# Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain

**Bernardo David**
Tokyo Tech
& IOHK

**Peter Gaži**
IOHK

**Aggelos Kiayias**
U. Edinburgh
& IOHK

**Alexander Russell**
U. Connecticut

Eurocrypt 2018

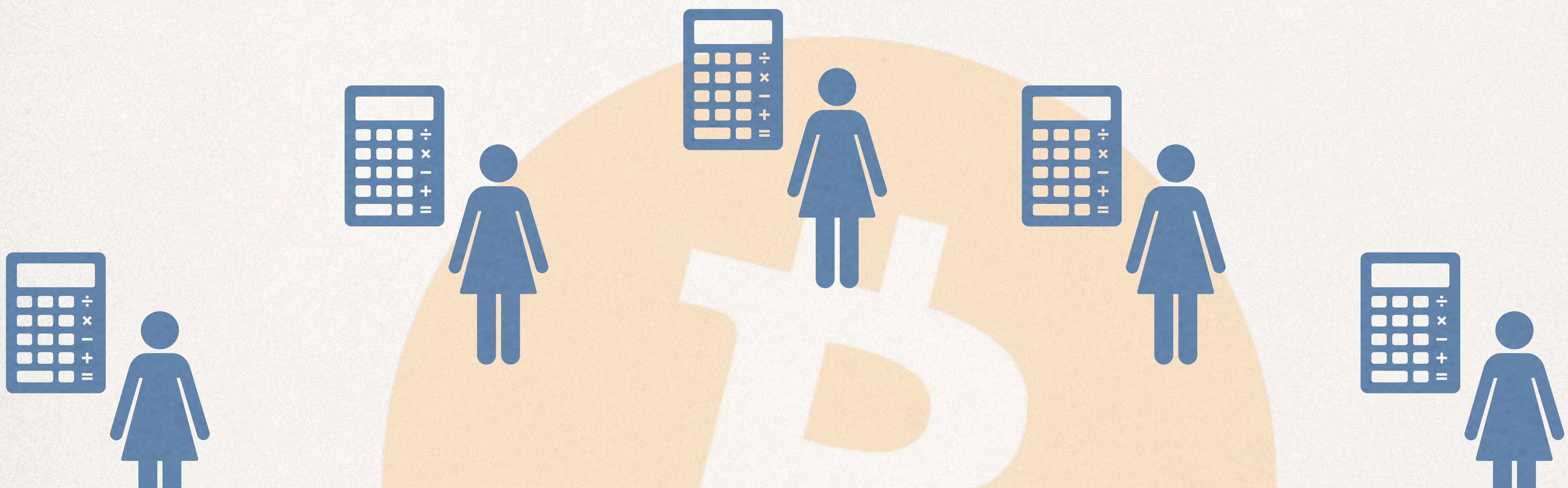
# Roadmap

- **Proof-of-work vs. Proof-of-stake blockchains**
- **Ouroboros Praos**
    - Protocol Description
    - Security Analysis

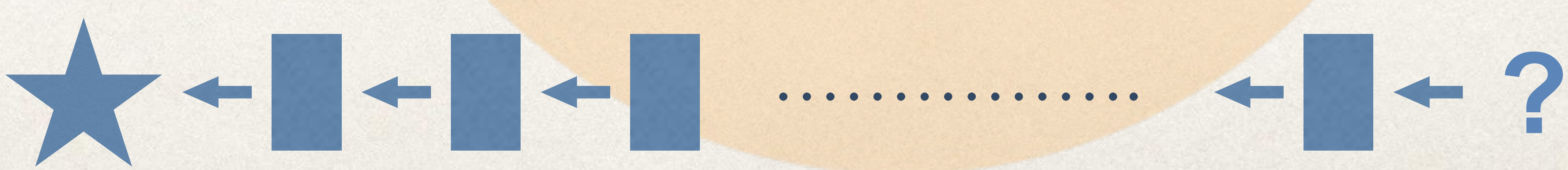
# The problem Bitcoin solves

- Allows  a collection of parties to agree on a dynamic, common sequence of transactions—a **ledger**.
    - **persistence:** past transactions in ledger are immutable
    - **liveness:** new transactions are eventually included
    - parties may arise and disappear
    - some parties may seek to disrupt the system

# Bitcoin as a leader election process, proof of work



- parties compete for the right to extend
- winning certificate: PoW solution
- *Pr[success]* proportional to computing power



# Bitcoin: Laudatory remarks

- **Simple**
  - neatly solves a challenge: consensus with a fluid population of participants
- **Sidesteps previous impossibility results**
  - thanks to a new assumption (honest majority of comp. power)
- **Amenable to formal analysis**
  - [GKL15,PSS17,BMTZ17]

# Bitcoin: Criticism

- relies on an ongoing **computational race**
- power consumption estimates:
  - on the order of GWs
  - almost **tripled over the last 6 months**

- **Attack cost proportional to the energy spent in the attack period.**

# Challenge: Replace “proof-of-work” with **alternate resource** lottery

- other physical resources, with different properties
  - disk space
  - useful computation/storage
  - ...
- virtual resource: coin itself

⇒ **Proof of Stake**

# **Proof of Stake:**
# stake-based lottery

- blockchain tracks ownership of coins among parties
- **Idea:** participants elected proportionally to stake

⇒ no need for physical resources

- hard to implement securely

# Previous proof-of-stake solutions with rigorous guarantees

Eventual (Nakamoto-style) Consensus:

- **Ouroboros** [KRDO16]
- **Snow White** [DPS16]

Blockwise Byzantine Agreement:

- **Algorand** [CM16]

# Ouroboros

Provably guarantees

- **persistence:** stable transactions are immutable
- **liveness:** new transactions included eventually

# Ouroboros

Provably guarantees

- **persistence:** stable transactions are immutable
- **liveness:** new transactions included eventually

if

- adversary has minority stake throughout
- adversary subject to corruption delay
- communication is synchronous

# Ouroboros Praos

Provably guarantees

- **persistence:** stable transactions are immutable
- **liveness:** new transactions included eventually

if

- adversary has minority stake throughout
- ~~adversary subject to corruption delay~~
- ~~communication is synchronous~~

# Ouroboros Praos in a Nutshell

First eventual-consensus PoS **secure**

- **in a semi-synchronous communication model**
- **despite fully adaptive corruptions**

via

- local, private leader selection
- forward-secure signatures
- blockchain hashing for randomness (scalability!)

# Ouroboros Praos:

Protocol Description

# Communication Model

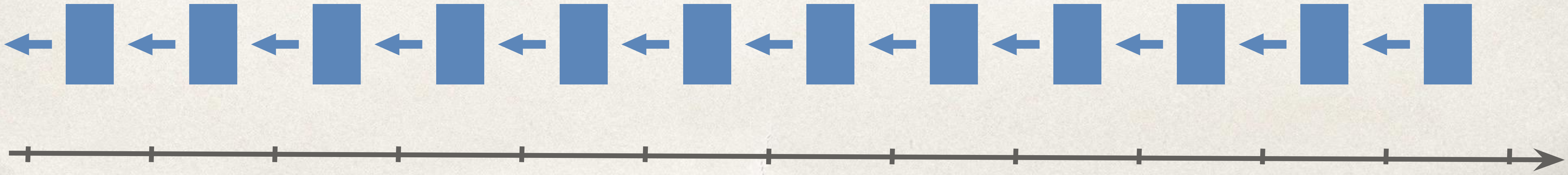
- assume **synchronized clocks**
- time divided into **slots**
- honest messages may be adversarially delayed by at most **Δ** slots
  - **Δ** is unknown to the protocol
- adversary may send arbitrary messages to arbitrary subsets, arriving at arbitrary times

# Ouroboros Praos: Protocol overview

- time divided into consecutive, disjoint **slots**

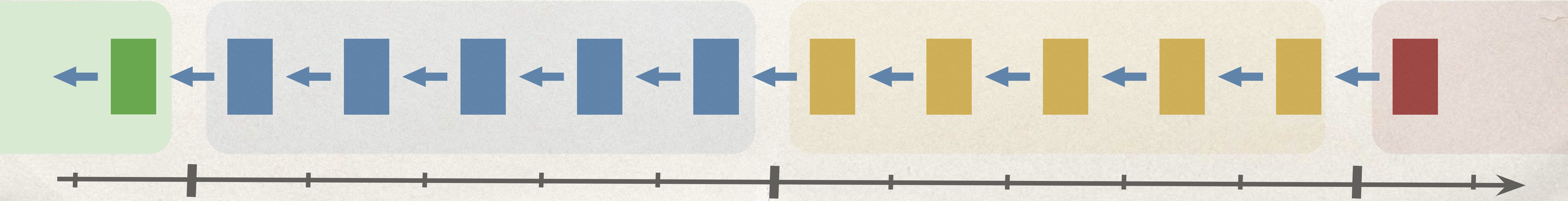
# Ouroboros Praos: Protocol overview

- time divided into consecutive, disjoint **slots**
  - at most 1 block per slot allowed



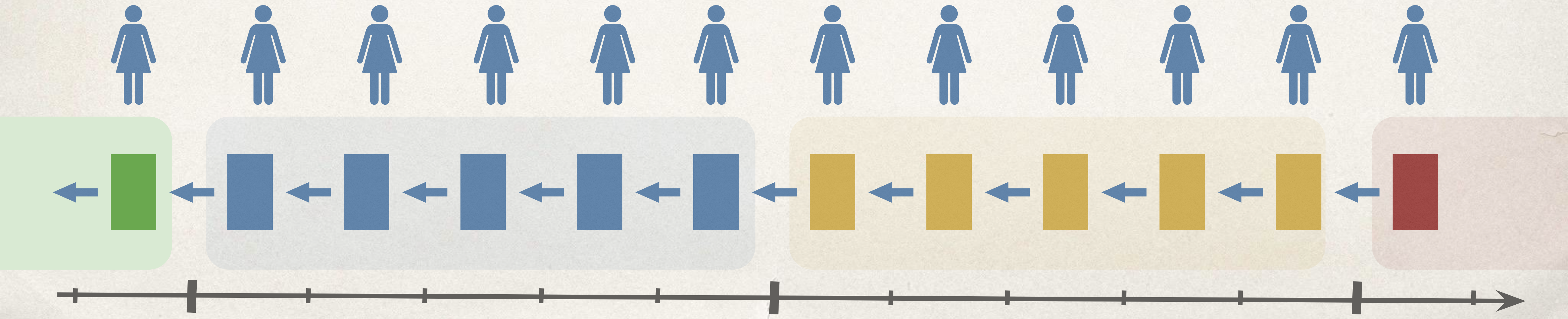
# Ouroboros Praos: Protocol overview

- time divided into consecutive, disjoint **slots**
- **epoch**: sequence of R slots



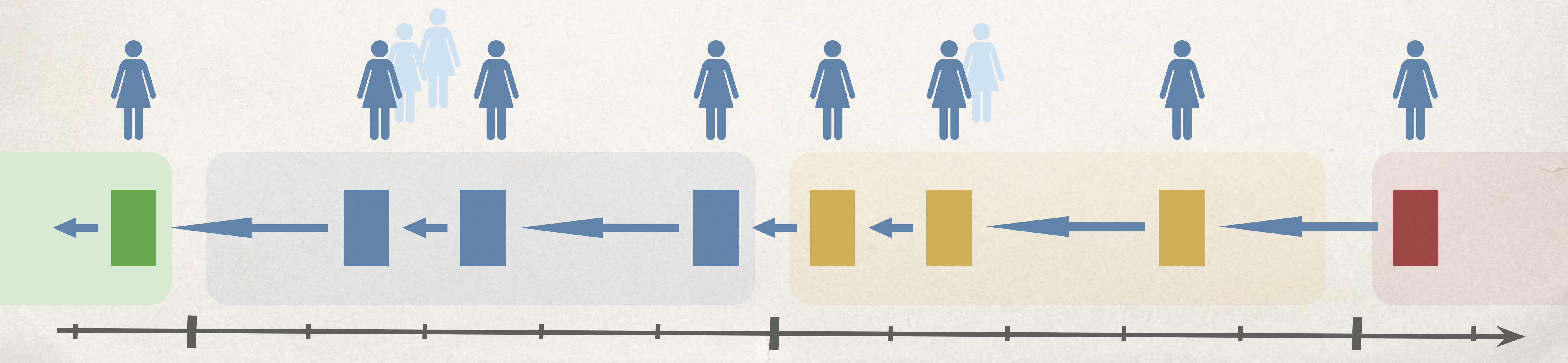
# Ouroboros Praos: Protocol overview

- time divided into consecutive, disjoint **slots**
- **epoch**: sequence of R slots
- **slot leader**: a player allowed to create block in that slot
  - selected proportionally to his/her stake



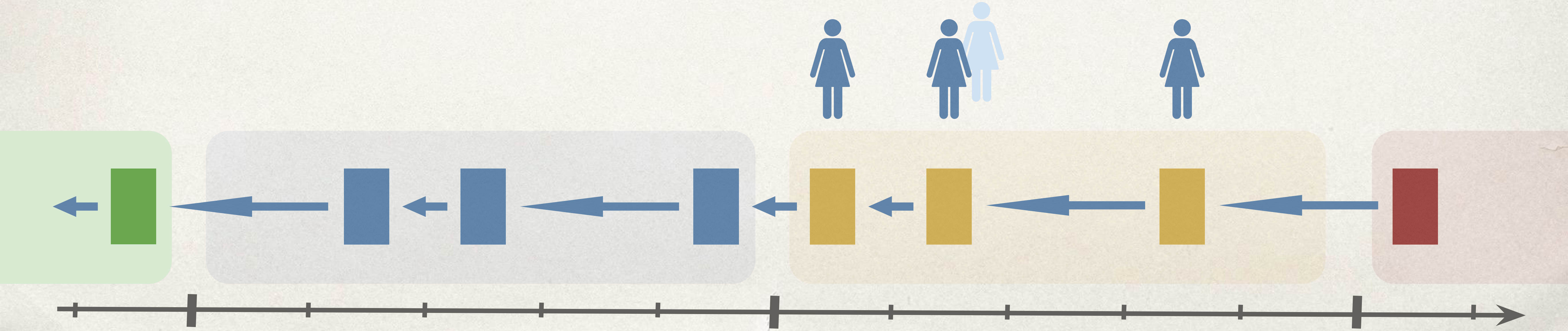
# Ouroboros Praos: Protocol overview

- time divided into consecutive, disjoint **slots**
- **epoch**: sequence of R slots
- **slot leader**: a player allowed to create block in that slot
  - selected proportionally to his/her stake
  - independent for each slot and each player
    - => empty slots, multi-leader slots



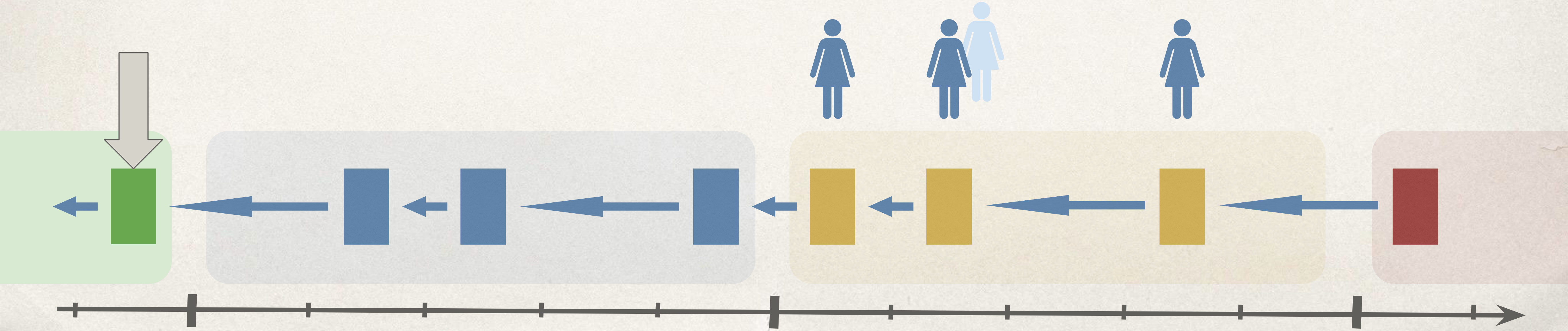
# Ouroboros Praos: Protocol overview

- time divided into consecutive, disjoint **slots**
- **epoch**: sequence of R slots
- **slot leader**: a player allowed to create block in that slot
    - selected proportionally to his/her stake
    - independent for each slot and each player
        - => empty slots, multi-leader slots



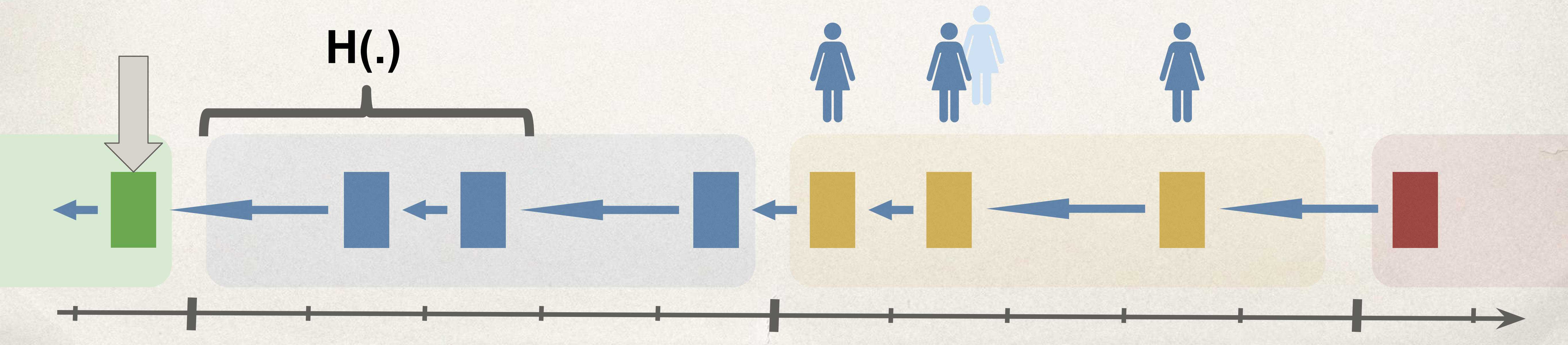
# Ouroboros Praos: Protocol overview

- time divided into consecutive, disjoint **slots**
- **epoch**: sequence of R slots
- **slot leader**: a player allowed to create block in that slot
- **stake distribution**: snapshot from last block 2 epochs ago



# Ouroboros Praos: Protocol overview

- time divided into consecutive, disjoint **slots**
- **epoch**: sequence of R slots
- **slot leader**: a player allowed to create block in that slot
- **stake distribution**: snapshot from last block 2 epochs ago
- **randomness**: hash of values in prefix of previous epoch

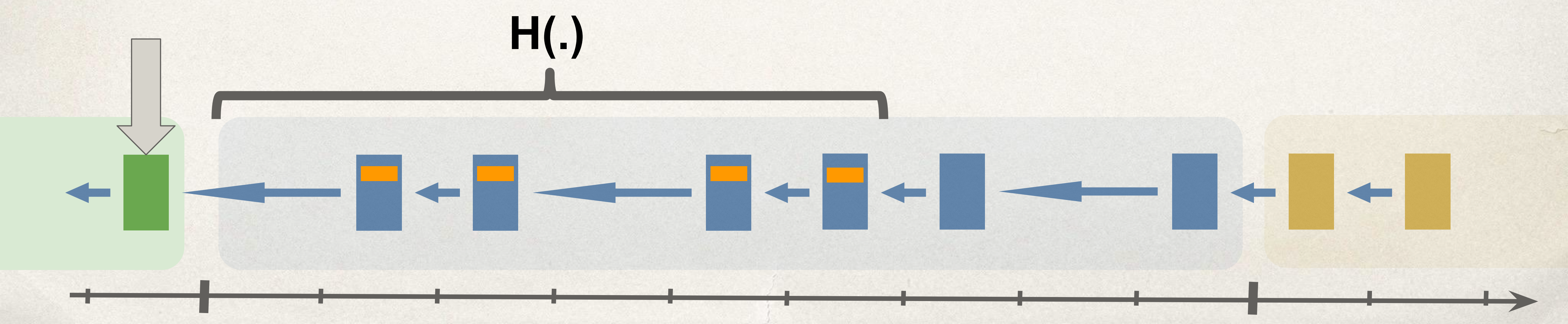


# Hashing for epoch randomness

unique,
pseudorandom

Verifiable Random Functions:

- **Evaluate**$_{sk}$($input$) = ($output$, $proof$)
- **Verify**$_{pk}$($input$, $output$, $proof$) = 0/1

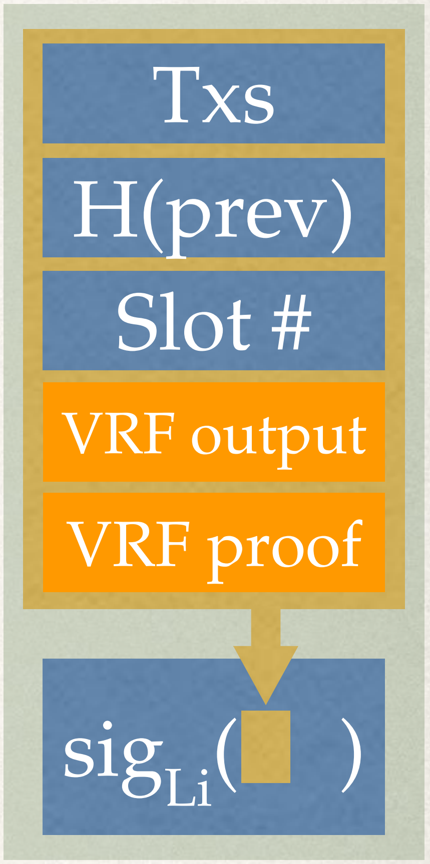


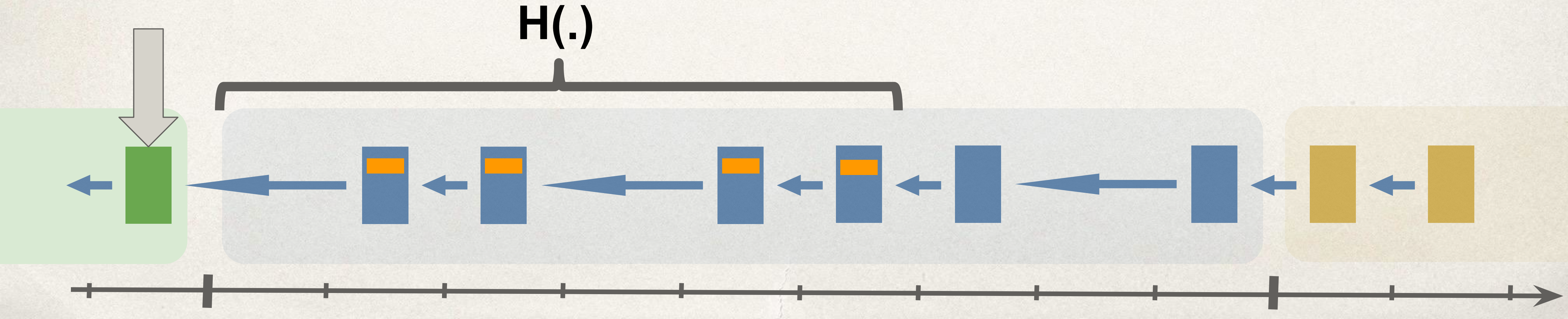
# Hashing for epoch randomness

Verifiable Random Functions:

- **Evaluate**$_{sk}$$(input) = (output, proof)$
- **Verify**$_{pk}$$(input, output, proof) = 0/1$

- every leader inserts a separate VRF **(value,proof)** into block

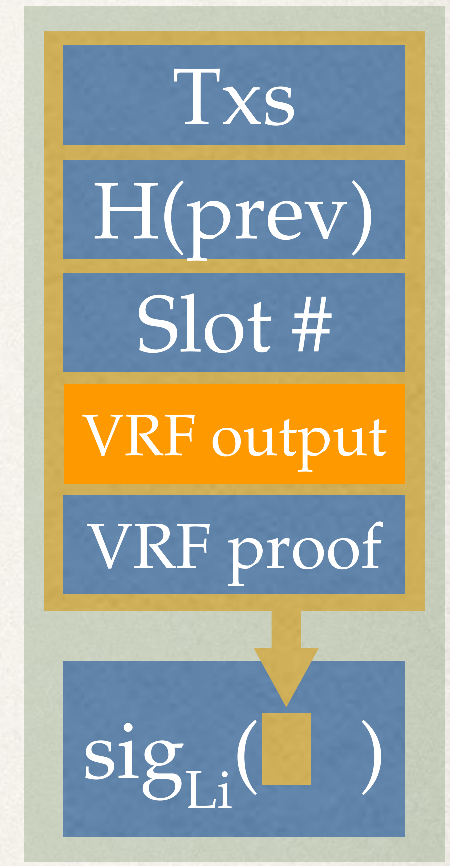


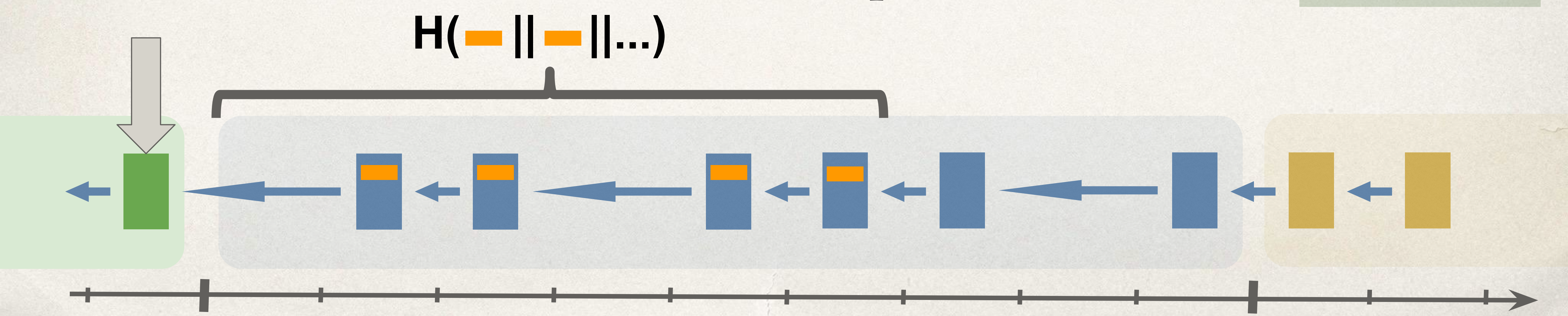
# Hashing for epoch randomness

Verifiable Random Functions:

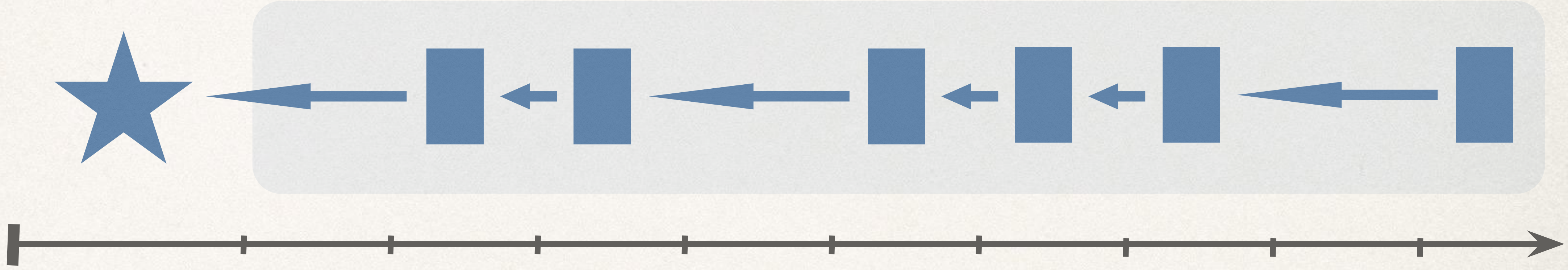
- $\textbf{Evaluate}_{sk}(\textit{input}) = (\textit{output}, \textit{proof})$
- $\textbf{Verify}_{pk}(\textit{input}, \textit{output}, \textit{proof}) = 0/1$




- every leader inserts a separate VRF **(value,proof)** into block
- hash of VRF values from initial ⅔ of epoch give randomness for the whole next epoch



# Single-epoch setting



Focus on one epoch of length R

- static **stake distribution**
- ideal **randomness**

# Leader selection: local, private

Verifiable Random Functions:

- $\textbf{Evaluate}_{sk}(input) = (output, proof)$
- $\textbf{Verify}_{pk}(input, output, proof) = 0/1$

Leader selection lottery for stakeholder $U_i$:

$$\textbf{Evaluate}_{\mathrm{sk}}(rnd, slot) < \phi(stake_{\mathrm{i}})$$

(*output*,*proof*)
included in the block

# Leader selection: local, private

Verifiable Random Functions:

- $\textbf{Evaluate}_{sk}(\mathit{input}) = (\mathit{output}, \mathit{proof})$
- $\textbf{Verify}_{pk}(\mathit{input}, \mathit{output}, \mathit{proof}) = 0/1$

Leader selection lottery for stakeholder $U_i$:

$$\textbf{Evaluate}_{\mathrm{sk}}(\mathit{rnd}, \mathit{slot}) < \phi(\mathit{stake}_{\mathrm{i}})$$

- similar idea previously in NXT, Algorand
- needs **unpredictability under malicious key generation**
- UC-functionality + efficient realization from CDH+RO

# Leader selection: local, private

Verifiable Random Functions:

- $\textbf{Evaluate}_{sk}(\mathit{input}) = (\mathit{output}, \mathit{proof})$
- $\textbf{Verify}_{pk}(\mathit{input}, \mathit{output}, \mathit{proof}) = 0/1$

Leader selection lottery for stakeholder $U_i$:

$$\textbf{Evaluate}_{\mathrm{sk}}(\mathit{rnd}, \mathit{slot}) < \boxed{\phi(\mathit{stake}_{\mathrm{i}})}$$

- similar idea previously in NXT, Algorand
- needs **unpredictability under malicious key generation**
- UC-functionality + efficient realization from CDH+RO

# Leader selection: choice of $\phi(.)$

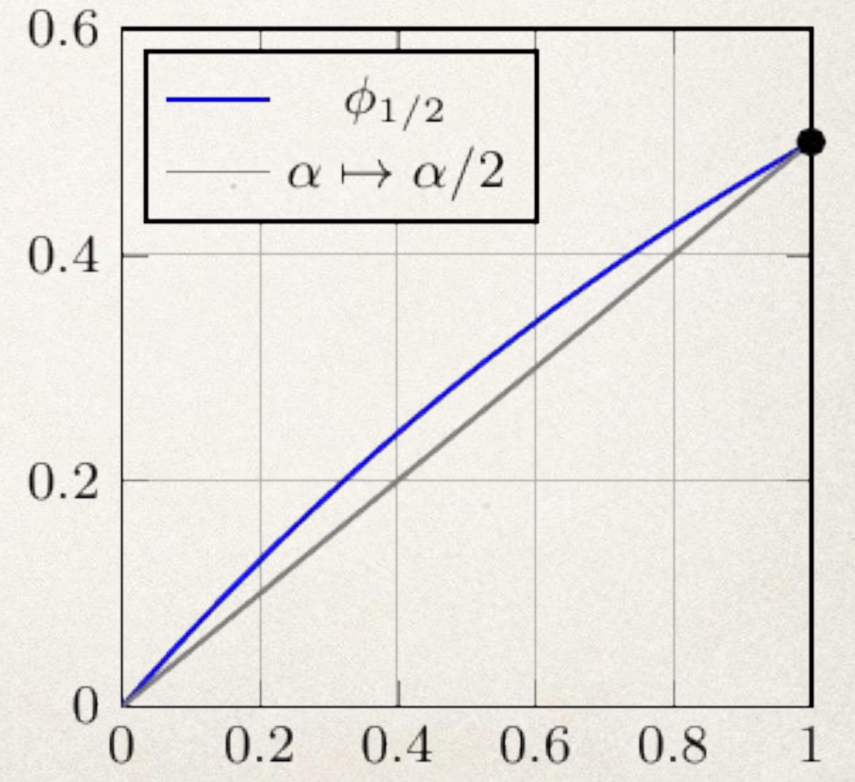
$$\phi_f(\alpha) \triangleq 1 - (1 - f)^\alpha$$

$\alpha \in [0,1]$

$f \in [0,1]$

- **ratio of non-empty slots** $f$ is a protocol parameter
- slightly sublinear growth
- maintains "independent aggregation"

$$1 - \phi\left(\sum_i \alpha_i\right) = \prod_i (1 - \phi(\alpha_i))$$



# Block signing: Key-evolving signatures

**KES** are signature schemes, where:

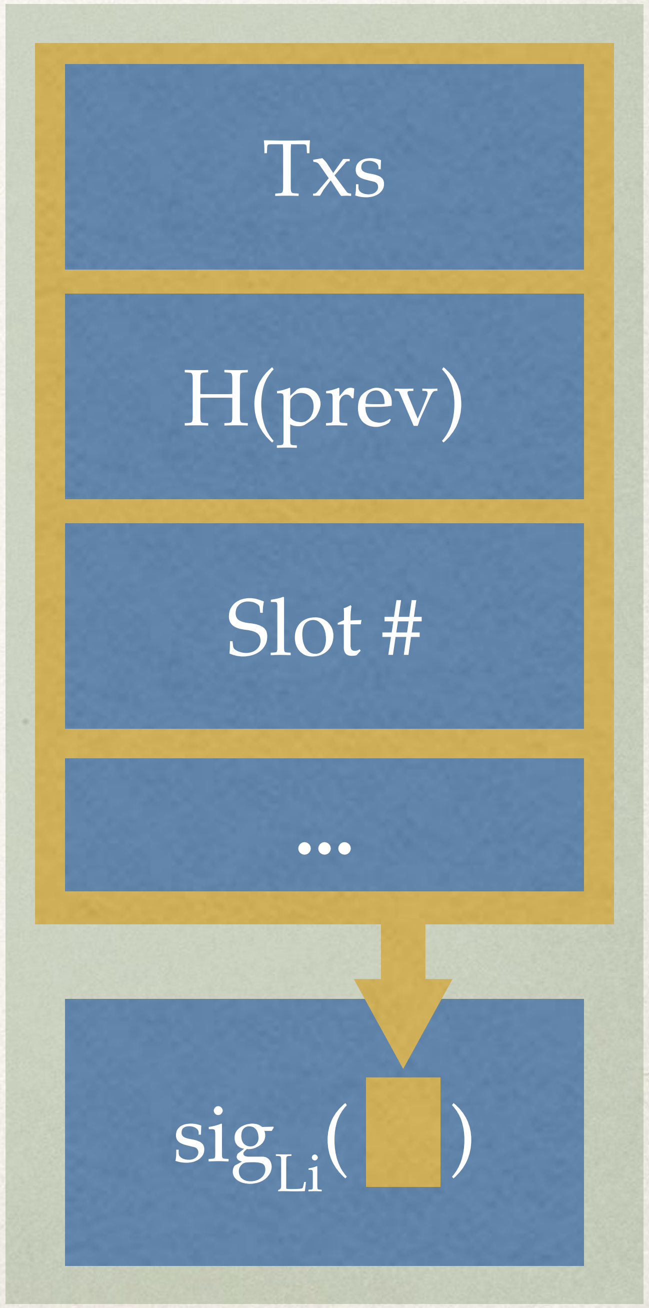
- *pk* remains the same
- *sk* updated in every step, old *sk* erased
- impossible to forge old signatures with new keys

# Block signing: Key-evolving signatures

**KES** are signature schemes, where:

- *pk* remains the same
- *sk* updated in every step, old *sk* erased
- impossible to forge old signatures with new keys

- used for signing blocks
- helps achieve **adaptive security**
- UC-functionality + realization



# Validity of a chain

A *valid blockchain* in single-epoch setting:


- increasing slot numbers
- each block contains:
  - correct VRF-pair proving eligibility
  - correct VRF-pair for randomness derivation
  - KES-signature by eligible leader

# The Protocol (single epoch)

- For each slot:
  - Collect all transactions.
  - Collect all broadcast blockchains. Cull according to validity; maintain the longest one **C**.
  - If *leader*, add a new block in this slot with all transactions (consistent with **C**) to the end of **C**. Sign it and broadcast.

# Ouroboros Praos:
## Security Analysis

# Proven Guarantees

- ✓ **Common Prefix** ($k$): Any 2 chains possessed by 2 honest parties: one is a prefix of the other except for at most $k$ last blocks.
- ✓ **Chain Growth** ($s$,$\boldsymbol{\tau}$): Any chain possessed by an honest party has at least $\boldsymbol{\tau}s$ blocks over any sequence of $s$ slots.
- ✓ **Chain Quality** ($k$): Any chain possessed by an honest party contains an honest block among last $k$ blocks.

# Proven Guarantees

- ✓ **Common Prefix** ($k$): Any 2 chains possessed by 2 honest parties: one is a prefix of the other except for at most $k$ last blocks.
- ✓ **Chain Growth** ($s$,$\boldsymbol{\tau}$): Any chain possessed by an honest party has at least $\boldsymbol{\tau}s$ blocks over any sequence of $s$ slots.
- ✓ **Chain Quality** ($k$): Any chain possessed by an honest party contains an honest block among last $k$ blocks.

These are known to imply what we want:

- ✓ **Persistence**
- ✓ **Liveness**

# Proof Outline

1. **CP, CG, CQ**
   - single-epoch setting, static corruption

# Proof Outline

1. **CP, CG, CQ**
   - single-epoch setting, static corruption
2. **Adaptive adversaries**
   - dominated by a “greedy” static adversary

# Proof Outline

1. **CP, CG, CQ**
   - single-epoch setting, static corruption
2. **Adaptive adversaries**
   - dominated by a “greedy” static adversary
3. **Lifting to multiple epochs**
   - security of the (stake dist., randomness)-update mechanism

# 1. Single-epoch, static CP, CG, CQ

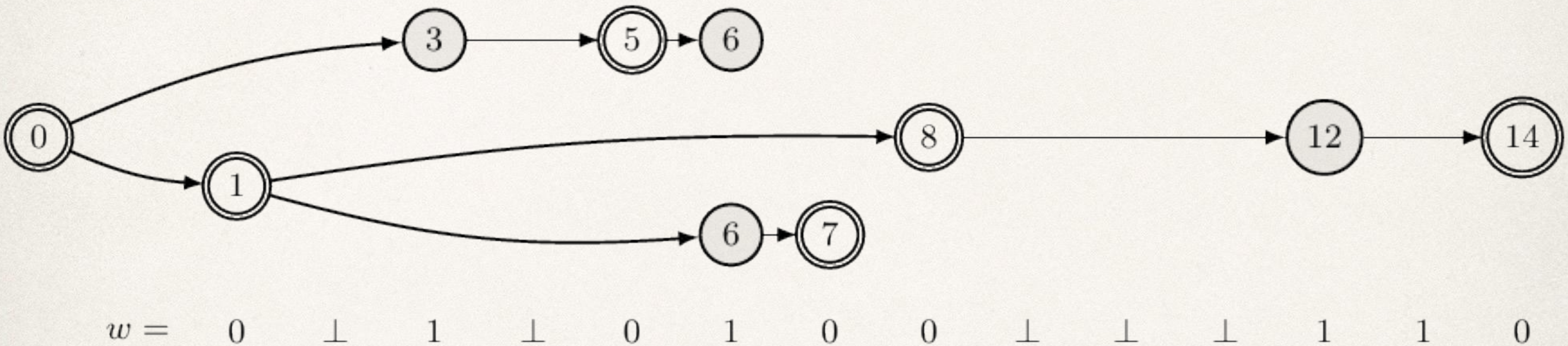
*Unlike a bitcoin adversary,* our adversary:

- knows which slots **he** controls ahead of time
- can generate multiple blocks per slot for free

**This additional power can be contained.**

- extension of a blockchain calculus from [KRDO17]
- here: only CP

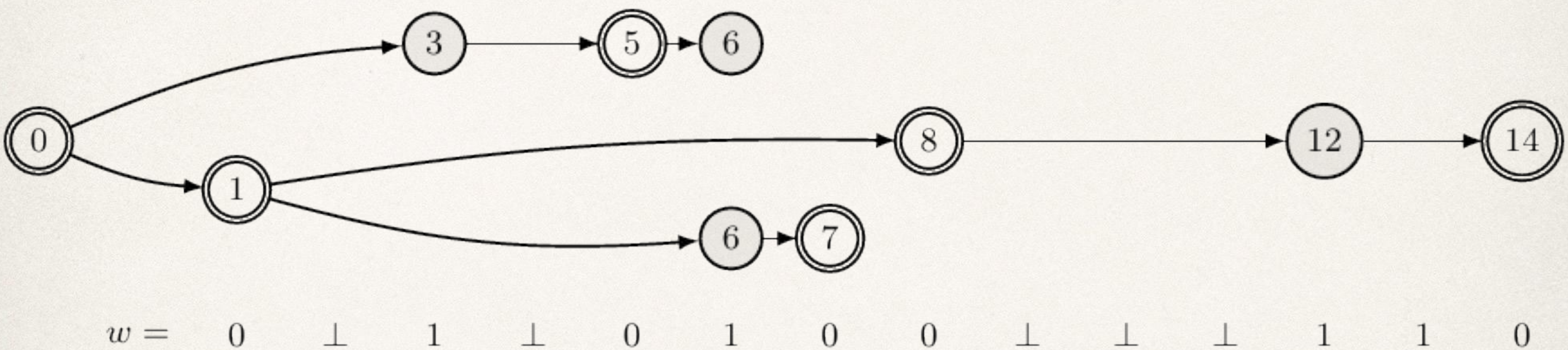
# Characteristic strings and forks




In a fixed execution...

- ***characteristic string:*** describes the leader assignment
- ***fork:*** tree that captures all constructed chains
- one ***char. string*** admits many ***forks***
- some ***forks*** are bad (create large CP-violation)

# Characteristic strings and forks




In the random experiment...

- symbols of ***char. string*** are i.i.d.
- **Goal:** w.h.p. we get a ***char. string*** that admits no bad ***forks***

# Reduction to synchronous case

**Synchronous case [KRDO17]**

- synchronous forks (special case)
- no empty slots (no $\perp$)

# Reduction to synchronous case

**Synchronous case [KRDO17]**

- synchronous forks (special case)
- no empty slots (no $\perp$)

**Reduction mapping** $\rho_{\Delta}(w)\colon \{0,1,\perp\}^* \to \{0,1\}^*$

- results in an "almost" binomial distribution
- preserves CP-violations!

# Bounding synchronous CP

**Theorem from [KRDO17,RMKQ17]:**
Draw $w=w_1 \ldots w_n$ from the binomial distribution with parameter $(1\text{-}\varepsilon)/2$. Then

$$\Pr[k\text{-CP violation}] \leq ne^{-\Omega(k)}.$$

**Proof:**

- martingale argument

# 2. Adaptive adversaries

# 2. Adaptive adversaries

- consider leadership elections for **individual coins**
  - equivalent thanks to “independent aggregation”

# 2. Adaptive adversaries

- consider leadership elections for **individual coins**
    - equivalent thanks to “independent aggregation”
- let the adversary corrupt individual coins
    - more powerful than before

# 2. Adaptive adversaries

- consider leadership elections for **individual coins**
    - equivalent thanks to “independent aggregation”
- let the adversary corrupt individual coins
    - more powerful than before
- yet-uncorrupted coins are indistinguishable
    - thanks to key-evolving signatures

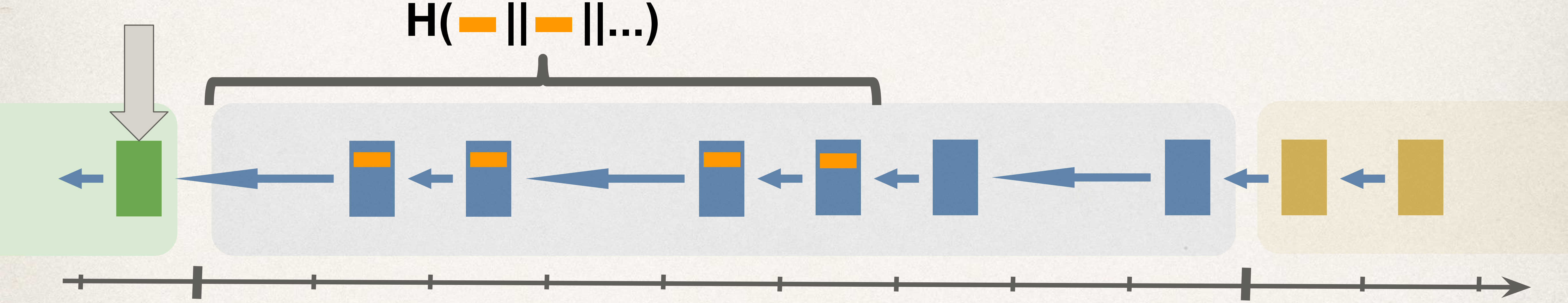
# 2. Adaptive adversaries

- consider leadership elections for **individual coins**
    - equivalent thanks to “independent aggregation”
- let the adversary corrupt individual coins
    - more powerful than before
- yet-uncorrupted coins are indistinguishable
    - thanks to key-evolving signatures
- “greedy” static adversary dominates any adaptive one

# 3. Lifting to multiple epochs

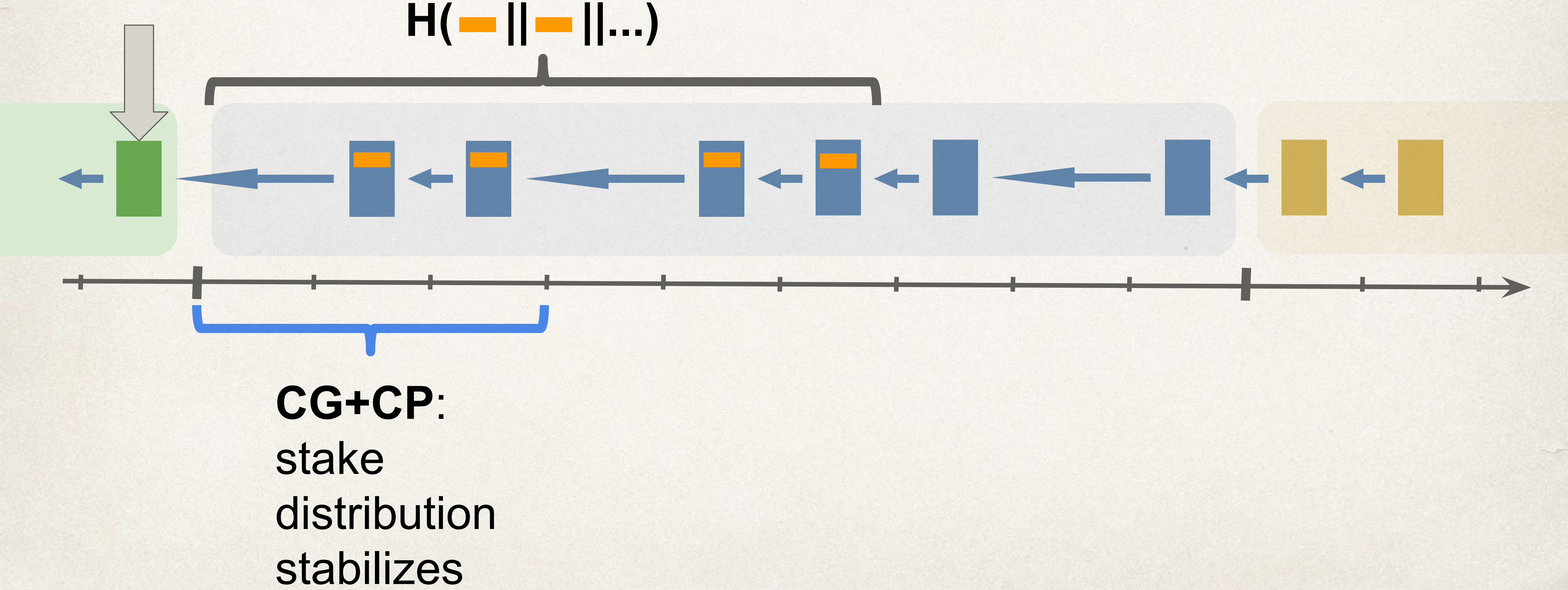
# 3. Lifting to multiple epochs

- **stake distribution**: snapshot from the last block 2 epochs ago
- **randomness**: hash of VRF-values in first ⅔ of previous epoch



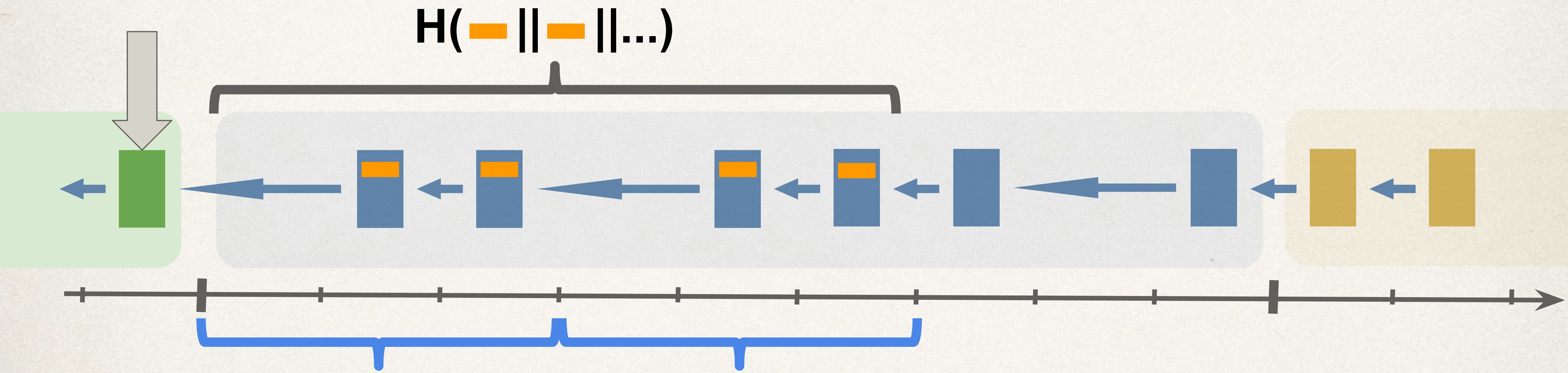
# 3. Lifting to multiple epochs

- **stake distribution**: snapshot from the last block 2 epochs ago
- **randomness**: hash of VRF-values in first ⅔ of previous epoch



# 3. Lifting to multiple epochs

- **stake distribution**: snapshot from the last block 2 epochs ago
- **randomness**: hash of VRF-values in first ⅔ of previous epoch

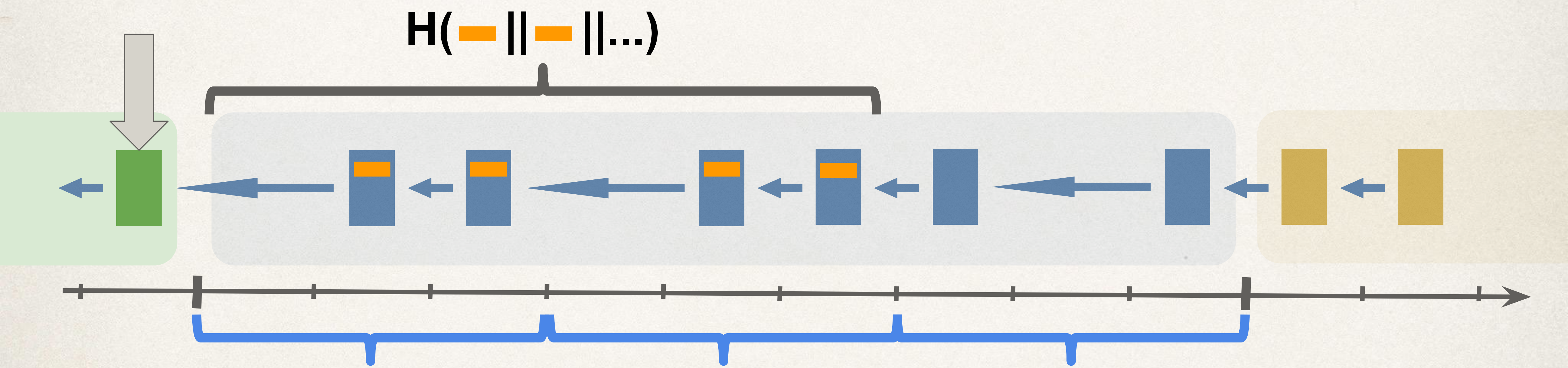



**CG+CP**:
stake
distribution
stabilizes

**CG+CQ**:
honest block
affects
randomness

# 3. Lifting to multiple epochs

- **stake distribution**: snapshot from the last block 2 epochs ago
- **randomness**: hash of VRF-values in first ⅔ of previous epoch



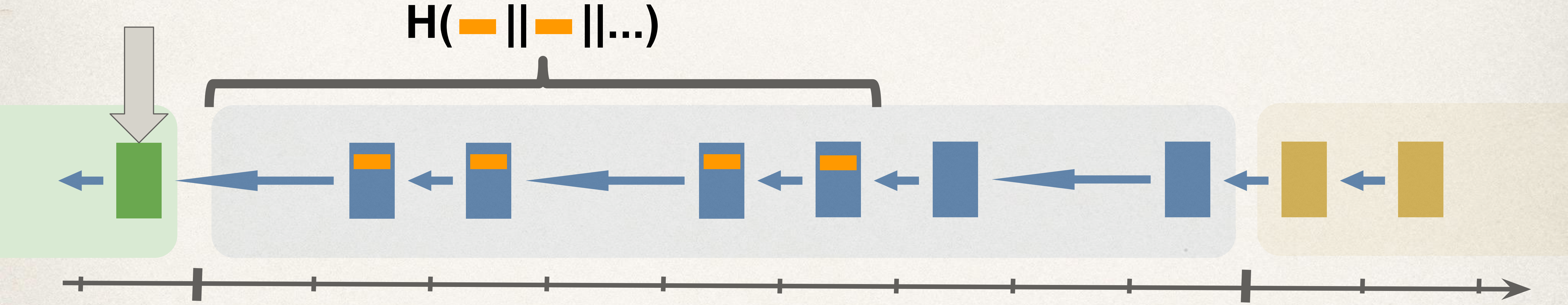

**CG+CP**: stake distribution stabilizes

**CG+CQ**: honest block affects randomness

**CG+CP**: randomness stabilizes

# 3. Lifting to multiple epochs

- **stake distribution**: snapshot from the last block 2 epochs ago
- **randomness**: hash of VRF-values in first ⅔ of previous epoch




- some "grinding" still possible
  - small number of "resamplings"
  - insufficient to boost exponentially small error probabilities

# Follow-up: Ouroboros Genesis

Improved Ouroboros Praos that:

- provides **bootstrapping from genesis block**
- **UC-realizes the Ledger functionality** from [BMTZ17]
- achieves security with **dynamic availability**

[Badertscher, Gaži, Kiayias, Russell, Zikas'18]

# Thank you for your attention!

- **Ouroboros:** [Crypto'17]

  https://eprint.iacr.org/2016/889

- **Ouroboros Praos:** [Eurocrypt'18]

  https://eprint.iacr.org/2017/573

- **Ouroboros Genesis:**

  https://eprint.iacr.org/2018/378