Asynchronous Robot Gathering

(A Tiny Tutorial on Distributed Computing Through Combinatorial Topology)

Armando Castañeda
Instituto de Matemáticas, UNAM

Joint work with:
Manuel Alcántara, David Flores, Sergio Rajsbaum, UNAM
Matthieu Roy, LAAS-CNRS

Gathering

A collection of robots start on vertices of a connected graph. They can move on vertices.

- **Termination.** Correct robots decide a vertex.
- **Validity.** If participating robots start on the same vertex, they stay there.
- **Agreement.** All final vertices are the same

Asynchronous Luminous Robots (ALR)

- n **fully asynchronous** robots with low memory (the less the better).
- **Luminous:** Each robot has a light to communicate information (the less the better).
- Two available **atomic** operations:
    1. **Look:** Takes a snapshot of the whole environment (position and lights colors).
    2. **Move:** Robot moves to an adjacent or same vertex and changes light color.
- Up to n-1 **crash failures** (robots stop or disappear).

Asynchronous Luminous Robots (ALR)

Computation proceed in a sequence of **asynchronous** rounds.

```
Algorithm choose(v, G):
    r = non-negative integer
    view = empty
    undecided = true
    While undecided do
        Move(v, r)
        view = Look(G) U view
        (v, r, undecided) = Compute(view)
    endWhile
    return v
```

Robots can **start at distinct times** or **do not even start**.

Asynchronous Luminous Robots (ALR)

Computation proceed in a sequence of **asynchronous** rounds.

Algorithm choose(v, G):
    r = non-negative integer
    $view$ = empty
    $undecided$ = true
    While $undecided$ **do**
        Move(v, r)
        $view$ = **Look**(G) U $view$
        (v, r, $undecided$) = **Compute**($view$)
    endWhile
    return v

Becomes visible

Robots can **start at distinct times** or **do not even start**.

Related Work

- Gathering has been studied a lot, usually without failures and in continuous space.
  [Flocchini et al. 2012, Agmon and Peleg 2006, Bramas and Tixeuil 2015, Cieliebak et al. 2012, Klasing et al. 2008, …]

- ALR model introduced in the past, without failures and without distinct starting times.
  [Shantanu Das et al. 2016]

- First time gathering is studied considering full asynchrony+failures.

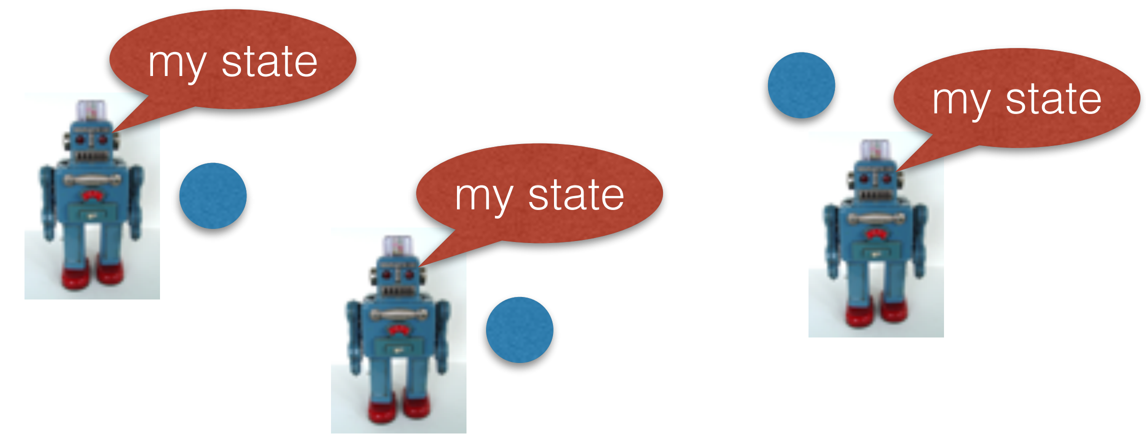
Impossibility of Gathering

For $n > 1$, the gathering is solvable if and only if the base graph $\mathcal{G}$ is a single vertex

- Proof based on the topology approach to distributed computing.
- Enough to analyze the case $n=2$.

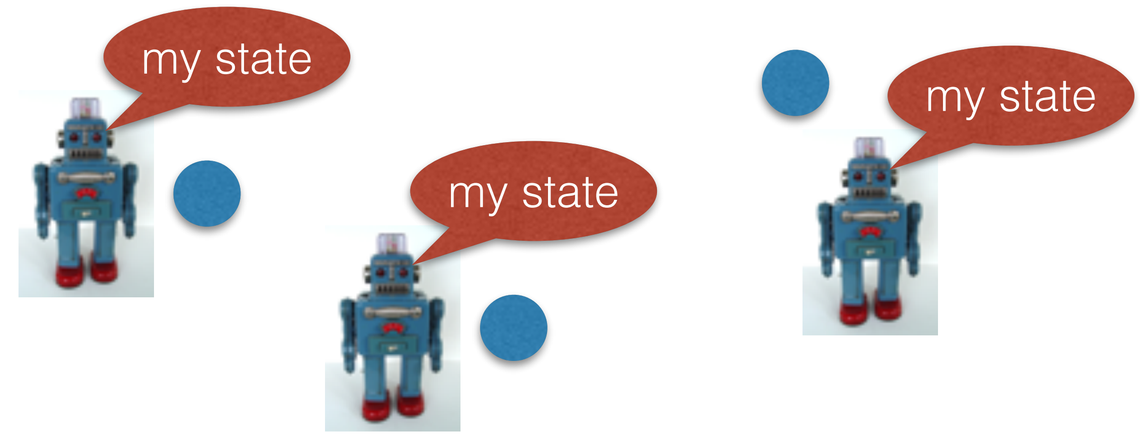
Distributed Computing and Topology

- Vertices represent robot/ process states.

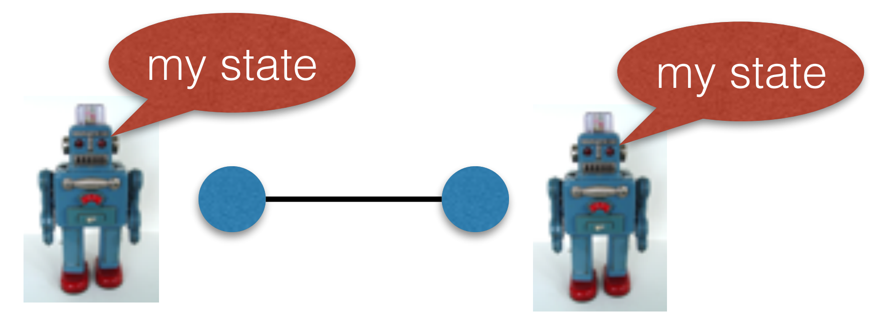


Distributed Computing and Topology

- Vertices represent robot/ process states.

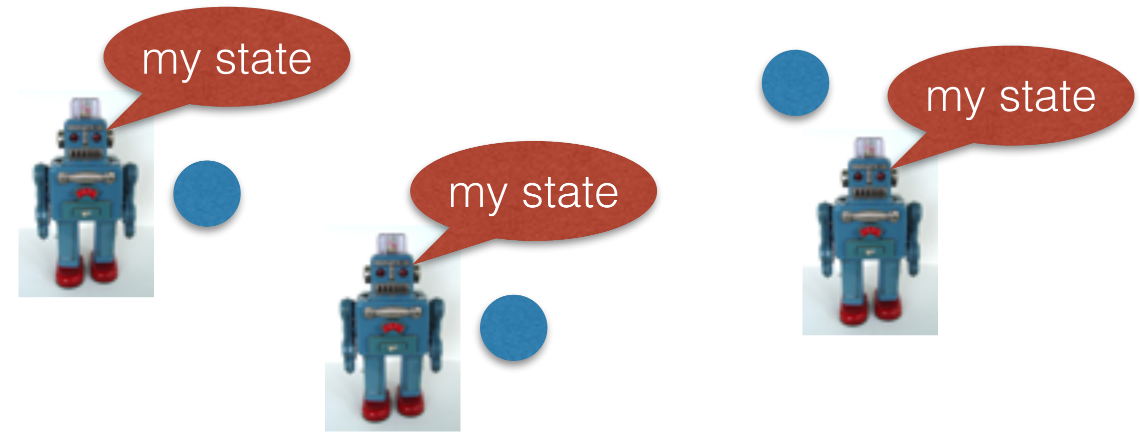


- Simplices (vertices, edges, triangles…) represent mutually compatible system states.

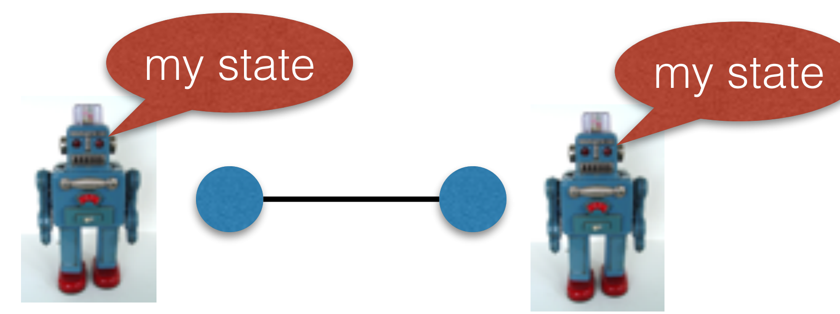


Distributed Computing and Topology

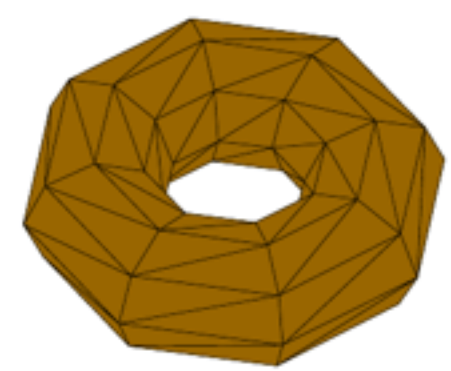
- Vertices represent robot/ process states.



- Simplices (vertices, edges, triangles…) represent mutually compatible system states.



- Complexes put all together.



- Target: Understand properties of the complexes.

Colorless Tasks

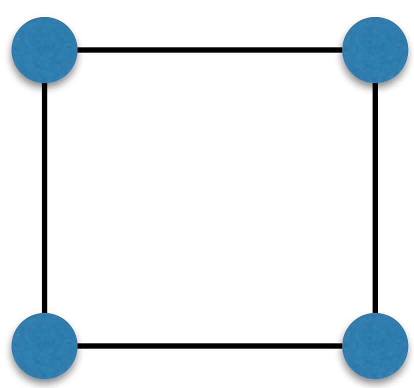
- A task is a triple $(\mathcal{I}, \mathcal{O}, \mathcal{F})$:

1. $\mathcal{I}$: input complex with valid input sets. Each set represent several configurations.
2. $\mathcal{O}$: Output complex with valid output sets. Each set represent several configurations.
3. $\mathcal{F}$: Input/Output function from input simplexes to output sub complexes.

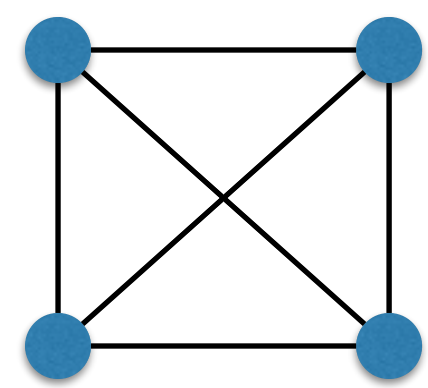
2-Robot Gathering Task

- Input Complex: Graph modeling possible input sets.

$\mathcal{G}$



$\mathcal{I}$

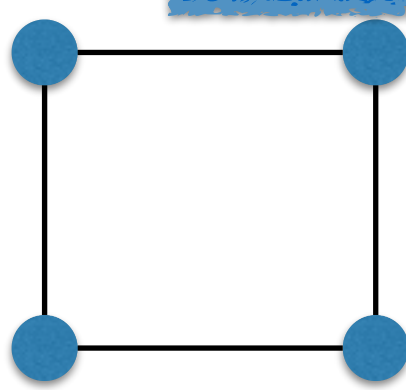


2-Robot Gathering Task

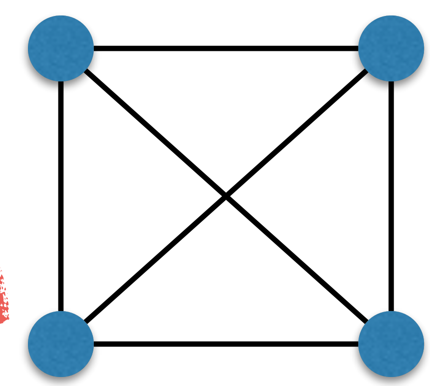
- Input Complex: Graph modeling possible input sets.

Robots start on the vertex

$\mathcal{G}$



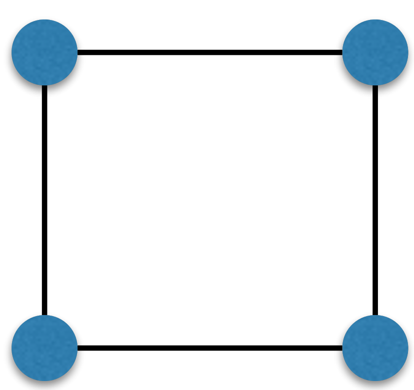
$\mathcal{I}$



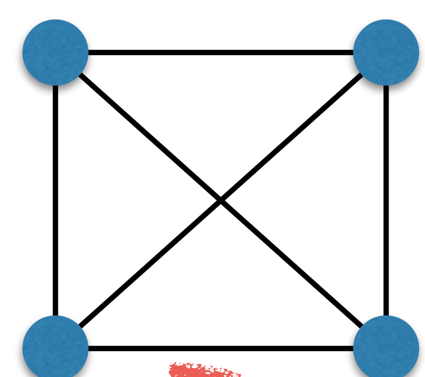
2-Robot Gathering Task

- Input Complex: Graph modeling possible input sets.

$\mathcal{G}$



$\mathcal{I}$



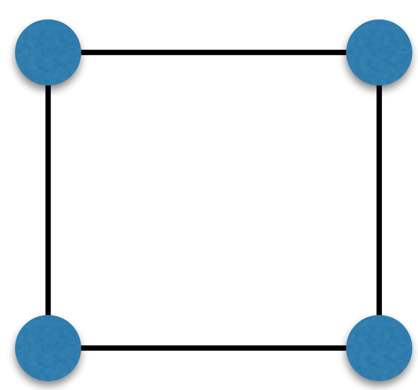
Robots start on the vertices
(it does not matter which one*)

*Colorless task

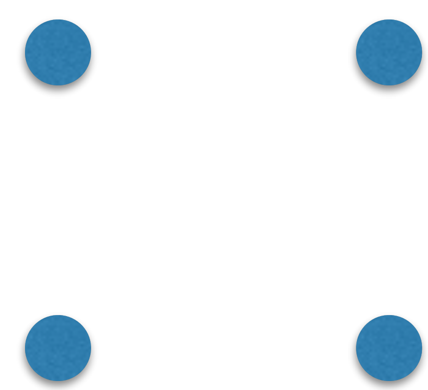
2-Robot Gathering Task

- <u>Output Complex:</u> Graph modeling possible output sets.

$\mathcal{G}$

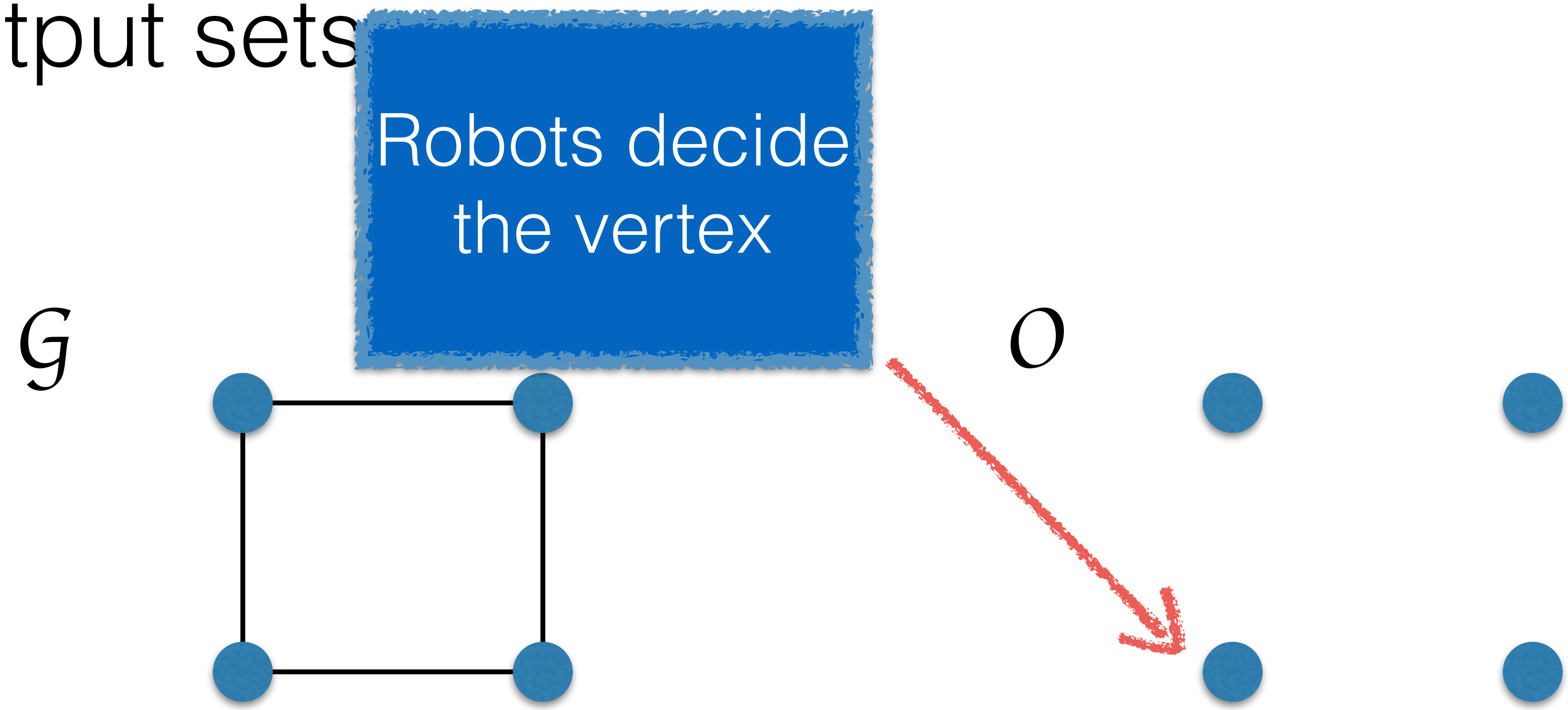


$\mathcal{O}$



2-Robot Gathering Task

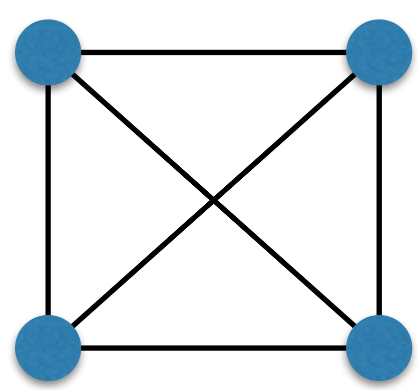
- <u>Output Complex:</u> Graph modeling possible output sets



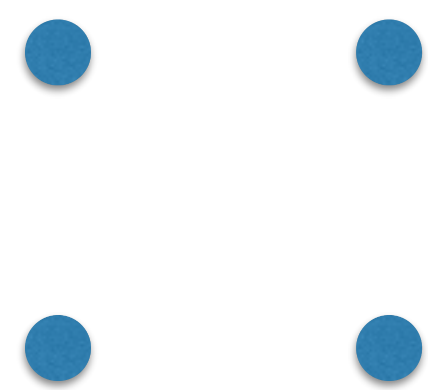
2-Robot Gathering Task

- <u>Input/Output Function:</u> Relation between inputs and outputs.

$\mathcal{I}$

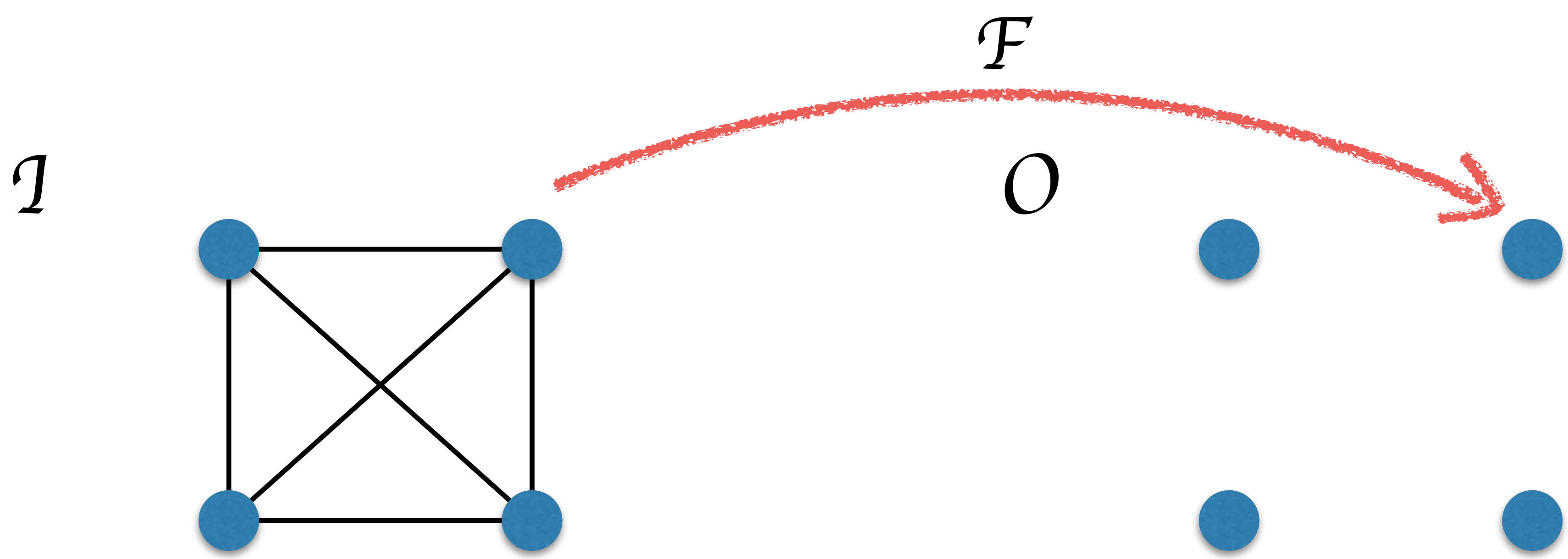


$\mathcal{O}$



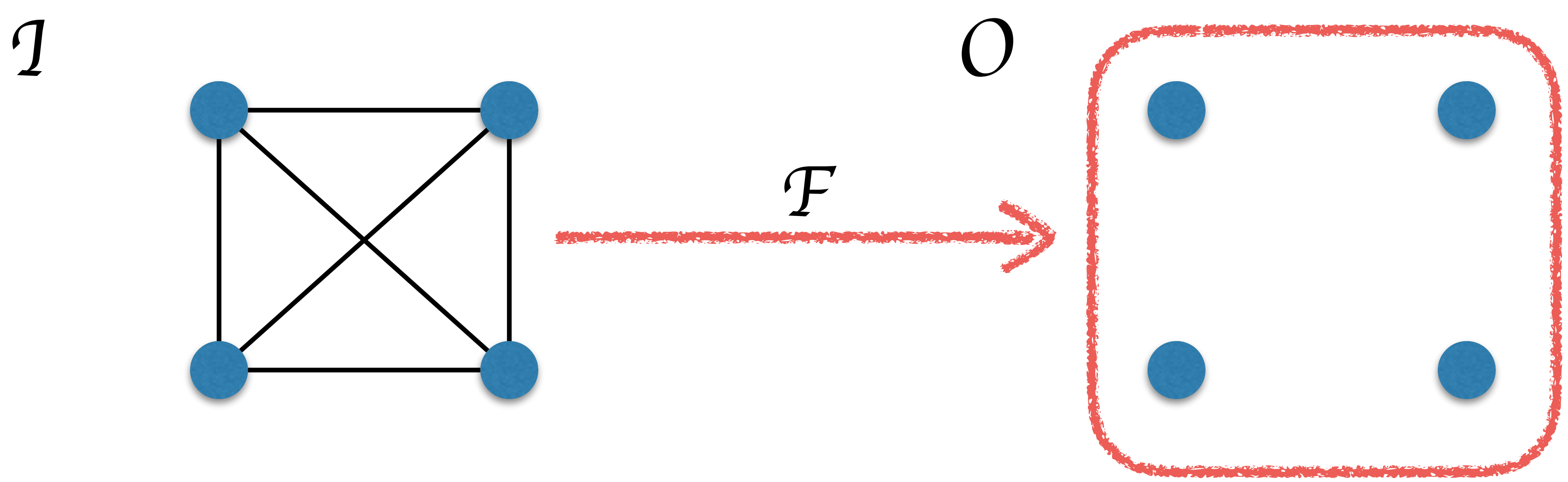
2-Robot Gathering Task

- Input/Output Function: Relation between inputs and outputs.



2-Robot Gathering Task

- <u>Input/Output Function:</u> Relation between inputs and outputs.



2-Robot Gathering Task

- For a given graph $\mathcal{G}$, the 2-robot gathering task is the triple $(\mathcal{I}, \mathcal{O}, \mathcal{F})$:

    1. $\mathcal{I}$: Complete graph with vertices $\mathcal{V}(\mathcal{G})$.
    2. $\mathcal{O}$: Empty graph with vertices $\mathcal{V}(\mathcal{G})$.
    3. $\mathcal{F}$: For every vertex v, $\mathcal{F}(v) = v$;
    For every edge e, $\mathcal{F}(e) = \mathcal{O}$.

Protocol Complex

- Complex $\mathcal{P}$ modeling all (or some relevant subset of) executions.
- Vertices: Local states.
- Simplices: Mutually compatible states (state of robots at the end of an execution).
- For every input configuration simplex s, $\mathcal{E}(s)$ = sub complex of $\mathcal{P}$ with all executions with initial state s.

Protocol Complex

- Complex $\mathcal{P}$ m … levant subset of) executi…

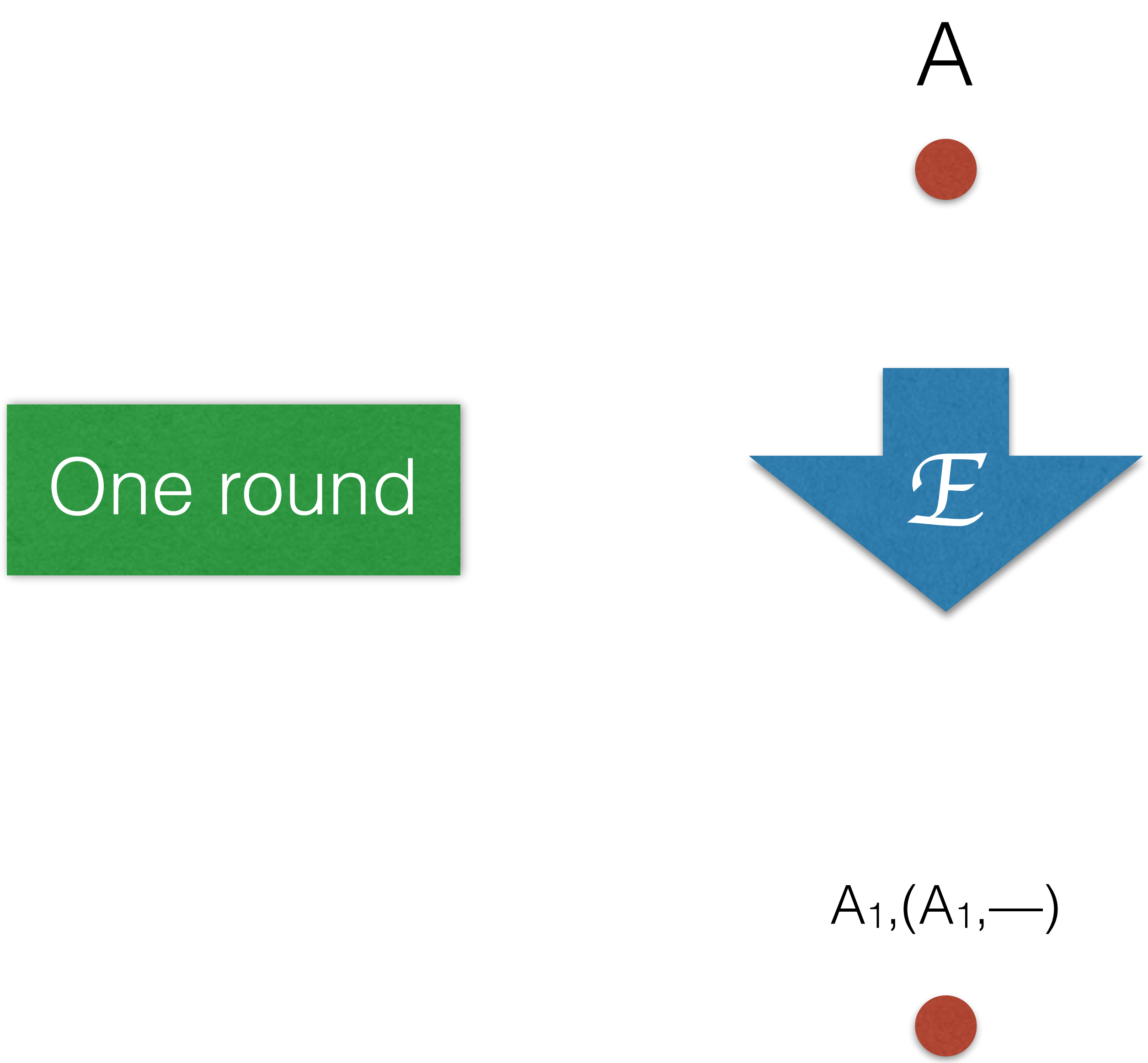
Tiny detail:
full-information (roughly)

Algorithm choose(v, $\mathcal{G}$):
  r = non-negative integer
  $view$ = empty
  $undecided$ = true
  While $undecided$ **do**
    Move(v, r)
    $view$ = **Look**($\mathcal{G}$) U $view$
    (v, r, $undecided$) = **Compute**($view$)
  endWhile
  return v

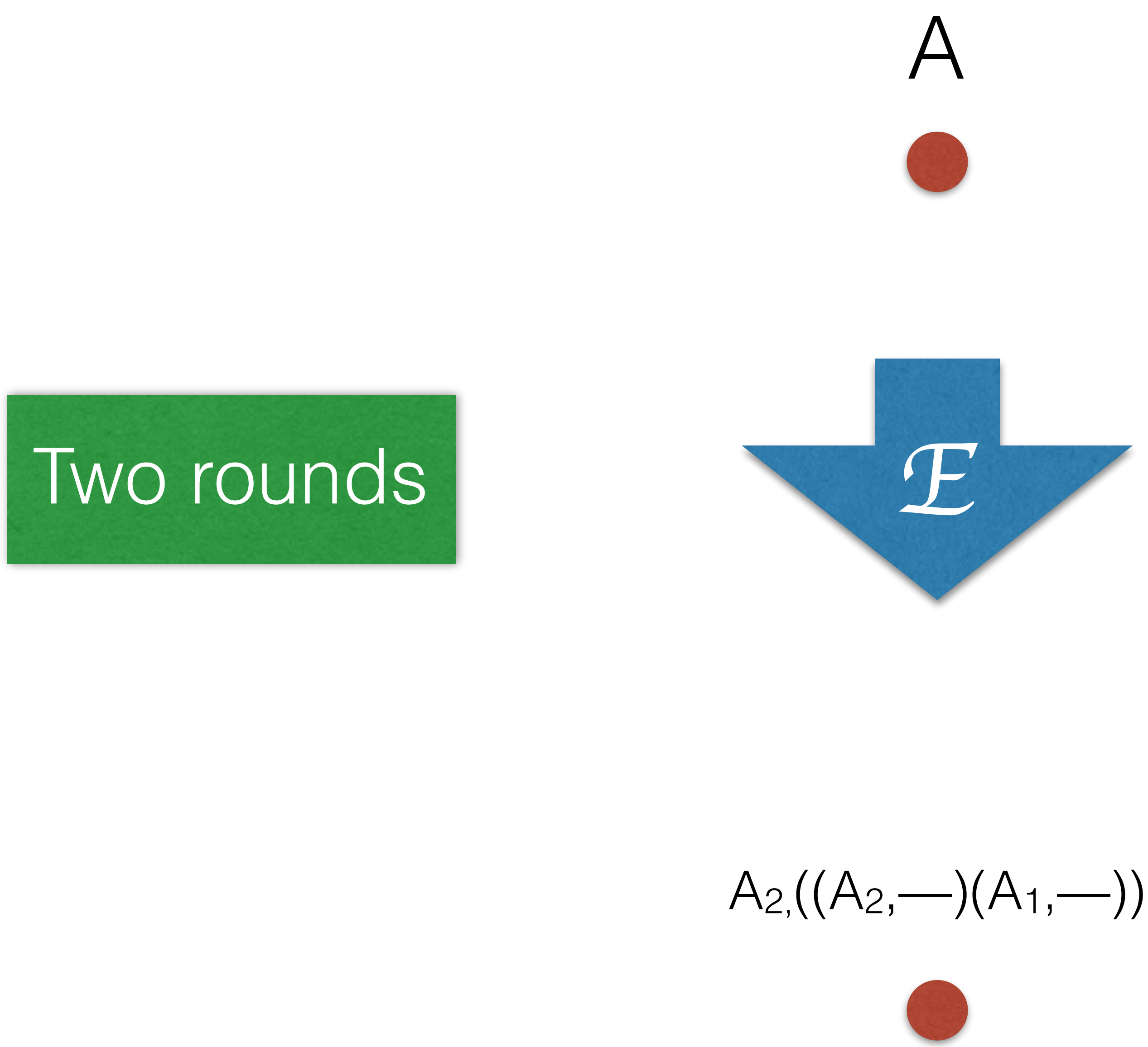
Encode all it has seen

- Vertices: L…
- Simplices … robots at t…
- For every … , $\mathcal{E}(s)$ = sub complex of $\mathcal{P}$ with all executions with initial state s.

2-Robot Protocol



2-Robot Protocol



2-Robot Protocol

A

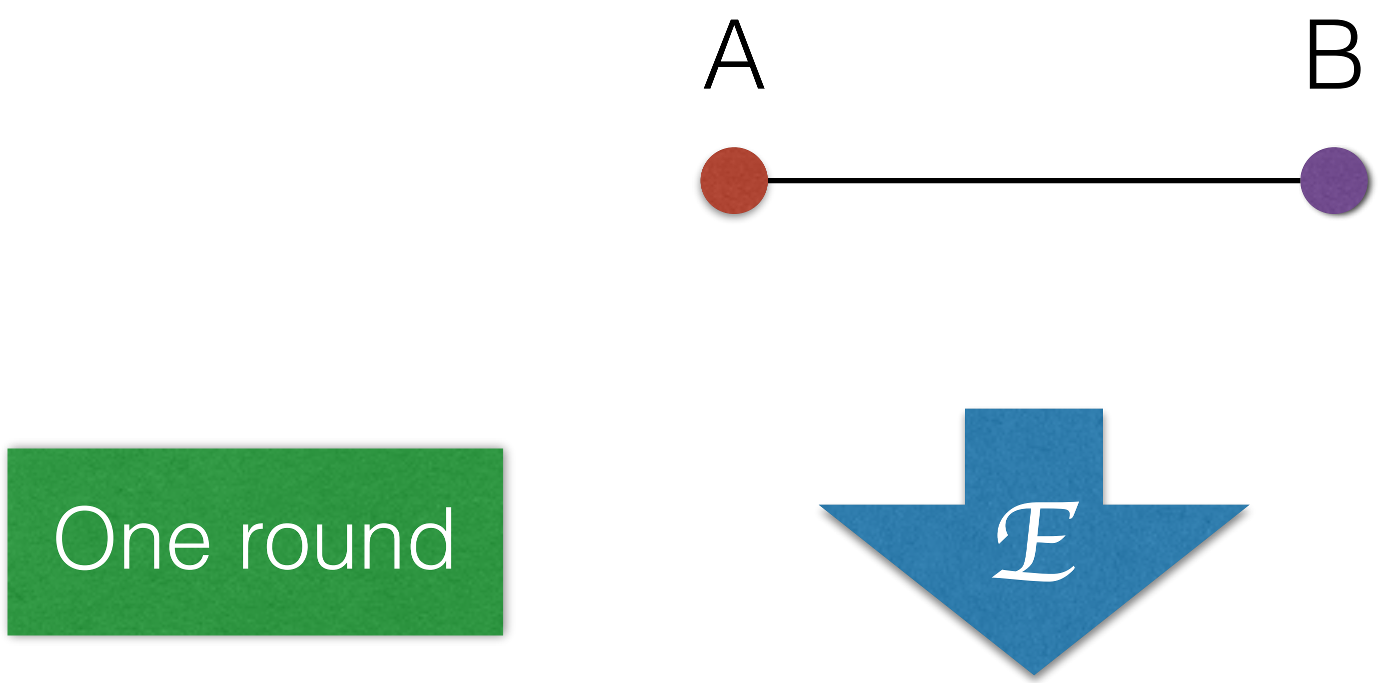
Two rounds

And so on …

$\mathcal{E}$

$A_2, ((A_2, —)(A_1, —))$

2-Robot Protocol

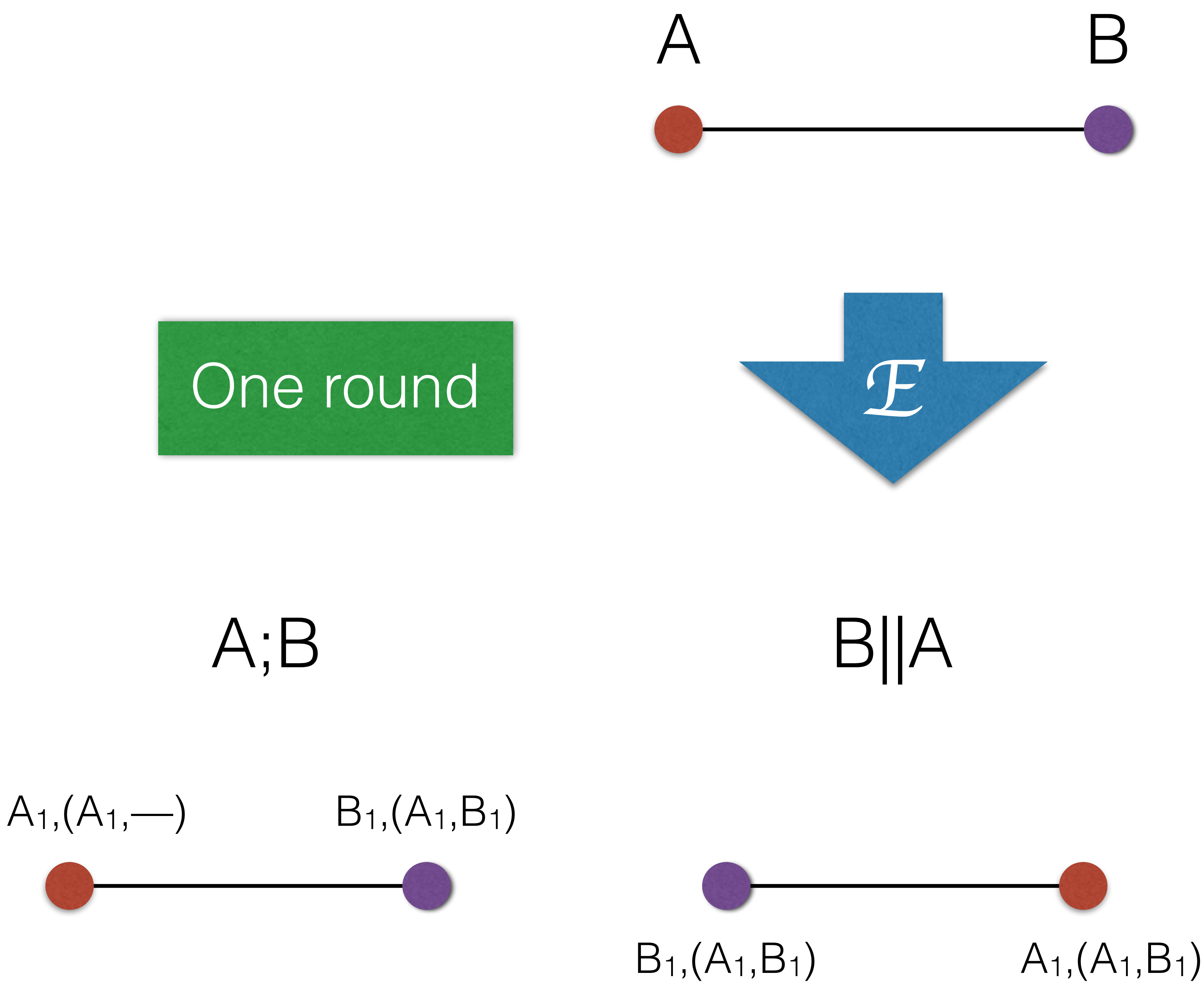


2-Robot Protocol

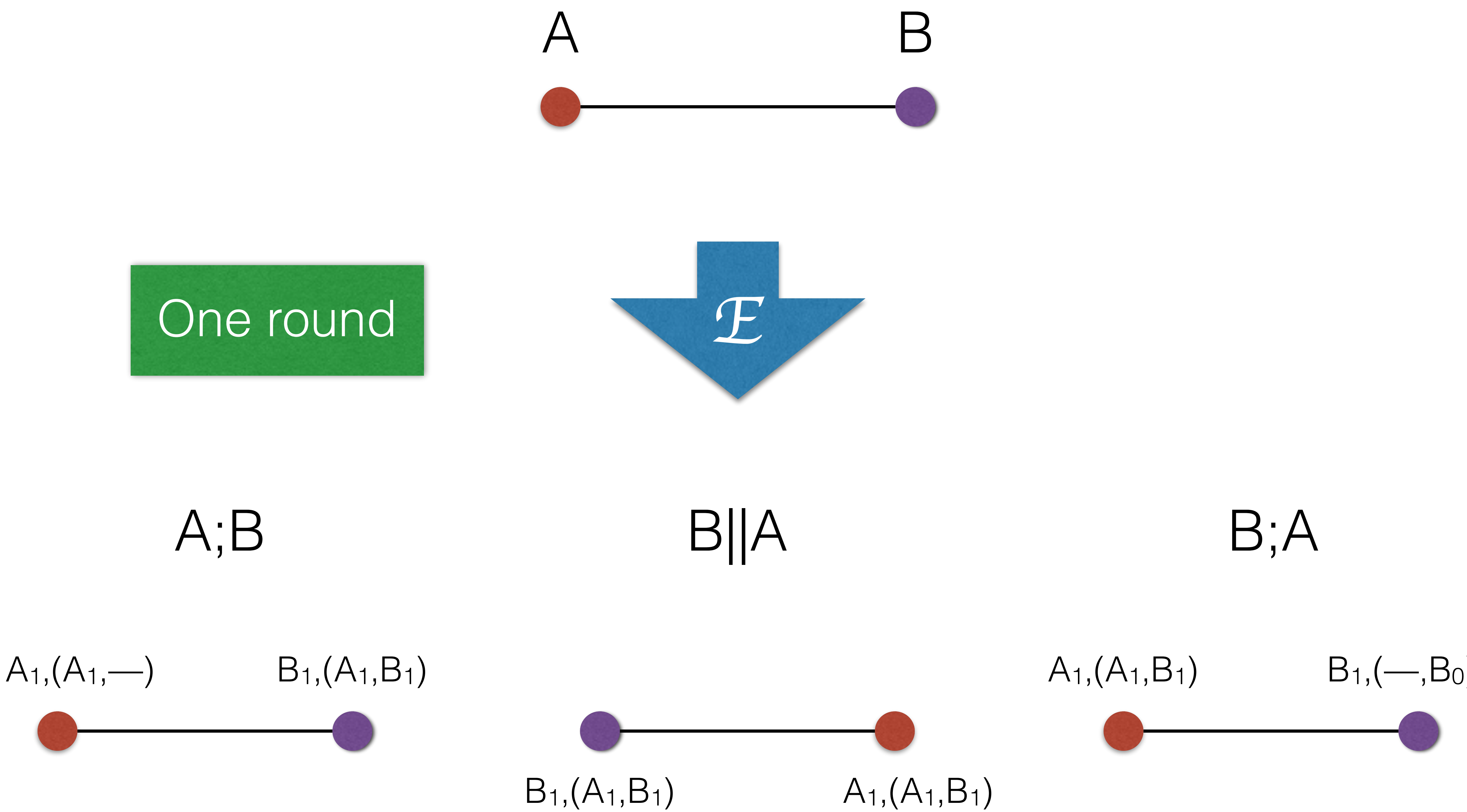
One round

A;B

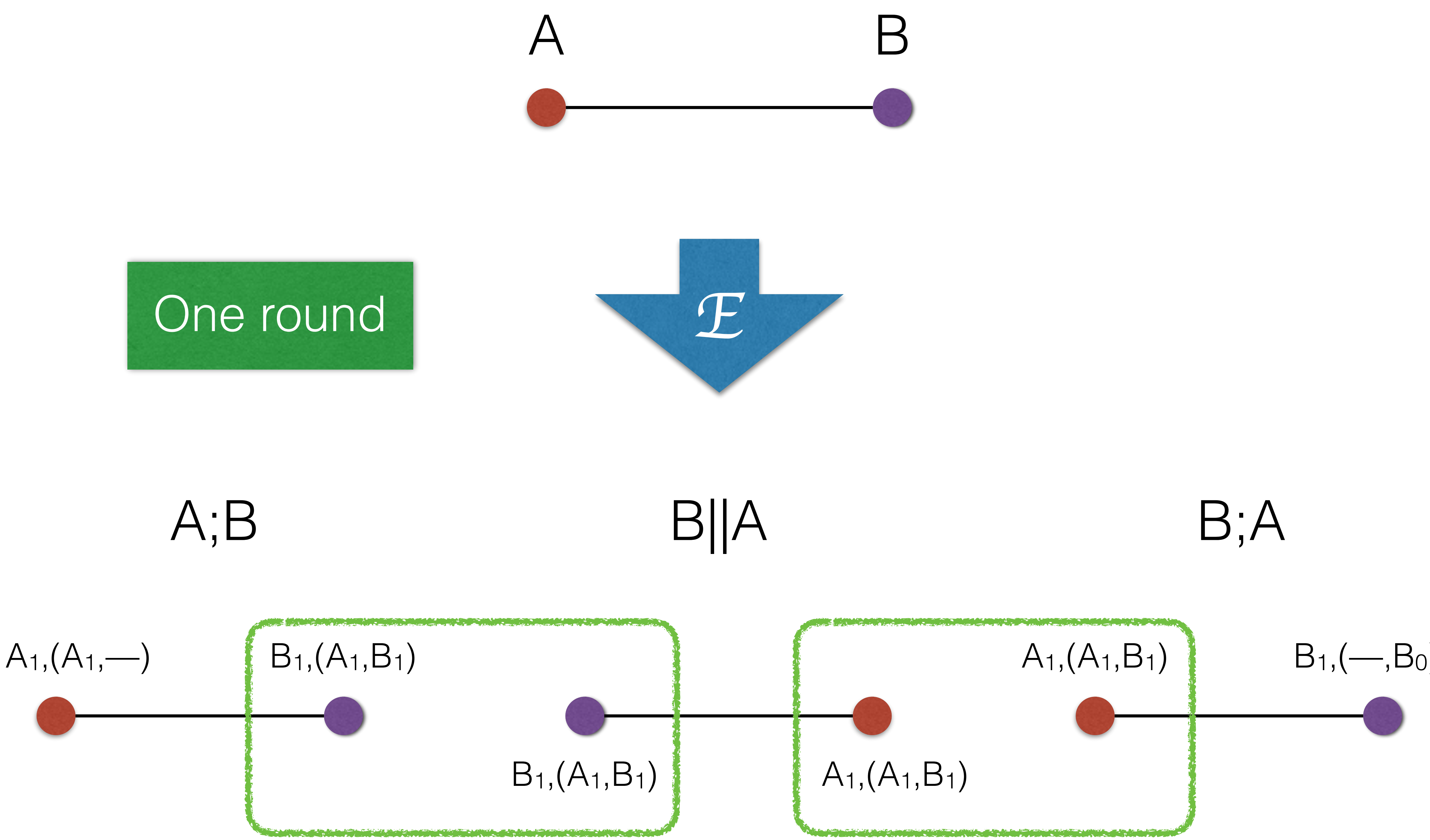
2-Robot Protocol



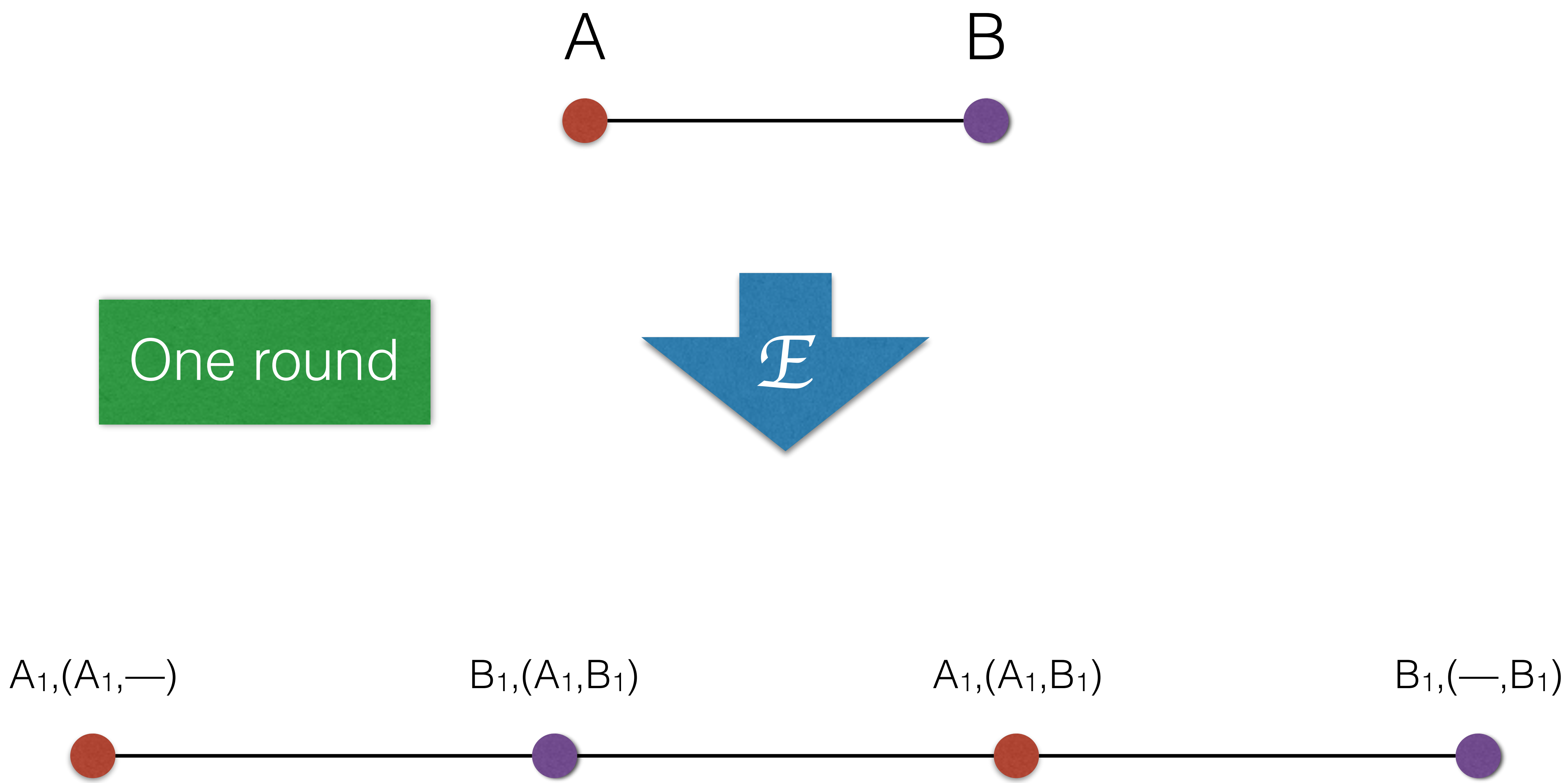
2-Robot Protocol



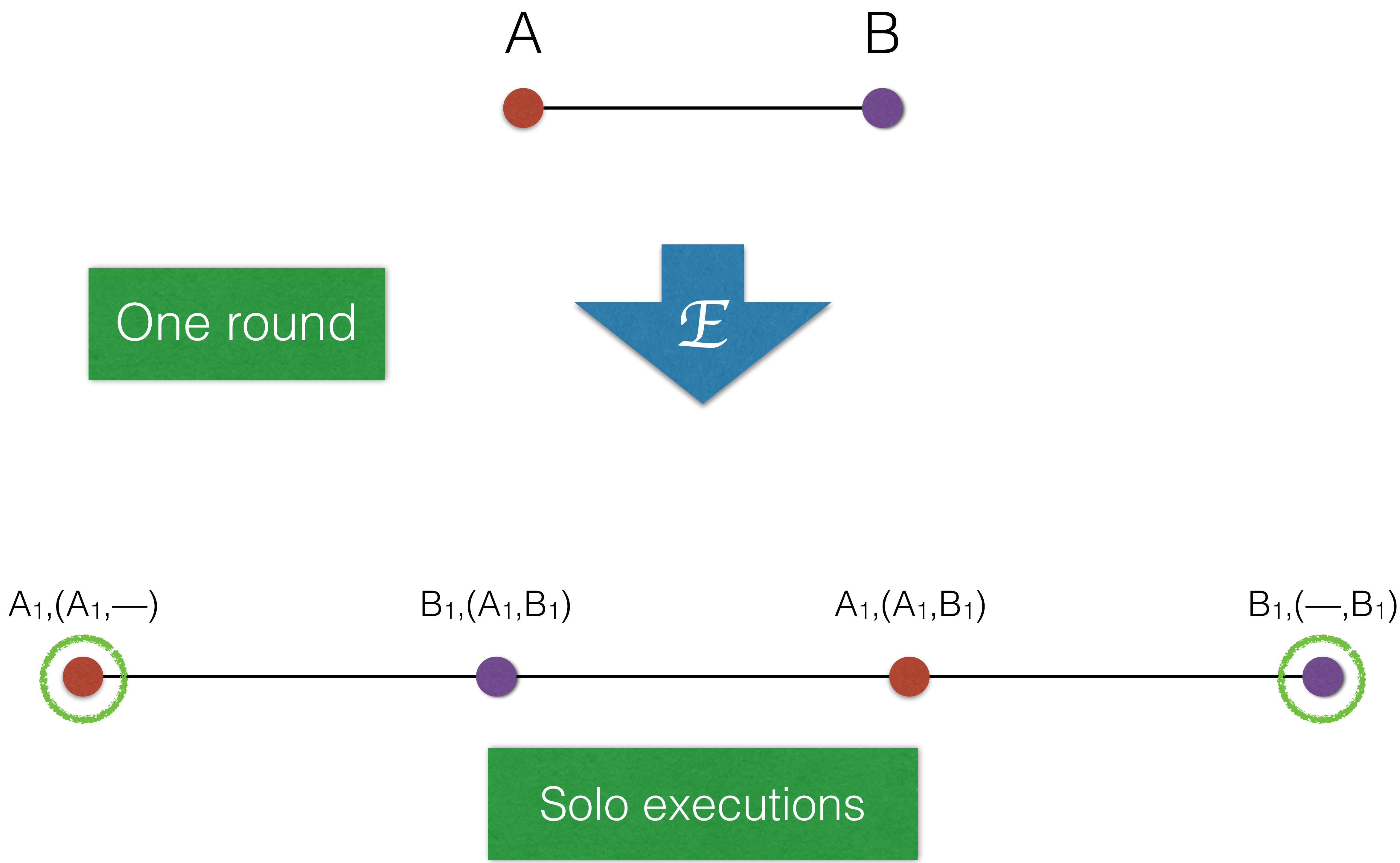
2-Robot Protocol



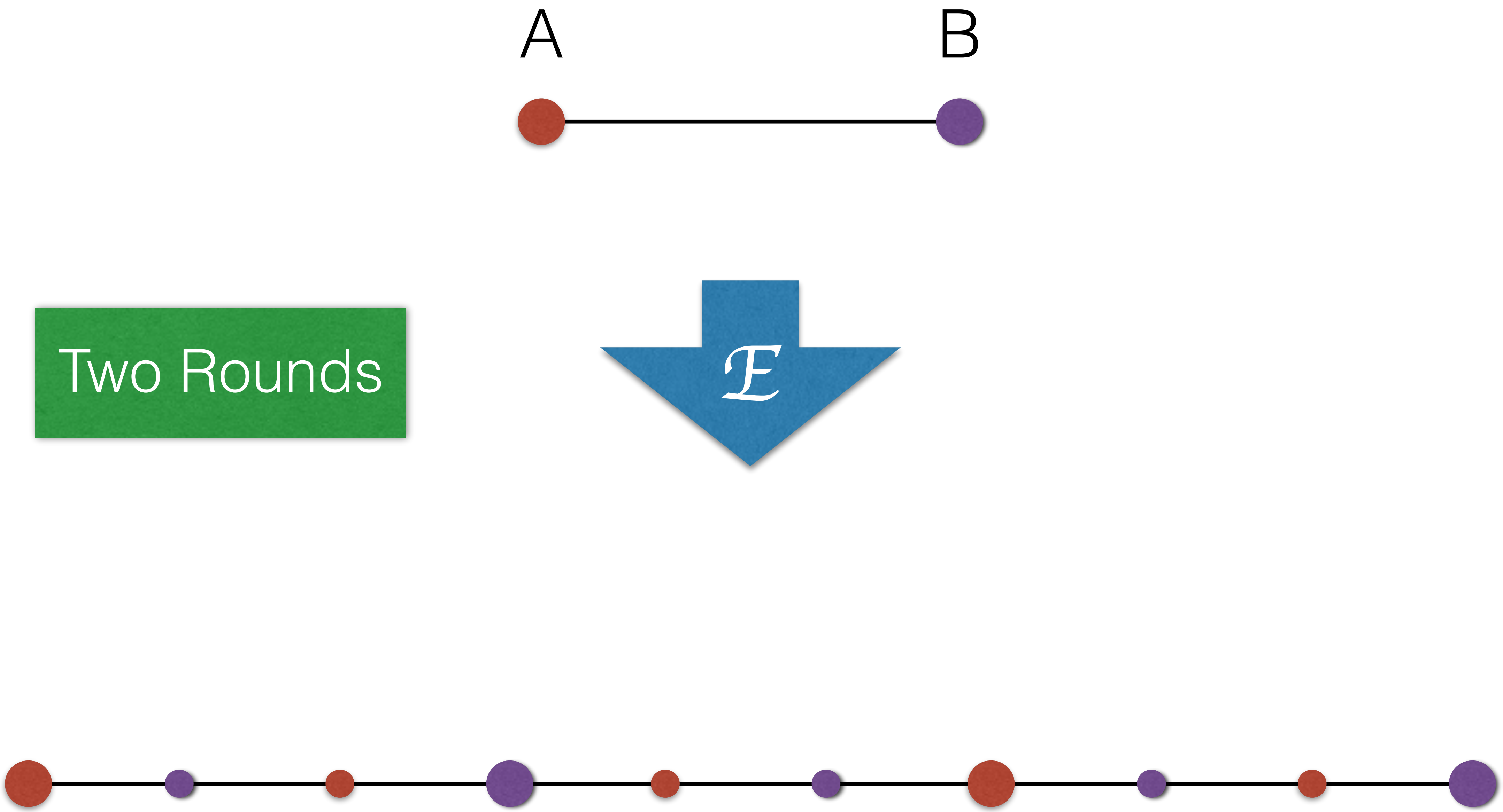
2-Robot Protocol



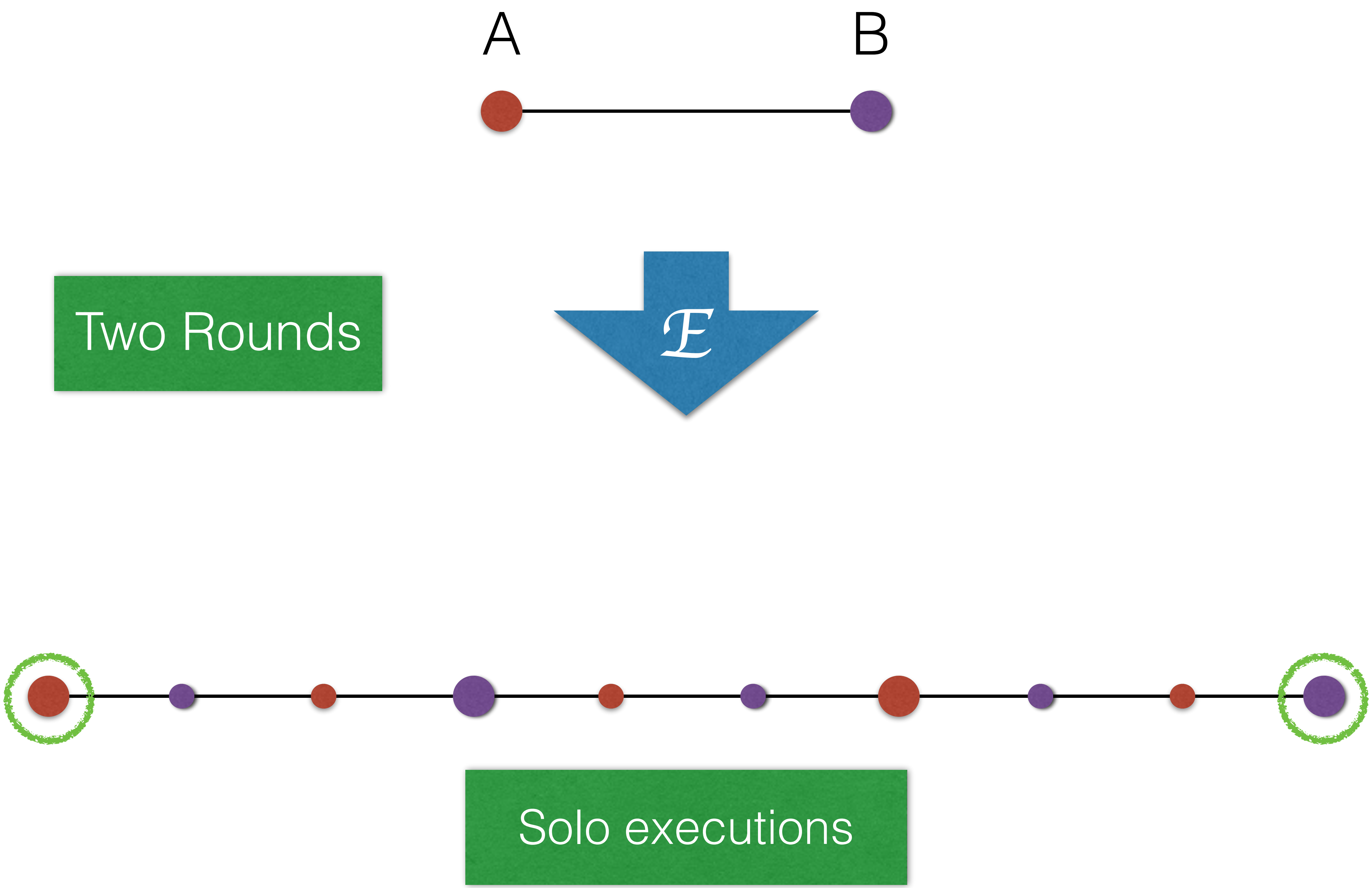
2-Robot Protocol



2-Robot Protocol



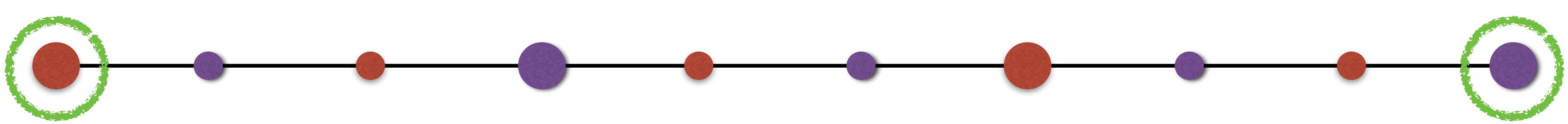
2-Robot Protocol



2-Robot Protocol

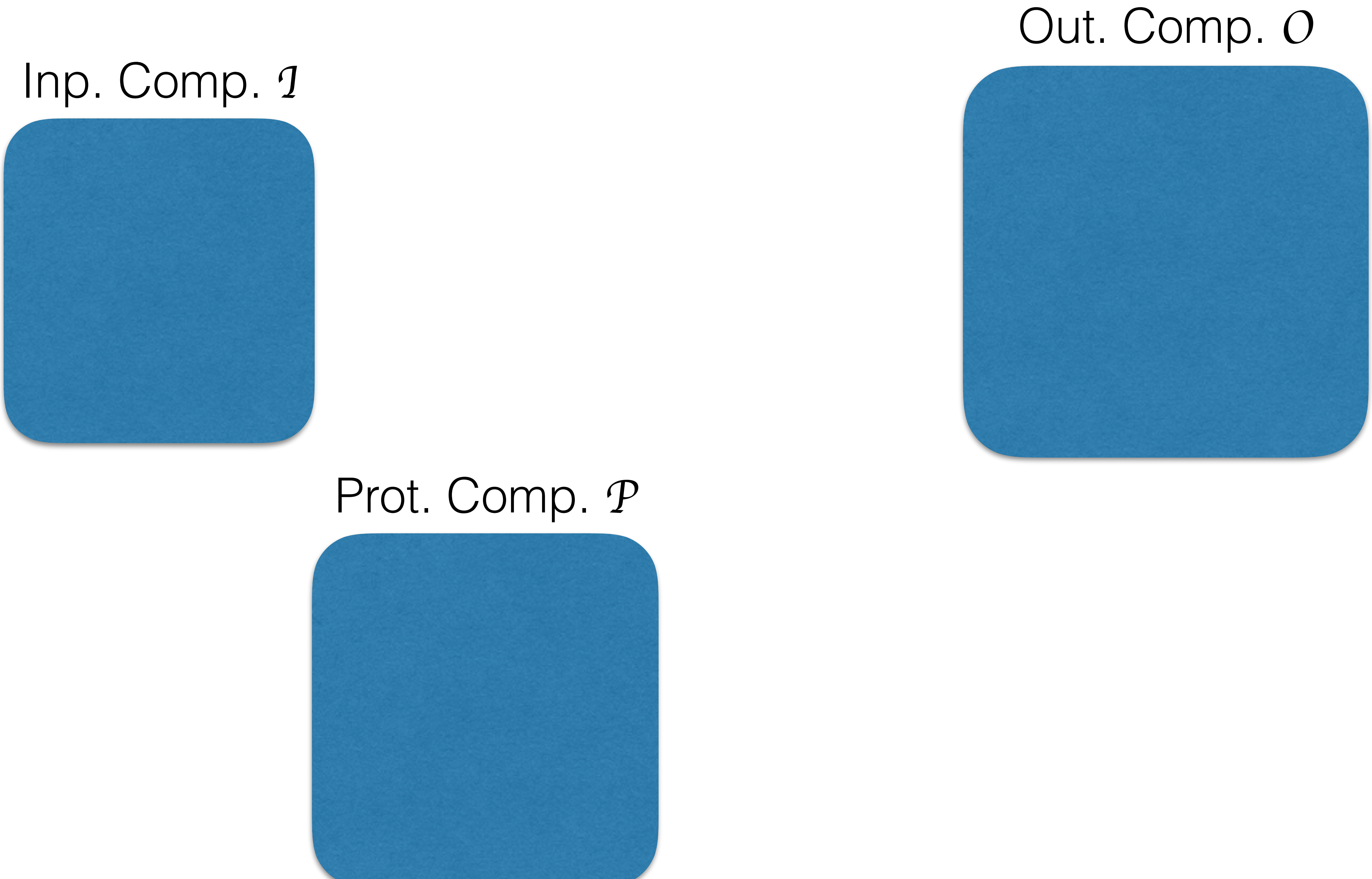
Two Rounds

And so on …

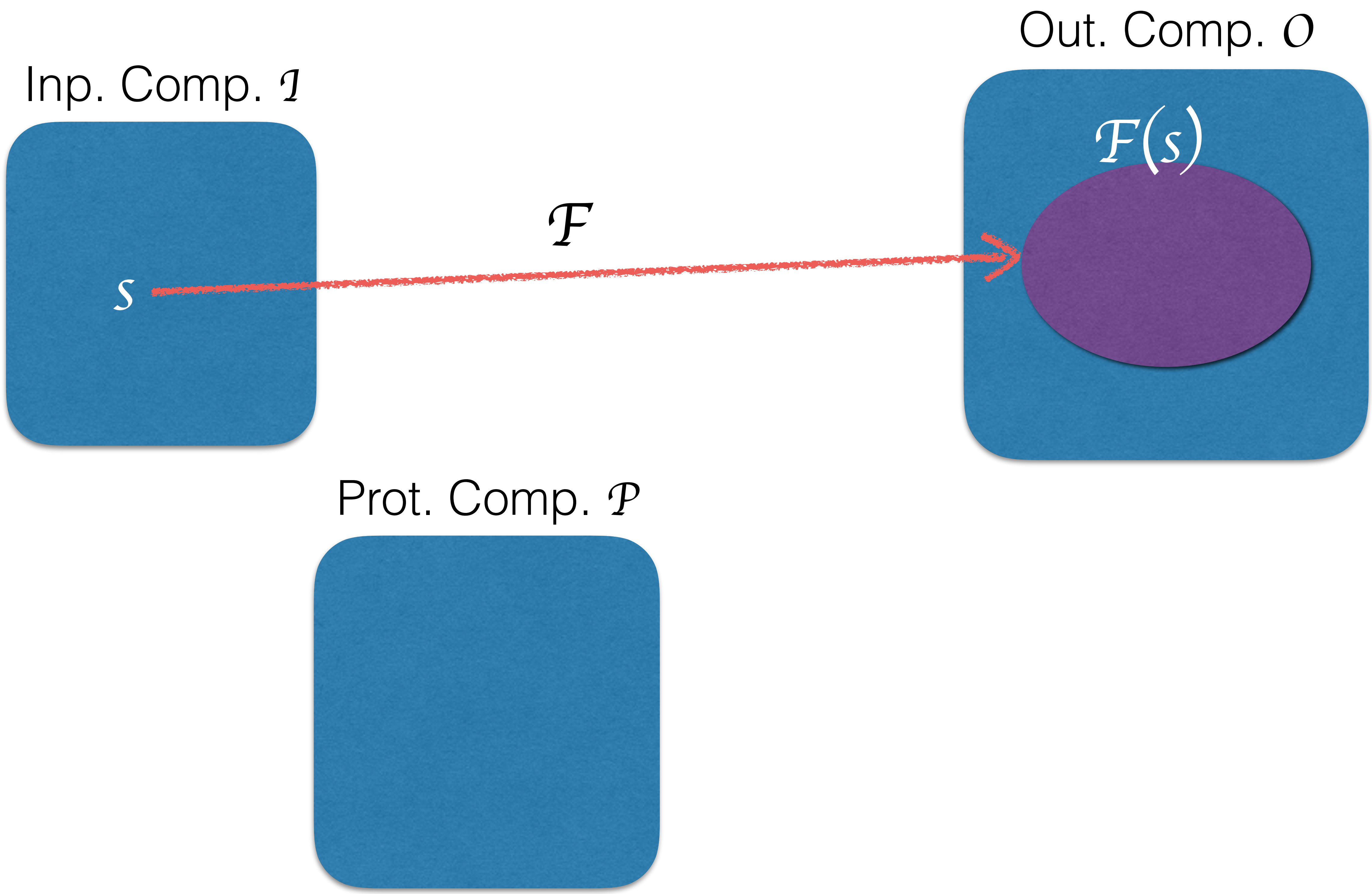


Solo executions

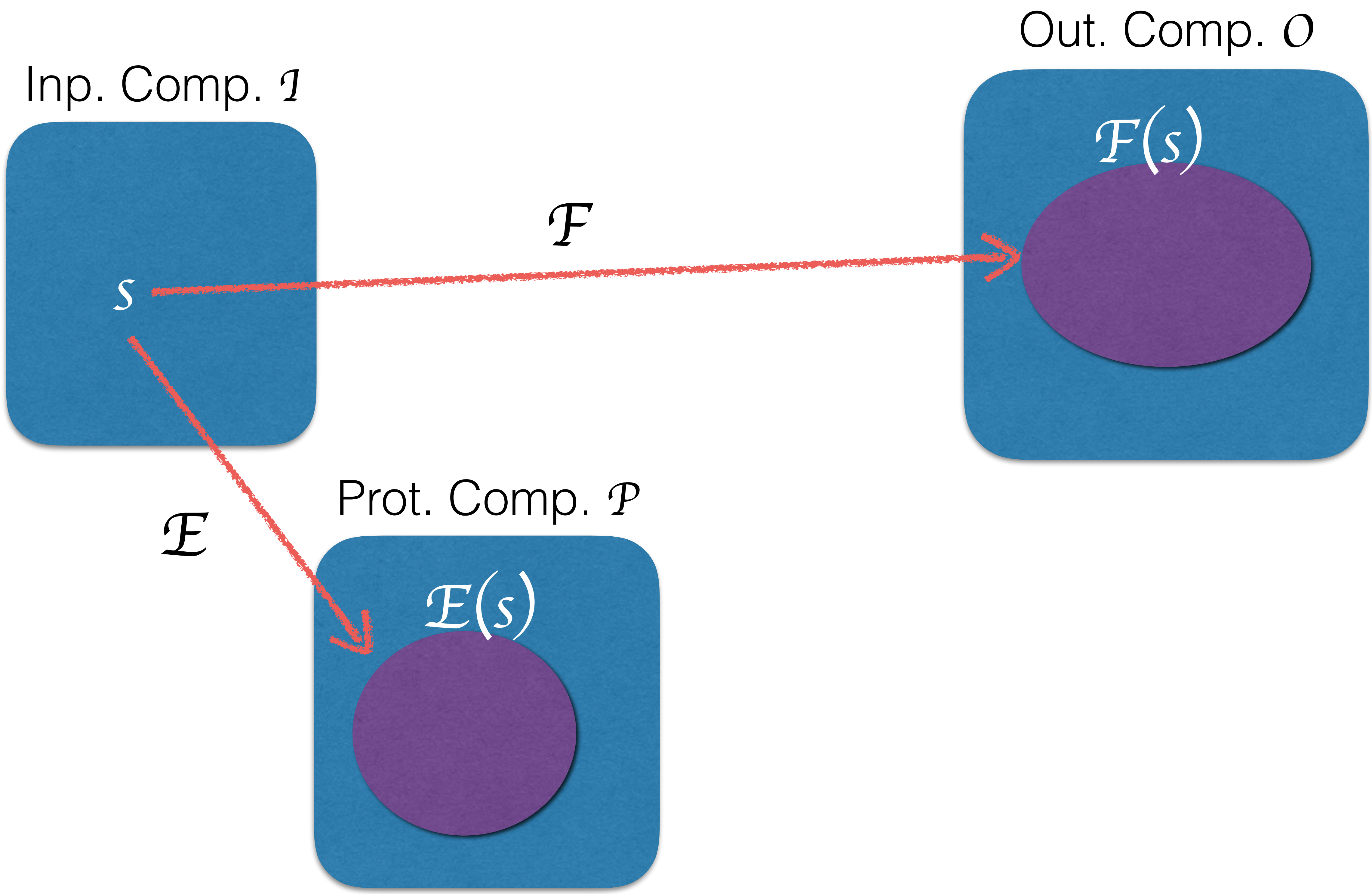
Solvability Condition



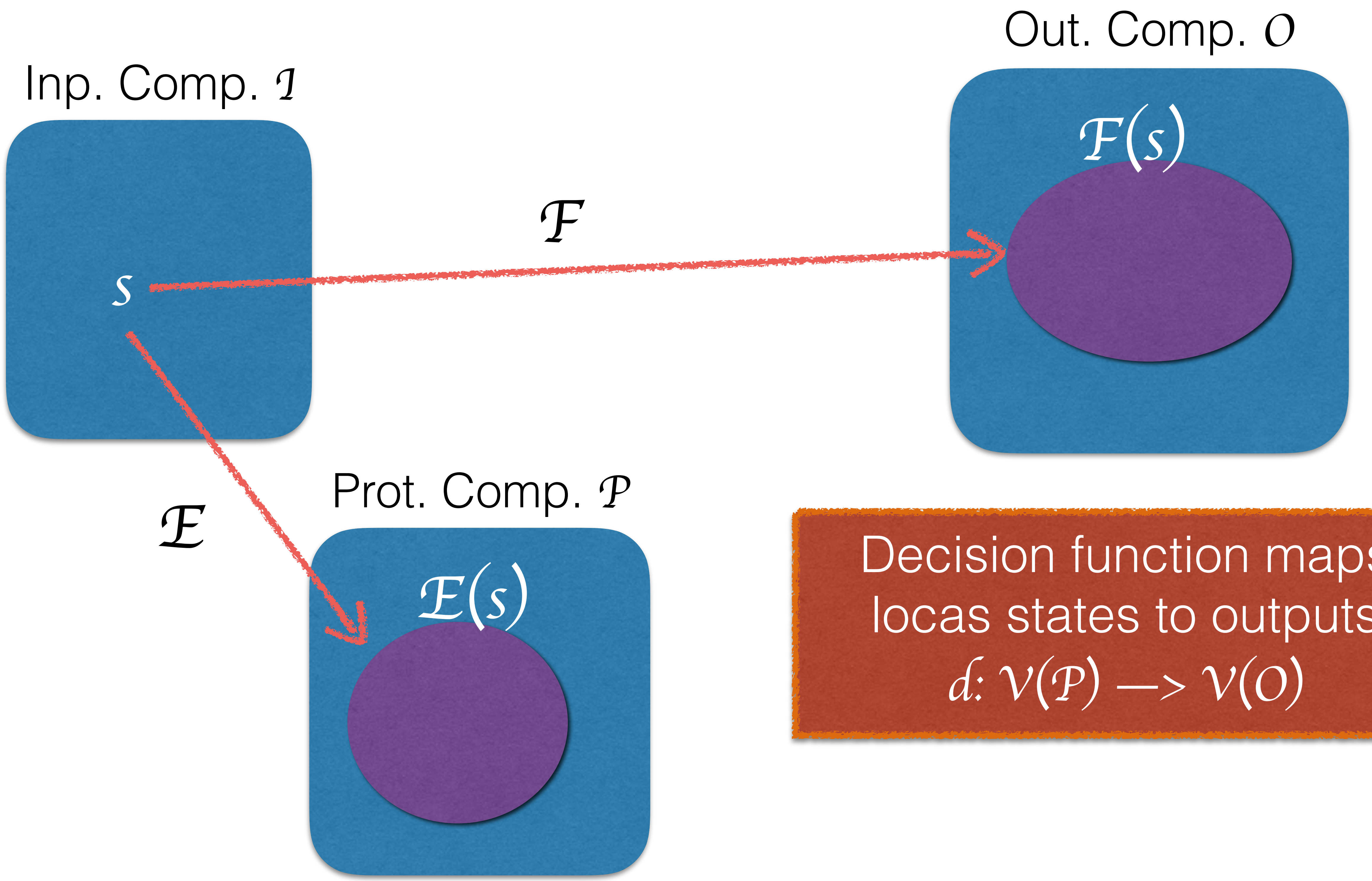
Solvability Condition



Solvability Condition

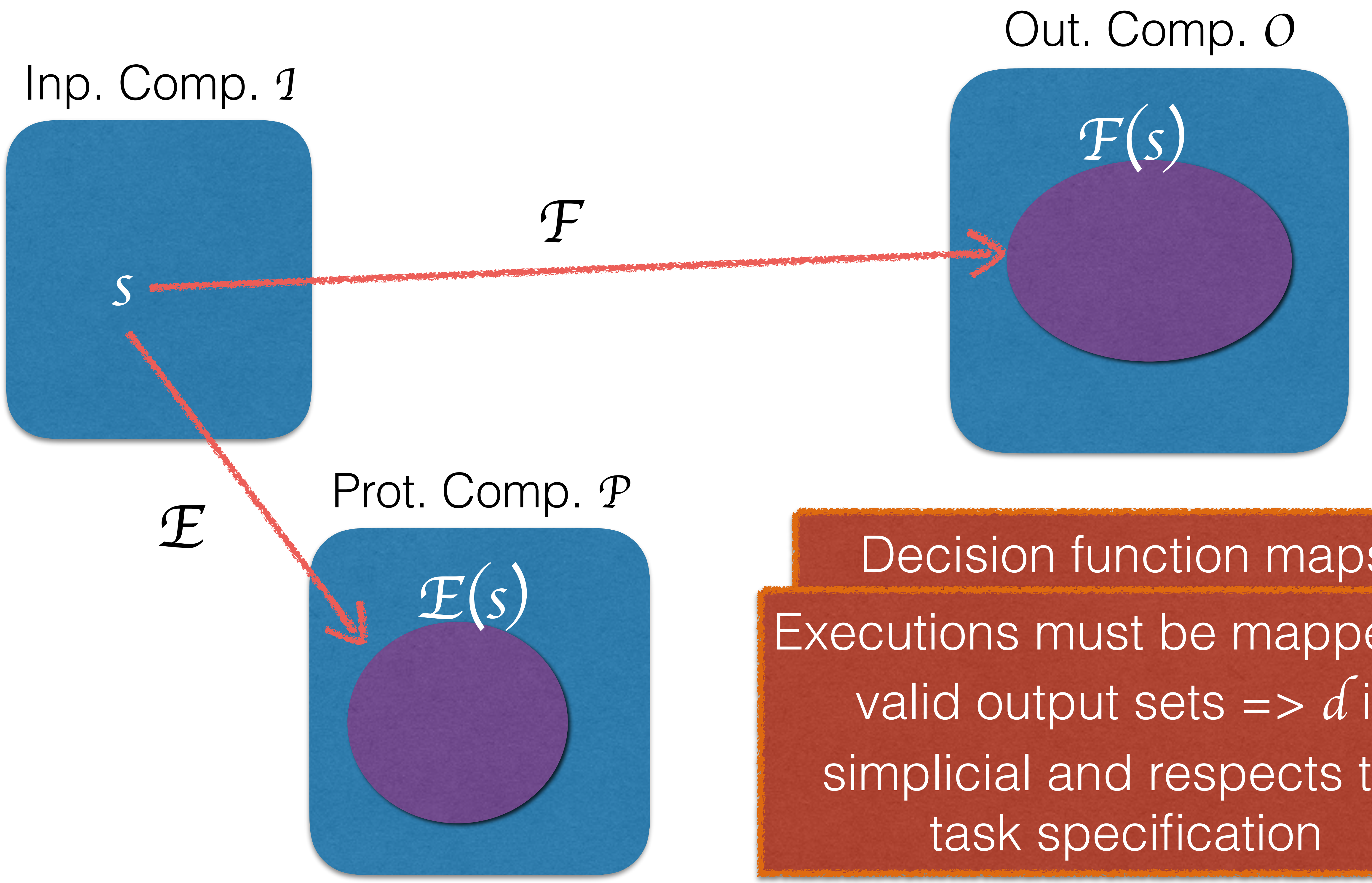


Solvability Condition



Decision function maps
locas states to outputs
$d: \mathcal{V}(\mathcal{P}) \text{—>} \mathcal{V}(\mathcal{O})$

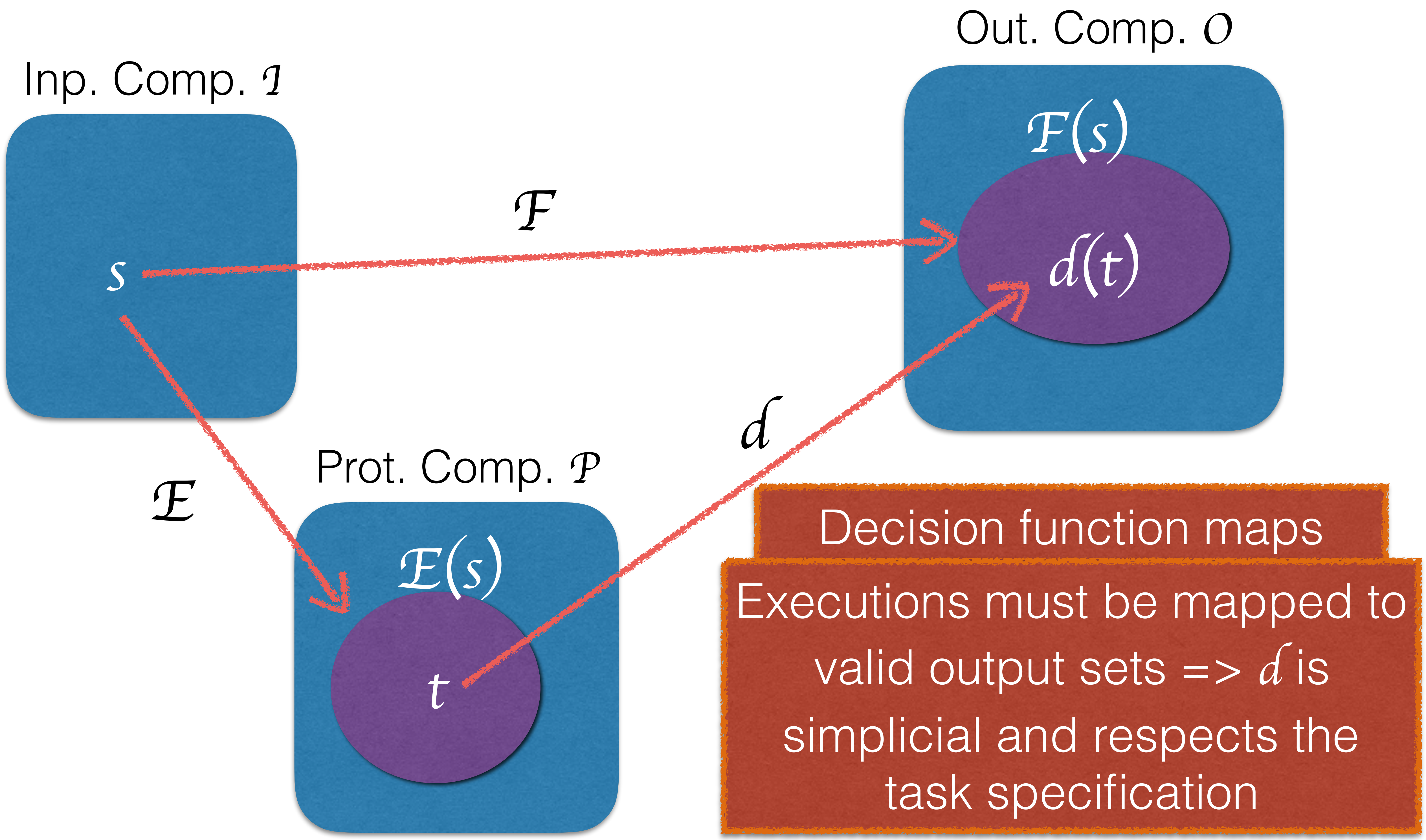
Solvability Condition



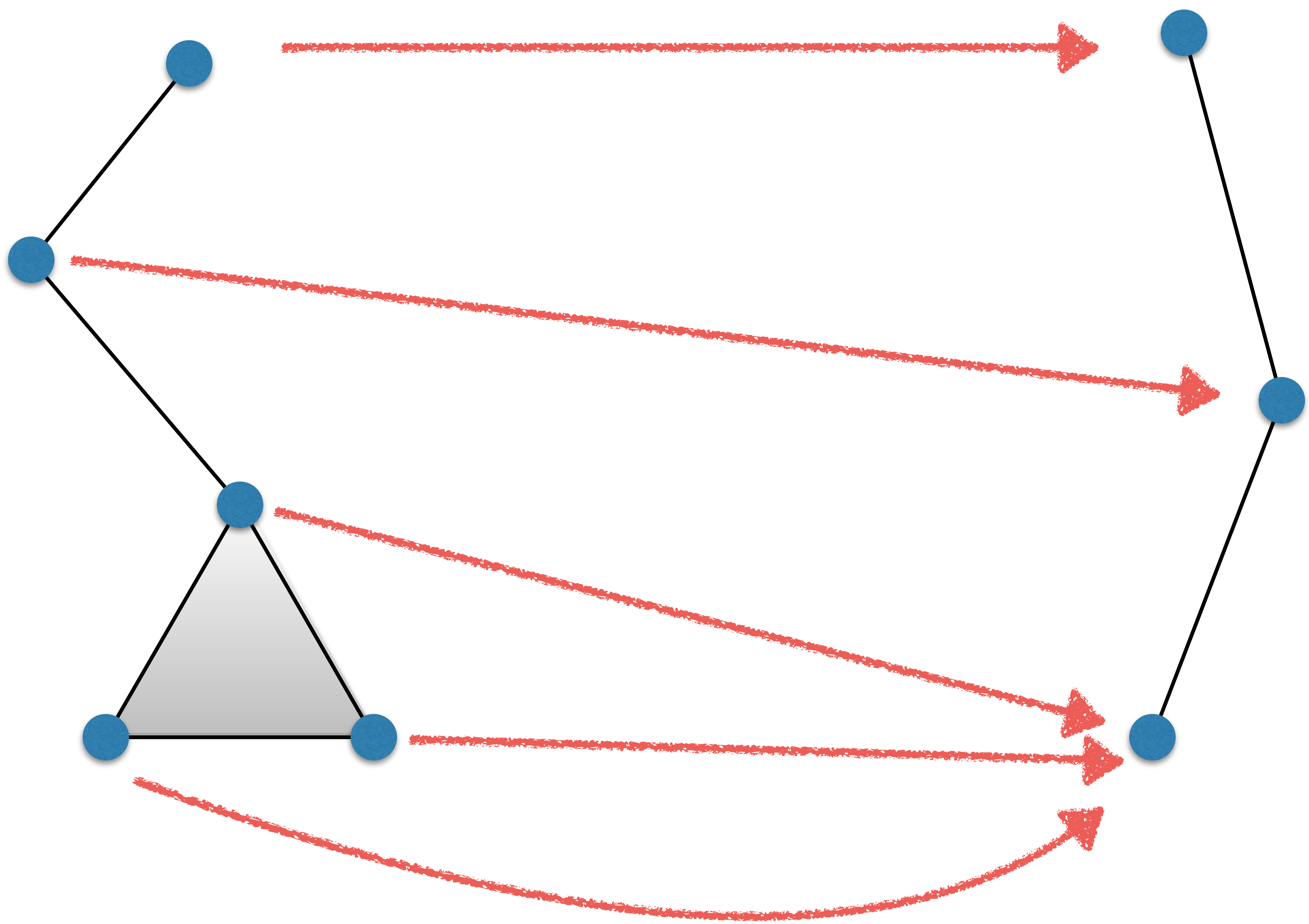
Decision function maps

Executions must be mapped to valid output sets => d is simplicial and respects the task specification

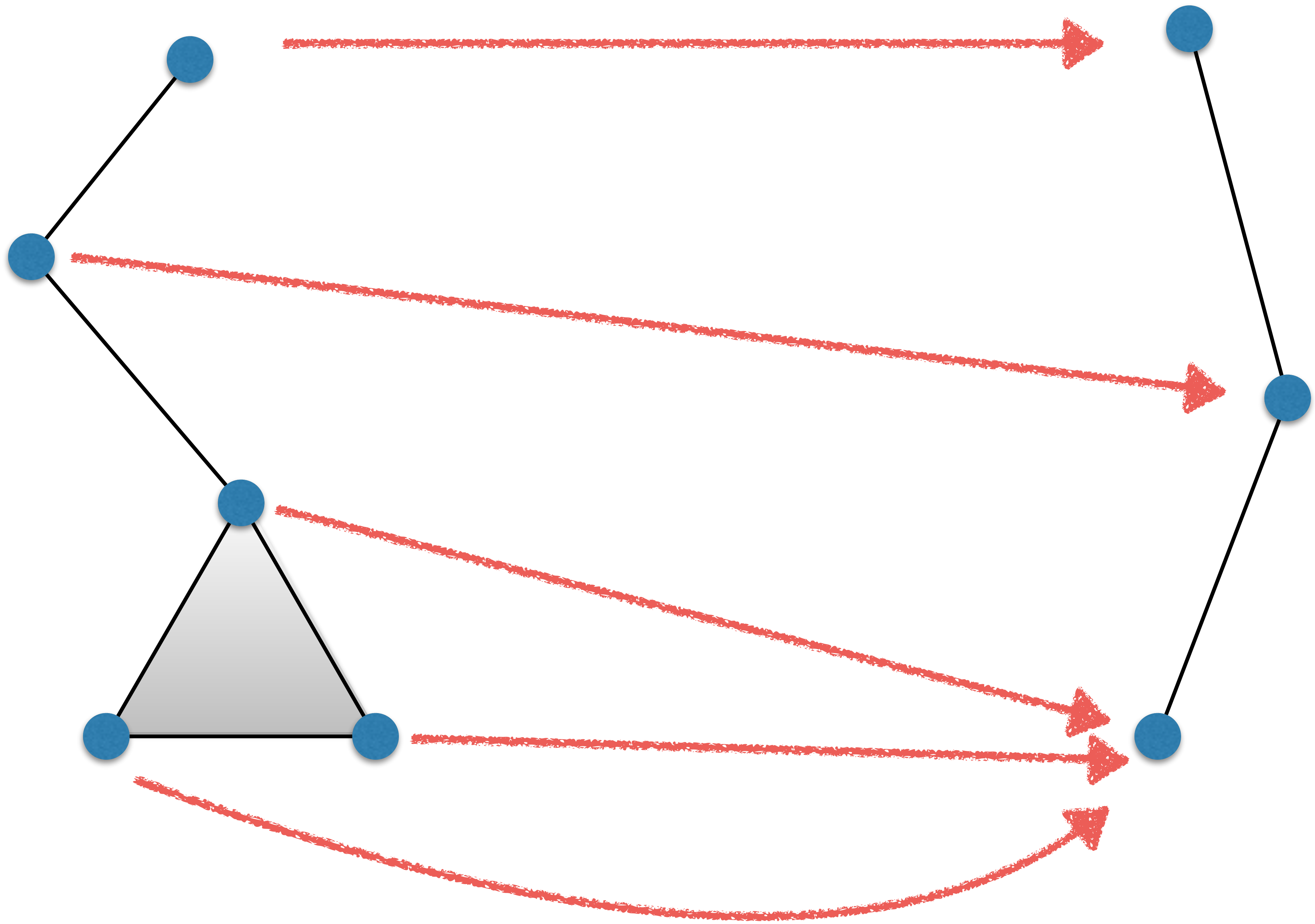
Solvability Condition



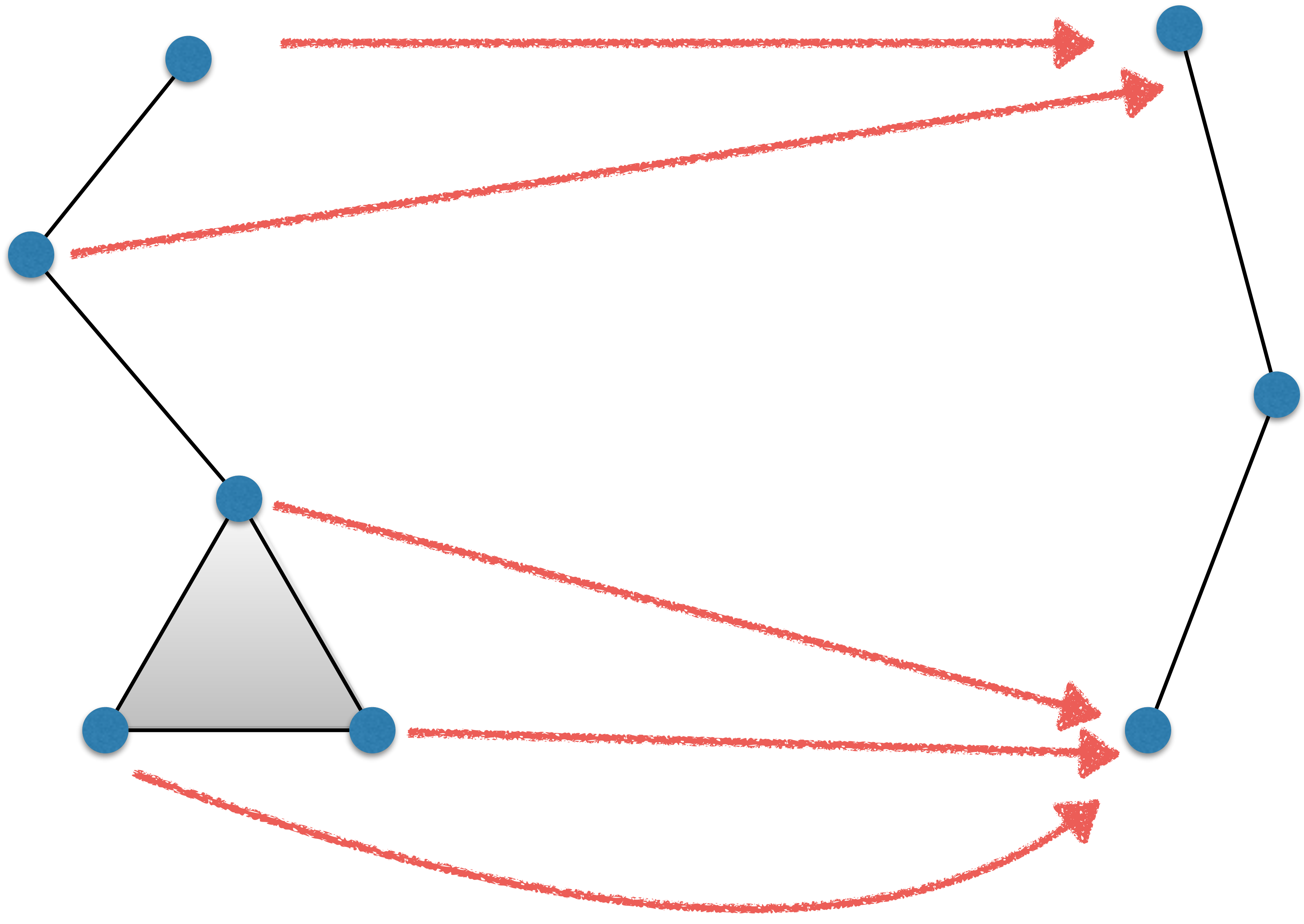
Simplicial Maps



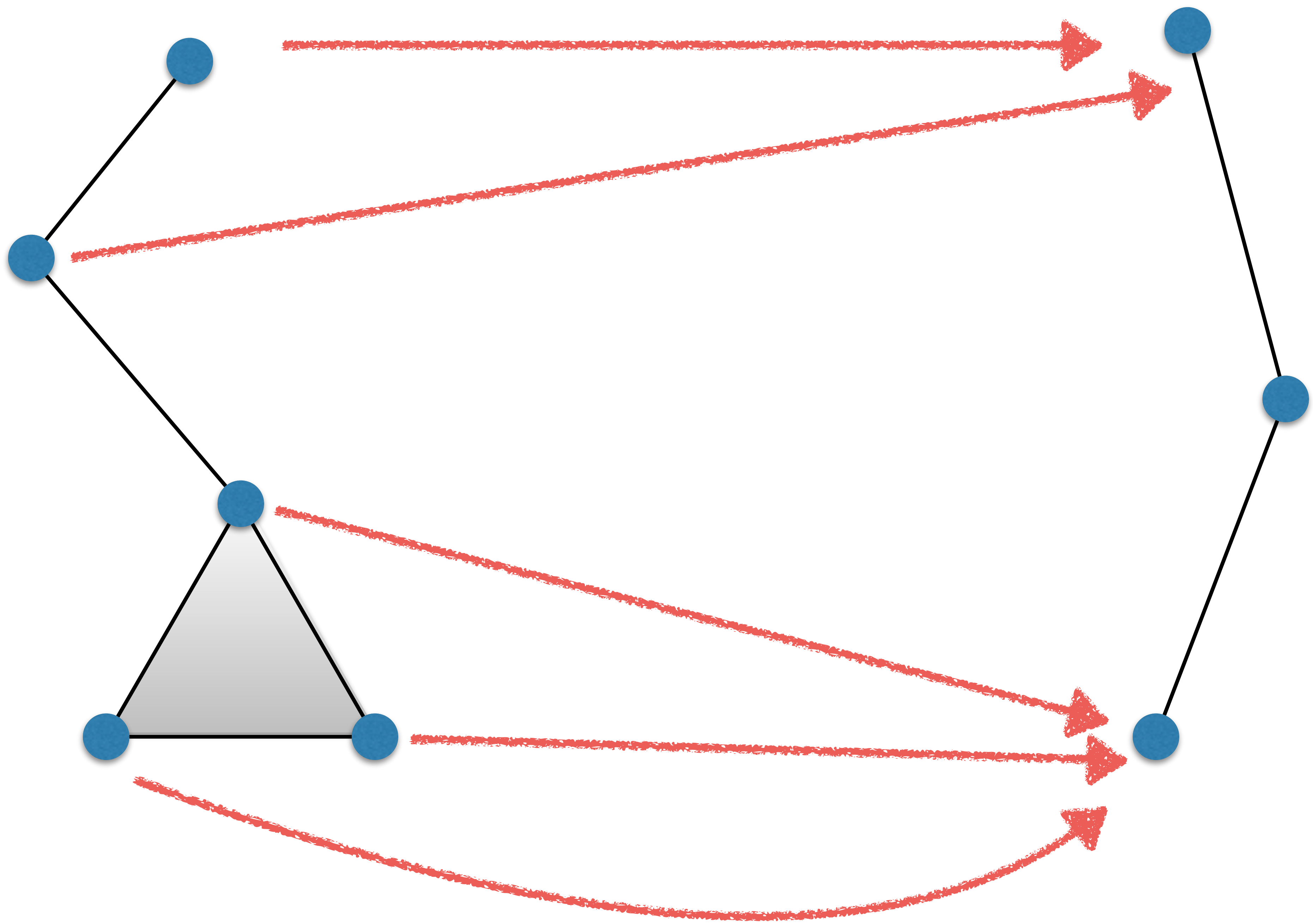
Simplicial Maps



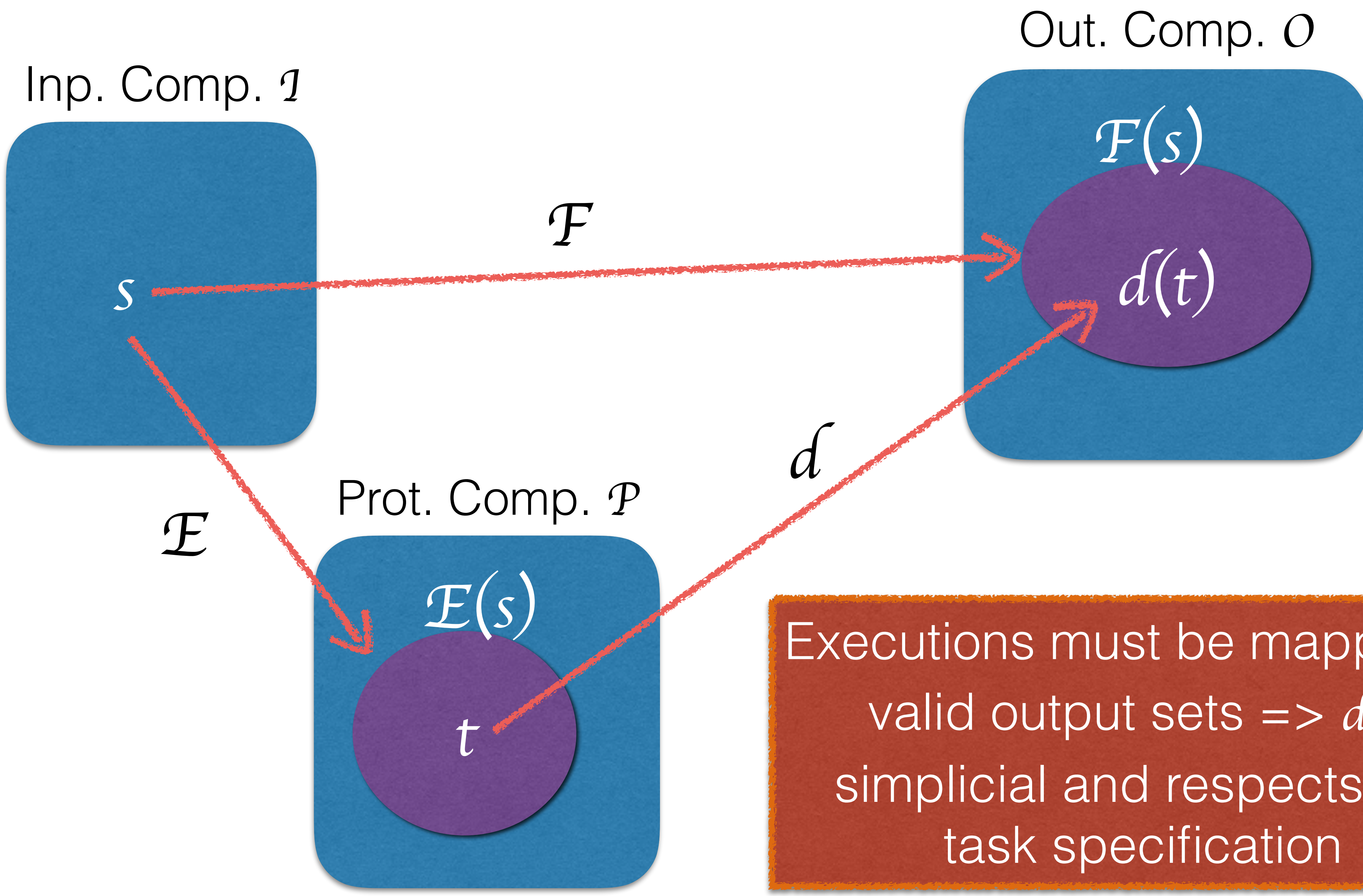
Simplicial Maps



Simplicial Maps



Solvability Condition

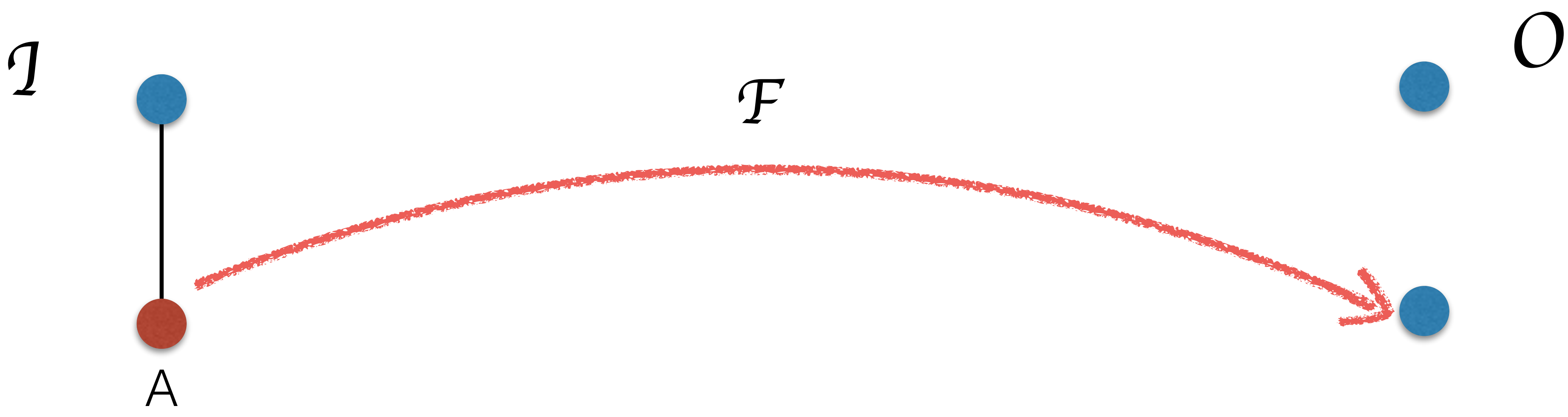


Executions must be mapped to valid output sets => d is simplicial and respects the task specification

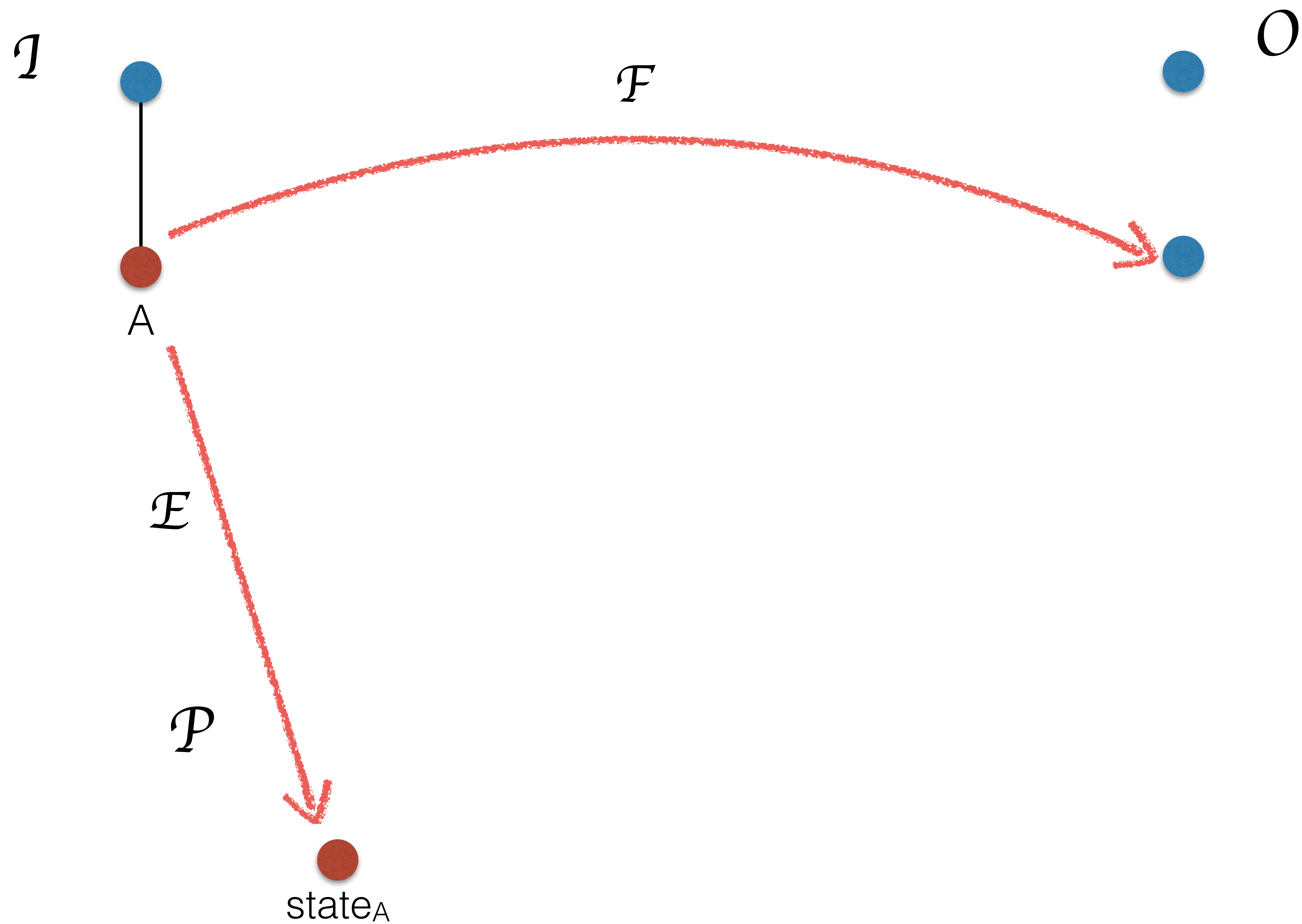
Gathering is Impossible

Gathering is Impossible

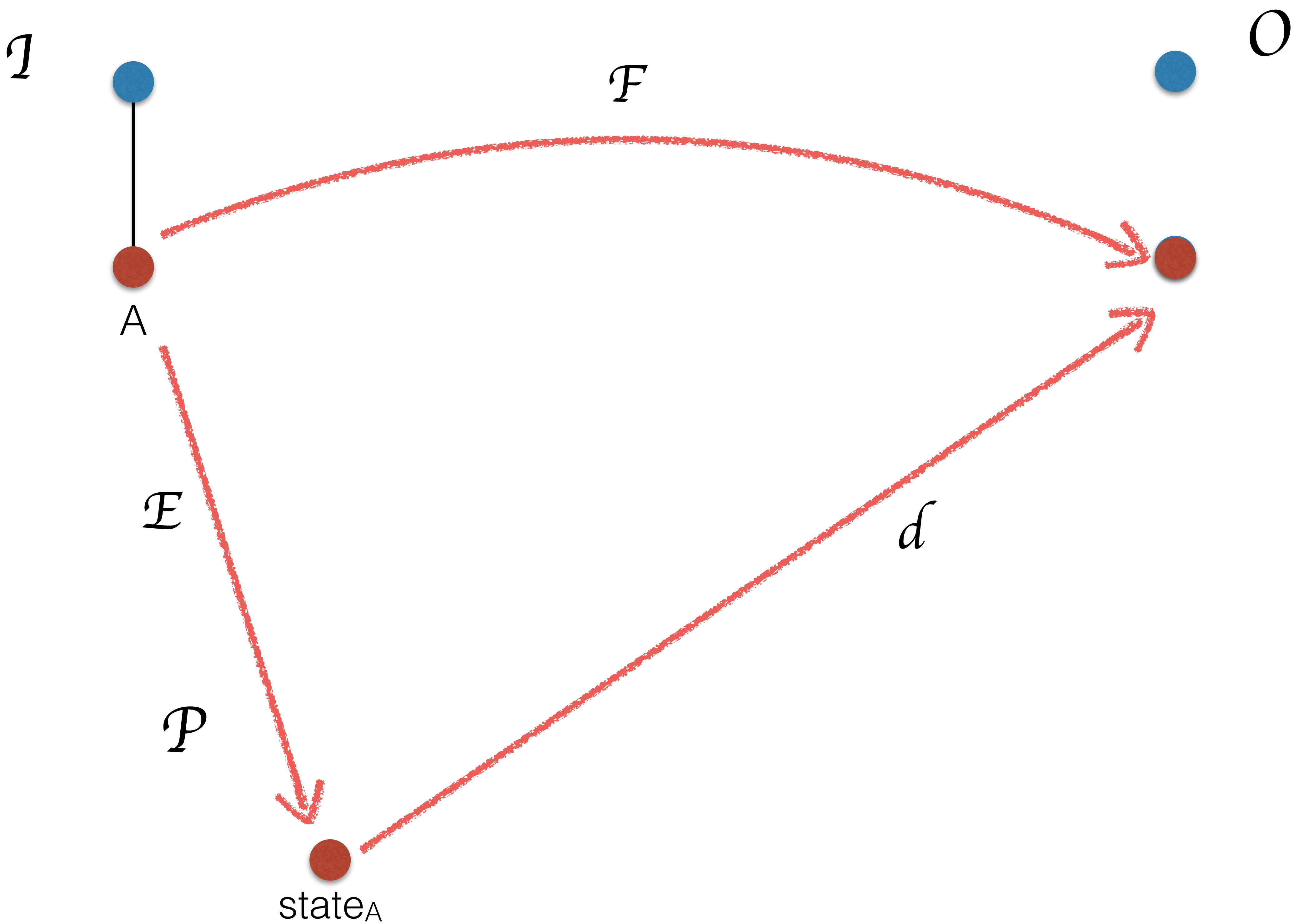
Gathering is Impossible



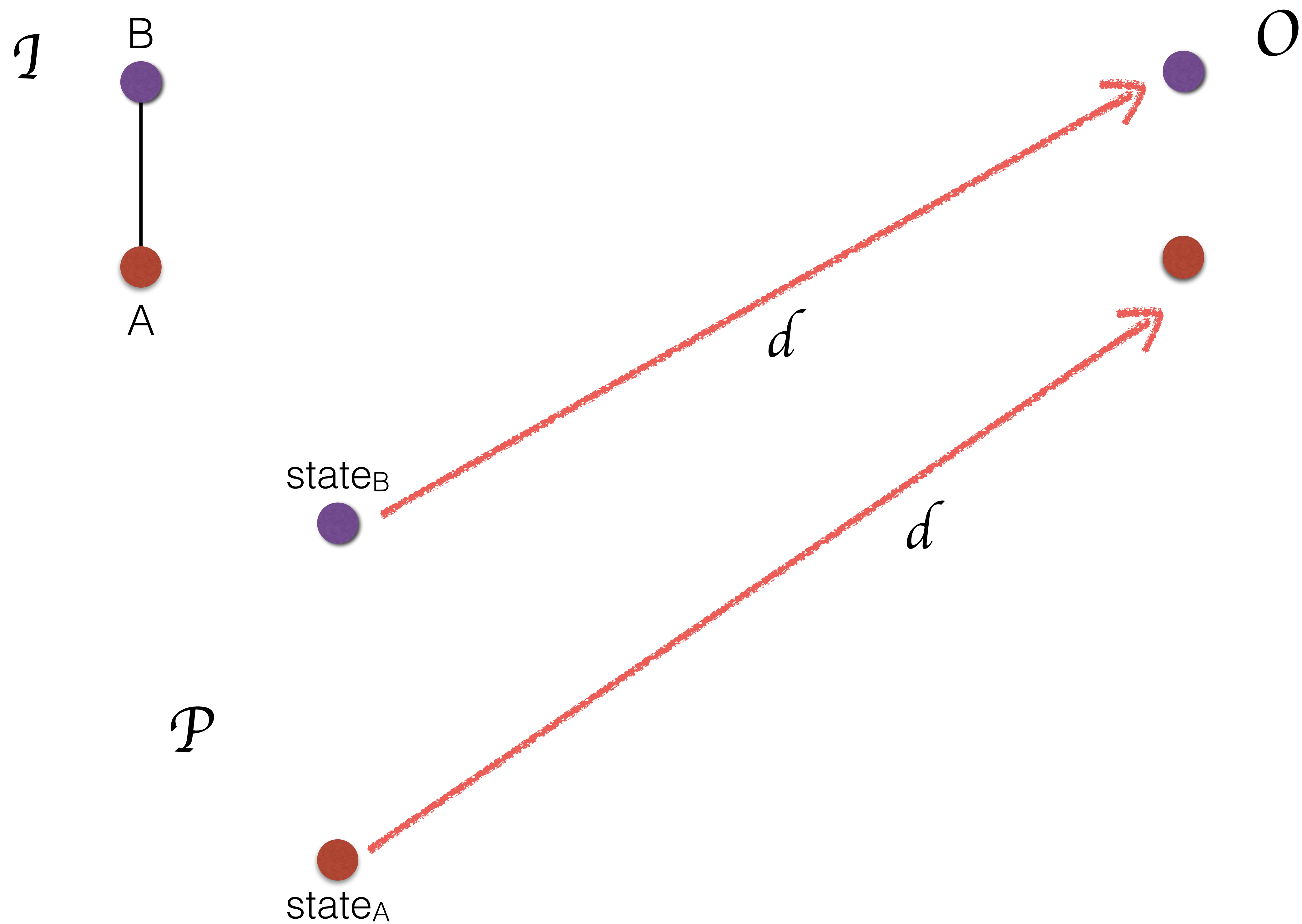
Gathering is Impossible



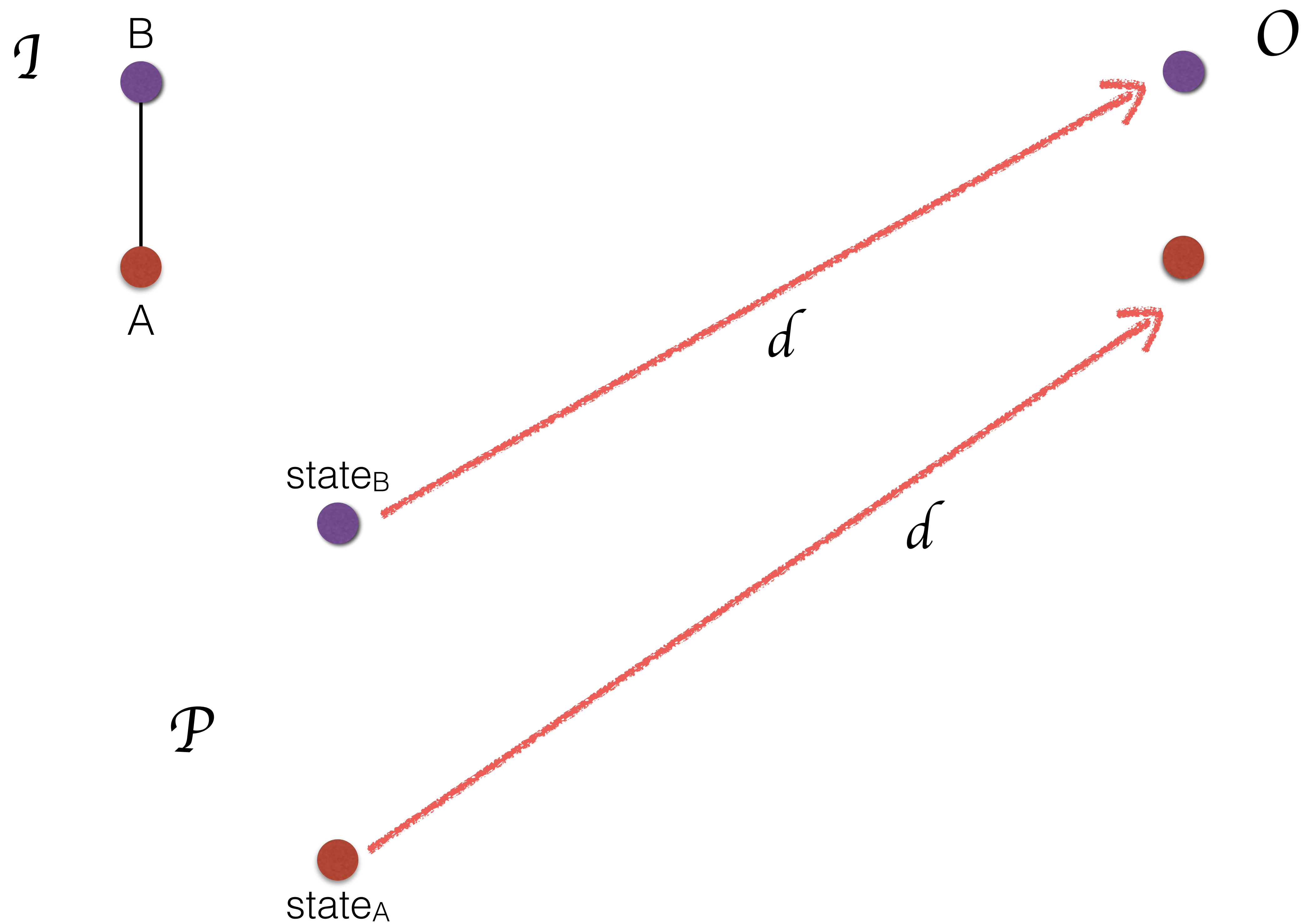
Gathering is Impossible



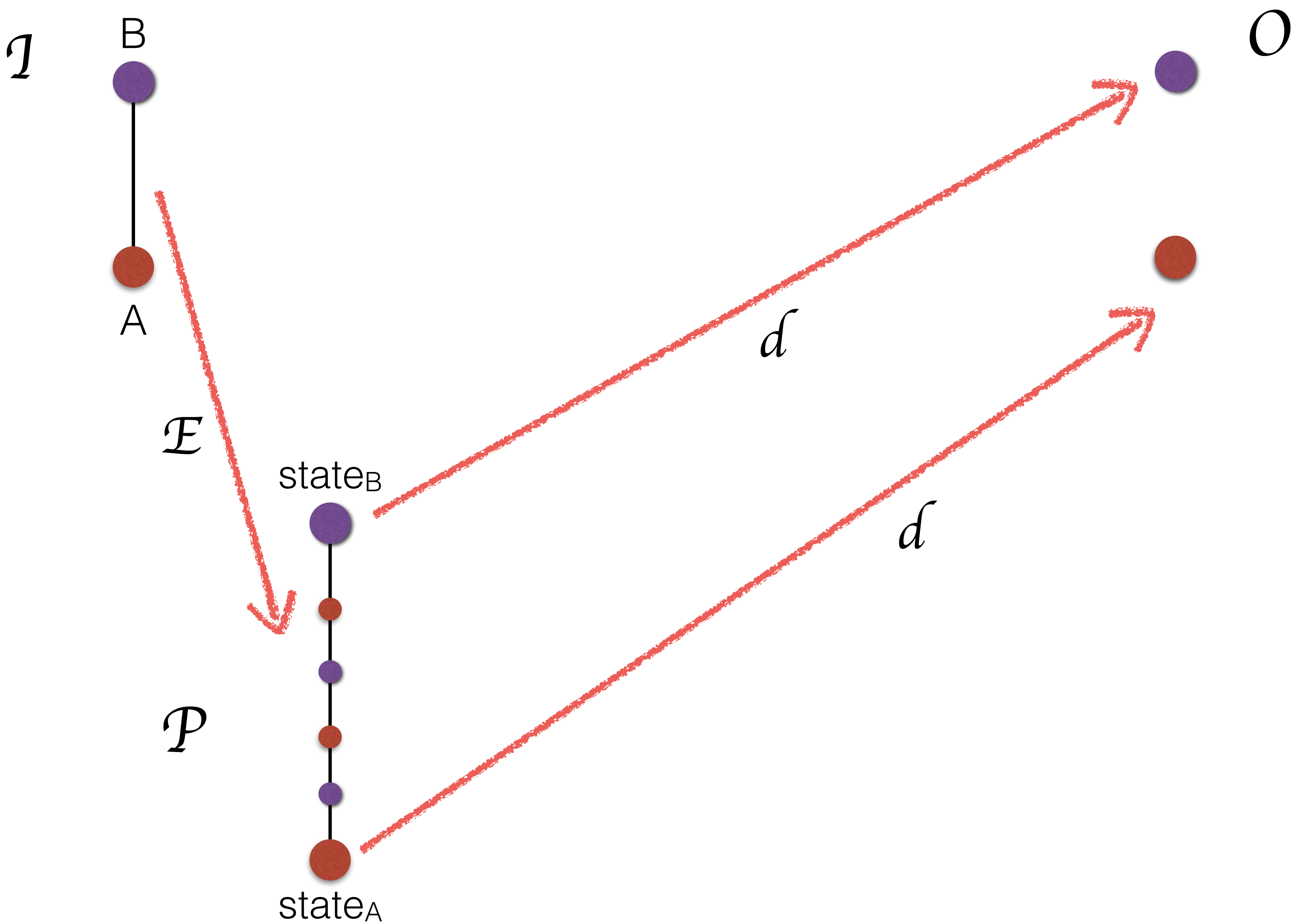
Gathering is Impossible



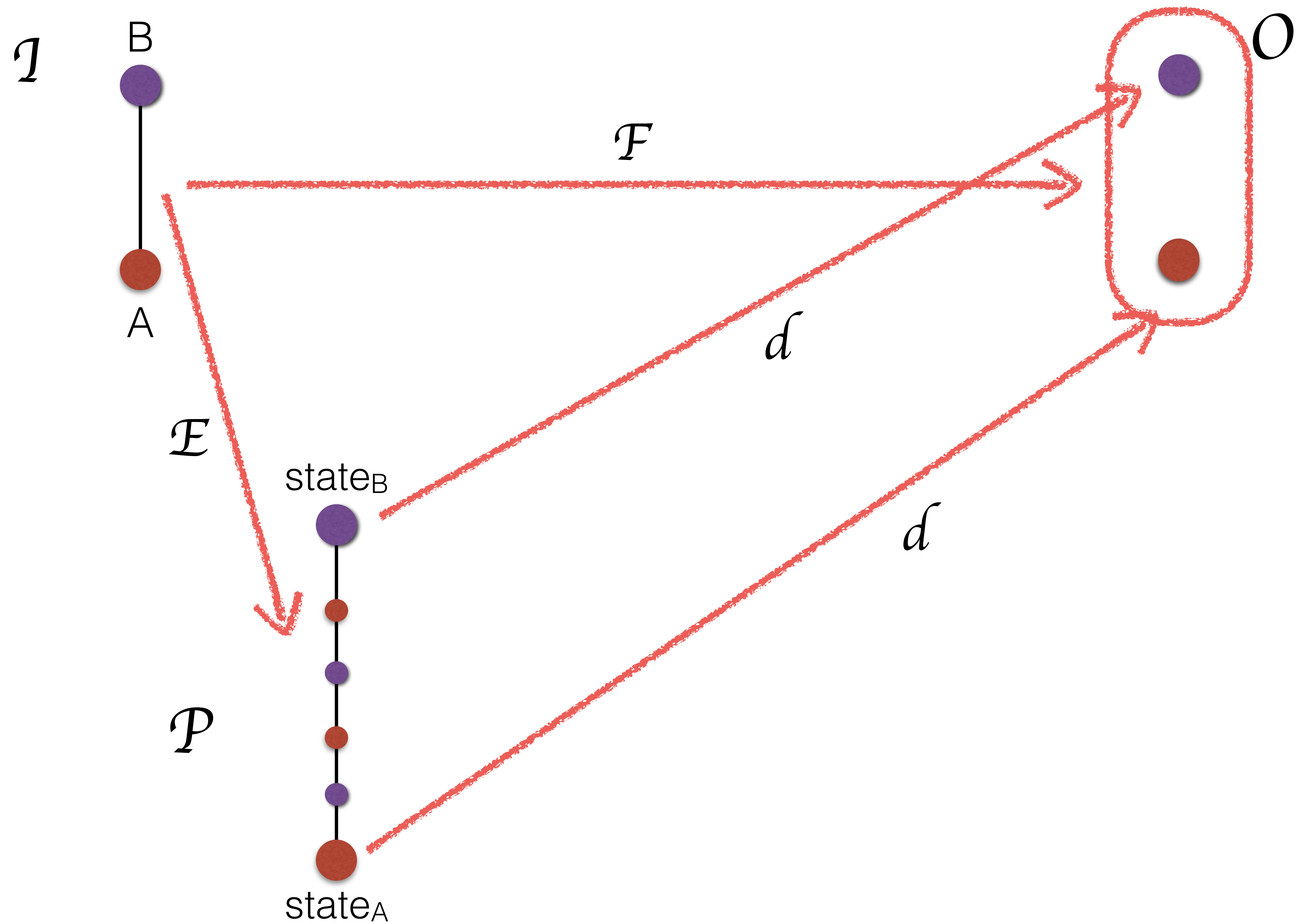
Gathering is Impossible



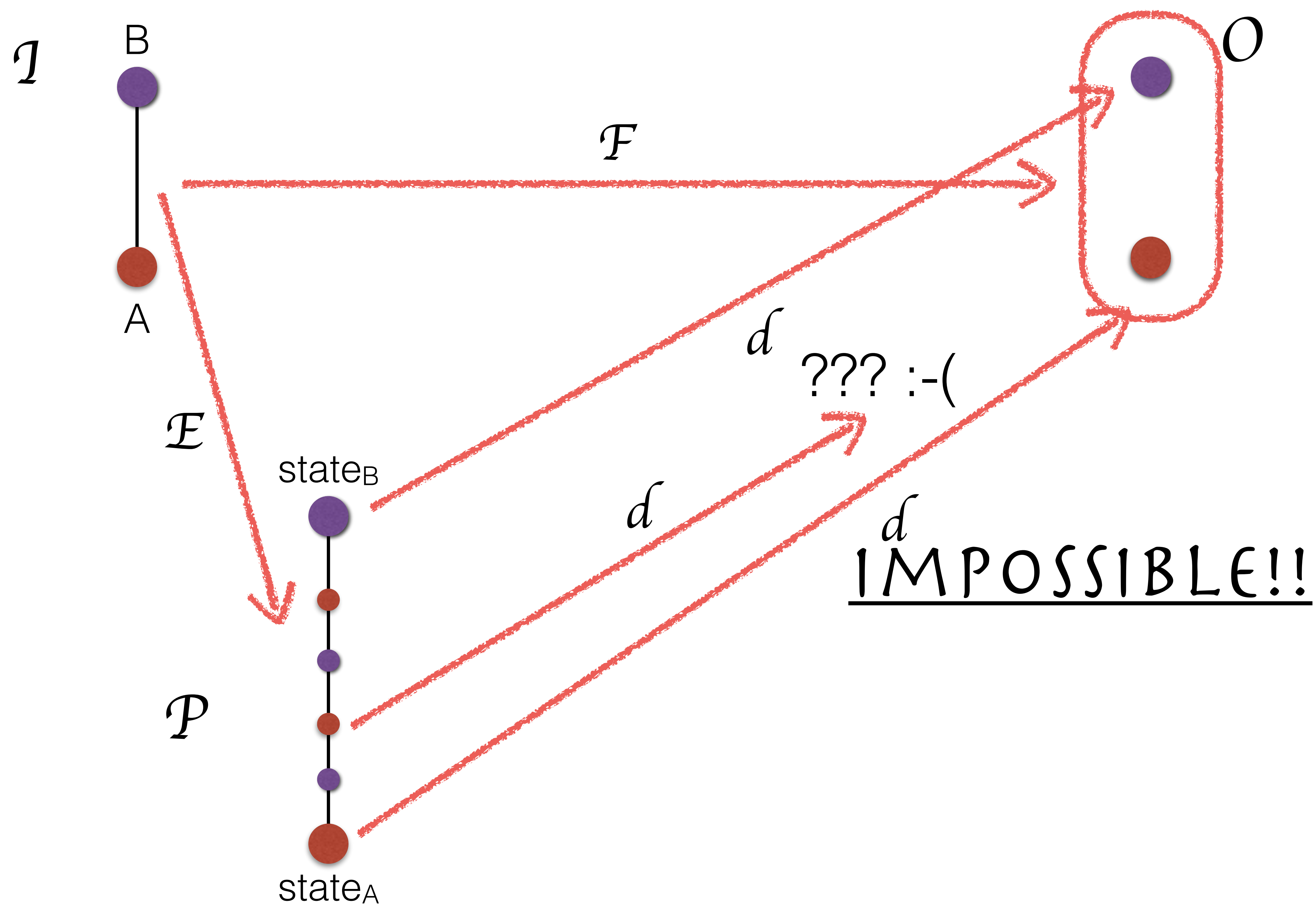
Gathering is Impossible



Gathering is Impossible



Gathering is Impossible



So … Relax

- What about gathering on an edge?
- **Edge Gathering**:
  - **Termination.** Correct robots decide a vertex.
  - **Validity.** If participating robots start on the same vertex, they stay there. If start on an edge, decide vertices of the edge.
  - **Edge Agreement.** Decided vertices belong to an edge (it could be the same vertex).

Solvability of Edge Gathering

For $n = 2$, edge gathering is solvable on any connected base graph $\mathcal{G}$

For $n > 2$, edge gathering is solvable if and only if the base graph $\mathcal{G}$ is acyclic

Edge Gathering on Trees

Algorithm 1 Edge Gathering for $N \geq 2$ robots on any tree $T = (V, E)$. Code for robot p_i.

Function GatheringTree(v_i, T)

Move$(v_i, 0)$ % p_i becomes visible to the others
for $r_i \leftarrow 1$ **to** $diam(T) - 1$ **do**
    $view_i \leftarrow$ **Look**(T) % positions and lights states of the others
    $max\_round_i \leftarrow \max\{r_j : (*, r_j) \in view_i\}$
    $S_i \leftarrow \{v_j : (v_j, max\_round_i) \in view_i \vee v_j = v_i\}$ % max round position and position of p_i
    $T_i \leftarrow$ smallest subtree of T spanning all vertices in S_i % subtree induced by positions in S_i
    if v_i is leaf of $T_i \wedge diam(T_i) > 0$ **then**
        $v_i \leftarrow$ vertex of T_i that is adjacent to v_i
    end if
    Move(v_i, r_i) % p_i makes visible its new position and updates its lights
end for
return v_i

Edge Gathering on Trees

Algorithm 1 Edge Gathering for $N \geq 2$ robots on any tree $T = (V, E)$. Code for robot p_i.

The farthest two robots might be

Function GatheringTree(v_i, T)

Move$(v_i, 0)$ % p_i becomes visible to the others
for $r_i \leftarrow 1$ **to** $diam(T) - 1$ **do**
    $view_i \leftarrow$ **Look**(T) % positions and lights states of the o
    $max\_round_i \leftarrow \max\{r_j : (*, r_j) \in view_i\}$
    $S_i \leftarrow \{v_j : (v_j, max\_round_i) \in view_i \vee v_j = v_i\}$ % max round position and position of p_i
    $T_i \leftarrow$ smallest subtree of T spanning all vertices in S_i % subtree induced by positions in S_i
    if v_i is leaf of $T_i \wedge diam(T_i) > 0$ **then**
        $v_i \leftarrow$ vertex of T_i that is adjacent to v_i
    end if
    Move(v_i, r_i) % p_i makes visible its new position and updates its lights
end for
return v_i

Edge Gathering on Trees

Algorithm 1 Edge Gathering for $N \geq 2$ robots on any tree $T = (V, E)$. Code for robot p_i.

Function GatheringTree(v_i, T)

```
Move(v_i, 0) % p_i becomes visible to the others
for r_i ← 1 to diam(T) − 1 do
    view_i ← Look(T) % positions and lights states of the others
    max_round_i ← max{r_j : (*, r_j) ∈ view_i}
    S_i ← {v_j : (v_j, max_round_i) ∈ view_i ∨ v_j = v_i} % max round p
    T_i ← smallest subtree of T spanning all vertices in S_i % subtree
    if v_i is leaf of T_i ∧ diam(T_i) > 0 then
        v_i ← vertex of T_i that is adjacent to v_i
    end if
    Move(v_i, r_i) % p_i makes visible its new position and updates its lights
end for
return v_i
```

Positions of leaders and mine

Edge Gathering on Trees

Algorithm 1 Edge Gathering for $N \geq 2$ robots on any tree $T = (V, E)$. Code for robot p_i.

Function GatheringTree(v_i, T)

Move$(v_i, 0)$ % p_i becomes visible to the others
for $r_i \leftarrow 1$ **to** $diam(T) - 1$ **do**
    $view_i \leftarrow$ **Look**(T) % positions and lights states of the others
    $max\_round_i \leftarrow \max\{r_j : (*, r_j) \in view_i\}$
    $S_i \leftarrow \{v_j : (v_j, max\_round_i) \in view_i \vee v_j = v_i\}$ % max round pos
    $T_i \leftarrow$ smallest subtree of T spanning all vertices in S_i ~~% subtree~~ in
    if v_i is leaf of $T_i \wedge diam(T_i) > 0$ **then**
        $v_i \leftarrow$ vertex of T_i that is adjacent to v_i
    end if
    Move(v_i, r_i) % p_i makes visible its new position and updates its lights
end for
return v_i

Tree with leaders and me

Edge Gathering on Trees

Algorithm 1 Edge Gathering for $N \geq 2$ robots on any tree $T = (V, E)$. Code for robot p_i.

Function GatheringTree(v_i, T)

```
Move(v_i, 0) % p_i becomes visible to the others
for r_i ← 1 to diam(T) − 1 do
    view_i ← Look(T) % positions and lights states of the others
    max_round_i ← max{r_j : (*, r_j) ∈ view_i}
    S_i ← {v_j : (v_j, max_round_i) ∈ view_i ∨ v_j = v_i} % max round p
    T_i ← smallest subtree of T spanning all vertices in S_i % subtre
    if v_i is leaf of T_i ∧ diam(T_i) > 0 then
        v_i ← vertex of T_i that is adjacent to v_i
    end if
    Move(v_i, r_i) % p_i makes visible its new position and updates
end for
return v_i
```

Need to move?

Edge Gathering on Trees

Algorithm 1 Edge Gathering for $N \geq 2$ robots on any tree $T = (V, E)$. Code for robot p_i.

```
Function GatheringTree(v_i, T)
Move(v_i, 0) % p_i becomes visible to the others
for r_i ← 1 to diam(T) − 1 do
    view_i ← Look(T) % positions and lights states of the others
    max_round_i ← max{r_j : (*, r_j) ∈ view_i}
    S_i ← {v_j : (v_j, max_round_i) ∈ view_i ∨ v_j = v_i} % max round position and position of p_i
    T_i ← smallest subtree of T spanning all vertices in S_i % subtree induced by positions in S_i
    if v_i is leaf of T_i ∧ diam(T_i) > 0 then
        v_i ← vertex of T_i that is adjacent to v_i
    end if
    Move(v_i, r_i) ~~% p_i makes visible its new~~ p ... ats
end for
return v_i
```

Move and update light

Edge Gathering on Trees

Algorithm 1 Edge Gathering for $N \geq 2$ robots on any tree $T = (V, E)$. Code for robot p_i.

Function GatheringTree(v_i, T)

Move$(v_i, 0)$ % p_i becomes visible to the others
for $r_i \leftarrow 1$ **to** $diam(T) - 1$ **do**
    $view_i \leftarrow$ **Look**(T) % positions and lights states of the others
    $max\_round_i \leftarrow \max\{r_j : (*, r_j) \in view_i\}$
    $S_i \leftarrow \{v_j : (v_j, max\_round_i) \in view_i \vee v_j = v_i\}$ % max round position and position of p_i
    $T_i \leftarrow$ smallest subtree of T spanning all vertices in S_i % subtree induced by positions in S_i
    if v_i is leaf of $T_i \wedge diam(T_i) > 0$ **then**
        $v_i \leftarrow$ vertex of T_i that is adjacent to v_i
    end if
    Move(v_i, r_i) % p_i makes visible its new position and updates its lights
end for
return v_i

Edge Agreement. For every prefix of an execution:
dist(pos(i), pos(j)) <= diam(T) - min{round(i),round(j)}

2-Robot Edge Gathering

1. **Precompute** a spanning tree $\mathcal{T}$ of $\mathcal{G}$
2. **Algorithm** $\mathcal{A}$: Algorithm for trees.
3. **For** $r=1$ to $diam(\mathcal{G})$ **do**
4.   **Look**($\mathcal{G}$)
5.   **If** distance of current positions on $\mathcal{G}$ > 1 **then**
6.   **Simulate** a round of $\mathcal{A}$ on $\mathcal{T}$
7.   **Move** to next vertex
8. **Return** current position

Cycles are Obstacles

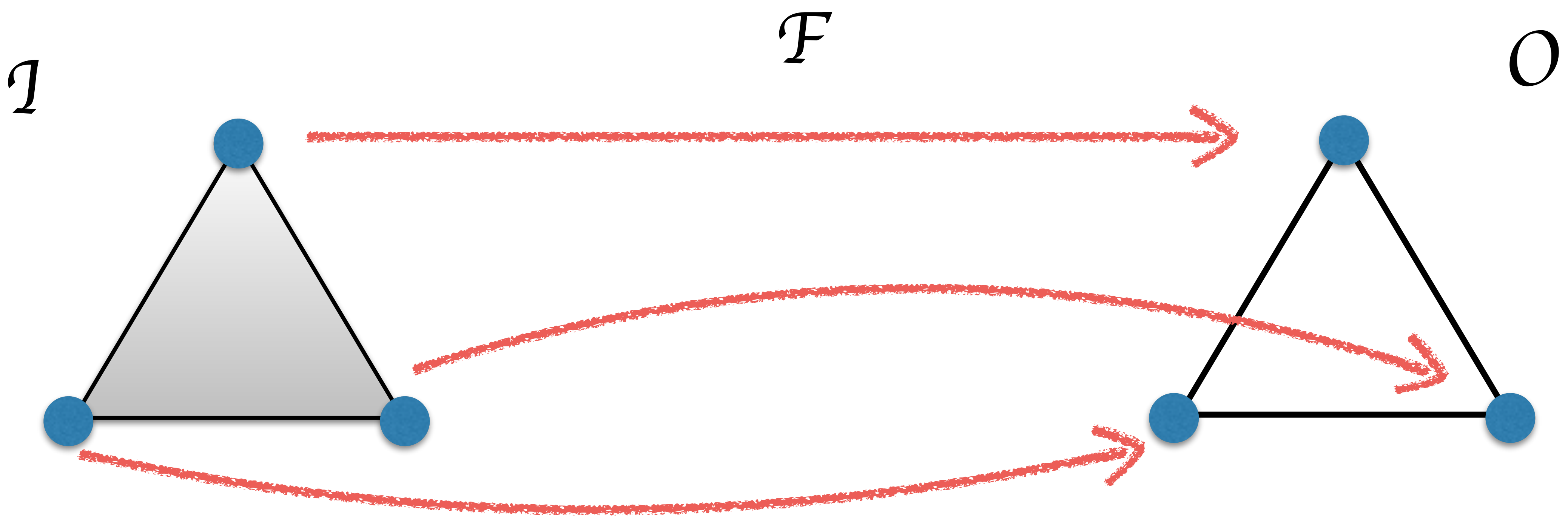
For n > 2, if the base graph $\mathcal{G}$ is has cycles, then edge gathering is unsolvable

Proof:

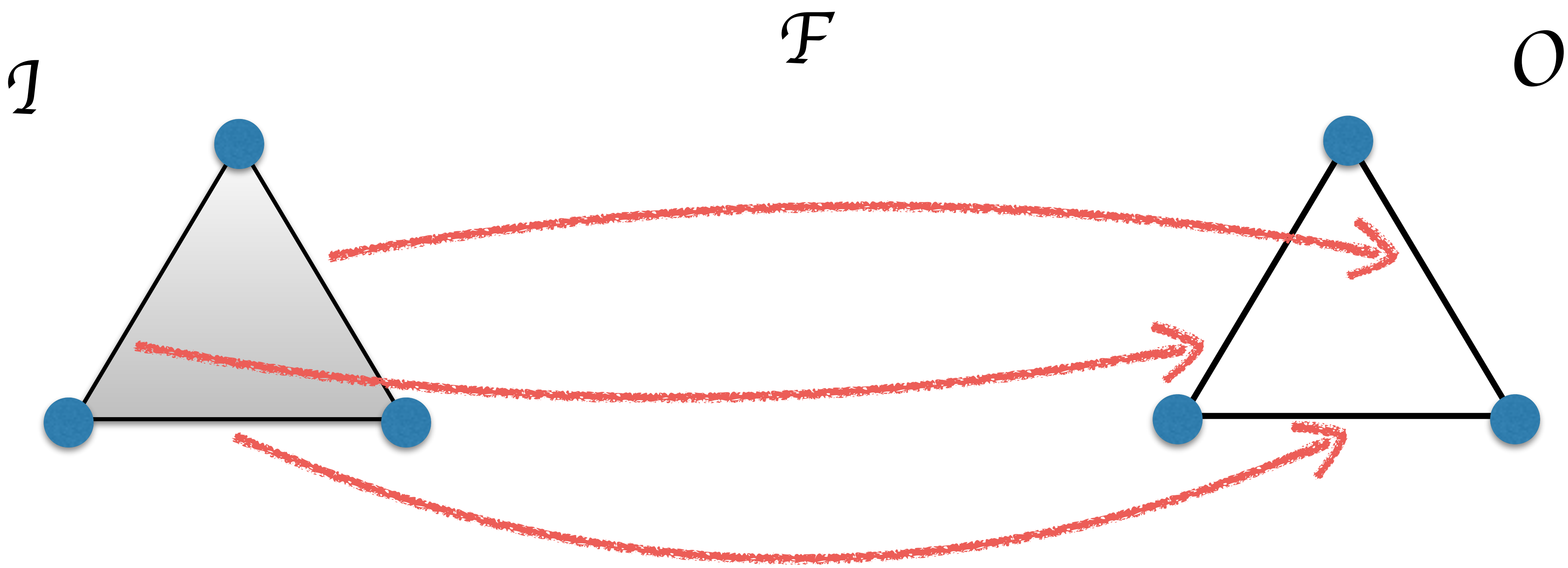
1. The case n= 3 is enough.
2. Prove the triangle is impossible.
3. Solve the triangle from any cyclic graph.

The Triangle is Impossible

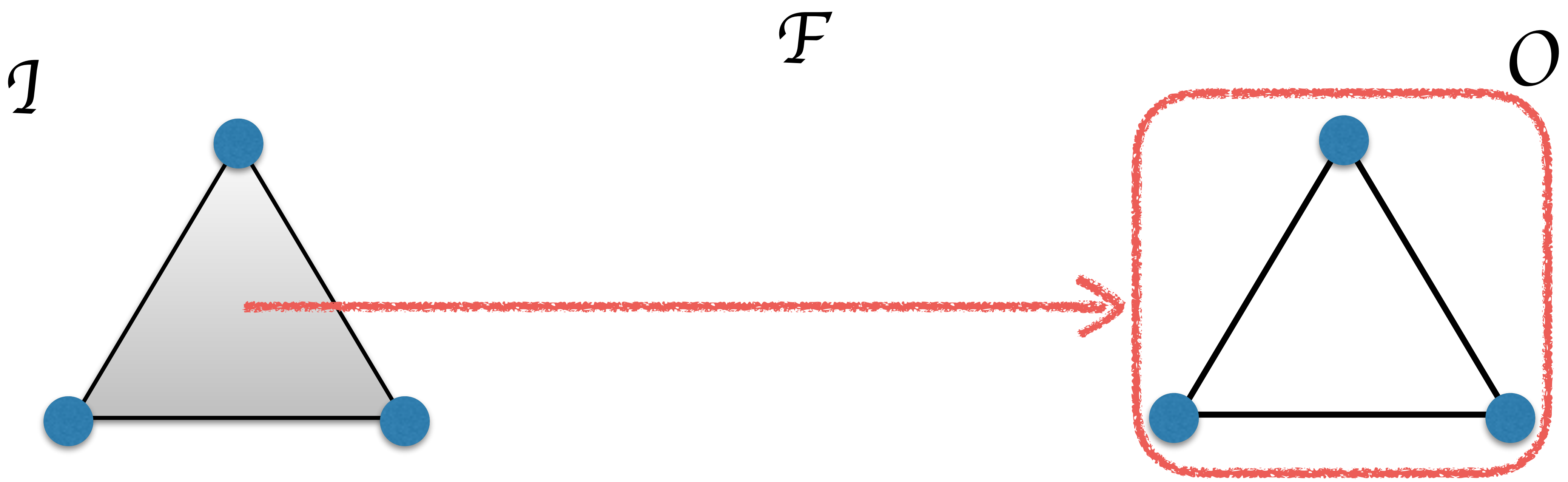
The Triangle is Impossible



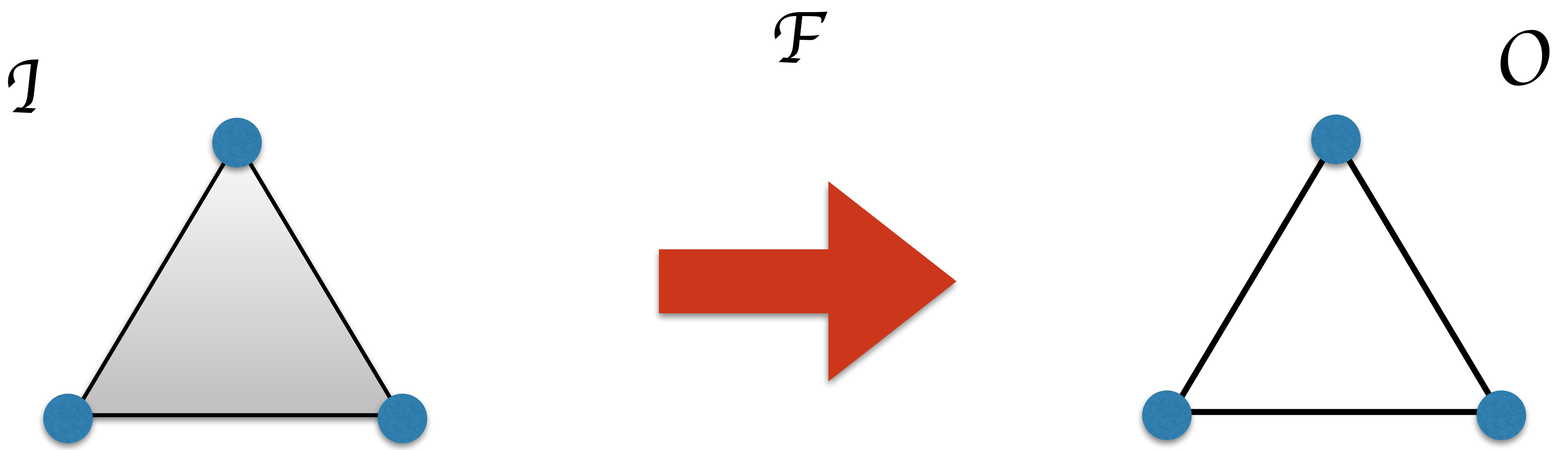
The Triangle is Impossible



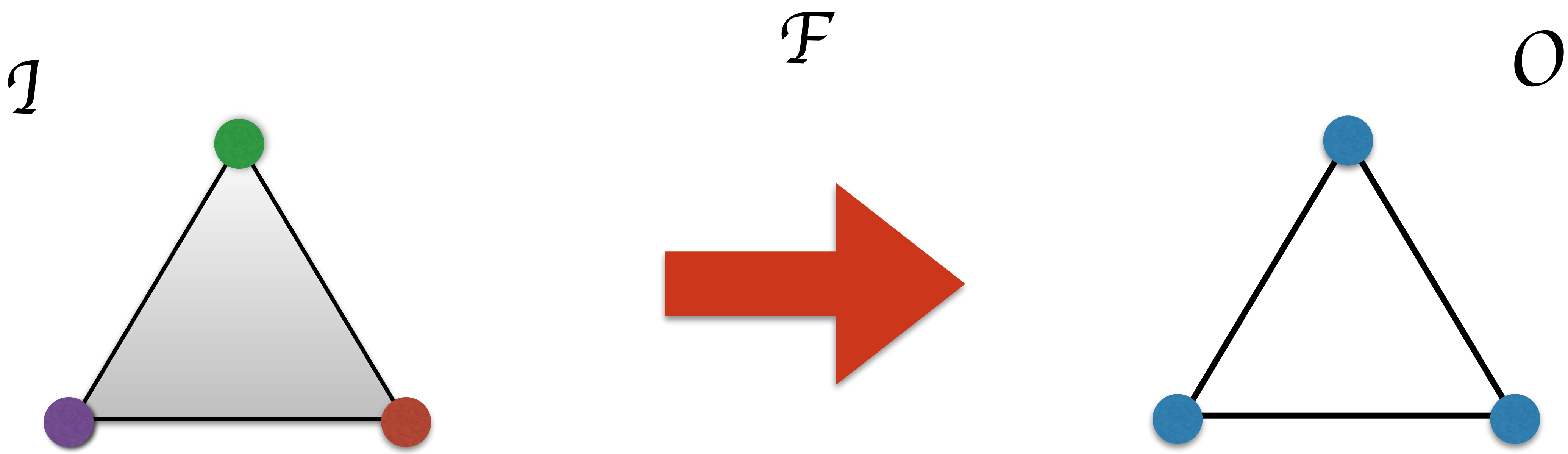
The Triangle is Impossible



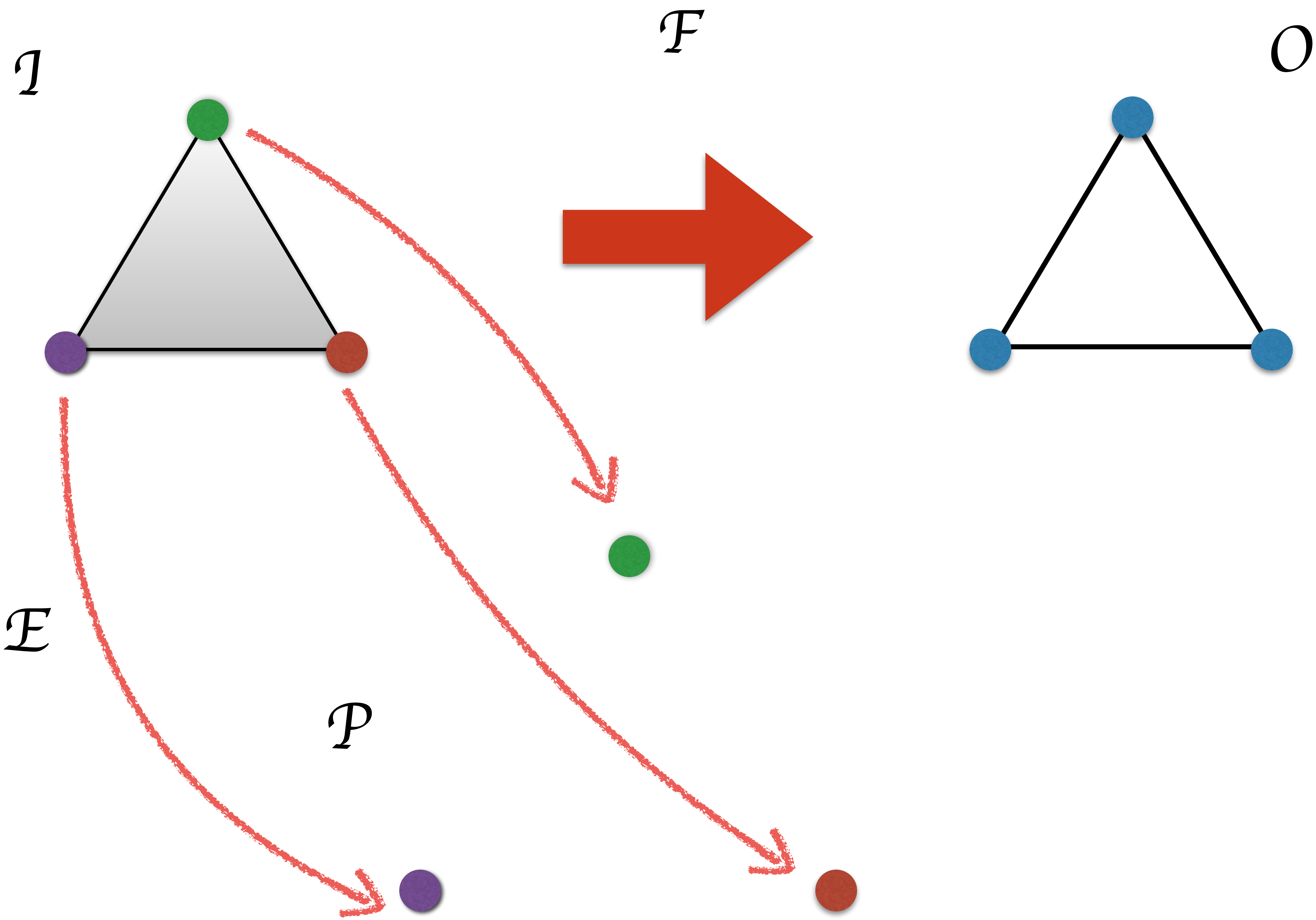
The Triangle is Impossible



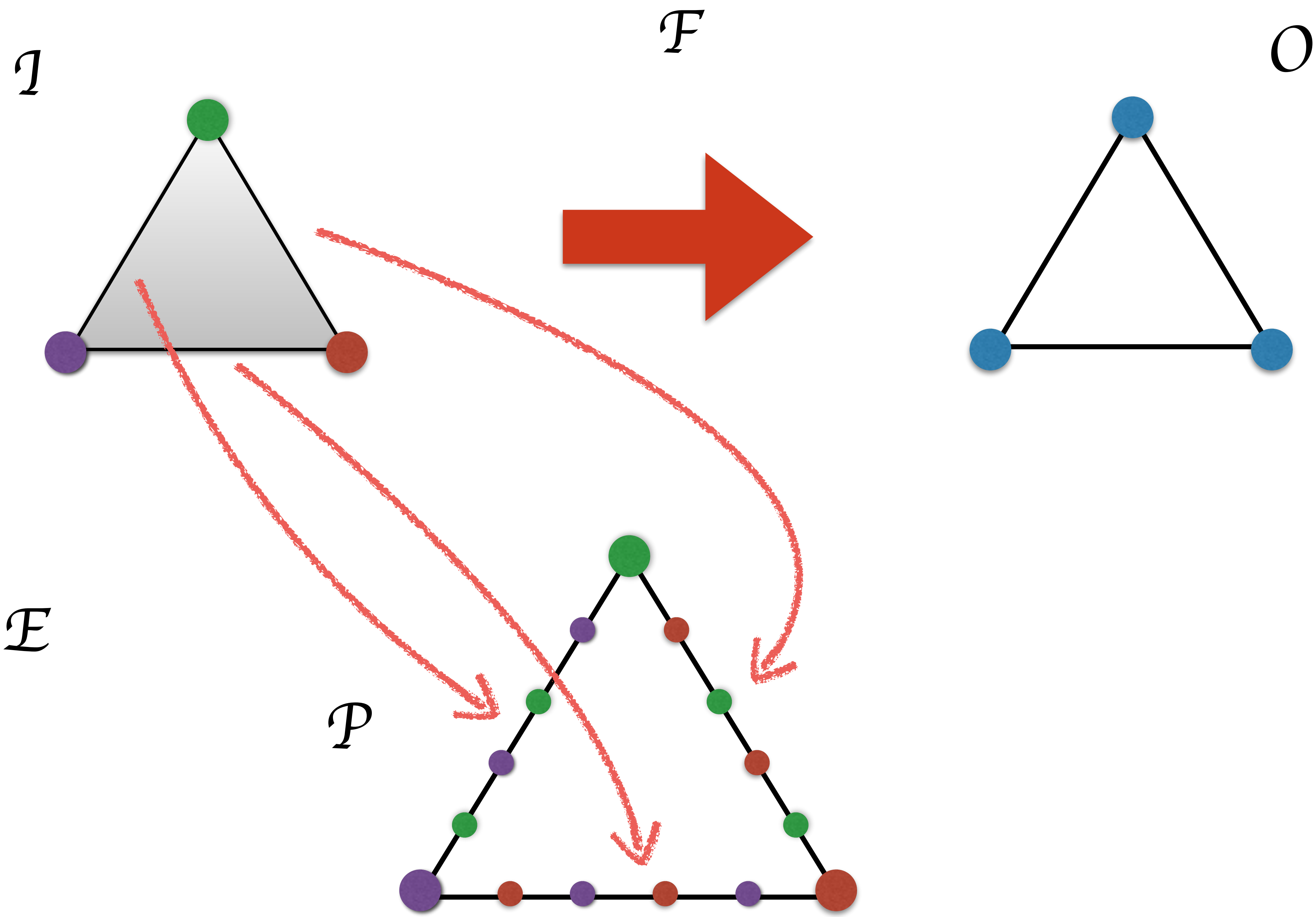
The Triangle is Impossible



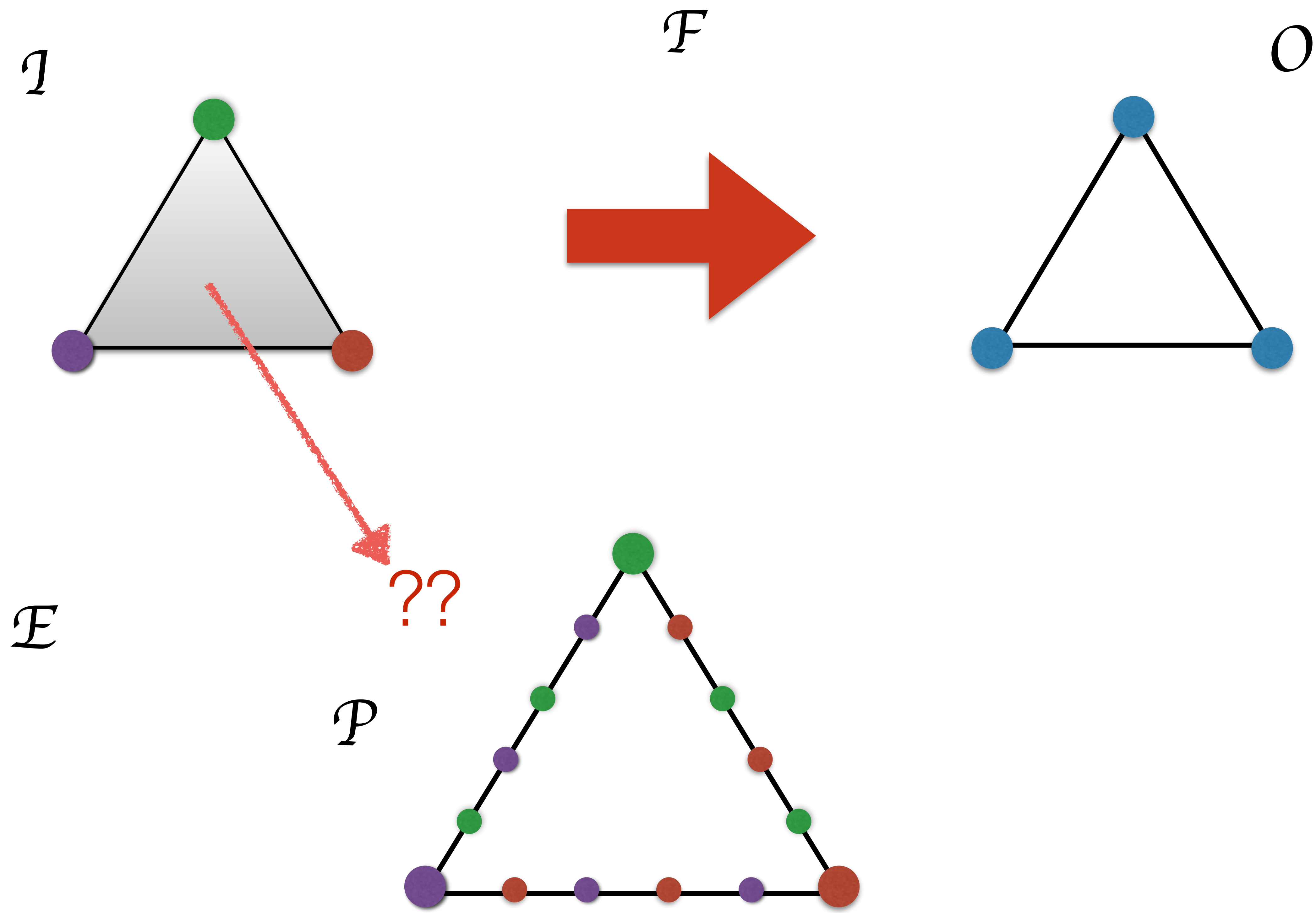
The Triangle is Impossible



The Triangle is Impossible



The Triangle is Impossible



Immediate Snapshot Executions (ISE)

- Subset of nice structured executions.
- Robots proceed in a sequence of concurrency classes:

$$\{A,B,C\}\ \{B\}\ \{A,C\}\ \{B\}\ \{B\}\ \{A,C\}\ \dots$$

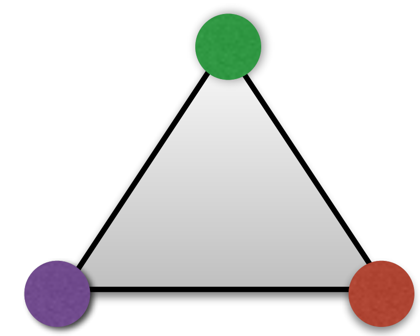
- Concurrency class: concurrent move, then concurrent look.

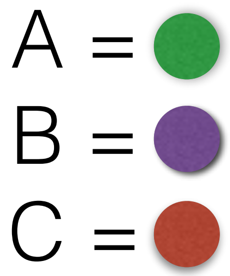
A =

B =

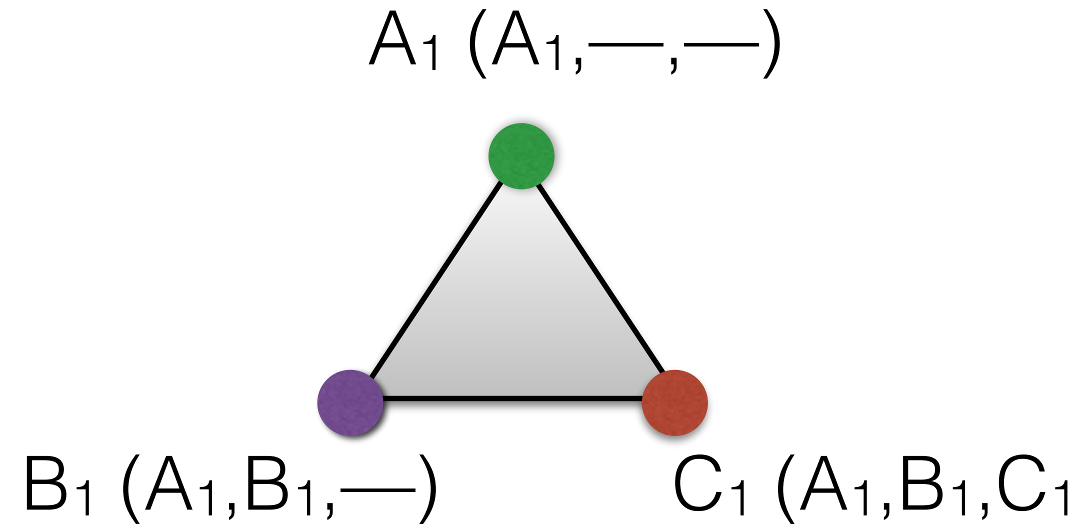
C =

{A}{B}{C}

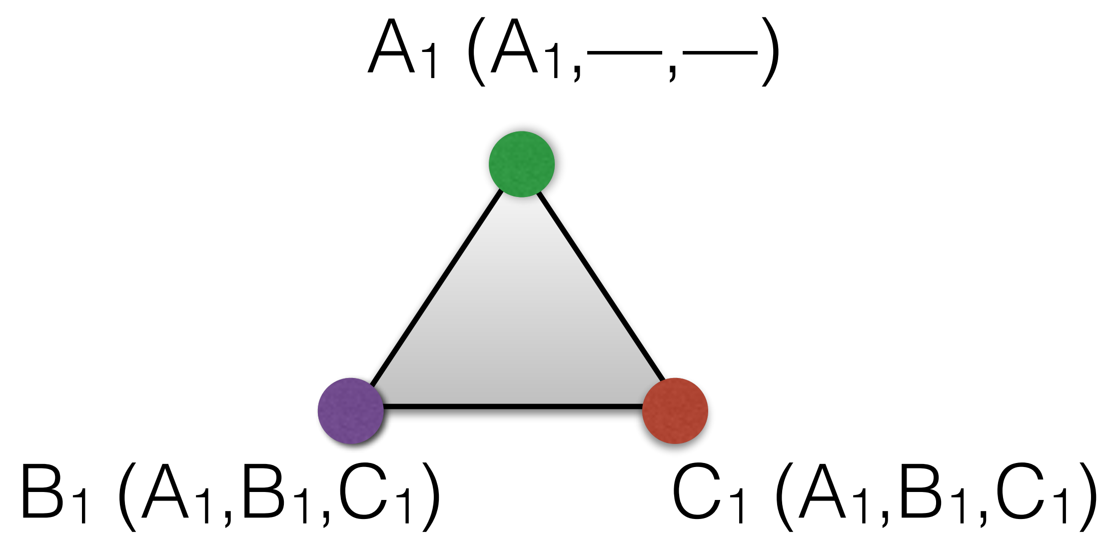


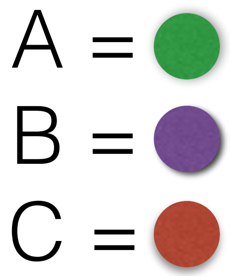


{A}{B}{C}

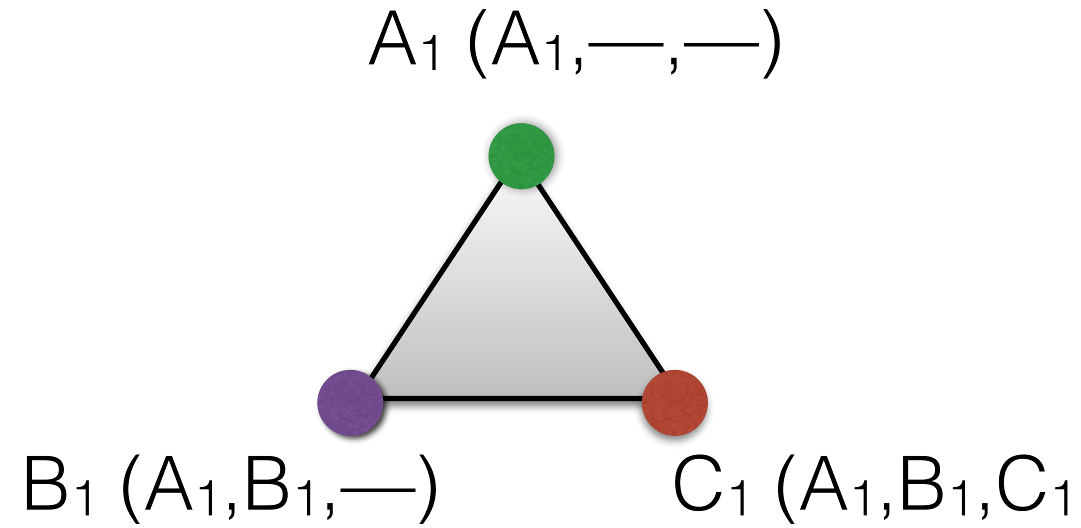


{A}{B,C}

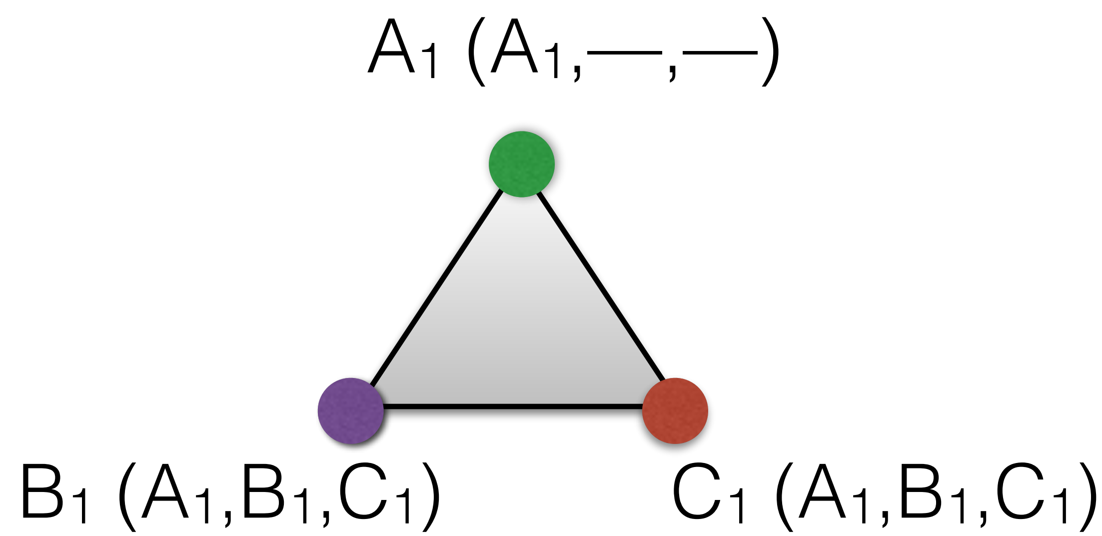



A =
B =
C =

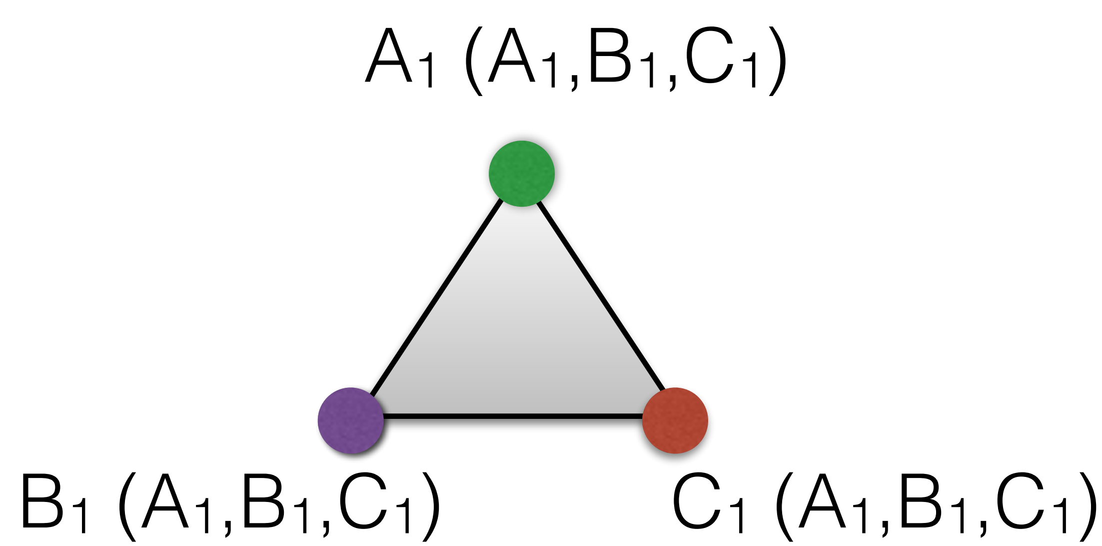
{A}{B}{C}

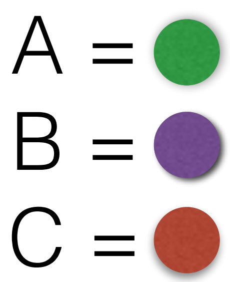

A_1 (A_1,—,—)
B_1 (A_1,B_1,—)
C_1 (A_1,B_1,C_1)

{A}{B,C}

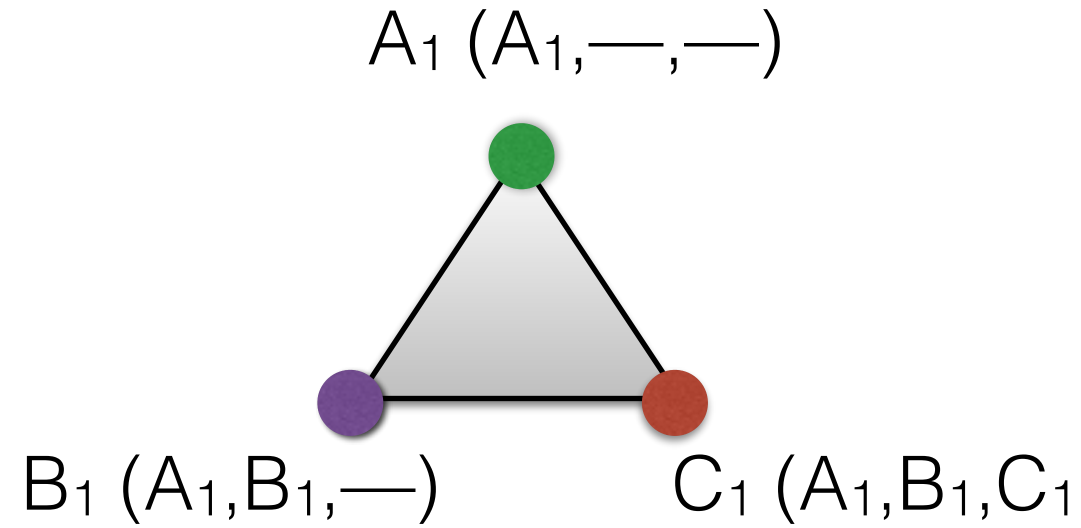

A_1 (A_1,—,—)
B_1 (A_1,B_1,C_1)
C_1 (A_1,B_1,C_1)

{A,B,C}


A_1 (A_1,B_1,C_1)
B_1 (A_1,B_1,C_1)
C_1 (A_1,B_1,C_1)

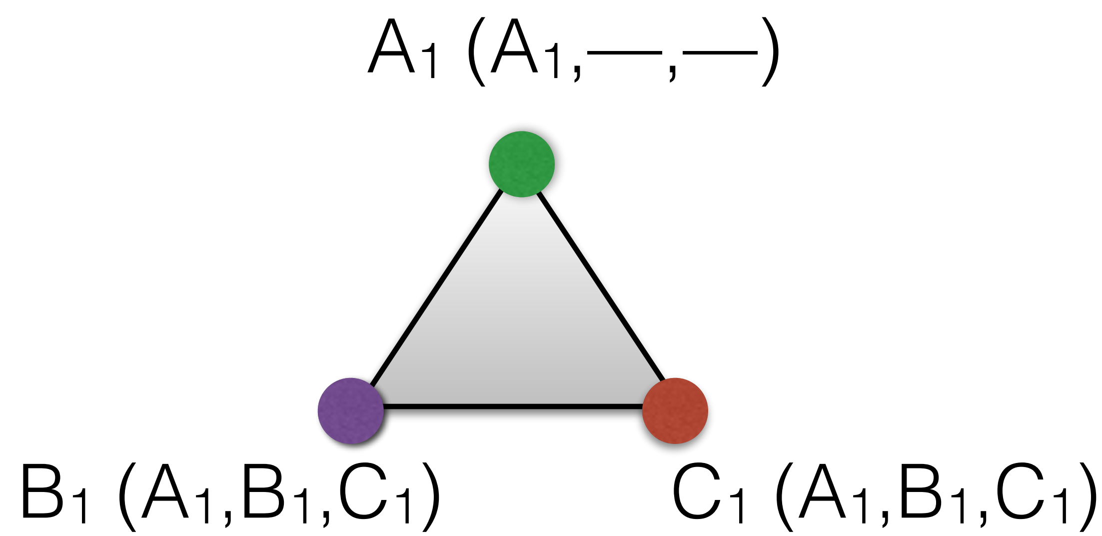


$\{A\}\{B\}\{C\}$

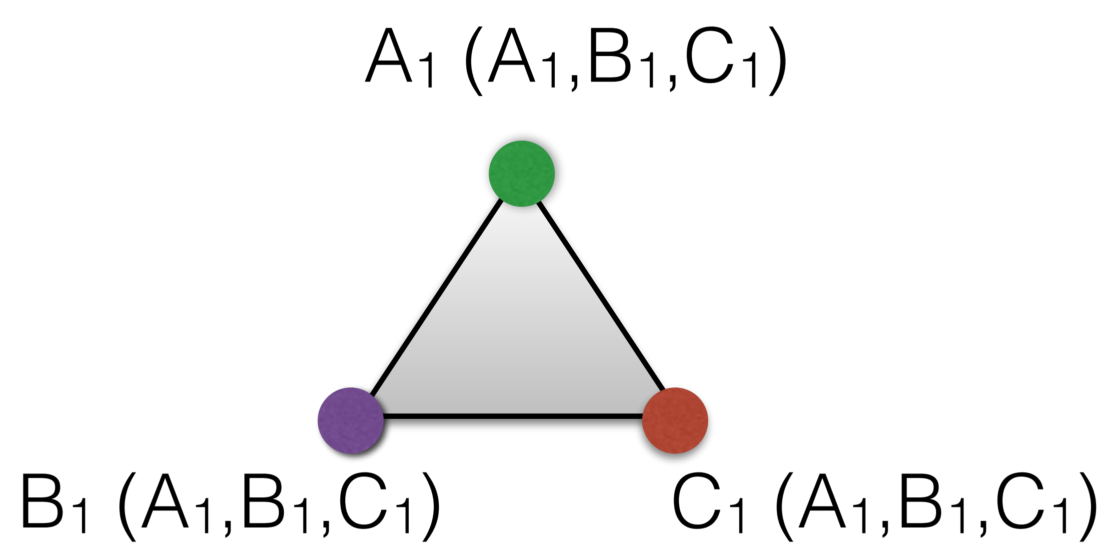


And so on …

$\{A\}\{B,C\}$

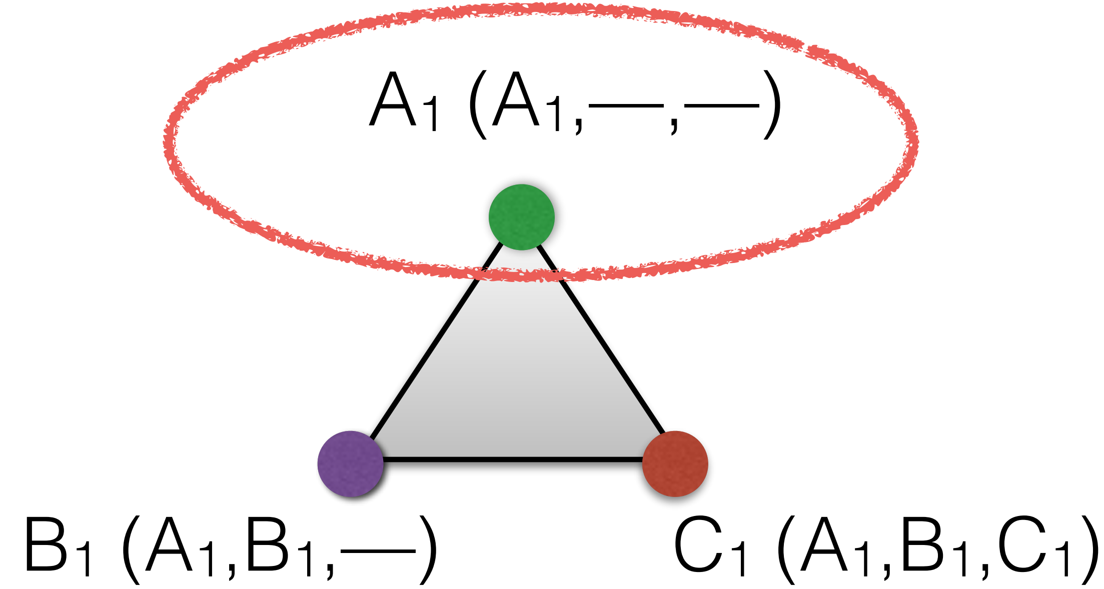


$\{A,B,C\}$



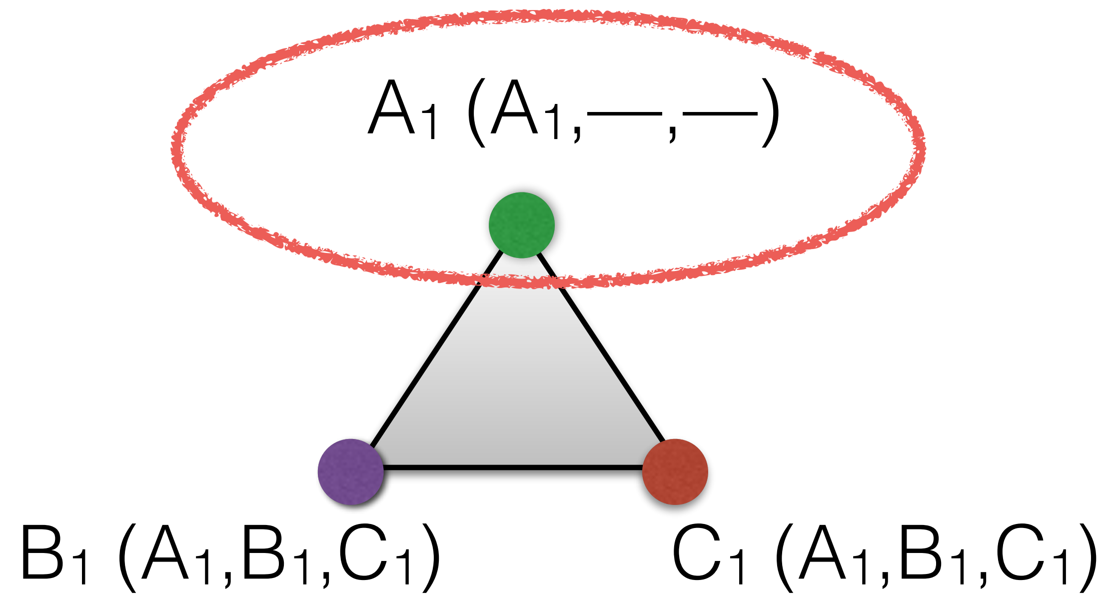
A =
B =
C =

{A}{B}{C}

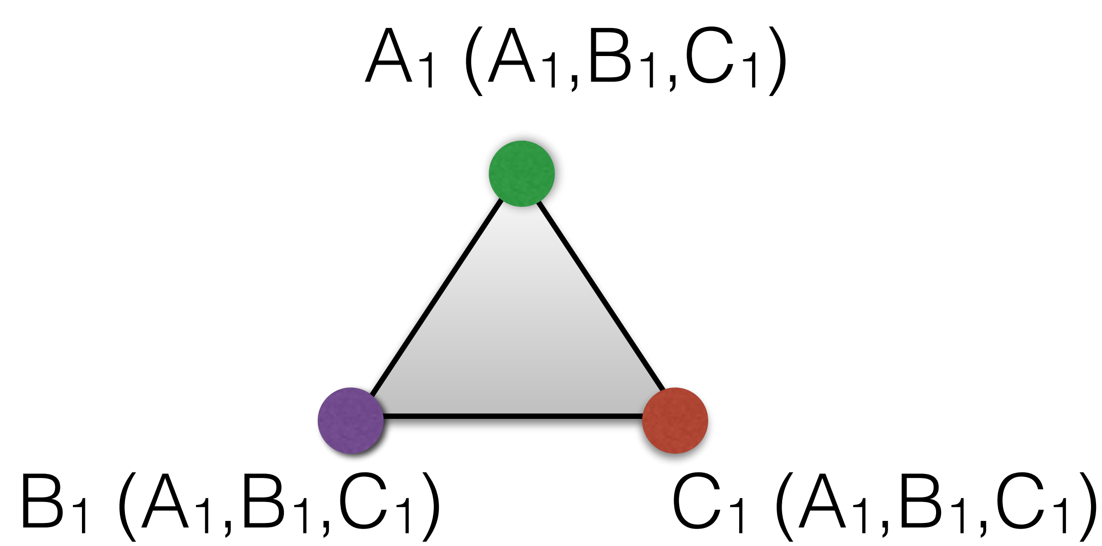


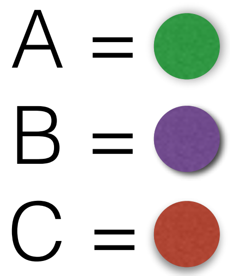
And so on …

{A}{B,C}

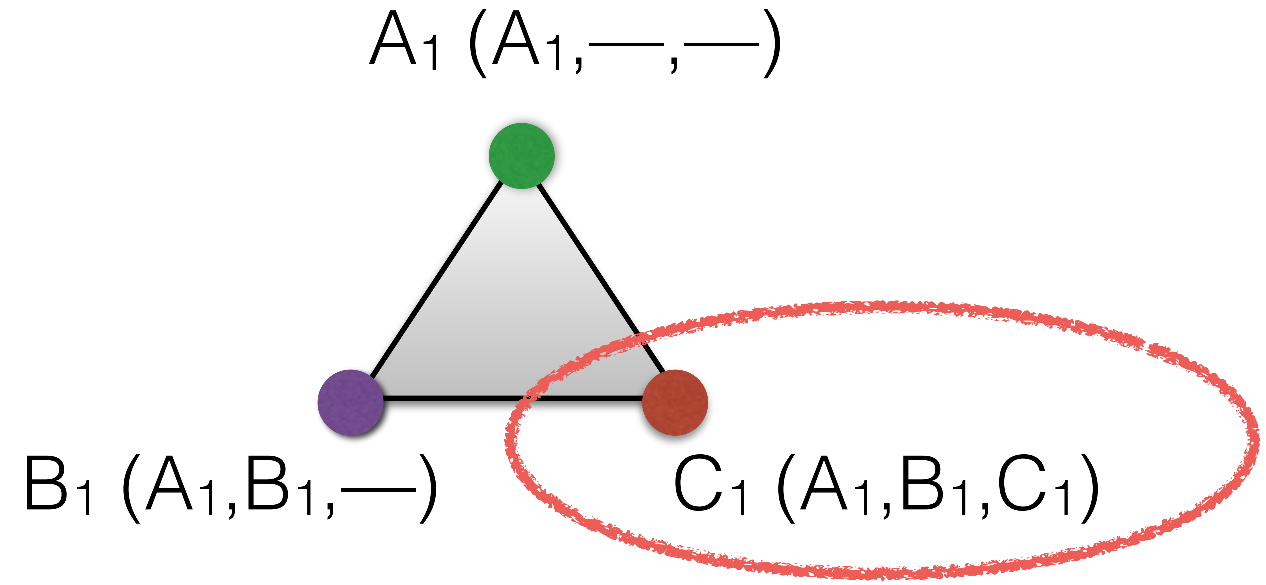


{A,B,C}



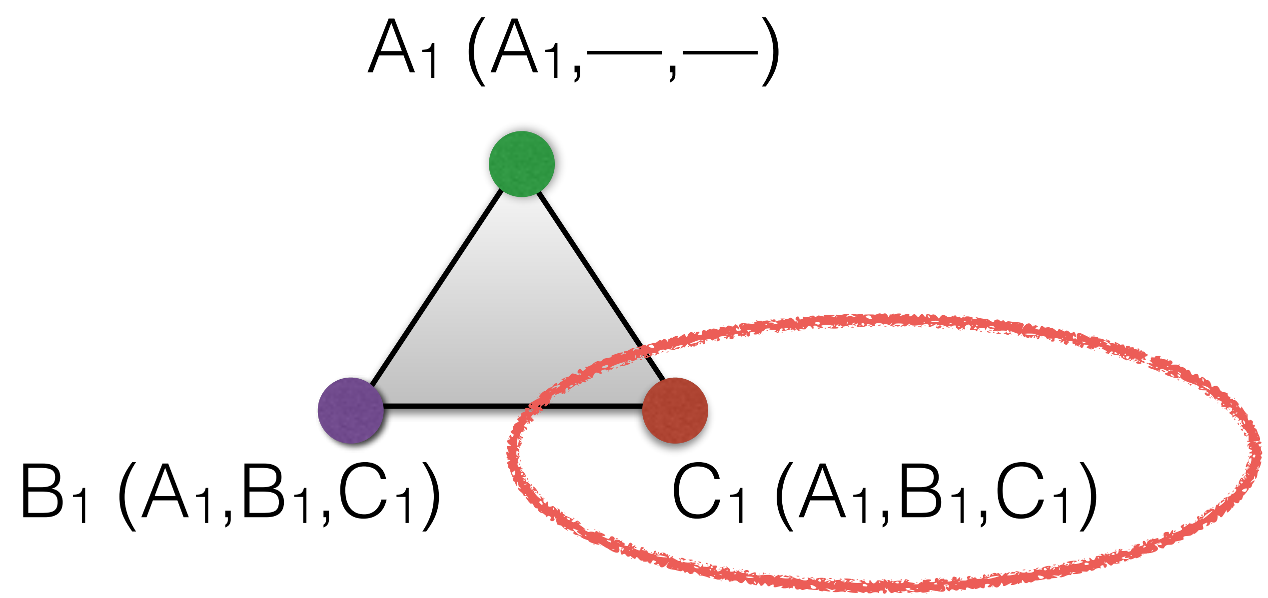


{A}{B}{C}

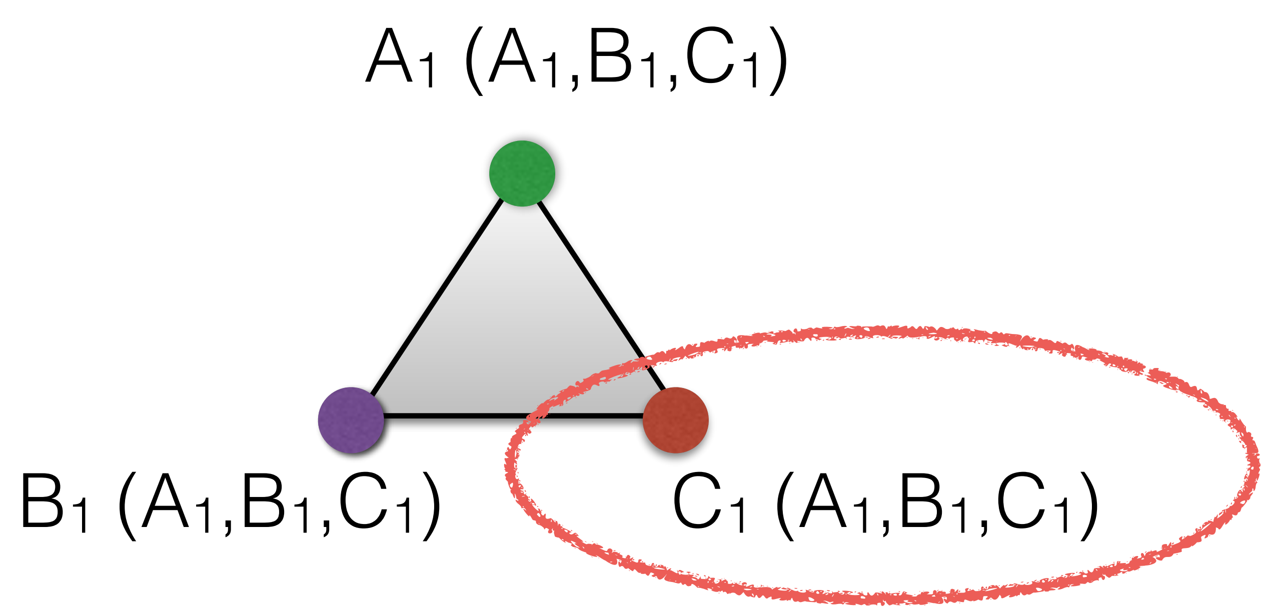


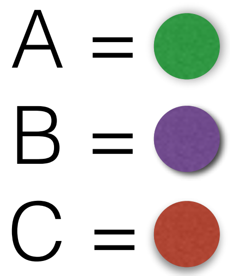
And so on …

{A}{B,C}

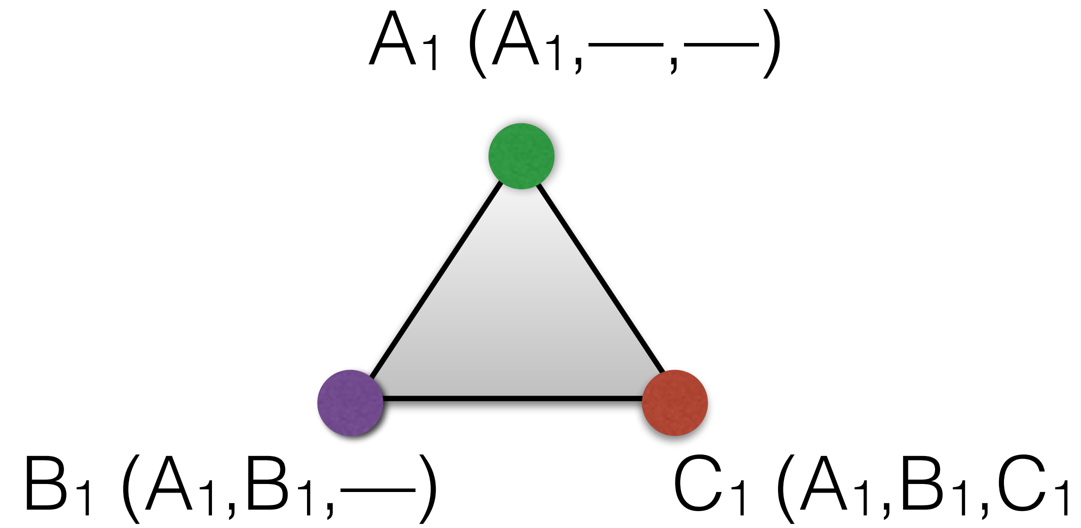


{A,B,C}



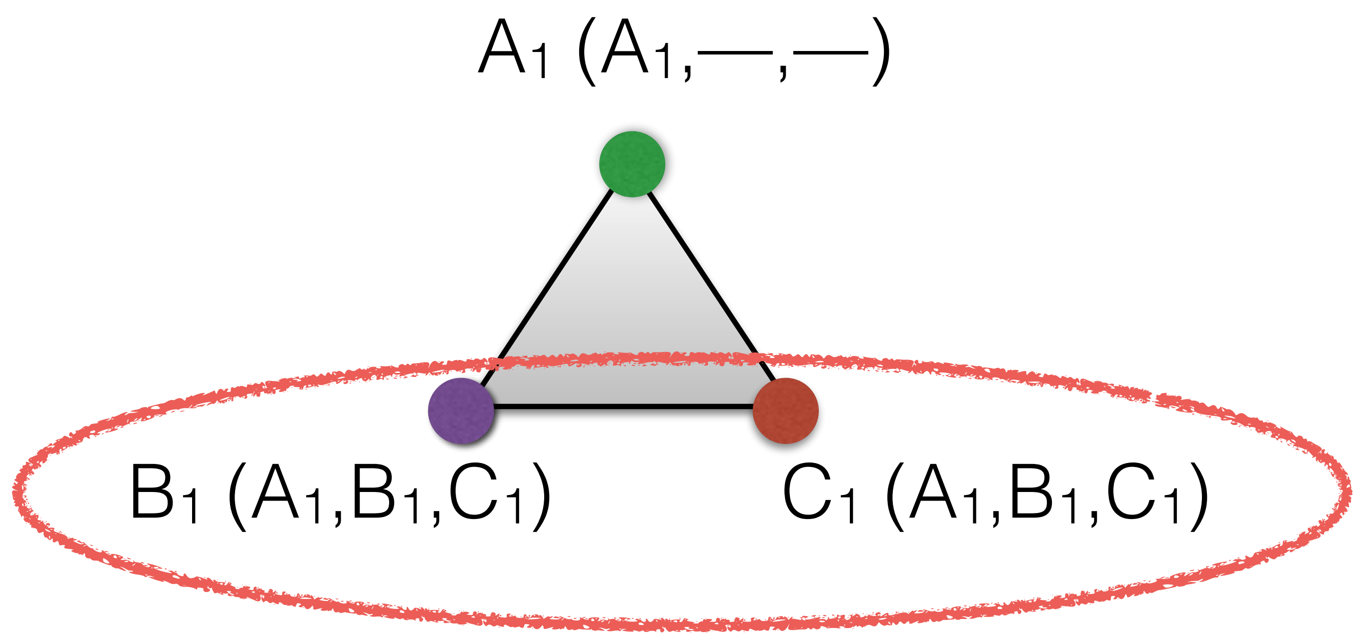


{A}{B}{C}

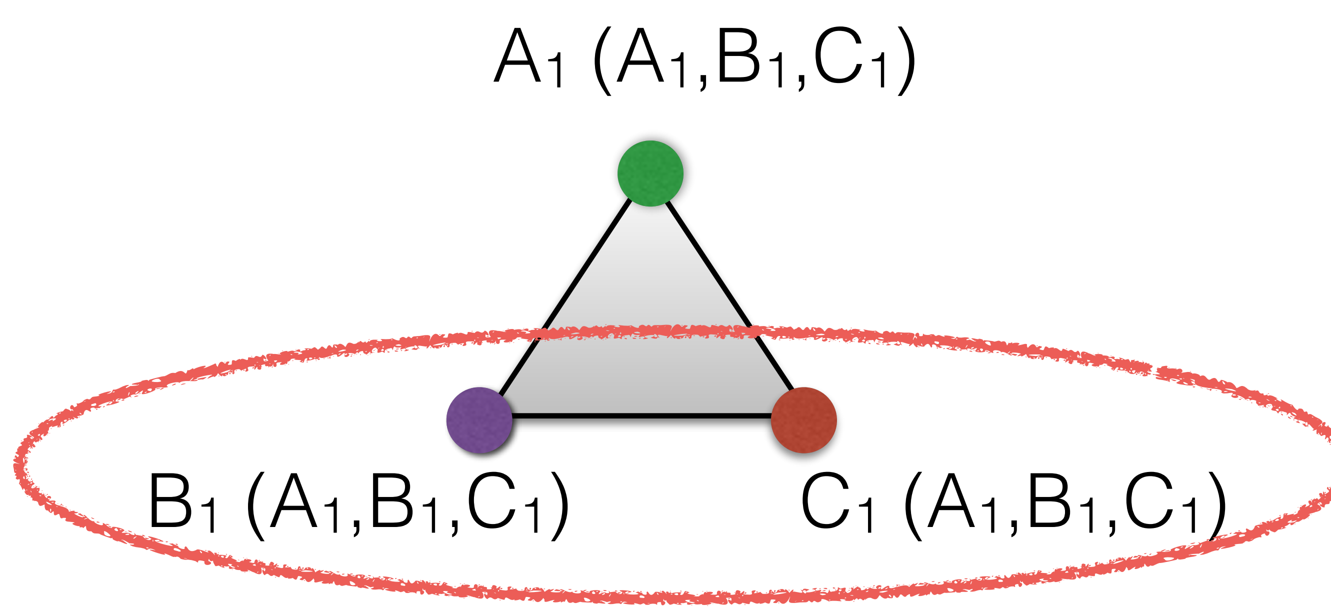


And so on …

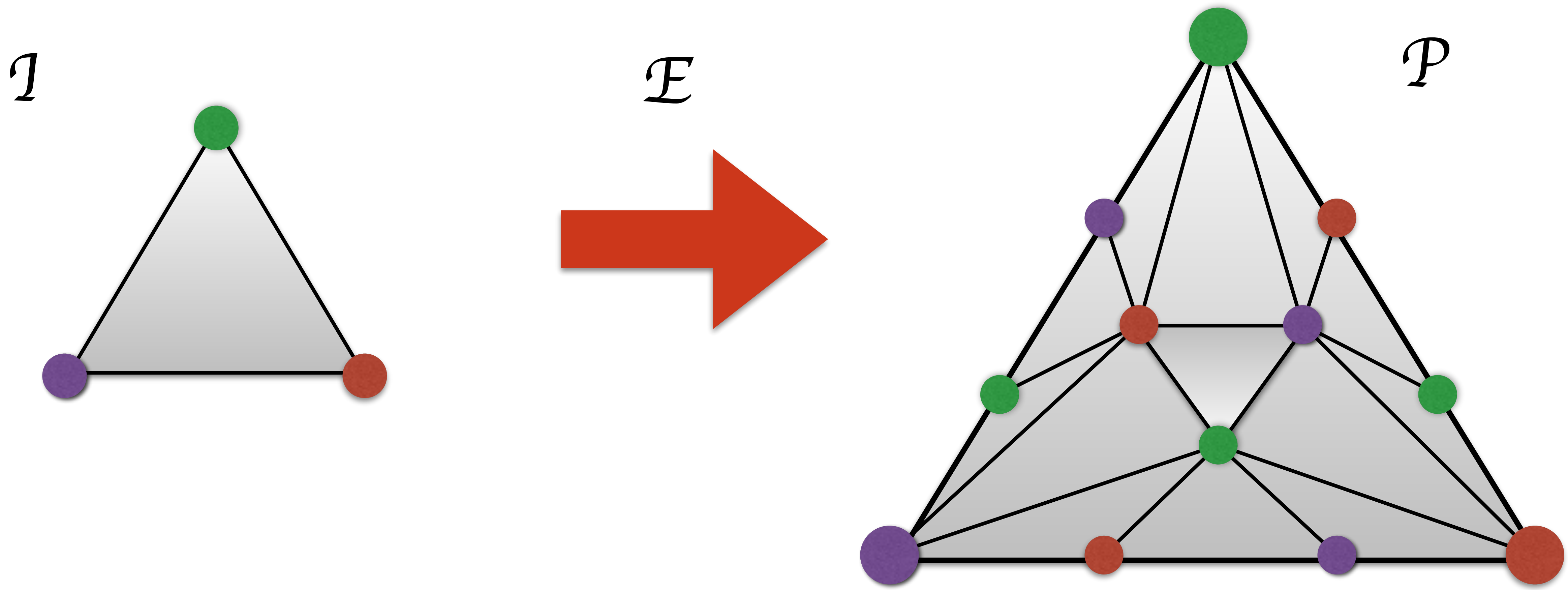
{A}{B,C}



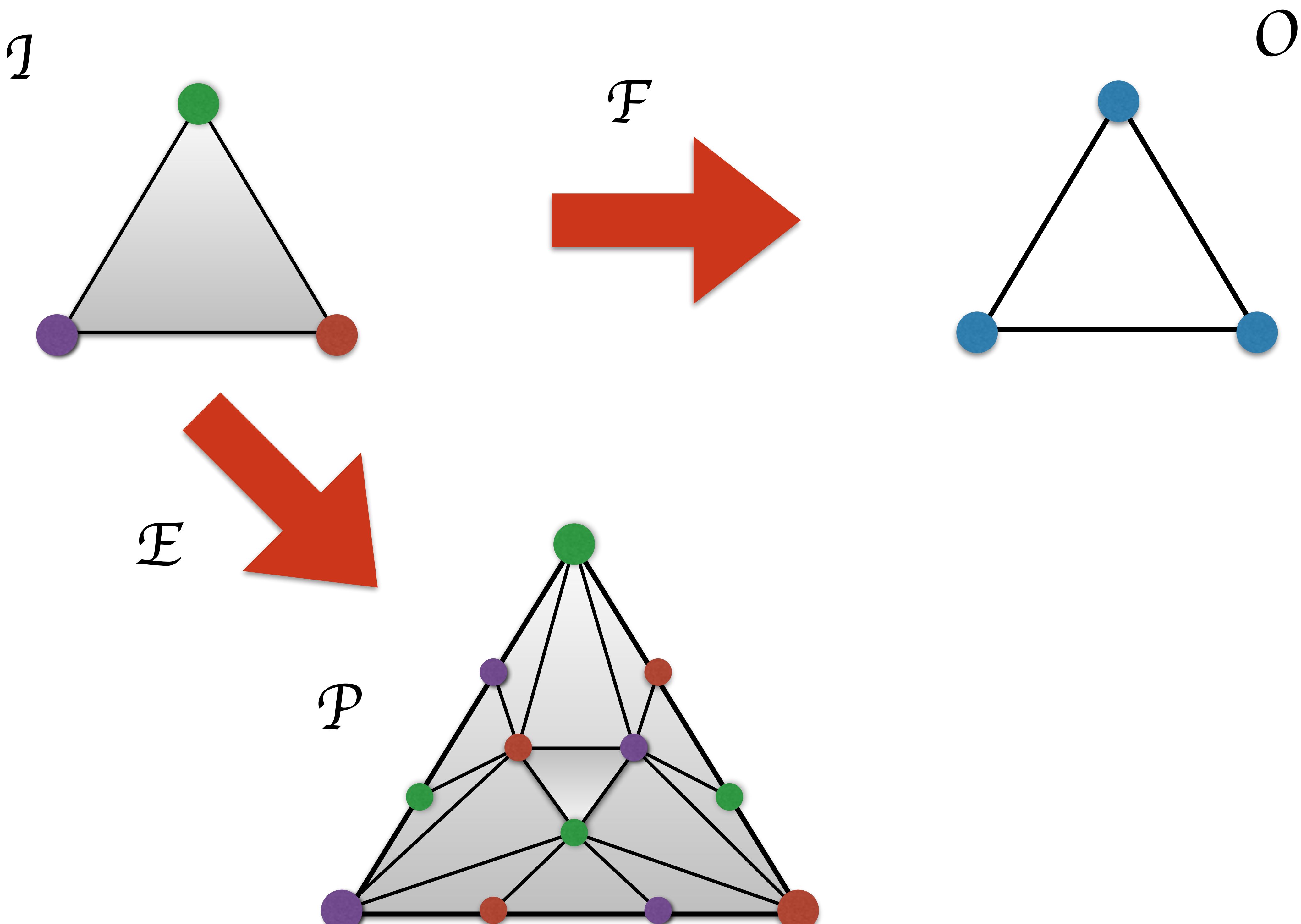
{A,B,C}



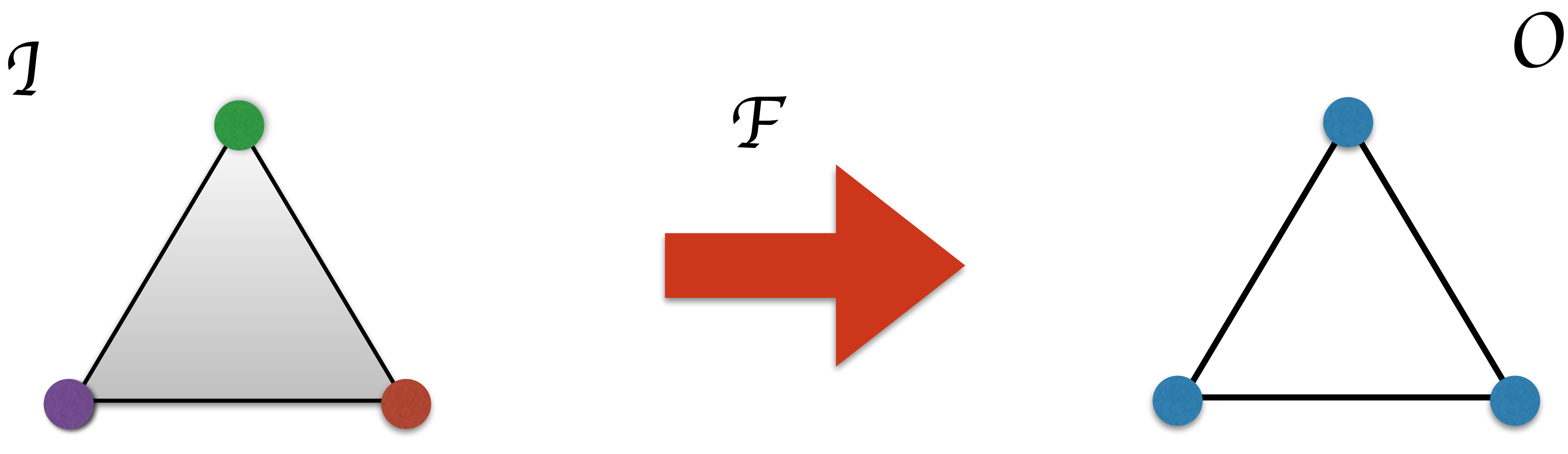
1-Round ISE Complex

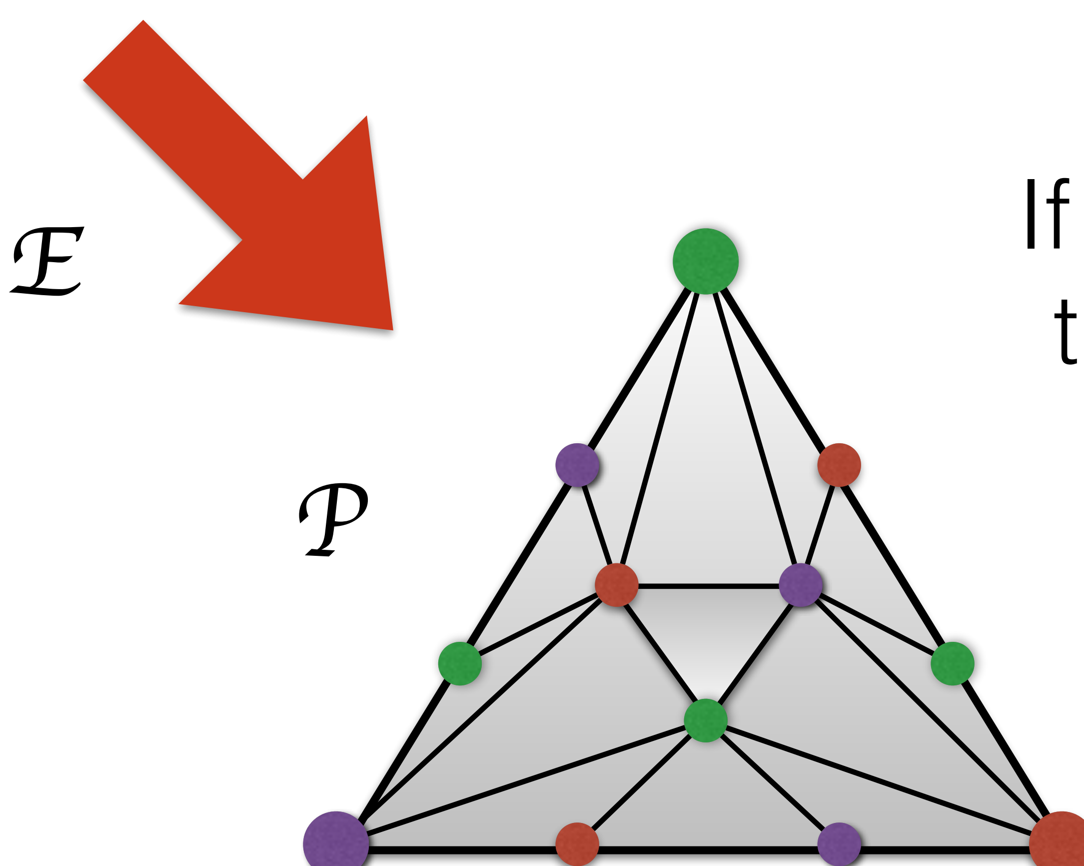


The Triangle is Impossible



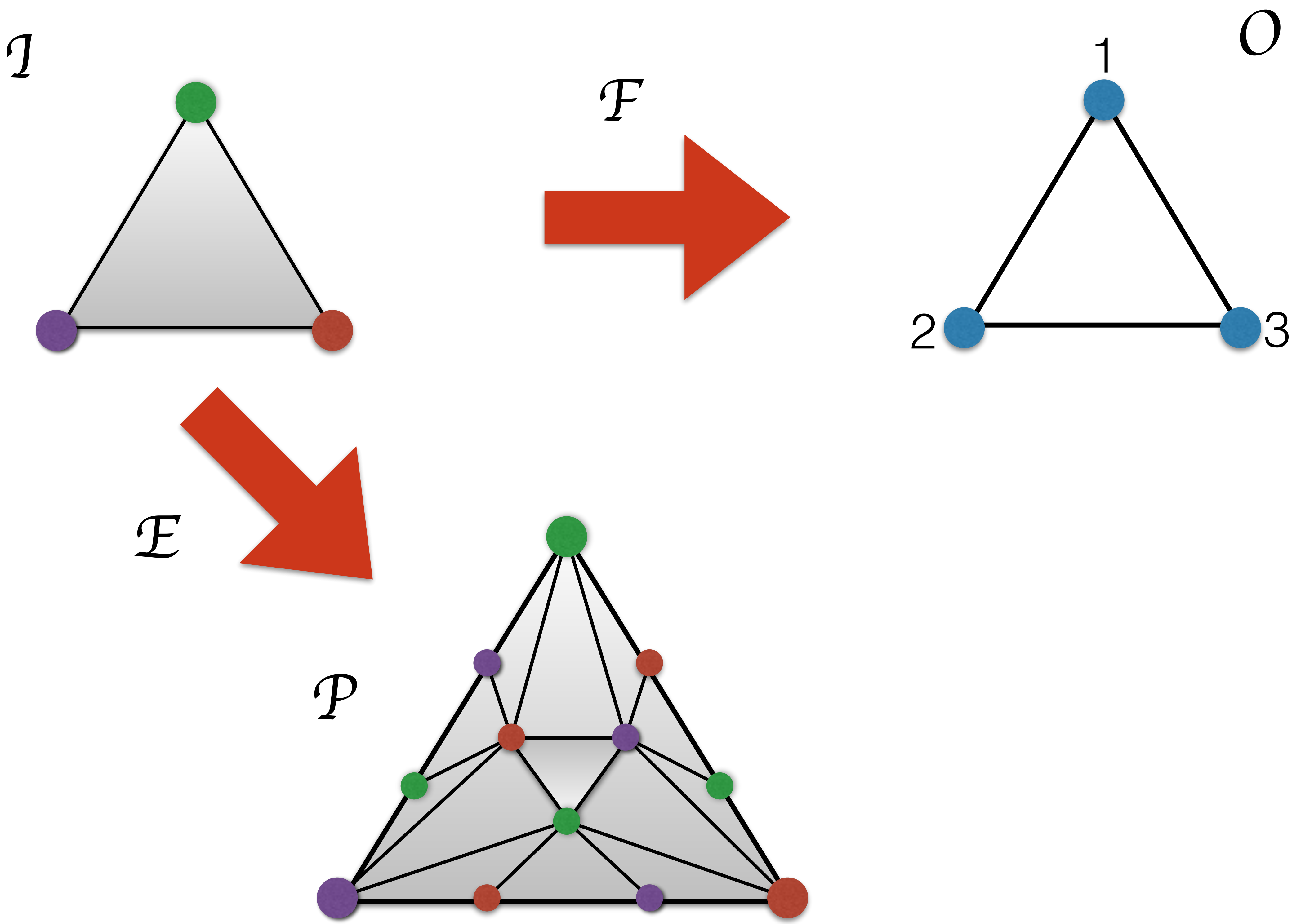
The Triangle is Impossible



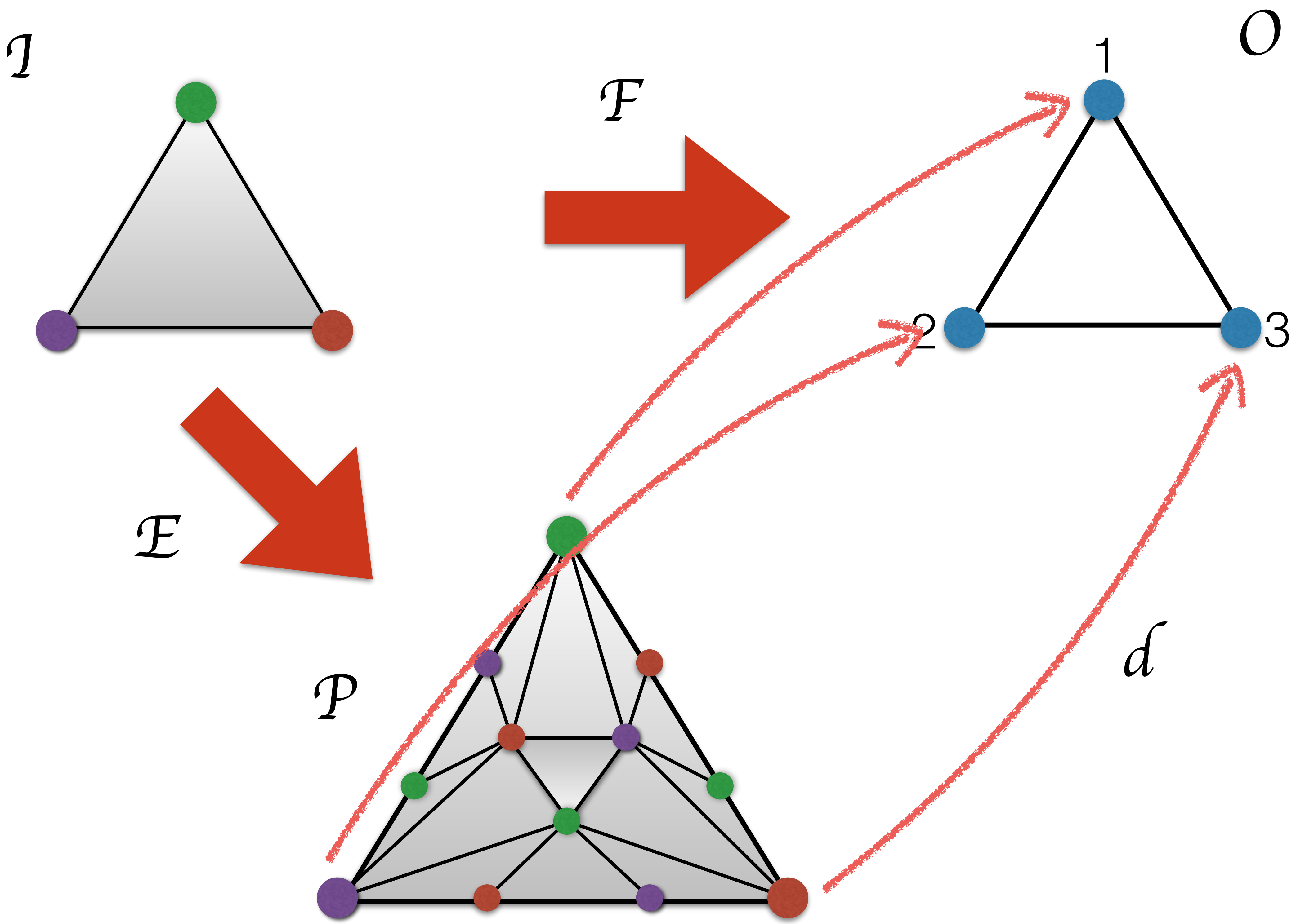


If there is an algorithm =>
there is a simplicial map
d: $\mathcal{V}(\mathcal{P}) \longrightarrow \mathcal{V}(\mathcal{O})$
respecting $\mathcal{F}$

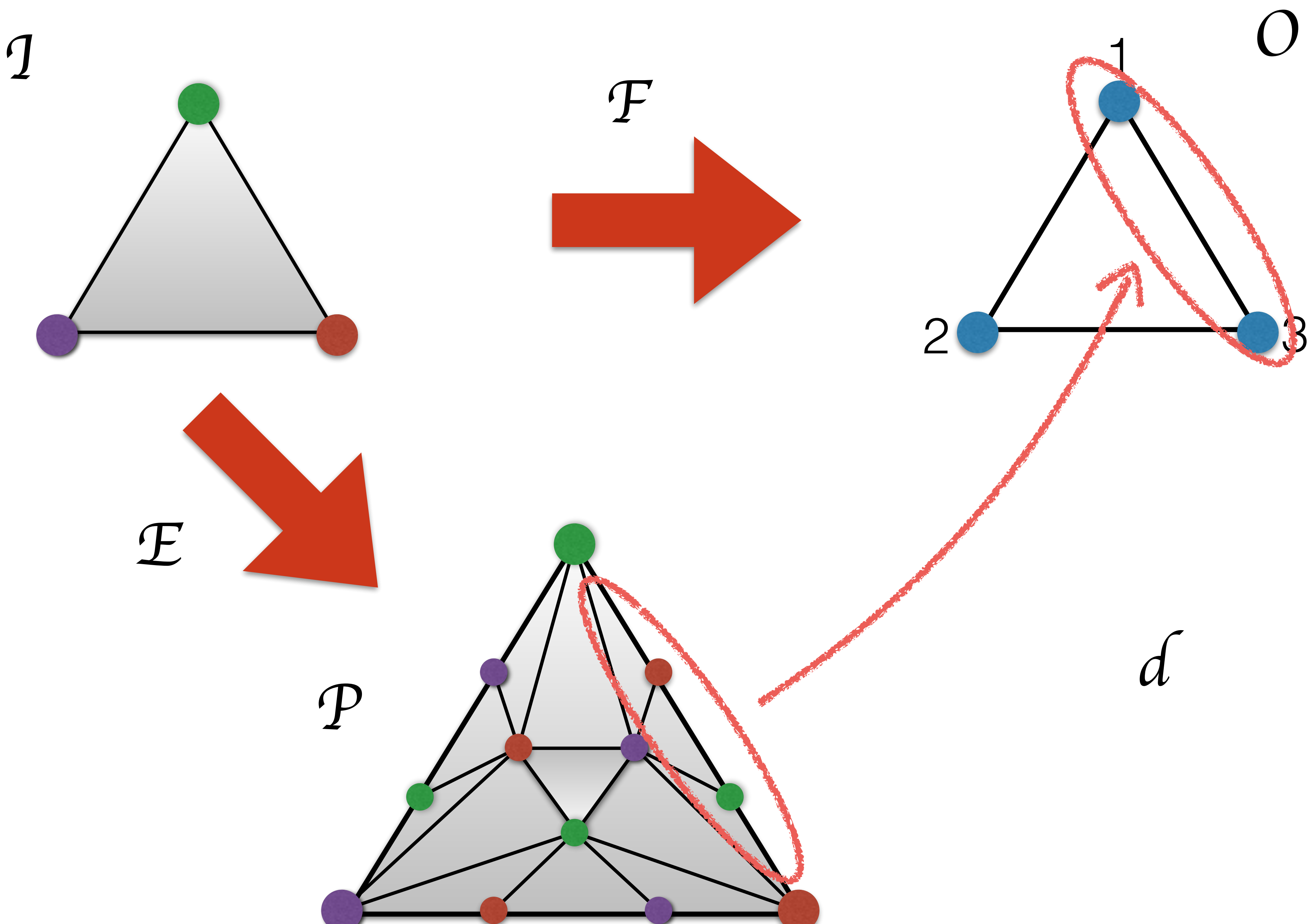
The Triangle is Impossible



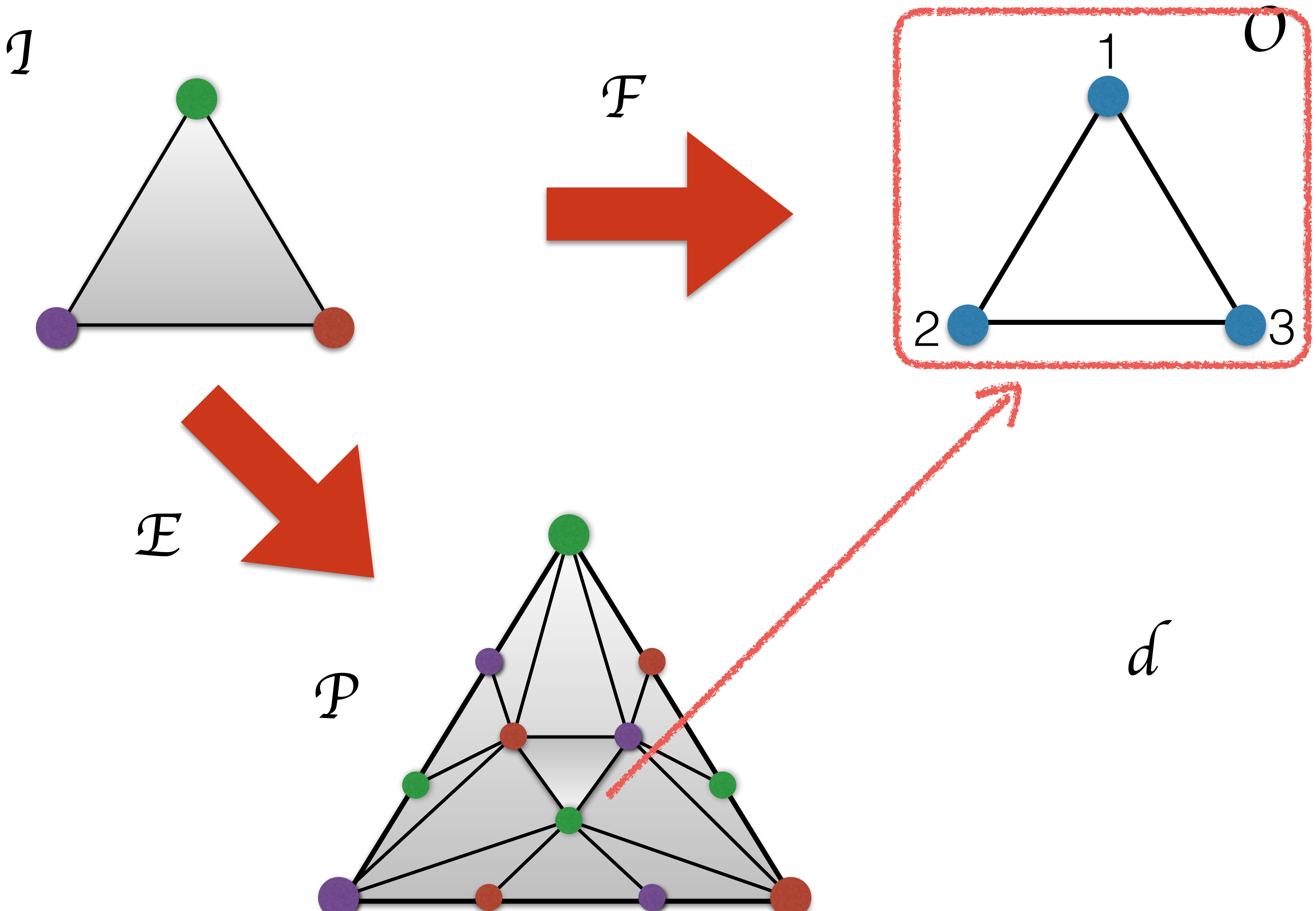
The Triangle is Impossible



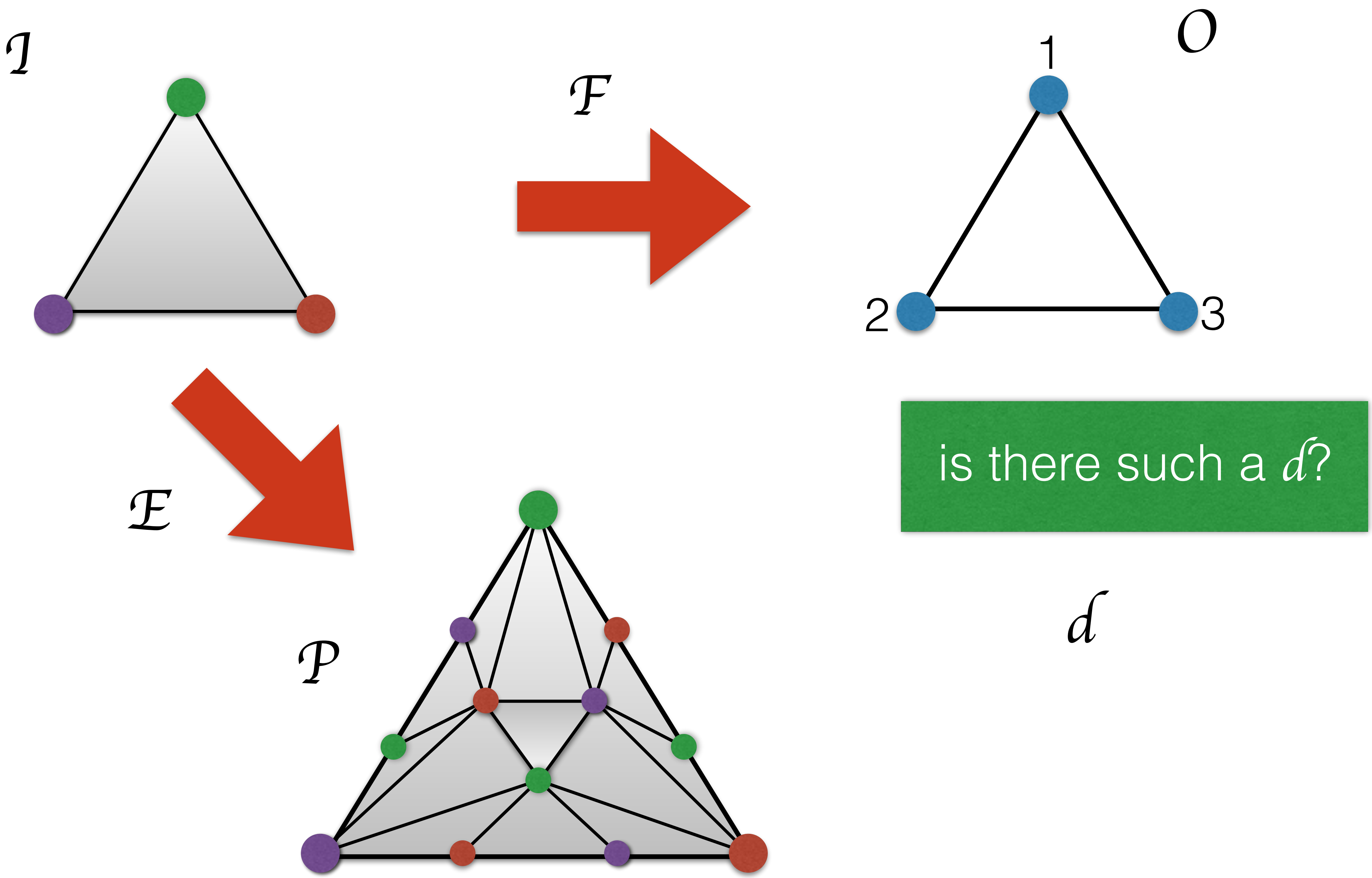
The Triangle is Impossible



The Triangle is Impossible

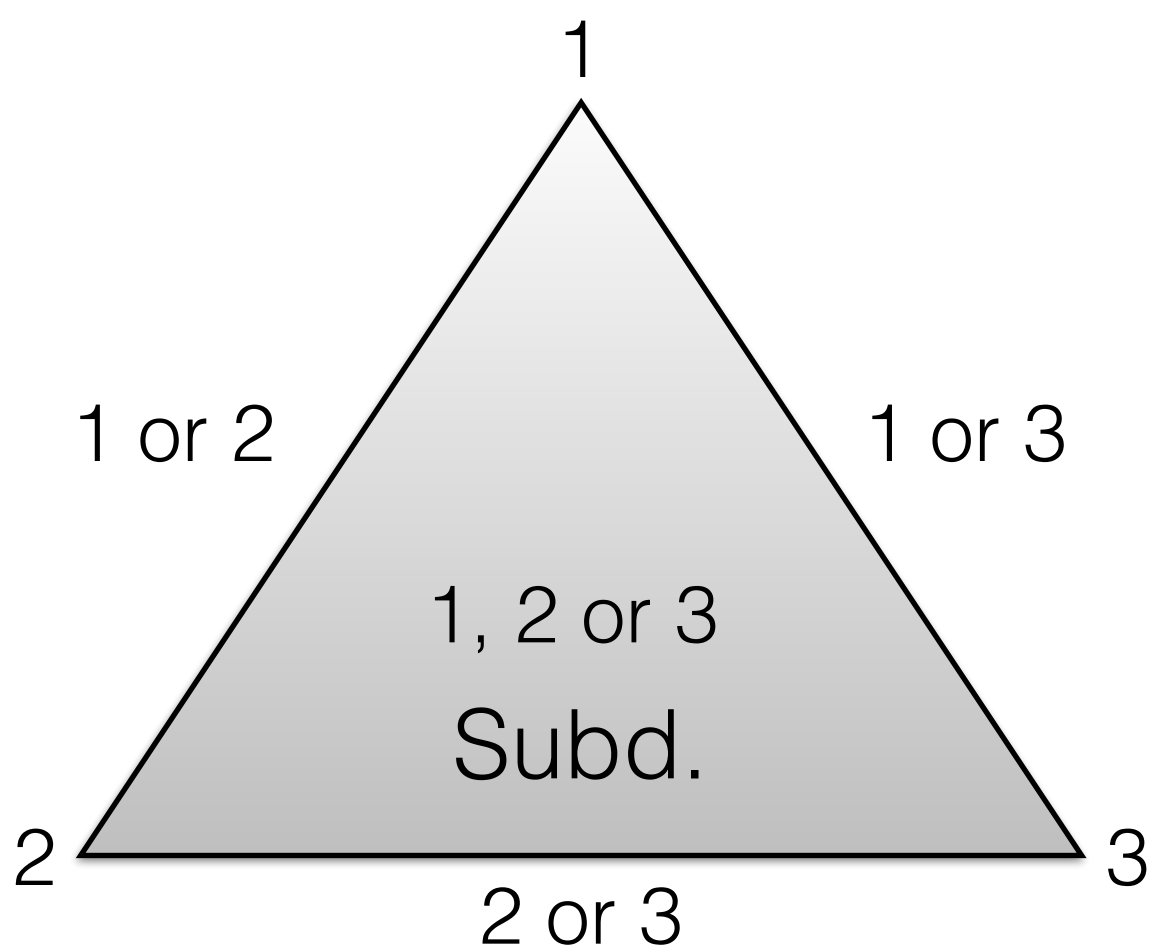


The Triangle is Impossible



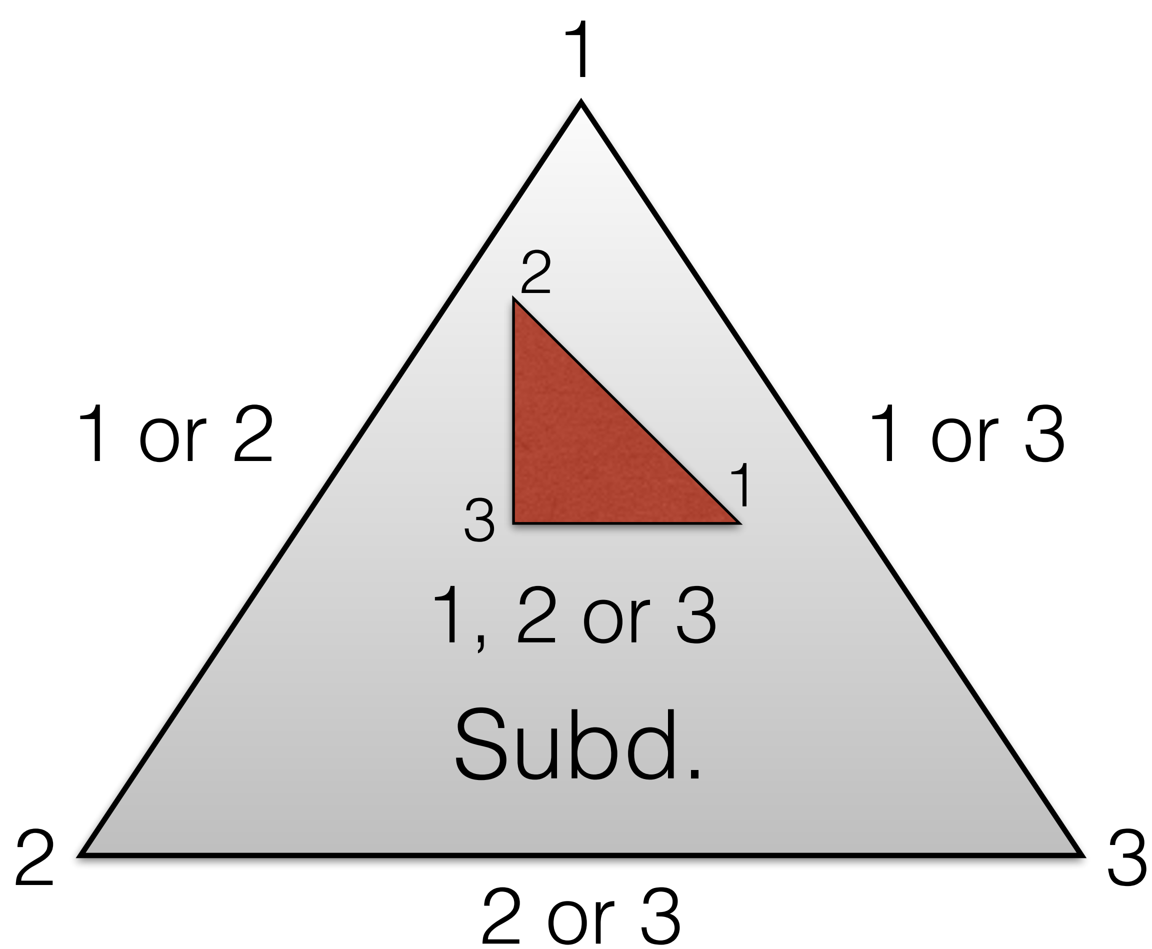
2-Dim Sperner's Lemma

Every subdivision of a triangle with a Sperner coloring has an odd number of 3-chromatic triangles

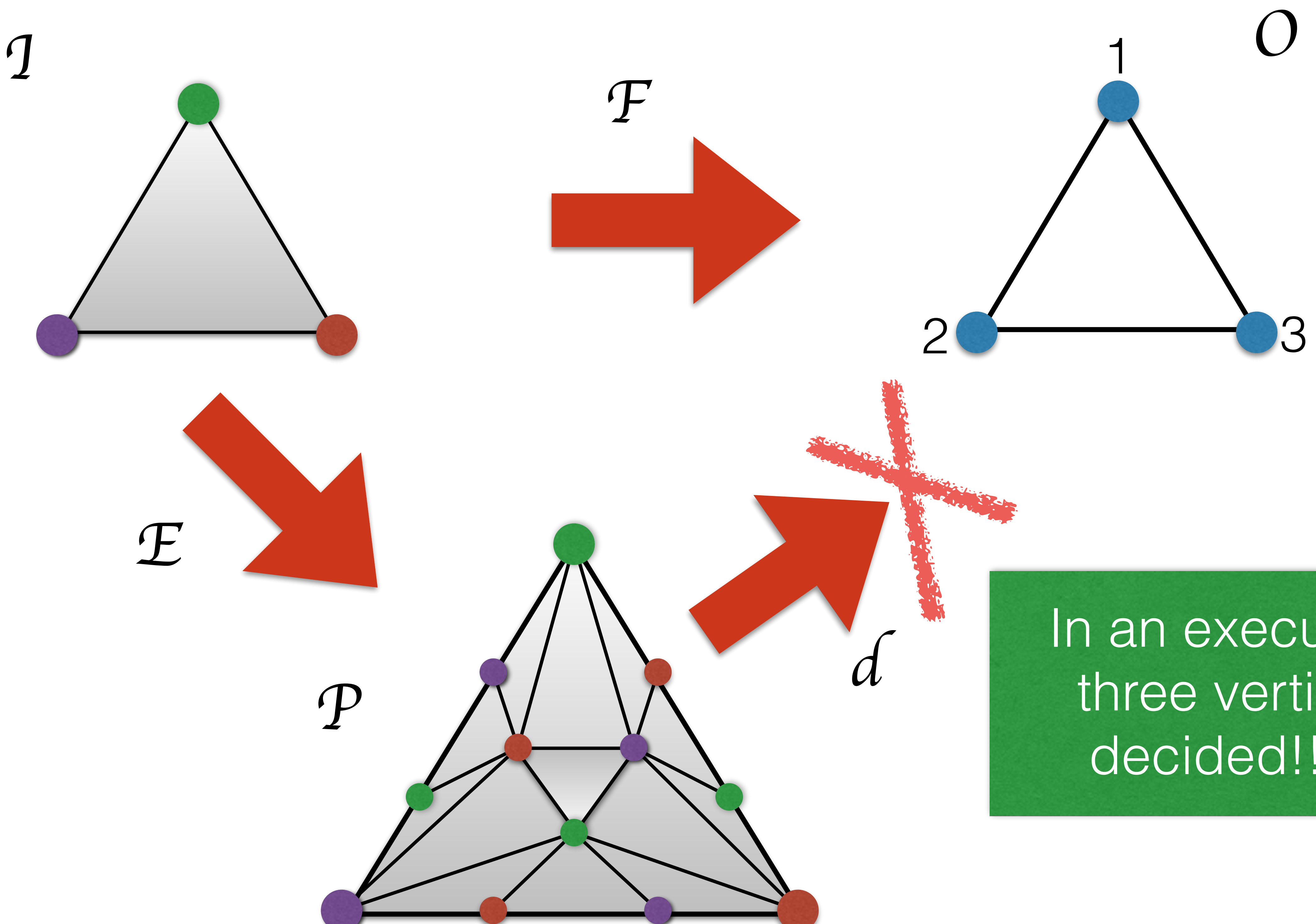


2-Dim Sperner's Lemma

Every subdivision of a triangle with a Sperner coloring has an odd number of 3-chromatic triangles



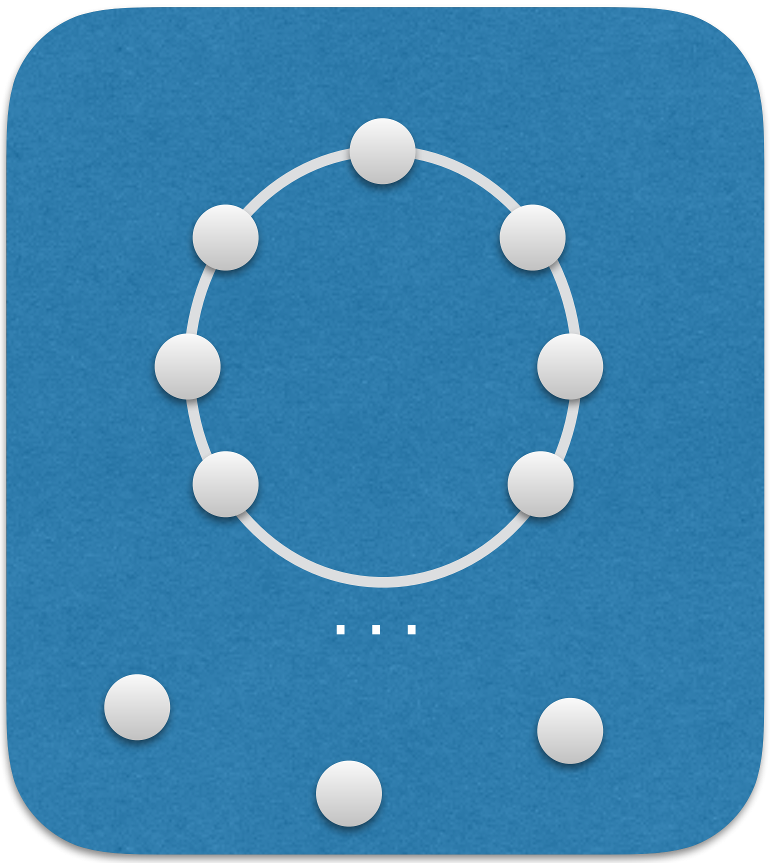
The Triangle is Impossible

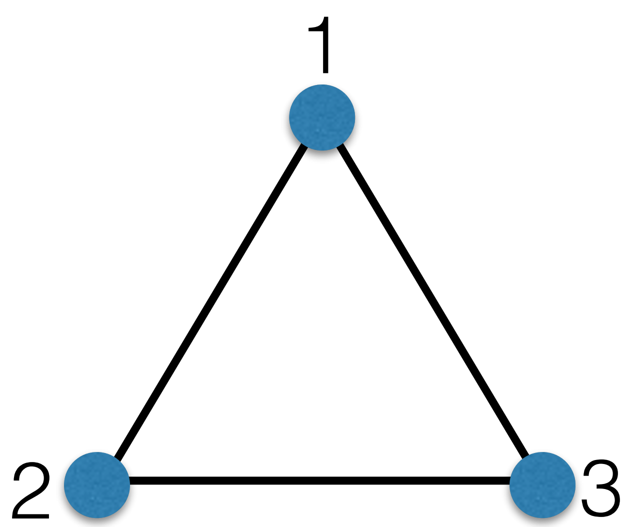


In an execution, three vertices decided!! :-(

Solving Triangle from Cyclic Graphs

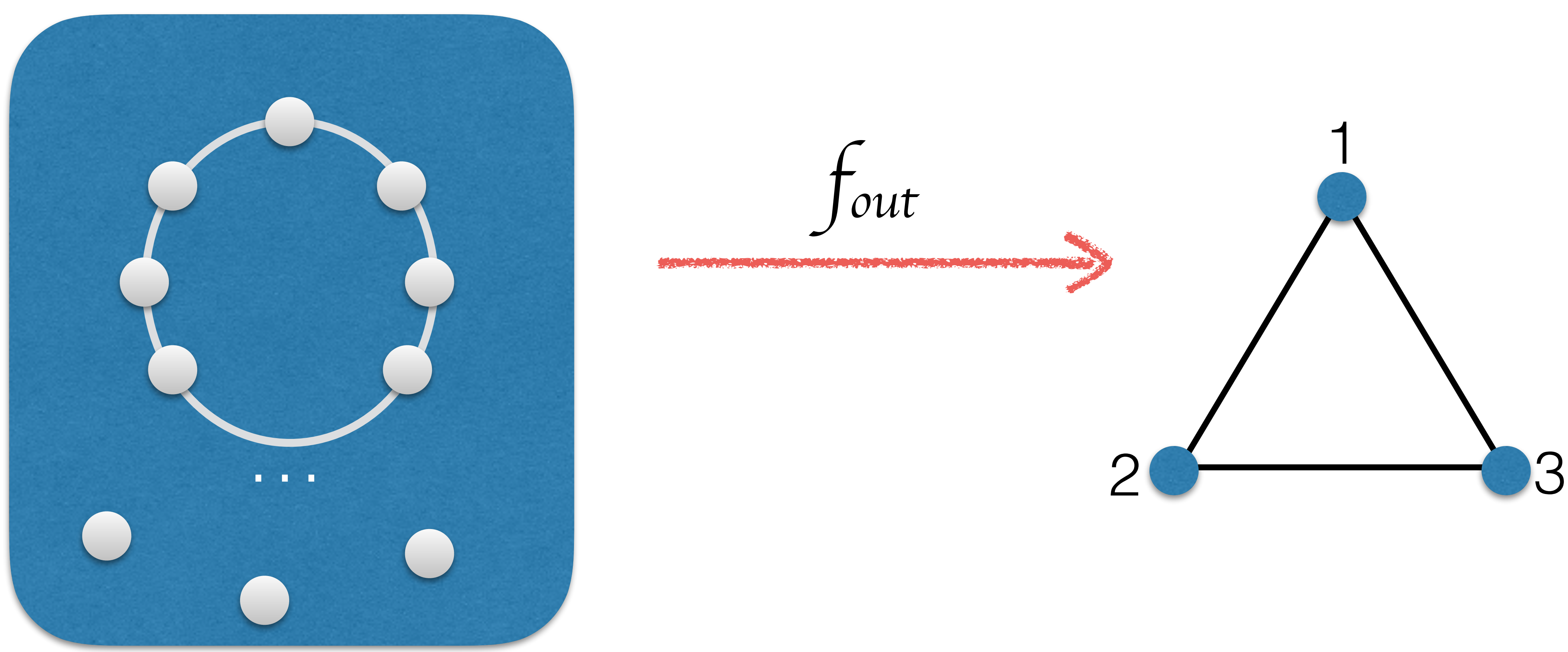
$\mathcal{G}$



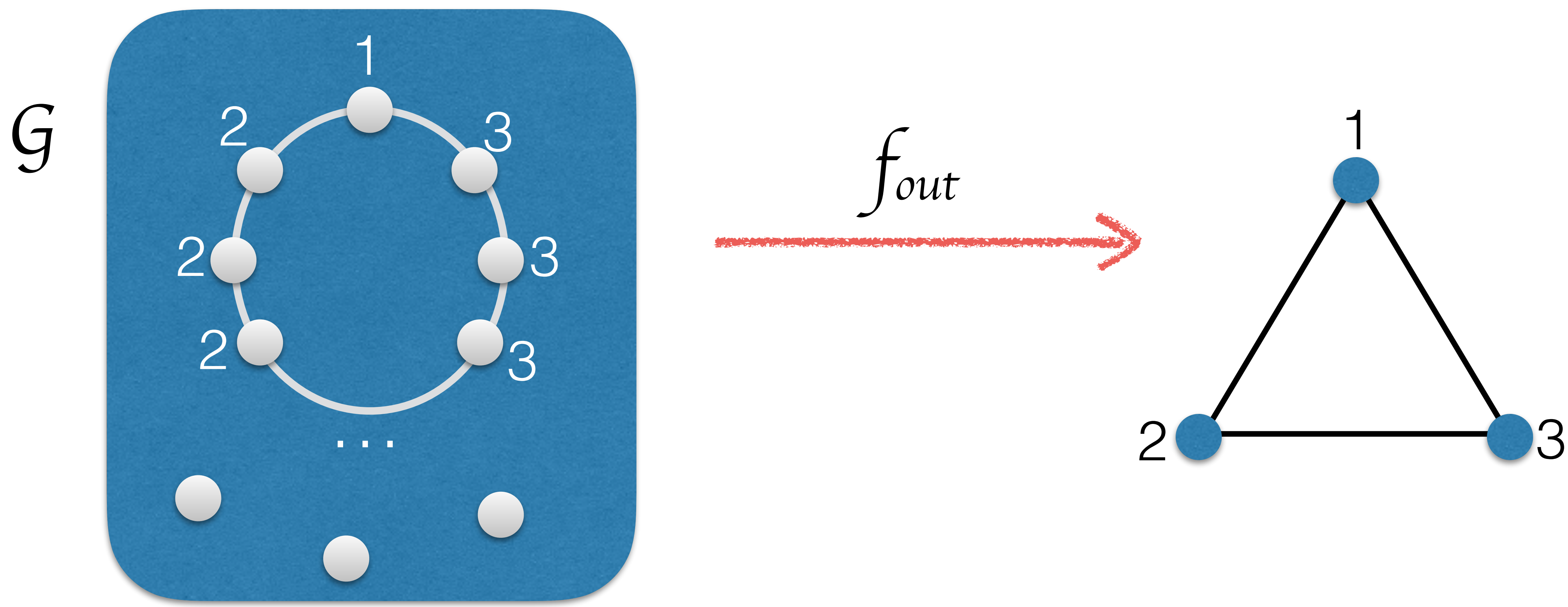


Solving Triangle from Cyclic Graphs

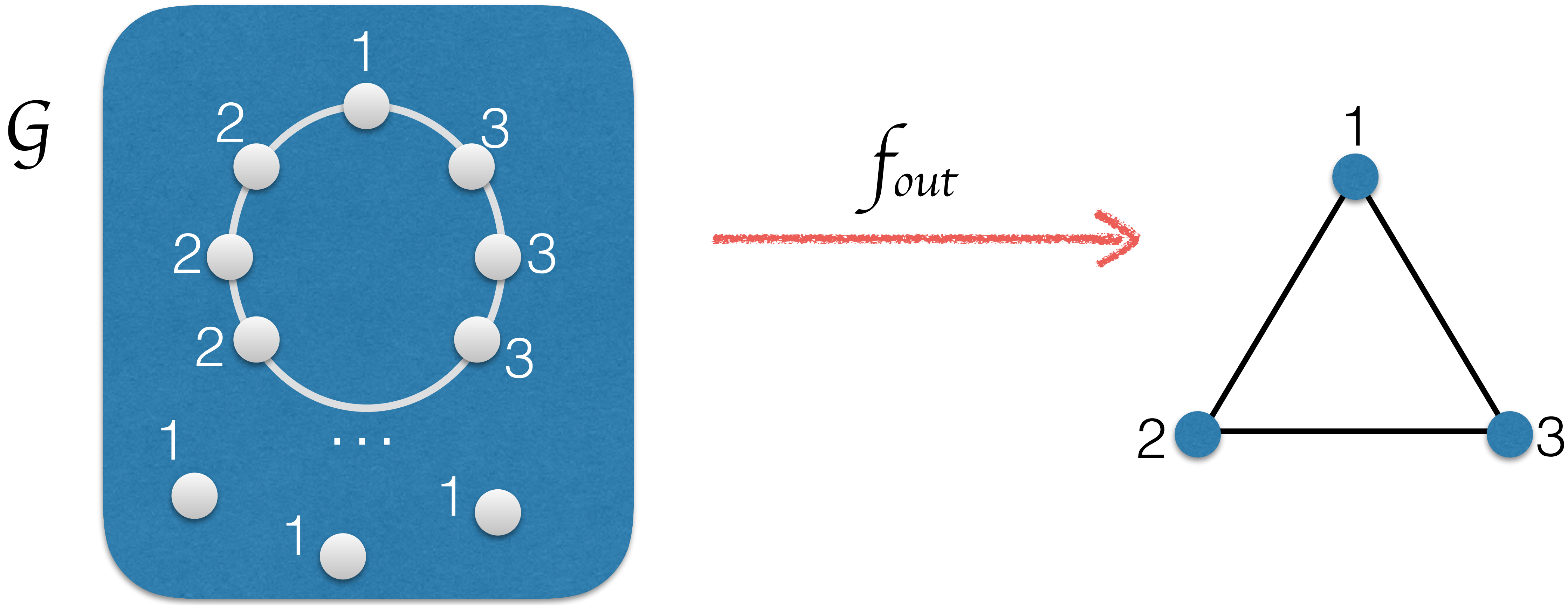
$\mathcal{G}$



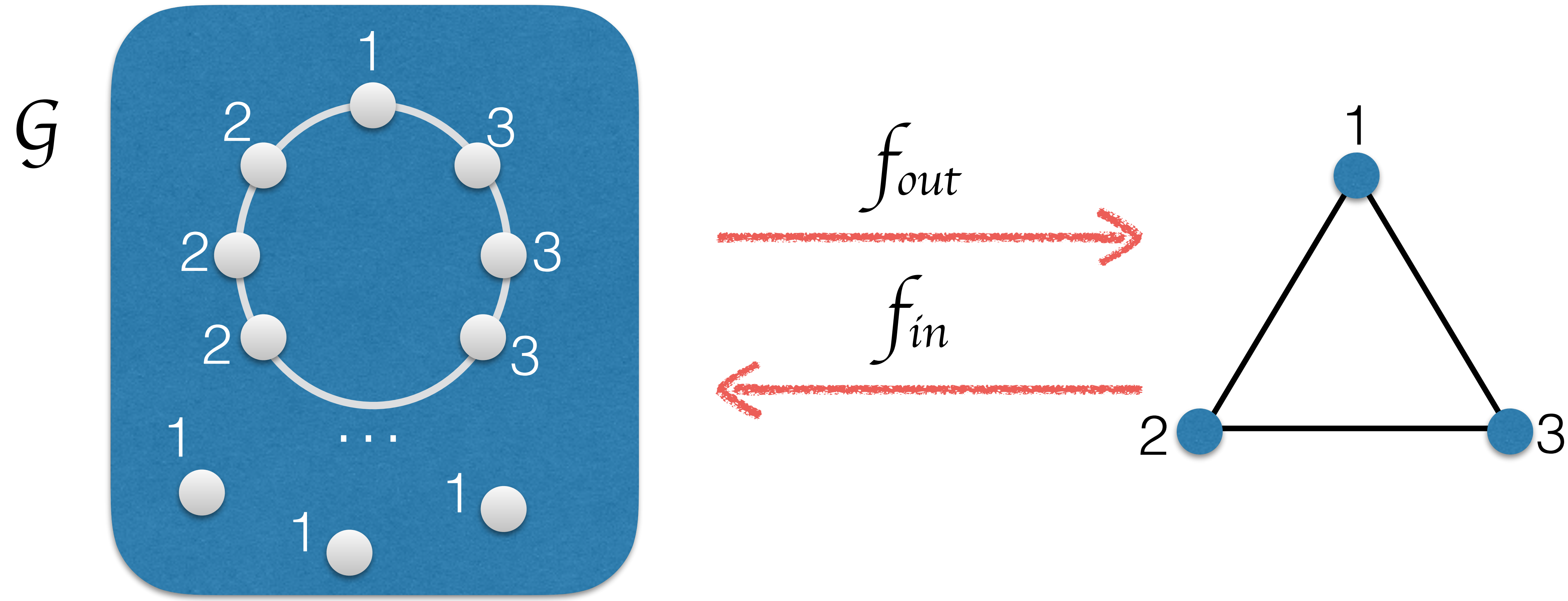
Solving Triangle from Cyclic Graphs



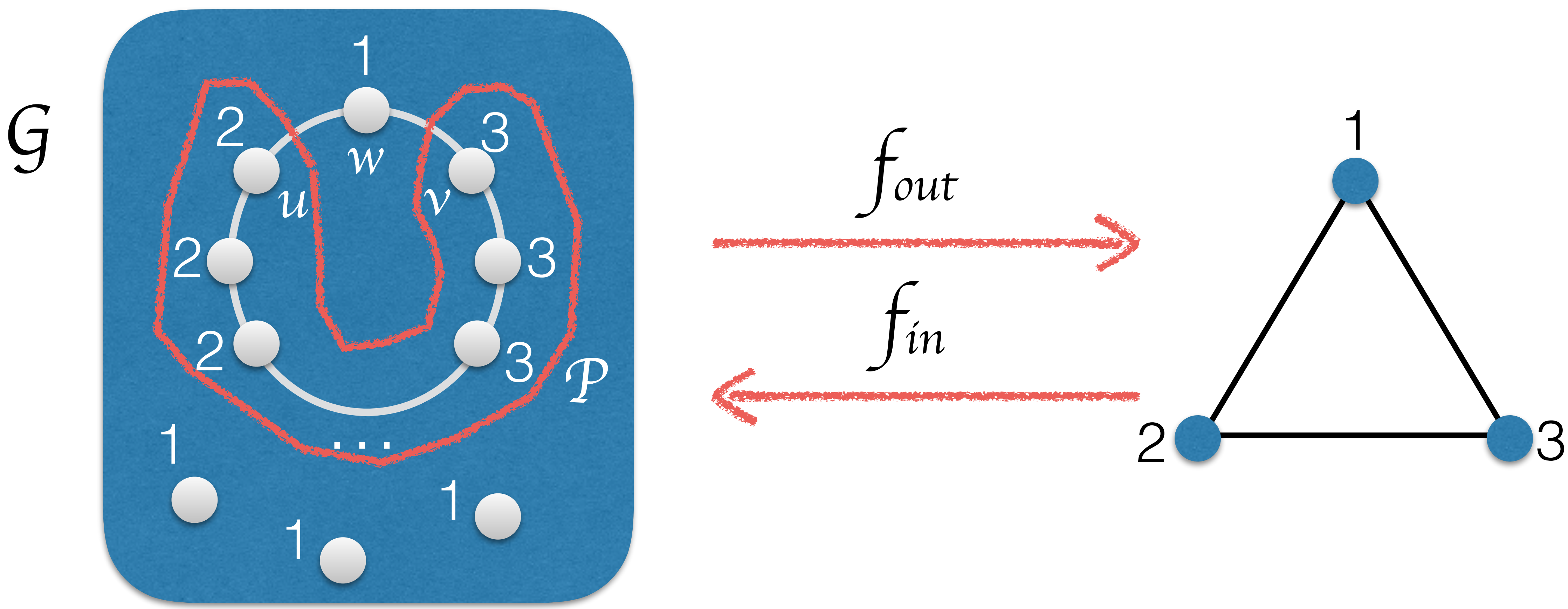
Solving Triangle from Cyclic Graphs



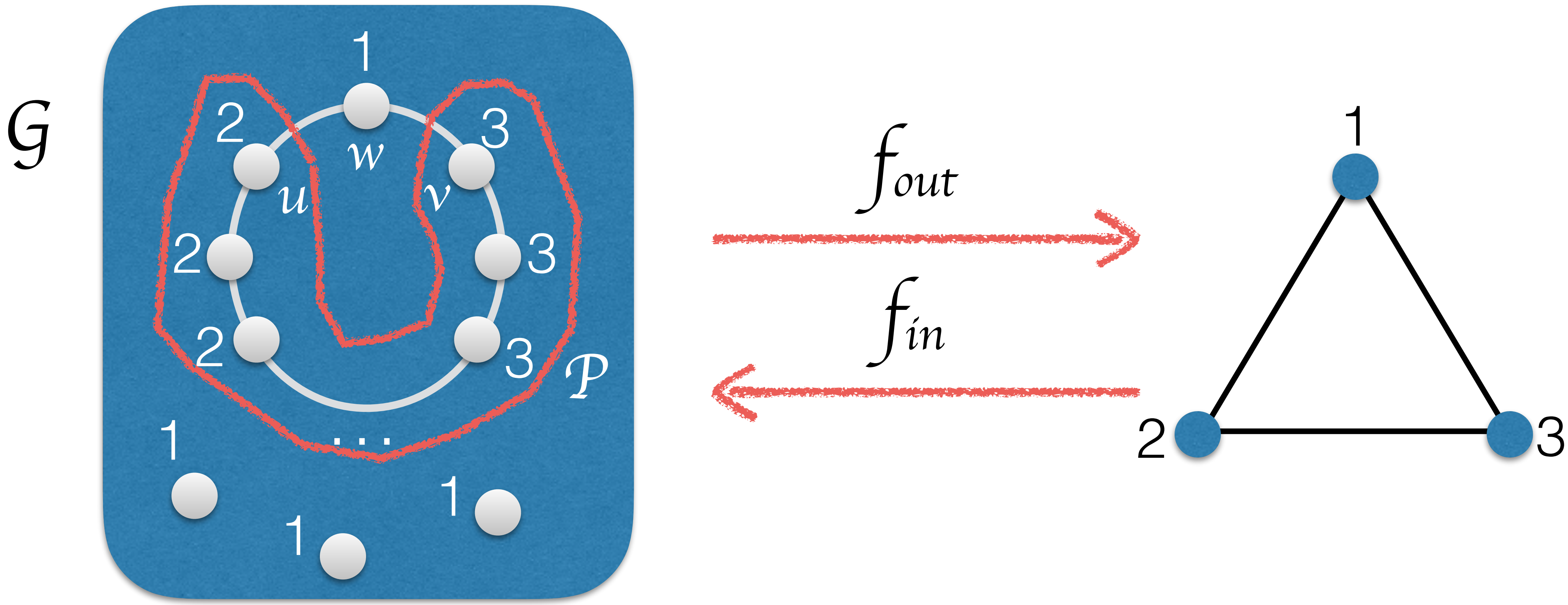
Solving Triangle from Cyclic Graphs



Solving Triangle from Cyclic Graphs



Solving Triangle from Cyclic Graphs



f$_{\text{in}}$(x):

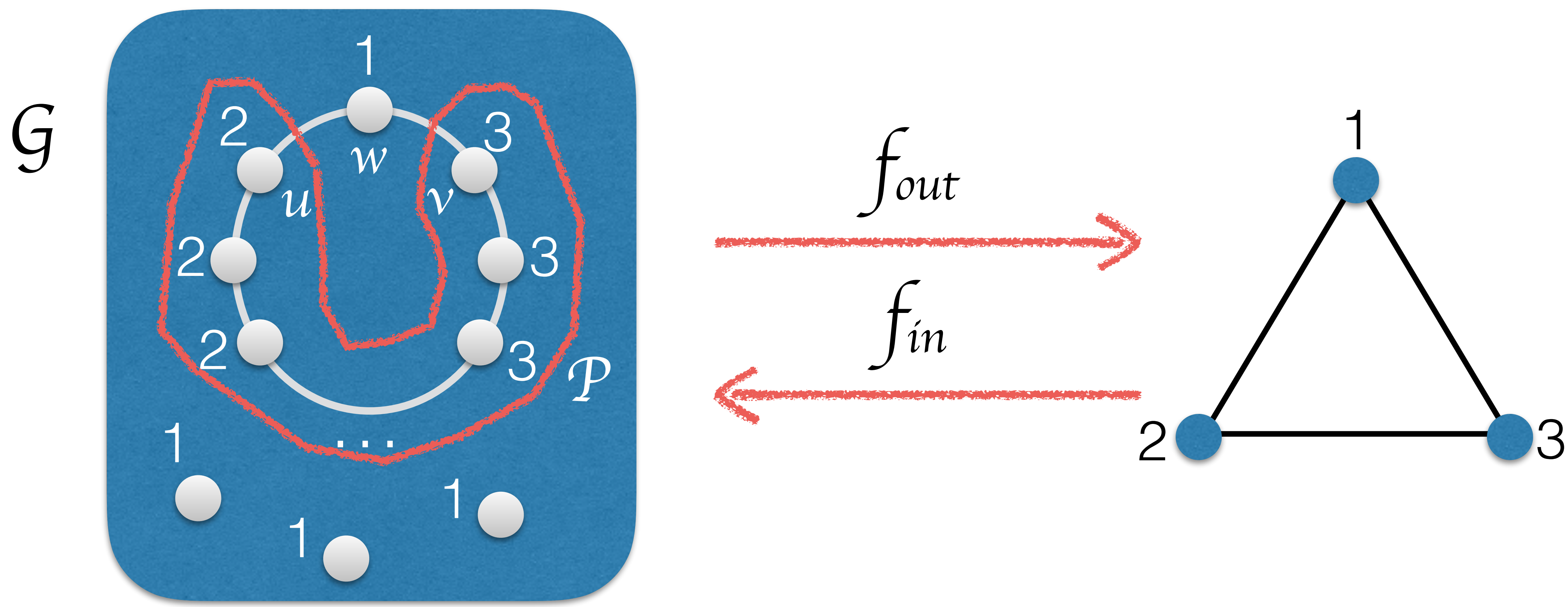
$\mathcal{A}$ = Edge gath. alg. on $\mathcal{P}$

if $x == 1$ **then return** w

elseif $x == 2$ then **return** $\mathcal{A}$.decide(u)

elseif $x == 3$ then **return** $\mathcal{A}$.decide(v)

Solving Triangle from Cyclic Graphs



$\mathrm{f_{in}}(x)$:

$\mathcal{A}$ = Edge gath. alg. on $\mathcal{P}$

if x == 1 **then return** w

elseif x == 2 then **return** $\mathcal{A}$.decide(u)

elseif x == 3 then **return** $\mathcal{A}$.decide(v)

EdgeGathTriangle(x):

$\mathcal{B}$ = Edge gath. alg. on $\mathcal{G}$

return f_{out}($\mathcal{B}$.decide(f_{in}(x)))

Let’s Do More

- **Edge Covering**:
  - **Termination.** Correct robots decide a vertex.
  - **Validity.** If participating robots start on the same vertex, they stay there. If start on an edge, decide vertices of the edge.
  - **Edge Covering.** If more than one decided vertex, decisions cover an edge.

Solvability of Edge Covering

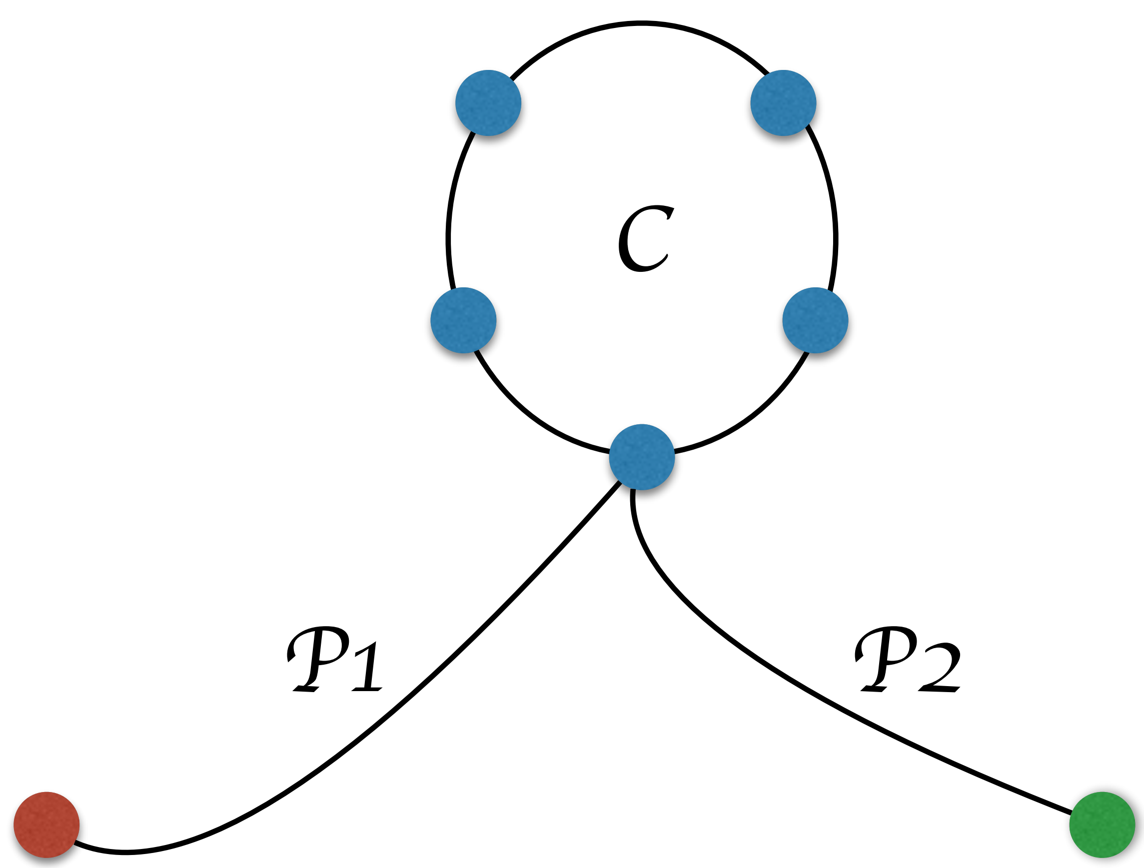
For n = 2, edge covering is solvable if and only if the base graph $\mathcal{G}$ is not bipartite

For n > 2, edge covering is imposible on every base graph $\mathcal{G}$

2-Robot Edge Covering Algorithm

For n = 2, if the base graph $\mathcal{G}$ is is not bipartite then edge covering is possible

There is an odd length path or cycle between any pair of nodes.

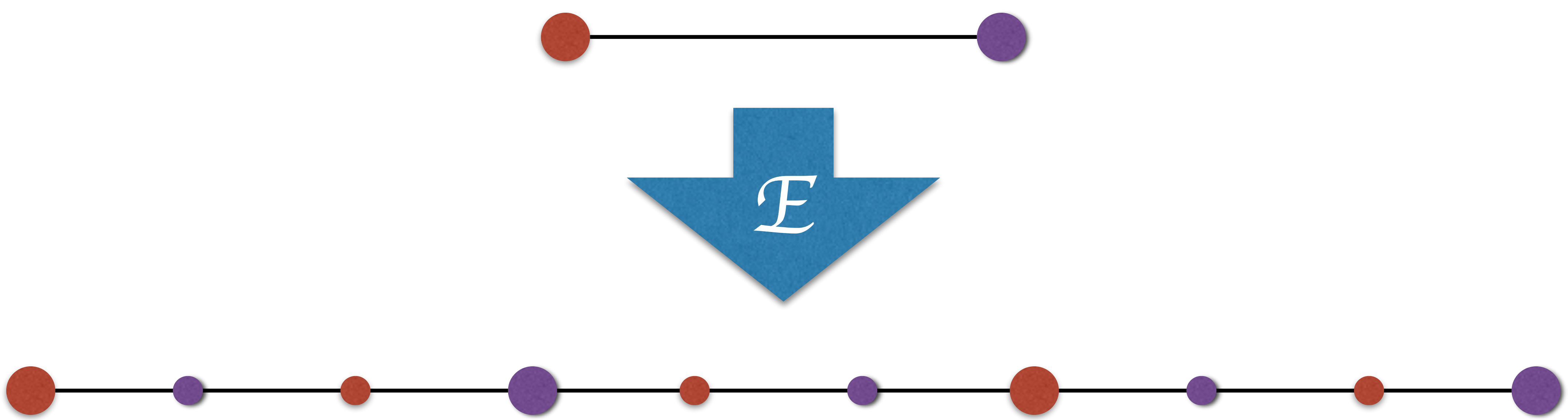


1) $|\mathcal{P}_1 — \mathcal{P}_2|$ is odd.
   Done

2) $|\mathcal{P}_1 — \mathcal{P}_2|$ is even.
   Take $\mathcal{P}_1 — \mathcal{C} — \mathcal{P}_2$.

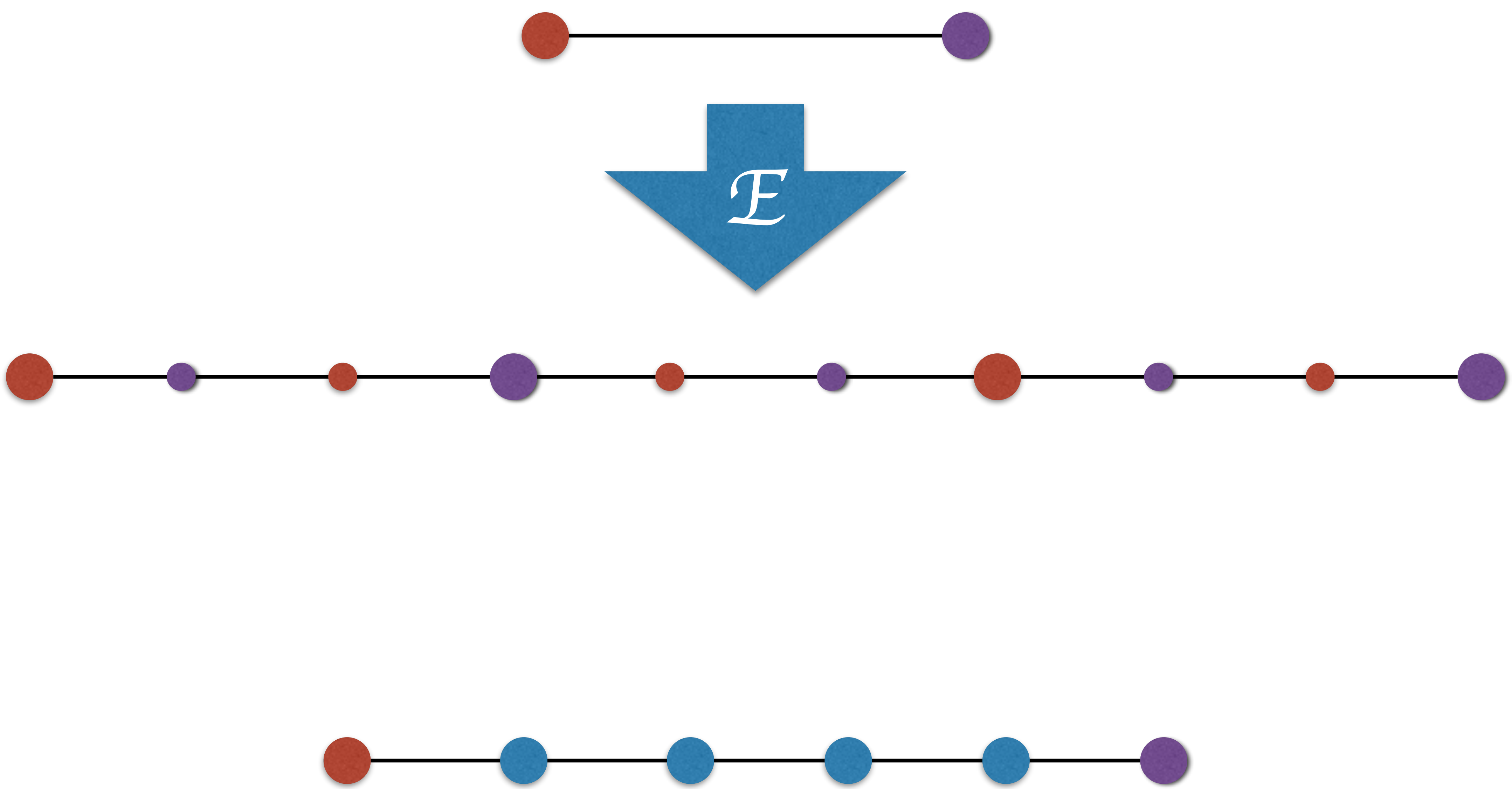
2-Robot Edge Covering Algorithm

Protocol complex (path) can be mapped to those paths.
Why? Length of the complex (path) is odd.



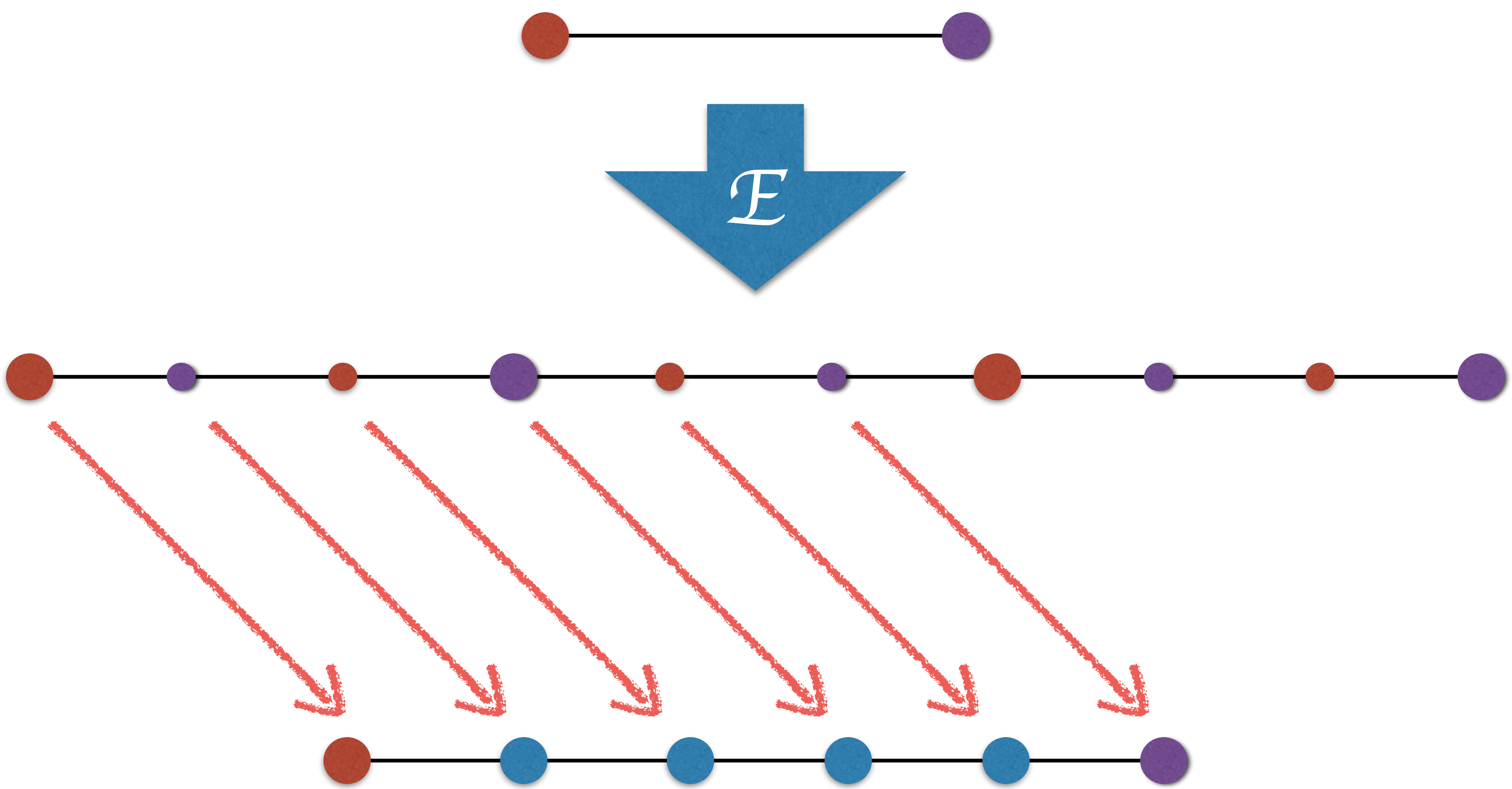
2-Robot Edge Covering Algorithm

Protocol complex (path) can be mapped to those paths.
Why? Length of the complex (path) is odd.



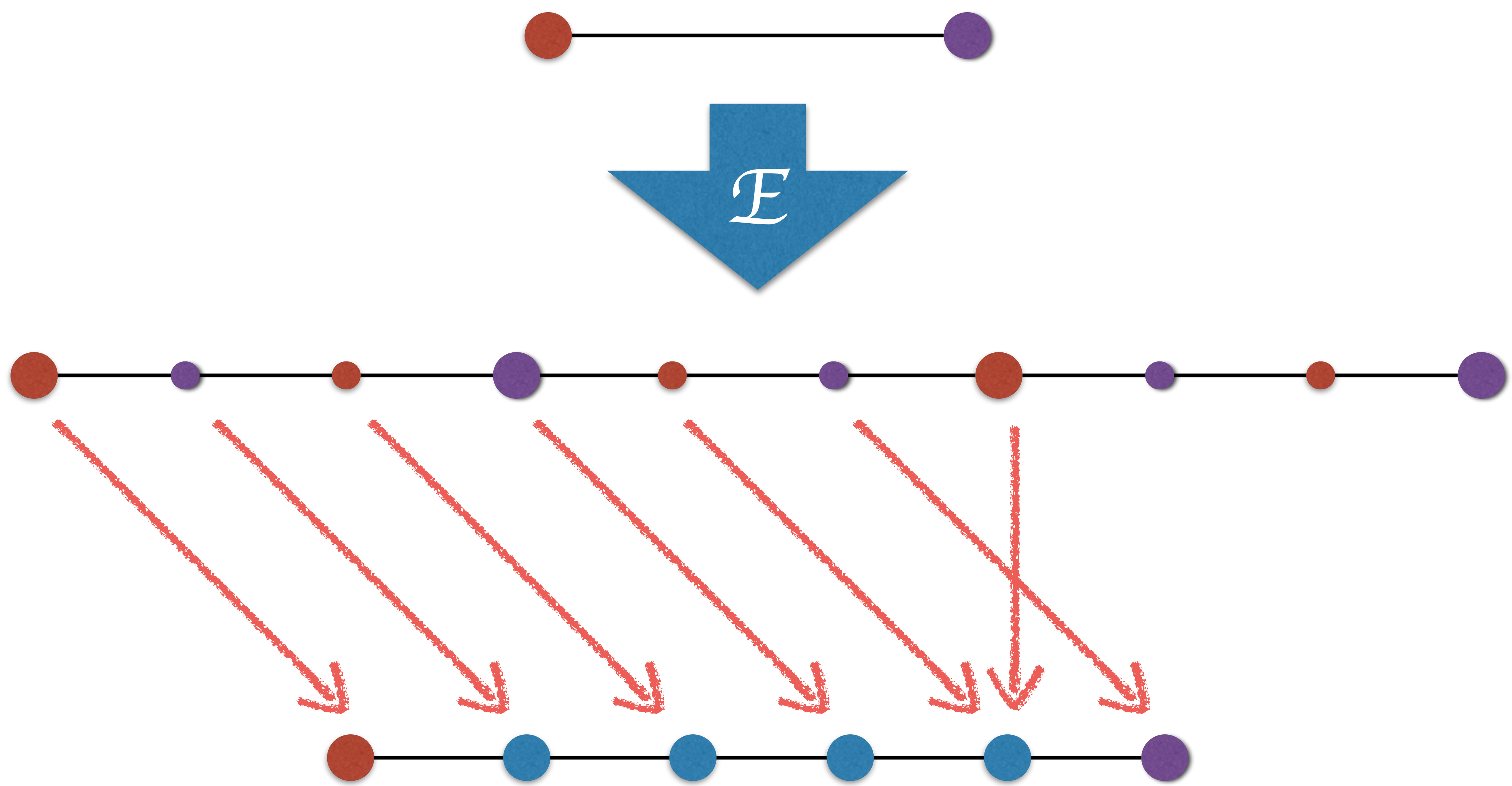
2-Robot Edge Covering Algorithm

Protocol complex (path) can be mapped to those paths.
Why? Length of the complex (path) is odd.



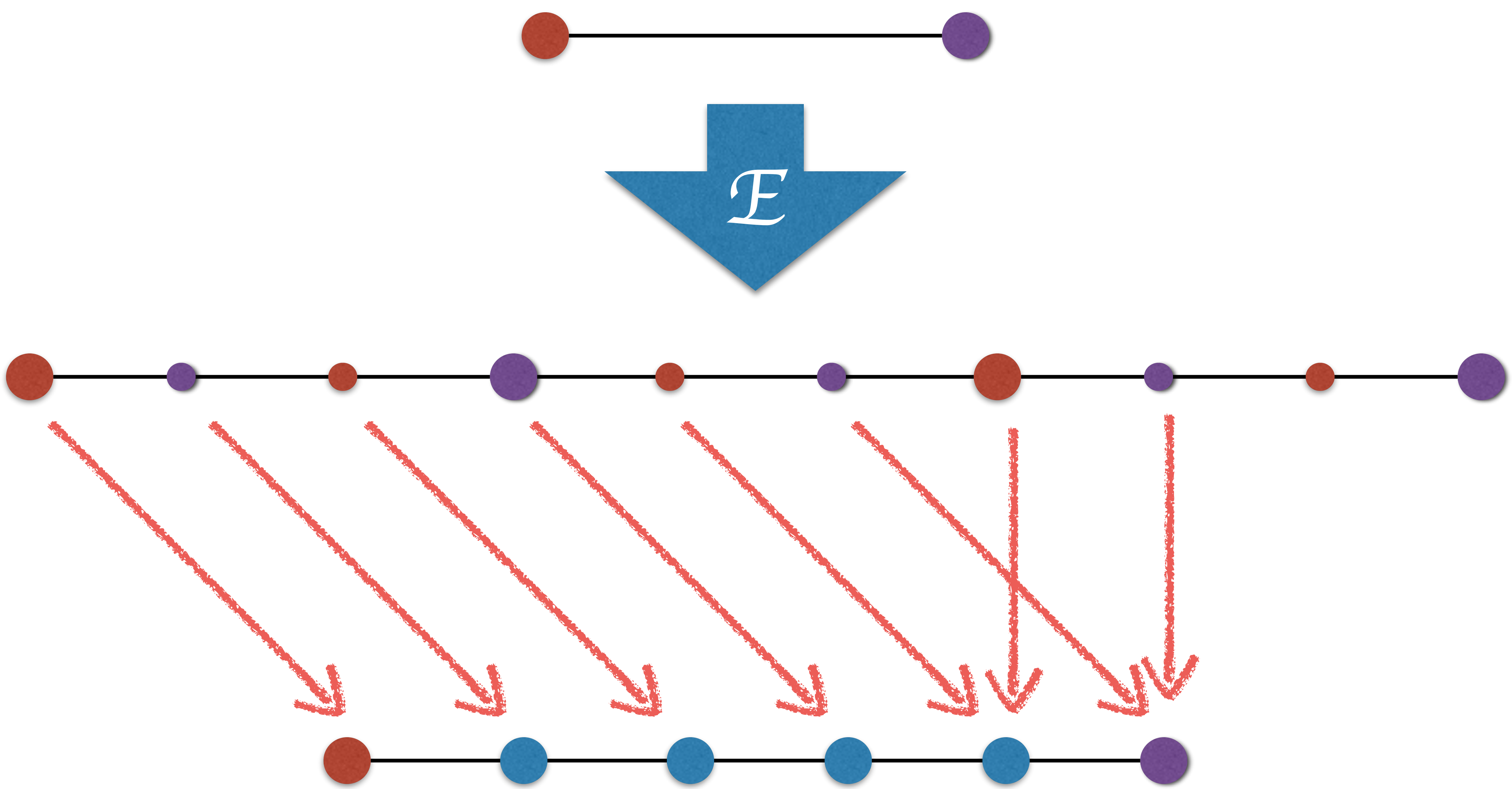
2-Robot Edge Covering Algorithm

Protocol complex (path) can be mapped to those paths.
Why? Length of the complex (path) is odd.



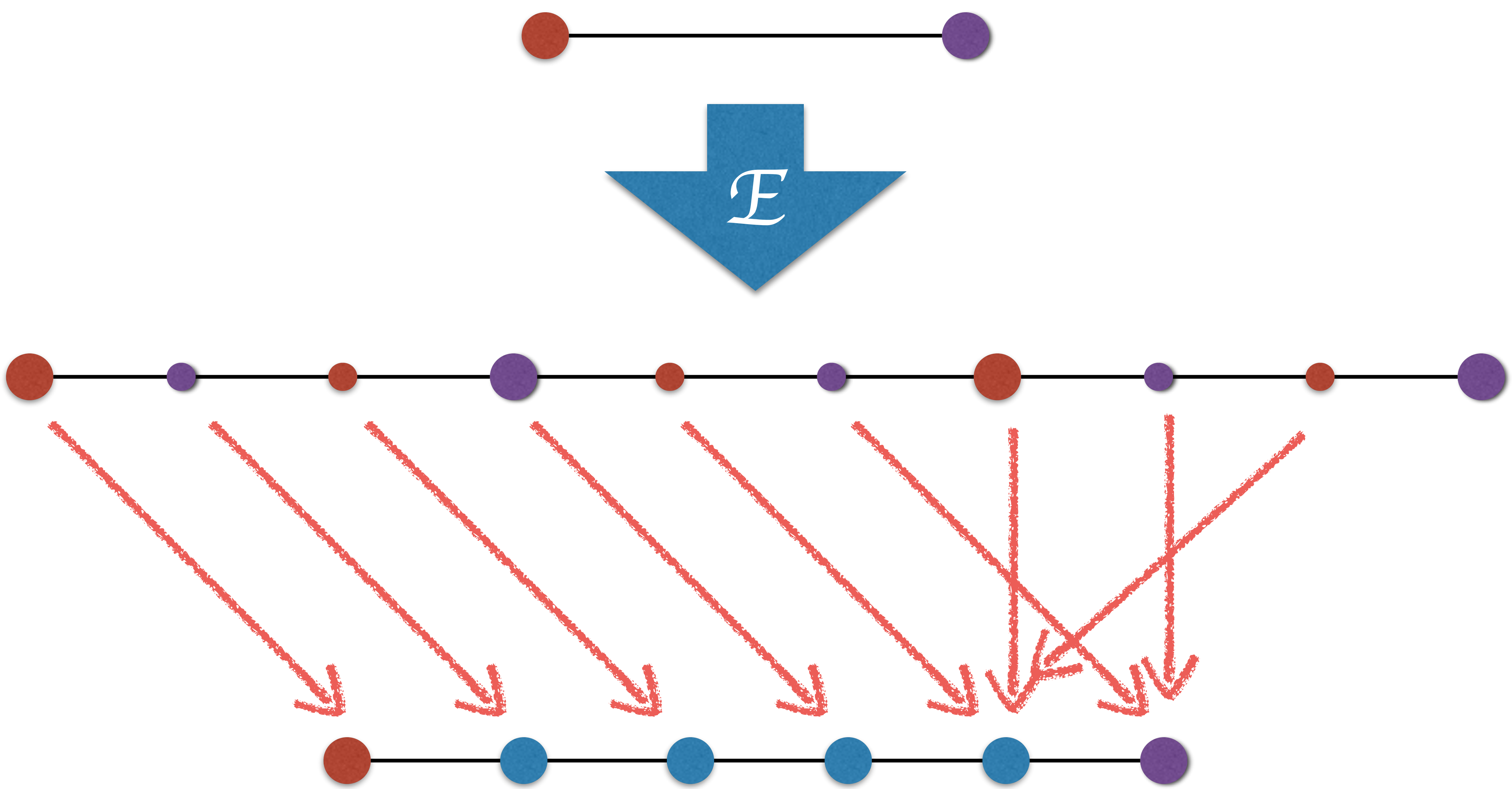
2-Robot Edge Covering Algorithm

Protocol complex (path) can be mapped to those paths.
Why? Length of the complex (path) is odd.



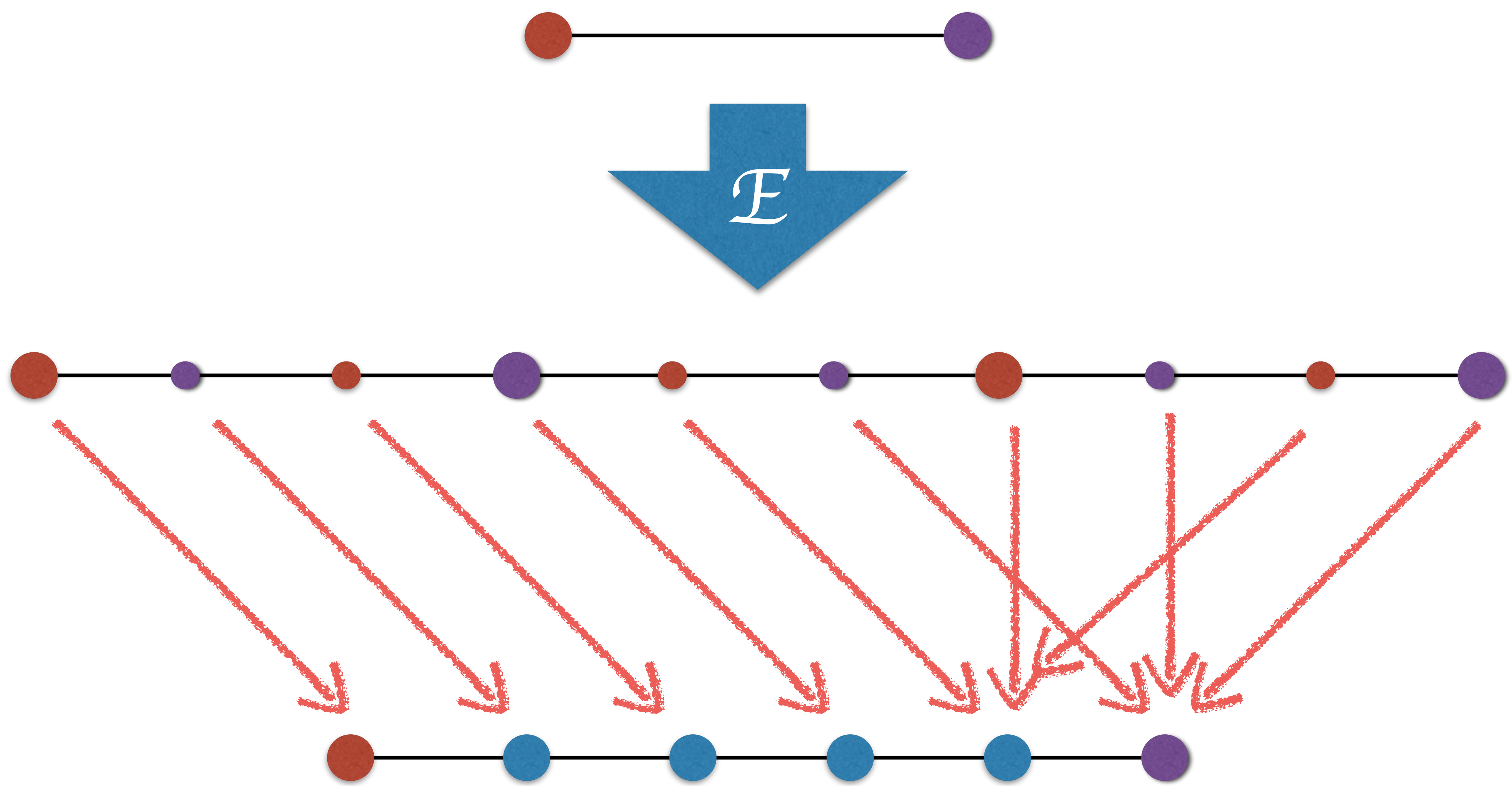
2-Robot Edge Covering Algorithm

Protocol complex (path) can be mapped to those paths.
Why? Length of the complex (path) is odd.



2-Robot Edge Covering Algorithm

Protocol complex (path) can be mapped to those paths.
Why? Length of the complex (path) is odd.



2-Robot Edge Covering Impossibility

For n = 2, if the base graph $\mathcal{G}$ is bipartite, then edge covering is impossible

- Bipartite => for some pair, there is no odd length path or cycle
- Protocol complex cannot be mapped to even length paths: Endpoints to endpoints and edges to edges.

2-Robot Edge Covering Impossibility

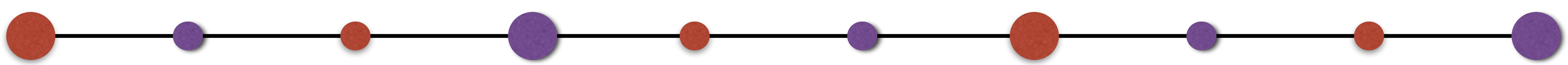
For n = 2, if the base graph $\mathcal{G}$ is bipartite, then edge covering is impossible

Proof:

1. Prove the edge is impossible.
2. Solve the edge from any bipartite graph.

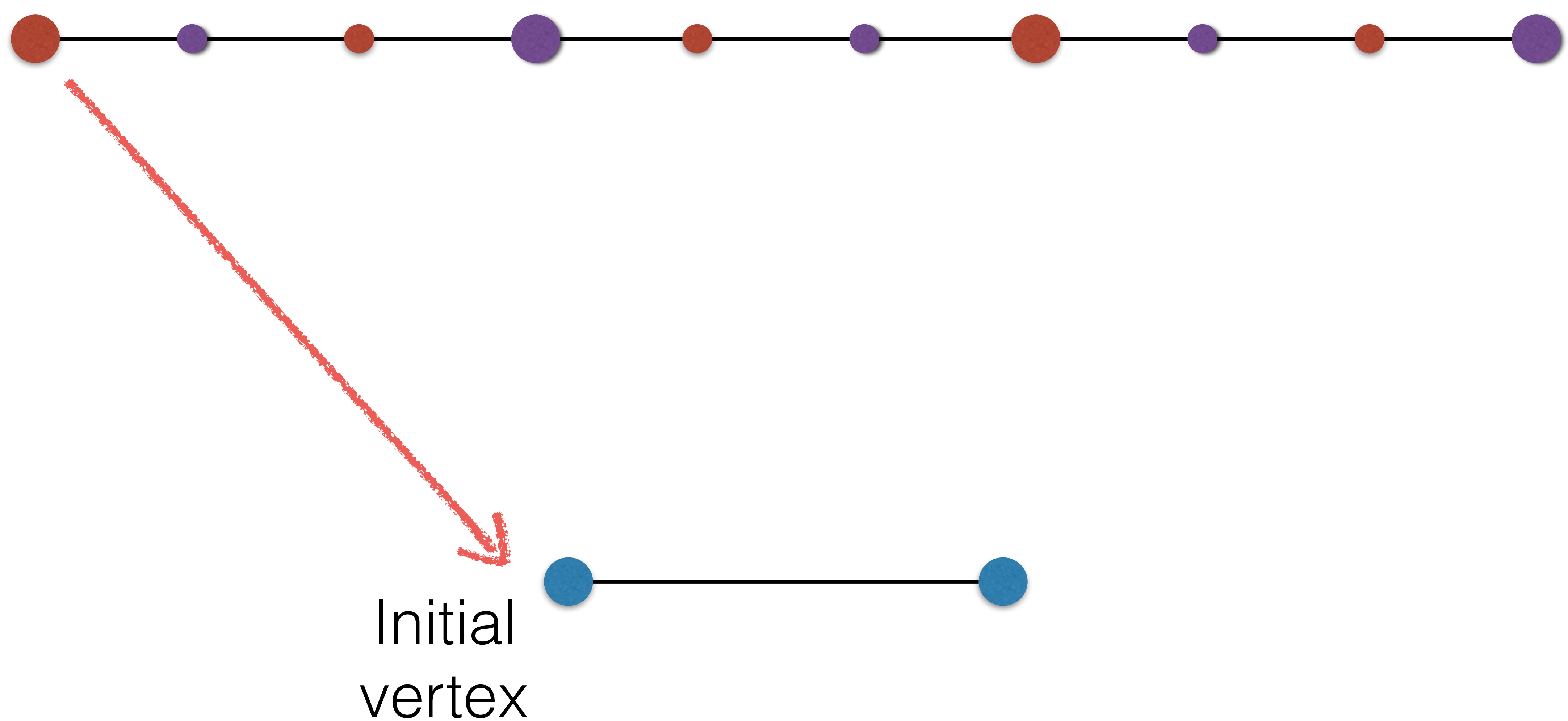
Impossibility of Edge Covering on Edge

What if the two robots start on the same vertex?



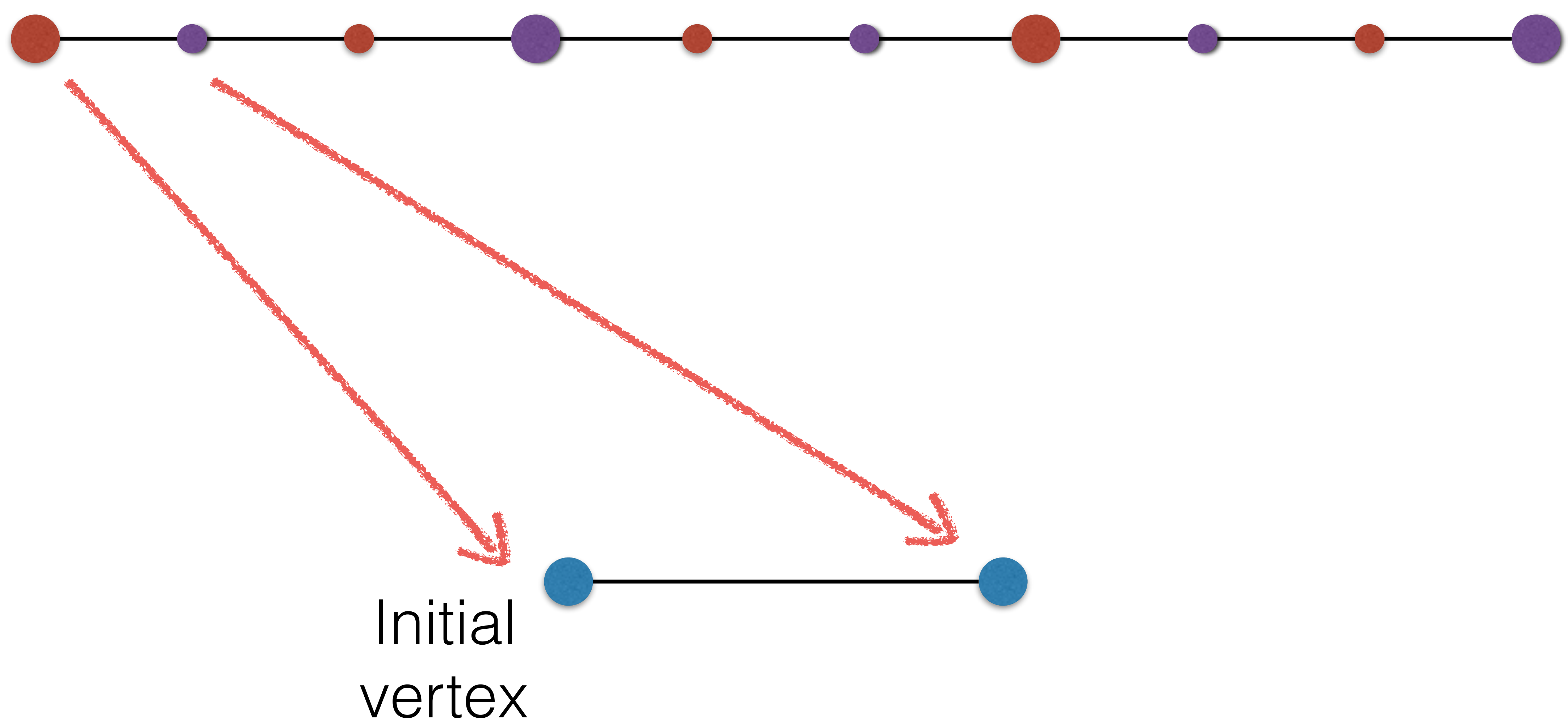
Impossibility of Edge Covering on Edge

What if the two robots start on the same vertex?



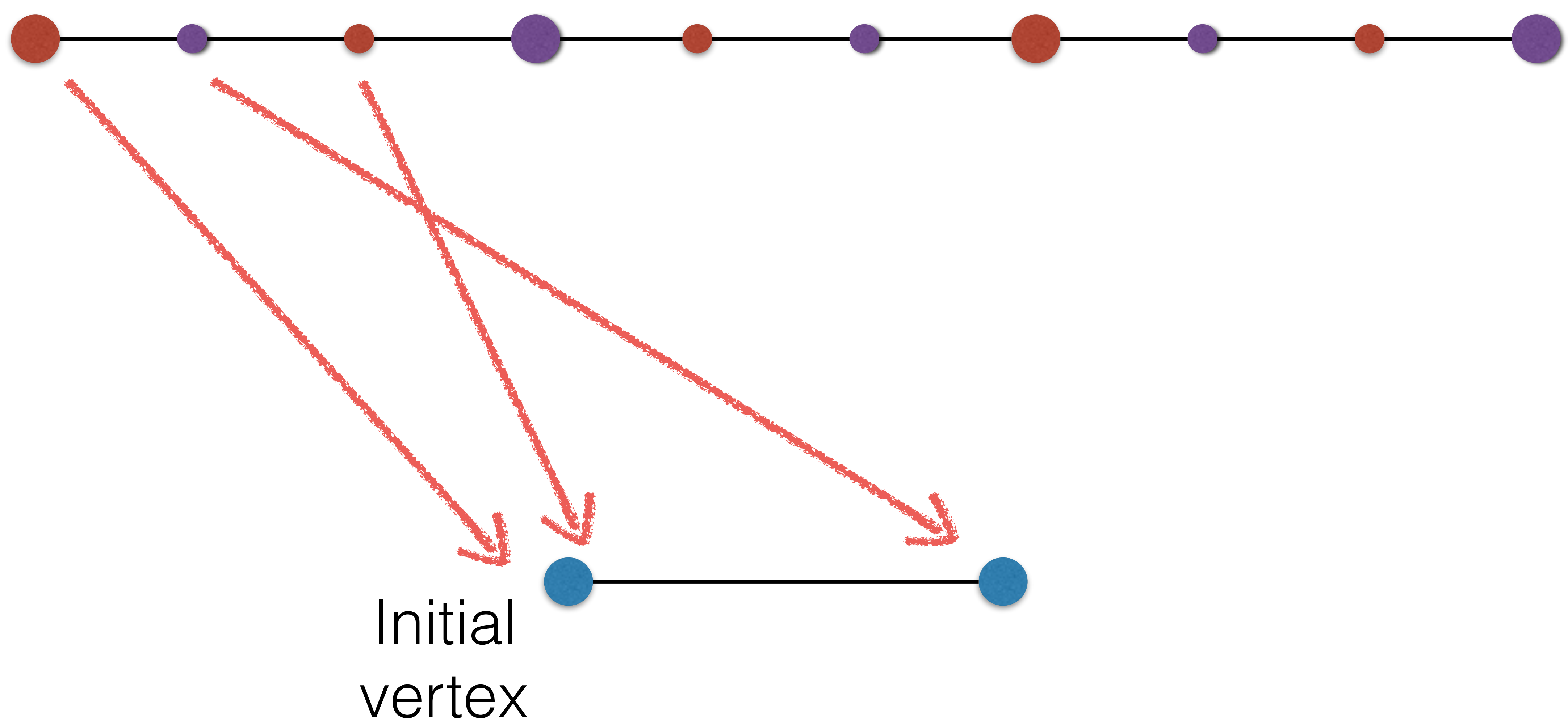
Impossibility of Edge Covering on Edge

What if the two robots start on the same vertex?



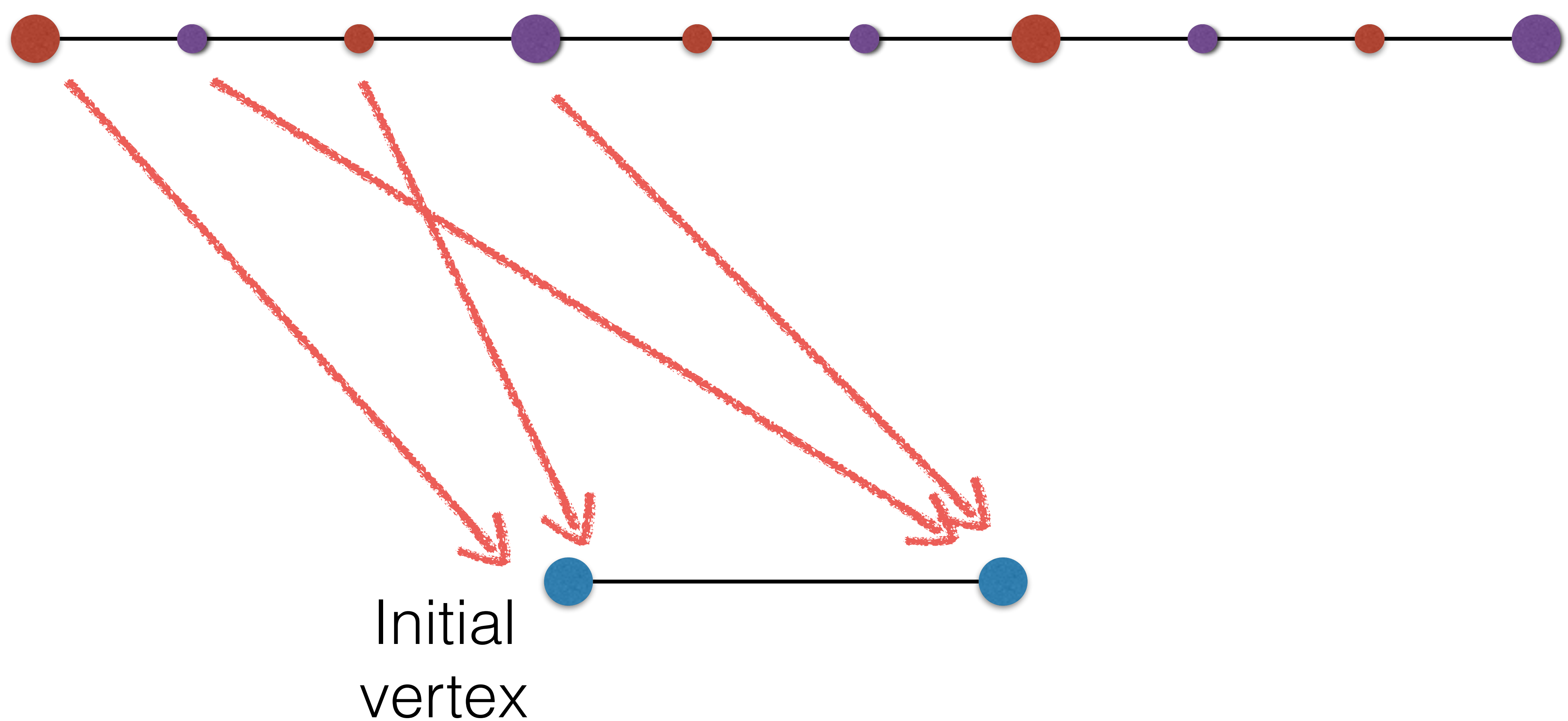
Impossibility of Edge Covering on Edge

What if the two robots start on the same vertex?



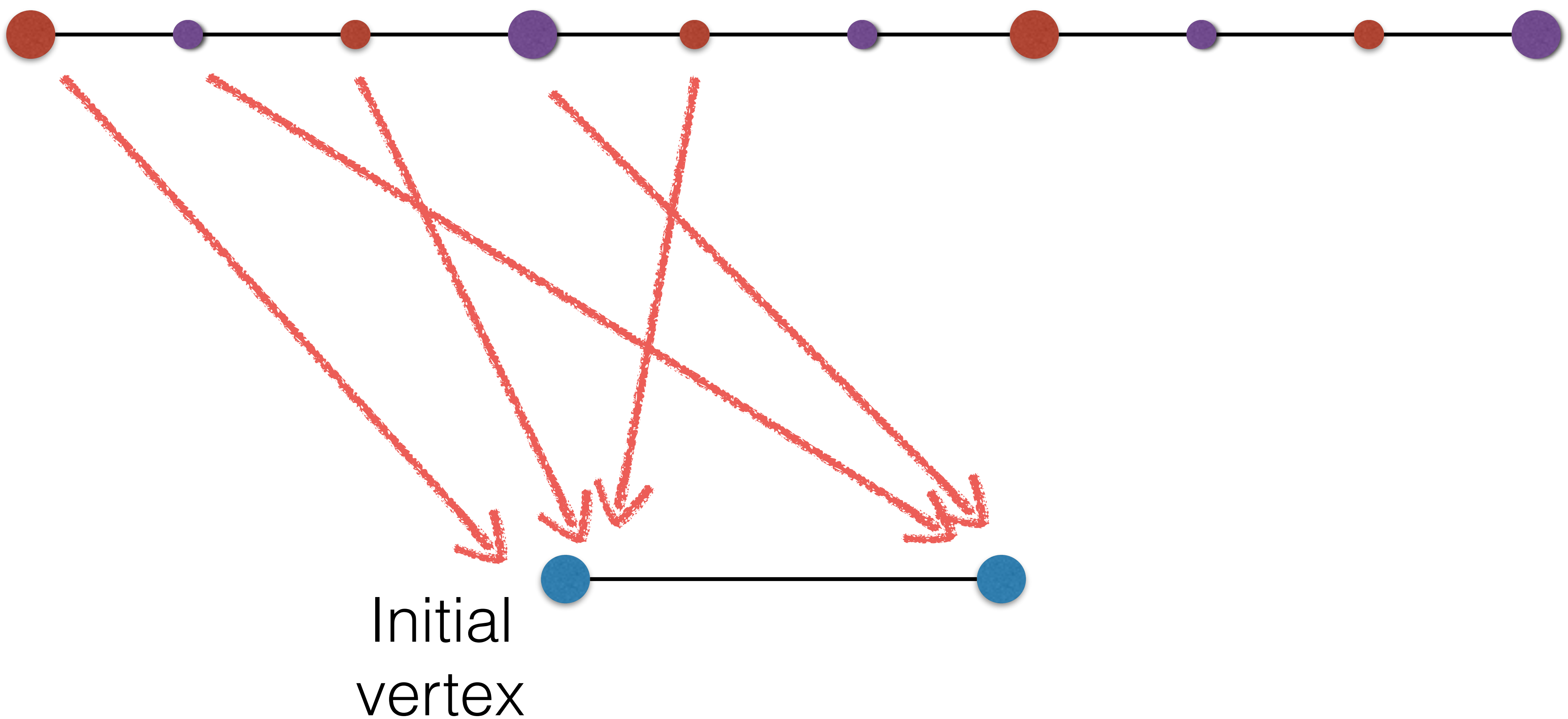
Impossibility of Edge Covering on Edge

What if the two robots start on the same vertex?



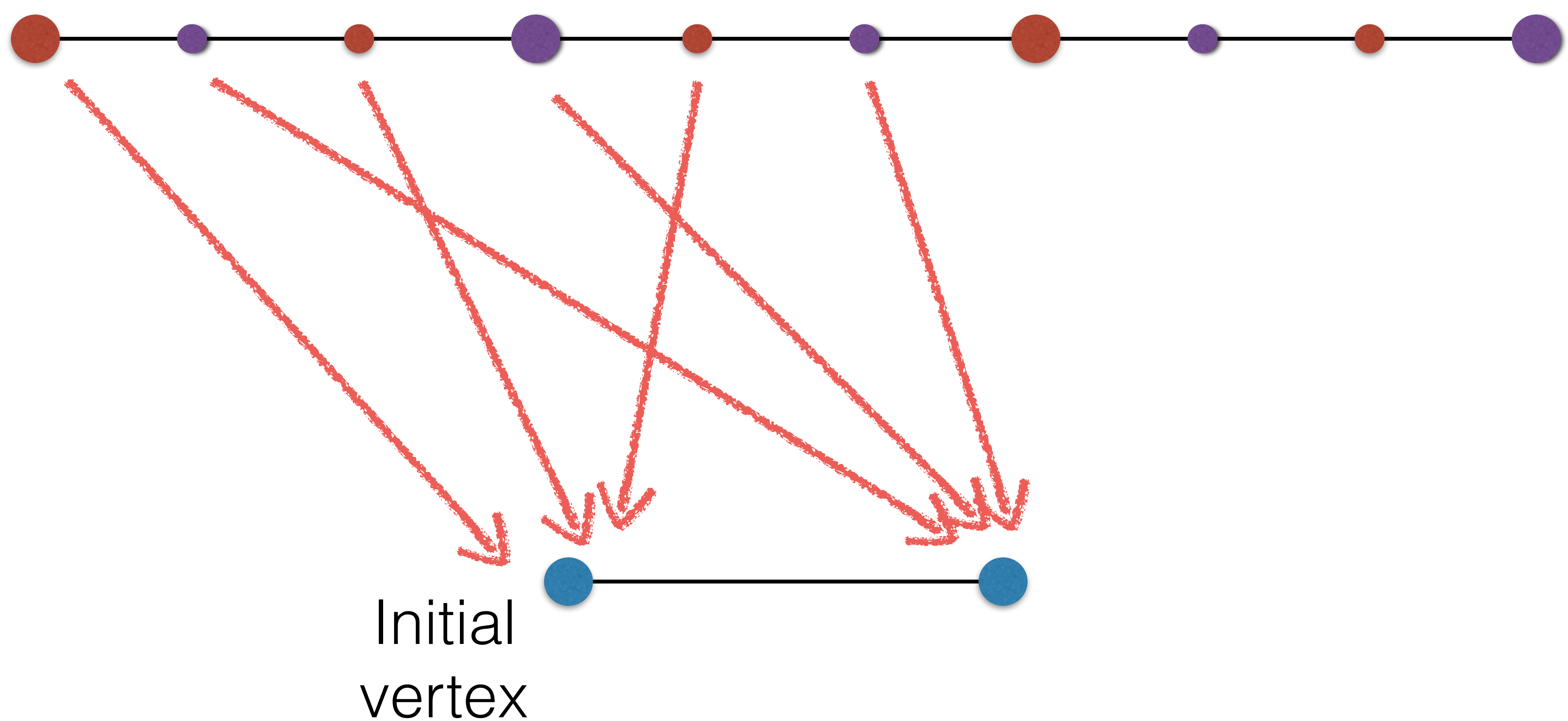
Impossibility of Edge Covering on Edge

What if the two robots start on the same vertex?



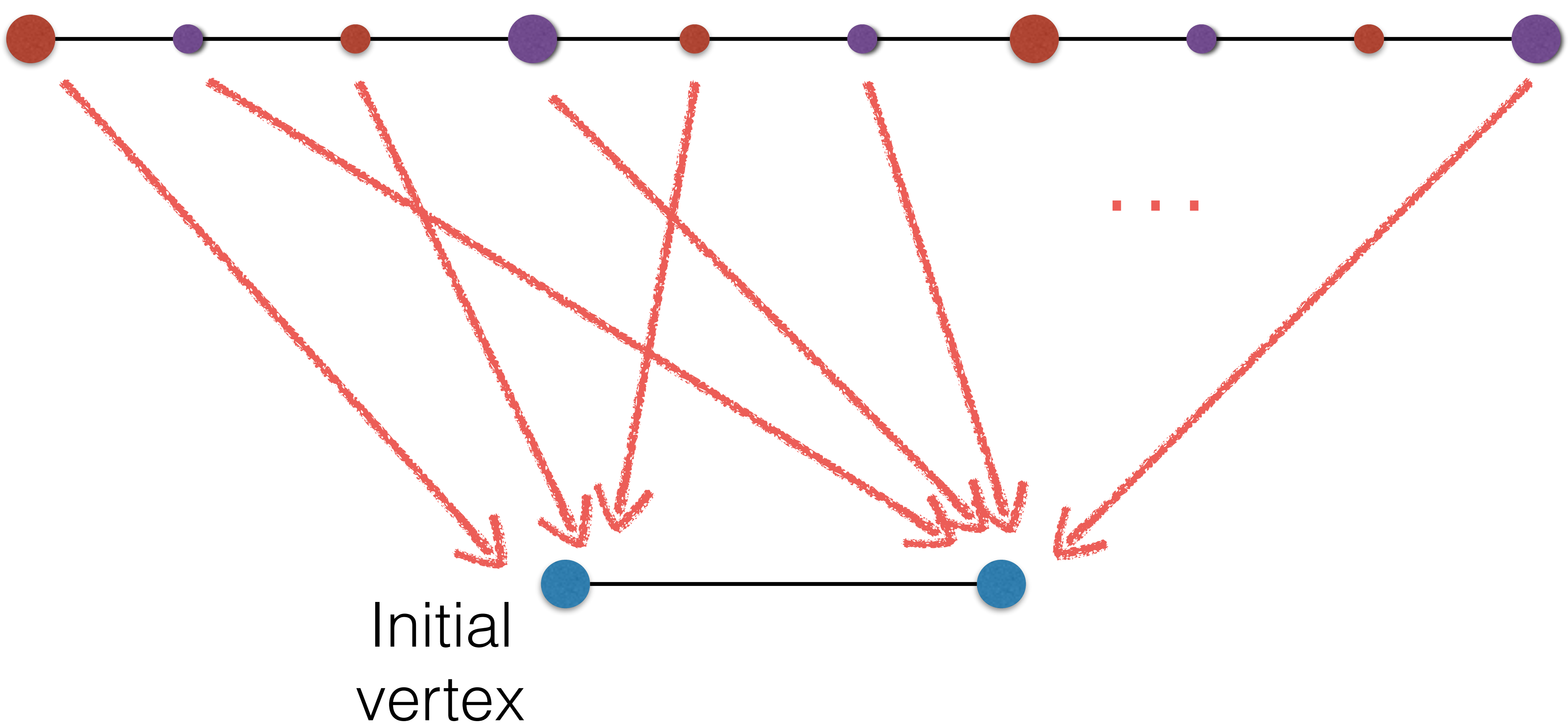
Impossibility of Edge Covering on Edge

What if the two robots start on the same vertex?



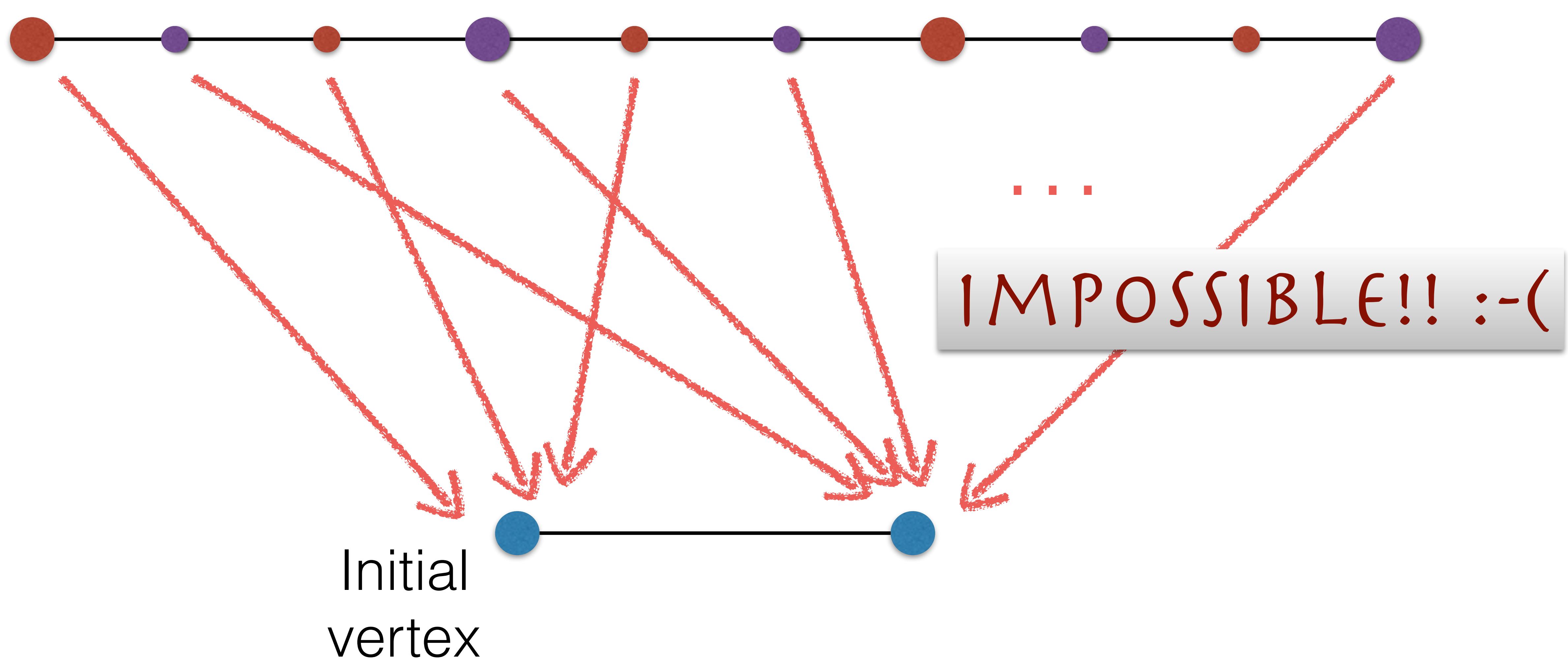
Impossibility of Edge Covering on Edge

What if the two robots start on the same vertex?



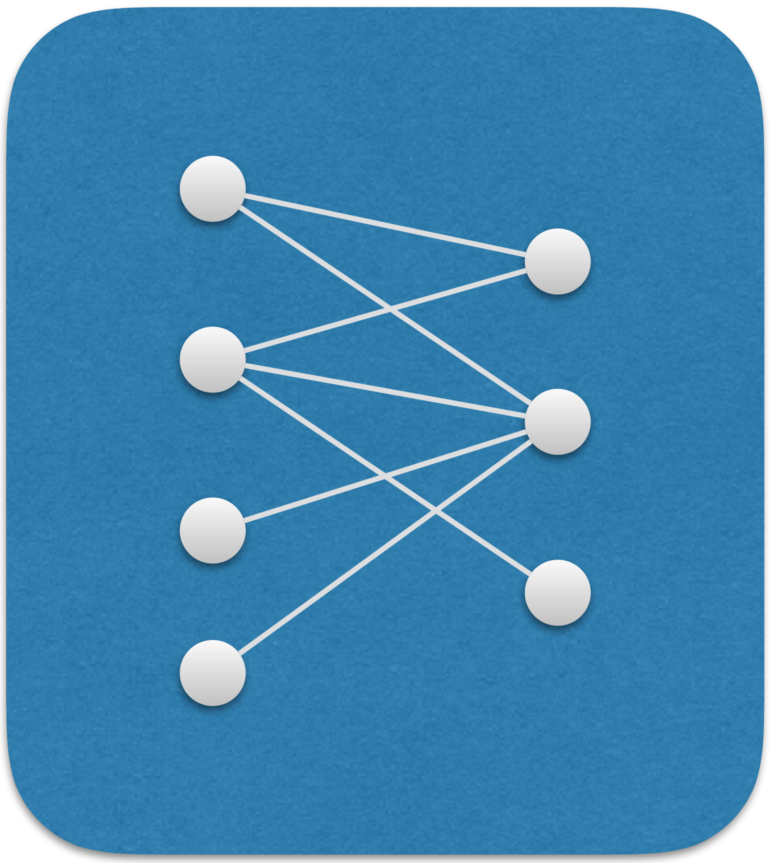
Impossibility of Edge Covering on Edge

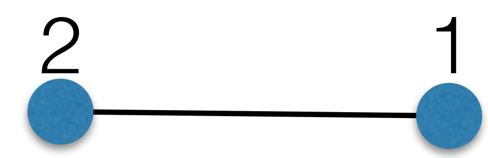
What if the two robots start on the same vertex?



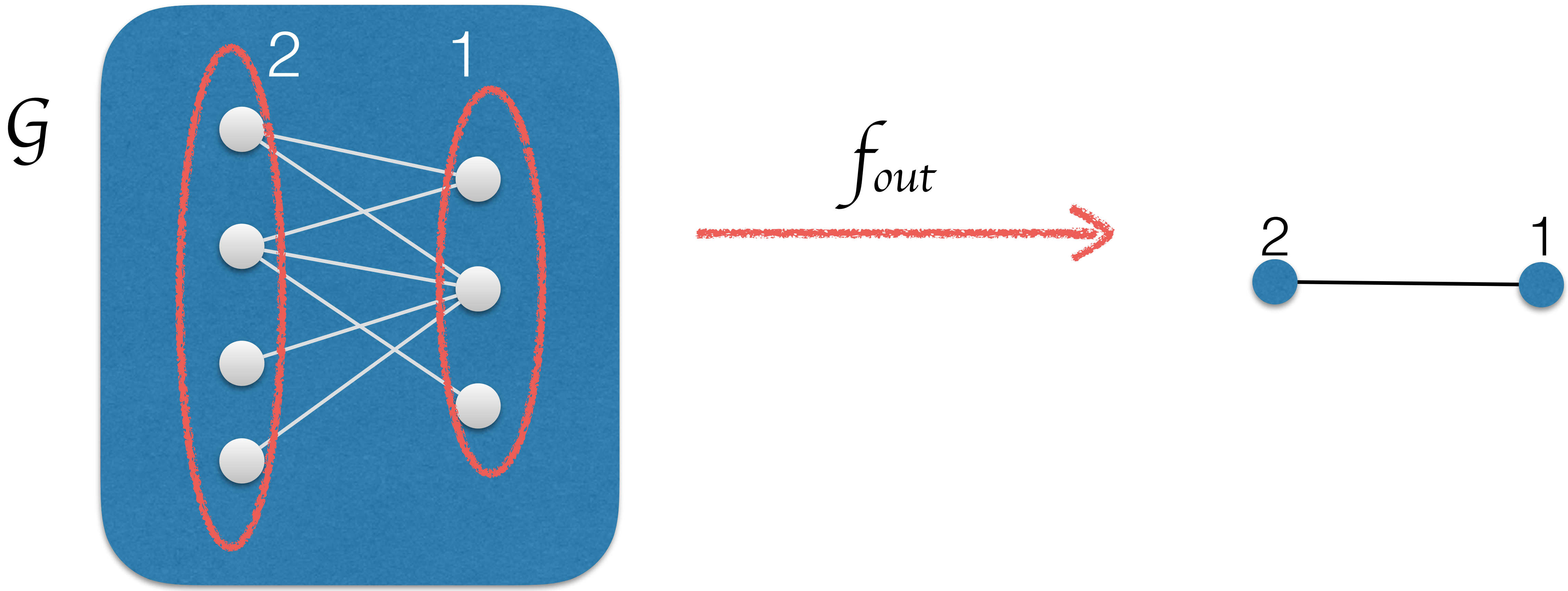
Solving Edge from Bipartite Graphs

$\mathcal{G}$

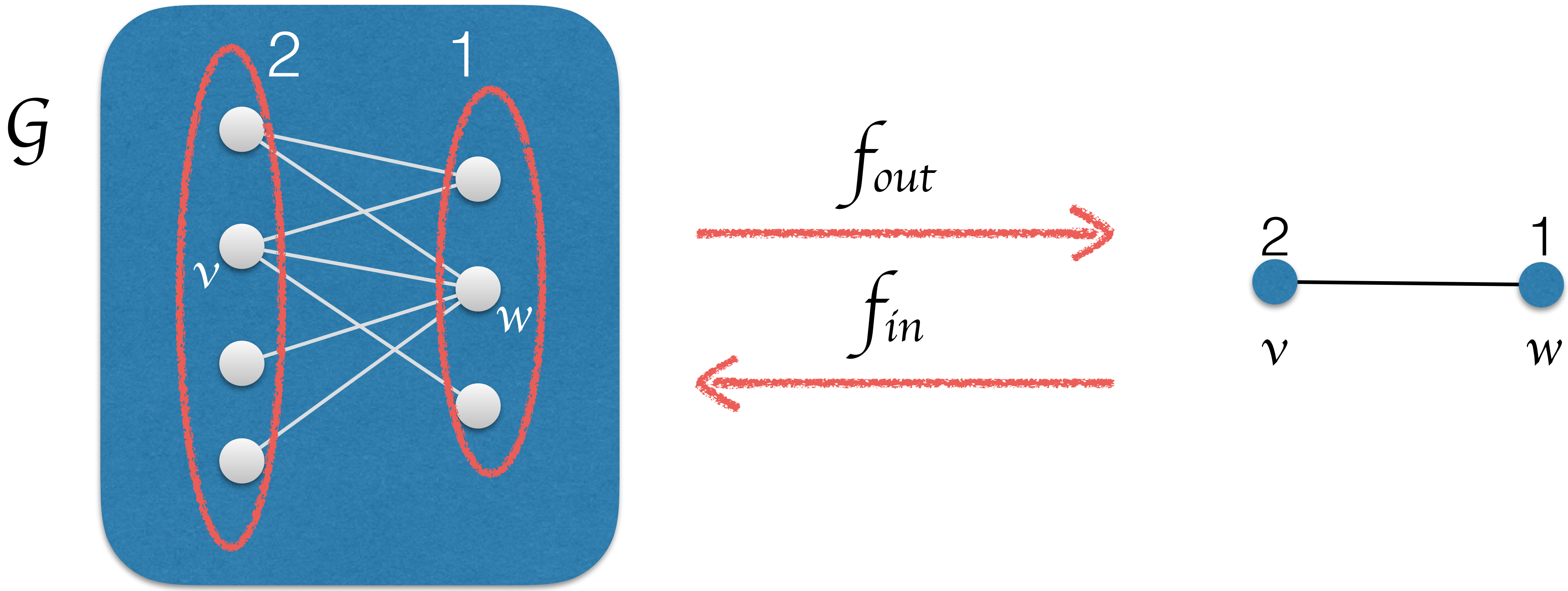




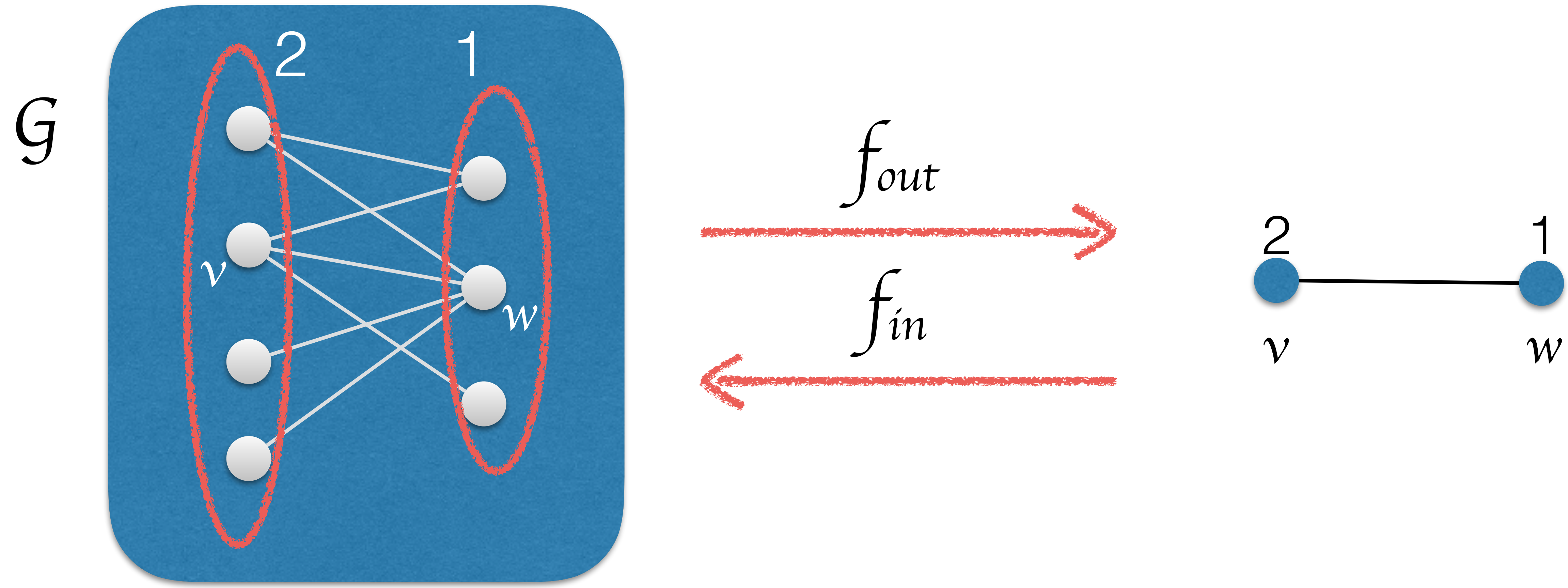
Solving Edge from Bipartite Graphs



Solving Edge from Bipartite Graphs



Solving Edge from Bipartite Graphs



EdgeCoveringEdge(x):

$\mathcal{B}$ = Edge covering alg. on $\mathcal{G}$

return $f_{out}(\mathcal{B}.\text{decide}(f_{in}(\text{x})))$

Edge Covering Impossibility

For n > 2, edge covering is imposible on every base graph $\mathcal{G}$

Proof:

1. Suppose there is an edge covering algorithm $\mathcal{A}$ on $\mathcal{G}$.
2. $\mathcal{A}$ solves 2-robot edge covering on $\mathcal{G}$.
3. $\mathcal{G}$ is not bipartite => $\mathcal{G}$ has cycles.
4. $\mathcal{A}$ solves edge gathering on $\mathcal{G}$ for n > 2 robots. Contradiction!!

Summary

1. Gathering. Impossible
2. Edge Gathering:
    - For $n=2$, solvable on any graph.
    - For $n>2$, solvable if an only if acyclic.
3. Edge Covering:
    - For $n=2$, solvable if an only if not bipartite.
    - For $n>2$, impossible.

ALR = R/W Wait-Free

A task (maybe non-colorless) is solvable in ALR if and only if it is solvable in Async. R/W Wait-Free

Reduction based proofs:

1. Same connectivity properties.
2. Gathering => Consensus
3. Edge Gathering => 2-Set Consensus
4. Edge Covering => WSB