Groupy: a group membership service

SODX

**gms1**

-module(gms1).

-export([start/1, start/2]).

start(Name) ->

Self = self(),

spawn\_link(fun()-> init(Name, Self) end).

init(Name, Master) ->

leader(Name, Master, []).

start(Name, Grp) ->

Self = self(),

spawn\_link(fun()-> init(Name, Grp, Self) end).

init(Name, Grp, Master) ->

Self = self(),

Grp ! {join, Self},

receive

{view, Leader, Slaves} ->

Master ! joined,

slave(Name, Master, Leader, Slaves)

end.

leader(Name, Master, Slaves) ->

receive

{mcast, Msg} ->

bcast(Name, {msg, Msg}, Slaves),

Master ! {deliver, Msg},

leader(Name, Master, Slaves);

{join, Peer} ->

NewSlaves = lists:append(Slaves, [Peer]),

bcast(Name, {view, self(), NewSlaves}, NewSlaves),

leader(Name, Master, NewSlaves);

stop ->

ok;

Error ->

io:format("leader ~s: strange message ~w~n", [Name, Error])

end.

bcast(\_, Msg, Nodes) ->

lists:foreach(fun(Node) -> Node ! Msg end, Nodes).

slave(Name, Master, Leader, Slaves) ->

receive

{mcast, Msg} ->

Leader ! {mcast, Msg},

slave(Name, Master, Leader, Slaves);

{join, Peer} ->

Leader ! {join, Peer},

slave(Name, Master, Leader, Slaves);

{msg, Msg} ->

Master ! {deliver, Msg},

slave(Name, Master, Leader, Slaves);

{view, Leader, NewSlaves} ->

slave(Name, Master, Leader, NewSlaves); %% TODO: COMPLETE

stop ->

ok;

Error ->

io:format("slave ~s: strange message ~w~n", [Name, Error])

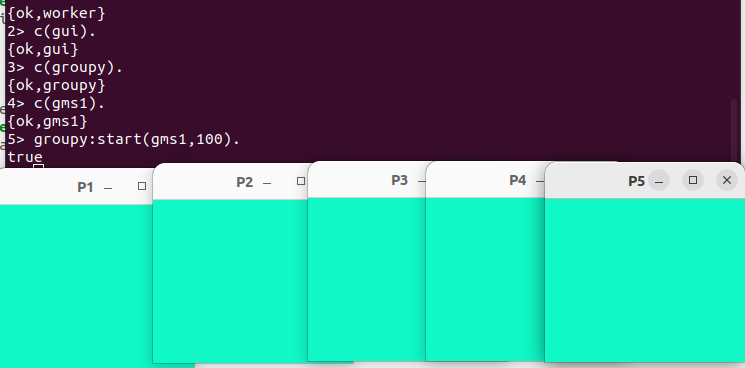
end.

Implementa un servei bàsic de filiació grupal.

* Gestiona missatges multicast, sol·licituds de nous membres i actualitzacions de vistes per als esclaus
* Encaminament de missatges entre el líder i el procés.
* Multidifusió bàsica de missatges a tots els nodes.

No maneja fallades de nodes, només l'addició de nous membres.

**Experiments:**



**gms1 - groupy2**

groupy2:

-module(groupy2).

-export([start/2, stop/0, stop/1]).

start(Module, Sleep) ->

P = worker:start("P1", Module, Sleep),

register(a, P),

Node1 = 'node2@sergi-Lenovo',

Node2 = 'node3@sergi-Lenovo',

Node3 = 'node4@sergi-Lenovo',

Node4 = 'node5@sergi-Lenovo',

P2 = rpc:call(Node1, worker, start, ["P2", Module, P, Sleep]),

P3 = rpc:call(Node2, worker, start, ["P3", Module, P, Sleep]),

P4 = rpc:call(Node3, worker, start, ["P4", Module, P, Sleep]),

P5 = rpc:call(Node4, worker, start, ["P5", Module, P, Sleep]),

Workers = [{b,P2},{c,P3},{d,P4},{e,P5}],

{ok, Workers}.

stop() ->

stop(a),

stop(b),

stop(c),

stop(d),

stop(e).

stop(Name) ->

case whereis(Name) of

undefined ->

ok;

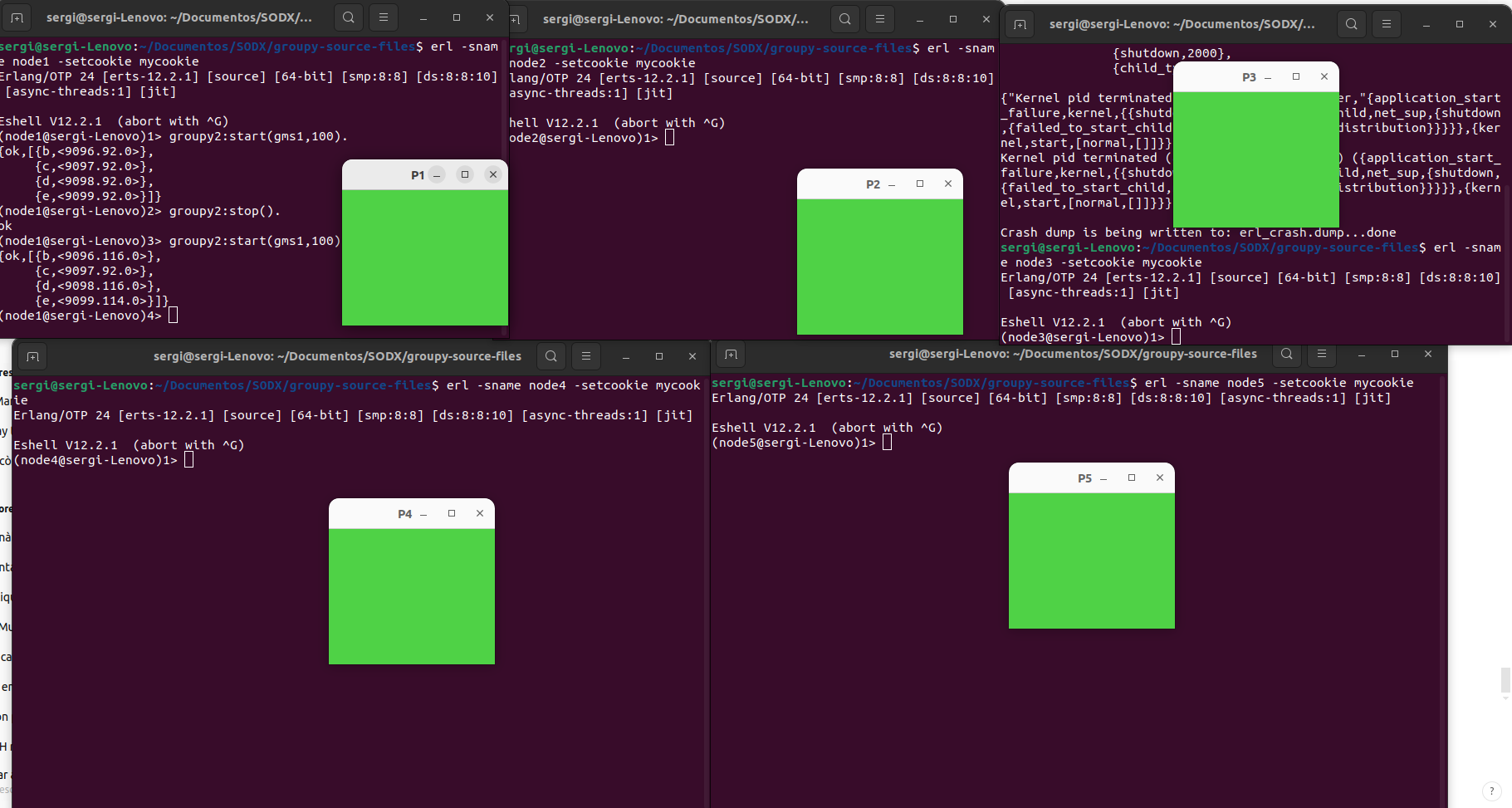
Pid ->

Pid ! stop

end.



**Experiments:**



**gms2**



-module(gms2).

-export([start/1, start/2]).

start(Name) ->

Self = self(),

spawn\_link(fun() -> init(Name, Self) end).

init(Name, Master) ->

leader(Name, Master, []).

start(Name, Grp) ->

Self = self(),

spawn\_link(fun() -> init(Name, Grp, Self) end).

init(Name, Grp, Master) ->

Self = self(),

Grp ! {join, Self},

receive

{view, Leader, Slaves} ->

Ref = erlang:monitor(process, Leader), % Monitor the leader

Master ! joined,

slave(Name, Master, Leader, Slaves, Ref)

end.

leader(Name, Master, Slaves) ->

receive

{mcast, Msg} ->

bcast(Name, {msg, Msg}, Slaves),

Master ! {deliver, Msg},

leader(Name, Master, Slaves);

{join, Peer} ->

NewSlaves = lists:append(Slaves, [Peer]),

bcast(Name, {view, self(), NewSlaves}, NewSlaves),

leader(Name, Master, NewSlaves);

stop ->

ok;

Error ->

io:format("leader ~s: strange message ~w~n", [Name, Error]),

leader(Name, Master, Slaves)

end.

slave(Name, Master, Leader, Slaves, Ref) ->

receive

{mcast, Msg} ->

Leader ! {mcast, Msg},

slave(Name, Master, Leader, Slaves, Ref);

{join, Peer} ->

Leader ! {join, Peer},

slave(Name, Master, Leader, Slaves, Ref);

{msg, Msg} ->

Master ! {deliver, Msg},

slave(Name, Master, Leader, Slaves, Ref);

{view, NewLeader, NewSlaves} ->

erlang:demonitor(Ref, [flush]), % Stop monitoring the old leader

NewRef = erlang:monitor(process, NewLeader), % Monitor the new leader

slave(Name, Master, NewLeader, NewSlaves, NewRef);

{'DOWN', \_Ref, process, Leader, \_Reason} ->

io:format("~s: leader ~p is down. Starting election.~n", [Name, Leader]),

election(Name, Master, Slaves);

stop ->

ok;

Error ->

io:format("~s: strange message ~w~n", [Name, Error]),

slave(Name, Master, Leader, Slaves, Ref)

end.

election(Name, Master, Slaves) ->

Self = self(),

case Slaves of

[Self | Rest] ->

io:format(" ~s: new leader.~n", [Name]),

leader(Name, Master, Rest);

[NewLeader | Rest] ->

Ref = erlang:monitor(process, NewLeader),

io:format(" ~s: following new leader ~p.~n", [Name, NewLeader]),

slave(Name, Master, NewLeader, Rest, Ref);

[] ->

io:format(" ~s:Exiting.~n", [Name]),

ok

end.

bcast(\_, Msg, Nodes) ->

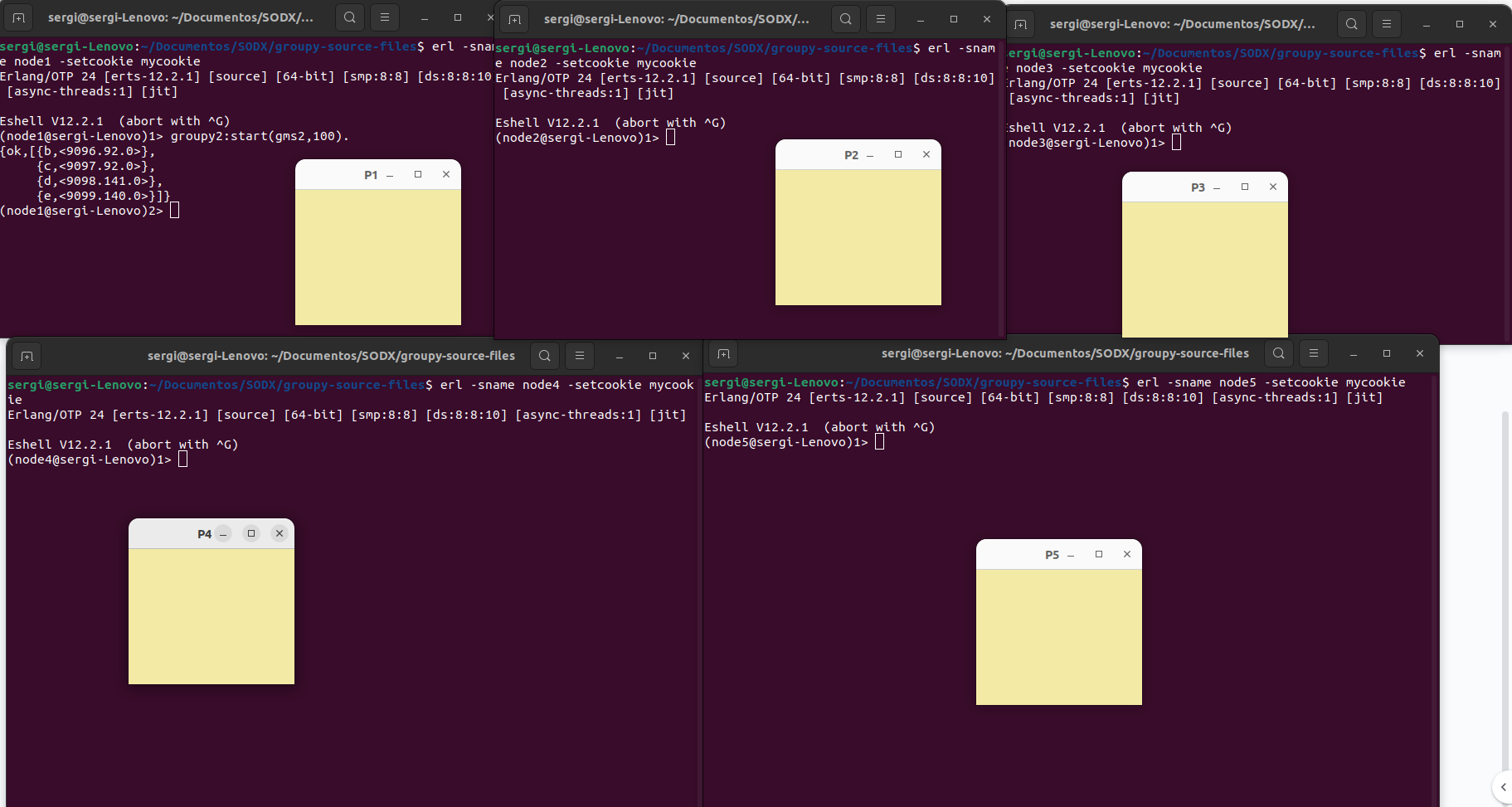
lists:foreach(fun(Node) -> Node ! Msg end, Nodes).

Estén gms1 per a manejar fallades de líders.

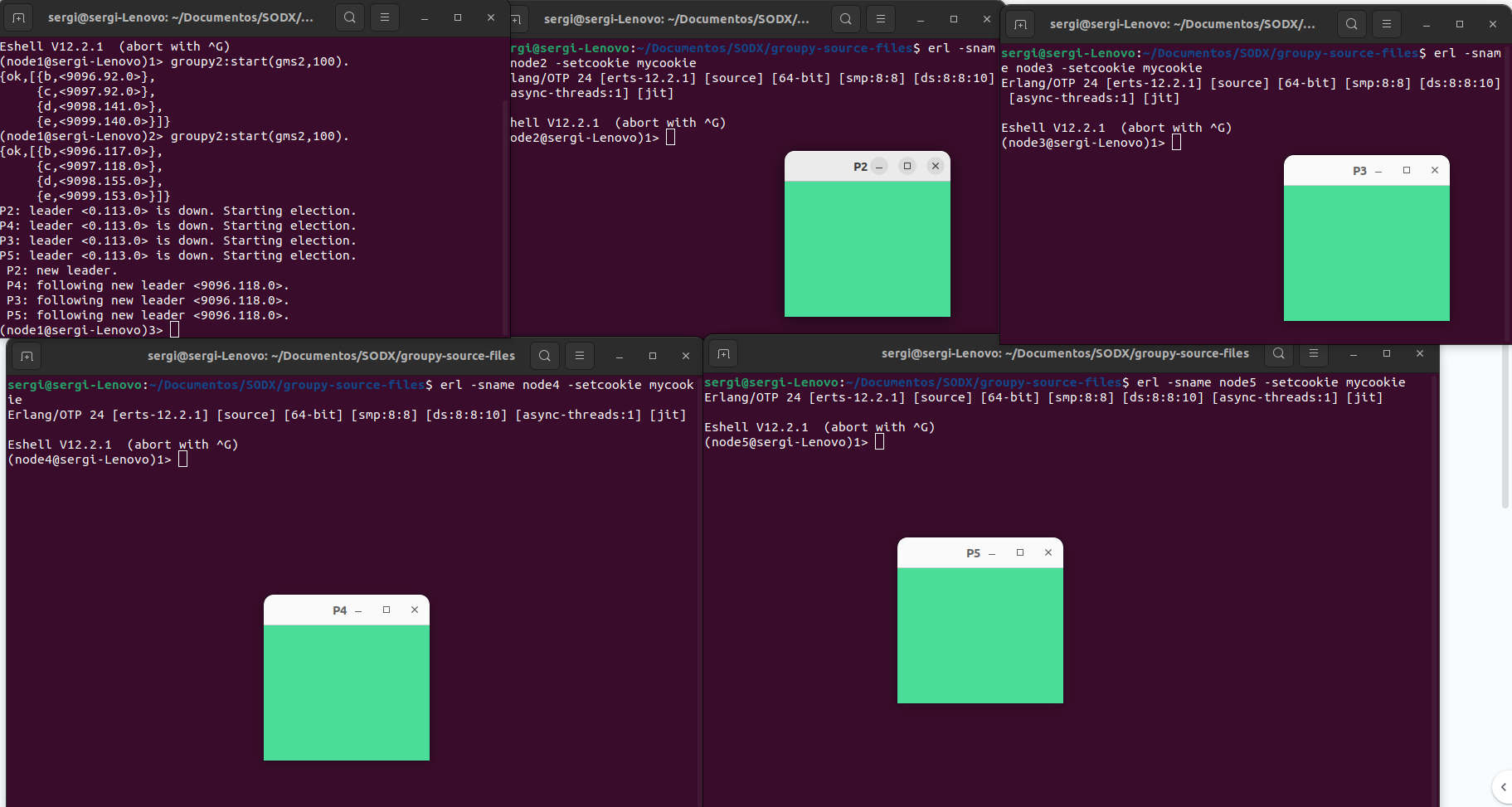
* Detecció de fallades: S'usa erlang:monitor/2 per a detectar si el líder falla.
* Elecció de líders: Un worker assumeix el rol de líder si detecta que el líder ha fallat.
* Procediments d'elecció per a assegurar la continuïtat del servei.
* S'afegeix un mecanisme de timeout en la inicialització d'esclaus per a evitar bloquejos si el líder no respon.

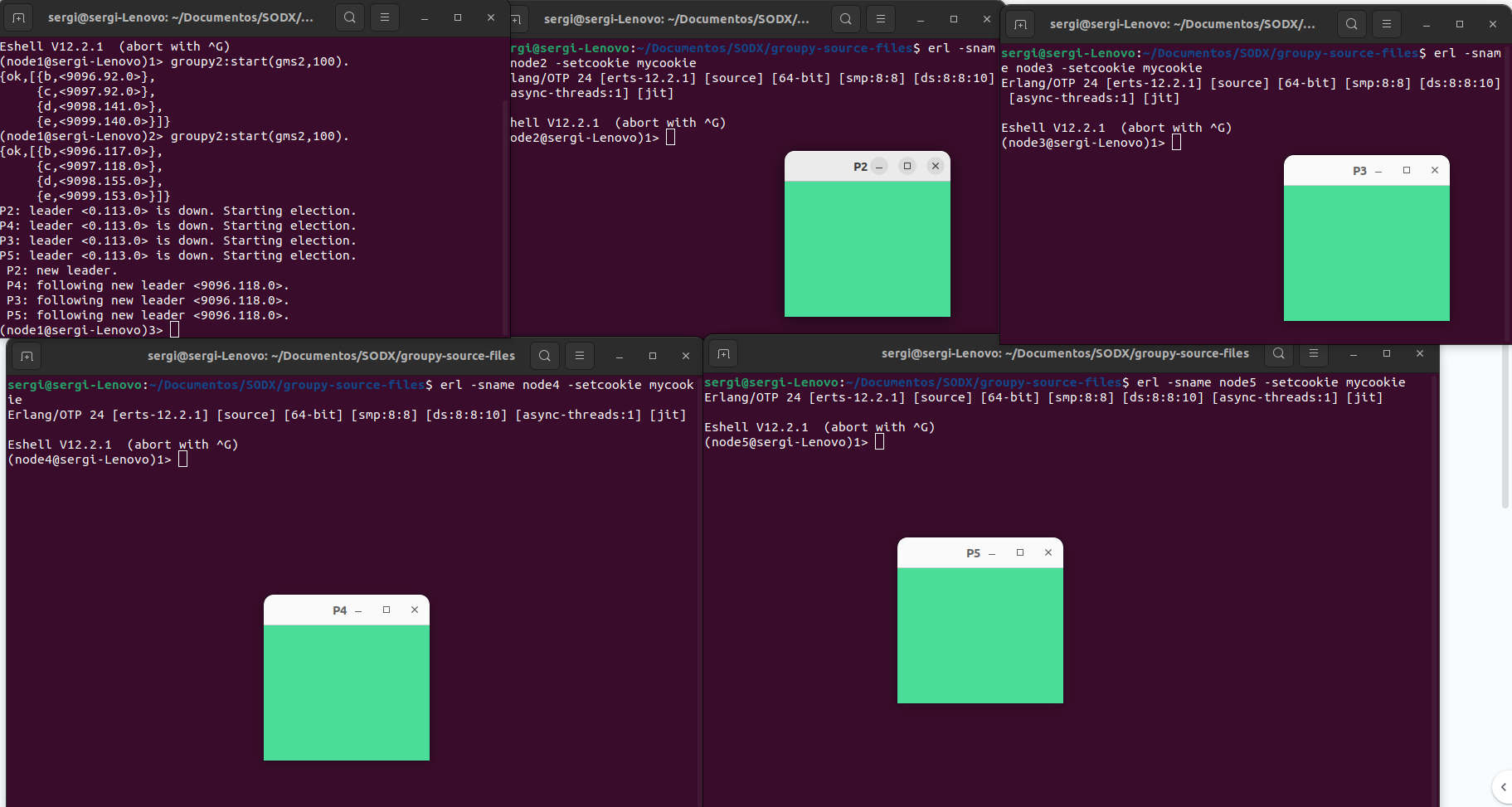
**Experiments:**

Executem el codi gms2 i veiem com tenim un per a cada màquina i tos perfectament sincronitzats amb un líder.

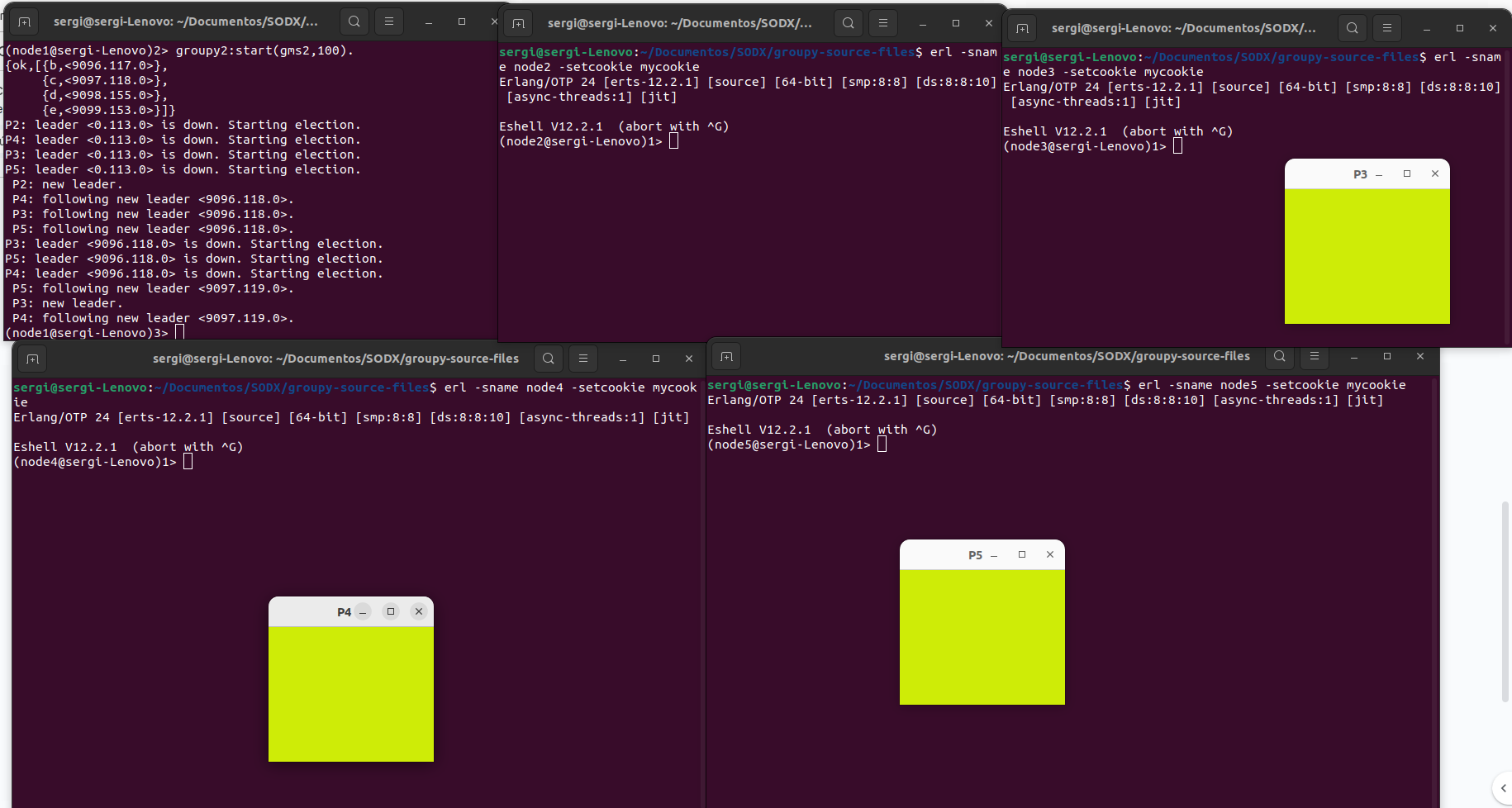


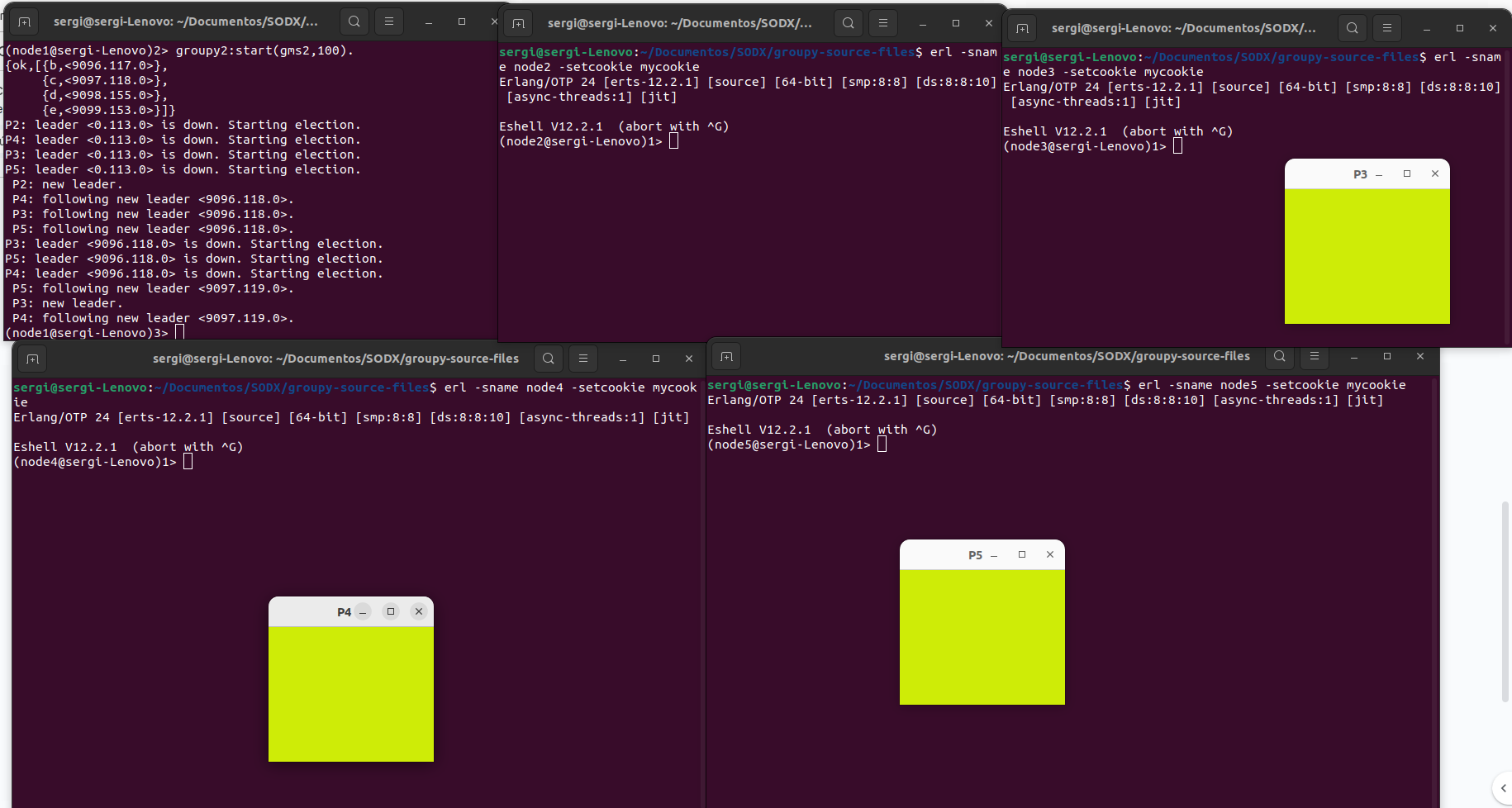
Si fem que caigui el lider podem veure com s’ha seleccionat un nou líder i segueixen tots perfectament sincronitzats





Ho podem tornar a comprovar tornant a fer que caigui el nou lider, ho podem repetir fins a que quedi només un i veurem com fan una nova elecció de lider, el qual tots segueixen i en tot moment sincronitzats





**gms2crash**

-module(gms2).

-export([start/1, start/2]).

-define(arghh, 100).

start(Name) ->

Self = self(),

spawn\_link(fun() -> init(Name, Self) end).

init(Name, Master) ->

{A1, A2, A3} = now(),

random:seed(A1, A2, A3),

leader(Name, Master, []).

start(Name, Grp) ->

Self = self(),

spawn\_link(fun() -> init(Name, Grp, Self) end).

init(Name, Grp, Master) ->

{A1, A2, A3} = now(),

random:seed(A1, A2, A3),

Self = self(),

Grp ! {join, Self},

receive

{view, Leader, Slaves} ->

Ref = erlang:monitor(process, Leader),

Master ! joined,

slave(Name, Master, Leader, Slaves, Ref)

end.

leader(Name, Master, Slaves) ->

receive

{mcast, Msg} ->

bcast(Name, {msg, Msg}, Slaves),

Master ! {deliver, Msg},

leader(Name, Master, Slaves);

{join, Peer} ->

NewSlaves = lists:append(Slaves, [Peer]),

bcast(Name, {view, self(), NewSlaves}, NewSlaves),

leader(Name, Master, NewSlaves);

stop ->

ok;

Error ->

io:format("leader ~s: strange message ~w~n", [Name, Error]),

leader(Name, Master, Slaves)

end.

slave(Name, Master, Leader, Slaves, Ref) ->

receive

{mcast, Msg} ->

Leader ! {mcast, Msg},

slave(Name, Master, Leader, Slaves, Ref);

{join, Peer} ->

Leader ! {join, Peer},

slave(Name, Master, Leader, Slaves, Ref);

{msg, Msg} ->

Master ! {deliver, Msg},

slave(Name, Master, Leader, Slaves, Ref);

{view, NewLeader, NewSlaves} ->

erlang:demonitor(Ref, [flush]),

NewRef = erlang:monitor(process, NewLeader),

slave(Name, Master, NewLeader, NewSlaves, NewRef);

{'DOWN', \_Ref, process, Leader, \_Reason} ->

io:format("~s: leader ~p is down. Starting election.~n", [Name, Leader]),

election(Name, Master, Slaves);

stop ->

ok;

Error ->

io:format("~s: strange message ~w~n", [Name, Error]),

slave(Name, Master, Leader, Slaves, Ref)

end.

election(Name, Master, Slaves) ->

Self = self(),

case Slaves of

[Self | Rest] ->

io:format(" ~s: new leader.~n", [Name]),

leader(Name, Master, Rest);

[NewLeader | Rest] ->

Ref = erlang:monitor(process, NewLeader),

io:format(" ~s: following new leader ~p.~n", [Name, NewLeader]),

slave(Name, Master, NewLeader, Rest, Ref);

[] ->

io:format(" ~s:Exiting.~n", [Name]),

ok

end.

bcast(Name, Msg, Nodes) ->

lists:foreach(fun(Node) ->

Node ! Msg,

crash(Name, Msg)

end, Nodes).

crash(Name, Msg) ->

case random:uniform(?arghh) of

?arghh ->

io:format("leader ~s CRASHED: msg ~w~n", [Name, Msg]),

exit(no\_luck);

\_ ->

ok

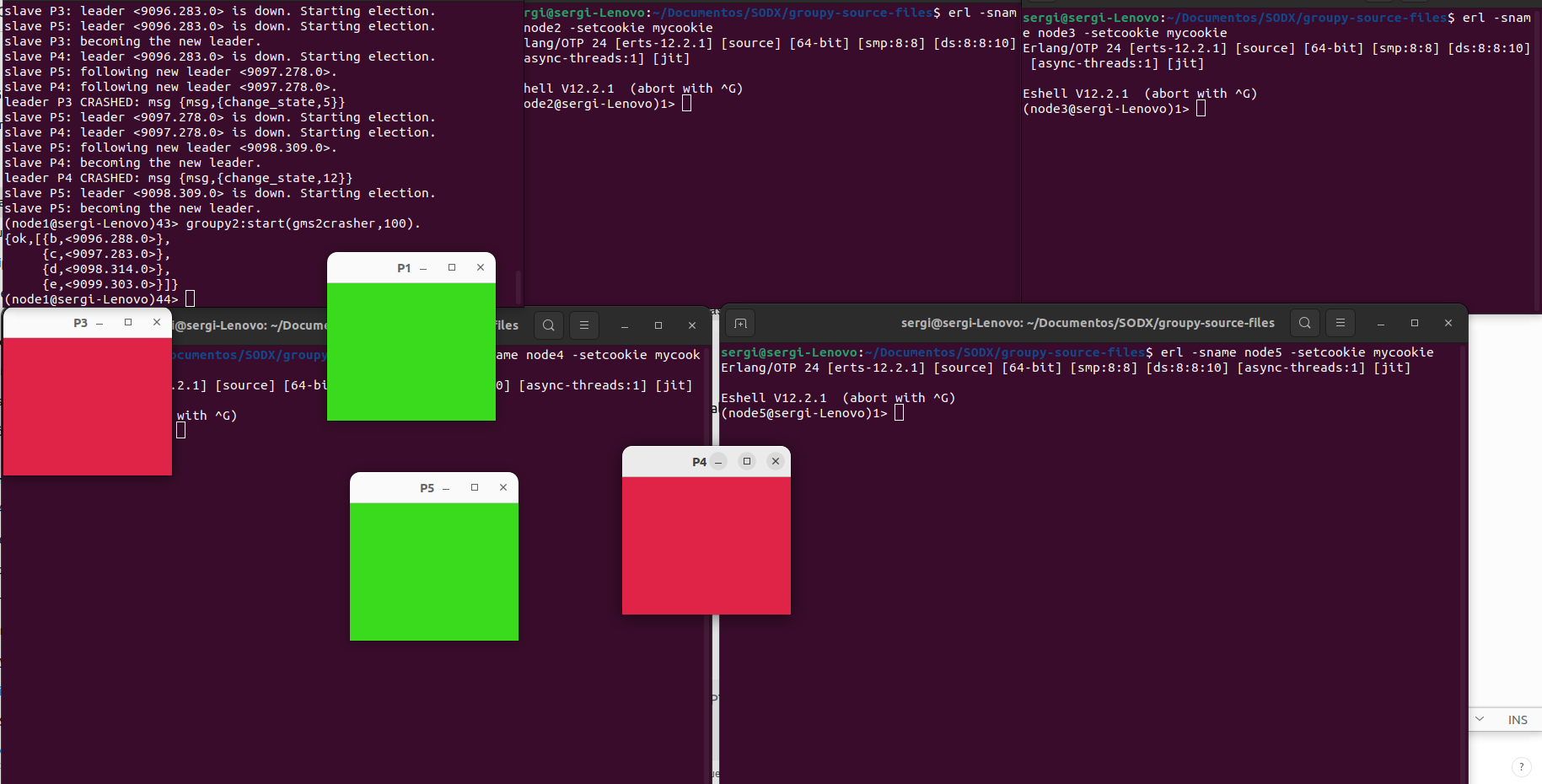
end.

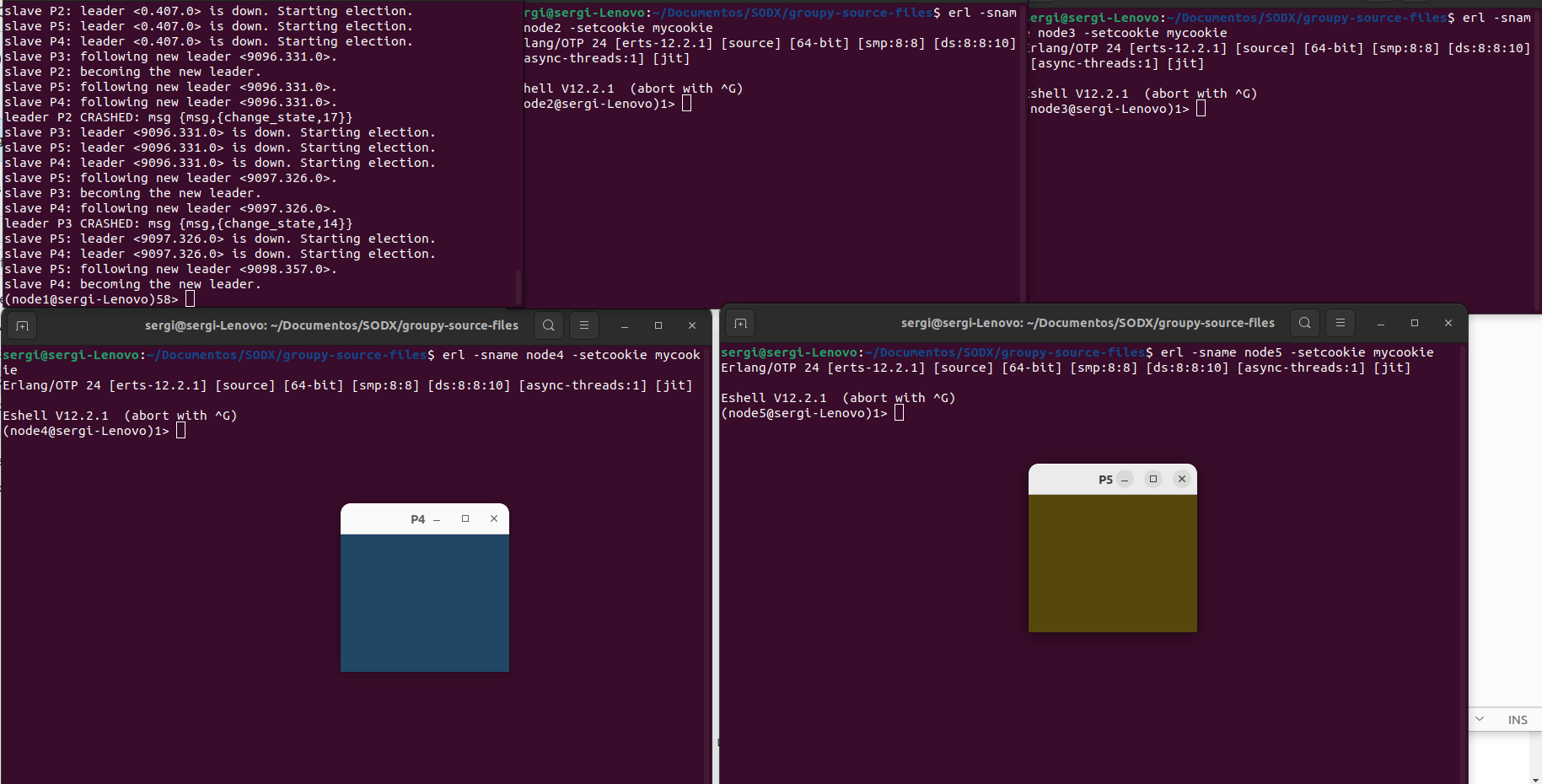
Introdueix fallades simulades per a avaluar com els nodes manegen líders que fallen durant la multidifusió. Ús de random per a forçar fallades aleatòries.

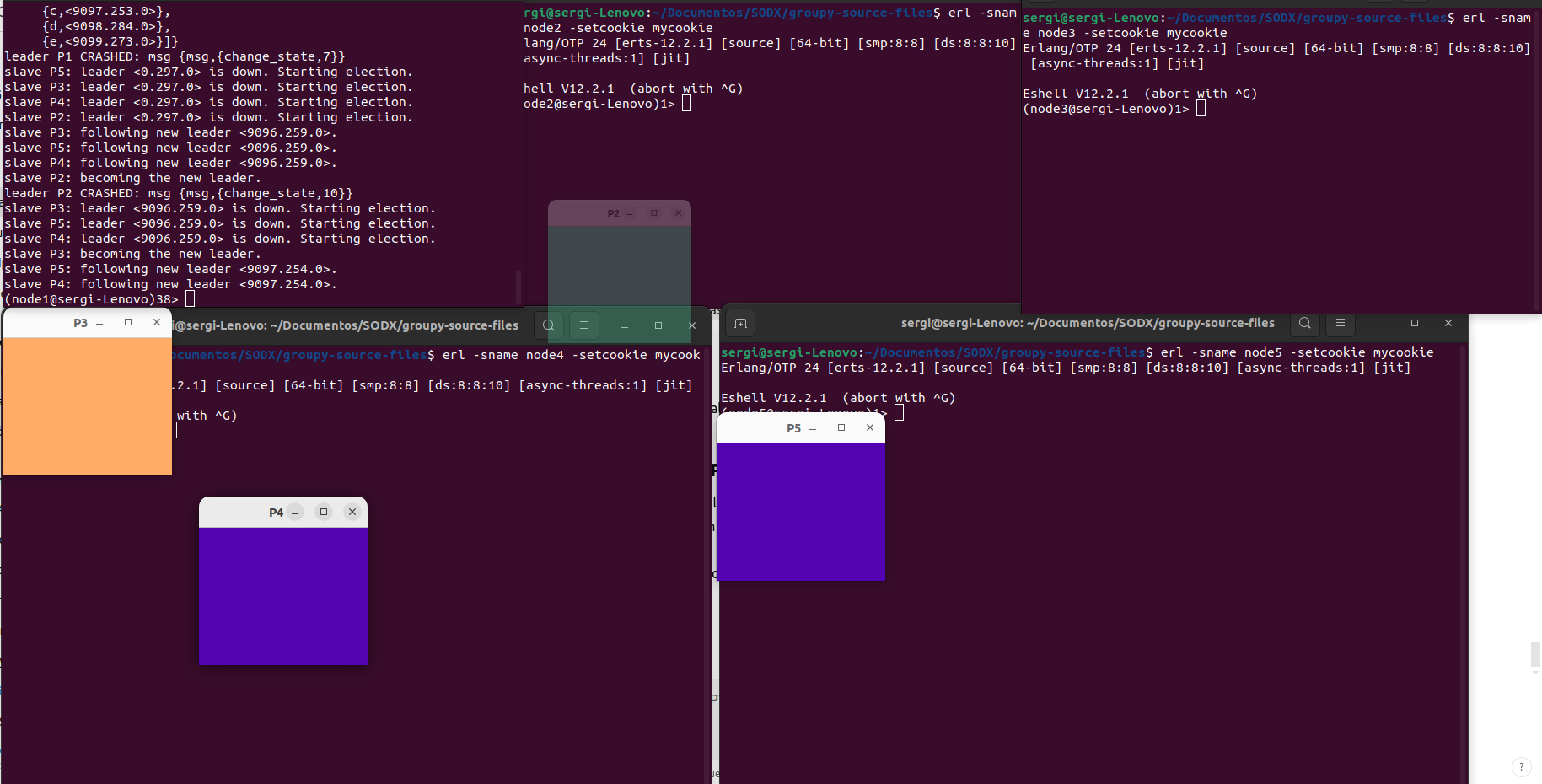
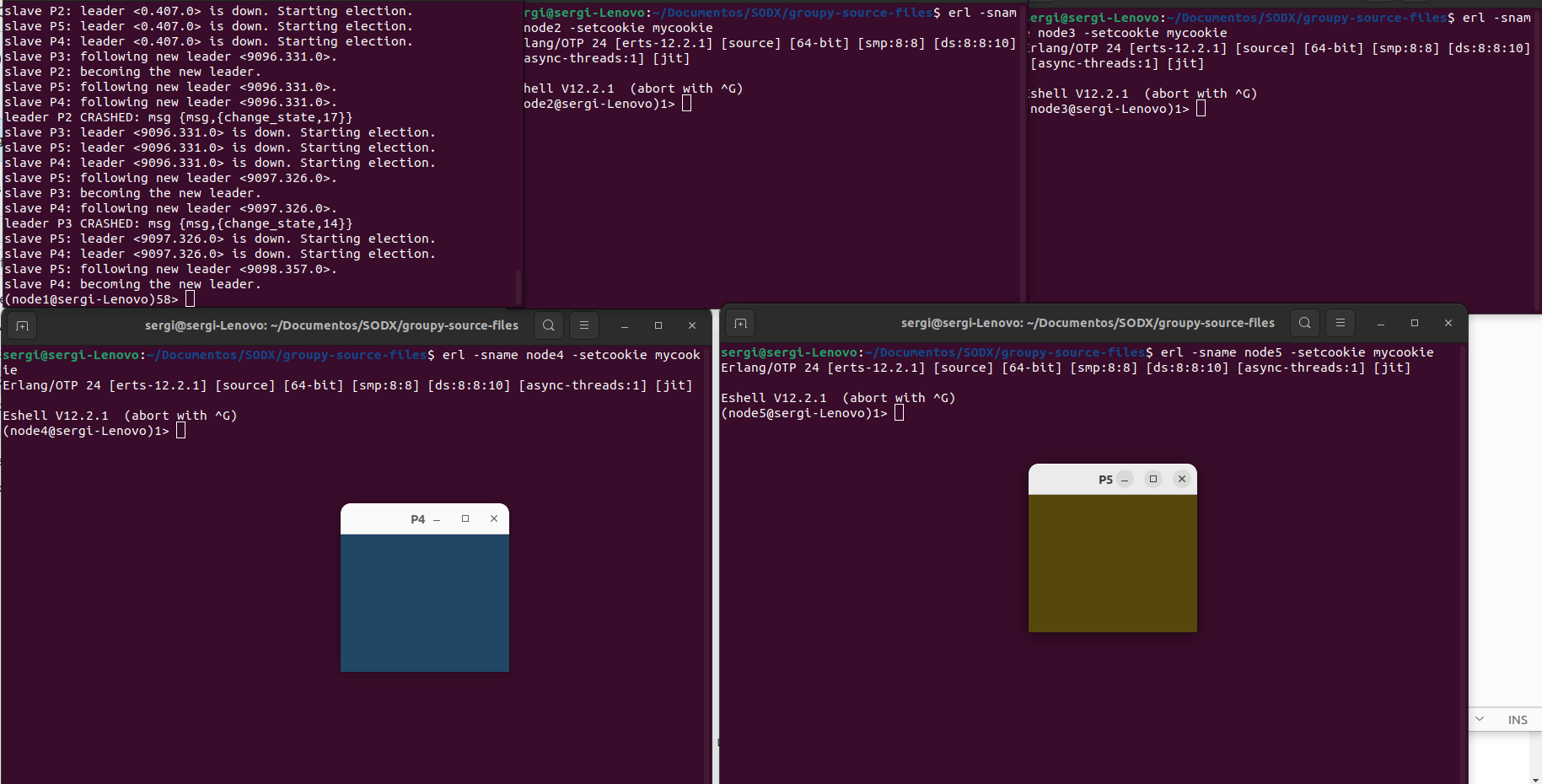
La caiguda de líders pot desincronizar als nodes.

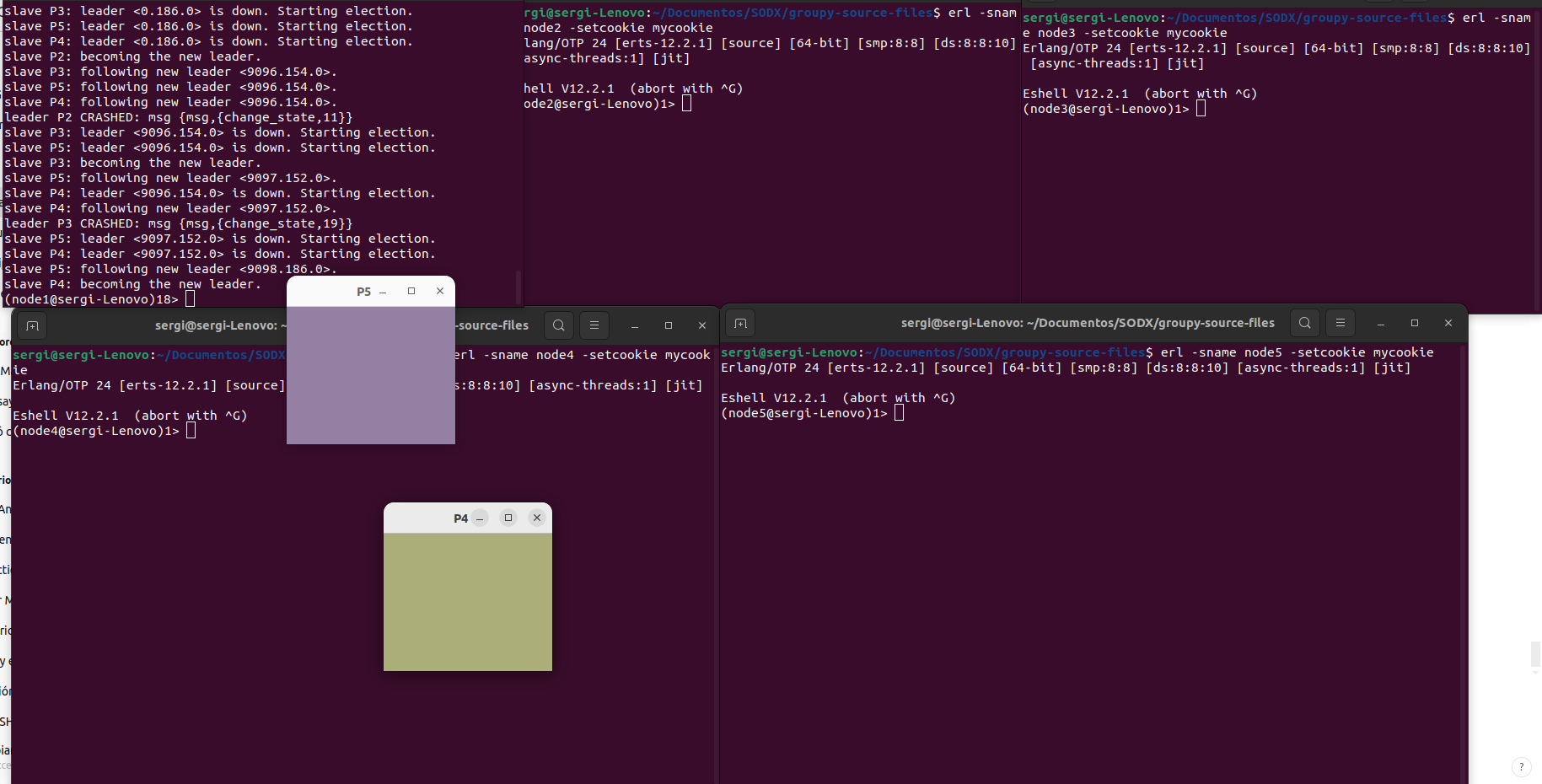
**Experiments:**

Podem veure en els experiments com funciona igual que abans, seleccionant un nou líder si el líder actual crashes, pero, a diferencia de l’anterior podem apreciar com es desincronitzen quan el líder cau, i a mesura que van caient els líders, es desincronitza mes.









**Open Questions**.

Why is this happening?

Els missatges no es lliuren de manera de confiança a causa de fallades del líder enmig de la multidifusió. Això causa que alguns nodes rebin missatges mentre que uns altres no.

**gms3**

-module(gms3).

-export([start/1, start/2]).

start(Name) ->

Self = self(),

spawn\_link(fun() -> init(Name, Self) end).

init(Name, Master) ->

{A1, A2, A3} = now(),

random:seed(A1, A2, A3),

leader(Name, Master, [], 0).

start(Name, Grp) ->

Self = self(),

spawn\_link(fun() -> init(Name, Grp, Self) end).

init(Name, Grp, Master) ->

{A1, A2, A3} = now(),

random:seed(A1, A2, A3),

Self = self(),

Grp ! {join, Self},

receive

{view, Leader, Slaves} ->

Ref = erlang:monitor(process, Leader),

Master ! joined,

slave(Name, Master, Leader, Slaves, Ref, 0, undefined)

after 1000 ->

Master ! {error, "no reply from leader"}

end.

leader(Name, Master, Slaves, N) ->

receive

{mcast, Msg} ->

NextN = N + 1,

bcast(Name, {msg, NextN, Msg}, Slaves),

Master ! {deliver, Msg},

leader(Name, Master, Slaves, NextN);

{join, Peer} ->

NewSlaves = lists:append(Slaves, [Peer]),

bcast(Name, {view, self(), NewSlaves}, NewSlaves),

leader(Name, Master, NewSlaves, N);

stop ->

ok;

Error ->

io:format("leader ~s: strange message ~w~n", [Name, Error])

end.

slave(Name, Master, Leader, Slaves, Ref, N, Last) ->

receive

{mcast, Msg} ->

Leader ! {mcast, Msg},

slave(Name, Master, Leader, Slaves, Ref, N, Last);

{msg, Seq, Msg} when Seq < N ->

slave(Name, Master, Leader, Slaves, Ref, N, Last);

{msg, Seq, Msg} ->

Master ! {deliver, Msg},

slave(Name, Master, Leader, Slaves, Ref, Seq + 1, {msg, Seq, Msg});

{view, NewLeader, NewSlaves} ->

erlang:demonitor(Ref, [flush]),

NewRef = erlang:monitor(process, NewLeader),

slave(Name, Master, NewLeader, NewSlaves, NewRef, N, Last);

{'DOWN', \_Ref, process, Leader, \_Reason} ->

election(Name, Master, Slaves, N, Last);

stop ->

ok;

Error ->

io:format("slave ~s: strange message ~w~n", [Name, Error]),

slave(Name, Master, Leader, Slaves, Ref, N, Last)

end.

election(Name, Master, Slaves, N, Last) ->

Self = self(),

case Slaves of

[Self | Rest] ->

case Last of

undefined -> ok;

\_ -> bcast(Name, Last, Rest)

end,

leader(Name, Master, Rest, N);

[NewLeader | Rest] ->

Ref = erlang:monitor(process, NewLeader),

slave(Name, Master, NewLeader, Rest, Ref, N, Last);

[] ->

io:format("slave ~s: no leader available~n", [Name]),

slave(Name, Master, undefined, [], undefined, N, Last)

end.

bcast(\_, Msg, Nodes) ->

lists:foreach(fun(Node) ->

Node ! Msg,

crash(Node, Msg)

end, Nodes).

crash(Name, Msg) ->

case random:uniform(100) of

100 ->

io:format("leader ~p CRASHED: msg ~p~n", [Name, Msg]),

exit(no\_luck);

\_ -> ok

end.

Els missatges es numeren per a evitar duplicats. i els esclaus mantenen còpies de l'últim missatge rebut per a sincronització després de fallades.

L'elecció inclou retransmissió de l'últim missatge del líder mort.

**Open Questions.**

i) How would we have to change the implementation to handle the possibly

lost messages?

Introduir un protocol de multidifusió de confiança on els nodes mantinguin un historial dels missatges rebuts i retransmetin qualsevol missatge perdut durant l'elecció d'un nou líder.

ii) How would this impact performance?

Retransmetre missatges perduts augmentaria la latència i el trànsit en la xarxa, però garantiria tots els nodes sincronitzats.

iii) What would happen if we wrongly suspect the leader to have crashed?

S'iniciaria una elecció innecessària, la qual cosa podria provocar una interrupció temporal en el servei. A més, els nodes podrien generar duplicats o desincronització.