Agreement in Fault Tolerant Distributed Systems

Rui Carlos Oliveira
HASLab
Universidade do Minho
(rco@di.uminho.pt)

Distributed Agreement

Distributed Agreement

- Agreement problems are at the core of any distributed fault-tolerant system

Distributed Agreement

- Agreement problems are at the core of any distributed fault-tolerant system
- Agreement requirements can be more or less stringent: one may need to agree on a unique leader, whether to commit or abort a distributed transaction, on the delivery order for a set of messages, etc.

Distributed Agreement

- Agreement problems are at the core of any distributed fault-tolerant system
- Agreement requirements can be more or less stringent: one may need to agree on a unique leader, whether to commit or abort a distributed transaction, on the delivery order for a set of messages, etc.
- While on a fault-free system agreement can be easily reached, in the presence of faults and depending on the assumed model, reaching agreement can be very hard or even impossible.

Forms of Agreement

Forms of Agreement

- Non-blocking Atomic Commitment

[Jim Gray, Notes on Database Operating Systems, LNCS 60, 1978]

[D. Skeen, NonBlocking Commit Protocols, 1981]

- Leader Election

[L.Sabel & K. Marzullo, Election Vs. Consensus in Asynchronous Systems, 1995]

- Consensus

[M. Fischer, N. Lynch, M. Paterson. Impossibility of Distributed Consensus with One Faulty Process, 1985]

- k-Set Agreement

[S. Chaudhuri, More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems, 1993]

- Atomic Broadcast, Group Membership, ...

Non-Blocking Atomic Commitment

Non-Blocking Atomic Commitment

- Consider the problem of atomically commiting on the outcome of a distributed transaction. All processes express their local success or failure on executing a transaction voting yes or no

Non-Blocking Atomic Commitment

- Consider the problem of atomically commiting on the outcome of a distributed transaction. All processes express their local success or failure on executing a transaction voting yes or no

Termination: Every correct process eventualy decides

Non-Blocking Atomic Commitment

- Consider the problem of atomically commiting on the outcome of a distributed transaction. All processes express their local success or failure on executing a transaction voting yes or no

Termination: Every correct process eventualy decides

Non-triviality: If all processes vote yes and there is no failure then commit should be decided

Non-Blocking Atomic Commitment

- Consider the problem of atomically commiting on the outcome of a distributed transaction. All processes express their local success or failure on executing a transaction voting yes or no

Termination: Every correct process eventualy decides

Non-triviality: If all processes vote yes and there is no failure then commit should be decided

Validity: If any process votes no then abort should be decided

Non-Blocking Atomic Commitment

- Consider the problem of atomically commiting on the outcome of a distributed transaction. All processes express their local success or failure on executing a transaction voting yes or no

Termination: Every correct process eventualy decides

Non-triviality: If all processes vote yes and there is no failure then commit should be decided

Validity: If any process votes no then abort should be decided

Uniform Agreement: No process decides differently

[P. A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery, 1987]

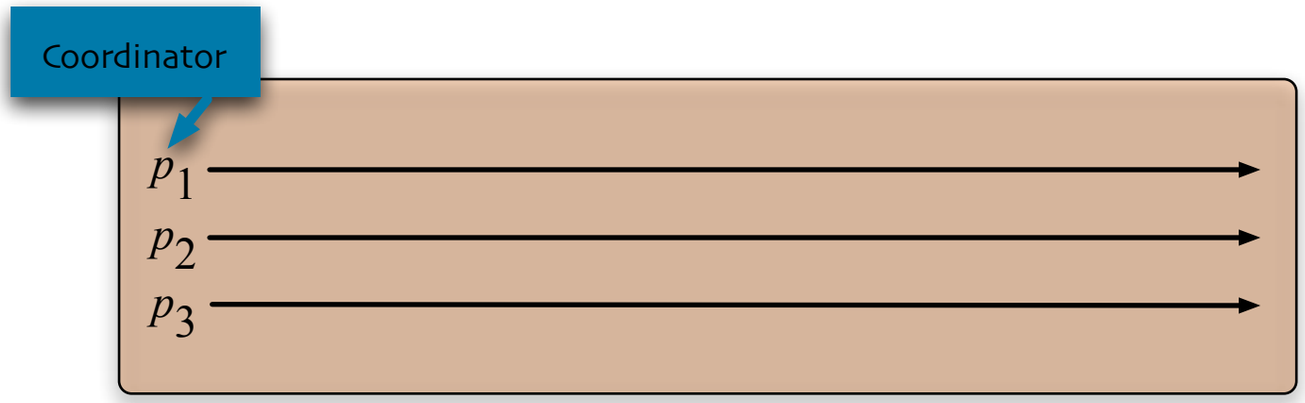
Non-Blocking Atomic Commitment

Non-Blocking Atomic Commitment

- Consider the 2-phase-commit protocol:

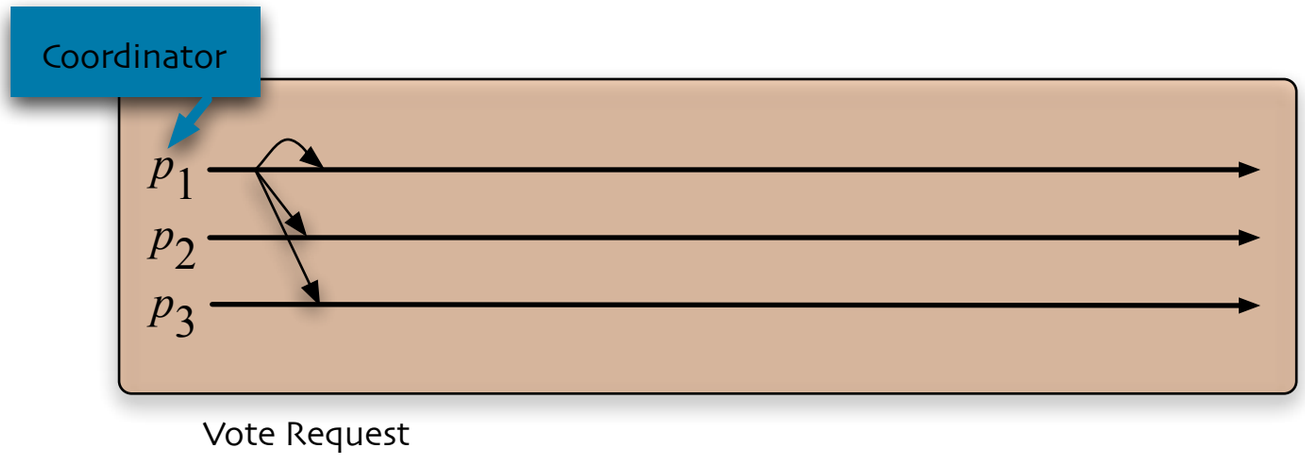
Non-Blocking Atomic Commitment

- Consider the 2-phase-commit protocol:



Non-Blocking Atomic Commitment

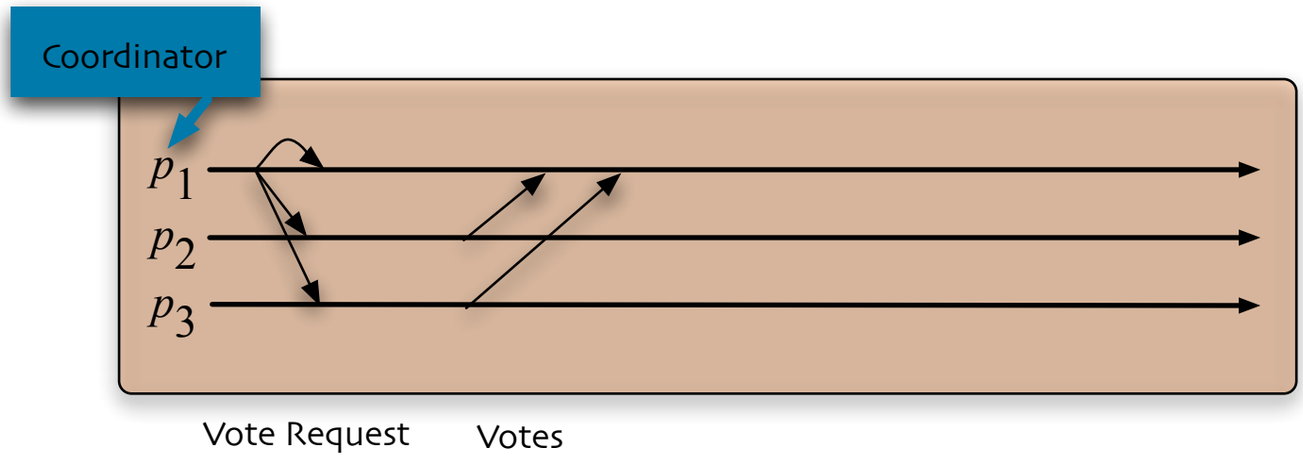
- Consider the 2-phase-commit protocol:



- Coordinator requests votes

Non-Blocking Atomic Commitment

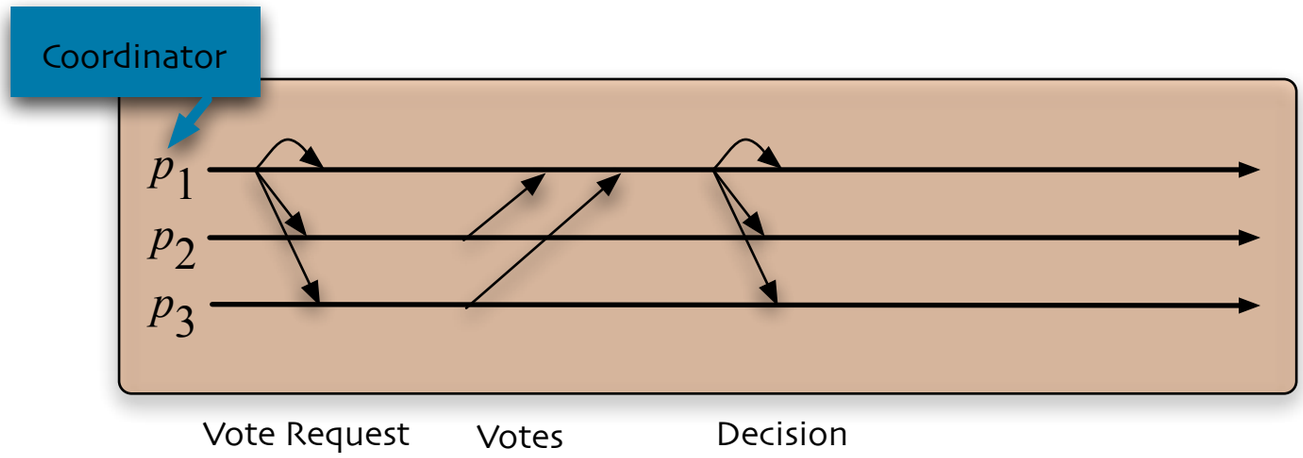
- Consider the 2-phase-commit protocol:



- Coordinator requests votes
- Others vote

Non-Blocking Atomic Commitment

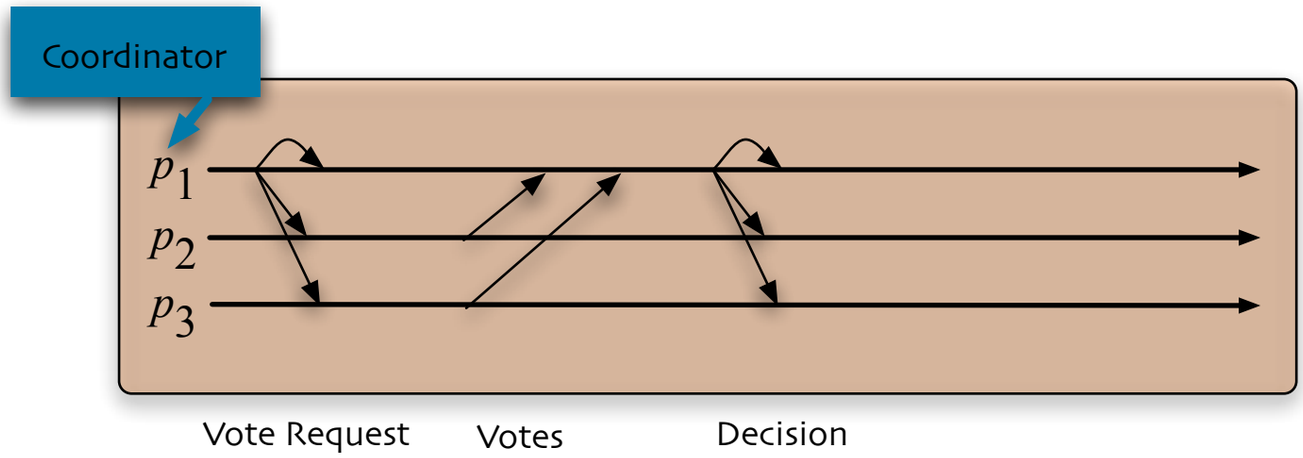
- Consider the 2-phase-commit protocol:



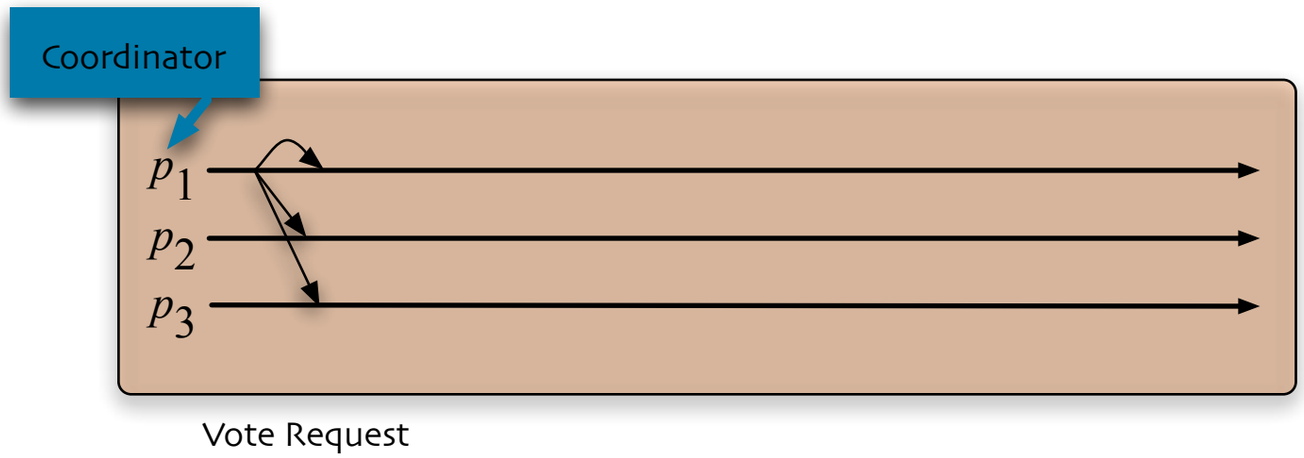
- Coordinator requests votes
- Others vote
- Coordinator decides

Non-Blocking Atomic Commitment

- Consider the 2-phase-commit protocol:

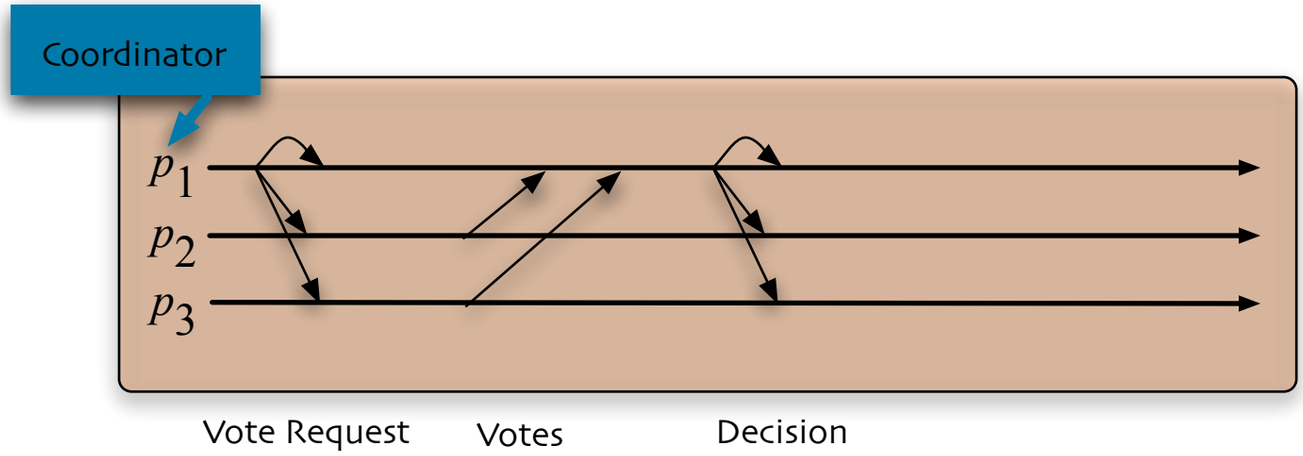


- Coordinator requests votes
- Others vote
- Coordinator decides

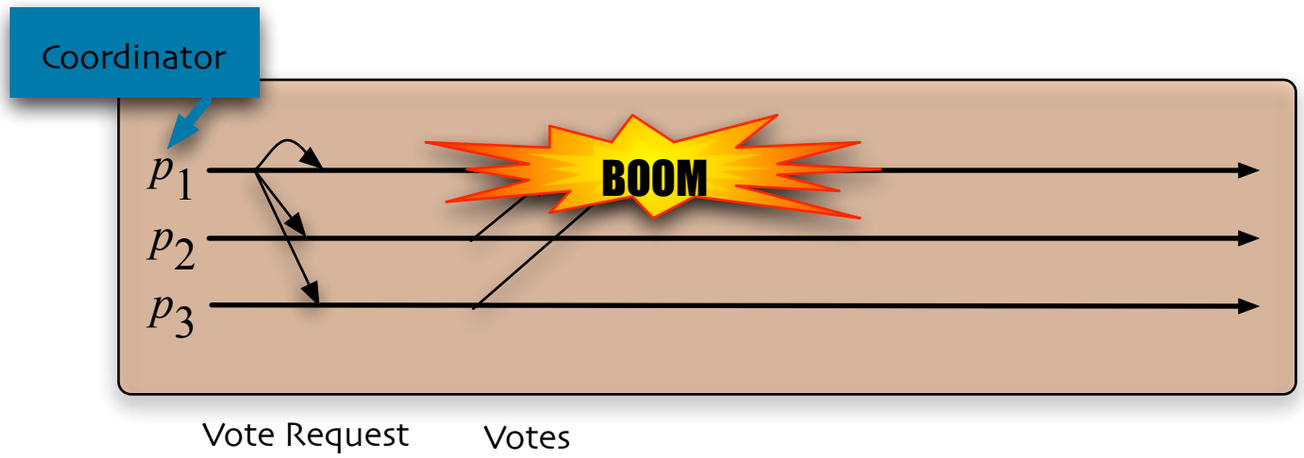


Non-Blocking Atomic Commitment

- Consider the 2-phase-commit protocol:



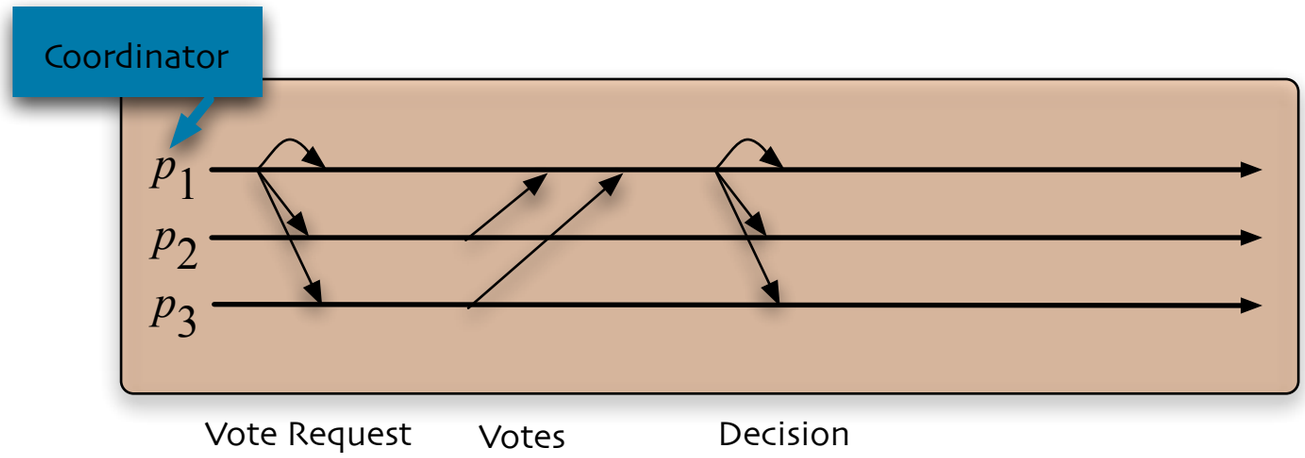
- Coordinator requests votes
- Others vote
- Coordinator decides



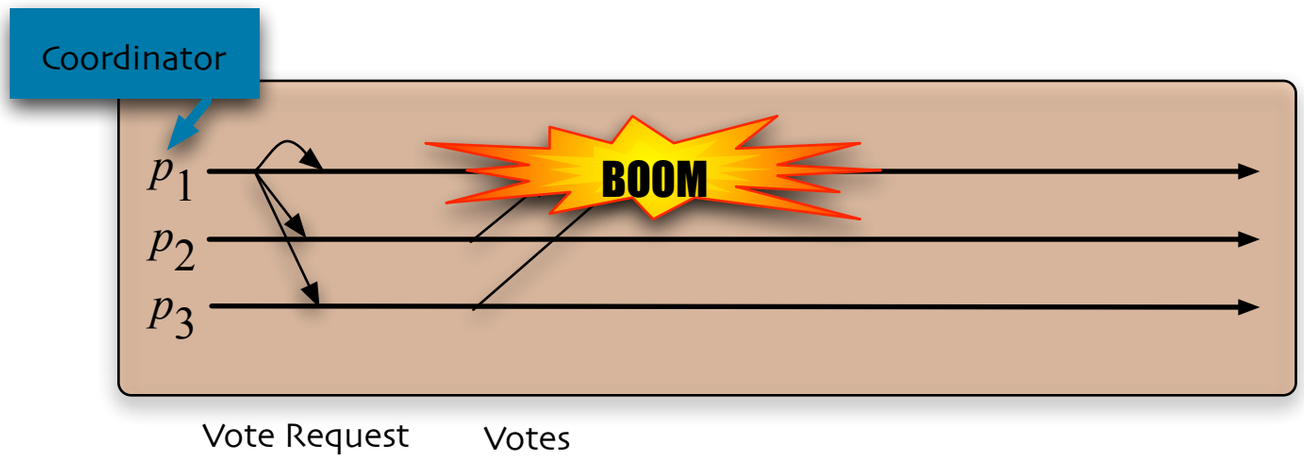
- What if coordinator fails?

Non-Blocking Atomic Commitment

- Consider the 2-phase-commit protocol:



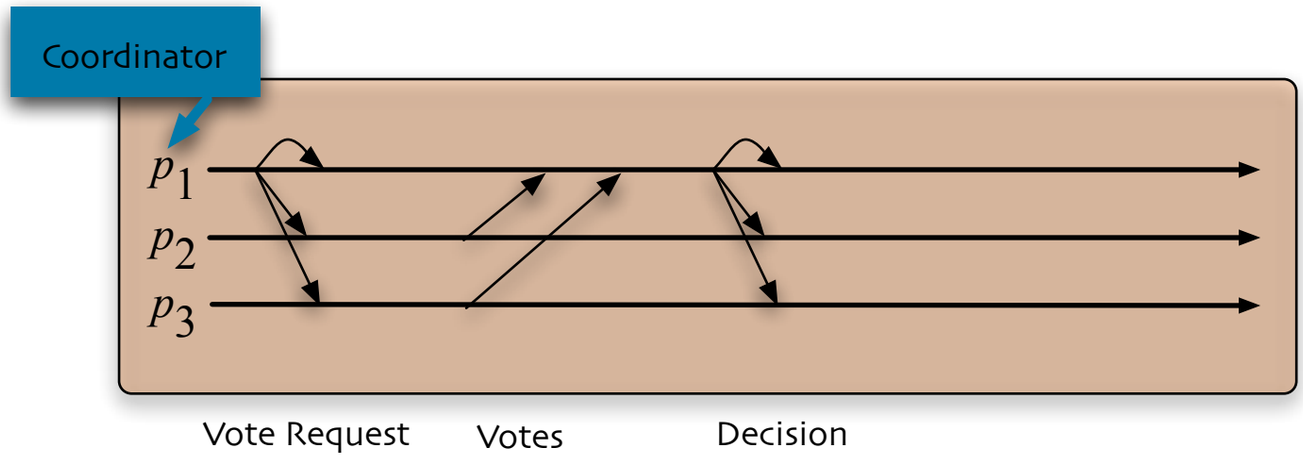
- Coordinator requests votes
- Others vote
- Coordinator decides



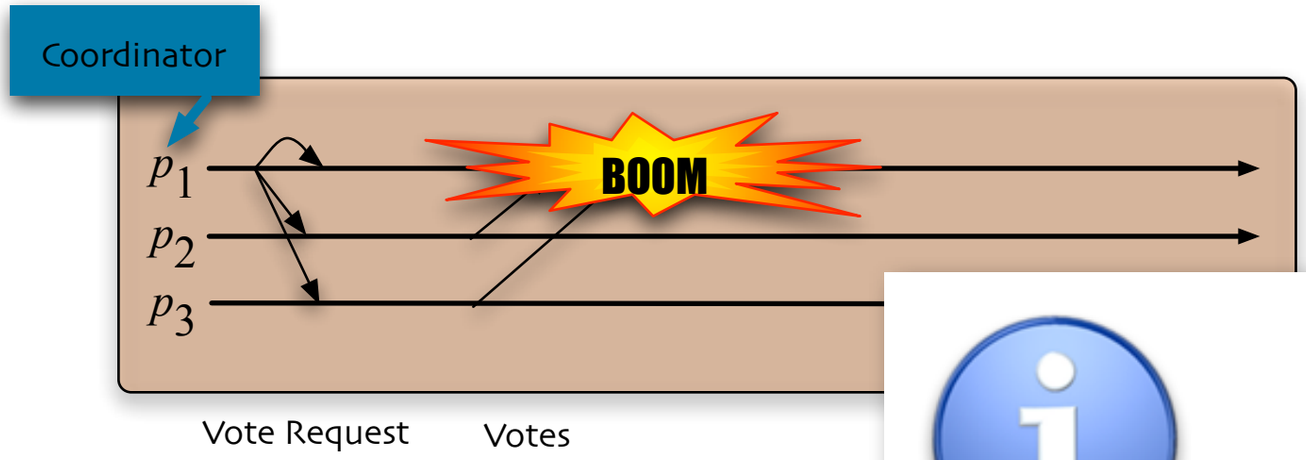
- What if coordinator fails?
- Did it decide? On what?

Non-Blocking Atomic Commitment

- Consider the 2-phase-commit protocol:



- Coordinator requests votes
- Others vote
- Coordinator decides

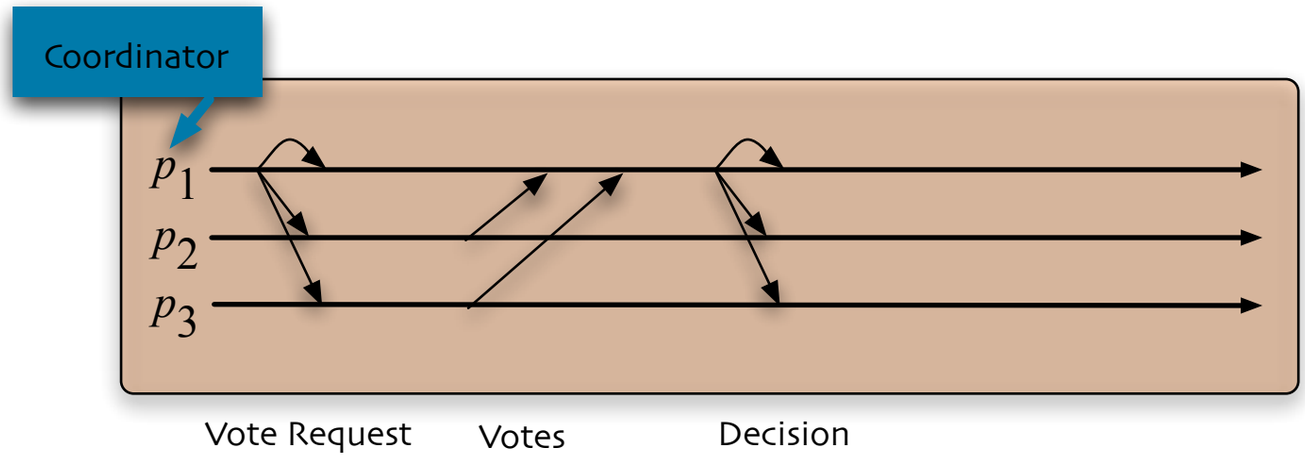


- What if coordinator fails?
- Did it decide? On what?

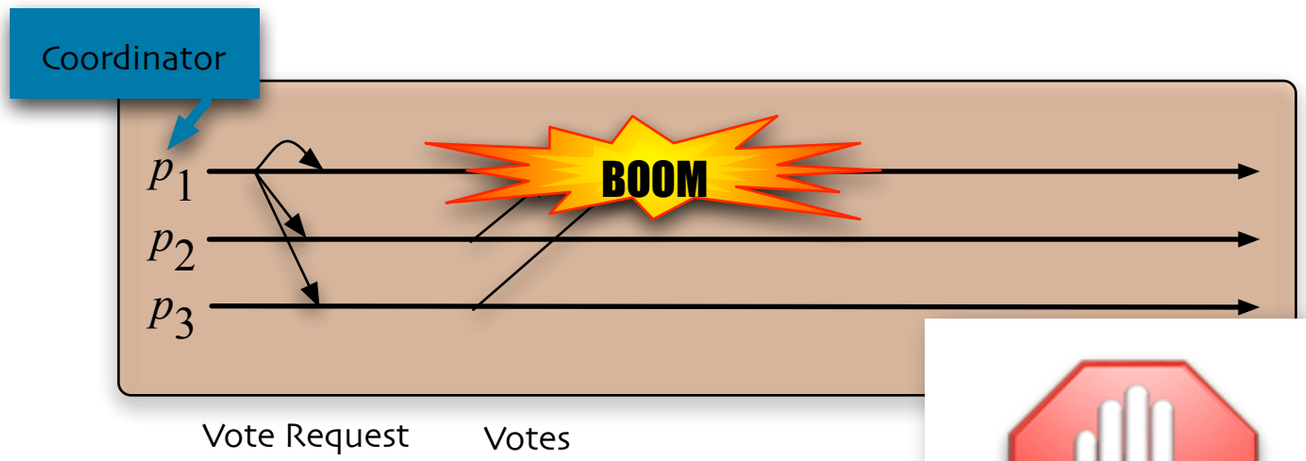
OK
in Crash-Recover model

Non-Blocking Atomic Commitment

- Consider the 2-phase-commit protocol:



- Coordinator requests votes
- Others vote
- Coordinator decides



- What if coordinator fails?
- Did it decide? On what?

Blocking
in Fail-Stop model

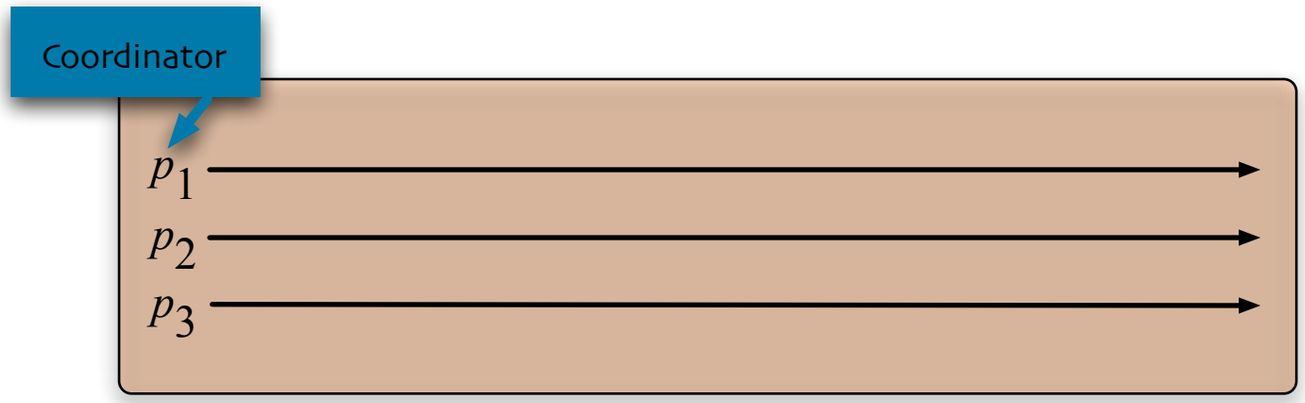
Non-Blocking Atomic Commitment

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:

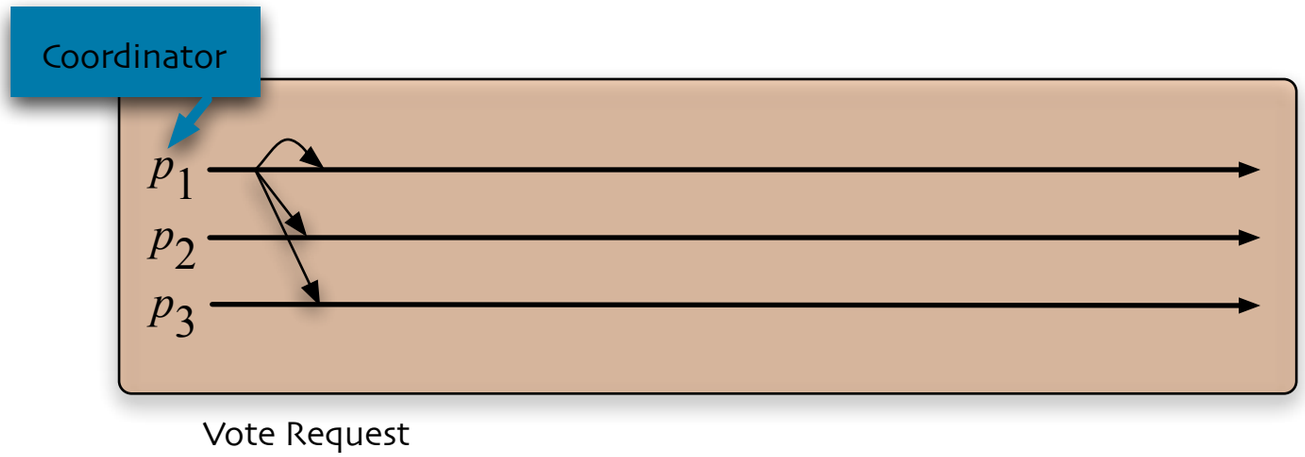
Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



Non-Blocking Atomic Commitment

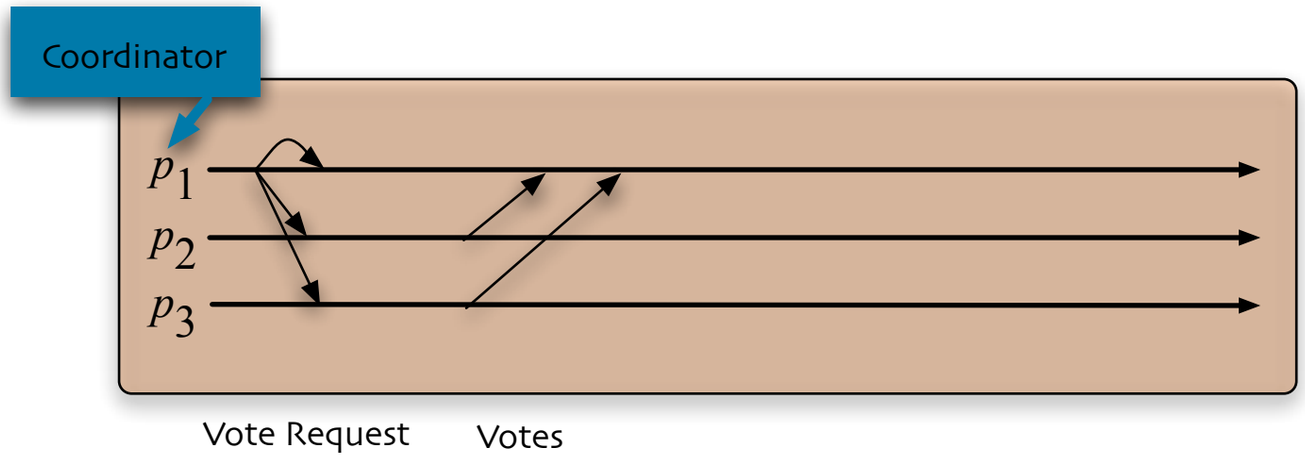
- Consider the 3-phase-commit protocol:



- Coordinator requests votes

Non-Blocking Atomic Commitment

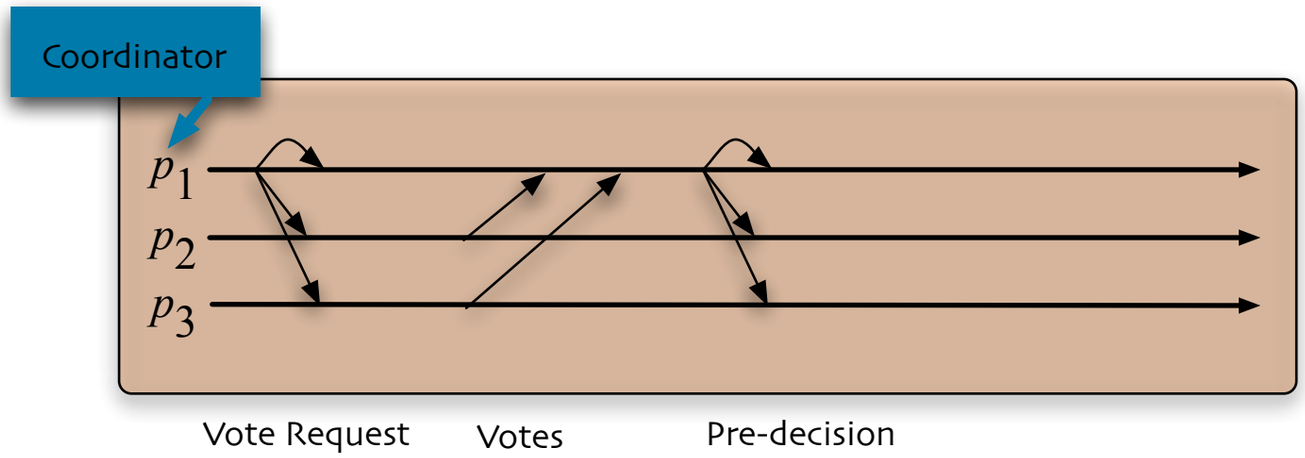
- Consider the 3-phase-commit protocol:



- Coordinator requests votes
- Others vote

Non-Blocking Atomic Commitment

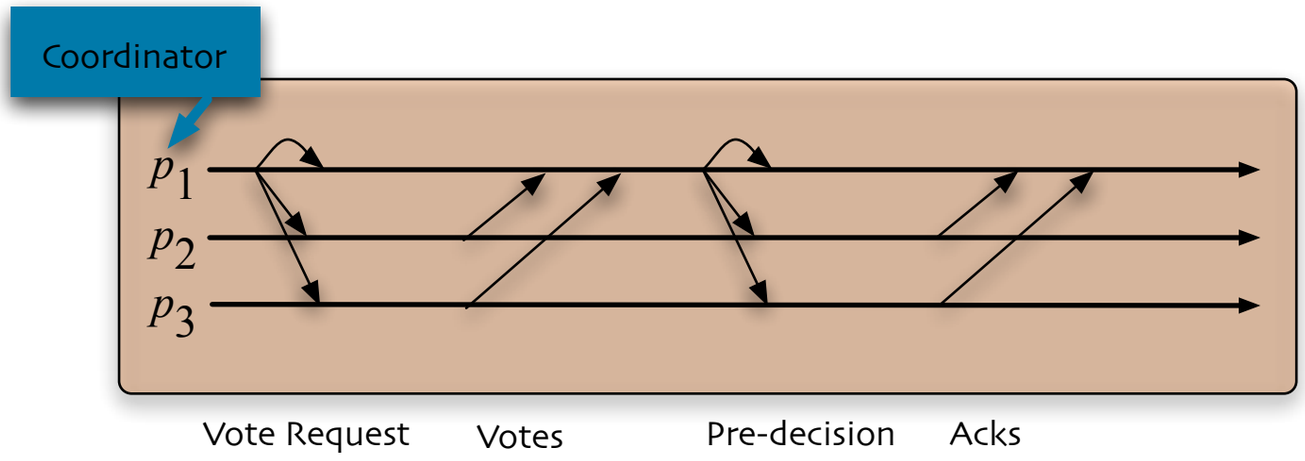
- Consider the 3-phase-commit protocol:



- Coordinator requests votes
- Others vote
- Coordinator pre-decides

Non-Blocking Atomic Commitment

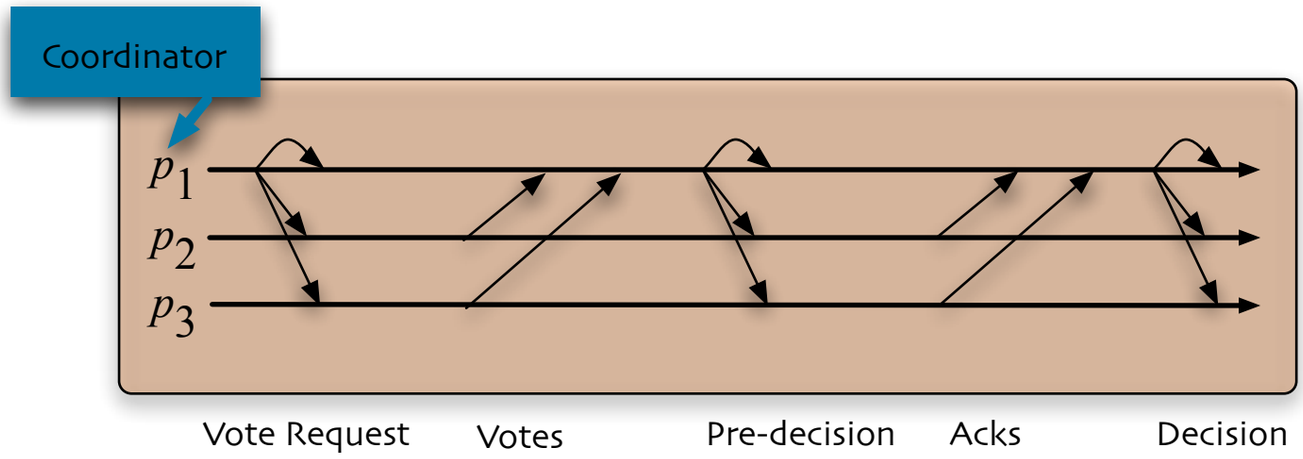
- Consider the 3-phase-commit protocol:



- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge

Non-Blocking Atomic Commitment

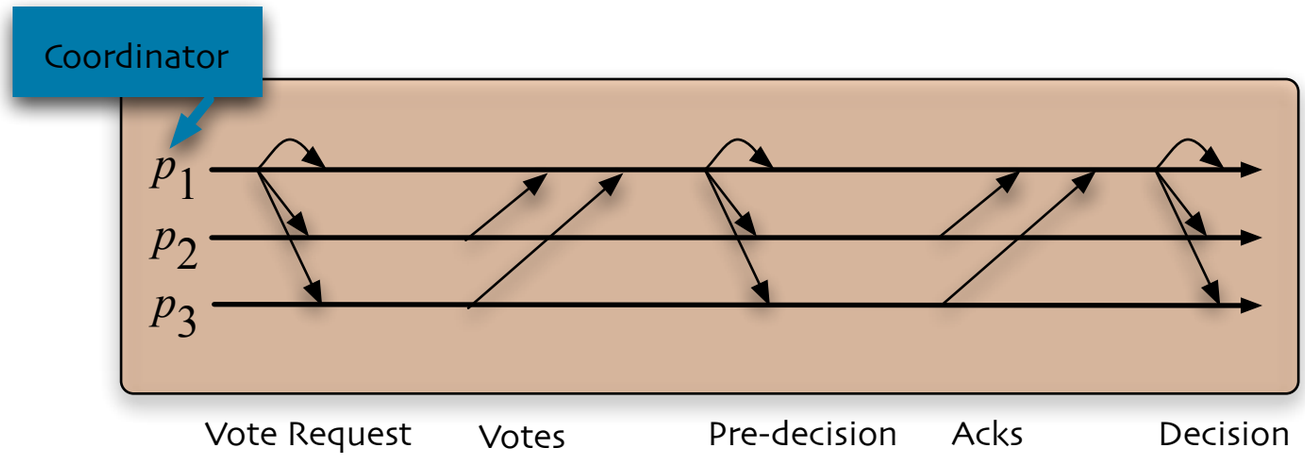
- Consider the 3-phase-commit protocol:



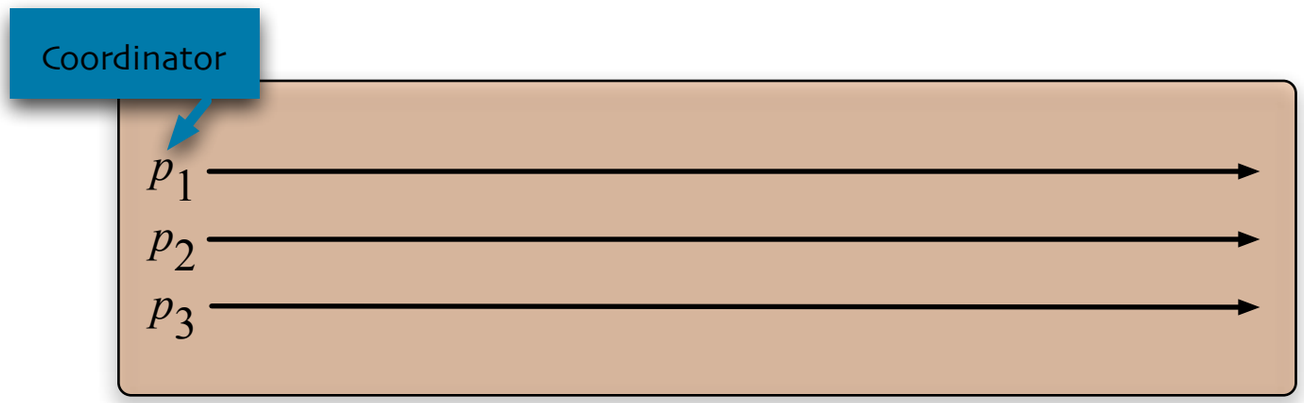
- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:

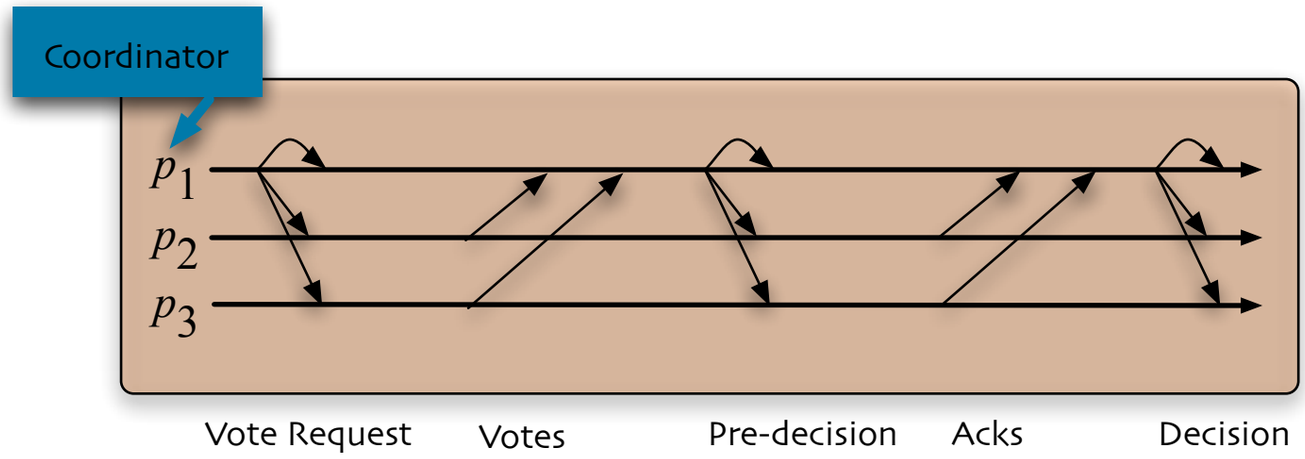


- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides

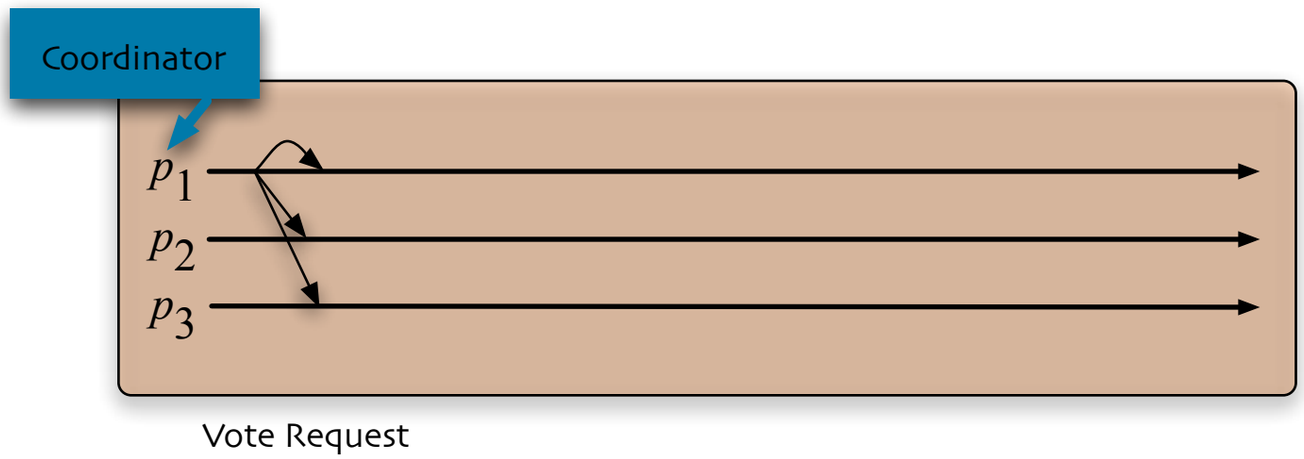


Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:

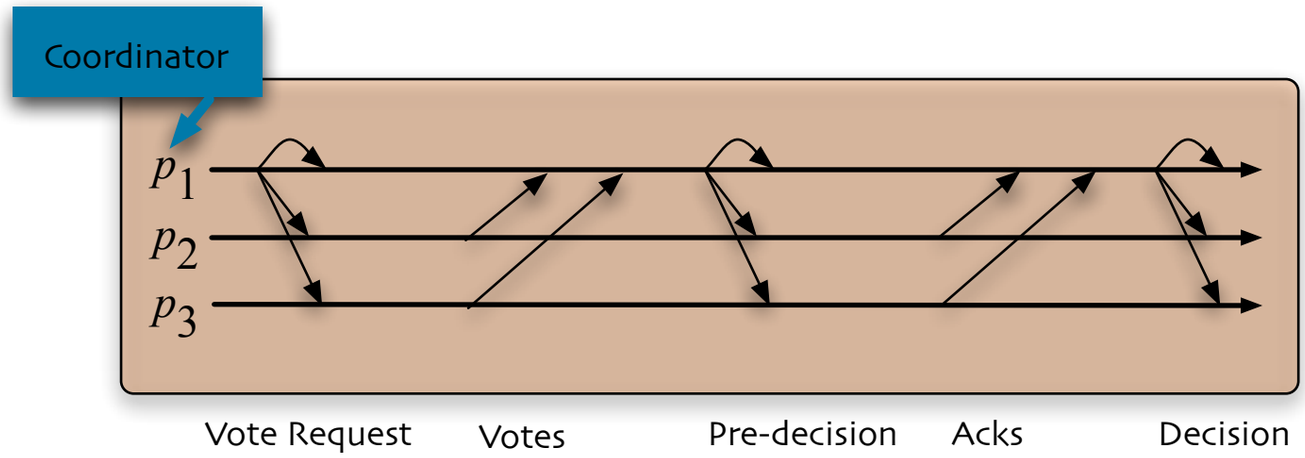


- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides

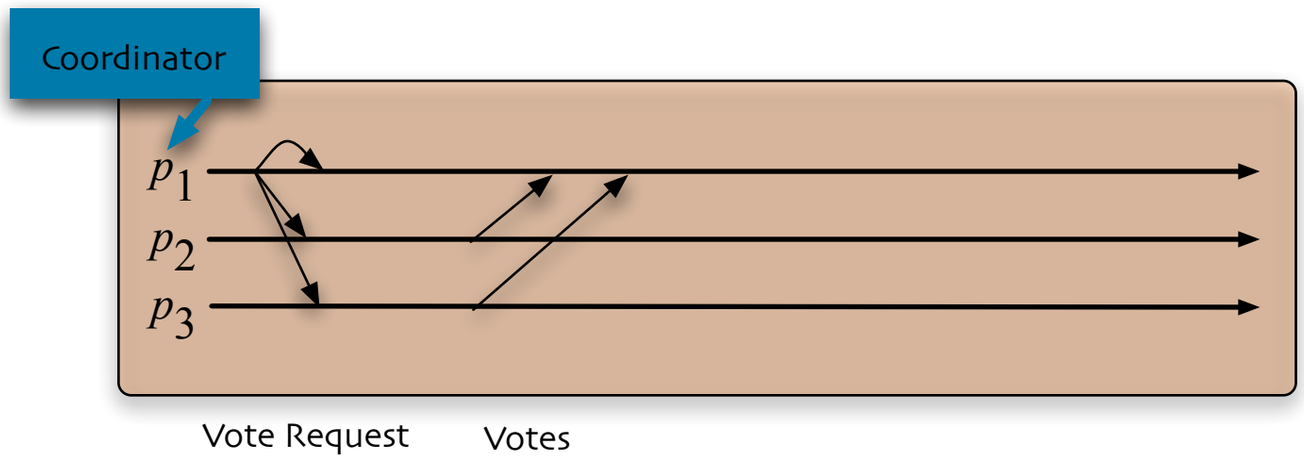


Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:

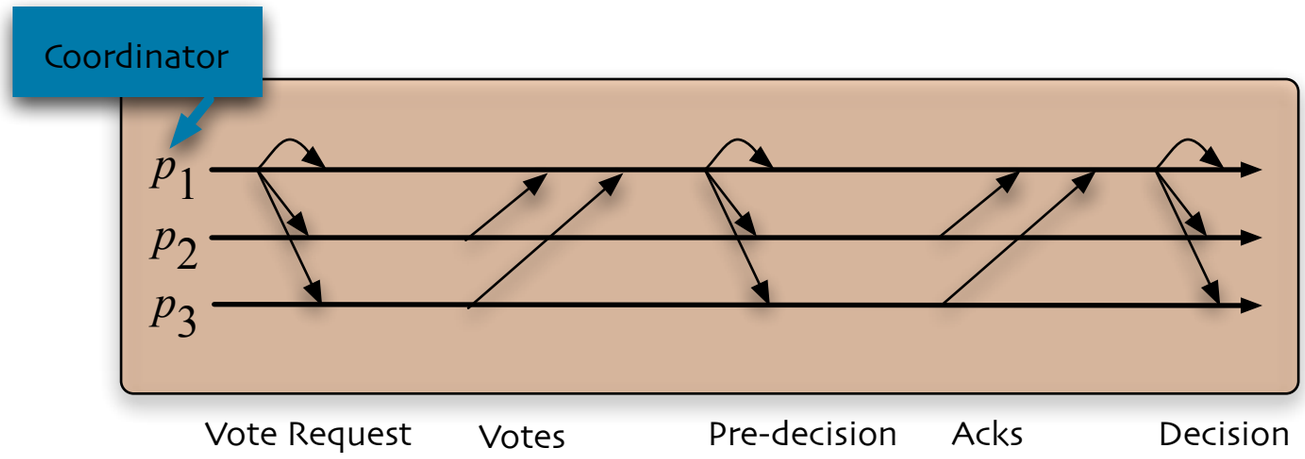


- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides

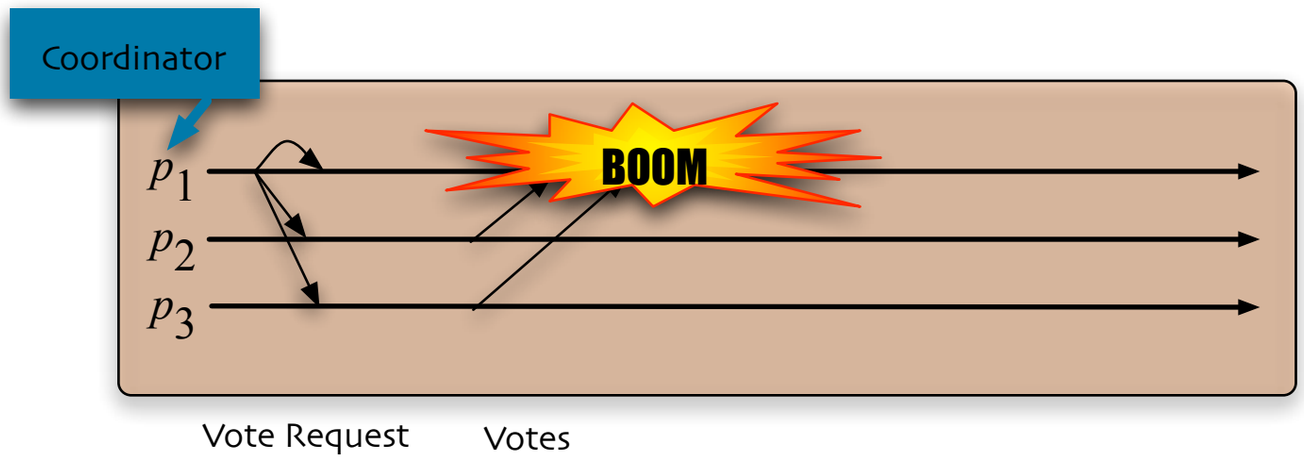


Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:

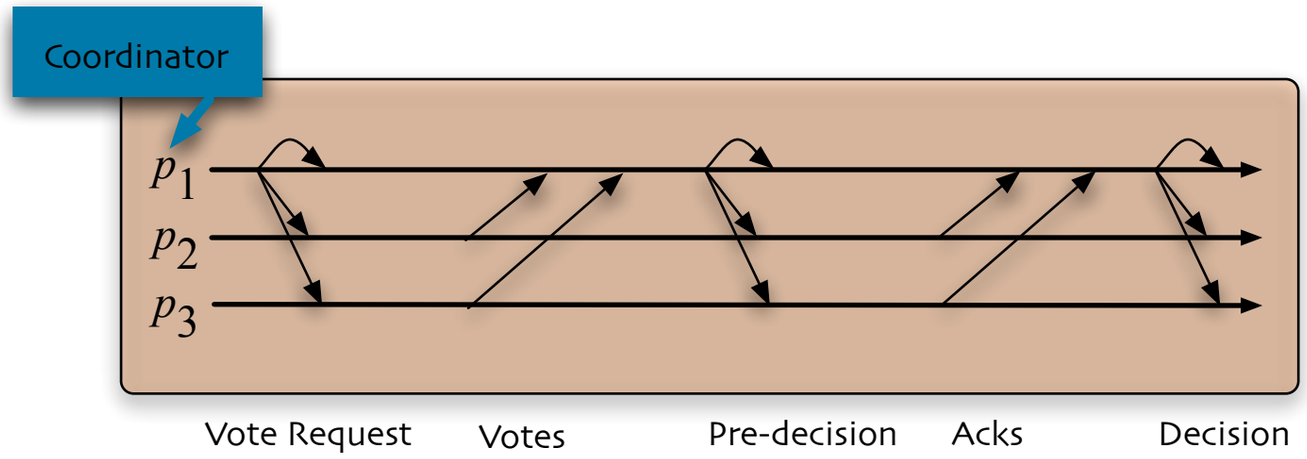


- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides

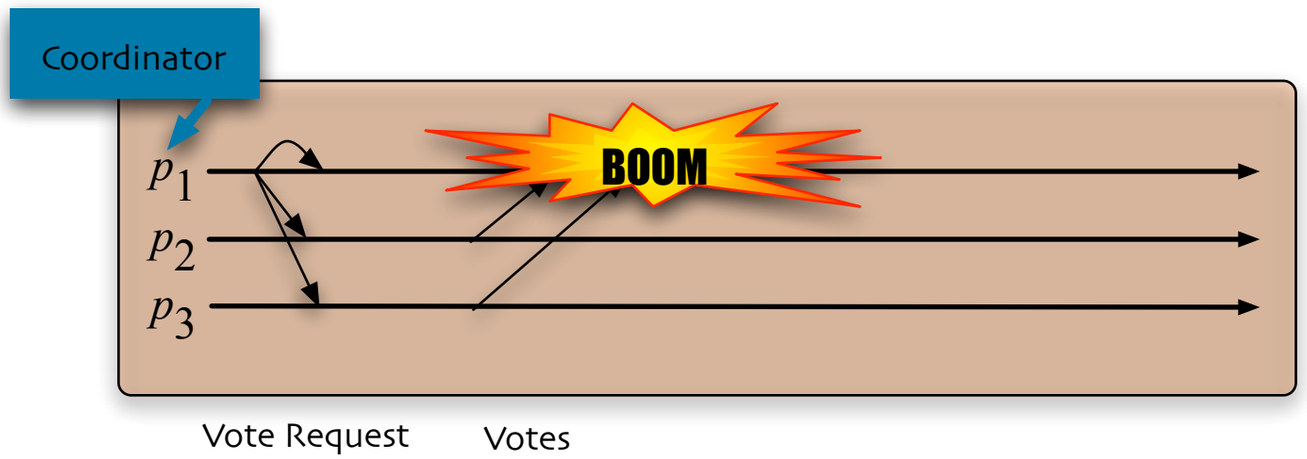


Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



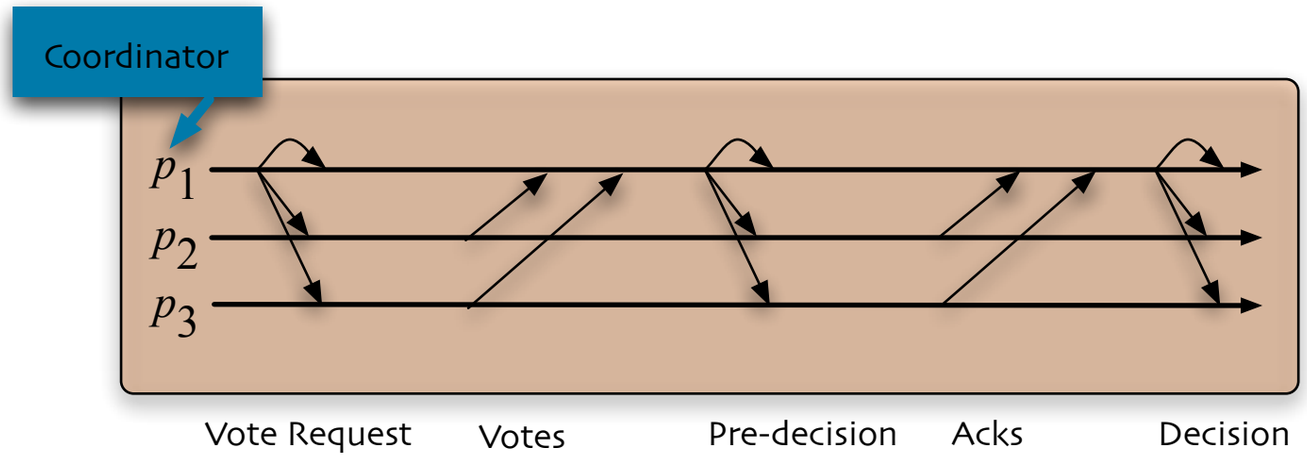
- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides



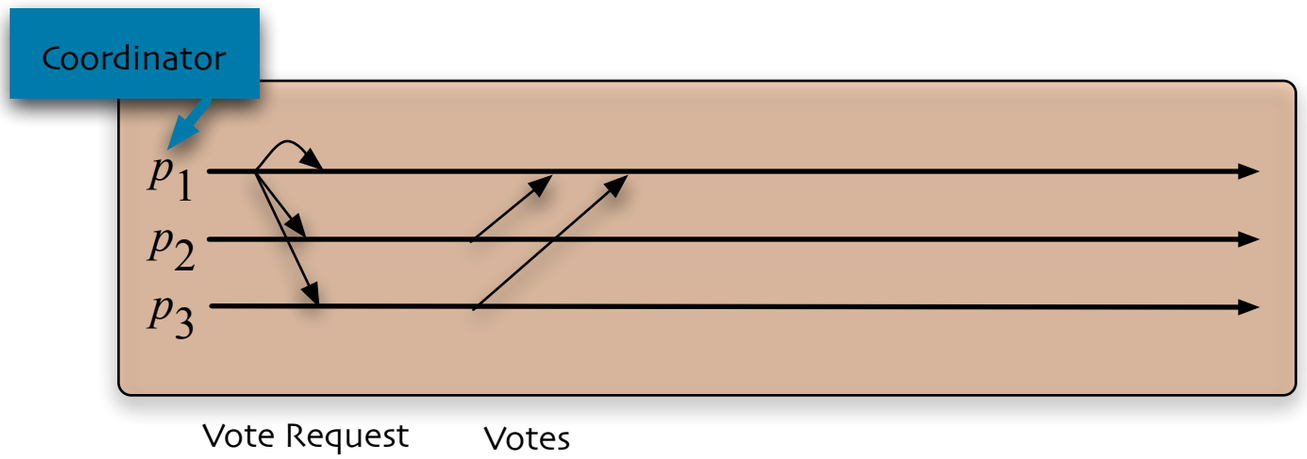
- What if coordinator fails after the votes?

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



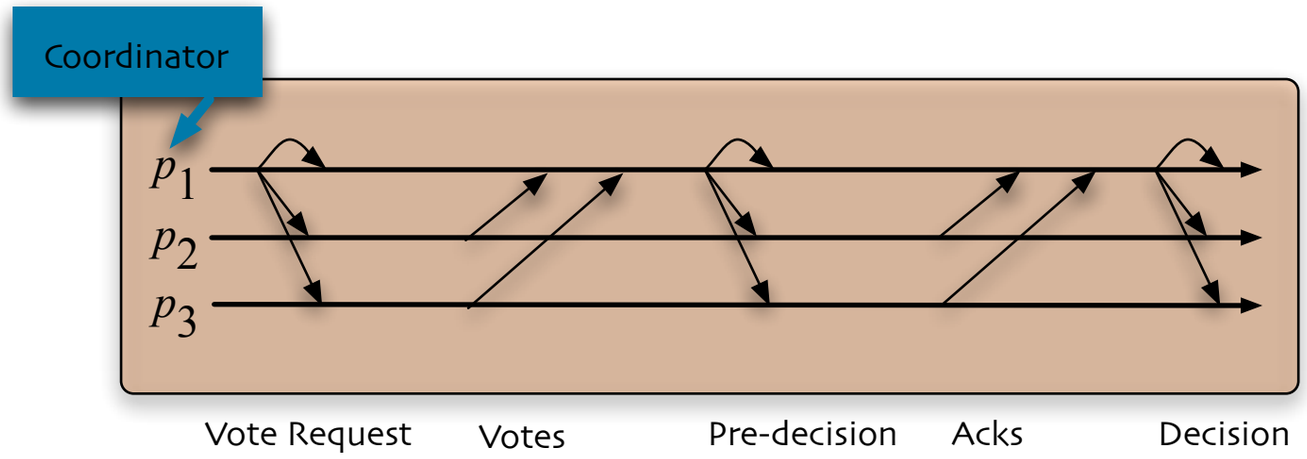
- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides



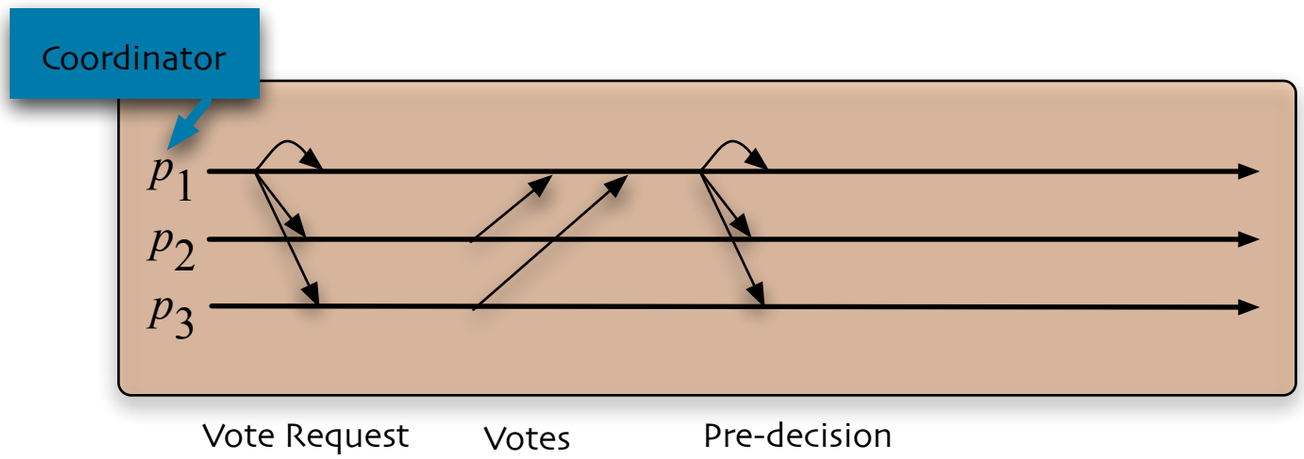
- What if coordinator fails after the votes?

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



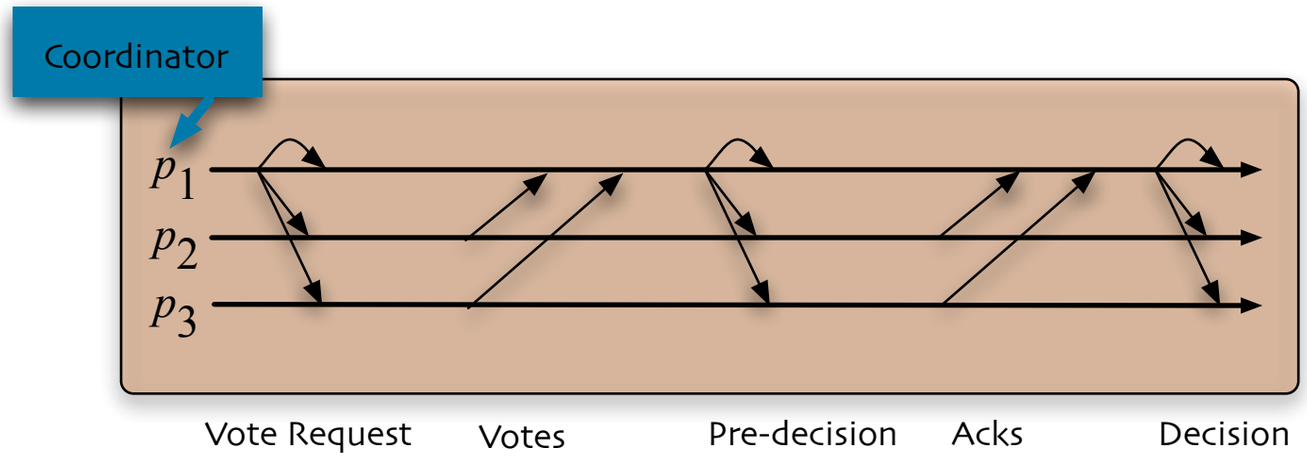
- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides



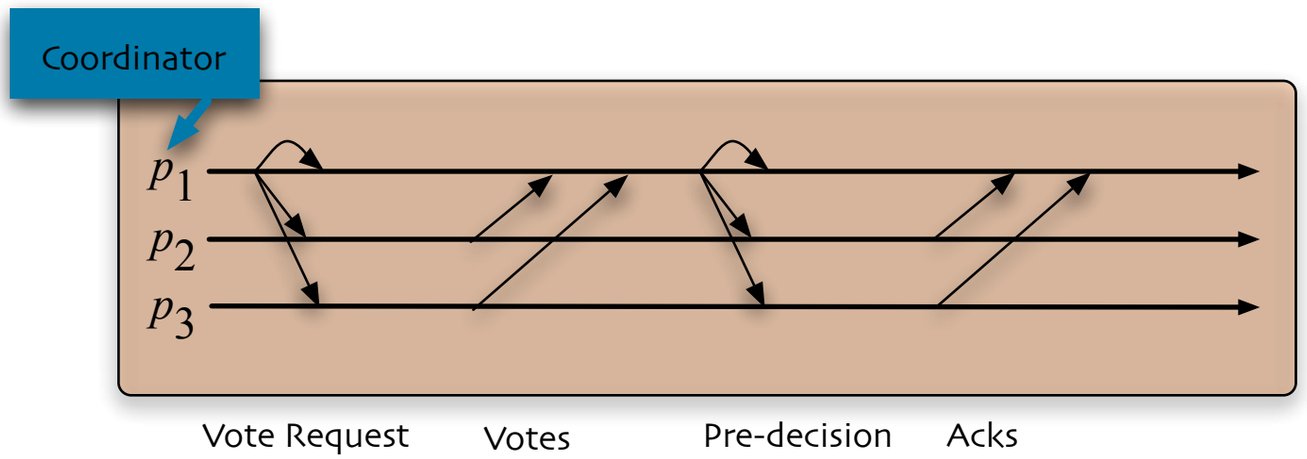
- What if coordinator fails after the votes?

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



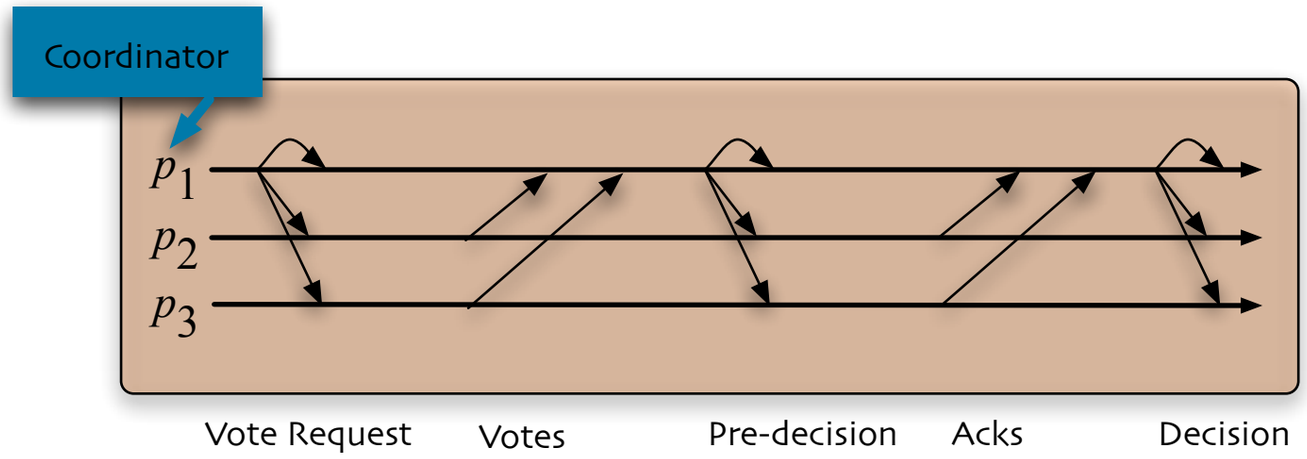
- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides



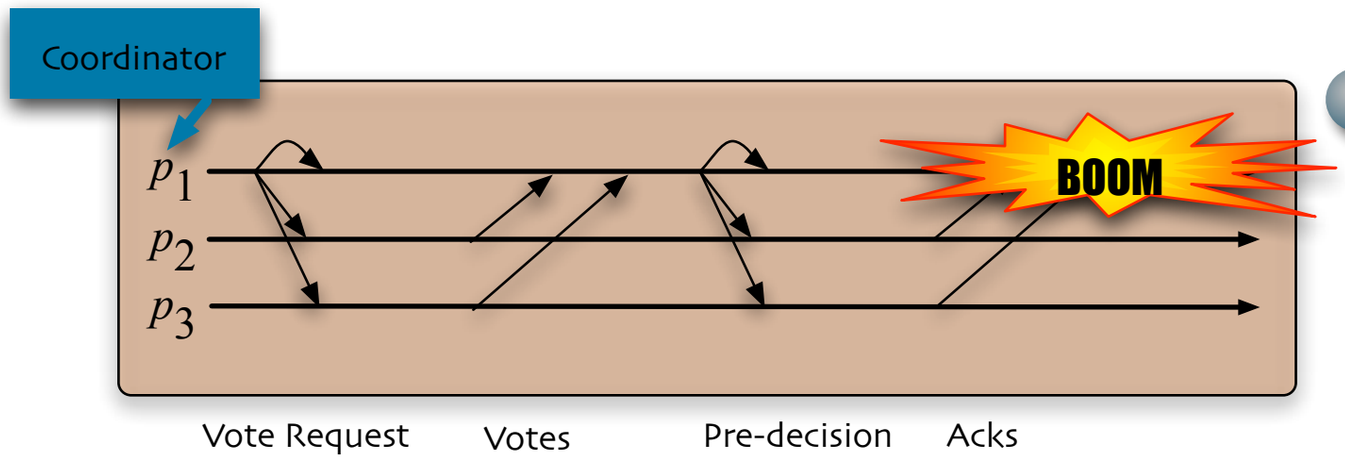
- What if coordinator fails after the votes?

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



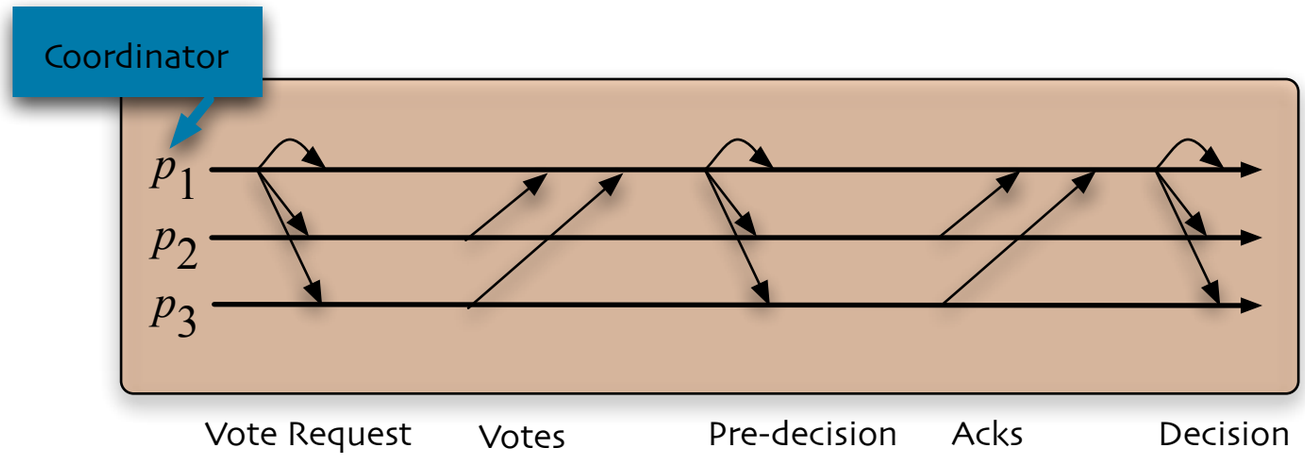
- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides



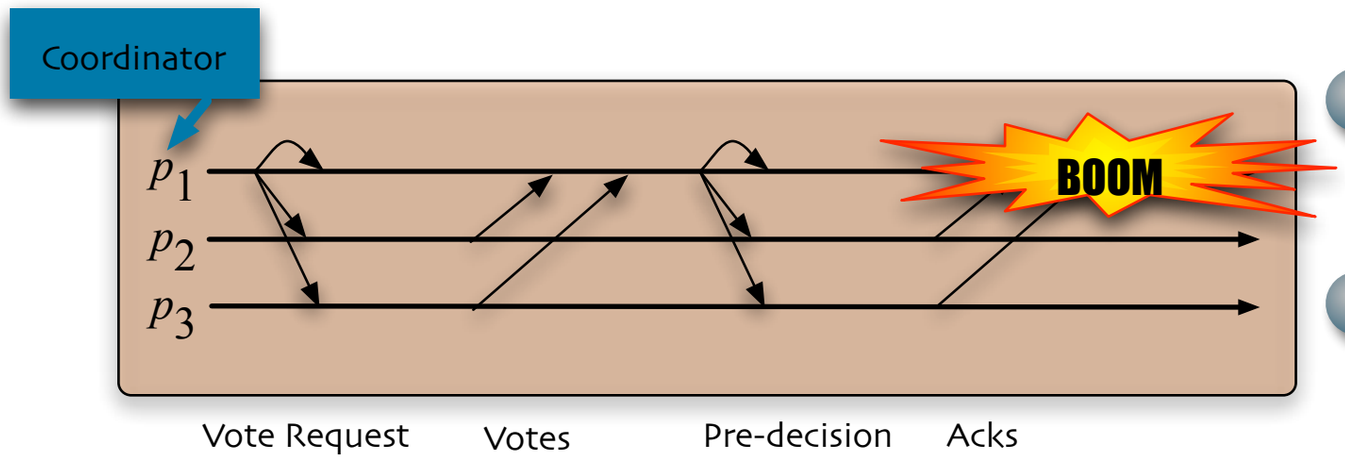
- What if coordinator fails after the votes?

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



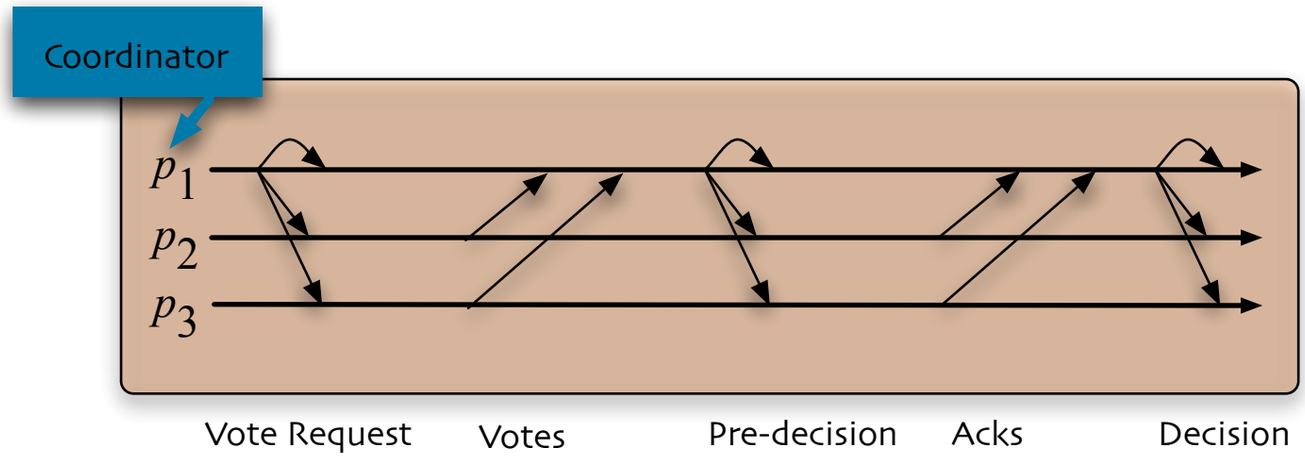
- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides



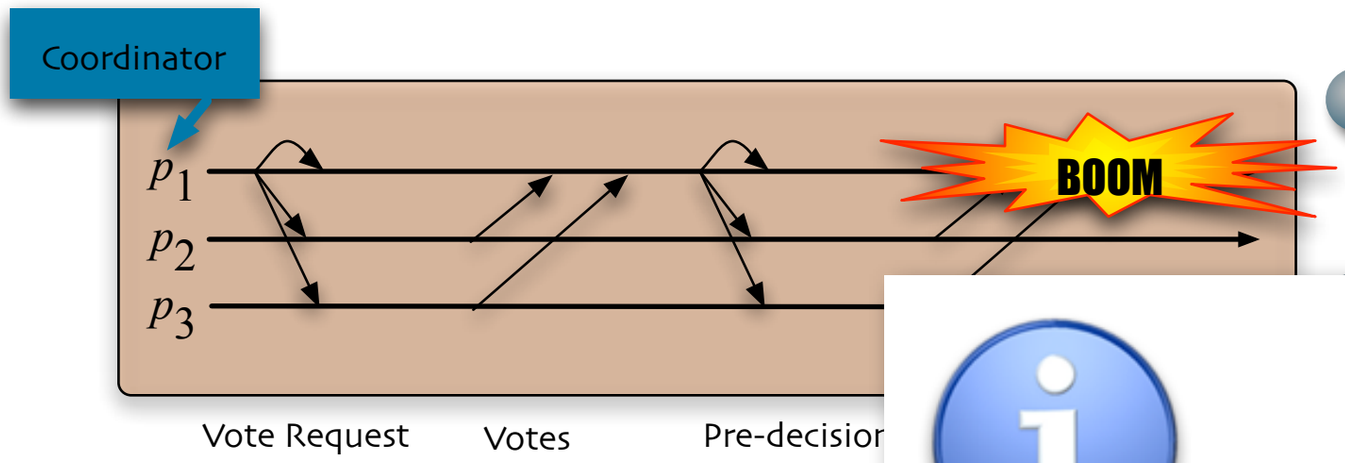
- What if coordinator fails after the votes?
- What if coordinator fails after the acks?

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides

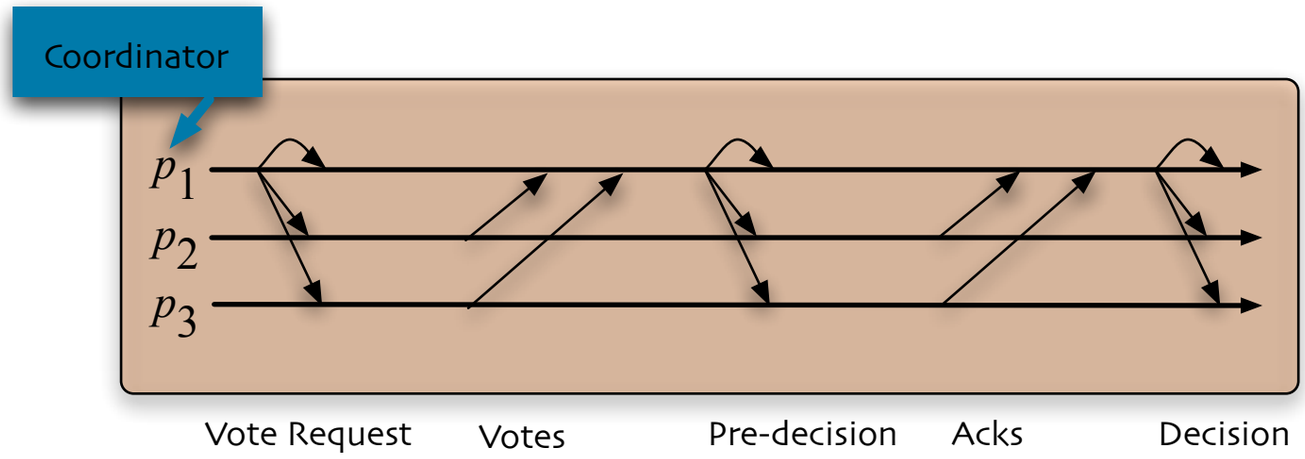


- What if coordinator fails after the votes?
- ...ails

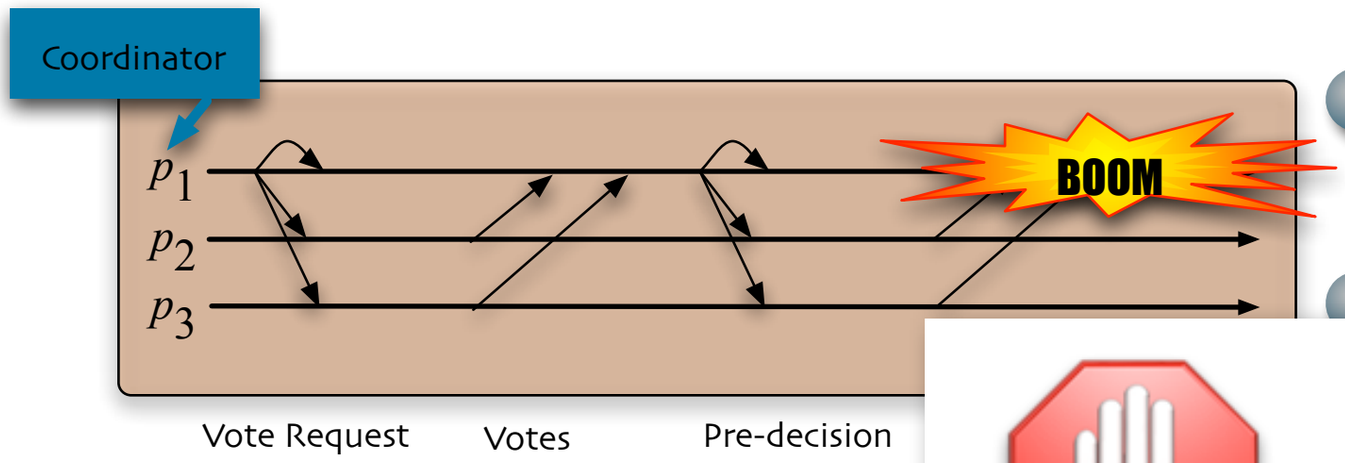
OK
in Fail-Stop model

Non-Blocking Atomic Commitment

- Consider the 3-phase-commit protocol:



- Coordinator requests votes
- Others vote
- Coordinator pre-decides
- Others acknowledge
- Coordinator decides



- What if coordinator fails after the votes?
- What if coordinator fails

Non terminating
in Crash-Stop model

Non-Blocking Atomic Commitment

Non-Blocking Atomic Commitment

- Does the problem, as stated, still makes sense in the Crash-Stop model?

Non-Blocking Atomic Commitment

- Does the problem, as stated, still makes sense in the Crash-Stop model?
- No, we need to consider the Weak Non-Blocking Atomic Commitment problem:

Non-Blocking Atomic Commitment

- Does the problem, as stated, still makes sense in the Crash-Stop model?
- No, we need to consider the Weak Non-Blocking Atomic Commitment problem:

> Non-triviality: If all processes vote yes and no participant is ever suspected of failure then commit should be decided

[R. Guerraoui, Revisiting the relationship between non-blocking atomic commitment and consensus problems, 1995]

Consensus

Consensus

- The Consensus problem is usually seen as an abstraction of most distributed agreement problems

Consensus

- The Consensus problem is usually seen as an abstraction of most distributed agreement problems
- Several key problems of dependable distributed systems depend on or are reducible to Consensus: weak atomic commitment, atomic multicast, group membership, view synchronous multicast

[R. Guerraoui, Revisiting the relationship between non blocking atomic commitment and consensus problems, 1995]
[R. Guerraoui and A. Schiper, The Generic Consensus Service, 2001]

Consensus

- The Consensus problem is usually seen as an abstraction of most distributed agreement problems
- Several key problems of dependable distributed systems depend on or are reducible to Consensus: weak atomic commitment, atomic multicast, group membership, view synchronous multicast

[R. Guerraoui, Revisiting the relationship between non blocking atomic commitment and consensus problems, 1995]
[R. Guerraoui and A. Schiper, The Generic Consensus Service, 2001]

- However, Consensus can be very hard to solve if one cannot accurately detect the failure of the processes

Consensus

Consensus

- Consider a finite set of processes where the correct ones vote either yes or no. Processes are expected to decide on a value satisfying the following properties:

Consensus

- Consider a finite set of processes where the correct ones vote either yes or no. Processes are expected to decide on a value satisfying the following properties:

Termination: every correct process eventually decides

Consensus

- Consider a finite set of processes where the correct ones vote either yes or no. Processes are expected to decide on a value satisfying the following properties:

Termination: every correct process eventually decides

Validity: the decision is on a voted value

Consensus

- Consider a finite set of processes where the correct ones vote either yes or no. Processes are expected to decide on a value satisfying the following properties:

Termination: every correct process eventually decides

Validity: the decision is on a voted value

Agreement: no two processes decide differently

Consensus

- Consider a finite set of processes where the correct ones vote either yes or no. Processes are expected to decide on a value satisfying the following properties:

Termination: every correct process eventually decides

Validity: the decision is on a voted value

Agreement: no two processes decide differently

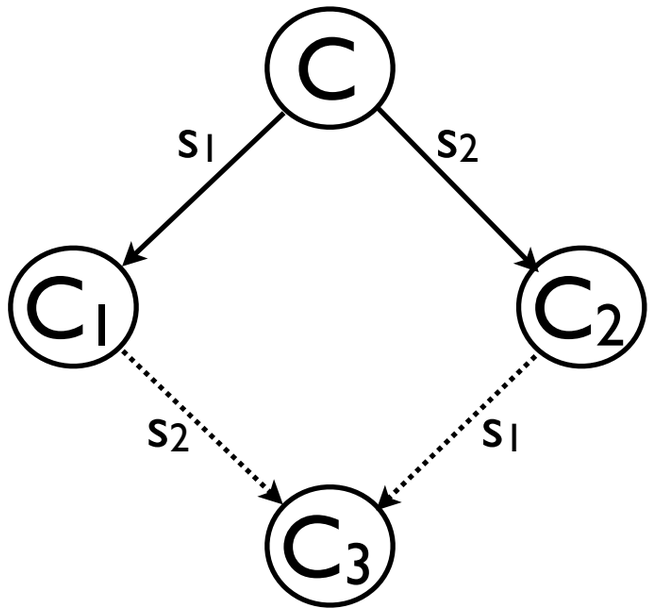
When we cannot: FLP impossibility result

- "Impossibility of Distributed Consensus with One Faulty Process", Fischer, Lynch and Patterson in 1985:
  - In a "pure" asynchronous system (even) with reliable communication channels,
  - When (at least) some process may fail by crashing, forever ceasing its computation,
  - No deterministic algorithm can solve consensus
- Isn't such a result so counterintuitive?!

When we cannot: FLP impossibility result

- Consider an asynchronous system model, a finite set of processes P completely connected by reliable channels
- A process p is modeled through an input register ip, an output register op and an unbounded amount of internal storage. A configuration of the system consists of the internal state of each process, together with the contents of the message buffer.
- Processes take deterministic events (p, m) determined by the messages they receive. A schedule from a configuration C is a finite or infinite sequence of events that can be applied, in turn, starting from C.

When we cannot: FLP impossibility result



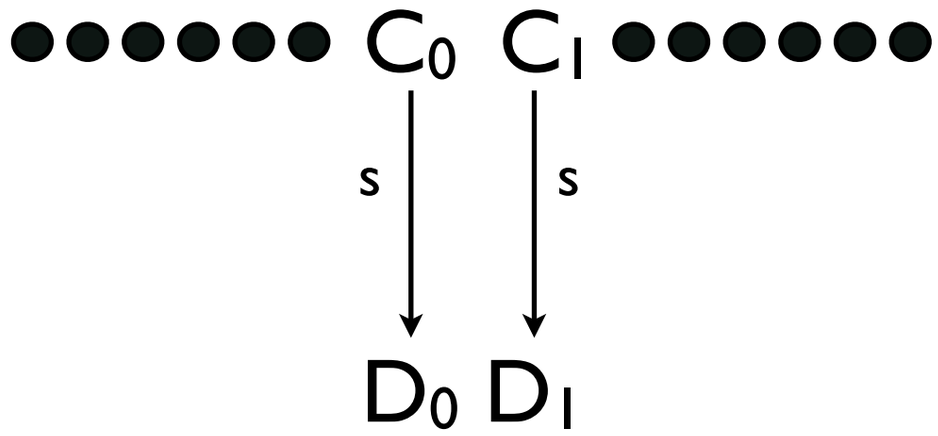
- Suppose that from some configuration C, the schedules s1, s2 lead to configurations C1, C2, respectively. If the sets of processes taking steps in s1 and s2, respectively, are disjoint, then s2 can be applied to C1 and s1 can be applied to C2, and both lead to the same configuration C3.

When we cannot: FLP impossibility result

- The algorithm has an initial bivalent configuration
  - Assume not:
  - Let C0 (0-valent) and C1 (1-valent) be adjacent configurations differing on the initial value of p

When we cannot: FLP impossibility result

- The algorithm has an initial <u>bivalent</u> configuration
  - Assume not:
  - Let Co (o-valent) and C1 (1-valent) be adjacent configurations differing on the initial value of p

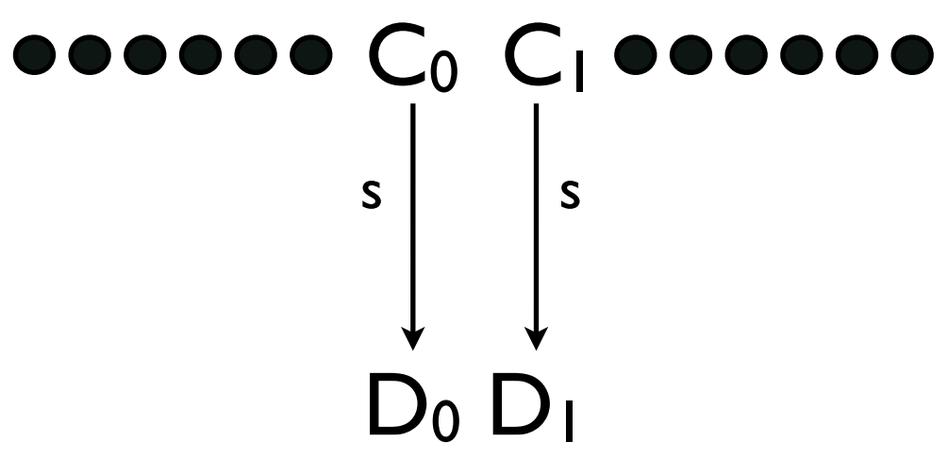


s - is a deciding schedule

s - has no steps of p

When we cannot: FLP impossibility result

- The algorithm has an initial bivalent configuration
  - Assume not:
  - Let C0 (0-valent) and C1 (1-valent) be adjacent configurations differing on the initial value of p



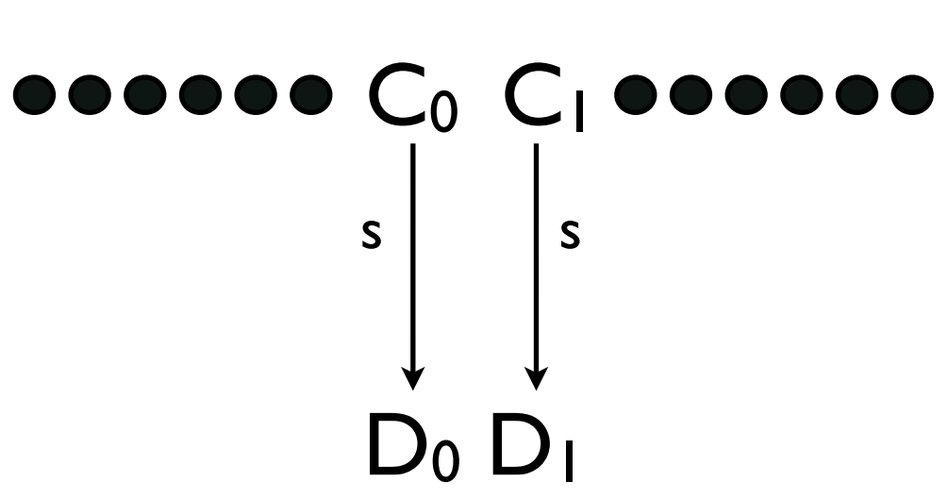
s - is a deciding schedule

s - has no steps of p

- S is a deciding schedule applicable both to C0 and to C1

When we cannot: FLP impossibility result

- The algorithm has an initial bivalent configuration
  - Assume not:
  - Let C0 (0-valent) and C1 (1-valent) be adjacent configurations differing on the initial value of p



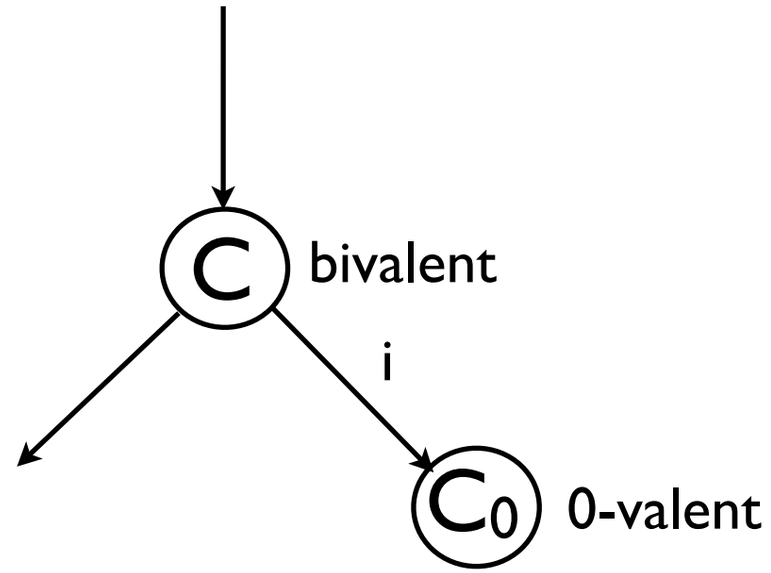
s - is a deciding schedule

s - has no steps of p

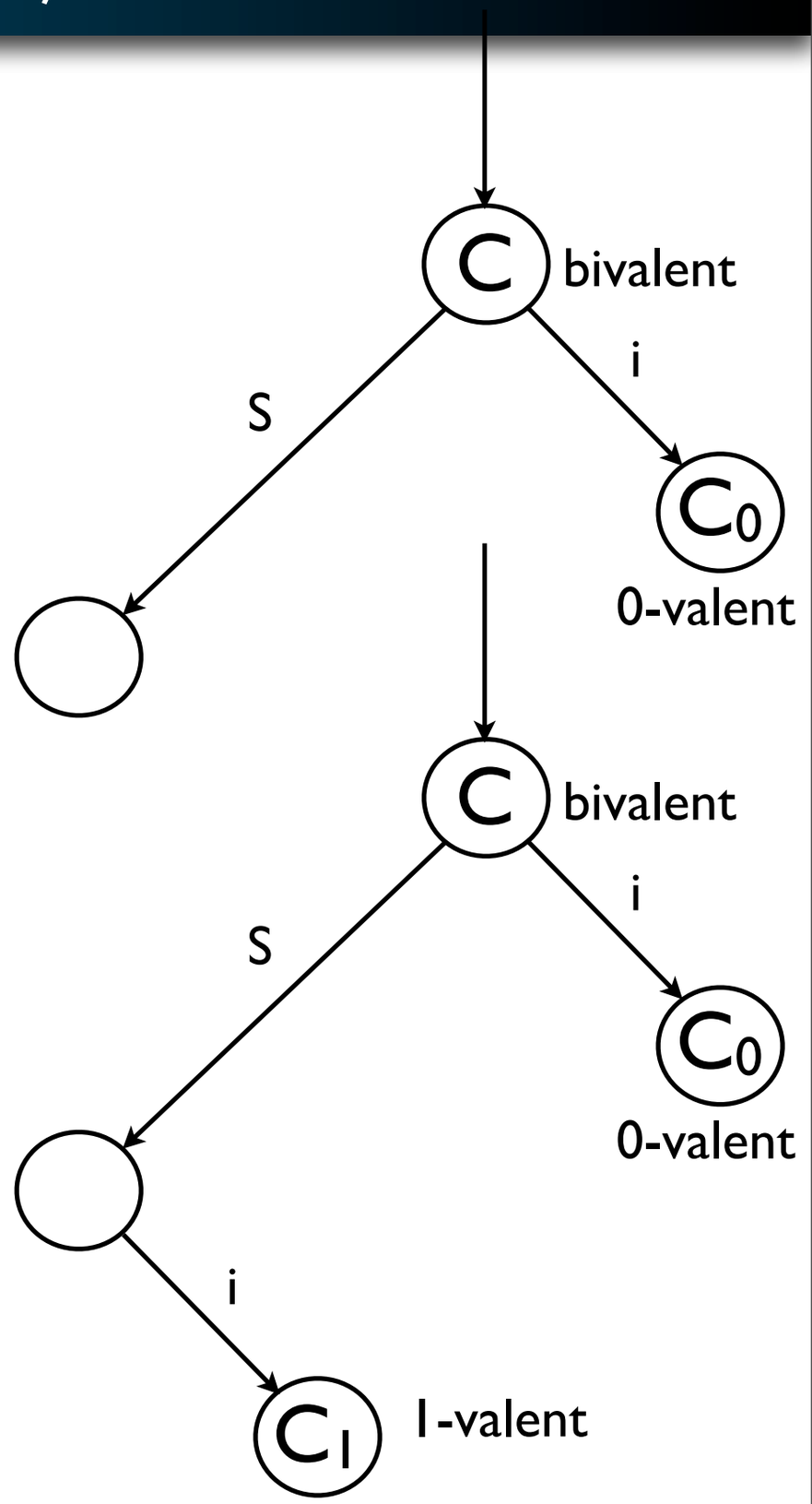
- S is a deciding schedule applicable both to C0 and to C1
- The decision at D0 and D1 must be the same. This implies that either C0 or C1 is bivalent!

When we cannot: FLP impossibility result

- Any deciding schedule eventually forks a bivalent into a univalent configuration:

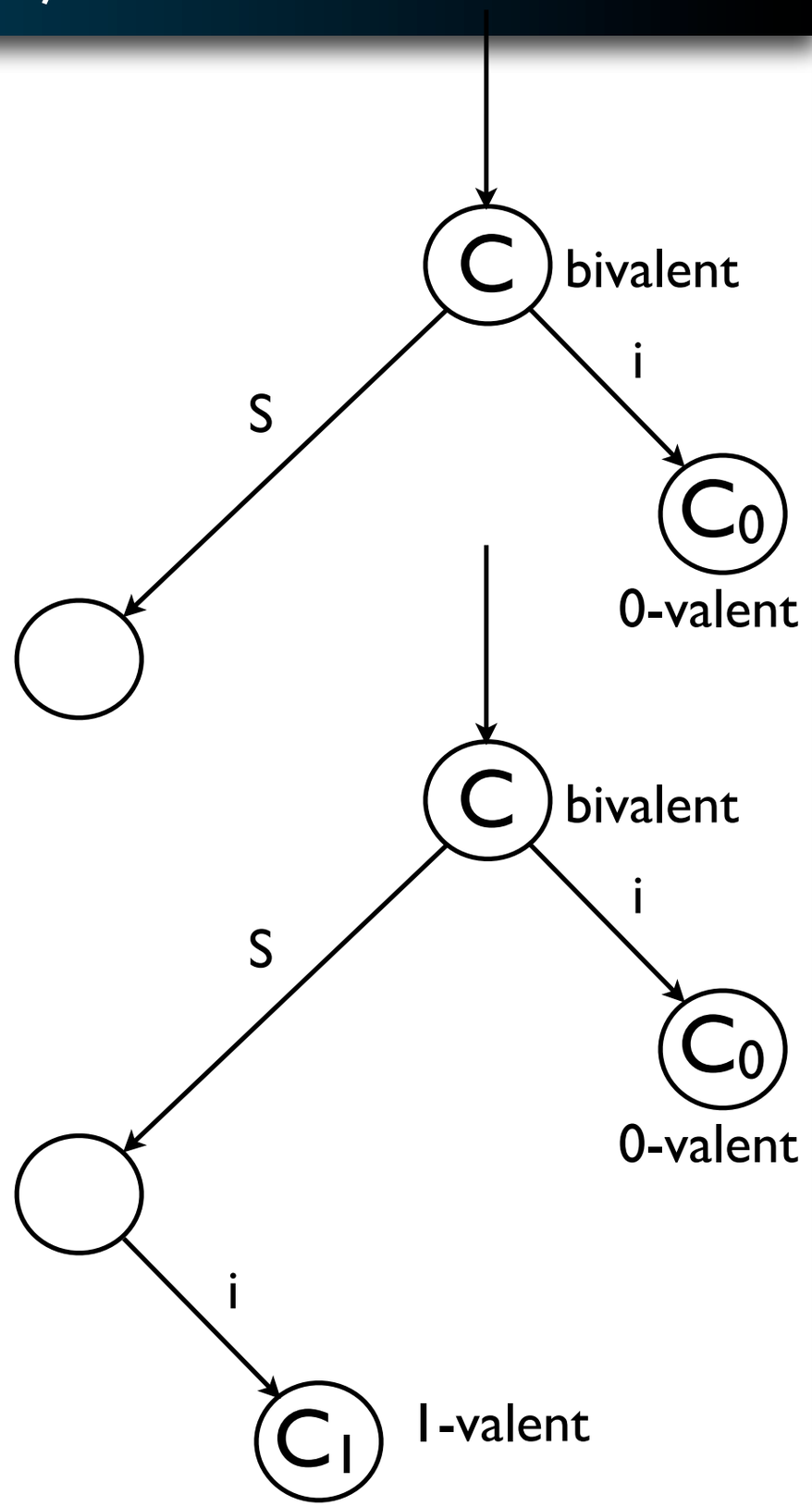


When we cannot: FLP impossibility result



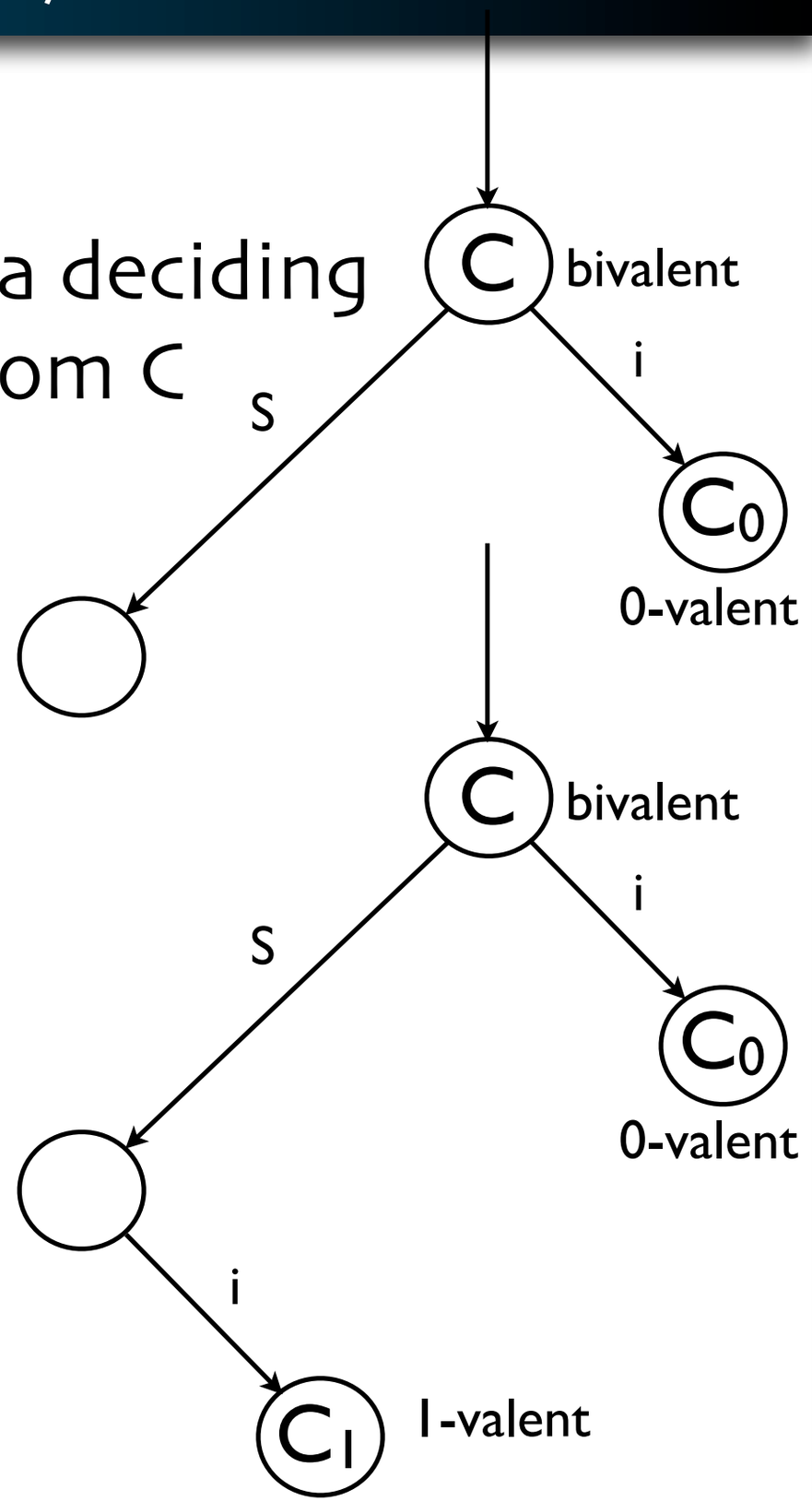
When we cannot: FLP impossibility result

- Suppose i crashes.



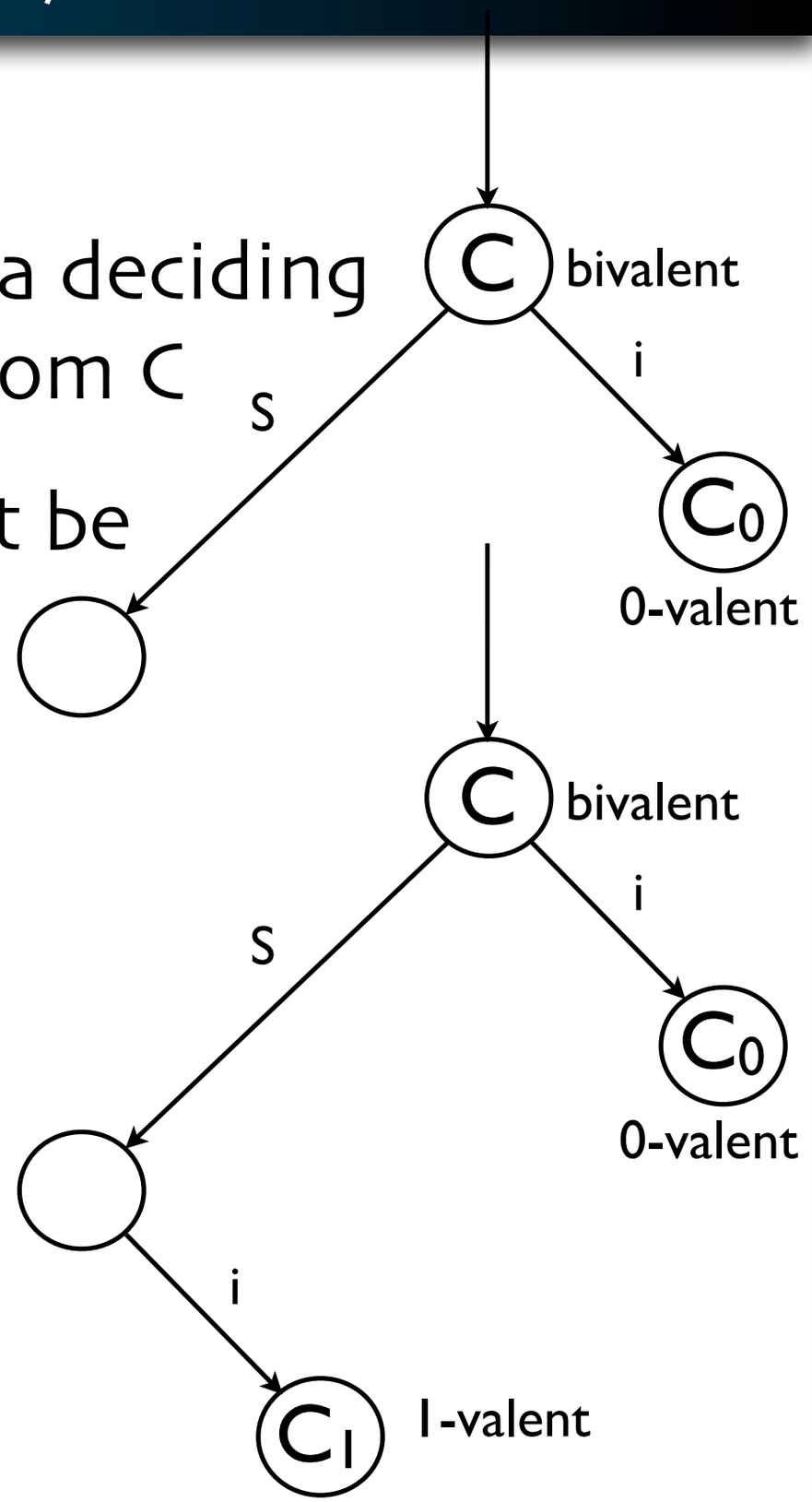
When we cannot: FLP impossibility result

- Suppose i crashes.
- Since one crash is tolerated there is a deciding schedule S (without steps from i) from C



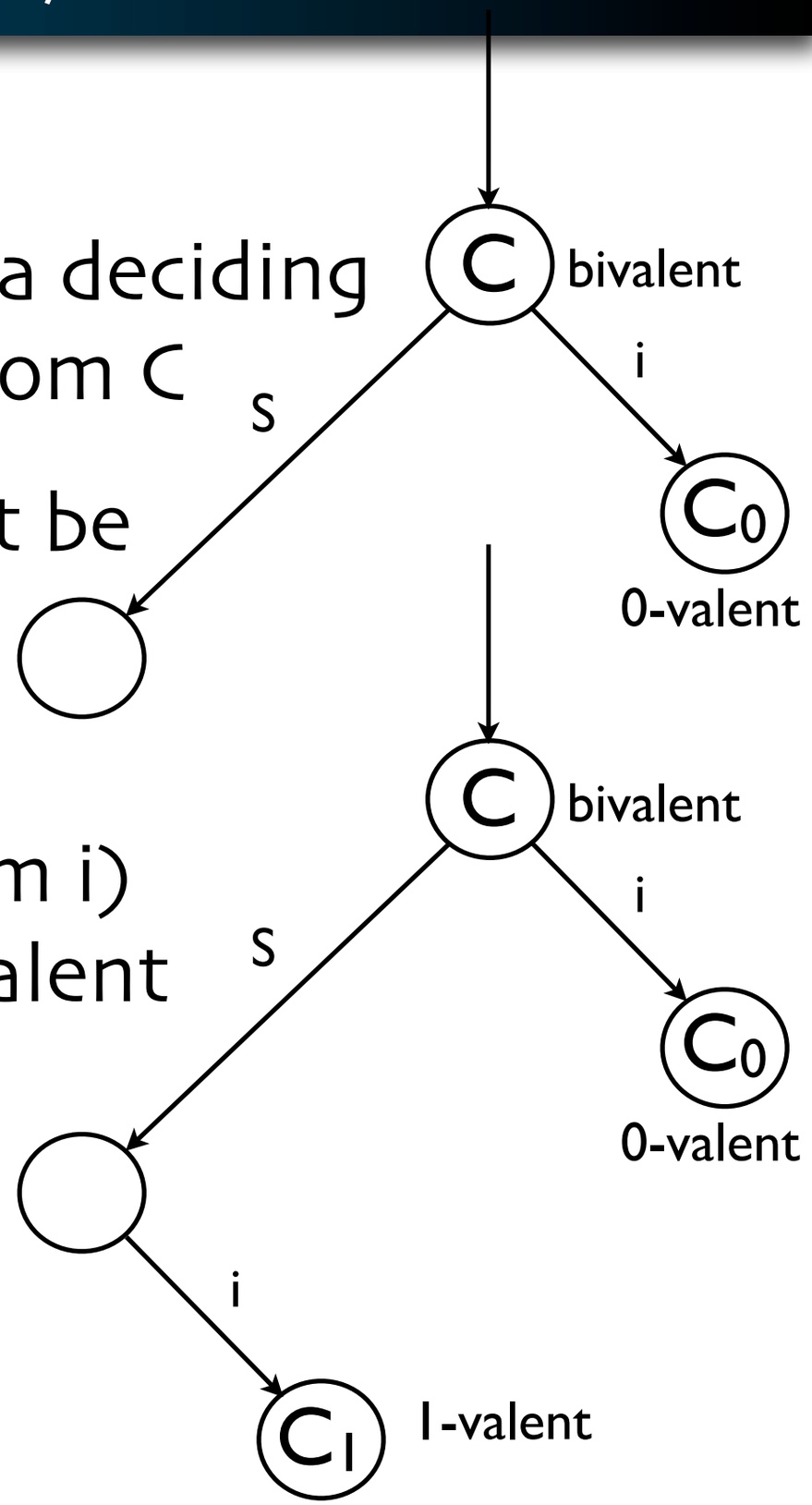
When we cannot: FLP impossibility result

- Suppose i crashes.
- Since one crash is tolerated there is a deciding schedule S (without steps from i) from C
- Any such S leads to a S(C) that must be o-valent



When we cannot: FLP impossibility result

- Suppose i crashes.
- Since one crash is tolerated there is a deciding schedule S (without steps from i) from C
- Any such S leads to a S(C) that must be o-valent
- Since C is bivalent there must be some schedule S (without steps from i) after which applying i leads to a 1-valent configuration (a "Hook"). But since S can be applied do C_0 this leads to a contradiction!



Trivial "Consensus" solutions

Trivial "Consensus" solutions

- Consider the variants of Consensus with just two of its properties:

Trivial "Consensus" solutions

- Consider the variants of Consensus with just two of its properties:

No Agreement

Trivial "Consensus" solutions

- Consider the variants of Consensus with just two of its properties:

No Agreement

```
Boolean Consensus (Boolean v)
{
    return v;
}
```


Trivial "Consensus" solutions

- Consider the variants of Consensus with just two of its properties:

No Agreement

```
Boolean Consensus (Boolean v)
{
    return v;
}
```

No Validity

Trivial "Consensus" solutions

- Consider the variants of Consensus with just two of its properties:

No Agreement

```
Boolean Consensus (Boolean v)
{
    return v;
}
```

No Validity

```
Boolean Consensus (Boolean v)
{
    return True;
}
```

Trivial "Consensus" solutions

- Consider the variants of Consensus with just two of its properties:

No Agreement

```
Boolean Consensus (Boolean v)
{
    return v;
}
```

No Validity

```
Boolean Consensus (Boolean v)
{
    return True;
}
```

No Termination

Trivial "Consensus" solutions

- Consider the variants of Consensus with just two of its properties:

No Agreement

```
Boolean Consensus (Boolean v)
{
    return v;
}
```

No Validity

```
Boolean Consensus (Boolean v)
{
    return True;
}
```

No Termination

```
Boolean Consensus (Boolean v)
{
    while(1);
}
```

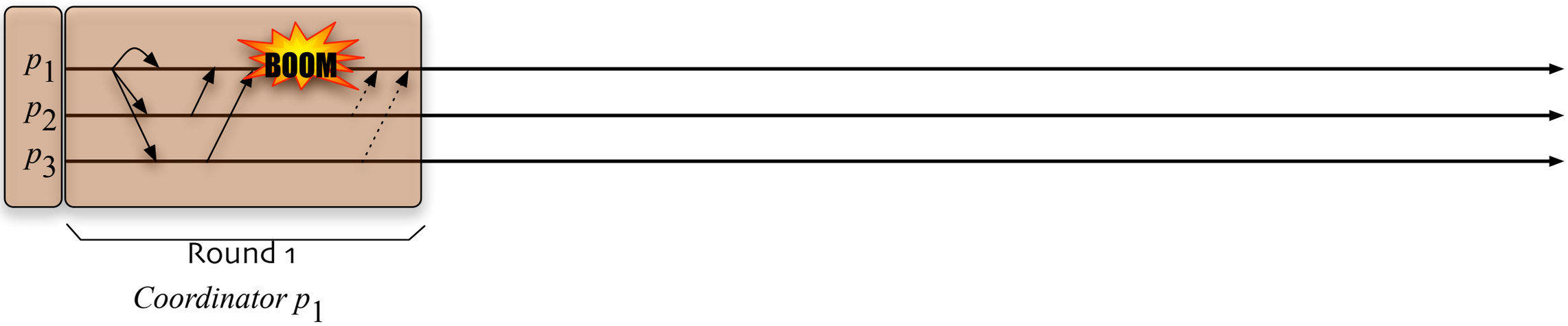

Solving Consensus

Solving Consensus

- Let's start by using a 3PC protocol to solve the problem. Asynchronous model, crash-stop faults.

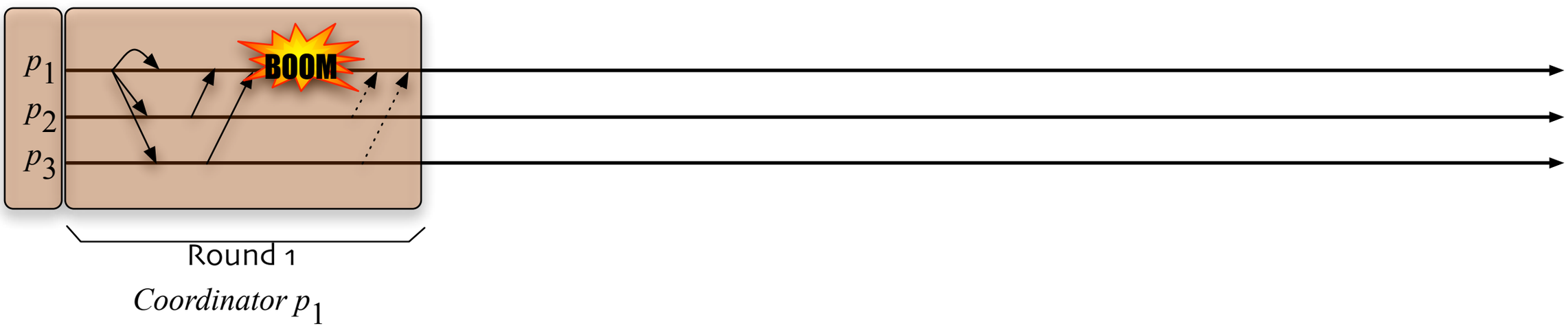
Solving Consensus

- Let's start by using a 3PC protocol to solve the problem. Asynchronous model, crash-stop faults.



Solving Consensus

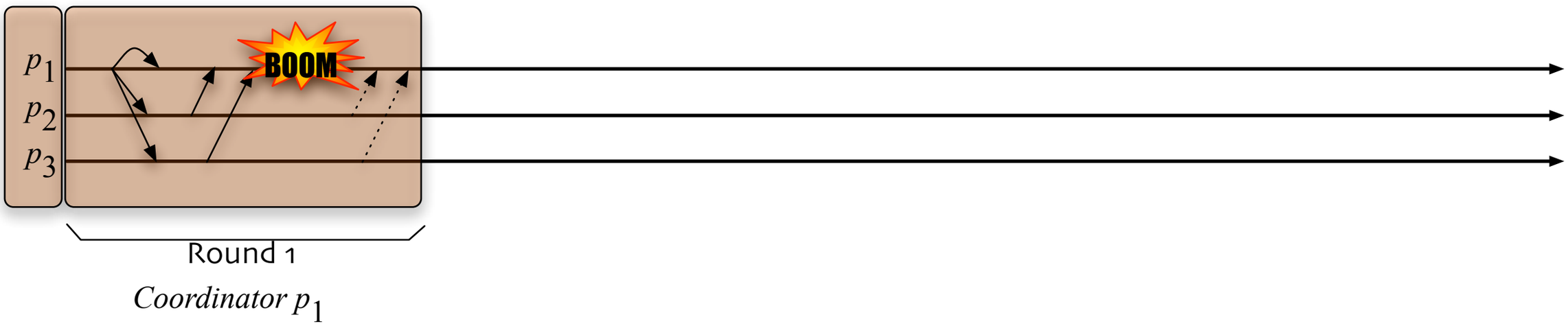
- Let's start by using a 3PC protocol to solve the problem. Asynchronous model, crash-stop faults.



Participants suspect the coordinator.

Solving Consensus

- Let's start by using a 3PC protocol to solve the problem. Asynchronous model, crash-stop faults.

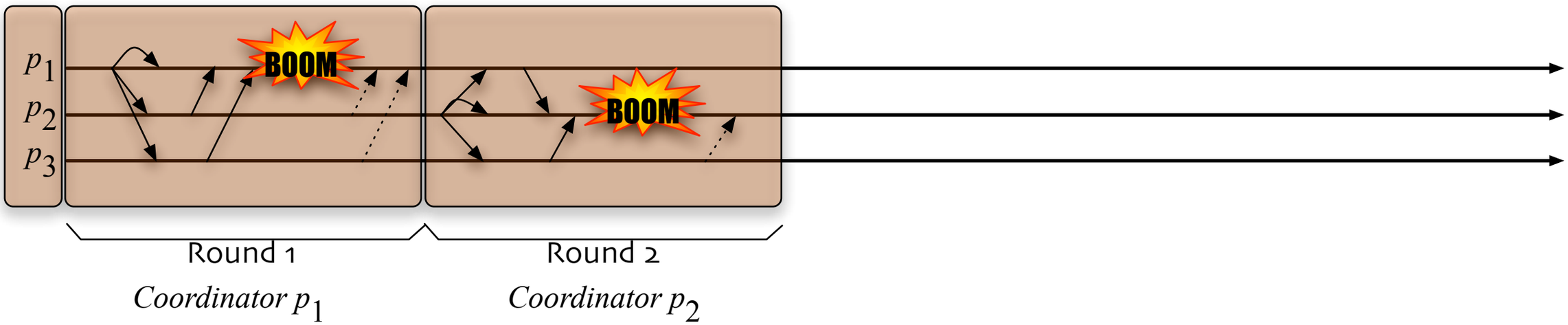


Participants suspect the coordinator.

Conundrum: shall I stay or shall I go?

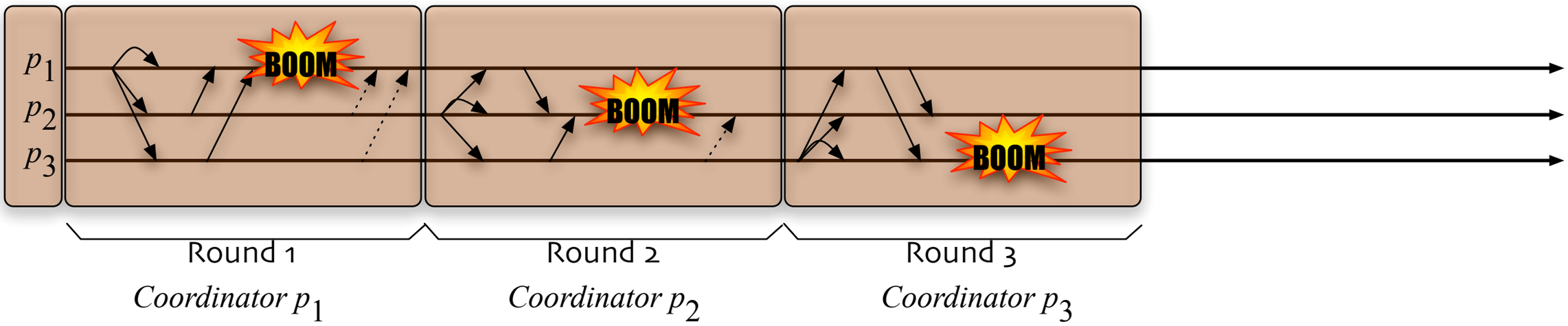
Solving Consensus

- Let's start by using a 3PC protocol to solve the problem. Asynchronous model, crash-stop faults.



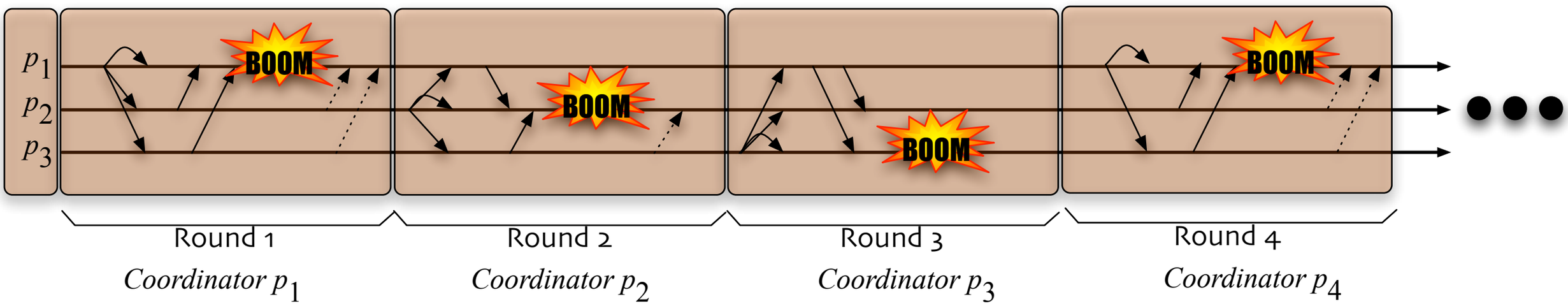
Solving Consensus

- Let's start by using a 3PC protocol to solve the problem. Asynchronous model, crash-stop faults.



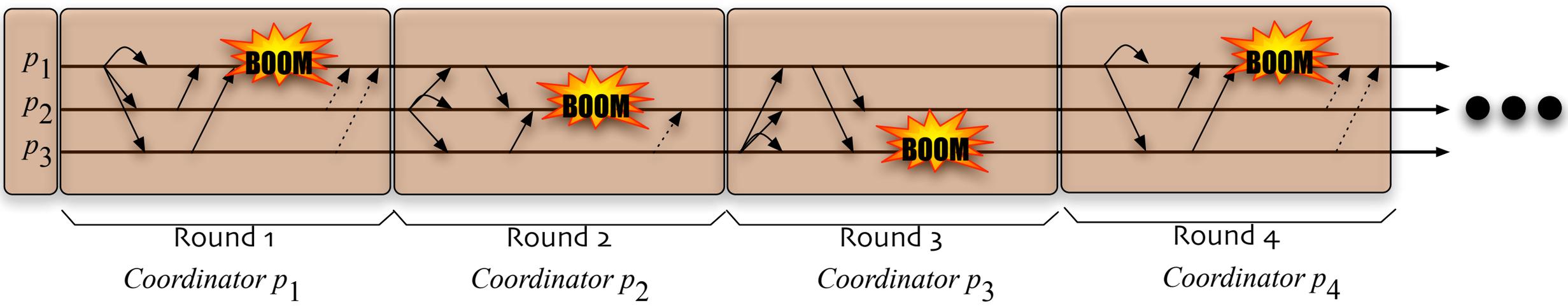
Solving Consensus

- Let's start by using a 3PC protocol to solve the problem. Asynchronous model, crash-stop faults.



Solving Consensus

- Let's start by using a 3PC protocol to solve the problem. Asynchronous model, crash-stop faults.



- ... ending up with the Chandra & Toueg's algorithm which is based on a Failure Detector Oracle

Unreliable failure detectors for reliable distributed systems, T. Chandra and S. Toueg, JACM, 1996

Consensus and Failure Detection

- What's a Failure Detector Oracle?

Consensus and Failure Detection

- What's a Failure Detector Oracle?
  - Forget about time-outs!

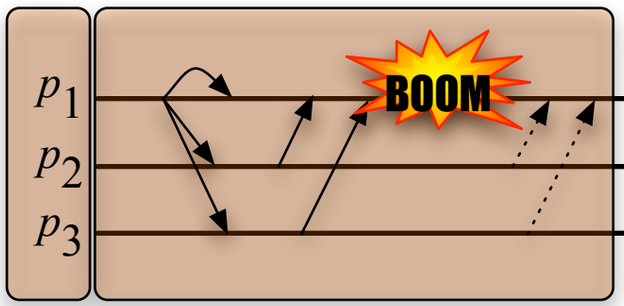
Consensus and Failure Detection

- What's a Failure Detector Oracle?
  - Forget about time-outs!
  - Consider something technologically more advanced

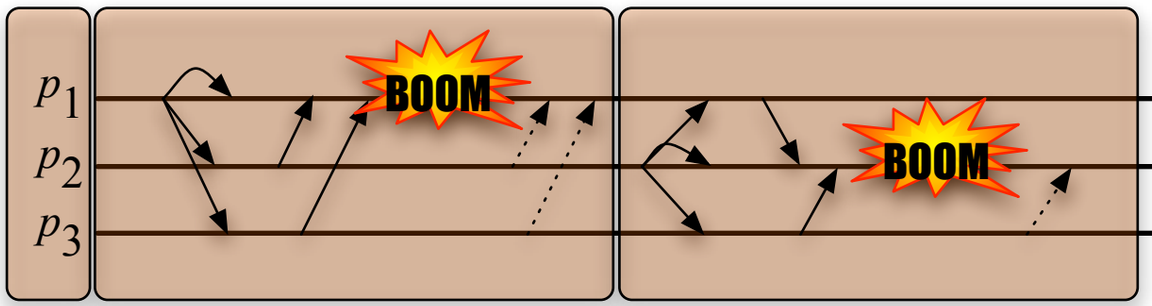
- What's a Failure Detector Oracle?
  - Forget about time-outs!
  - Consider something technologically more advanced

- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant

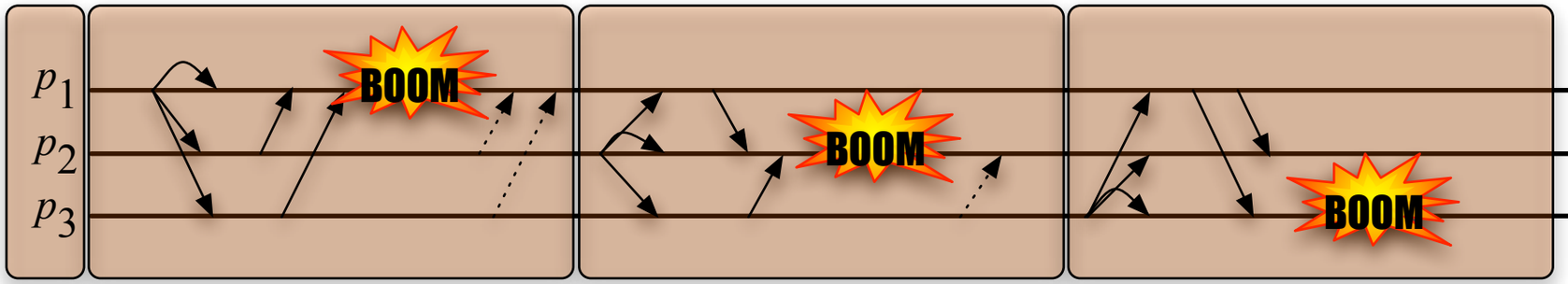
- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant



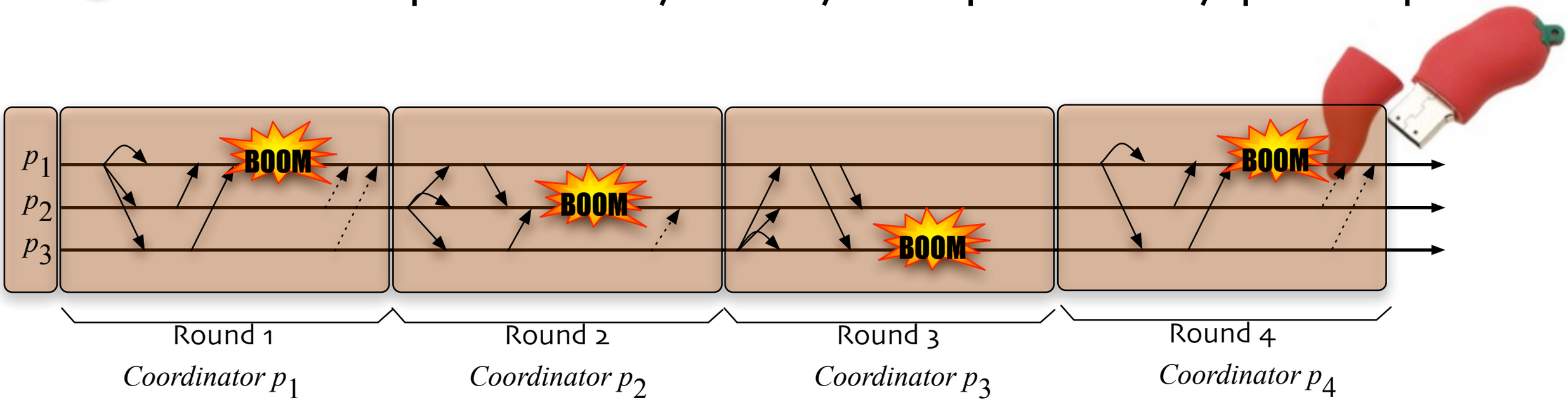
- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant



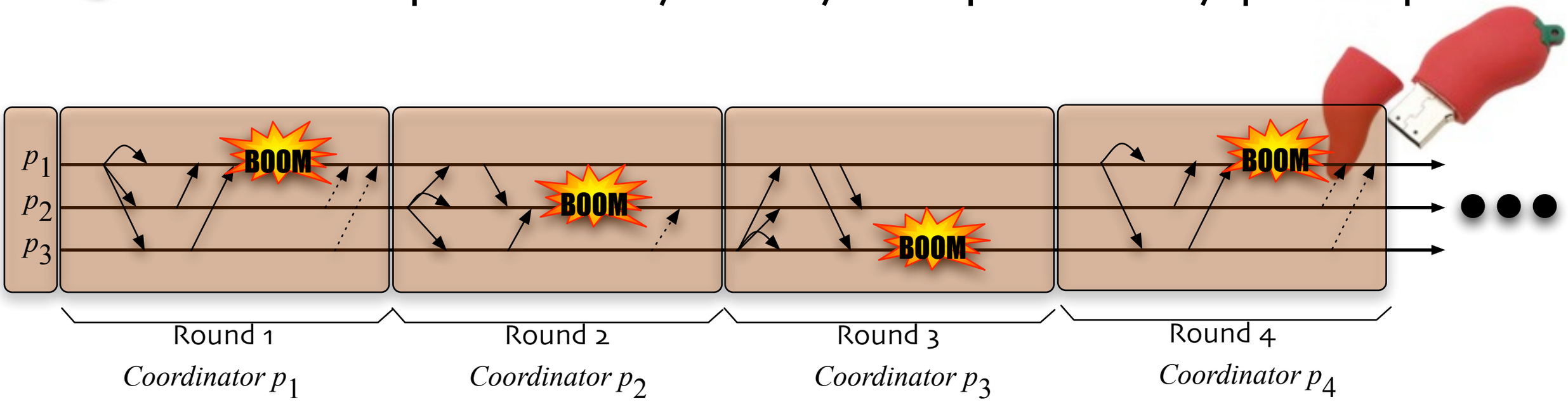
- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant



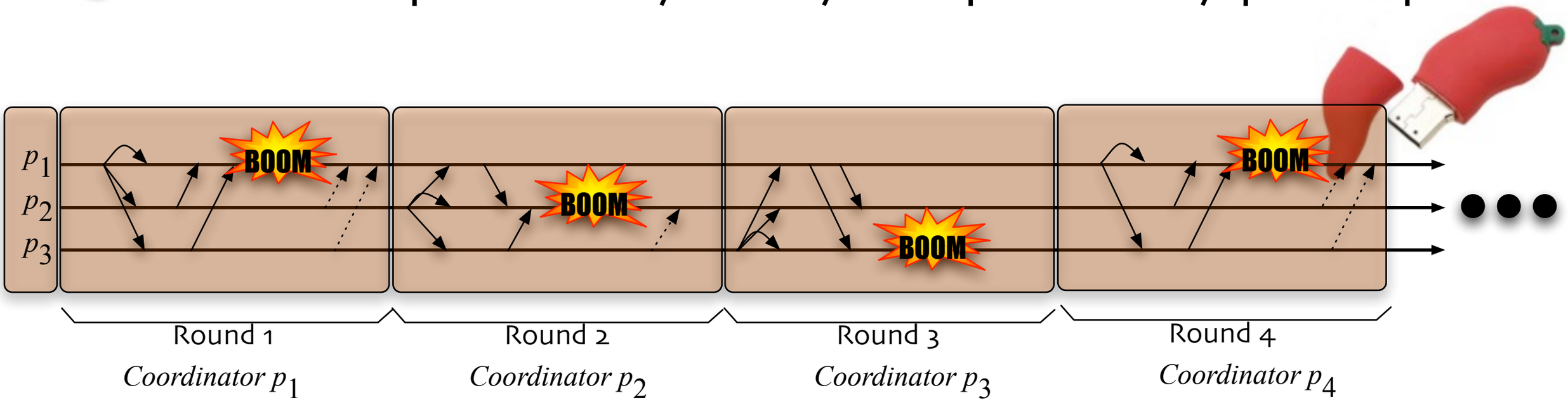
- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant



- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant

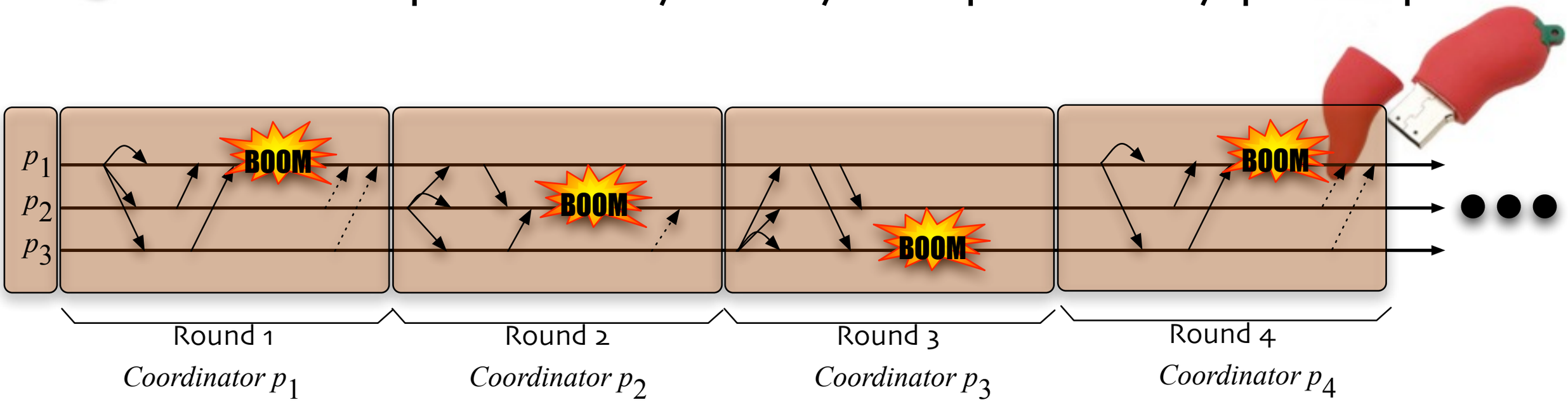


- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant

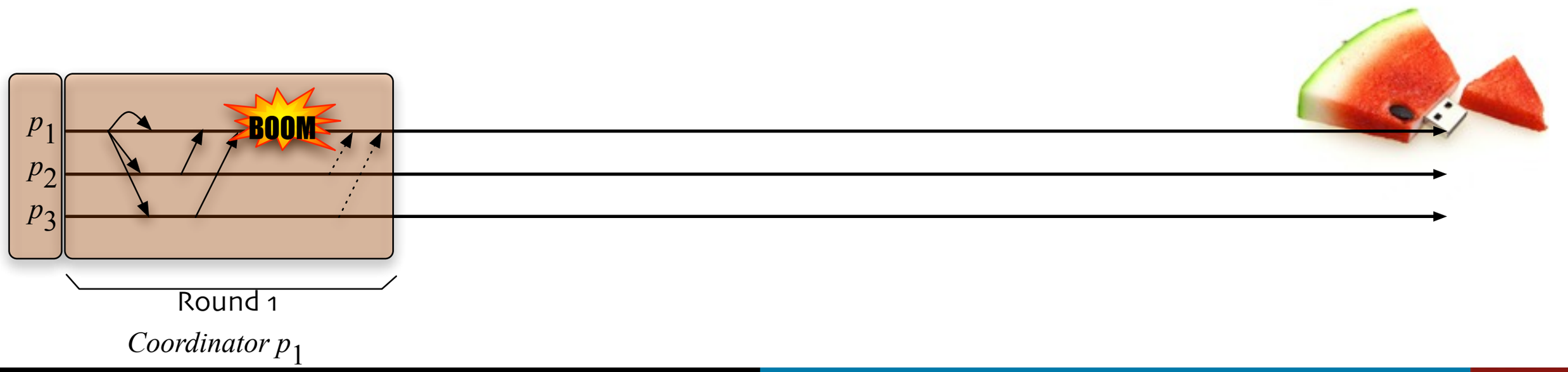


- Or, when inquired they never suspect any participant

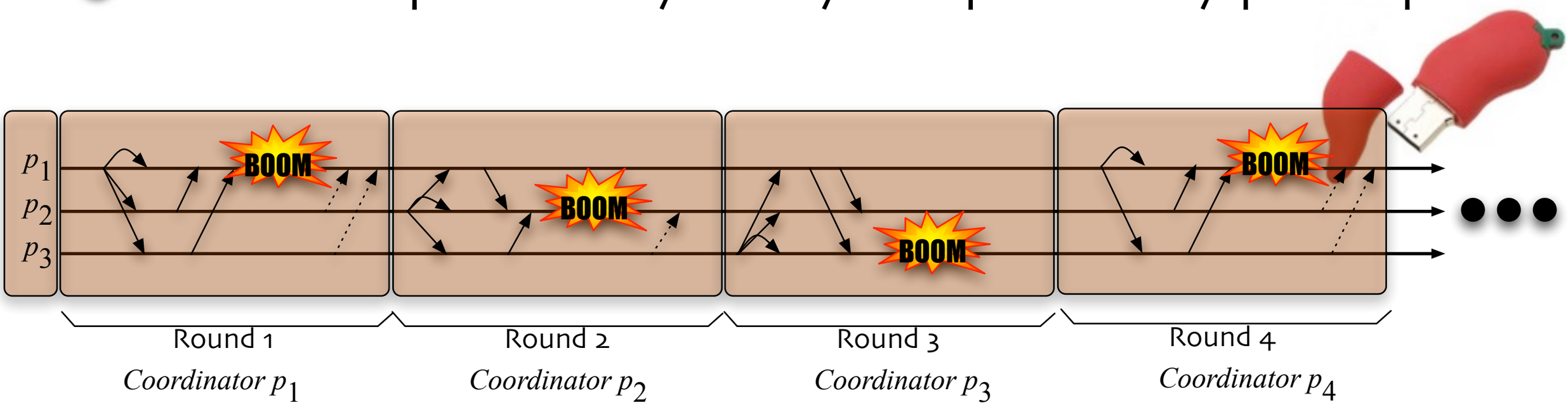
- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant



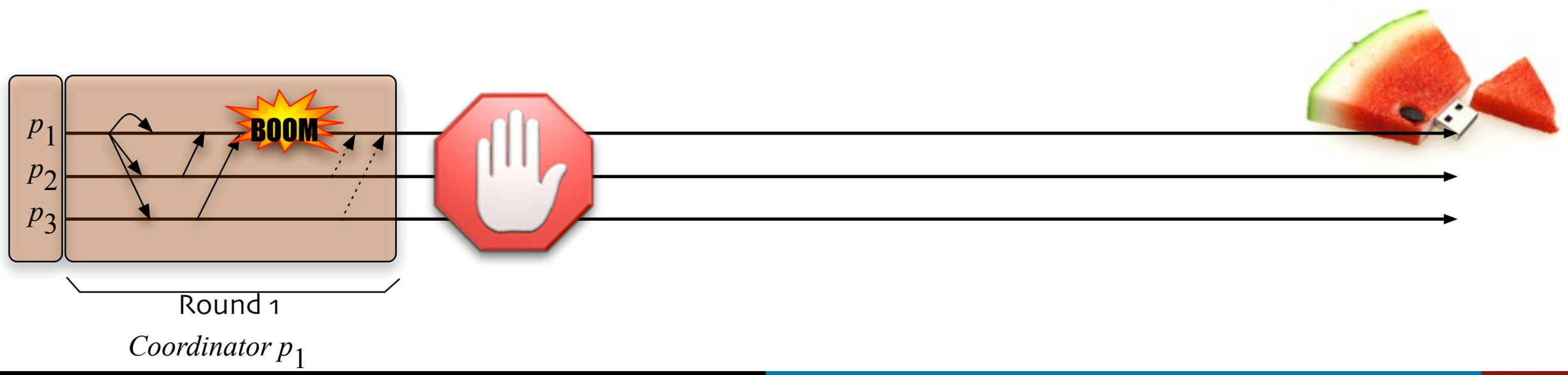
- Or, when inquired they never suspect any participant



- Admit one gets the cheapest set of FD modules
  - When inquired they always suspect every participant



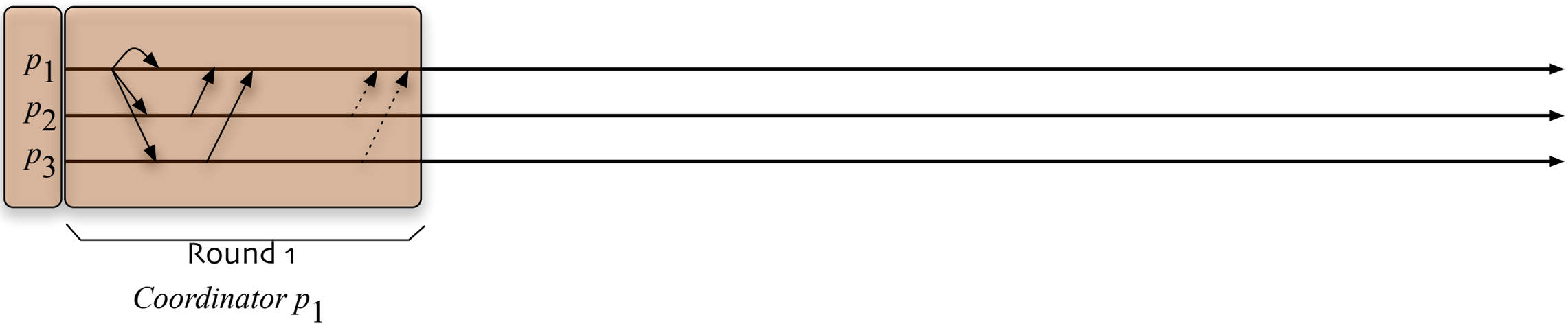
- Or, when inquired they never suspect any participant



- Now, admit one gets the most expensive set of FD modules
  - When inquired they never make mistakes!

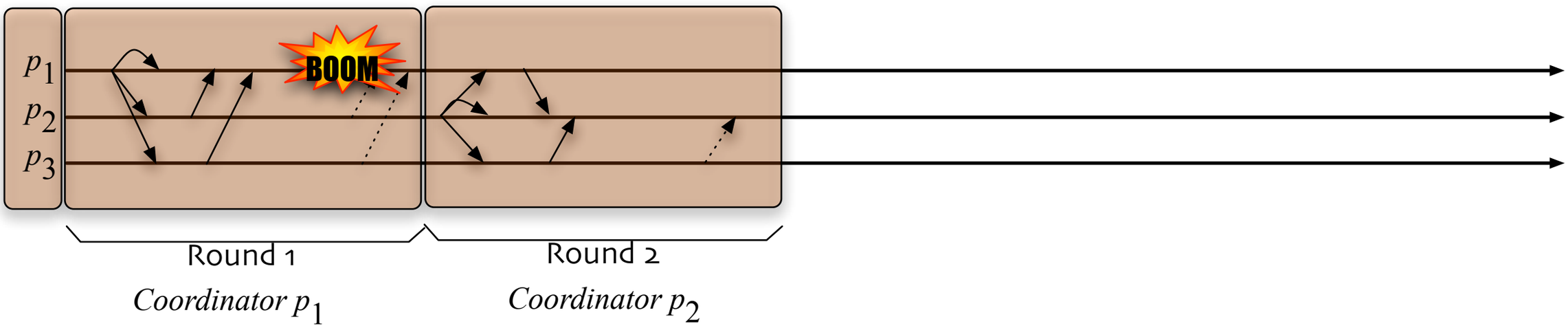
- Now, admit one gets the most expensive set of FD modules
  - When inquired they never make mistakes!

Consensus is reached right on the first round when there are no failures...



- Now, admit one gets the most expensive set of FD modules
  - When inquired they never make mistakes!

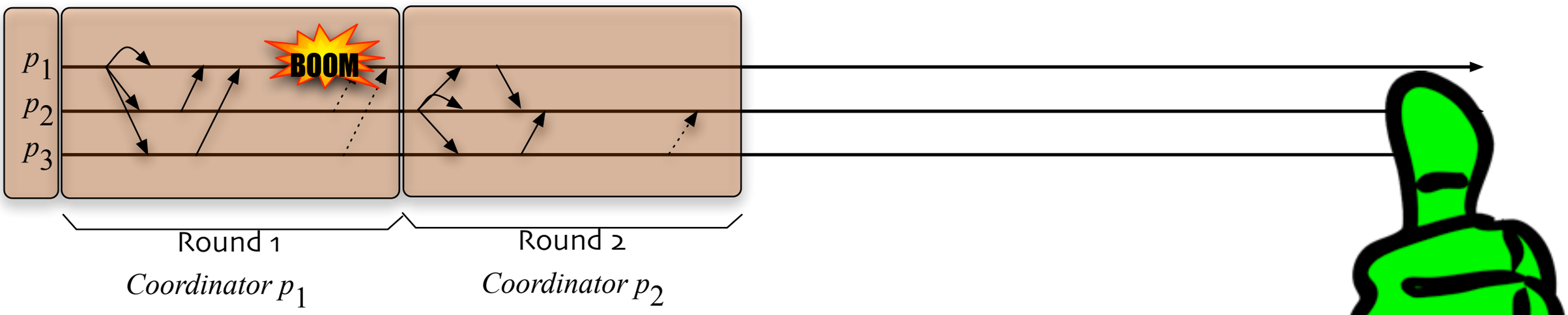
Consensus is reached right on the first round when there are no failures...



... or in the second round should P1 fail.

- Now, admit one gets the most expensive set of FD modules
  - When inquired they never make mistakes!

Consensus is reached right on the first round when there are no failures...



... or in the second round should P1 fail.

Unreliable Failure Detectors Specifications

Unreliable Failure Detectors Specifications

- Completeness

Unreliable Failure Detectors Specifications

- Completeness
  - Strong: Eventually every process that crashes is permanently suspected by every correct process

Unreliable Failure Detectors Specifications

- Completeness
  - Strong: Eventually every process that crashes is permanently suspected by every correct process
  - Weak: Eventually every process that crashes is permanently suspected by some correct process

Unreliable Failure Detectors Specifications

- Completeness
  - Strong: Eventually every process that crashes is permanently suspected by every correct process
  - Weak: Eventually every process that crashes is permanently suspected by some correct process
- Accuracy

Unreliable Failure Detectors Specifications

- Completeness
  - Strong: Eventually every process that crashes is permanently suspected by every correct process
  - Weak: Eventually every process that crashes is permanently suspected by some correct process
- Accuracy
  - Strong: Correct processes are never suspected

Unreliable Failure Detectors Specifications

- Completeness
  - Strong: Eventually every process that crashes is permanently suspected by every correct process
  - Weak: Eventually every process that crashes is permanently suspected by some correct process
- Accuracy
  - Strong: Correct processes are never suspected
  - Weak: Some correct process is never suspected

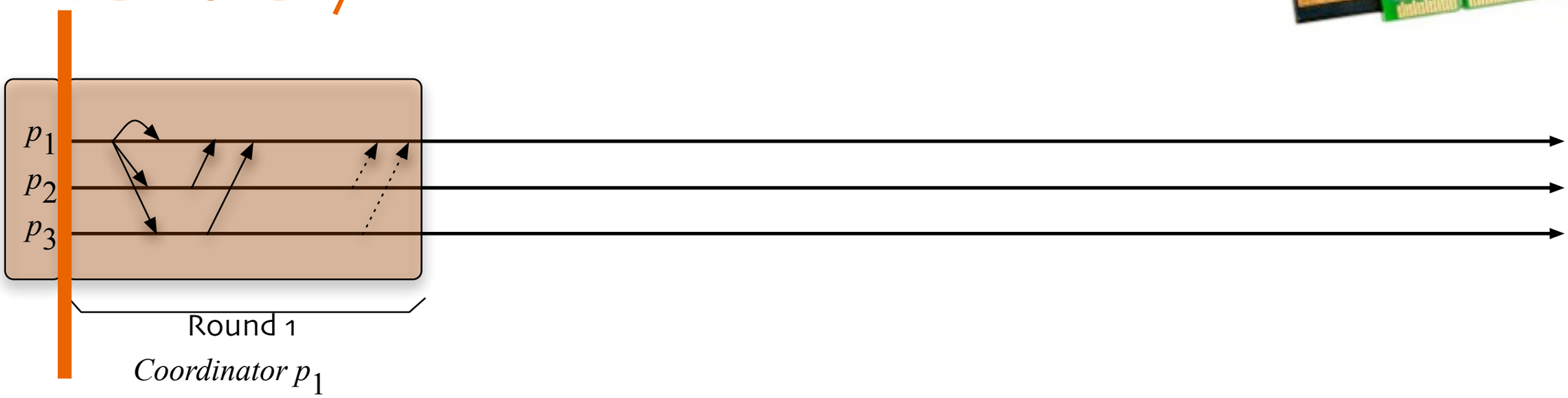
Strong Failure Detector

- Consider a set of FD modules satisfying
  Strong Completeness and
  Weak Accuracy: Some correct process is never suspected

Strong Failure Detector

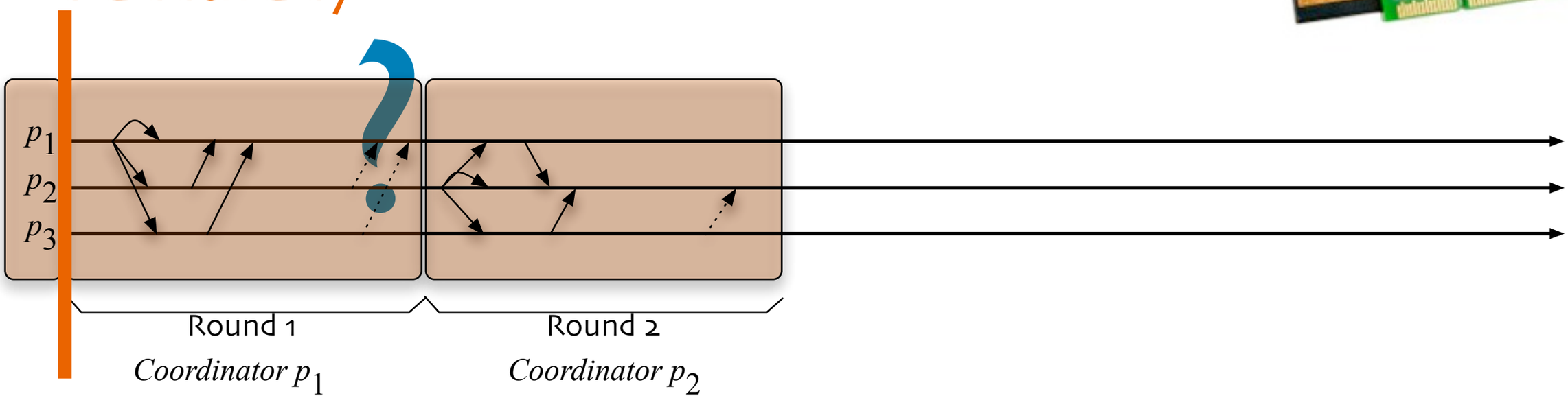
- Consider a set of FD modules satisfying
  Strong Completeness and
  Weak Accuracy: Some correct process is never suspected

t accuracy



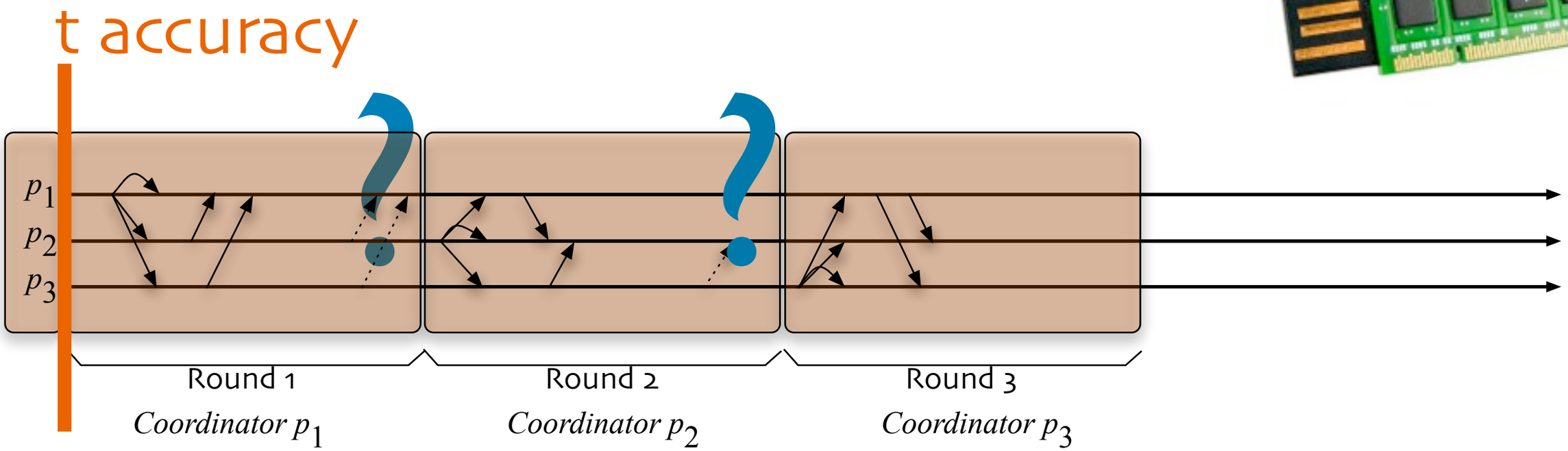
Strong Failure Detector

- Consider a set of FD modules satisfying
  Strong Completeness and
  Weak Accuracy: Some correct process is never suspected



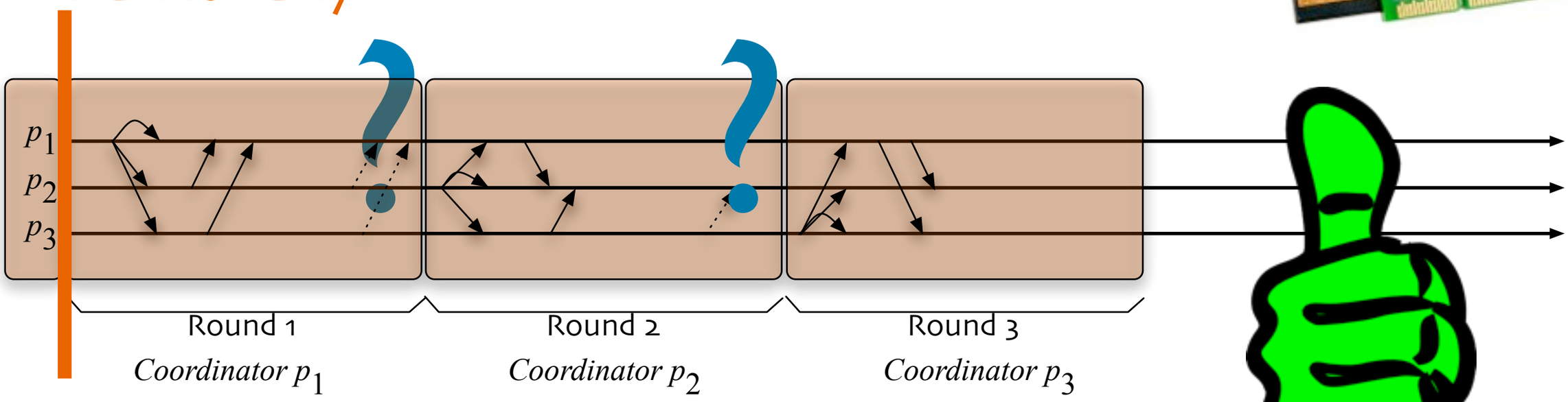
Strong Failure Detector

- Consider a set of FD modules satisfying
  Strong Completeness and
  Weak Accuracy: Some correct process is never suspected



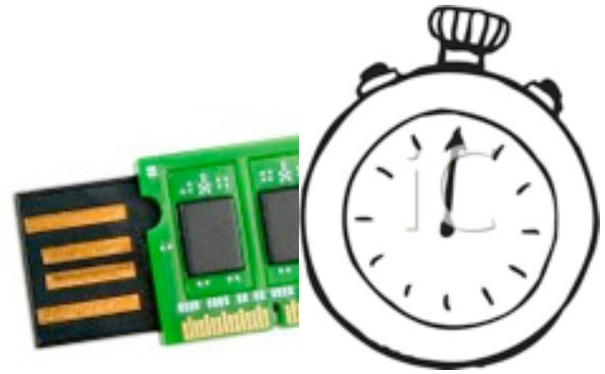
Strong Failure Detector

- Consider a set of FD modules satisfying
  Strong Completeness and
  Weak Accuracy: Some correct process is never suspected



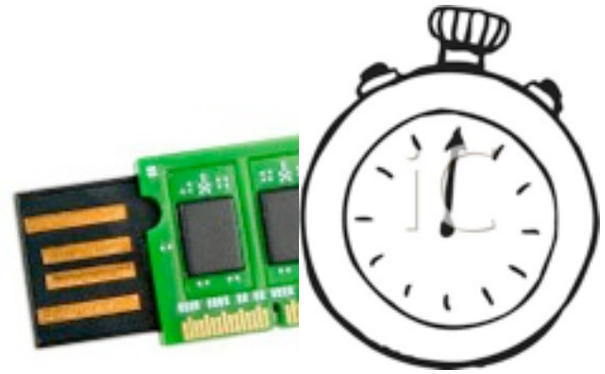
Eventual Strong Failure Detector

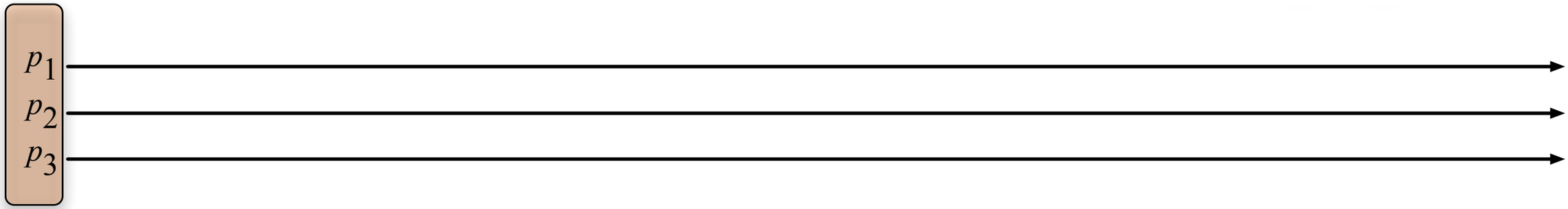
- Now, consider a set of FD modules satisfying Strong Completeness and
  Eventual Weak Accuracy: Eventually some correct process is never suspected



Eventual Strong Failure Detector

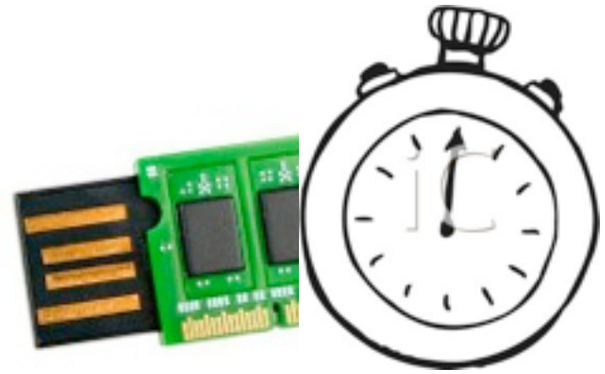
- Now, consider a set of FD modules satisfying Strong Completeness and
  Eventual Weak Accuracy: Eventually some correct process is never suspected

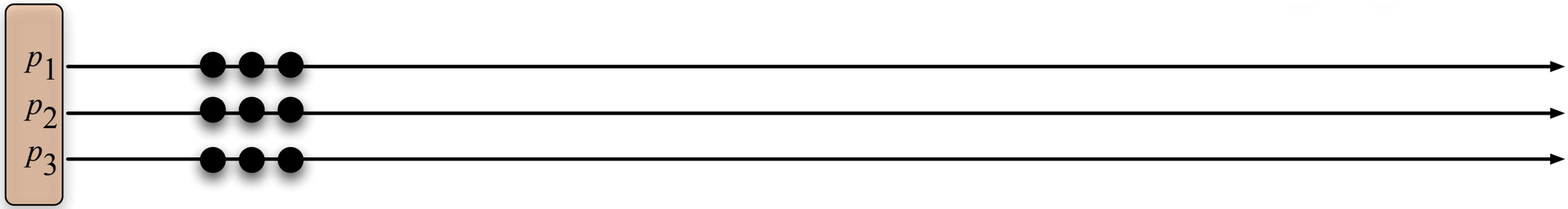




Eventual Strong Failure Detector

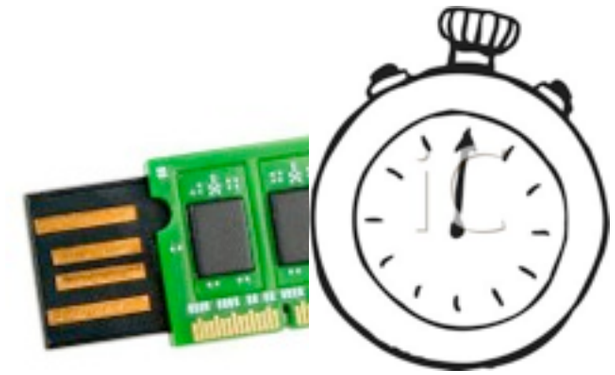
- Now, consider a set of FD modules satisfying Strong Completeness and
  Eventual Weak Accuracy: Eventually some correct process is never suspected

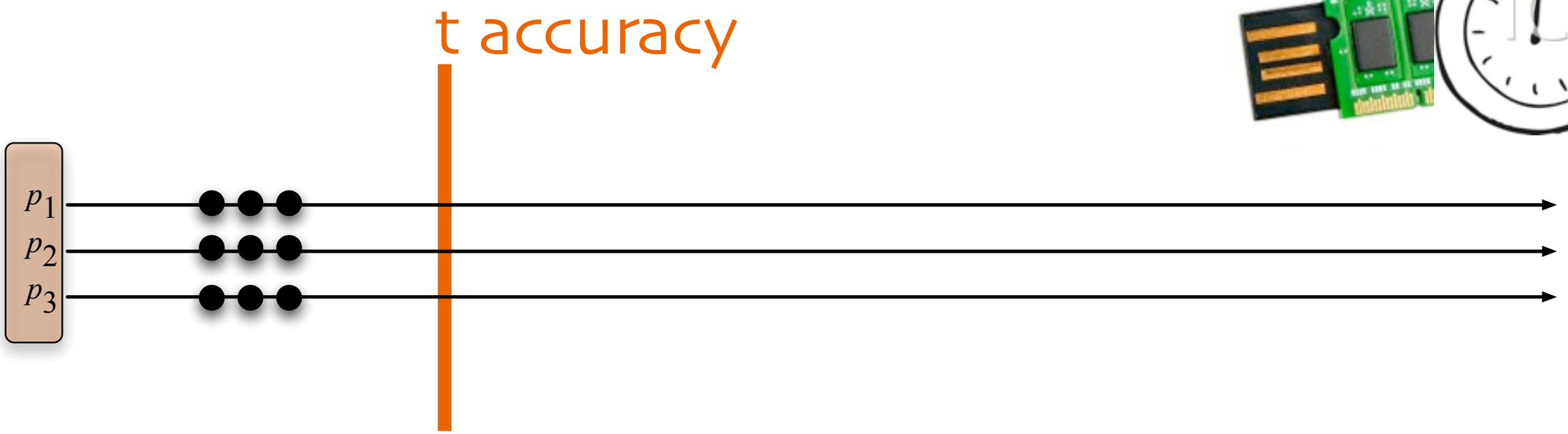




Eventual Strong Failure Detector

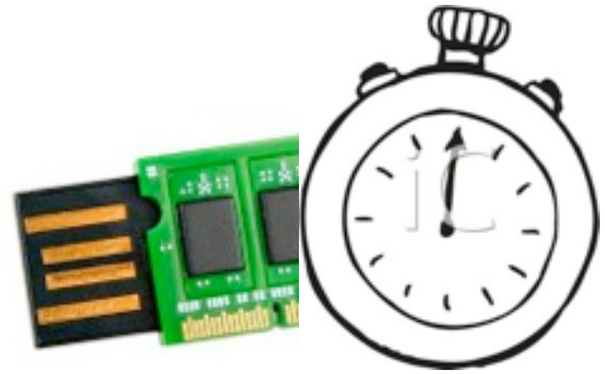
- Now, consider a set of FD modules satisfying Strong Completeness and
  Eventual Weak Accuracy: Eventually some correct process is never suspected

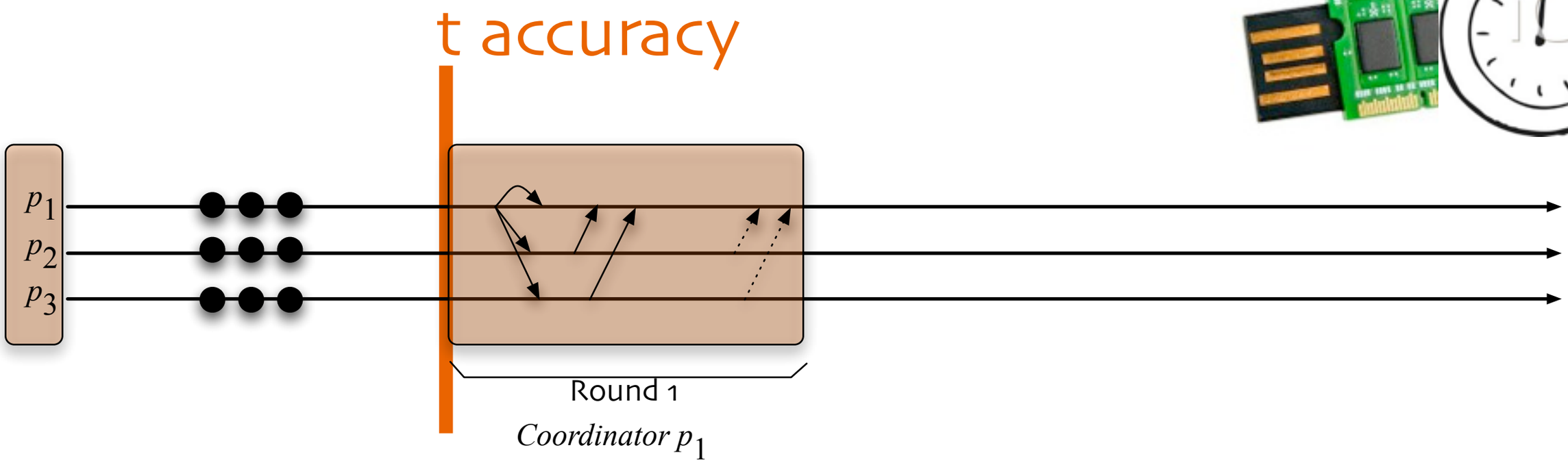




Eventual Strong Failure Detector

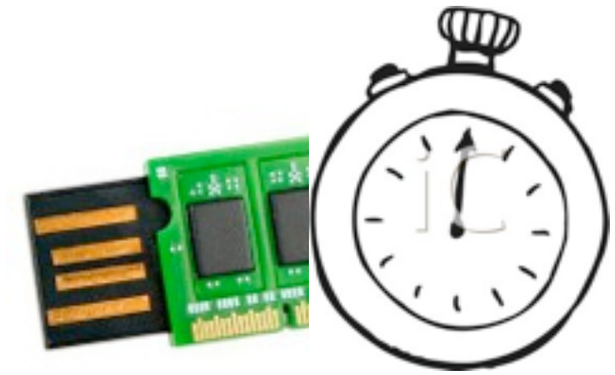
- Now, consider a set of FD modules satisfying Strong Completeness and
  Eventual Weak Accuracy: Eventually some correct process is never suspected

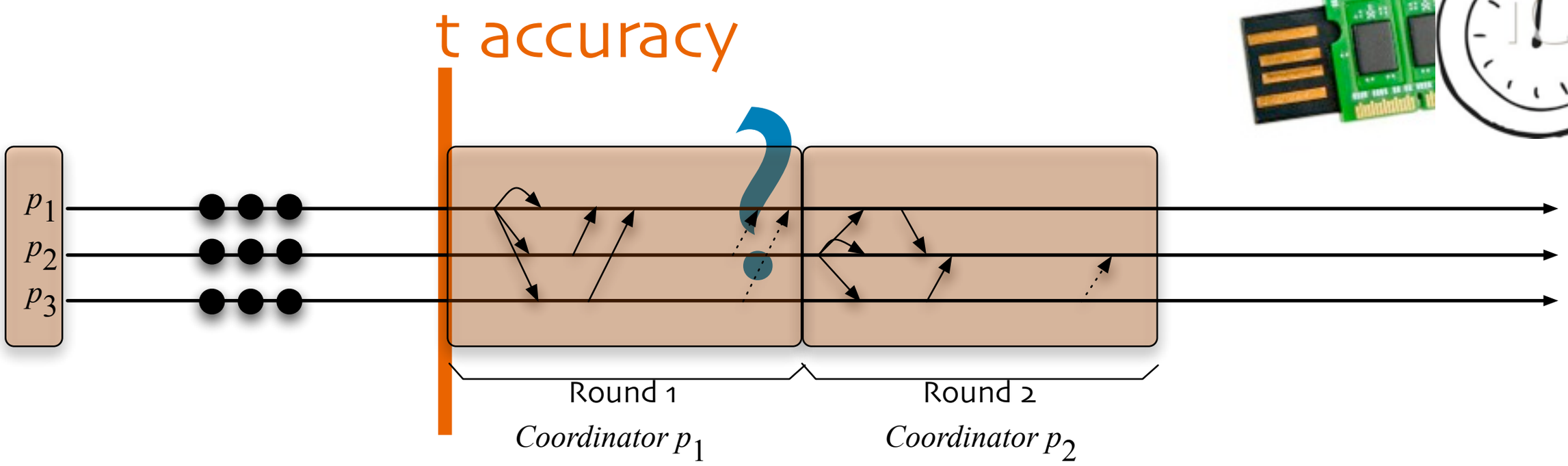




Eventual Strong Failure Detector

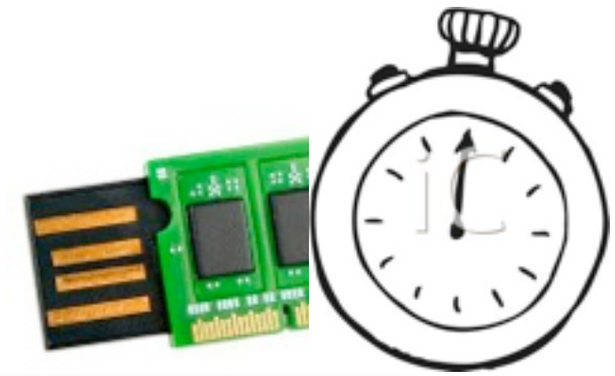
- Now, consider a set of FD modules satisfying Strong Completeness and
  Eventual Weak Accuracy: Eventually some correct process is never suspected

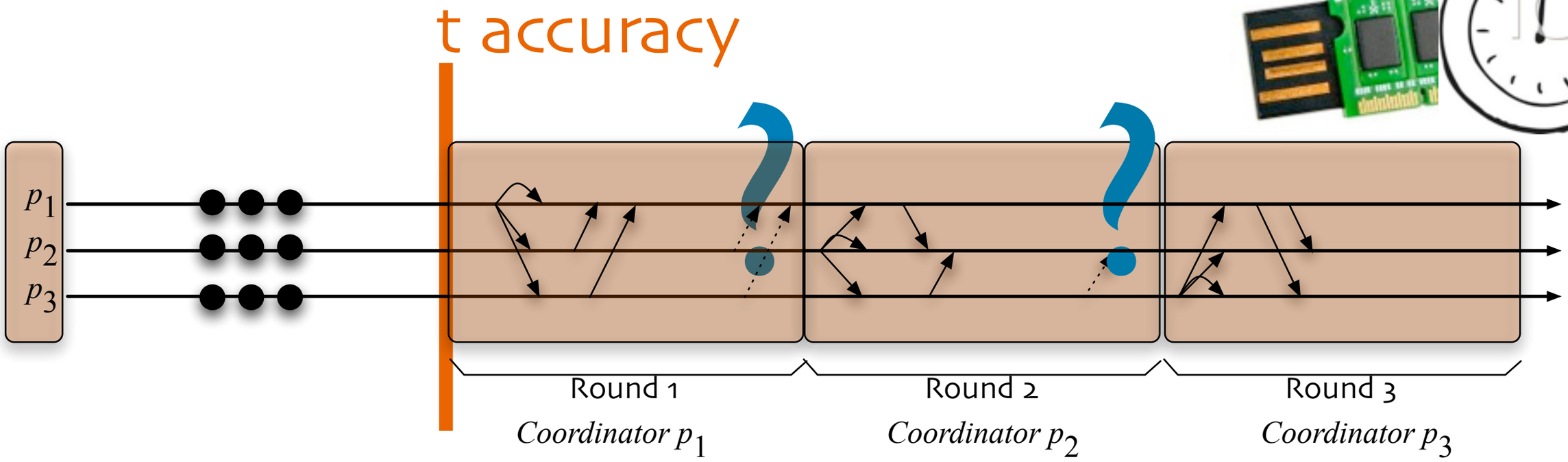




Eventual Strong Failure Detector

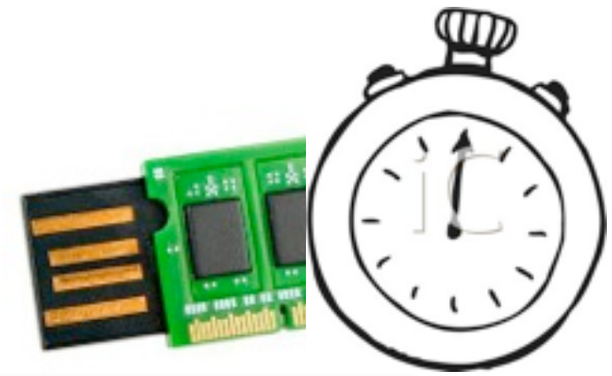
- Now, consider a set of FD modules satisfying Strong Completeness and
  Eventual Weak Accuracy: Eventually some correct process is never suspected





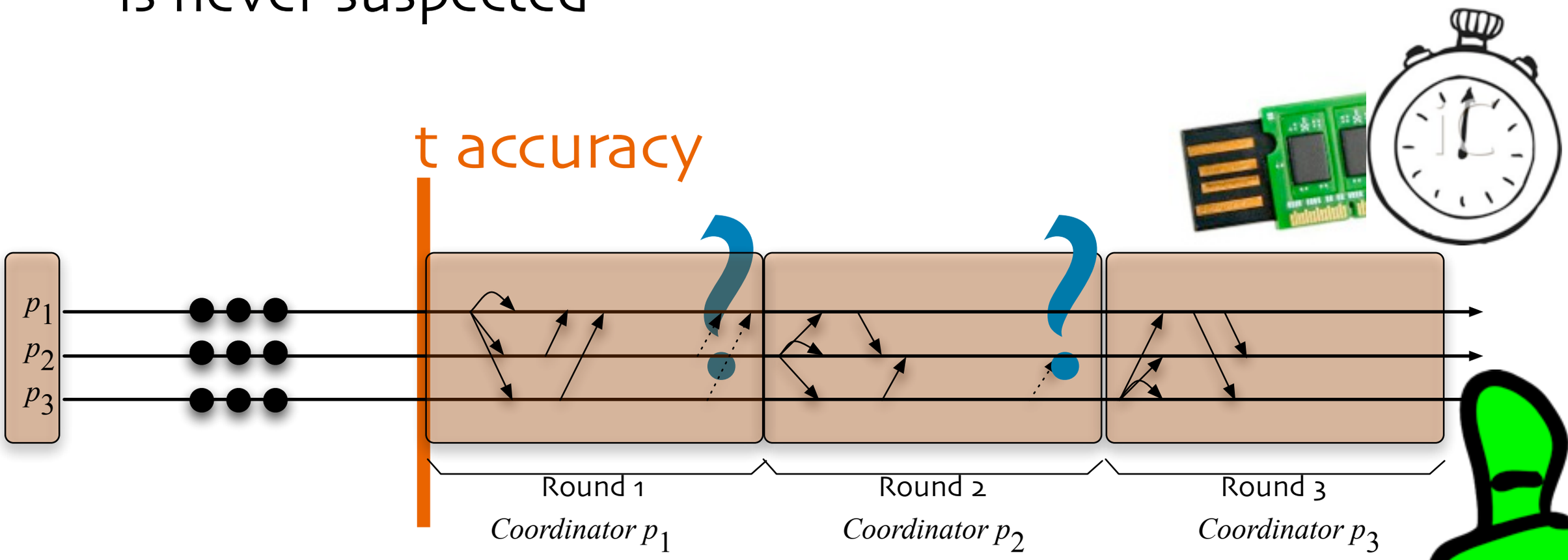
Eventual Strong Failure Detector

- Now, consider a set of FD modules satisfying Strong Completeness and
  Eventual Weak Accuracy: Eventually some correct process is never suspected



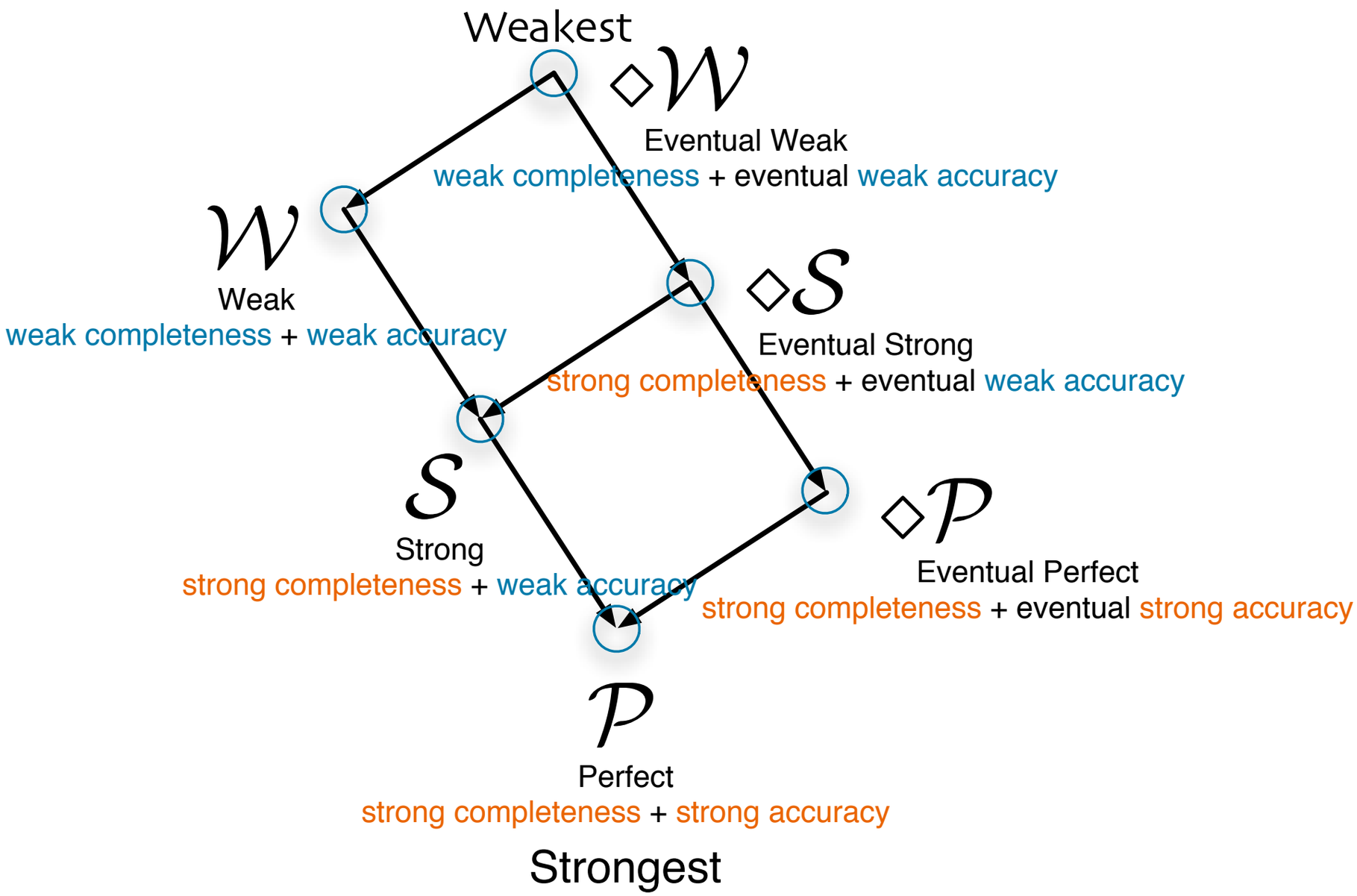
Eventual Strong Failure Detector

- Now, consider a set of FD modules satisfying Strong Completeness and Eventual Weak Accuracy: Eventually some correct process is never suspected



- Why is Strong Completeness required?

A lattice of failure detector classes



[V. Hadzilacos, S. Toueg and T. Chandra, The weakest failure detector for solving consensus, 1996.]

Consensus

Chandra & Toueg Algorithm

procedure $propose(v_p)$
  $estimate_p \leftarrow v_p$ $\{estimate_p \text{ is } p\text{'s estimate of the decision value}\}$
  $state_p \leftarrow undecided$
  $r_p \leftarrow 0$ $\{r_p \text{ is } p\text{'s current round number}\}$
  $ts_p \leftarrow 0$ $\{ts_p \text{ is the last round in which } p \text{ updated } estimate_p, \text{ initially } 0\}$

  $\{Rotate\ through\ coordinators\ until\ decision\ is\ reached\}$

  while $state_p = undecided$
    $r_p \leftarrow r_p + 1$
    $c_p \leftarrow (r_p \bmod n) + 1$ $\{c_p \text{ is the current coordinator}\}$

Consensus

Chandra & Toueg Algorithm

Phase 1: {*All processes p send* $estimate_p$ *to the current coordinator*}
send $(p, r_p, estimate_p, ts_p)$ to c_p

Phase 2: {*The current coordinator gathers* $\lceil \frac{(n+1)}{2} \rceil$ *estimates and proposes a new estimate*}
if $p = c_p$ **then**
wait until [for $\lceil \frac{(n+1)}{2} \rceil$ processes q : received $(q, r_p, estimate_q, ts_q)$ from q]
$msgs_p[r_p] \leftarrow \{(q, r_p, estimate_q, ts_q) \mid p$ received $(q, r_p, estimate_q, ts_q)$ from $q\}$
$t \leftarrow$ largest ts_q such that $(q, r_p, estimate_q, ts_q) \in msgs_p[r_p]$
$estimate_p \leftarrow$ select one $estimate_q$ such that $(q, r_p, estimate_q, t) \in msgs_p[r_p]$
send $(p, r_p, estimate_p)$ to all

Consensus

Chandra & Toueg Algorithm

Phase 3: $\{$*All processes wait for the new estimate proposed by the current coordinator*$\}$

wait until [received $(c_p, r_p, estimate_{c_p})$ from c_p **or** $c_p \in \mathcal{D}_p$]$\{$*Query the failure detector*$\}$

if [received $(c_p, r_p, estimate_{c_p})$ from c_p] **then** $\{p$ *received* $estimate_{c_p}$ *from* $c_p\}$

$estimate_p \leftarrow estimate_{c_p}$

$ts_p \leftarrow r_p$

send (p, r_p, ack) to c_p

else send $(p, r_p, nack)$ to c_p $\{p$ *suspects that* c_p *crashed*$\}$

Phase 4: $\left\{ \begin{array}{l} \textit{The current coordinator waits for } \lceil \frac{(n+1)}{2} \rceil \textit{ replies. If they indicate that } \lceil \frac{(n+1)}{2} \rceil \\ \textit{processes adopted its estimate, the coordinator R-broadcasts a decide message} \end{array} \right\}$

if $p = c_p$ **then**

wait until [for $\lceil \frac{(n+1)}{2} \rceil$ processes q : received (q, r_p, ack) **or** $(q, r_p, nack)$]

if [for $\lceil \frac{(n+1)}{2} \rceil$ processes q : received (q, r_p, ack)] **then**

$R\text{-}broadcast(p, r_p, estimate_p, decide)$

Consensus

Chandra & Toueg Algorithm

$\{If\ p\ R\text{-}delivers\ a\ decide\ message,\ p\ decides\ accordingly\}$

when $R\text{-}deliver(q, r_q, estimate_q, decide)$
  if $state_p = undecided$ **then**
    $decide(estimate_q)$
    $state_p \leftarrow decided$

Discussion on Failure Detection

- How are these 'USB FD modules' implemented?
- Can the model be asynchronous?
- What if the model is not asynchrnous?
  - Timed-asynchronous

    [F. Cristian and C. Fetzer, The Timed Asynchronous Distributed System Model, 1999]
  - Quasi-synchronous, wormholes

    [P. Veríssimo and C. Almeida, Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models, 1995]
  - Synchronous

Consensus

Consensus

- With f < n/2 an Eventual Weak failure detector is both necessary and sufficient to solve Consensus

Consensus

- With f < n/2 an Eventual Weak failure detector is both necessary and sufficient to solve Consensus
  - With f < n/2 the Eventual Weak and Eventual Strong failure detector classes are equivalent

Consensus

- With f < n/2 an Eventual Weak failure detector is both necessary and sufficient to solve Consensus
  - With f < n/2 the Eventual Weak and Eventual Strong failure detector classes are equivalent
  - Omega (= Eventual Weak): There is a time after which all the correct processes always trust the same correct process.

Consensus

- With f < n/2 an Eventual Weak failure detector is both necessary and sufficient to solve Consensus
  - With f < n/2 the Eventual Weak and Eventual Strong failure detector classes are equivalent
  - Omega (= Eventual Weak): There is a time after which all the correct processes always trust the same correct process.

[V. Hadzilacos, S. Toueg and T. Chandra, The weakest failure detector for solving consensus, 1996.]

Consensus

- With f < n/2 an Eventual Weak failure detector is both necessary and sufficient to solve Consensus
  - With f < n/2 the Eventual Weak and Eventual Strong failure detector classes are equivalent
  - Omega (= Eventual Weak): There is a time after which all the correct processes always trust the same correct process.

[V. Hadzilacos, S. Toueg and T. Chandra, The weakest failure detector for solving consensus, 1996.]

Consensus

- With f < n/2 an Eventual Weak failure detector is both necessary and sufficient to solve Consensus
  - With f < n/2 the Eventual Weak and Eventual Strong failure detector classes are equivalent
  - Omega (= Eventual Weak): There is a time after which all the correct processes always trust the same correct process.

[V. Hadzilacos, S. Toueg and T. Chandra, The weakest failure detector for solving consensus, 1996.]

- With f < n a Strong failure detector is necessary and sufficient to solve Consensus

Consensus

- Indulgent algorithms: algorithms, based on a failure detector oracle, that never violate the safety properties. If the failure detector misbehaves the algorithm does not terminate.

- Indulgent algorithms: algorithms, based on a failure detector oracle, that never violate the safety properties. If the failure detector misbehaves the algorithm does not terminate.
- Several algorithms for f < n/2, based on an Eventual Strong failure detector, favor different aspects exploting different communication patterns:

Consensus

- Indulgent algorithms: algorithms, based on a failure detector oracle, that never violate the safety properties. If the failure detector misbehaves the algorithm does not terminate.
- Several algorithms for f < n/2, based on an Eventual Strong failure detector, favor different aspects exploting different communication patterns:

Centralized: [T. Chandra and S.Toueg, Unreliable failure detectors for reliable distributed systems, 1996]

Consensus

- Indulgent algorithms: algorithms, based on a failure detector oracle, that never violate the safety properties. If the failure detector misbehaves the algorithm does not terminate.
- Several algorithms for f < n/2, based on an Eventual Strong failure detector, favor different aspects exploting different communication patterns:

Centralized: [T. Chandra and S.Toueg, Unreliable failure detectors for reliable distributed systems, 1996]

Distributed: [A. Schiper, Early Consensus in an Asynchronous System with a Weak Failure Detector, 1997]

Consensus

- Indulgent algorithms: algorithms, based on a failure detector oracle, that never violate the safety properties. If the failure detector misbehaves the algorithm does not terminate.
- Several algorithms for f < n/2, based on an Eventual Strong failure detector, favor different aspects exploting different communication patterns:

Centralized: [T. Chandra and S.Toueg, Unreliable failure detectors for reliable distributed systems, 1996]

Distributed: [A. Schiper, Early Consensus in an Asynchronous System with a Weak Failure Detector, 1997]

Mutable: [J. Pereira and R. oliveira, The Mutable Consensus Protocol, 2004]

Consensus

- Indulgent algorithms: algorithms, based on a failure detector oracle, that never violate the safety properties. If the failure detector misbehaves the algorithm does not terminate.
- Several algorithms for f < n/2, based on an Eventual Strong failure detector, favor different aspects exploting different communication patterns:

Centralized: [T. Chandra and S.Toueg, Unreliable failure detectors for reliable distributed systems, 1996]

Distributed: [A. Schiper, Early Consensus in an Asynchronous System with a Weak Failure Detector, 1997]

Mutable: [J. Pereira and R. oliveira, The Mutable Consensus Protocol, 2004]

Consensus

- Indulgent algorithms: algorithms, based on a failure detector oracle, that never violate the safety properties. If the failure detector misbehaves the algorithm does not terminate.
- Several algorithms for f < n/2, based on an Eventual Strong failure detector, favor different aspects exploting different communication patterns:

Centralized: [T. Chandra and S.Toueg, Unreliable failure detectors for reliable distributed systems, 1996]

Distributed: [A. Schiper, Early Consensus in an Asynchronous System with a Weak Failure Detector, 1997]

Mutable: [J. Pereira and R. oliveira, The Mutable Consensus Protocol, 2004]