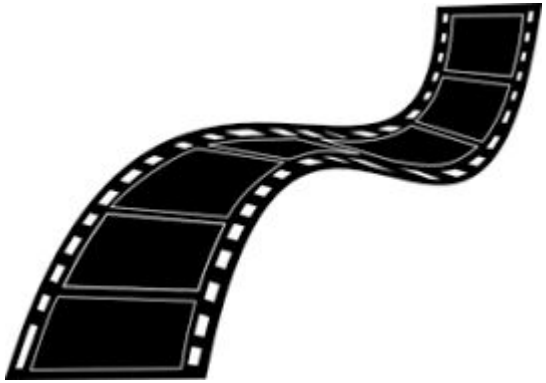
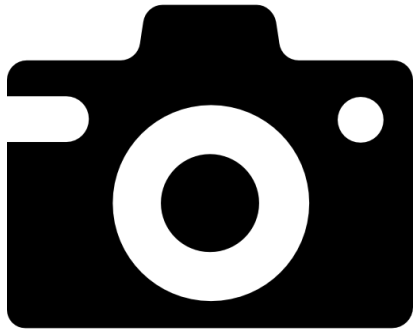


Distributed snapshots

Mariana Hernández Rocha
Salvador García González
Fabián Orduña Ferreira



Distributed Snapshots: Determining global States of Distributed Systems*

K. Mani Chandy; Leslie Lamport

*Chandy, K. Mani, and Leslie Lamport. "Distributed snapshots: Determining global states of distributed systems." *ACM Transactions on Computer Systems (TOCS)* 3.1 (1985): 63-75.

Agenda

1

Introducción

Contexto
Motivación

2

Condiciones

Problemas
Supuestos
Estado global

3

Algoritmo

Modelo del sistema
Supuestos
Algoritmo y terminación
Propiedades del estado global guardado

1. Introducción

Distributed Snapshots

Objetivo

Presentar un algoritmo con el cual los procesos en un sistema distribuido **determinan el estado global del sistema**

Autores

Leslie Lamport

K.M. Chandy

Conceptos Clave

- Proceso
- Canal
- Sistema distribuido
- Evento

Intuición del algoritmo

IDEA

- Para determinar el estado global del sistema:
  - Cada proceso **p** enlista la cooperación con otros procesos, los cuales deben registrar sus propios estados locales y enviarlos a **p.**
  - No todos los procesos pueden registrar su estado al mismo tiempo ya que no hay reloj en común.
  - Se crea un algoritmo en el cual cada proceso registra su estado y los estados de los canales de comunicación con el objetivo de formar un estado global del sistema.

EJEMPLO

- Un grupo de fotógrafos, donde cada uno de ellos observan una foto panorámica y dinámica
- La panorámica no puede ser capturada por un solo fotógrafo, si no que toman distintos snapshots y los juntan para formar la escena completa

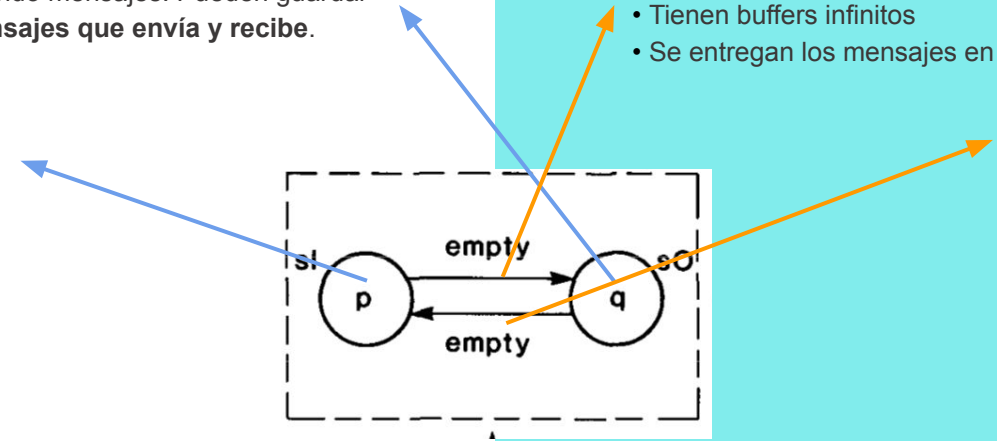
Definiciones importantes:

- **Proceso (p):**

Definido por un conjunto de estados, cada proceso se comunica enviando y recibiendo mensajes. Pueden guardar **su propio estado** y **los mensajes que envía y recibe**.

- **Canal (c):**

Sirve de **comunicación entre procesos**. Se asume que:
- Son libres de errores
- Tienen buffers infinitos
- Se entregan los mensajes en el orden de envío



- **S = Conjunto de posibles estados globales de un sistema distribuido**

Definiciones importantes:

- **Estado global de D**

Set de **estados** de **procesos** y **canales**.

- Estado inicial proceso: Cada proceso en su estado inicial (elemento del conjunto **S**)
- Estado inicial canal: La secuencia vacia.

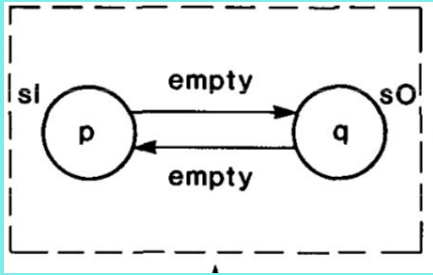
- **Evento (e):**

En un proceso **p** es una **acción atómica** que puede cambiar el estado de **p** y el estado de a lo más un canal. Formalmente lo podemos definir con una quinteta de **<p, s, s', M, c>**

- **Sistema Distribuido (D):**

Descrito como una **gráfica dirigida con vértices: procesos** y **aristas: canales**, tal que:

• Existe un conjunto finito de procesos
• Existe un conjunto finito de canales



Ejemplo de conservación de un solo token

- Dos posibles estados: **S** = {S1, S0}
  - S0 Representa que el proceso **no contiene** el token
  - S1 Representa que el proceso **contiene** el token
- Dos procesos: **p** y **q**
- Dos canales **c** y **c'**
- El token pasa de un proceso a otro proceso
- Cada proceso tiene dos eventos:
  - 1) Transiciona de S1 a S0
  - 2) Transiciona de S0 a S1

In-p In-c

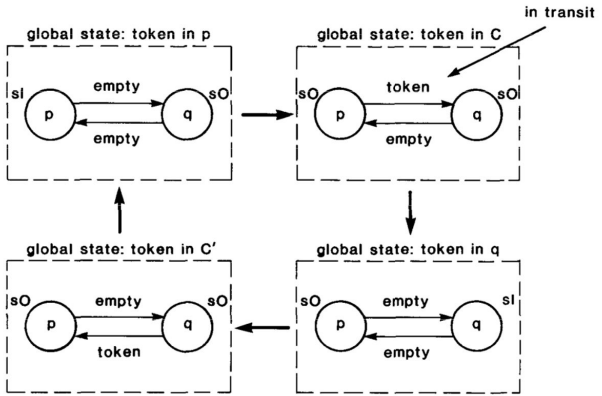


Fig. 4. Global states and transitions of the single-token conservation system.

In-c' In-q

2. Condiciones

Definiciones requeridas para algoritmo

Sea n = # mensajes **enviados a c** antes que **p** registre su estado

Sea n' = # mensajes **enviados a c** antes que **c** registre su estado

m = número de mensajes **recibidos a través de c** antes que **q** registre su estado

m' = número de mensajes **recibidos a través de c** antes que **c** registre su estado

Algoritmo de registro de estado global (Condición 1)

p envía token (mensaje) a través de **c**

p registra su estado
p tiene el token

c, q, c' registran sus estados:
c tiene el token
c' y **q** no tienen token

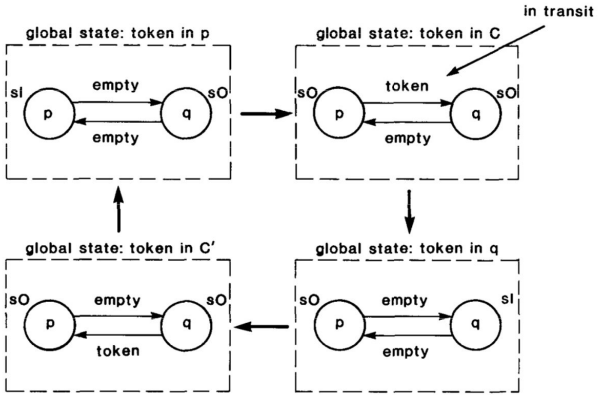


Fig. 4. Global states and transitions of the single-token conservation system.

- En este ejemplo se observa la inconsistencia al **mostrar dos tokens en el sistema**, uno en **p** y otro en **c**
- **¡No es factible!**

Inconsistente si n < n'
n=0, n'=1

Sea n = # mensajes **enviados a c** antes **p** registre su estado

Sea n' = # mensajes **enviados a c** antes que **c** registre su estado

Algoritmo de registro de estado global (Condición 1)

- En este ejemplo se observa la inconsistencia al no mostrar tokens en el sistema
- **¡No es factible!**

Inconsistente si n > n'

n = 1, n' = 0

Entonces n = n'

p envía token a través de **c**

c registra su estado
c no tiene token

c', p, q registran sus estados
c', p, q no tienen el token

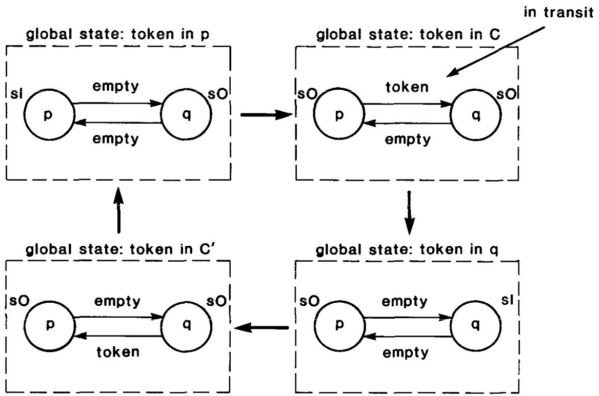


Fig. 4. Global states and transitions of the single-token conservation system.

Sea n = # mensajes **enviados a c** antes **p** registre su estado

Sea n' = # mensajes **enviados a c** antes que **c** registre su estado

Algoritmo de registro de estado global (Condición 2)

Se envía token

- Casos equivalentes a n < n’ y n > n
- ¡No es factible!

Entonces m = m’

Se registra estado de c

Se registra estado de c’, p, q

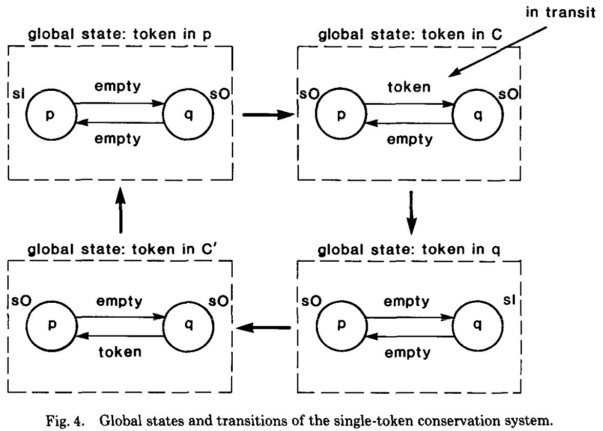


Fig. 4. Global states and transitions of the single-token conservation system.

m = número de mensajes **recibidos a través de c** antes que **q** registre su estado

m’ = número de mensajes **recibidos a través de c** antes que **c** registre su estado

Algoritmo de registro de estado global (Condición 3 y 4)

En cada estado:

- Número de mensajes **recibidos** en un canal **no** puede exceder el número de mensajes **enviados** a través del canal

Entonces..

El **estado del canal c** registrado es:

- La secuencia de mensajes **enviados** al canal antes que **p** registre su estado
- Menos la secuencia **recibida** a través del canal, antes que el estado de **q** registre su estado

Si n' = m' entonces el estado del canal es el vacío
Si n'>= m', entonces el estado son los mensajes m'+1, …, n

Por consecuencia
n>= m
n'>=m'

Algoritmo de registro de estado global (Intuición)

Enlistando las propiedades 1 a 4:

1) n = n'
2) m = m'
3) n' >= m'
4) n >= m

Utilizando las propiedades, se puede escribir una primera versión del algoritmo, por medio del cual **q** puede guardar el estado del canal **c**:

1) **p** envía un mensaje especial llamado **marcador**, después de que el n-ésimo mensaje que se envía a través de **c** (y antes de enviar más mensajes)

2) El estado de **c** es como se definió en la slide pasada: la diferencia entre mensajes enviados antes que **p** sea registrado y los mensajes recibidos a través de **c** antes que **q** registre su estado

3) **q** registra su estado después de recibir el **marcador** y antes de que reciba más mensajes.

3. Algoritmo

Algoritmo

Regla para enviar marcador (desde proceso p):

- Para cada canal **c** desde y hacia **p**:
  - **p** envía un **marcador** a **c** después de que **p** registre su estado y antes que **p** envíe más mensajes

Regla para recibir marker (al proceso q):

- Cuando **q** reciba el marcador:
  - Si **q** no ha registrado su estado
    - **q** registra su estado
    - **q** registra **c** como vacio
  - Otro caso:
    - **q** registra el estado de **c** como mensajes recibidos en **c** después que el estado de **q** fue registrado y antes que **q** reciba el marcador

Algoritmo (inicialización)

Puede ser iniciado por uno o más procesos, cada uno registra su estado **espontáneamente, sin recibir marcador**.

Si un proceso **p** registra su estado y **hay un canal de p a q**, entonces **q** registrará su estado en tiempo finito, por que **p** enviará un marcador a través del canal y **q** recibirá el marcador en tiempo finito (por inducción).

Acabar en tiempo finito se asegura para cada proceso **q** si:

- **q** espontáneamente registra su estado
- Existe un **camino desde el proceso p.**

Si es un grafo fuertemente conexo y existe al menos un proceso espontáneo que registra su estado, entonces todos los procesos registran sus estados en tiempo finito.

4. Detección de propiedad estable

Definiciones:

- **Detección de propiedad estable**

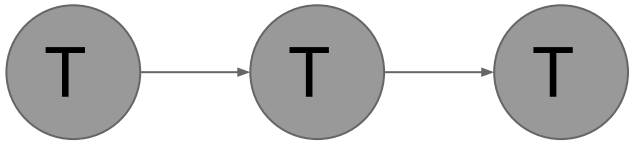
Una propiedad estable es aquella que persiste. Una vez que es verdadera, siempre permanece verdadera. Por ejemplo:
- El cómputo terminó
- El sistema está en deadlock

- **Función next(S,e)**

Es el estado global inmediatamente después de la ocurrencia del evento **e** en el estado global **S**. Tal que:
- El estado del proceso **p** es **s'**
- Si hay un canal apuntando a **p**, entonces el estado de **c** en **next(S,e)** es con **M** borrado de su head.
- Si un canal parte de **p**, entonces el estado de **c** en **next(S,e)** es con **M** añadida a la cola.

Detección de propiedad estable

- **y(S)** toma valores (True,False). Recibe el nombre de función predicado con dominio **S** en **D**
- **y(S)** es una propiedad estable de **D** si **y(S')** implica **y(S)** para todo **S'**
- Idea Básica: Se determina un estado global **S** del sistema y se computa **y(S)** para ver si la propiedad estable se mantiene



Detección de propiedad estable

Se define como:

Entrada: Propiedad estable
Saida: Un booleano **b** con la propiedad**:**

- **y(S1) -> b**
- **b -> y(Sn)**

Con **S1** y **Sn** los estados globales iniciales y finales

Solución:

- b = True => La propiedad de estabilidad se mantiene cuanto el algoritmo termina

- b = False => La propiedad de estabilidad no se mantiene cuando el algoritmo comienza

- Registra un estado global S*
- b := y(S*)

The Distributed Snapshot of K.M Chandy and L. Lamport*

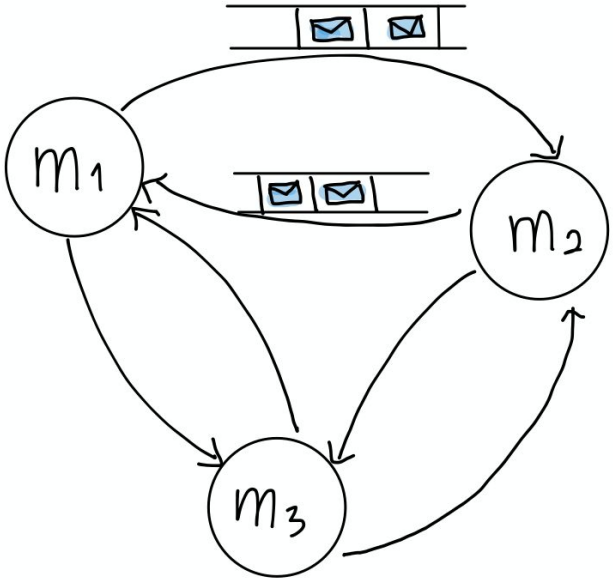
E. Dijkstra

Dijkstra, Edsger W. "The distributed snapshot of KM Chandy and L. Lamport." *Control Flow and Data Flow: Concepts of Distributed Programming*. Springer, Berlin, Heidelberg, 1986. 513-517.

Contenido

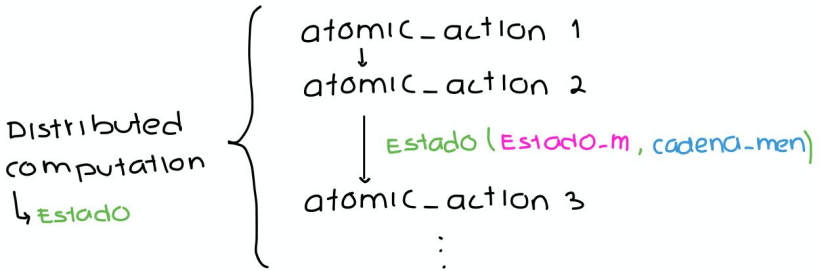
1. Supuestos

Distributed system



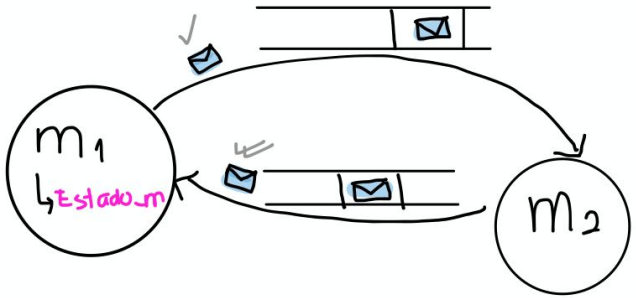
Distributed computation

Sucesión de acciones atómicas (*atomic actions*).



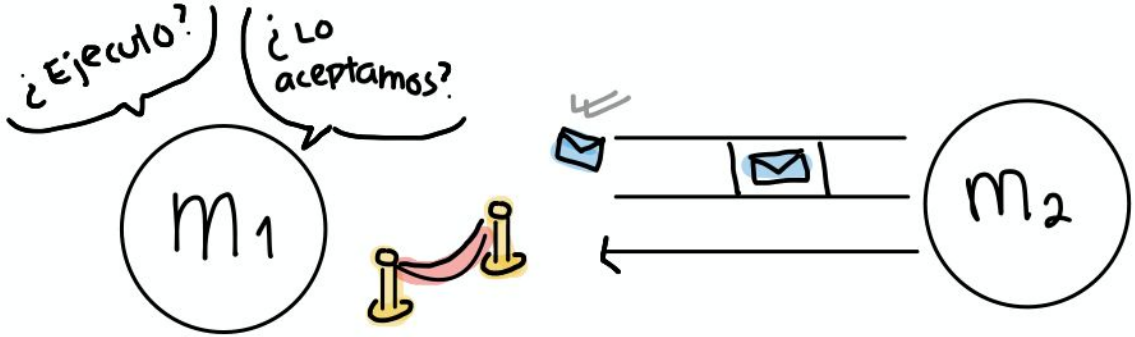
Atomic action

Las ejecutan las máquinas.



Message-acceptance

Para que un mensaje se acepte, tiene que llegar.
La llegada de un mensaje **permite** que se ejecute una acción atómica, pero **no implica** que se ejecutará.



Stable predicate

Si se cumple en un estado en particular, se detendrá en todos los posibles estados posteriores.

Distributed snapshot algorithm

Objetivo: recolectar información de los estados hasta detectar *estabilidad.*

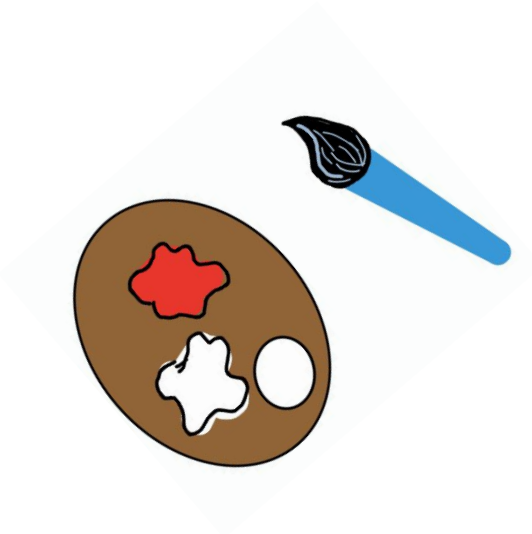
SSS

2. Primera descripción del algoritmo

Snapshot algorithm

```
atomic_action.setColor(machine.color)
message.setColor(atomic_action.color)
```

Inicialmente (S0), todas las **máquinas** y todos los **mensajes** son blancos.
Al ejecutar el algoritmo, cada máquina cambiará su color de blanco a rojo **sólo una vez**.
Eventualmente todas las máquinas, las acciones y los mensajes en los buffer serán rojos (S1).



¿Cuándo hacer rojas las máquinas?

Cuando ningún mensaje rojo sea aceptado en una acción blanca.

Recordar:

```
message.setColor(atomic_action.color)
```

Snapshot state
SSS

1. En cada máquina:
   es el estado en el momento en que ocurre la transición.

2. En cada buffer:
   la cadena de mensajes blancos de él (buffer) aceptados en una acción roja.

Objetivo

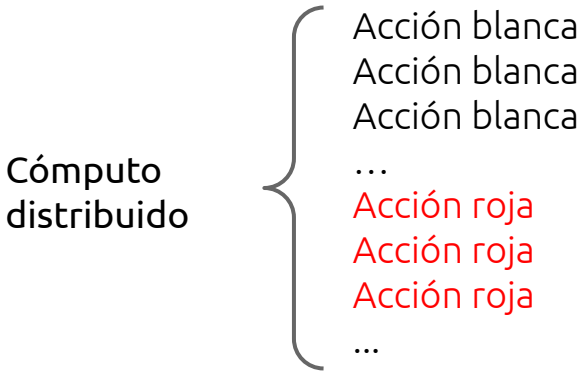
Encontrar un **cómputo equivalente** al algoritmo *snapshot*.

S0 → SSS → S1

Cómputo equivalente

Buscamos encontrar un **cómputo equivalente** al obtenido con el algoritmo snapshot (S0 → SSS → S1).

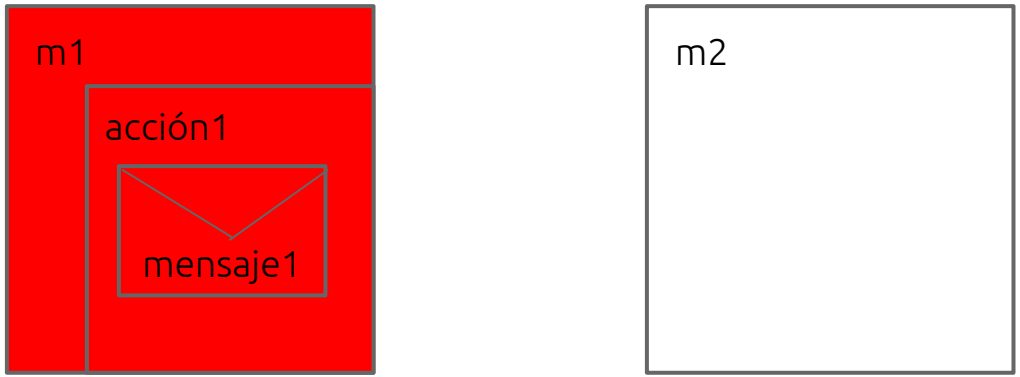
Ese cómputo equivalente debe verse:



¿Cómo obtener la equivalencia?

Conmutar las acciones 🔁

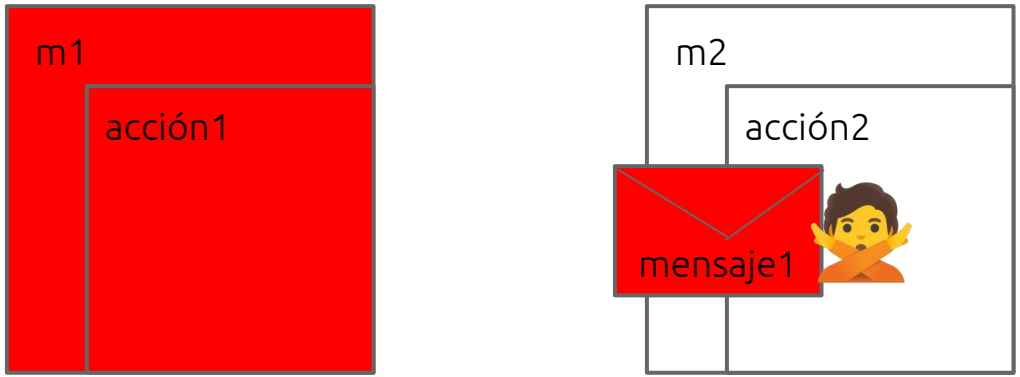
Caso en el que **SÍ** conmutan las acciones:



¿Cómo obtener la equivalencia?

Conmutar las acciones 🔁

Caso en el que **SÍ** conmutan las acciones:



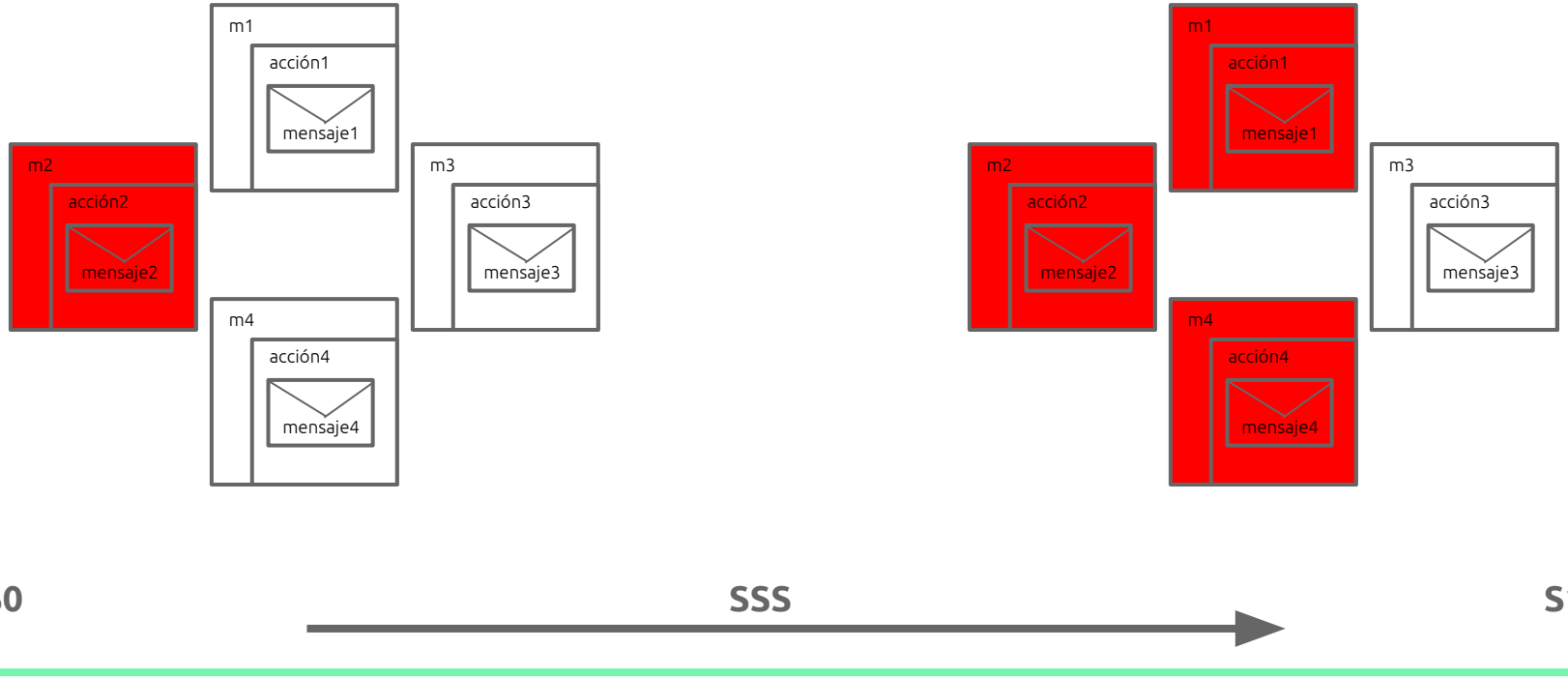
¿Cómo obtener la equivalencia?

Los intercambios reducen *inversions*.

Inversion: cuando una acción roja precede a una blanca (como el ejemplo anterior).

Después de un número finito de inversiones, producimos el **cómputo equivalente** (todas las acciones blancas van antes que las rojas).

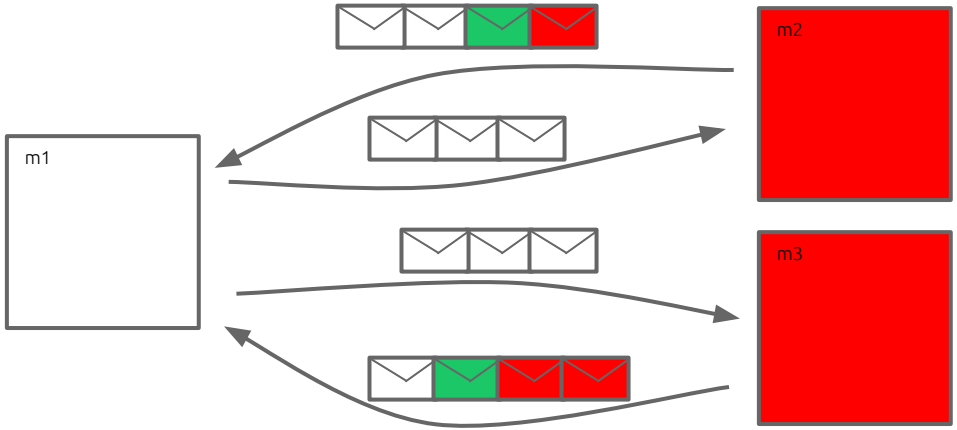
¿Cómo obtener la equivalencia?



3. Detalles de implementación

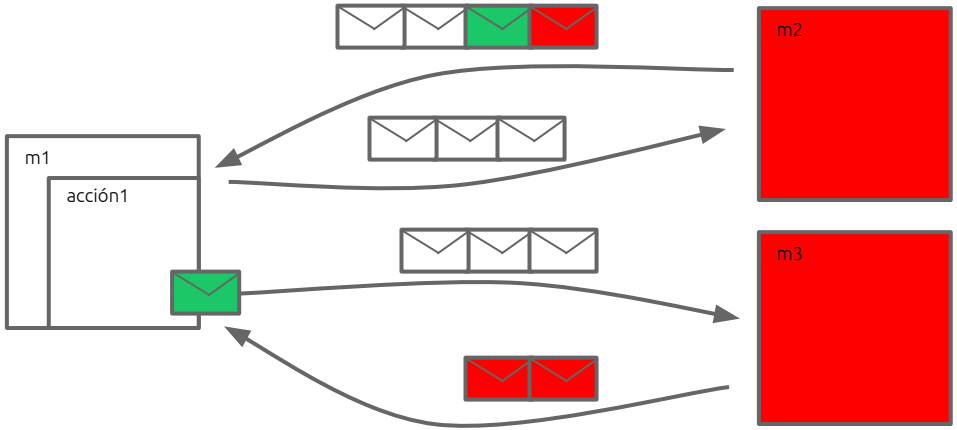
1. Cadena de mensajes para buffers de entrada

Una máquina roja tiene que registrar la cadena de mensajes blancos aceptados por cada buffer de entrada.



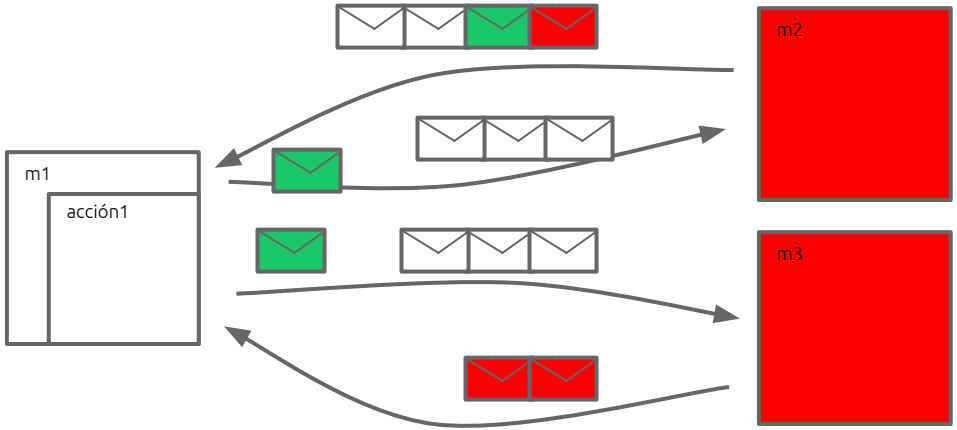
- *markers*

2. Cuándo cambiar de color la máquina



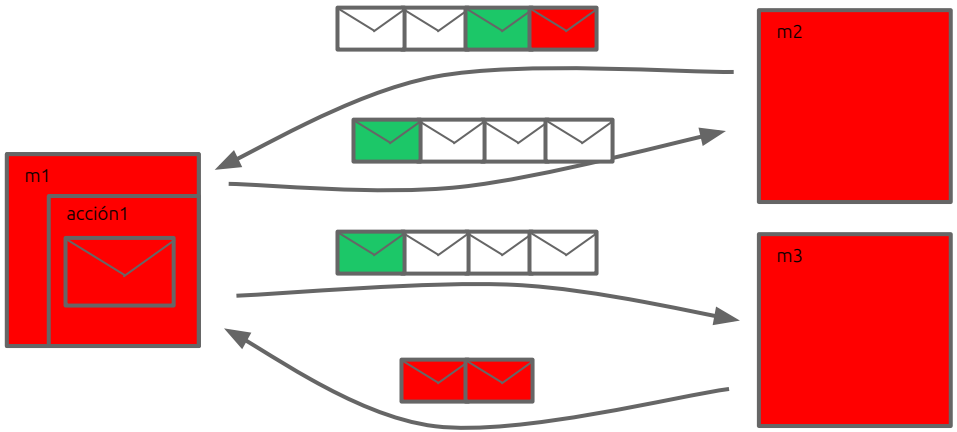
- *markers*

2. Cuándo cambiar de color la máquina



- *markers*

2. Cuándo cambiar de color la máquina



- *markers*