Computability in distributed computing

Sergio Rajsbaum
Instituto de Matemáticas
UNAM

From the book coauthored with Maurice Herlihy and Dmitry Kozlov to be published by Elsevier

Computability in distributed computing

an introduction

Sergio Rajsbaum
Instituto de Matemáticas
UNAM

From the book coauthored with Maurice Herlihy and Dmitry Kozlov to be published by Elsevier

Distributed computability has a topological nature

- Discovered in 1993: Herlihy, Shavit, Borowski, Gafni, Saks, Zaharoughlu
- Further developed by Attiya, Castaneda, Kouznetsov, Raynal, Travers, Corentin, etc.
- Work from semantics community Eric Goubault, M. Raussen, and others

Two stories about love

Two stories about love

Using topology

The stories

- Cheating wives

  (A.k.a. muddy children, from knowledge theory)

- Two insecure lovers

  (A.k.a. Coordinated attack, from databases and networking)

Cheating wives

First story

Cheating wives

Cheating wives

- There were one million married couples.

Cheating wives

- There were one million married couples.
- 40 wives were unfaithful

Cheating wives

- There were one million married couples.
- 40 wives were unfaithful
- Each husband knew whether other men's wives were unfaithful but he did now know whether his wife was unfaithful.

Cheating wives

- There were one million married couples.
- 40 wives were unfaithful
- Each husband knew whether other men's wives were unfaithful but he did now know whether his wife was unfaithful.
- The King of the country announced "There is at least one unfaithful wife" and publicized the following decree

Cheating wives decree

He asks the following question over and over:

can you tell for sure whether or not you are a cuckold?

Cheating wives decree

He asks the following question over and over:

can you tell for sure whether or not you are a cuckold?

Assuming that all of the men are intelligent, honest, and answer simultaneously, what will happen?

Analysis of the puzzle

First operational, then combinatorial

Operational analysis (1)

First, suppose that exactly <u>one</u> is cuckold

Operational analysis (1)

First, suppose that exactly <u>one</u> is cuckold

- He sees nobody else, can conclude that he is the one

Operational analysis (1)

First, suppose that exactly <u>one</u> is cuckold

- He sees nobody else, can conclude that he is the one
- The others cannot tell whether or not they are cuckolds

Operational analysis (1)

First, suppose that exactly <u>one</u> is cuckold

- He sees nobody else, can conclude that he is the one
- The others cannot tell whether or not they are cuckolds
- At the first question, exactly one says “yes”

Operational analysis (1)

First, suppose that exactly <u>one</u> is cuckold

- He sees nobody else, can conclude that he is the one
- The others cannot tell whether or not they are cuckolds
- At the first question, exactly one says "yes"
- At the second, all others say "no"

Operational analysis (2)

Now, suppose that exactly <u>two</u> are cuckolds

Operational analysis (2)

Now, suppose that exactly <u>two</u> are cuckolds

- They know at least two are cuckolds, because nobody spoke in first round

Operational analysis (2)

Now, suppose that exactly <u>two</u> are cuckolds

- They know at least two are cuckolds, because nobody spoke in first round
- They see only one cuckold

Operational analysis (2)

Now, suppose that exactly <u>two</u> are cuckolds

- They know at least two are cuckolds, because nobody spoke in first round
- They see only one cuckold
- At the second question, exactly two says “yes”

Operational analysis (2)

Now, suppose that exactly <u>two</u> are cuckolds

- They know at least two are cuckolds, because nobody spoke in first round
- They see only one cuckold
- At the second question, exactly two says "yes"
- At the third, all others say "no"

Operational analysis (3)

Suppose that exactly *k* are cuckolds, by induction...

Operational analysis (3)

Suppose that exactly *k* are cuckolds, by induction...

- At the *k*-th question, exactly *k* say "yes"

Operational analysis (3)

Suppose that exactly *k* are cuckolds, by induction...

- At the *k*-th question, exactly *k* say “yes”
- At the *(k+1)*-th, all others say “no”

Combinatorial analysis

Local states

Combinatorial analysis

Local states

- A local state is a man's state of knowledge

Combinatorial analysis

Local states

- A local state is a man’s state of knowledge
- It is represented by a vector: in position *i* has 0 if man *i* is known to be clean, and *1* if cuckold

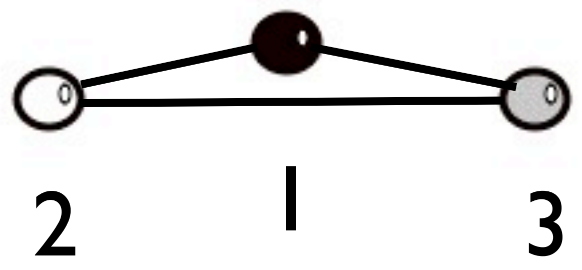
Combinatorial analysis

Local states

- A local state is a man's state of knowledge
- It is represented by a vector: in position *i* has 0 if man *i* is known to be clean, and *1* if cuckold
- Because man *i* does not know its own status, its input vector has $\perp$ in position i

Global inputs

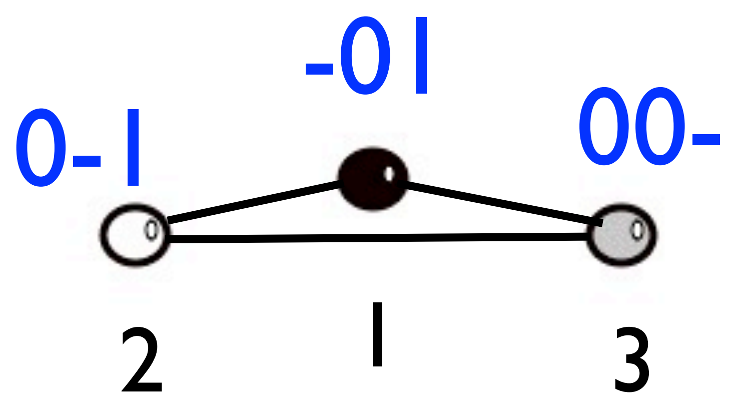
Each possible input configuration is represented as a simplex, linking compatible states for the men



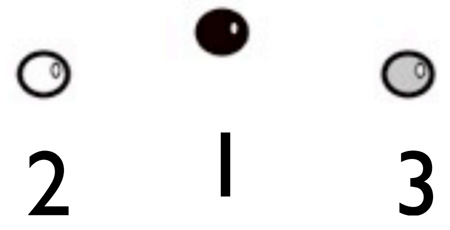
meaning that the men can be in these states together

Global inputs

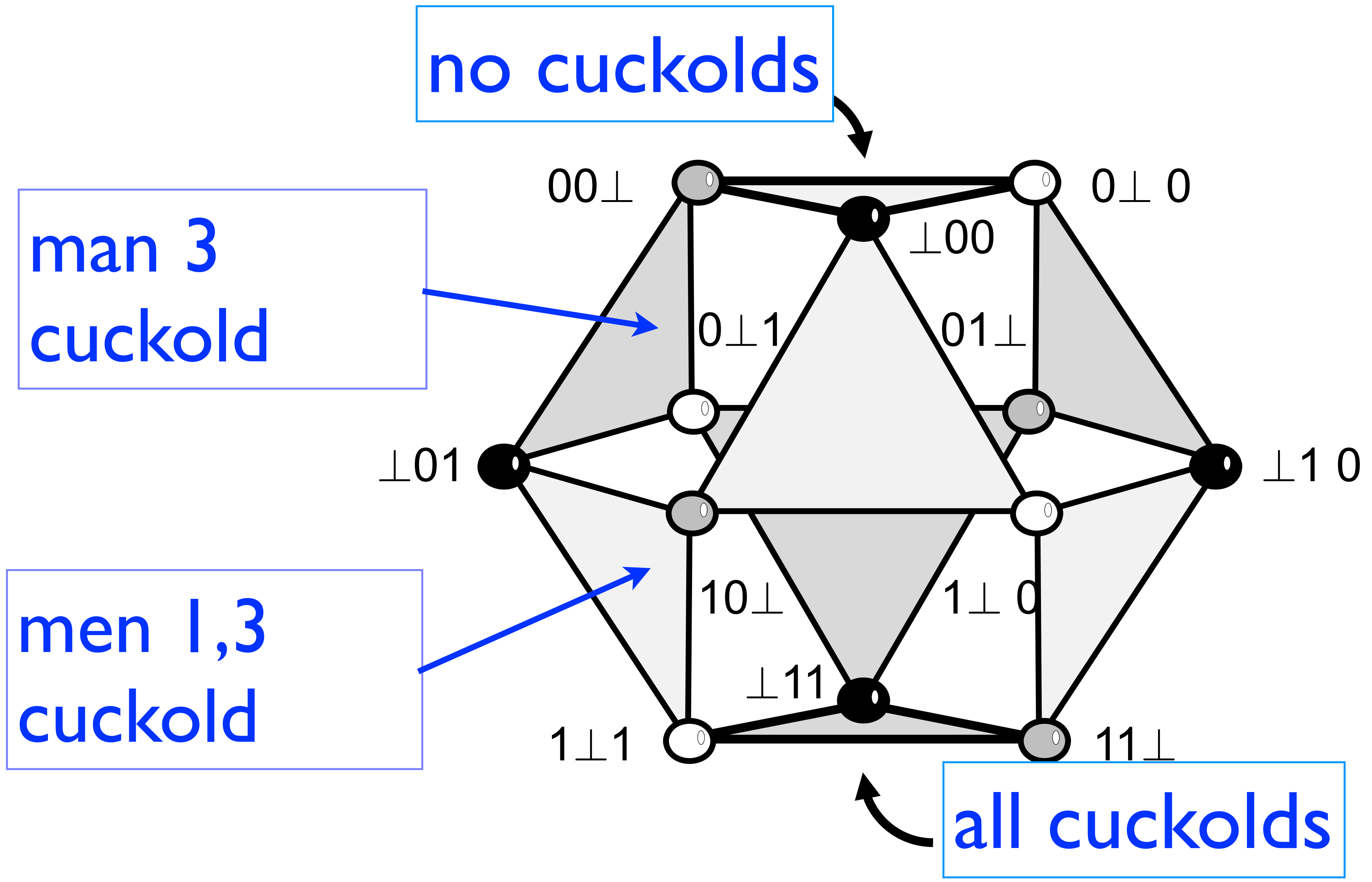
Each possible input configuration is represented as a simplex, linking compatible states for the men

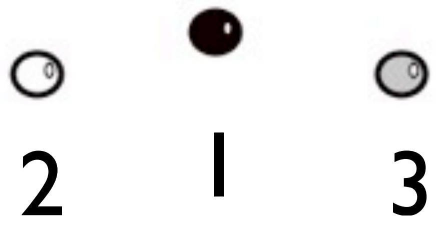


meaning that the men can be in these states together

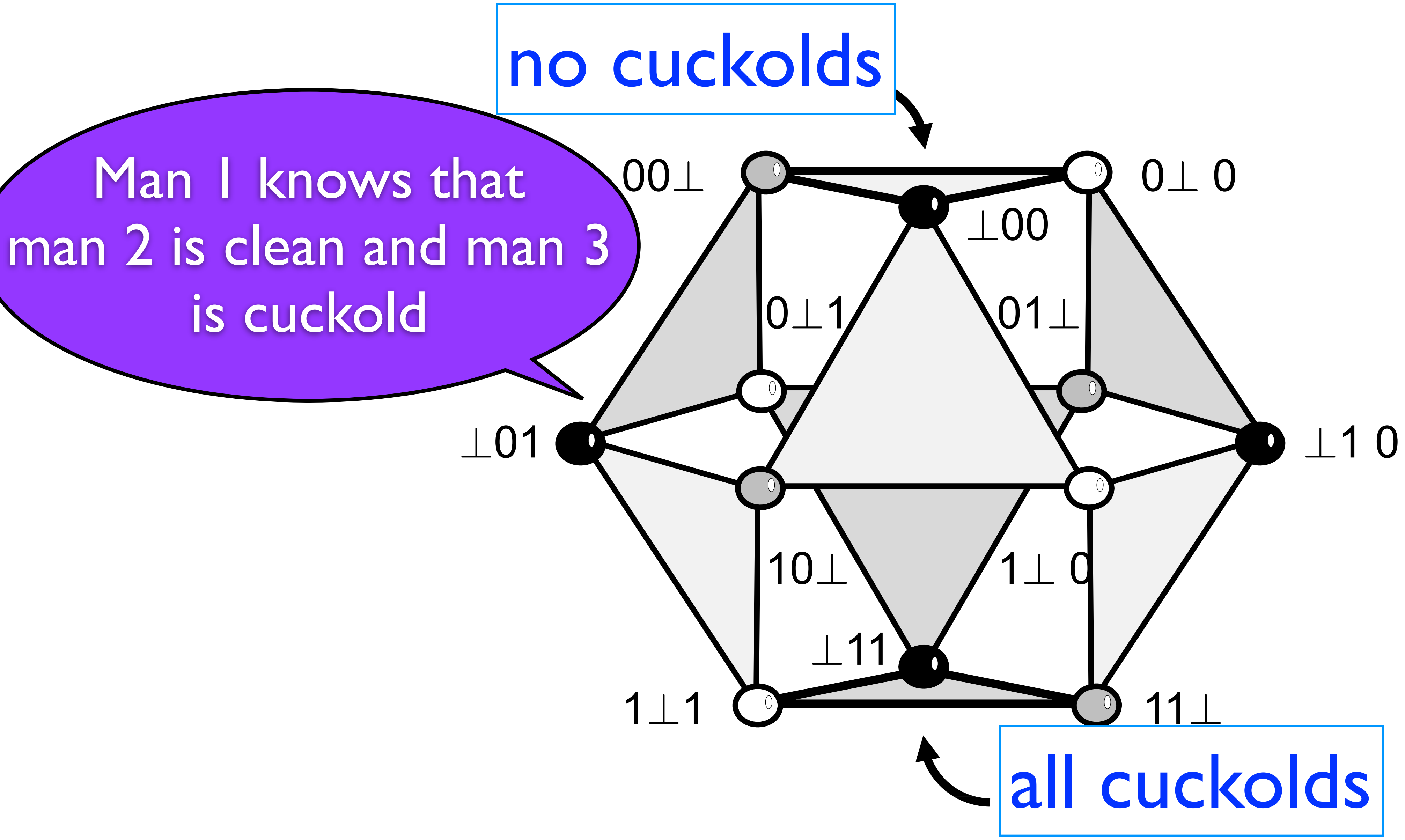


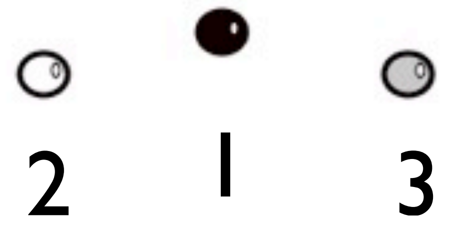
Initial Complex



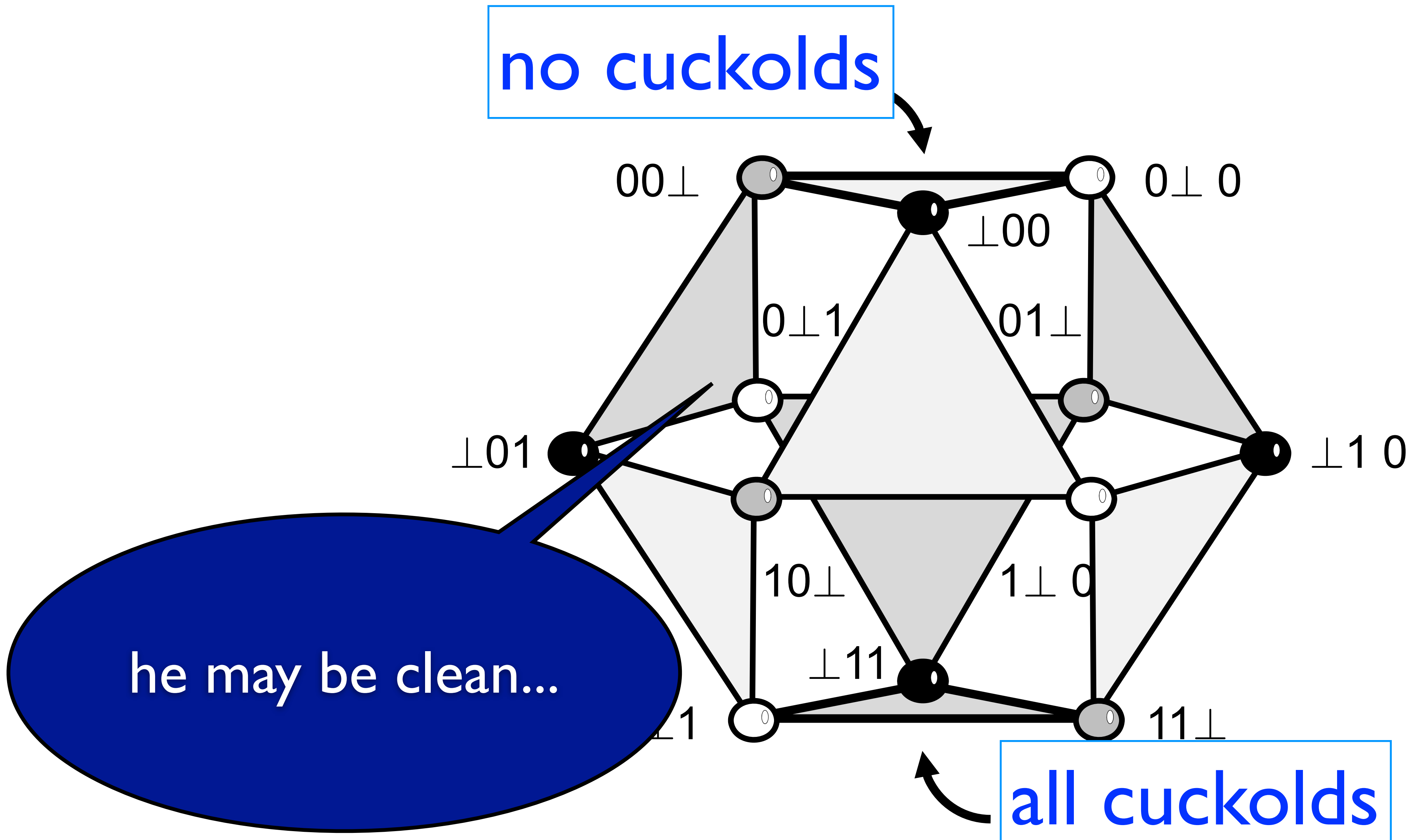


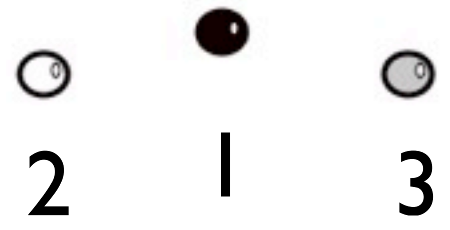
Initial Complex



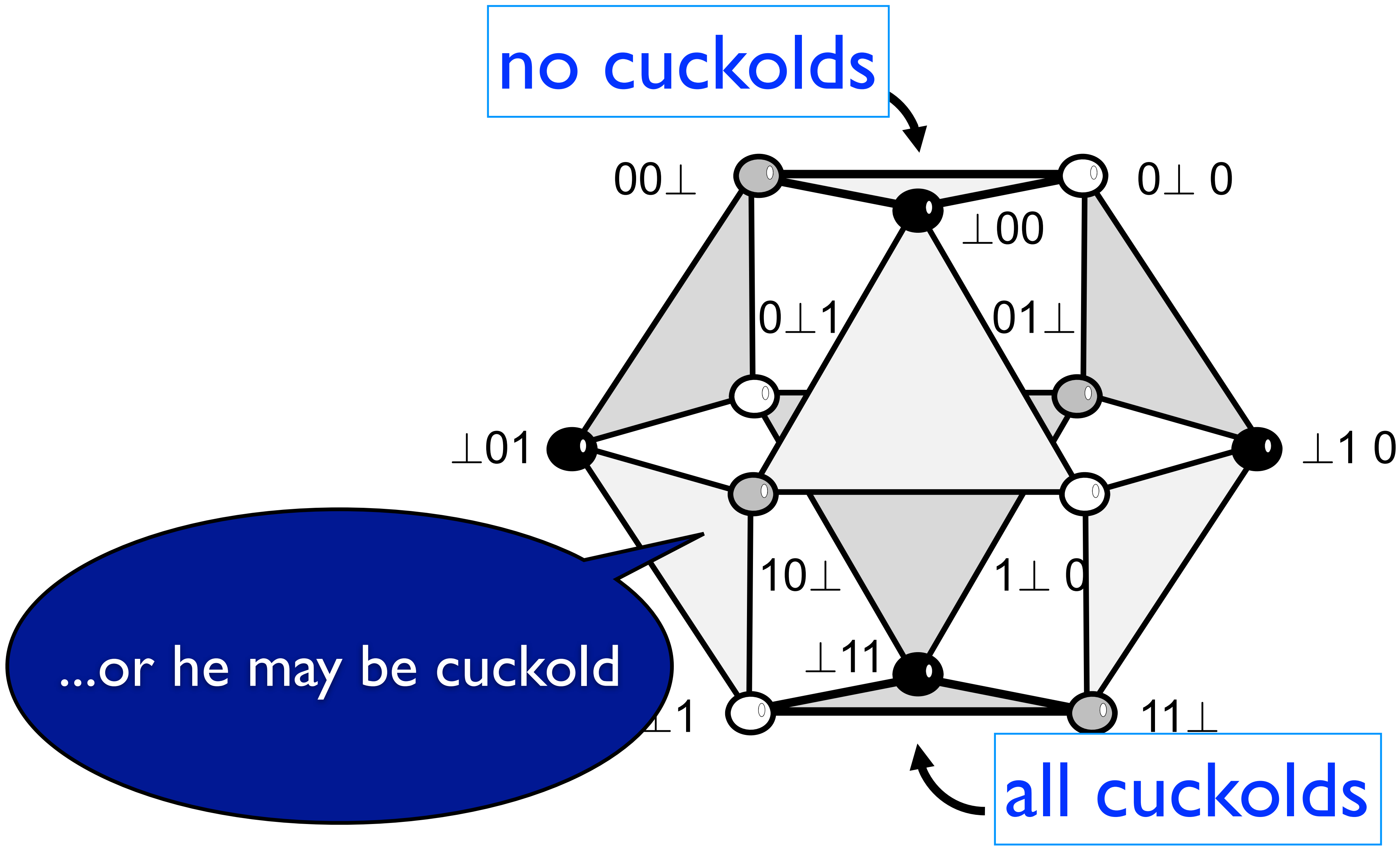


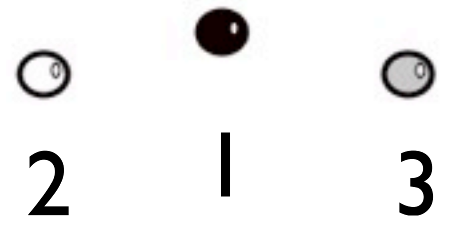
Initial Complex



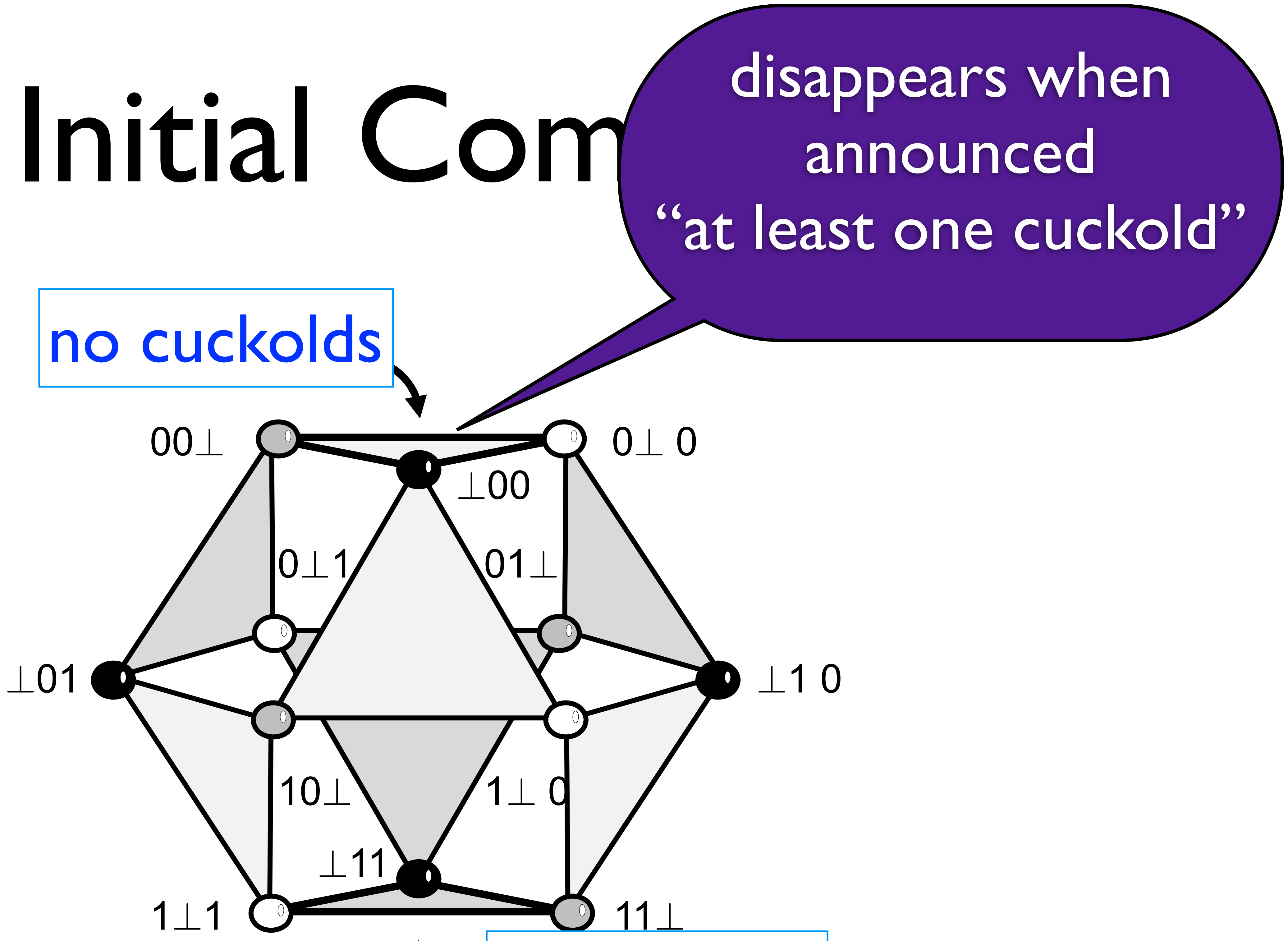


Initial Complex





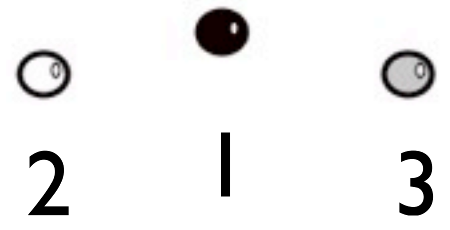
Initial Com



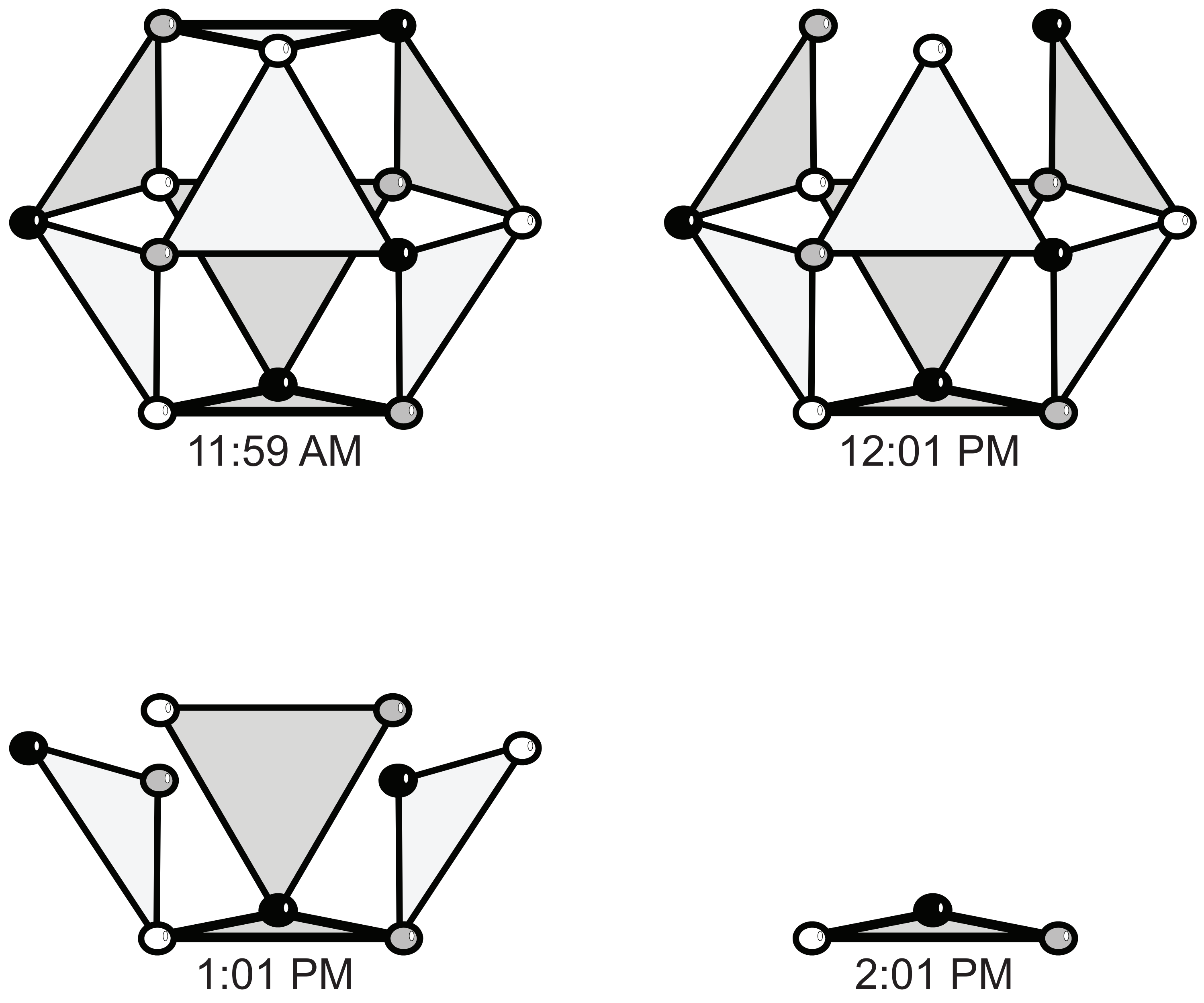
no cuckolds

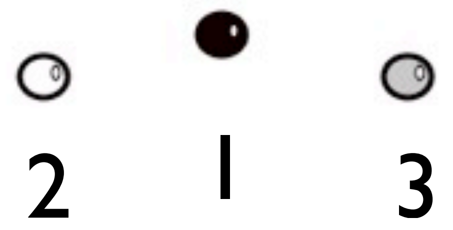
all cuckolds

that is, men know that each 2-simplex is a possible initial state, except for the one where all are clean

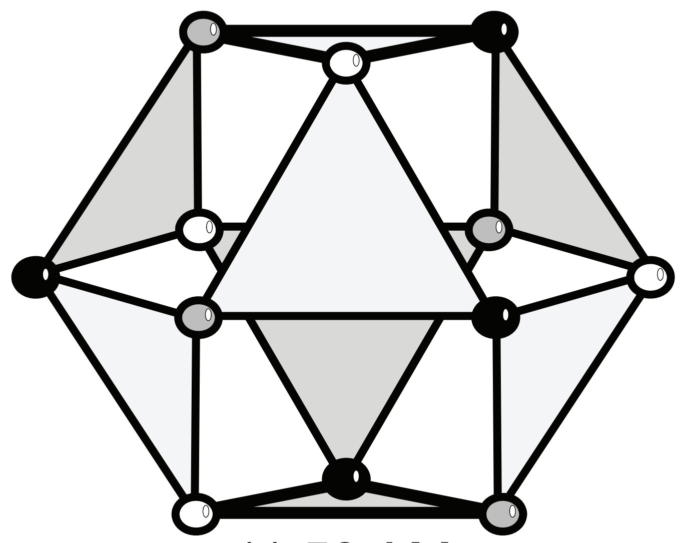


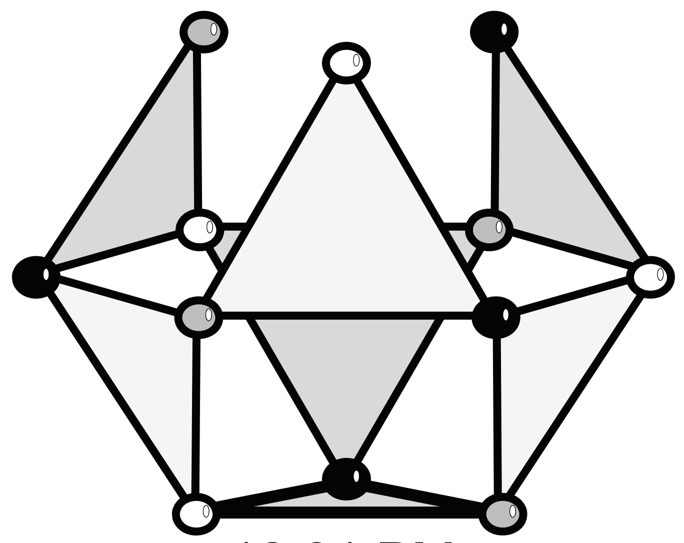
Evolution

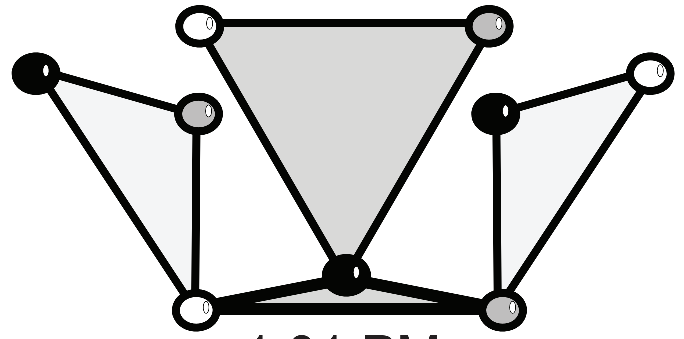


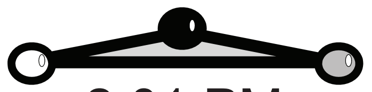


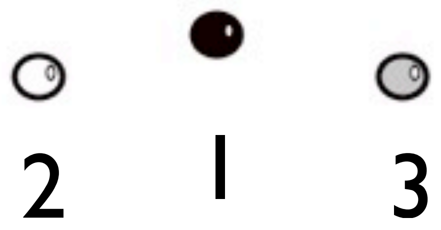
Evolution


11:59 AM


12:01 PM

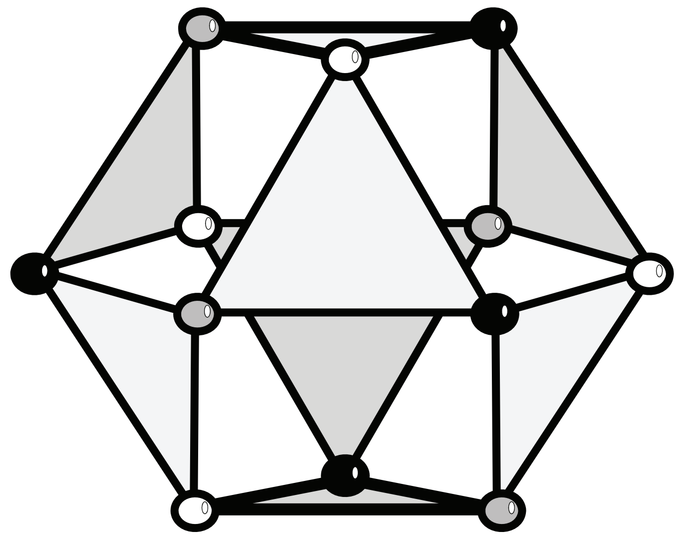

1:01 PM

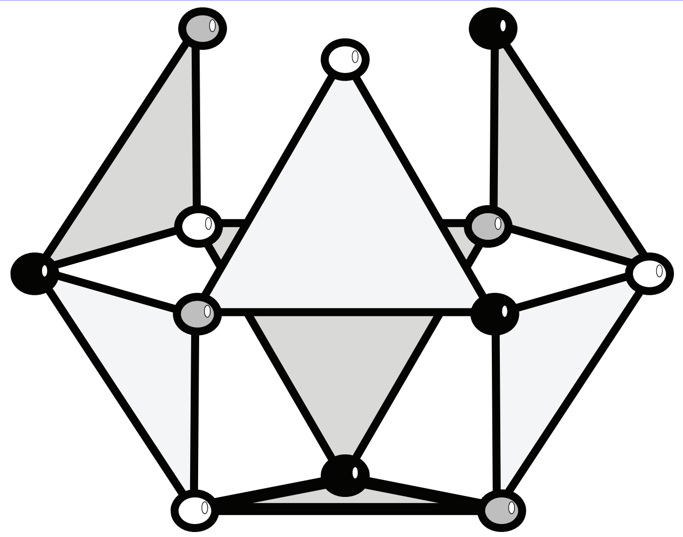

2:01 PM

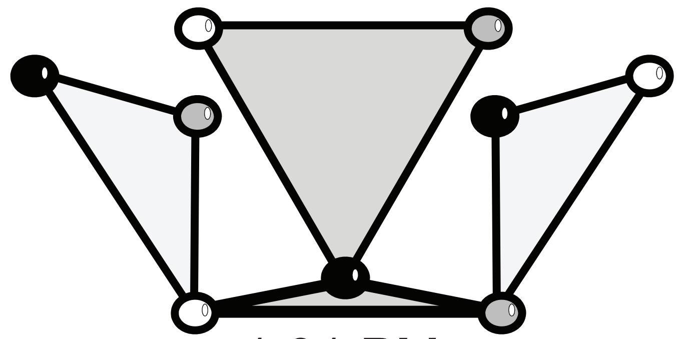


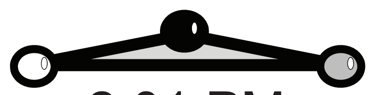
Evolution

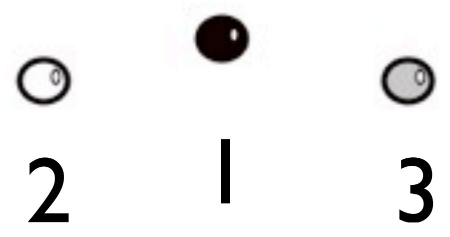
3 vertexes exposed, where someone knows its status


11:59 AM

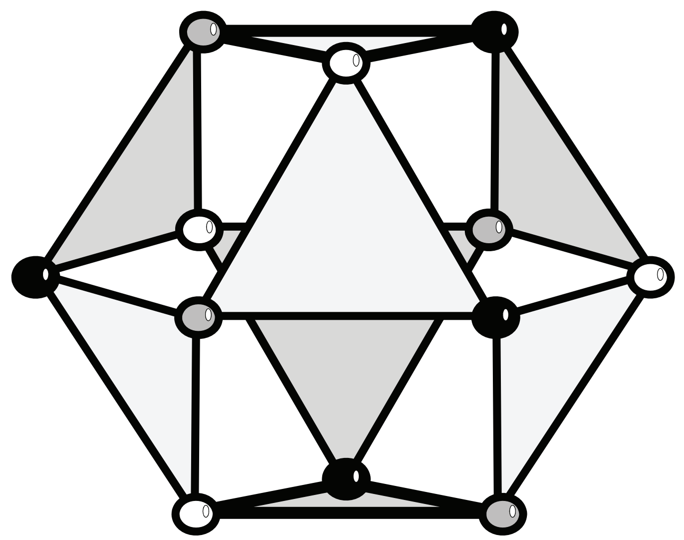

12:01 PM

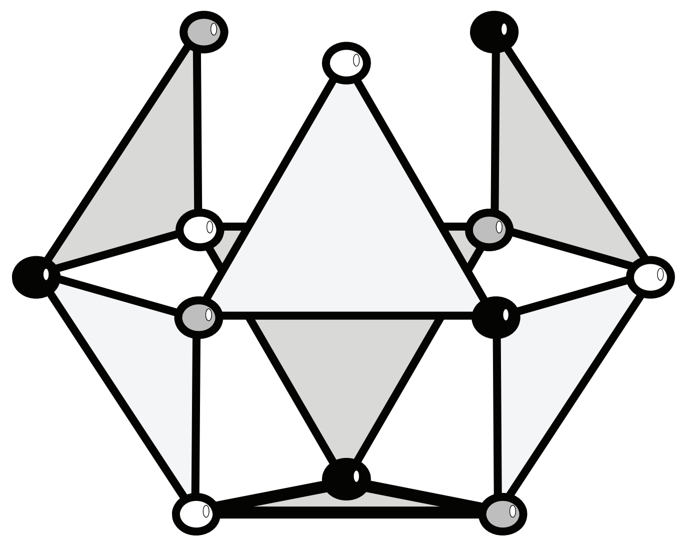

1:01 PM


2:01 PM

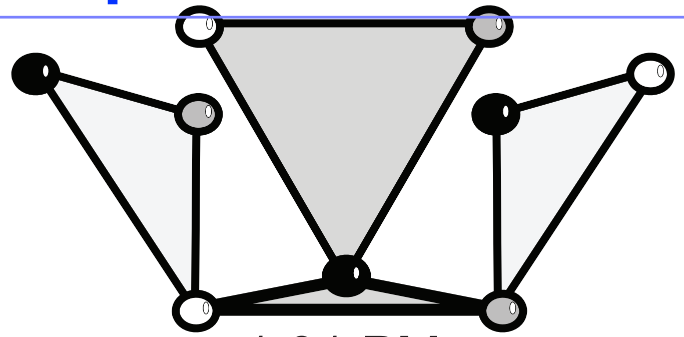


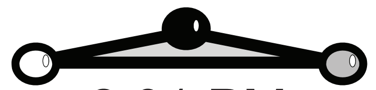
Evolution

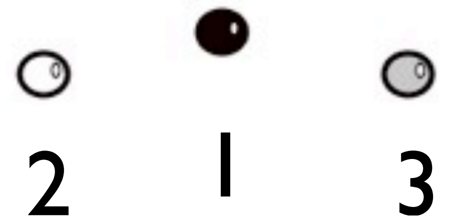

11:59 AM


12:01 PM

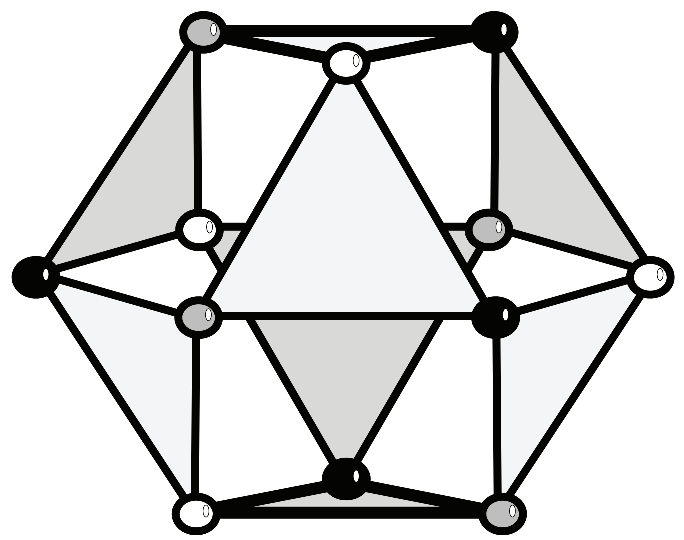
Nobody spoke previous round,
6 vertexes exposed

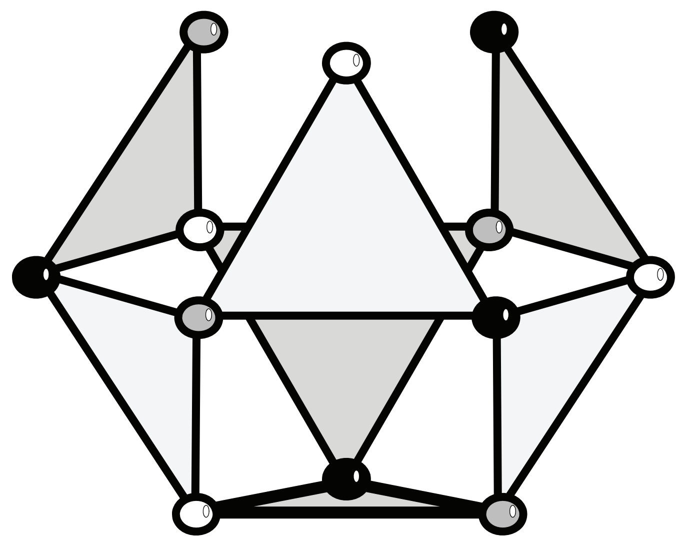

1:01 PM

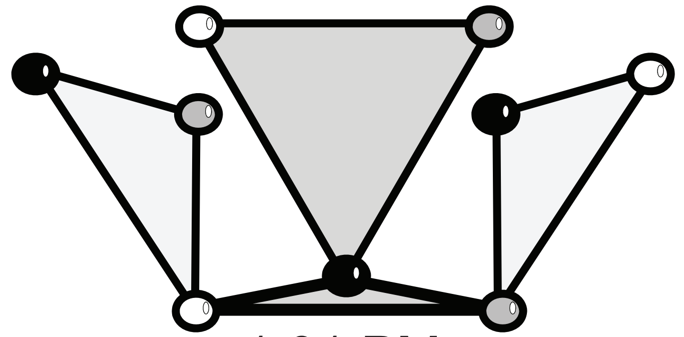

2:01 PM



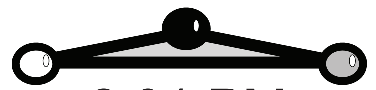
Evolution

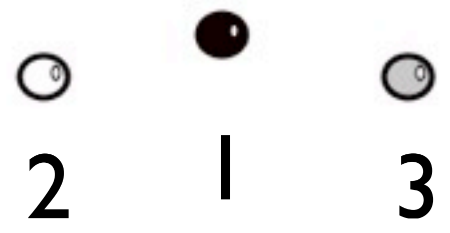

11:59 AM


12:01 PM

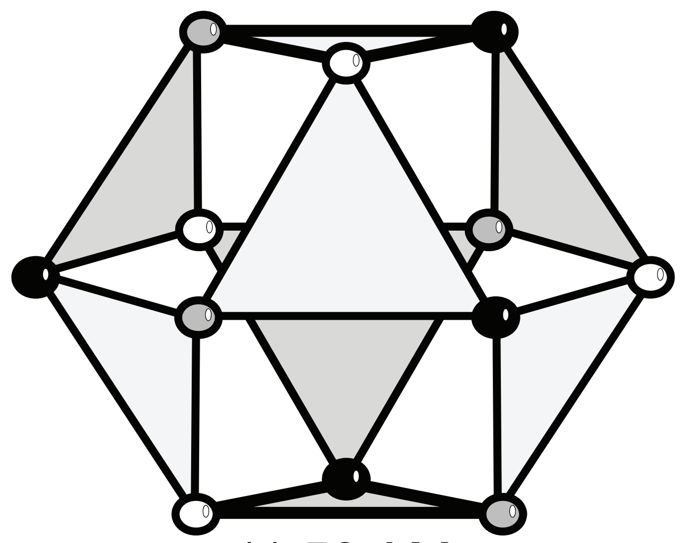

1:01 PM

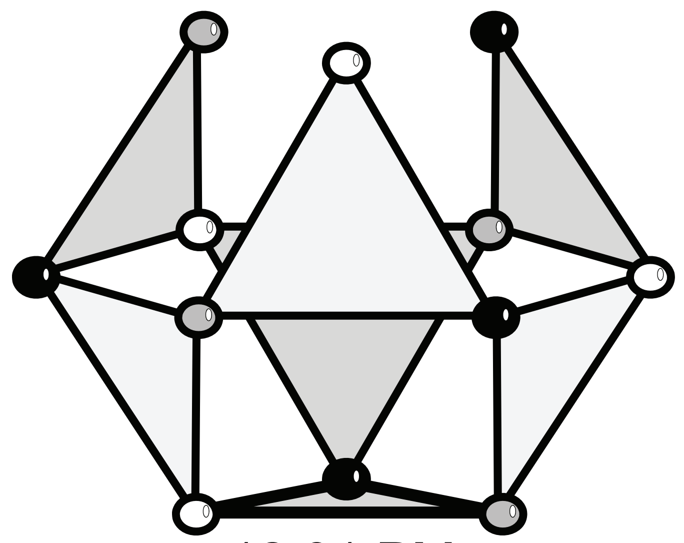
All 3 announce “cuckolds”

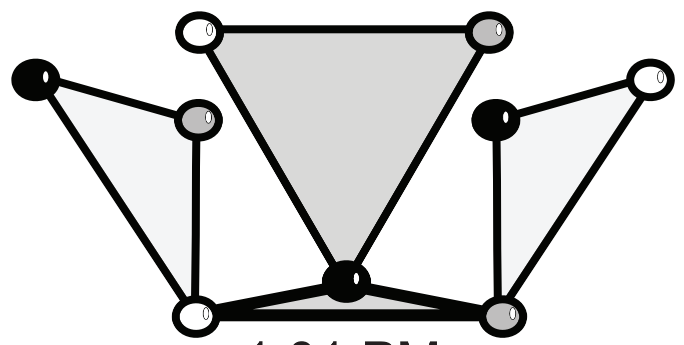

2:01 PM

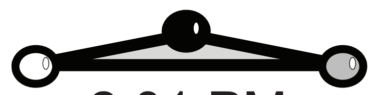


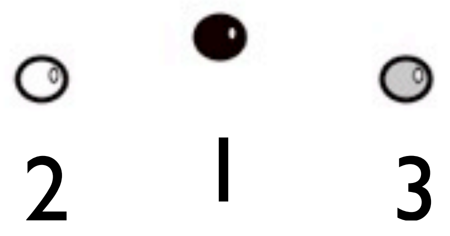
Evolution


11:59 AM

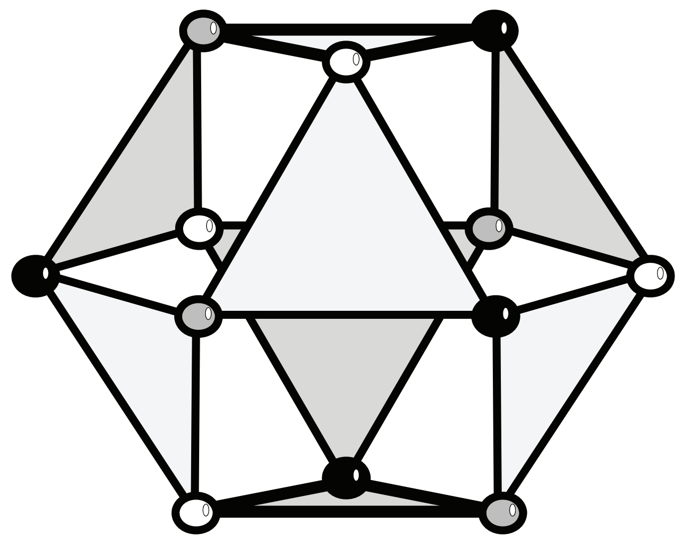

12:01 PM


1:01 PM

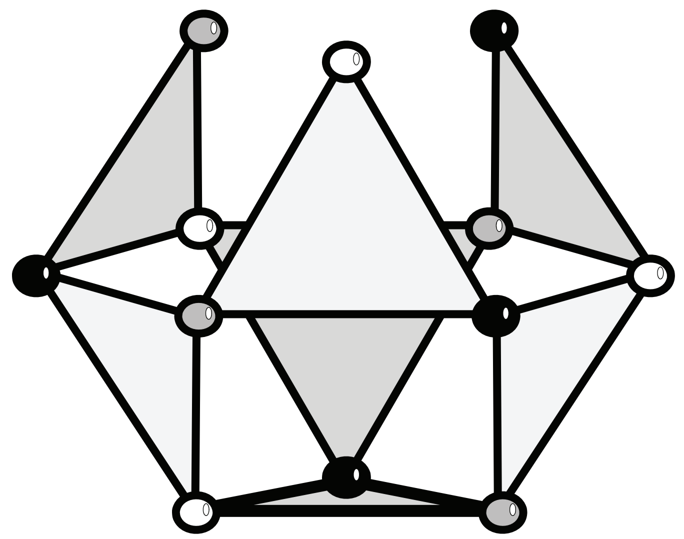

2:01 PM



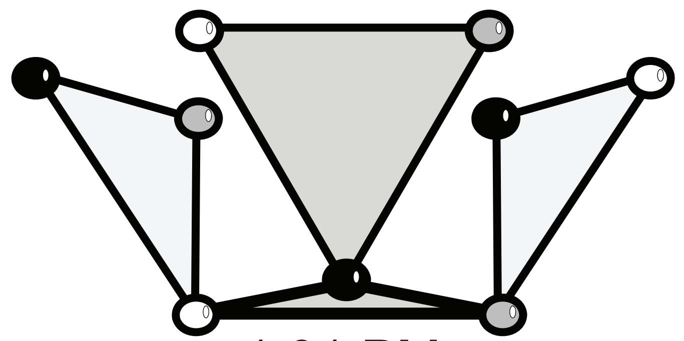
Decisions



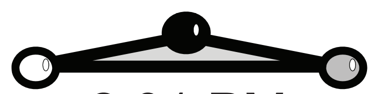
11:59 AM



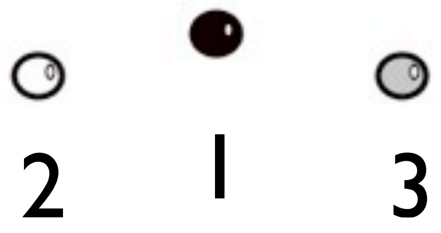
12:01 PM



1:01 PM

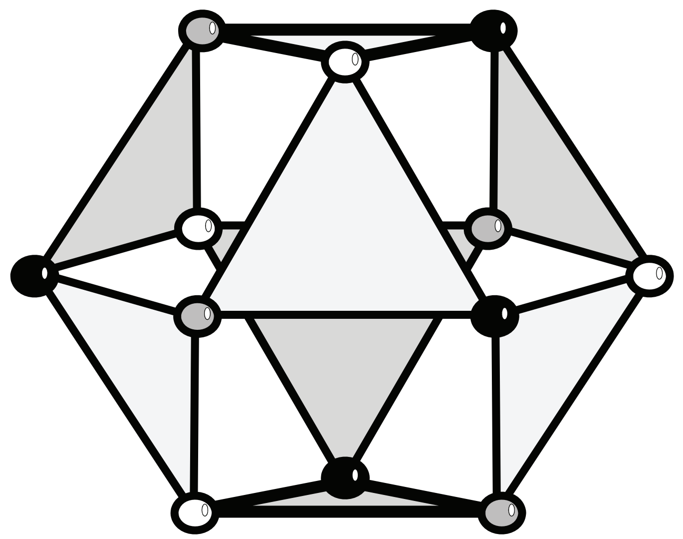


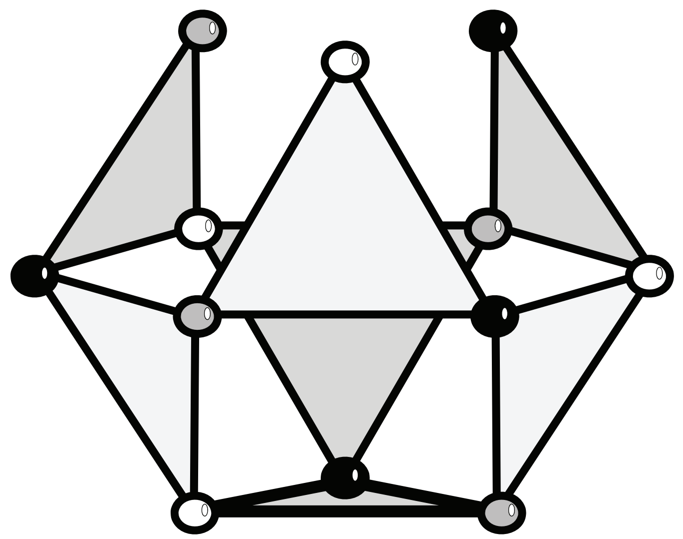
2:01 PM

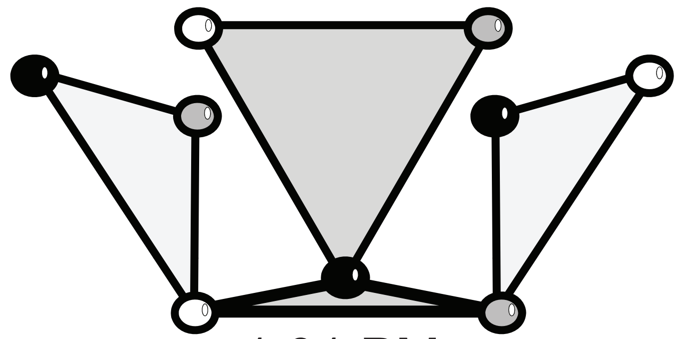


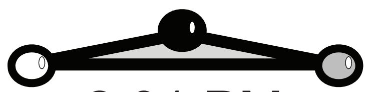
Decisions

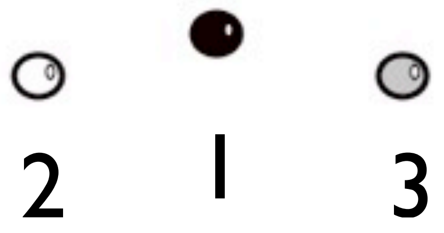
No decisions


11:59 AM


12:01 PM

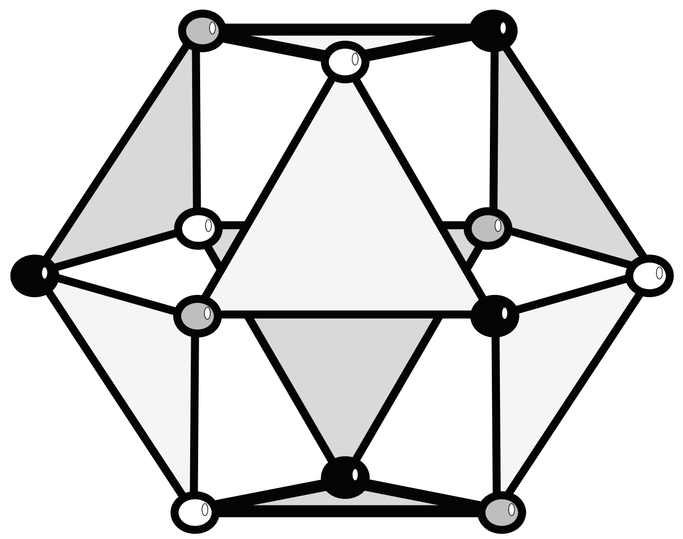

1:01 PM

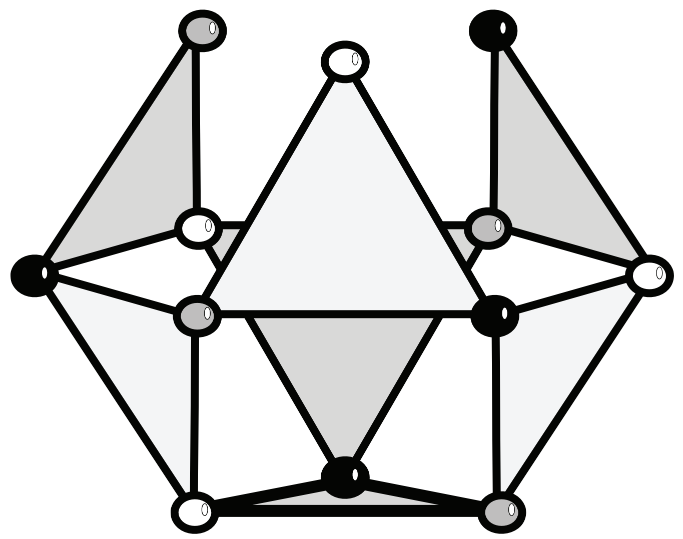

2:01 PM

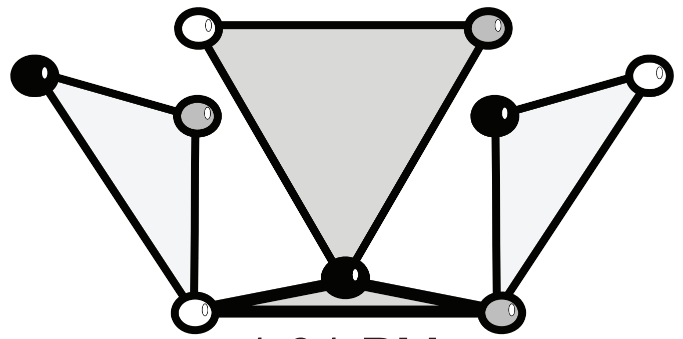


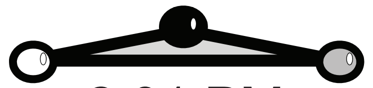
Decisions

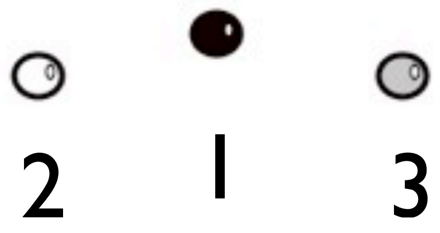
3 vertexes labeled, “cuckold”


11:59 AM

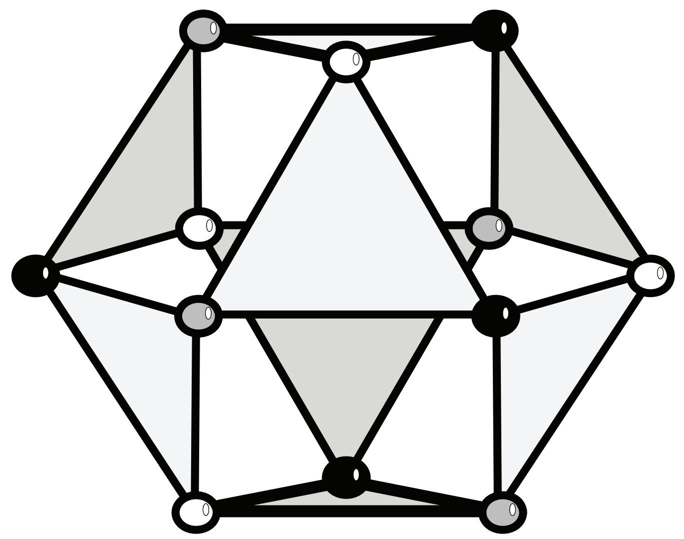

12:01 PM

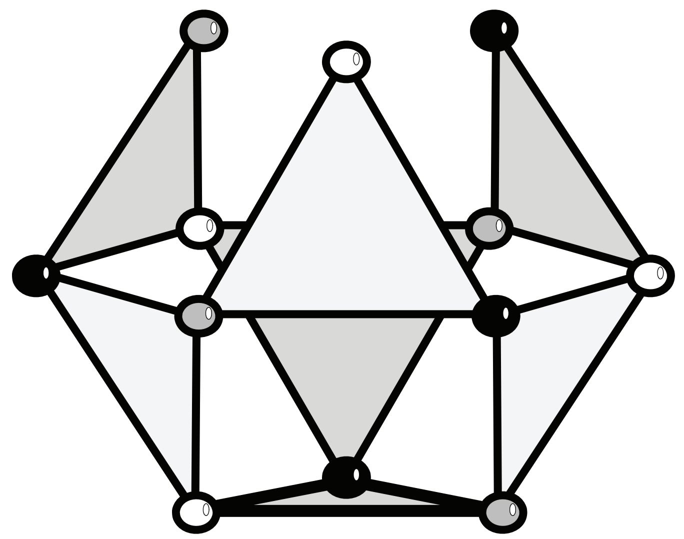

1:01 PM


2:01 PM

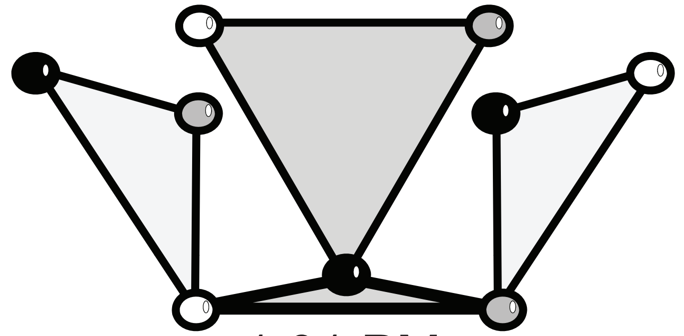


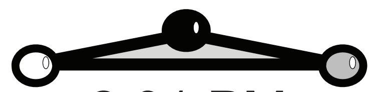
Decisions

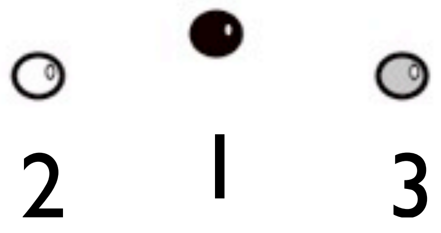

11:59 AM


12:01 PM

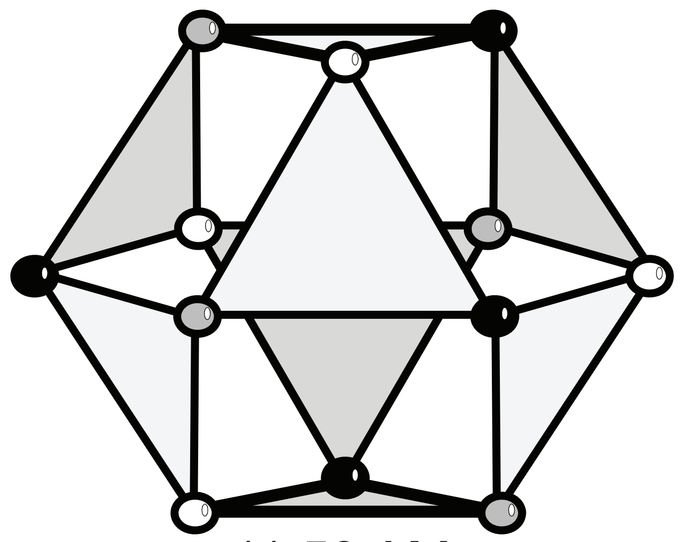
Nobody spoke previous round,
6 vertexes labeled "cuckold"

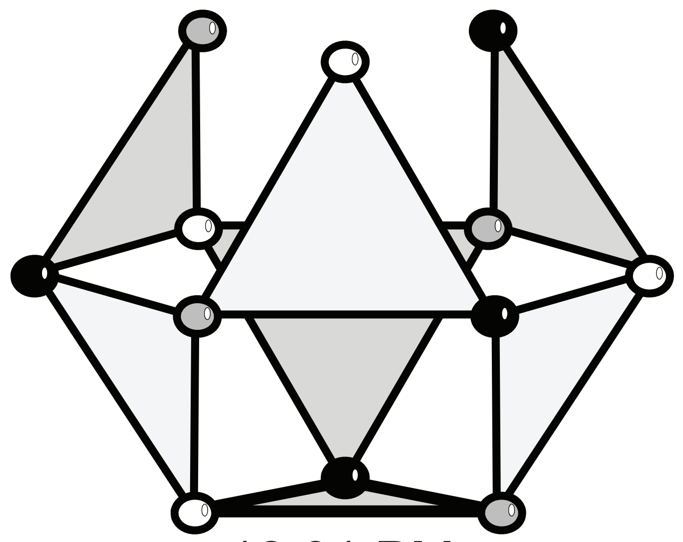

1:01 PM

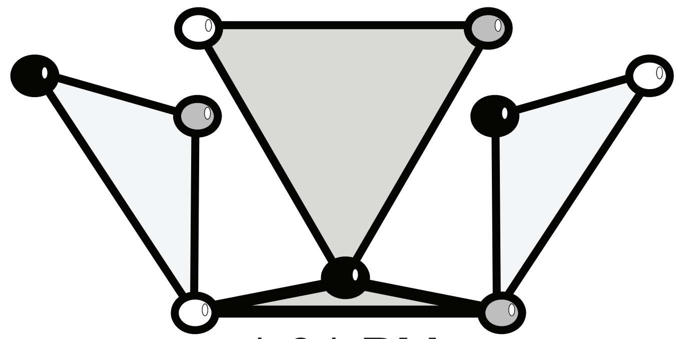

2:01 PM

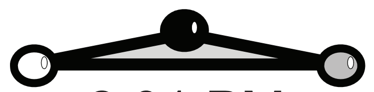


Decisions

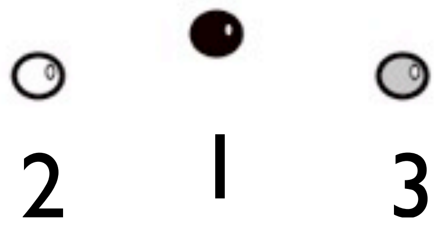

11:59 AM


12:01 PM

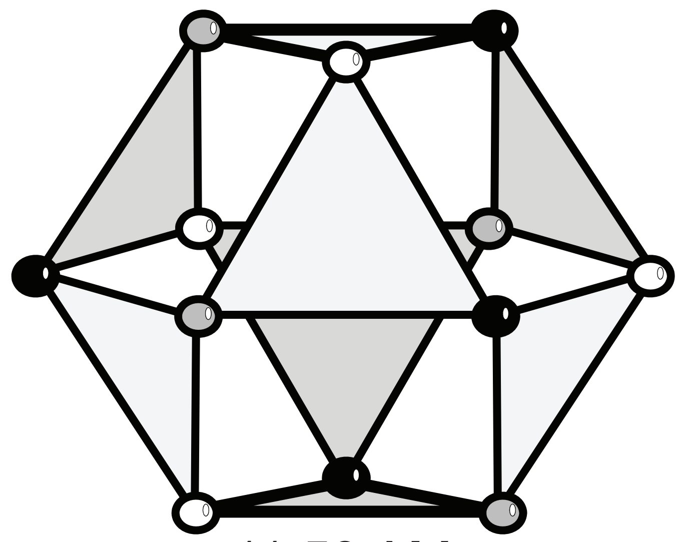

1:01 PM

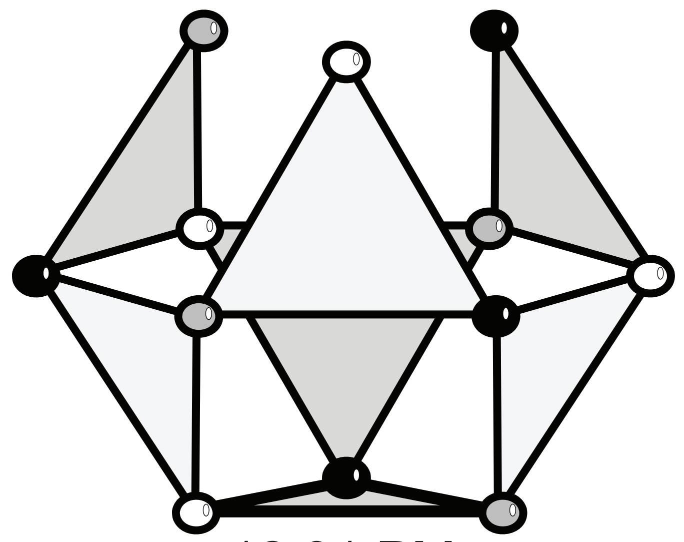

2:01 PM

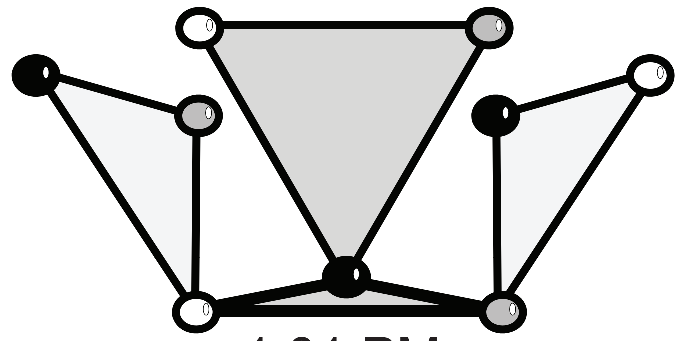
3 vertexes labeled "cuckold"

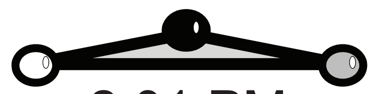


Decisions


11:59 AM

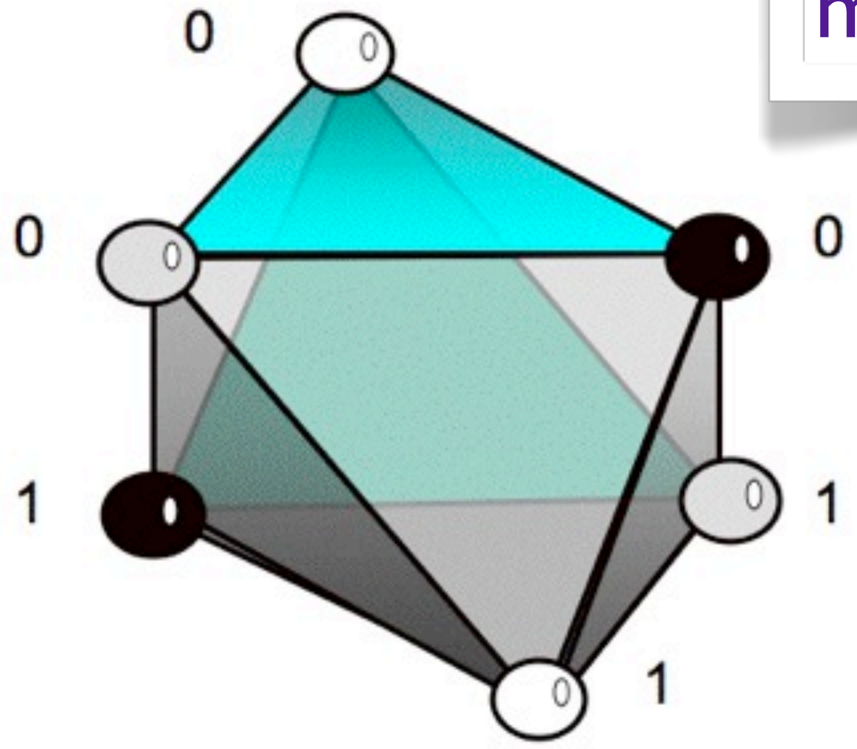

12:01 PM


1:01 PM


2:01 PM

Output complex

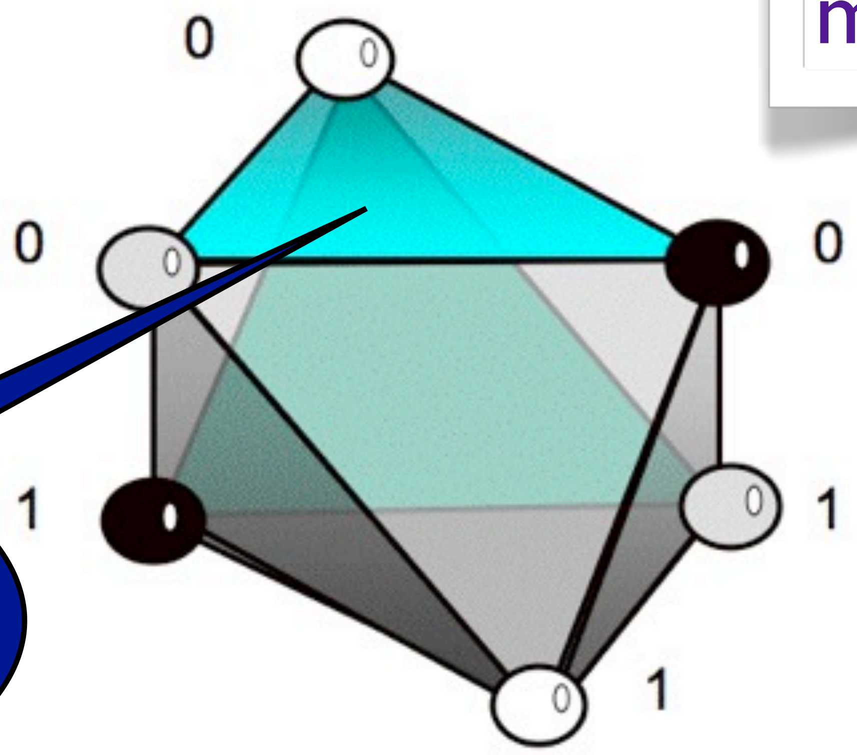
Decisions induce a map to this complex



Each man should say "yes" or "no"
All combinations are possible...

Output complex

Decisions induce a map to this complex



... except all “no” after King’s announcement

Each man should say “yes” or “no”
All combinations are possible...

Solving the cheating wives task

Each man decides an output value, on one of its local states

Decisions define a simplicial map from input complex to output complex that respects the task's specification

In this task communication is very limited. More generally, for any task...

Solving any task

In the basic, wait-free model

A task is solvable if and only if there exists a *subdivision* of the input complex and a simplicial map to the output complex that respects the task's specification

Herlihy, Shavit 1993

Wait-free: asynchronous model where any number of processes can crash

Two insecure lovers

Second story

Coordination

We often need to ensure that two things happen together or not at all.

For example, a banking system needs to ensure that if an automatic teller dispenses cash, then the corresponding account balance is debited, and vice-versa.

Two insecure lovers

Two insecure lovers

- Alice and Bob want to schedule a meeting.

Two insecure lovers

- Alice and Bob want to schedule a meeting.
- If both attend, they win, but if only one attends, defeat and humiliation is felt.

Two insecure lovers

- Alice and Bob want to schedule a meeting.
- If both attend, they win, but if only one attends, defeat and humiliation is felt.
- As a result, neither will show up without a guarantee that the other will show up at the same time.

Two insecure lovers

- Alice and Bob want to schedule a meeting.
- If both attend, they win, but if only one attends, defeat and humiliation is felt.
- As a result, neither will show up without a guarantee that the other will show up at the same time.
- Communication is be SMS only.

Communication problems

- Normally, it takes a message one hour to arrive.
- However, it is possible that it is gets lost.

The puzzle

Fortunately, on this particular night,
all the messages arrive safely.

How long will it take Alice and Bob
to coordinate their meeting?

Analysis of the puzzle

First operational, then combinatorial

Operational analysis (1)

Suppose Alice initiates the communication

Operational analysis
(1)

Operational analysis (1)

- Suppose Bob receives a message at 1:00 from Alice saying "meet at midnight". Should Bob show up?

Operational analysis (1)

- Suppose Bob receives a message at 1:00 from Alice saying "meet at midnight". Should Bob show up?
- Although her message was in fact delivered, Alice does not know. She therefore considers it possible that Bob did not receive the message.

Operational analysis (1)

- Suppose Bob receives a message at 1:00 from Alice saying "meet at midnight". Should Bob show up?
- Although her message was in fact delivered, Alice does not know. She therefore considers it possible that Bob did not receive the message.
- Hence Alice cannot decide to show up, given her current state of knowledge.

Operational analysis (1)

- Suppose Bob receives a message at 1:00 from Alice saying "meet at midnight". Should Bob show up?
- Although her message was in fact delivered, Alice does not know. She therefore considers it possible that Bob did not receive the message.
- Hence Alice cannot decide to show up, given her current state of knowledge.
- Knowing this, Bob will not show up based solely on Alice's message.

Operational analysis (2)

Operational analysis (2)

- Naturally, Bob reacts by sending an acknowledgment back to Alice, which arrives at 2:00

Operational analysis (2)

- Naturally, Bob reacts by sending an acknowledgment back to Alice, which arrives at 2:00
- Will Alice plan to show up?

Operational analysis (2)

- Naturally, Bob reacts by sending an acknowledgment back to Alice, which arrives at 2:00
- Will Alice plan to show up?
- Unfortunately, Alice's predicament is similar to Bob's predicament at 1:00, she cannot yet decide to show up

No number of successfully delivered acknowledgments will be enough to ensure that show up safely!

The key insight is that the difficulty is not caused by what actually happens (all messages actually arrive) but by the uncertainty regarding what might have happened.

Combinatorial analysis

Combinatorial analysis

- Initially Alice has two possible decisions: meet at dawn, or meet at noon the next day.

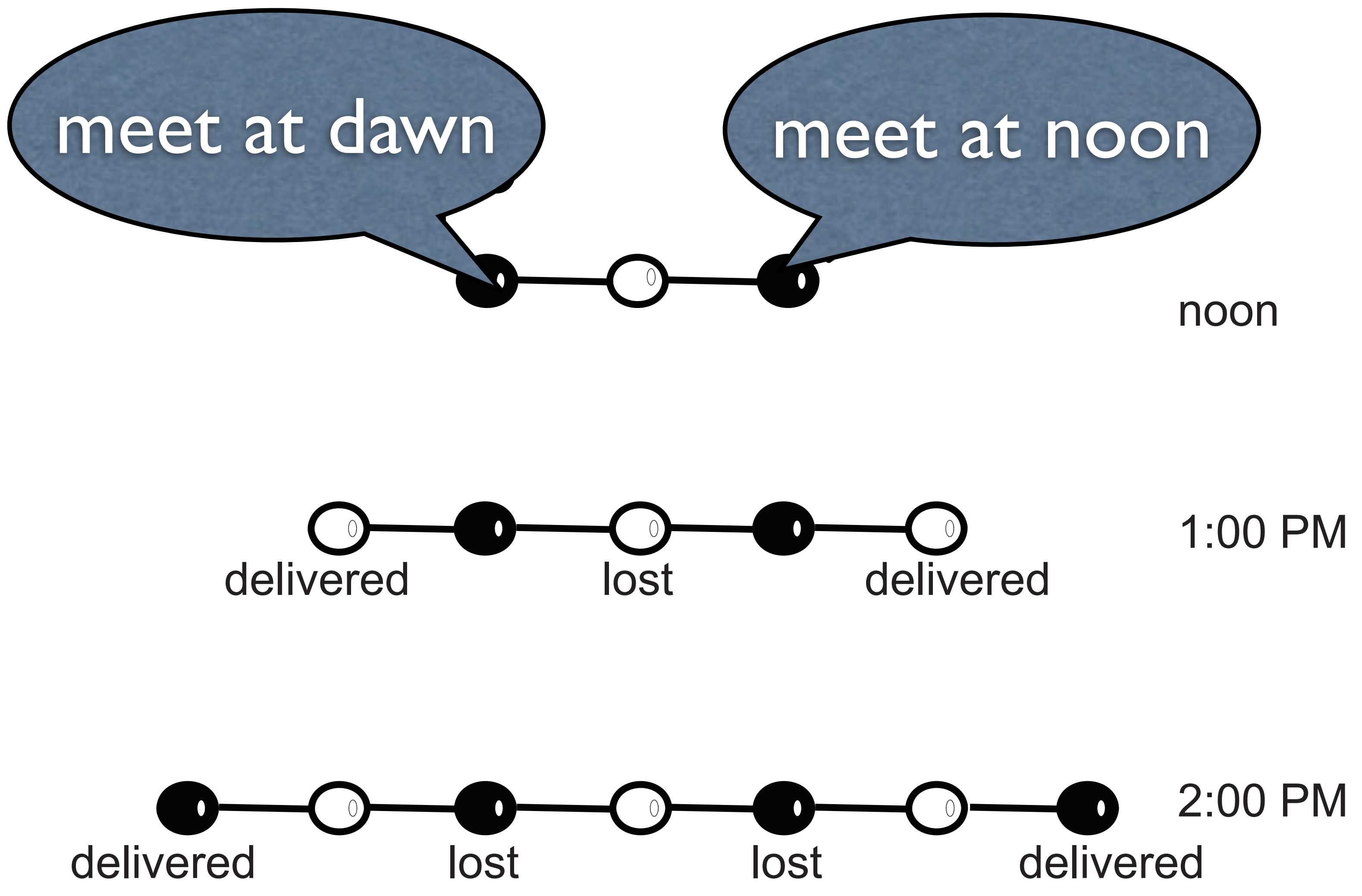
Combinatorial analysis

- Initially Alice has two possible decisions: meet at dawn, or meet at noon the next day.
- Bob has only one initial state, the white vertex in the middle, waiting to hear Alice's preference.

Combinatorial analysis

- Initially Alice has two possible decisions: meet at dawn, or meet at noon the next day.
- Bob has only one initial state, the white vertex in the middle, waiting to hear Alice's preference.
- This vertex belongs to two edges (simplexes)

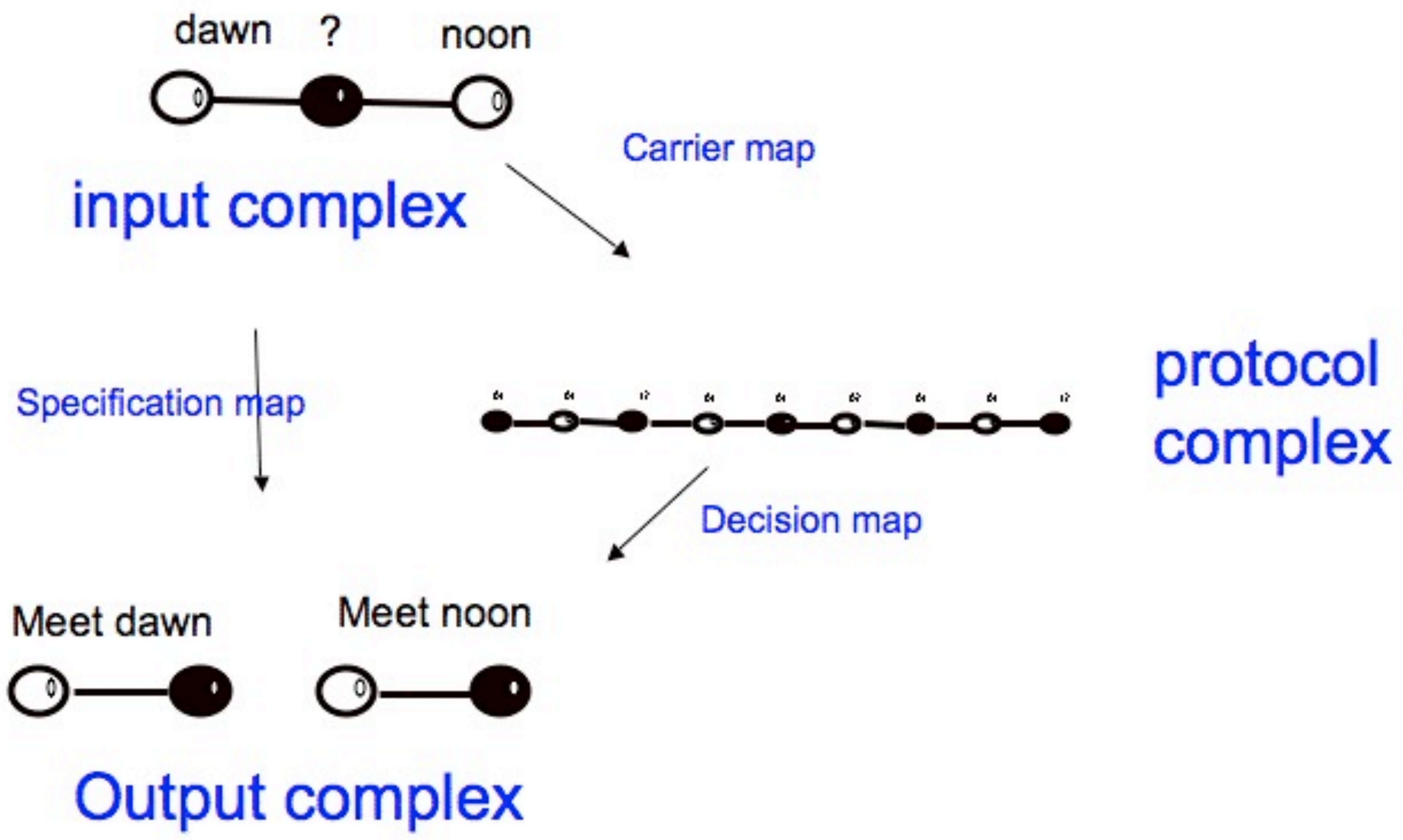
Evolution


meet at dawn
meet at noon
noon
1:00 PM
delivered
lost
delivered
2:00 PM
delivered
lost
lost
delivered

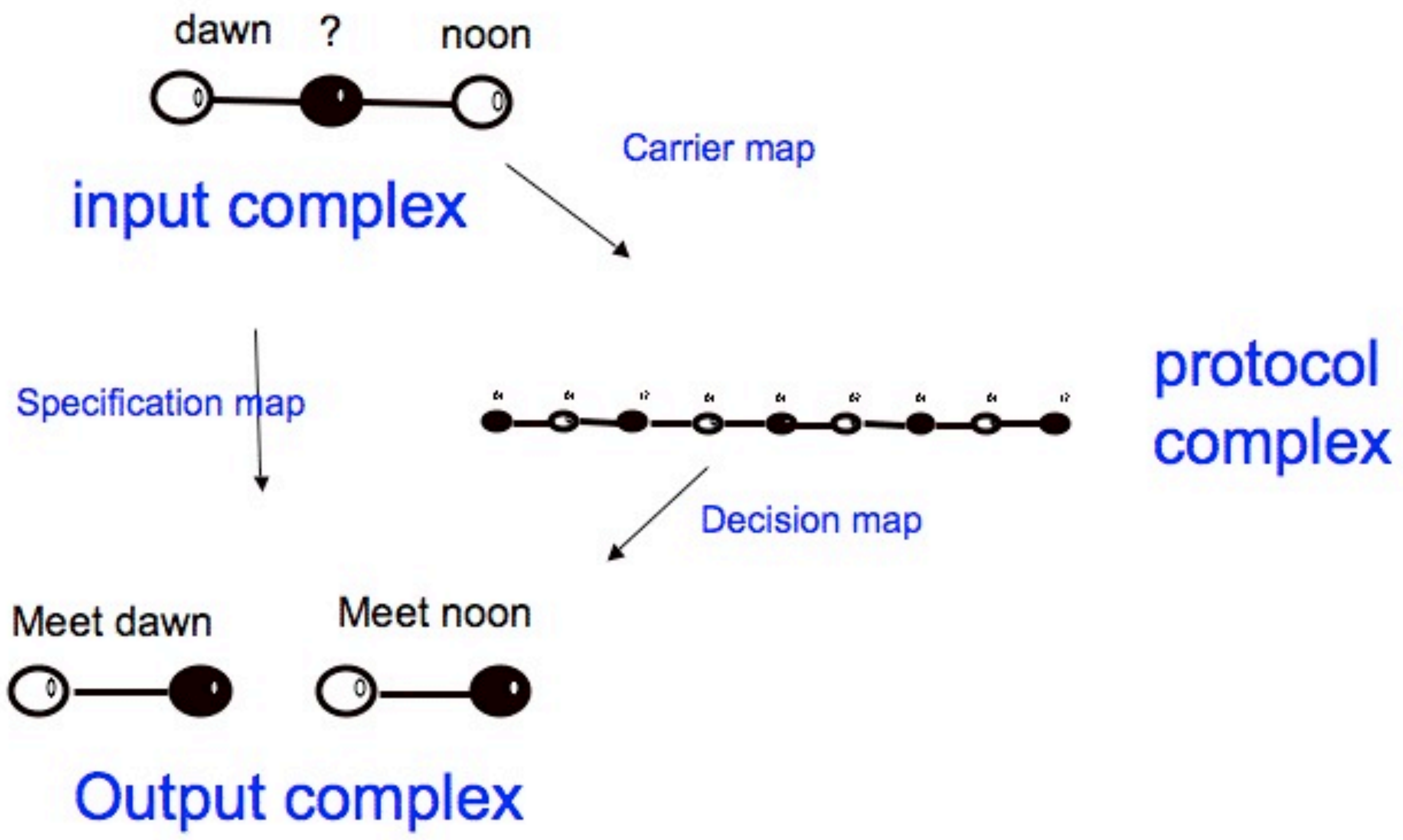
Topology implies impossibility

No number of successfully delivered acknowledgments will be enough to ensure that show up safely, because the complex is subdivided, and remains connected!

No number of successfully delivered acknowledgments will be enough to ensure that show up safely!



Because not possible to map a connected input complex into a disconnected output complex

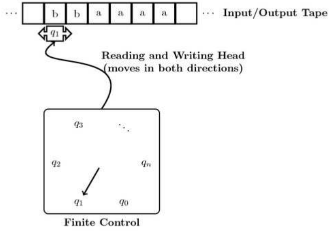


Distributed Computing

Three epochs of computing (1)

In the beginning there was <u>sequential computing</u>

- The model of choice for theory of computation– Turing machine
- provides a precise definition of a "mechanical procedure"



- Turing Year 2012 birth centenary
- STOC, and Symposium on Switching and Automata Theory (1966), today FOCS

Three epochs of computing (2)

Then there was parallel computing

- Model of choice– PRAM
- Several processes computing in parallel
- All executing computation steps synchronously
- No process and no communication failures
- Symp. Parallel Algor. and Architectures, SPAA (1989)

- In 2007 Kanbalam put UNAM at number 28 among universities, 1,368 processors at a cost of 3 million dollars

Three epochs of computing (2)

Parallel computing

- 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.

Three epochs of computing (3)

Distributed computing is everywhere!

- Nearly every activity in our society works as a distributed system made up of human and computer processes
- “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”

Why is distributed computing different?

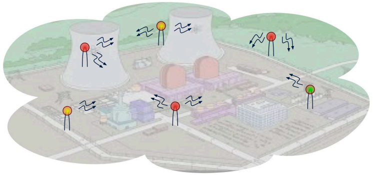
A system observed by several monitors

- Does a system satisfy a certain property ϕ ?

- Property ϕ is typically expressed in a linear temporal logics

Failure free $\sim$ parallel

- Monitors can exchange their obser-
  vations and agree on the state of the system, to evaluate ϕ

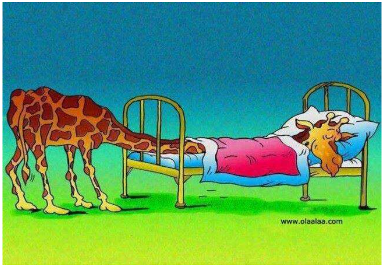


Distributed system being monitored

- Techniques such as augmenting events with vector-clock information, Chandy-Lamport snapshots

Distributed computing

- Monitors may *fail* by undetectable crashes
- Asynchronous communication- unpredictable delays



The science of perspectives

- Each monitor has its own perspective of the global state of the system

The science of perspectives

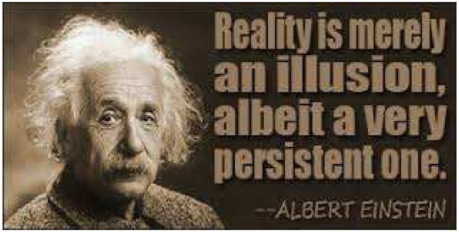
- Nobody can observe the global state

XVIII c.

One's subjective experience can be true, but such experience is inherently limited by its failure to account for other truths or a totality of truth.

The science of perspectives

- Is there a global state??


Reality is merely an illusion, albeit a very persistent one.
--ALBERT EINSTEIN

The science of perspectives

Multiperspectivism in literature, movies, etc.

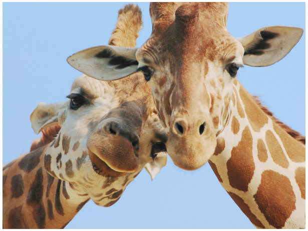
(Roshomon, Kurosawa)

- A mode of storytelling in which multiple viewpoints are employed for the presentation of a story
- To draw attention to various kinds of differences and similarities between the points of view presented therein.

the only authentic approach to the problem of reality is one which allows multiple perspectives to be heard in debate with each other (Schonfield)

The science of perspectives

In distributed computing we study how perspectives evolve with time, as unreliable communication takes place.



- And we know by talking we may get closer to each other: approximate agreement [DLPSW86]
- but never get there: consensus is impossible (even if only one process can crash, asynchronous) [FLP85]

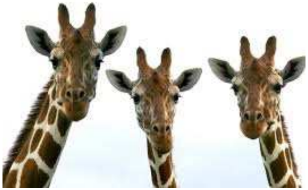
The science of perspectives

Why can't we agree?
(or why can't we get closer to each other faster)

- Even in execution with no failures agreement may be impossible
- The *possibility* of other worlds existing (failures), affect what is achievable

Consequence for the monitors

Given that there are different perspectives,
different opinions about the validity of ϕ are unavoidable!

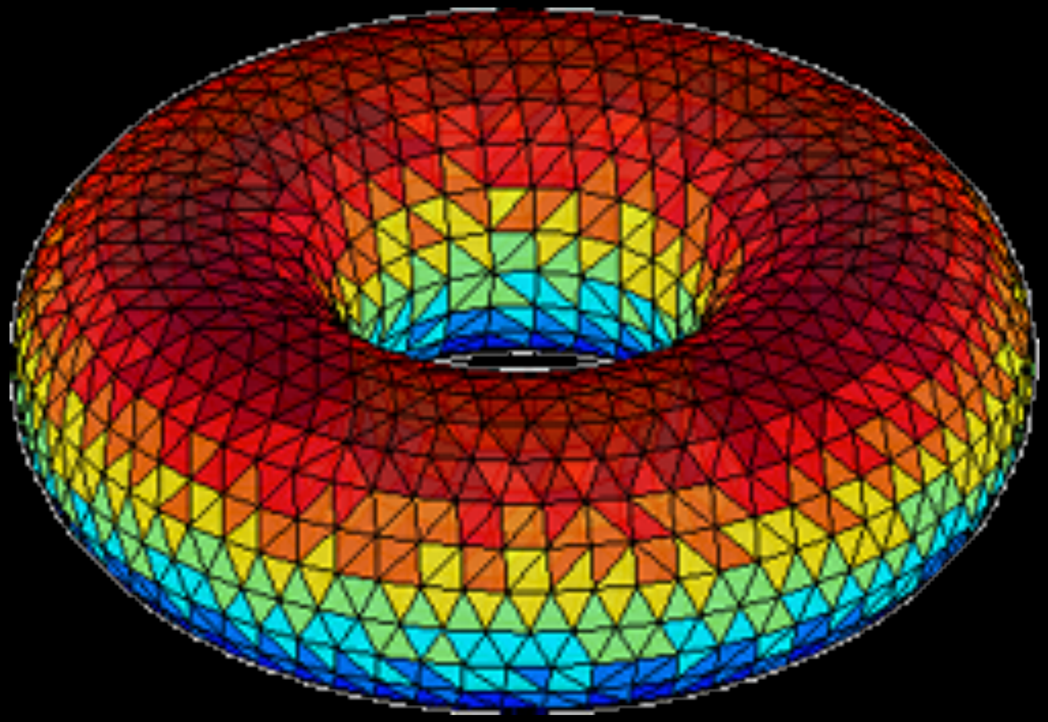


- The number of opinions needed depends on ϕ [Fraigniaud, R, Travers RV2014]

Distributed Computing and Topology

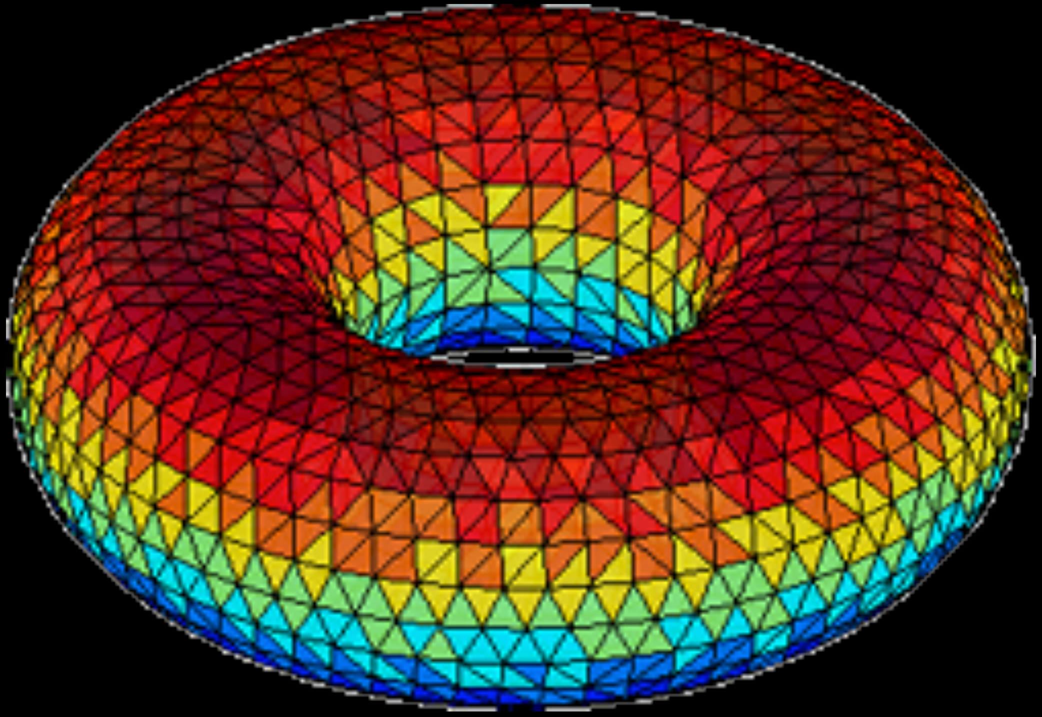
Topology

Placing together all these views yields a simplicial complex



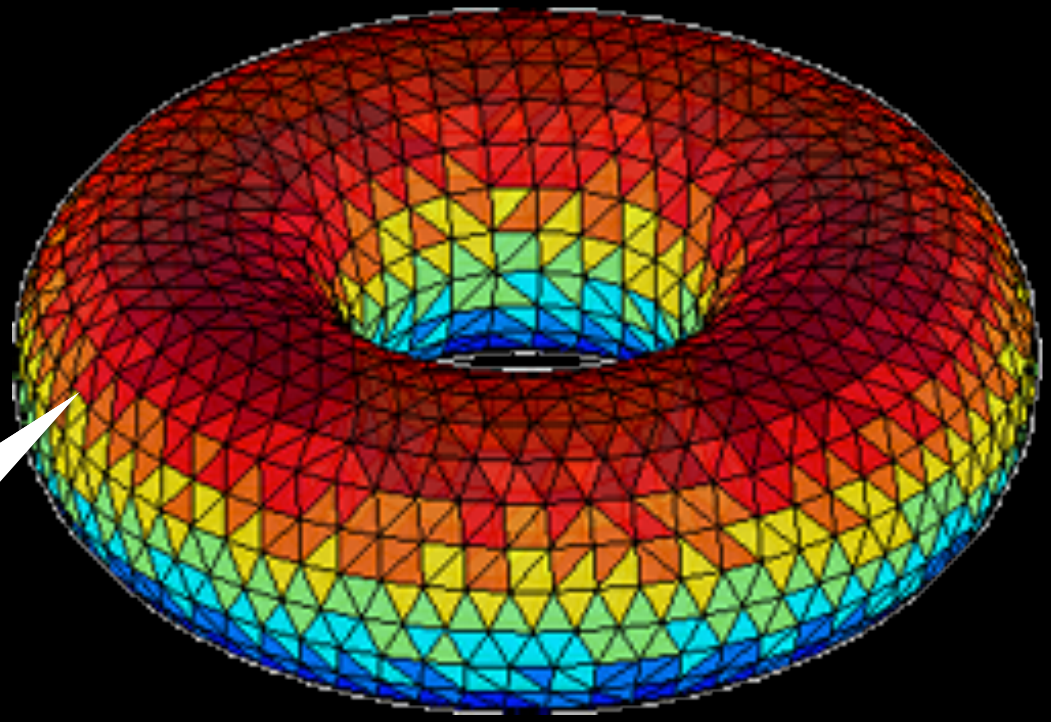
Topology

Placing together all these views yields a simplicial complex



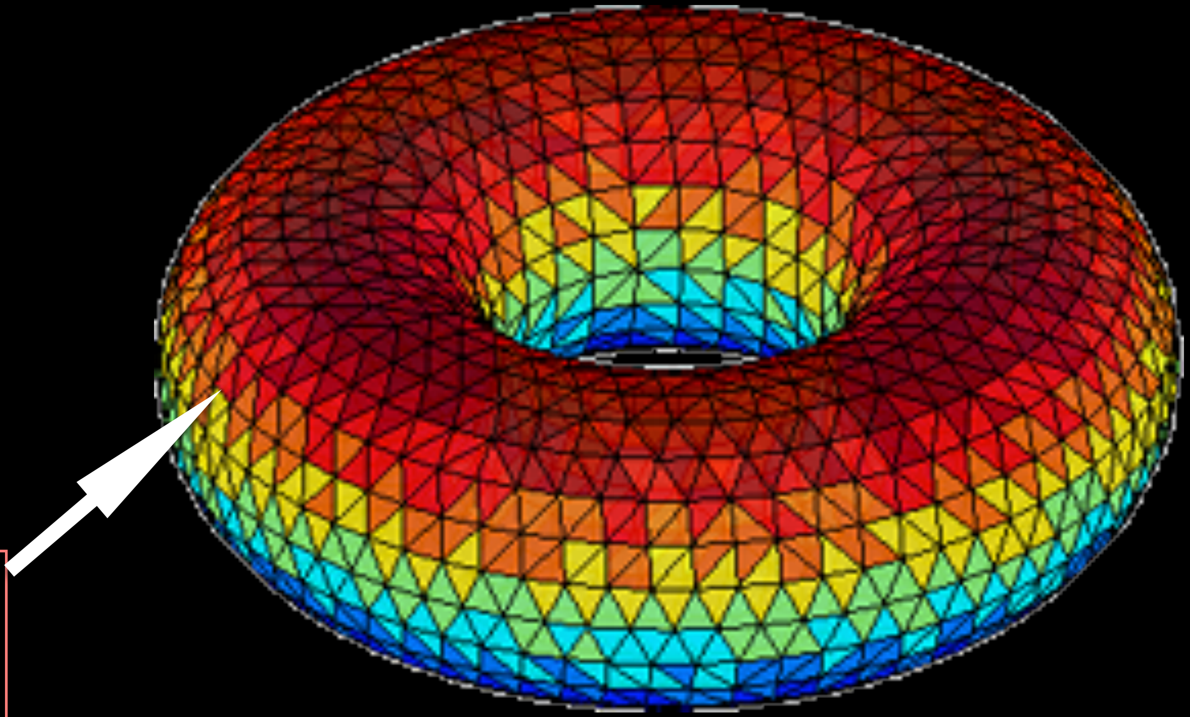
"Frozen" representation all possible interleavings and failure scenarios into a single, static, simplicial complex

Topology



Each simplex is an interleaving

Topology



Each simplex is
an interleaving

views label vertices
of a simplex

Topological invariants

Topological invariants

- Preserved as computation unfolds

Topological invariants

- Preserved as computation unfolds
- Come from the nature of the faults and asynchrony in the system

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

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

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

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

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

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

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:

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

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

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

Short History

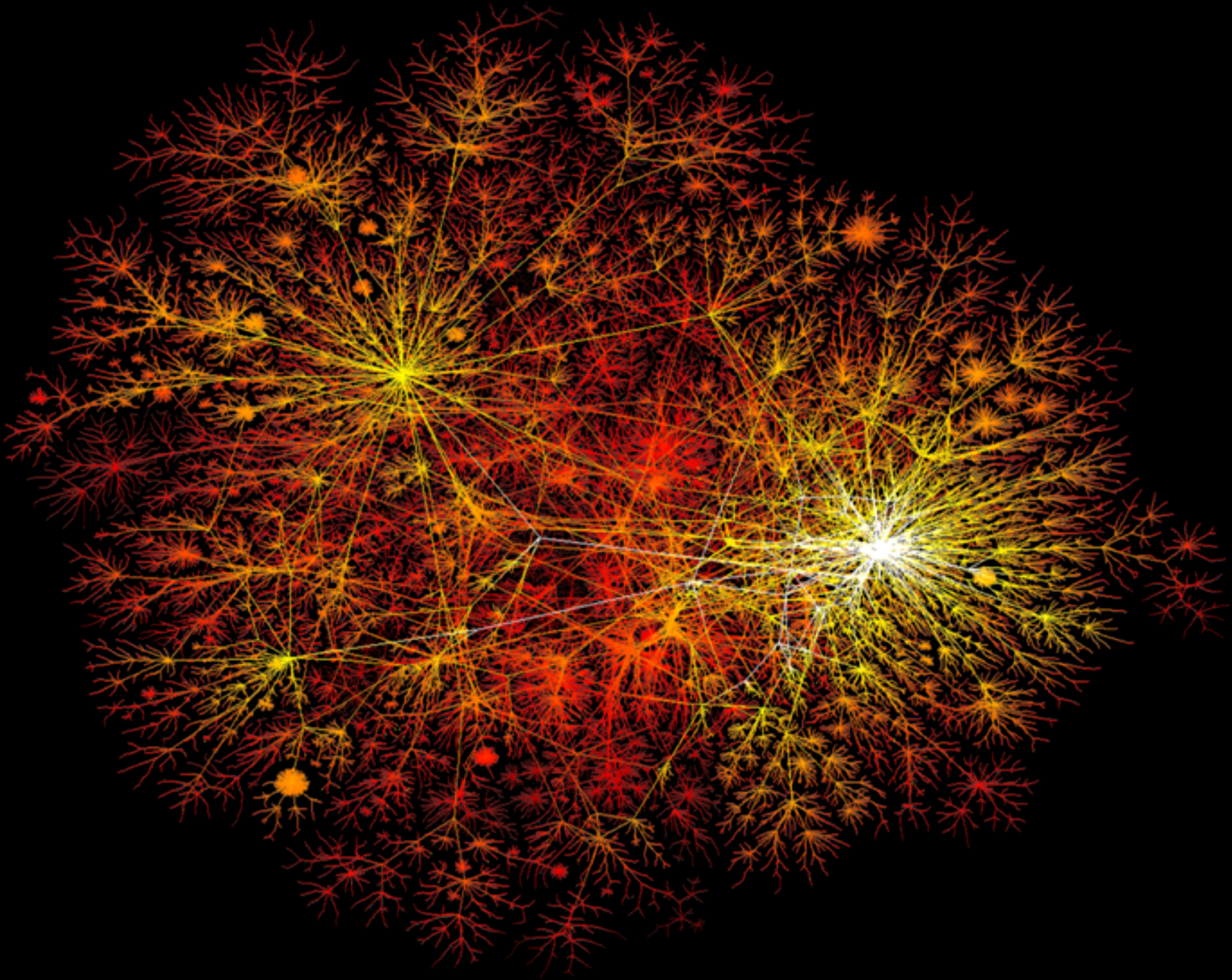
Discovered in PODC 1988 when only 1 process may crash (dimension=1) by Biran, Moran and Zaks, after consensus FLP impossibility of PODS 1983

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

Theory of distributed computing research

- Models of distributed computing systems:

Theory of distributed computing research

- Models of distributed computing systems: communication, timing, failures, which models are central?

Theory of distributed computing research

- Models of distributed computing systems: communication, timing, failures, which models are central?
- Distributed Problems:

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,…

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

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:

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

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

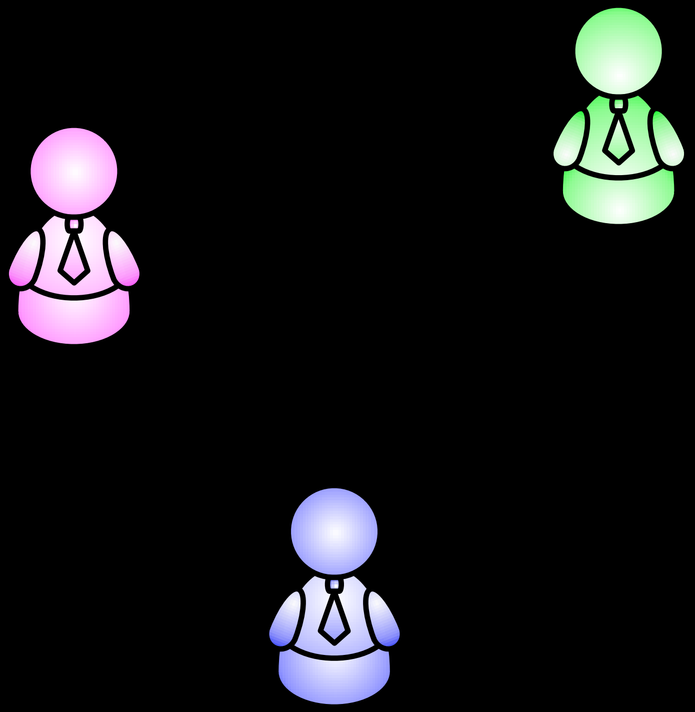
(a Turing Machine for DC)

Ingredients of a model

- processes
- communication
- failures

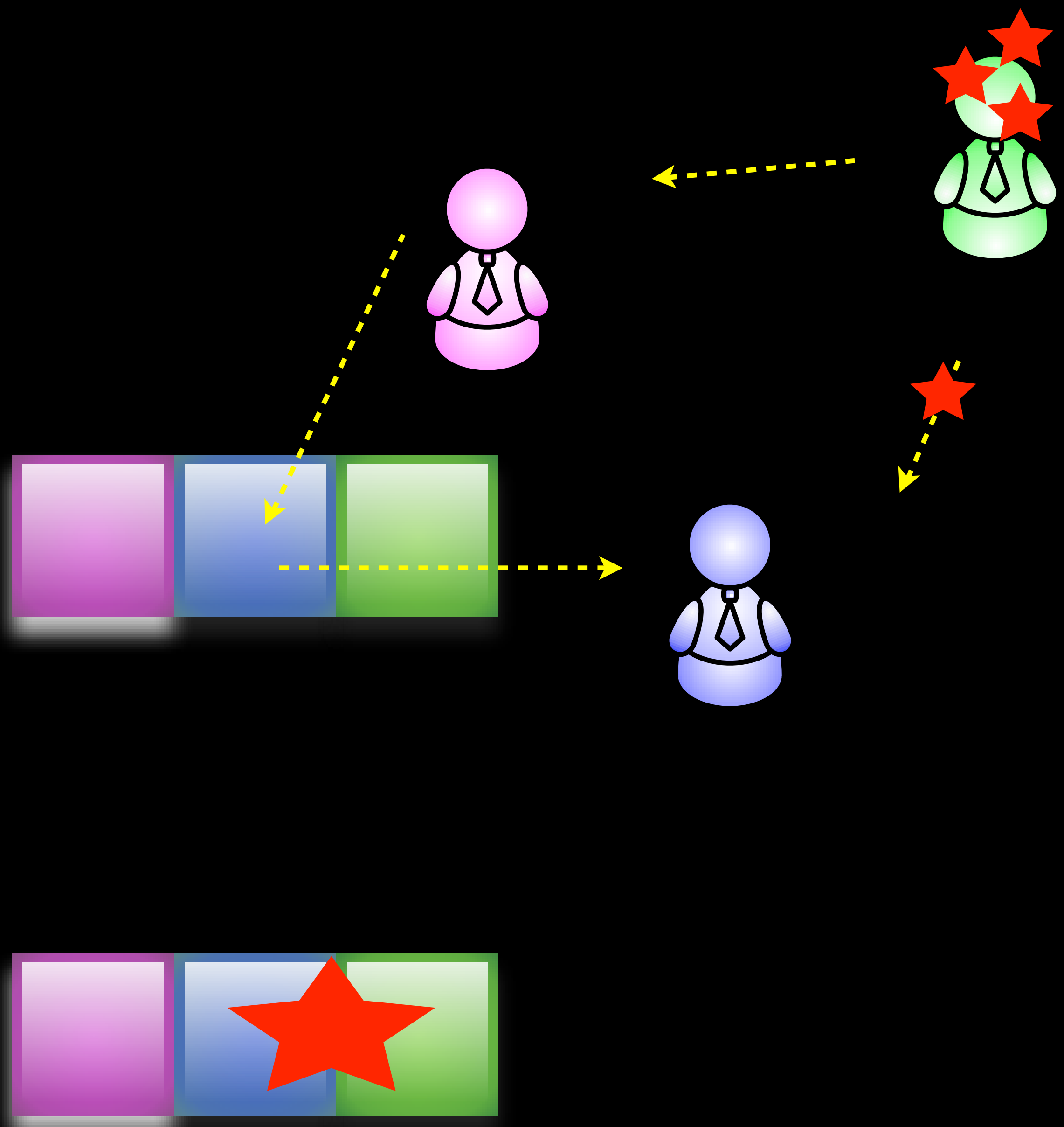
Ingredients of a model

- processes
- communication
- failures



Ingredients of a model

- processes
- communication
- failures



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

single-reader/single-writer

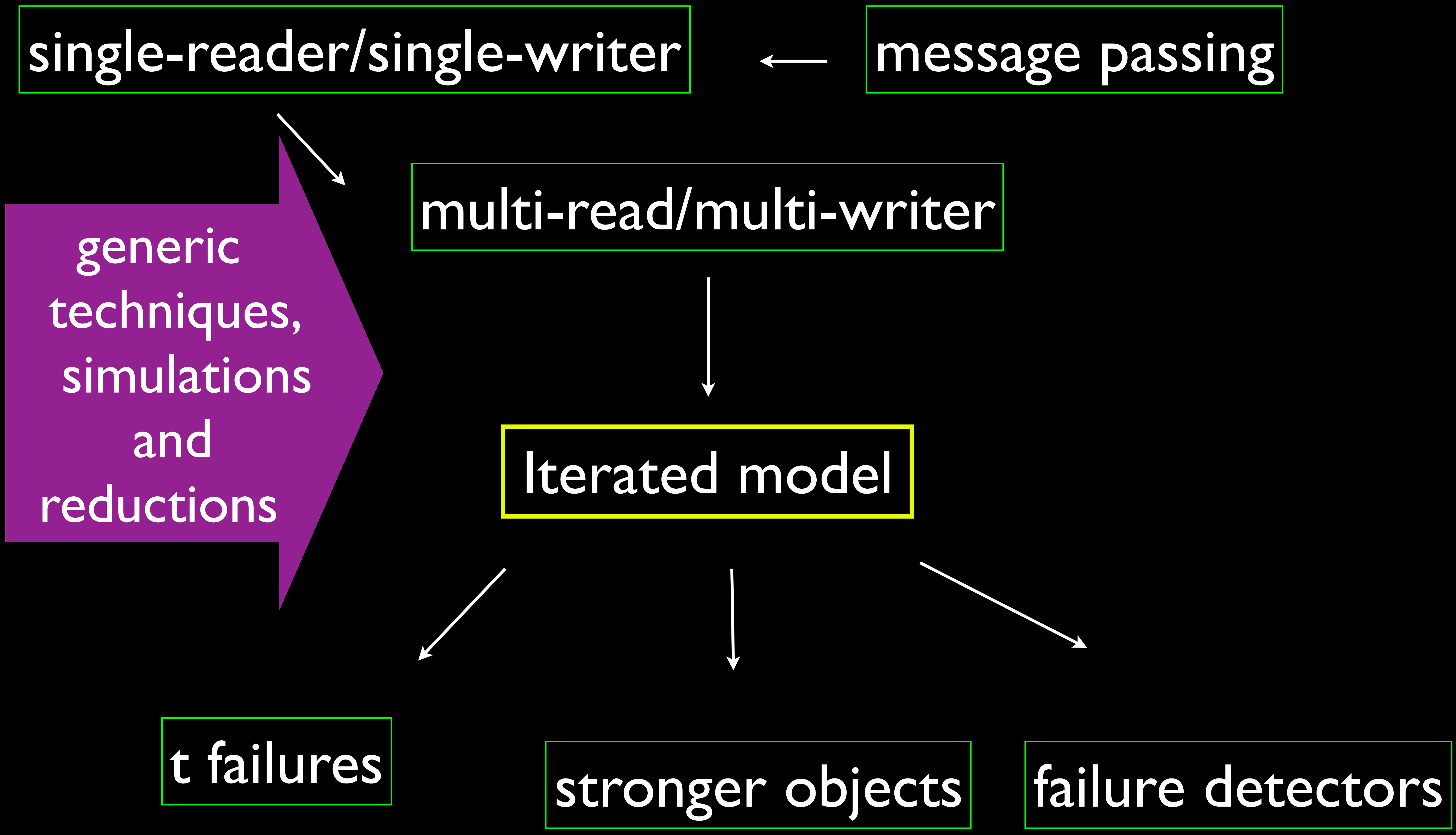
message passing

multi-read/multi-writer

t failures

stronger objects

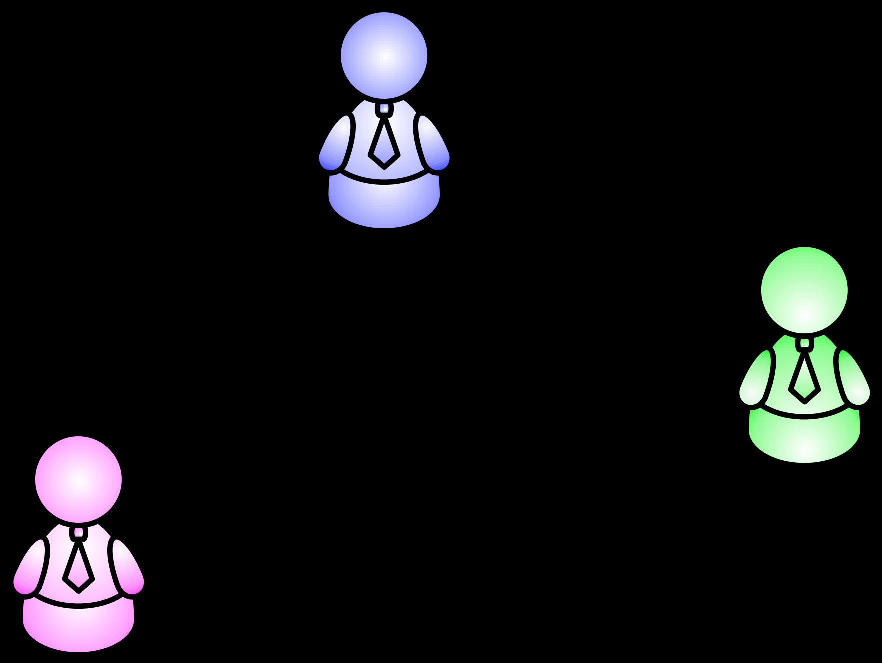
failure detectors


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

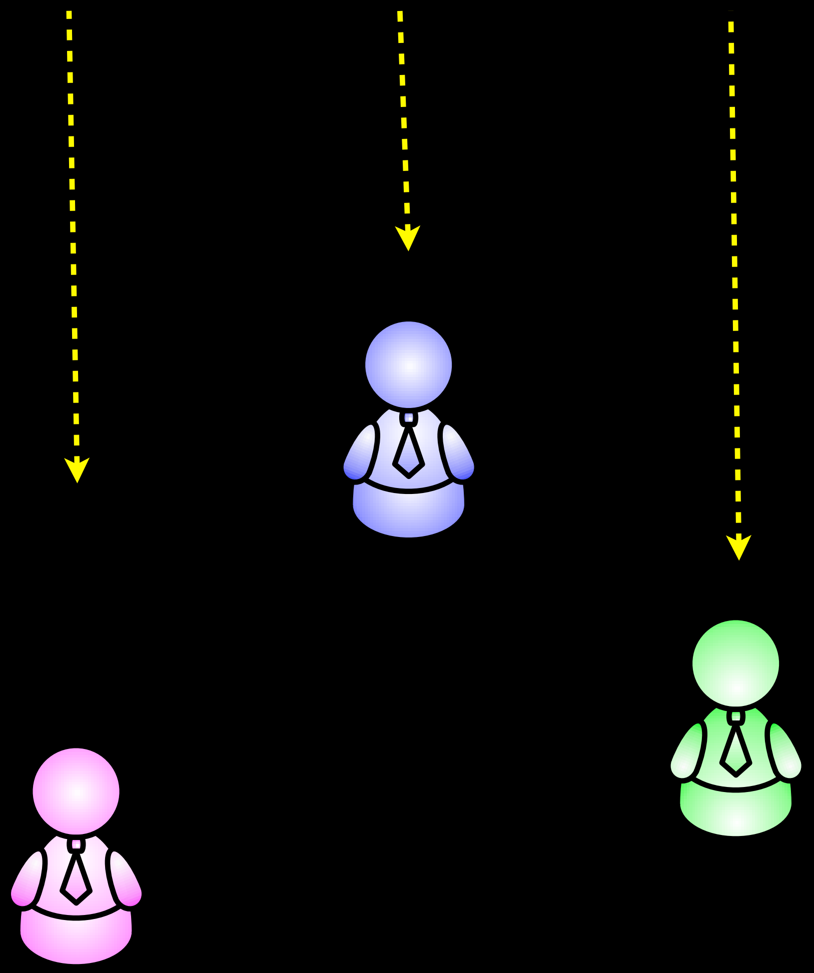
Iterated shared memory

(a Turing Machine for DC ?)

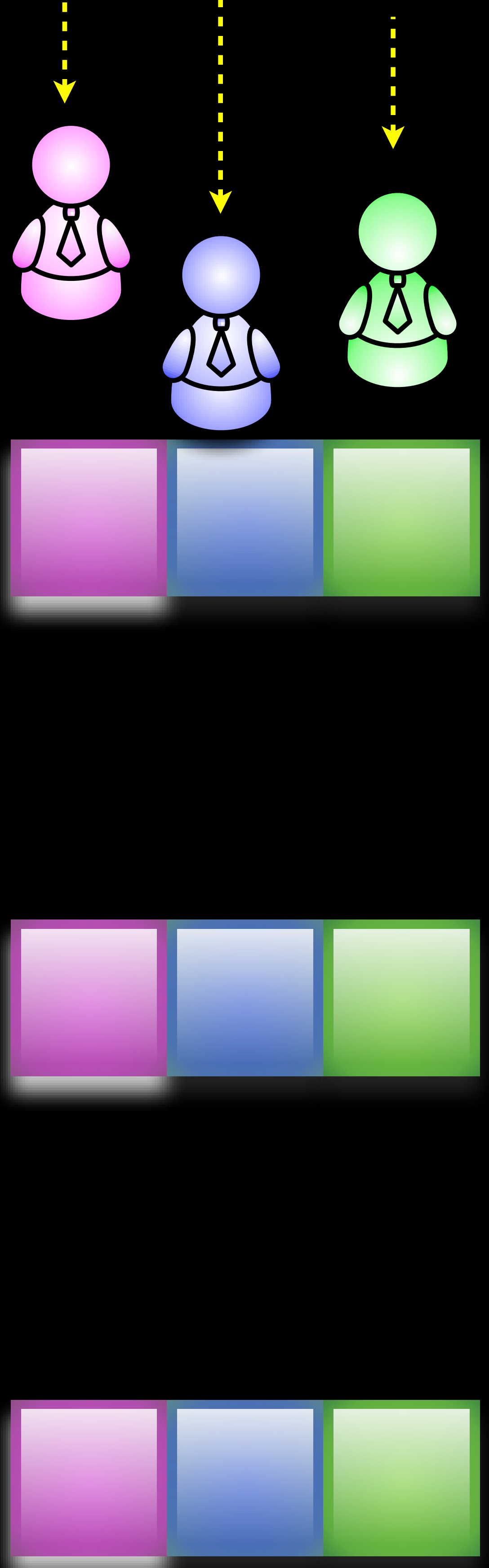
n Processes



asynchronous, wait-free

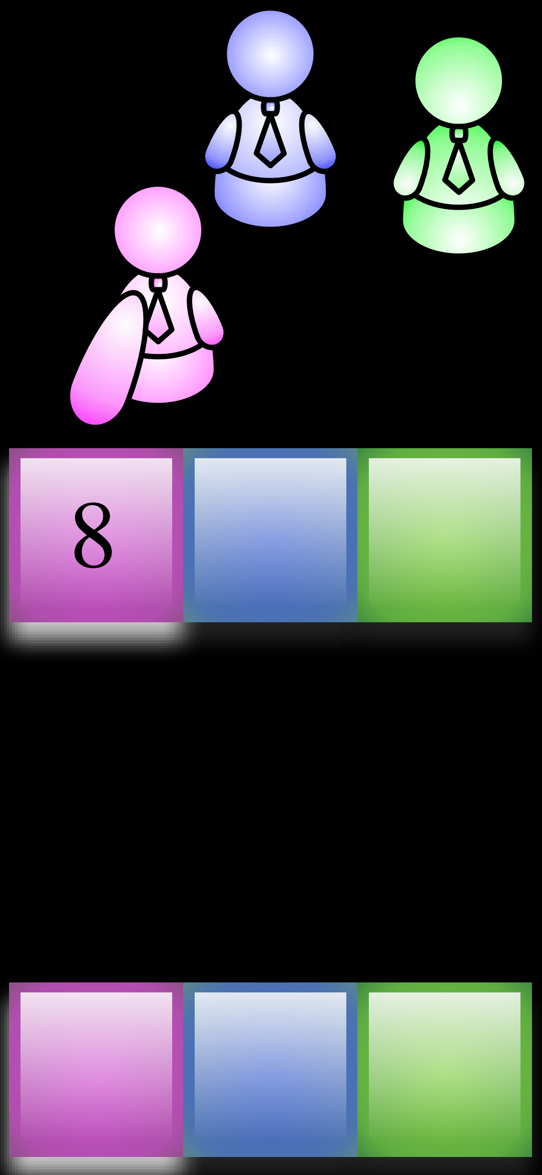


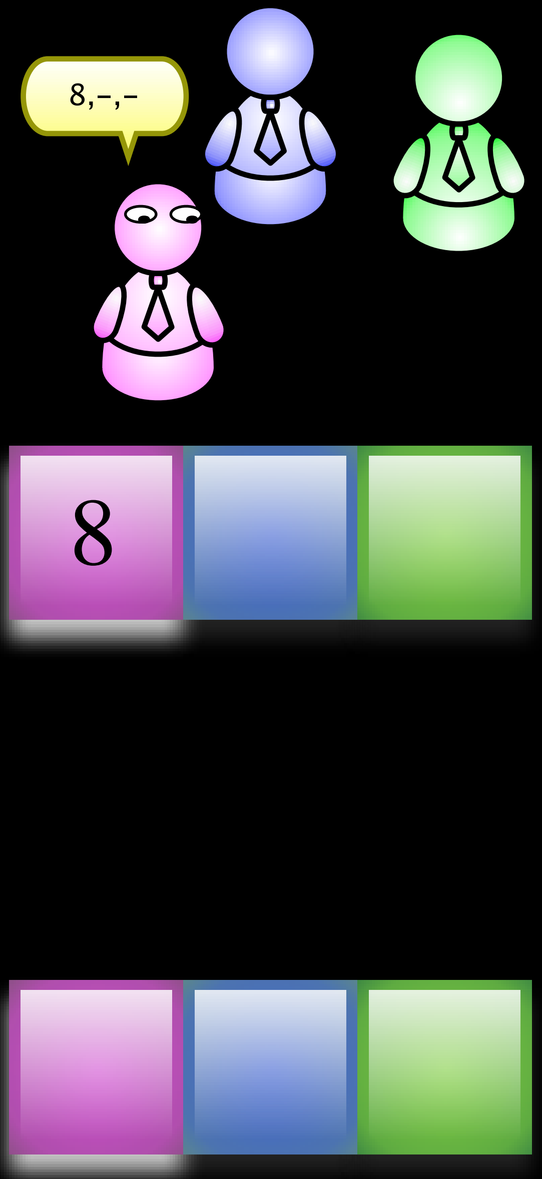
Unbounded sequence of read/write shared arrays

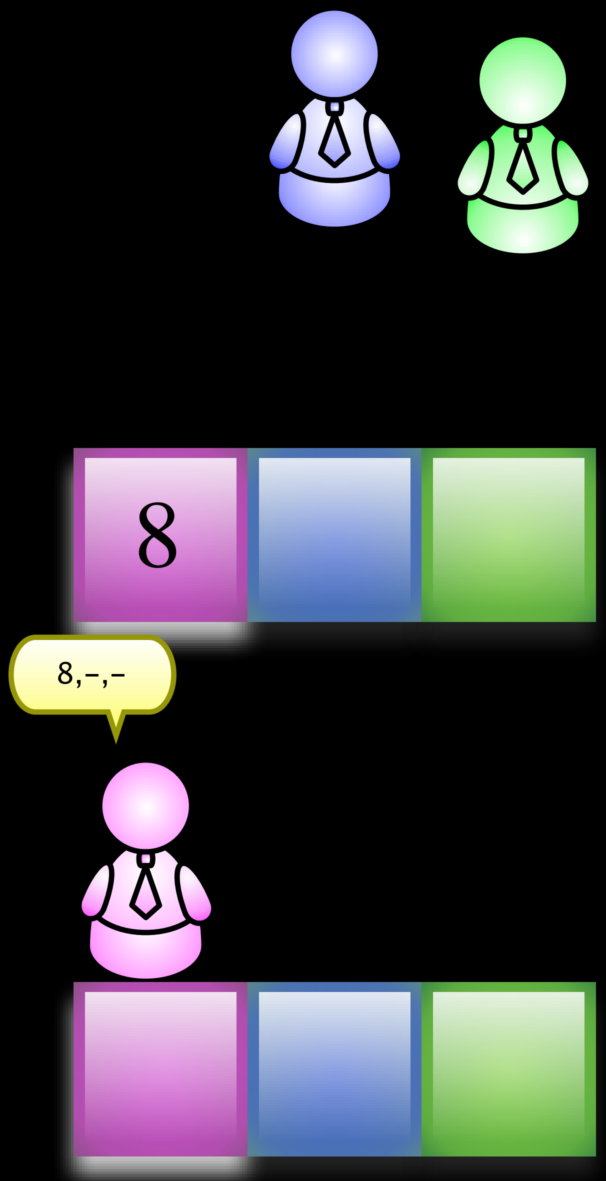


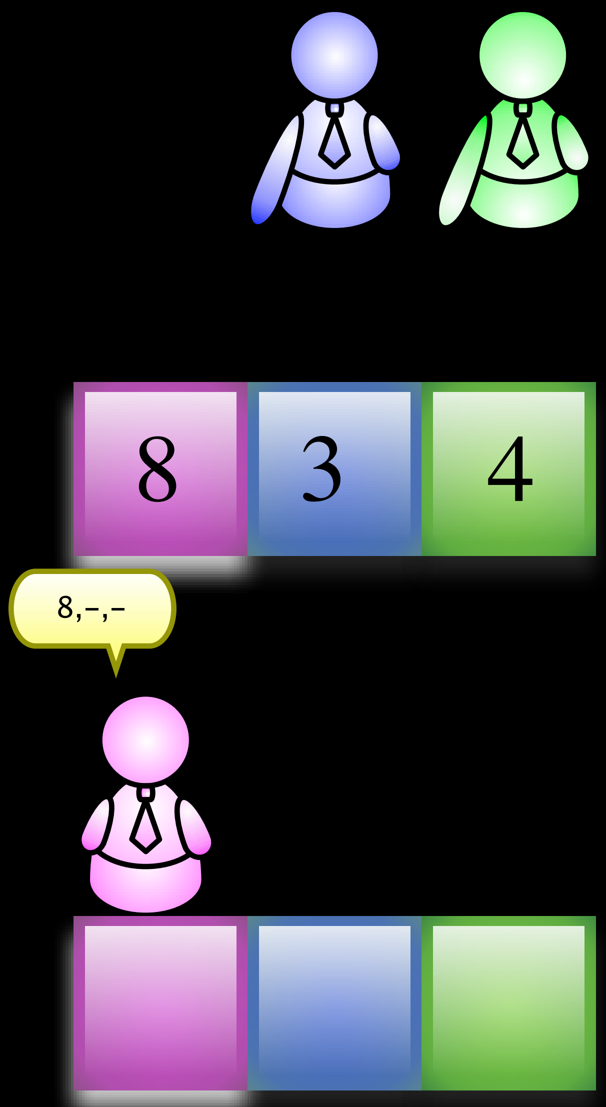
- use each one once
- in order

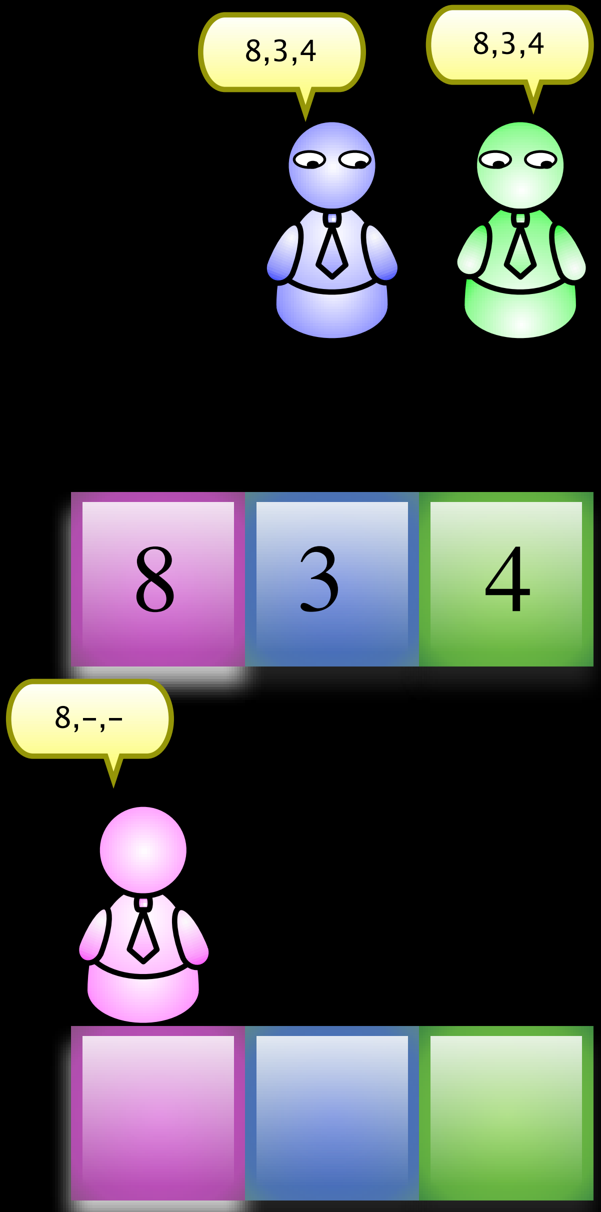
write, then read

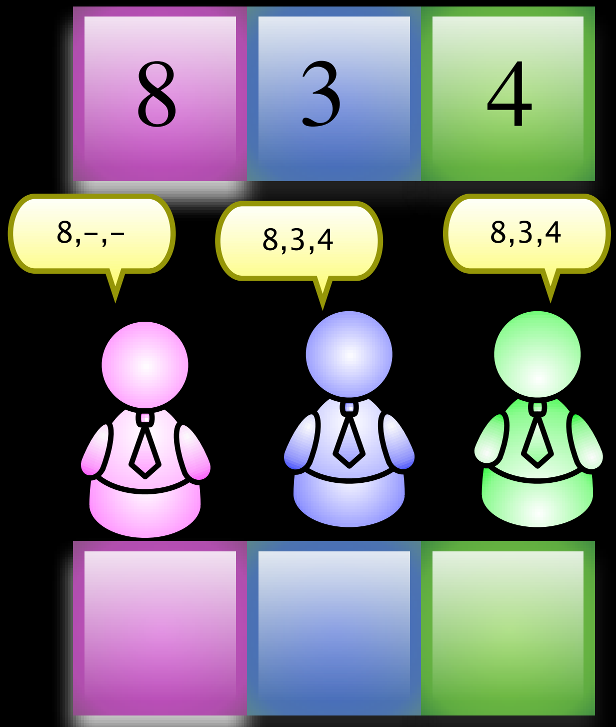



8,–,–
8

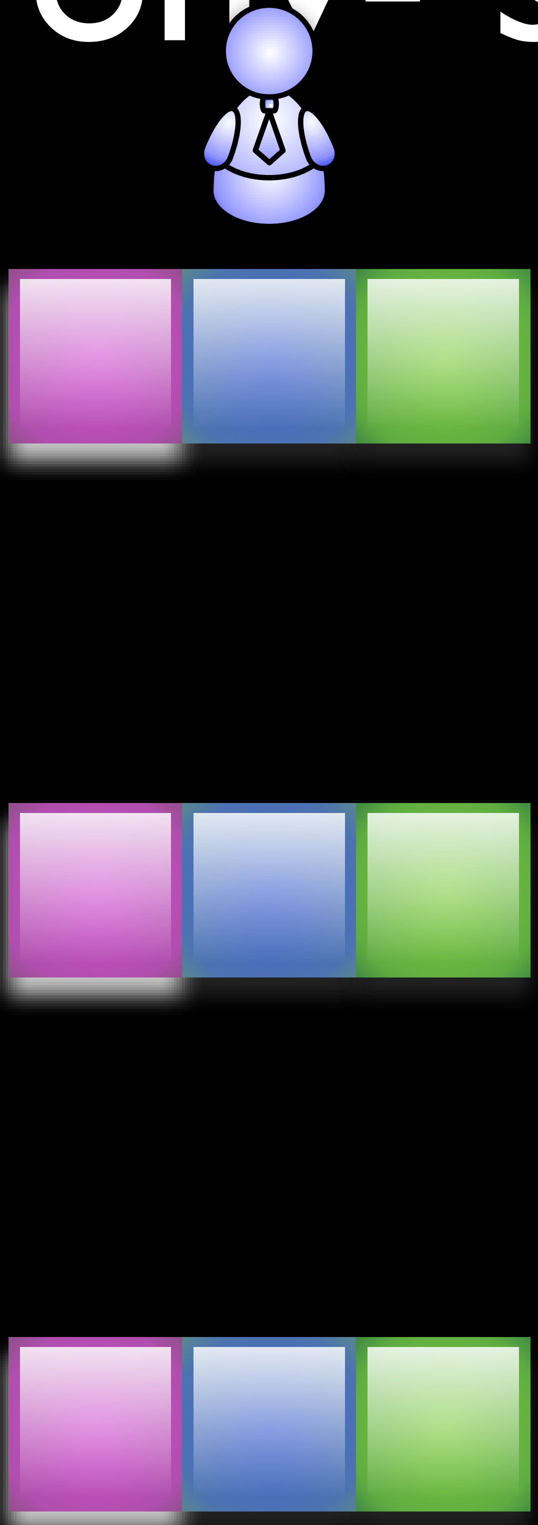

8
8,–,–


8
3
4
8,–,–

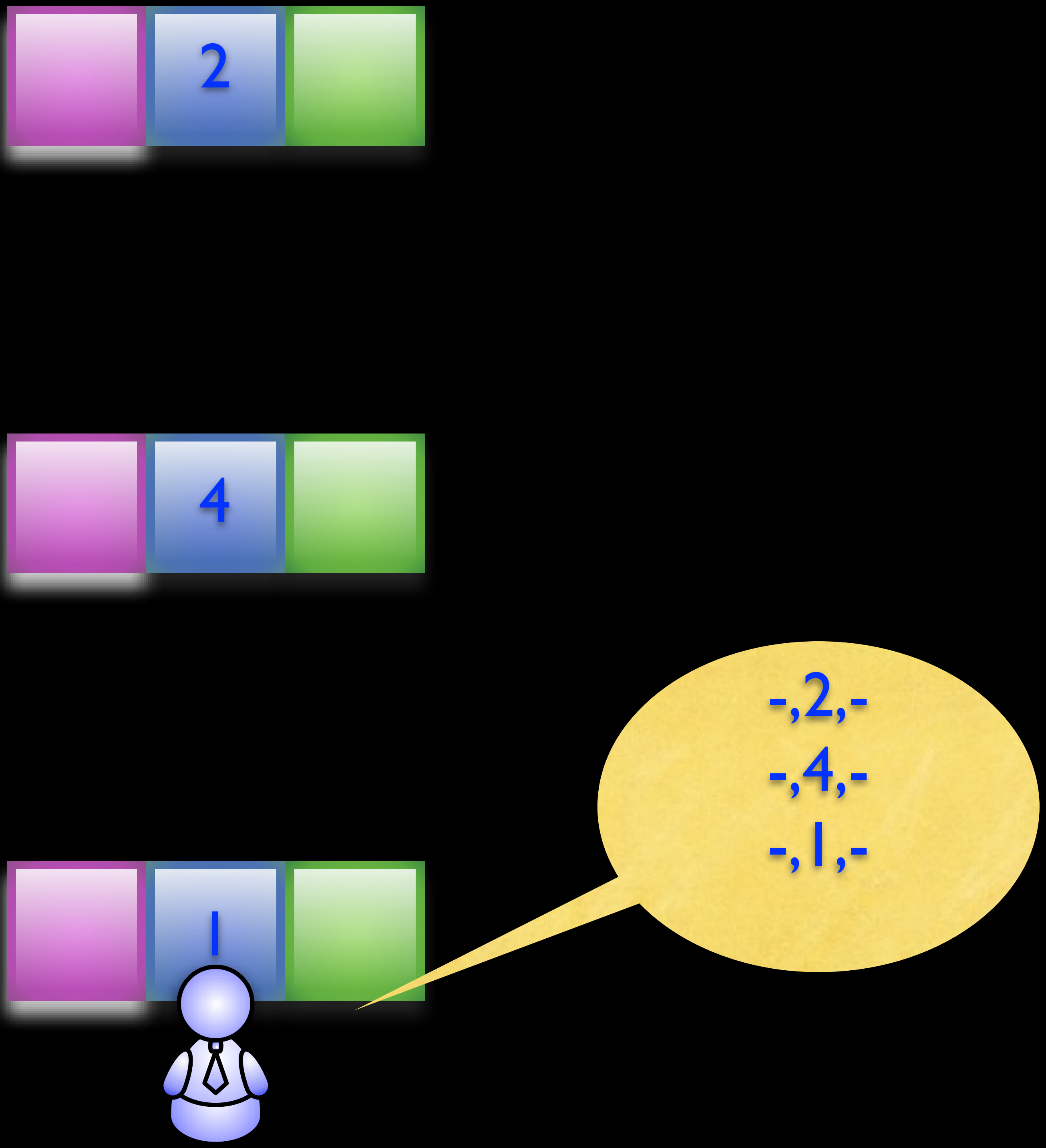

8,3,4
8,3,4
8
3
4
8,–,–


8
3
4
8,–,–
8,3,4
8,3,4

Asynchrony- solo run

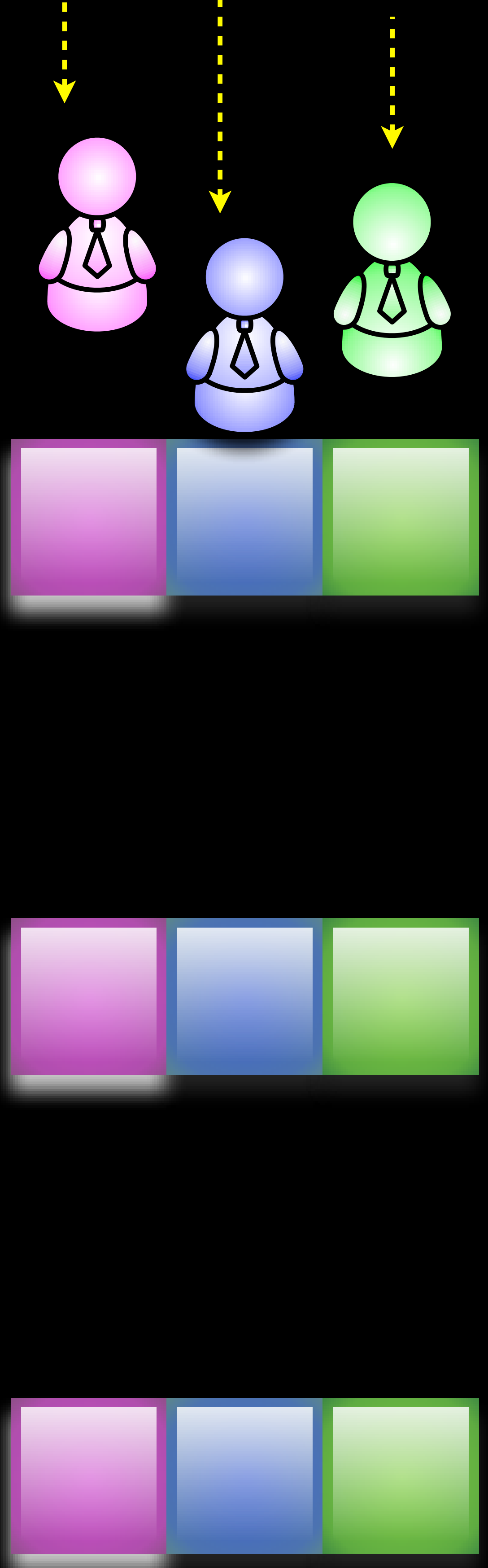


Asynchrony- solo run

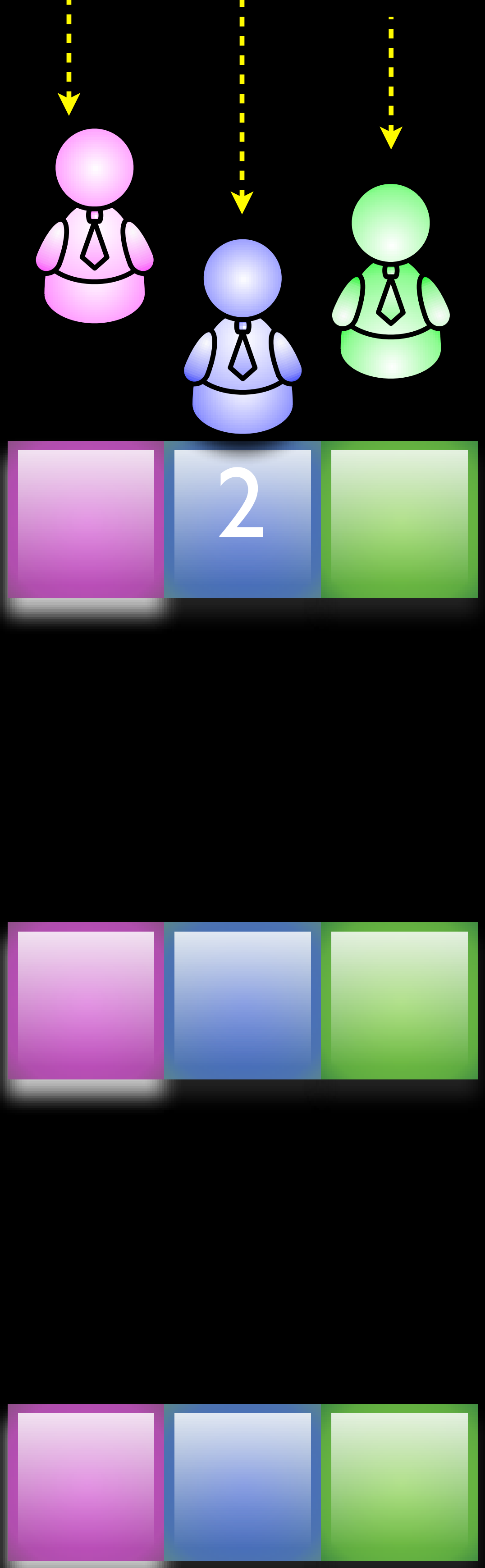


every copy is new

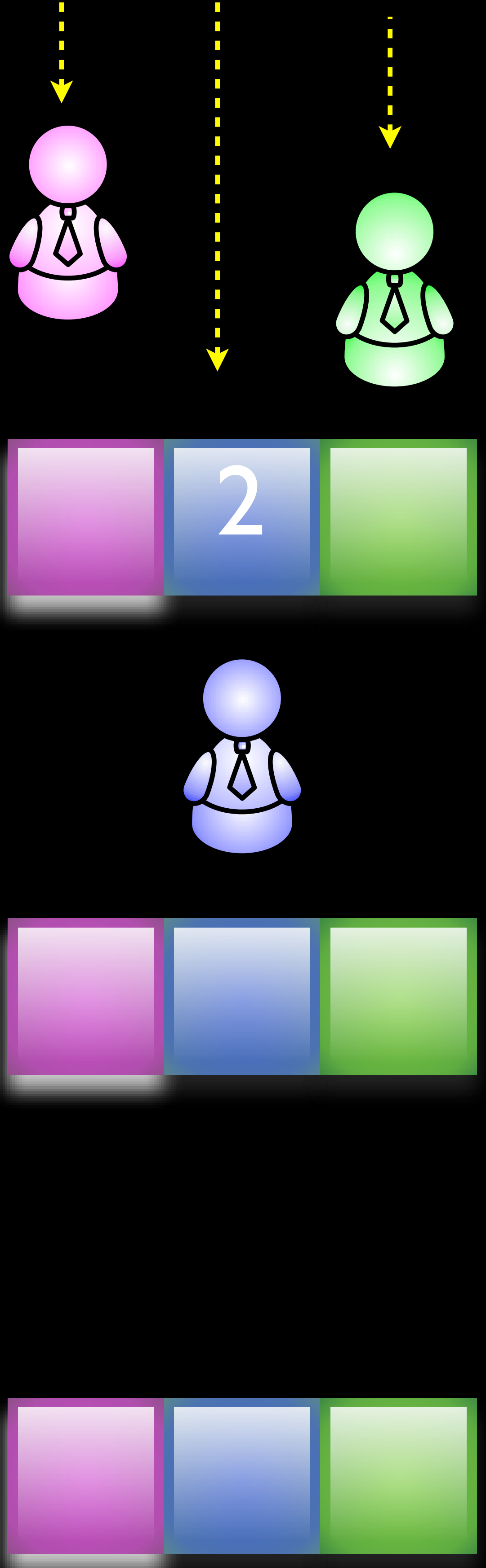
- arrive in arbitrary order
- last one sees all



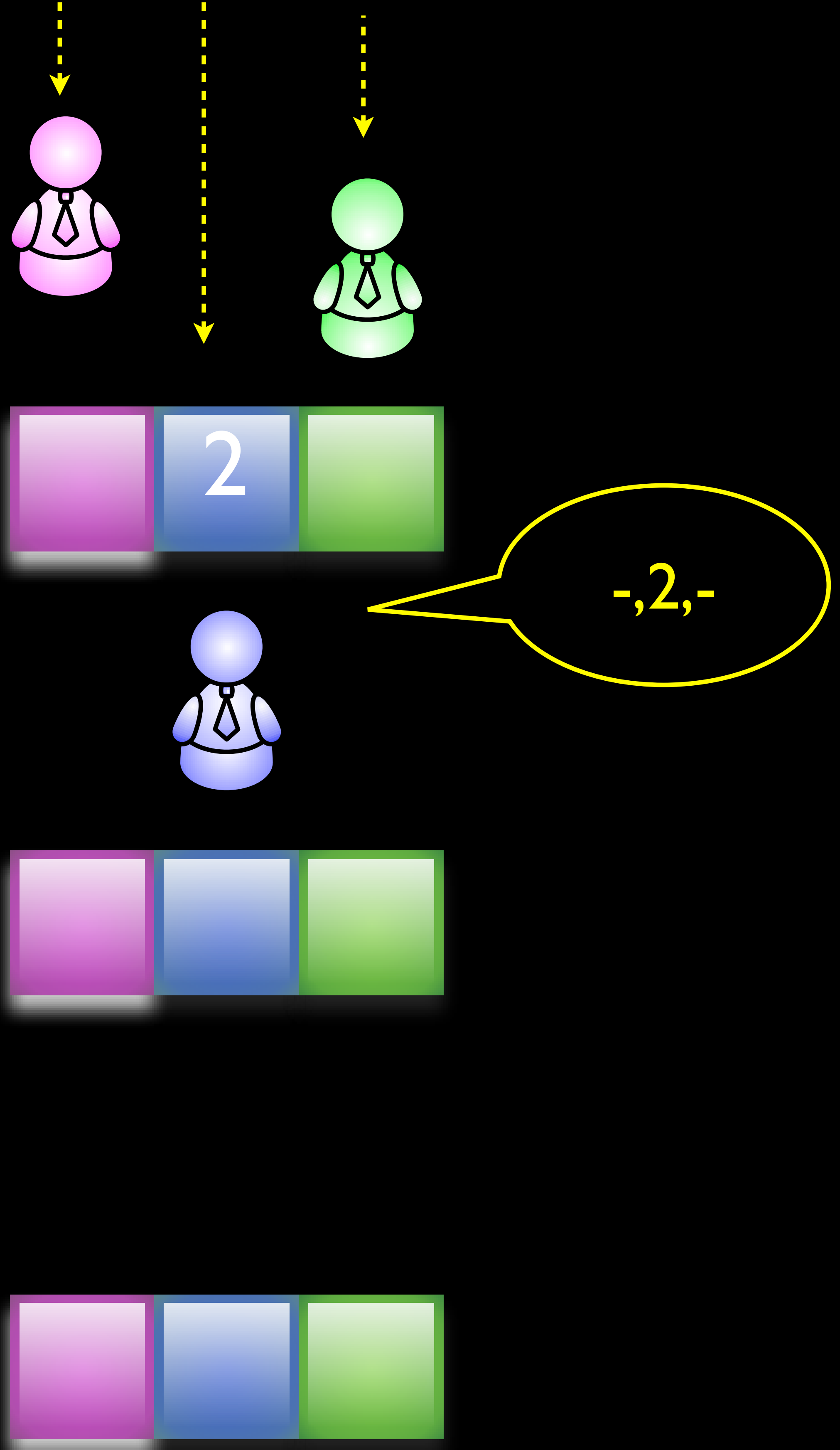
- arrive in arbitrary order
- last one sees all



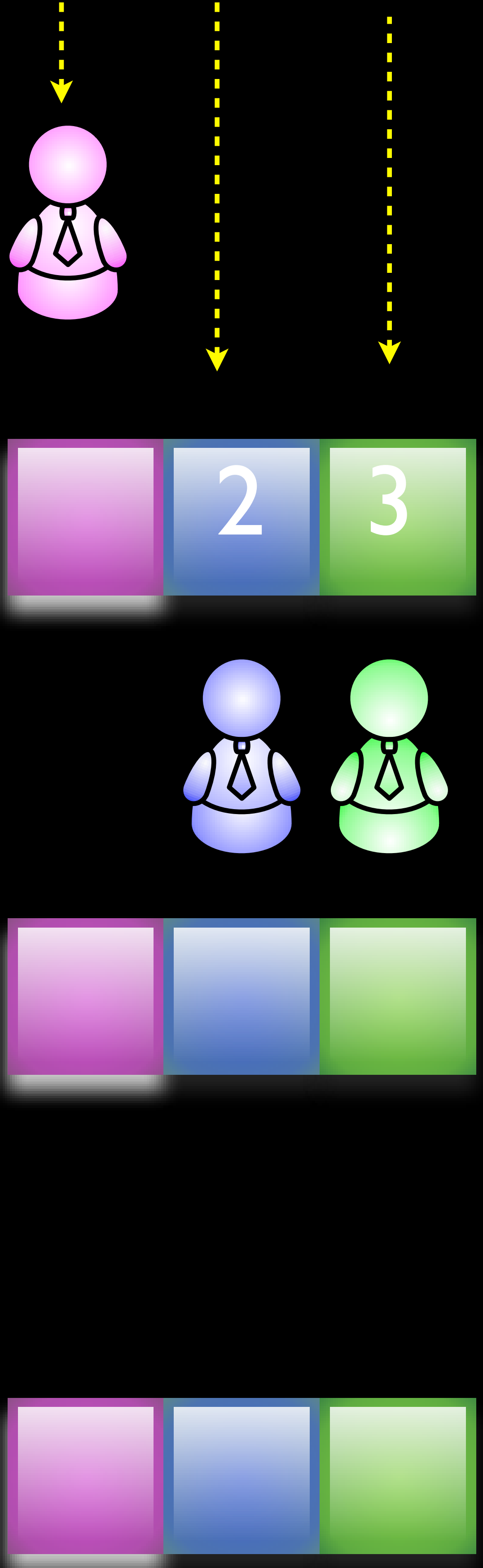
- arrive in arbitrary order
- last one sees all



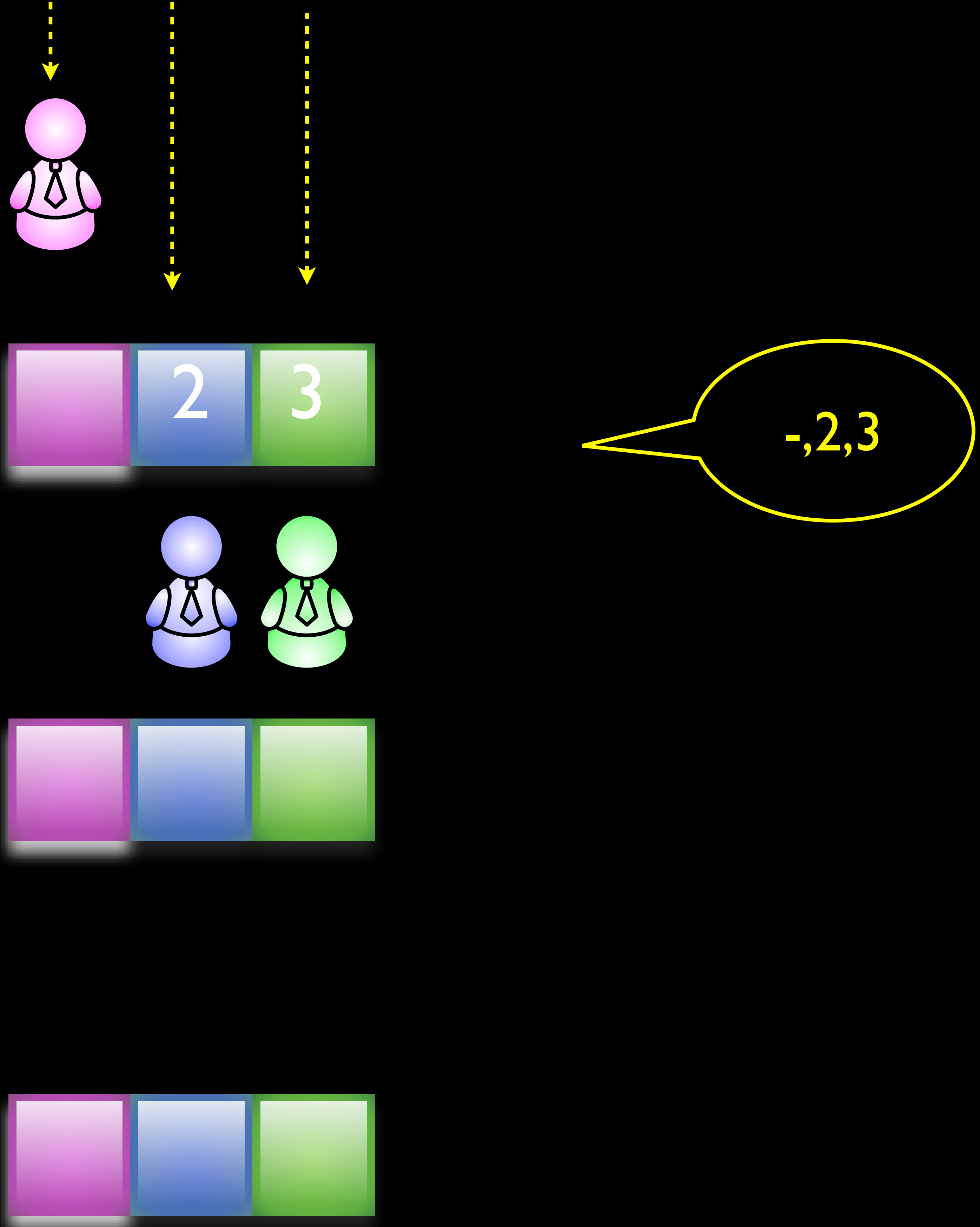
- arrive in arbitrary order
- last one sees all



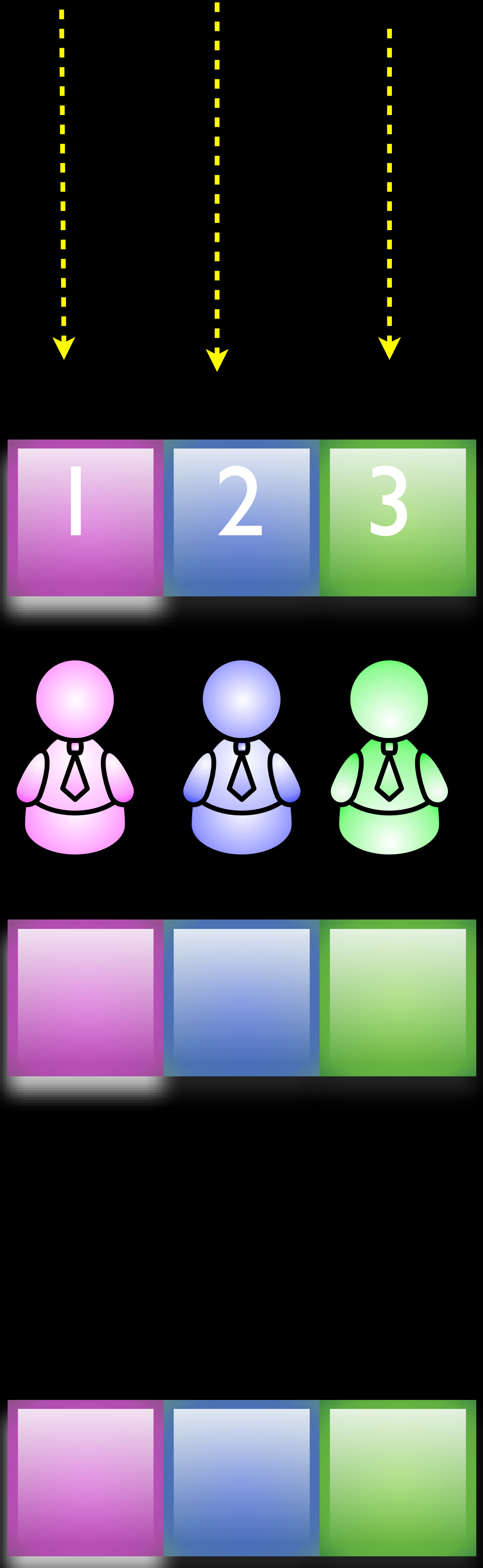
- arrive in arbitrary order
- last one sees all



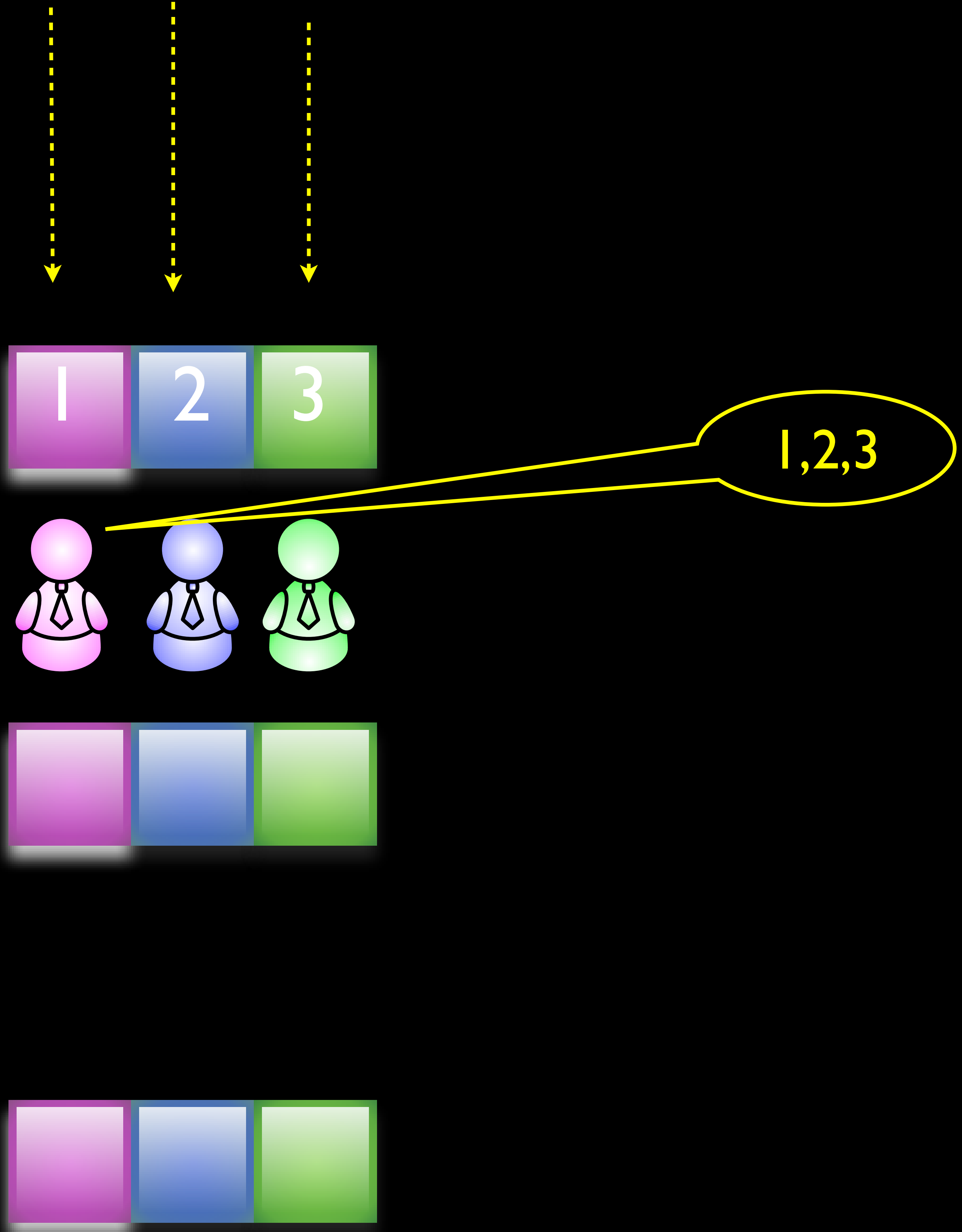
- arrive in arbitrary order
- last one sees all



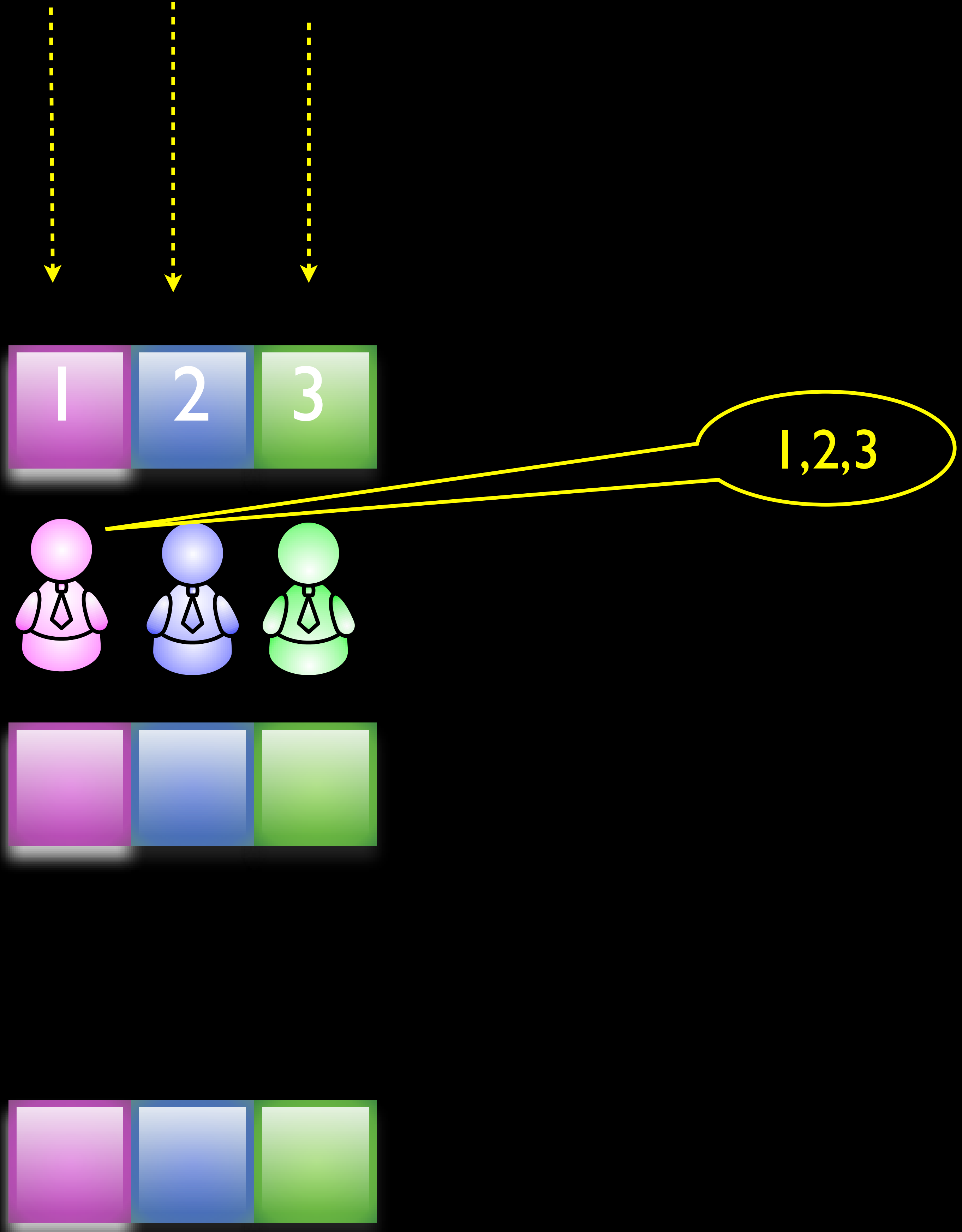
- arrive in arbitrary order
- last one sees all



- arrive in arbitrary order
- last one sees all

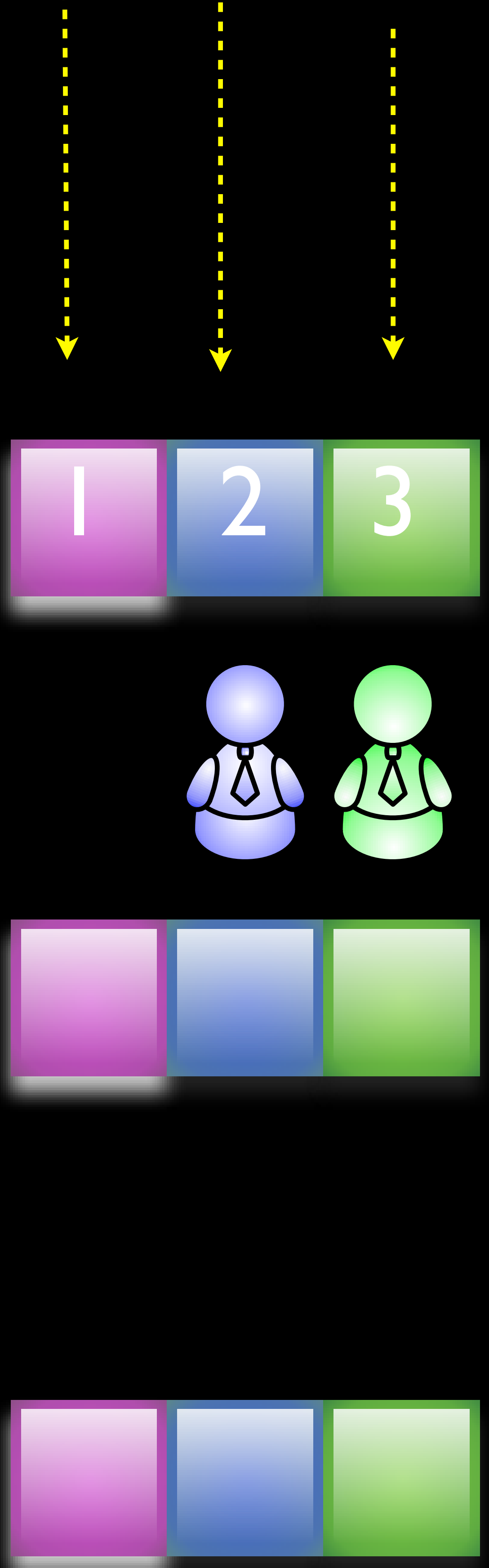


- arrive in arbitrary order
- last one sees all

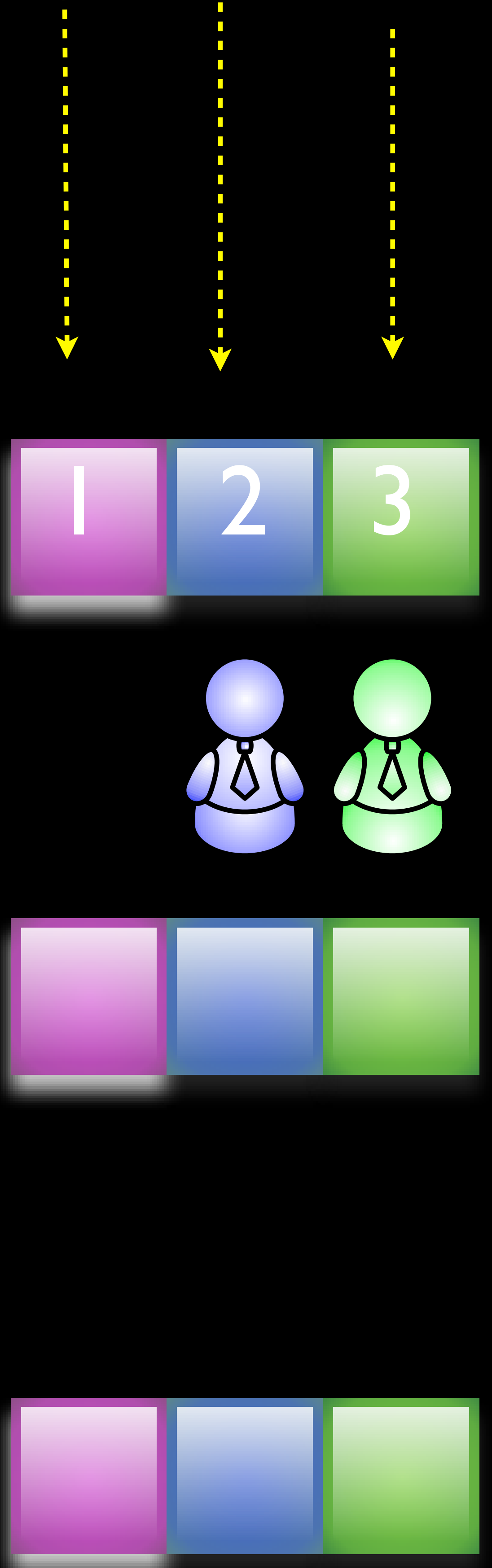


- arrive in arbitrary order
- last one sees all

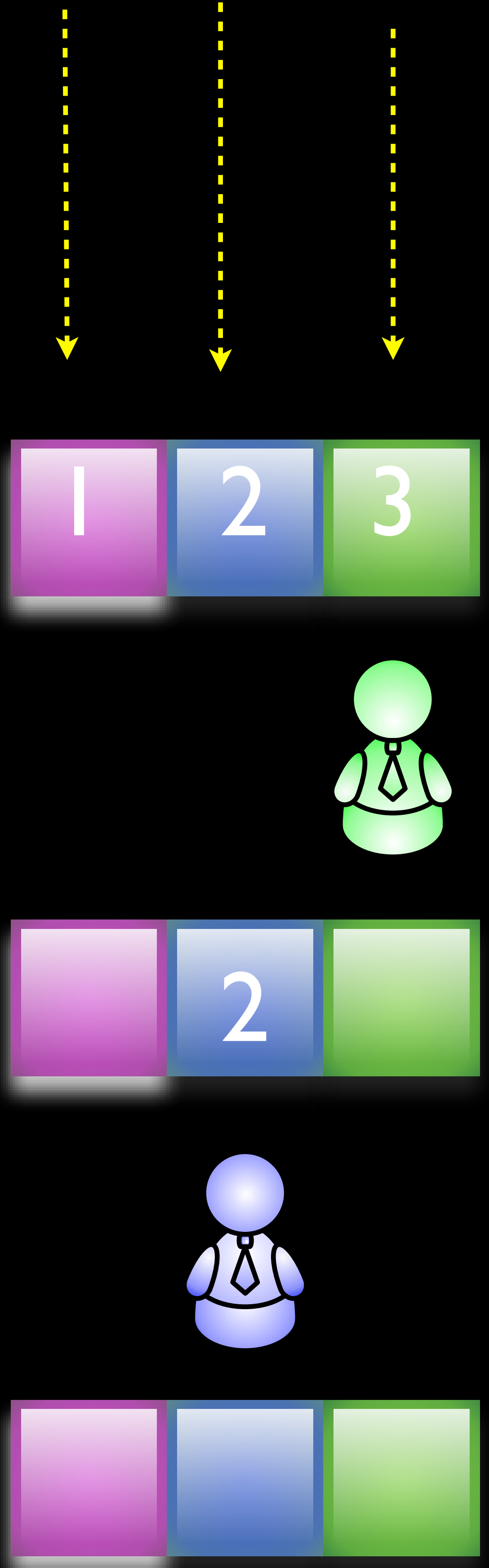
returns 1,2,3



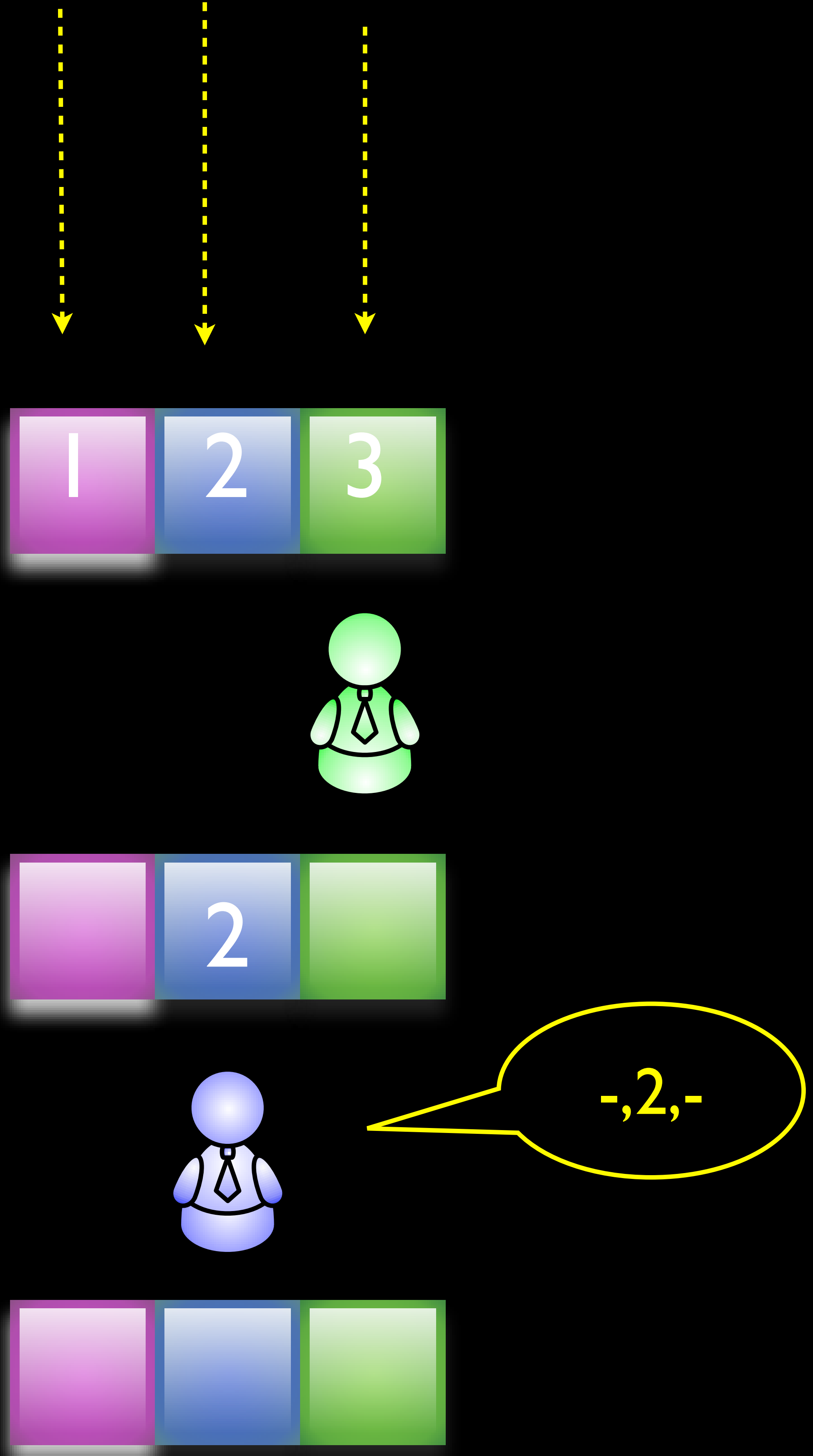
- remaining 2 go to next memory



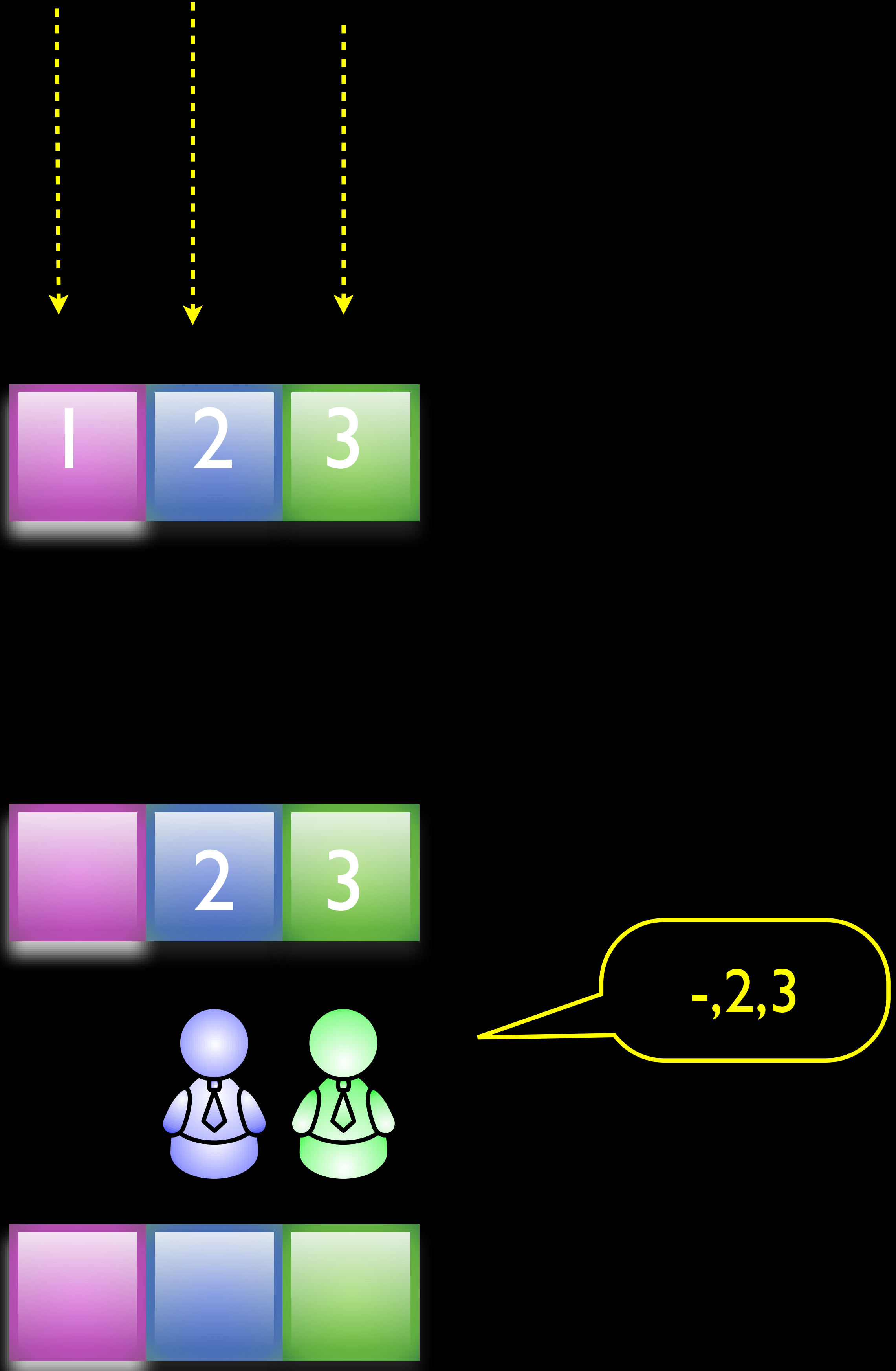
- remaining 2 go to next memory



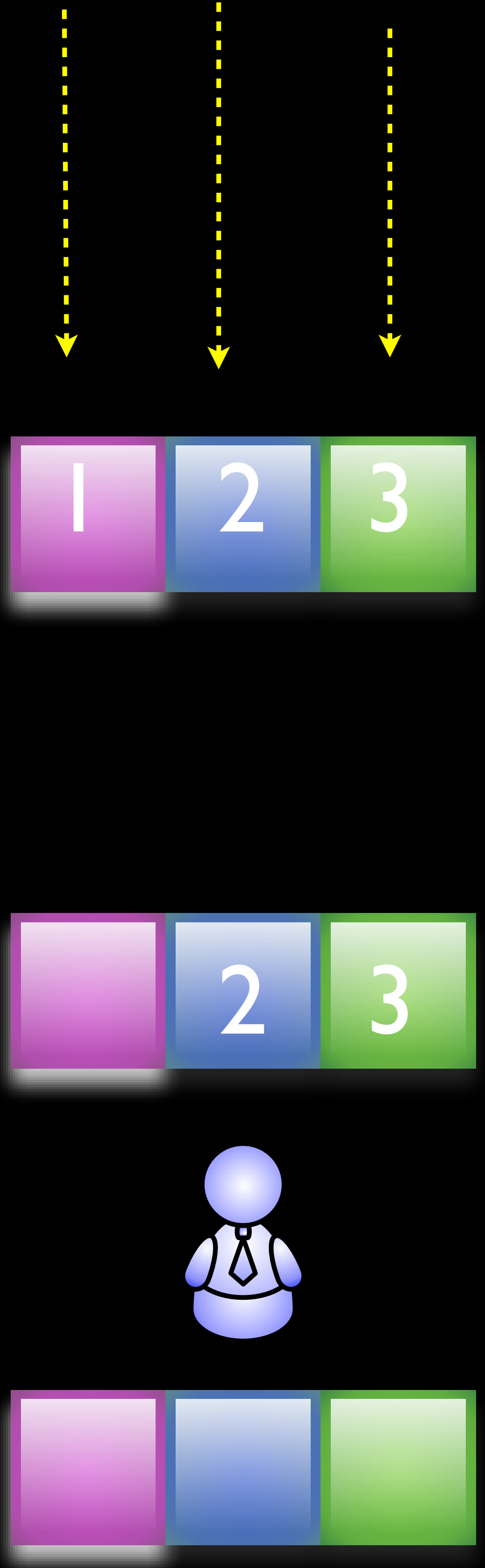
- remaining 2 go to next memory



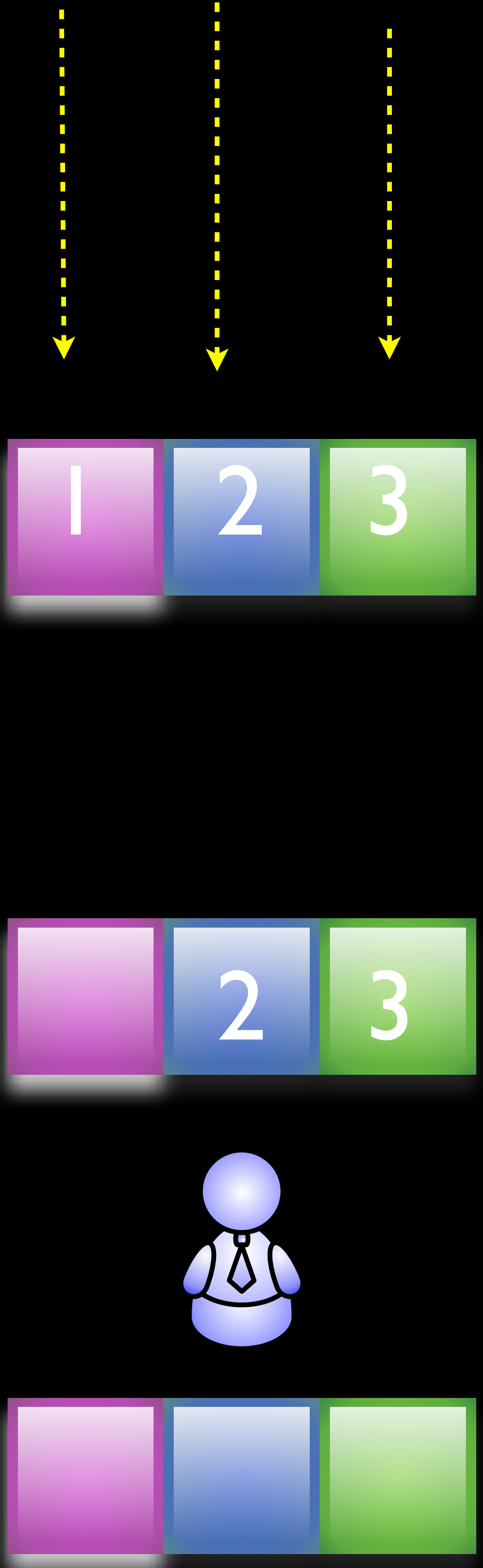
- 3rd one returns -,2,3

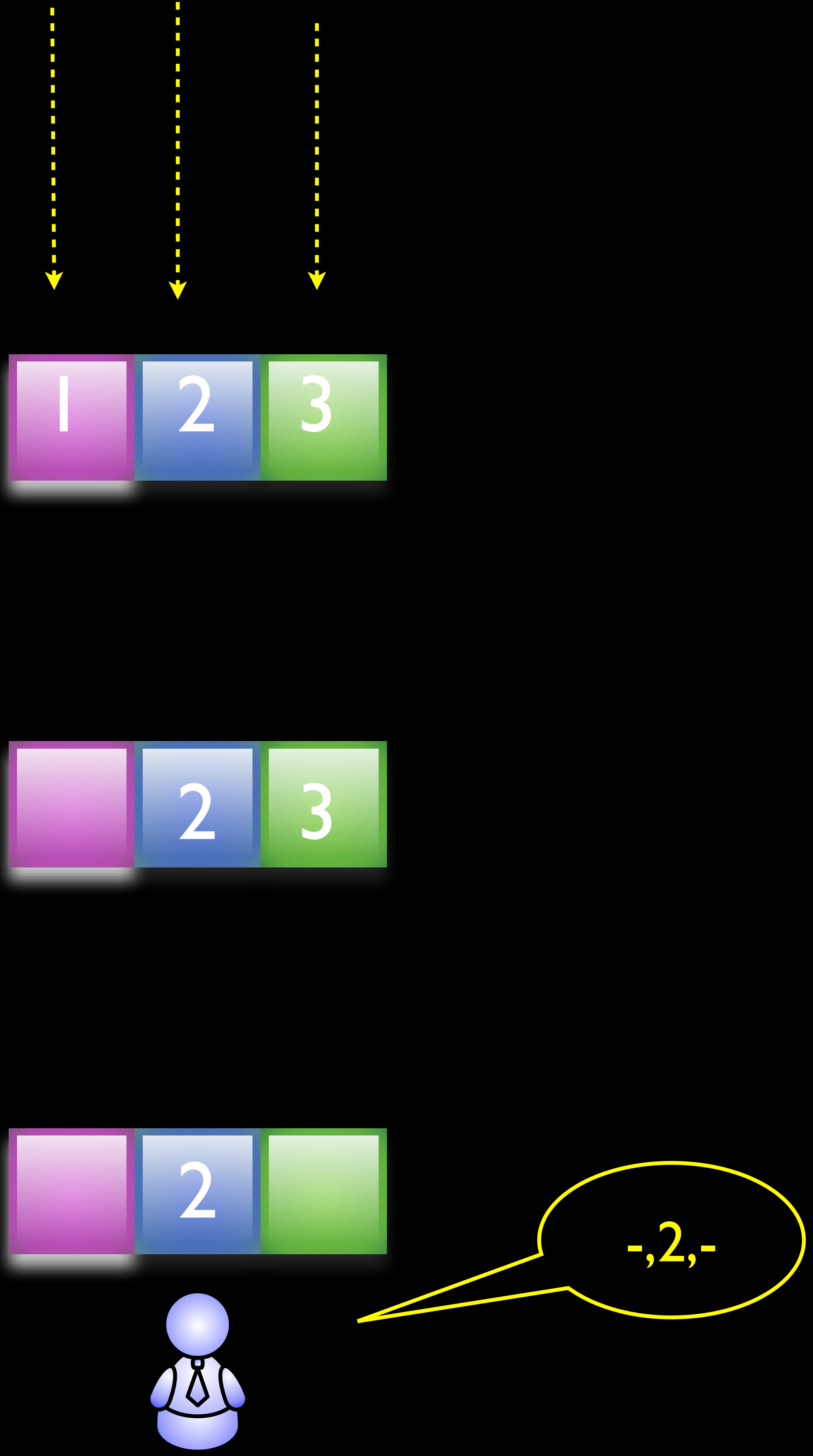


- 3rd one returns -,2,3

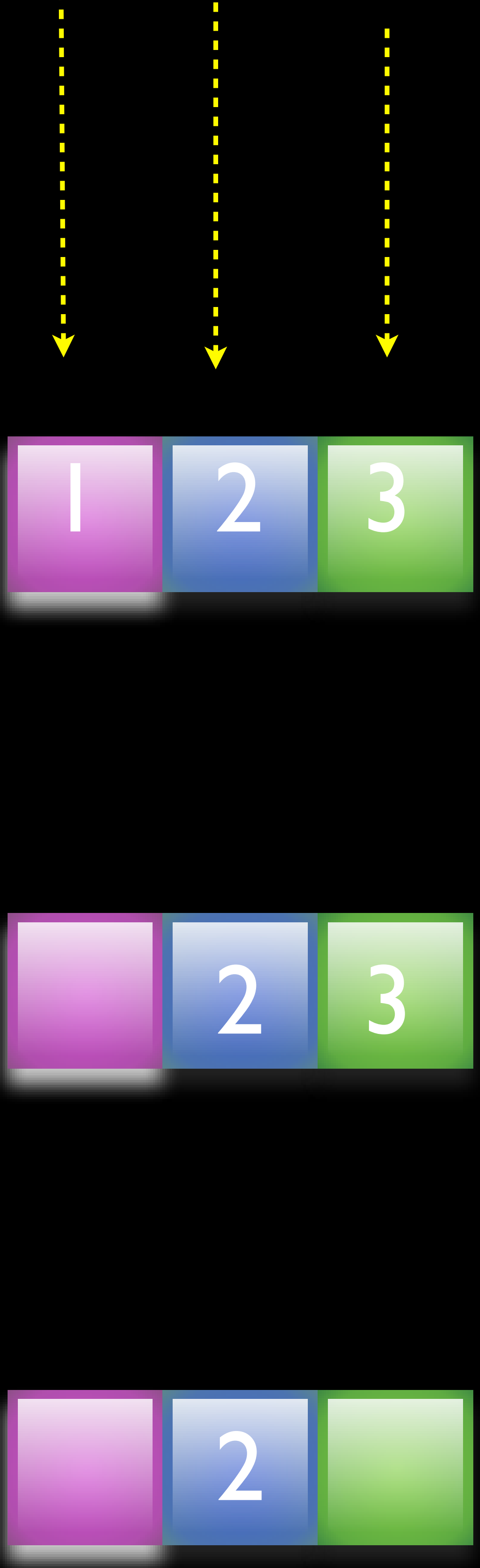


- 2nd one goes alone

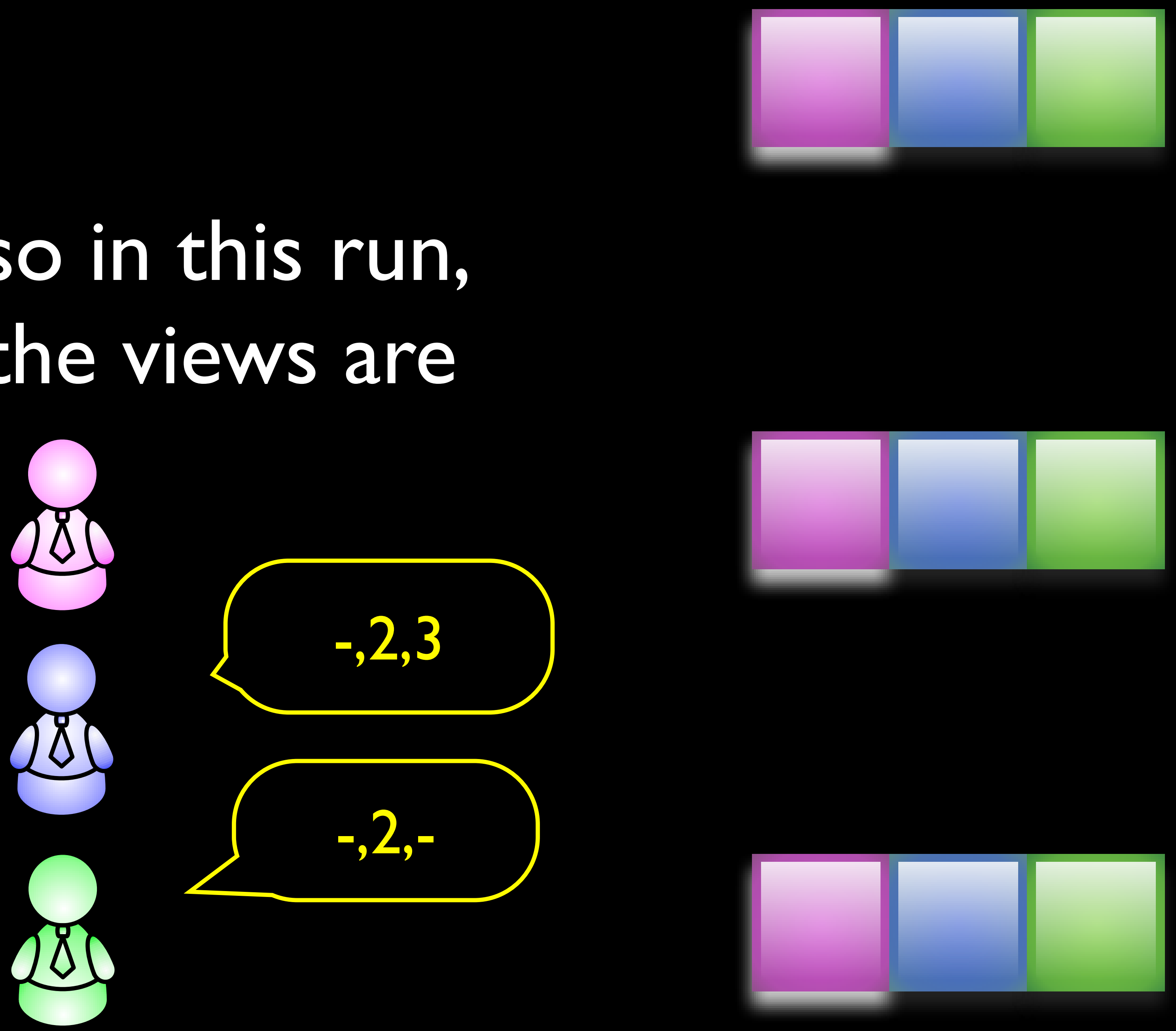



1
2
3
2
3
2
-,2,-

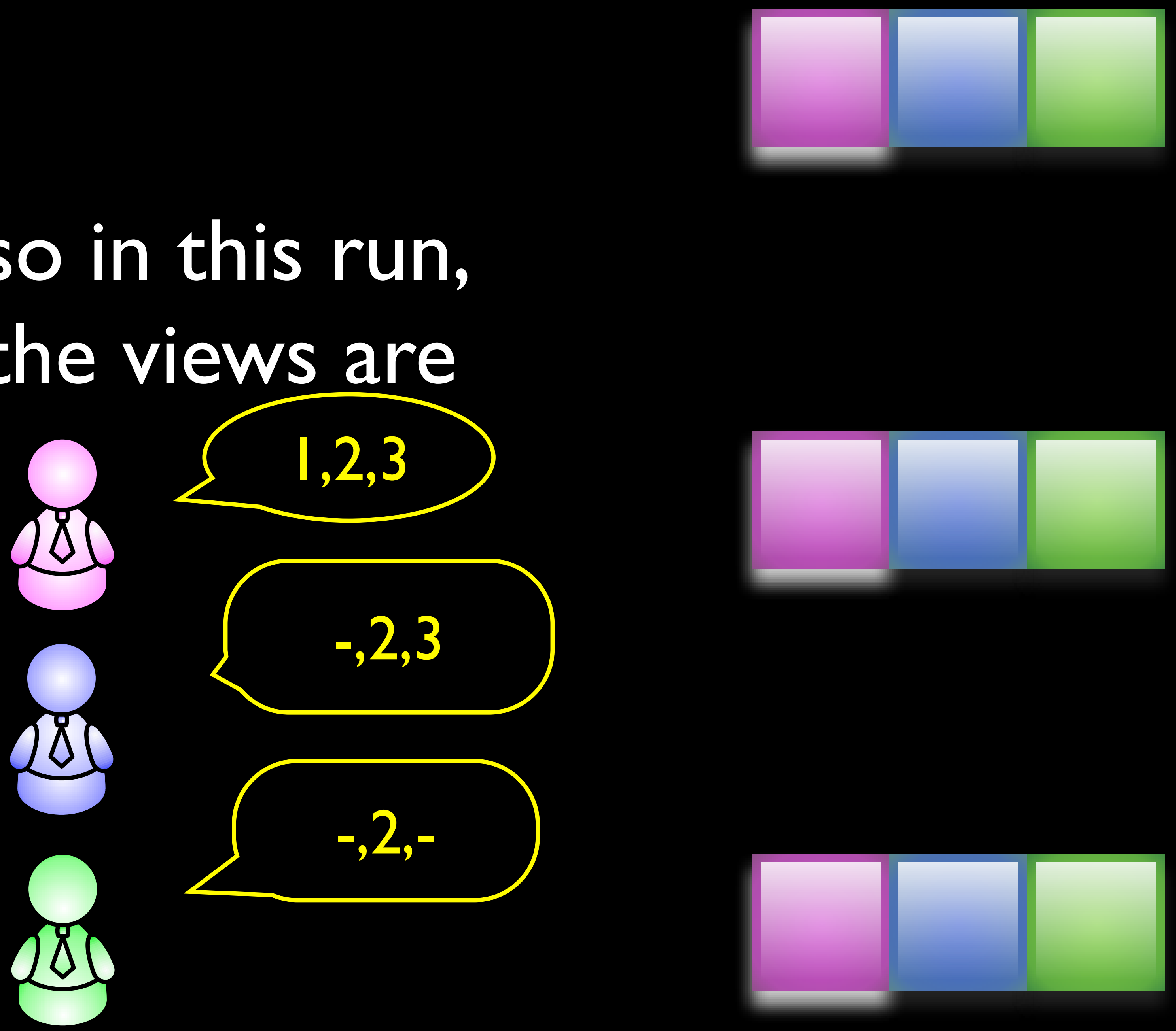
- returns -,2,-

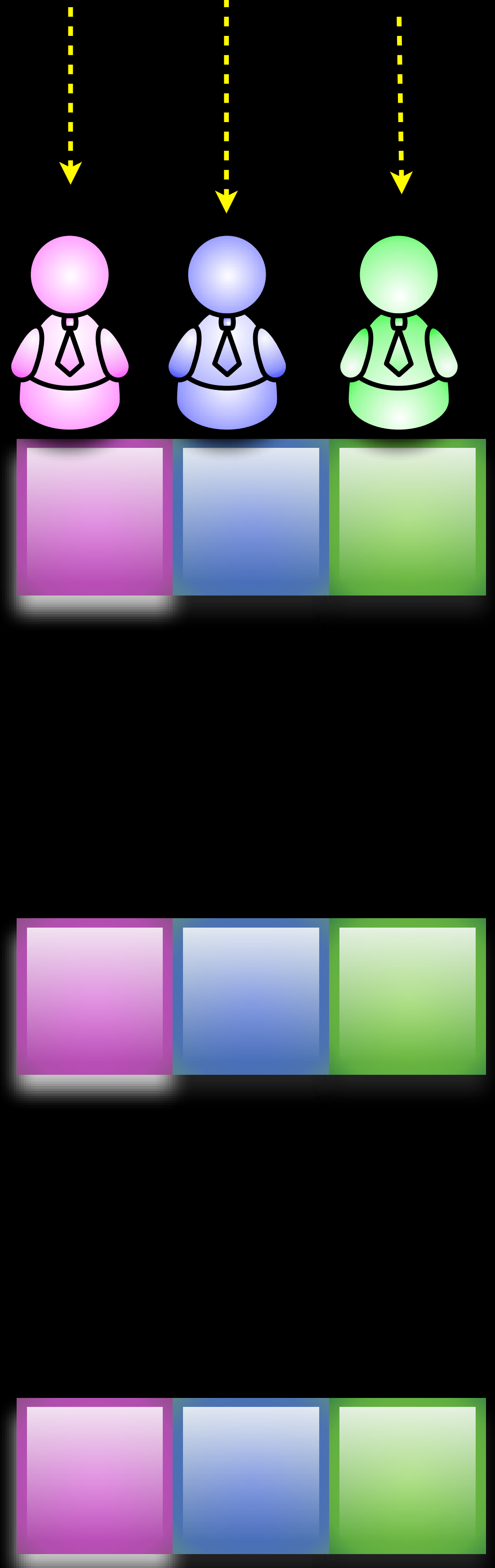


so in this run,
the views are

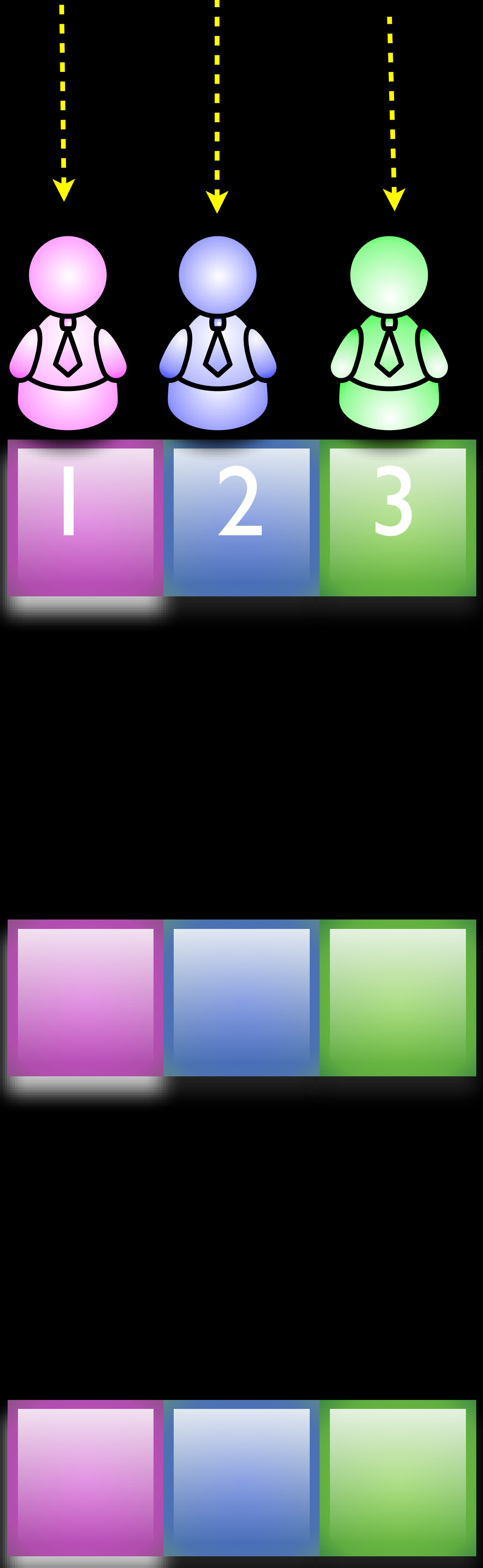


so in this run,
the views are



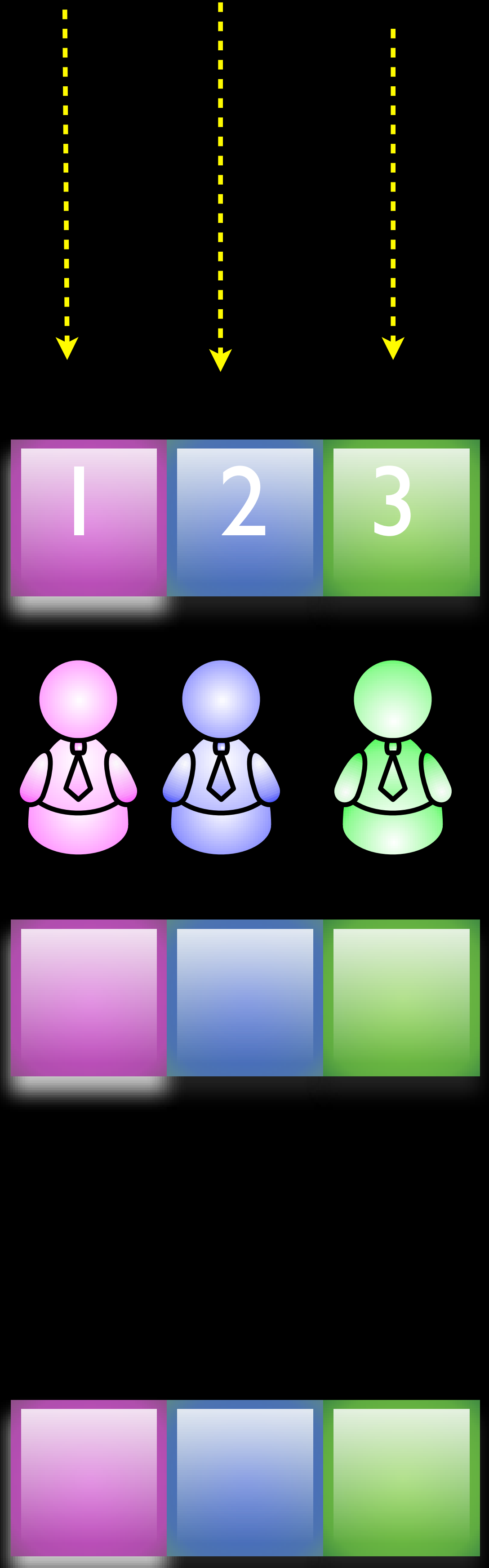


another run

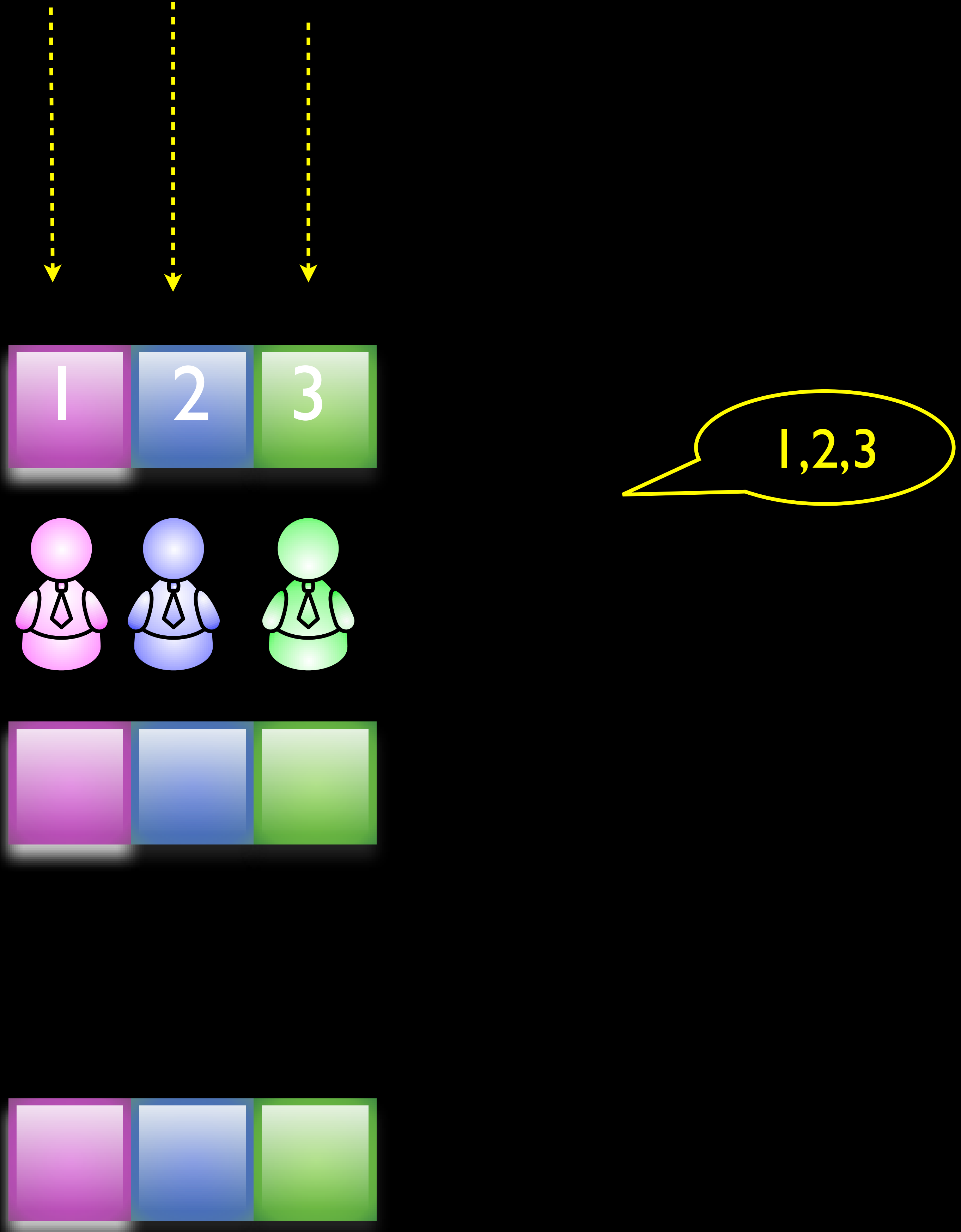


- arrive in arbitrary order

- all see all

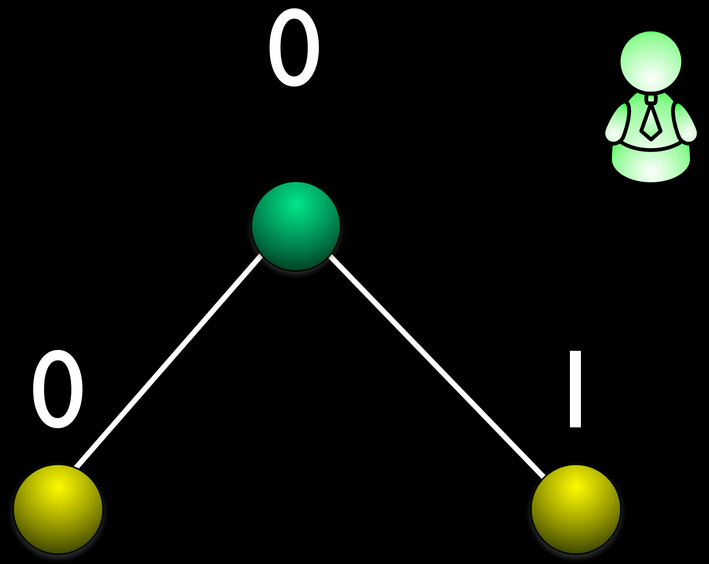


- all see all

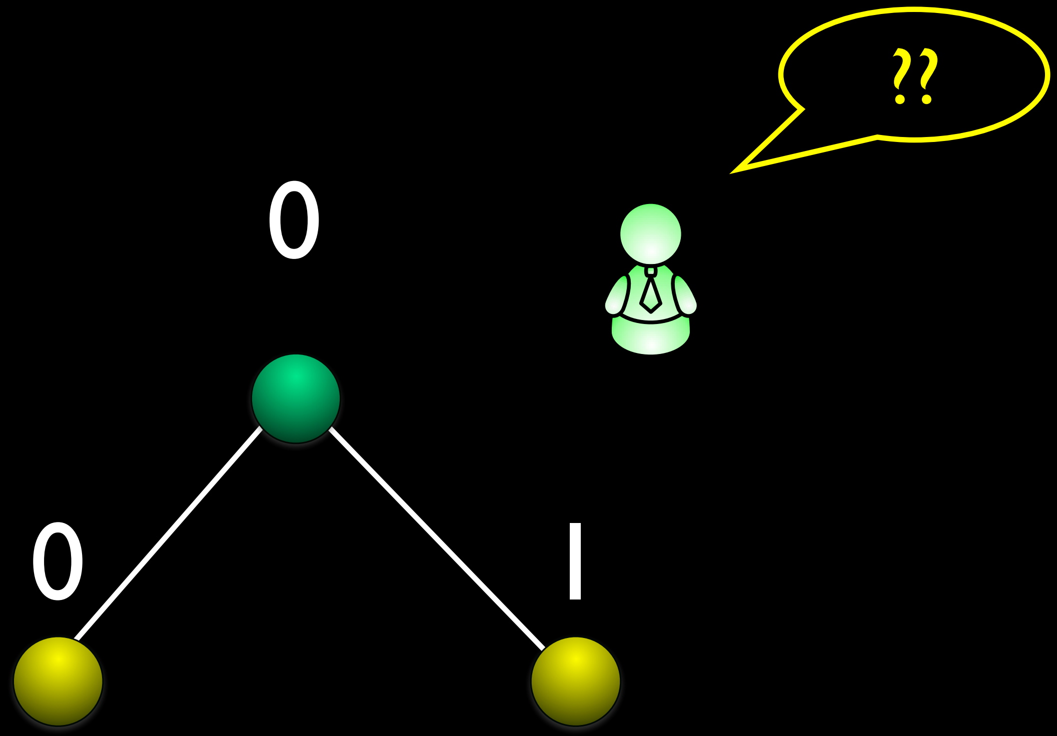


View graph

indistinguishability

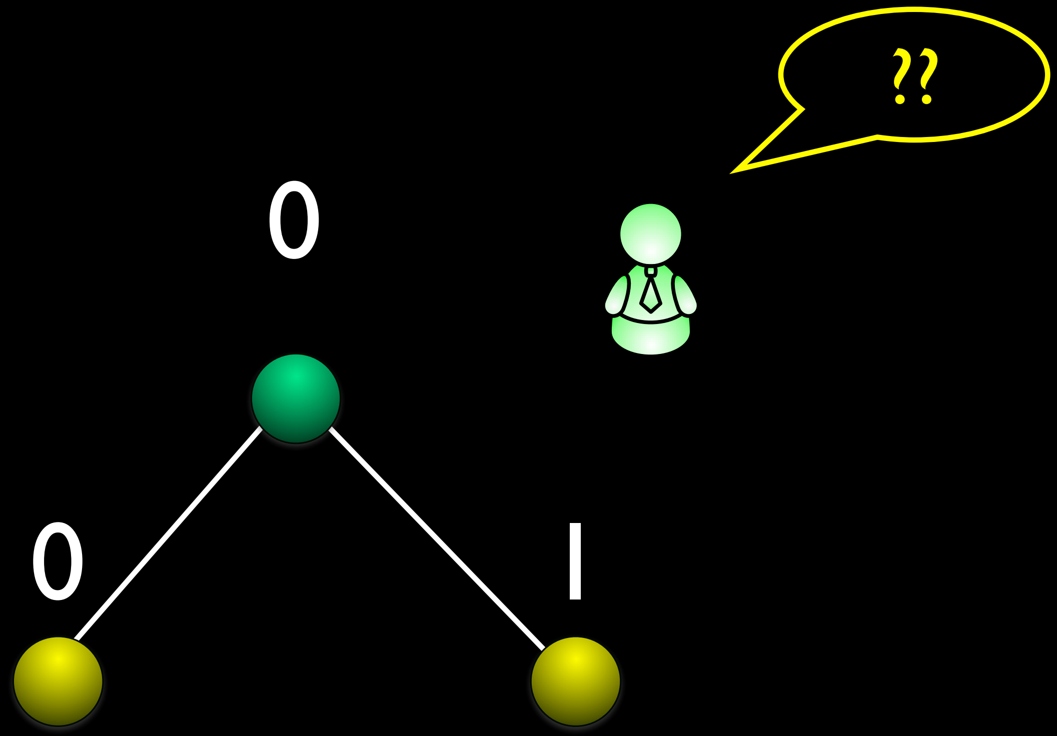


indistinguishability



indistinguishability

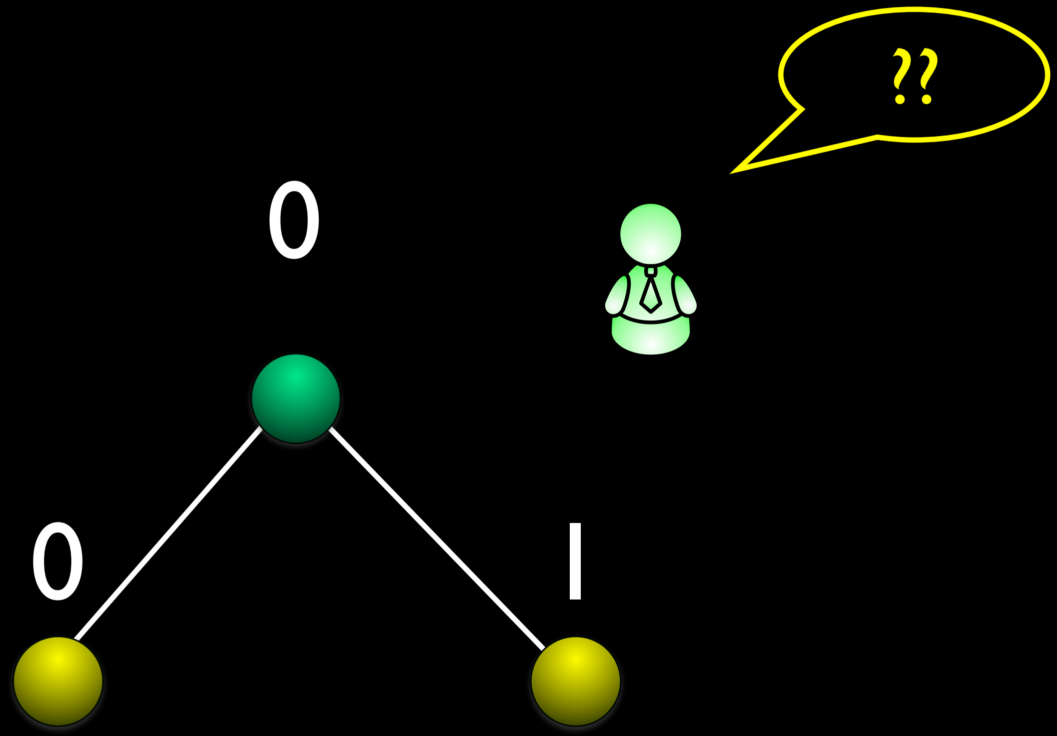
- The most essential distributed computing issue is that a process has only a local perspective of the world



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

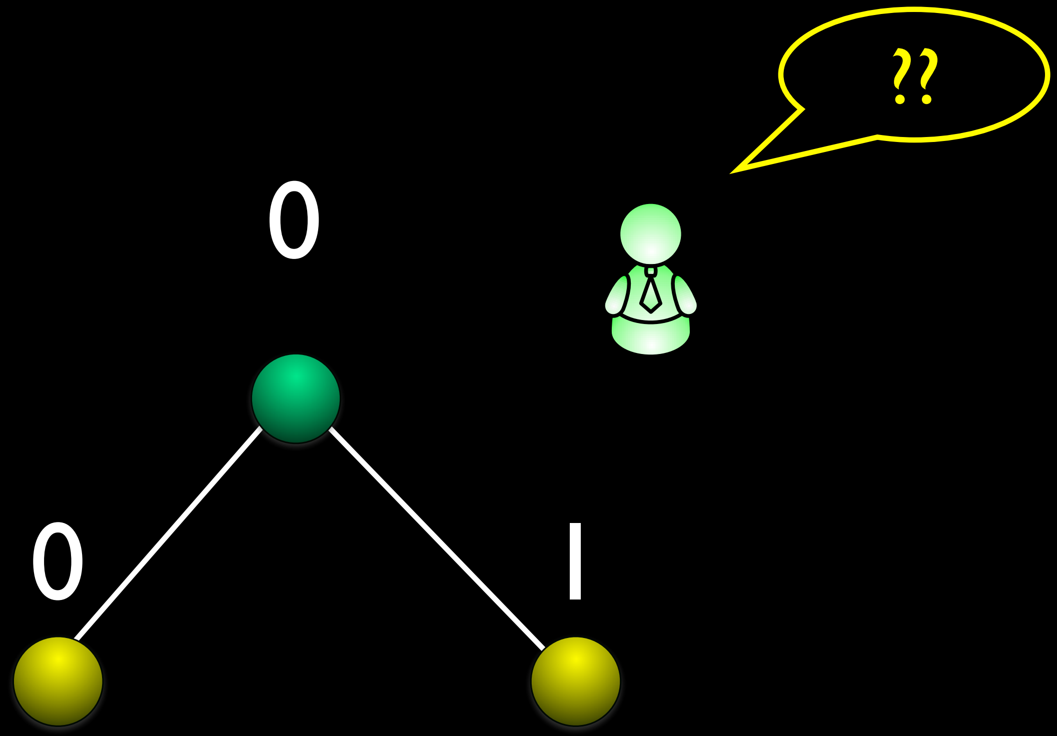


indistinguishability

- 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



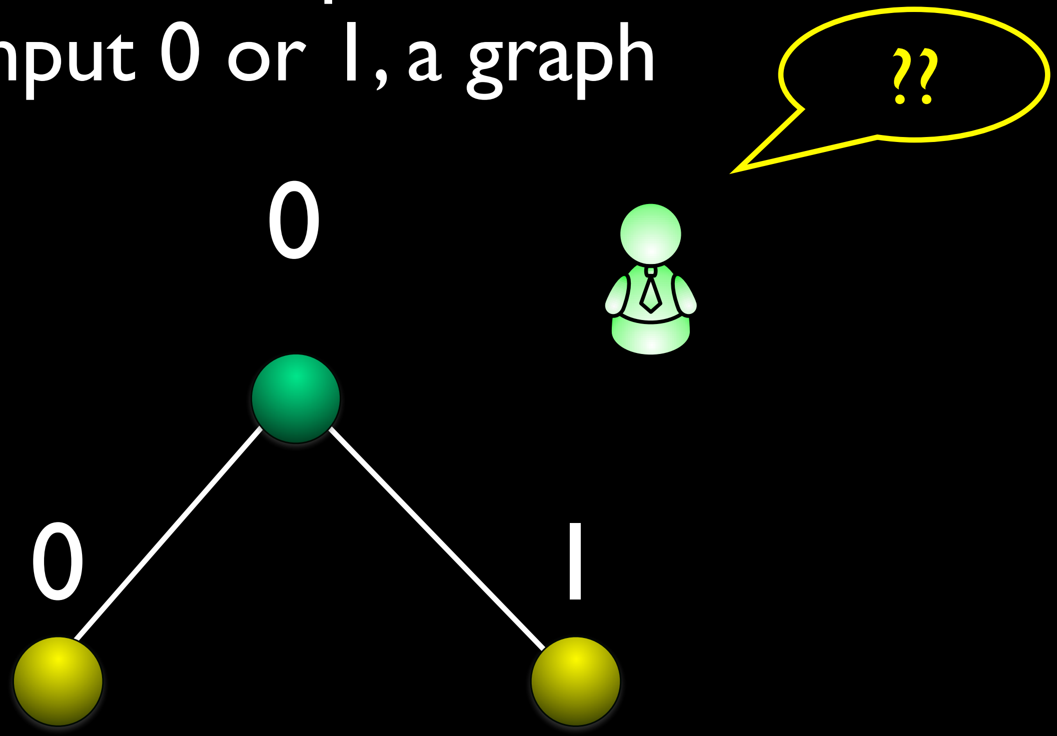
indistinguishability

- The most essential distributed computing issue is that a process has only a local perspective of the world

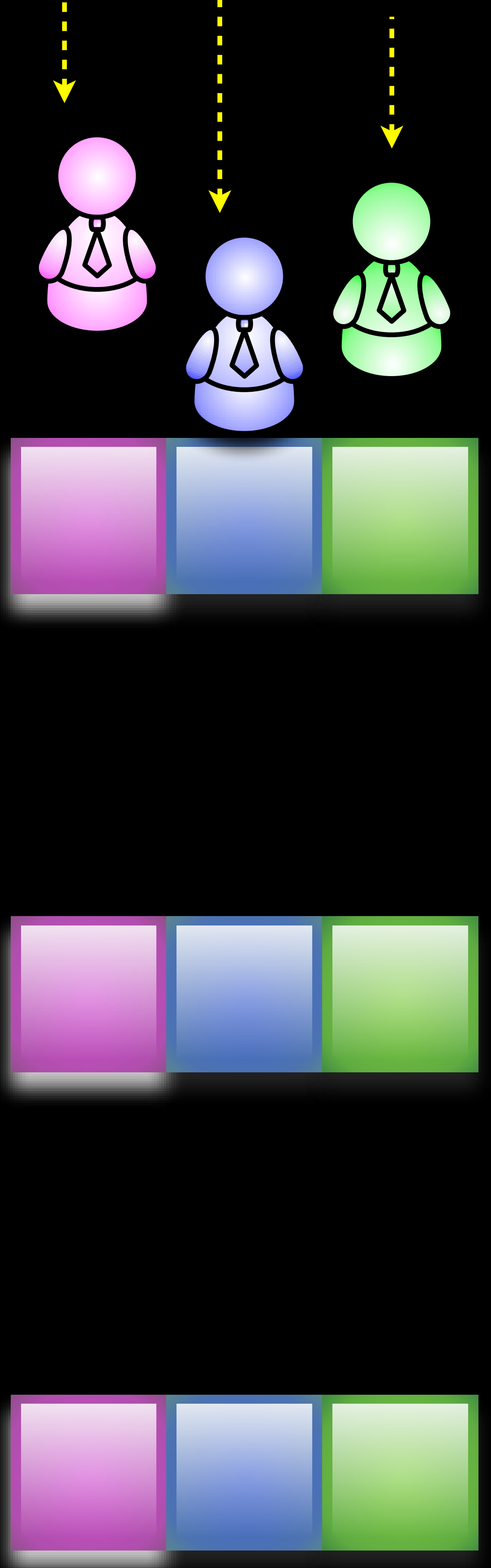
- Represent with a vertex labeled with id (green) and a local state this perspective

- E.g., its input is 0

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

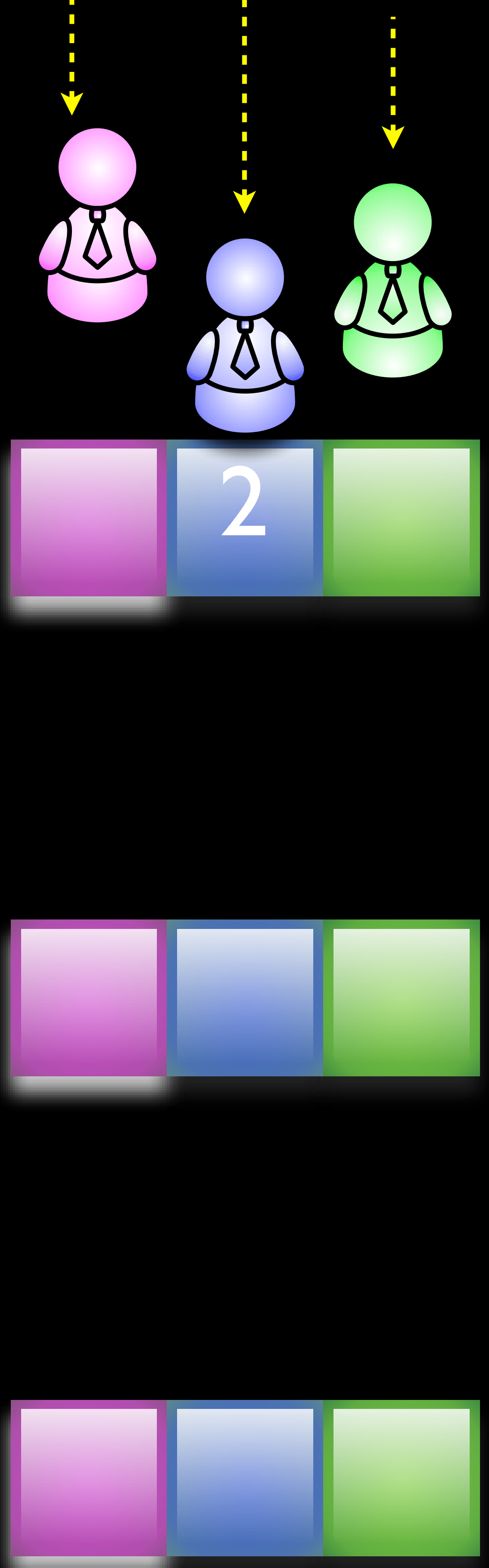


Indistinguishability graph for 2 processes

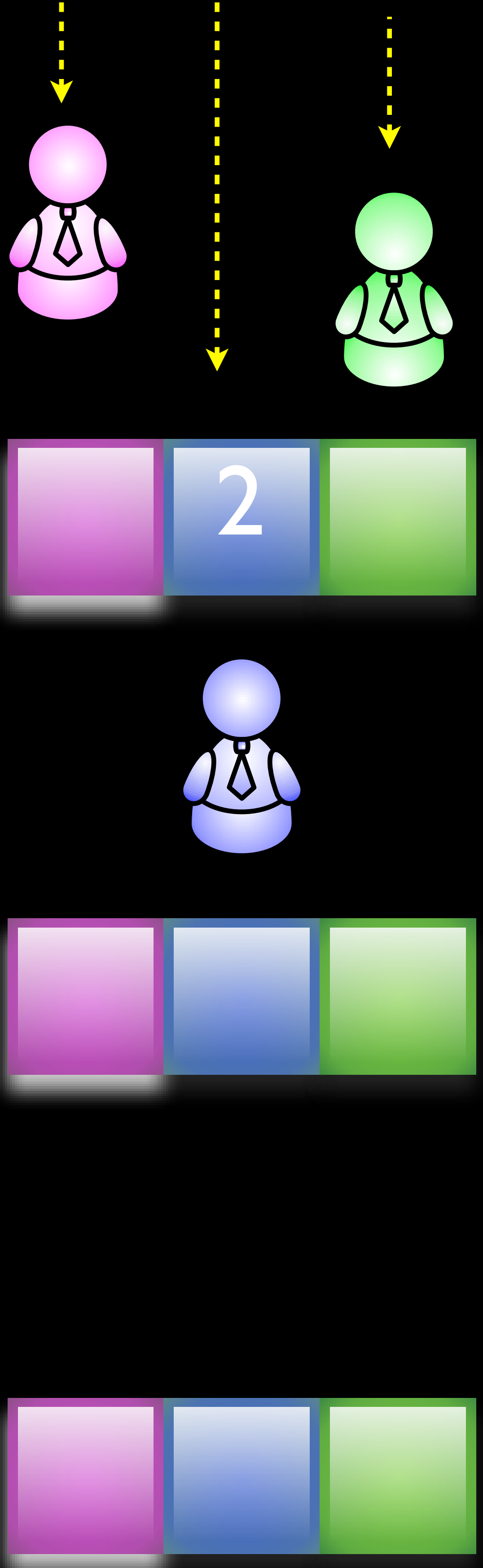


- focus on 2 processes

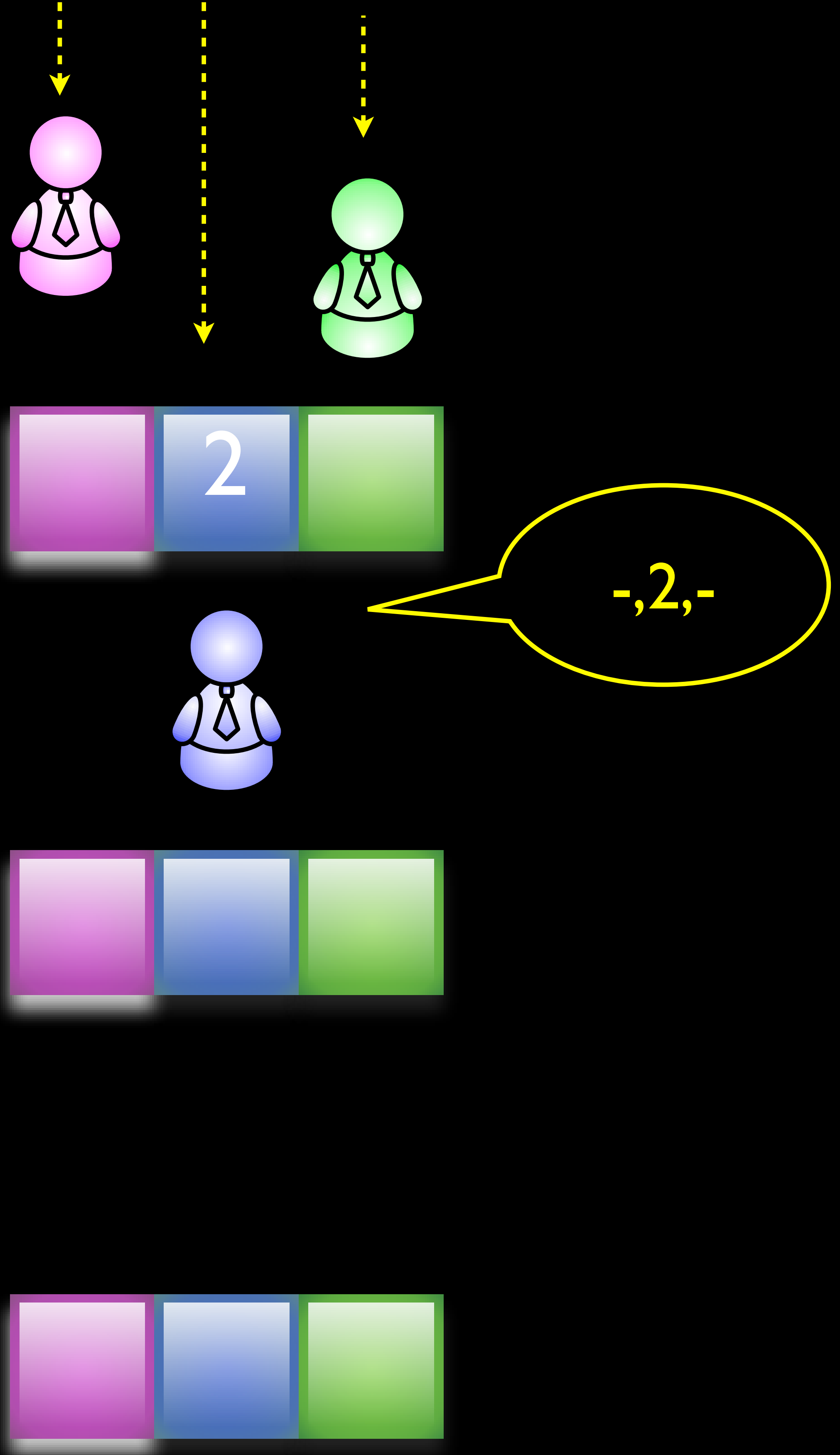
- there may be more that arrive after



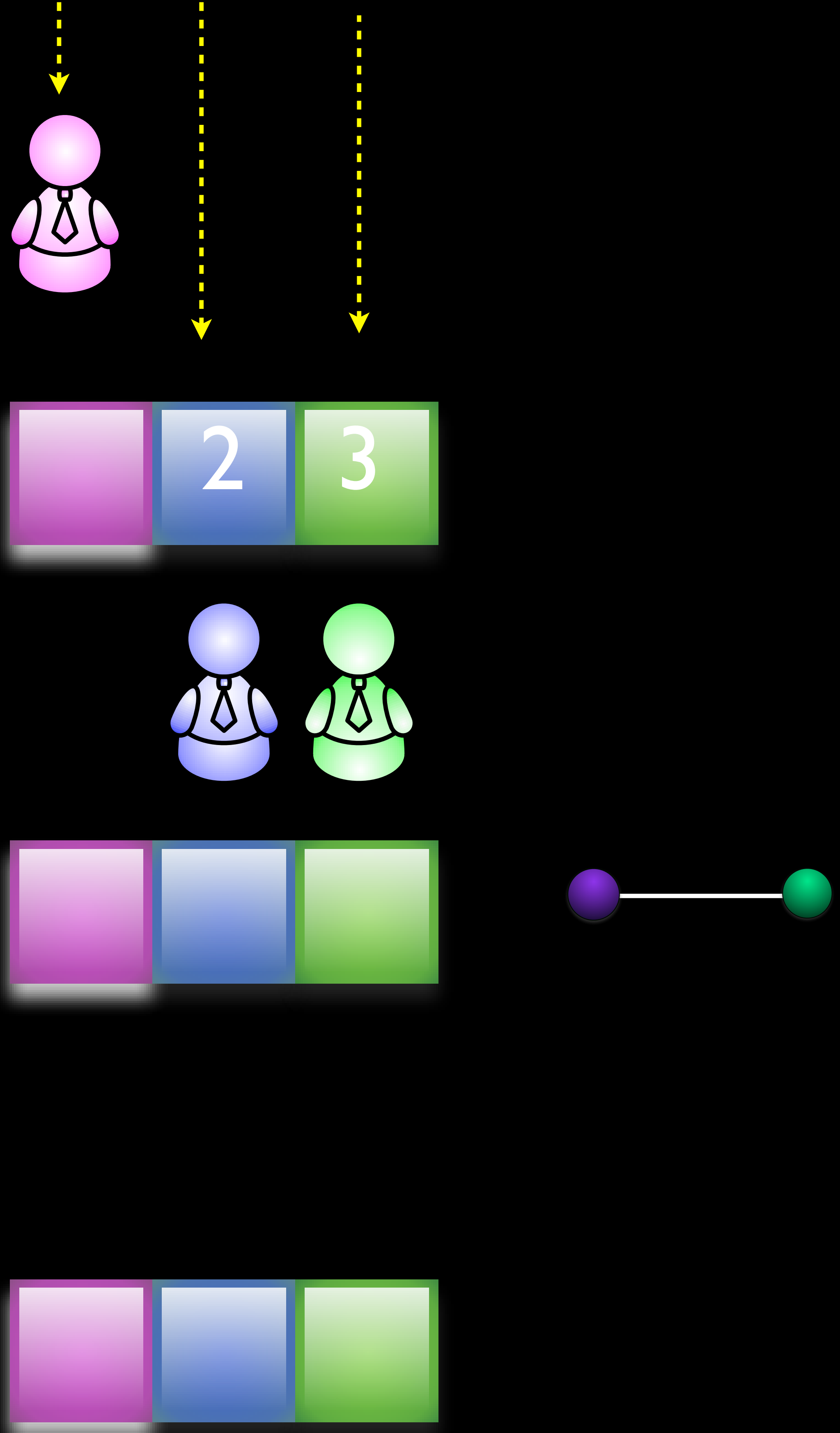
sees only itself



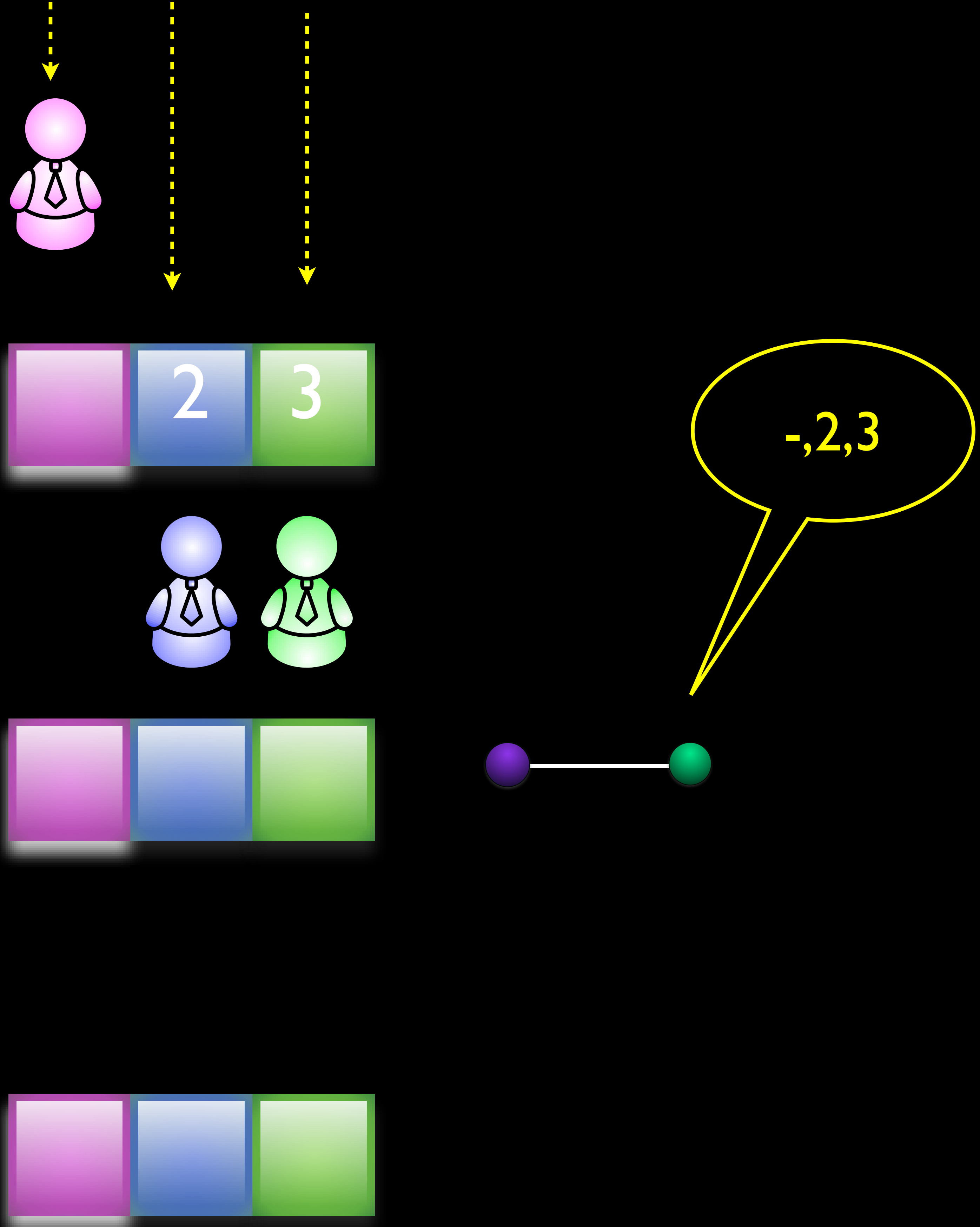
sees only itself



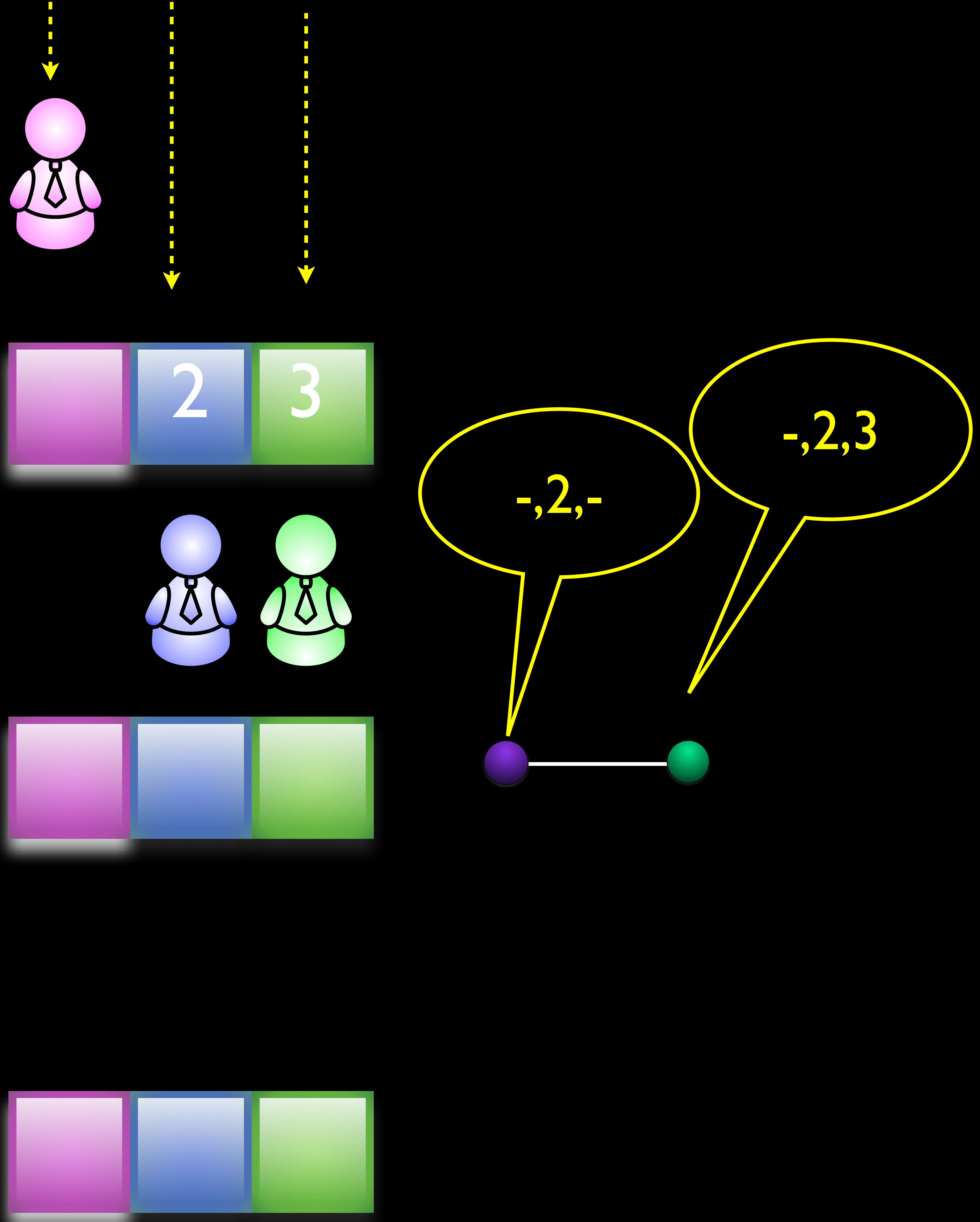
- green sees both
- but ...



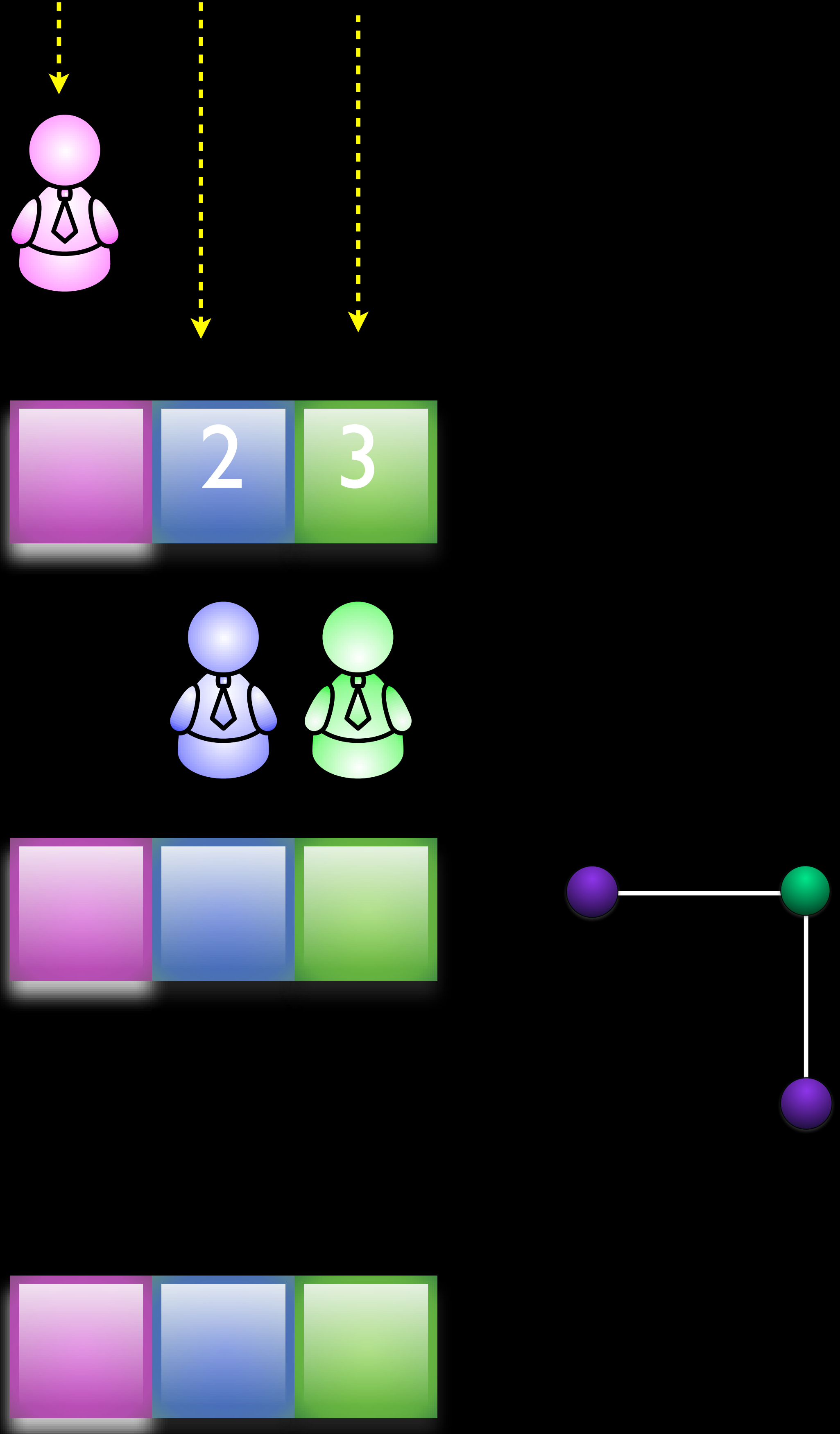
- green sees both
- but ...



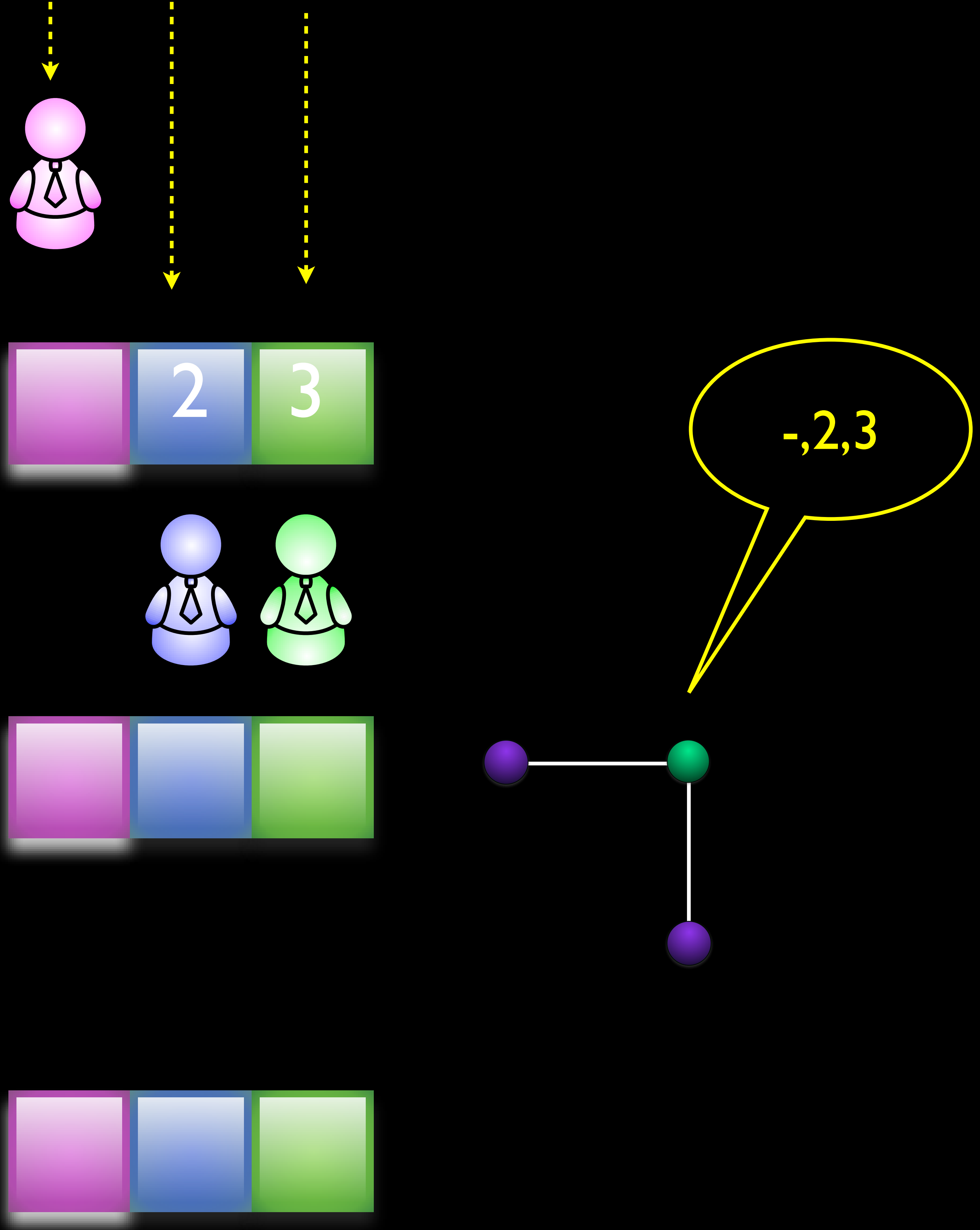
- green sees both
- but ...



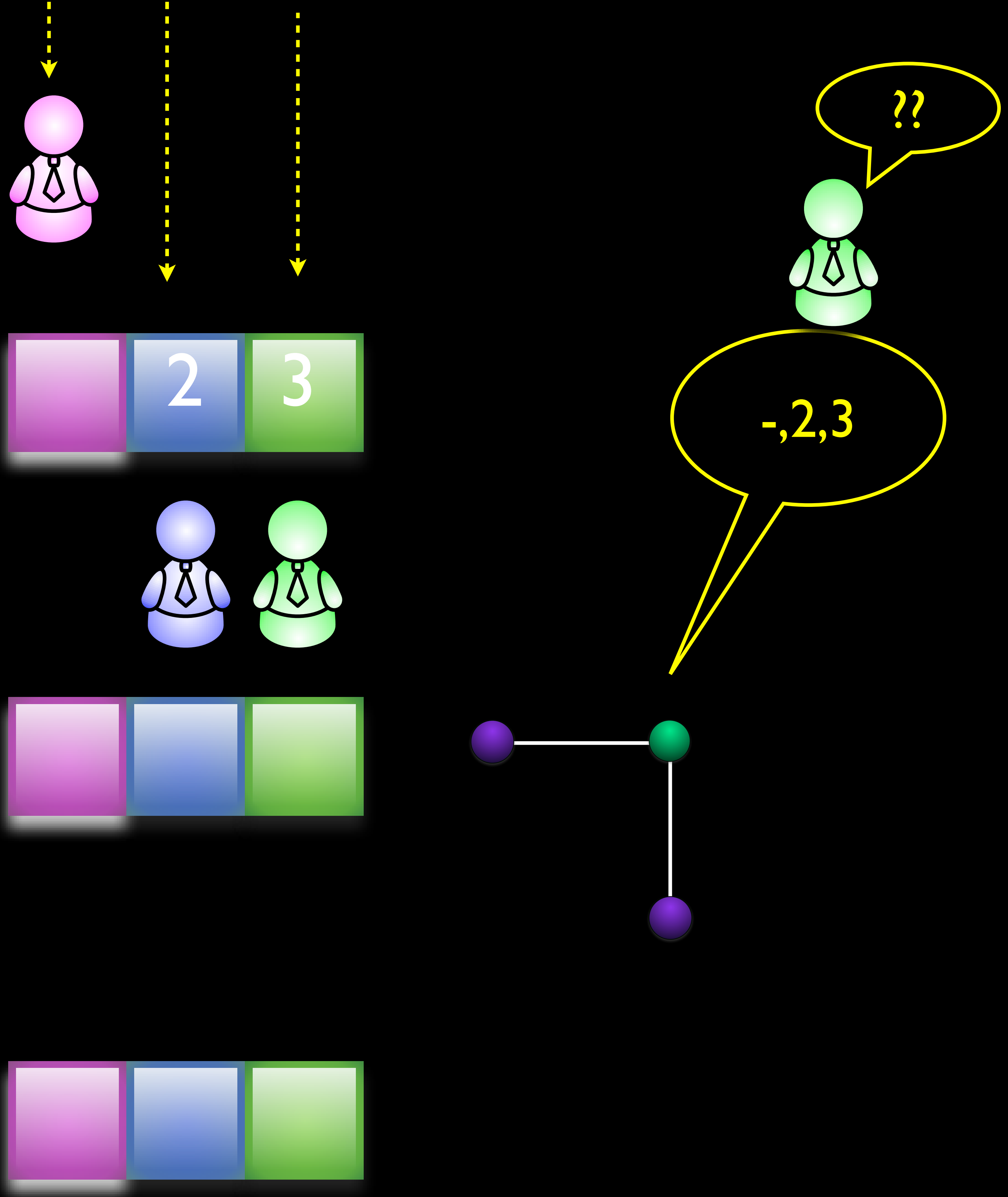
- green sees both
- but, doesn't know if seen by the other



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

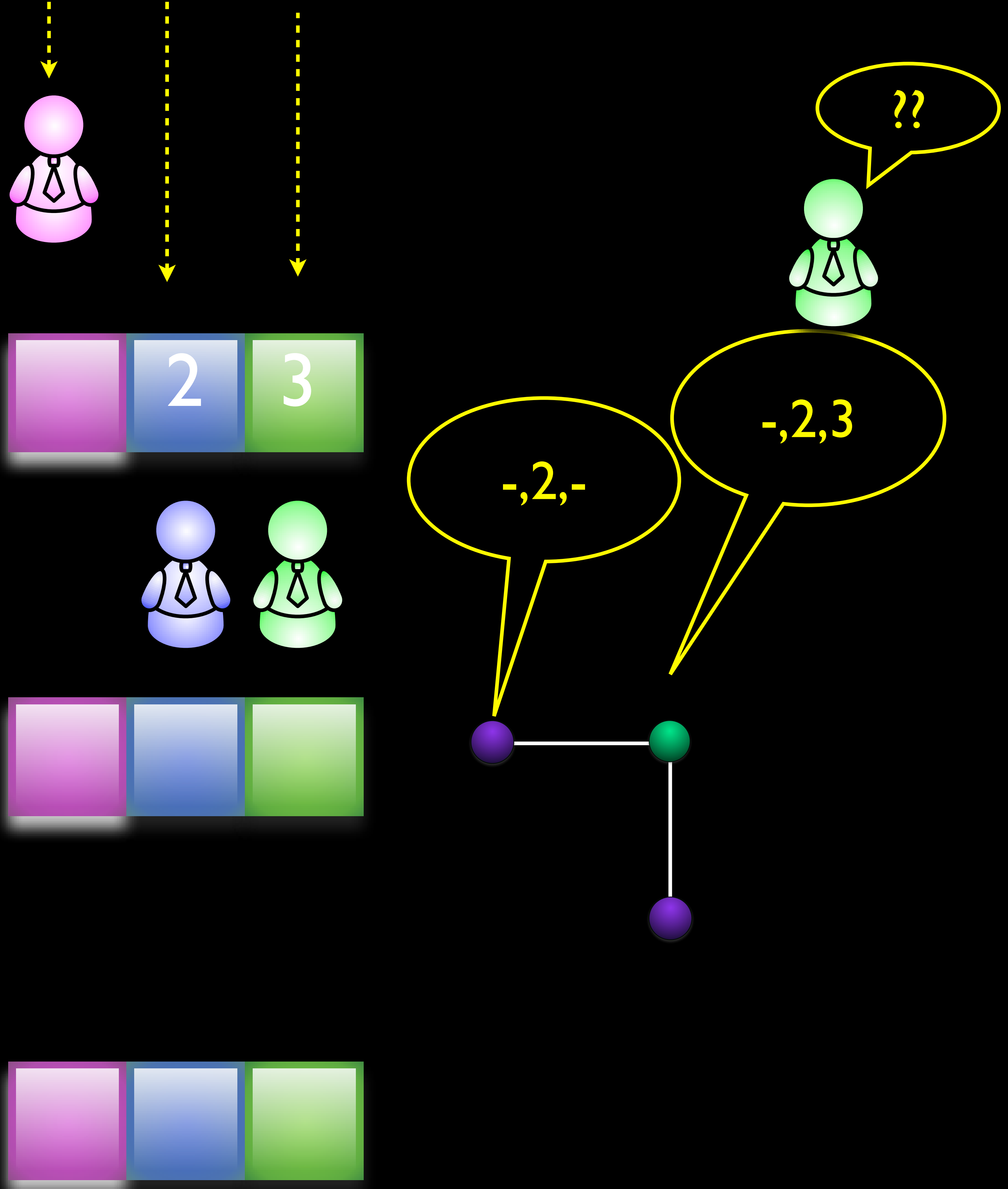


- green sees both
- but, doesn't know if seen by the other



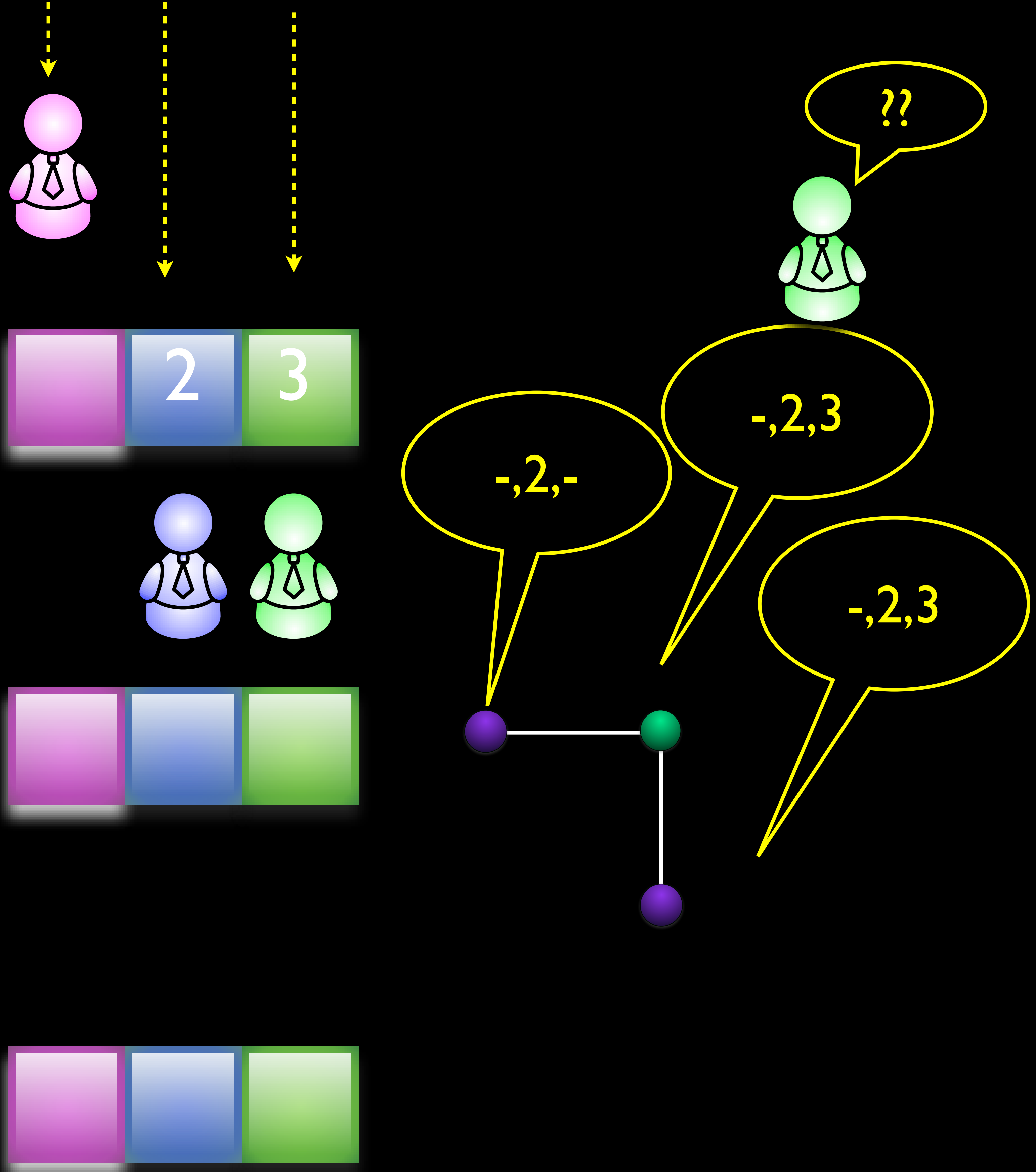
- green sees both

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

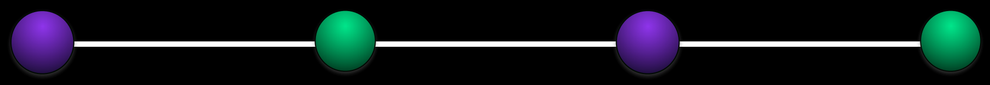


- green sees both

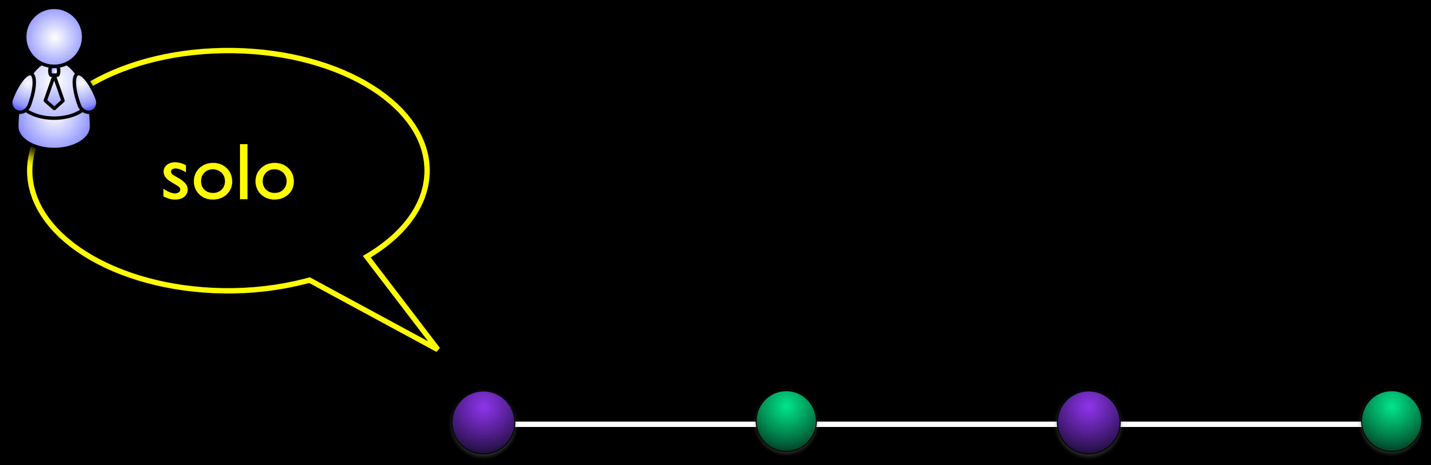
- but, doesn't know if seen by the other



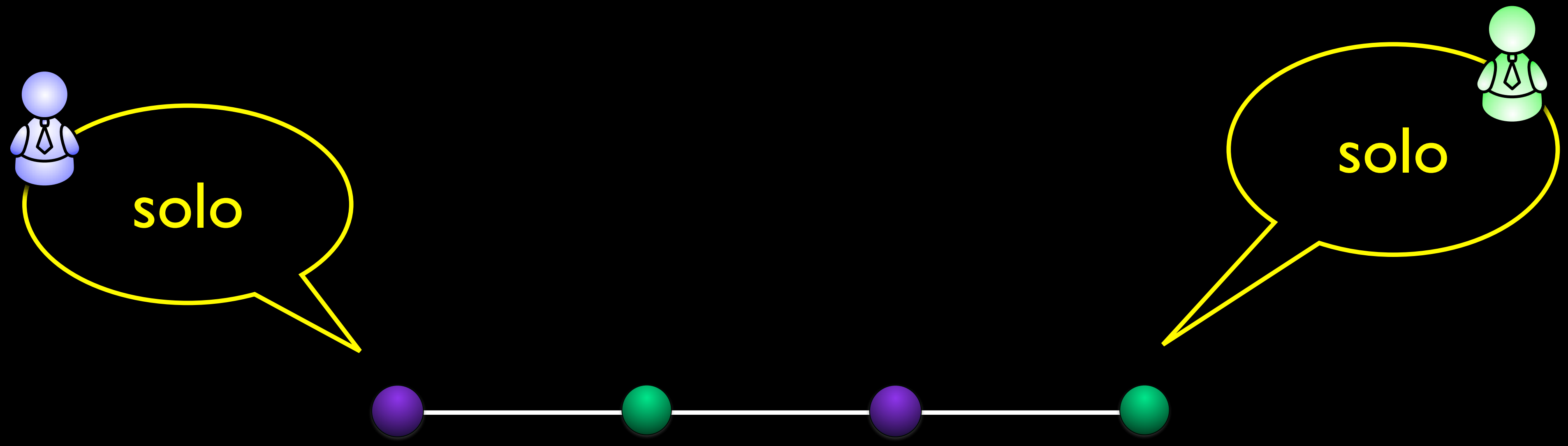
one round graph for 2 processes



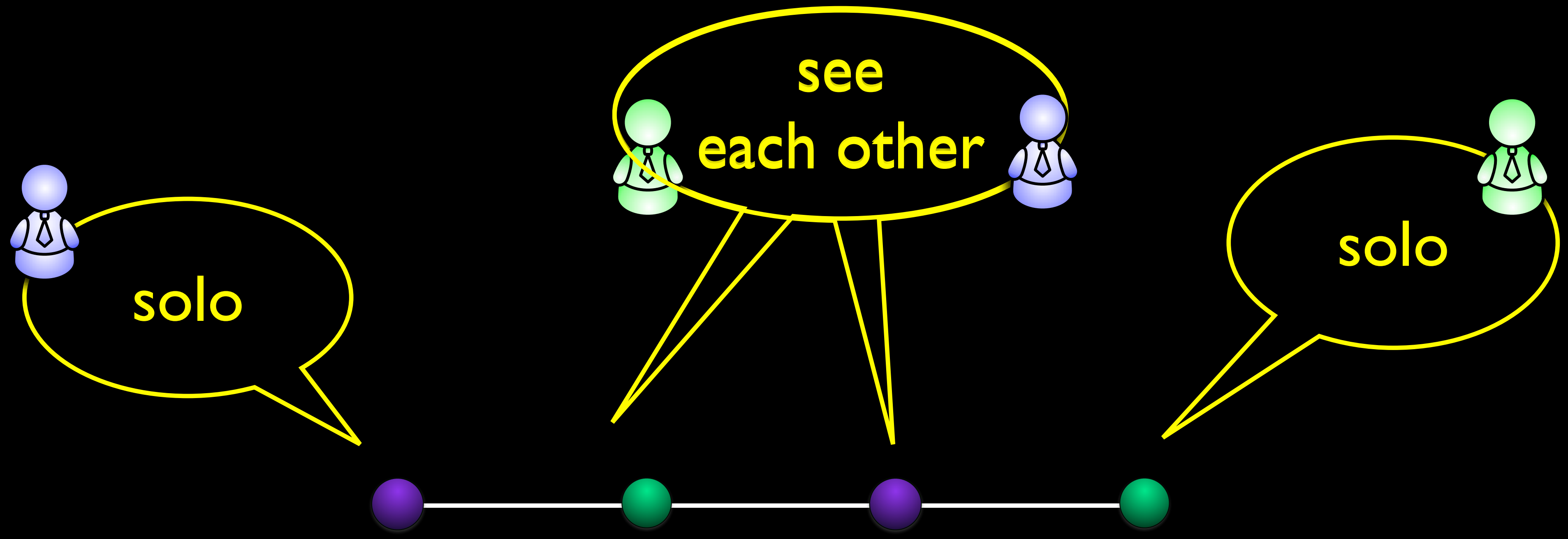
one round graph for 2 processes



one round graph for 2 processes



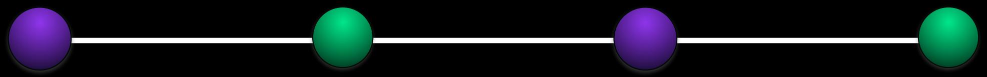
one round graph for 2 processes



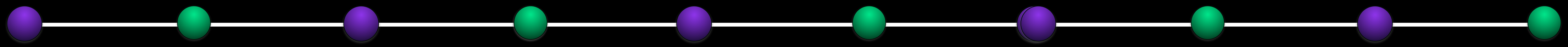
iterated runs

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

round 1:

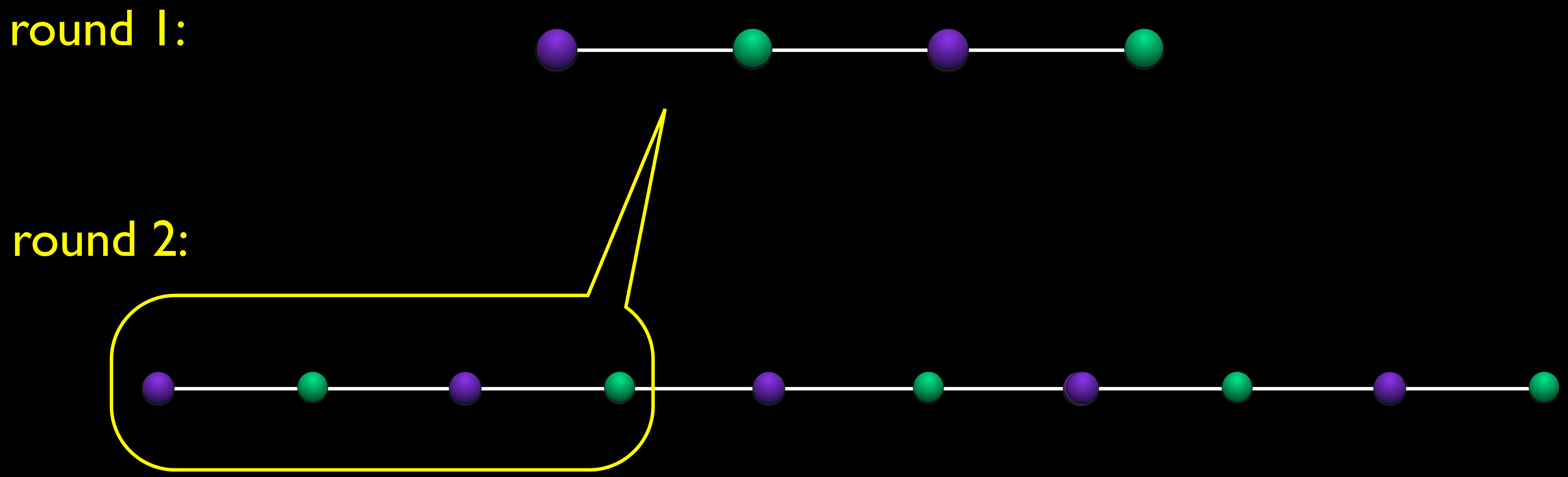


round 2:



iterated runs

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

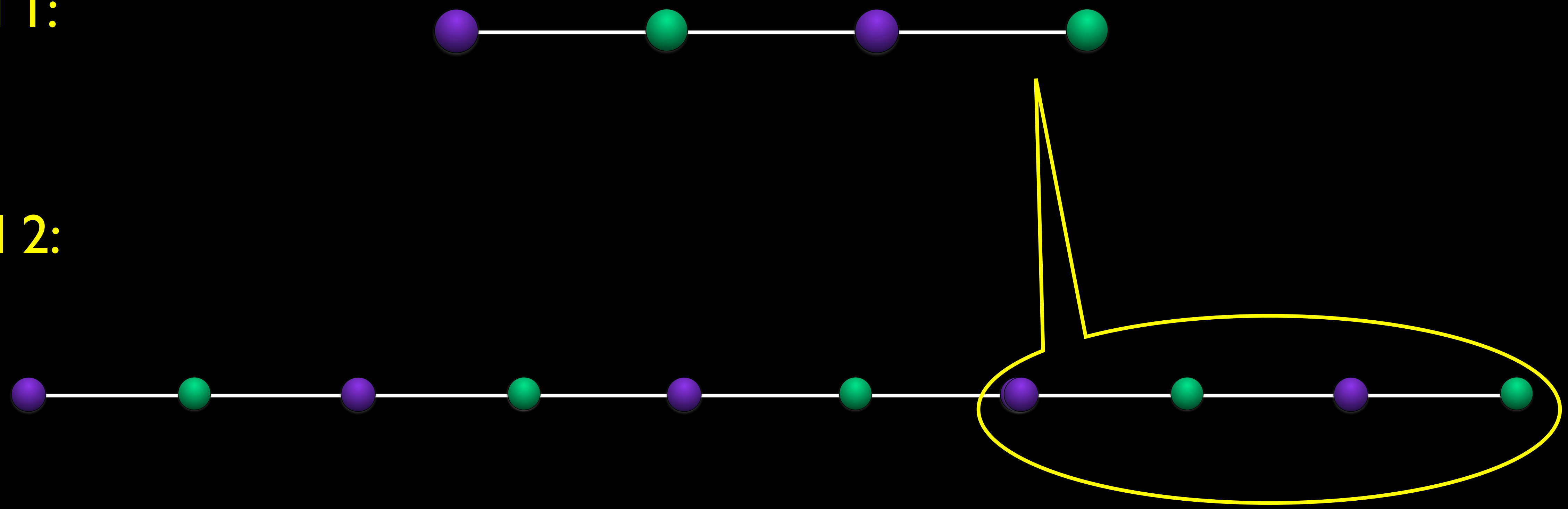


iterated runs

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

round 1:

round 2:

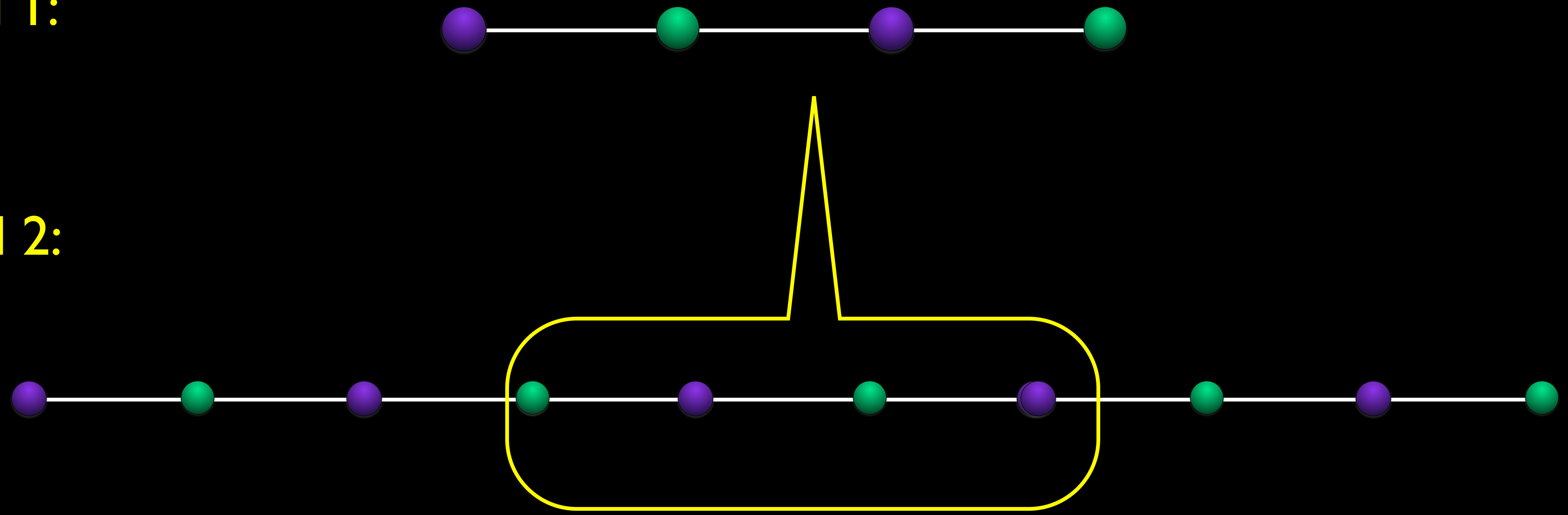


iterated runs

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

round 1:

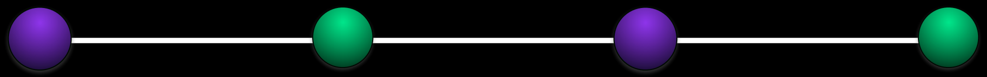
round 2:



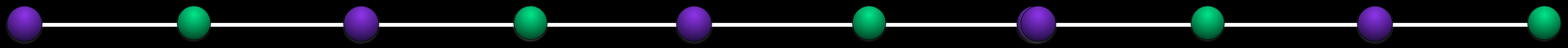
iterated runs

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

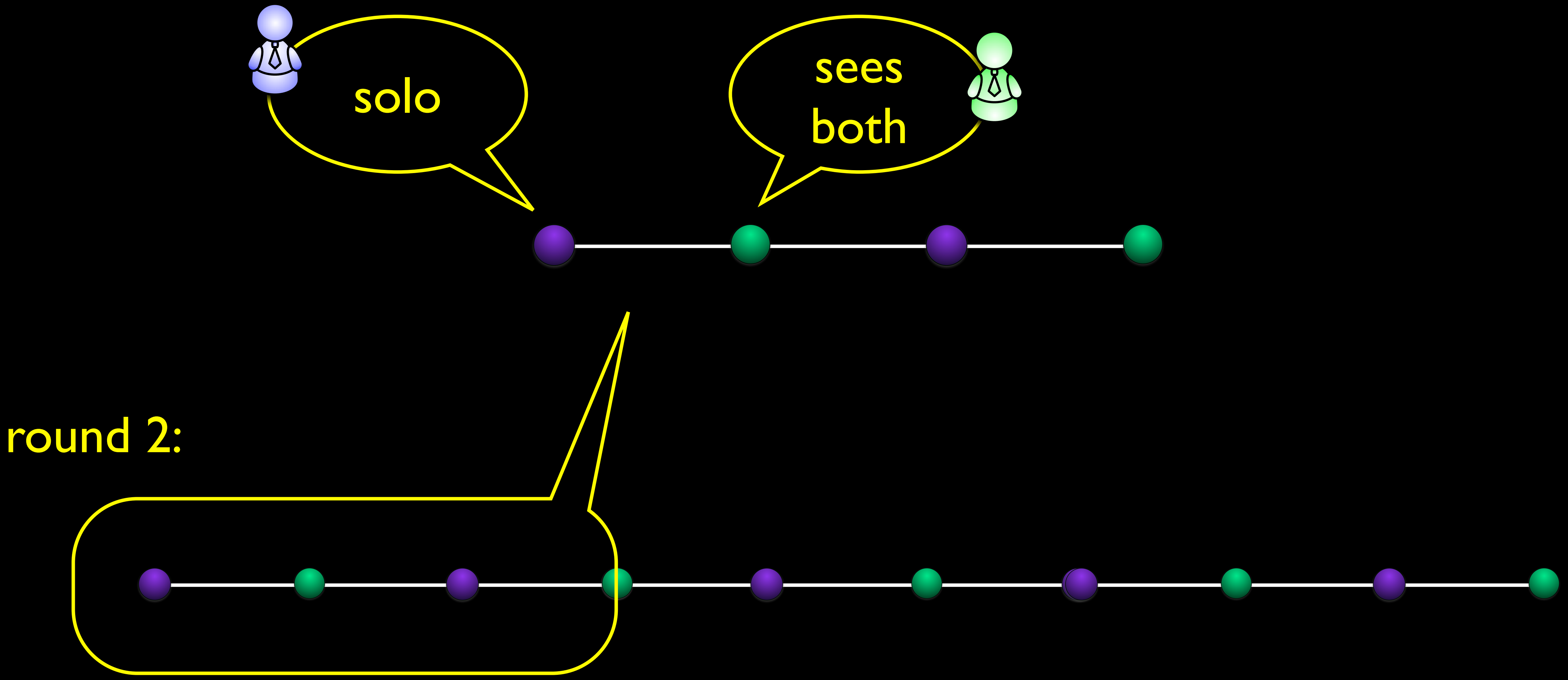
round 1:



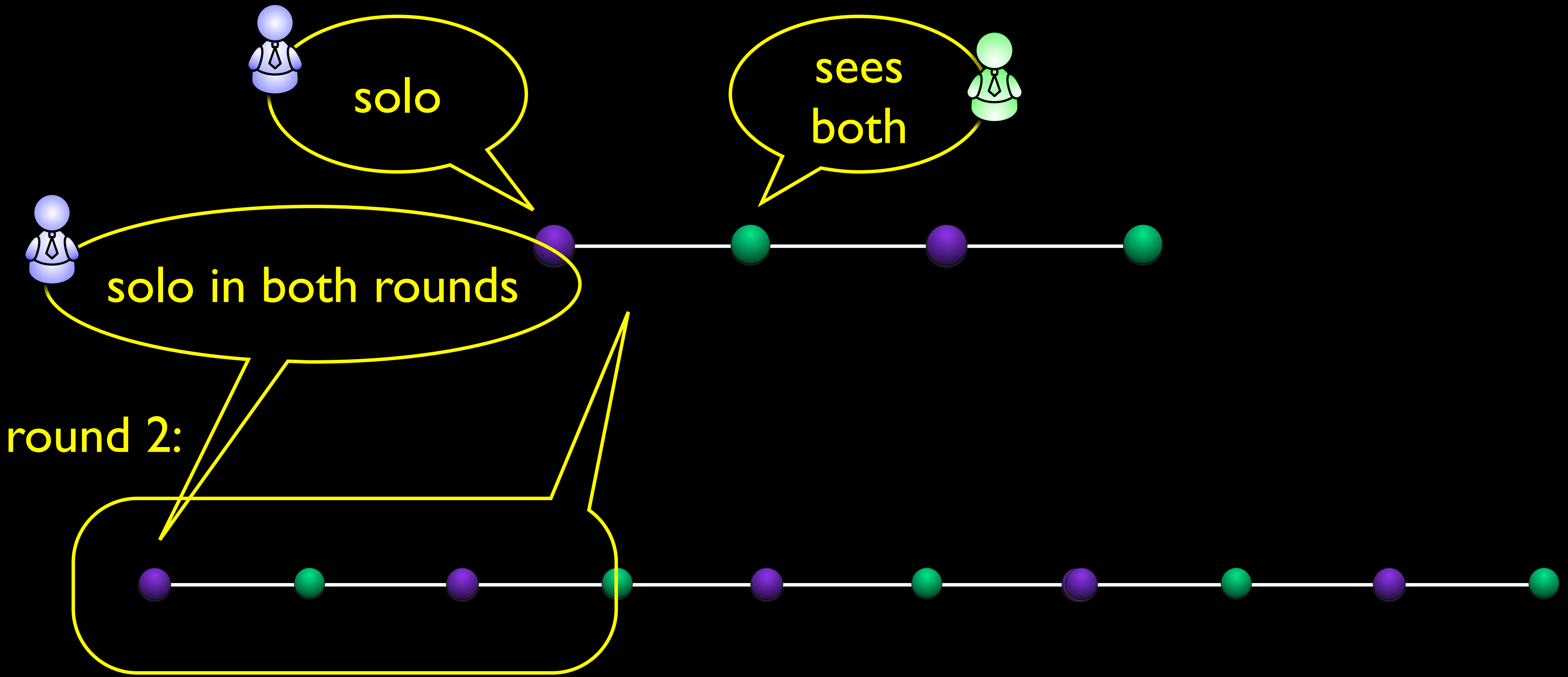
round 2:



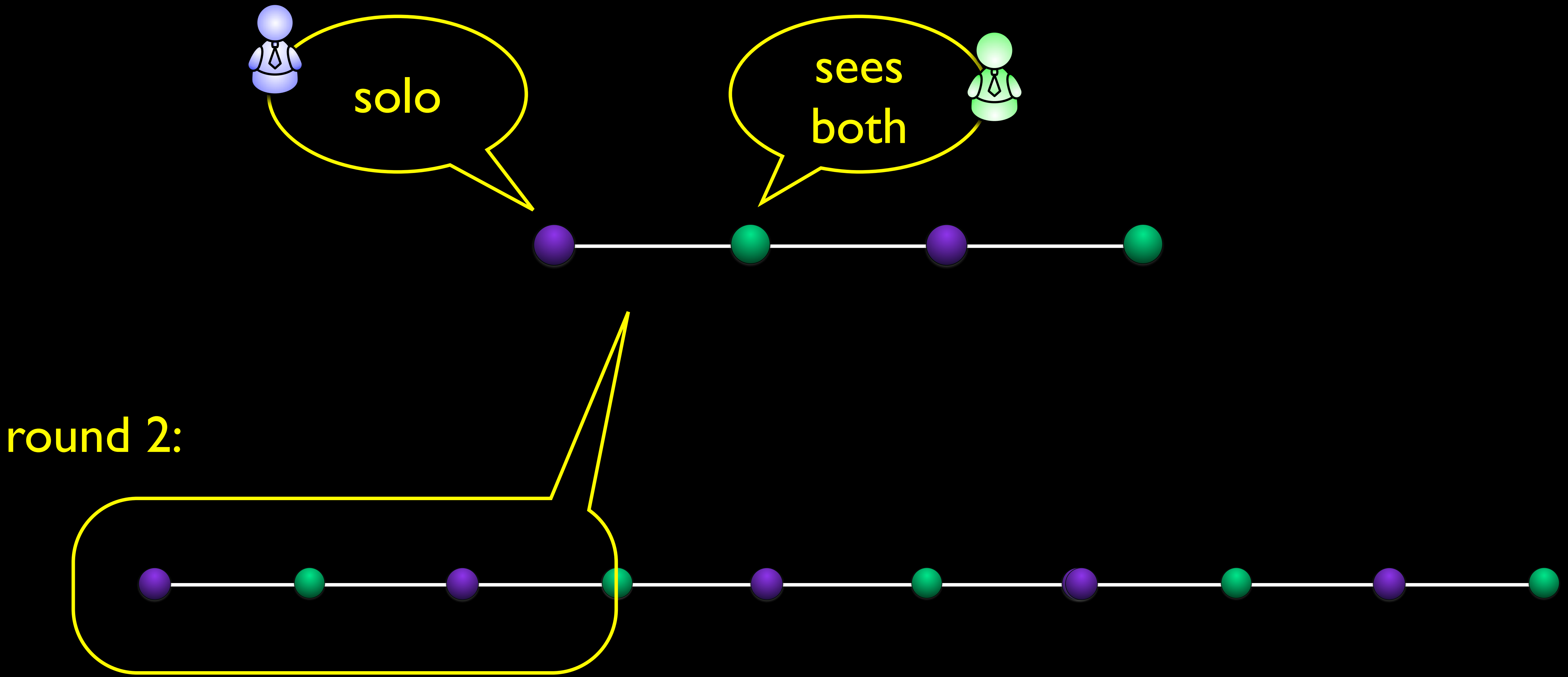
iterated runs



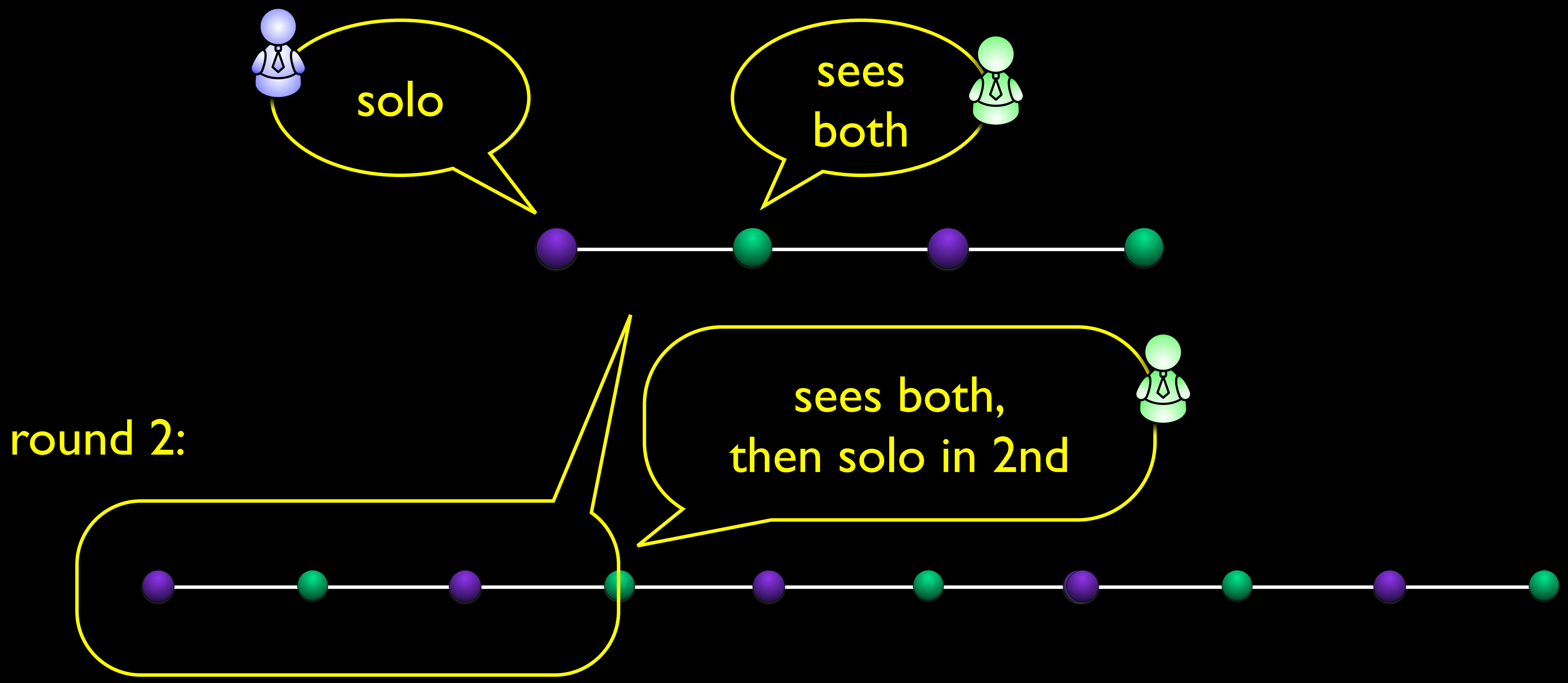
iterated runs



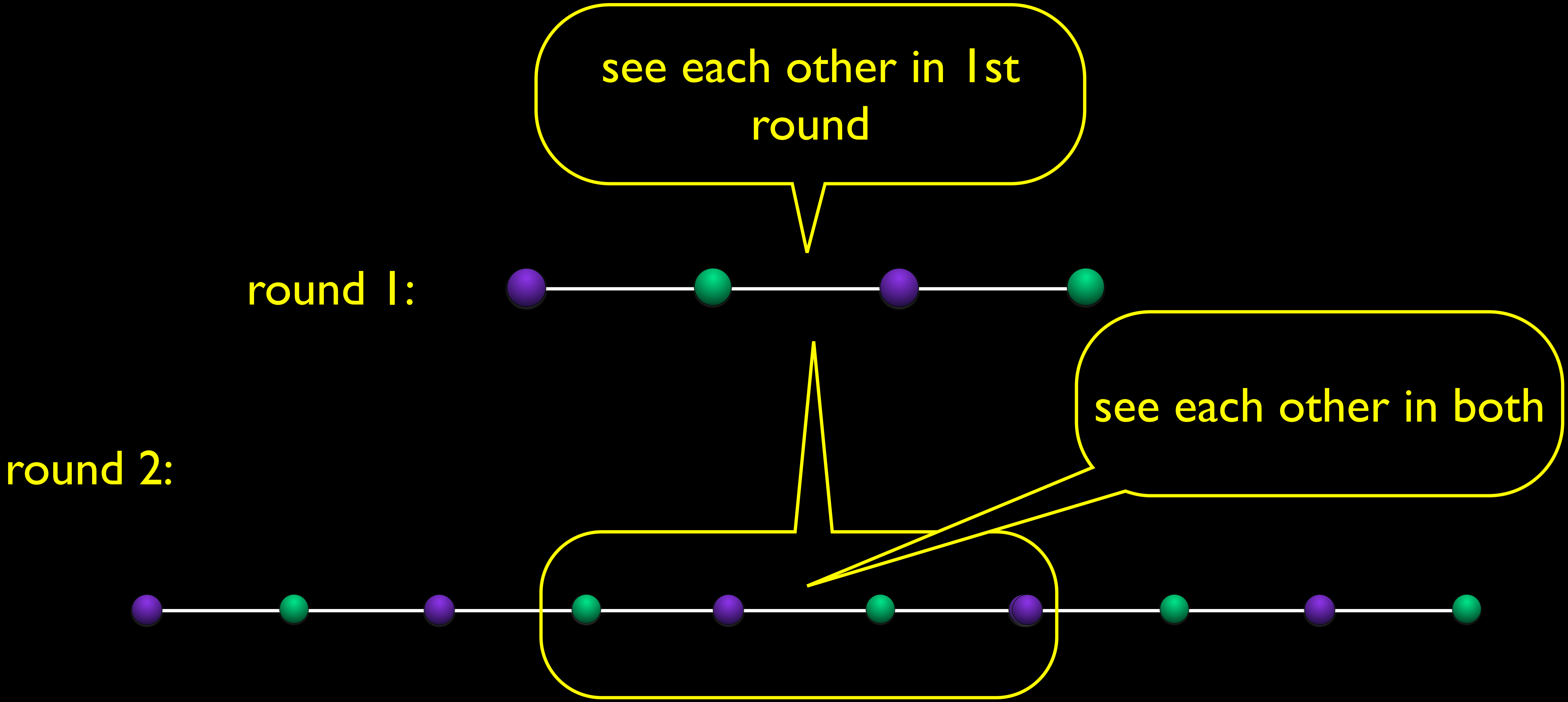
iterated runs



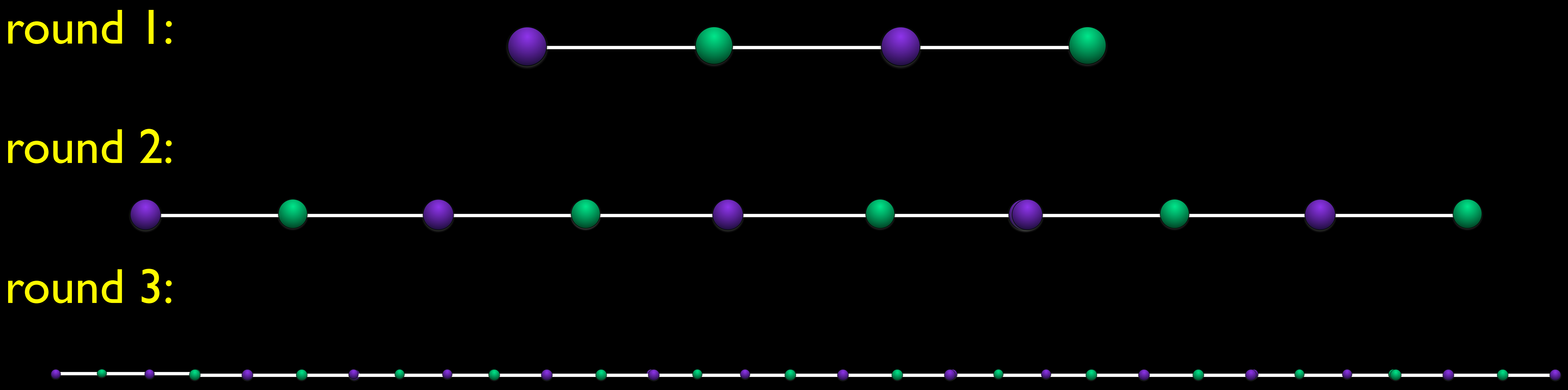
iterated runs



iterated runs



More rounds



Topological invariant: protocol graph after *k* rounds

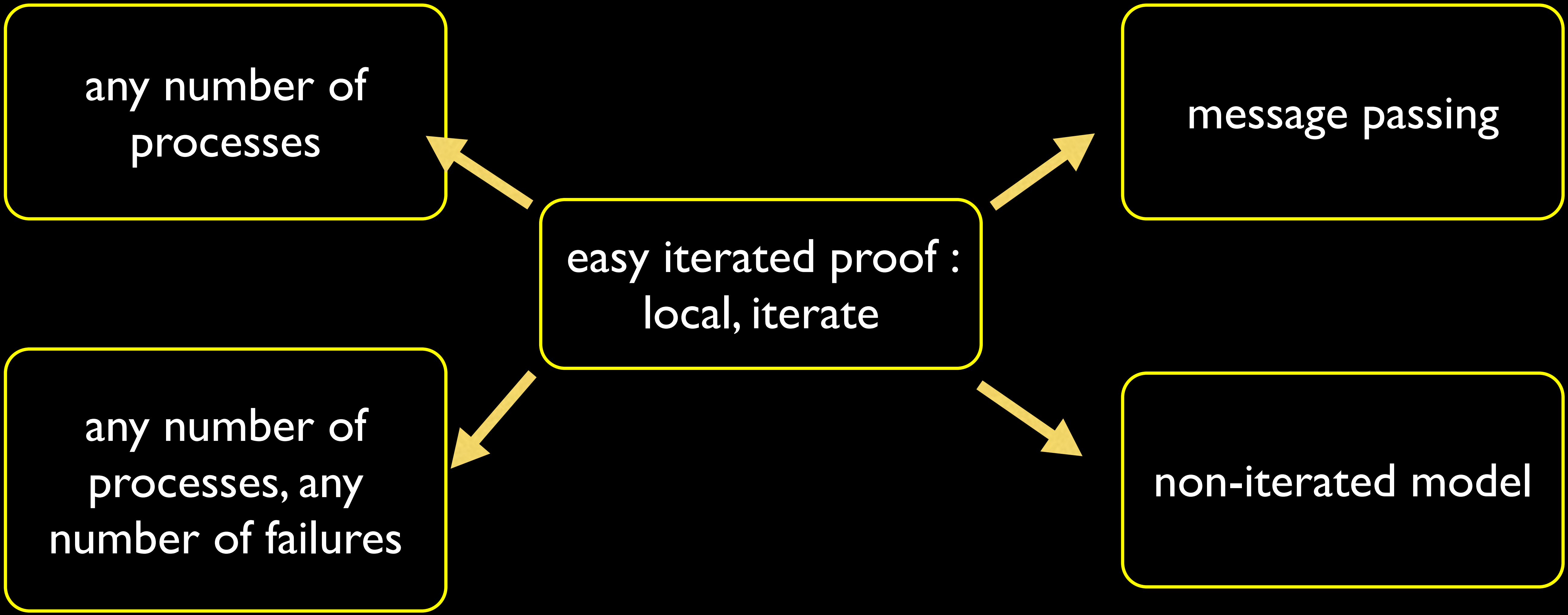
-longer
-but always connected

Wait-free theorem for 2 processes

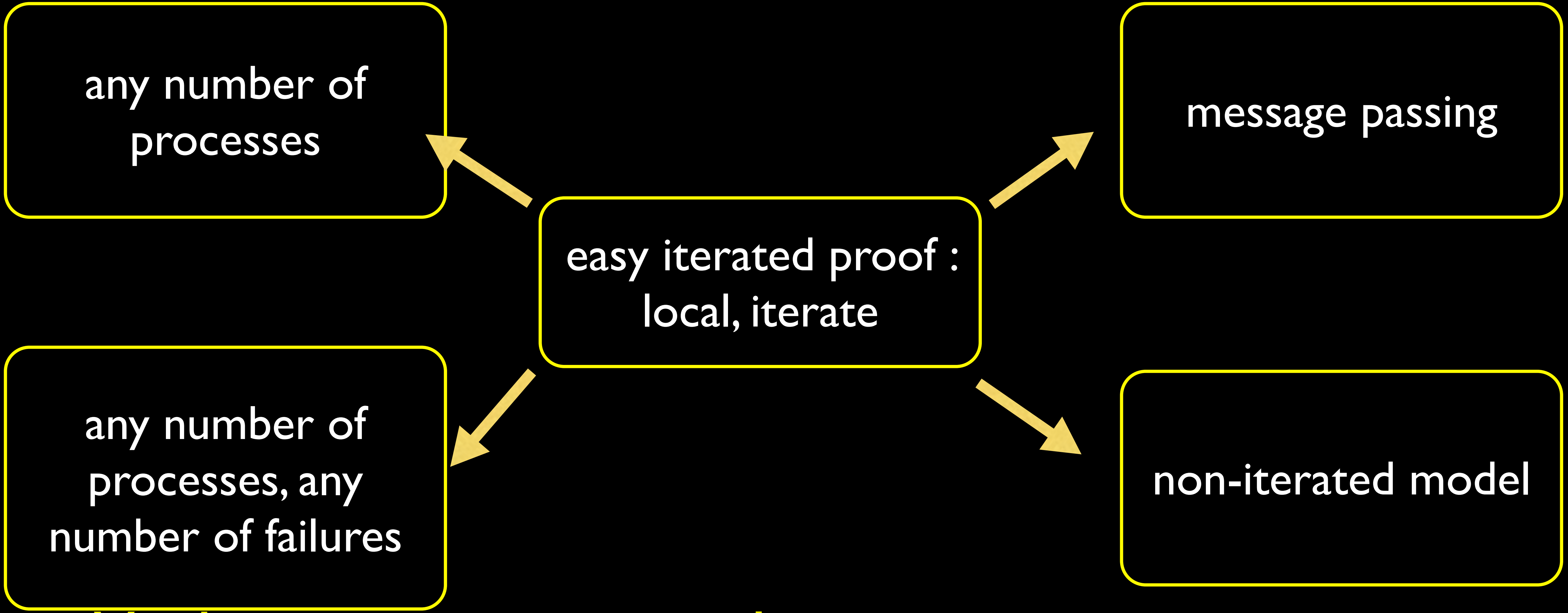
For any protocol in the iterated model,
its graph after *k* rounds is

-longer
-but always connected

Iterated approach: theorem holds in other models

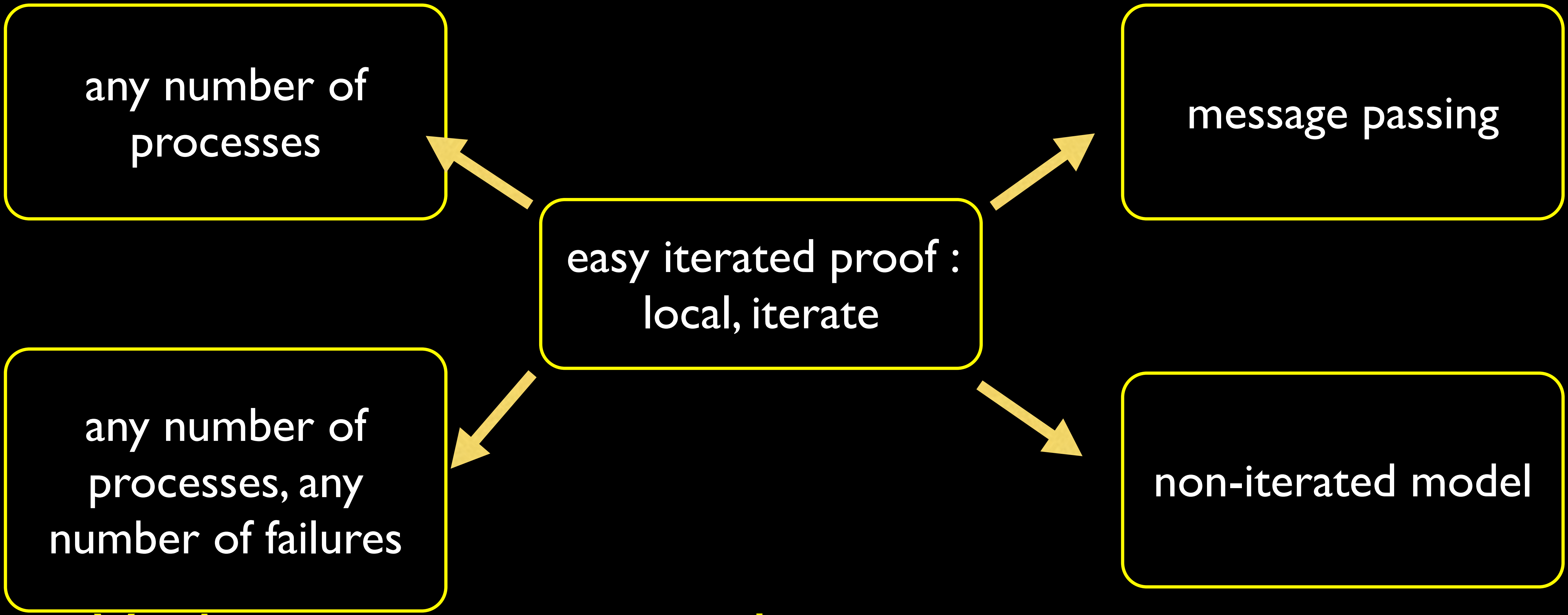

any number of processes
message passing
easy iterated proof : local, iterate
any number of processes, any number of failures
non-iterated model

Iterated approach: theorem holds in other models


any number of processes
message passing
easy iterated proof : local, iterate
any number of processes, any number of failures
non-iterated model

- Via known, generic simulation

Iterated approach: theorem holds in other models


any number of processes
message passing
easy iterated proof : local, iterate
any number of processes, any number of failures
non-iterated model

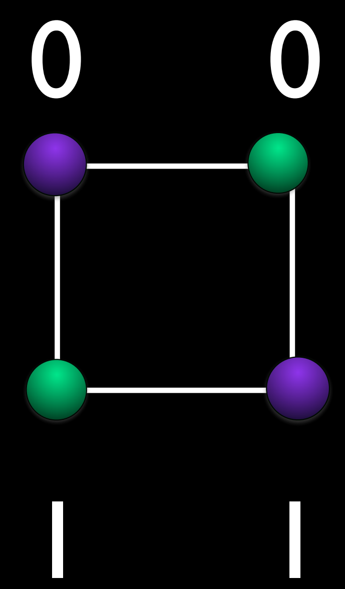
- Via known, generic simulation
- Instead of ad hoc proofs (some known) for each case

implications in terms of

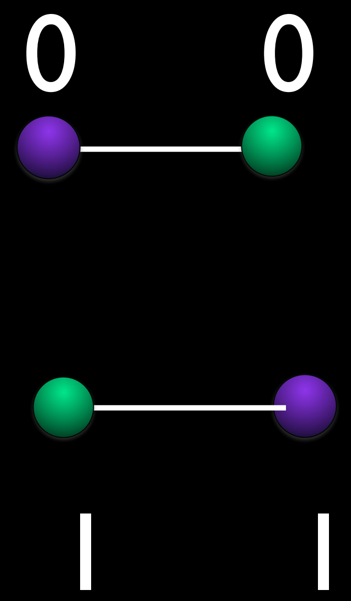
- solvability
- complexity
- computability

Distributed problems

binary consensus



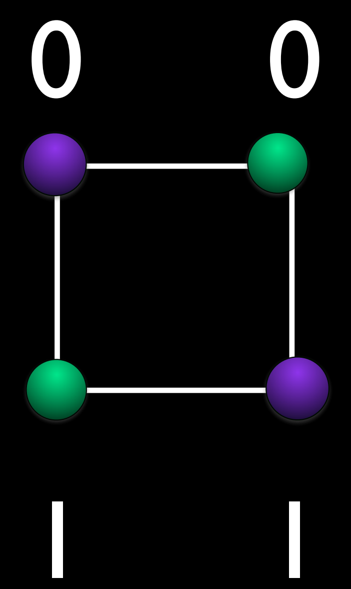
Input Graph



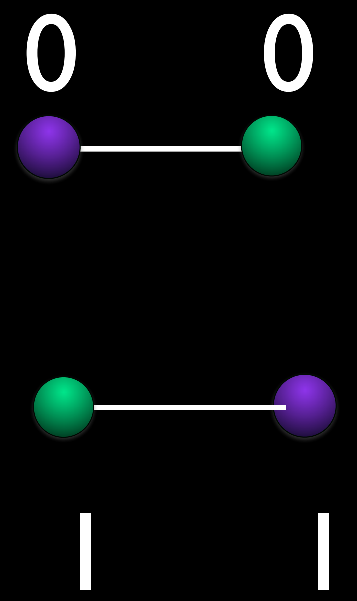
Output Graph

Distributed problems

binary consensus



Input Graph

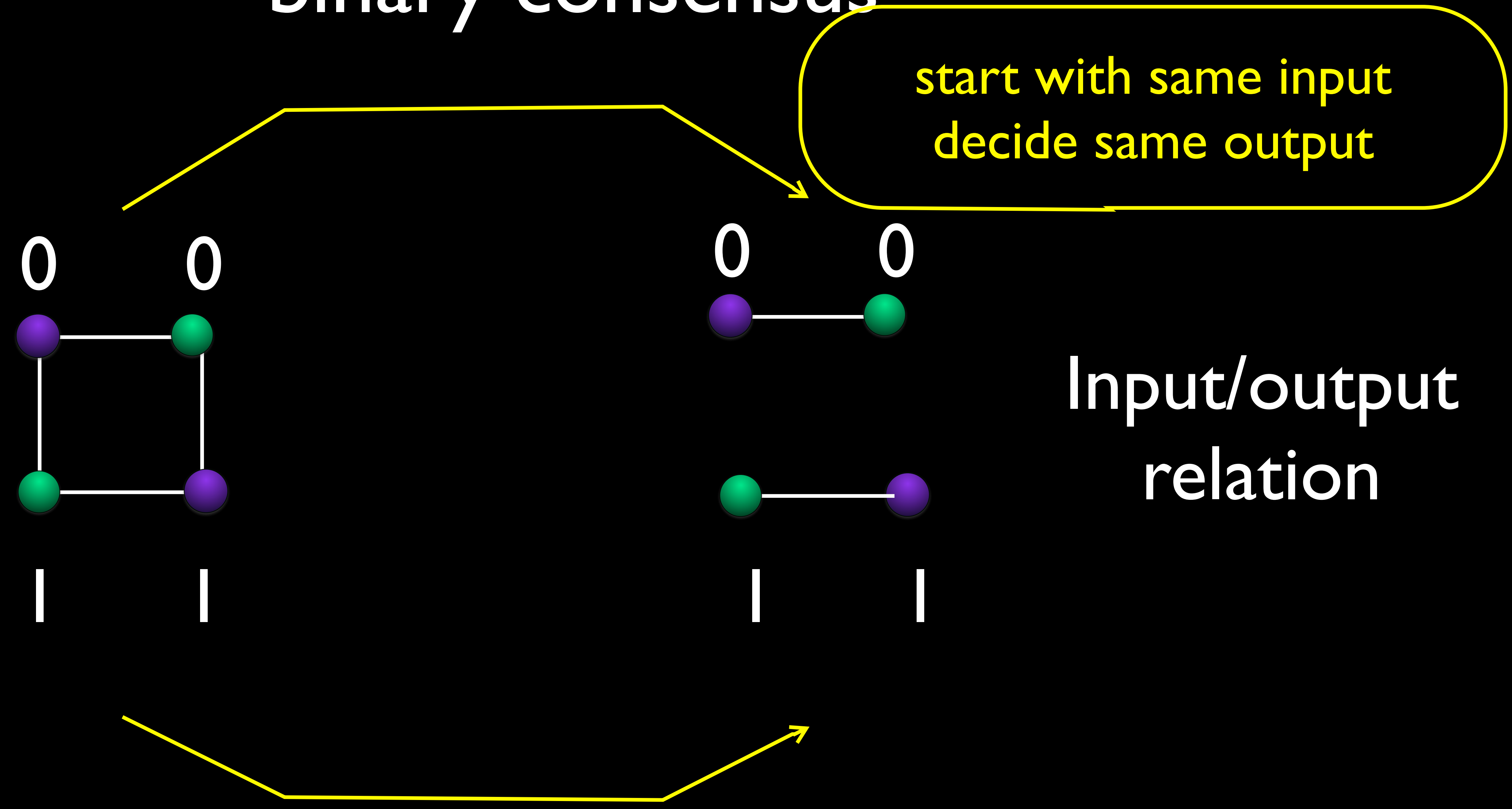


Output Graph

Input/output relation

Distributed problems

binary consensus



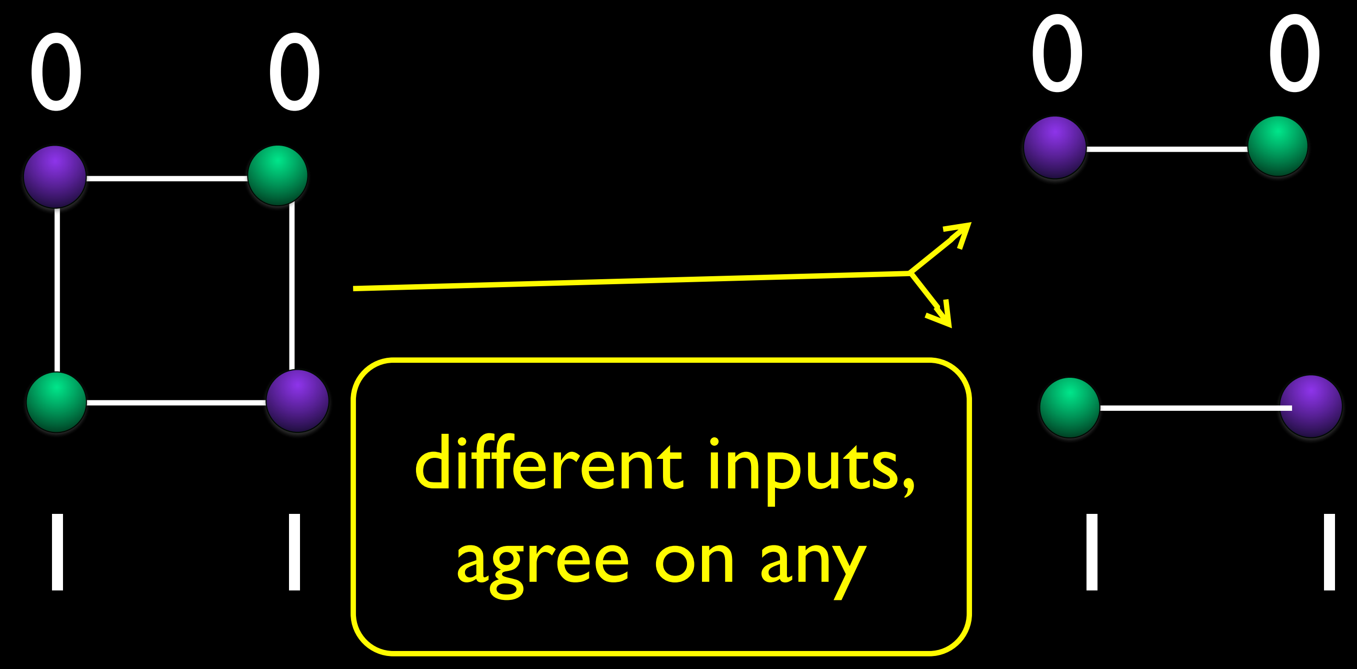
Input/output relation

Input Graph

Output Graph

Distributed problems

binary consensus

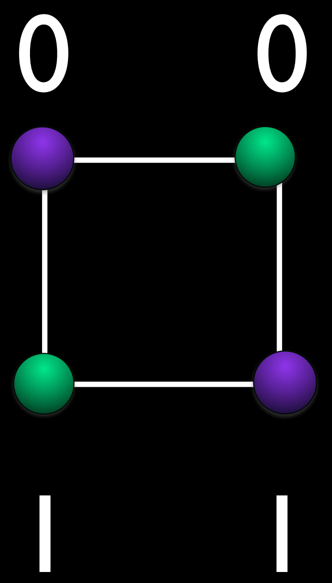


Input/output relation

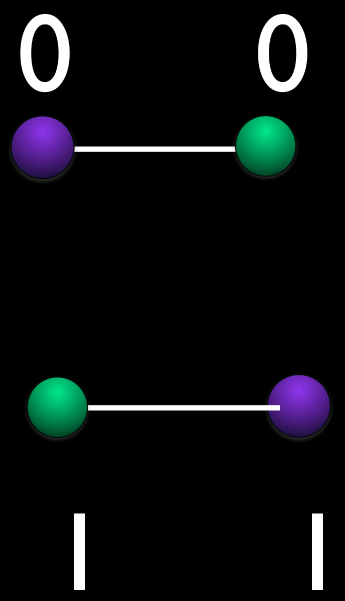
Input Graph

Output Graph

Binary consensus is not solvable due to connectivity

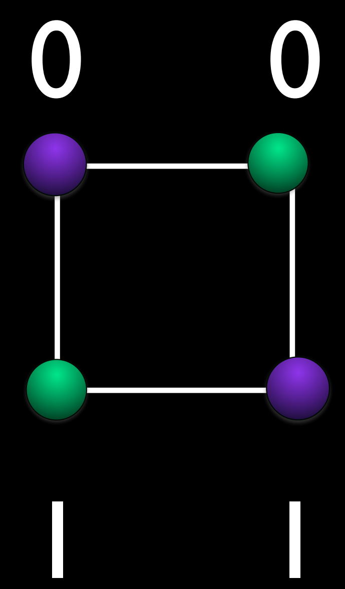


Input Graph

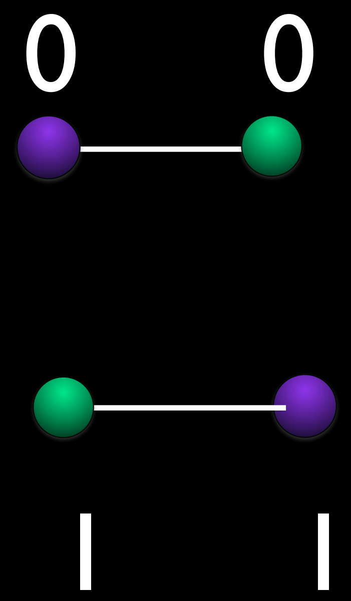


Output Graph

Binary consensus is not solvable due to connectivity



Input Graph

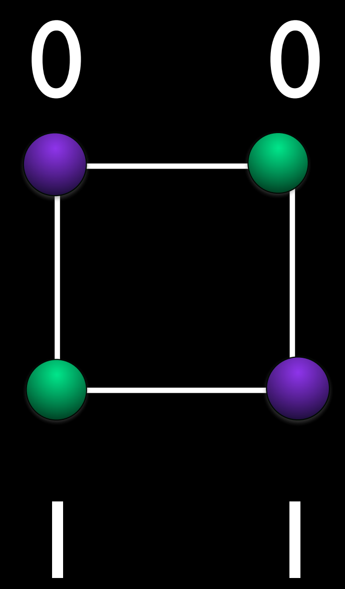


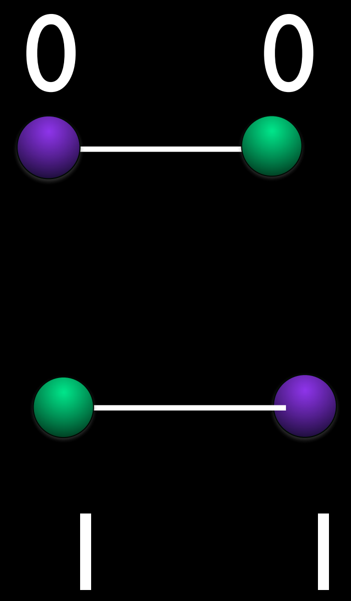
Output Graph

Input/output relation

Binary consensus is not solvable due to connectivity

Each edge is an initial configuration of the protocol





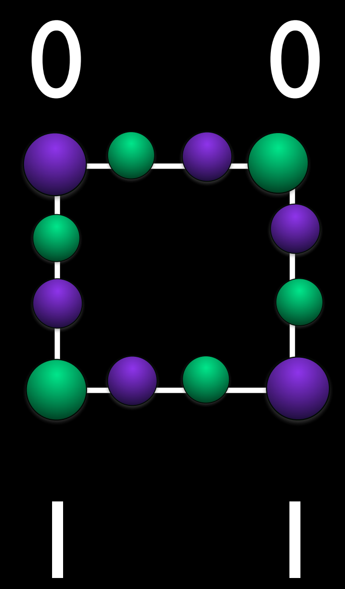
Input/output relation

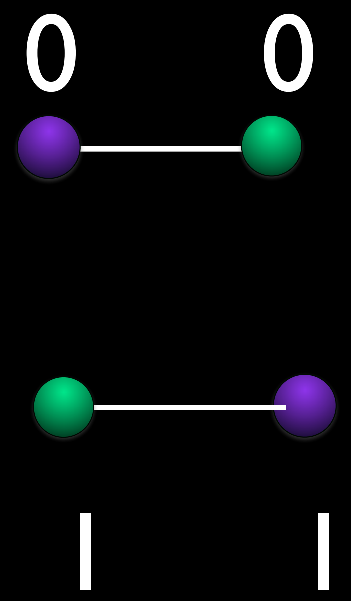
Input Graph

Output Graph

Binary consensus is not solvable due to connectivity

subdivided after 1 round





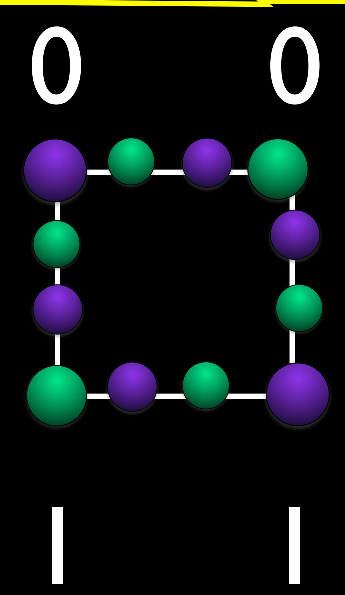
Input/output relation

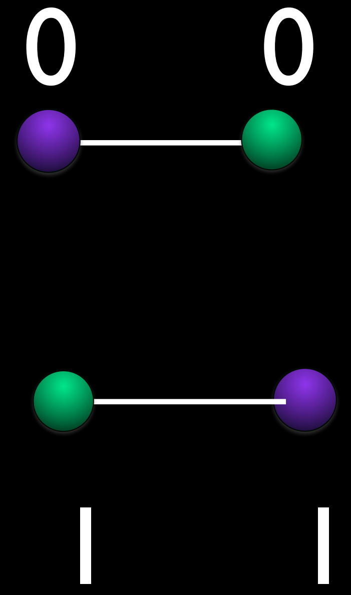
Input Graph

Output Graph

Binary consensus is not solvable due to connectivity

no solution in 1 round



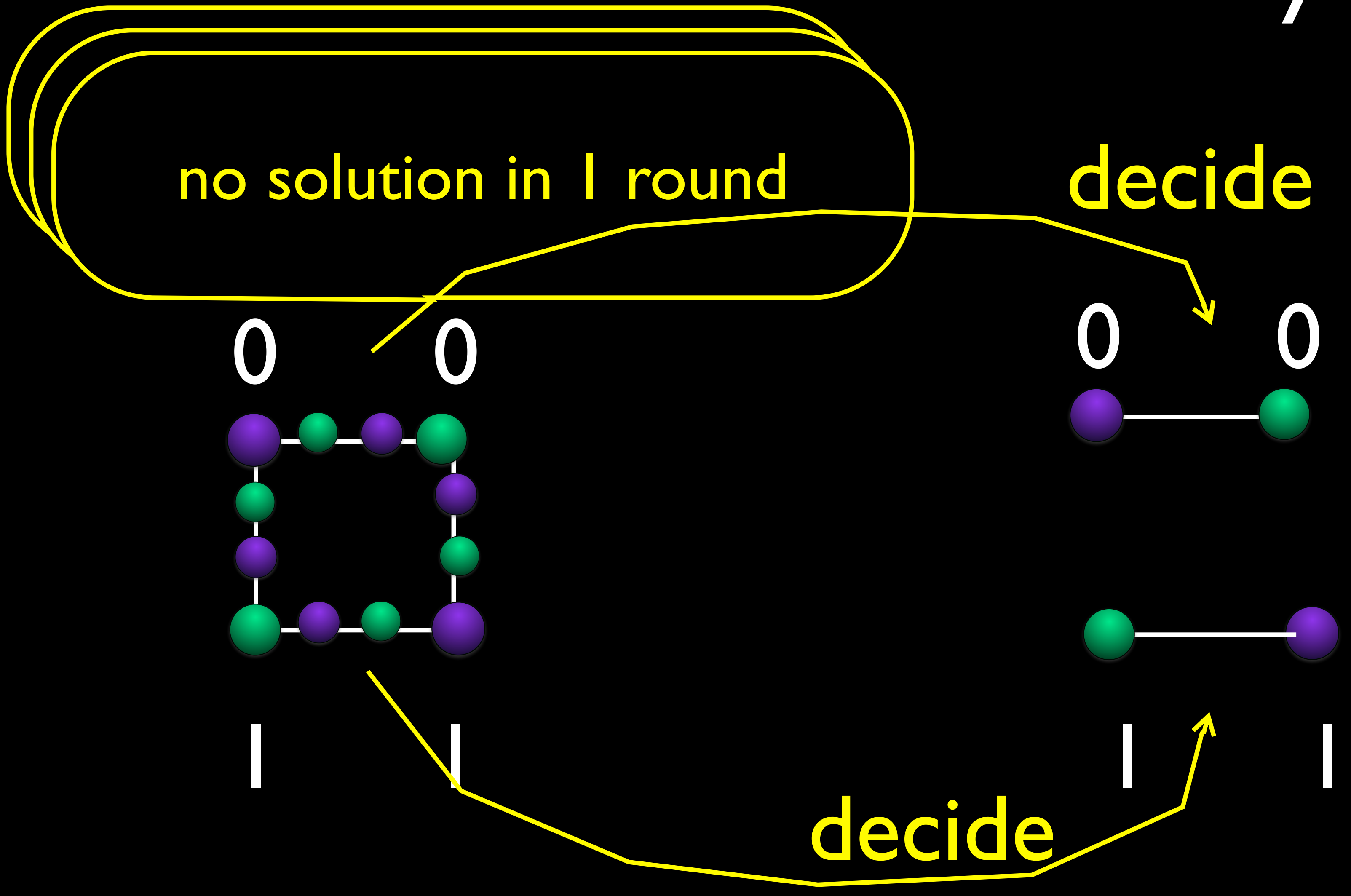


Input/output relation

Input Graph

Output Graph

Binary consensus is not solvable due to connectivity

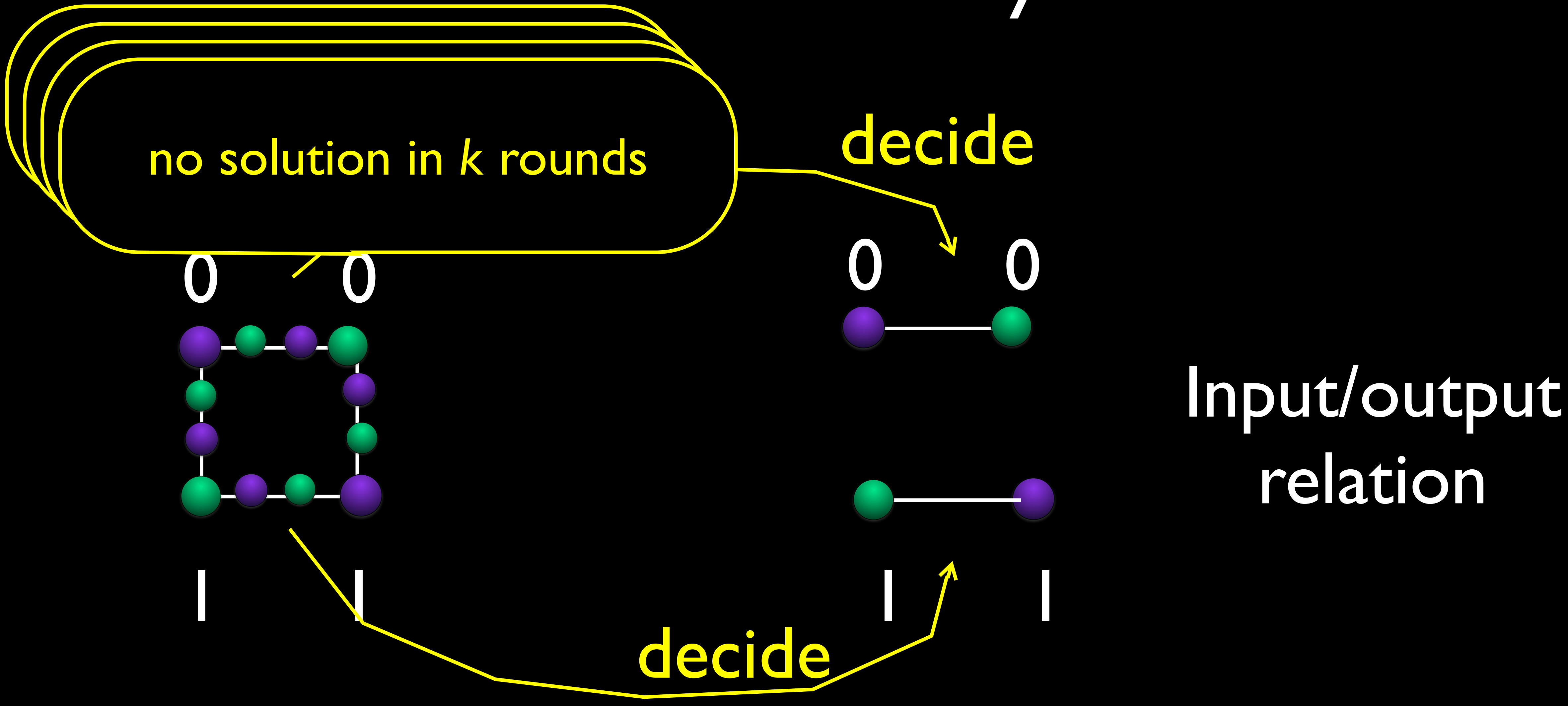


Input/output relation

Input Graph

Output Graph

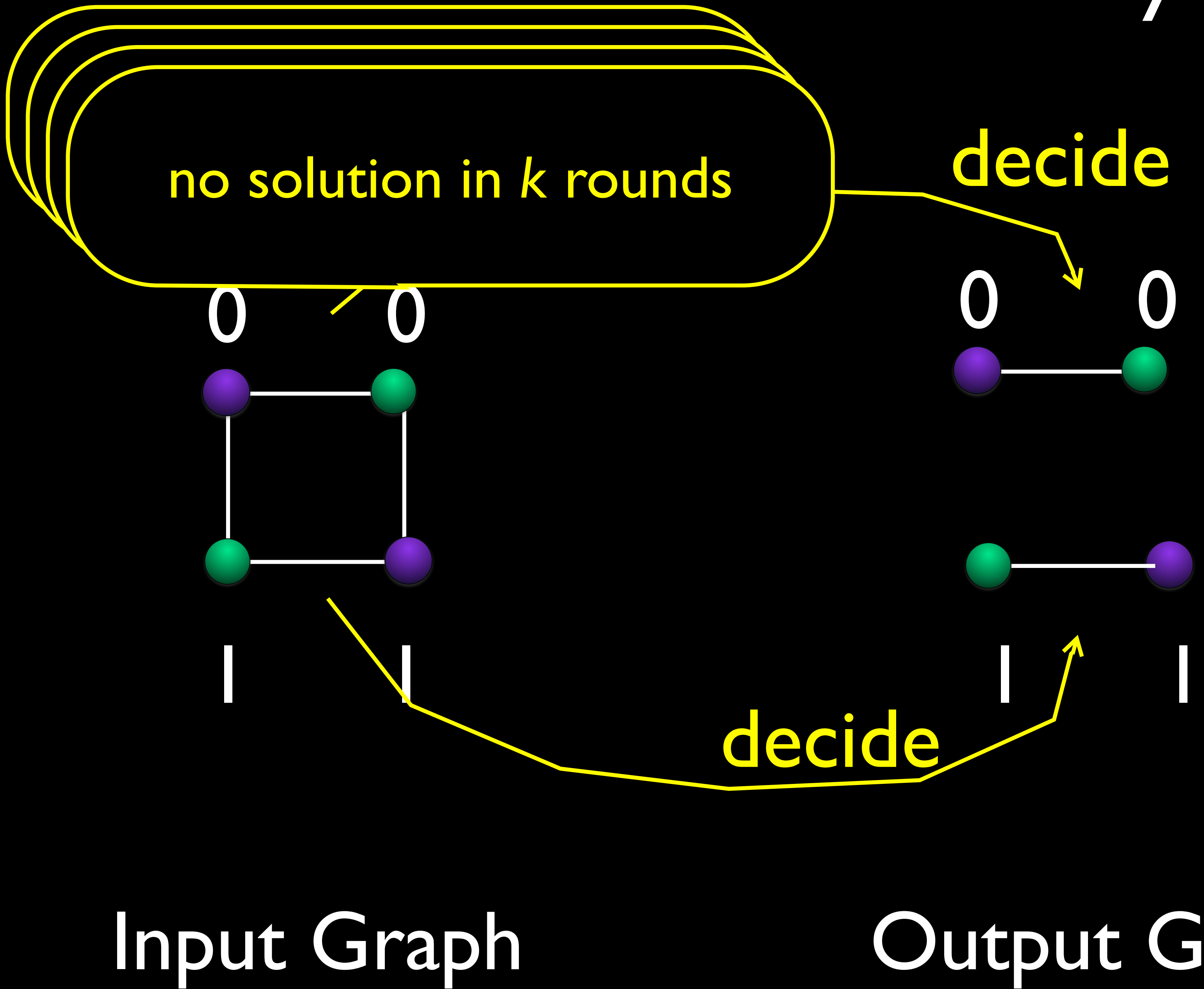
Binary consensus is not solvable due to connectivity



Input Graph

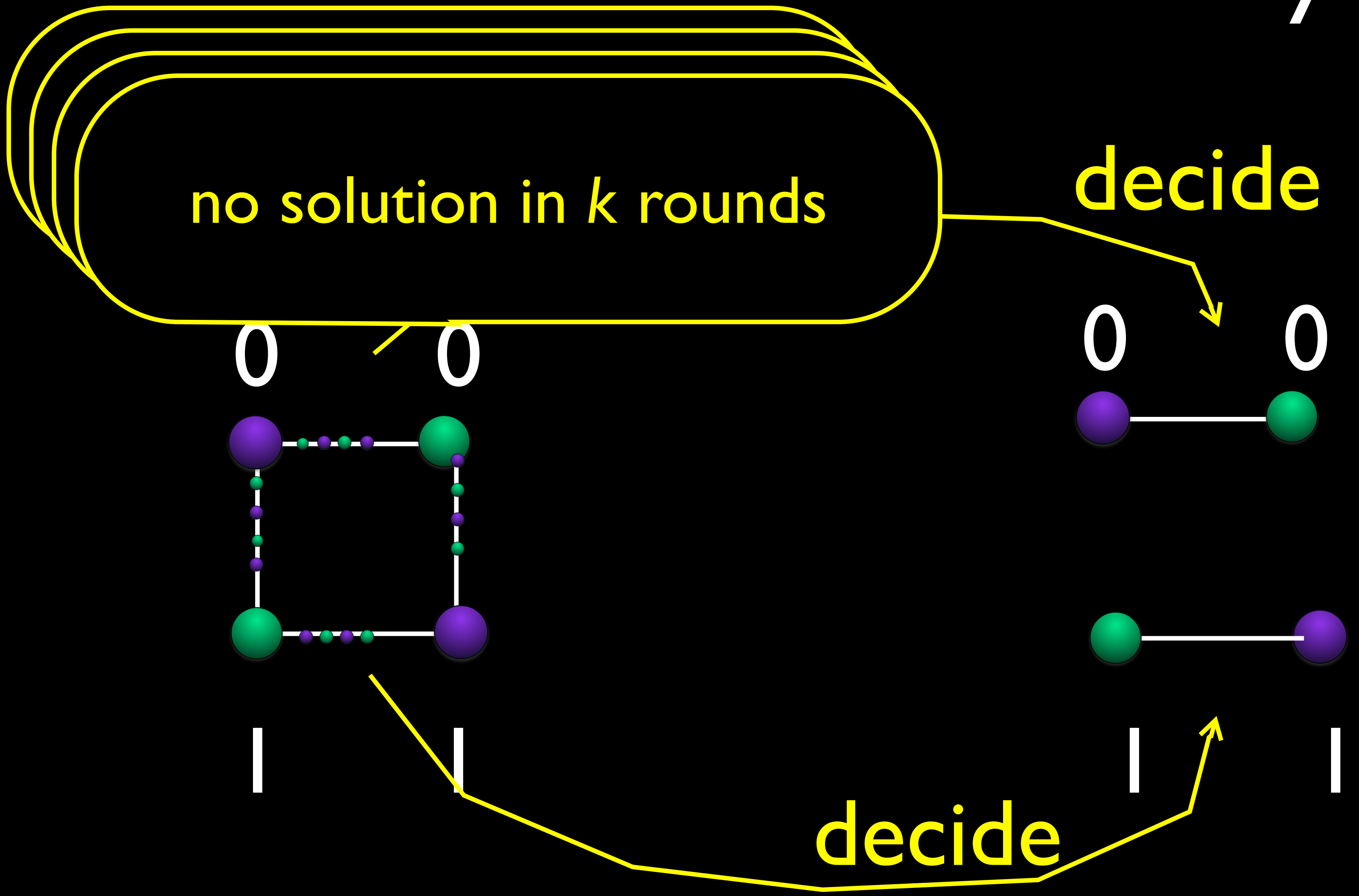
Output Graph

Binary consensus is not solvable due to connectivity



Input/output relation

Binary consensus is not solvable due to connectivity



Input/output relation

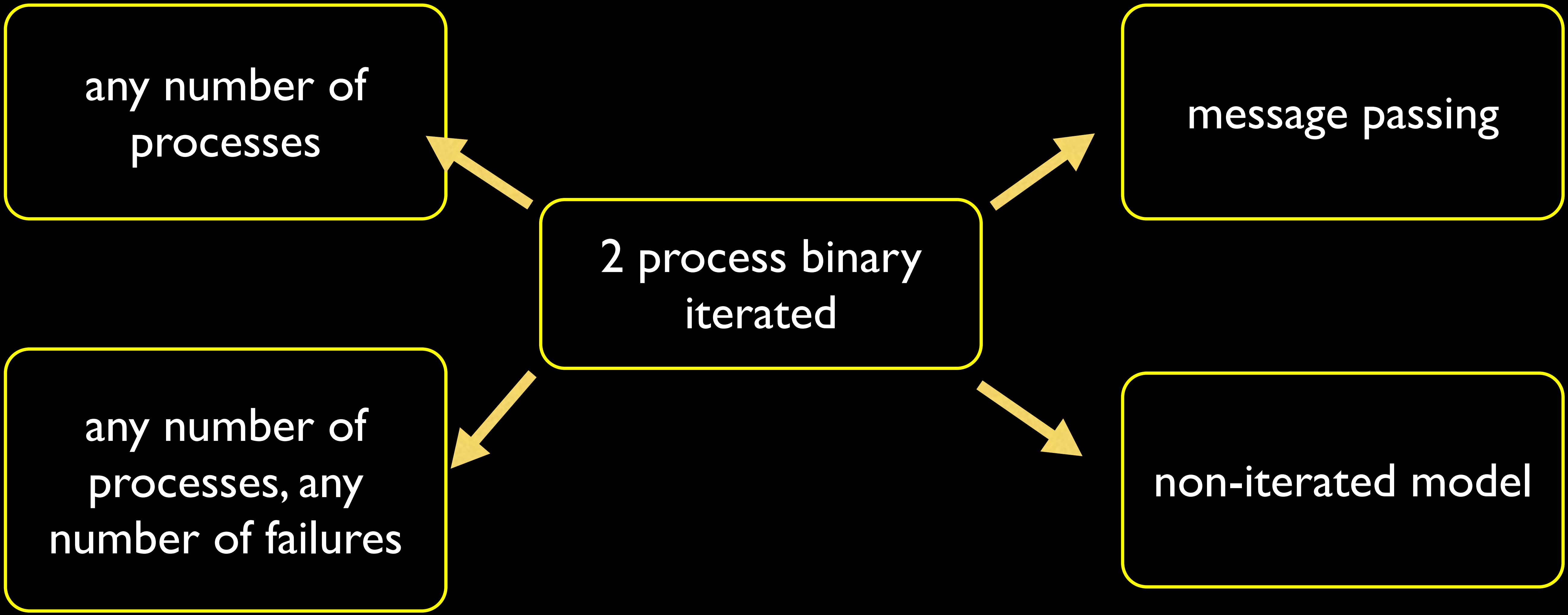
Input Graph

Output Graph

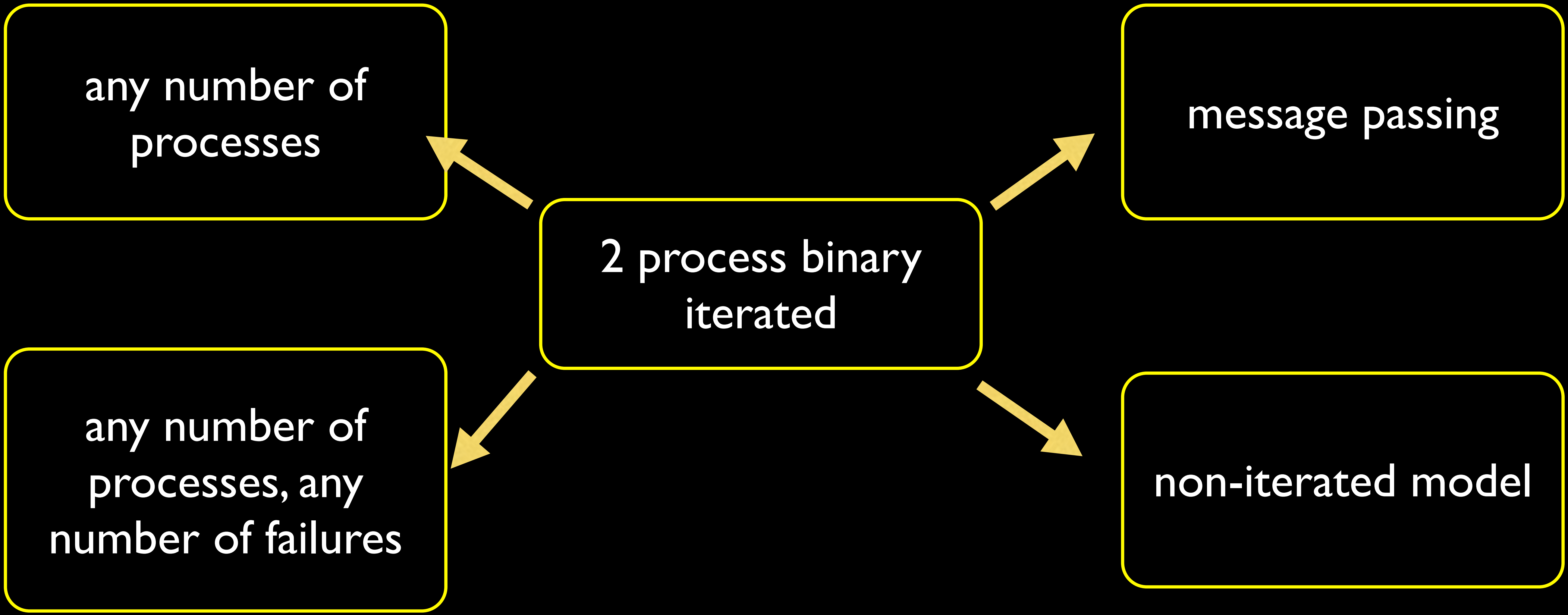
corollaries:

consensus impossible in the iterated model

consensus impossibility holds in other models

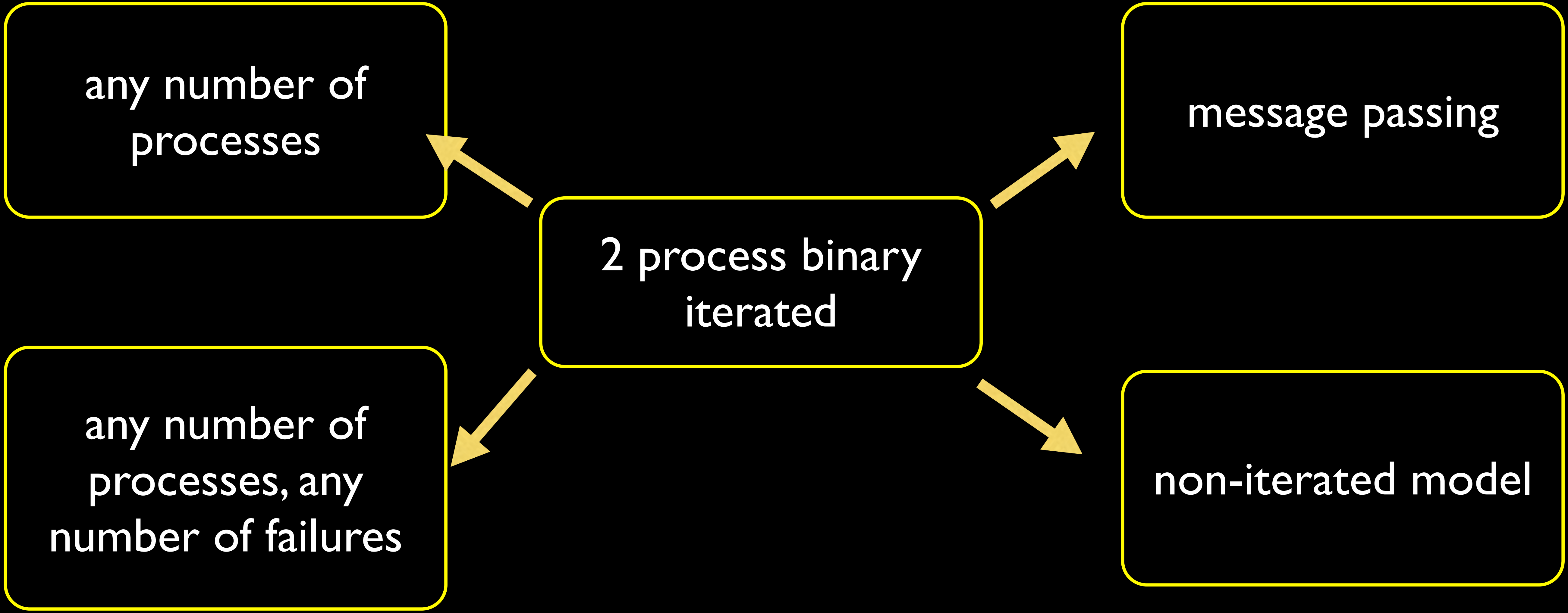

2 process binary iterated
any number of processes
message passing
any number of processes, any number of failures
non-iterated model

consensus impossibility holds in other models


any number of processes
message passing
2 process binary iterated
any number of processes, any number of failures
non-iterated model

- Via known, generic simulation

consensus impossibility holds in other models


any number of processes
message passing
2 process binary iterated
any number of processes, any number of failures
non-iterated model

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

Decidability

Decidability

- Given a task for 2 processes, is it solvable in the iterated model?

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

Decidability

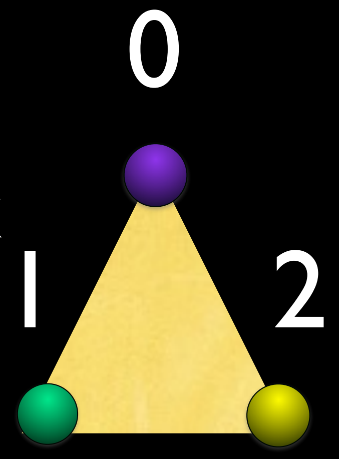
- Given a task for 2 processes, is it solvable in the iterated model?
- Yes, there is an algorithm to decide: a graph connectivity problem
- Then extend result to other models , via generic simulations, instead of ad hoc proofs

Beyond 2 processes

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

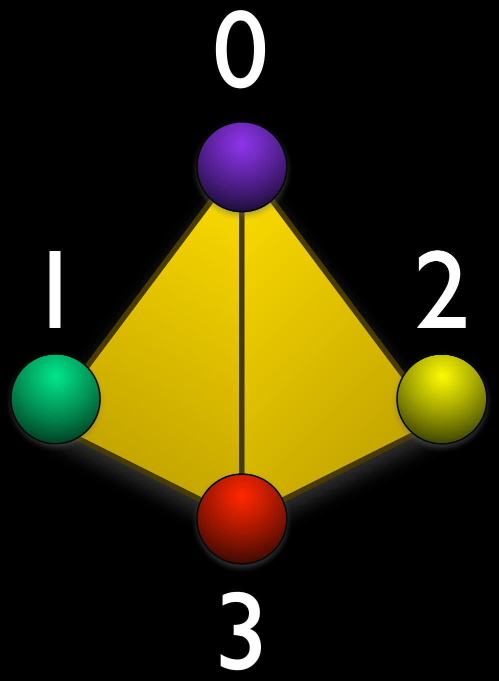
2-dim simplex

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



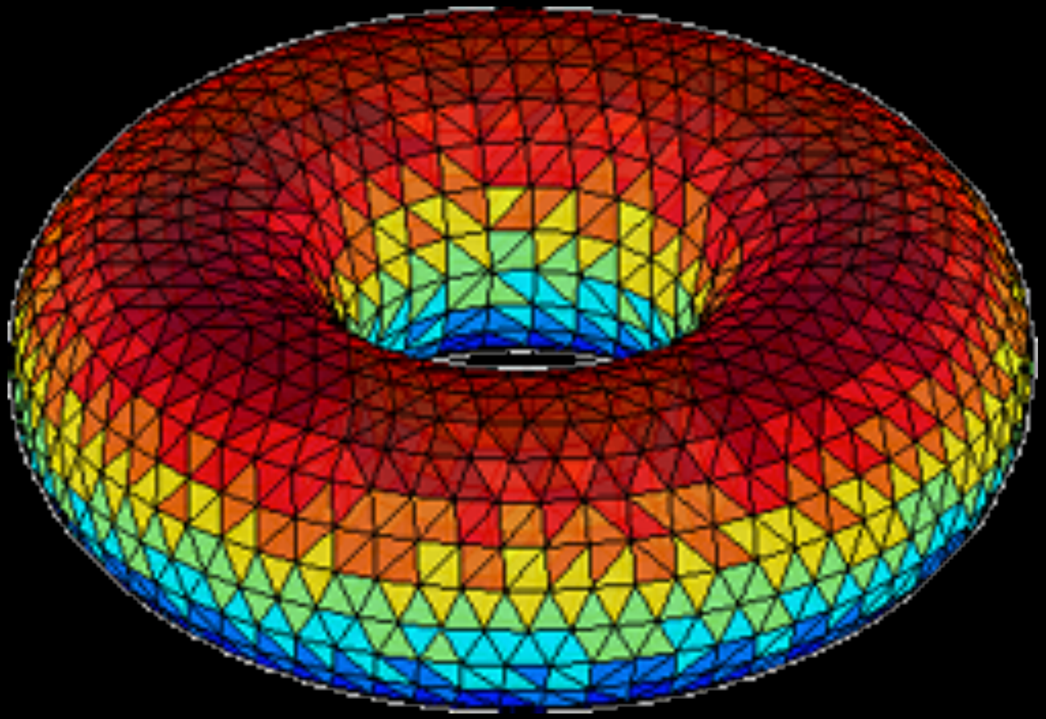
3-dim simplex

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

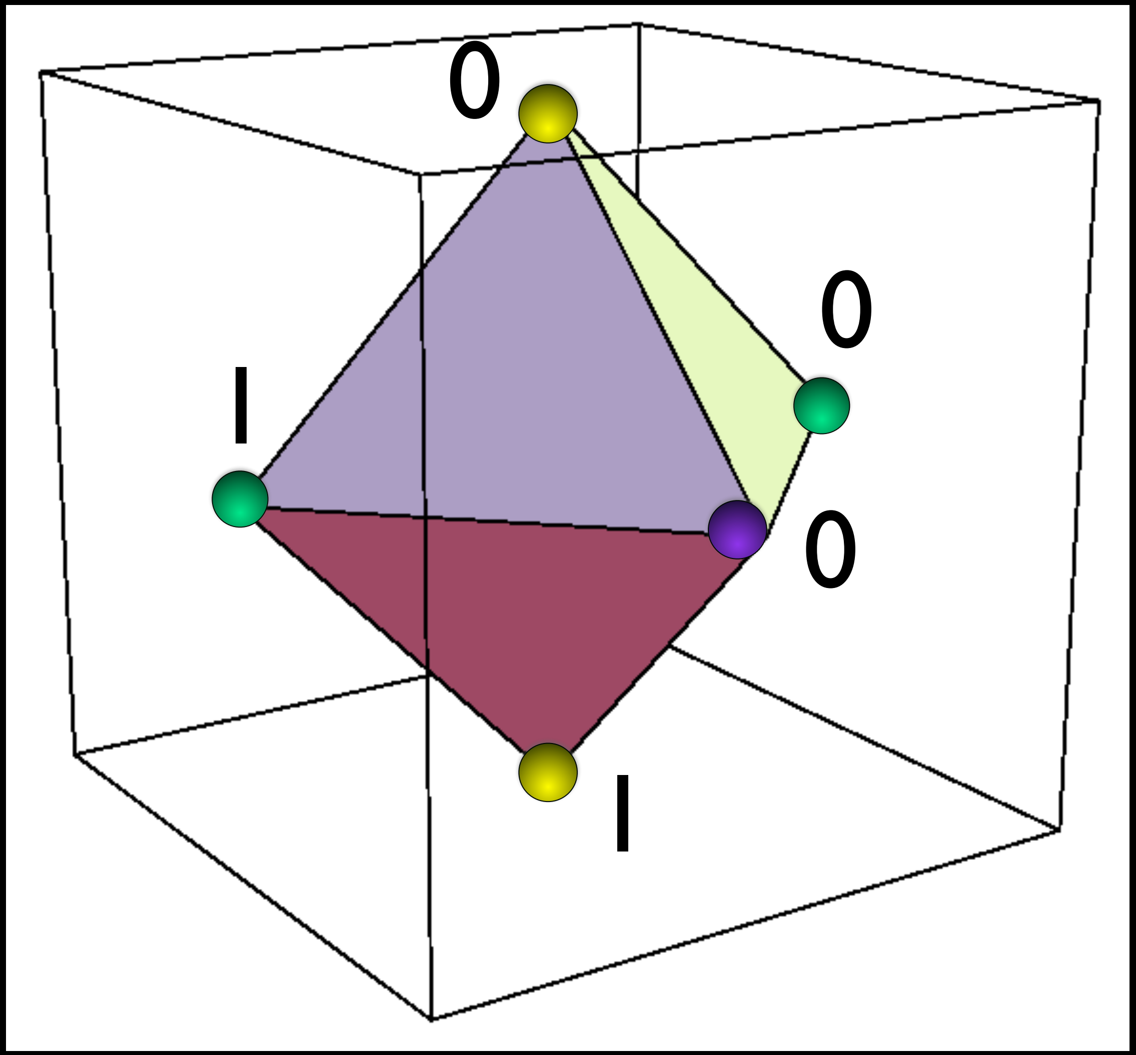


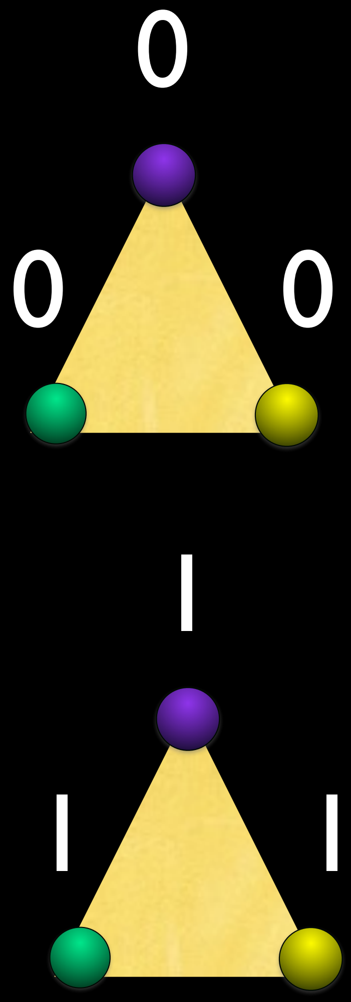
complexes

Collection of simplexes closed under containment



consensus task
3 processes



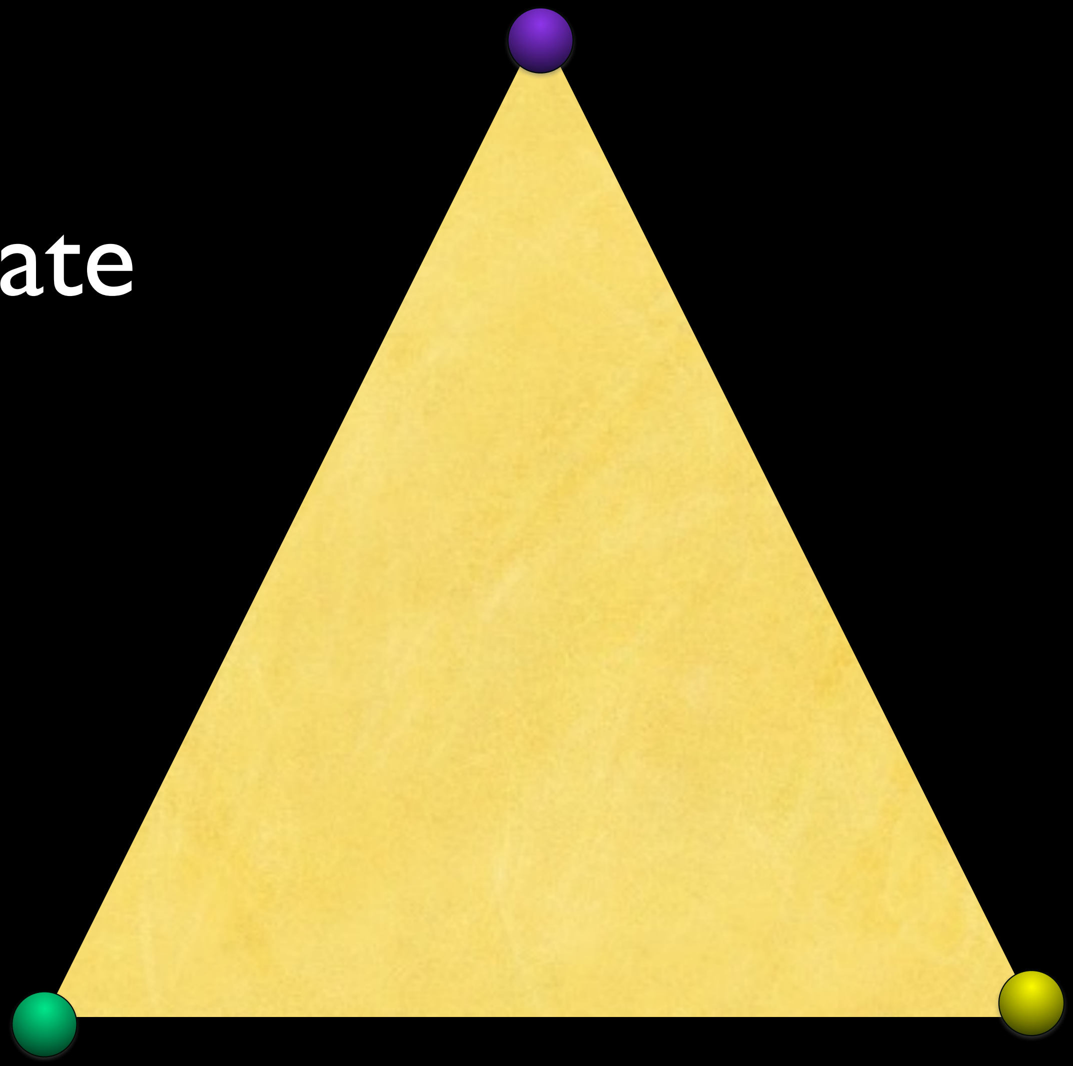


Input Complex

Output Complex

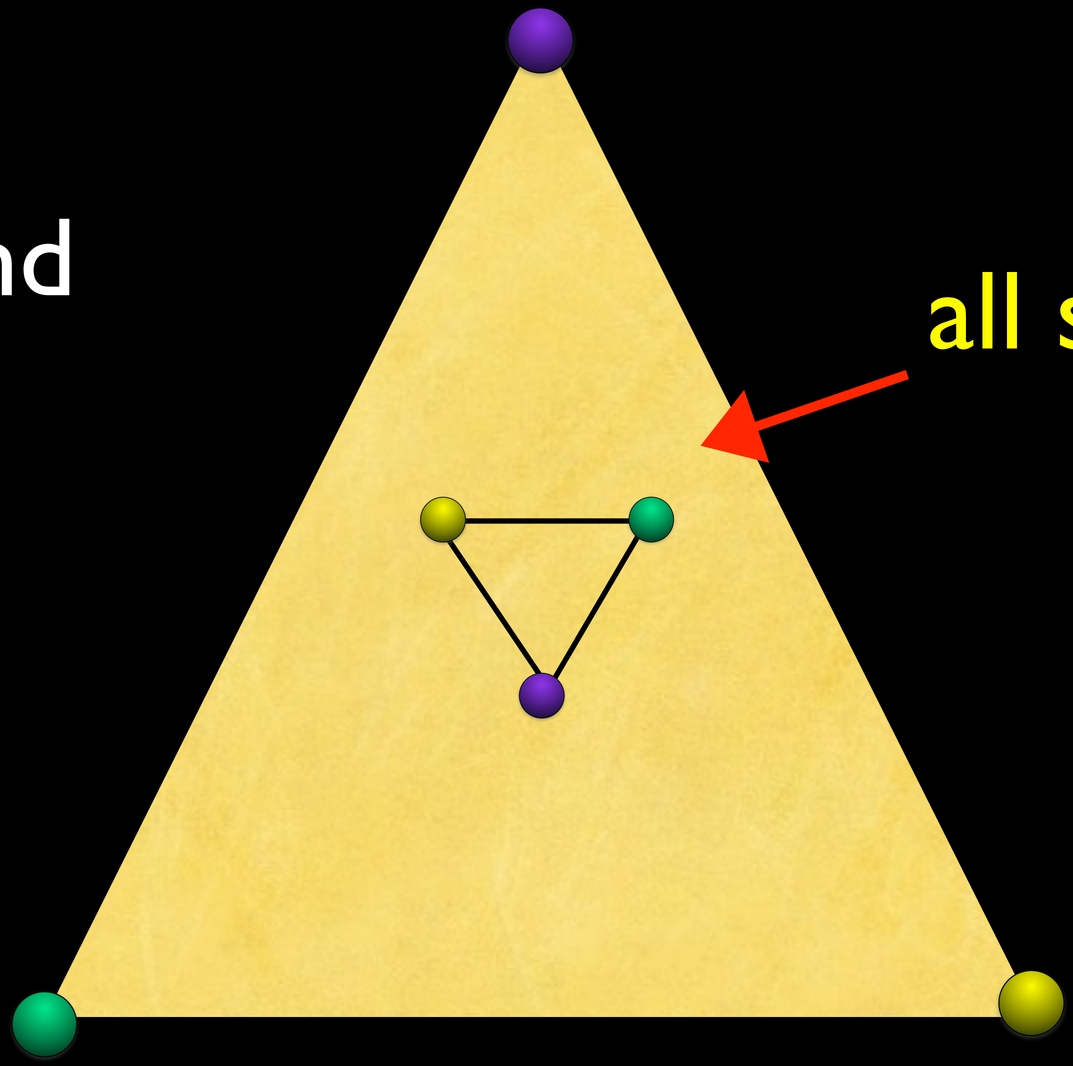
Iterated model

One initial state



Iterated model

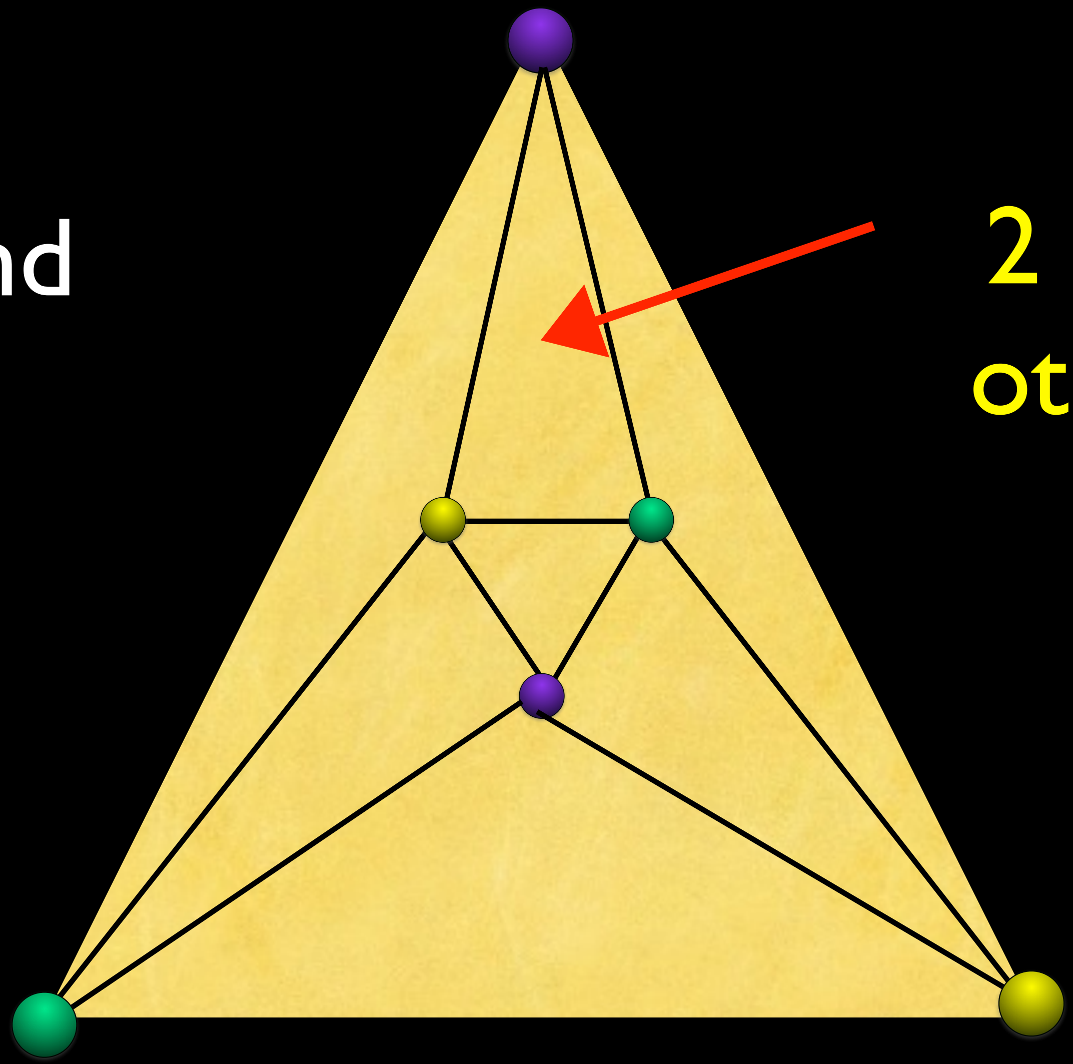
after 1 round



all see each other

Iterated model

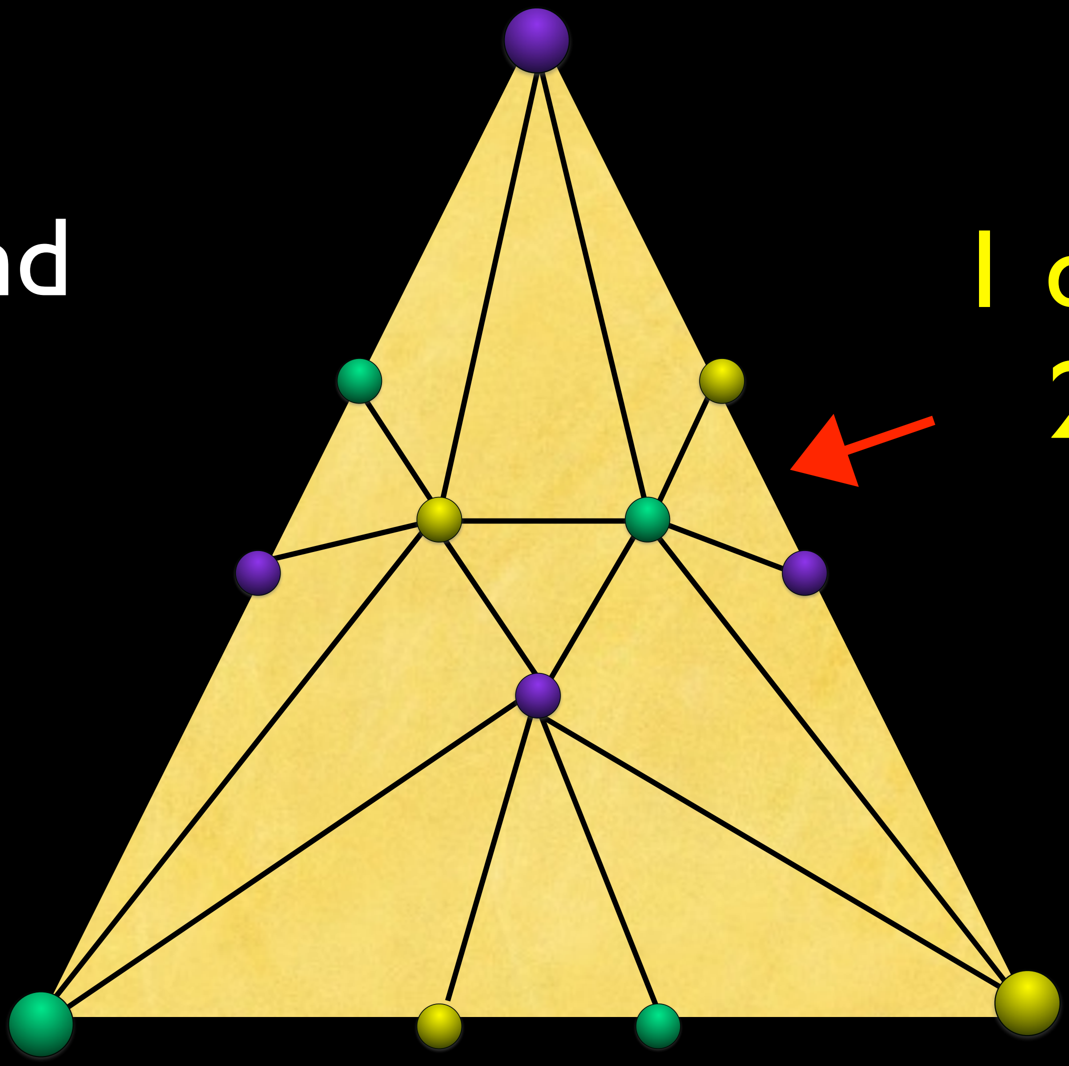
after 1 round



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

Decidability

- Given a task for 3 processes, is it solvable in the iterated model?

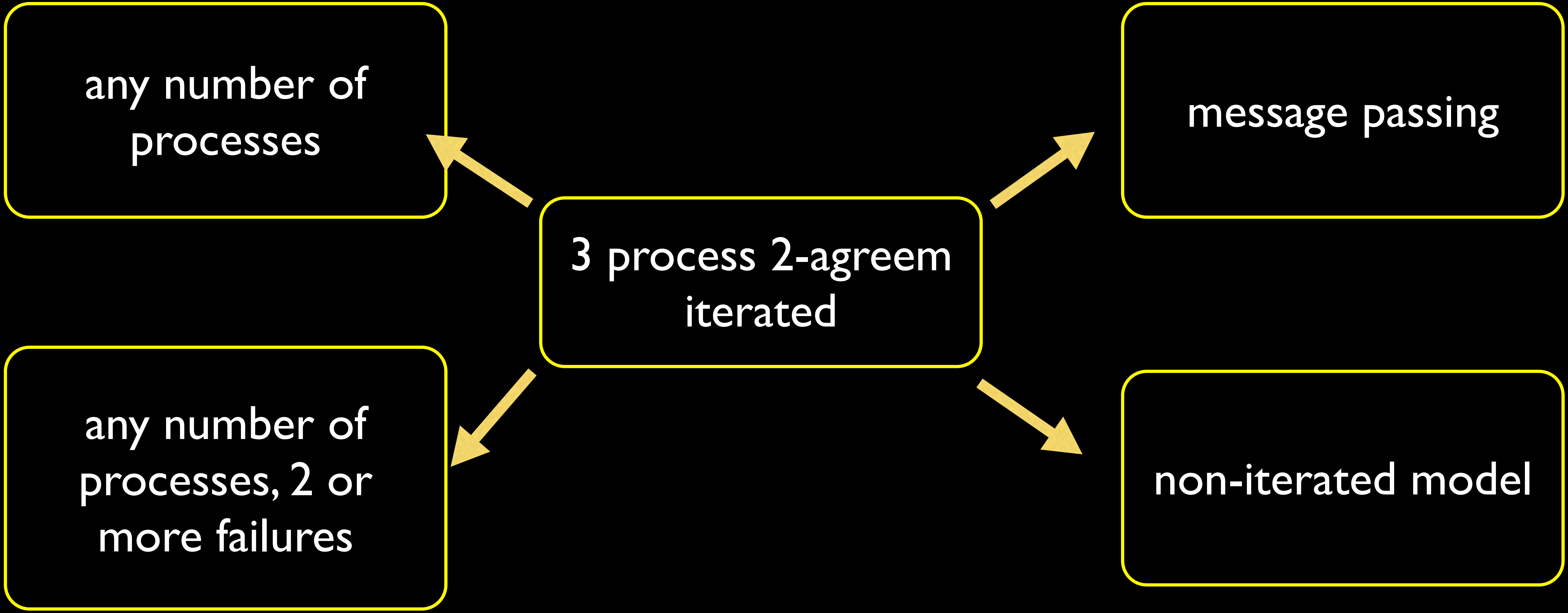
Decidability

- 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

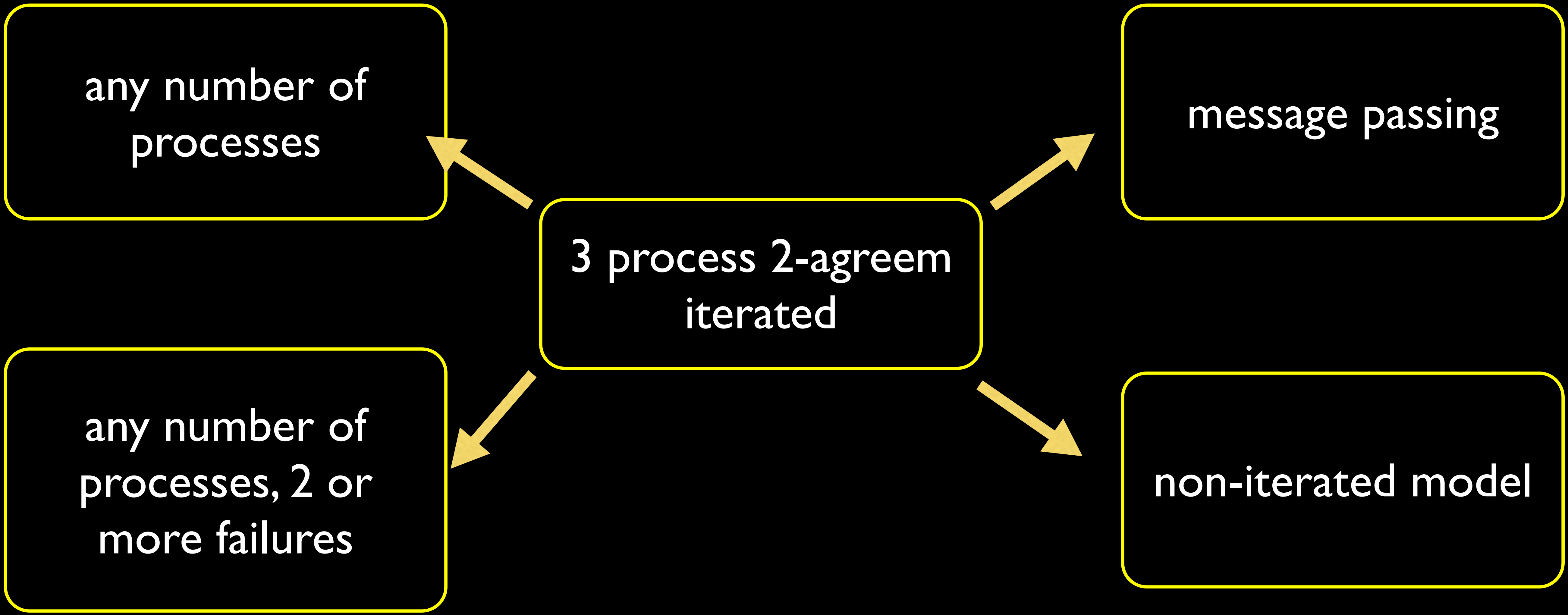
Decidability

- 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

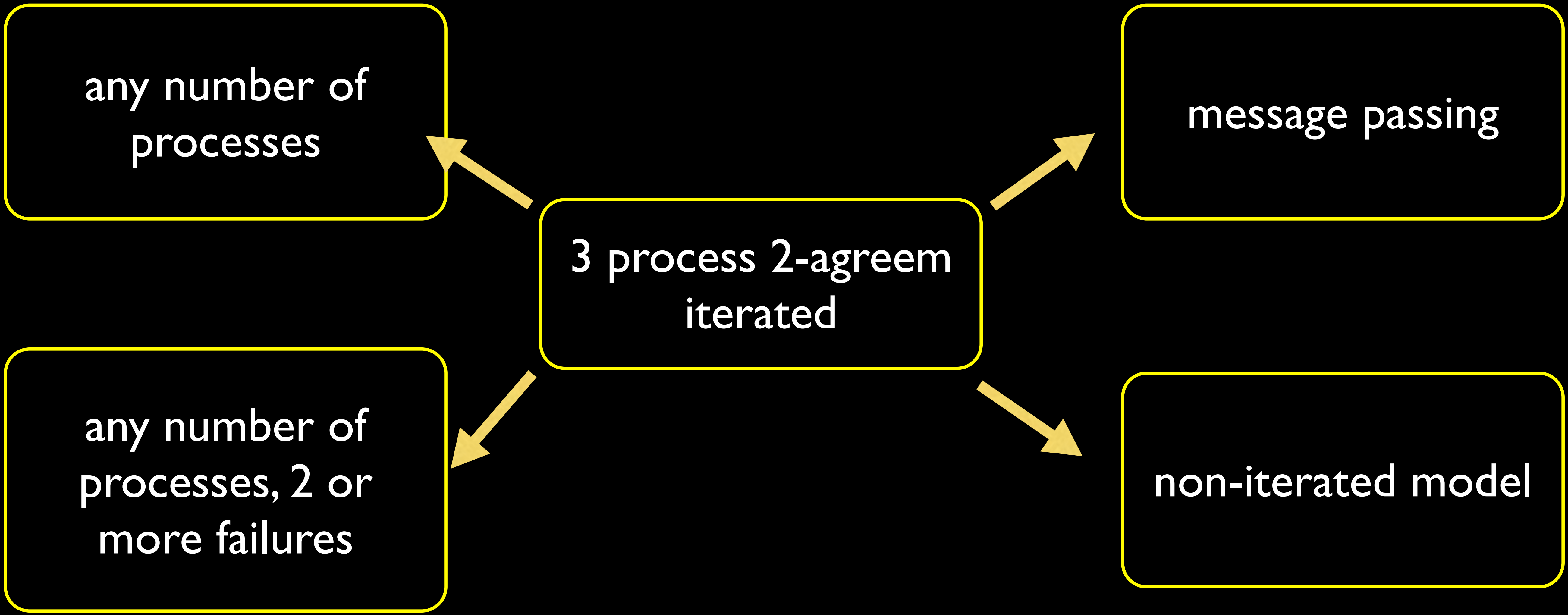

any number of processes
message passing
3 process 2-agreem iterated
any number of processes, 2 or more failures
non-iterated model

Extension to other models


any number of processes
message passing
3 process 2-agreem iterated
any number of processes, 2 or more failures
non-iterated model

- Via known, generic simulation

Extension to other models


any number of processes
message passing
3 process 2-agreem iterated
any number of processes, 2 or more failures
non-iterated model

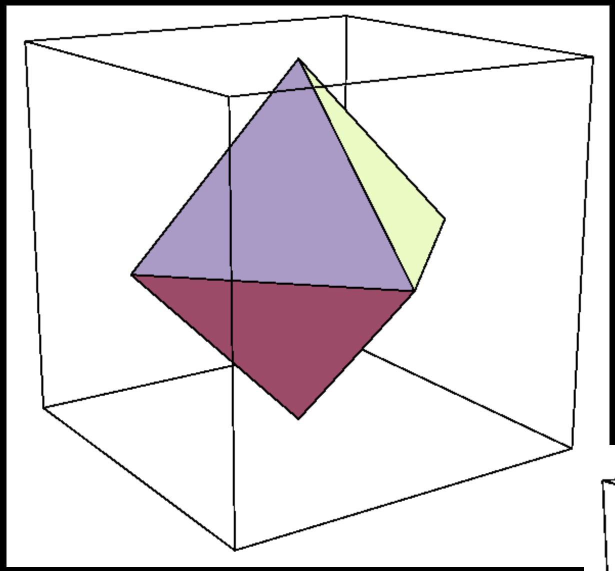
- Via known, generic simulation
- Instead of ad hoc proofs for each case

Conclusions

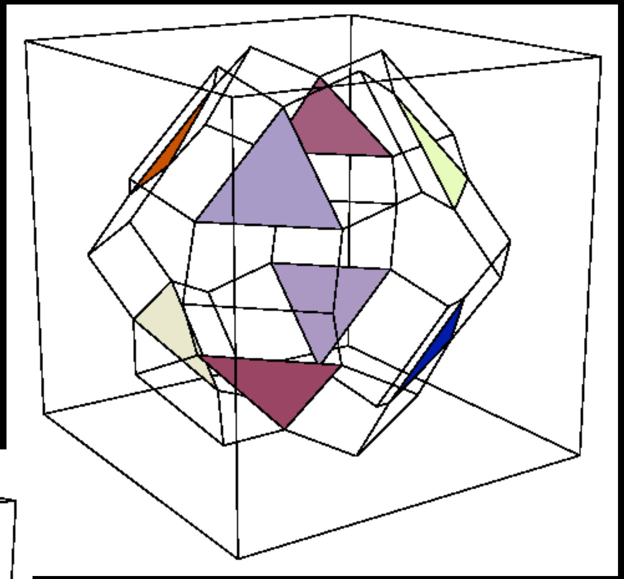
Conclusions

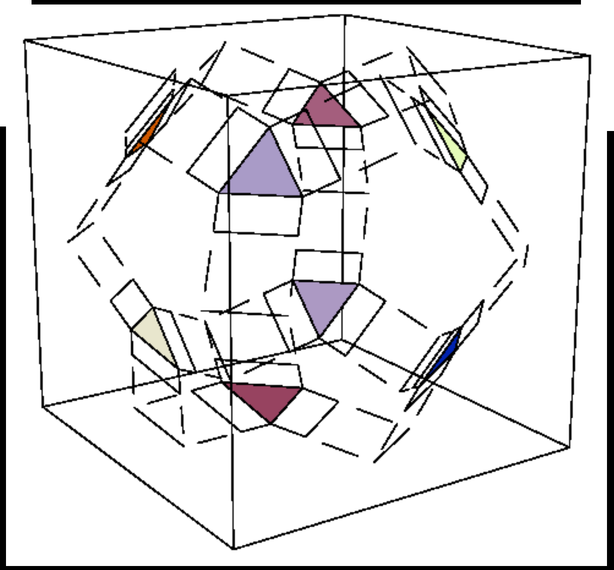
- In distributed computing there are too many different issues of interest, no single model can capture them all

Synchronous protocol complex evolution



Connected but not 1-connected





Conclusions

Conclusions

Conclusions

- But the iterated model (with extensions not discussed here) captures essential distributed computing aspects

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