29/11/23

Dependable Distributed Systems
Master of Science in Engineering in Computer Science

AA 2023/2024

LECTURE 26: BROADCAST IN PRESENCE OF BYZANTINE PROCESSES

Recap on Byzantine processes

Byzantine processes may

1. deviate arbitrarily from the instructions that an algorithm assigns to them
   - creating fake messages
   - dropping messages
   - delay the deliveries
   - altering the content of messages
   - …
2. act as if they were deliberately preventing the algorithm from reaching its goals

Basic step to fight Byzantine processes

Using cryptographic mechanisms to implement the authenticated perfect links abstraction

… But, cryptography alone does not allow to tolerate Byzantine processes

- Considering an arbitrary-faulty sender, asking him/her to digitally sign every broadcast message does not help at all (it may simply sign the two different messages)

↓ manipulation of content

Correct and faulty state

As in the crash failure model, we distinguish between *faulty* and *correct* processes

NOTE: a Byzantine process may act arbitrarily, and no mechanism can guarantee anything that relates to its actions.

We do not define any “uniform” variants of primitives in the Byzantine failure model.

P2P communication channel

Authenticated Perfect Link

Module 2.5: Interface and properties of authenticated perfect point-to-point links

Module:

Name: AuthPerfectPointToPointLinks, **instance** *al*.

Events:

Request: ⟨ *al*, *Send* | q, m ⟩: Requests to send message m to process q.

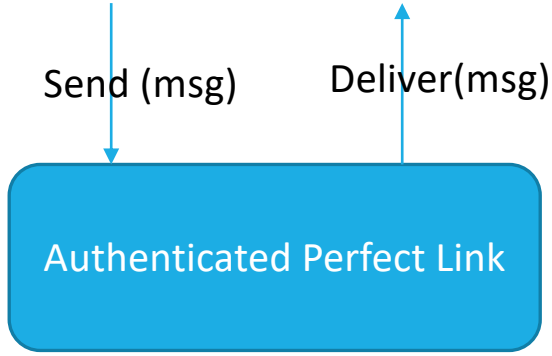
Indication: ⟨ *al*, *Deliver* | p, m ⟩: Delivers message m sent by process p.

Properties:

AL1: *Reliable delivery:* If a correct process sends a message m to a correct process q, then q eventually delivers m.

AL2: *No duplication:* No message is delivered by a correct process more than once.

AL3: *Authenticity:* If some correct process q delivers a message m with sender p and process p is correct, then m was previously sent to q by p.



Same as Perfect point-to-point links

Byzantine consistent broadcast specification

Module 3.11: Interface and properties of Byzantine consistent broadcast

Module:

Name: ByzantineConsistentBroadcast, **instance** *bcb*, with sender s.

Events:

Request: $\langle$ *bcb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes. Executed only by process s.

Indication: $\langle$ *bcb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

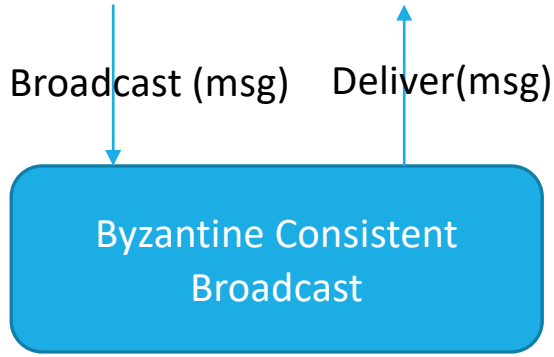
Properties:

BCB1: *Validity:* If a correct process p broadcasts a message m, then every correct process eventually delivers m.

BCB2: *No duplication:* Every correct process delivers at most one message.

BCB3: *Integrity:* If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

BCB4: *Consistency:* If some correct process delivers a message m and another correct process delivers a message m', then $m = m'$.



Byzantine consistent broadcast implementation

Algorithm 3.16: Authenticated Echo Broadcast

Implements:
ByzantineConsistentBroadcast, **instance** *bcb*, with sender s.

Uses:
AuthPerfectPointToPointLinks, **instance** *al*.

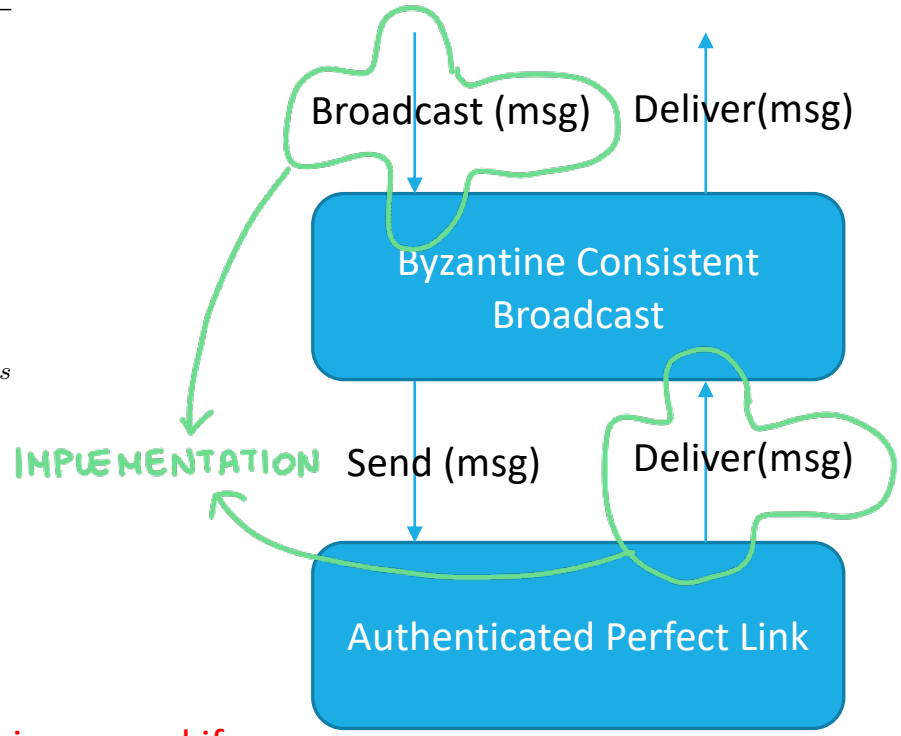
upon event ⟨ *bcb*, *Init* ⟩ **do**
sentecho := FALSE;
delivered := FALSE;
echos := $[\perp]^N$;

upon event ⟨ *bcb*, *Broadcast* | m ⟩ **do** // only process s
forall $q \in \Pi$ **do**
trigger ⟨ *al*, *Send* | q, [SEND, m] ⟩;

upon event ⟨ *al*, *Deliver* | p, [SEND, m] ⟩ **such that** $p = s$ **and** *sentecho* = FALSE **do**
sentecho := TRUE;
forall $q \in \Pi$ **do**
trigger ⟨ *al*, *Send* | q, [ECHO, m] ⟩;

upon event ⟨ *al*, *Deliver* | p, [ECHO, m] ⟩ **do**
if $echos[p] = \perp$ **then**
$echos[p] := m$;

upon exists $m \neq \perp$ such that $\#(\{p \in \Pi \mid echos[p] = m\}) > \frac{N+f}{2}$
and *delivered* = FALSE **do**
delivered := TRUE;
trigger ⟨ *bcb*, *Deliver* | s, m ⟩;



Correctness is ensured if N>3f

Algorithm 3.16: Authenticated Echo Broadcast

Implements:
ByzantineConsistentBroadcast, **instance** *bcb*, with sender s.

Uses:
AuthPerfectPointToPointLinks, **instance** *al*. ECHO menages

upon event $\langle$ *bcb*, *Init* $\rangle$ **do**
sentecho := FALSE;
delivered := FALSE;
echos := $[\perp]^N$; vector: I store ECHO received from other pro

upon event $\langle$ *bcb*, *Broadcast* | m $\rangle$ **do** // only process s
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [SEND, m] $\rangle$;] CORRECT

upon event $\langle$ *al*, *Deliver* | p, [SEND, m] $\rangle$ **such that** $p = s$ **and** *sentecho* = FALSE **do** 1 time I received an ECHO
sentecho := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [ECHO, m] $\rangle$; received m to the sender

upon event $\langle$ *al*, *Deliver* | p, [ECHO, m] $\rangle$ **do**
if *echos*[p] = $\perp$ **then** in my pocket
echos[p] := m;

upon exists $m \neq \perp$ such that $\#(\{p \in \Pi \mid echos[p] = m\}) > \frac{N+f}{2}$
and *delivered* = FALSE **do**
delivered := TRUE;
trigger $\langle$ *bcb*, *Deliver* | s, m $\rangle$;

Can I trust the process?

If I want to tolerate 1 faulty pr
↓
I need 4 process

Correctness is ensured if

N>3f

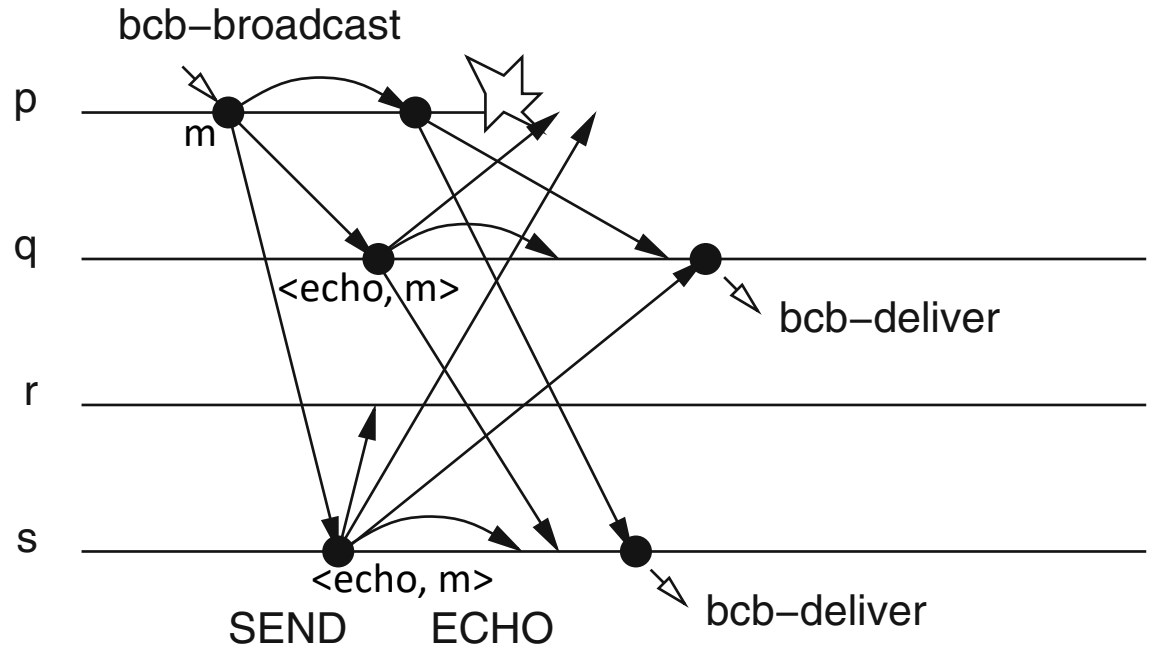
4 > 3·1

5 faulty

N > 3·5 → 16 > 15

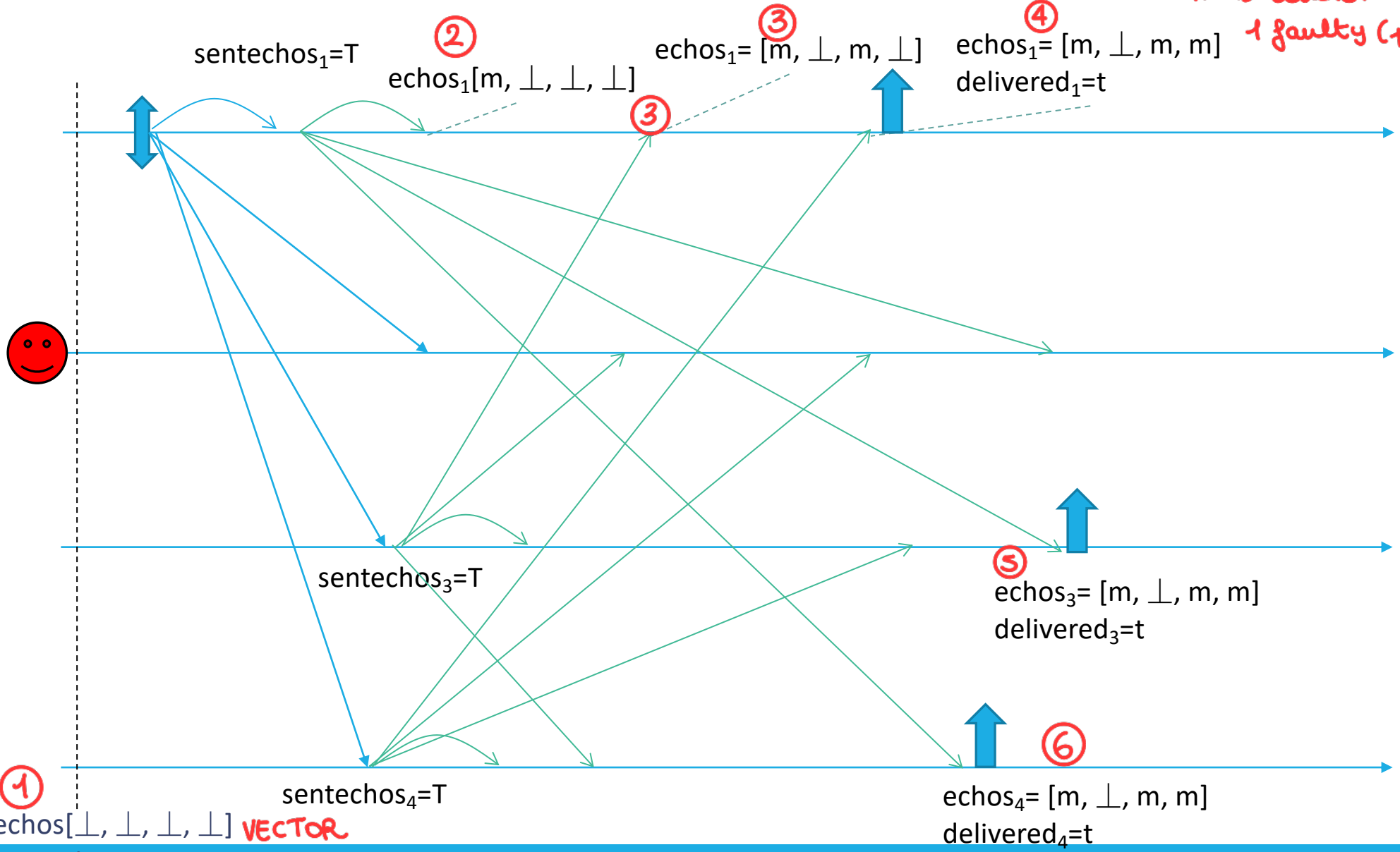
16 correct
5 faulty } 21 process

Byzantine consistent broadcast example



Assumption for Correctness: N > 3f -> f=1, N=4 with p1 acting as sender (correct sender)

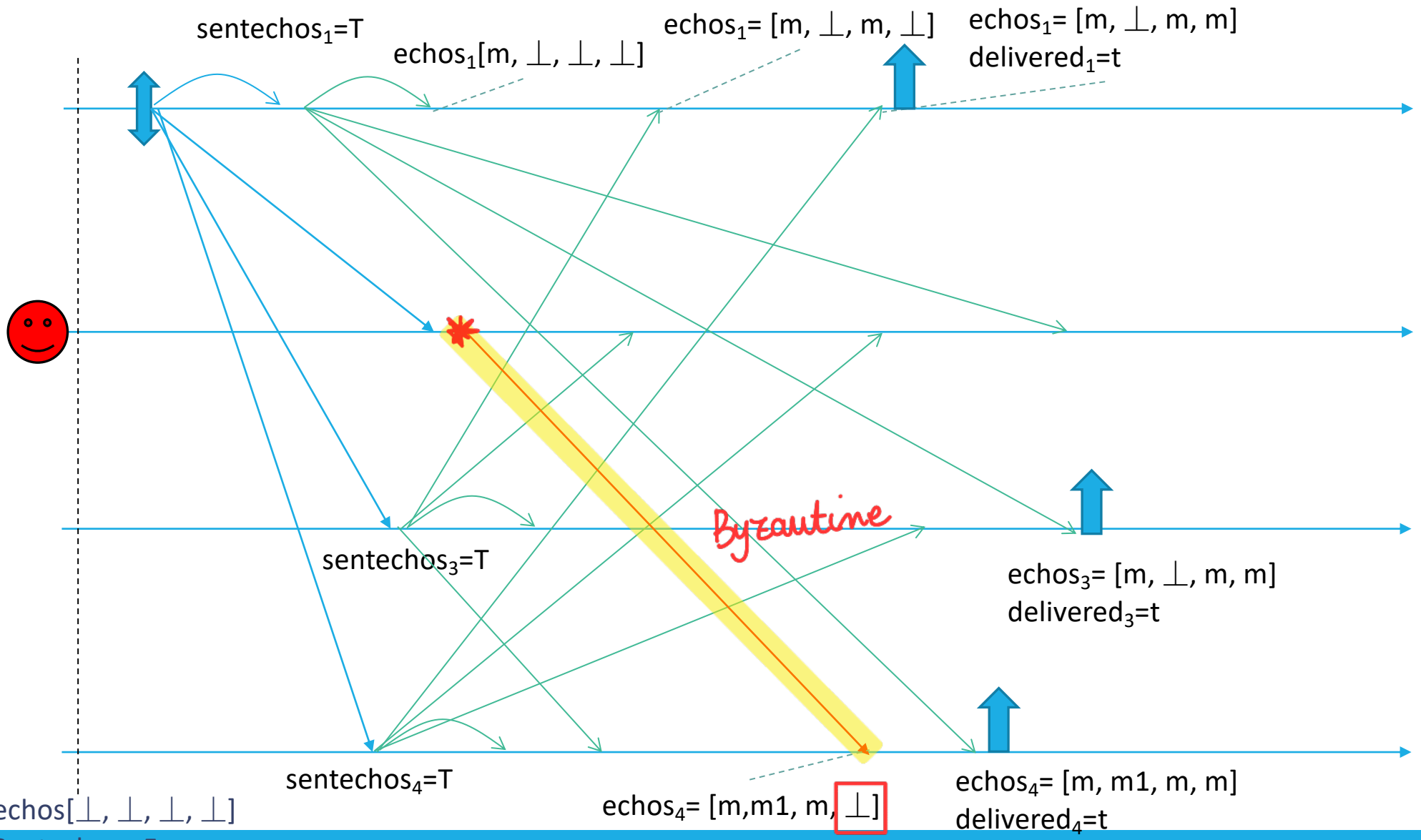
m is consistent
1 faulty (p2)



Sentechos= F

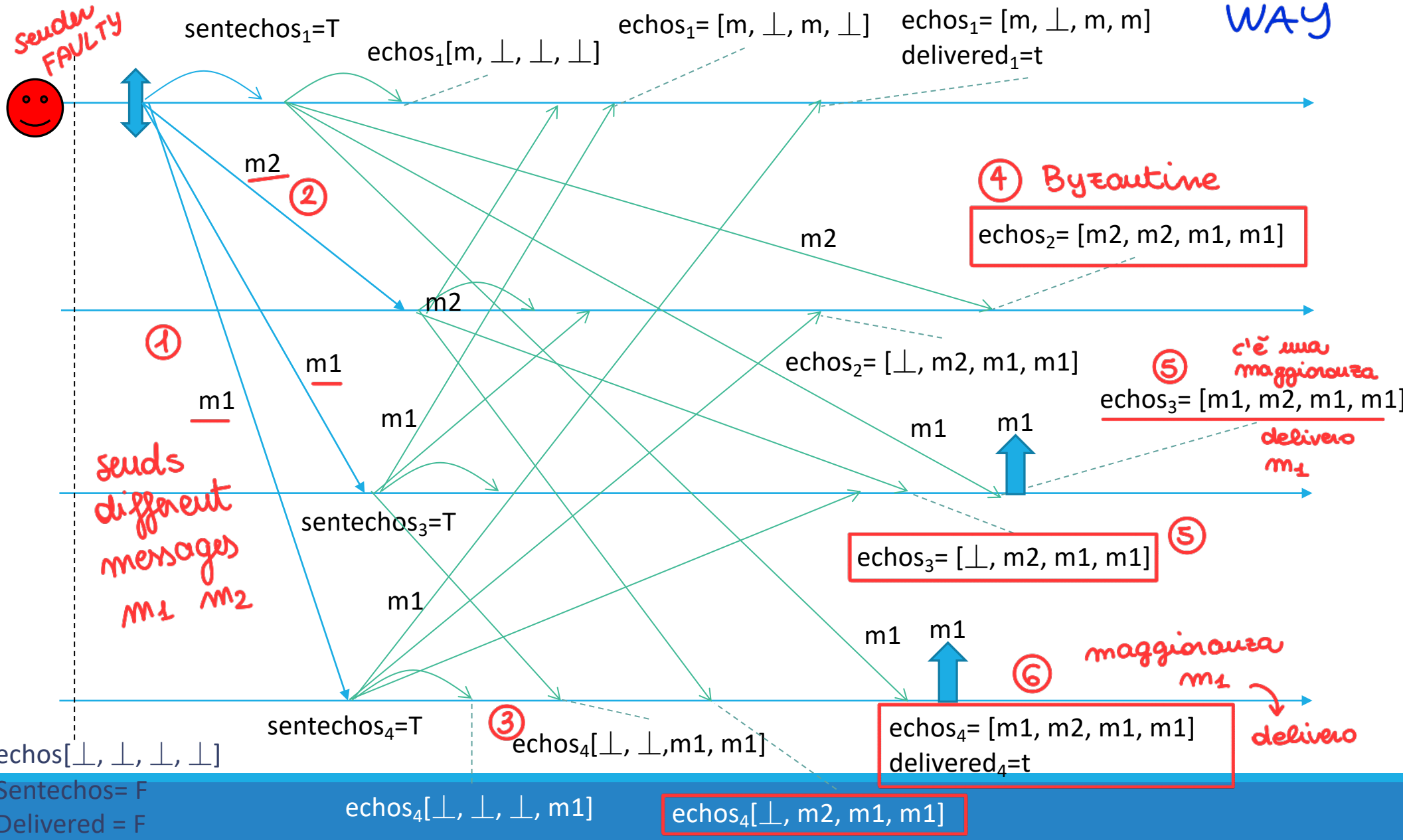
Delivered = F

Assumption for Correctness: N > 3f -> f=1, N=4 with p1 acting as sender (correct sender)



HERE THE FAULTY ANSWER IN AN INCORRECT WAY

Assumption for Correctness: N > 3f -> f=1, N=4 with p1 acting as sender (faulty sender)



Byzantine Reliable Broadcast specification

Module 3.12: Interface and properties of Byzantine reliable broadcast

Module:

Name: ByzantineReliableBroadcast, **instance** *brb*, with sender s.

Events:

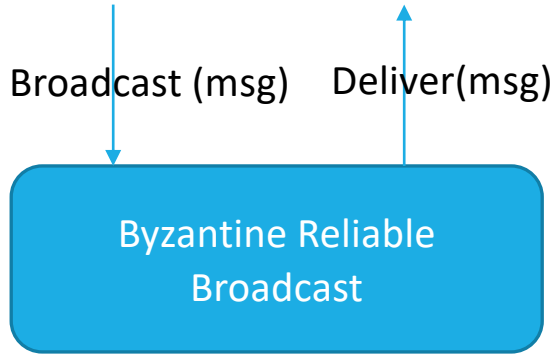
Request: $\langle$ *brb*, *Broadcast* | m $\rangle$: Broadcasts a message m to all processes. Executed only by process s.

Indication: $\langle$ *brb*, *Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p.

Properties:

BRB1–BRB4: Same as properties BCB1–BCB4 in Byzantine consistent broadcast (Module 3.11).

BRB5: *Totality:* If some message is delivered by any correct process, every correct process eventually delivers a message.



Byzantine Reliable Broadcast implementation

Algorithm 3.18: Authenticated Double-Echo Broadcast

Implements:
ByzantineReliableBroadcast, **instance** *brb*, with sender s.

Uses:
AuthPerfectPointToPointLinks, **instance** *al*.

upon event $\langle$ *brb*, *Init* $\rangle$ **do**
sentecho := FALSE;
sentready := FALSE;
delivered := FALSE;
echos := $[\bot]^N$;
readys := $[\bot]^N$;

upon event $\langle$ *brb*, *Broadcast* | m $\rangle$ **do** // only process s
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [SEND, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [SEND, m] $\rangle$ **such that** $p = s$ **and** *sentecho* $=$ FALSE **do**
sentecho := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [ECHO, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [ECHO, m] $\rangle$ **do**
if *echos*$[p] = \bot$ **then**
echos$[p] := m$;

upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid echos[p] = m\}) > \frac{N+f}{2}$
and *sentready* $=$ FALSE **do**
sentready := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [READY, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [READY, m] $\rangle$ **do**
if *readys*$[p] = \bot$ **then**
readys$[p] := m$;

upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid readys[p] = m\}) > f$
and *sentready* $=$ FALSE **do**
sentready := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [READY, m] $\rangle$;

upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid readys[p] = m\}) > 2f$
and *delivered* $=$ FALSE **do**
delivered := TRUE;
trigger $\langle$ *brb*, *Deliver* | s, m $\rangle$;

Algorithm 3.18: Authenticated Double-Echo Broadcast

Implements:
ByzantineReliableBroadcast, **instance** *brb*, with sender s.

Uses:
AuthPerfectPointToPointLinks, **instance** *al*.

upon event $\langle$ *brb*, *Init* $\rangle$ **do**
sentecho := FALSE;
sentready := FALSE;
delivered := FALSE;
echos := $[\bot]^N$;
readys := $[\bot]^N$; ready to deliver

upon event $\langle$ *brb*, *Broadcast* | m $\rangle$ **do** // only process s
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [SEND, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [SEND, m] $\rangle$ **such that** $p = s$ **and** *sentecho* = FALSE **do**
sentecho := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [ECHO, m] $\rangle$;

upon event $\langle$ *al*, *Deliver* | p, [ECHO, m] $\rangle$ **do**
if *echos*$[p] = \bot$ **then**
echos$[p] := m$; store locally : 1° echo I receiving

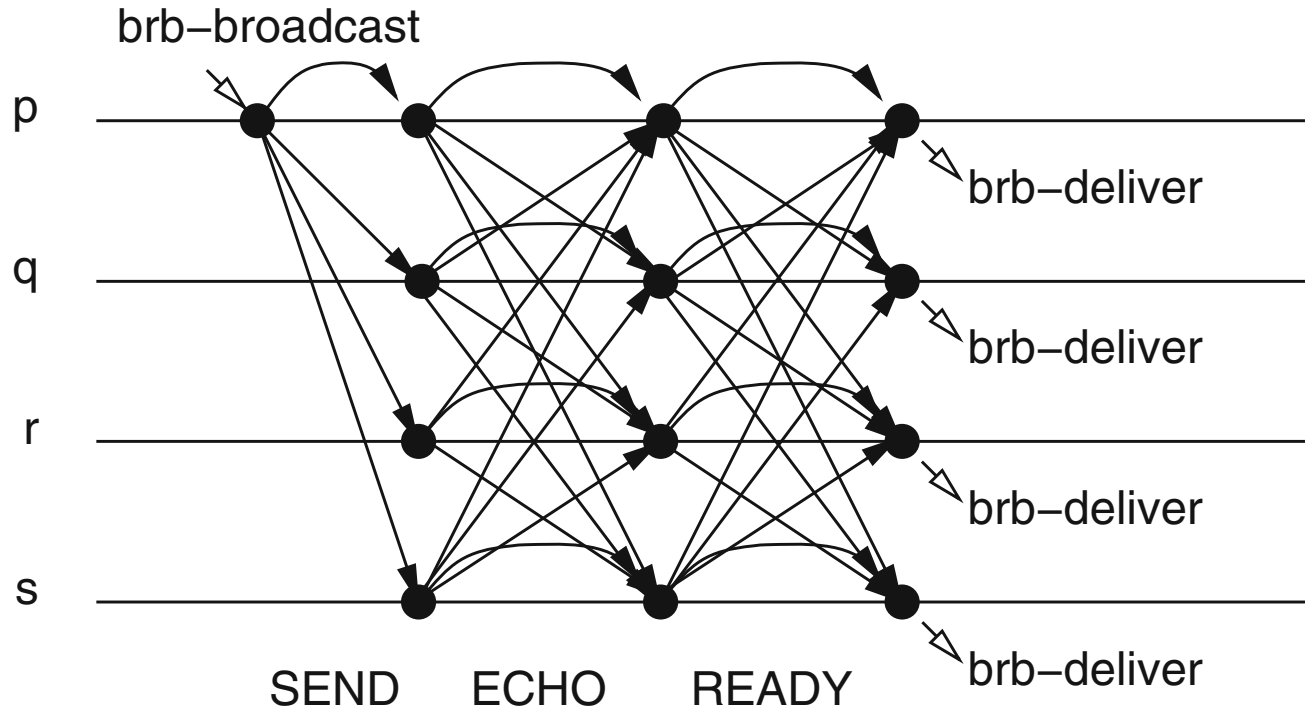
upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid$ *echos*$[p] = m\}) > \frac{N+f}{2}$
and *sentready* = FALSE **do**
sentready := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [READY, m] $\rangle$; } propagate m for which I have most votes

upon event $\langle$ *al*, *Deliver* | p, [READY, m] $\rangle$ **do**
if *readys*$[p] = \bot$ **then**
readys$[p] := m$;

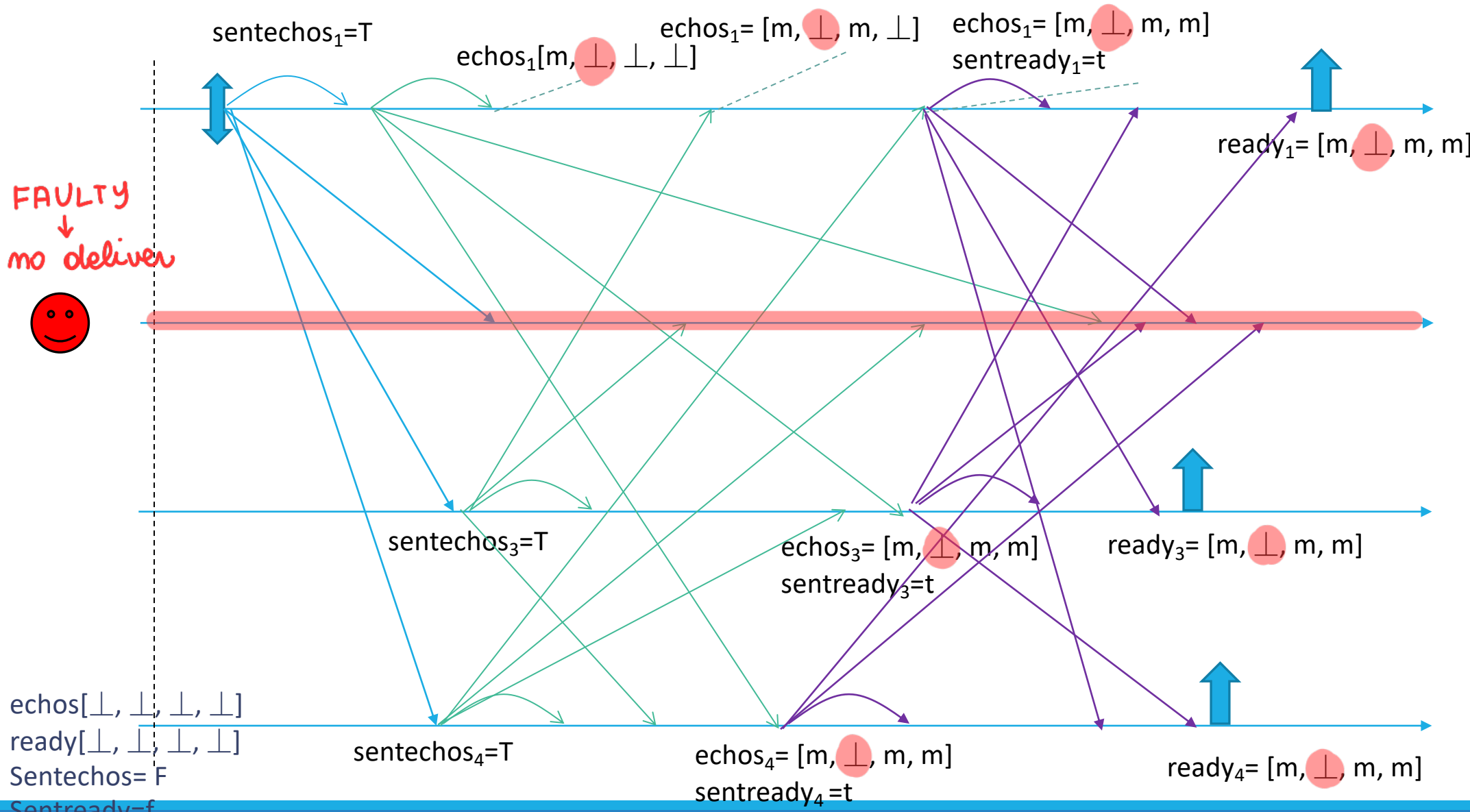
upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid$ *readys*$[p] = m\}) > f$
and *sentready* = FALSE **do**
sentready := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ *al*, *Send* | q, [READY, m] $\rangle$;

upon exists $m \neq \bot$ such that $\#(\{p \in \Pi \mid$ *readys*$[p] = m\}) > 2f$
and *delivered* = FALSE **do**
delivered := TRUE;
trigger $\langle$ *brb*, *Deliver* | s, m $\rangle$;

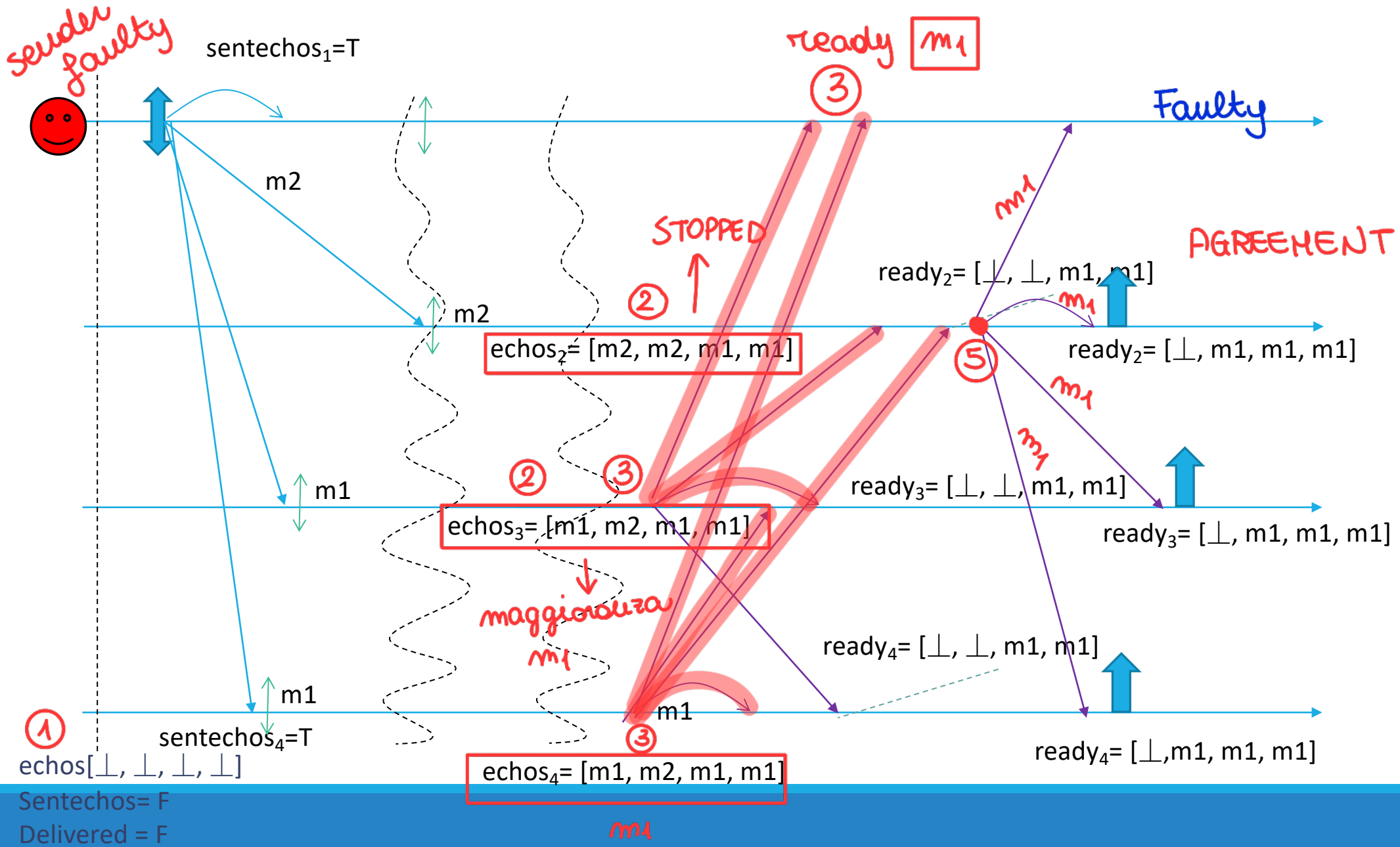
Byzantine Reliable Broadcast example



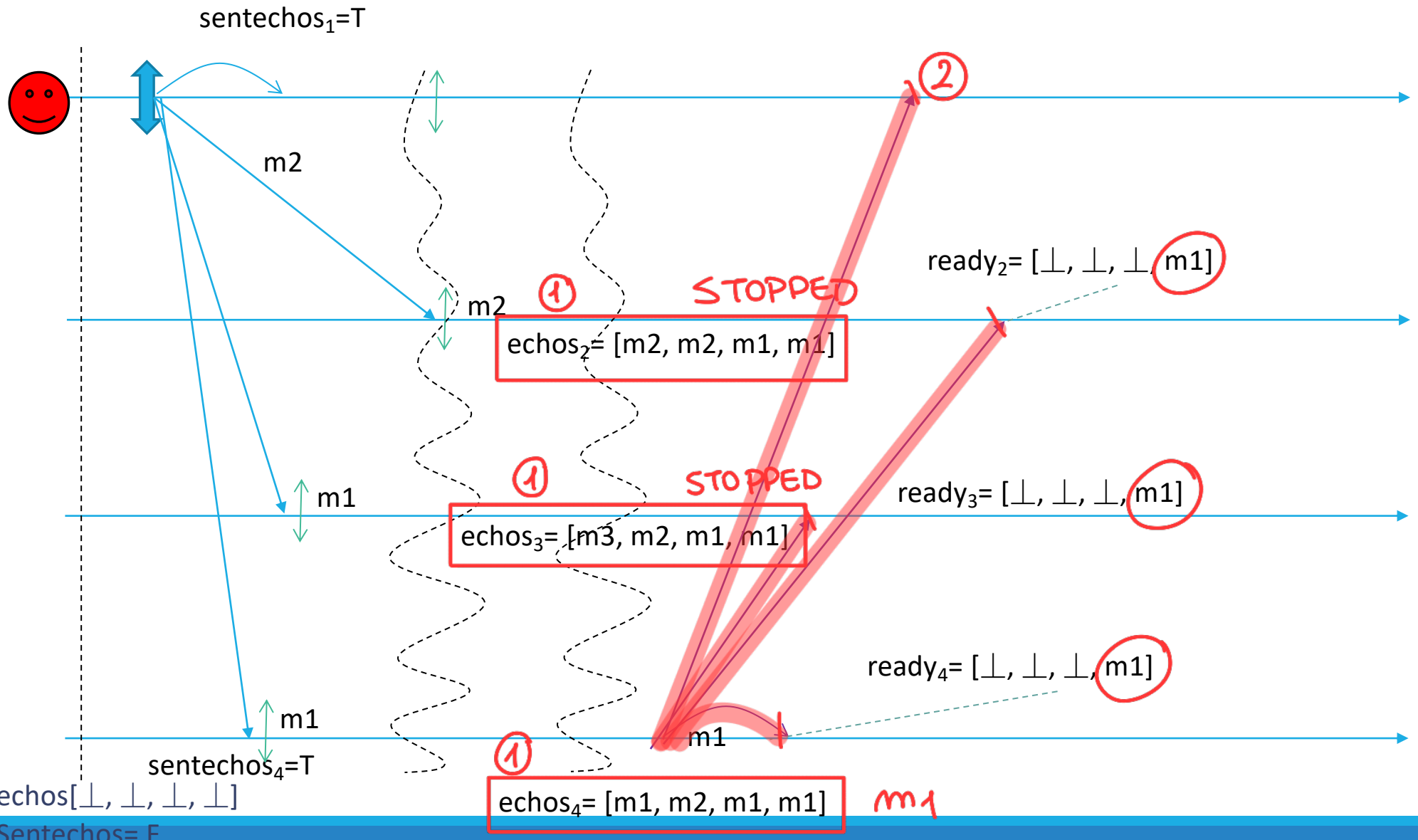
Assumption for Correctness: N > 3f -> f=1, N=4 with p1 acting as sender (correct sender)



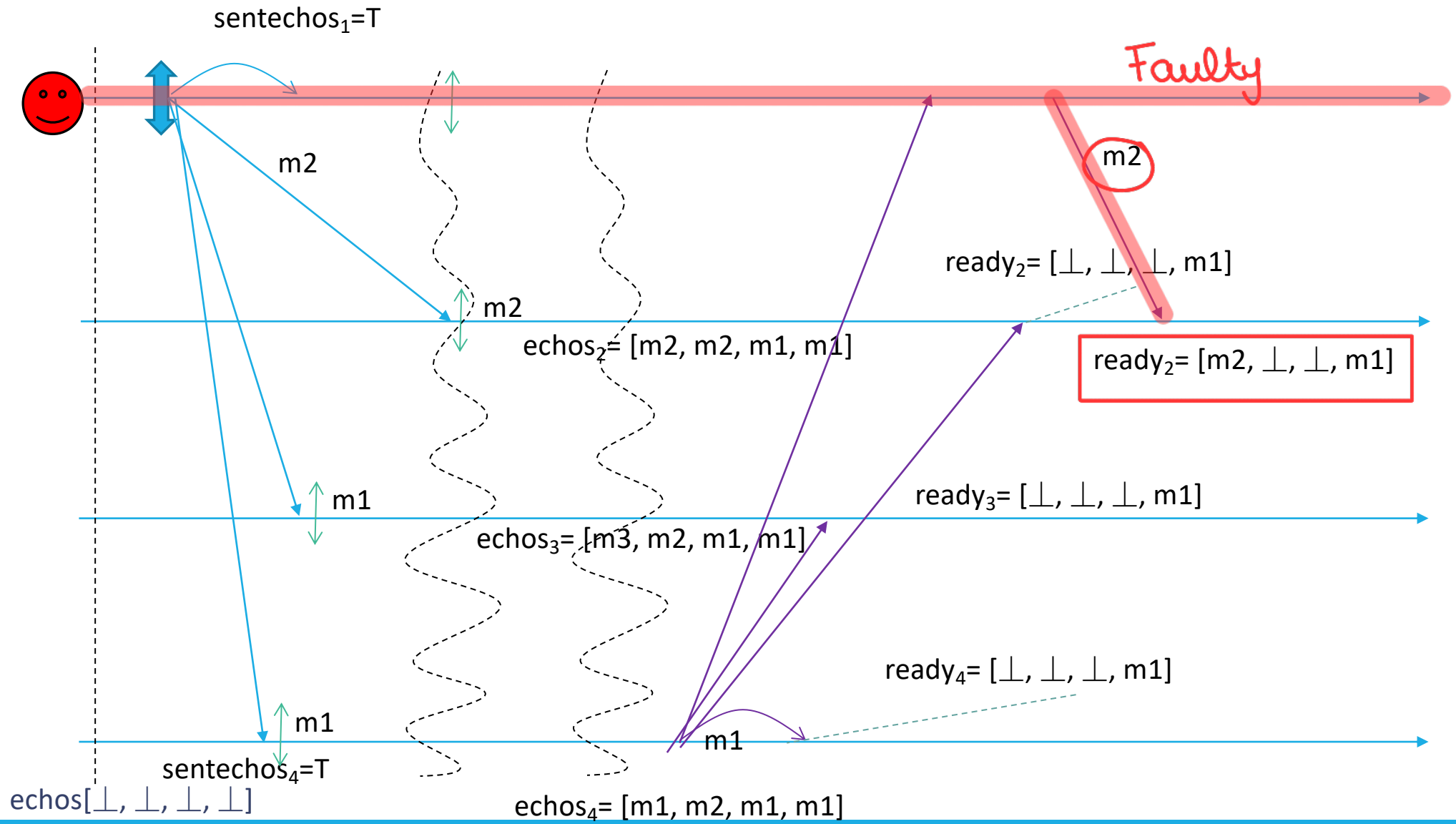
Assumption for Correctness: N > 3f -> f=1, N=4 with p1 acting as sender (faulty sender)



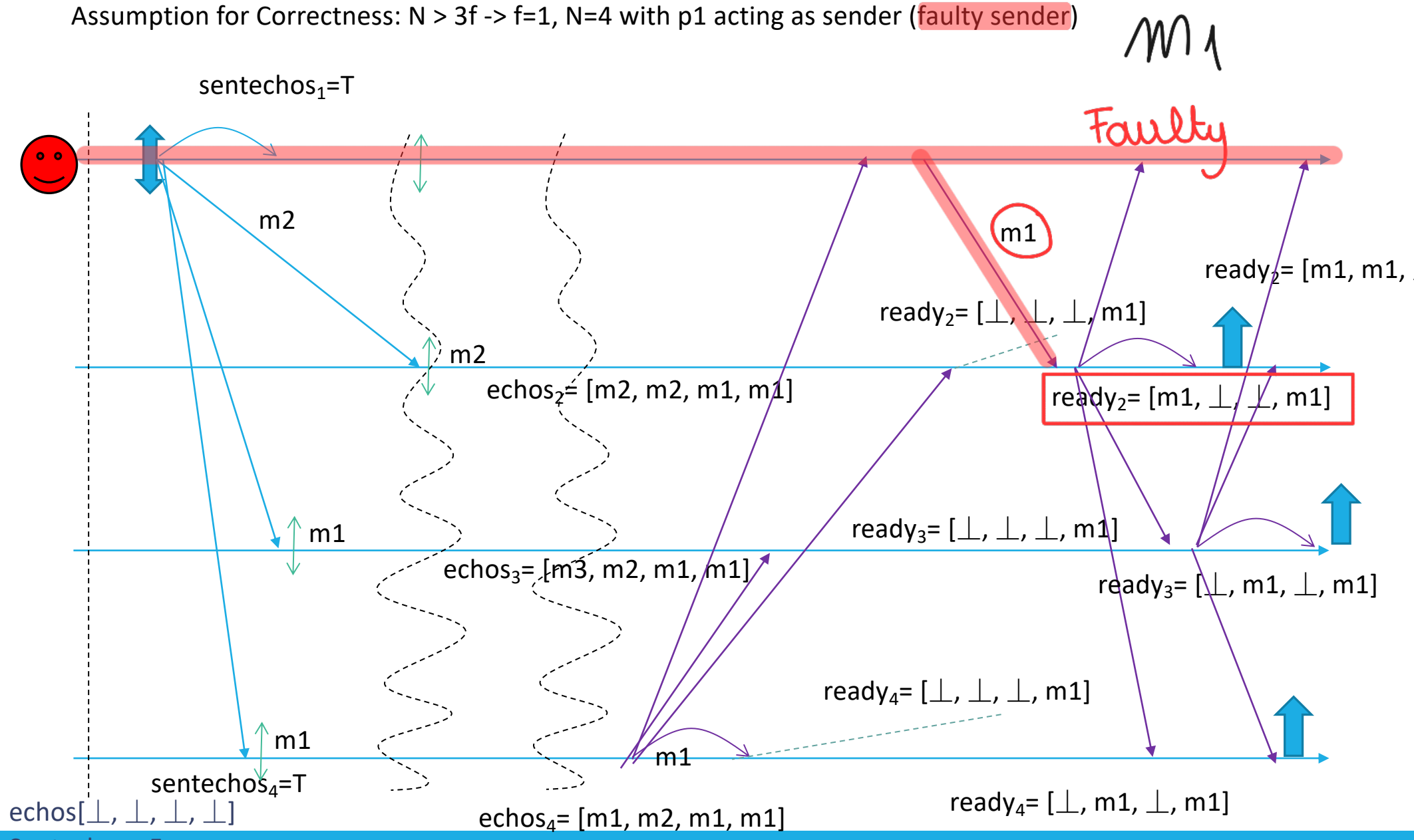
Assumption for Correctness: N > 3f -> f=1, N=4 with p1 acting as sender (faulty sender)



Assumption for Correctness: N > 3f -> f=1, N=4 with p1 acting as sender (faulty sender)



Assumption for Correctness: N > 3f -> f=1, N=4 with p1 acting as sender (faulty sender)



References

C. Cachin, R. Guerraoui and L. Rodrigues. Introduction to Reliable and Secure Distributed Programming, Springer, 2011

- Chapter 2 – Section 2.4.6
- Chapter 3 – Section 3.10 (except 3.10.4), Section 3.11