Byzantine Computability and Combinatorial Topology

Hammurabi Mendes
University of Rochester

Applied Algebraic Topology Research Network
November 10, 2015

joint work with Maurice Herlihy, Christine Tasson, done at Brown University

Outline

1. Introduction
2. Asynchronous Byzantine Systems
3. Synchronous Byzantine Systems
4. Conclusion & Future Work

Outline

1. Introduction
2. Asynchronous Byzantine Systems
3. Synchronous Byzantine Systems
4. Conclusion & Future Work

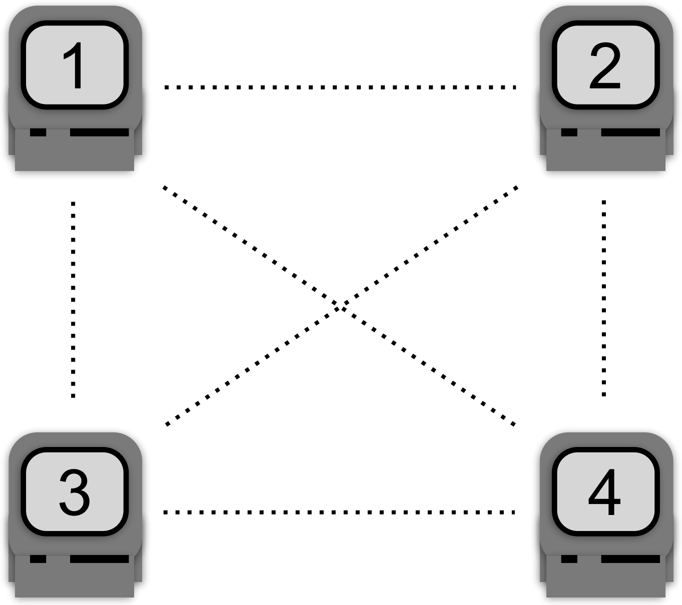
Introduction

Introduction

Tasks:

Introduction

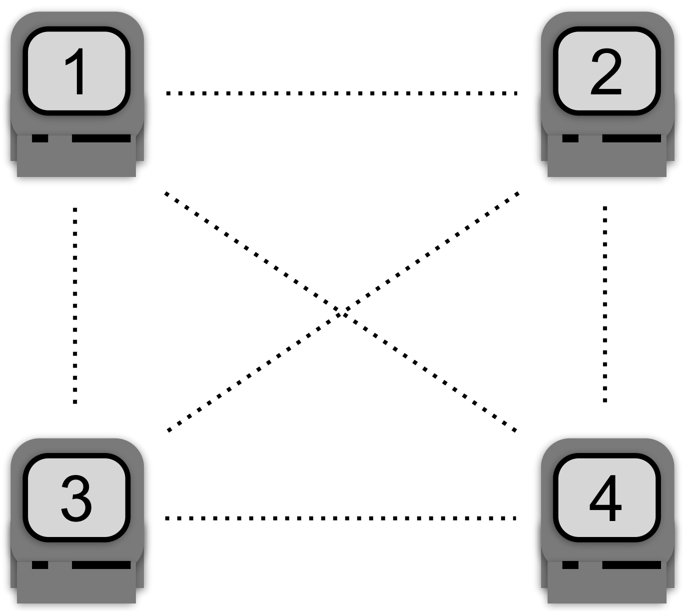
Tasks:



Introduction

Tasks:

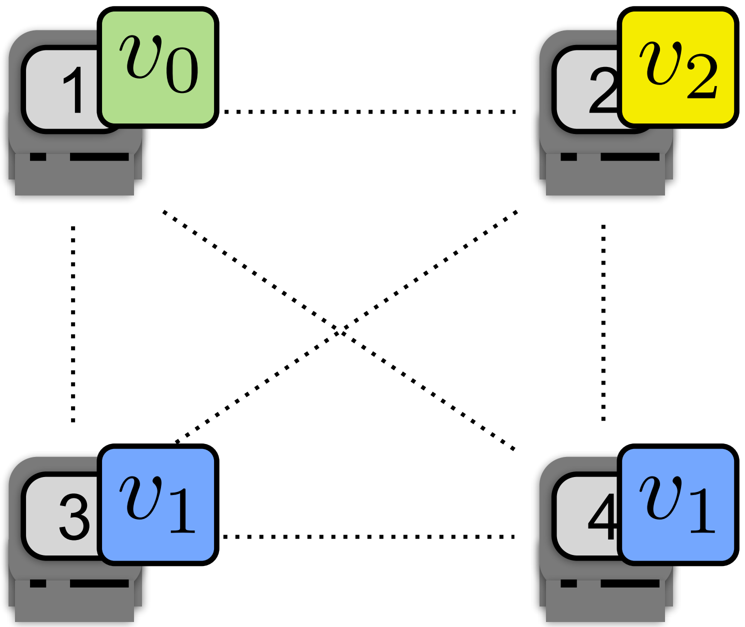
Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$



Introduction

Tasks:

Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$



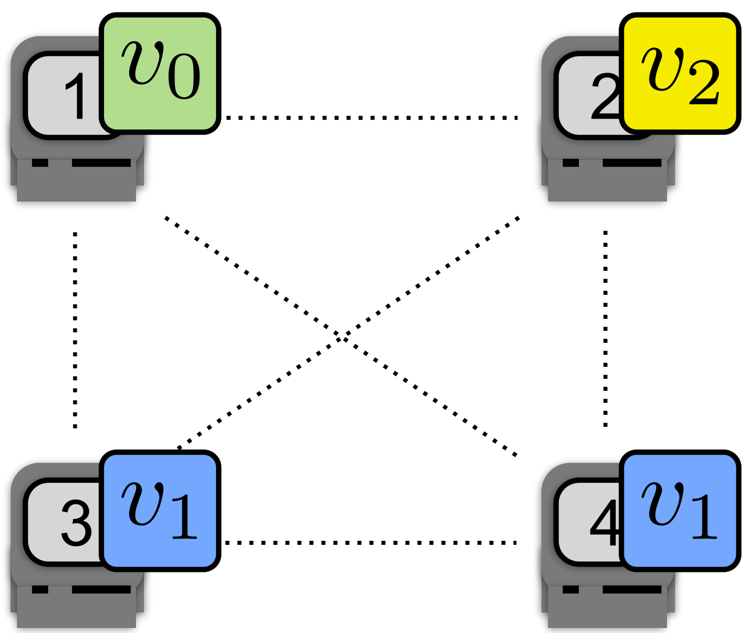
input

Introduction

Tasks:

Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$

Processes output values a single proposed value



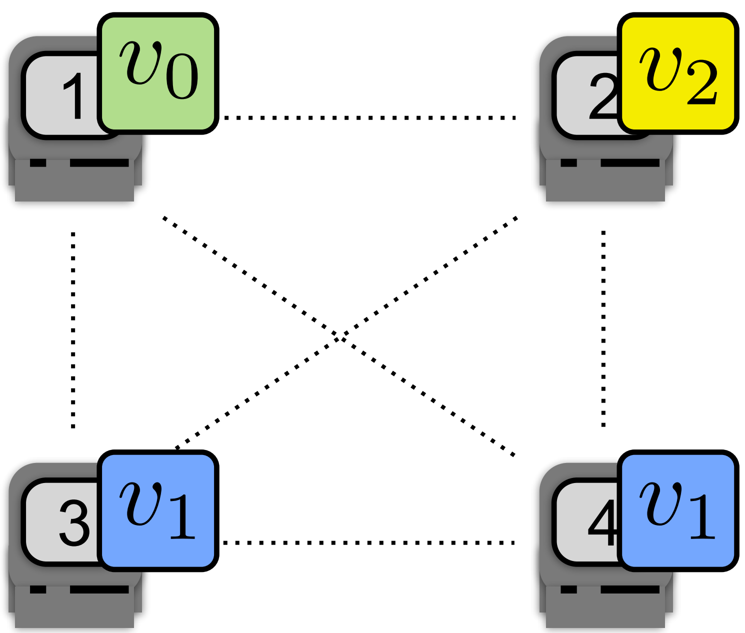
input

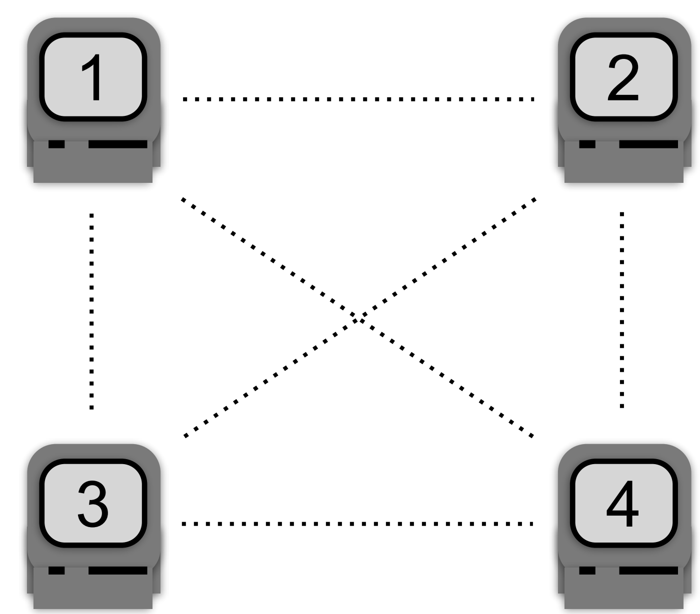
Introduction

Tasks:

Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$

Processes output values a single proposed value





input

Tasks:

Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$

Processes output values a single proposed value

Tasks:

Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$

Processes output values a single proposed value

input

output

Introduction

Tasks:

Processes input values from a set { v_0 v_1 v_2 v_3 }

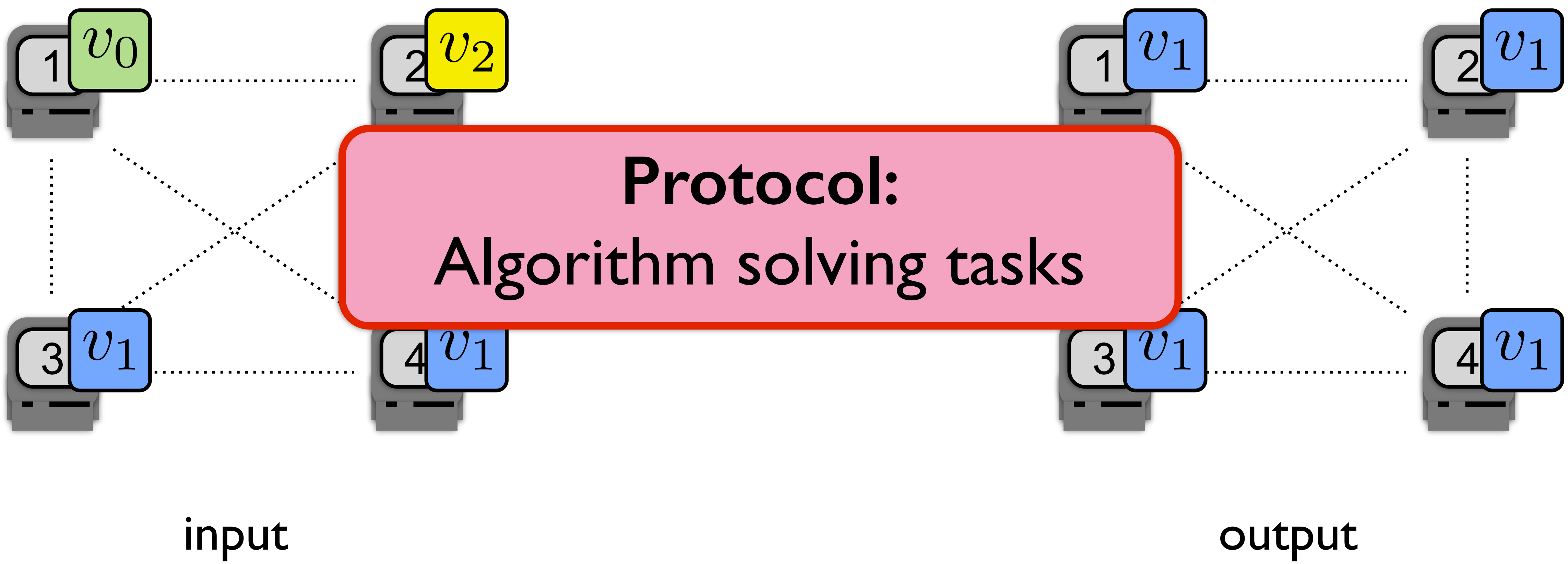
Processes output values a single proposed value

Consensus task

Tasks:

Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$

Processes output values a single proposed value



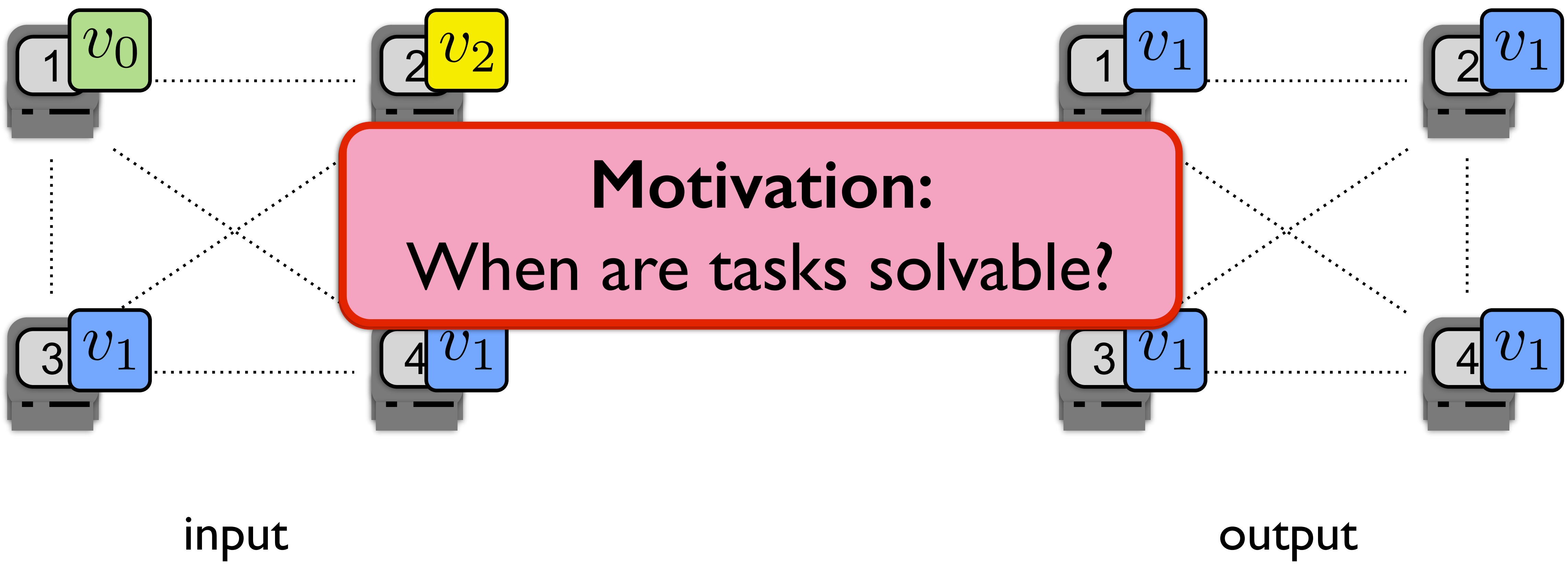
Consensus task

Introduction

Tasks:

Processes input values from a set { v_0 v_1 v_2 v_3 }

Processes output values a single proposed value



Motivation:
When are tasks solvable?

Consensus task

Tasks:

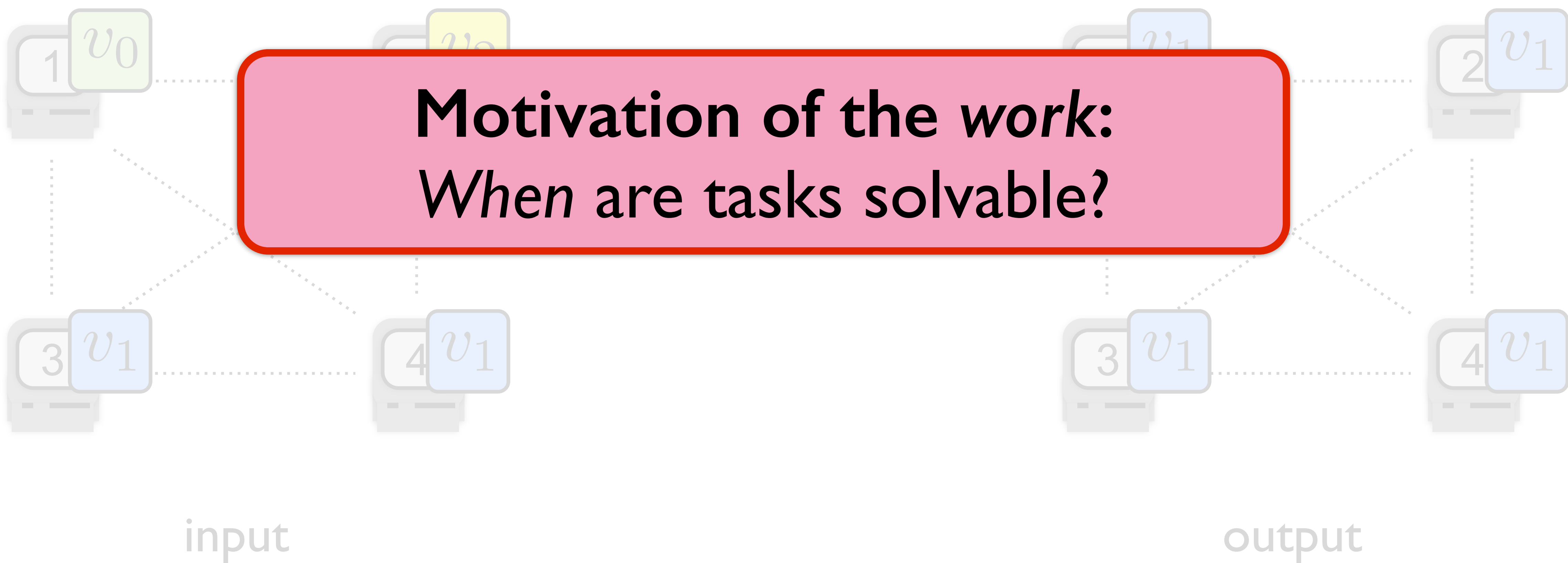
Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$

Processes output values a single proposed value

Tasks:

Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$

Processes output values a single proposed value



Motivation of the *work*:
When are tasks solvable?

Consensus task

Introduction

Tasks:

Processes input values from a set $\{ v_0 \; v_1 \; v_2 \; v_3 \}$

Processes output values a single proposed value

Motivation of the *work*:

When are tasks solvable?

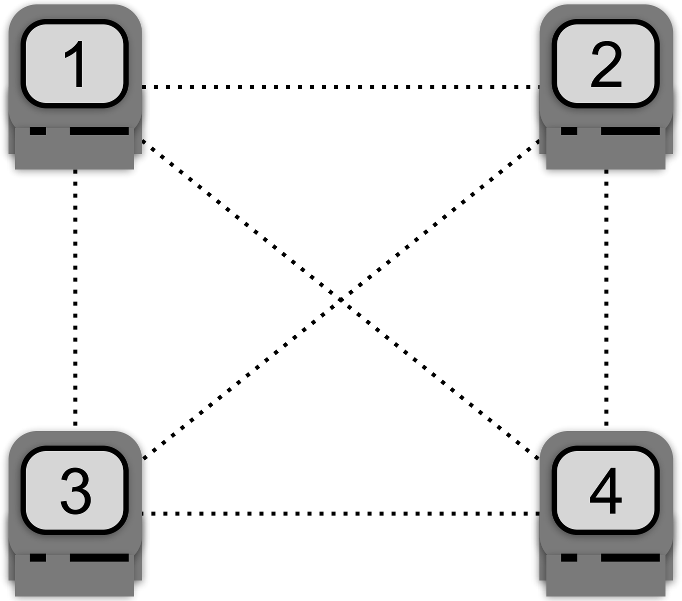
Motivation of the *talk*:

Overview the research area for the AATRN

Create collaboration bridges

Introduction

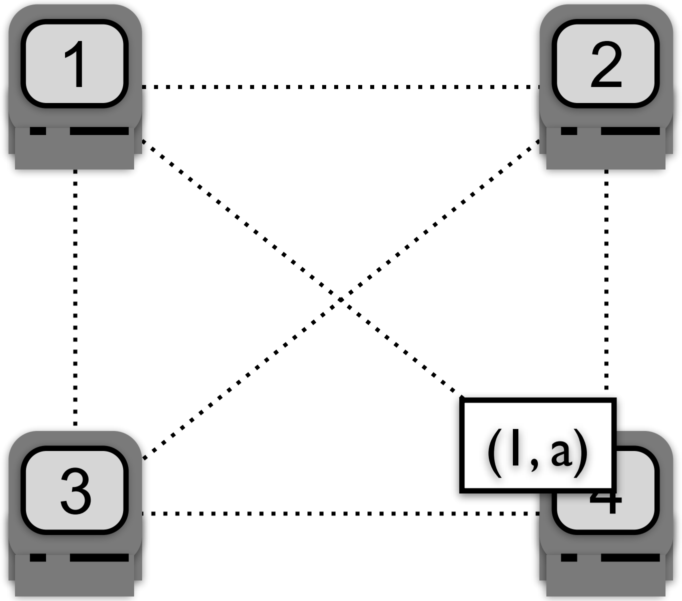
Tasks:



- Message-passing
  - Complete communication graph
  - Senders reliably identified
- FIFO delivery for each pair

Introduction

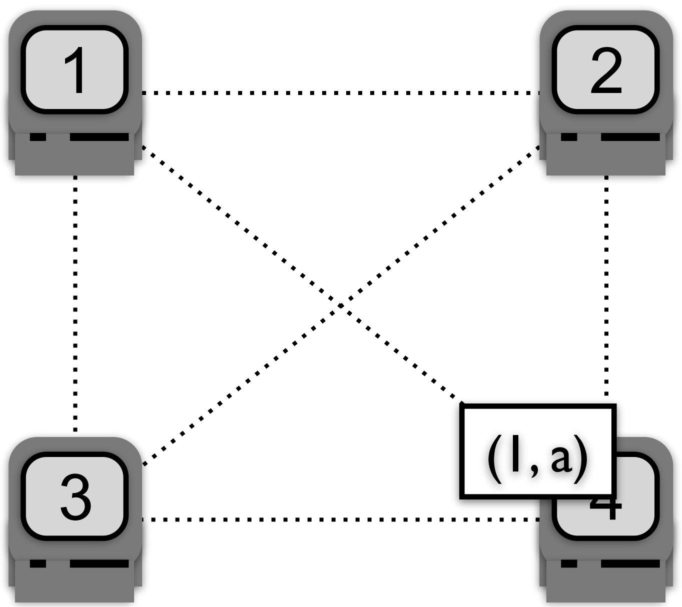
Tasks:



- Message-passing
  - Complete communication graph
  - Senders reliably identified
- FIFO delivery for each pair

Introduction

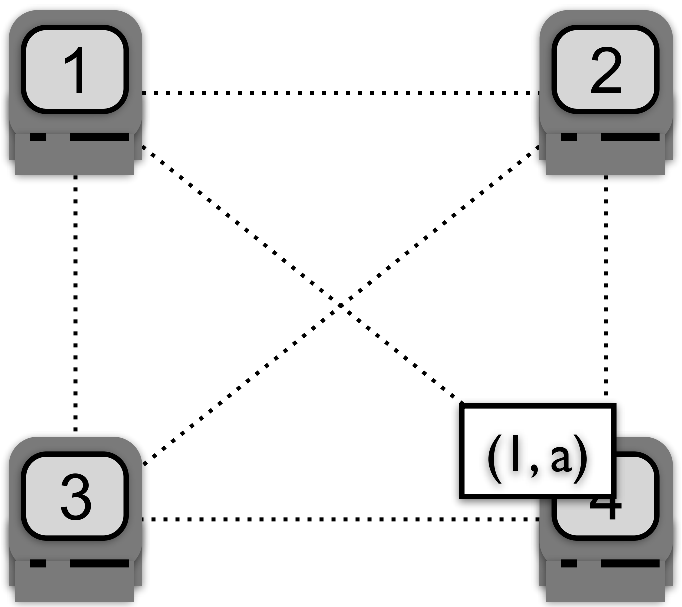
Tasks:



- Message-passing
  - Complete communication graph
  - Senders reliably identified
- FIFO delivery for each pair

Trivial?

Tasks:



- Message-passing
  - Complete communication graph
  - Senders reliably identified
- FIFO delivery for each pair

Failures and (a)synchrony are the difficulties

Failures & Synchrony

Failures & Synchrony

Processes subject to failures

Failures & Synchrony

Processes subject to failures

Crash failures .... halting failures

Failures & Synchrony

Processes subject to failures

- Crash failures .... halting failures
- Byzantine failures .... arbitrary failures

Failures & Synchrony

Processes subject to failures

- Crash failures .... halting failures
- Byzantine failures .... arbitrary failures

Messages subject to delivery semantics

Failures & Synchrony

Processes subject to failures

Crash failures .... halting failures

Byzantine failures .... arbitrary failures

Messages subject to delivery semantics

Synchronous systems .... round tick: messages delivered

Failures & Synchrony

Processes subject to failures

- Crash failures .... halting failures
- Byzantine failures .... arbitrary failures

Messages subject to delivery semantics

- Synchronous systems .... round tick: messages delivered
- Asynchronous systems .... messages delivered eventually

Failures & Synchrony

Processes subject to failures

- Crash failures ······ halting failures
- * Byzantine failures ······ arbitrary failures

Messages subject to delivery semantics

- * Synchronous systems ······ round tick: messages delivered
- * Asynchronous systems ······ messages delivered eventually

Asynchrony + Failures

Asynchronous Systems,
Byzantine Failures

$n = 4, t = 1$

Asynchrony + Failures

of processes

$$n = 4, t = 1$$

Asynchronous Systems,
Byzantine Failures

Asynchrony + Failures

\# of processes

known max # of Byzantine procs

$$n = 4, t = 1$$

Asynchronous Systems, Byzantine Failures

Asynchrony + Failures

of processes

known max # of Byzantine procs

$$n = 4, t = 1$$

Asynchronous Systems, Byzantine Failures

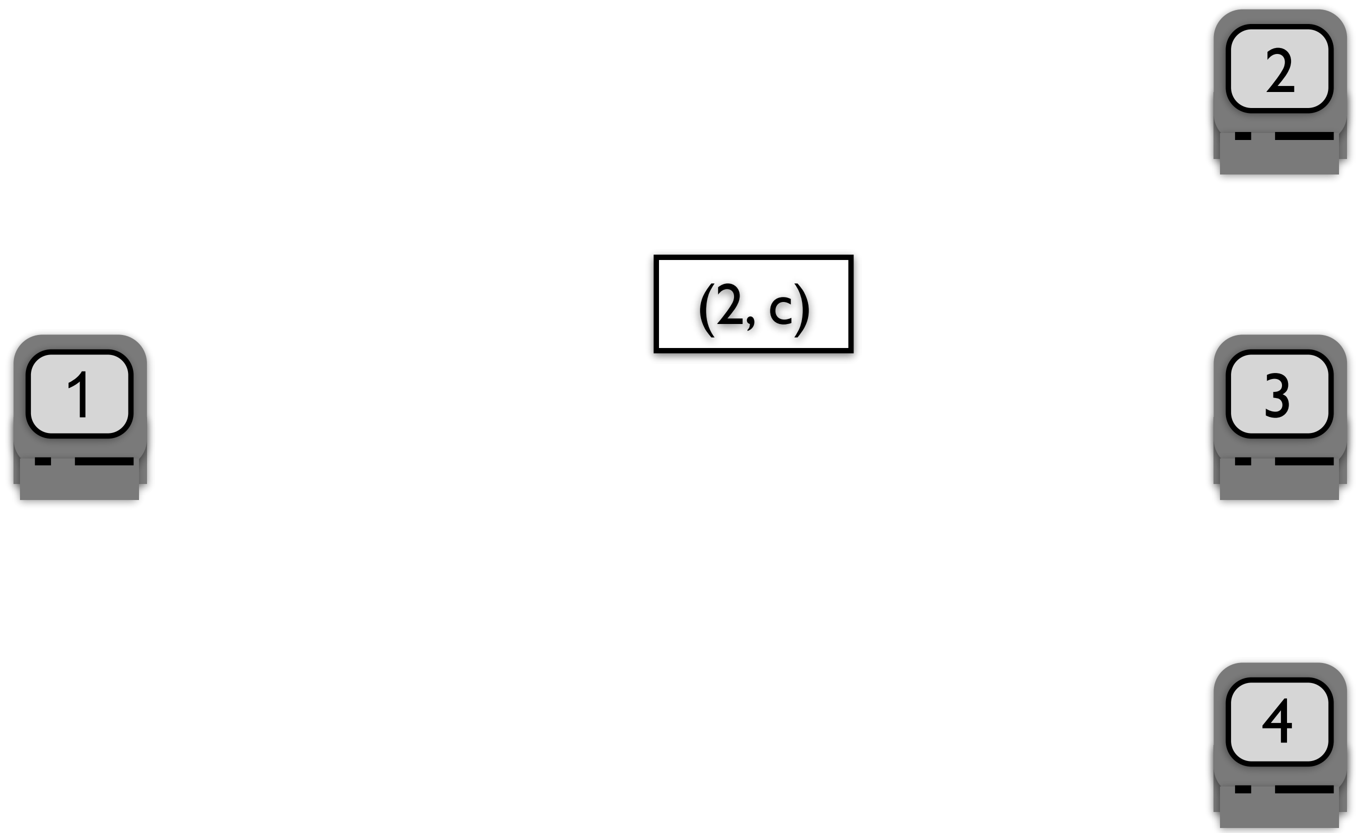
Asynchrony + Failures

of processes

known max # of Byzantine procs

$$n = 4, t = 1$$

Asynchronous Systems, Byzantine Failures



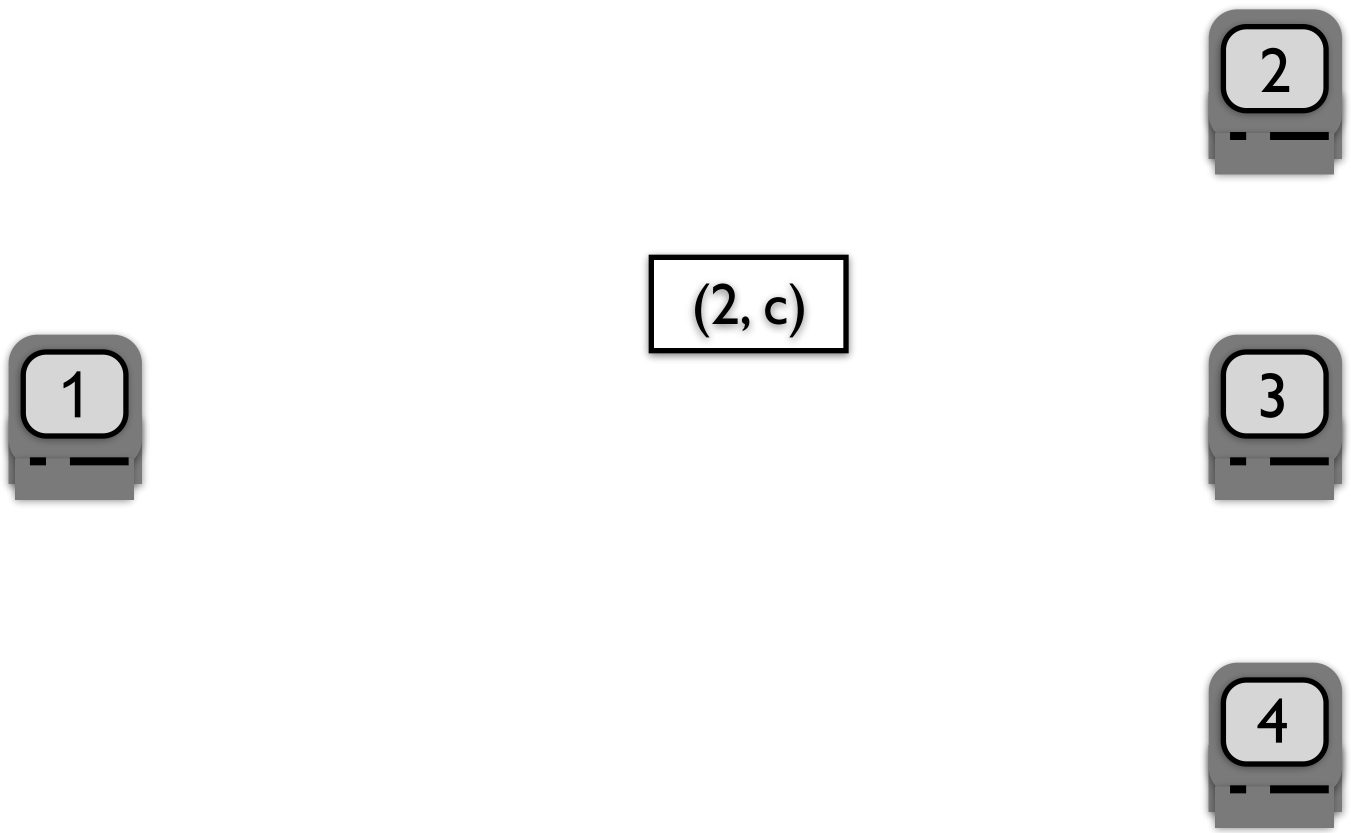
Asynchrony + Failures

of processes

known max # of Byzantine procs

$$n = 4, t = 1$$

Asynchronous Systems, Byzantine Failures



Don’t know when it is delivered

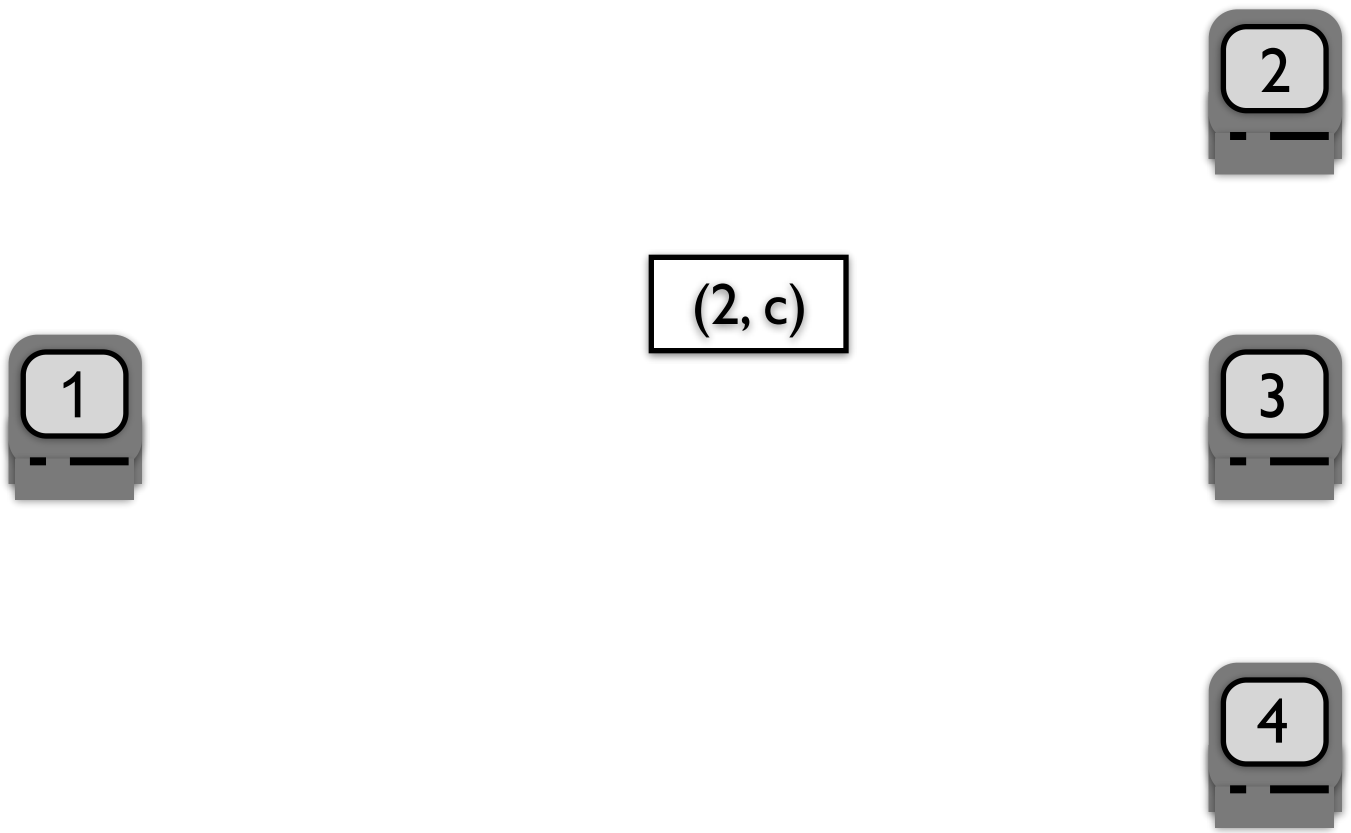
Asynchrony + Failures

of processes

known max # of Byzantine procs

$$n = 4, t = 1$$

Asynchronous Systems, Byzantine Failures



Don't know when it is delivered

When it does, the sender is correct

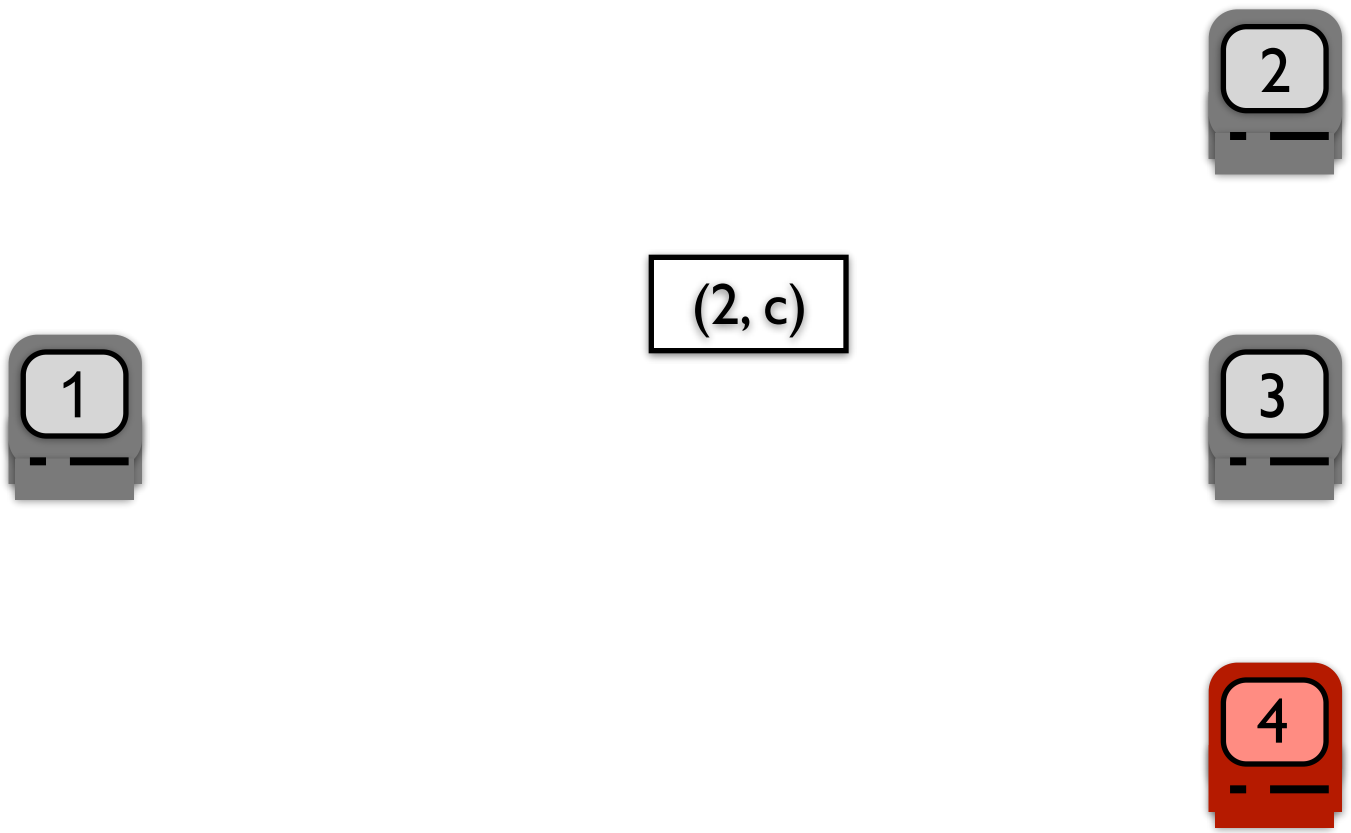
Asynchrony + Failures

of processes

known max # of Byzantine procs

$$n = 4, t = 1$$

Asynchronous Systems, Byzantine Failures



Don't know when it is delivered

When it does, the sender is correct

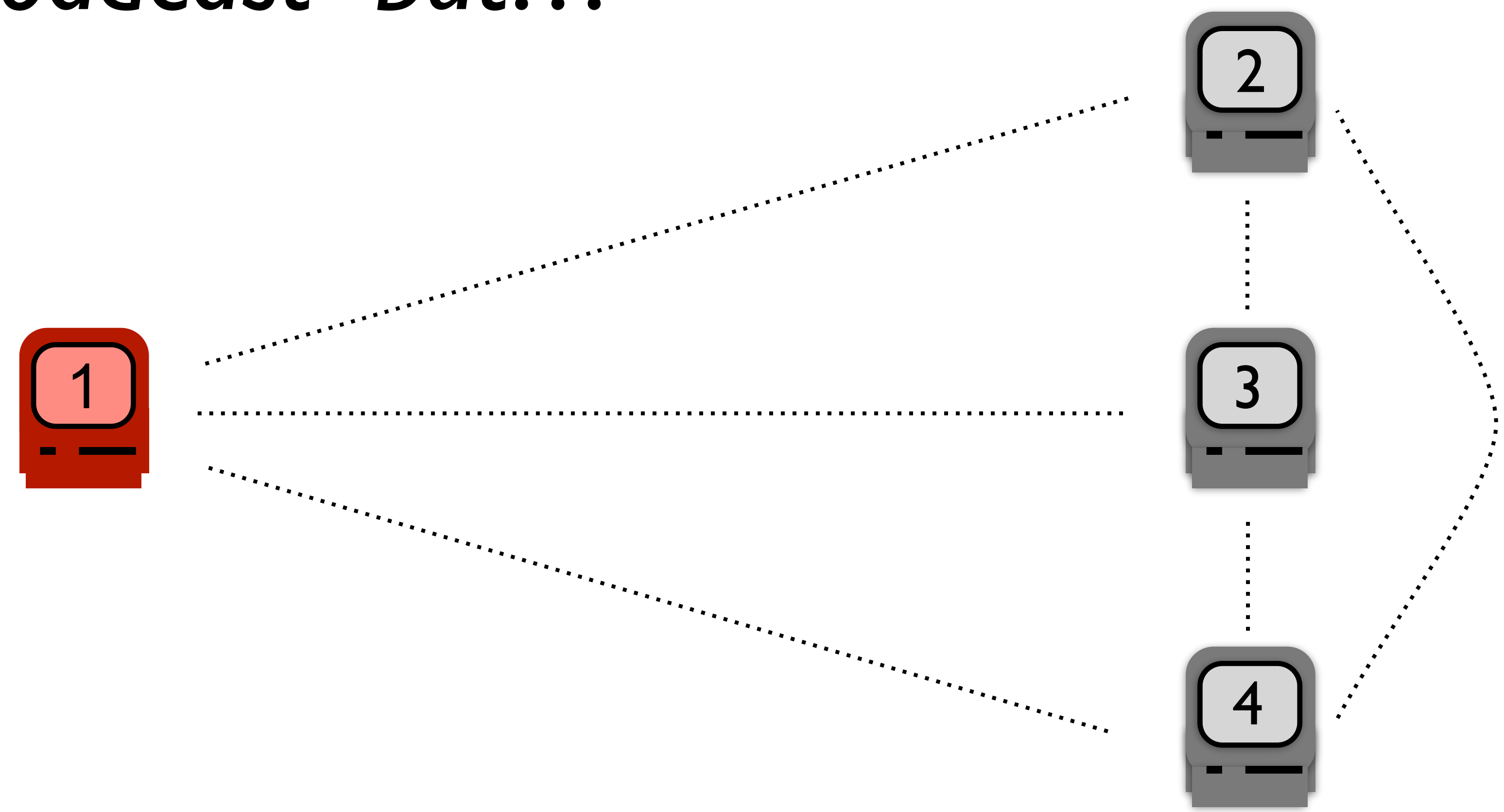
Reliable Broadcast

The Equivocation Issue

Call it "broadcast" but…

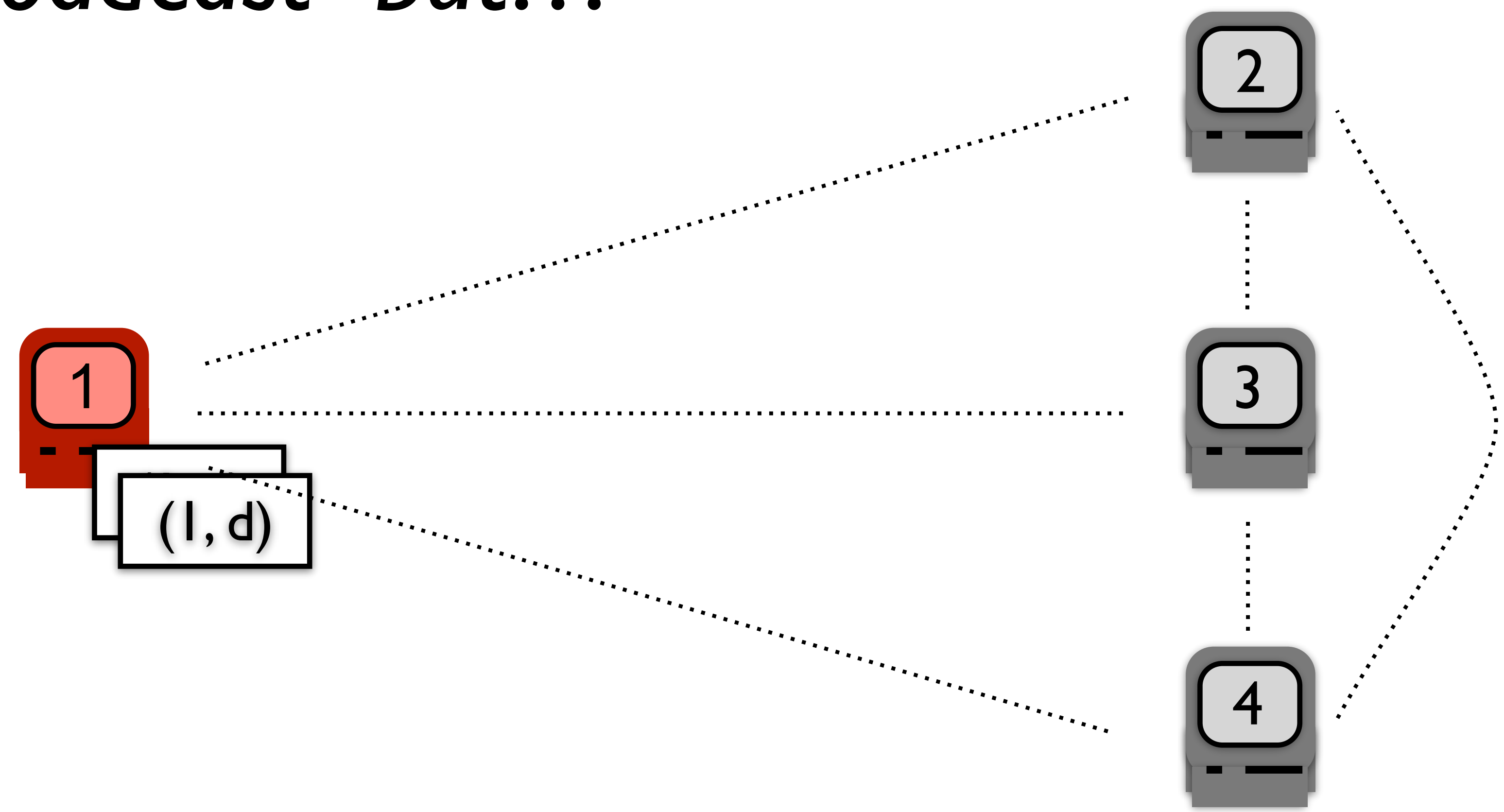
The Equivocation Issue

Call it "broadcast" but…



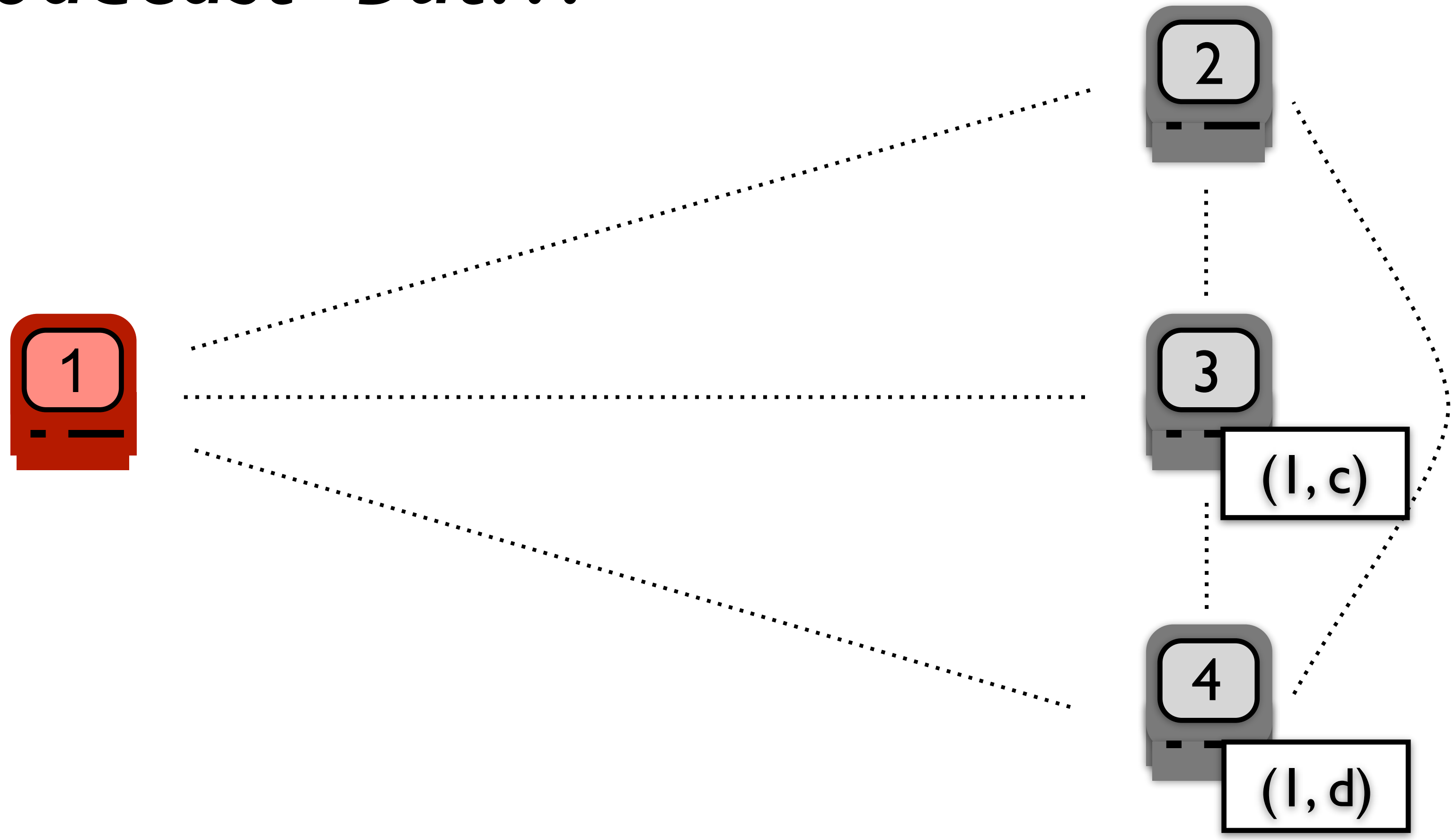
The Equivocation Issue

Call it "broadcast" but…



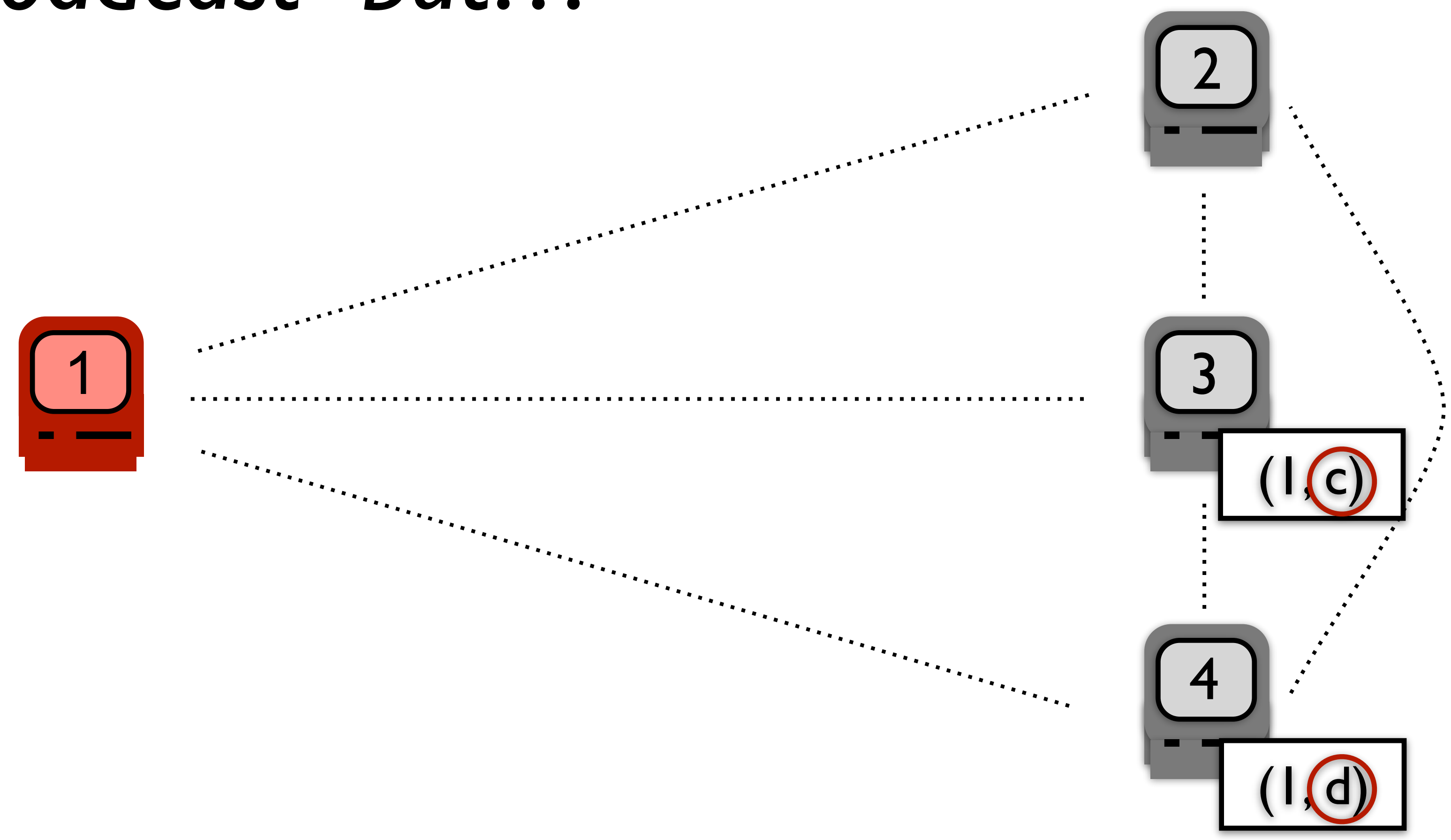
The Equivocation Issue

Call it "broadcast" but…



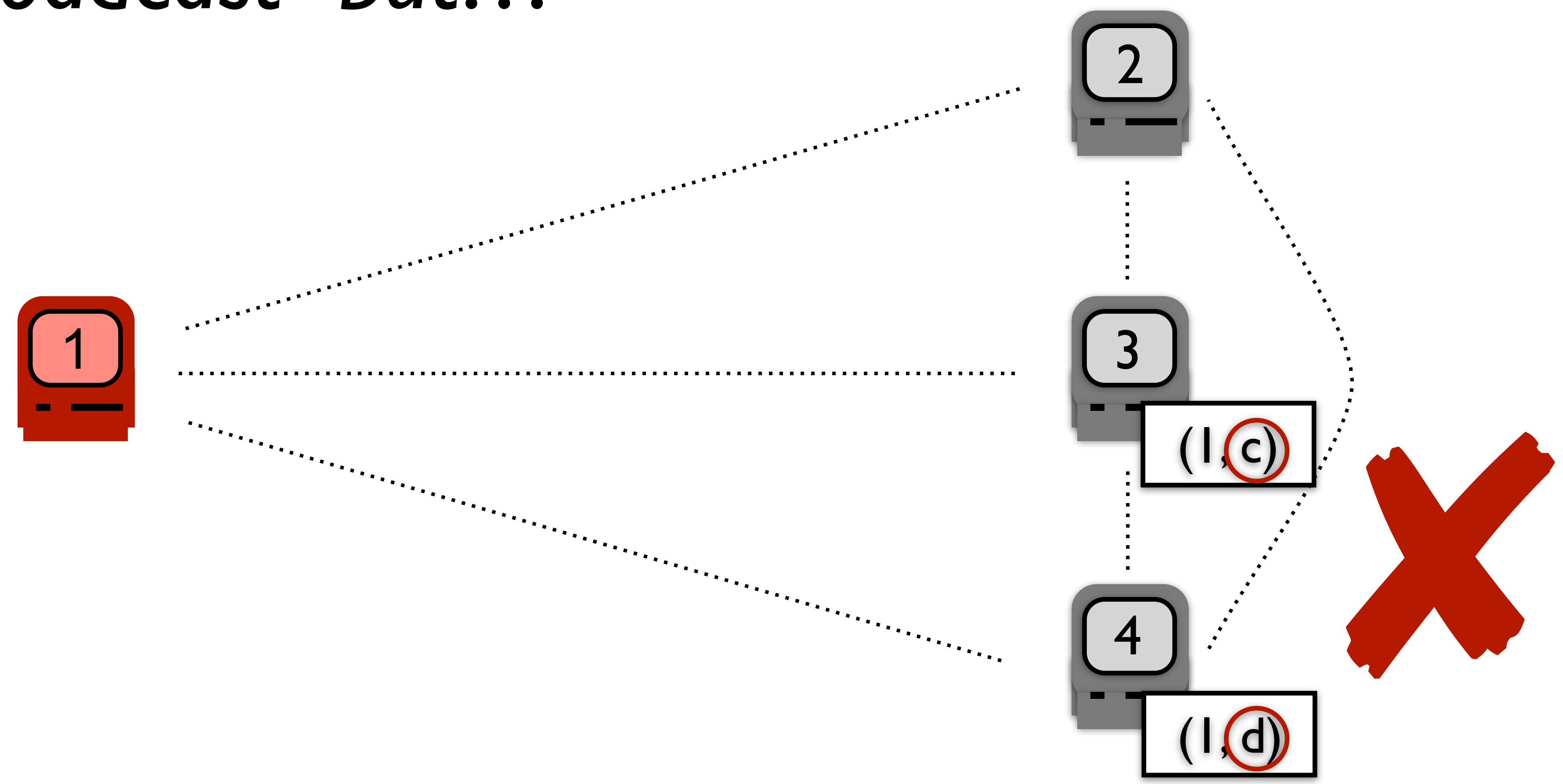
The Equivocation Issue

Call it "broadcast" but…



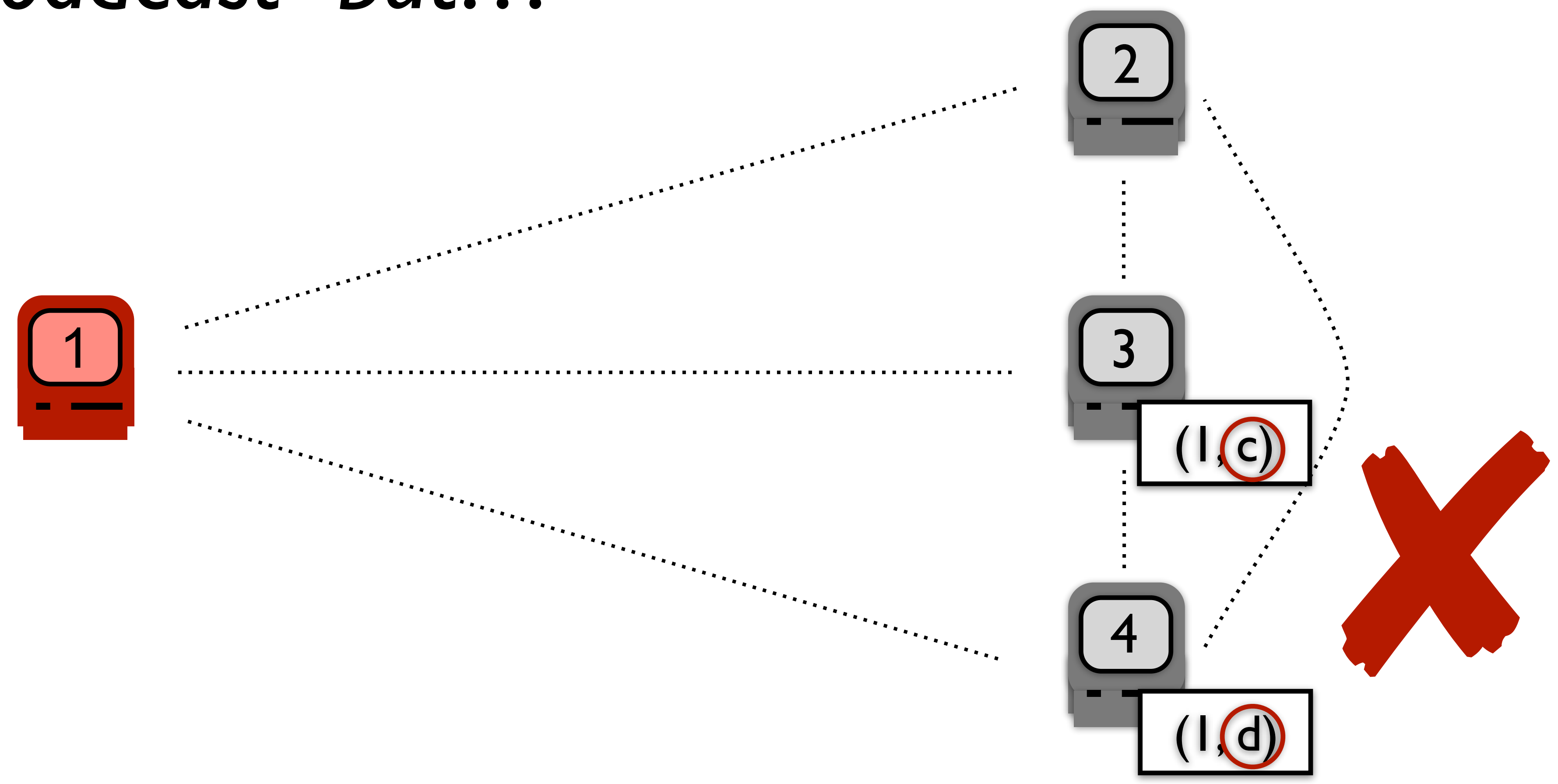
The Equivocation Issue

Call it "broadcast" but…



The Equivocation Issue

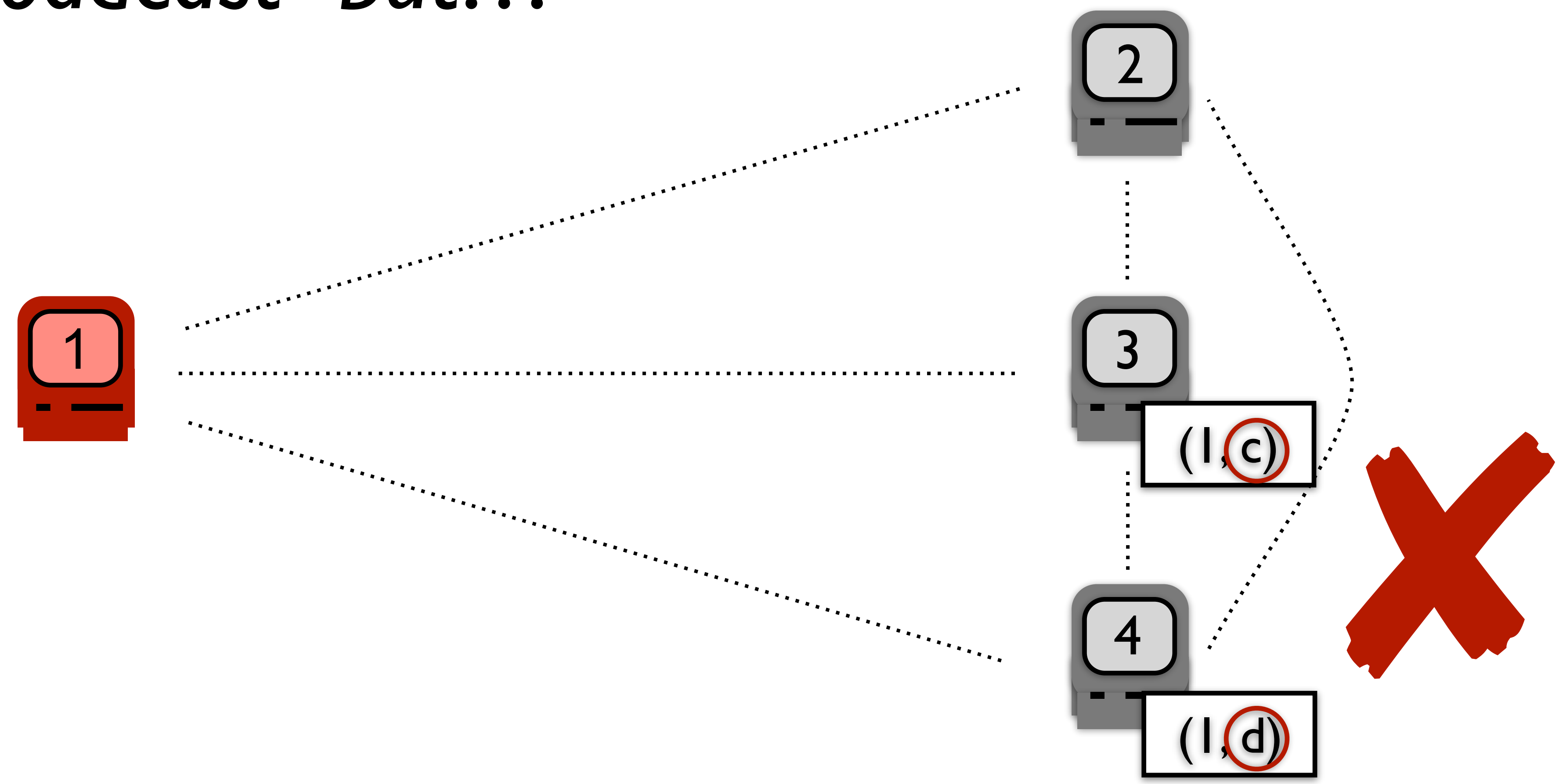
Call it "broadcast" but…



Using a primitive called Reliable Broadcast

The Equivocation Issue

Call it "broadcast" but…



Using a primitive called Reliable Broadcast

Message contents are consistent

Full-Information Protocols

Full-Information Protocols

Protocol = distributed algorithm

$s \leftarrow I_i$
for $r : 1 \rightarrow R$ **do**
    Send s via reliable broadcast
    while less than $(n+1) - t$ messages **do**
        Receive an r-round message M
        $s \leftarrow s \cup \{M\}$
return $\delta(s)$

Full-Information Protocols

Protocol = distributed algorithm

initial state: input

$s \leftarrow I_i$
for $r : 1 \rightarrow R$ **do**
Send s via reliable broadcast
while less than $(n+1) - t$ messages **do**
Receive an r-round message M
$s \leftarrow s \cup \{M\}$
return $\delta(s)$

Full-Information Protocols

Protocol = distributed algorithm

initial state: input

$s \leftarrow I_i$
for $r : 1 \rightarrow R$ **do**
Send s via reliable broadcast
while less than $(n+1) - t$ messages **do**
Receive an r-round message M
$s \leftarrow s \cup \{M\}$
return $\delta(s)$

Full-Information Protocols

Protocol = distributed algorithm

initial state: input

$s \leftarrow I_i$
for $r : 1 \rightarrow R$ **do**
  Send s via reliable broadcast
  while less than $(n+1) - t$ messages **do**
    Receive an r-round message M
    $s \leftarrow s \cup \{M\}$
return $\delta(s)$

Full-Information Protocols

Protocol = distributed algorithm

initial state:
input

$s \leftarrow I_i$
for $r : 1 \rightarrow R$ **do**
  Send s via reliable broadcast
  while less than $(n+1) - t$ messages **do**
    Receive an r-round message M
    $s \leftarrow s \cup \{M\}$
return $\delta(s)$

wait as much
as you can…

Full-Information Protocols

Protocol = distributed algorithm

initial state: input

exchange states

$s \leftarrow I_i$
for $r : 1 \rightarrow R$ **do**
  Send s via reliable broadcast
  while less than $(n+1) - t$ messages **do**
    Receive an r-round message M
    $s \leftarrow s \cup \{M\}$
return $\delta(s)$

wait as much as you can…

Full-Information Protocols

Protocol = distributed algorithm

initial state: input

exchange states

decide

$s \leftarrow I_i$
for $r : 1 \rightarrow R$ **do**
Send s via reliable broadcast
while less than $(n+1)-t$ messages **do**
Receive an r-round message M
$s \leftarrow s \cup \{M\}$
return $\delta(s)$

wait as much as you can…

Full-Information Protocols

Protocol = distributed algorithm

initial state: input

exchange states

decide

$s \leftarrow I_i$
for $r : 1 \rightarrow R$ **do**
  Send s via reliable broadcast
  while less than $(n+1) - t$ messages **do**
    Receive an r-round message M
    $s \leftarrow s \cup \{M\}$
return $\delta(s)$

wait as much as you can…

Modeling Tasks

Tasks and Simplicial Complexes

Tasks modeled as simplicial complexes

Tasks and Simplicial Complexes

Tasks modeled as simplicial complexes

Solvability in terms of topological properties

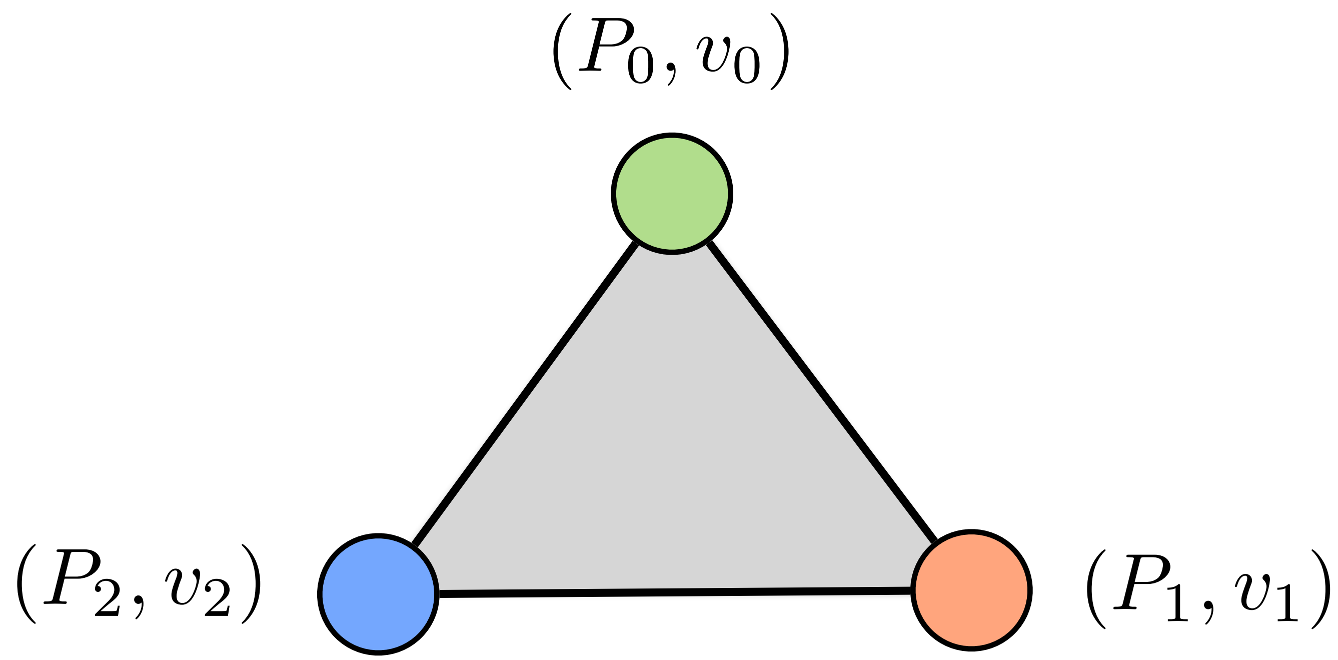
Tasks and Simplicial Complexes

Tasks modeled as simplicial complexes

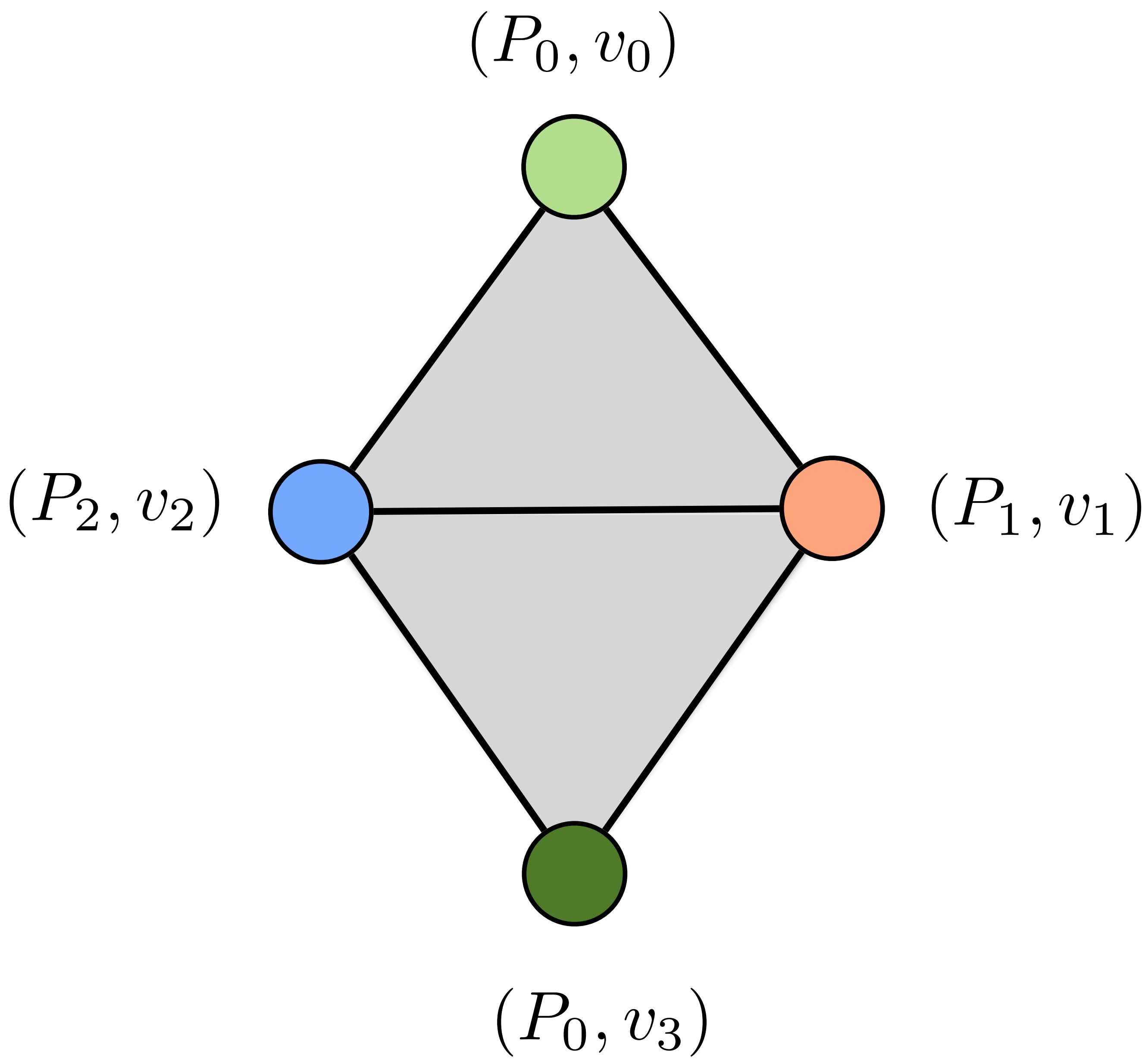
Solvability in terms of topological properties

Let's start with crash failures

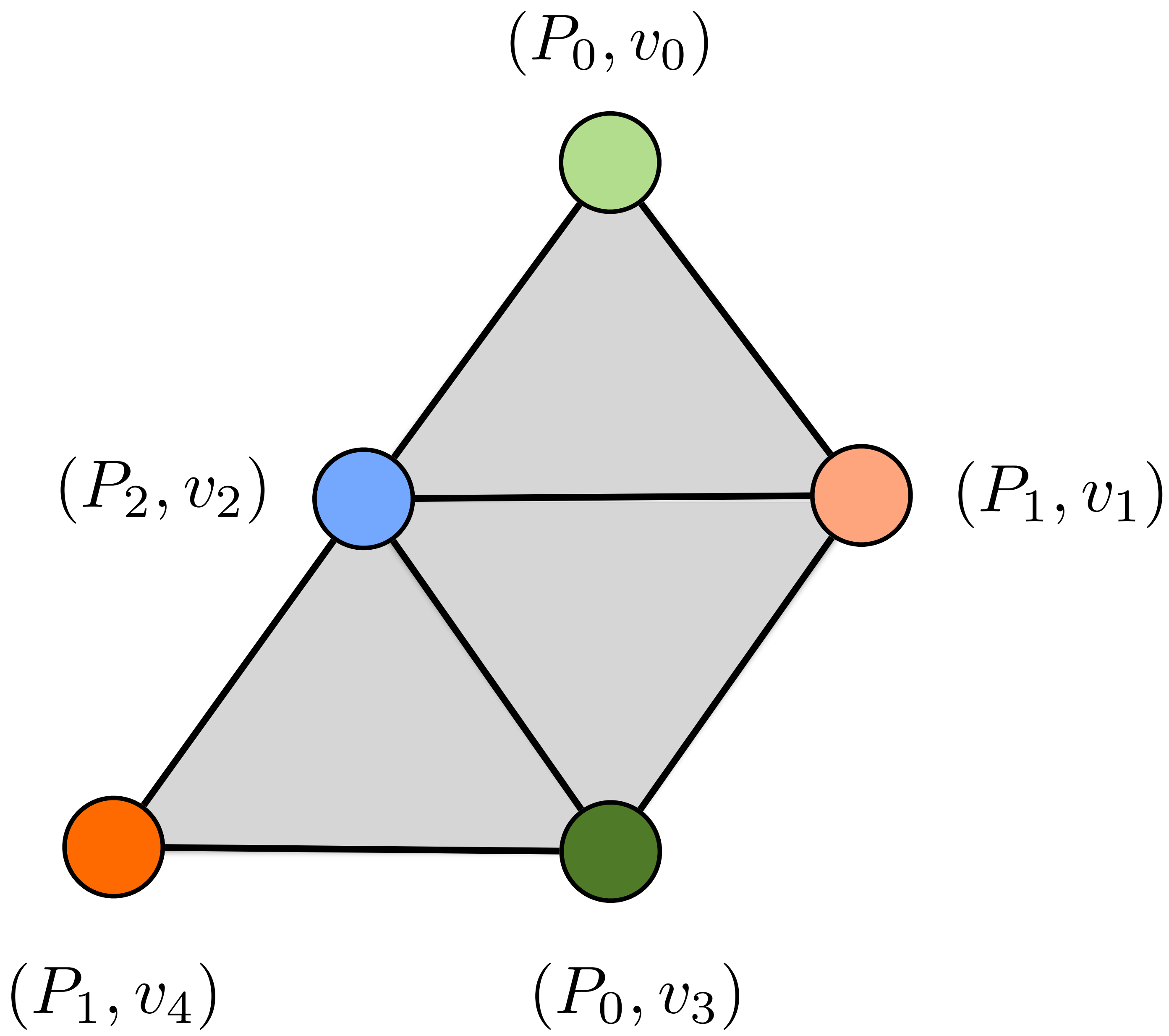
Many Simplexes → Simplicial Complex



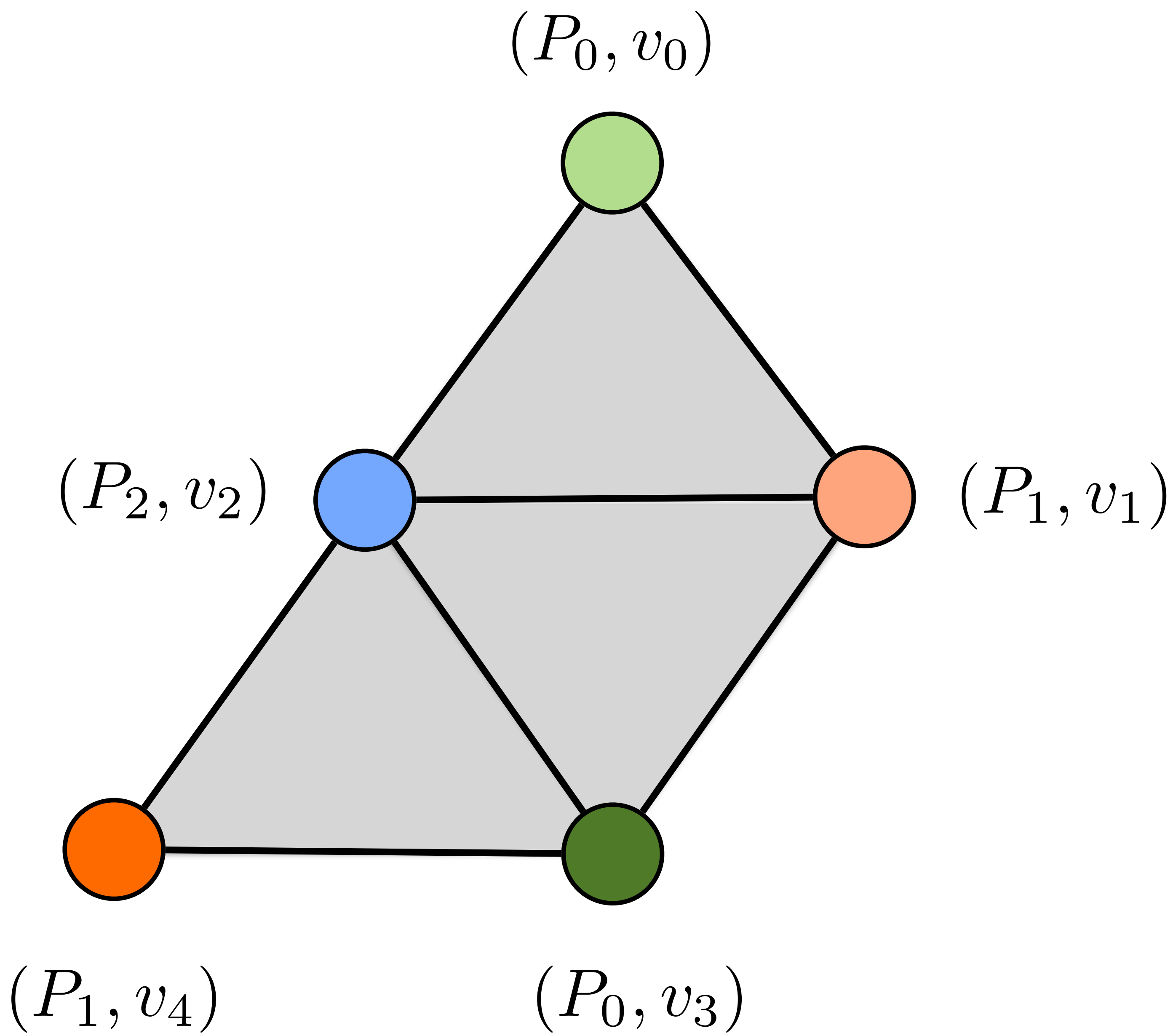
Many Simplexes → Simplicial Complex



Many Simplexes → Simplicial Complex



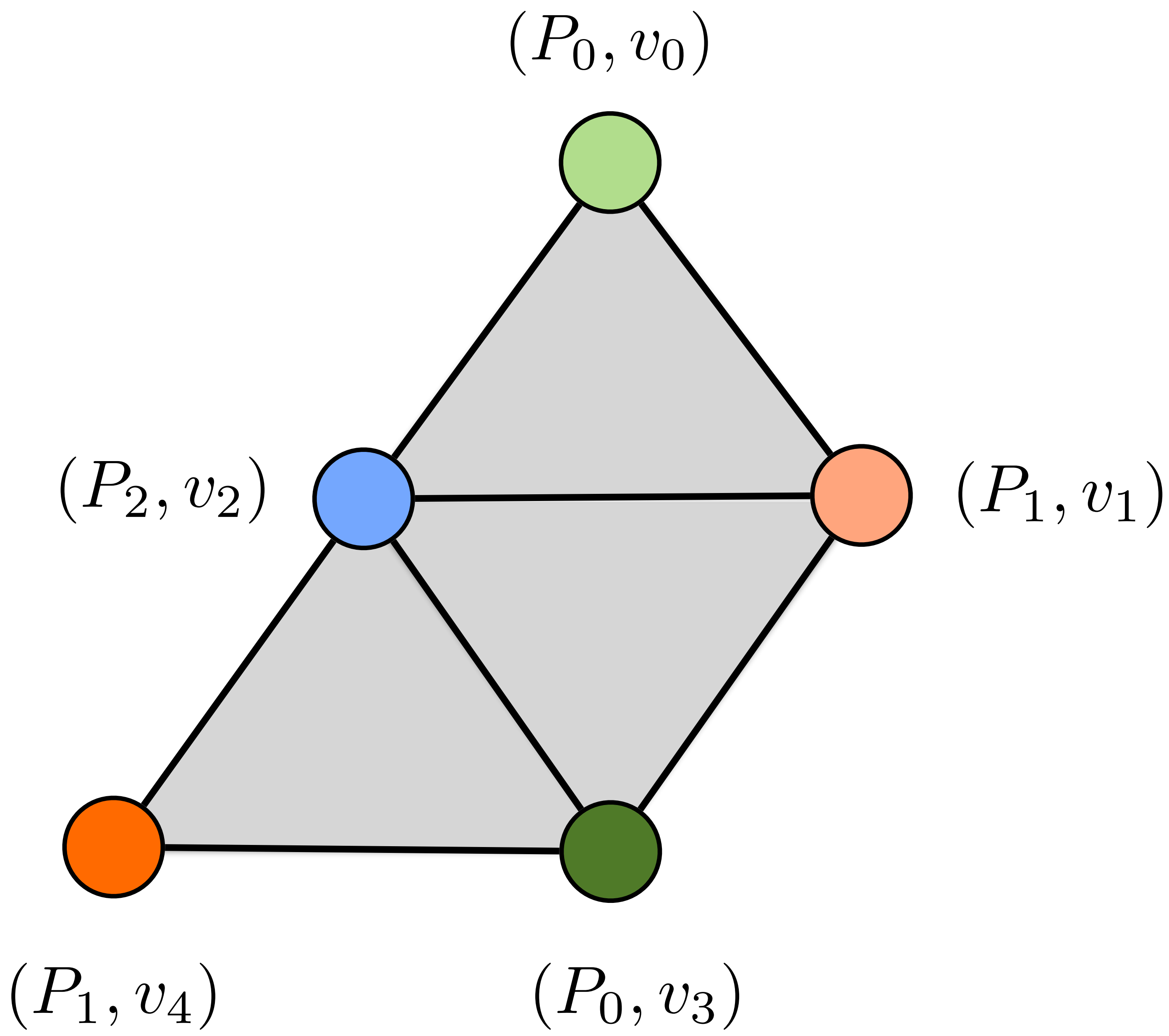
Many Simplexes → Simplicial Complex

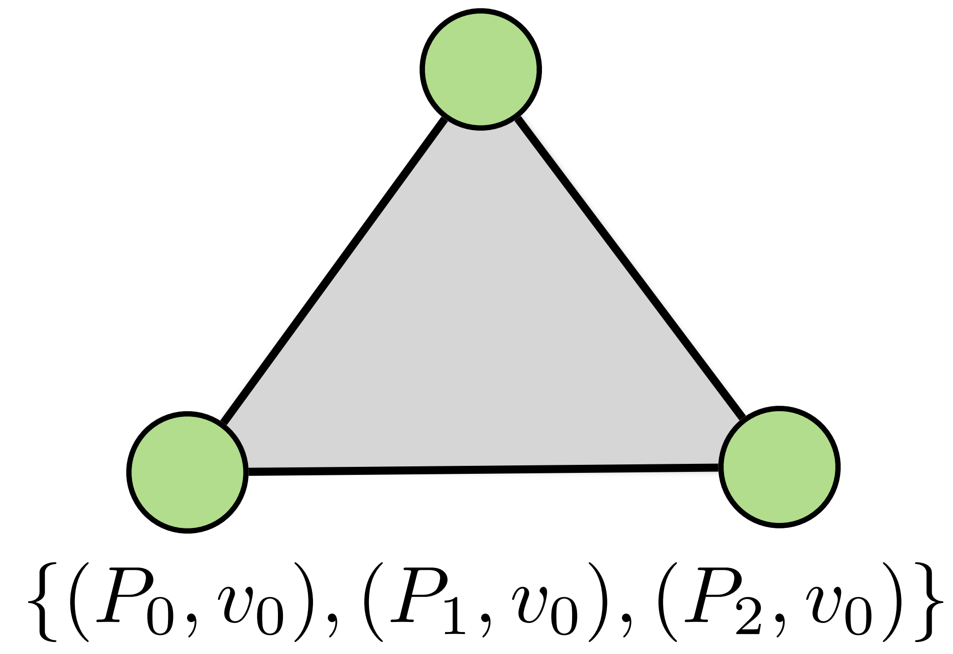


...

$\mathcal{I}$ (input complex)

Many Simplexes → Simplicial Complex

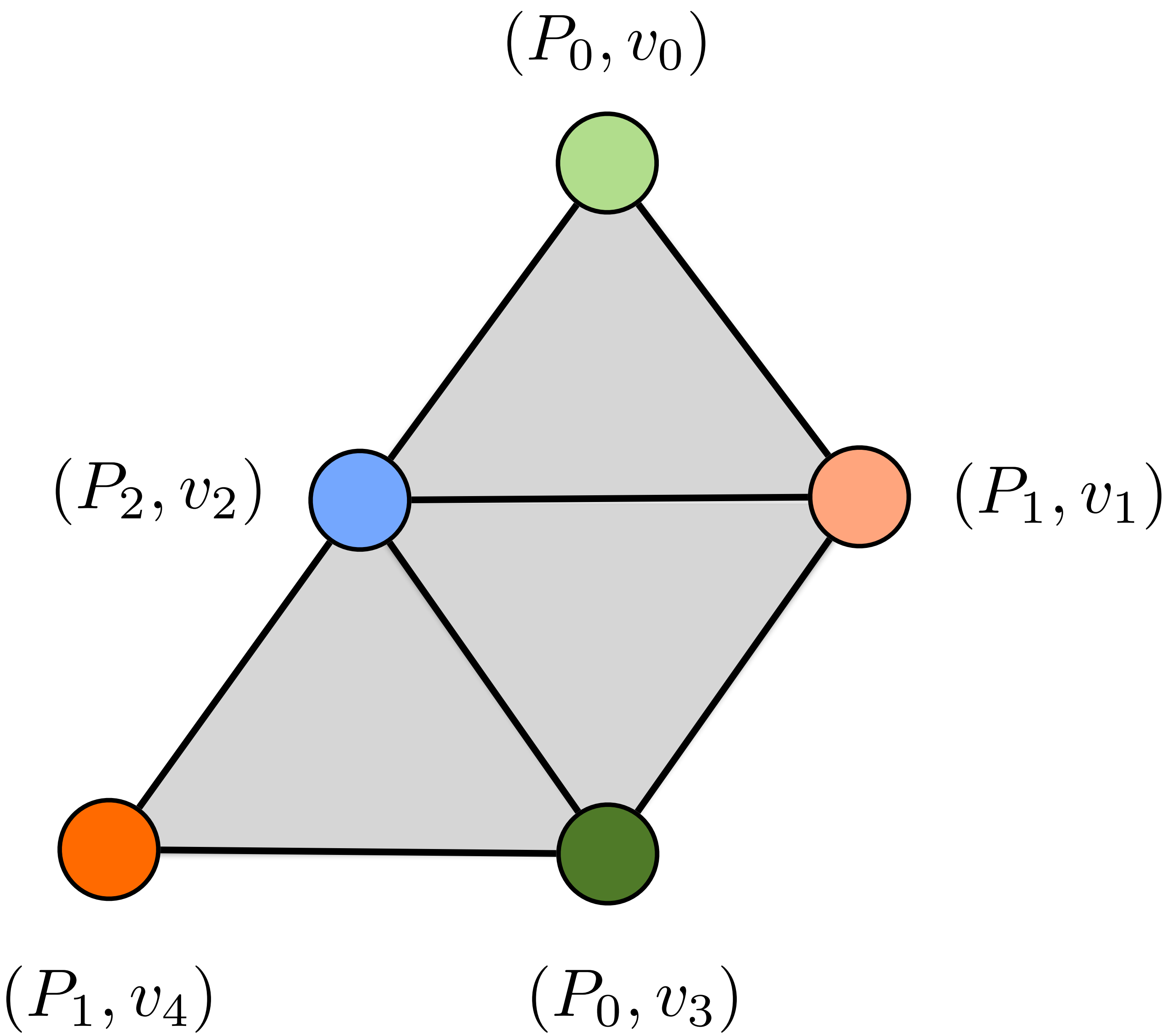


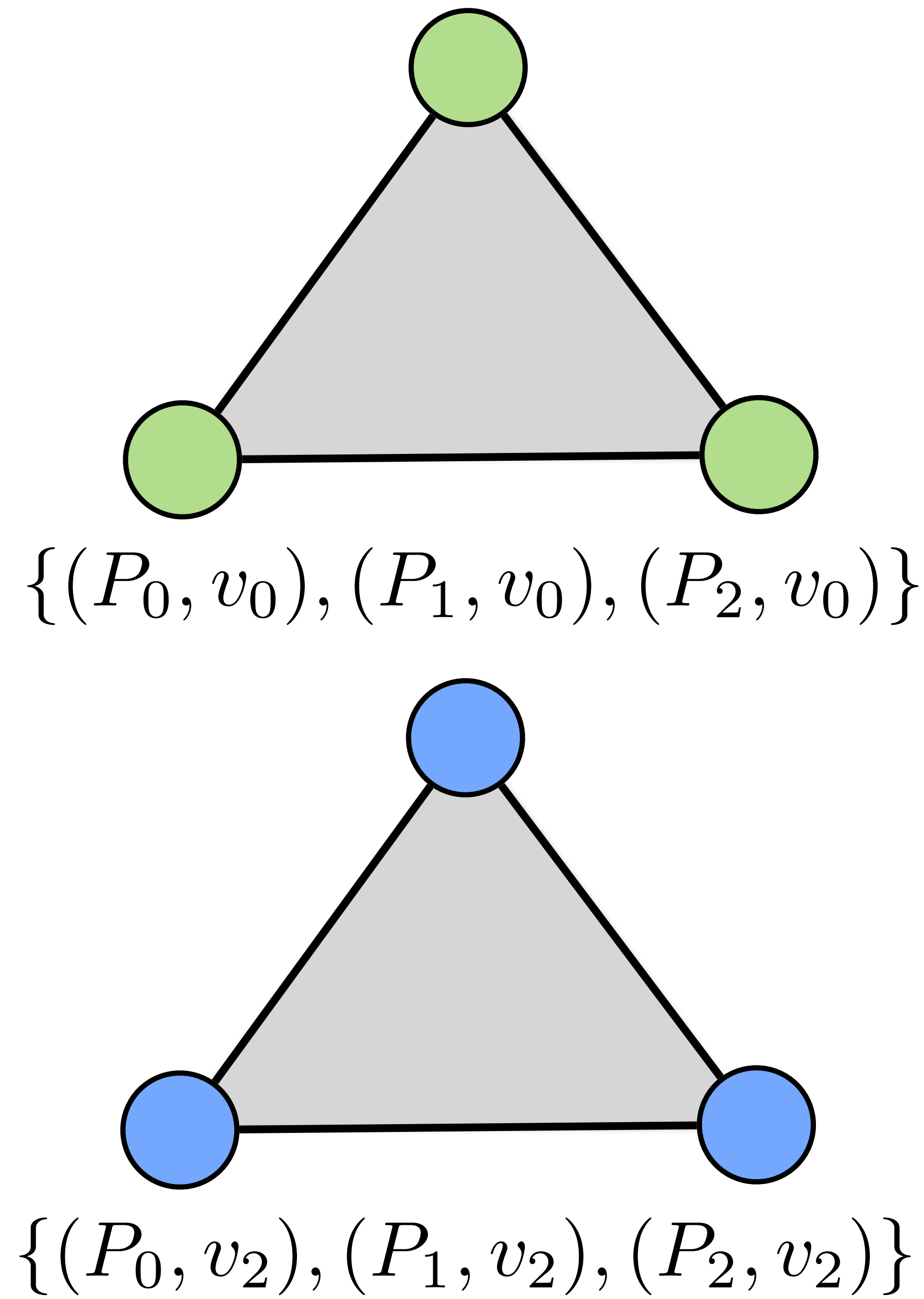


...

$\mathcal{I}$ (input complex)

Many Simplexes → Simplicial Complex

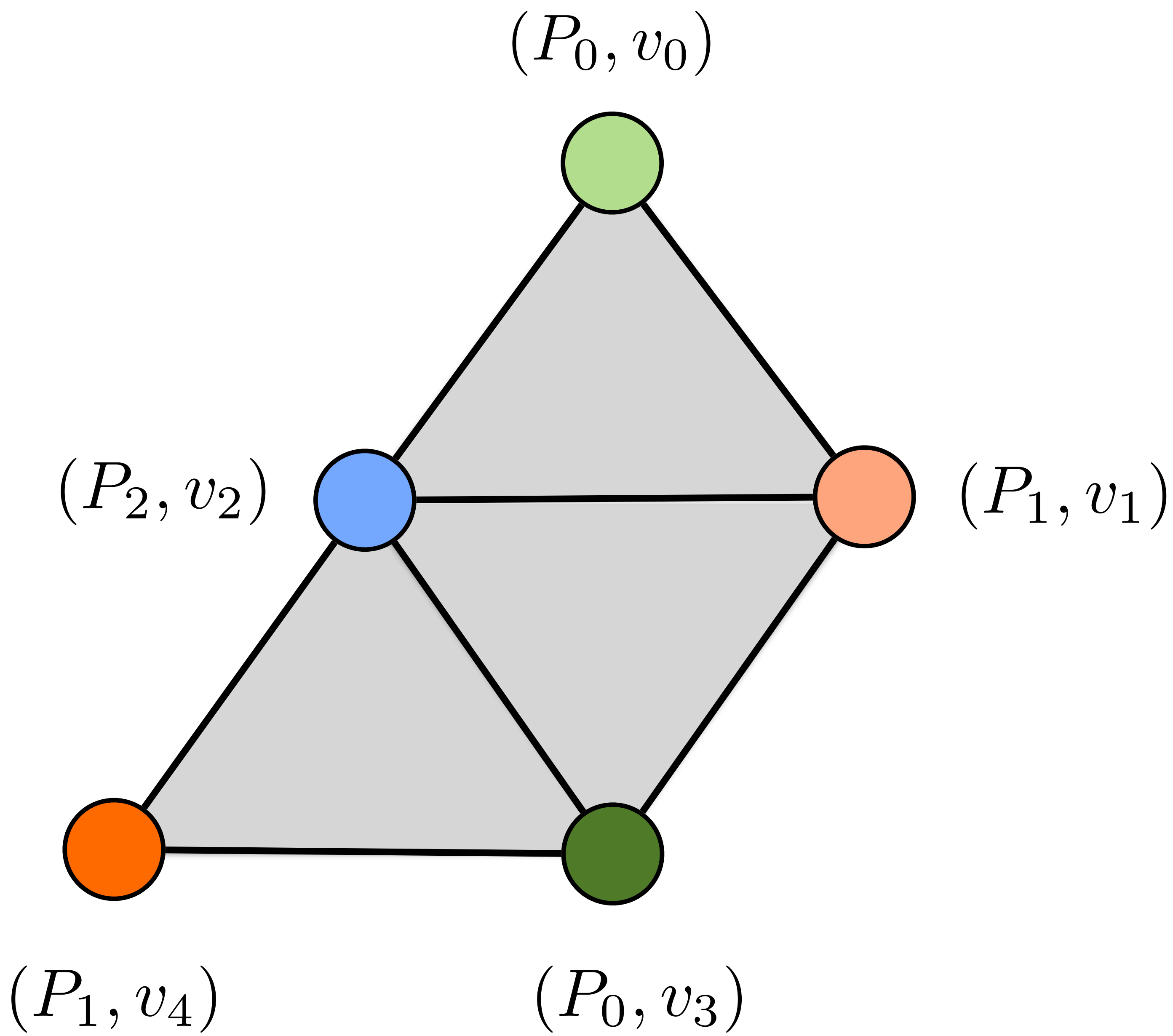




...

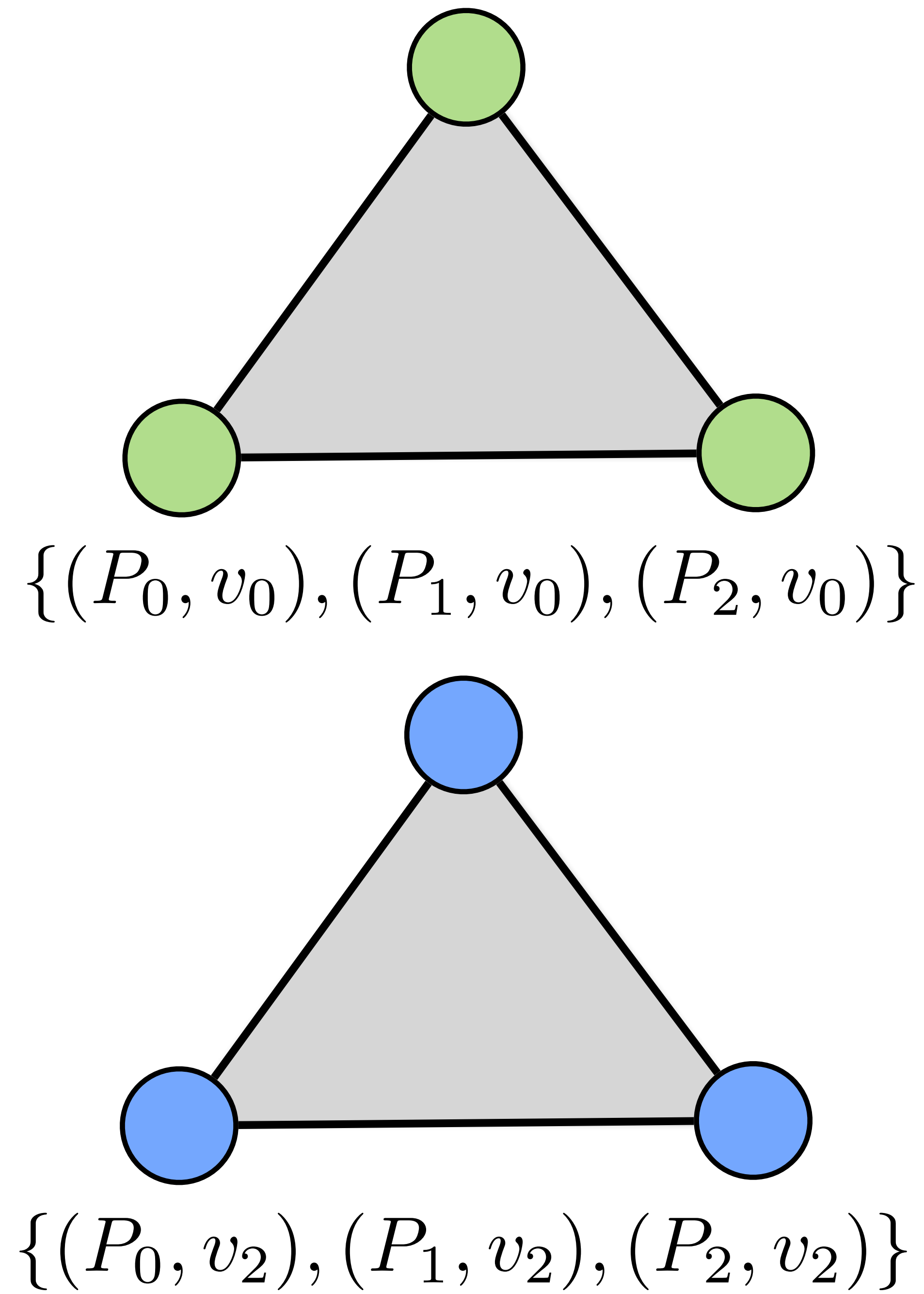
$\mathcal{I}$ (input complex)

Many Simplexes → Simplicial Complex



...

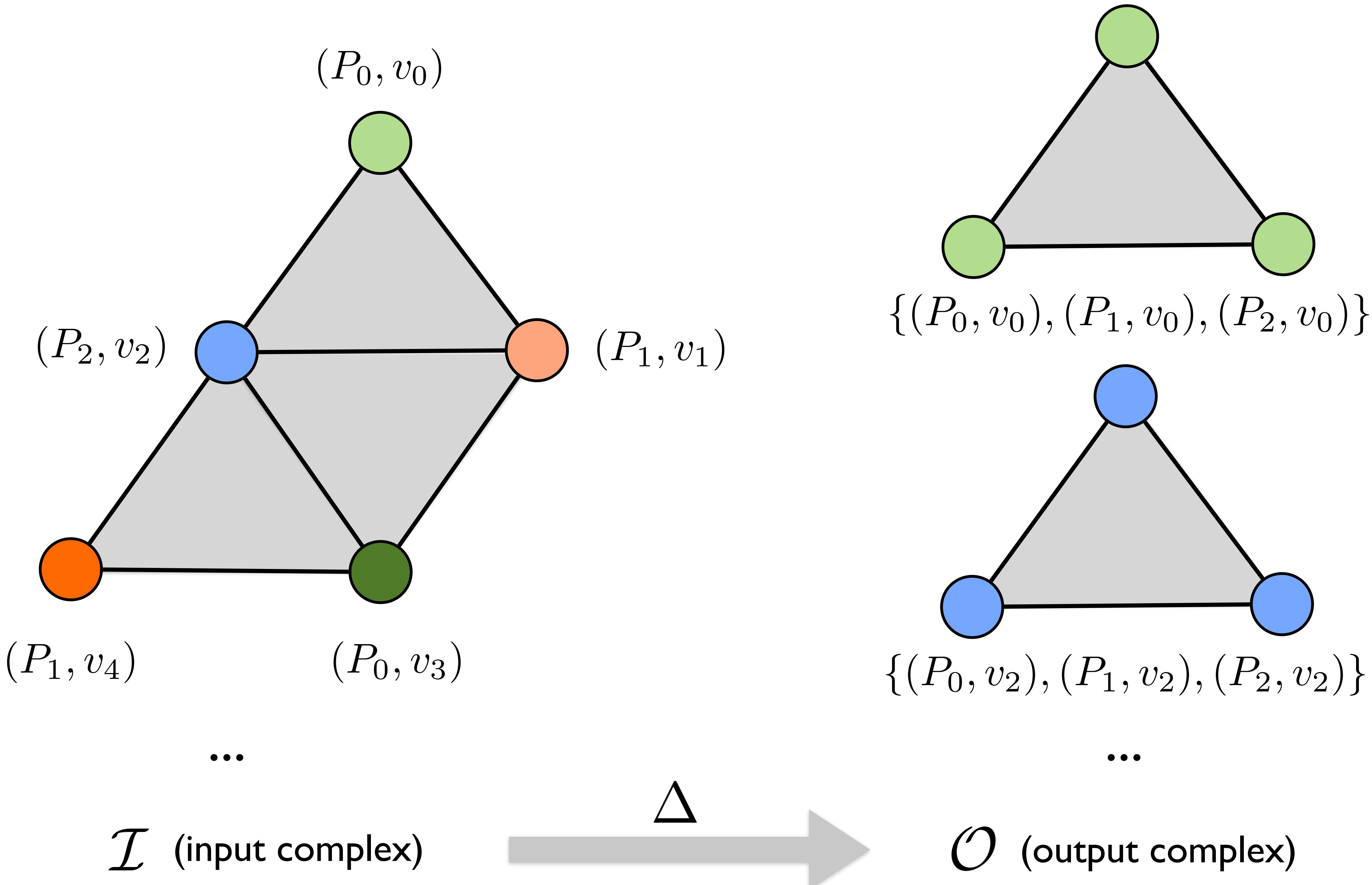
$\mathcal{I}$ (input complex)



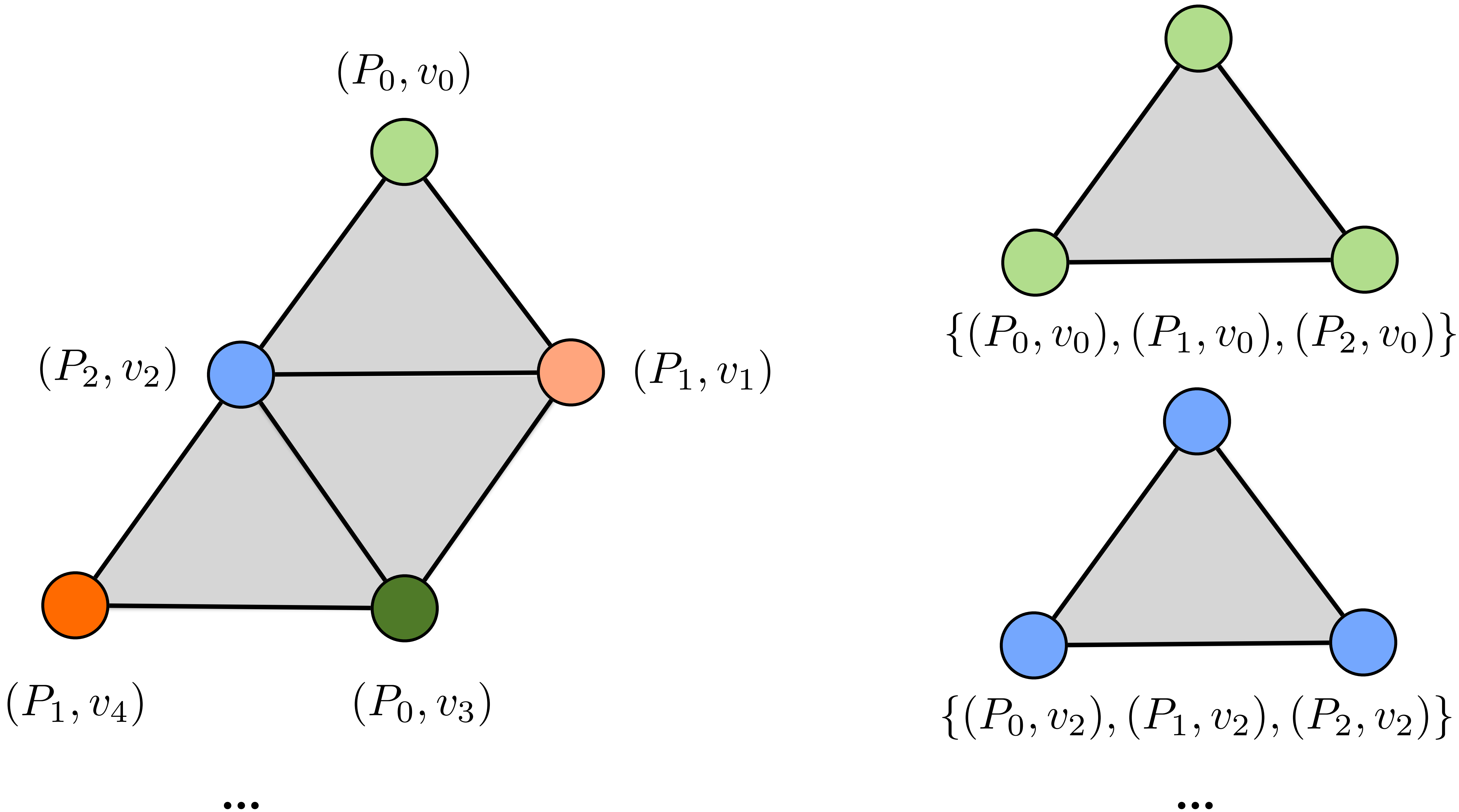
...

$\mathcal{O}$ (output complex)

Many Simplexes → Simplicial Complex



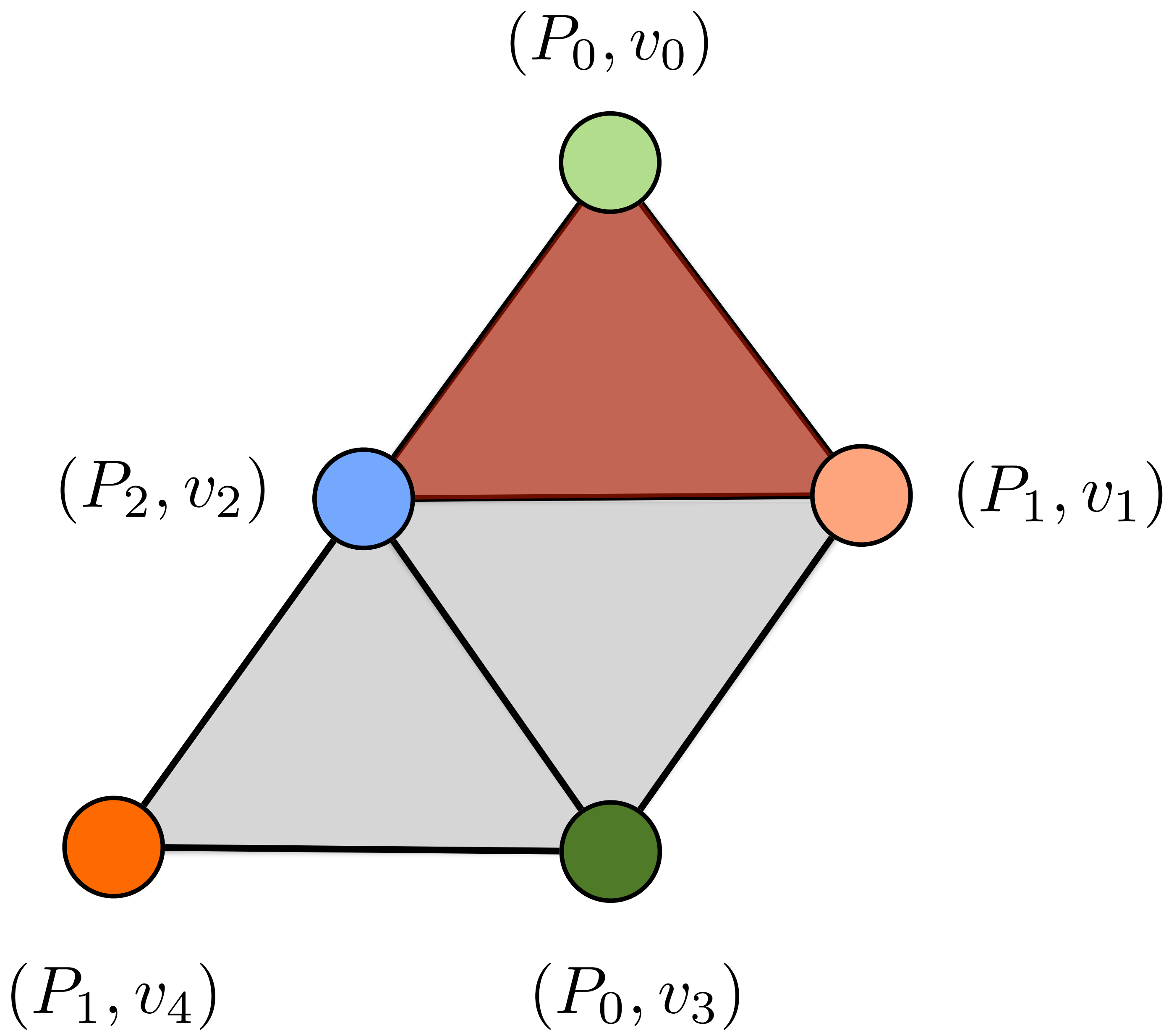
The map Δ

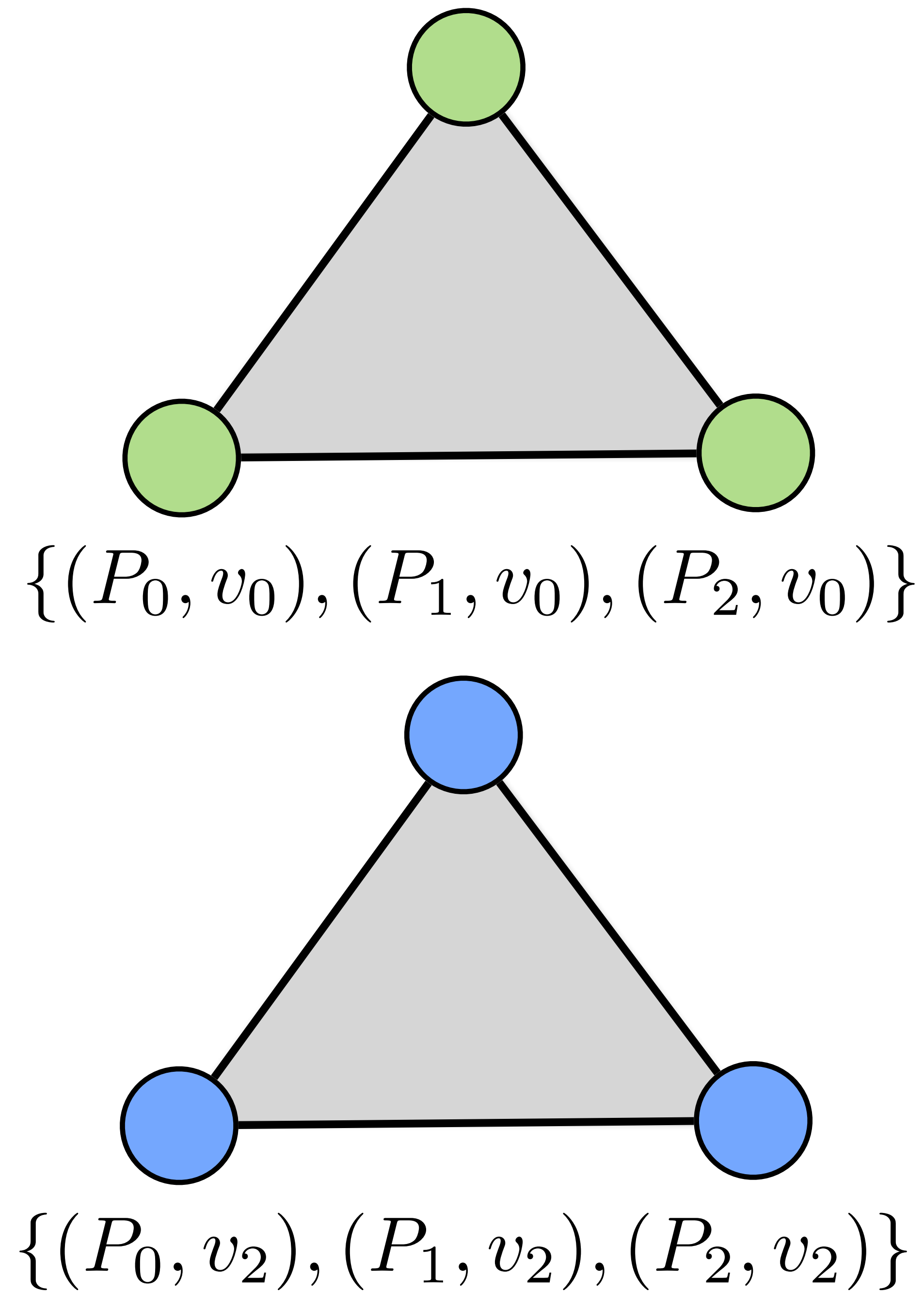


$\mathcal{I}$ (input complex)

$\mathcal{O}$ (output complex)

The map Δ





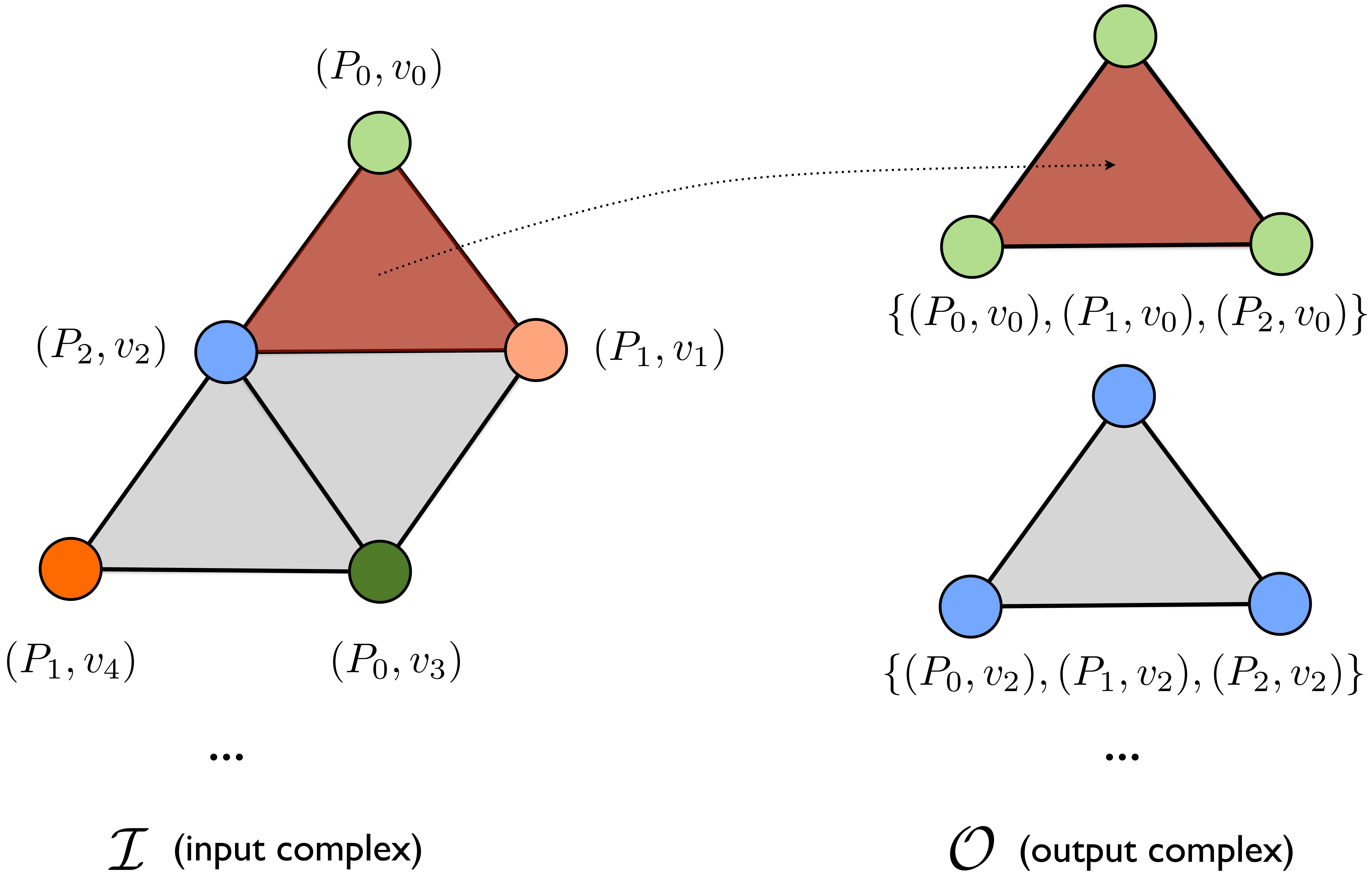
...

...

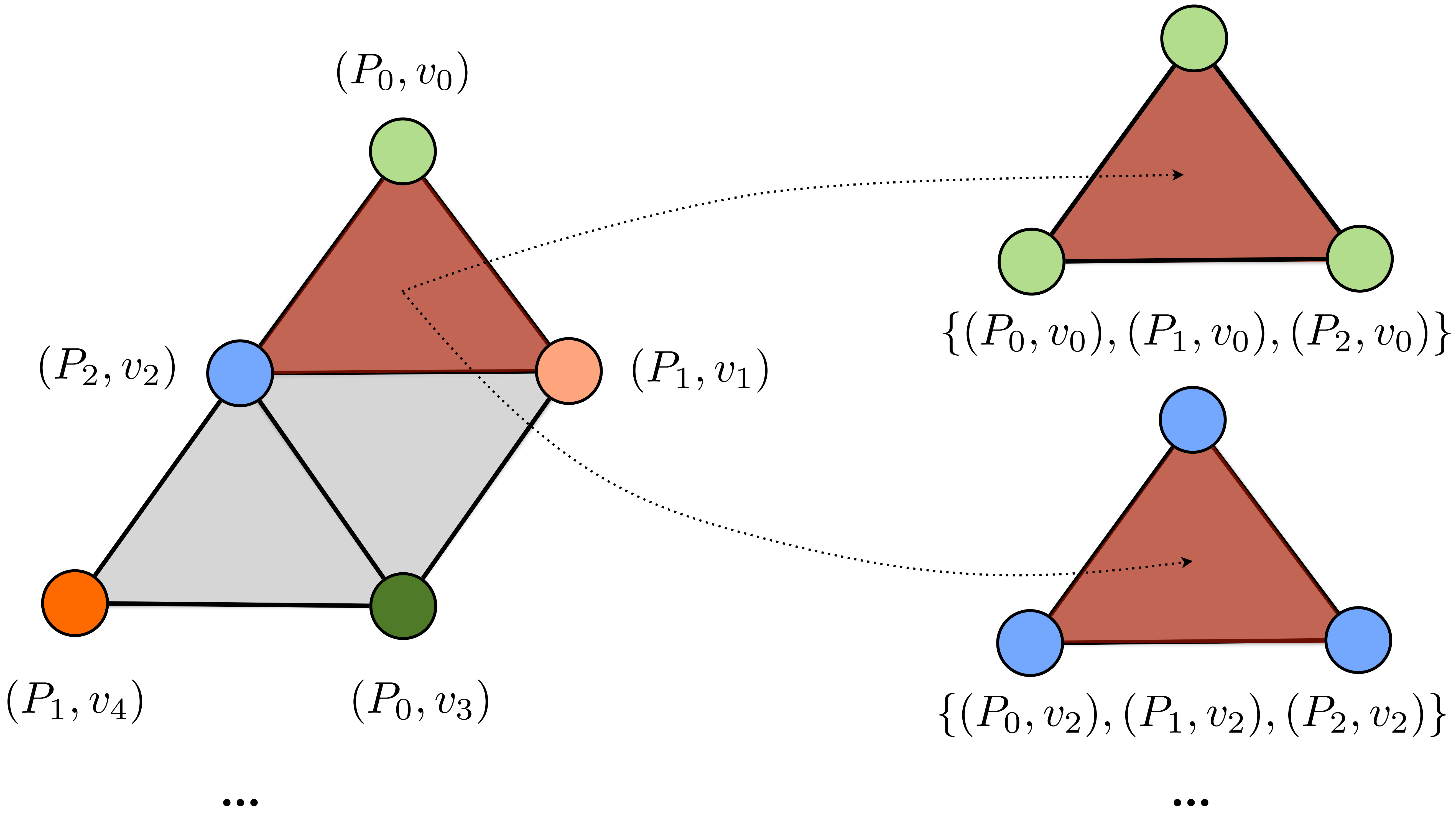
$\mathcal{I}$ (input complex)

$\mathcal{O}$ (output complex)

The map Δ



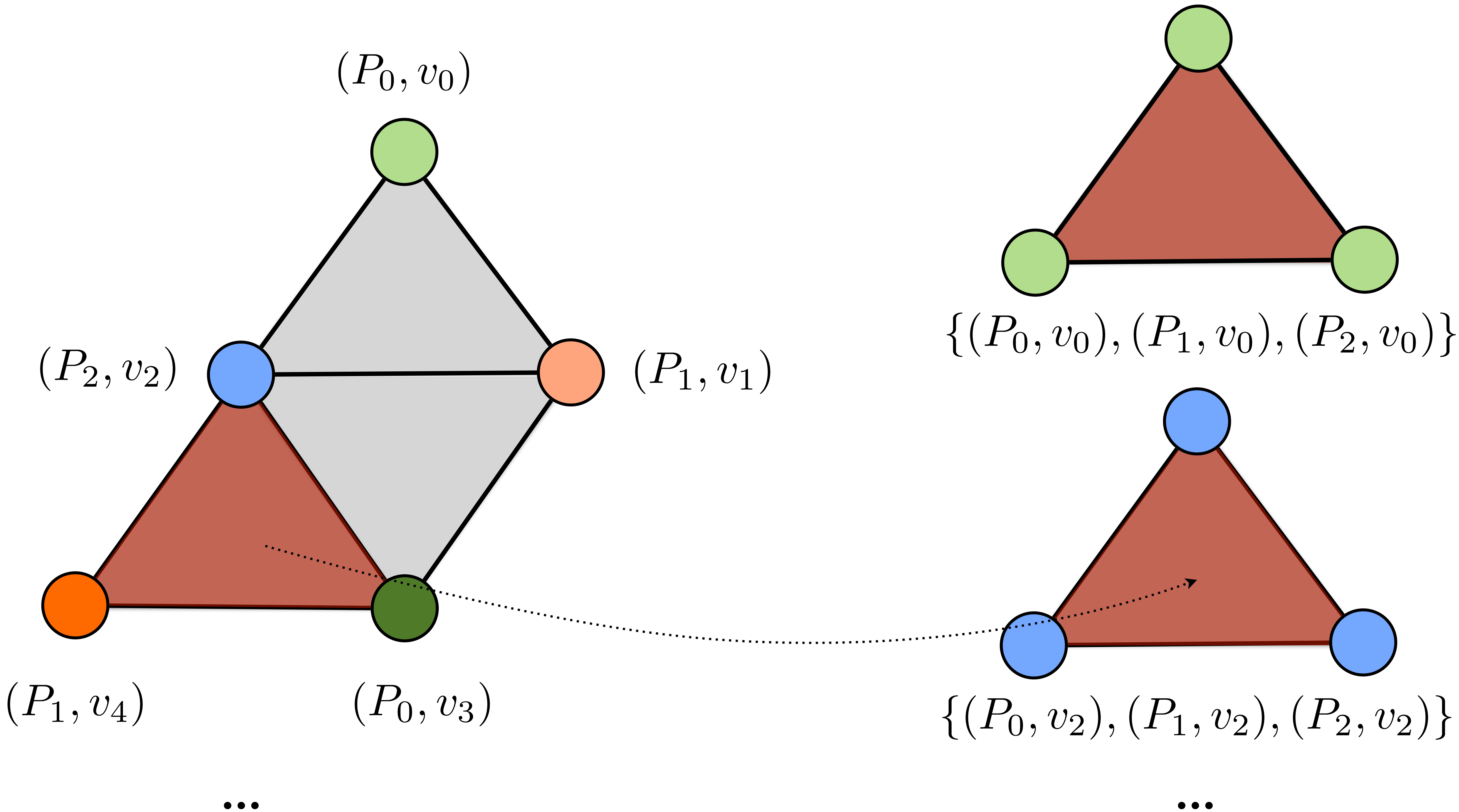
The map Δ



$\mathcal{I}$ (input complex)

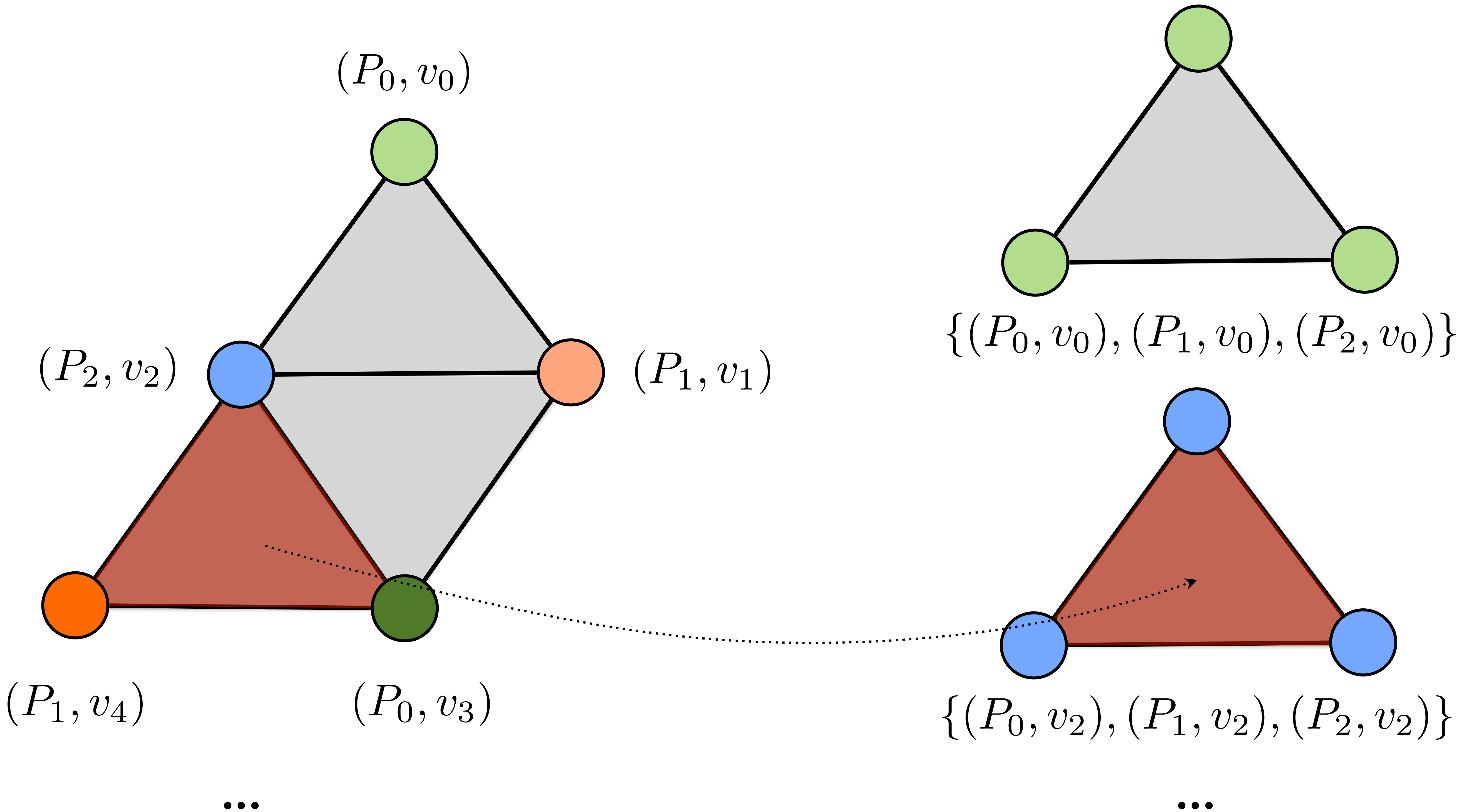
$\mathcal{O}$ (output complex)

The map Δ



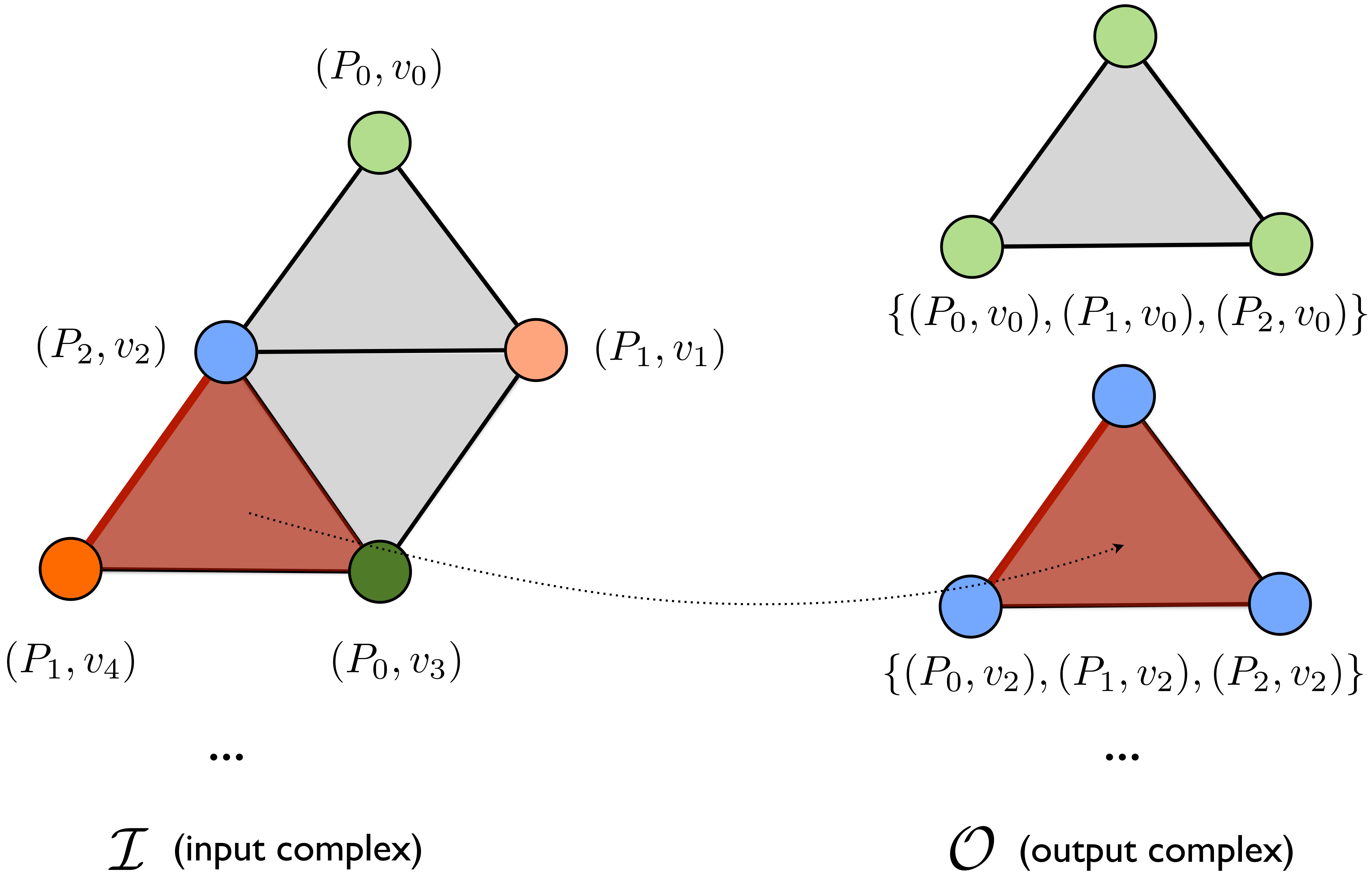
$\mathcal{I}$ (input complex) $\mathcal{O}$ (output complex)

The map Δ

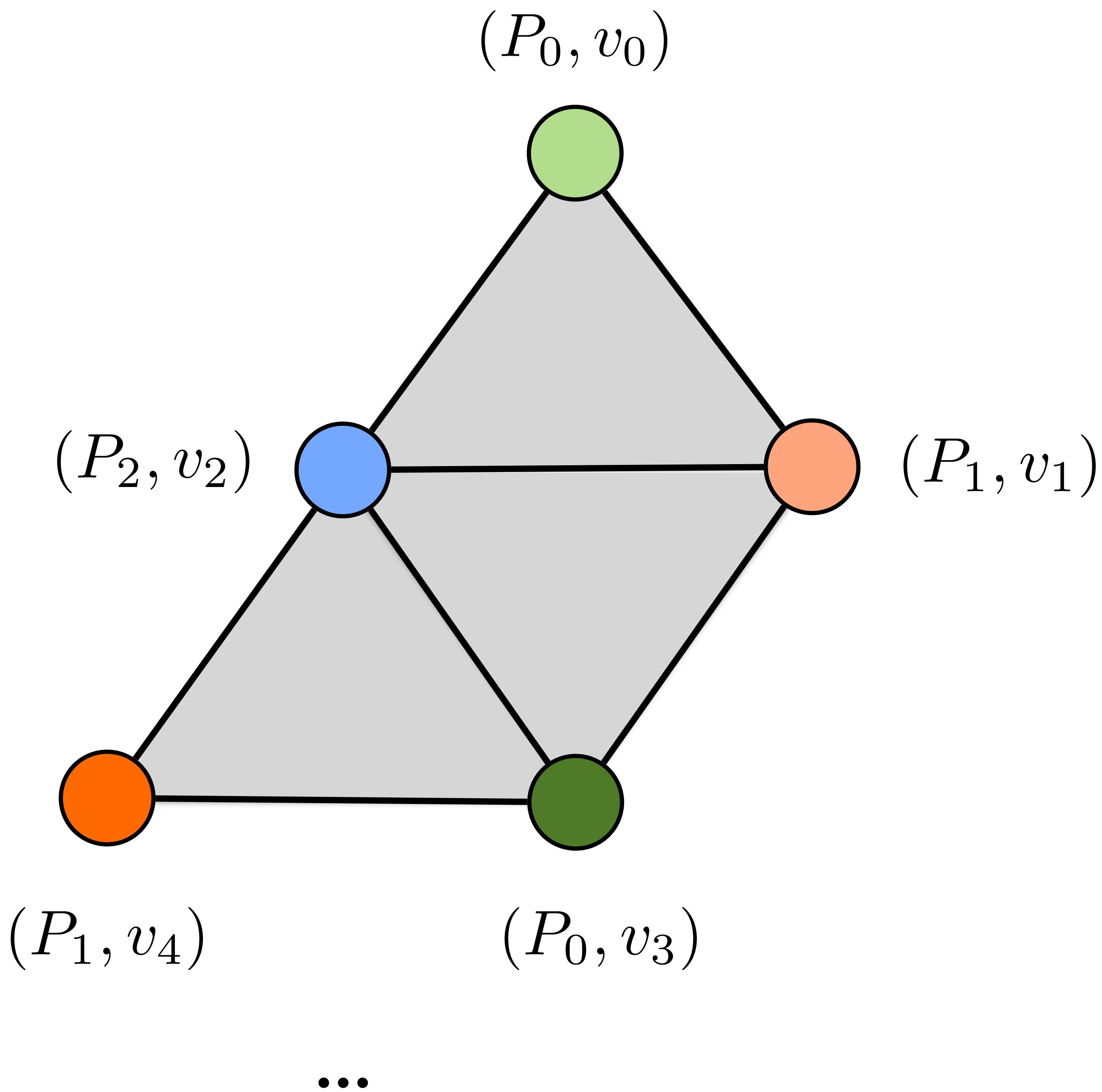


$\mathcal{I}$ (input complex) $\mathcal{O}$ (output complex)

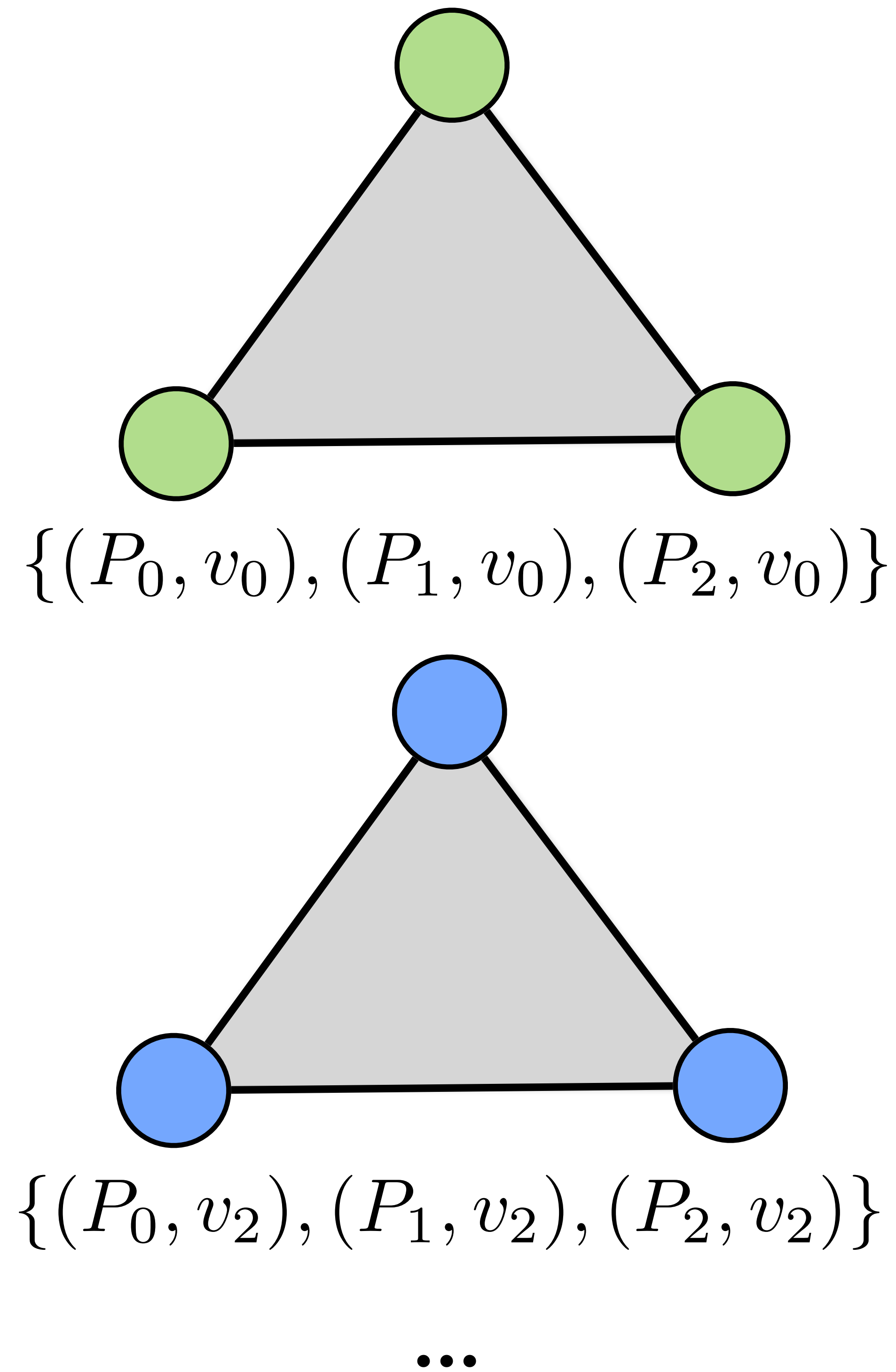
The map Δ



The map Δ



$\mathcal{I}$ (input complex)



$\mathcal{O}$ (output complex)

Formal Specification

In the crash-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

Formal Specification

In the crash-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

$\sigma \in \mathcal{I}$ if σ is an initial configuration

Formal Specification

In the crash-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

$\sigma \in \mathcal{I}$ if σ is an initial configuration

dim$(\sigma) = n$

Formal Specification

In the crash-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

$\sigma \in \mathcal{I}$ if σ is an initial configuration

$\dim(\sigma) = n$

$\tau \in \mathcal{O}$ if τ is a final configuration

Formal Specification

In the crash-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

$\sigma \in \mathcal{I}$ if σ is an initial configuration

$\dim(\sigma) = n$

$\tau \in \mathcal{O}$ if τ is a final configuration

$\dim(\sigma) \geq n - t$

Formal Specification

In the crash-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

$\sigma \in \mathcal{I}$ if σ is an initial configuration

$\dim(\sigma) = n$

$\tau \in \mathcal{O}$ if τ is a final configuration

$\dim(\sigma) \geq n - t$

$\tau \in \Delta(\sigma)$ if finishing with τ ...

Formal Specification

In the crash-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

$\sigma \in \mathcal{I}$ if σ is an initial configuration

dim$(\sigma) = n$

$\tau \in \mathcal{O}$ if τ is a final configuration

dim$(\sigma) \geq n - t$

$\tau \in \Delta(\sigma)$ if finishing with τ ...

... is fine when starting with σ

Solvability in Terms of Topology

Solvability in Terms of Topology

The Topological Structure of Asynchronous Computability

MAURICE HERLIHY

Brown University, Providence, Rhode Island

AND

NIR SHAVIT

Tel-Aviv University, Tel-Aviv Israel

1999!

Solvability in Terms of Topology

The Topological Structure of Asynchronous Computability

MAURICE HERLIHY

Brown University, Providence, Rhode Island

AND

NIR SHAVIT

Tel-Aviv University, Tel-Aviv Israel

1999!

Crash-failure task is solvable if and only if

Solvability in Terms of Topology

The Topological Structure of Asynchronous Computability

MAURICE HERLIHY

Brown University, Providence, Rhode Island

AND

NIR SHAVIT

Tel-Aviv University, Tel-Aviv Israel

1999!

Crash-failure task is solvable if and only if

continuous map $f : |\mathcal{I}| \to |\mathcal{O}|$ carried by Δ

Solvability in Terms of Topology

The Topological Structure of Asynchronous Computability

MAURICE HERLIHY

Brown University, Providence, Rhode Island

AND

NIR SHAVIT

Tel-Aviv University, Tel-Aviv Israel

1999!

Crash-failure task is solvable if and only if

continuous map $f : |\mathcal{I}| \to |\mathcal{O}|$ carried by Δ

respects Δ

Outline

1. Introduction

2. Asynchronous Byzantine Systems

3. Synchronous Byzantine Systems

4. Conclusion & Future Work

Outline

1. Introduction

2. Asynchronous Byzantine Systems

3. Synchronous Byzantine Systems

4. Conclusion & Future Work

Adversarial Model

Adversarial Model

Byzantine processes

Adversarial Model

Byzantine processes

up to t chosen by *adversary*

Adversarial Model

Byzantine processes

up to t chosen by *adversary*

Non-faulty processes output values...

Adversarial Model

Byzantine processes

up to t chosen by *adversary*

Non-faulty processes output values...

consistent

Adversarial Model

Byzantine processes

up to t chosen by *adversary*

Non-faulty processes output values...

consistent

...input values of non-faulty processes

Adversarial Model

Byzantine processes

up to t chosen by *adversary*

Non-faulty processes output values...

consistent

...input values of non-faulty processes

$\triangle$

Adversarial Model

Byzantine processes

up to t chosen by *adversary*

Non-faulty processes output values...

consistent

...input values of non-faulty processes

Δ

Adversarial Model

Byzantine processes

up to t chosen by *adversary*

Non-faulty processes output values...

consistent

...input values of non-faulty processes

$\triangle$

$\triangle$ constrains non-faulty processes *only*

Adversarial Model

Byzantine processes

up to t chosen by *adversary*

Non-faulty processes output values...

consistent

...input values of non-faulty processes

Δ

Δ constrains non-faulty processes *only*

$\mathcal{I}$ and $\mathcal{O}$ represent non-faulty processes *only*

What about Equivocation?

What about Equivocation?

Different information to different processes?

What about Equivocation?

Different information to different processes? ✘

What about Equivocation?

Different information to different processes? ✗

Reliable Broadcast!

What about Equivocation?

Different information to different processes? ✗

Reliable Broadcast!

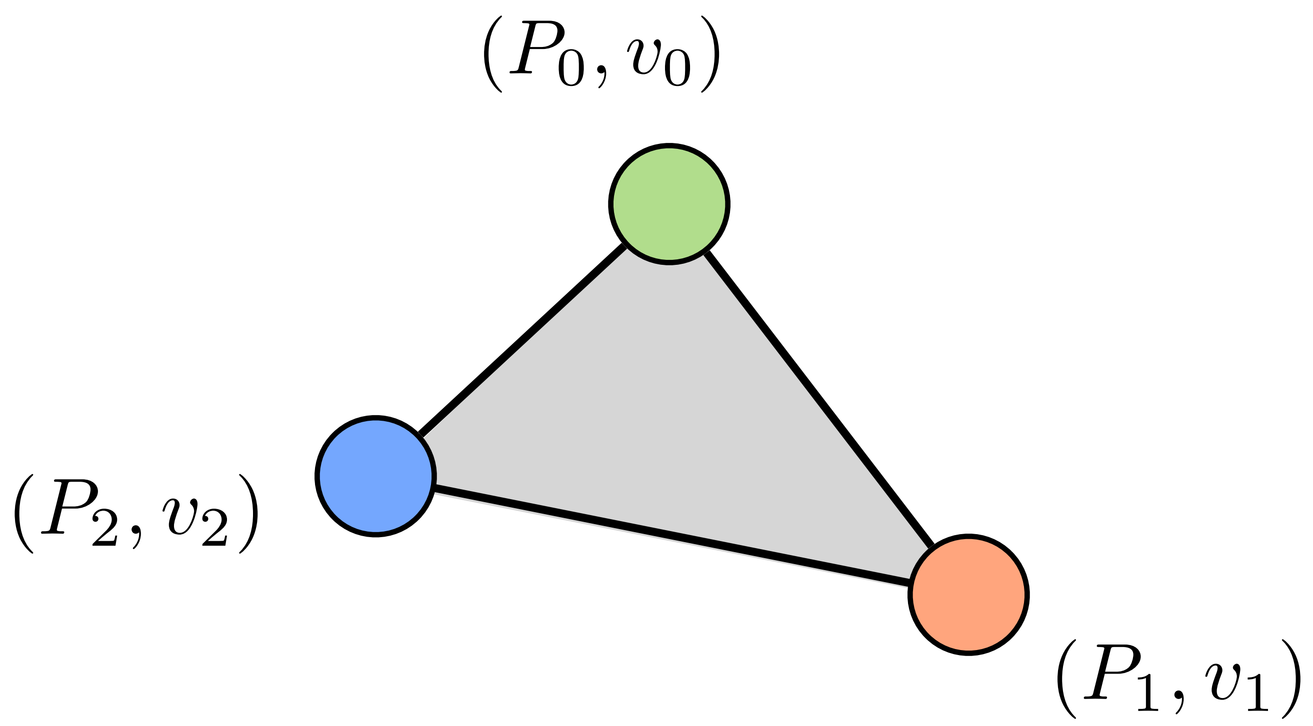
Real problem is:

What about Equivocation?

Different information to different processes? ✘

Reliable Broadcast!

Real problem is:

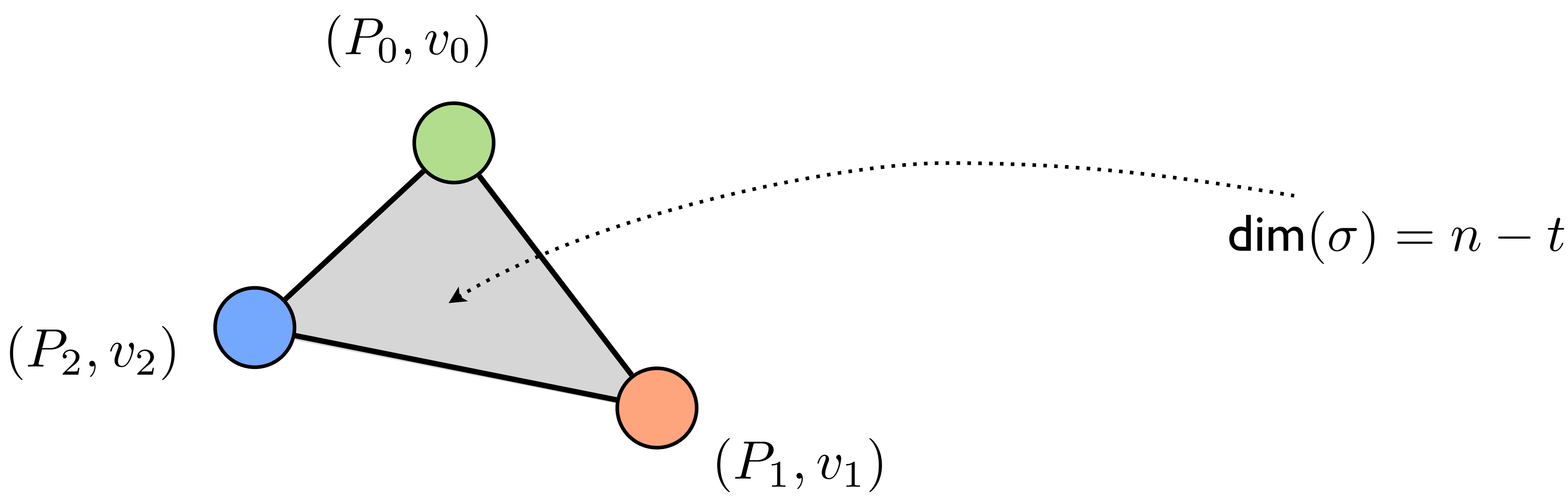


What about Equivocation?

Different information to different processes? ✘

Reliable Broadcast!

Real problem is:

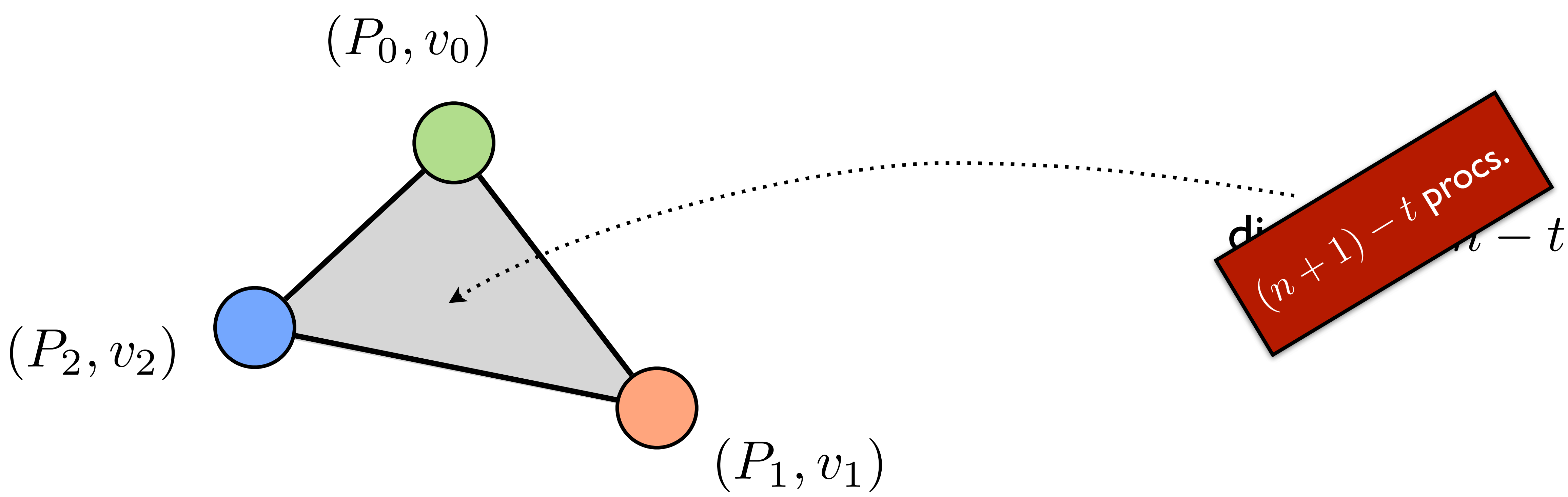


What about Equivocation?

Different information to different processes? ✗

Reliable Broadcast!

Real problem is:

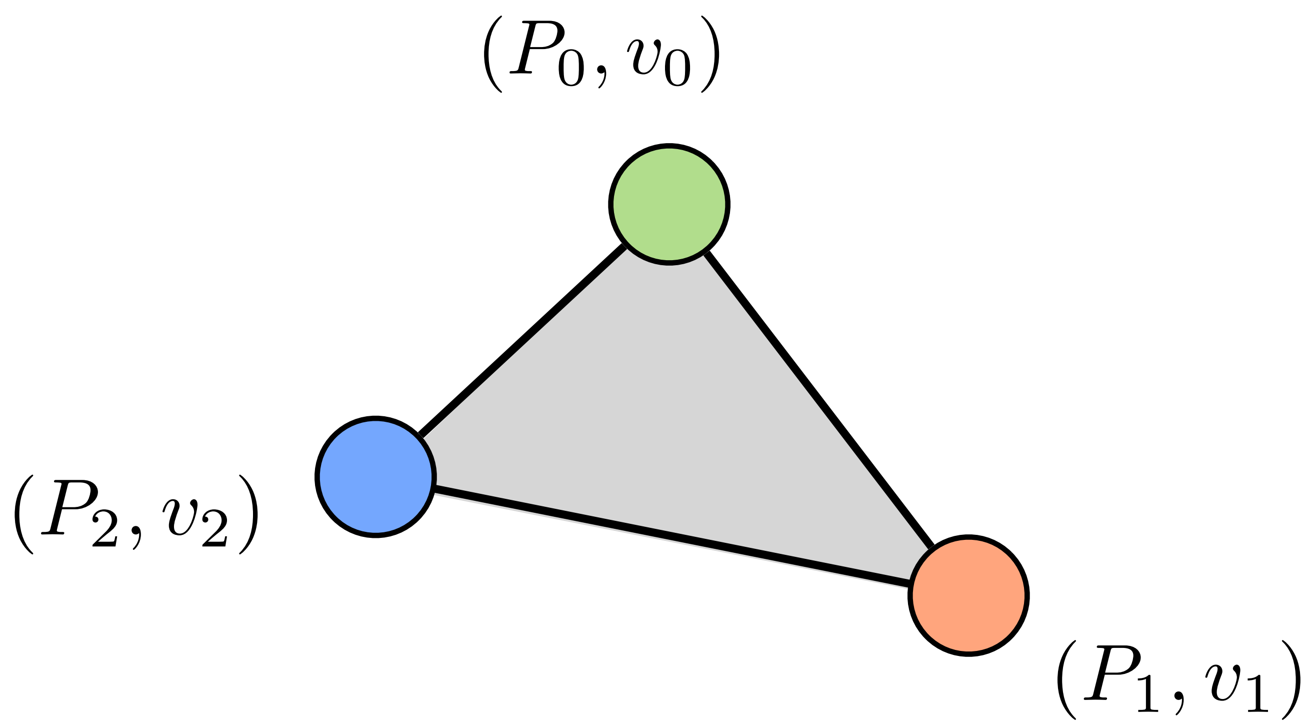


What about Equivocation?

Different information to different processes? ✗

Reliable Broadcast!

Real problem is:

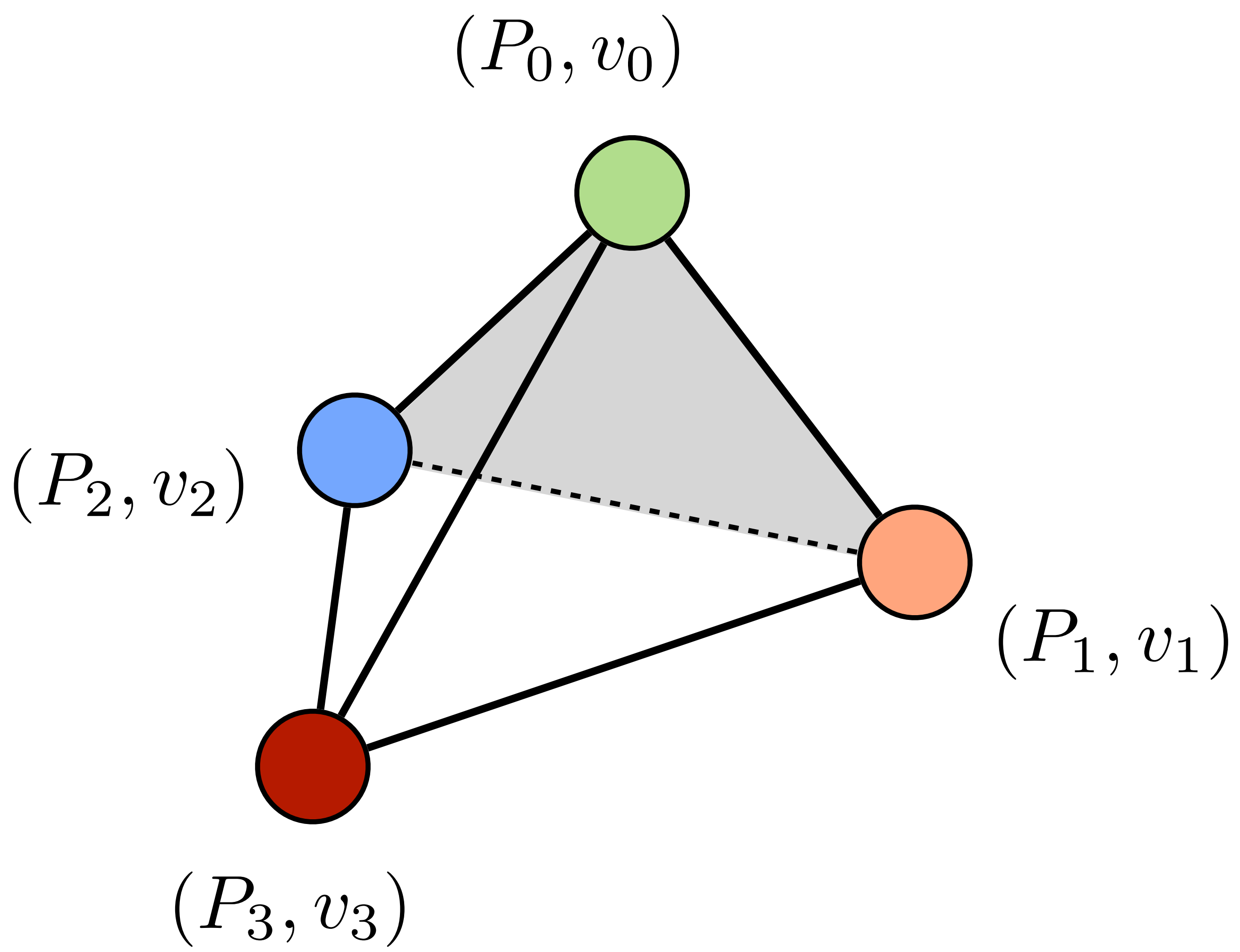


What about Equivocation?

Different information to different processes? ✘

Reliable Broadcast!

Real problem is:

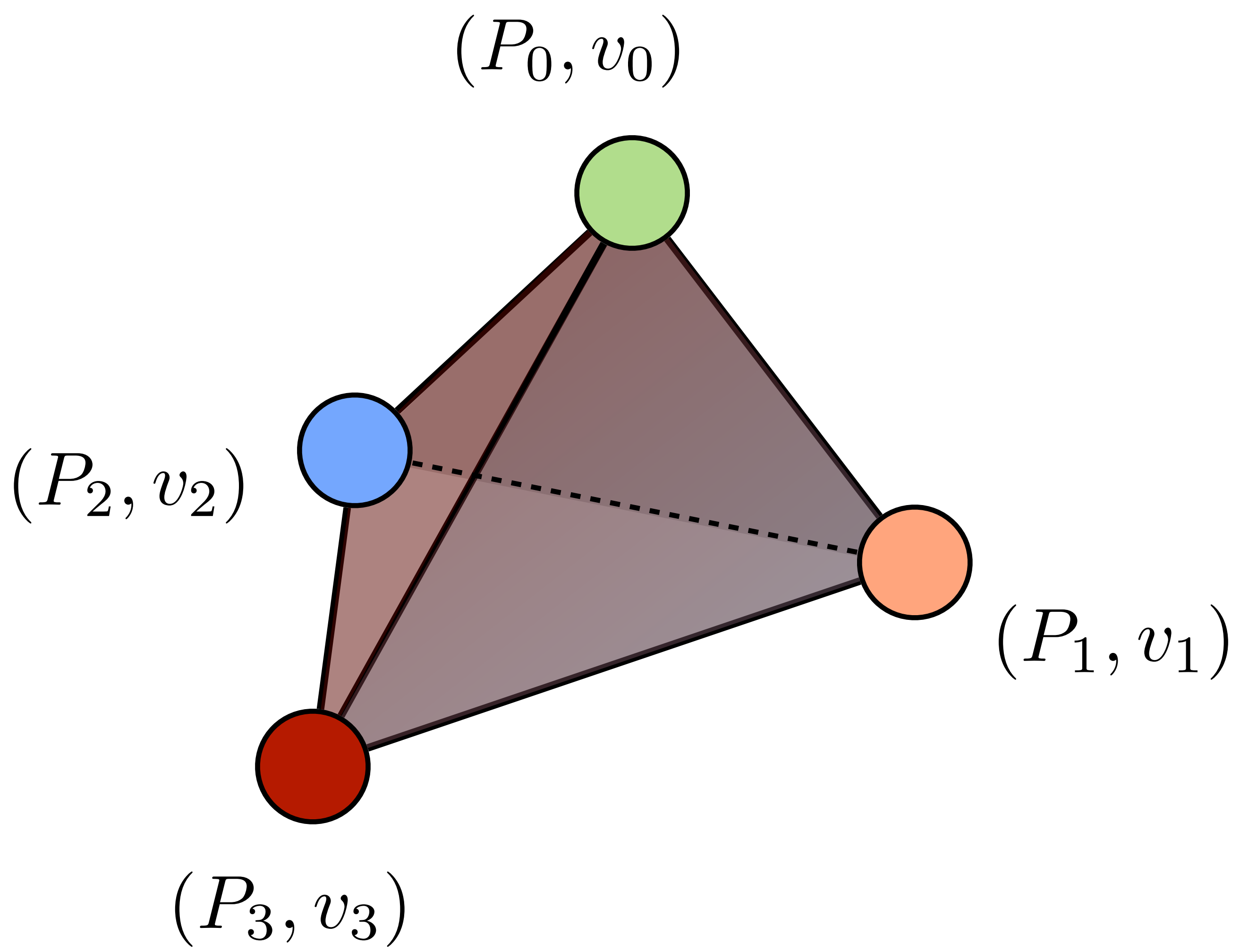


What about Equivocation?

Different information to different processes? ✘

Reliable Broadcast!

Real problem is:

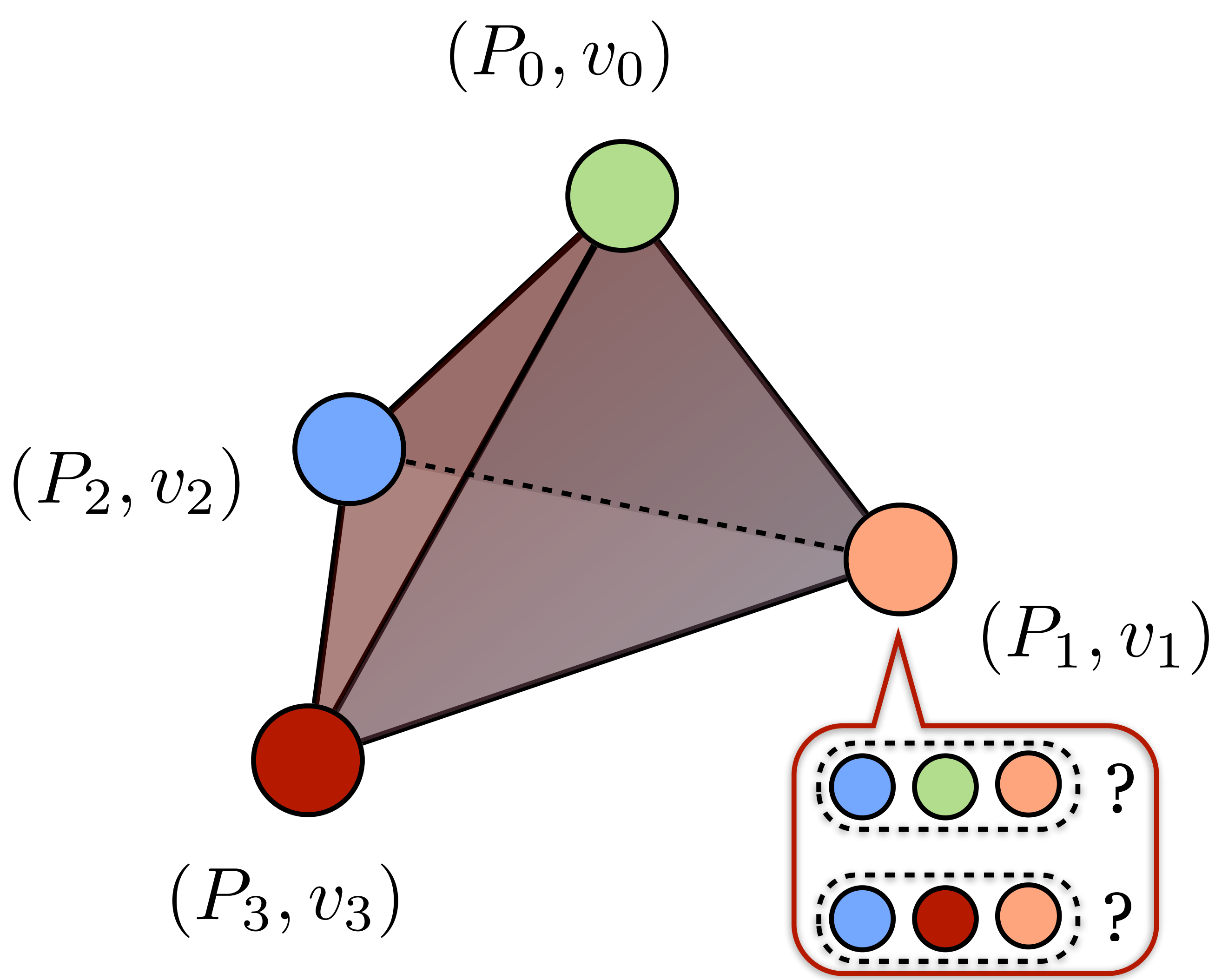


What about Equivocation?

Different information to different processes? ✘

Reliable Broadcast!

Real problem is:

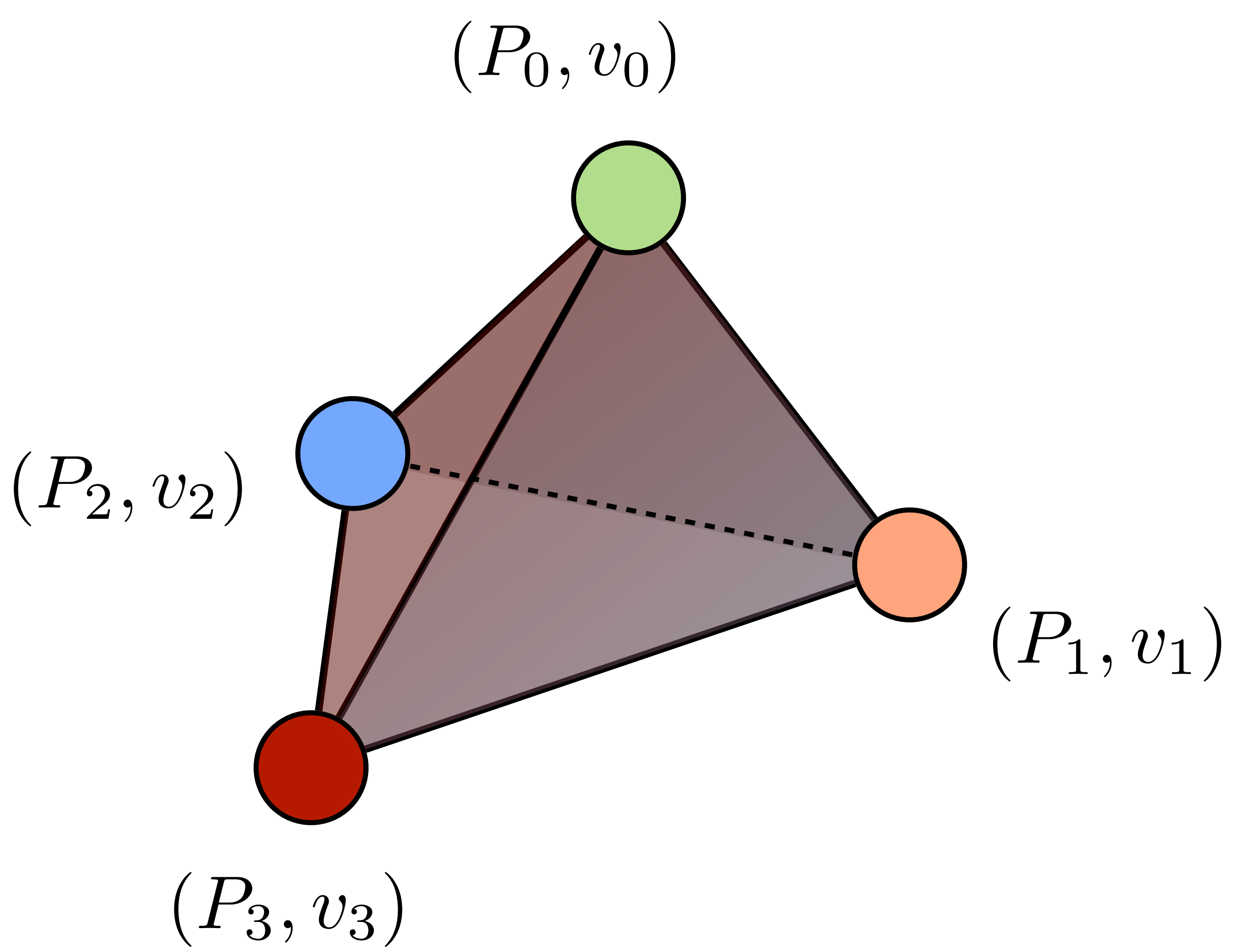


What about Equivocation?

Different information to different processes? ✘

Reliable Broadcast!

Real problem is:

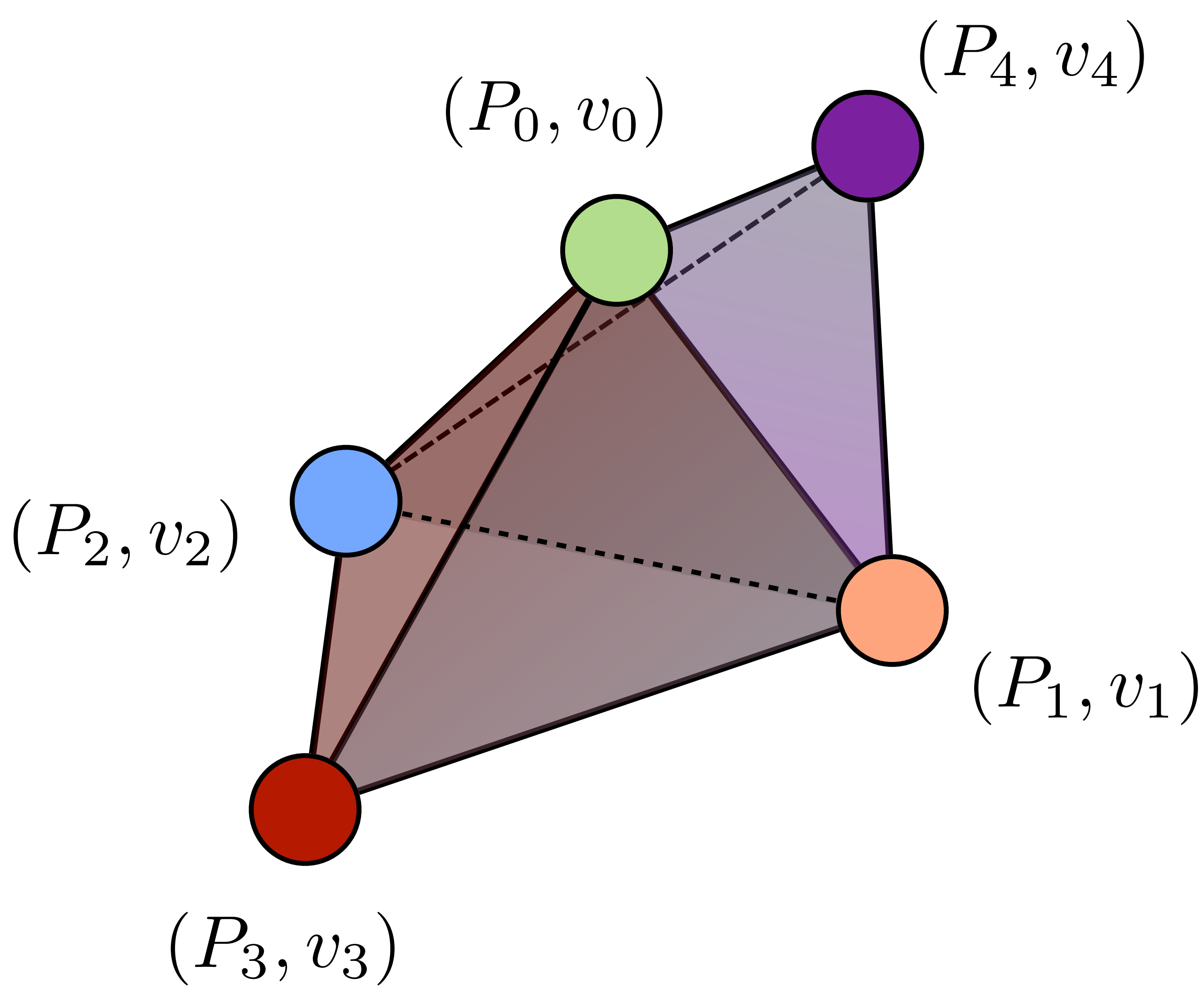


What about Equivocation?

Different information to different processes? ✗

Reliable Broadcast!

Real problem is:

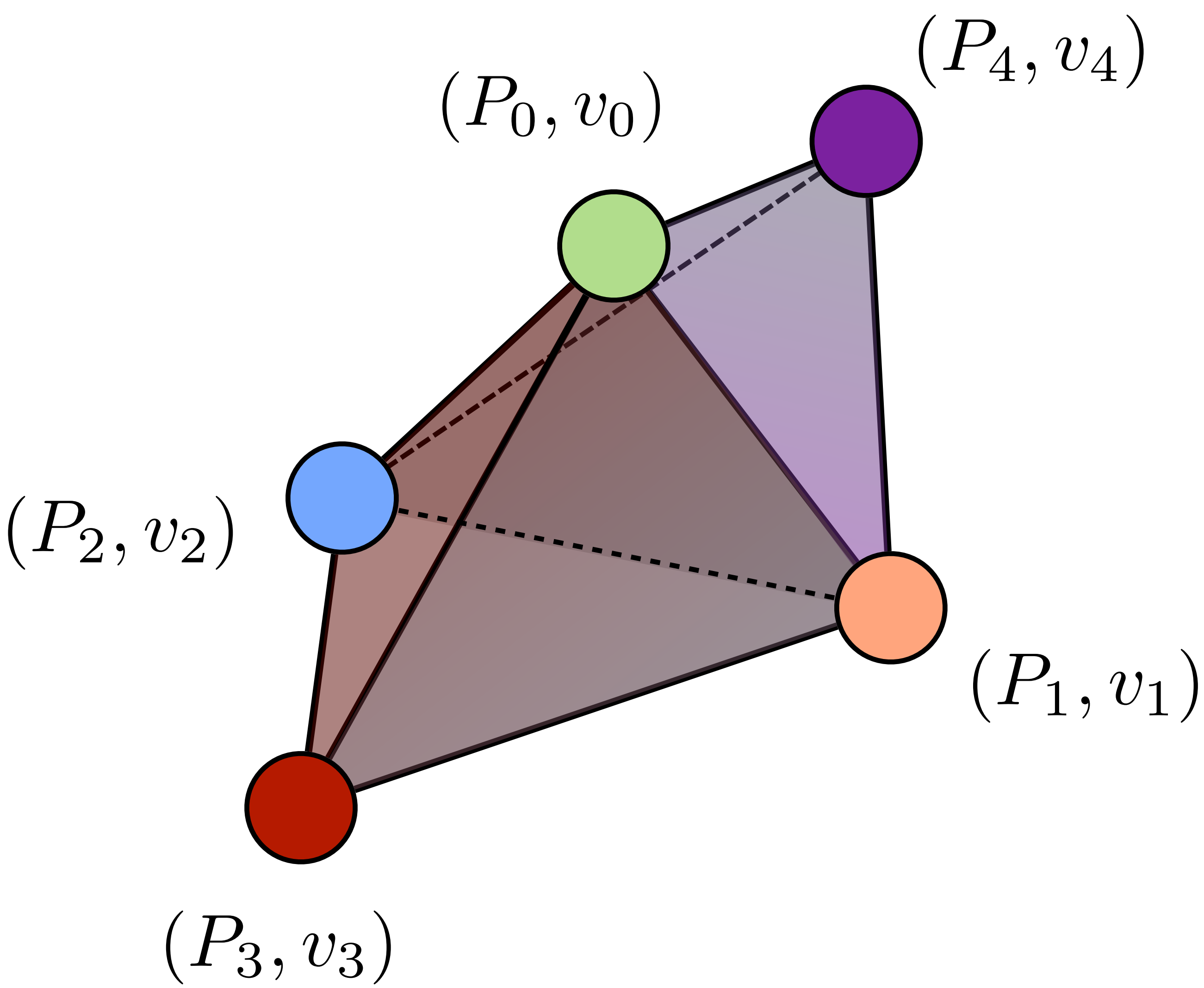


What about Equivocation?

Different information to different processes? ✗

Reliable Broadcast!

Real problem is:



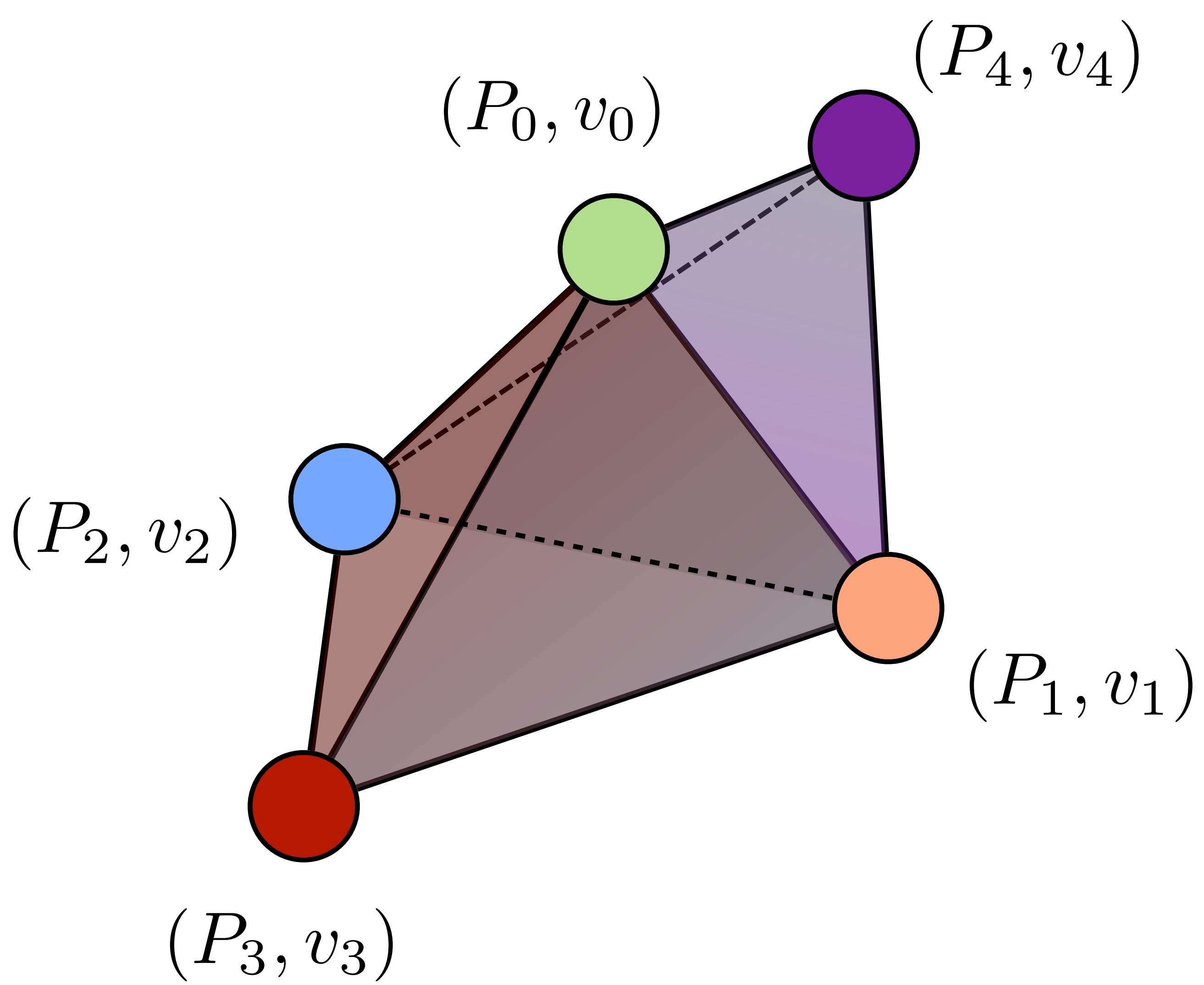
Byzantine processes pick fake inputs...

What about Equivocation?

Different information to different processes? ✗

Reliable Broadcast!

Real problem is:



Byzantine processes
pick fake inputs…

… yet behave "correctly"

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

$\sigma \in \mathcal{I}$ if σ is an initial configuration

$\dim(\sigma) = n$

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

(non-faulty)

$\sigma \in \mathcal{I}$ if σ is an initial configuration

$\text{dim}(\sigma) = n$

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

(non-faulty)

$\sigma \in \mathcal{I}$ if σ is an initial configuration

$\text{dim}(\sigma) = n$

Initially $(n+1)$ procs, but any set of $0 \ldots t$ of them are faulty

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

(non-faulty)

Initially $(n+1)$ procs, but any set of $0 \ldots t$ of them are faulty

$\sigma \in \mathcal{I}$ if σ is an initial configuration

$\text{dim}(\sigma) = n$

$\tau \in \mathcal{O}$ if τ is a final configuration

$\text{dim}(\sigma) \geq n - t$

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

(non-faulty)

$\sigma \in \mathcal{I}$ if σ is an initial configuration

$\text{dim}(\sigma) = n$

Initially $(n+1)$ procs, but any set of $0 \ldots t$ of them are faulty

(non-faulty)

$\tau \in \mathcal{O}$ if τ is a final configuration

$\text{dim}(\sigma) \geq n - t$

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

(non-faulty)

$\sigma \in \mathcal{I}$ if σ is an initial configuration

dim$(\sigma) = n$

Initially $(n+1)$ procs, but any set of $0 \ldots t$ of them are faulty

(non-faulty)

$\tau \in \mathcal{O}$ if τ is a final configuration

dim$(\sigma) \geq n - t$

$\tau \in \Delta(\sigma)$ if finishing with τ ...

... is fine when starting with σ

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

(non-faulty)

Initially $(n+1)$ procs, but any set of $0 \ldots t$ of them are faulty

$\sigma \in \mathcal{I}$ if σ is an initial configuration

dim$(\sigma) = n$

(non-faulty)

$\tau \in \mathcal{O}$ if τ is a final configuration

dim$(\sigma) \geq n - t$

$\tau \in \Delta(\sigma)$ if finishing with τ ...

... is fine when starting with σ

(non-faulty)

Formal Specification

In the Byzantine-failure model: $(\mathcal{I}, \mathcal{O}, \Delta)$

(non-faulty)

$\sigma \in \mathcal{I}$ if σ is an initial configuration

Initially $(n+1)$ procs, but any set of $0 \ldots t$ of them are faulty

dim$(\sigma) = n$

(non-faulty)

$\tau \in \mathcal{O}$ if τ is a final configuration

dim$(\sigma) \geq n - t$

$\tau \in \Delta(\sigma)$ if finishing with τ ...

... is fine when starting with σ

(non-faulty)

(non-faulty)

When are these Byzantine tasks solvable?

Reduction to Crash-Failure Tasks

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

$$\Leftrightarrow$$

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

$\Leftrightarrow$

its *dual* crash-failure task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ solvable.

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

$\Leftrightarrow$

its *dual* crash-failure task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ solvable.

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

$$\Leftrightarrow$$

its *dual* crash-failure task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ solvable.

translated from $(\mathcal{I}, \mathcal{O}, \Delta)$

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

$$\Leftrightarrow$$

its *dual* crash-failure task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ solvable.

translated from $(\mathcal{I}, \mathcal{O}, \Delta)$

In [STOC14], we show what is the dual task,

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

$$\Leftrightarrow$$

its *dual* crash-failure task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ solvable.

translated from $(\mathcal{I}, \mathcal{O}, \Delta)$

In [STOC14], we show what is the dual task,

and how the equivalence holds

*Algorithmic methods**

used to prove theorem

*Algorithmic methods**
used to prove theorem

** simulations, reductions*

[STOC14]

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

$$\Leftrightarrow$$

its *dual* crash-failure task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ solvable.

Reduction to Crash-Failure Tasks

Theorem: (Equivalence Theorem)

A Byzantine task $(\mathcal{I}, \mathcal{O}, \Delta)$ solvable

$\Leftrightarrow$

its *dual* crash-failure task $(\tilde{\mathcal{I}}, \tilde{\mathcal{O}}, \tilde{\Delta})$ solvable.

$\Updownarrow$

protocol $\Leftrightarrow \exists$ continuous $f : |\tilde{\mathcal{I}}| \to |\tilde{\mathcal{O}}|$ carried by $\tilde{\Delta}$

Colorless Tasks

Colorless Tasks

Colorless Tasks

Start with any of

Colorless Tasks

Start with any of v_0 v_2 v_1 v_3

Colorless Tasks

Start with any of v_0 v_2 v_1 v_3

Finish with $\leq k$ of

Colorless Tasks

Start with any of

v_0	v_2
v_1	v_3

Finish with $\leq k$ of

$(k = 2)$

Colorless Tasks

Start with any of

v_0	v_2
v_1	v_3

Finish with $\leq k$ of

$(k = 2)$

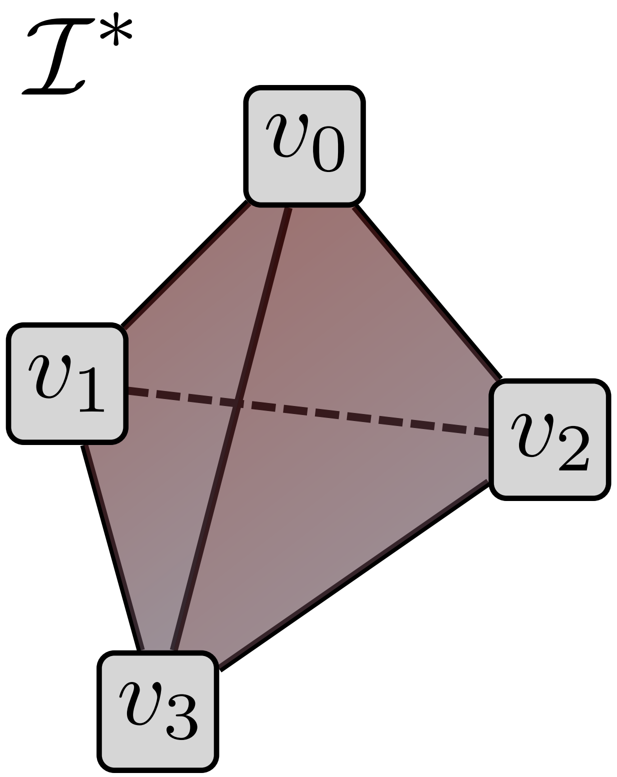
v_0	v_2
v_1	v_3

Colorless Tasks

Start with any of v_0 v_2 v_1 v_3

Finish with $\leq k$ of v_0 v_2 v_1 v_3

$(k = 2)$

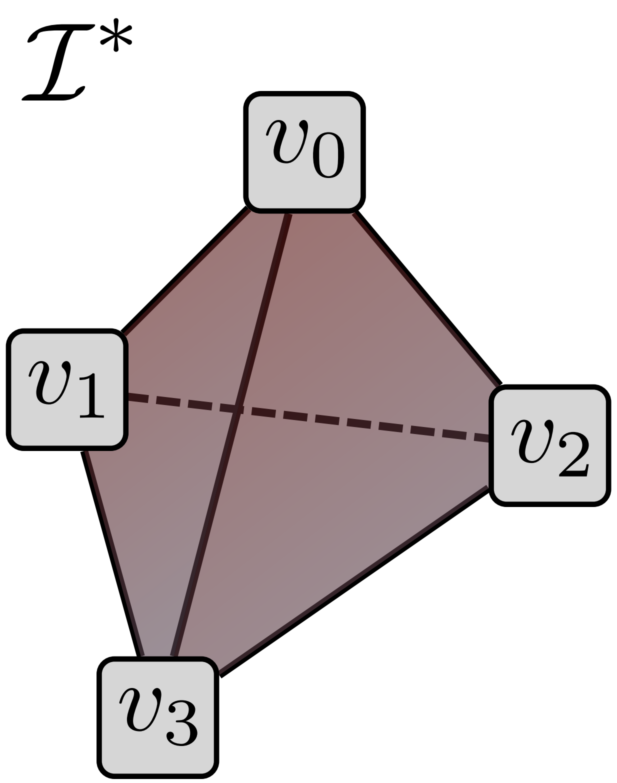


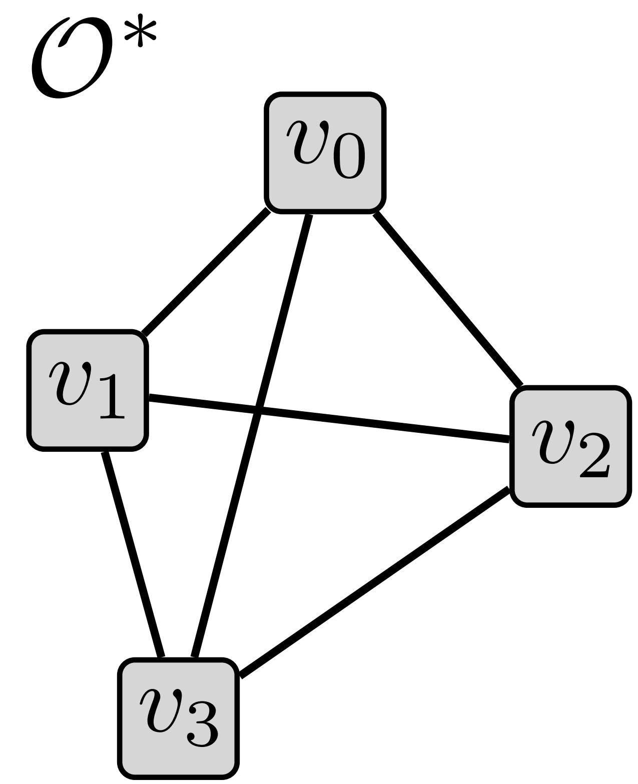
Colorless Tasks

Start with any of v_0 v_2 v_1 v_3

Finish with $\leq k$ of v_0 v_2 v_1 v_3

$(k = 2)$

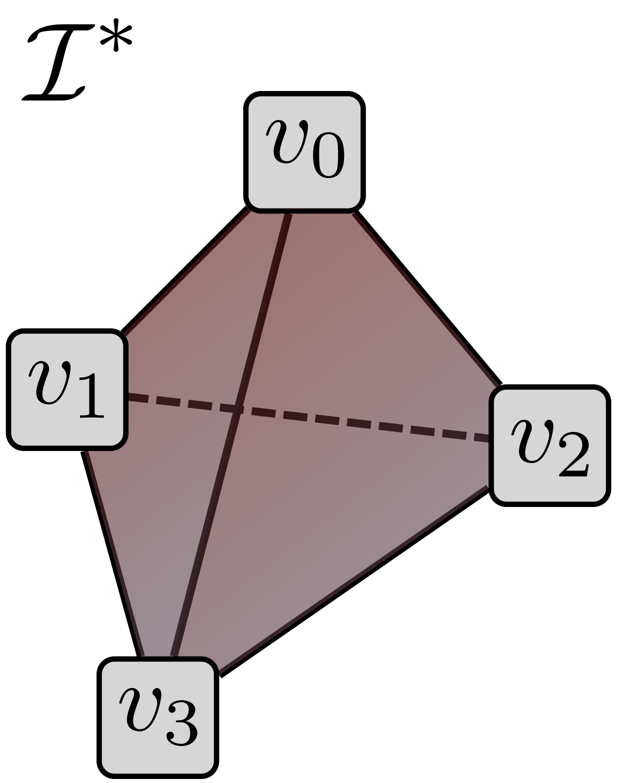




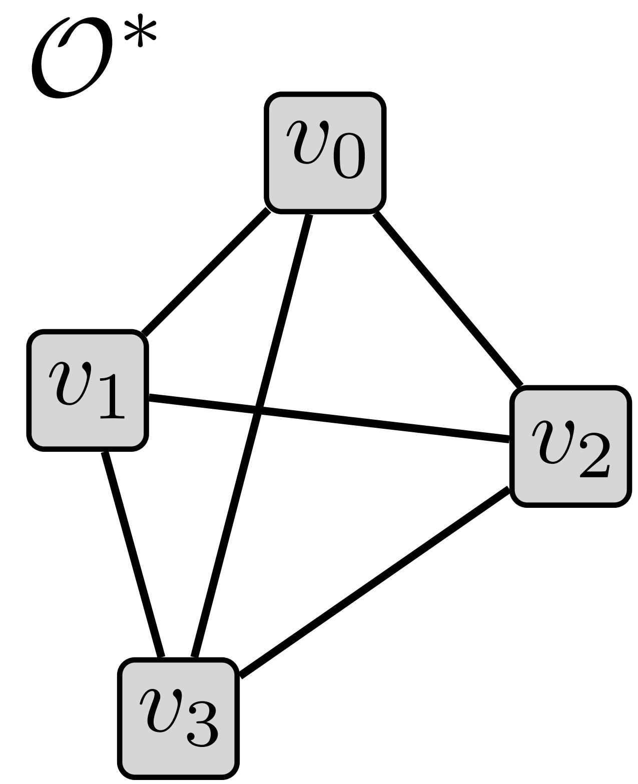
Colorless Tasks

Start with any of v_0 v_2 v_1 v_3

Finish with $\leq k$ of v_0 v_2 v_1 v_3 $(k = 2)$

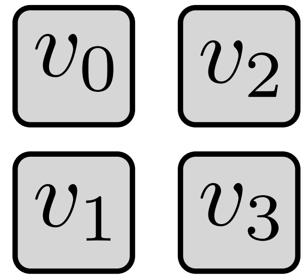


$\Delta^* = \text{skel}^{k-1}$ 1

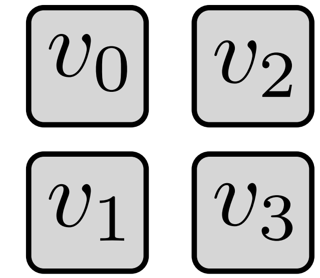


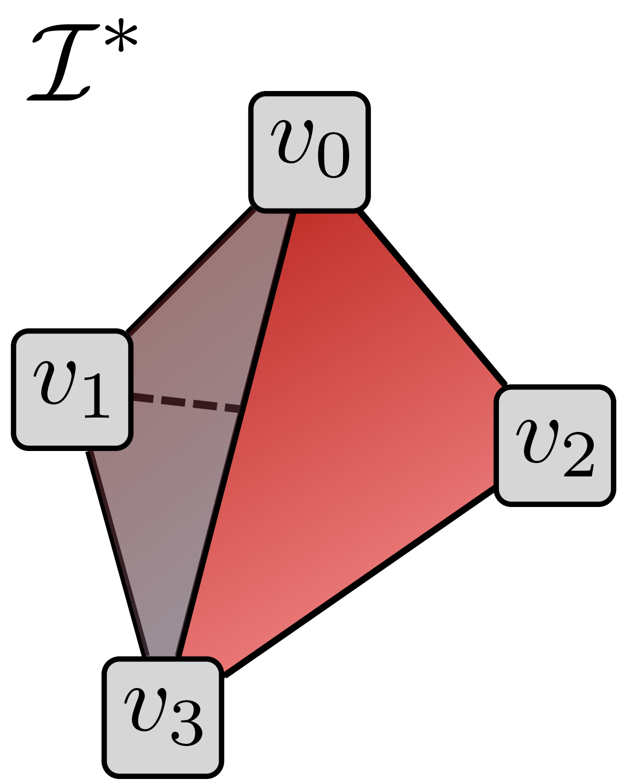
Colorless Tasks

Start with any of

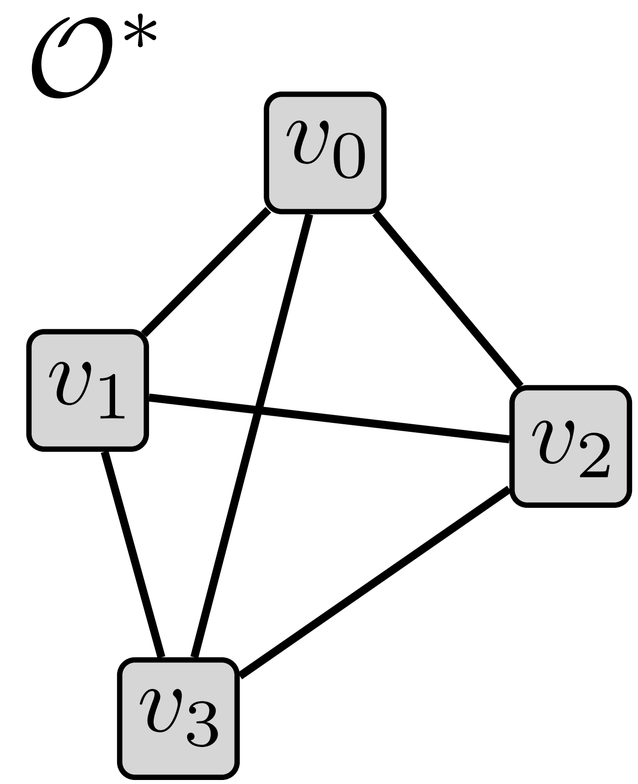


Finish with $\leq k$ of $(k = 2)$





$\Delta^* = \text{skel}^{k-1}$ 1

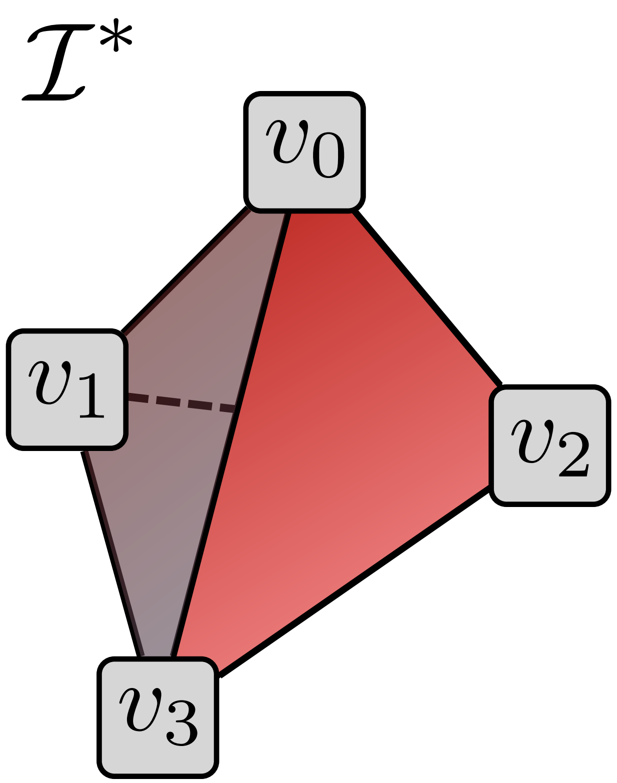


Colorless Tasks

Start with any of v_0 v_2 v_1 v_3

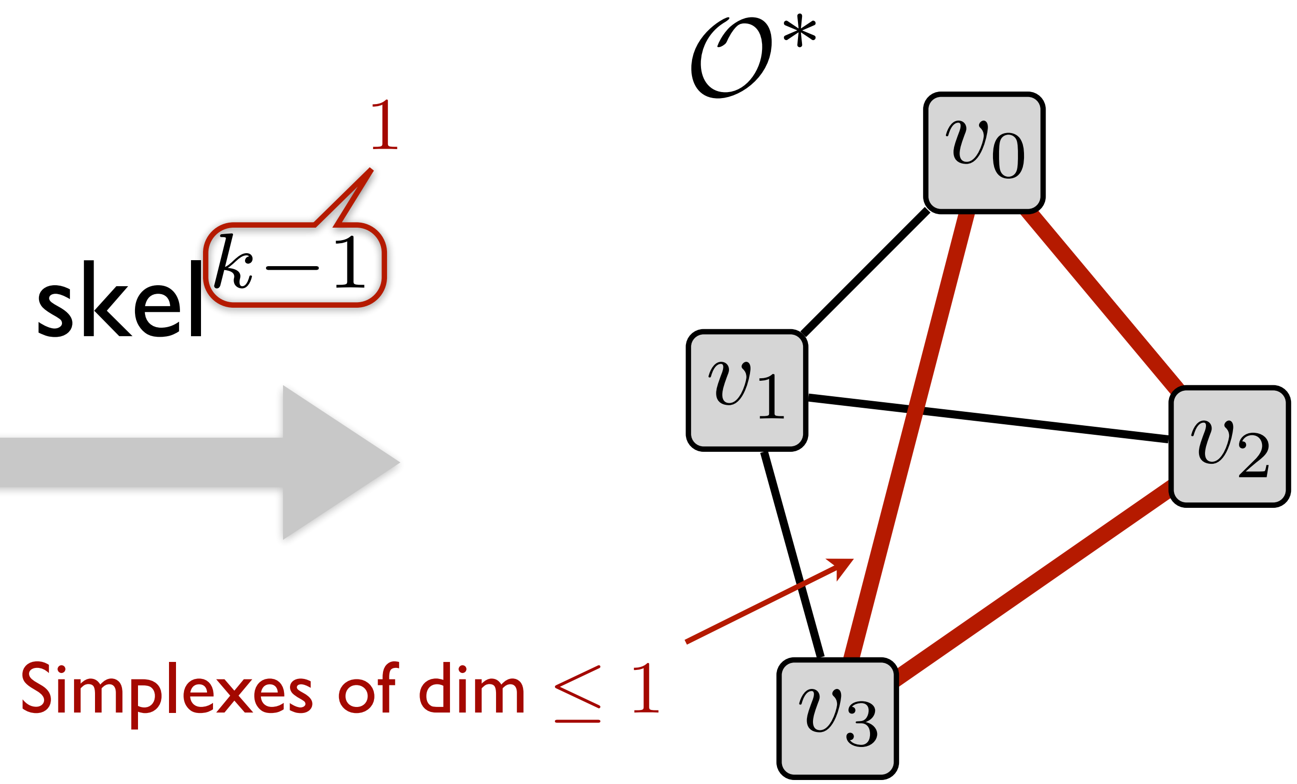
Finish with $\leq k$ of v_0 v_2 v_1 v_3

$(k = 2)$



$\Delta^* = \text{skel}^{k-1}$

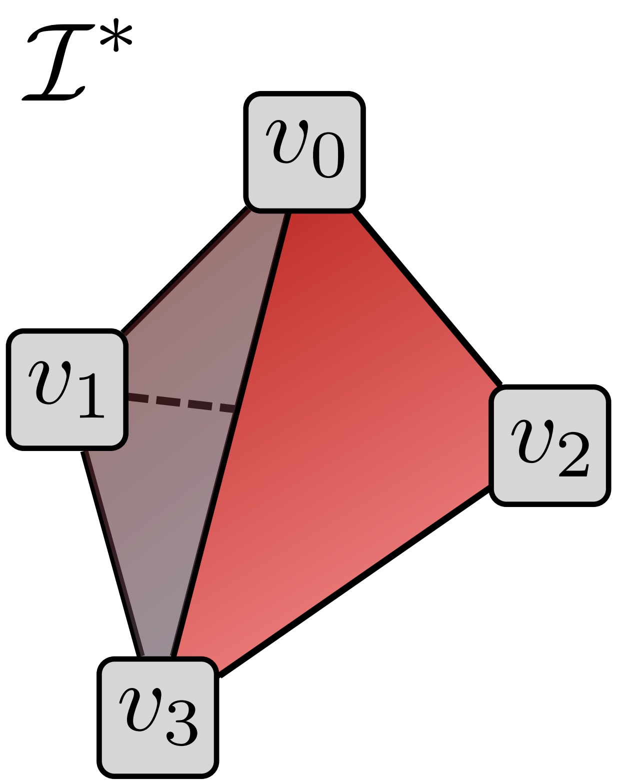
1



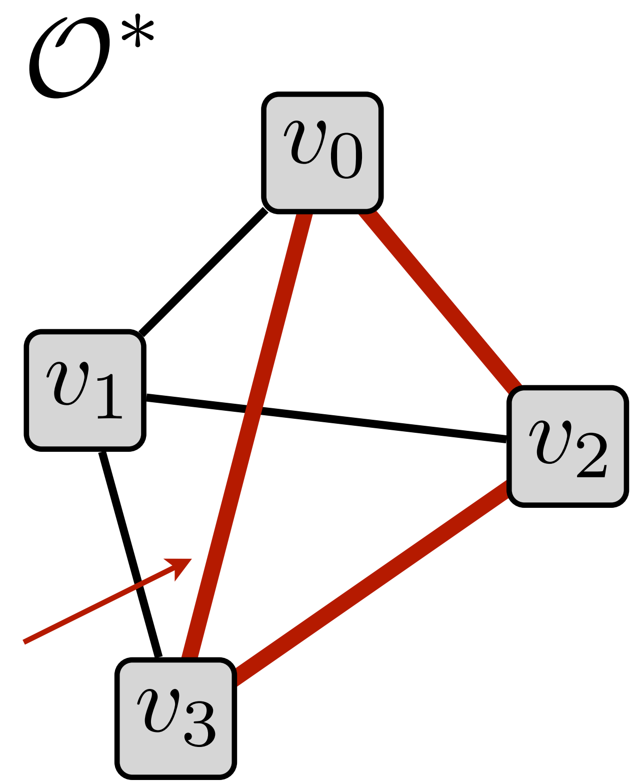
Colorless Tasks

Start with any of v_0 v_2 v_1 v_3

Finish with $\leq k$ of v_0 v_2 v_1 v_3 $(k = 2)$



$\Delta^* = \text{skel}^{k-1}$ (1)



Strict k-set agreement

Theorems for Colorless Tasks

Theorems for Colorless Tasks

Theorem:

The strict $(t+1)$-set agreement $(\mathcal{I}^*, \mathcal{O}^*, \text{skel}^t)$ has a t-resilient Byzantine asynchronous protocol *iff*

Theorems for Colorless Tasks

Theorem:

The strict $(t+1)$-set agreement $(\mathcal{I}^*, \mathcal{O}^*, \text{skel}^t)$ has a t-resilient Byzantine asynchronous protocol *iff*

$$n + 1 > t(\text{dim}(\mathcal{I}^*) + 2) \text{ or } \text{dim}(\mathcal{I}^*) \leq t.$$

Theorems for Colorless Tasks

Theorem:

The strict $(t+1)$-set agreement $(\mathcal{I}^*, \mathcal{O}^*, \text{skel}^t)$ has a t-resilient Byzantine asynchronous protocol *iff*

$$n + 1 > t(\dim(\mathcal{I}^*) + 2) \text{ or } \dim(\mathcal{I}^*) \leq t.$$

Theorems for Colorless Tasks

Theorem:

The strict $(t+1)$-set agreement $(\mathcal{I}^*, \mathcal{O}^*, \text{skel}^t)$ has a t-resilient Byzantine asynchronous protocol *iff*

$$n + 1 > t(\text{dim}(\mathcal{I}^*) + 2) \text{ or } \text{dim}(\mathcal{I}^*) \leq t.$$

(application of our Equivalence Theorem)

Theorems for Colorless Tasks

Theorems for Colorless Tasks

Theorem:

For any colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, if

Theorems for Colorless Tasks

Theorem:

For any colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, if

1. $n + 1 > t(\text{dim}(\mathcal{I}^*) + 2)$

Theorems for Colorless Tasks

Theorem:

For any colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, if

1. $n + 1 > t(\text{dim}(\mathcal{I}^*) + 2)$
2. $\exists$ continuous map $f : |\,\text{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ carried by Δ^*,

Theorems for Colorless Tasks

Theorem:

For any colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, if

1. $n + 1 > t(\text{dim}(\mathcal{I}^*) + 2)$
2. $\exists$ continuous map $f : |\,\text{skel}^t(\mathcal{I}^*)| \rightarrow |\mathcal{O}^*|$ carried by Δ^*,

we have a t-resilient Byzantine asynchronous protocol.

Theorems for Colorless Tasks

Theorem:

For any colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, if

1. $n + 1 > t(\text{dim}(\mathcal{I}^*) + 2)$
2. $\exists$ continuous map $f : |\,\text{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ carried by Δ^*,

we have a t-resilient Byzantine asynchronous protocol.

Proof sketch:

Theorems for Colorless Tasks

Theorem:

For any colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, if

1. $n + 1 > t(\text{dim}(\mathcal{I}^*) + 2)$
2. $\exists$ continuous map $f : |\,\text{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ carried by Δ^*,

we have a t-resilient Byzantine asynchronous protocol.

Proof sketch:

1. Run the Byzantine strict $(t+1)$-set agreement protocol, landing on a simplex in $\text{skel}^t(\mathcal{I}^*)$.

Theorems for Colorless Tasks

Theorem:

For any colorless task $(\mathcal{I}^*, \mathcal{O}^*, \Delta^*)$, if

1. $n + 1 > t(\text{dim}(\mathcal{I}^*) + 2)$
2. $\exists$ continuous map $f : |\,\text{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ carried by Δ^*,

we have a t-resilient Byzantine asynchronous protocol.

Proof sketch:

1. Run the Byzantine strict $(t+1)$-set agreement protocol, landing on a simplex in $\text{skel}^t(\mathcal{I}^*)$.

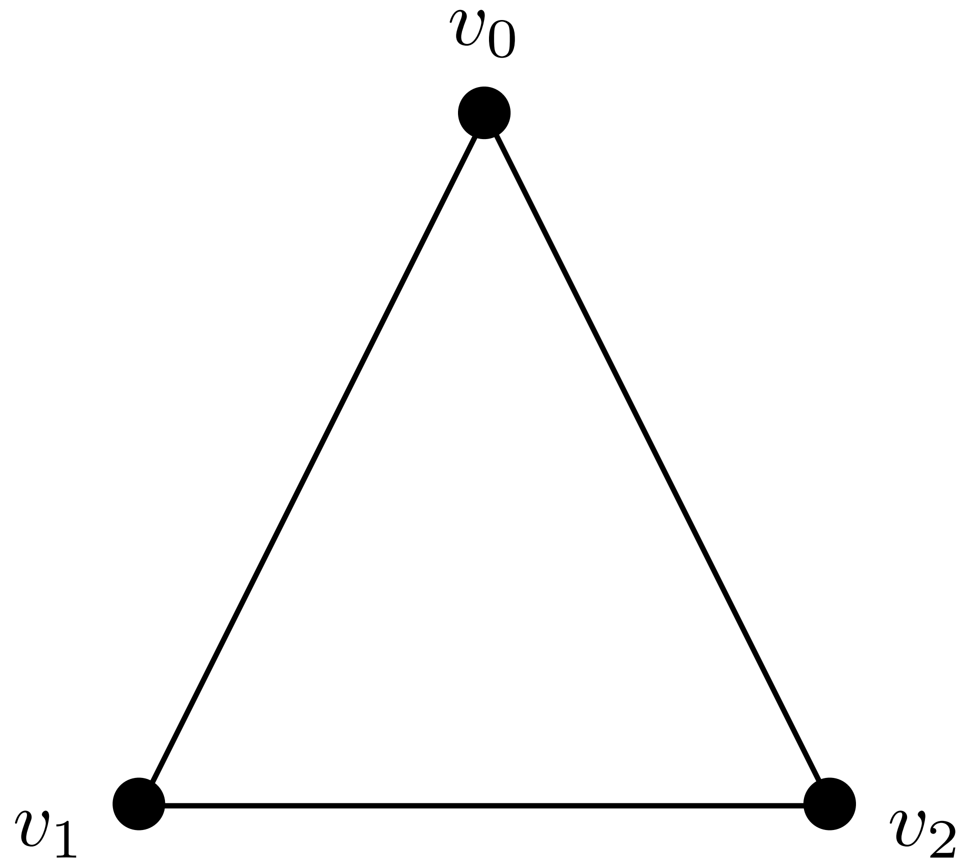
* Exchange values via Reliable Broadcast, and pick the 'smallest' one

Proof Sketch (contd.)

2 Run the Byzantine barycentric agreement protocol N times, landing on a simplex in $\text{Bary}^N \text{skel}^t(\mathcal{I}^*)$.

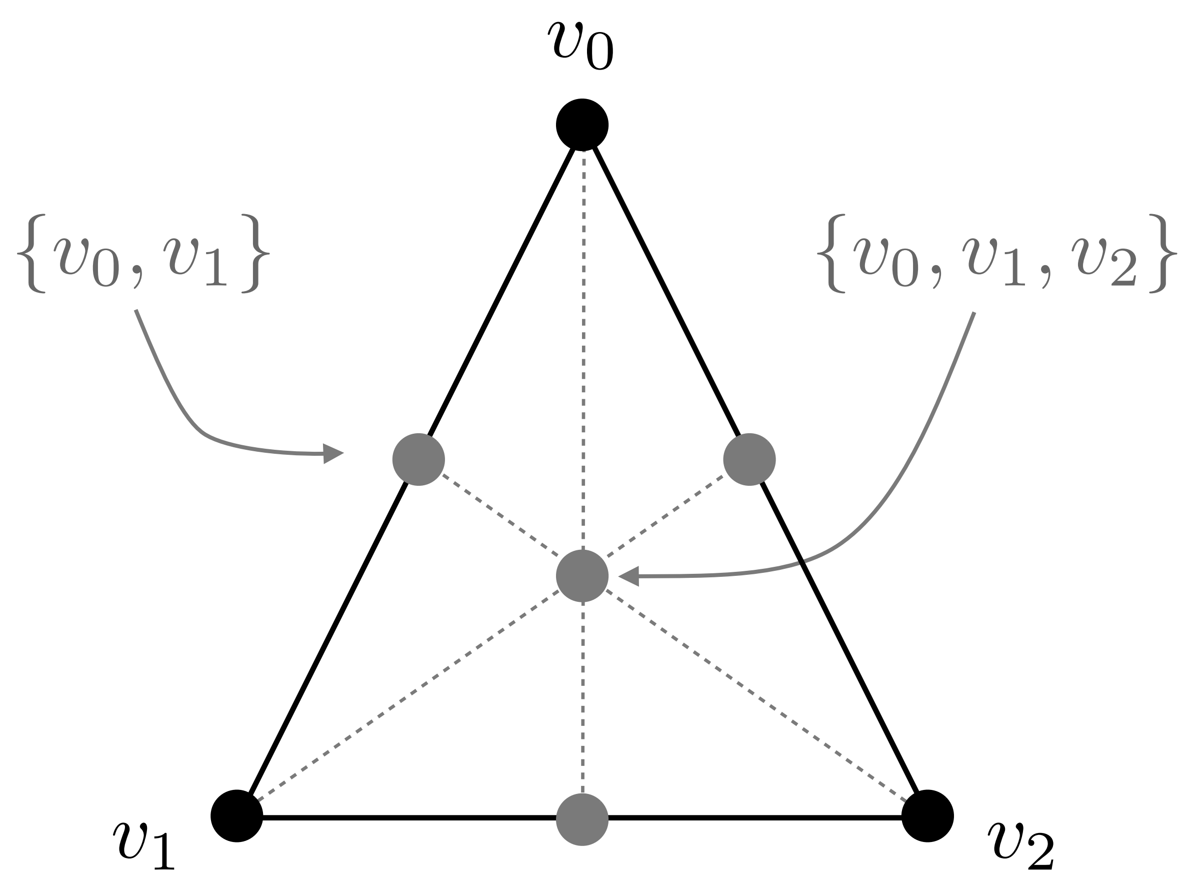
Proof Sketch (contd.)

2 Run the Byzantine barycentric agreement protocol N times, landing on a simplex in $\text{Bary}^N \text{skel}^t(\mathcal{I}^*)$.



Proof Sketch (contd.)

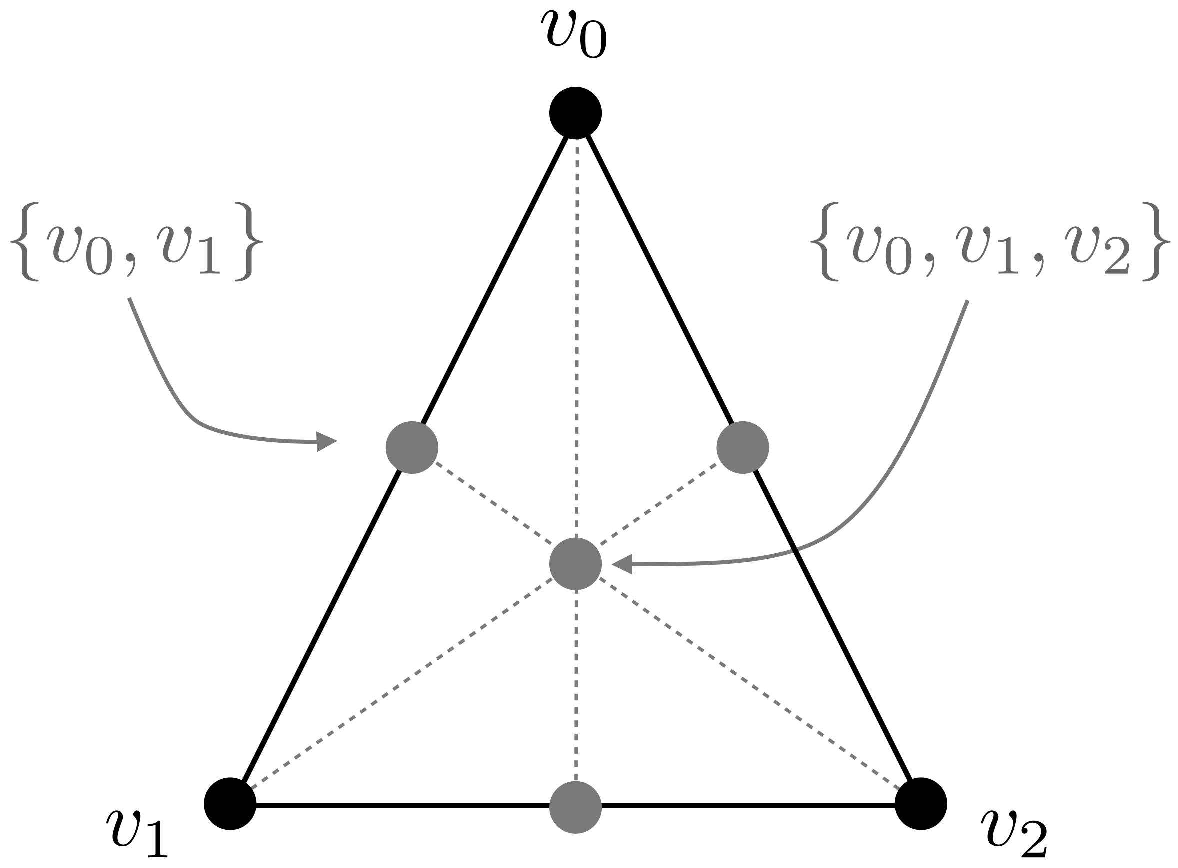
2 Run the Byzantine barycentric agreement protocol N times, landing on a simplex in $\text{Bary}^N \text{skel}^t(\mathcal{I}^*)$.



Proof Sketch (contd.)

2 Run the Byzantine barycentric agreement protocol N times, landing on a simplex in $\text{Bary}^N \text{skel}^t(\mathcal{I}^*)$.

Barycentric Agreement task

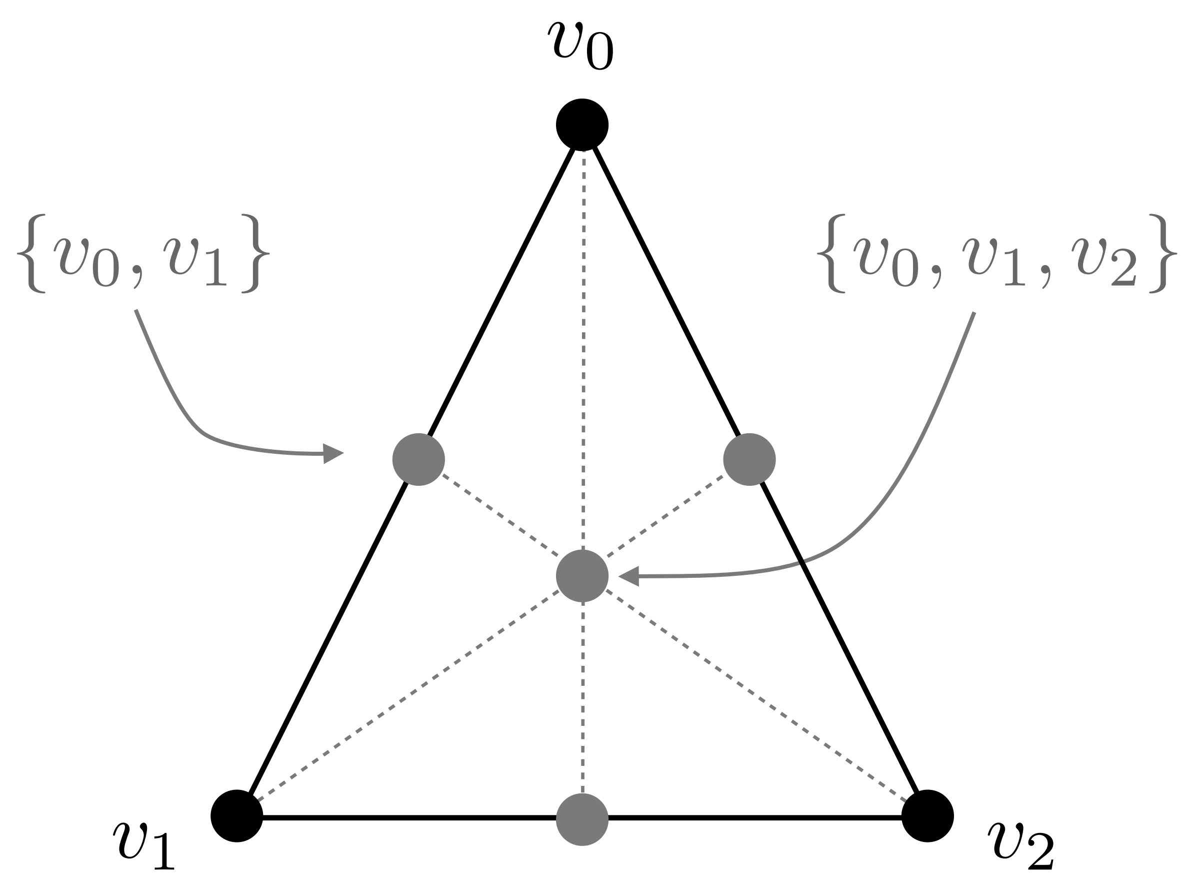


Proof Sketch (contd.)

2 Run the Byzantine barycentric agreement protocol N times, landing on a simplex in $\text{Bary}^N \text{skel}^t(\mathcal{I}^*)$.

Barycentric Agreement task

Agree on vertices of a single simplex of Bary σ

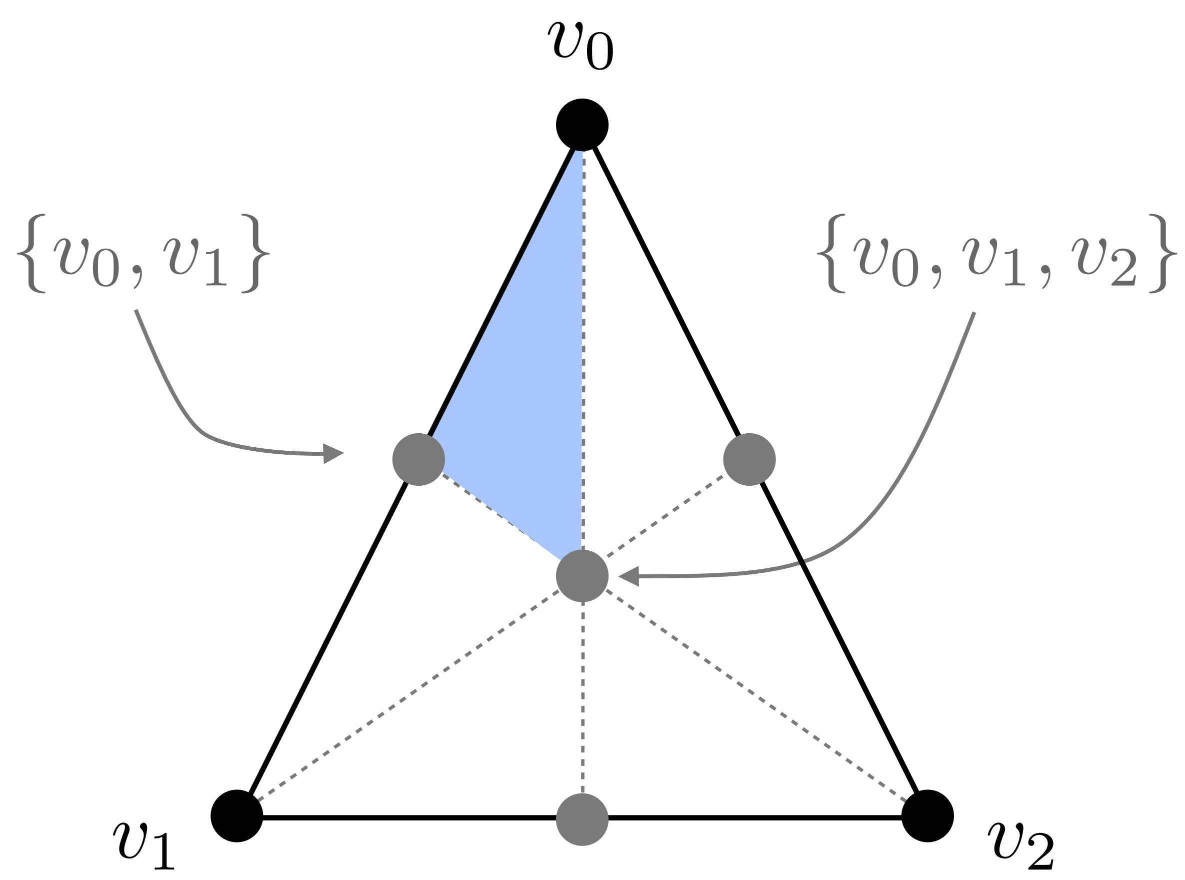


Proof Sketch (contd.)

2 Run the Byzantine barycentric agreement protocol N times, landing on a simplex in $\text{Bary}^N \text{skel}^t(\mathcal{I}^*)$.

Barycentric Agreement task

Agree on vertices of a single simplex of Bary σ

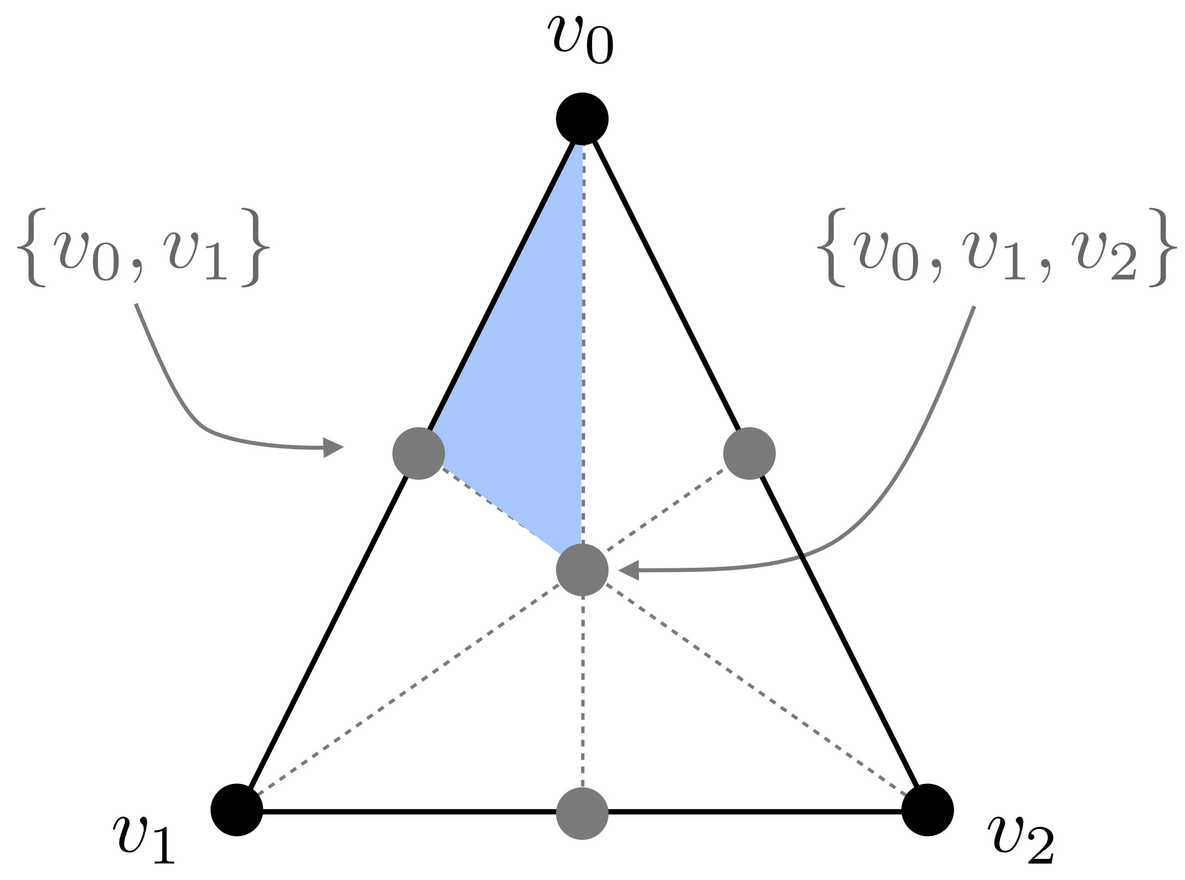


Proof Sketch (contd.)

2 Run the Byzantine barycentric agreement protocol N times, landing on a simplex in $\text{Bary}^N \text{skel}^t(\mathcal{I}^*)$.

Barycentric Agreement task

Agree on vertices of a single simplex of Bary σ



(protocol based on the ϵ-multidimensional agreement!) [STOC 13]

Proof Sketch (contd.)

Proof Sketch (contd.)

By the *Simplicial Approximation Theorem*, $f : |\,\mathsf{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ has a *simplicial approximation*

Proof Sketch (contd.)

hypothesis

By the *Simplicial Approximation Theorem*, $f : |\,\text{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ has a *simplicial approximation*

Proof Sketch (contd.)

hypothesis

By the *Simplicial Approximation Theorem,* $f : |\,\mathsf{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ has a *simplicial approximation*

$$\mathsf{Bary}^N\,\mathsf{skel}^t(\mathcal{I}^*) \to \mathcal{O}^* \text{ for some } N > 0.$$

Proof Sketch (contd.)

hypothesis

By the *Simplicial Approximation Theorem*, $f : |\,\text{skel}^t(\mathcal{I}^*)| \rightarrow |\mathcal{O}^*|$ has a *simplicial approximation*

$$\text{Bary}^N \text{skel}^t(\mathcal{I}^*) \rightarrow \mathcal{O}^* \text{ for some } N > 0.$$

fine-grain the input, so we can "approximate" f by a *simplicial* map

Proof Sketch (contd.)

hypothesis

By the *Simplicial Approximation Theorem*, $f : |\,\text{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ has a *simplicial approximation*

$$\text{Bary}^N \text{skel}^t(\mathcal{I}^*) \to \mathcal{O}^* \text{ for some } N > 0.$$

fine-grain the input, so we can "approximate" f by a *simplicial* map

We then...

3 Apply $\phi : \text{Bary}^N \text{skel}^t(\mathcal{I}^*) \to \mathcal{O}^*$ to choose vertices in $\mathcal{O}^*$.

Proof Sketch (contd.)

hypothesis

By the *Simplicial Approximation Theorem*, $f : |\,\text{skel}^t(\mathcal{I}^*)| \to |\mathcal{O}^*|$ has a *simplicial approximation*

$$\text{Bary}^N \text{skel}^t(\mathcal{I}^*) \to \mathcal{O}^* \text{for some } N > 0.$$

fine-grain the input, so we can "approximate" f by a *simplicial* map

We then...

3 Apply $\phi : \text{Bary}^N \text{ skel}^t(\mathcal{I}^*) \to \mathcal{O}^*$ to choose vertices in $\mathcal{O}^*$.

(because it's a simplicial approximation, choosing outputs based on the approximation is consistent with choosing outputs based on f)

Outline

1. Introduction
2. Asynchronous Byzantine Systems
3. Synchronous Byzantine Systems
4. Conclusion & Future Work

Outline

1. Introduction
2. Asynchronous Byzantine Systems
3. Synchronous Byzantine Systems
4. Conclusion & Future Work

Outline

1. Introduction
2. Asynchronous Byzantine Systems
3. Synchronous Byzantine Systems *quick overview*
4. Conclusion & Future Work

Protocol Complex

Protocol Complex

Before

Protocol Complex

Before

Tasks modeled as simplicial complexes

Protocol Complex

Before

Tasks modeled as simplicial complexes

Now

Protocol Complex

Before

Tasks modeled as simplicial complexes

Now

Executions also modeled as simplicial complexes

Protocol Complex

Before

Tasks modeled as simplicial complexes

Now

Executions also modeled as simplicial complexes

Global state evolving throughout the rounds

Protocol Complex

Before

Tasks modeled as simplicial complexes

Now

Executions also modeled as simplicial complexes

Global state evolving throughout the rounds

Protocol Complex

Protocol Complexes

Protocol Complexes

Consensus **task**

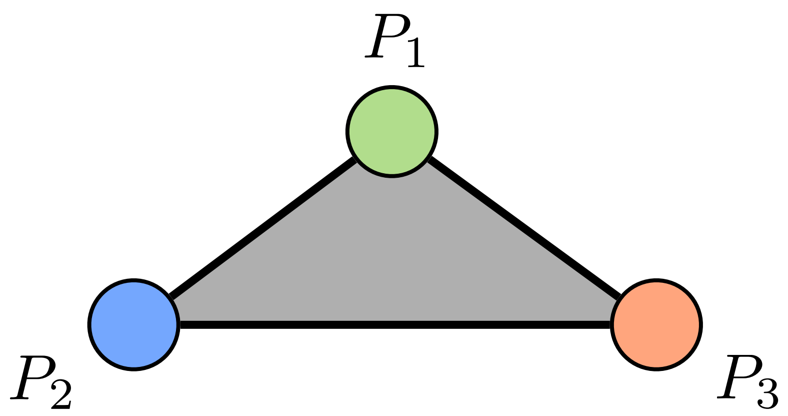
Protocol Complexes

Consensus **task** <u>t = 1</u>

Protocol Complexes

Consensus task t = 1

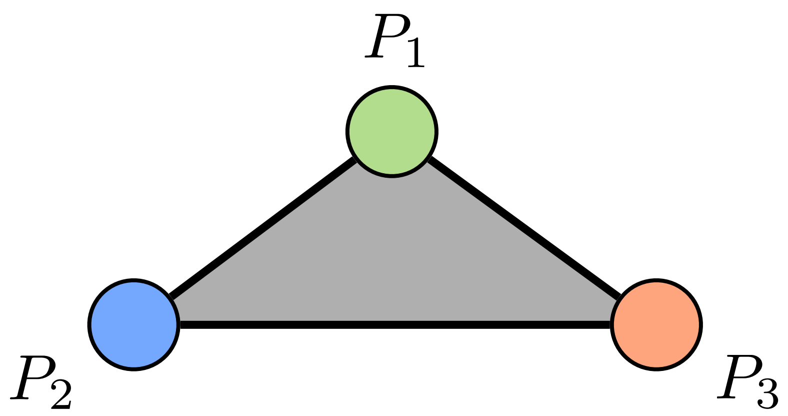
input



Protocol Complexes

Consensus task t = 1

input

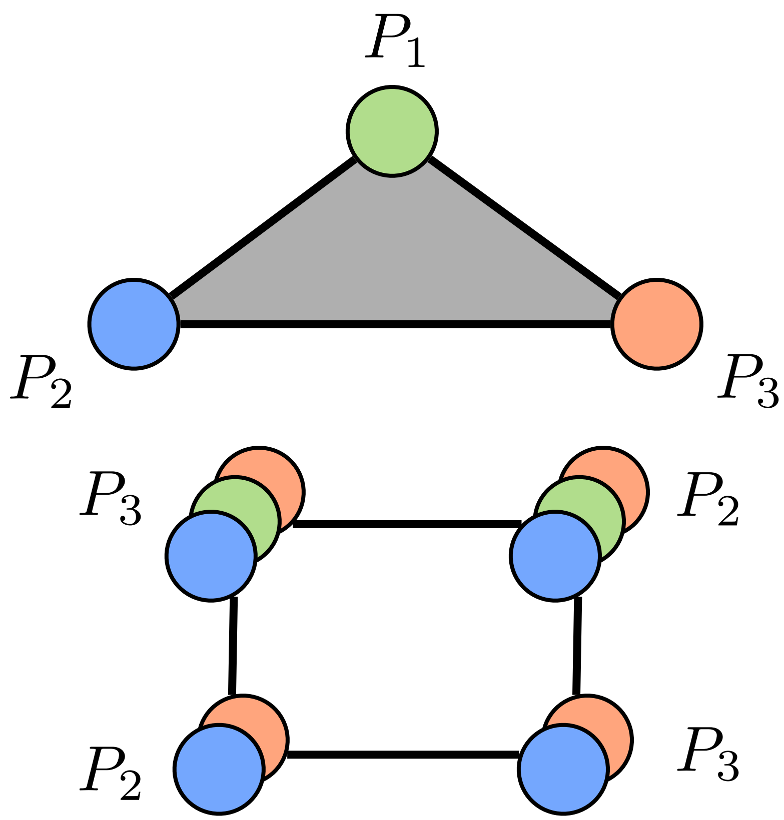


round 1:
P_1 fails

Protocol Complexes

Consensus task t = 1

input



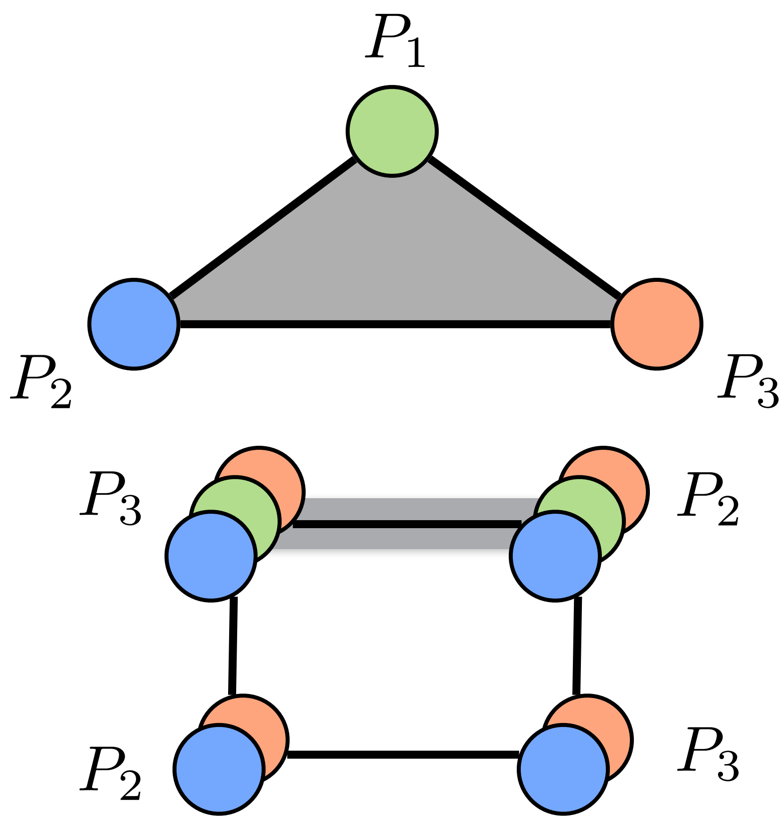
round 1:
P_1 fails

Protocol Complexes

Consensus task t = 1

input

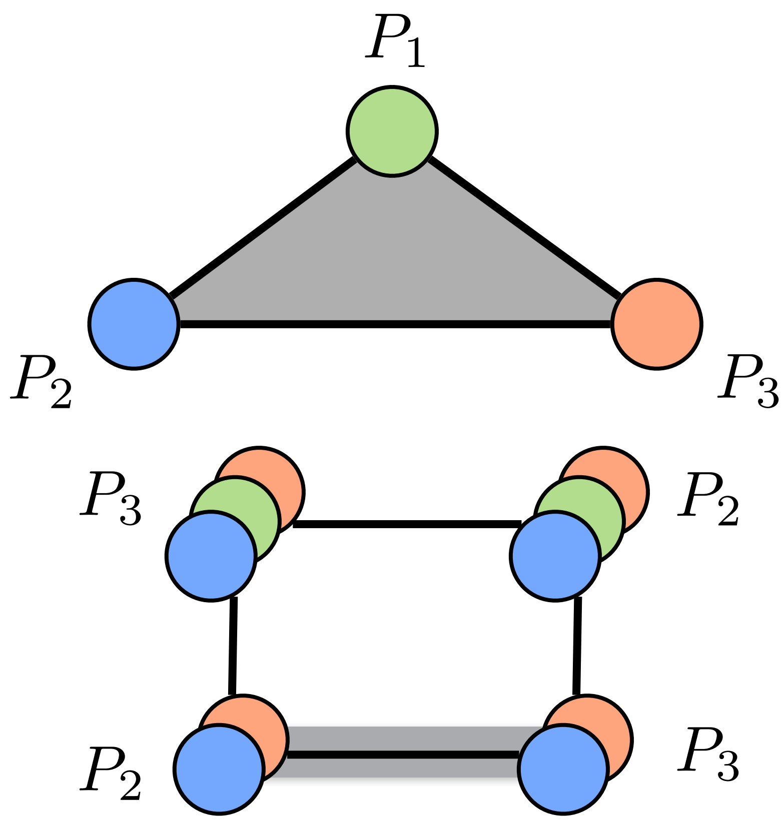
round 1:
P_1 fails



Protocol Complexes

Consensus task t = 1

input

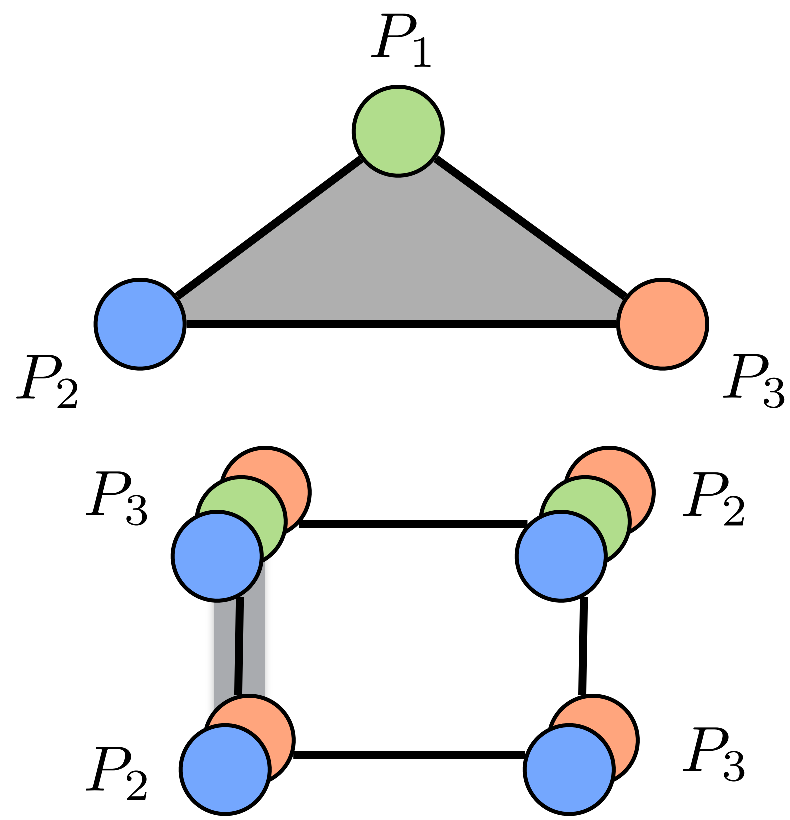


round 1: P_1 fails

Protocol Complexes

Consensus task t = 1

input

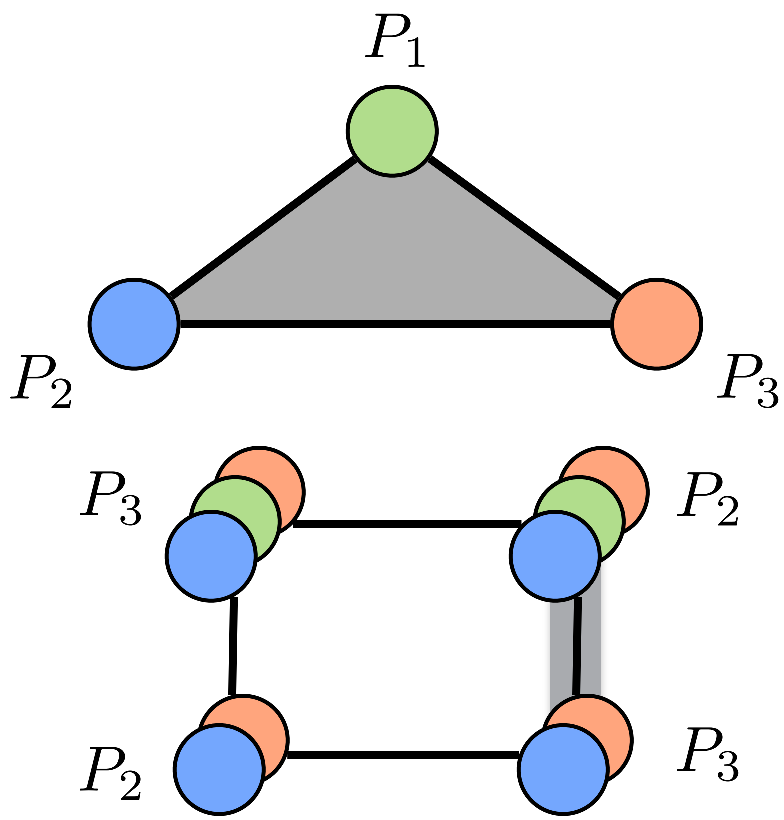


round 1:
P_1 fails

Protocol Complexes

Consensus task t = 1

input



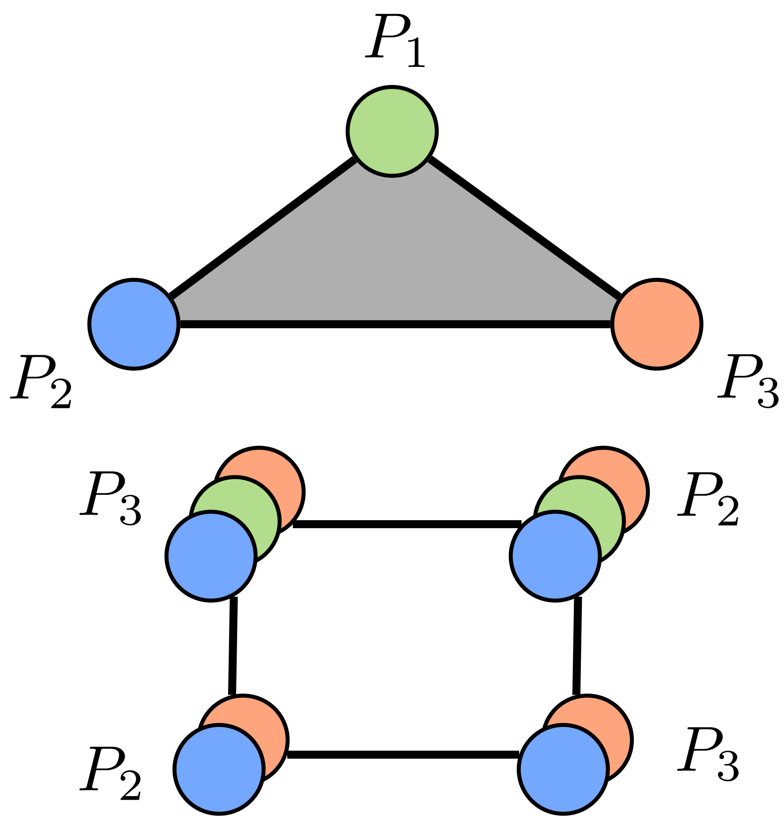
round 1:
P_1 fails

Protocol Complexes

Consensus task t = 1

input

round 1:
P_1 fails

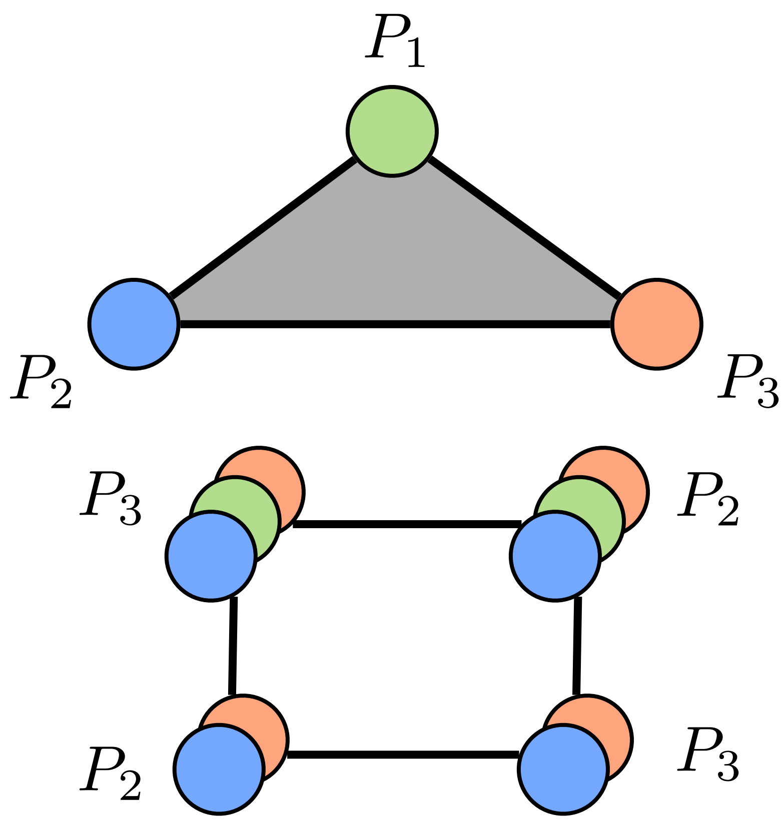


Protocol Complexes

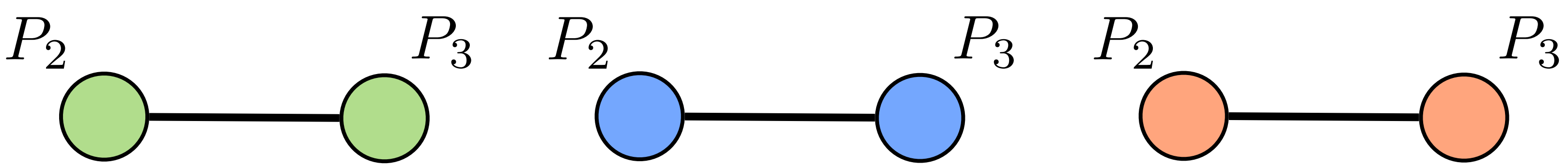
Consensus task t = 1

input

round 1: P_1 fails



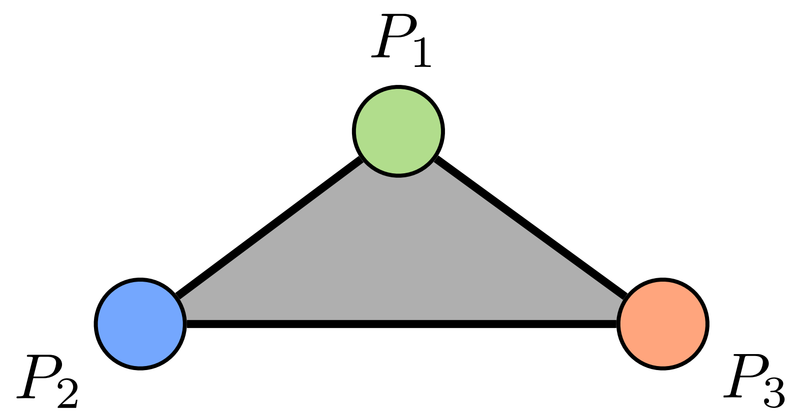
output



Protocol Complexes

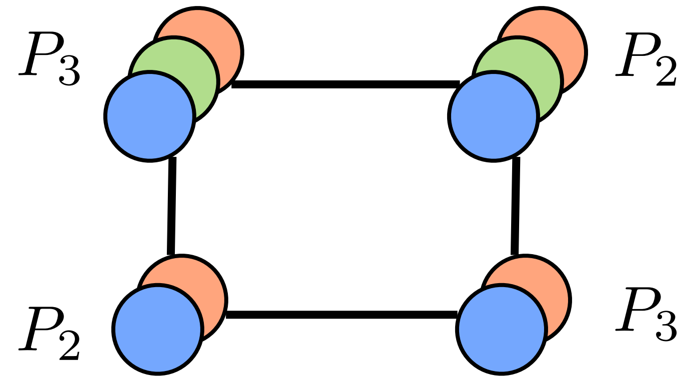
Consensus task t = 1

input

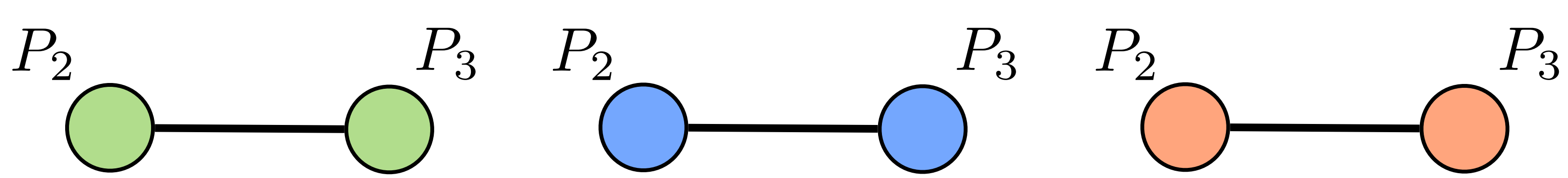


connected

round 1:
P_1 fails



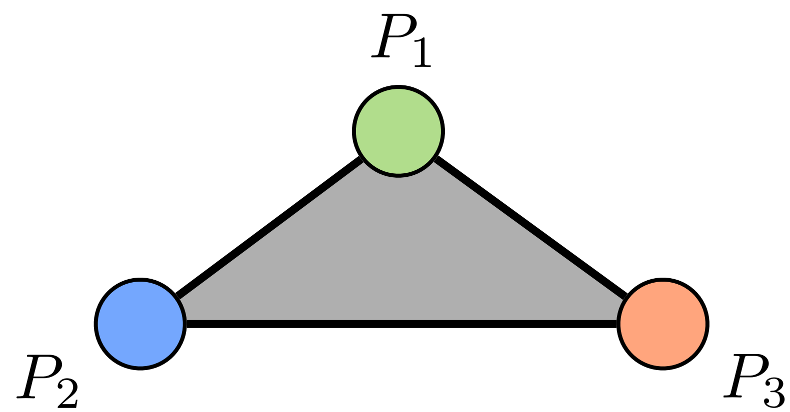
output



Protocol Complexes

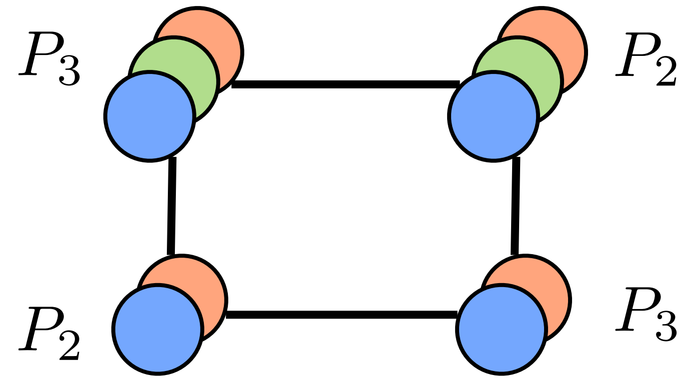
Consensus task t = 1

input



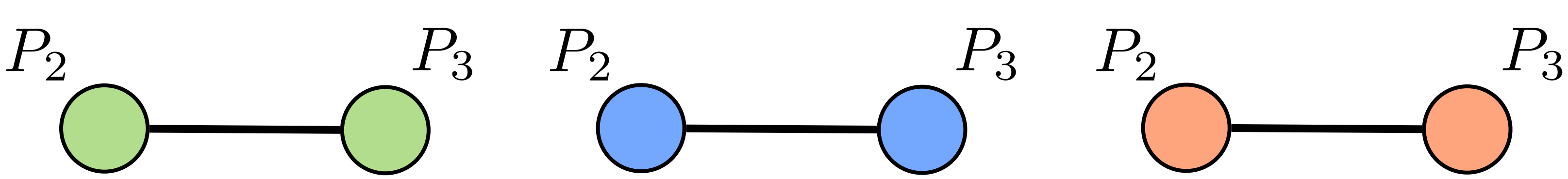
connected

round 1:
P_1 fails



connected

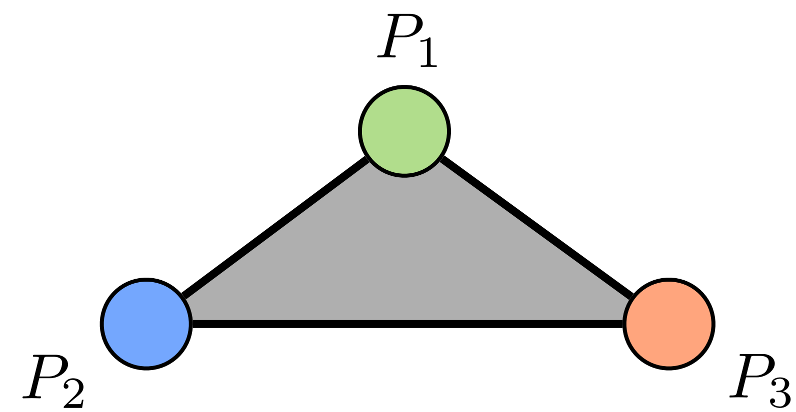
output



Protocol Complexes

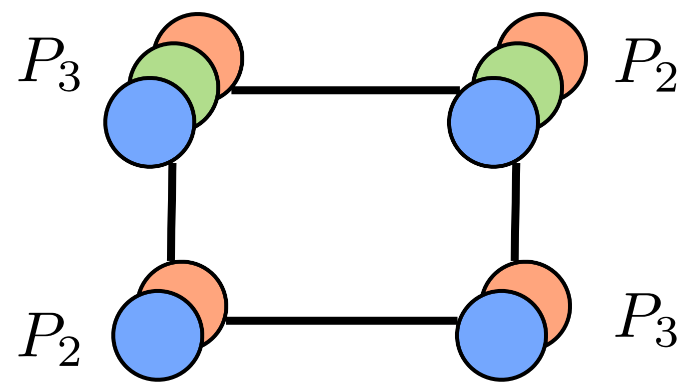
Consensus task t = 1

input



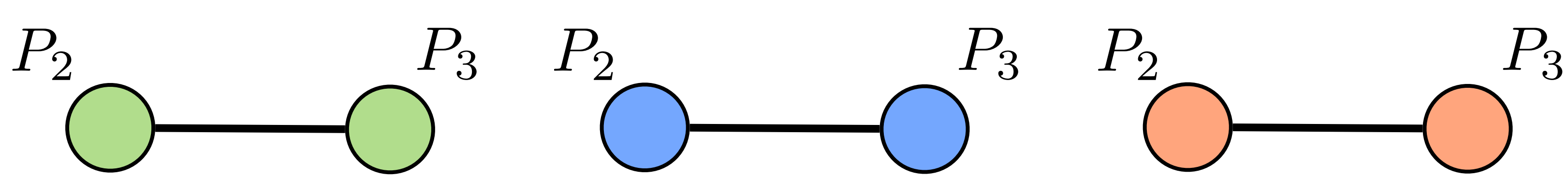
connected

round 1:
P_1 fails



connected

output



disconnected

Protocol Complexes

Consensus task <u>t = 1</u>

input

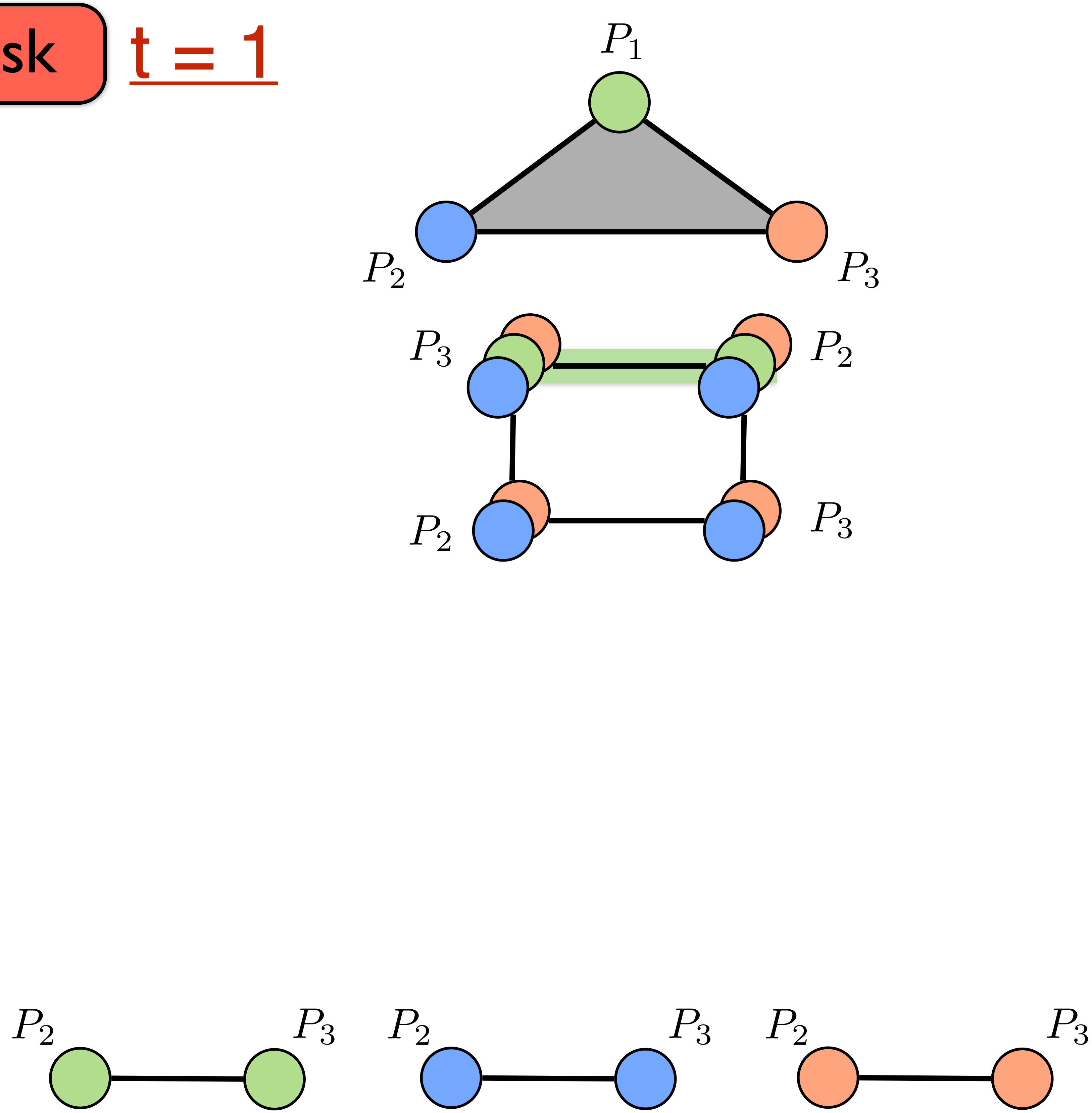
connected

round 1: P_1 fails 

connected

output

disconnected

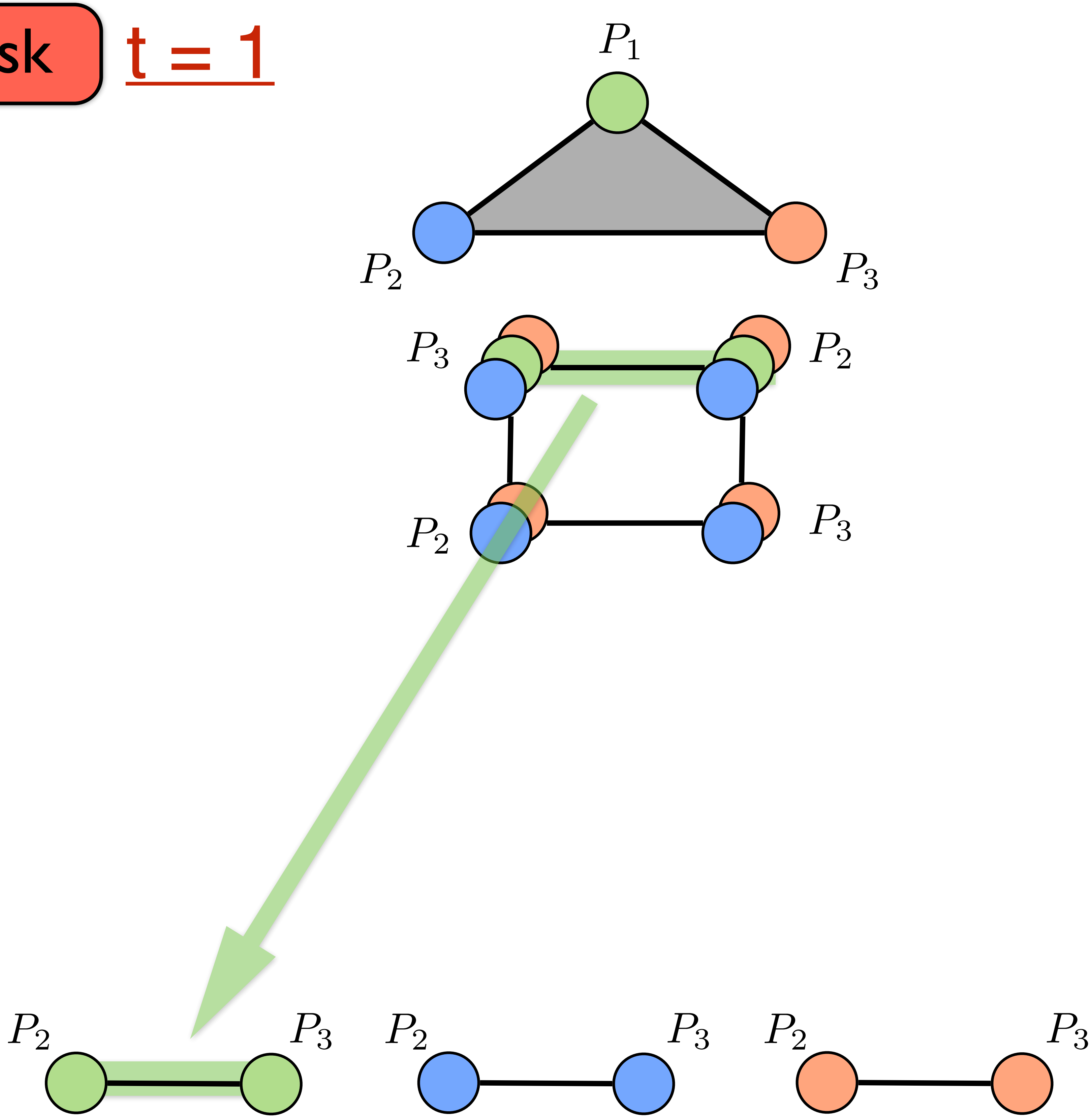


Protocol Complexes

Consensus task t = 1

input

connected



round 1:
P_1 fails

connected

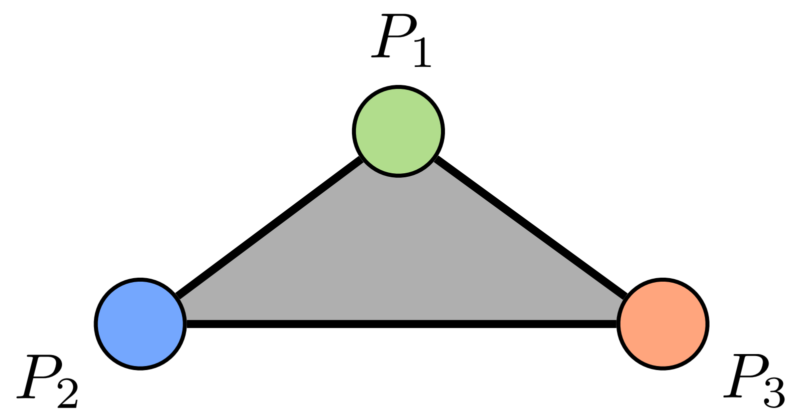
output

disconnected

Protocol Complexes

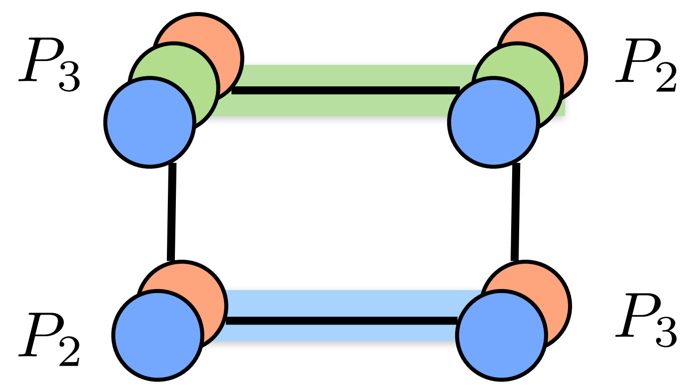
Consensus task t = 1

input



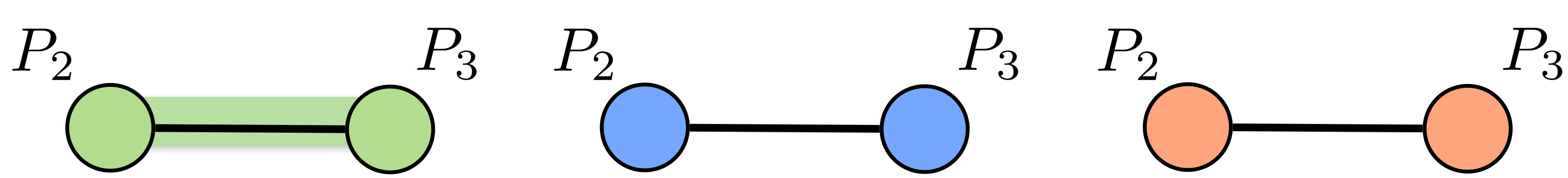
connected

round 1: P_1 fails



connected

output



disconnected

Protocol Complexes

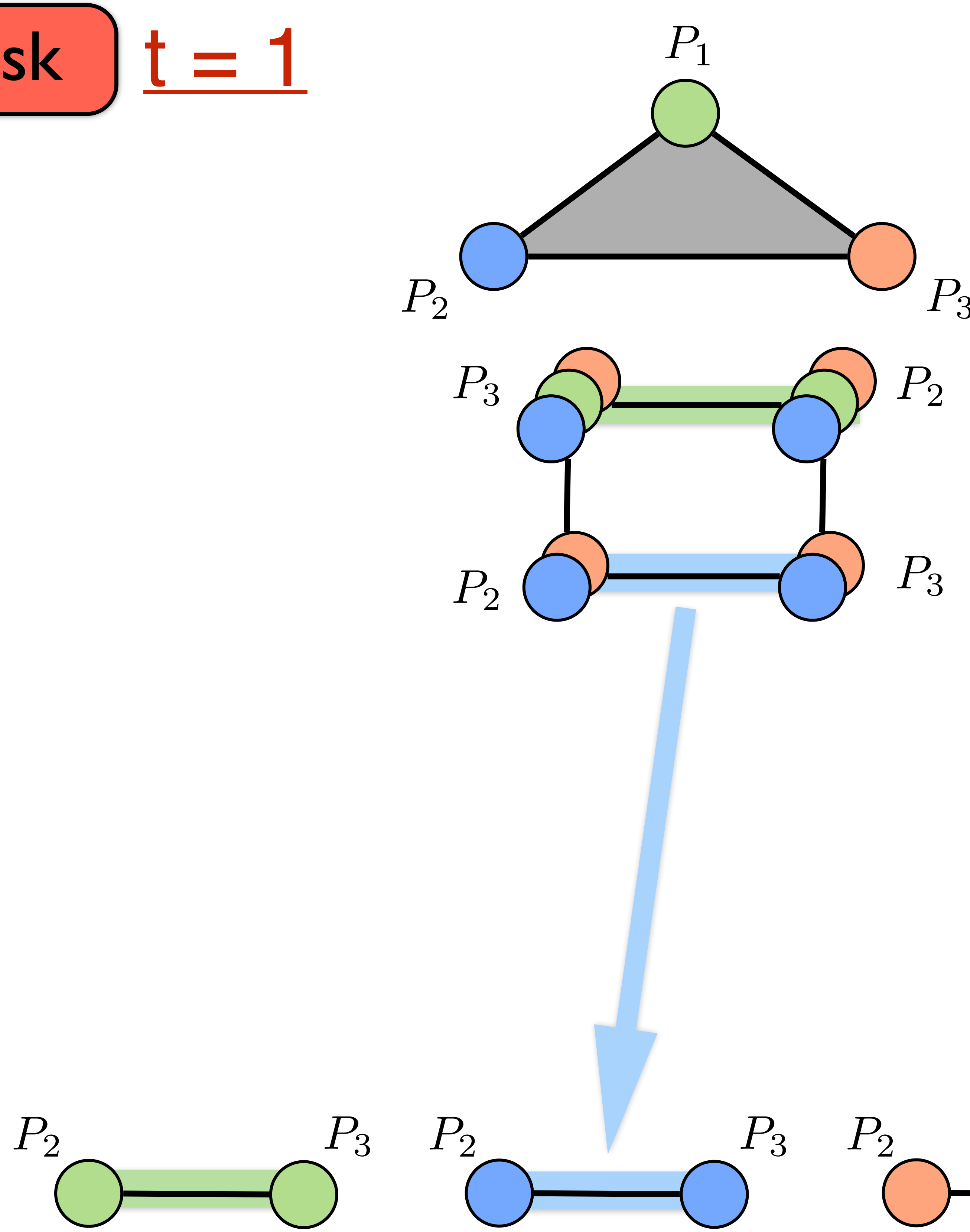
Consensus task t = 1

input

connected

round 1:
P_1 fails

connected



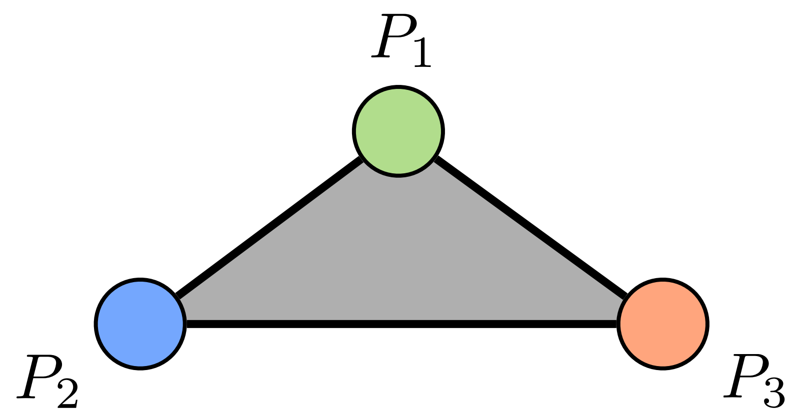
output

disconnected

Protocol Complexes

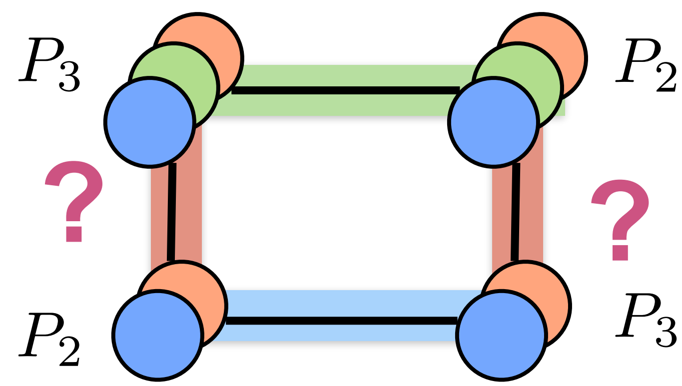
Consensus task t = 1

input



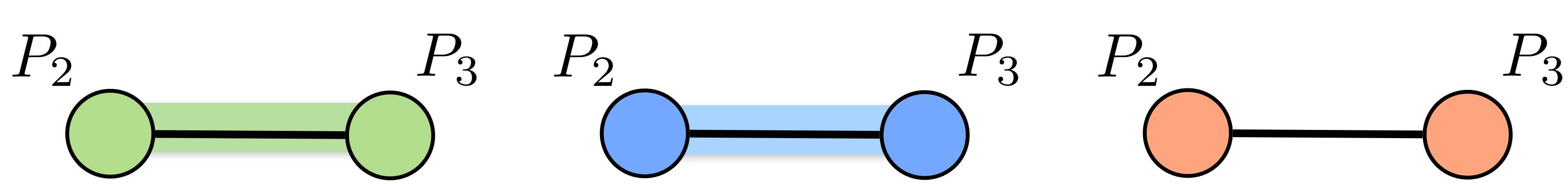
connected

round 1:
P_1 fails



connected

output

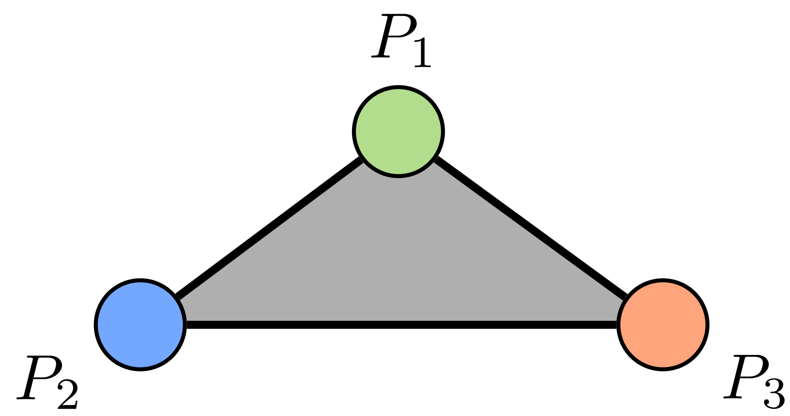


disconnected

Protocol Complexes

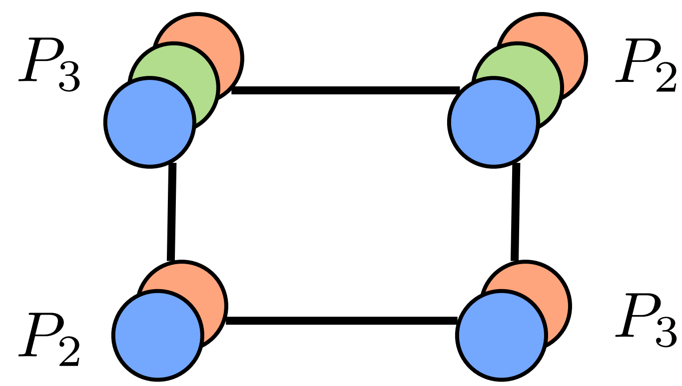
Consensus task t = 1

input



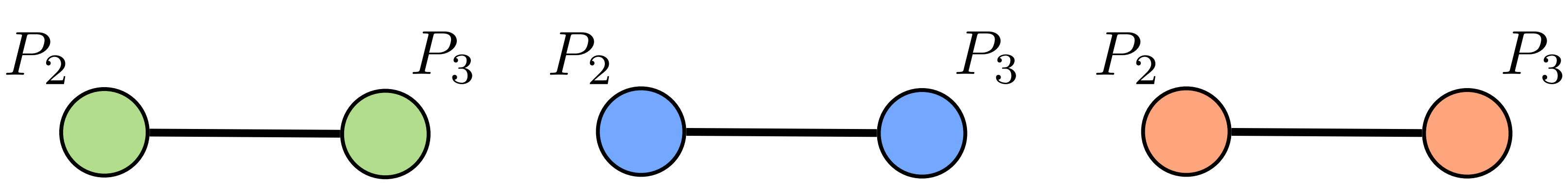
connected

round 1:
P_1 fails



connected

output

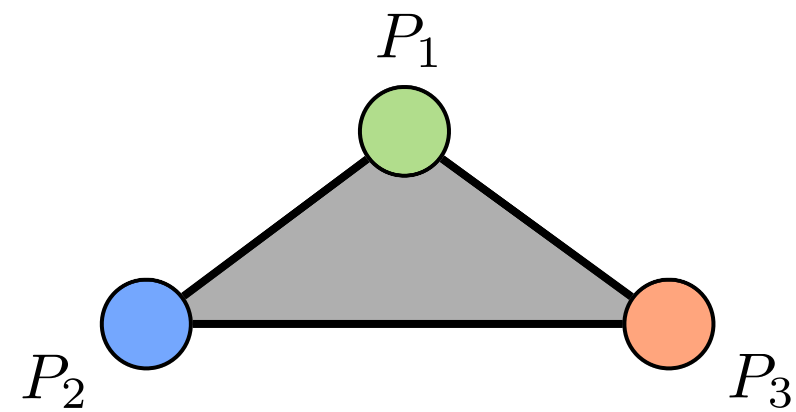


disconnected

Protocol Complexes

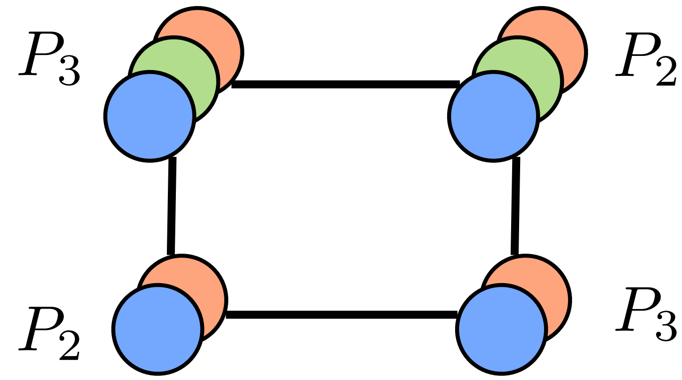
Consensus task t = 1

input



connected

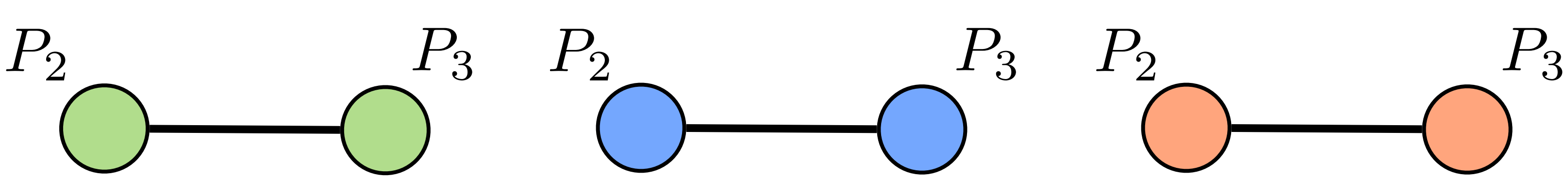
round 1:
P_1 fails



connected

round 2:
no failures

output

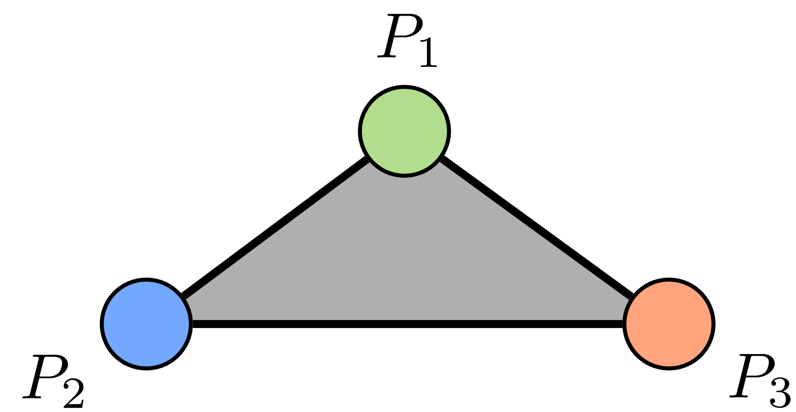


disconnected

Protocol Complexes

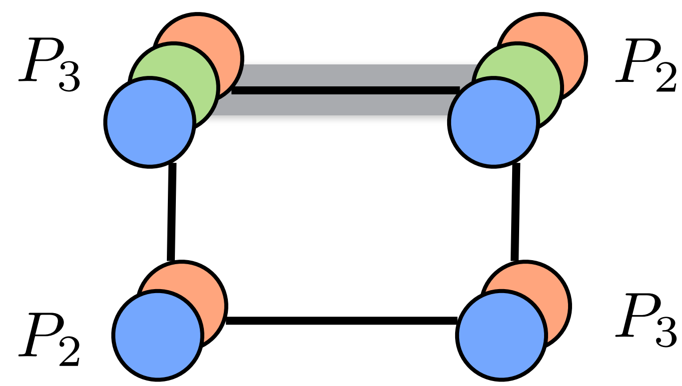
Consensus task t = 1

input



connected

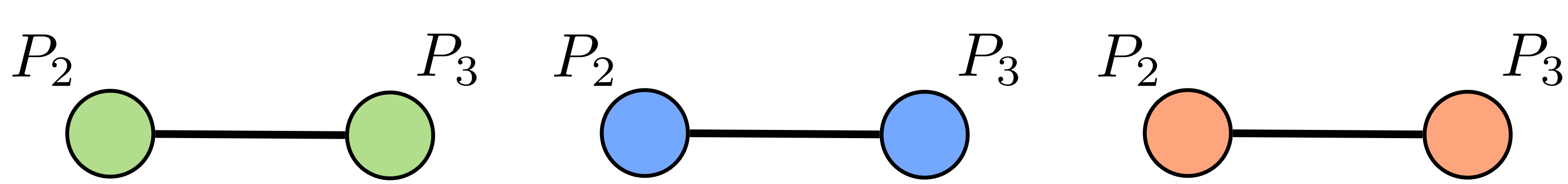
round 1:
P_1 fails



connected

round 2:
no failures

output

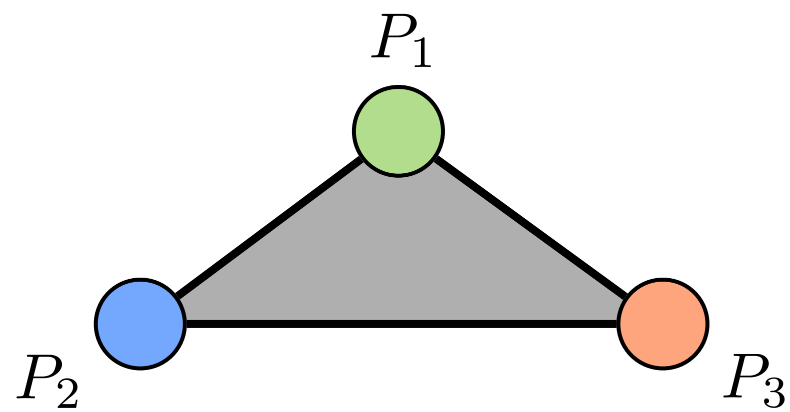


disconnected

Protocol Complexes

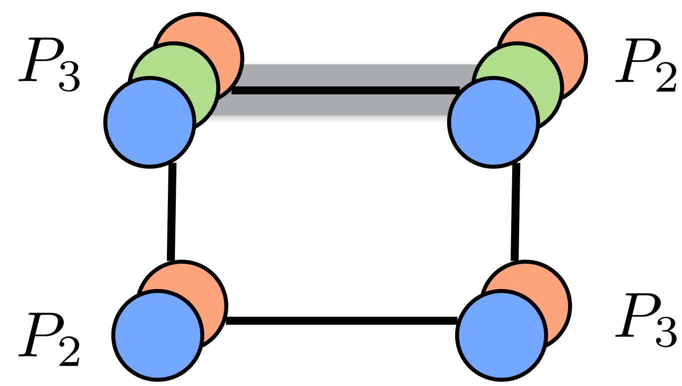
Consensus task <u>t = 1</u>

input



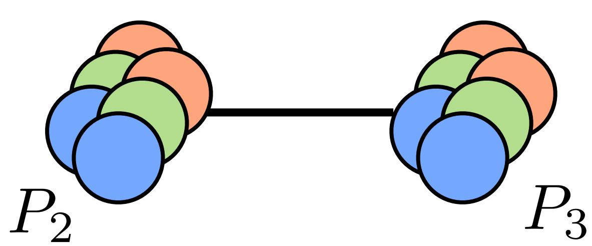
connected

round 1:
P_1 fails

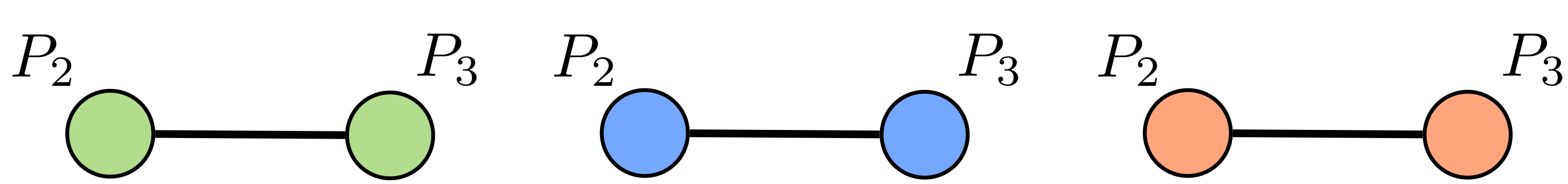


connected

round 2:
no failures



output

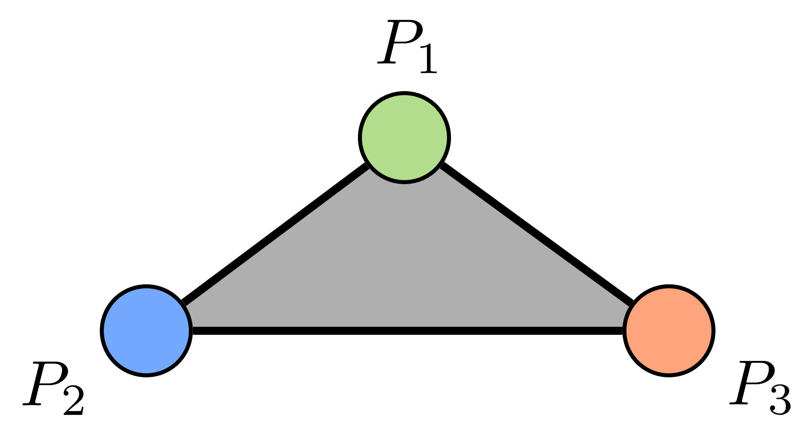


disconnected

Protocol Complexes

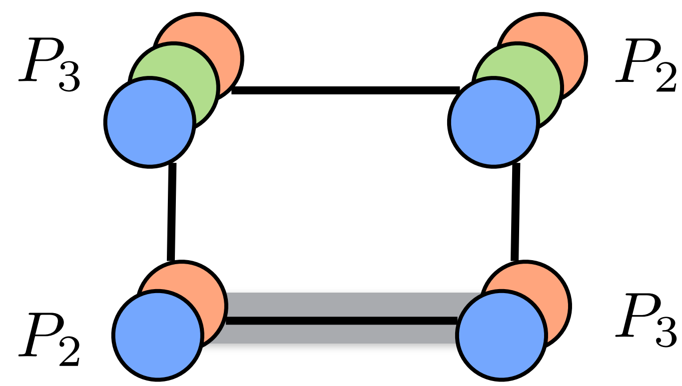
Consensus task t = 1

input



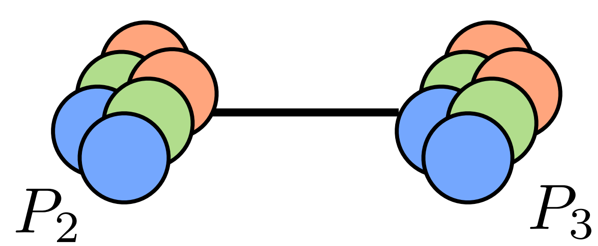
connected

round 1: P_1 fails

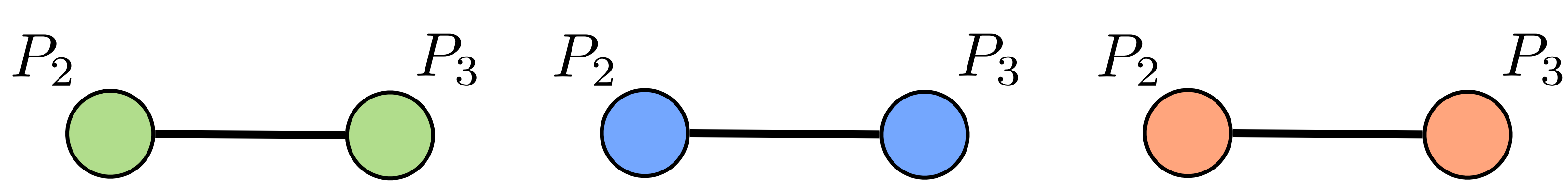


connected

round 2: no failures



output

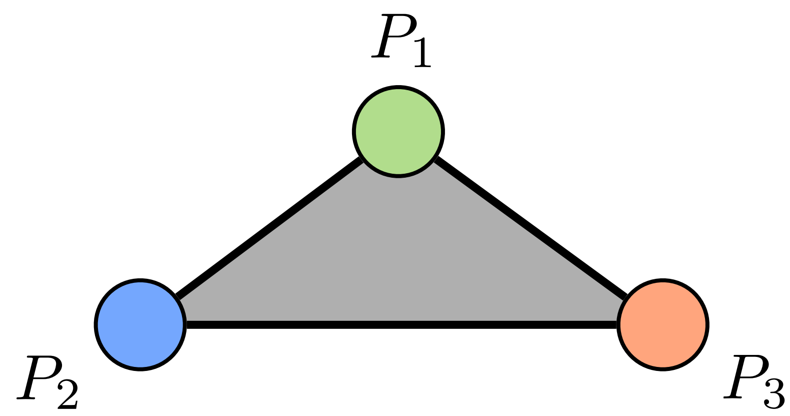


disconnected

Protocol Complexes

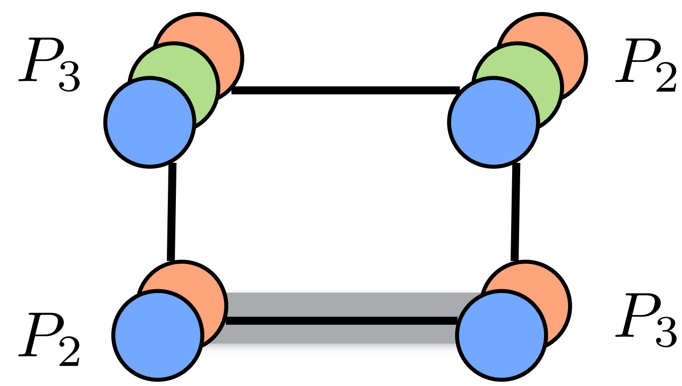
Consensus task t = 1

input



connected

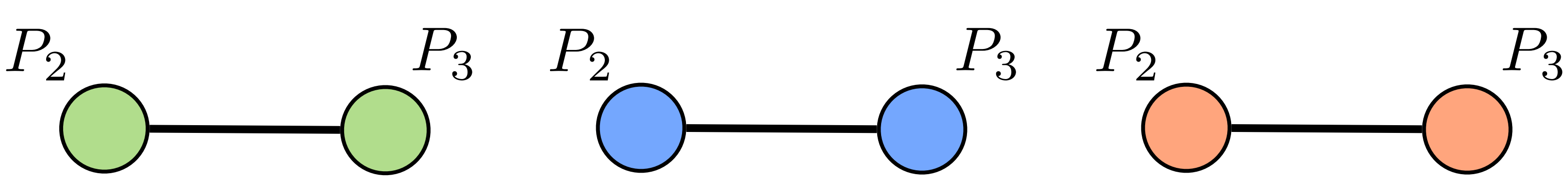
round 1:
P_1 fails



connected

round 2:
no failures

output



disconnected

Protocol Complexes

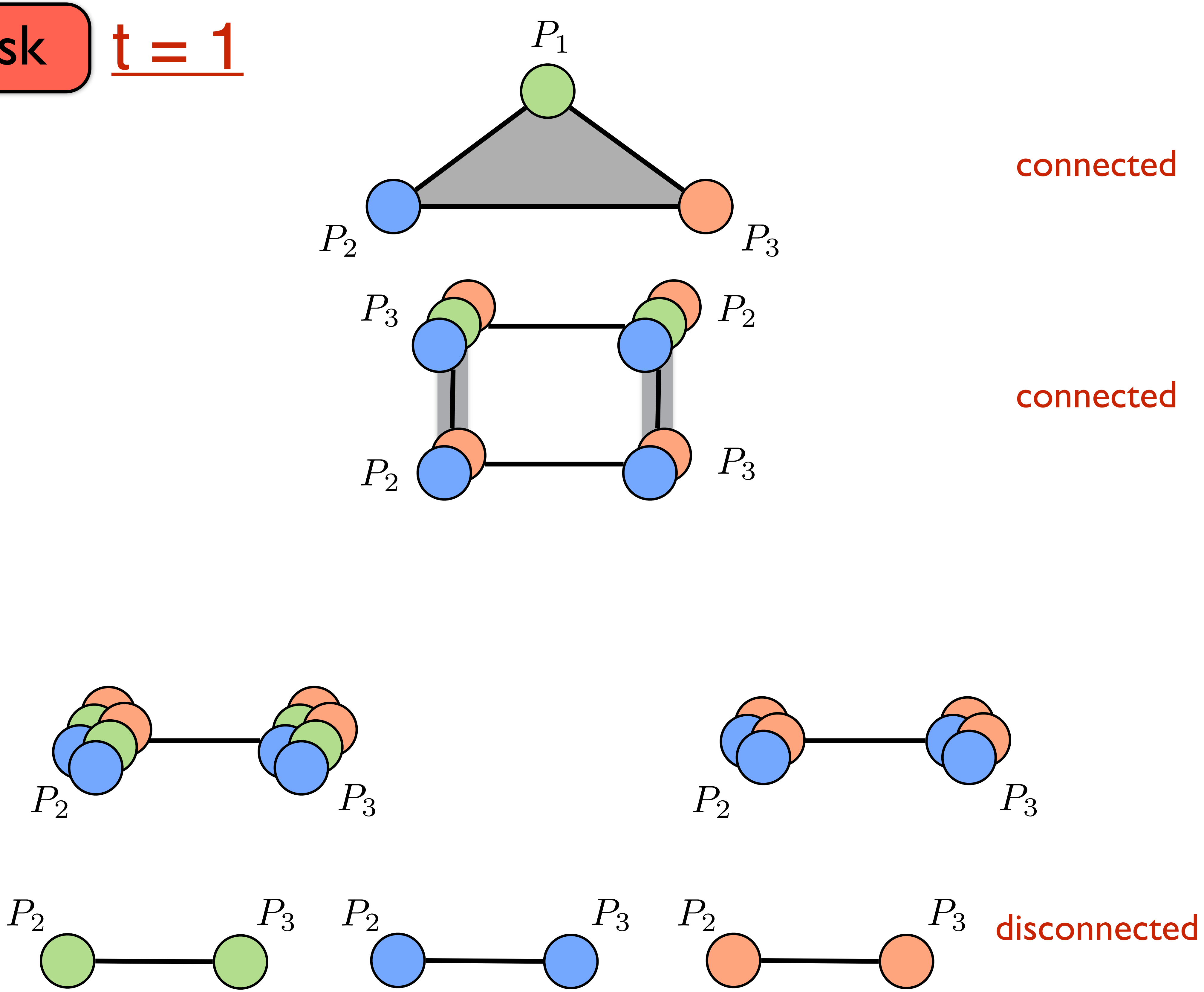
Consensus task t = 1

input

round 1:
P_1 fails

round 2:
no failures

output



Protocol Complexes

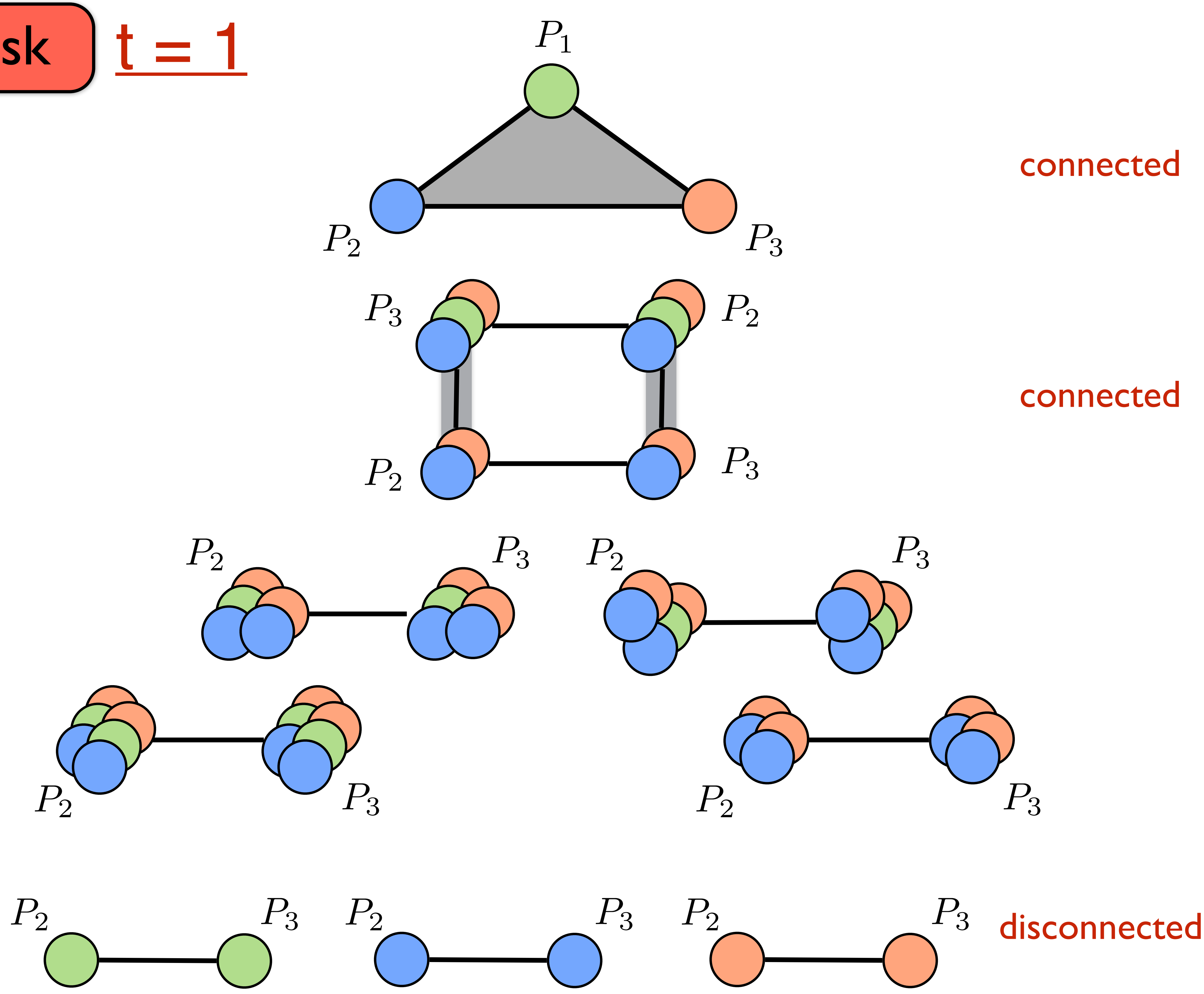
Consensus task t = 1

input

round 1: P_1 fails

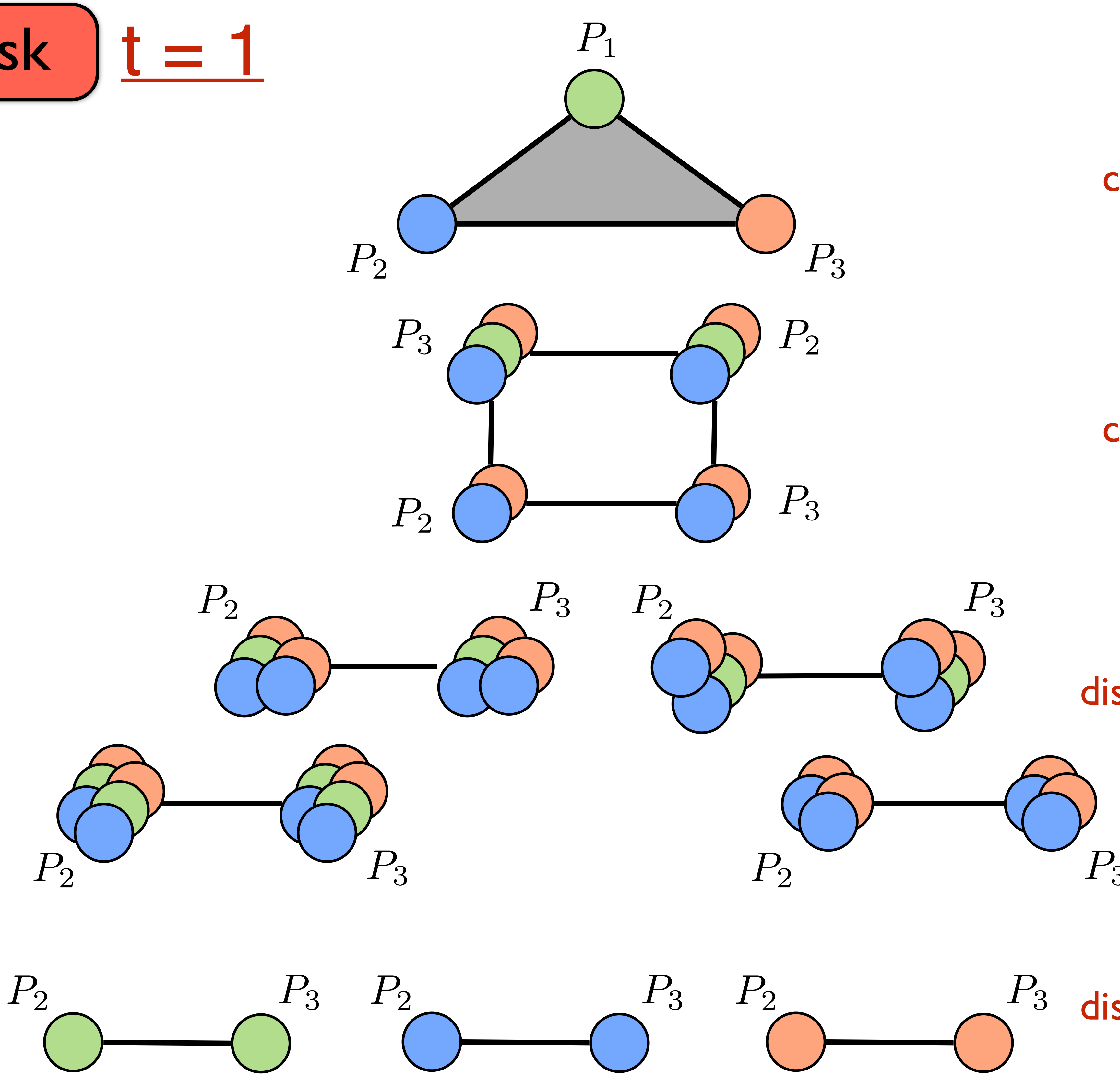
round 2: no failures

output



Protocol Complexes

Consensus task t = 1



input — connected

round 1: P_1 fails

connected

round 2: no failures — disconnected

output — disconnected

Protocol Complexes

Consensus task t = 1

input

connected

round 1: P_1 fails

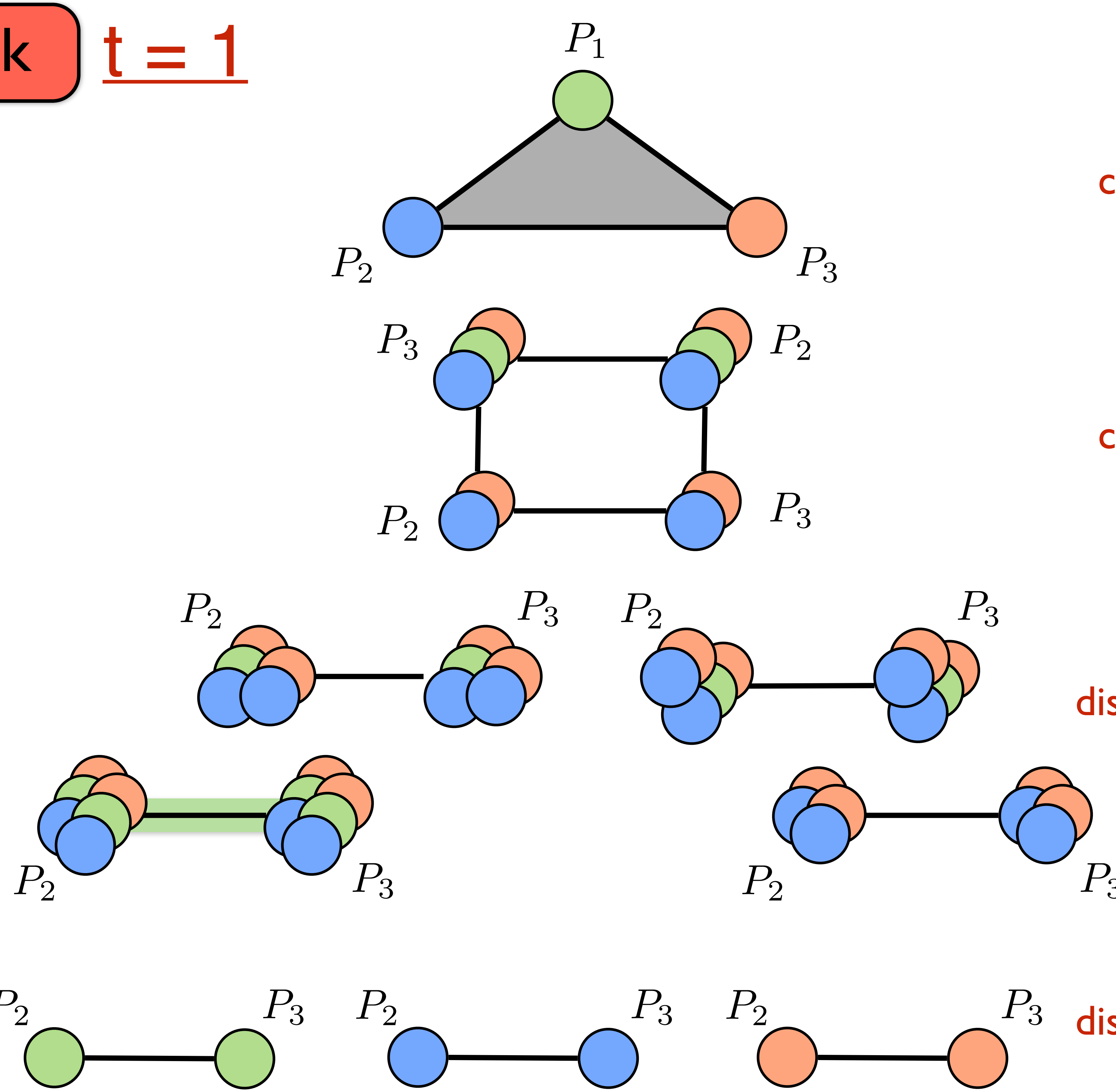
connected

round 2: no failures

disconnected

output

disconnected



Protocol Complexes

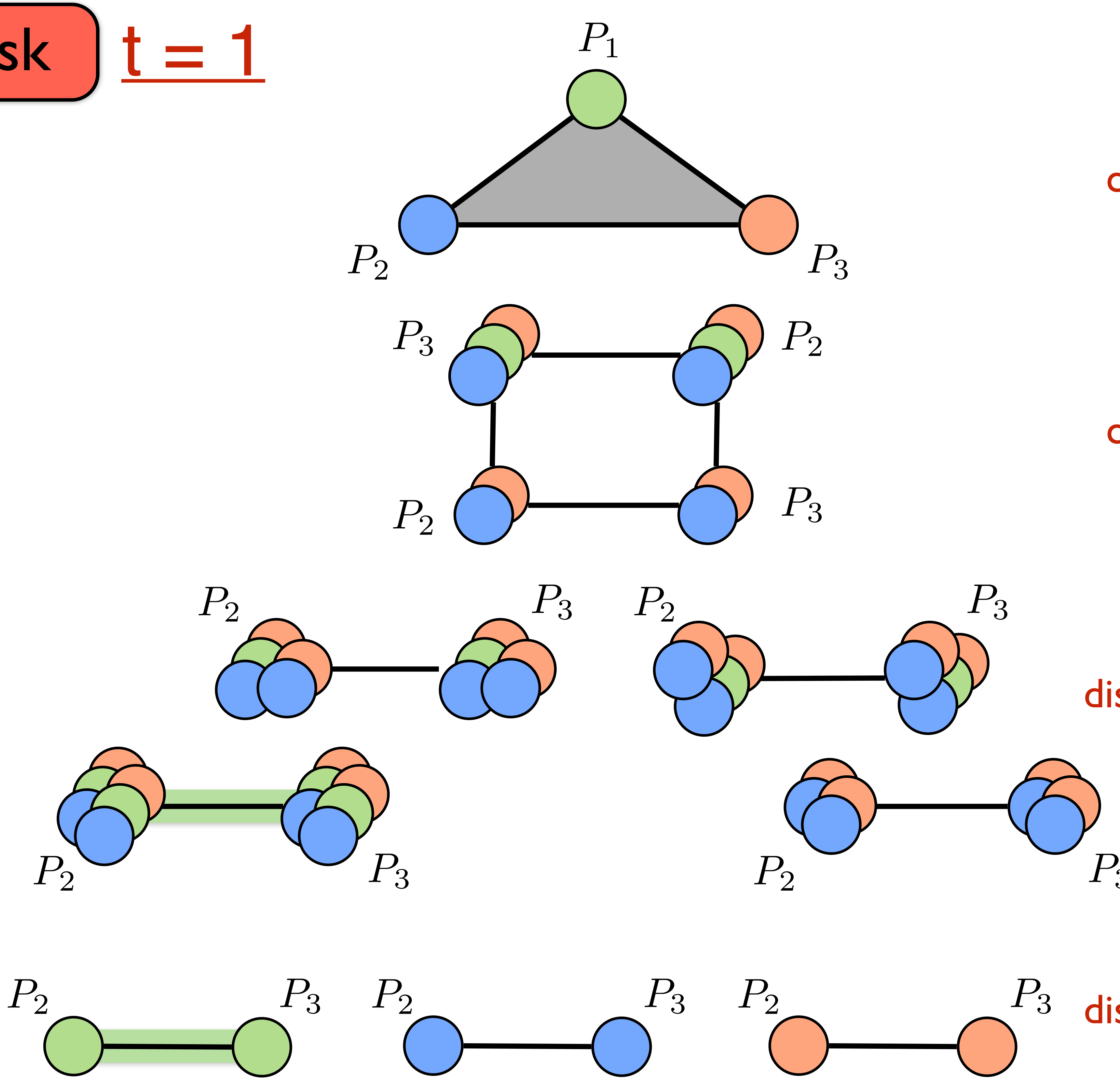
Consensus task t = 1

input

round 1: P_1 fails

round 2: no failures

output



connected

connected

disconnected

disconnected

Protocol Complexes

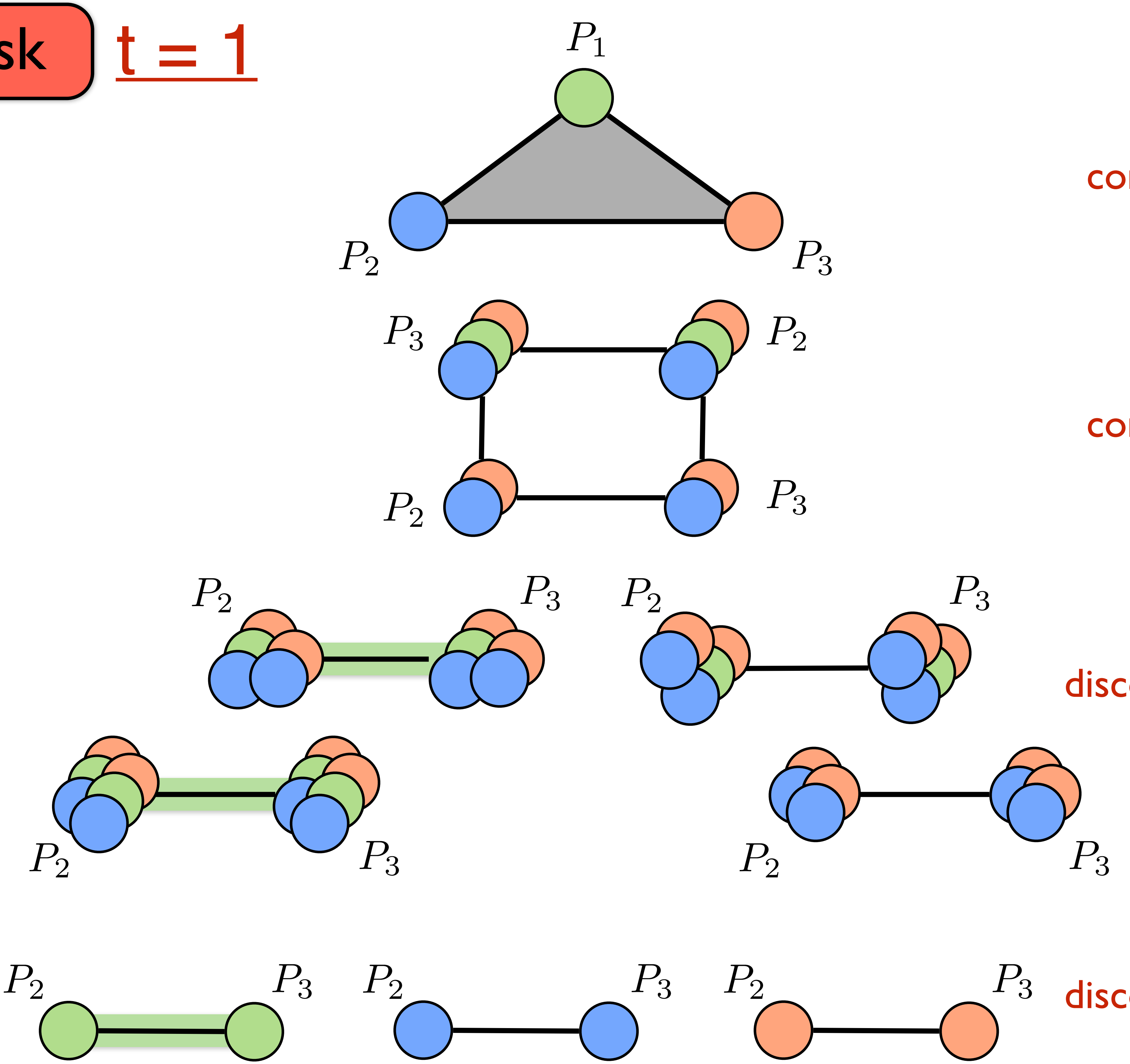
Consensus task t = 1

input — connected

round 1: P_1 fails  — connected

round 2: no failures — disconnected

output — disconnected



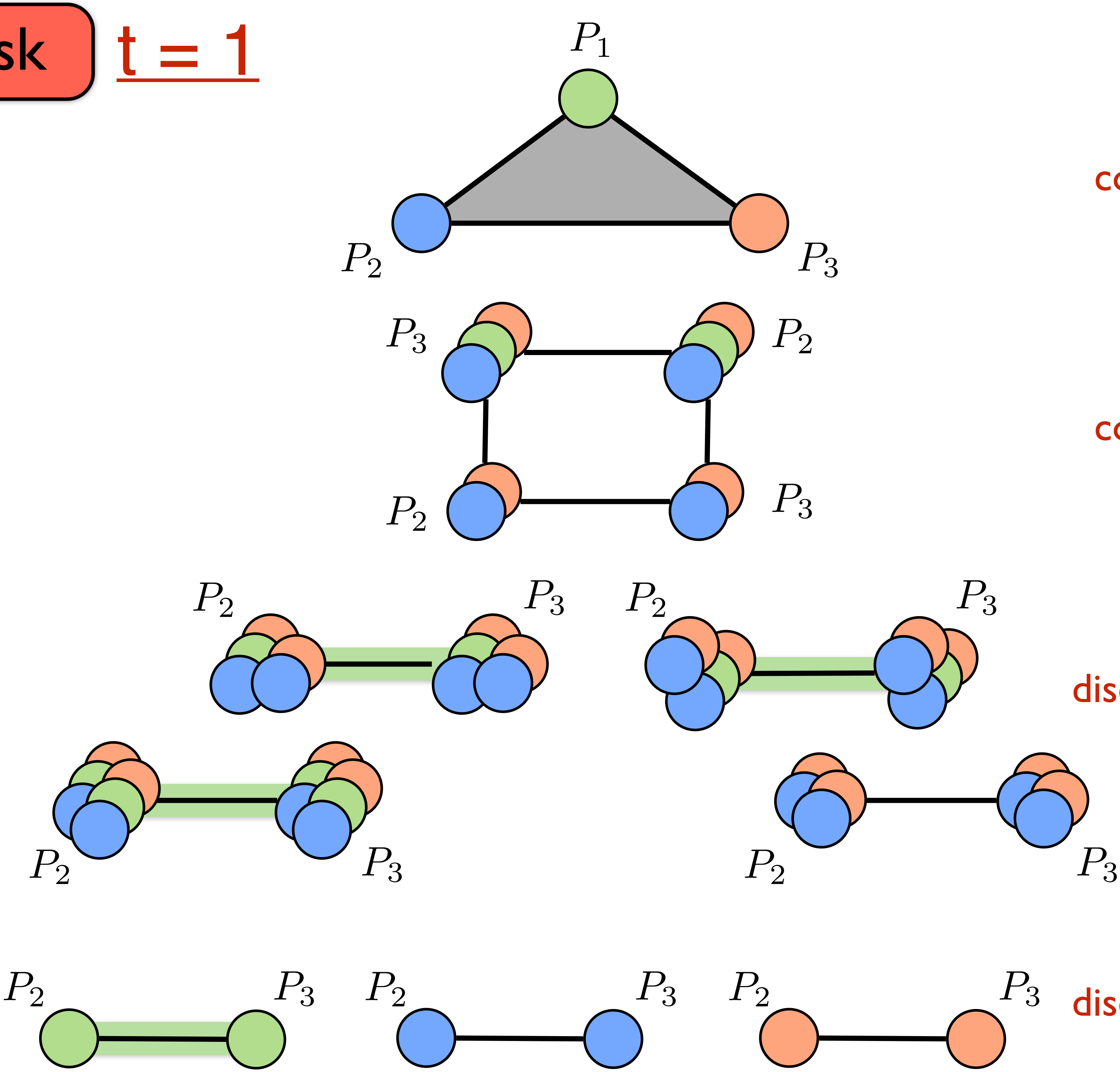
Protocol Complexes

Consensus task t = 1

input

connected

round 1:
P_1 fails



connected

round 2:
no failures

disconnected

output

disconnected

Protocol Complexes

Consensus task t = 1

input

connected

round 1:
P_1 fails

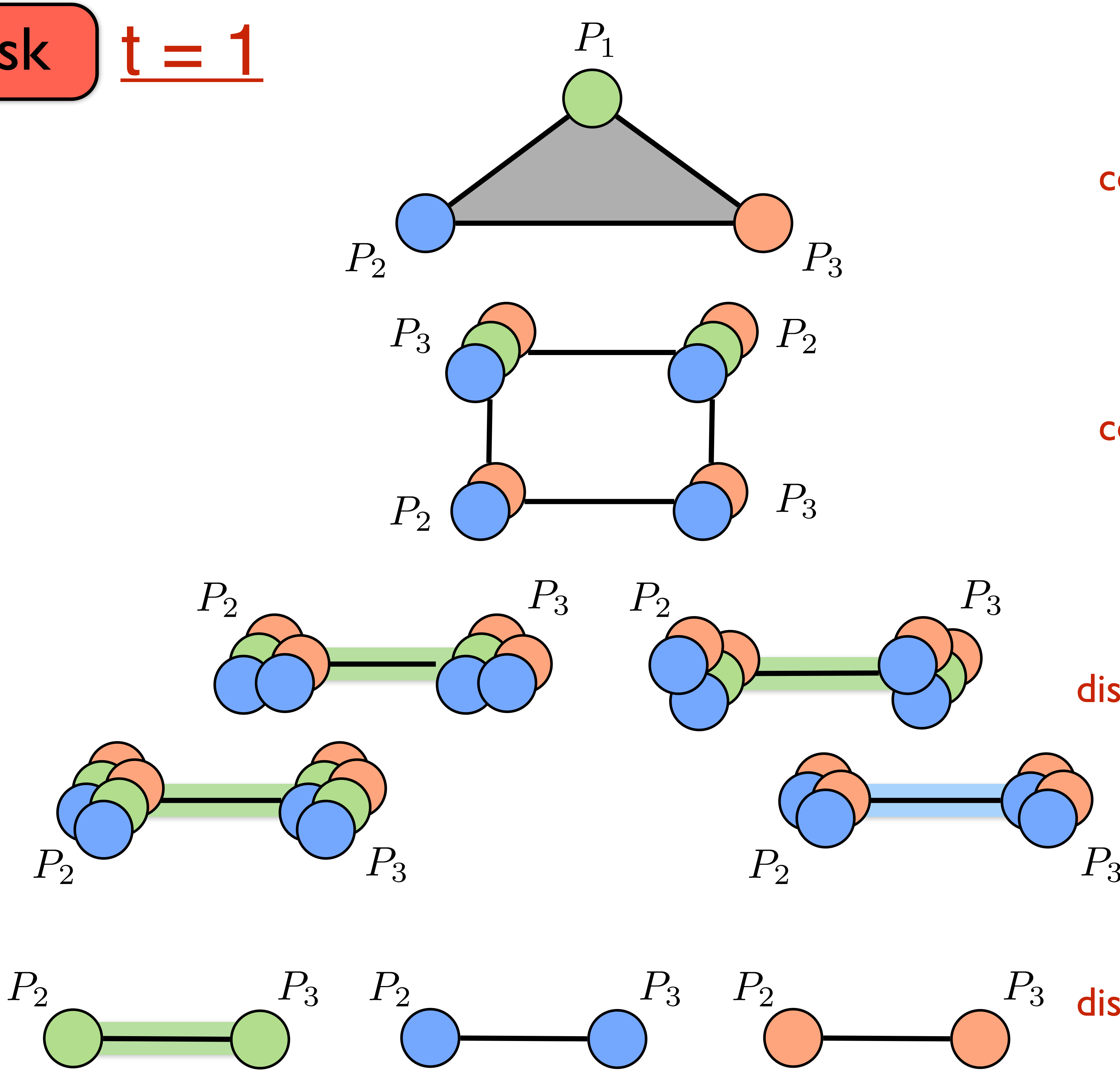
connected

round 2:
no failures

disconnected

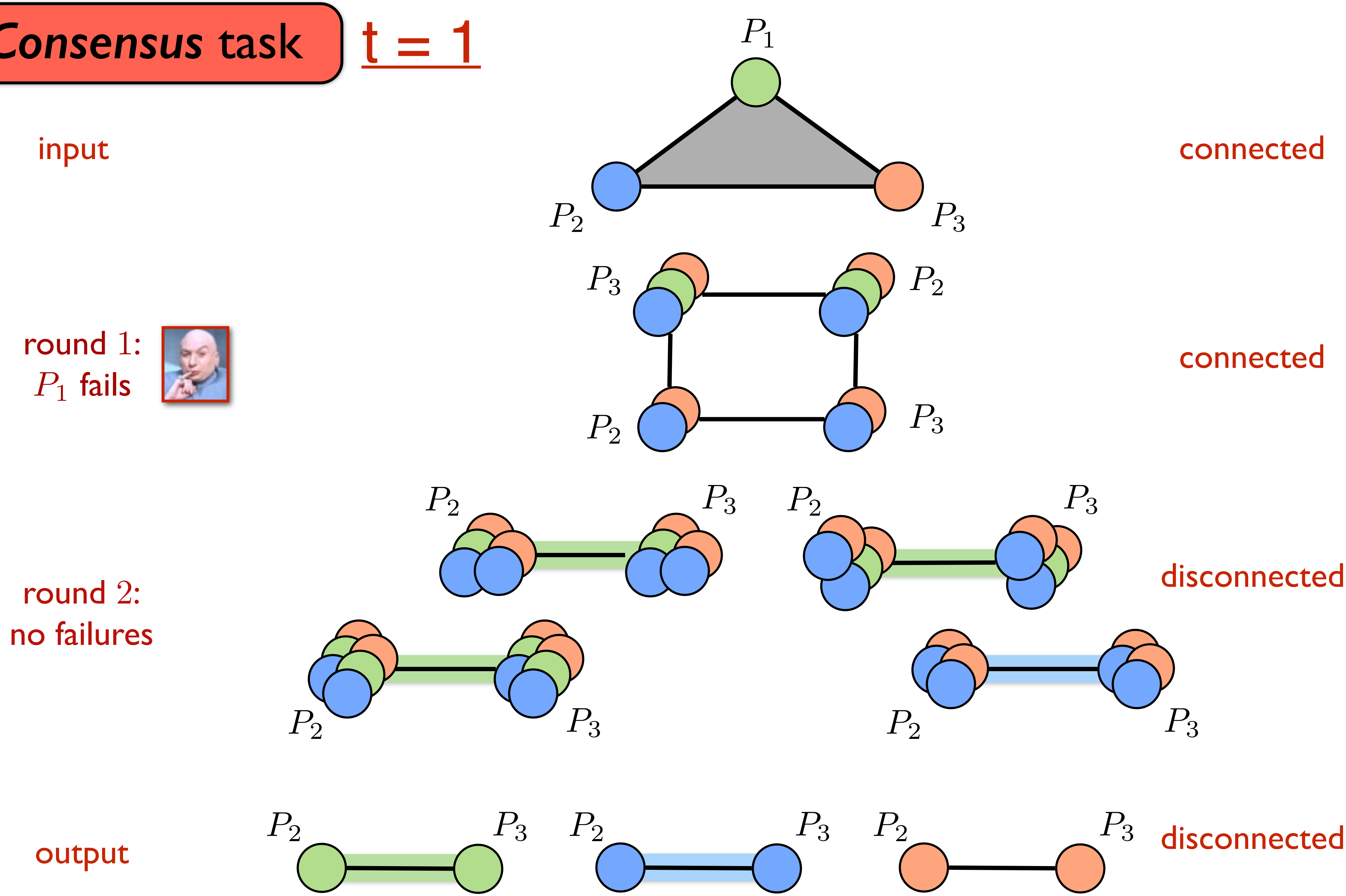
output

disconnected



Protocol Complexes

Consensus task t = 1



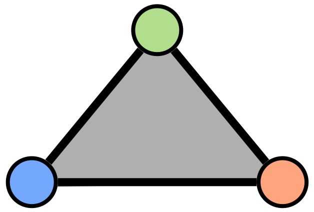
With Crash Failures…

Herlihy & Rajsbaum 2000

With Crash Failures…

Herlihy & Rajsbaum 2000

$\mathcal{K}_0 = \mathcal{I}^*$

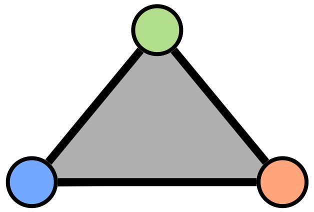


With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

$\mathcal{K}_0 = \mathcal{I}^*$

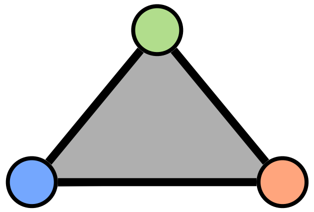


With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

$\mathcal{K}_0 = \mathcal{I}^*$



$\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$

$(k-1)$-connected

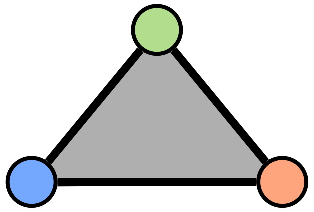
With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

k failures per round

$\mathcal{K}_0 = \mathcal{I}^*$ $\quad$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$



$(k-1)$-connected

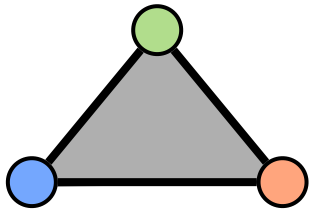
With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

k failures per round

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$



$(k-1)$-connected $(k-1)$-connected

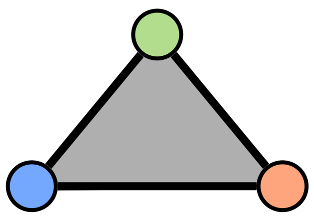
With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

k failures per round

$\mathcal{K}_0 = \mathcal{I}^*$



$\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$

$(k-1)$-connected

$\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$

$(k-1)$-connected

$\mathcal{K}_3 = \mathcal{R}_c(\mathcal{K}_2, k)$

$(k-1)$-connected

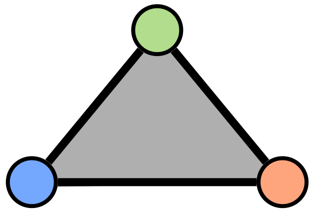
With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

k failures per round

$\mathcal{K}_0 = \mathcal{I}^*$ $\quad \mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\quad \mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\quad \mathcal{K}_3 = \mathcal{R}_c(\mathcal{K}_2, k)$



…

$(k-1)$-connected $\quad (k-1)$-connected $\quad (k-1)$-connected

With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

k failures per round

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\mathcal{K}_3 = \mathcal{R}_c(\mathcal{K}_2, k)$

…

$(k-1)$-connected $(k-1)$-connected $(k-1)$-connected

Theorem:

While the protocol complex is $(k-1)$-connected, we cannot solve the k-set agreement task

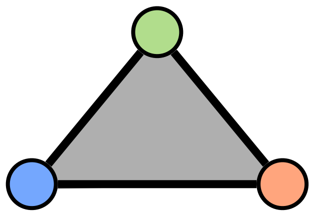
With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

k failures per round

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\mathcal{K}_3 = \mathcal{R}_c(\mathcal{K}_2, k)$



…

$(k-1)$-connected $(k-1)$-connected $(k-1)$-connected

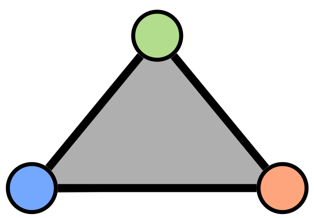
With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

k failures per round

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\mathcal{K}_3 = \mathcal{R}_c(\mathcal{K}_2, k)$



…

$(k-1)$-connected $(k-1)$-connected $(k-1)$-connected

Theorem:

We cannot solve k-set agreement with t failures in $\lfloor t/k \rfloor$ or less rounds.

With Crash Failures…

Herlihy & Rajsbaum 2000

adversarial execution

k failures per round

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\mathcal{K}_3 = \mathcal{R}_c(\mathcal{K}_2, k)$

$(k-1)$-connected $(k-1)$-connected $(k-1)$-connected

Theorem:

We cannot solve k-set agreement with t failures in $\lfloor t/k \rfloor$ or less rounds.

We have a $\lfloor t/k \rfloor + 1$ protocol

Byzantine Equivocation

Is equivocation a problem in synchronous systems?

Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

Byzantine Equivocation

Is equivocation a problem in synchronous systems? sort of

Ex.: 4 *processes,* 1 *failure*

Byzantine Equivocation

Is equivocation a problem in synchronous systems? sort of

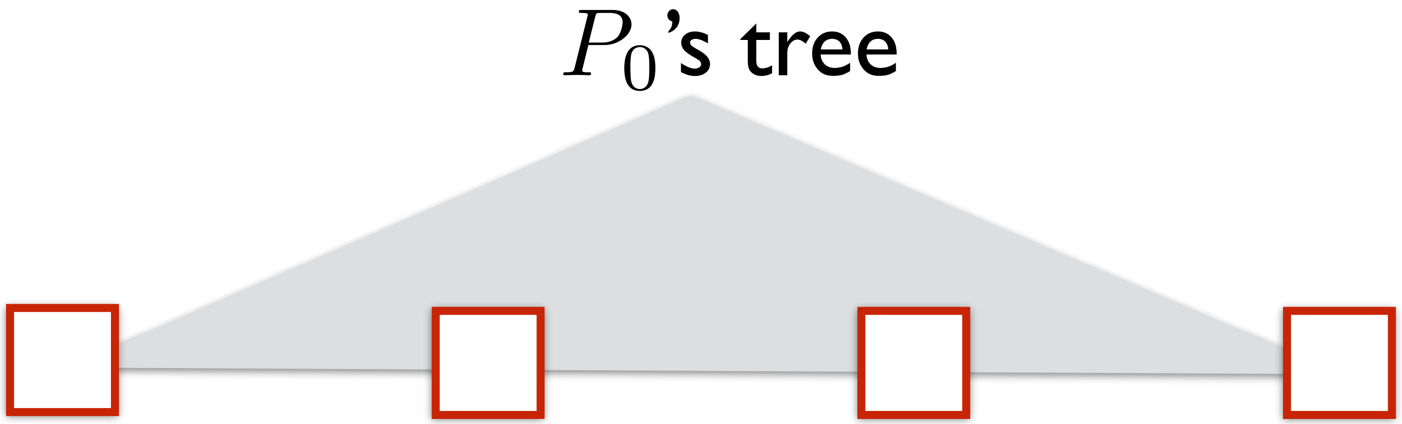
you know this

Ex.: 4 *processes,* 1 *failure*

Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

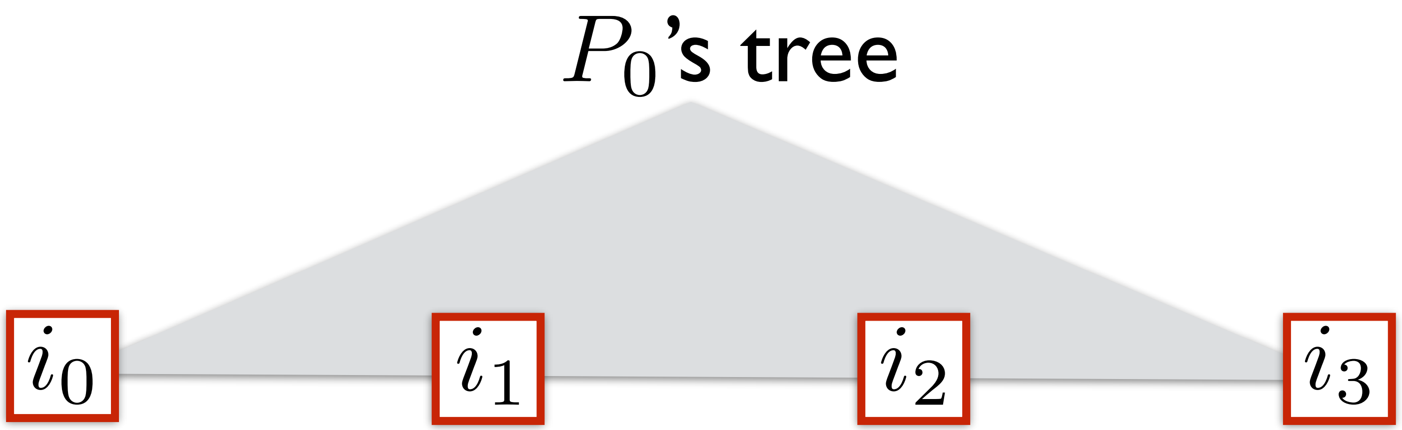
Ex.: 4 *processes,* 1 *failure*



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

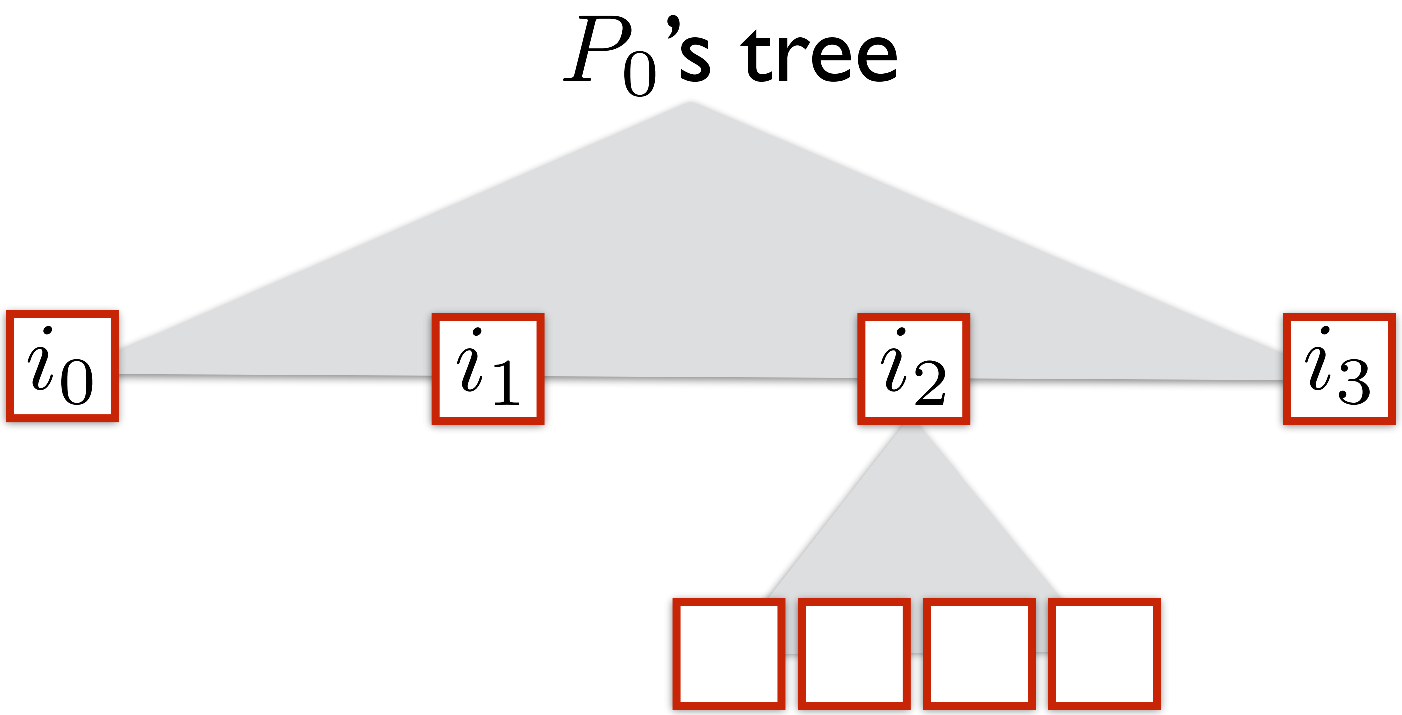
Ex.: 4 processes, 1 failure



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

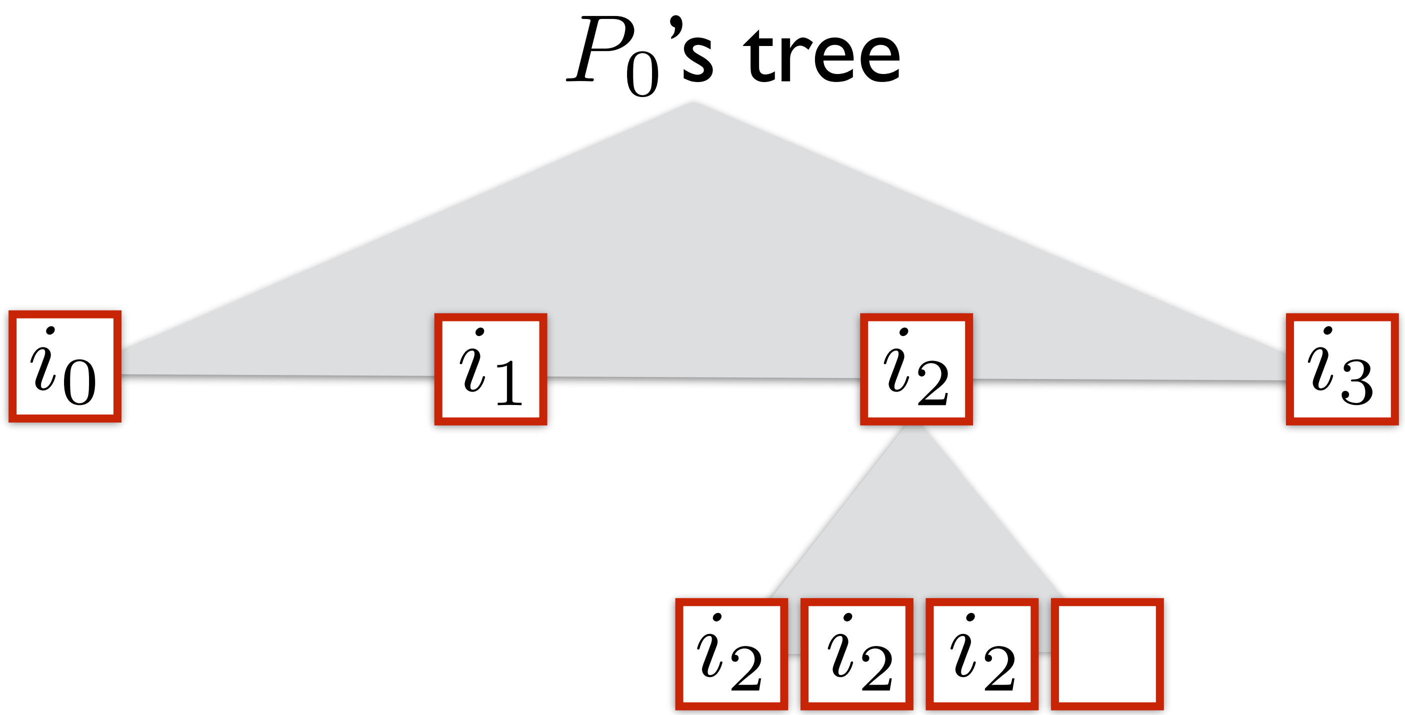
Ex.: 4 processes, 1 failure



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

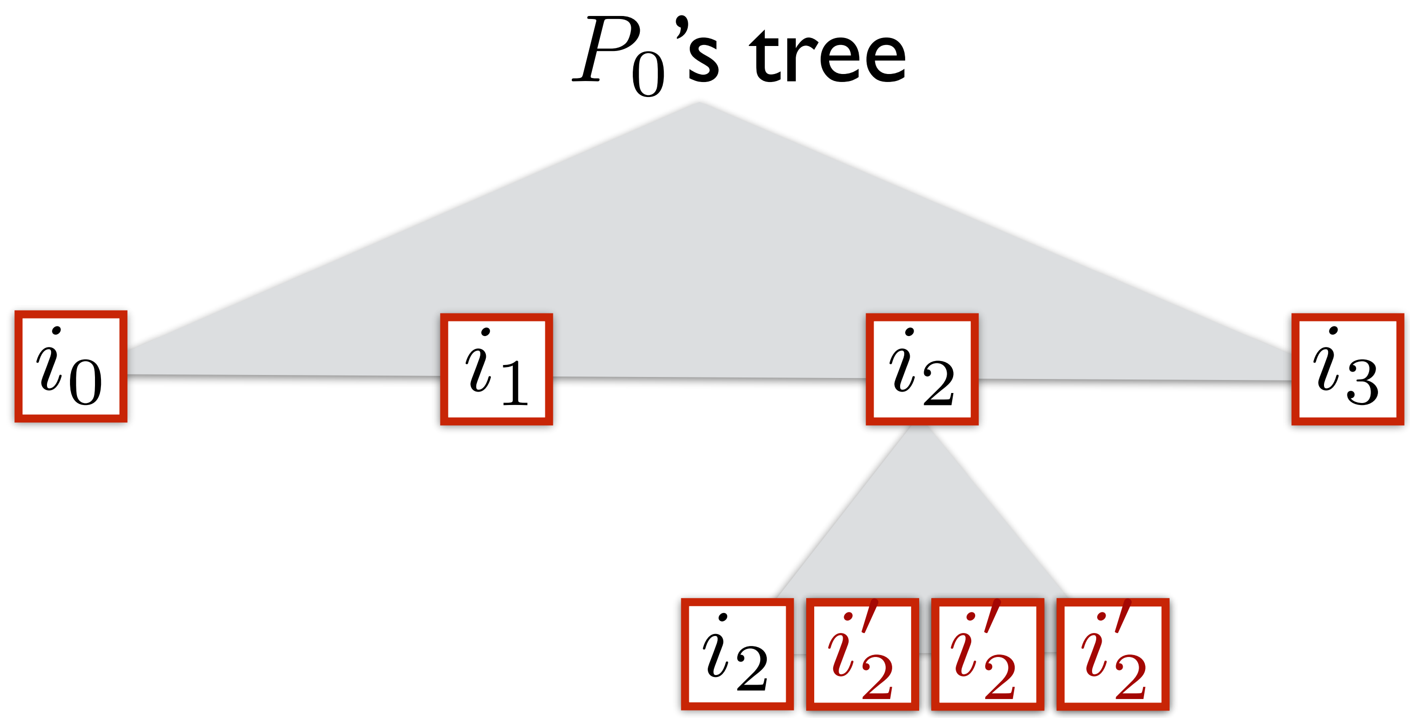
Ex.: 4 *processes,* 1 *failure*



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

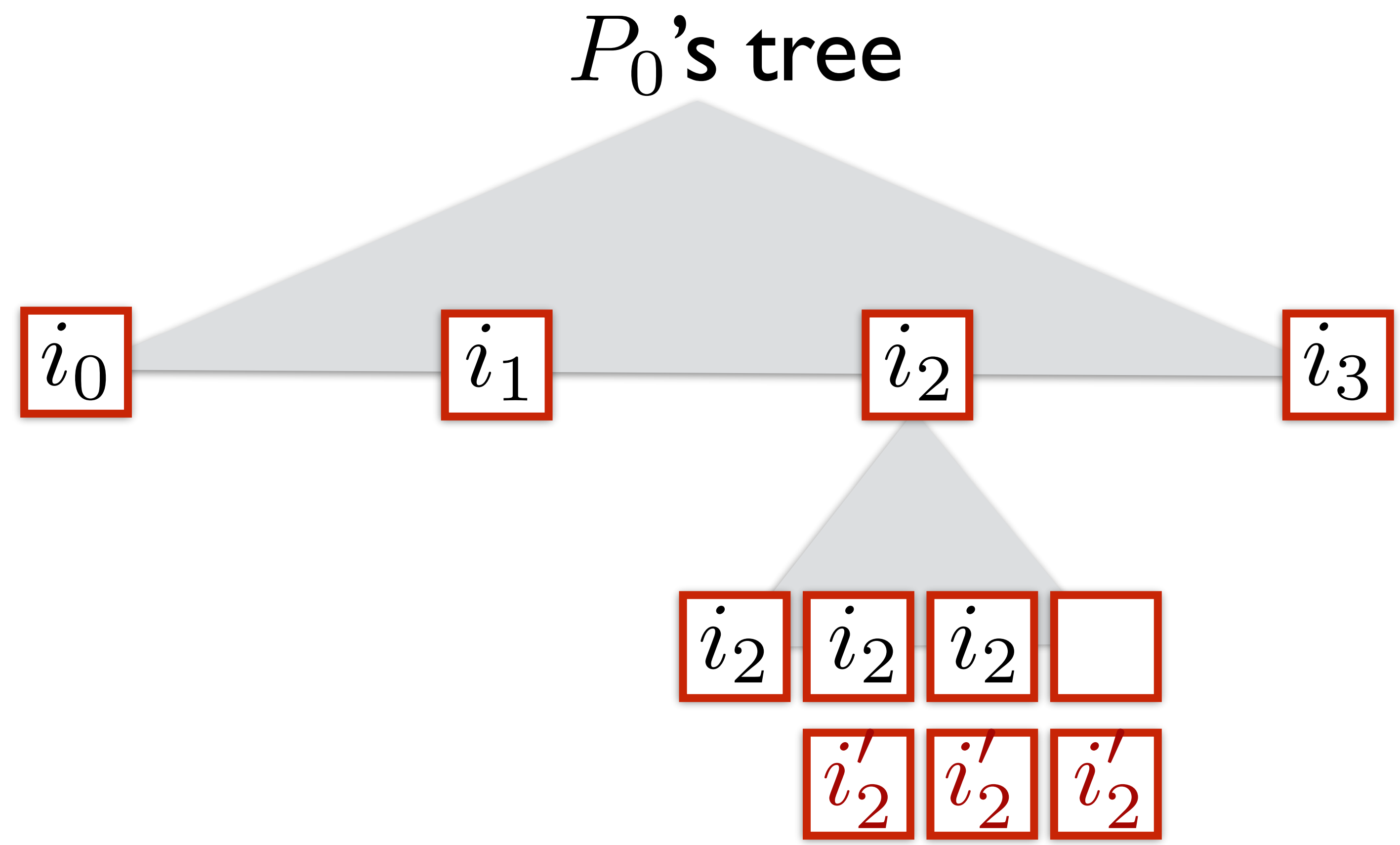
Ex.: 4 processes, 1 failure



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

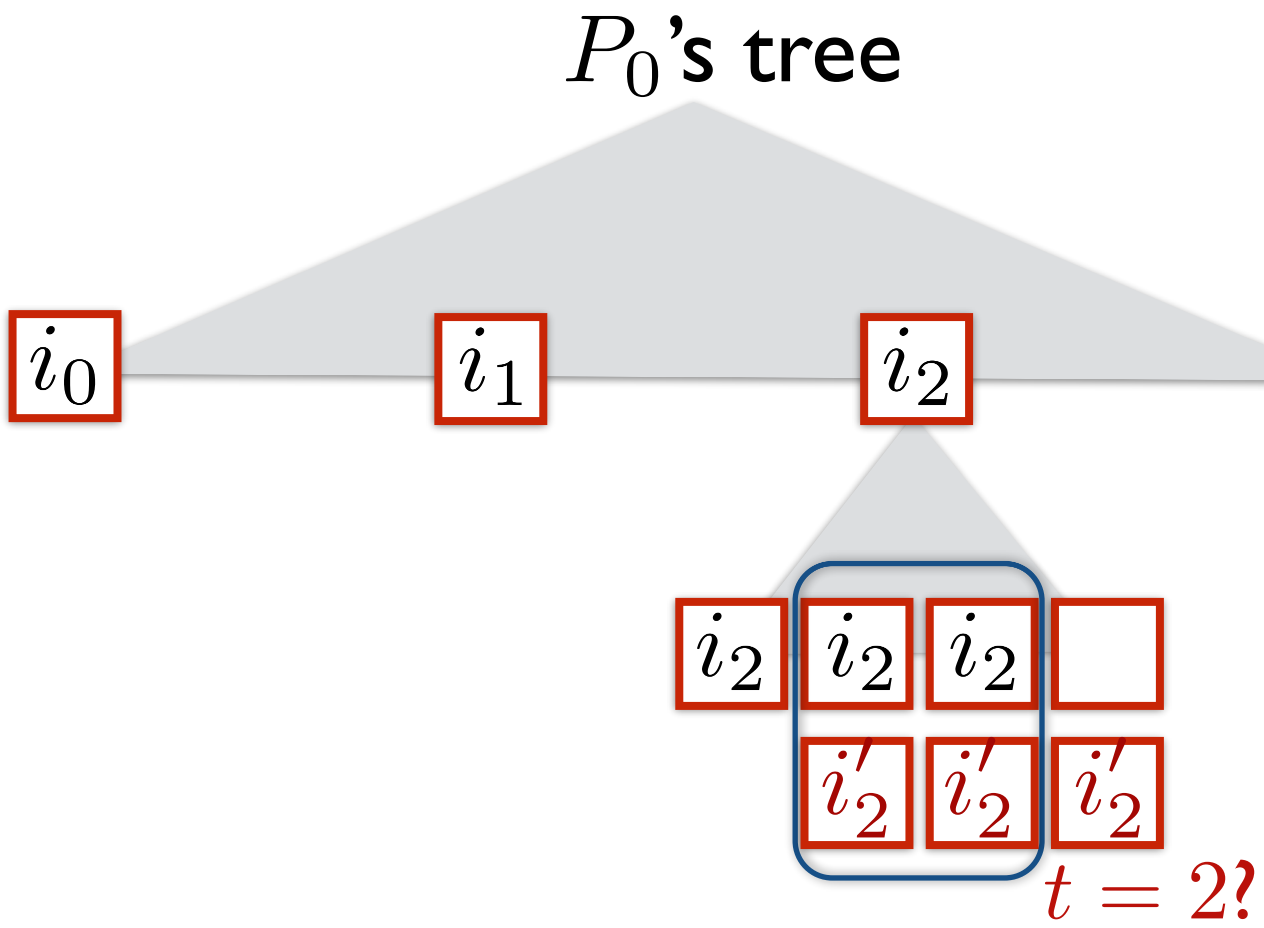
Ex.: 4 processes, 1 failure



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

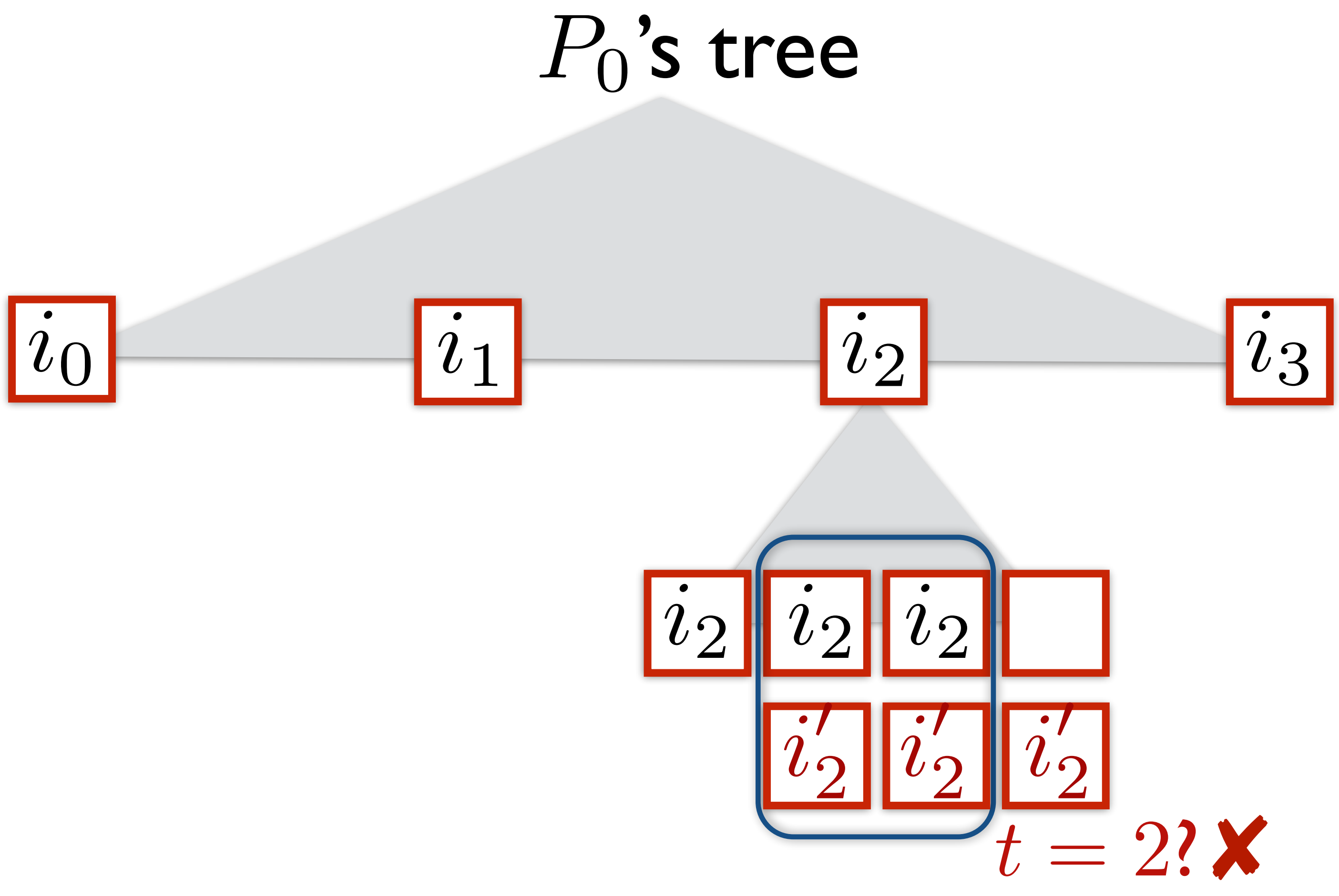
Ex.: 4 processes, 1 failure



Byzantine Equivocation

Is equivocation a problem in synchronous systems? sort of

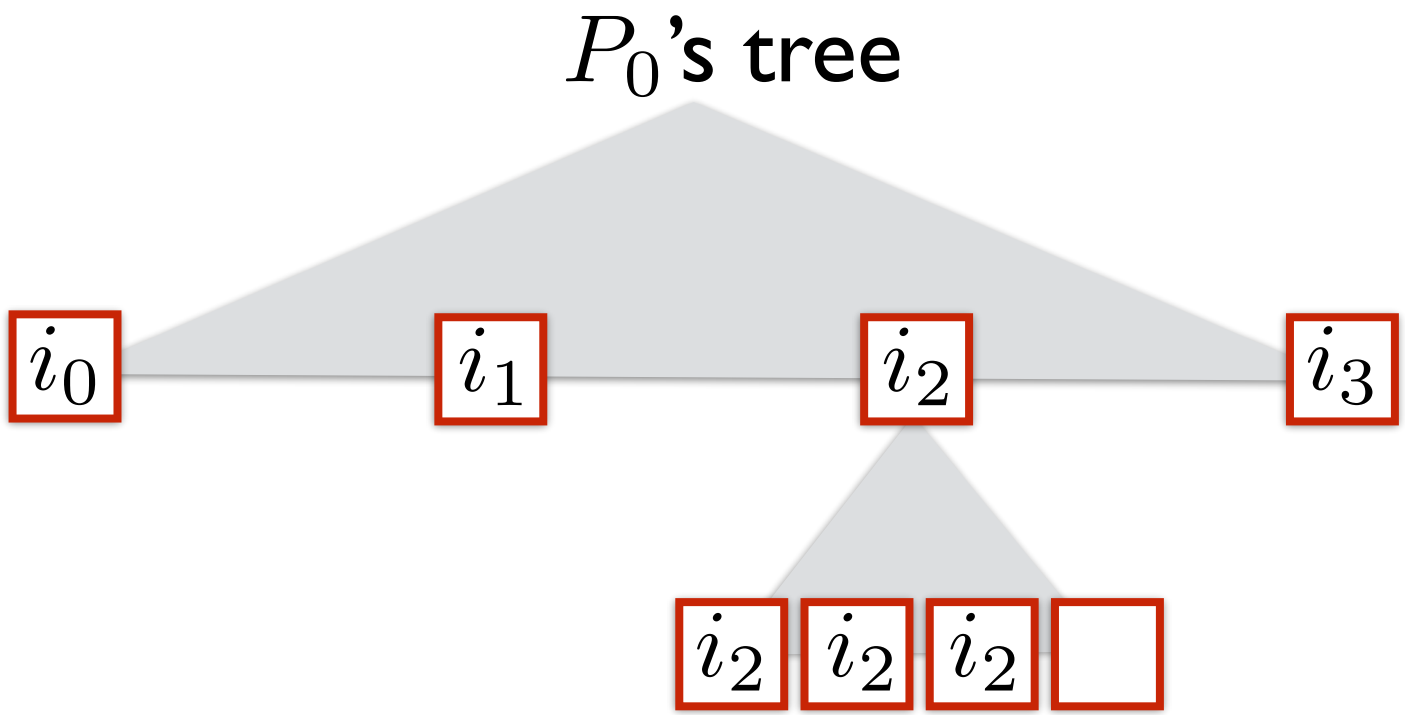
Ex.: 4 *processes,* 1 *failure*



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

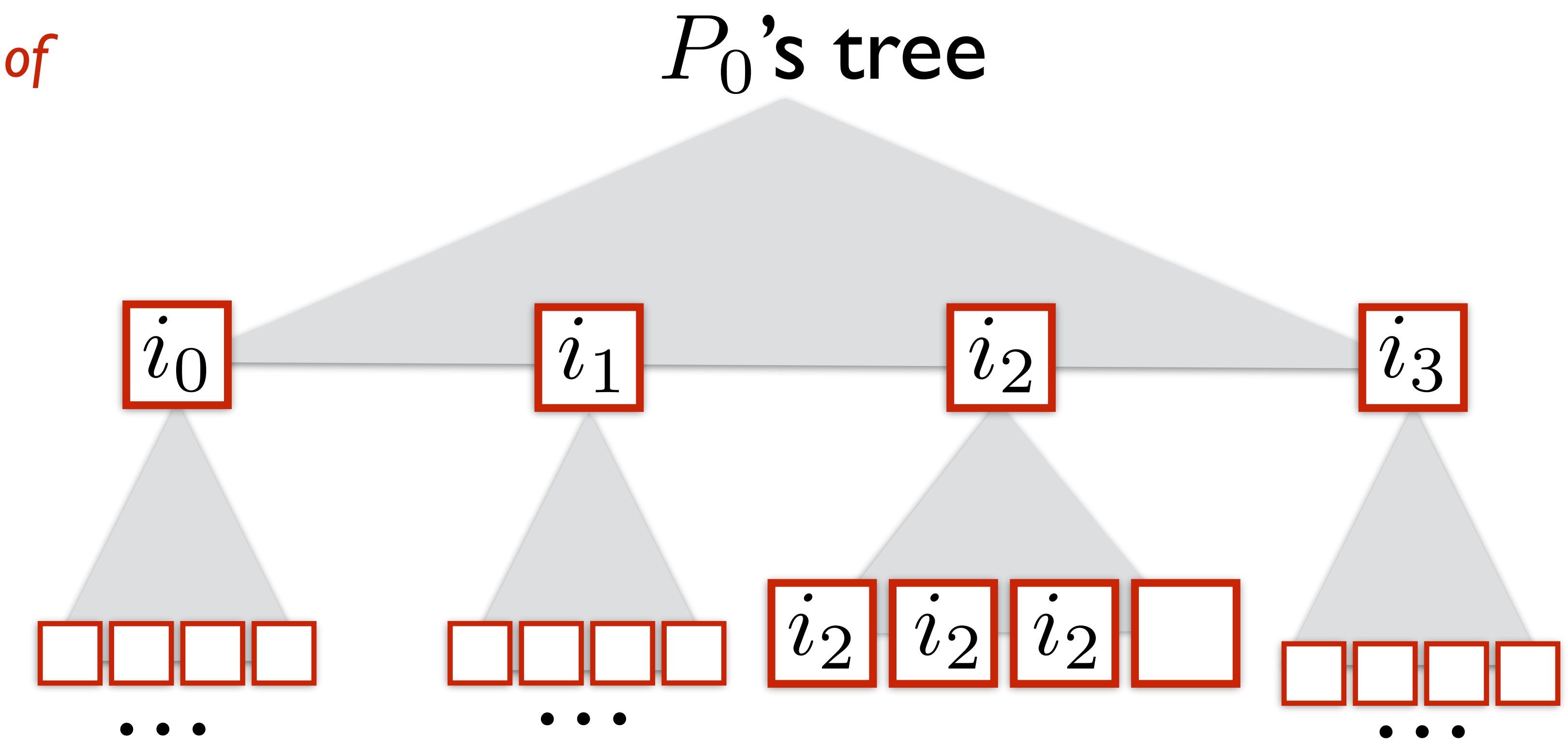
Ex.: 4 processes, 1 failure



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

Ex.: 4 processes, 1 failure

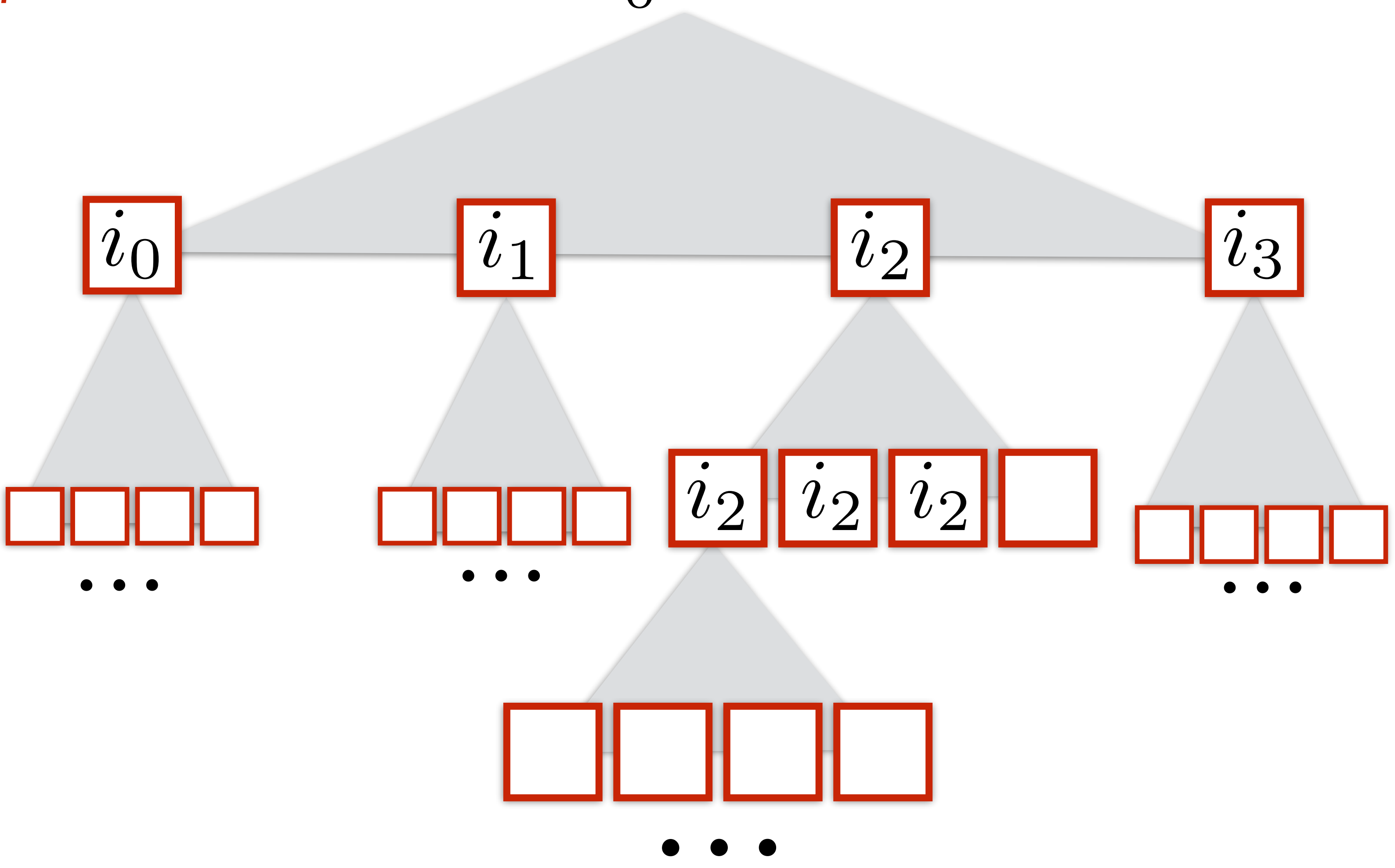


Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

Ex.: 4 *processes,* 1 *failure*

P_0's tree

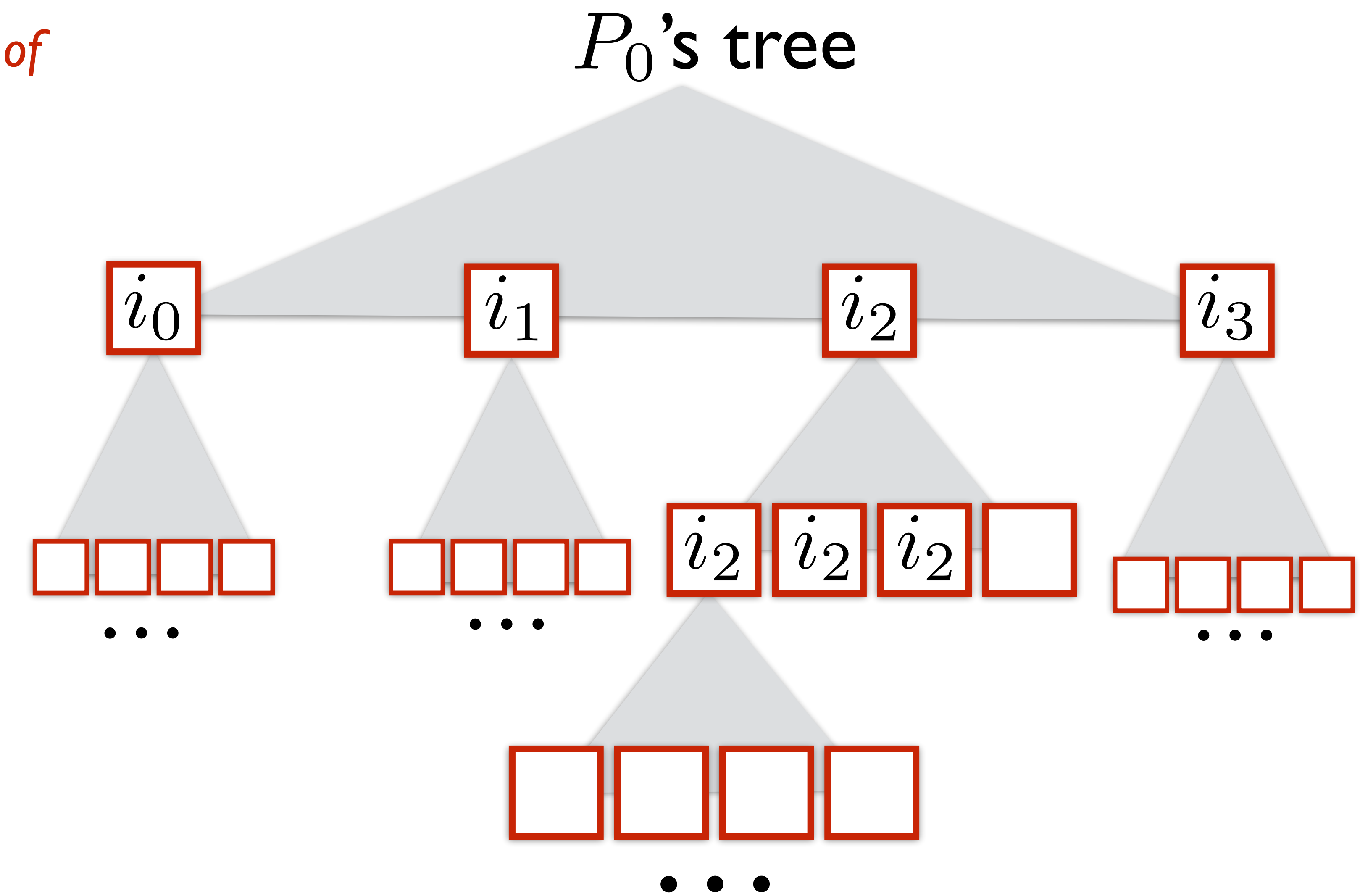


Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

Ex.: 4 processes, 1 failure

Inner nodes:



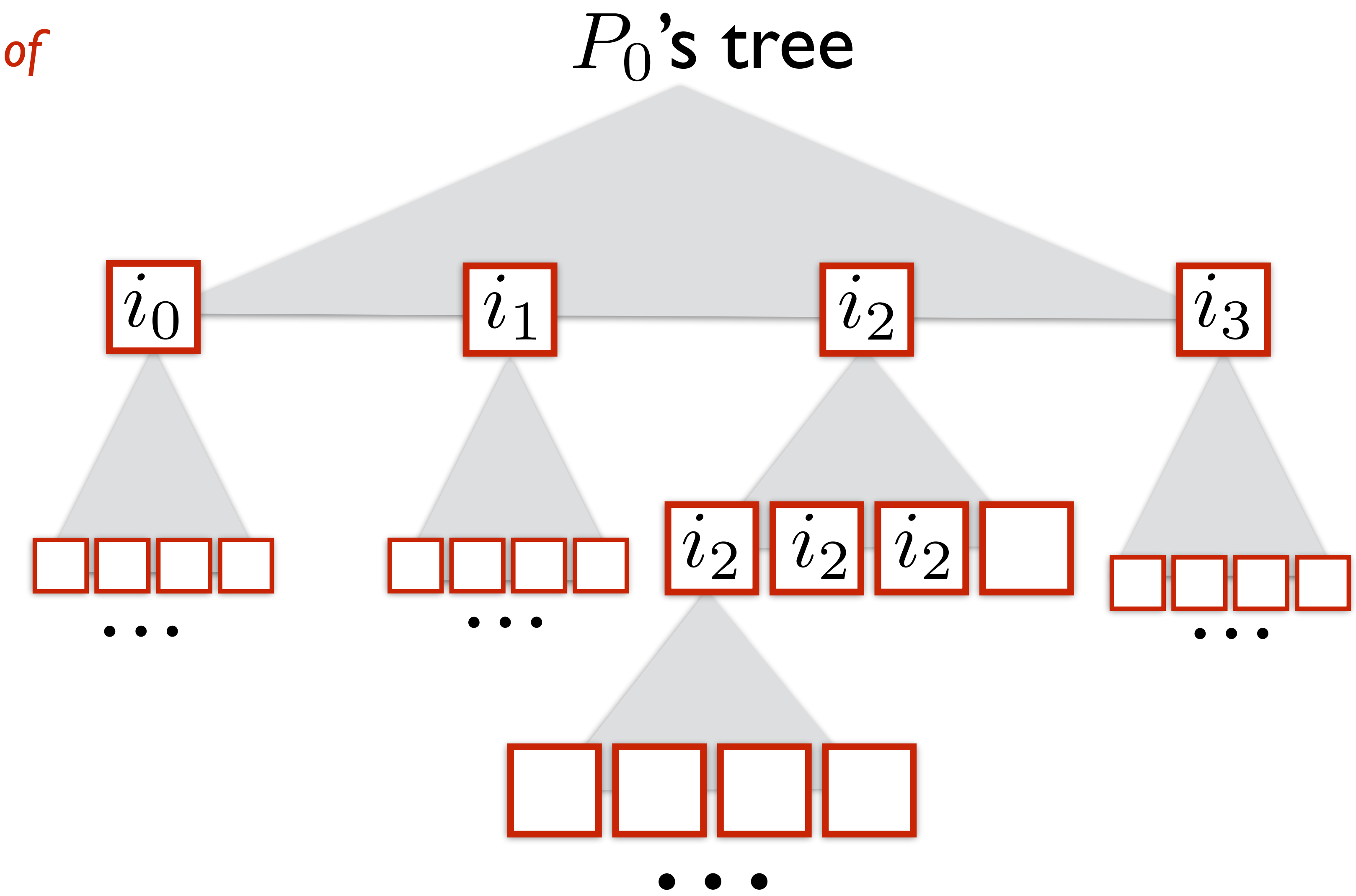
Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

Ex.: 4 processes, 1 failure

Inner nodes:

validated



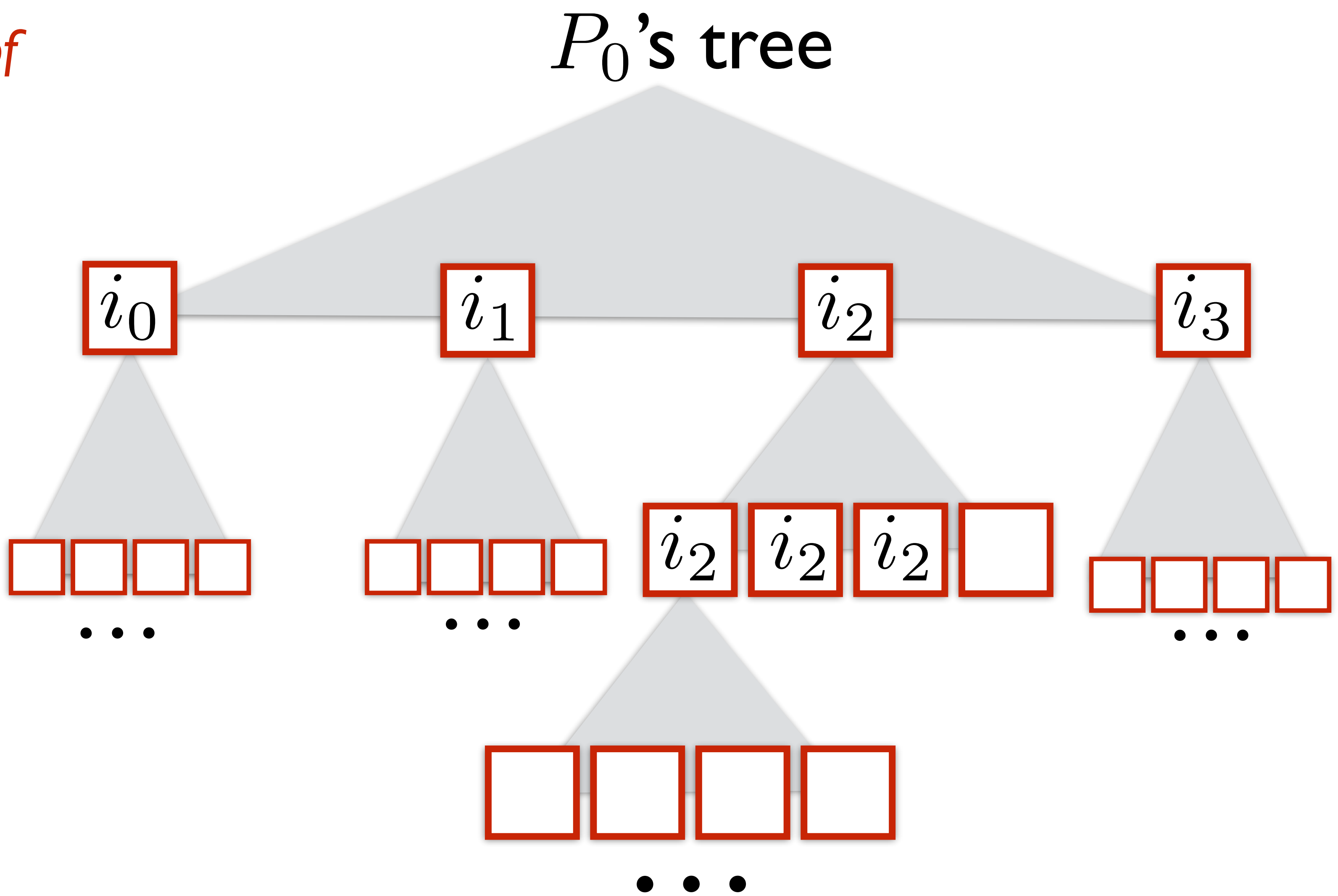
Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

Ex.: 4 processes, 1 failure

Inner nodes: validated

Leaf nodes:



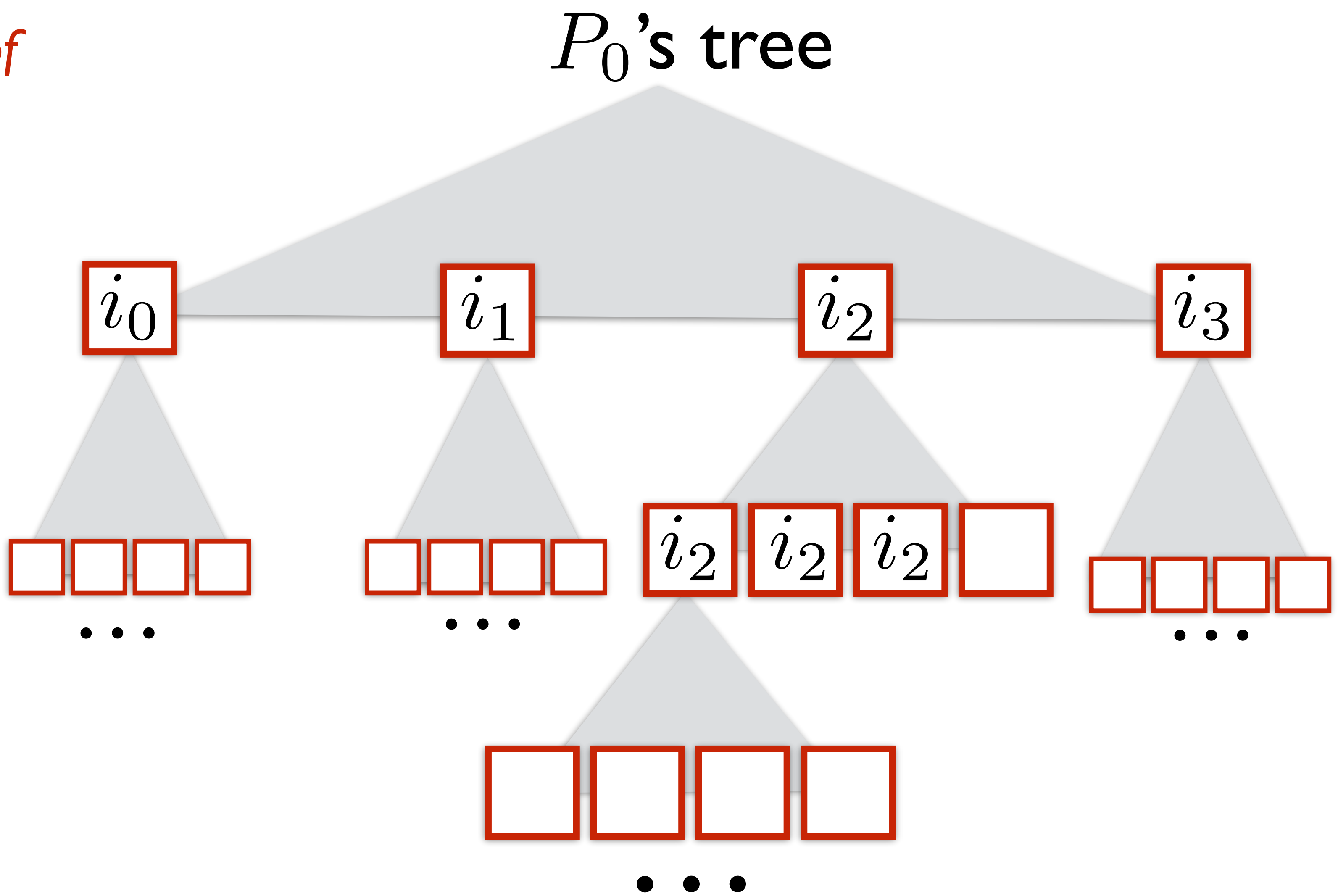
Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

Ex.: 4 processes, 1 failure

Inner nodes: validated

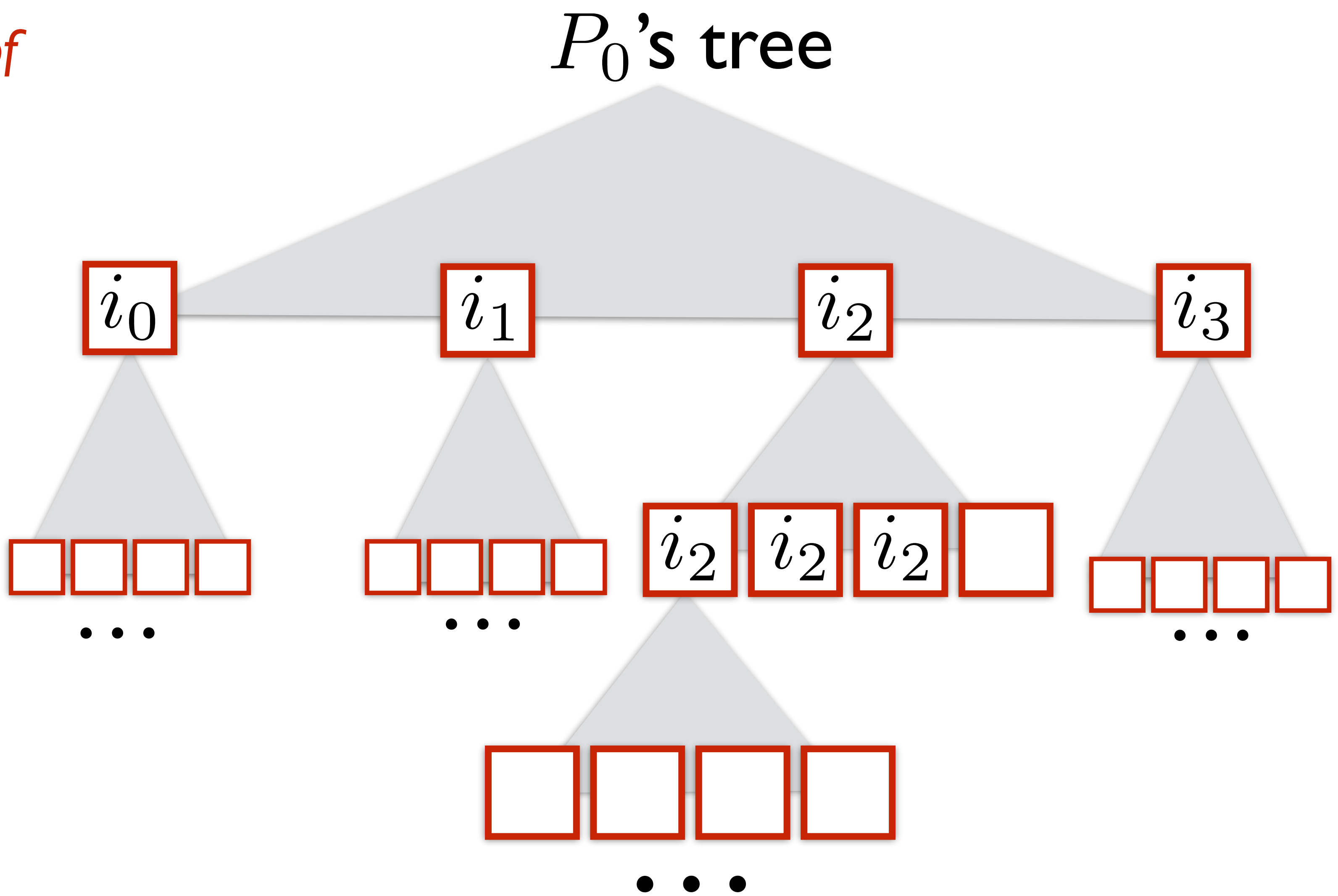
Leaf nodes: not validated



Byzantine Equivocation

Is equivocation a problem in synchronous systems? *sort of*

Ex.: 4 processes, 1 failure



Inner nodes: validated

Leaf nodes: not validated

Last-round equivocation is problematic…

Byzantine Equivocation

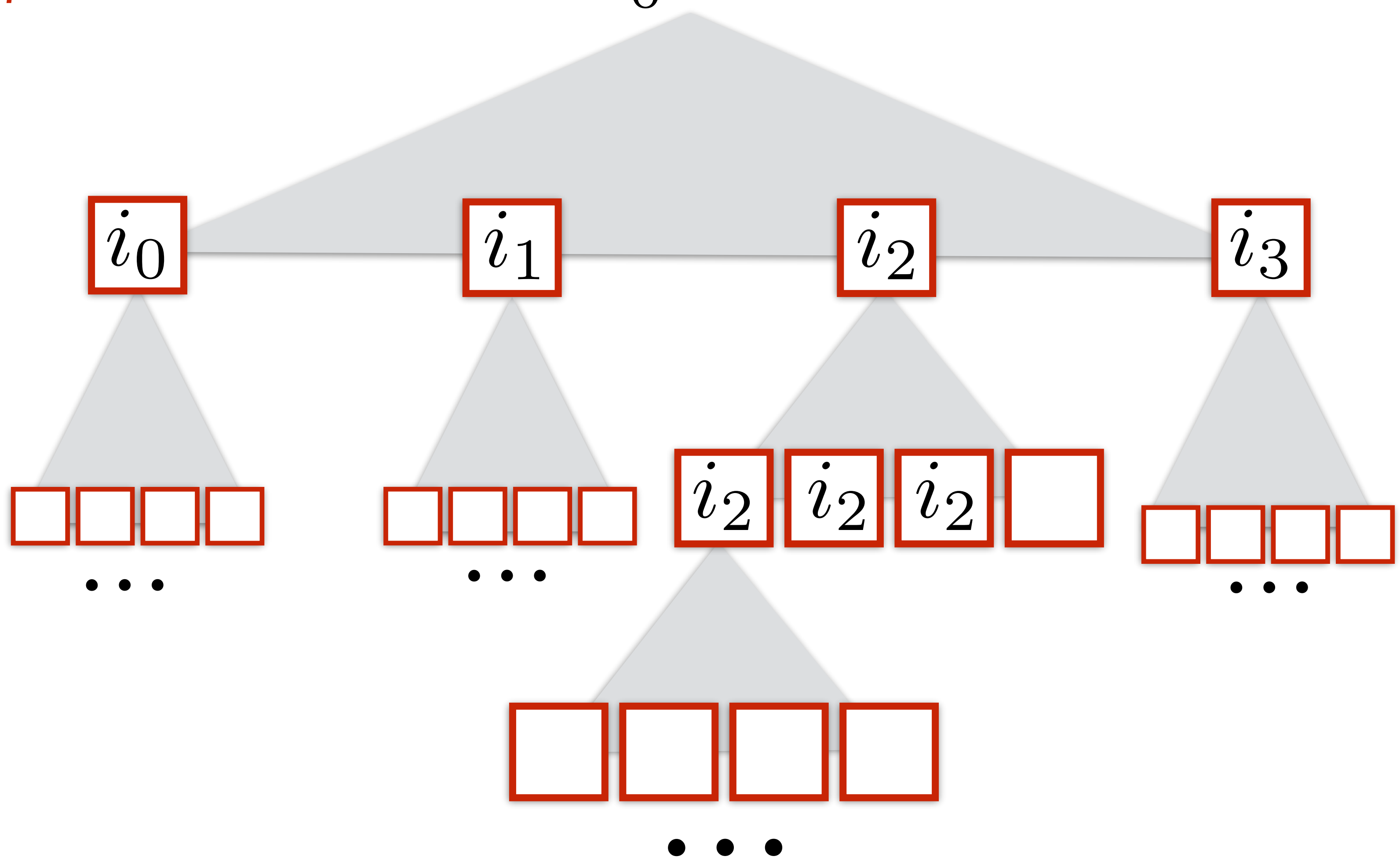
Is equivocation a problem in synchronous systems? *sort of*

Ex.: 4 processes, 1 failure

Inner nodes: validated

Leaf nodes: not validated

P_0's tree



Last-round equivocation is problematic…

… if Byzantine processes do not “reveal" themselves

Strategy

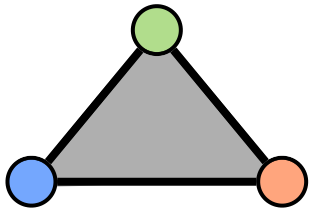
Strategy

We define a ***new*** round operator

Strategy

We define a ***new*** round operator

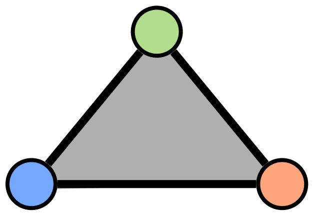
$\mathcal{K}_0 = \mathcal{I}^*$



Strategy

We define a ***new*** round operator

$\mathcal{K}_0 = \mathcal{I}^*$ $\quad$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\quad$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$



...

$(k-1)$**-connected** $\quad$ $(k-1)$**-connected**

We define a *new* round operator

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$

$(k-1)$-connected $(k-1)$-connected

$\lfloor t/k \rfloor$ rounds

Strategy

We define a ***new*** round operator

$\mathcal{K}_0 = \mathcal{I}^*$ $\quad$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\quad$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$

...

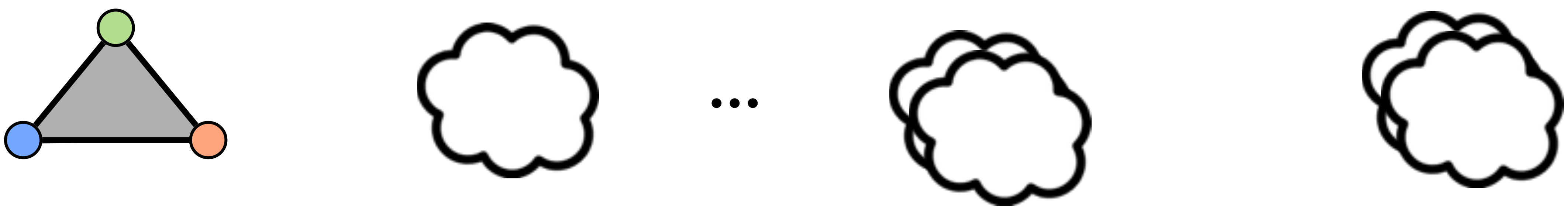
$(k-1)$-connected $\quad$ $(k-1)$-connected

$\lfloor t/k \rfloor$ rounds $\quad$ if $t \bmod k \neq 0$

Strategy

We define a ***new*** round operator

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\mathcal{K}_3 = \mathcal{R}_e(\mathcal{K}_2)$



$(k-1)$-connected $(k-1)$-connected $(k-1)$-connected

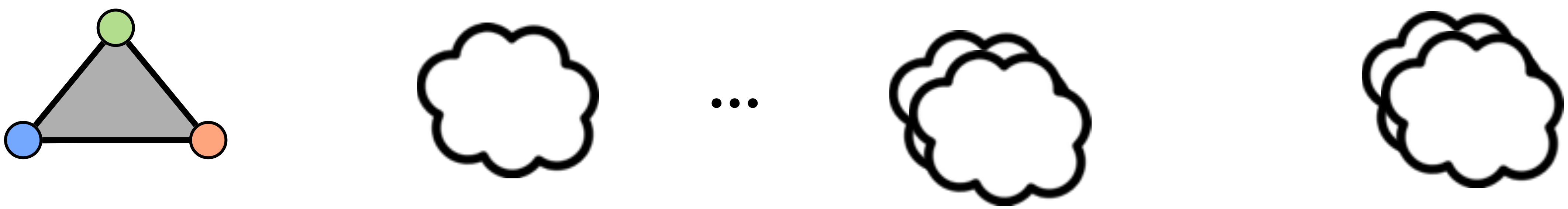
$\lfloor t/k \rfloor$ rounds

if t mod $k \neq 0$

Strategy

We define a ***new*** round operator

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\mathcal{K}_3 = \mathcal{R}_e(\mathcal{K}_2)$



$(k-1)$-connected $(k-1)$-connected $(k-1)$-connected

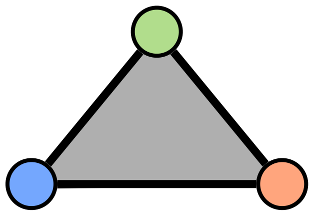
$\lfloor t/k \rfloor$ rounds if t mod $k \neq 0$

$\lceil t/k \rceil$ rounds

Strategy

We define a *new* round operator

$\mathcal{K}_0 = \mathcal{I}^*$ | $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ | ... | $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ | $\mathcal{K}_3 = \mathcal{R}_e(\mathcal{K}_2)$



$(k-1)$-connected | $(k-1)$-connected | $(k-1)$-connected

$\lfloor t/k \rfloor$ rounds

if $t \bmod k \neq 0$

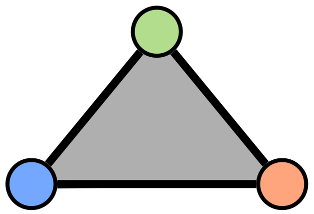
$\lceil t/k \rceil$ rounds

k-set agreement protocol

Strategy

We define a ***new*** round operator

$\mathcal{K}_0 = \mathcal{I}^*$



$\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$

$(k-1)$-connected

...

$\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$

$(k-1)$-connected

$\mathcal{K}_3 = \mathcal{R}_e(\mathcal{K}_2)$

$(k-1)$-connected

$\lfloor t/k \rfloor$ rounds

if $t \bmod k \neq 0$

$\lceil t/k \rceil$ rounds

k-set agreement protocol

Generalize the consensus protocol

The Equivocation Operator

The Equivocation Operator

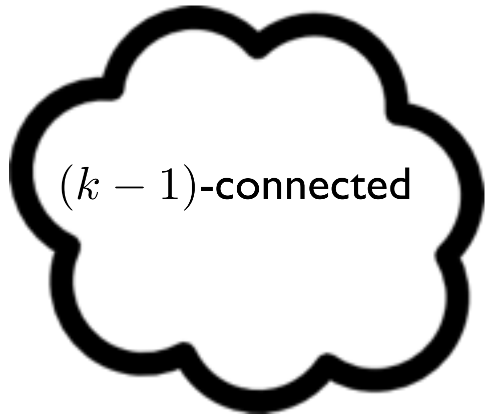
$\lfloor t/k \rfloor$ rounds,
k failures/round

$\cdots$

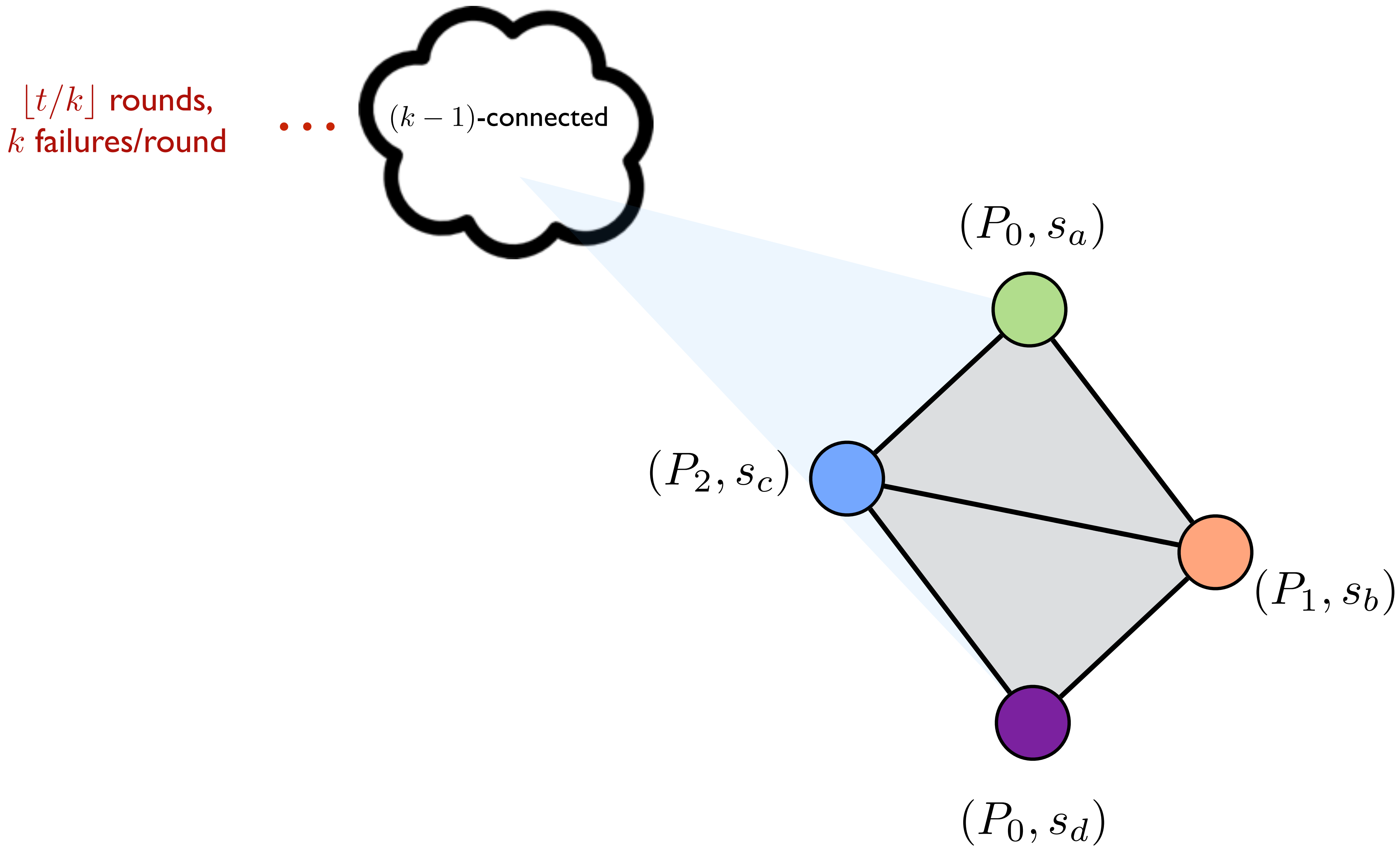
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round

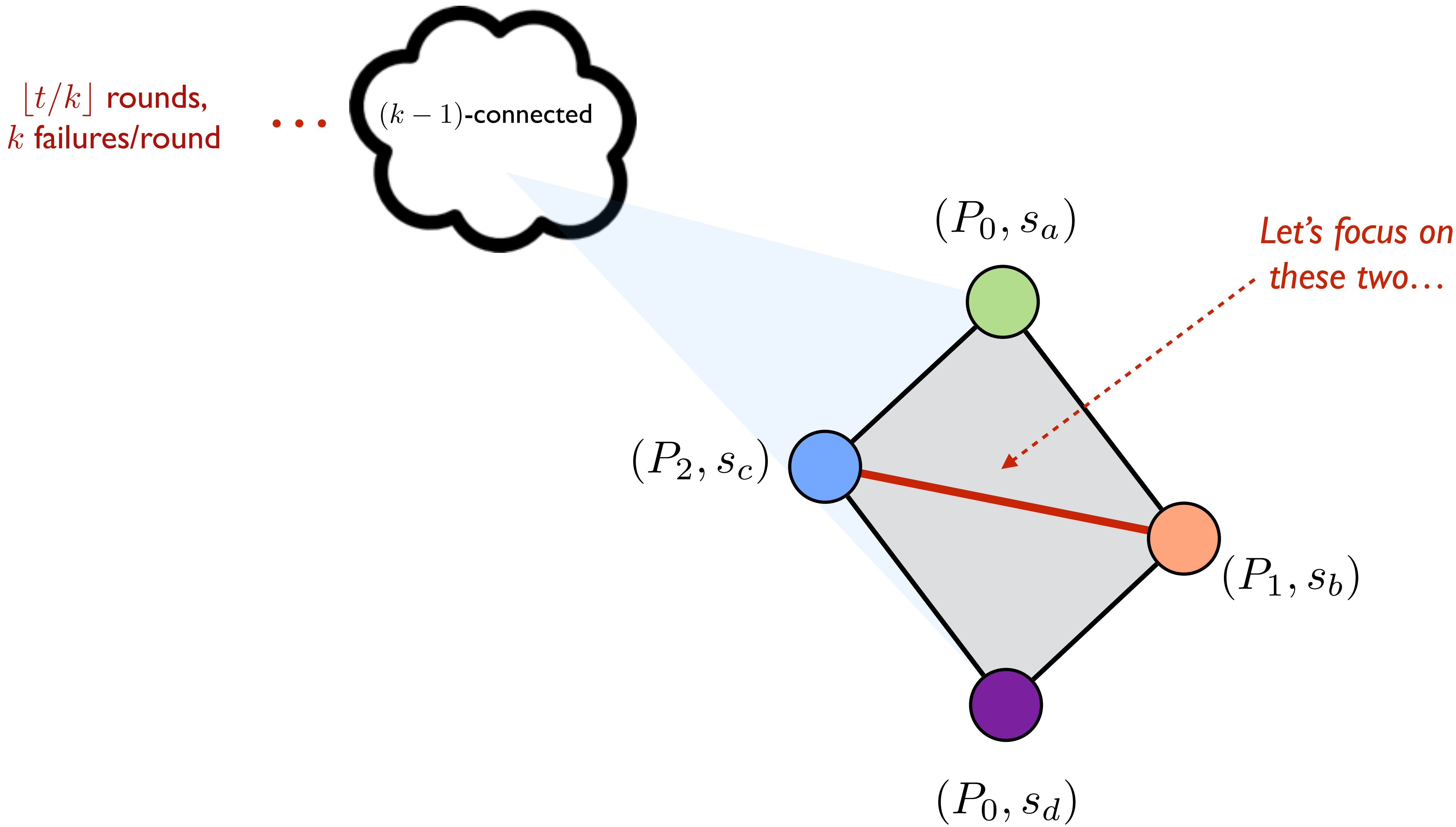
. . .



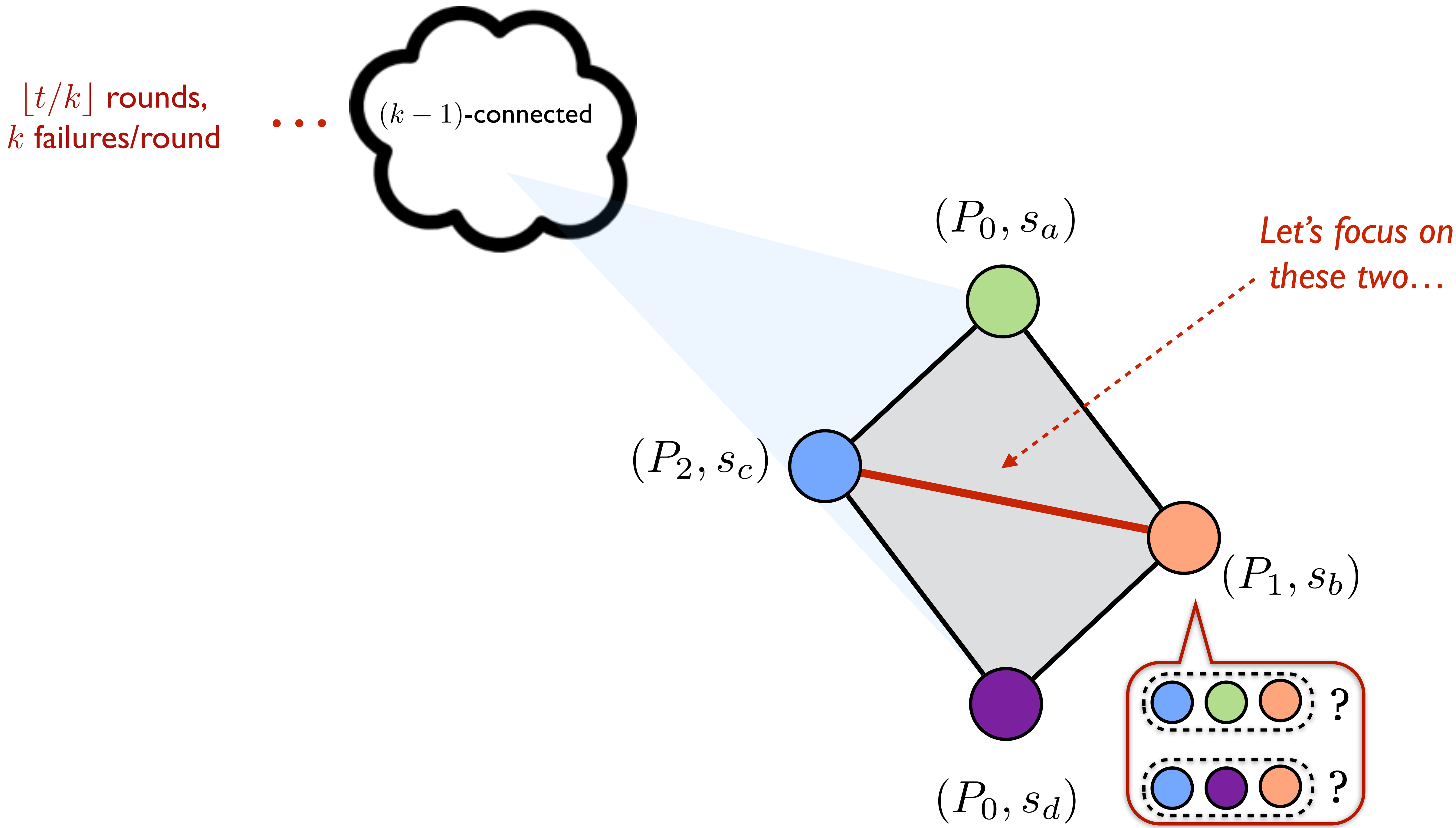
The Equivocation Operator



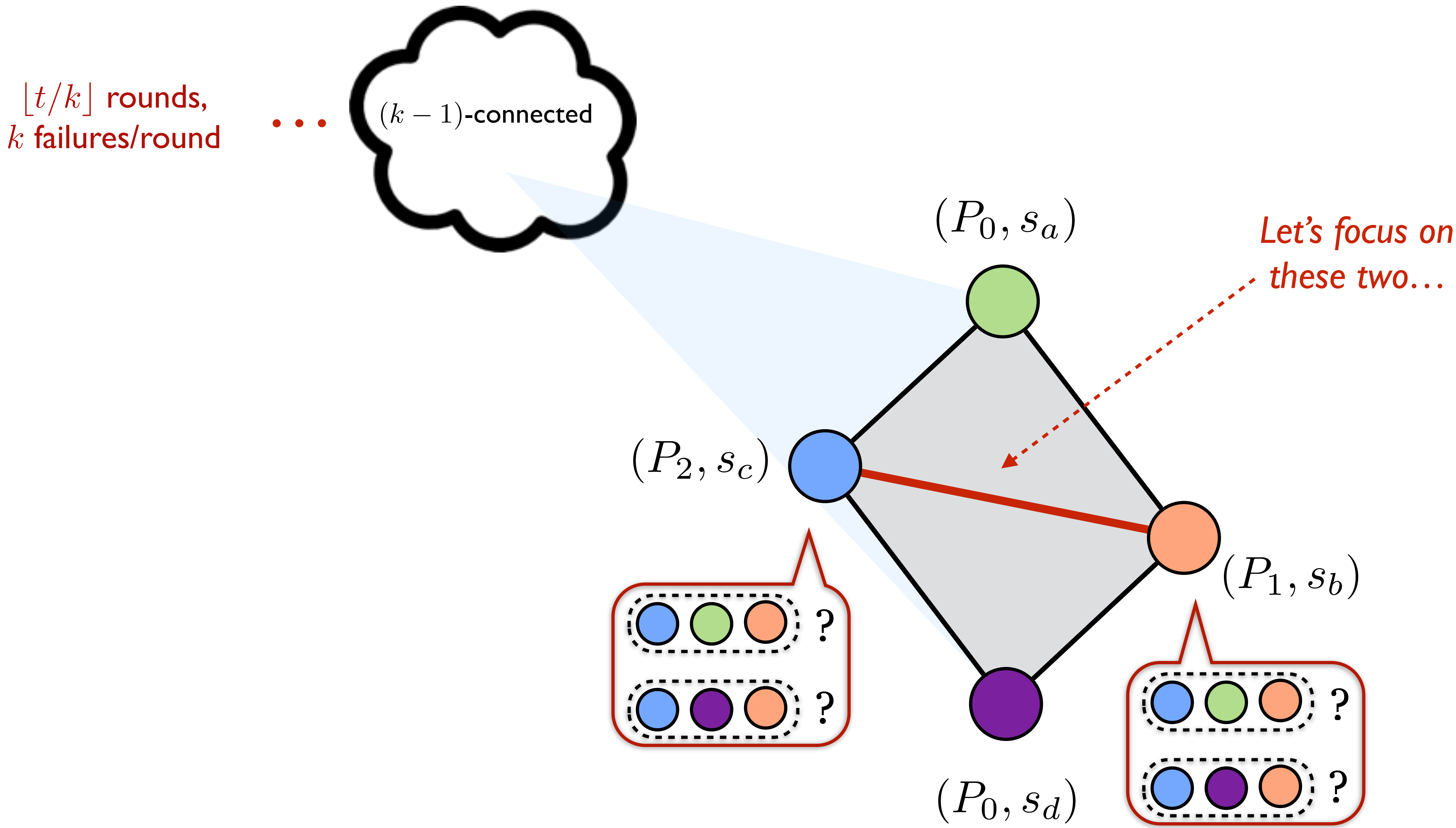
The Equivocation Operator



The Equivocation Operator

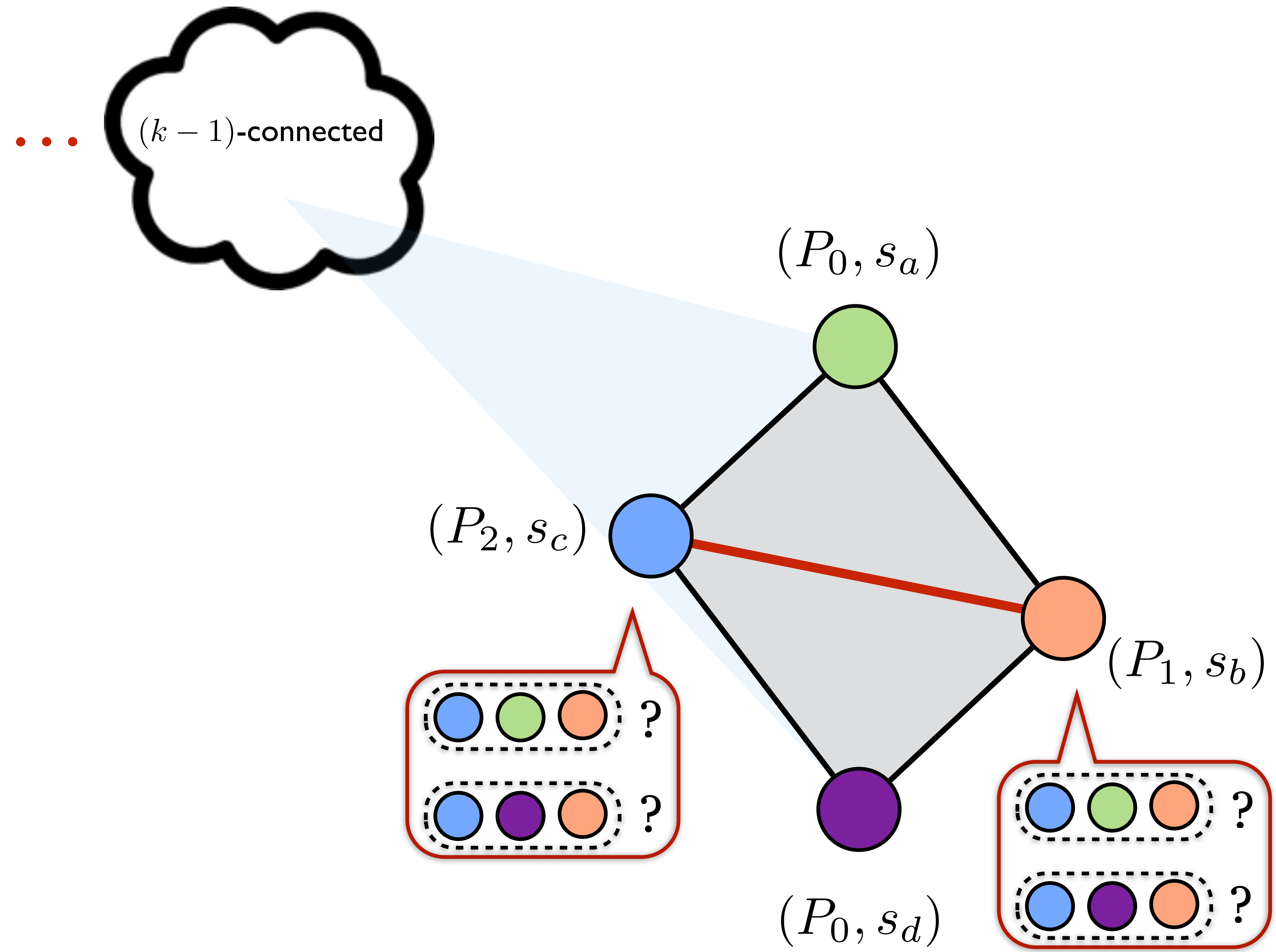


The Equivocation Operator



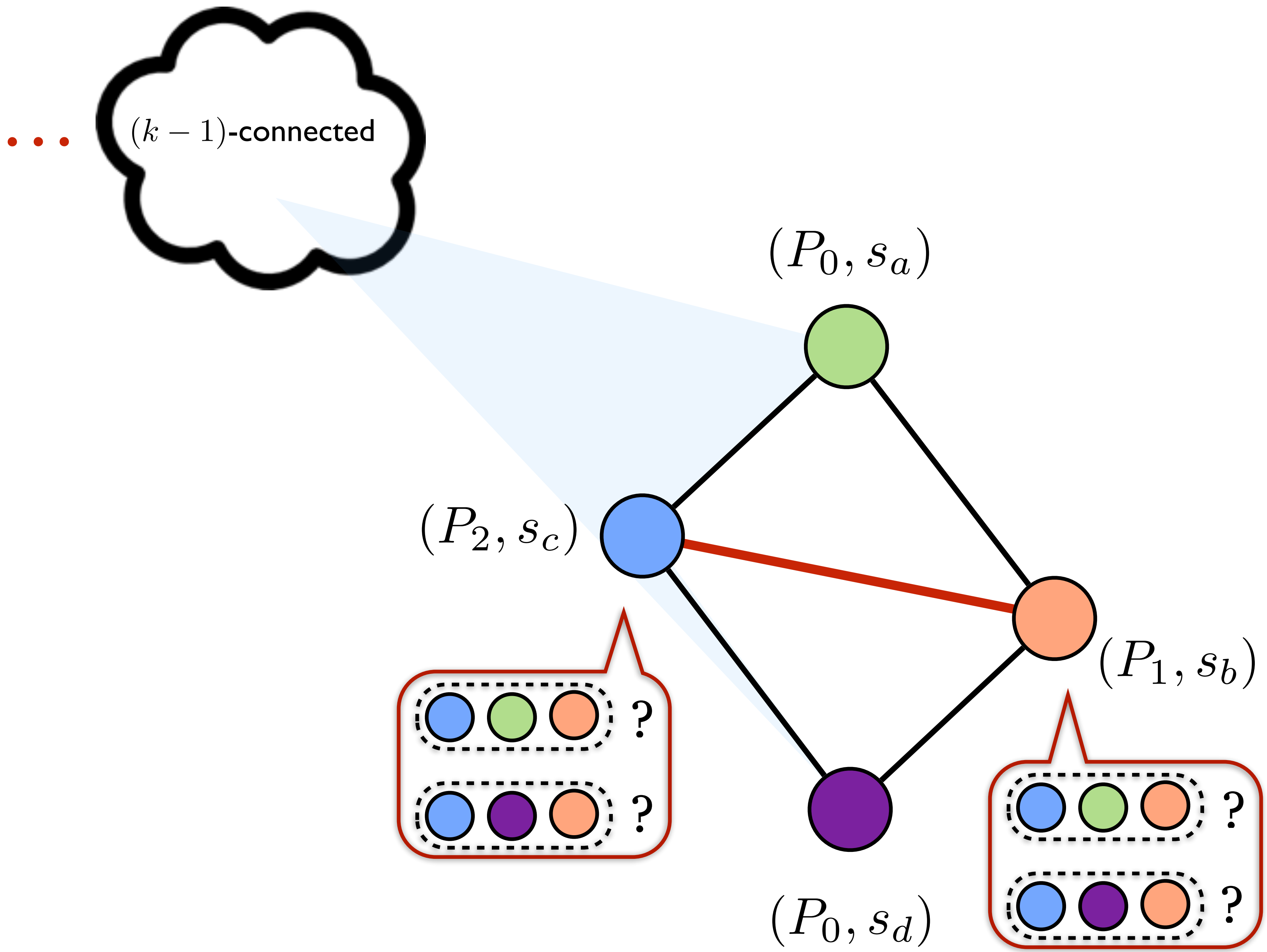
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round

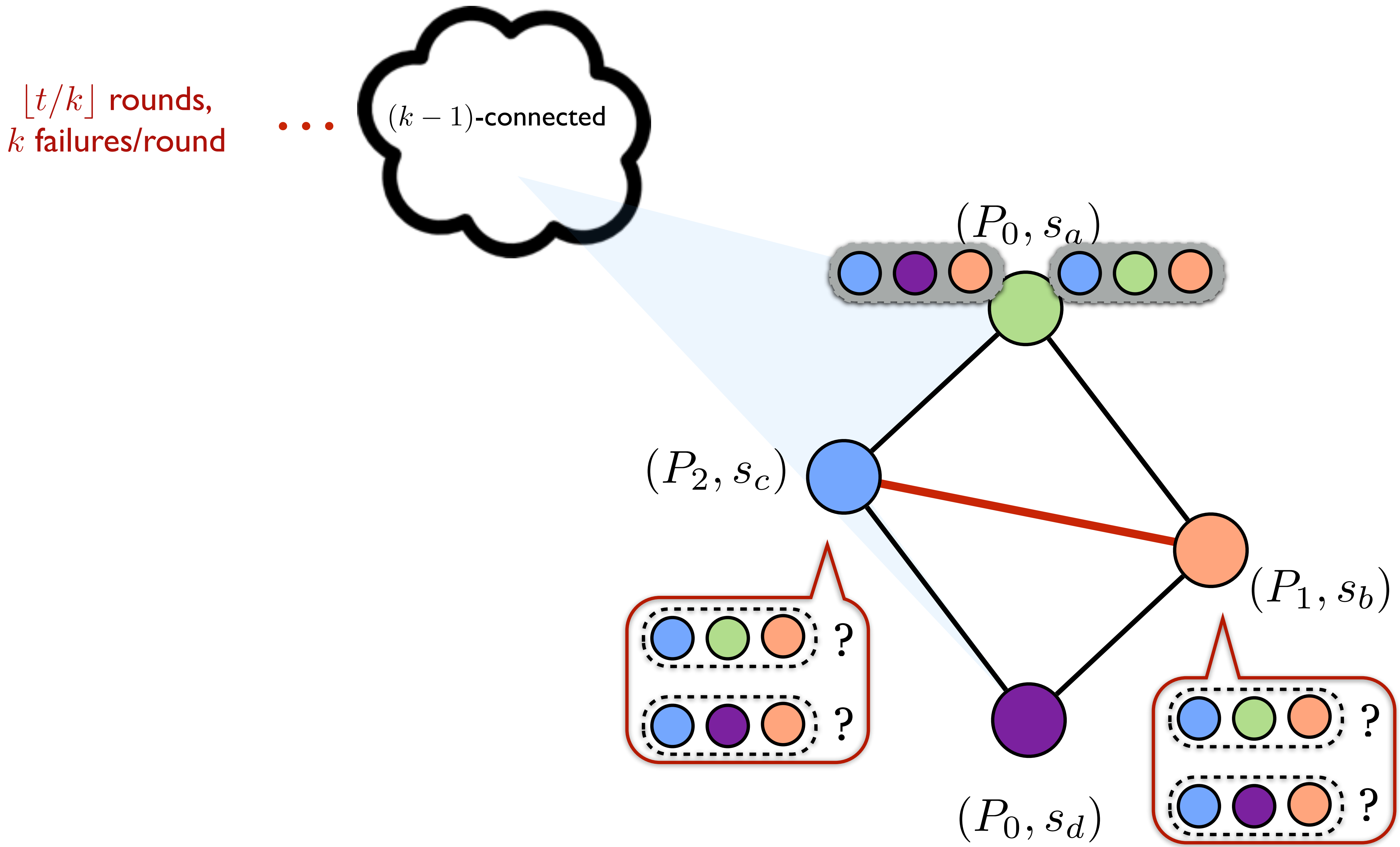


The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round

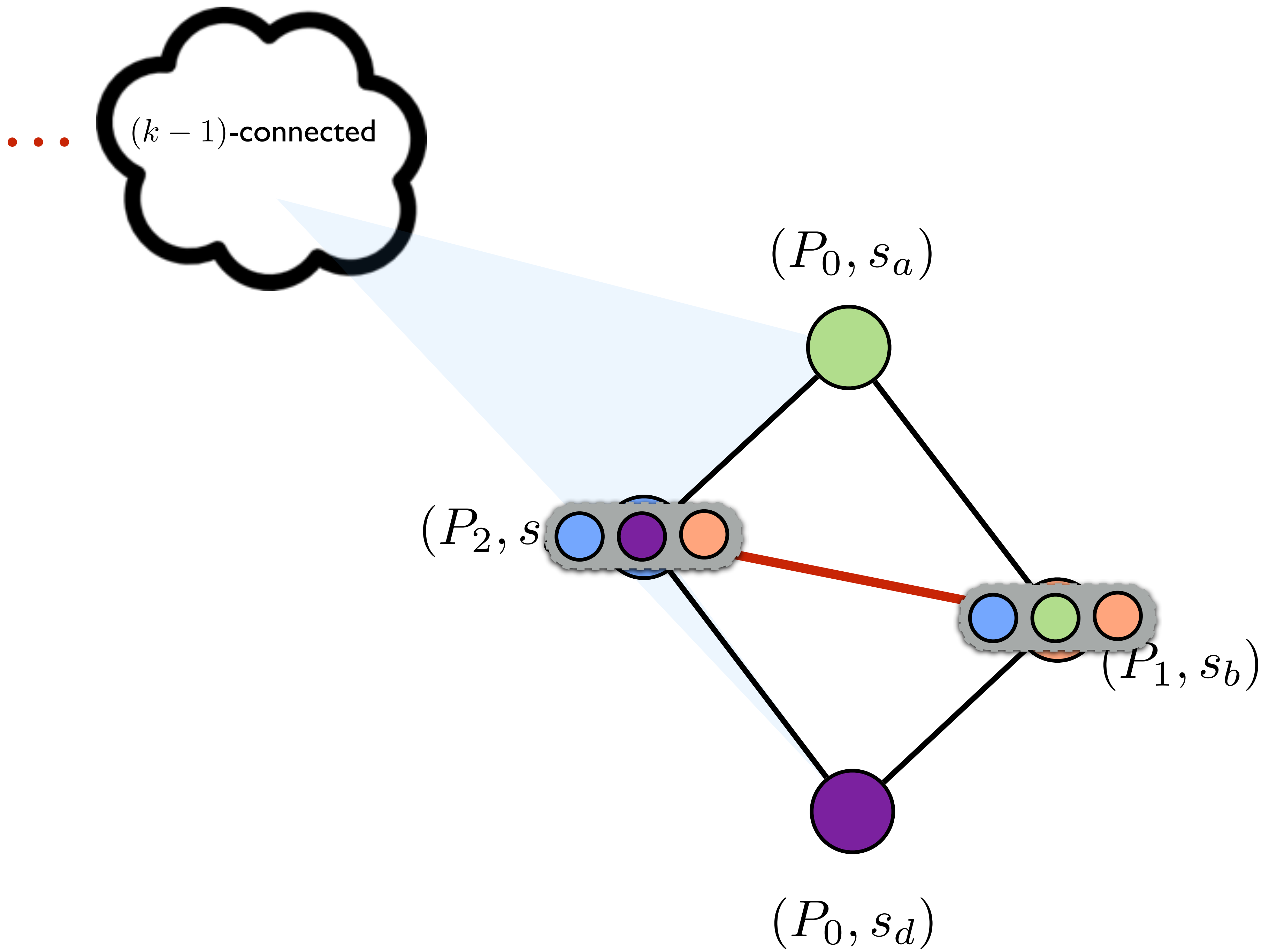


The Equivocation Operator



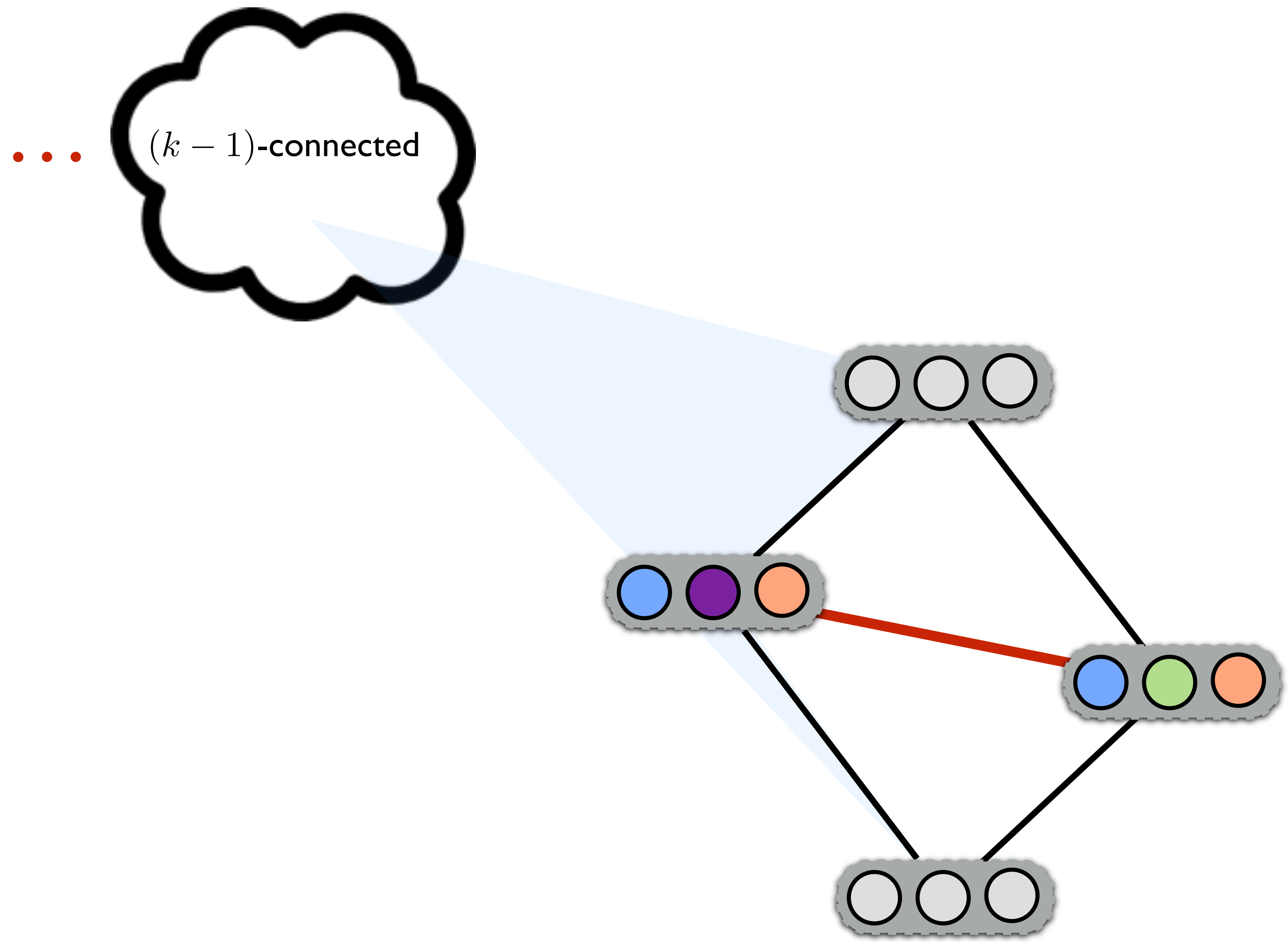
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round



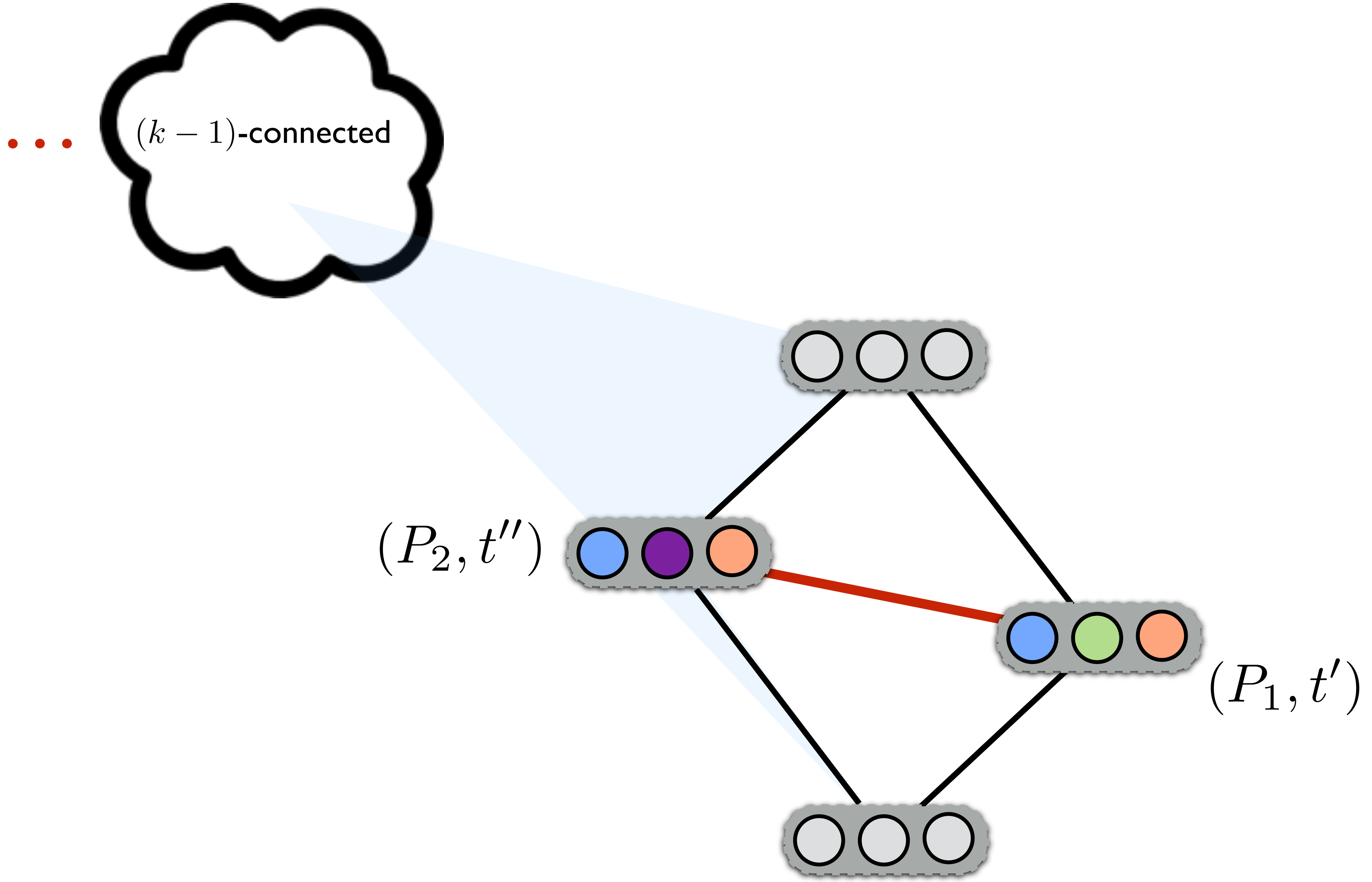
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round



The Equivocation Operator

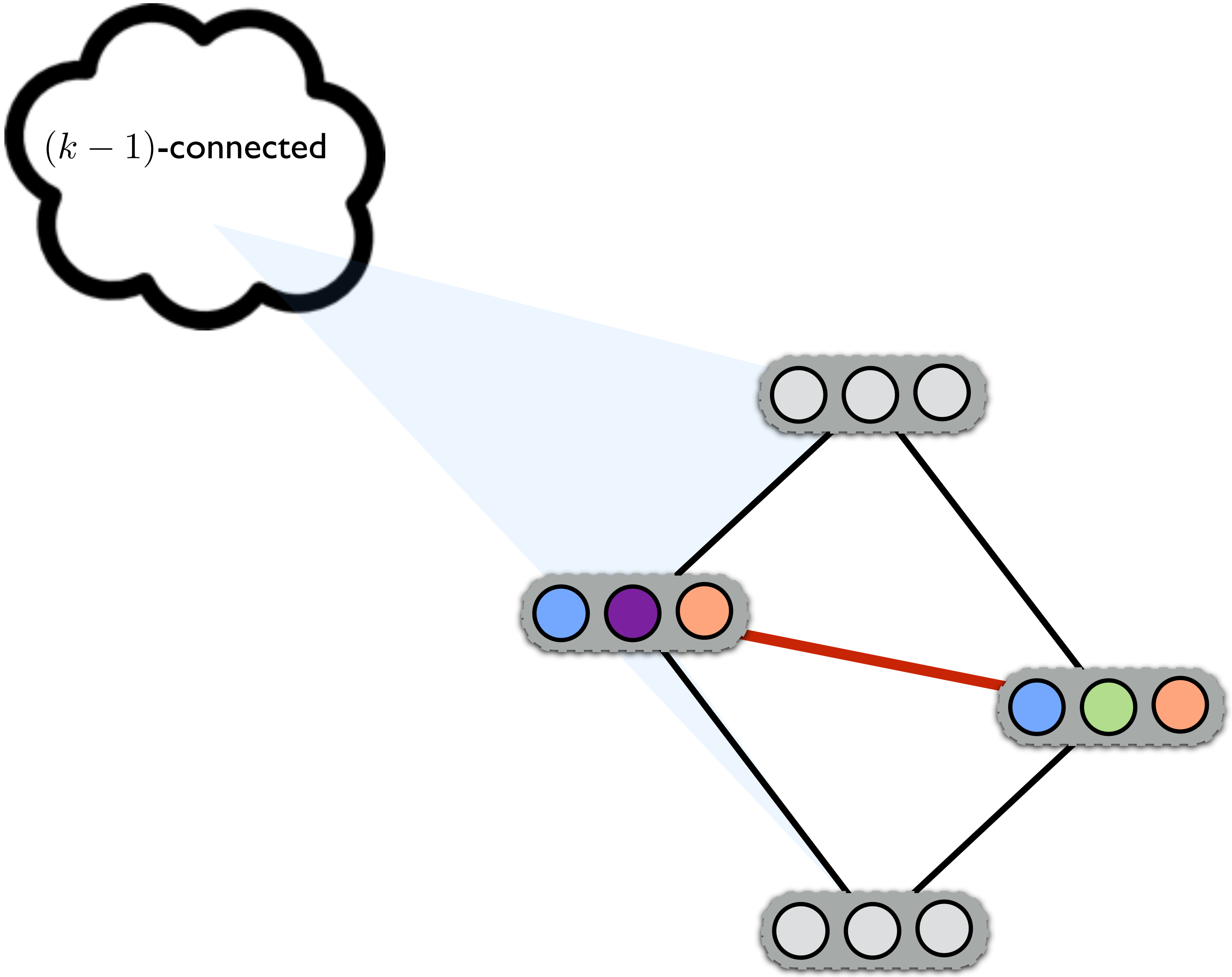
$\lfloor t/k \rfloor$ rounds,
k failures/round



The Equivocation Operator

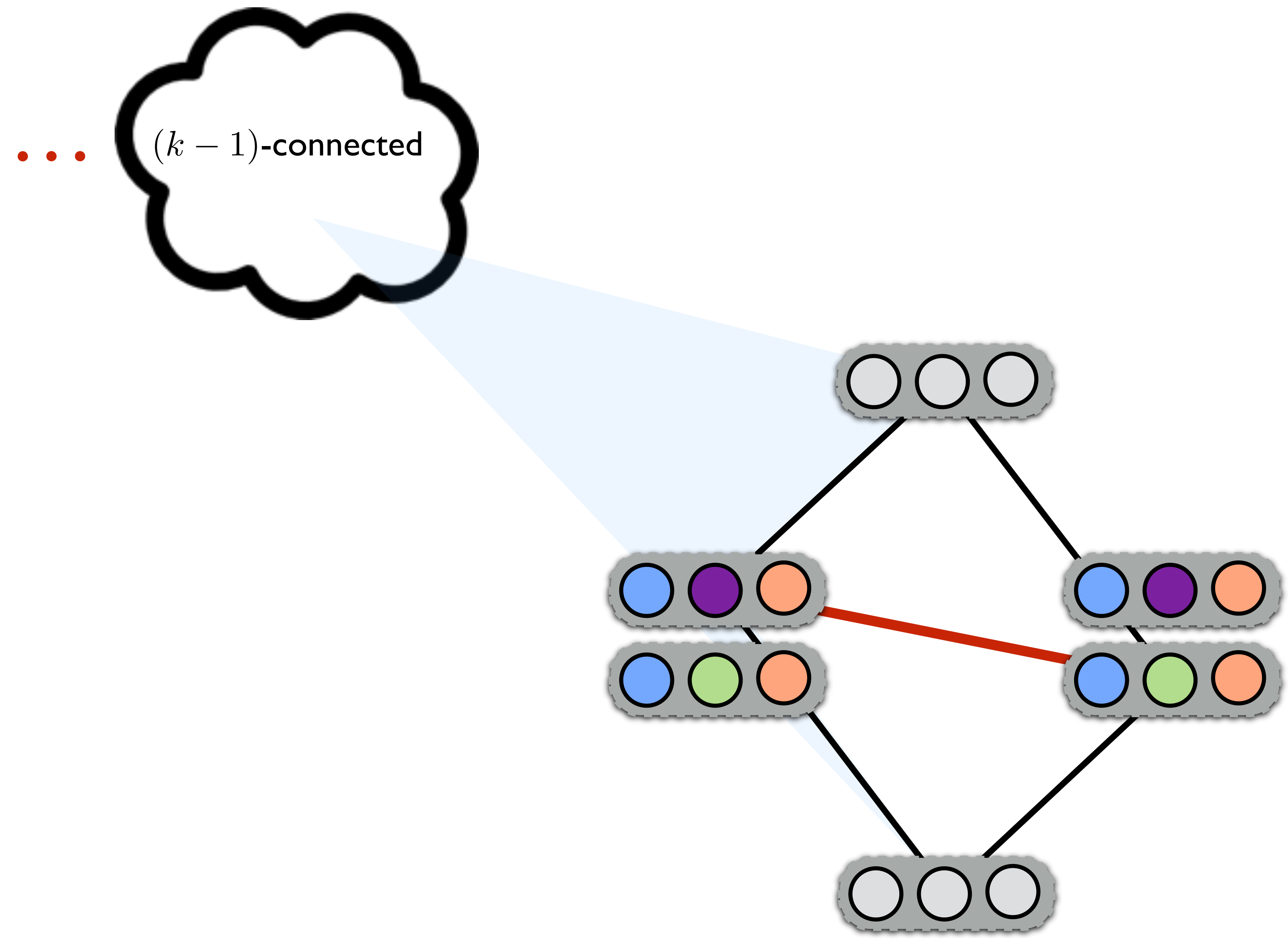
$\lfloor t/k \rfloor$ rounds,
k failures/round

. . .



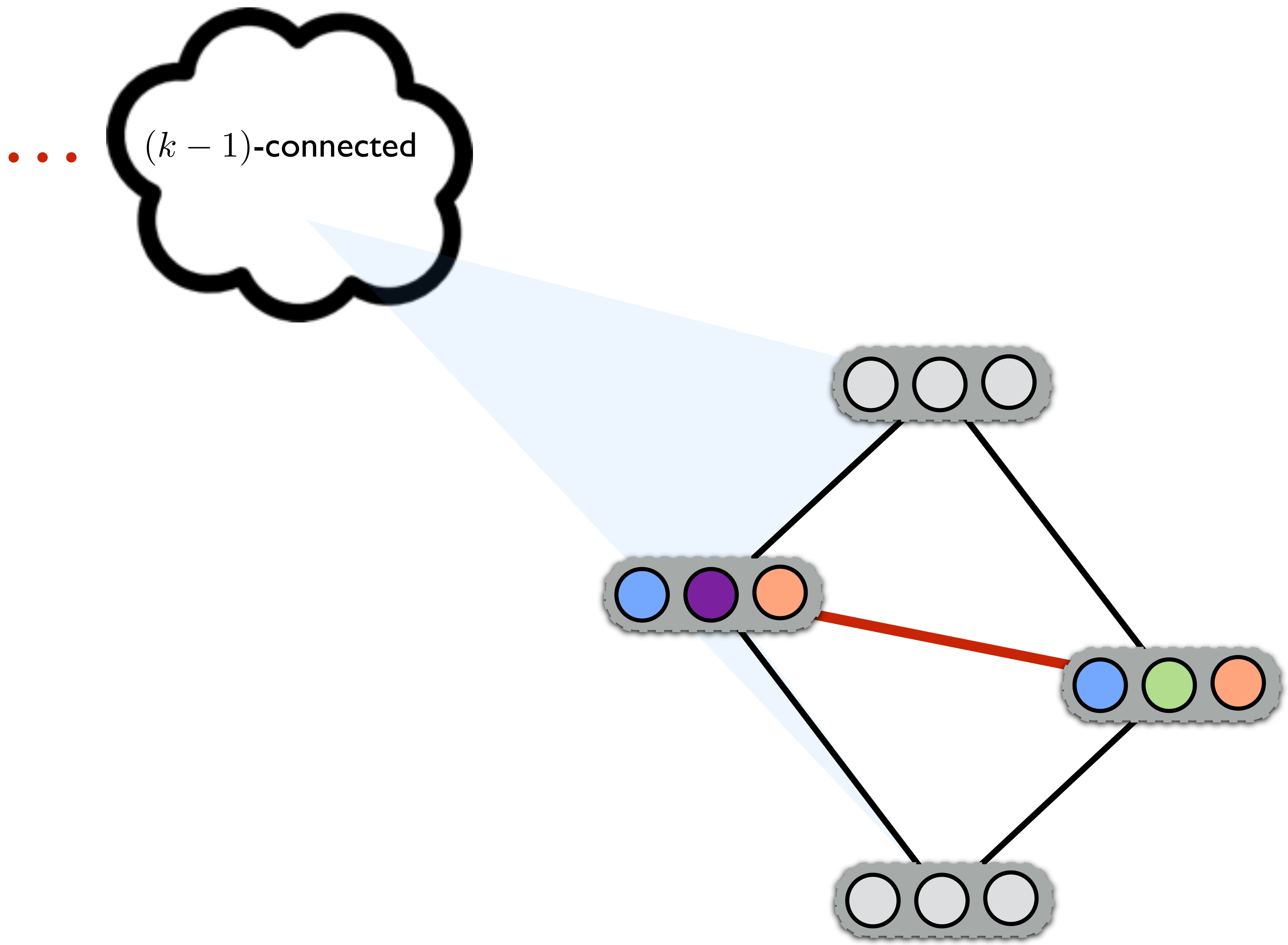
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round



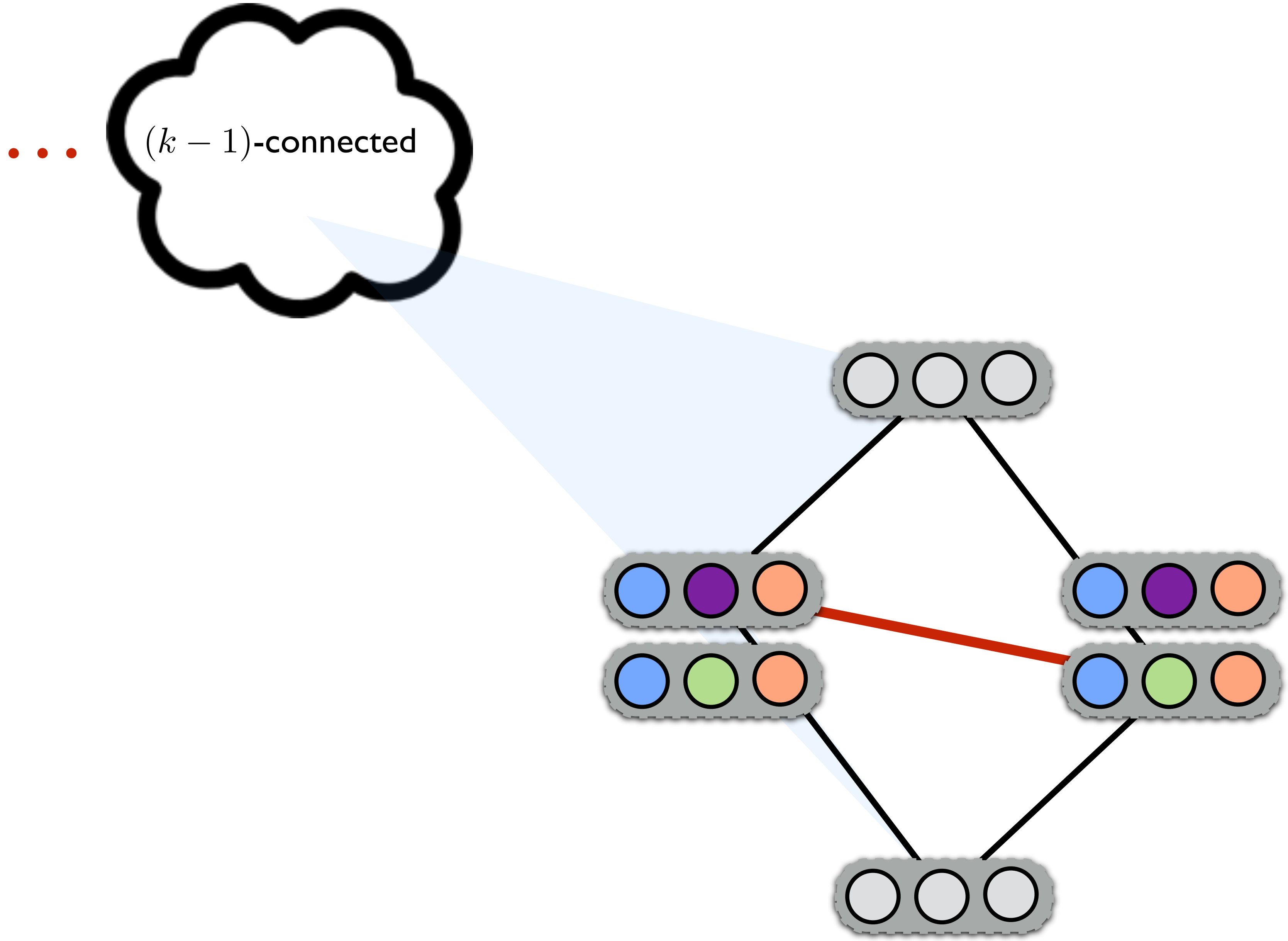
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round



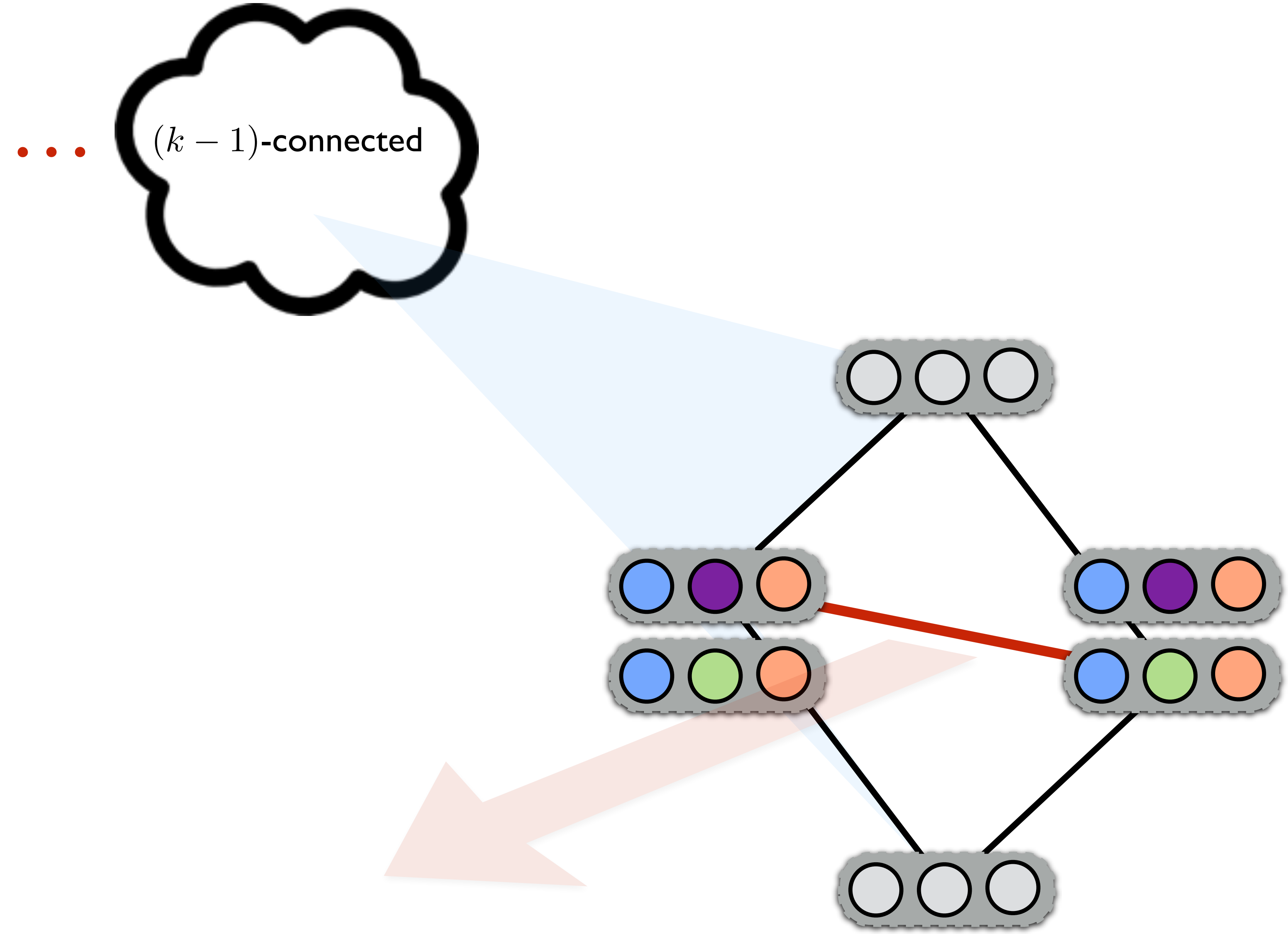
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round

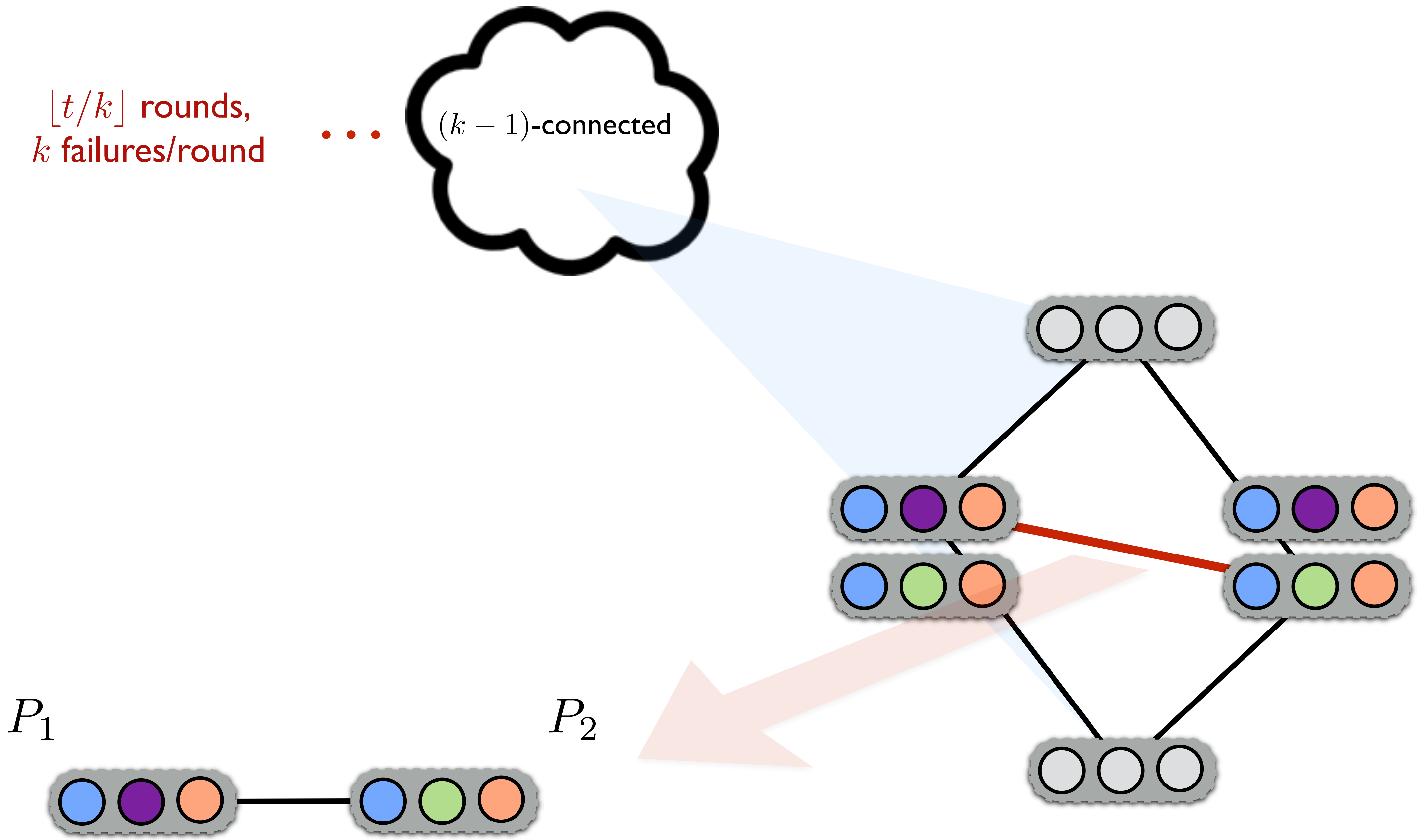


The Equivocation Operator

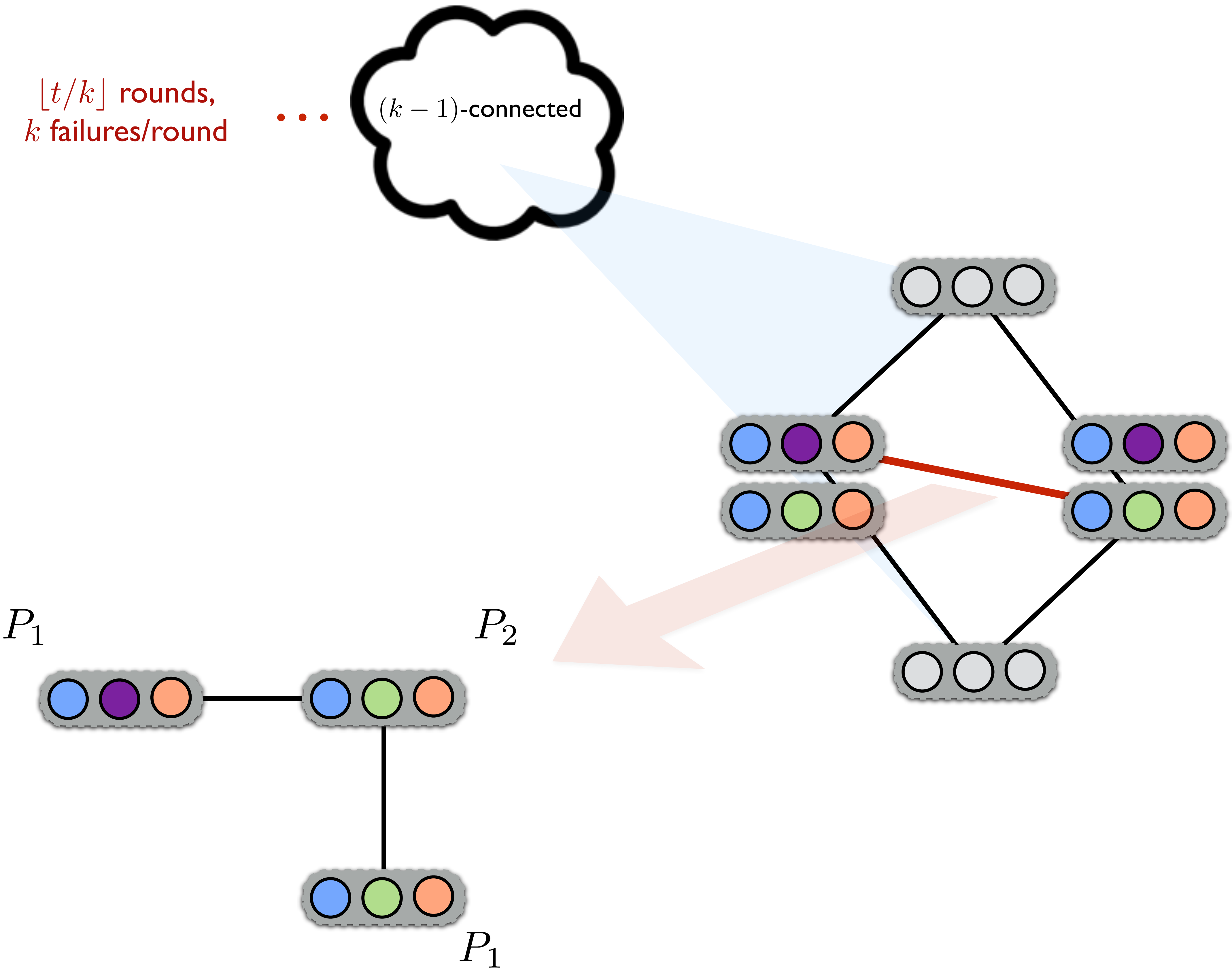
$\lfloor t/k \rfloor$ rounds,
k failures/round



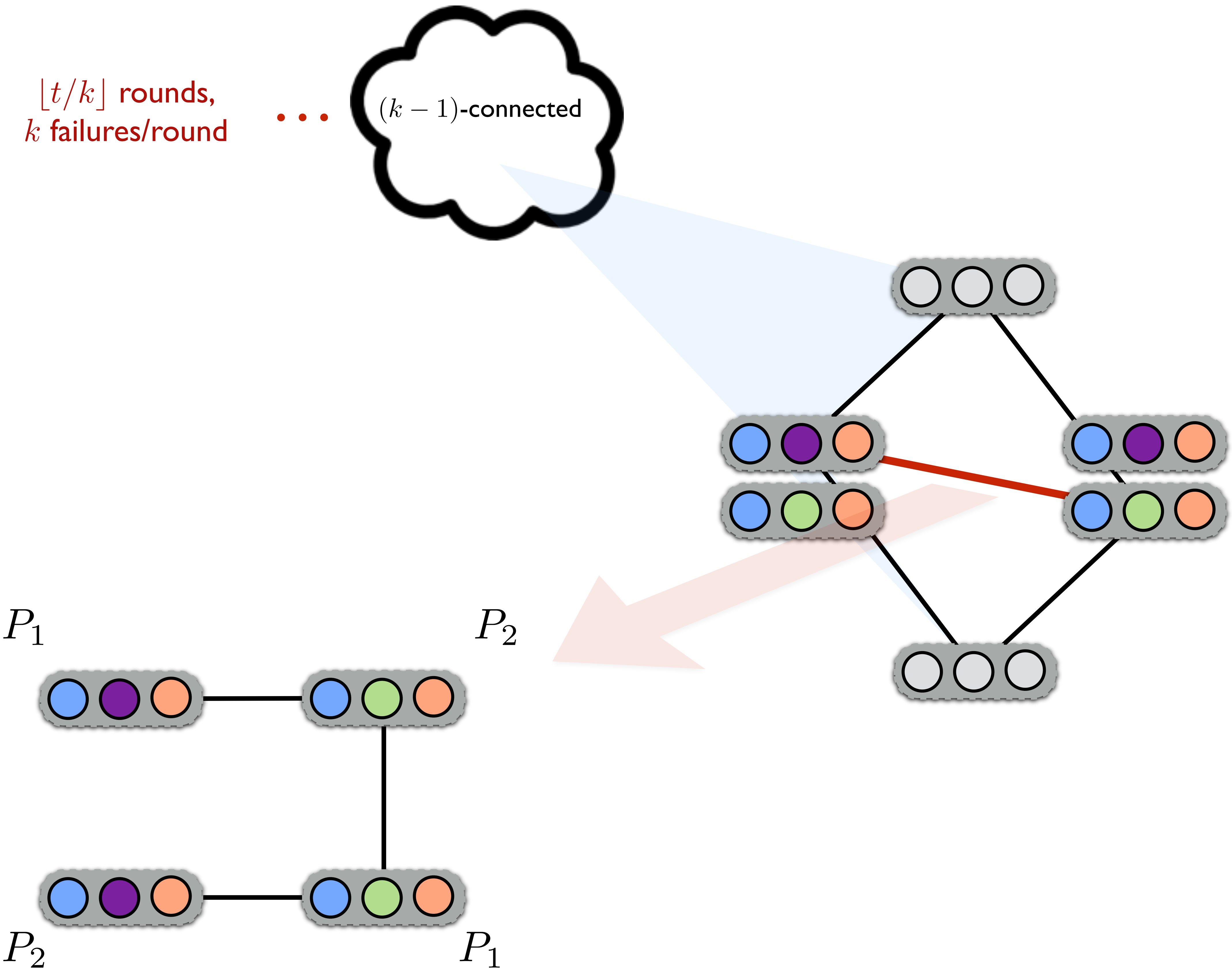
The Equivocation Operator



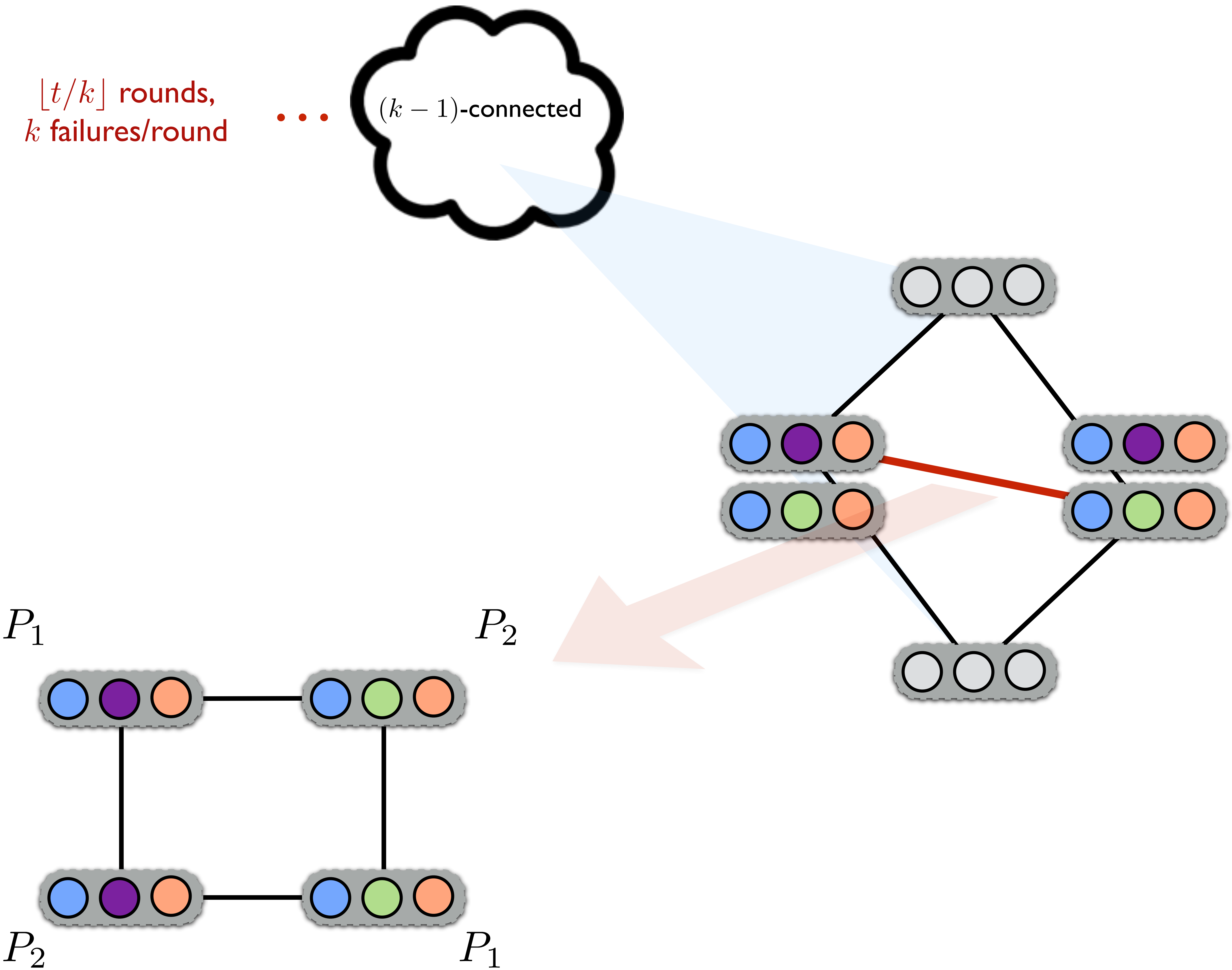
The Equivocation Operator



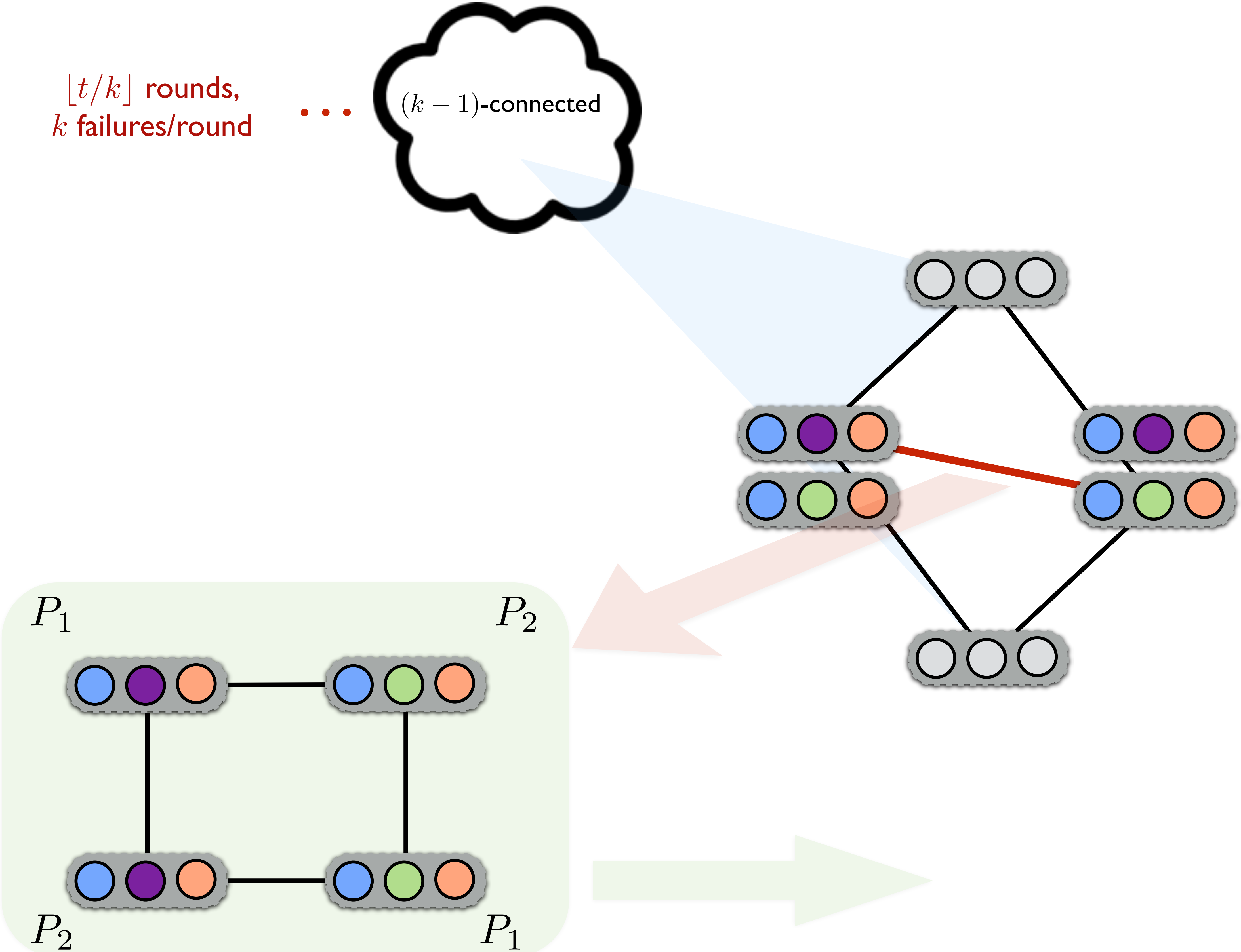
The Equivocation Operator



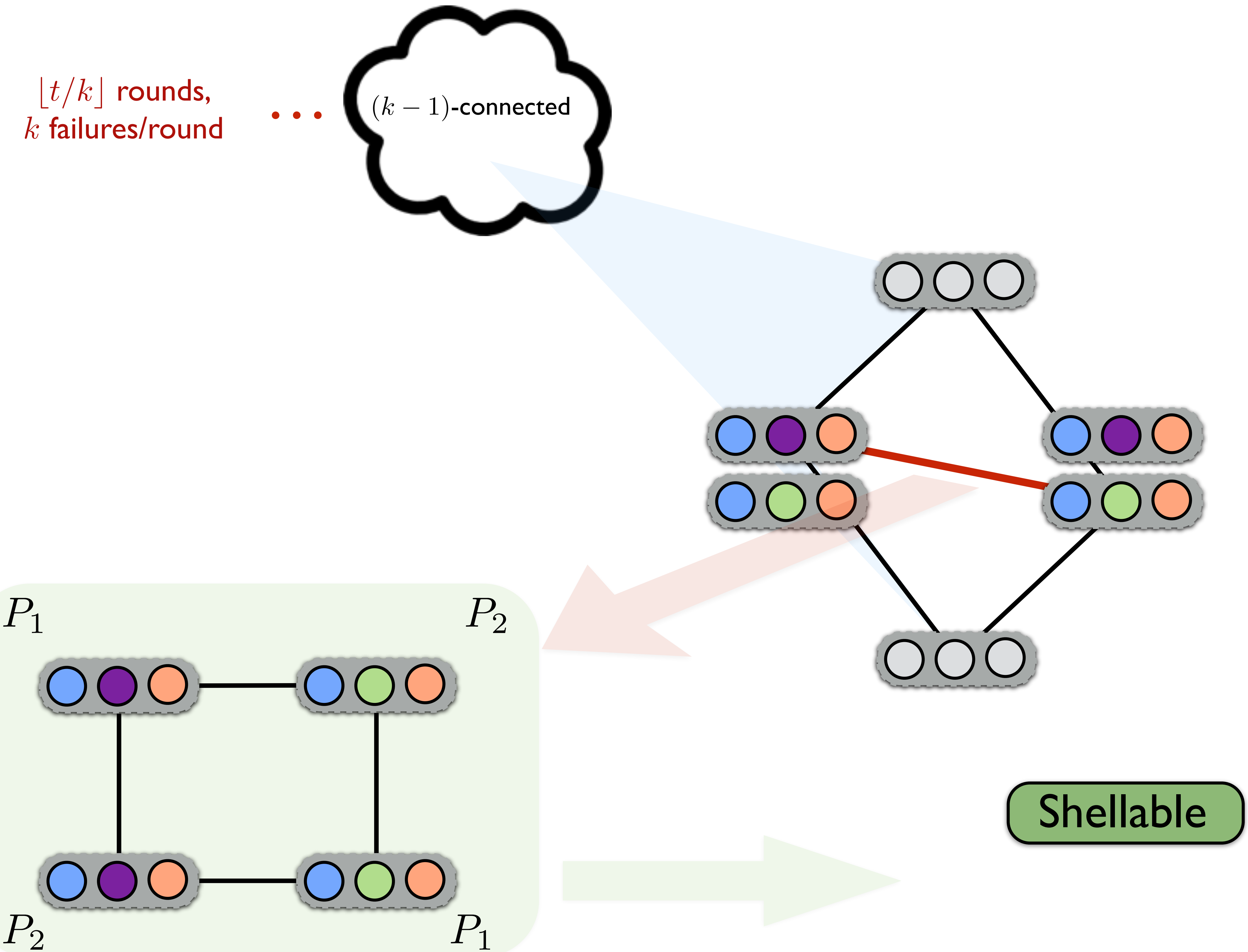
The Equivocation Operator



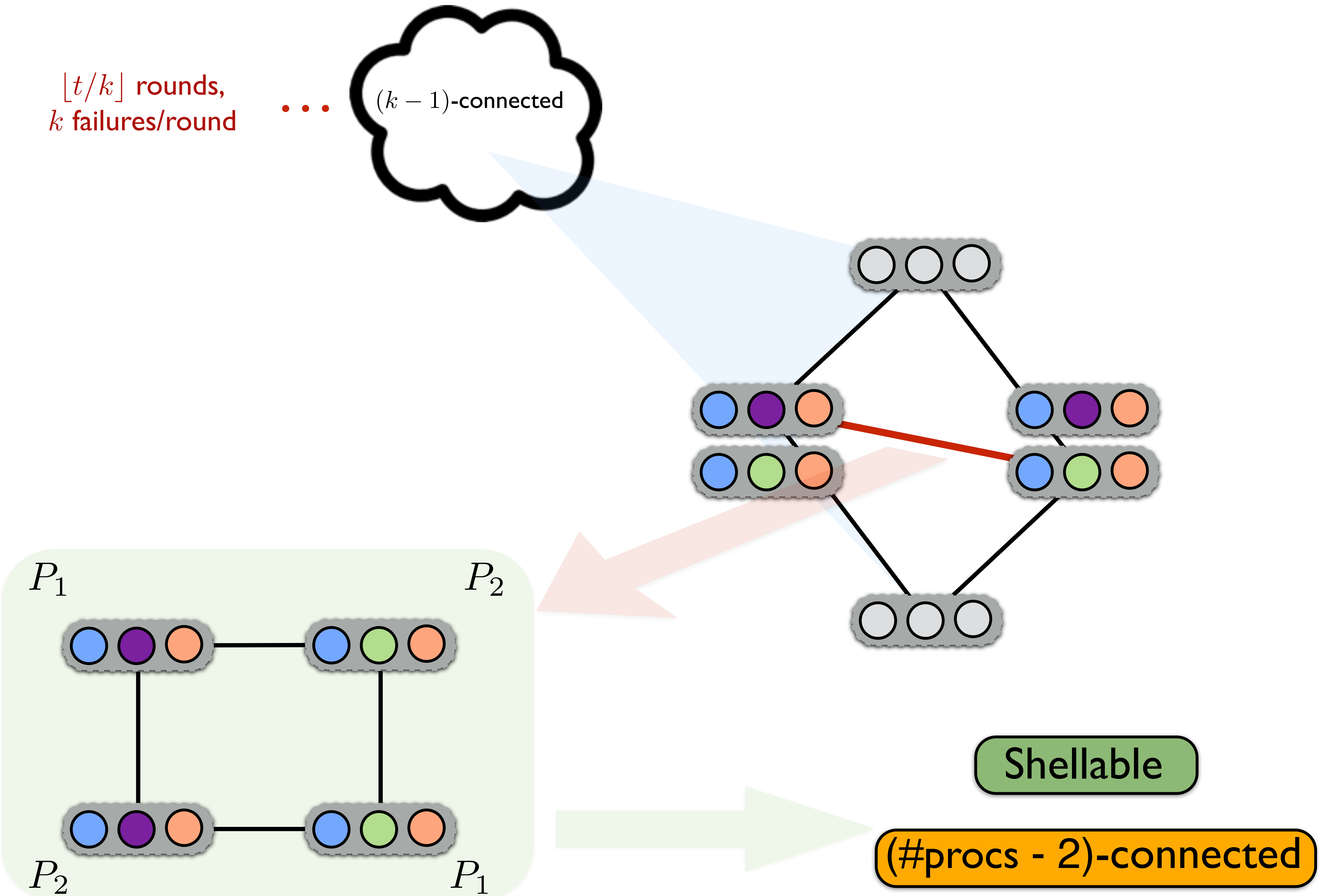
The Equivocation Operator



The Equivocation Operator

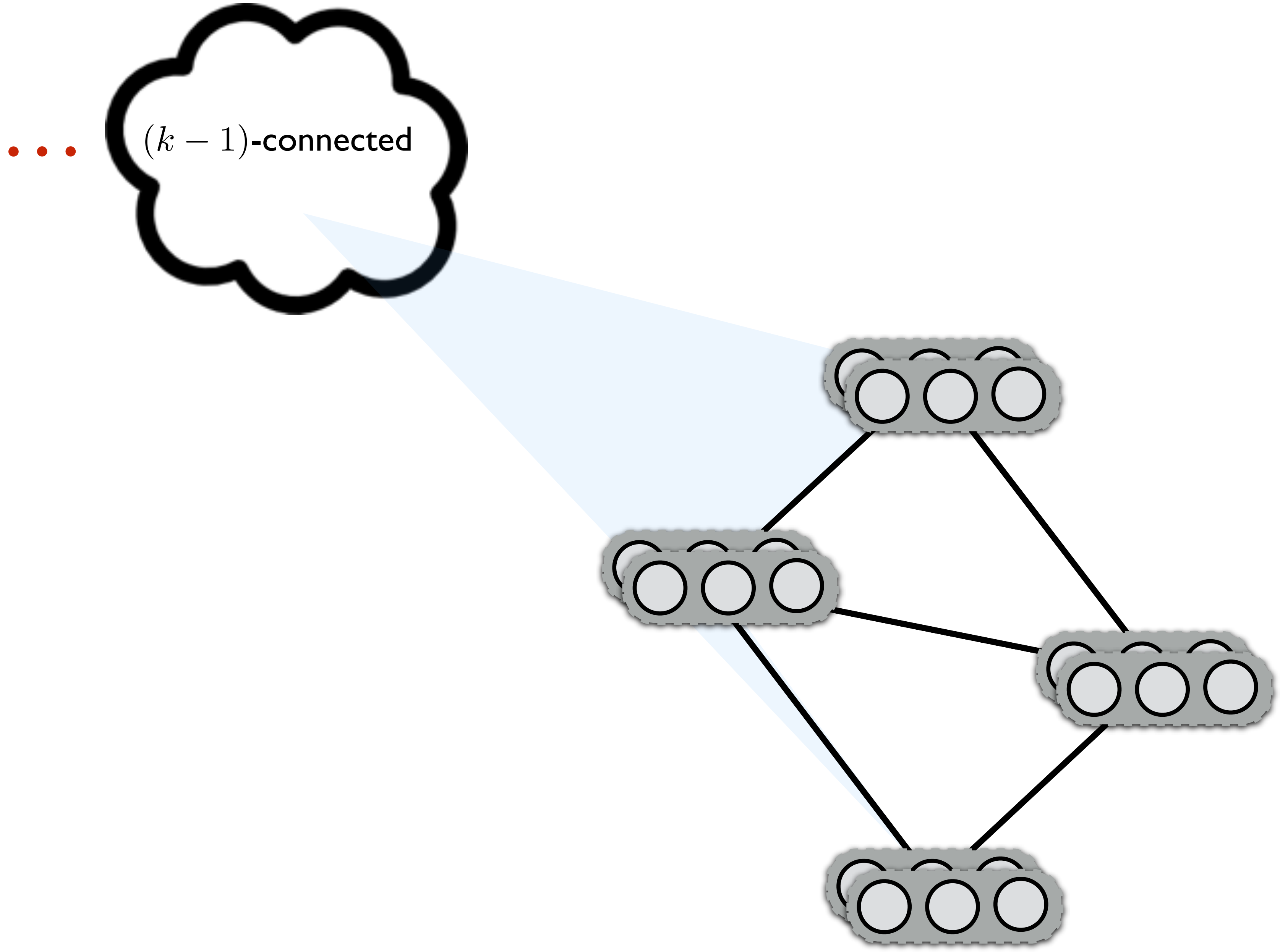


The Equivocation Operator



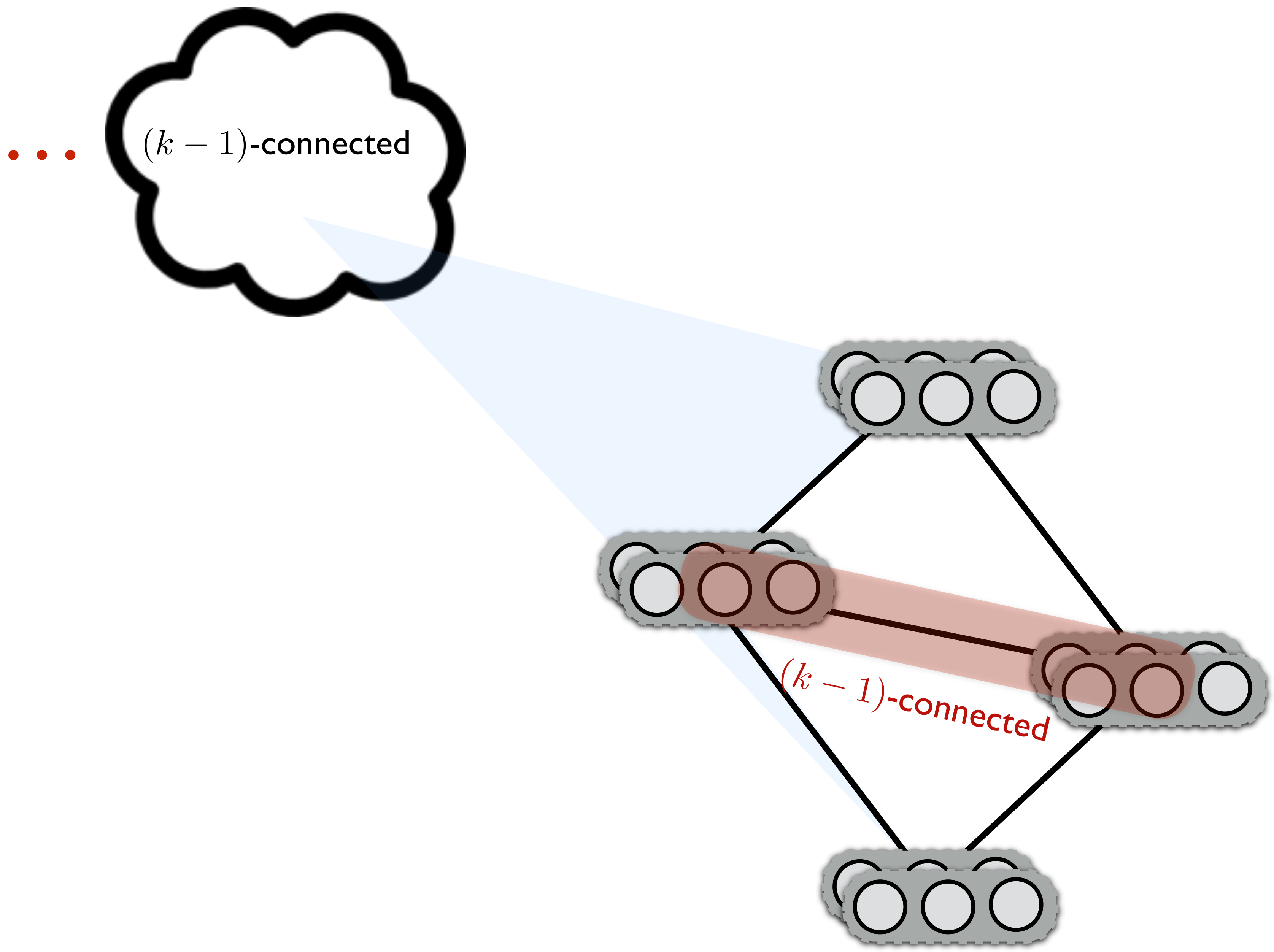
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round



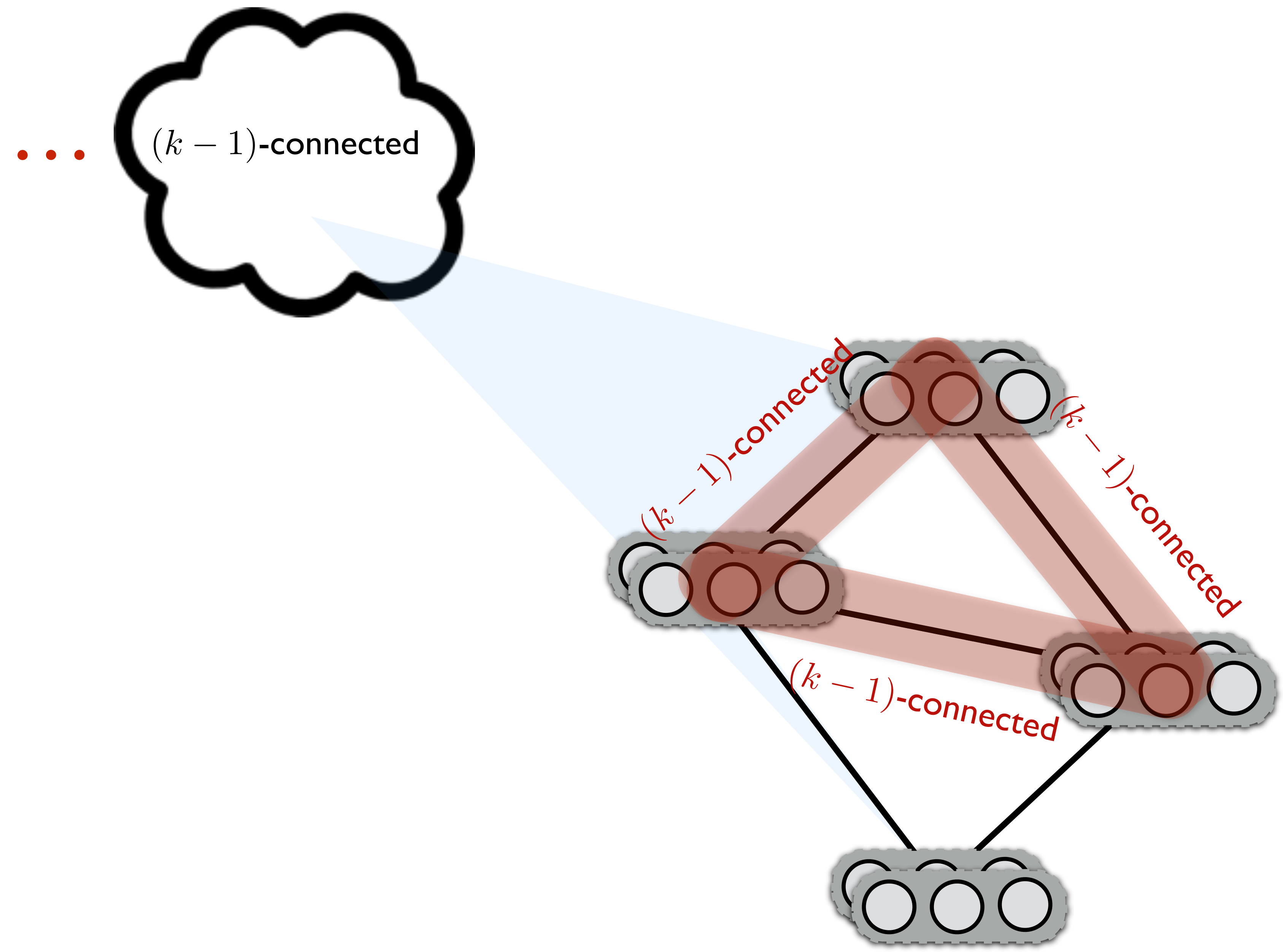
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round



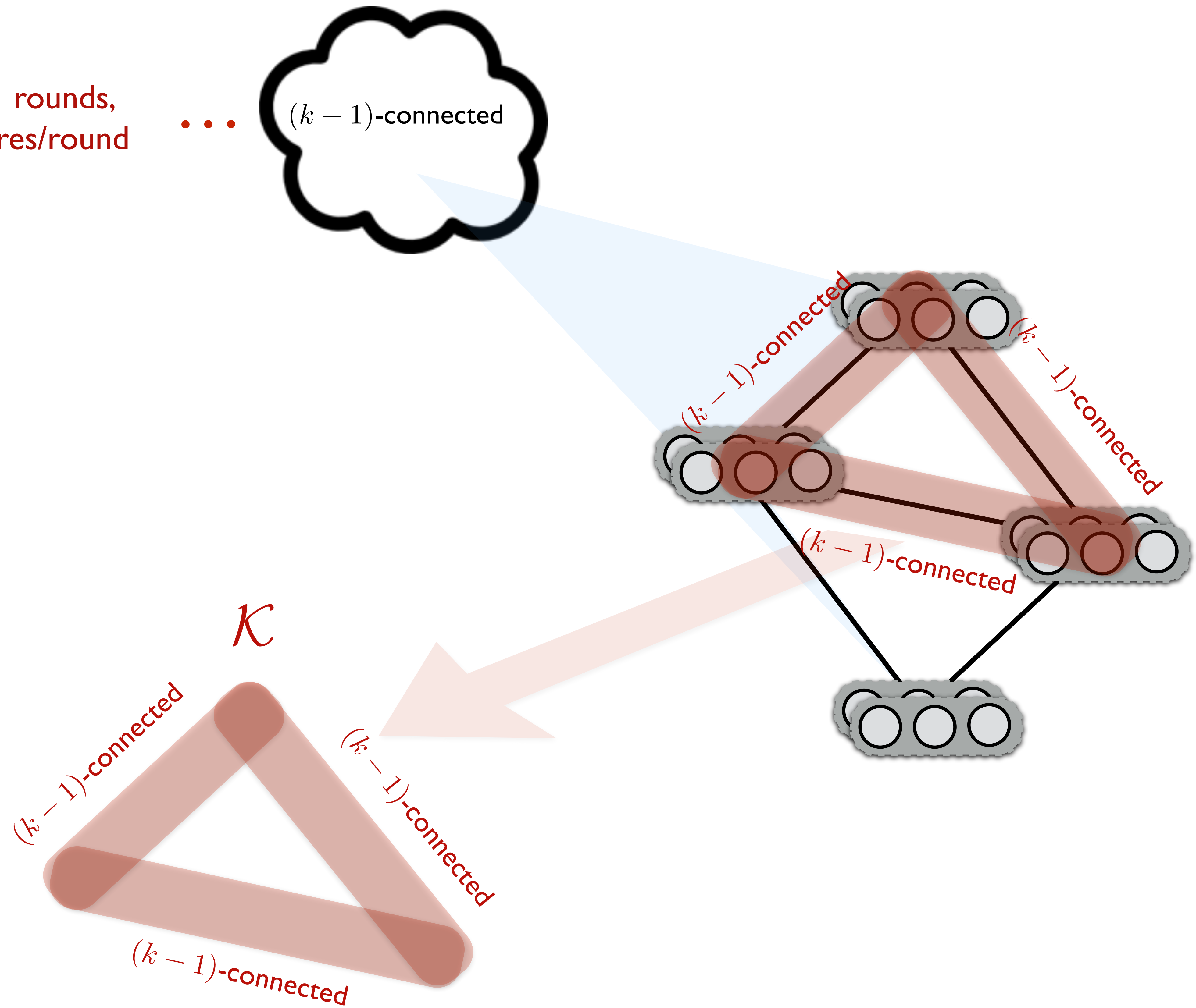
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round



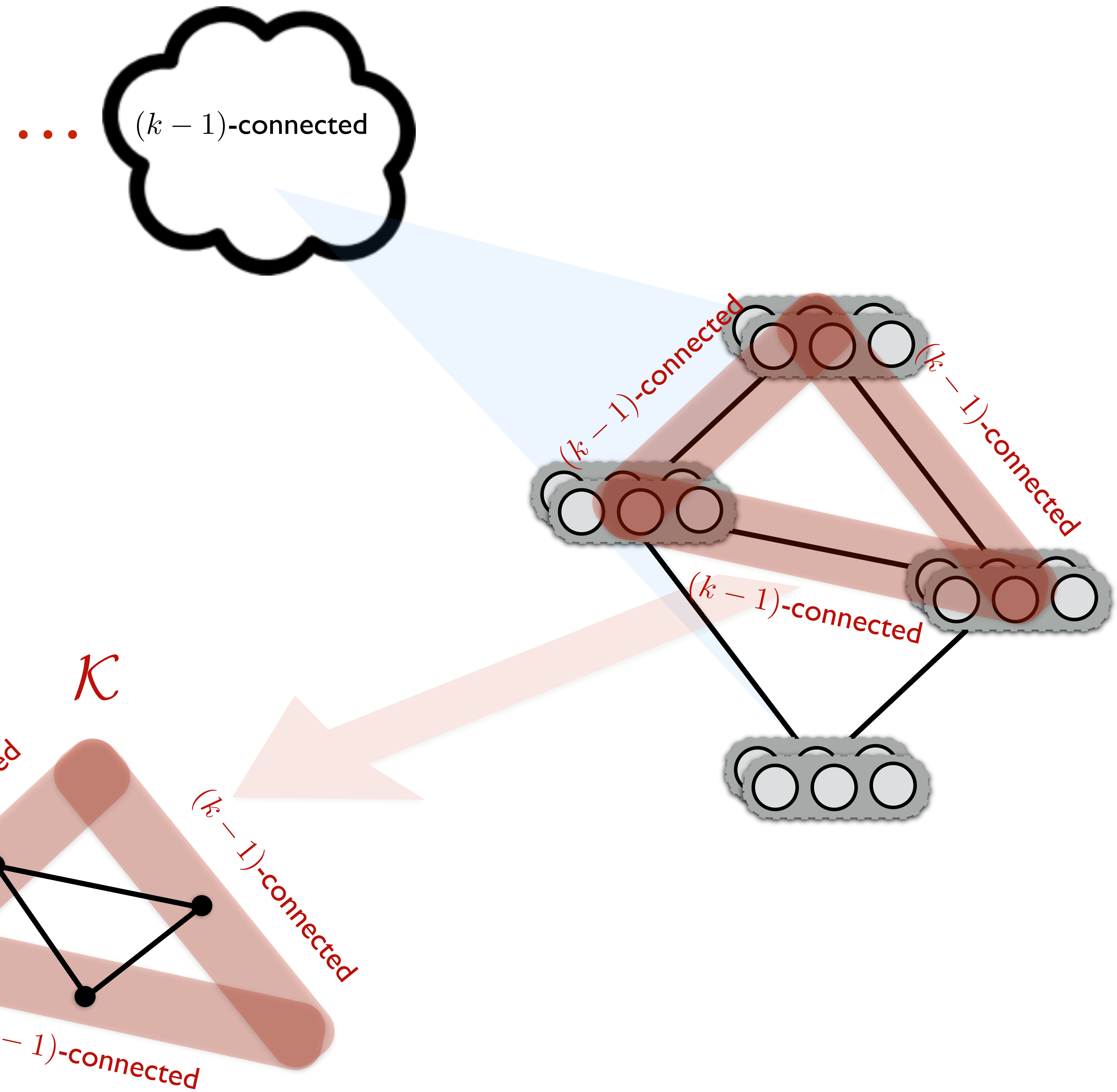
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round

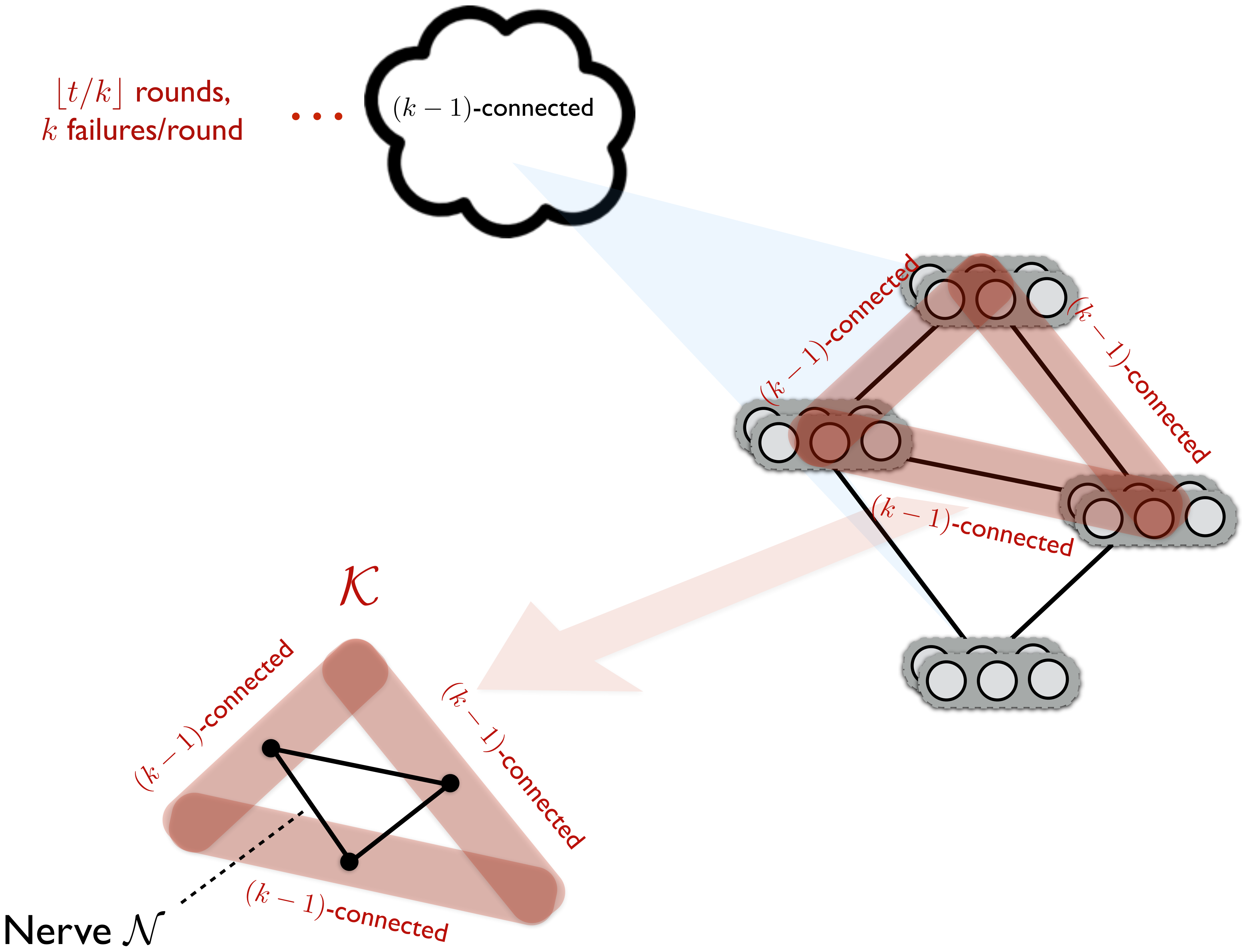


The Equivocation Operator

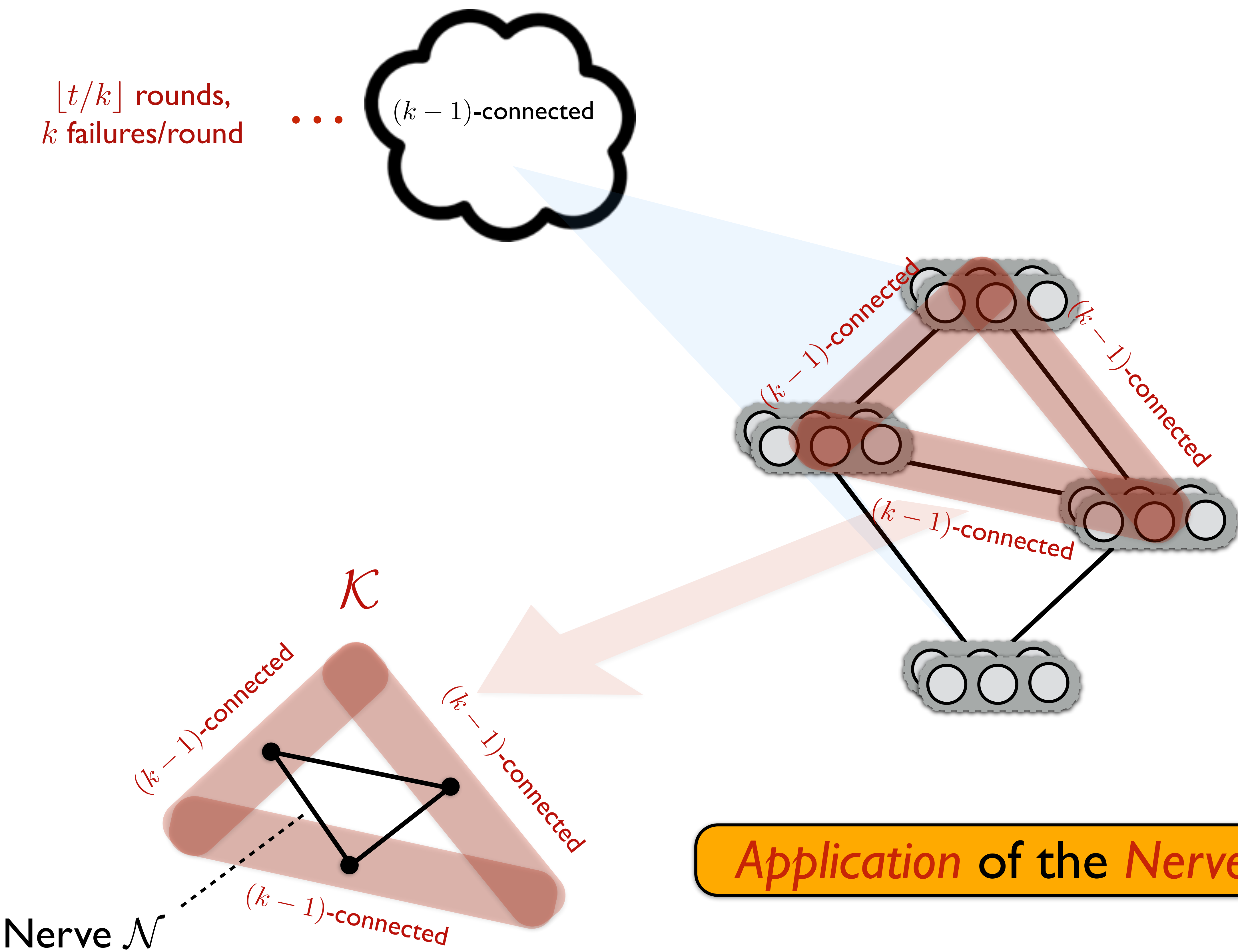
$\lfloor t/k \rfloor$ rounds,
k failures/round



The Equivocation Operator



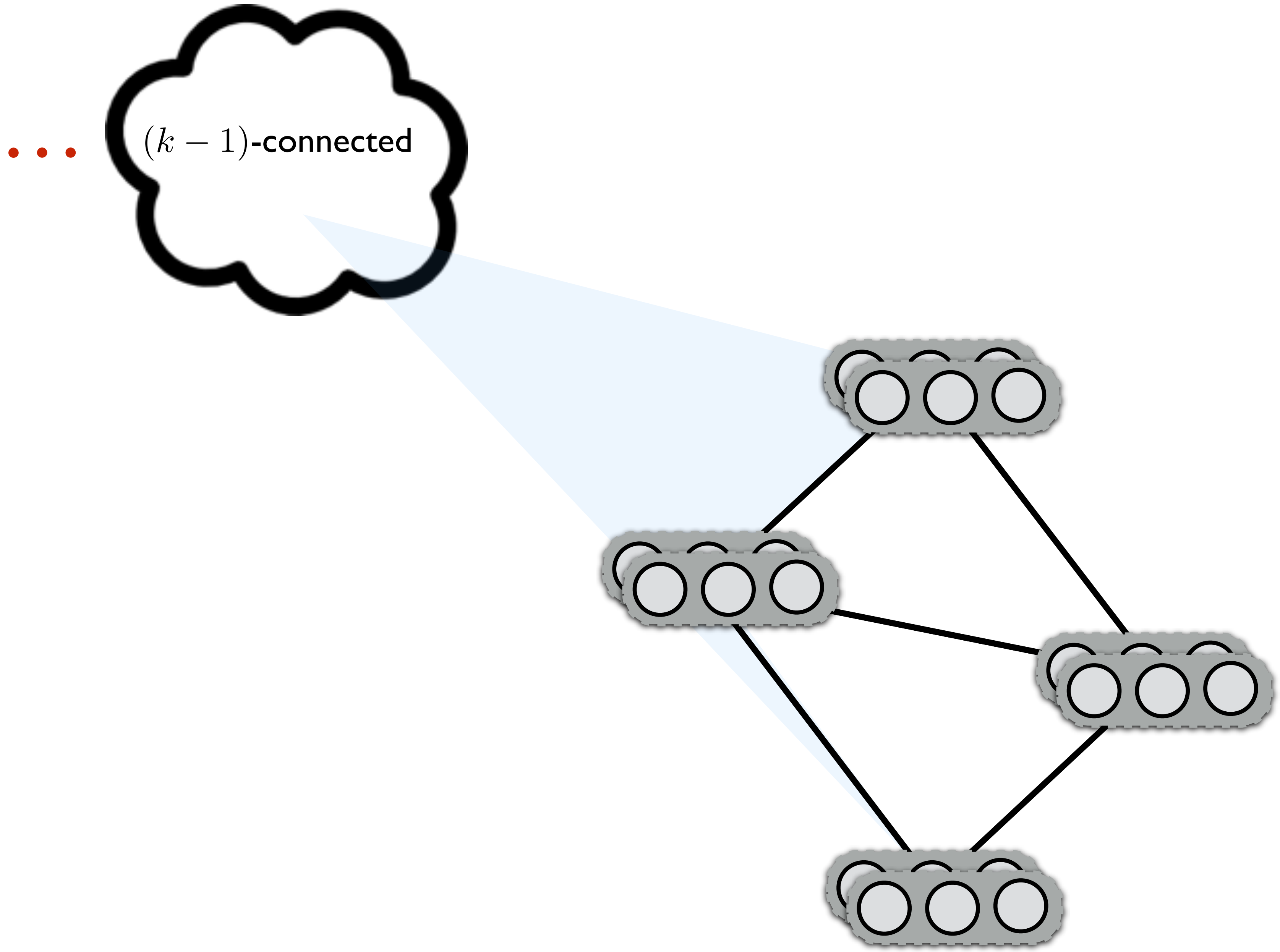
The Equivocation Operator



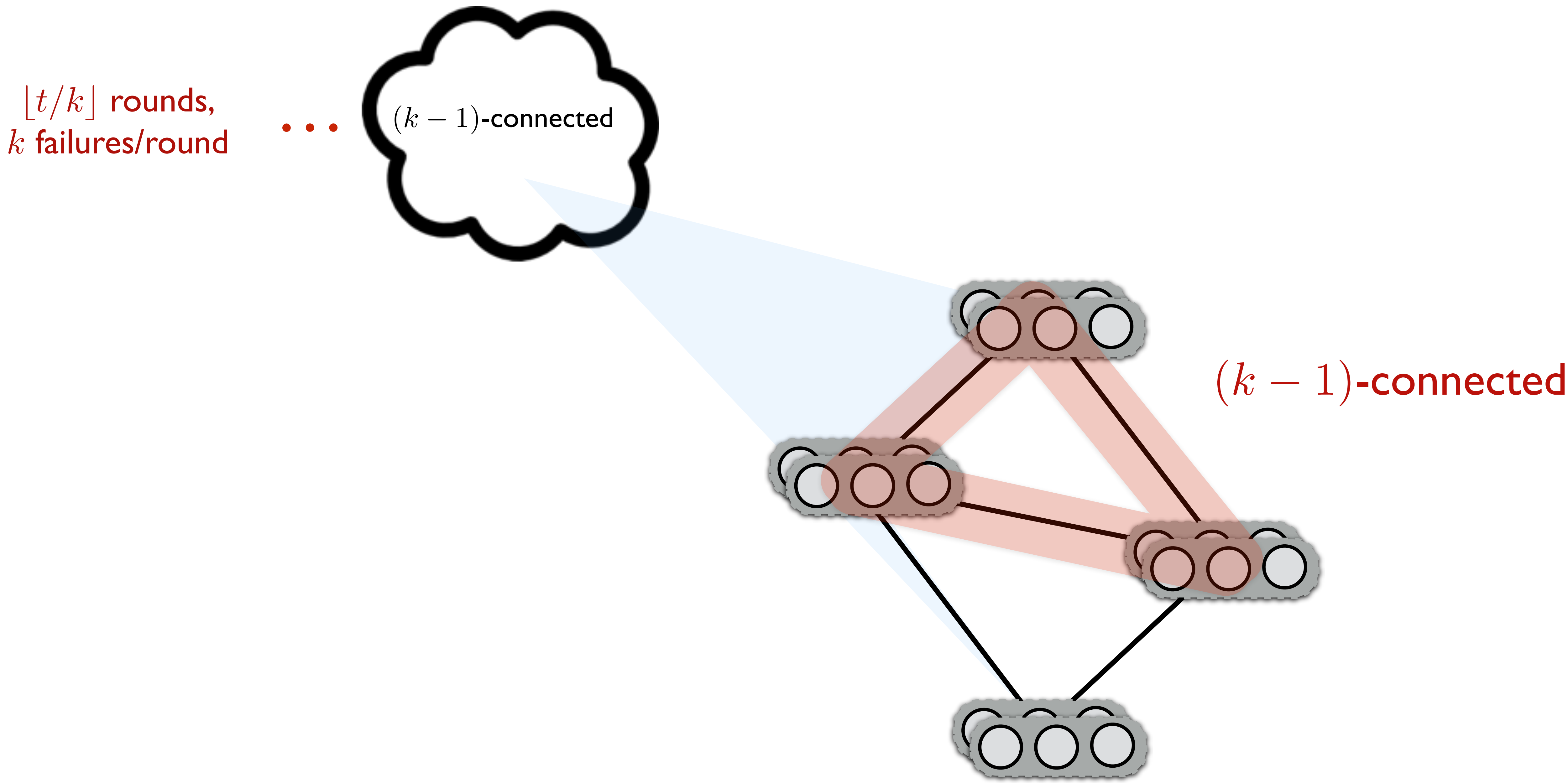
Application of the *Nerve Lemma*

The Equivocation Operator

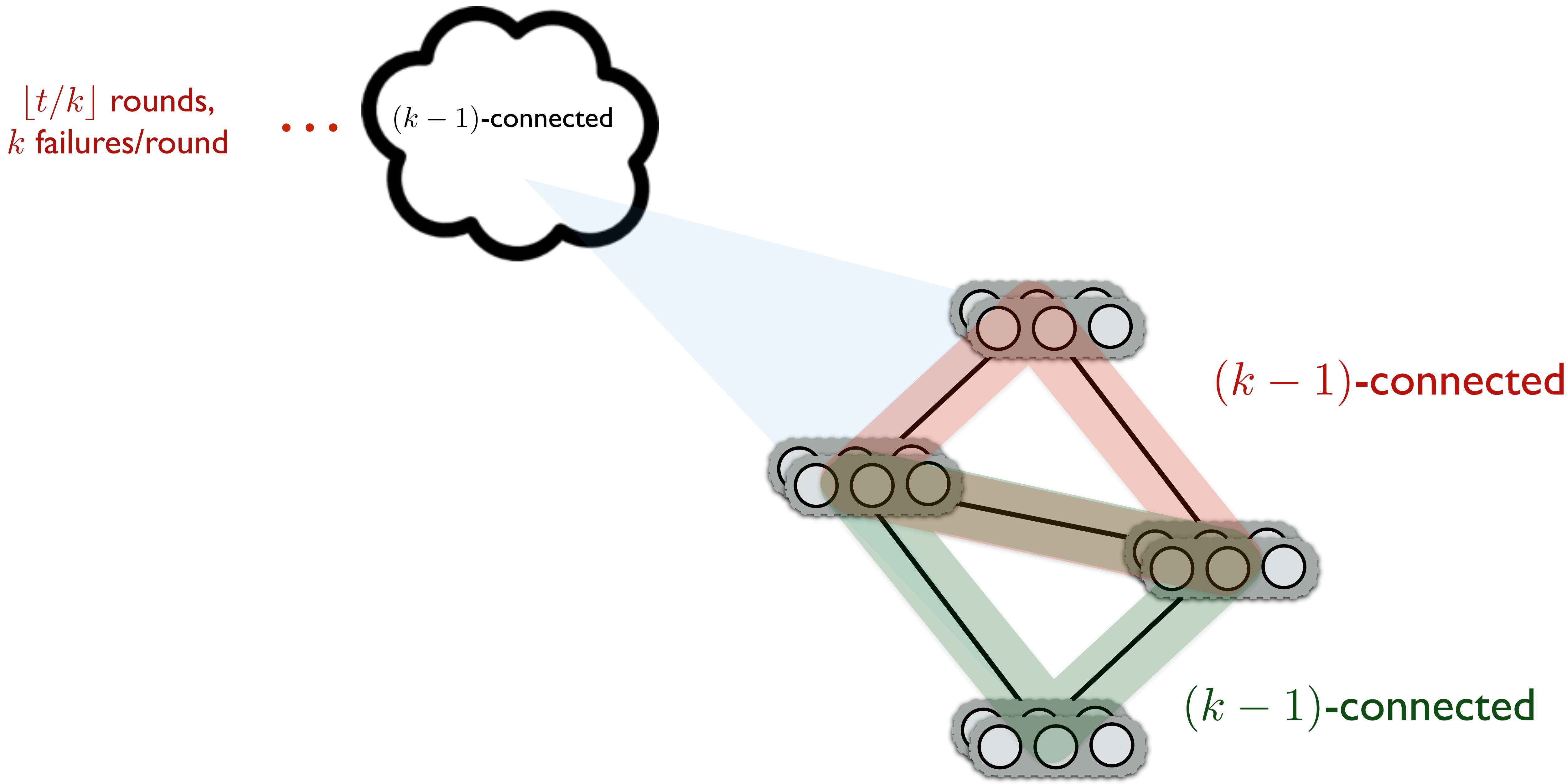
$\lfloor t/k \rfloor$ rounds,
k failures/round



The Equivocation Operator

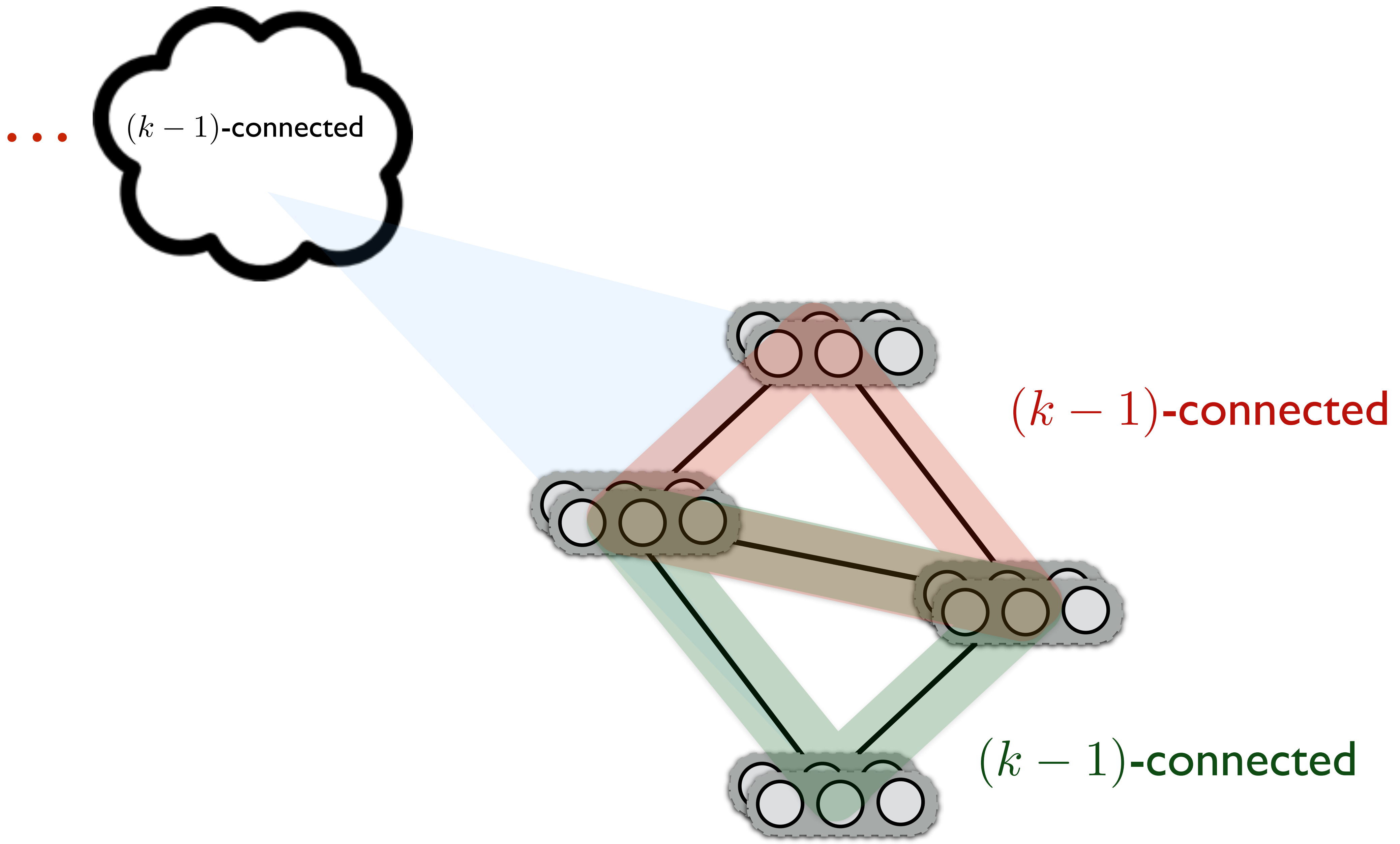


The Equivocation Operator



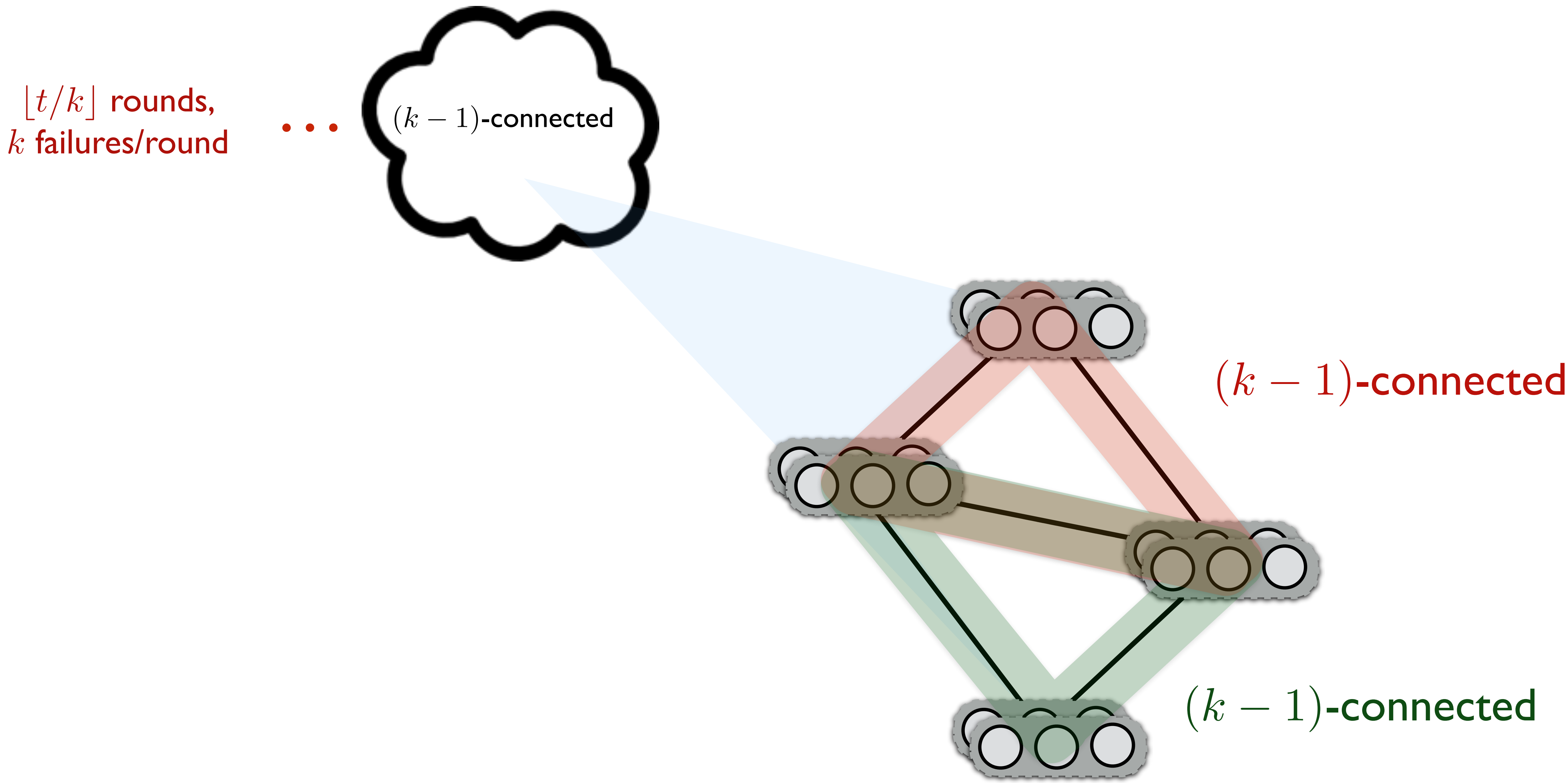
The Equivocation Operator

$\lfloor t/k \rfloor$ rounds,
k failures/round



Extend throughout structure

The Equivocation Operator



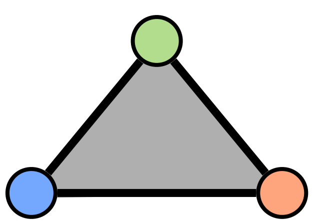
Extend throughout structure

Subsequent applications of the Nerve Lemma

Strategy, Again

We match the bound with an *algorithm*

$\mathcal{K}_0 = \mathcal{I}^*$ $\quad$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\quad$ $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\quad$ $\mathcal{K}_3 = \mathcal{R}_e(\mathcal{K}_2)$



...

$(k-1)$-connected $\quad$ $(k-1)$-connected $\quad$ $(k-1)$-connected

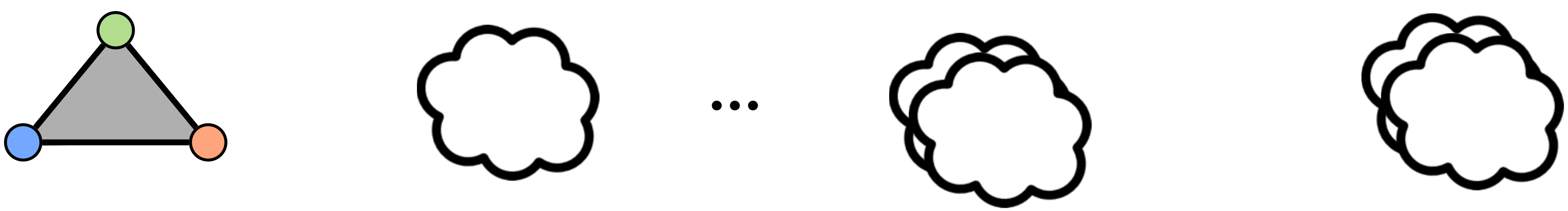
$\lfloor t/k \rfloor$ rounds $\quad$ if $t \bmod k \neq 0$

$\lceil t/k \rceil$ rounds

Strategy, Again

We match the bound with an *algorithm*

$\mathcal{K}_0 = \mathcal{I}^*$ $\quad \mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ $\quad \mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\quad \mathcal{K}_3 = \mathcal{R}_e(\mathcal{K}_2)$



$(k-1)$-connected $\quad (k-1)$-connected $\quad (k-1)$-connected

$\lfloor t/k \rfloor$ rounds $\quad$ if $t \bmod k \neq 0$

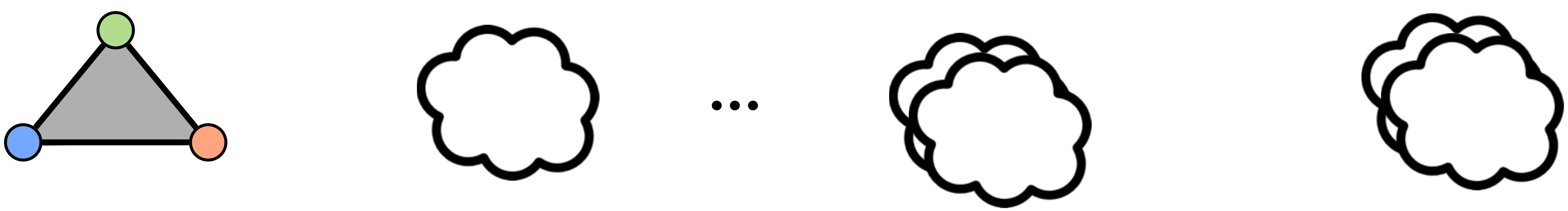
$\lceil t/k \rceil$ rounds

k-set agreement protocol

Strategy, Again

We match the bound with an *algorithm*

$\mathcal{K}_0 = \mathcal{I}^*$ $\mathcal{K}_1 = \mathcal{R}_c(\mathcal{K}_0, k)$ ... $\mathcal{K}_2 = \mathcal{R}_c(\mathcal{K}_1, k)$ $\mathcal{K}_3 = \mathcal{R}_e(\mathcal{K}_2)$



$(k-1)$-connected $(k-1)$-connected $(k-1)$-connected

$\lfloor t/k \rfloor$ rounds

if $t \bmod k \neq 0$

$\lceil t/k \rceil$ rounds

k-set agreement protocol

Generalize the consensus protocol

Outline

1. Introduction
2. Asynchronous Byzantine Systems
3. Synchronous Byzantine Systems
4. Conclusion & Future Work

Outline

1. Introduction
2. Asynchronous Byzantine Systems
3. Synchronous Byzantine Systems
4. Conclusion & Future Work

Conclusion

Conclusion

Conclusion

- Asynchronous Byzantine computability by *reduction*

Conclusion

- Asynchronous Byzantine computability by *reduction*

  1. Algorithmic primitives incorporated into the model

Conclusion

- Asynchronous Byzantine computability by *reduction*

1. Algorithmic primitives incorporated into the model

    - Reliable Broadcast

Conclusion

- Asynchronous Byzantine computability by *reduction*
  1. Algorithmic primitives incorporated into the model
     - Reliable Broadcast
  2. A "layer of interpretation" that empowers the model

Conclusion

- Asynchronous Byzantine computability by *reduction*

1. Algorithmic primitives incorporated into the model
    - Reliable Broadcast
2. A "layer of interpretation" that empowers the model
3. Mix of topological/algorithmic arguments, results fundamentally topological

Conclusion

- Asynchronous Byzantine computability by *reduction*
  1. Algorithmic primitives incorporated into the model
     - Reliable Broadcast
  2. A "layer of interpretation" that empowers the model
  3. Mix of topological/algorithmic arguments, results fundamentally topological

- Synchronous Byzantine computability by *shellability*

Conclusion

- Asynchronous Byzantine computability by *reduction*
  1. Algorithmic primitives incorporated into the model
     - Reliable Broadcast
  2. A "layer of interpretation" that empowers the model
  3. Mix of topological/algorithmic arguments, results fundamentally topological

- Synchronous Byzantine computability by *shellability*
  1. Different layer of interpretation

Conclusion

- Asynchronous Byzantine computability by *reduction*
  1. Algorithmic primitives incorporated into the model
     - Reliable Broadcast
  2. A "layer of interpretation" that empowers the model
  3. Mix of topological/algorithmic arguments, results fundamentally topological

- Synchronous Byzantine computability by *shellability*
  1. Different layer of interpretation
     - Round-by-round interpretation of messages

Conclusion

- Asynchronous Byzantine computability by *reduction*
  1. Algorithmic primitives incorporated into the model
     - Reliable Broadcast
  2. A “layer of interpretation” that empowers the model
  3. Mix of topological/algorithmic arguments, results fundamentally topological

- Synchronous Byzantine computability by *shellability*
  1. Different layer of interpretation
     - Round-by-round interpretation of messages
     - We cannot validate messages from the last round

Conclusion

- Asynchronous Byzantine computability by *reduction*
  1. Algorithmic primitives incorporated into the model
     - Reliable Broadcast
  2. A "layer of interpretation" that empowers the model
  3. Mix of topological/algorithmic arguments, results fundamentally topological

- Synchronous Byzantine computability by *shellability*
  1. Different layer of interpretation
     - Round-by-round interpretation of messages
     - We cannot validate messages from the last round
  2. Topological upper bound, algorithmic lower bound

Future Work

Future Work (i.e. Research Questions)

Future Work (i.e. Research Questions)

- Randomized Protocols
  - Many *impossible* problems (in a deterministic setting) now become *possible*

Future Work (i.e. Research Questions)

- Randomized Protocols
  - Many *impossible* problems (in a deterministic setting) now become *possible*
- Complexity
  - Particularly in asynchronous systems
  - Proofs are not constructive

Future Work (i.e. Research Questions)

- Randomized Protocols
    - Many *impossible* problems (in a deterministic setting) now become *possible*
- Complexity
    - Particularly in asynchronous systems
    - Proofs are not constructive
- Failure Detectors
    - Allow us to detect the crash of peer processes
    - Again, many *impossible* problems now become *possible*

References

Published:

[1] Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, STOC'13, pages 391–400, New York, NY, USA, 2013. ACM.

[2] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and VijayK. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, pages 1–19, 2015.

[3] Hammurabi Mendes, Christine Tasson, and Maurice Herlihy. Distributed computability in Byzantine asynchronous systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 704–713, New York, NY, USA, 2014. ACM.

ArXiV:

Hammurabi Mendes, Maurice Herlihy. Tight Bounds for Connectivity and Set Agreement in Byzantine Synchronous Systems. arxiv.org/abs/1505.04224

References

Published:

[1] Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, STOC'13, pages 391–400, New York, NY, USA, 2013. ACM.

[2] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and VijayK. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, pages 1–19, 2015.

[3] Hammurabi Mendes, Christine Tasson, and Maurice Herlihy. Distributed computability in Byzantine asynchronous systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 704–713, New York, NY, USA, 2014. ACM.

ArXiV:

Hammurabi Mendes, Maurice Herlihy. Tight Bounds for Connectivity and Set Agreement in Byzantine Synchronous Systems. arxiv.org/abs/1505.04224

References

Published:

[1] Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, STOC'13, pages 391–400, New York, NY, USA, 2013. ACM.

[2] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and VijayK. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, pages 1–19, 2015.

[3] Hammurabi Mendes, Christine Tasson, and Maurice Herlihy. Distributed computability in Byzantine asynchronous systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 704–713, New York, NY, USA, 2014. ACM.

ArXiV:

Hammurabi Mendes, Maurice Herlihy. Tight Bounds for Connectivity and Set Agreement in Byzantine Synchronous Systems. arxiv.org/abs/1505.04224

References

Published:

[1] Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, STOC'13, pages 391–400, New York, NY, USA, 2013. ACM.

[2] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and VijayK. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, pages 1–19, 2015.

[3] Hammurabi Mendes, Christine Tasson, and Maurice Herlihy. Distributed computability in Byzantine asynchronous systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 704–713, New York, NY, USA, 2014. ACM.

ArXiV:

Hammurabi Mendes, Maurice Herlihy. Tight Bounds for Connectivity and Set Agreement in Byzantine Synchronous Systems. arxiv.org/abs/1505.04224

References

Published:

[1] Hammurabi Mendes and Maurice Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *Proceedings of the 45th annual ACM Symposium on Theory of Computing*, STOC'13, pages 391–400, New York, NY, USA, 2013. ACM.

[2] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and VijayK. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, pages 1–19, 2015.

[3] Hammurabi Mendes, Christine Tasson, and Maurice Herlihy. Distributed computability in Byzantine asynchronous systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 704–713, New York, NY, USA, 2014. ACM.

ArXiV:

Hammurabi Mendes, Maurice Herlihy. Tight Bounds for Connectivity and Set Agreement in Byzantine Synchronous Systems. arxiv.org/abs/1505.04224

Thank You!

hmendes@cs.rochester.edu