Event-B Course

# 3. A Mechanical Press Controller

Jean-Raymond Abrial

September-October-November 2011

# Outline

1. Informal presentation of the example

2. Presentation of some design patterns

3. Writing the requirement document

4. Proposing a refinement strategy

5. Development of the model using refinements and design patterns

# 1. Informal Presentation of the Example

# Presentation of the Example

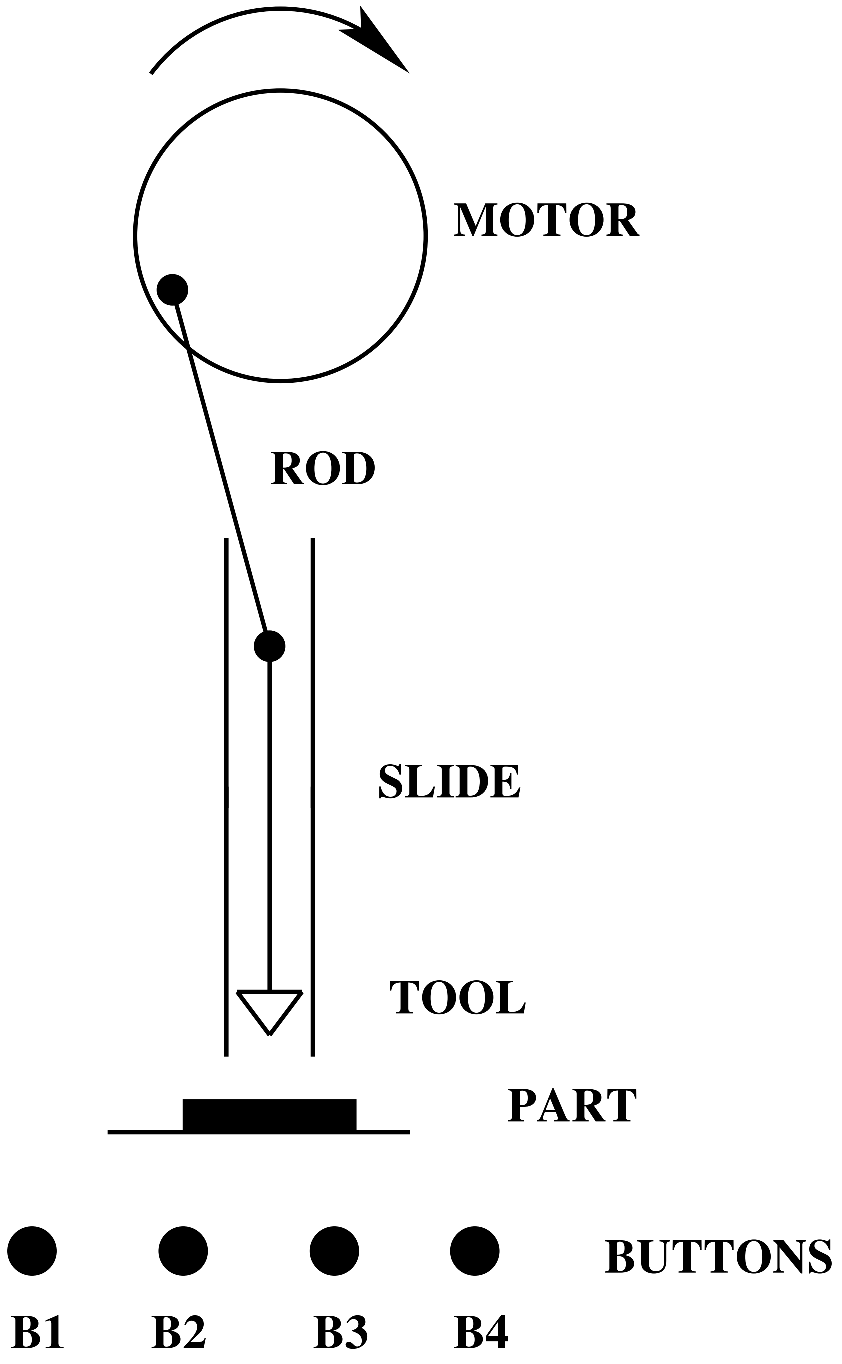
- A mechanical press controller

- Adapted from a real system

- The real system is coming from INRST:

  Institut National de la Recherche sur la Sécurité du Travail

# Mechanical Press Schema



## Basic Equipment

- A Vertical Slide with a tool at its lower extremity

- An electrical Rotating Motor

- A Rod connecting the motor to the slide.

- A Clutch engaging or disengaging the motor on the rod

- When the clutch is disengaged, the slide stops “immediately”

# Basic Commands

- Button B1: start motor

- Button B2: stop motor

- Button B3: engage clutch

- Button B4: disengage clutch

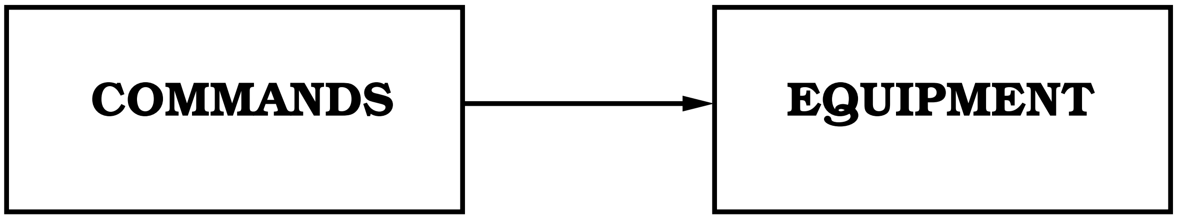
# Basic User Actions

- Action 1: Change the tool at the lower extremity of the slide

- Action 2: Put a part to be treated under the slide

- Action 3: Remove the part

# First Schematic View



COMMANDS

EQUIPMENT

# A Typical User Session

1. start the motor (button B1)
2. change the tool (action 1)
3. put a part (action 2),
4. engage the clutch (button B3): the press now works,
5. disengage the clutch (button B4): the press does not work,
6. remove the part (action 3),
7. repeat zero or more times steps 3 to 6,
8. repeat zero or more times steps 2 to 7,
9. stop the motor (button B2).

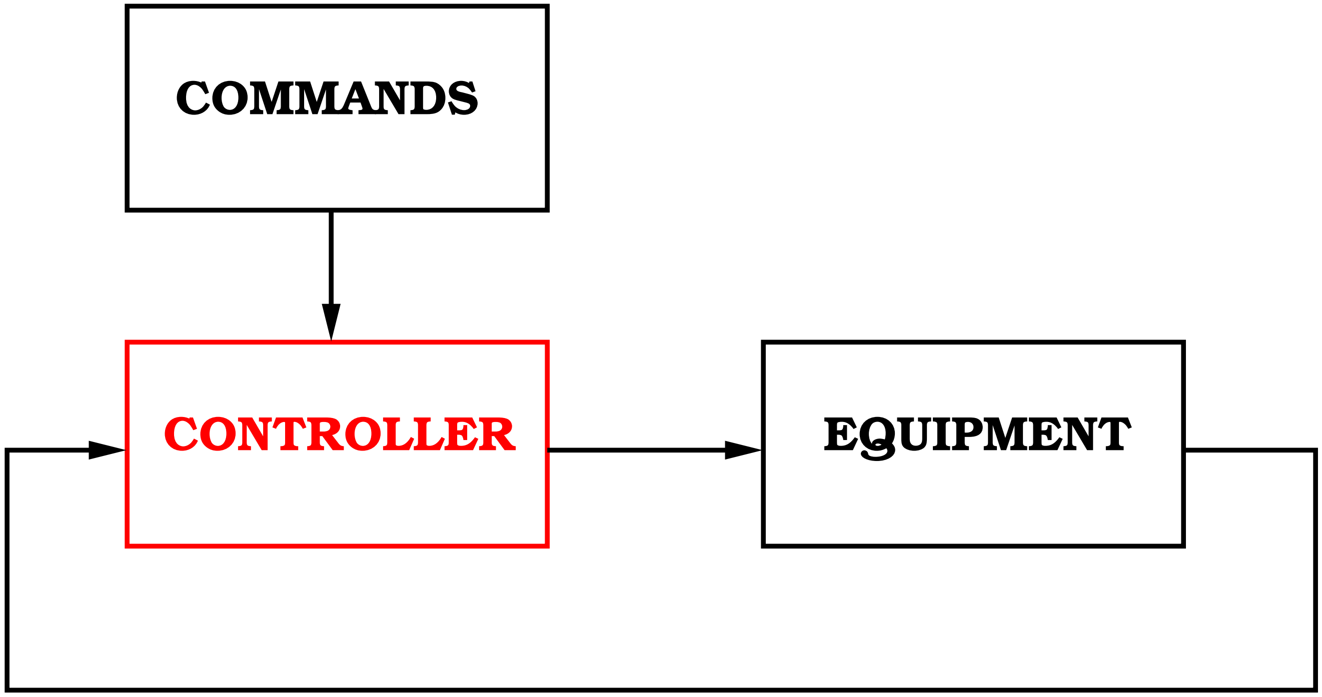
# Danger: Necessity of a Controller

- step 2 (change the tool),

- step 3 (put a part),

- step 6 (remove the part) are all DANGEROUS

# Second Schematic View

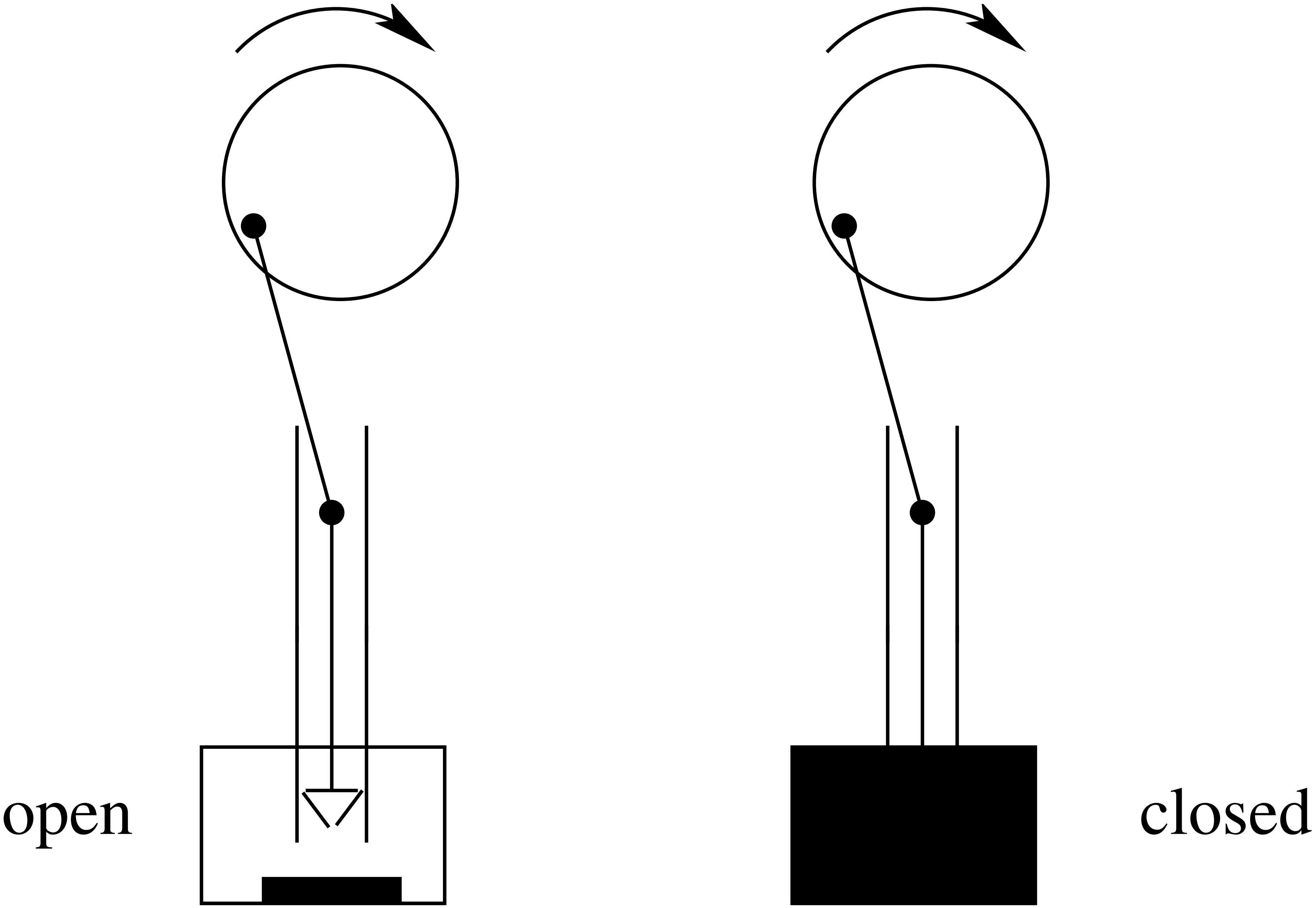


# More Elaborate Commands for Protecting the User

- Controlling the way the clutch is engaged or disengaged

- Protection by means of a Front Door

# The Front Door



# The Front Door: Behavior

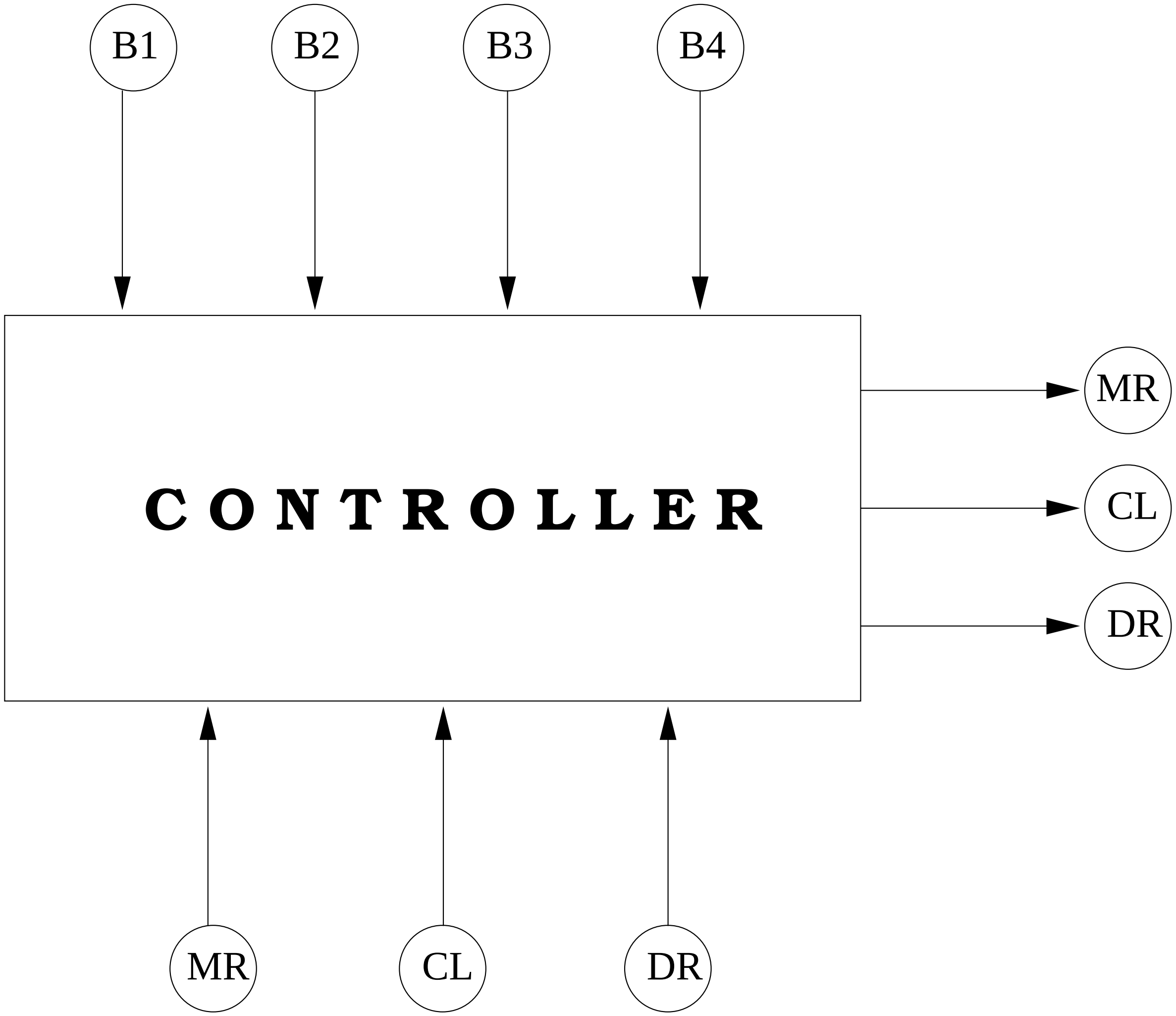
- Initially, the door is open

- When the user presses button B3 to engage the clutch,
  the door is first closed BEFORE engaging the clutch

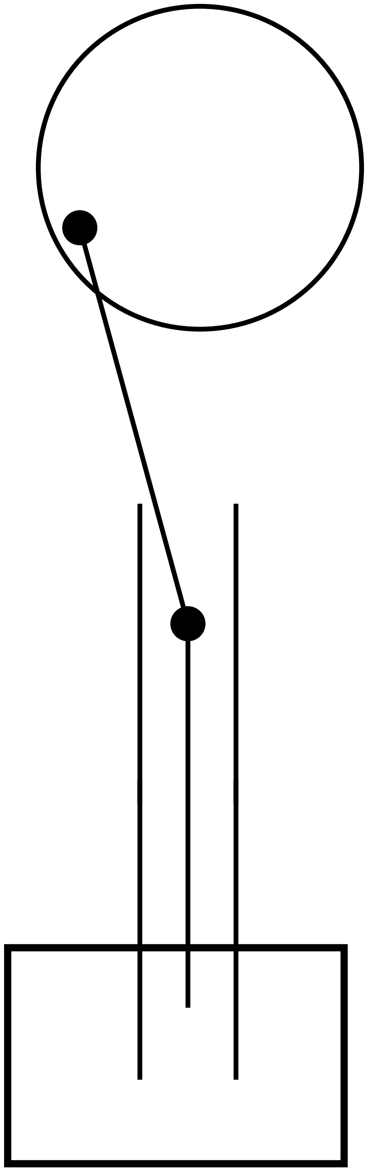
- When the user presses button B4 to disengage the clutch,
  the door is opened AFTER disengaging the clutch

- Notice: The door has no button.

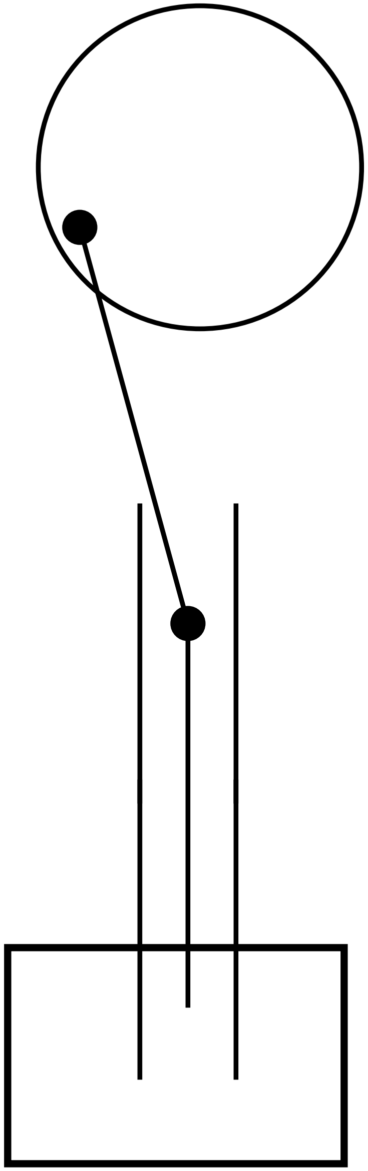
# Summary of Connections



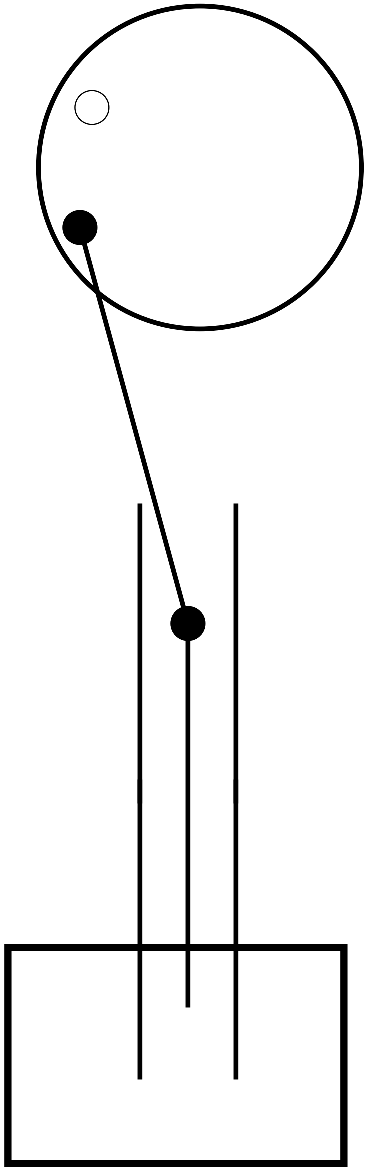
# Initial Situation



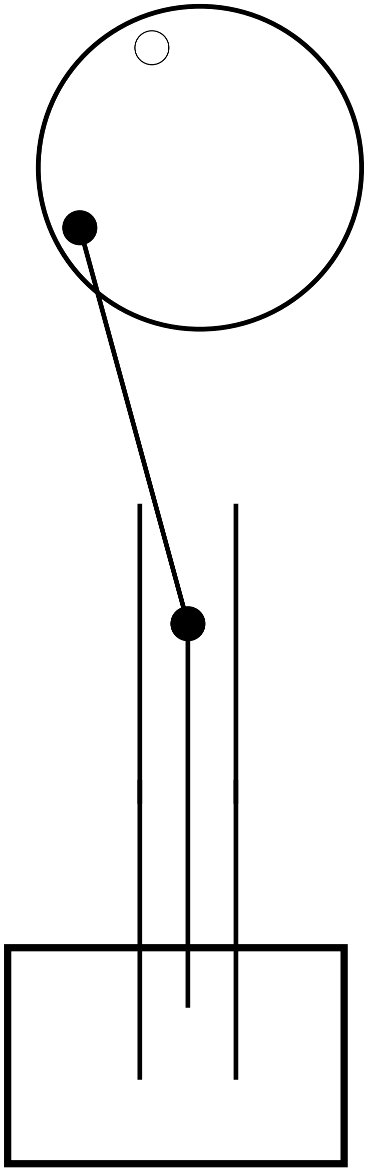
# Starting the Motor: Pressing Button B1



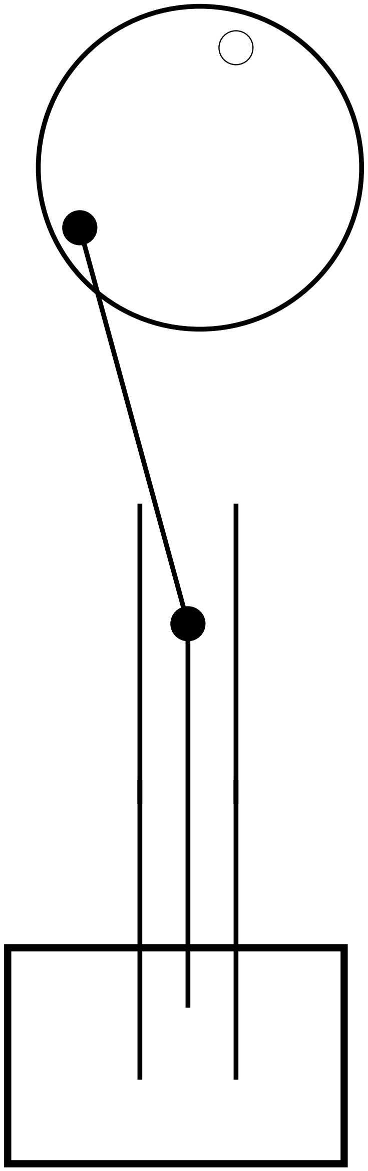
# The Motor Works



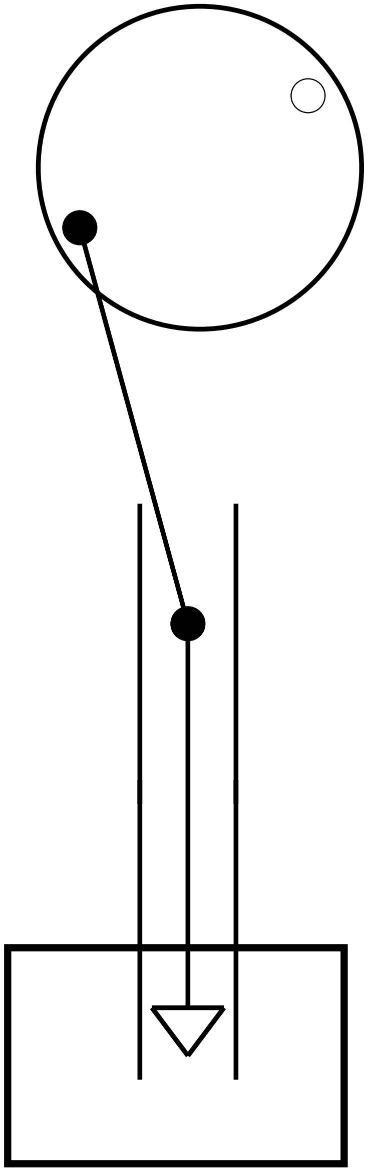
# The Motor Works



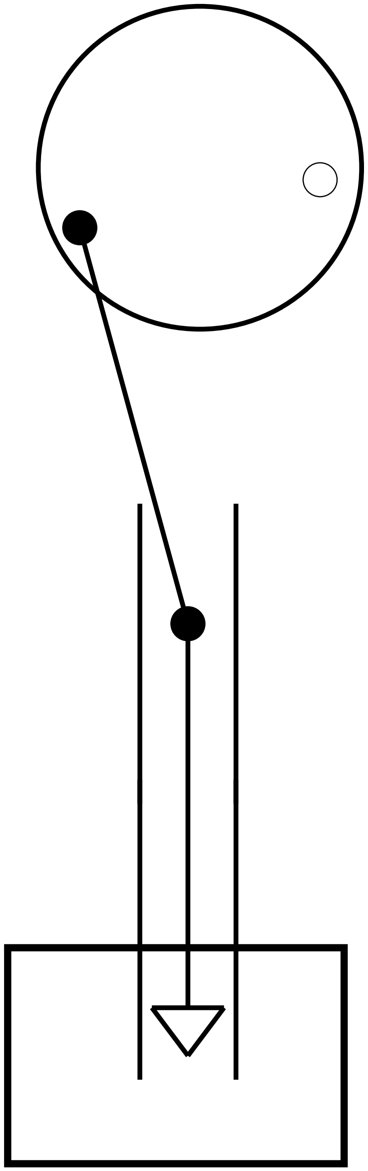
# The Motor Works



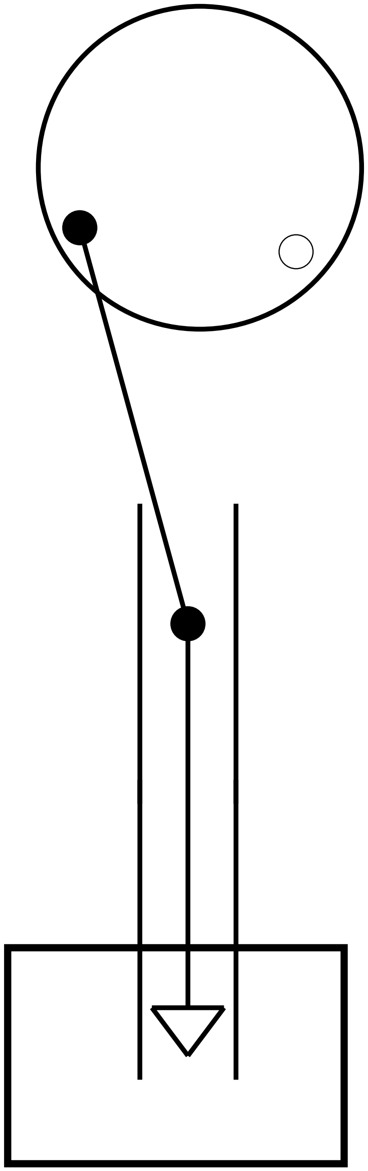
# Adding a Tool



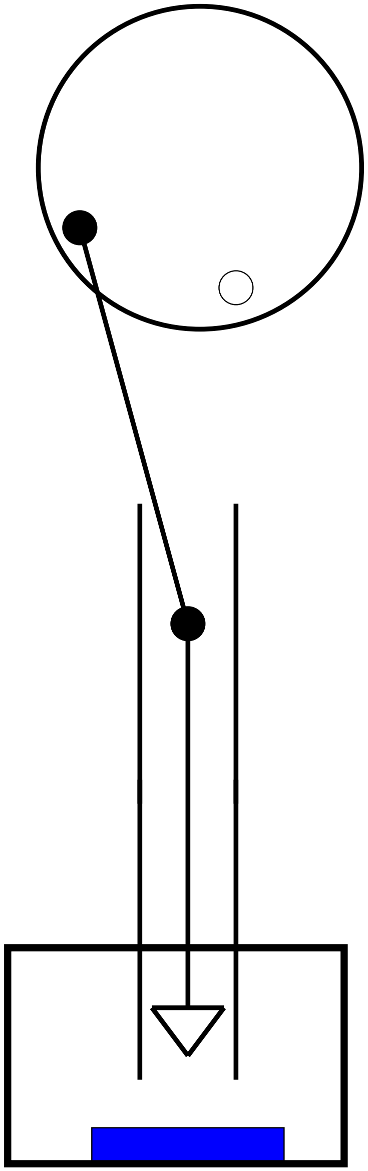
# The Motor Works



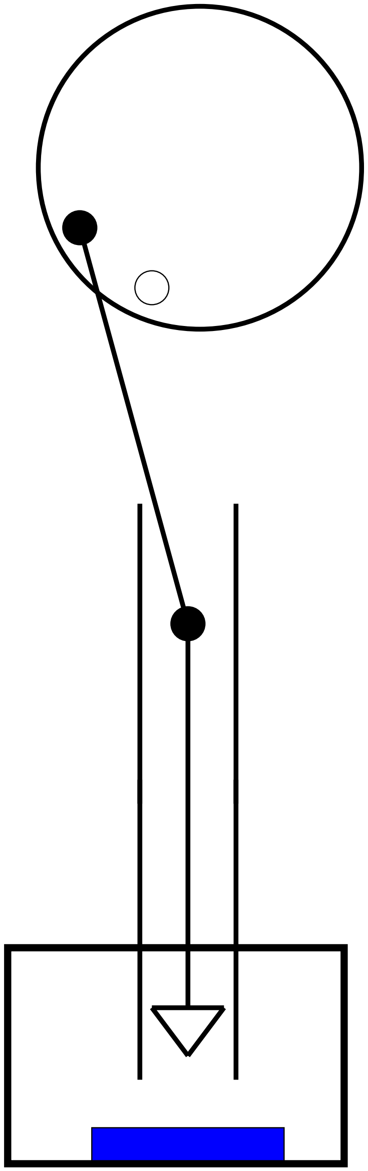
# The Motor Works



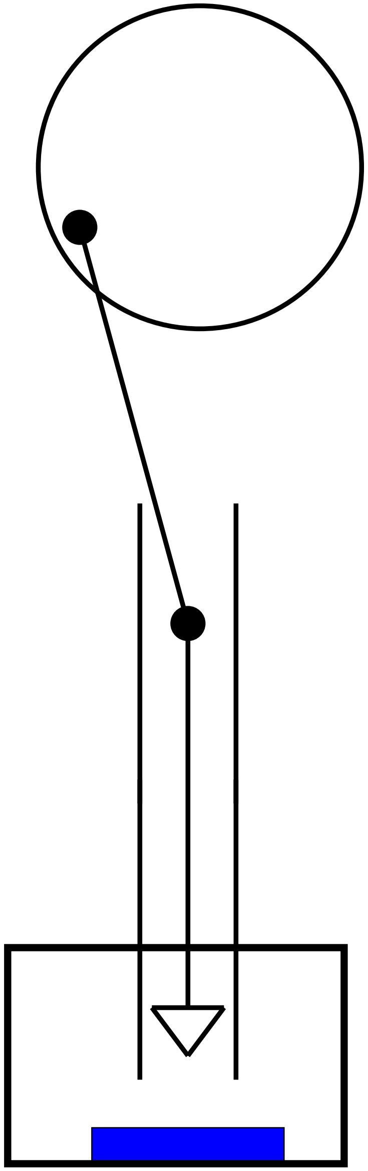
# Putting a Part



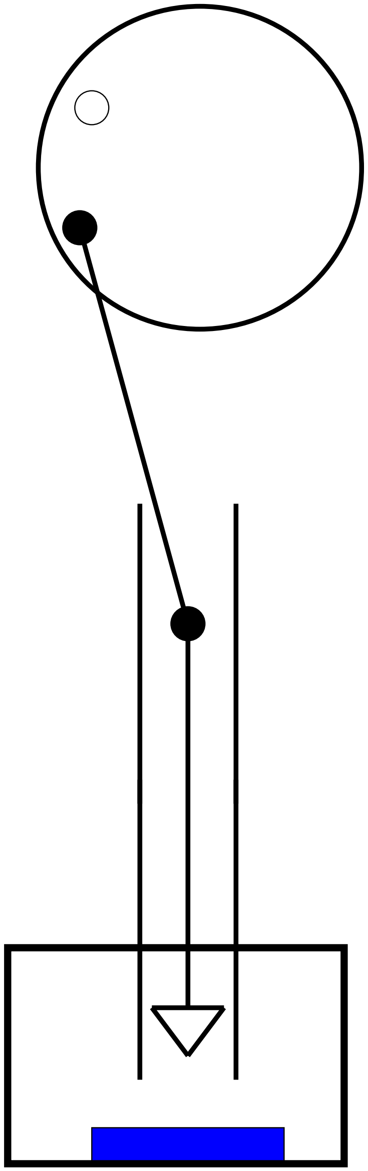
# The Motor Works



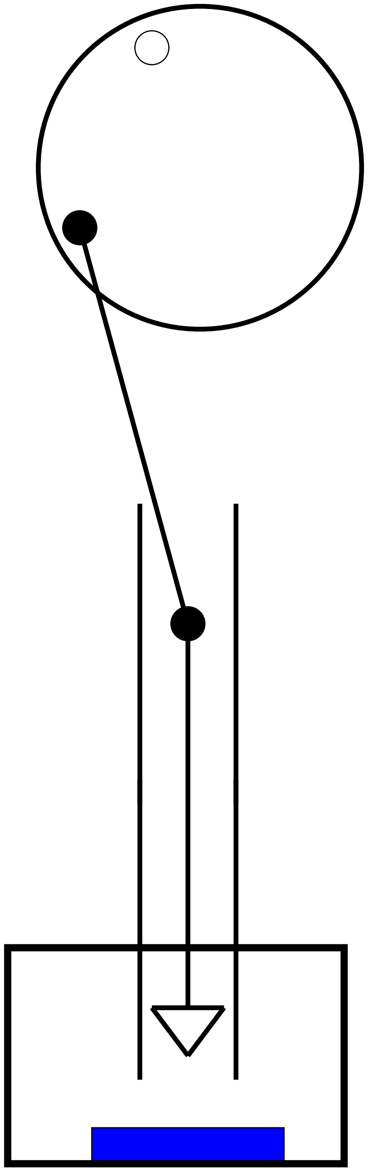
# The Motor Works



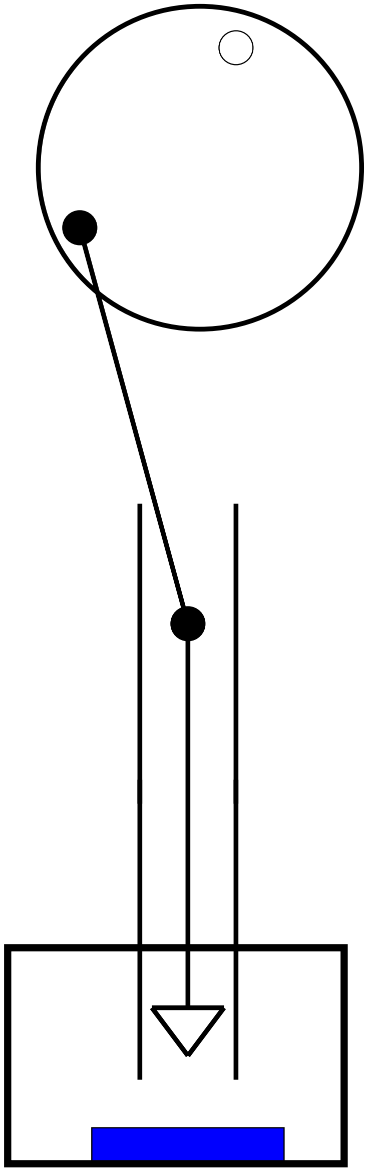
# The Motor Works



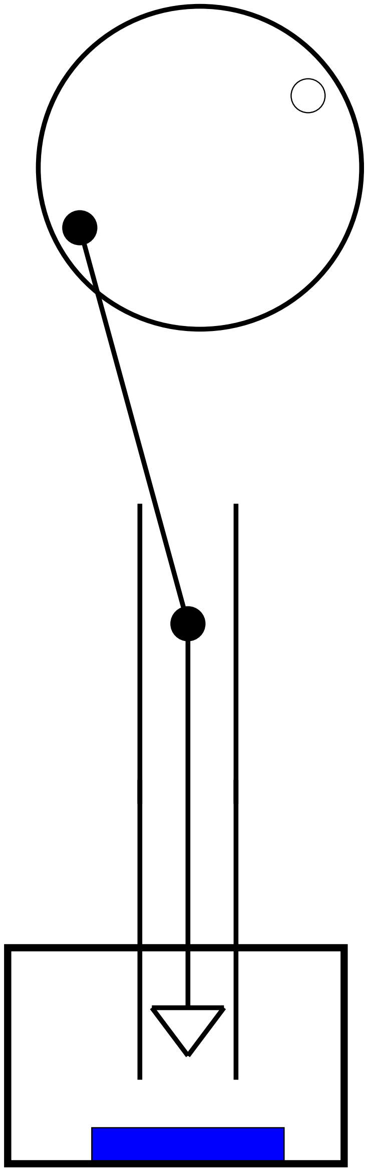
# The Motor Works



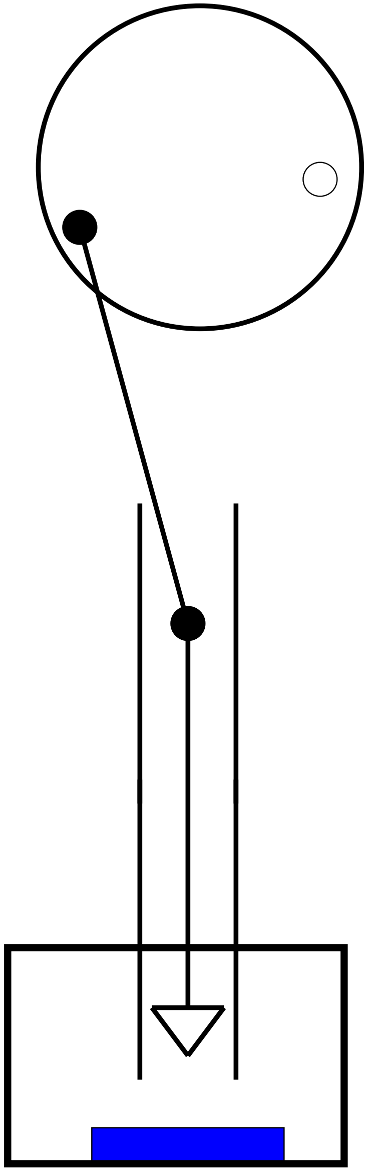
# The Motor Works



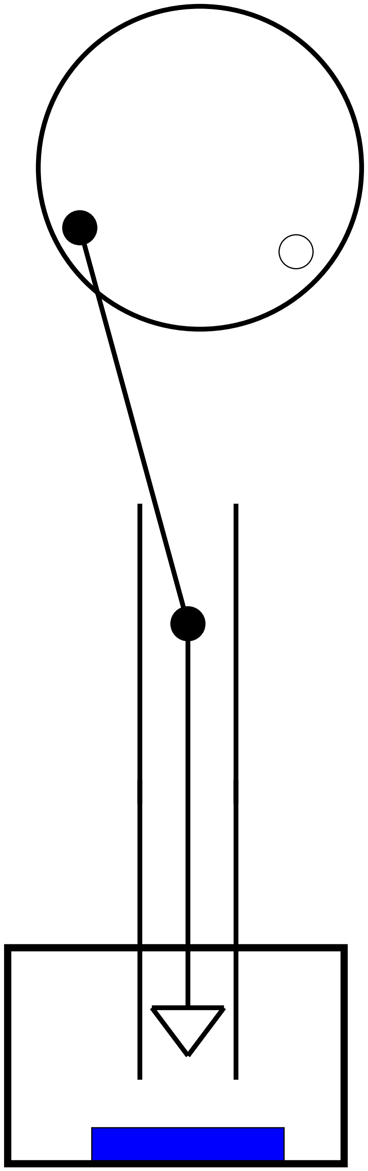
# The Motor Works



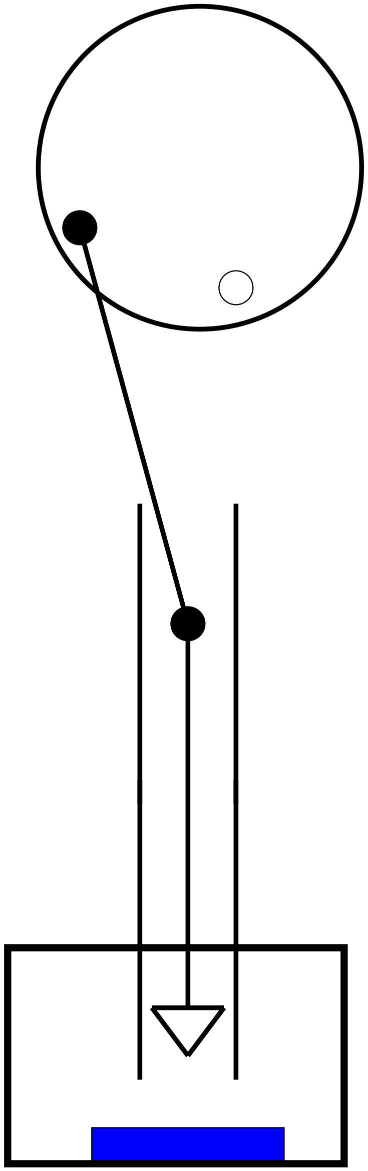
# The Motor Works



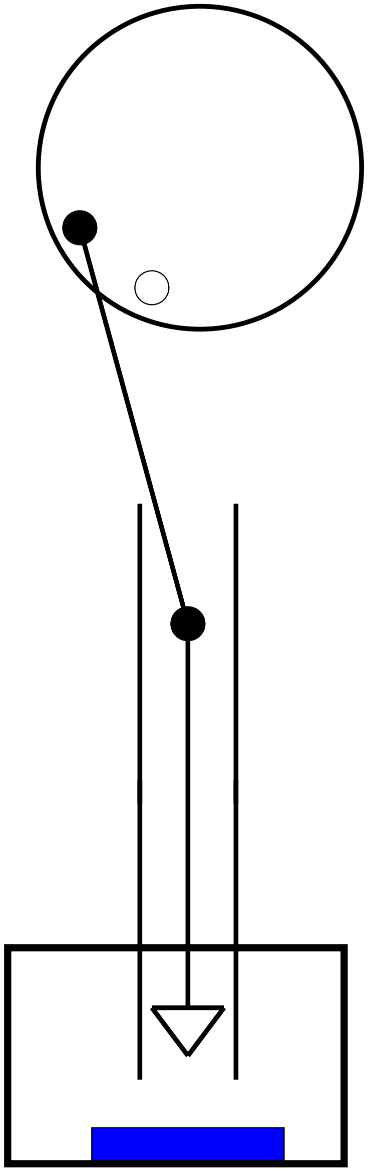
# The Motor Works



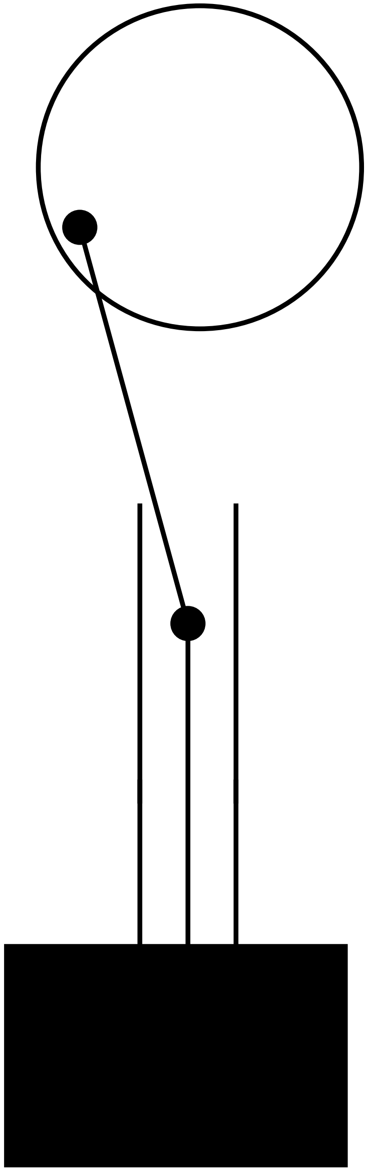
# The Motor Works



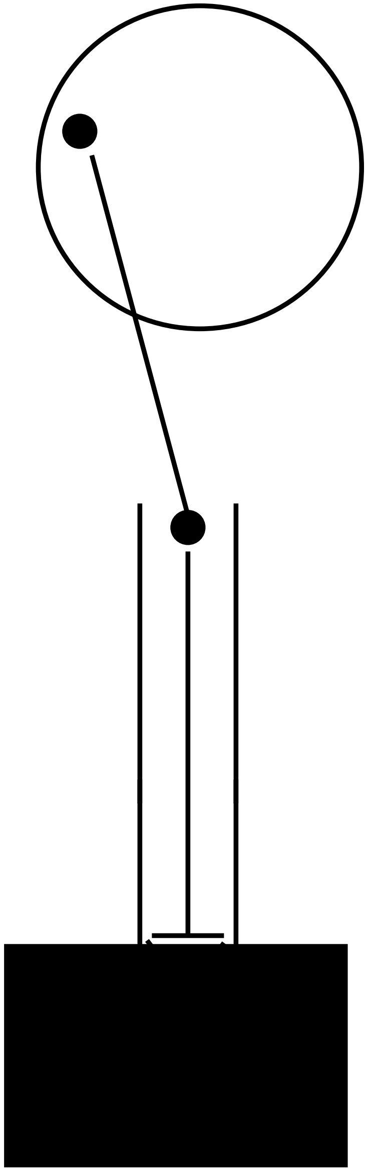
# The Motor Works



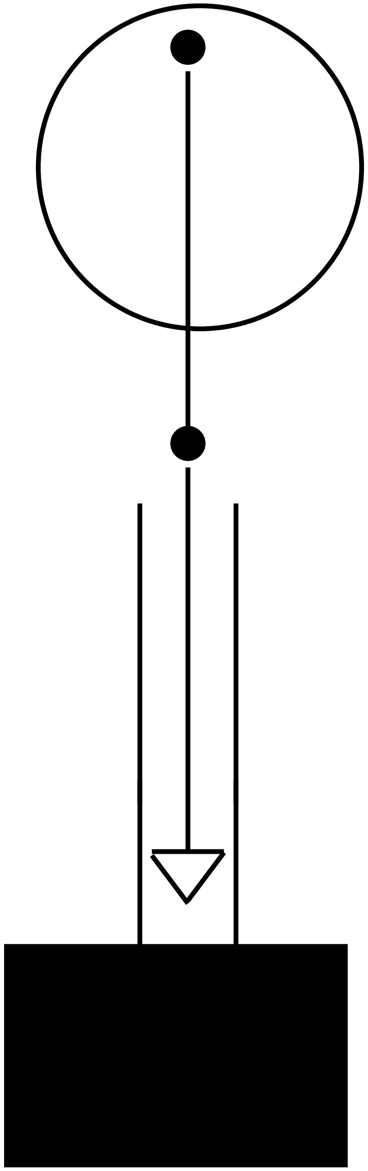
# Engaging the Clutch: Pressing Button B3



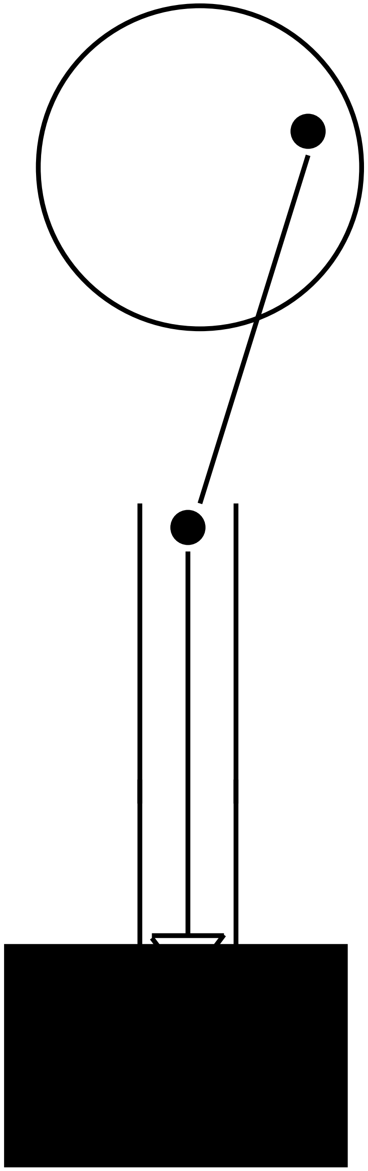
# The Press Works



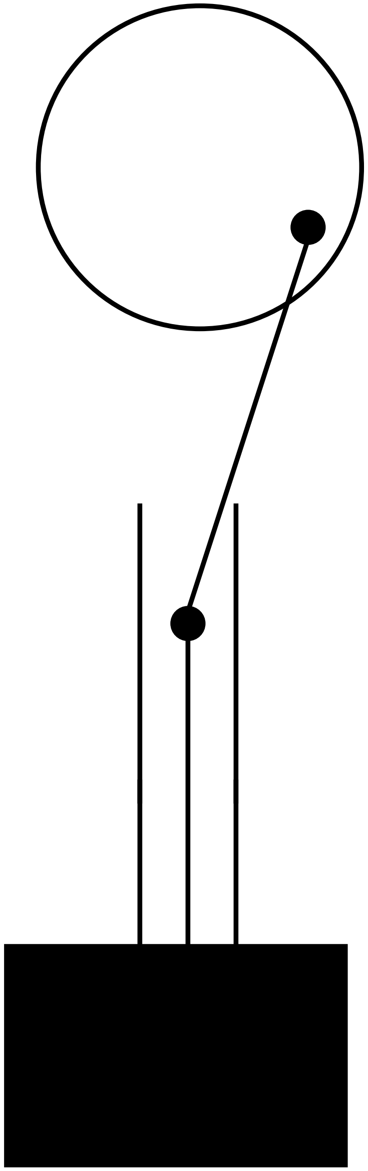
# The Press Works



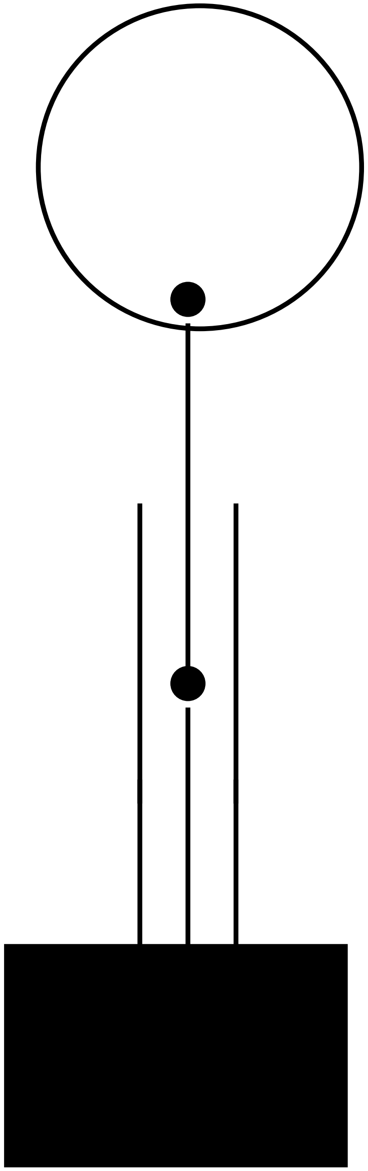
# The Press Works



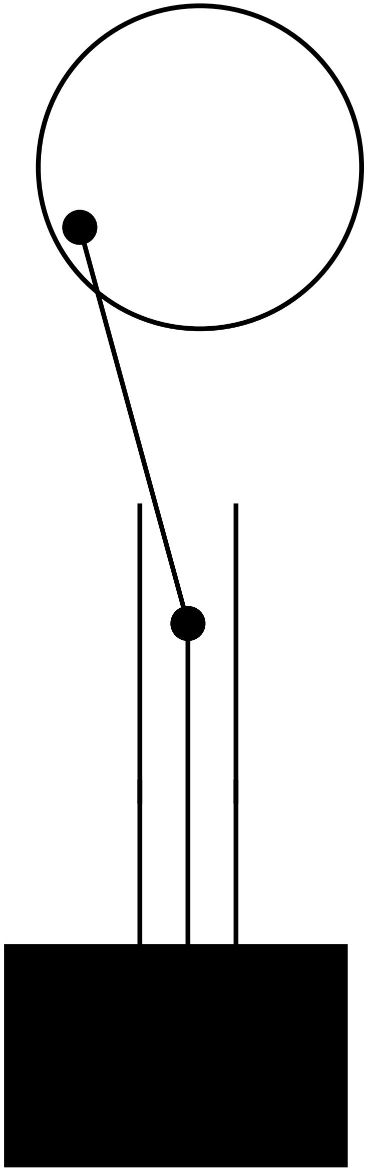
# The Press Works



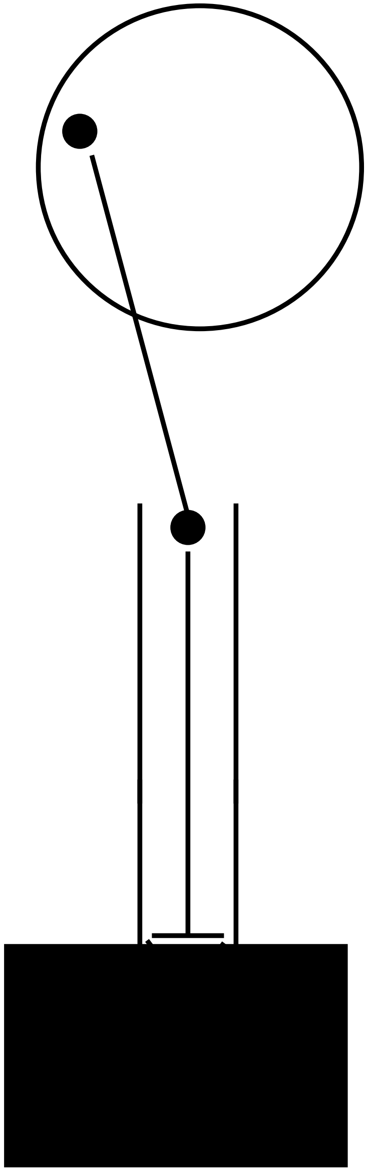
# The Press Works



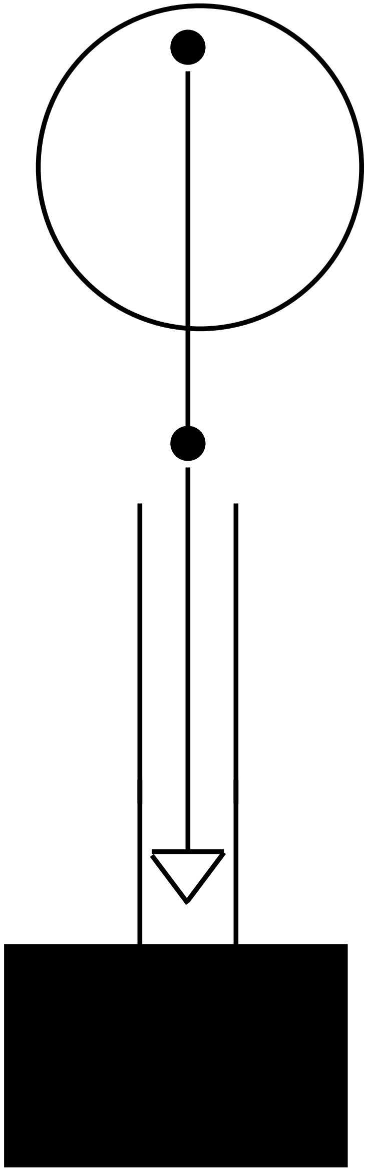
# The Press Works



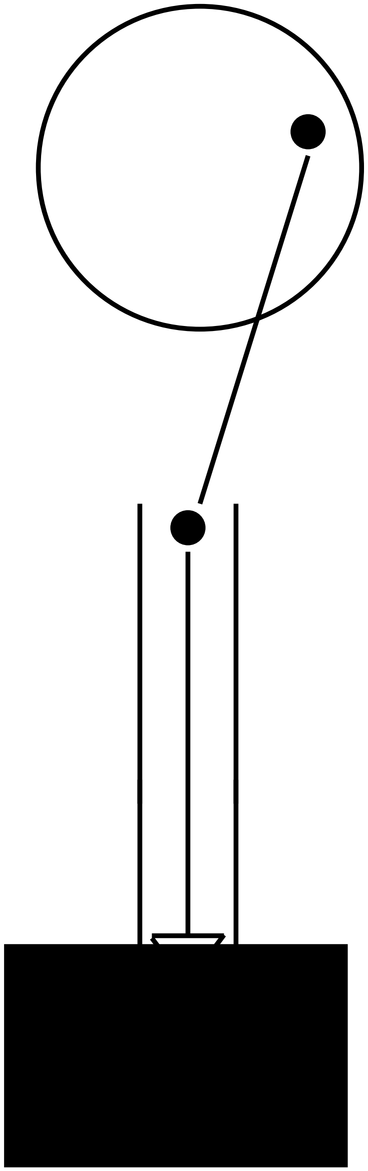
# The Press Works



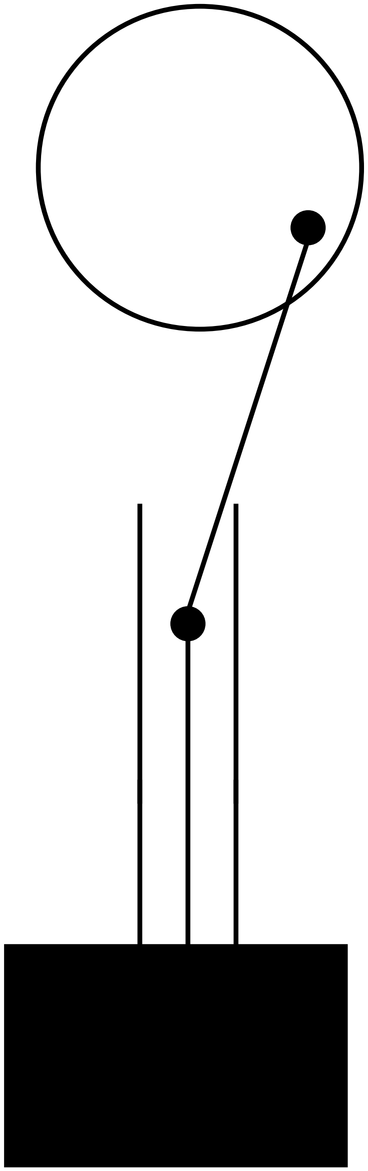
# The Press Works



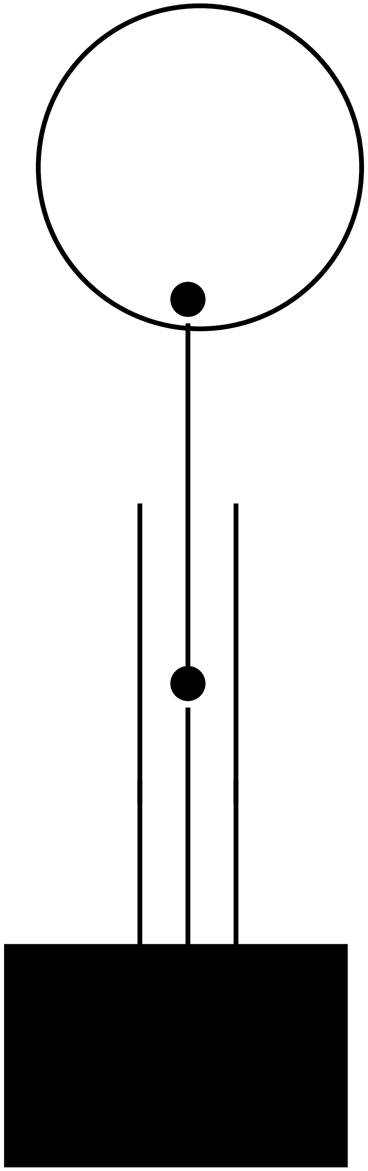
## The Press Works



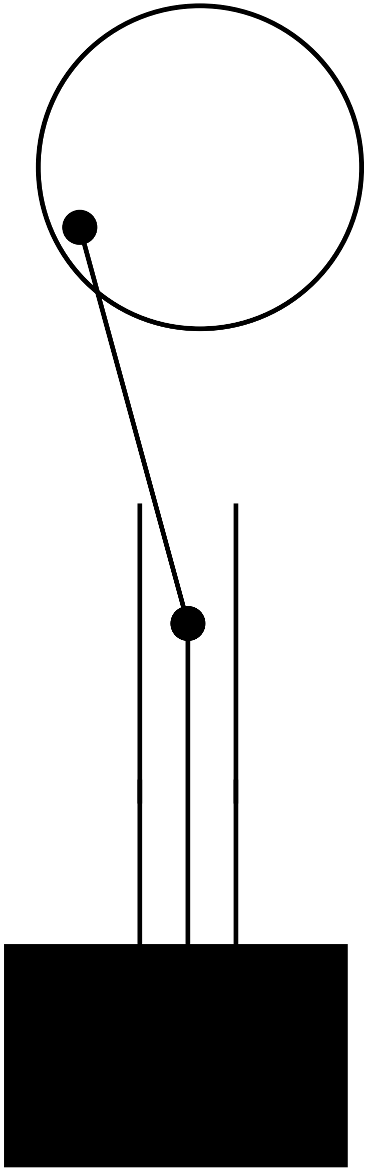
# The Press Works



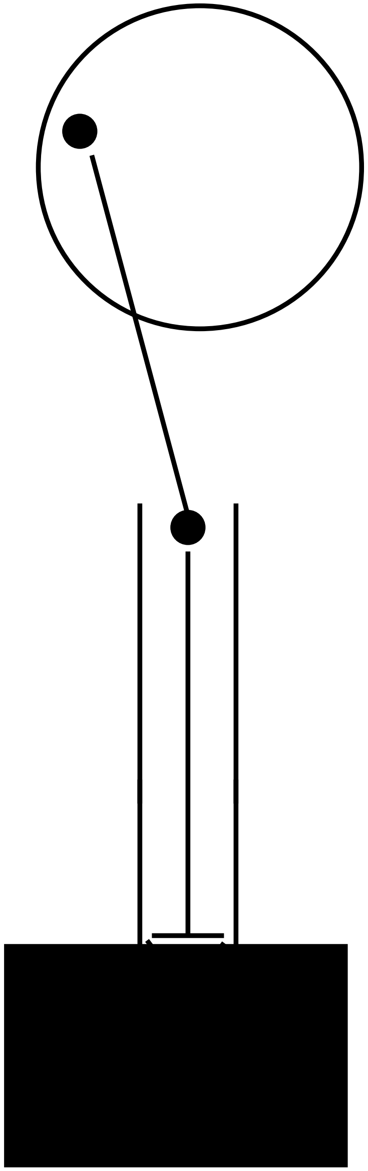
# The Press Works



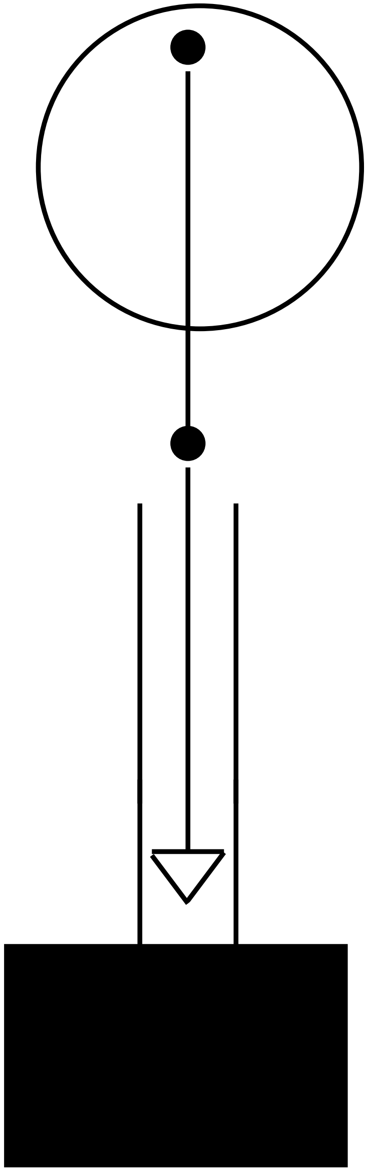
# The Press Works



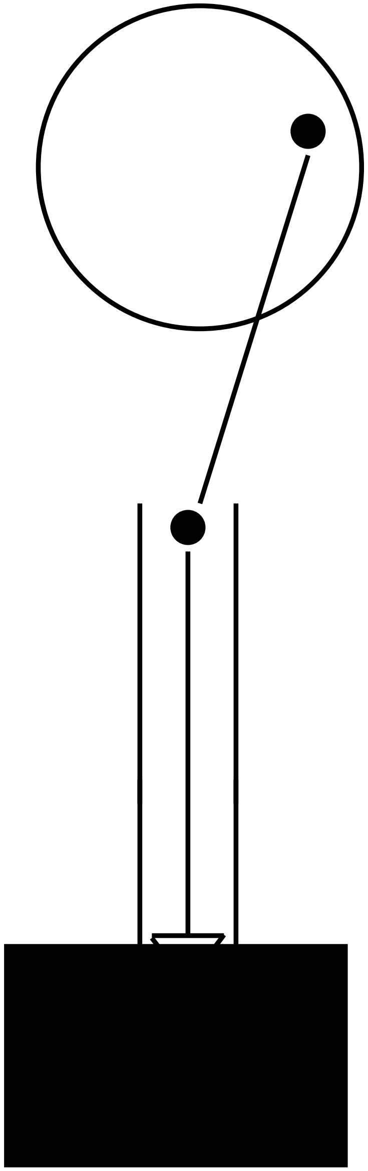
# The Press Works



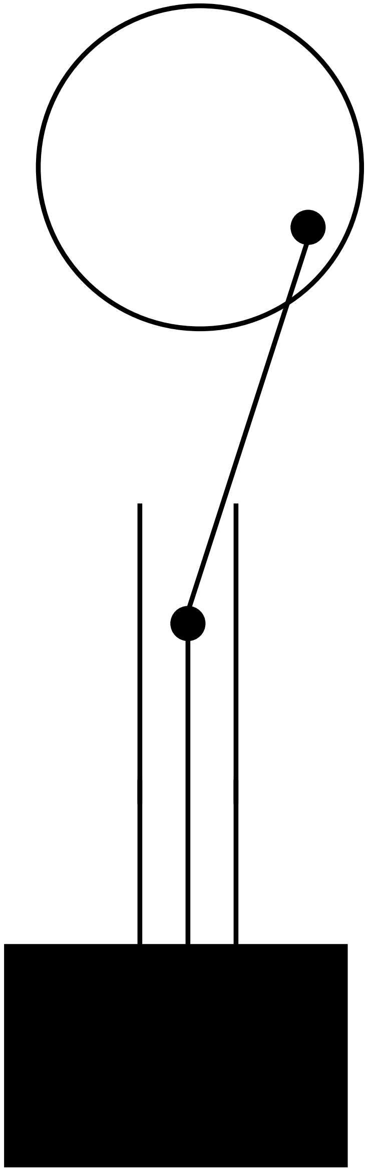
# The Press Works



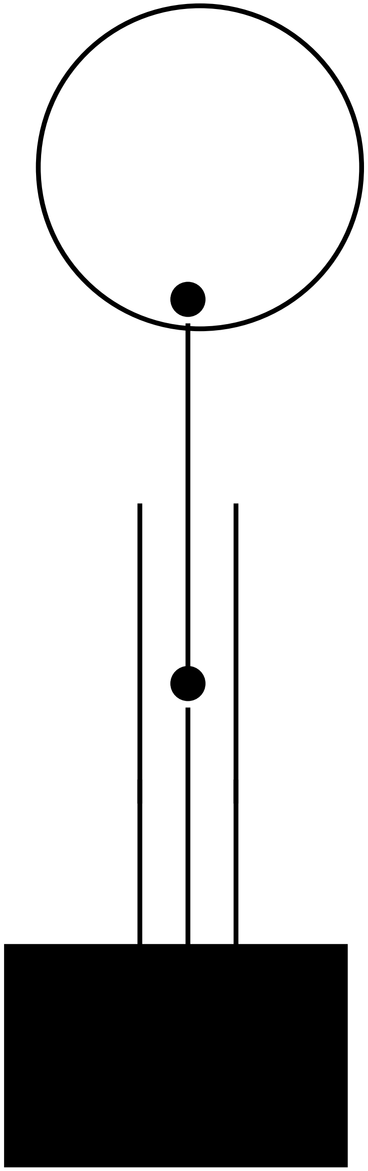
# The Press Works



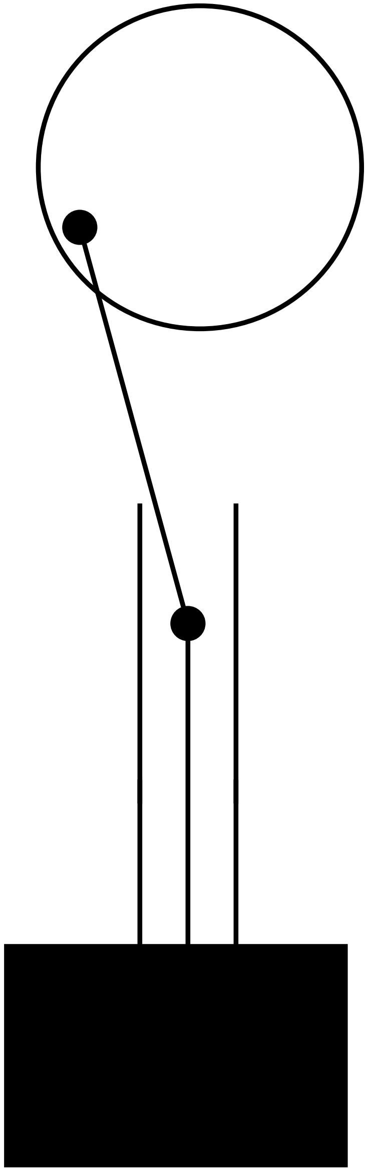
# The Press Works



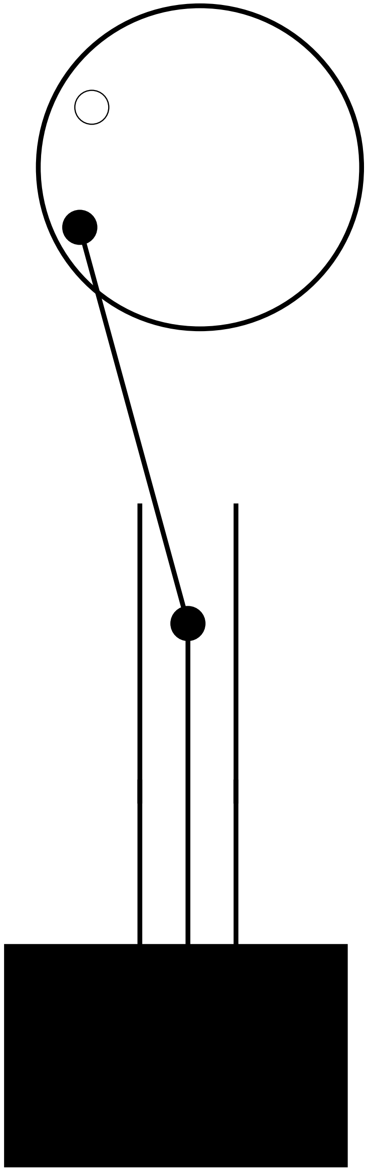
# The Press Works



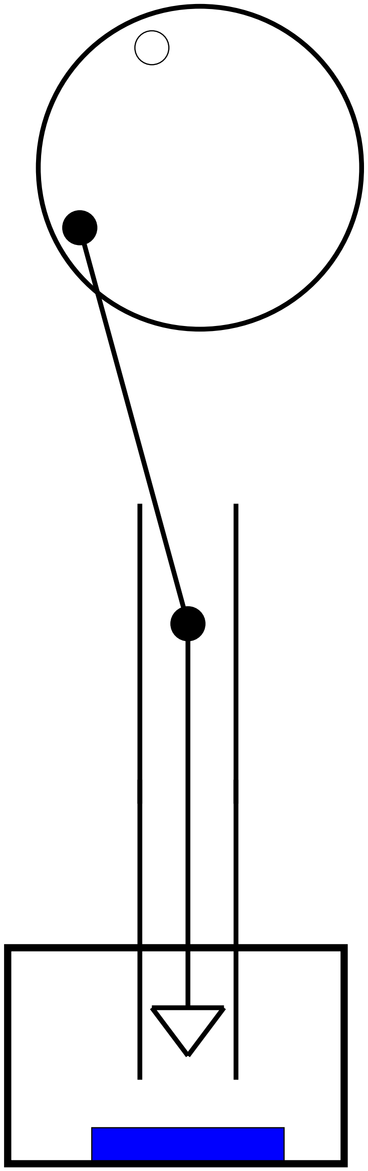
# The Press Works



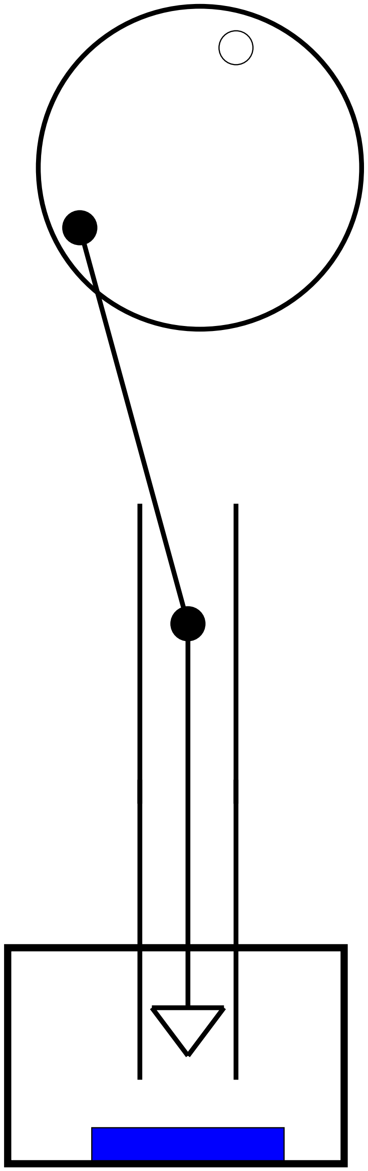
# Disengaging the Clutch: Pressing Button B4



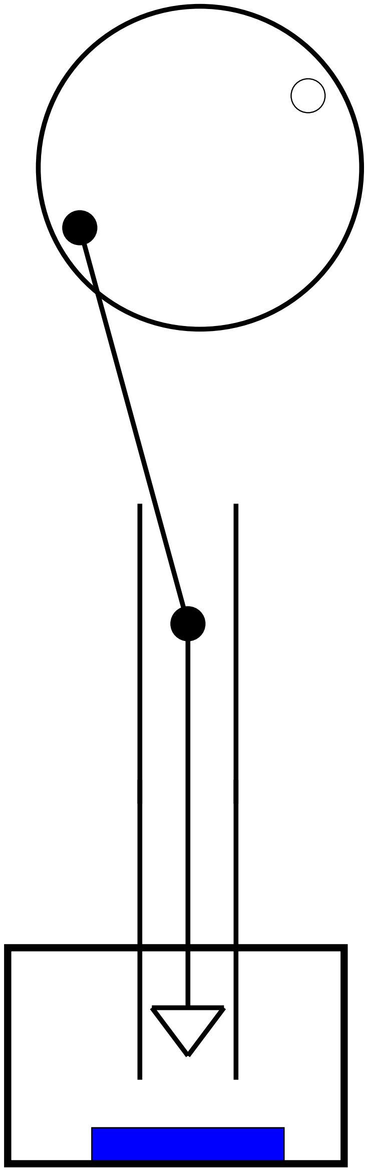
# The Motor Works



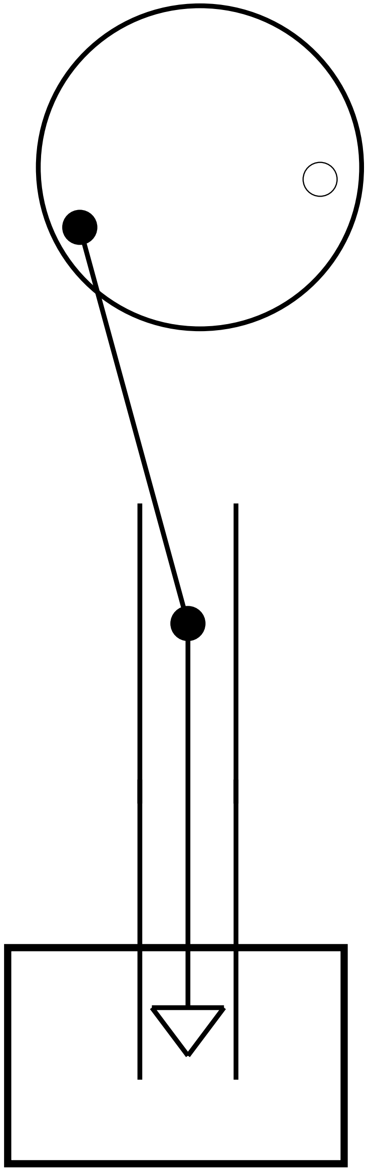
# The Motor Works



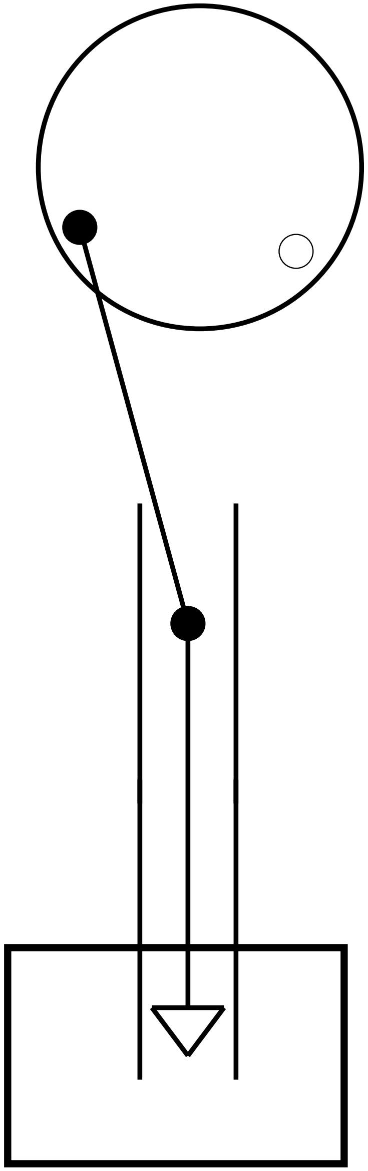
# The Motor Works



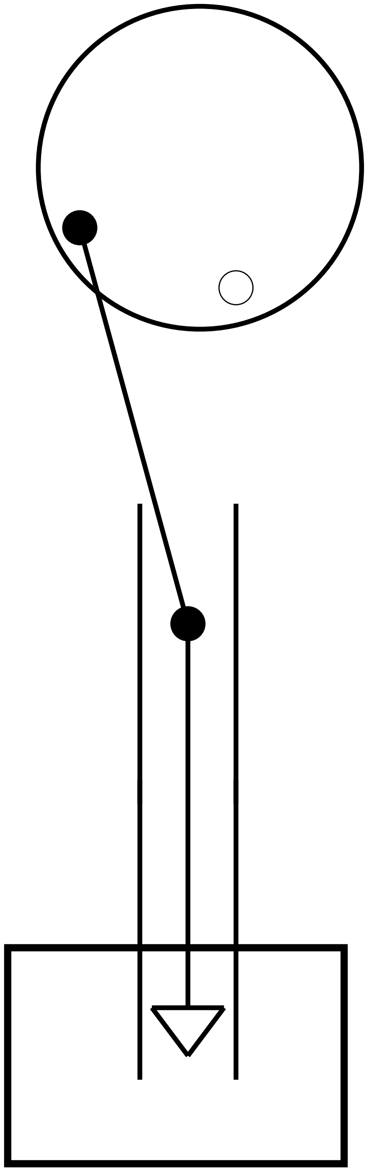
# Removing the Part



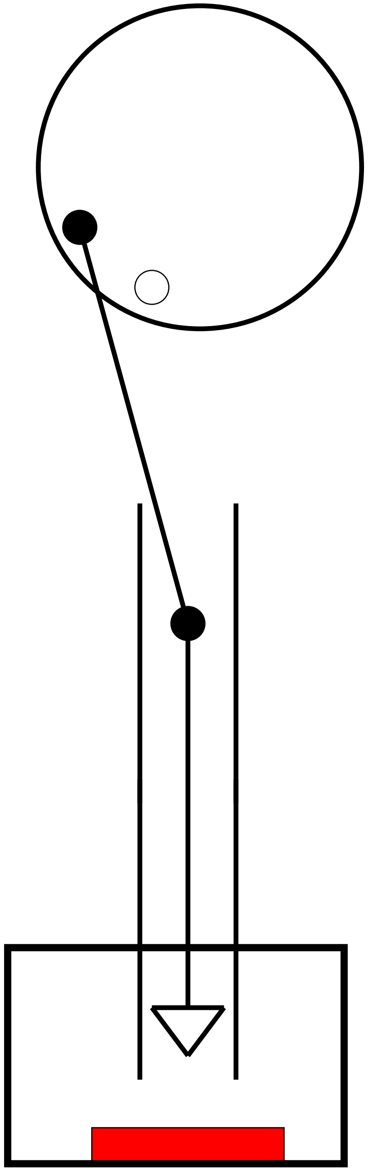
# The Motor Works



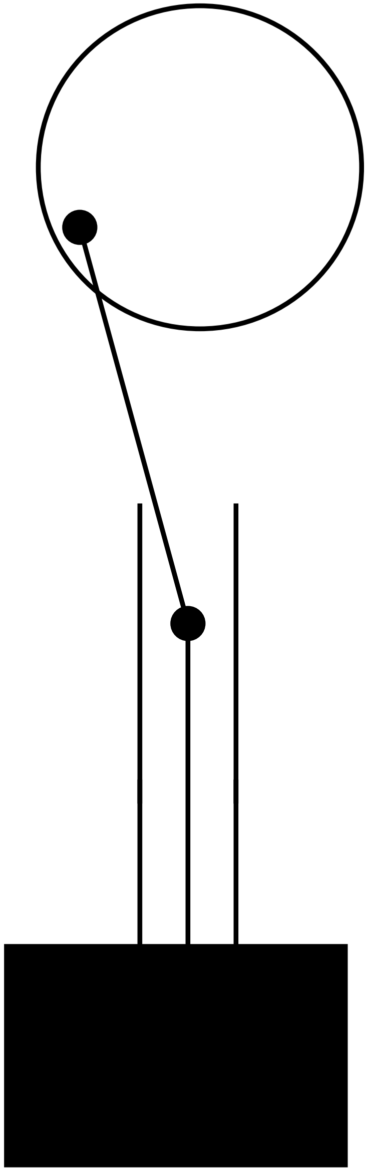
# The Motor Works



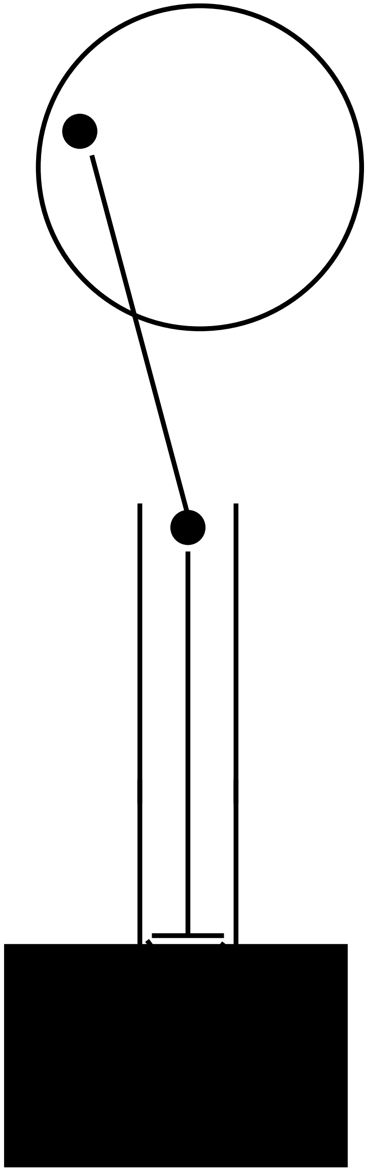
# Putting a Part



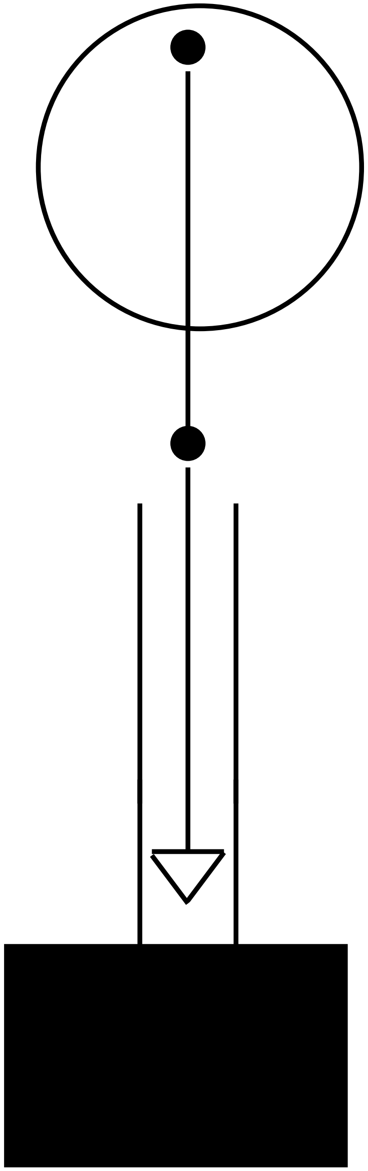
# Engaging the Clutch: Pressing Button B3



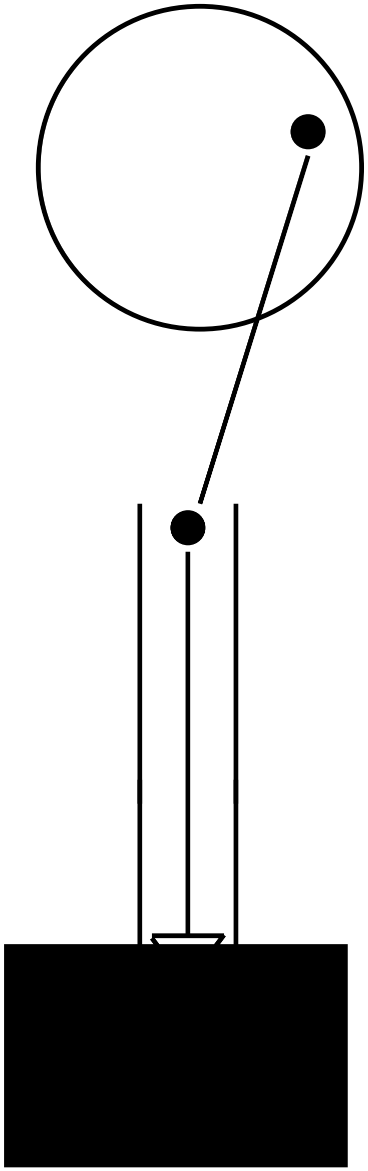
## The Press works



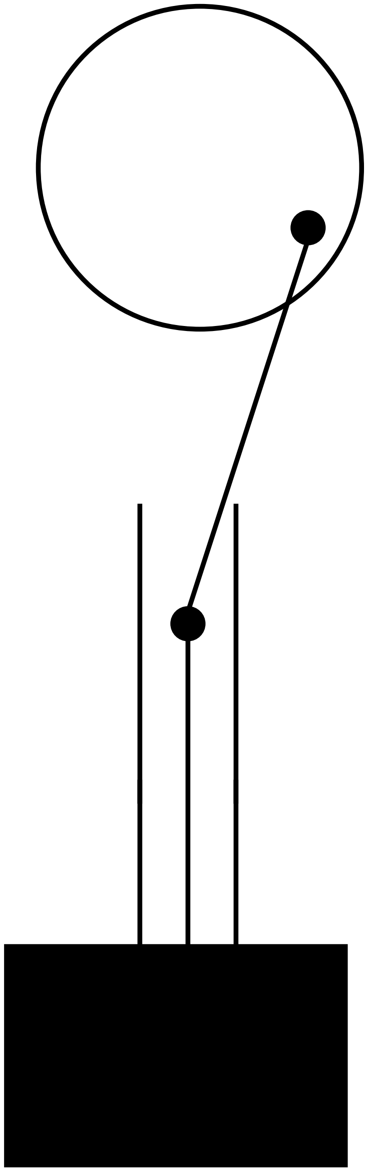
# The Press works



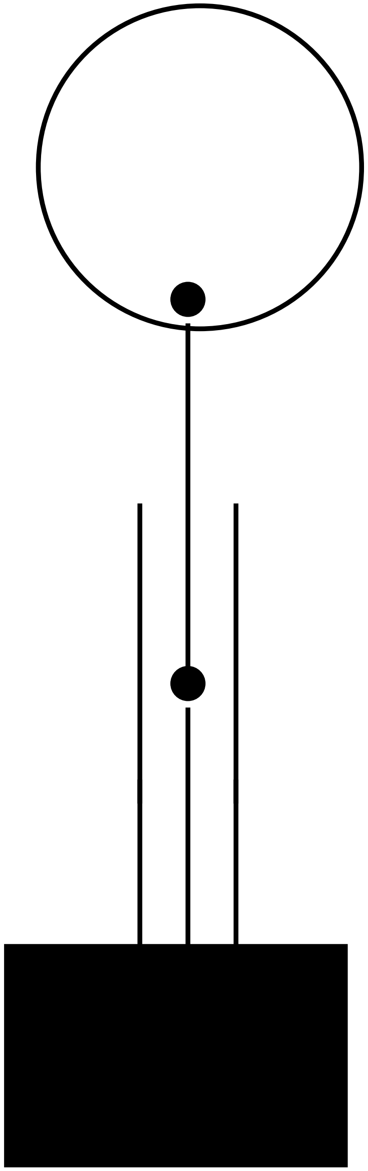
# The Press works



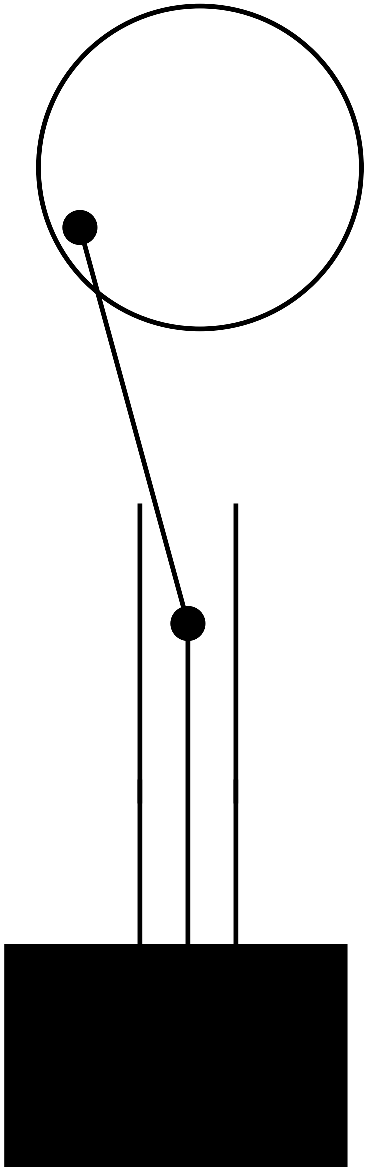
# The Press works



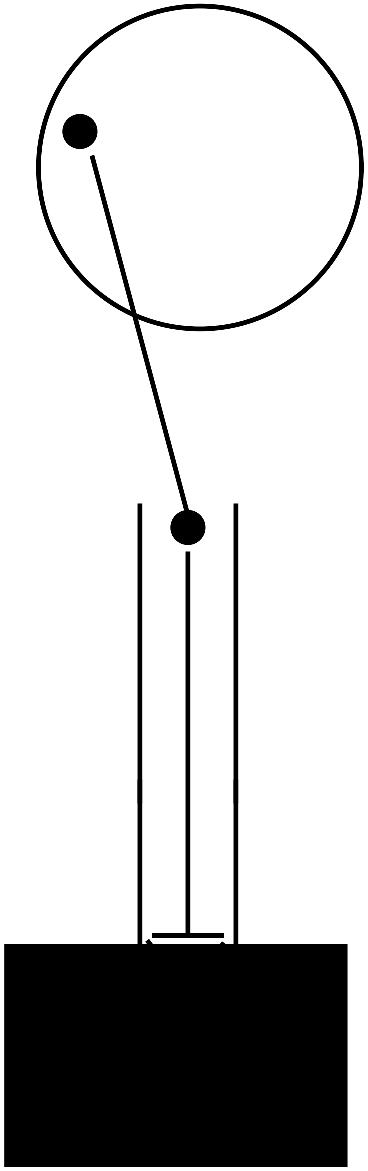
# The Press works



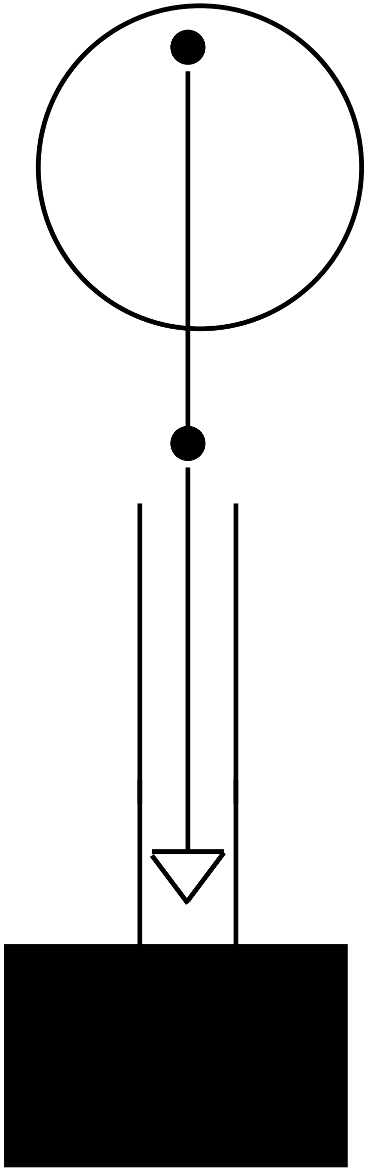
# The Press works



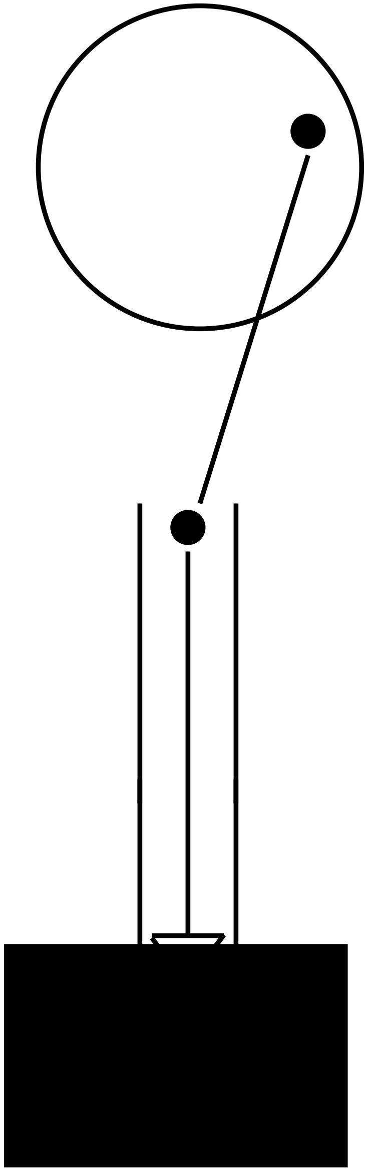
# The Press works



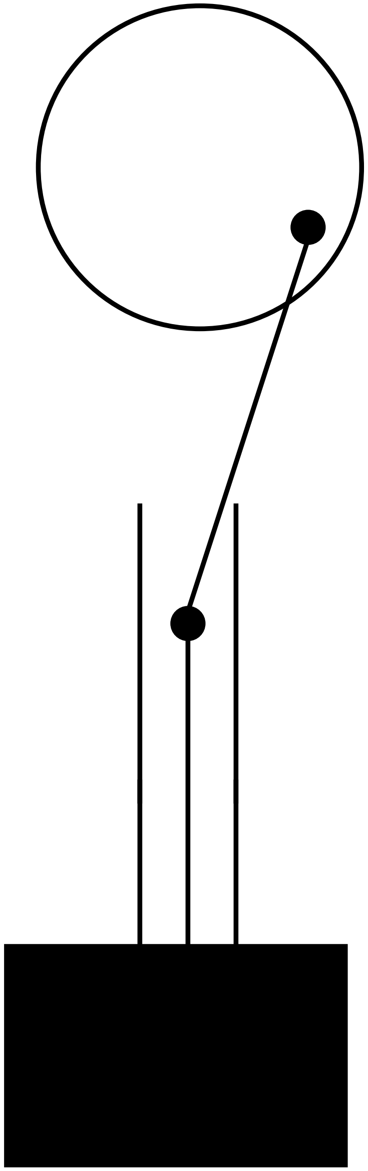
# The Press works



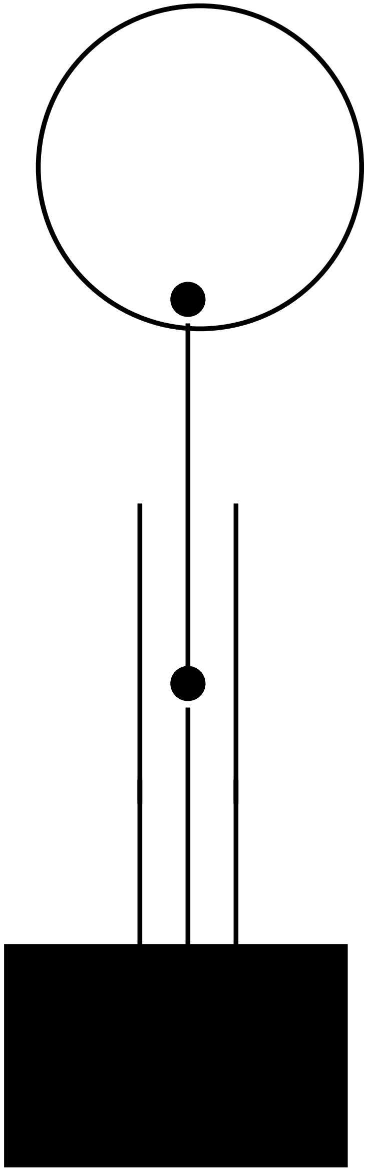
# The Press works



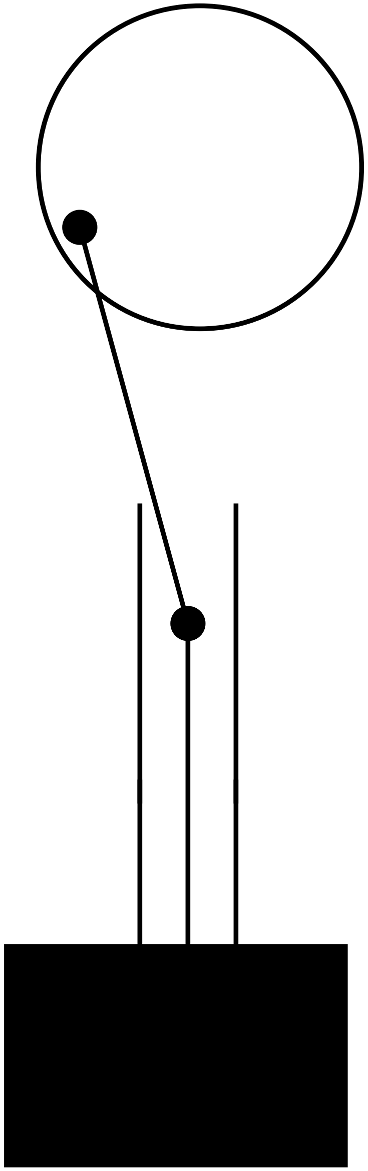
# The Press works



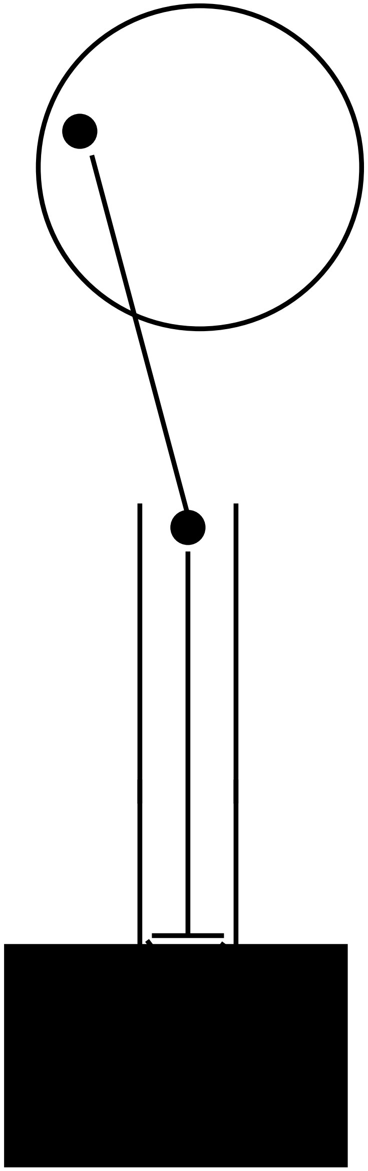
# The Press works



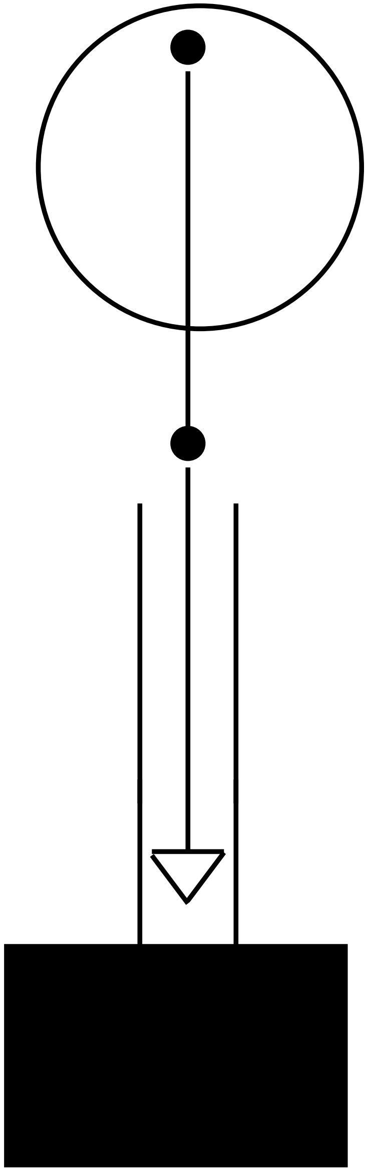
# The Press works



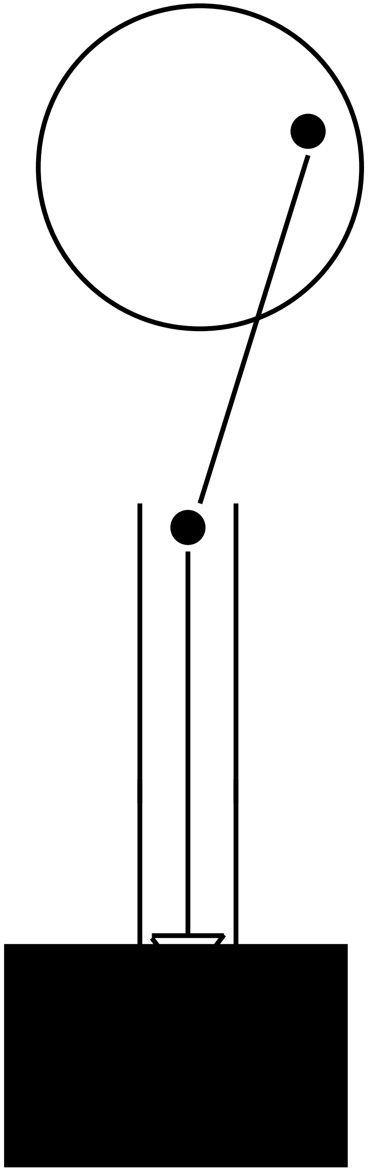
# The Press works



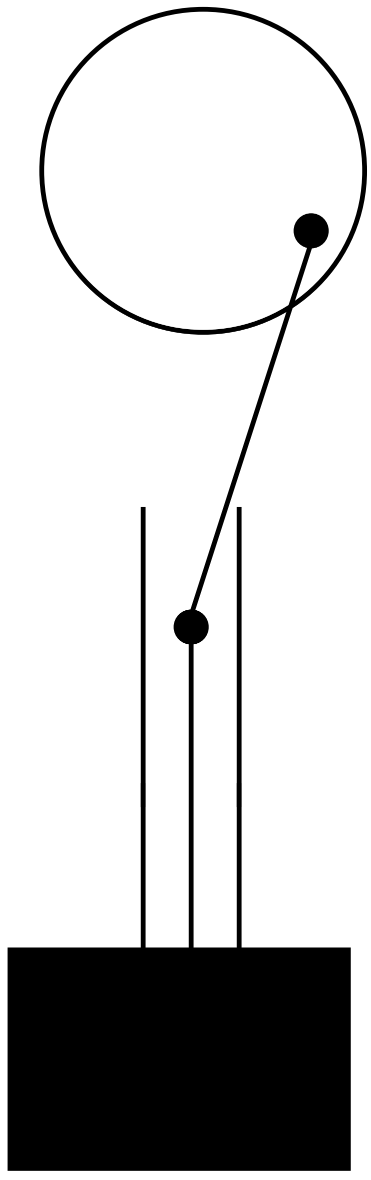
# The Press works



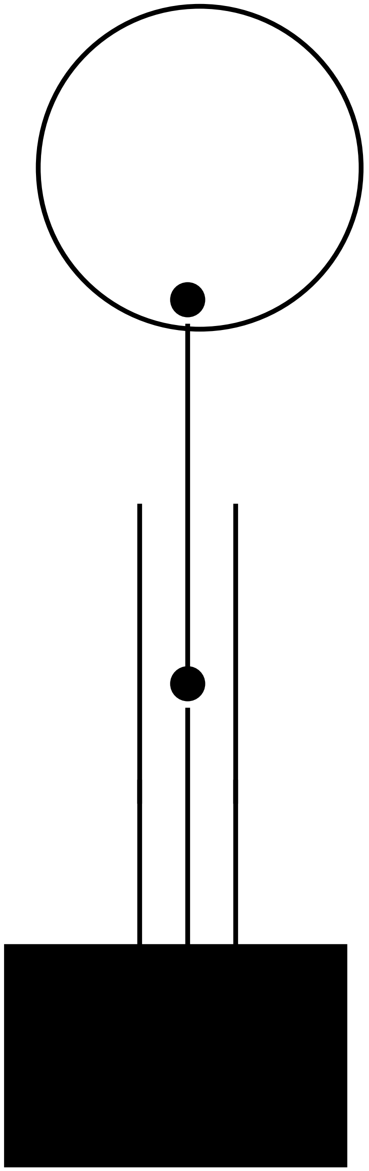
# The Press works



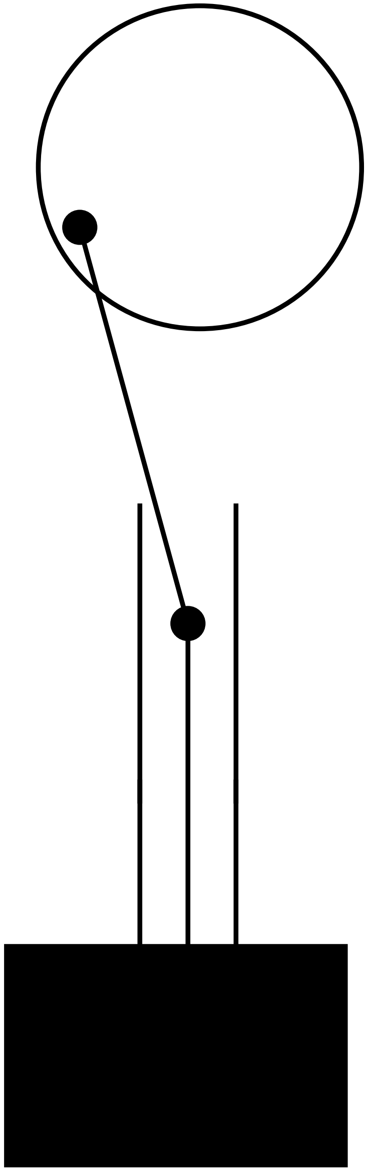
# The Press works



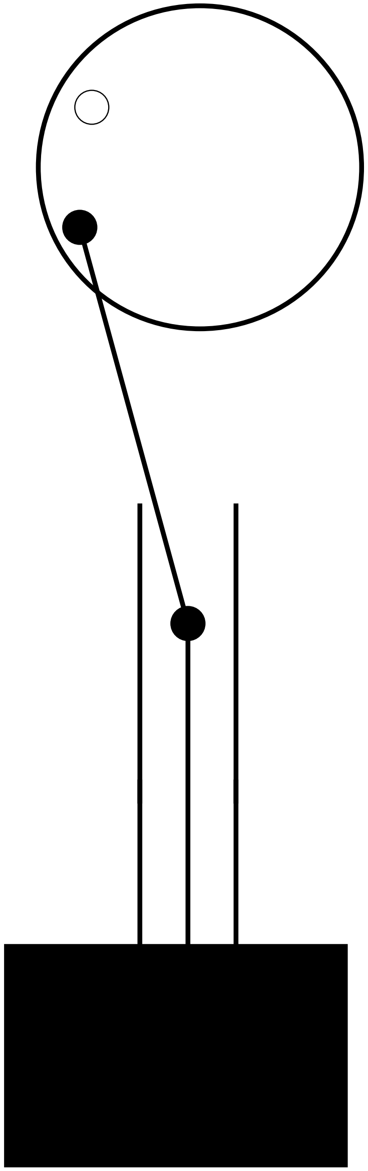
# The Press works



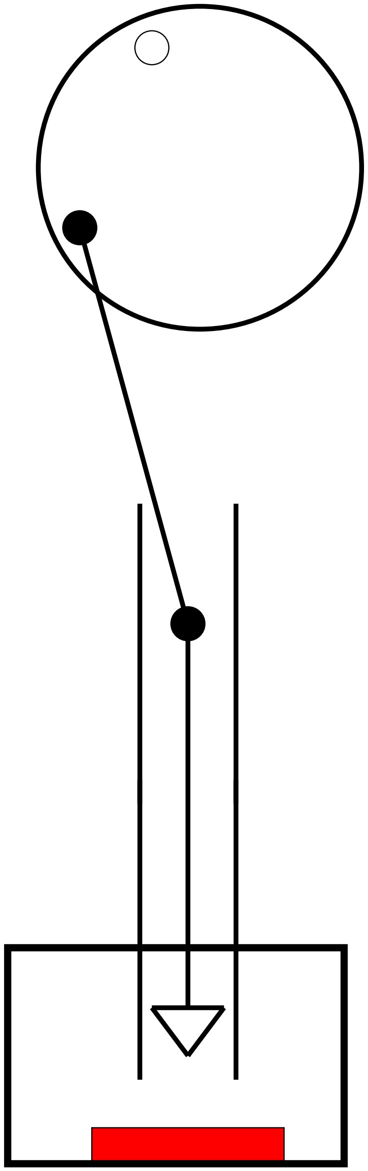
# The Press works



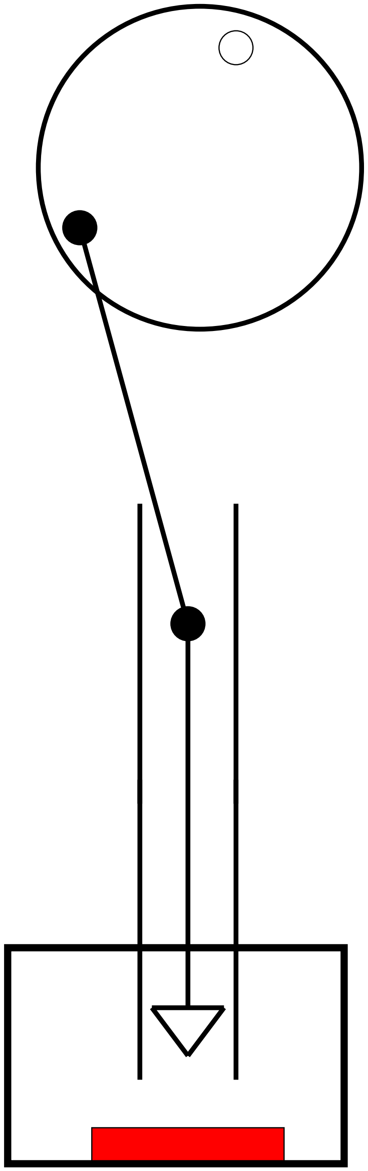
# Disengaging the Clutch: Pressing Button B4



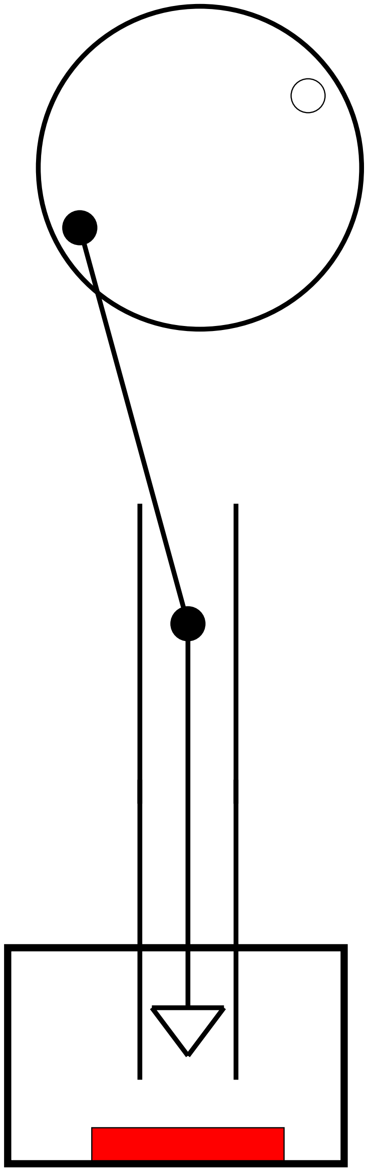
# The Motor Works



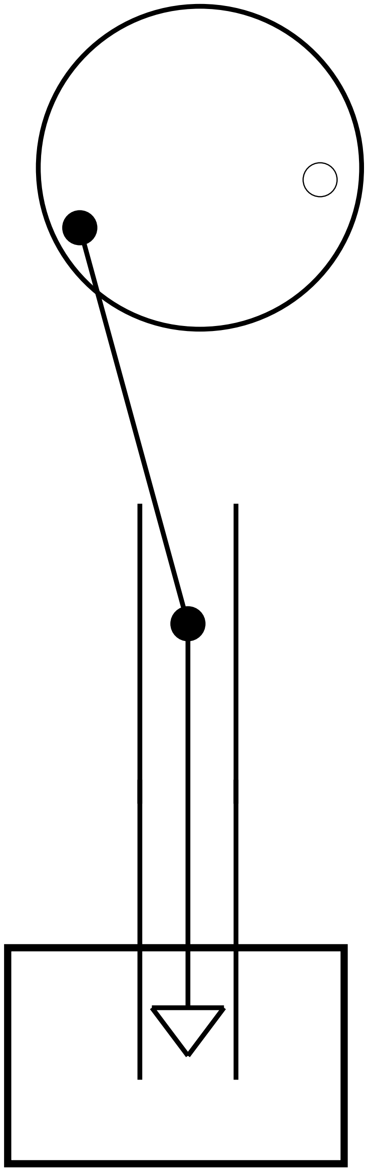
# The Motor Works



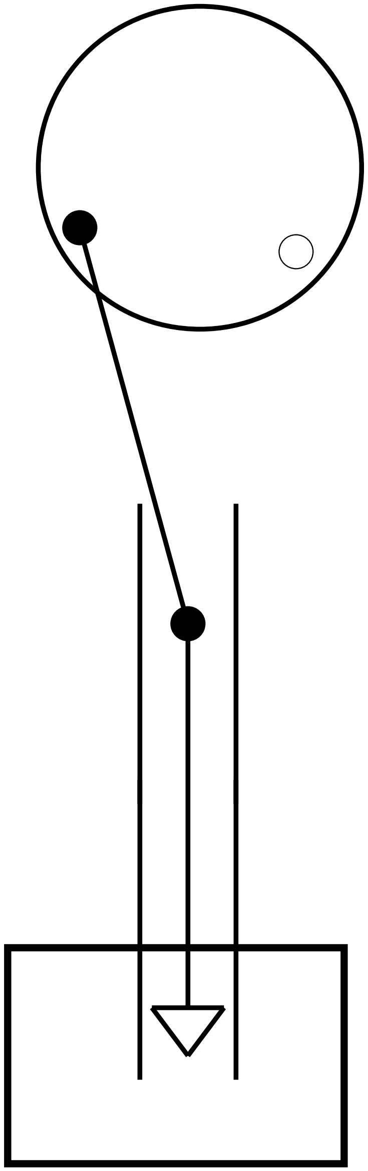
# The Motor Works



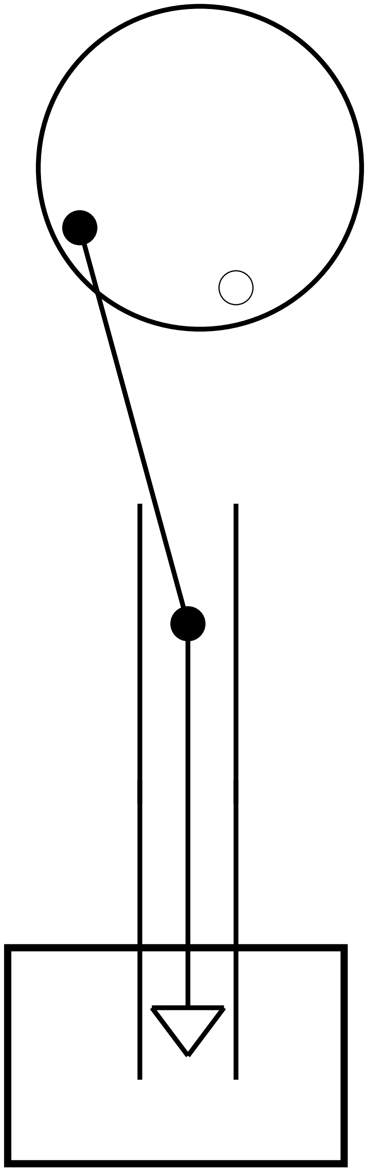
# Removing the Part



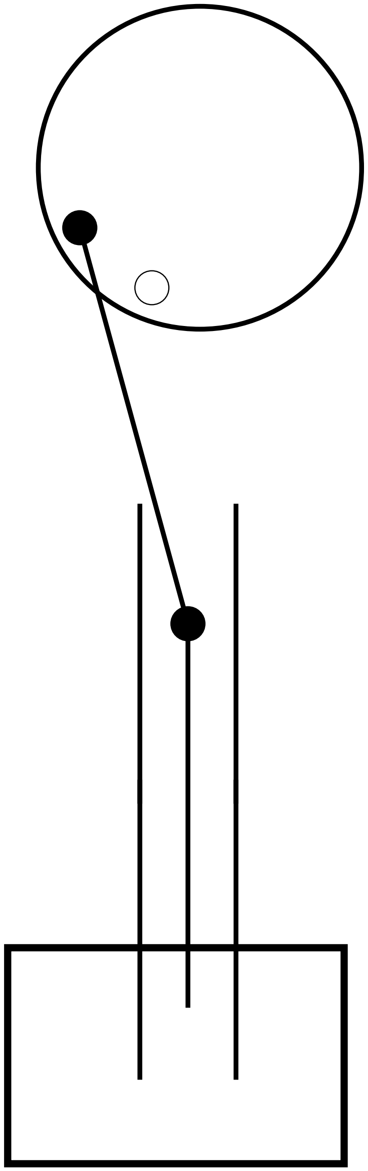
# The Motor Works



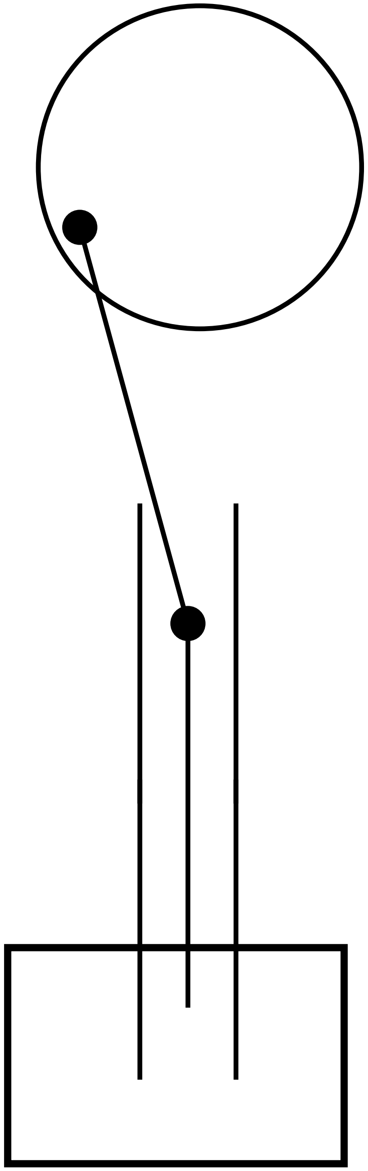
# The Motor Works



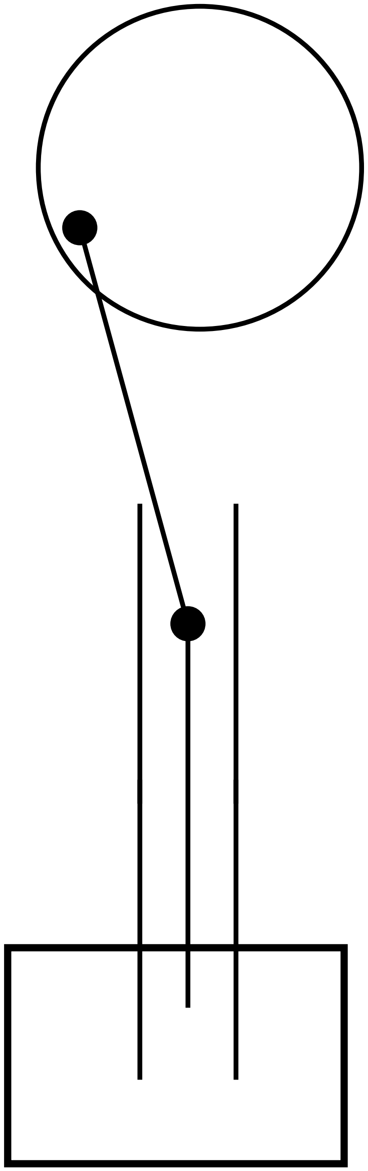
# Removing the Tool



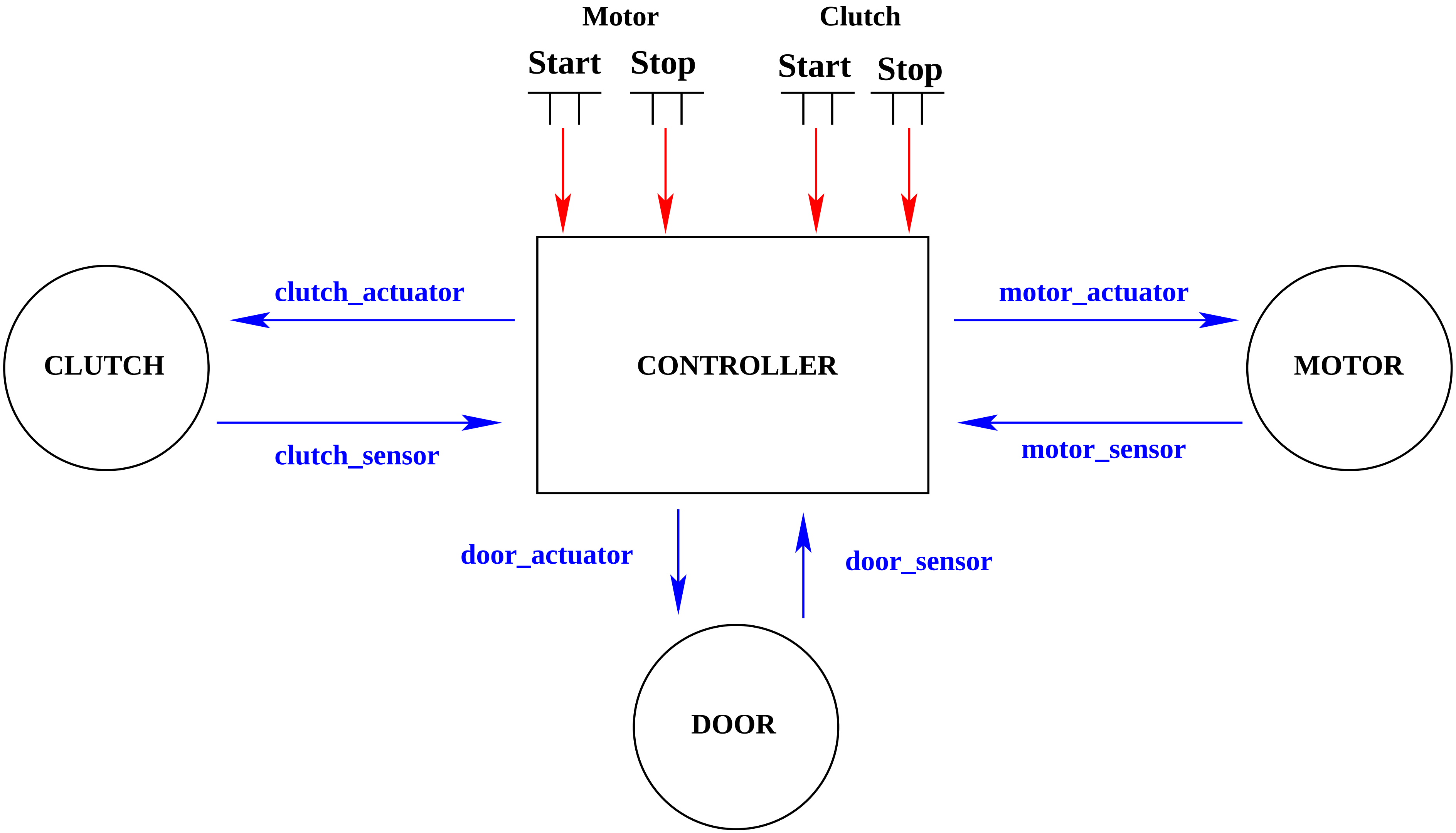
# Stopping Motor: Pressing Button B2



# Final Situation



# Overview


Motor
Clutch
Start
Stop
Start
Stop
clutch_actuator
CLUTCH
clutch_sensor
CONTROLLER
motor_actuator
MOTOR
motor_sensor
door_actuator
door_sensor
DOOR

# 2. Presentation of some Design Patterns

# Motivations

- A number of similar behaviors

- Some complex situations to handle

# Similar Behavior (1)

- A specific action results eventually in having a specific reaction:
    - Pushing button B1 results eventually in starting the motor
    - Pushing button B4 results eventually in disengaging the clutch
    - . . .

# Similar Behavior (2)

- Correlating two pieces of equipment:

  - When the clutch is engaged then the motor must work

  - When the clutch is engaged then the door must be closed

# Similar Behavior (3)

- Making an action dependent of another one:

- Engaging the clutch implies closing the door first

- Disengaging the clutch means opening the door afterwards

# Motivation: Example of Some Complex Situation

- Here is a sequence of events:

(1) User pushes button B1 (start motor)

(1’) User does not remove his finger from button B1

(2) Controller sends the starting command to the motor

(3) Motor starts and sends feedback to the controller

(4) Controller is aware that the motor works

(5) User pushes button B2 (stop motor)

(6) Controller sends the stop command to the motor

(7) Motor stops and sends feedback to the controller

(8) Controller is aware that the motor does not work

(9) Controller must not send the starting command to the motor

# Motivation: Example of Some Complex Situation

- Here is a sequence of events:

(1) User pushes button B1 (start motor)

(2) Controller sends the starting command to the motor

(3.1) Motor starts and sends feedback to the controller

(3.2) User pushes button B2 (stop motor)

- (3.1) and (3.2) may occur simultaneously

- If controller treats (3.1) before (3.2): motor is stopped

- If controller treats (3.2) before (3.1): motor is not stopped

# Design Patterns in Formal Developments

- We want to build systems which are correct by construction

- We want to have more methods for doing so

- "Design pattern" is an Object Oriented concept

- We would like to borrow this concept for doing formal developments

- A preliminary tentative with reactive system developments

- Advantage: systematic developments and also refinement of proofs

# A Personal View on Design Patterns

- This is an engineering concept

- It can be used outside OO

- The goal of each DP is to solve a certain category of problems

- But the design pattern has to be adapted to the problem at hand

- Is it compatible with formal developments?

- Let’s apply this approach to the design of reactive systems

# A Wikipedia View on Design Patterns (1)

- A design pattern isn’t a finished design that can be transformed into code

- It is a template for how to solve a problem that can be used in many different situations

- Patterns originated as an architectural concept by Christopher Alexander

- "Design Patterns: Elements of Reusable Object-Oriented Software" published in 1994 (Gamma et al)

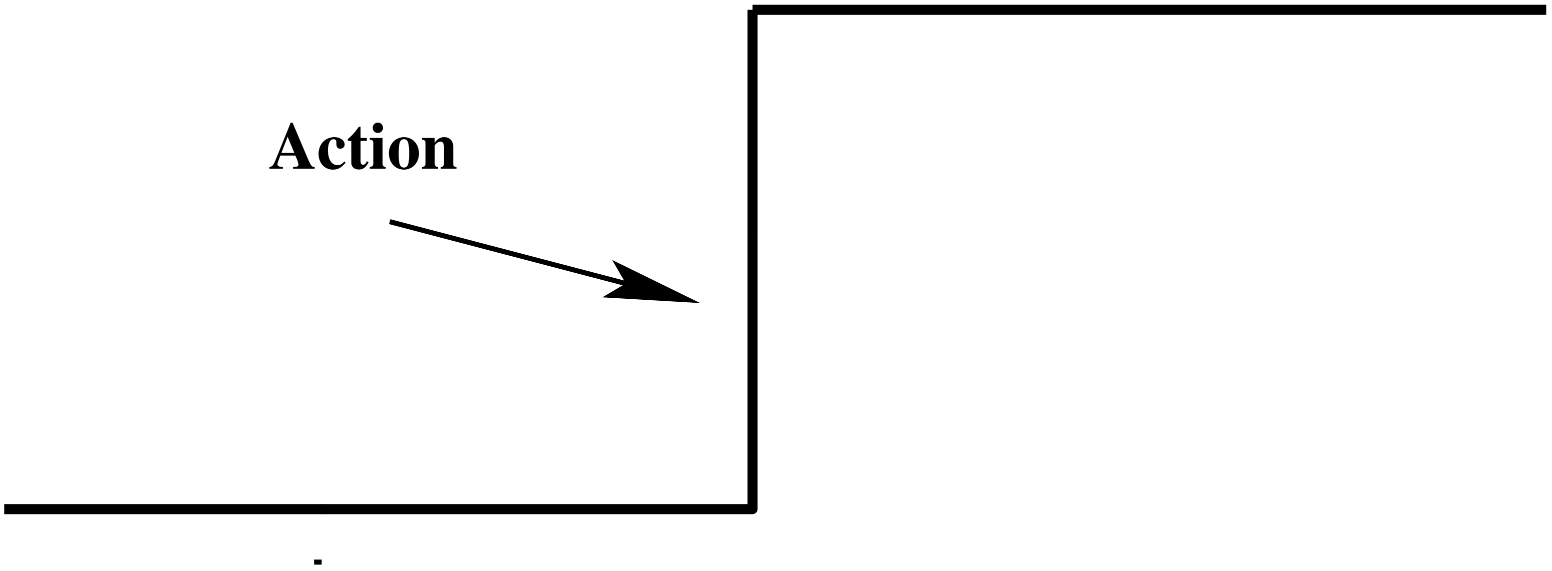
# A Wikipedia View on Design Patterns (2)

- Design pattern can speed up the development process by providing tested and proven development paradigms

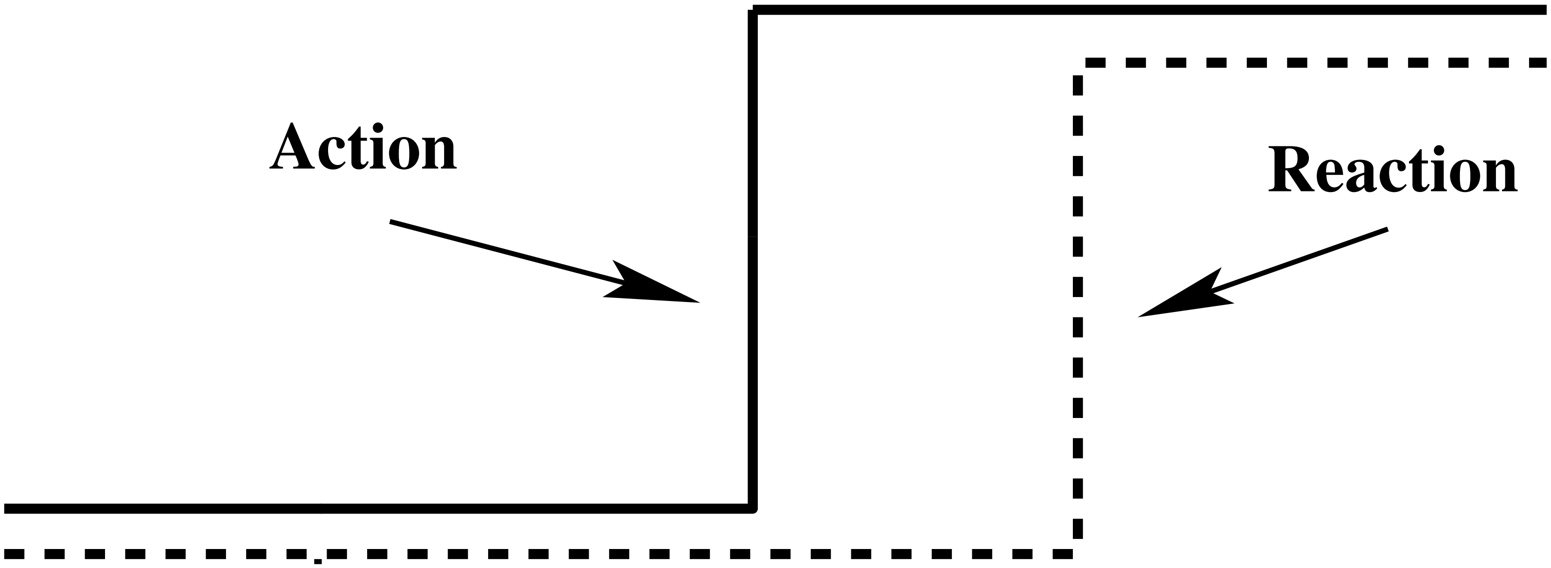
- The documentation for a design pattern should contain enough information about the problem that the pattern addresses, the context in which it is used, and the suggested solution.

- Some feel that the need for patterns results from using computer languages or techniques with insufficient abstraction

# An Action Pattern



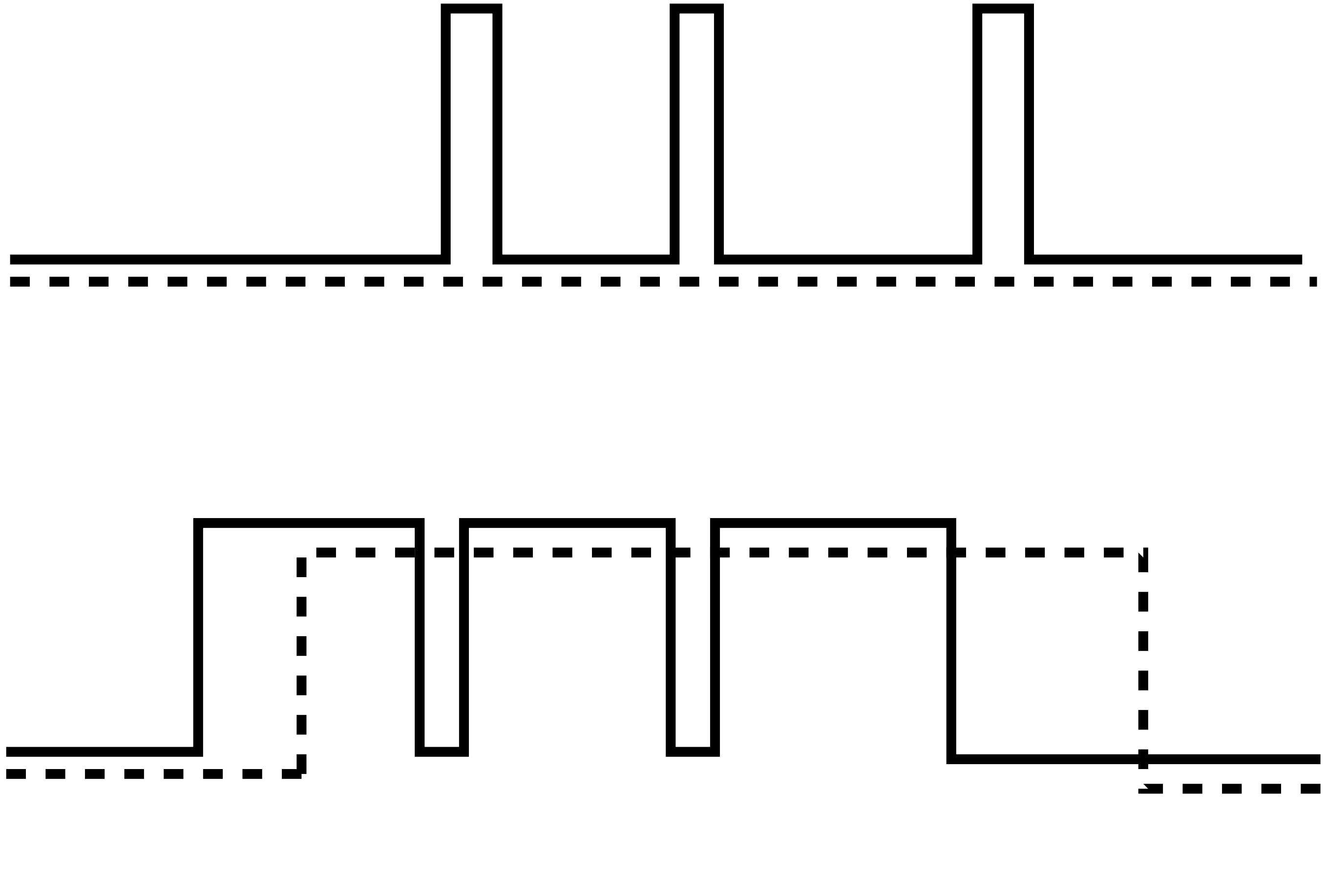
# Action and Reaction Patterns



# Action and Weak Reaction Patterns

- Sometimes, the reaction has not enough time to react

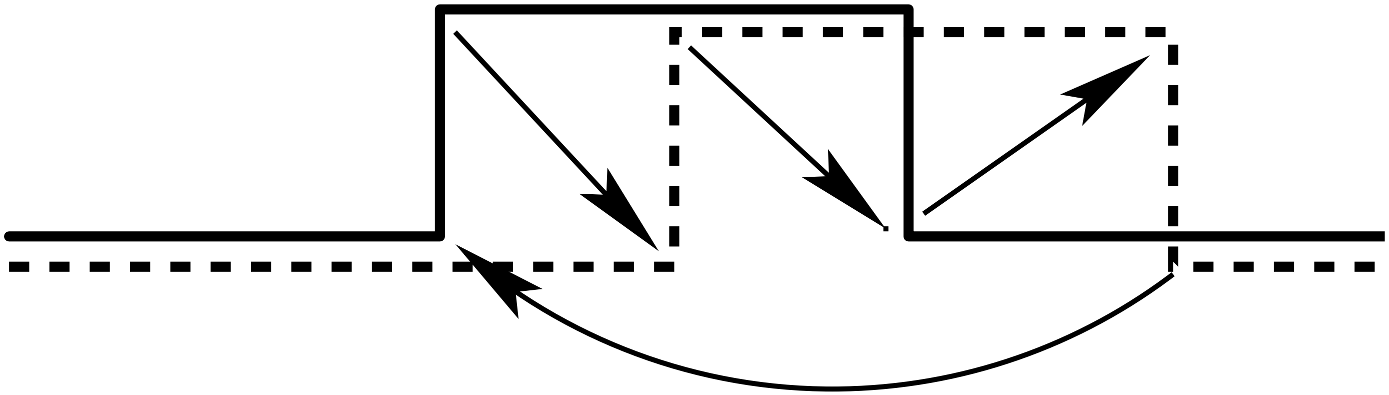
- Because the action moves too quickly



# Action and Strong Reaction Patterns

- Sometimes, the reaction always follows the action

- They are both synchronized



# Constructing Models of Actions and Reactions

- We built first a model of a weak reaction

- The strong reaction will be a refinement of the weak one

# Model for weak action and reaction: the State

**variables:** $a$
$r$

**pat0_1:** $a \in \{0,1\}$

**pat0_2:** $r \in \{0,1\}$

- $a$ denotes the action

- $r$ denotes the reaction

# Model for weak action and reaction: the State

**variables:** $a$ $r$ $ca$ $cr$

**pat0_1:** $a \in \{0,1\}$

**pat0_2:** $r \in \{0,1\}$

**pat0_3:** $ca \in \mathbb{N}$

**pat0_4:** $cr \in \mathbb{N}$

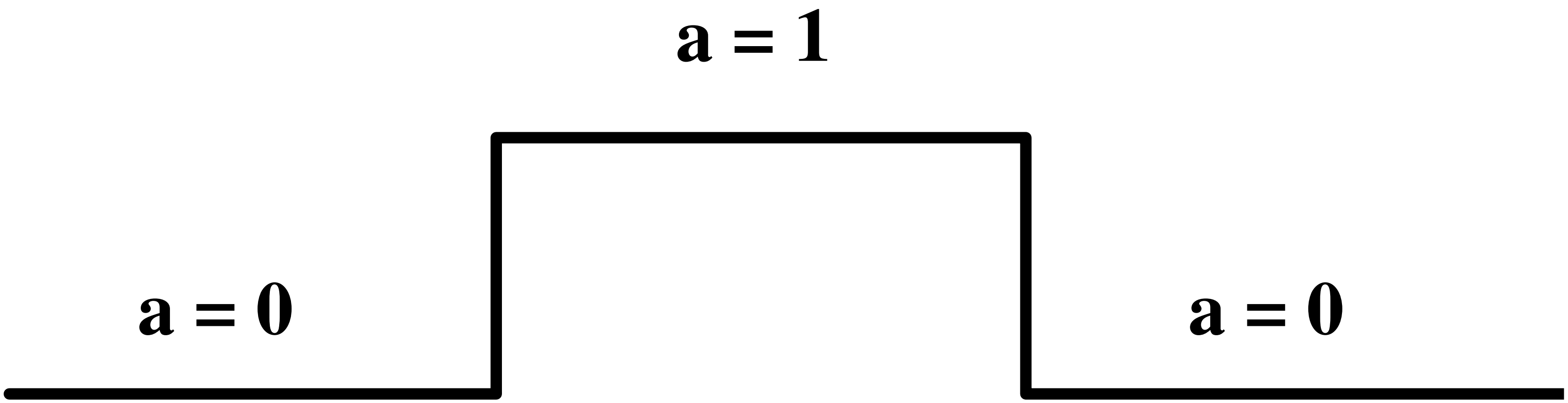
**pat0_5:** $cr \leq ca$

- $ca$ and $cr$ denote how many times $a$ and $r$ are set to 1
- **pat0_5** formalizes the weak reaction

# Model for weak action and reaction: the Events (1)

```
a_on
   when
      $a = 0$
   then
      $a := 1$
      $ca := ca + 1$
   end
```

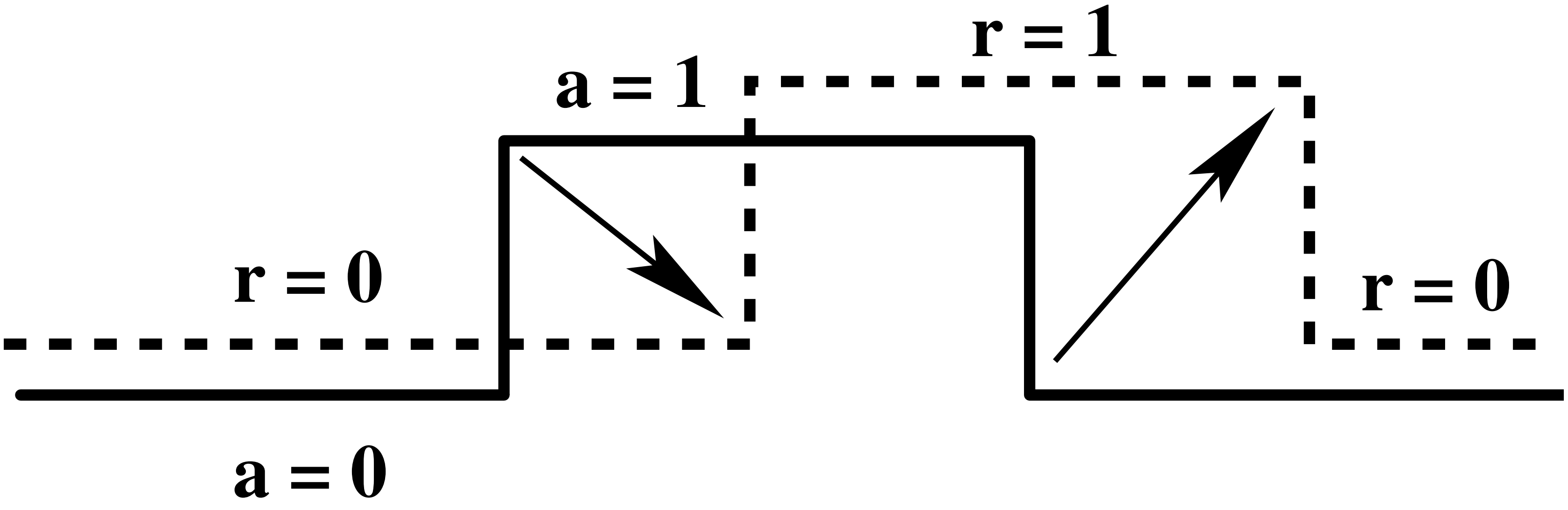
```
a_off
   when
      $a = 1$
   then
      $a := 0$
   end
```



# Model for weak action and reaction: the Events (2)

```
r_on
  when
    $r = 0$
    $a = 1$
  then
    $r := 1$
    $cr := cr + 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end
```



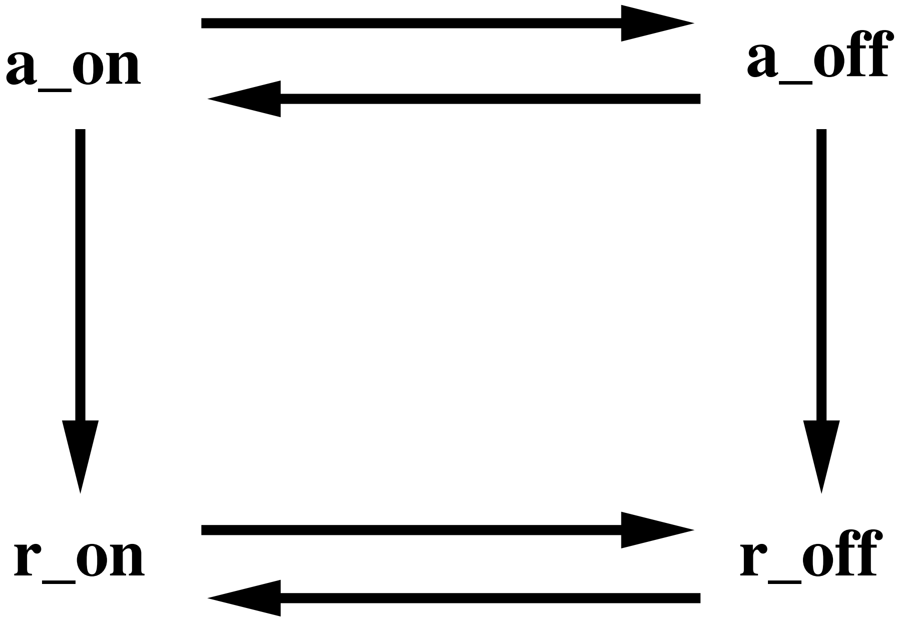
# Summary of Events

```
a_on
  when
    $a = 0$
  then
    $a := 1$
    $ca := ca + 1$
  end
```

```
a_off
  when
    $a = 1$
  then
    $a := 0$
  end
```

```
r_on
  when
    $r = 0$
    $a = 1$
  then
    $r := 1$
    $cr := cr + 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end
```



# Summary of Weak Synchronization

**variables:** $a$, $r$, $ca$, $cr$

**pat0_1:** $a \in \{0,1\}$

**pat0_2:** $r \in \{0,1\}$

**pat0_3:** $ca \in \mathbb{N}$

**pat0_4:** $cr \in \mathbb{N}$

**pat0_5:** $cr \leq ca$

```
init
    $a := 0$
    $r := 0$
    $ca := 0$
    $cr := 0$
```

```
a_on
  when
    $a = 0$
  then
    $a := 1$
    $ca := ca + 1$
  end
```

```
a_off
  when
    $a = 1$
  then
    $a := 0$
  end
```

```
r_on
  when
    $r = 0$
    $a = 1$
  then
    $r := 1$
    $cr := cr + 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end
```

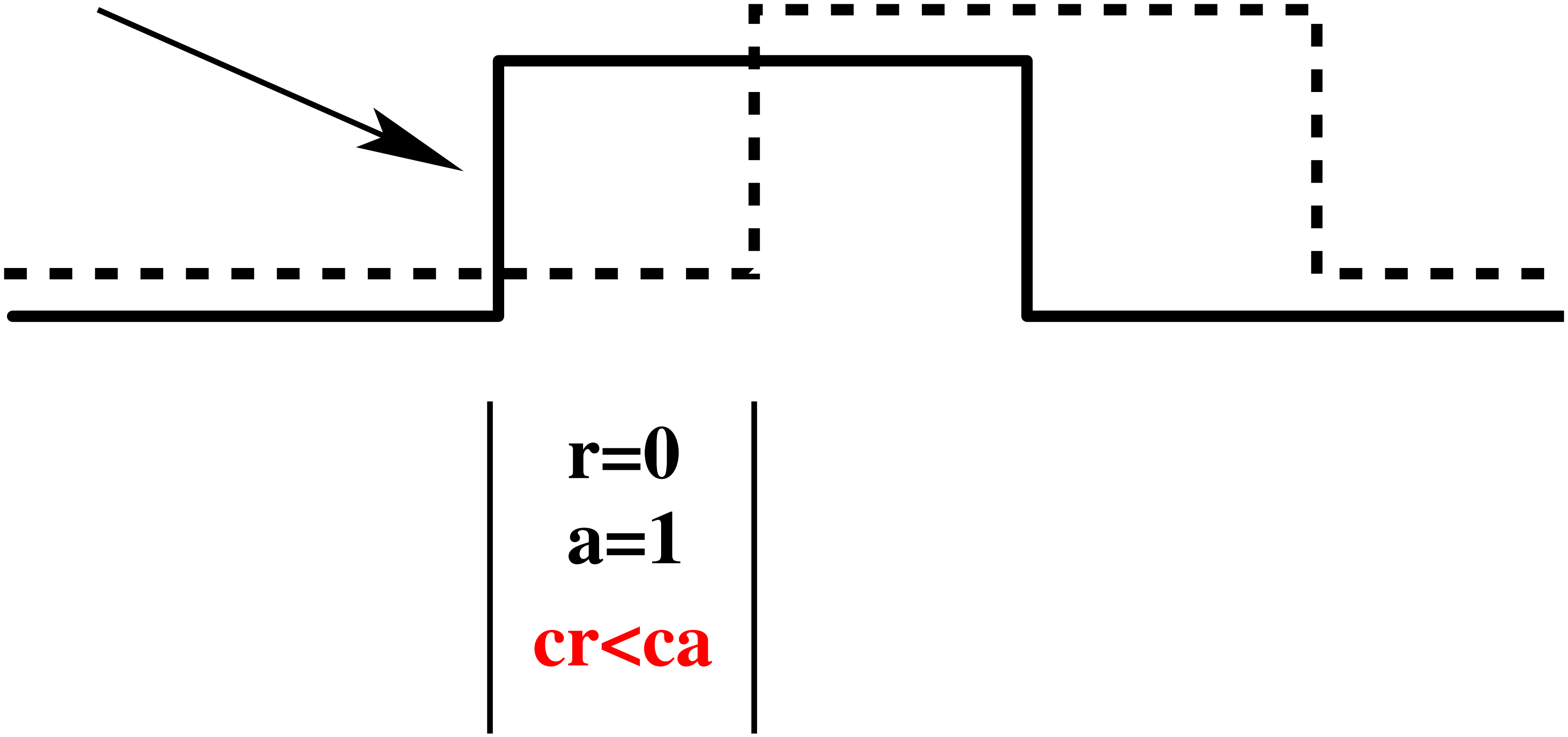
Nothing guarantees that the invariants are preserved

# D E M 0 (Showing a Problem and Finding a Solution)

# Intuition Behind the new Invariant

**pat0_6:** $r = 0 \wedge a = 1 \Rightarrow cr < ca$

ca is incremented



r=0

a=1

cr<ca

# Summary of the State of the weak Reaction

**pat0_1:** $a \in \{0, 1\}$

**pat0_2:** $r \in \{0, 1\}$

**pat0_3:** $ca \in \mathbb{N}$

**pat0_4:** $cr \in \mathbb{N}$

**pat0_5:** $cr \leq ca$

**pat0_6:** $r = 0 \;\wedge\; a = 1 \;\Rightarrow\; cr < ca$

# Summary of the Events of the weak Reaction

The counters have

been removed

```
a_on
  when
    $a = 0$
  then
    $a := 1$
  end
```

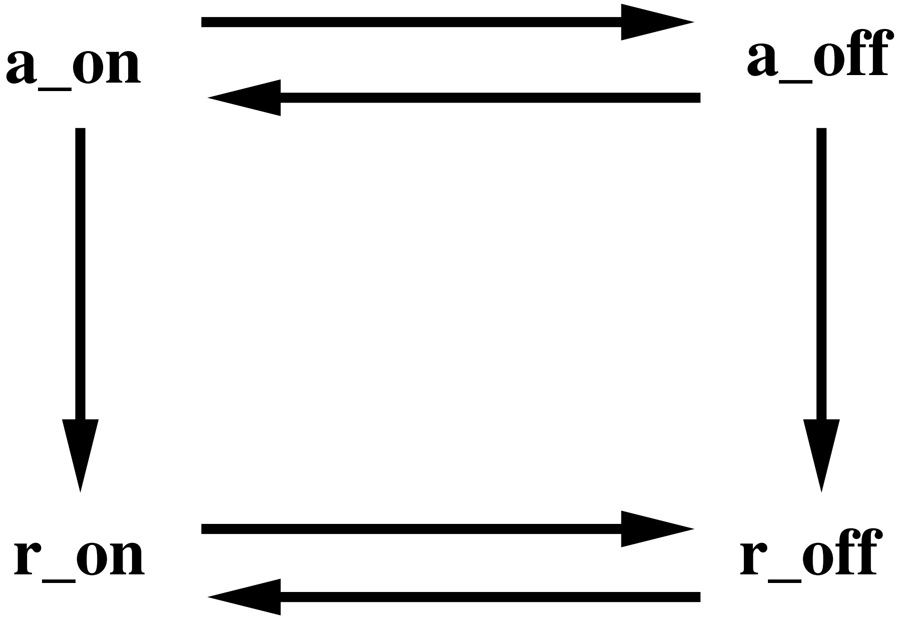
```
a_off
  when
    $a = 1$
  then
    $a := 0$
  end
```

```
init
  $a := 0$
  $r := 0$
```

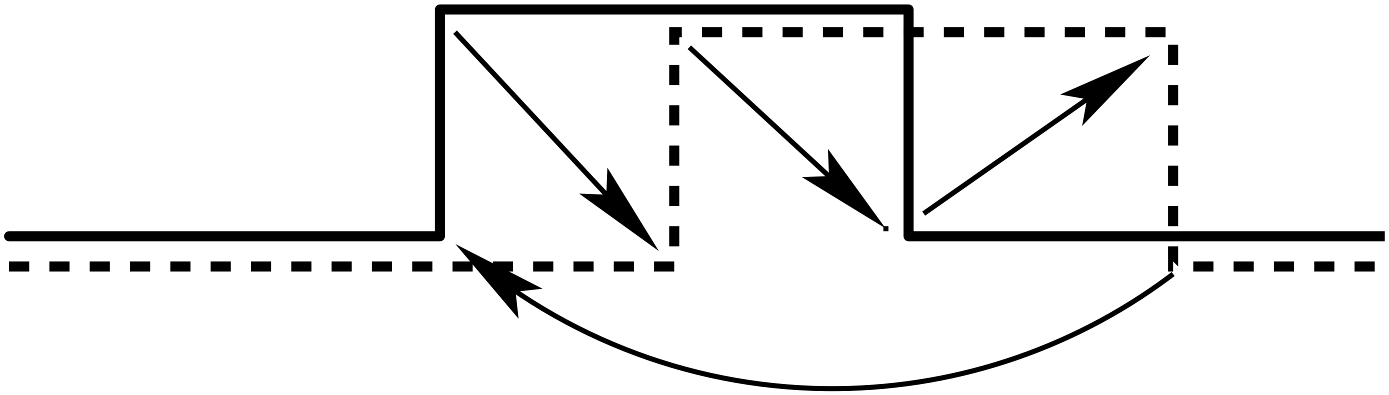
```
r_on
  when
    $r = 0$
    $a = 1$
  then
    $r := 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end
```

# Weak Synchronization of Events



# Refinement: Strong action and reaction



- We add the following invariant

$$\boxed{\textbf{pat1\_1:} \quad ca \leq cr + 1}$$

- Remember invariant **pat0_5**

$$\boxed{\textbf{pat0\_5:} \quad cr \leq ca}$$

We have thus: $ca = cr \ \vee \ ca = cr + 1$

# Summary (so far) of the Strong Sncchronization

**pat1_1:** $ca \leq cr + 1$

```
a_on
  when
    $a = 0$
  then
    $a := 1$
    $ca := ca + 1$
  end
```

```
a_off
  when
    $a = 1$
  then
    $a := 0$
  end
```

```
r_on
  when
    $r = 0$
    $a = 1$
  then
    $r := 1$
    $cr := cr + 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end
```

Nothing guarantees that the invariant is preserved

# D E M 0 (Showing Problems and Finding Solutions)

# Merging the two invariants

- Putting together these two invariants

**pat1_2:** $a = 0 \quad \Rightarrow \quad ca = cr$

**pat1_3:** $a = 1 \ \wedge \ r = 1 \quad \Rightarrow \quad ca = cr$

- leads to the following

**pat1_4:** $a = 0 \ \vee \ r = 1 \quad \Rightarrow \quad ca = cr$

# Simplifying the Invariants

**pat0_5:** $cr \leq ca$

**pat0_6:** $a = 1 \ \wedge \ r = 0 \ \Rightarrow \ cr < ca$

**pat1_1:** $ca \leq cr + 1$

**pat1_4:** $a = 0 \ \vee \ r = 1 \ \Rightarrow \ ca = cr$

This can be simplified to

**pat2_1:** $a = 1 \ \wedge \ r = 0 \ \Rightarrow \ ca = cr + 1$

**pat2_2:** $a = 0 \ \vee \ r = 1 \ \Rightarrow \ ca = cr$

# Summary of the State for the Strong Reaction

**pat0_1:** $a \in \{0, 1\}$

**pat0_2:** $r \in \{0, 1\}$

**pat0_3:** $ca \in \mathbb{N}$

**pat0_4:** $cr \in \mathbb{N}$

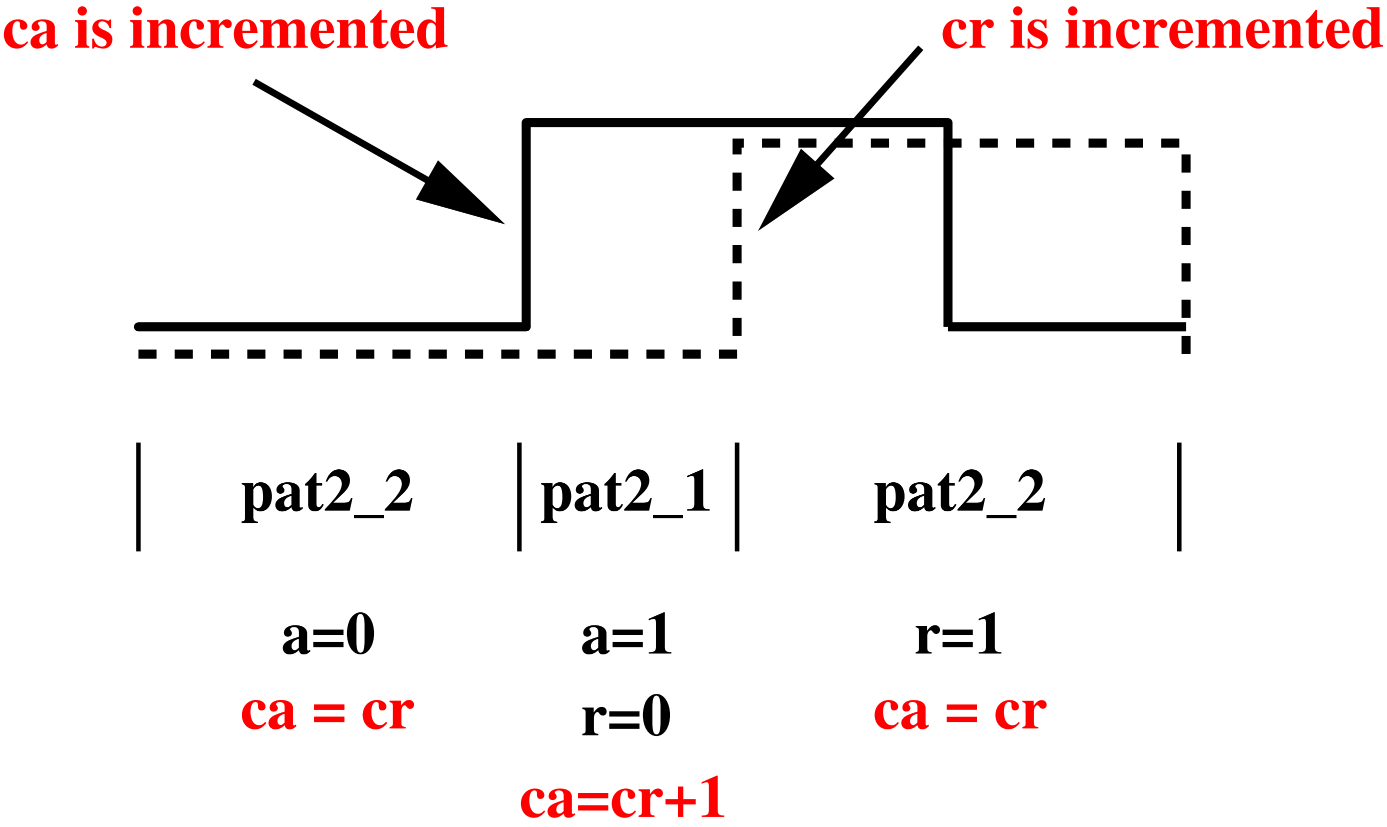
**pat2_1:** $a = 1 \;\land\; r = 0 \;\Rightarrow\; ca = cr + 1$

**pat2_2:** $a = 0 \;\lor\; r = 1 \;\Rightarrow\; ca = cr$

# Intuition Behind the two Invariants

**pat2_1:** $a = 1 \wedge r = 0 \Rightarrow ca = cr + 1$

**pat2_2:** $a = 0 \vee r = 1 \Rightarrow ca = cr$



ca is incremented

cr is incremented

| pat2_2 | pat2_1 | pat2_2 |
|---|---|---|
| a=0 | a=1 | r=1 |
| ca = cr | r=0 | ca = cr |
| | ca=cr+1 | |

# Summary of the Events for the Strong Reaction

The counters have

been removed

```
a_on
    when
        $a = 0$
        $r = 0$
    then
        $a := 1$
    end
```

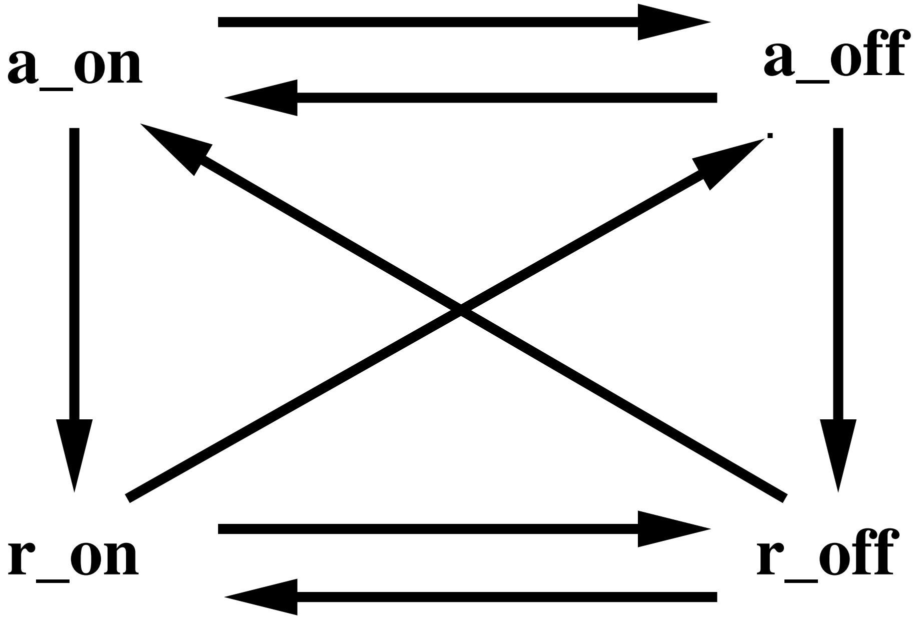
```
a_off
    when
        $a = 1$
        $r = 1$
    then
        $a := 0$
    end
```

```
init
    $a := 0$
    $r := 0$
```

```
r_on
    when
        $r = 0$
        $a = 1$
    then
        $r := 1$
    end
```

```
r_off
    when
        $r = 1$
        $a = 0$
    then
        $r := 0$
    end
```

# Strong Synchronization of Events



# What we Have Learned

- Proof failures helped us improving our models

- When an invariant preservation proof fails on an event,
  there are two solutions:
    - adding a new invariant
    - strengthening the guard

- Modelling considerations helped us choosing one or the other

- At the end, we reached a stable situation (fixpoint)

# 3. Writing the Requirement Document

# Requirements: Describing Equipment

| | |
|---|---|
| The system has got the following pieces of equipment: a Motor, a Clutch, and a Door | EQP_1 |

| | |
|---|---|
| Four Buttons are used to start and stop the motor, and engage and disengage the clutch | EQP_2 |

| | |
|---|---|
| A Controller is supposed to manage this equipment | EQP_3 |

# Requirements: Connection Constraints

| | |
|---|---|
| Buttons and Controller are weakly synchronized | FUN_1 |

| | |
|---|---|
| Controller are Equipment are strongly synchronized | FUN_2 |

# Requirements: Relationship Between Motor and Clutch

| | |
|---|---|
| When the clutch is engaged, the motor must work | SAF_1 |

| | |
|---|---|
| When the clutch is engaged, the door must be closed | SAF_2 |

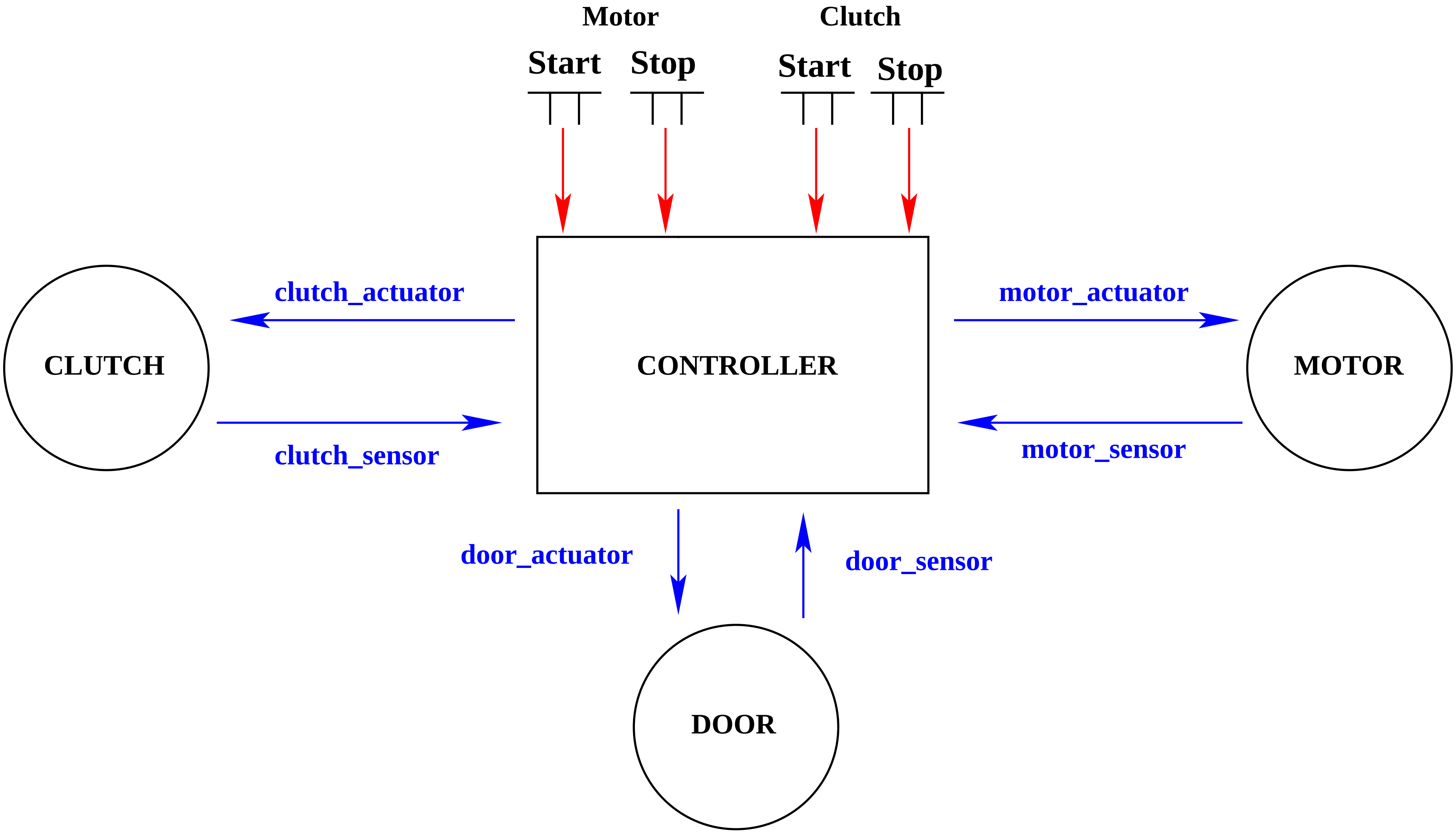
# Requirements: Relationship Between Door and Clutch

| | |
|---|---|
| When the clutch is engaged, the door cannot be closed several times, ONLY ONCE | FUN_3 |

| | |
|---|---|
| When the door is closed, the clutch cannot be disengaged several times, ONLY ONCE | FUN_4 |

| | |
|---|---|
| Opening and closing the door are not independent. It must be synchronized with disengaging and engaging the clutch | FUN_5 |

# Overview


Motor
Start
Stop
Clutch
Start
Stop
clutch_actuator
CLUTCH
clutch_sensor
CONTROLLER
motor_actuator
MOTOR
motor_sensor
door_actuator
door_sensor
DOOR

# 4. Proposing a Refinement Strategy

# Refinement Strategy

- Initial model: Connecting the controller to the motor

- 1st refinement: Connecting the motor buttons to the controller

- 2nd refinement: Connecting the controller to the clutch

- 3rd refinement: Constraining the clutch and the motor

# Refinement Strategy (cont’d)

- 4th refinement: Connecting the controller to the door

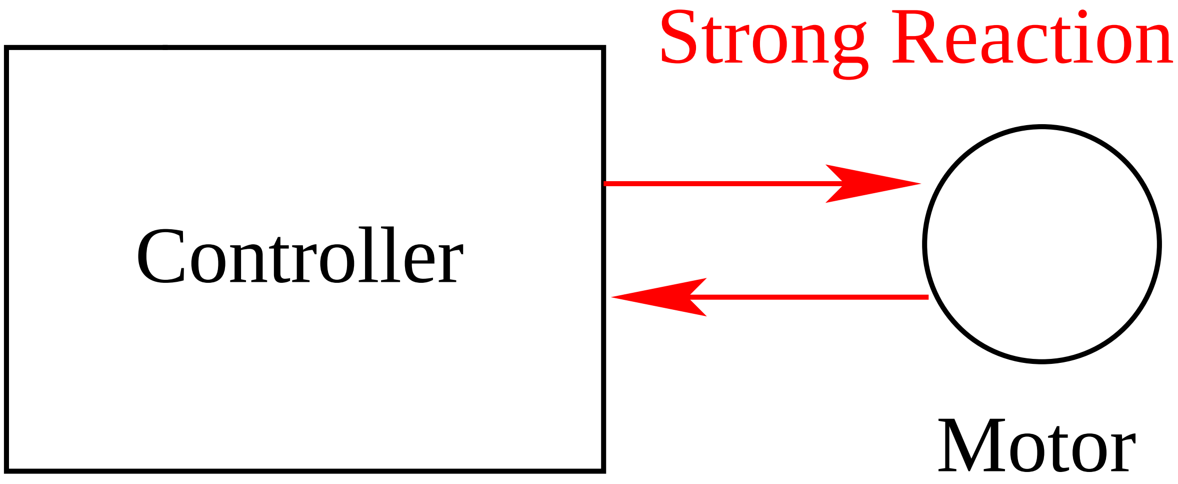
- 5th refinement: Constraining the clutch and the door

- 6th refinement: More constraints between clutch and door

- 7th refinement: Connecting the clutch buttons to the controller

# 5. Development of the Model using Refinements and Design Patterns

# Initial Model: Connecting the Controller to the Motor



| | |
|---|---|
| Controller are Equipment are strongly synchronized | FUN_2 |

# Model for strong action and reaction: the Final Events

The counters have

been removed

```
a_on
  when
    $a = 0$
    $r = 0$
  then
    $a := 1$
  end
```

```
a_off
  when
    $a = 1$
    $r = 1$
  then
    $a := 0$
  end
```

```
init
    $a := 0$
    $r := 0$
```

```
r_on
  when
    $r = 0$
    $a = 1$
  then
    $r := 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end
```

# Initial Model: the Context

**set:** $STATUS$

**constants:** $stopped$<br>$working$

**axm0_1:** $STATUS = \{stopped, working\}$

**axm0_2:** $stopped \neq working$

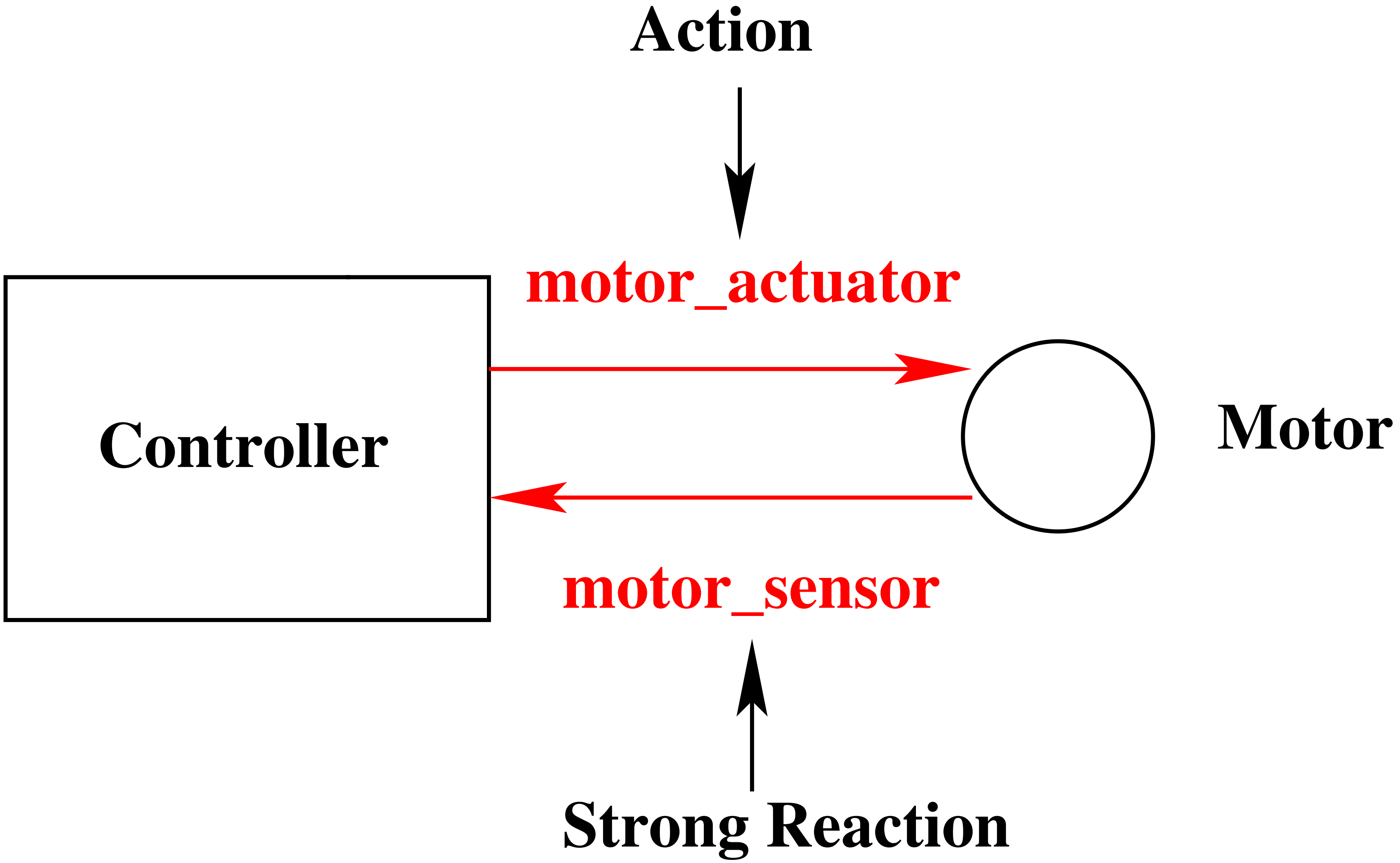
# Initial Model: the State

**variables:** $motor\_actuator$
$motor\_sensor$

**inv0_1:** $motor\_sensor \in STATUS$

**inv0_2:** $motor\_actuator \in STATUS$

# Initial Model: the Synchronization



# Pattern Instantiation

- We instantiate the strong pattern as follows:

$$\begin{array}{lll}
\boldsymbol{a} & \leadsto & \boldsymbol{motor\_actuator} \\
\boldsymbol{r} & \leadsto & \boldsymbol{motor\_sensor} \\
\mathbf{0} & \leadsto & \boldsymbol{stopped} \\
\mathbf{1} & \leadsto & \boldsymbol{working}
\end{array}$$

| | | |
|---|---|---|
| a_on | ⇝ | treat_start_motor |
| a_off | ⇝ | treat_stop_motor |
| r_on | ⇝ | Motor_start |
| r_off | ⇝ | Motor_stop |

- Convention: Controller events start with "treat_"

# Initial Model: Initialization

init

$a := 0$

$r := 0$

init

$motor\_actuator := stopped$

$motor\_sensor := stopped$

# Initial Model: Controller Events (1)

```
a_on
    when
        $a = 0$
        $r = 0$
    then
        $a := 1$
    end
```

```
treat_start_motor
    when
        $motor\_actuator = stopped$
        $motor\_sensor = stopped$
    then
        $motor\_actuator := working$
    end
```

# Initial Model: Environment Event (1)

```
r_on
    when
        $r = 0$
        $a = 1$
    then
        $r := 1$
    end
```

```
Motor_start
    when
        $motor\_sensor = stopped$
        $motor\_actuator = working$
    then
        $motor\_sensor := working$
    end
```

# Initial Model: Controller Events (2)

```
a_off
    when
        $a = 1$
        $r = 1$
    then
        $a := 0$
    end
```

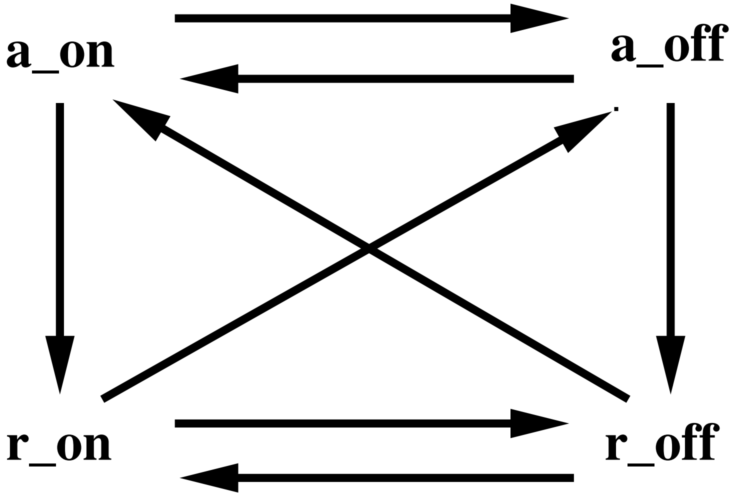
```
treat_stop_motor
    when
        $motor\_actuator = working$
        $motor\_sensor = working$
    then
        $motor\_actuator := stopped$
    end
```

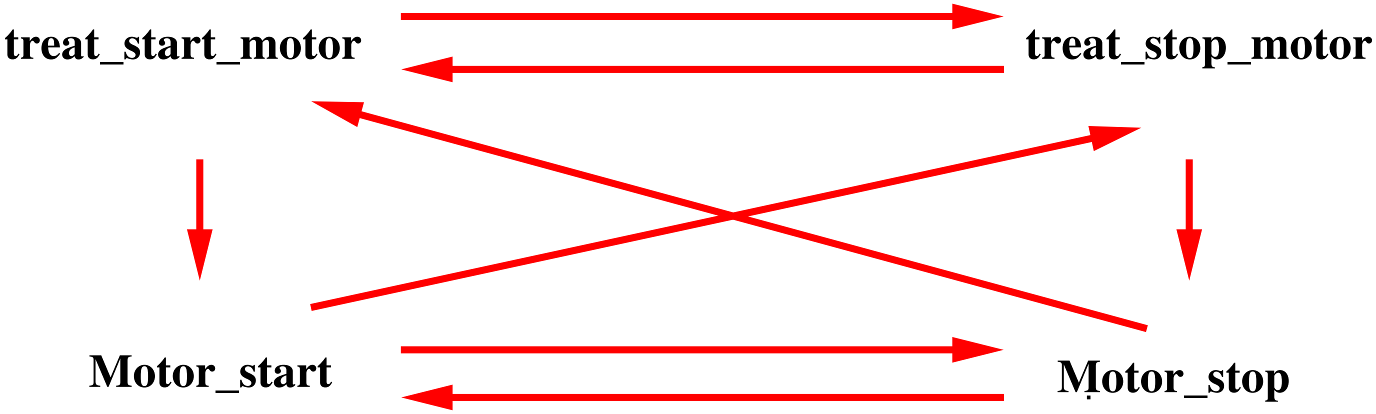
# Initial Model: Environment Event (2)

```
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end
```

```
Motor_stop
  when
    $motor\_sensor = working$
    $motor\_actuator = stopped$
  then
    $motor\_sensor := stopped$
  end
```

# Synchronization



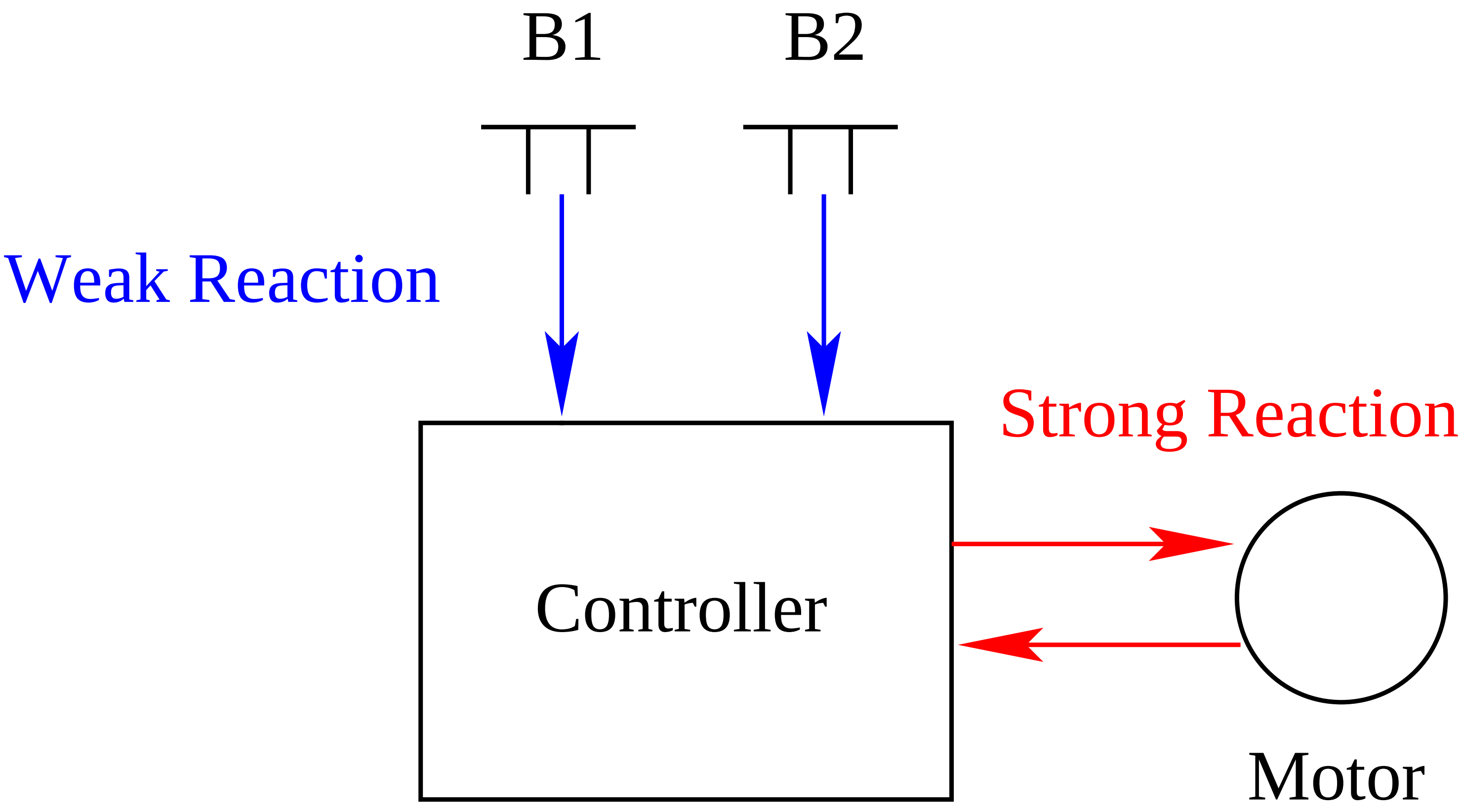


# Initial Model: Summary of the Events

- Environment
  - motor_start
  - motor_stop

- Controller
  - treat_start_motor
  - treat_stop_motor

# 1st Reft.: Connecting the Motor Buttons to the Controller



| | |
|---|---|
| Buttons and Controller are weakly synchronized | FUN_1 |

# Model for weak action and reaction: the Final Events

The counters have

been removed

```
a_on
  when
    $a = 0$
  then
    $a := 1$
  end
```

```
a_off
  when
    $a = 1$
  then
    $a := 0$
  end
```

```
init
  $a := 0$
  $r := 0$
```

```
r_on
  when
    $r = 0$
    $a = 1$
  then
    $r := 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end
```

# First Refinement: the State

**variables:** $\ldots$<br>

$start\_motor\_button$

$stop\_motor\_button$

$start\_motor\_impulse$

$stop\_motor\_impulse$

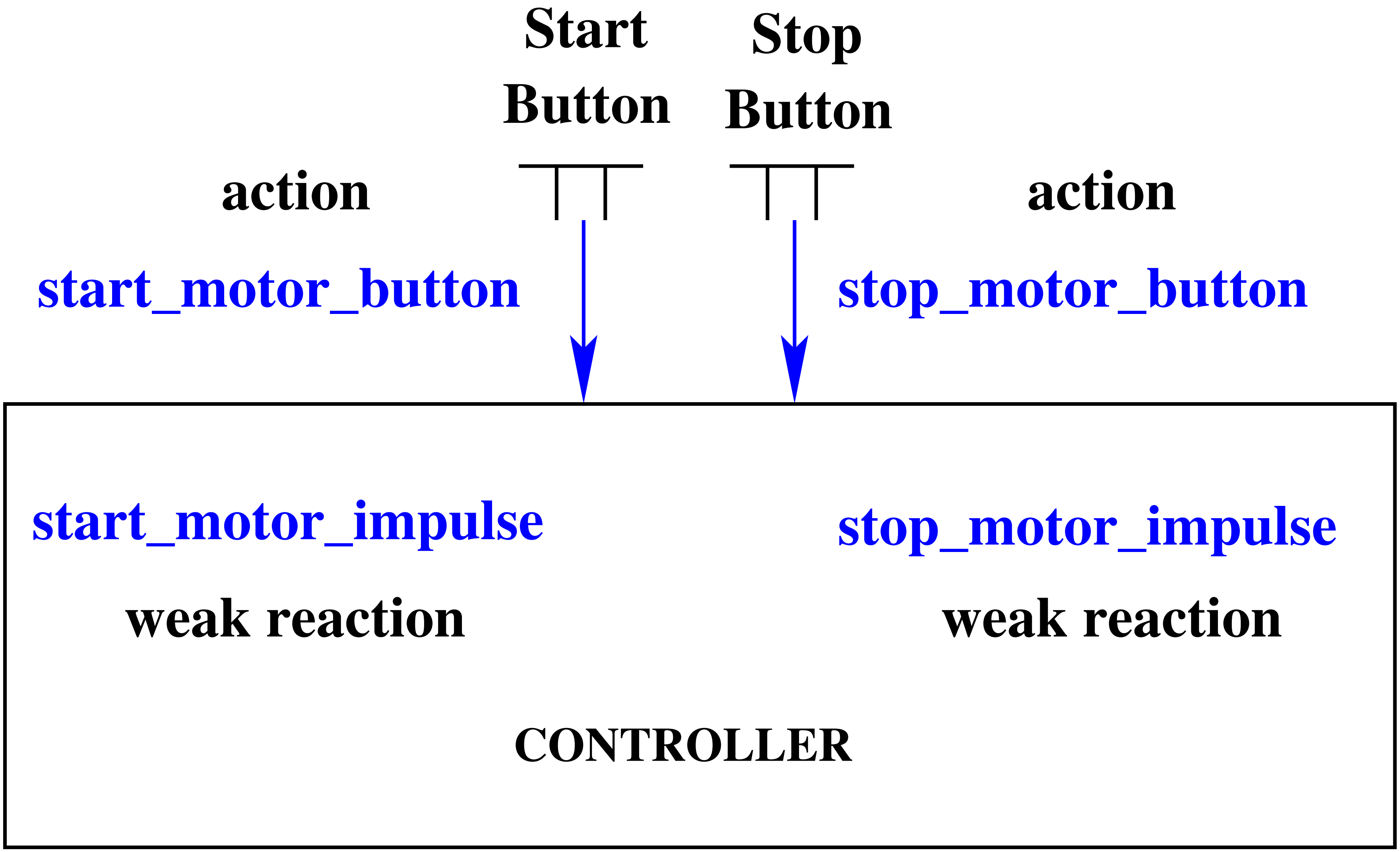
**inv1_1:** $stop\_motor\_button \in \mathrm{BOOL}$

**inv1_2:** $start\_motor\_button \in \mathrm{BOOL}$

**inv1_3:** $stop\_motor\_impulse \in \mathrm{BOOL}$

**inv1_4:** $start\_motor\_impulse \in \mathrm{BOOL}$

## First Refinement: the State



# Pattern Instantiation

- We instantiate the pattern as follows:

$$\begin{array}{lcl}
a & \rightsquigarrow & \mathit{start\_motor\_button} \\
r & \rightsquigarrow & \mathit{start\_motor\_impulse} \\
0 & \rightsquigarrow & \mathbf{FALSE} \\
1 & \rightsquigarrow & \mathbf{TRUE}
\end{array}$$

| | | |
|---|---|---|
| a_on | $\rightsquigarrow$ | push_start_motor_button |
| a_off | $\rightsquigarrow$ | release_stop_motor_button |
| r_on | $\rightsquigarrow$ | treat_push_start_motor_button |
| r_off | $\rightsquigarrow$ | treat_release_start_motor_button |

- We rename treat_start_motor as treat_push_start_motor_button

# 1st Refinement: Refinement of Initialization

init

$$a := 0$$
$$r := 0$$

init

$$motor\_actuator := stopped$$
$$motor\_sensor := stopped$$
$$start\_motor\_button := \mathrm{FALSE}$$
$$start\_motor\_impulse := \mathrm{FALSE}$$

# First Refinement: New Environment Events (1)

```
a_on
  when
    $a = 0$
  then
    $a := 1$
  end
```

```
push_start_motor_button
  when
    $start\_motor\_button = \mathrm{FALSE}$
  then
    $start\_motor\_button := \mathrm{TRUE}$
  end
```

```
a_off
  when
    $a = 1$
  then
    $a := 0$
  end
```

```
release_start_motor_button
  when
    $start\_motor\_button = \mathrm{TRUE}$
  then
    $start\_motor\_button := \mathrm{FALSE}$
  end
```

# First Refinement: Refining Controller Events (1)

```
r_on

  when
    $r = 0$
    $a = 1$

  then
    $r := 1$

  end
```

```
treat_push_start_motor_button
  refines
    treat_start_motor
  when
    $start\_motor\_impulse = \mathrm{FALSE}$
    $start\_motor\_button = \mathrm{TRUE}$
    $motor\_actuator = stopped$
    $motor\_sensor = stopped$
  then
    $start\_motor\_impulse := \mathrm{TRUE}$
    $motor\_actuator := working$
  end
```

- This is the most important slide of the talk

- We can see how patterns can be superposed

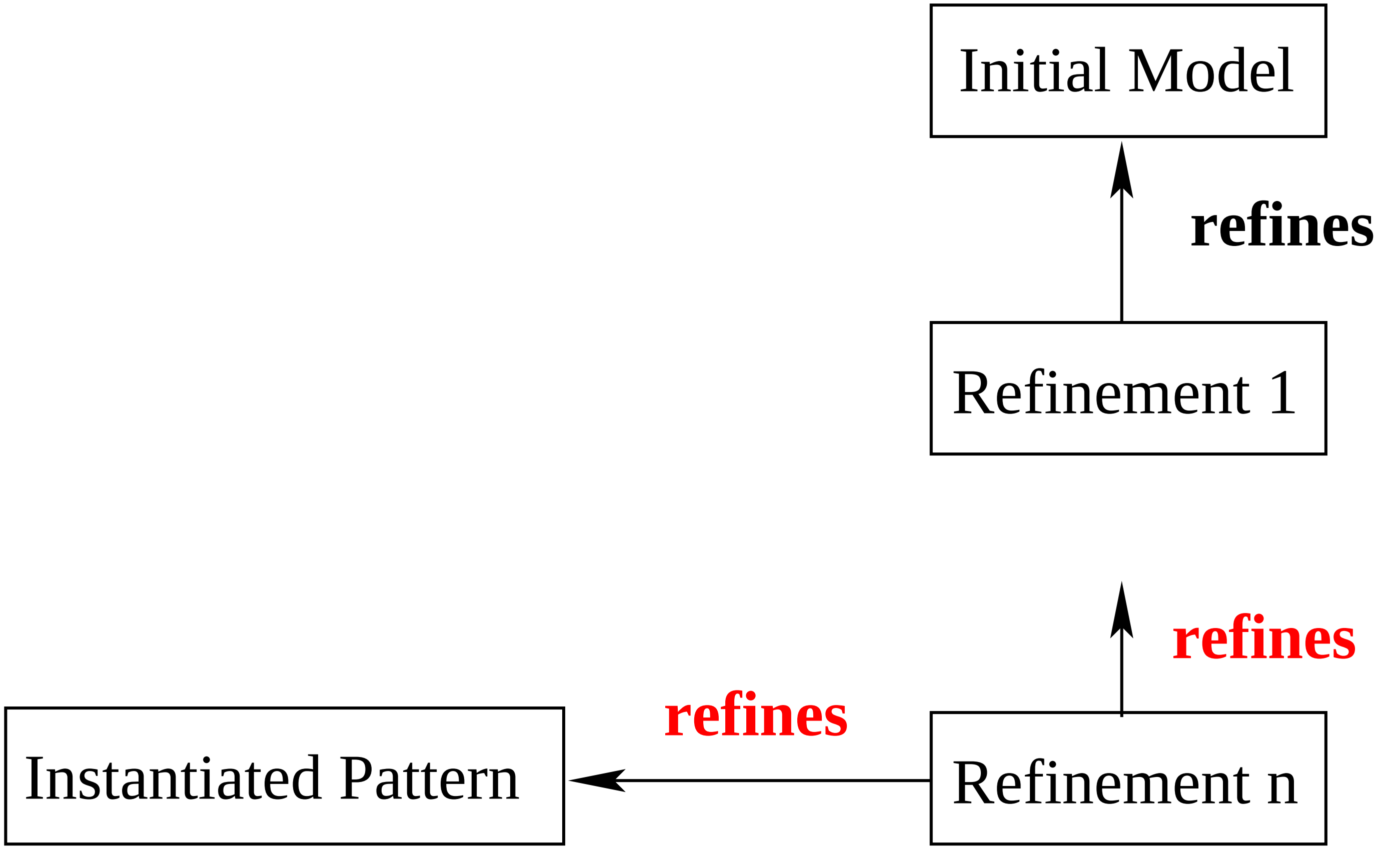
```
a_on
   when
      $a = 0$
      $r = 0$
   then
      $a := 1$
   end
```

```
treat_start_motor
   when
      $motor\_actuator = stopped$
      $motor\_sensor = stopped$
   then
      $motor\_actuator := working$
   end
```

```
r_on
   when
      r = 0
      a = 1
   then
      r := 1
   end
```

```
treat_push_start_motor_button
   when
      $start\_motor\_impulse = \mathrm{FALSE}$
      $start\_motor\_button = \mathrm{TRUE}$
      $motor\_actuator = stopped$
      $motor\_sensor = stopped$
   then
      $start\_motor\_impulse := \mathrm{TRUE}$
      $motor\_actuator := working$
   end
```

# Design Pattern Integration within a Development



# First Refinement: New Controller Events (1)

```
r_off
    when
        $r = 1$
        $a = 0$
    then
        $r := 0$
    end
```

```
treat_release_start_motor_button
    when
        $start\_motor\_impulse = \mathrm{TRUE}$
        $start\_motor\_button = \mathrm{FALSE}$
    then
        $start\_motor\_impulse := \mathrm{FALSE}$
    end
```

# Pattern Instantiation

- We instantiate the pattern as follows:

| | | |
|---|---|---|
| $\boldsymbol{a}$ | $\rightsquigarrow$ | $\boldsymbol{stop\_motor\_button}$ |
| $\boldsymbol{r}$ | $\rightsquigarrow$ | $\boldsymbol{stop\_motor\_impulse}$ |
| $\mathbf{0}$ | $\rightsquigarrow$ | $\mathbf{FALSE}$ |
| $\mathbf{1}$ | $\rightsquigarrow$ | $\mathbf{TRUE}$ |

| | | |
|---|---|---|
| a_on | $\rightsquigarrow$ | push_stop_motor_button |
| a_off | $\rightsquigarrow$ | release_stop_motor_button |
| r_on | $\rightsquigarrow$ | treat_push_stop_motor_button |
| r_off | $\rightsquigarrow$ | treat_release_stop_motor_button |

# 1st Refinement: More Refinement of Initialization

init

$a := 0$

$r := 0$

init

$motor\_actuator := stopped$

$motor\_sensor := stopped$

$start\_motor\_button := \mathrm{FALSE}$

$start\_motor\_impulse := \mathrm{FALSE}$

$stop\_motor\_button := \mathrm{FALSE}$

$stop\_motor\_impulse := \mathrm{FALSE}$

# First Refinement: New Environment Events

```
a_on
  when
    $a = 0$
  then
    $a := 1$
  end
```

```
push_stop_motor_button
  when
    $stop\_motor\_button = \mathrm{FALSE}$
  then
    $stop\_motor\_button := \mathrm{TRUE}$
  end
```

```
a_off
  when
    $a = 1$
  then
    $a := 0$
  end
```

```
release_stop_motor_button
  when
    $stop\_motor\_button = \mathrm{TRUE}$
  then
    $stop\_motor\_button := \mathrm{FALSE}$
  end
```

# First Refinement: Refining Controller Events (2)

```
r_on
    when
        $r = 0$
        $a = 1$
    then
        $r := 1$
    end
```

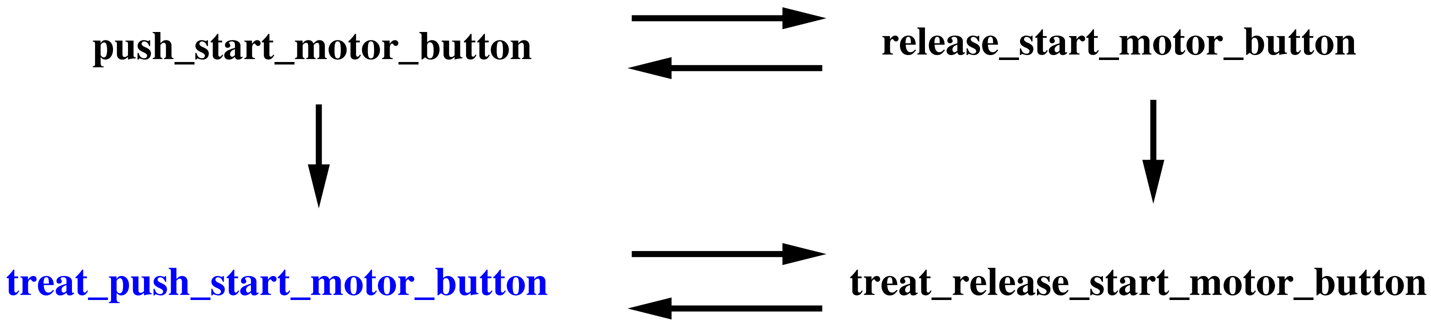
```
treat_push_stop_motor_button
    refines
        treat_stop_motor
    when
        $stop\_motor\_impulse = \mathrm{FALSE}$
        $stop\_motor\_button = \mathrm{TRUE}$
        $motor\_sensor = working$
        $motor\_actuator = working$
    then
        $stop\_motor\_impulse := \mathrm{TRUE}$
        $motor\_actuator := stopped$
    end
```

# First Refinement: New Controller Events (2)

```
r_off
    when
        $r = 1$
        $a = 0$
    then
        $r := 0$
    end
```

```
treat_release_stop_motor_button
    when
        $stop\_motor\_impulse = \mathrm{TRUE}$
        $stop\_motor\_button = \mathrm{FALSE}$
    then
        $stop\_motor\_impulse := \mathrm{FALSE}$
    end
```

# Independent Synchronizations



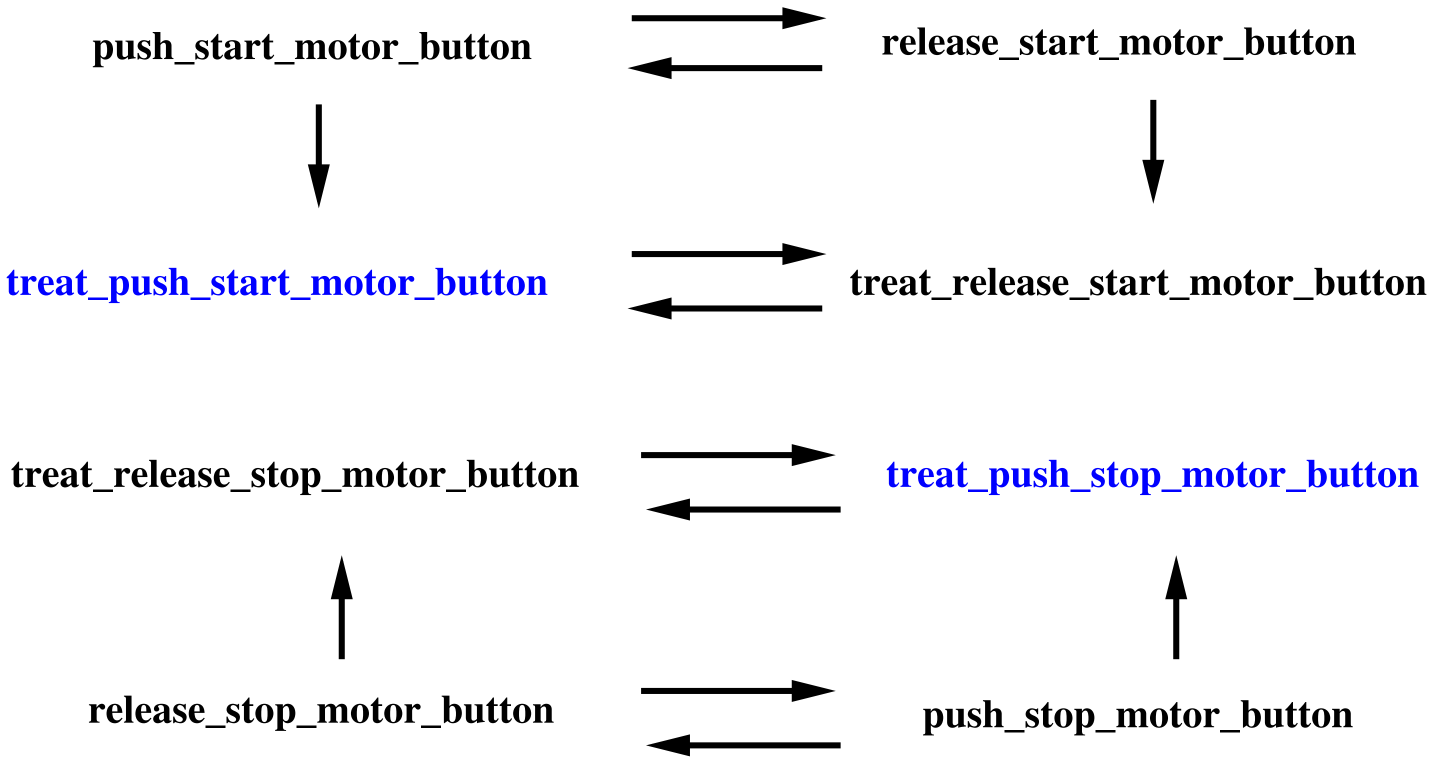
push_start_motor_button

release_start_motor_button

treat_push_start_motor_button

treat_release_start_motor_button

# Independent Synchronizations



push_start_motor_button

release_start_motor_button

treat_push_start_motor_button

treat_release_start_motor_button

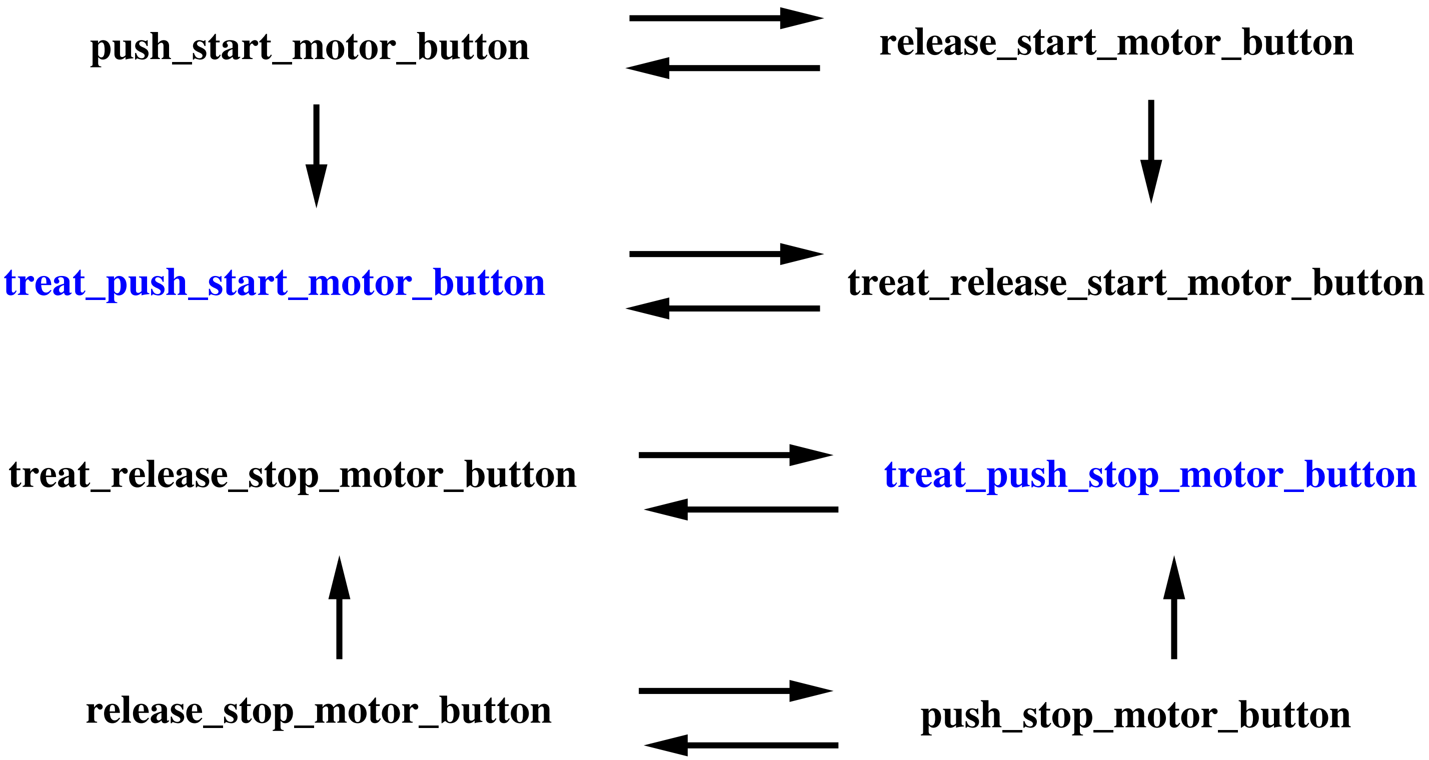
treat_release_stop_motor_button

treat_push_stop_motor_button

release_stop_motor_button

push_stop_motor_button

# Independent Synchronizations



push_start_motor_button

release_start_motor_button

treat_push_start_motor_button

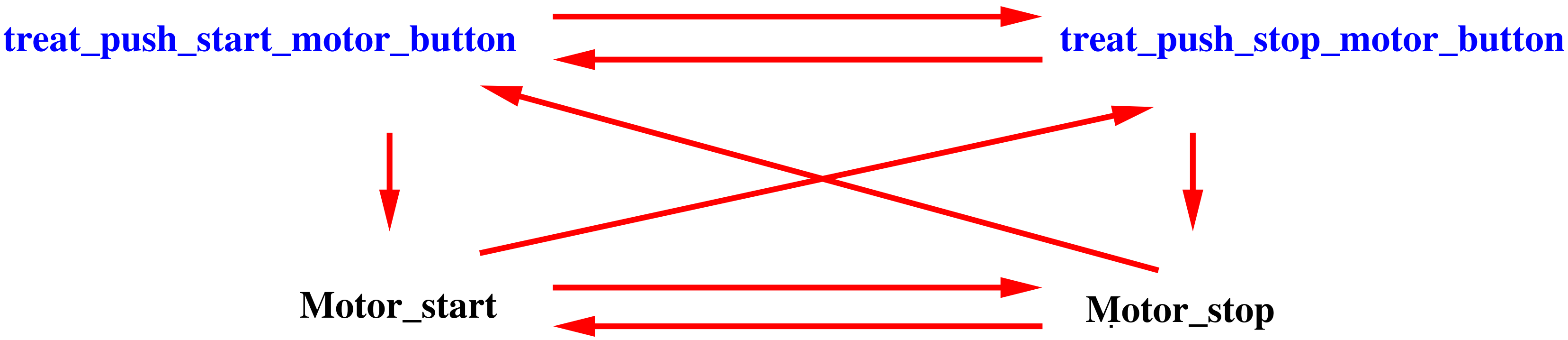
treat_release_start_motor_button

treat_release_stop_motor_button

treat_push_stop_motor_button

release_stop_motor_button

push_stop_motor_button



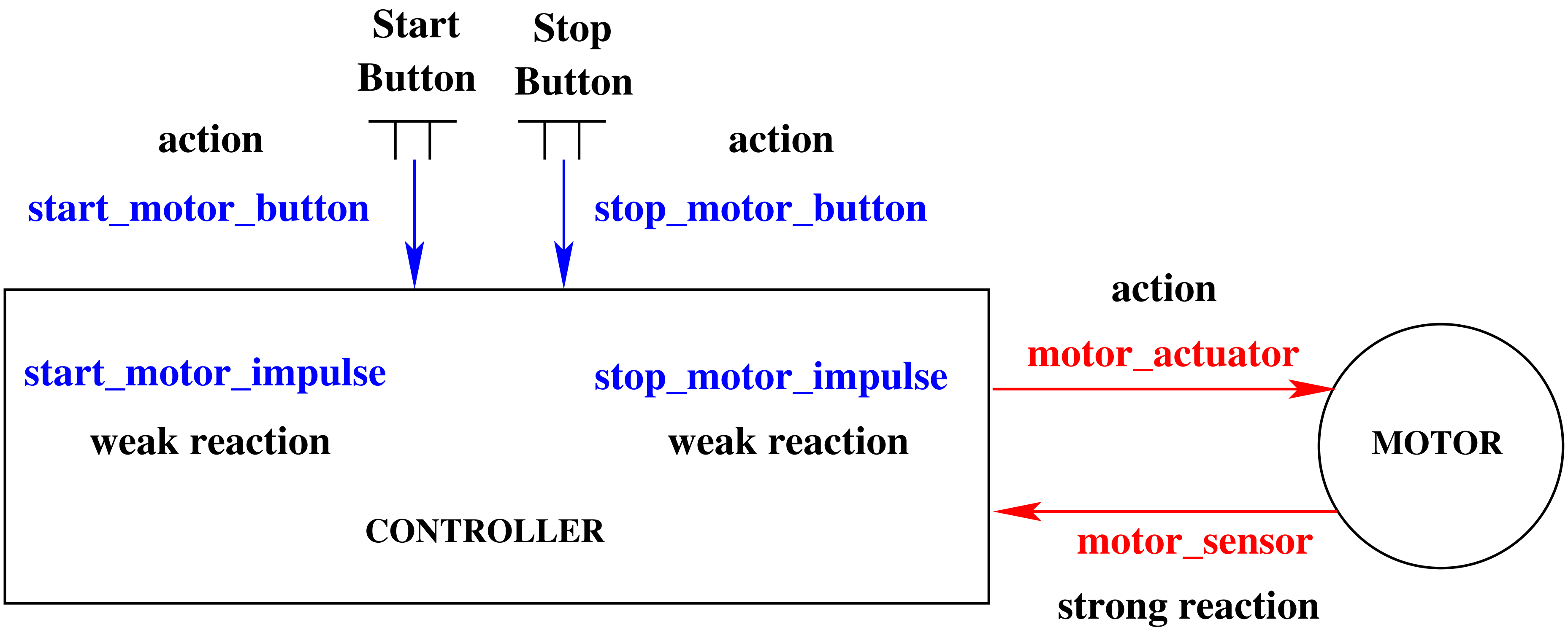
treat_push_start_motor_button

treat_push_stop_motor_button

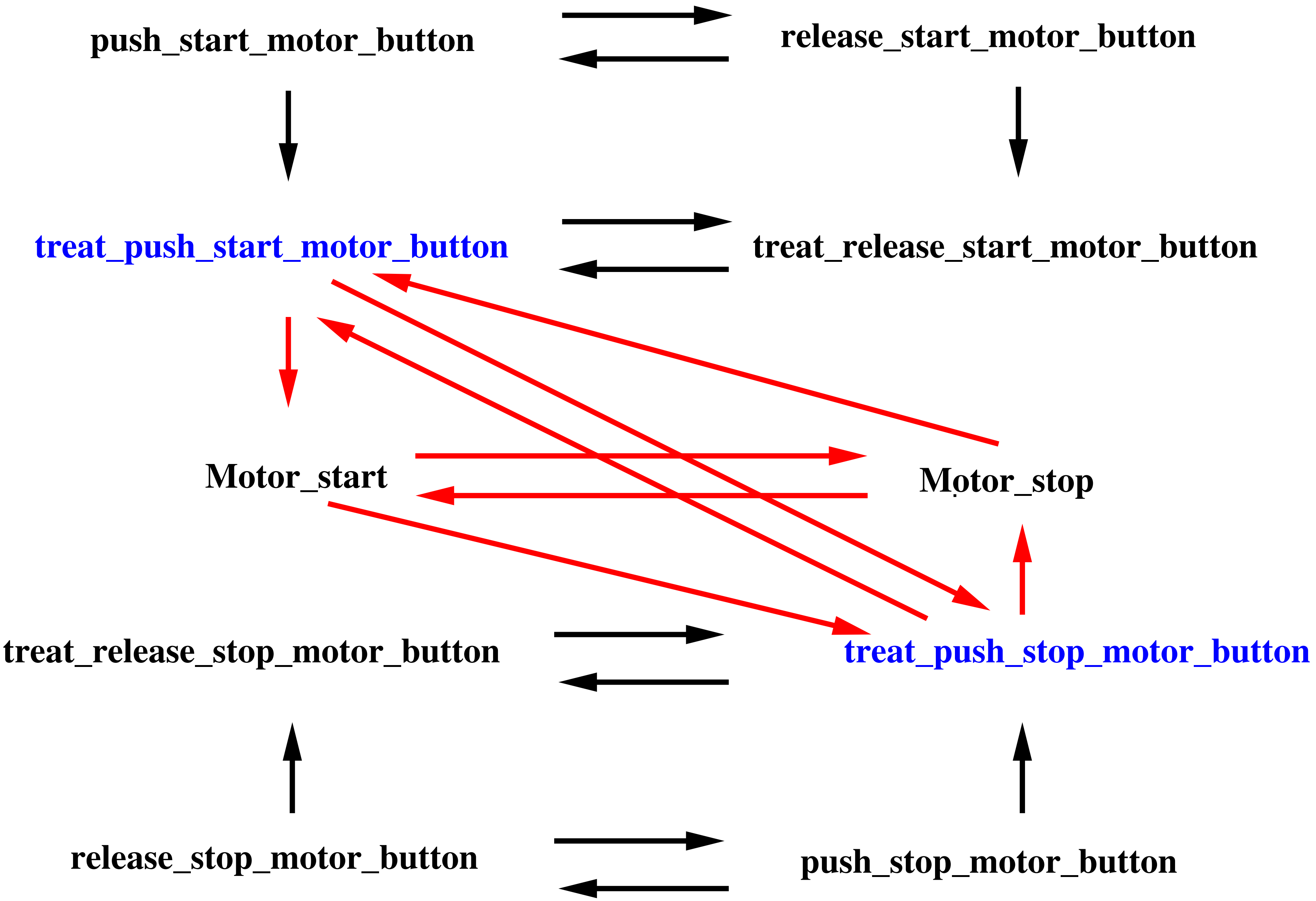
Motor_start

Motor_stop

# Weak and Strong Reactions Together



# Combined Synchronizations



push_start_motor_button

release_start_motor_button

treat_push_start_motor_button

treat_release_start_motor_button

Motor_start

Motor_stop

treat_release_stop_motor_button

treat_push_stop_motor_button

release_stop_motor_button

push_stop_motor_button

# Problems with treat_push_start_motor_button

```
treat_push_start_motor_button
    refines
        treat_start_motor
    when
        $start\_motor\_impulse = \mathrm{FALSE}$
        $start\_motor\_button = \mathrm{TRUE}$
        $motor\_actuator = stopped$
        $motor\_sensor = stopped$
    then
        $start\_motor\_impulse := \mathrm{TRUE}$
        $motor\_actuator := working$
    end
```

- What happens when the following hold

$$\neg\,(motor\_actuator = stopped \quad \wedge \quad motor\_sensor = stopped)$$

- We need another event

# Problems with treat_push_start_motor_button

```
treat_push_start_motor_button
  refines
    treat_start_motor
  when
    $start\_motor\_impulse = \mathrm{FALSE}$
    $start\_motor\_button = \mathrm{TRUE}$
    $motor\_actuator = stopped$
    $motor\_sensor = stopped$
  then
    $start\_motor\_impulse := \mathrm{TRUE}$
    $motor\_actuator := working$
  end
```

```
treat_push_start_motor_button_false

  when
    $start\_motor\_impulse = \mathrm{FALSE}$
    $start\_motor\_button = \mathrm{TRUE}$
    $\neg\ (motor\_actuator = stopped\ \wedge$
       $motor\_sensor = stopped)$
  then
    $start\_motor\_impulse := \mathrm{TRUE}$

  end
```

- In the second case, the button has been pushed but the internal conditions are not met

- However, we need to record that the button has been pushed:

$$start\_motor\_impulse := \mathrm{TRUE}$$

# Problems with treat_push_stop_motor_button

```
treat_push_stop_motor_button
  refines
    treat_stop_motor
  when
    $stop\_motor\_impulse = \mathrm{FALSE}$
    $stop\_motor\_button = \mathrm{TRUE}$
    $motor\_sensor = working$
    $motor\_actuator = working$
  then
    $stop\_motor\_impulse := \mathrm{TRUE}$
    $motor\_actuator := stopped$
  end
```

```
treat_push_stop_motor_button_false


  when
    $stop\_motor\_impulse = \mathrm{FALSE}$
    $stop\_motor\_button = \mathrm{TRUE}$
    $\neg\,(motor\_sensor = working \;\wedge$
       $motor\_actuator = working)$
  then
    $stop\_motor\_impulse := \mathrm{TRUE}$

  end
```

- In the second case, the button has been pushed but the internal conditions are not met

- However, we need to record that the button has been pushed:

$$stop\_motor\_impulse := \mathrm{TRUE}$$

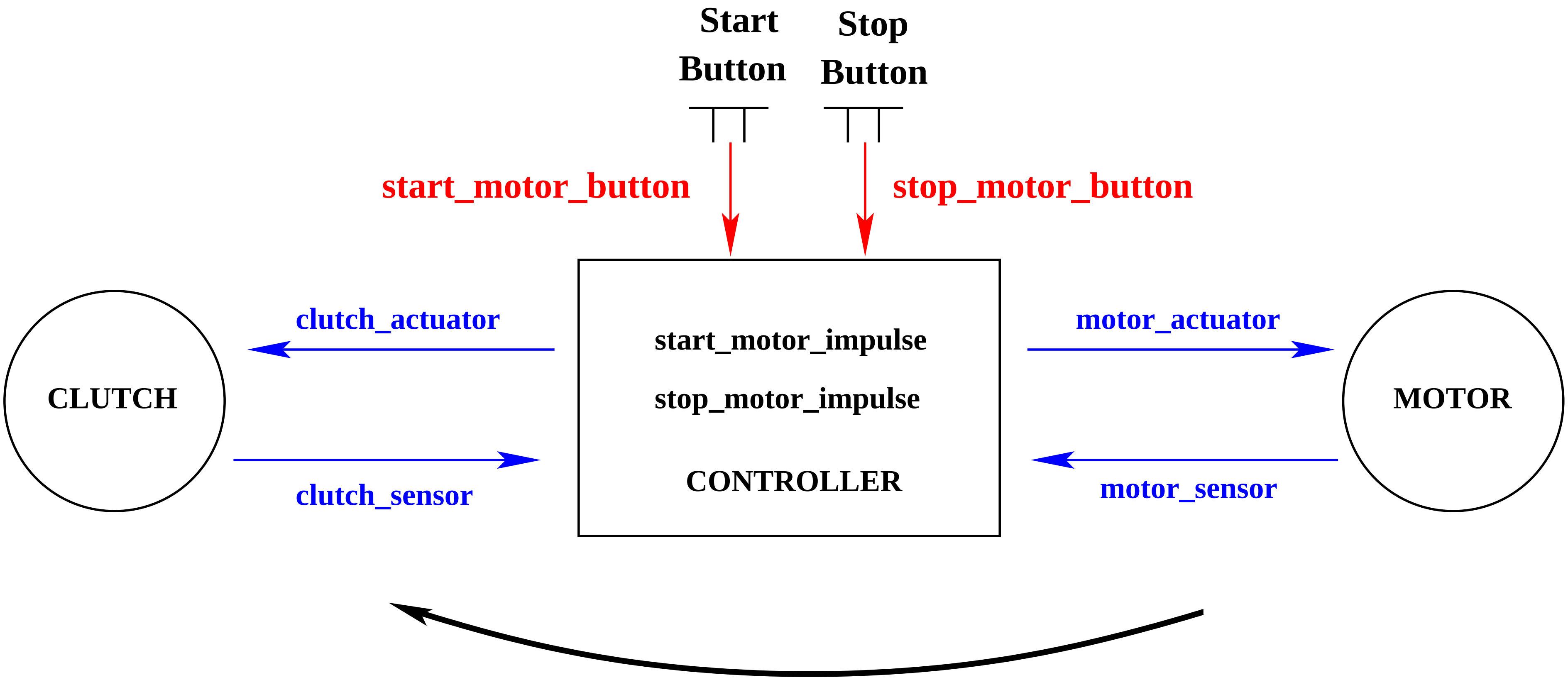
# First Refinement: Summary of the Events (1)

- Environment
    - motor_start
    - motor_stop
    - push_start_motor_button
    - release_start_motor_button
    - push_stop_motor_button
    - release_stop_motor_button

# First Refinement: Summary of the Events (2)

- Controller
    - treat_push_start_motor_button
    - treat_push_start_motor_button_false
    - treat_push_stop_motor_button
    - treat_push_stop_motor_button_false
    - treat_release_start_motor_button
    - treat_release_stop_motor_button

# 2nd Refinement: Connecting the Controller to the Clutch



## 2nd Refinement: Connecting the Controller to the Clutch

- We introduce the set in a new context:

$$CLUTCH = \{engaged, disengaged\}$$

- We copy the initial model where we instantiate:

$$\begin{array}{lcl}
motor & \rightsquigarrow & clutch \\
STATUS & \rightsquigarrow & CLUTCH \\
working & \rightsquigarrow & engaged \\
stopped & \rightsquigarrow & disengaged
\end{array}$$

# Second Refinement: Summary of the Events (1)

- Environment
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button

# Second Refinement: Summary of the Events (2)

- Controller
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch

# Third Refinement: Constraining the Clutch and the Motor

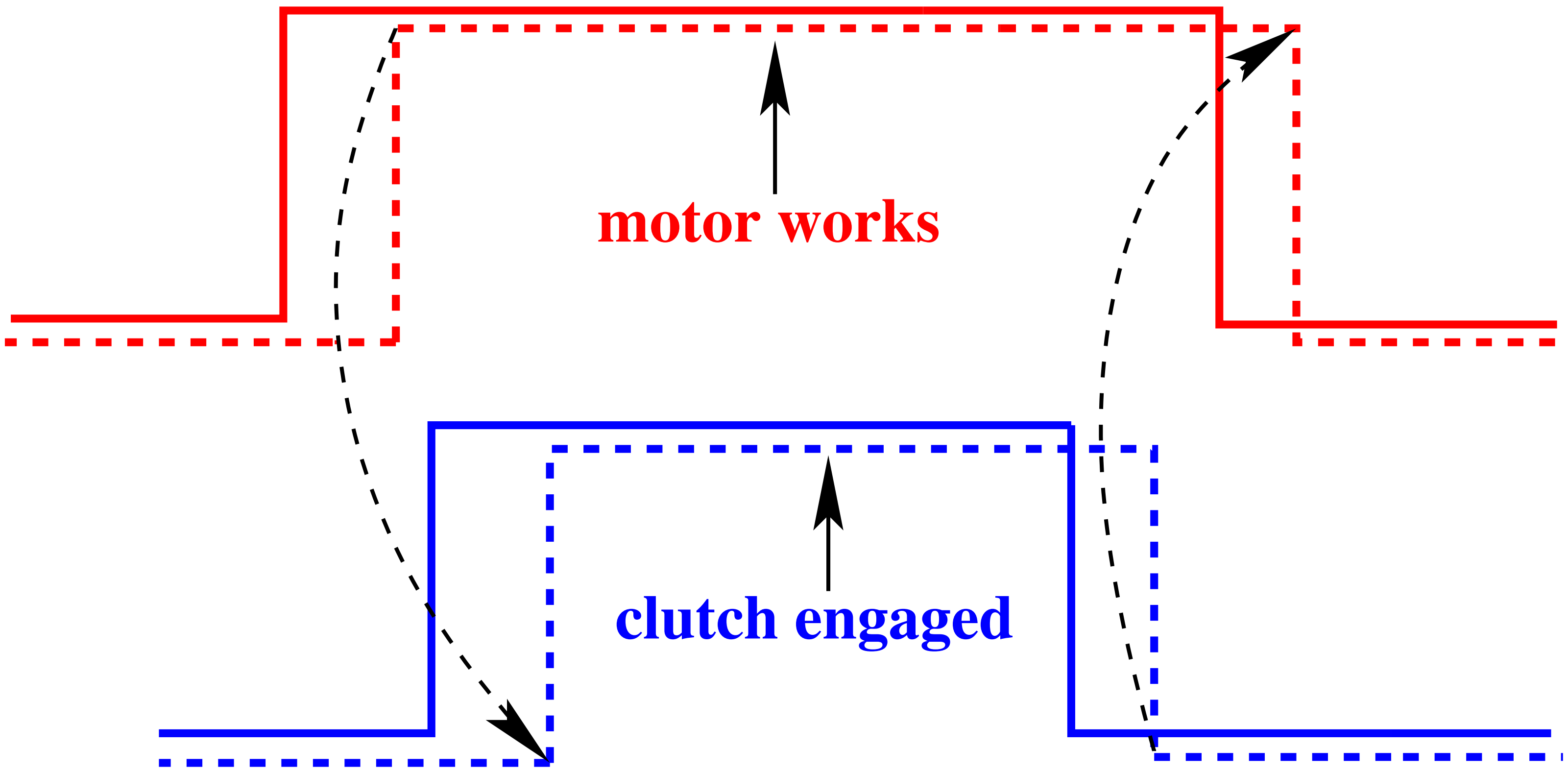
- An additional safety constraint

| | |
|---|---|
| When the clutch is engaged, the motor must work | SAF_1 |

- For this we develop ANOTHER DESIGN PATTERN

- It is called: Weak synchronization of two Strong Reactions

# Pattern: Weak Synchronization of Strong Reactions

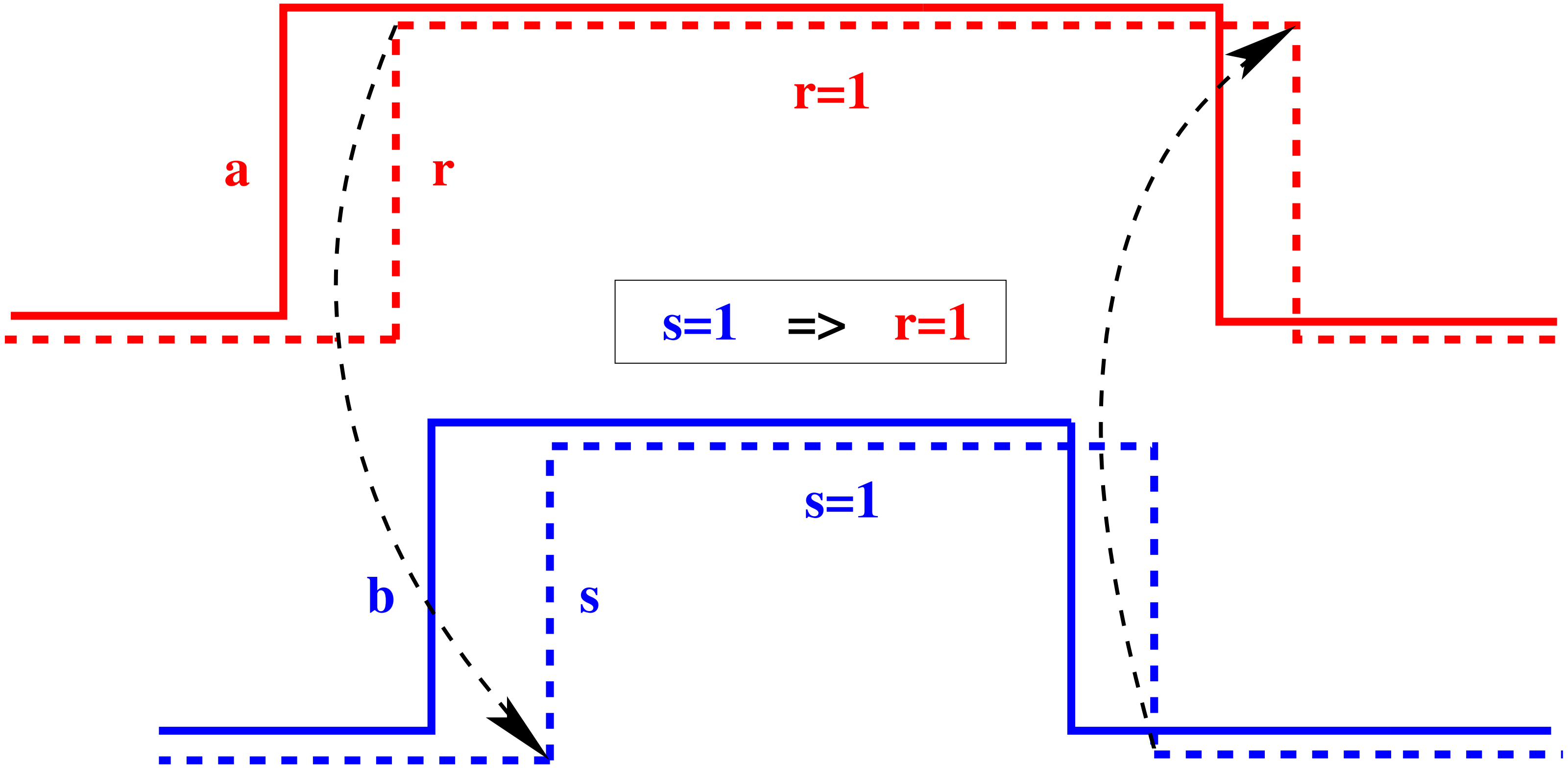


When the clutch is engaged

then

the motor must work

# Pattern: Weak Synchronization of Strong Reactions

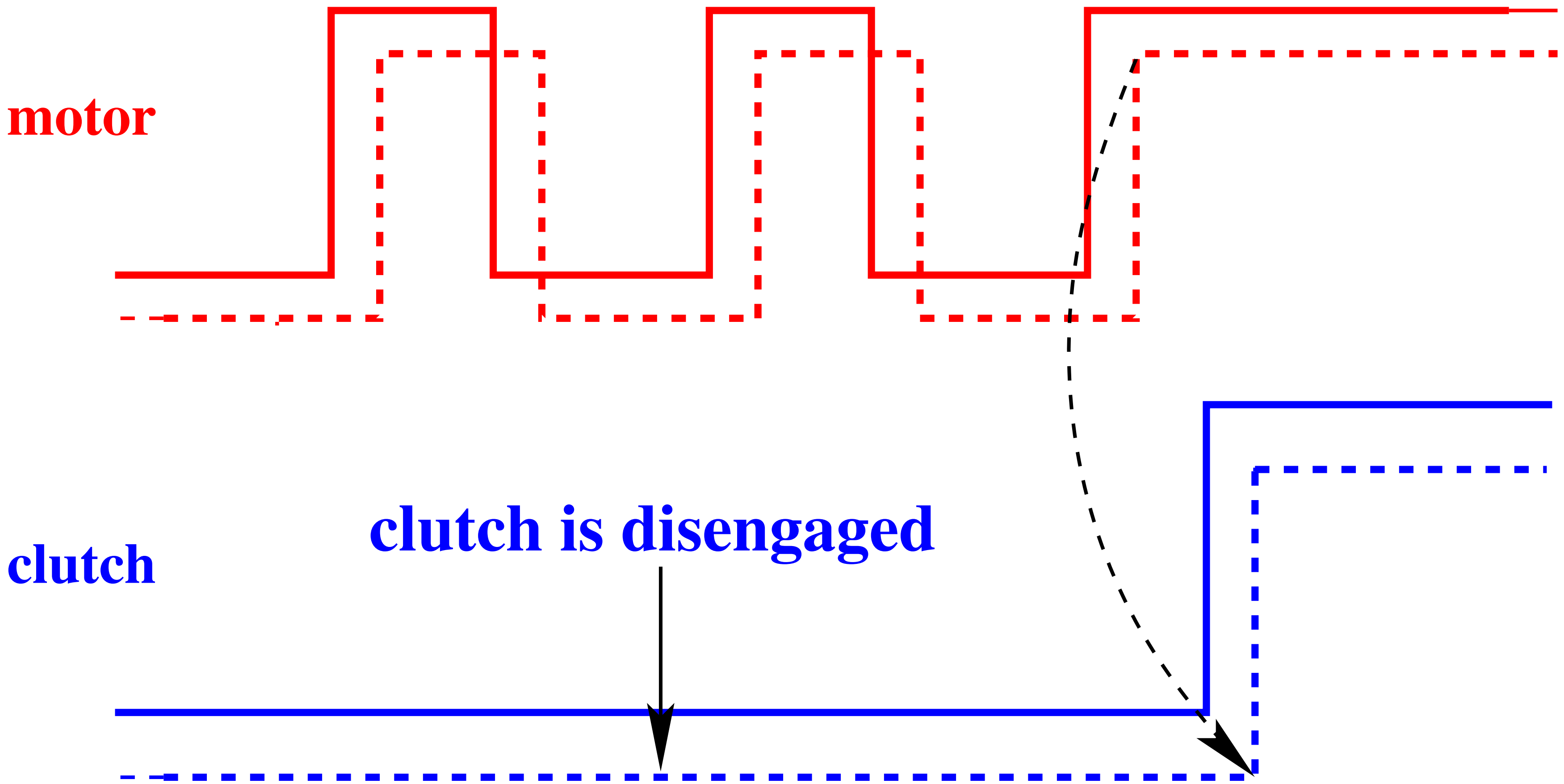


When the clutch is engaged

then

the motor must work

# The Synchronization is Weak (1)

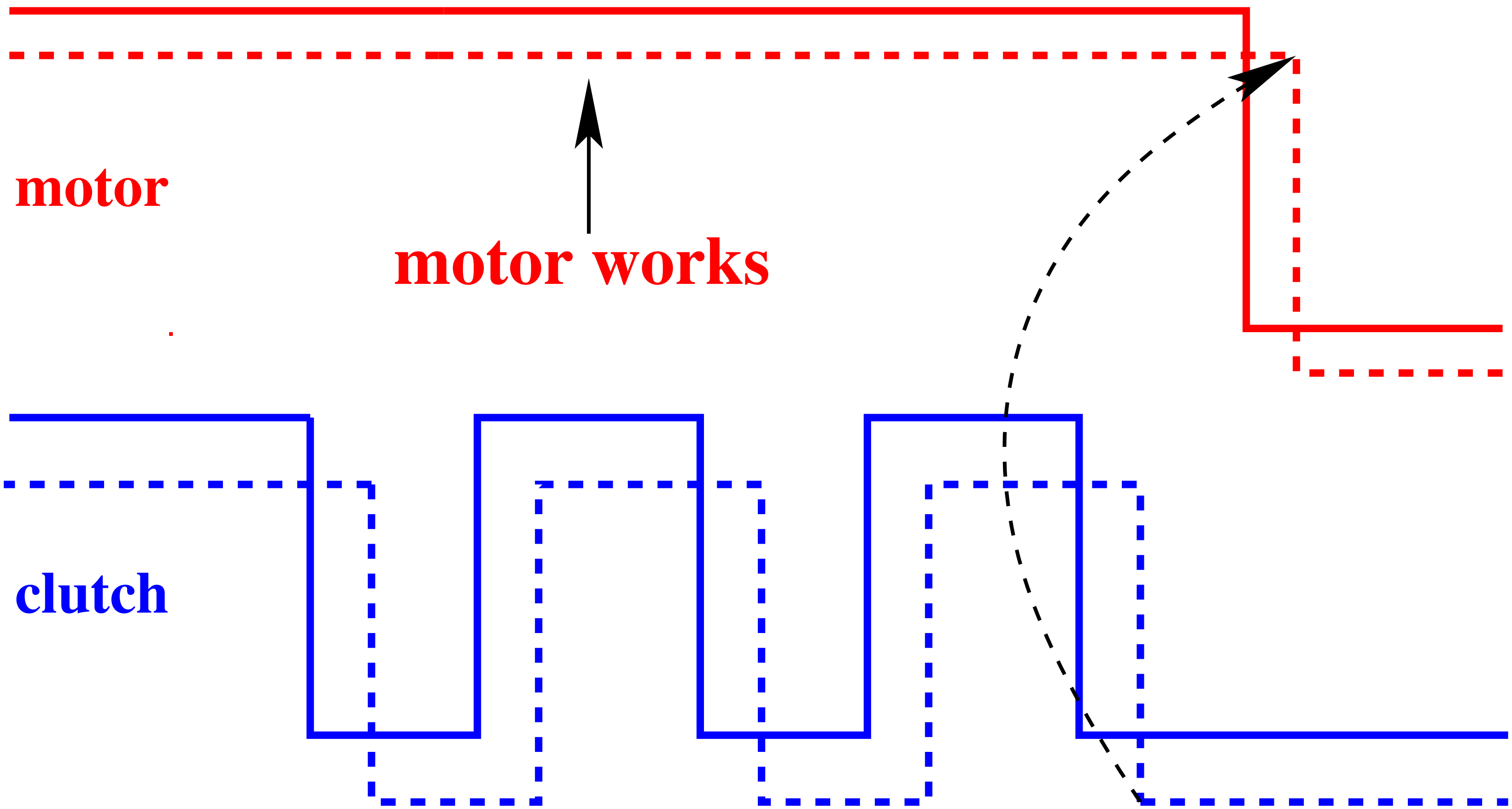


When the clutch is disengaged,

then

the motor can be started and stopped several times

# The Synchronization is Weak (2)

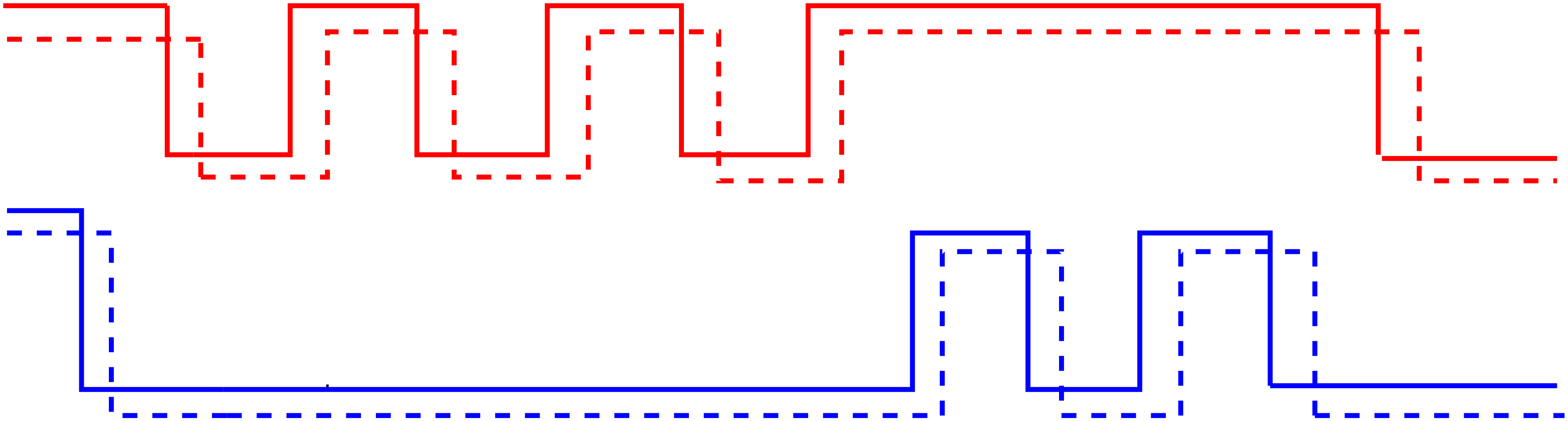


When the motor works,

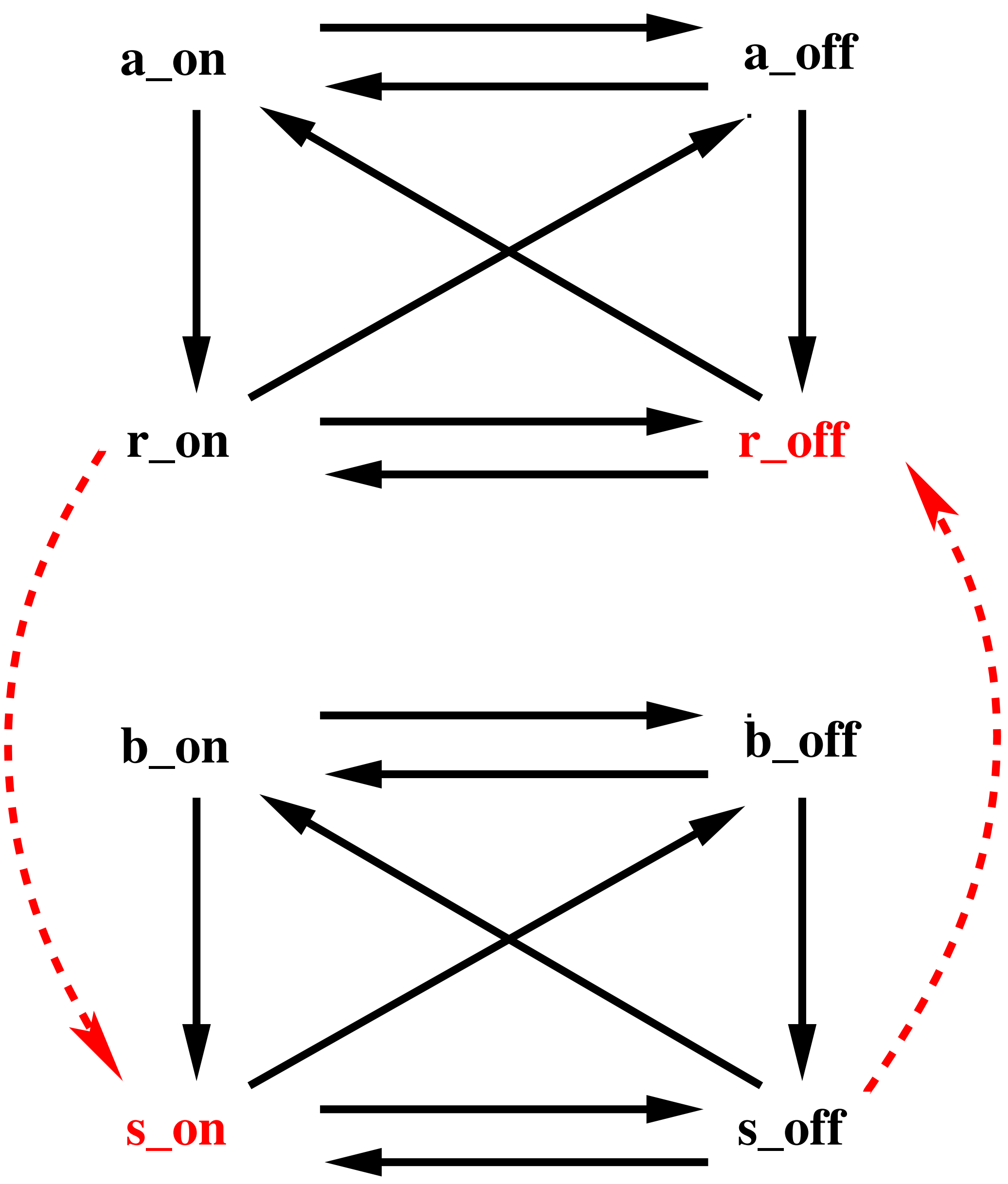
then

the clutch can be engaged and disengaged several times

# Putting the Two Together



# Synchronizing the Reactions Without Touching them



# The Initial State Situation

**dbl0_1:** $a \in \{0,1\}$

**dbl0_2:** $r \in \{0,1\}$

**dbl0_3:** $ca \in \mathbb{N}$

**dbl0_4:** $cr \in \mathbb{N}$

**dbl0_5:** $a = 1 \land r = 0 \Rightarrow ca = cr + 1$

**dbl0_6:** $a = 0 \lor r = 1 \Rightarrow ca = cr$

**dbl0_7:** $b \in \{0,1\}$

**dbl0_8:** $s \in \{0,1\}$

**dbl0_9:** $cb \in \mathbb{N}$

**dbl0_10:** $cs \in \mathbb{N}$

**dbl0_11:** $b = 1 \land s = 0 \Rightarrow cb = cs + 1$

**dbl0_12:** $b = 0 \lor s = 1 \Rightarrow cb = cs$

# The Initial Event Situation (1)

```
a_on
when
   $a = 0$
   $r = 0$
then
   $a := 1$
   $ca := ca + 1$
end
```

```
r_on
when
   $r = 0$
   $a = 1$
then
   $r := 1$
   $cr := cr + 1$
end
```

```
a_off
when
   $a = 1$
   $r = 1$
then
   $a := 0$
end
```

```
r_off
when
   $r = 1$
   $a = 0$
then
   $r := 0$
end
```

# The Initial Event Situation (2)

```
b_on
when
   $b = 0$
   $s = 0$
then
   $b := 1$
   $cb := cb + 1$
end
```

```
s_on
when
   $s = 0$
   $b = 1$
then
   $s := 1$
   $cs := cs + 1$
end
```

```
b_off
when
   $b = 1$
   $s = 1$
then
   $b := 0$
end
```

```
s_off
when
   $s = 1$
   $b = 0$
then
   $s := 0$
end
```

# The Synchronizing Invariant

**dbl1_1:** $s = 1 \quad \Rightarrow \quad r = 1$

- It seems sufficient to add the following guards

```
s_on
  when
    $s = 0$
    $b = 1$
    $r = 1$
  then
    $s := 1$
    $cs := cs + 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
    $s = 0$
  then
    $r := 0$
  end
```

- But we do not want to touch these events

# Introducing Additional Invariants to Remove the red guards

```
s_on
  when
    $s = 0$
    $b = 1$
              $r = 1$
  then
    $s := 1$
    $cs := cs + 1$
  end
```

```
r_off
  when
    $r = 1$
    $a = 0$
              $s = 0$
  then
    $r := 0$
  end
```

- We introduce the following additional invariants

**dbl1_2:** $b = 1 \Rightarrow r = 1$

**dbl1_3:** $a = 0 \Rightarrow s = 0$

# Maintaining Invariant dbl1_2 (1)

**dbl1_2:** $b = 1 \quad \Rightarrow \quad r = 1$

In order to maintain this invariant, we have to refine b_on

```
b_on
when
   $b = 0$
   $s = 0$
then
   $b := 1$
   $cb := cb + 1$
end
```

$\rightsquigarrow$

```
b_on
when
   $b = 0$
   $s = 0$
   $r = 1$
then
   $b := 1$
   $cb := cb + 1$
end
```

# Maintaining (Contraposition of) Invariant dbl1_2 (2)

**dbl1_2:** $b = 1 \Rightarrow r = 1 \qquad (r = 0 \Rightarrow b = 0)$

In order to maintain this invariant, we have to refine r_off

```
r_off
   when
      $r = 1$
      $a = 0$
   then
      $r := 0$
   end
```

$\leadsto$

```
r_off
   when
      $r = 1$
      $a = 0$
      $b = 0$
   then
      $r := 0$
   end
```

- But, again, we do not want to touch this event

# Introducing a new invariant to Remove the Red Guard

```
r_off
  when
    $r = 1$
    $a = 0$
                $b = 0$
  then
    $r := 0$
  end
```

- We introduce the following invariant

**dbl1_4:** $a = 0 \;\Rightarrow\; b = 0$

# Maintaining Invariant dbl1_3 (1)

**dbl1_3:** $a = 0 \quad \Rightarrow \quad s = 0$

In order to maintain this invariant, we have to refine a_off

```
a_off
when
    $a = 1$
    $r = 1$
then
    $a := 0$
end
```

$\rightsquigarrow$

```
a_off
when
    $a = 1$
    $r = 1$
    $s = 0$
then
    $a := 0$
end
```

# Maintaining (Contraposition of) Invariant dbl1_3 (2)

**dbl1_3:** $a = 0 \;\Rightarrow\; s = 0 \qquad (s = 1 \;\Rightarrow\; a = 1)$

In order to maintain this invariant, we have to refine s_on

```
s_on
   when
      $s = 0$
      $b = 1$
   then
      $s := 1$
      $cs := cs + 1$
   end
```

$\leadsto$

```
s_on
   when
      $s = 0$
      $b = 1$
      $a = 1$
   then
      $s := 1$
      $cs := cs + 1$
   end
```

- But, again, we do not want to touch this event

# Introducing a new invariant to Remove the Red Guard

```
s_on
  when
    $s = 0$
    $b = 1$
                $a = 1$
  then
    $s := 1$
    $cs := cs + 1$
  end
```

- We have to introduce the following invariant

$$\boxed{b = 1 \;\Rightarrow\; a = 1}$$

- Fortunately, this is **dbl1_4** ($a = 0 \;\Rightarrow\; b = 0$) contraposed

# Maintaining Invariant dbl1_4 (1)

**dbl1_4:** $a = 0 \quad \Rightarrow \quad b = 0$

In order to maintain this invariant, we have to refine a_off again

```
a_off
when
    $a = 1$
    $r = 1$
    $s = 0$
then
    $a := 0$
end
```

$\leadsto$

```
a_off
when
    $a = 1$
    $r = 1$
    $s = 0$
    $b = 0$
then
    $a := 0$
end
```

# Maintaining (Contraposition of) Invariant dbl1_4 (2)

**dbl1_4:** $a = 0 \;\Rightarrow\; b = 0 \qquad (b = 1 \;\Rightarrow\; a = 1)$

In order to maintain this invariant, we have to refine b_on again

```
b_on
when
   $b = 0$
   $s = 0$
   $r = 1$
then
   $b, cb := 1, cb + 1$
end
```

$\leadsto$

```
b_on
when
   $b = 0$
   $s = 0$
   $r = 1$
   $a = 1$
then
   $b, cb := 1, cb + 1$
end
```

# Summary of Refinement: Reactions have not been Touched

**dbl1_1:** $s = 1 \quad \Rightarrow \quad r = 1$

**dbl1_2:** $b = 1 \quad \Rightarrow \quad r = 1$

**dbl1_3:** $a = 0 \quad \Rightarrow \quad s = 0$

**dbl1_4:** $a = 0 \quad \Rightarrow \quad b = 0$

```
b_on
when
   $b = 0$
   $s = 0$
   $r = 1$
   $a = 1$
then
   $b, cb := 1, cb + 1$
end
```

```
a_off
when
   $a = 1$
   $r = 1$
   $s = 0$
   $b = 0$
then
   $a := 0$
end
```

# Intuition about the Invariants

**dbl1_1:** $s = 1 \quad \Rightarrow \quad r = 1$<br>
**dbl1_2:** $b = 1 \quad \Rightarrow \quad r = 1$<br>
**dbl1_3:** $a = 0 \quad \Rightarrow \quad s = 0$ $\qquad (s = 1 \quad \Rightarrow \quad a = 1)$<br>
**dbl1_4:** $a = 0 \quad \Rightarrow \quad b = 0$ $\qquad (b = 1 \quad \Rightarrow \quad a = 1)$

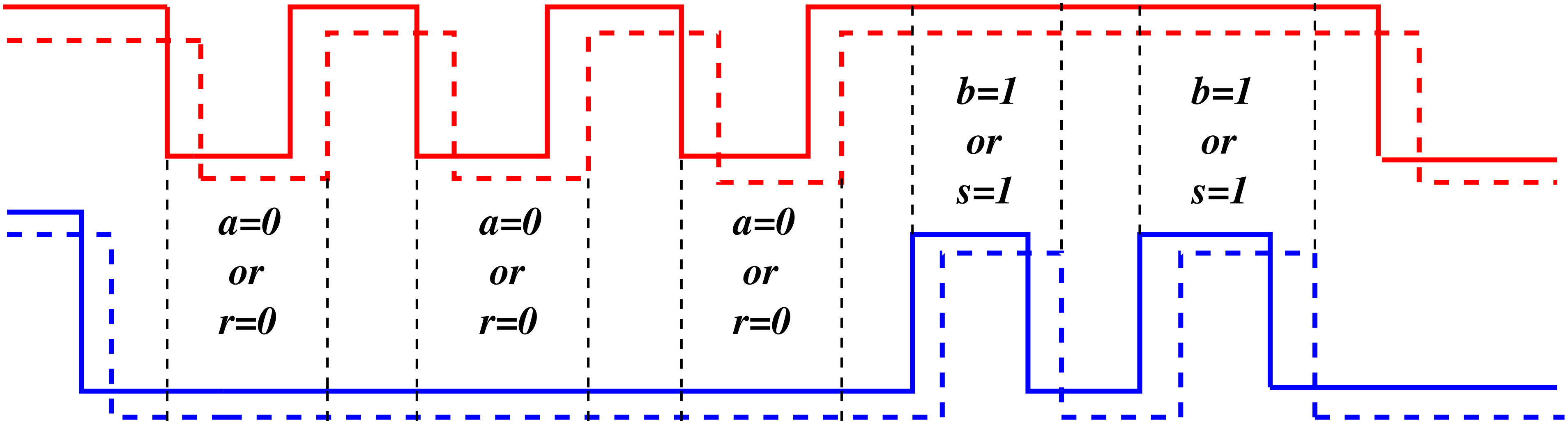
This can be put into a single invariant

**dbl1_5:** $b = 1 \lor s = 1 \quad \Rightarrow \quad a = 1 \land r = 1$

with the following contraposed form

**dbl1_6:** $a = 0 \lor r = 0 \quad \Rightarrow \quad b = 0 \land s = 0$

# Intuition about the Invariants



Reminder: - - - is the motor and - - - is the clutch

**dbl1_5:** $b = 1 \ \vee \ s = 1 \ \Rightarrow \ a = 1 \ \wedge \ r = 1$

**dbl1_6:** $a = 0 \ \vee \ r = 0 \ \Rightarrow \ b = 0 \ \wedge \ s = 0$

# Looking at the Result

```
a_on
when
    $a = 0$
    $r = 0$
then
    $a := 1$
end
```

```
a_off
when
    $a = 1$
    $r = 1$
    $s = 0$
    $b = 0$
then
    $a := 0$
end
```

```
r_on
when
    $r = 0$
    $a = 1$
then
    $r := 1$
end
```

```
r_off
when
    $r = 1$
    $a = 0$
then
    $r := 0$
end
```

```
b_on
when
    $b = 0$
    $s = 0$
    $r = 1$
    $a = 1$
then
    $b := 1$
end
```

```
b_off
when
    $b = 1$
    $s = 1$
then
    $b := 0$
end
```

```
s_on
when
    $s = 0$
    $b = 1$
then
    $s := 1$
end
```

```
s_off
when
    $s = 1$
    $b = 0$
then
    $s := 0$
end
```

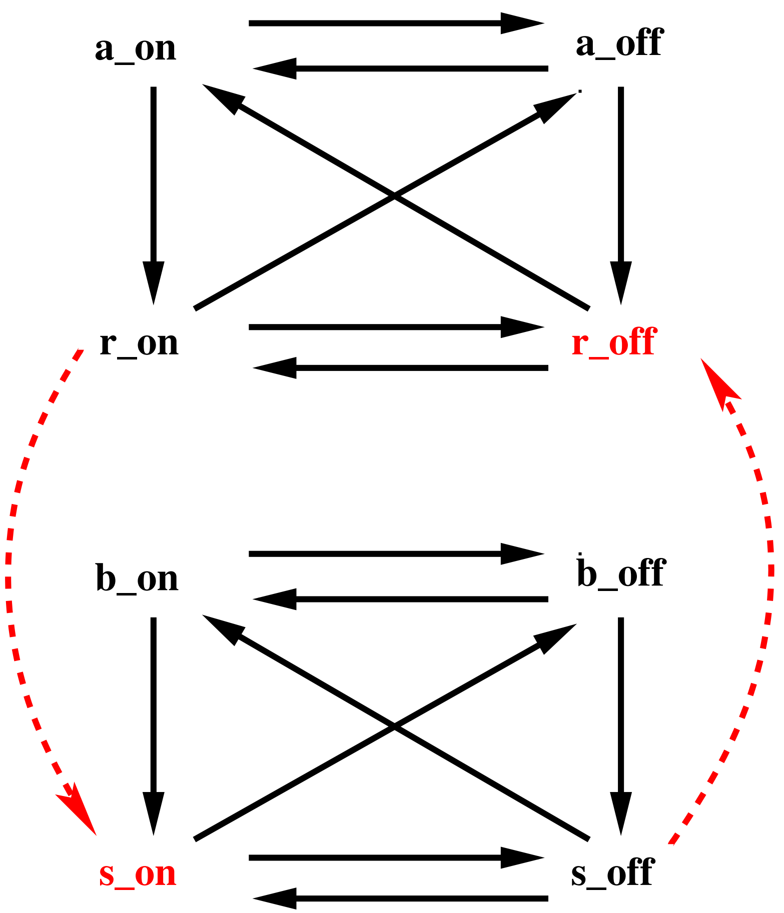
**dbl1_1:** $s = 1 \;\Rightarrow\; r = 1 \qquad (r = 0 \;\Rightarrow\; s = 0)$

**dbl1_2:** $b = 1 \;\Rightarrow\; r = 1 \qquad (r = 0 \;\Rightarrow\; b = 0)$

**dbl1_3:** $a = 0 \;\Rightarrow\; s = 0 \qquad (s = 1 \;\Rightarrow\; a = 1)$

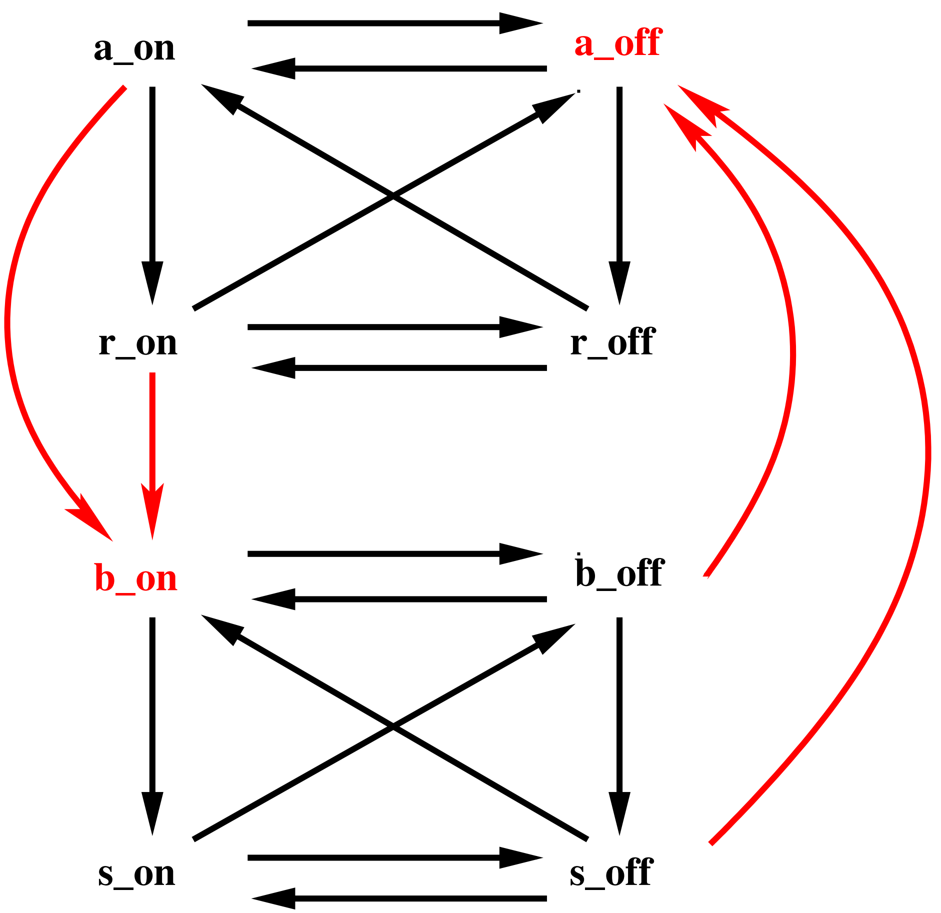
**dbl1_4:** $a = 0 \;\Rightarrow\; b = 0 \qquad (b = 1 \;\Rightarrow\; a = 1)$

# Weak Synchronization of Strong Reactions: the Problem



**dbl1_1:** $s = 1 \;\Rightarrow\; r = 1 \qquad (r = 0 \;\Rightarrow\; s = 0)$

# Weak Synchronization of Strong Reactions: the Solution



```
b_on
when
    $b = 0$
    $s = 0$
    $r = 1$
    $a = 1$
then
    $b := 1$
end
```

```
a_off
when
    $a = 1$
    $r = 1$
    $s = 0$
    $b = 0$
then
    $a := 0$
end
```

# Back to 3rd Reft.: Constraining the Clutch and the Motor

| | |
|---|---|
| When the clutch is engaged, the motor must work | SAF_1 |

**inv3_1:** $$clutch\_sensor = engaged \\ \Rightarrow \\ motor\_sensor = working$$

- This is an instance of the previous design pattern

# Pattern Instantiation

- We instantiate the pattern as follows:

| | | | | | |
|---|---|---|---|---|---|
| $a$ | $\rightsquigarrow$ | $\mathit{motor\_actuator}$ | a_on | $\rightsquigarrow$ | treat_push_start_motor_button |
| $r$ | $\rightsquigarrow$ | $\mathit{motor\_sensor}$ | a_off | $\rightsquigarrow$ | treat_push_stop_motor_button |
| $0$ | $\rightsquigarrow$ | $\mathit{stopped}$ | r_on | $\rightsquigarrow$ | Motor_start |
| $1$ | $\rightsquigarrow$ | $\mathit{working}$ | r_off | $\rightsquigarrow$ | Motor_stop |

| | | | | | |
|---|---|---|---|---|---|
| $b$ | $\rightsquigarrow$ | $\mathit{clutch\_actuator}$ | b_on | $\rightsquigarrow$ | treat_start_clutch |
| $s$ | $\rightsquigarrow$ | $\mathit{clutch\_sensor}$ | b_off | $\rightsquigarrow$ | treat_stop_clutch |
| $0$ | $\rightsquigarrow$ | $\mathit{disengaged}$ | s_on | $\rightsquigarrow$ | Clutch_start |
| $1$ | $\rightsquigarrow$ | $\mathit{engaged}$ | s_off | $\rightsquigarrow$ | Clutch_stop |

# Translating the pattern invariants (1)

**dbl1_1:** $s = 1 \quad \Rightarrow \quad r = 1$

**dbl1_2:** $b = 1 \quad \Rightarrow \quad r = 1$

**inv3_1:**

$$\begin{array}{l} clutch\_sensor = engaged \\ \Rightarrow \\ motor\_sensor = working \end{array}$$

**inv3_2:**

$$\begin{array}{l} clutch\_actuator = engaged \\ \Rightarrow \\ motor\_sensor = working \end{array}$$

# Translating the pattern invariants (2)

**dbl1_3:** $a = 0 \quad \Rightarrow \quad s = 0$

**dbl1_4:** $a = 0 \quad \Rightarrow \quad b = 0$

**inv3_3:**

$$\begin{array}{l} motor\_actuator = stopped \\ \Rightarrow \\ clutch\_sensor = disengaged \end{array}$$

**inv3_4:**

$$\begin{array}{l} motor\_actuator = stopped \\ \Rightarrow \\ clutch\_actuator = disengaged \end{array}$$

# Adapting the Events of the Pattern (1)

```
b_on
when
    $b = 0$
    $s = 0$
    $r = 1$
    $a = 1$
then
    $b := 1$
end
```

```
treat_start_clutch
    when
        $clutch\_actuator = disengaged$
        $clutch\_sensor = disengaged$
        $motor\_sensor = working$
        $motor\_actuator = working$
    then
        $clutch\_actuator := engaged$
    end
```

# Adapting the events of the pattern (2)

```
a_off
when



    $a = 1$
    $r = 1$
    $s = 0$
    $b = 0$
then
    $a := 0$

end
```

```
treat_push_stop_motor_button
    when
        $stop\_motor\_impulse = \mathrm{FALSE}$
        $stop\_motor\_button = \mathrm{TRUE}$
        $motor\_actuator = working$
        $motor\_sensor = working$
        $clutch\_sensor = disengaged$
        $clutch\_actuator = disengaged$
    then
        $motor\_actuator := stopped$
        $stop\_motor\_impulse := \mathrm{TRUE}$
    end
```

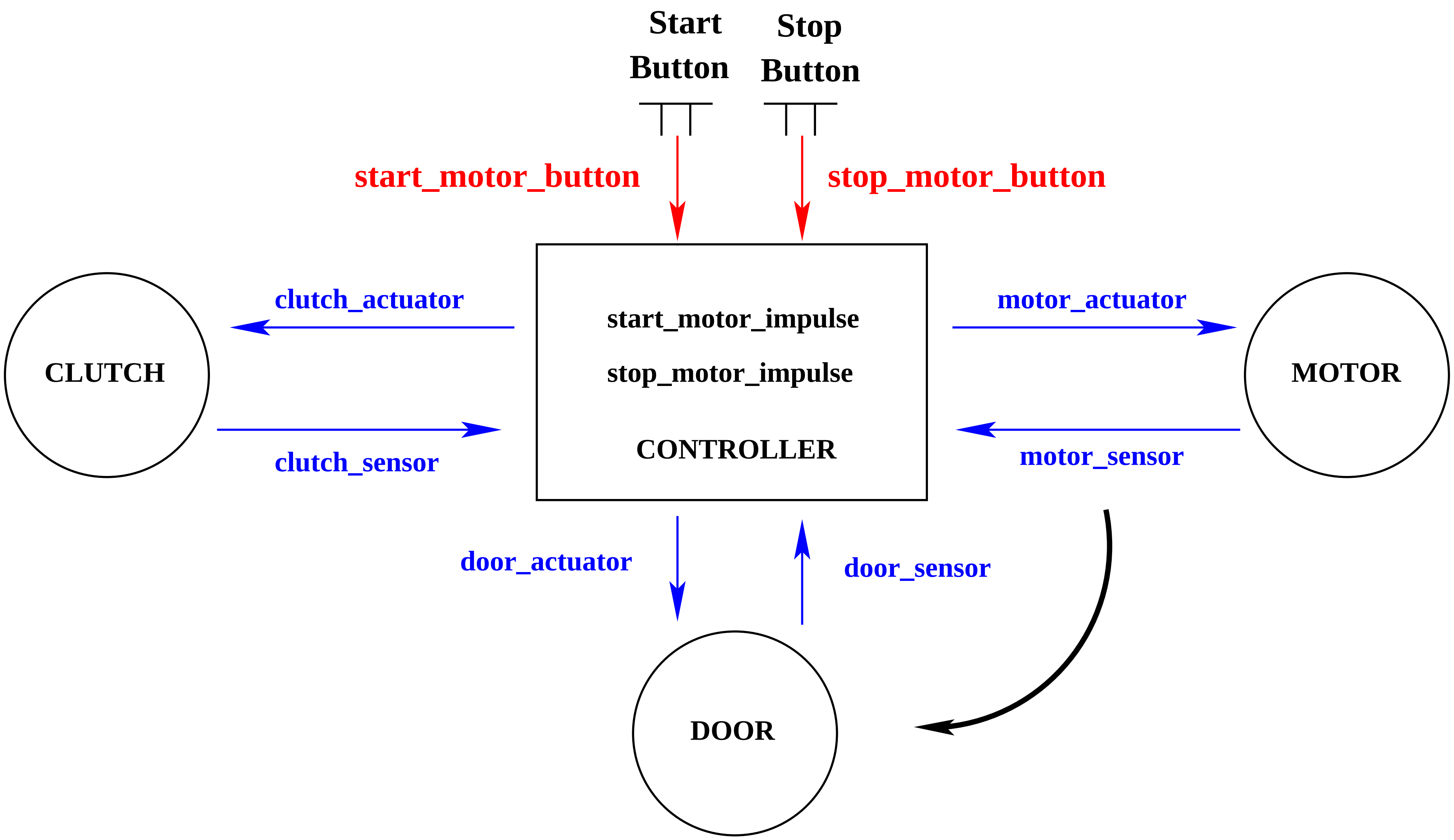
# Third Refinement: Summary of the Events (1)

- Environment (no new events)
    - motor_start
    - motor_stop
    - clutch_start
    - clutch_stop
    - push_start_motor_button
    - release_start_motor_button
    - push_stop_motor_button
    - release_stop_motor_button

# Third Refinement: Summary of the Events (2)

- Controller (no new events)
    - treat_push_start_motor_button
    - treat_push_start_motor_button_false
    - treat_push_stop_motor_button
    - treat_push_stop_motor_button_false
    - treat_release_start_motor_button
    - treat_release_stop_motor_button
    - treat_start_clutch
    - treat_stop_clutch

# Fourth Refinement: Connecting the Controller to the Door



- We copy (after renaming "motor" to "door") what has been done in the initial model

# Fourth Refinement: Connecting the Controller to the Door

- We introduce the set in a new context:

$$DOOR = \{open, closed\}$$

- We copy the initial model where we instantiate:

$$\begin{array}{lcl} motor & \leadsto & door \\ STATUS & \leadsto & DOOR \\ working & \leadsto & closed \\ stopped & \leadsto & open \end{array}$$

# Fourth Refinement: Summary of the Events (1)

- Environment
    - motor_start
    - motor_stop
    - clutch_start
    - clutch_stop
    - door_close
    - door_open
    - push_start_motor_button
    - release_start_motor_button
    - push_stop_motor_button
    - release_stop_motor_button

# Fourth Refinement: Summary of the Events (2)

- Controller
    - treat_push_start_motor_button
    - treat_push_start_motor_button_false
    - treat_push_stop_motor_button
    - treat_push_stop_motor_button_false
    - treat_release_start_motor_button
    - treat_release_stop_motor_button
    - treat_start_clutch
    - treat_stop_clutch
    - treat_close_door
    - treat_open_door

# Fifth Refinement: Constraining the Clutch and the Door

- An additional safety constraint

| | |
|---|---|
| When the clutch is engaged, the door must be closed | SAF_2 |

- We copy (after renaming "motor" to "door") what has been done in the third model:

| | |
|---|---|
| When the clutch is engaged, the motor must work | SAF_1 |

# Fifth Reft.: Something was forgotten Concerning the Door

# Fifth Reft.: Something was forgotten Concerning the Door

- Can you guess it?

# Fifth Reft.: Something was forgotten Concerning the Door

- Can you guess it?

- When the motor is not working, we must allow users:
    - to change the tool
    - to replace the part to be treated

# Fifth Reft.: Something was forgotten Concerning the Door

- Can you guess it?

- When the motor is not working, we must allow users:
  - to change the tool
  - to replace the part to be treated

- Hence the following additional requirement (which was forgotten)

| | |
|---|---|
| When the motor is stopped, the door must be open | SAF_3 |

# Fifth Reft.: Something was forgotten Concerning the Door

- Can you guess it?

- When the motor is not working, we must allow users:
  - to change the tool
  - to replace the part to be treated

- Hence the following additional requirement (which was forgotten)

| | |
|---|---|
| When the door is closed, the motor must work | SAF_3’ |

- SAF_3’ is the contraposed form of SAF_3

# Fifth Refinement: Taking Care of the New Constraint

- Additional safety constraint

| | |
|---|---|
| When the door is closed, the motor must work | SAF_3’ |

- We copy (after renaming "clutch" to "door") what has been done in the third model:

| | |
|---|---|
| When the clutch is engaged, the motor must work | SAF_1 |

# Fifth Reft.: Summary of the Safety Requirements

| | |
|---|---|
| When the clutch is engaged, the motor must work | SAF_1 |

| | |
|---|---|
| When the clutch is engaged, the door must be closed | SAF_2 |

| | |
|---|---|
| When the door is closed, the motor must work | SAF_3’ |

- Requirement SAF_1 is now redundant: SAF_2 ∧ SAF_3’ ⇒ SAF_1

# Possible New Refinement Strategy

- Initial model: Connecting the controller to the motor

- 1st refinement: Connecting the motor button to the controller

- 2nd refinement: Connecting the controller to the clutch

- 3rd (4th) refinement: Connecting the controller to the door

# Possible New Refinement Strategy (cont'd)

- 4th (5th) refinement: Constraining the clutch and the door

  Constraining the motor and the door

- 5th (6th) refinement: More constraints between clutch and door

- 6th (7th) refinement: Connecting the clutch button to the controller

# Fifth Refinement: Summary of the Events (1)

- Environment (no new events)
    - motor_start
    - motor_stop
    - clutch_start
    - clutch_stop
    - door_close
    - door_open
    - push_start_motor_button
    - release_start_motor_button
    - push_stop_motor_button
    - release_stop_motor_button

# Fifth Refinement: Summary of the Events (2)

- Controller (no new events)
    - treat_push_start_motor_button
    - treat_push_start_motor_button_false
    - treat_push_stop_motor_button
    - treat_push_stop_motor_button_false
    - treat_release_start_motor_button
    - treat_release_stop_motor_button
    - treat_start_clutch
    - treat_stop_clutch
    - treat_close_door
    - treat_open_door

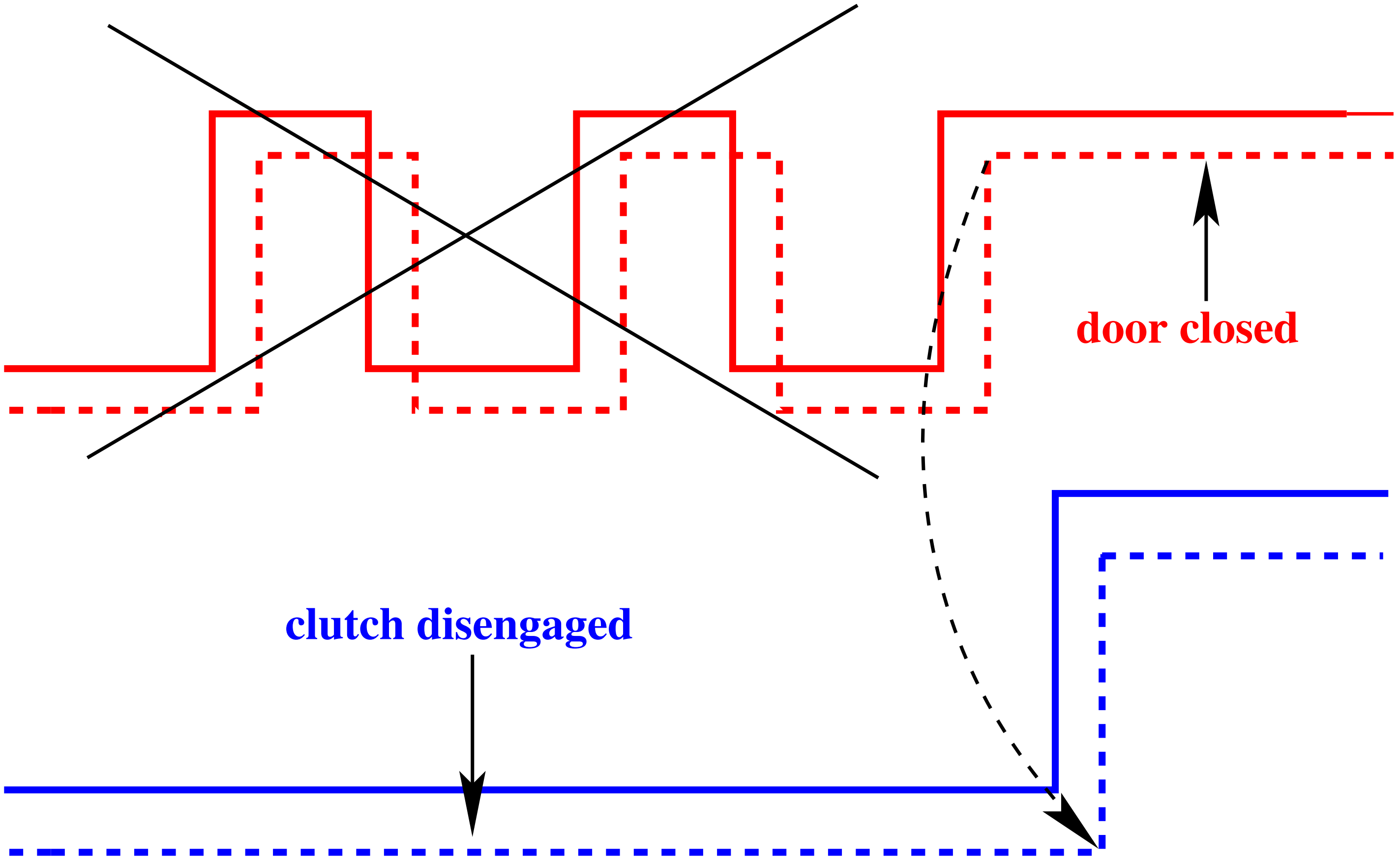
# Sixth Reft.: More Constraints between Clutch and Door

- Adding two functional constraints

| | |
|---|---|
| When the clutch is disengaged, the door cannot be closed several times, ONLY ONCE | FUN_3 |

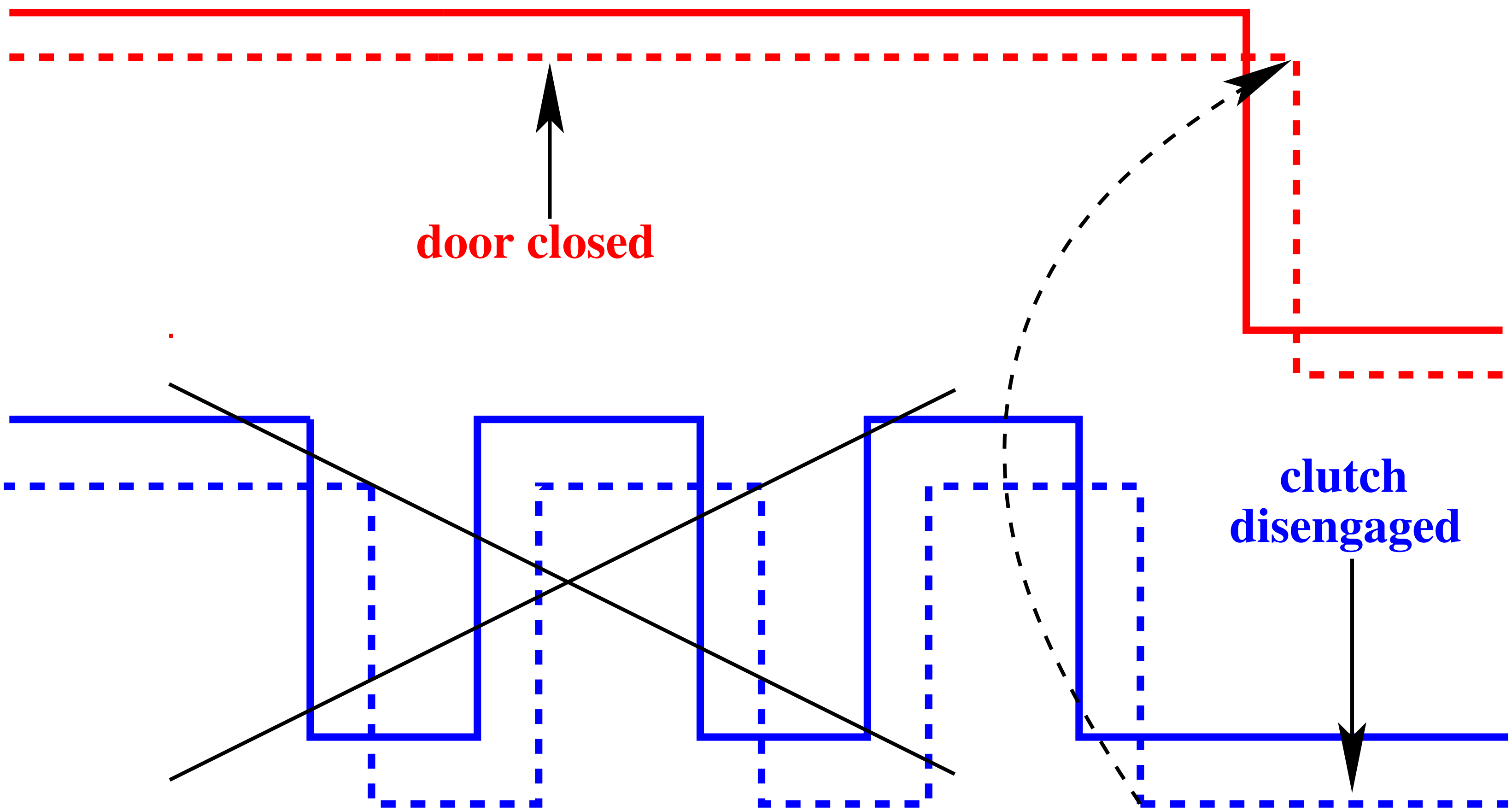
| | |
|---|---|
| When the door is closed, the clutch cannot be disengaged several times, ONLY ONCE | FUN_4 |

# Problem with the Weak Synchronization of Strong Reactions



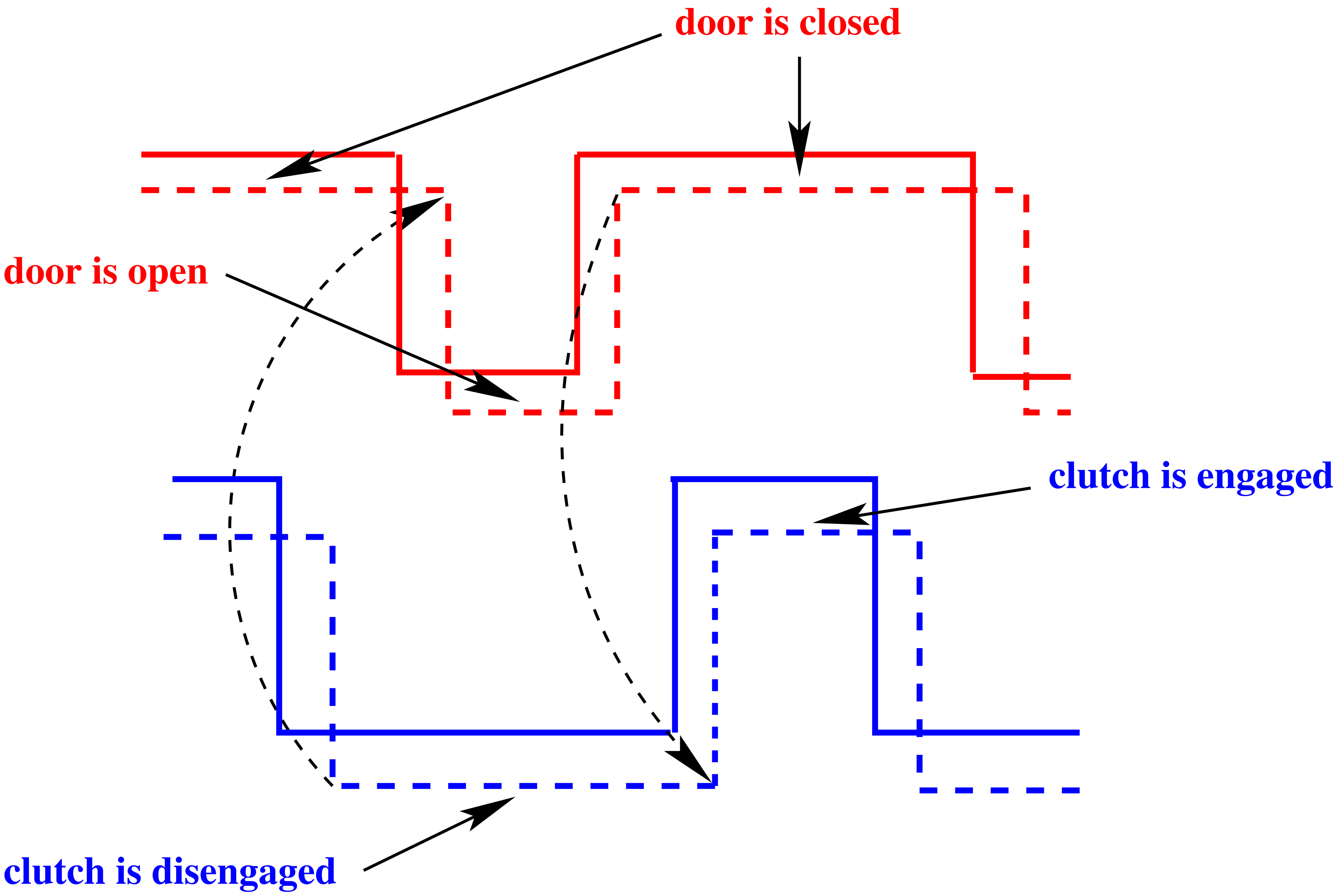
- When the clutch is disengaged, the door cannot be closed several times

# Problem with the Weak Synchronization of Strong Reactions

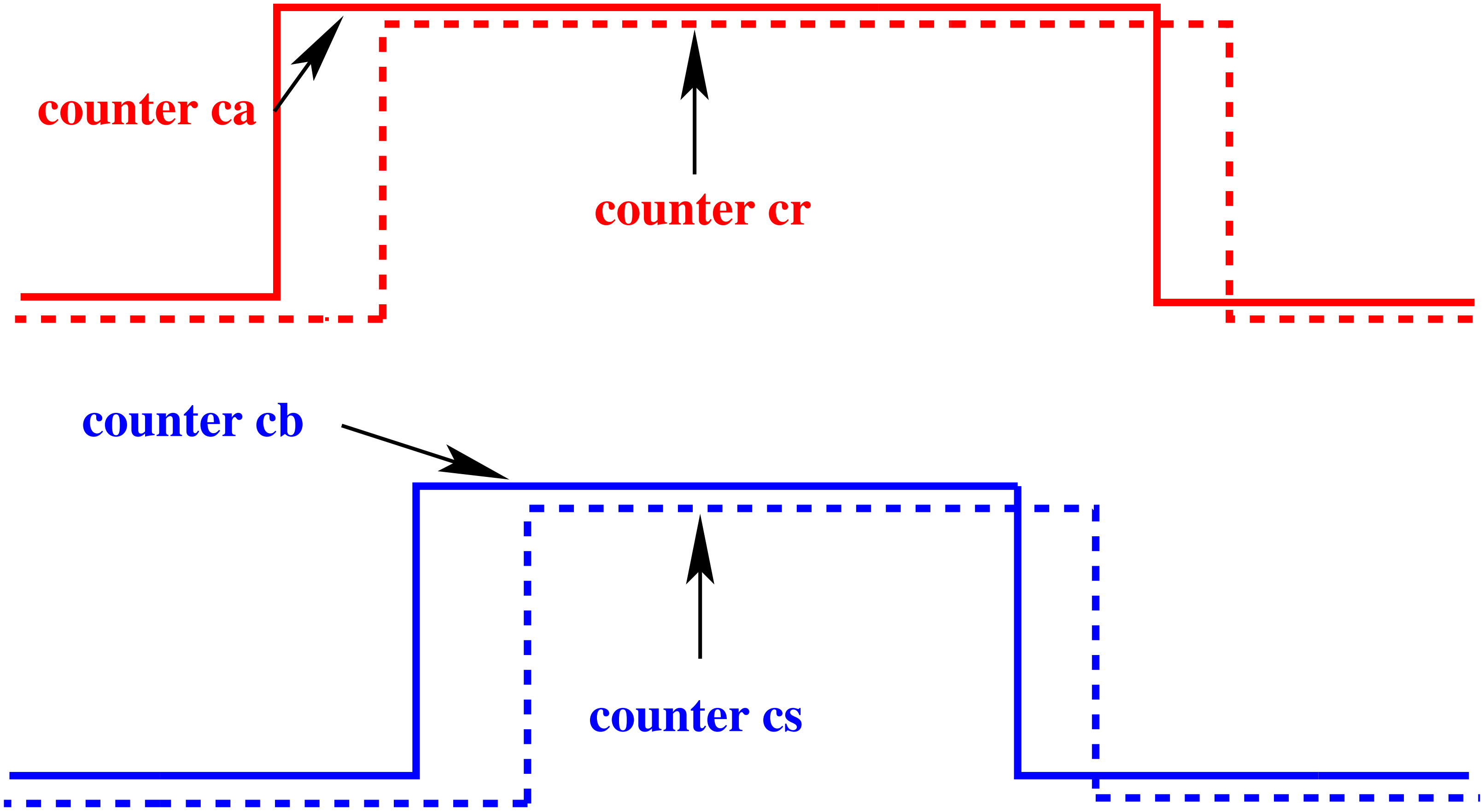


- When the door is closed, the clutch cannot be disengaged several times

# The Full Picture



# Strong Synchronization of two Strong Reactions

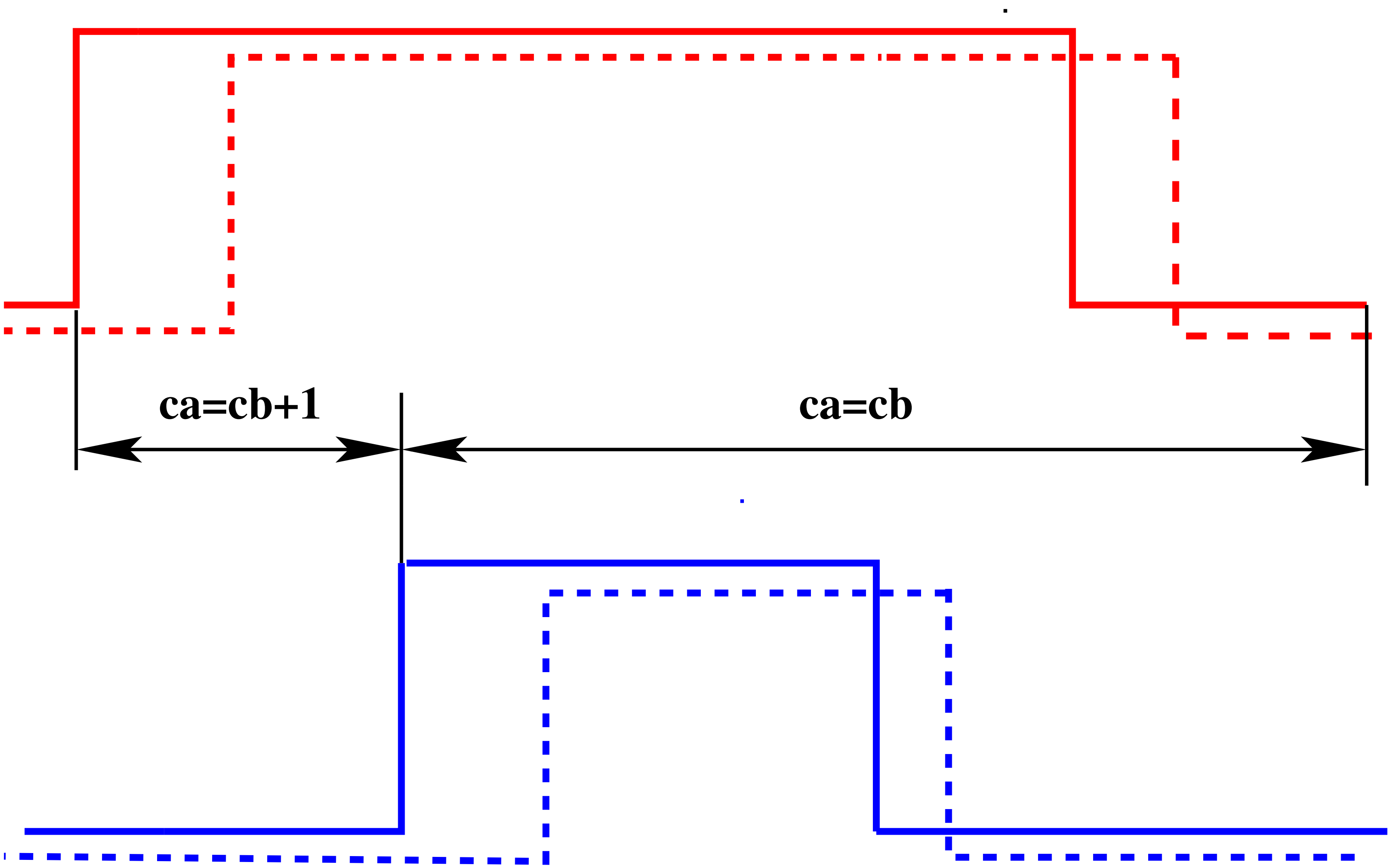


What we want:

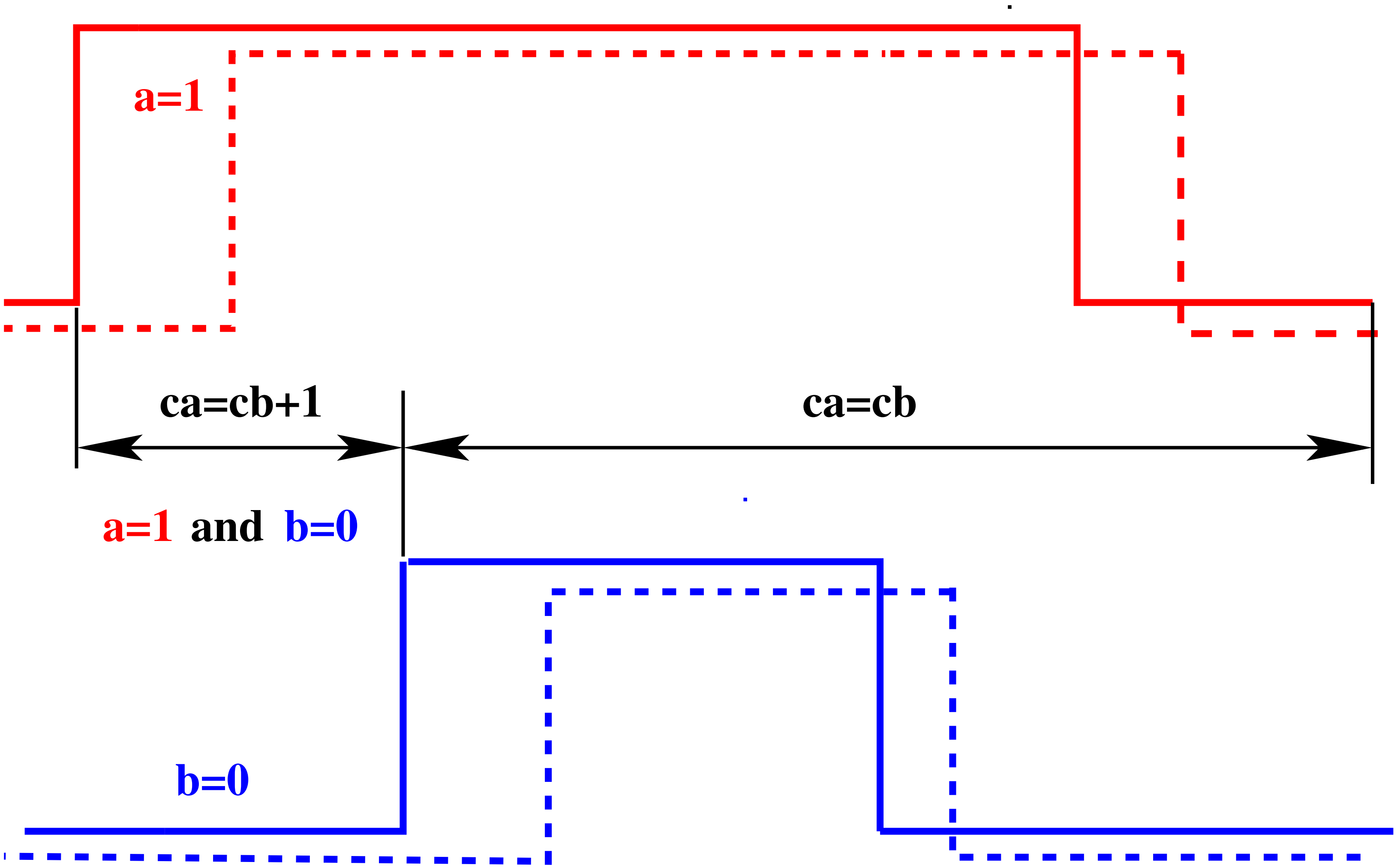
$$ca = cb \ \lor \ ca = cb + 1$$

$$cr = cs \ \lor \ cr = cs + 1$$

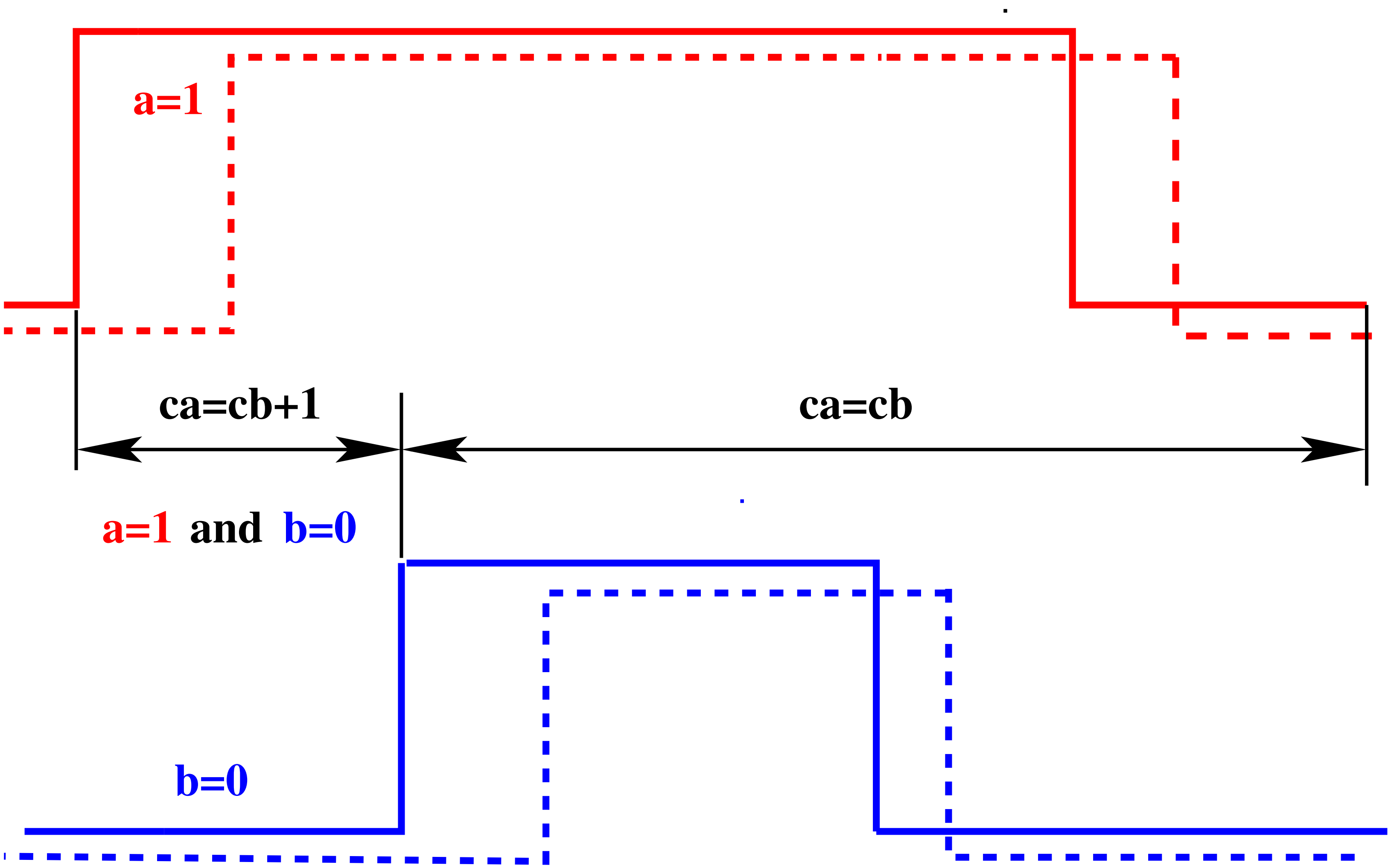
# How about counters $ca$ and $cb$?



# In Search of a Solution

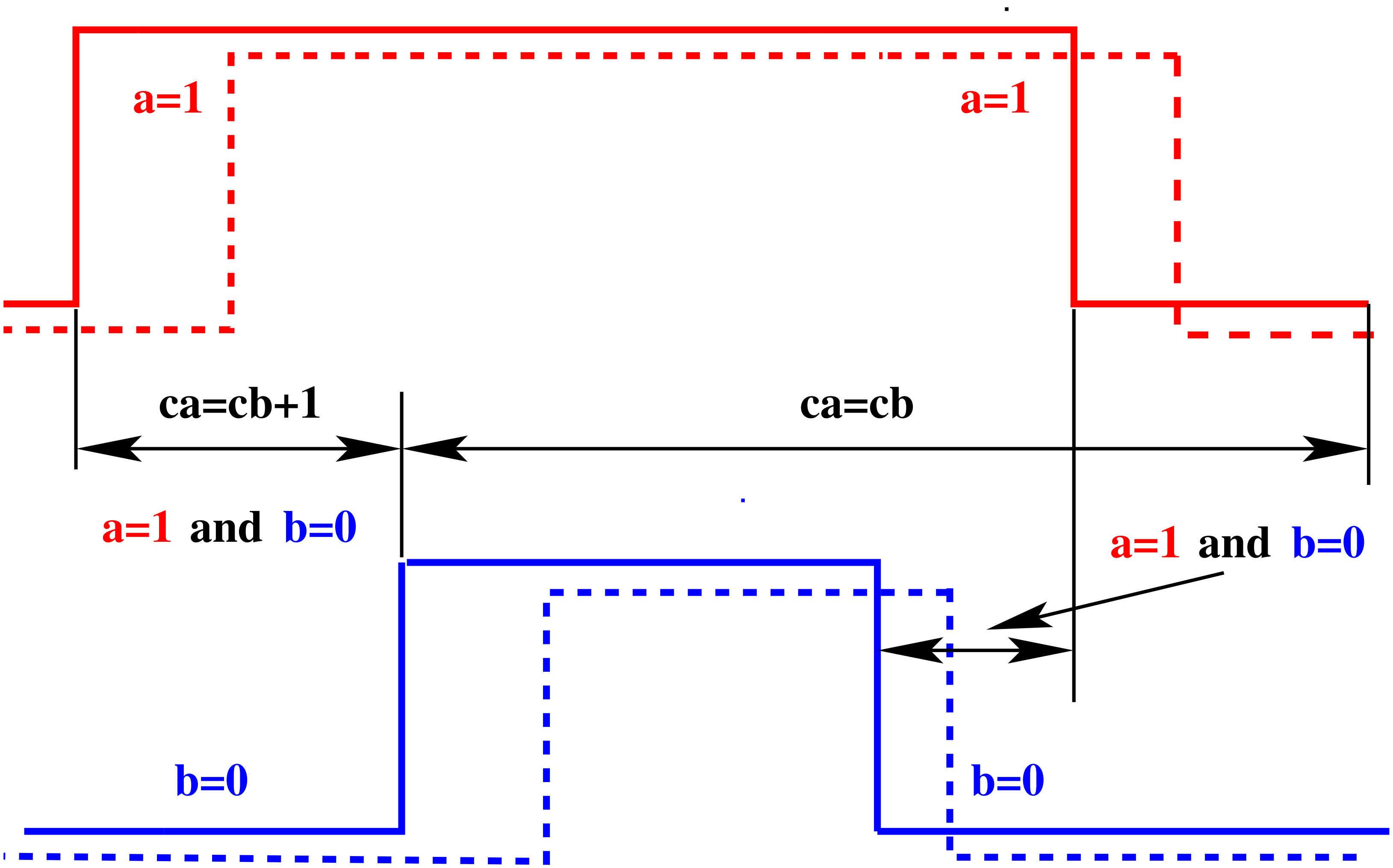


# In Search of a Solution

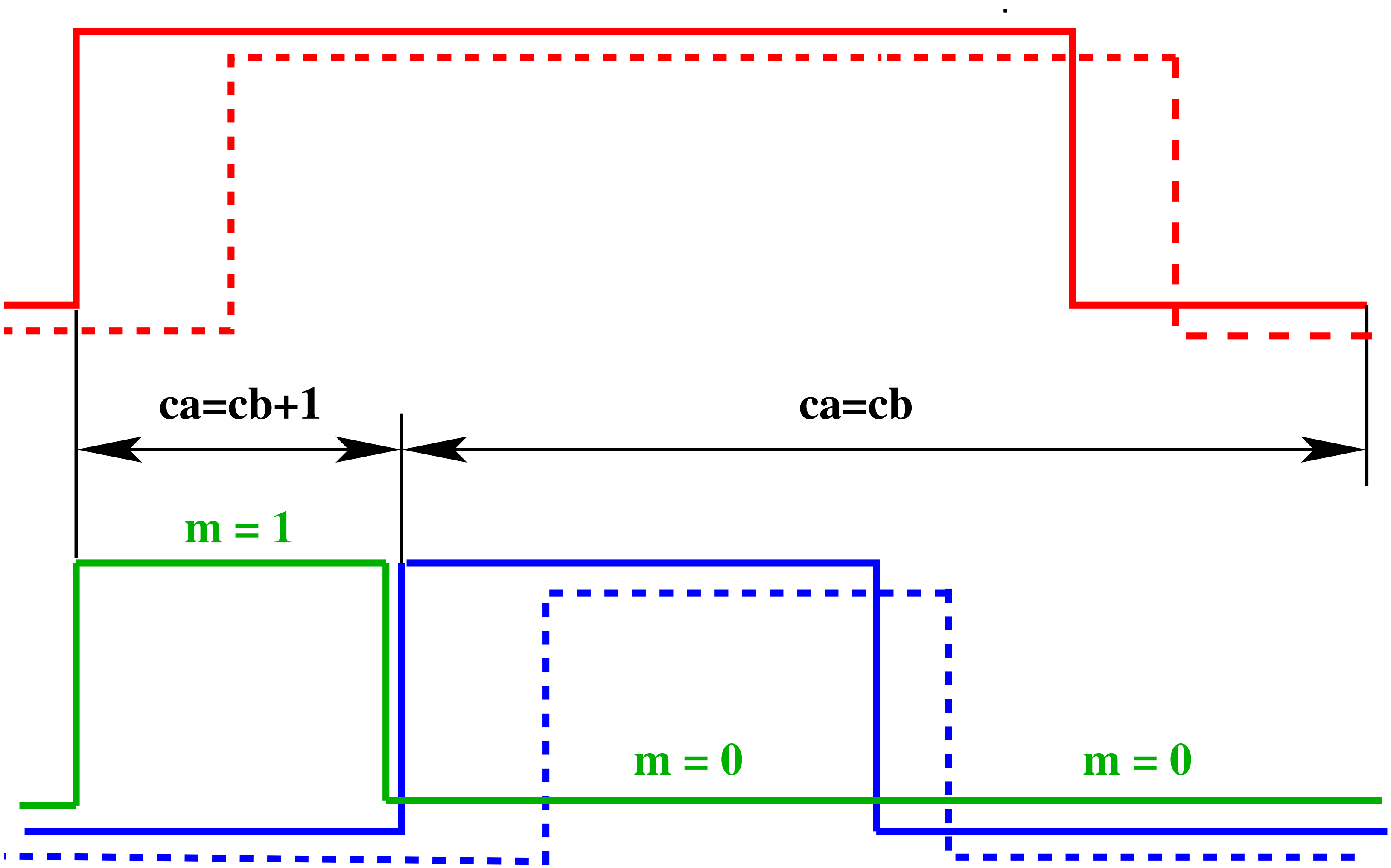


$$a = 1 \ \wedge \ b = 0 \quad \Rightarrow \quad ca = cb + 1 \qquad \textbf{?}$$

# This Solution Does not Work

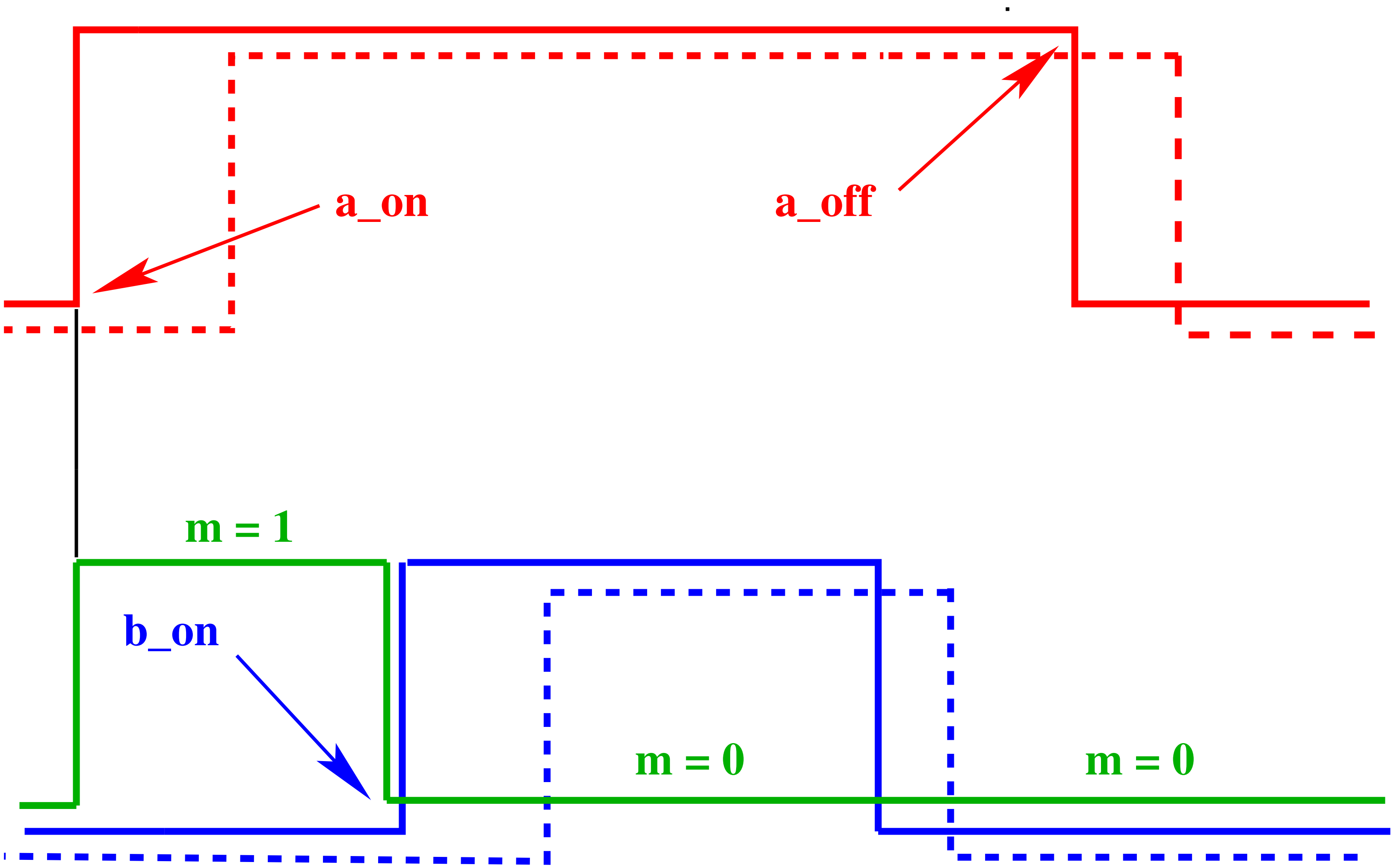


# The Solution: an Additional Variable $m$



$$m = 1 \quad \Rightarrow \quad ca = cb + 1$$
$$m = 0 \quad \Rightarrow \quad ca = cb$$

# The Events

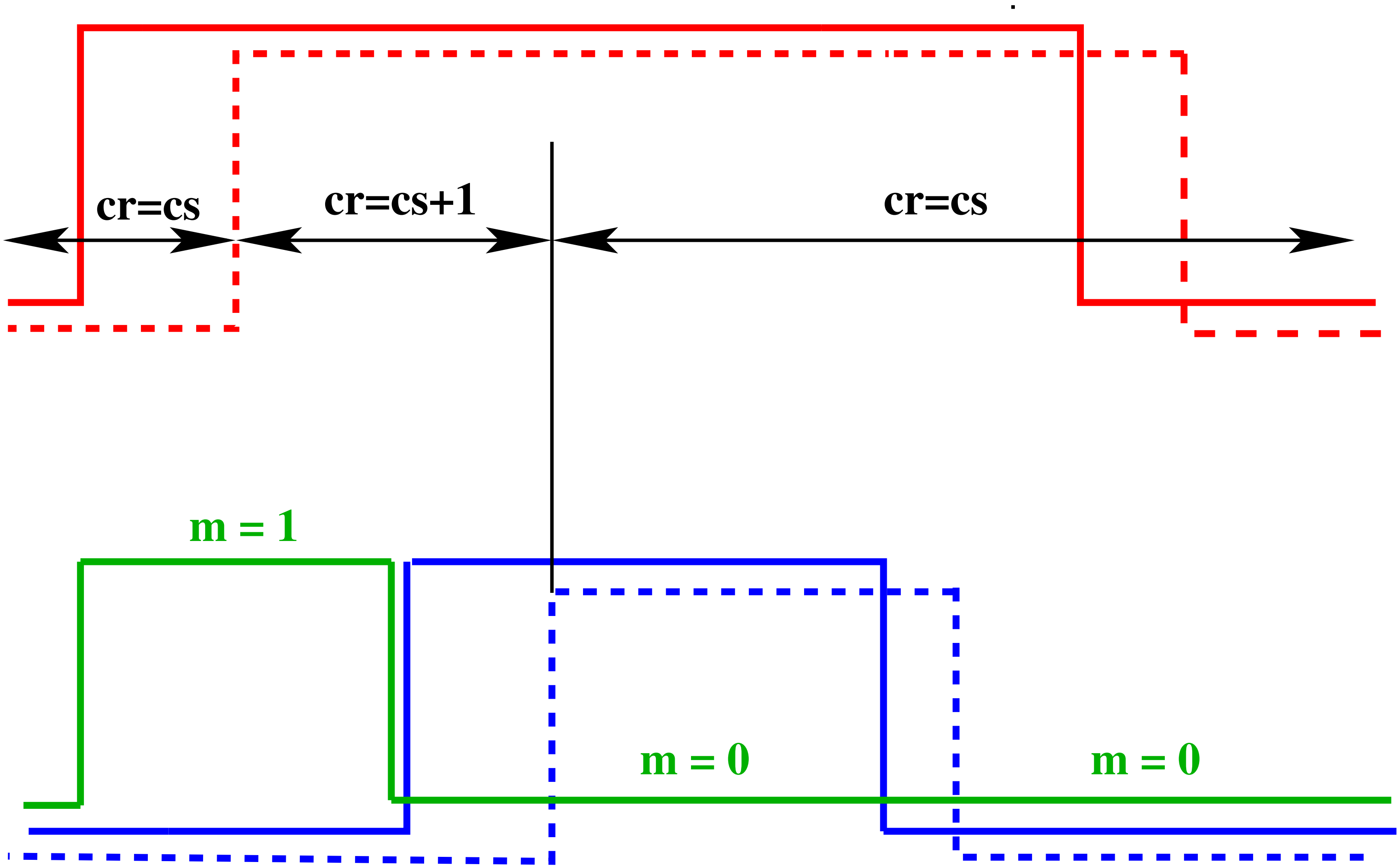


# The Modified Events

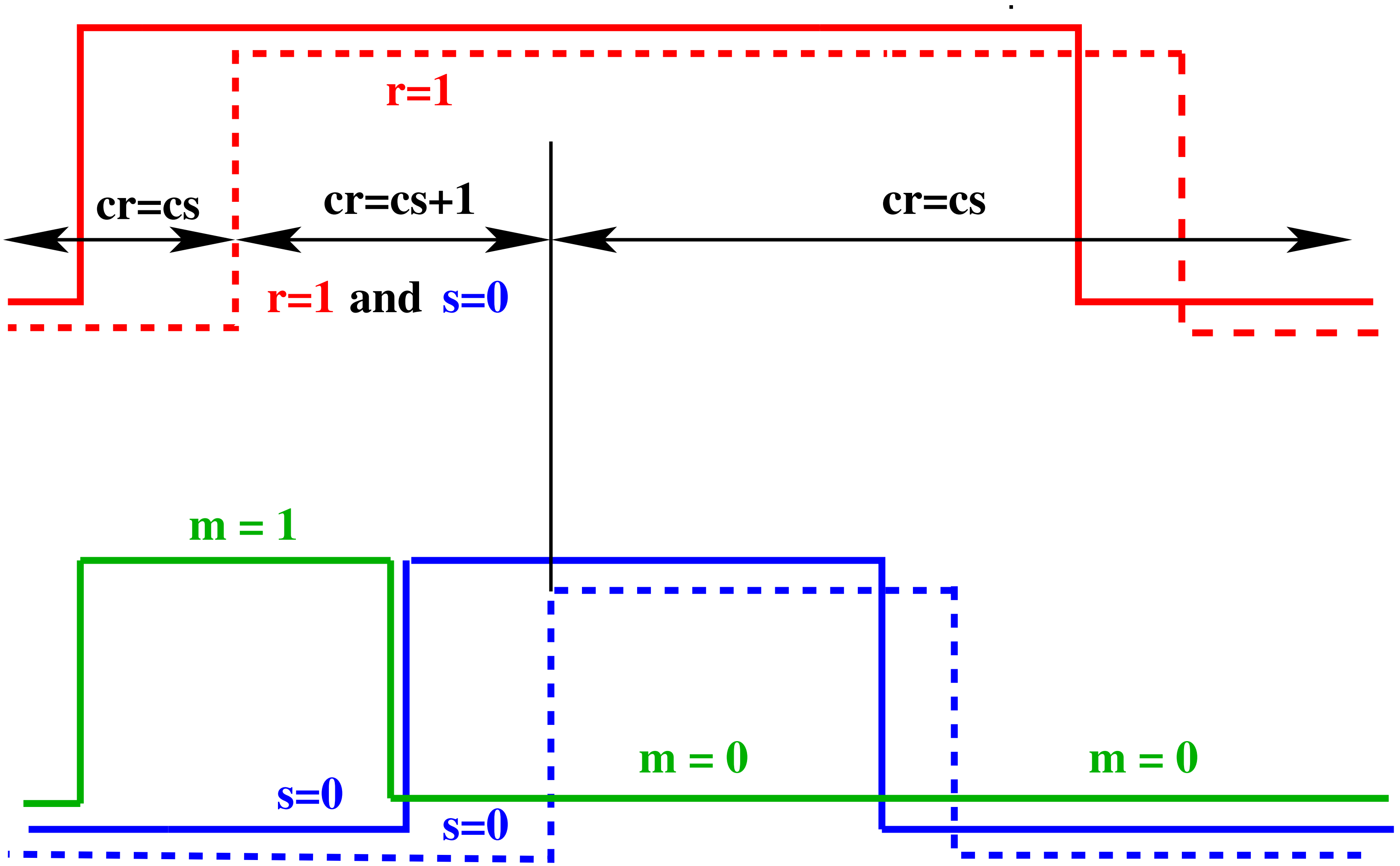
```
a_on
when
    $a = 0$
    $r = 0$
then
    $a := 1$
    $ca := ca + 1$
    $m := 1$
end
```

```
b_on
when
    $r = 1$
    $a = 1$
    $b = 0$
    $s = 0$
    $m = 1$
then
    $b := 1$
    $cb := cb + 1$
    $m := 0$
end
```

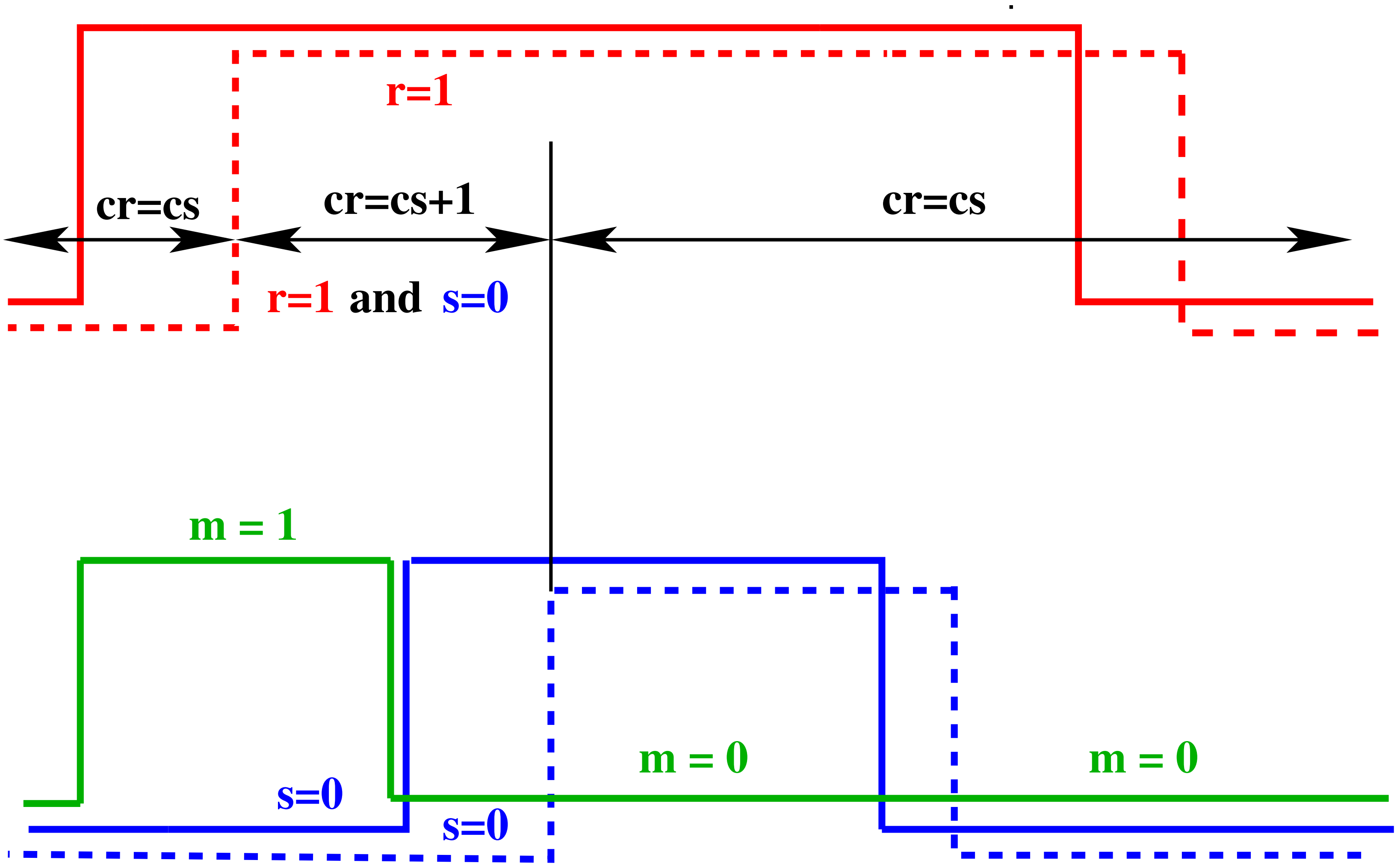
# How about counters $cr$ and $cs$



# In Search of a Solution

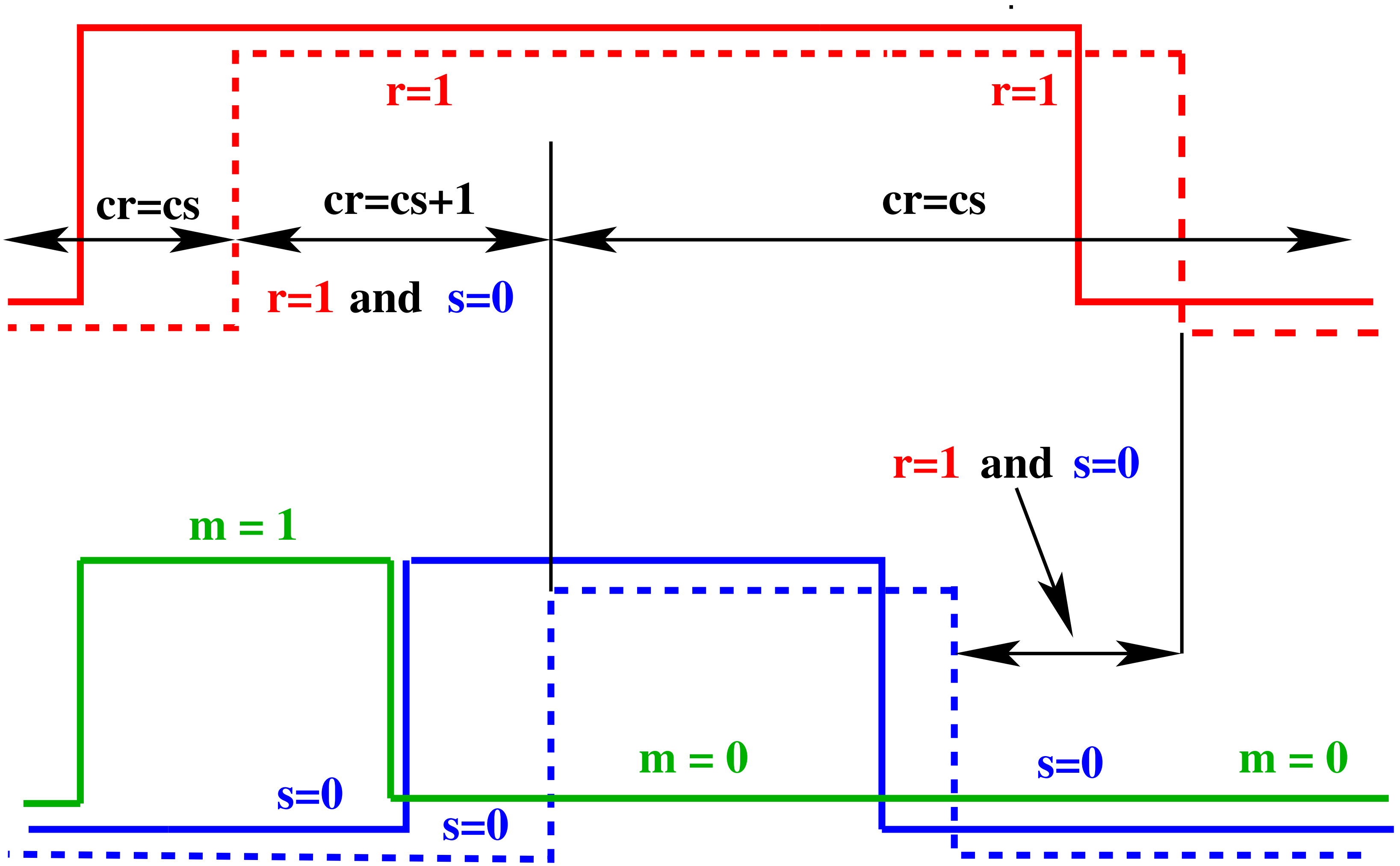


# In Search of a Solution

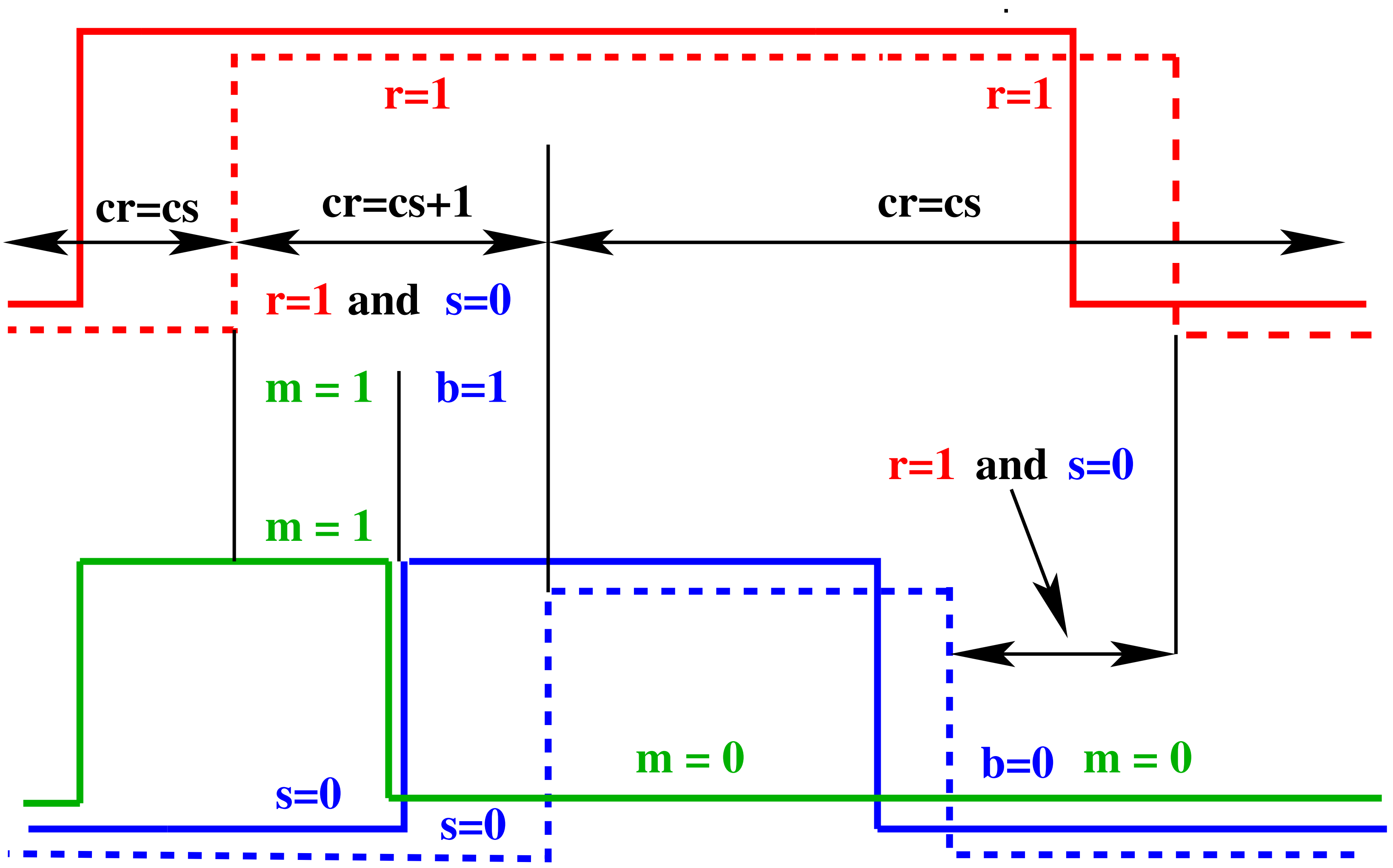


$$r = 1 \;\wedge\; s = 0 \;\;\Rightarrow\;\; cr = cs + 1 \qquad \text{?}$$

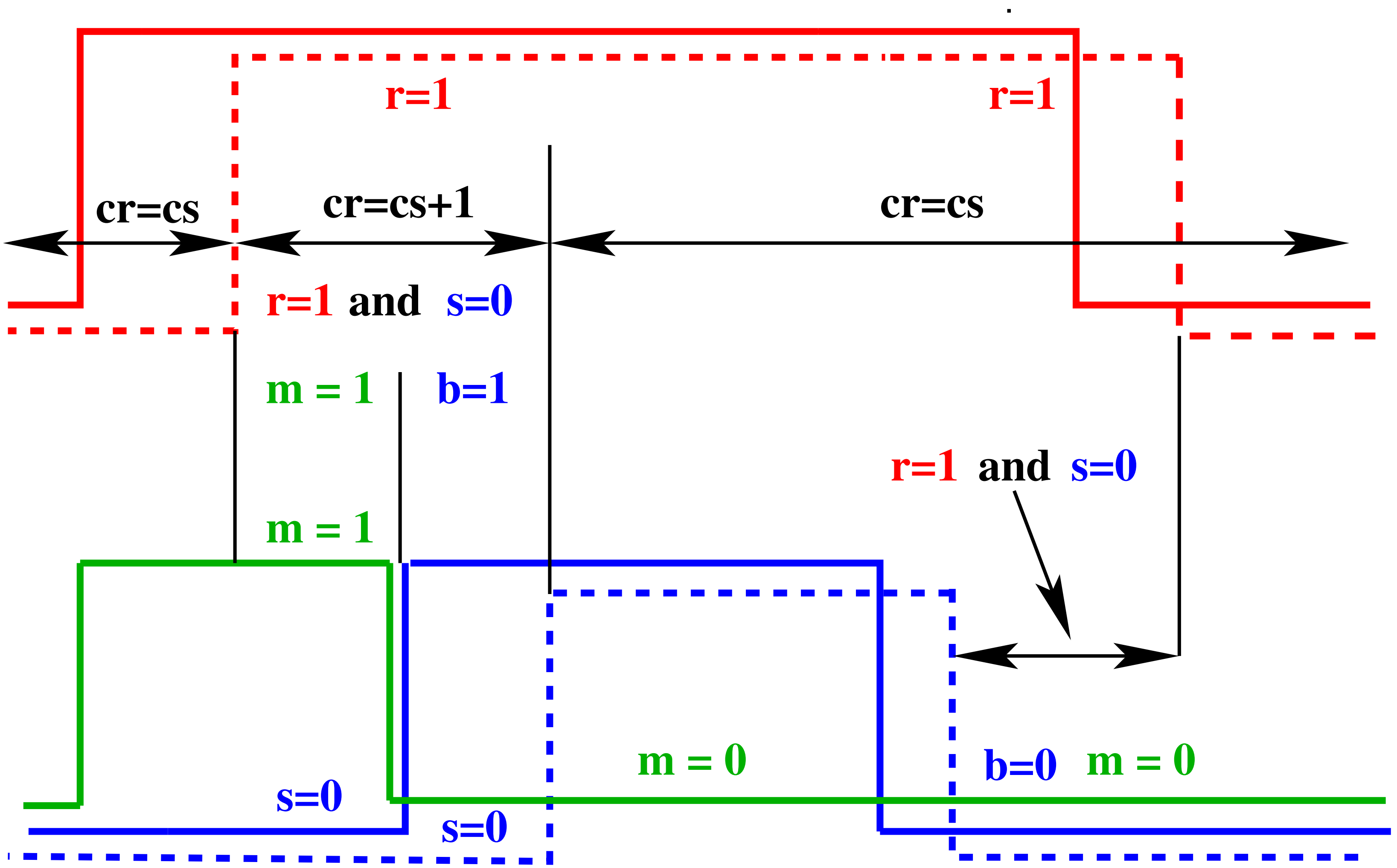
# This Solution Does not Work



# The Solution



# The Solution



$$r = 1 \ \wedge \ s = 0 \ \wedge \ (m = 1 \ \vee \ b = 1) \ \Rightarrow \ cr = cs + 1$$

$$r = 0 \ \vee \ s = 1 \ \vee \ (m = 0 \ \wedge \ b = 0) \ \Rightarrow \ cr = cs$$

# Summary of Refinement: the state

**dbl2_1:** $m \in \{0, 1\}$

**dbl2_2:** $m = 1 \quad \Rightarrow \quad ca = cb + 1$

**dbl2_3:** $m = 0 \quad \Rightarrow \quad ca = cb$

**dbl2_4:** $r = 1 \ \wedge \ s = 0 \ \wedge \ (m = 1 \ \vee \ b = 1) \quad \Rightarrow \quad cr = cs + 1$

**dbl2_5:** $r = 0 \ \vee \ s = 1 \ \vee \ (m = 0 \ \wedge \ b = 0) \quad \Rightarrow \quad cr = cs$

# Summary of Refinement: the state

**dbl2_1:** $m \in \{0, 1\}$

**dbl2_2:** $m = 1 \quad \Rightarrow \quad ca = cb + 1$

**dbl2_3:** $m = 0 \quad \Rightarrow \quad ca = cb$

**dbl2_4:** $r = 1 \ \land \ s = 0 \ \land \ (m = 1 \ \lor \ b = 1) \quad \Rightarrow \quad cr = cs + 1$

**dbl2_5:** $r = 0 \ \lor \ s = 1 \ \lor \ (m = 0 \ \land \ b = 0) \quad \Rightarrow \quad cr = cs$

- The following theorems are easy to prove

**thm2_1:** $ca = cb \ \lor \ ca = cb + 1$

**thm2_2:** $cr = cs \ \lor \ cr = cs + 1$

# More Invariant

**dbl2_1:** $m \ \in \ \{0,1\}$

**dbl2_2:** $m = 1 \quad \Rightarrow \quad ca = cb + 1$

**dbl2_3:** $m = 0 \quad \Rightarrow \quad ca = cb$

**dbl2_4:** $r = 1 \ \wedge \ s = 0 \ \wedge \ (m = 1 \ \vee \ b = 1) \quad \Rightarrow \quad cr = cs + 1$

**dbl2_5:** $r = 0 \ \vee \ s = 1 \ \vee \ (m = 0 \ \wedge \ b = 0) \quad \Rightarrow \quad cr = cs$

**dbl2_6:** $a = 0 \quad \Rightarrow \quad m = 0$

- The last new invariants was discovered while doing the proof

- It requires adding the guard $m = 0$ in event a_off

- The proofs are now (almost) completely automatic

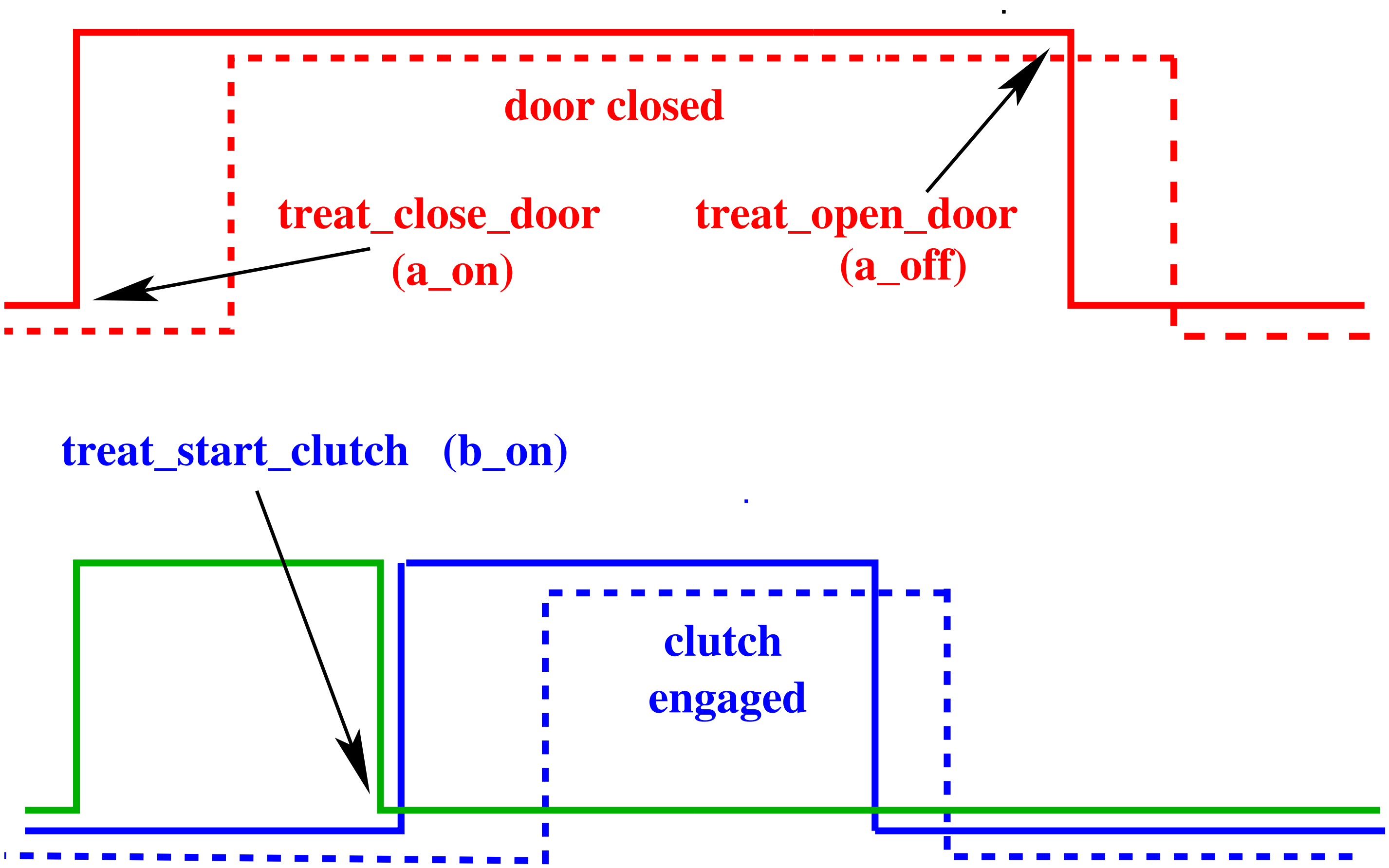
# The Final Modified Events

```
a_on
when
    $a = 0$
    $r = 0$
then
    $a := 1$
    $ca := ca + 1$
    $m := 1$
end
```

```
b_on
when
    $r = 1$
    $a = 1$
    $b = 0$
    $s = 0$
    $m = 1$
then
    $b := 1$
    $cb := cb + 1$
    $m := 0$
end
```

```
a_off
when
    $a = 1$
    $r = 1$
    $b = 0$
    $s = 0$
    $m = 0$
then
    $a := 0$
end
```

# Instantiation



# Pattern Instantiation for the 6th Refinement

- We instantiate the pattern as follows:

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $a$ | $\rightsquigarrow$ | $door\_actuator$ | $b$ | $\rightsquigarrow$ | $clutch\_actuator$ |
| $r$ | $\rightsquigarrow$ | $door\_sensor$ | $s$ | $\rightsquigarrow$ | $clutch\_sensor$ |
| $0$ | $\rightsquigarrow$ | $open$ | $0$ | $\rightsquigarrow$ | $disengaged$ |
| $1$ | $\rightsquigarrow$ | $closed$ | $1$ | $\rightsquigarrow$ | $engaged$ |

|  |  |  |
|---|---|---|
| a_on | $\rightsquigarrow$ | treat_close_door |
| a_off | $\rightsquigarrow$ | treat_open_door |
| b_on | $\rightsquigarrow$ | treat_start_clutch |

# 6th Refinement: Adapting the events of the pattern (2)

```
a_on
when
    $a = 0$
    $r = 0$
then
    $a := 1$
    $m := 1$
end
```

```
treat_close_door
    when
        $door\_actuator = open$
        $door\_sensor = open$
        $motor\_actuator = working$
        $motor\_sensor = working$
    then
        $door\_actuator := closed$
        $m := 1$
    end
```

# 6th Refinement: Adapting the events of the pattern (2)

```
b_on
when
  $b = 0$
  $s = 0$
  $r = 1$
  $a = 1$
  $m = 1$
then
  $b := 1$
  $m := 0$
end
```

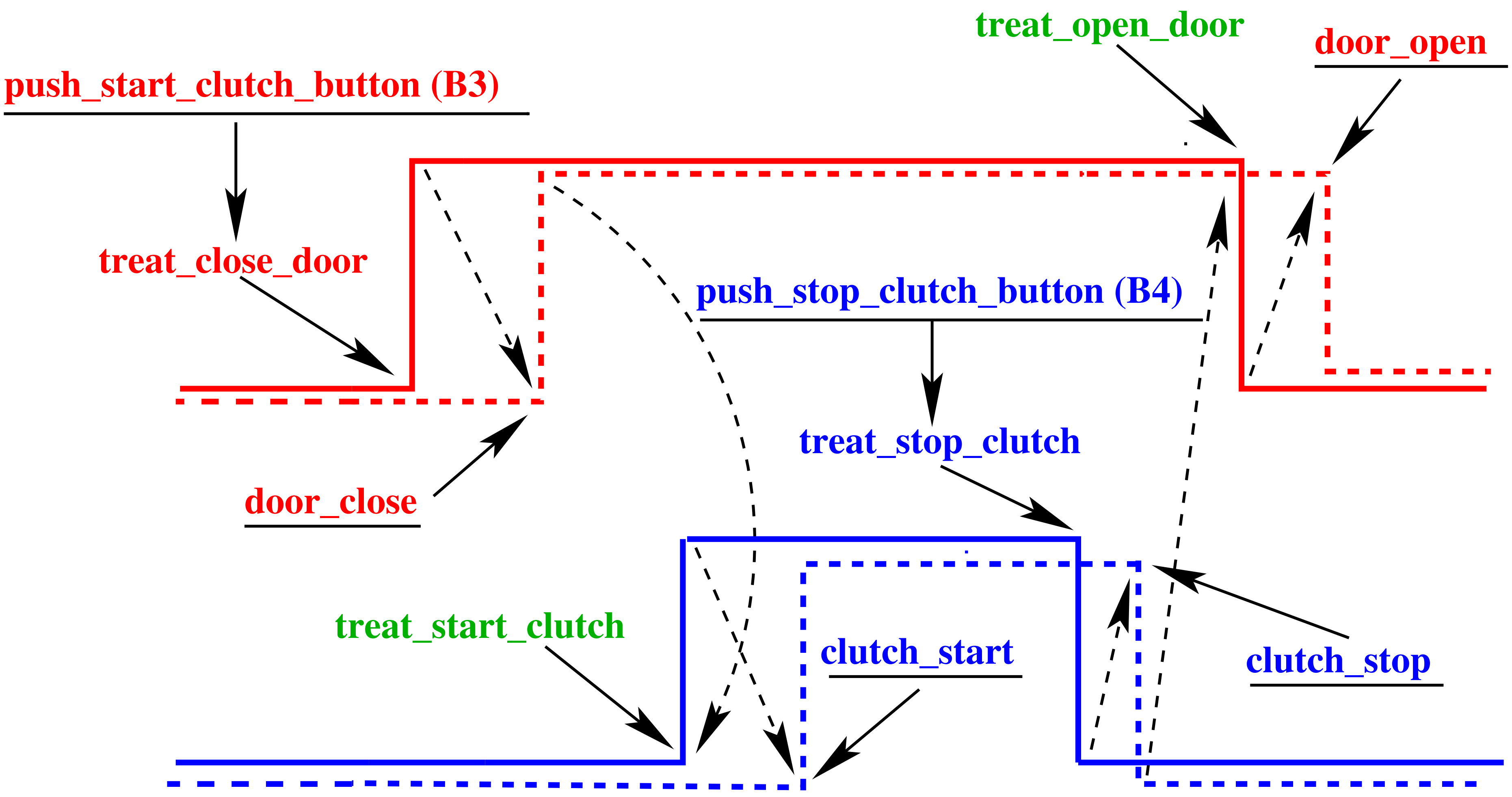
```
treat_start_clutch
  when
    $motor\_actuator = working$
    $motor\_sensor = working$
    $clutch\_actuator = disengaged$
    $clutch\_sensor = disengaged$
    $door\_sensor = closed$
    $door\_actuator = closed$
    $m = 1$
  then
    $clutch\_actuator := engaged$
    $m := 0$
  end
```

# 6th Refinement: Adapting the events of the pattern (3)

```
a_off
when
    $a = 1$
    $r = 1$
    $s = 0$
    $b = 0$
    $m = 0$
then
    $a := 0$
end
```

```
treat_open_door
    when
        $door\_actuator = closed$
        $door\_sensor = closed$
        $clutch\_sensor = disengaged$
        $clutch\_actuator = disengaged$
        $m = 0$
    then
        $door\_actuator := open$
    end
```

# The Complete Synchronization of Door and Clutch



- treat_close_door is the result of depressing button B3
- treat_stop_clutch is the result of depressing button B4
- treat_start_clutch and treat_open_door are automatic

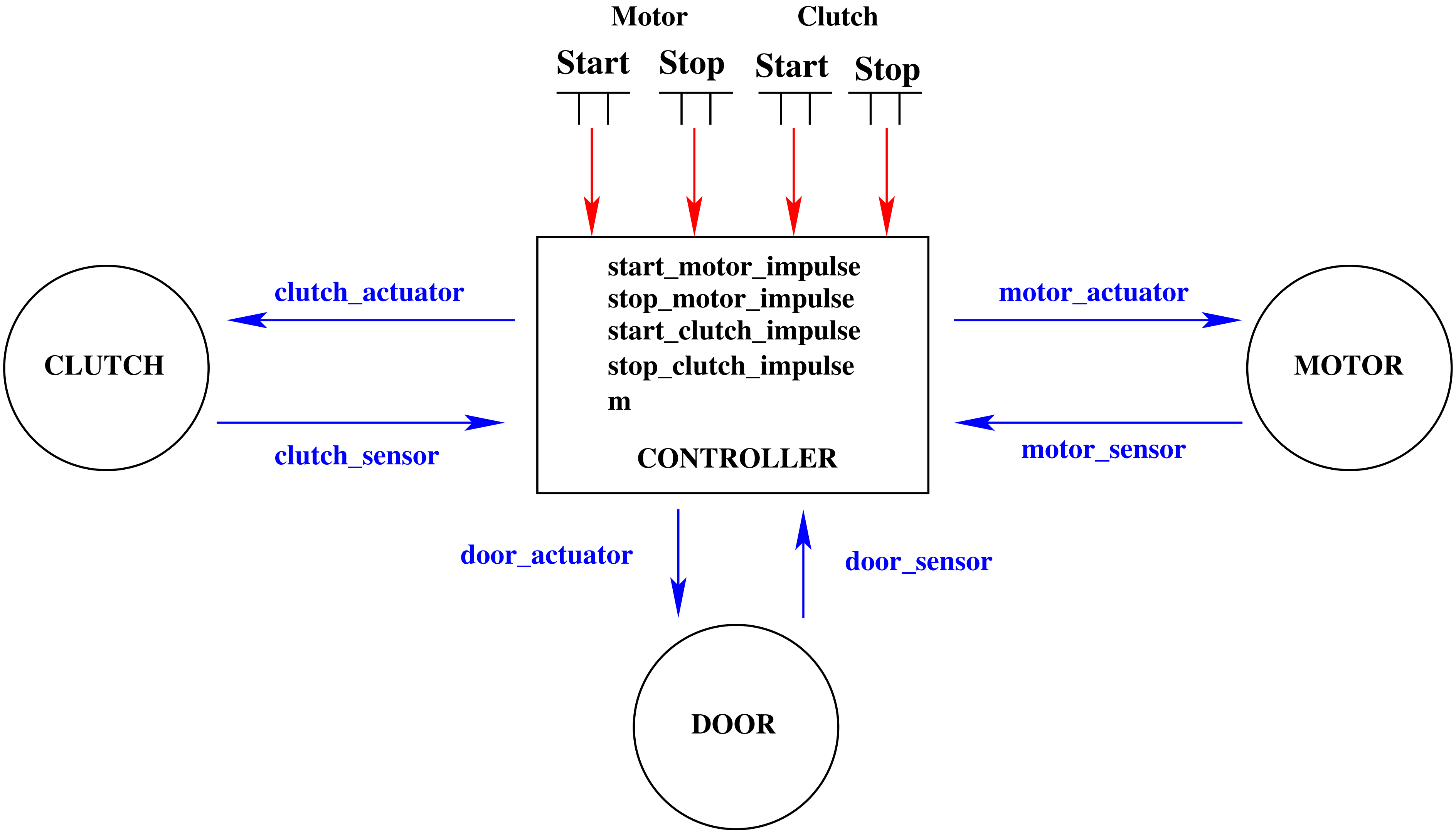
# Sixth Refinement: Summary of the Events (1)

- Environment (no new events)
    - motor_start
    - motor_stop
    - clutch_start
    - clutch_stop
    - door_close
    - door_open
    - push_start_motor_button
    - release_start_motor_button
    - push_stop_motor_button
    - release_stop_motor_button

# Sixth Refinement: Summary of the Events (2)

- Controller (no new events)
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch
  - treat_close_door
  - treat_open_door

# 7th Reft.: Connecting the Controller to the Clutch Buttons



# Reminder: Synchronizing Door and Clutch

- There are no door buttons

- The door must be closed before engaging the clutch

- The door must be opened after disengaging the clutch

- It is sufficient to connect:

    - button B3 to the door (closing the door)

    - button B4 to the clutch (disengaging the clutch)

# Seventh Refinement: Summary of the Events (Environment)

- motor_start
- motor_stop
- clutch_start
- clutch_stop
- door_close
- door_open
- push_start_motor_button
- release_start_motor_button
- push_stop_motor_button
- release_stop_motor_button
- push_start_clutch_button
- release_start_clutch_button
- push_stop_clutch_button
- release_stop_clutch_button

# Seventh Refinement: Summary of the Events (Controller)

- treat_push_start_motor_button
- treat_push_start_motor_button_false
- treat_push_stop_motor_button
- treat_push_stop_motor_button_false
- treat_release_start_motor_button
- treat_release_stop_motor_button
- treat_start_clutch
- treat_stop_clutch
- treat_close_door
- treat_open_door
- treat_close_door_false
- treat_stop_clutch_false
- treat_release_start_clutch_button
- treat_release_stop_clutch_button

# Decomposing the Final Model: Environment

- The environment events

- The environment variables modified by environment events

- The sensor variables modified by environment events

- The actuator variables read by environment events

- The controller variables not seen by environment events

- No environment variables in this model

# Decomposing the Final Model: Controller

- The controller events

- The controller variables modified by controller events

- The sensor variables read by controller events

- The actuator variables modified by controller events

- The environment variables not seen by controller events

- No environment variables in this model

# Summary: Variables of the Last Refinement (1)

- 7 sensor variables:
    - $motor\_sensor$
    - $clutch\_sensor$
    - $door\_sensor$
    - $start\_motor\_button$
    - $stop\_motor\_button$
    - $start\_clutch\_button$
    - $stop\_clutch\_button$

# Summary: Variables of the Last Refinement (2)

- 3 actuator variables:
    - $motor\_actuator$
    - $clutch\_actuator$
    - $door\_actuator$
- 5 controller variables (without the counter variables):
    - $start\_motor\_impulse$
    - $stop\_motor\_impulse$
    - $start\_clutch\_impulse$
    - $stop\_clutch\_impulse$
    - $m$

# Summary: Events of the Last Refinement

- 14 environment events,

- 14 controller events,

- 130 lines for environment events,

- 180 lines for controller events.

# Summary: Usage of the Design Patterns

- 4 weak reactions: 4 buttons (B1, B2, B3, B4)

- 3 strong reactions: 3 devices (motor, clutch, door)

- 3 strong-weak reactions: motor-clutch, clutch-door, motor-door

- 1 strong-strong reaction: clutch-door

# Summary: Number of Invariants

- Weak reaction: 6
- Strong reaction: 3
- Strong-weak reaction: 16
- Strong-strong reaction: 6
- Total: 31

- Press (typing): 15
- Total: 15

# Summary: Number of Proof Obligations

- Weak reaction: 18
- Strong reaction: 12
- Strong-weak reaction: 60
- Strong-strong reaction: 29
- Total: 119

- Press: 0

- PO saving: 4x18 + 3x12 + 3x60 + 29 = 317

# Summary: Proofs

- Design patterns: 119 (all automatic)

- Press: 0

# Conclusion

- This design pattern approach is very fruitful

- It results in a very systematic formal development

- Many other patterns have to be developed

- More automation has to be provided (plug-in)