Consensus

Robert Burgess

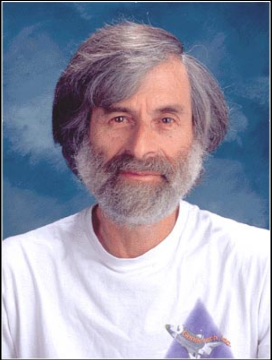
What is consensus?

- Assume a collection of processes that can propose values. A consensus algorithm ensures that a single one among the proposed values is chosen . . . We won't try to specify precise liveness requirements.
- The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value . . . every protocol for this problem has the possibility of nontermination . . .

What is consensus?

- Only a proposed value may be chosen.
- Only one, unique value may be chosen.
- All correct processes must eventually choose that value.

Paxos



Leslie Lamport

Paxos

- The Part-Time Parliament (1998)

Paxos

- The Part-Time Parliament (1998)

  Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state machine approach to the design of distributed systems.

Paxos

- The Part-Time Parliament (1998)

  Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state machine approach to the design of distributed systems.

- Paxos Made Simple (2001)

Paxos

- The Part-Time Parliament (1998)

  Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state machine approach to the design of distributed systems.

- Paxos Made Simple (2001)

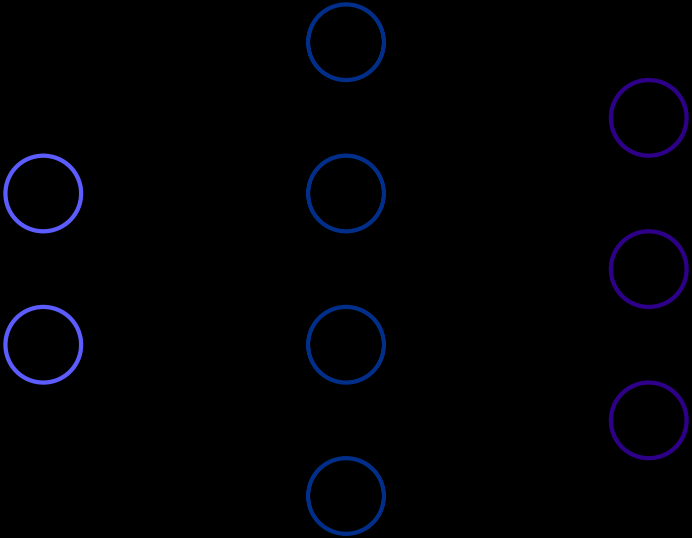
  The Paxos algorithm, when presented in plain English, is very simple.

Asynchronous network

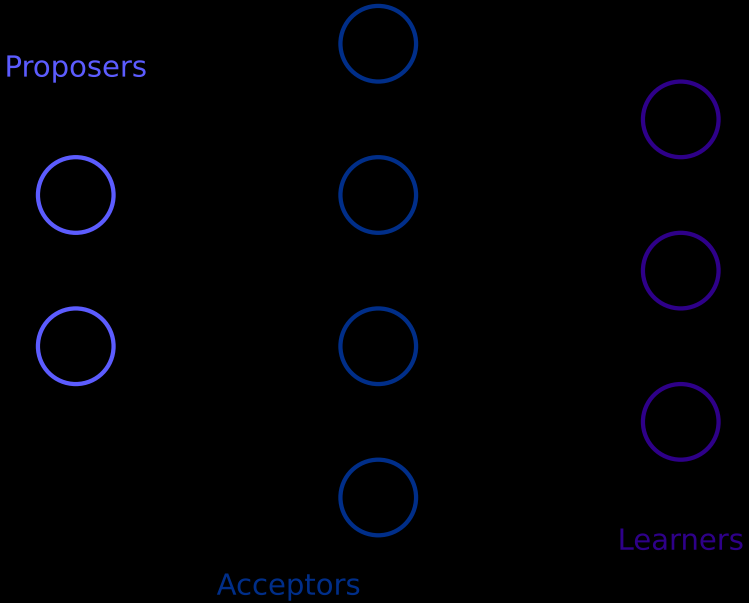
Processes can fail or restart
Messages can be

- lost
- duplicated
- reordered
- held arbitrarily long

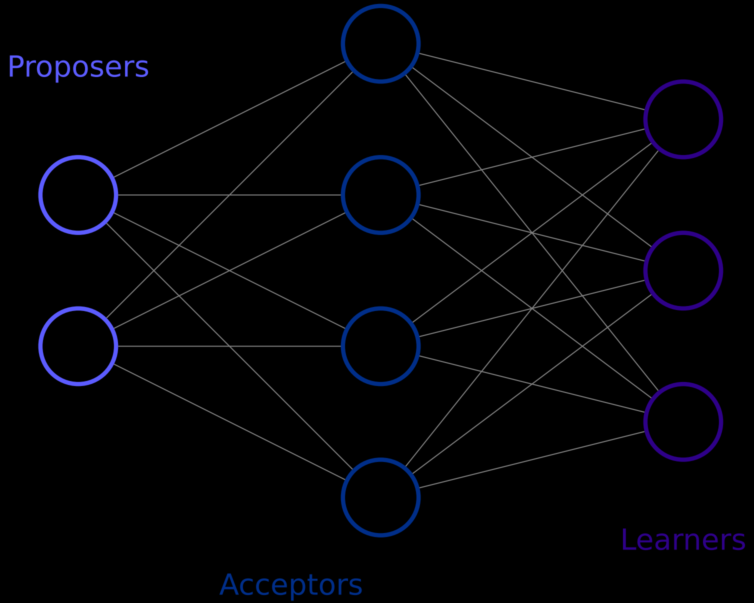
Processes



Processes



Processes



Any process might fail

There must be multiple acceptors.

Only choose a single value

A majority of acceptors must agree on the choice.

Property 1

An acceptor must accept the first proposal it receives.

Wait—what?

Majority-must-agree + Must-accept-first =
Acceptors must be able to accept multiple proposals

Wait—what?

Majority-must-agree + Must-accept-first =
Acceptors must be able to accept multiple proposals

- Number all proposals uniquely to distinguish them

Property 2

If a proposal with value v is chosen, then every higher-numbered proposal *that is chosen* has value v.

Property 2a

If a proposal with value *v* is chosen, then every higher-numbered proposal *accepted by any acceptor* has value *v*.

Property 2b

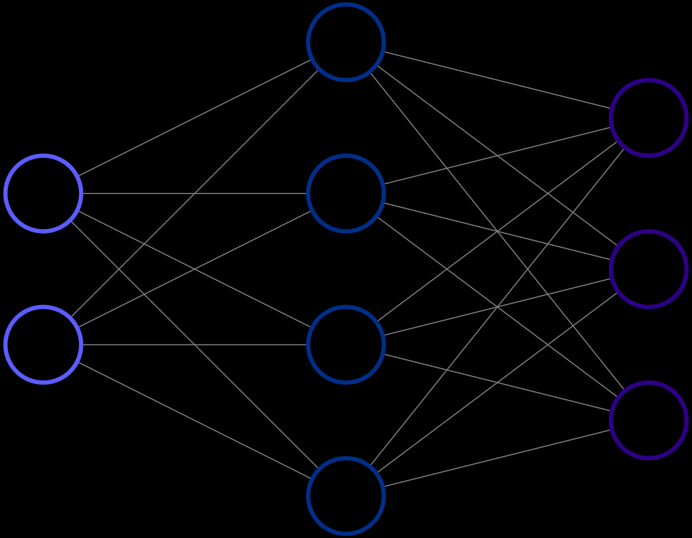
If a proposal with value *v* is chosen, then every higher-numbered proposal *issued by any proposer* has value *v*.

Property 2c

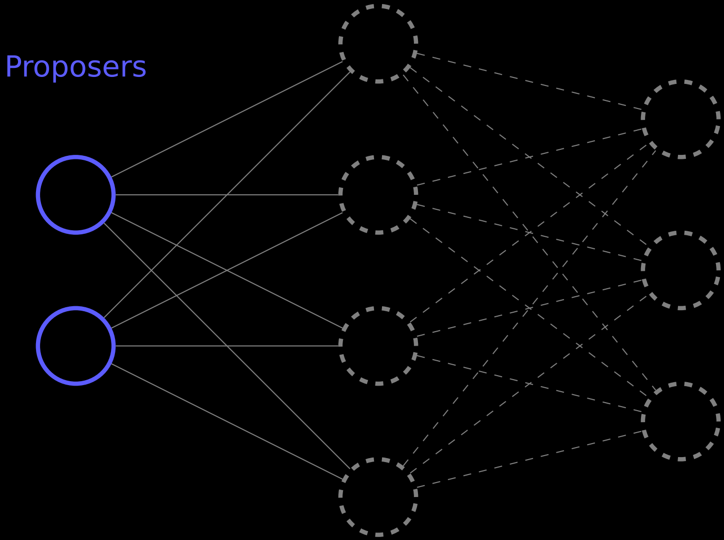
For any v and n, if a proposal with value v and number n is issued, then there is a set S consisting of a majority of acceptors such that either

- no acceptor in S has accepted any proposal numbered less than n, or
- v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S.

Proposers



Proposers



Prepare requests

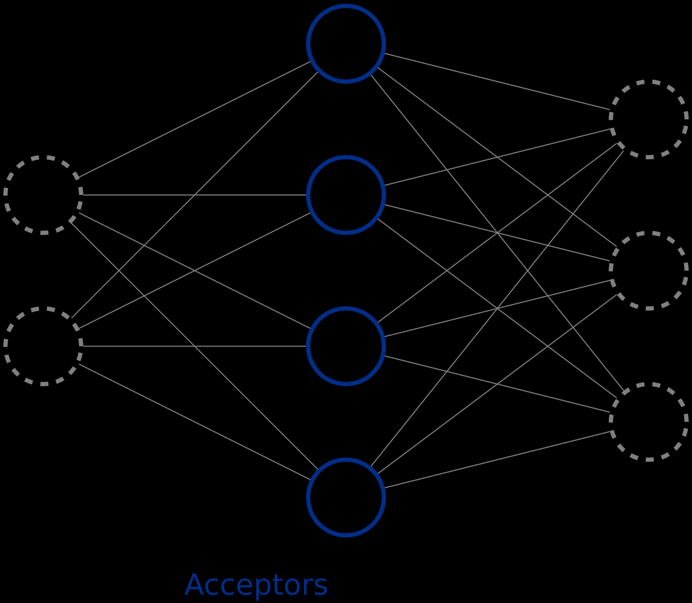
Instead of predicting the future

- Proposer sends **prepare** *n* to acceptors
- Each acceptor replies with
  - A promise to reject lower proposals in future
  - If any, the highest accepted lower proposal

Accept request

- If a majority promise
  - Proposer sends **propose** *n***,** *v*
- If there were accepted proposals
  - *v* must match the highest one
    (Otherwise, *v* can be arbitrary.)

Acceptors



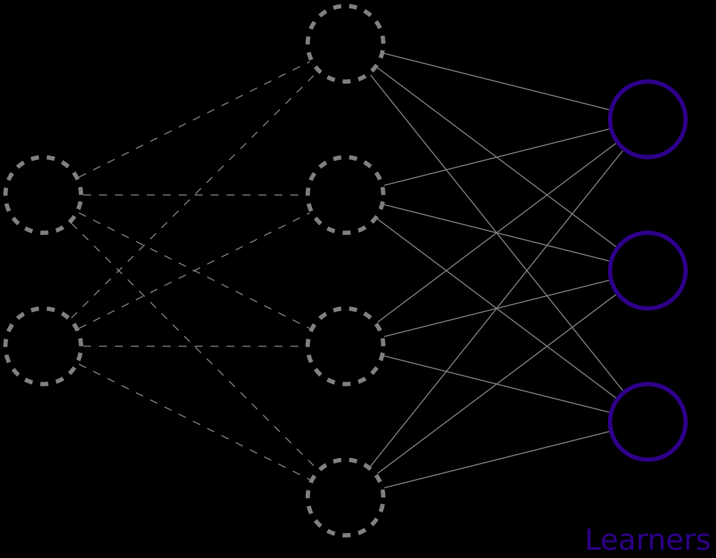
Property 1a

An acceptor can accept a proposal numbered n iff it has not responded to a prepare request having a number greater than n.

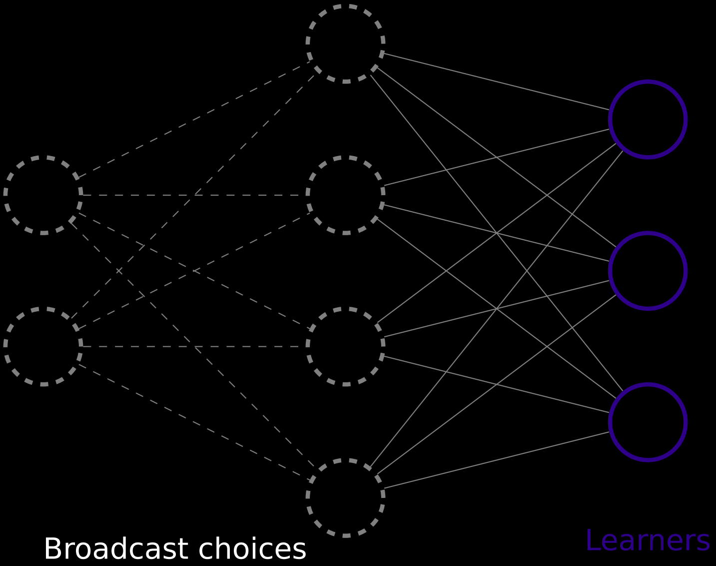
Responding to prepare requests

- An acceptors may respond to any prepare request
- To optimize, ignore requests lower than promised

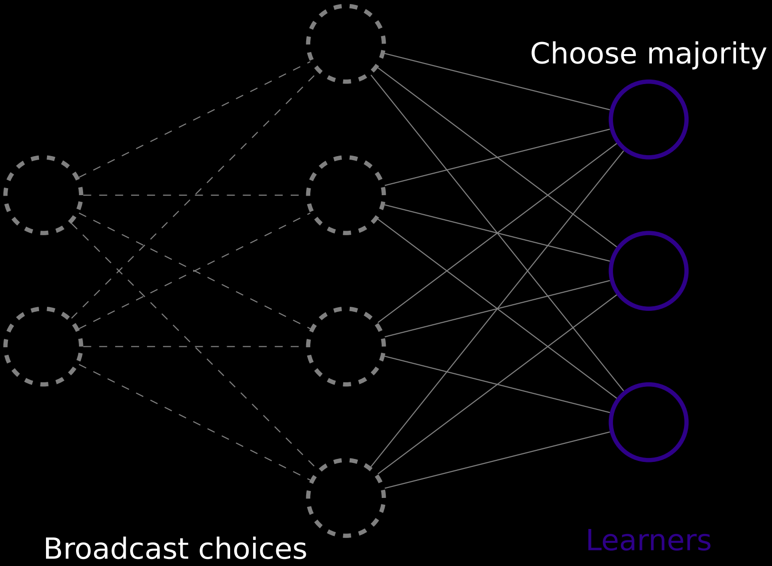
Learners



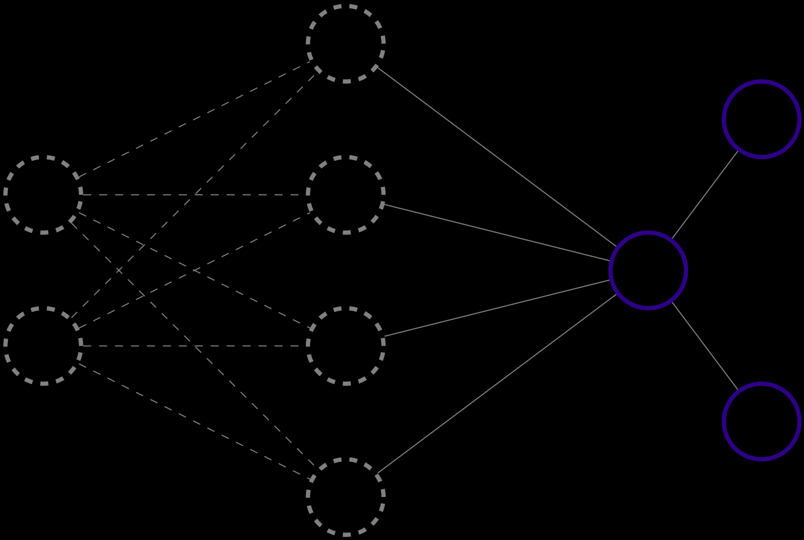
Learners



Learners



Distinguished learner (optimization)



Progress

Progress

1. P_1 receives promises for n_1

Progress

1. P_1 receives promises for n_1
2. P_2 receives promises for $n_2 > n_1$

Progress

1. P_1 receives promises for n_1
2. P_2 receives promises for $n_2 > n_1$
3. P_1 sends proposal numbered n_1, rejected

Progress

1. P_1 receives promises for n_1
2. P_2 receives promises for $n_2 > n_1$
3. P_1 sends proposal numbered n_1, rejected
4. P_1 receives promises for $n_1' > n_2$

Progress

1. P_1 receives promises for n_1
2. P_2 receives promises for $n_2 > n_1$
3. P_1 sends proposal numbered n_1, rejected
4. P_1 receives promises for $n'_1 > n_2$
5. P_2 sends proposal numbered n_2, rejected

Progress

1. P_1 receives promises for n_1
2. P_2 receives promises for $n_2 > n_1$
3. P_1 sends proposal numbered n_1, rejected
4. P_1 receives promises for $n'_1 > n_2$
5. P_2 sends proposal numbered n_2, rejected
6. P_1 receives promises for $n'_2 > n'_1$

Progress

1. P_1 receives promises for n_1
2. P_2 receives promises for $n_2 > n_1$
3. P_1 sends proposal numbered n_1, rejected
4. P_1 receives promises for $n_1' > n_2$
5. P_2 sends proposal numbered n_2, rejected
6. P_1 receives promises for $n_2' > n_1'$
7. P_1 sends proposal numbered n_1', rejected

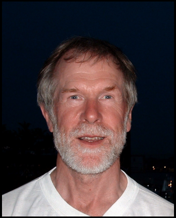
Progress

1. P_1 receives promises for n_1
2. P_2 receives promises for $n_2 > n_1$
3. P_1 sends proposal numbered n_1, rejected
4. P_1 receives promises for $n_1' > n_2$
5. P_2 sends proposal numbered n_2, rejected
6. P_1 receives promises for $n_2' > n_1'$
7. P_1 sends proposal numbered n_1', rejected
8. *ad infinitum*...

Impossibility


Michael Fischer


Nancy Lynch

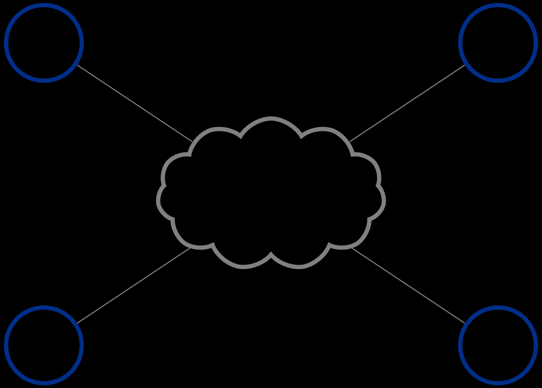

Michael Paterson

Impossibility

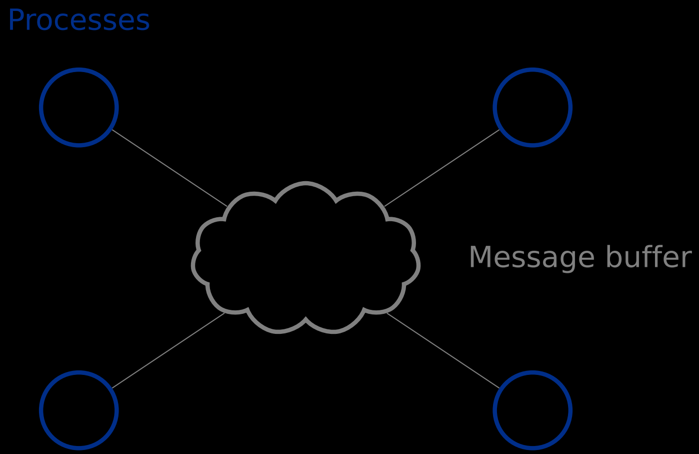
- Impossibility of Distributed Consensus with One Faulty Process (1983)

  > *The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that* **every protocol for this problem has the possibility of nontermination***, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.*

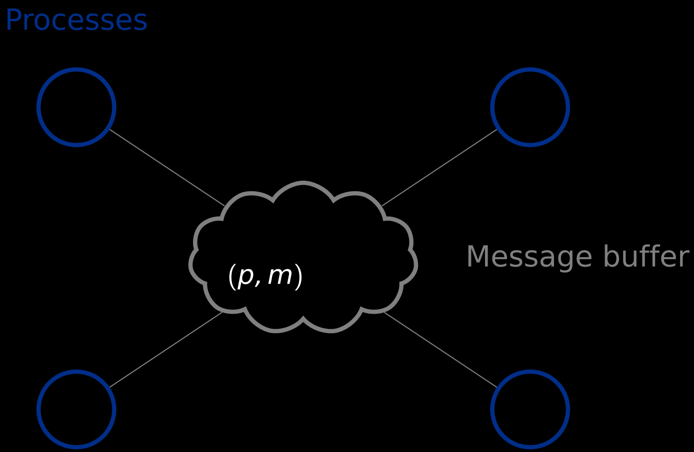
System



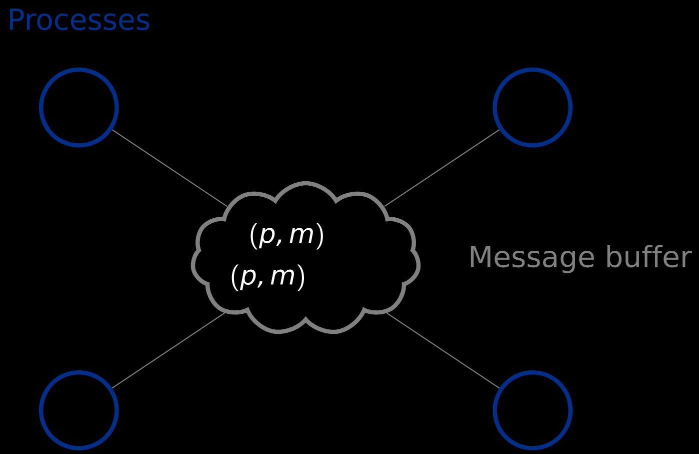
System



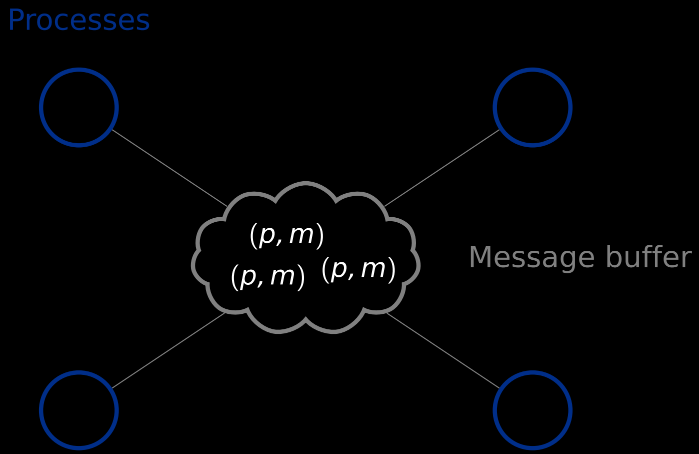
System



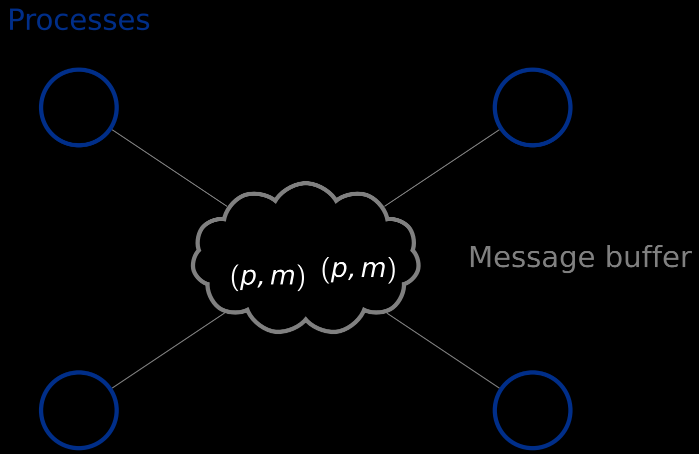
System



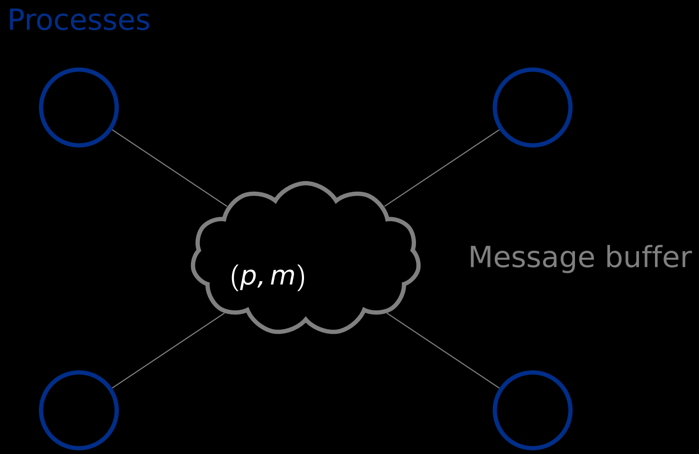
System



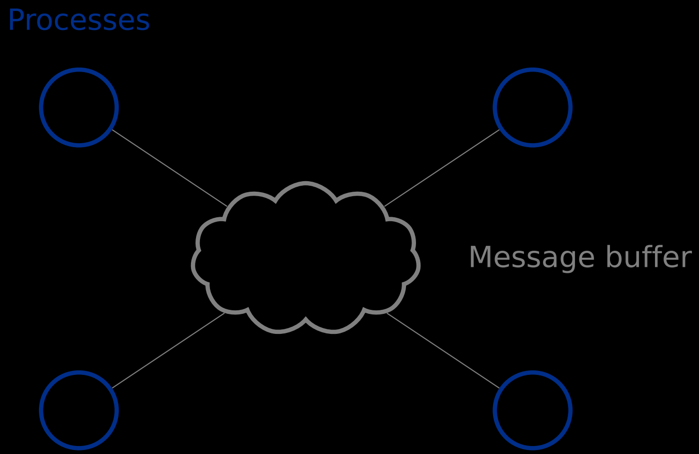
System



System

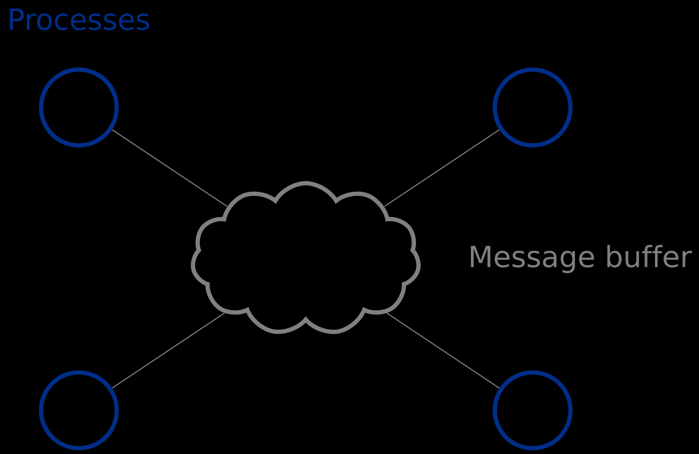


System



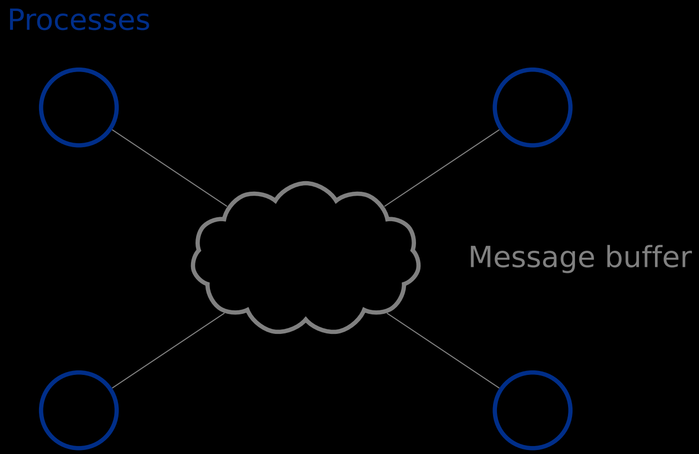
Delivering a message is one *step*

System



The actual message and transition define the *event*

System



The state of each process and the buffer is a *configuration*

More terminology

- **Schedule**: Finite or infinite sequence of events δ that can be applied from configuration C
- **Reachable**: The result of any $\delta(C)$ is reachable from C
- **Accessible**: Reachable from the initial configuration
- **Run**: Sequence of steps associated with a schedule
- **Deciding Run**: Run in which some process decides
- **Bivalent configuration**: Can still decide either value
- **Univalent configuration**: Can only decide a particular value

Partially correct

Encapsulates requirements for a consensus algorithm

- No accessible configuration has more than one decision value (correctness)
- For each $v \in \{0, 1\}$ some accessible configuration has decision value v (non-triviality)

Admissible run

Encapsulates our assumptions about the system

- At most one process is faulty
- All messages sent to nonfaulty processes are eventually received

Totally correct in spite of one fault

- Partially correct (consensus)
- Every admissible run is a deciding run (every possible run will eventually decide, i.e. terminate)

Theorem

No consensus protocol is totally correct in spite of one fault (i.e. for any correct consensus algorithm, under our system assumptions, at least one conceivable run will never terminate)

Lemma 1

Roughly, schedules are commutative

Lemma 2

There is a bivalent initial configuration

Lemma 3

Let C be a bivalent configuration of P, and let $e = (p, m)$ be an event that is applicable to C. Let $\mathcal{C}$ be the set of configurations reachable from C without applying e, and let $\mathcal{D} = e(\mathcal{C})$. Then, $\mathcal{D}$ contains a bivalent configuration.

An admissible run

- Order processes arbitrarily in a queue
- Order message buffer earliest to latest
- Divide into stages, each stage ending when head of queue processes its first message and gets moved to back of queue

A non-deciding admissible run

- Begin in a bivalent initial configuration (Lemma 2)
- Schedule messages within stage to guarantee ending in a bivalent configuration (Lemma 3)

Conclusions

- Consensus is impossible.
- But you can do it.
- Paxos works well in practice and is very famous.
- Other systems exist that make different system assumptions, terminate with probability 1, . . .