Distributed Algorithms

Termination Detection

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

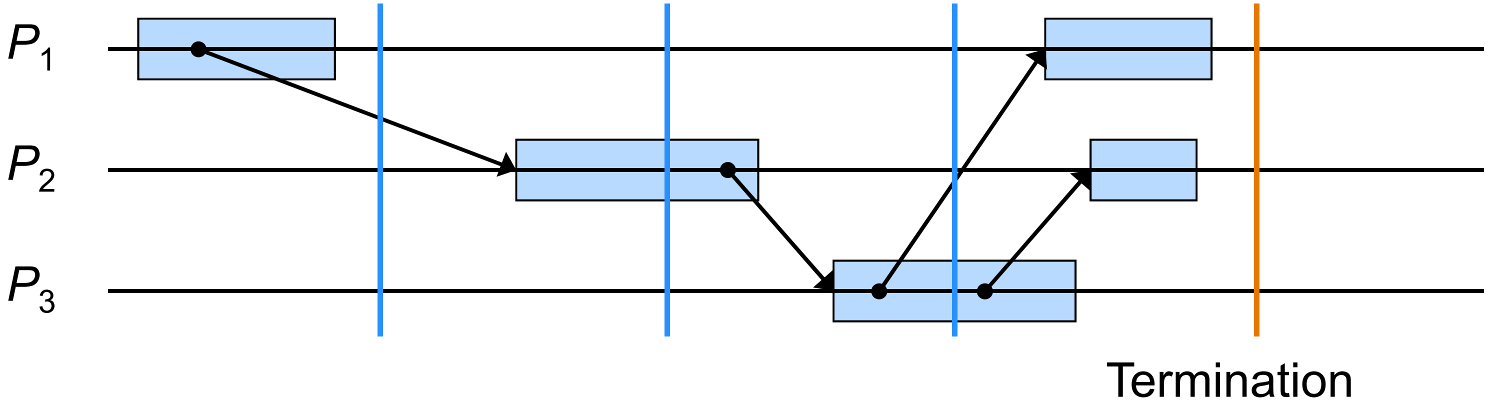
Architecture of Application Systems
Faculty for Computer Science and Electrical Engineering
University of Rostock

Overview

- Termination in different system models
- Algorithms for termination detection
  - Double Counting Algorithm
  - Time Zone Algorithm
  - Vector Algorithm
  - Credit Algorithm

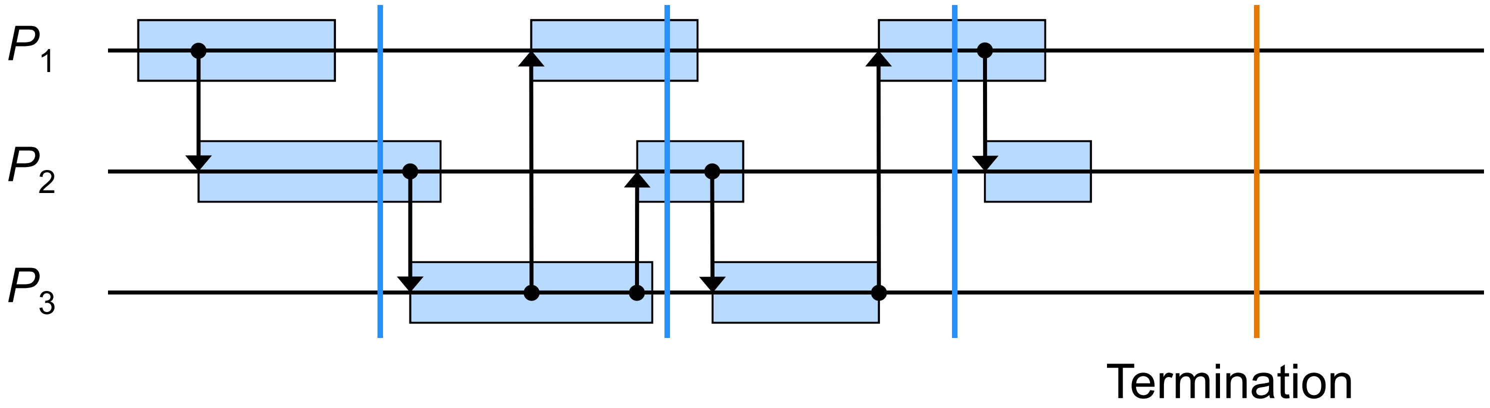
Asynchronous Model

- Processes are active or passive
- Only active processes can send basic messages
- An active process can become passive at anytime
- Passive processes can be reactivated by basic messages
- Termination detection: Determine, whether all processes are passive and no messages are in transit at a point in time



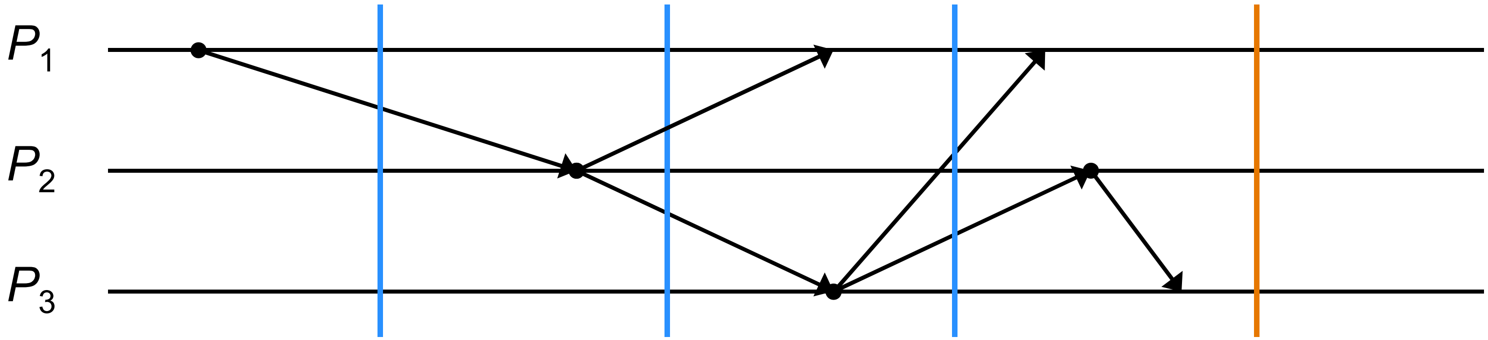
Process Model

- Difference to the asynchronous model
  - Messages have no delay
  - ⇒ Vertical arrows in space time diagram
- Termination detection
  - Determine, whether all processes are *passive* at a certain point in time



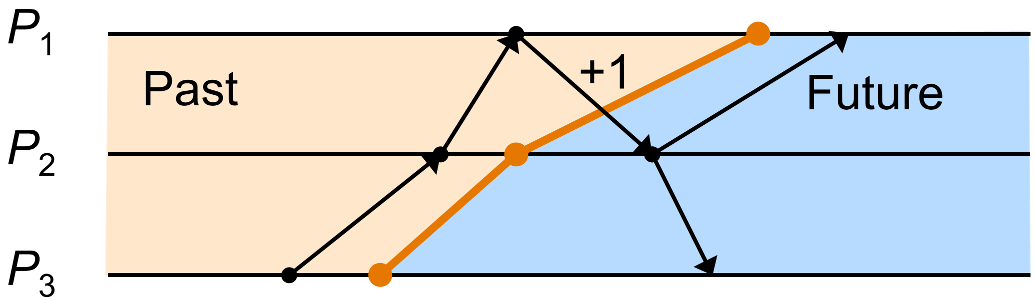
Atom Model

> Difference to the asynchronous model
  > Actions are atomic and timeless
> Termination detection
  > Determine, whether no messages are in transit at a certain point in time



Simple Counting Algorithm

> Observer visits all nodes one after the other and separately sums up the basic messages sent and received
> Diverging sums indicate that a message was sent, but has not arrived yet!
> Thus, if both sums are identical, no messages are in transit and the basic algorithm has terminated, hasn‘t it?

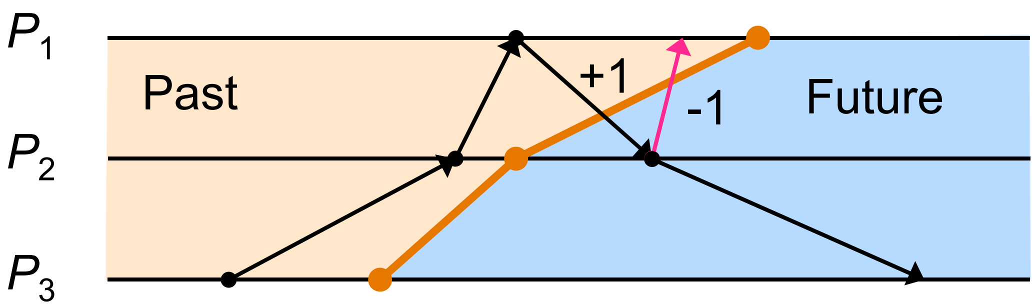


Observer visits every node and reports:
- 3 messages sent
- 2 messages received

⇒ #sent – #received = 1
⇒ no termination

Simple Counting Algorithm

- Unfortunately, this simple algorithm does not work
- The condition that the counters are equal is only necessary, but not sufficient for termination
- Counterexample shown in figure
  - observer reports 3 messages received and 3 messages sent
  - the message that was sent in the „future“, but received in the „past“, balanced the difference → phantom termination



Observer visits each node and reports:

- 3 messages received
- 3 messages sent

⇒ #sent – #received = 0

⇒ Termination (false)

Potential Solutions

1. Freeze system
2. Subsequent check ⇒ double counting algorithm
3. Detecting or avoiding inconsistent time cuts through logical time stamps ⇒ time zone algorithm
4. Differentiated counting ⇒ vector algorithm

Freezing the System

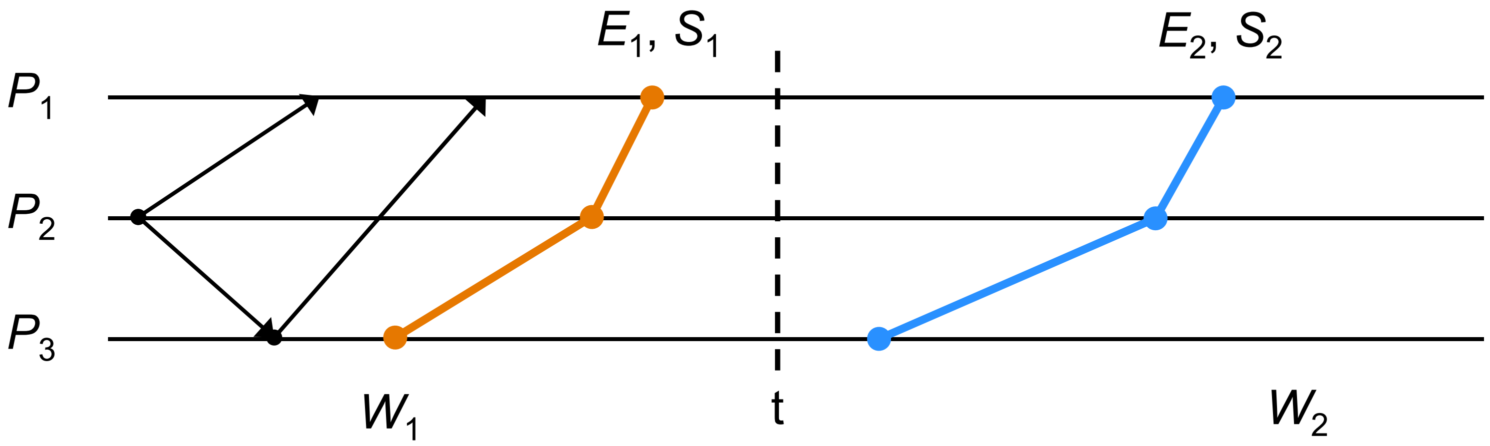
- First wave freezes the system
  - No process accepts further messages
  - Messages arriving in the meantime are buffered and regarded as being still in transit
  - ⇒ No new messages are sent in the frozen system
- Second wave sums up the messages sent and received
  - If both sums are equal, the algorithm has terminated
- Third wave unfreezes the system again
- Obvious disadvantage of the algorithm is a massively decreased concurrency

Double Counting Algorithm

- Observer visits all nodes twice and determines both times the sums of the messages received and sent
- If all four sums are equal, the basic algorithm has terminated:

  $$E_1 = S_1 = E_2 = S_2$$

- If termination was not detected, use second wave of previous round as the first wave of new round



Double Counting Algorithm

- Number of control rounds is a priori not limited by the number of basic messages
  - There may be a very slow basic message; while it is in transit arbitrarily many control rounds may be started
  - Solution: a process receiving a basic message without sending a new one starts a new round of the termination detection
- Double counting algorithm is *re-entrant*
  - Local states of visited processes are not changed
  - Thus, several concurrent initiators can test for termination

Control Topologies

- The waves for detecting termination can be realized in different ways
- Sequential Waves
  - E.g., through the construction of a logical ring and two subsequent sequential ring circuits of a token that sums up the counter readings separately for both circulations
- Parallel Waves
  - E.g., through the construction of a spanning tree and two subsequent accumulations of the counter readings, each from the leafs to the initiator
  - Achieves a better time complexity through parallel messages

Using the Echo Algorithm

- Use a single run of the echo algorithm for two accumulations of the counter readings
- The two counters (#messages sent/received) are saved
  - for the first time, when a node becomes red (forth wave) and
  - for the second time, when it becomes green (back wave)
- The four resulting sums are propagated along with the echo messages towards the initiator

- Works because a green node cannot have a white neighbor

⇒ A message from a green node cannot reach a white node

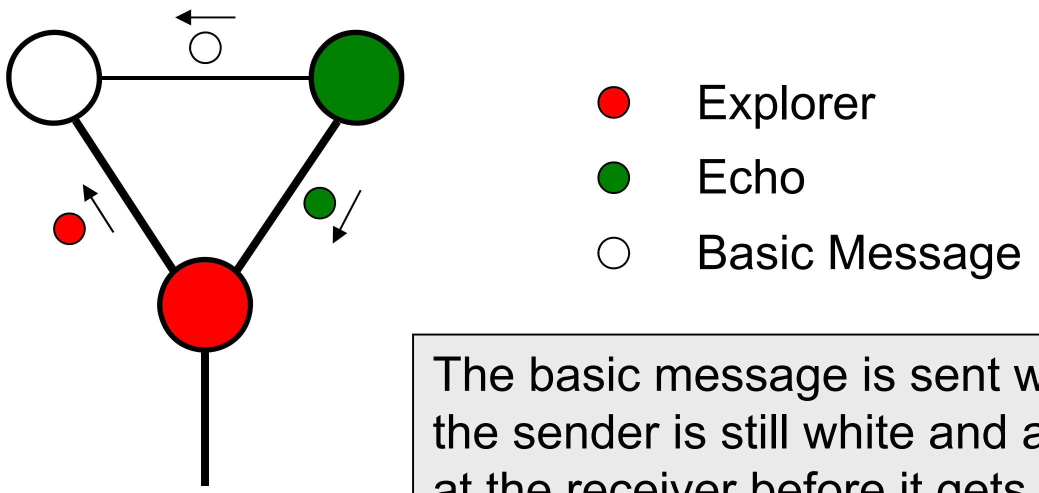
⇒ A message from “the future” cannot be received in “the past”

Using the Echo Algorithm

> With a constructed spanning tree, the usage of the forth wave and the back wave of a single run does not work!

> If a spanning tree is constructed on a topology that is not a tree, there is at least one edge that is not part of the tree

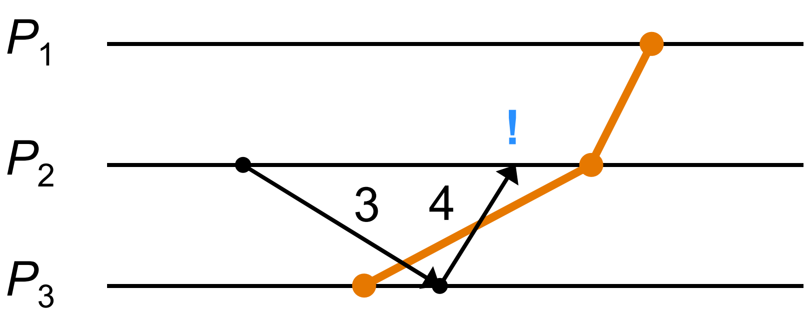
> Over this edge, a basic message can get from a green node to a white node balancing the difference of the sums



The basic message is sent while the sender is still white and arrives at the receiver before it gets red.

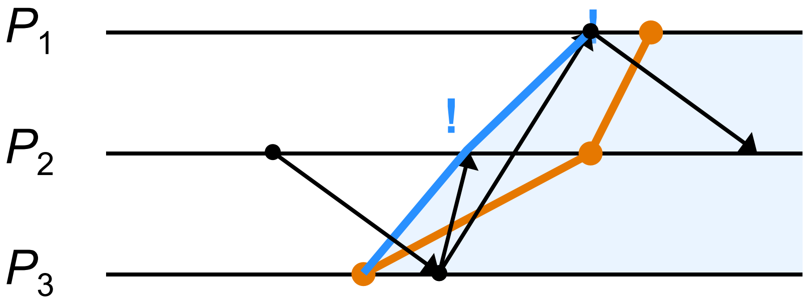
Time Zone Algorithm

- Again, an observer visits all nodes one after the other and builds a send and receive sum, respectively
- But now, the visit of the observer increments a time zone counter on the visited node
- Current value of time zone counter is attached to every basic message from the sending node
- Thus, messages from the future can be recognized setting a flag that is evaluated by the following wave and then reset
- Execute waves, until both sums are equal and the flag is *not* set



Moving Forward the Cut Line

- If a message from the future is received on a node, the counters are saved
- When the observer passes by later, the saved counters are uses instead of the current counters
- Thus, the cut line is moved backward in time such that messages from the future reside completely in the future



Time Zone Algorithm

- Recognizes inconsistent cuts
- Again, unlimited number of control rounds

- Disadvantages
  - Basic messages are affected
  - Not re-entrant because the local state of nodes is changed
  - Waves of several initiators can disturb each other

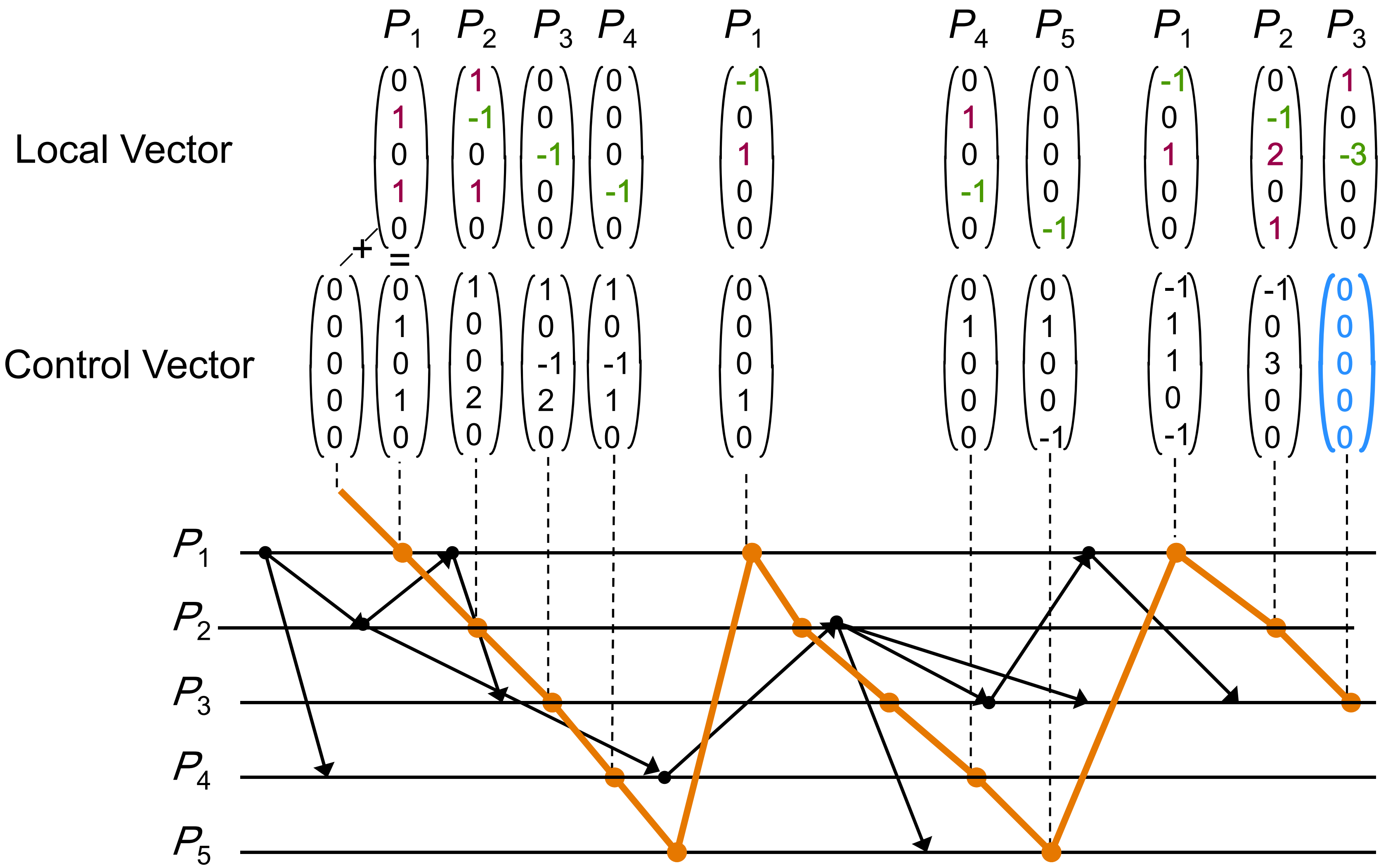
- Double counting algorithm is both more elegant and more universal

Vector Algorithm

- Each process P_A has a local vector V_A with length n that is initialized to the zero vector
- If P_A sends a basic message to P_B, $V_A[B]$ is increased by 1
  - ⇒ in the sender's vector, the receiver's component is incremented
- If P_B receives a basic message, $V_B[B]$ is decreased by 1
  - ⇒ in the receiver's vector, the receiver's component is decremented
- A control vector C (also with length n and initialized to the zero vector) visits all processes subsequently
- On a visit, the local vector V is added to C and V itself is set back to the zero vector
- Termination is detected, when the control vector becomes the zero vector again

Vector Algorithm – Example Trace

sent
received



Vector Algorithm

- Improvement
  - If the next node's component is 0, that node is skipped
  - ⇒ Avoids potentially useless visits
- Advantages
  - Independent from the net topology
  - Low number of control messages
  - Basic messages remain untouched
- Disadvantages
  - Length of the control message (vector) → $O(n)$
  - Algorithm is *not* re-entrant

Credit Algorithm

> Based on global system invariant stating that the sum of all credits is 1 all the time
> Primary process starts the distributed algorithm with credit 1
> If a process sends a message, the message receives half of the current credit of the process
> If an active process receives a message, its credit increases by the credit of the message
> If a process becomes passive, it sends its current credit to the primary process

> The following presentation of the algorithm assumes the asynchronous system model

Credit Algorithm

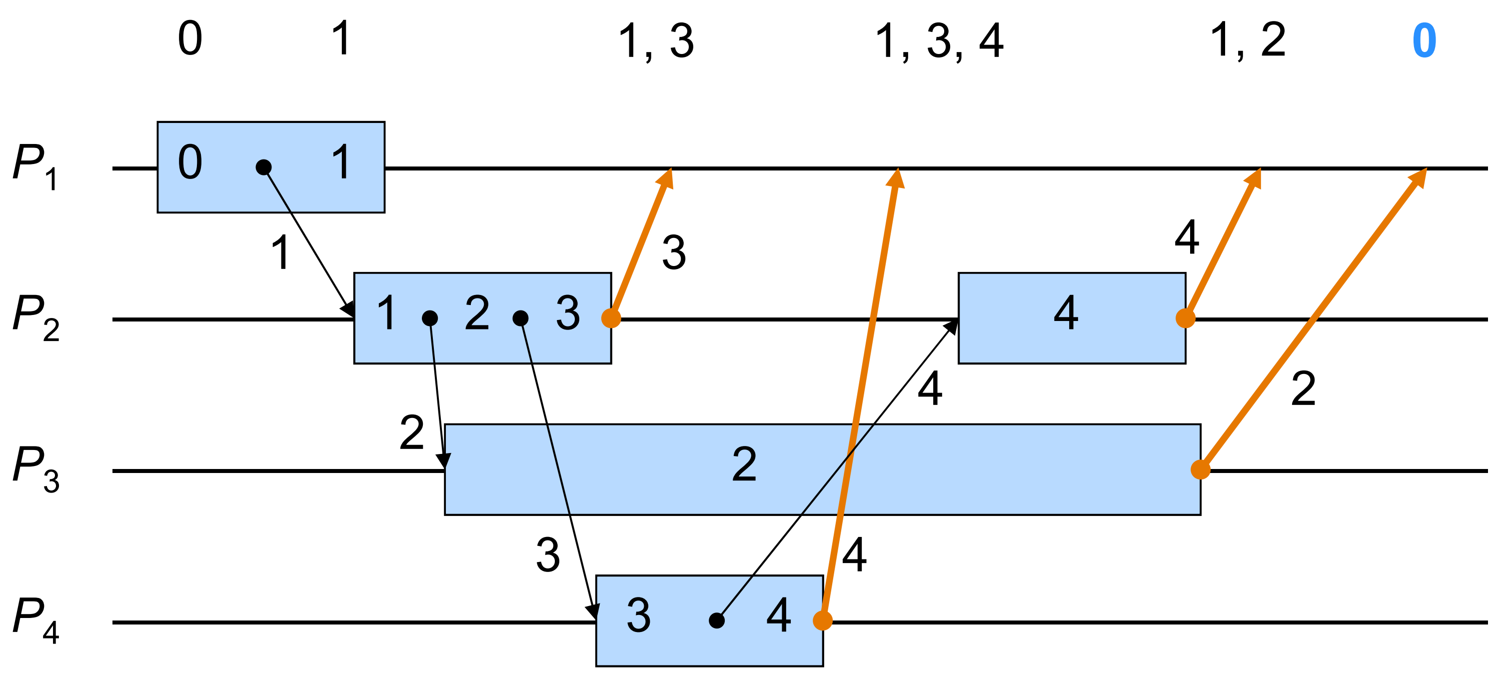
- Main invariant
  - The credit sum is always 1
- Further characteristics
  - Basic messages always carry a credit > 0 with them
  - Active processes always have a credit > 0
- Termination
  - If the primary process has credit 1 again

Representation of the Credit Portions

- Floating point numbers inconvenient
- More efficient to store the negative dual logarithm of the credit portion instead: $c = -\text{ld}\ 2^{-d}$
  - $k = 1/1 \rightarrow c = 0$
  - $k = 1/2 \rightarrow c = 1$
  - $k = 1/4 \rightarrow c = 2$
  - …
- Halving of the credit $c := c + 1$

- Bit vector for storing the credit portions
- Recombination of credit portions through binary addition

Example Trace of the Credit Algorithm



Control Message

Exemplary Exam Questions

1. Explain termination in the asynchronous system model as well as in an atom model!
2. Explain why the simple counting algorithm fails and how the double counting algorithm overcomes this problem!
3. How can the echo algorithm be used to recognize termination?
4. Describe the functionality of the vector algorithm to recognize termination!
5. What is the common disadvantage of the time zone algorithm and the vector algorithm?
6. What distinguishes the credit algorithm from the others and how does it work exactly?

Literature

1. F. Mattern. Verteilte Basisalgorithmen. Springer-Verlag, 1989. Kapitel 4: Verteilte Terminierung
2. F. Mattern. Algorithms for distributed termination detection. Distributed Computing, 2(3):161---175, 1987.

Thank you for your kind attention!

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

gero.muehl@uni-rostock.de
http://wwwava.informatik.uni-rostock.de