Los generales bizantinos y Blockchain

SISTEMAS DISTRIBUIDOS E INTERNET

Vicente García Díaz
garciavicente@uniovi.es

¿Computación distribuida?

- Múltiples **nodos** activos al mismo tiempo
- Hardware y software con ciertos grados de libertad
- …compartiendo algunos recursos o información común
- …por lo que es necesario que haya una **coordinación** y una **interacción**

Sistemas distribuidos tolerantes a fallos

- Internet, intranets, …
- Superordenadores
- Redes de sensores
- Redes de robots
- Banca electrónica
- Bases de datos
- Sistemas de reserva
- Sistemas peer-to-peer
- Videojuegos
- …

Objetivo

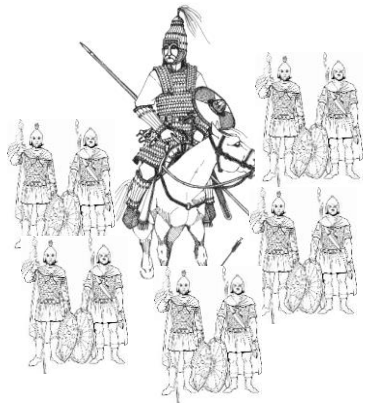
¿Cómo podemos diseñar y construir un sistema distribuido que pueda sobrevivir el peor escenario posible en cuanto a fallos?

- Podemos clasificarlos en dos grandes bloques:
  - Fallos “no bizantinos”
    - Fallos en componentes internos o externos que interfieren en el buen funcionamiento del sistema
    - ¿Qué hacemos?
      - Reinicio o utilización de otros nodos redundantes
  - Fallos “bizantinos”
    - Fallo debido a nodos “traidores” que envían mensajes conflictivos
    - ¿Qué hacemos??????

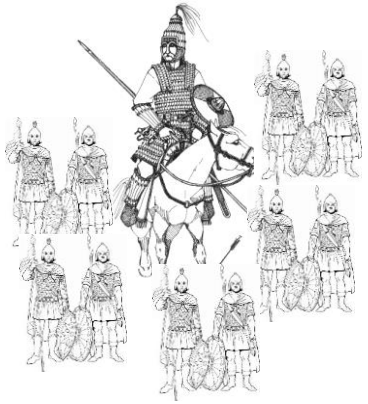
Problema de los dos generales

¿Cuándo atacamos?

- Sólo se pueden comunicar a través de un mensajero y deben sincronizarse



Mensaje: Atacar el lunes



Mensaje: OK

La paradoja de los dos generales

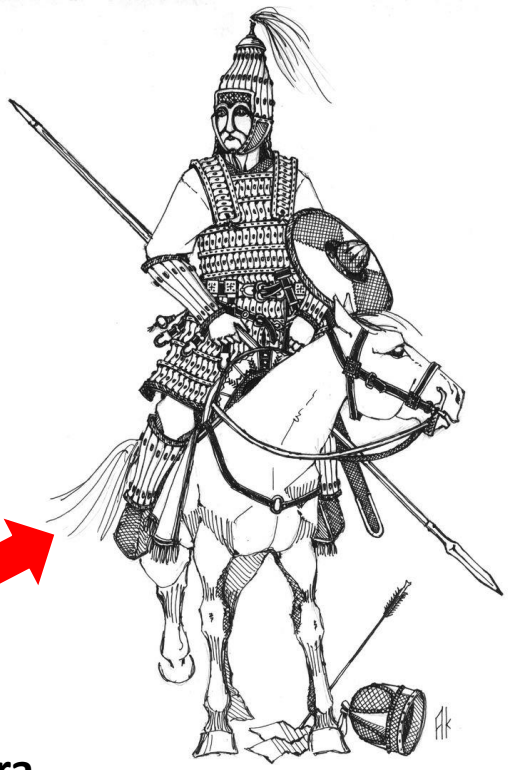
Esto es una red de computadores no fiable

Quiere atacar

Si me respondes, atacaremos el martes a las 9:00

OK, atacaremos el martes a las 9:00 entonces

Ok, confirmado. Atacaremos el martes



Esto son aplicaciones informáticas

Es suficiente un mensaje satisfactorio con una respuesta satisfactoria para llegar a un acuerdo, pero…

¿Cómo se aborda este problema?

- Es imposible estar 100 % seguros
  - The Two Generals Problem [Akkoyunlu, Ekanadham & Huber, 1975]

- Debemos aceptar la incertidumbre del canal de comunicaciones para mitigarla hasta un grado aceptable

- Si solo desconfiamos de la red:
  - Podemos, por ejemplo, enviar 50 mensajeros con el mismo mensaje y esperar a que al menos uno lo entregue en su destino
- Si desconfiamos de los participantes de la red:
  - Podemos, por ejemplo, enviar 50 mensajeros con el mismo mensaje y esperar a ver las respuestas

Problema de los generales bizantinos

Generales bizantinos

- Problema planteado en
  - The Byzantine Generals Problem [Lamport, Shostack, Peace, 1982]

- Plantea de forma metafórica un problema que se da entre un conjunto de sistemas o componentes informáticos que tienen un objetivo común

- La idea es encontrar un plan de acción común a partir de una estructura jerárquica, donde uno de los sistemas proporciona una orden a partir de la cual el resto de los sistemas tienen que operar

- Puede que alguno de los sistemas no sea fiable

Planteamiento

- Supongamos un escenario de guerra con n **generales bizantinos** que están asediando una ciudad desde diferentes lugares

- Llamaremos **comandante** al único general que puede dar la orden de atacar o retirarse

- Llamaremos **tenientes** a los demás generales

- Los generales se comunican a través de mensajeros para indicar una de las dos posibles órdenes del comandante: “atacar” o “retirarse”.

Problema

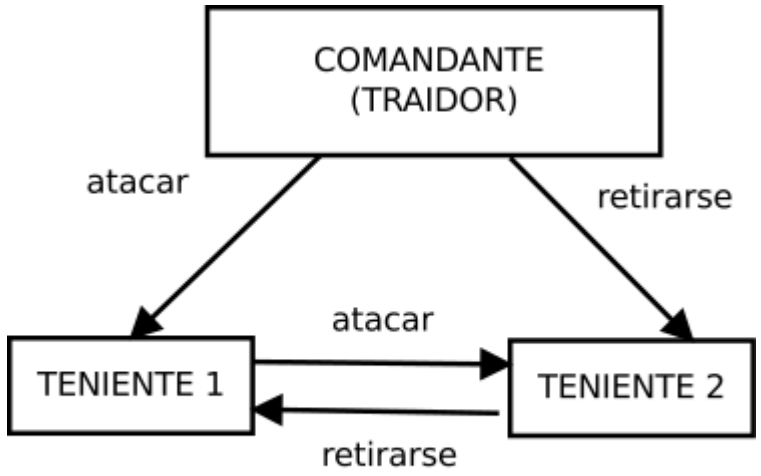
- Varios generales serán **leales**
  - Pero habrá uno o más generales que pueden ser **traidores** que ofrezcan información errónea

- ¿Qué buscamos?
  - **Si el comandante es leal, todos los tenientes leales llevarán a cabo su orden**
  - **Todos los tenientes leales tomarán la misma decisión**

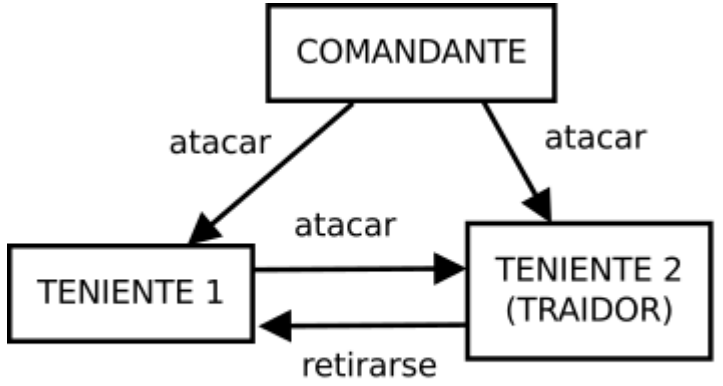
- Consideraciones
  - Cada mensaje que se envía llega correctamente
  - Cada receptor de un mensaje conoce quién lo envió
  - Se detectará cuando no hay un mensaje

Una solución

- **Todos se pueden comunicar con todos**
- Caso de 3 generales ($n = 3$)



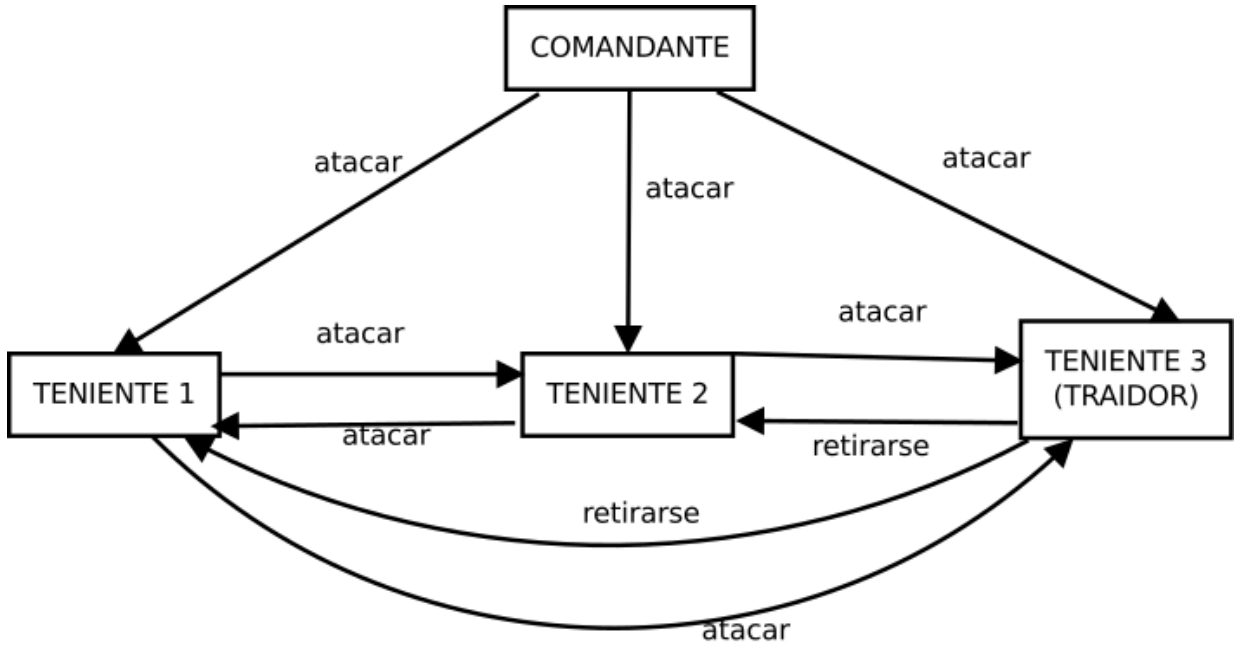
DILEMA TENIENTE 1 ¿QUIÉN ES EL TRAIDOR?



DILEMA TENIENTE 1 ¿QUIÉN ES EL TRAIDOR?

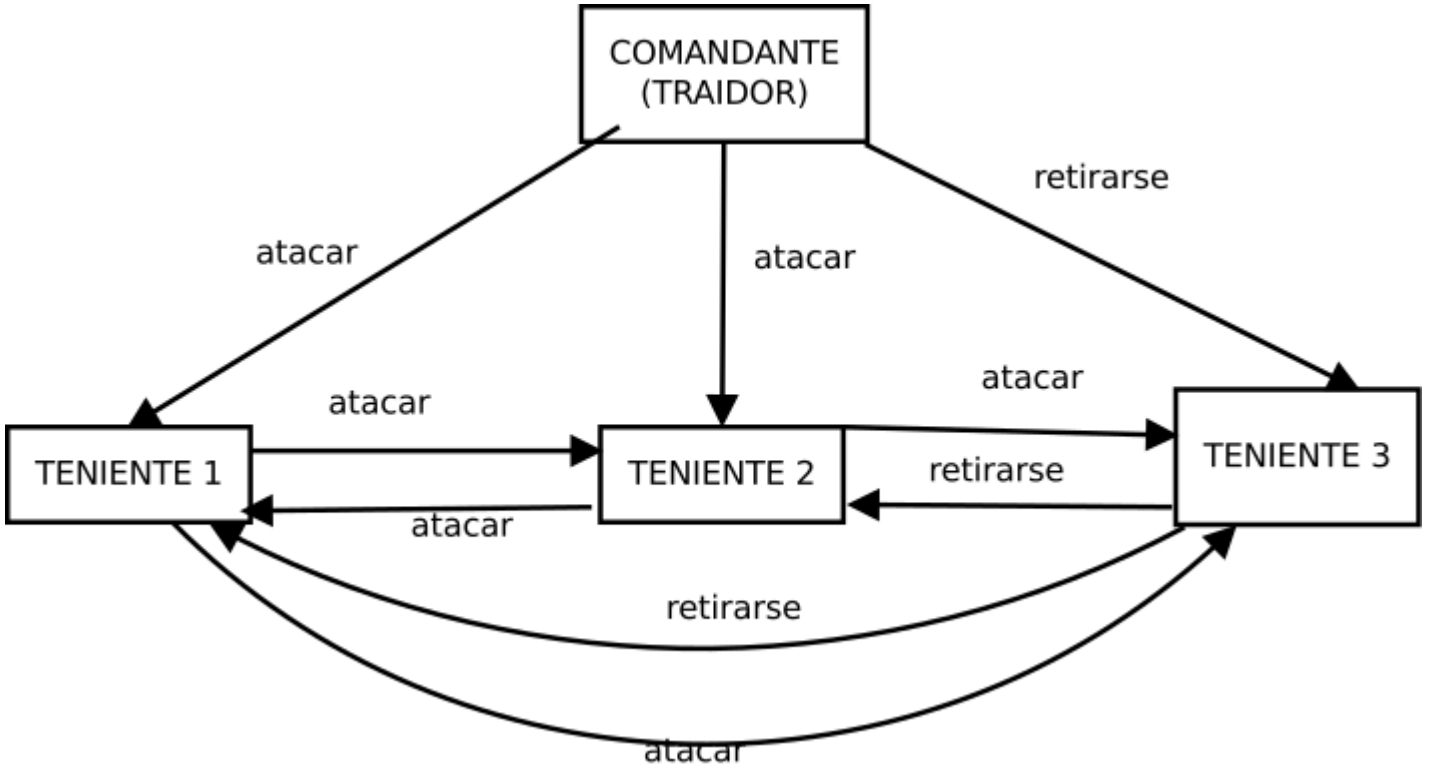
Una solución (II)

- **Todos se pueden comunicar con todos**
- Caso de 4 generales ($n = 4$)
- Se decidirá por consenso según la función Mayoría: $M(v1, v2, v3)$



Una solución (III)

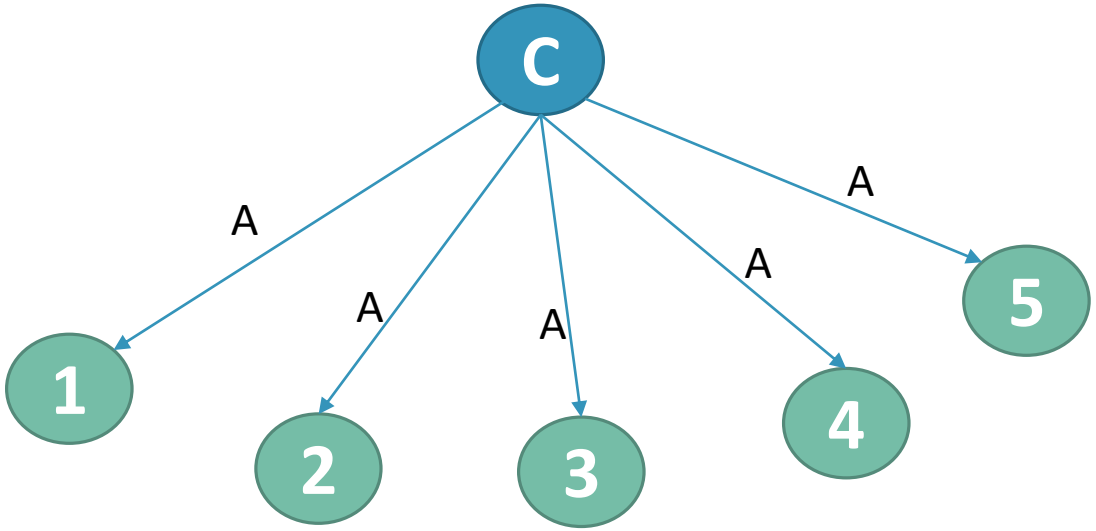
- **Todos se pueden comunicar con todos**
- Caso de 4 generales ($n = 4$)



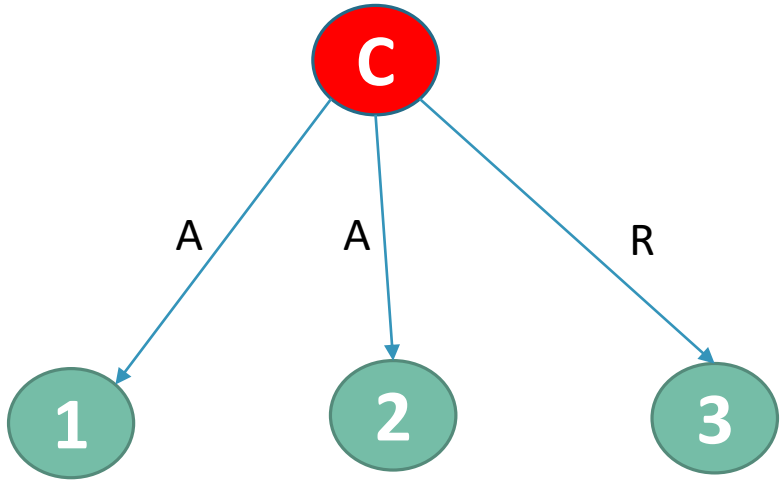
A tener en cuenta

Generalizando a n generales, si tenemos m traidores necesitamos que $\boldsymbol{n \geq 3m + 1}$

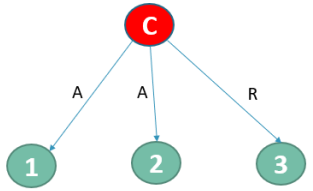
Ejemplo para m = 0



Ejemplo para n = 4 y m = 1



Ejemplo para n = 4 y m = 1



Teniente 1

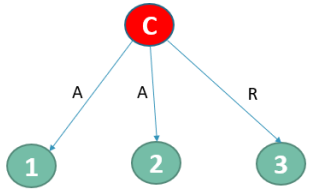
Oral messages

OM(1)

OM(0)

¿Consenso entre los generales leales?				
	OM(1)	OM(0)		Mayoría
T1	A	A	R	A

Ejemplo para n = 4 y m = 1



Teniente 2

OM(1)

A

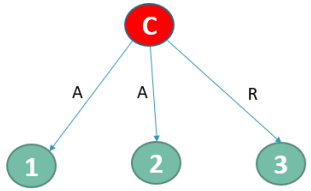
OM(0)

A

R

¿Consenso entre los generales leales?				
	OM(1)	OM(0)		Mayoría
T1	A	A	R	A
T2	A	A	R	A

Ejemplo para n = 4 y m = 1



Teniente 3

OM(1)

R

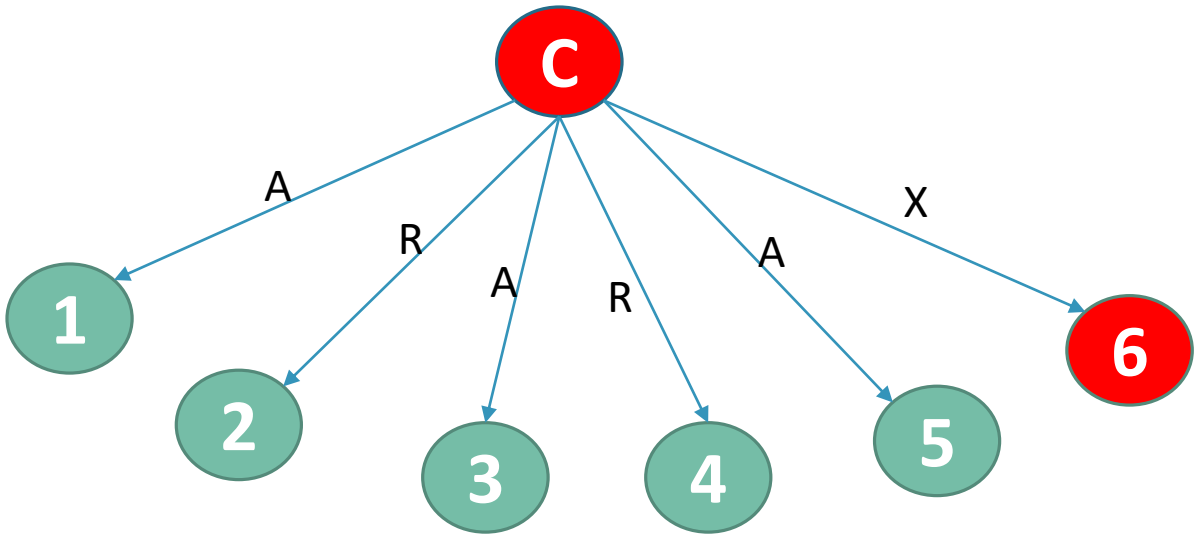
OM(0)

A

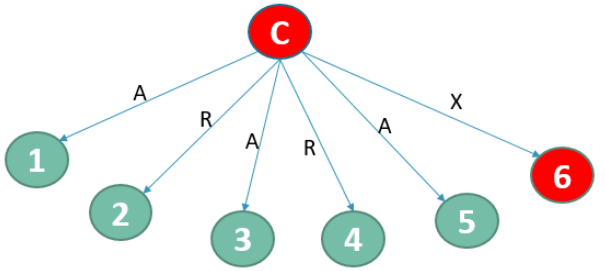
A

¿Consenso entre los generales leales?				
	OM(1)	OM(0)		Mayoría
T1	A	A	R	A
T2	A	A	R	A
T3	R	A	A	A

Ejemplo para n = 7 y m = 2

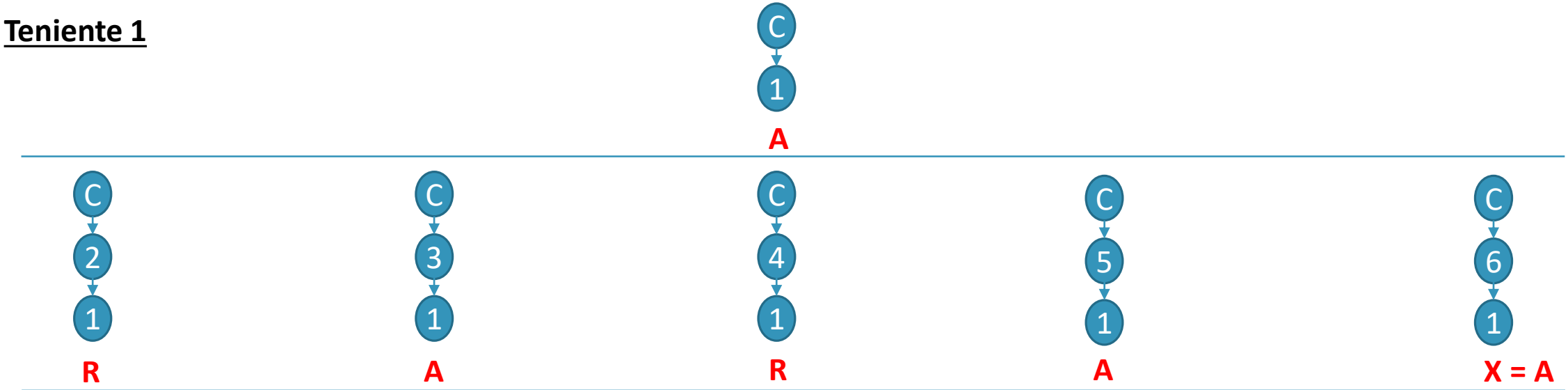


Ejemplo para n = 7 y m = 2



Teniente 1

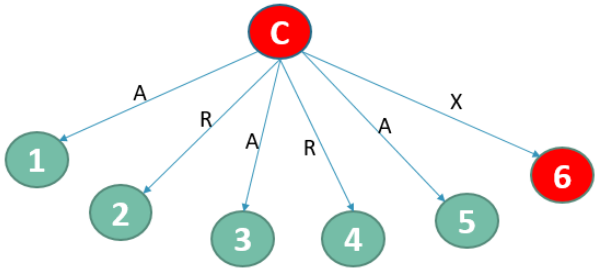
OM(2)



OM(1)

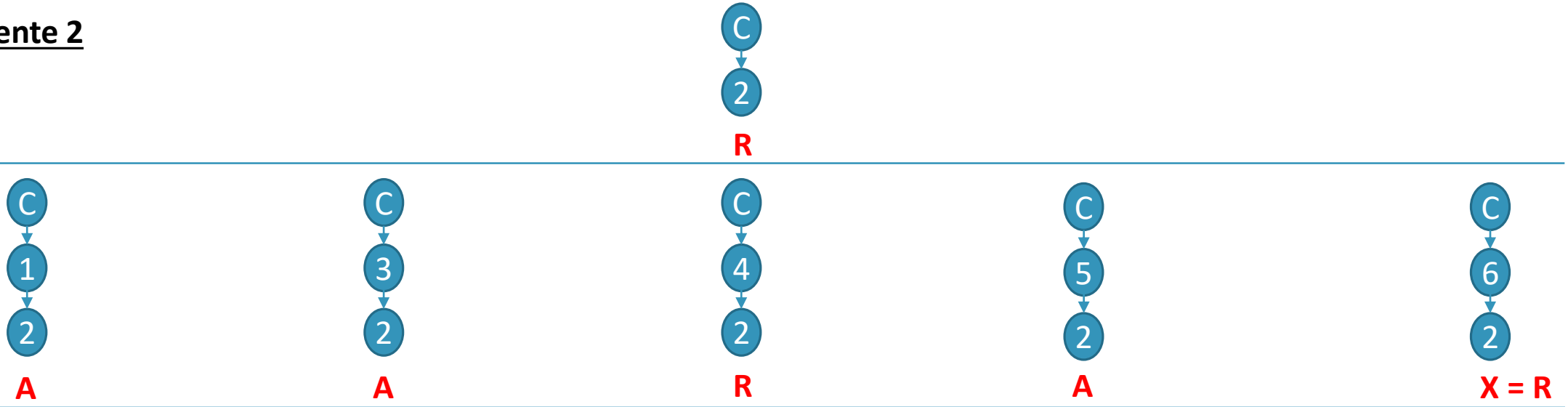
¿Consenso entre los generales leales?							
	OM(2)	OM(1)					Mayoría
T1	A	R	A	R	A	X = A	A

Ejemplo para n = 7 y m = 2



Teniente 2

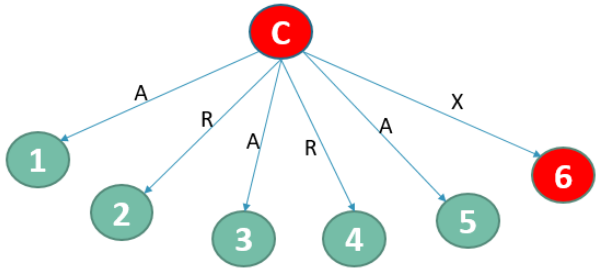
OM(2)



OM(1)

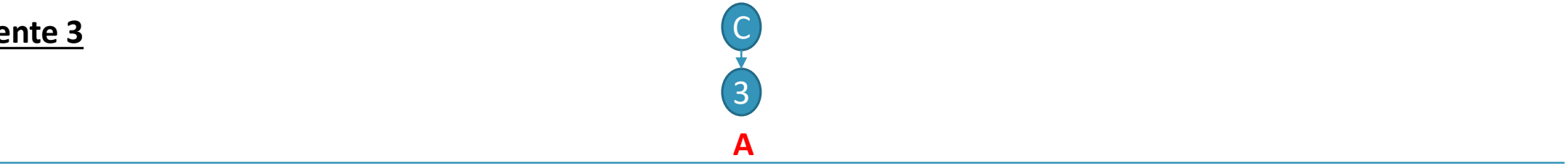
¿Consenso entre los generales leales?							
	OM(2)	OM(1)					Mayoría
T1	A	R	A	R	A	X = A	A
T2	R	A	A	R	A	X = R	Nada

Ejemplo para n = 7 y m = 2

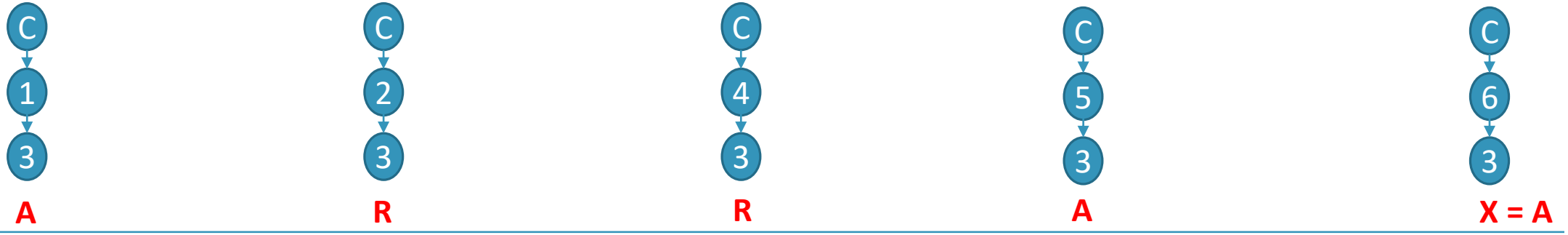


Teniente 3

OM(2)

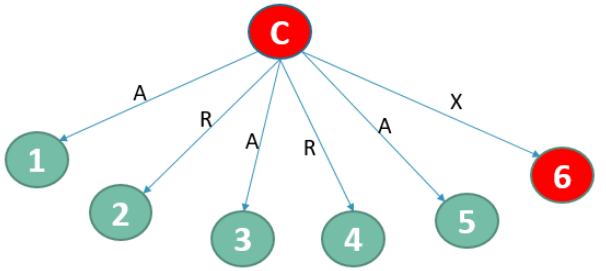


OM(1)



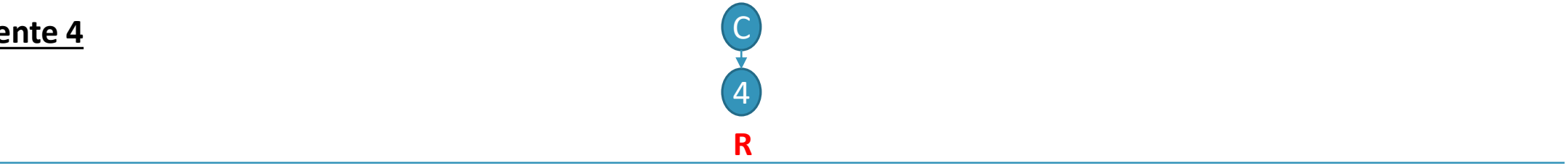
¿Consenso entre los generales leales?							
	OM(2)		OM(1)				Mayoría
T1	A	R	A	R	A	X = A	A
T2	R	A	A	R	A	X = R	Nada
T3	A	A	R	R	A	X = A	A

Ejemplo para n = 7 y m = 2

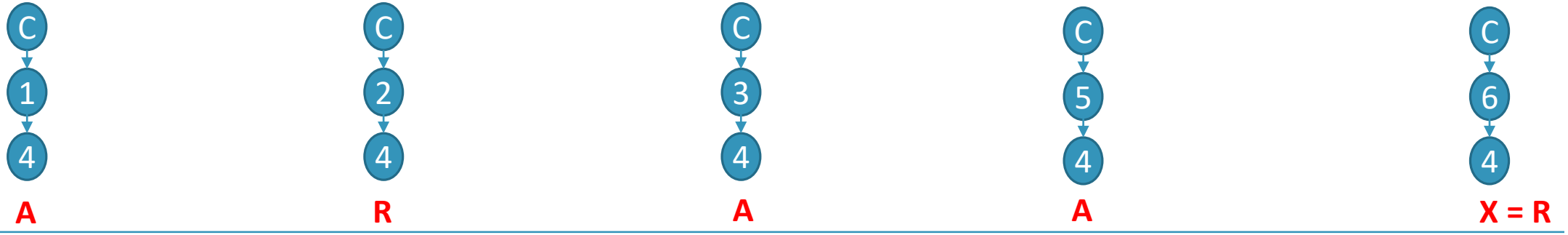


Teniente 4

OM(2)

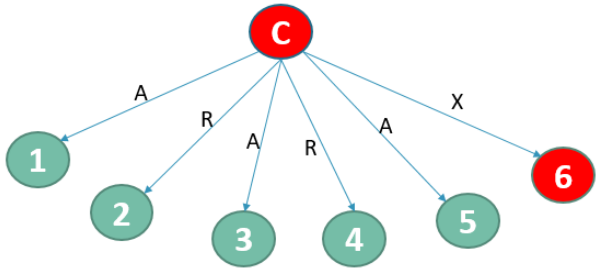


OM(1)



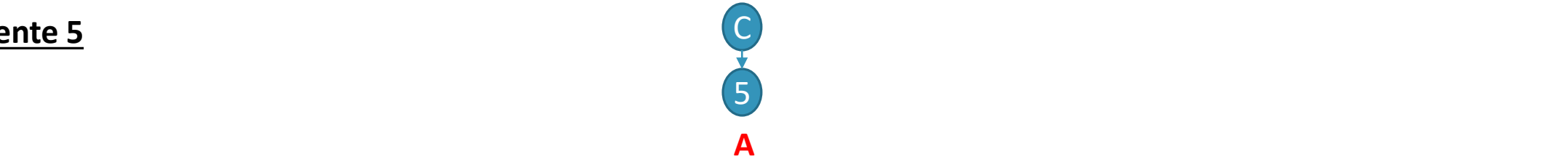
¿Consenso entre los generales leales?							
	OM(2)	OM(1)					Mayoría
T1	A	R	A	R	A	X = A	A
T2	R	A	A	R	A	X = R	Nada
T3	A	A	R	R	A	X = A	A
T4	R	A	R	A	A	X = R	Nada

Ejemplo para n = 7 y m = 2

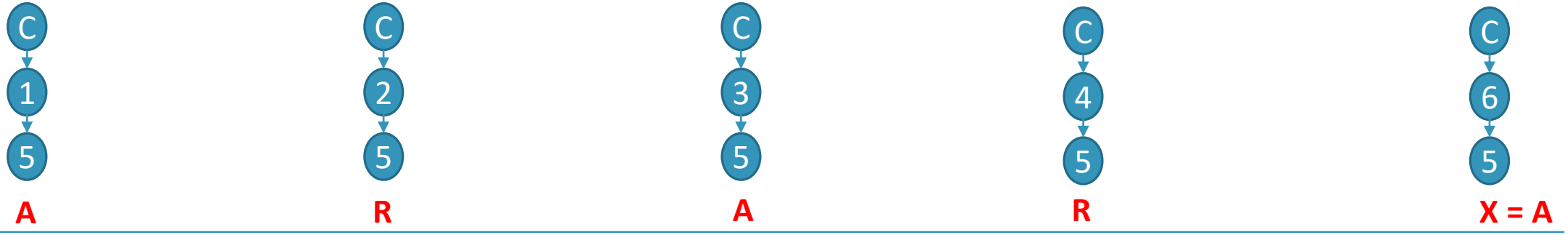


Teniente 5

OM(2)



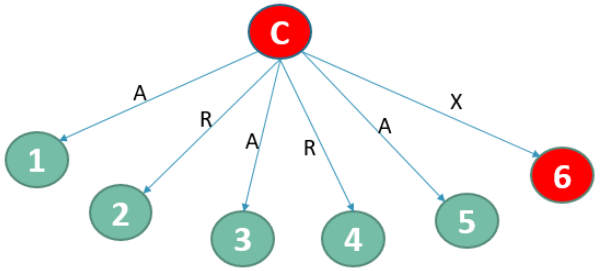
OM(1)



¿Consenso entre los generales leales?							
	OM(2)	OM(1)					Mayoría
T1	A	R	A	R	A	X = A	A
T2	R	A	A	R	A	X = R	Nada
T3	A	A	R	R	A	X = A	A
T4	R	A	R	A	A	X = R	Nada
T5	A	A	R	A	R	X = A	A

Los tenientes leales no han llegado a un consenso

Ejemplo para n = 7 y m = 2



Teniente 6

OM(2)

C → 6 → **X**

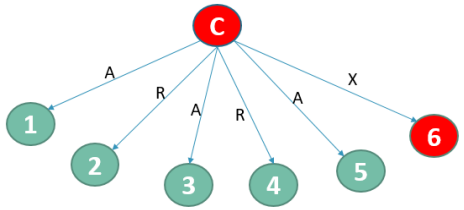
OM(1)

C → 1 → 6	C → 2 → 6	C → 3 → 6	C → 4 → 6	C → 5 → 6
A	**R**	**A**	**R**	**A**

Algoritmo de Lamport, Shostak y Pease

- Definición recursiva, con un caso base de $m = 0$ y recursividad con $m > 0$
- Algoritmo para $OM(0)$
  1. El comandante envía su orden a cada teniente
  2. El teniente utiliza esa orden
- Algoritmo para $OM(m)$, con $m > 0$
  1. El comandante envía su orden a cada teniente
  2. Para cada i, siendo v_i el valor que el teniente i recibe del comandante, el teniente actuará como comandante en $OM(m-1)$ para enviar el valor v_i a cada uno de los otros $n-2$ tenientes
  3. Para cada i y j, siendo $i \neq j$ y v_i el valor que el teniente i recibe del teniente j en el paso 2 (utilizando el algoritmo $OM(m-1)$), el teniente i utiliza ese valor para calcular $Majority(v_1, v_2, \dots v_n)$.

Ejemplo para n = 7 y m = 2

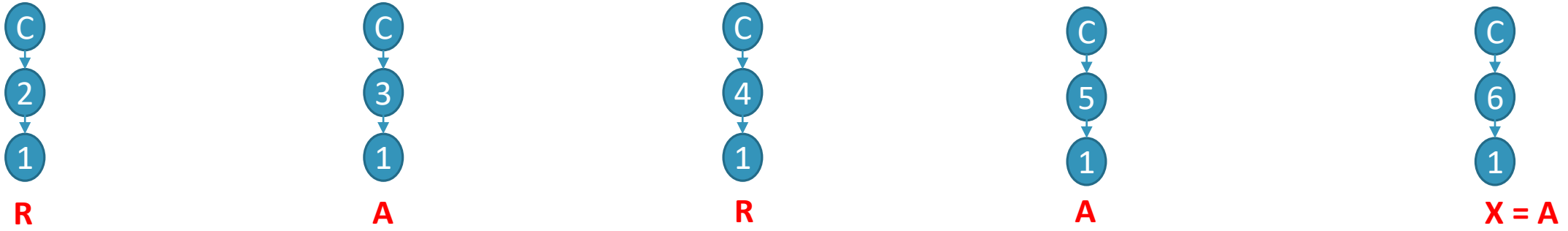


Teniente 1

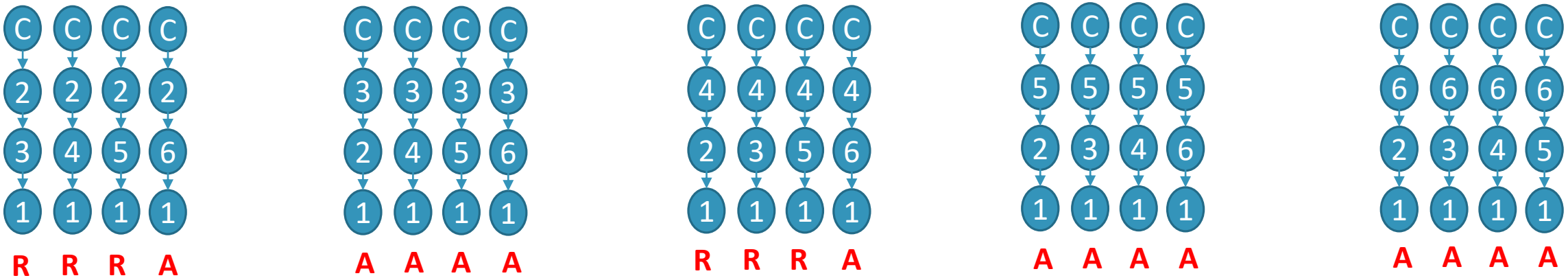
OM(2)

T1	OM(1)	OM(0)					Mayoría
		Mayoría(T2)	*Mayoría(T3)*	*Mayoría(T4)*	*Mayoría(T5)*	*Mayoría(T6)*	
		R	A	R	A	A	
		R	A	R	A	A	
		R	A	R	A	A	
		A	A	A	A	A	
	A	R	A	R	A	A	A

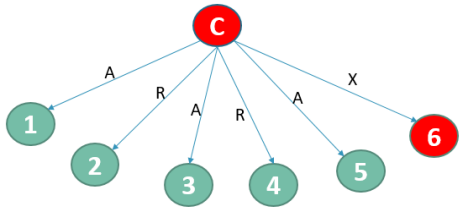
OM(1)



OM(0)



Ejemplo para n = 7 y m = 2

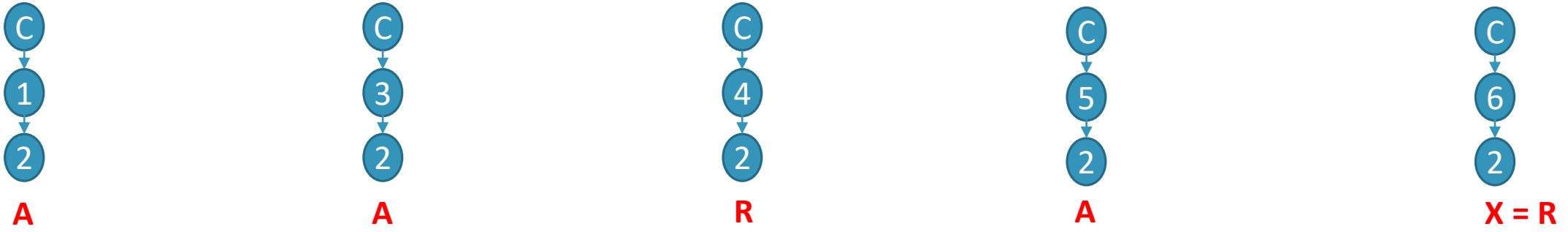


Teniente 2

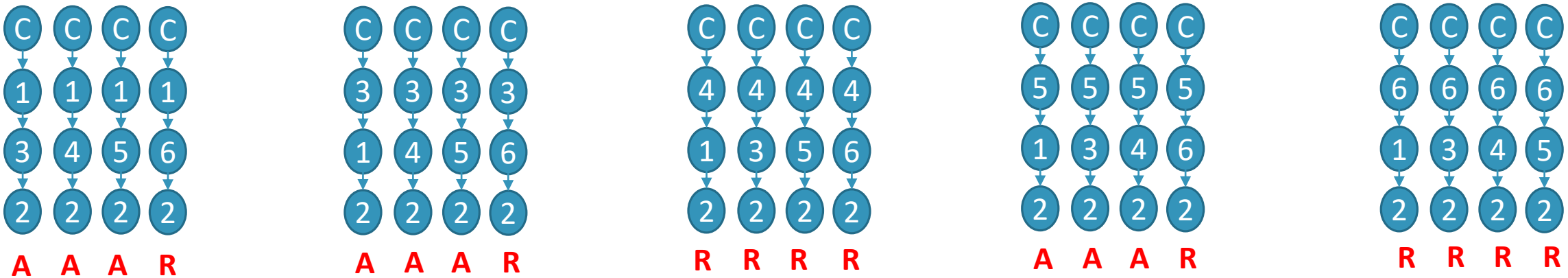
OM(2)

T2	OM(1)	OM(0)					Mayoría
		Mayoría(T1)	*Mayoría(T3)*	*Mayoría(T4)*	*Mayoría(T5)*	*Mayoría(T6)*	
		A	A	R	A	R	
		A	A	R	A	R	
		A	A	R	A	R	
		R	R	R	R	R	
	Nada	A	A	R	A	R	A

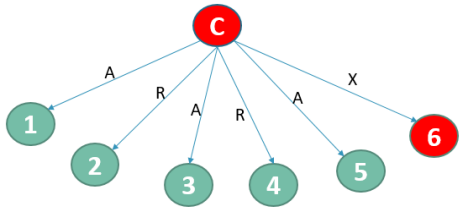
OM(1)



OM(0)



Ejemplo para n = 7 y m = 2

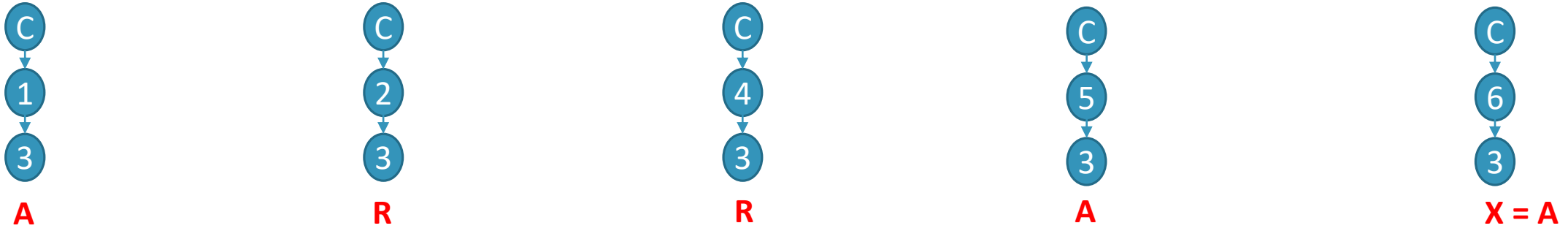


Teniente 3

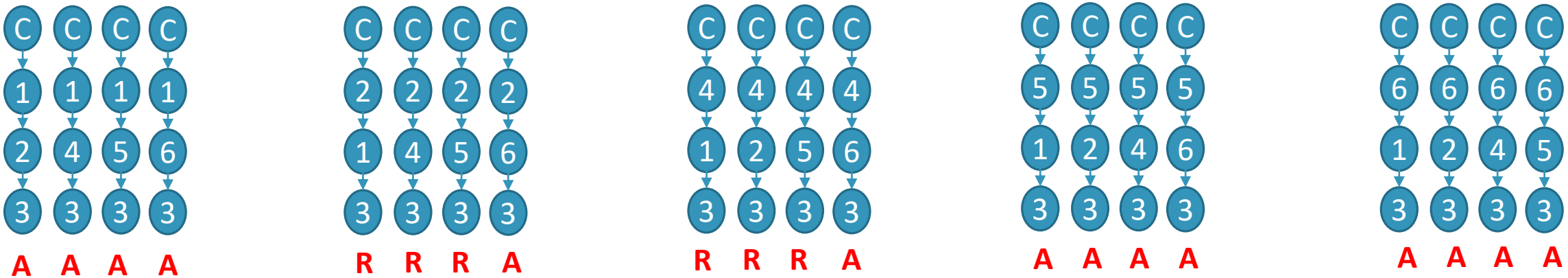
OM(2)

T3	OM(1)	OM(0)					Mayoría
		Mayoría(T1)	*Mayoría(T2)*	*Mayoría(T4)*	*Mayoría(T5)*	*Mayoría(T6)*	
		A	R	R	A	A	
		A	R	R	A	A	
		A	R	R	A	A	
		A	A	A	A	A	
	A	A	R	R	A	A	A

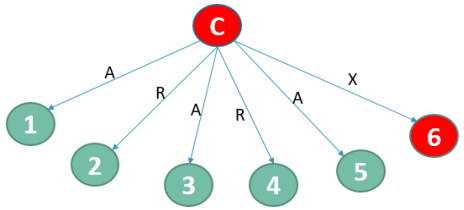
OM(1)



OM(0)



Ejemplo para n = 7 y m = 2



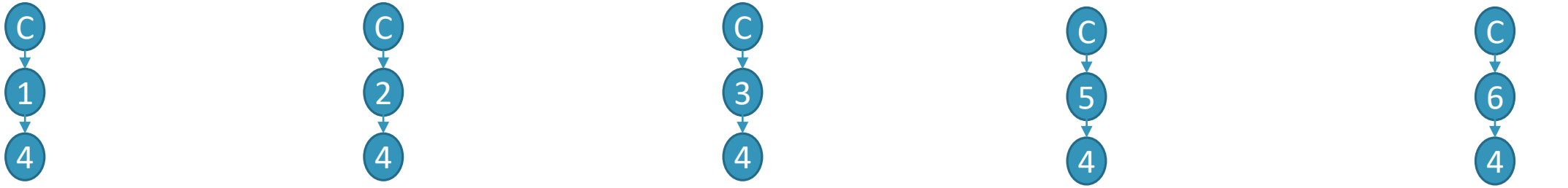
Teniente 4

OM(2)

R

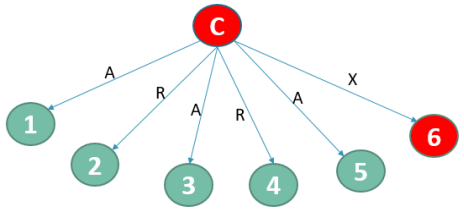
T4	OM(1)	OM(0)					Mayoría
		Mayoría(T1)	*Mayoría(T2)*	*Mayoría(T3)*	*Mayoría(T5)*	*Mayoría(T6)*	
		A	R	A	A	R	
		A	R	A	A	R	
		A	R	A	A	R	
		R	R	R	R	R	
	Nada	A	R	A	A	R	A

OM(1)



OM(0)

Ejemplo para n = 7 y m = 2

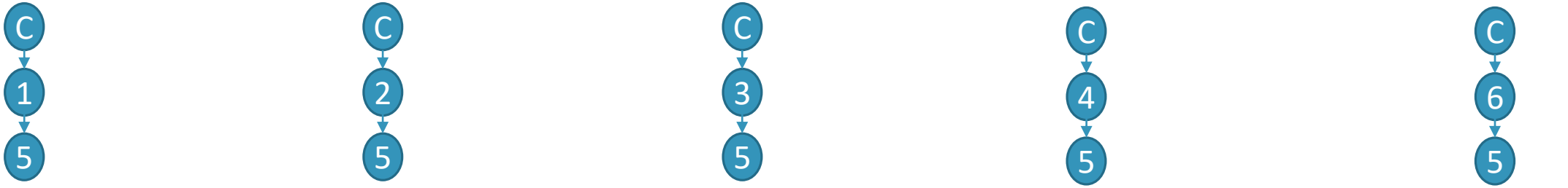


Teniente 5

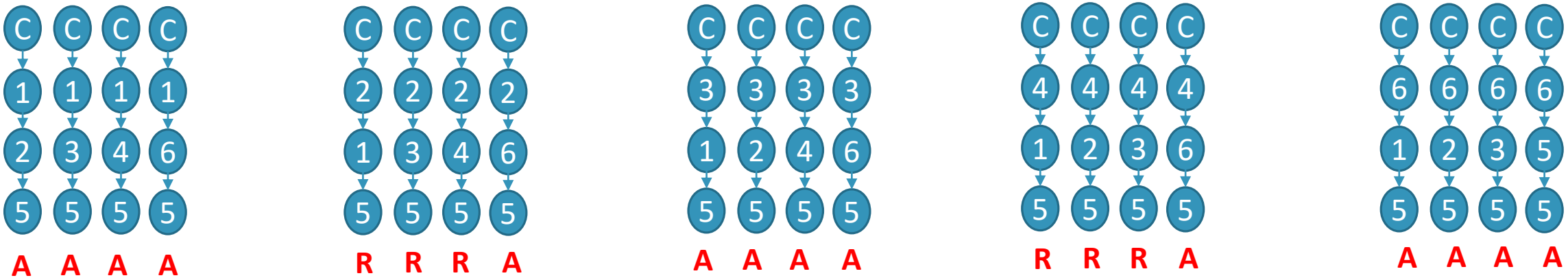
OM(2)

T5	OM(1)	OM(0)					Mayoría
		Mayoría(T1)	*Mayoría(T2)*	*Mayoría(T3)*	*Mayoría(T4)*	*Mayoría(T6)*	
		A	R	A	R	A	
		A	R	A	R	A	
		A	R	A	R	A	
		A	A	A	A	A	
	A	A	R	A	R	A	A

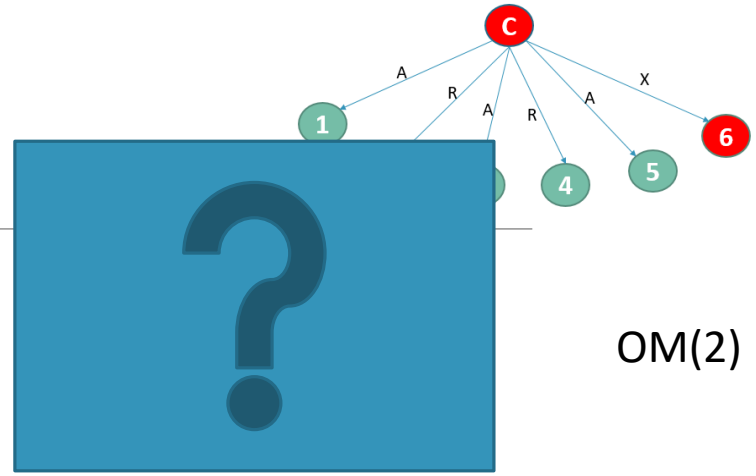
OM(1)



OM(0)



Ejemplo para n = 7 y m = 2



Teniente 6

OM(2)

C → 6 → X

OM(1)

C → 1 → 6 → A

C → 2 → 6 → R

C → 3 → 6 → A

C → 4 → 6 → R

C → 5 → 6 → A

OM(0)

C → 1 → 2 → 6 → A

C → 1 → 3 → 6 → A

C → 1 → 4 → 6 → A

C → 1 → 5 → 6 → A

C → 2 → 1 → 6 → R

C → 2 → 3 → 6 → R

C → 2 → 4 → 6 → R

C → 2 → 5 → 6 → R

C → 3 → 1 → 6 → A

C → 3 → 2 → 6 → A

C → 3 → 4 → 6 → A

C → 3 → 5 → 6 → A

C → 4 → 1 → 6 → R

C → 4 → 2 → 6 → R

C → 4 → 3 → 6 → R

C → 4 → 5 → 6 → R

C → 5 → 1 → 6 → A

C → 5 → 2 → 6 → A

C → 5 → 3 → 6 → A

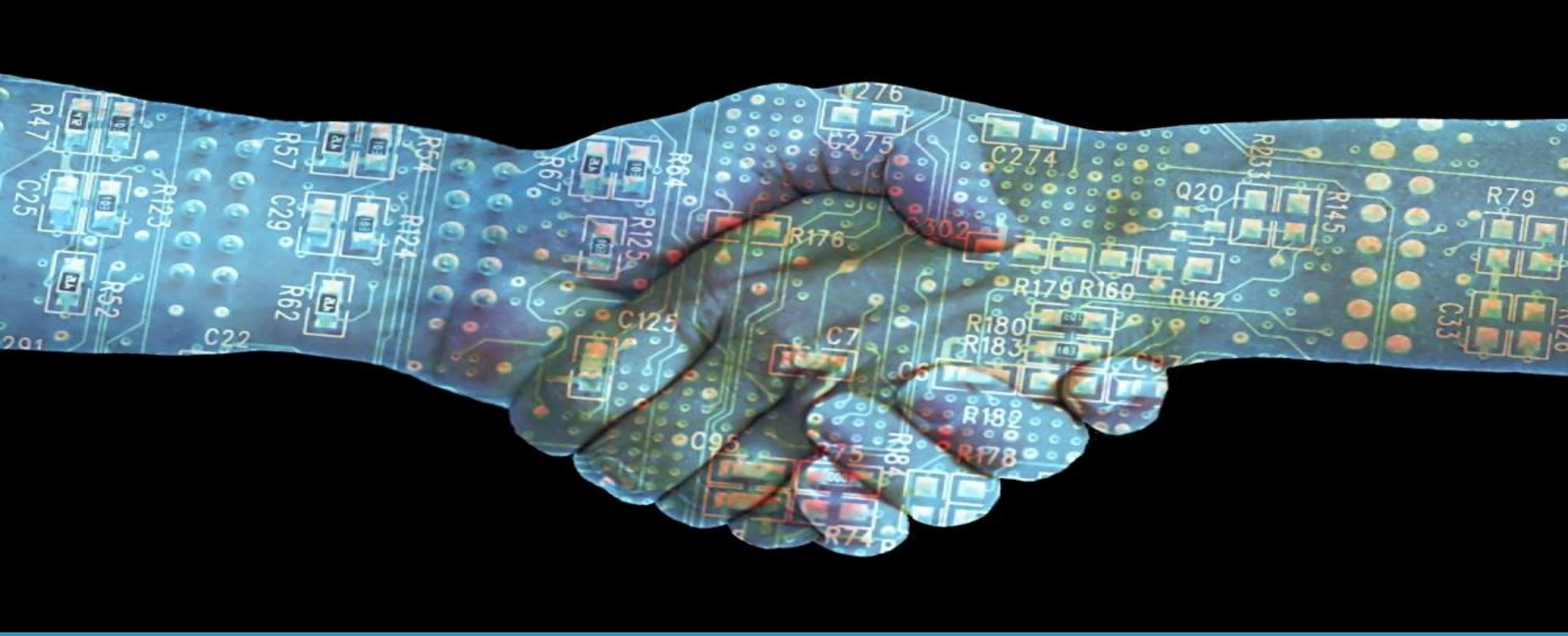
C → 5 → 4 → 6 → A

Complejidad

- Primer paso OM(m) => n – 1 mensajes
- Segundo paso OM(m-1) => (n – 1) x (n – 2) mensajes
- Tercer paso OM(m-2) => (n – 1) x (n – 2) x (n – 3) mensajes
- …
- Cálculo de la complejidad $= \prod_{i=1}^{i=m+1}(n - i) = \boldsymbol{O\left(n^{m+1}\right)}$

m	Mensajes enviados
0	$O(n)$
1	$O(n^2)$
2	$O(n^3)$
3	$O(n^4)$
…	
99	$O(n^{100})$

…hay otras versiones del problema, ¿y si **no** todos los nodos se pueden comunicar con todos directamente?


R47
C25
R57
C29
R62
C22
R64
R67
R125
C125
R176
C275
C274
C302
Q20
R233
R145
R79
R179
R160
R162
C33
R180
R183
C7
R182
R178
R74

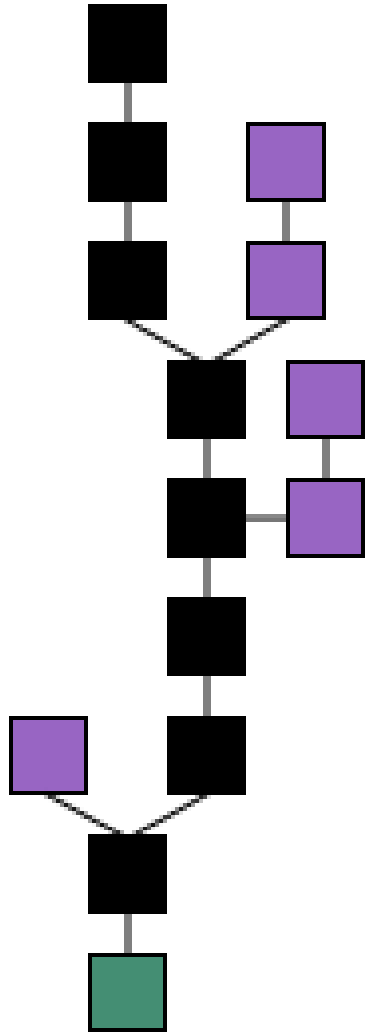
Blockchain

¿Blockchain?

- Es un lista de registros altamente tolerable a fallos “bizantinos”

- Bloques
  - Hash
  - Puntero hash
  - Timestamp
  - Datos

- Es una base de datos **abierta y distribuida** que puede almacenar transacciones entre partes de forma eficiente y verificable a través del tiempo



Aplicaciones de las cadenas de bloques

- Prácticamente cualquiera en la que se registren eventos…
- Registros médicos
- Actividades de gestión
- Procesamiento de transacciones
- Trazabilidad alimentaria
- Sistemas de votación
- …

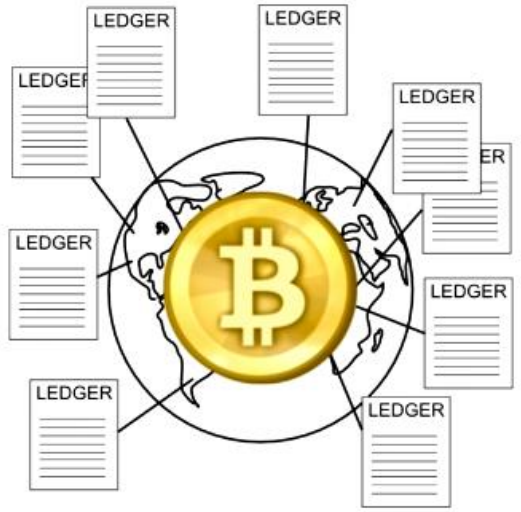
Bitcoin

- Bitcoin es una criptomoneda basada totalmente en un blockchain

Pilares fundamentales

- Los dos pilares fundamentales son la **criptografía** y el "**libro de transacciones o de contabilidad**"

- Cuando hacemos un pago, la información se transmite al resto de nodos de la red

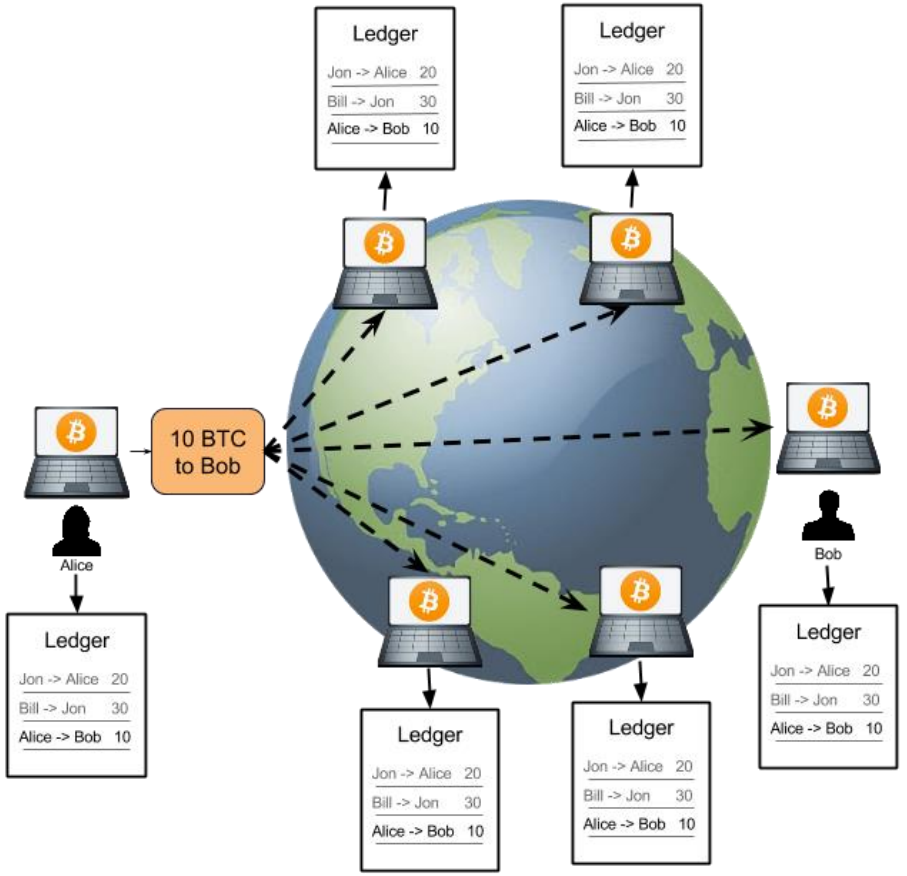


Contabilidad

De	A	Cant.
Ana	José	5
Juan	Maria	15
…		

Mantenimiento colectivo del libro de contabilidad

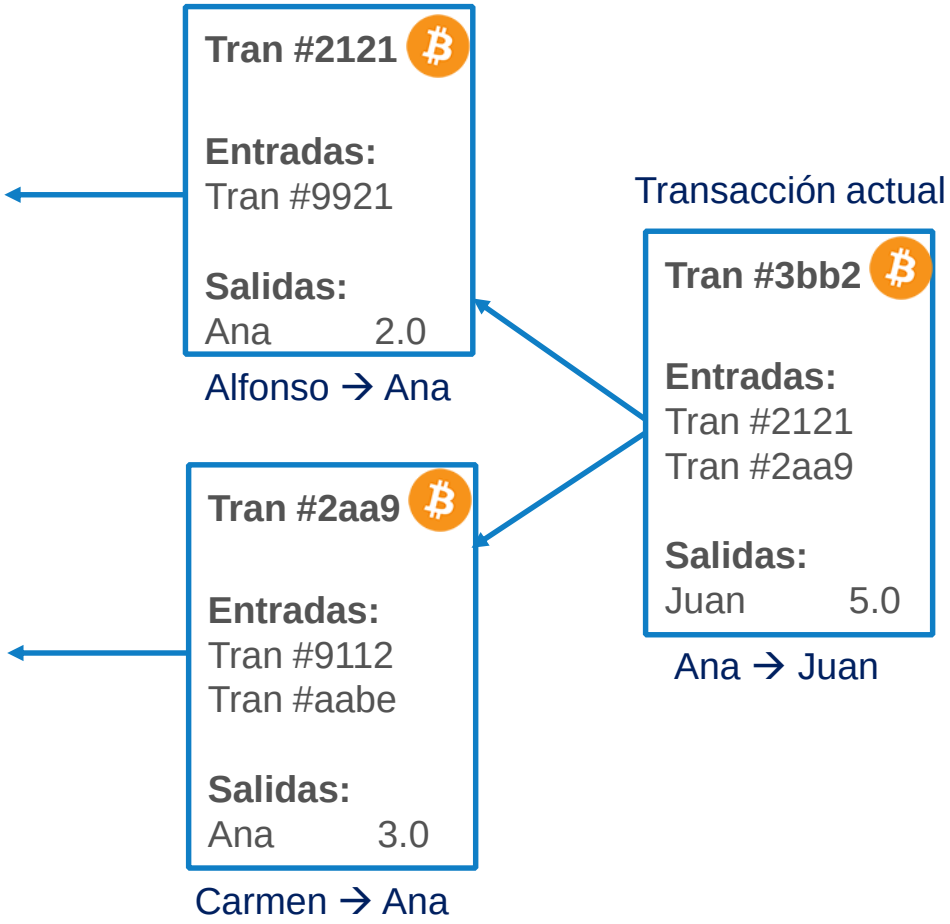
- El libro de contabilidad está mantenido por un grupo de equipos en lugar de por una única entidad
- Las transacciones son públicas



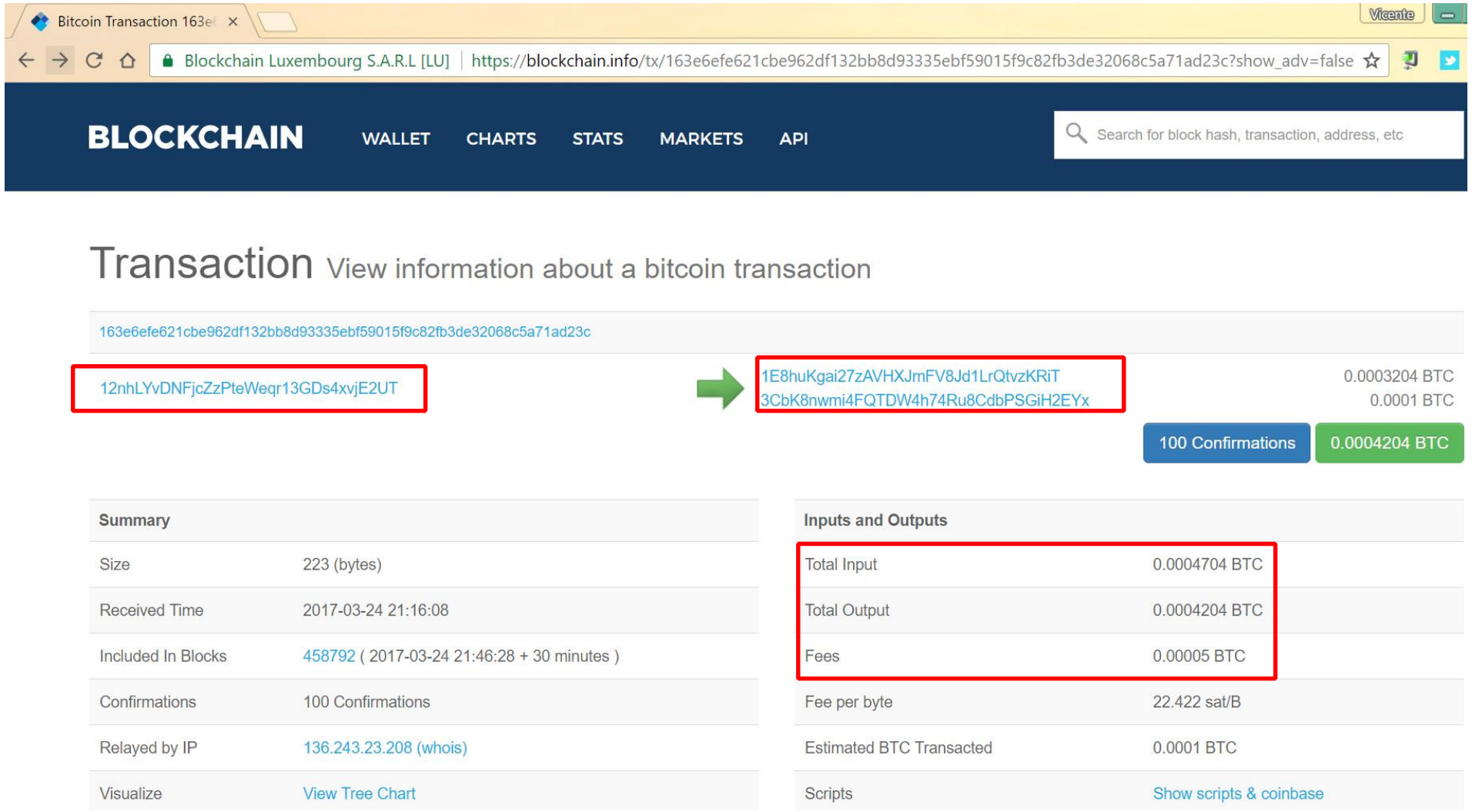
Balance de cuentas

- No existe un registro con el balance de cuentas de los usuarios

- Es decir, en lugar de un **libro de contabilidad**, técnicamente lo que tiene Bitcoin es una **lista enorme de transacciones**



Ejemplo de información de la cadena de bloques



Código. Bloque

```python
import hashlib, datetime

class Block:
    def __init__(self, index, timestamp, data, previous_hash):
        self.index = index
        self.timestamp = timestamp
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.hash_block()

    def hash_block(self):
        sha = hashlib.sha256()
        data = str(self.index) + str(self.timestamp) + str(self.data) + str(self.previous_hash)
        sha.update(data.encode("utf-8"))
        return sha.hexdigest()
```

block.py

Código. Bloque “Génesis”

```
class Block:
    …

    @staticmethod
    def create_genesis_block():
        return Block(0, datetime.datetime.now(), "Genesis Block", "-1")

    @staticmethod
    def next_block(last_block):
        next_index = last_block.index + 1
        next_timestamp = datetime.datetime.now()
        next_data = "Datos del bloque " + str(next_index)
        next_previous_hash = last_block.hash
        return Block(next_index, next_timestamp, next_data, next_previous_hash)
```

block.py

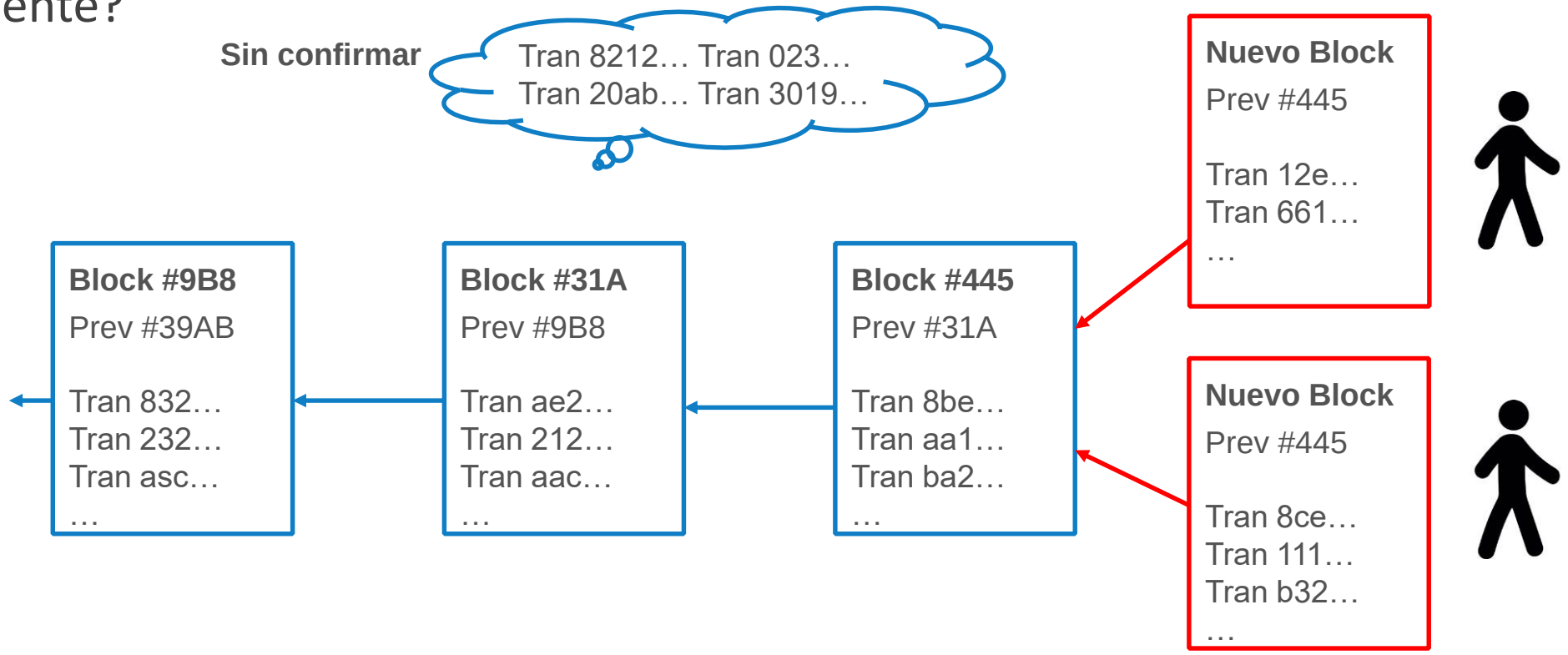
Creación de 15 bloques

```
from block import *

blockchain = [Block.create_genesis_block()]
previous_block = blockchain[0]

num_blocks = 15

for i in range(0, num_blocks):
  block_to_add = Block.next_block(previous_block)
  blockchain.append(block_to_add)
  previous_block = block_to_add

  print("El bloque #{} ha sido añadido a la cadena".format(block_to_add.index))
  print("\tHash anterior: {}".format(block_to_add.previous_hash))
  print("\tHash: {}\n".format(block_to_add.hash))
```

blockchain.py

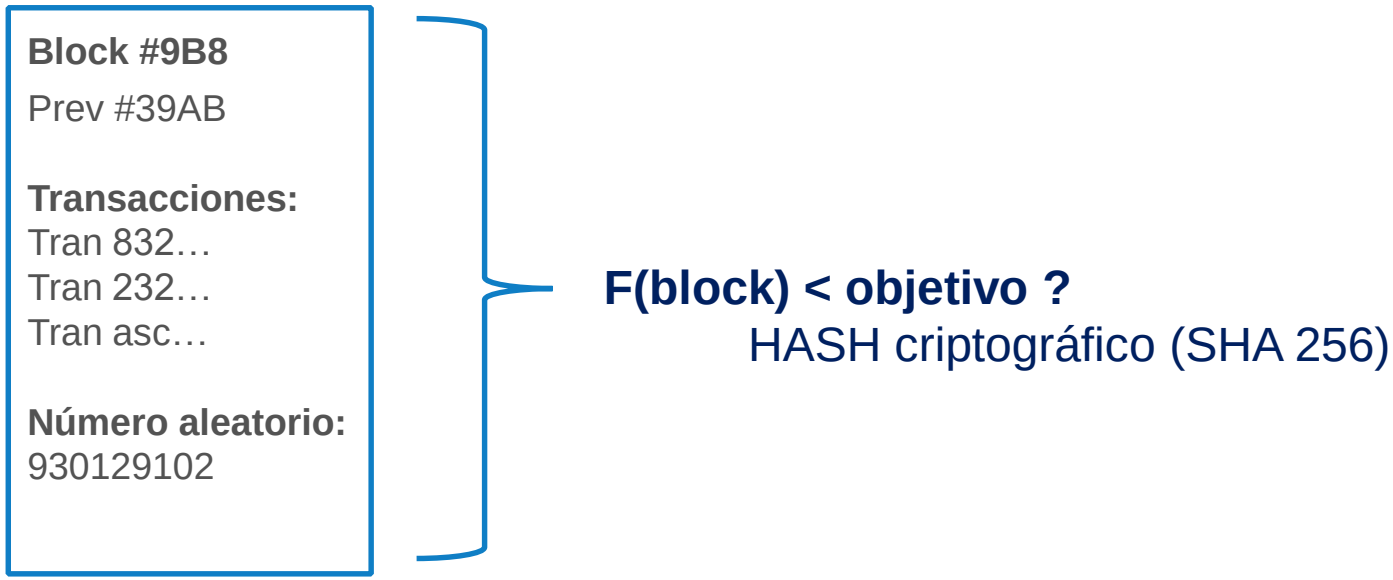
Los bloques, las transacciones y los mineros

- Tendremos transacciones **confirmadas** y **no confirmadas**
- Pero podrían crear bloques al mismo tiempo, ¿cómo se elige cual debería ser el siguiente?



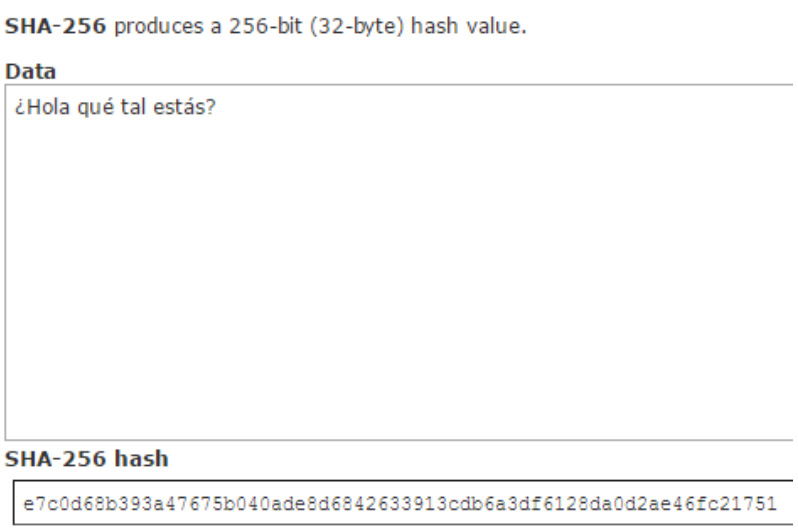
Algoritmo Proof of Work

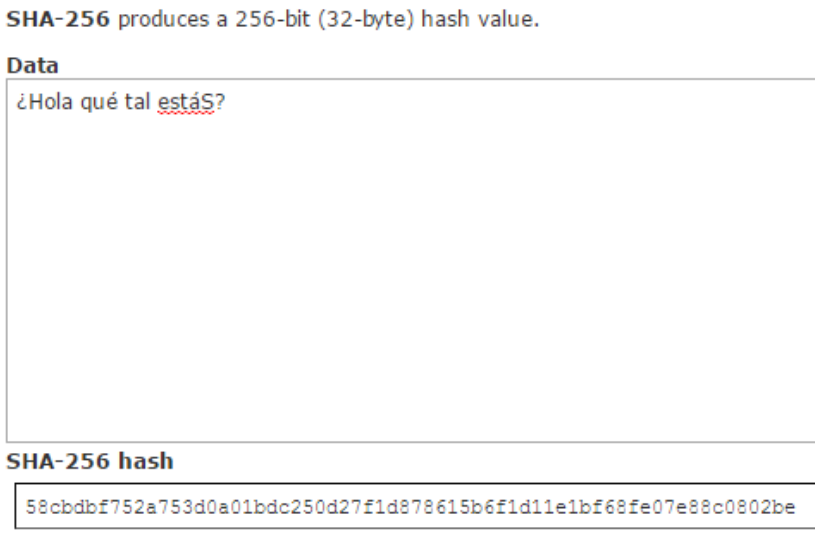
- Cada bloque debe contener la respuesta a un problema matemático concreto
- Parecido a adivinar la combinación de una caja fuerte
- Quienes intentan resolver los problemas son conocidos como mineros



Importancia del algoritmo SHA256

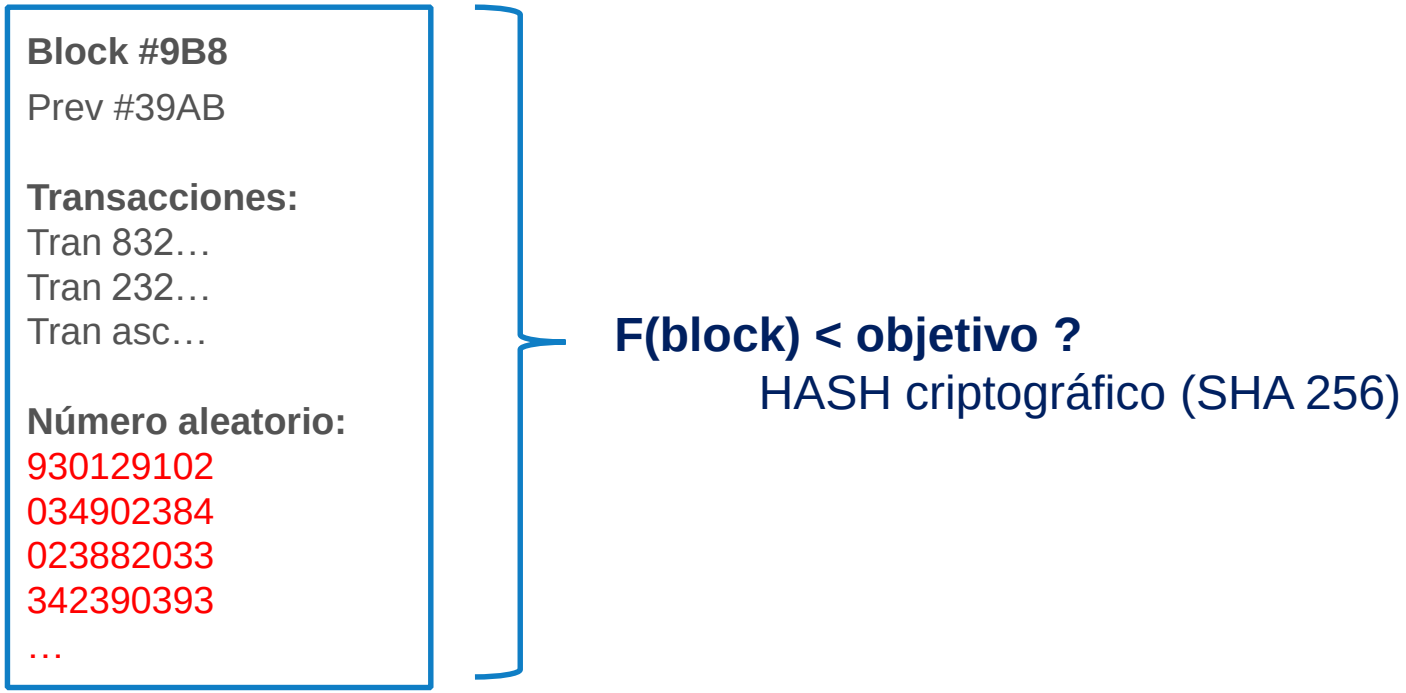
- Es un algoritmo HASH criptográfico con 64 dígitos hexadecimales
- Existe una gran diferencia en las salidas para entradas muy parecidas
  - La salida es totalmente impredecible
  - Sólo podremos hacer suposiciones aleatorias
- http://www.xorbin.com/tools/sha256-hash-calculator





Dificultad para solucionar el problema

- 10 minutos de media en encontrar una solución
  - Quien primero lo resuelva, gana
  - Es muy poco probable que dos personas lo resuelvan al mismo tiempo



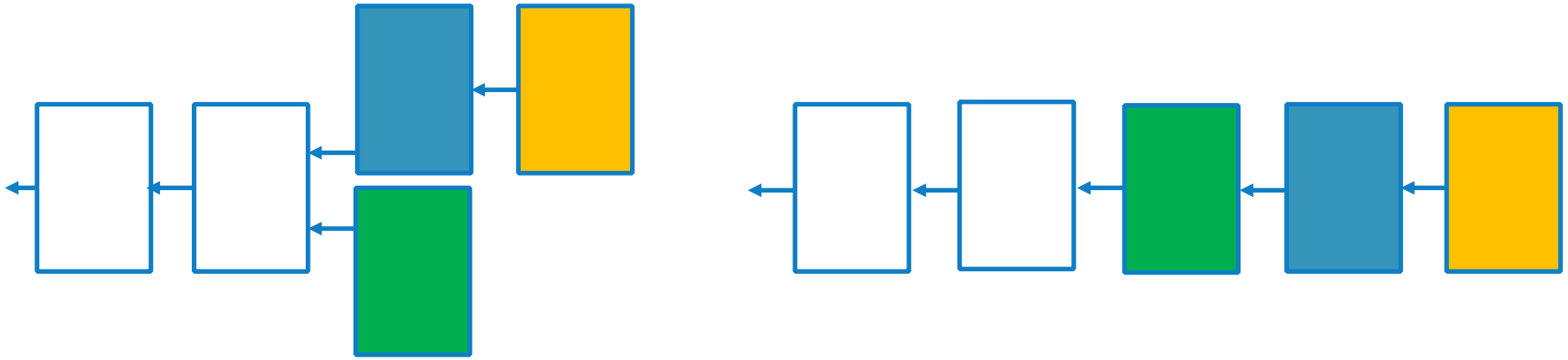
Código. PoW

```
import hashlib, time

string = "Juan envía 10€ a Maria "
complete = False
n = 0

while complete == False:
    current_string = string + str(n)
    hash = hashlib.sha256(current_string.encode("utf-8"))
    hash_value = hash.hexdigest()
    n = n + 1

    print("Cadena: {} -- Hash: {}". format(current_string, hash_value))

    if hash_value.startswith('000000'):
        complete = True

    time.sleep(1)
```

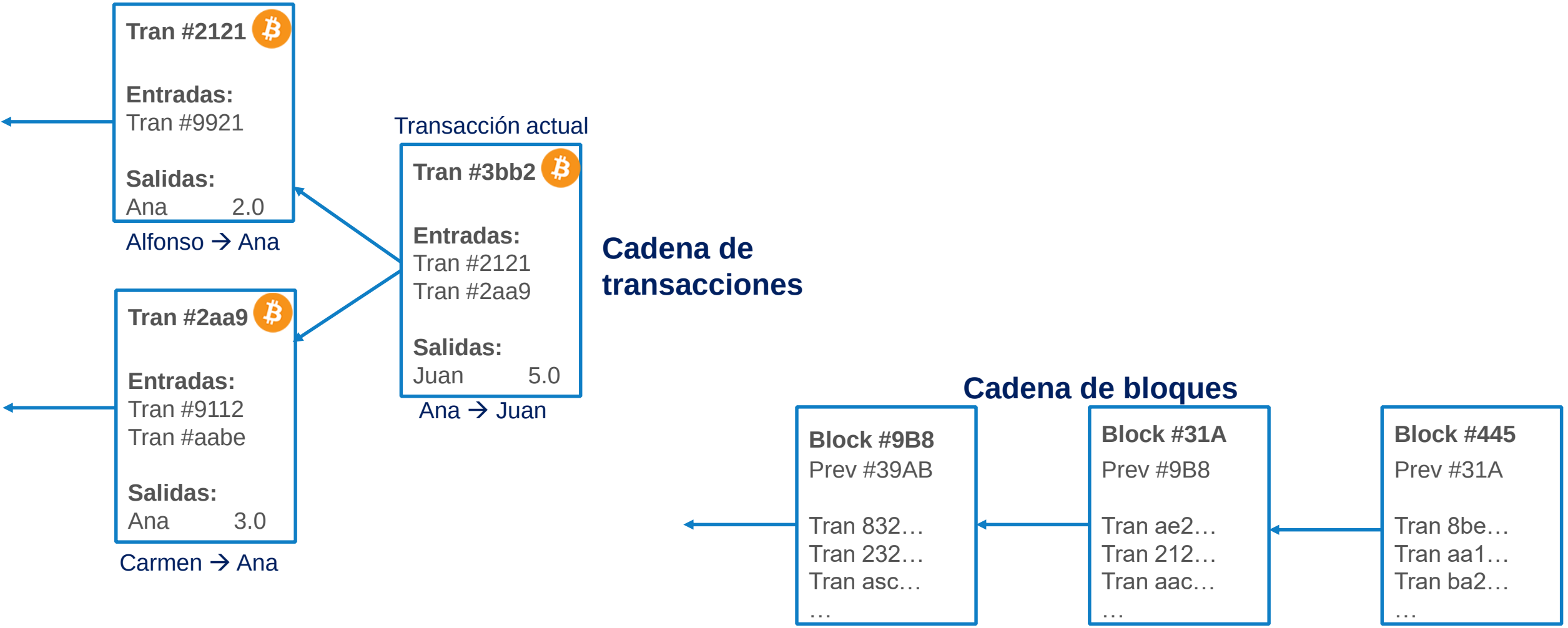
proof_of_work.py

¿Y si se resuelve el problema al mismo tiempo?

- Es altamente improbable, sin embargo teóricamente puede ocurrir
- En ese caso se continua como si nada
  - Tendremos dos ramas (o más) en los bloques de transacciones
- El “problema” se termina cuando se resuelve el siguiente bloque
- La regla es que siempre hay que cambiar lo más rápido posible a la rama más larga disponible

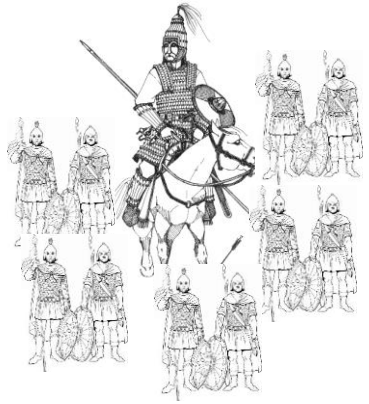


Cadena de bloques VS cadena de transacciones



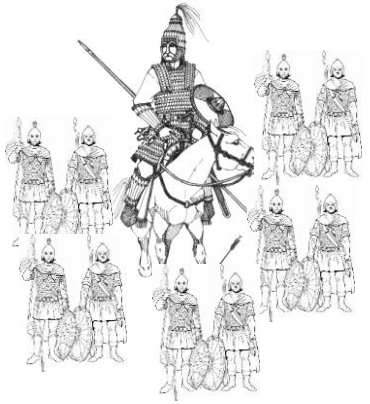
Prueba de trabajo

- El mensajero es interceptado



Mensaje: Atacar el lunes

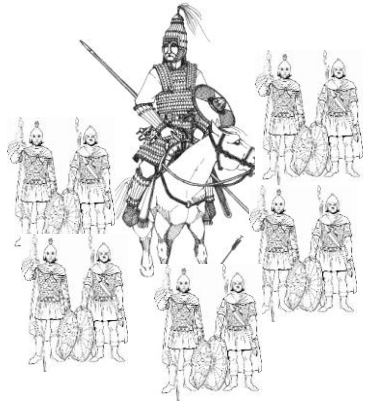
Mensaje: Atacar el martes



Mensaje: Atacar el martes

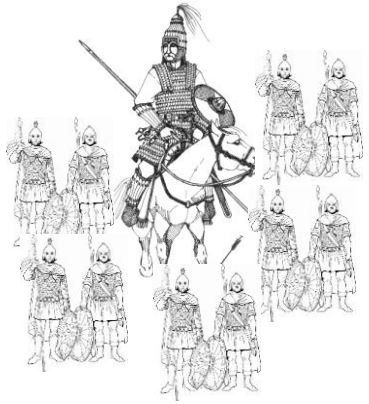
Prueba de trabajo (II)

- El mensajero es interceptado con Proof of Work



Mensaje: Atacar el lunes+ FA291
Hash: 000000A3298D…

Mensaje: Atacar el martes + 112B291
Hash: 00000056CC1A…



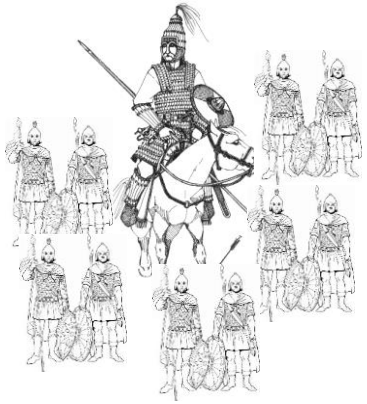
Mensaje: Atacar el martes + 112B291
Hash: 00000056CC1A…

Prueba de trabajo (III)

- El mensajero es interceptado con Proof of Work colaborativo

Mensaje 1: Atacar el lunes
Mensaje 2: Llevad agua
Mensaje 3: Llevad banderas
Valor aleatorio: F238291
Hash: 000000A3298D…

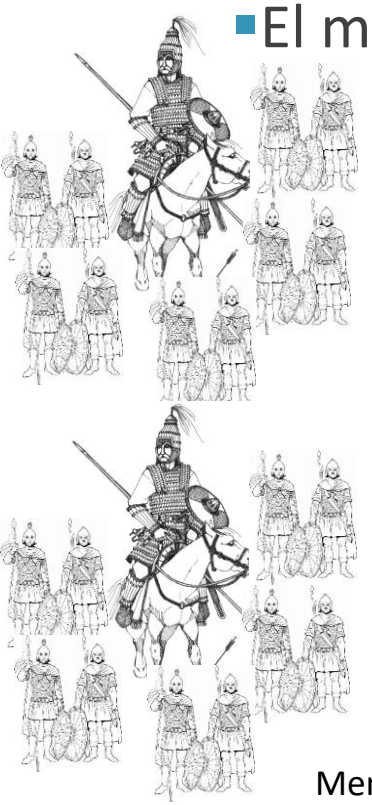
Mensaje 1: Atacar el jueves
Mensaje 2: Atacar el martes
Mensaje 3: Retirada
Valor aleatorio: 229382A
Hash: 000000AAACC234…

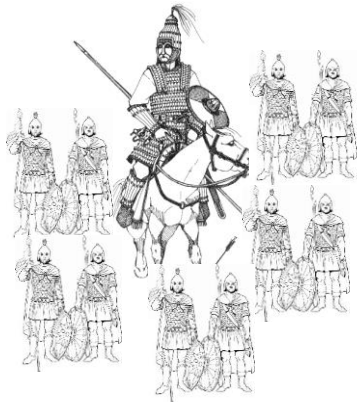


Mensaje 1: Atacar el jueves
Mensaje 2: Atacar el martes
Mensaje 3: Retirada
Valor aleatorio: 229382A
Hash: 000000AAACC234…

Prueba de trabajo (IV)

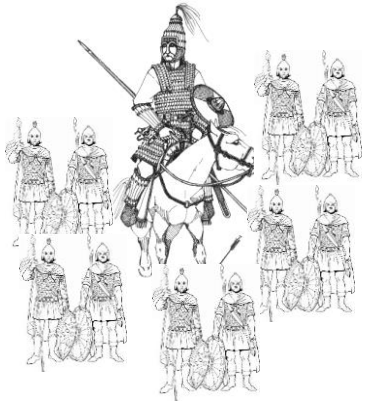
- El mensajero es interceptado con Proof of Work colaborativo y múltiples mensajeros





Mensaje 1: Atacar el lunes
Mensaje 2: Llevad agua
Mensaje 3: Llevad banderas
Valor aleatorio: F238291
Hash: 000000A3298D…

Mensaje 1: Atacar el jueves
Mensaje 2: Atacar el martes
Mensaje 3: Retirada
Valor aleatorio: 229382A
Hash: 000000AAACC234…



Mensaje 1: Atacar el jueves
Mensaje 2: Atacar el martes
Mensaje 3: Retirada
Valor aleatorio: 229382A
Hash: 000000AAACC234…