Computability Abstractions for Fault-tolerant Asynchronous Distributed Computing

Julien Stainer
under the supervision of
Michel Raynal

March 18th, 2015

Outline

Computing is Increasingly Distributed

In nearly all computing environments

- resources are (physically) distributed
  (multicores, clusters, grid, cloud, web, p2p...)

Computing is Increasingly Distributed

In nearly all computing environments

- resources are (physically) distributed
  (multicores, clusters, grid, cloud, web, p2p...)
- inputs are (physically) distributed
  (distant users, distant sensors, distant web services...)

Computing is Increasingly Distributed

In nearly all computing environments

- resources are (physically) distributed
  (multicores, clusters, grid, cloud, web, p2p...)
- inputs are (physically) distributed
  (distant users, distant sensors, distant web services...)
- computing speed, bandwidth, latencies, distances between nodes are heterogeneous

Computing is Increasingly Distributed

In nearly all computing environments

- resources are (physically) distributed
  (multicores, clusters, grid, cloud, web, p2p...)
- inputs are (physically) distributed
  (distant users, distant sensors, distant web services...)
- computing speed, bandwidth, latencies, distances between nodes are heterogeneous
- failures are the norm, not the exception

Computing is Increasingly Distributed

In nearly all computing environments

- resources are (physically) distributed
  (multicores, clusters, grid, cloud, web, p2p...)
- inputs are (physically) distributed
  (distant users, distant sensors, distant web services...)
- computing speed, bandwidth, latencies, distances between nodes are heterogeneous
- failures are the norm, not the exception

We need models to describe these environments.
We need to know what can be computed in such conditions.

Asynchrony vs. Synchrony

- Heterogeneous systems and unpredictable networks;

Asynchrony vs. Synchrony

- Heterogeneous systems and unpredictable networks;
- Varying speeds across time and space.

Asynchrony vs. Synchrony

- Heterogeneous systems and unpredictable networks;
- Varying speeds across time and space.
- Timeouts to execute in lock-steps?

Asynchrony vs. Synchrony

- Heterogeneous systems and unpredictable networks;
- Varying speeds across time and space.
- Timeouts to execute in lock-steps?
  - Not always possible (failures);

Asynchrony vs. Synchrony

- Heterogeneous systems and unpredictable networks;
- Varying speeds across time and space.
- Timeouts to execute in lock-steps?
  - Not always possible (failures);
  - synchronizes on the slowest.

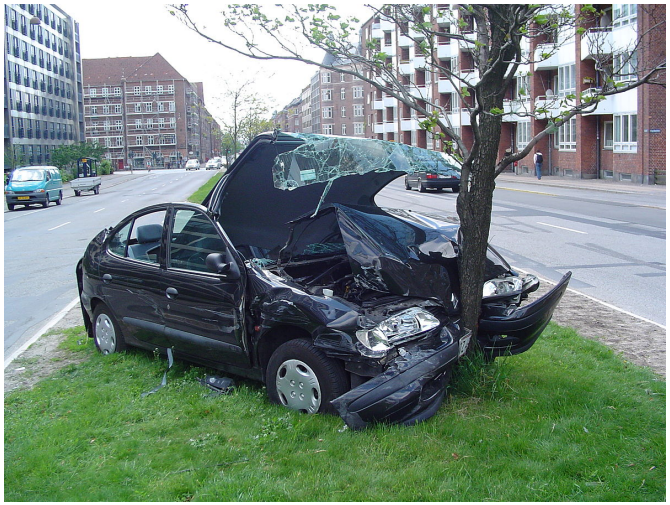
Asynchrony vs. Synchrony

- Heterogeneous systems and unpredictable networks;
- Varying speeds across time and space.
- Timeouts to execute in lock-steps?
  - Not always possible (failures);
  - synchronizes on the slowest.

We would like to have algorithms for asynchronous systems in which the relative speeds of processes are finite but unbounded.

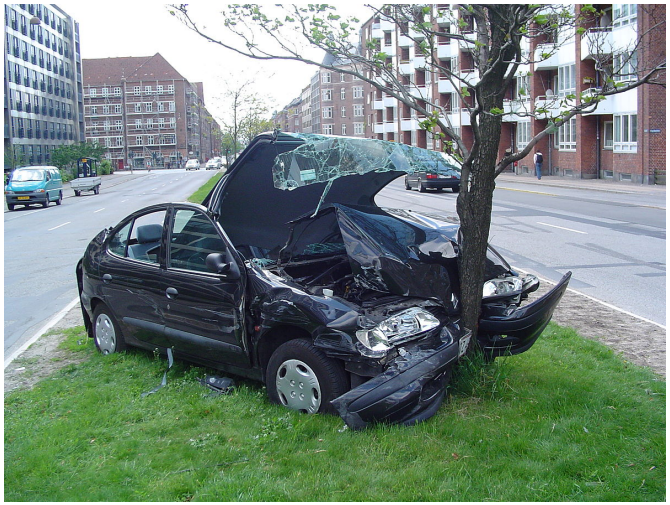
Wait-free $(n-1)$-Resilient Models

- This thesis considers wait-free solvability in $(n-1)$-resilient systems:



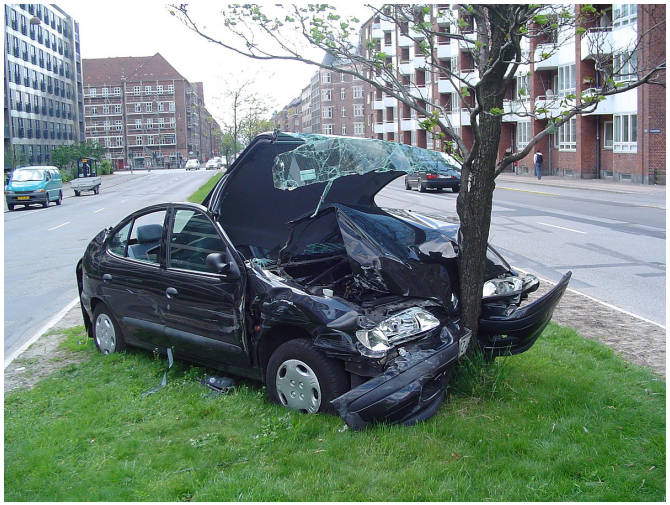
Wait-free $(n-1)$-Resilient Models

- This thesis considers wait-free solvability in $(n-1)$-resilient systems:
  - whatever the number of failures,



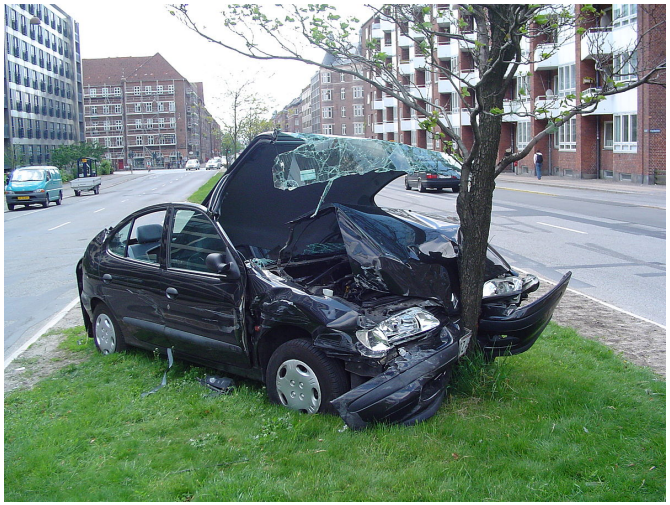
Wait-free $(n-1)$-Resilient Models

- This thesis considers wait-free solvability in $(n-1)$-resilient systems:
  - whatever the number of failures,
  - whatever the level of concurrency,



Wait-free $(n-1)$-Resilient Models

- This thesis considers wait-free solvability in $(n-1)$-resilient systems:
  - whatever the number of failures,
  - whatever the level of concurrency,
  - processes have to make progress.



Communication via Messages and Distributed Objects

Communication is needed to collaborate.

Communication via Messages and Distributed Objects

Communication is needed to collaborate.

- At the lowest level, sending and receiving messages synchronously or not.
  (processor bus, network, MPI framework...)

Communication via Messages and Distributed Objects

Communication is needed to collaborate.

- At the lowest level, sending and receiving messages synchronously or not.
  (processor bus, network, MPI framework...)
- To offer a more convenient programming abstraction, shared registers are often made available to the programmer.

Communication via Messages and Distributed Objects

Communication is needed to collaborate.

- At the lowest level, sending and receiving messages synchronously or not.
  (processor bus, network, MPI framework...)
- To offer a more convenient programming abstraction, shared registers are often made available to the programmer.
- In order to achieve modularity, more complex shared objects can encapsulate solutions to building blocks problems.
  (consensus, shared data structures...)

Communication via Messages and Distributed Objects

Communication is needed to collaborate.

- At the lowest level, sending and receiving messages synchronously or not.
  (processor bus, network, MPI framework...)
- To offer a more convenient programming abstraction, shared registers are often made available to the programmer.
- In order to achieve modularity, more complex shared objects can encapsulate solutions to building blocks problems.
  (consensus, shared data structures...)

The communication primitives available to the processes have an impact on what can be computed in asynchronous systems.

Agreement Problems

- The consensus object is a fundamental building block in distributed computing.

Agreement Problems

- The consensus object is a fundamental building block in distributed computing.
- It offers to the processes a primitive allowing each of them to propose a value and returns a unique proposed value to all of them.

Agreement Problems

- The consensus object is a fundamental building block in distributed computing.
- It offers to the processes a primitive allowing each of them to propose a value and returns a unique proposed value to all of them.
- Weaker versions of the consensus object allowing up to k values to be returned in the system have been studied.

Agreement Problems

- The consensus object is a fundamental building block in distributed computing.
- It offers to the processes a primitive allowing each of them to propose a value and returns a unique proposed value to all of them.
- Weaker versions of the consensus object allowing up to k values to be returned in the system have been studied.

Universal Construction

When consensus objects and registers are available, any shared object with a sequential specification can be implemented[a].

[a] Maurice Herlihy: Wait-Free Synchronization. ACM TOPLAS (1991)

Impossibilities and Failure Detectors

Consensus Impossibility

In the presence of failures, solving the consensus in an asynchronous system (message-passing[a] or shared memory[b] communication) is impossible.

[a] Fischer, Lynch, Paterson: Impossibility of Distributed Consensus with One Faulty Process. J. ACM (1985)

[b] Loui, Abu-Amara: Memory requirements for agreement among unreliable asynchronous processes. Advances in Computing Research (1987)

Implementing a Shared Memory

In an asynchronous message-passing system, if half of the processes can crash, it is impossible to implement a shared memory.

Impossibilities and Failure Detectors

- To work around this kind of impossibilities, the notion of failure detector[1] has been introduced.

[1] Tushar Deepak Chandra, Sam Toueg: Unreliable Failure Detectors for Reliable Distributed Systems. J. ACM (1996)

Impossibilities and Failure Detectors

- To work around this kind of impossibilities, the notion of failure detector[1] has been introduced.
- A failure detector provides system-controlled read-only variables giving some information to the processes on the failures in the current execution.

[1] Tushar Deepak Chandra, Sam Toueg: Unreliable Failure Detectors for Reliable Distributed Systems. J. ACM (1996)

Impossibilities and Failure Detectors

- To work around this kind of impossibilities, the notion of failure detector[1] has been introduced.
- A failure detector provides system-controlled read-only variables giving some information to the processes on the failures in the current execution.
- Failure detectors can be compared on the possibility to simulate one with another.

[1] Tushar Deepak Chandra, Sam Toueg: Unreliable Failure Detectors for Reliable Distributed Systems. J. ACM (1996)

Impossibilities and Failure Detectors

- To work around this kind of impossibilities, the notion of failure detector[1] has been introduced.
- A failure detector provides system-controlled read-only variables giving some information to the processes on the failures in the current execution.
- Failure detectors can be compared on the possibility to simulate one with another.
- Any problem solvable with a failure detector has an associated weakest failure detector[2].

[1] Tushar Deepak Chandra, Sam Toueg: Unreliable Failure Detectors for Reliable Distributed Systems. J. ACM (1996)

[2] Prasad Jayanti, Sam Toueg: Every problem has a weakest failure detector. PODC 2008

The Weakest Failure Detector for k-Set Agreement in Wait-Free Message-Passing Systems

The starting point of this thesis was the quest for the weakest failure detector for the k-set agreement in asynchronous message-passing systems.

It is known:

- in asynchronous shared memory[3,4],

[3] Piotr Zielinski: Anti-Omega: the weakest failure detector for set agreement. Distributed Computing (2010)

[4] E. Gafni, P. Kuznetsov: On set consensus numbers. Distr. Comp. (2011)

[5] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg: The Weakest Failure Detector for Solving Consensus. J. ACM (1996)

[6] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, A. Tielmann: The Weakest Failure Detector for Message Passing Set-Agreement. DISC 2008

The Weakest Failure Detector for k-Set Agreement in Wait-Free Message-Passing Systems

The starting point of this thesis was the quest for the weakest failure detector for the k-set agreement in asynchronous message-passing systems.

It is known:

- in asynchronous shared memory[3,4],
- in the case $k = 1$ [5],

[3] Piotr Zielinski: Anti-Omega: the weakest failure detector for set agreement. Distributed Computing (2010)

[4] E. Gafni, P. Kuznetsov: On set consensus numbers. Distr. Comp. (2011)

[5] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg: The Weakest Failure Detector for Solving Consensus. J. ACM (1996)

[6] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, A. Tielmann: The Weakest Failure Detector for Message Passing Set-Agreement. DISC 2008

The Weakest Failure Detector for k-Set Agreement in Wait-Free Message-Passing Systems

The starting point of this thesis was the quest for the weakest failure detector for the k-set agreement in asynchronous message-passing systems.

It is known:

- in asynchronous shared memory[3,4],
- in the case $k = 1$ [5],
- in the case $k = n - 1$ [6].

[3] Piotr Zielinski: Anti-Omega: the weakest failure detector for set agreement. Distributed Computing (2010)

[4] E. Gafni, P. Kuznetsov: On set consensus numbers. Distr. Comp. (2011)

[5] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg: The Weakest Failure Detector for Solving Consensus. J. ACM (1996)

[6] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, A. Tielmann: The Weakest Failure Detector for Message Passing Set-Agreement. DISC 2008

Underlying Issues

- In the case $k = 1$, the failure detector Σ [7] is needed to prevent partitioning.

[7] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui: Tight failure detection bounds on atomic object implementations. J. ACM (2010)

[8] François Bonnet, Michel Raynal: On the road to the weakest failure detector for k-set agreement in message-passing systems. TCS (2011)

Underlying Issues

- In the case $k = 1$, the failure detector Σ [7] is needed to prevent partitioning.
- It is known that when $1 < k < n - 1$, Σ is not needed: it can be solved without shared memory and in presence of partitioning.

[7] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui: Tight failure detection bounds on atomic object implementations. J. ACM (2010)

[8] François Bonnet, Michel Raynal: On the road to the weakest failure detector for k-set agreement in message-passing systems. TCS (2011)

Underlying Issues

- In the case $k = 1$, the failure detector Σ [7] is needed to prevent partitioning.
- It is known that when $1 < k < n - 1$, Σ is not needed: it can be solved without shared memory and in presence of partitioning.
- Σ_k [8] that prevent the system from partitioning in more than k sets across the execution has been proved necessary for any value of k.

[7] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui: Tight failure detection bounds on atomic object implementations. J. ACM (2010)

[8] François Bonnet, Michel Raynal: On the road to the weakest failure detector for k-set agreement in message-passing systems. TCS (2011)

Iterated Models to Study Asynchronous Computability

Iterated models allow us to consider more structured set of executions while preserving the asynchronous shared memory computability.

Iterated Models to Study Asynchronous Computability

Iterated models allow us to consider more structured set of executions while preserving the asynchronous shared memory computability.

- The failure-free synchronous message-passing model weakened by the message-adversary TOUR [9];

[9] Yehuda Afek, Eli Gafni: Asynchrony from Synchrony. ICDCN 2013

Iterated Models to Study Asynchronous Computability

Iterated models allow us to consider more structured set of executions while preserving the asynchronous shared memory computability.

- The failure-free synchronous message-passing model weakened by the message-adversary TOUR [9];
- The iterated immediate snapshot model[10].

[9] Yehuda Afek, Eli Gafni: Asynchrony from Synchrony. ICDCN 2013

[10] Elizabeth Borowsky, Eli Gafni: A Simple Algorithmically Reasoned Characterization of Wait-Free Computations (Extended Abstract). PODC 1997

Leading Questions During this Thesis

- How do computability degrades when allowing a limited but dynamic partitioning?

Leading Questions During this Thesis

- How do computability degrades when allowing a limited but dynamic partitioning?
- What is needed to solve agreement problems in presence of partitioning?

Leading Questions During this Thesis

- How do computability degrades when allowing a limited but dynamic partitioning?
- What is needed to solve agreement problems in presence of partitioning?
- How to express, in the iterated models, the computability brought by failure detectors in asynchronous systems?

Leading Questions During this Thesis

- How do computability degrades when allowing a limited but dynamic partitioning?
- What is needed to solve agreement problems in presence of partitioning?
- How to express, in the iterated models, the computability brought by failure detectors in asynchronous systems?
- What can we build between message-passing and shared memory communication?

How can we compare and link some of the numerous models?

Articles Published During this Thesis

SSS'11 🔴 🟢 🔵
Relations Linking Failure Detectors Associated with k-Set Agreement in Message-Passing Systems

OPODIS'12
When and How Process Groups Can Be Used to Reduce the Renaming Space

Euro-Par'12 🟢 🔵
From a Store-Collect Object and Ω to Efficient Asynchronous Consensus

NCA'12 🔴 🟢 🔵
Chasing the Weakest Failure Detector for k-Set Agreement in Message-Passing Systems

SIROCCO'12 🟢 🟣
Increasing the Power of the Iterated Immediate Snapshot Model with Failure Detectors

CISIS'12 🟢 🔵
A Simple Asynchronous Shared Memory Consensus Algorithm Based on Omega and Closing Sets

SIROCCO'13 🔴 🟢 🔵 🟠
Simultaneous Consensus vs Set Agreement: A Message-Passing-Sensitive Hierarchy of Agreement Problems

PODC'13 🟢 🟣
Synchrony Weakened by Message Adversaries vs Asynchrony Restricted by Failure Detectors

TCS 512 (2013)
Trust-aware peer sampling: Performance and privacy tradeoffs

AINA'14
A Simple Broadcast Algorithm for Recurrent Dynamic Systems

LATIN'14 🔴 🔵 🟣 🟠
Computing in the Presence of Concurrent Solo Executions

SIROCCO'14 🟠
Reliable Shared Memory Abstraction on Top of Asynchronous Byzantine Message-Passing Systems

OPODIS'14 🔴 🔵
Distributed Universality

🔵 Agreement Problems
🟣 Iterated Models
🔴 Partitioning Systems
🟢 Failure Detectors
🟠 Shared Memory vs. Message-Passing

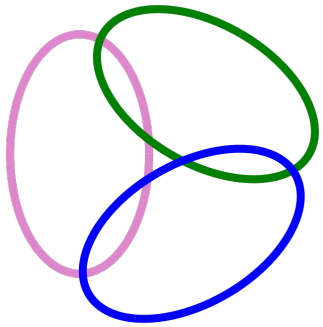
Michel Raynal, Julien Stainer:

Synchrony Weakened by Message Adversaries
vs.
Asynchrony Restricted by Failure Detectors.

PODC 2013: 166-175

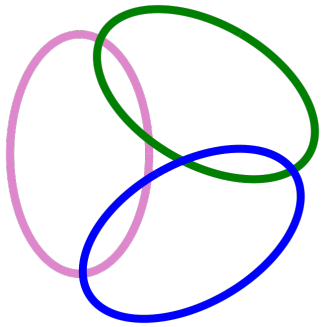
Preventing Partitioning: Σ

- Σ provides each process with a set of process identities called quorum;



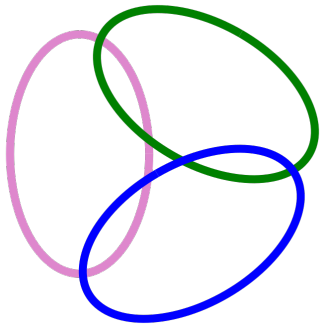
Preventing Partitioning: Σ

- Σ provides each process with a set of process identities called quorum;
- any two quorums taken at any time on any processes intersect;



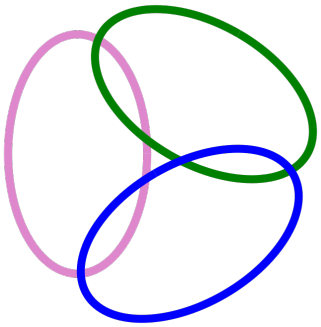
Preventing Partitioning: Σ

- Σ provides each process with a set of process identities called quorum;
- any two quorums taken at any time on any processes intersect;
- eventually the quorums only contain correct processes.



Preventing Partitioning: Σ

- Σ provides each process with a set of process identities called quorum;
- any two quorums taken at any time on any processes intersect;
- eventually the quorums only contain correct processes.



Σ is the weakest failure detector to simulate a memory in the asynchronous message-passing system $\mathcal{AMP}[\emptyset]$ [a]

[a] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui: Tight failure detection bounds on atomic object implementations. J. ACM (2010)

Breaking Symmetry: Ω

- Ω provides each process with the identity of a process considered as the leader;

Breaking Symmetry: Ω

- Ω provides each process with the identity of a process considered as the leader;
- the leader is eventually:
  - the same for each process;
  - correct.

Breaking Symmetry: Ω

- Ω provides each process with the identity of a process considered as the leader;
- the leader is eventually:
  - the same for each process;
  - correct.

Ω is the weakest failure detector to solve the consensus in the asynchronous shared memory system $\mathcal{ASM}[\emptyset]$ [a]

$\langle \Sigma, \Omega \rangle$ is the weakest failure detector to solve the consensus in the asynchronous message-passing system $\mathcal{AMP}[\emptyset]$ [b]

---

[a] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg: The Weakest Failure Detector for Solving Consensus. J. ACM (1996)

[b] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, Sam Toueg: The weakest failure detectors to solve certain fundamental problems in distributed computing. PODC 2004

The Synchronous Message-passing Model: $\mathcal{SMP}[\emptyset]$

- The execution is stripped in a sequence of rounds;

The Synchronous Message-passing Model: $\mathcal{SMP}[\emptyset]$

- The execution is stripped in a sequence of rounds;
- each round is made of three phases:
  - processes send messages to each other,

The Synchronous Message-passing Model: $\mathcal{SMP}[\emptyset]$

- The execution is stripped in a sequence of rounds;
- each round is made of three phases:
  - processes send messages to each other,
  - they receive the round messages addressed to them,

The Synchronous Message-passing Model: $\mathcal{SMP}[\emptyset]$

- The execution is stripped in a sequence of rounds;
- each round is made of three phases:
  - processes send messages to each other,
  - they receive the round messages addressed to them,
  - they compute locally their new states.

The Synchronous Message-passing Model: $\mathcal{SMP}[\emptyset]$

- The execution is stripped in a sequence of rounds;
- each round is made of three phases:
  - processes send messages to each other,
  - they receive the round messages addressed to them,
  - they compute locally their new states.
- There are no process failures.

Message Adversaries

- The adversary removes messages in $\mathcal{SMP}[\emptyset]$.

Message Adversaries

- The adversary removes messages in $\mathcal{SMP}[\emptyset]$.
- Properties define the patterns of messages that can be removed
  - during a round;
  - across the execution.

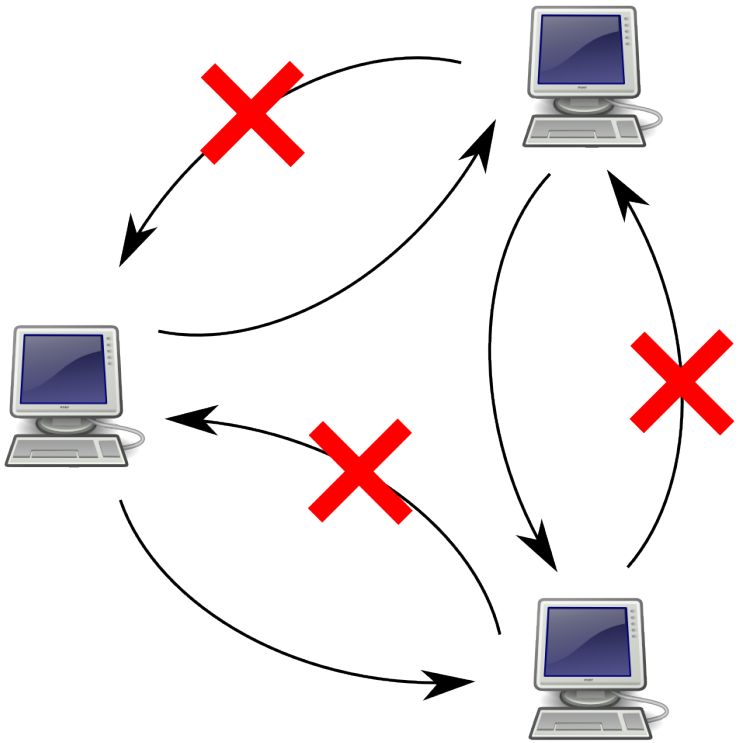
Message Adversaries

- The adversary removes messages in $\mathcal{SMP}[\emptyset]$.
- Properties define the patterns of messages that can be removed
  - during a round;
  - across the execution.

Adversaries weaken the synchronous crash-free model $\mathcal{SMP}[\emptyset]$.

Shared Memory from Synchrony: the Adversary TOUR

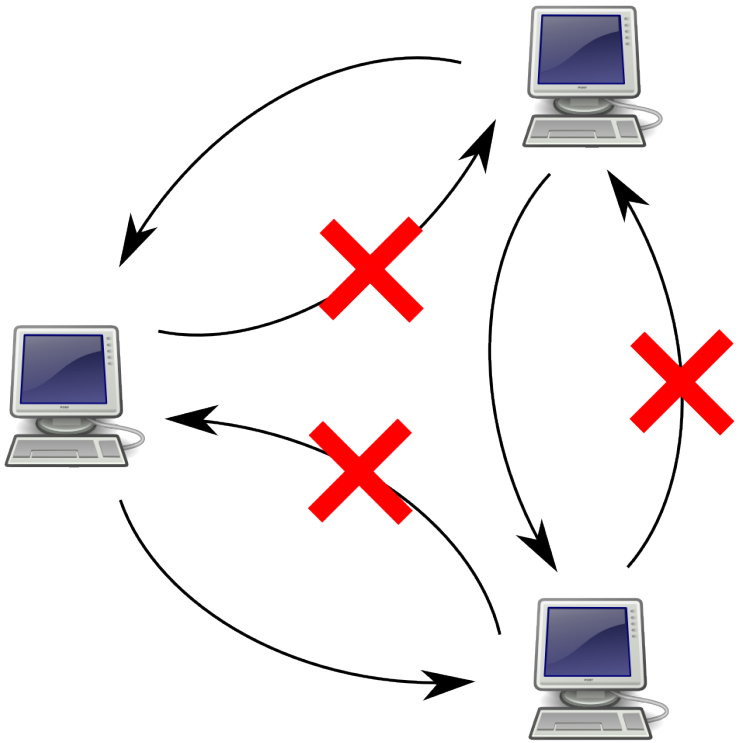
- TOUR [11] can remove any message but it preserves a tournament in any round.
- In any round and between any pair of processes, it has to let one of the two messages exchanged untouched.



[11] Yehuda Afek, Eli Gafni: Asynchrony from Synchrony. ICDCN 2013

Shared Memory from Synchrony: the Adversary TOUR

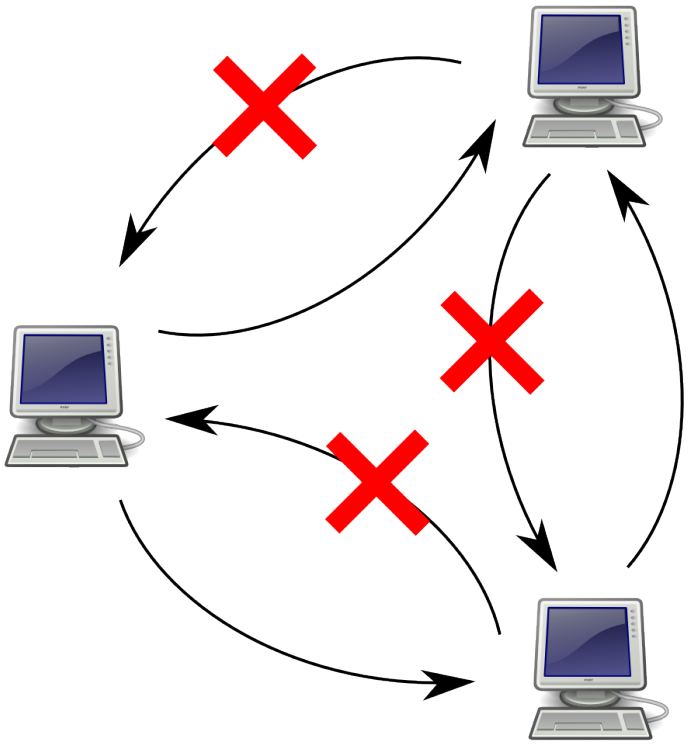
- TOUR [11] can remove any message but it preserves a tournament in any round.
- In any round and between any pair of processes, it has to let one of the two messages exchanged untouched.



[11] Yehuda Afek, Eli Gafni: Asynchrony from Synchrony. ICDCN 2013

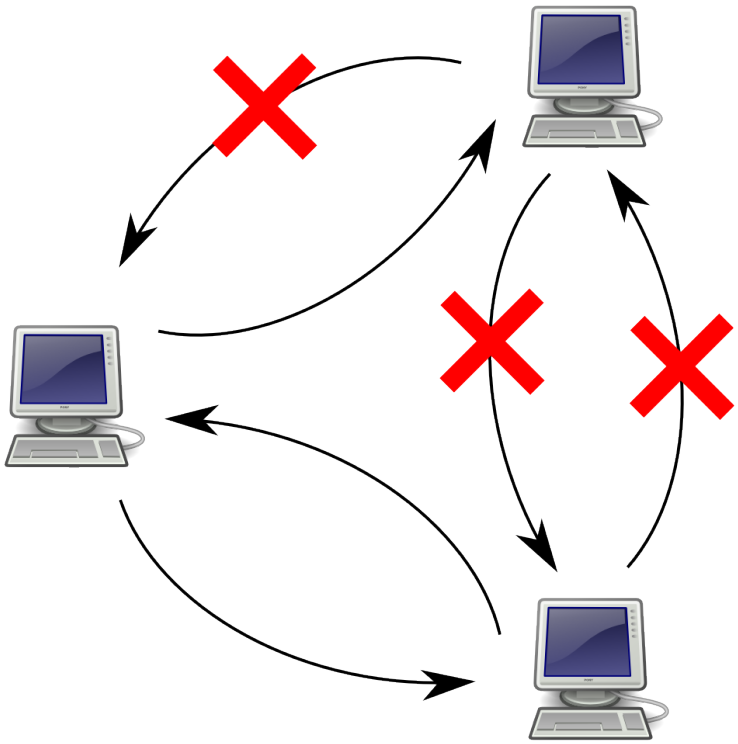
Ω from Synchrony: the Adversary SOURCE

- SOURCE can remove any message but it eventually preserves all messages sent by a given source



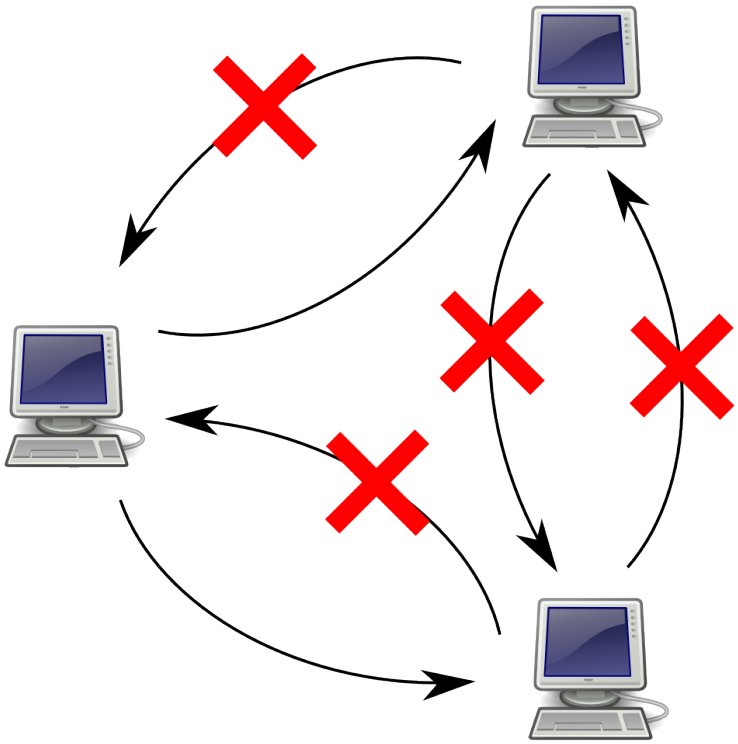
Ω from Synchrony: the Adversary SOURCE

- SOURCE can remove any message but it eventually preserves all messages sent by a given source



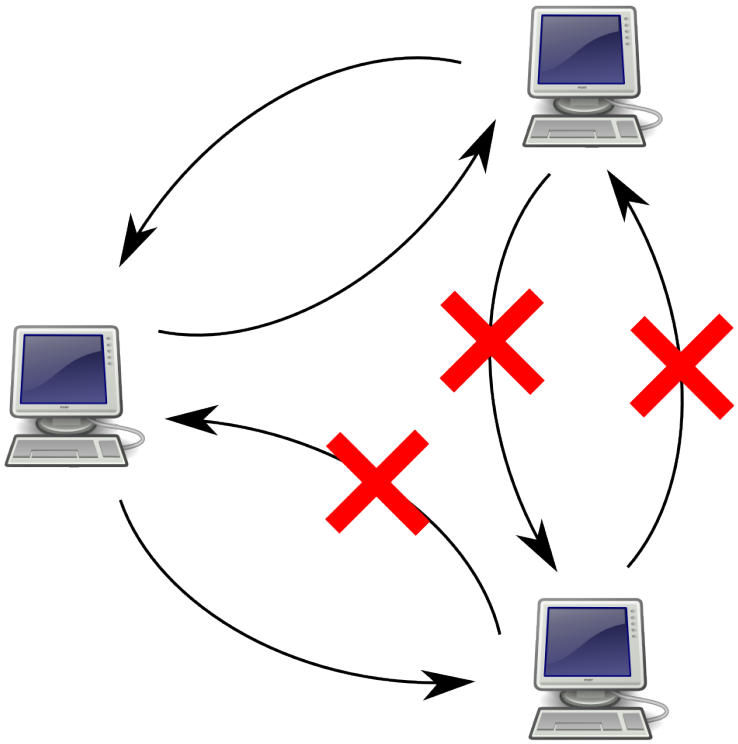
Ω from Synchrony: the Adversary SOURCE

- SOURCE can remove any message but it eventually preserves all messages sent by a given source



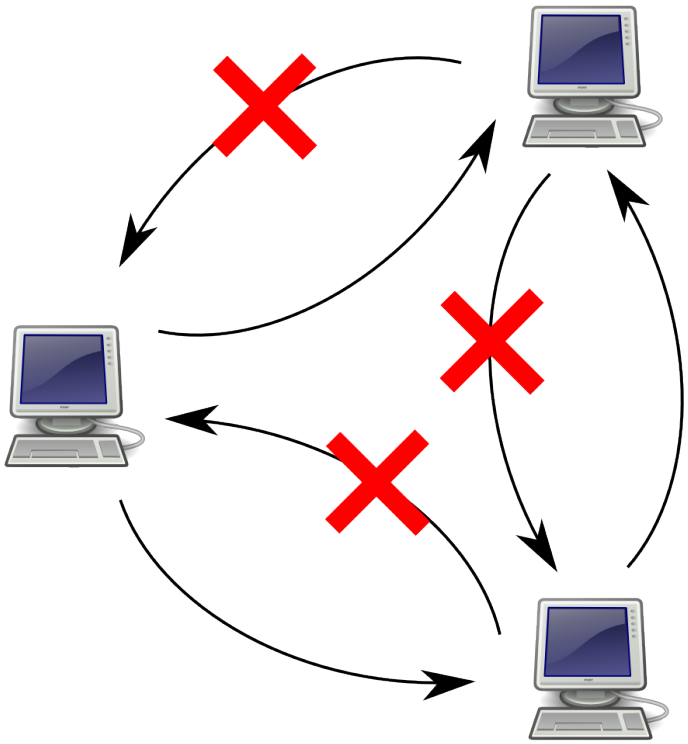
Ω from Synchrony: the Adversary SOURCE

- SOURCE can remove any message but it eventually preserves all messages sent by a given source



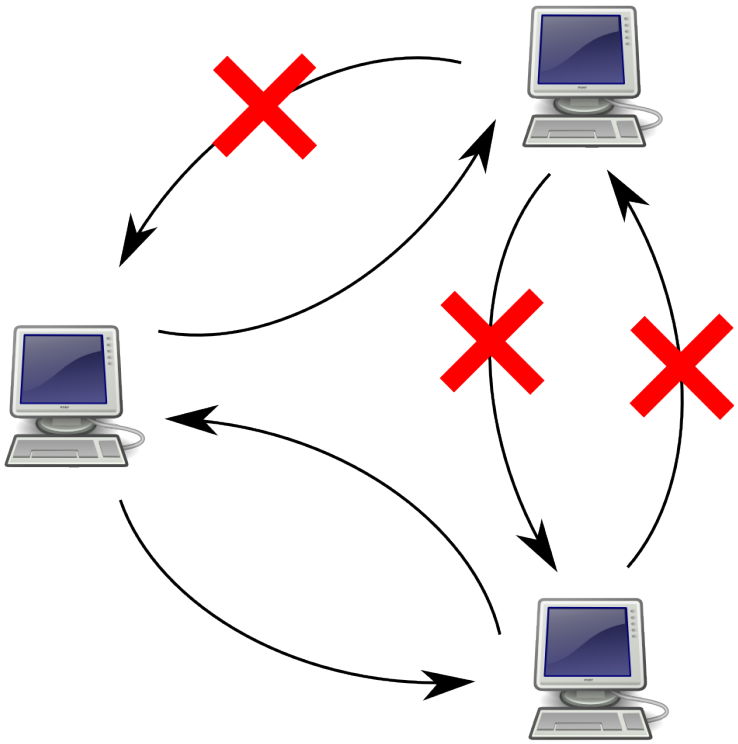
Ω from Synchrony: the Adversary SOURCE

- SOURCE can remove any message but it eventually preserves all messages sent by a given source



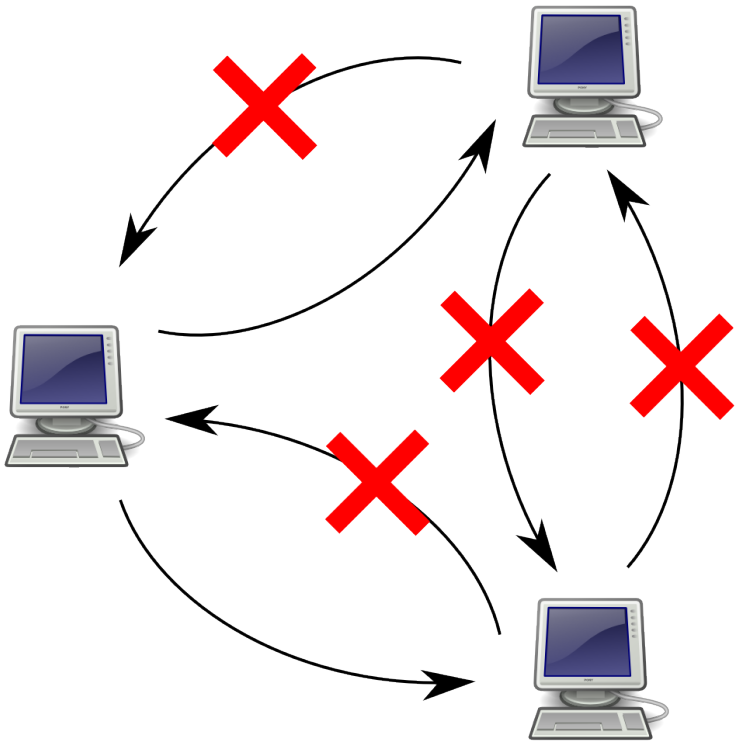
Ω from Synchrony: the Adversary SOURCE

- SOURCE can remove any message but it eventually preserves all messages sent by a given source



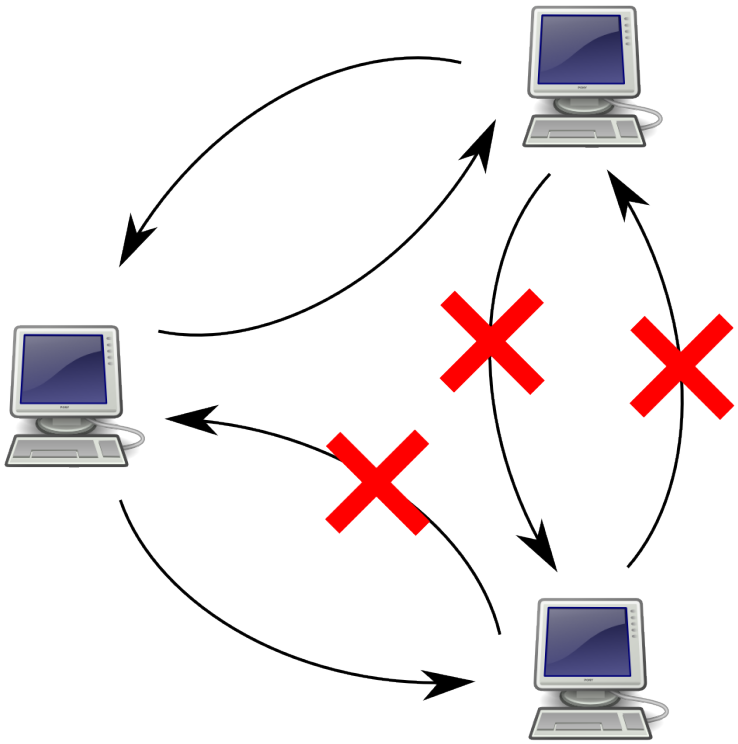
Ω from Synchrony: the Adversary SOURCE

- SOURCE can remove any message but it eventually preserves all messages sent by a given source



Ω from Synchrony: the Adversary SOURCE

- SOURCE can remove any message but it eventually preserves all messages sent by a given source



Σ from Synchrony: the Adversary QUORUM

- QUORUM can remove any message but in each round each process receives messages from an entire quorum:
- in any two rounds r_1 and r_2, for any two processes p_1 and p_2, there is a process p_3 such that:
  - p_1 receives the message of p_3 during r_1 and
  - p_2 receives the message of p_3 during r_2;
- There is at least one process that is infinitely often able to send messages (directly or not) to any other process.

Equivalence Results

$$\mathcal{SMP}[adv:\emptyset] \simeq \mathcal{AMP}[nocrash] \simeq \mathcal{ASM}[nocrash]$$

$$\mathcal{SMP}[adv:SOURCE+QUORUM] \simeq \mathcal{AMP}[fd:\Sigma+\Omega]$$

$$\mathcal{SMP}[adv:SOURCE+TOUR] \simeq \mathcal{ASM}[fd:\Omega] \qquad \mathcal{SMP}[adv:QUORUM] \simeq \mathcal{AMP}[fd:\Sigma]$$

$$\mathcal{SMP}[adv:SOURCE] \simeq \mathcal{AMP}[fd:\Omega] \qquad \mathcal{SMP}[adv:TOUR] \simeq \mathcal{ASM}[\emptyset]$$

$$\mathcal{SMP}[adv:\infty] \simeq \mathcal{AMP}[\emptyset]$$

(computability w.r.t. colorless tasks)

Remarks and Perspectives

- Expressing the calculability of the two asynchronous models associated with failure detectors through message adversaries gives us a new way to compare them in a common framework.

Remarks and Perspectives

- Expressing the calculability of the two asynchronous models associated with failure detectors through message adversaries gives us a new way to compare them in a common framework.
- Strongly correct processes play an essential role in dynamic systems.

Remarks and Perspectives

- Expressing the calculability of the two asynchronous models associated with failure detectors through message adversaries gives us a new way to compare them in a common framework.
- Strongly correct processes play an essential role in dynamic systems.

- What are the message adversaries that allow agreement tasks to be solved?

Remarks and Perspectives

- Expressing the calculability of the two asynchronous models associated with failure detectors through message adversaries gives us a new way to compare them in a common framework.
- Strongly correct processes play an essential role in dynamic systems.

- What are the message adversaries that allow agreement tasks to be solved?
- Is there a matching message adversary for any failure detector?

Remarks and Perspectives

- Expressing the calculability of the two asynchronous models associated with failure detectors through message adversaries gives us a new way to compare them in a common framework.
- Strongly correct processes play an essential role in dynamic systems.

- What are the message adversaries that allow agreement tasks to be solved?
- Is there a matching message adversary for any failure detector?
- What happens when considering colored tasks?

Maurice Herlihy, Sergio Rajsbaum, Michel Raynal, Julien Stainer:

Computing in the Presence of Concurrent Solo Executions.

LATIN 2014: 214-225

Wait-free Algorithms and Solo Executions

- Wait-free models in both meanings:
  - as a progress condition: each process makes progress in a finite number of steps, whatever the level of concurrence;

Wait-free Algorithms and Solo Executions

- Wait-free models in both meanings:
  - as a progress condition: each process makes progress in a finite number of steps, whatever the level of concurrence;
  - as a resiliency condition: the computation has to be valid even if all processes but one crash.

Wait-free Algorithms and Solo Executions

- Wait-free models in both meanings:
  - as a progress condition: each process makes progress in a finite number of steps, whatever the level of concurrence;
  - as a resiliency condition: the computation has to be valid even if all processes but one crash.

- Slow and crashed processes are indistinguishable:
  some processes
  - may have to behave as if they were alone;
  - do not have access to other processes inputs.

Shared Memory, Message passing and Solo Executions

- If processes share a memory, then at most one of them can be in that situation for a given execution.

Shared Memory, Message passing and Solo Executions

- If processes share a memory, then at most one of them can be in that situation for a given execution.
- If processes exchange asynchronous messages, then all of them may have to behave as if they were alone.

Shared Memory, Message passing and Solo Executions

- If processes share a memory, then at most one of them can be in that situation for a given execution.
- If processes exchange asynchronous messages, then all of them may have to behave as if they were alone.

What could be computed in intermediate models in which up to d processes may run solo?

d-Solo Models

- An iterated model generalizing the iterated immediate snapshot model.

d-Solo Models

- An iterated model generalizing the iterated immediate snapshot model.
- The execution is stripped in a sequence of rounds;
- a one-shot communication object for each round;

d-Solo Models

- An iterated model generalizing the iterated immediate snapshot model.
- The execution is stripped in a sequence of rounds;
- a one-shot communication object for each round;
- each process writes a value and retrieves the previously or simultaneously written values.

d-Solo Models

- An iterated model generalizing the iterated immediate snapshot model.
- The execution is stripped in a sequence of rounds;
- a one-shot communication object for each round;
- each process writes a value and retrieves the previously or simultaneously written values.
- in the d-solo model, the first set of simultaneous accesses can miss each other.
  - If they do, then this set contains at most d processes.

A spectrum of models that spans from message-passing ($d = n$) to shared memory ($d = 1$).

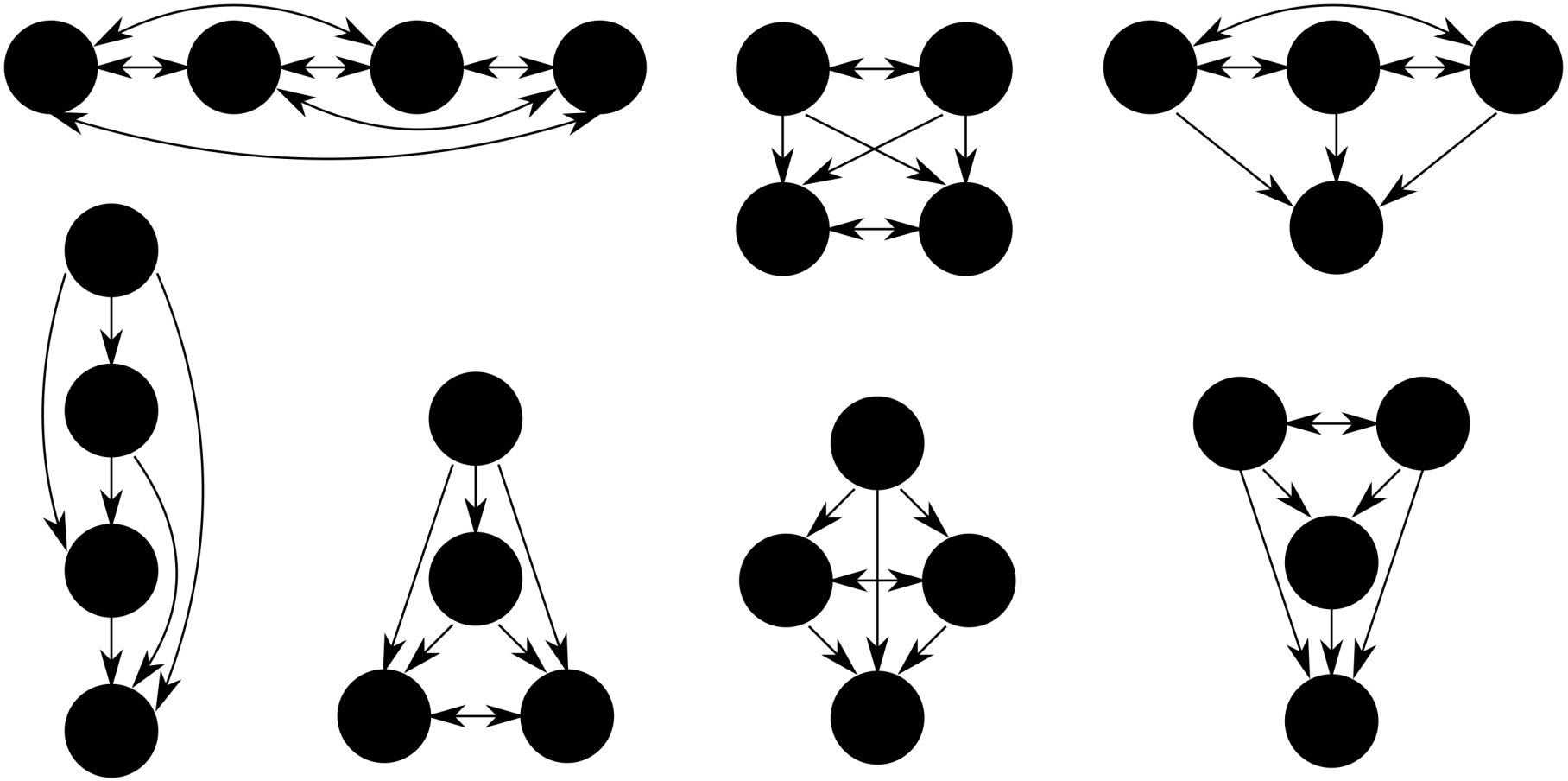
From the Immediate Snapshot Object...

- Each process p provides a value v_p to the object and retrieves a set of values (a view).

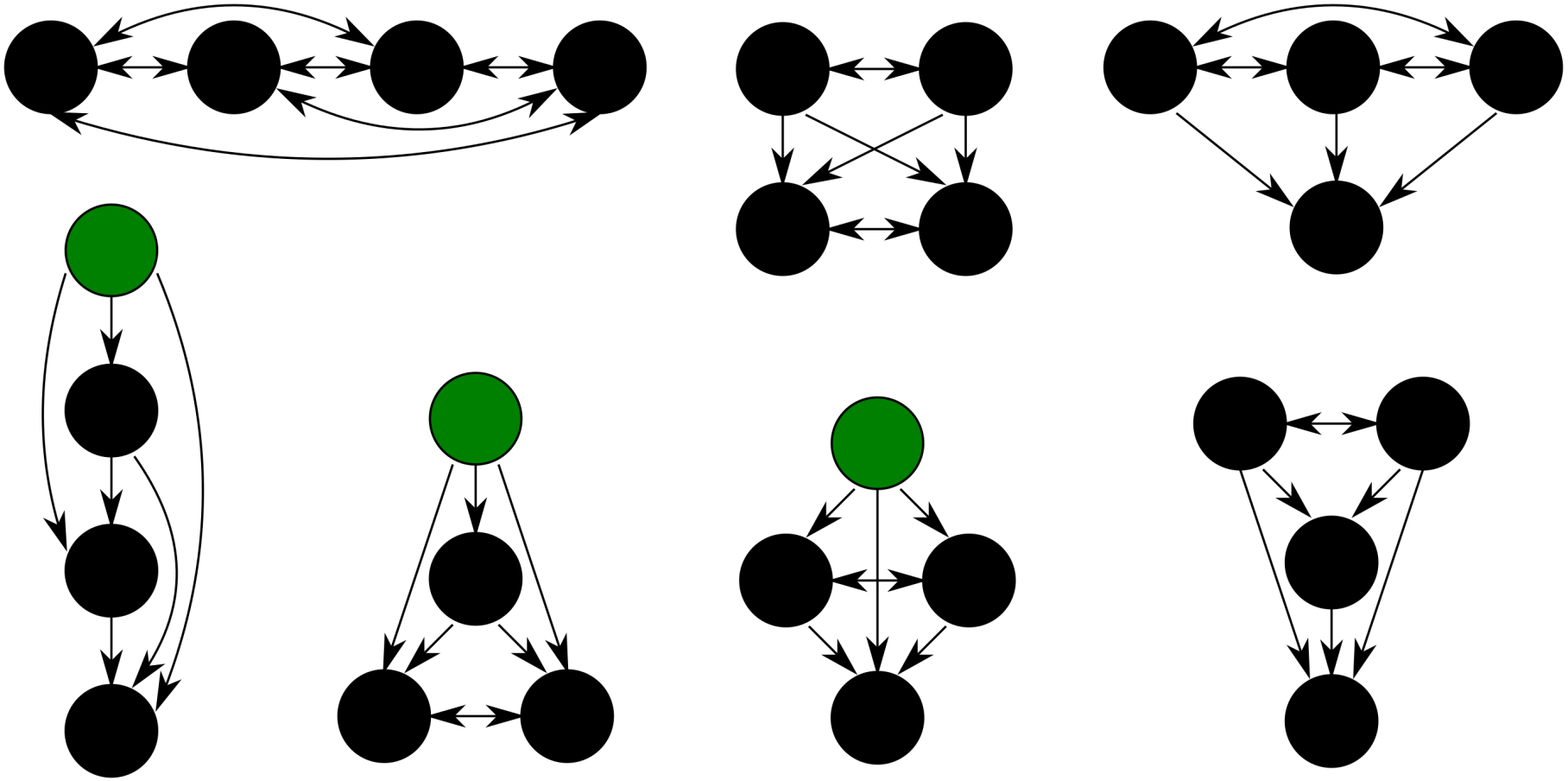
From the Immediate Snapshot Object...

- Each process p provides a value v_p to the object and retrieves a set of values (a view).
- As with the immediate snapshot object, any ordered partition $(\pi_1, \ldots, \pi_x)$ of the set of the processes accessing the object describe a valid behavior for the object:
  - the view of any process belonging to π_i is $\bigcup_{j \leq i} \{(p, v_p), p \in \pi_j\}$.

From the Immediate Snapshot Object...



From the Immediate Snapshot Object...



...to the CO^d Communication Object

- Additionally, any ordered partition $(\rho_1, \ldots, \rho_x)$ of the set of processes accessing the object describe another authorized behavior for the object if $|\rho_1| \leq d$:

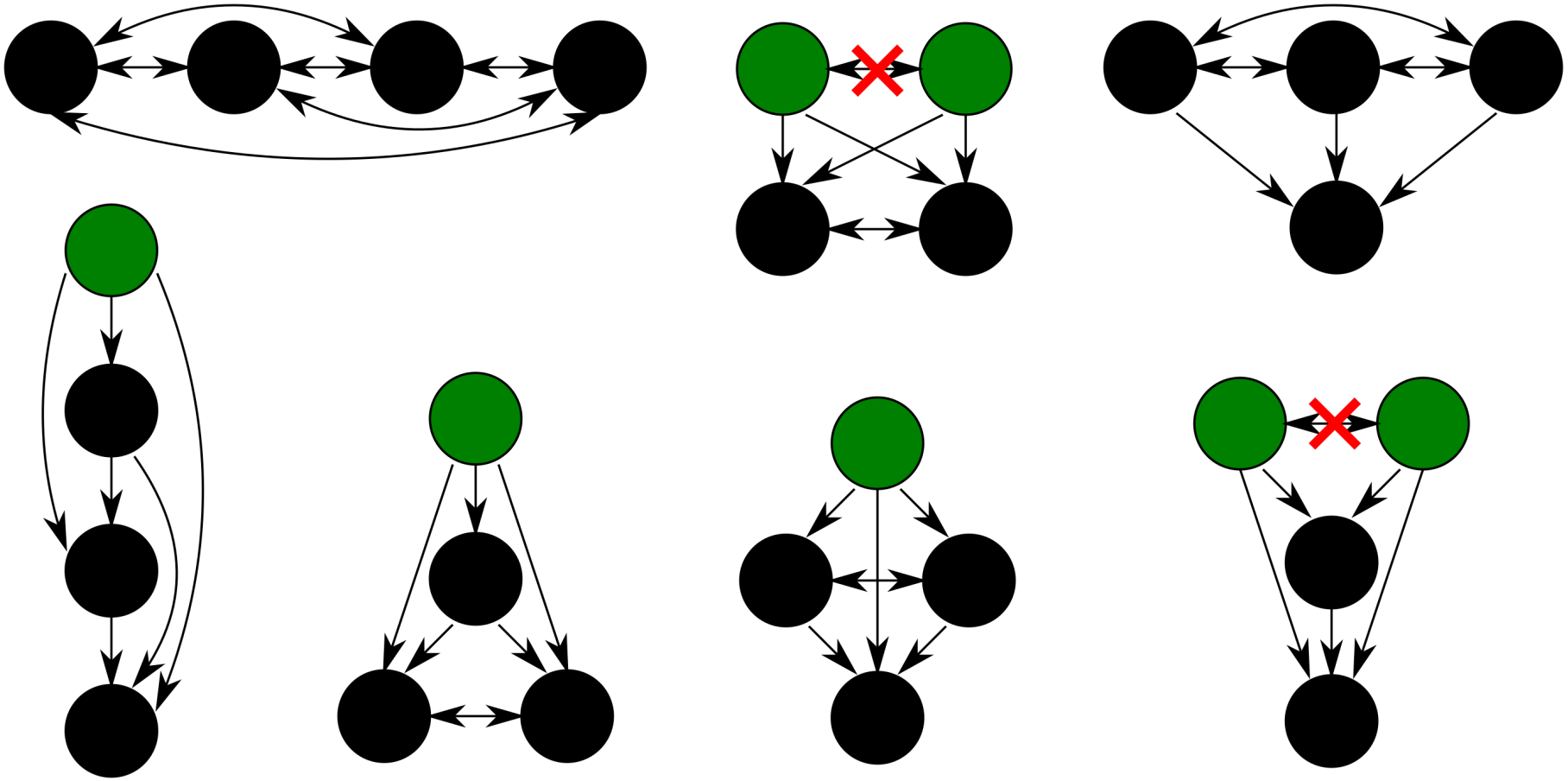
...to the CO^d Communication Object

- Additionally, any ordered partition $(\rho_1, \ldots, \rho_x)$ of the set of processes accessing the object describe another authorized behavior for the object if $|\rho_1| \leq d$:
  - if $i > 1$, then the view of any process belonging to ρ_i is $\bigcup_{j \leq i} \{(p, v_p), p \in \rho_j\}$;

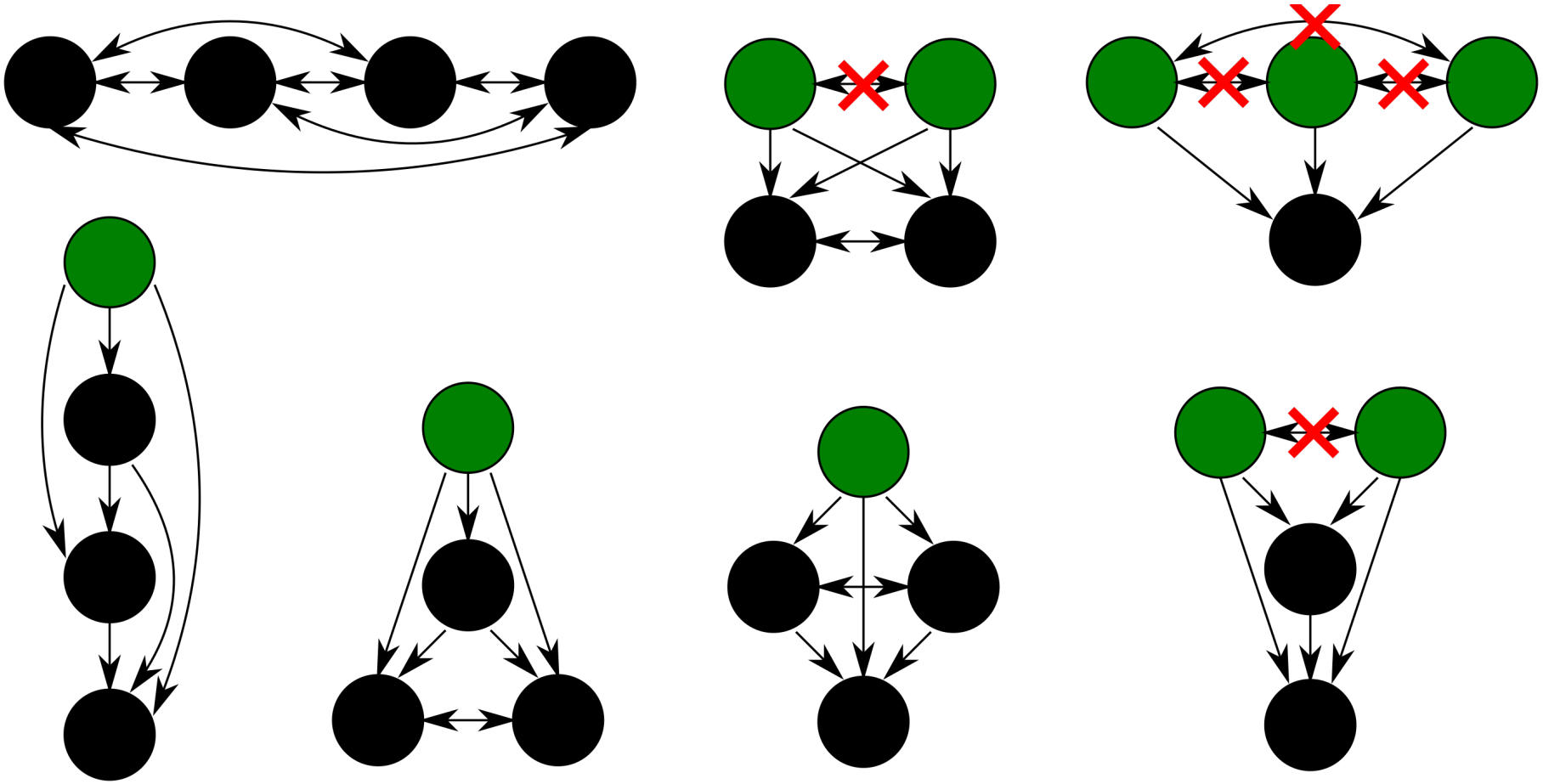
...to the CO^d Communication Object

- Additionally, any ordered partition $(\rho_1, \ldots, \rho_x)$ of the set of processes accessing the object describe another authorized behavior for the object if $|\rho_1| \leq d$:
  - if $i > 1$, then the view of any process belonging to ρ_i is $\bigcup_{j \leq i} \{(p, v_p), p \in \rho_j\}$;
  - the view of a process p of ρ_1 is $\{v_p\}$.

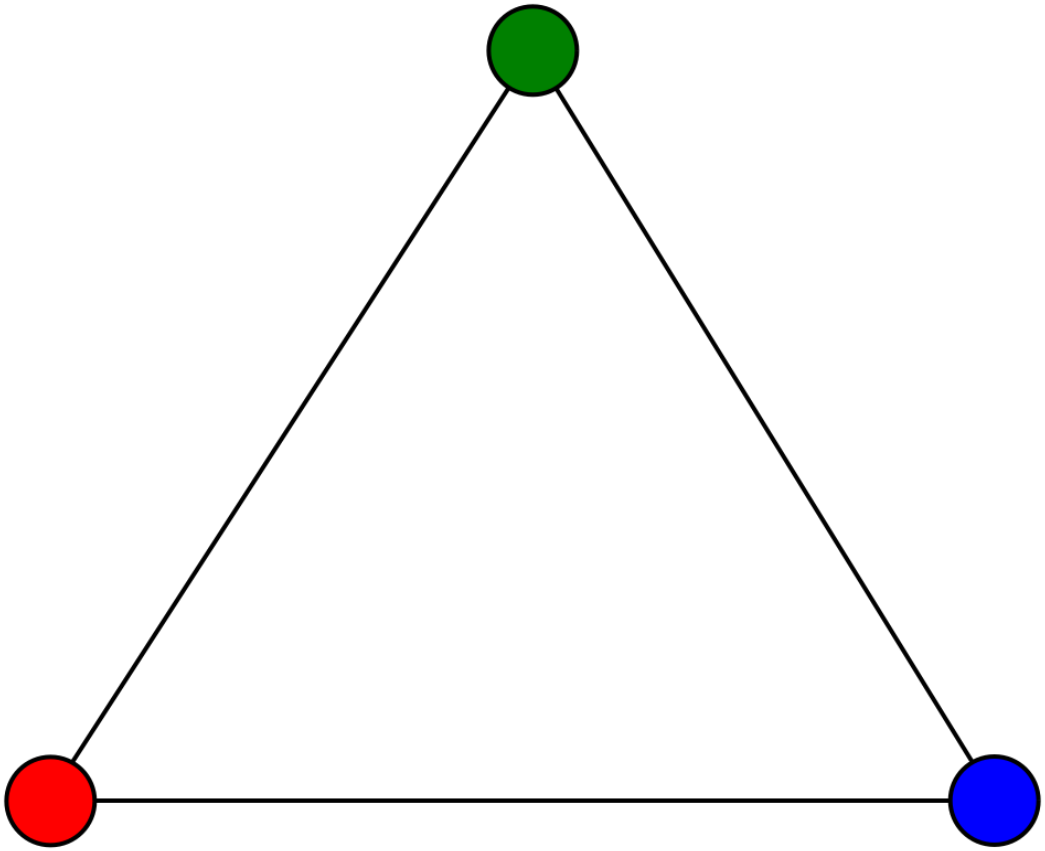
...to the CO^d Communication Object



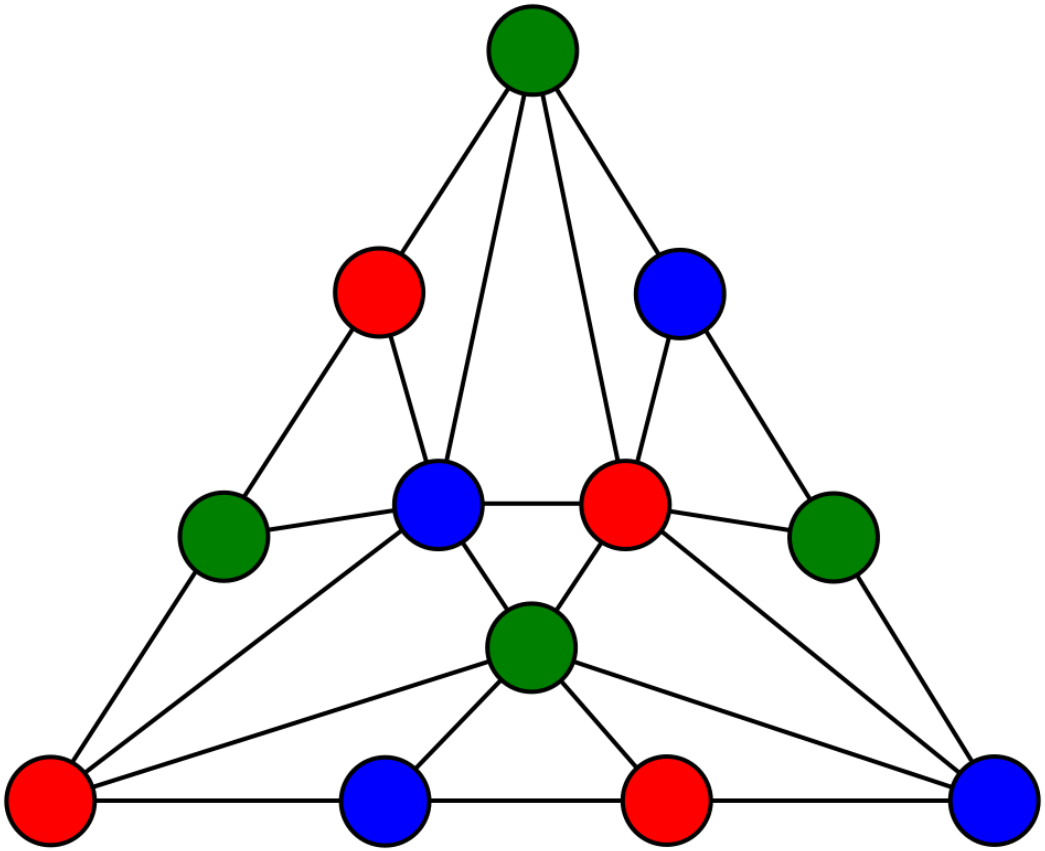
...to the CO^d Communication Object



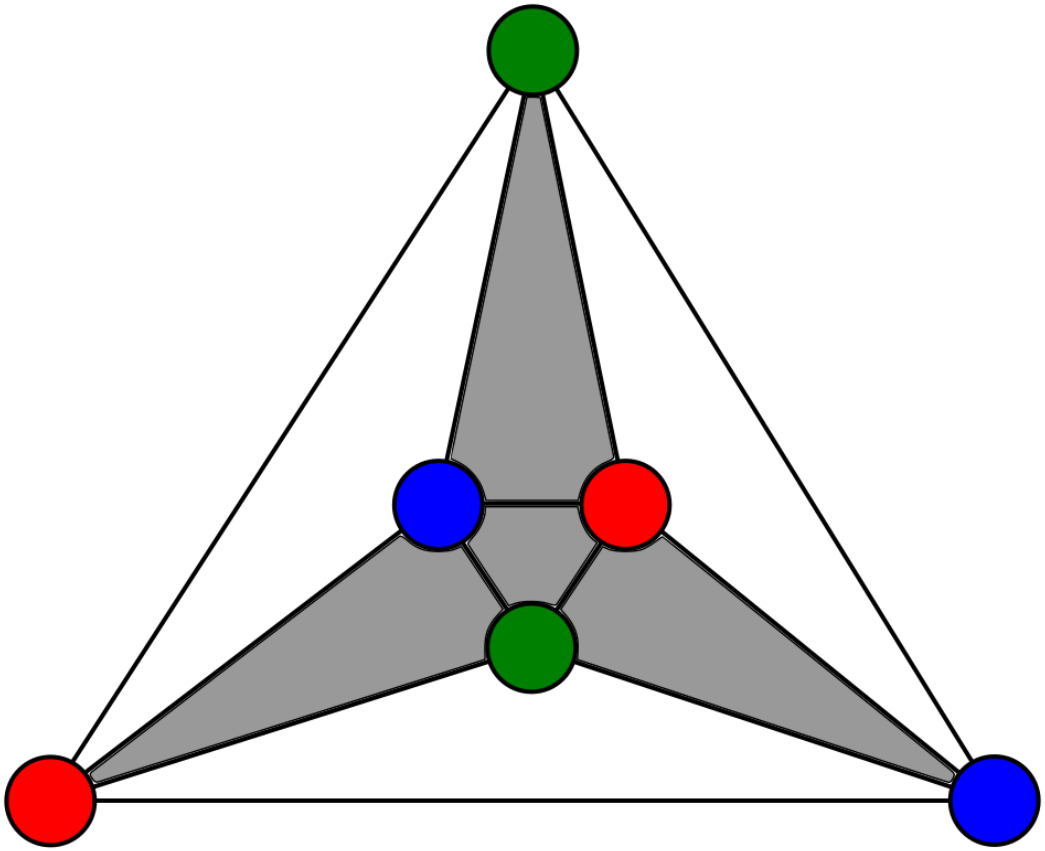
The Subdivided Complex of the Possible States after one Access to an Immediate Snapshot Object



The Subdivided Complex of the Possible States after one Access to an Immediate Snapshot Object



The Additional Possible States after one Access to a CO^2 Object



The Colorless Algorithm in the d-solo model

- We consider the case of a colorless algorithm:
  - processes do not use their identities during the computation;

The Colorless Algorithm in the d-solo model

- We consider the case of a colorless algorithm:
  - processes do not use their identities during the computation;
  - they use the object as a set: during each round a process writes the last view it retrieved (initially its input value) ignoring writers identities and multiple occurrences of the same view;

The Colorless Algorithm in the d-solo model

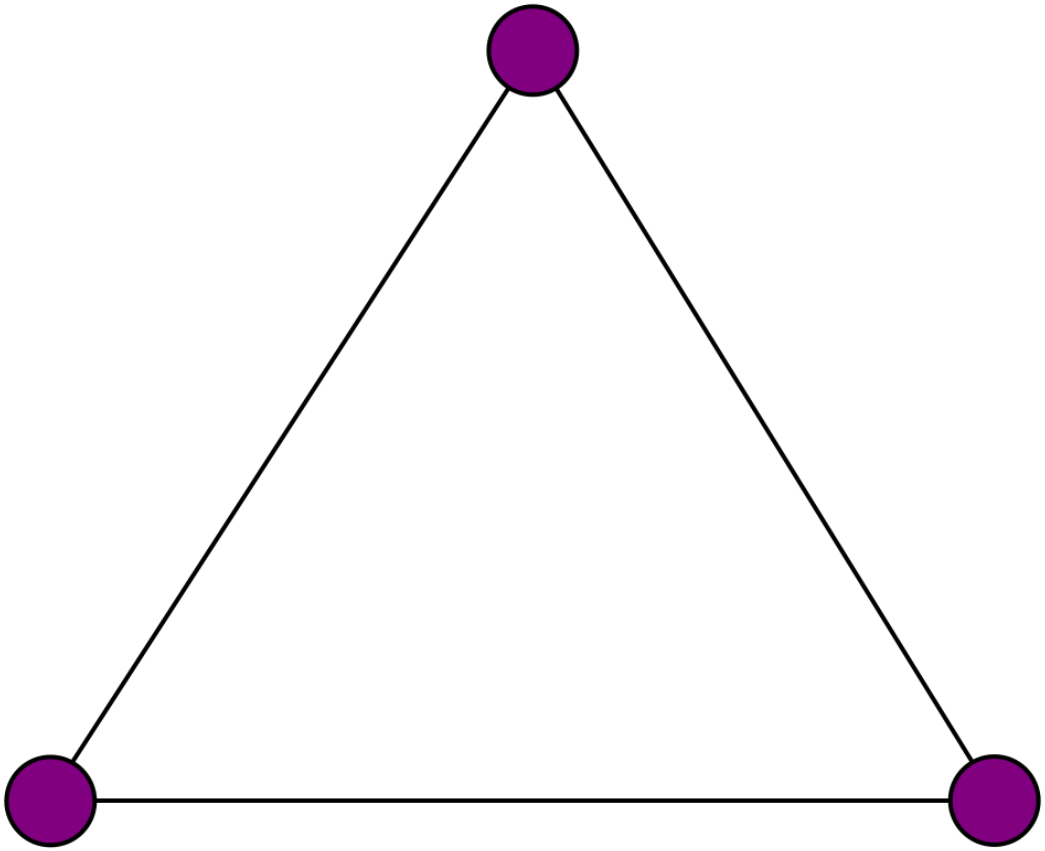
- We consider the case of a colorless algorithm:
  - processes do not use their identities during the computation;
  - they use the object as a set: during each round a process writes the last view it retrieved (initially its input value) ignoring writers identities and multiple occurrences of the same view;
  - they compute their output from their view after R rounds.

The Colorless Algorithm in the d-solo model

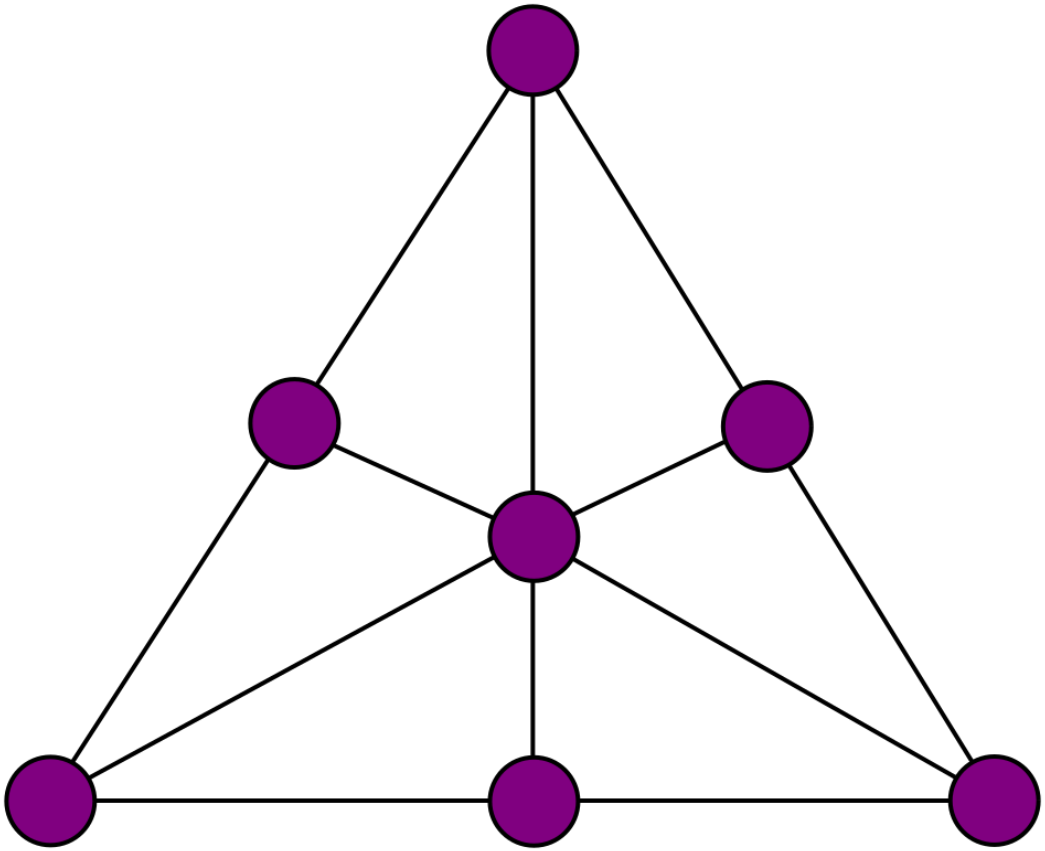
- We consider the case of a colorless algorithm:
  - processes do not use their identities during the computation;
  - they use the object as a set: during each round a process writes the last view it retrieved (initially its input value) ignoring writers identities and multiple occurrences of the same view;
  - they compute their output from their view after R rounds.

- It allows us to describe all the possible states of the system after the execution of R rounds by a subdivided complex without coloring vertices with process identities.

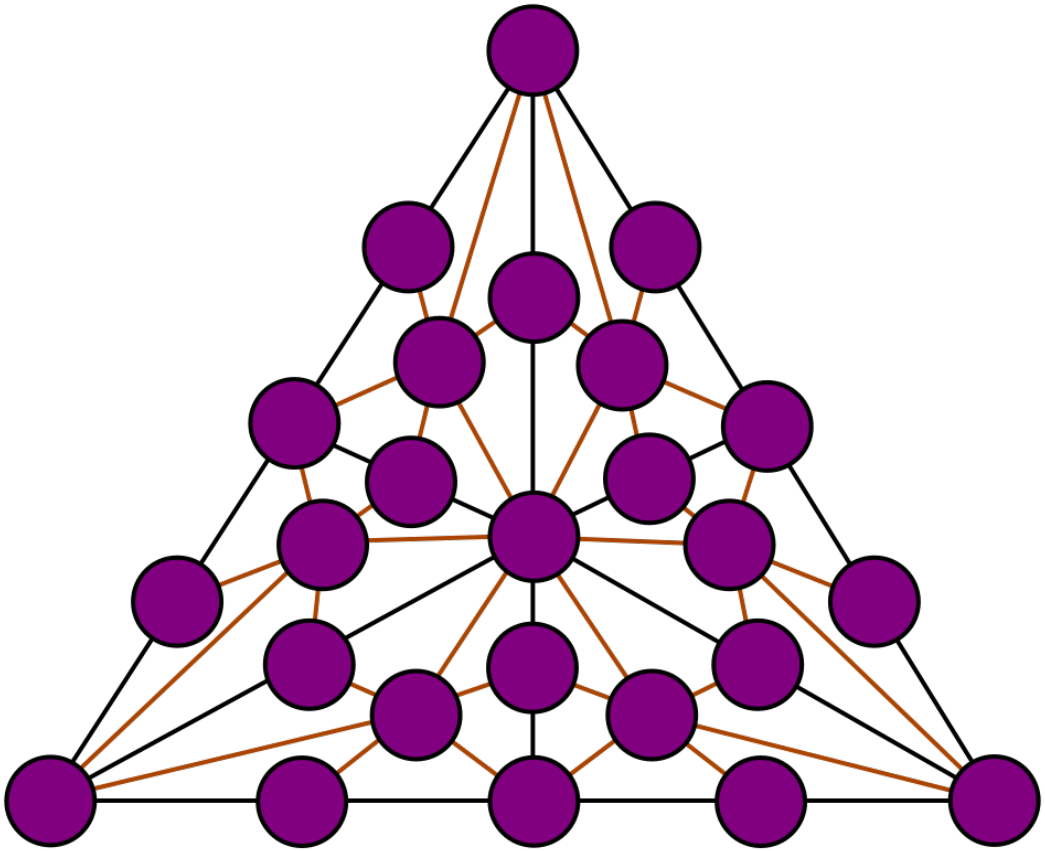
The Colorless Complex of the Possible States in the Colorless Algorithm (Immediate Snapshot)



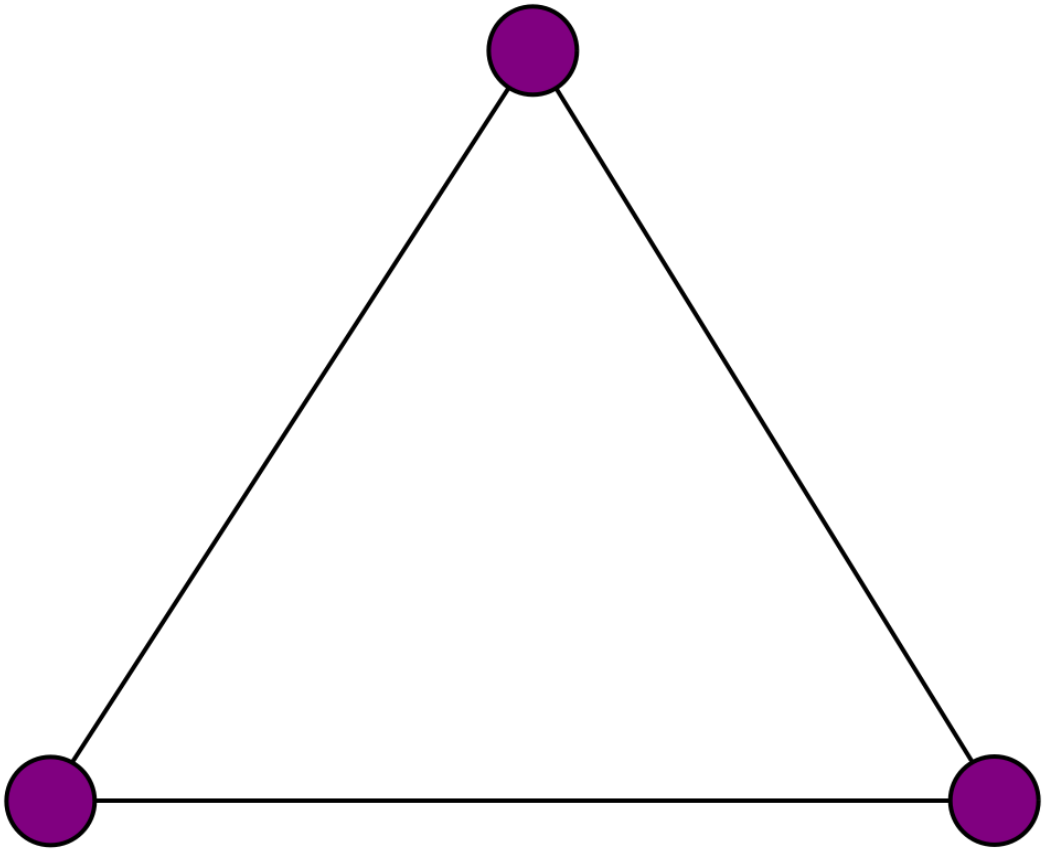
The Colorless Complex of the Possible States in the Colorless Algorithm (Immediate Snapshot)



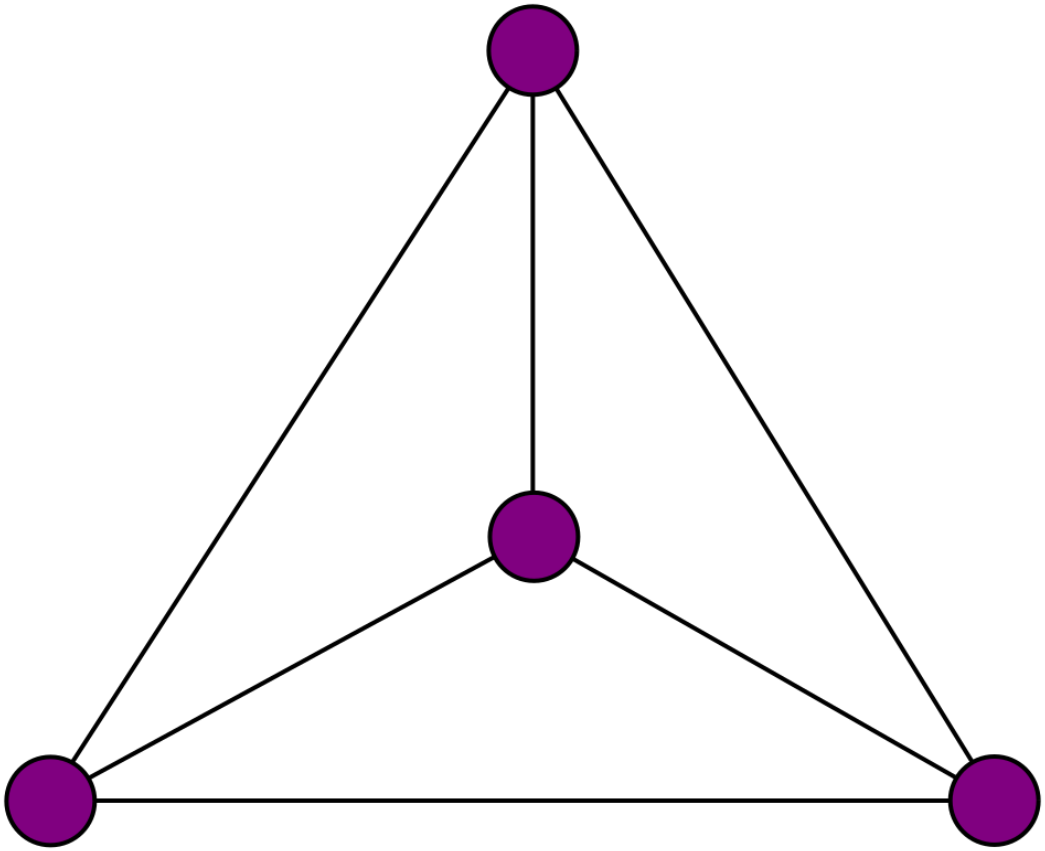
The Colorless Complex of the Possible States in the Colorless Algorithm (Immediate Snapshot)



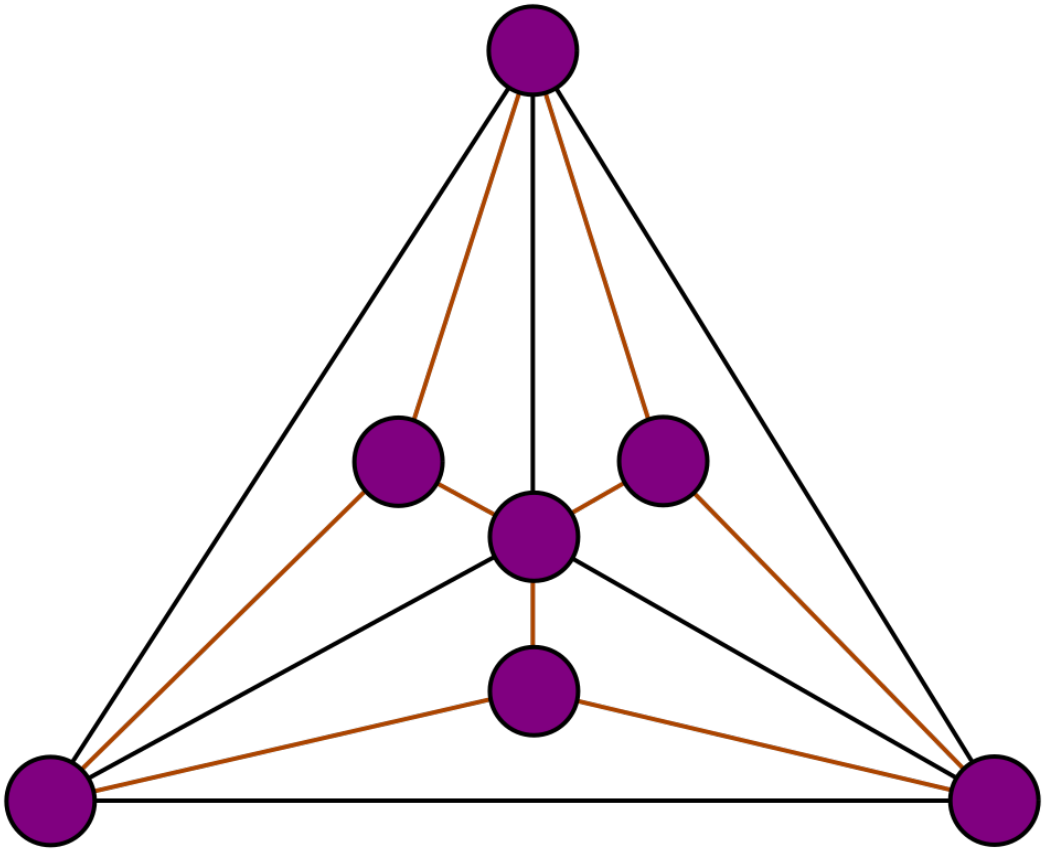
The Colorless Complex of the Additional Possible States in the Colorless Algorithm (CO^2)



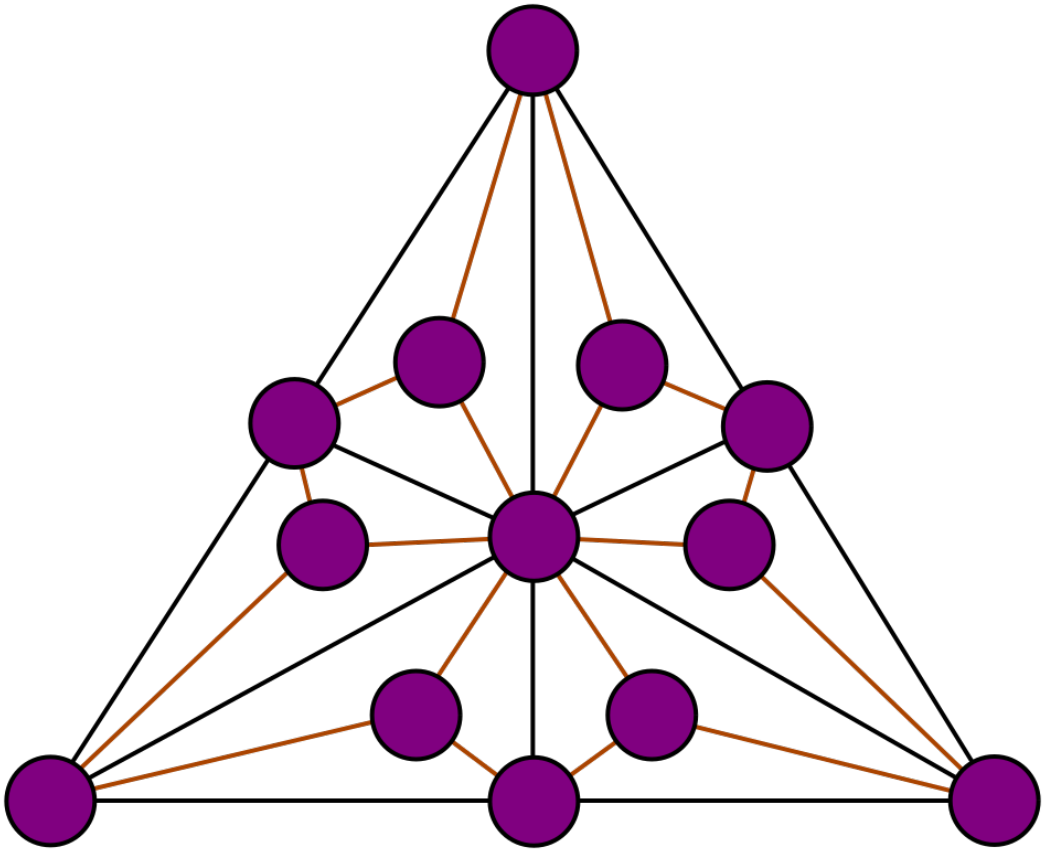
The Colorless Complex of the Additional Possible States in the Colorless Algorithm (CO^2)



The Colorless Complex of the Additional Possible States in the Colorless Algorithm (CO^2)



The Colorless Complex of the Additional Possible States in the Colorless Algorithm (CO^2)



Task Solvability in the d-Solo Model

- A Colorless Task is specified by:

Task Solvability in the d-Solo Model

- A Colorless Task is specified by:
  - the (colorless) complex of all possible input configurations;

Task Solvability in the d-Solo Model

- A Colorless Task is specified by:
  - the (colorless) complex of all possible input configurations;
  - the (colorless) complex of output configurations;

Task Solvability in the d-Solo Model

- A Colorless Task is specified by:
  - the (colorless) complex of all possible input configurations;
  - the (colorless) complex of output configurations;
  - a monotonic carrier map associating each input configuration to a set of allowed output configurations.

Task Solvability in the d-Solo Model

- A Colorless Task is specified by:
  - the (colorless) complex of all possible input configurations;
  - the (colorless) complex of output configurations;
  - a monotonic carrier map associating each input configuration to a set of allowed output configurations.

Theorem

A colorless task is solvable by a colorless algorithm in the d-solo model with n processes if and only if there is a number of rounds $R \geq 0$ and a simplicial map from the R-iterated d-subdivision of the $n-1$ skeleton of the (colorless) input complex to the (colorless) output complex that is carried by the colorless task carrier map.

The (d, ϵ)-Approximate Agreement Problem

- Generalizing the ϵ-Approximate Agreement that is universal for the shared memory model

The (d, ϵ)-Approximate Agreement Problem

- Generalizing the ϵ-Approximate Agreement that is universal for the shared memory model
- Each process proposes a value from an Euclidian space.

The (d, ϵ)-Approximate Agreement Problem

- Generalizing the ϵ-Approximate Agreement that is universal for the shared memory model

- Each process proposes a value from an Euclidian space.
- **Termination**: all correct processes decide in a finite number of steps.

The (d, ϵ)-Approximate Agreement Problem

- Generalizing the ϵ-Approximate Agreement that is universal for the shared memory model

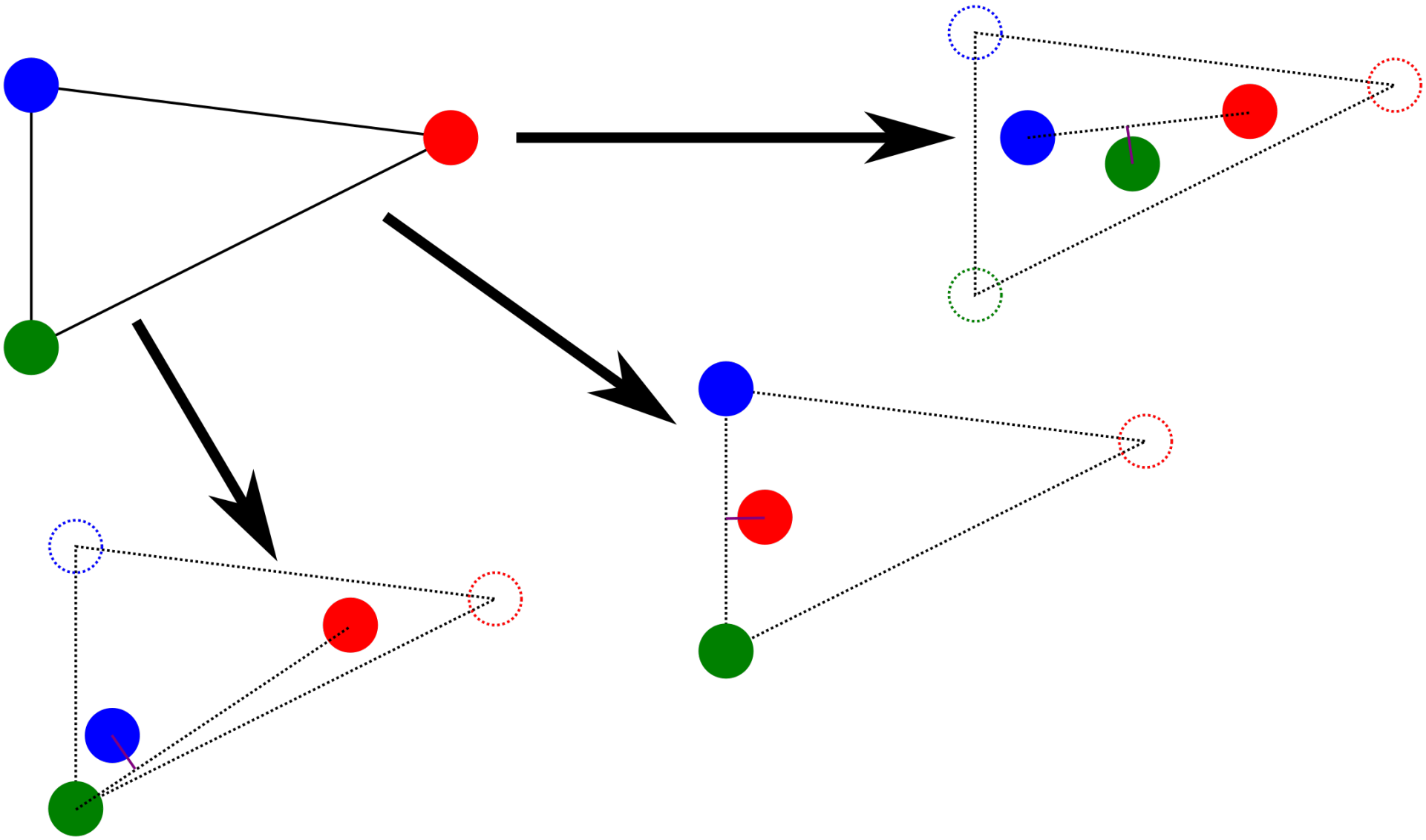
- Each process proposes a value from an Euclidian space.
- **Termination**: all correct processes decide in a finite number of steps.
- **Validity**: all the decided values belong to the convex hull of the set of proposed values.

The (d, ϵ)-Approximate Agreement Problem

- Generalizing the ϵ-Approximate Agreement that is universal for the shared memory model

- Each process proposes a value from an Euclidian space.
- **Termination**: all correct processes decide in a finite number of steps.
- **Validity**: all the decided values belong to the convex hull of the set of proposed values.
- **Agreement**: there is a set S of up to d processes that can decide any valid value while other processes have to decide within a distance of ϵ from the convex hull of the values decided by processes of S.

The (d, ϵ)-Approximate Agreement Problem



(d, ϵ)-Approximate Agreement in the d-Solo Models

- For any ϵ, any d and any n, if the volume of the d-faces of the input complex is bounded, there is a number of round R such that the colorless algorithm solves the (d, ϵ)-approximate agreement problem in the d-solo model.

(d, ϵ)-Approximate Agreement in the d-Solo Models

- For any ϵ, any d and any n, if the volume of the d-faces of the input complex is bounded, there is a number of round R such that the colorless algorithm solves the (d, ϵ)-approximate agreement problem in the d-solo model.

- For any ϵ, any d and any $n, n > d$, if there is a simplex of the input complex containing a large enough regular d-simplex, then the (d, ϵ)-approximate agreement problem is impossible to solve in the $(d + 1)$-solo model.

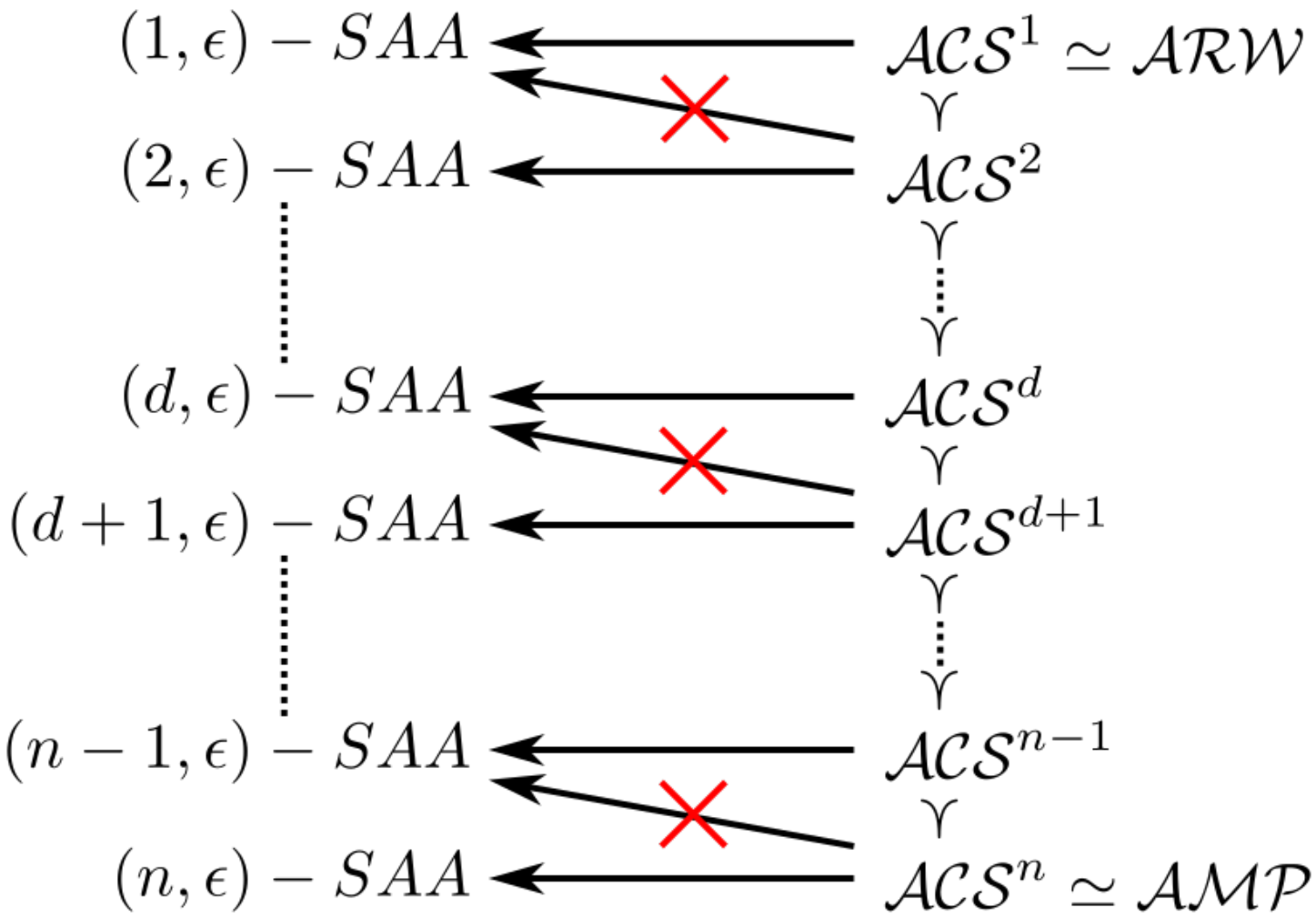
(d, ϵ)-Approximate Agreement in the d-Solo Models

- For any ϵ, any d and any n, if the volume of the d-faces of the input complex is bounded, there is a number of round R such that the colorless algorithm solves the (d, ϵ)-approximate agreement problem in the d-solo model.

- For any ϵ, any d and any $n, n > d$, if there is a simplex of the input complex containing a large enough regular d-simplex, then the (d, ϵ)-approximate agreement problem is impossible to solve in the $(d + 1)$-solo model.

Since these conditions are compatible, the hierarchy of the d-solo models is strict.

A Strict Hierarchy Spanning from Shared Memory to Message-Passing



Status and Further Investigation

- We built a hierarchy of iterated models, spanning from shared memory to message-passing.

Status and Further Investigation

- We built a hierarchy of iterated models, spanning from shared memory to message-passing.
- Calculability decrease strictly along this hierarchy, when the allowed number of parallel solo executions increases.

Status and Further Investigation

- We built a hierarchy of iterated models, spanning from shared memory to message-passing.
- Calculability decrease strictly along this hierarchy, when the allowed number of parallel solo executions increases.
- At the different levels of the hierarchy, there is a matching generalization of ϵ-agreement.

Status and Further Investigation

- We built a hierarchy of iterated models, spanning from shared memory to message-passing.
- Calculability decrease strictly along this hierarchy, when the allowed number of parallel solo executions increases.
- At the different levels of the hierarchy, there is a matching generalization of ϵ-agreement.

- Can we solve a stronger generalization of ϵ-agreement if we do not restrict to colorless algorithms?

Status and Further Investigation

- We built a hierarchy of iterated models, spanning from shared memory to message-passing.
- Calculability decrease strictly along this hierarchy, when the allowed number of parallel solo executions increases.
- At the different levels of the hierarchy, there is a matching generalization of ϵ-agreement.

- Can we solve a stronger generalization of ϵ-agreement if we do not restrict to colorless algorithms?
- How does computability evolve if we allow more behavior for the communication object?

Summary

During this thesis, we explored

- the way asynchronous models enriched with failure detector can relate to iterated models;

Summary

During this thesis, we explored

- the way asynchronous models enriched with failure detector can relate to iterated models;
- the relations between shared memory and message-passing and the possibility of intermediary models;

Summary

During this thesis, we explored

- the way asynchronous models enriched with failure detector can relate to iterated models;
- the relations between shared memory and message-passing and the possibility of intermediary models;
- the impact of potential partitioning on agreement tasks.

Summary

During this thesis, we explored

- the way asynchronous models enriched with failure detector can relate to iterated models;
- the relations between shared memory and message-passing and the possibility of intermediary models;
- the impact of potential partitioning on agreement tasks.

The processes that infinitely often able to communicate, directly or not, with all the others have a special role across the different models.

Summary

During this thesis, we explored

- the way asynchronous models enriched with failure detector can relate to iterated models;
- the relations between shared memory and message-passing and the possibility of intermediary models;
- the impact of potential partitioning on agreement tasks.

The processes that infinitely often able to communicate, directly or not, with all the others have a special role across the different models.

Partitioning, if contained, is not an end. There are important tasks that are solvable without the ability to implement a memory.

Perspectives

In the future, I would like to further study

- the computability links between asynchronous systems and dynamic networks in presence of partitioning;

Perspectives

In the future, I would like to further study

- the computability links between asynchronous systems and dynamic networks in presence of partitioning;
- the models allowing to express correlated/heterogeneous failures;

Perspectives

In the future, I would like to further study

- the computability links between asynchronous systems and dynamic networks in presence of partitioning;
- the models allowing to express correlated/heterogeneous failures;
- the possible high level abstractions we can offer to ease programming against byzantine failures;

Perspectives

In the future, I would like to further study

- the computability links between asynchronous systems and dynamic networks in presence of partitioning;
- the models allowing to express correlated/heterogeneous failures;
- the possible high level abstractions we can offer to ease programming against byzantine failures;
- the mathematical structure of the set of possible executions in presence of partitioning.

Thank you for your attention!