

Snapshots

A **snapshot** of an execution of a distributed algorithm should return a configuration of an execution *in the same computation*.

Snapshots can be used for:

- ▶ Restarting after a failure.
- ▶ Debugging.
- ▶ Off-line determination of **stable properties**, which remain true as soon as they have become true.
Examples: deadlock, garbage.



Challenge: Take a snapshot without (temporarily) freezing the execution.

Snapshots

A **snapshot** of an execution of a distributed algorithm should return a configuration of an execution *in the same computation*.

Snapshots can be used for:

- ▶ Restarting after a failure.
- ▶ Debugging.
- ▶ Off-line determination of **stable properties**, which remain true as soon as they have become true.
Examples: deadlock, garbage.



Challenge: Take a snapshot without (temporarily) freezing the execution.

Snapshots

A **snapshot** of an execution of a distributed algorithm should return a configuration of an execution *in the same computation*.

Snapshots can be used for:

- ▶ Restarting after a failure.
- ▶ Debugging.
- ▶ Off-line determination of **stable properties**, which remain true as soon as they have become true.
Examples: deadlock, garbage.



Challenge: Take a snapshot without (temporarily) freezing the execution.

Snapshots

We distinguish **basic** messages of the underlying **distributed algorithm** and **control** messages of the **snapshot algorithm**.

A **snapshot** of a (basic) execution consists of:

- ▶ a **local snapshot** of the (basic) state of each process, and
- ▶ the **channel state** of (basic) messages in transit for each channel.

A snapshot is **meaningful** if it is a configuration of an execution in the same computation as the actual execution.

Snapshots

We distinguish **basic** messages of the underlying **distributed algorithm** and **control** messages of the **snapshot algorithm**.

A **snapshot** of a (basic) execution consists of:

- ▶ a **local snapshot** of the (basic) state of each process, and
- ▶ the **channel state** of (basic) messages in transit for each channel.

A snapshot is **meaningful** if it is a configuration of an execution in the same computation as the actual execution.

Snapshots

We distinguish **basic** messages of the underlying **distributed algorithm** and **control** messages of the **snapshot algorithm**.

A **snapshot** of a (basic) execution consists of:

- ▶ a **local snapshot** of the (basic) state of each process, and
- ▶ the **channel state** of (basic) messages in transit for each channel.

A snapshot is **meaningful** if it is a configuration of an execution in the same computation as the actual execution.

We need to avoid the following situations.

1. Process p takes a local snapshot, and then sends a message m to process q , where:
 - q takes a local snapshot after the receipt of m ,
 - or m is included in the channel state of pq .
2. p sends m to q , and then takes a local snapshot, where:
 - q takes a local snapshot before the receipt of m ,
 - and m is not included in the channel state of pq .

We need to avoid the following situations.

1. Process p takes a local snapshot, and then sends a message m to process q , where:
 - q takes a local snapshot after the receipt of m ,
 - or m is included in the channel state of pq .
2. p sends m to q , and then takes a local snapshot, where:
 - q takes a local snapshot before the receipt of m ,
 - and m is not included in the channel state of pq .

Chandy-Lamport algorithm

Consider a **directed** network with *FIFO* channels.

Initiators take a **local snapshot** of their state, and send a control message **⟨marker⟩** to their neighbors.

When a process that hasn't yet taken a snapshot receives **⟨marker⟩**, it

- ▶ takes a **local snapshot** of its state, and
- ▶ sends **⟨marker⟩** to all its neighbors.

Process q computes as **channel state** of pq

If channels are FIFO, this produces a meaningful snapshot.

Message complexity: $\Theta(E)$

Worst-case time complexity: $O(D)$

Chandy-Lamport algorithm

Consider a **directed** network with *FIFO* channels.

Initiators take a **local snapshot** of their state, and send a control message **⟨marker⟩** to their neighbors.

When a process that hasn't yet taken a snapshot receives **⟨marker⟩**, it

- ▶ takes a **local snapshot** of its state, and
- ▶ sends **⟨marker⟩** to all its neighbors.

Process q computes as **channel state** of pq

If channels are FIFO, this produces a meaningful snapshot.

Message complexity: $\Theta(E)$

Worst-case time complexity: $O(D)$

Chandy-Lamport algorithm

Consider a **directed** network with *FIFO* channels.

Initiators take a **local snapshot** of their state, and send a control message **⟨marker⟩** to their neighbors.

When a process that hasn't yet taken a snapshot receives **⟨marker⟩**, it

- ▶ takes a **local snapshot** of its state, and
- ▶ sends **⟨marker⟩** to all its neighbors.

Process q computes as **channel state** of pq

If channels are FIFO, this produces a meaningful snapshot.

Message complexity: $\Theta(E)$

Worst-case time complexity: $O(D)$

Chandy-Lamport algorithm

Consider a **directed** network with *FIFO* channels.

Initiators take a **local snapshot** of their state, and send a control message **⟨marker⟩** to their neighbors.

When a process that hasn't yet taken a snapshot receives **⟨marker⟩**, it

- ▶ takes a **local snapshot** of its state, and
- ▶ sends **⟨marker⟩** to all its neighbors.

Process q computes as **channel state** of pq the messages it receives via pq after taking its local snapshot and before receiving **⟨marker⟩** from p .

If channels are FIFO, this produces a meaningful snapshot.

Message complexity: $\Theta(E)$

Worst-case time complexity: $O(D)$

Chandy-Lamport algorithm

Consider a **directed** network with *FIFO* channels.

Initiators take a **local snapshot** of their state, and send a control message **⟨marker⟩** to their neighbors.

When a process that hasn't yet taken a snapshot receives **⟨marker⟩**, it

- ▶ takes a **local snapshot** of its state, and
- ▶ sends **⟨marker⟩** to all its neighbors.

Process q computes as **channel state** of pq the messages it receives via pq after taking its local snapshot and before receiving **⟨marker⟩** from p .

If channels are FIFO, this produces a meaningful snapshot.

Message complexity: $\Theta(E)$

Worst-case time complexity: $O(D)$

Chandy-Lamport algorithm

Consider a **directed** network with *FIFO* channels.

Initiators take a **local snapshot** of their state, and send a control message **⟨marker⟩** to their neighbors.

When a process that hasn't yet taken a snapshot receives **⟨marker⟩**, it

- ▶ takes a **local snapshot** of its state, and
- ▶ sends **⟨marker⟩** to all its neighbors.

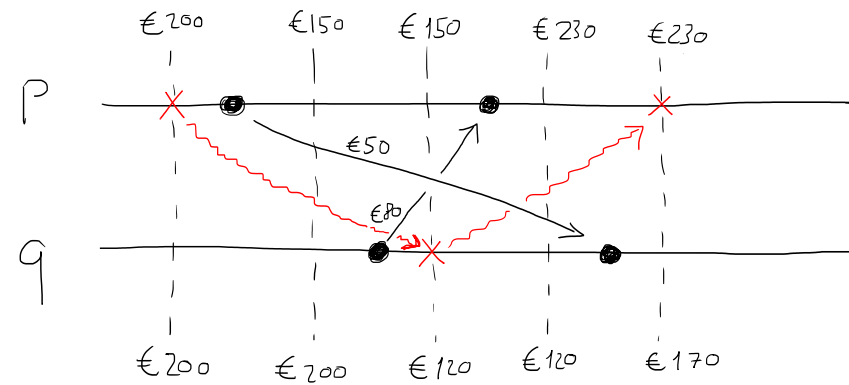
Process q computes as **channel state** of pq the messages it receives via pq after taking its local snapshot and before receiving **⟨marker⟩** from p .

If channels are FIFO, this produces a meaningful snapshot.

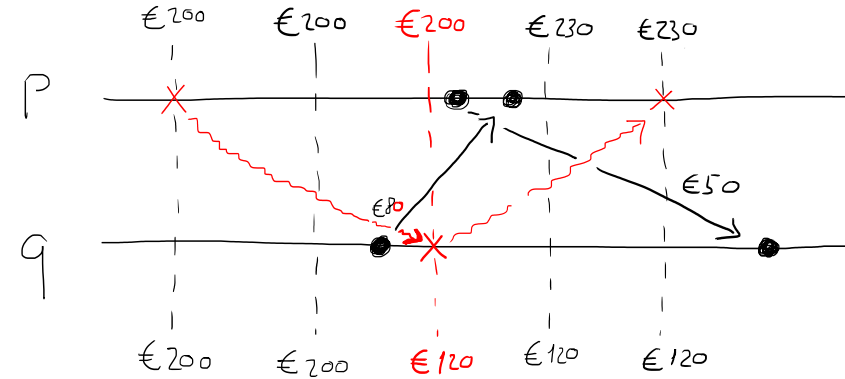
Message complexity: $\Theta(E)$

Worst-case time complexity: $O(D)$

Note: a snapshot may not correspond to a configuration in the actual execution

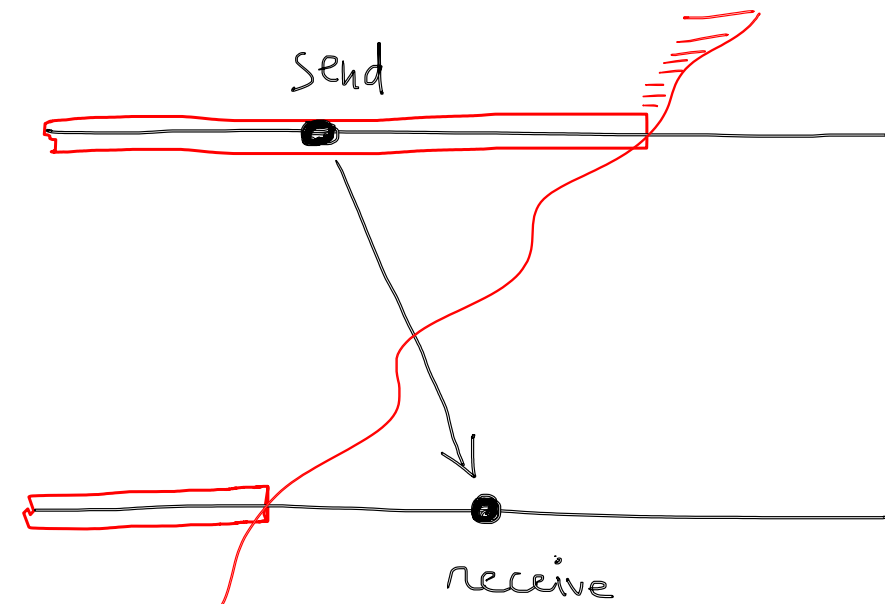


Snapshot
 $\left\{ \begin{array}{l} p: \text{€}200, q_p: \text{€}80 \\ q: \text{€}120, p_q: \text{€}0 \end{array} \right.$



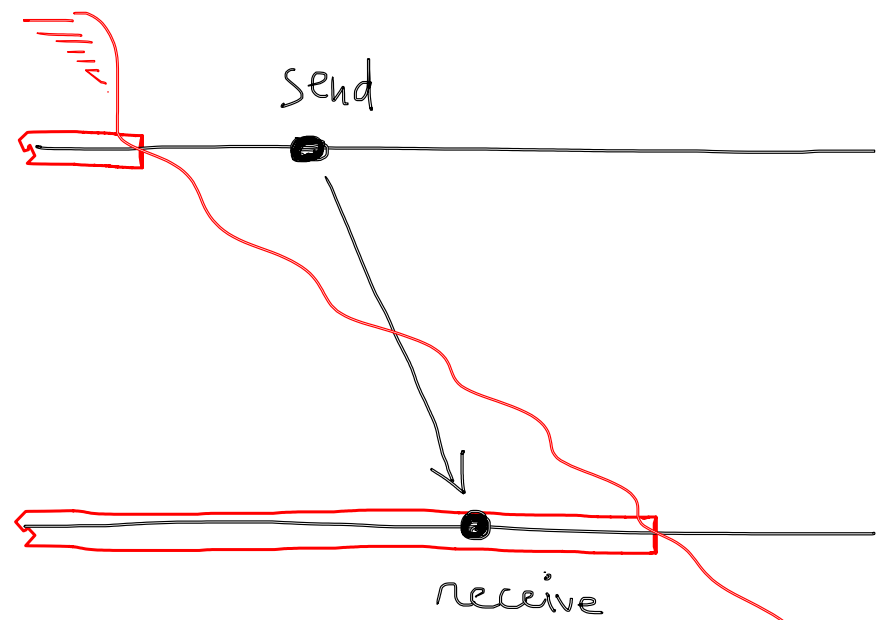
Snapshot
 $\left\{ \begin{array}{l} p: \text{€}150, q_p: \text{€}80 \\ q: \text{€}170, p_q: \text{€}0 \end{array} \right.$

Snapshot = configuration through a **consistent cut**



consistent cut

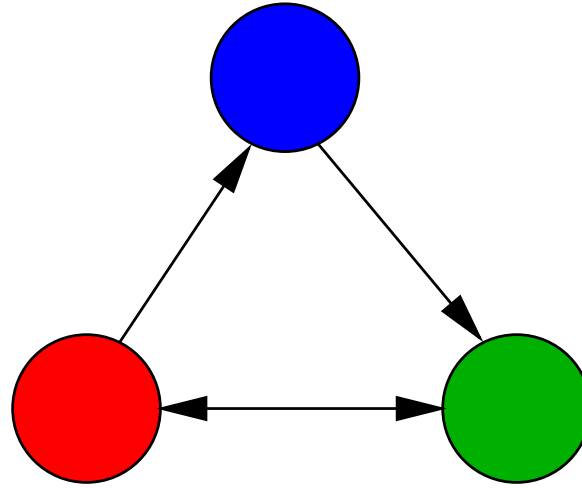
$\forall e \in \text{cut}$
 $f < e \Rightarrow f \in \text{cut}$



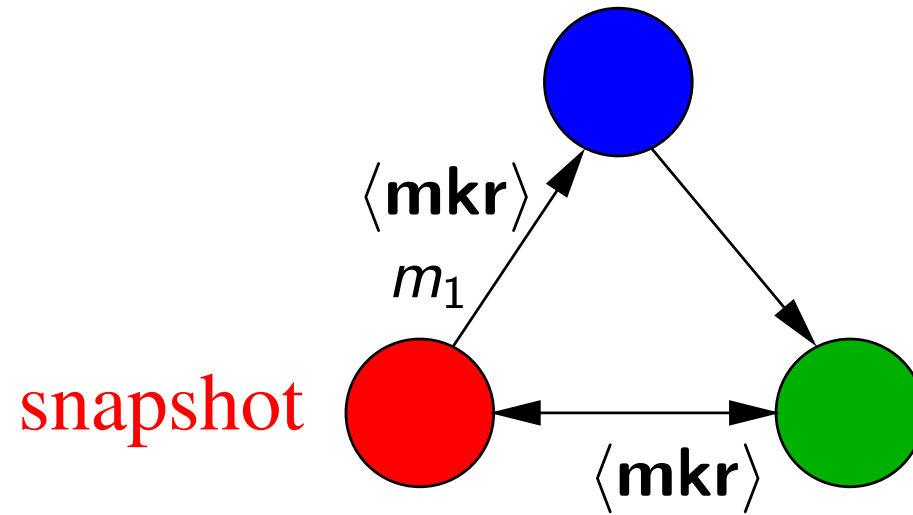
inconsistent cut

$\exists e \in \text{cut}$
 $f < e \ \& \ f \notin \text{cut}$

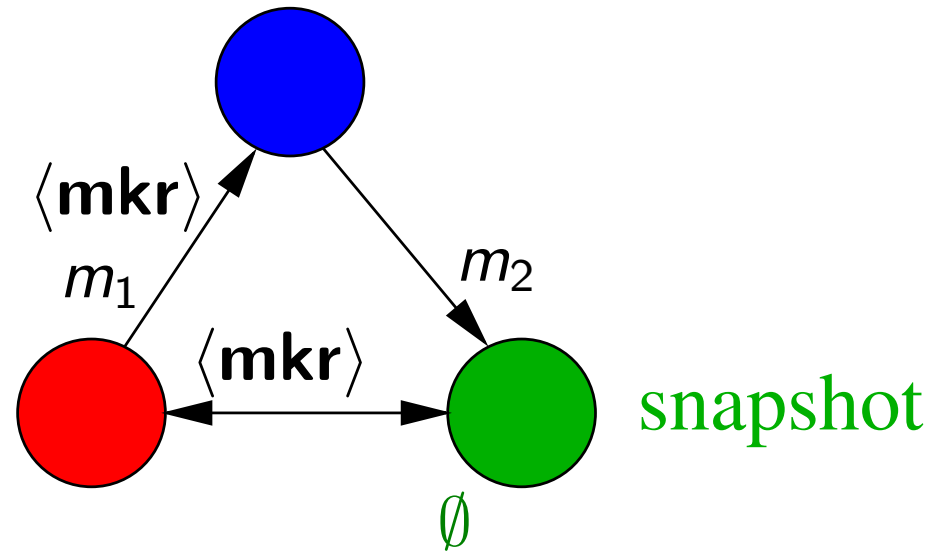
Chandy-Lamport algorithm - Example



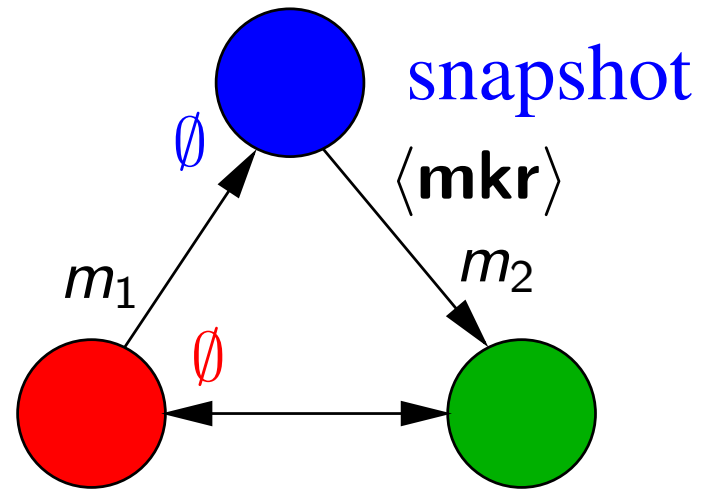
Chandy-Lamport algorithm - Example



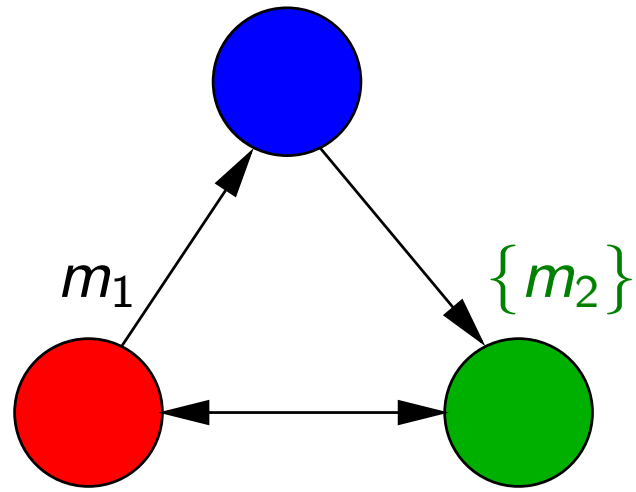
Chandy-Lamport algorithm - Example



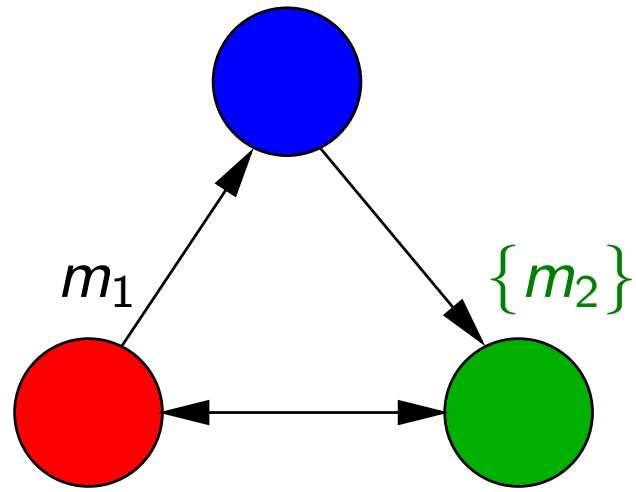
Chandy-Lamport algorithm - Example



Chandy-Lamport algorithm - Example

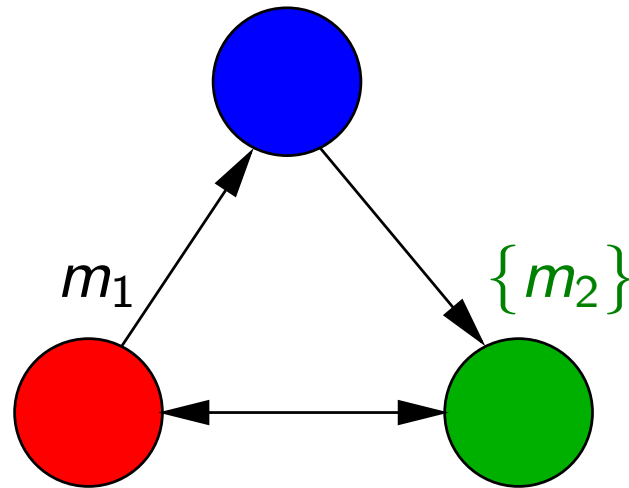


Chandy-Lamport algorithm - Example



The snapshot (processes red/blue/green, channels $\emptyset, \emptyset, \emptyset, \{m_2\}$) isn't a configuration in the actual execution.

Chandy-Lamport algorithm - Example



The snapshot (processes red/blue/green, channels $\emptyset, \emptyset, \emptyset, \{m_2\}$) isn't a configuration in the actual execution.

The send of m_1 isn't causally before the send of m_2 .

So the snapshot is a configuration of an execution that is in the same computation as the actual execution.

Chandy-Lamport algorithm - Correctness

Claim: If a **post-snapshot** event e is causally before an event f , then f is also post-snapshot.

This implies that the snapshot is a configuration of an execution that is in the same *computation* as the actual execution.

Proof: The case that e and f occur at the same process is trivial.

Let e be a send and f the corresponding receive event.

Let e occur at p and f at q .

e is post-snapshot at p , so p sent **marker** to q before e .

Channels are FIFO, so q receives this **marker** before f .

Hence f post-snapshot at q .

Chandy-Lamport algorithm - Correctness

Claim: If a **post-snapshot** event e is causally before an event f , then f is also post-snapshot.

This implies that the snapshot is a configuration of an execution that is in the same *computation* as the actual execution.

Proof: The case that e and f occur at the same process is trivial.

Let e be a send and f the corresponding receive event.

Let e occur at p and f at q .

e is post-snapshot at p , so p sent $\langle \mathbf{marker} \rangle$ to q before e .

Channels are FIFO, so q receives this $\langle \mathbf{marker} \rangle$ before f .

Hence f post-snapshot at q .

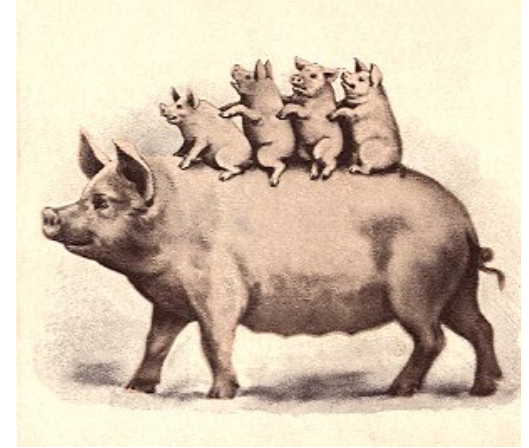
Question

How can the Chandy-Lamport algorithm be adapted to obtain a snapshot algorithm that works for non-FIFO channels?

Lai-Yang algorithm

Suppose channels are *non-FIFO*. We use **piggybacking**.

Initiators take a **local snapshot** of their state.



When a process has taken its local snapshot, it appends *true* to each outgoing basic message.

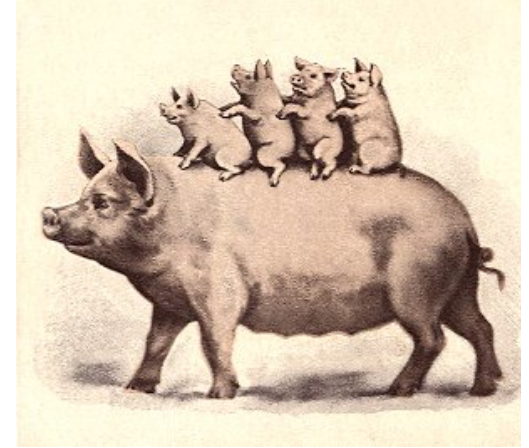
When a process that hasn't yet taken a snapshot receives a message with *true* or a *control message* (see next slide) for the first time, it takes a **local snapshot** of its state *before reception of this message*.

Process q computes as **channel state** of pq

Lai-Yang algorithm

Suppose channels are *non-FIFO*. We use **piggybacking**.

Initiators take a **local snapshot** of their state.



When a process has taken its local snapshot, it appends *true* to each outgoing basic message.

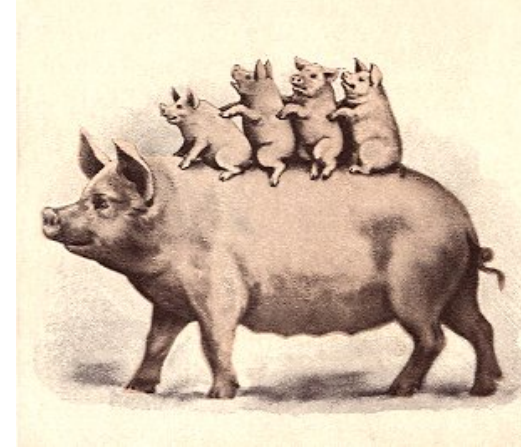
When a process that hasn't yet taken a snapshot receives a message with *true* or a *control message* (see next slide) for the first time, it takes a **local snapshot** of its state *before reception of this message*.

Process q computes as **channel state** of pq

Lai-Yang algorithm

Suppose channels are *non-FIFO*. We use **piggybacking**.

Initiators take a **local snapshot** of their state.



When a process has taken its local snapshot, it appends *true* to each outgoing basic message.

When a process that hasn't yet taken a snapshot receives a message with *true* or a *control message* (see next slide) for the first time, it takes a **local snapshot** of its state *before reception of this message*.

Process q computes as **channel state** of pq the basic messages without the tag *true* that it receives via pq after its local snapshot.

Lai-Yang algorithm - Control messages

Question: How does q know when it can determine the channel state of pq ?

Lai-Yang algorithm - Control messages

Question: How does q know when it can determine the channel state of pq ?

p sends a **control message** to q , informing q how many basic messages without the tag *true* p sent into pq .

These control messages also ensure that all processes eventually take a local snapshot.

Lai-Yang algorithm - Multiple snapshots

Question: How can multiple subsequent snapshots be supported ?

Lai-Yang algorithm - Multiple snapshots

Question: How can multiple subsequent snapshots be supported ?

Answer: Each snapshot is provided with a sequence number.

Basic message carry the sequence number of the last snapshot at the sender (instead of *true* or *false*).

Control messages carry the sequence number of their snapshot.