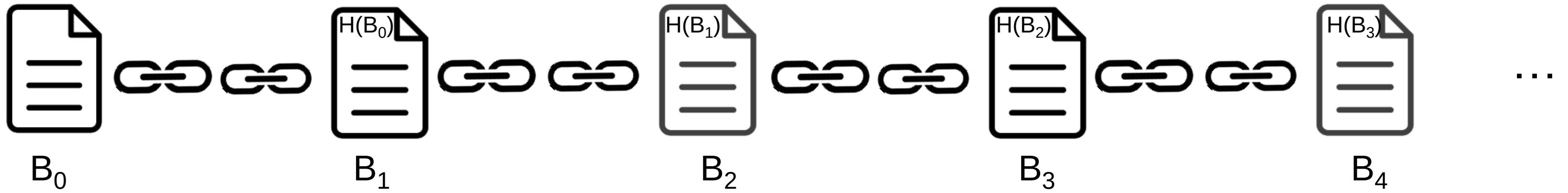
Correctness of Tendermint-core Blockchains

Y. Amoussou-Guenou[^,*], A. Del Pozzo[^], M. Potop-Butucaru[*],
S.Tucci-Piergiovanni[^]

[^] Institut LIST, CEA, Université Paris-Saclay
[*] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6

BLOCKCHAIN

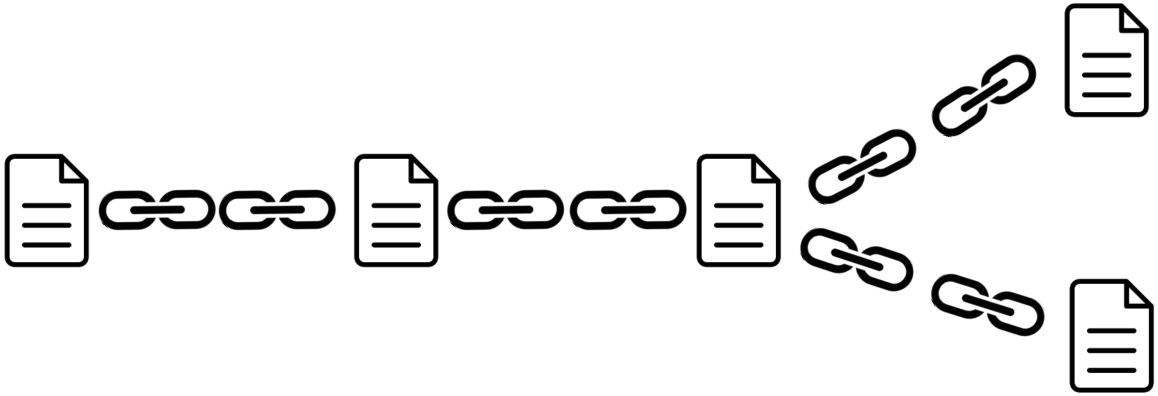
- Potentially unbounded set of processes that communicate in a network through message passing
- **Distributed ledger**, ledger replicated by each processes
- **Tamper-resistant**, by cryptographic mechanism
- Build in an **append only** manner



- A sequence of blocks, each block containing transactions
- Each block contains the hash of the prior block in the chain

FORKS

- **When adding a block in the Blockchain, others processes**
  - Should be aware of it
  - Should add the block in their local copy of the blockchain

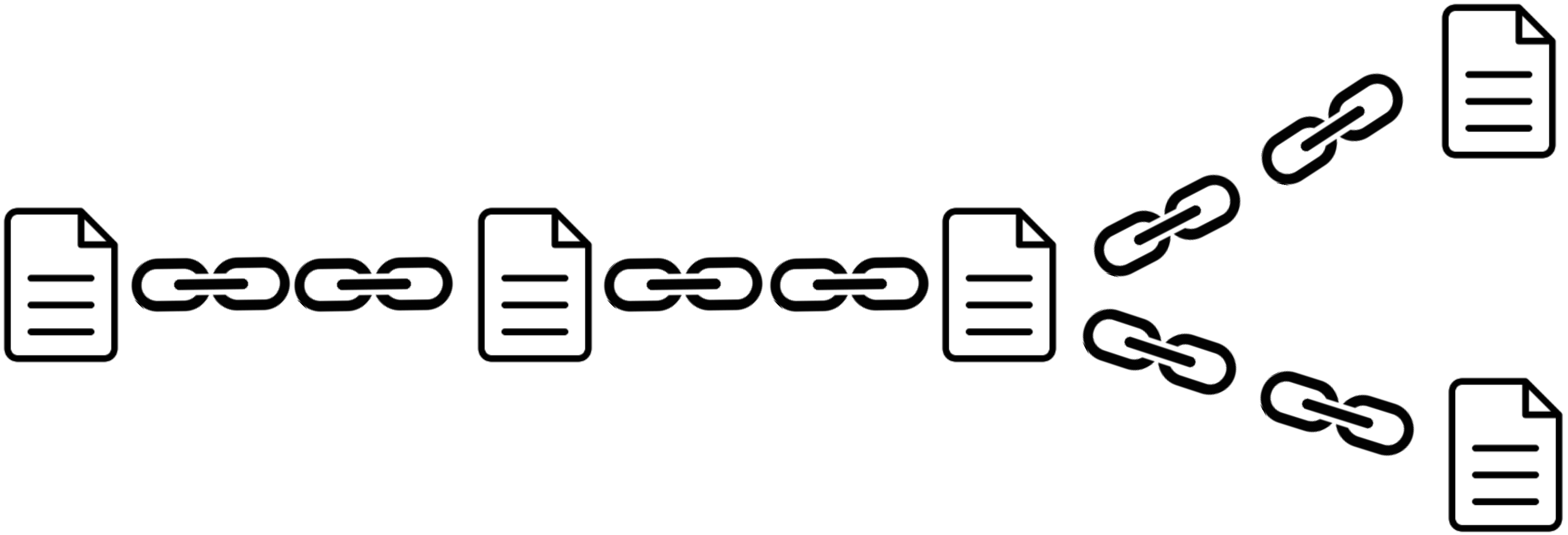


- The presence of such a structure can be harmful to the system, and the goal is to avoid it

OUTLINE
CONTRIBUTIONS

- The use of Consensus to build a Blockchain, e.g. Tendermint
- Formalization of Tendermint
- Conditions under which the protocol works
- Proofs of correctness of Tendermint

AVOIDING THE FORKS



(ONE-SHOT) CONSENSUS

- A process is **correct** if it follows the given protocol

- **Termination**

  Every correct process eventually decides some value

- **Integrity**

  No correct process decides twice

- **Agreement**

  If there is a correct process that decides a value B, then eventually all the correct processes decide B

- **Validity**

  A decided value is valid, it satisfies the predefined predicate

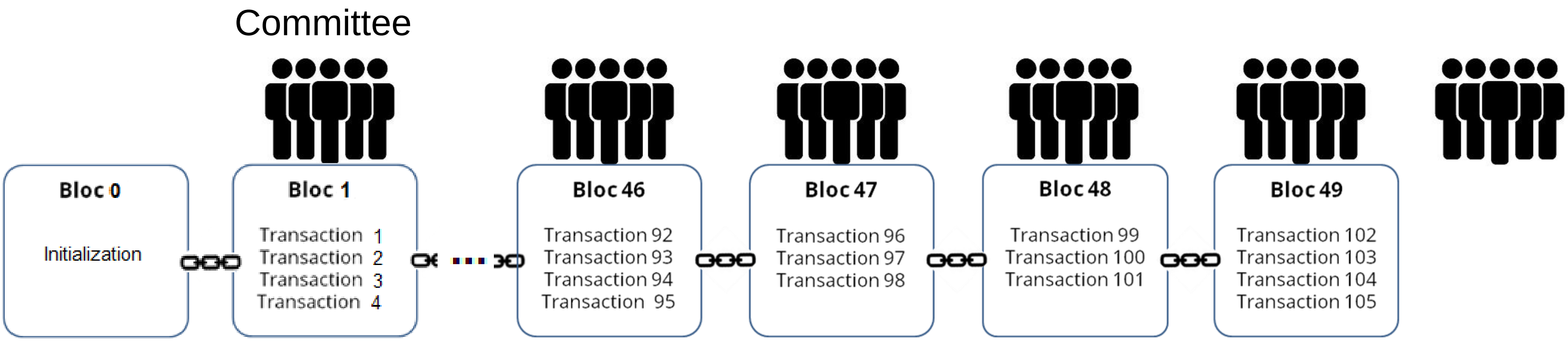
T. Crain, V. Gramoli, M. Larrea, and M. Raynal, ‘*(Leader/Randomization/Signature)-free Byzantine Consensus for Consortium Blockchains’*, 2017.

WHAT IS TENDERMINT ?

- Tendermint is a blockchain used in different applications
- Tendermint is the **first** proposed blockchain to claim solving the Consensus, but **has never been formalized**

J. Kwon, ‘Tendermint: Consensus without Mining’, 2014.

HOW DOES IT WORK ?



- The blockchain network is composed of an unknown number **n** of processes
- To append a new block, a **committee** of processes of fixed size **N** is **deterministically** selected, and known by every process
- That committee runs a one-shot consensus protocol to decide on the next block
- The decision of the committee is sent to all processes, and is the next block to be appended
- The next committee rewards the previous one

REPEATED CONSENSUS

- Every process produces a sequence of value/decision. We call that sequence the ***output*** of the process

- **Properties :**
  - Termination

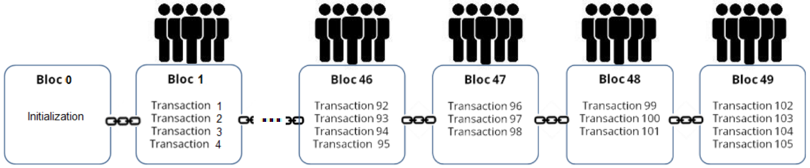
    *Every **correct** process has an infinite output*

  - Agreement

    For all k, the k^{th} value of any two correct processes is the same

  - Validity

    *Each value in the output of any correct process is **valid**, it satisfies a predefined predicate*



T. Crain, V. Gramoli, M. Larrea, and M. Raynal, '*(Leader/Randomization/Signature)-free Byzantine Consensus for Consortium Blockchains'*, 2017.

C. Delporte-Gallet, S. Devismes, H. Fauconnier, F. Petit, and S. Toueg, *'With Finite Memory Consensus Is Easier Than Reliable Broadcast'*, in *Principles of Distributed Systems*, Berlin, Heidelberg, 2008.

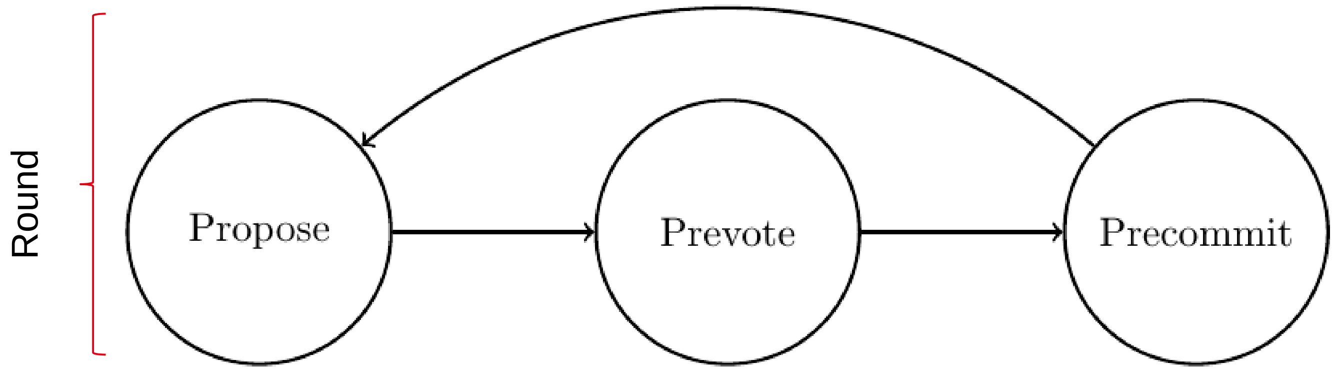
SYSTEM MODEL

- The total number of processes by committee is ***N = 3f+1***
  f is the maximum number of Byzantine process
- The communication **is eventually synchronous**
- Messages are **signed** and signatures cannot be forged
- Broadcast
  - Gossip
  - Best effort broadcast
- **Finite** arrival model

R. Baldoni, M. Bertier, M. Raynal, and S. Tucci-Piergiovanni, ‘Looking for a Definition of Dynamic Distributed Systems’, in *Parallel Computing Technologies*, 2007, pp. 1–14.

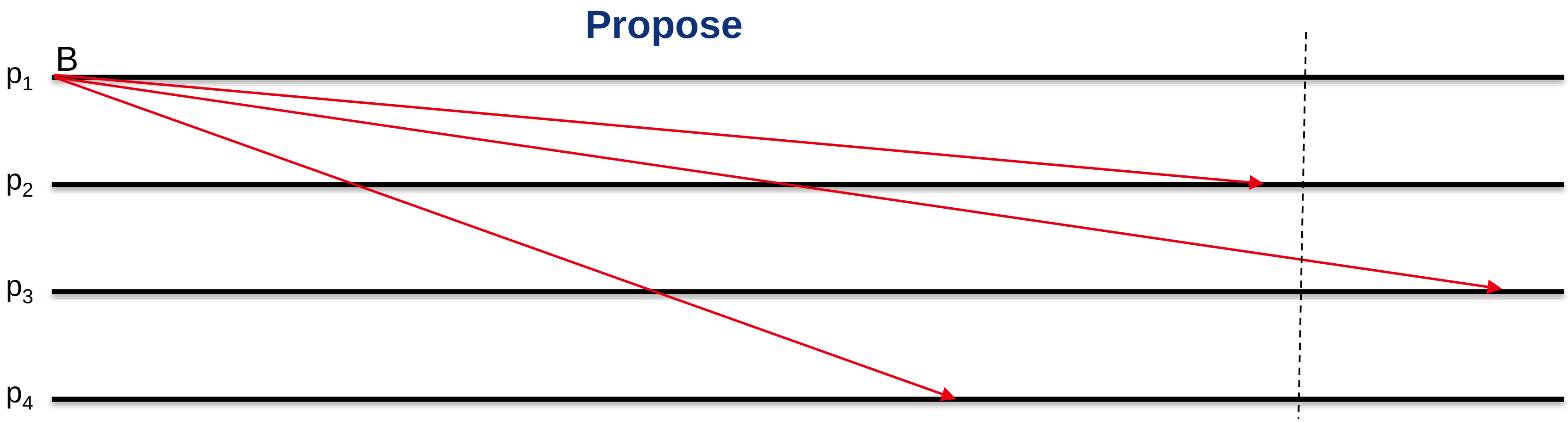
M. J. Fischer, N. A. Lynch, and M. S. Paterson, ‘Impossibility of Distributed Consensus with One Faulty Process’, *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

HOW DOES TENDERMINT WORKS ?



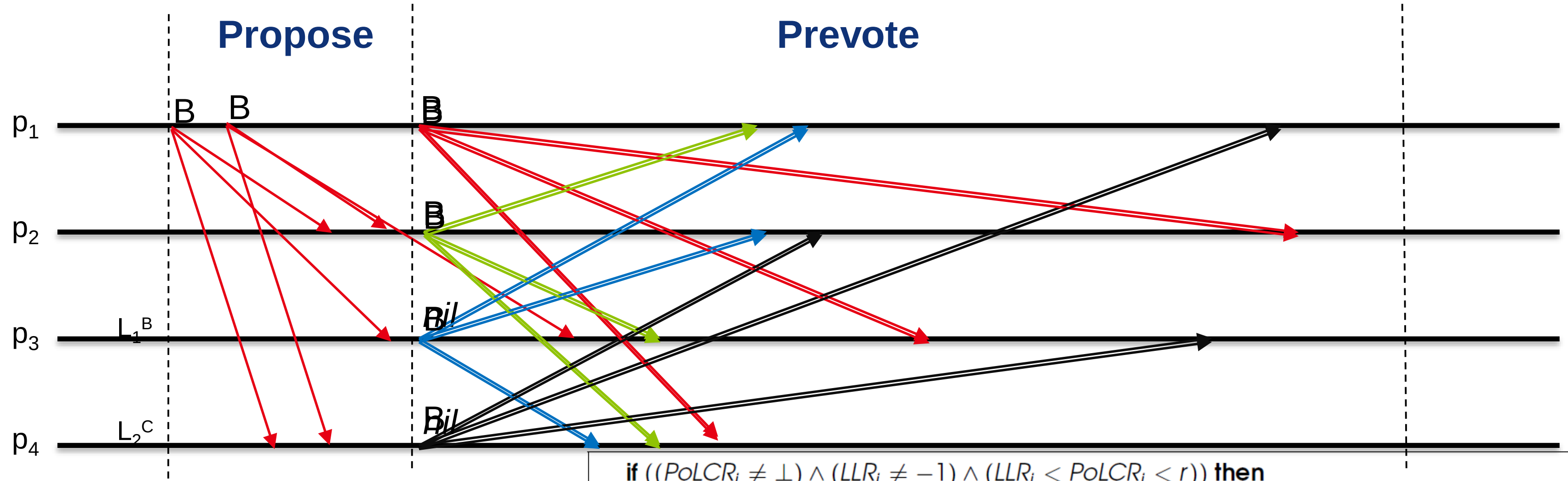
- When a process delivers a message, it broadcasts it

PROPOSE STEP



if $(p_l == \text{proposer}(H, r))$ **then**
  if $(LLR_l \neq -1)$ **then** $PoLCR_l = LLR_l$; $B \leftarrow lockedBlock_l$;
    else $B \leftarrow \text{createNewBlock}(signature)$;
  endif
  trigger broadcast $\langle \texttt{PROPOSE}, (B, H, r, PoLCR_l)_l \rangle$;
else
  set $timerProposer$ to $TimeOutPropose$;
  wait until $((timerProposer \text{ expired}) \vee (proposalReceived_l^{H,r} \neq \perp))$;
  if $((timerProposer \text{ expired}) \wedge (proposalReceived_l^{H,r} == \perp))$ **then**
    $TimeOutPropose \leftarrow TimeOutPropose + 1$;
  endif
endif

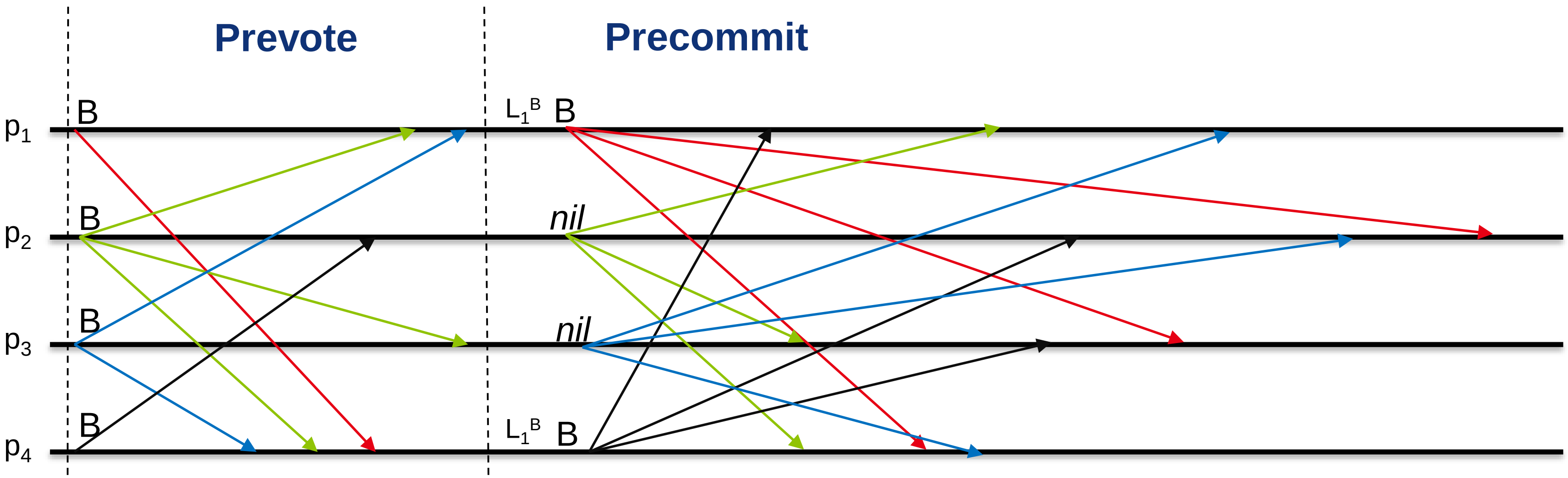
PREVOTE STEP



- p_1 is not locked
- p_2 is not locked
- p_3 locks on B at round 1
- p_4 locks on C at round 2

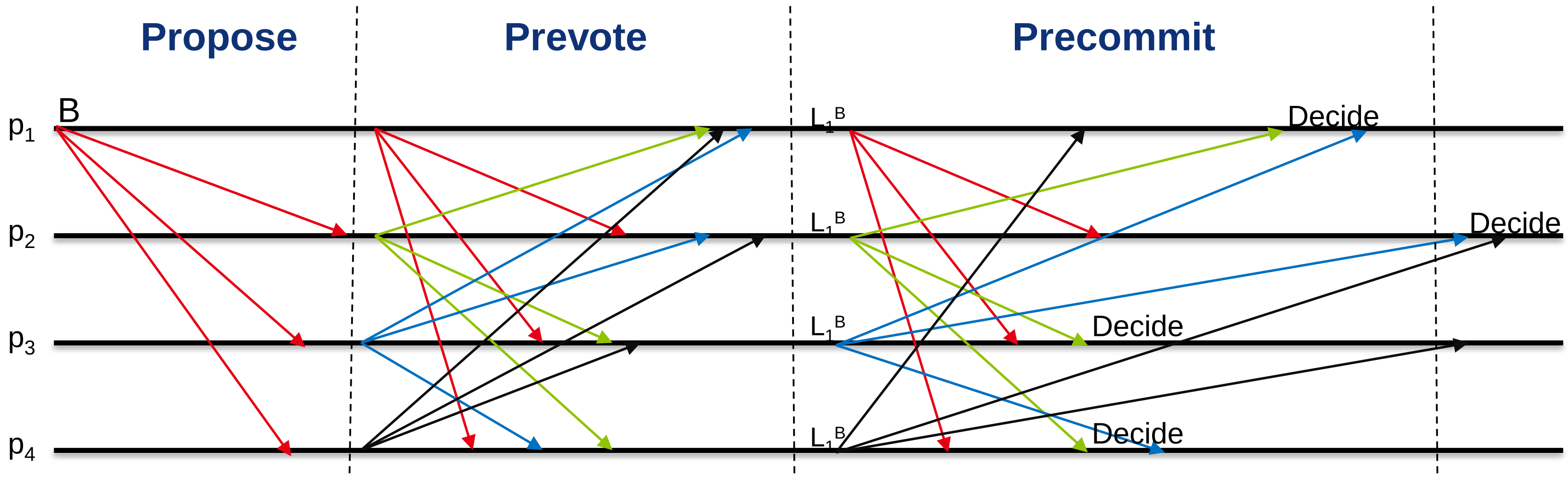
if $((PoLCR_i \neq \perp) \wedge (LLR_i \neq -1) \wedge (LLR_i < PoLCR_i < r))$ **then**
  wait until $|prevotesReceived_i^{H,PoLCR}| > 2/3$;
  if $(\exists B' : (\text{is23Maj}(B', prevotesReceived_i^{H,PoLCR_i})) \wedge (B' \neq lockedBlock_i))$ **then**
    $lockedBlock_i \leftarrow nil$;
  endif
endif
if $(lockedBlock_i \neq nil)$ **then trigger** broadcast $\langle \texttt{PREVOTE}, (lockedBlock_i, H, r)_i \rangle$;
else if $(\text{isValid}(proposalReceived_i^{H,r}))$ **then**
  trigger broadcast $\langle \texttt{PREVOTE}, (proposalReceived_i^{H,r}, H, r)_i \rangle$;
endif
else trigger broadcast $\langle \texttt{PREVOTE}, (nil, H, r)_i \rangle$;
endif
wait until $((\text{is23Maj}(nil, prevotesReceived_i^{H,r})) \vee (\exists B'' : (\text{is23Maj}(B'', prevotesReceived_i^{H,r}))) \vee$
  $(|prevotesReceived_i^{H,r}| > 2/3))$; %Delivery of any 2n/3 prevotes for the round r%
if $(\neg(\text{is23Maj}(nil, prevotesReceived_i^{H,r})) \wedge \neg(\exists B'' : (\text{is23Maj}(B'', prevotesReceived_i^{H,r}))))$ **then**
  set $timerPrevote$ **to** $TimeOutPrevote$;
  wait until ($timerPrevote$ expired) ;
  if ($timerPrevote$ expired) **then** $TimeOutPrevote \leftarrow TimeOutPrevote + 1$; **endif**

PRECOMMIT STEP



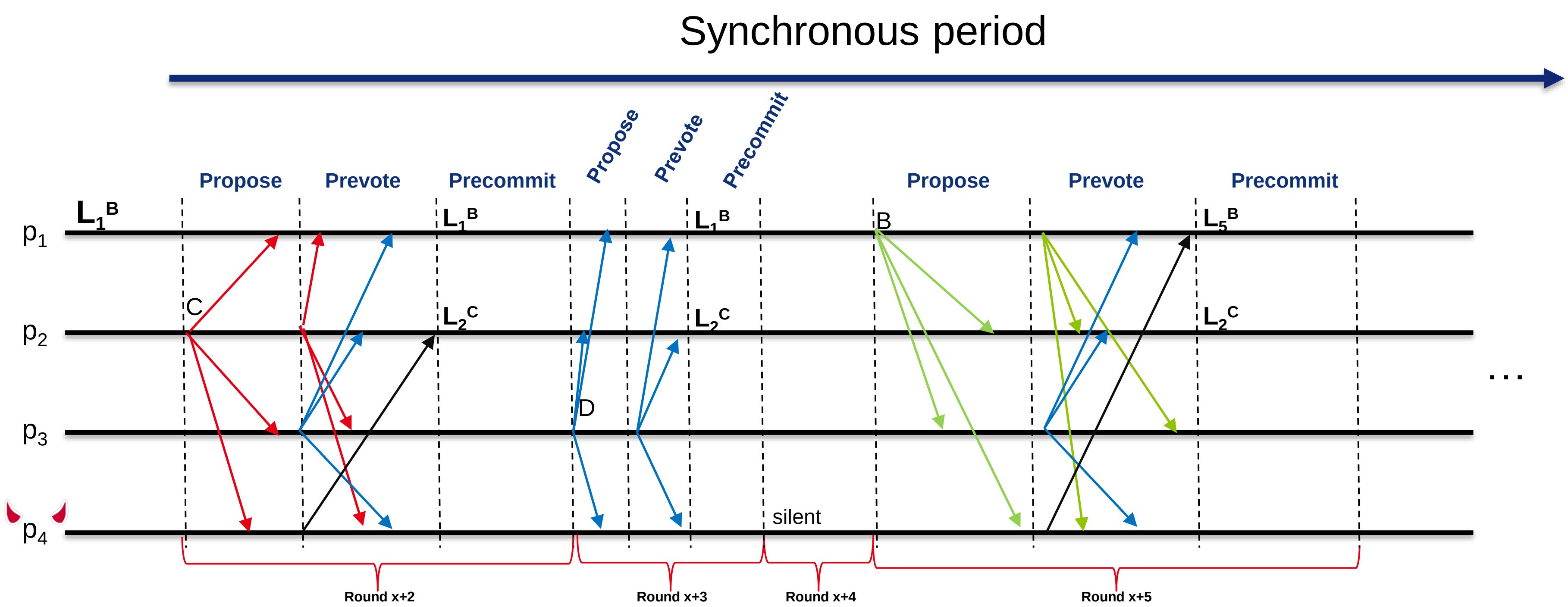
if $(\exists B' : (\text{is23Maj}(B', prevotesReceived_l^{H,r})))$ **then**
    $lockedBlock_l \leftarrow B'$;
    trigger broadcast $\langle \texttt{PRECOMMIT}, (B', H, r)_l \rangle$;
    $LLR_l \leftarrow r$;
else if $(\text{is23Maj}(nil, prevotesReceived_l^{H,r}))$ **then**
    $lockedBlock_l \leftarrow nil$; $LLR_l \leftarrow -1$;
    trigger broadcast $\langle \texttt{PRECOMMIT}, (nil, H, r)_l \rangle$;
endif
else trigger broadcast $\langle \texttt{PRECOMMIT}, (nil, H, r)_l \rangle)$;
endif
wait until $((\text{is23Maj}(nil, prevotesReceived_l^{H,r})) \vee (|precommitsReceived_l^{H,r}| > 2/3))$

EXAMPLE OF EXECUTION



when ($\exists B'$: is23Maj(B', $precommitsReceived_i^{H,r'}$)) :
return B′ ;%Terminate the consensus for the super-round H by deciding B'%

LIVE LOCK



- The live lock occurs because processes do not have the same view at the end of each round
- **Remark**: When *f* > *1*, the byzantine processes need to coordinate to make such attack

TENDERMINT SYSTEM MODEL

- The total number of processes by committee is $n = 3f+1$
  - f is the maximum number of Byzantine process
- The communication is eventually synchronous
- Messages are signed and signatures cannot be forged
- **Additional assumption: Eventually 2f+1 processes will lock on the same proposed value**

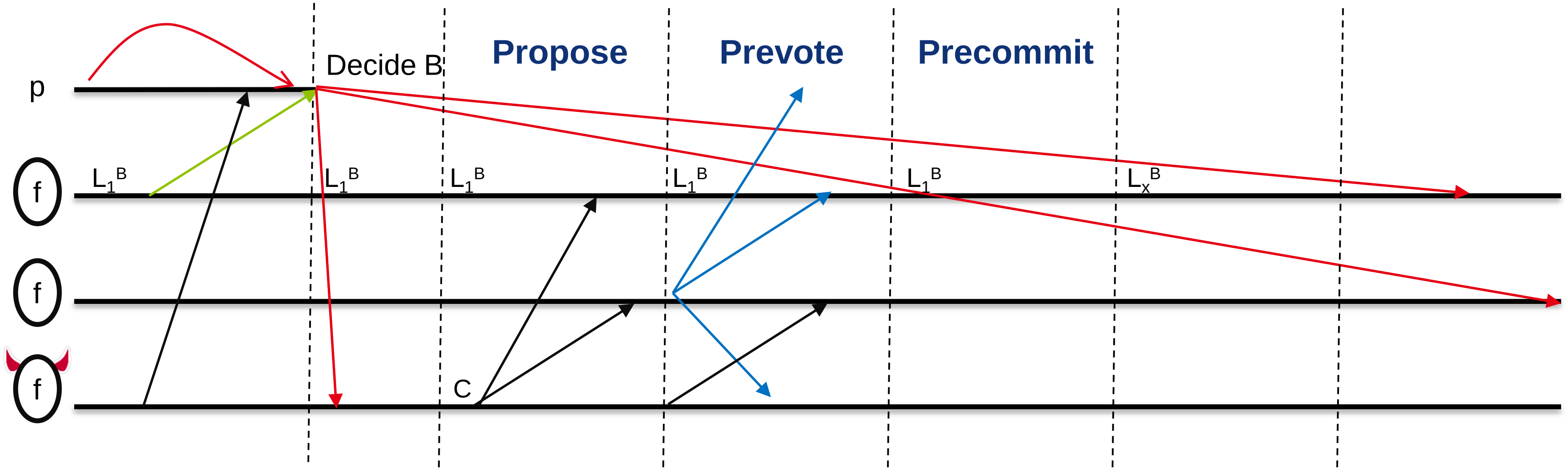
PROOFS SKETCH: TERMINATION

Termination: Every correct process eventually decides some value

- During the **synchronous period**, there is a time from which messages from correct processes are delivered in their corresponding step
- When a correct process p_i is the proposer, it proposal will be prevoted by processes whose locks are smaller than p_i's
  - **Eventually a proposed value will be accepted by at least *2f+1* processes**
  - There will be *2f+1* processes that will prevote
  - Eventually correct processes will deliver them, then will precommit, and decide

PROOFS SKETCH: AGREEMENT

Agreement: If there is a correct process that decides a value B, then eventually all the correct processes decide B



REPEATED CONSENSUS

- **Termination**
- **Agreement**
- **Validity**

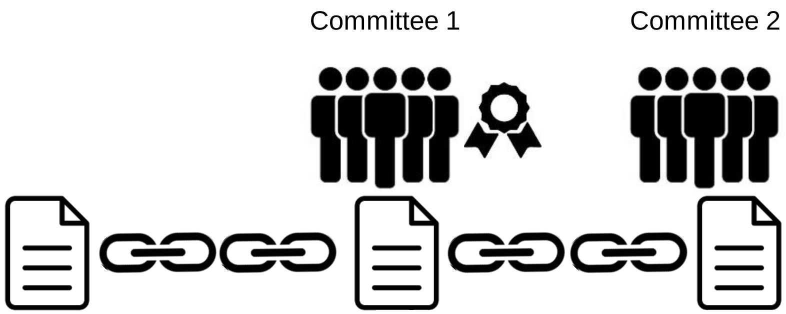
Function repeatedConsensus(Π) ; %Repeated Consensus for the set Π of processes%

Init :
$H \leftarrow 1$ %*Height*% ; $B \leftarrow \perp$; $V \leftarrow \perp$ %Set of validators% ; $TimeOutCommit \leftarrow \Delta_{\text{Commit}}$;
$commitsReceived_i^H \leftarrow \emptyset$; $toReward_i^H \leftarrow \emptyset$;%Set of processes to reward%

---

while (true) **do**
  $B \leftarrow \perp$;
  $V \leftarrow validatorSet(H)$; %Application and blockchain dependant%
  if ($p_i \in V$) **then**
    $B \leftarrow$ consensus($H, V, \boxed{toReward_i^{H-1}}$); %$H^{th}$ Consensus instance%
    trigger broadcast $\langle \texttt{COMMIT}, (B, H)_i \rangle$;
  else
    wait until ($\exists B'$: |atLeastOneThird(B', $commitsReceived_i^H$)|) ;
    $B \leftarrow B'$;
  endif
  set *timerCommit* to *TimeOutCommit*;
  wait until(*timerCommit* expired);
  trigger decide(B);
  $H \leftarrow H + 1$;
endwhile

upon event delivery $\langle \texttt{COMMIT}, (B', H')_j \rangle$:
  if ((($B', H')_j \notin commitsReceived_i^{H'}$) $\wedge$ ($p_j \in validatorSet(H')$)) **then**
    $commitsReceived_i^{H'} \leftarrow commitsReceived_i^{H'} \cup (B', H')_j$;
    $toReward_i^{H'} \leftarrow toReward_i^{H'} \cup p_j$;
    trigger broadcast $\langle \texttt{COMMIT}, (B', H')_j \rangle$;
  endif



MESSAGE COMPLEXITY OF TENDERMINT

- **Tendermint:**
  - Complexity of $O(n^3)$
  - Each round has an $O(n^2)$ message complexity and there can be O(n) rounds
  - Intuitively, there is a View Change each round without sending the whole messages of a round, thanks to the lock mechanism
  - The cost is that process may wait for $2f+1$ rounds before deciding
  - Called the Linear View Change in [2]

- **Classical algorithms such as PBFT [1]:**
  - Complexity of $O(n^4)$
  - Each round has an $O(n^2)$ message complexity, a View-Change has a cost of $O(n)$, and the $f = O(n)$ first rounds may be faulty

[1] M. Castro and B. Liskov, 'Practical Byzantine Fault Tolerance', in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI),* 1999.

[2] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, 'HotStuff: BFT Consensus in the Lens of Blockchain', 2018.

CONCLUSIONS

- **Formalize the version of Tendermint implemented.**
  - Helps identify some bugs
  - Leads to a proposition of a new version which aims to solve the consensus without the assumption
- **Capture in which model Tendermint works**
- **Proof of correctness**

FUTURE WORKS

- **Lower bounds on rounds with the lock mechanism**

- **Incentives**
  - Study of a fair reward mechanism
  - Study of a fair selection mechanism
  - Rational vs Byzantine

```
Function repeatedConsensus(Π) ; %Repeated Consensus for the set Π of processes%

Init :
    H ← 1 %Height% ; B ← ⊥; V ← ⊥ %Set of validators% ; TimeOutCommit ← Δ_Commit;
    commitsReceived_i^H ← ∅ ; toReward_i^H ← ∅ ;%Set of processes to reward%
------------------------------------------------------------------------
while (true) do
    B ← ⊥;
    V ← validatorSet(H); %Application and blockchain dependant%
    if (p_i ∈ V) then
        B ← consensus(H, V, toReward_i^{H−1}); %H^th Consensus instance%
        trigger broadcast ⟨COMMIT, (B, H)_i⟩ ;
    else
        wait until (∃B′ : |atLeastOneThird(B′, commitsReceived_i^H)|) ;
        B ← B′ ;
    endif
    set timerCommit to TimeOutCommit;
    wait until(timerCommit expired);
    trigger decide(B);
    H ← H + 1;
endwhile
```

Y. Amoussou-Guenou, A. Del Pozzo, M. Potop-Butucaru, and S. Tucci-Piergiovanni, ‘Correctness and Fairness of Tendermint-core Blockchains’, *arXiv:1805.08429*, May 2018.

Thank You !