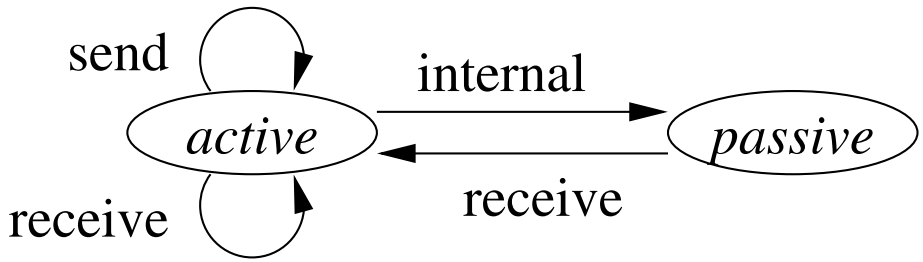
Termination detection

The *basic* algorithm is terminated if (1) each process is passive, and (2) no basic messages are in transit.



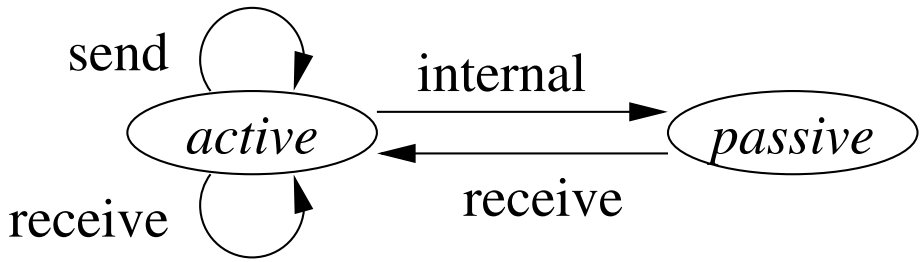
The *control* algorithm concerns termination detection and announcement.

Announcement is simple; we focus on detection.

Termination detection shouldn't influence basic computations.

Termination detection

The *basic* algorithm is terminated if (1) each process is passive, and (2) no basic messages are in transit.



The *control* algorithm concerns termination detection and announcement.

Announcement is simple; we focus on detection.

Termination detection shouldn't influence basic computations.

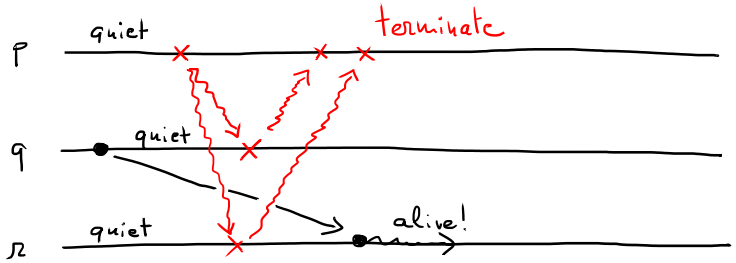
Question

Why is the following termination detection algorithm not correct ?

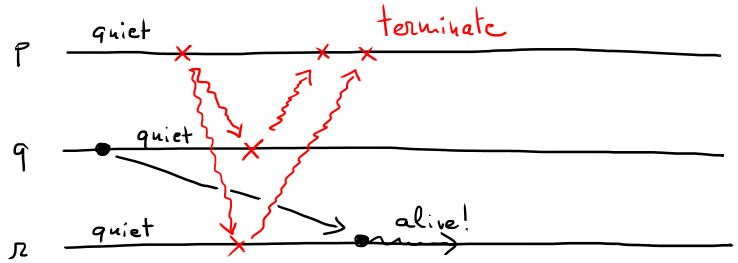
- If a process becomes quiet

  then it starts a wave (tagged with its id).
- Only quiet processes take part in the wave.
- If the wave completes, its initiator calls *Announce*.

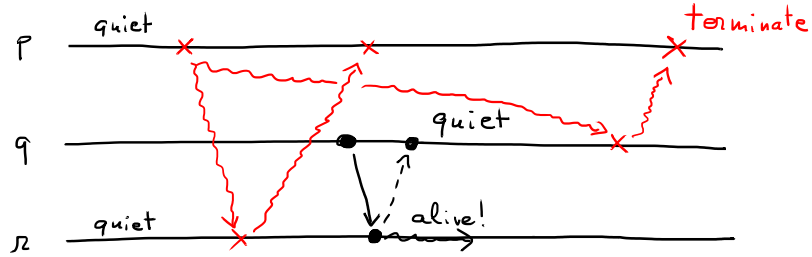
wave-like termination v.1 (no ack)



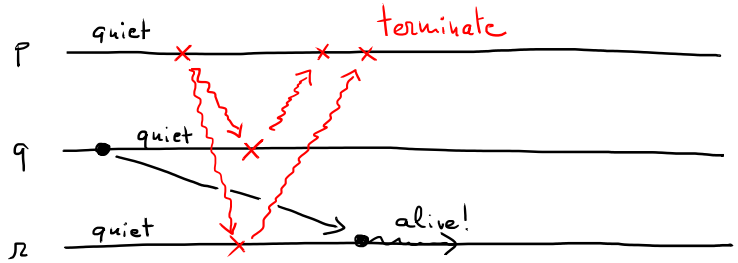
wave-like termination v.1 (no ack)



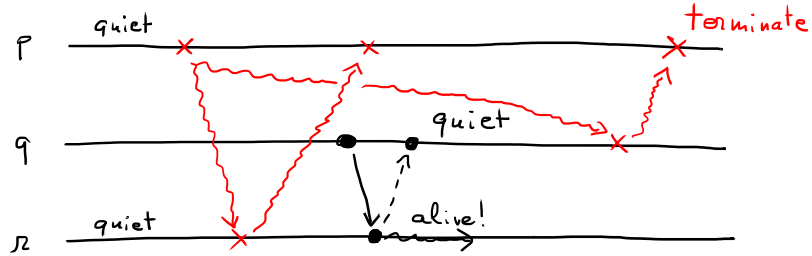
wave-like termination v.2 (with ack)



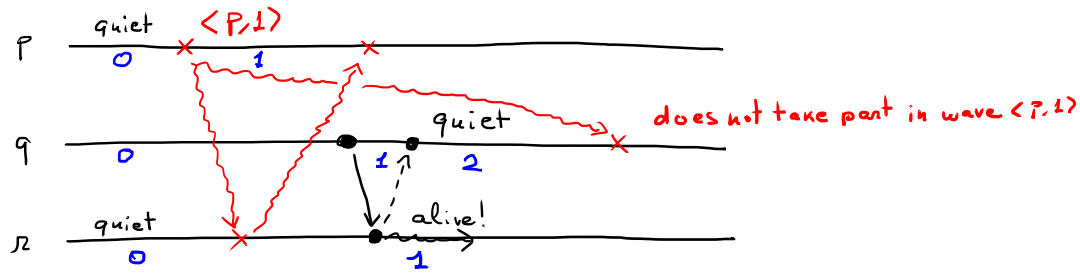
wave-like termination v.1 (no ack)



wave-like termination v.2 (with ack)



wave-like termination v.3 (with ack + timestamp) = Rana's algorithm (decentralized)



Dijkstra-Scholten algorithm

Requires a centralized basic algorithm, and an undirected network.

A *tree* T is maintained, which has the initiator p_0 as the root, and includes all active processes. *Initially*, T consists of p_0.

cc_p estimates (from above) the number of children of process p in T.

- When p sends a basic message, $cc_p \leftarrow cc_p + 1$.
- Let this message be received by q.
  - If q isn't yet in T, it joins T with parent p and $cc_q \leftarrow 0$.
  - If q is already in T, it sends a control message to p that it isn't a new child of p. Upon receipt of this message, $cc_p \leftarrow cc_p - 1$.
- When a noninitiator p is passive and $cc_p = 0$, it quits T and informs its parent that it is no longer a child.
- When the initiator p_0 is passive and $cc_{p_0} = 0$, it calls *Announce*.

Dijkstra-Scholten algorithm

Requires a centralized basic algorithm, and an undirected network.

A *tree* T is maintained, which has the initiator p_0 as the root, and includes all active processes. *Initially*, T consists of p_0.

cc_p estimates (from above) the number of children of process p in T.

- When p sends a basic message, $cc_p \leftarrow cc_p + 1$.
- Let this message be received by q.
  - If q isn't yet in T, it joins T with parent p and $cc_q \leftarrow 0$.
  - If q is already in T, it sends a control message to p that it isn't a new child of p. Upon receipt of this message, $cc_p \leftarrow cc_p - 1$.
- When a noninitiator p is passive and $cc_p = 0$, it quits T and informs its parent that it is no longer a child.
- When the initiator p_0 is passive and $cc_{p_0} = 0$, it calls *Announce*.

Dijkstra-Scholten algorithm

Requires a centralized basic algorithm, and an undirected network.

A *tree* T is maintained, which has the initiator p_0 as the root, and includes all active processes. *Initially*, T consists of p_0.

cc_p estimates (from above) the number of children of process p in T.

- When p sends a basic message, $cc_p \leftarrow cc_p + 1$.
- Let this message be received by q.
  - If q isn't yet in T, it joins T with parent p and $cc_q \leftarrow 0$.
  - If q is already in T, it sends a control message to p that it isn't a new child of p. Upon receipt of this message, $cc_p \leftarrow cc_p - 1$.
- When a noninitiator p is passive and $cc_p = 0$, it quits T and informs its parent that it is no longer a child.
- When the initiator p_0 is passive and $cc_{p_0} = 0$, it calls *Announce*.

Dijkstra-Scholten algorithm

Requires a centralized basic algorithm, and an undirected network.

A *tree* T is maintained, which has the initiator p_0 as the root, and includes all active processes. *Initially*, T consists of p_0.

cc_p estimates (from above) the number of children of process p in T.

- When p sends a basic message, $cc_p \leftarrow cc_p + 1$.
- Let this message be received by q.
  - If q isn't yet in T, it joins T with parent p and $cc_q \leftarrow 0$.
  - If q is already in T, it sends a control message to p that it isn't a new child of p. Upon receipt of this message, $cc_p \leftarrow cc_p - 1$.
- When a noninitiator p is passive and $cc_p = 0$, it quits T and informs its parent that it is no longer a child.
- When the initiator p_0 is passive and $cc_{p_0} = 0$, it calls *Announce*.

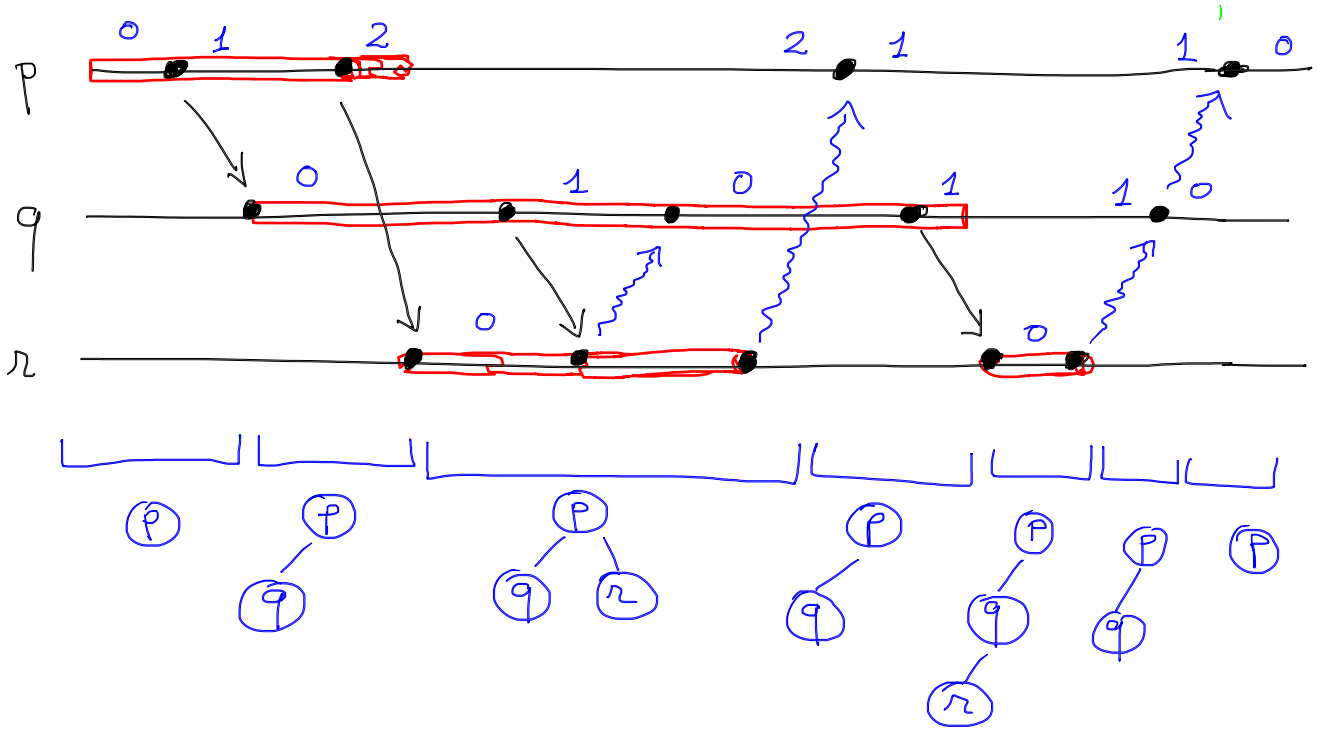
Dijkstra-Scholten algorithm

Requires a centralized basic algorithm, and an undirected network.

A *tree* T is maintained, which has the initiator p_0 as the root, and includes all active processes. *Initially*, T consists of p_0.

cc_p estimates (from above) the number of children of process p in T.

- When p sends a basic message, $cc_p \leftarrow cc_p + 1$.
- Let this message be received by q.
  - If q isn't yet in T, it joins T with parent p and $cc_q \leftarrow 0$.
  - If q is already in T, it sends a control message to p that it isn't a new child of p. Upon receipt of this message, $cc_p \leftarrow cc_p - 1$.
- When a noninitiator p is passive and $cc_p = 0$, it quits T and informs its parent that it is no longer a child.
- When the initiator p_0 is passive and $cc_{p_0} = 0$, it calls *Announce*.


p
q
r
0
1
2
2
1
1
0
0
1
0
1
1
0
0
0
p
q
r
p
q
p
q
r
p
q
p
q
r
p
q
p
p

Question

What is a drawback of the Dijkstra-Scholten as well as Rana's algorithm ?

Question

What is a drawback of the Dijkstra-Scholten as well as Rana's algorithm ?

Answer: Requires one control message for every basic message.

Weight-throwing termination detection

Requires a centralized basic algorithm;
allows a directed network.

The initiator has weight 1, all noninitiators have weight 0.

When a process *sends* a basic message, it transfers part of its weight to this message.

When a process *receives* a basic message, it adds the weight of this message to its own weight.

When a noninitiator becomes passive, it returns its weight to the initiator.

When the initiator becomes passive, and has regained weight 1, it calls *Announce*.

Weight-throwing termination detection

Requires a centralized basic algorithm;
allows a directed network.

The initiator has weight 1, all noninitiators have weight 0.

When a process *sends* a basic message, it transfers part of its weight to this message.

When a process *receives* a basic message, it adds the weight of this message to its own weight.

When a noninitiator becomes passive, it returns its weight to the initiator.

When the initiator becomes passive, and has regained weight 1, it calls *Announce*.

Question

What is a weakness of weight-throwing termination detection ?

Weight-throwing termination detection - Underflow

Underflow: The weight of a process can become too small to be divided further.

Weight-throwing termination detection - Underflow

Underflow: The weight of a process can become too small to be divided further.

Solution 1: The process gives itself extra weight, and informs the initiator that there is additional weight in the system.

An ack from the initiator is needed before the extra weight can be used, to avoid race conditions.

Weight-throwing termination detection - Underflow

Underflow: The weight of a process can become too small to be divided further.

Solution 1: The process gives itself extra weight, and informs the initiator that there is additional weight in the system.

An ack from the initiator is needed before the extra weight can be used, to avoid race conditions.

Solution 2: The process initiates a weight-throwing termination detection sub-call, and only returns its weight to the initiator when it has become passive and this sub-call has terminated.