Distributed Systems
Fault Tolerance

timofei.istomin@unitn.it

credits for the slides to:
Giuliano Mega
Luca Mottola
Davide Frey

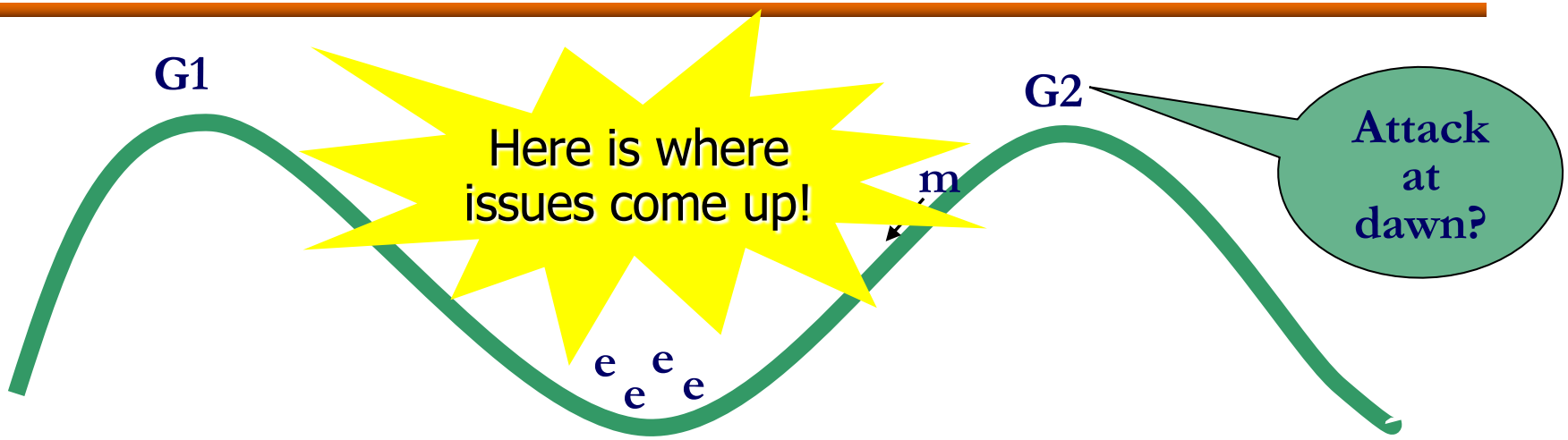
Outline

- Introduction
- An example: client-server communication
- Reliable group communication
- Agreement in process groups
- Atomic commitment

Agreement in process groups

- A number of tasks may require that the members of a group agree on some decision before continuing
  - who is the coordinator, who has access to a resource, if a distributed transaction should be committed or aborted…
- We want the processes to reach an agreement
  - Depending on the algorithm, either all group members or only correct ones
- Many protocols exist, solving slightly different agreement problems, under various assumptions

Consensus with *communication* failures



- Do you remember the coordinated attack?
  - If both generals attack: they win
  - If only one attacks: they lose
- Messengers for communication
  - can be lost/captured/delayed
- How do the generals ensure either win or no attack?

Consensus with *communication* failures

- The coordinated attack problem is not solvable in the presence of arbitrary *communication* failures even in synchronous systems
- In practice… make channels reliable (enough)

Consensus with *process* failures

- Then … what happens when communication is reliable but processes are allowed to fail?
- Consensus can't be solved in asynchronous systems with even a single crash failure …
  - Fischer, Lynch and Patterson 1985
  - We don't know whether a process has crashed or it is just slow…
  - In practice, protocols exist that fail in cases very difficult to trigger and/or preserve safety
- … but it's possible in synchronous systems even with byzantine failures!

Some Terminology

- In our problems, we deal with two kinds of processes:
  - those that behave according to the protocol specification: correct
  - those that do not: faulty
    - even if a single fault occurs during the execution
- We also want our distributed algorithms to respect some fundamental properties:
  - safety: "bad" things never happen
  - liveness: "good" things eventually happen

The consensus problem

- A group of processes must agree on some value
- Each process i *proposes* a value v_i
- Properties of distributed consensus:
  - **Agreement:** all correct processes should agree on the same value v
  - **Validity:** v should have been proposed by some process
  - **Termination:** all correct processes eventually agree

A more difficult problem is uniform consensus*:*

the **agreement** property becomes the **uniform agreement** property, i.e., **all** processes (correct or not) agree on the same value

The consensus problem

Process	Proposes	Agrees on
1	v_1=A	v
2	v_2=B	v
3	v_3=C	v
4	fails	not expected to agree

All agree on the same value
v ∈ {A,B,C}

Faulty

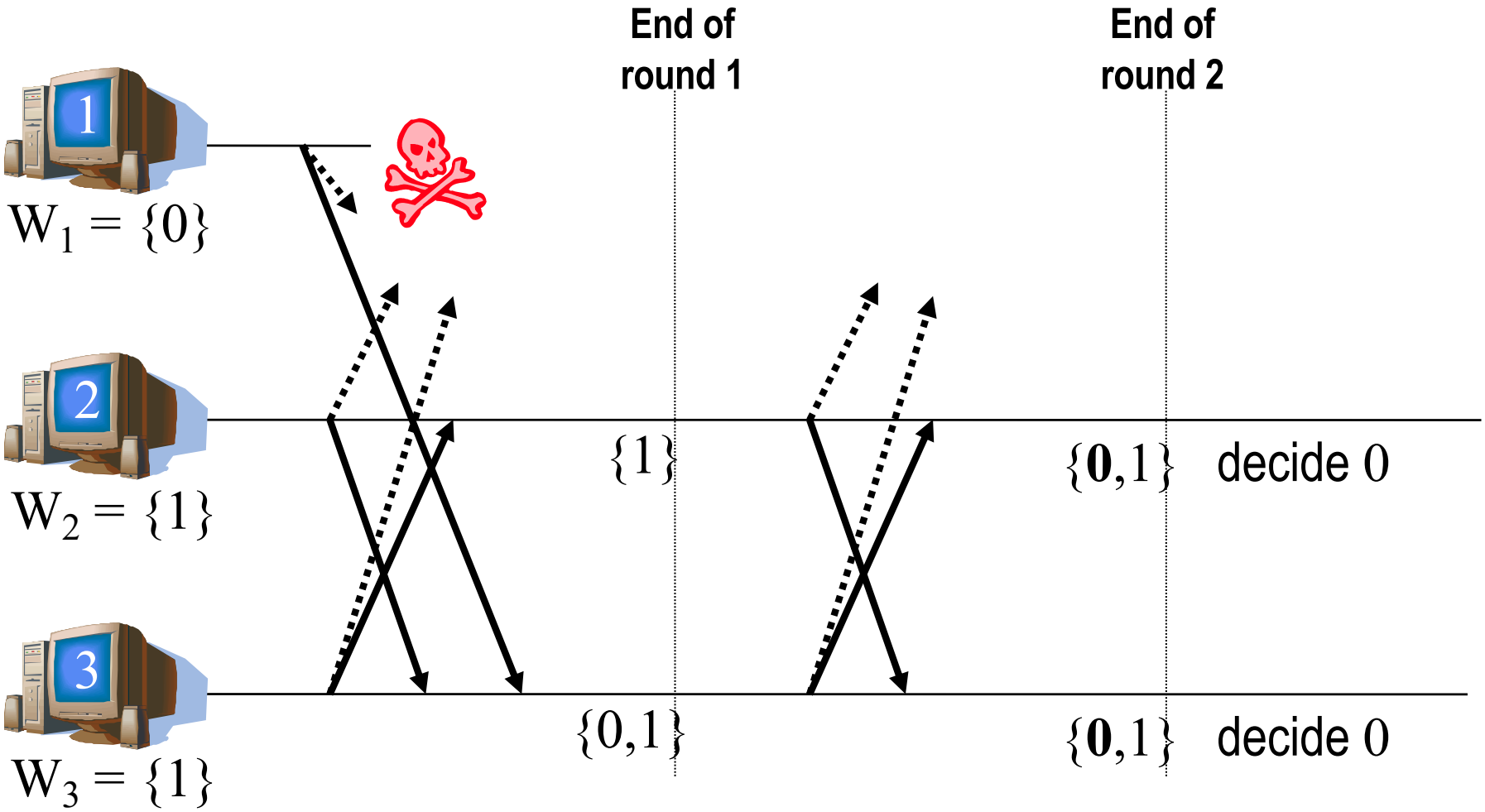
Synchronous fail-stop consensus

- Relaxed failure assumptions:
  - channels are reliable
  - processes fail by crashing
  - the system is synchronous
- "Synchronous network" model [Lynch 96]:
  - processes execute in synchronized rounds
  - messages sent in a round are received by the end of it
- Under these assumptions…
  - … assuming it is sufficient that a single non-faulty (correct) process is enough
  - … the problem can be solved provided that the processes take at least $f + 1$ ***rounds***, with f being a bound on the number of faulty processes (crash failures)

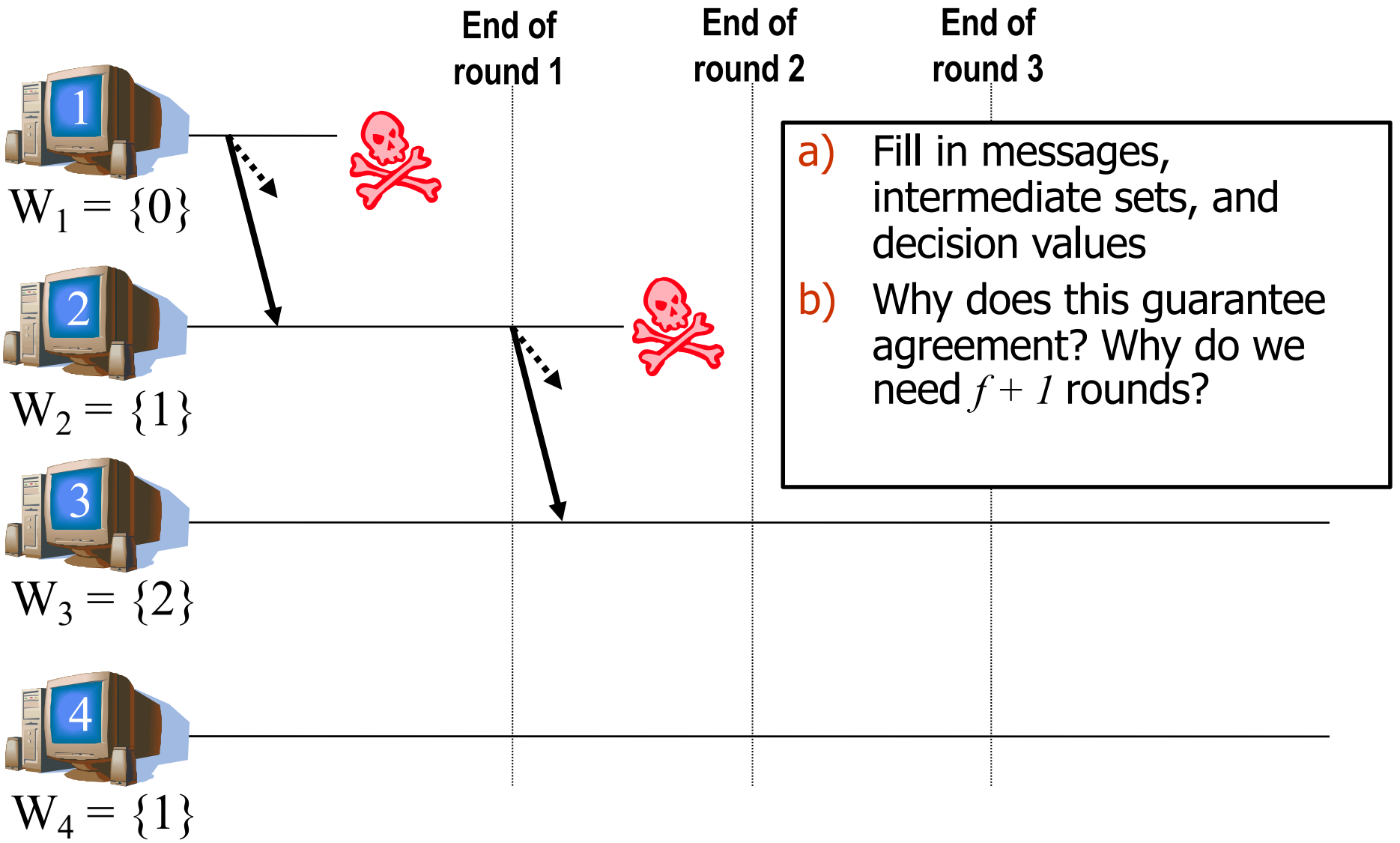
FloodSet

- A simple protocol solving synchronous consensus
- Each process maintains a set $W_i \subseteq V$, initialized with the proposed value $v_i \subseteq V$
- The following steps are repeated for $f+1$ rounds
  - Each process P_i broadcasts W_i
  - For each received W_j, $i \neq j$, each process P_i sets $W_i := W_i \cup W_j$
- The decision is made after $f+1$ rounds
  - If $|W_i|=1$ the single element is chosen
  - If $|W_i|>1$ then $h(W_i)$ is chosen, where h is a rule over W_i, e.g., the smallest element

Example with $f = 1$, $h = \min$



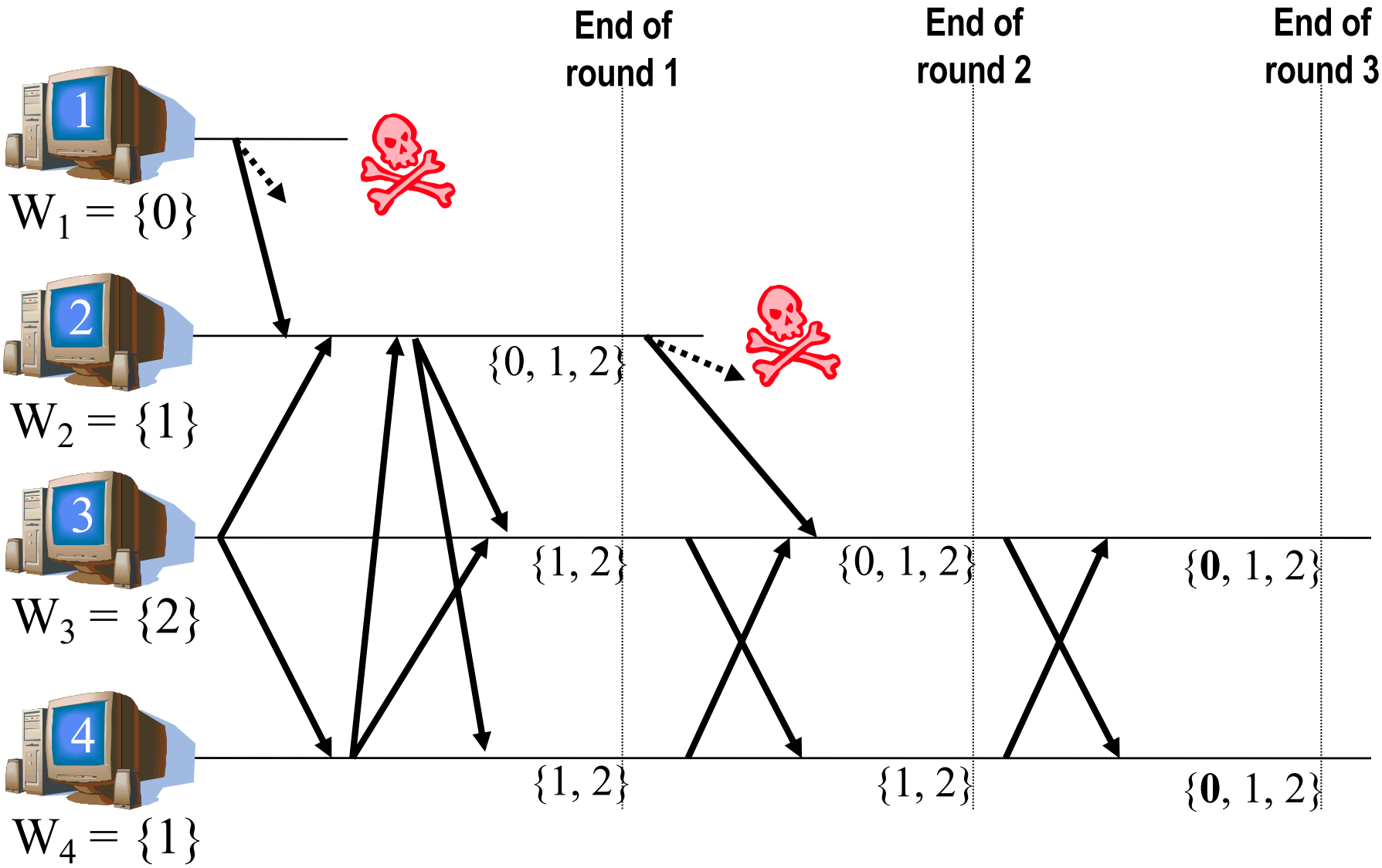
Exercise: $f = 2$, h = min



a) Fill in messages, intermediate sets, and decision values

b) Why does this guarantee agreement? Why do we need $f + 1$ rounds?

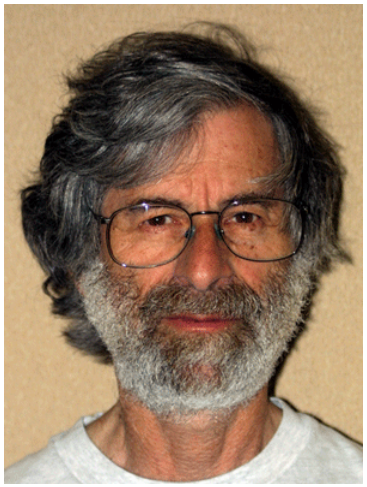
Exercise: f = 2, h = min



FloodSet algorithm

- Why does this work?
  - f failures: worst case, one failure per round
  - with $f + 1$ rounds, at least one failure-free round
  - enough for everyone's W set to be equal
  - if the decision rule is deterministic and the same for everyone, the result is the same

Byzantine Consensus



- Consensus is trickier with Byzantine failures
- The problem can be described in terms of armies and generals [Lamport 1982]
  - N generals on the hills must reach a coordinated decision on whether to attack or retreat
  - But some generals are traitors: they can lie to each other, or collude
- The first proposed solution: Oral Messages algorithm
  - Synchronous system
  - Requires $n=3f+1$ processes to tolerate f traitors

Ensures there are enough "loyal" processes, to overcome the false information introduced by the "traitors"...

Byzantine Consensus

- Proven not solvable in general in asynchronous systems even with a single crash failure
- In practice, protocols exist that are very difficult to break
  - Paxos
  - Raft
  - Bitcoin
  - etc…
- Covered in Distributed Systems 2 ...