

Department of Computer Science

PURDUE
UNIVERSITY

CS505: Distributed Systems

Lecture 14: More Agreement Problems



▶ **Uniform Reliable Broadcast**

▶ **Terminating Reliable Broadcast**

▶ **Leader Election**

Uniform Reliable Broadcast

▶ By now you know the difference between

1. Agreement

A process which fails may diverge (we can win in terms of liveness)

2. Uniform agreement

No process may do something that others do or vice versa, even if it is bound to fail (needed for safety)

▶ Choice depends largely

- On subsequent runs
- The use of the corresponding module within others

Specification

I. No duplication

- No message is delivered more than once

II. No creation

- No message is delivered unless it was broadcast

III. Validity

- If p_i and p_j are correct, then every message broadcast by p_i is eventually delivered by p_j

IV. Uniform agreement

- If one (correct or not) process delivers a message m , every correct process eventually delivers m

Proposition

correct $\leftarrow \Pi$

delivered, forward, ack[Message] $\leftarrow \emptyset$

upon crash of p_i **do**

 correct \leftarrow correct $\setminus \{p_i\}$

upon broadcast(m) **do**

 forward \leftarrow forward $\cup \{m\}$

 send(m) to all

upon receive(m) from $\{p_i\}$ **do**

 ack[m] \leftarrow ack[m] $\cup \{p_i\}$

if $m \notin$ forward **then**

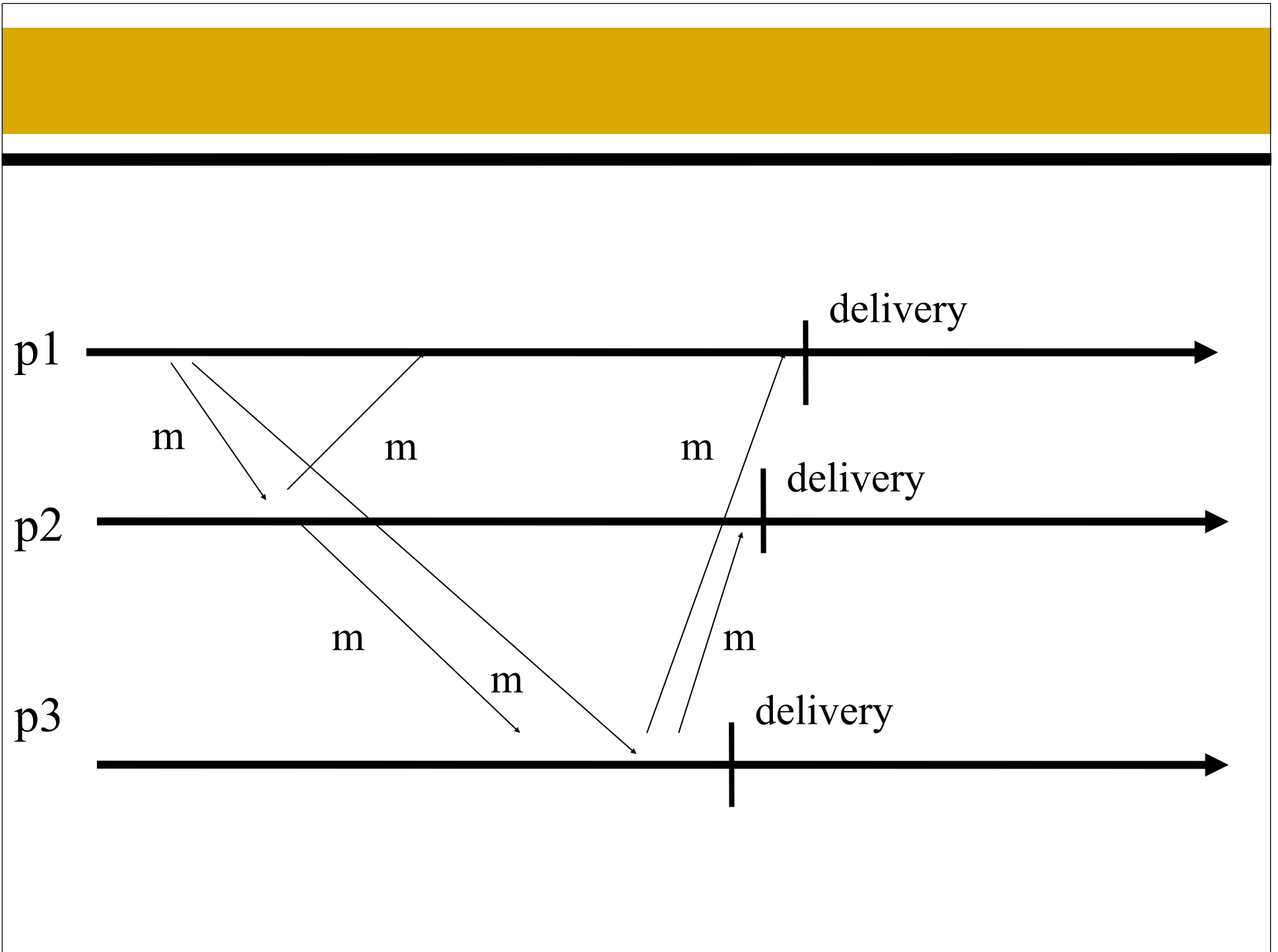
 forward \leftarrow forward $\cup \{m\}$

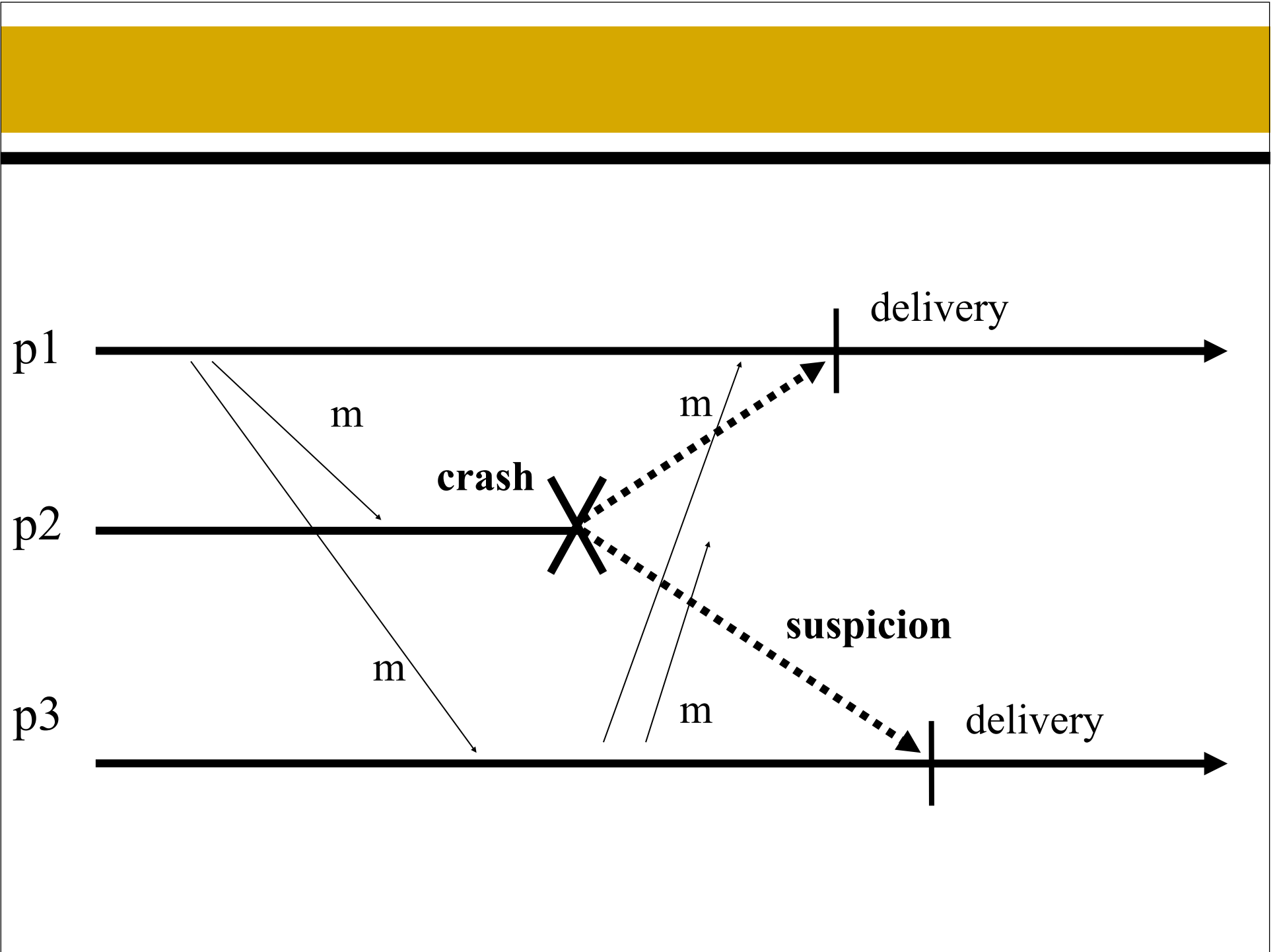
 send(m) to all

upon $\exists m \mid$ (correct \subseteq ack[m]) and ($m \notin$ delivered) **do**

 delivered \leftarrow delivered $\cup \{m\}$

 deliver(m)





Assessment

- ▶ **“Costly” implementation**
 - Requires P
- ▶ **Can we implement it *without* P ?**
 - A majority of correct processes is sufficient...
 - What about false suspicions?
- ▶ **θ outputs a set of *trusted* processes [ATD'99]**
 - θ -Completeness
There is a time after which correct processes do not trust any crashed process
 - θ -Accuracy
If there is a correct process, then at any time, every process trusts at least one correct process

URB with θ

```
ack[Message]  $\leftarrow \emptyset$ 
```

```
upon broadcast(m) do ack[m]  $\leftarrow$  {self}  
  fork task diffuse(m)  
  return
```

```
task diffuse(m) do  
  while(true)  
    send(m) to all \ \ fair-lossy channel  
    if  $D \in \text{ack}[m]$  and m has not been delivered then  
      deliver(m)
```

```
upon receive(m) from  $p_i$  do  
  ack[m]  $\leftarrow$  ack[m]  $\cup$   $\{p_i\}$   
  if not forked diffuse(m) yet  
    ack[m]  $\leftarrow$  ack[m]  $\cup$  {self}  
    fork task diffuse(m)
```

Assessment

▶ θ

1. Allows us to trust crashed processes
2. Allows us to not trust correct processes

▶ A process p waits for delivery

- Until it received acknowledgements from all trusted processes
- 1. Not a problem; p just might wait for too many acks, but eventually D will not trust any crashed processes anymore
- 2. Not a problem; as long as there is a correct process we will trust some correct process, say q
 - p can go ahead and deliver if we have q 's ack
 - p knows then that at least one correct process (q) exists and it has the message, thus it will handle it eventually even if p itself crashes (the message “survives”)

Terminating Reliable Broadcast

- ▶ ***Like*** with reliable broadcast, correct processes in TRB agree on the set of messages they deliver
- ▶ ***Like*** with uniform reliable broadcast, every correct process in uniform TRB delivers every message delivered by any process
- ▶ ***Unlike*** in (uniform) reliable broadcast, every correct process delivers a message, even if the broadcaster crashes

- ▶ **The problem is defined for a specific broadcaster process p_i**
- ▶ **Process p_i is supposed to broadcast a message m (not φ)**
- ▶ **The other processes need to deliver m if p_i is correct but may deliver φ if p_i crashes**

Specification

I. No duplication

- No message is delivered more than once

II. No creation

- No message other than φ is delivered unless it was broadcast

III. Validity

- If p_i and p_j are correct, then every message broadcast by p_i is eventually delivered by p_j

IV. (Uniform) agreement

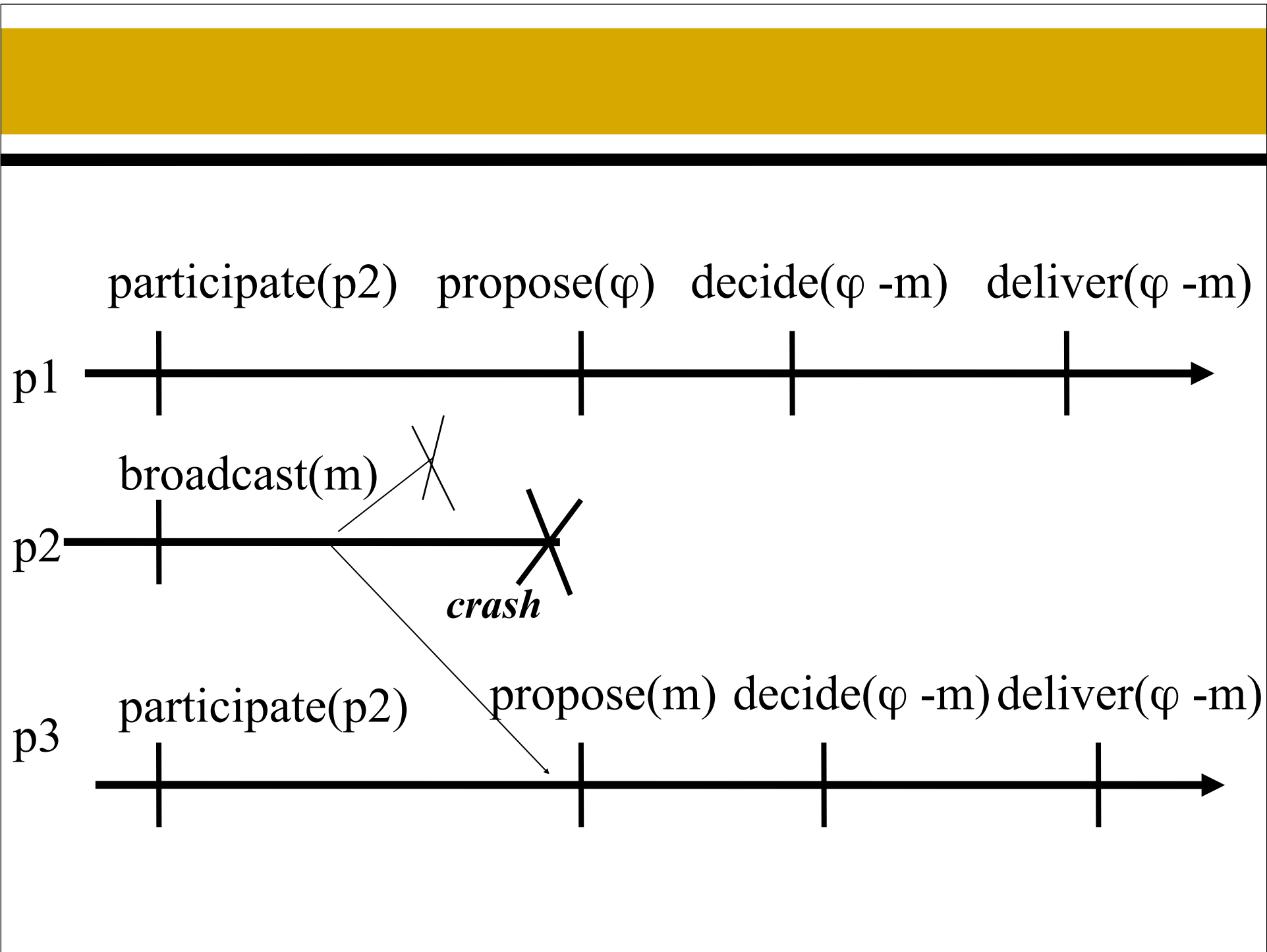
- If one correct (any) process delivers a message m , every correct process eventually delivers m

V. Termination

- Every correct process eventually delivers a message

Proposition

```
s, prop ← ⊥; correct ← ∅  
  
upon crash of  $p_i$  do  
    correct ← correct \ { $p_i$ }  
  
upon broadcast(m) do  
    s ← self  
    send(m) to all  
  
upon participate(src) do  
    s ← src  
  
upon receive(m) from  $p_i$  do  
    prop ← m  
  
upon  $s \neq \perp$  and  $s \notin \text{correct}$  do  
    prop ←  $\varphi$   
  
upon  $s \neq \perp$  and  $\text{prop} \neq \perp$  do  
    propose(prop)  
  
upon decide(decision) do  
    deliver(decision, s)
```



Assessment

▶ Why do we need consensus?

- Reliable broadcast?

▶ Requires P

- P is sufficient (the previous algorithm proves it)
- Is P necessary?

▶ Can we implement P with TRB?

- Then TRB requires at least P
- Every process p_i periodically ($1/n$) does broadcast ("hello") others participate (p_i)
- Whenever a process does `deliver` (p_i, φ) it suspects p_i

Leader Election

I. Agreement

No two processes are leader at the same time

II. Termination

At any time, some process is eventually leader

III. Stability

Unless it crashes, a leader remains permanently leader

With P

▶ Processes have a priori knowledge of a function O , $O(p_i) = \{p_j, \dots, p_k\}$, depicting a total order among processes (a *royal hierarchy*)

▶ **Validity**

If process p_i is **leader** at time t , then every process in $O(p_i)$ has crashed by t

▶ **Termination**

If some process is correct, then at any time, some process is eventually leader

Proposition

init

suspected $\leftarrow \emptyset$

if $O(\text{self}) = \emptyset$ **then**

leader()

upon crash of p_i **do**

suspected \leftarrow suspected $\cup \{p_i\}$

if $O(\text{self}) \setminus \text{suspected} = \emptyset$ **then**

leader()

Do We Need P ?

- ▶ We have just shown that P is sufficient to solve leader election
- ▶ Is P necessary?
 - Yes
 - Proof omitted

References

- ▶ ***Revisiting the Weakest Failure Detector for Uniform Reliable Broadcast.*** M. Aguilera, S. Toueg, B. Deianov. 13th International Symposium on Distributed Computing, 1999.
- ▶ **Stable Leader Election.** M. Aguilera, C. Delporte-Gallet, H. Fauconnier, S. Toueg. 15th International Conference on Distributed Computing, 2001.