Oracles and off-chain networks

Getting data in and work out of the blockchain

Leander Jehl

Oracles

Oracle

Access data outside the blockchain

An **Oracle** is a smart contract that publishes information about real world data on the chain.

Oracle

Example: Rain ensurance

- Ensurance contract needs weather data to
  - Pay out policies
  - Determine prices

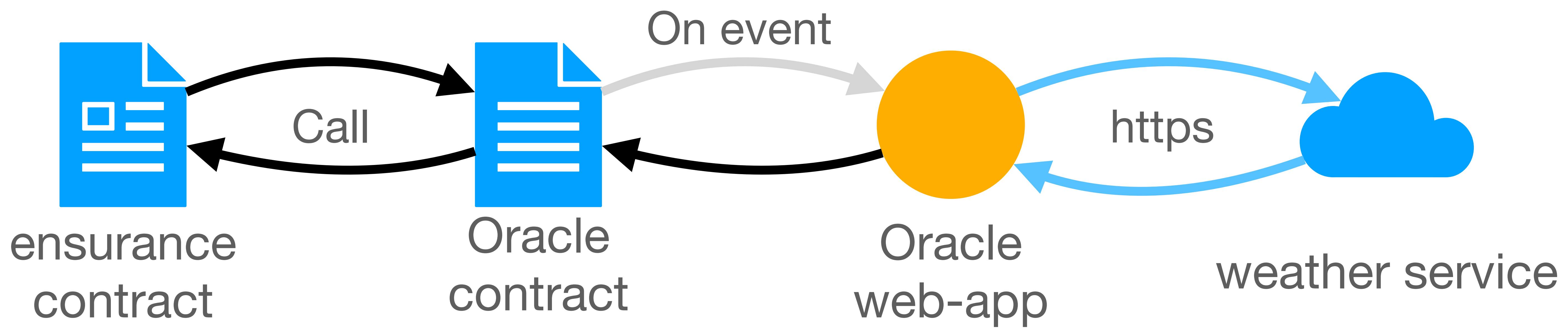
ensurance contract

weather service (yr.no)

Oracle

Example: Rain ensurance

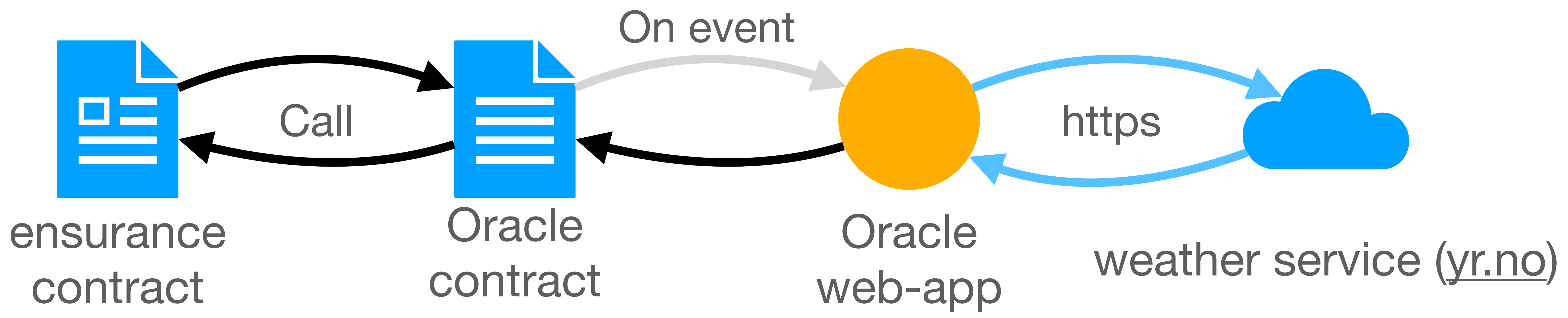
- Ensurance contract needs weather data to
  - Pay out policies
  - Determine prices



Oracle

Example: Rain ensurance

- Ensurance contract calls oracle contract
- Oracle contract emits event
- Oracle web app listens to event
- Web app gets data from api
- Web app invokes contract

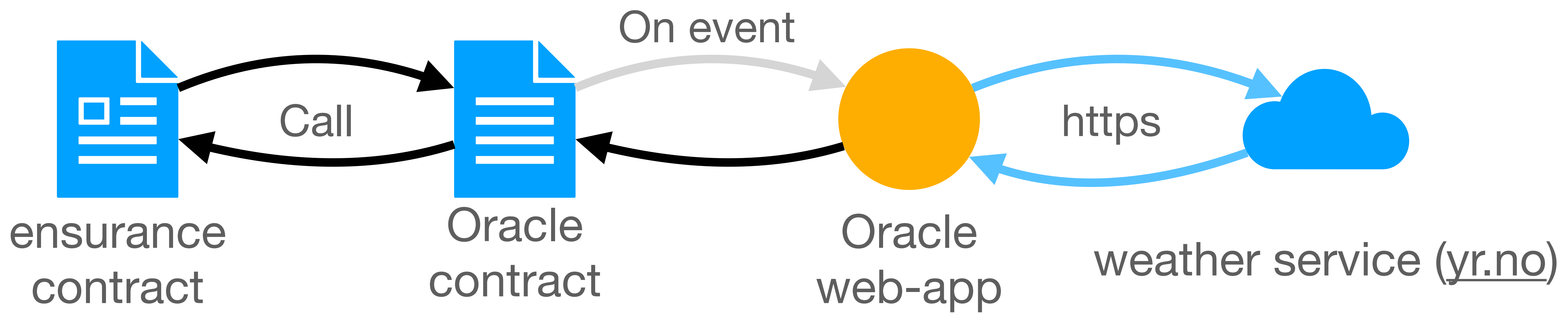


Oracle

Example: Rain ensurance

- Ensurance contract calls oracle contract
- Oracle contract emits event
- Oracle web app listens to event
- Web app gets data from api
- Web app invokes contract

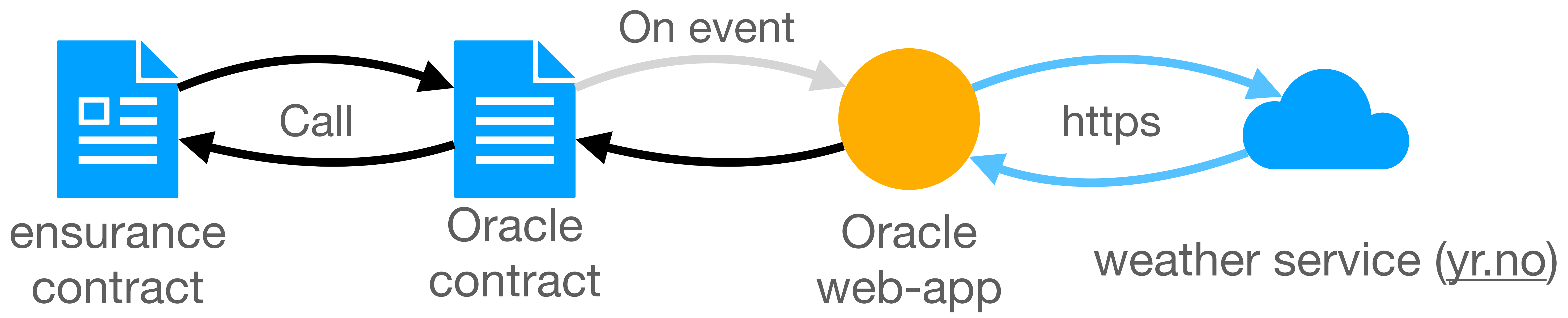
Check cryptozombies tutorial



Oracle

Example: Rain ensurance

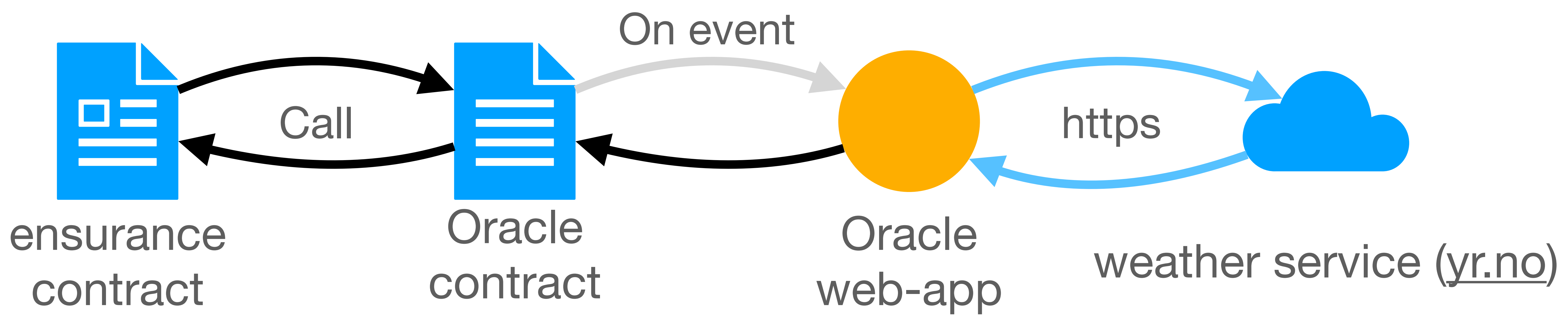
- Why should we use an extra oracle contract?
  - Can update if we need to update oracle
- Who do we need to trust?
  - Oracle provider, and API provider



Oracle

Example: Rain ensurance

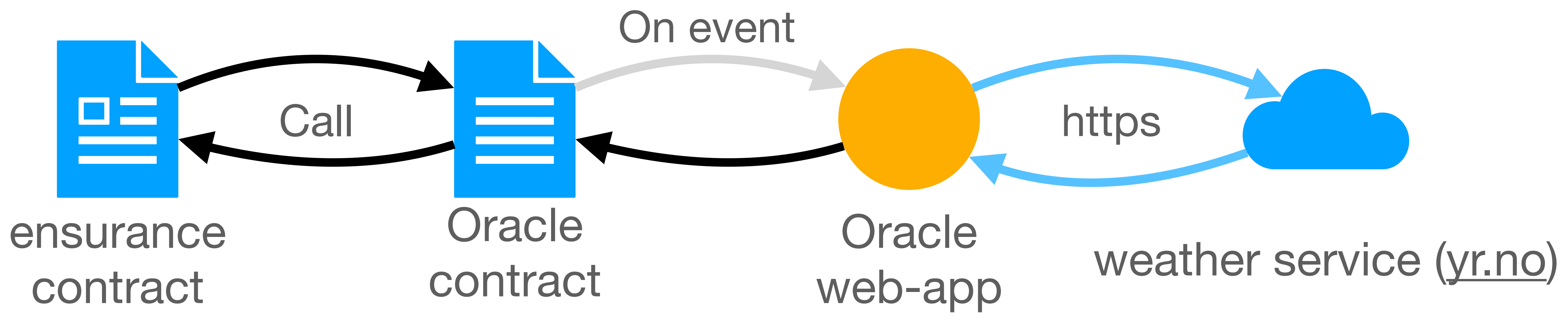
- Can we avoid trusting the oracle?
    - Yes, run oracle web-app in trusted execution (Intel SGX)



Oracle

Variations

- Access private data, e.g. using login
  - Yes, run oracle web-app in trusted execution (Intel SGX)
- Provide oracle service, that anyone can use



Off chain networks / Layer 2

Off-chain network / Layer 2

General idea

- **Idea:** If two parties agree, they can do a transaction outside of the chain without paying fees.
  - Once they disagree, they can use the chain to settle the dispute.
  - Can increase transaction throughput
  - Can give low fees

Payment Channels

Example: Uni-directional payment channel

- **Idea:** Allow any number of payments from A to B within given limit
- A creates contract with balance.
- A can send signed statements of B’s balance to B
- B can cash in his balance with the contract
- If B does not cash in, A can terminate the contract and get back the balance, after expiration date.

Check example
https://solidity-by-example.org/0.6/app/uni-directional-payment-channel/

Payment Channels

Example: Bi-directional payment channel

- **Idea:** Allow any number of payments between A and B within given limit,
- A and B both pay a balance to contract
- A and B can send signed statements of their balances to each other, with increasing nonces
- A or B can submit balance, signed by both to contract. This triggers countdown
- If other party does note submit a balance with larger nonce, balances are payed out.

Check example
https://solidity-by-example.org/0.6/app/bi-directional-payment-channel/

Payment Channels

Example: Bi-directional payment channel

- **Idea:** Allow any number of payments between A and B within given limit,

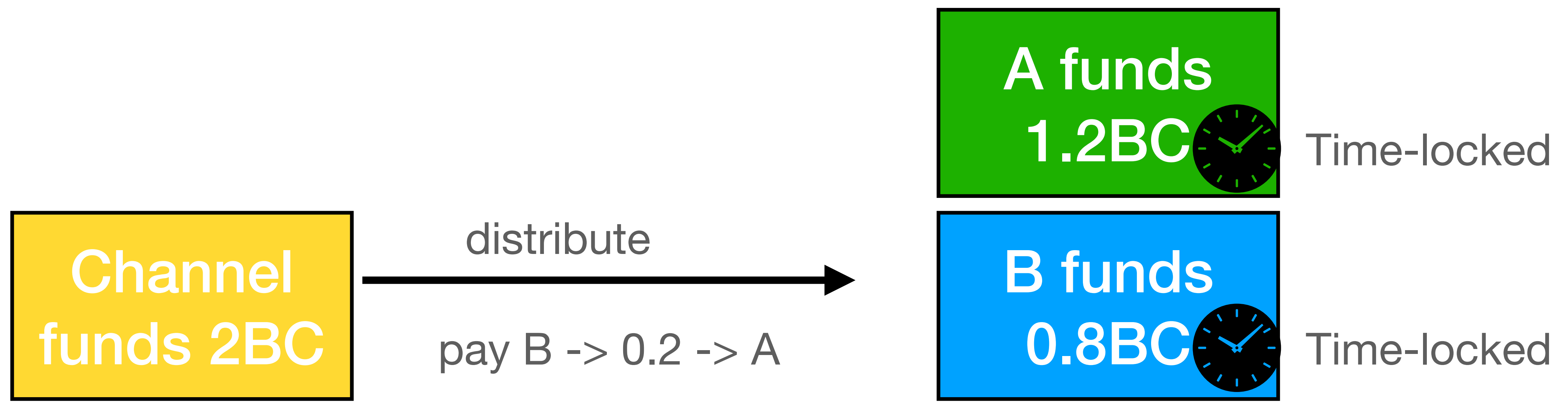
Problem:

- Timeout
- Locked funds

Payment Channels

Example: Lightning channels on UTXO

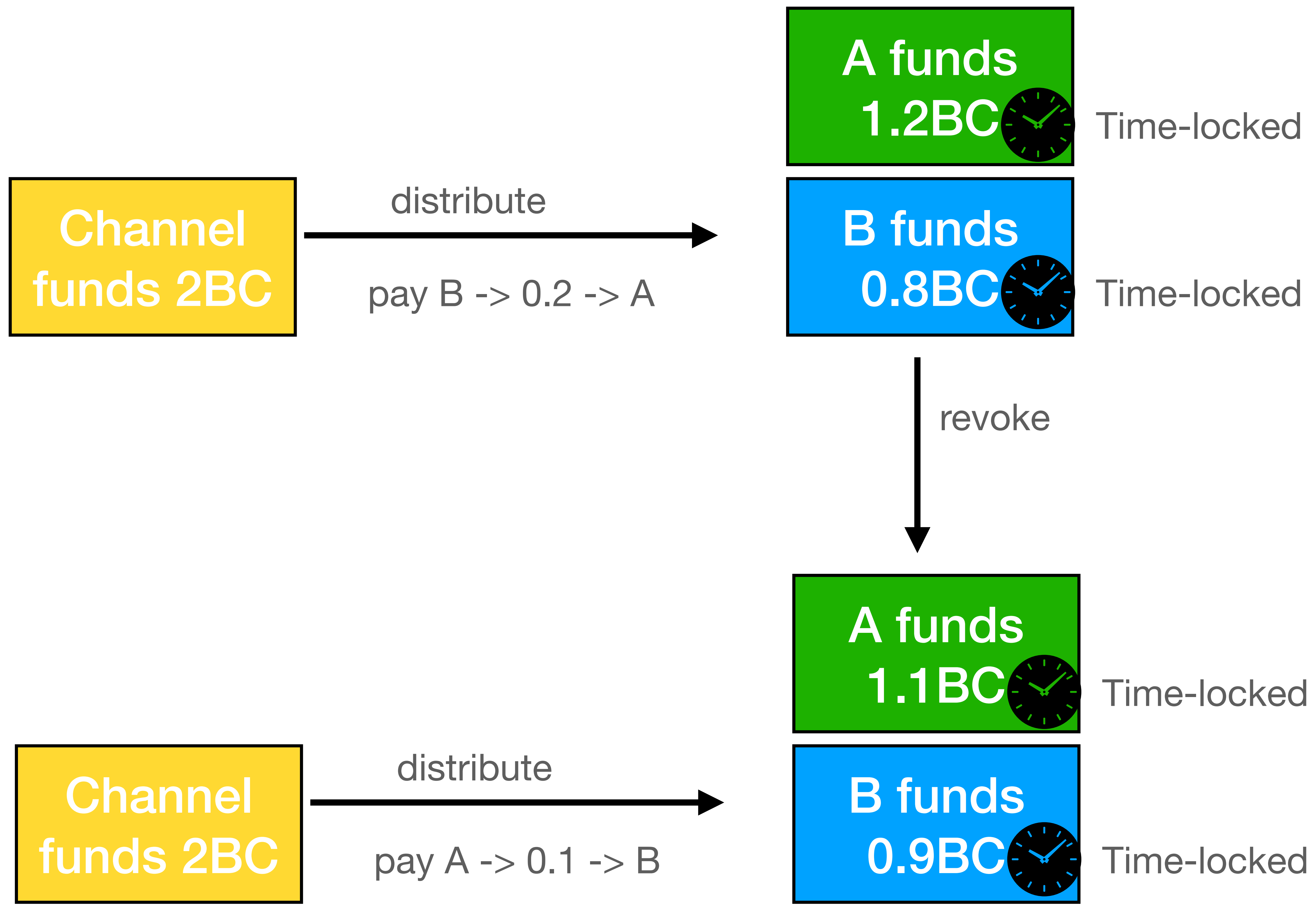
- **Funds are locked in one Output**
- **First channel payment:**
  Create valid transaction,
  to distribute funds
  (not submitted)
- **Second payment**
  Create valid transaction to
  distribute funds, and
  revocation transaction



Payment Channels

Example: Lightning channels on UTXO

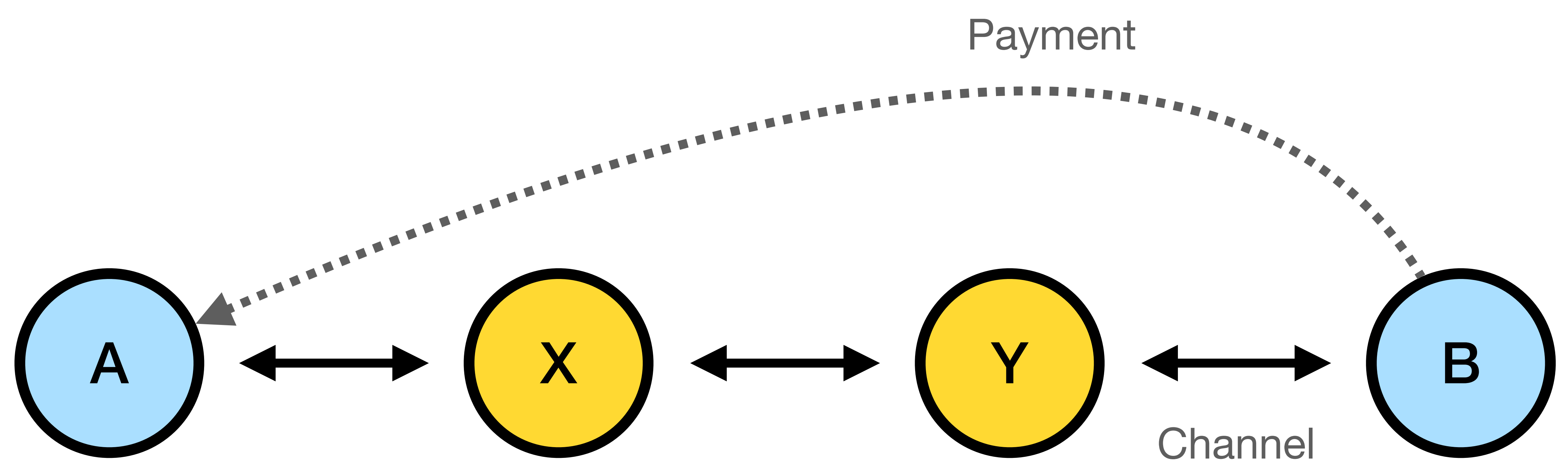
- **Funds are locked in one Output**
- **First channel payment:**
  Create valid transaction,
  to distribute funds
  (not submitted)
- **Second payment**
  Create valid transaction to
  distribute funds, and
  revocation transaction



Payment Channels

Example: Multi hop payment

- **Idea:** payment across multipe channels
- Pay fees to intermediates (X and Y)

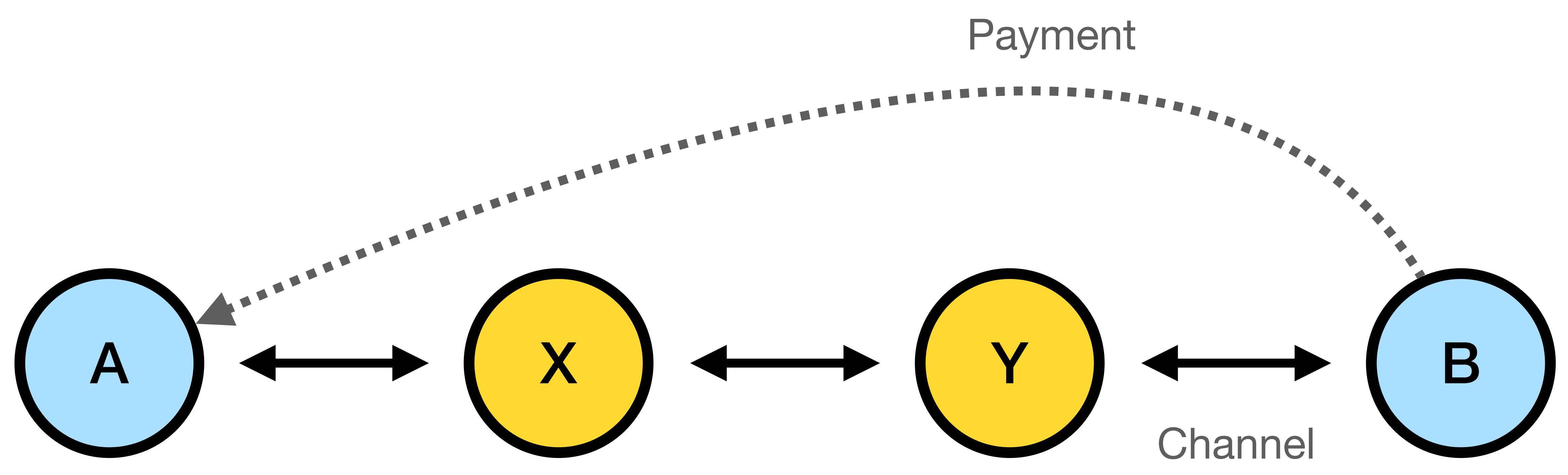


- **How:** Conditional payments, with secret known to A
- B -> Y; Y -> X; … s. t. payment is only valid if participants know the secret.
- Friendly settlement: Secret forwarded A -> X -> Y
- Unfriendly settlement: A publishes secret on chain, X and Y can see secret.

Payment Channels

Example: Payment routing

- **Find path from B to A**



Problem:

- Limitted & changing channel capacity
- Fees play a role
- Privacy of transaction plays a role, *e.g. avoid intermediaries knowing who pays what to whom.*

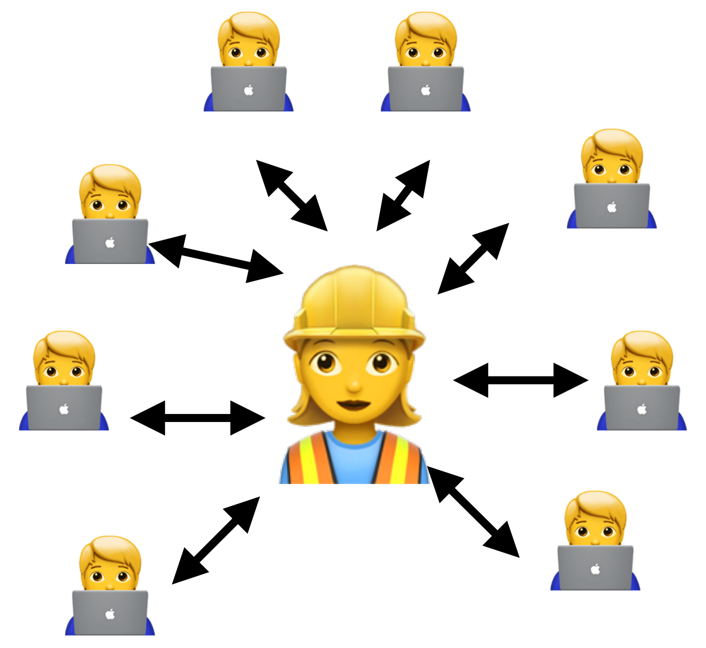
Payment Channels

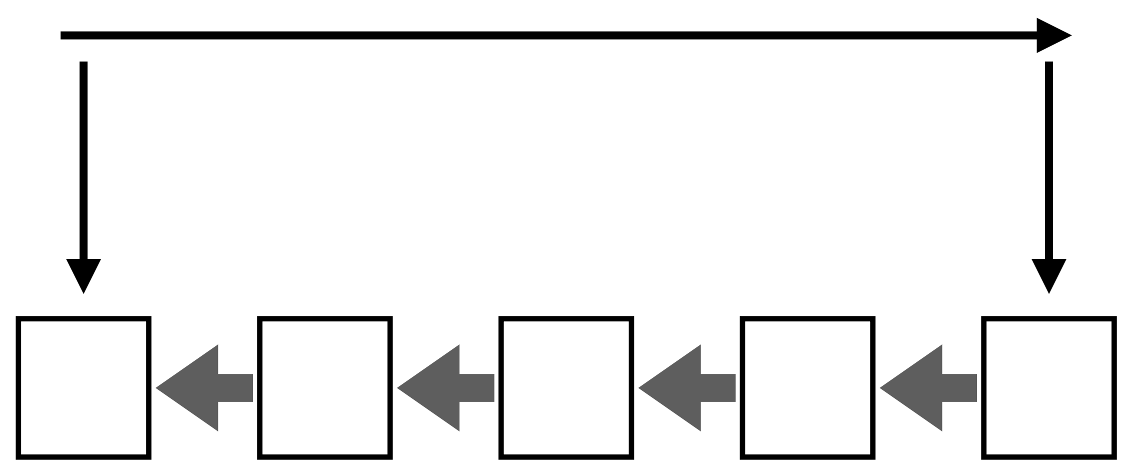
Example: Other channels

- **Virtual channels:**
  - Given two payement channels A <-> I and I <-> B, create a virtual channel between A <-> B.
  - Intermediate is only involved in opening and closing the virtual channel.
  - Fewer fees
- **State channels:**
  - A channel where we can create smart contracts.
  - Only channel members can interact with these contracts.

Commit chains

- **Idea:** Similar to payment channel with single central node (operator)
- Operator regularly publishes root of state (merkle tree root)
- To finalize operations, need to wait for next state root.
- Can retrieve funds, on chain, according to last state root.
- Members need to check, that state updates are correct.



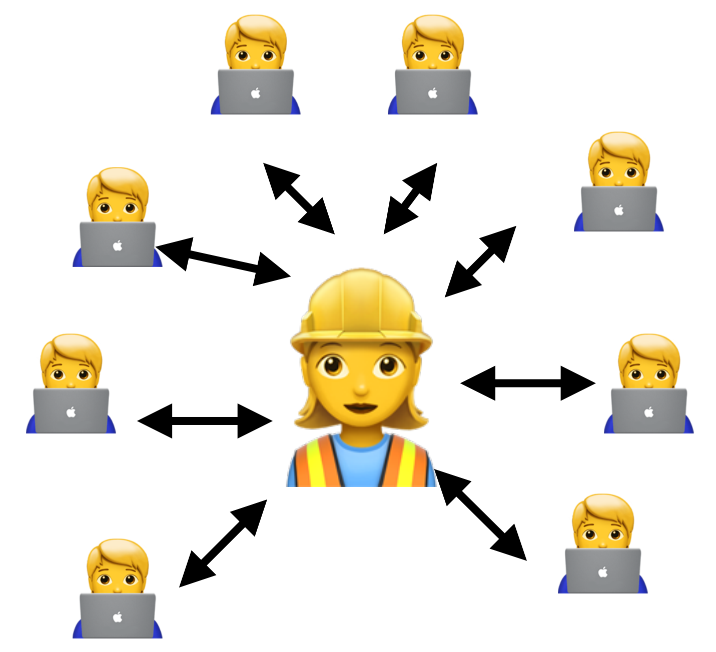
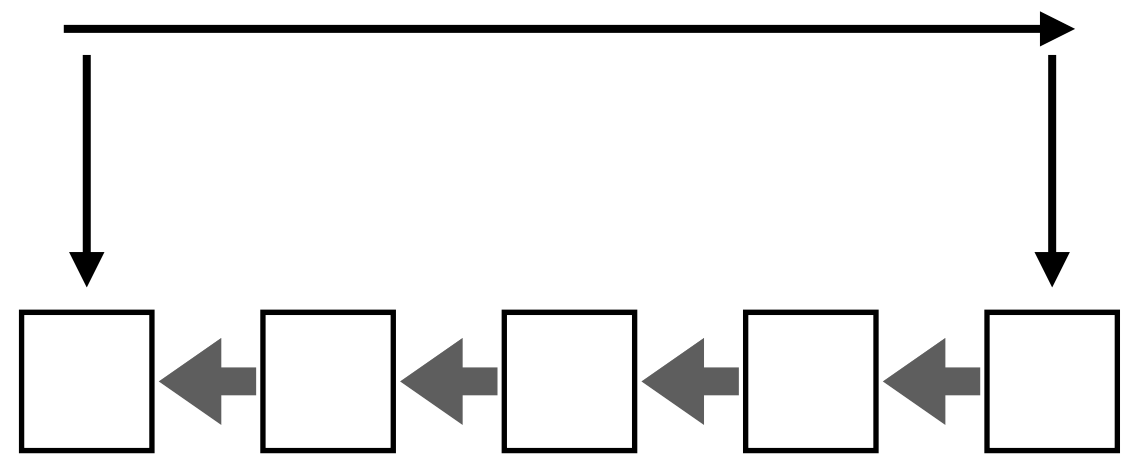


Commit chains

What is submitted to the blockchain?

- Merkle root of new state:
  Need to check that transition is correct

- Zero-knowledge proof:
  Ensures correct transition
  Needs to be checked

Channels and Commit chains

Assumptions

- **Synchrony:**
    - *Transactions submitted to the blockchain are executed within a max time bound*
    - Needed to submit complaint in time
- **Online:**
    - *Participants need to stay online.*
    - Needed to detect/react to misbehaviour

Off Chain comparison

	On chain transaction	Channel	Commit chain
Cheep fees	✗	✓	✓
Fast confirmation	✗	✓	✗
Can go offline	✓	✗	✗
Unlimitted capacity	✓	✗	✓
Joining	Not necessary	Setup cost	No cost