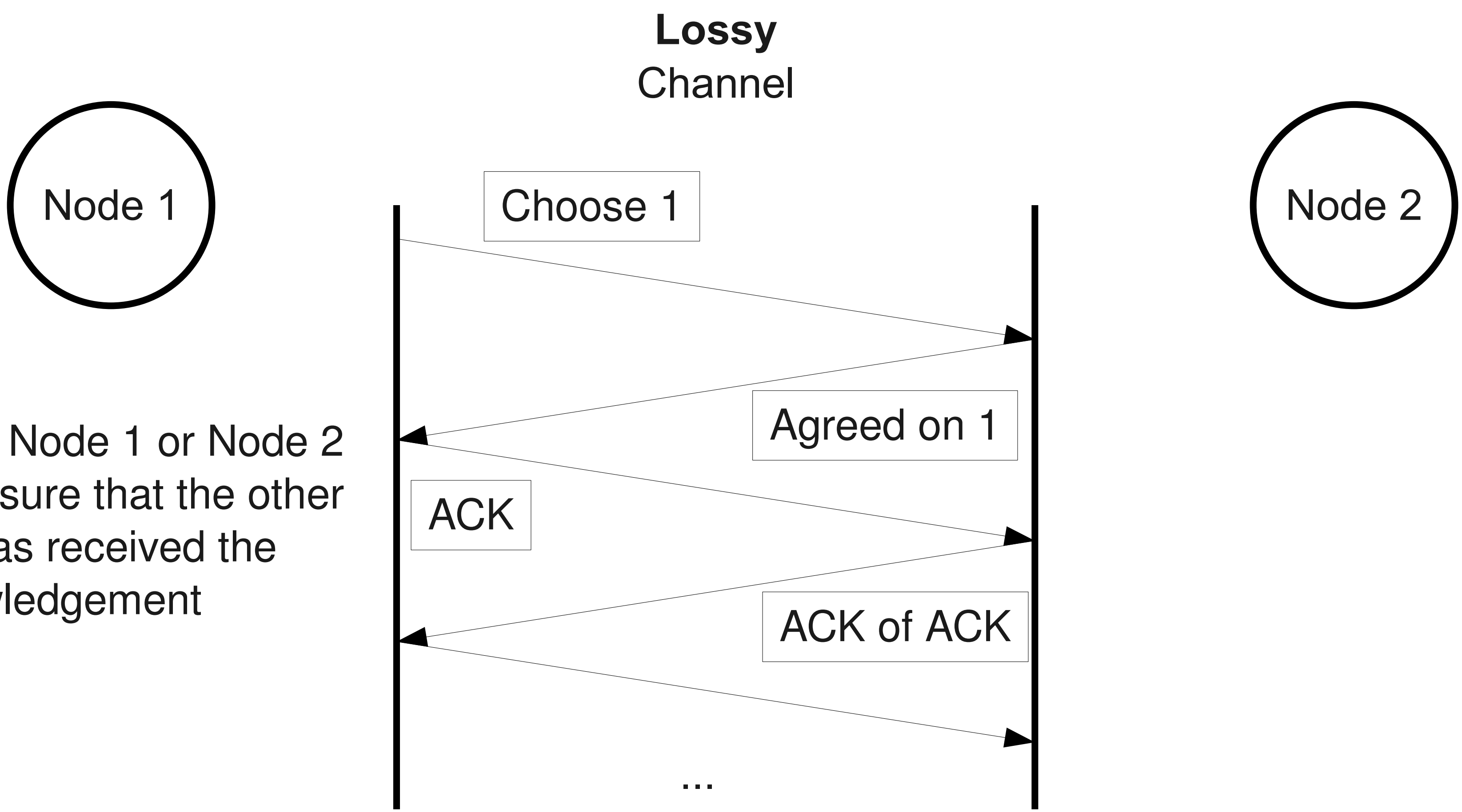
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Consensus

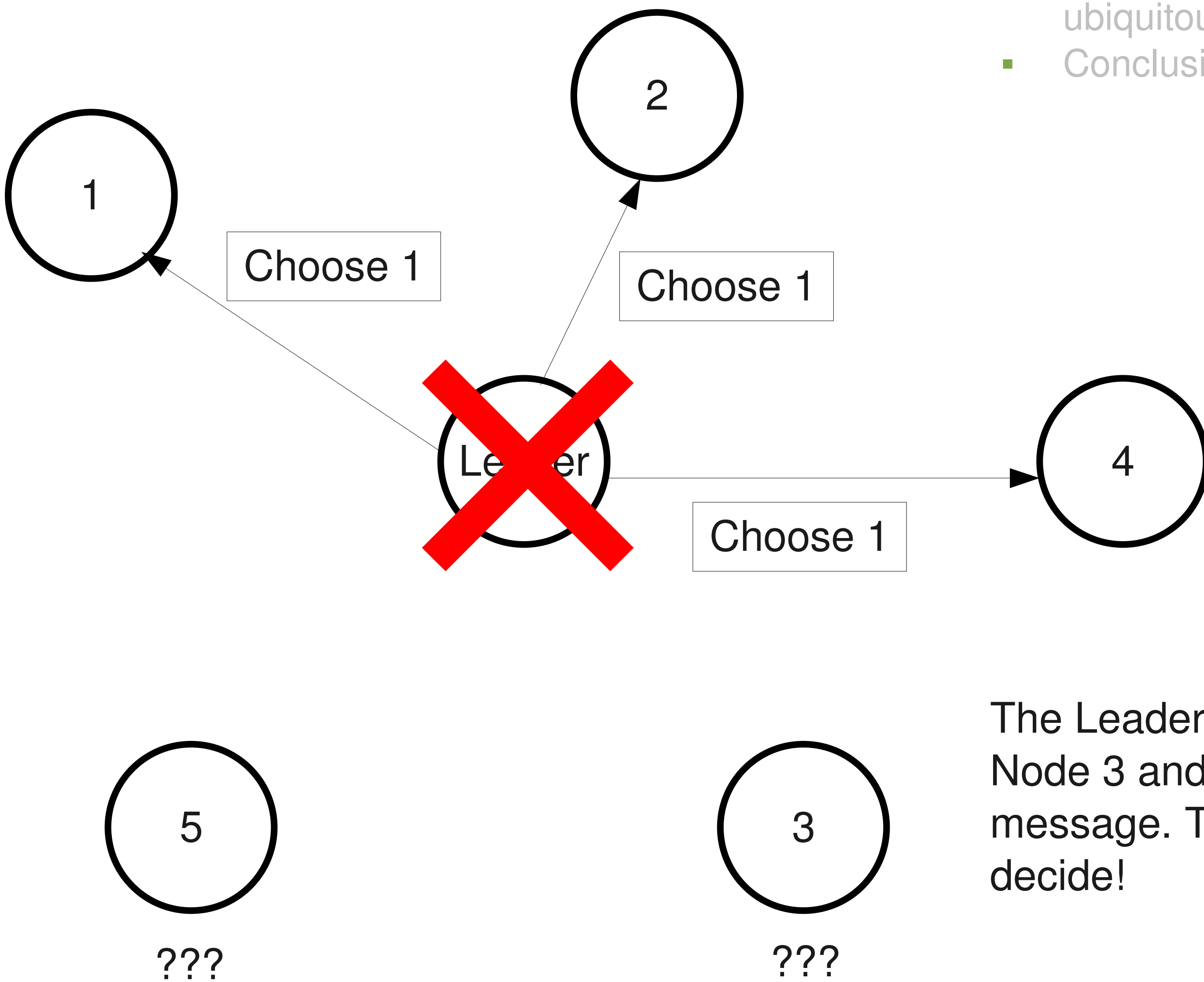
Presented by Benjamin Knecht
Mentor: Johannes Schneider
Papers chosen by Peter Widmayer

Outline

- Introduction
- Paper 1: Harmful dogmas
- Paper 2: Agreement with ubiquitous faults
- Conclusion



Neither Node 1 or Node 2 can be sure that the other node has received the acknowledgement



The Leader crashed and Node 3 and 5 didn't get a message. The system cannot decide!

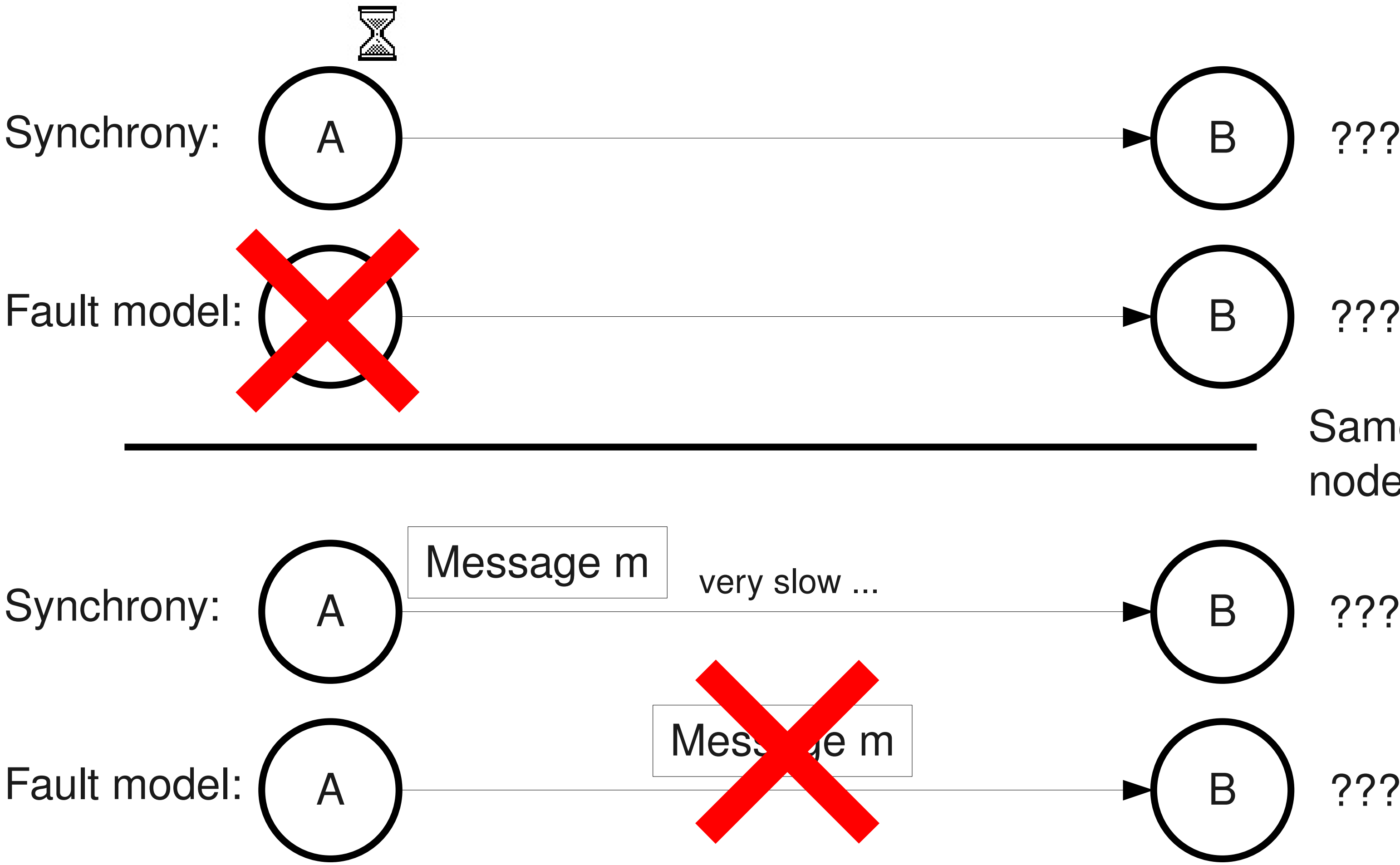
- What is Consensus?
  - Termination: Every non-faulty node eventually decides
  - Agreement: All non-faulty nodes decide on the same value
  - Validity: The decided value must be the input of at least one node

- In an asynchronous system with $n > 1$ nodes no deterministic algorithm can solve the consensus problem, if there are node failures.
- So everybody looks at synchronous systems ...
- ... or randomized consensus algorithms

- Three features of the classic modelling decisions overcomplicate the problem
- Those features have gained the status of dogmas:
  - Synchrony vs. Fault
  - Process vs. Link failures
  - Flaws in the definition of consensus
- Heard – Of model as new proposal

- Dogma 1: Synchrony vs. Fault
  - Synchrony model
    - Are processes and links synchronous or asynchronous?
  - Fault model
    - Do processes crash? Are the links reliable?
  - But: Can you really distinguish those two models so easily?

- Difference of failure and synchrony issues



Same consequence for node B in every case

- Dogma 1: Synchrony vs. fault
  - So called component failure model
  - Consequences:
    - Models with reliable links (unreliable links are ignored)
    - Increasing synchrony to handle impossibility
    - No investigations for alternative fault models

- Dogma 2: Process vs. Link failures
  - What is crashing/failing?
  - Is there a problem at the sender or receiver?
  - Failed components are not trusted any more!

  - Point of failure is often unknown
  - Failed components may recover
  - Not trusting a failed component is harmful in a environment with transient faults

- Dogma 3: Definition flaws
  - Transmission: Every non-faulty process eventually decides.
  - Once a process fails, it doesn't have to terminate anymore, even if it recovers.
  - Too weak in presence of transient failures
  - → Every process eventually decides!

- Model proposal: Heard – Of model
  - Only transmission failures occur
    - Don't care if something failed or was just too slow
    - Process and link failures are handled the same way
    - Every single process – correct or faulty – has to terminate
  - Component failures can be represented by transmission failures
  - Crash recovery is handled

- Communication failure model
  - Faults can happen anywhere
  - Recovery after those faults is possible
  - Avoids undesirable situations by not pointing fingers
  - Synchronous communication (α, β) is faulty if $\alpha \neq \beta$ (α is the sent message and β the received one)
  - This may change after every clock cycle
  - Therefore detection of failures for future predictions is not helpful

- Fault types and their combinations
  - Omissions: (α, β): with $\alpha \neq \Omega = \beta$
    - Loss of a message
  - Additions: (α, β): with $\alpha = \Omega \neq \beta$
    - Creation of message without sending one
  - Corruption: (α, β): with $\Omega \neq \alpha \neq \beta \neq \Omega$
    - Alteration of message content
  - Byzantine: All three of the above occur

- Impossibility of strong majorities
  - k-agreement problem: at least k nodes decide eventually on the same value
  - Strong majority if $k = \left\lceil \frac{n}{2} \right\rceil + 1$
  - Minimum degree of a graph G is $d(G)$
  - If there are $d(G)$ communication faults (omission, addition, corruption or a combination), a strong majority cannot be reached.
  - Same goes for $\left\lceil \frac{d(G)}{2} \right\rceil$ Byzantine faults

- Possibility of unanimity
  - Unanimity if all nodes agree ($k=n$)
  - We allow at most f faults per clock cycle
  - Edge-connectivity $c(G)$ of graph G is the minimum number of edges one must remove to disconnect the graph
  - Edge-connectivity introduces redundant paths from one node to another and thus can be used for fault tolerant protocols

- Solution to omissions
  - Assume number of faults $f \leq c(G) - 1$
  - A node needs to broadcast its message in each time step until $T(G) - 1$
  - Each node receiving a message at time $t < T(G)$ will broadcast the message until $T(G) - 1$
  - Ends after timeout $T = T^*(G)$
  - T^* is the minimum timeout value

- Solutions to additions
  - Each node just sends its result in every time step and leaves no room for additions
  - Number of faults doesn't matter
  - Terminates after all results propagated through network. That's the diameter $D(G)$ of graph G
    - → Timeout: $T = D(G)$
  - A Spanning tree solves this nicely

- Solution to corruptions
  - Number of faults doesn't matter
  - If value is 1, propagate it
  - If value is 0, wait for messages
  - Since no messages are lost or additionally created, receiving a message is information enough
  - After $T = D(G)$ decide on your value

- Omissions and corruptions
  - Send value in every clock cycle until $T(G)$
    - Omission won't affect unanimity
  - Only send value if it is 1
    - So messages may be corrupted, but enough information
- Omissions and additions
  - Send in every clock cycle to avoid additions
  - No corruption so messages can be trusted

- Additions and corruptions
  - Complex problem: Only sending 1's leaves space for additions and sending every clock cycle doesn't work because messages cannot be trusted!

- Solution: Time Slice
  - Only send 0's during even clock cycles
  - Only send 1's during odd clock cycles
  - However the problem of additions is not solved

- Solution: Reliable Neighbour Transmission
  - Use all $c(G)$ paths to your neighbour to send message
  - Also send along the path the message should take
  - Every receiving node, knows if a path is valid and discards wrong messages
  - Correct messages are forwarded for a certain time
  - Again unanimity is guaranteed with $f \leq c(G) - 1$

- Byzantine faults
  - We can again use the Reliable Neighbour Transmission technique
  - For $f \leq \left\lceil \frac{c(G)}{2} \right\rceil - 1$ Byzantine faults per clock cycle it is possible to achieve unanimity

- Tightness
  - For graphs with $d(G)=c(G)$ the bounds for impossibility and possibility are very tight
    - Rings
    - Toruses
    - Hypercubes
    - Complete graphs
    - etc.

- What about arbitrary graphs?
- Performance was not an issue in paper
- Really a more practical approach?
- Only works for synchronous systems?

Questions?