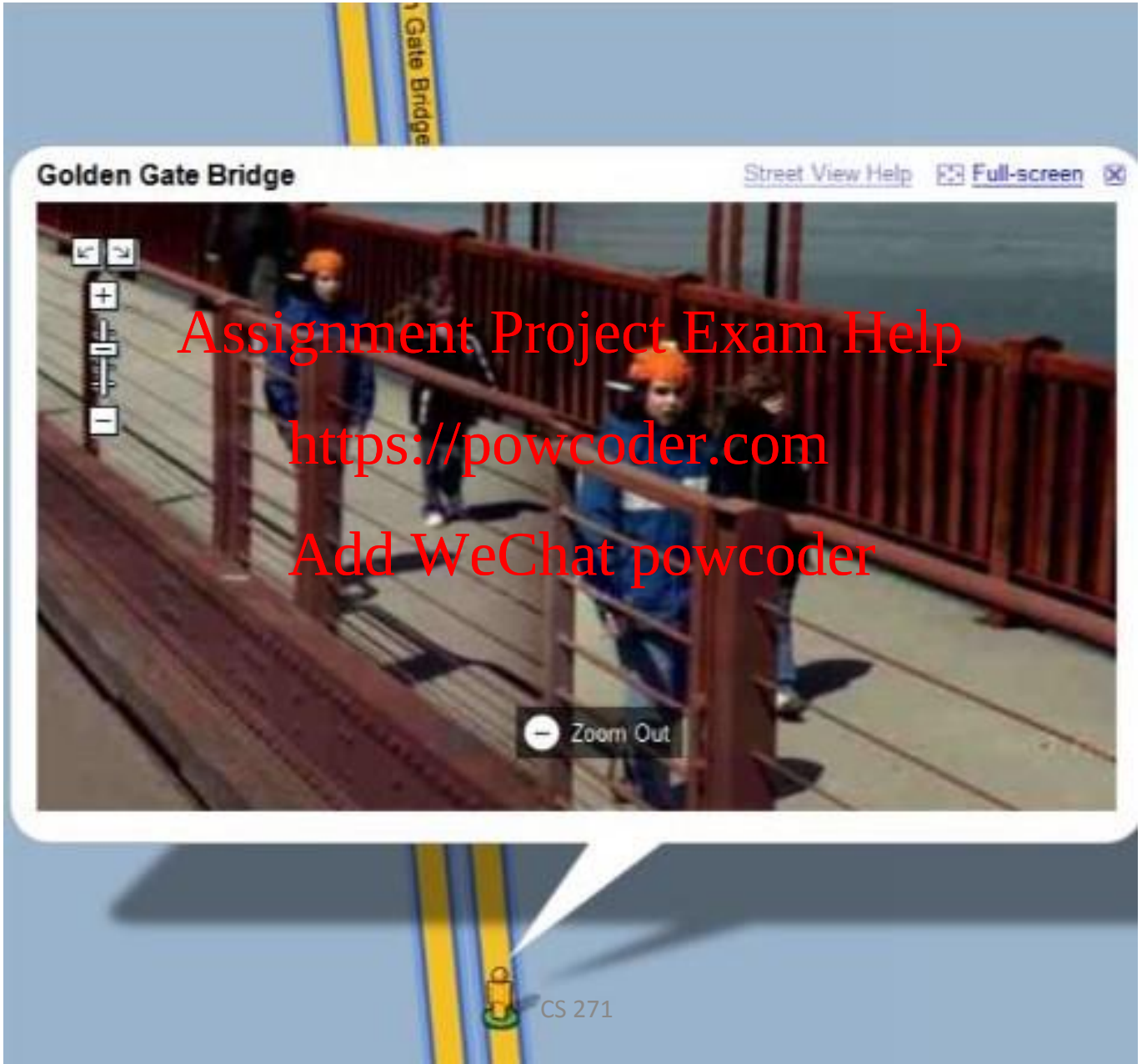
GLOBAL STATES AND CHECKPOINTS

Motivation



Detecting global properties

- Want to discover if a property holds in a distributed system
- Three examples:
  - Distributed garbage collection: if there are no longer any references to objects, the memory taken up by the objects should be reclaimed.
  - Distributed deadlock detection: when each of a collection of processes waits for another process to send it a message, and where there is a cycle in the graph of this "wait-for" relationship.
  - Distributed termination detection: detect if a distributed algorithm has terminated. Need to test if each process has halted and no more messages in the network.

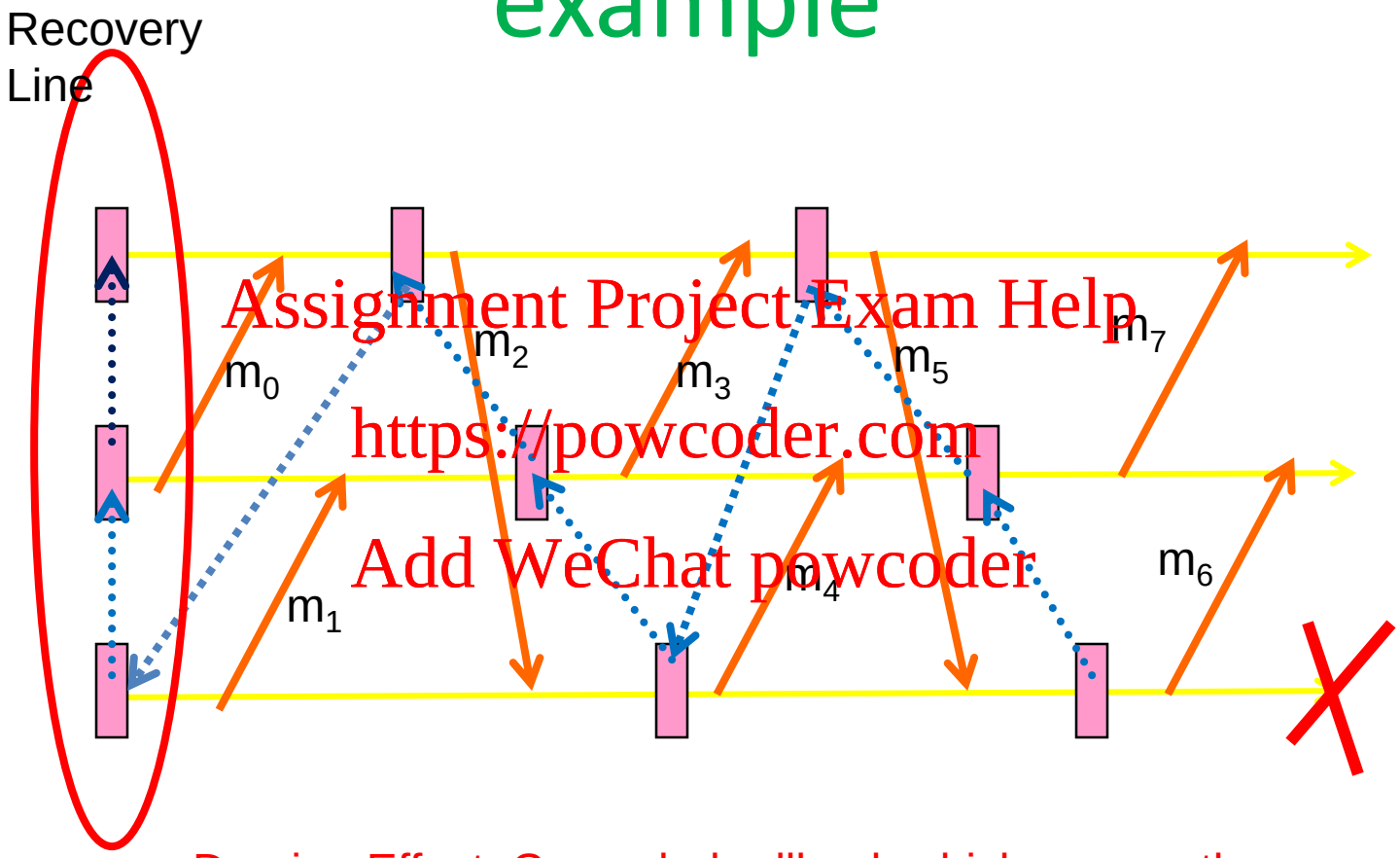
Distributed Checkpoints and Rollback Recovery

- Fault tolerance is achieved by periodically using stable storage to save the processes' states during the failure-free execution.
- Upon a failure, a failed process rolls back from one of its saved states, thereby reducing the amount of lost computation.
- Each of the saved states is called a checkpoint

Checkpoint based Recovery

- **Uncoordinated checkpointing**: Each process takes its checkpoints independently
- **Coordinated checkpointing**: Processes coordinate their checkpoints in order to save a system-wide consistent state.

Domino effect: uncoordinated example

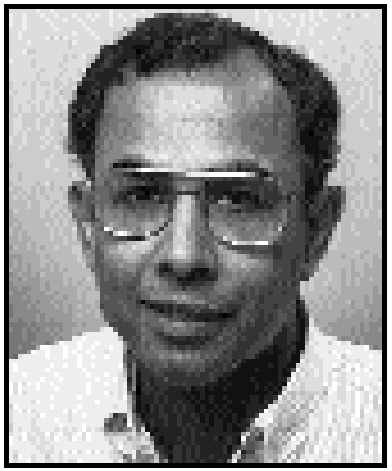


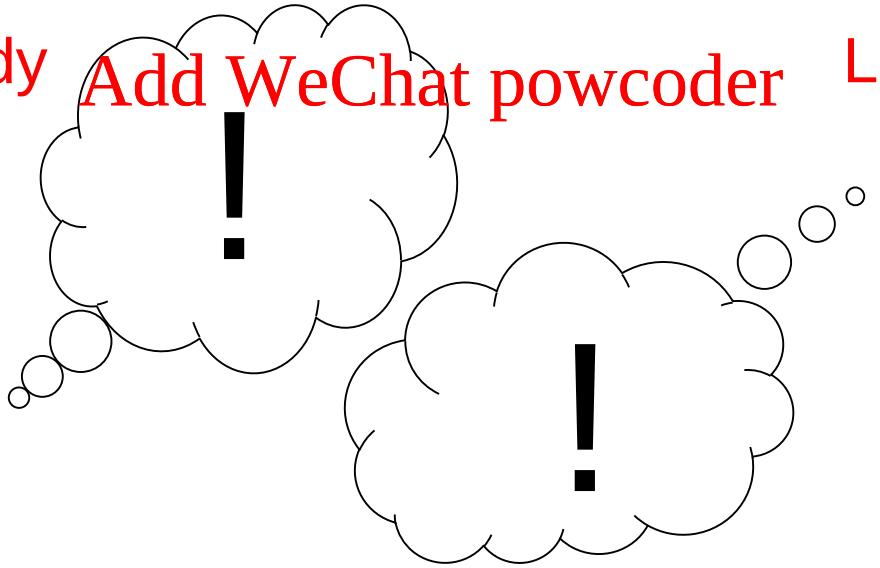
Domino Effect: Cascaded rollback which causes the system to roll back too far in the computation (even to the beginning), in spite of all the checkpoints

Coordinated Non-blocking

- Processes could coordinate, but …
- Do we really need to block …?

K. Mani Chandy





Leslie Lamport

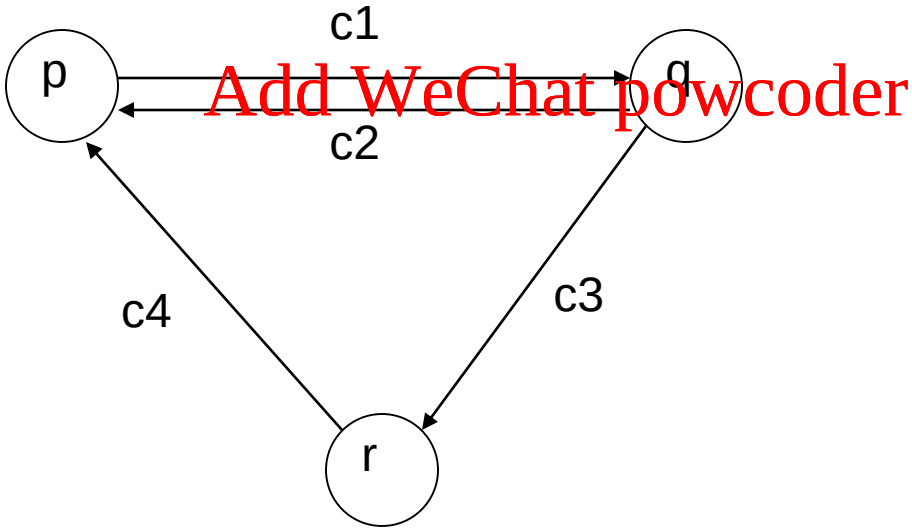
Global State

Chandy and Lamport—TOCS 1985

- Global state of a distributed system
  - Local state of each process
  - Messages sent but not received
- Many applications need the state of the system
  - Failure recovery, distributed deadlock detection
  - Detect stable properties.
- Problem: how can you figure out the state of a distributed system?
  - Each process is independent
  - Network does not have any processing power.
- Distributed snapshot: a **consistent global state**

Distributed System Model

- Assume each process communicates with another process using unidirectional FIFO point-to-point channels (e.g, TCP connections)



A Simple Example

A Variant of producer-consumer example

- Producer code:

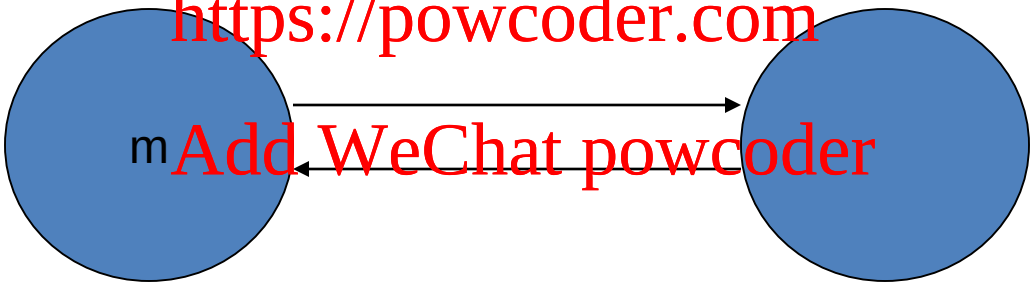
```
while (1)
{
        produce m;
        send m;


        wait for ack;
}
```

- Consumer code:

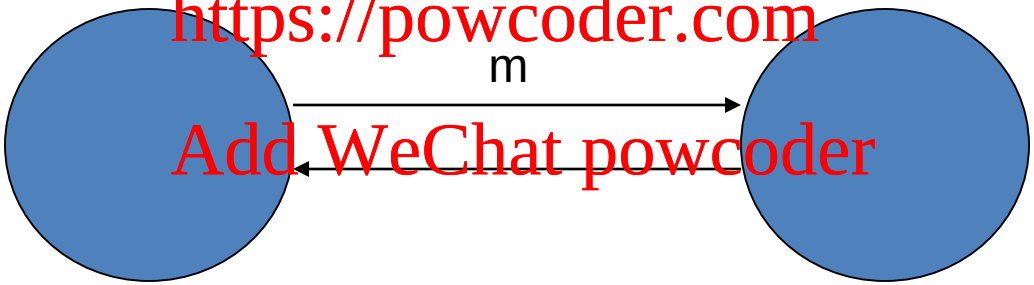
```
while (1)
{
        recv m;
        consume m;
        send ack;
}
```

Example: Initial State



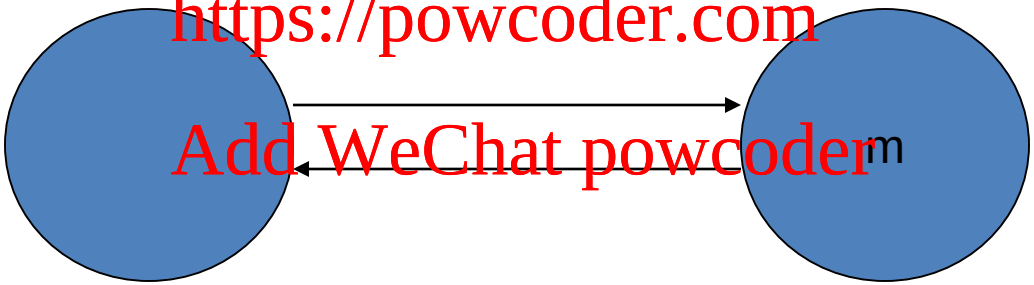
Producer **Consumer**

Example



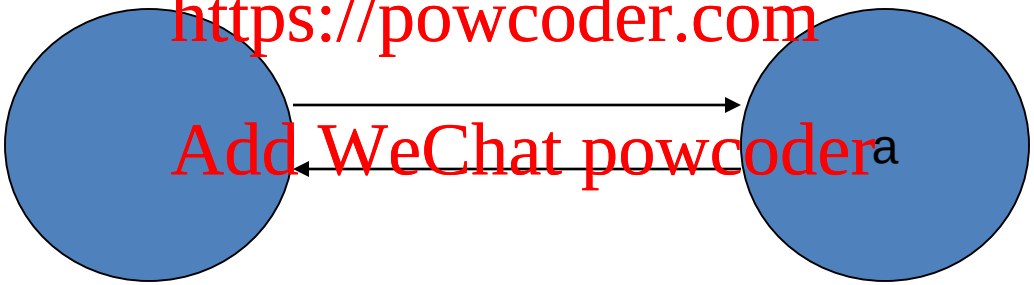
Producer **Consumer**

Example



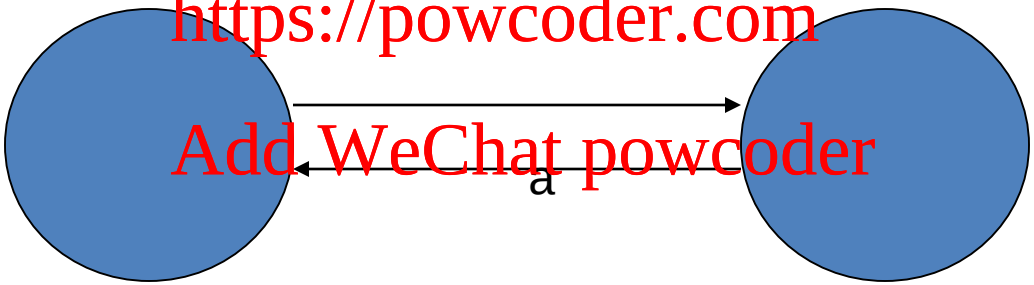
Producer **Consumer**

Example



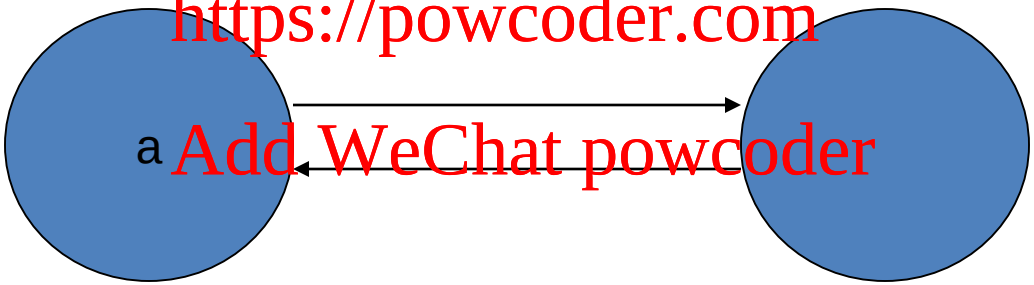
Producer **Consumer**

Example



Producer **Consumer**

Example



Producer **Consumer**

A naïve snapshot algorithm

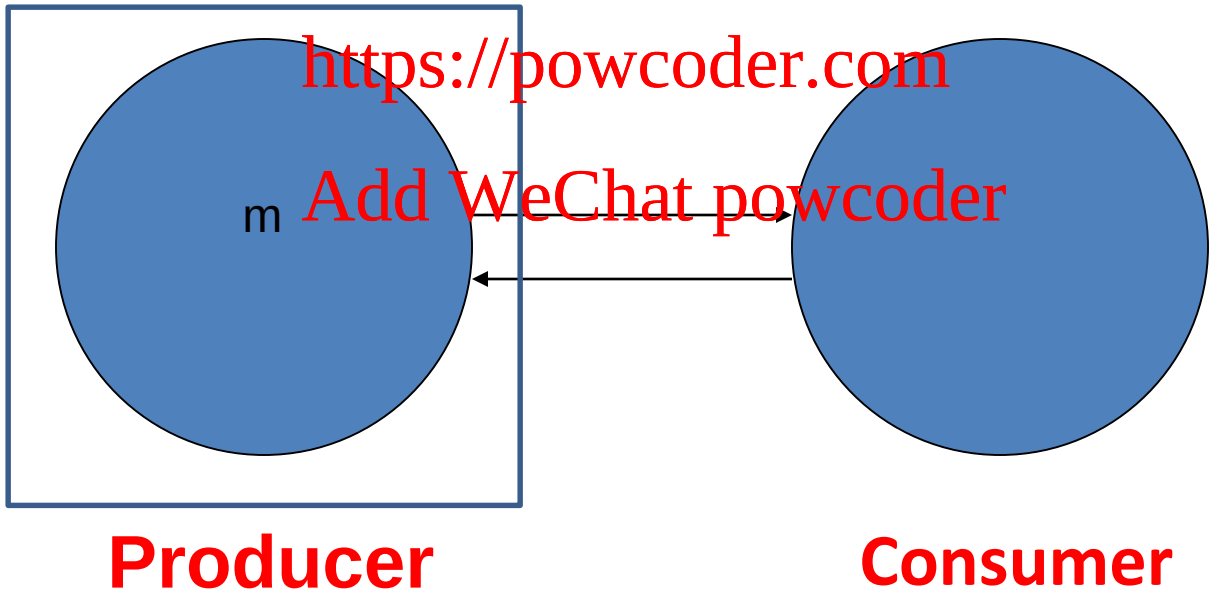
- Processes record their state at *any* arbitrary point
- A designated process collects these states

\+ So simple!!

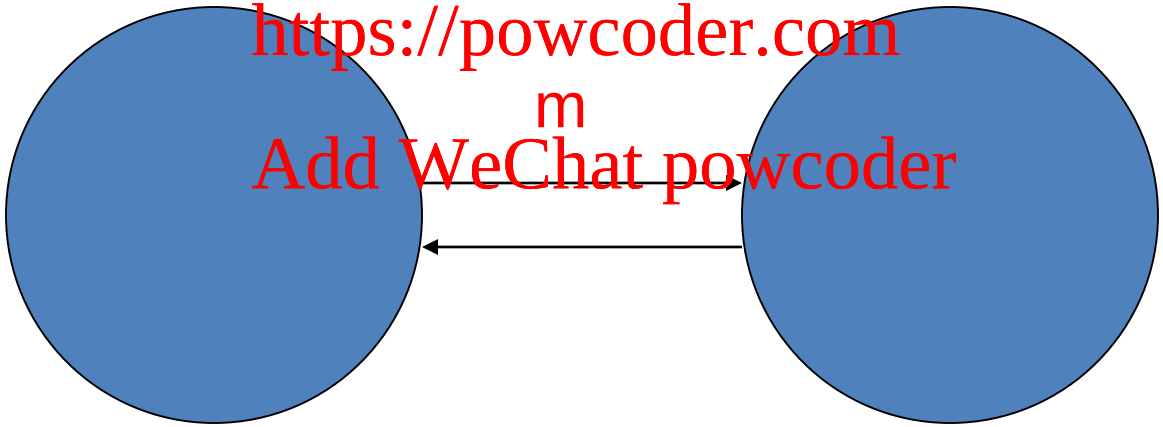
\- Correct??

Example
Producer Consumer problem

Producer records its state



Example

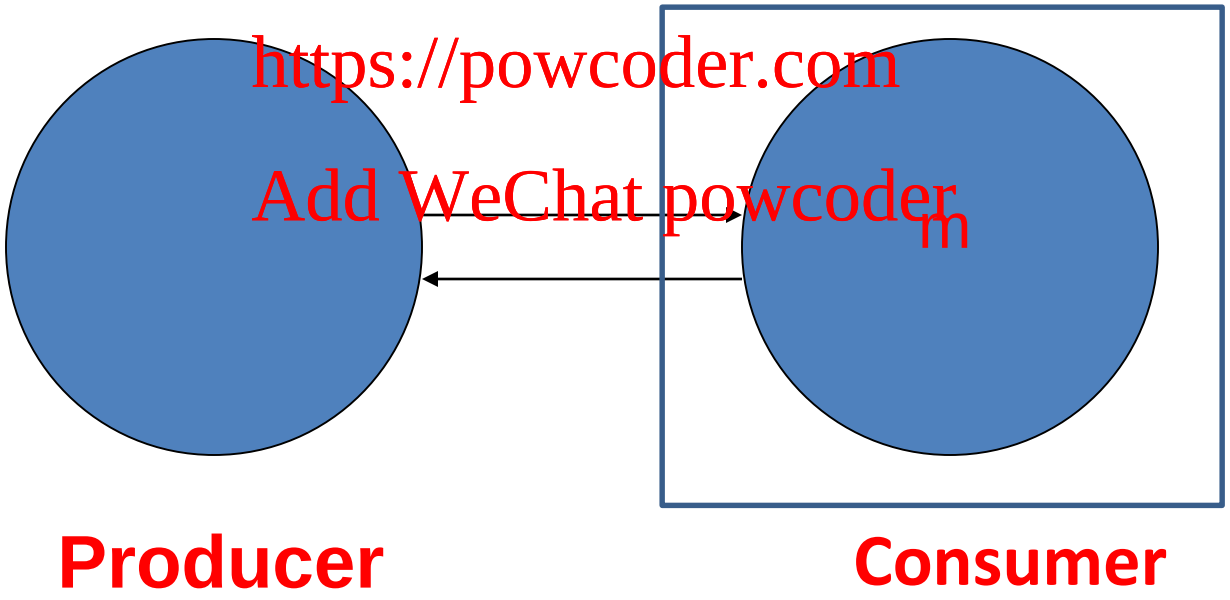


Producer

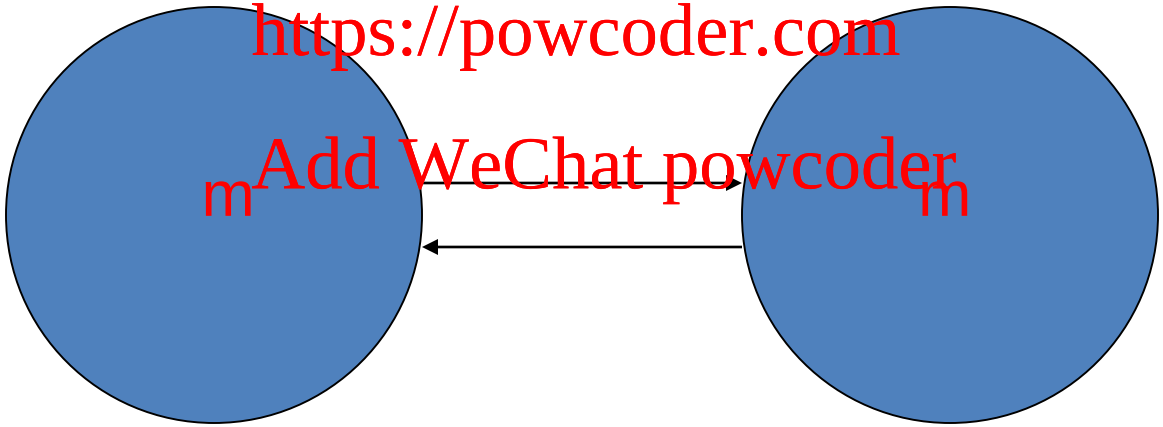
Consumer

Example

Consumer records its state

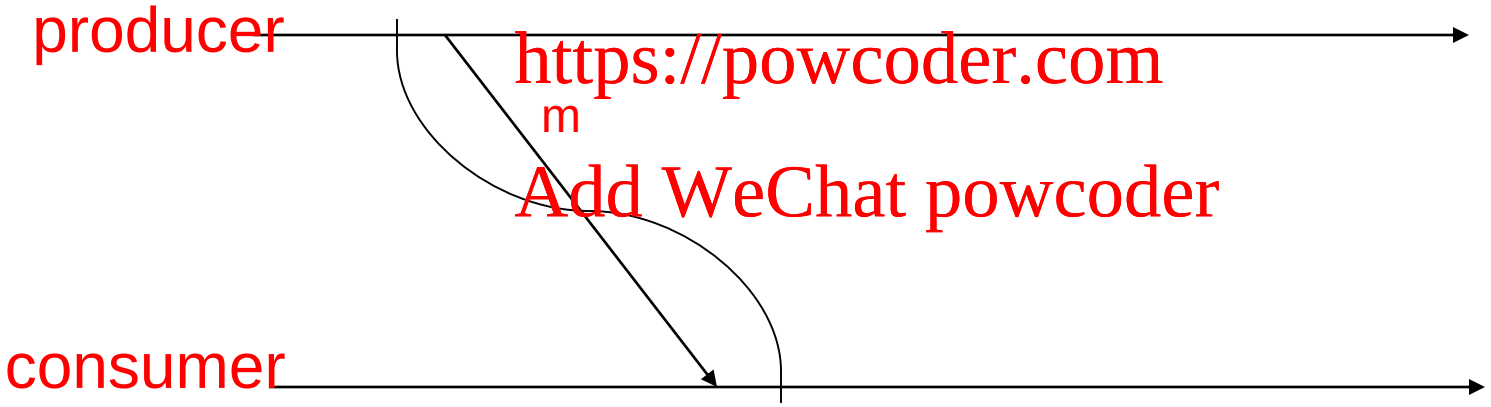


Example
The recorded state



Where did we err?

- What did we do wrong?



Error!!

- The sender has *no* record of the sending
- The receiver *has* the record of the receipt
- Result
  - Global state has record of the receive event but no send event – violating the happened before concept!!

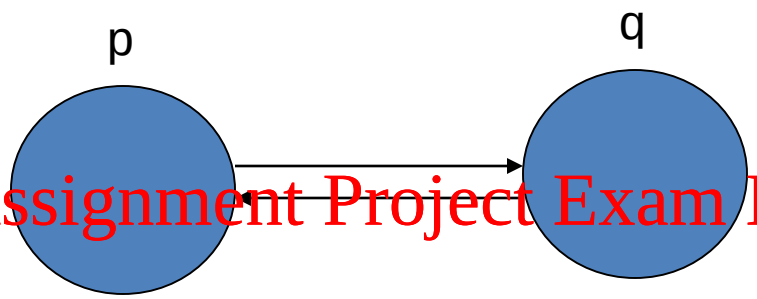
The Notion of Consistency

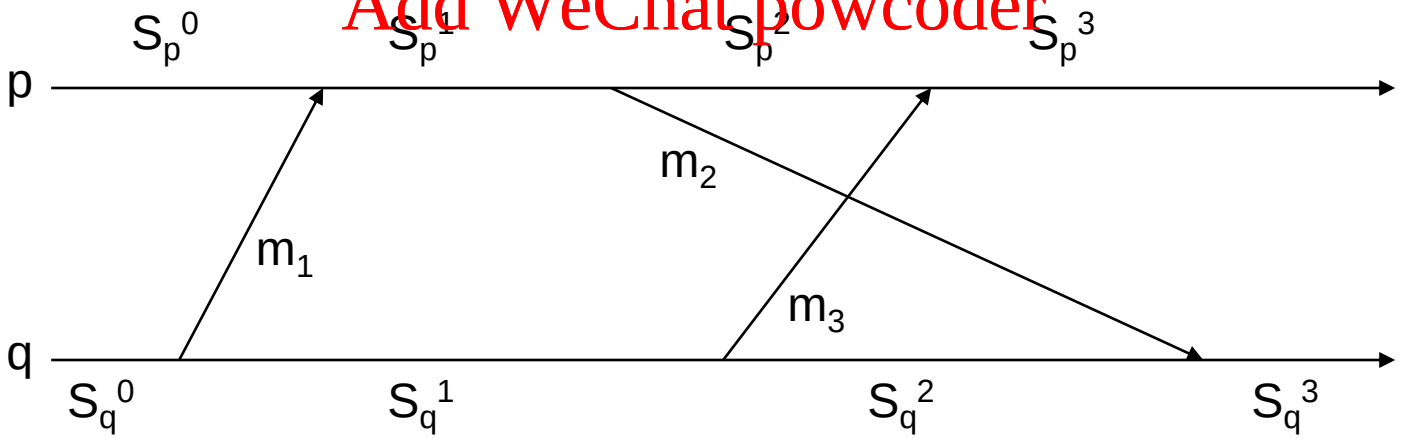
- A global state is consistent if it *could* have been observed by an external observer

- If a $\longrightarrow$ b then it is never the case that b is observed by an external observer and not a

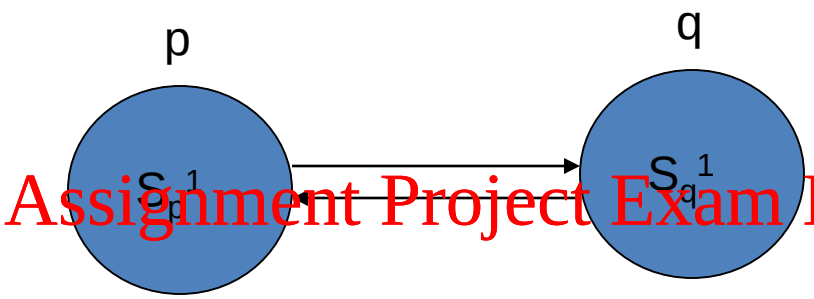
- All feasible states are consistent

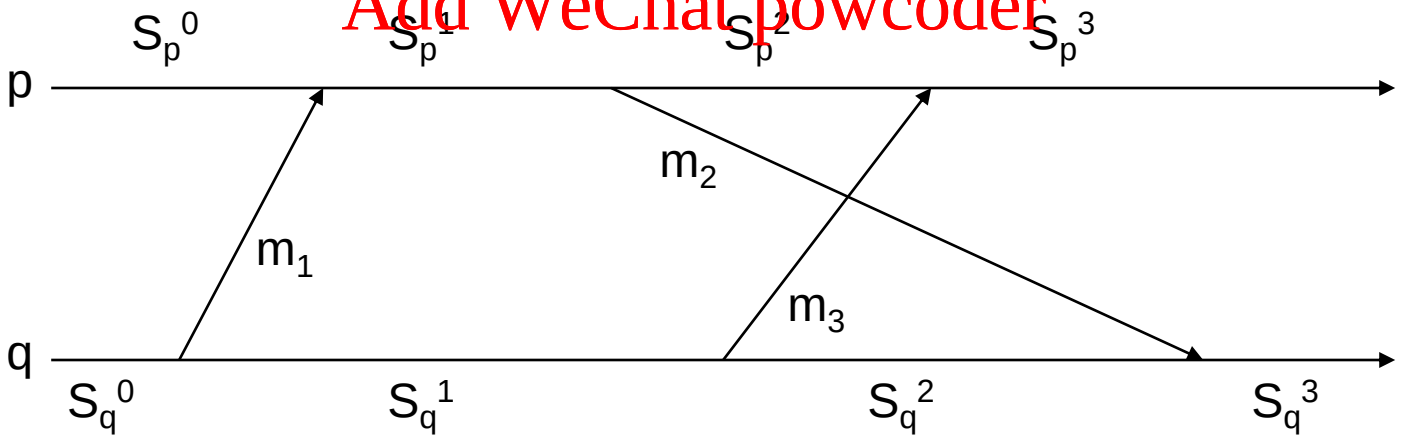
An Example



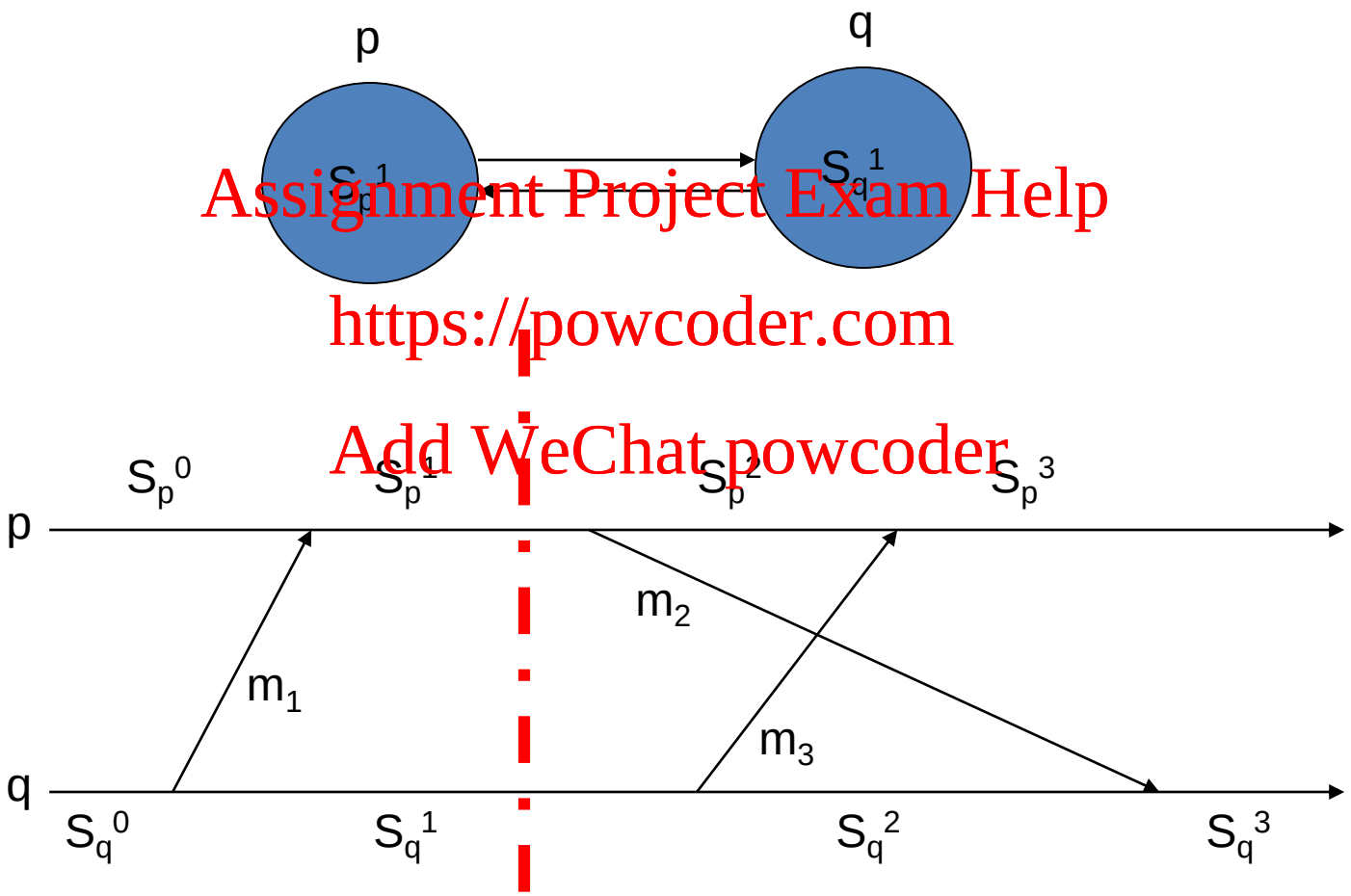


A Consistent State?

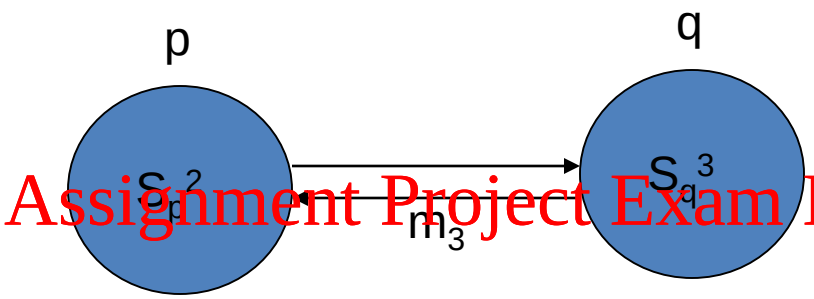


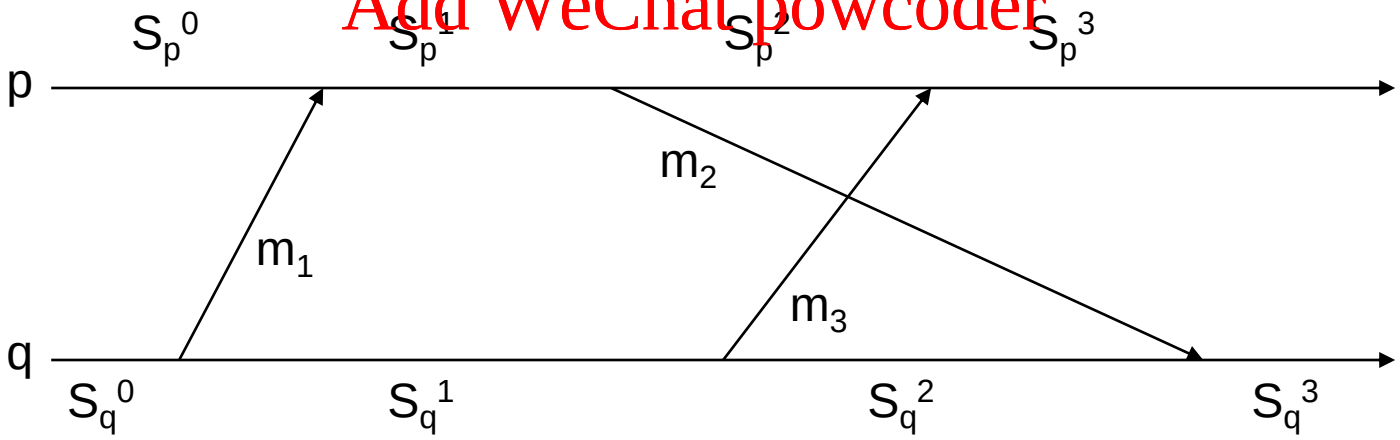


Yes

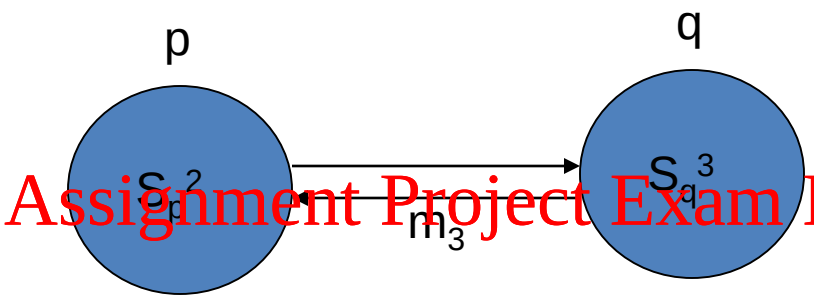


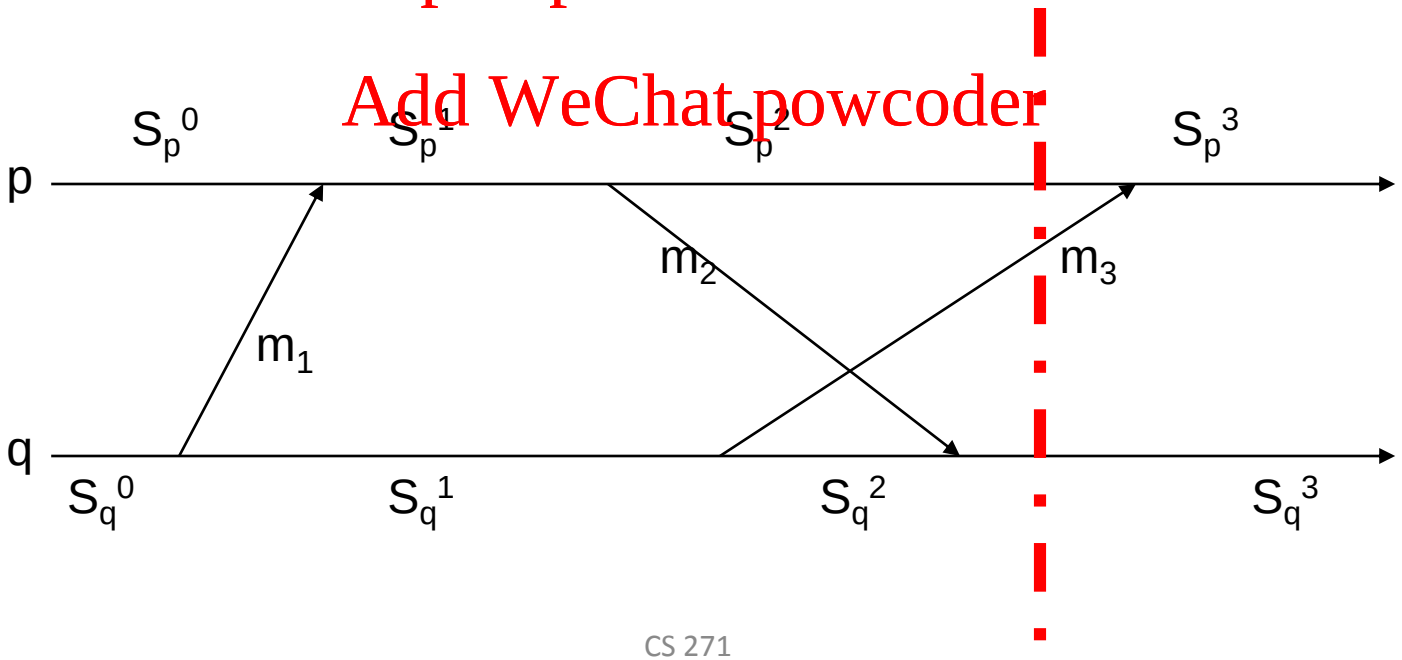
A Consistent State?



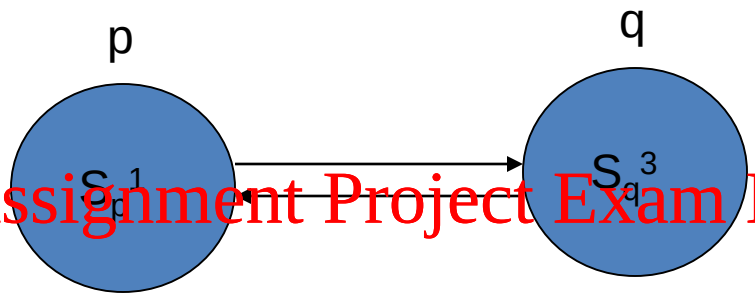


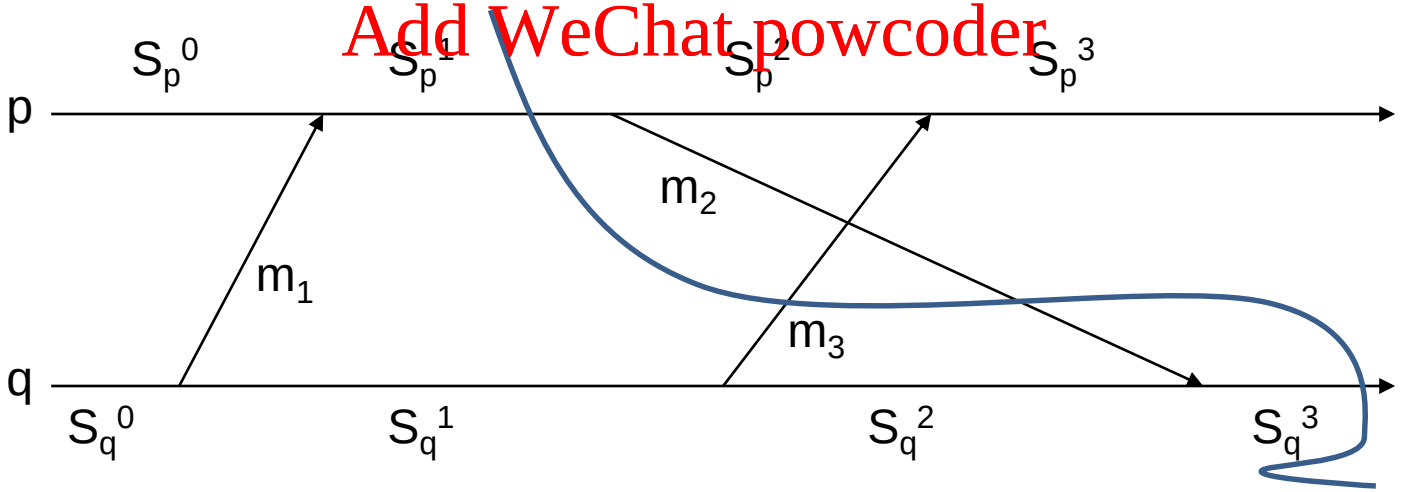
Yes



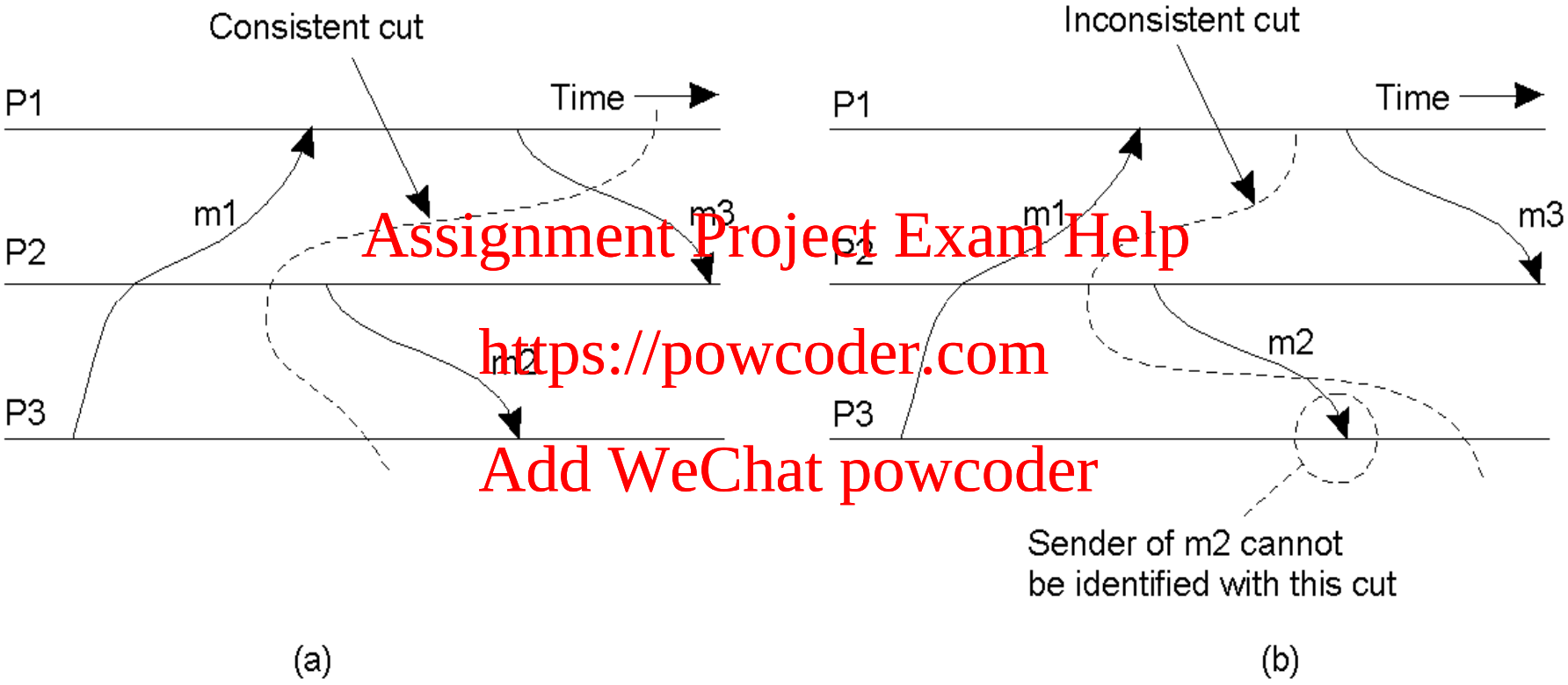


What about……





Consistent State



a) A consistent cut

b) An inconsistent cut

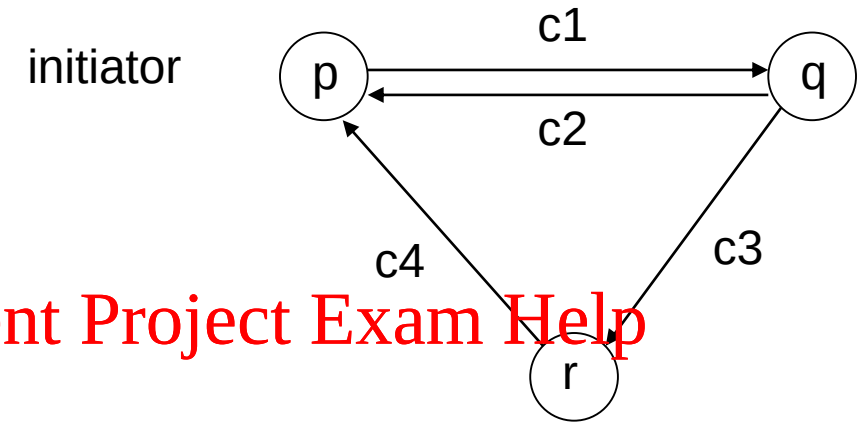
Distributed Snapshot Algorithm

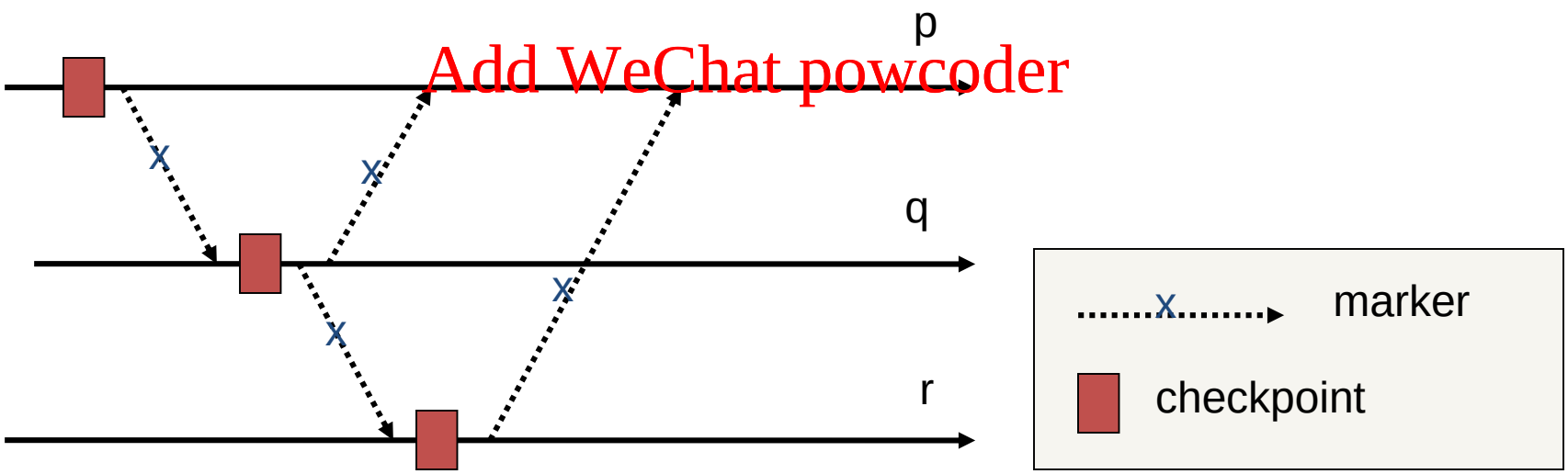
- Any process can initiate the algorithm
  - Save local state
  - Send MARKERs on every outgoing channel
- On receiving a first marker on a channel c:
  - Process saves local state and state of c is empty
  - Send MARKERs on all outgoing channels, and save messages on all other incoming channels (not c).
- On receiving subsequent marker on a channel:
  - stop saving messages for that channel
  - Saved messages are the state of the channel

Distributed Snapshot

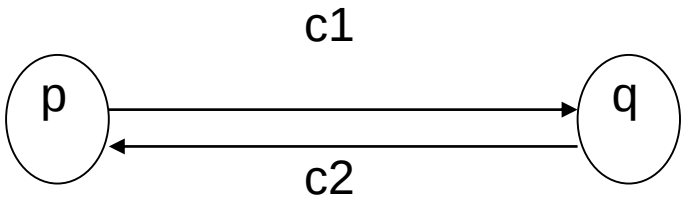
- A process finishes when
  - It receives a marker on each incoming channel and processes them all
  - State: local state plus state of all channels
  - Send state to initiator
- Any process can initiate snapshot
  - Multiple snapshots may be in progress
    - Each is distinguished by tagging the marker with the initiator ID (and sequence number)

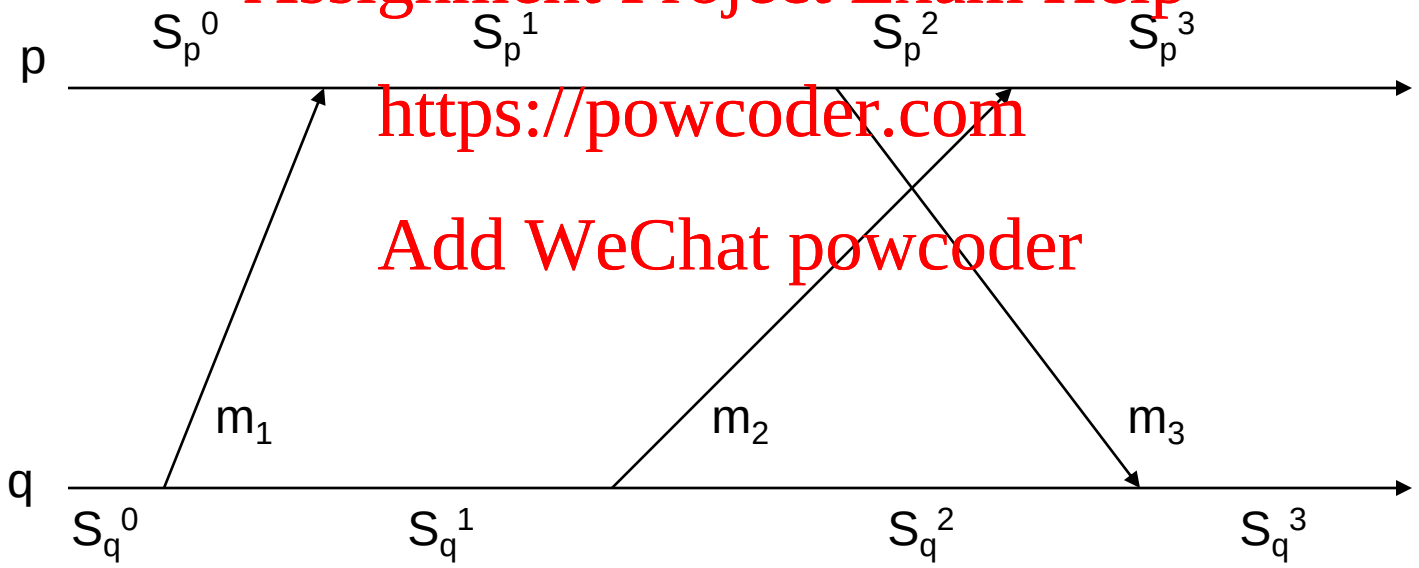
Example



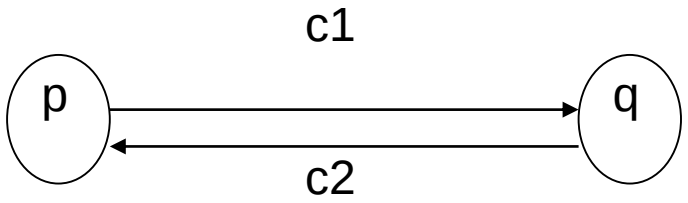


Execution Example

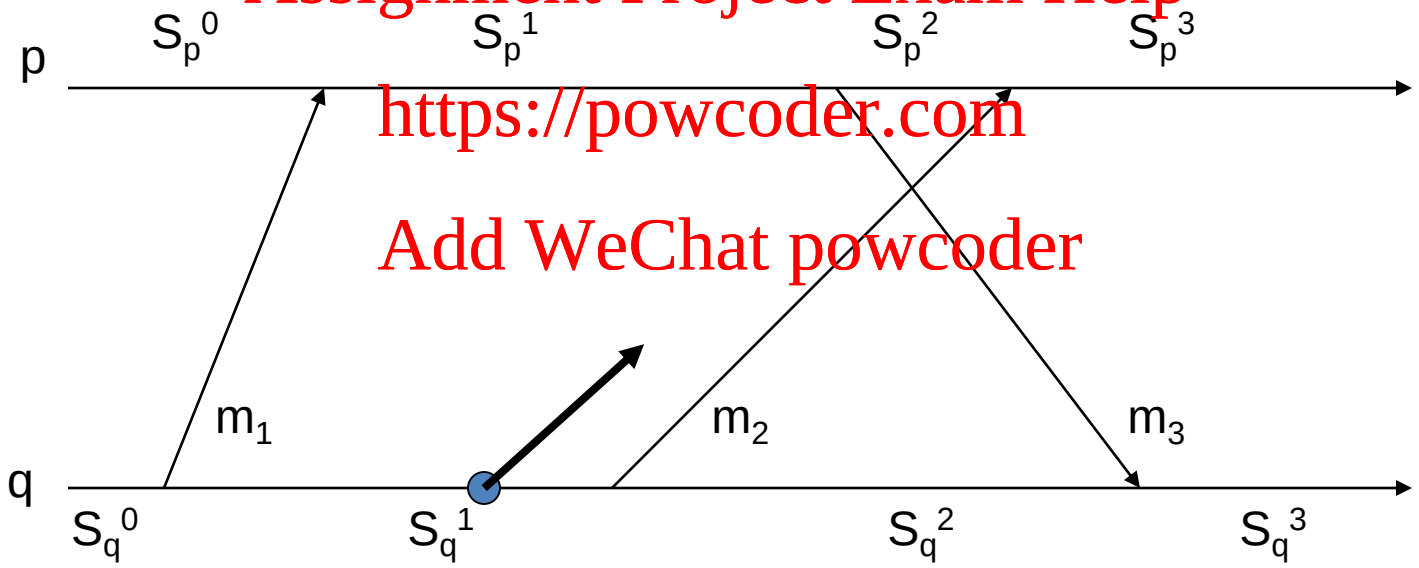




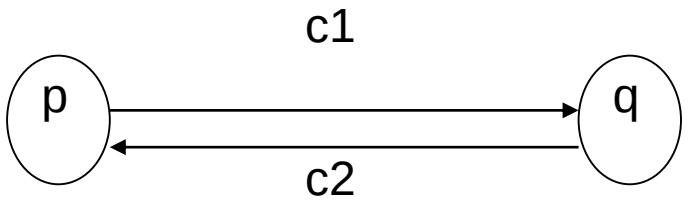
Execution Example



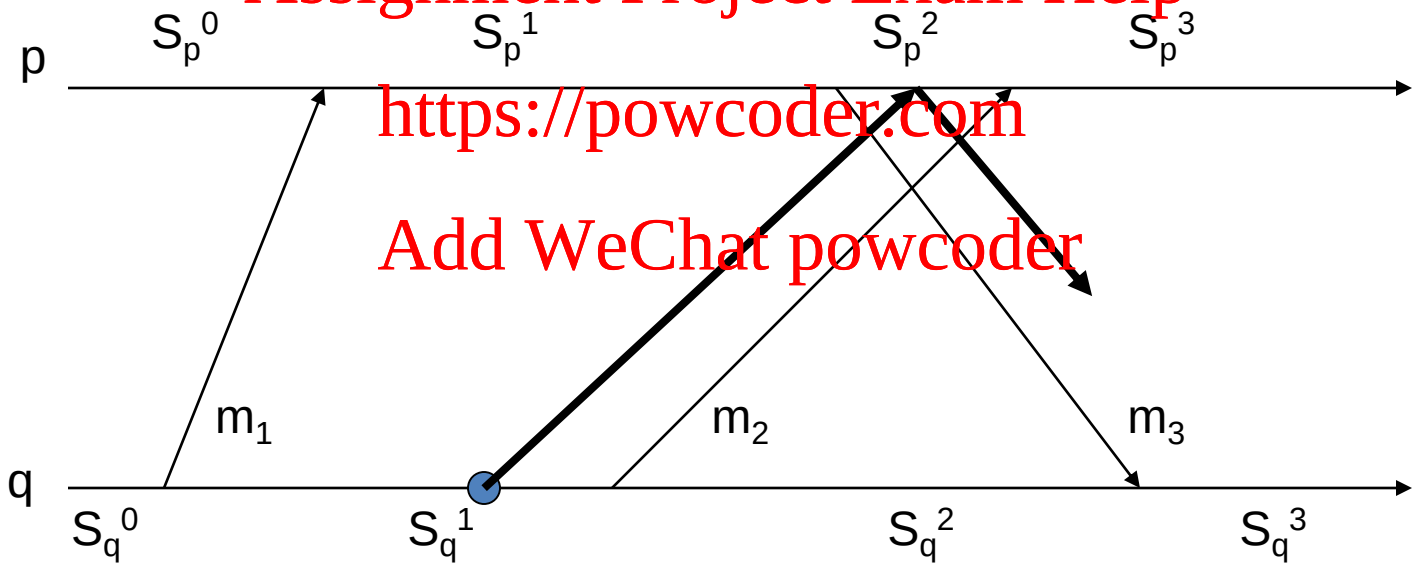
q records state as S_q^1 , sends marker to p



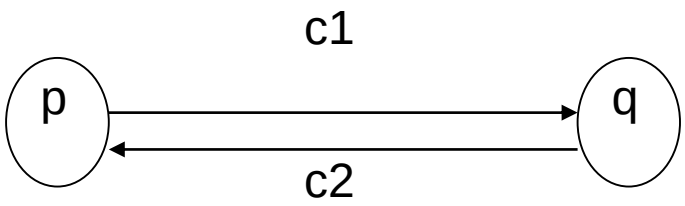
Execution Example



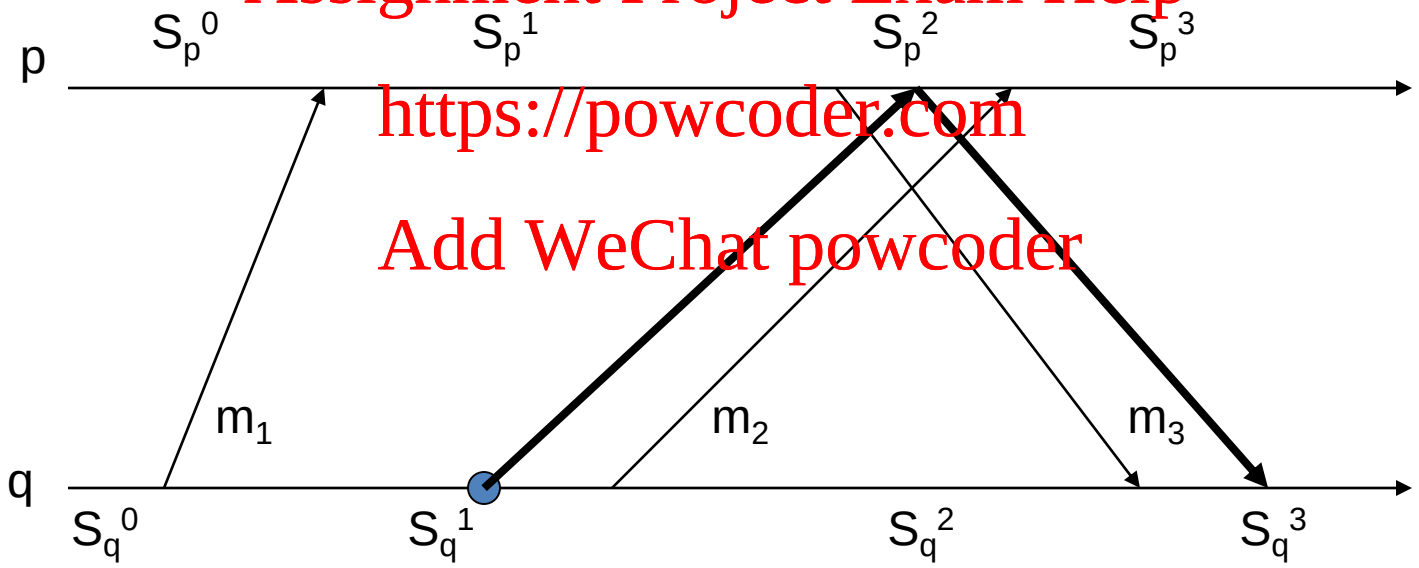
p records state as S_p^2, channel state as empty



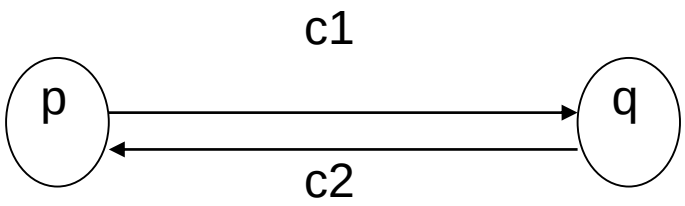
Execution Example



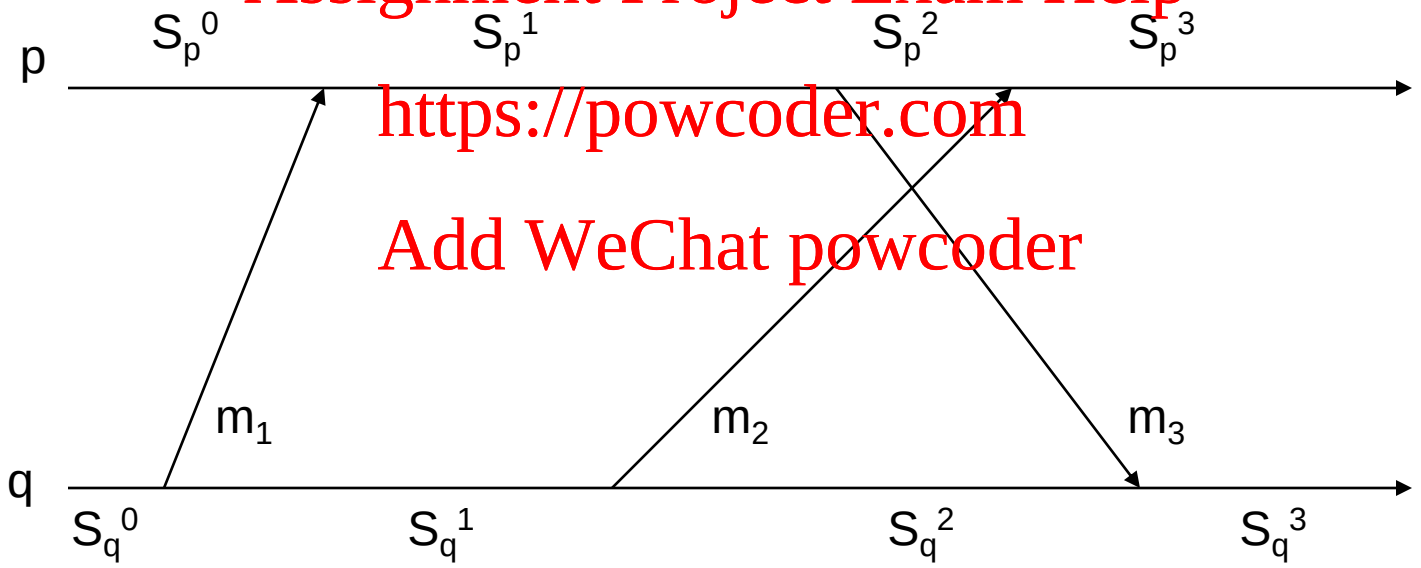
q records channel state as m_3



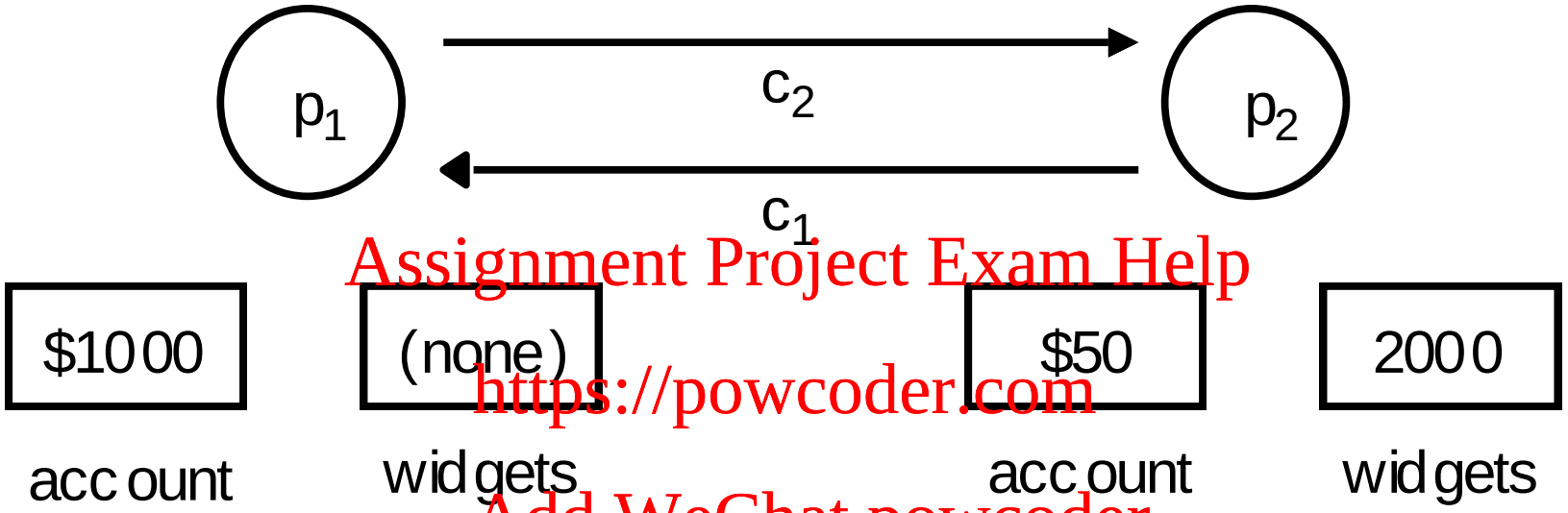
Execution Example



Recorded Global State = $((S_p^2, S_q^1), (0,m_3))$



Two processes trading in "widgets"



- Process p_1 sends orders for widgets over c_2 to p_2, enclosing payment at the rate of $10 per widget.
- In exchange, process p_2 sends widgets along channel c_1 to p_1.

Execution Example

1. p_1 records its state and sends a marker over c_2.

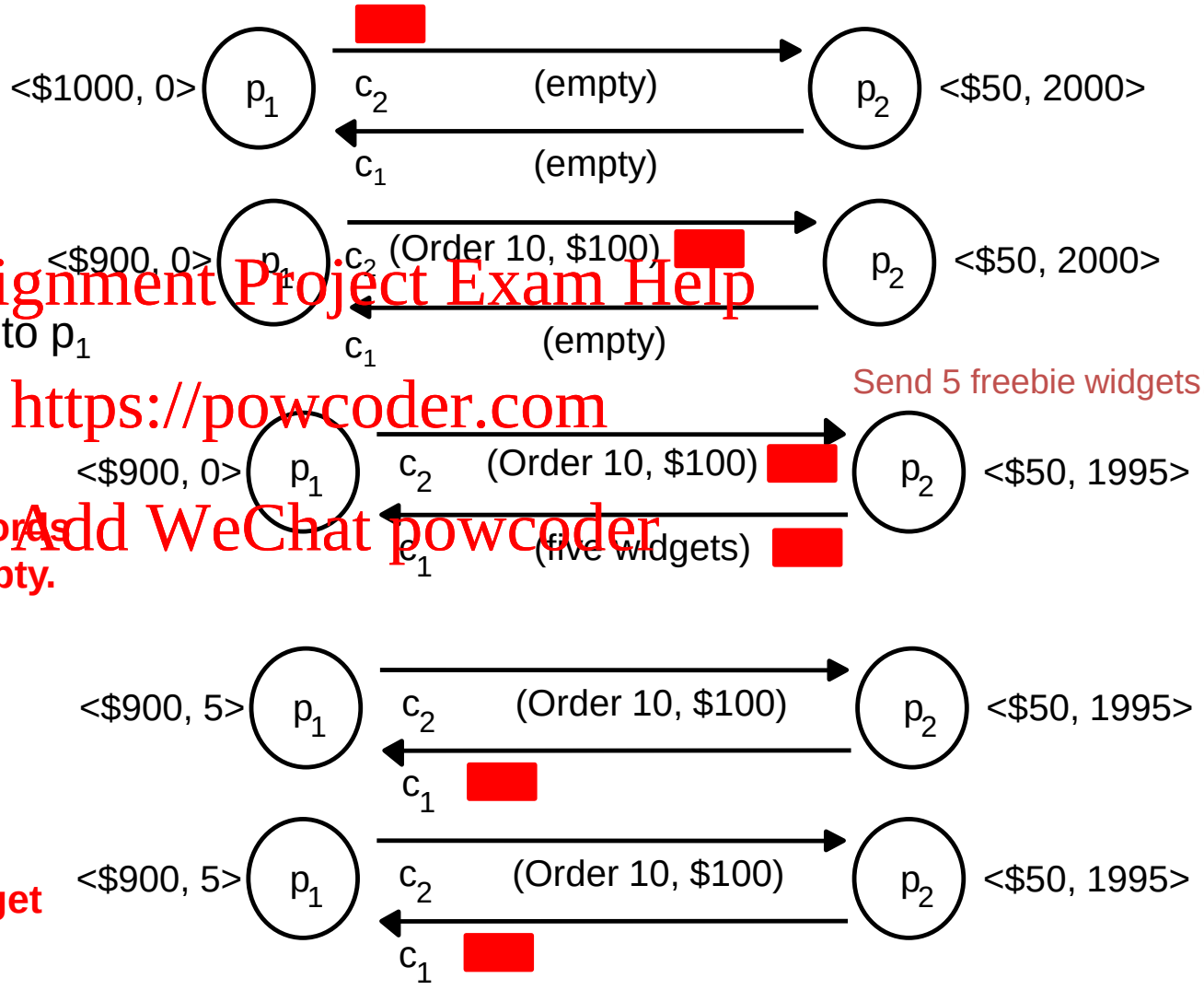
e_0: p_1 sends (10, $100).

e_1: p_2 sends 5 free widgets to p_1

2. p_2 receives marker & records its state & channel c_2 as empty. Then sends marker over c_1

e_2: p_1 receives five widgets

3. p_1 receives the marker, & records state of c_1 as 5 widget



The Retrieved State

What is this State? Does it correspond to any of the actual global states the system went through?

The Execution of the Algorithm

e_0: p_1 sends (10, \$100).

e_1: p_2 sends 5 widges to p_1

e_2: p_1 receives five widgets

The Execution of the Algorithm

e_0: p_1 sends (10, \$100).

e_1: p_2 sends 5 widges to p_1

e_2: p_1 receives five widgets

e_1: p_2 sends 5 widges to p_1

e_0: p_1 sends (10, \$100)

e_2: p_1 receives five widgets

The Execution of the Algorithm

e_0: p_1 sends (10, \$100).

e_1: p_2 sends 5 widges to p_1

e_2: p_1 receives five widgets

e_1: p_2 sends 5 widges to p_1

TAKE SNAPSHOT

e_0: p_1 sends (10, \$100)

e_2: p_1 receives five widgets

Model

- Finite set of processes. Finite set of directed channels. Modeled as a Directed Graph.
- Channels are FIFO, error free.
- A process is a finite set of states: an initial state and a set of events.
- State of channel is msgs sent but not received.

Correctness Proof

- Global State: set of process and channel states.
- Let seq = $e_1, e_2, \ldots, e_n$ be a dist comp.
- Let S_i be the state before event e_i.
- Let S_{init} be the state in which the algorithm started and S_{final} the state when it terminated.
- S_{snap} be the recorded state.
- Show that S_{snap} is reachable from S_{init} and S_{final} is reachable from S_{snap}

Correctness Proof

- There is a sequence seq' such that:

1. seq' is a permutation of seq
2. $S_{init} = S_{snap}$ or S_{init} occurs earlier than S_{snap}
3. $S_{final} = S_{snap}$ or S_{snap} occurs earlier than S_{final}

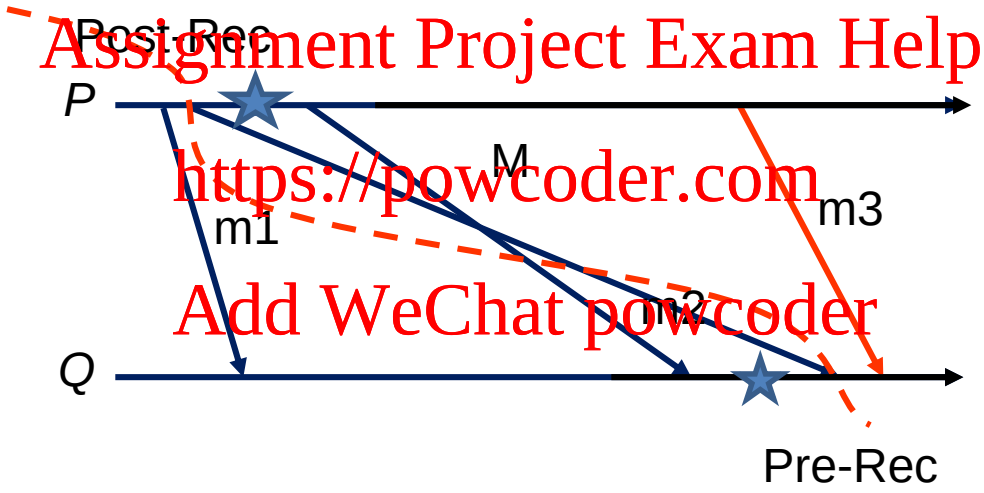
- e is a pre-recording event iff e is in p and p records state after e in seq.
- E is post-recording event iff e is in p and p records state before e in seq.

Correctness Proof

1. All events e_i where i < init are pre-recording.
2. All events e_i where i > final are post-recording.
3. There can be some post-recording events before some pre-recording events.

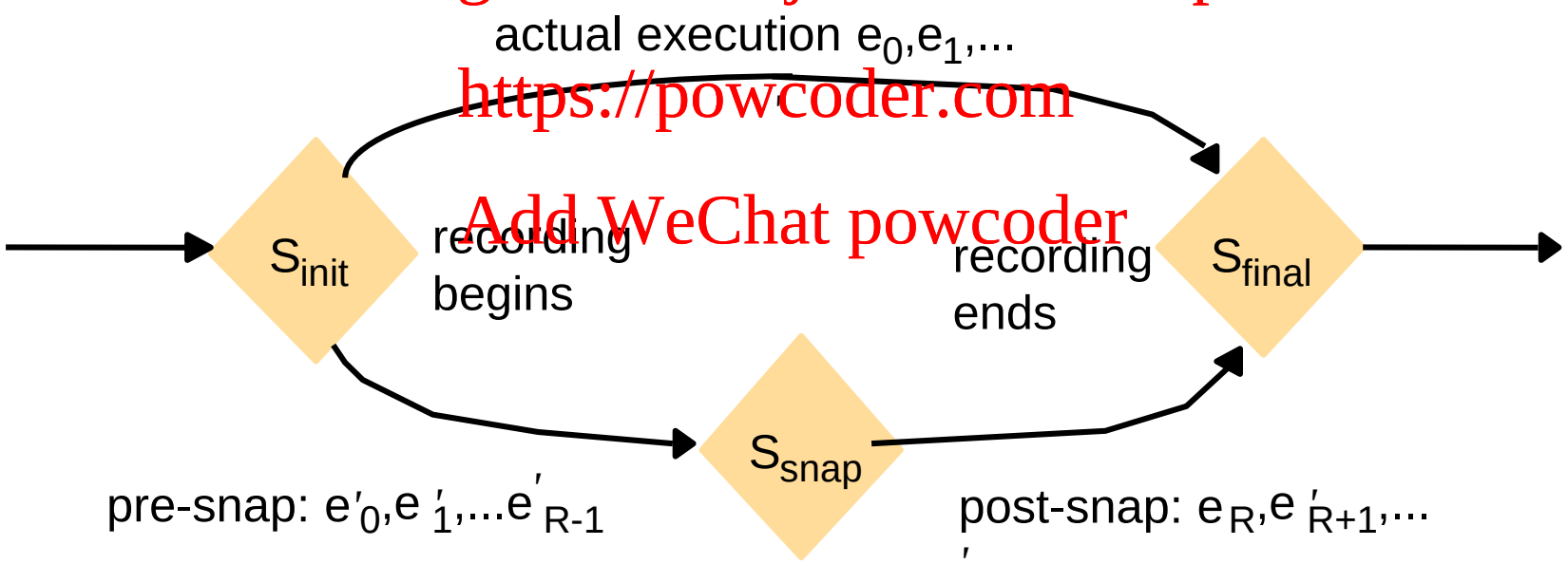
- Possible on same process? NO
- What about on different processes?
- *A pair of events a, b can be scheduled in any order if there is no causal order between them, so (a; b) is equivalent to (b; a)*

Checkpoint Proof: Different Processes Case



- **Violates FIFO property**

Reachability between states in the snapshot algorithm



Why does it work?

Let an observer observe the following actions:

pre[i] pre[k] post[k] pre[j] post[i] pre[l] post[j] post[l] …

≡

pre[i] pre[k] pre[j] post[k] post[i] pre[l] post[j] post[l] …

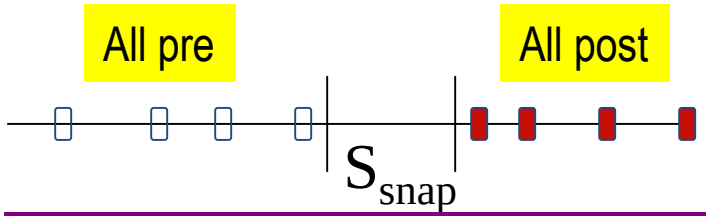
≡

pre[i] pre[k] pre[j] post[k] pre[l] post[i] post[j] post[l] …

≡

pre[i] pre[k] pre[j] pre[l] post[k] post[i] post[j] post[l] …

S_{snap}



Easy conceptualization of the snapshot state

Returns a correct global state

- Obtain seq′ by reordering events of seq between first snap and last snap, putting all pre-recording events before all post-recording events, preserving causality.

- Returned state is exactly the global state of seq′ between the pre-recording and post-recording events.

Correctness

- Termination:
  - Communication graph is strongly connected
  - All snap eventually, because of either snap input or marker message.
  - If there is a communication path from p_i to p_k, then p_k will record its state a finite period of time after p_i
  - Markers eventually sent and received on all channels.

What about Channel States?

- Recorded state of channel C from p to q:
  - Sequence of msgs received by q before Marker received

Minus

  - Sequence of msgs received by q before state recording

What about Channel States?

- What we want:
  - Sequence of msgs sent by p before state recording

Minus

  - Sequence of msgs received by q before state recording

What about Channel States?

- Recorded state of channel C from p to q:
  - Sequence of msgs received by q before Marker received

  Minus
  - Sequence of msgs received by q before state recording
- What we want:
  - Sequence of msgs sent by p before state recording

  Minus
  - Sequence of msgs received by q before state recording

What about Channel States?

- Recorded state of channel C from p to q:
  - Sequence of msgs received by q before Marker received

  Minus
  - Sequence of msgs received by q before state recording
- What we want:
  - Sequence of msgs sent by p before state recording

  Minus
  - Sequence of msgs received by q before state recording

What about Channel States?

- Recorded state of channel C from p to q:
  - Sequence of msgs received by q before Marker received

  Minus
  - Sequence of msgs received by q before state recording
- What we want:
  - Sequence of msgs sent by p before state recording

  Minus
  - Sequence of msgs received by q before state recording
- They are equal!

Detecting Stable Properties

- A predicate y (S) is a stable property if once y is true for state S it remains true (unless you interfere with system) for all subsequent states.
- Run protocol and record state S*.
  - If y(S_{init}) is true then y(S*) is true
  - If y(S*) is true then y (S_{final}) is true
- y(S*) true implies property holds
- y(S*) false does NOT implies property does not hold.