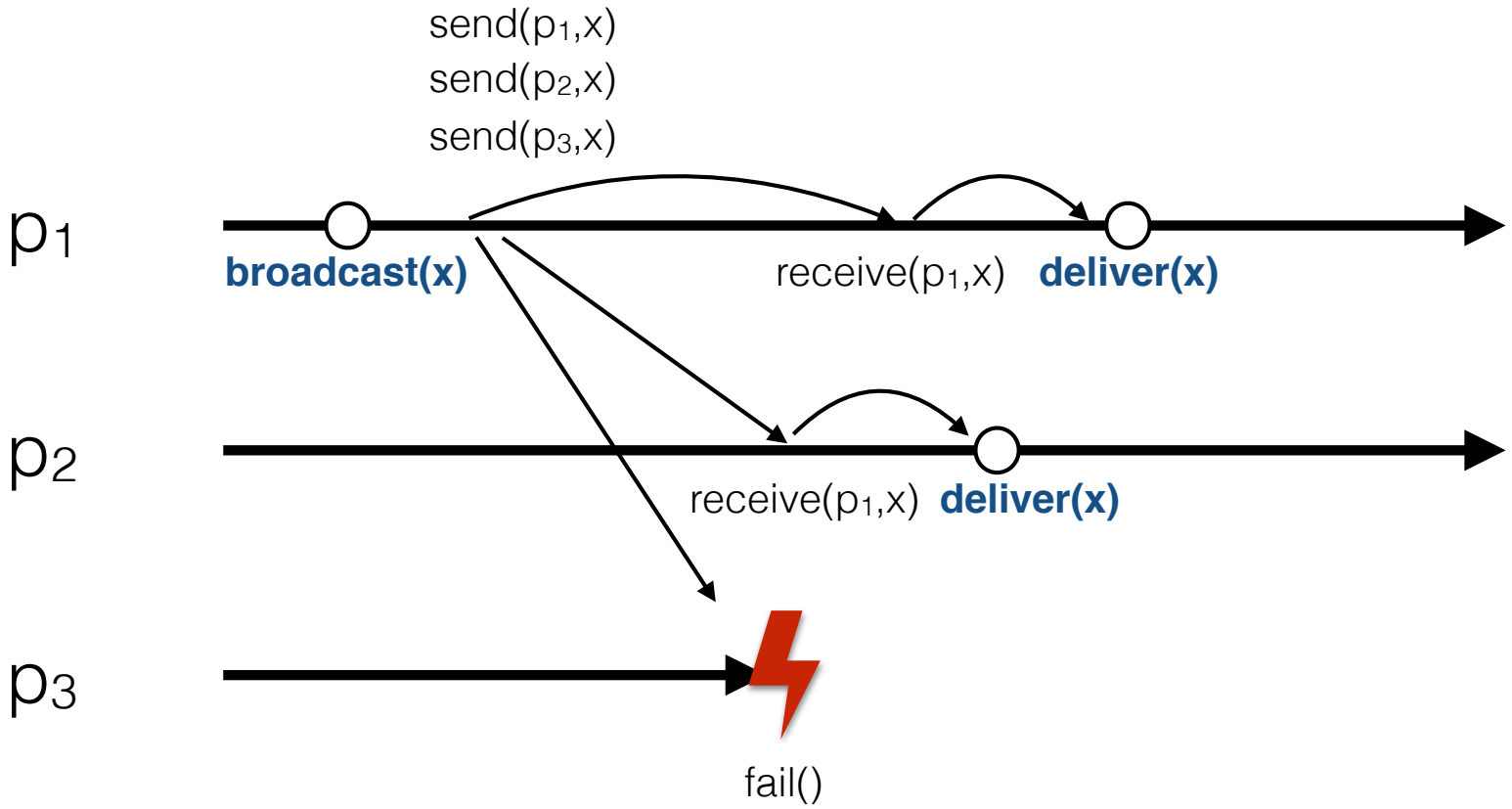
Modelling and validating distributed systems with TLA+

Carla Ferreira

Universidade NOVA de Lisboa

30th April 2019

Best effort broadcast



- For any two correct processes i and j, every message **broadcast** by i is eventually **delivered** by j.
- No message is **delivered** more than once.
- If a correct process j **delivers** a message m, then m was **broadcast** to j by some process i.

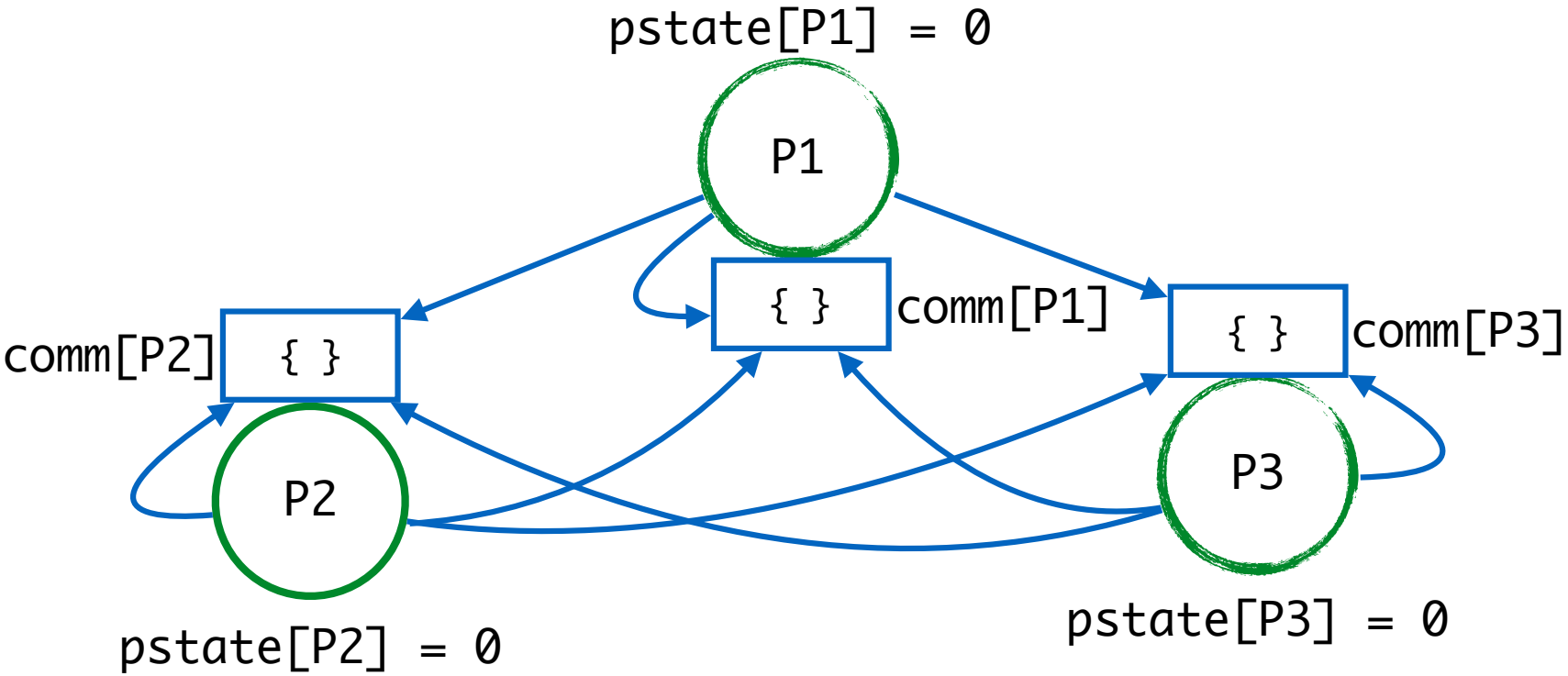
Best effort broadcast

```
---------------------------- MODULE BestEffortBroadcast ----------------------------
EXTENDS Naturals, FiniteSets

CONSTANTS
    Process,        (* set of processes *)
    MaxBroadcasts   (* maximum number of broadcast messages *)

ASSUME
    /\ Process # {}
    /\ MaxBroadcasts > 0


VARIABLES
    comm,           (* point-to-point communication between processes *)
    broadcasted,    (* global set of broadcasted messages *)
    delivered,      (* global set of delivered messages *)
    pstate,         (* process local state *)
    alive,          (* set of alive/active processes *)
    correct         (* set of correct processes *)
```

Best effort broadcast

```
---------------------------- MODULE BestEffortBroadcast ----------------------------
VARIABLES
    comm,           (* point-to-point communication between processes *)
    broadcasted,    (* global set of broadcasted messages *)
    delivered,      (* global set of delivered messages *)
    pstate,         (* process local state *)
    alive,          (* set of alive/active processes *)
    correct         (* set of correct processes *)

Message == [sdr : Process, mid : 0..MaxBroadcasts] (* broadcast message *)

MessageDel == [rcv : Process, msg : Message] (* deliver message *)

TypeInv ==
    /\ comm \in [Process -> SUBSET Message]
    /\ broadcasted \in SUBSET Message
    /\ delivered \in SUBSET MessageDel
    /\ pstate \in [Process -> Nat] (* local state stores the message count *)
    /\ alive \in SUBSET Process
    /\ correct \in SUBSET Process

Init ==
    /\ comm = [p \in Process |-> {}]
    /\ broadcasted = {}
    /\ delivered = {}
    /\ pstate = [p \in Process |-> 0]
    /\ alive = Process
    /\ correct \in SUBSET Process
```

broadcasted	{ }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

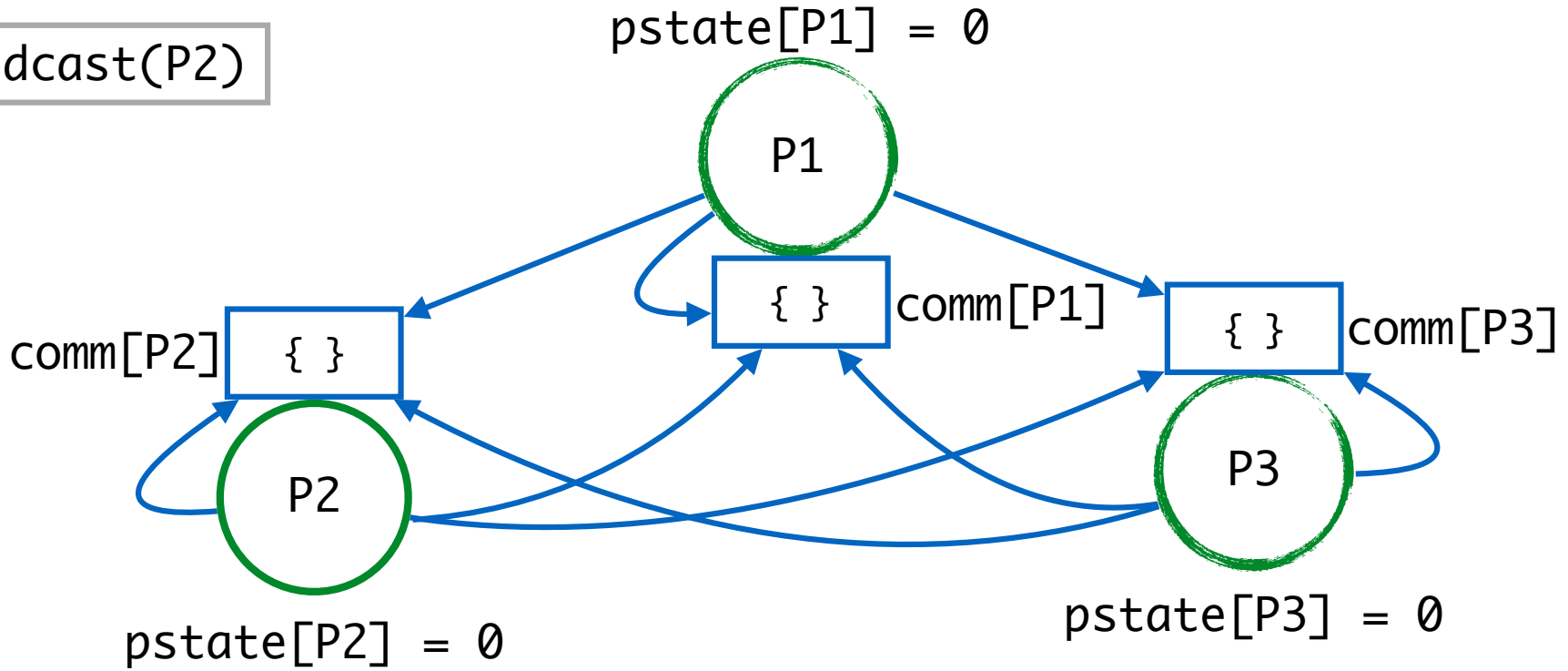


Best effort broadcast

```
---------------------------- MODULE BestEffortBroadcast ----------------------------
TypeInv ==
    /\ comm \in [Process -> SUBSET Message]   /\   broadcasted \in SUBSET Message
    /\ delivered \in SUBSET MessageDel        /\   pstate \in [Process -> Nat]
    /\ alive \in SUBSET Process.              /\   correct \in SUBSET Process
------------------------------------------------------------------------------
broadcast(p) ==
    /\ p \in alive
    /\ Cardinality(broadcasted) < MaxBroadcasts
    /\ LET msg == [sdr |-> p, mid |-> pstate[p]]
       IN
        /\ broadcasted' = broadcasted \union { msg }
        /\ comm'= [q \in Process |-> comm[q] \union { msg }]
        /\ pstate' = [pstate EXCEPT ![p] = pstate[p] + 1]
    /\ UNCHANGED<<delivered,alive,correct>>
```

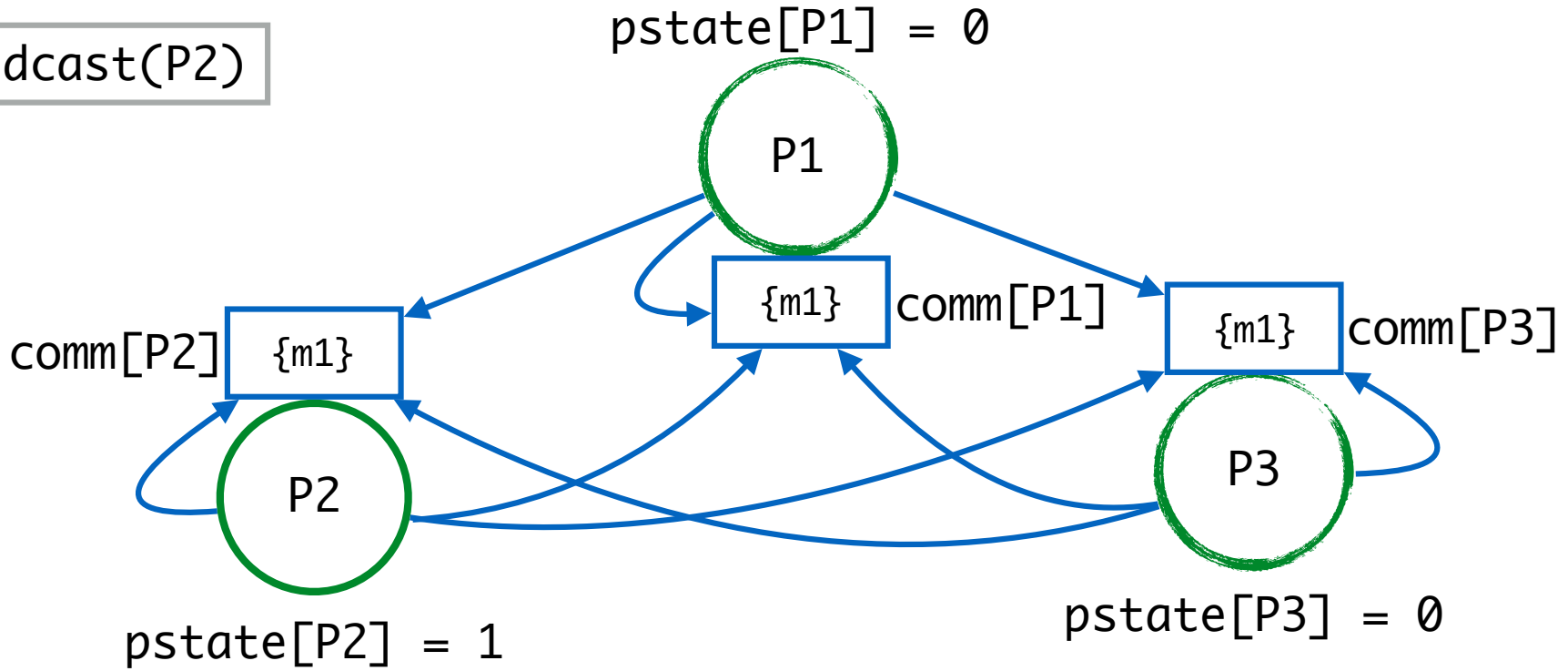
broadcasted	{ }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

broadcast(P2)



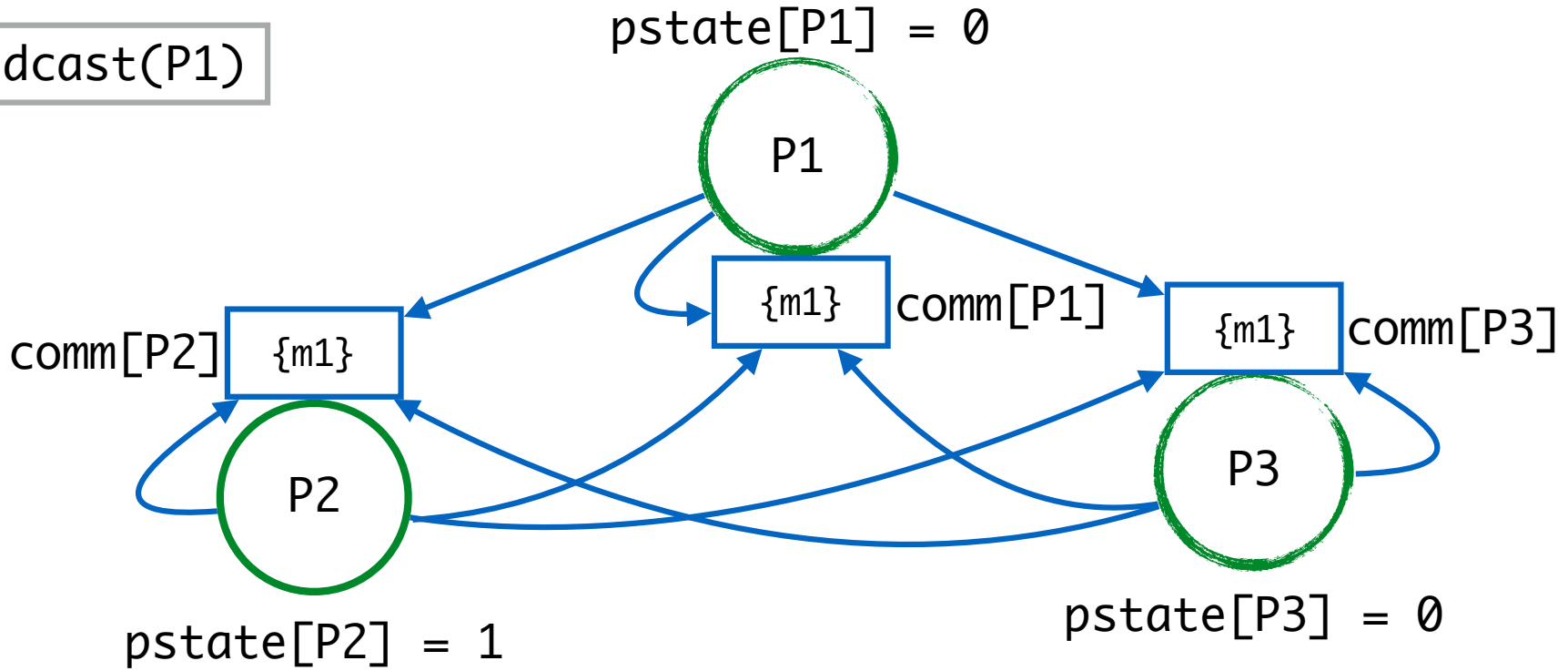
broadcasted	{ m1 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

broadcast(P2)



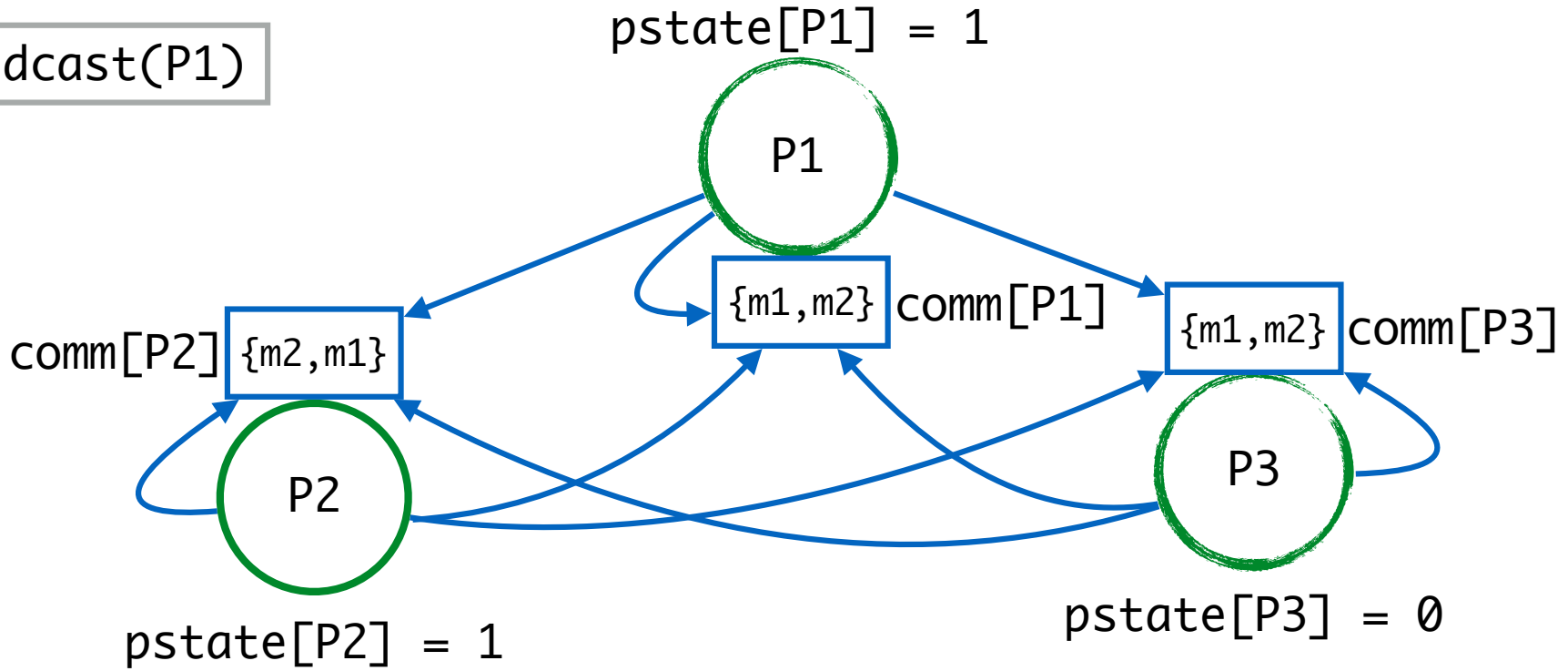
broadcasted	{ m1 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

broadcast(P1)



broadcasted	{ m1, m2 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

broadcast(P1)

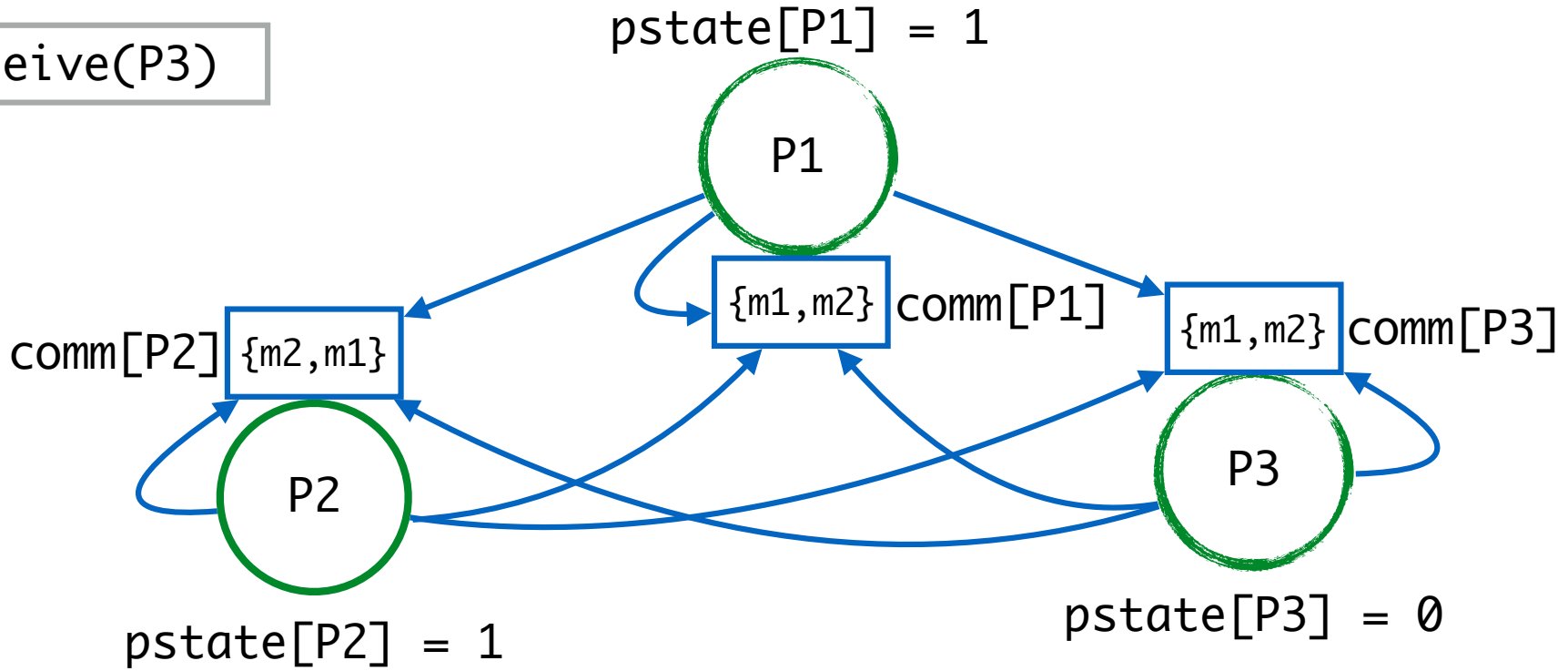


Best effort broadcast

```
---------------------------- MODULE BestEffortBroadcast ----------------------------
TypeInv ==
    /\ comm \in [Process -> SUBSET Message]   /\   broadcasted \in SUBSET Message
    /\ delivered \in SUBSET MessageDel        /\   pstate \in [Process -> Nat]
    /\ alive \in SUBSET Process.              /\   correct \in SUBSET Process
------------------------------------------------------------------------------
receive(p) ==
    /\ p \in alive
    /\ \E m \in comm[p] :
        /\ comm' = [comm EXCEPT ![p] = comm[p] \ {m}]
        /\ delivered' = delivered \union {[rcv |-> p, msg |-> m]}
        /\ UNCHANGED<<broadcasted,alive,pstate,correct>>
```

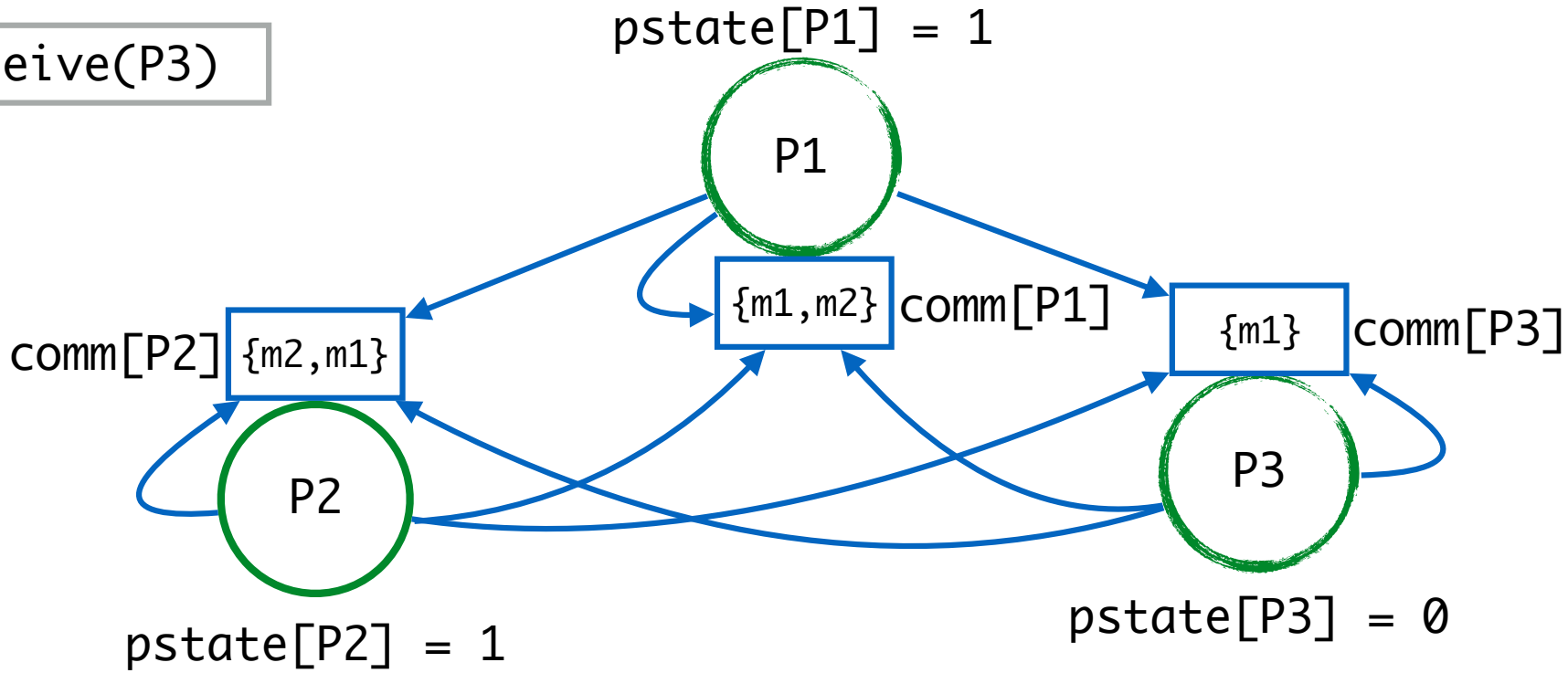
broadcasted	{ m1, m2 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

receive(P3)



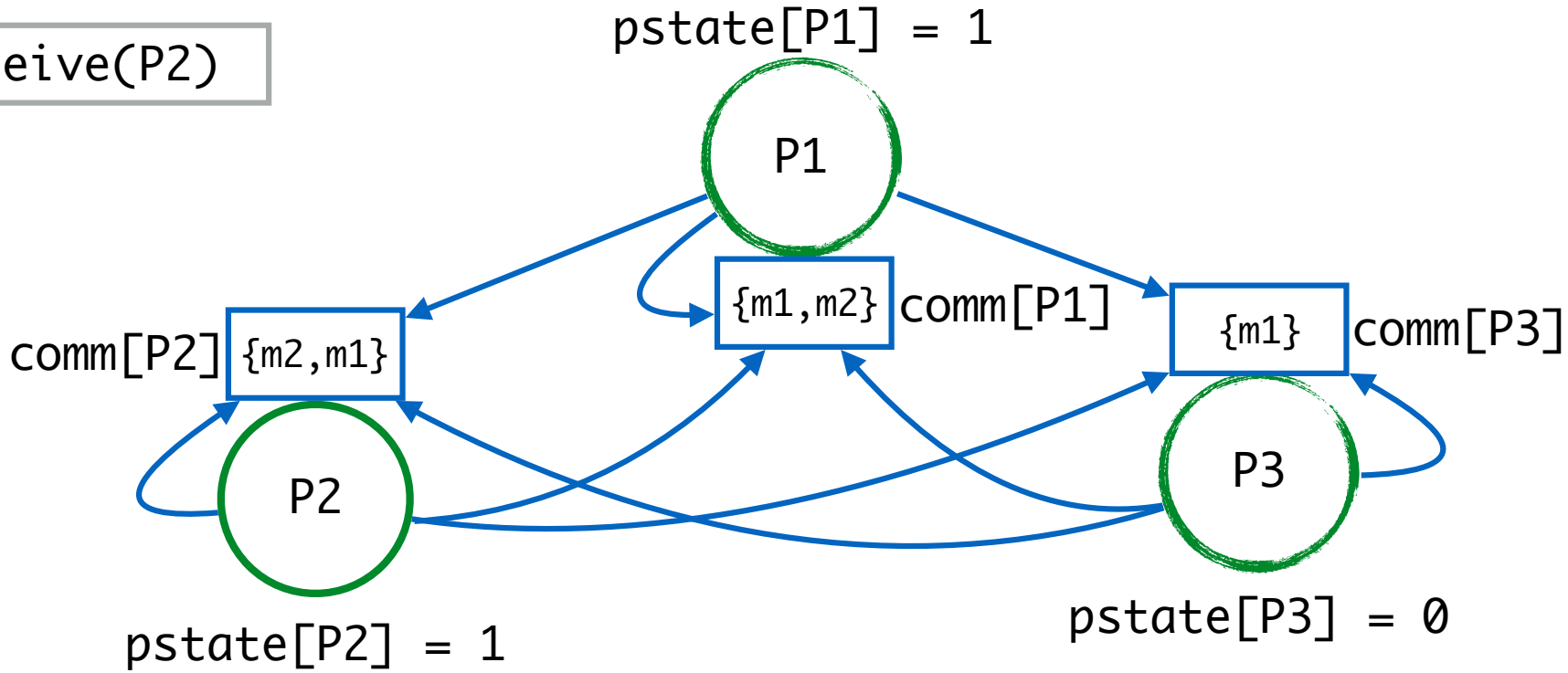
broadcasted	{ m1, m2 }
delivered	{ P3_m2 }
alive	{ P1, P2, P3 }
correct	{ P2 }

receive(P3)



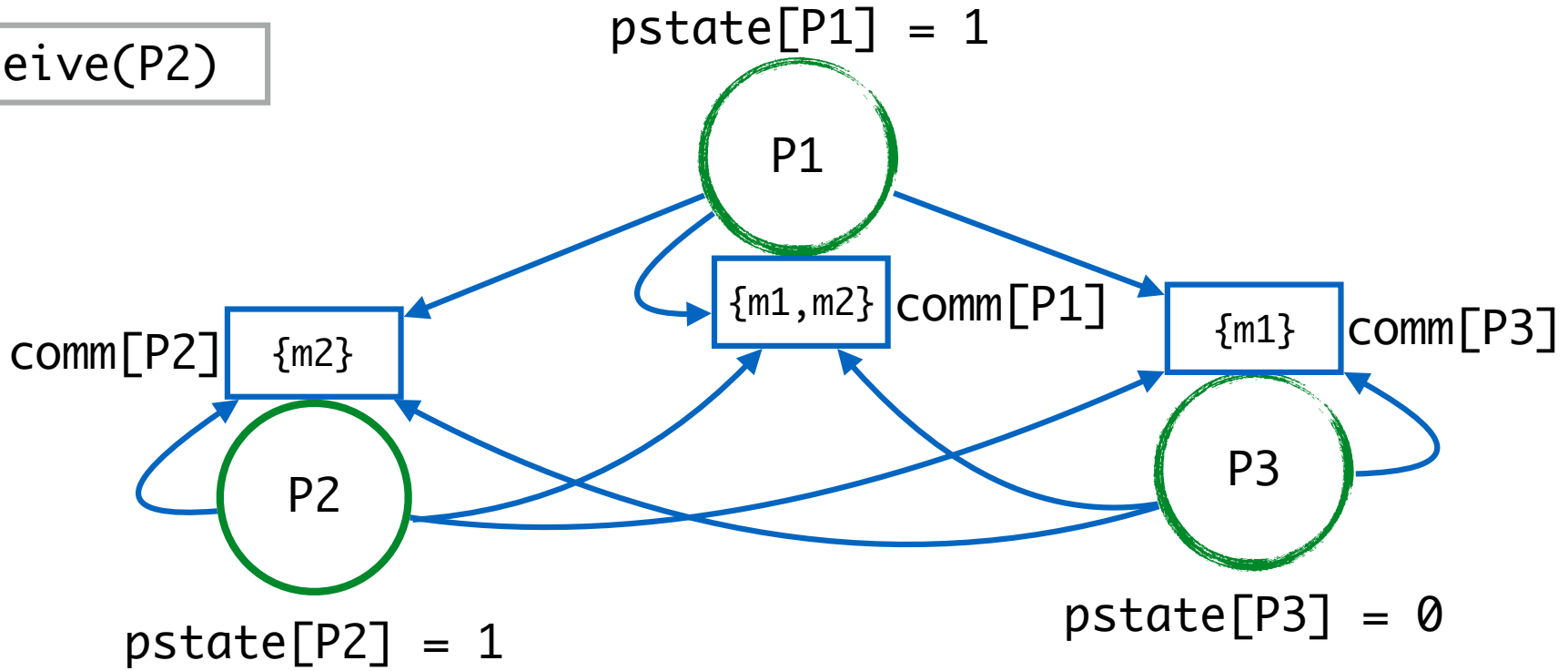
broadcasted	{ m1, m2 }
delivered	{ P3_m2 }
alive	{ P1, P2, P3 }
correct	{ P2 }

receive(P2)



broadcasted	{ m1, m2 }
delivered	{ P2_m1, P3_m2 }
alive	{ P1, P2, P3 }
correct	{ P2 }

receive(P2)

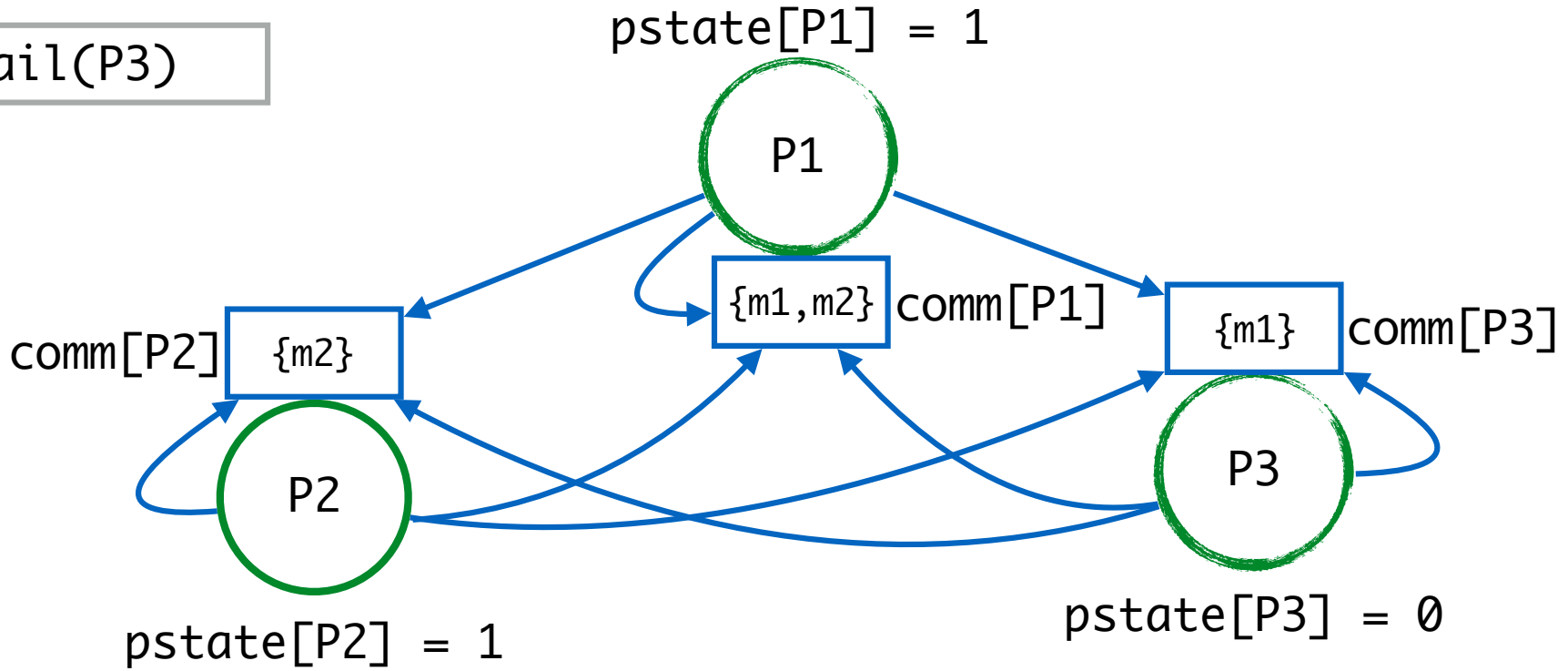


Best effort broadcast

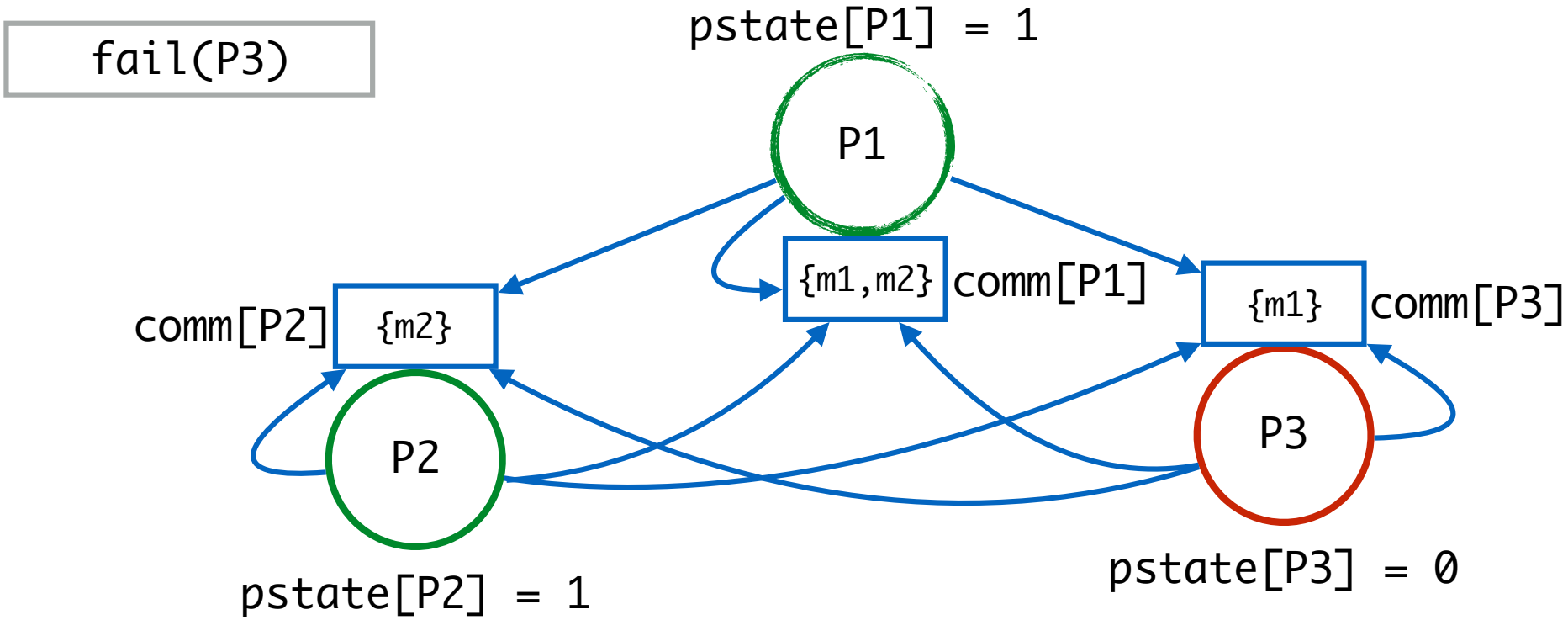
```
---------------------------- MODULE BestEffortBroadcast ----------------------------
TypeInv ==
    /\ comm \in [Process -> SUBSET Message]   /\   broadcasted \in SUBSET Message
    /\ delivered \in SUBSET MessageDel        /\   pstate \in [Process -> Nat]
    /\ alive \in SUBSET Process.              /\   correct \in SUBSET Process
------------------------------------------------------------------------------
fail(p) ==
    /\ p \in alive
    /\ p \notin correct
    /\ alive' = alive \ {p}
    /\ UNCHANGED<<comm,broadcasted,delivered,pstate,correct>>
```

broadcasted	{ m1, m2 }
delivered	{ P2_m1, P3_m2 }
alive	{ P1, P2, P3 }
correct	{ P2 }

fail(P3)



broadcasted	{ m1, m2 }
delivered	{ P2_m1, P3_m2 }
alive	{ P1, P2 }
correct	{ P2 }



Best effort broadcast

```
---------------------------- MODULE BestEffortBroadcast ----------------------------

Next == \E p \in Process : broadcast(p) \/ receive(p) \/ fail(p)

------------------------------------------------------------------------------

Spec == /\ Init
        /\ [][Next]_<<comm, broadcasted, delivered, pstate, alive, correct>>
==============================================================================
```

To model check with TLC this specification we need to define some models:

No. of processes	No. of broadcasted messages	No. of distinct states	Time
2	1	81	14s
3	2	11043	15s
3	5	20387619	5m 47s

Best effort broadcast

```
broadcast(p) ==
    /\ p \in alive
    /\ Cardinality(broadcasted) < MaxBroadcasts
    /\ LET msg == [sdr |-> p, mid |-> pstate[p]]
       IN
        /\ broadcasted' = broadcasted \union { msg }
        /\ comm'= [q \in Process |-> comm[q] \union { msg }]
        /\ pstate' = [pstate EXCEPT ![p] = pstate[p] + 1]
    /\ UNCHANGED<<delivered,alive,correct>>

receive(p) ==
    /\ p \in alive
    /\ \E m \in comm[p] :
        /\ comm' = [comm EXCEPT ![p] = comm[p] \ {m}]
        /\ delivered' = delivered \union {[rcv |-> p, msg |-> m]}
        /\ UNCHANGED<<broadcasted,alive,pstate,correct>>

fail(p) ==
    /\ p \in alive
    /\ p \notin correct
    /\ alive' = alive \ {p}
    /\ UNCHANGED<<comm,broadcasted,delivered,pstate,correct>>
```

Is this a “good” specification of the protocol?

Best effort broadcast (2nd try)

```
---------------------------- MODULE BestEffortBroadcast ----------------------------
EXTENDS Naturals, FiniteSets

CONSTANTS
    Process,        (* set of processes *)
    MaxBroadcasts   (* maximum number of broadcast messages *)

ASSUME
    /\ Process # {}
    /\ MaxBroadcasts > 0


VARIABLES
    comm,           (* point-to-point communication between processes *)
    broadcasted,    (* global set of broadcasted messages *)
    delivered,      (* global set of delivered messages *)
    pstate,         (* process local state *)
    alive,          (* set of alive/active processes *)
    correct         (* set of correct processes *)
```

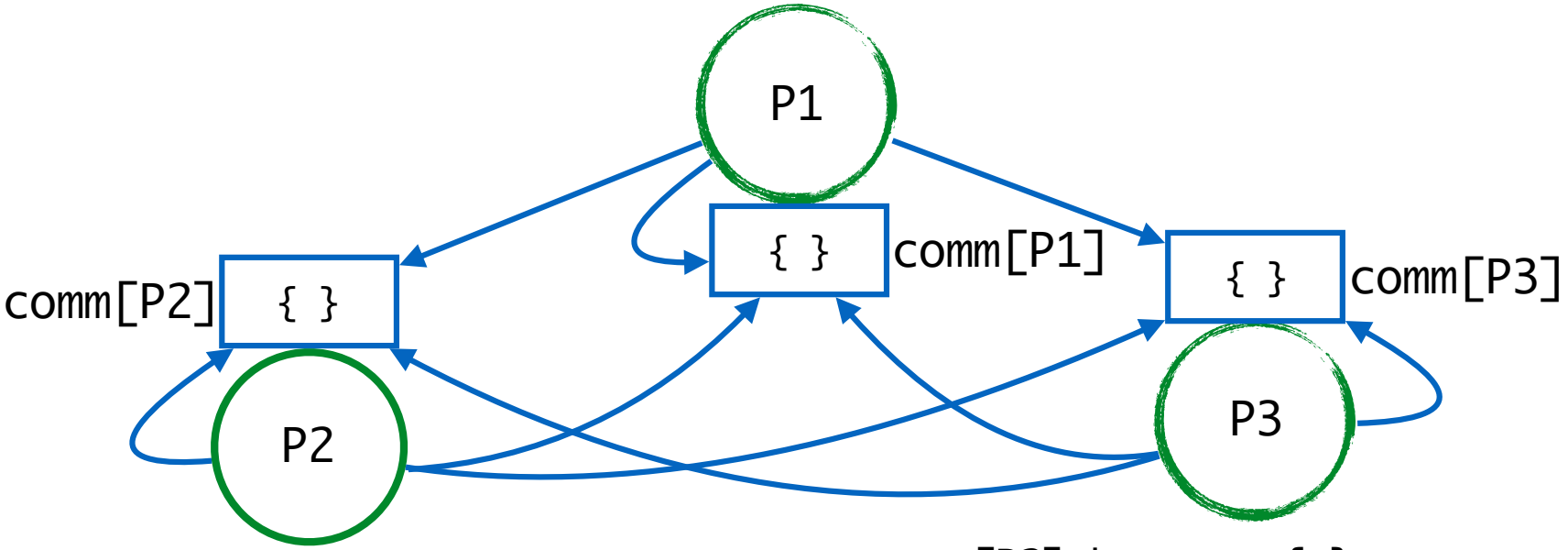
Exactly the same as before!

Best effort broadcast (2nd try)

```
---------------------------- MODULE BestEffortBroadcast ----------------------------
Message == [sdr : Process, mid : 0..MaxBroadcasts]  (* broadcast message *)

MessageDel == [rcv : Process, msg : Message]        (* deliver message *)

                                                    (* process local state *)
LocalState == [bcast: SUBSET Message,               (* messages bradcasted *)
               msent: [Process -> SUBSET Message],  (* messages sent *)
               mcount : Nat]                        (* broacast messages count *)

VARIABLES
    comm, broadcasted, delivered, pstate, alive, correct

TypeInv ==
    /\ comm \in [Process -> SUBSET Message]
    /\ broadcasted \in SUBSET Message
    /\ delivered \in SUBSET MessageDel
    /\ pstate \in [Process -> LocalState]
    /\ alive \in SUBSET Process
    /\ correct \in SUBSET Process

Init ==
    /\ comm = [p \in Process |-> {}]
    /\ broadcasted = {}
    /\ delivered = {}
    /\ pstate = [p \in Process |-> [bcast |-> {},
                                    msent |-> [q \in Process |-> {}],
                                    mcount |-> 0]]
    /\ alive = Process
    /\ correct \in SUBSET Process
```

broadcasted	{ }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

pstate[P1].bcast = { }
pstate[P1].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P1].mcount = 0



pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

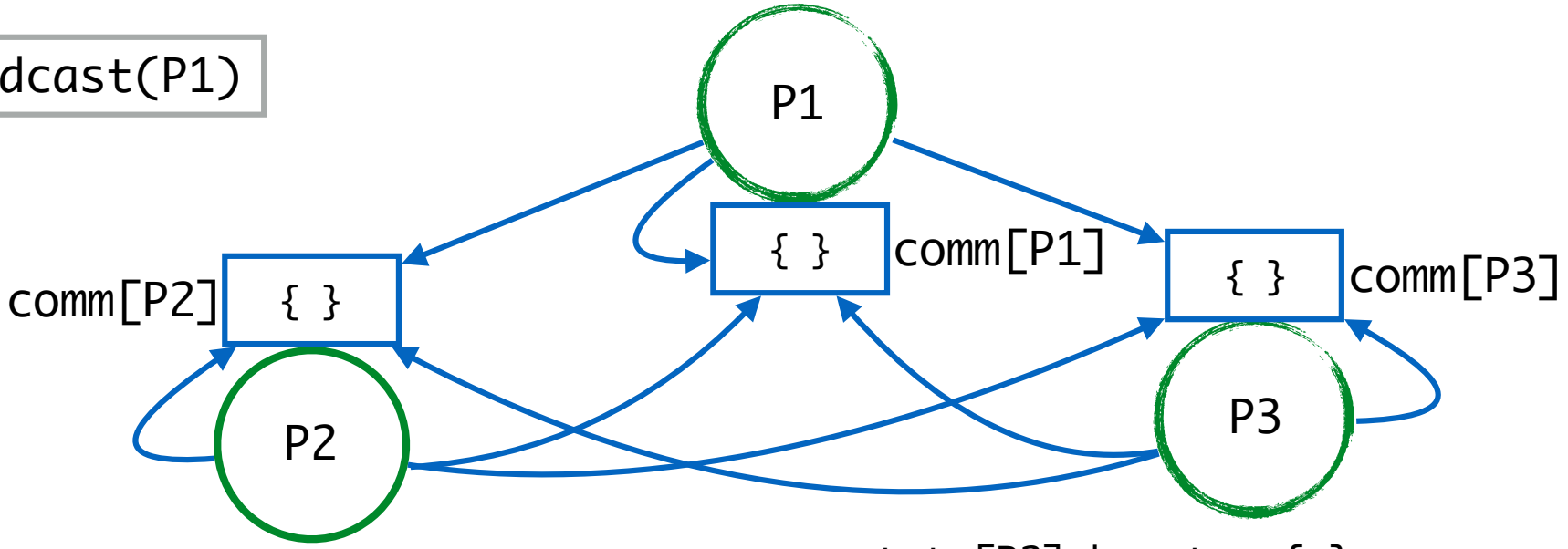
Best effort broadcast (2nd try)

```
---------------------------- MODULE BestEffortBroadcast ----------------------------
TypeInv ==
    /\ comm \in [Process -> SUBSET Message]   /\   broadcasted \in SUBSET Message
    /\ delivered \in SUBSET MessageDel        /\   pstate \in [Process -> LocalState]
    /\ alive \in SUBSET Process.              /\   correct \in SUBSET Process
------------------------------------------------------------------------------------
broadcast(p) ==
    /\ p \in alive
    /\ Cardinality(pstate[p].bcast) < MaxBroadcasts
    /\ LET msg == [sdr |-> p, mid |-> pstate[p].mcount]
       IN
        /\ broadcasted' = broadcasted \union { msg }
        /\ pstate' = [pstate EXCEPT ![p] = [bcast |-> pstate[p].bcast \union {msg},
                                            msent |-> pstate[p].msent,
                                            mcount |-> pstate[p].mcount + 1]]
    /\ UNCHANGED<<comm,delivered,alive,correct>>
```

broadcasted	{ }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

pstate[P1].bcast = { }
pstate[P1].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P1].mcount = 0

broadcast(P1)



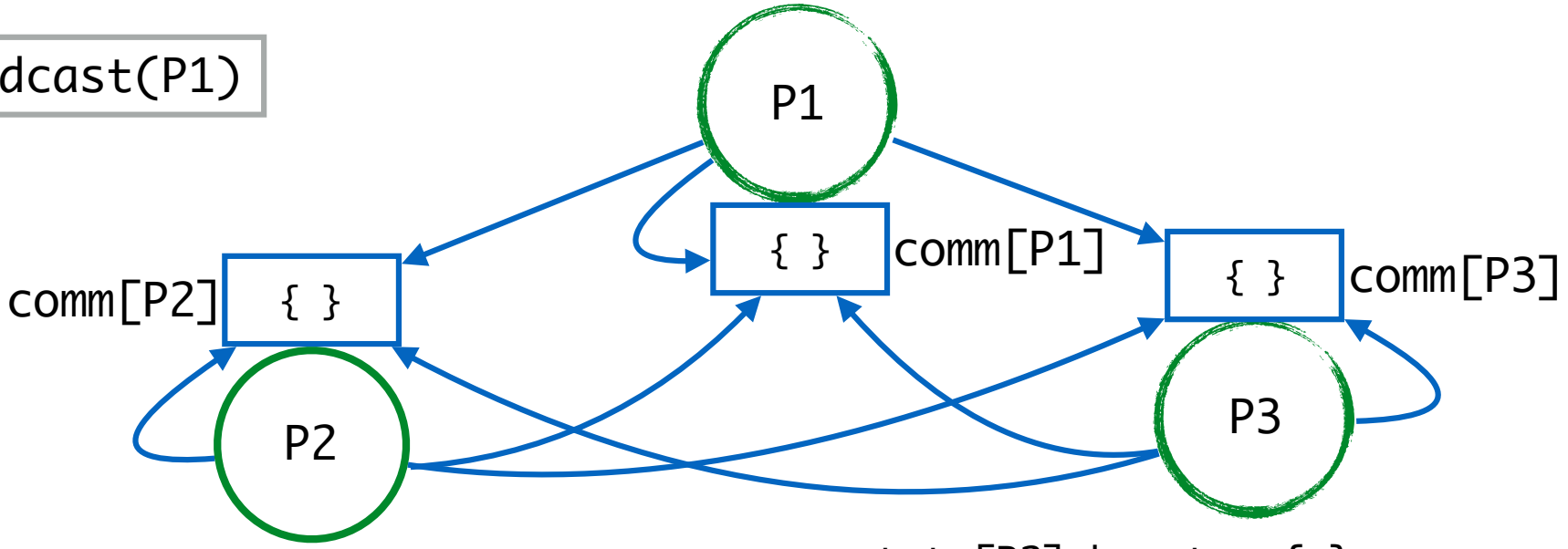
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

broadcasted	{ m1 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

```
pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P1].mcount = 1
```

broadcast(P1)




```
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0
```

```
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0
```

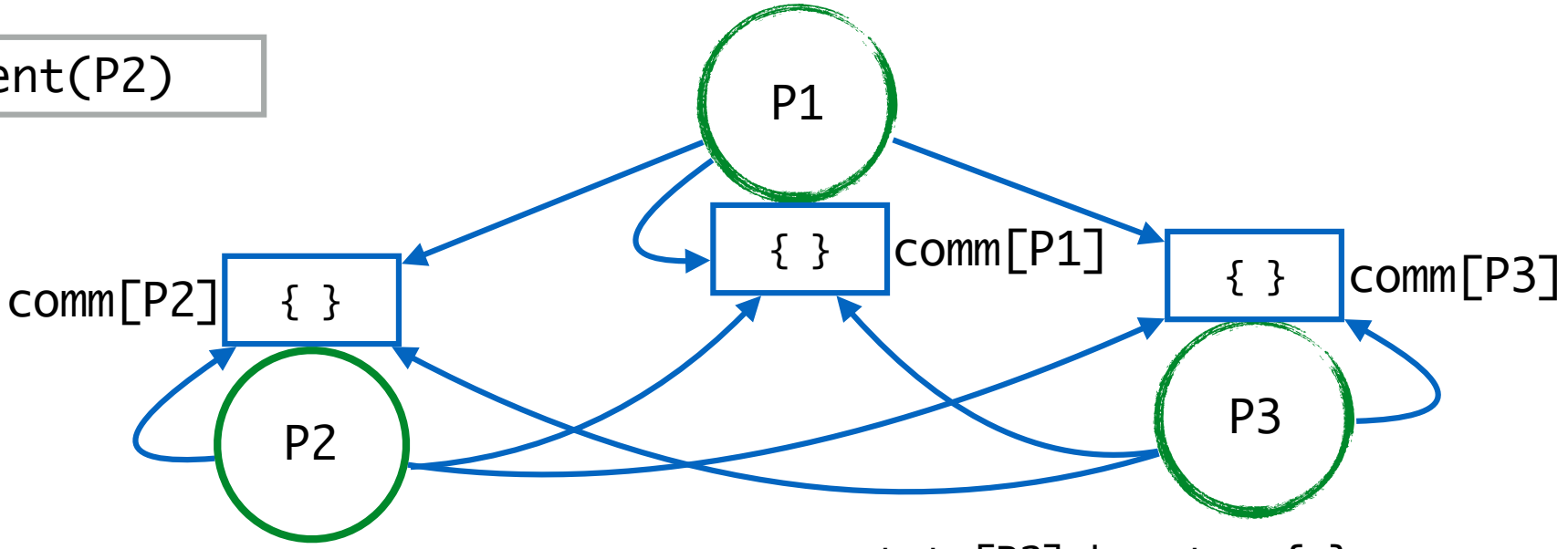
Best effort broadcast (2nd try)

```
---------------------------- MODULE BestEffortBroadcast ----------------------------
TypeInv ==
    /\ comm \in [Process -> SUBSET Message]   /\   broadcasted \in SUBSET Message
    /\ delivered \in SUBSET MessageDel        /\   pstate \in [Process -> LocalState]
    /\ alive \in SUBSET Process.              /\   correct \in SUBSET Process
------------------------------------------------------------------------------------
send(p) ==
    /\ p \in alive
    /\ \E m \in pstate[p].bcast, q \in Process :
        /\ m \notin pstate[p].msent[q] (* m was not sent to q *)
        /\ comm' = [comm EXCEPT ![q] = comm[q] \union {m}]
        /\ pstate' = [pstate EXCEPT ![p] =
                            [bcast |-> pstate[p].bcast,
                             msent |-> [pstate[p].msent
                                        EXCEPT ![q] = pstate[p].msent[q] \union {m}],
                             mcount |-> pstate[p].mcount]]
        /\ UNCHANGED<<broadcasted,delivered,alive,correct>>
```

broadcasted	{ m1 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P1].mcount = 1

sent(P2)



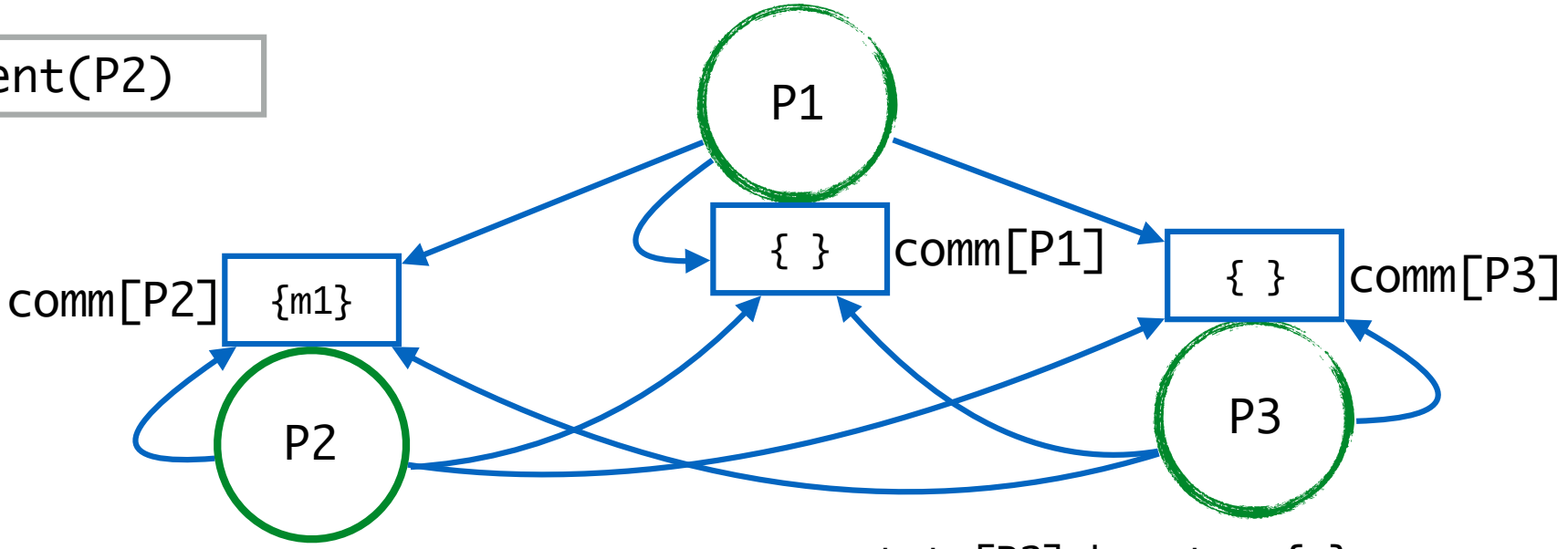
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

broadcasted	{ m1 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{m1},P3↦{}}
pstate[P1].mcount = 1

sent(P2)



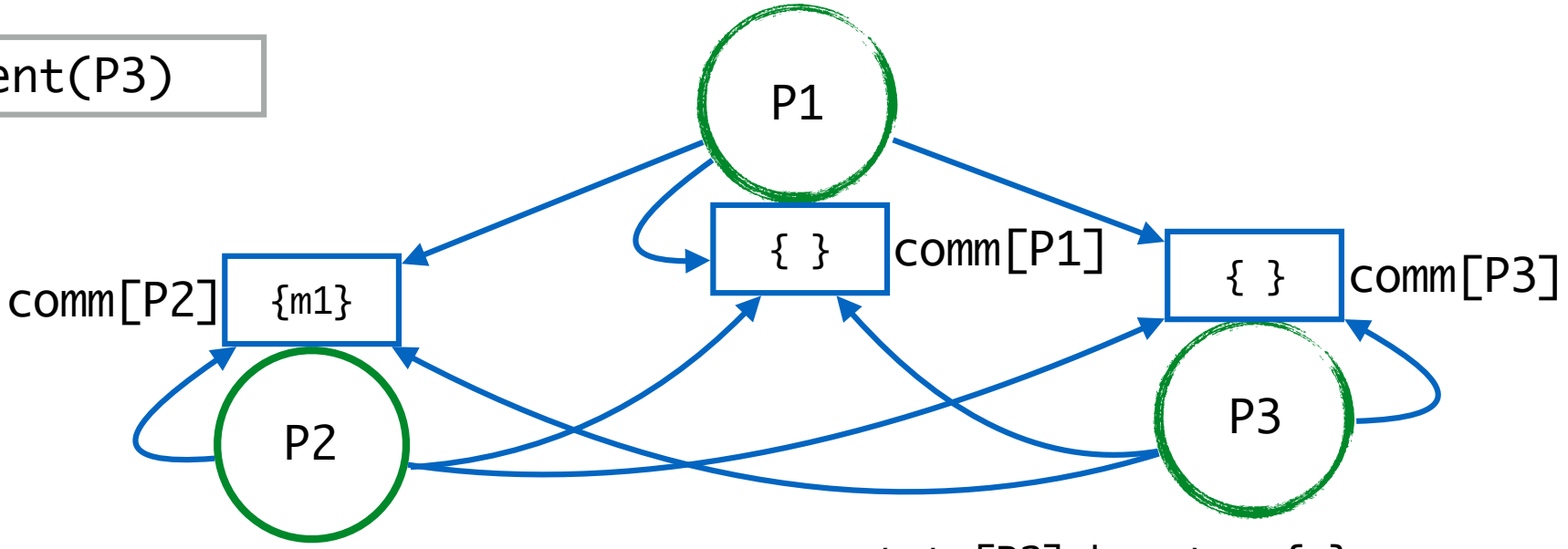
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

broadcasted	{ m1 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{m1},P3↦{}}
pstate[P1].mcount = 1

sent(P3)



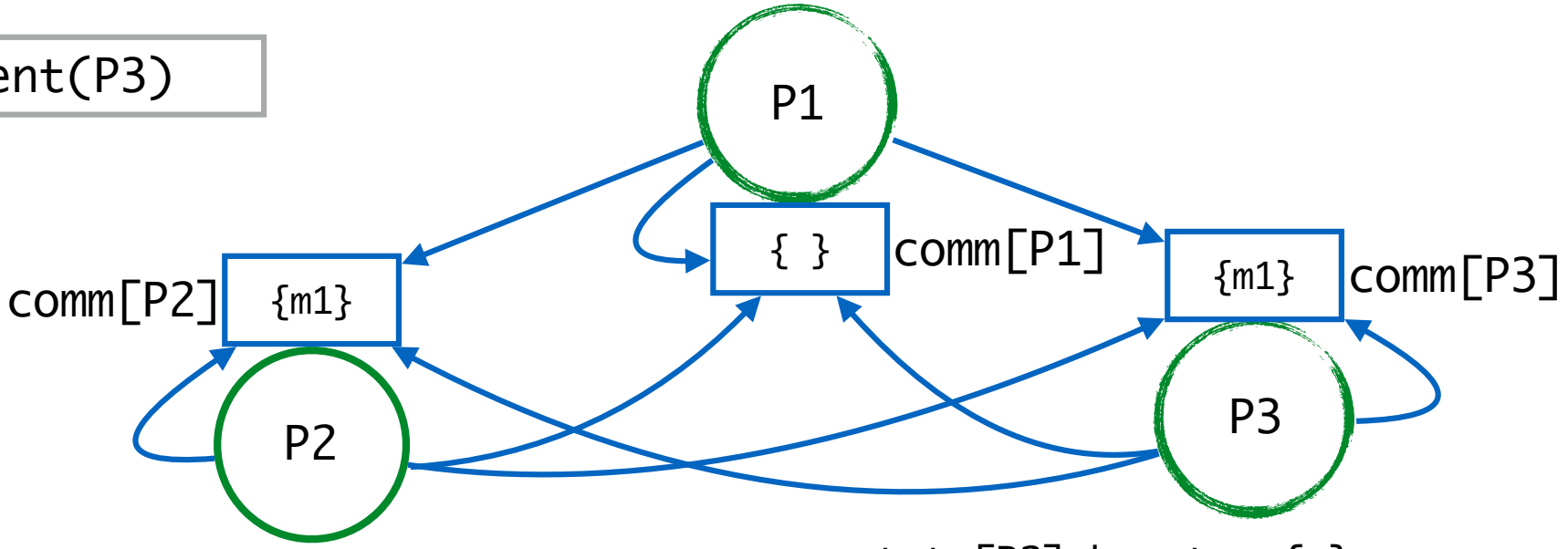
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

broadcasted	{ m1 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

```
pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{m1},P3↦{m3}}
pstate[P1].mcount = 1
```

sent(P3)



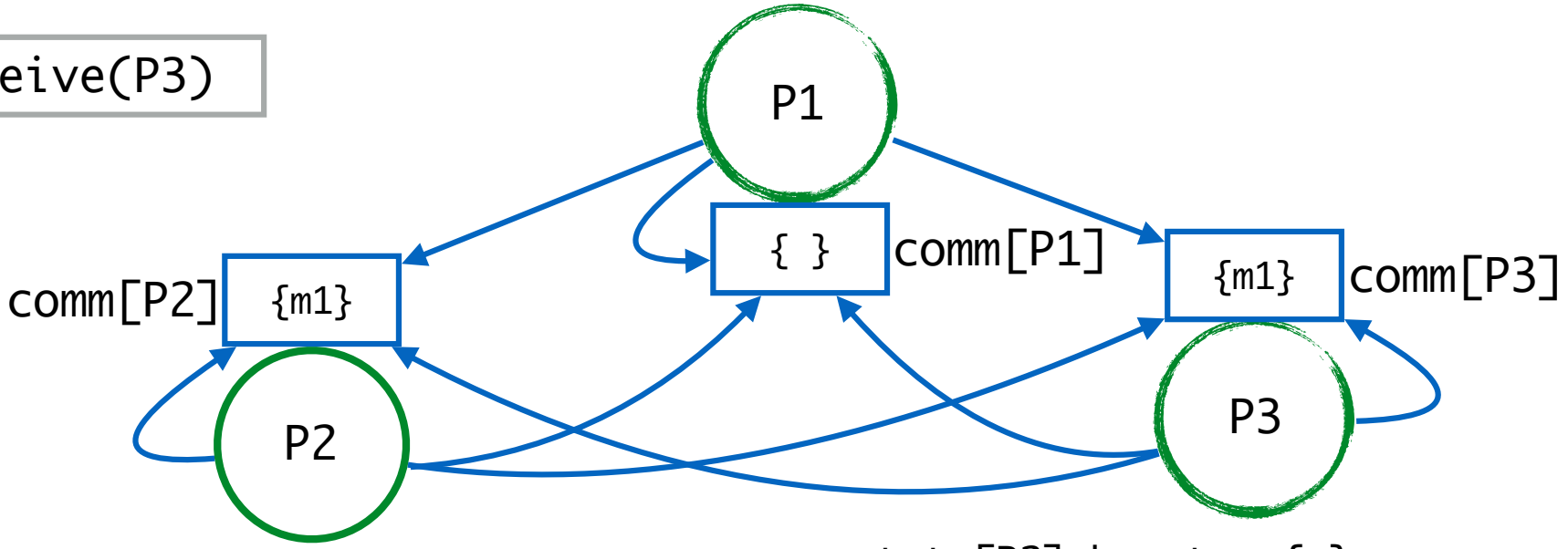

```
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0
```

```
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0
```

broadcasted	{ m1 }
delivered	{ }
alive	{ P1, P2, P3 }
correct	{ P2 }

```
pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{m1},P3↦{m3}}
pstate[P1].mcount = 1
```

receive(P3)



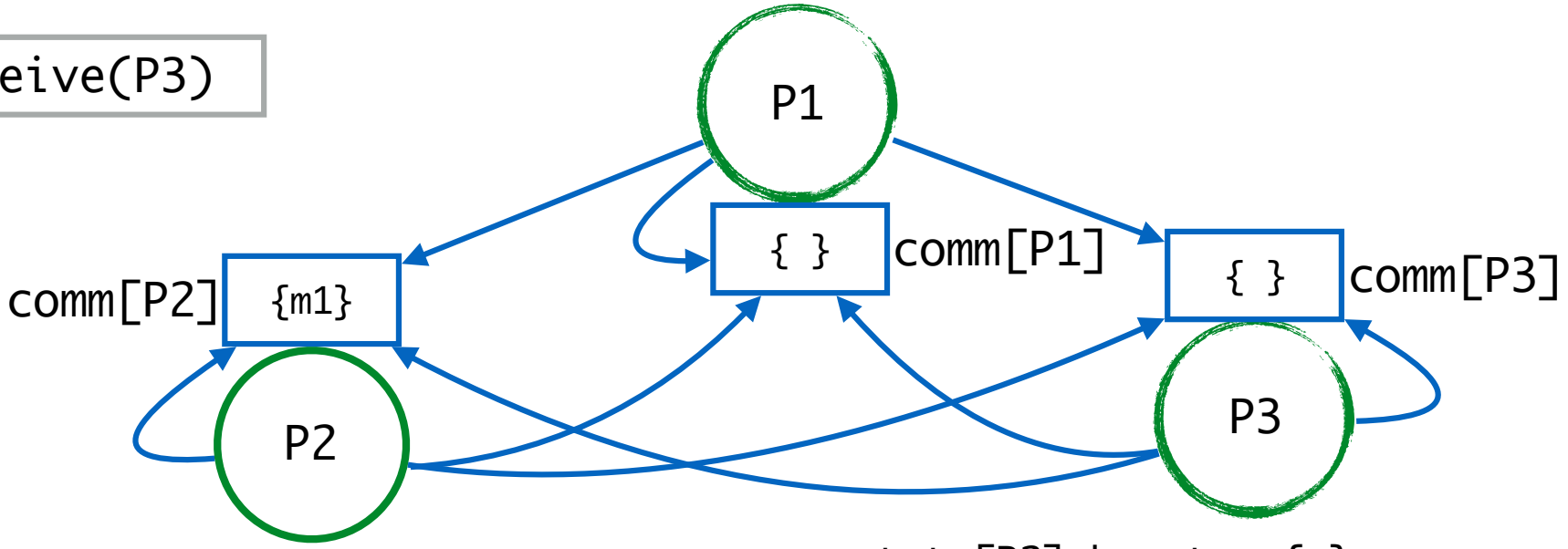

```
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0
```

```
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0
```

broadcasted	{ m1 }
delivered	{ P3_m1 }
alive	{ P1, P2, P3 }
correct	{ P2 }

pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{m1},P3↦{m3}}
pstate[P1].mcount = 1

receive(P3)



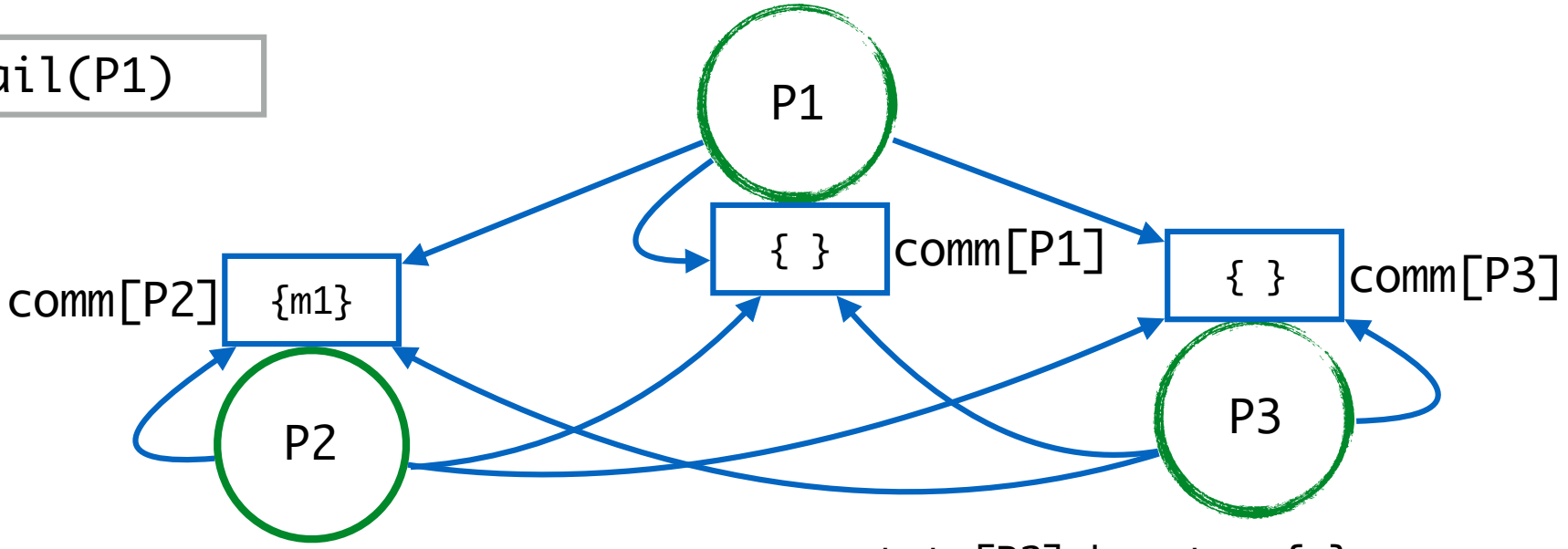
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

broadcasted	{ m1 }
delivered	{ P3_m1 }
alive	{ P1, P2, P3 }
correct	{ P2 }

pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{m1},P3↦{m3}}
pstate[P1].mcount = 1

fail(P1)



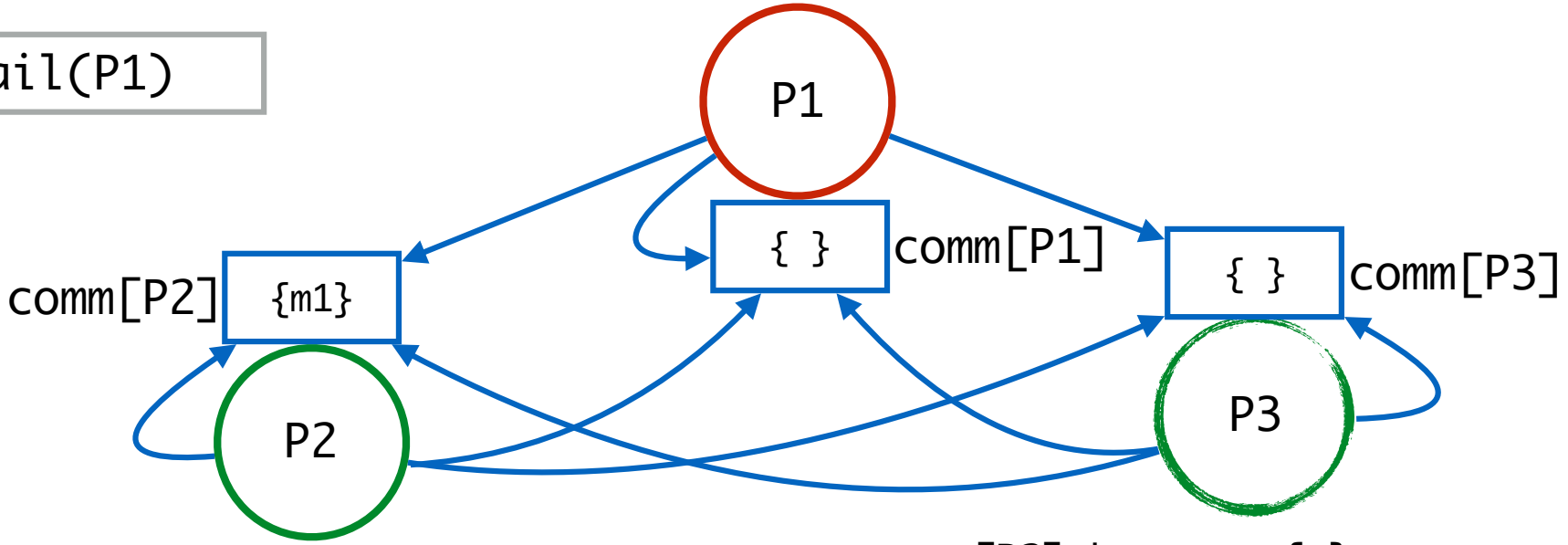
pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

broadcasted	{ m1 }
delivered	{ P3_m1 }
alive	{ P2, P3 }
correct	{ P2 }

pstate[P1].bcast = { m1 }
pstate[P1].msent = {P1↦{},P2↦{m1},P3↦{m3}}
pstate[P1].mcount = 1

fail(P1)



pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

pstate[P2].bcast = { }
pstate[P2].msent = {P1↦{},P2↦{},P3↦{}}
pstate[P2].mcount = 0

Best effort broadcast (2nd try)

```
---------------------------- MODULE BestEffortBroadcast ----------------------------

Next == \E p \in Process : broadcast(p) \/ receive(p) \/ fail(p)

------------------------------------------------------------------------------

Spec == /\ Init
        /\ [][Next]_<<comm, broadcasted, delivered, pstate, alive, correct>>
==============================================================================
```

Just checking type safety!
What about specific protocol properties?

System's properties

- Safety
  - Something bad never happens
  - No two processes can access the critical section at the same time
- Liveness
  - Something good eventually happens
  - If a client wants to access a resource, it will eventually get access to that resource

TLA

- Temporal Logic of Actions:
  - Action formulas: describe state and state transitions
  - Temporal formulas: describe state sequences (traces)

Action Formulas

- `[A]<<e>> == A \/ e' = e`
- `<A><<e>> == A /\ ~(e' = e)`

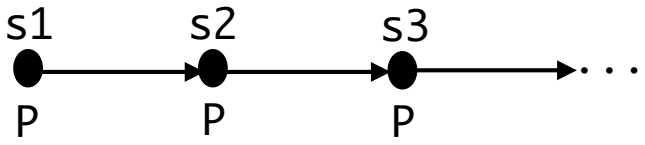
```
---------------------------- MODULE OneBitClock ----------------------------
VARIABLE b

Init == (b=0) \/ (b=1)

TypeInv == b \in {0,1}

Next == IF b = 0 THEN b' = 1
        ELSE b' = 0

Spec == Init /\ [][Next]_<<b>>
----------------------------------------------------------------------------
```

Temporal Formulas

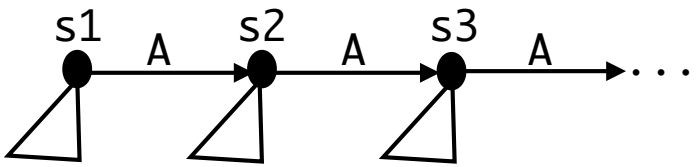
- Temporal operators
  - `[] F` — F is always true
  - `<> F` — F is eventually true
  - `F ~> G` — F leads to G
  - `WF_<<e>> (A)` — Weak fairness for action A
  - `SF_<<e>> (A)` — Strong fairness for action A

Temporal Formulas

- []F **F is always true**
  - Formula []P, where P is a state predicate, is true iff P is true in every state



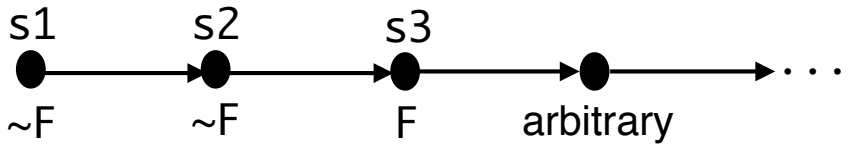
  - Formula [][A]_<<e>>, where A is an action and e a state function, is true iff every successive step is an [A]_<<e>> step



Temporal Formulas

- <>F **F is eventually true**
  - <>F == ~[](~F)

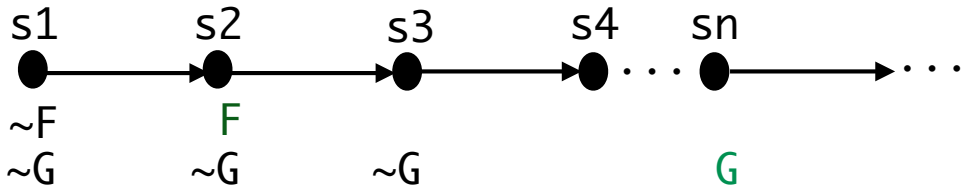
    F is not always false



Temporal Formulas

- `F ~> G` **F leads to G**
  - `F ~> G == []( F => <>G )`

    Whenever `F` is true, then `G` is eventually true



- Every request leads to a response

  `request ~> response`

Temporal Formulas

- []<>F **Infinitely often** (Progress)
  - E.g. the traffic light is green infinitely often.
- <>[]F **Eventually always** (Stability)
  - E.g. eventually all messages are delivered.

Liveness

- To prove liveness properties it is necessary to make some assumptions about the system environment.
- TLA has two forms of fairness:
  - Strong fairness
  - Weak fairness

Temporal Formulas

- Temporal operators
  - `[] F` F is always true
  - `<> F` F is eventually true
  - `F ~> G` F leads to G
  - `WF_<<e>> (A)` Weak fairness for action A
  - `SF_<<e>> (A)` Strong fairness for action A

Weak fairness

- `WF_<<e>> (A)` **Weak fairness for action A**
  - `(<>[] ENABLED <A>_<<e>>) => ([]<> <A>_<<e>>)`

    If **A** ever becomes forever enabled, then an **A** step must eventually occur

Strong fairness

- `SF_<<e>> (A)` **Strong fairness for action A**
  - `([]<> ENABLED <A>_<<e>>) => ([]<> <A>_<<e>>)`

    If **A** is infinitely often enabled, then infinitely many **A** steps occur

Fairness in TLA

- **Weak fairness** of **A** asserts that an **A** step must eventually occur if **A** is continuously enabled
  - continuously -> without interruption
- **Strong fairness** of **A** asserts that an **A** step must eventually occur if **A** is continually enabled
  - continually -> repeatedly, possible with interruptions

Liveness properties

- Validity
  - For any two correct processes i and j, every message broadcast by i is eventually delivered by j.

- Agreement
  - If a message m is delivered by some correct process i, then m is eventually delivered by every correct process j.

Express this properties in TLA.