

Why Extension-based Proofs Fail

Rati Gelashvili

NeuralMagic

Based on joint works with:

Dan Alistarh, James Aspnes, Faith Ellen, Leqi Zhu

Faith Ellen, Leqi Zhu



Outline

The Model: Asynchronous Shared Memory

A Story Of Two Famous Results:

- Impossibility of Consensus
 - [Fischer, Lynch, Paterson '85]: Dijkstra Prize 2001
- Impossibility of k-Set Agreement
 - [Herlihy, Shavit '99], [Saks, Zaharoglou' 00]: Gödel Prize 2004
 - [Borowsky, Gafni '93]: Dijkstra Prize 2017

“Conventional”
vs
“Topological”

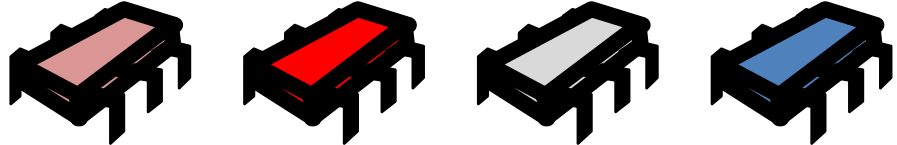

Extension-Based Proof System

- Why did conventional techniques fail?

Putting New Insights to Use

- Revisionist simulations: a new type of memory lower bound

Asynchronous Shared Memory

- n processors 
 - take steps at arbitrary speeds, may crash
- shared memory locations 
- each step is an access to a shared memory location
 - using a synchronization instruction, e.g. `read()`, `write(x)`

Consensus

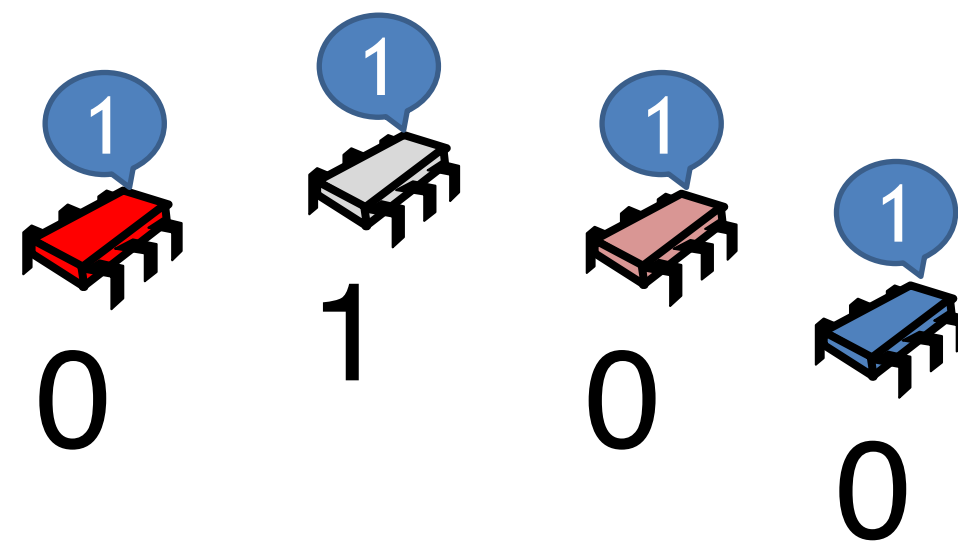
Each process get a binary input, must return a binary output

Agreement

- all outputs must be the same

Validity

- the output value must be an input to one of the processes



k-Set Agreement

Each process get an input in $[0,k]$, must return an output in $[0,k]$

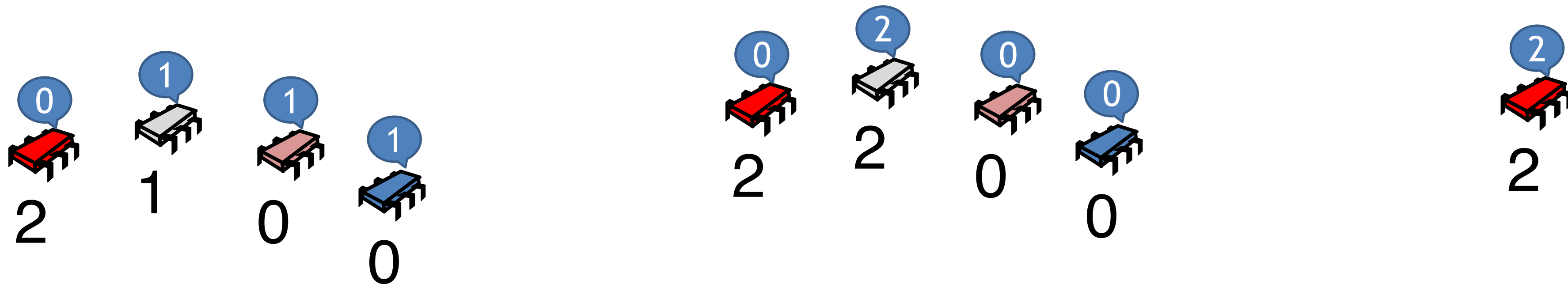
Agreement

- at most k different outputs may be returned

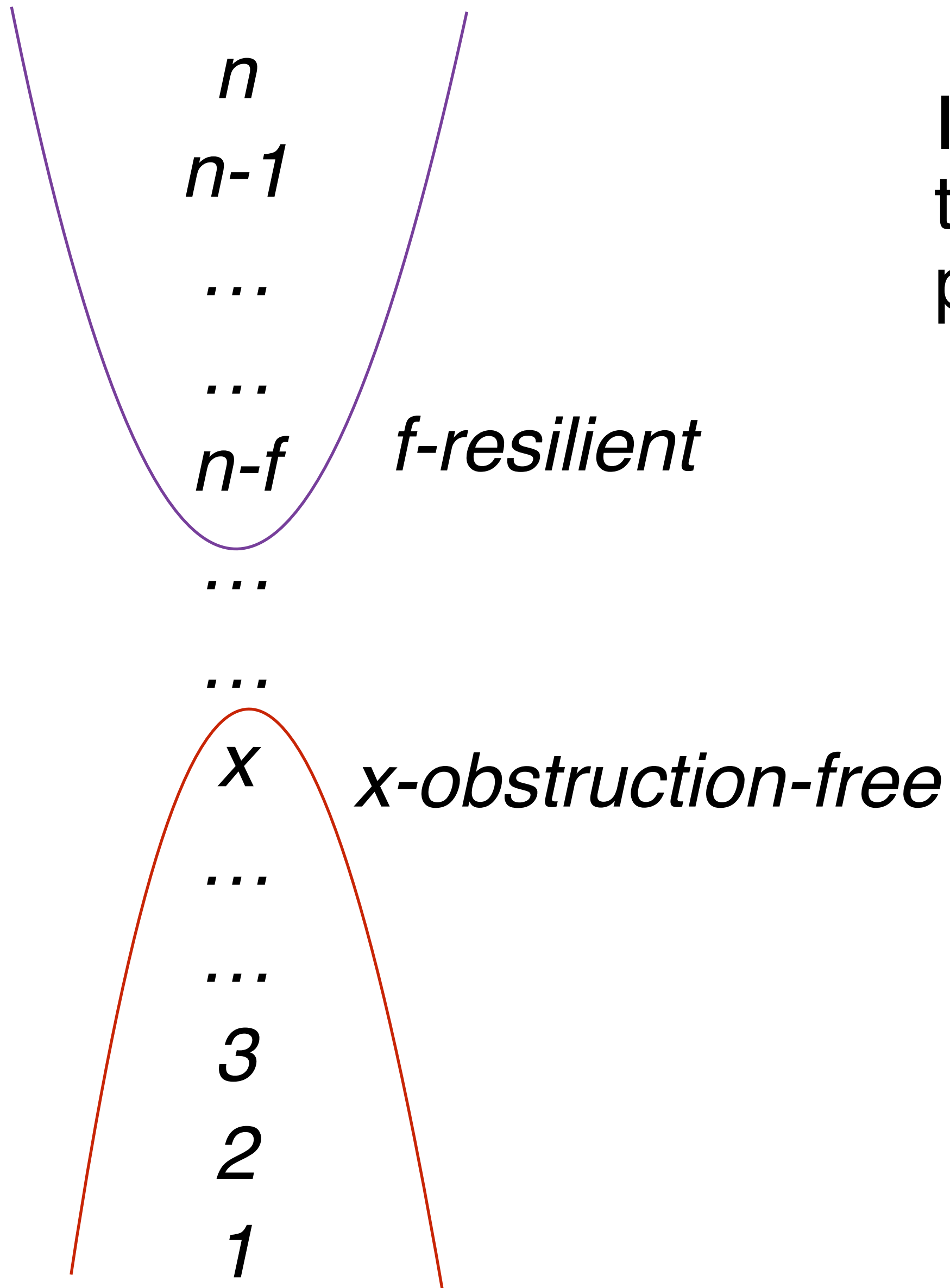
Validity

- each output value must be an input to one of the processes

Equivalent to Consensus when $k=1$



Termination Conditions



In executions where the number of processes that keep taking steps satisfies a condition, a process must terminate

wait-free

- equivalent to $(n-1)$ -resilient and n -obstruction-free
- a process must return after finitely many steps

obstruction-free

- same as 1-obstruction-free
- a process is required to return only after taking finitely many consecutive steps

1-resilient

- a process is required to return only if at least $(n-1)$ processes keep taking steps (at most 1 process has failed)

FLP Result

Statement: no algorithm can solve consensus in a 1-resilient way!

Proof idea:

Such an algorithm would have to admit an execution in which all processes take infinitely many steps and do not terminate.

This would contradict 1-resiliency!

FLP Proof: Valency

A *configuration* describes a global state at some point during algorithm execution: states of all processes and contents of all registers

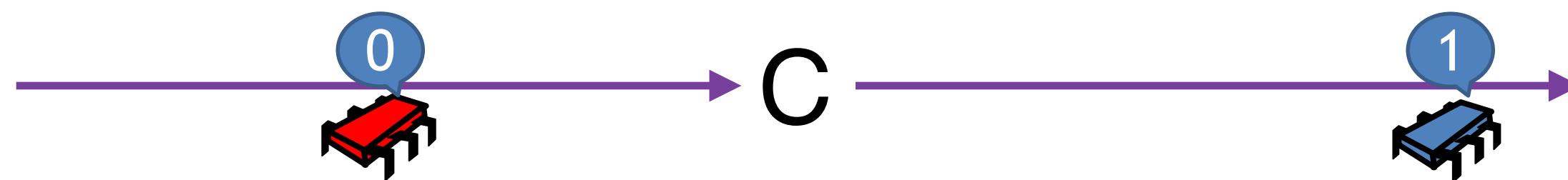
Valency of a configuration C :

C is *0-valent* if in some execution 0 is returned.

C is *1-valent* if in some execution 1 is returned.

C is *bivalent* if C is both 0-valent and 1-valent

If C is bivalent, no process may have already returned



FLP Proof: Extensions

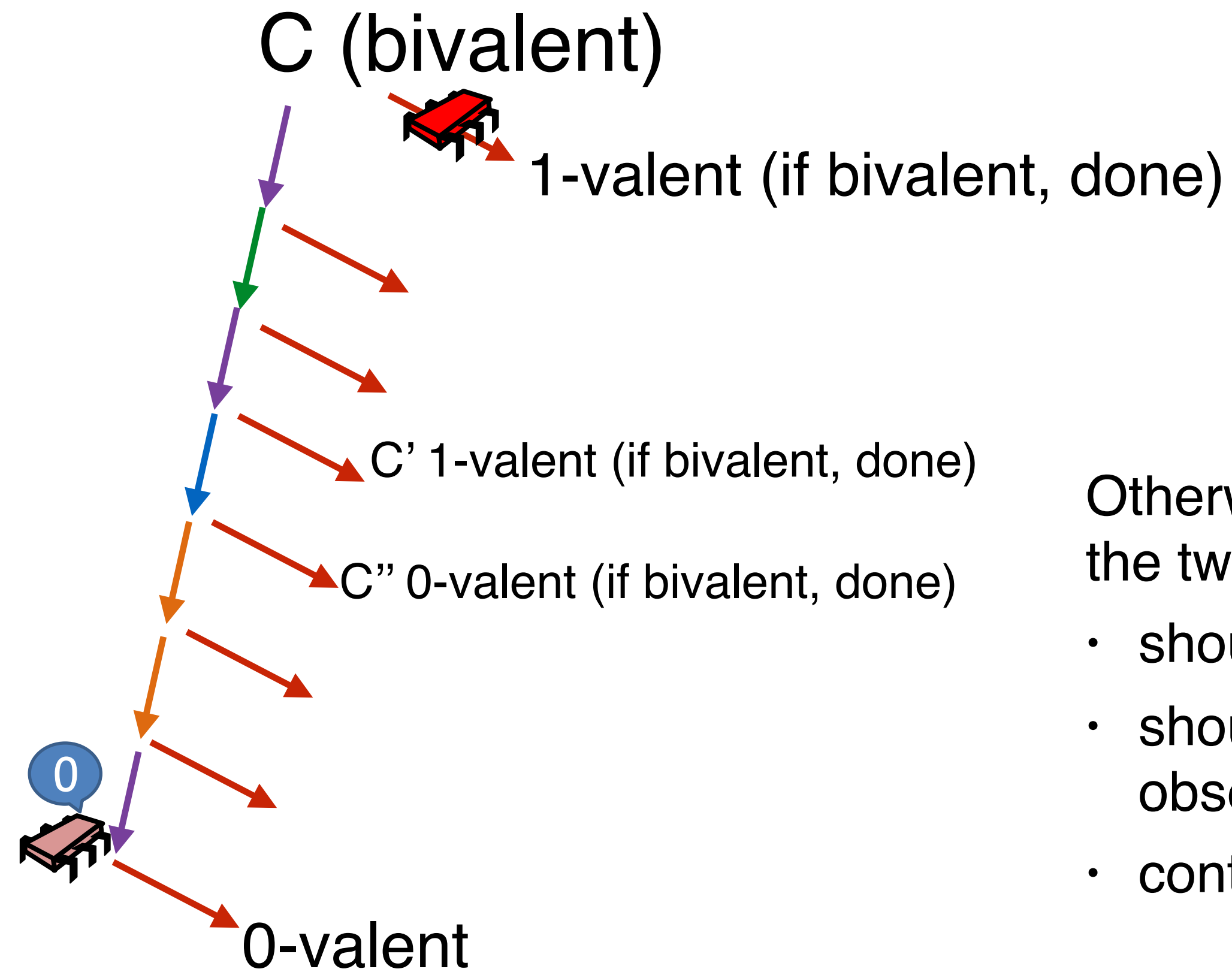
If C is bivalent, no process may have already returned

Key Ingredients Of The Proof:

1. there exists a bivalent initial configuration
2. it is possible to extend a bivalent configuration to another bivalent configuration such that any given process takes at least 1 more step

Iterating round robin, this implies an infinite execution where all processes take infinitely many steps, and gives the desired contradiction (with 1-resiliency)

FLP Proof: Flavor of How Bivalency is Argued



Example case:

 can't be a read

Otherwise, run remaining $n-1$ processes round-robin from each of the two configurations C' and C''

- should return because of 1-resiliency (only blue process is failed)
- should return the same values because of indistinguishability (no observable difference)
- contradicts valencies of these configurations (only 0 and only 1)

FLP Result and Proof: Summary

The FLP result was influential

Perhaps more important is the proof approach.

Valency-based proof approach of inductively extending executions prevalent in hundreds of impossibility and lower bound proofs!

Yet, nobody managed to show that no algorithm can solve k -set agreement for $k > 1$ in a wait-free way using this approach

Are we just not smart enough or is there some fundamental problem?

k-Set Impossibility

Statement: no algorithm can solve k-set agreement for $k > 1$ in a wait-free way!

How the proof works:

1. König's Lemma: For any given wait-free algorithm, there exists a bound B such that no execution takes more than B steps
2. If no execution of a correct algorithm takes more than B steps, then there exists a correct algorithm in a related (IIS) model, where each process takes exactly B steps
3. Sperner's Lemma: If each process takes B steps before it returns in the IIS model, then there exists an execution in which all different input values are returned

If $k+1$ processes start with $k+1$ different inputs, this contradicts agreement!

k-Set Impossibility: König's Lemma

In every infinite tree in which each node has finite children, there is an infinite path

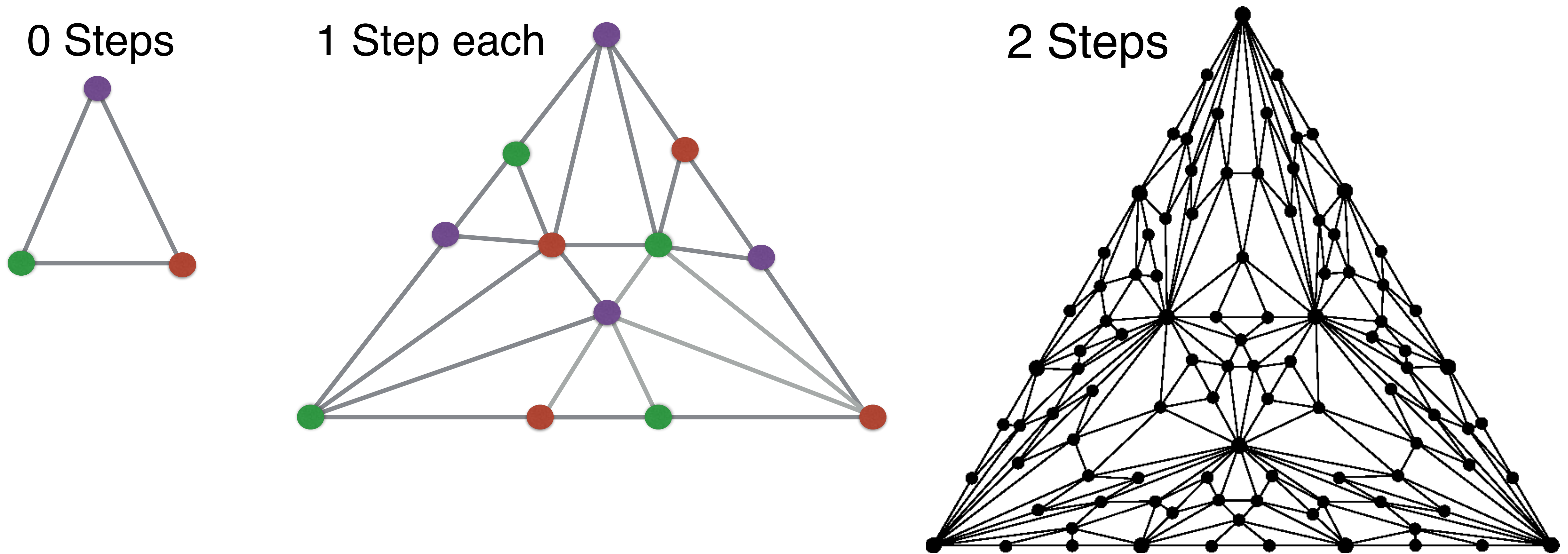
Draw tree of all execution schedules of a wait-free algorithm:

- Every node has at most n children (number of processes that have not yet returned in the corresponding configuration)
- There is no infinite path (wait-freedom)

Hence, tree is finite \Rightarrow exists B such that no execution takes more than B steps

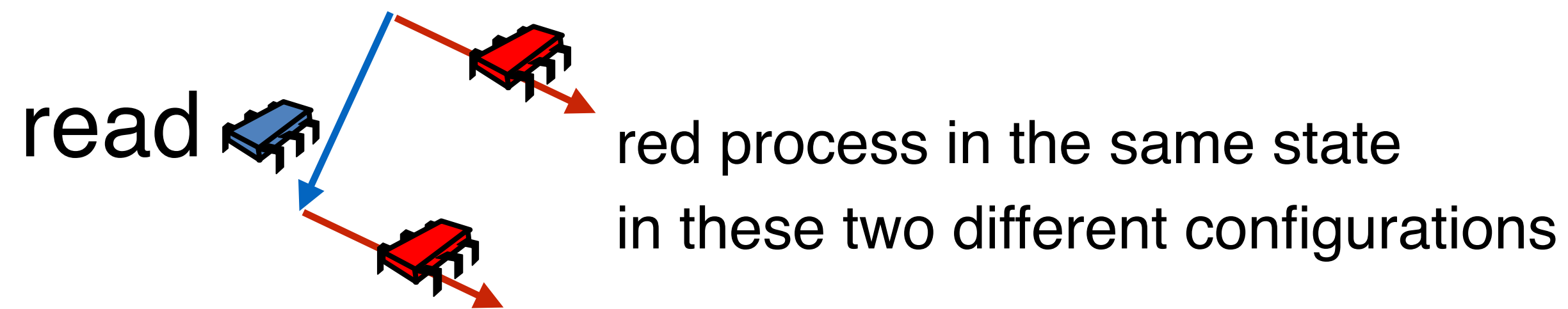
k-Set Impossibility: Topological Representation

For 3 processes in the IIS model. Nodes are possible process states, connected if they can co-exist (triangles are configurations before processes return)



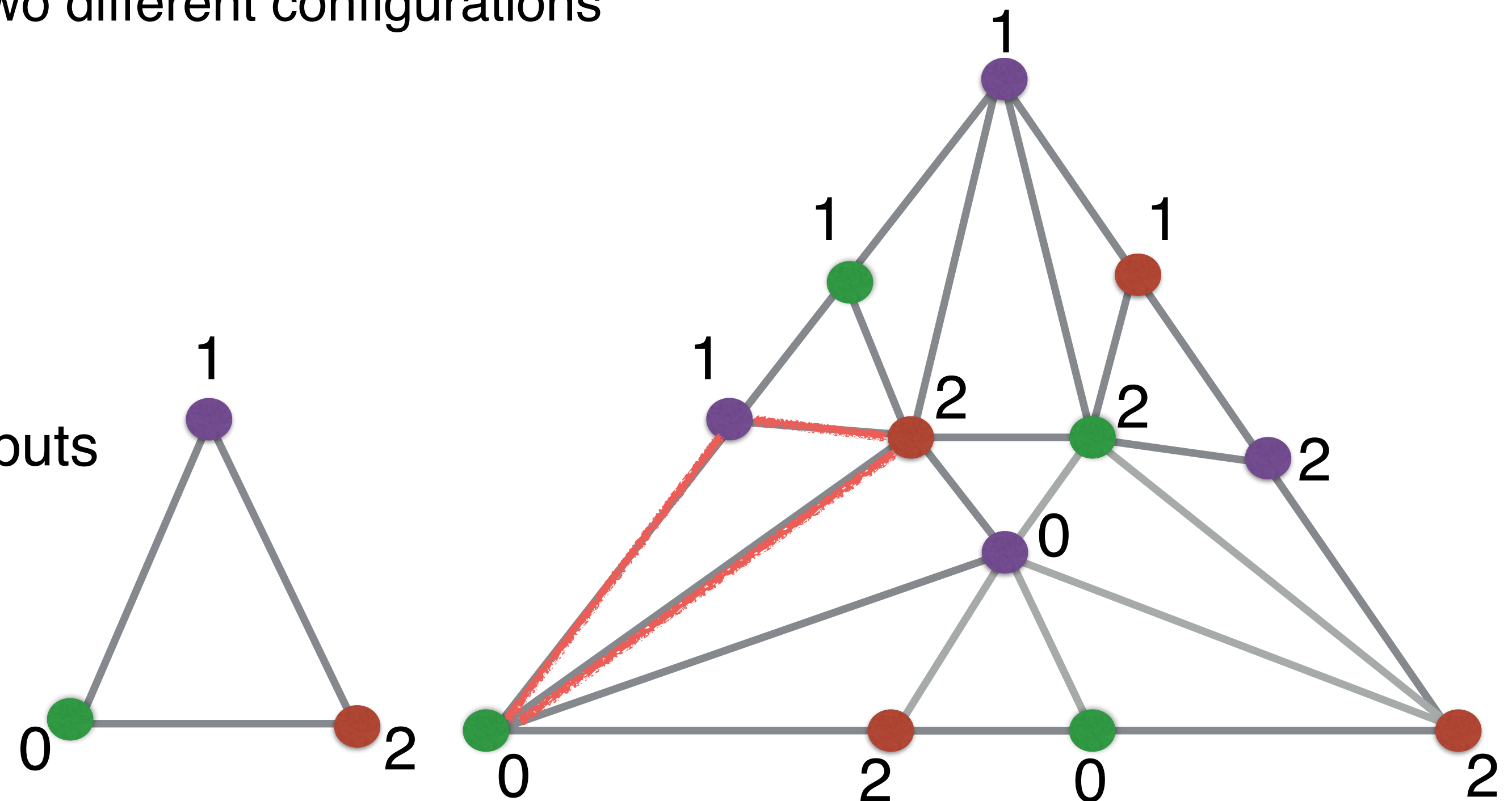
k-Set Impossibility: Topological Representation

How can a same process state occur in different configurations?



Does a correct output mapping exist?

- Each node is mapped to an output
- In every triangle at most 2 different outputs
- Boundary condition satisfied



k-Set Impossibility: Summary

Is combinatorial topology necessary for this proof?

[AC'13], [AP'16]: “Non-topological” proofs of set-agreement impossibility

- Take arbitrary proof of Sperner’s Lemma
- Avoid explicit topological representation equivalent to the distributed model
- Instead, directly re-prove Sperner’s lemma in the model’s context
 - Can avoid proving all properties of subdivisions because that were proved to establish the equivalence of the explicit representation

What does it mean to “need topology”?

Our Approach: Extension-Based Proofs

Define a proof system that encapsulates “conventional” techniques, in particular ones that build counter-example executions by repeated extensions based on valency (much more than just FLP)

Show that the impossibility of k -set agreement cannot be proved in this proof system

Adversarial Argument

Example: an algorithm A should work on all schedules, we can have A execute against *an adversarial scheduler*. By playing the role of adversarial scheduler, we prove lower bounds or show A cannot be correct

A proof for impossibility result should contradict all algorithms

- We play the role of an Adversarial Algorithm (**AA**)
- We represent a proof as a Prover that can query AA

Extension-Based Proof

AA pretends to be a correct algorithm that prover must disprove

The prover explores new configurations from already explored configurations by asking different types of queries to AA

- “assign a particular input to a new process, tell me its resulting state”
- “instruct one of the processes to take the next step, tell me its resulting state”

We also support valency queries by the prover, i.e. for consensus:

“Is there an execution of a given set of processes from this configuration in which 0 is decided? If so, tell me this execution”

Extension-Based Proof

Prover starts in an initial configuration with no assigned inputs

The execution is empty

Prover asks finitely many queries

- May win immediately if it catches AA in lying
- Otherwise, should commit to a non-empty extension to its execution (includes input assignment as the first step of each process)
- Exploration continues from the new execution

In the wait-free case: as soon as the execution is longer than the bound B for AA's algorithm (from König's lemma), the prover wins

So Why Do Extension-Based Proofs Fail?

We design a strategy for AA, such that:

- Every explored execution eventually terminates
 - B is larger than the maximum length
- In no explored execution are $k+1$ different values returned

Two stages

- First AA answers queries by letting processes take more steps as necessary to avoid returning different values in same execution
- After the prover commits to sufficiently long extension, there will be only finitely many executions prover can explore. AA will terminate all of them without contradiction

Why Do Extension-Based Proofs Fail? Cont'd

We represent the Prover's knowledge of AA with a simplicial complex, where unlabeled nodes are unexplored

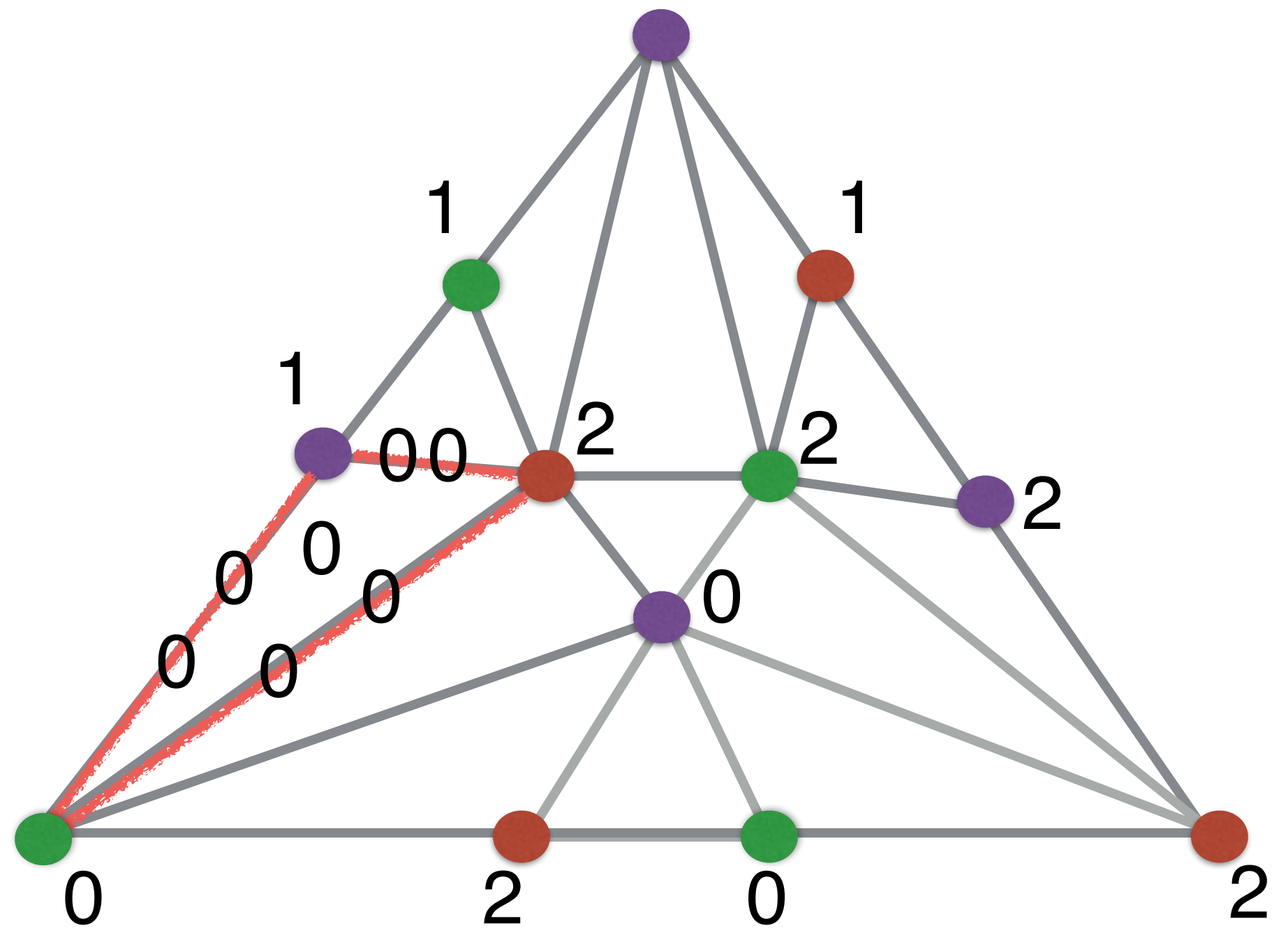
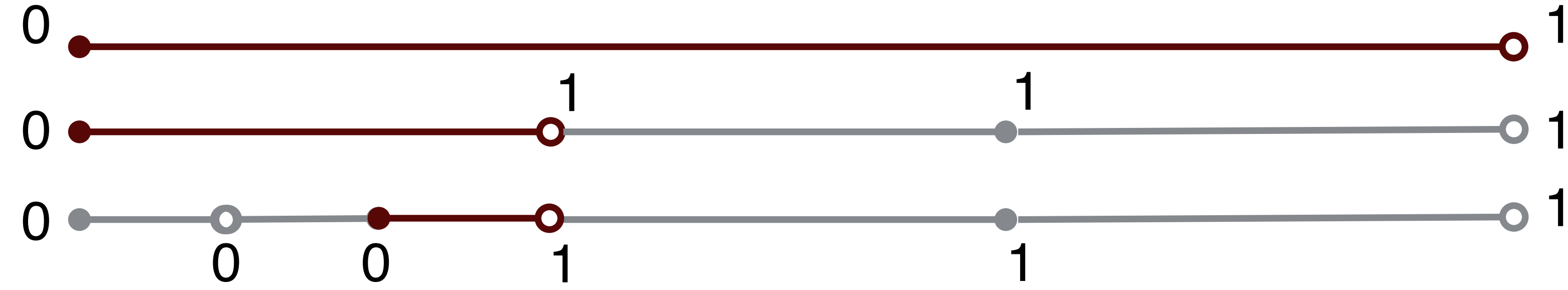
- AA can keep many unlabeled nodes between any nodes that are labeled with output, which correspond to process states that returned. This fixes a bug in [Hoest-Shavit'06]

Once the prover commits to a long extension, AA has enough leeway to pretend that the bad execution was somewhere else, i.e. it did not start with the committed execution.

Hence, even though counter-example execution exists, prover can't find it

Topological View and why FLP prover works for k=1

consensus, k = 1



Space Complexity

Obstruction-free consensus: $\Omega(\sqrt{n})$ [FHS'93], $n-1$ [Zhu'16]

Obstruction-free k -set agreement:

Best Algorithm: $n-k+1$ memory locations

Best Lower Bound: 2

Major open problem

Covering Arguments [Burns,Lynch'92]

All space lower bounds inductively extend executions

- reach configurations satisfying valency properties, repeat
- additionally, cover increasing number of registers

A maintained property for consensus

- both output values can still be returned
- thus, processes cannot return, will take more steps

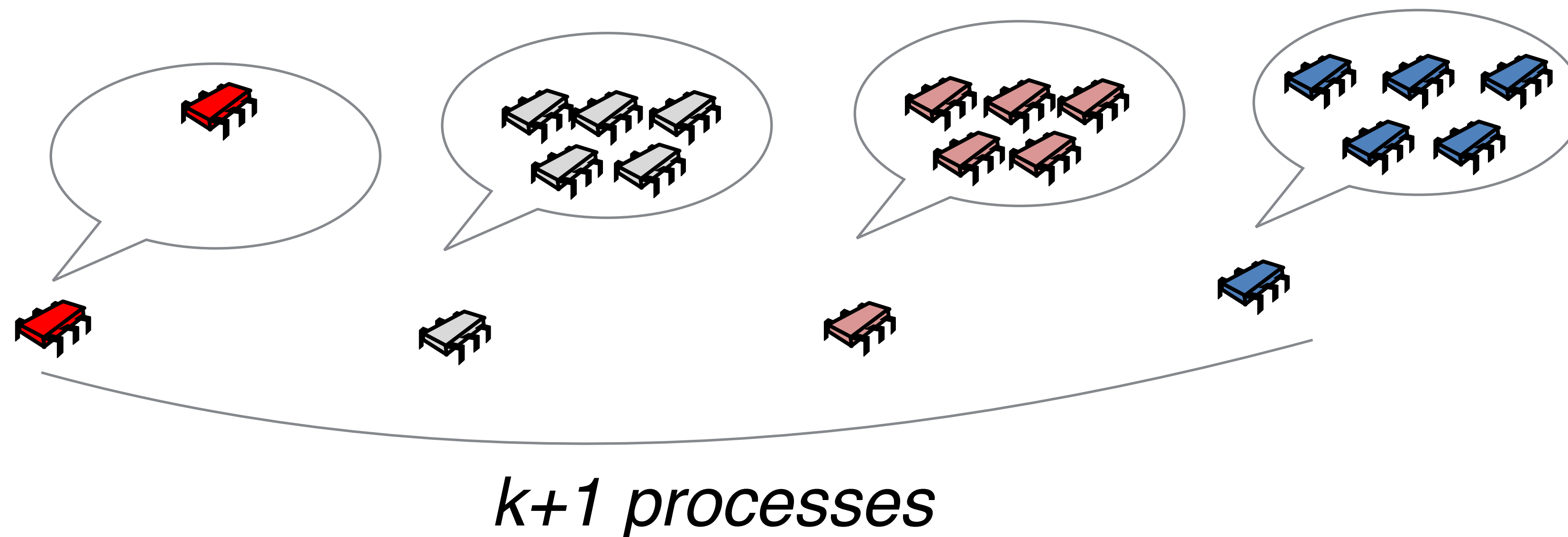
Extension-based!

Sometimes steps added earlier in the execution - affects query model

Our Results [EGZ'17]

A general lower bound of n/k based on a simulation:

- $k+1$ real processes are solving wait-free k -set agreement (impossible) by simulating processes with their own input
- n simulated processes are running obstruction-free k -set agreement that uses less than n/k registers



Our Results [EGZ'17]

A general lower bound of n/k based on a simulation:

- $k+1$ real processes are solving wait-free k -set agreement (impossible) by simulating processes with their own input
- n simulated processes are running obstruction-free k -set agreement that uses less than n/k registers

Complicated Mechanics

- Simulators try to cover simulated registers by simulated processes - once everything is covered output value can be computed!
- Hence, each simulator is doing a covering lower bound
- “Revisionist”: Simulators may change history of their simulated processes (but stay consistent to what other simulators have seen)
- Maintaining consistency is extremely technical (augmented snapshot, vector timestamps)

Final Thoughts

We provide a language for reasoning about capabilities of proof techniques for impossibility results in distributed computing

So, is topology necessary?

How many simplexes should the prover verify in the worst case to get a contradiction