ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Applications for Broadcast

[**Reliable**, **Uniform**, **Causal**, and **Total-Order**]

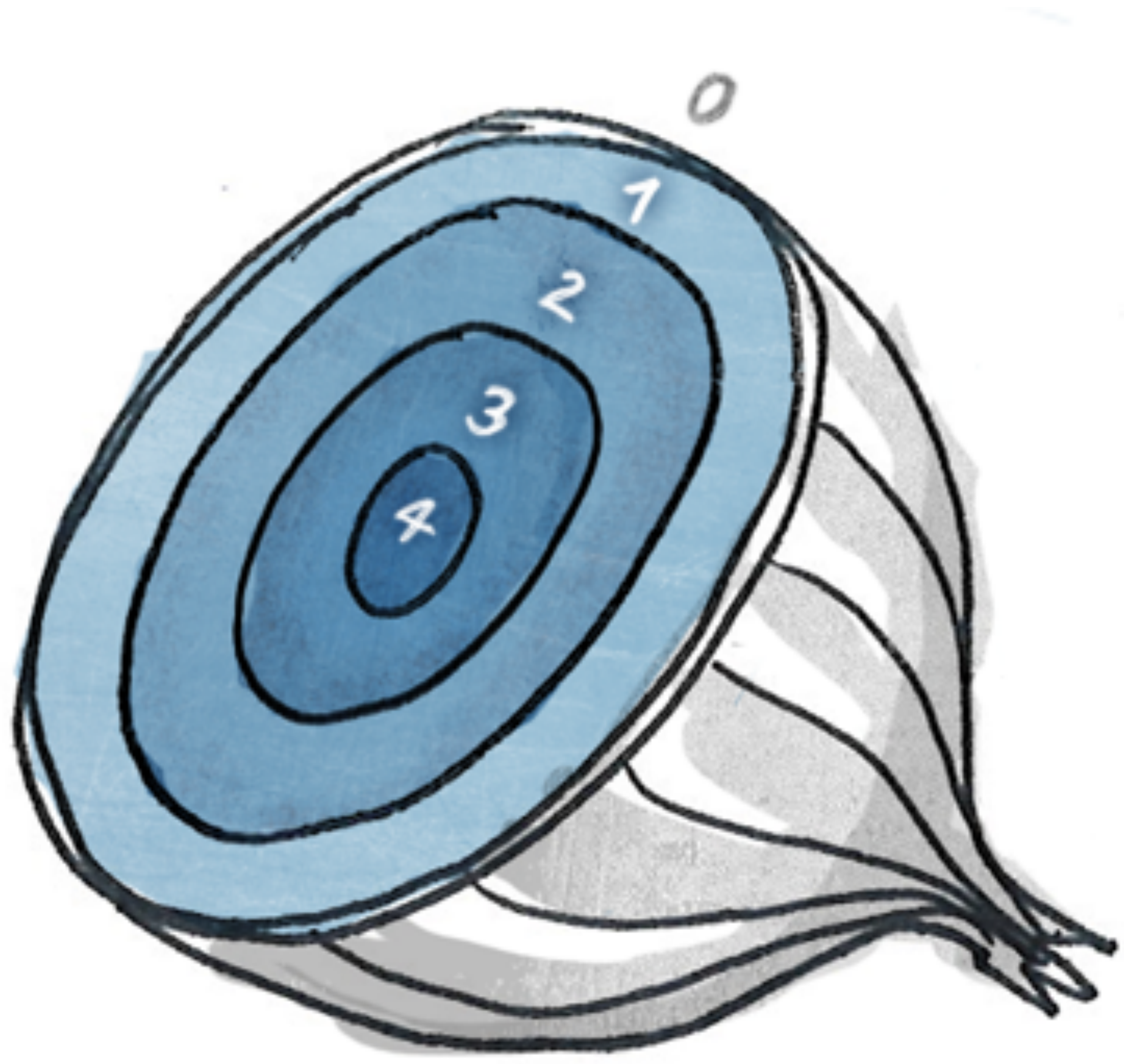
Adi Seredinschi
Distributed Programming Laboratory

MIDTERM

23/11/2015

- ★ Lectures
  - — including this one
- ★ Exercises
  - — class (+ optional book)

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

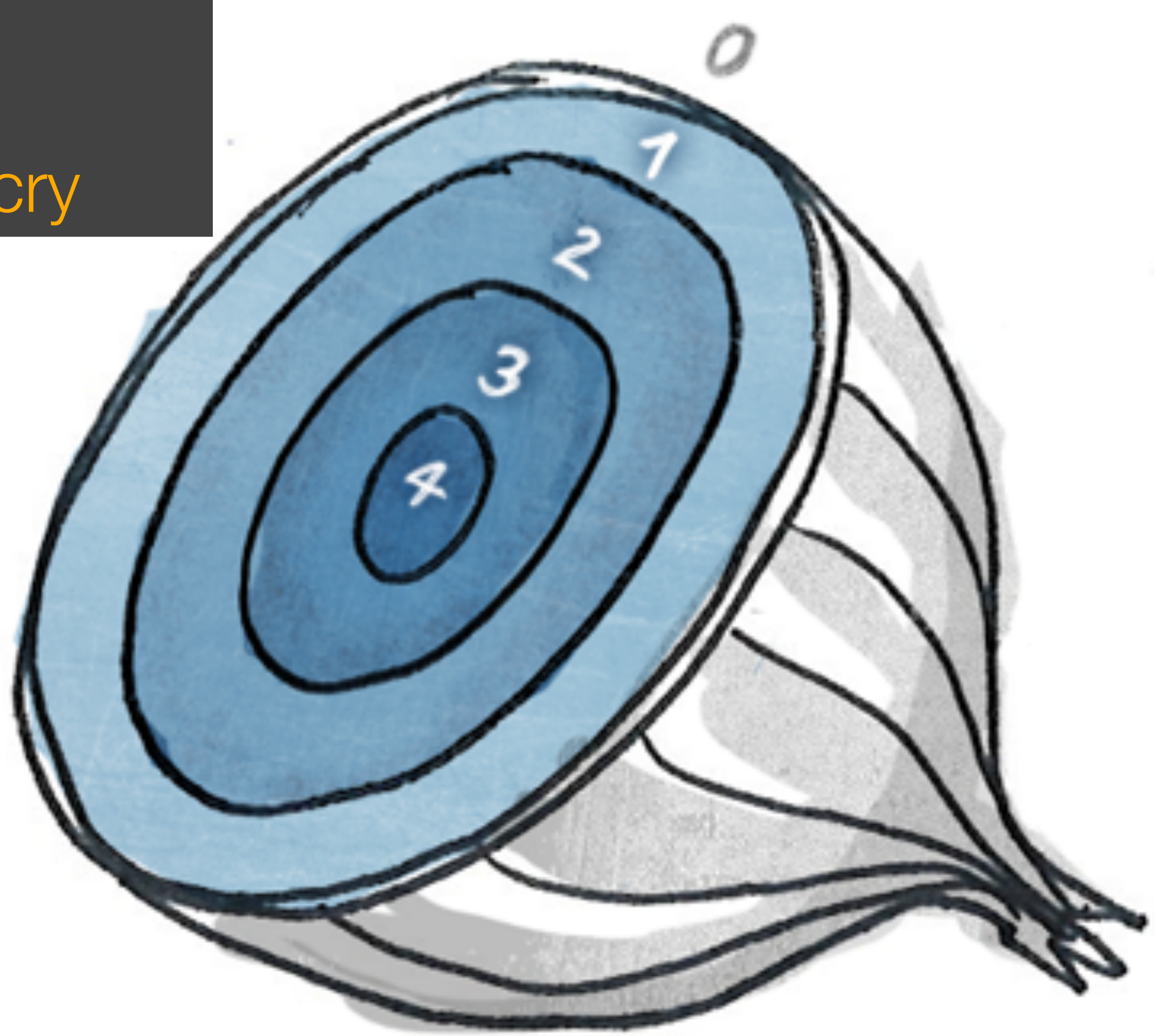


ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

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

Answer:

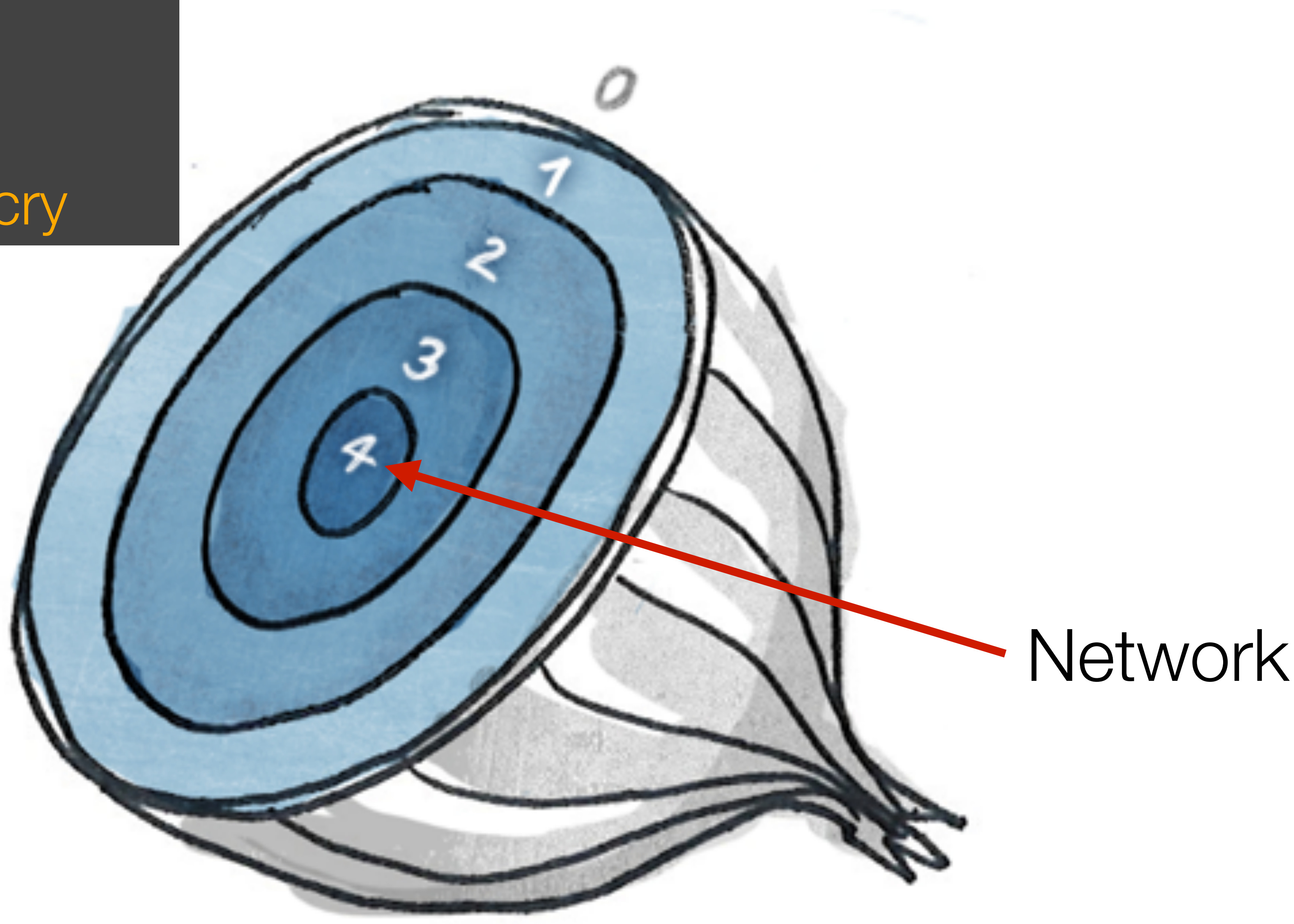
1. Layering
2. Abstraction
3. They make you cry



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

Answer:

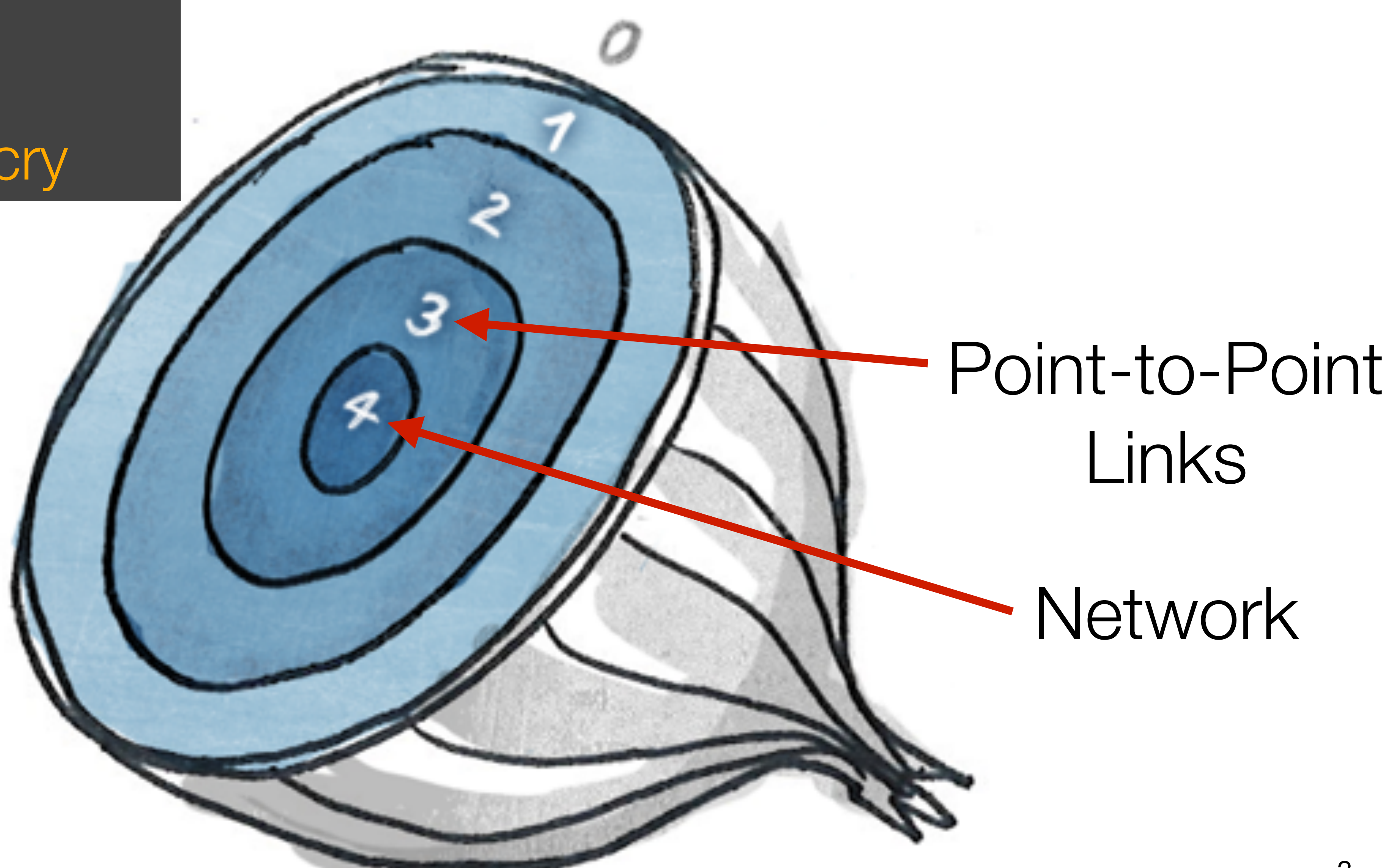
1. Layering
2. Abstraction
3. They make you cry



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

Answer:

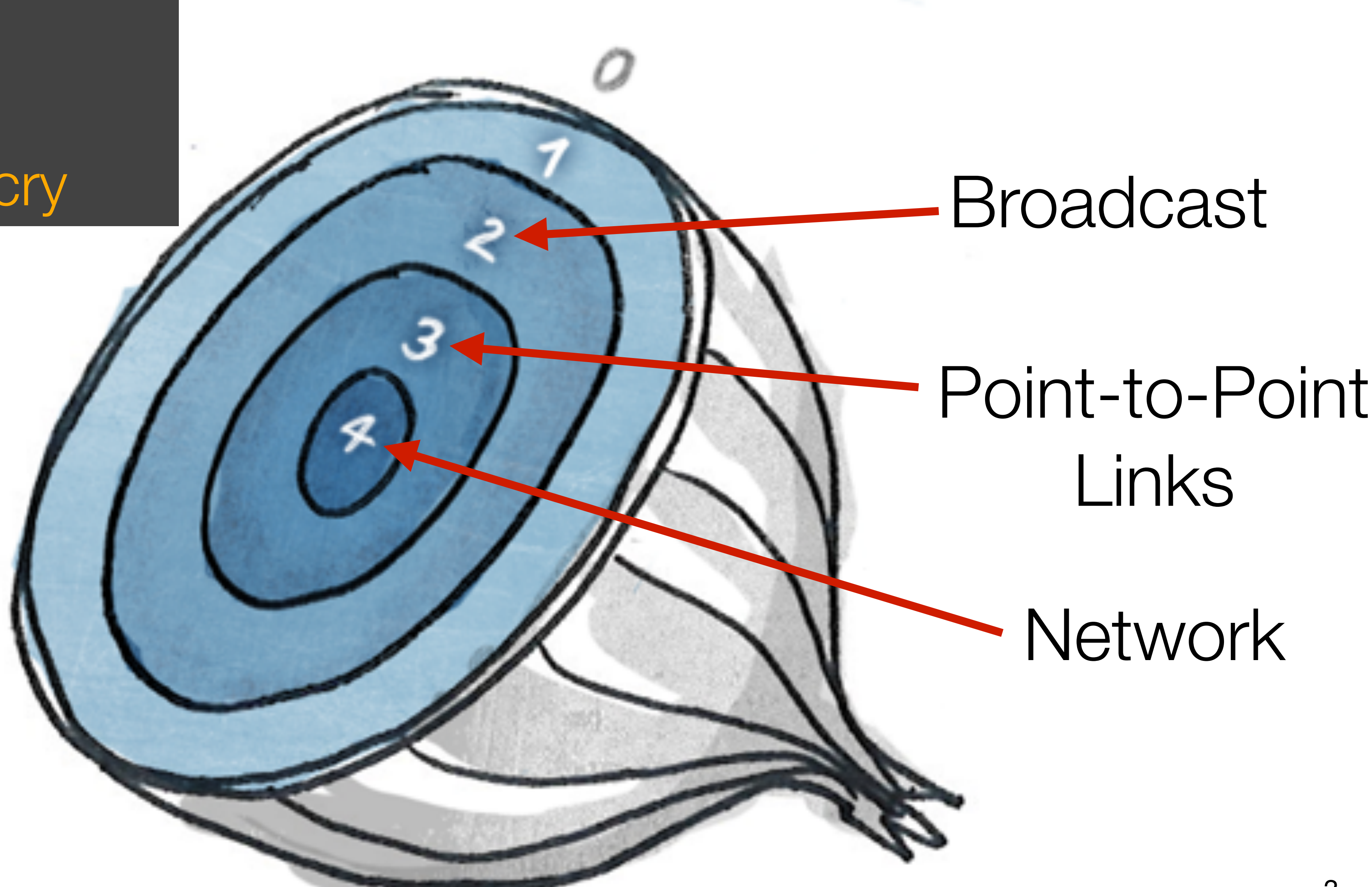
1. Layering
2. Abstraction
3. They make you cry



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

Answer:

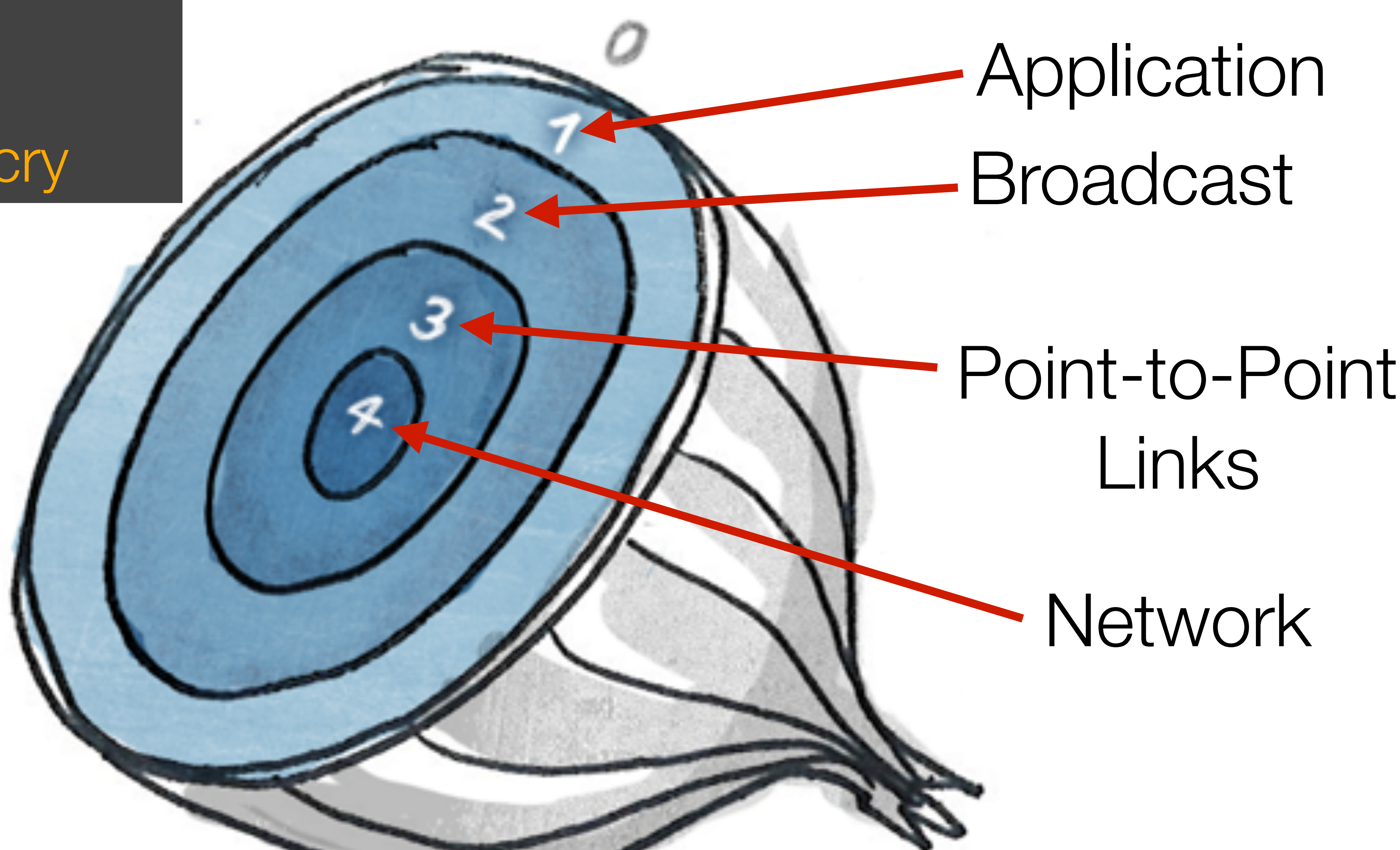
1. Layering
2. Abstraction
3. They make you cry



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

Answer:

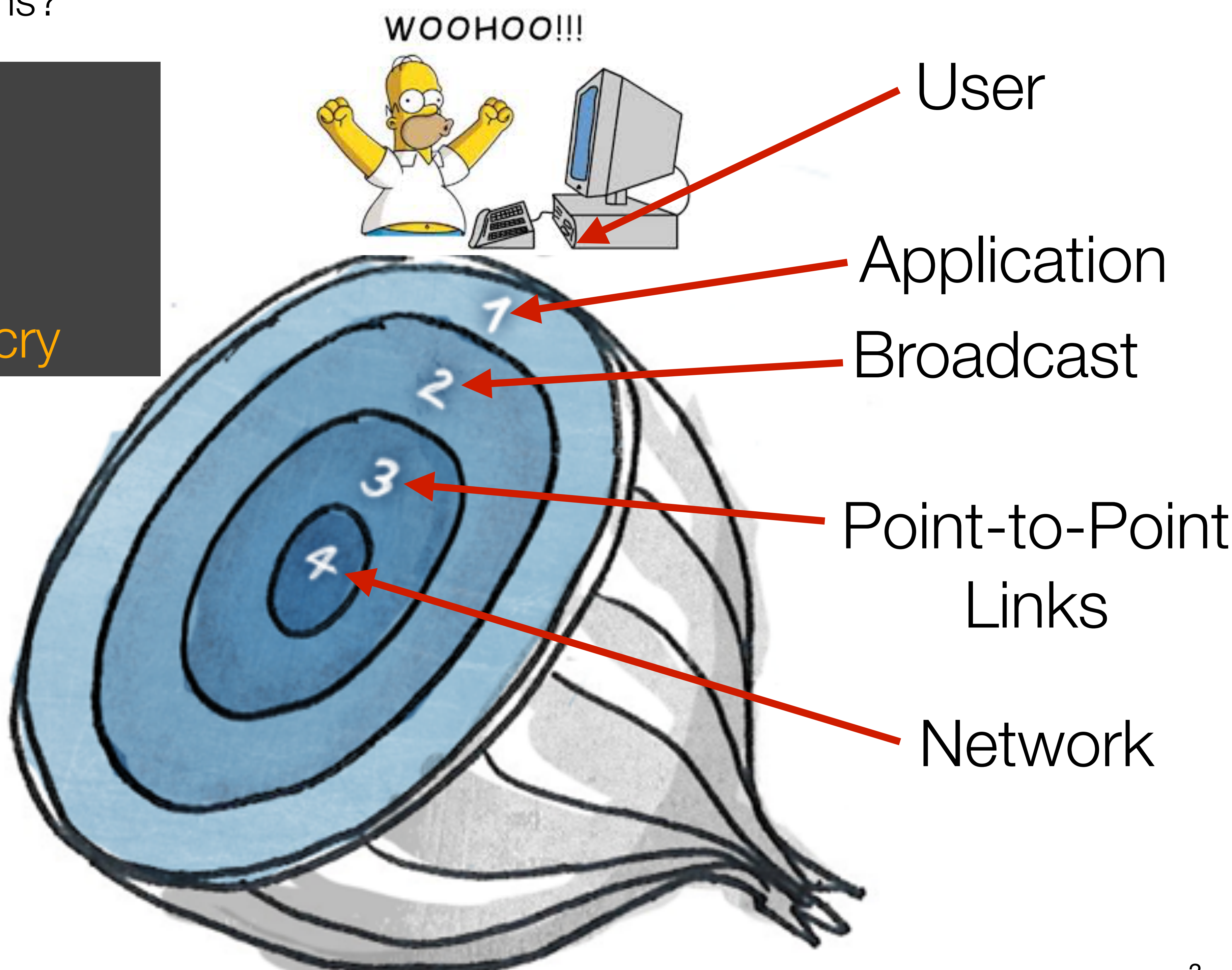
1. Layering
2. Abstraction
3. They make you cry



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

Answer:

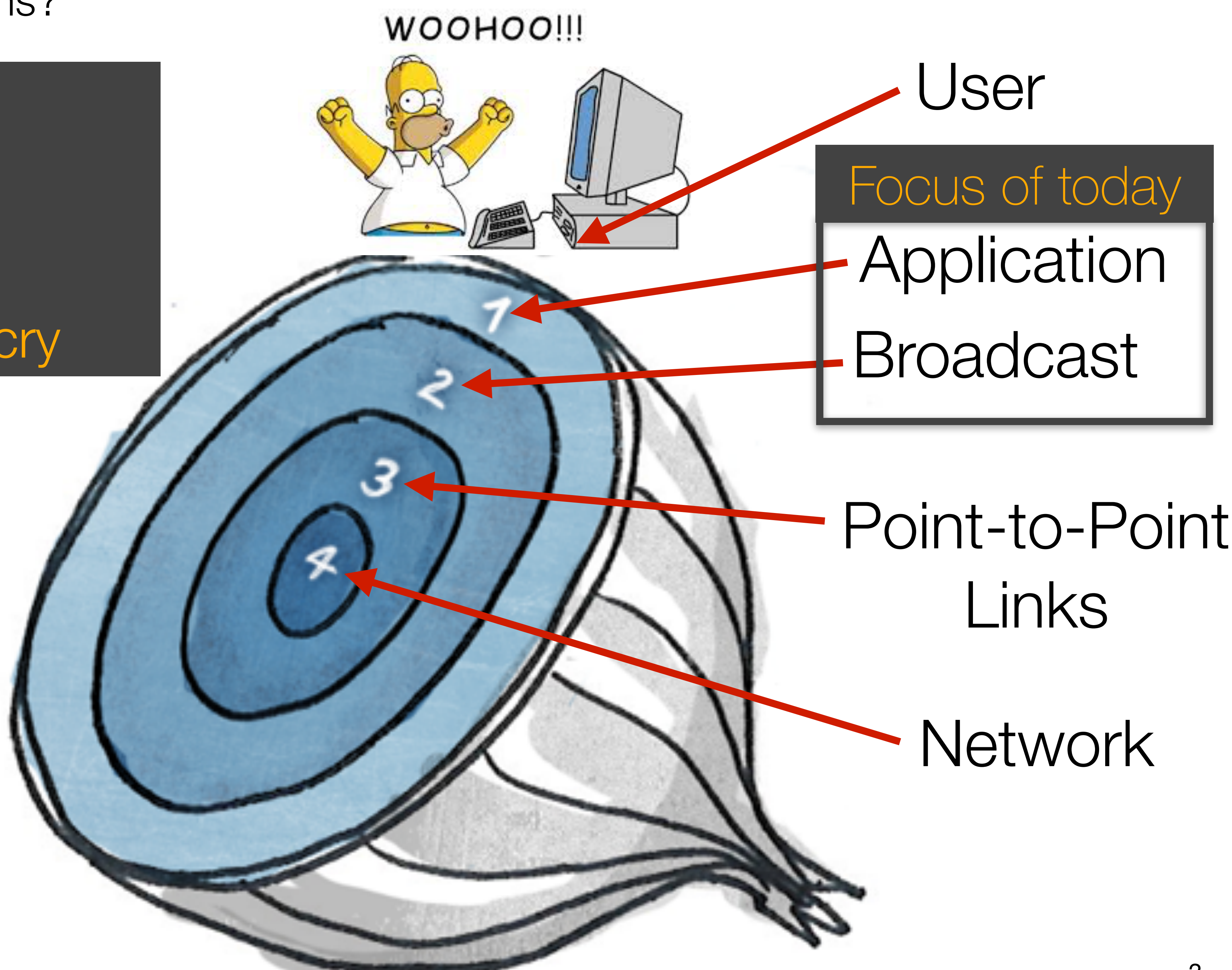
1. Layering
2. Abstraction
3. They make you cry



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

Answer:

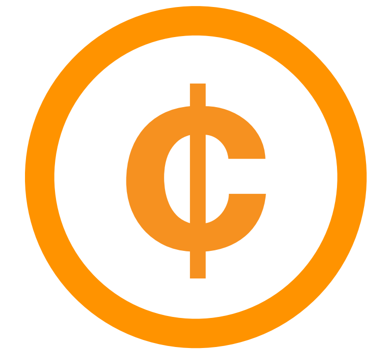
1. Layering
2. Abstraction
3. They make you cry



Let’s design some applications

Broadcast
(we will investigate different properties..)

1. **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**

Let's design some applications

Broadcast
(we will investigate different properties..)

1. **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**

Causal Broadcast

1. **S4: Storage for Social Networks**
   - A simplified version of Twitter

[BONUS PROJECT]

CAMIPRO-Bitcoin

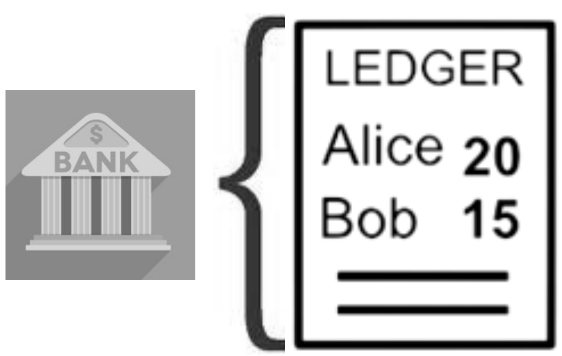
Conceptual goals:

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

CAMIPRO-Bitcoin

Conceptual goals:

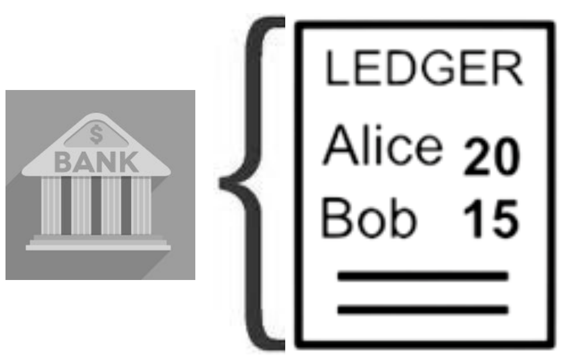
1. Replace traditional CAMIPRO
   - Based on CHF
2. Make banks obsolete



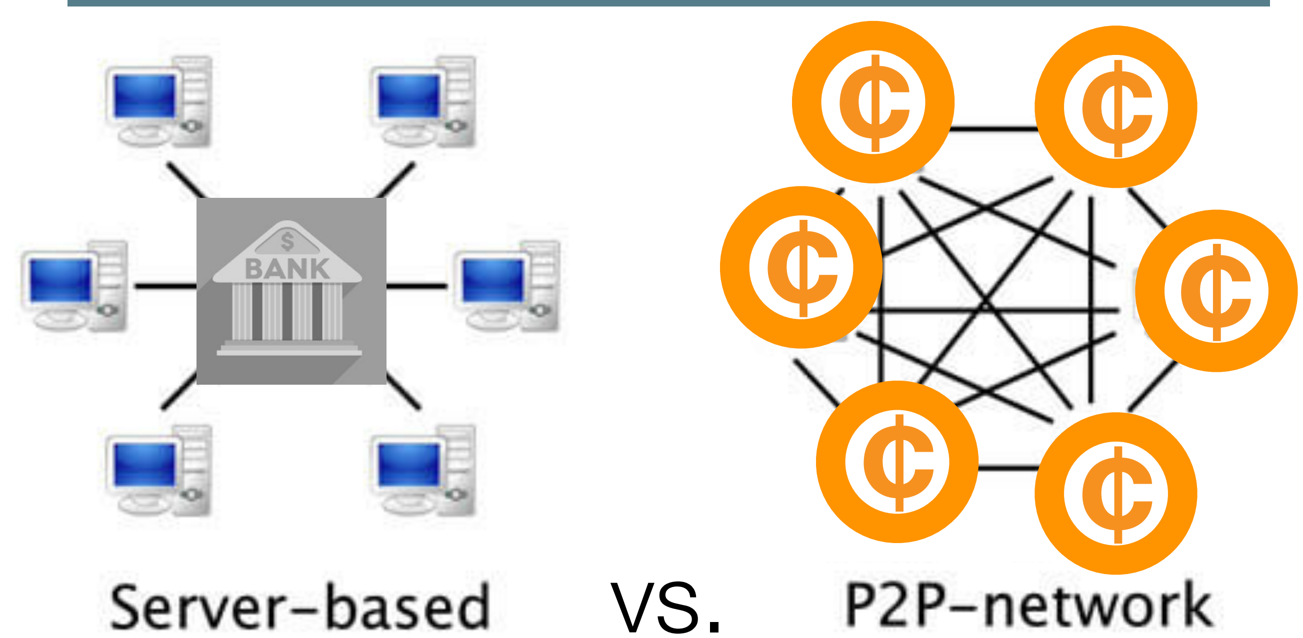
CAMIPRO-Bitcoin

Conceptual goals:

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

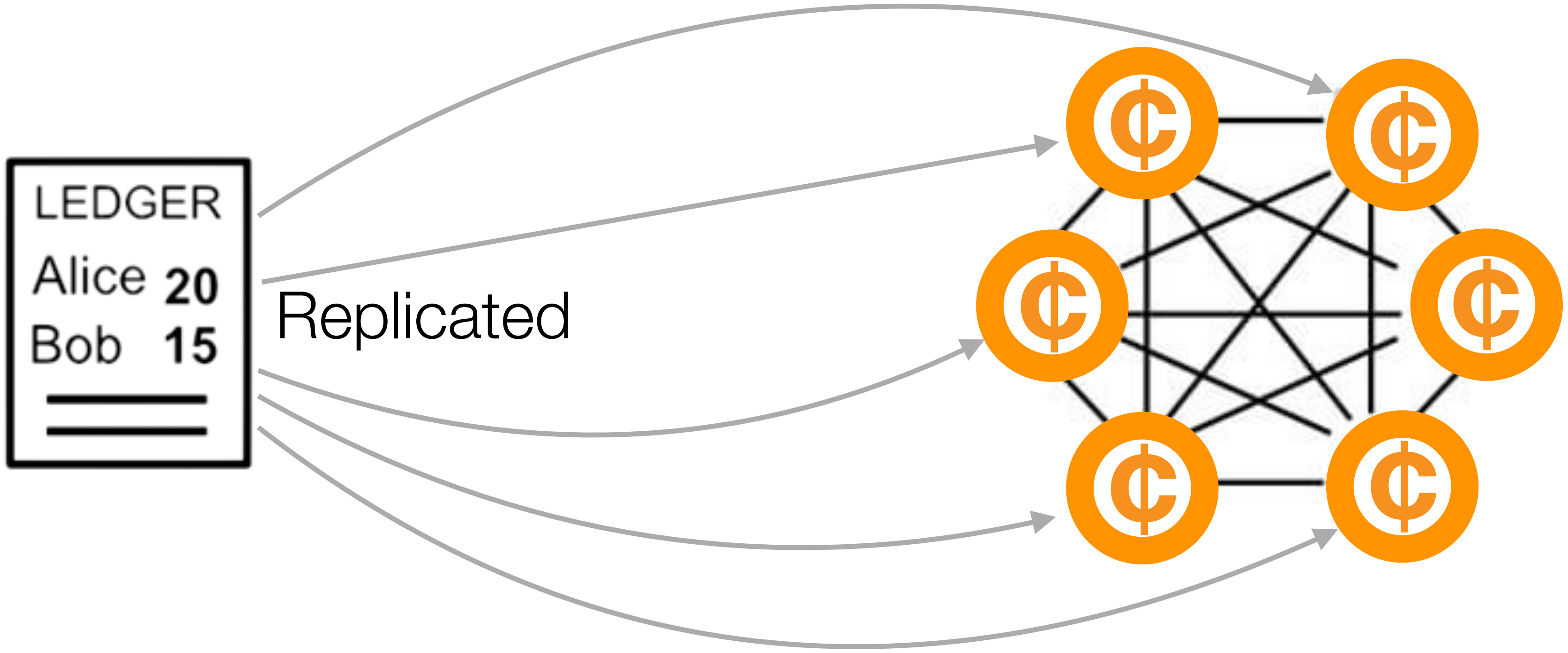


IN PRACTICE



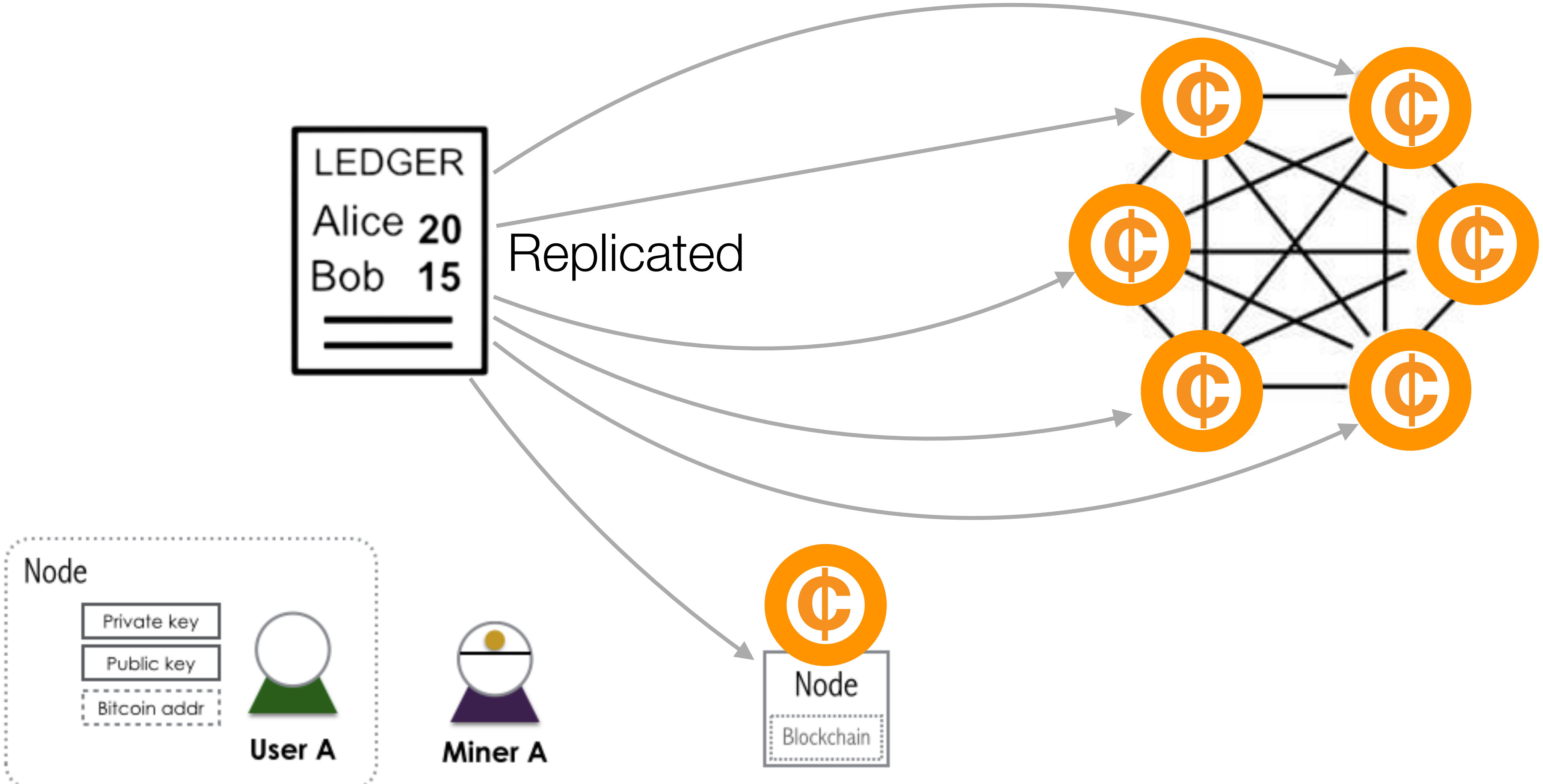
Main concepts

- Ledger



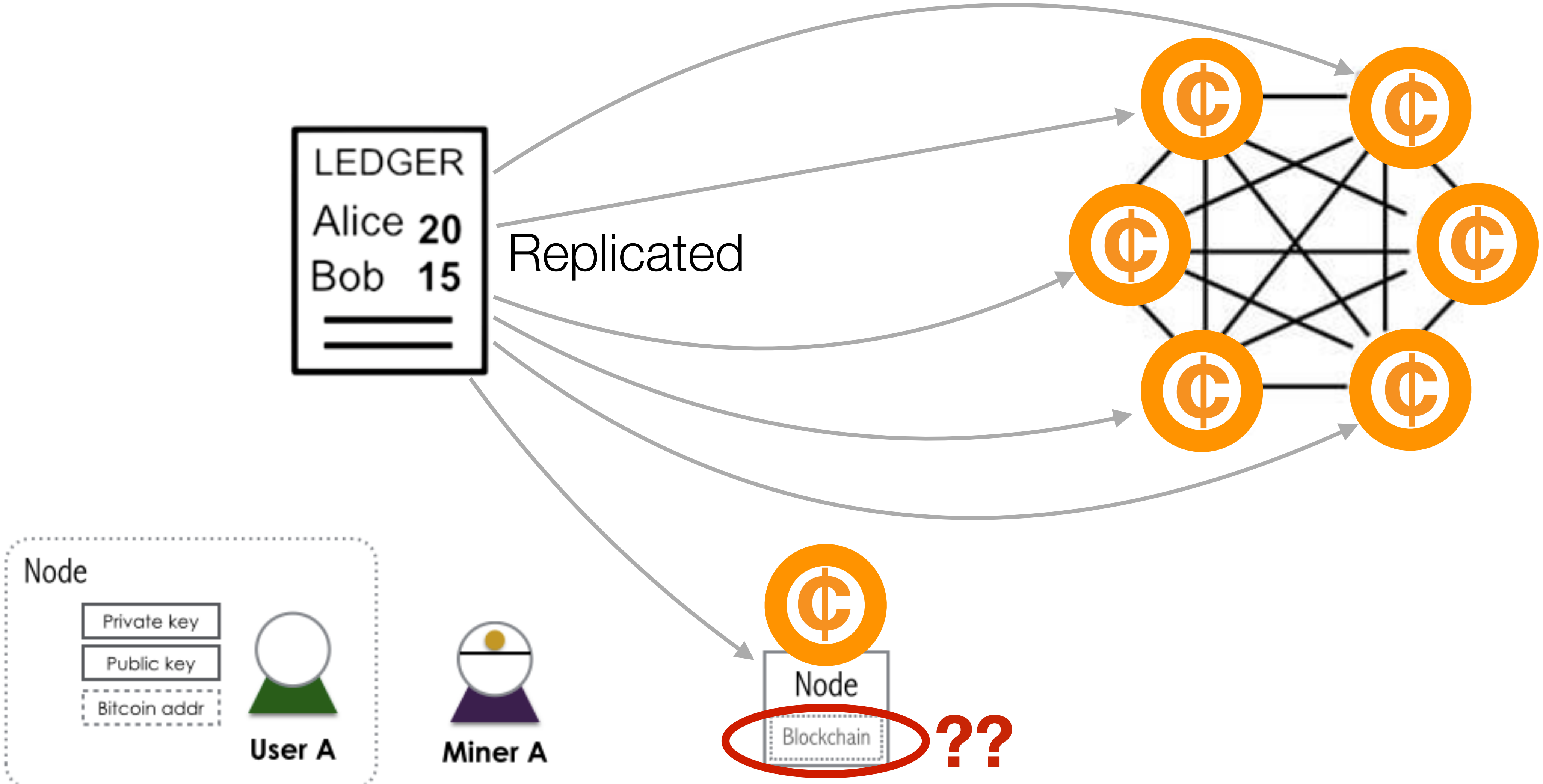
Main concepts

- Ledger
- Node



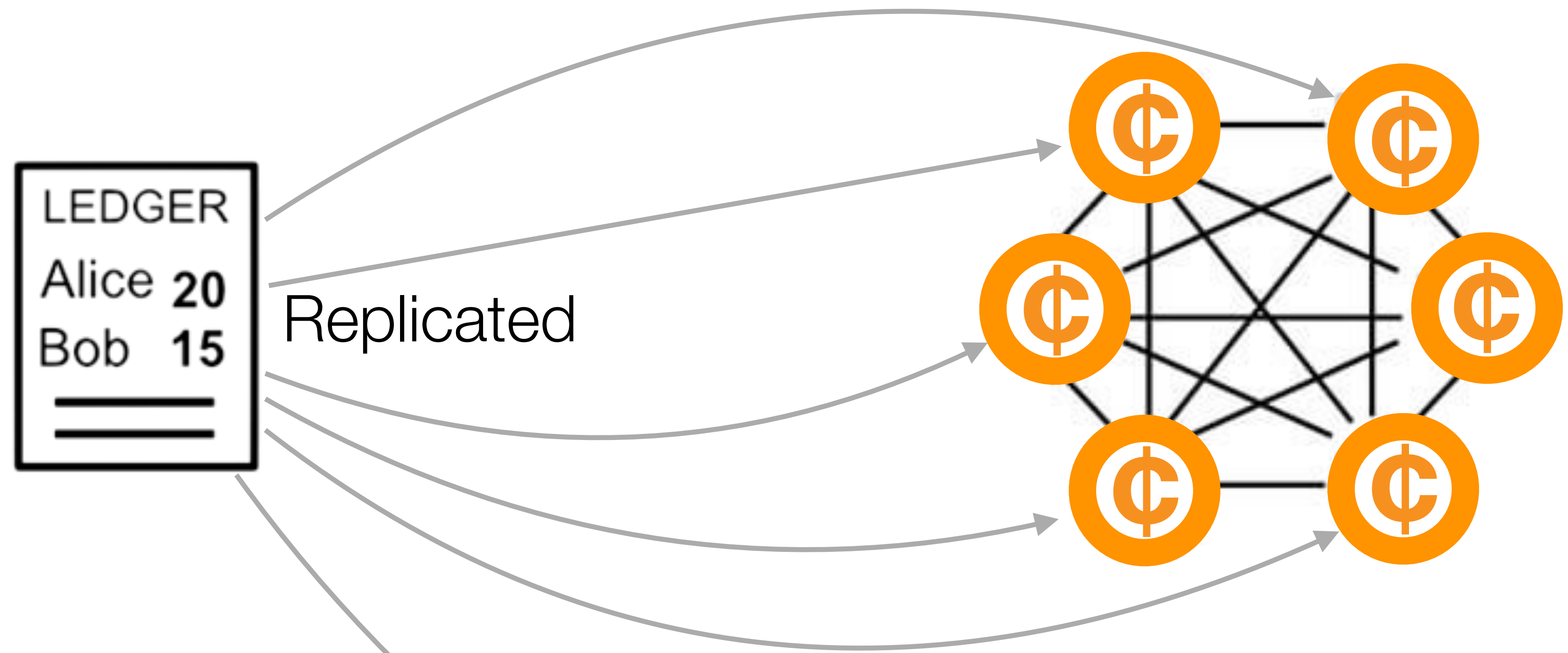
Main concepts

- Ledger
- Node

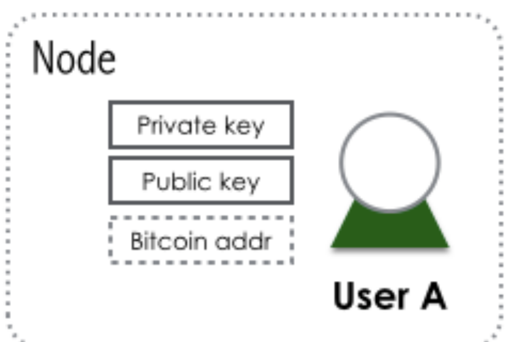


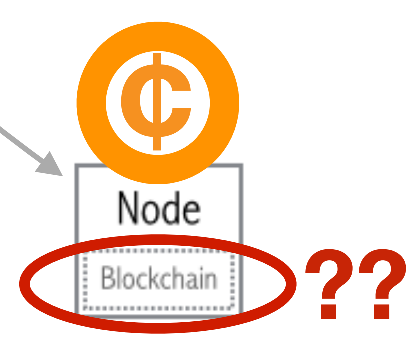
Main concepts

- Ledger

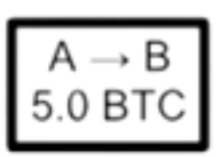


- Node





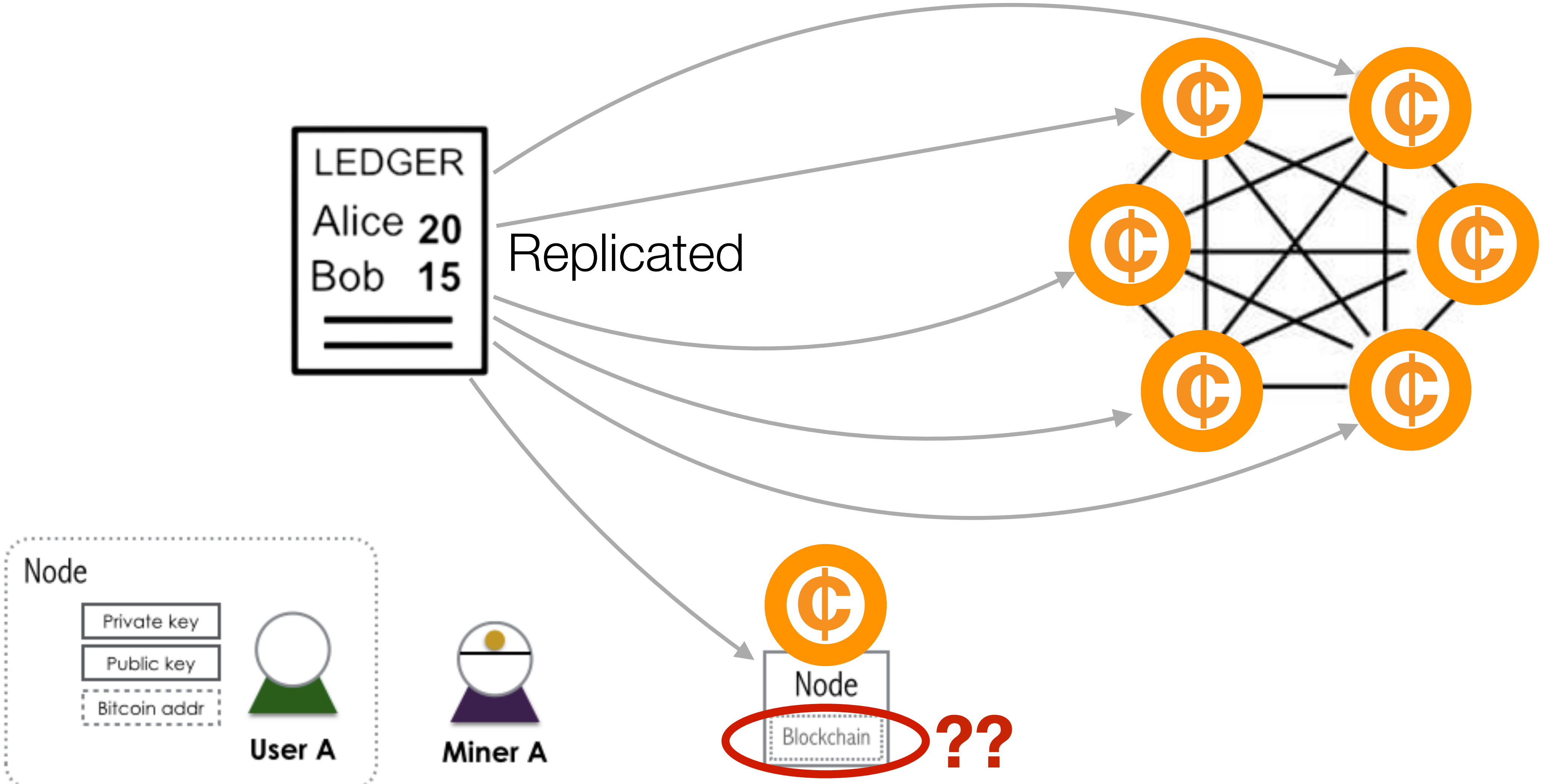
- Transaction



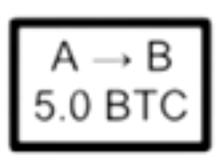
"Alice gives some money to Bob"

Main concepts

- Ledger
- Node



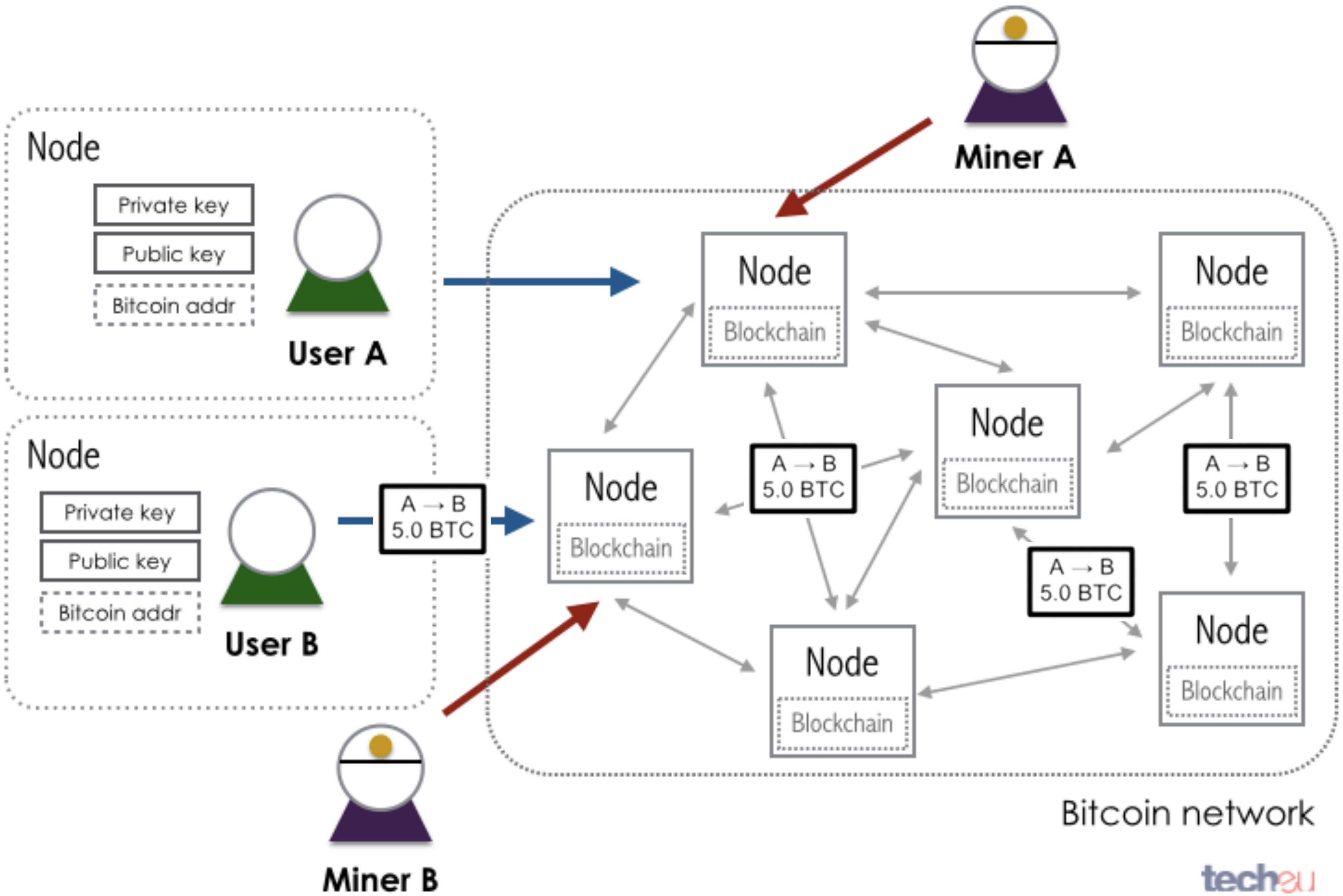
- Transaction



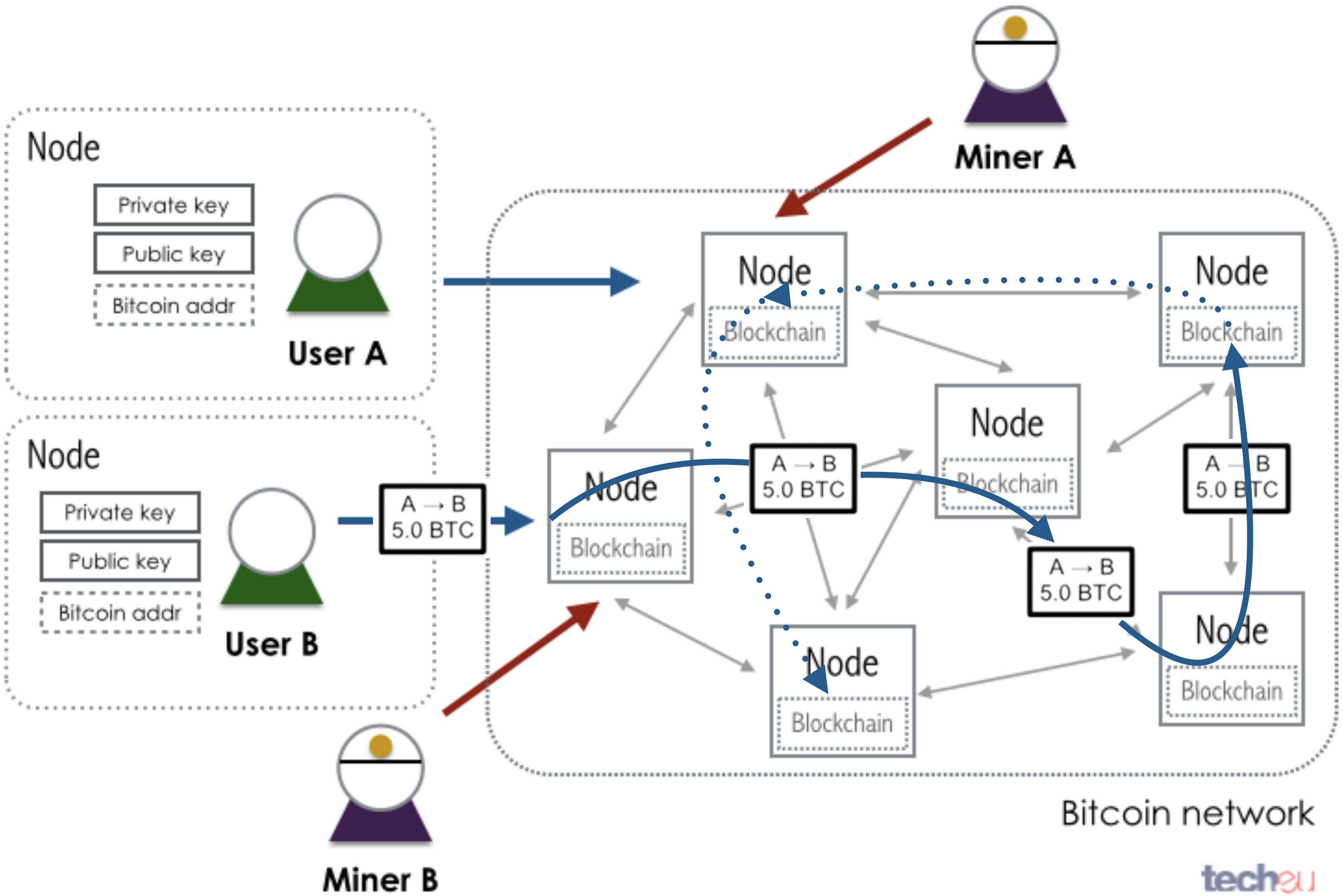
"Alice gives some money to Bob"

Let's see how they all fit together..

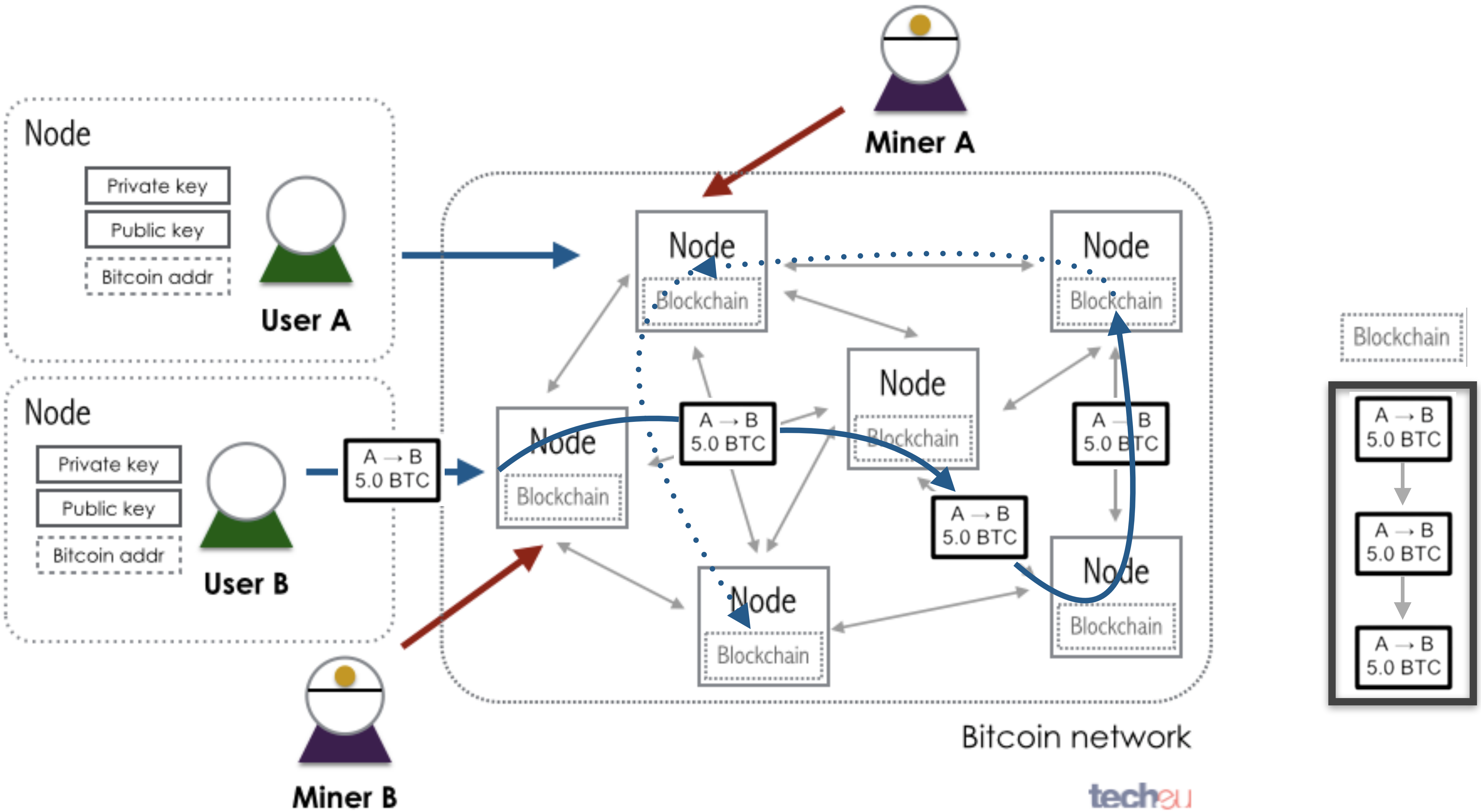
High-level interactions



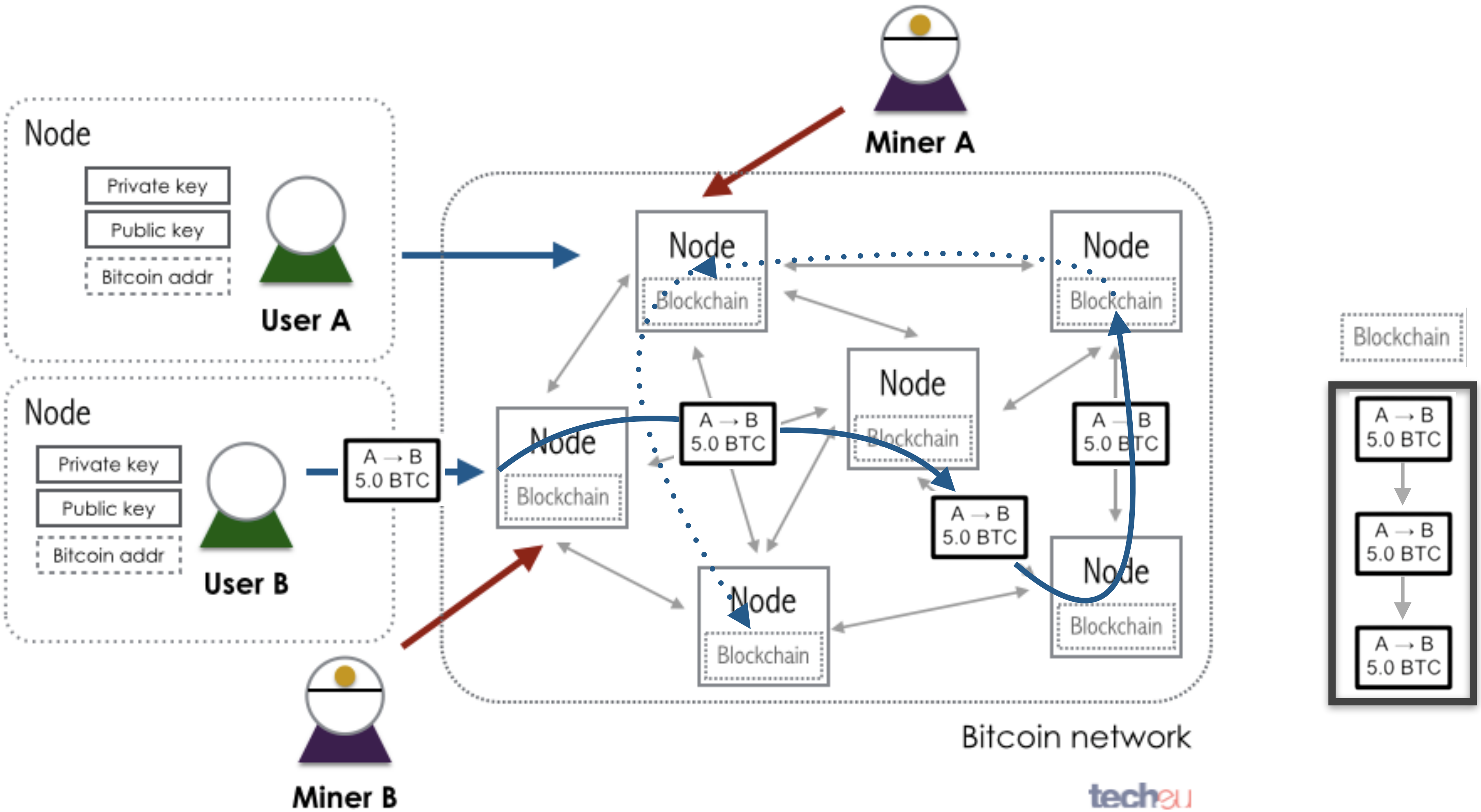
High-level interactions



High-level interactions

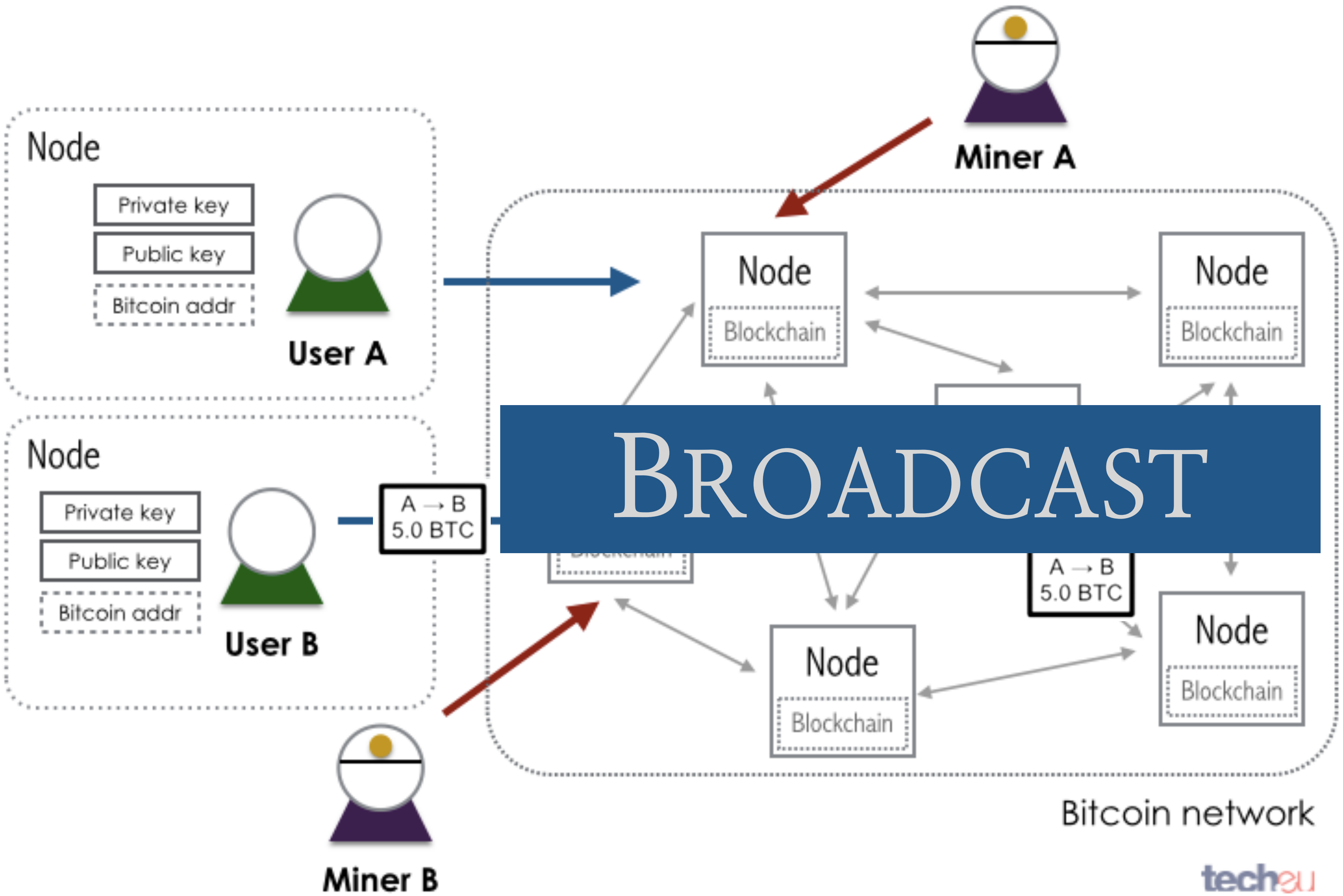


High-level interactions



Recognise any abstractions?

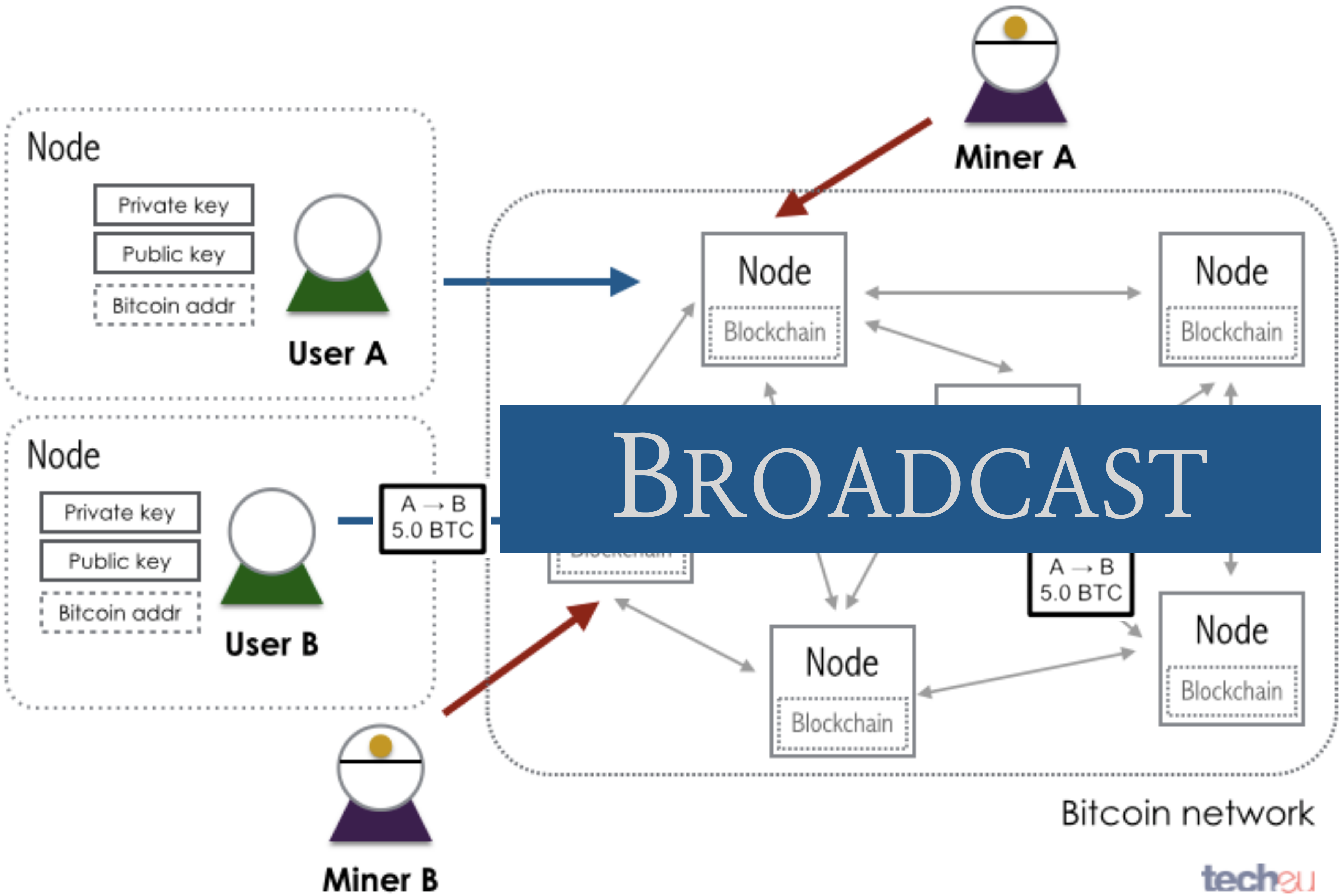
High-level interactions



What kind of broadcast?

High-level interactions

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE



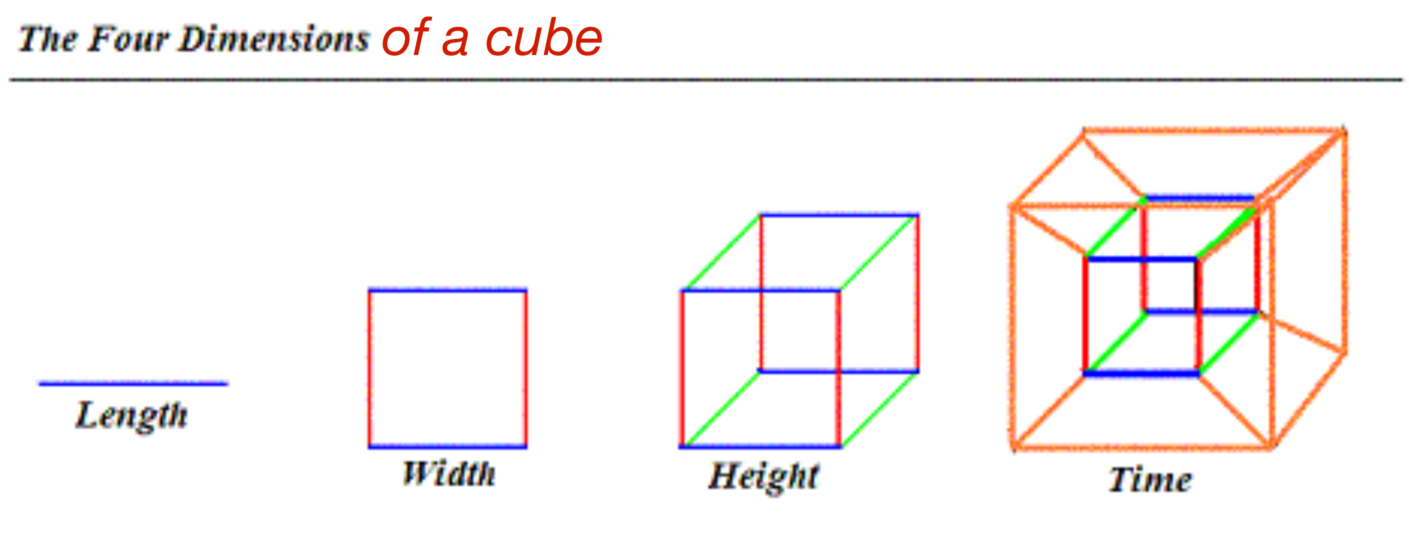
BROADCAST

What kind of broadcast?

PROPERTIES (guarantees)

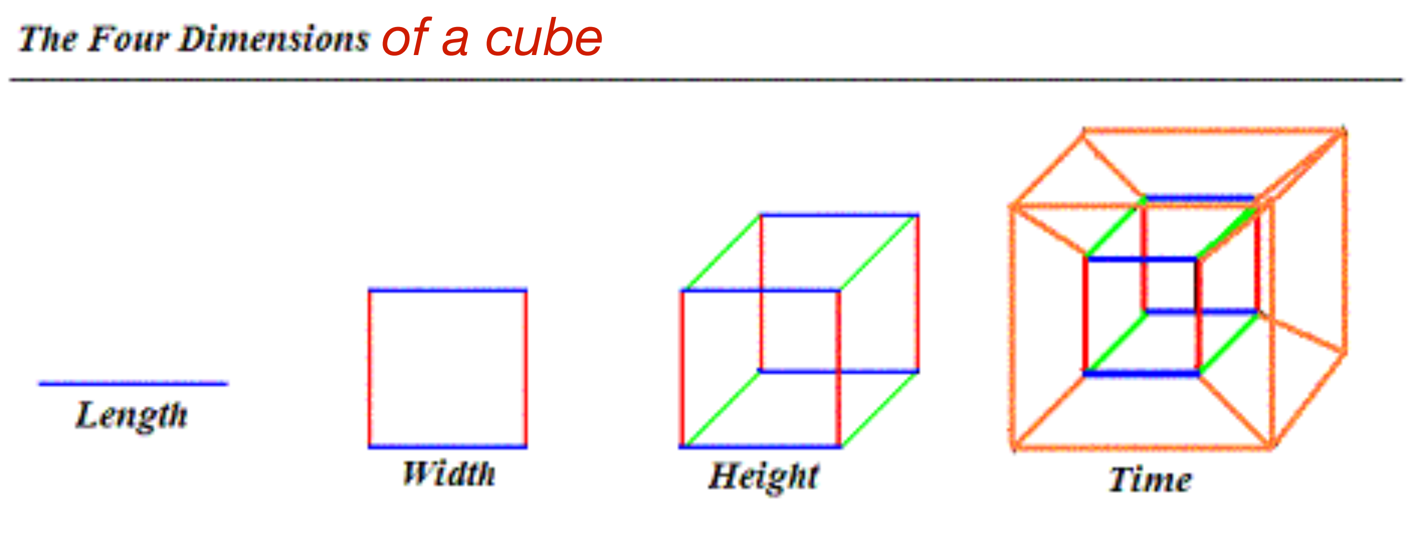
PERFORMANCE

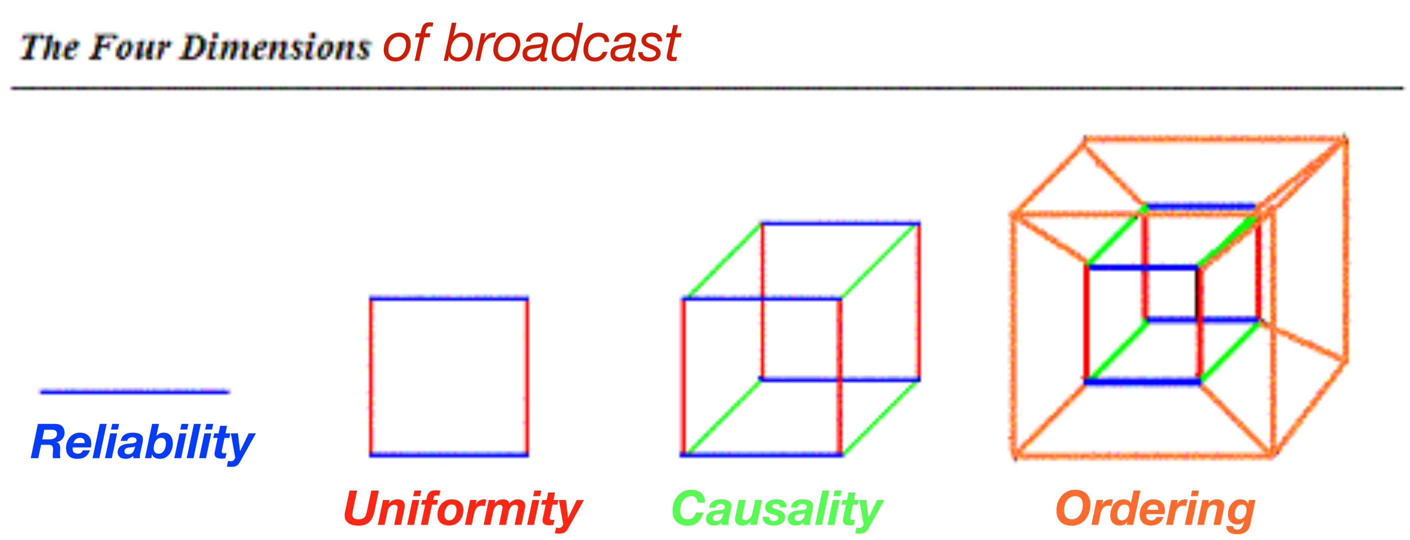
PROPERTIES



PROPERTIES

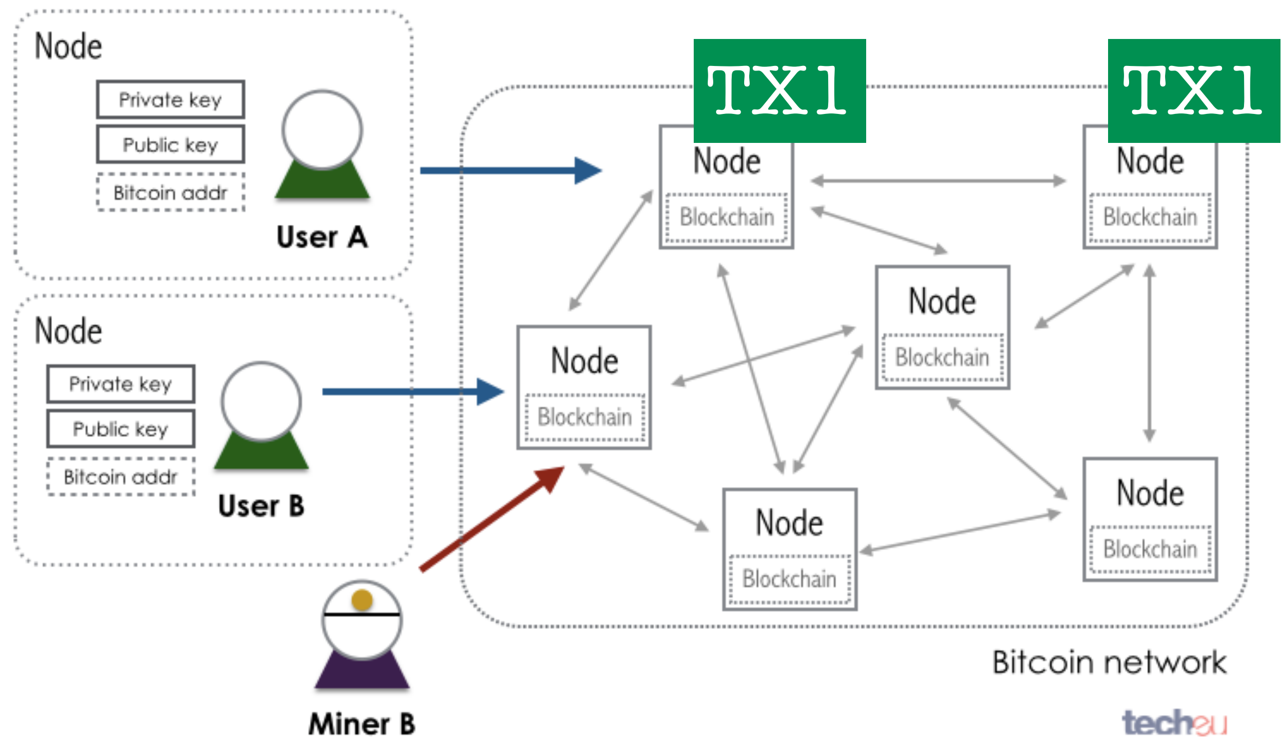
Analogy
(not formal)





PROPERTIES

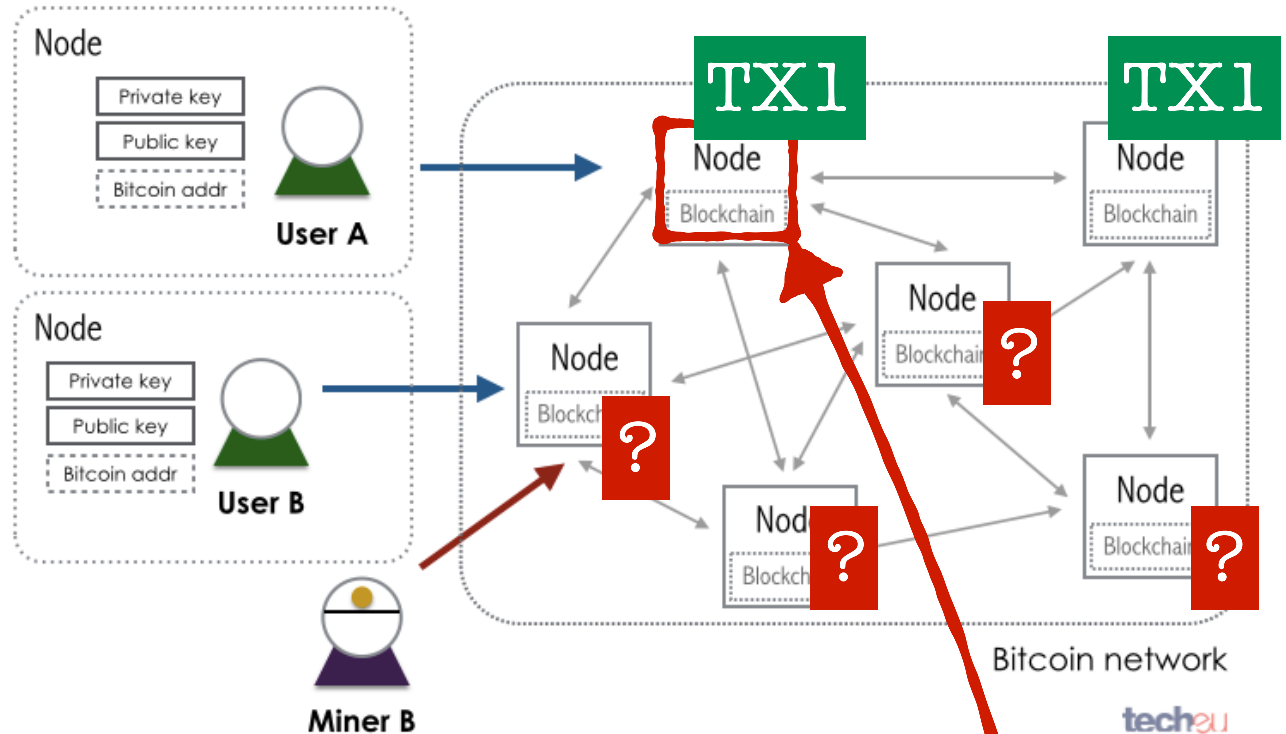
Do we need Reliability?



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

PROPERTIES

Do we need Reliability?

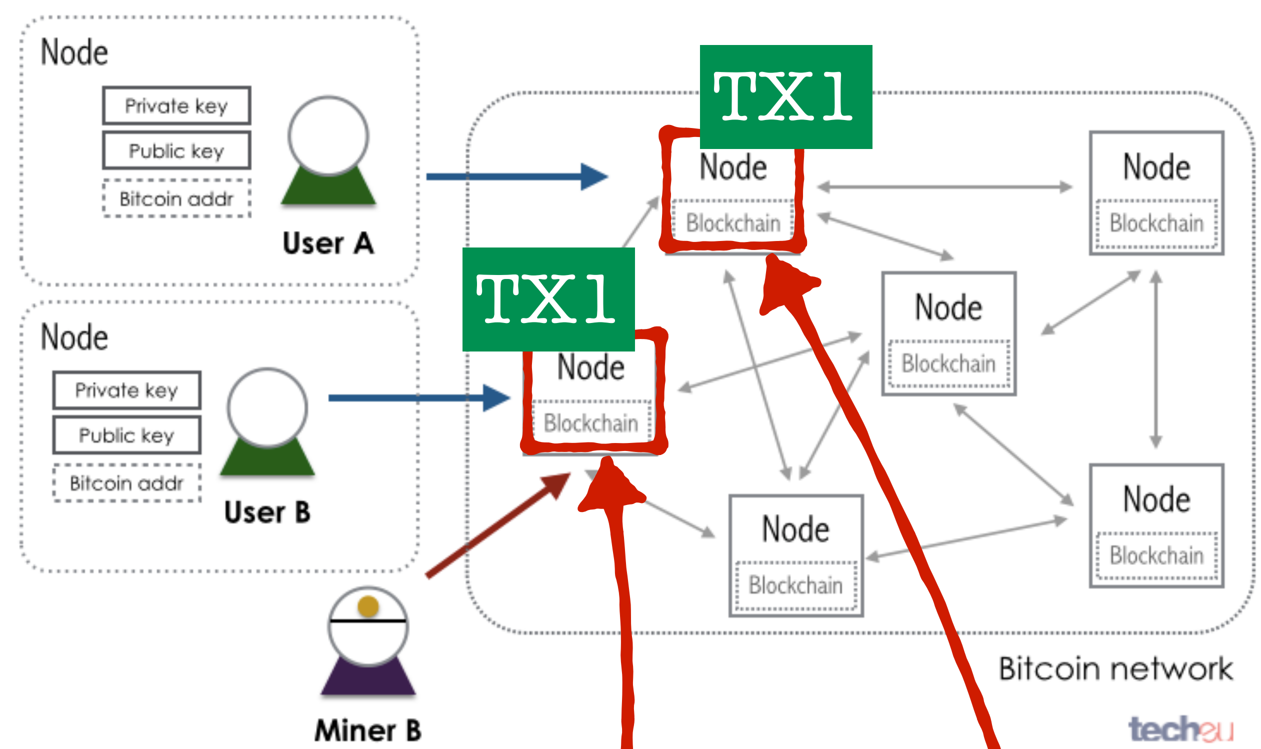


- Consider the following:
  - User A starts TX1
  - Use best-effort broadcast
    - Validity + !Duplication + !Creation
    - <u>"the burden of reliability is on the sender"</u>
    - Lacks Agreement ⇒ Nodes diverge

What if the sender crashes?

PROPERTIES

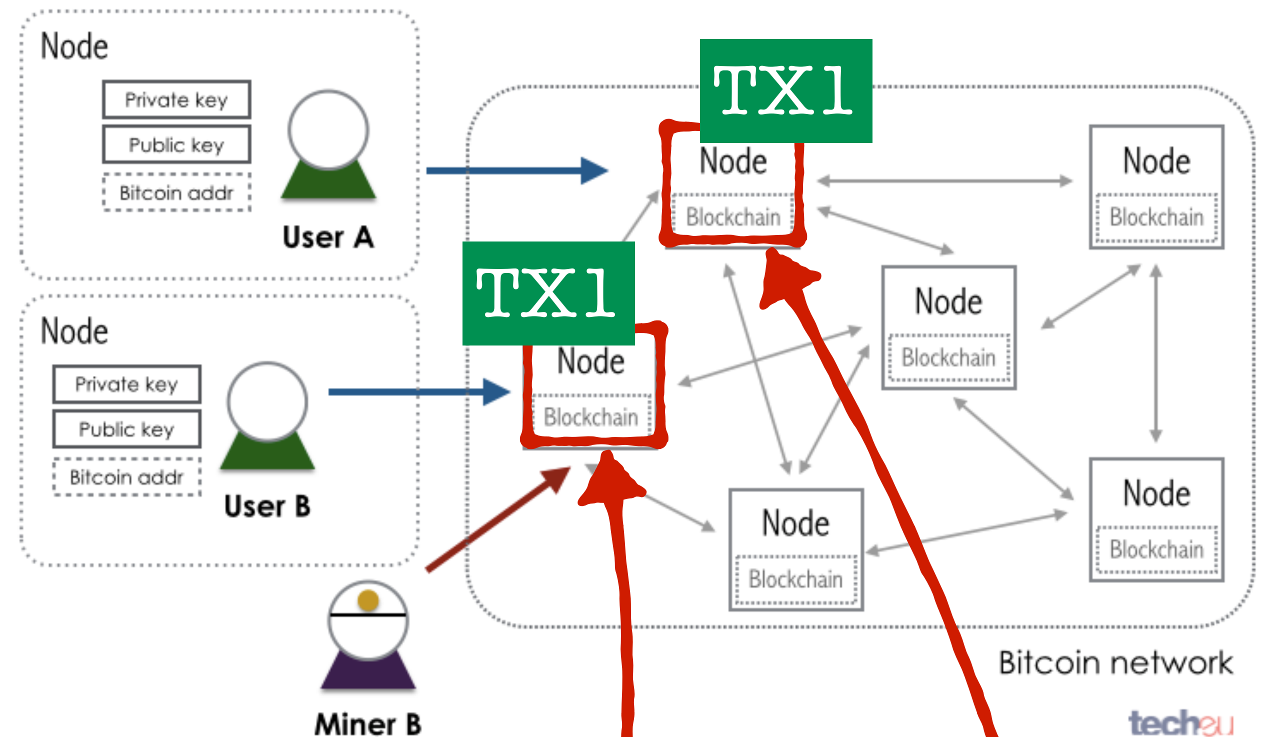
Do we need Uniformity?



- 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?

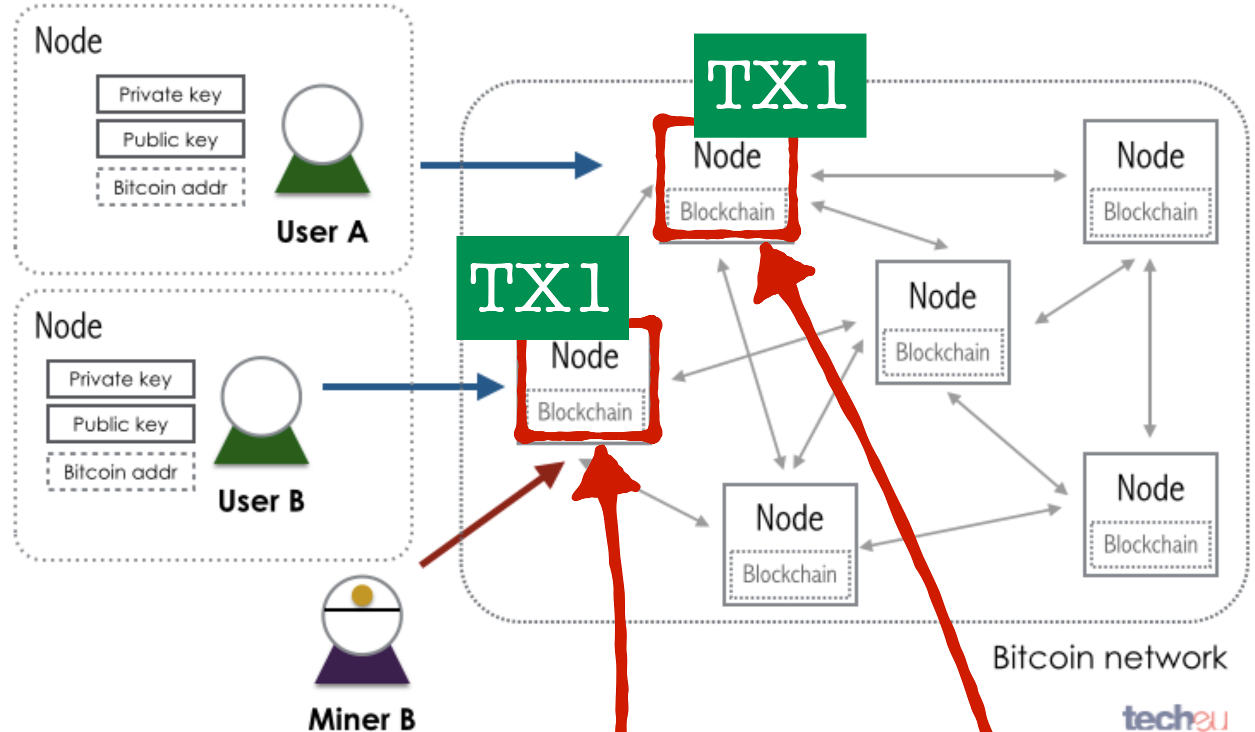


- Consider the following:
  - User A starts TX1
  - Use regular reliable broadcast
    - Validity + !Duplication + !Creation + Agreement for correct nodes
  - Is it OK to deliver and crash?

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

PROPERTIES

Do we need Uniformity?



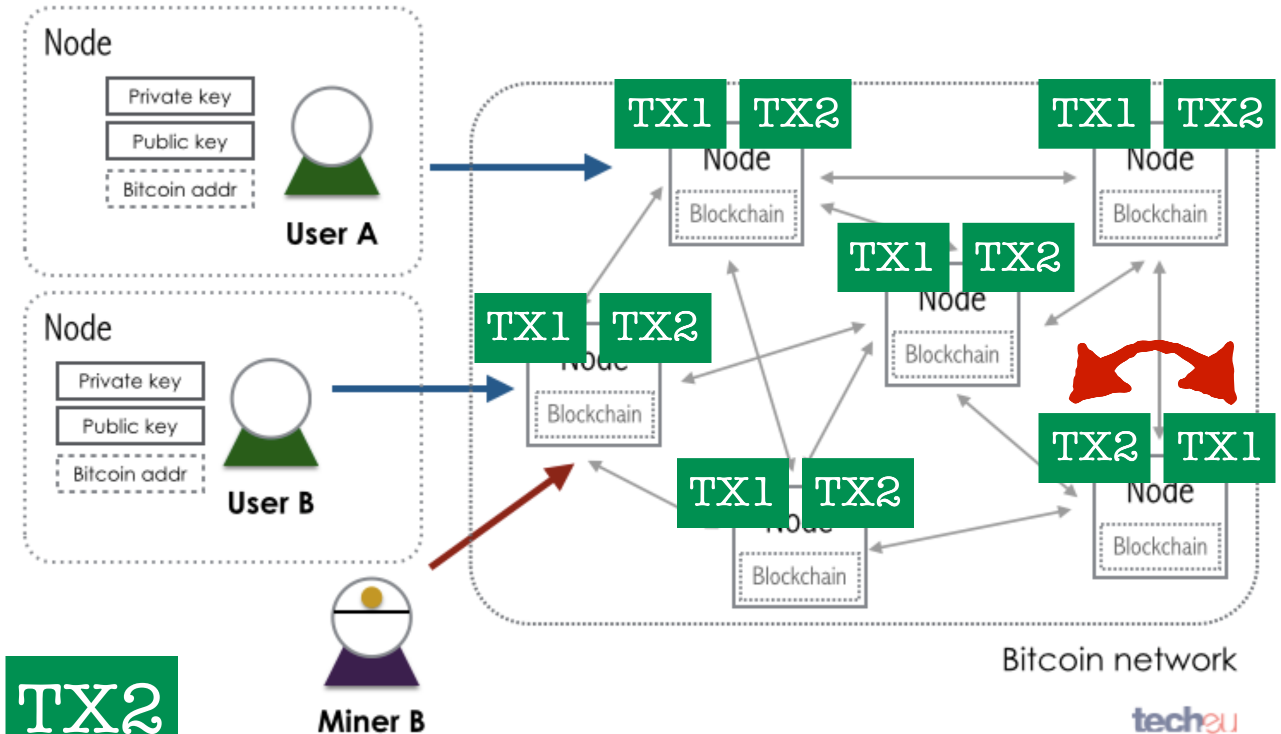
- Consider the following:
  - User A starts TX1
  - Use regular reliable broadcast
    - Validity + !Duplication + !Creation + Agreement for correct nodes
  - Is it OK to deliver and crash?

What if User B observes TX1 before the nodes crash?

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

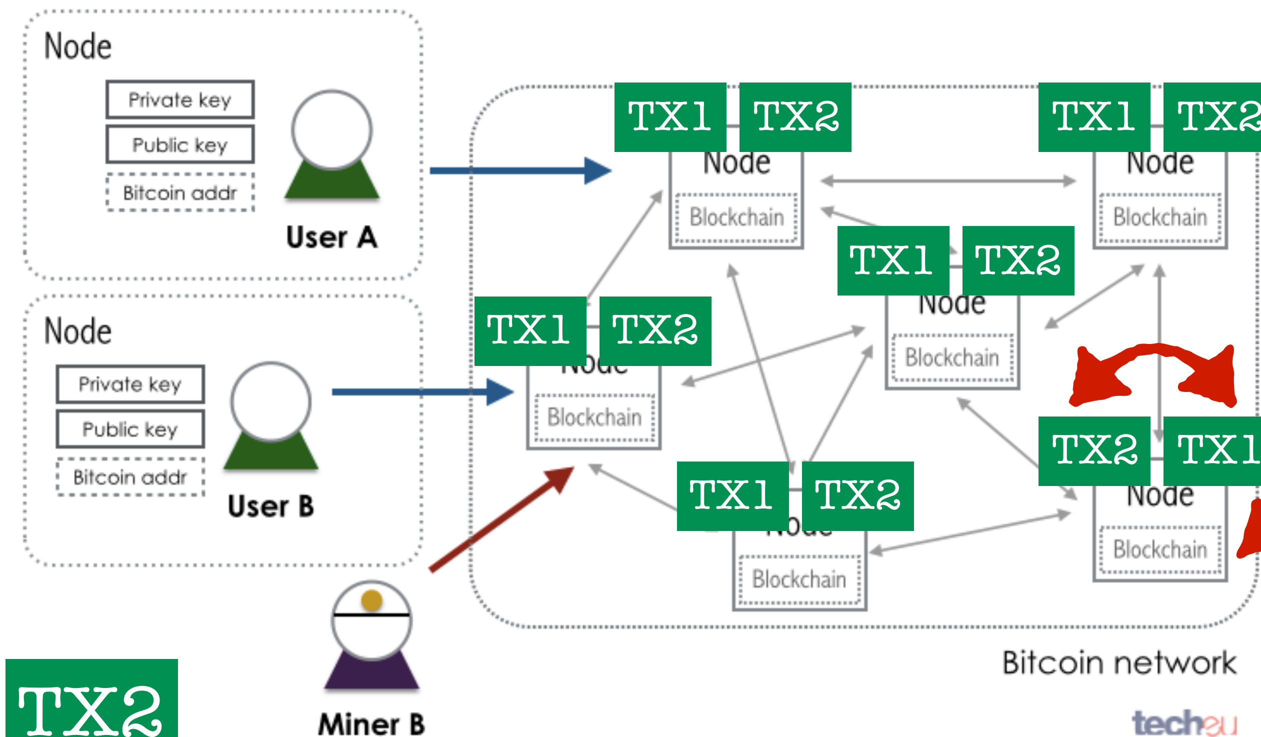
PROPERTIES

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



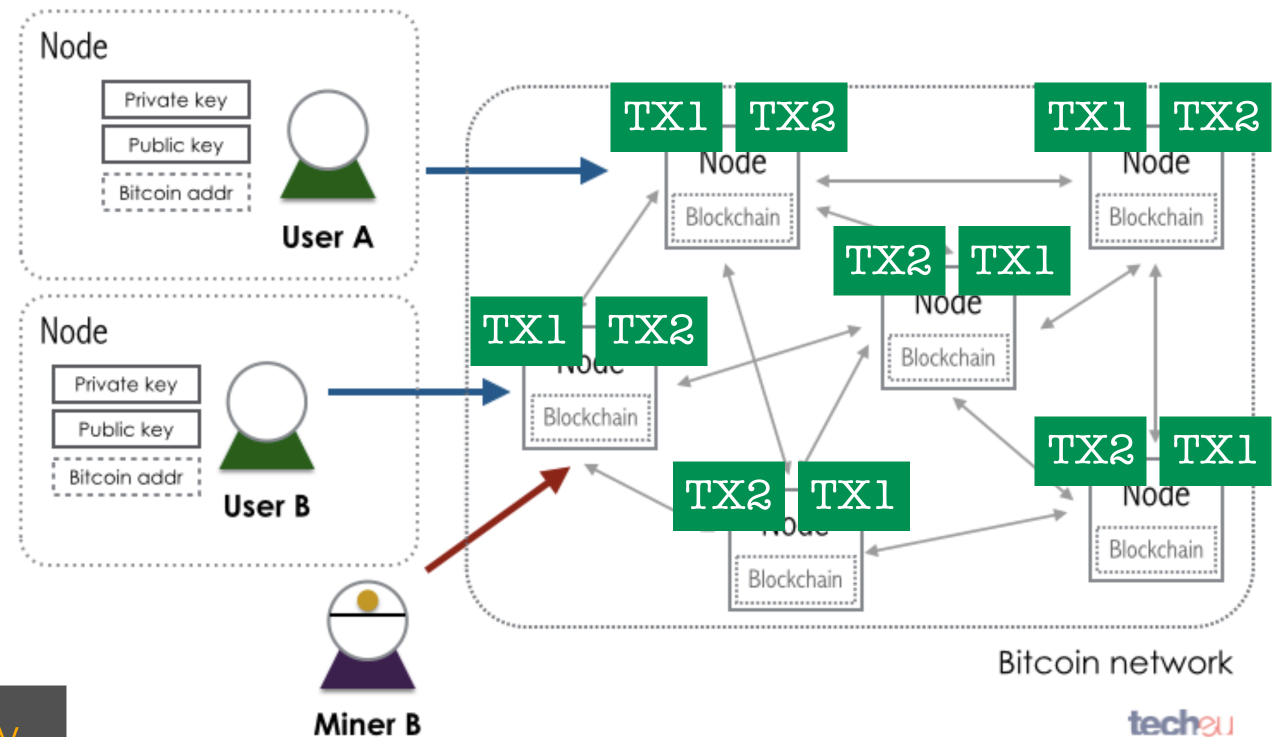
Do we need Causality? (partial ordering)

- 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?



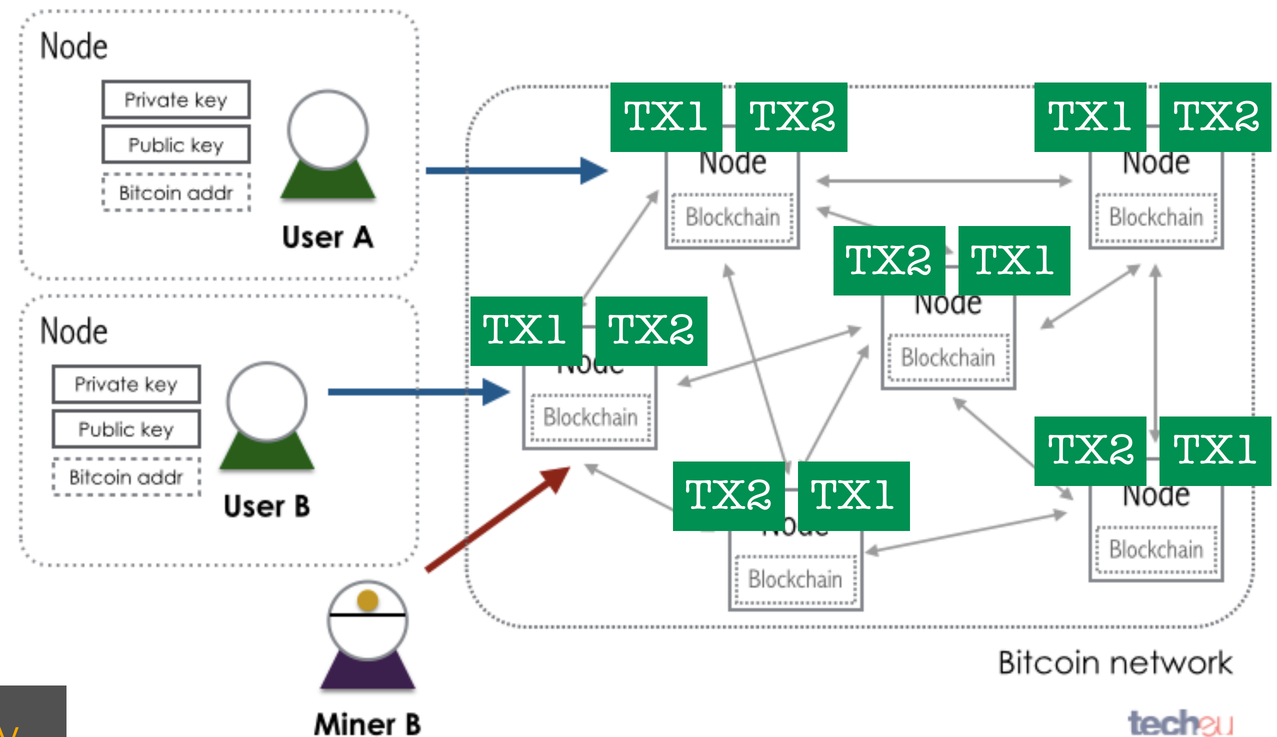
- Consider the following:
  - User A starts TX1
  - User B starts TX2

  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?



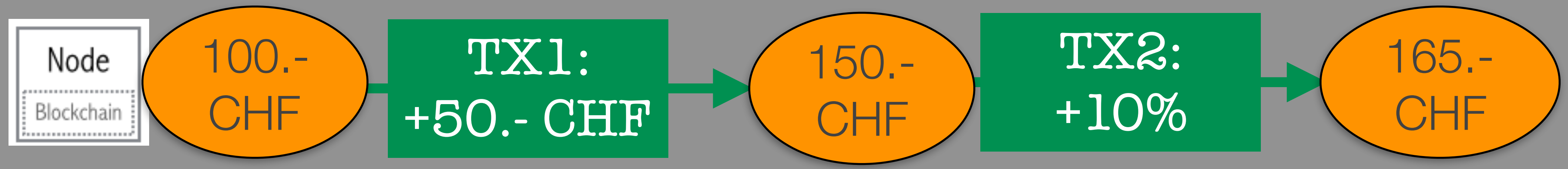
- Consider the following:
  - User A starts TX1
  - User B starts TX2

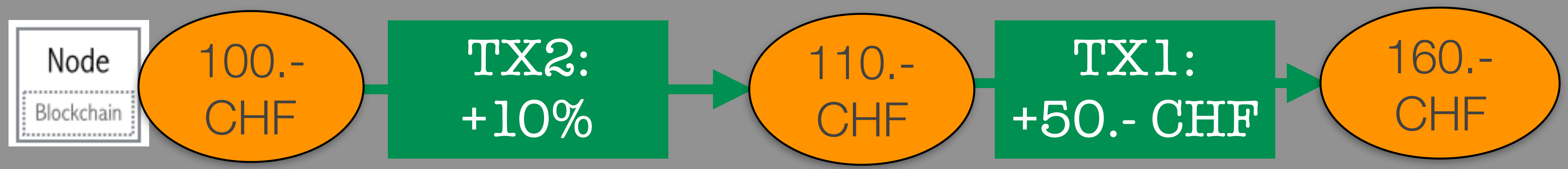
No dependency among these two

  - 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

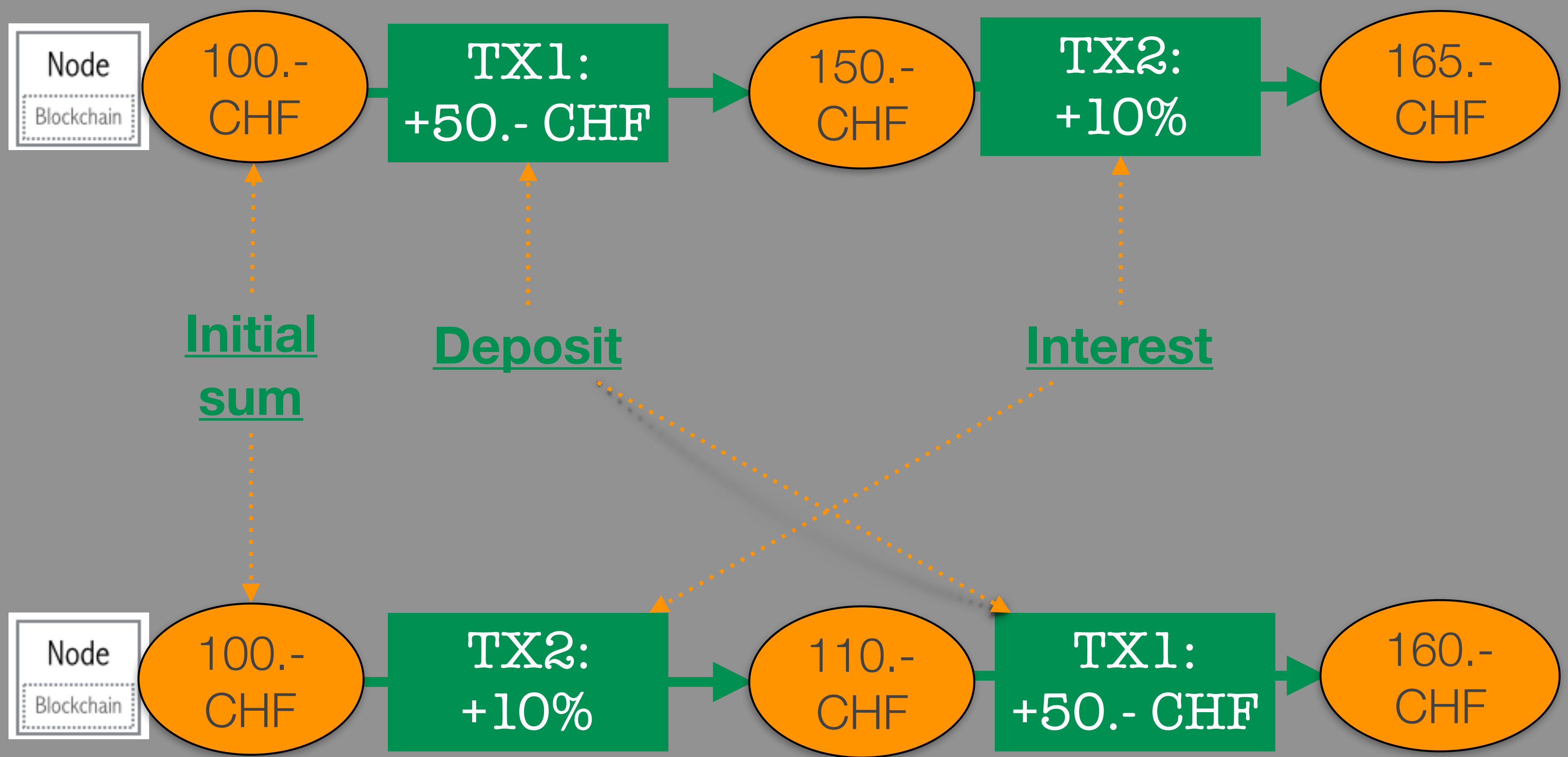
What if TX1 and TX2 are not commutative?

Commutativity
counter-example



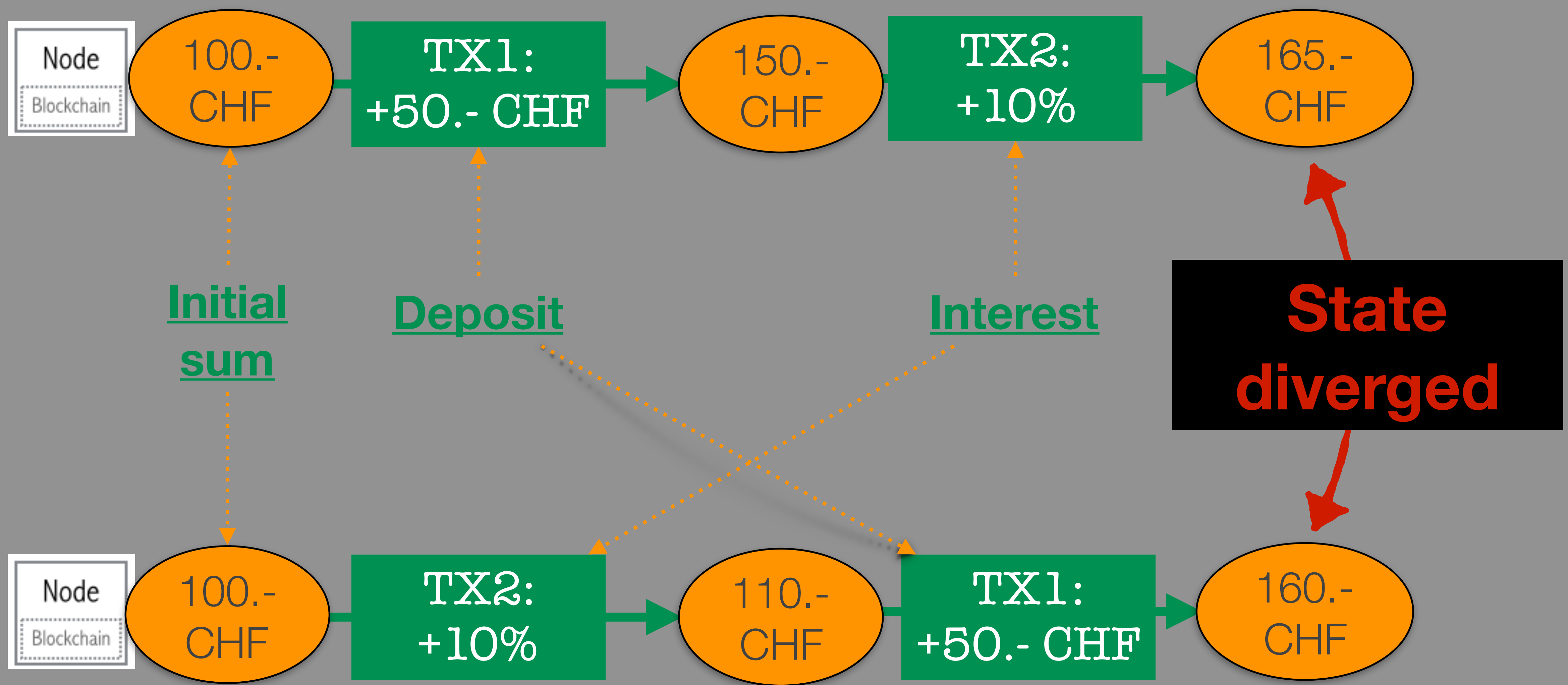


Commutativity
counter-example



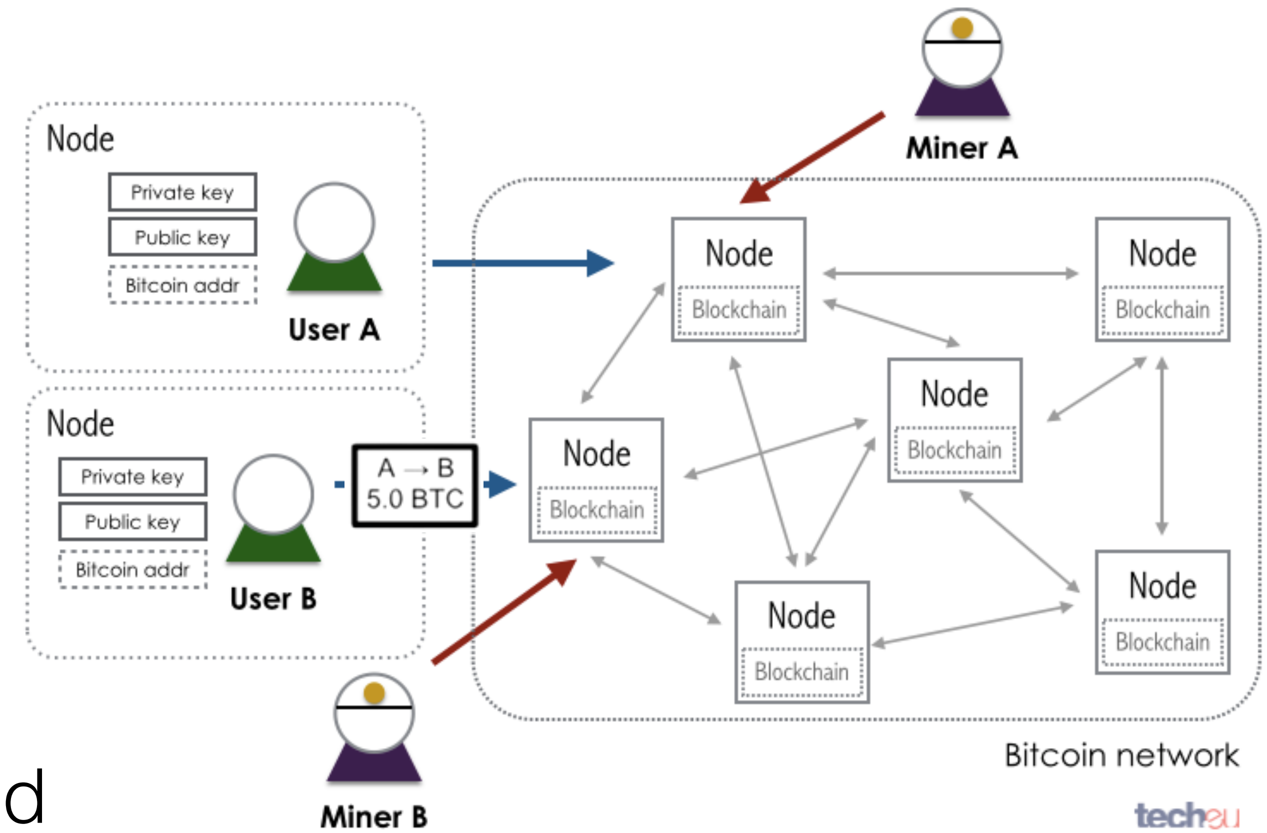
Commutativity

counter-example



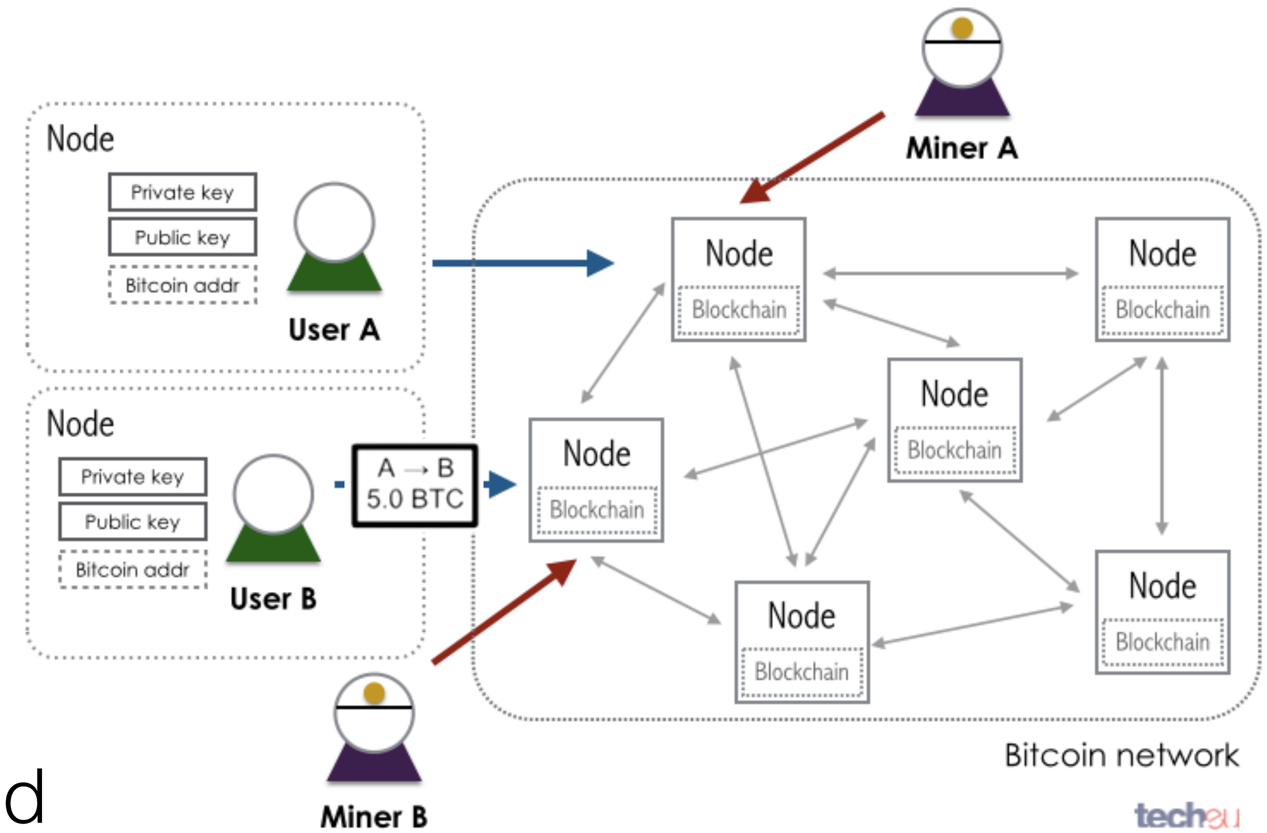
PROPERTIES

- 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

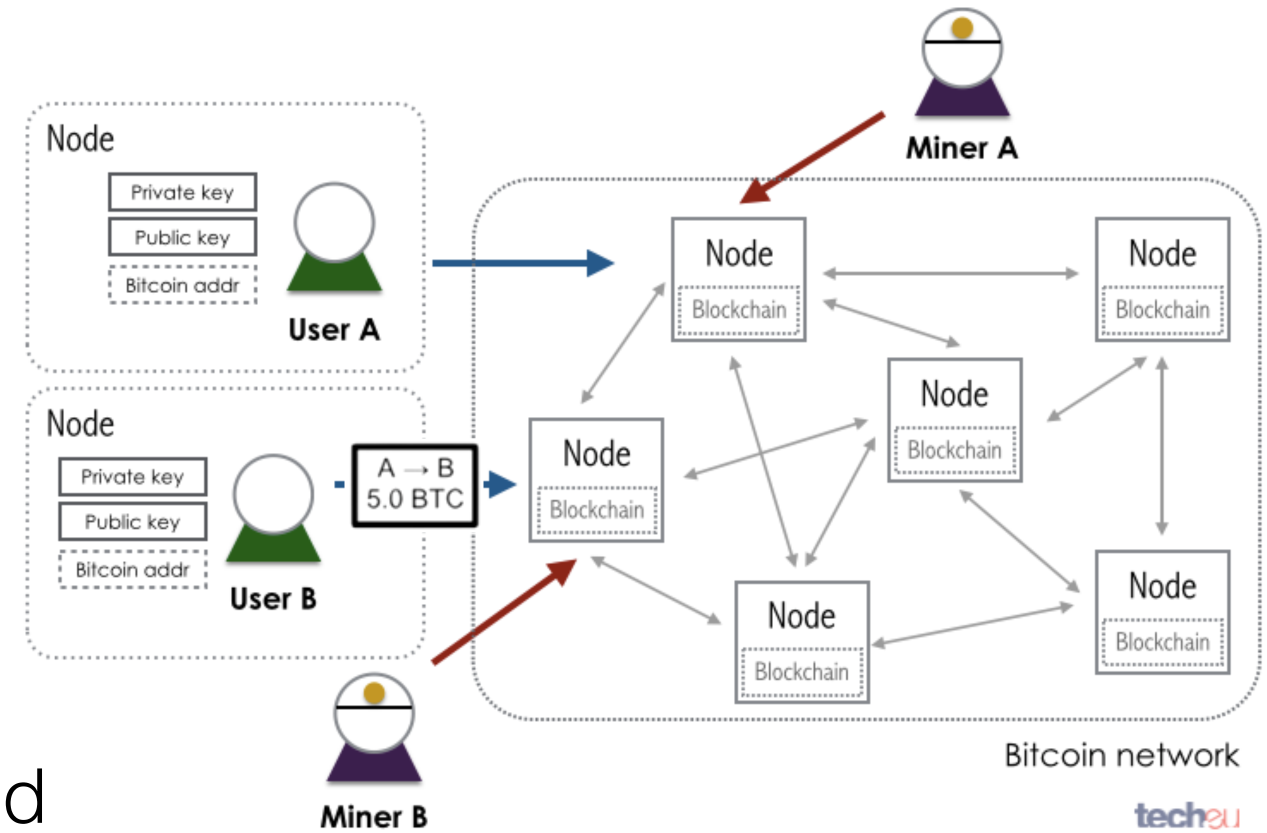


Contrast:

- 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)

CAMIPRO BITCOIN

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

Specification

Specification

Module:

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

Properties:

- **RB1, RB2, RB3, RB4**
- **Causal Order (CO)**
- **Total order (TO)**

What's left
????

Specification

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

User

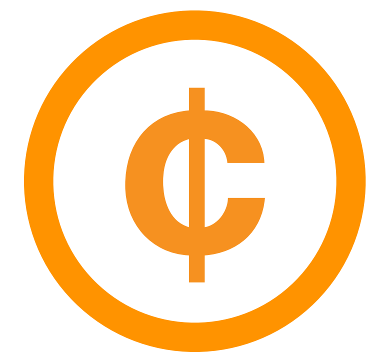
Request ↓ **Indication** ↑

CAMIPRO-Bitcoin

Module:

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

Properties:



- **RB1, RB2, RB3, RB4**
- **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

S4:
(Student's) Simple Storage Service

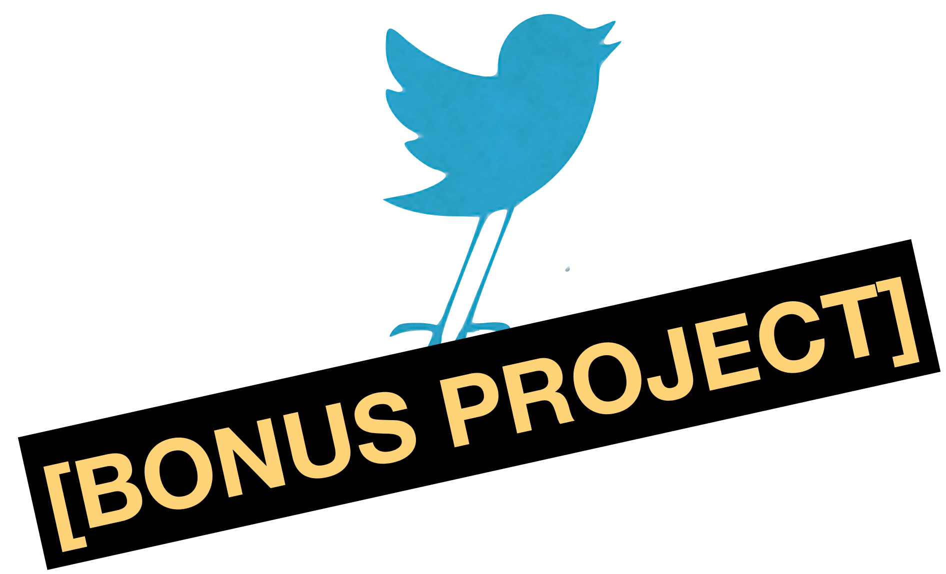
Overview:

1. Setup & System model
2. Operations
3. Goals
4. Technicalities
5. One last look at causal consistency

S4: (Student's) Simple Storage Service

Overview:

1. Setup & System model
2. Operations
3. Goals
4. Technicalities
5. One last look at causal consistency

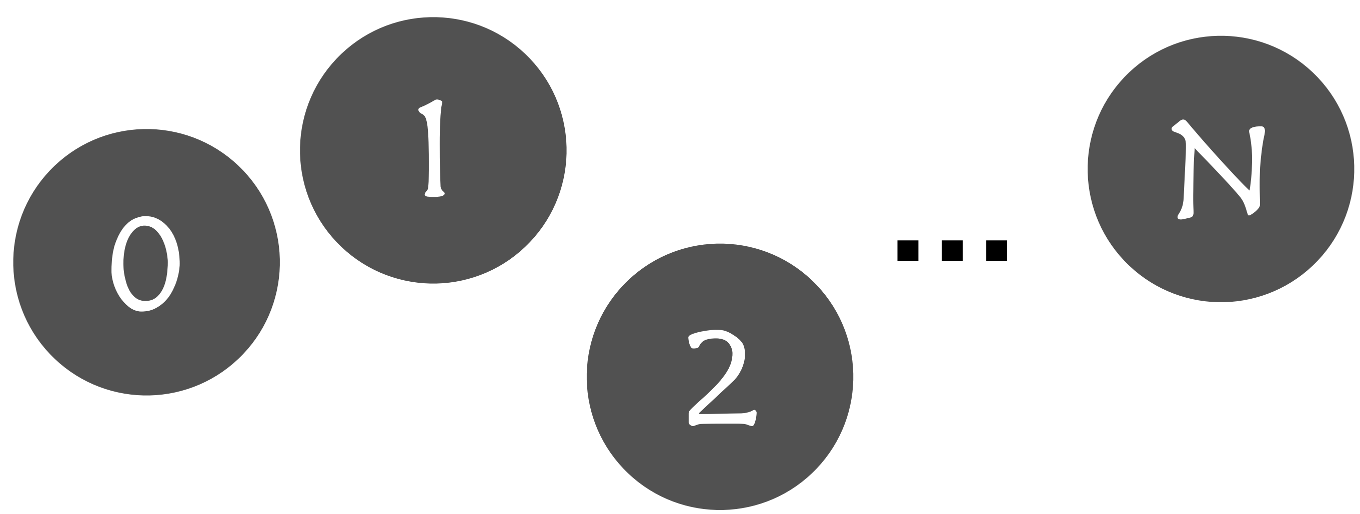


S4 — System model

Setup:

- Users

- Storage replicas (Nodes)



- Objects: stored on replicas

⟨key, value⟩

⟨A, 0⟩

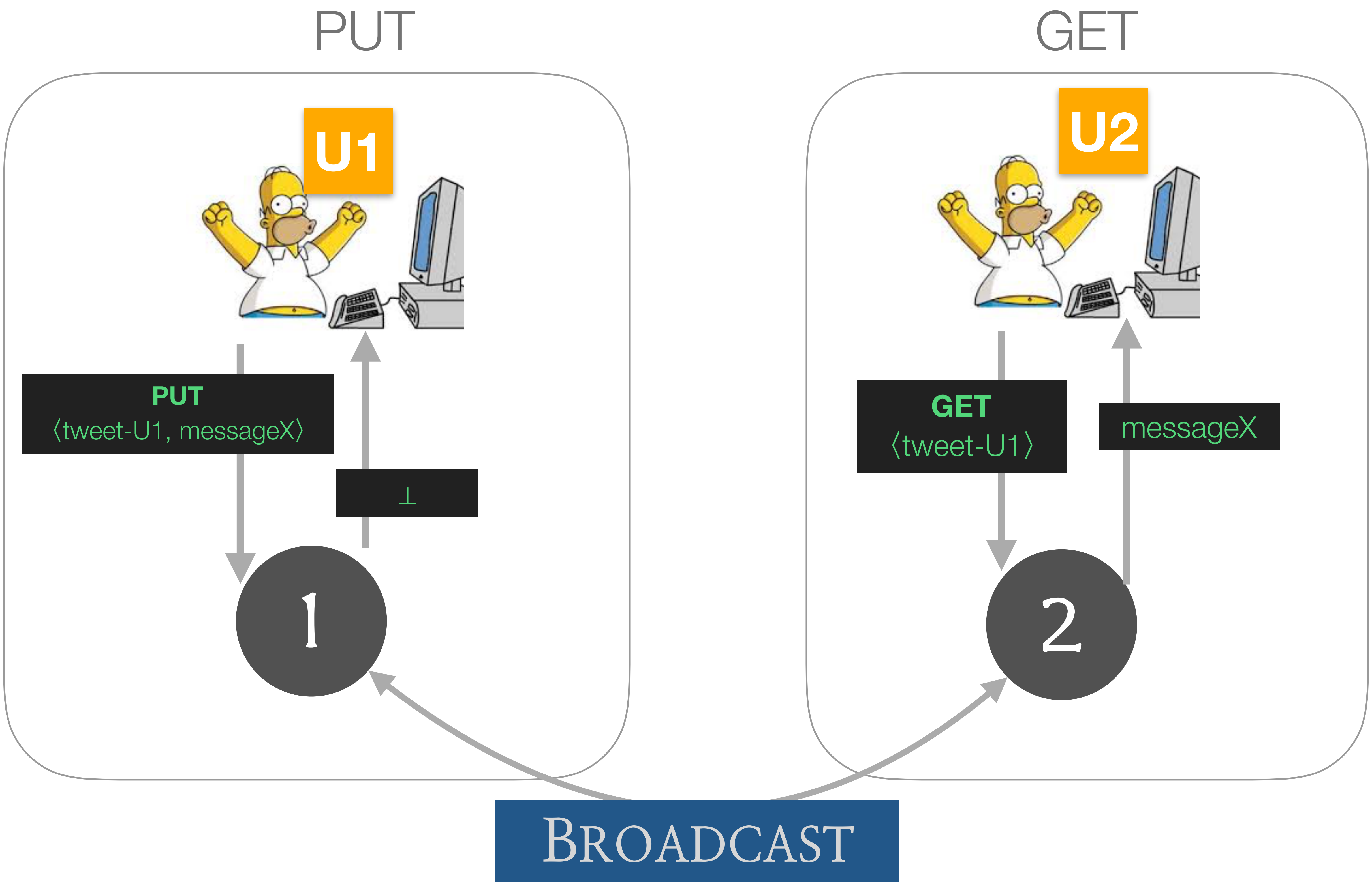
⟨B, 1⟩

⟨C, 9⟩

⟨tweetY, BYE⟩

⟨MSGx, hello⟩

S4 — Operations

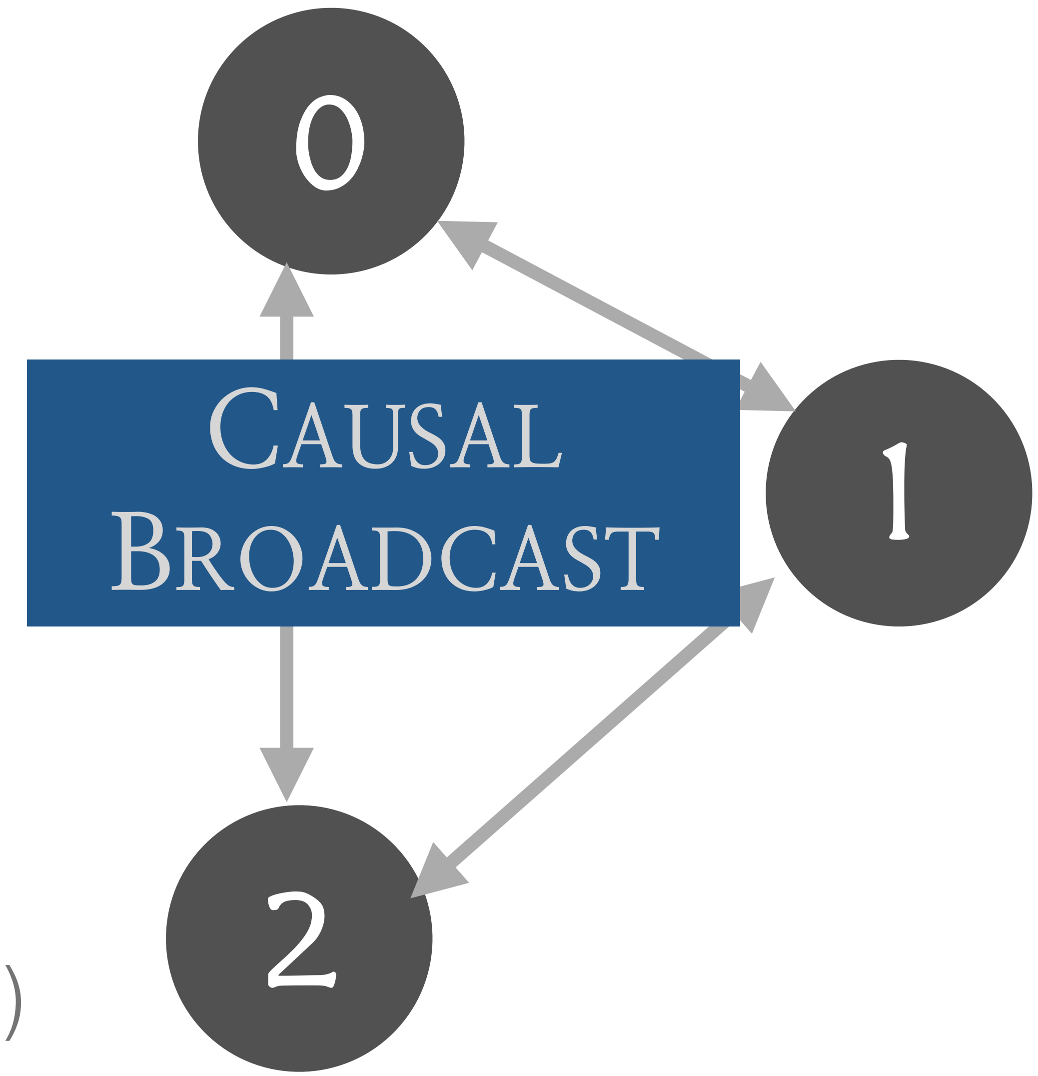


S4 — Goals

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Goals:

1. **Reliability**
   - Three replicas
   - Communicate through message-passing

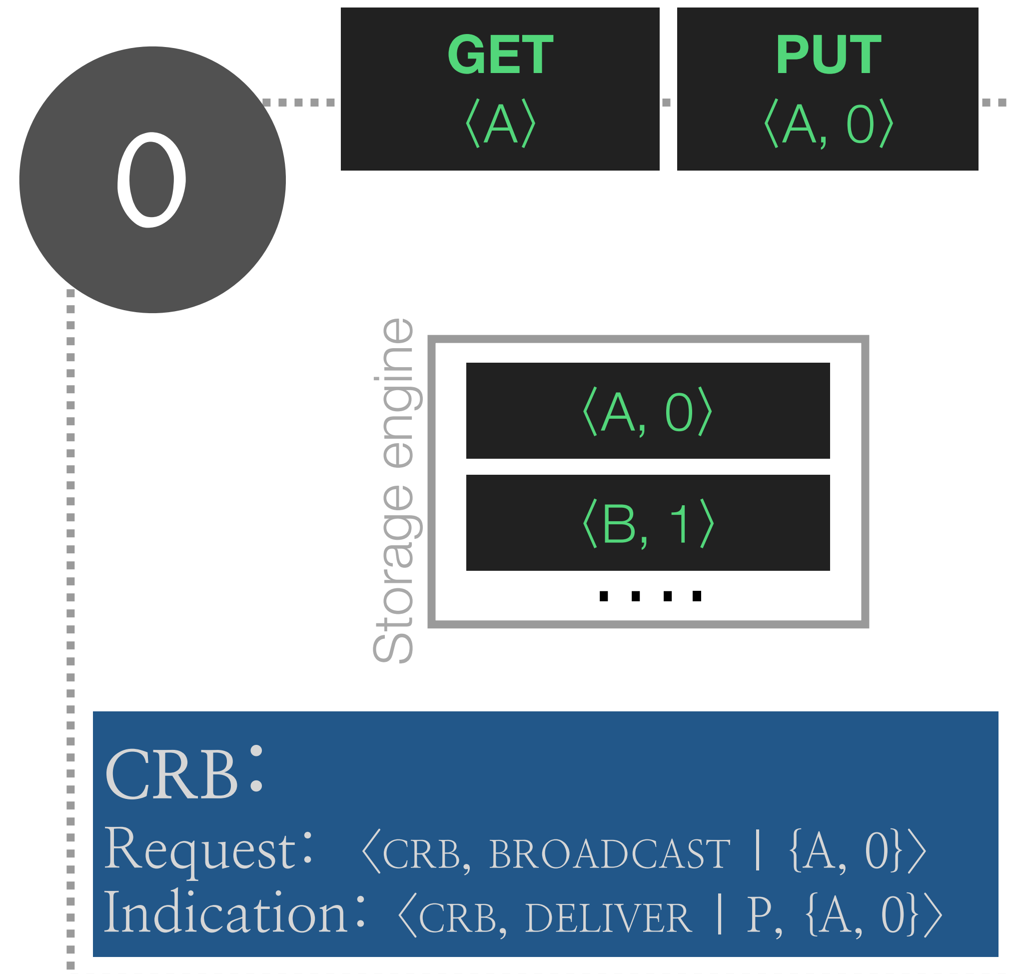


2. **Consistency**
   - We want to use it in the Internet (WAN)
     - (Unlike CAMIPRO-Bitcoin, which was optimized for the EPFL network)
     - Can't afford total-order, too expensive!
   - Causal consistency

Technicalities

Nodes:

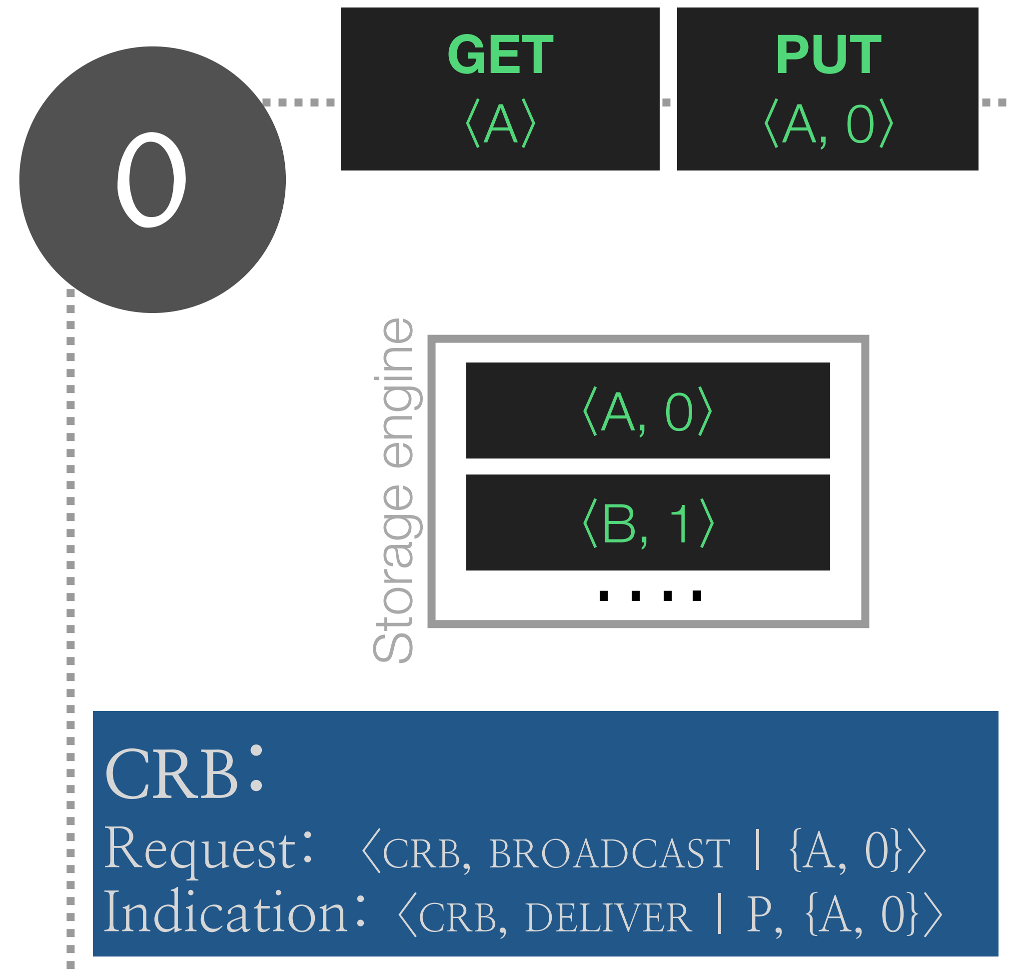
- Each node is a Linux process
- Contains:
    - A storage engine
    - A causal-broadcast implementation (*crb*)



Technicalities

Nodes:

- Each node is a Linux process
- Contains:
  - A storage engine
  - A causal-broadcast implementation (*crb*)



We start a process by calling:

```
s4 addr-0 port-0 addr-1 port-1 addr-2 port-2 n f
```

input file

Node 0: Address+Port

Node 1: Address+Port

Node 2: Address+Port

ID∈{0,1,2}

Technicalities

To start all nodes,
we execute the commands:

```
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 0 0.input
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 1 1.input
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 2 2.input
```

Technicalities

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

**To start all nodes,
we execute the commands:**

input files

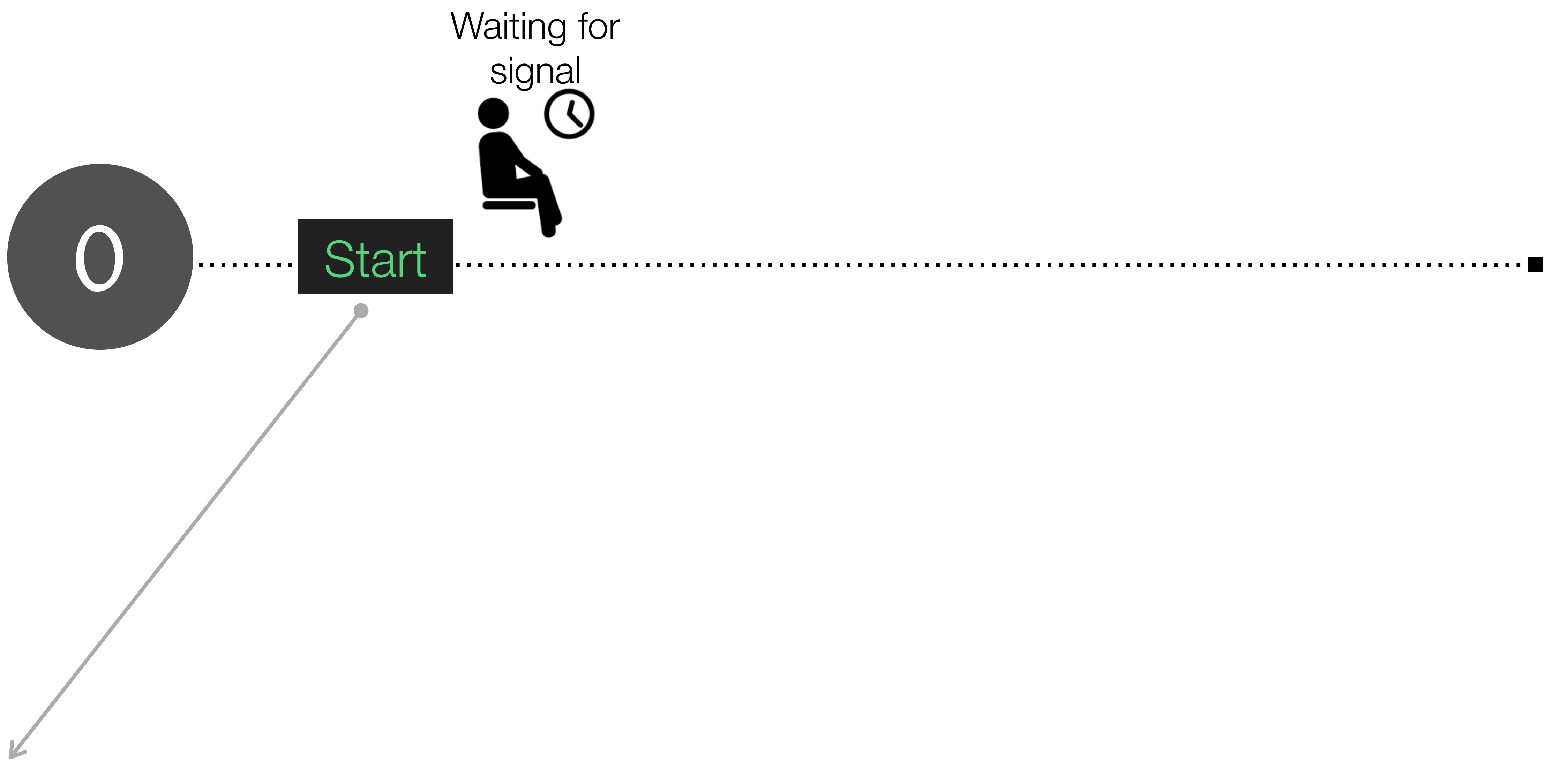
```
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 0 0.input
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 1 1.input
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 2 2.input
```

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

INPUT FILES,
OUTPUT FILES AND
SIGNALS

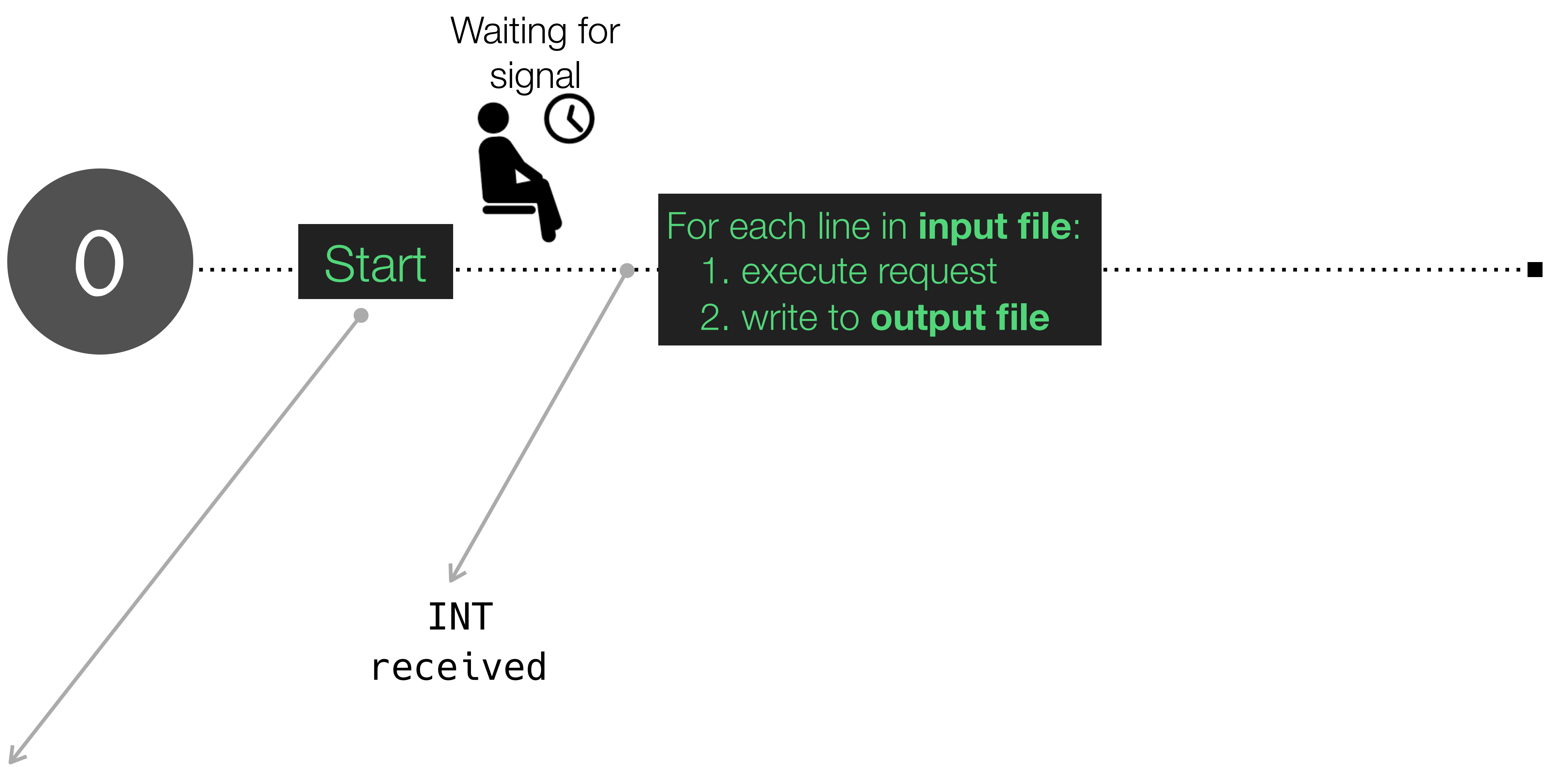
0

INPUT FILES,
OUTPUT FILES AND
SIGNALS



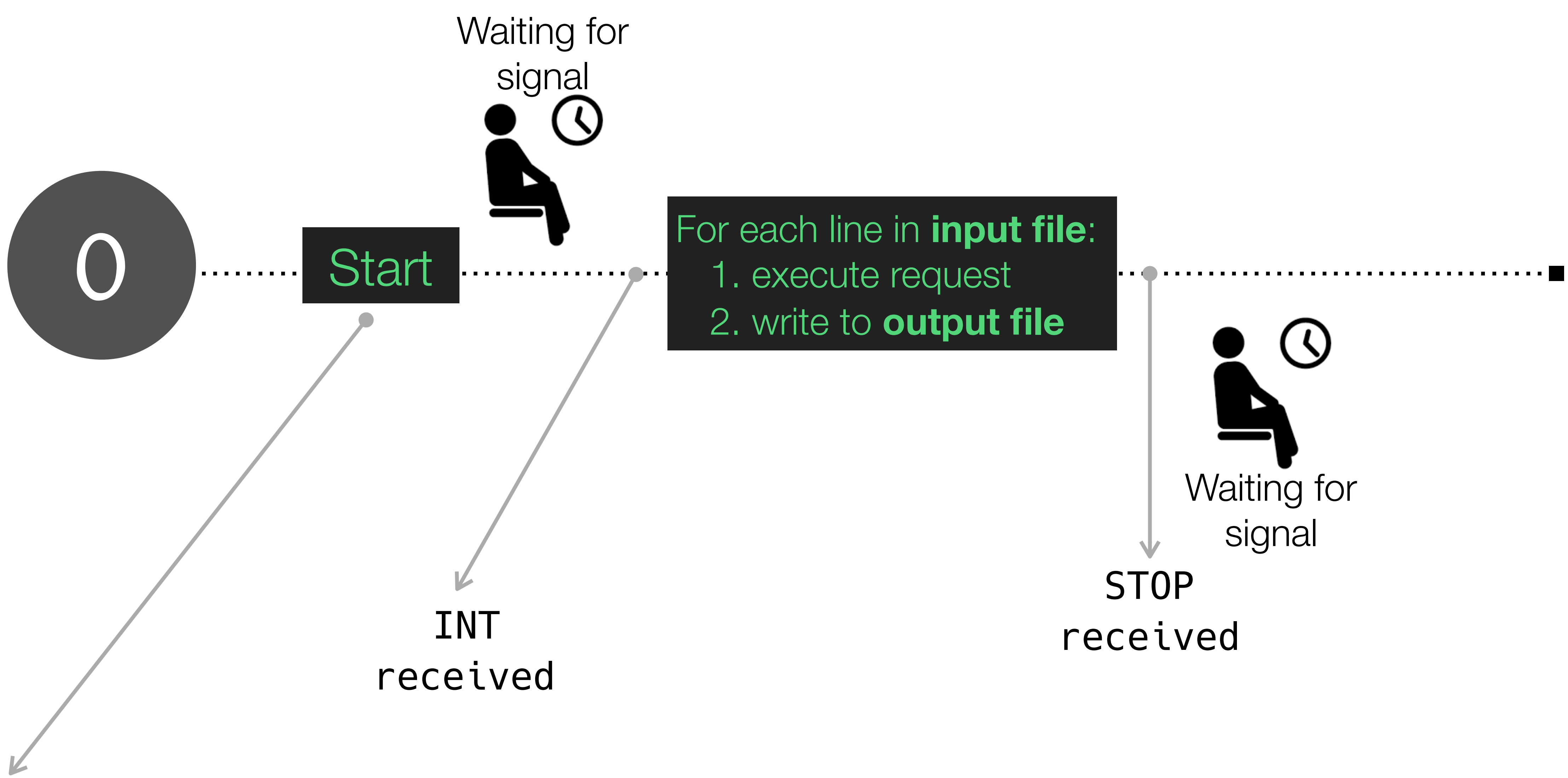

```
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 0 0.input
```

INPUT FILES, OUTPUT FILES AND SIGNALS




```
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 0 0.input
```

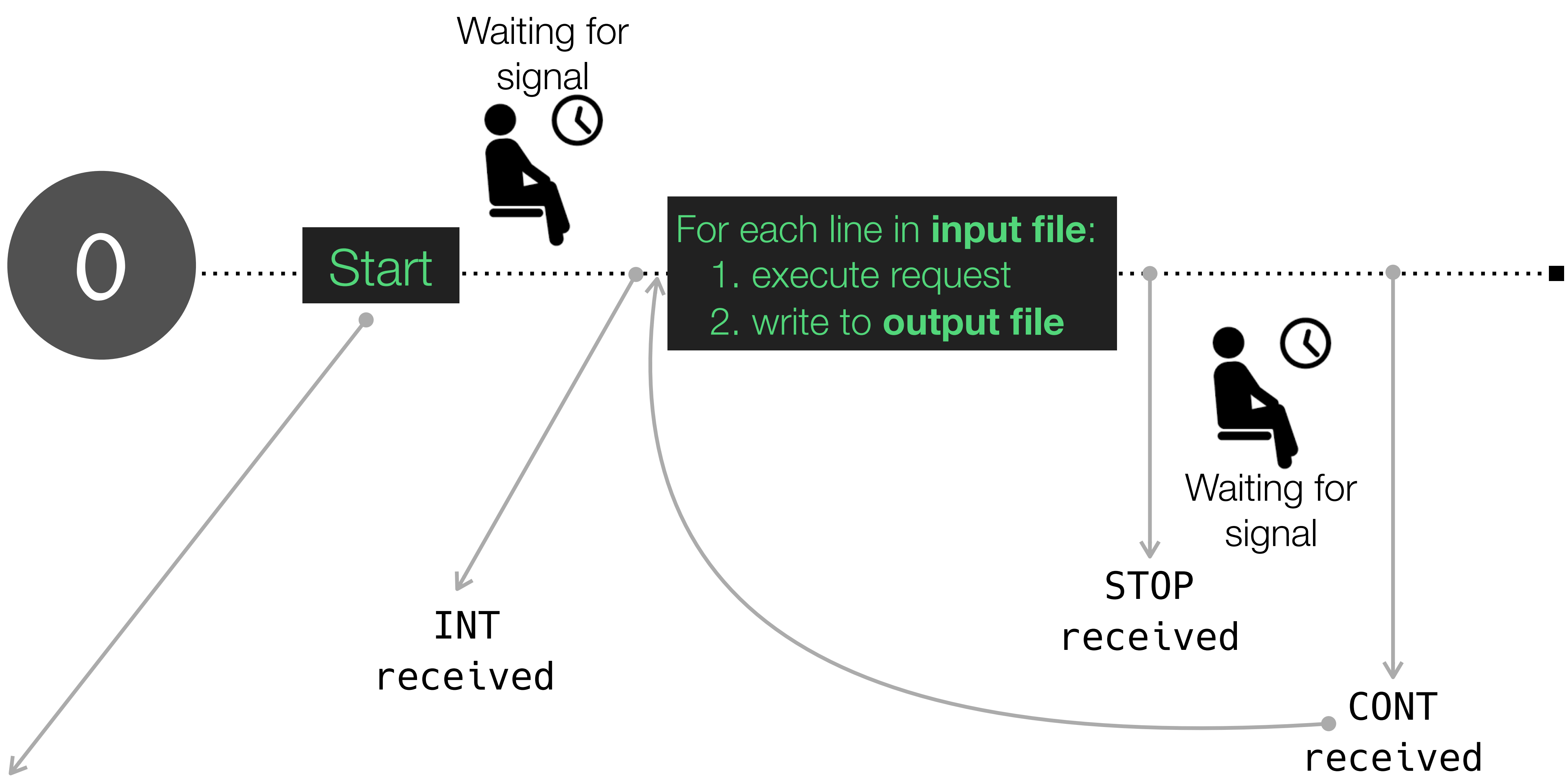
INPUT FILES, **OUTPUT** FILES AND **SIGNALS**




```
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 0 0.input
```

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

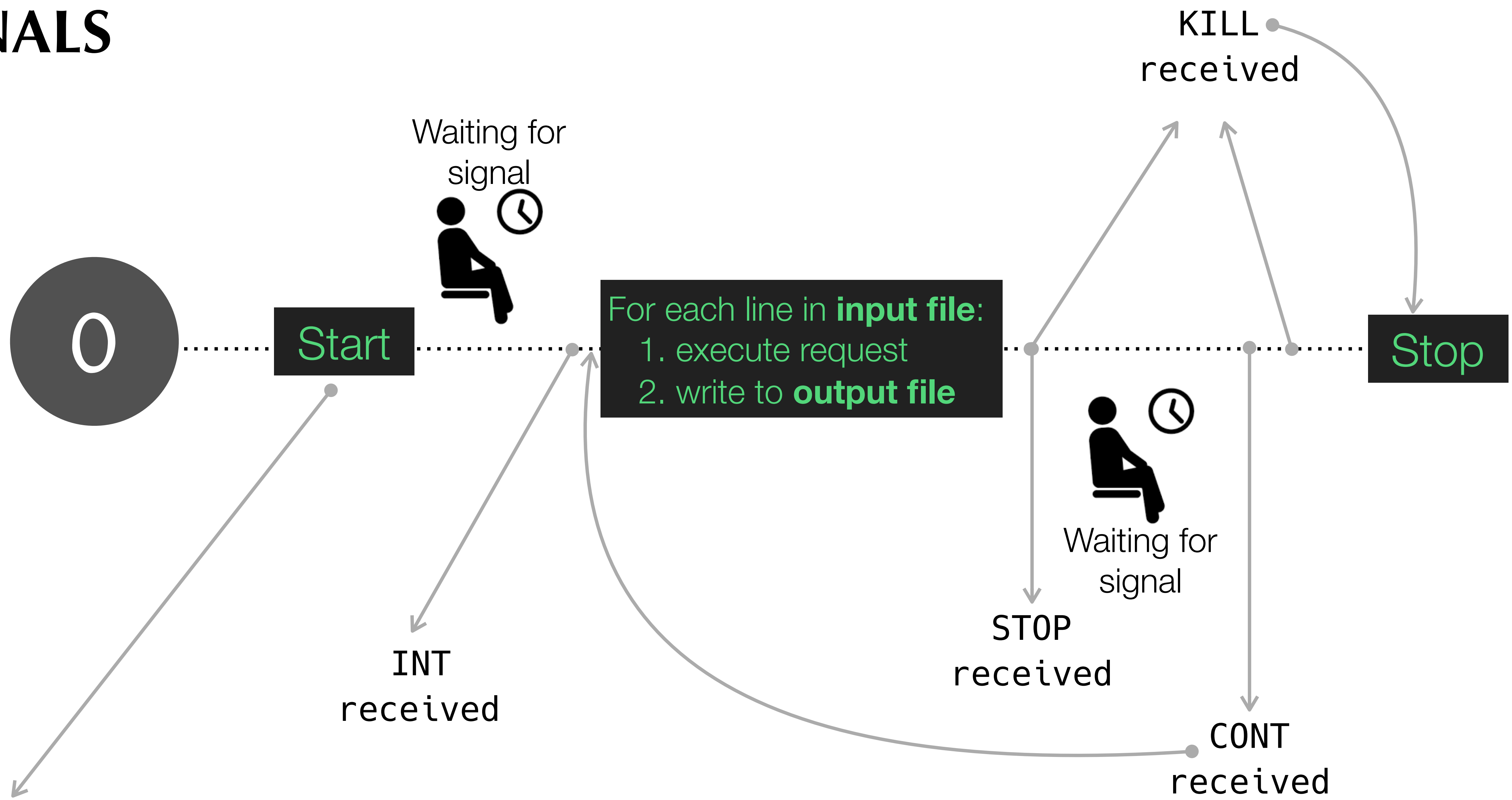
INPUT FILES,
OUTPUT FILES AND
SIGNALS




```
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 0 0.input
```

Technicalities

INPUT FILES, **OUTPUT** FILES AND **SIGNALS**



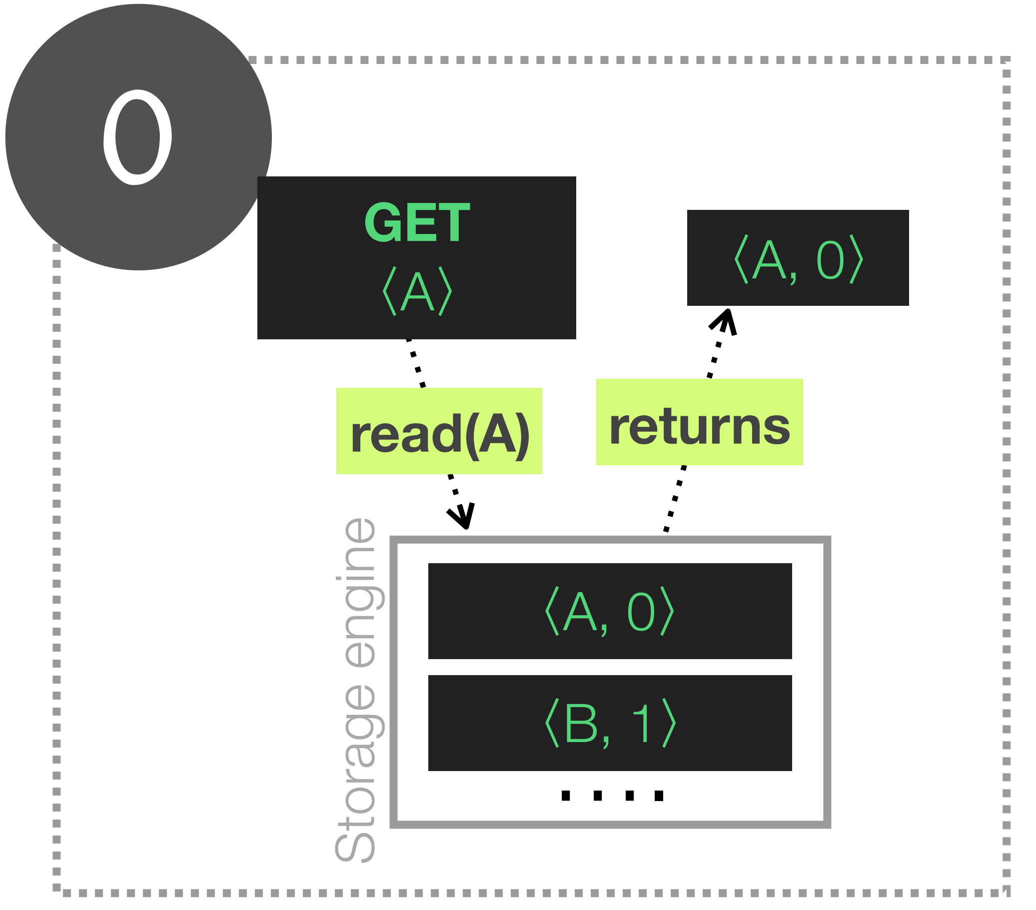

```
s4 127.0.0.1 8900 127.0.0.1 8901 127.0.0.1 8902 0 0.input
```

Technicalities

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

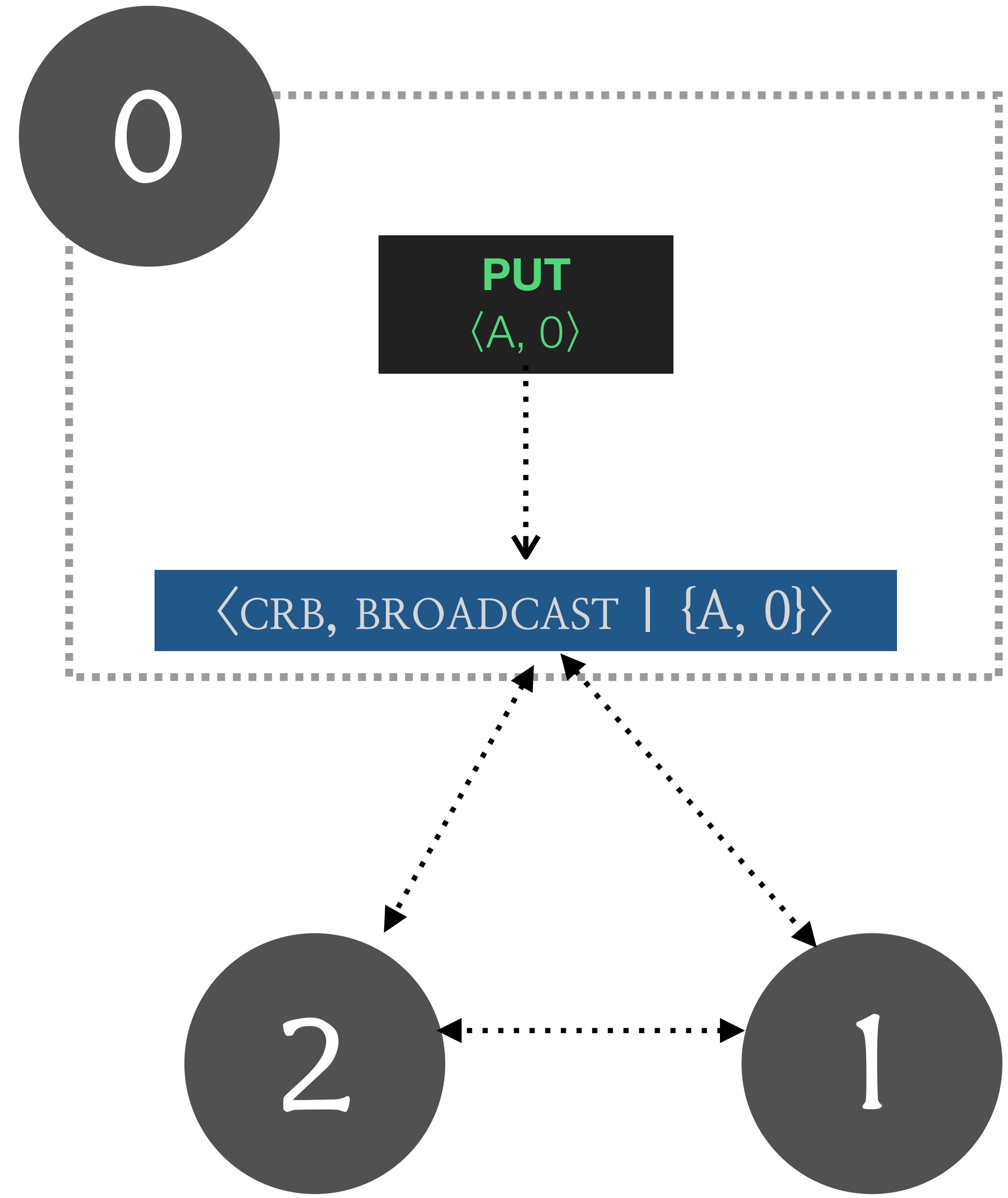
OPERATIONS — GET

- Translates to a local read operation
- Directly from the Storage engine
- Write the read value in the output file
- No need to contact the other nodes



OPERATIONS — PUT

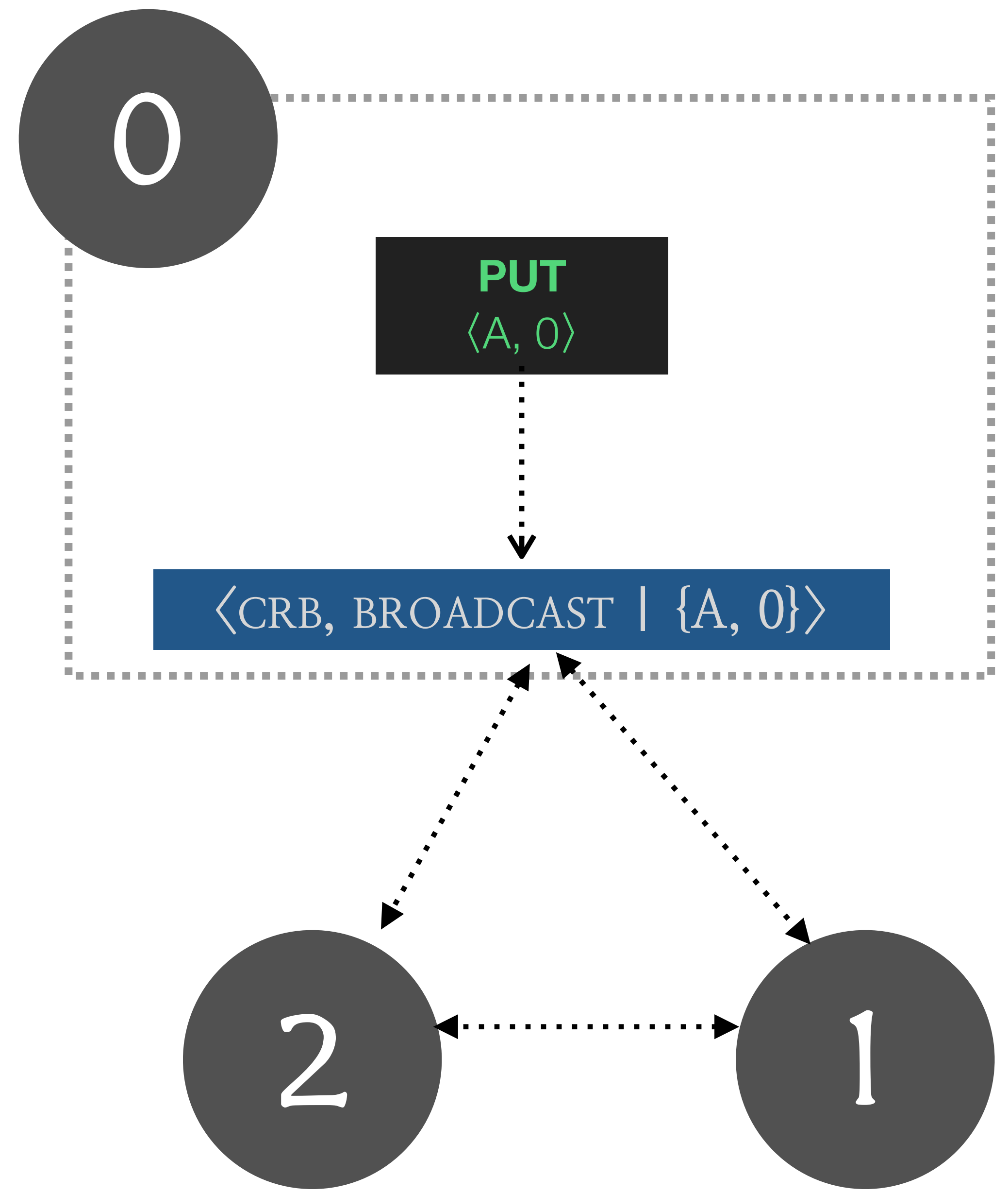
- Translates to a causal-order (crb) broadcast request
- Use the algorithm from the class
- No need to write anything (⊥) in the output file



ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

OPERATIONS — PUT

- Translates to a causal-order (crb) broadcast request
- Use the algorithm from the class
- No need to write anything (⊥) in the output file



What happens when a node delivers a message?

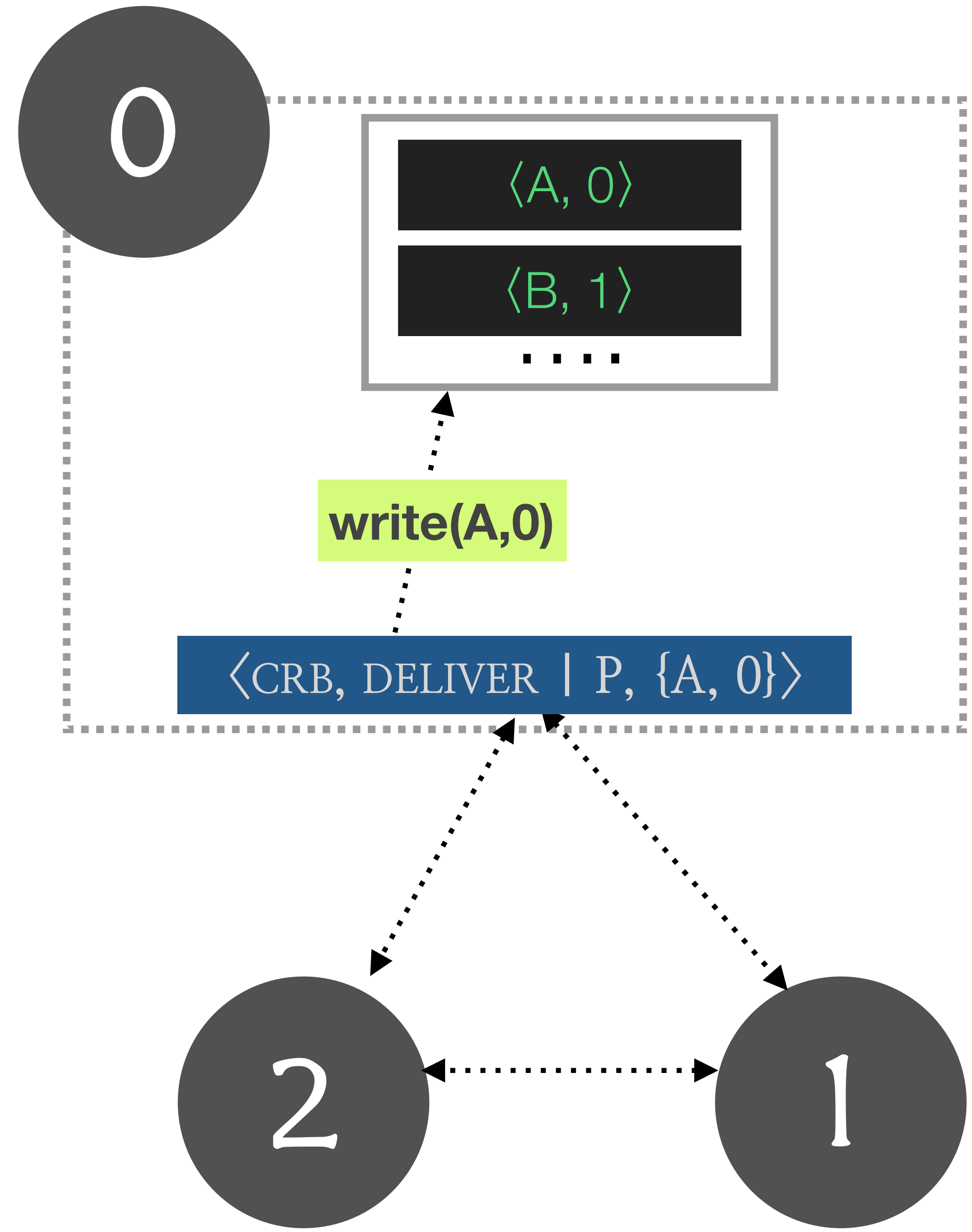
⟨CRB, DELIVER | P, {A, 0}⟩

Technicalities

OPERATIONS

⟨CRB, DELIVER | P, {A, 0}⟩

- Triggers an update in the storage engine

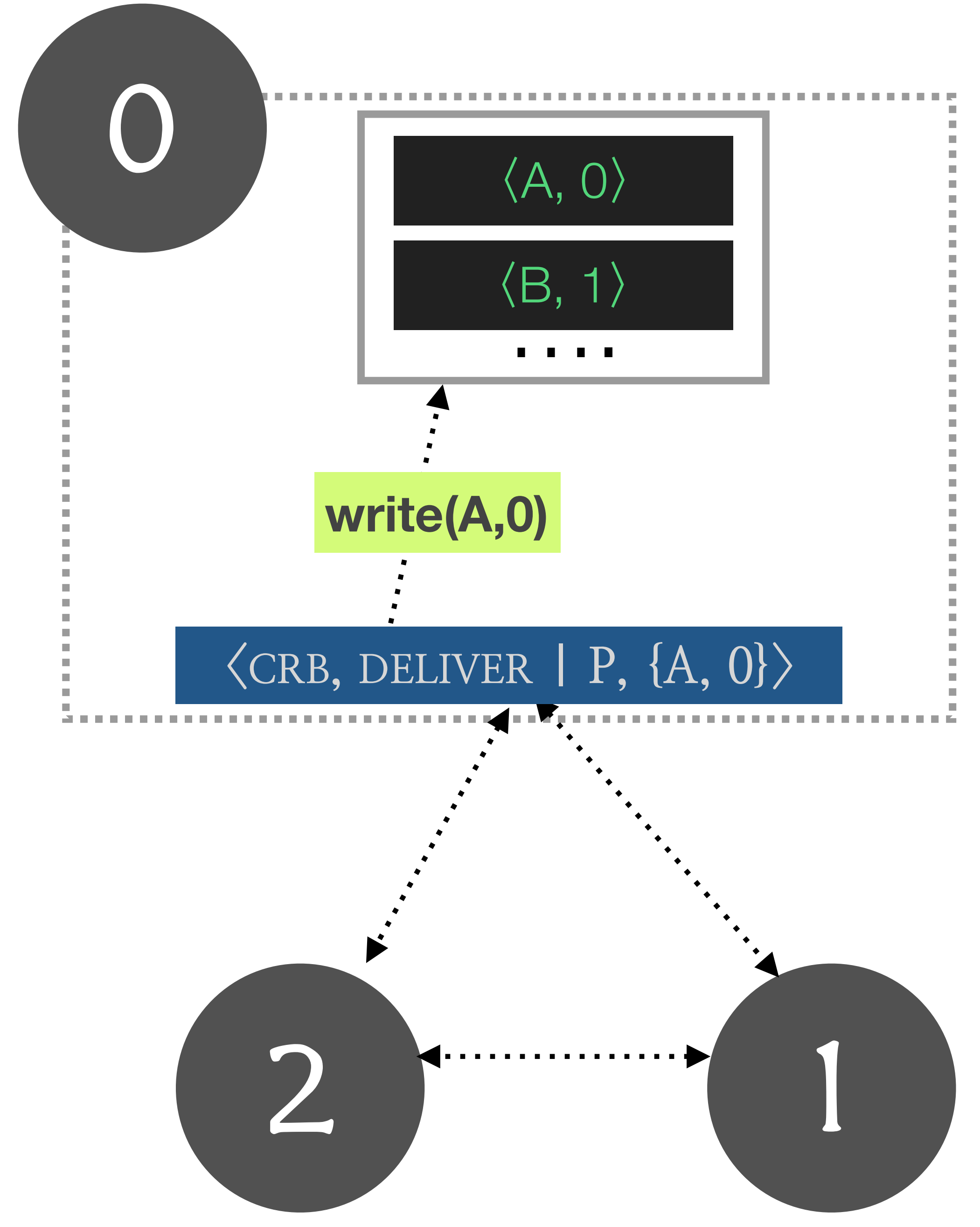


ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

OPERATIONS

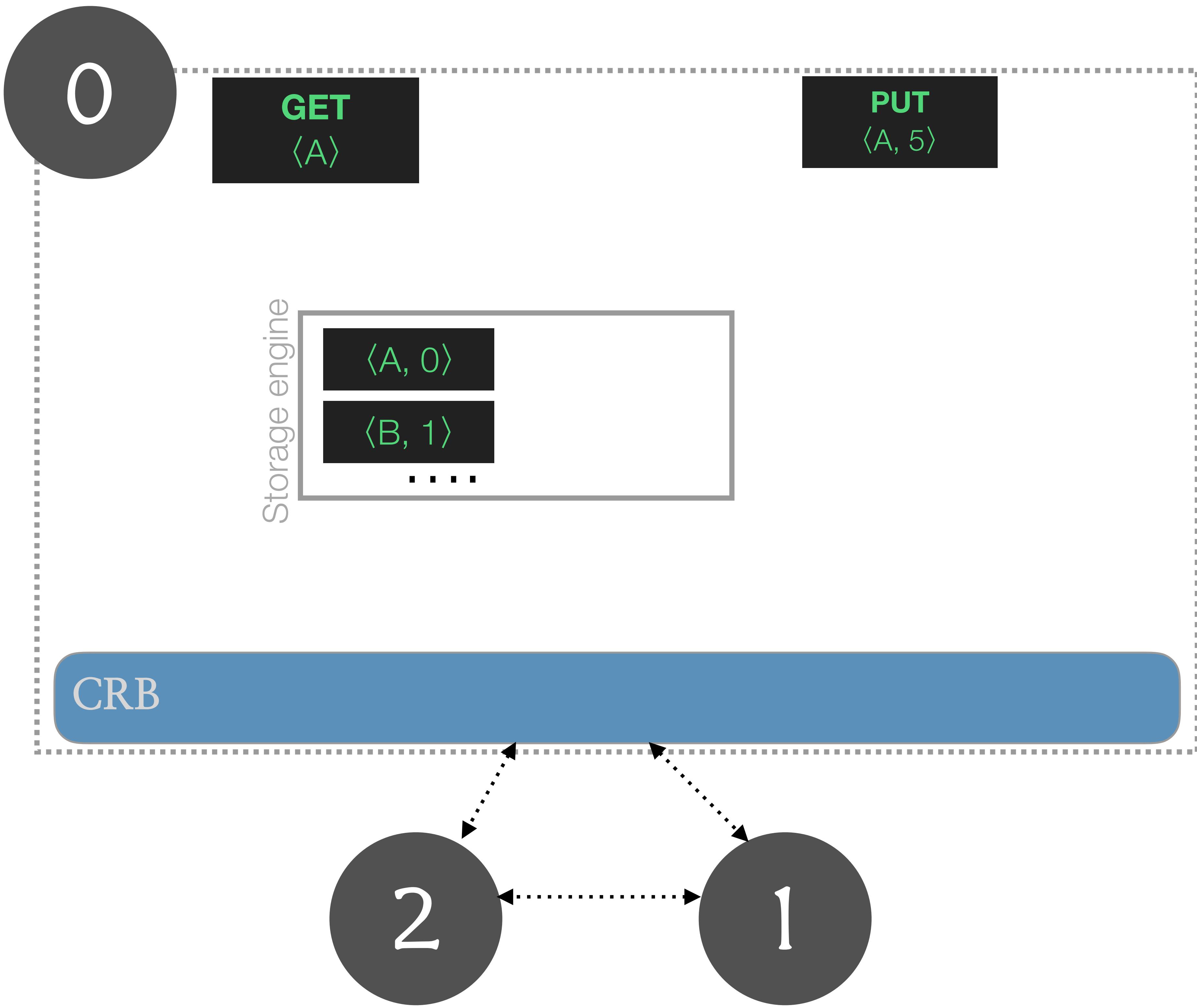
⟨CRB, DELIVER | P, {A, 0}⟩

- Triggers an update in the storage engine



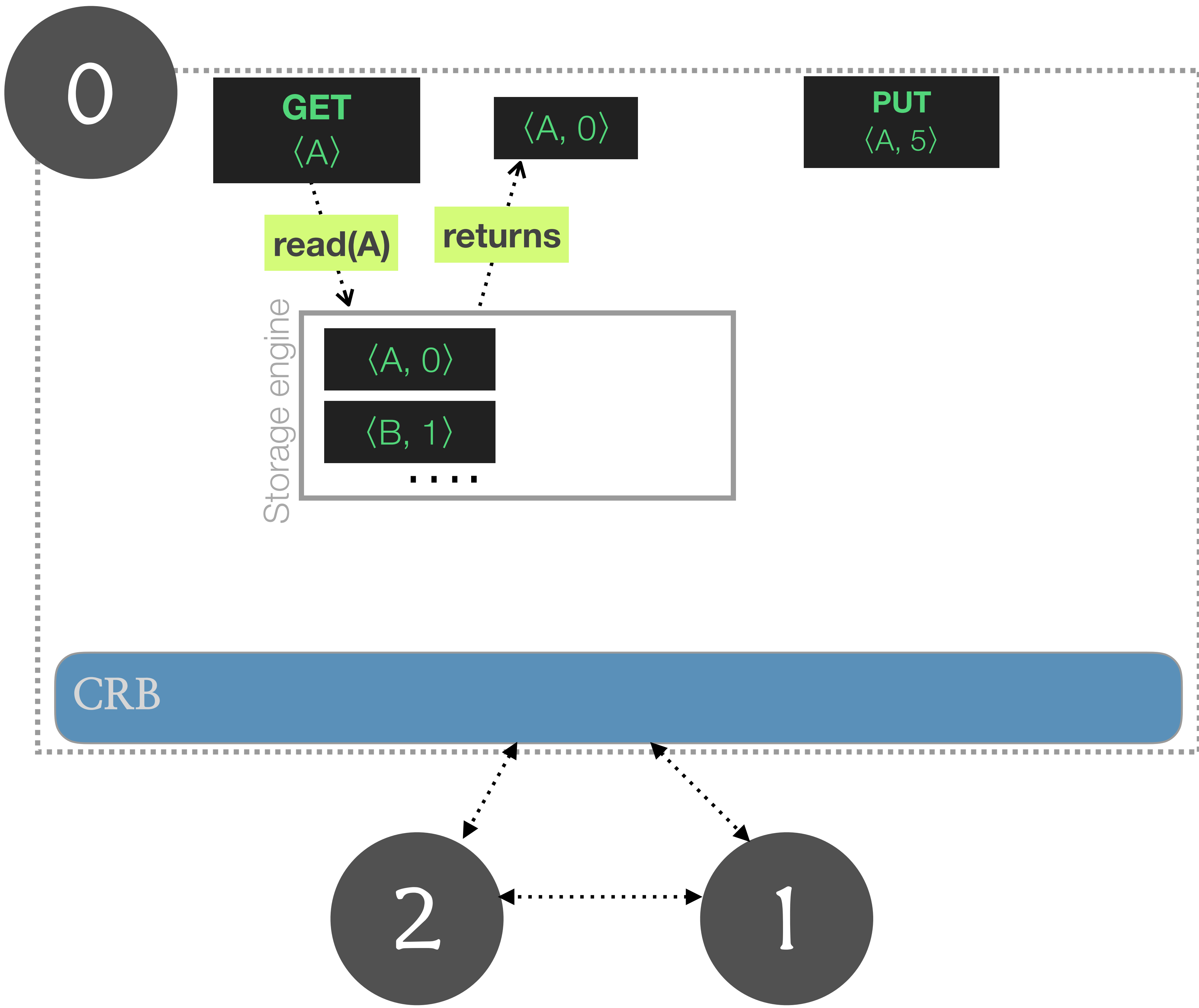
Now let's put all the pieces together…

OVERVIEW OF A NODE

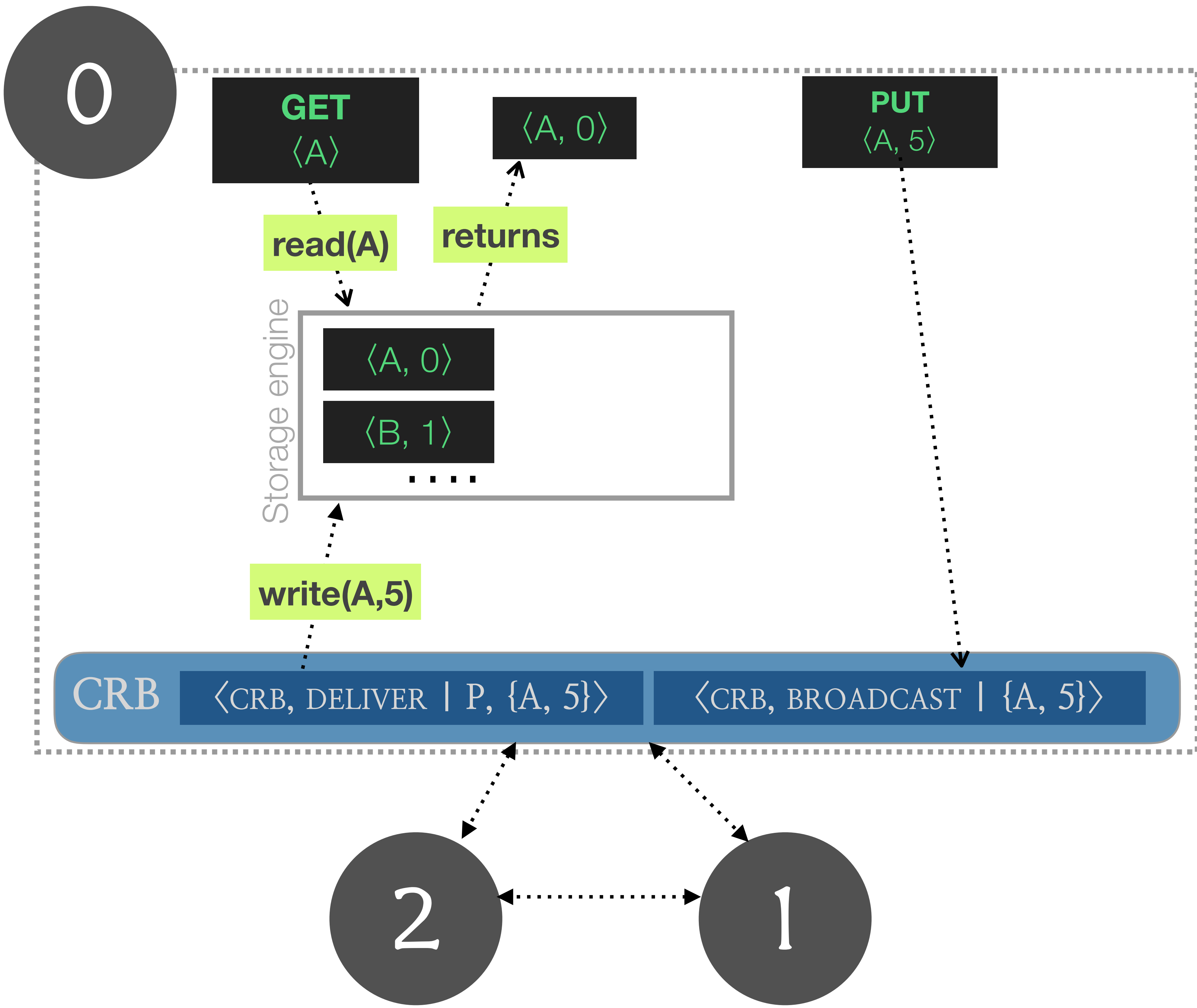


ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

OVERVIEW OF A NODE

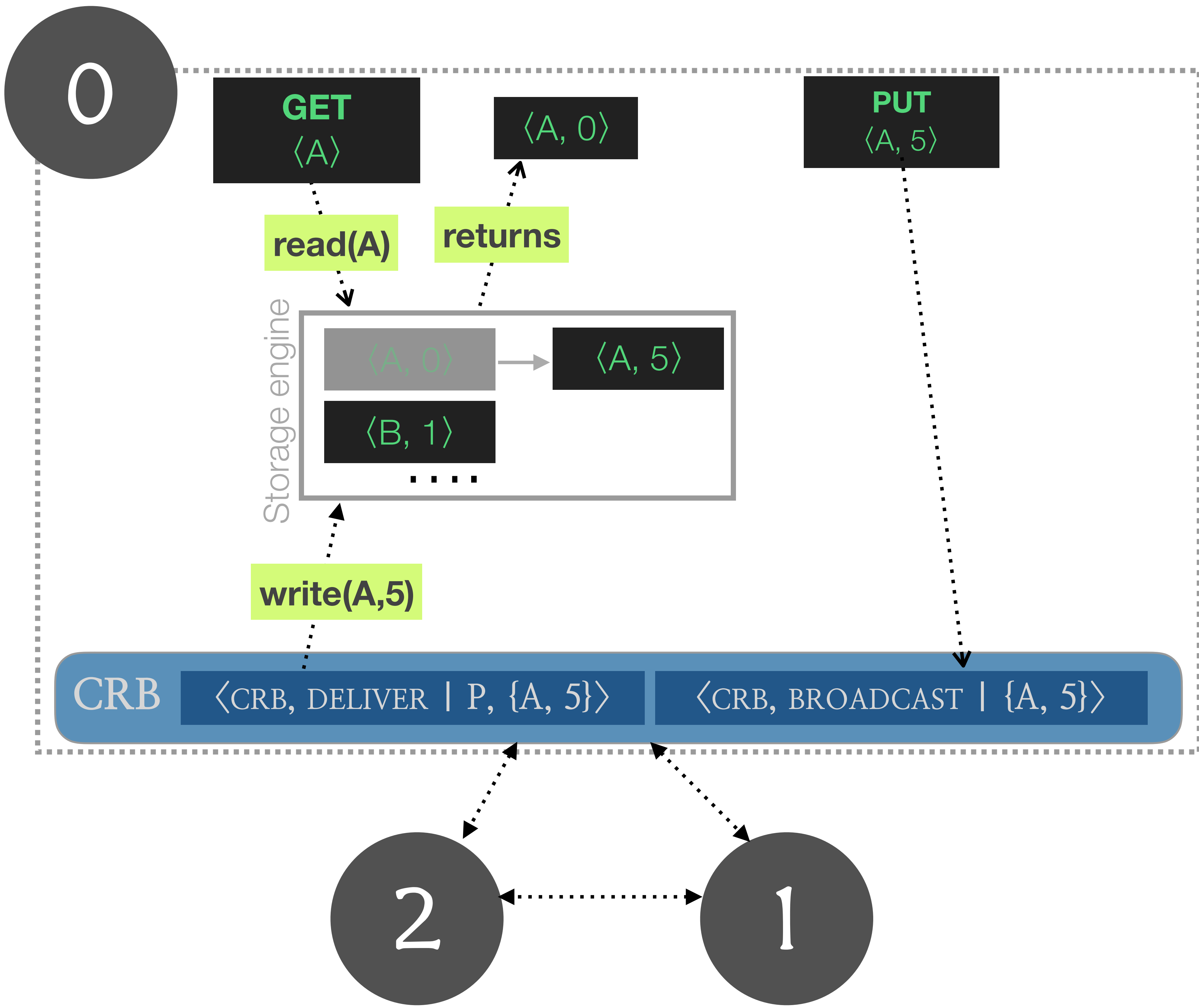


OVERVIEW OF A NODE



ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

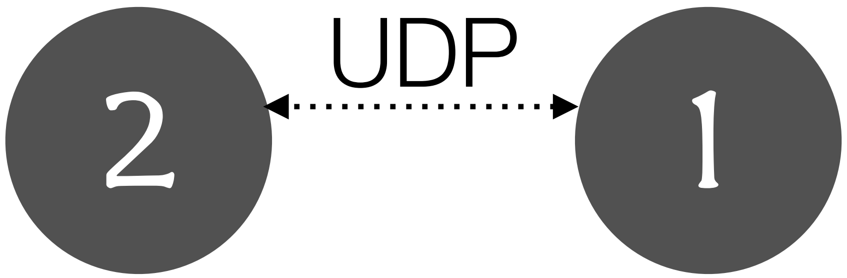
OVERVIEW OF A NODE



Technicalities

MESSAGES AND **OBJECTS**

- Use UDP
- Prohibited:
  - TCP (or any other reliable communication protocol)
  - Any other IPC

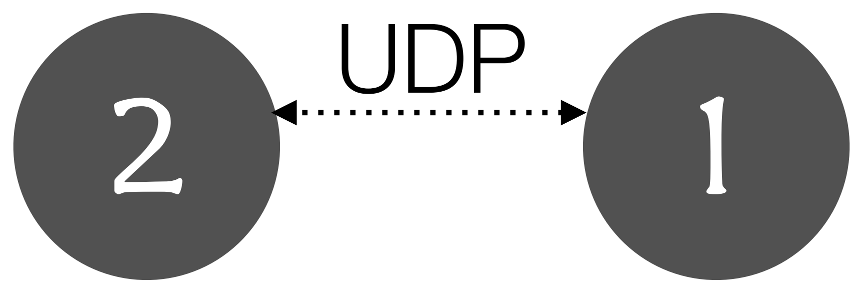


- For simplicity sake:
  - Both keys and values are short ASCII strings
  - At most 10 chars each

Technicalities

MESSAGES AND OBJECTS

- Use UDP
- Prohibited:
  - TCP (or any other reliable communication protocol)
  - Any other IPC



⟨Key, Value⟩

- For simplicity sake:
  - Both keys and values are short ASCII strings
  - At most 10 chars each

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

TESTING AND GRADING

```
put,A,1        /* write to object A value 1  */
put,B,2        /* write to object B value 2  */
get,A          /* read the value of object A */
get,B          /* read the value of object B */
```

.input	.output
put,A,1	⊥ (ignore in the output file)
put,B,2	⊥ (ignore in the output file)
get,A	A,1
get,B	B,2

<u>The results of all the GET operations</u>

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

TESTING AND GRADING

For each line in **input file**:
1. execute request
2. write to **output file**

INPUT

```
put,A,1        /* write to object A value 1  */
put,B,2        /* write to object B value 2  */
get,A          /* read the value of object A */
get,B          /* read the value of object B */
```

INPUT + OUTPUT

.input	.output
put,A,1	⊥ (ignore in the output file)
put,B,2	⊥ (ignore in the output file)
get,A	A,1
get,B	B,2

The results of all the GET operations

S4

Overview:

1. Setup & System model

2. Operations

3. Goals

4. Technicalities
   - Java template
   - Compilation
   - Testing
     1. Correctness
     2. Performance

5. One last look at causal consistency

S4

Overview:

1. Setup & System model
2. Operations
3. Goals
4. Technicalities
   - Java template
   - Compilation
   - Testing
     1. Correctness
     2. Performance

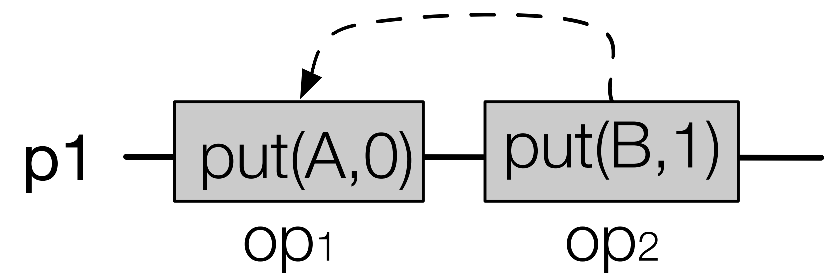
NOT COVERED
SEE PROJECT DOCUMENTATION

5. One last look at causal consistency

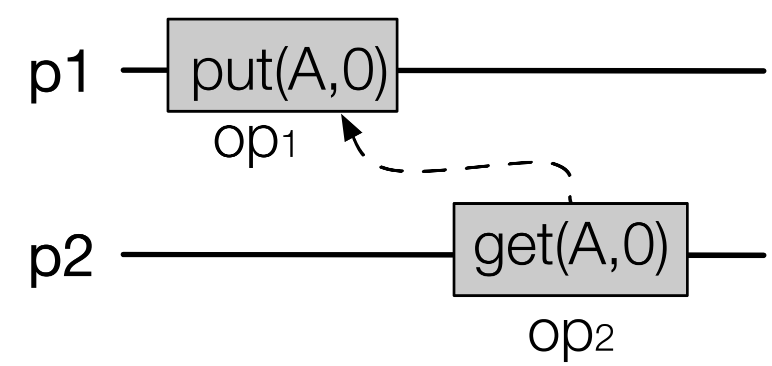
Causal consistency — one last look

- Given two operations, op1 and op2

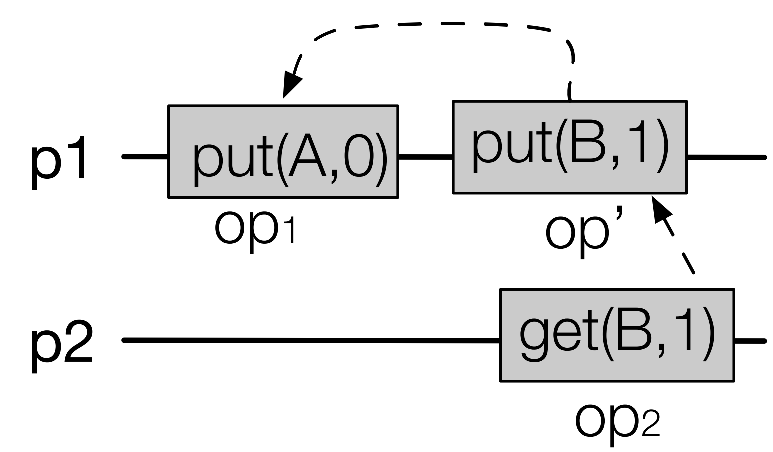
- op1 → op2 (causally precedes)
  if any of the following three cases applies:



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

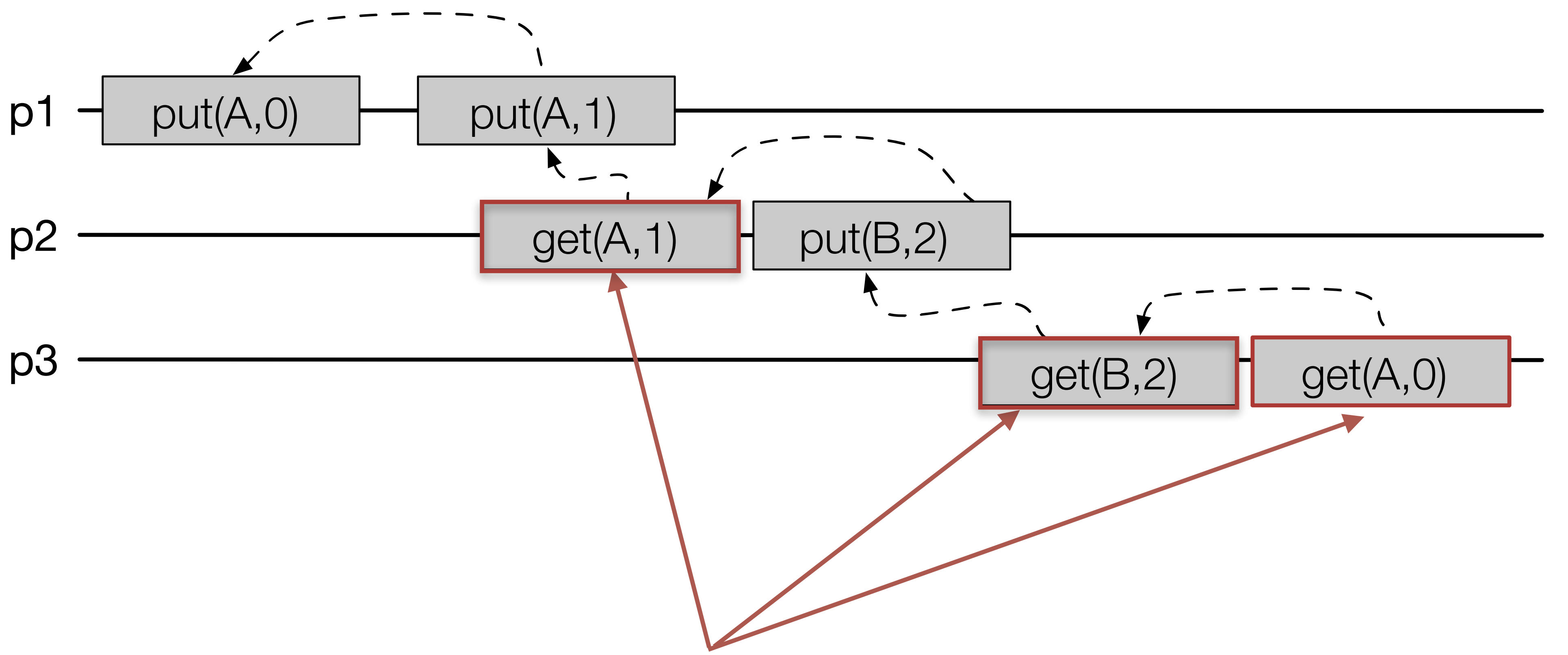


(b) Local: A process invokes the PUT operation op1 and another process invokes the GET operation op2, where op2 observes the written value of op1



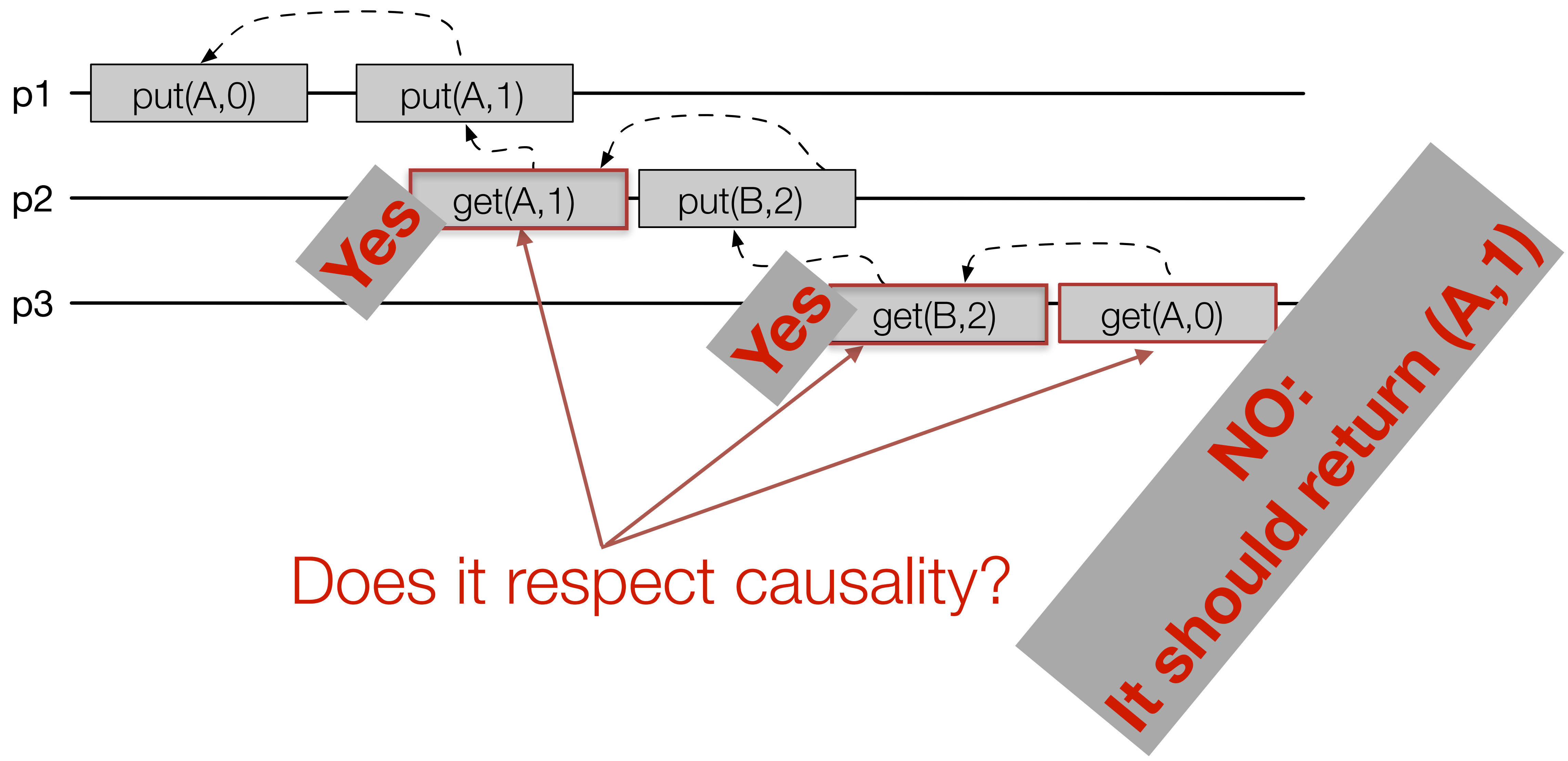
(c) Transitivity: There exists an intermediate operation op′ such that op1 → op′ and op′ → op2.

Causal consistency — one last look



Does it respect causality?

Causal consistency — one last look



Further reading

1. **Bitcoin — not needed for exam**
   - Nakamoto, Satoshi (24 May 2009).
     "Bitcoin: A Peer-to-Peer Electronic Cash System”.
     https://bitcoin.org/bitcoin.pdf
     (canonical Bitcoin)

Causal Broadcast

2. **Social networking (= bonus project)**
   - CS-451 Bonus Project (2015).
     “A Storage System for Social Networks”.
     https://github.com/LPD-EPFL/da15-s4
   - Ahamad, M., Neiger, G., Burns, J. E., Kohli, P., & Hutto, P. W. (1995). **Causal memory: definitions, implementation, and programming. (§5).** Distributed Computing, 9(1).