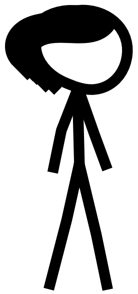
FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

Søren Eller Thomsen, *Aarhus University*
Bas Spitters, *Aarhus University*

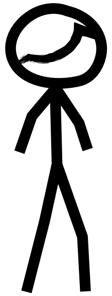
FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

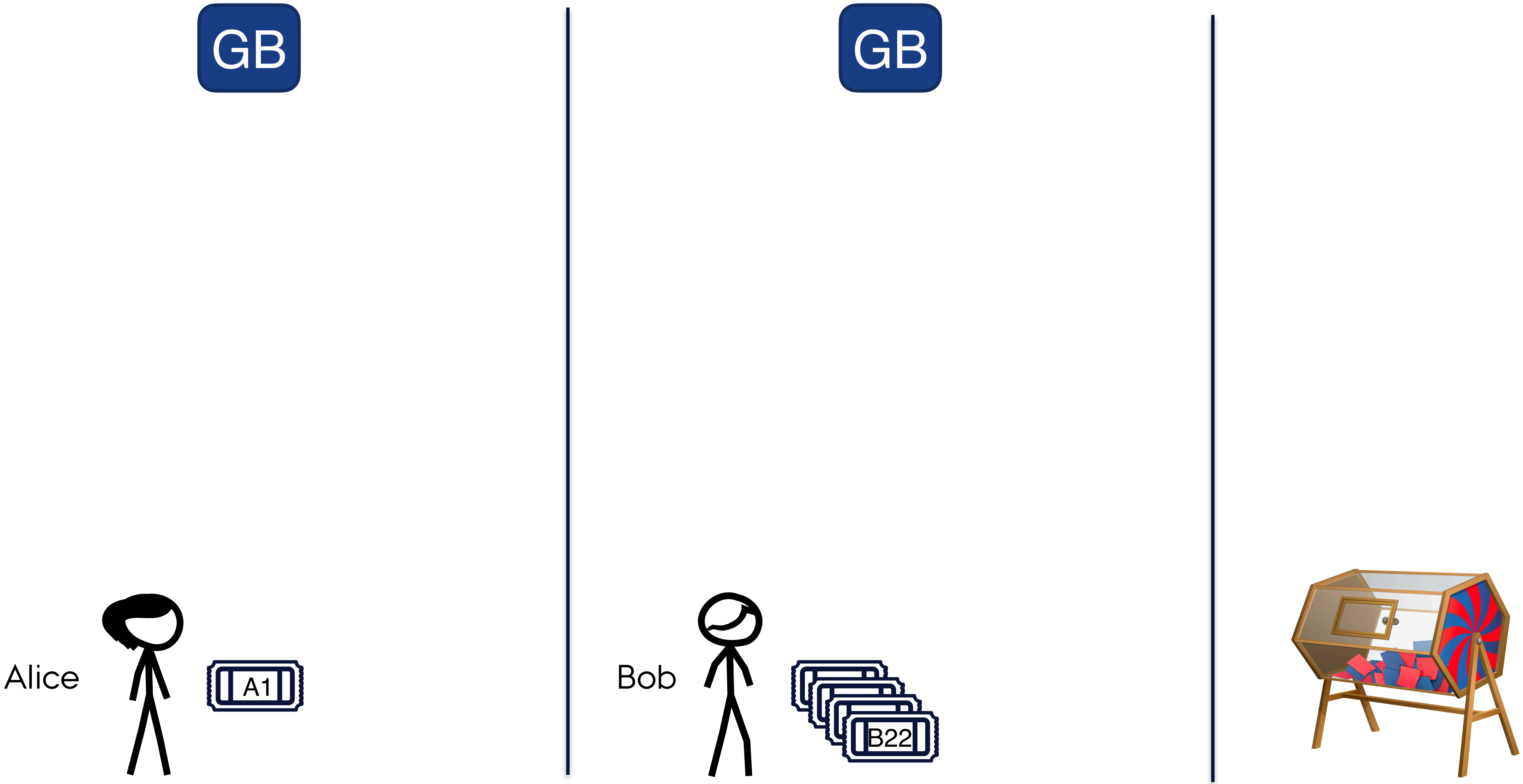
Søren Eller Thomsen, *Aarhus University*
Bas Spitters, *Aarhus University*

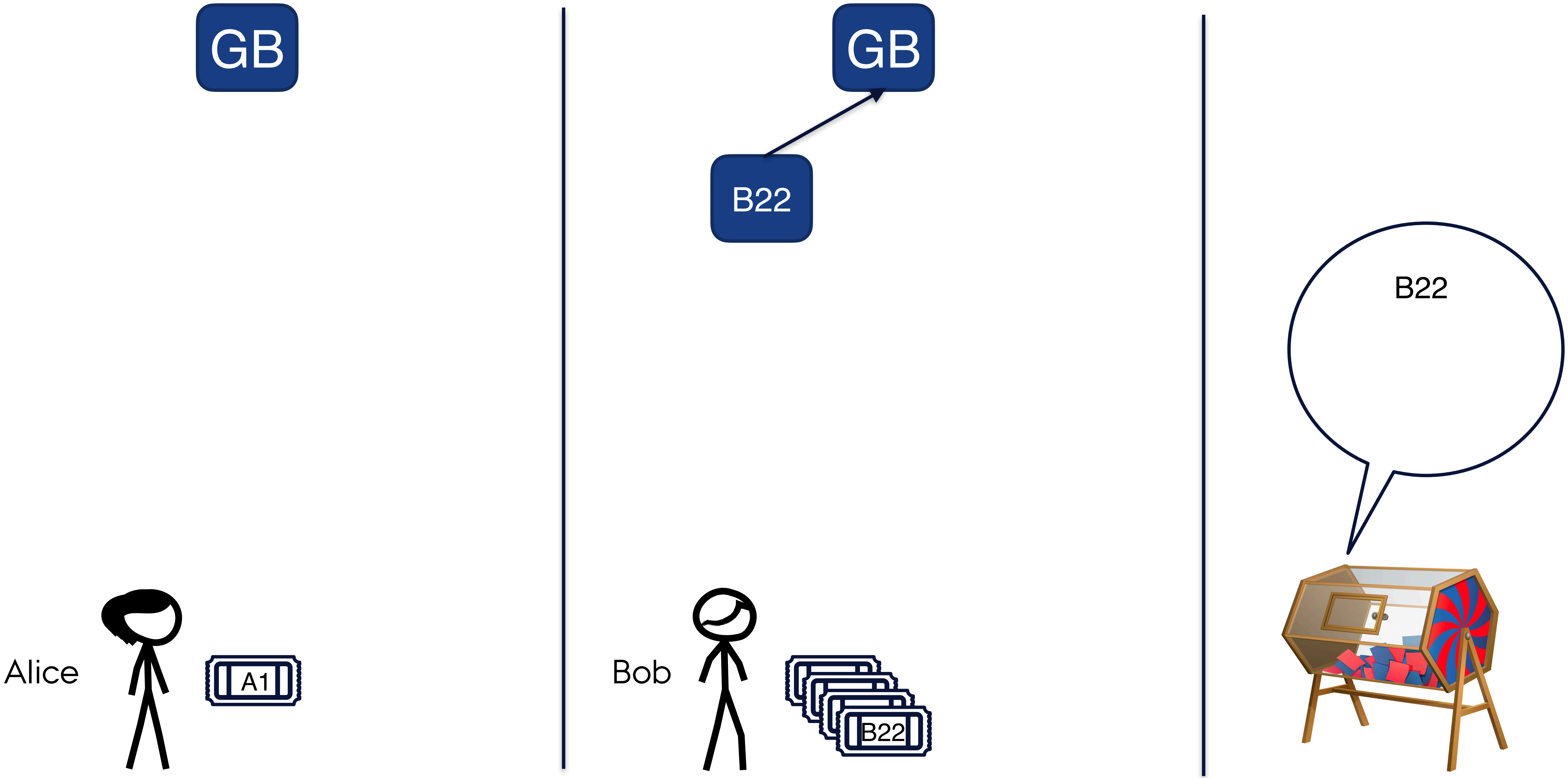
Alice

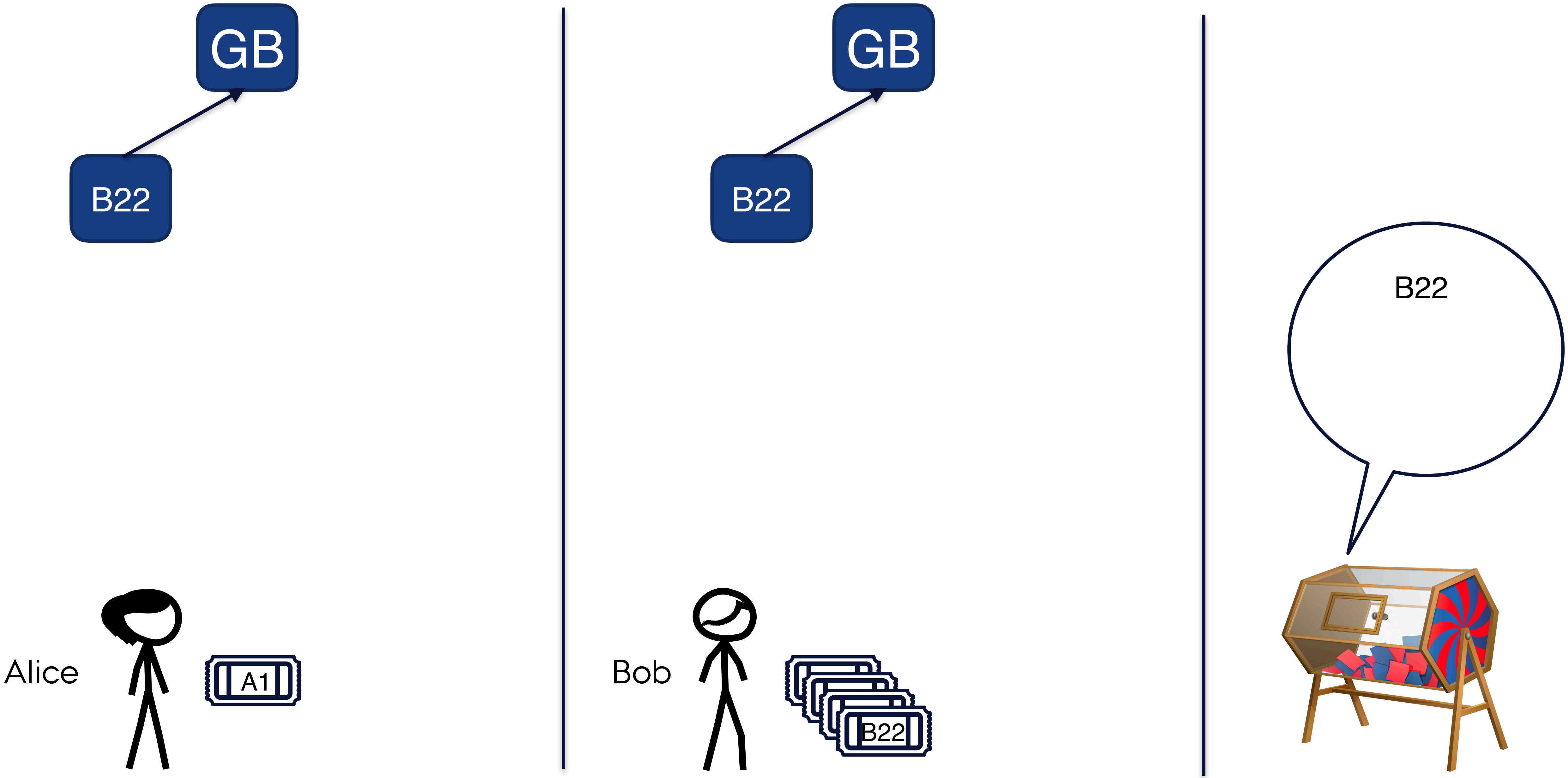


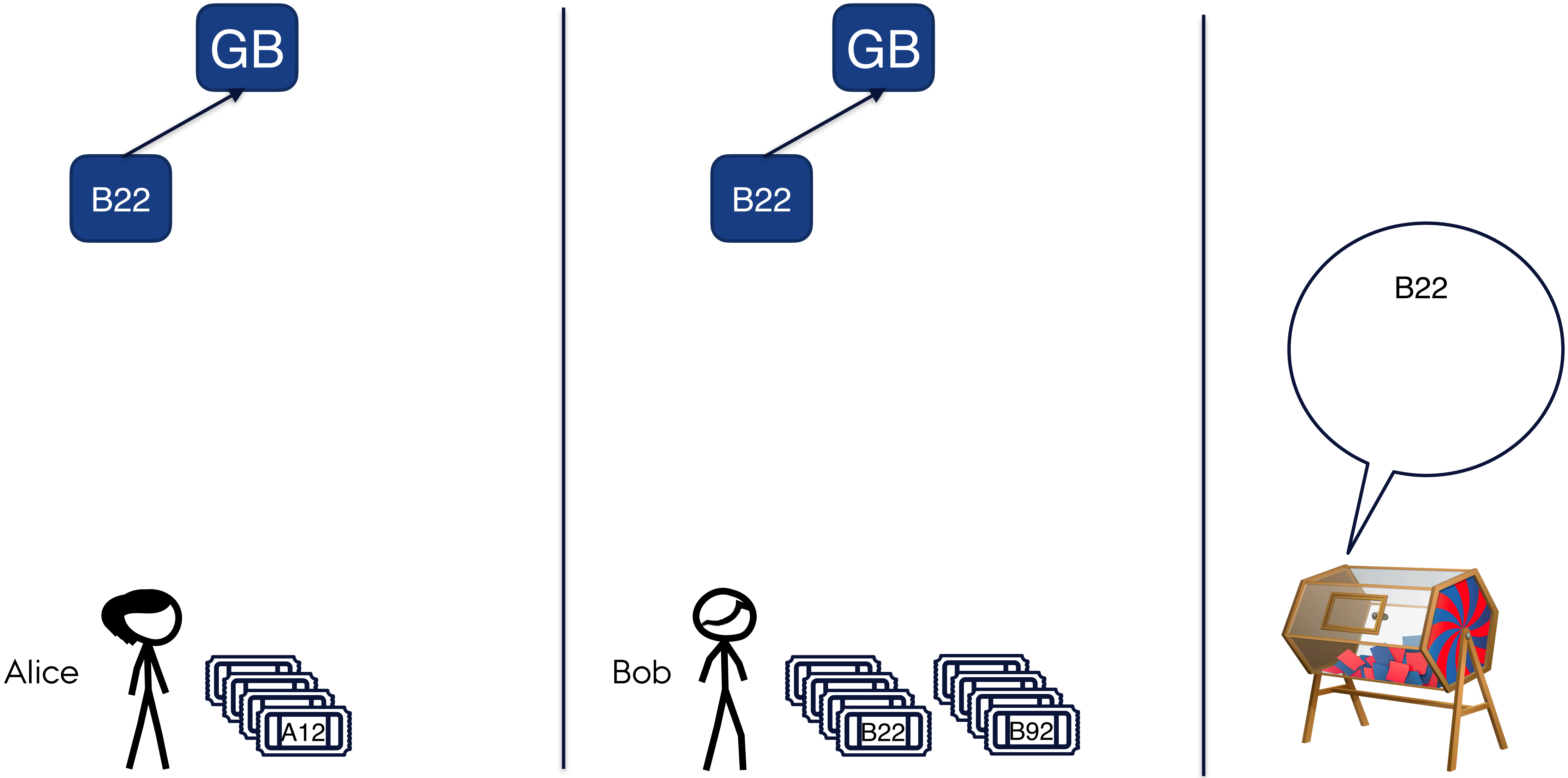
Bob

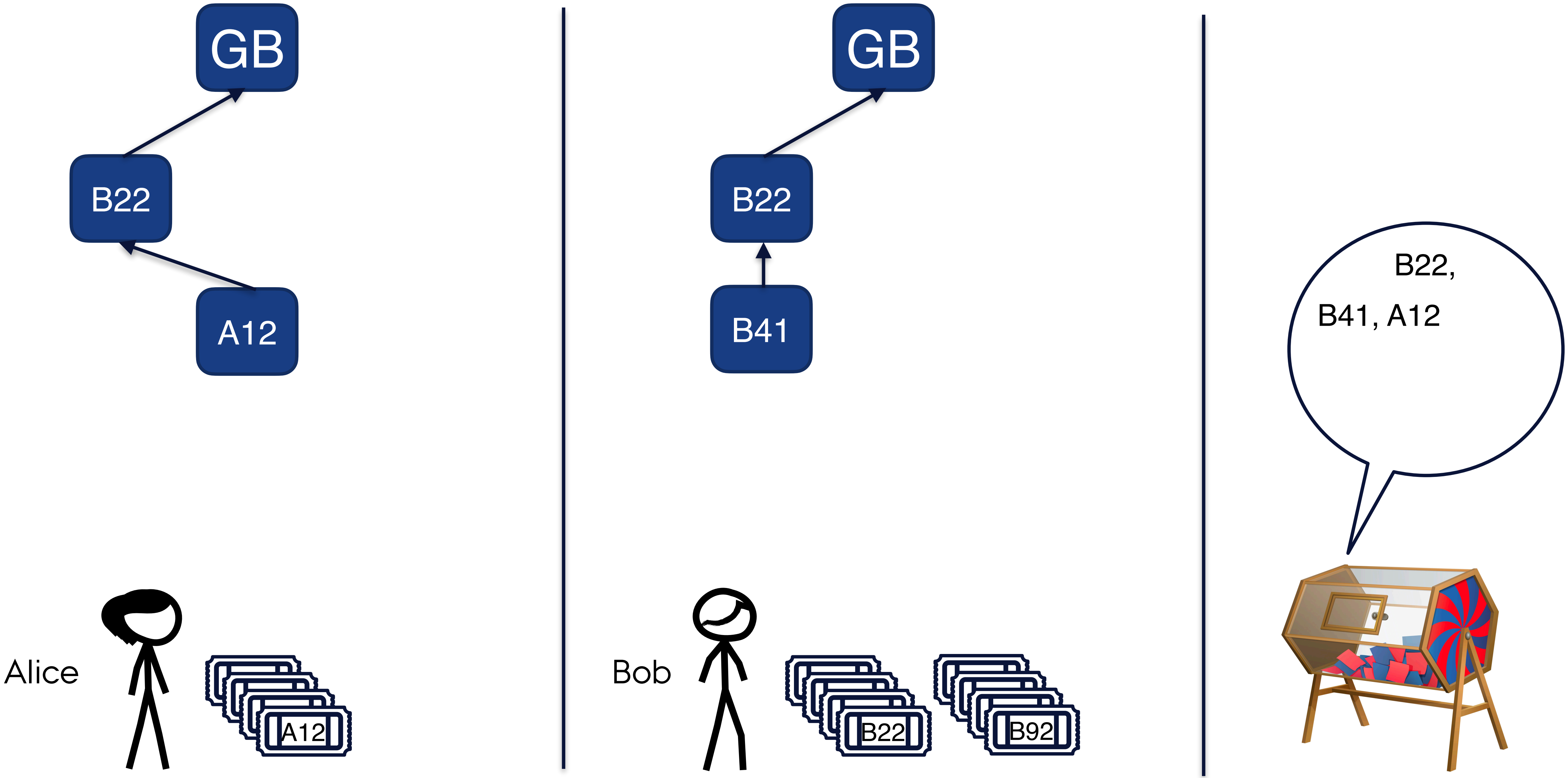


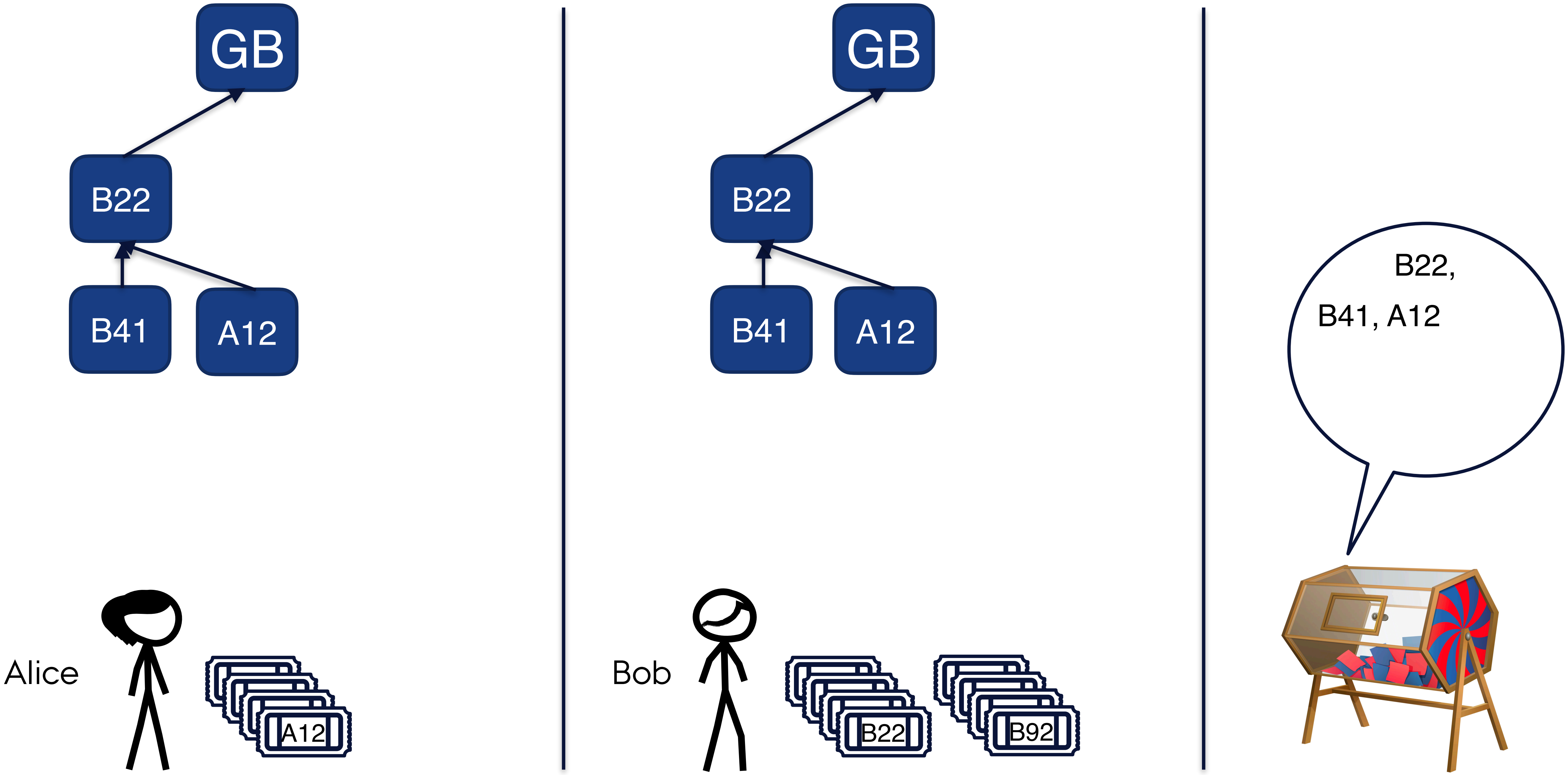

GB
GB
Alice
A1
Bob
B22

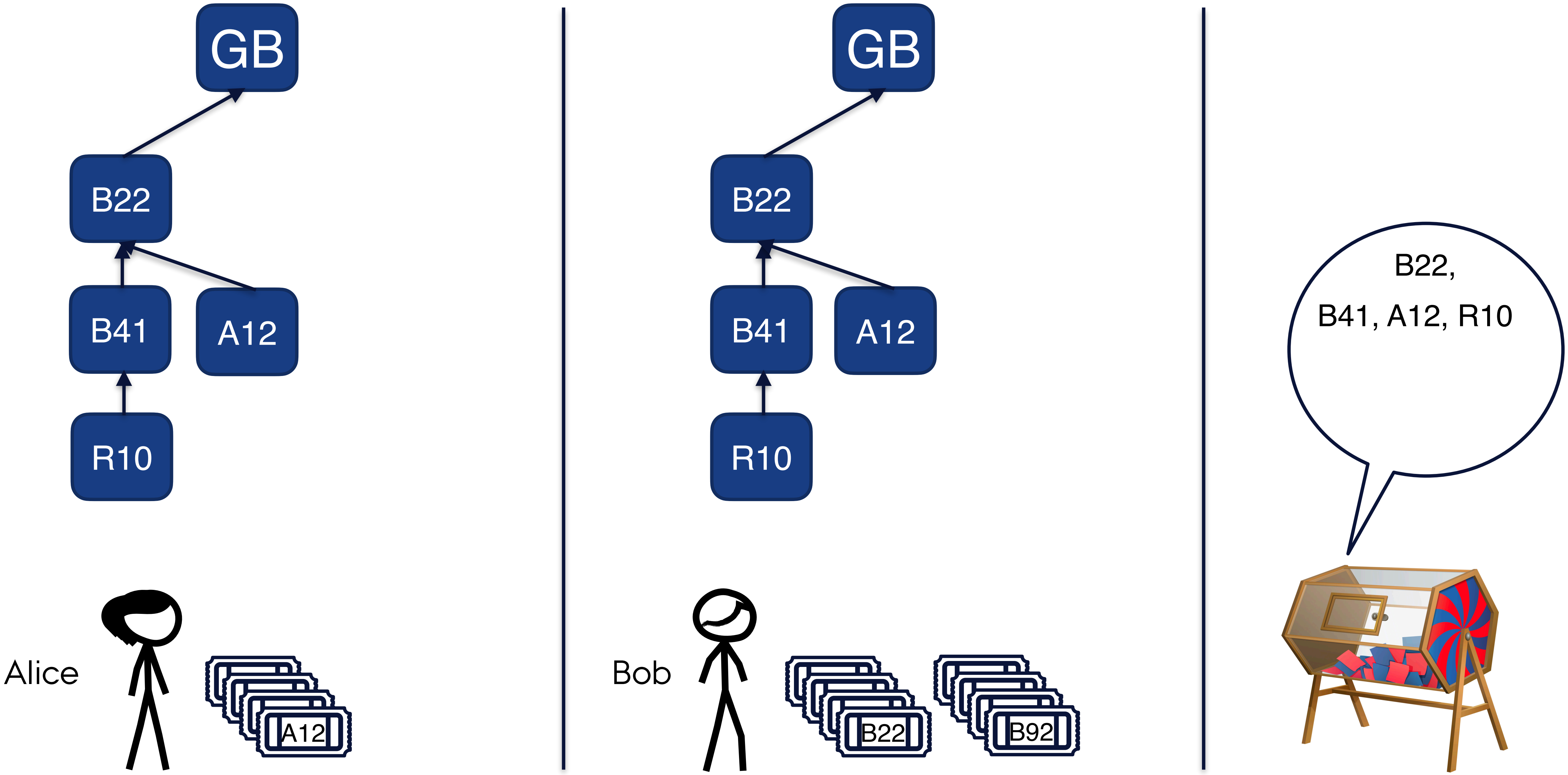

GB
Alice
A1
GB
B22
Bob
B22
B22


GB
B22
Alice
A1
GB
B22
Bob
B22
B22


GB
B22
Alice
A12
GB
B22
Bob
B22
B92
B22


GB
B22
A12
Alice
A12
GB
B22
B41
Bob
B22
B92
B22,
B41, A12


GB
B22
B41
A12
Alice
A12
GB
B22
B41
A12
Bob
B22
B92
B22,
B41, A12


GB
B22
B41
A12
R10
Alice
A12
GB
B22
B41
A12
R10
Bob
B22
B92
B22,
B41, A12, R10

FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

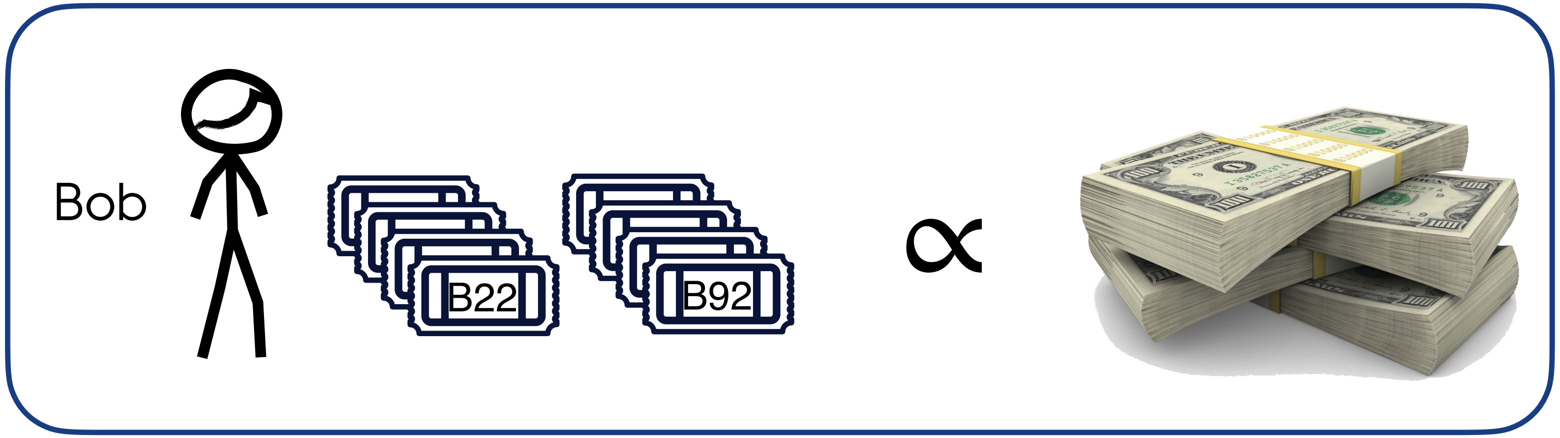
Søren Eller Thomsen, *Aarhus University*
Bas Spitters, *Aarhus University*

FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

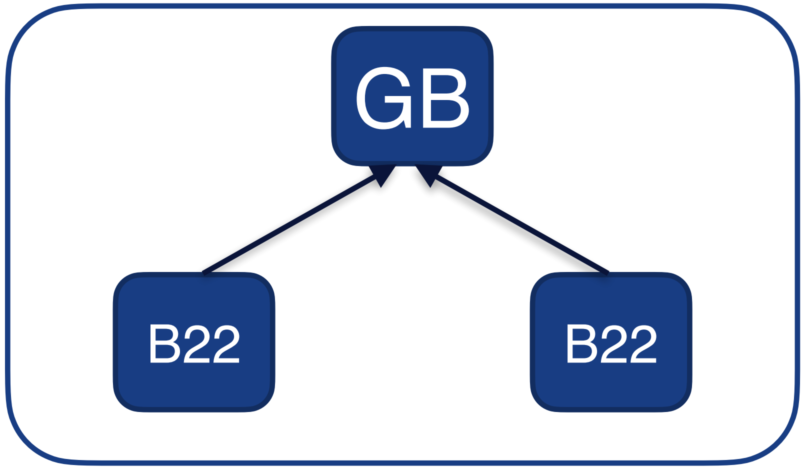
Søren Eller Thomsen, *Aarhus University*
Bas Spitters, *Aarhus University*

PROOF OF STAKE

- Number of tickets are proportional to the amount of stake each player owns.



- A winning ticket can (but shouldn't!) be used to create **multiple different blocks**.

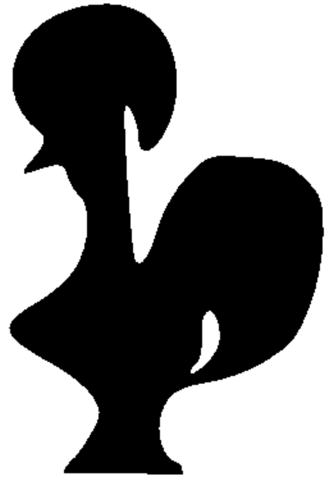


FORMALIZING NAKAMOTO-STYLE PROOF OF STAKE

Søren Eller Thomsen, *Aarhus University*
Bas Spitters, *Aarhus University*

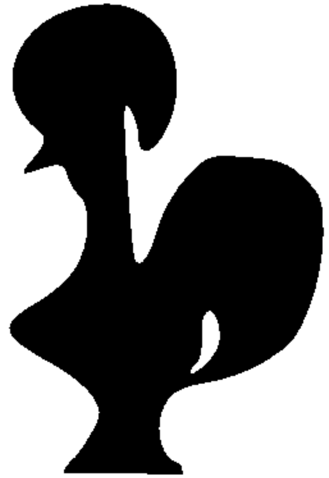
“FORMALIZING” (AKA. CONTRIBUTIONS)

1. We define formal semantics of executions of an abstract PoS NSB in Coq.
2. We give the *first* mechanized proof of the core combinatorics of this protocol. Specifically we prove:
    a) Chain Growth.
    b) Chain Quality.
    c) Common Prefix (n > 3t).
3. We develop a new methodology for verifying protocols by their abstract functional interfaces.



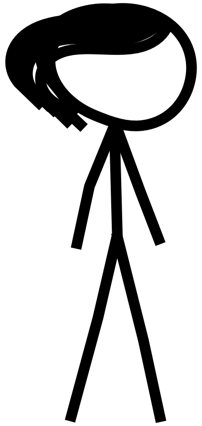
“FORMALIZING” (AKA. CONTRIBUTIONS)

1. We define formal semantics of executions of an abstract PoS NSB in Coq.
2. We give the *first* mechanized proof of the core combinatorics of this protocol. Specifically we prove:
   a) Chain Growth.
   b) Chain Quality.
   c) Common Prefix (n > 3t).
3. We develop a new methodology for verifying protocols by their abstract functional interfaces.



MODELLING - OVERVIEW

HONEST PARTIES

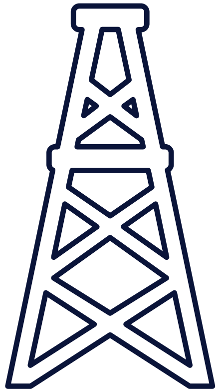


- Local State
- Delivery
- Baking

GLOBAL STATE

- Set of parties and states
- State for adversary
- State for network

NETWORK



- Functions on a global state

ADVERSARY

- Opaque adversarial stateful function

MODELLING - HONEST PARTIES

The state monad.

```
Definition honest_bake : Slot -> Transactions -> State LocalState Messages := …
```

```
Definition honest_rcv : Slot -> Messages -> State LocalState unit := …
```

```
Record LocalState :=
mkLocalState
  { tT : treeType
  ; pk : Party
  ; tree : tT }.
```

MODELLING - HONEST PARTIES

The state monad.

```
Definition honest_bake : Slot -> Transactions -> State LocalState Messages := …
```

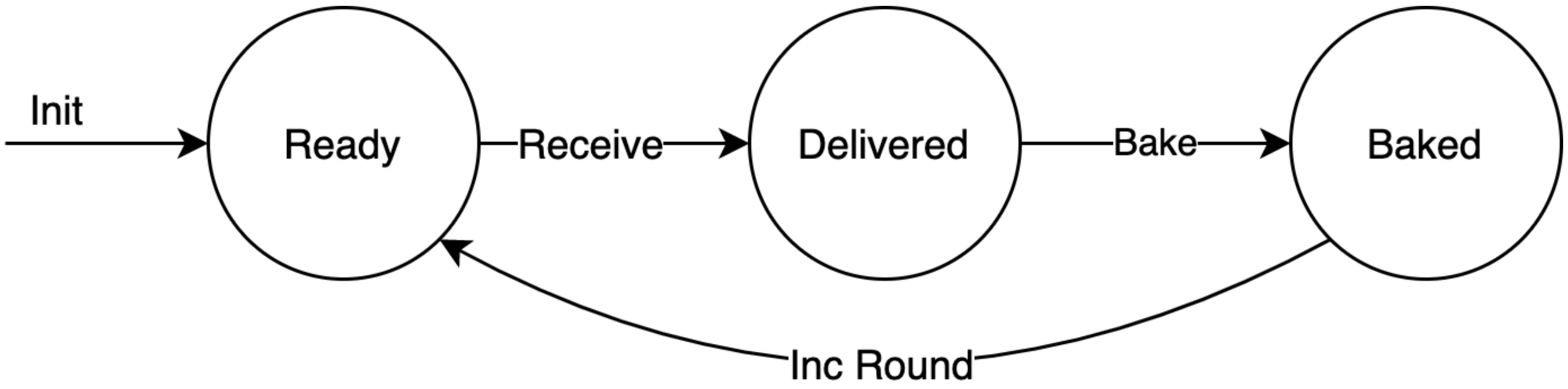
```
Definition honest_rcv : Slot -> Messages -> State LocalState unit := …
```

```
Record LocalState :=
mkLocalState
  { tT : treeType
  ; pk : Party
  ; tree : tT }.
```

```
Record mixin_of T := Mixin
         { extendTree : T -> Block -> T
         ; bestChain : Slot -> T -> Chain
         ; allBlocks: T -> BlockPool
         ; tree0 : T

         ; _ : allBlocks tree0 =i [:: GenesisBlock]
         ; _ : forall t b, allBlocks (extendTree t b) =i allBlocks t ++ [:: b]
         ; _ : forall t s, valid_chain (bestChain s t)
         ; _ : forall c s t, valid_chain c -> {subset c <= [seq b <- allBlocks t | sl b <= s]} -> |c| <=bestChain s t|
         ; _ : forall s t, {subset (bestChain s t) <= [seq b <- allBlocks t | sl b <= s]}}.
```

REACHABLE WORLDS



THEOREMS

```
Theorem chain_growth :
  forall w N1 N2,
  N0 ⇓ N1 -> N1 ⇓ N2 ->
  w <= |lucky_slots_worlds N1 N2| ->
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.
```

```
Theorem chain_quality :
  forall N p l b_j b_i c w,
  let bc := bestChain (t_now N) (tree l) in
  let f := [:: b_j] ++ c ++ [:: b_i] in
  N0 ⇓ N ->
  forging_free N ->
  collision_free N ->
  has_state p N l -> is_honest p ->
  fragment f bc ->
  honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
  w <= |honest_blocks f|.
```

```
Theorem common_prefix :
  forall k N1 N2,
  N0 ⇓ N1 ->
  N1 ⇓ N2 ->
  forging_free N2 ->
  collision_free N2 ->
  forall p1 p2 l1 l2,
  let bc1 := bestChain (t_now N1) (tree l1) in
  let bc2 := bestChain (t_now N2) (tree l2) in
  is_honest p1 -> is_honest p2 ->
  has_state p1 N1 l1 -> has_state p2 N2 l2 ->
  prune_time k bc1 ≤ bc2  \/
  exists t1 t2,[/\ t1 <= k
              , t_now N1 <= t2 <= t_now N2
                 & |super_slots_range t1 t2|
                   <= 2 * | adv_slots_range t1 t2| ].
```

THEOREMS

```
Theorem chain_growth :
  forall w N1 N2,
  N0 ⇓ N1 -> N1 ⇓ N2 ->
  w <= |lucky_slots_worlds N1 N2| ->
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.
```

```
Theorem chain_quality :
  forall N p l b_j b_i c w,
  let bc := bestChain (t_now N) (tree l) in
  let f := [:: b_j] ++ c ++ [:: b_i] in
  N0 ⇓ N ->
  forging_free N ->
  collision_free N ->
  has_state p N l -> is_honest p ->
  fragment f bc ->
  honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
  w <= |honest_blocks f|.
```

```
Theorem common_prefix :
  forall k N1 N2,
  N0 ⇓ N1 ->
  N1 ⇓ N2 ->
  forging_free N2 ->
  collision_free N2 ->
  forall p1 p2 l1 l2,
  let bc1 := bestChain (t_now N1) (tree l1) in
  let bc2 := bestChain (t_now N2) (tree l2) in
  is_honest p1 -> is_honest p2 ->
  has_state p1 N1 l1 -> has_state p2 N2 l2 ->
  prune_time k bc1 ≤ bc2  \/
  exists t1 t2,[/\ t1 <= k
               , t_now N1 <= t2 <= t_now N2
                 & |super_slots_range t1 t2|
                   <= 2 * | adv_slots_range t1 t2| ].
```

THEOREMS

```
Theorem chain_growth :
  forall w N1 N2,
  N0 ⇓ N1 -> N1 ⇓ N2 ->
  w <= |lucky_slots_worlds N1 N2| ->
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.
```

```
Theorem chain_quality :
  forall N p l b_j b_i c w,
  let bc := bestChain (t_now N) (tree l) in
  let f := [:: b_j] ++ c ++ [:: b_i] in
  N0 ⇓ N ->
  forging_free N ->
  collision_free N ->
  has_state p N l -> is_honest p ->
  fragment f bc ->
  honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
  w <= |honest_blocks f|.
```

```
Theorem common_prefix :
  forall k N1 N2,
  N0 ⇓ N1 ->
  N1 ⇓ N2 ->
  forging_free N2 ->
  collision_free N2 ->
  forall p1 p2 l1 l2,
  let bc1 := bestChain (t_now N1) (tree l1) in
  let bc2 := bestChain (t_now N2) (tree l2) in
  is_honest p1 -> is_honest p2 ->
  has_state p1 N1 l1 -> has_state p2 N2 l2 ->
  prune_time k bc1 ≤ bc2  \/
  exists t1 t2,[/\ t1 <= k
               , t_now N1 <= t2 <= t_now N2
                 & |super_slots_range t1 t2|
                   <= 2 * | adv_slots_range t1 t2| ].
```

THEOREMS

```
Theorem chain_growth :
  forall w N1 N2,
  N0 ⇓ N1 -> N1 ⇓ N2 ->
  w <= |lucky_slots_worlds N1 N2| ->
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.
```

```
Theorem chain_quality :
  forall N p l b_j b_i c w,
  let bc := bestChain (t_now N) (tree l) in
  let f := [:: b_j] ++ c ++ [:: b_i] in
  N0 ⇓ N ->
  forging_free N ->
  collision_free N ->
  has_state p N l -> is_honest p ->
  fragment f bc ->
  honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
  w <= |honest_blocks f|.
```

```
Theorem common_prefix :
  forall k N1 N2,
  N0 ⇓ N1 ->
  N1 ⇓ N2 ->
  forging_free N2 ->
  collision_free N2 ->
  forall p1 p2 l1 l2,
  let bc1 := bestChain (t_now N1) (tree l1) in
  let bc2 := bestChain (t_now N2) (tree l2) in
  is_honest p1 -> is_honest p2 ->
  has_state p1 N1 l1 -> has_state p2 N2 l2 ->
  prune_time k bc1 ≤ bc2  \/
  exists t1 t2,[/\ t1 <= k
              , t_now N1 <= t2 <= t_now N2
                & |super_slots_range t1 t2|
                  <= 2 * | adv_slots_range t1 t2| ].
```

THEOREMS

```
Theorem chain_growth :
  forall w N1 N2,
  N0 ⇓ N1 -> N1 ⇓ N2 ->
  w <= |lucky_slots_worlds N1 N2| ->
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.
```

```
Theorem chain_quality :
  forall N p l b_j b_i c w,
  let bc := bestChain (t_now N) (tree l) in
  let f := [:: b_j] ++ c ++ [:: b_i] in
  N0 ⇓ N ->
  forging_free N ->
  collision_free N ->
  has_state p N l -> is_honest p ->
  fragment f bc ->
  honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
  w <= |honest_blocks f|.
```

```
Theorem common_prefix :
  forall k N1 N2,
  N0 ⇓ N1 ->
  N1 ⇓ N2 ->
  forging_free N2 ->
  collision_free N2 ->
  forall p1 p2 l1 l2,
  let bc1 := bestChain (t_now N1) (tree l1) in
  let bc2 := bestChain (t_now N2) (tree l2) in
  is_honest p1 -> is_honest p2 ->
  has_state p1 N1 l1 -> has_state p2 N2 l2 ->
  prune_time k bc1 ≤ bc2  \/
  exists t1 t2,[/\ t1 <= k
                , t_now N1 <= t2 <= t_now N2
                  & |super_slots_range t1 t2|
                    <= 2 * | adv_slots_range t1 t2| ].
```

THEOREMS

Condition on abstract lottery!

```
Theorem chain_growth :
  forall w N1 N2,
  N0 ⇓ N1 -> N1 ⇓ N2 ->
  w <= |lucky_slots_worlds N1 N2| ->
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.
```

```
Theorem chain_quality :
  forall N p l b_j b_i c w,
  let bc := bestChain (t_now N) (tree l) in
  let f := [:: b_j] ++ c ++ [:: b_i] in
  N0 ⇓ N ->
  forging_free N ->
  collision_free N ->
  has_state p N l -> is_honest p ->
  fragment f bc ->
  honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
  w <= |honest_blocks f|.
```

```
Theorem common_prefix :
  forall k N1 N2,
  N0 ⇓ N1 ->
  N1 ⇓ N2 ->
  forging_free N2 ->
  collision_free N2 ->
  forall p1 p2 l1 l2,
  let bc1 := bestChain (t_now N1) (tree l1) in
  let bc2 := bestChain (t_now N2) (tree l2) in
  is_honest p1 -> is_honest p2 ->
  has_state p1 N1 l1 -> has_state p2 N2 l2 ->
  prune_time k bc1 ≤ bc2  \/
  exists t1 t2,[/\ t1 <= k
               , t_now N1 <= t2 <= t_now N2
                 & |super_slots_range t1 t2|
                   <= 2 * | adv_slots_range t1 t2| ].
```

THEOREMS

Condition on abstract lottery!

```
Theorem chain_growth :
  forall w N1 N2,
  N0 ⇓ N1 -> N1 ⇓ N2 ->
  w <= |lucky_slots_worlds N1 N2| ->
  |honest_tree_chain N1| + w <= |honest_tree_chain N2|.
```

```
Theorem chain_quality :
  forall N p l b_j b_i c w,
  let bc := bestChain (t_now N) (tree l) in
  let f := [:: b_j] ++ c ++ [:: b_i] in
  N0 ⇓ N ->
  forging_free N ->
  collision_free N ->
  has_state p N l -> is_honest p ->
  fragment f bc ->
  honest_advantage_ranges_gt w (sl b_j - sl b_i) ->
  w <= |honest_blocks f|.
```

```
Theorem common_prefix :
  forall k N1 N2,
  N0 ⇓ N1 ->
  N1 ⇓ N2 ->
  forging_free N2 ->
  collision_free N2 ->
  forall p1 p2 l1 l2,
  let bc1 := bestChain (t_now N1) (tree l1) in
  let bc2 := bestChain (t_now N2) (tree l2) in
  is_honest p1 -> is_honest p2 ->
  has_state p1 N1 l1 -> has_state p2 N2 l2 ->
  prune_time k bc1 ≤ bc2  \/
  exists t1 t2,[/\ t1 <= k
               , t_now N1 <= t2 <= t_now N2
                & |super_slots_range t1 t2|
                  <= 2 * | adv_slots_range t1 t2| ].
```

CONCLUSION

- We provide a formal model of the execution semantics of a NSB PoS and are the first to prove both safety and liveness for *any* BFT consensus algorithm.

- Details: https://eprint.iacr.org/2020/917

- Code: https://github.com/AU-COBRA/PoS-NSB

- Contact: sethomsen@cs.au.dk