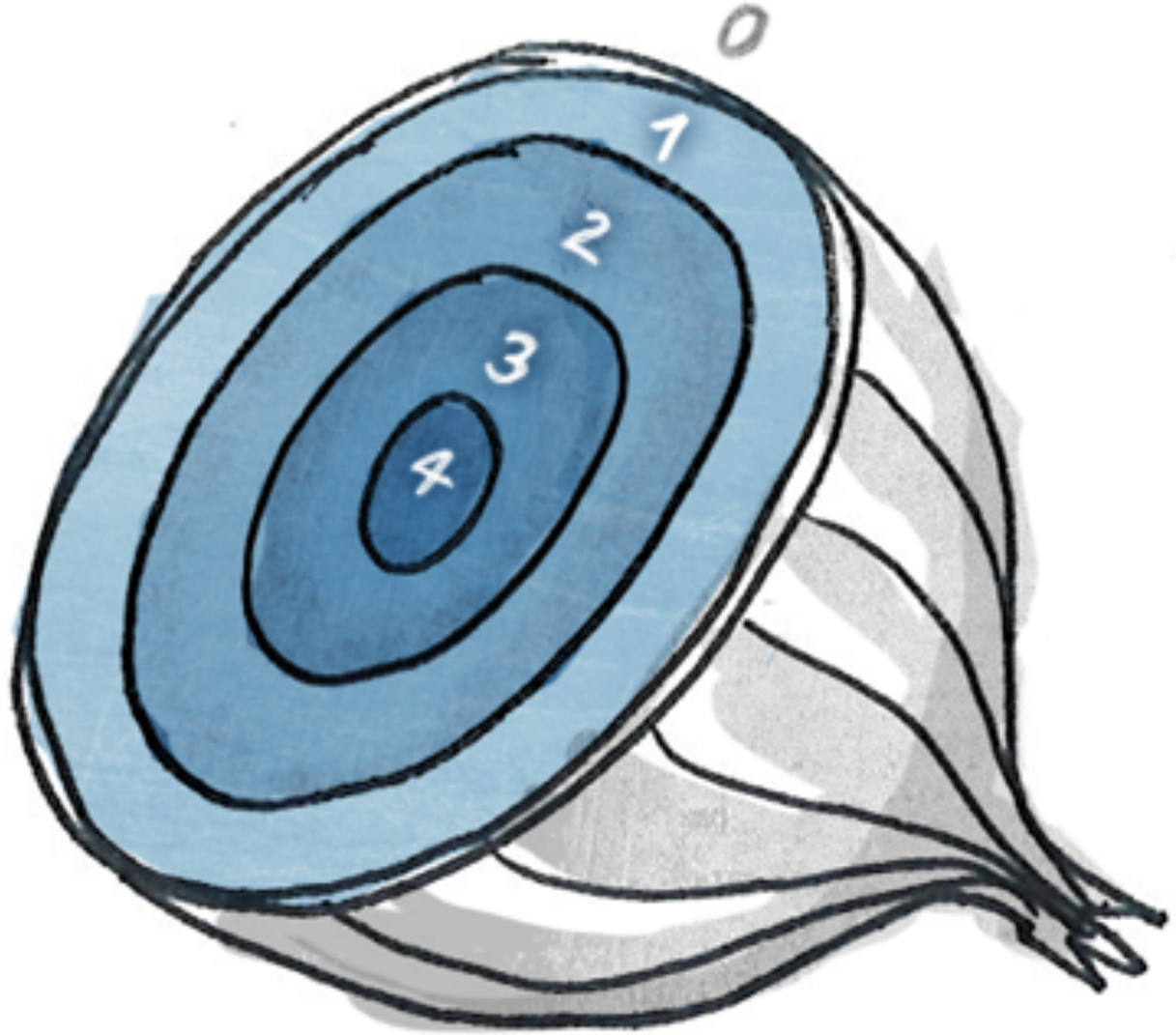
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Application for Broadcast
[**Reliable**, **Uniform**, **Causal**, and **Total-Order**]

Adi Seredinschi
Distributed Programming Laboratory

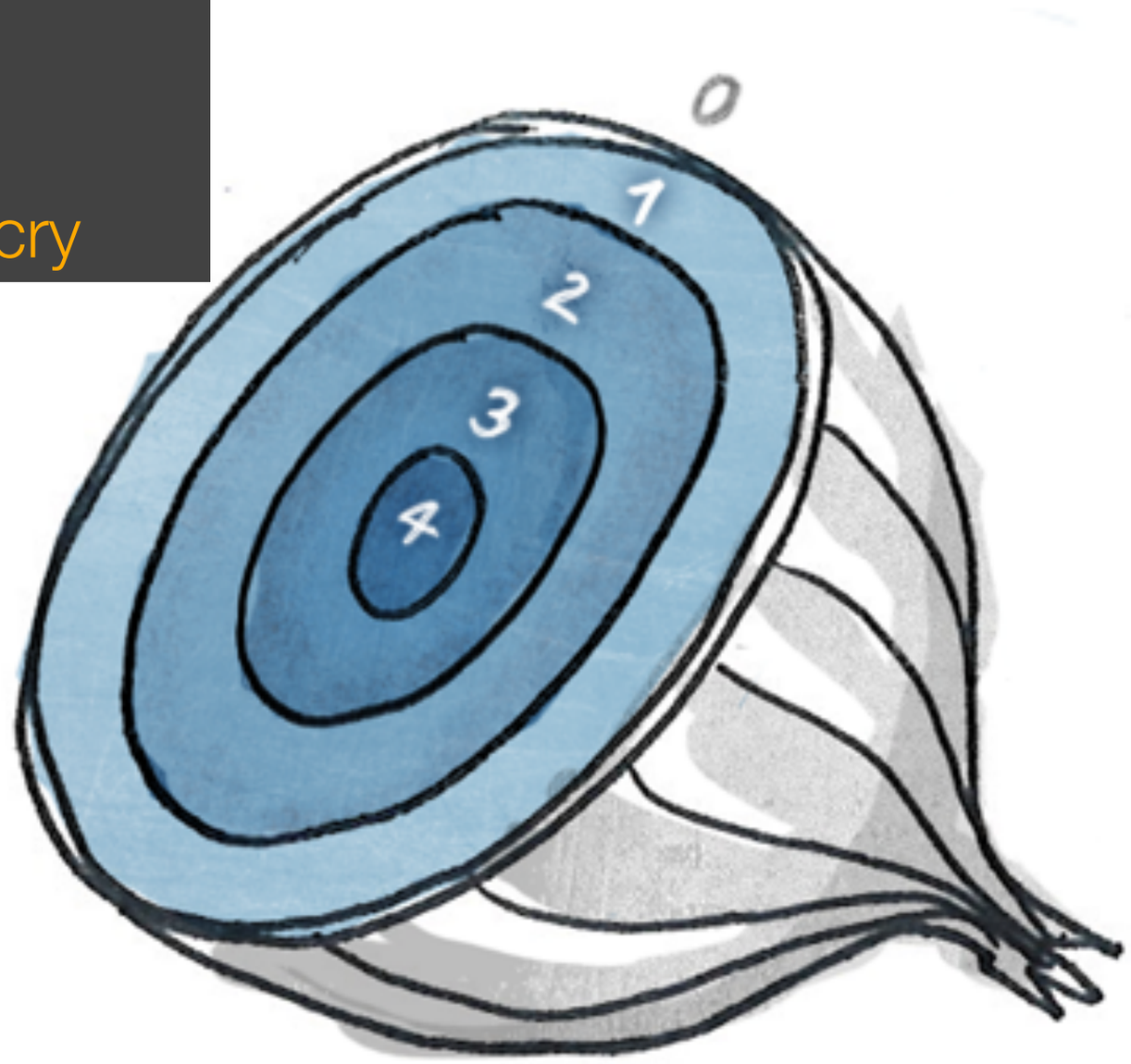
Question:
What do Distributed Systems have in common with Onions?



Question:
What do Distributed Systems have in common with Onions?

Answer:

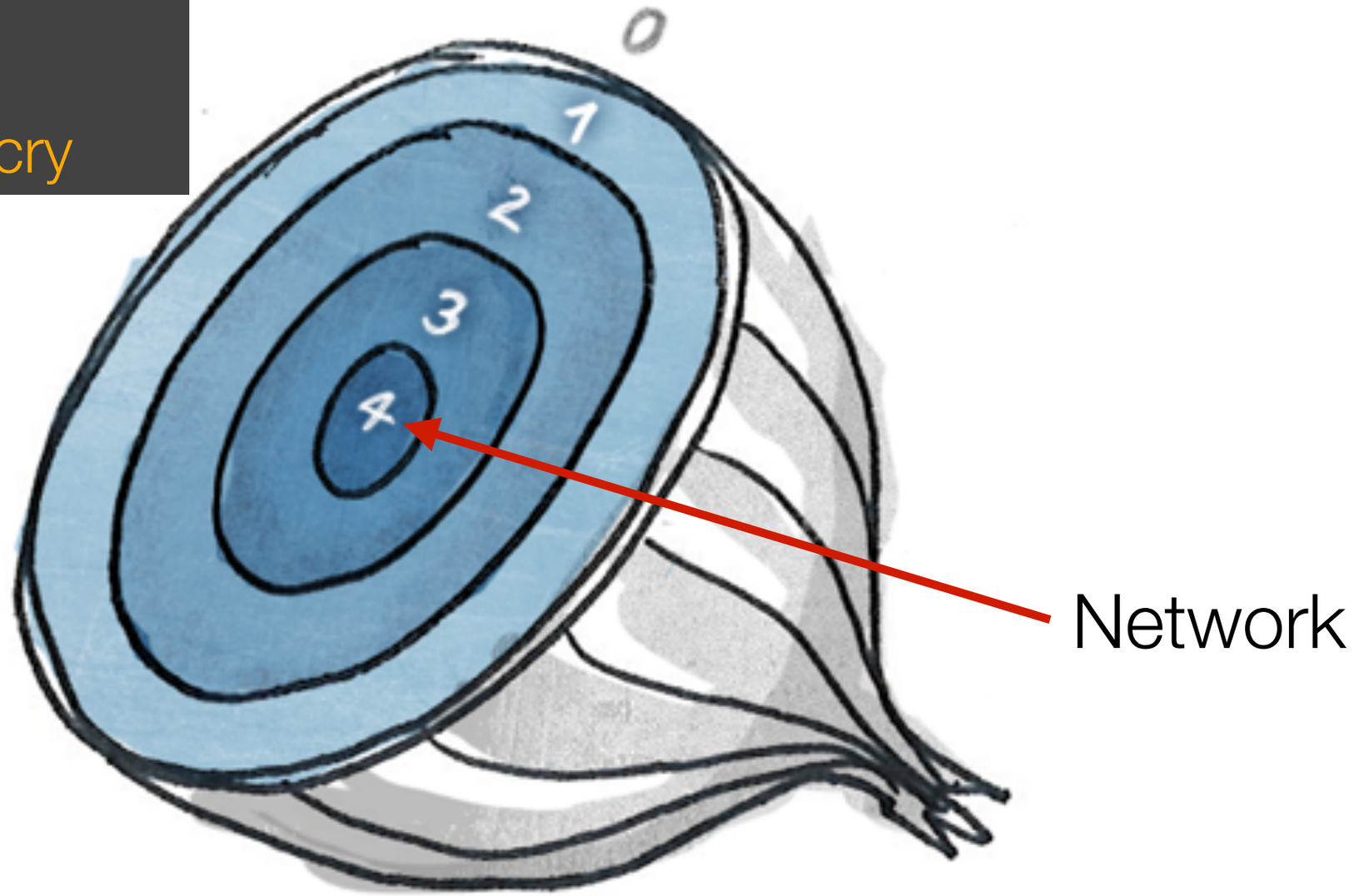
1. Layering
2. Abstraction
3. They make you cry



Question:
What do Distributed Systems have in common with Onions?

Answer:

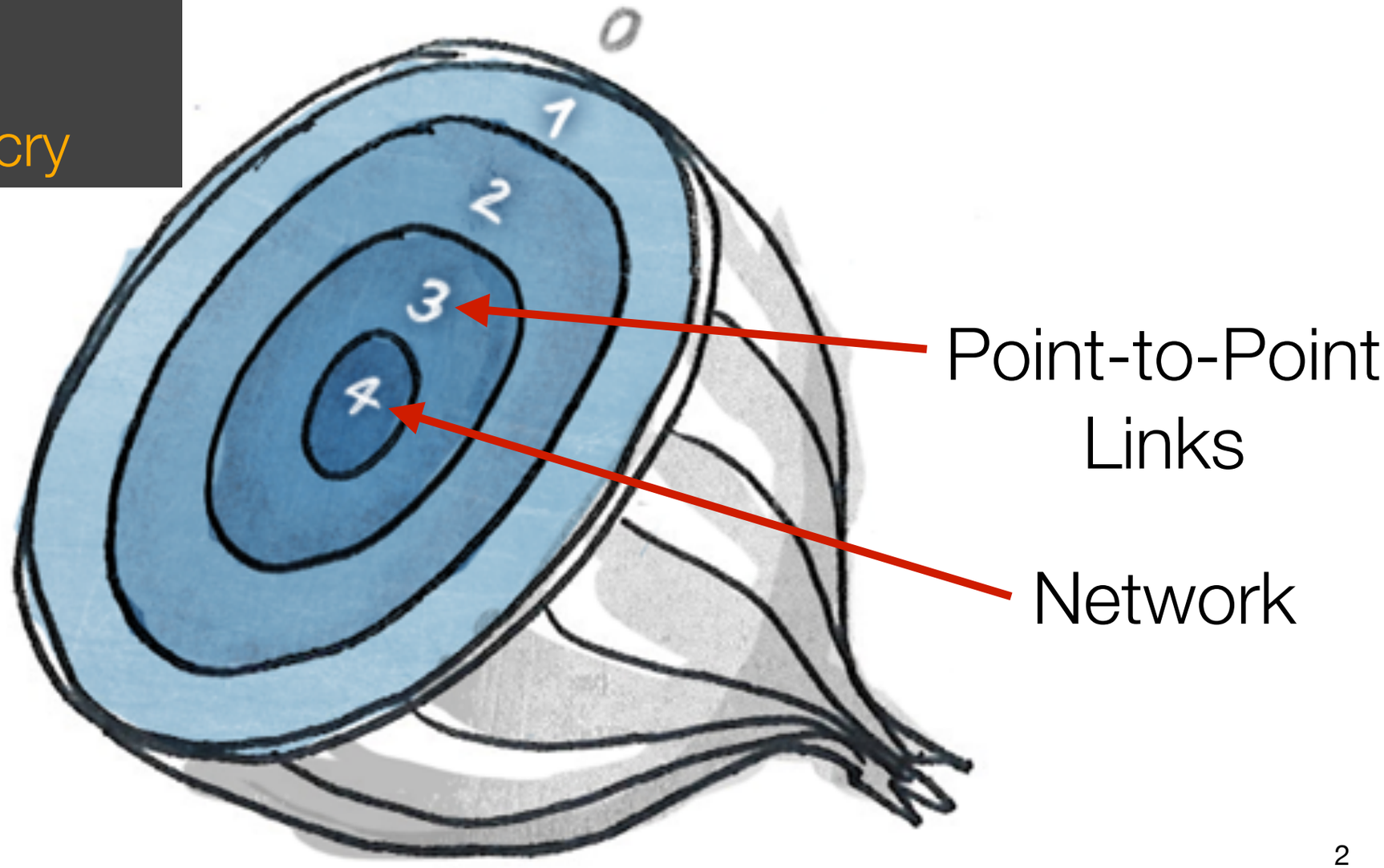
1. Layering
2. Abstraction
3. They make you cry



Question:
What do Distributed Systems have in common with Onions?

Answer:

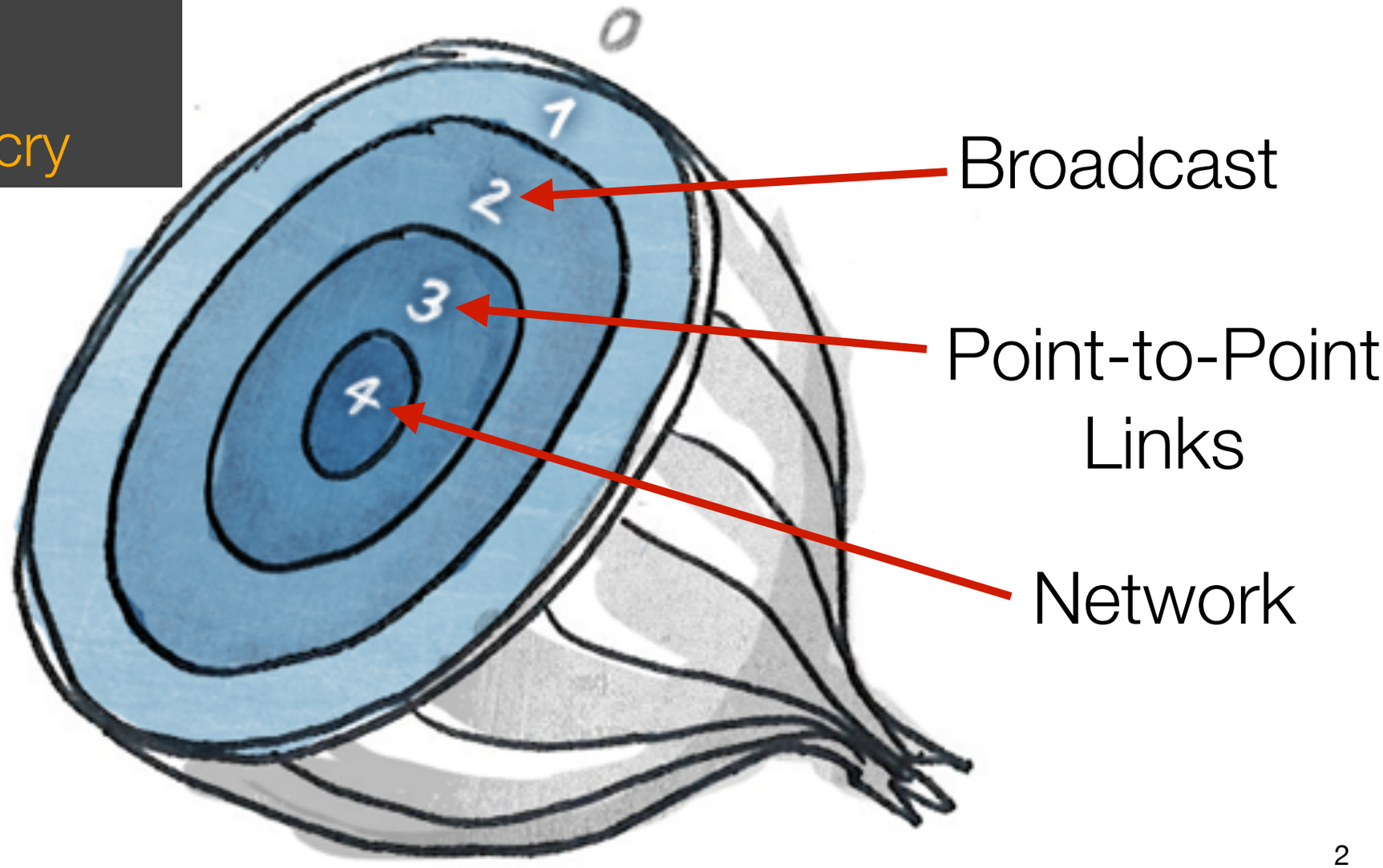
1. Layering
2. Abstraction
3. They make you cry



Question:
What do Distributed Systems have in common with Onions?

Answer:

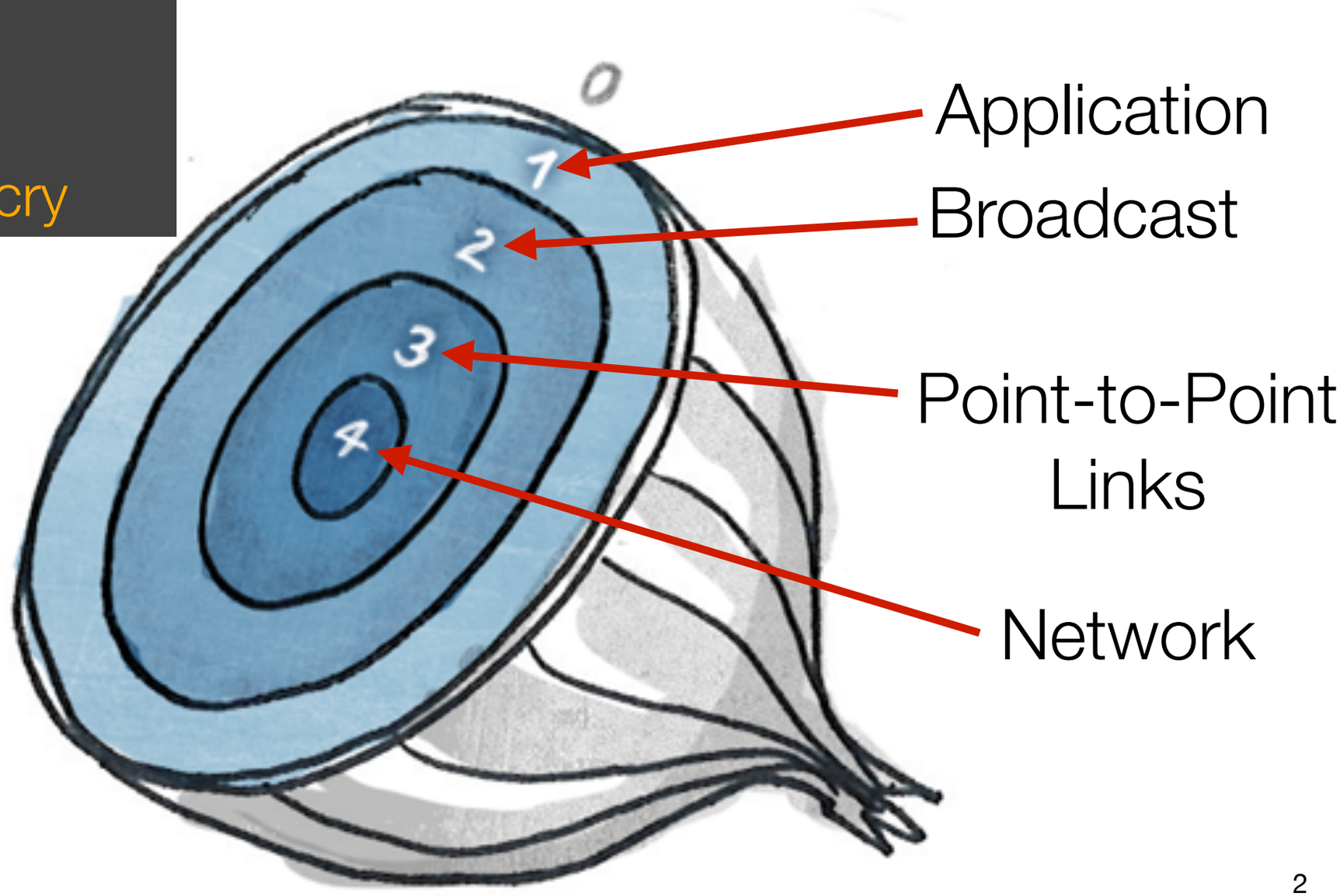
1. Layering
2. Abstraction
3. They make you cry



Question:
What do Distributed Systems have in common with Onions?

Answer:

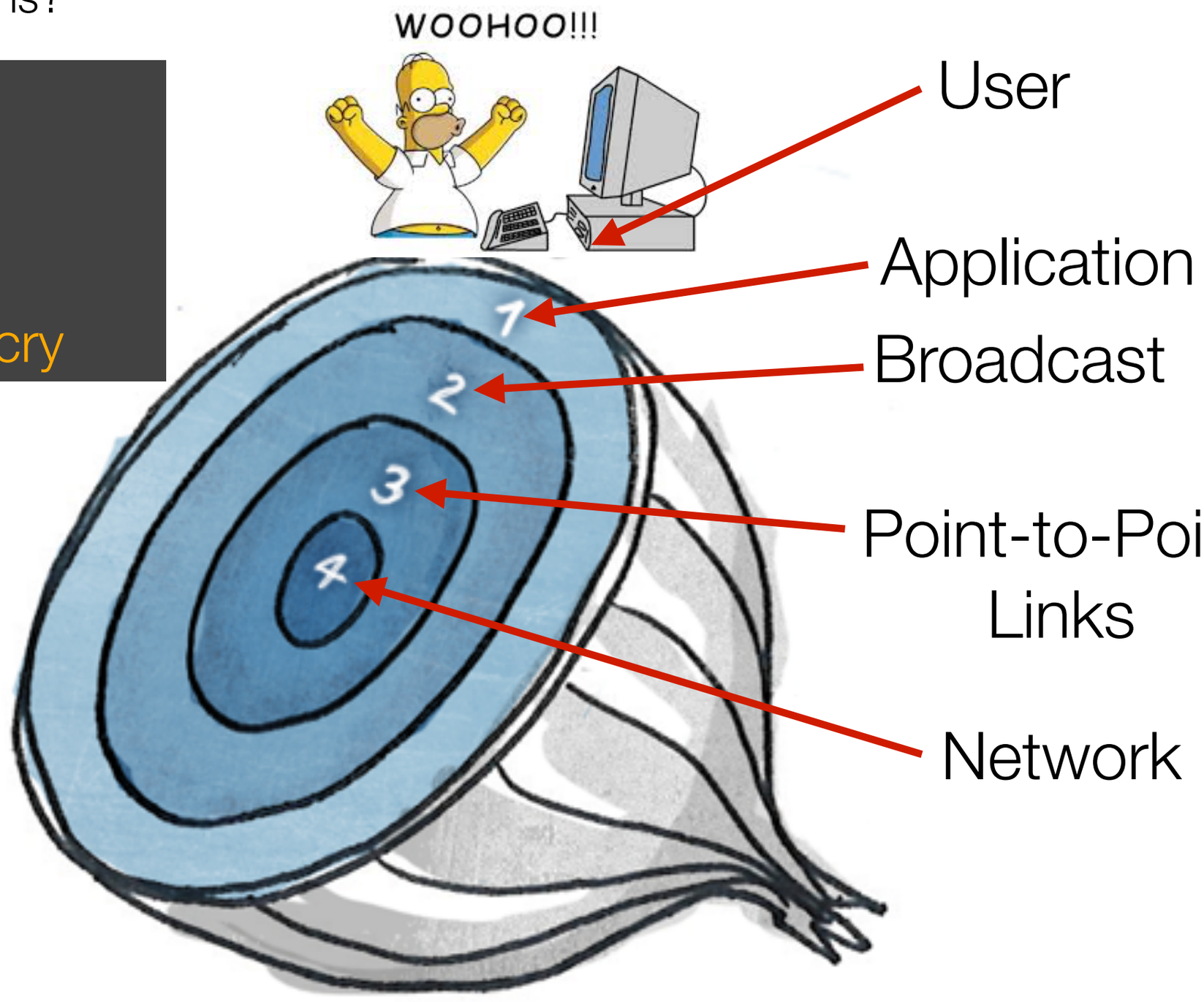
1. Layering
2. Abstraction
3. They make you cry



Question:
What do Distributed Systems have in common with Onions?

Answer:

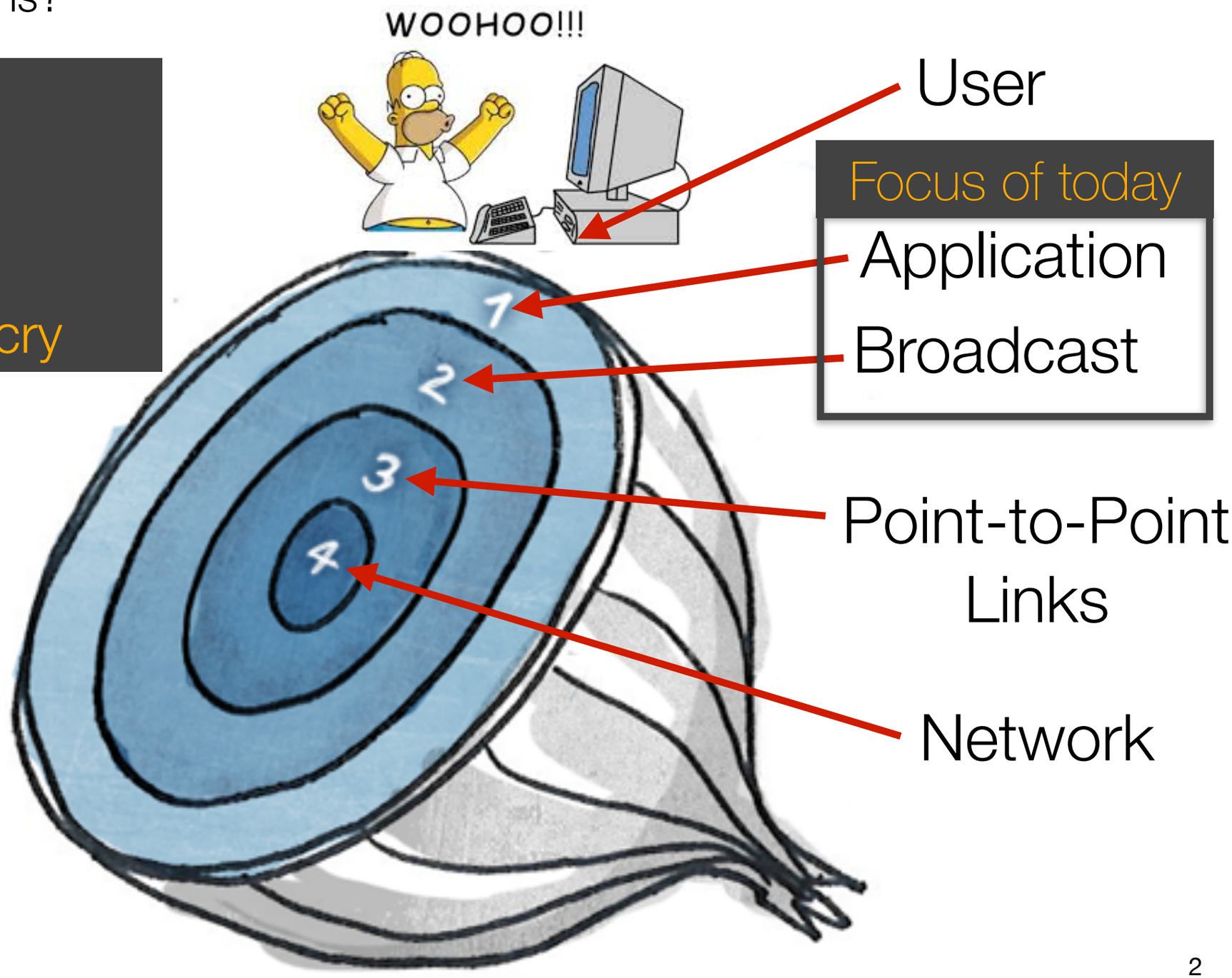
1. Layering
2. Abstraction
3. They make you cry



Question:
What do Distributed Systems have in common with Onions?

Answer:

1. Layering
2. Abstraction
3. They make you cry



Let’s design a simple application

Broadcast
(we will investigate different properties..)

CAMIPRO-Bitcoin

- The canonical Bitcoin design
  - Uses gossip (best-effort broadcast)
    - Relies heavily on crypto — no time to cover that
- **We will not discuss the canonical design**
  - **Instead, we will design our own version of Bitcoin**
  - **Optimized for CAMIPRO**

CAMIPRO-Bitcoin

Conceptual goals:

1. Replace traditional CAMIPRO
   - Based on CHF
2. Make banks obsolete

CAMIPRO-Bitcoin

Conceptual goals:

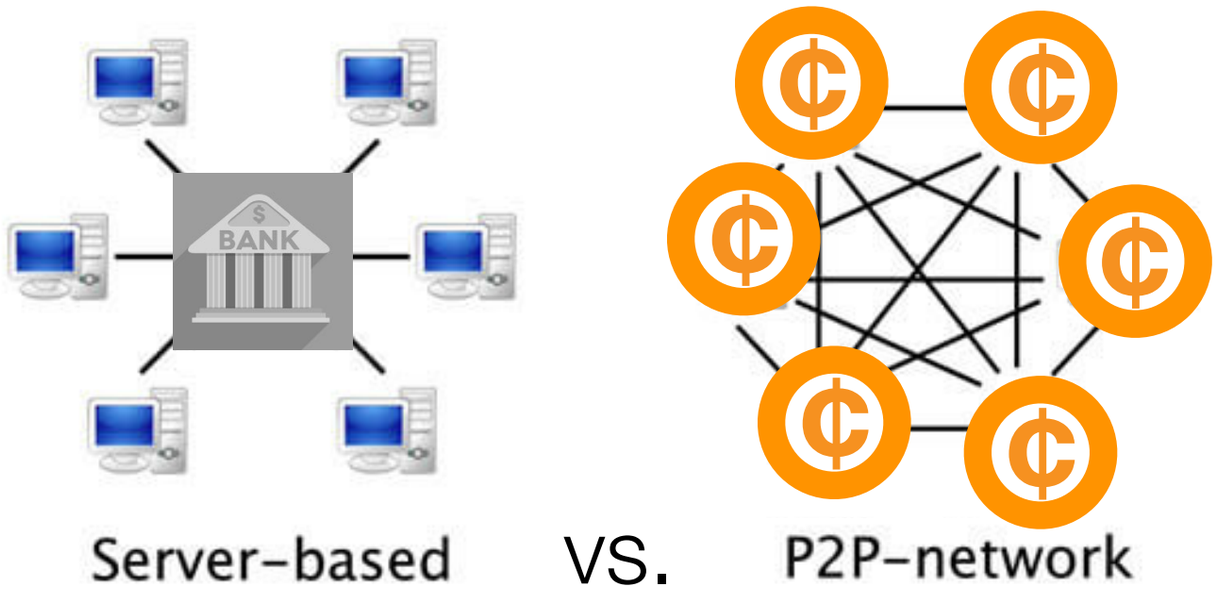
1. Replace traditional CAMIPRO
   - Based on CHF
2. Make banks obsolete

CAMIPRO-Bitcoin

Conceptual goals:

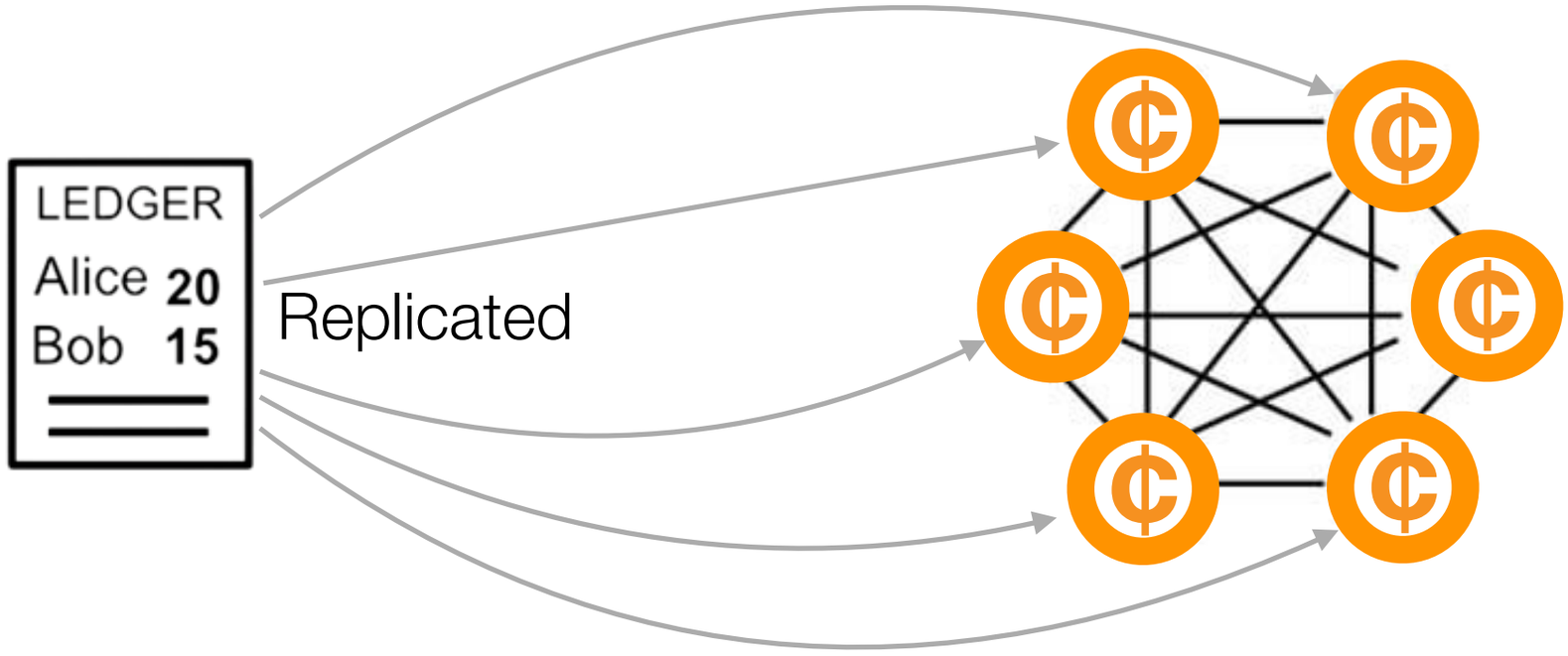
1. Replace traditional CAMIPRO
   - Based on CHF
2. Make banks obsolete

IN PRACTICE



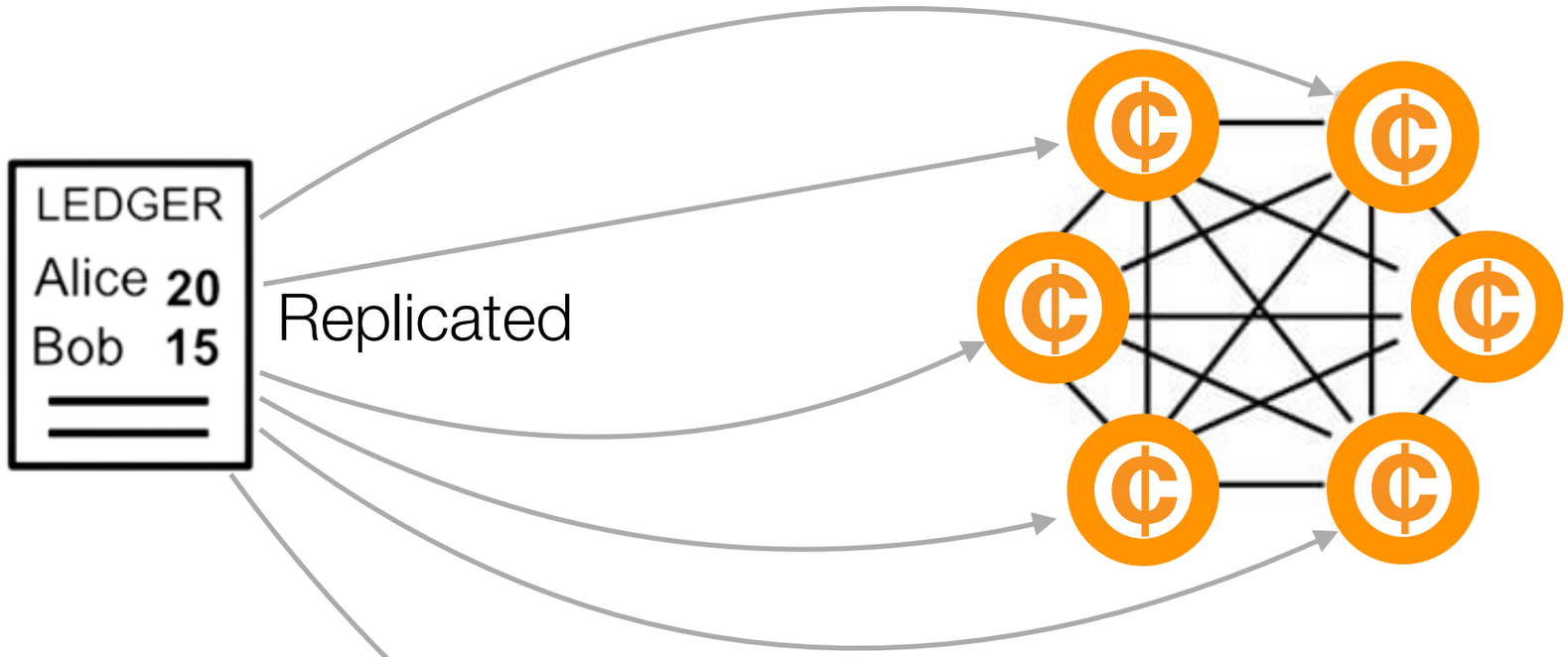
Main concepts

- Ledger

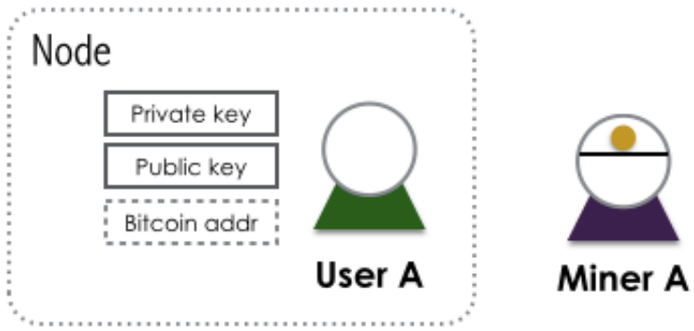


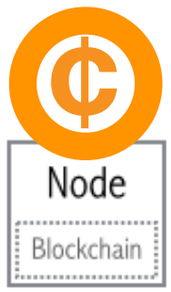
Main concepts

- Ledger



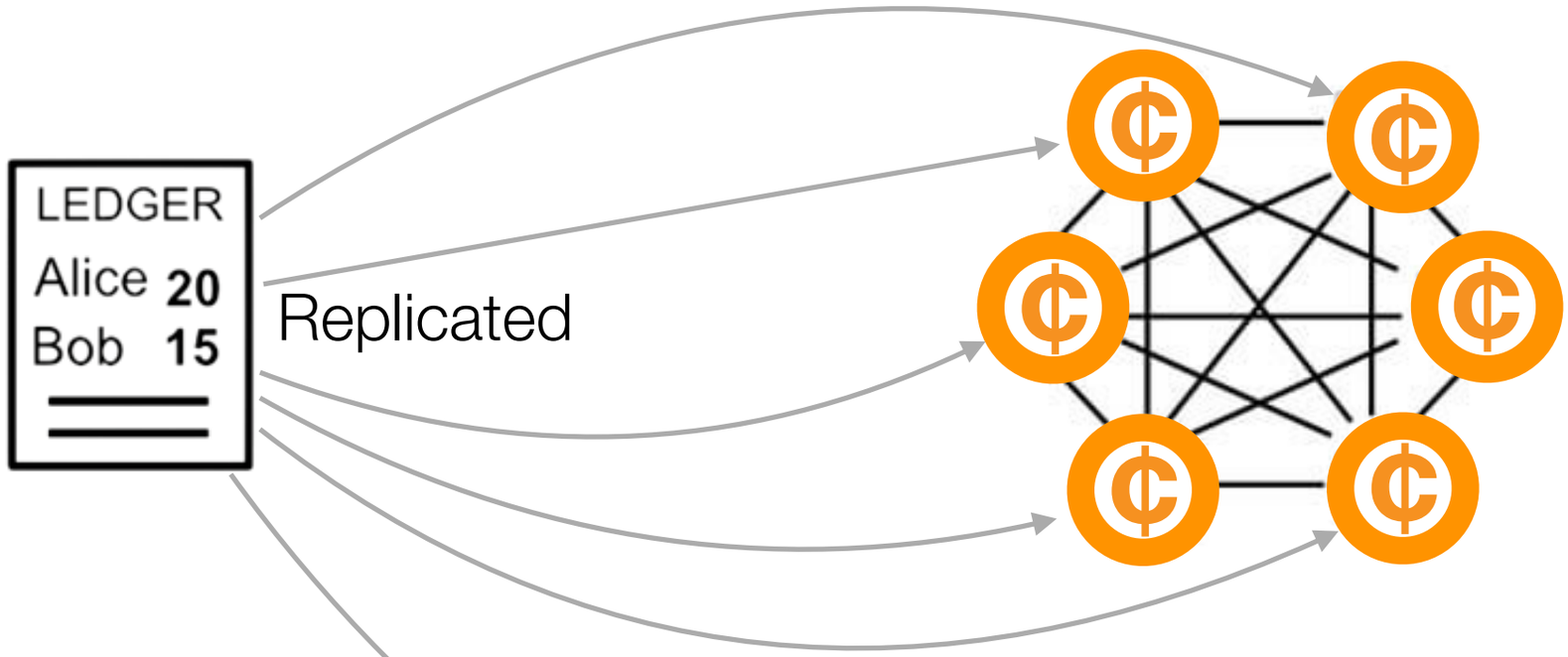
- Node



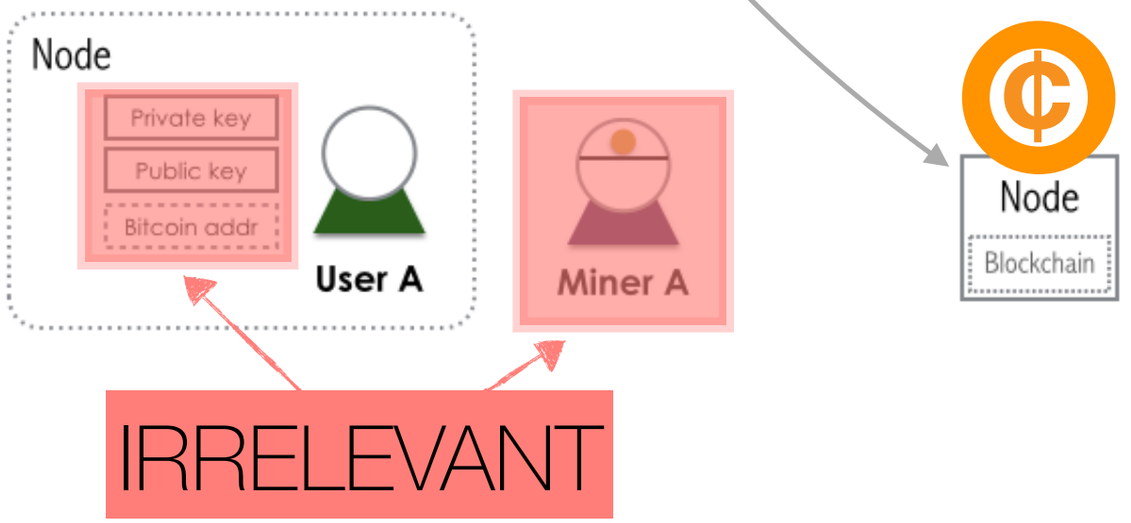


Main concepts

- Ledger

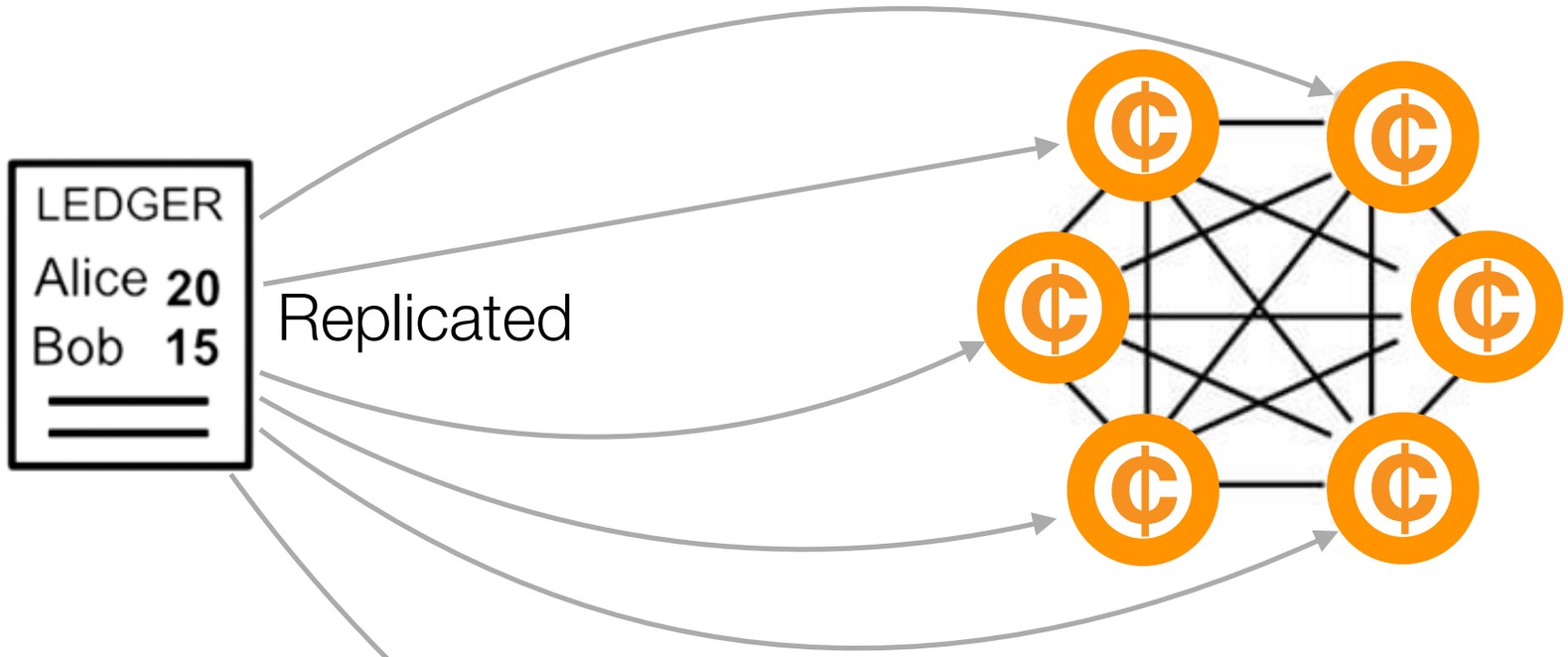


- Node

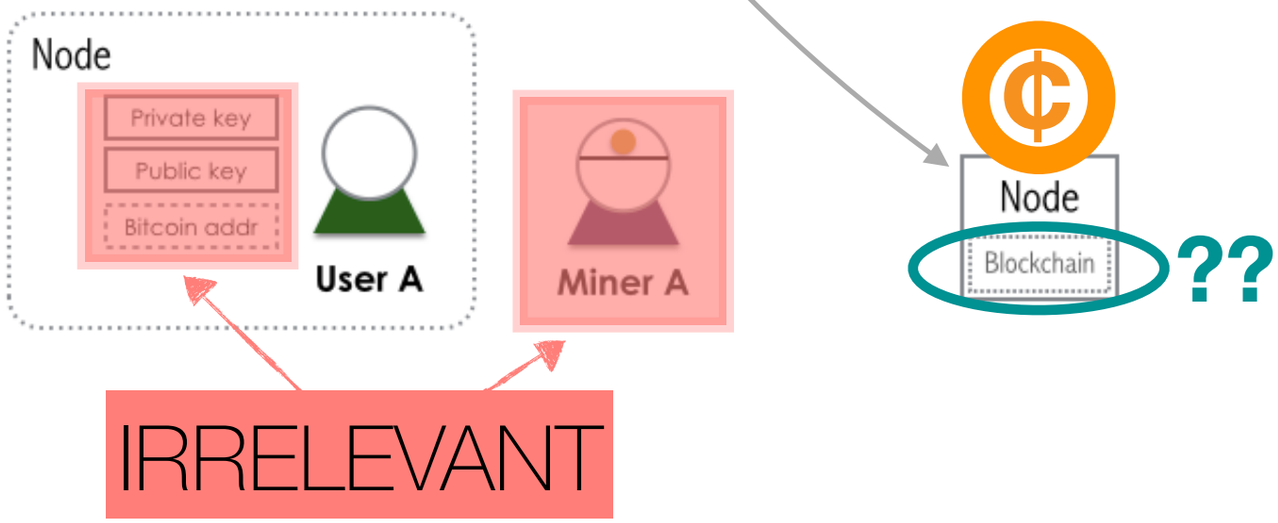


Main concepts

- Ledger

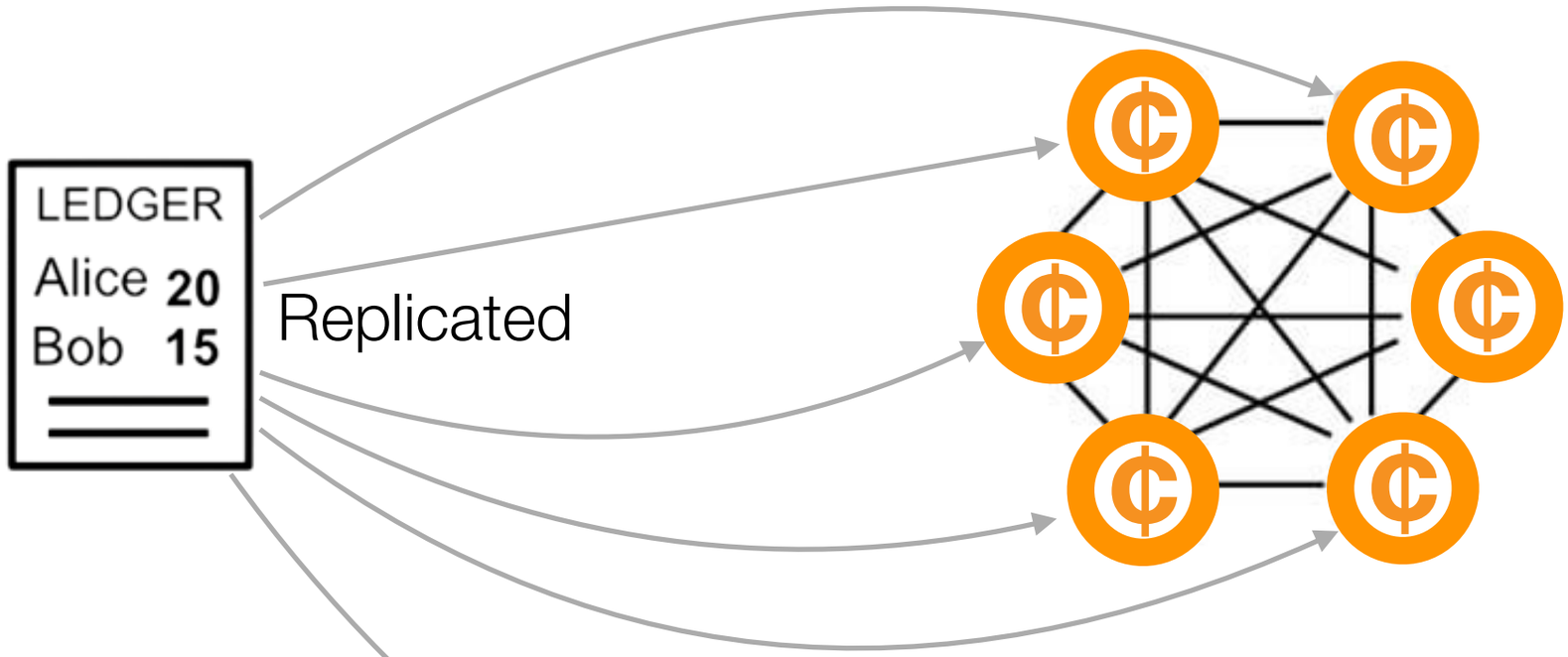


- Node

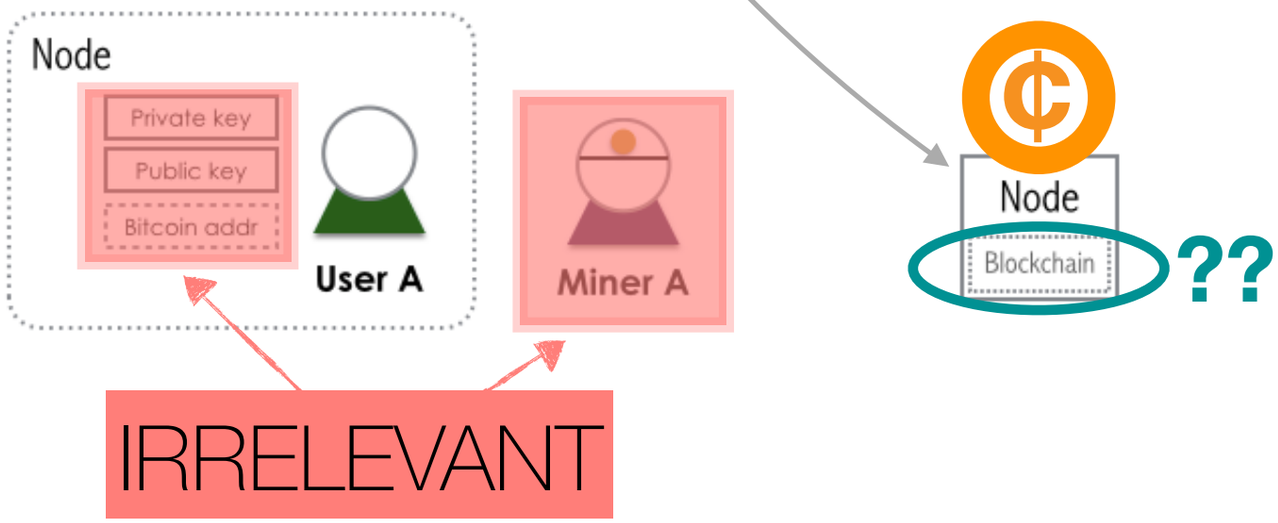


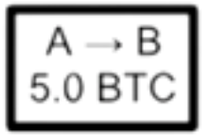
Main concepts

- Ledger



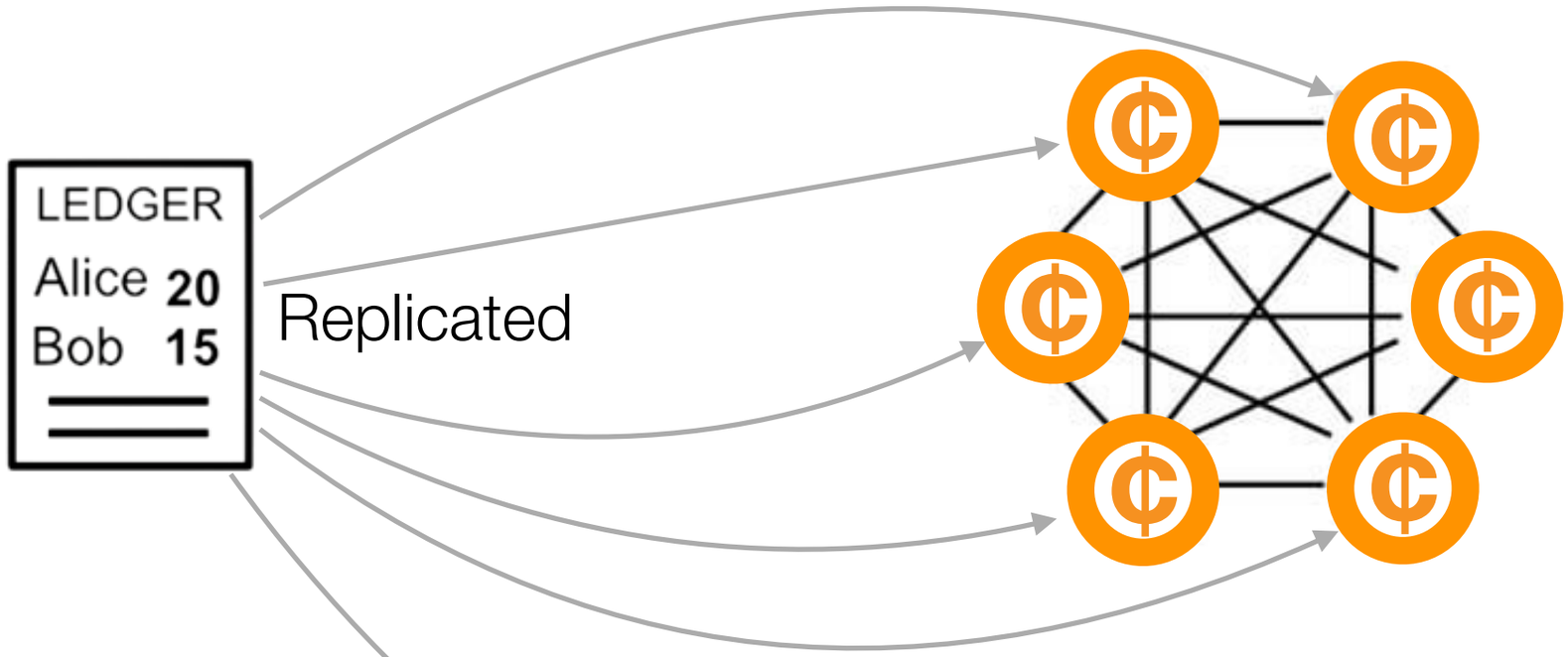
- Node



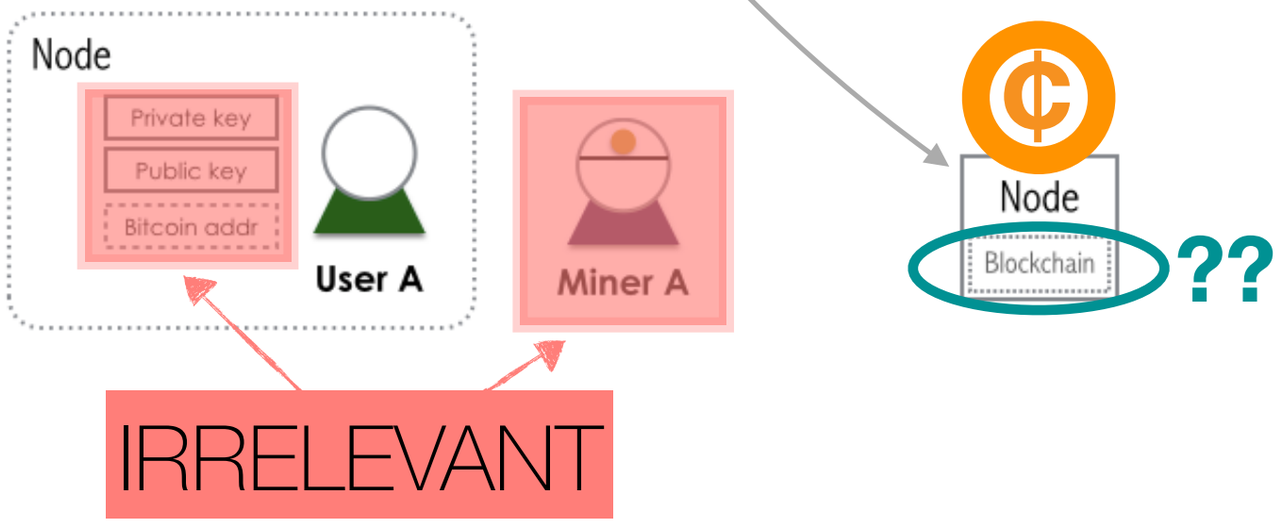
- Transaction  “Alice gives some money to Bob”

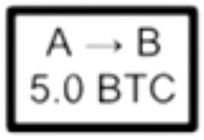
Main concepts

- Ledger



- Node

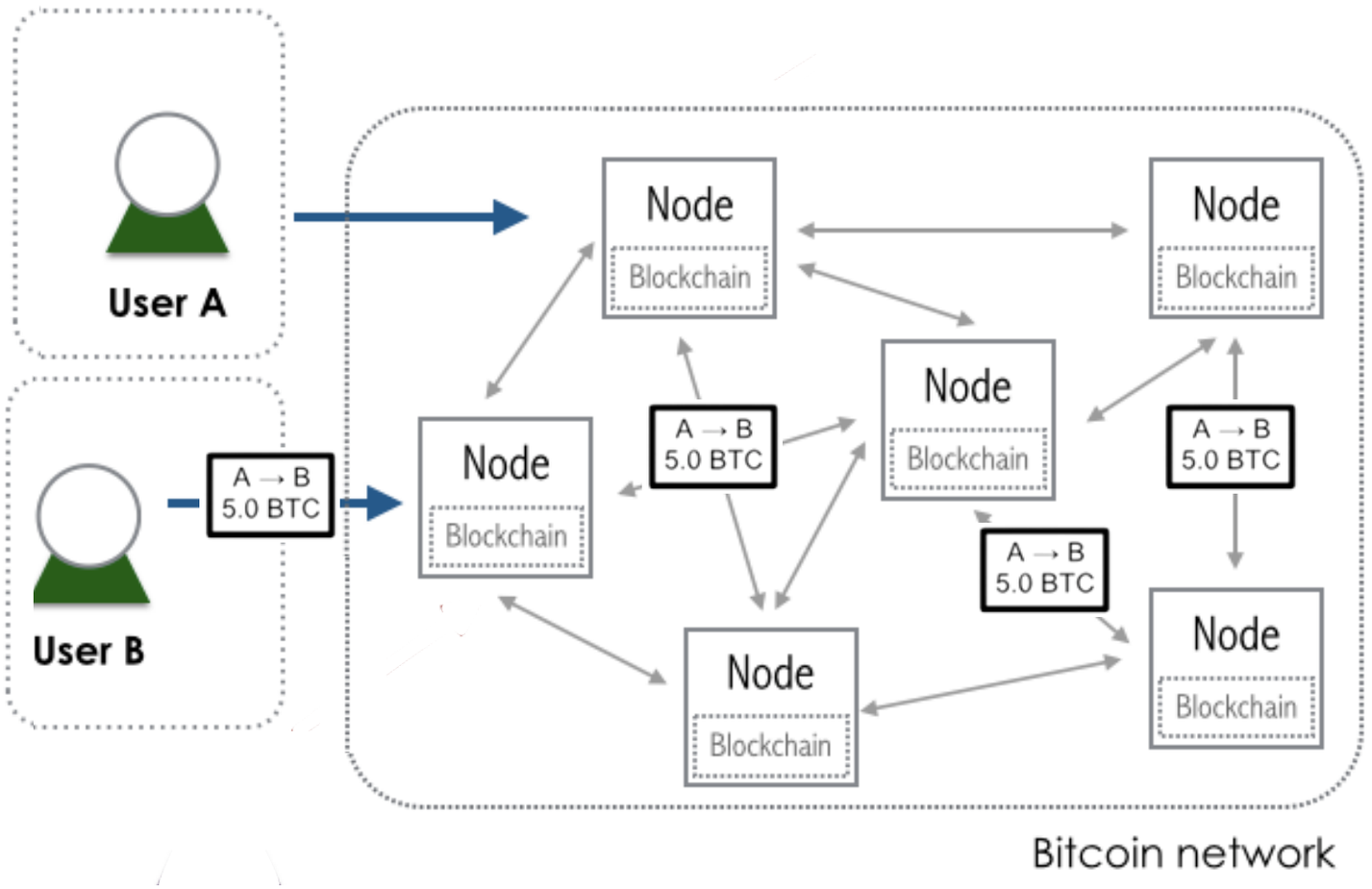


- Transaction 

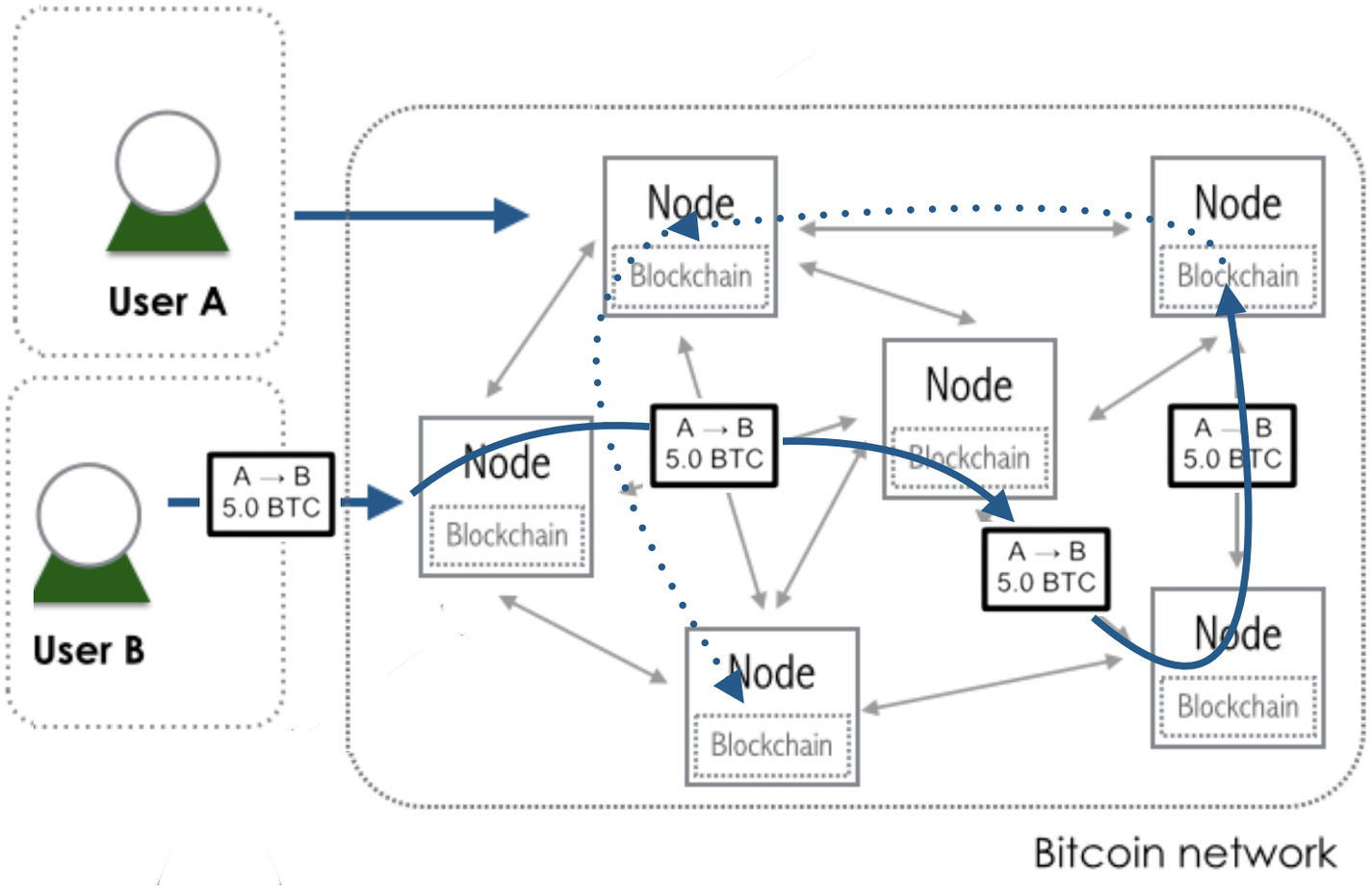
"Alice gives some money to Bob"

Let's see how they all fit together..

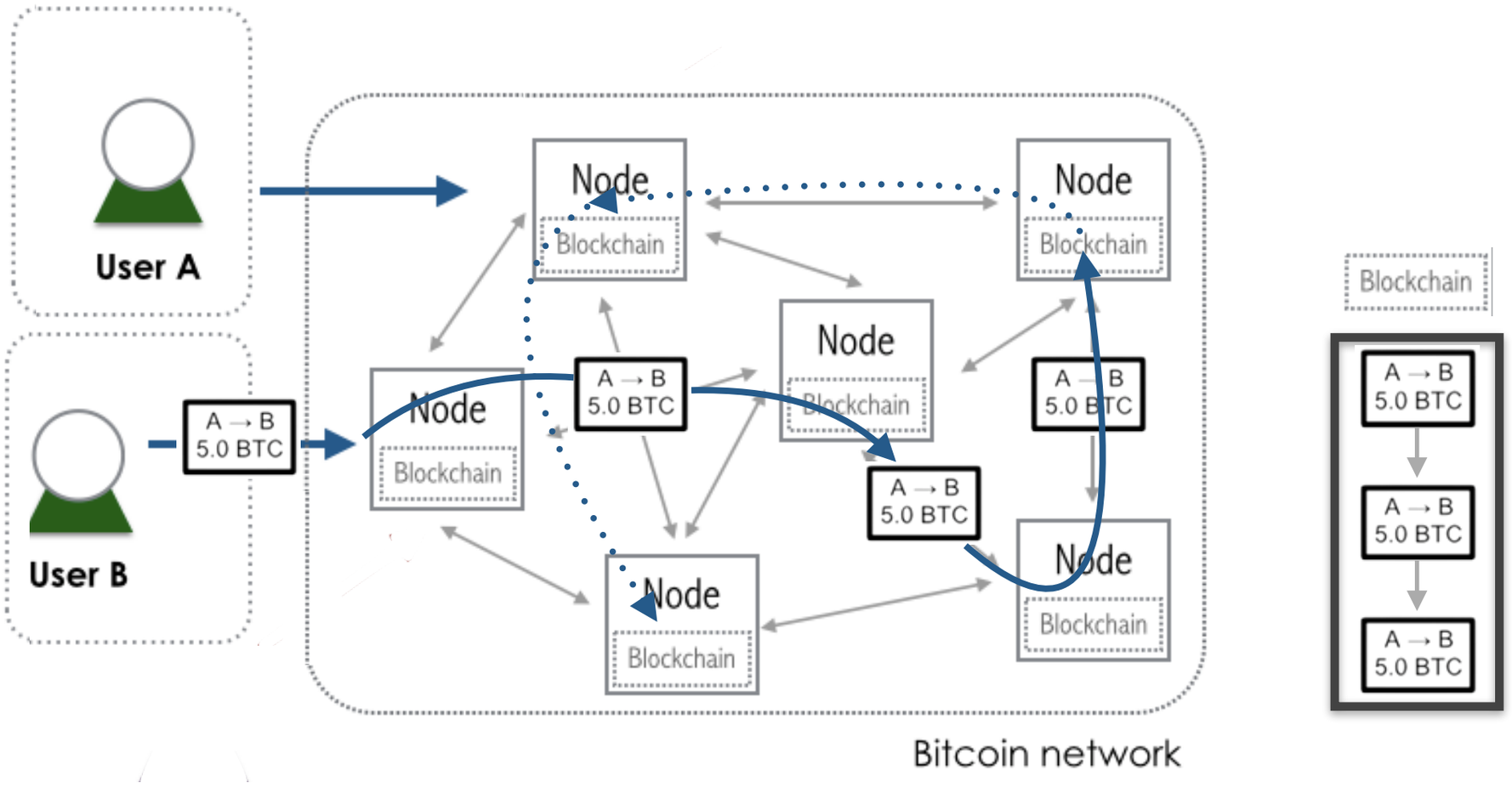
High-level interactions



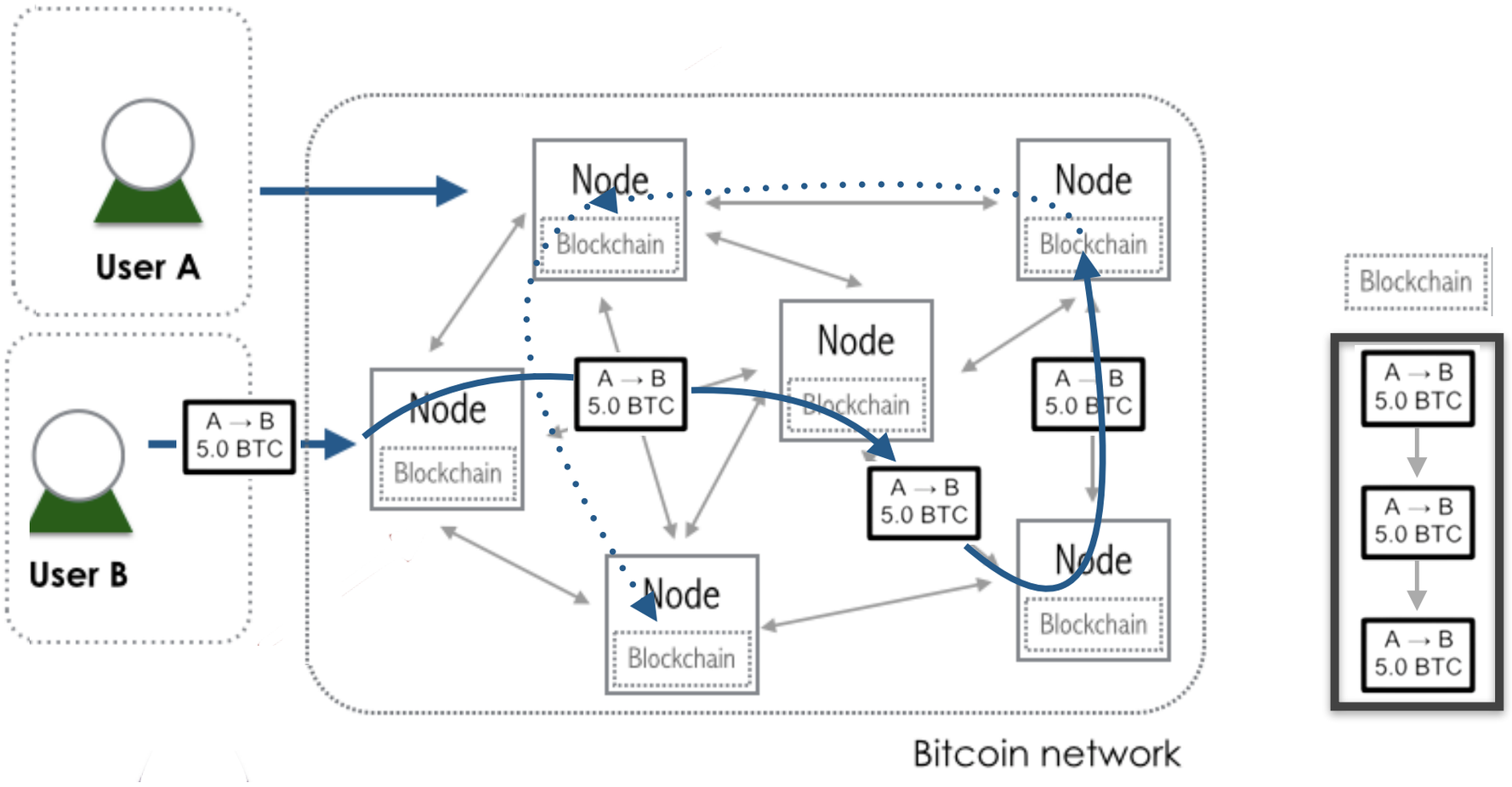
High-level interactions



High-level interactions

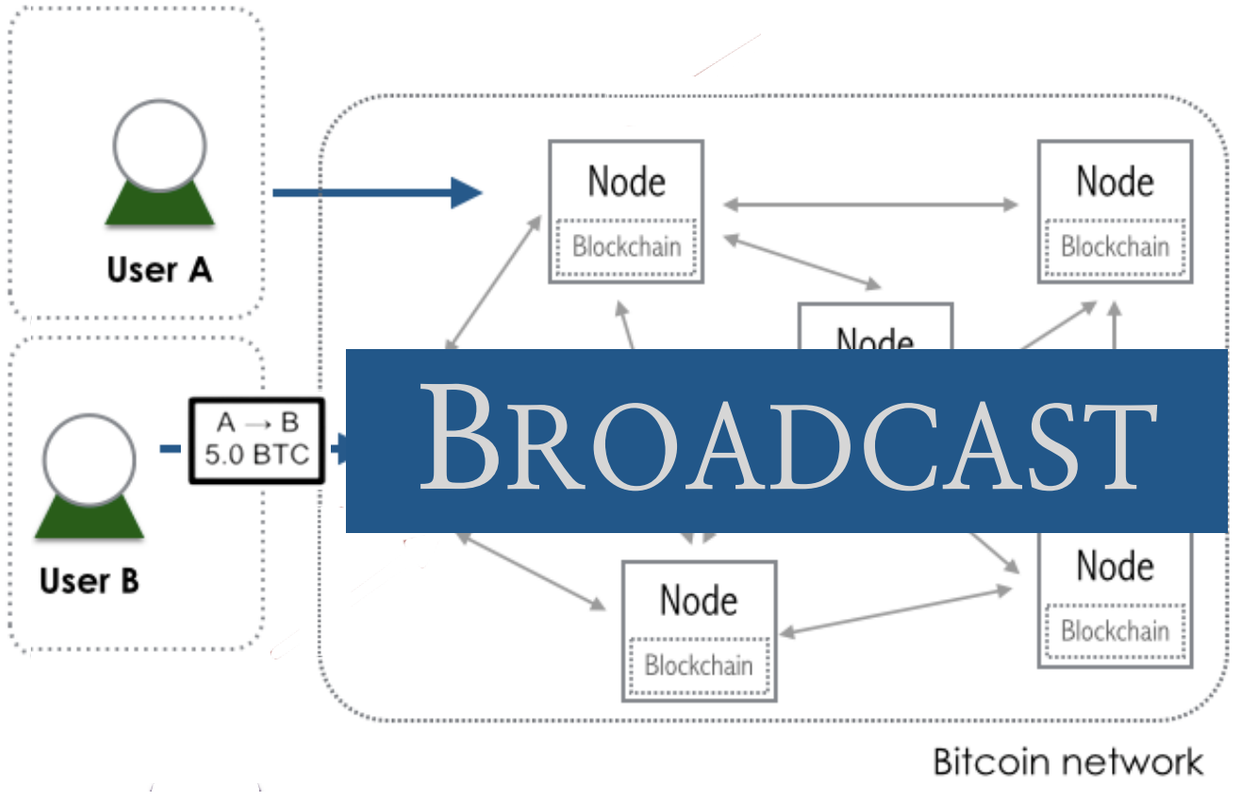


High-level interactions



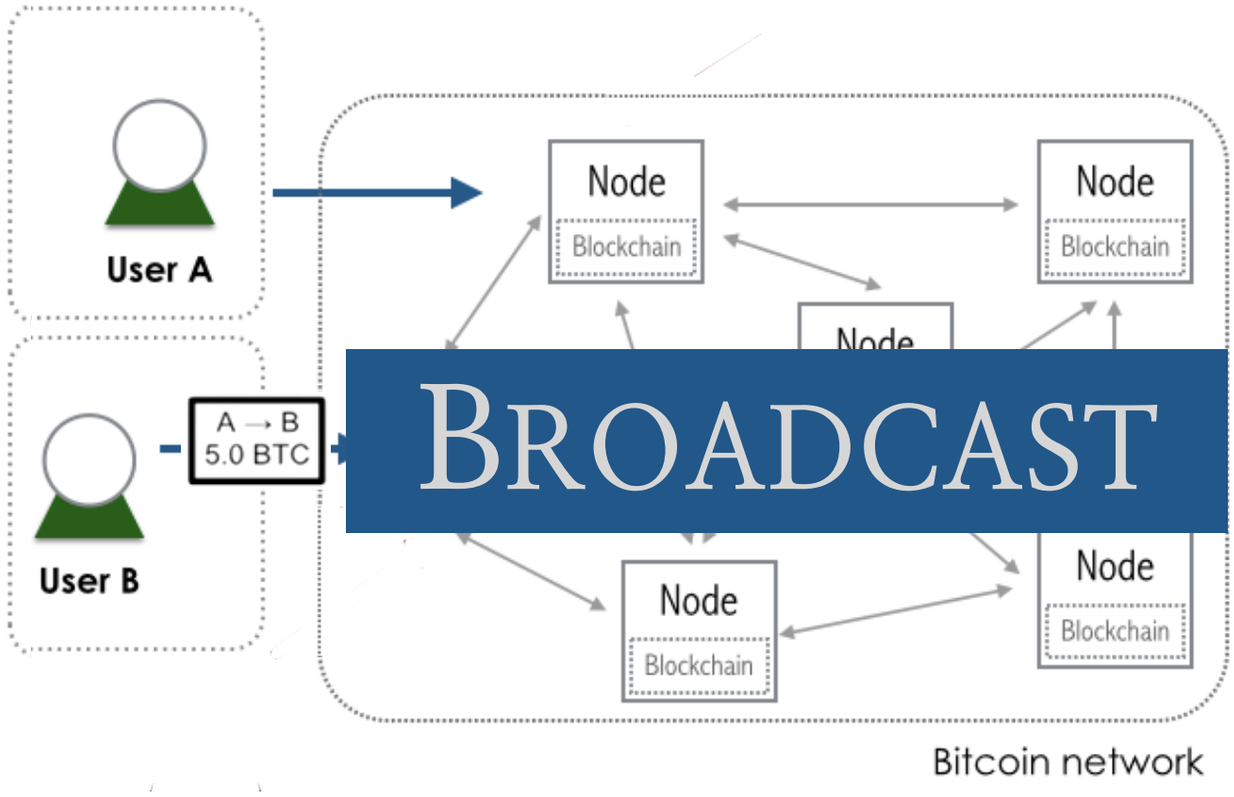
Recognise any abstractions?

High-level interactions



What kind of broadcast?

High-level interactions

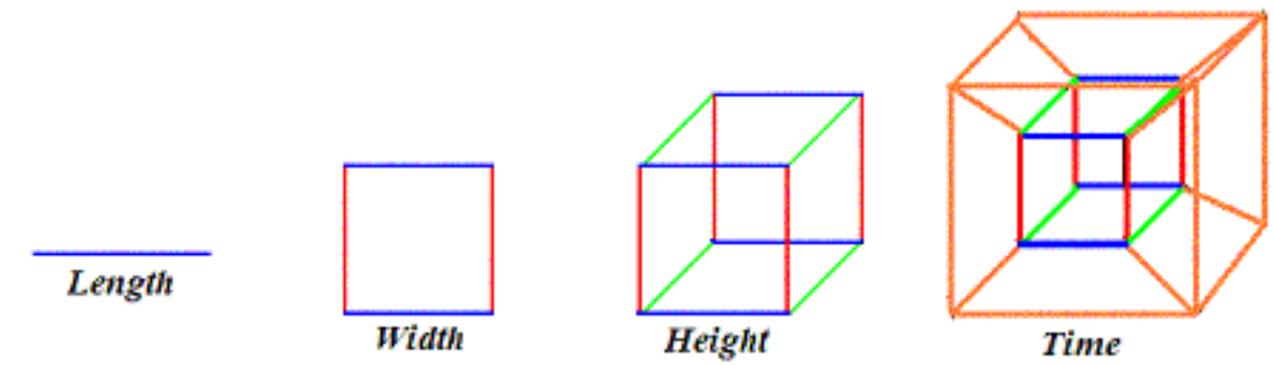


What kind of broadcast?

PROPERTIES (guarantees)

PERFORMANCE

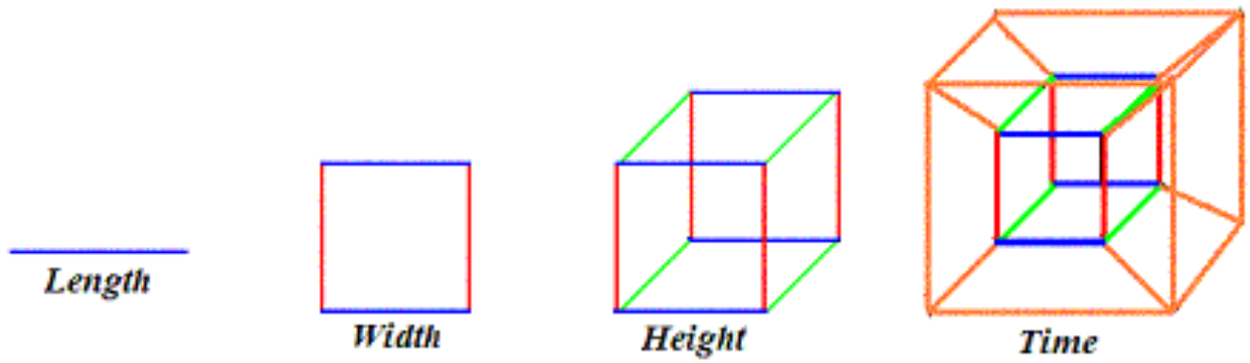
PROPERTIES



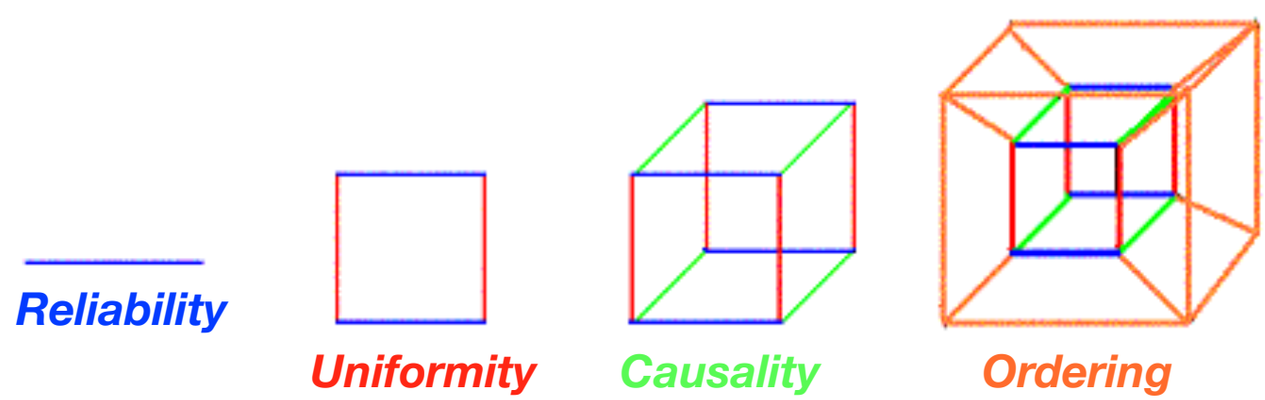
PROPERTIES

Analogy
(not formal)

The Four Dimensions *of a cube*

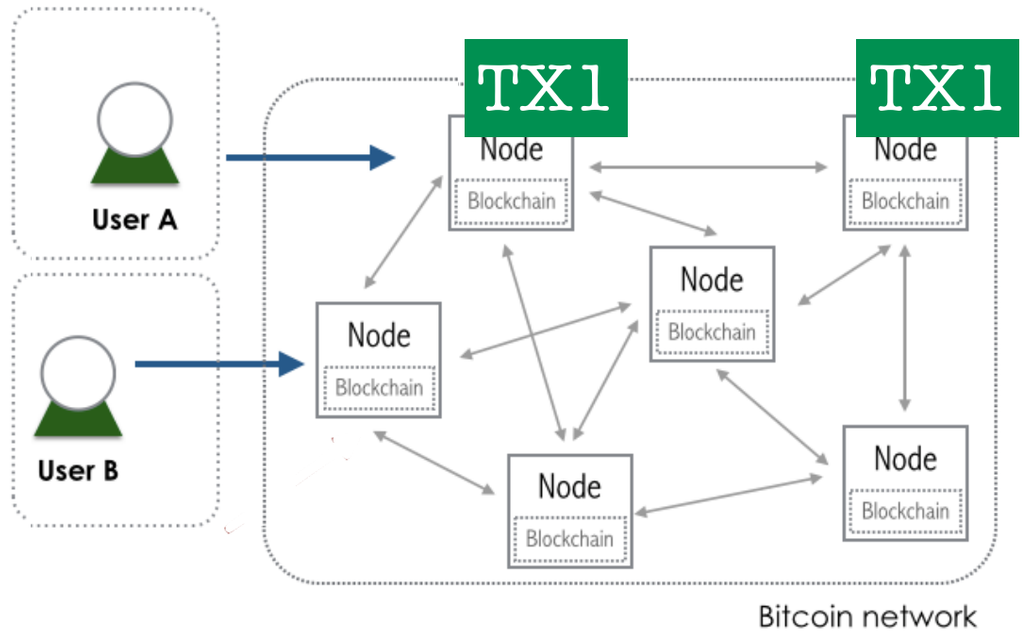


The Four Dimensions *of broadcast*



PROPERTIES

Do we need Reliability?

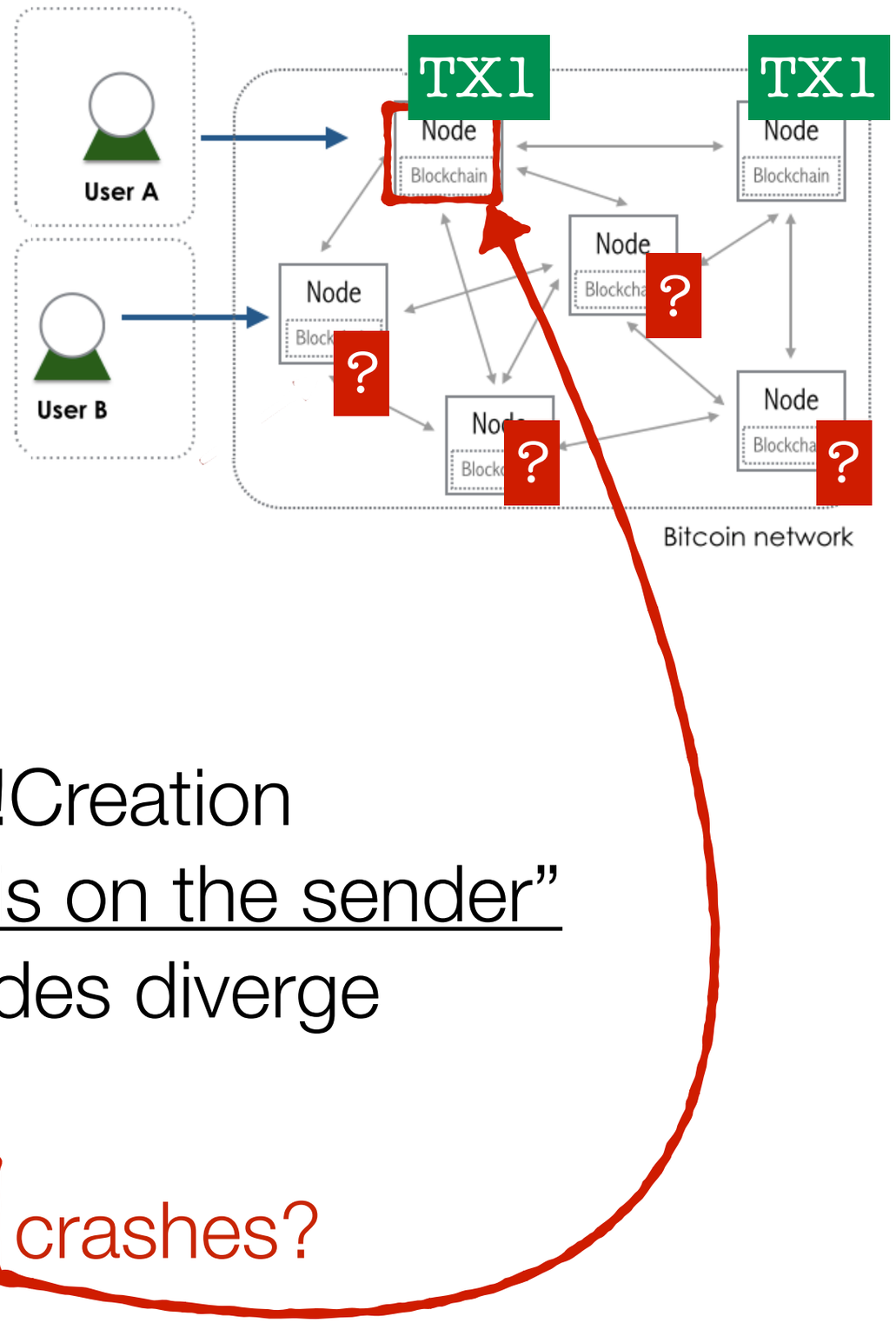


- Consider the following:
  - User A starts `TX1`
  - Use best-effort broadcast
    - Validity + !Duplication + !Creation
    - “the burden of reliability is on the sender”
    - Lacks Agreement ⇒ Nodes diverge

PROPERTIES

Do we need Reliability?

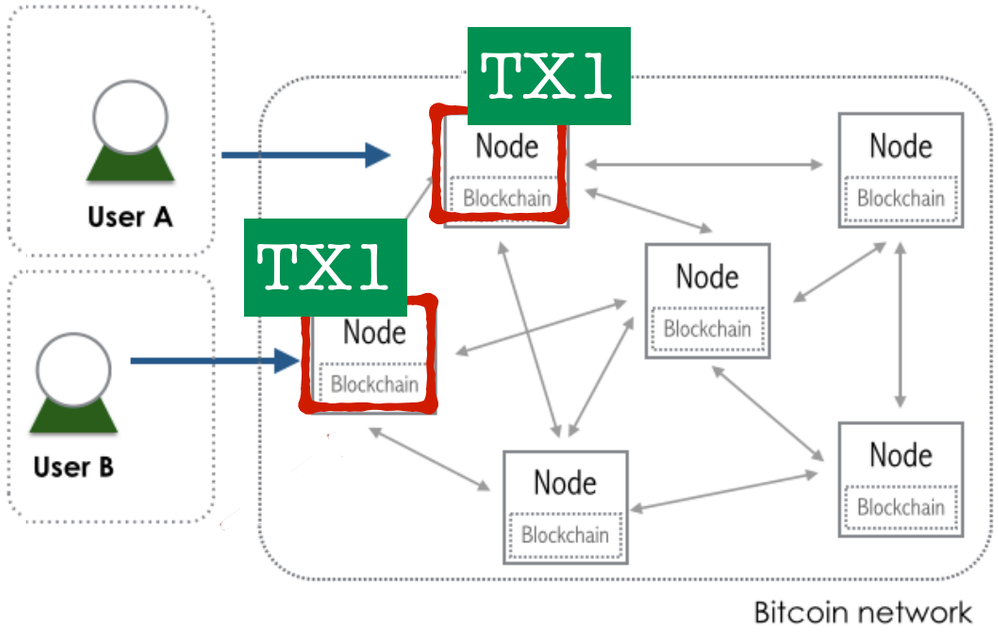
- Consider the following:
  - User A starts `TX1`
  - Use best-effort broadcast
    - Validity + !Duplication + !Creation
    - “the burden of reliability is on the sender”
    - Lacks Agreement ⇒ Nodes diverge



PROPERTIES

Do we need Uniformity?

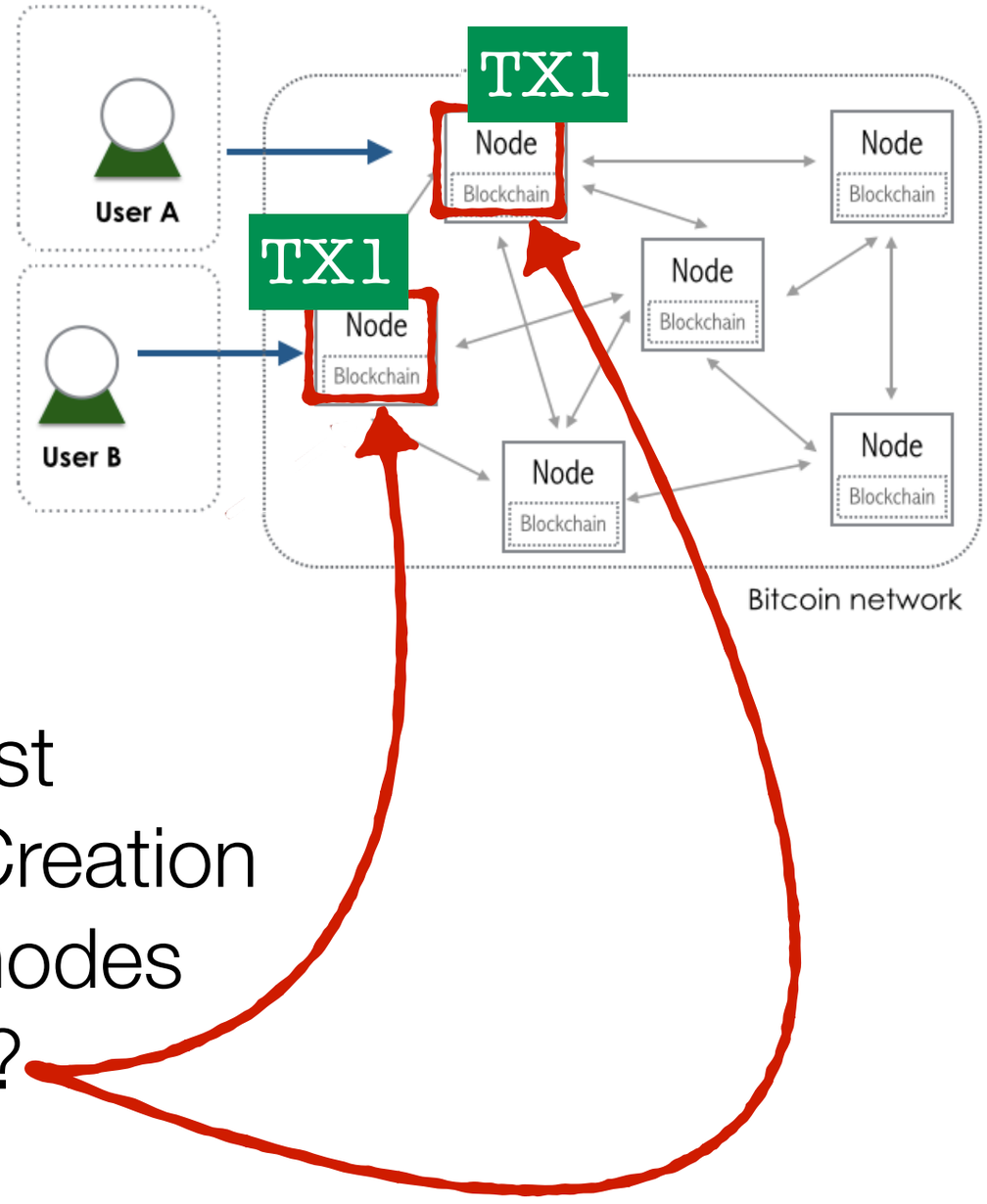
- Consider the following:
  - User A starts `TX1`
  - Use regular reliable broadcast
    - Validity + !Duplication + !Creation + Agreement for correct nodes
  - Is it OK to deliver and crash?



PROPERTIES

Do we need Uniformity?

- Consider the following:
  - User A starts `TX1`
  - Use regular reliable broadcast
    - Validity + !Duplication + !Creation + Agreement for correct nodes
  - Is it OK to deliver and crash?

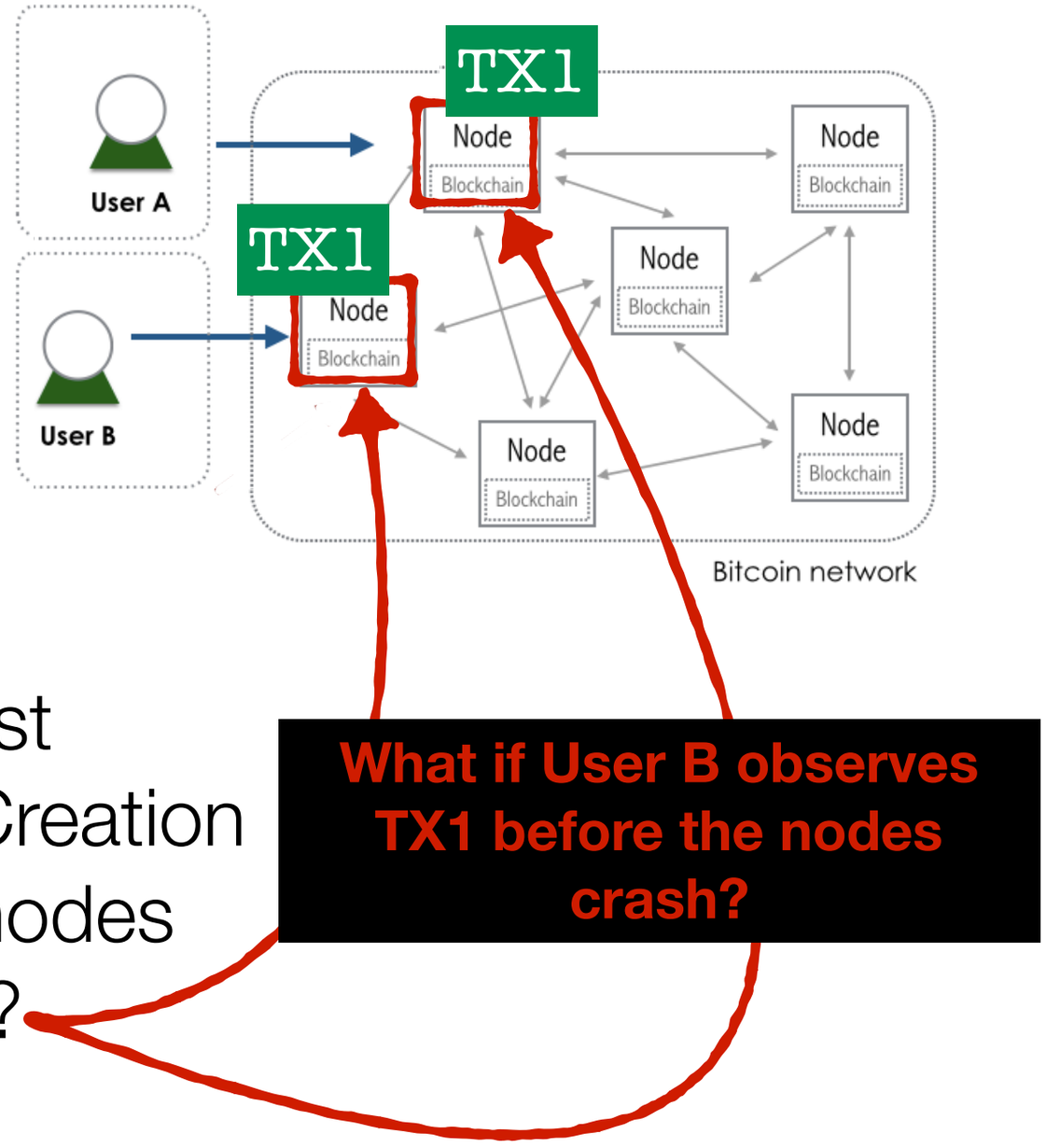


"Uniformity is important if nodes interact with the external world"

PROPERTIES

Do we need Uniformity?

- Consider the following:
  - User A starts `TX1`
  - Use regular reliable broadcast
    - Validity + !Duplication + !Creation + Agreement for correct nodes
  - Is it OK to deliver and crash?

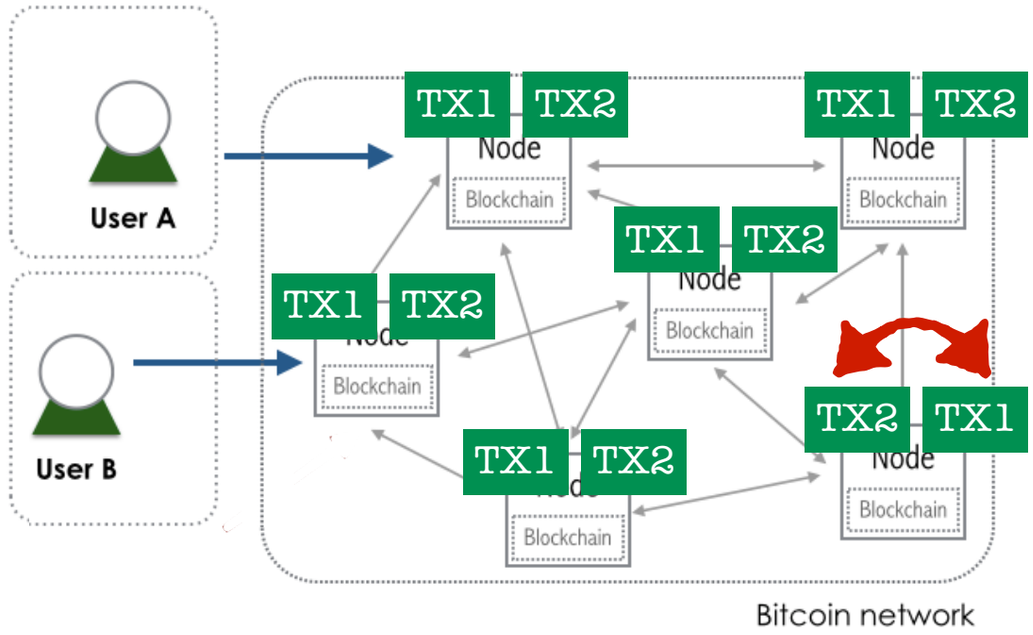


"Uniformity is important if nodes interact with the external world"

PROPERTIES

Do we need Causality? (partial ordering)

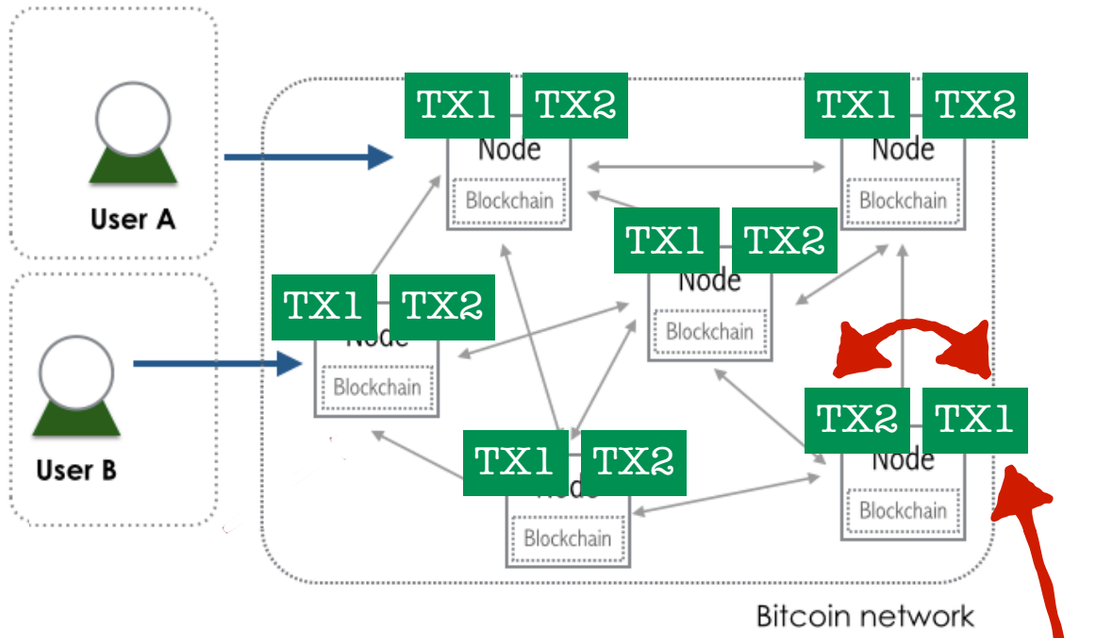
- Consider the following:
  - User A starts `TX1` and `TX2`
  - Use uniform reliable broadcast
    - Validity + !Duplication + !Creation + Uniform Agreement
      ⇒ Applies to all nodes
  - All nodes deliver both TX, but the order may differ



PROPERTIES

Do we need Causality? (partial ordering)

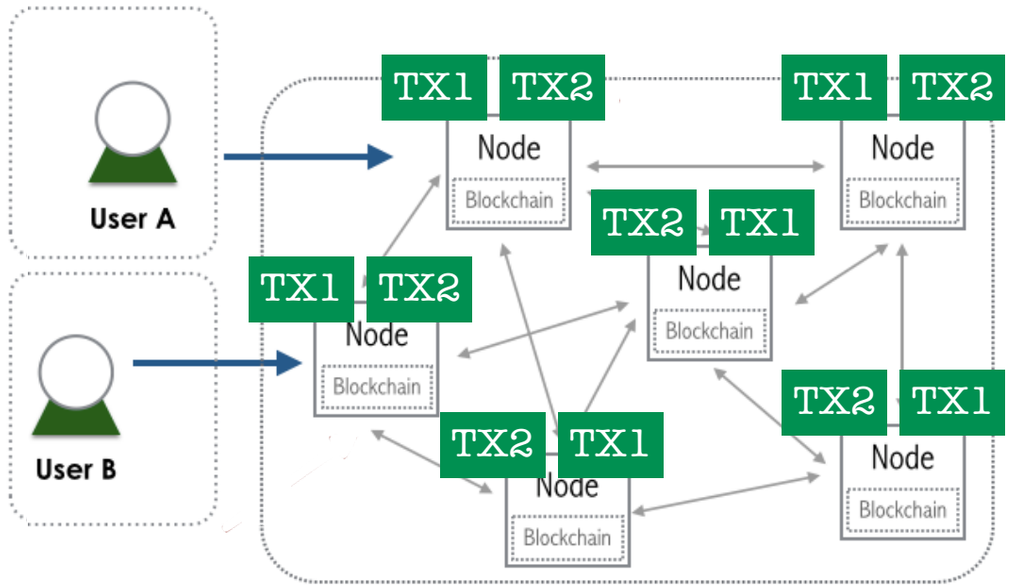
- Consider the following:
  - User A starts `TX1` and `TX2`
  - Use uniform reliable broadcast
    - Validity + !Duplication + !Creation + Uniform Agreement
      ⇒ Applies to all nodes
  - All nodes deliver both TX, but the order may differ



What if `TX2` depends on money from `TX1` ?

PROPERTIES

Do we need (Total) Ordering?

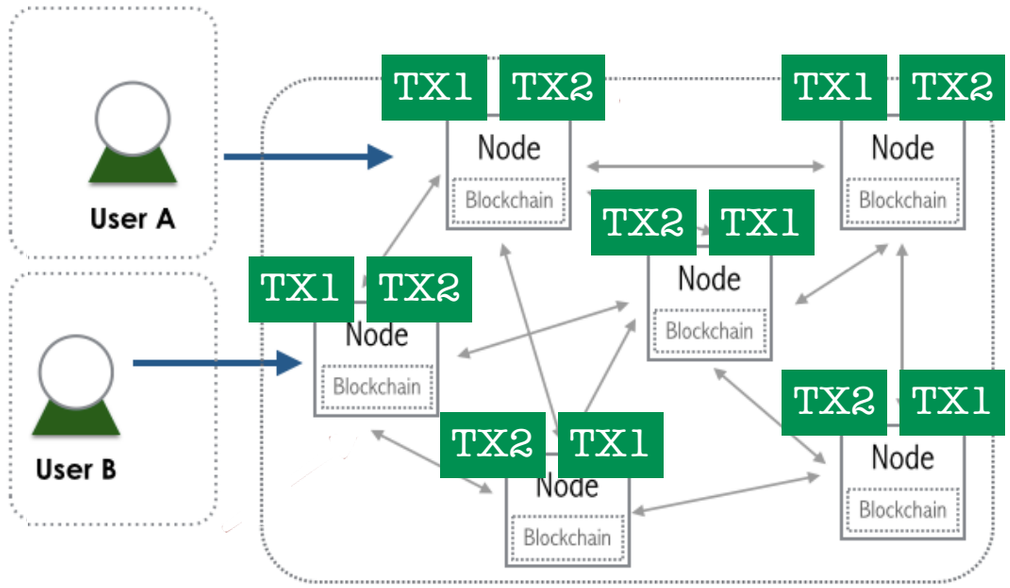


- Consider the following:
  - User A starts TX1
  - User B starts TX2

  No dependency among these two
  - Use causal-order uniform reliable broadcast
    - Validity + !Duplication + !Creation + Uniform Agreement + Causality
      ⇒ Respect causal dependencies among TXs
  - All nodes deliver both TX, but the order may differ

PROPERTIES

Do we need (Total) Ordering?

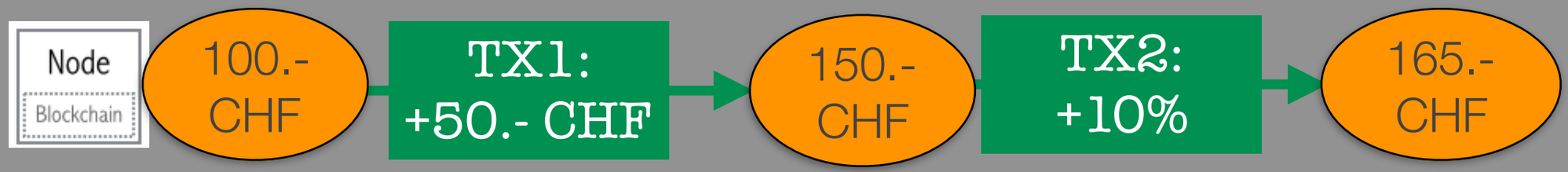


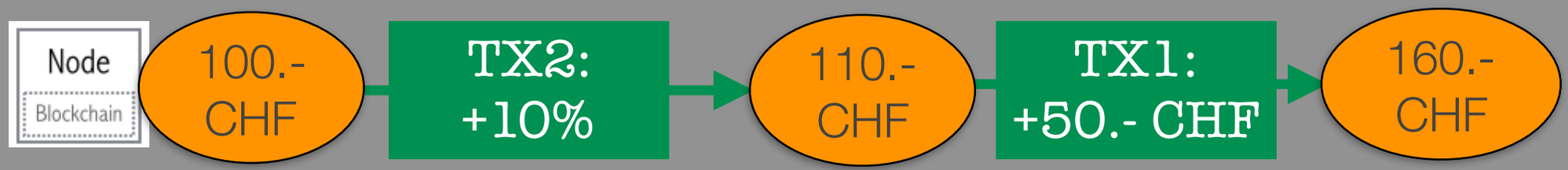
- Consider the following:
  - User A starts TX1
  - User B starts TX2

  No dependency among these two
  - Use causal-order uniform reliable broadcast
    - Validity + !Duplication + !Creation + Uniform Agreement + Causality
      ⇒ Respect causal dependencies among TXs
  - All nodes deliver both TX, but the order may differ

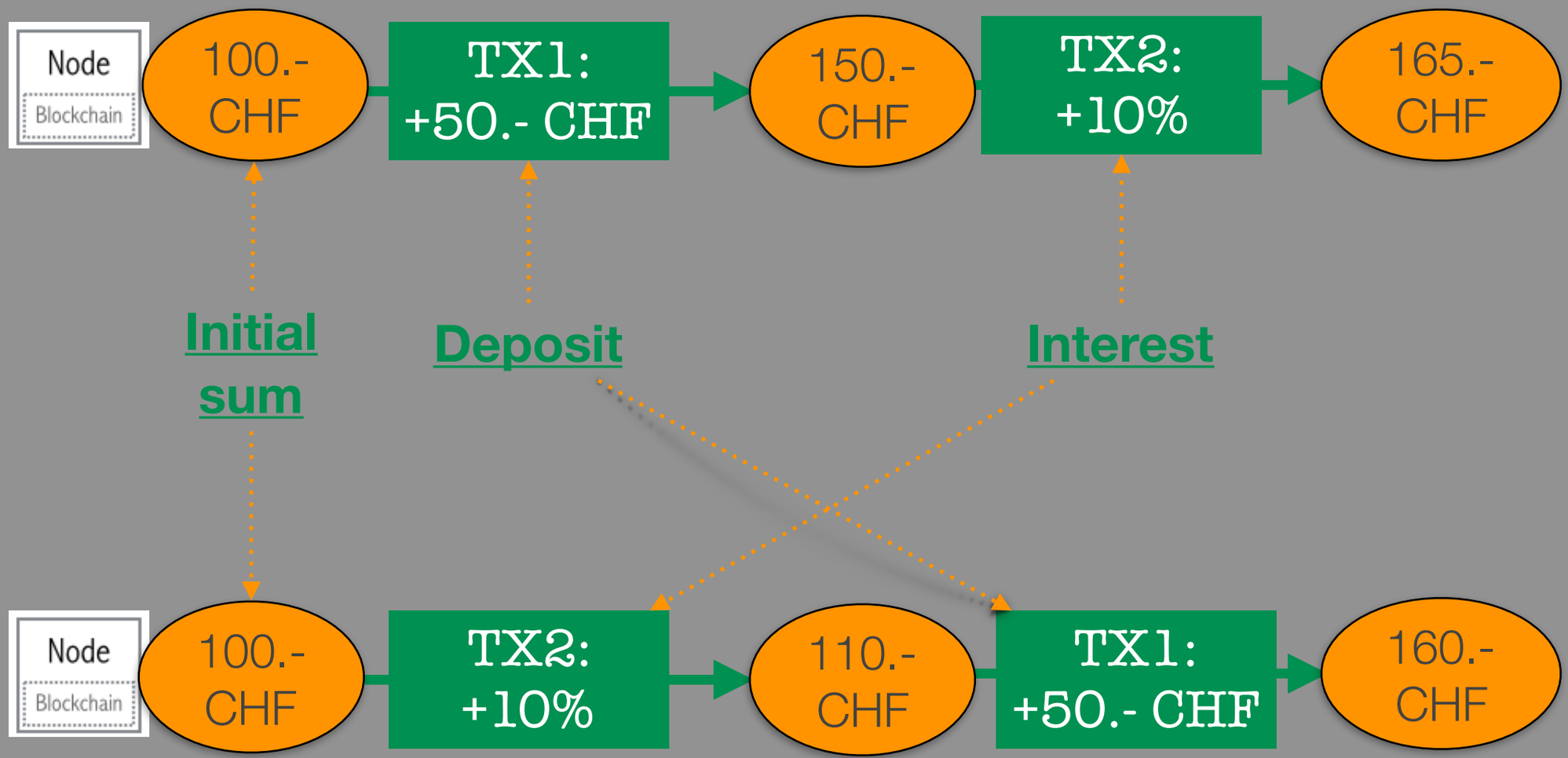
What if TX1 and TX2 are not commutative?

Commutativity
counter-example

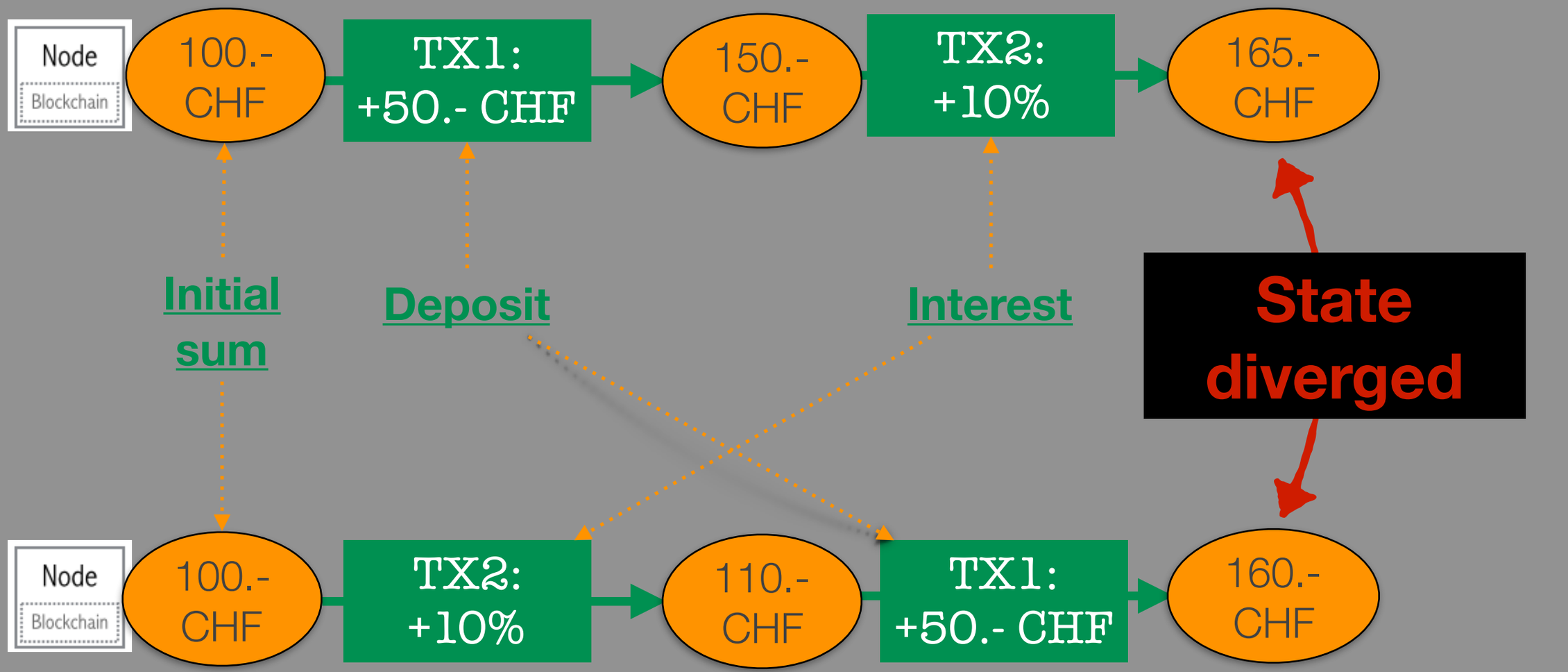




Commutativity
counter-example

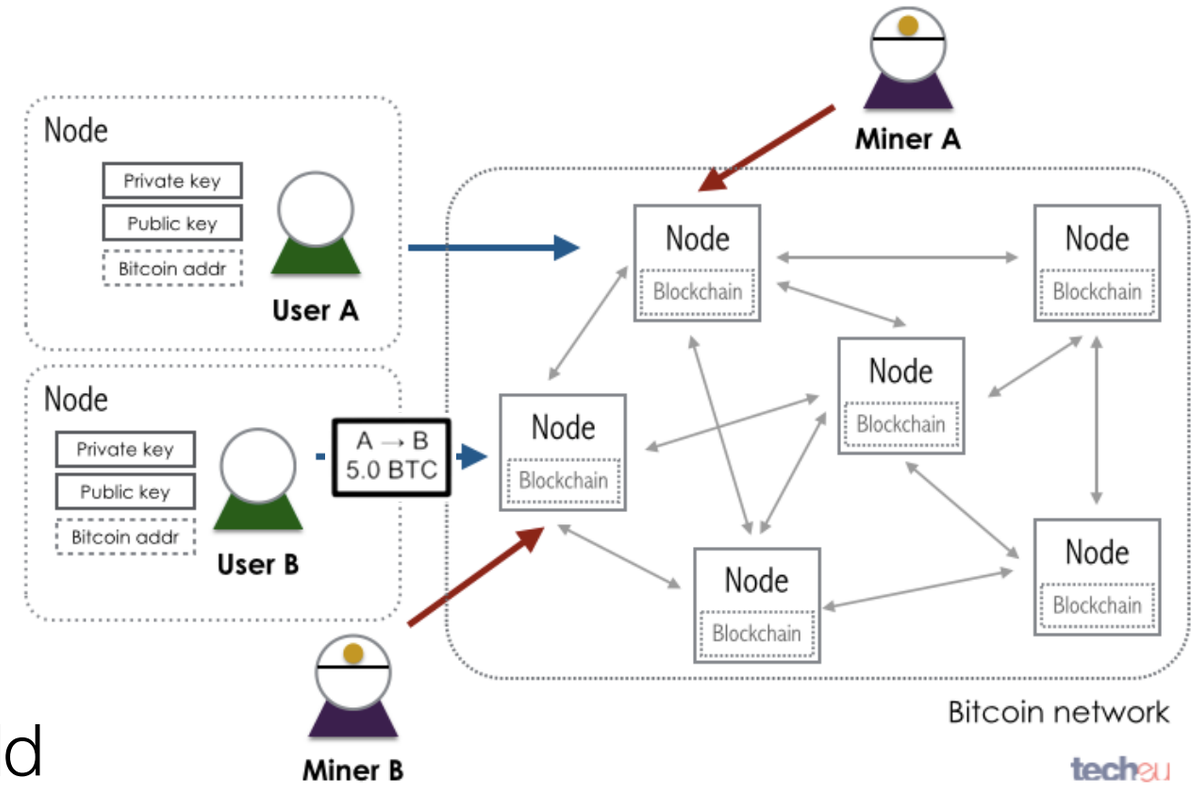


Commutativity
counter-example



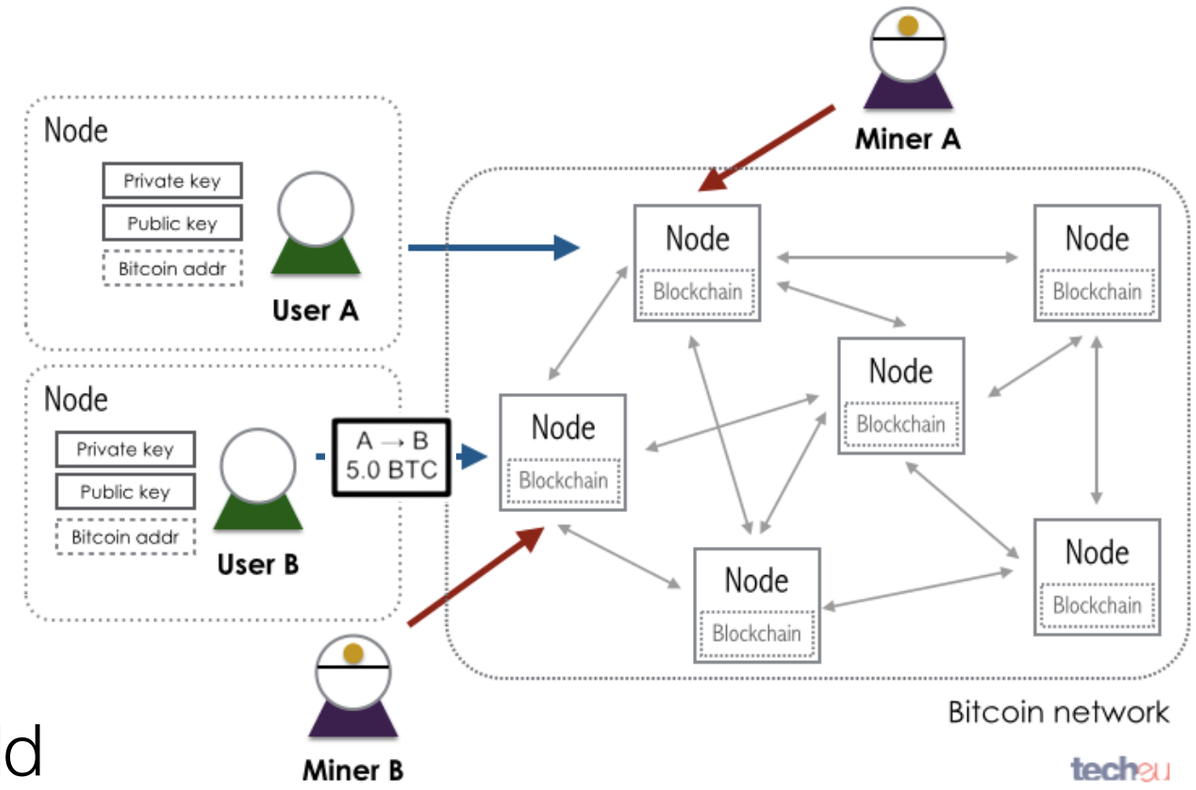
PROPERTIES

- Reliability
  - Sender crashes
  - Agreement
- Uniformity
  - Again, crashes
  - Interaction with outside world
- Causality
  - Partial order
  - Dependencies among TXs
- Total order
  - Commutativity



PROPERTIES

- Reliability
  - Sender crashes
  - Agreement
- Uniformity
  - Again, crashes
  - Interaction with outside world
- Causality
  - Partial order
  - Dependencies among TXs
- Total order
  - Commutativity

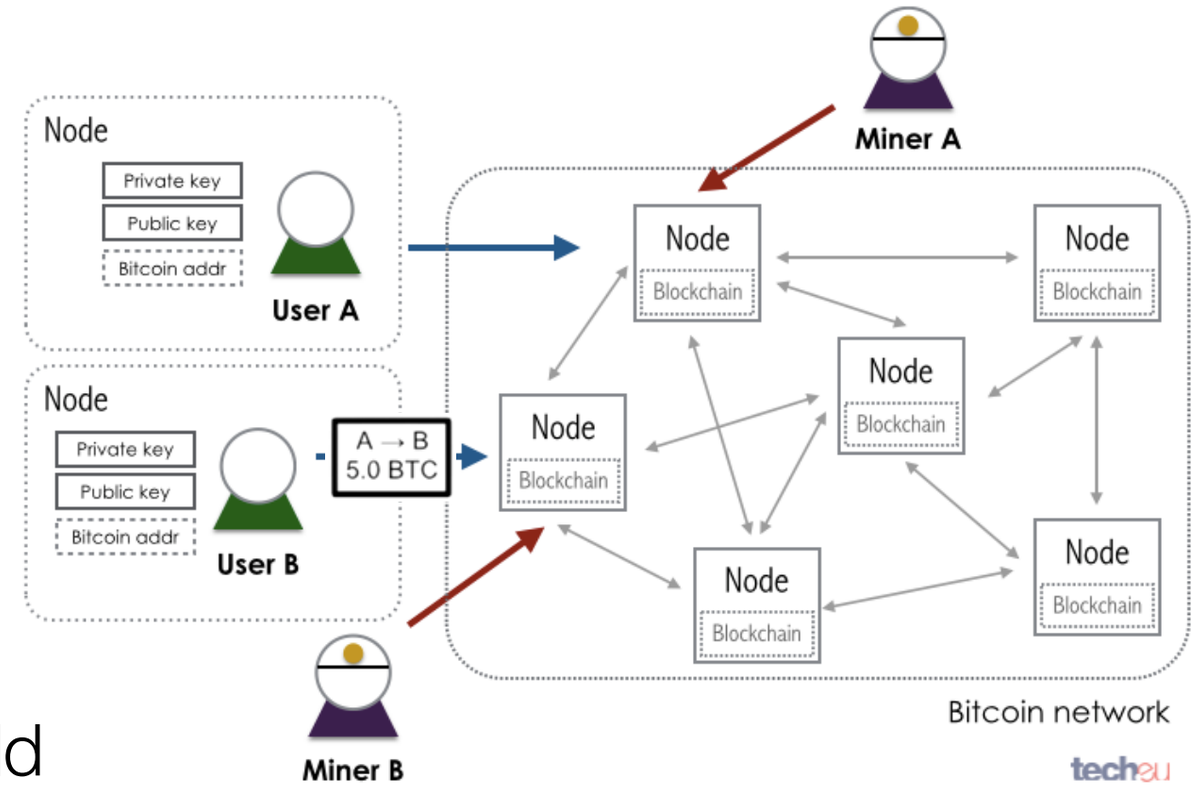


Contrast:

- Canonical Bitcoin requires best-effort broadcast
- Allows temp. inconsistencies
- Makes up with crypto

PROPERTIES

- Reliability
  - Sender crashes
  - Agreement
- Uniformity
  - Again, crashes
  - Interaction with outside world
- Causality
  - Partial order
  - Dependencies among TXs
- Total order
  - Commutativity



Contrast:

- Canonical Bitcoin requires best-effort broadcast
- Allows temp. inconsistencies
- Makes up with crypto

All properties are desirable

PERFORMANCE

Goals:

1. Replace traditional CAMIPRO
2. Make banks obsolete

AN INFORMAL COMPARISON…

PERFORMANCE

Goals:

1. Replace traditional CAMIPRO
2. Make banks obsolete

AN INFORMAL COMPARISON…

CANONICAL BITCOIN

- Best-effort broadcast
- Optimized for Internet (WAN)
- **TX~10 minutes**

How useful would that be?
(in the lunch queue)

PERFORMANCE

Goals:

1. Replace traditional CAMIPRO
2. Make banks obsolete

AN INFORMAL COMPARISON…

CANONICAL BITCOIN

- Best-effort broadcast
- Optimized for Internet (WAN)
- **TX~10 minutes**

How useful would that be?
(in the lunch queue)

CAMIPRO BITCOIN

- CAMIPRO-Bitcoin-Broadcast
- Optimized for small network (EPFL LAN)
- TX~1second (back of the envelope)
- Acceptable latency & throughput

Specification

Specification

Module:

Name: CAMIPRO-Bitcoin-Broadcast, **instance** *cbit*

Properties:

- **URB1, URB2, URB3, URB4**
- **Causal Order (CO)**
- **Total order (TO)**

What's left ????

Specification

Module:

Name: CAMIPRO-Bitcoin-Broadcast, **instance** *cbit*

Properties:

- **URB1, URB2, URB3, URB4**
- **Causal Order (CO)**
- **Total order (TO)**

Events:

Request: ⟨*cbit, Start* | TX⟩:
Attempts to commit TX

Indication: ⟨*cbit, Status* | TX, s⟩:
Indicates the status s∈{“Abort”,“Commit”} of TX

Specification

Module:

Name: CAMIPRO-Bitcoin-Broadcast, **instance** *cbit*

Properties:

- **URB1, URB2, URB3, URB4**
- **Causal Order (CO)**
- **Total order (TO)**

CAMIPRO-Bitcoin

Request **Indication**

CAMIPRO-Bitcoin-Broadcast

Events:

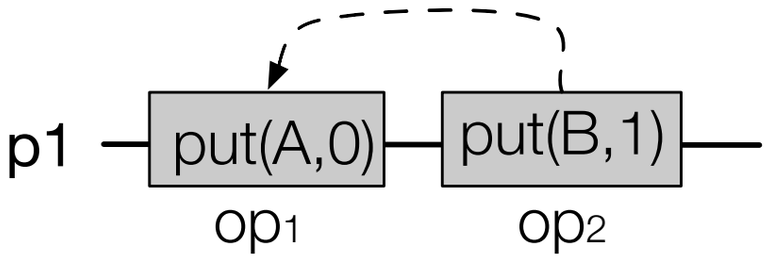
Request: ⟨*cbit, Start* | TX⟩:
Attempts to commit TX

Indication: ⟨*cbit, Status* | TX, s⟩:
Indicates the status s∈{"Abort","Commit"} of TX

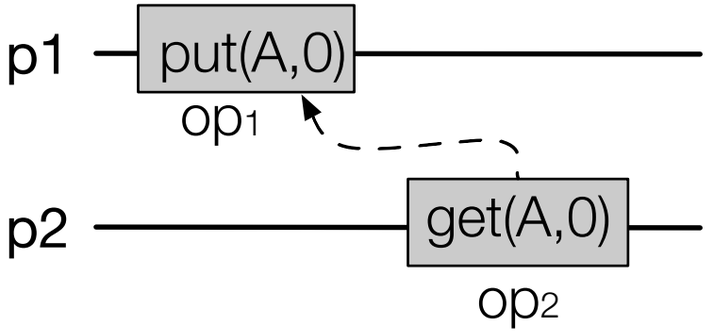
Causal consistency — one last look

- Given two operations, op1 and op2

- op1 → op2 (causally precedes)
  if any of the following three cases applies:

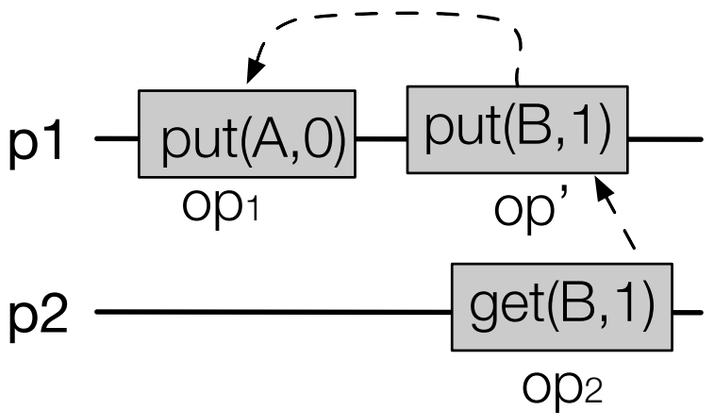


(a) FIFO: A process invokes op1 and then invokes op2

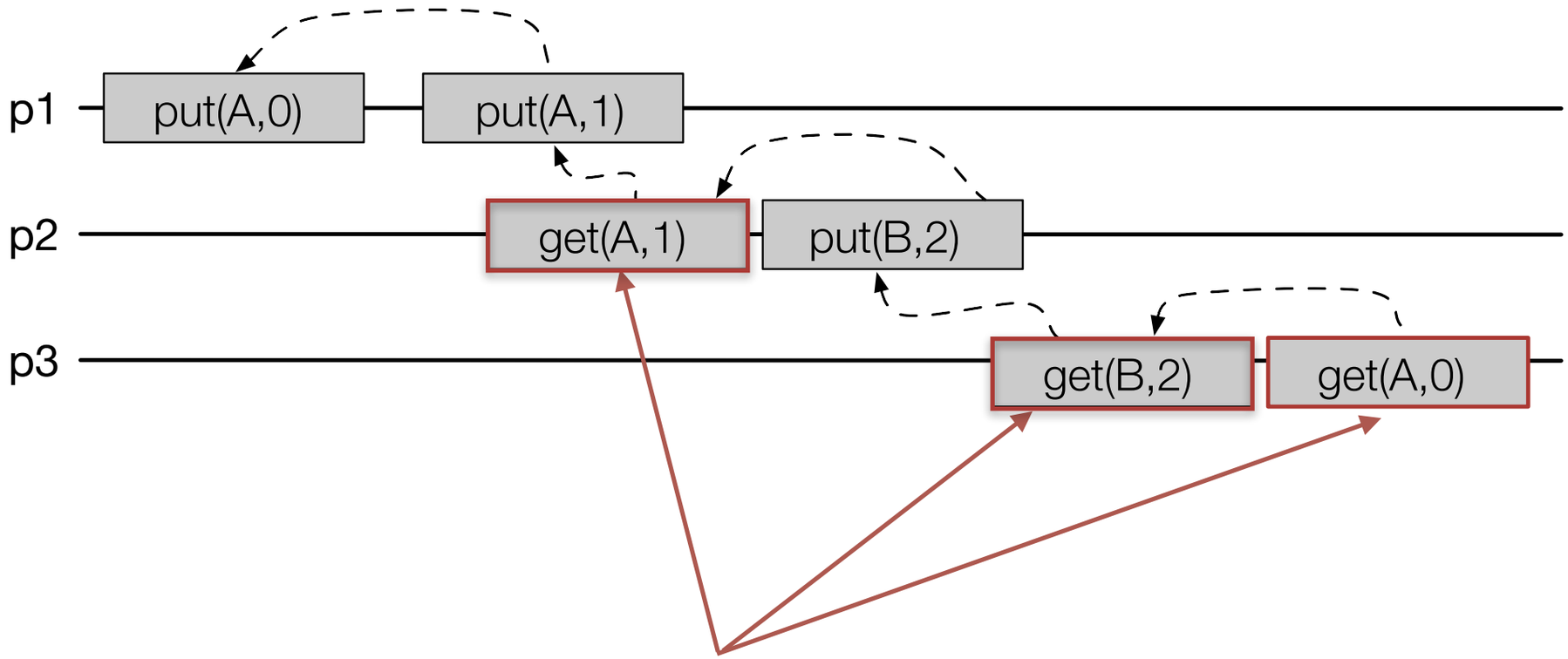


(b) Local: A process invokes the PUT operation op1 and another process invokes the GET operation op2, where op2 observes the written value of op1

(c) Transitivity: There exists an intermediate operation op′ such that op1 → op′ and op′ → op2.

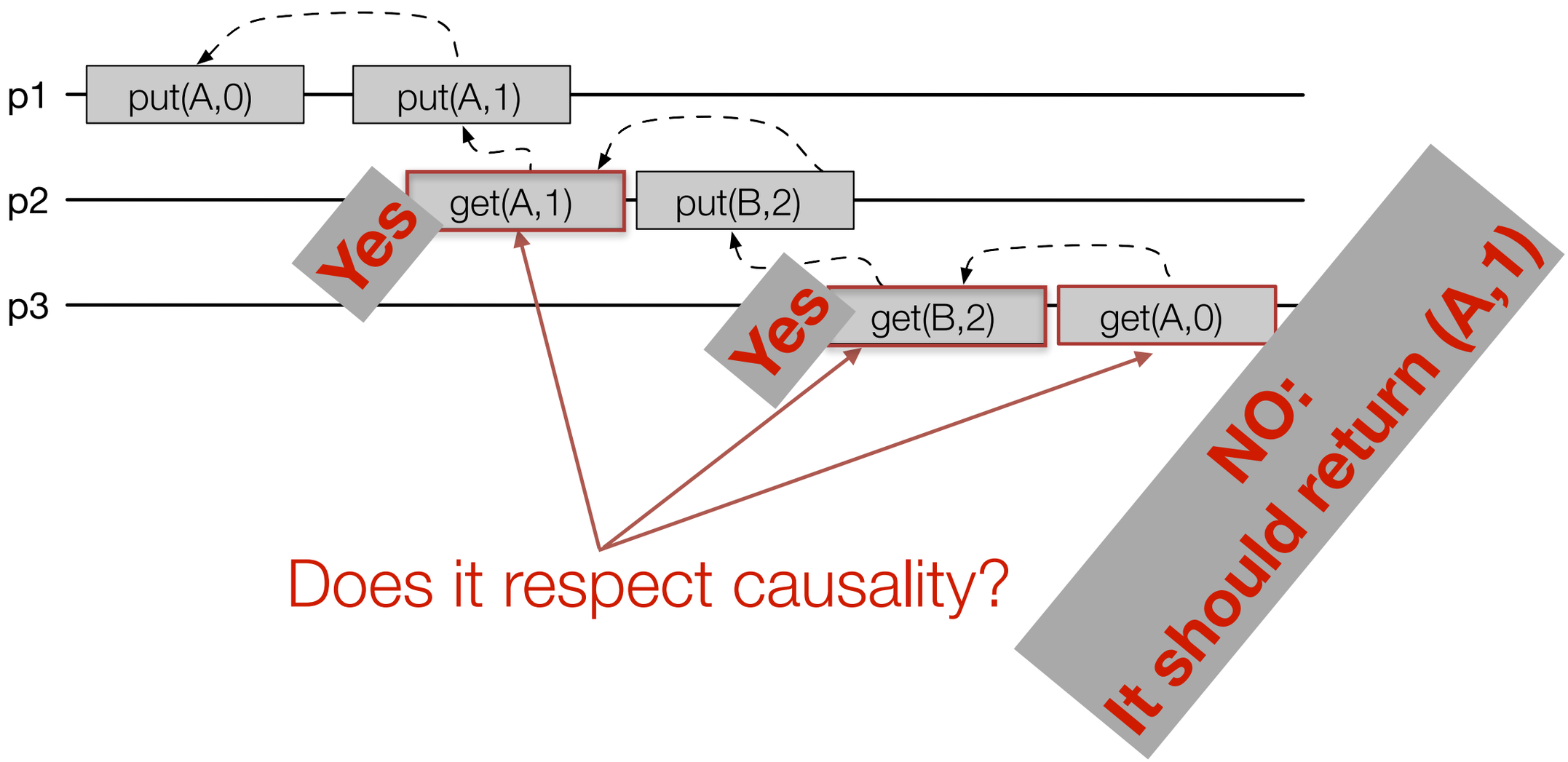


Causal consistency — one last look



Does it respect causality?

Causal consistency — one last look



Does it respect causality?