State Machine Replication Among Strangers, Fast and Self-Sufficient

https://ia.cr/2025/616

CRYPTO '25

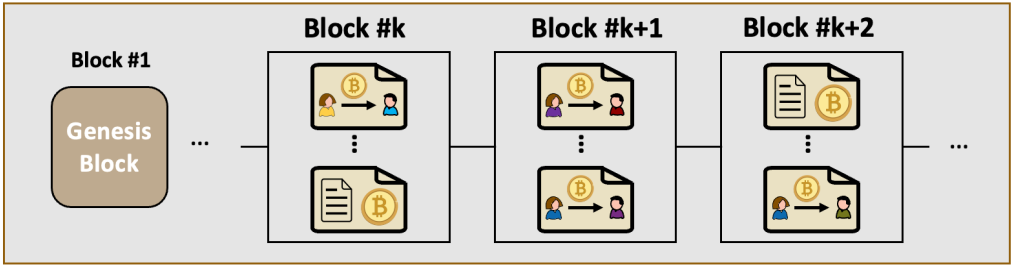
[1]Texas A&M University, [2]University of Edinburgh, [3]IOG

Juan Garay[1], Aggelos Kiayias[2,3], **Yu Shen**[2]

"Among Strangers" (a.k.a. The Permissionless Model)

- The "traditional" distributed system.
  - Nodes are known **a priori**.
  - As in most deployed networks of computers.
- The "permissionless" model.
  - Nodes do **NOT** know each other (not even their exact number!)
  - Nodes come and go.
  - **Anyone** can join.

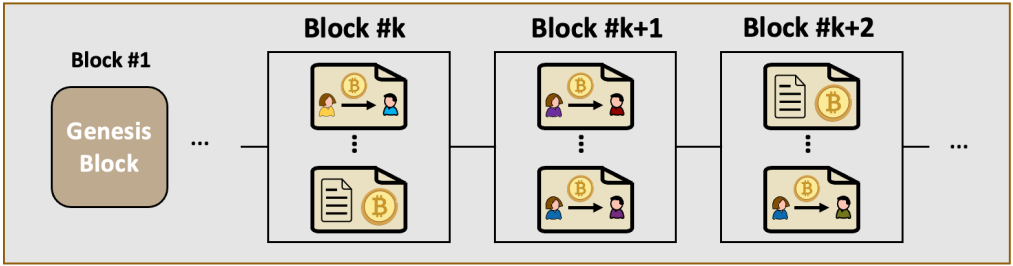
Blockchains (Ledger Consensus/State Machine Replication)



- **Consistency:** $\forall i, j \in \mathcal{H}, t \leq t' : \mathsf{Log}_i[t] \preceq \mathsf{Log}_j^*[t']$.
- **Liveness:** $(\forall i \in \mathcal{H} : \mathsf{tx} \in \mathsf{I}_i[t]) \implies (\forall i \in \mathcal{H} : \mathsf{tx} \in \mathsf{Log}_i[t+u])$.[1]

[1] $\mathsf{Log}_j[t]$ = Log of P_i at time t; $\mathsf{Log}_i^*[t]$ = with transactions in progress. $\mathsf{I}_i[t]$ = transaction input of P_i at time t consistent with $\mathsf{Log}_i[t]$.

Blockchains (Ledger Consensus/State Machine Replication)

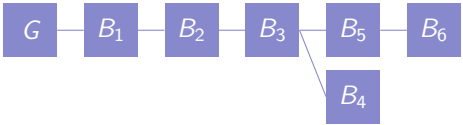


- **Consistency:** $\forall i, j \in \mathcal{H}, t \leq t' : \mathsf{Log}_i[t] \preceq \mathsf{Log}_j^*[t']$.
- **Liveness:** $(\forall i \in \mathcal{H} : \mathsf{tx} \in \mathsf{I}_i[t]) \implies (\forall i \in \mathcal{H} : \mathsf{tx} \in \mathsf{Log}_i[t+u])$.[1]
- More properties are of interest: fast settlement, fairness, self timekeeping, etc.

[1] $\mathsf{Log}_j[t]$ = Log of P_i at time t; $\mathsf{Log}_i^*[t]$ = with transactions in progress. $\mathsf{I}_i[t]$ = transaction input of P_i at time t consistent with $\mathsf{Log}_i[t]$.

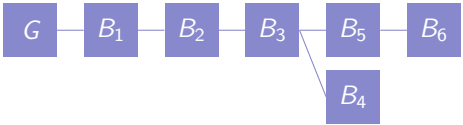
Transaction Settlement Time

- Honest parties ("miners") always choose the **longest** (heaviest) chain they received.
  - If a party wants to **erase** a transaction, it has to find a longer chain!
  - If transaction is "**sufficiently deep**," it cannot do this unless it has a "majority of hashing power."



Transaction Settlement Time

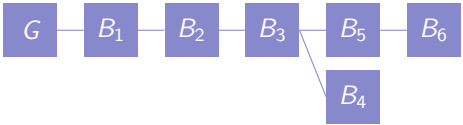
- Honest parties ("miners") always choose the **longest** (heaviest) chain they received.
  - If a party wants to **erase** a transaction, it has to find a longer chain!
  - If transaction is "**sufficiently deep**," it cannot do this unless it has a "majority of hashing power."



- For example, Bitcoin transactions are considered as settled after **6** blocks.
- To be **cryptographically secure**, transactions are settled after $\mathrm{polylog}(\kappa)$ rounds [GKL17].

Transaction Settlement Time

- Honest parties ("miners") always choose the **longest** (heaviest) chain they received.
  - If a party wants to **erase** a transaction, it has to find a longer chain!
  - If transaction is "**sufficiently deep**," it cannot do this unless it has a "majority of hashing power."



- For example, Bitcoin transactions are considered as settled after **6** blocks.
- To be **cryptographically secure**, transactions are settled after $\mathrm{polylog}(\kappa)$ rounds [GKL17].
- *Permissioned* SMR protocols can achieve (expected-)constant settlement time (i.e., "fast").

Fairness in Accessing SMR

- An intriguing question: Who can insert symbols to the permissionless SMR?
  - A special symbol needs to be included to grant access.
  - In Bitcoin: "**coinbase**" transactions.

Fairness in Accessing SMR

- An intriguing question: Who can insert symbols to the permissionless SMR?
  - A special symbol needs to be included to grant access.
  - In Bitcoin: "**coinbase**" transactions.
- **Fairness:** Any honest party gets a chance to introduce a coinbase transaction with probability in proportion to her computational power.

Fairness in Accessing SMR

- An intriguing question: Who can insert symbols to the permissionless SMR?
  - A special symbol needs to be included to grant access.
  - In Bitcoin: "**coinbase**" transactions.
- **Fairness:** Any honest party gets a chance to introduce a coinbase transaction with probability in proportion to her computational power.
- ✗ Bitcoin does **not** achieve fairness.
  - Network delay can create forks.
  - Malicious parties can discard honest blocks (e.g., block withholding attacks).
  - Bitcoin has bad "chain quality" (cf. [GKL15]).

Fairness in Accessing SMR

- An intriguing question: Who can insert symbols to the permissionless SMR?
  - A special symbol needs to be included to grant access.
  - In Bitcoin: "**coinbase**" transactions.
- **Fairness:** Any honest party gets a chance to introduce a coinbase transaction with probability in proportion to her computational power.
- ✗ Bitcoin does **not** achieve fairness.
  - Network delay can create forks.
  - Malicious parties can discard honest blocks (e.g., block withholding attacks).
  - Bitcoin has bad "chain quality" (cf. [GKL15]).
- **Fast Fairness:** Fairness in expected-constant time.

"Self-Sufficient" Protocols

- Hardware clocks are **drifting** clocks.
  - Crystal oscillator drfits by 10 seconds within a day/week/month.
  - Network time protocol (NTP) is typically adopted to synchronize software clocks ("global clock").

https://commons.wikimedia.org/w/index.php?curid=31253284

"Self-Sufficient" Protocols

- Hardware clocks are **drifting** clocks.
  - Crystal oscillator drfits by 10 seconds within a day/week/month.
  - Network time protocol (NTP) is typically adopted to synchronize software clocks ("global clock").

https://commons.wikimedia.org/w/index.php?curid=31253284

- An SMR protocol is said to be **self-sufficient** if it keeps its own time under the mere assumption that parties have drifting local clocks.

Clocks in Bitcoin

- Bitcoin miners use their system clock (synced by NTP) to insert block timestamps.
- If system clock is out-of-sync (when difference from **median** peer clocks exceeds 10 minutes), a warning[2] is pop-up for human operator.

```
bilingual_str msg{strprintf(_(
    "Your computer's date and time appear to be more than %d minutes out of sync with the network, "
    "this may lead to consensus failure. After you've confirmed your computer's clock, this message "
    "should no longer appear when you restart your node. Without a restart, it should stop showing "
    "automatically after you've connected to a sufficient number of new outbound peers, which may "
    "take some time. You can inspect the `timeoffset` field of the `getpeerinfo` and `getnetworkinfo` "
    "RPC methods to get more info."
), Ticks<std::chrono::minutes>(WARN_THRESHOLD))};
```

[2] https://github.com/bitcoin/bitcoin/blob/v29.0/src/node/timeoffsets.cpp

Clocks in Bitcoin

- Bitcoin miners use their system clock (synced by NTP) to insert block timestamps.
- If system clock is out-of-sync (when difference from **median** peer clocks exceeds 10 minutes), a warning[2] is pop-up for human operator.

```
bilingual_str msg{strprintf(_(
    "Your computer's date and time appear to be more than %d minutes out of sync with the network, "
    "this may lead to consensus failure. After you've confirmed your computer's clock, this message "
    "should no longer appear when you restart your node. Without a restart, it should stop showing "
    "automatically after you've connected to a sufficient number of new outbound peers, which may "
    "take some time. You can inspect the `timeoffset` field of the `getpeerinfo` and `getnetworkinfo` "
    "RPC methods to get more info."
), Ticks<std::chrono::minutes>(WARN_THRESHOLD))};
```

→ Bitcoin is **NOT** self-sufficient.

[2] https://github.com/bitcoin/bitcoin/blob/v29.0/src/node/timeoffsets.cpp

Prior Work

- **Fast settlement**
  - ○ Fast settlement for **non-conflicting** transactions (Prism [Bag+19], Ledger Combiner [Fit+20]).
  - ✗ Slow settlement for **conflicting** transactions (Smart contract, **Bitcoin Script**).
- **Fast fairness**
  - ✗ Fairness in polylog(κ) rounds (Fruitchain [PS17]).
- **Self Timekeeping**
  - ✗ Clock synchronization with **imperfect** clocks (Timekeeper [GKS22]).

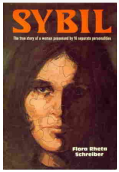
[Bag+19] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. "Prism: Deconstructing the Blockchain to Approach Physical Limits". CCS '19.

[Fit+20] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. "Ledger Combiners for Fast Settlement". TCC '20.

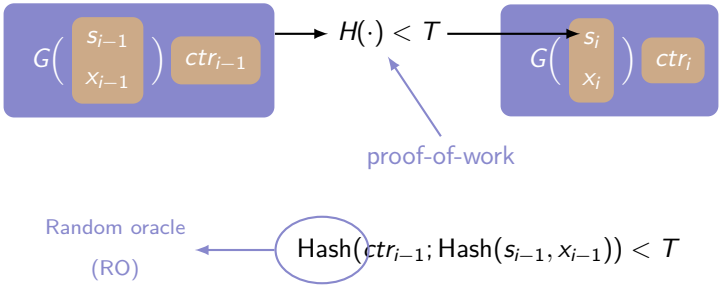
[PS17] Rafael Pass and Elaine Shi. "FruitChains: A Fair Blockchain". PODC '17.

[GKS22] Juan A. Garay, Aggelos Kiayias, and Yu Shen. "Permissionless Clock Synchronization with Public Setup". TCC '22.

Proofs of Work (aka "Crypto Puzzles")

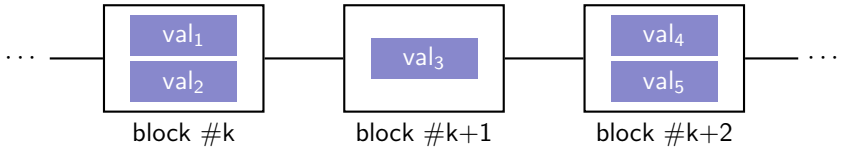


- Moderately hard functions: Spam mitigation, denial of service protection, ...
- Most impactful application: Design of blockchain protocols such as Bitcoin



1/2 Consensus (a.k.a. BA) Protocol [GKL15]

- Parties mine PoWs for each **block** — as in standard Bitcoin backbone protocol.
- Parties mine PoWs for each **input** (value + nonce); they keep transmitting **"PoW-ed" inputs** until they are recorded on chain.



- After the blockchain grows sufficiently, chop off the last $\mathrm{polylog}(\kappa)$ blocks and return the **median** value among unique inputs in the common prefix.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". Eurocrypt '15.

2x1 PoWs: Composition of PoW-based Protocols

Naïve double PoW (Not secure!)

$h \leftarrow G(x, s)$
if $H(h, ctr) < T$ then ...

$h' \leftarrow G(x', s')$
if $H(h', ctr') < T'$ then ...

Given $((x, s), ctr)$
Verify $H(G(x, s), ctr) < T$

Given $((x', s'), ctr')$
Verify $H(G(x', s'), ctr') < T'$

2×1 PoW

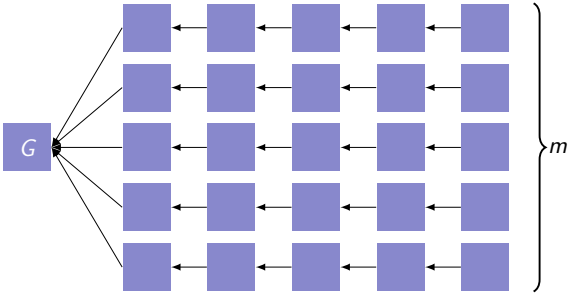
$h \leftarrow G(x, s)$
$h' \leftarrow G(x', s')$
$w \leftarrow H(h, h', ctr)$

if $w < T$ then ...
if $[w]^{\mathrm{R}} < T'$ then ...

Given $((x, s), (*, *), ctr)$
Verify $H(G(x, s), G(*, *), ctr) < T$
Given $((*, *), (x', s'), ctr')$
Verify $H(G(*, *), G(x', s'), ctr') < T'$

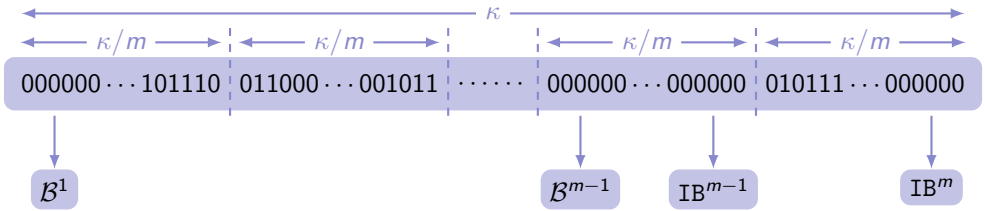
Parallel Blockchains

- **Basic Idea:** Extend 2×1 PoW to m×1 **PoW**.
- Fully independent when $m = \Theta(\text{polylog}\kappa)$.



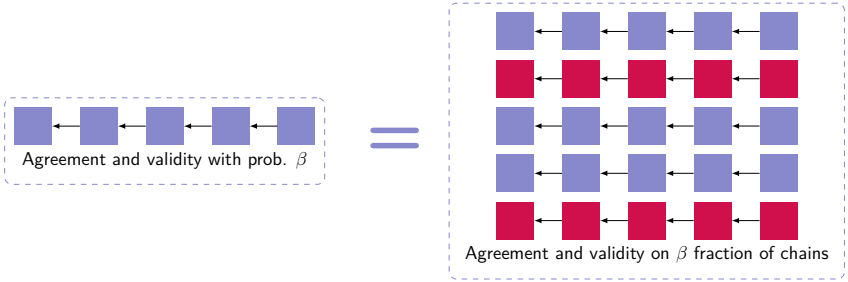
Parallel Blockchains (Cont'd)

- **Basic Idea:** Extend 2×1 PoW to m×1 **PoW**.
- Fully independent when $m = \Theta(\text{polylog}\kappa)$.
- We can run PoW BAs in parallel.
  - 2×1 PoW (block + transaction) in each instance.



Phase-based Parallel Chains

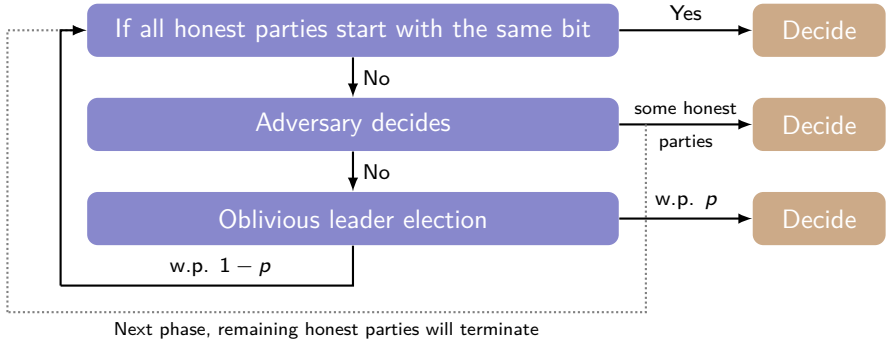
- In [GKL15] (honest-majority PoW consensus in polylog(κ) rounds):
  - Agreement and validity with **overwhelming** prob. after polylog rounds.
  - Agreement and validity with **constant** prob. after **constant** rounds.
- With sufficently many parallel chains:



[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". Eurocrypt '15.

King Consensus [BGP89; FG03]

- Proceeds in phases until termination (In each phase each party has an input bit).



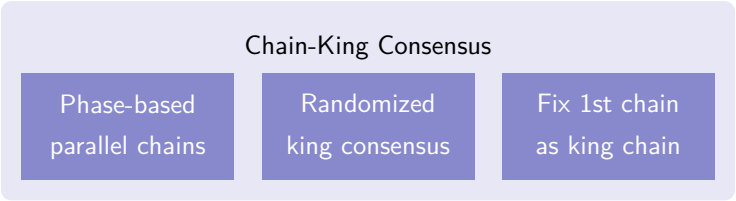
Chain-King Consensus [GKS24]

- Oblivious leader election (OLE) using only RO?
- A simple construction: Fix the **1st chain** as the "King Chain".

[GKS24] Juan A. Garay, Aggelos Kiayias, and Yu Shen. **"Proof-of-Work-Based Consensus in Expected-Constant Time"**. Eurocrypt '24.

Chain-King Consensus [GKS24]

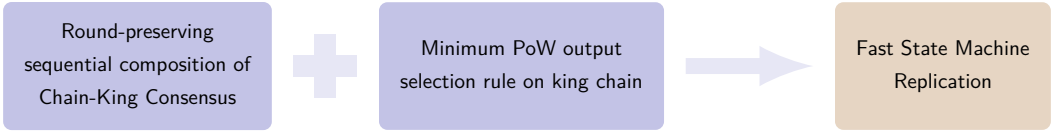
- Oblivious leader election (OLE) using only RO?
- A simple construction: Fix the **1st chain** as the "King Chain".
- With parallel chains, adversary power is "**diluted**" so that he cannot always win on a specific chain.



[GKS24] Juan A. Garay, Aggelos Kiayias, and Yu Shen. "Proof-of-Work-Based Consensus in Expected-Constant Time". Eurocrypt '24.

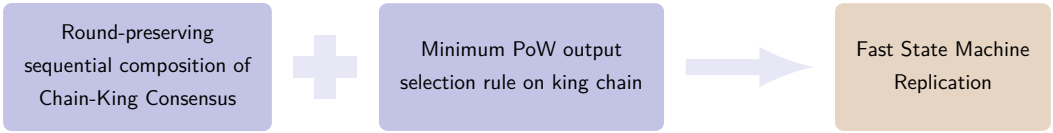
Permissionless SMR with Fast Settlement

- Decide output of king chain using input-block with minimum PoW (smallest hash).
  - With **constant prob.**, an invocation of chain-king consensus outputs a batch of transactions proposed by **honest parties**.



Permissionless SMR with Fast Settlement

- Decide output of king chain using input-block with minimum PoW (smallest hash).
  - With **constant prob.**, an invocation of chain-king consensus outputs a batch of transactions proposed by **honest parties**.



- A few more things have been done here:
  - Extended the above protocol to the **variable** difficulty setting.
  - A bootstrapping algorithm to help fresh parties "catch-up" in **constant** time.

Fast Fairness

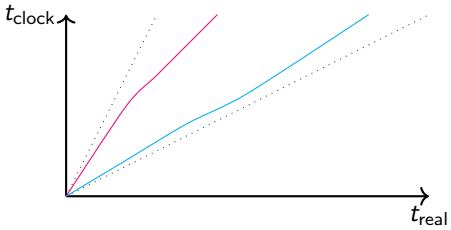
- Bitcoin bears bad “Chain-Quality.”

Fast Fairness

- Bitcoin bears bad "Chain-Quality."
- A "pre-agreement" on coinbase transactions suffices to achieve fast fairness.
  - In the first phase in Chain-King Consensus, parties use 2×1 PoW to mine their own coinbase trasnactions, submit them to the **first** chain.
  - By the end of the phase, parties stick to the coinbase transaction with **minimun hash** in the first chain their local view.
  - Then, a new invocation of Chain-King Consensus starts at the second phase.
- For any party P with p% computational power...
  - With probability p%, P produces the coinbase transaction with minimum hash.
  - With probability $(1 - \epsilon)$, parties agree on P's coinbase transaction on the first chain.

Self Sufficiency

- An SMR protocol is said to be **self-sufficient** if it keeps its own time under the mere assumption that parties have drifting local clocks.

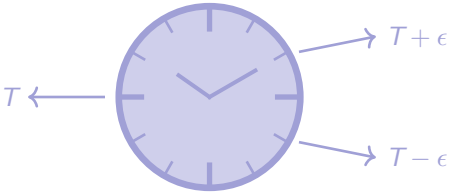


- **Assumption:** Hardware clocks run within a linear envelope[3] of real time.

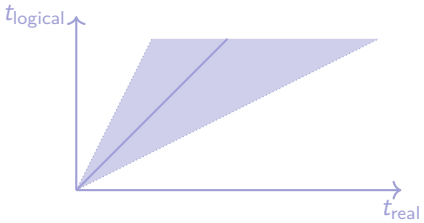
[3] A function $f : \mathbb{R} \to \mathbb{R}$ is within a (U, L)-linear envelope if and only if it holds that $L \cdot x - c \leq f(x) \leq U \cdot x + c$.

Clock Synchronization

- **Bounded Skew:** Honest parties maintain close logical clocks.
- **Accuracy:** Honest parties report logical time within a linear envelope of the real time.



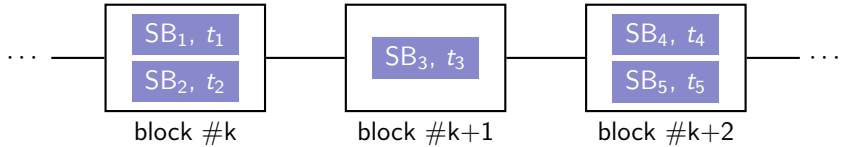
Bounded Skew



Accuracy

Clock Adjustment Algorithm

- Honest-majority PoW-based BA can be extended to timekeeping [GKS22].
  - Inputs record also timestamps (now we call them "synchronization beacons").
  - Parties bookkeep the local arrival time for each beacon.



	SB_1	SB_2	SB_3	SB_4	SB_5
P	t'_1	t'_2	t'_3	t'_4	t'_5

[GKS22] Juan A. Garay, Aggelos Kiayias, and Yu Shen. "Permissionless Clock Synchronization with Public Setup". TCC '22.

Clock Adjustment Algorithm (Cont'd)

- At the end of an interval, parties adjust their local clocks:
  - Compute the difference between timestamp and local arrival time for each beacon on chain.
  - Add the **median** of their differences to local time.

$$\text{new-clock} = \text{old-clock} + \mathbf{med}\{\text{SB}.\mathit{timestamp} - \text{SB}.\mathit{arrivalTime} \mid \text{SB} \in \mathcal{C}\}$$

Clock Adjustment Algorithm (Cont'd)

- At the end of an interval, parties adjust their local clocks:
  - Compute the difference between timestamp and local arrival time for each beacon on chain.
  - Add the **median** of their differences to local time.

$$\text{new-clock} = \text{old-clock} + \textbf{med}\{\text{SB}.\textit{timestamp} - \text{SB}.\textit{arrivalTime} \mid \text{SB} \in \mathcal{C}\}$$

- ✗ This does **NOT** work with **drifting** clocks!
  - To agree on beacons, interval duration needs to be set as $\text{polylog}(\kappa)$
  - → With drifting clocks, skew $= \Theta(\text{polylog}(\kappa))$.
  - It only works for imperfect clocks (cf. [Bad+21; GKS22]).

Clock Adjustment Algorithm (Cont'd)

- At the end of an interval, parties adjust their local clocks:
  - Compute the difference between timestamp and local arrival time for each beacon on chain.
  - Add the **median** of their differences to local time.

$$\text{new-clock} = \text{old-clock} + \textbf{med}\{\text{SB}.\textit{timestamp} - \text{SB}.\textit{arrivalTime} \mid \text{SB} \in \mathcal{C}\}$$

- ✗ This does **NOT** work with **drifting** clocks!
  - To agree on beacons, interval duration needs to be set as $\text{polylog}(\kappa)$
  - → With drifting clocks, skew $= \Theta(\text{polylog}(\kappa))$.
  - It only works for imperfect clocks (cf. [Bad+21; GKS22]).
- **Solution:** Use parallel blockchains to acquire a set of clock values in **constant** time, where a large fraction of them are "good."

Clock Synchronization

- **Problem:** A small fraction of **unknown** parallel chains suggest malicious clock values!

Clock Synchronization

- **Problem:** A small fraction of **unknown** parallel chains suggest malicious clock values!
- **Example 1:** Malicious parties suggest crazy clock values.

- **Solution*:** Prune the η largest and smallest clocks.

Clock Synchronization

- **Problem:** A small fraction of **unknown** parallel chains suggest malicious clock values!
- **Example 1:** Malicious parties suggest crazy clock values.

- **Solution*:** Prune the η largest and smallest clocks.
- **Example 2:** Malicious parties suggest confusing clock values.

Clock Synchronization (Cont'd)

- **Solution:** Apply **Approximate Agreement** [Dol+86] over the set of clock values.
  1. **Reduce:** Remove the η largest and smallest elements.
  2. **Select:** Divide the remaining clocks into chunks of η elements; for each chunk, select a representative (the first element).
  3. Return the **average** over clock representatives.

[Dol+86] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. "Reaching approximate agreement in the presence of faults". J. ACM.

Clock Synchronization (Cont'd)

- **Solution:** Apply **Approximate Agreement** [Dol+86] over the set of clock values.
  1. **Reduce:** Remove the η largest and smallest elements.
  2. **Select:** Divide the remaining clocks into chunks of η elements; for each chunk, select a representative (the first element).
  3. Return the **average** over clock representatives.

$$\text{new-clock} \triangleq \textbf{avg}(\textbf{select}(\textbf{reduce}(\langle \text{clock}_1, \ldots \text{clock}_m \rangle, \eta), \eta)), \text{ where}$$

$$\text{clock}_i \triangleq \text{old-clock} + \textbf{med}\{\texttt{SB.timestamp} - \texttt{SB.arrivalTime} \mid \texttt{SB} \in \mathbb{C}_i\}.$$

[Dol+86] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. "Reaching approximate agreement in the presence of faults". J. ACM.

Clock Synchronization (Cont'd)

- **Solution:** Apply **Approximate Agreement** [Dol+86] over the set of clock values.
  1. **Reduce:** Remove the η largest and smallest elements.
  2. **Select:** Divide the remaining clocks into chunks of η elements; for each chunk, select a representative (the first element).
  3. Return the **average** over clock representatives.

$$\text{new-clock} \triangleq \textbf{avg}(\textbf{select}(\textbf{reduce}(\langle \text{clock}_1, \ldots \text{clock}_m \rangle, \eta), \eta)), \text{ where}$$

$$\text{clock}_i \triangleq \text{old-clock} + \textbf{med}\{\text{SB.timestamp} - \text{SB.arrivalTime} \mid \text{SB} \in \mathbb{C}_i\}.$$

- **Result:** Clocks with **bounded skews** (linear w.r.t. network delay and clock drfit rate).

[Dol+86] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. "Reaching approximate agreement in the presence of faults". J. ACM.

Takeaways

In the permissionless PoW setting with drifting clocks, we achieve:

- **Fast Settlement**
  - **All** incoming transactions are confirmed in expected-constant time.
- **Fast Fairness**
  - A coinbase transaction, selected w.p. proportional to computational power, is introduced every expected-constant-time interval.
- **Self-Sufficient Timekeeping**
  - Protocol participants maintain bounded skews.
  - State machine exports an accurate SMR time w.r.t. real time.

Thank You

[GKS25] https://ia.cr/2025/616

References

[Bad+21] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. "Dynamic Ad Hoc Clock Synchronization". In: **Advances in Cryptology – EUROCRYPT 2021, Part III**. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12698. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Cham, Switzerland, 2021, pp. 399–428.

[Bag+19] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. "Prism: Deconstructing the Blockchain to Approach Physical Limits". In: **ACM CCS 2019: 26th Conference on Computer and Communications Security**. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. London, UK: ACM Press, 2019, pp. 585–602.

[BGP89] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. "Towards Optimal Distributed Consensus (Extended Abstract)". In: **30th Annual Symposium on Foundations of Computer Science**. Research Triangle Park, NC, USA: IEEE Computer Society Press, 1989, pp. 410–415.

[Dol+86] Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. "Reaching approximate agreement in the presence of faults". In: **J. ACM** 33.3 (1986), pp. 499–516.

References

[FG03] Matthias Fitzi and Juan A. Garay. "Efficient player-optimal protocols for strong and differential consensus". In: **22nd ACM Symposium Annual on Principles of Distributed Computing**. Ed. by Elizabeth Borowsky and Sergio Rajsbaum. Boston, MA, USA: Association for Computing Machinery, 2003, pp. 211–220.

[Fit+20] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. "Ledger Combiners for Fast Settlement". In: **TCC 2020: 18th Theory of Cryptography Conference, Part I**. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12550. Lecture Notes in Computer Science. Durham, NC, USA: Springer, Cham, Switzerland, 2020, pp. 322–352.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications". In: **Advances in Cryptology – EUROCRYPT 2015, Part II**. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9057. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer Berlin Heidelberg, Germany, 2015, pp. 281–310.

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol with Chains of Variable Difficulty". In: **Advances in Cryptology – CRYPTO 2017, Part I**. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Cham, Switzerland, 2017, pp. 291–323.

[GKS22] Juan A. Garay, Aggelos Kiayias, and Yu Shen. "Permissionless Clock Synchronization with Public Setup". In: **TCC 2022: 20th Theory of Cryptography Conference, Part III**. Ed. by Eike Kiltz and Vinod Vaikuntanathan. Vol. 13749. Lecture Notes in Computer Science. Chicago, IL, USA: Springer, Cham, Switzerland, 2022, pp. 181–211.

References

[GKS24] Juan A. Garay, Aggelos Kiayias, and Yu Shen. "Proof-of-Work-Based Consensus in Expected-Constant Time". In: **Advances in Cryptology – EUROCRYPT 2024, Part III**. Ed. by Marc Joye and Gregor Leander. Vol. 14653. Lecture Notes in Computer Science. Zurich, Switzerland: Springer, Cham, Switzerland, 2024, pp. 96–125.

[GKS25] Juan A. Garay, Aggelos Kiayias, and Yu Shen. "State Machine Replication Among Strangers, Fast and Self-sufficient". In: **Advances in Cryptology – CRYPTO 2025**. Ed. by Yael Tauman Kalai and Seny F. Kamara. Cham: Springer Nature Switzerland, 2025, pp. 3–36.

[PS17] Rafael Pass and Elaine Shi. "FruitChains: A Fair Blockchain". In: **36th ACM Symposium Annual on Principles of Distributed Computing**. Ed. by Elad Michael Schiller and Alexander A. Schwarzmann. Washington, DC, USA: Association for Computing Machinery, 2017, pp. 315–324.