# Consensus Number

魏恒峰

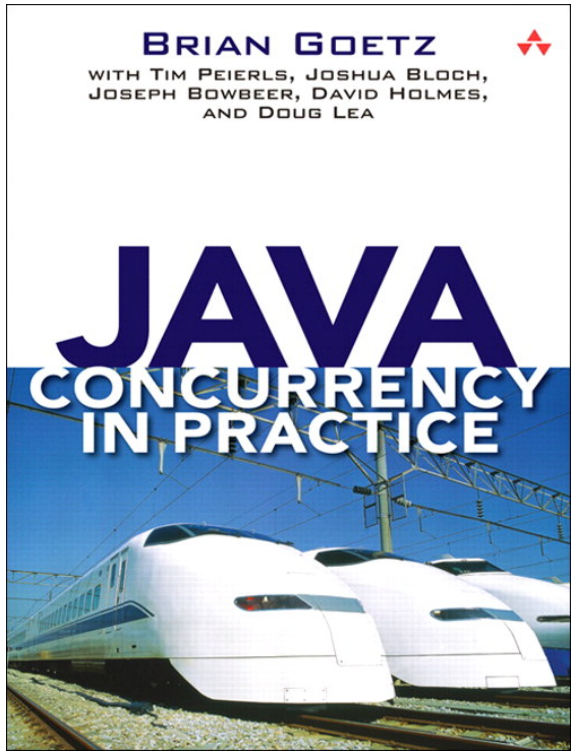
hfwei@nju.edu.cn

2017 年 12 月 14 日


Feel Free to
Ask Questions

Are you Familar with **Concurrent Programming**?

The Key: **Synchronization**!




Using the **Synchronization Primitives**
Provided by Your Favorite Languages.

`synchronized`
`Semaphore`

`BlockingQueue`
`ConcurrentMap`

`Phaser`
`Barrier`

| | | |
|---|---|---|
| synchronized | BlockingQueue | Phaser |
| Semaphore | ConcurrentMap | Barrier |

```
class AtomicInteger:
  get()
  set(int newValue)

  getAndIncrement()
  getAndDecrement()
  getAndSet(int newValue)

  compareAndSet(int expectedValue, int newValue)
```

```
compareAndSet(int expectedValue, int newValue)
compareAndSwap(int expectedValue, int newValue)
```

# CAS — CMPXCHG

**impl.**

**usage**

# Tasks: Implementing Consensus using Given Primitives.




# Consensus


"It looks like we have a consensus."

**Propose**

**Decide**

**Propose**

**Decide**

Definition (The Consensus Problem)

- Agreement: All (non-faulty) processes must **agreen on the same value**.
- Validity: The common decision value must be the value **proposed** by some process.
- Termination: Each (non-faulty) process must **eventually** decide on a value.

**Definition (Consensus Protocol)**

(Informally), a consensus protocol is a system of $n$ processes that communicate through a set of shared objects.

Propose

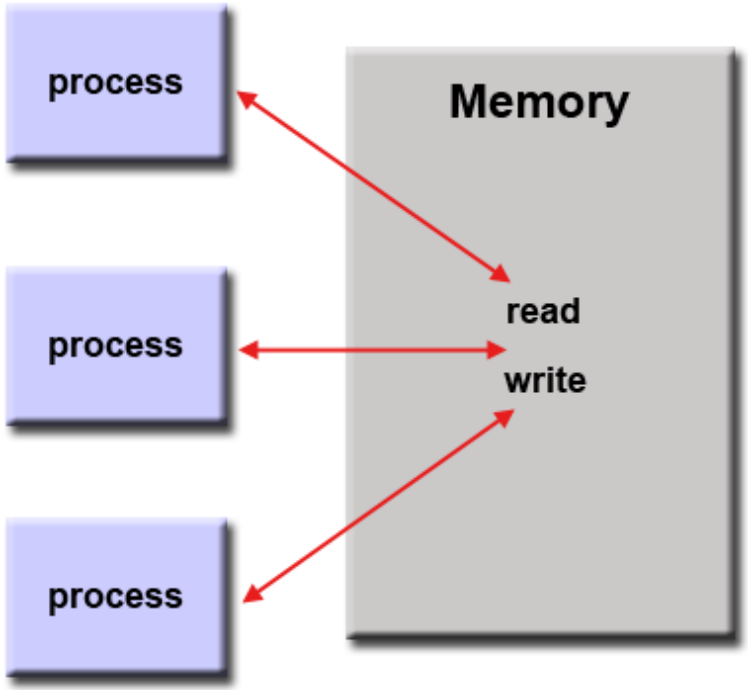
Communicate

Decide

More clarification on “termination”

termination requires wait-free implementation of the consensus object

binary consensus problem




Processes communicate through shared objects.

(redraw)

consensus object (fig here)

```
1    public interface Consensus<T> {
2      T decide(T value);
3    }
```

consensus protocol

```
1  public abstract class ConsensusProtocol<T>
2                        implements Consensus<T> {
3    protected T[] proposed = (T[]) new Object[N];
4
5    void propose(T value) {
6      proposed[ThreadID.get()] = value;
7    }
8
9    public abstract T decide(T value);
10 }
```

implement X using Y

```
1  public class CASConsensus<T>
2       extends ConsensusProtocol<T> {
3    private final int FIRST = -1;
4    private AtomicInteger r = new AtomicInteger(FIRST);
5
6    @Override
7    public T decide(T value) {
8      propose(value);
9
10     int i = ThreadID.get();
11     if (r.compareAndSet(FIRST, i))   // I won
12       return proposed[i];
13     else                             // I lose
14       return proposed[r.get()];
15   }
16 }
```

**Theorem (Computational Power of CAS)**

*A register providing compareAndSet() and get() methods can solve the consensus problem for any number of threads.*

**Theorem (Computational Power of CAS)**

*A register providing compareAndSet() and get() methods can solve the consensus problem for any number of threads.*

In terms of "Consensus Number":

**Theorem (Consensus Number of CAS)**

*A register providing compareAndSet() and get() methods has infinite* **consensus number**.

## Definition (Solving Consensus)

"$X$ solves $n$-process consensus" if there exists a consensus protocol

$$P_1, \cdots, P_n; W, X$$

- $W$ is a set of (atomic) read/write registers;
- $W$ and $X$ may be initialized to any state.

### Definition (Consensus Number)

The **consensus number** for $X$ is the largest $n$ for which $X$ solves $n$-thread consensus.

If no largest $n$ exists, the consensus number is said to be infinite.

## Lemma ($Y$ Implements $X$)

## Theorem (Consensus Number as ...)

in the following, main results (table)

beautiful ideas and proofs


前方高能
-WARNING-


KEEP
CALM
AND
TAKE A
DEEP BREATH

# Protocol in terms of set:

**Protocol**

$$\mathcal{P} = \{\text{All executions of this protocol}\}$$

**Execution**

$$e = \sigma_0 \xrightarrow{o_1} \sigma_1 \xrightarrow{o_2} \cdots \xrightarrow{o_{n-1}} \sigma_n$$

**State**

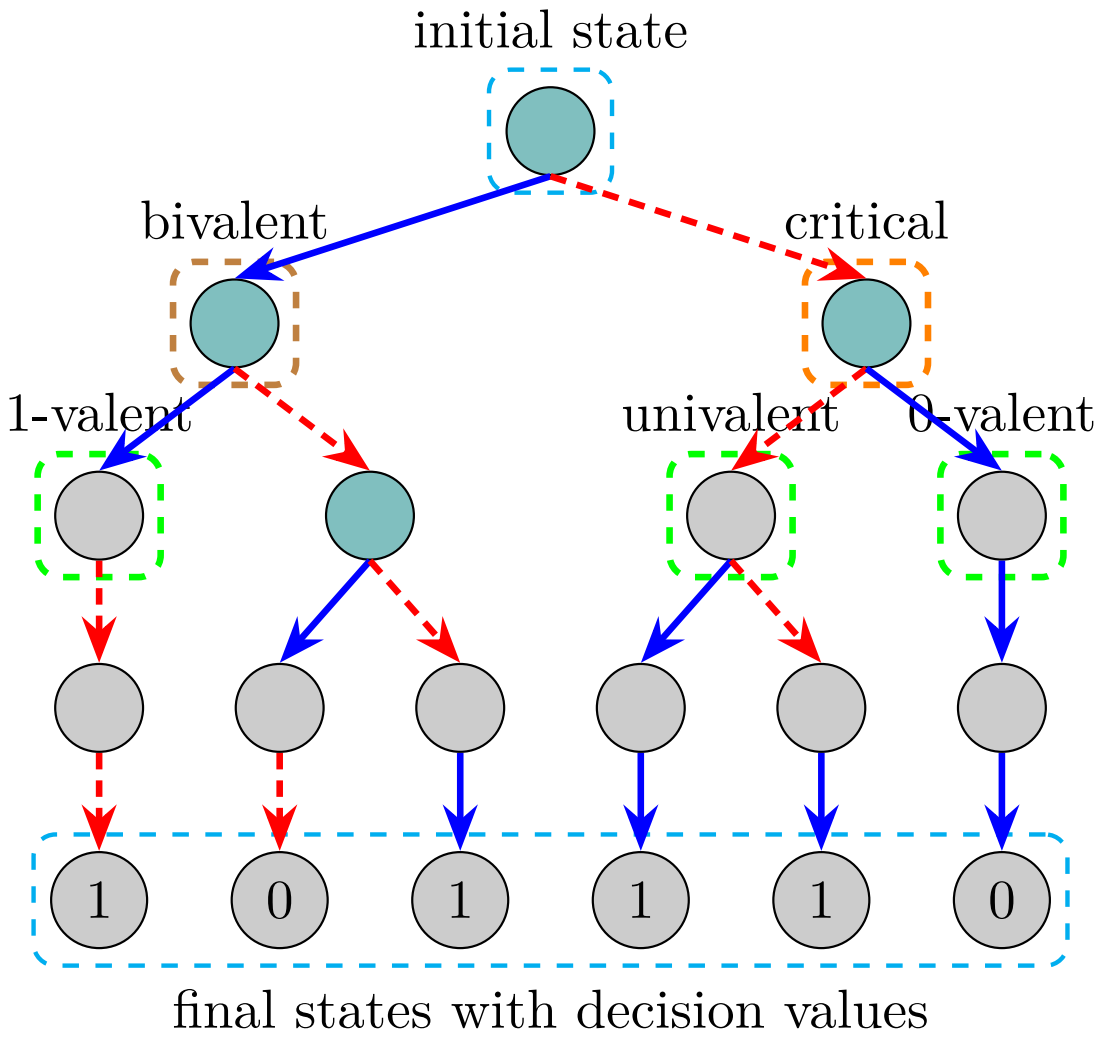
$\sigma_i$ : states of individual threads + states of shared objects

**Operation**

$o_i$ : method calls to a shared object

# Modeling $\mathcal{P}$ as a Computation "Tree"

## (Binary Consensus for 2 Threads)



## Theorem (Bivalent Initial State)

*Every 2-thread binary consensus protocol has a bivalent initial state.*

## Theorem (Bivalent Initial State)

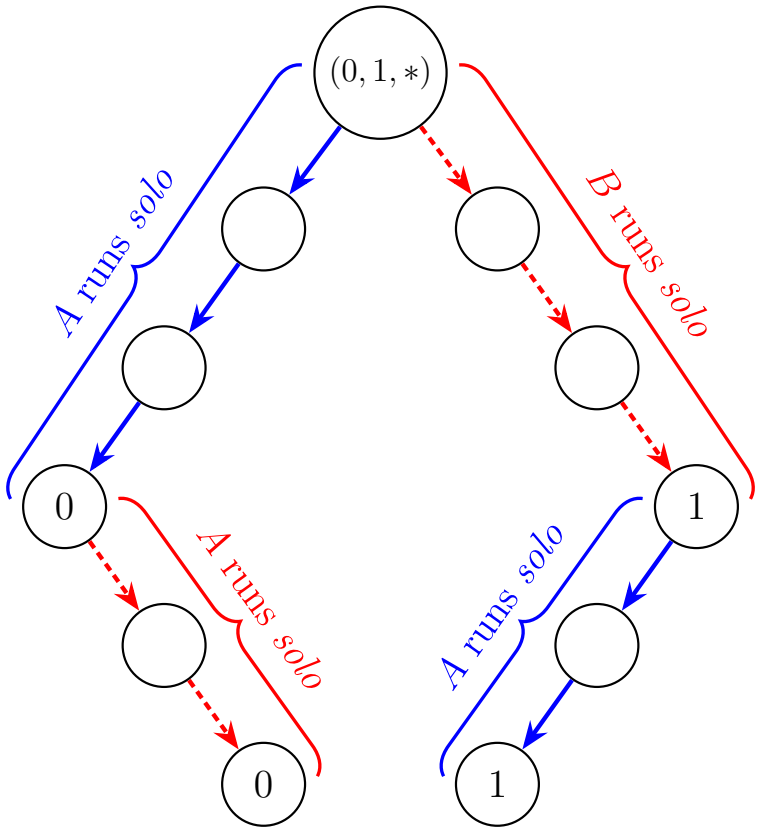
*Every 2-thread binary consensus protocol has a bivalent initial state.*

$$(A, B, O) : (0, 0, *) \quad (1, 1, *) \quad (0, 1, *) \quad (1, 0, *)$$

## Theorem (Bivalent Initial State)

*Every 2-thread binary consensus protocol has a bivalent initial state.*

$$(A, B, O) : (0, 0, *) \quad (1, 1, *) \quad (0, 1, *) \quad (1, 0, *)$$



## Lemma (Bivalent Initial State)

*Every $n$-thread (binary) consensus protocol has a bivalent initial state.*

## Lemma (Bivalent Initial State)

*Every $n$-thread (binary) consensus protocol has a bivalent initial state.*

## Theorem (Existence of Critical States)

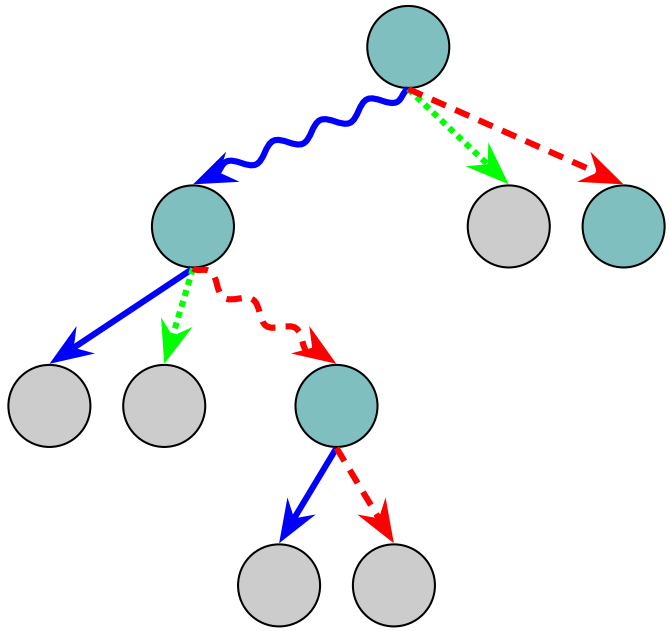
*Every wait-free consensus protocol has a critical state.*

## Lemma (Bivalent Initial State)

*Every $n$-thread (binary) consensus protocol has a bivalent initial state.*

## Theorem (Existence of Critical States)

*Every wait-free consensus protocol has a critical state.*




relax
breathe
smile

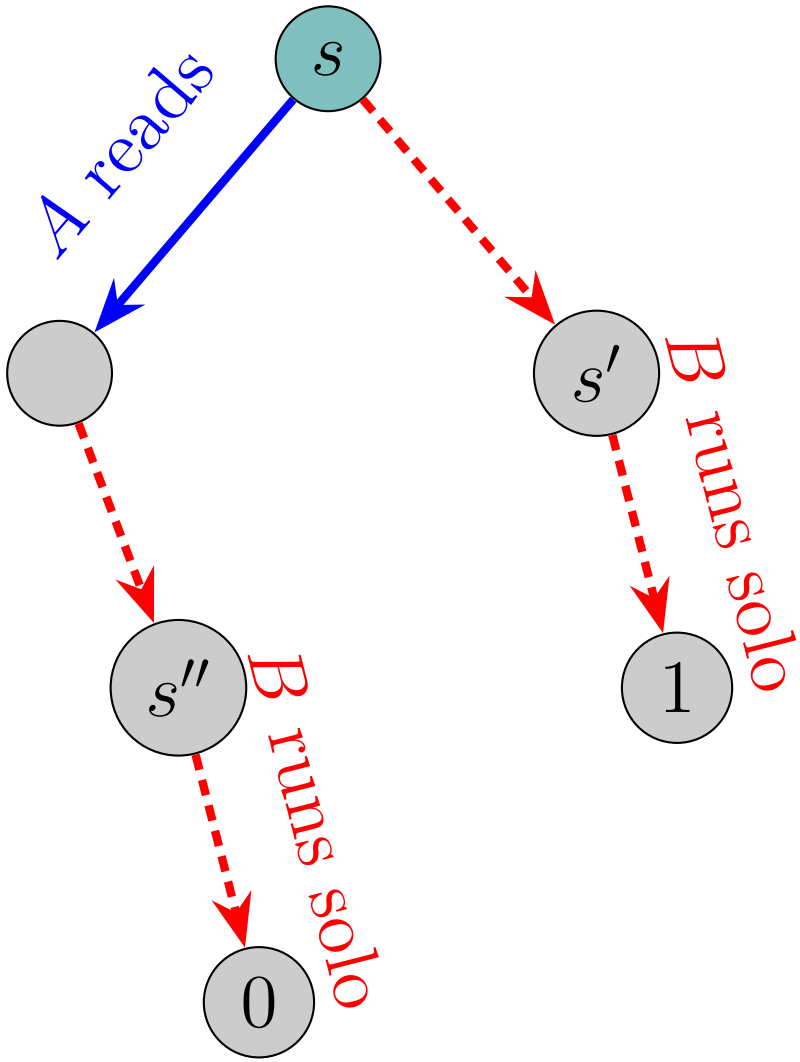
Theorem (Consensus Number of Atomic Registers)

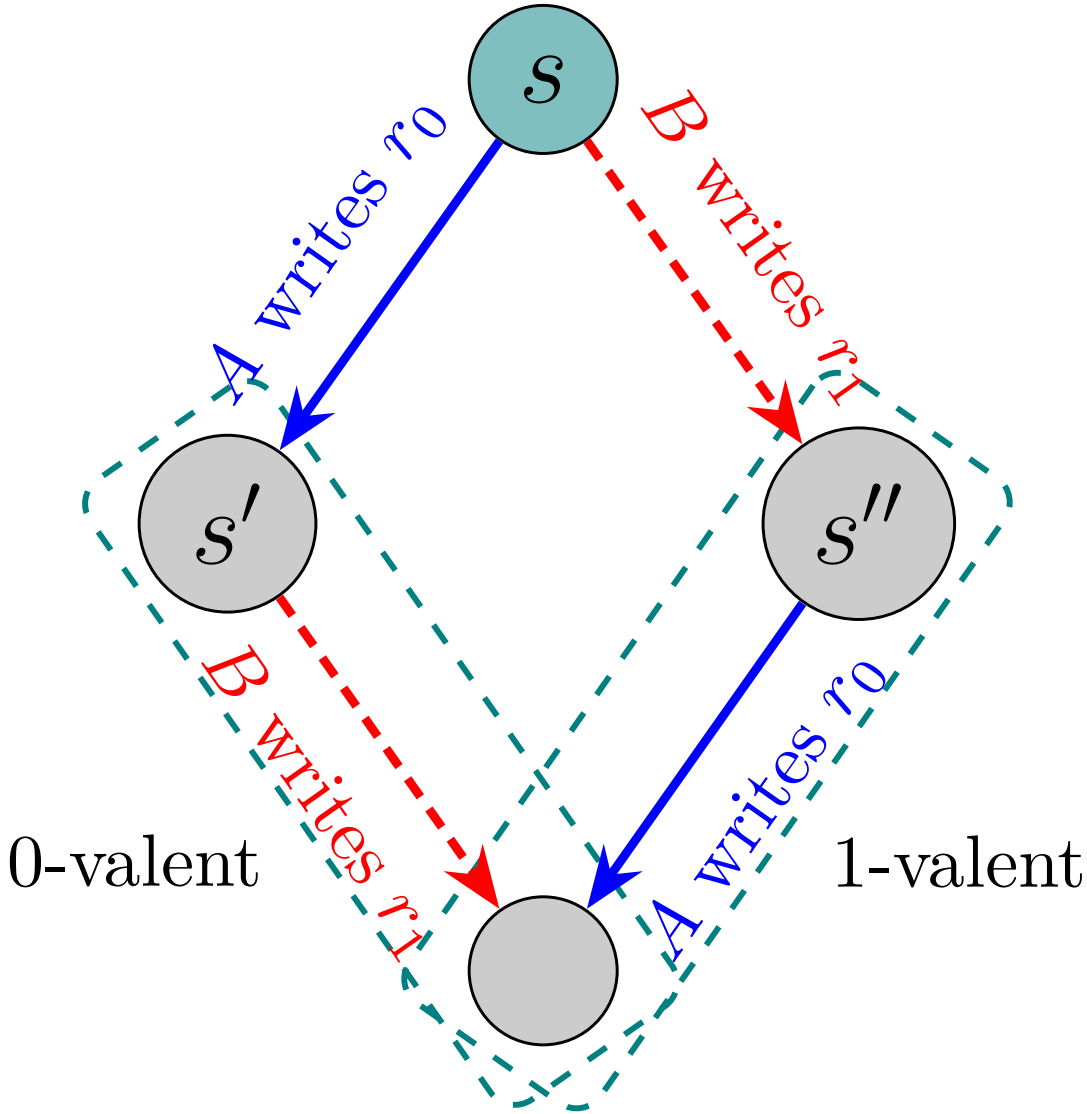
*Atomic registers have consensus number* $1$.

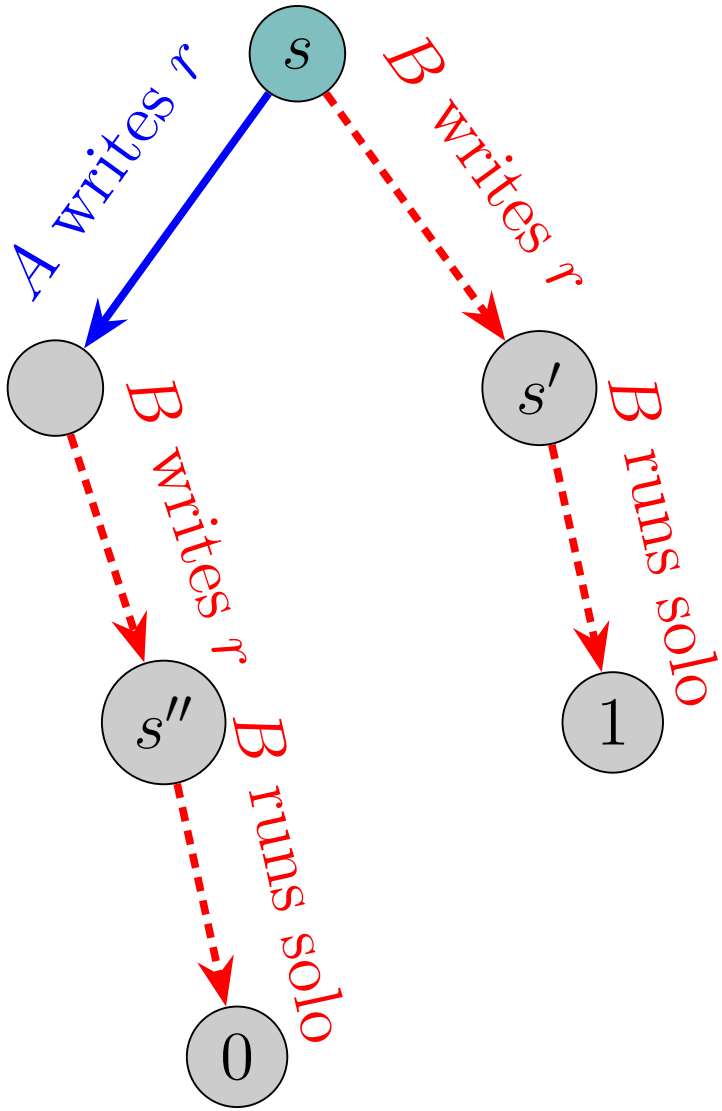
Proof.

(1) Run the protocol until it reaches a critical state $s$.

(2) What is the next?


$s$
A reads
$s'$
B runs solo
$s''$
B runs solo
1
0


$s$
$A$ writes $r_0$
$B$ writes $r_1$
$s'$
$s''$
$B$ writes $r_1$
$A$ writes $r_0$
0-valent
1-valent


$s$
$A$ writes $r$
$B$ writes $r$
$s'$
$B$ runs solo
1
$B$ writes $r$
$s''$
$B$ runs solo
0

## Theorem


Thank
You!