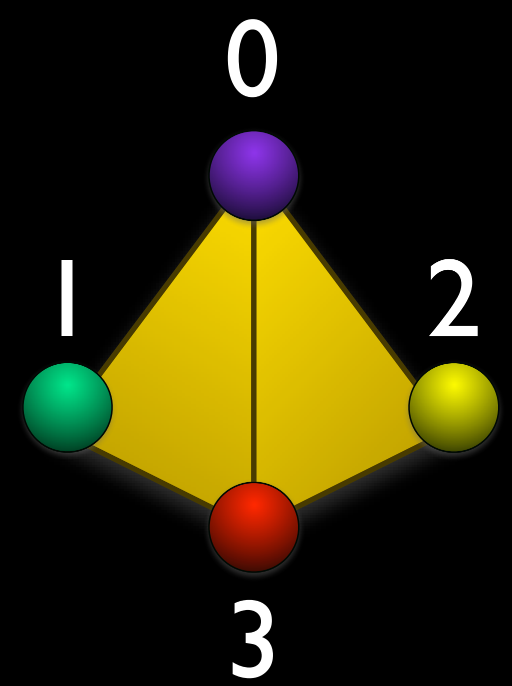
2-dim simplex

- three local states in some execution
- 2-dimensional simplex
- e.g. inputs 0,1,2



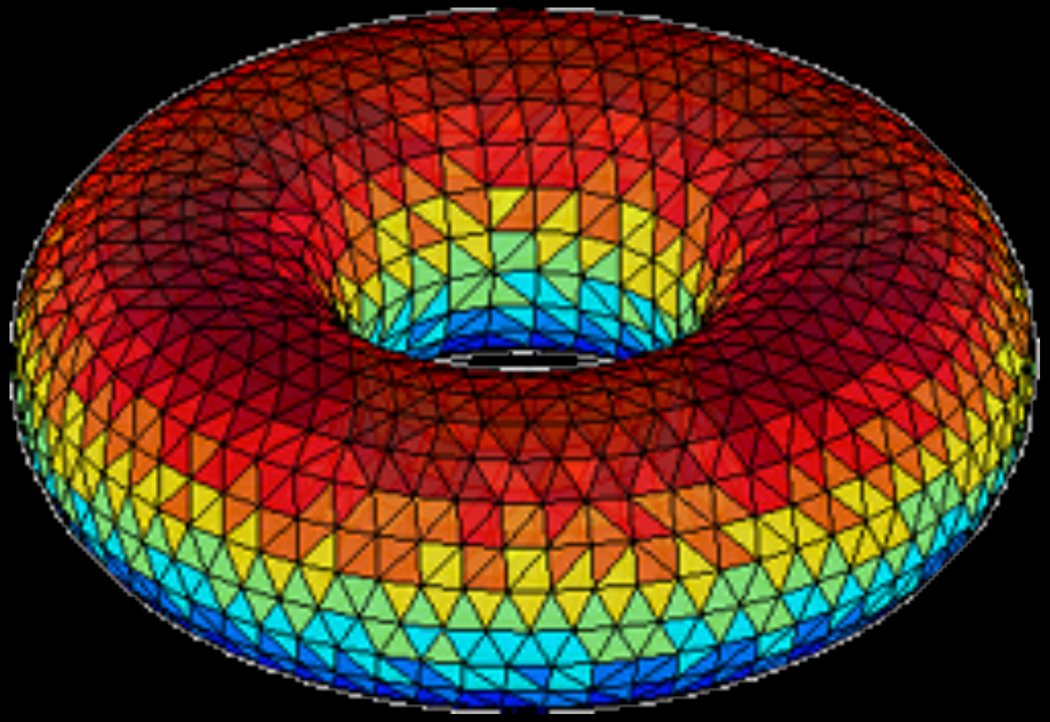
3-dim simplex

- 4 local states in some execution
- 3-dim simplex
- e.g. inputs 0,1,2,3



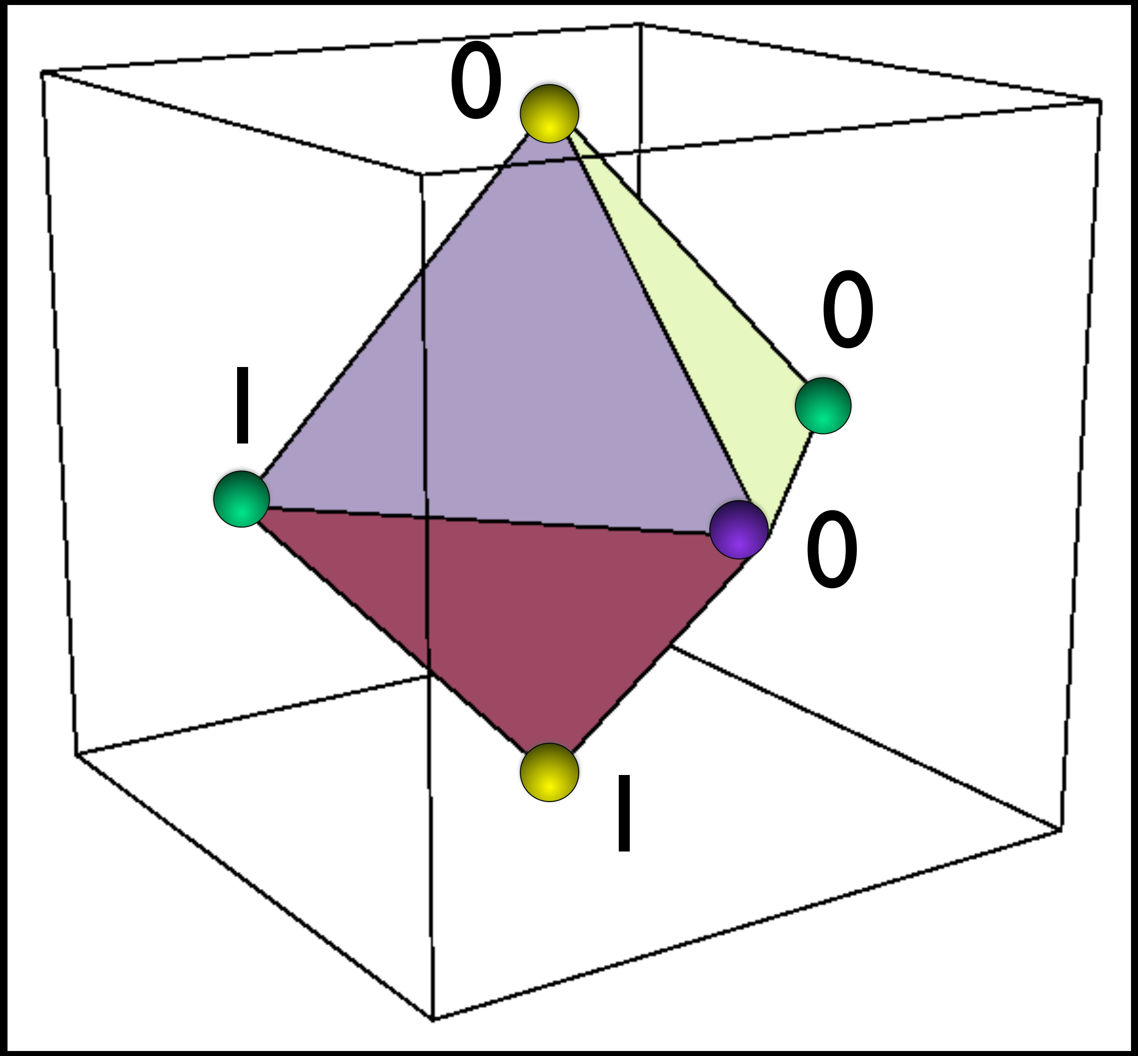
complexes

Collection of simplexes closed under containment

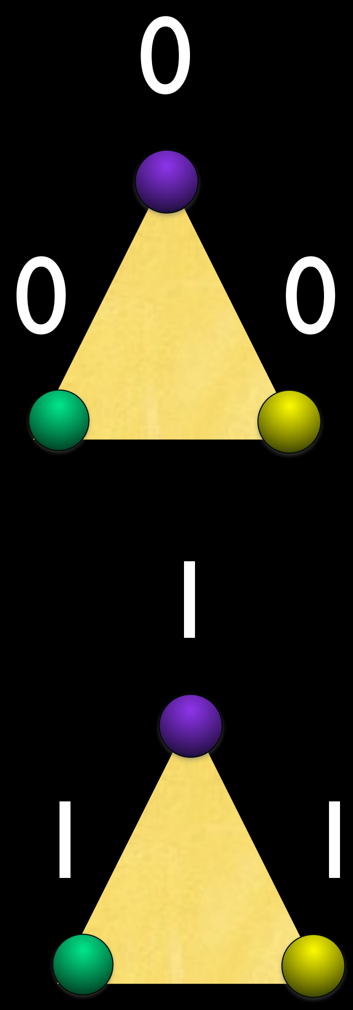


consensus task

3 processes



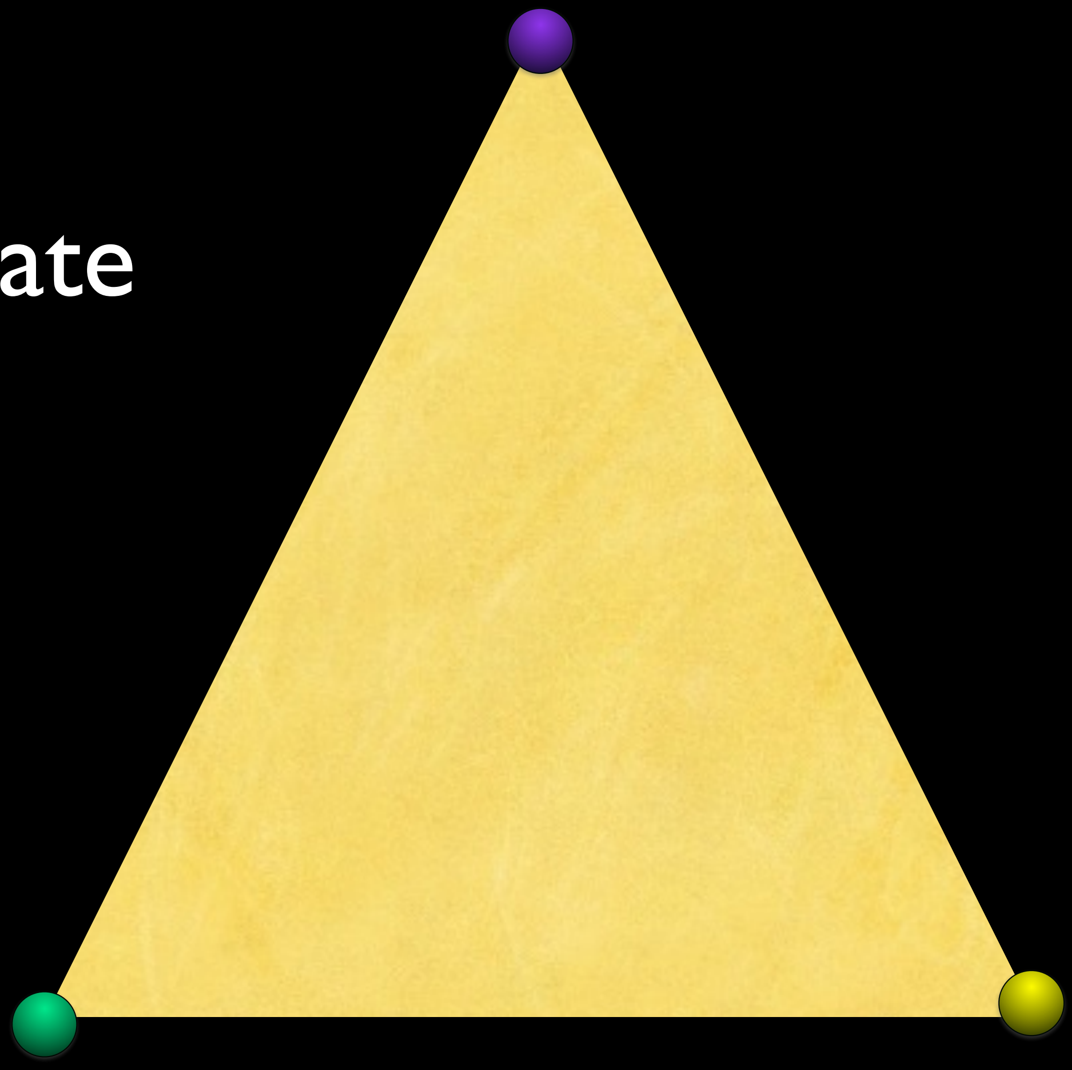
Input Complex



Output Complex

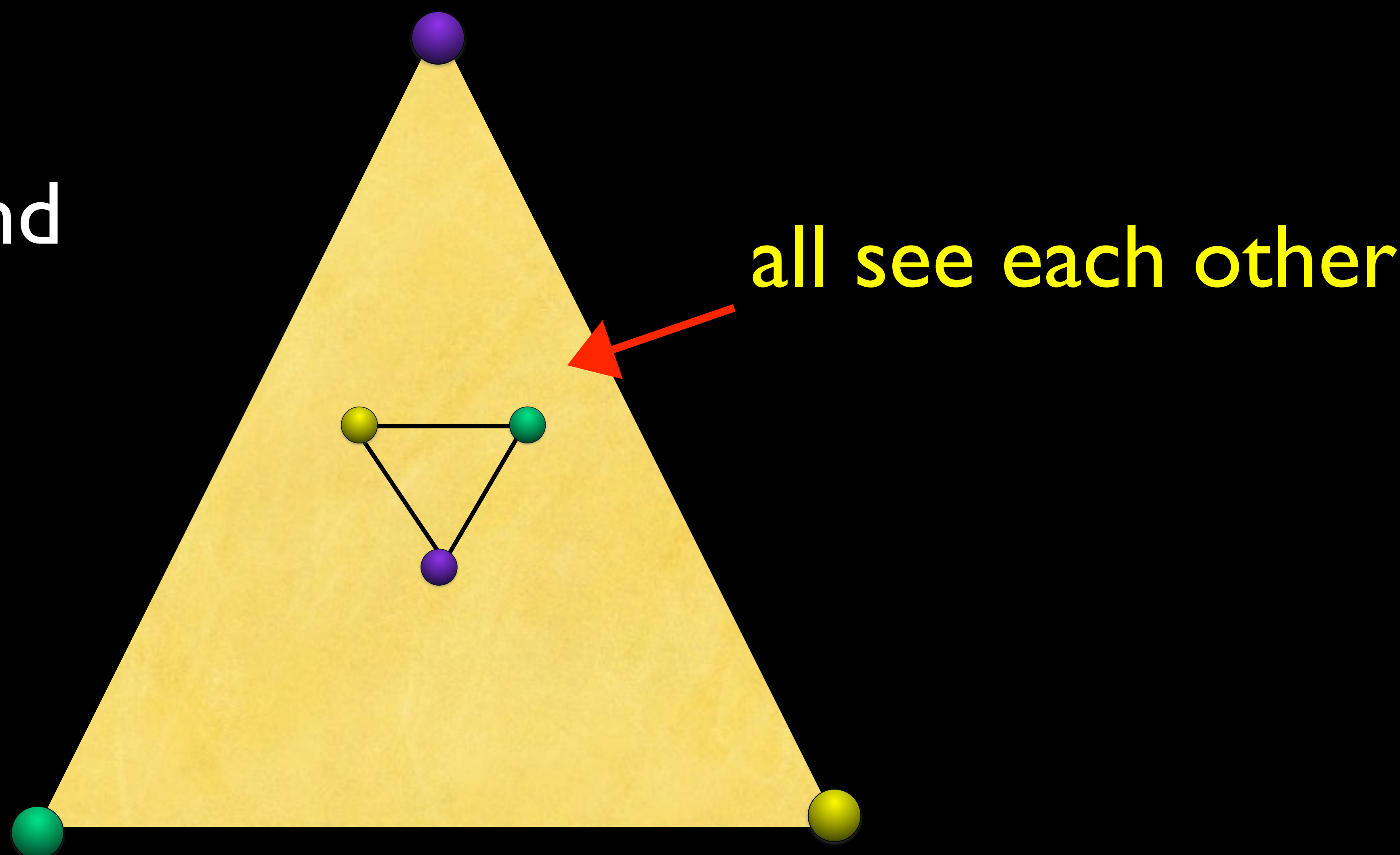
Iterated model

One initial state



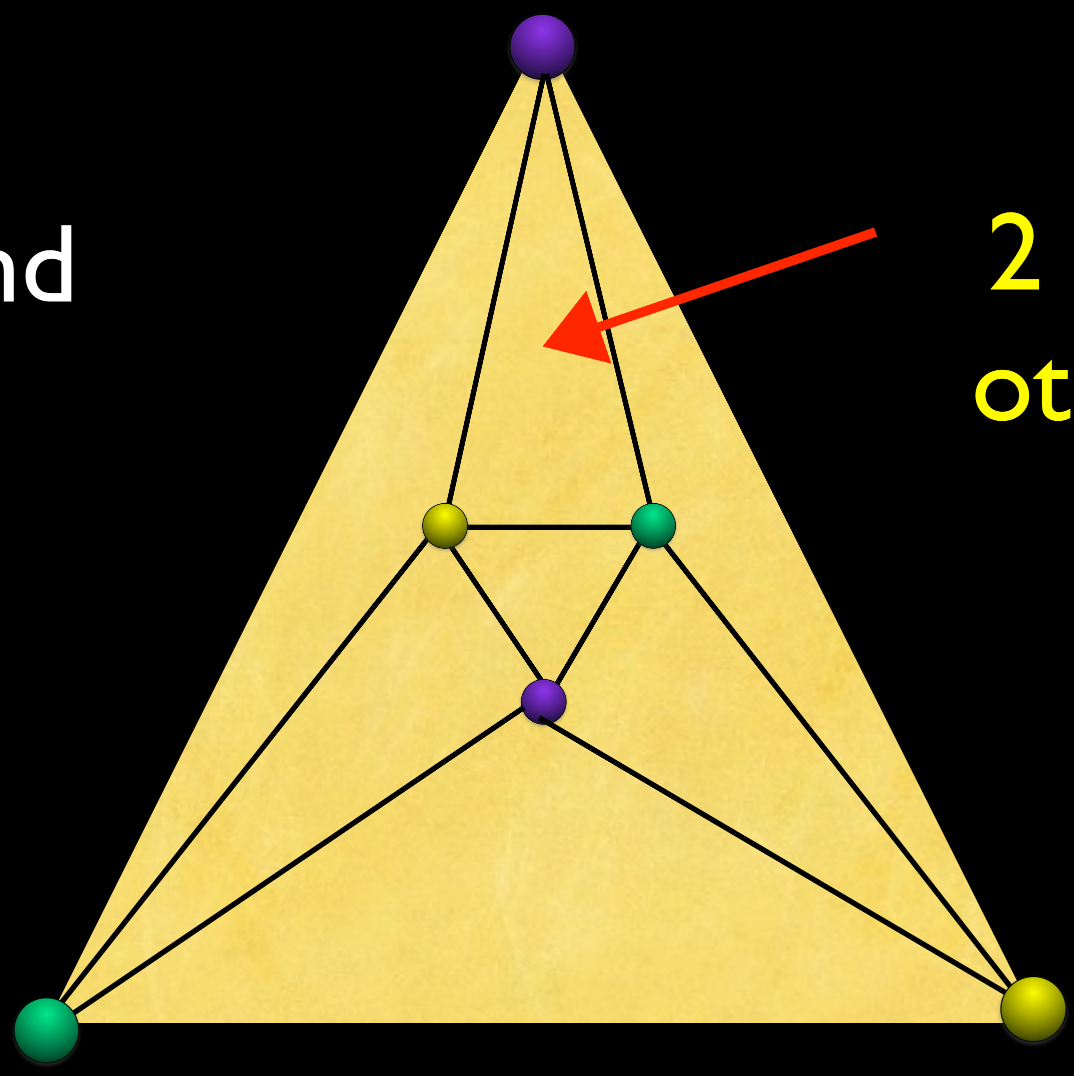
Iterated model

after 1 round



Iterated model

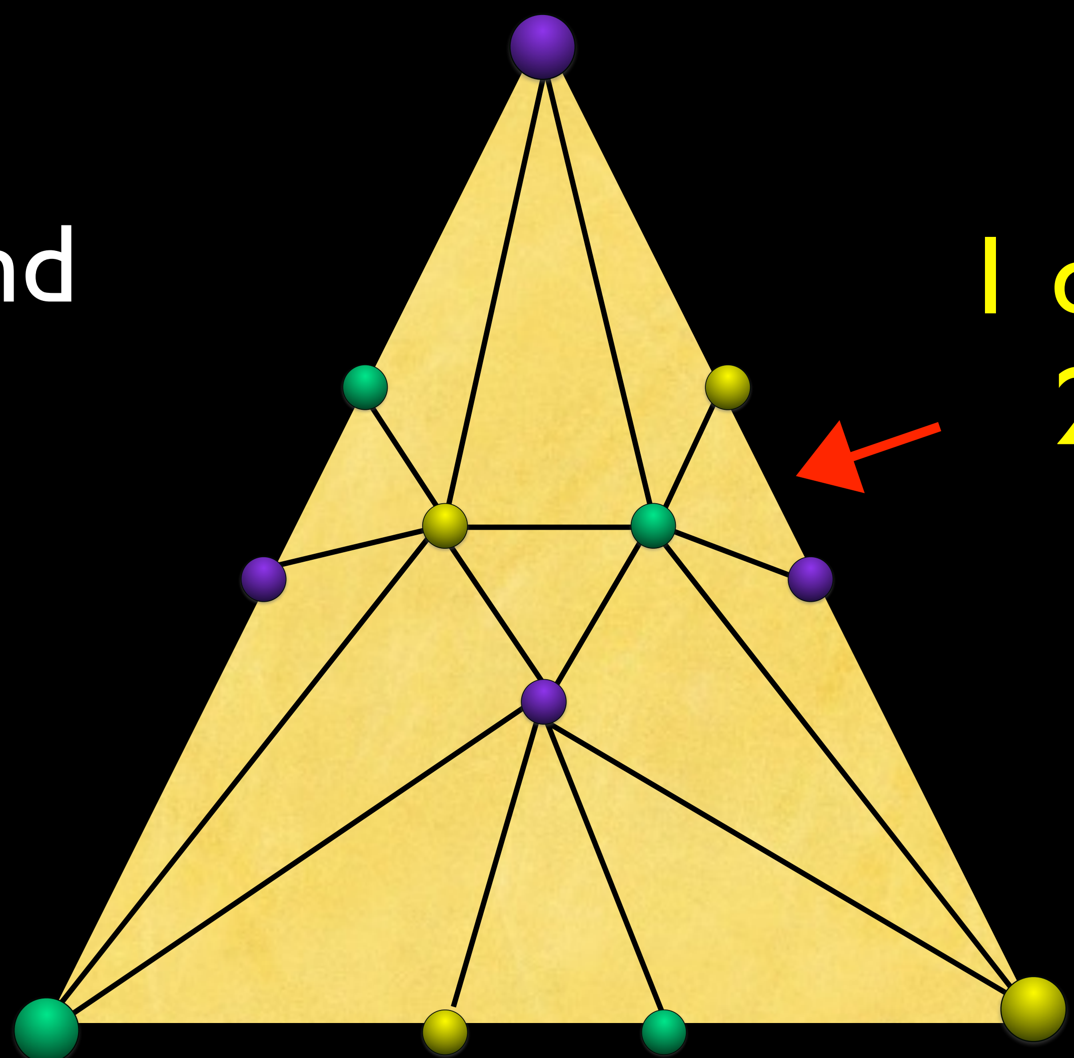
after 1 round



2 don’t know if
other saw them

Iterated model

after 1 round



1 doesn't know if
2 other saw it

Wait-free theorem for *n* processes

For any protocol in the iterated model,
its complex after *k* rounds is

- a chromatic subdivision of the input complex

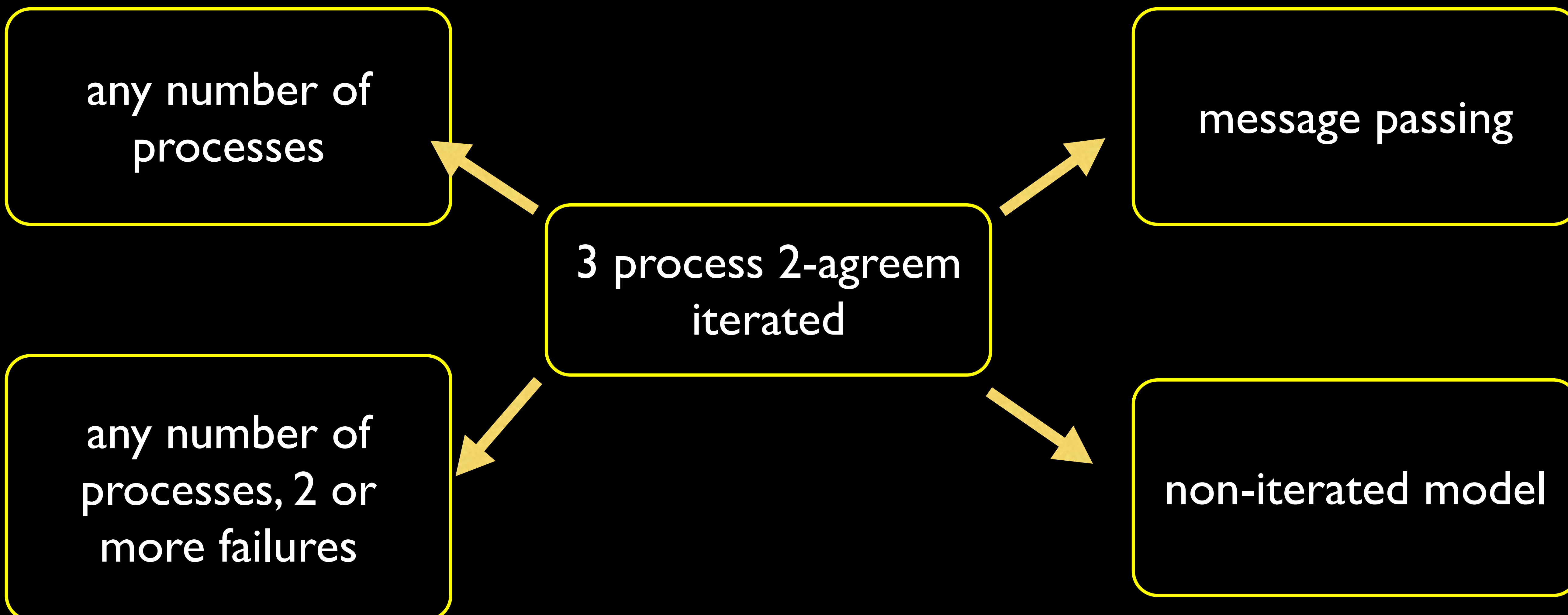
General wait-free iterated solvability theorem

A task is solvable if and only if the input complex can be chromatically subdivided and mapped into the output complex continuously respecting colors and the task specification

Decidability

- Given a task for 3 processes, is it solvable in the iterated model?
- No! there are tasks that are solvable if and only if a loop is contractible in a 2-dimensional complex
- Then extend result to other models, via generic simulations, instead of ad hoc proofs

Extension to other models

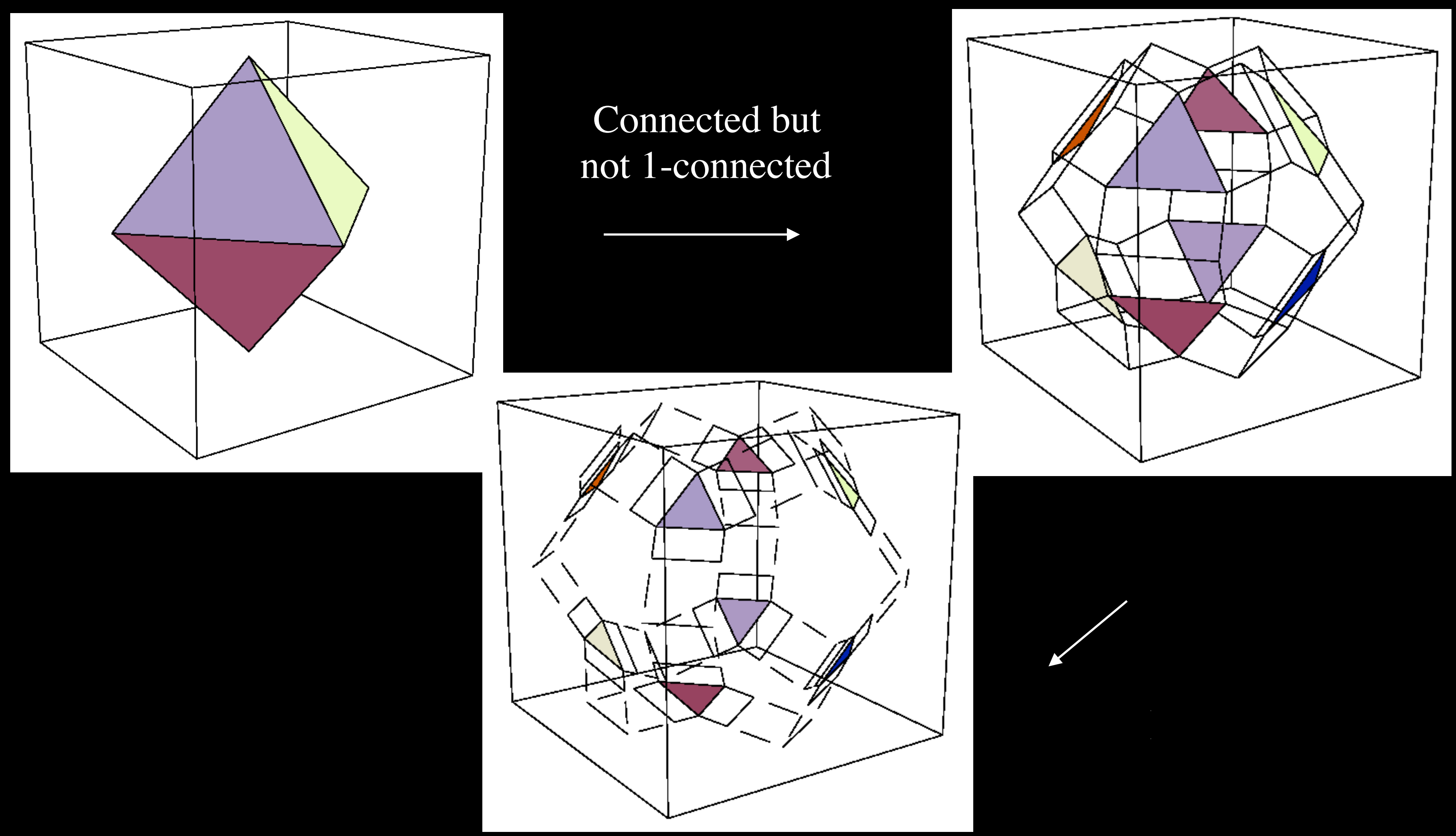


- Via known, generic simulation
- Instead of ad hoc proofs for each case

Conclusions

- In distributed computing there are too many different issues of interest, no single model can capture them all

Synchronous protocol complex evolution



Conclusions

- But the iterated model (with extensions not discussed here) captures essential distributed computing aspects
- and topology is the essential feature for computability and complexity results

END