Technical objectives

Fast interactions

Game should not feel as a “blockchain game”, with constant transaction signing. StarkNet interaction should ideally be limited to no more than starting and finishing games.

Keeping it trustless and P2P

We prefer technical designs where no server middleware is needed and games can be run completely client-to-client via WebRTC, with cryptographic security guarantees.

A road towards “fog of war”

If incomplete information games are not achievable at first with a completely P2P infrastructure, a roadmap should be established from the beginning as to how to eventually get there.

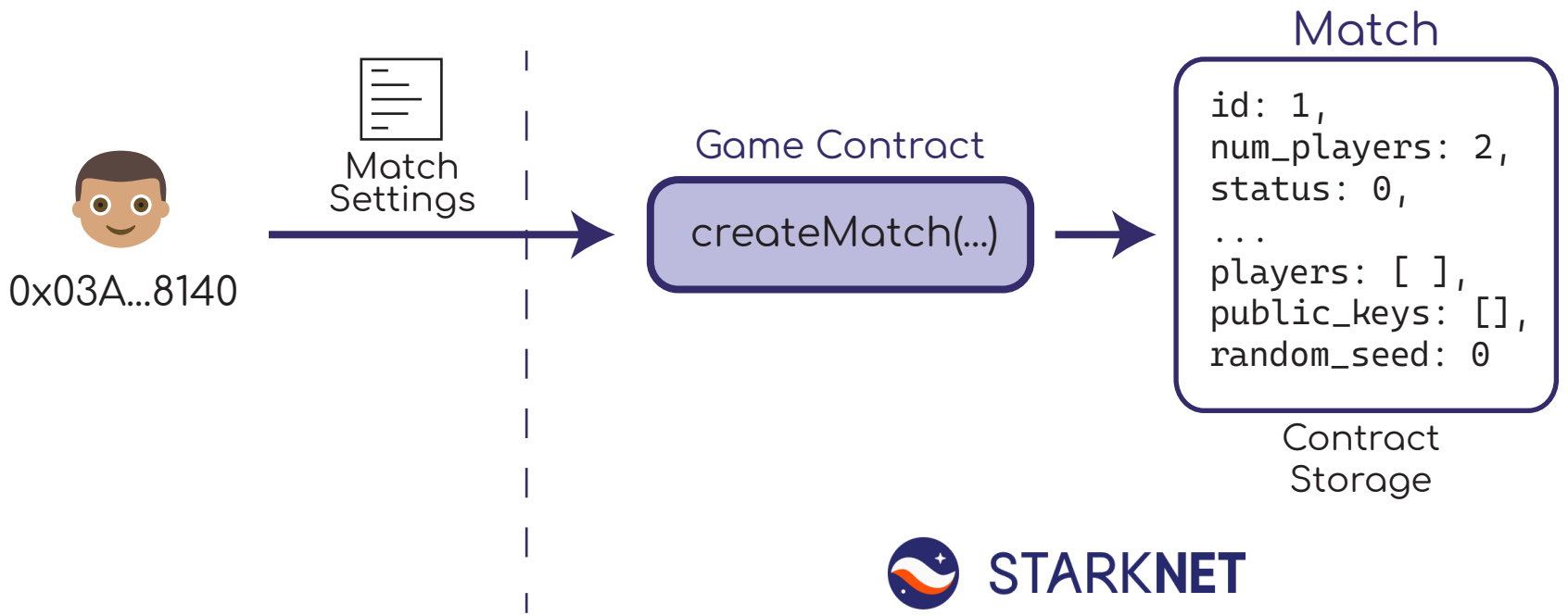
Game + Protocol

The technical implementation should allow for our game to be played, but to also function as a generic P2P game protocol, over which other games could be built in the future.

Game flow

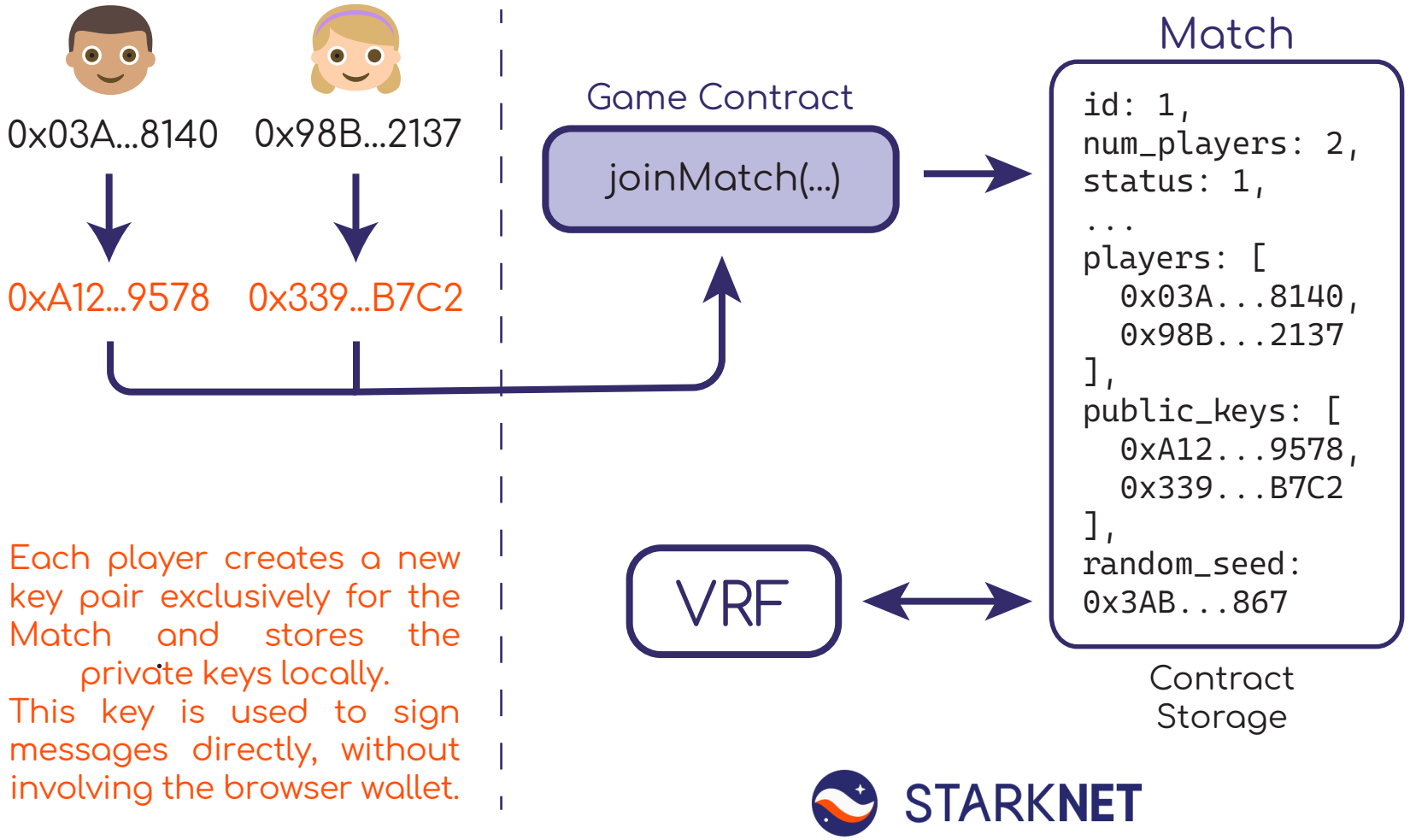
1 Start a Match

A match can be started by anyone via a contract interaction with an external function, by providing some settings (like number of players, expiration, etc). The Match receives a unique ID and is stored on-chain.



2 Players join the Match

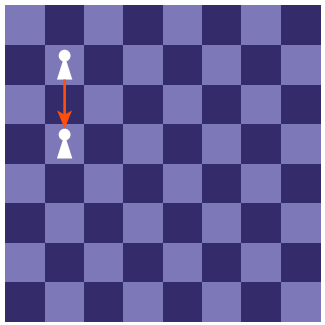
Players can join open Matches by providing new public keys. Once the required number of players has joined, a random number seed is requested to a VRF service. This seed is later used for any necessary randomness.



3 The game begins!

Players interact directly peer-to-peer (P2P) via WebRTC. No middle server is required. Player actions are signed and sent with a reference to the previous game state. All players keep a list of all actions locally. The game state can be verified by a view function in the Game contract.

0x03A...8140



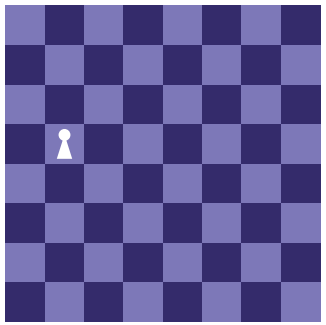
The player action is encoded, signed and sent to other players, with a reference to the previous state.

```
{match: 1, turn: 0,
prev_state: {...}, action: {
unit: 8, start_position: 9,
end_position: 25},
signature: '0x935 ... 0D32'}
```

P2P (WebRTC)

0x98B...2137

Players receives the action, verify the signature and update the game state. This step can be done locally or by calling a view function in the game contract.



New Game State

Game Contract

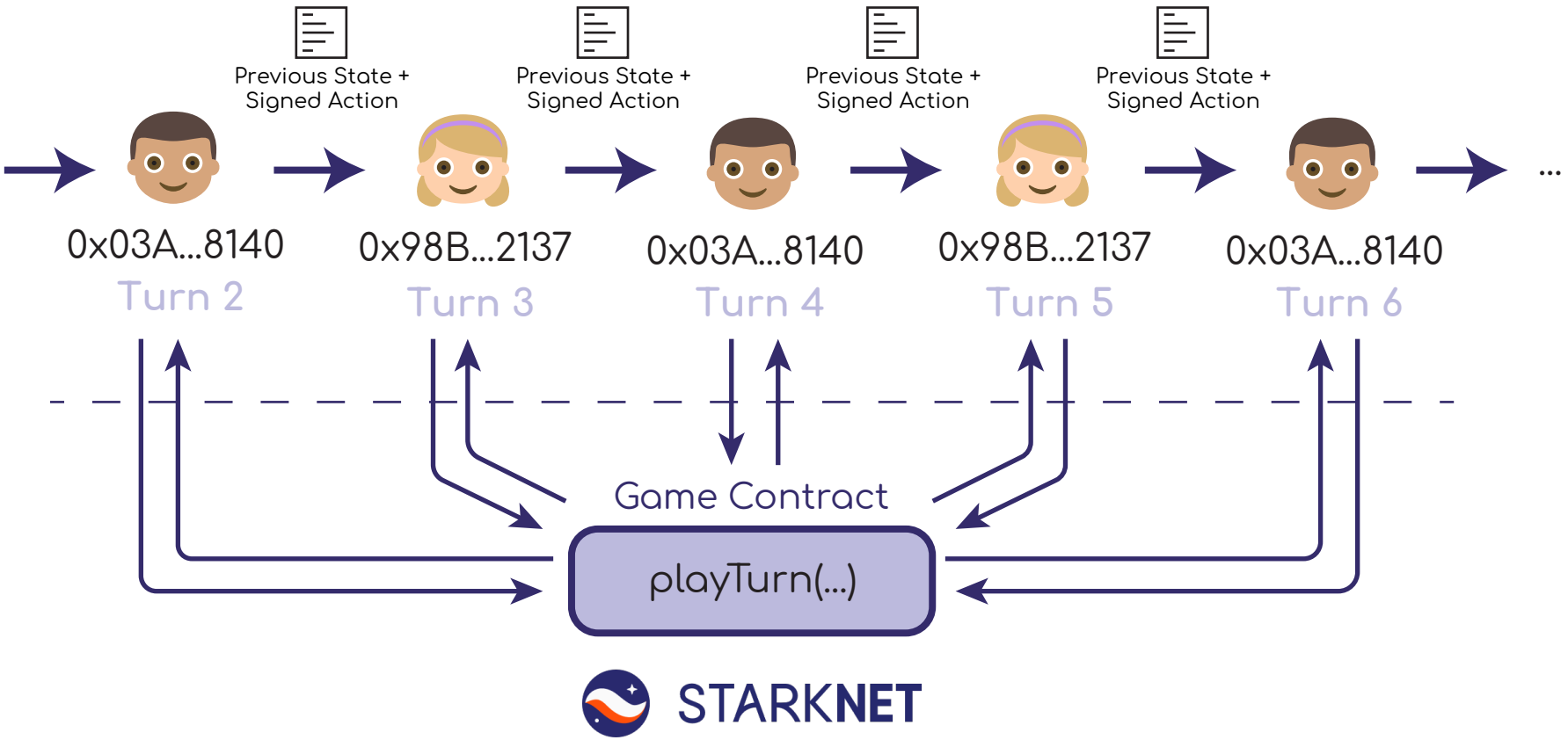
playTurn(...)

The whole game logic exists in this view function, which can verify player's signatures and calculate state transitions.

STARKNET

4 The game continues...

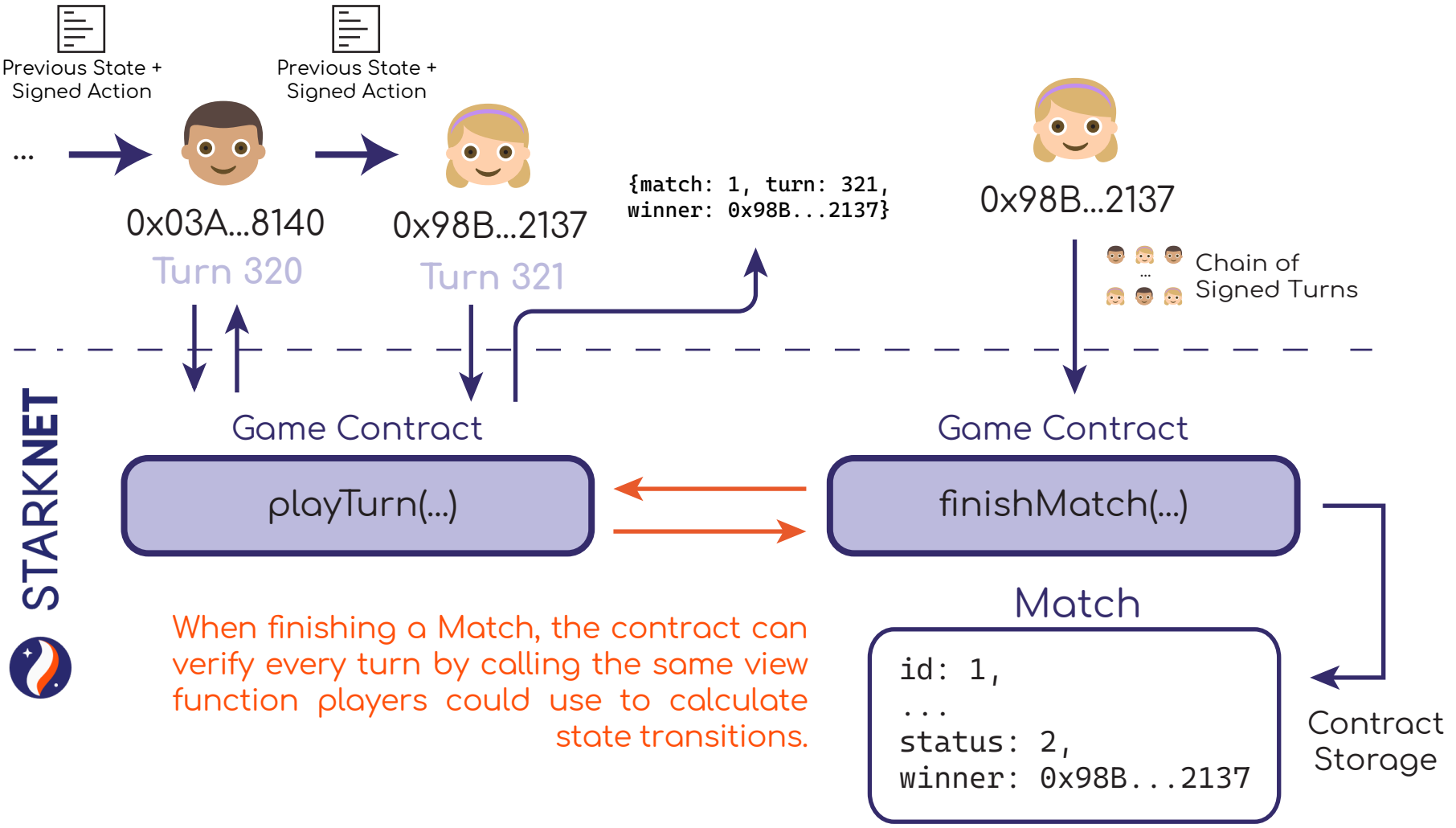
Players continue sending each other their signed actions. This forms a chain of actions, which is stored locally by every player. Each new state is calculated by executing the signed action from the previous state, making it impossible for a player to modify the chain, without breaking it. Updates can be done locally or by calling a view function on the Game contract.



(Note that turns don't need to consist of a single action. A player could send multiple signed actions to other players, before sending a "finish turn" action, which would allow the next player to continue)

5 The game finishes!

Eventually, a player executes an action which ends the Match. Now a whole chain of signed actions exists, going from the initial state to a winner being decided. The winner can finish the Match by sending this data to an external function in the Game Contract, which verifies each turn.



Notes & Remarks

1 No real need for whole chain verification

It should not be neccessary to verify the whole chain of turns, as both players could simply agree on the winner (and the winner actually has proof). If proving the whole chain is too costly, it could be limited to a challenge:

1) Someone claims to be the winner.
2) If other players doesn't challenge the result, the storage is updated.
3) If someone challenges the result, a complete set of turns would need to be submitted and verified on-chain.

2 Timeouts and unfinished games

To avoid players leaving a game unfinished, an on-chain challenge could also be implemented:

1) A player leaves the game or stops playing.
2) Active players can call an external function on the Game Contract, sending the last signed move, and asking the inactive player to continue playing.
3) If the inactive player decides to return, he can sends his next turn to the contract. If not, a state transition is allowed after a time limit, in which the player is considered eliminated.
4) This state-transition is stored on-chain, so that it can be eventually verified.