Termination Detection: The Model

- We require that a process is either **active** or **inactive**.
- An inactive process may not send messages.
- Whereas an active process may turn inactive, an inactive process stays inactive unless it receives a message.
- Determine whether the system can be shut down.

- Is the model sufficiently general? What, if inactive processes request work?
  - Do not consider a request to be message.
  - We stay within the model, provided an inactive process stays inactive unless receiving a message, i.e., unless receiving work.
- What is the problem?
  - Messages can turn an inactive process active, since it may be a work assignment.
  - Determine whether all processes are inactive and whether there are no more messages in the system.
The Basic Idea

- Imagine the processes to be arranged in a ring.
- Process 1 inserts a token, traveling from process $i$ to process $i + 1$ and from process $p$ back to process 1.
- The token only leaves a process if the process turns inactive.
- Process 1 determines whether the system can be shut down.

The problem: an inactive process may receive a message and turn active right after the token left.

Who triggered this message?

Get the culprits: all processes are initially colored white.

- Any process $i$ which sends a message to a process $j$ with $j < i$ is a suspect for reactivating a process: it turns black.
- If a black process receives a token, it colors the token black.
Dijkstra’s Token Termination Detection Algorithm I

(1) When process 1 turns inactive, it tries to detect termination: it turns white and sends a white token to process 2.

(2) If process \(i\) sends a message to process \(j\) and \(i > j\), then \(i\) turns black.
   // The next step guarantees that a black active process prevents // termination.

(3) Assume that process \(i > 1\) has just received the token. Process \(i\) keeps the token as long as it is active. If it turns inactive, it colors the token
   - black, if \(i\) is black,
   - otherwise \(i\) is white and the color of the token is left unchanged.
Then it passes the token along to process \(i + 1\), respectively to process 1, if \(i = p\). Afterwards process \(i\) turns white.
What may happen, if a process $i$ sends a message to a process $j$ to its right?

- Unless messages are delivered in order, it may happen that the token overtakes the message!
- Process $j$ may be white when it later receives the token and it may therefore forward a white token.
- Then process $j$ goes to sleep and becomes active again after receiving the slow message!

Assume that messages are delivered in order. If process 1 receives a white token from process $p$, then all processes are inactive.
How Good Is The Solution?

- How expensive is the token?
  - the token consumes time at most $O(p)$ due to the “handshakes” (send/receive operations) between adjacent processes on the ring.
  - Moreover, process 1 does not have to wait for the token to return, but may turn active again when receiving a message.
  - The token is nothing but an inexpensive background process which guarantees that termination is quickly recognized.

- What about “in order delivery”?
  - MPI guarantees that messages are non-overtaking: if a process sends message $M_1$ and subsequently message $M_2$ to the same process, then $M_1$ will be received before $M_2$.
  - However MPI does not guarantee that messages are delivered in-order.

- For MPI the solution is not good enough.
We equip all processes additionally with a message count.

- Initially all processes are white and the message count of process \( i \) is the difference of all messages sent by \( i \) and all messages received by \( i \).
- Whenever a process receives a message it decrements its message count and increments its count if it sends a message: we utilize that the sum of message counts is zero iff all messages have been delivered.

- We use the token to sum message counts.

But we do not have access to all message counts simultaneously.
(1) If process 1 becomes inactive, it turns white and sends a white token with its overall message count to process 2.

(2) If a process $i$ sends or receives a message, then it turns black.
   - If process 1 receives a white token, then the token has passed only white processes and neither process has sent or received a message between the last two successive visits of the token.
   - Thus message counts did not change!
   - It may however happen that messages are still in flight, but then the token’s message count will be non-zero.

(3) Assume that process $i > 1$ has just received the token. Process $i$ keeps the token as long as it is active. If it turns inactive:
   - if $i$ is black, then the token turns black. Otherwise the color of the token is unchanged.
   - Process $i$ adds its message count to the message field of the token and forwards the token to process $i + 1$, resp. to process 1, if $i = p$.
   - Afterwards process $i$ turns white.
- Assume that process 1 is white, when it receives a white token from process \( p \) and that the count field of the token is zero.
- Then all processes are inactive and there are no more messages in the system.
- Hence process 1 may send a shut down message to all processes.

Again, the token is an inexpensive background process.
No more assumptions are required.
An Alternative Solution

If the only messages are requests for work:

- Process 1 receives the entire load. Assign weight 1 to process 1 and weight 0 to the remaining processes.
- Whenever process $i$ requests and receives work from process $j$, set $w_i = w_j/2$ and $w_j = w_j/2$: only powers $2^{-k}$ change hands.
- If a process finishes its task and if the entire weight it handed out has been returned:
  - it hands its weight back to the donating process.
  - The donor adds the received weight to its current weight.
- If process 1 is done and has received weight 1, it sends a “shutdown token” into the process ring.
  - There may still be requests “in flight”.
  - The token is recycled until difference zero is determined: at this time all processes are inactive and all messages are delivered.