

# Distributed Systems

## Group Membership and View Synchronous Communication

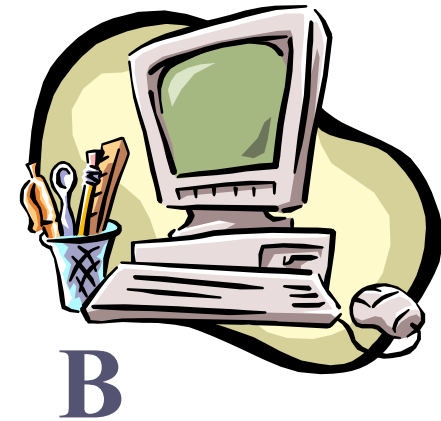
Prof R. Guerraoui

Distributed Programming Laboratory

# Group Membership



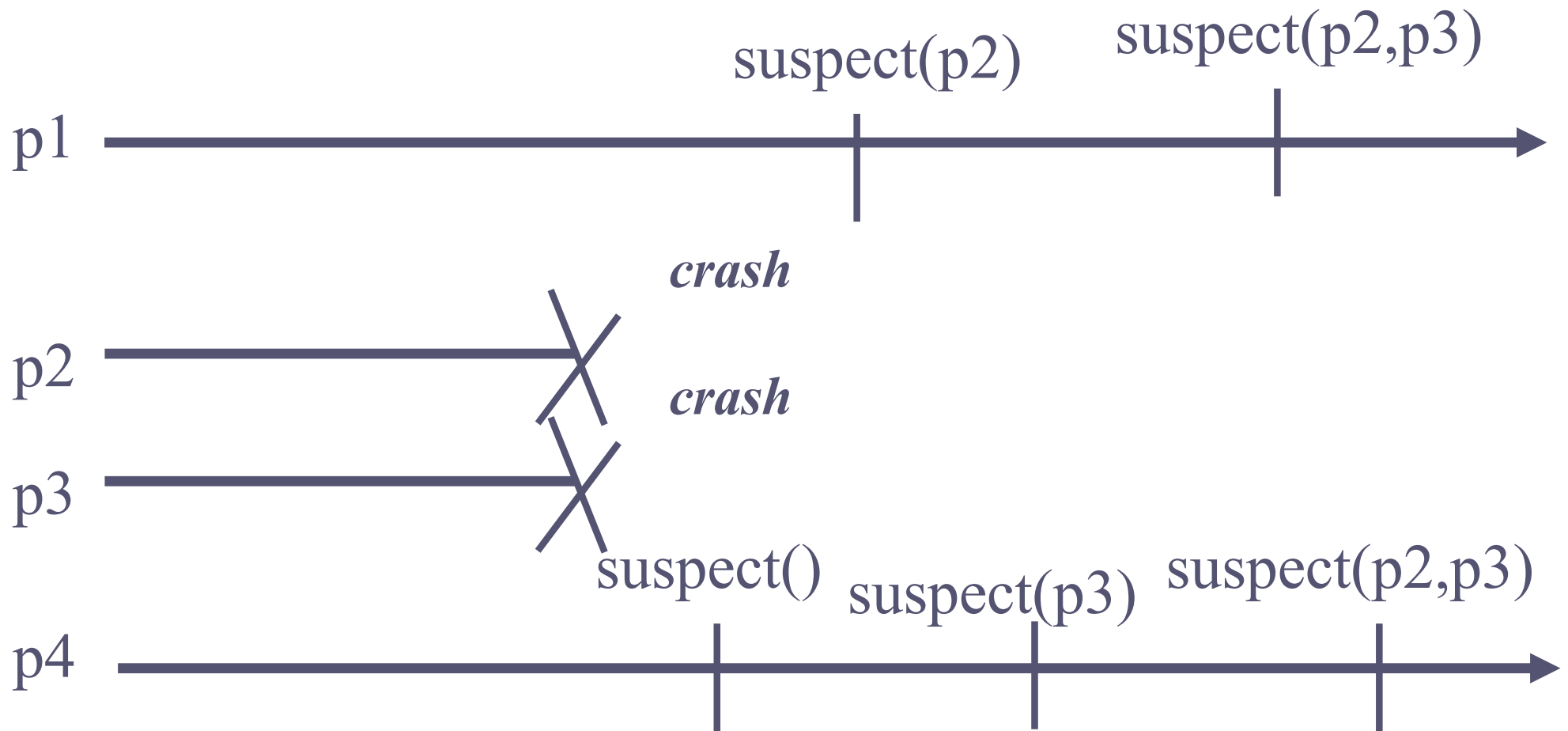
Who is there?



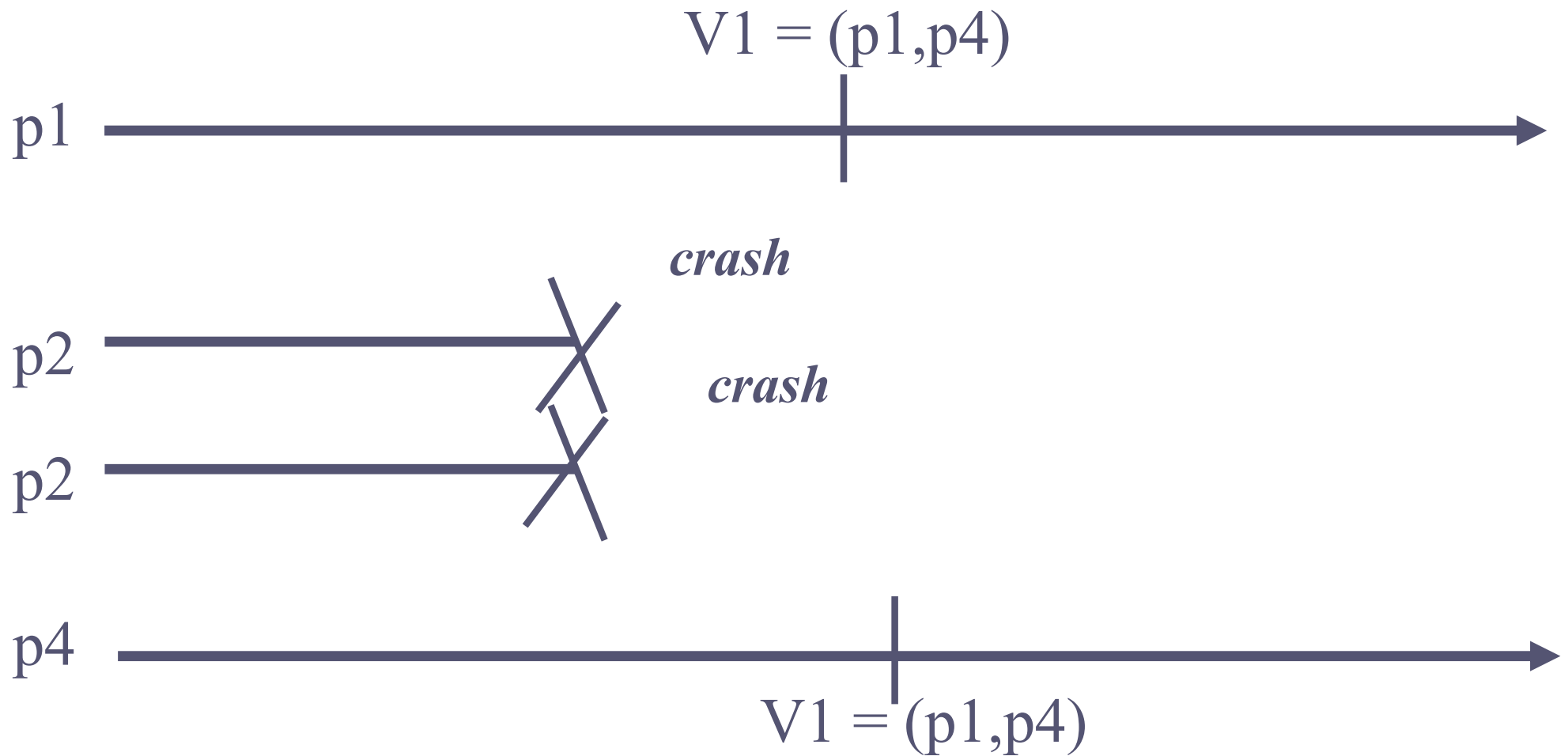
# Group Membership

- In many distributed applications, processes need to know which processes are ***participating*** in the computation and which are not
- Failure detectors provide such information; however, that information is ***not coordinated*** (see next slide) even if the failure detector is perfect

# Perfect Failure Detector



# Group Membership



# Group Membership

- To illustrate the concept, we focus here on a group membership abstraction to coordinate the information about *crashes*
- In general, a group membership abstraction can also typically be used to coordinate the processes *joining* and *leaving* explicitly the set of processes (i.e., without crashes)

# Group Membership

- ***Like*** with a failure detector, the processes are informed about failures; we say that the processes ***install views***
- ***Like*** with a perfect failure detector, the processes have accurate knowledge about failures
- ***Unlike*** with a perfect failure detector, the information about failures are ***coordinated***: the processes install the same sequence of views

# Group Membership

***Memb1. Local Monotonicity:*** If a process installs view  $(j, M)$  after installing  $(k, N)$ , then  $j > k$  and  $M < N$

***Memb2. Agreement:*** No two processes install views  $(j, M)$  and  $(j, M')$  such that  $M \neq M'$

***Memb3. Completeness:*** If a process  $p$  crashes, then there is an integer  $j$  such that every correct process eventually installs view  $(j, M)$  such that  $p \notin M$

***Memb4. Accuracy:*** If some process installs a view  $(i, M)$  and  $p \notin M$ , then  $p$  has crashed



# Group Membership

## • *Events*

• Indication: <membView, V>

## • *Properties:*

• *Memb1, Memb2, Memb3, Memb4*

# Algorithm (gmp)

☞ **Implements:** groupMembership (gmp).

☞ **Uses:**

- PerfectFailureDetector (P).
- UniformConsensus(Ucons).

☞ **upon event** < Init > **do**

- ☞ view := (0,S);
- ☞ correct := S;
- ☞ wait := true;

# Algorithm (gmp – cont'd)

☛ **upon event**  $\langle \text{crash}, p_i \rangle$  **do**

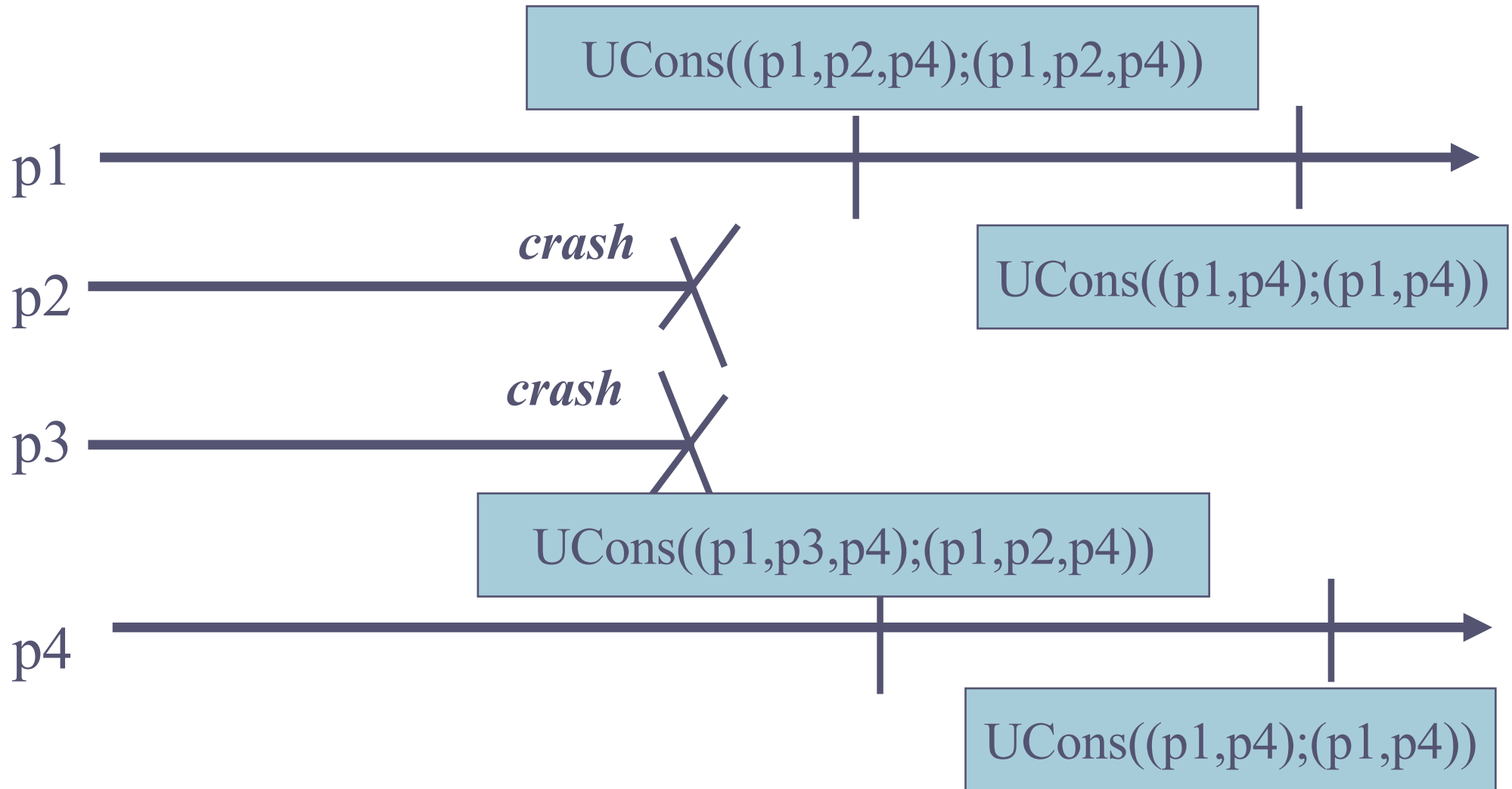
☛  $\text{correct} := \text{correct} \setminus \{p_i\};$

- **upon event**  $(\text{correct} < \text{view.memb})$  and  $(\text{wait} = \text{false})$  **do**
  - $\text{wait} := \text{true};$
  - **trigger** $\langle \text{ucPropose}, (\text{view.id} + 1, \text{correct}) \rangle;$

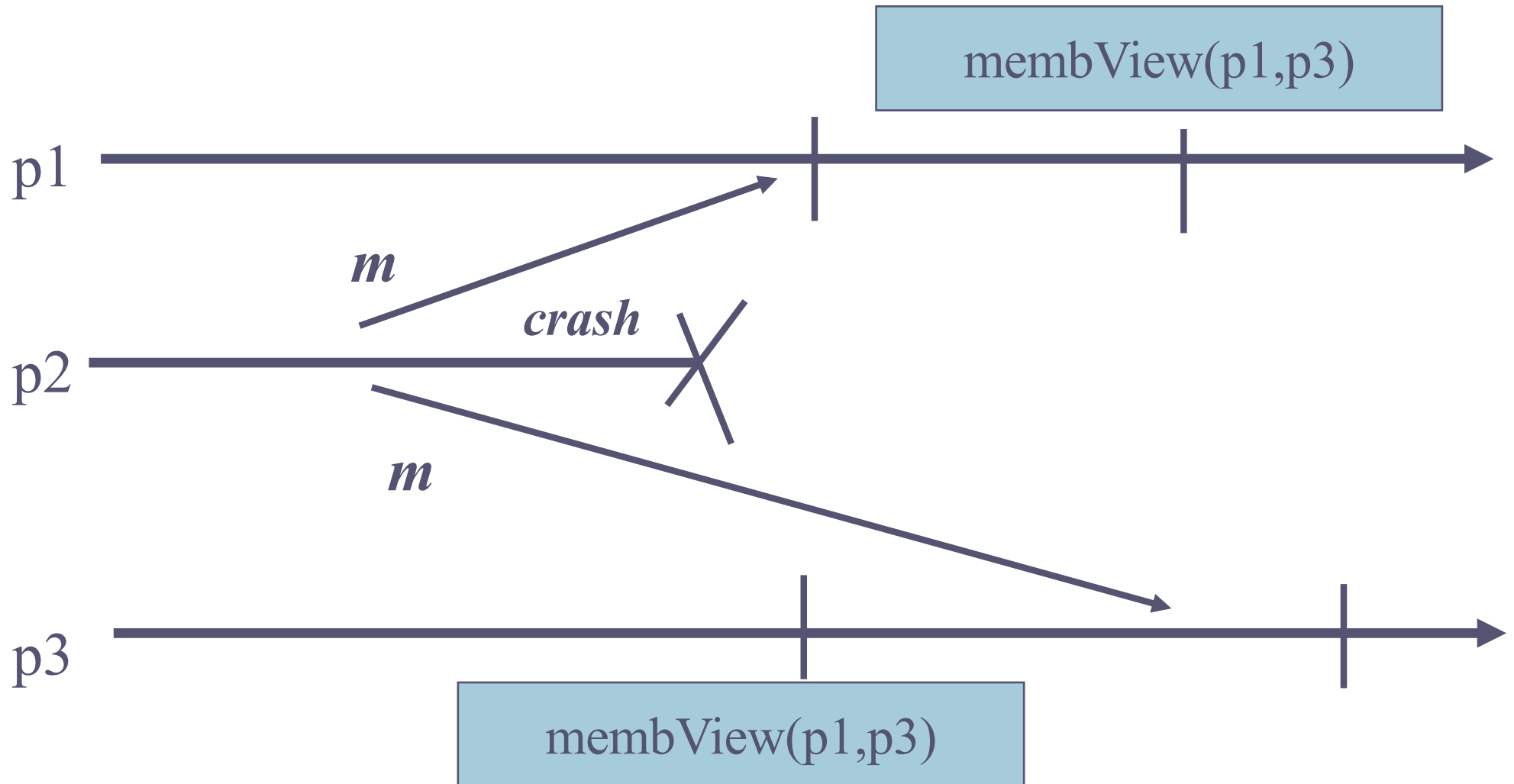
# Algorithm (gmp – cont'd)

- **upon event** < ucDecided, (id, memb)> **do**
  - view := (id, memb);
  - wait := false;
  - **trigger** < membView, view>;

# Algorithm (gmp)



# Group Membership and Broadcast



# View Synchrony

- ***View synchronous broadcast*** is an abstraction that results from the combination of group membership and reliable broadcast
- ***View synchronous broadcast*** ensures that the delivery of messages is coordinated with the installation of views

# View Synchrony

Besides the properties of *group membership* (***Memb1-Memb4***) and *reliable broadcast* (***RB1-RB4***), the following property is ensured:

***VS:*** A message is ***vsDelivered*** in the view where it is ***vsBroadcast***



# View Synchrony

## Events

Request:

<vsBroadcast, m>

• Indication:

• <vsDeliver, src, m>

• <vsView, V>

# View Synchrony

If the application keeps *vsBroadcasting* messages, the *view synchrony* abstraction might never be able to *vsInstall* a new view; the abstraction would be impossible to implement

We introduce a specific event for the abstraction to *block* the application from *vsBroadcasting* messages; this only happens when a process crashes

# View Synchrony

## Events

- Request:

  - $\langle \text{vsBroadcast}, m \rangle; \langle \text{vsBlock}, \text{ok} \rangle$

- Indication:

  - $\langle \text{vsDeliver}, \text{src}, m \rangle; \langle \text{vsView}, V \rangle; \langle \text{vsBlock} \rangle$

# Algorithm (vsc)

- ☛ **Implements:** ViewSynchrony (vs).

- ☛ **Uses:**

  - ☛ GroupMembership (gmp).

  - ☛ TerminatingReliableBroadcast(trb).

  - ☛ BestEffortBroadcast(beb).

# Algorithm (vsc – cont'd)

- ☛ **upon event < Init > do**
  - ☛  $\text{view} := (0, S); \text{nextView} := \perp;$
  - ☛  $\text{pending} := \text{delivered} := \text{trbDone} := \emptyset;$
  - ☛  $\text{flushing} := \text{blocked} := \text{false};$

# Algorithm (vsc – cont'd)

- ☛ **upon event**  $\langle \text{vsBroadcast}, m \rangle$  and  $(\text{blocked} = \text{false})$   
**do**
  - ☛  $\text{delivered} := \text{delivered} \cup \{ m \};$
  - ☛ **trigger**  $\langle \text{vsDeliver}, \text{self}, m \rangle;$
  - ☛ **trigger**  $\langle \text{bebBroadcast}, [\text{Data}, \text{view.id}, m] \rangle;$

# Algorithm (vsc – cont'd)

- ☛ **upon event** <bebDeliver,src,[Data,vid,m]> **do**
  - ☛ If(view.id = vid) and ( $m \notin \text{delivered}$ ) and (blocked = false) then
    - ☛  $\text{delivered} := \text{delivered} \cup \{ m \}$
  - ☛ **trigger** <vsDeliver, src, m >;

# Algorithm (vsc – cont'd)

- ☛ **upon event** < membView, V > **do**
  - addtoTail (pending, V);
- ☛ **upon** (pending  $\neq \emptyset$ ) and (flushing = false) **do**
  - ☛ nextView := removeFromhead (pending);
  - ☛ flushing := true;
  - ☛ **trigger** <vsBlock>;



# Algorithm (vsc – cont'd)

- ☛ **Upon** <vsBlockOk> **do**
  - ☛ blocked := true;
  - ☛ trbDone :=  $\emptyset$ ;
  - ☛ **trigger** <trbBroadcast, self, (view.id,delivered)>;

# Algorithm (vsc – cont'd)

- ☛ **Upon**  $\langle \text{trbDeliver}, p, (\text{vid}, \text{del}) \rangle$  **do**
  - ☛  $\text{trbDone} := \text{trbDone} \cup \{p\};$
  - ☛ **forall**  $m \in \text{del}$  and  $m \notin \text{delivered}$  **do**
    - ☛  $\text{delivered} := \text{delivered} \cup \{m\};$
    - ☛ **trigger**  $\langle \text{vsDeliver}, \text{src}, m \rangle;$

# Algorithm (vsc – cont'd)

- ☛ **Upon** (trbDone = view.memb) and (blocked = true) **do**
  - ☛ view := nextView;
  - ☛ flushing := blocked := false;
  - ☛ delivered :=  $\emptyset$ ;
  - ☛ **trigger** <vsView, view>;

# Consensus-Based View Synchrony

Instead of launching parallel instances of TRBs, plus a group membership, we use one consensus instance and parallel broadcasts for every view change

Roughly, the processes exchange the messages they have delivered when they detect a failure, and use consensus to agree on the membership and the message set

# Algorithm 2 (vsc)

- ☛ **Implements:** ViewSynchrony (vs).

- ☛ **Uses:**

  - ☛ UniformConsensus (uc).

  - ☛ BestEffortBroadcast(beb).

  - ☛ PerfectFailureDetector(P).

# Algorithm 2 (vsc – cont'd)

- ☛ **upon event** < Init > **do**
  - ☛ view := (0,S);
  - ☛ correct := S;
  - ☛ flushing := blocked := false;
  - ☛ delivered := dset :=  $\emptyset$ ;

## Algorithm 2 (vsc – cont'd)

- ▮ **upon event** <vsBroadcast,m) and (blocked = false) **do**
  - ▮ delivered := delivered  $\cup$  { m }
  - ▮ **trigger** <vsDeliver, self,m>;
  - ▮ **trigger** <bebBroadcast,[Data,view.id,m] >;

## Algorithm 2 (vsc – cont'd)

```
☛ upon event <bebDeliver,src,[Data,vid,m]> do  
  ☛ if (view.id = vid) and (m  $\notin$  delivered) and (blocked  
    = false) then  
    ☛ delivered := delivered  $\cup$  { m };  
    ☛ trigger <vsDeliver, src, m >;
```



# Algorithm 2 (vsc – cont'd)

☛ **upon event**  $\langle \text{crash}, p \rangle$  **do**

- $\text{correct} := \text{correct} \setminus \{ p \};$
- **if** flushing = false **then**
  - flushing := true;
  - **trigger**  $\langle \text{vsBlock} \rangle;$

# Algorithm 2 (vsc – cont'd)

☛ **Upon** <vsBlockOk> **do**

☛ blocked := true;

☛ **trigger** <bebBroadcast, [DSET,view.id,delivered] >;

# Algorithm 2 (vsc – cont'd)

- ☛ **Upon**  $\langle \text{bebDeliver}, \text{src}, [\text{DSET}, \text{vid}, \text{del}] \rangle$  **do**
  - ☛  $\text{dset} := \text{dset} \cup (\text{src}, \text{del});$
  - ☛ **if forall**  $p \in \text{correct}, (p, \text{mset}) \in \text{dset}$  **then**  
**trigger**  $\langle \text{ucPropose}, \text{view.id} + 1, \text{correct}, \text{dset} \rangle;$

## Algorithm 2 (vsc – cont'd)

- ☛ **Upon**  $\langle \text{ucDecided}, \text{id}, \text{memb}, \text{vsdset} \rangle$  **do**
  - ☛ **forall**  $(p, \text{mset}) \in \text{vsdset}: p \in \text{memb}$  **do**
    - ☛ **forall**  $(\text{src}, m) \in \text{mset}: m \notin \text{delivered}$  **do**
      - ☛  $\text{delivered} := \text{delivered} \cup \{m\}$
      - ☛ **trigger**  $\langle \text{vsDeliver}, \text{src}, m \rangle;$
  - ☛  $\text{view} := (\text{id}, \text{memb}); \text{flushing} := \text{blocked} := \text{false}; \text{dset} := \text{delivered} := \emptyset;$
  - ☛ **trigger**  $\langle \text{vsView}, \text{view} \rangle;$

# Uniform View Synchrony

We now combine the properties of

***group membership (Memb1-Memb4)*** –  
which is already uniform

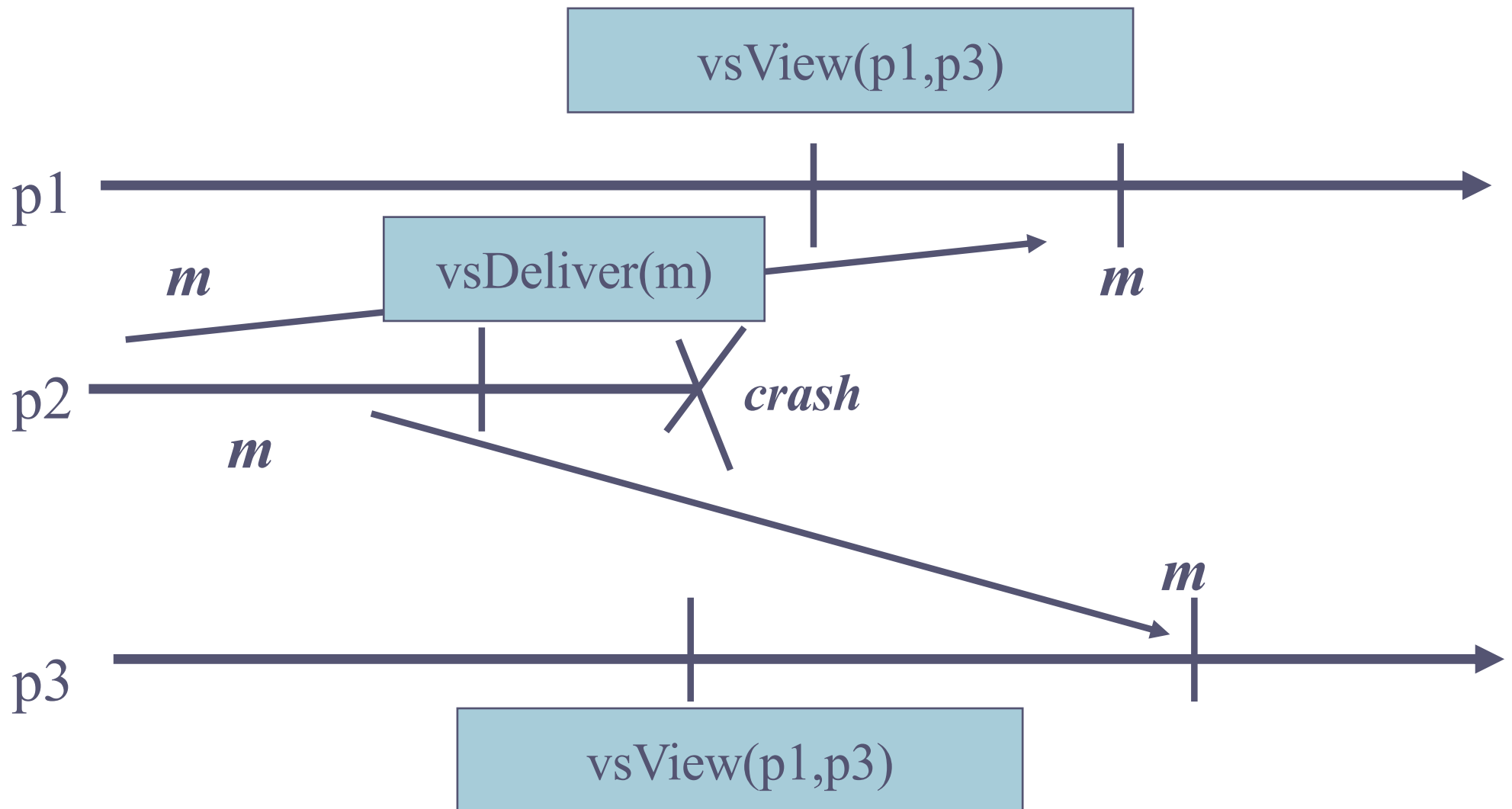
***uniform reliable broadcast (RB1-RB4)*** –

***VS:*** A message is ***vsDelivered*** in the view  
where it is ***vsBroadcast*** – which is already  
uniform

# Uniform View Synchrony

Using uniform reliable broadcast instead of best effort broadcast in the previous algorithms does not ensure the uniformity of the message delivery

# Uniformity?



# Algorithm 3 (uvsc)

- ☛ **upon event** < Init > **do**
  - ☛  $\text{view} := (0, S);$
  - ☛  $\text{correct} := S;$
  - ☛  $\text{flushing} := \text{blocked} := \text{false};$
  - ☛  $\text{undelivered} := \text{delivered} := \text{dset} := \emptyset;$
  - ☛ for all  $m$ :  $\text{ack}(m) := \emptyset;$



# Algorithm 3 (uvsc – cont'd)

- ☛ **upon event**  $\langle \text{vsBroadcast}, m \rangle$  and  $(\text{blocked} = \text{false})$   
**do**
  - ☛  $\text{delivered} := \text{delivered} \cup \{m\};$
  - ☛ **trigger**  $\langle \text{bebBroadcast}, [\text{Data}, \text{view.id}, m] \rangle;$

# Algorithm 3 (uvsc – cont'd)

```
☛ upon event <bebDeliver,src,[Data,vid,m]> do  
  ☛ if (view.id = vid) then  
    ☛  $\text{ack}(m) := \text{ack}(m) \cup \{\text{src}\};$   
  ☛ if  $m \notin \text{delivered}$  then  
    ☛  $\text{delivered} := \text{delivered} \cup \{m\}$   
  ☛ trigger <bebBroadcast, [Data,view.id,m] >;
```

# Algorithm 3 (uvsc – cont'd)

- ☛ **upon event** ( $\text{view} \leq \text{ack}(m)$ ) and ( $m \notin \text{udelivered}$ )  
**do**
  - ☛  $\text{udelivered} := \text{udelivered} \cup \{ m \}$
  - ☛ **trigger**  $\langle \text{vsDeliver}, \text{src}(m), m \rangle$ ;

# Algorithm 3 (uvsc – cont'd)

☛ **upon event** < crash, p > **do**

- $\text{correct} := \text{correct} \setminus \{ p \};$

- **if** flushing = false **then**

  - flushing := true;

  - **trigger** <vsBlock>;

# Algorithm 3 (uvsc – cont'd)

- ☛ **Upon** <vsBlockOk> **do**
  - ☛ blocked := true;
  - ☛ **trigger** <bebBroadcast, [DSET,view.id,delivered] >;
- ☛ **Upon** <bebDeliver, src, [DSET,vid,del] > **do**
  - ☛ dset := dset  $\cup$  (src,del);
  - ☛ **if forall**  $p \in \text{correct}, (p, \text{mset}) \in \text{dset}$   
**then trigger** <ucPropose, view.id+1, correct, dset >;

# Algorithm 3 (uvsc – cont'd)

- ☛ **Upon**  $\langle \text{ucDecided}, \text{id}, \text{memb}, \text{vsdset} \rangle$  **do**
  - ☛ **forall**  $(p, \text{mset}) \in \text{vs-dset}: p \in \text{memb}$  **do**
    - ☛ **forall**  $(\text{src}, m) \in \text{mset}: m \notin \text{udelivered}$  **do**
      - ☛  $\text{udelivered} := \text{udelivered} \cup \{m\}$
      - ☛ **trigger**  $\langle \text{vsDeliver}, \text{src}, m \rangle;$
- ☛  $\text{view} := (\text{id}, \text{memb}); \text{flushing} := \text{blocked} := \text{false}; \text{dset} := \text{delivered} := \text{udelivered} := \emptyset;$
- ☛ **trigger**  $\langle \text{vsView}, \text{view} \rangle;$