CS375:
Logic and Theory of Computing

Fuhua (Frank) Cheng

Department of Computer Science

University of Kentucky

Table of Contents:

- **Week 1: Preliminaries (set algebra, relations, functions) (read Chapters 1-4)**
- **Weeks 2-5: Regular Languages, Finite Automata (Chapter 11)**
- **Weeks 6-8: Context-Free Languages, Pushdown Automata (Chapters 12)**
- **Weeks 9-11: Turing Machines (Chapter 13)**

Table of Contents (conti):

- **Weeks 12-13: Propositional Logic (Chapter 6), Predicate Logic (Chapter 7), Computational Logic (Chapter 9), Algebraic Structures (Chapter 10)**

8. Turing Machines and Equivalent Models – Turing Machines

Goal: study Turing machines and Church-Turing thesis,

and show that

there are no models of computation more powerful than Turing machines.

8. Turing Machines

Turing machines (TM):

- an abstract model of computation
- provide a precise, formal definition of what it means for a function to be computable
- Many other definitions of computation have been proposed over the years
- Turing Machine definition seems to be the simplest

8. Turing Machines

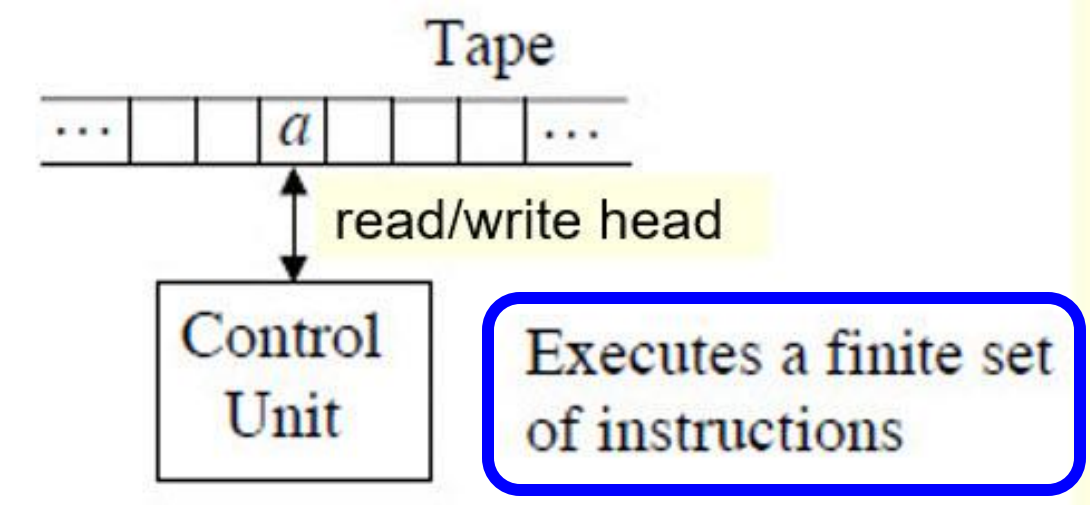
Such kind of tape of course does not exist in real life. You simply assume we have a tape with many cells so that everything we need for our work (not large for now) can be stored in those cells.

Turing machine (TM):

a simple computer w/ an infinite amount of **storage** in the form of cells on an infinite tape,

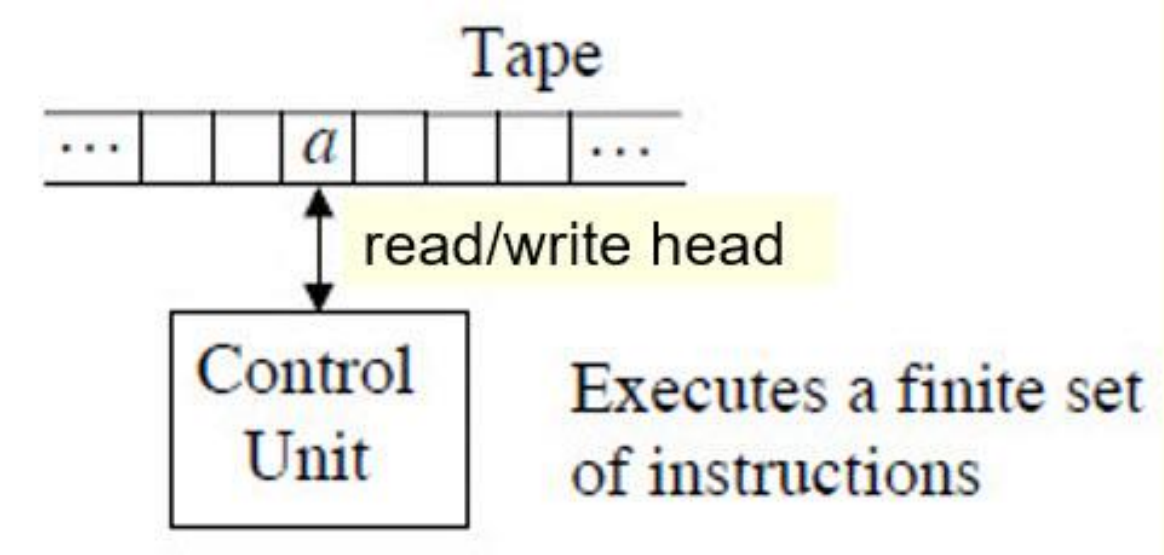
&

a **control unit** that contains a finite set of state transition instructions



8. Turing Machines

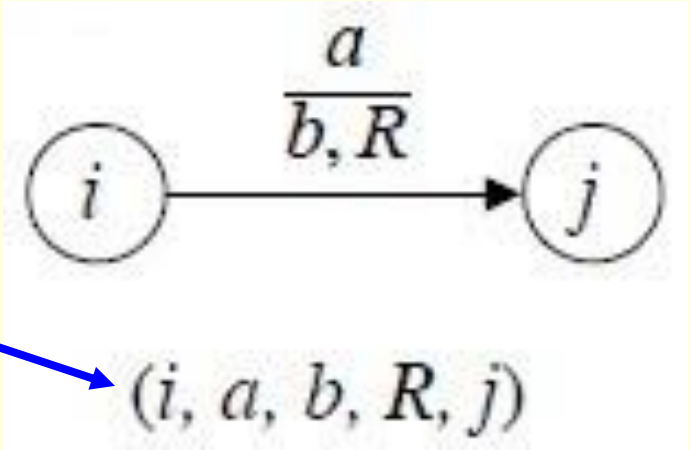
An instruction **reads** the symbol in the current tape cell,
writes a symbol to the same cell, and then
moves the tape head one cell left or right or
remains at the same cell,
after which the machine enters a new state.



8. Turing Machines

Assumptions:

- The symbol Λ denotes a blank cell.
- The tape head (read/write head) is initially at the left end of a nonempty input string (unless specified otherwise)
- There is a designated start state.
- There is one "halt" state.
- The moves are denoted by *L, S, R* for move left, stay, and move right.

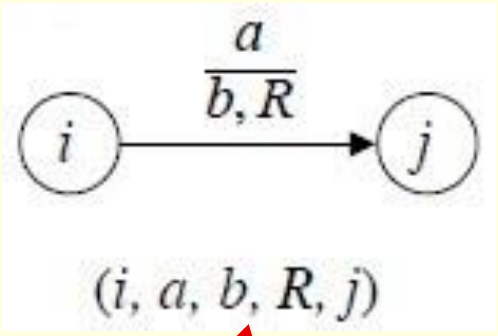


Instructions

8. Turing Machines

Each instruction contains 5 parts:

1. Current machine state
2. Tape symbol read from current tape cell
3. Tape symbol to write into current tape cell
4. Direction for the tape head to move
5. Next machine state



This instruction executes if: current state = *i*
current input symbol = a

Things the machine does: writes *b* into that cell
moves right one cell
enters state *j*

The **control unit** is implemented as a **circuit board** with, in the early days, vacuum tubes, called a **CPU**.

Each vacuum tube represents a state and the vacuum tubes are connected with wires, like **directed edges** in a **deterministic finite-state machine**.

A lit vacuum tube represents the finite state machine is at that state. So the control unit is implemented as a piece of hardware. Since there are billion even trillion states in a control CPU, the size of an early day CPU could be as big as a whole warehouse.

These days, a CPU is just as small as a **finger nail**, like the one used in your cell phone.

8. Turing Machines - some history

What is the Nobel Prize equivalent in computer science area?

Turing Award

8. Turing Machines - some history

Named after Dr. Alan Turing
(1912 – 1954)

- an English Mathematician, computer scientist, logician

8. Turing Machines - some history

For developing the foundation for modern day computers

i.e., Turing machines (1936)

- a piece of work that changed the world/human history

8. Turing Machines - some history

Doctoral Advisor: Alonzo Church

whose work (Church-Turing thesis) will also be studied

8. Turing Machines - some history

Dr. Alan Turing died when he was only 42

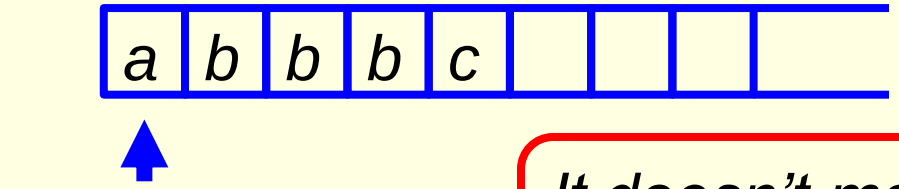
- but his contribution will forever last

8. Turing Machines

Acceptance

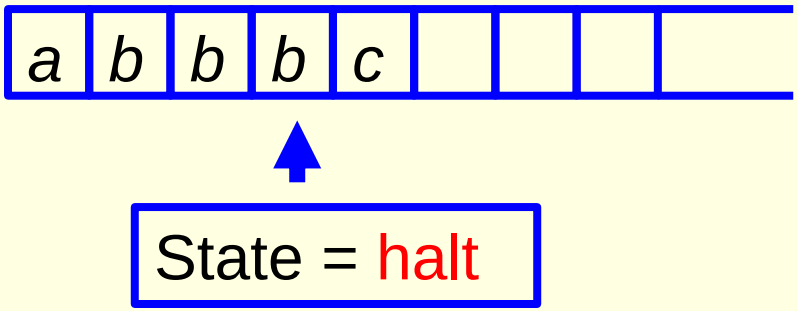
An input string on the tape is *accepted* by the TM if the machine enters the halt state. The language of a TM is the set of all strings accepted by the machine.

Initially:



It doesn't matter if the end of the string is reached

After i steps:



Why?

abbbc is accepted

Question: would the computer execute the following C++ program?

```
int main ( )
{
    cout << “Hello, how are your?” << endl;
} Hello
```

8. Turing Machines

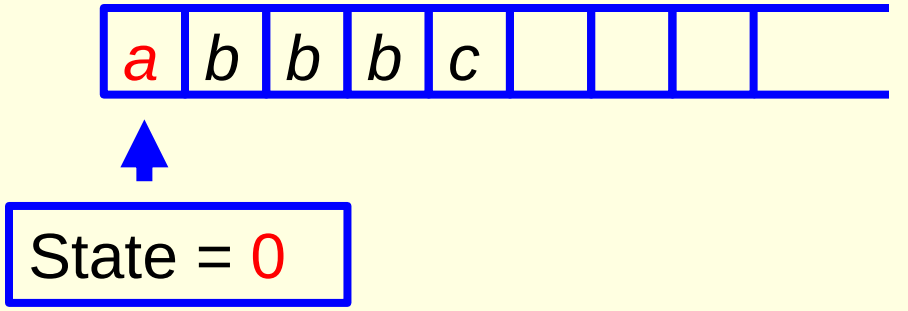
To operate a Turing machine,

you must know how to operate/manage its states

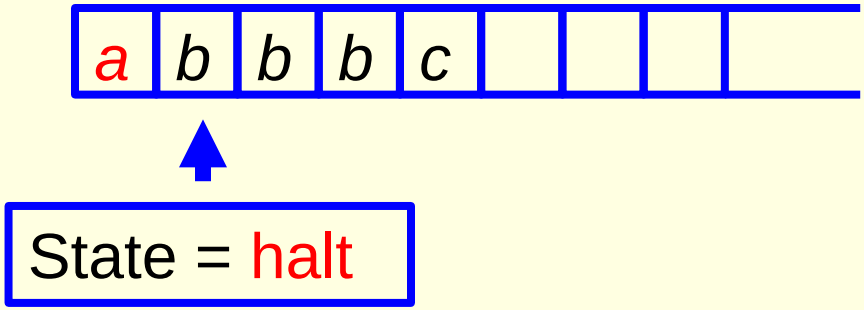
handle/manipulate

Example. *Given the simple TM : (0, a, a, R, halt). What is the language of the TM?*

Initially:



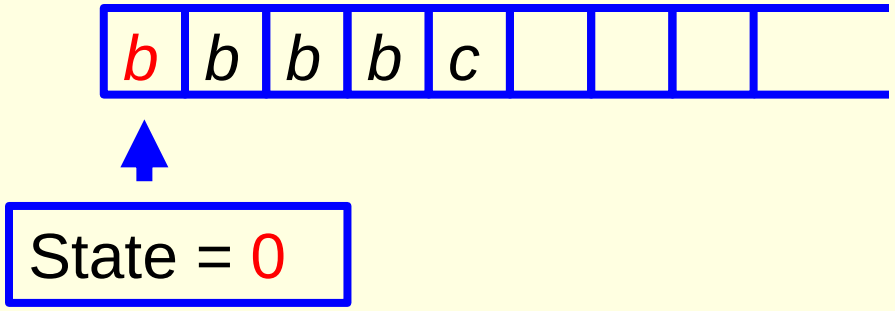
After 1 step:



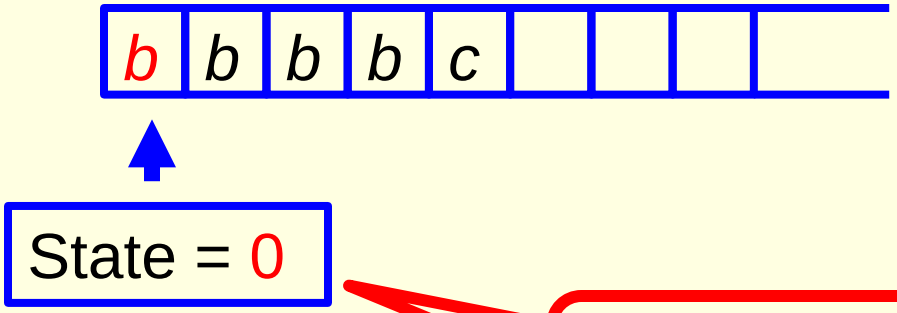
abbbc is accepted

Example. *Given the simple TM : (0, a, a, R, halt). What is the language of the TM?*

Initially:



After 1 step:



The instruction is not executed

bbbbc not accepted

Example. *Given the simple TM : (0, a, a, R, halt). What is the language of the TM?*

Solution:

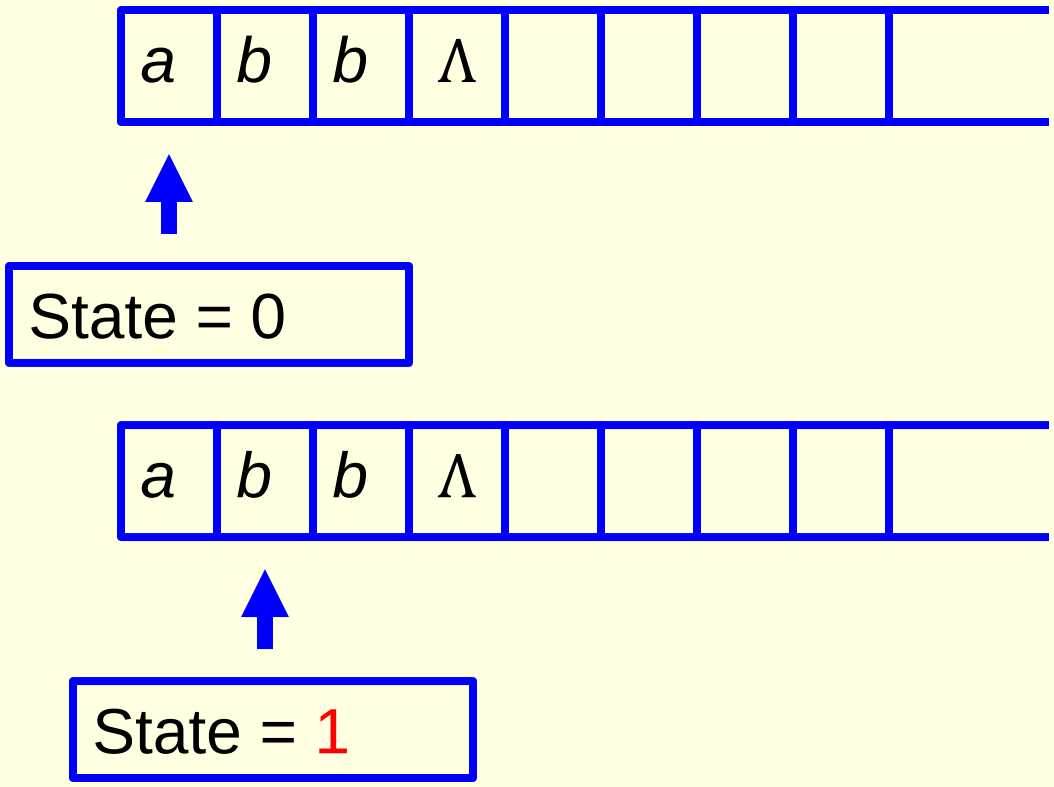
Any string beginning with a will be accepted.

Language of the TM = {all the strings beginning with a }

Example. Construct a TM to accept the language

$\{ ab^n \mid n \in \mathbf{N} \}$

Consider



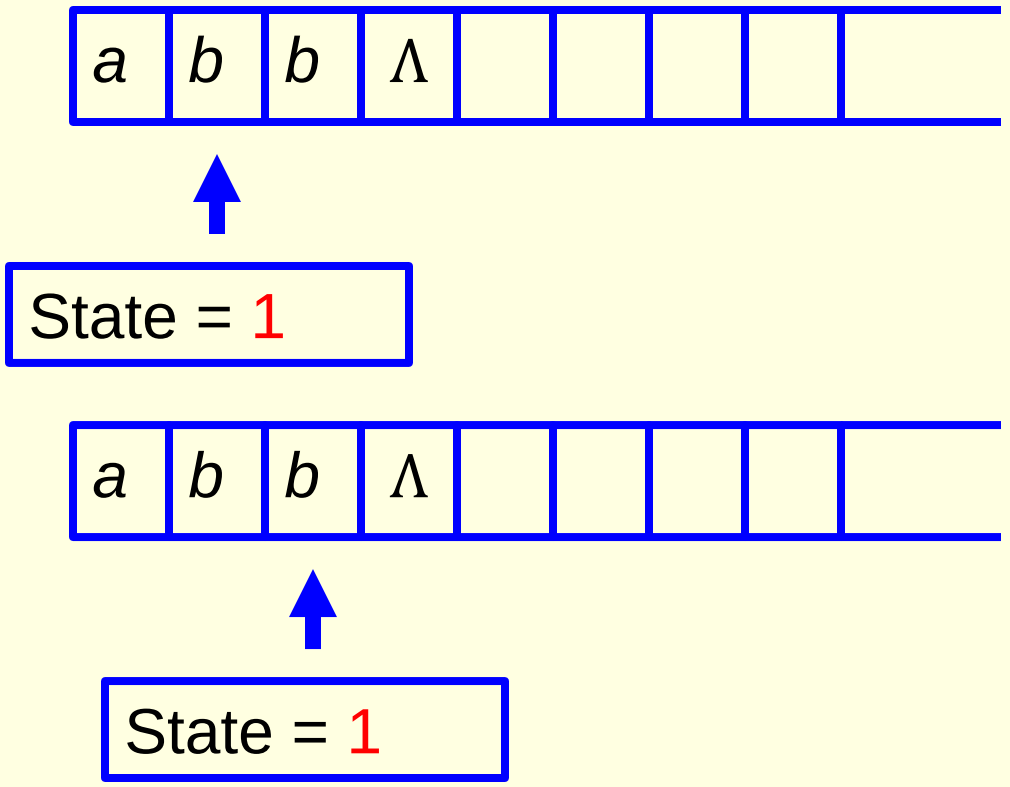
To get

I need: (0, a, a, R, 1)

Example. Construct a TM to accept the language

$\{ ab^n \mid n \in \mathbf{N} \}$

To get from



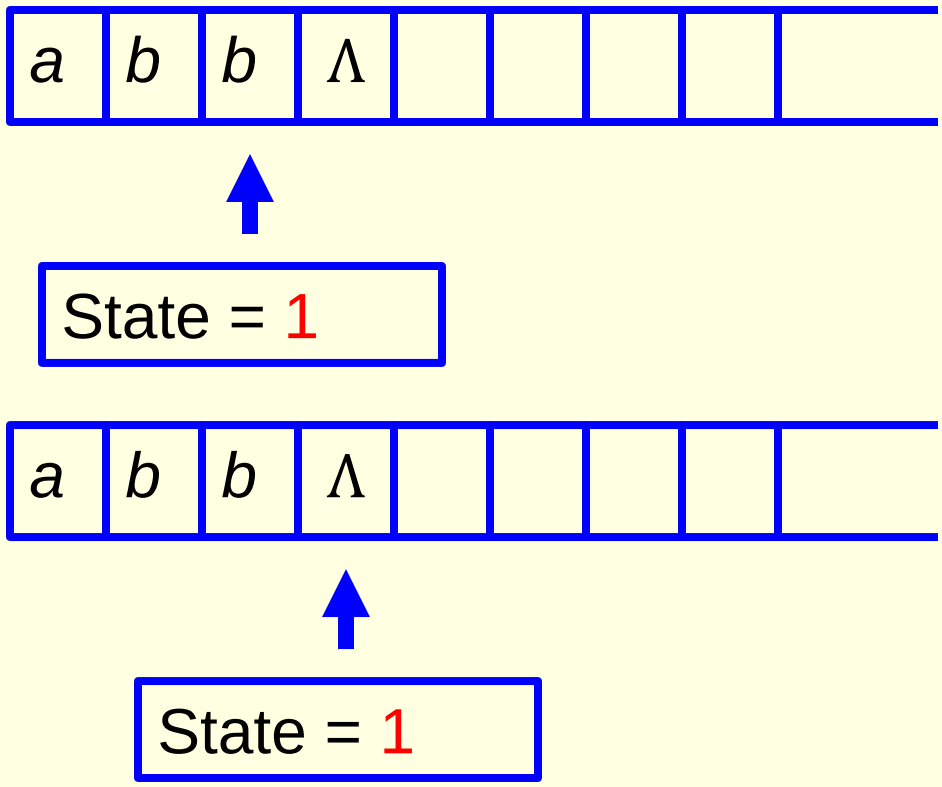
to

I need: (1, b, b, R, 1)

Example. Construct a TM to accept the language

$\{ ab^n \mid n \in \mathbf{N} \}$

To get from



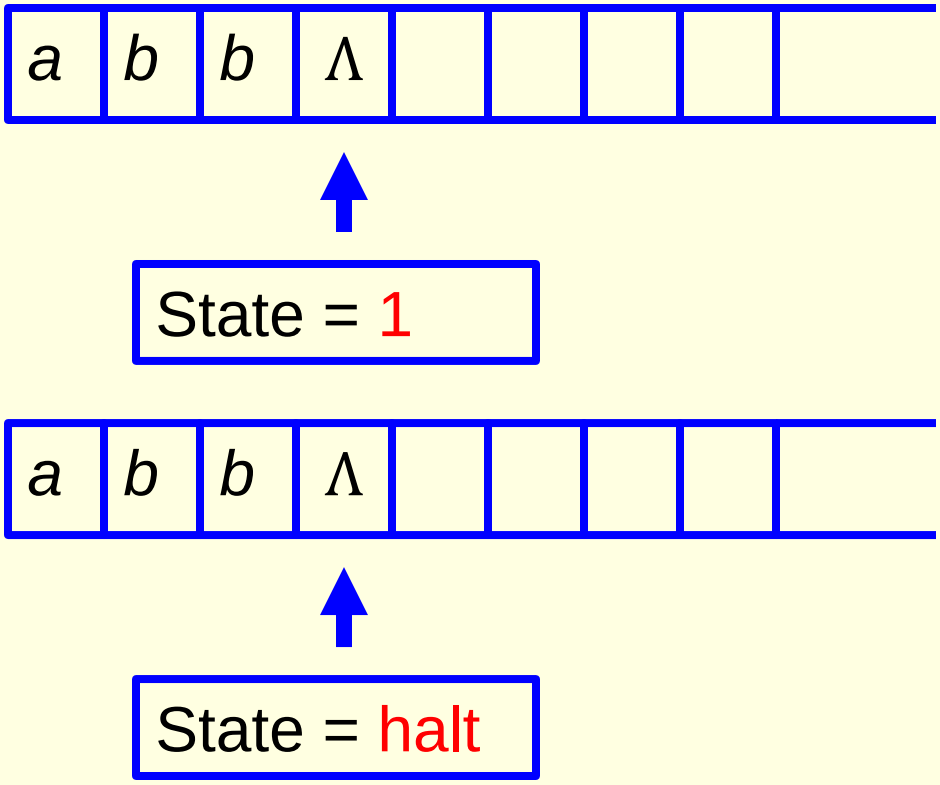
to

I can use: (1, b, b, R, 1)

Example. Construct a TM to accept the language

$\{ ab^n \mid n \in \mathbf{N} \}$

To get from



to

I need: (1, Λ, Λ, S, halt)

Example. Construct a TM to accept the language

$\{ ab^n \mid n \in \mathbf{N} \}$

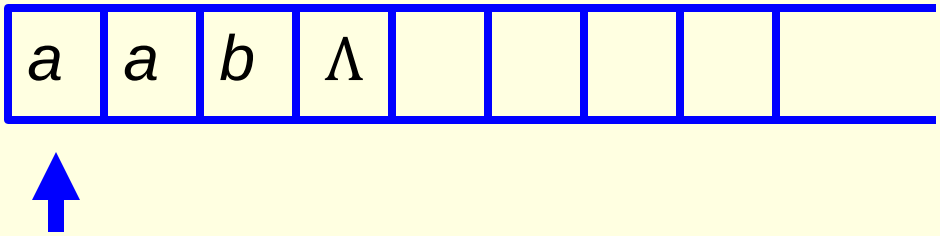
Instructions we have used/created so far:

(0, *a*, *a*, *R*, *1*) (1, *b*, *b*, *R*, *1*) (1, *Λ*, *Λ*, *S*, *halt*)

Example. Construct a TM to accept the language

$\{ ab^n \mid n \in \mathbf{N} \}$

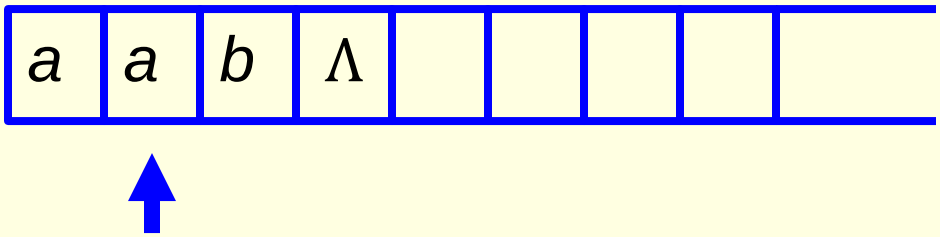
If we have



using: (0, a, a, R, 1)

State = 0

we get



State = 1

But then we get stuck. So aab is not accepted

Example. Construct a TM to accept the language

$\{ ab^n \mid n \in \mathbf{N} \}$

Solution.

(0, *a*, *a*, *R*, *1*) (1, *b*, *b*, *R*, *1*) (1, *Λ*, *Λ*, *S*, *halt*)

Question: *would*

(*0*, *a*, *a*, *R*, *0*) (*0*, *b*, *b*, *R*, *0*) (*0*, *Λ*, *Λ*, *S*, *halt*)

work?

Pause for a second (1)

Do you know what makes the design of a Turing Machine standing out?

the concept of using finite automata to control the functions of the control unit of a Turing Machine.

Pause for a second (2)

A PDA can implement one and only one stack

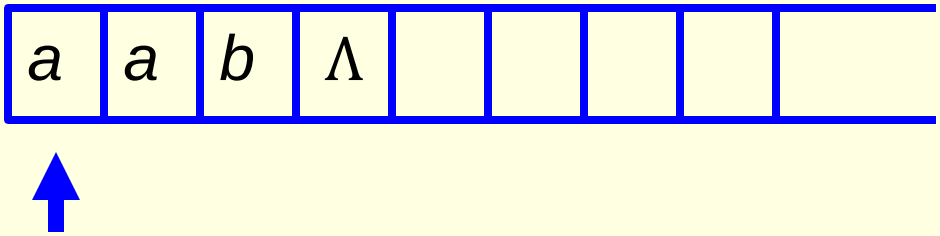
A Turing Machine can implement multiple (one, two, three, …, hundreds, thousands, …, millions, …) stacks implicitly for each task. Hence, a Turing Machine can function as the combination of tens, hundreds, …, millions of PDAs.

Question: *would*

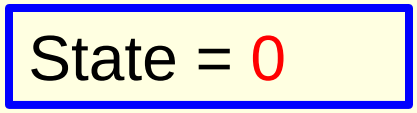
(0, a, a, R, 0) (0, b, b, R, 0) (0, Λ, Λ, S, halt)

work?

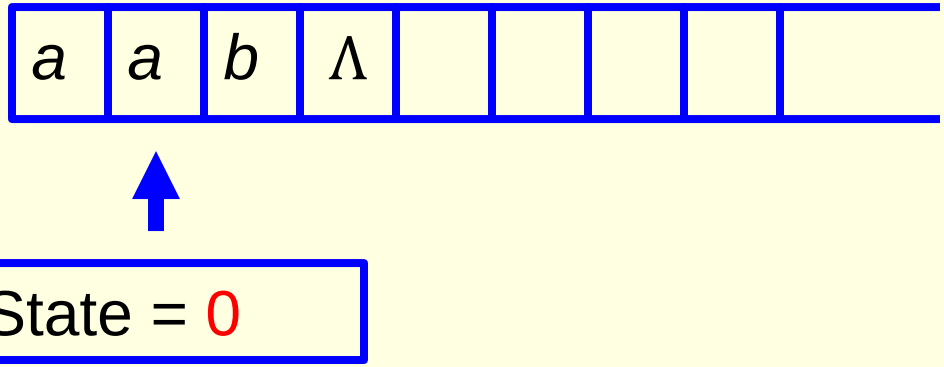
Consider



using: (0, a, a, R, 0)



we get

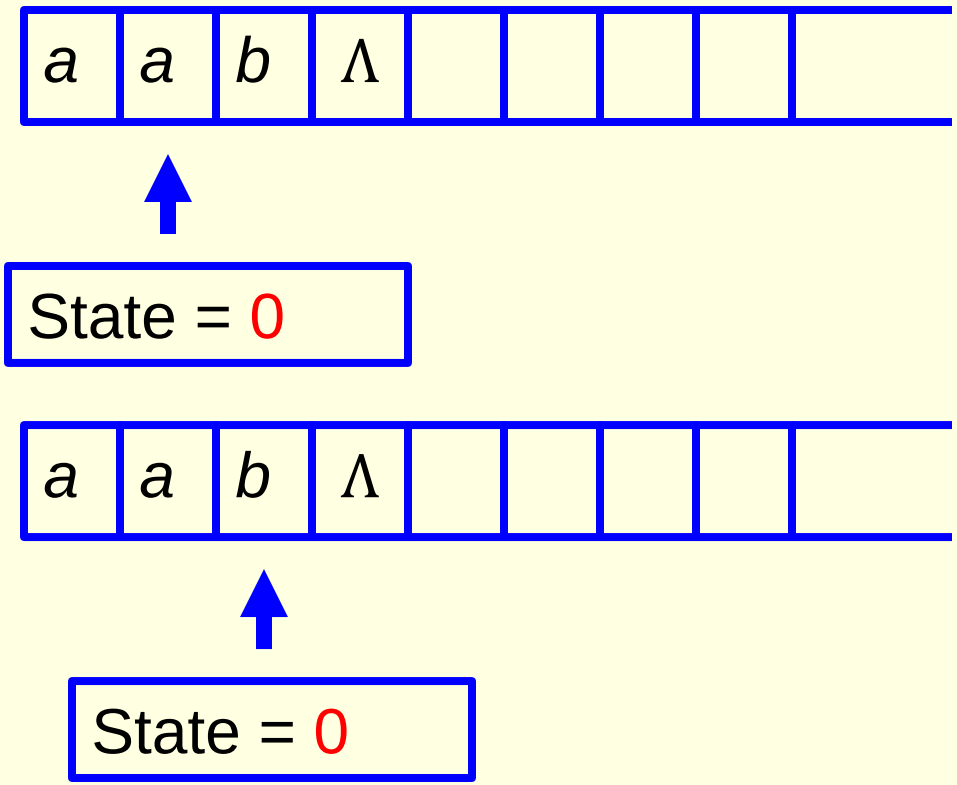


Question: *would*

(0, a, a, R, 0) *(0, b, b, R, 0)* *(0, Λ, Λ, S, halt)*

work?

We can use
(0, a, a, R, 0)
again



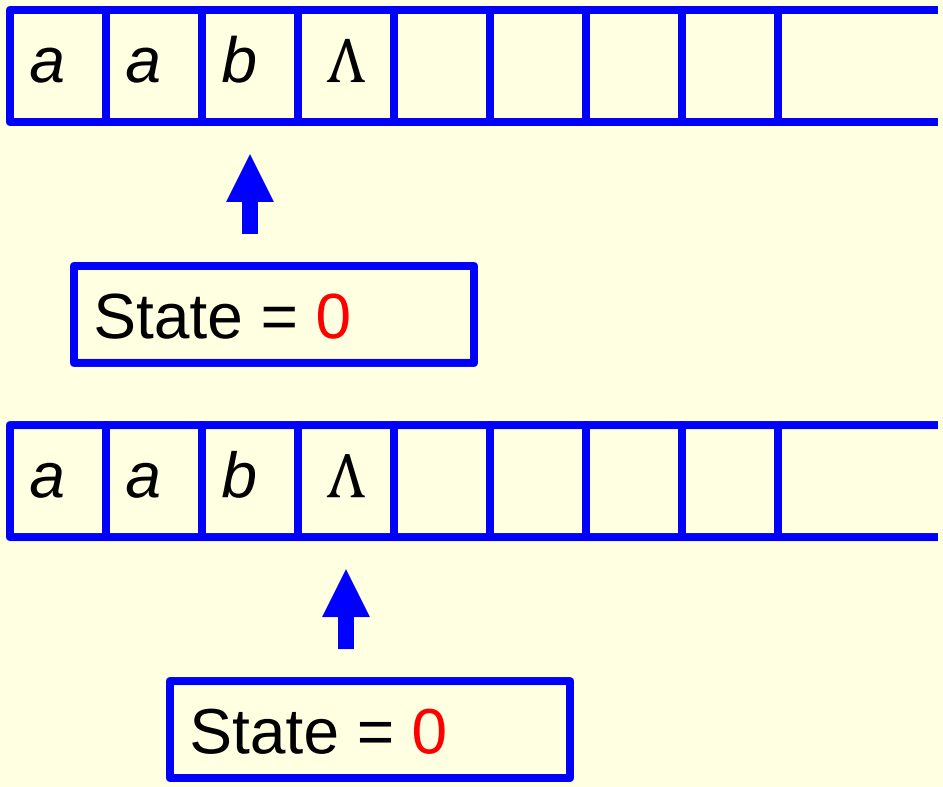
we get

Question: *would*

(0, a, a, R, 0) *(0, b, b, R, 0)* *(0, Λ, Λ, S, halt)*

work?

Then by using
(0, b, b, R, 0)



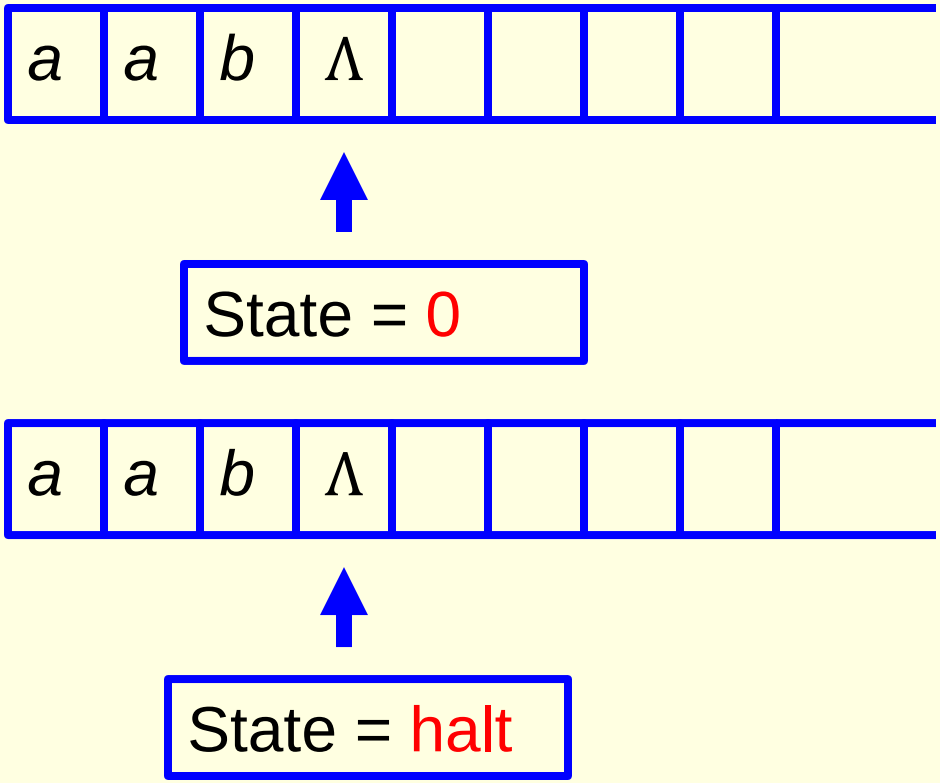
we get

Question: *would*

(0, a, a, R, 0) (0, b, b, R, 0) (0, Λ, Λ, S, halt)

work?

Then by using
(0, Λ, Λ, S, halt)



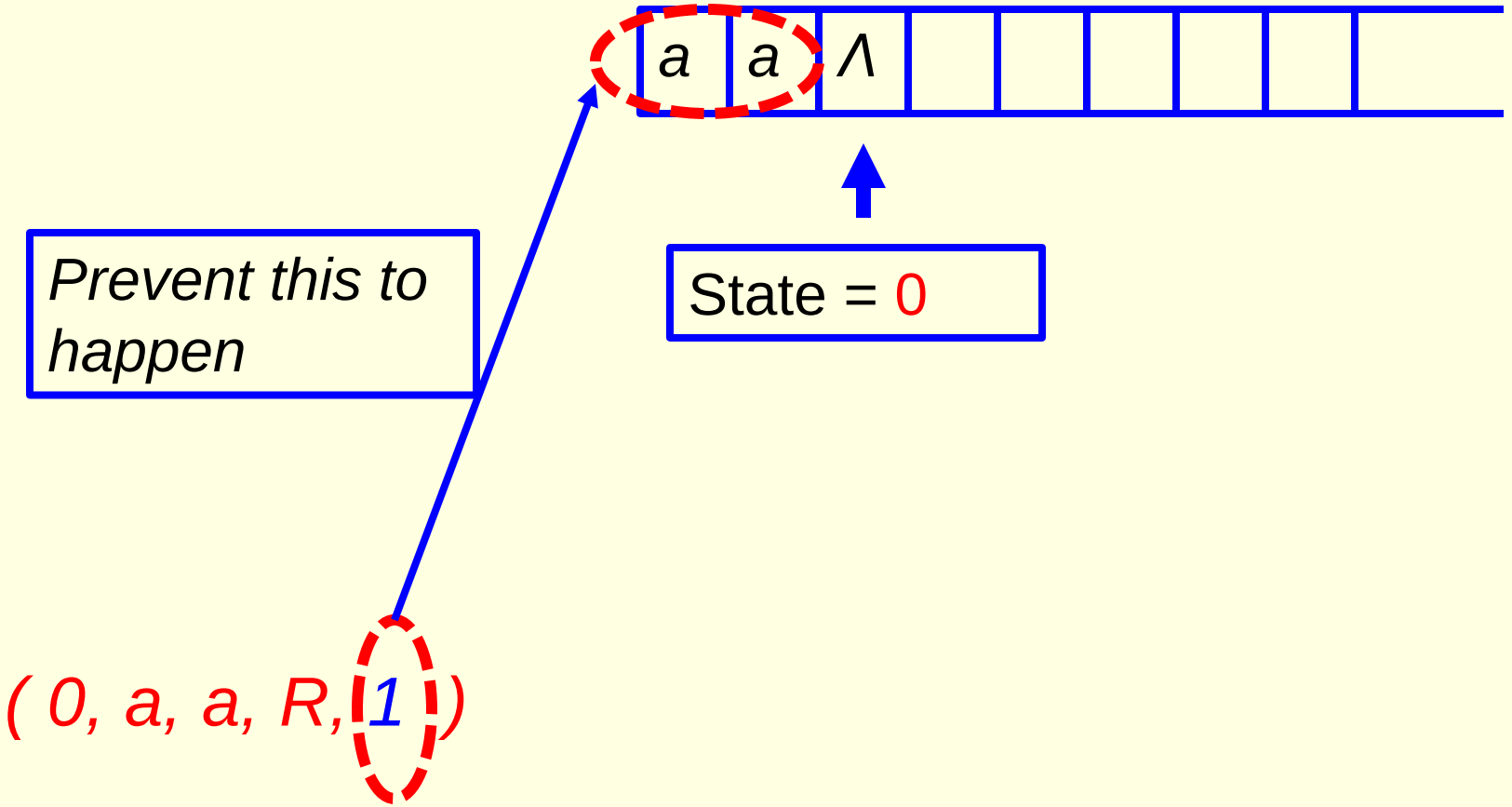
we get

So aab is accepted

Question: *would*

(0, a, a, R, 0) *(0, b, b, R, 0)* *(0, Λ, Λ, S, halt)*

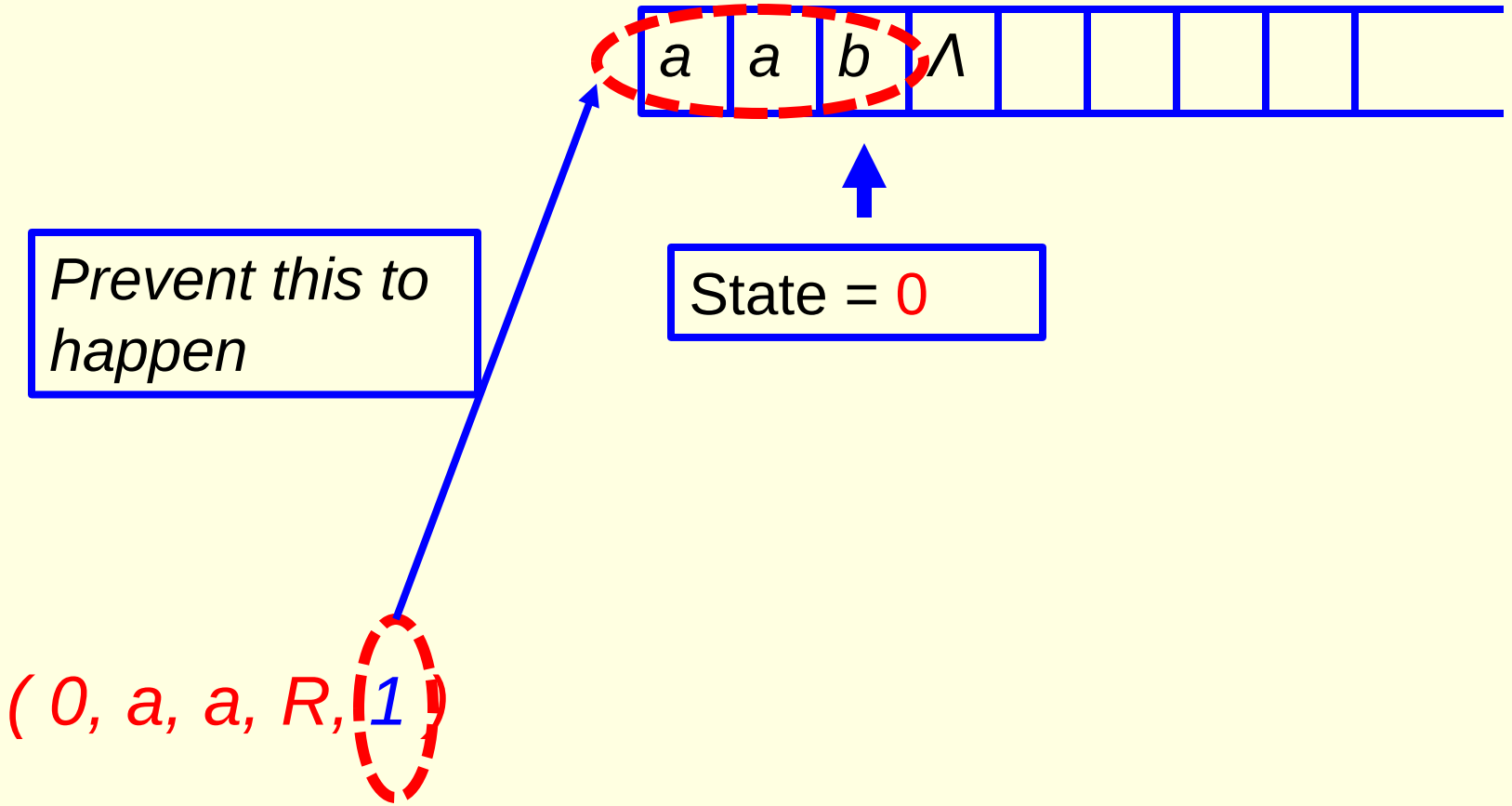
work? ***NO***



Question: *would*

(0, a, a, R, 0) *(0, b, b, R, 0)* *(0, Λ, Λ, S, halt)*

work? ***NO***

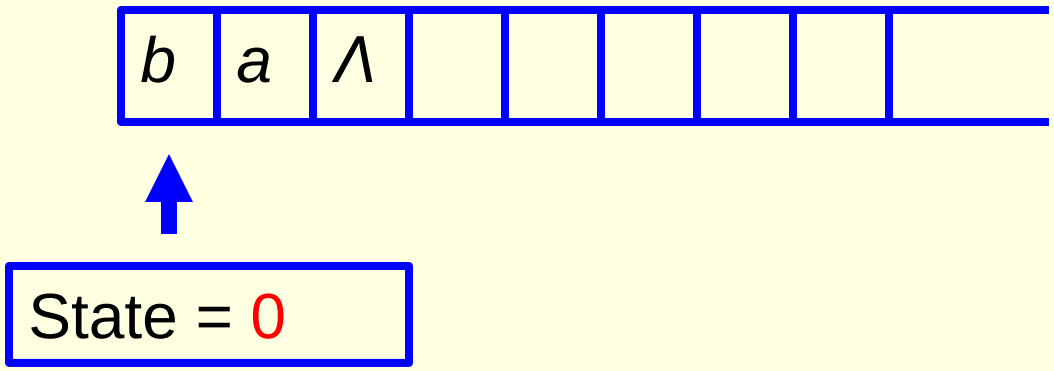


Question: *would*

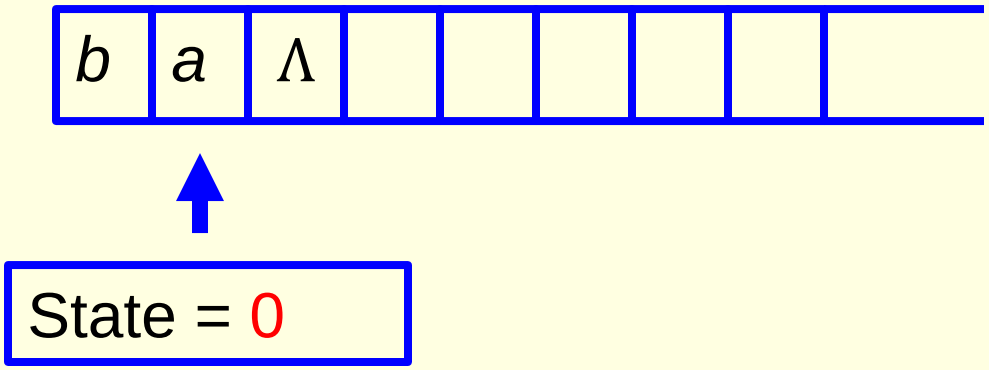
(0, a, a, R, 0) (0, b, b, R, 0) (0, Λ, Λ, S, halt)

work? ***NO***

Also consider



With (0, b, b, R, 0) we get

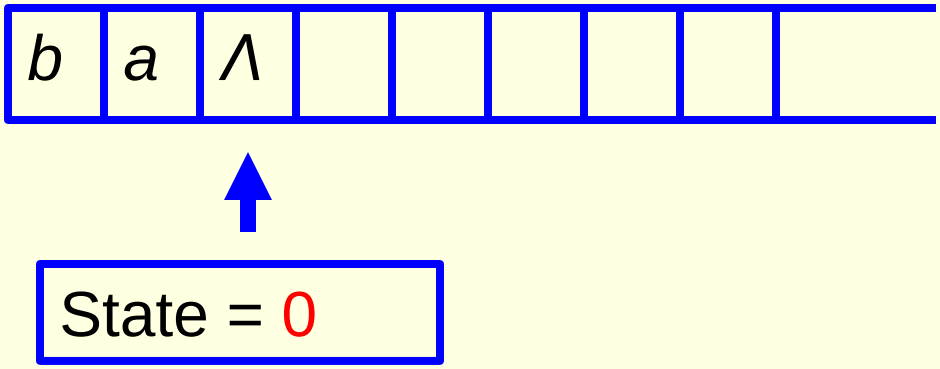


Question: *would*

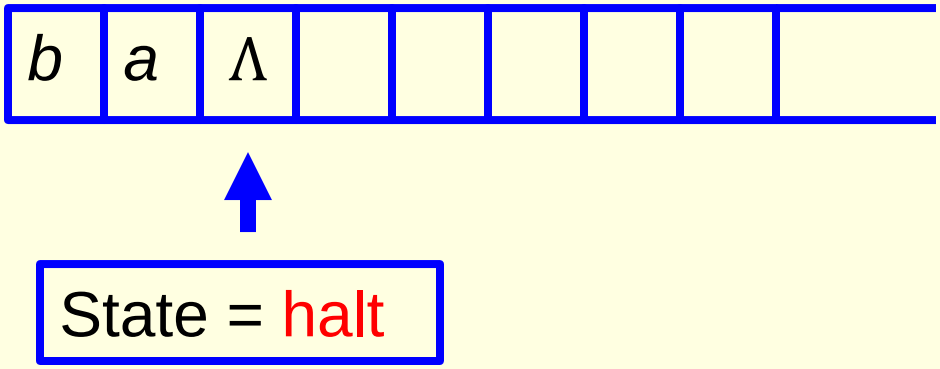
(0, a, a, R, 0) *(0, b, b, R, 0)* *(0, Λ, Λ, S, halt)*

work? ***NO***

With (0, a, a, R, 0)
we get



With (0, Λ, Λ, S, halt)
we get

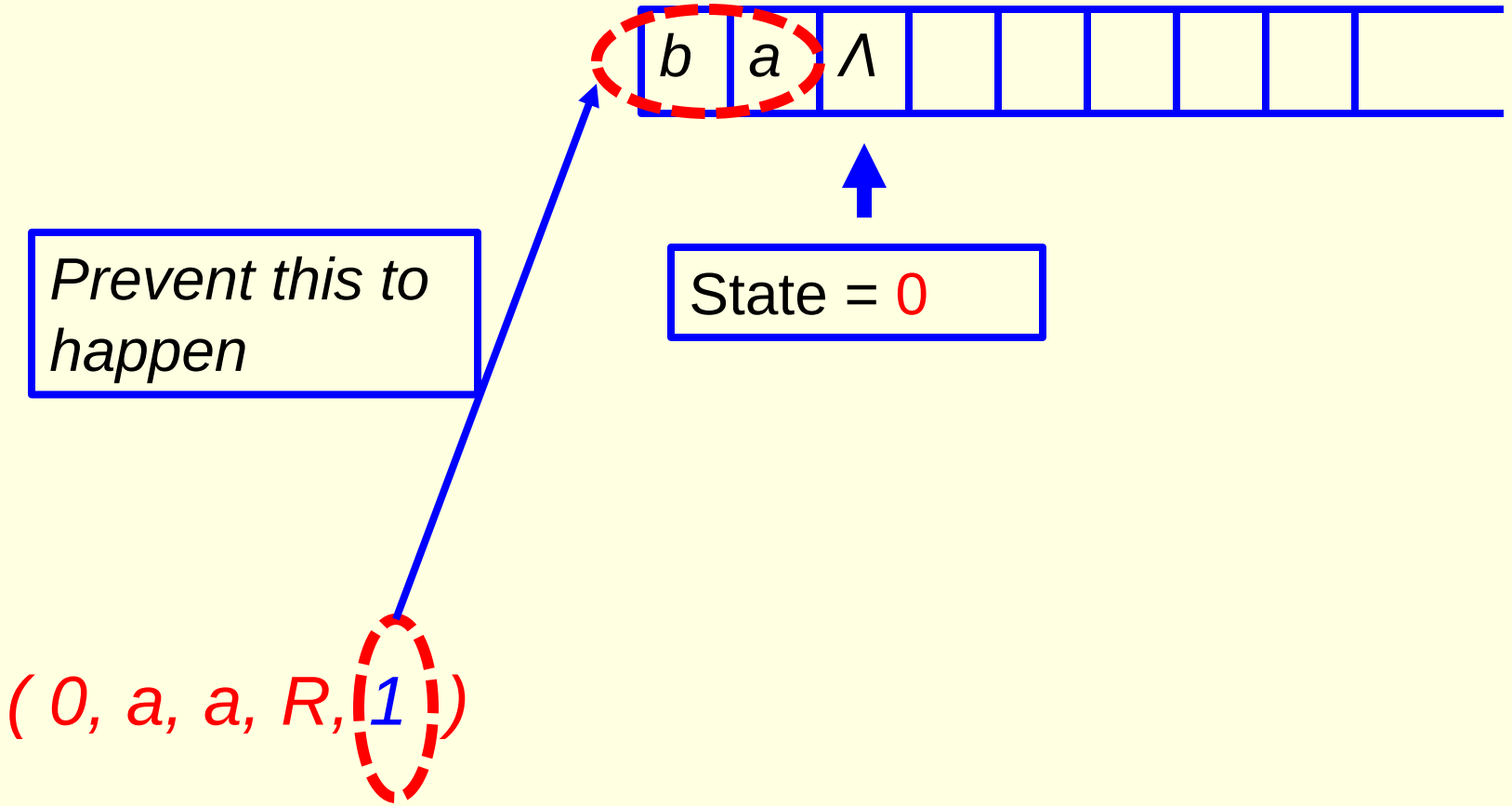


So ba is accepted

Question: *would*

(0, a, a, R, 0) *(0, b, b, R, 0)* *(0, Λ, Λ, S, halt)*

work? ***NO***



8. Turing Machines

Quiz. Construct a TM to accept the language

$$\{ ab^n a \mid n \in \mathbf{N} \}$$

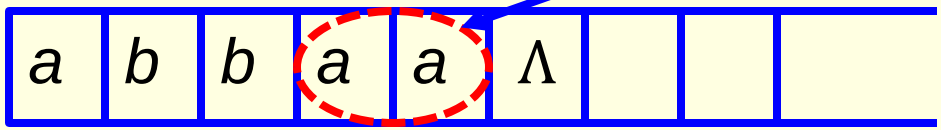
Solution.

(0, a, a, R, 1) *(1, b, b, R, 1)* *(1, a, a, R, 2)*

(2, Λ, Λ, S, halt).

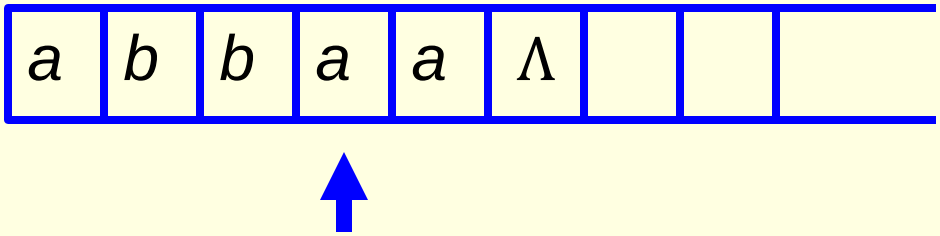
Why?

To prevent this to happen

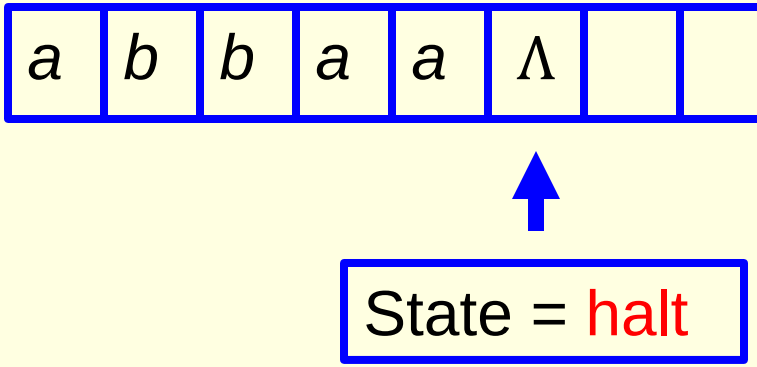


Would { (0, *a*, *a*, *R*, 1) , (1, *b*, *b*, *R*, 1) ,
(1, *a*, *a*, *R*, 1), (1, *Λ*, *Λ*, *S*, *halt*) } *work?* **NO**

Consider:



With the above instruction set, we would have:



So abbaa would be accepted

8. Turing Machines

Quiz. Construct a TM to accept the language

$$\{ aab^n a \mid n \in \mathbf{N} \}$$

Would the following TM work?

(0, a, a, R, 1) *(1, a, a, R, 2)* *(2, b, b, R, 2)*

(2, a, a, R, 3) *(3, Λ, Λ, S, halt).*

8. Turing Machines

Quiz. Construct a TM to accept the language

$$\{ aab^n aa \mid n \in \mathbf{N} \}$$

Would the following TM work?

(0, a, a, R, 1) *(1, a, a, R, 2)* *(2, b, b, R, 2)*

(2, a, a, R, 3) *(3, a, a, R, 4)* *(4, Λ, Λ, S, halt).*

HW Question. (a) What TM would accept the language

$$\{ b^n a \mid n \in \mathbf{N} \} \ ?$$

HW Question. (b) What TM would accept the language

$$\{ b^n aa \mid n \in \mathbf{N} \} \ ?$$

Example. *Find a TM to move any string over {a, b} to the left one cell position. Assume the tape head ends at the left end of any nonempty output string.*

Why do we want to move strings around?

b/c in addition to doing computation, a major function of a computer is to process documents.

To perform operations on a document such as cut/paste, you need to be able to move strings around.

Example. *Find a TM to move any string over {a, b} to the left one cell position. Assume the tape head ends at the left end of any nonempty output string.*

This function is also needed when executing a program.

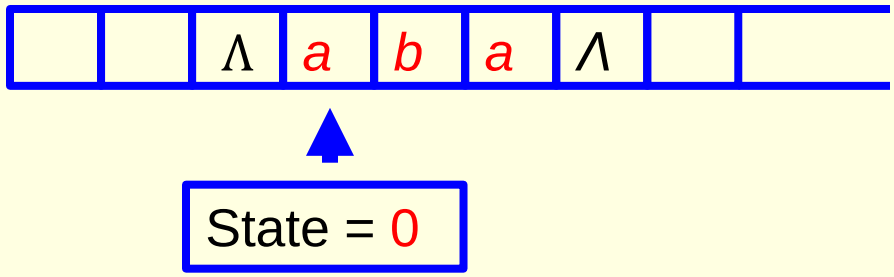
Simplest example: 'cout' command in C++

cout << "Input the next integer: " << endl;

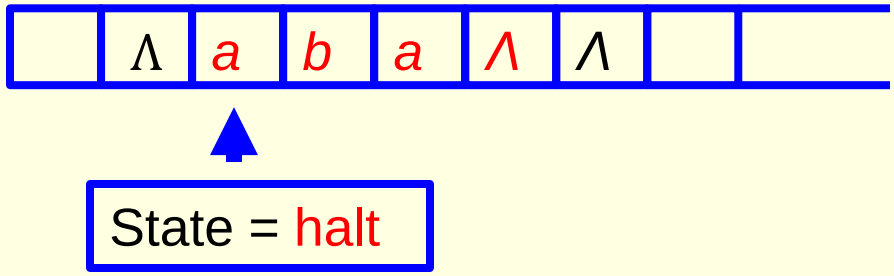
In this case, the string "Input the next integer:" has to be copied and moved to a portion of the tape (memory) reserved as an output queue to be sent to the screen or the printer.

Example. *Find a TM to move any string over {a, b} to the left one cell position. Assume the tape head ends at the left end of any nonempty output string.*

Initially,

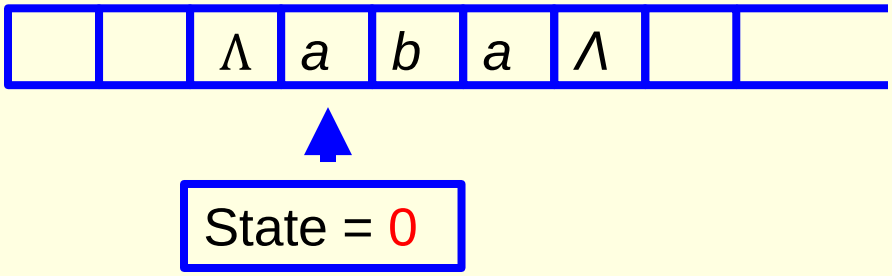


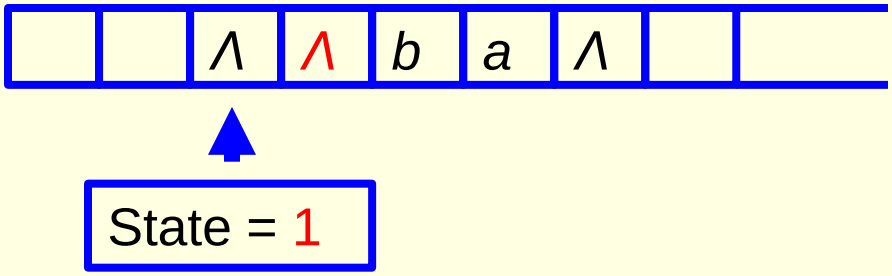
At the end,



Find a or b to move:

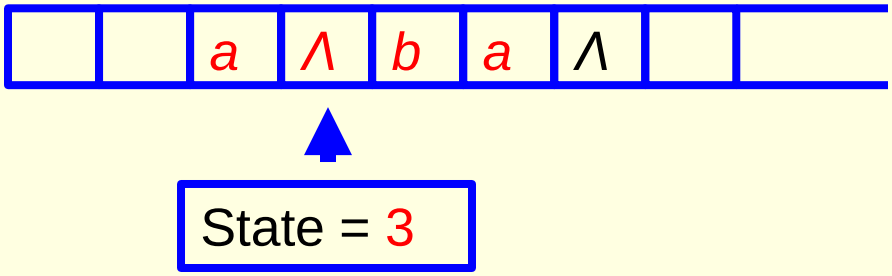
(0, a, Λ, L, 1)	found a
(0, b, Λ, L, 2)	found b
(0, Λ, Λ, L, 4)	no more a's or b's





Write a or b :

(1, Λ, *a*, *R*, 3)	Write *a*
(2, Λ, *b*, *R*, 3)	Write *b*
(3, Λ, Λ, *R*, 0)	Skip Λ



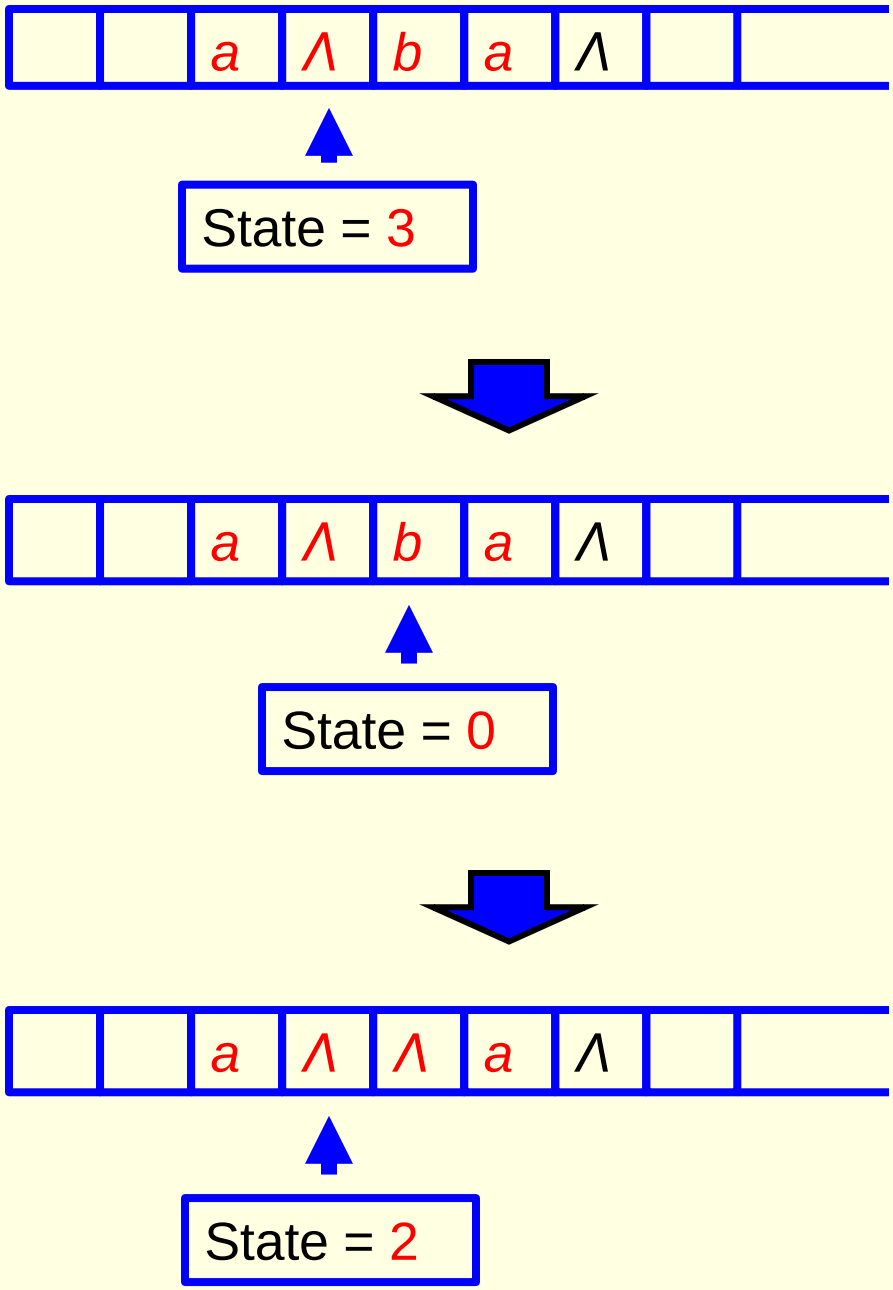
Why do we need a state 3 here?

Find a or b to move:

(0, a, Λ, L, 1)	found a
(0, b, Λ, L, 2)	found b
(0, Λ, Λ, L, 4)	no more a's or b's

Write a or b :

(1, Λ, *a*, *R*, 3)	Write *a*
(2, Λ, *b*, *R*, 3)	Write *b*
(3, Λ, Λ, *R*, 0)	Skip Λ

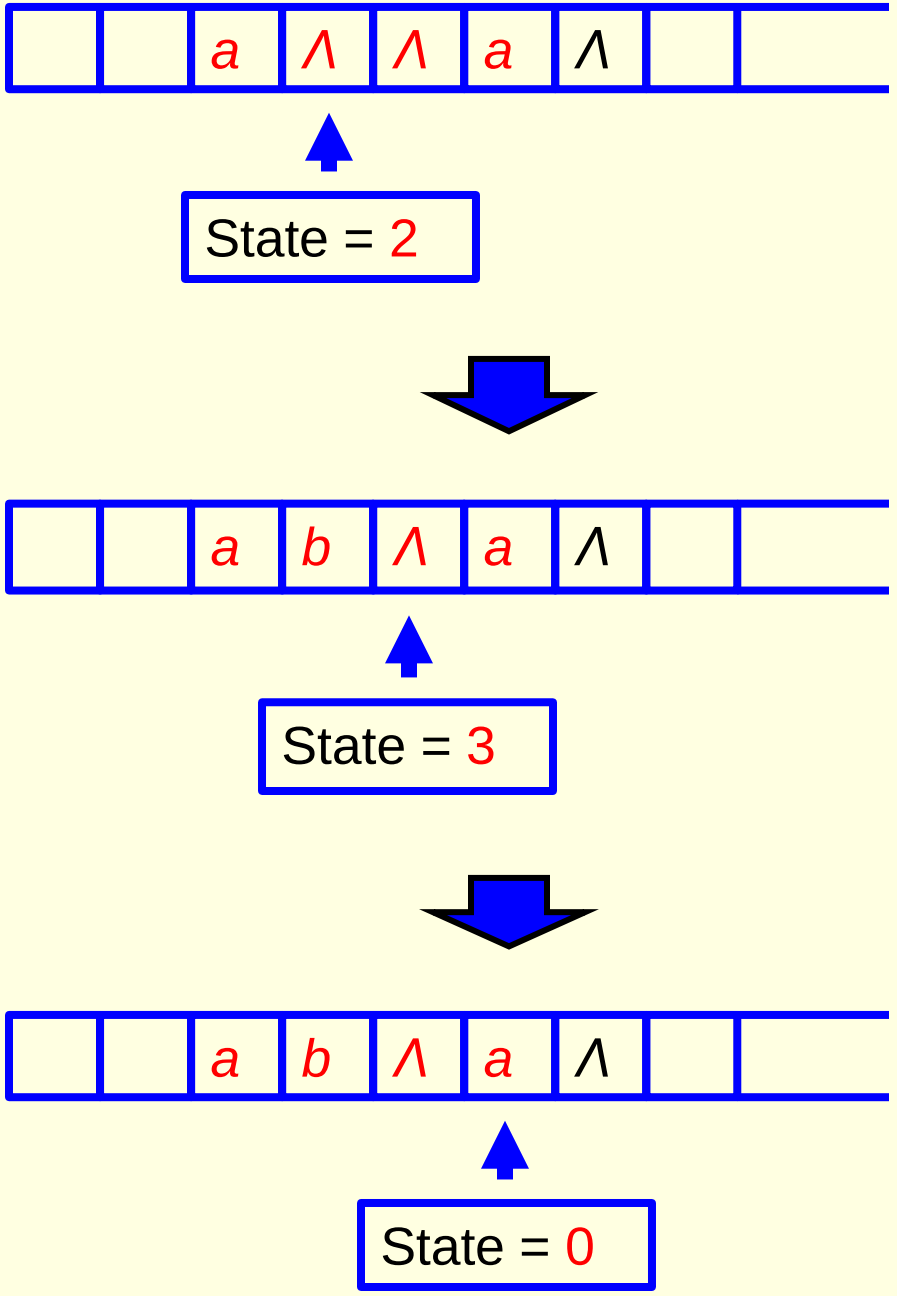


Find a or b to move:

(0, a, Λ, L, 1)	found a
(0, b, Λ, L, 2)	found b
(0, Λ, Λ, L, 4)	no more a's or b's

Write a or b :

(1, Λ, *a*, *R*, 3)	Write *a*
(2, Λ, *b*, *R*, 3)	Write *b*
(3, Λ, Λ, *R*, 0)	Skip Λ

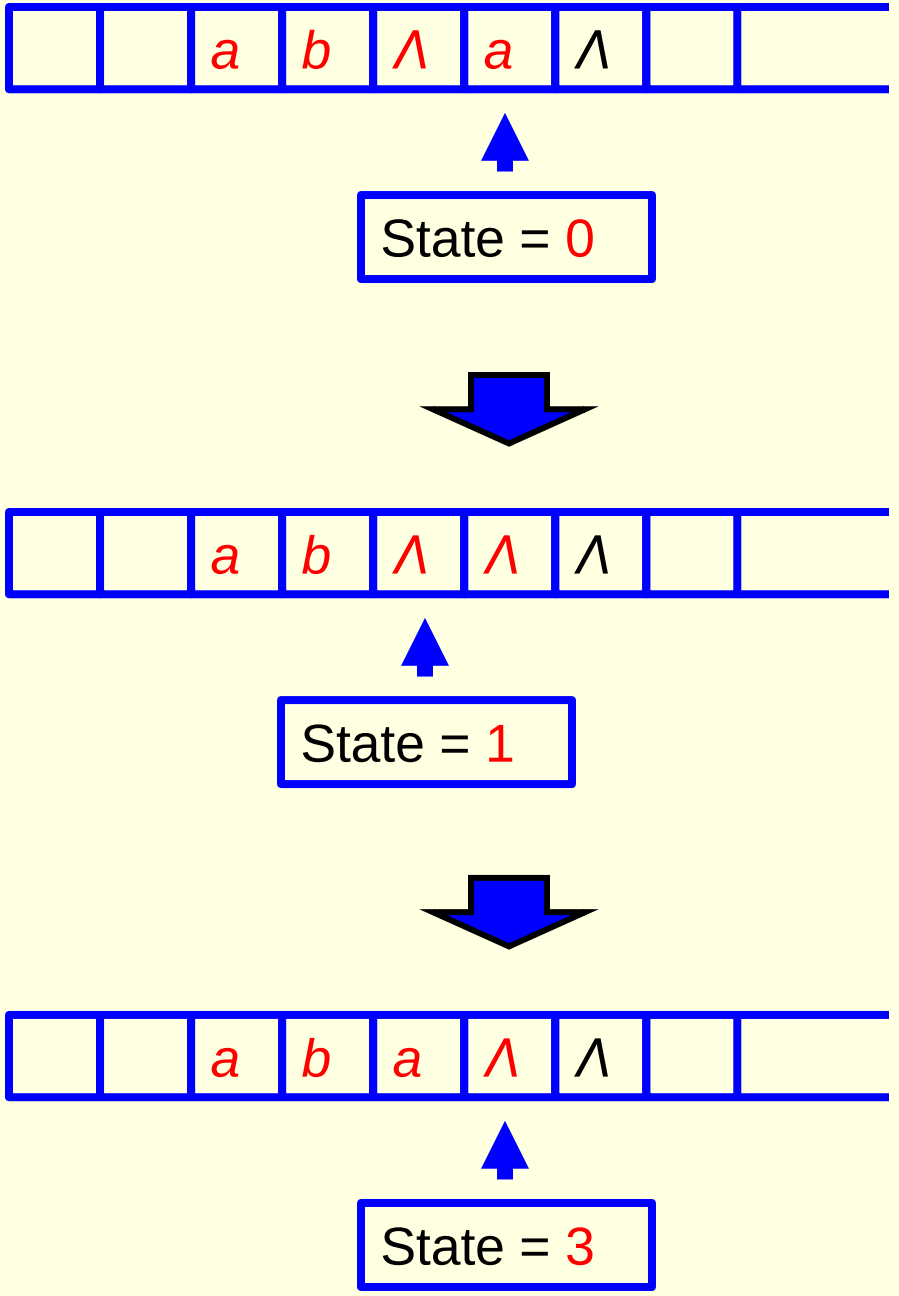


Find a or b to move:

(0, a, Λ, L, 1)	found a
(0, b, Λ, L, 2)	found b
(0, Λ, Λ, L, 4)	no more a's or b's

Write a or b :

(1, Λ, *a*, *R*, 3)	Write *a*
(2, Λ, *b*, *R*, 3)	Write *b*
(3, Λ, Λ, *R*, 0)	Skip Λ



Find a or b to move:

(0, a, Λ, L, 1) found a

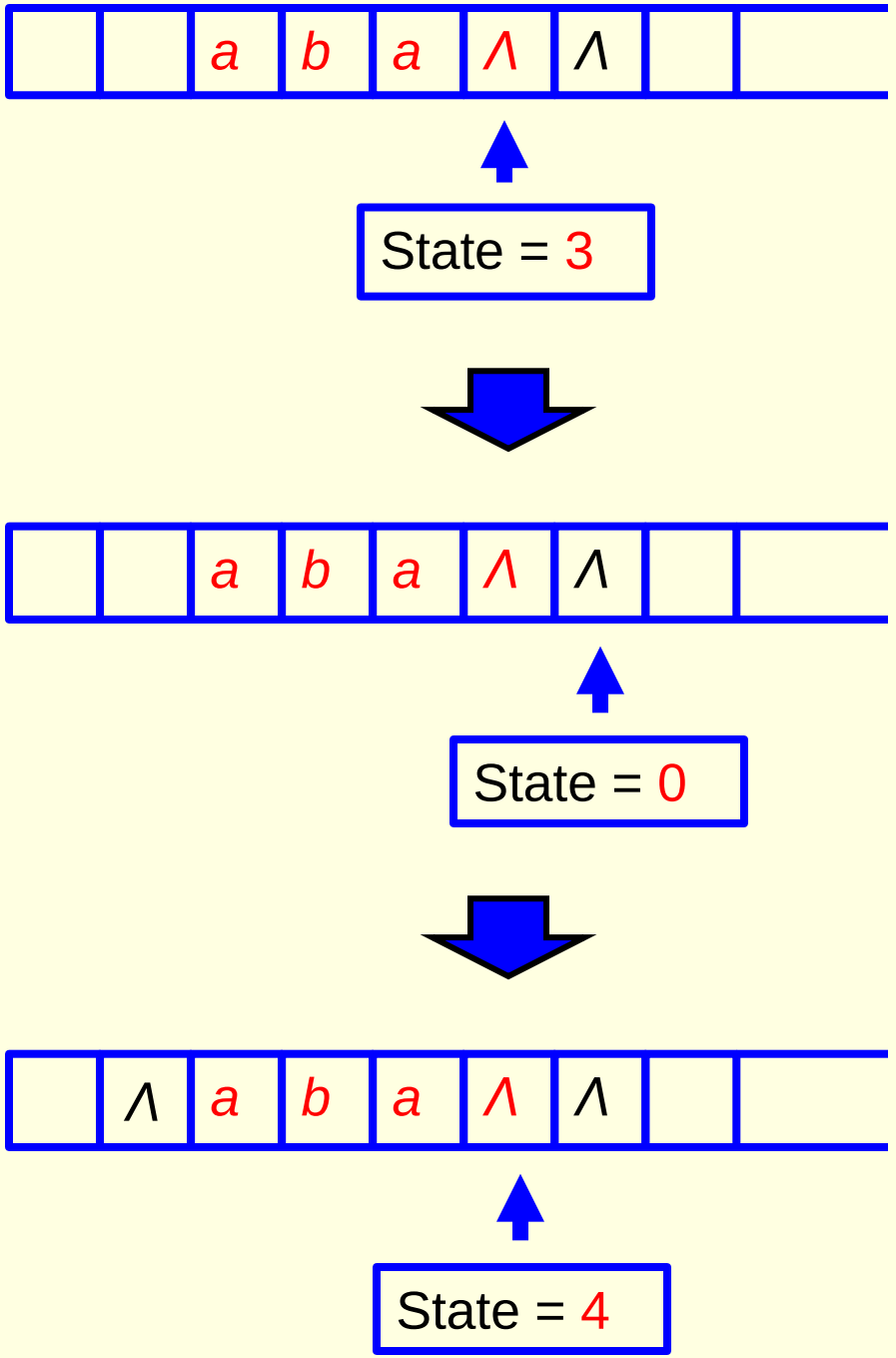
(0, b, Λ, L, 2) found b

(0, Λ, Λ, L, 4) no more a's or b's

Write a or b :

(1, Λ, *a*, *R*, 3)	Write *a*
(2, Λ, *b*, *R*, 3)	Write *b*
(3, Λ, Λ, *R*, 0)	Skip Λ

Why do we need a state 4 here?



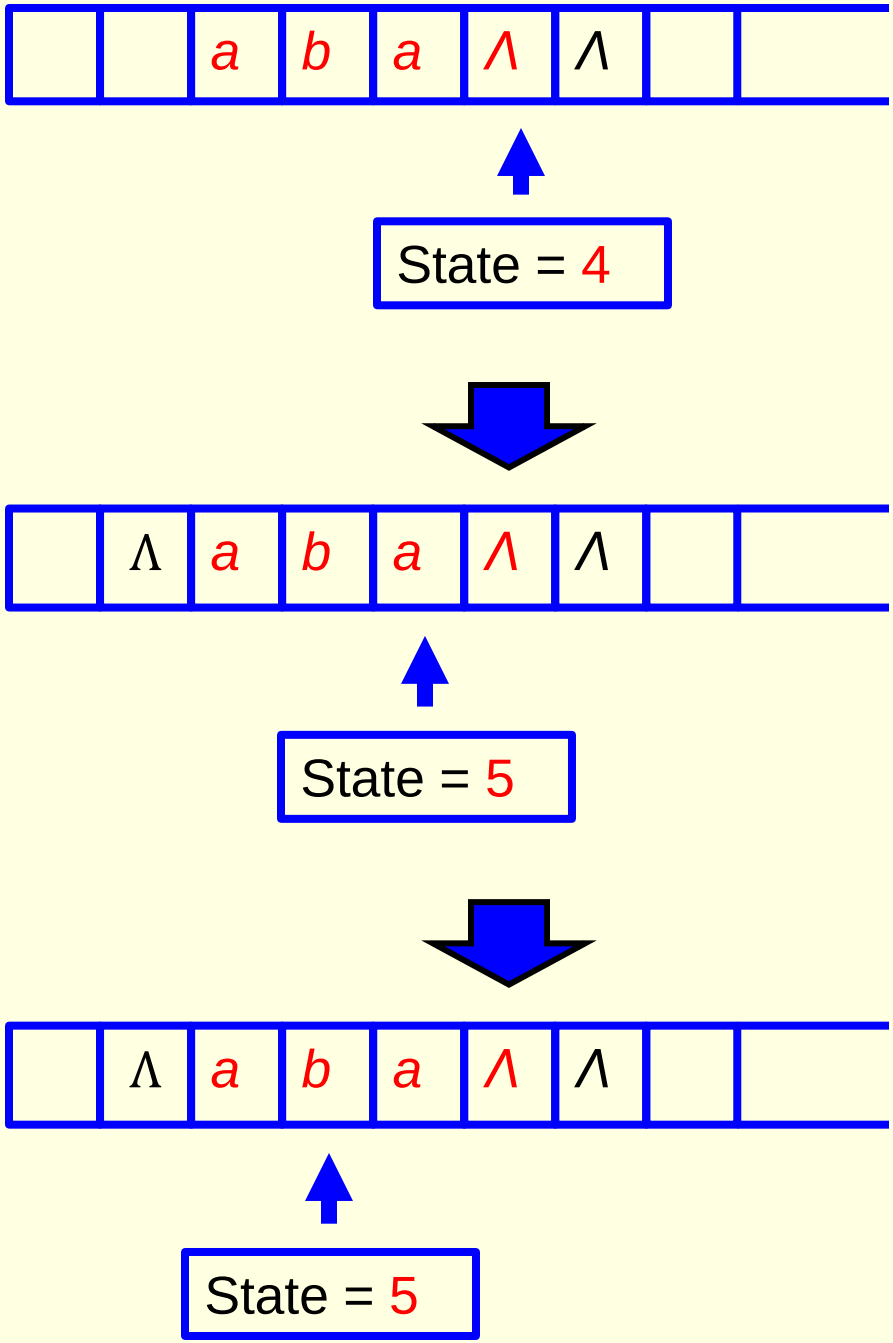
Move to left end of output:

$(4, \Lambda, \Lambda, L, 5)$

$(5, a, a, L, 5)$

$(5, b, b, L, 5)$

$(5, \Lambda, \Lambda, R, \text{halt}).$



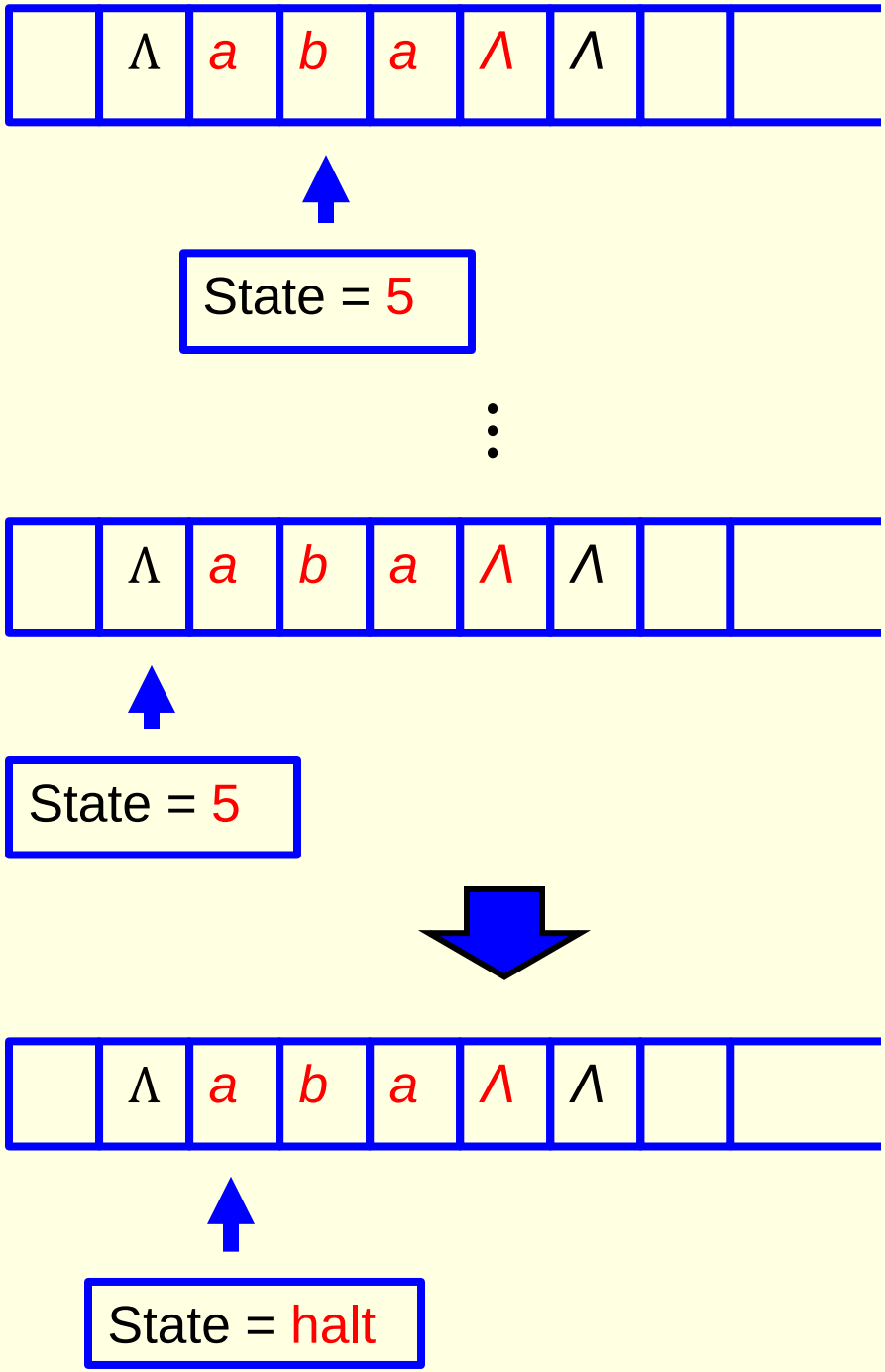
Move to left end of output:

$(4, \Lambda, \Lambda, L, 5)$

$(5, a, a, L, 5)$

$(5, b, b, L, 5)$

$(5, \Lambda, \Lambda, R, \text{halt})$.



Solution. Hence, we have

Find a or b to move:

(0, a, Λ, L, 1) found a

(0, b, Λ, L, 2) found b

(0, Λ, Λ, L, 4) no more a's or b's

Write *a* or *b*:

(1, Λ, *a*, *R*, 3) Write *a*

(2, Λ, *b*, *R*, 3) Write *b*

(3, Λ, Λ, *R*, 0) Skip Λ

Move to left end of output:

(4, Λ, Λ, *L*, 5)

(5, *a*, *a*, *L*, 5)

(5, *b*, *b*, *L*, 5)

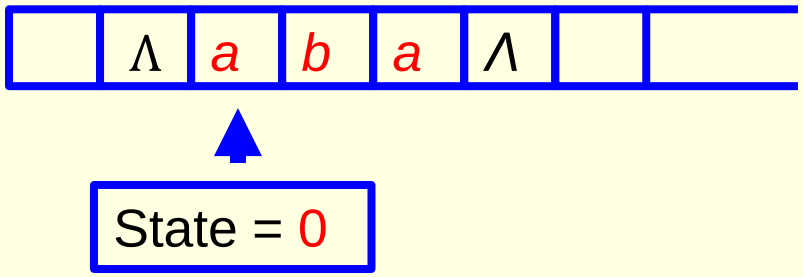
(5, Λ, Λ, *R*, halt).

Question. How would you construct a Turing Machine to move an input string over {a, b} to the left two cells position ?

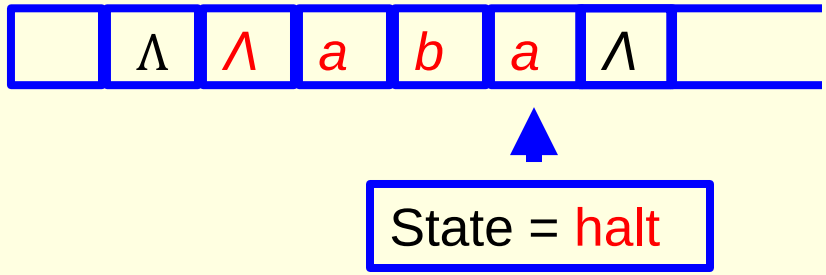
One Solution: use the above process twice.

Question. *Find a TM to move any string over {a, b} to the right one cell position. The tape head initially is at the left end of the input string. The tape head ends at the right end of the output string.*

Initially,

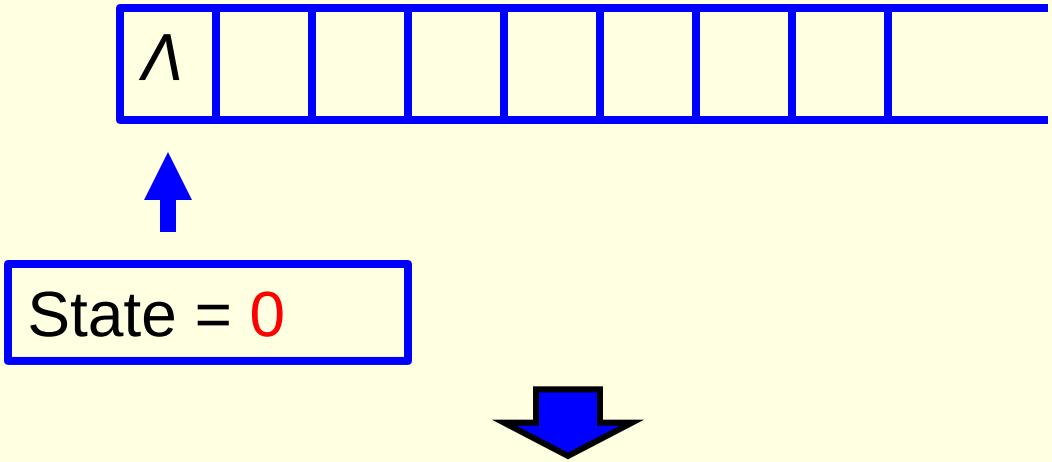


At the end,

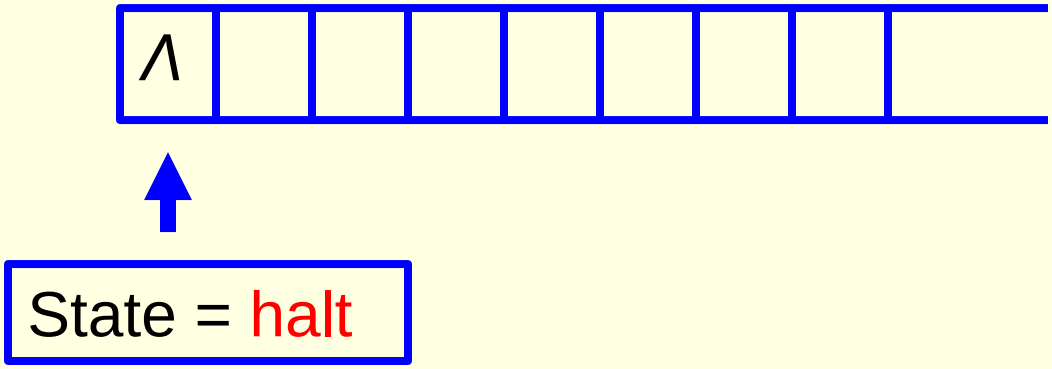


A TM moves a string to the right one cell position

However,



Nothing needs to be done in this case



So, we need
(0, Λ, Λ, S, halt)

A TM moves a string to the right one cell position

(0, a, Λ, R, ____)	found an a
(0, b, Λ, R, ____)	found a b
(0, Λ, Λ, S, *halt*)	Done

(1, a, a, R, ____)	found an a & to write an a
(1, b, a, R, ____)	found a b & to write an a
(1, Λ, a, S, *halt*)	Done

(2, a, b, R, ____)	found an a & to write a b
(2, b, b, R, ____)	found a b & to write a b
(2, Λ, b, S, *halt*)	Done

A TM moves a string to the right one cell position

(0, a, Λ, R, *1*)	found an a
(0, b, Λ, R, *2*)	found a b
(0, Λ, Λ, S, *halt*)	Done

(1, a, a, R, *1*)	found an a & to write an a
(1, b, a, R, *2*)	found a b & to write an a
(1, Λ, a, S, *halt*)	Done

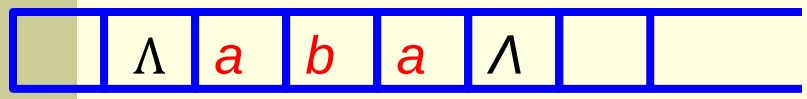
(2, a, b, R, *1*)	found an a & to write a b
(2, b, b, R, *2*)	found a b & to write a b
(2, Λ, b, S, *halt*)	Done

A TM moves a string to the right one cell position

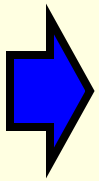
(0, a, Λ, R, 1) found an a

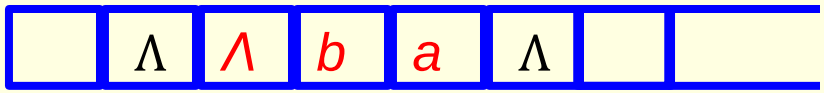
(0, b, Λ, R, 2) found a b

(0, Λ, Λ, S, halt) Done



State = **0**

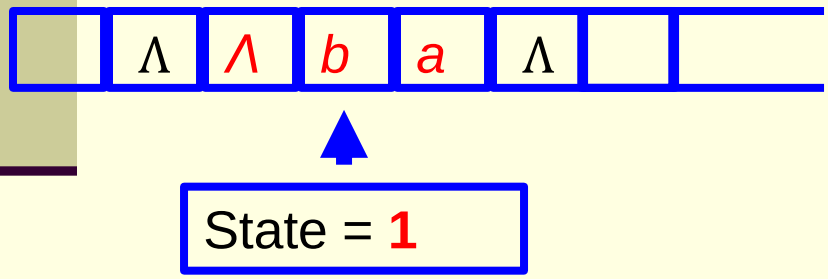


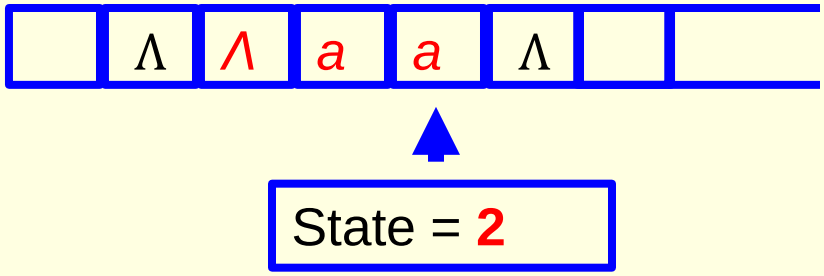


State = **1**

A TM moves a string to the right one cell position

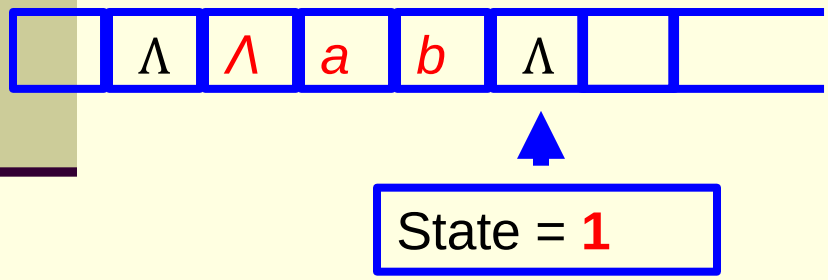
(1, a, a, R, *1*)	found an a & to write an a
(1, b, a, R, *2*)	found a b & to write an a
(1, Λ, a, S, *halt*)	Done

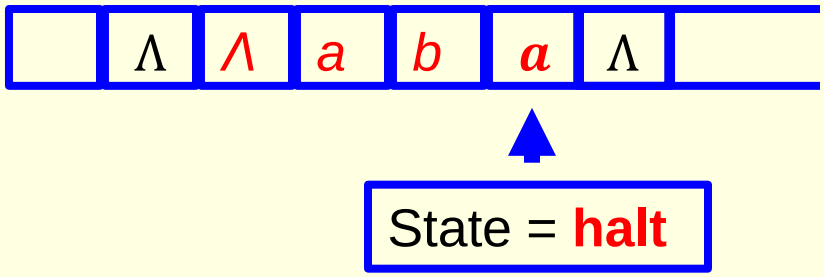




A TM moves a string to the right one cell position

(1, a, a, R, *1*)	found an a & to write an a
(1, b, a, R, *2*)	found a b & to write an a
(1, Λ, a, S, *halt*)	Done





Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Solution. Let start state be 0.

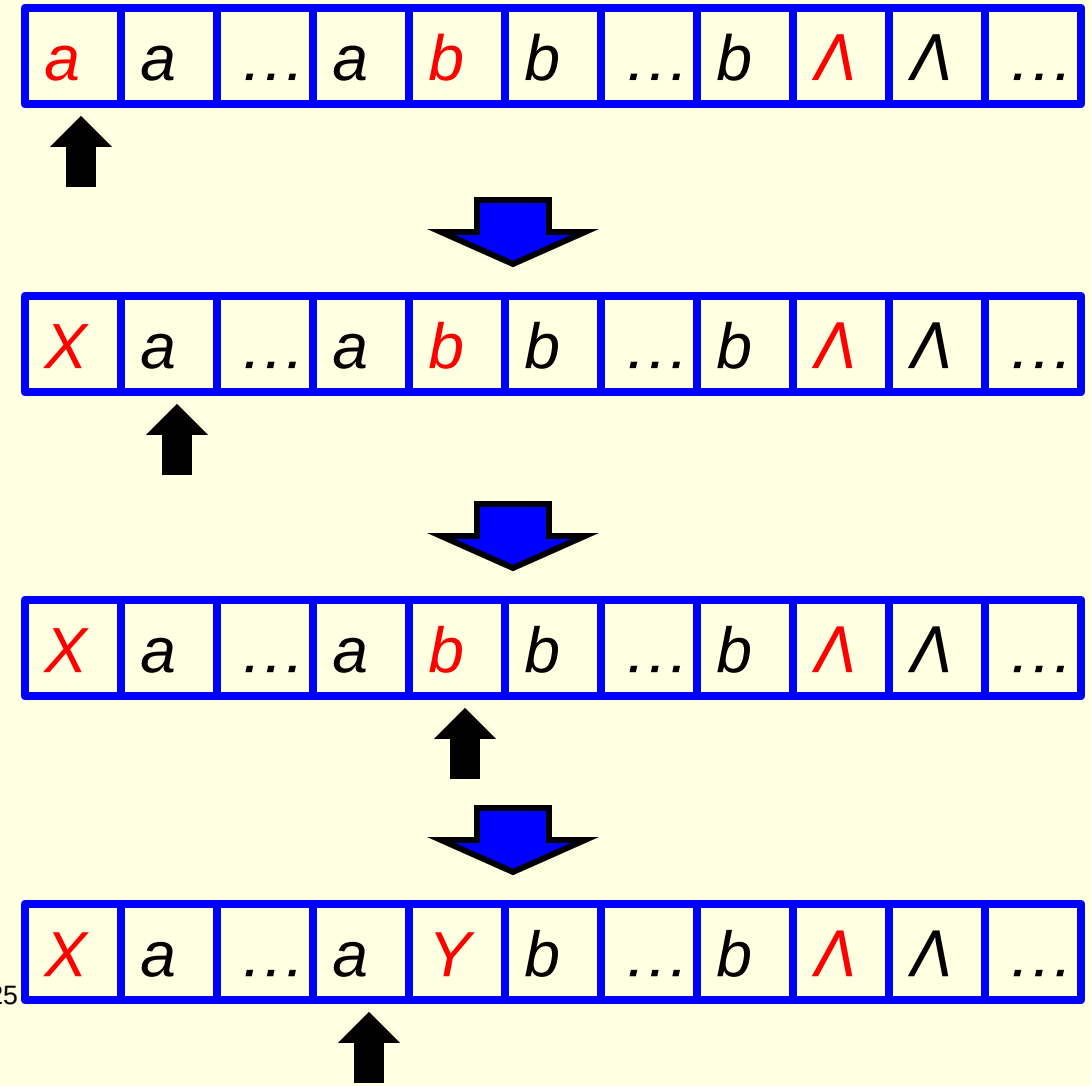
The idea is to see whether *a* is in the current cell.

If so, write *X* and scan right looking for a *b* to pair with it by replacing the b by *Y*.

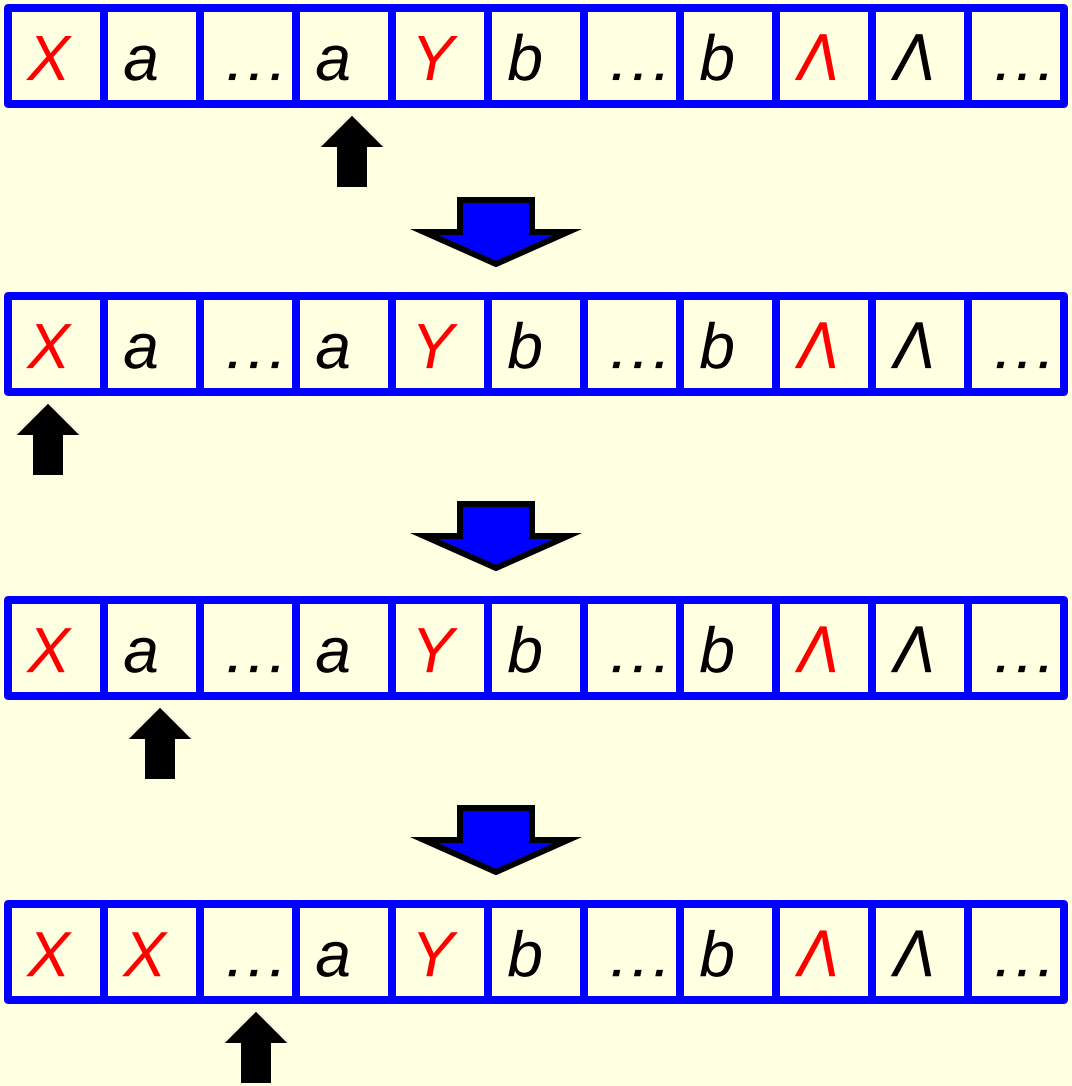
Repeat the process iteratively.

(A TM does not have a stack, but it can build an implicit stack. Actually as many implicit stacks as possible. How?)

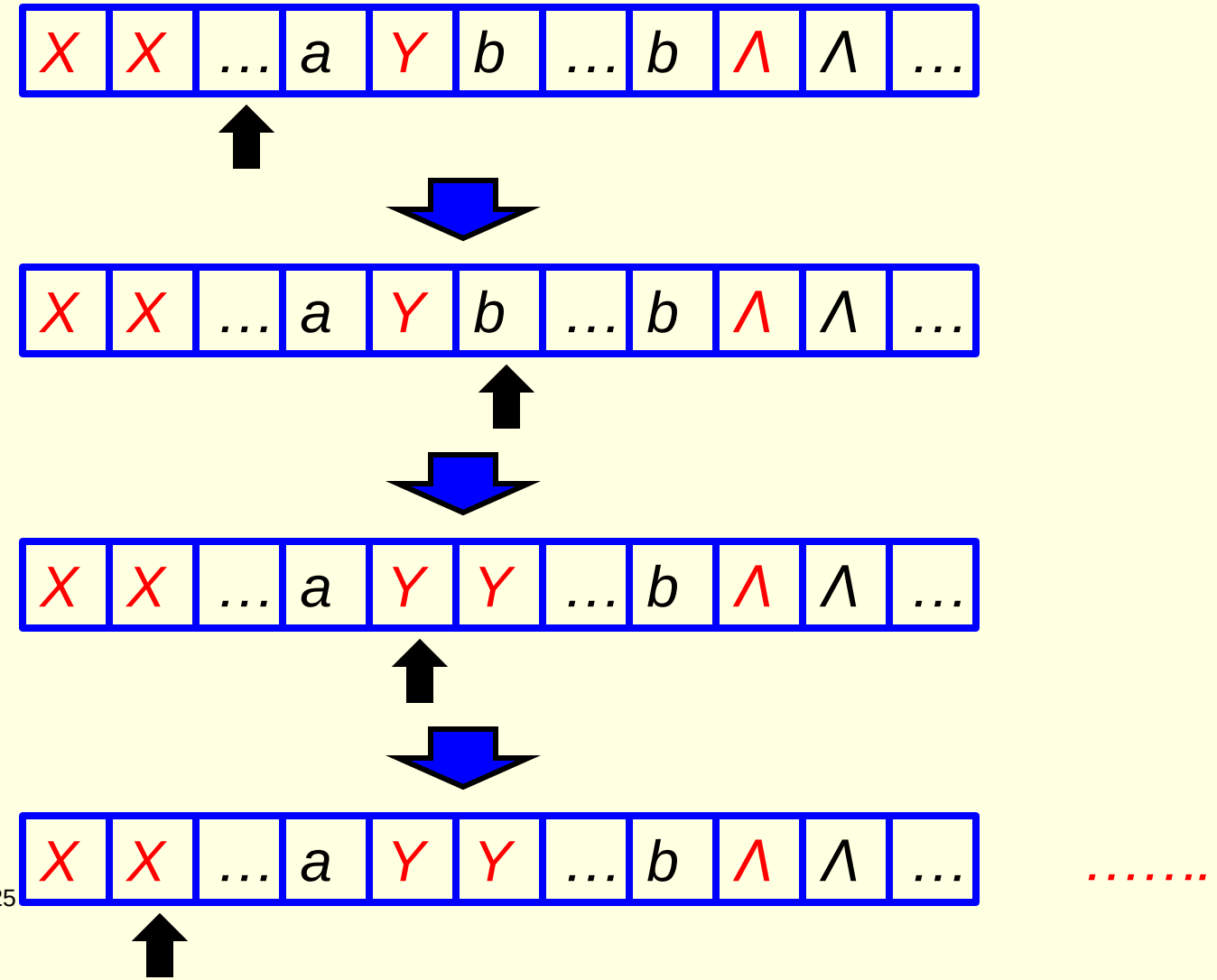
Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.



Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.



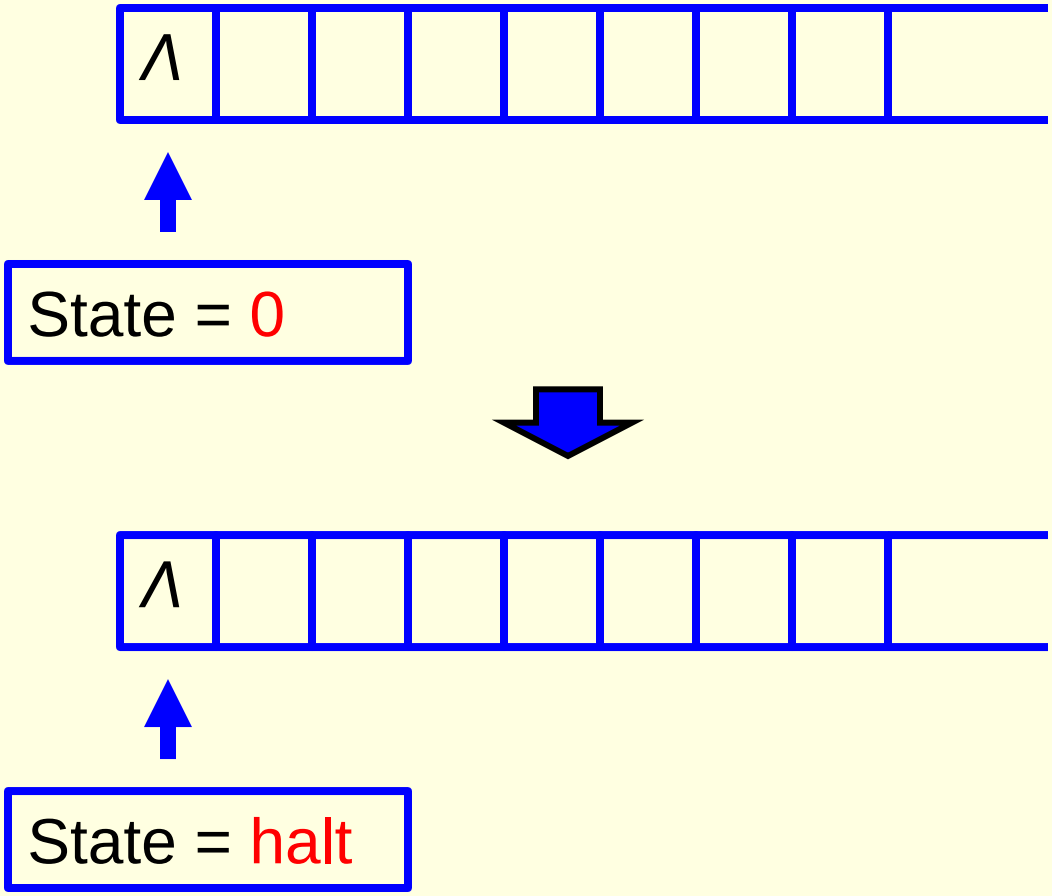
Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.



Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

To accept Λ, we need :

(0, Λ, Λ, S, halt)



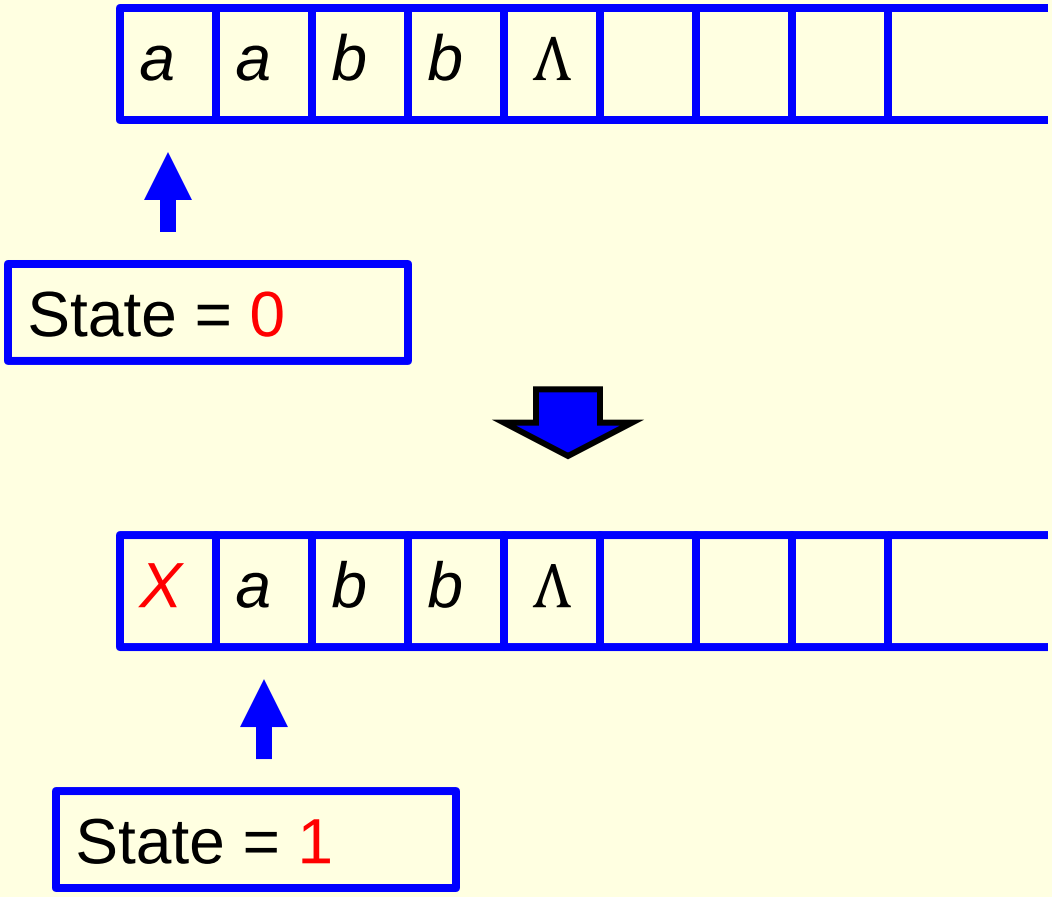
So Λ is accepted

Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

For a non-empty string, mark a with X :

(0, a, X, R, 1)

0 means we are looking for an 'a' to mark.
1 means we are looking for a 'b' to mark.

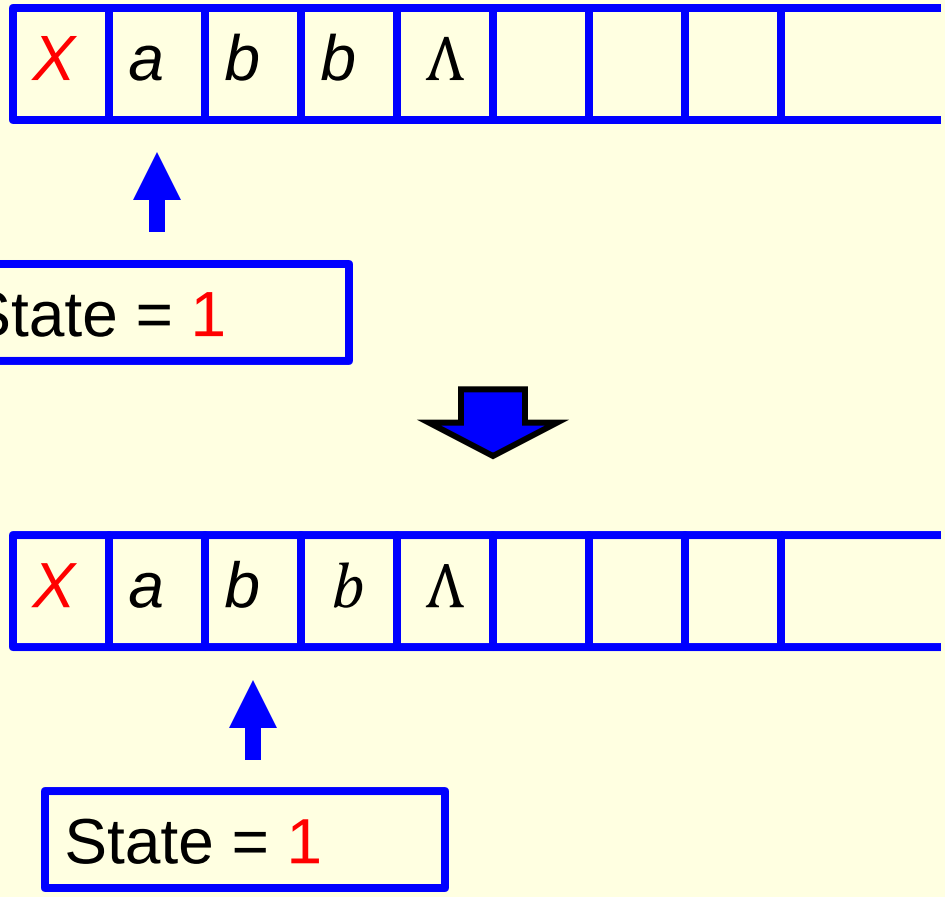


Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Scan right looking for a "b" to pair with the marked "a" :

(1, a, a, R, 1)

We keep the state 1 to indicate a "b" has not been reached yet

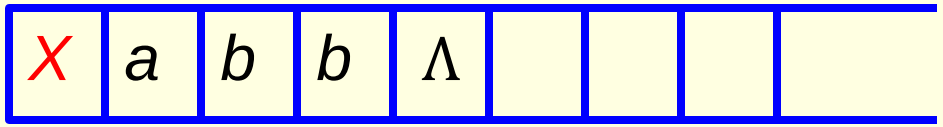


Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

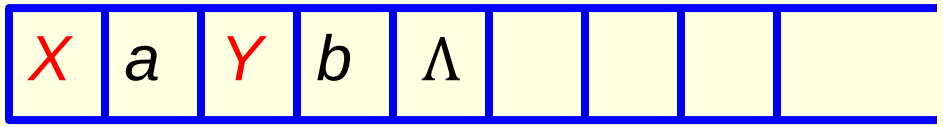
b found, marked.
Scan back left
looking for X :

(1, b, Y, L, 2)

A b has been found to pair with the a just marked. Now go back to get another a to mark



State = 1



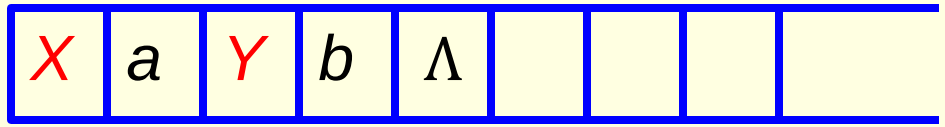
State = 2

Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

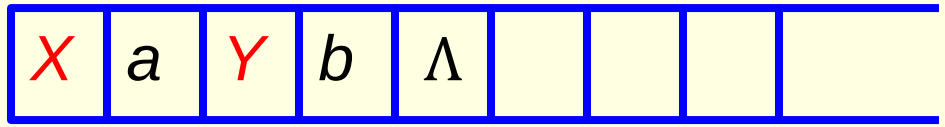
Scan back left looking for X:

(2, a, a, L, 2)

X has not been found yet



State = 2

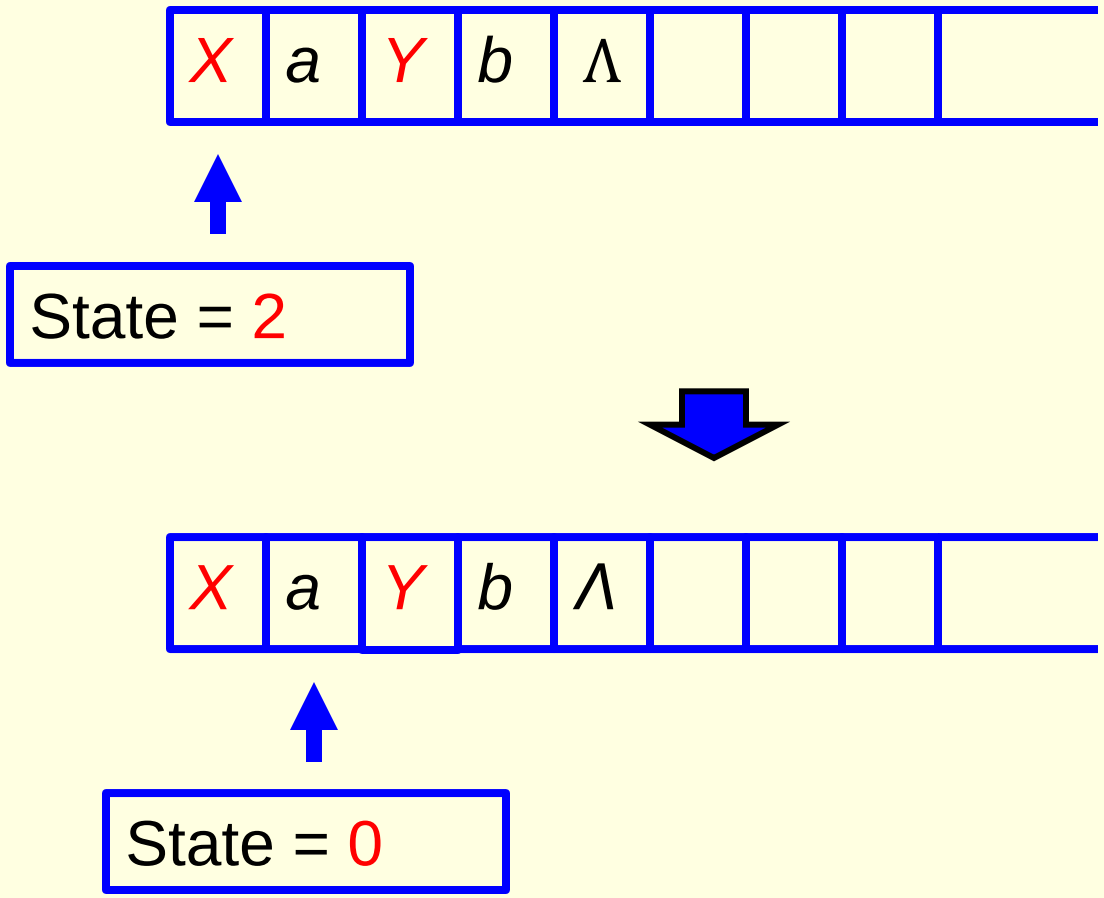


State = 2

Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

X found. Turn around. Ready to mark the next a:

(2, X, X, R, 0)

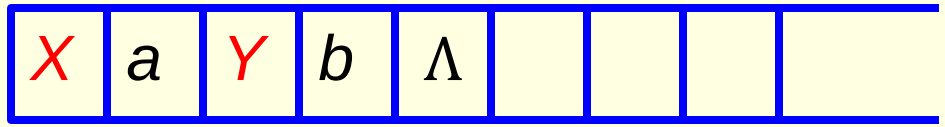


X has just been found. Turn around. Ready to mark another a, the one to the right of this X.

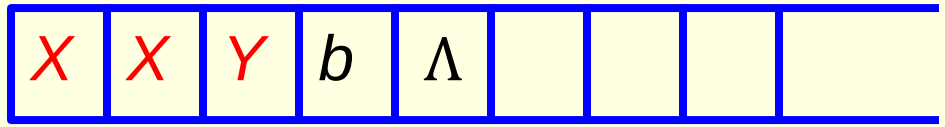
Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Mark next a with X:

(0, a, X, R, 1)



State = 0



State = 1

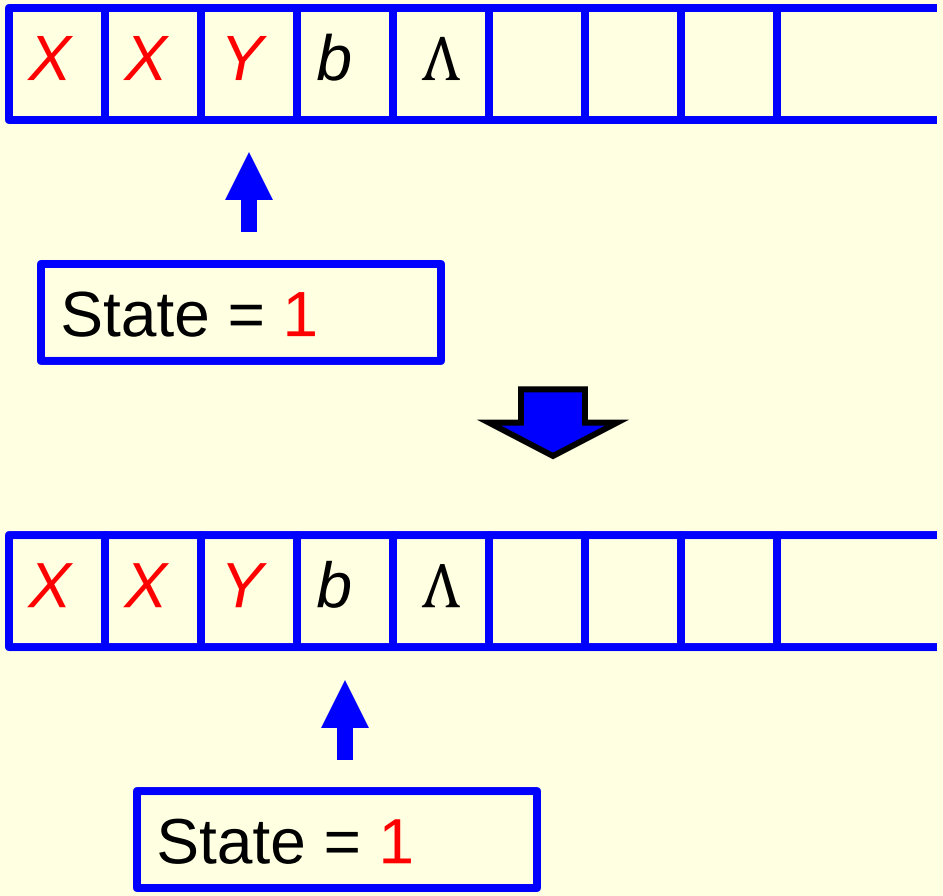
Mark another a.
Now look for a b to pair with it.

Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Looking for a b to pair with the a just marked :

(1, Y, Y, R, 1)

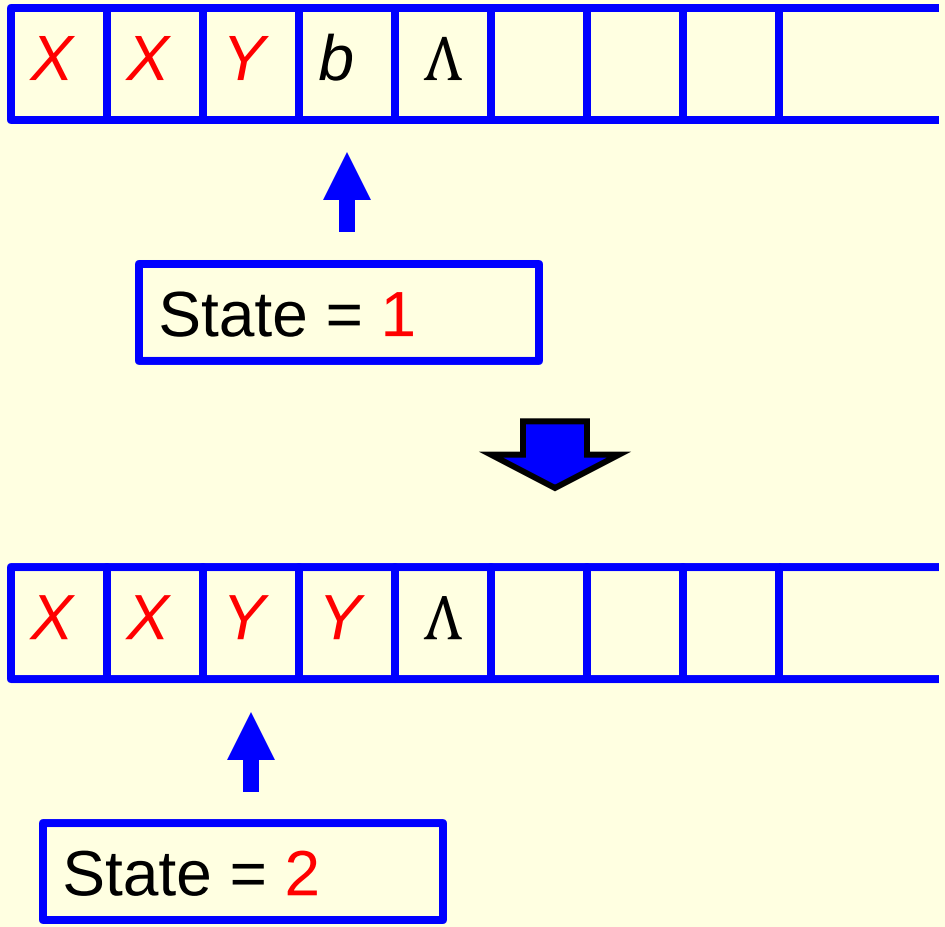
Still looking, but we are one cell closer to that b now.



Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Next b found, marked. Scan back left looking for X:

(1, b, Y, L, 2)



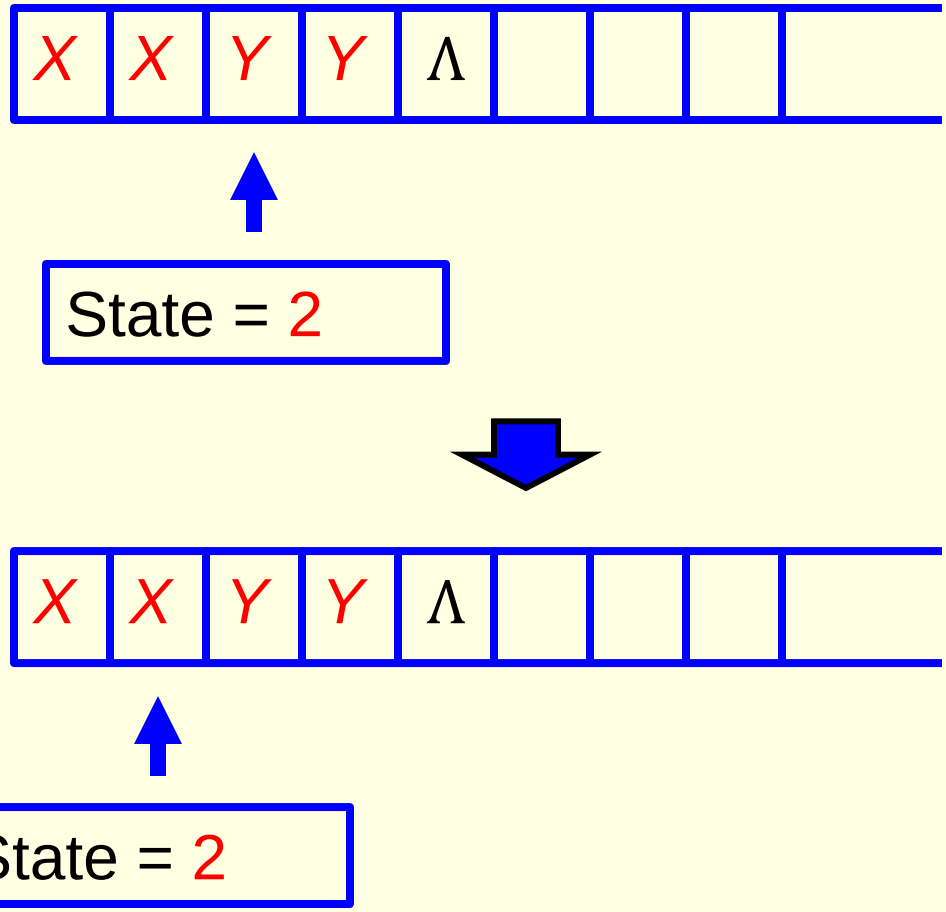
A b has just been marked with Y. Ready to go back to find another a to mark

Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Scan back left looking for X:

(2, Y, Y, L, 2)

X has not been found yet

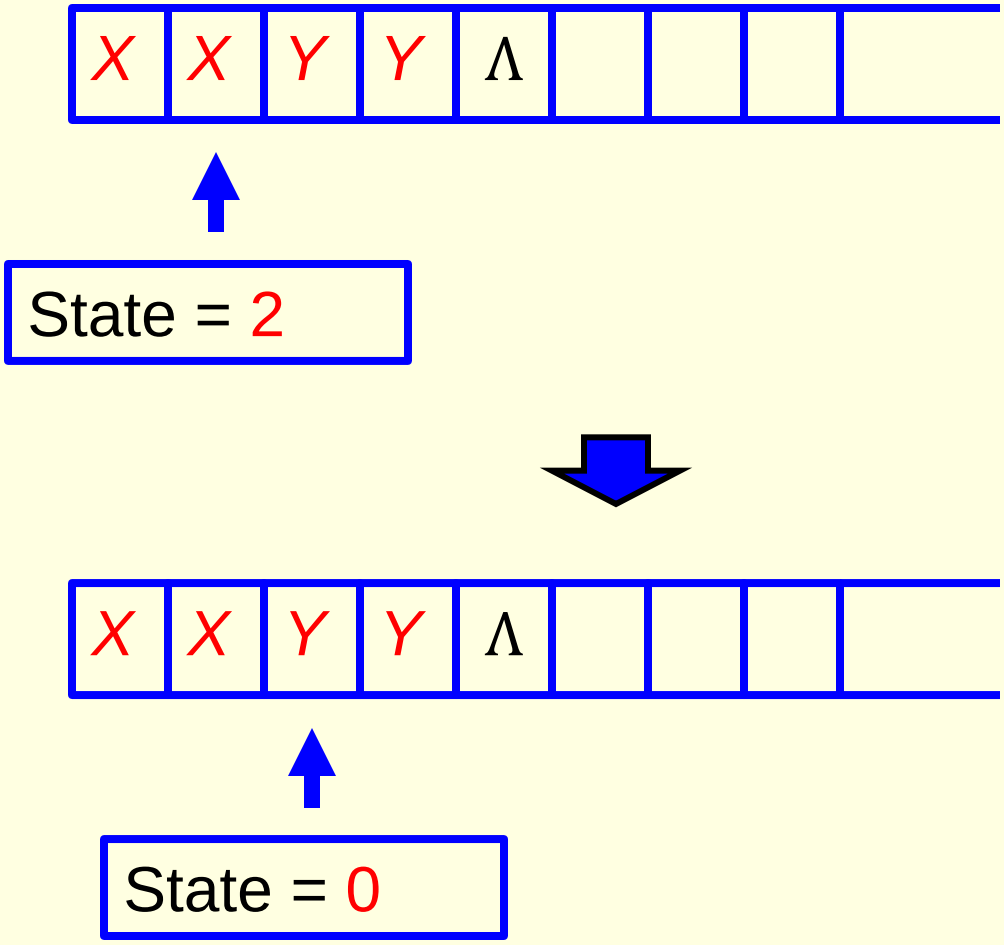


Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

X found. Turn around. Ready to mark the next a:

(2, X, X, R, 0)

Looking for another a to mark

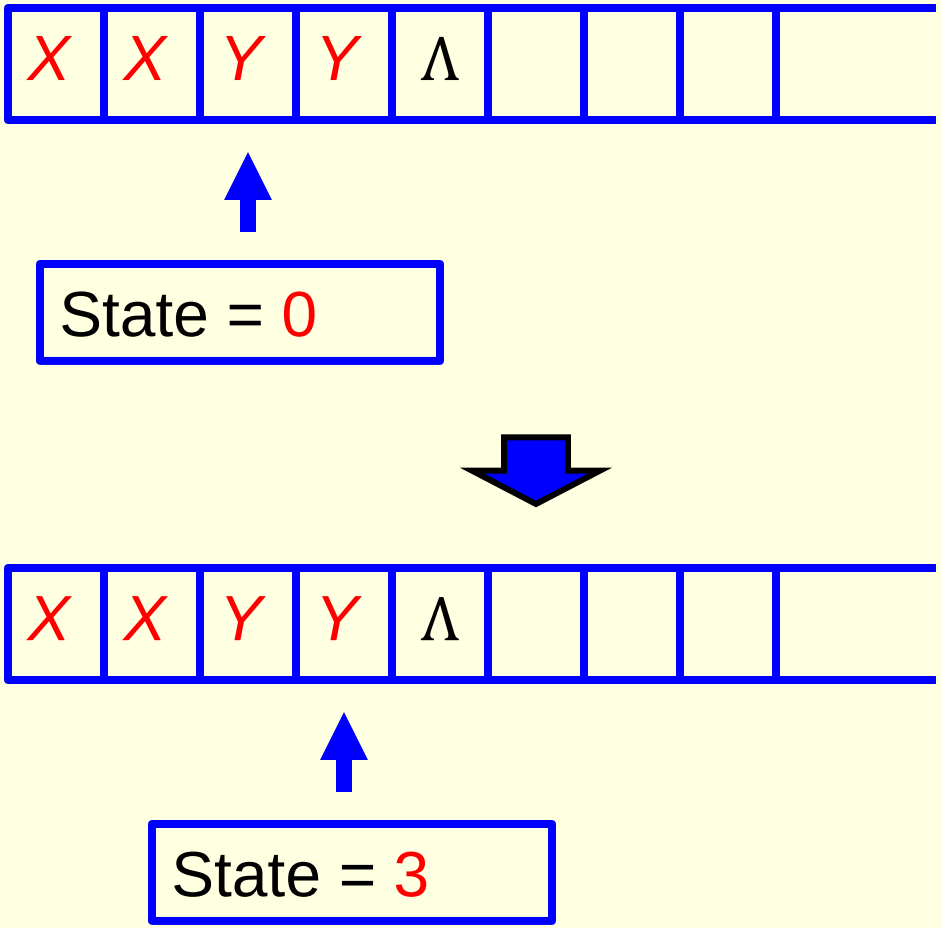


Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

No more a's :

(0, Y, Y, R, 3)

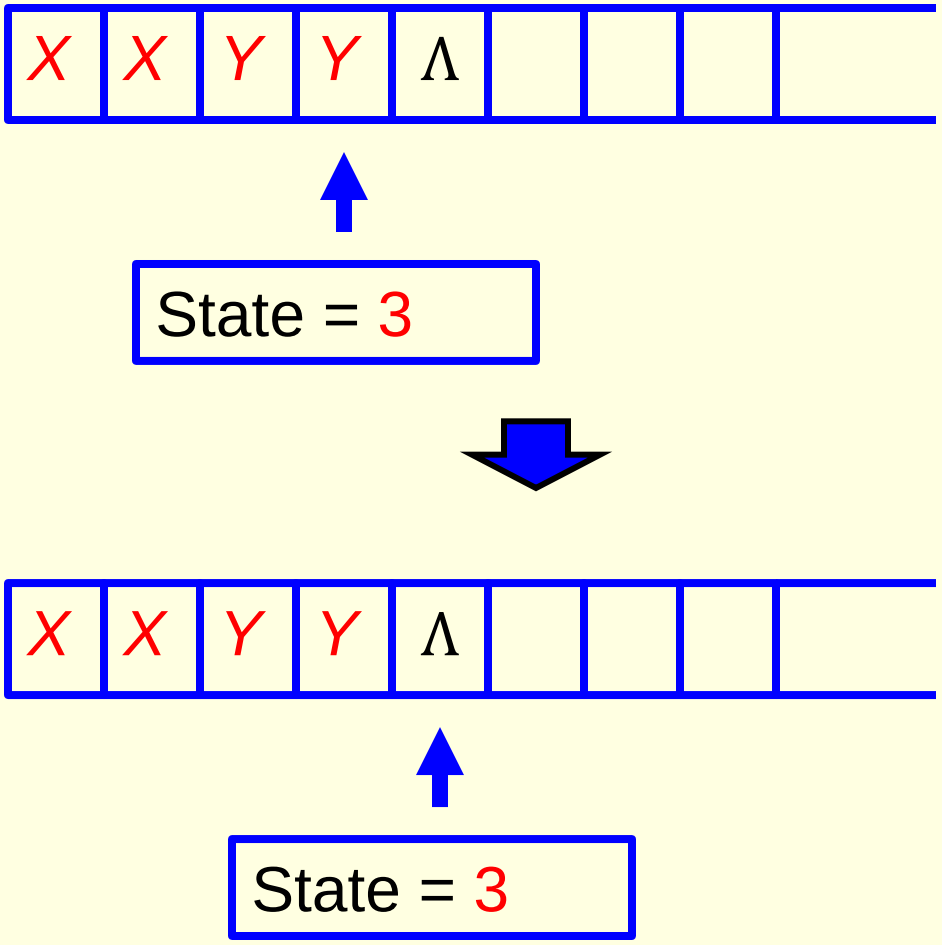
When we reach an Y in state 0, it means there is no more a's to mark



Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Scan right looking for Λ and halt:

(3, Y, Y, R, 3)



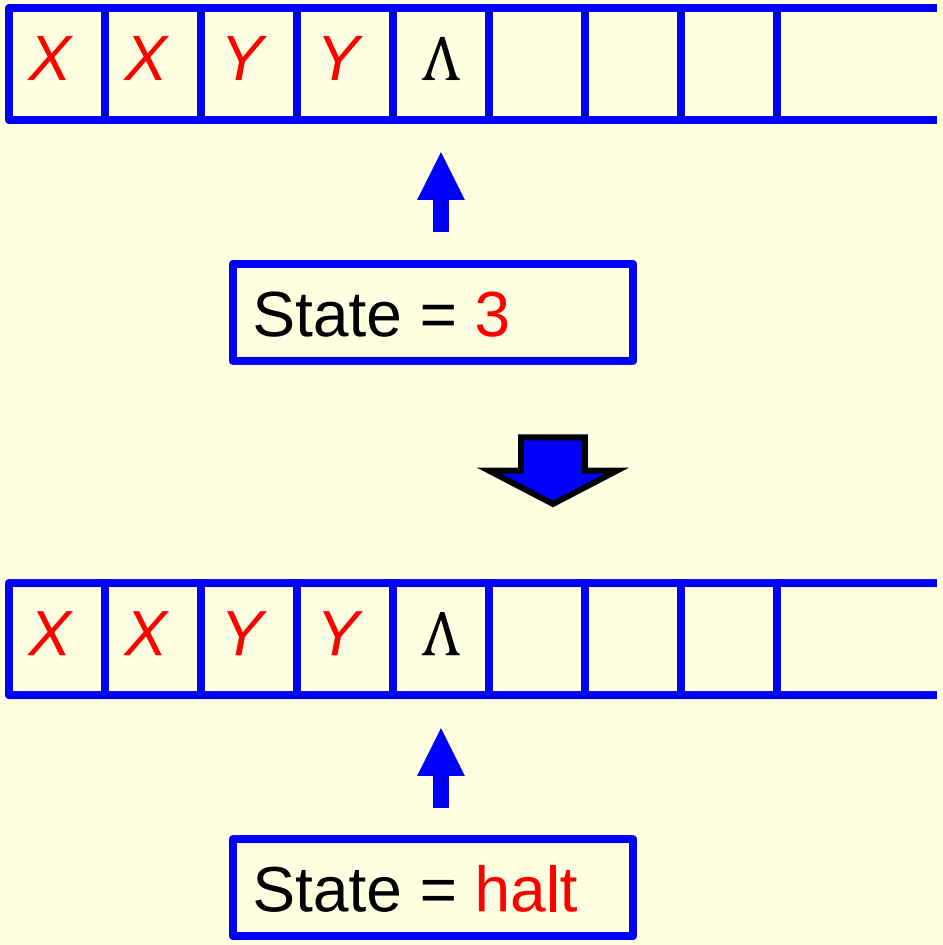
There will only be Y's left in the remaining part of the string before we reach Λ if the string has the same number of a's and b's.

Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Scan right looking for Λ and halt:

(3, Λ, Λ, S, halt)

String accepted



Example. Find a TM to accept $\{ a^n b^n \mid n \in \mathbf{N} \}$.

Hence, we have

(0, Λ, Λ, S, halt) accept Λ

(0, a, X, R, 1) mark a with X

(0, Y, Y, R, 3) no more a's

Scan back left looking for X:

(2, a, a, L, 2) X not found yet

(2, Y, Y, L, 2) X not found yet

(2, X, X, R, 0) X found, turn around

Scan right looking for b to pair with a:

(1, a, a, R, 1) b not found yet

(1, Y, Y, R, 1) b not found yet

(1, b, Y, L, 2) mark b with Y

Scan right looking for Λ and halt:

(3, Y, Y, R, 3)

(3, Λ, Λ, S, halt)

8. Turing Machines

Implicit stacks

TMs are very powerful.

E.g., the preceding example can be generalized to a TM that accepts the non-context free language

$$\{\ a^n b^n c^n \mid n \in \boldsymbol{N}\ \}$$

(See Example in slide 42 of "Context Free Languages and Pushdown Automata V")

So a TM can handle two stacks. (one for a and one for b)

In fact, a TM can handle any number of stacks.

Quiz. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$

First replace an ‘a’ from front by X, then keep moving right till you find a ‘b’ and replace this ‘b’ by Y.

Again, keep moving right till you find a ‘c’, replace it by Z and move left.

Now keep moving left till you find a X.

When you find it, move a right, then follow the same procedure as above.

8. Turing Machines - some more history

Alan Turing is a genius.

He made important contribution in many areas in a relatively short life (1912–1954)

cryptanalysis, logic, philosophy, mathematical biology, cognitive science, ***artificial intelligence****, artificial life and, most importantly, computer science*

Unfortunately, Alan Turing is a gay.

Back then, being a gay in England was considered a felony, so Alan Turing had to live a secret gay life.

8. Turing Machines - some more history

In 1952, Alan Turing was arrested for having sex with a 19-year old male.

He was given two choices:

go to jail, or

take an experimental hormone "therapy"

to correct his homosexual "PROBLEM".

8. Turing Machines - some more history

He accepted the second choice.

The therapy, however, made him miserable, and his breasts continued to swell.

(and his breasts swelled dramatically).

8. Turing Machines - some more history

In the midst of many groundbreaking works, Alan Turing was found dead by his housekeeper in his bed room one morning in 1954.

There was a bitten apple on the nightstand. 

8. Turing Machines - some more history

The autopsy showed that he was poisoned by cyanide and the apple was soaked in cyanide solvent.

He was 42, only 42, that year.

Four people attended his funeral. One of them was his mom.

8. Turing Machines

Can be used to compute functions

Turing Machines with Output

Specify the form of the output on the tape when the machine halts.

Example.

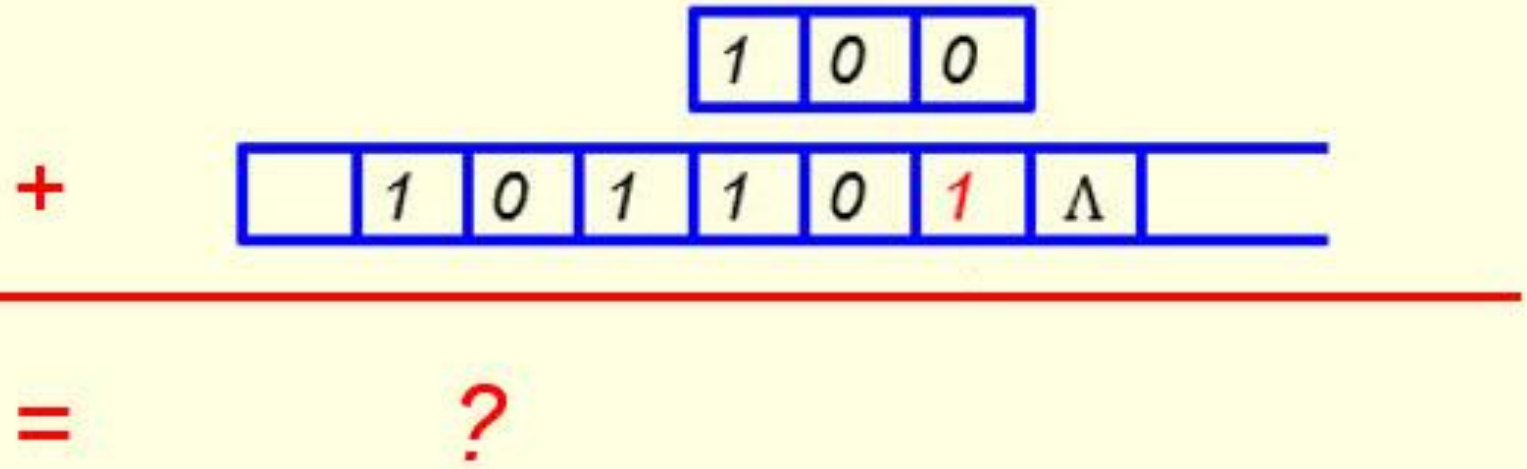
Find a TM to add 4 to a natural number represented in binary.

Start with the tape head at the right end of the input string and halt with the tape head at the left end of the output string.

8. Turing Machines

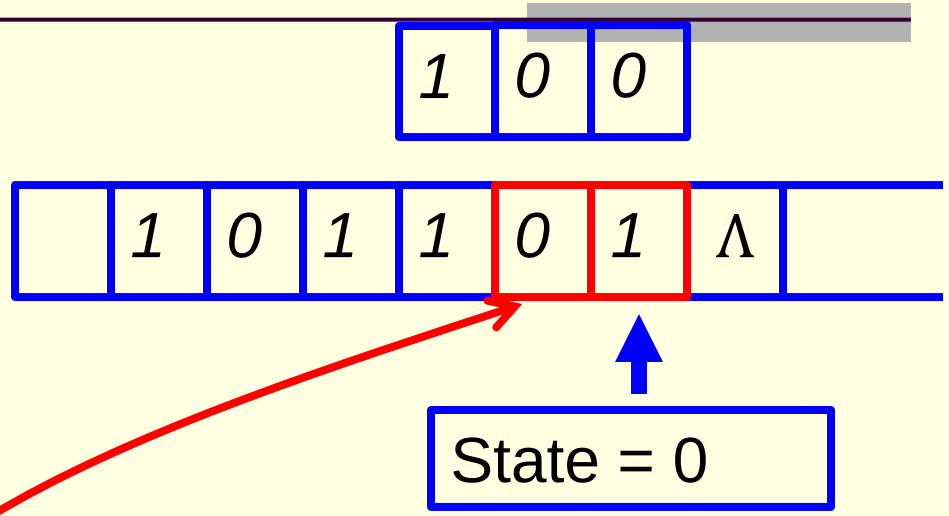
Example. $4 + 45 = ?$

$$4 = 100_2$$
$$45 = 101101_2$$

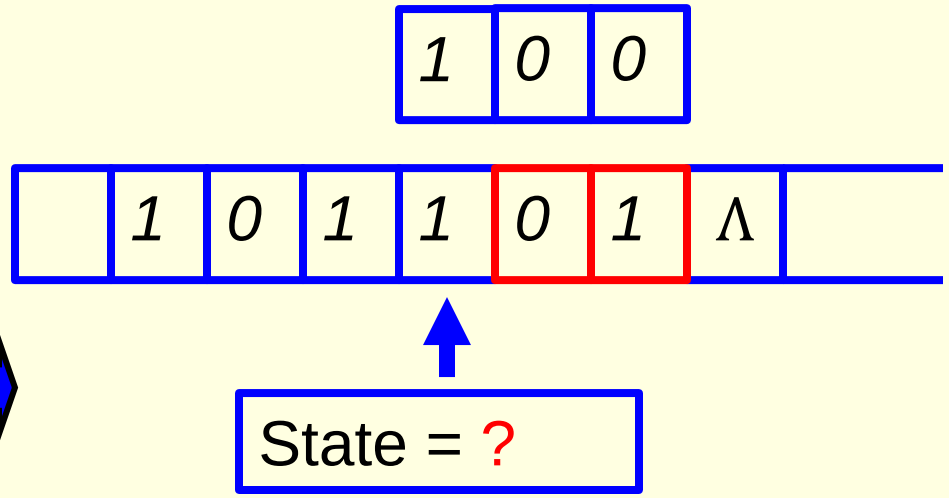


Solution.

Initially,
(start state = 0)

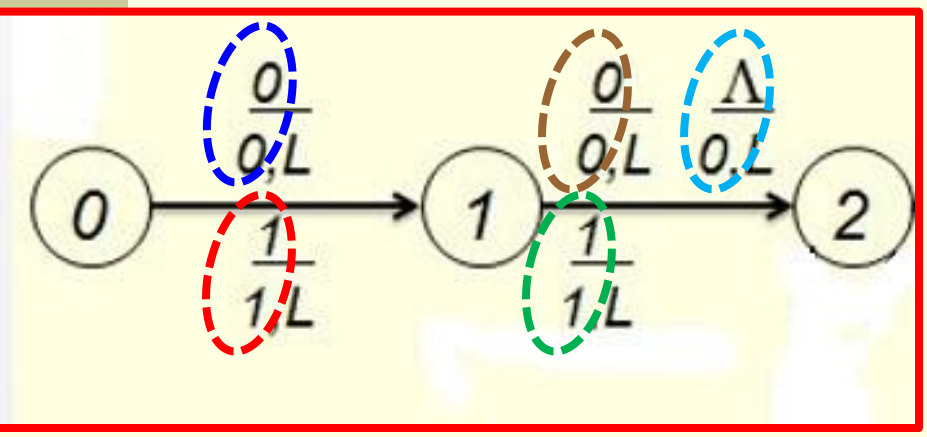


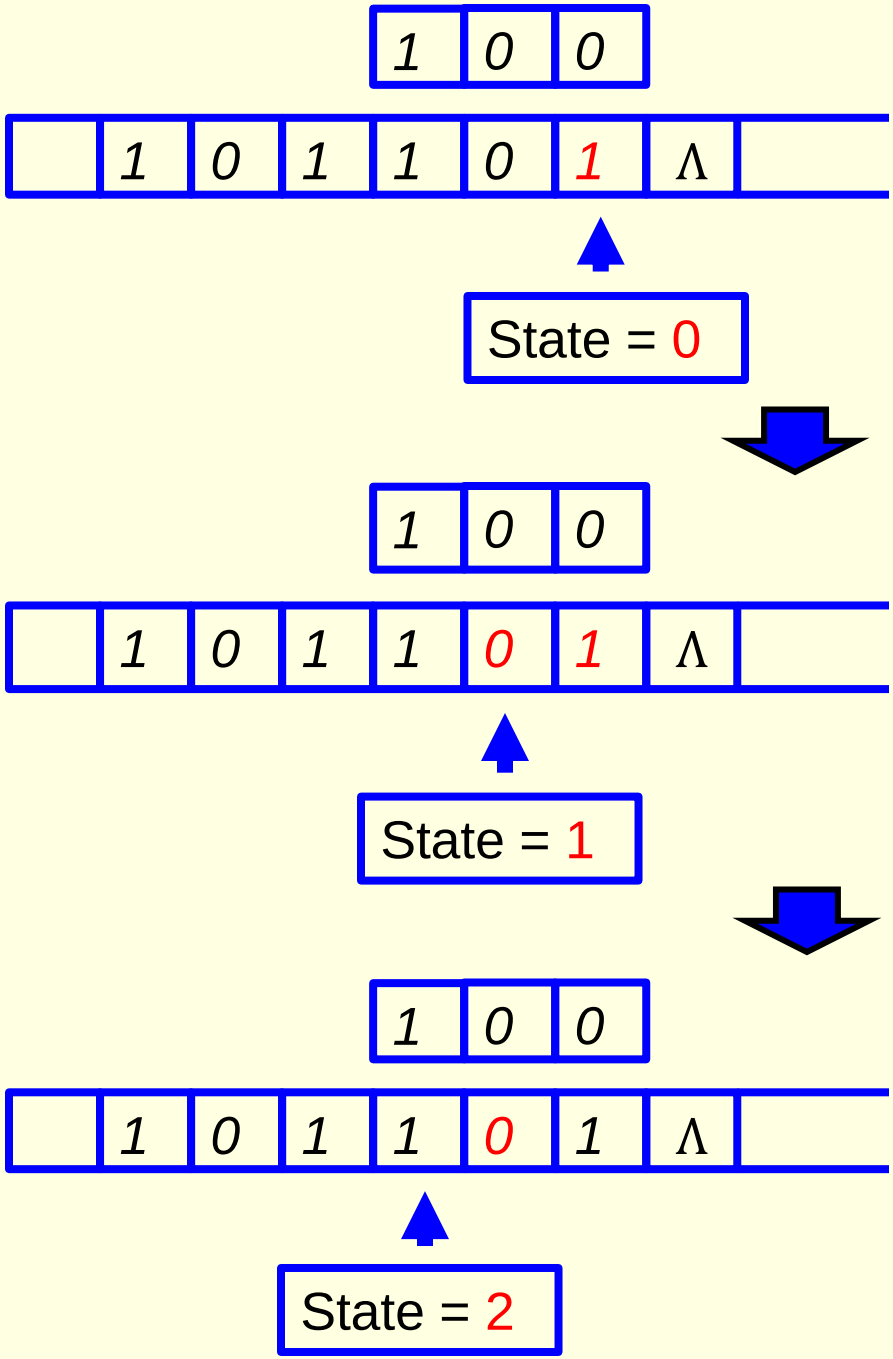
These two digits are of no concern to us, can move immediately to the third digit



Move two cells left:

$(0, 0, 0, L, 1)$
$(0, 1, 1, L, 1)$
$(1, 0, 0, L, 2)$
$(1, 1, 1, L, 2)$
$(1, \Lambda, 0, L, 2)$

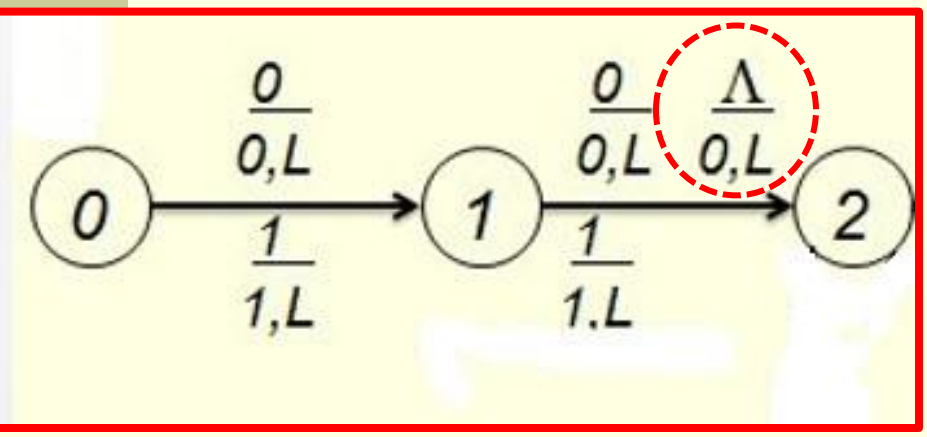


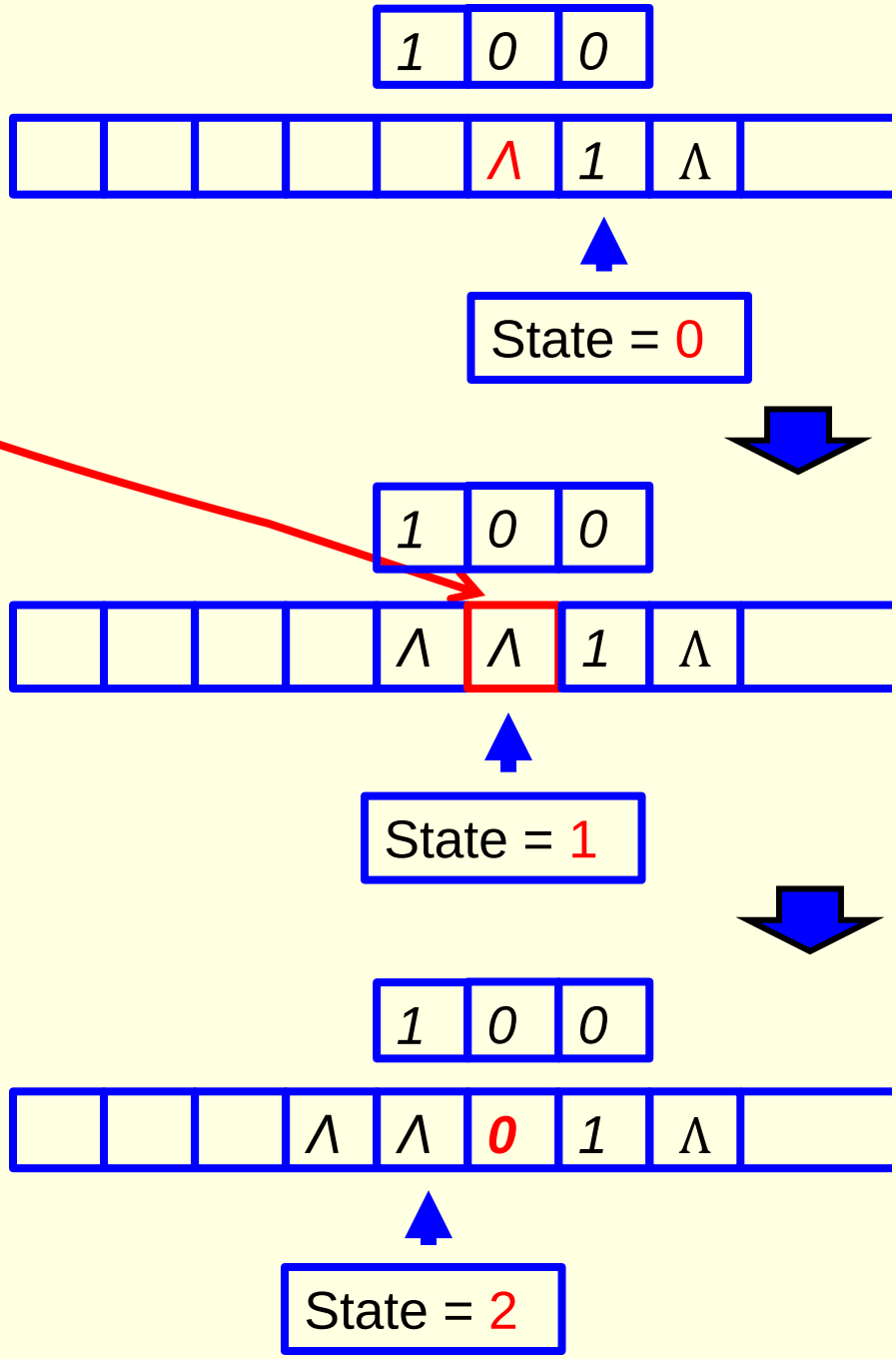


Move two cells left:

$(0, 0, 0, L, 1)$
$(0, 1, 1, L, 1)$
$(1, 0, 0, L, 2)$
$(1, 1, 1, L, 2)$
$(1, \Lambda, 0, L, 2)$

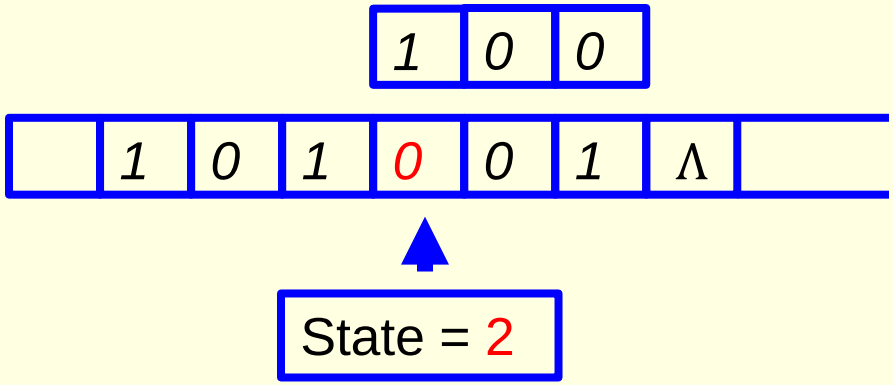
Note that

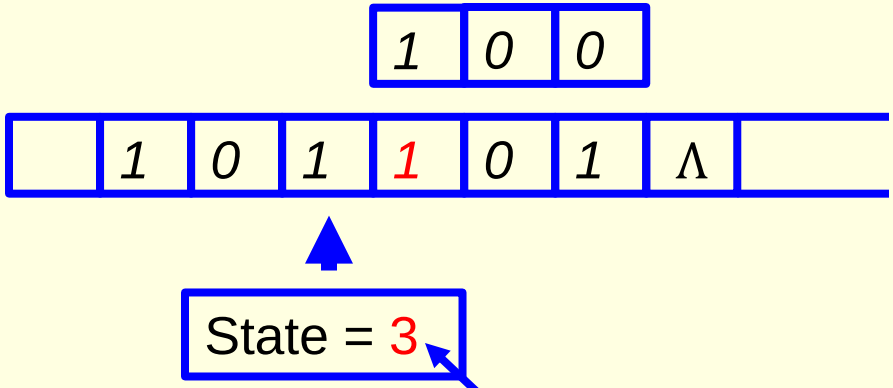


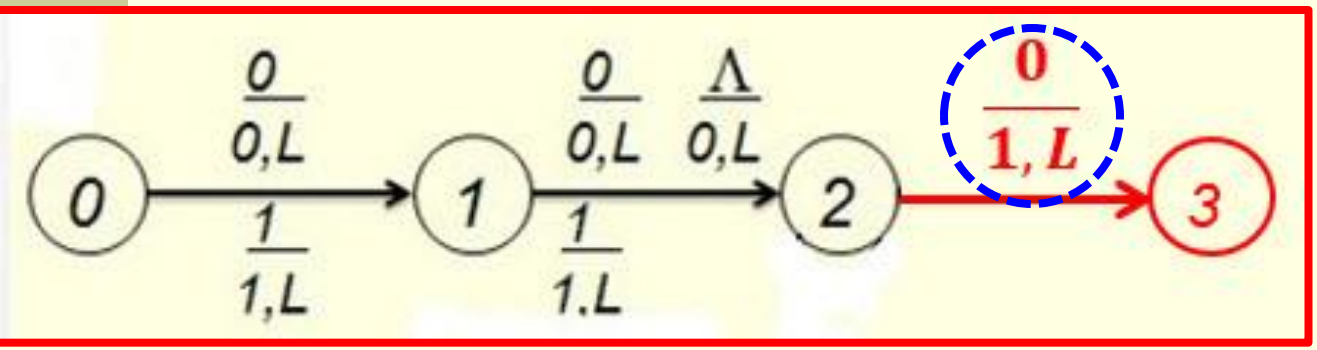


Add 1 Case 2-1:

Add 1:	
(2, 0, 1, *L*, 3)	Move left
(2, 1, 0, *L*, 2)	Carry
(2, Λ, 1, *S*, halt)	Done



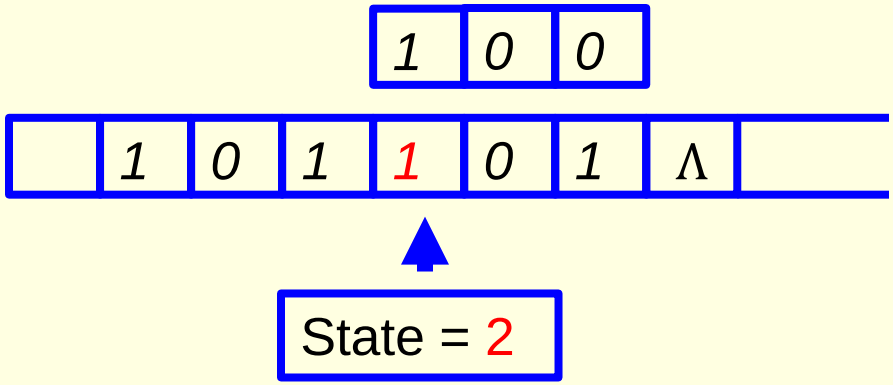


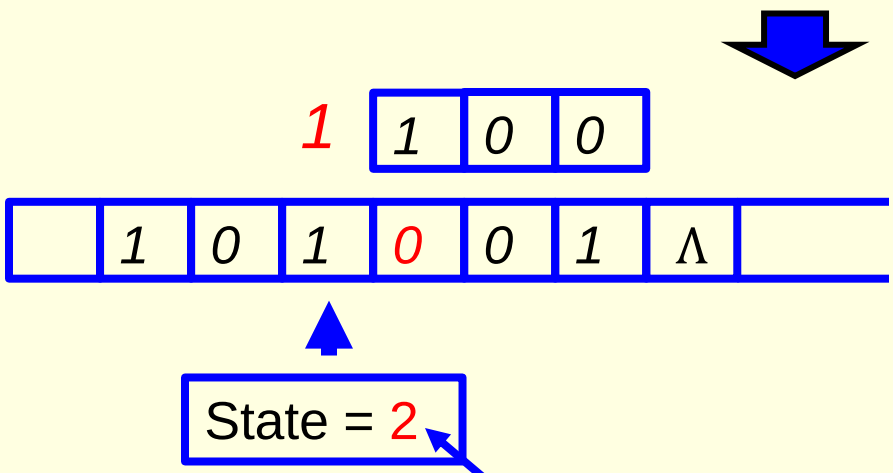


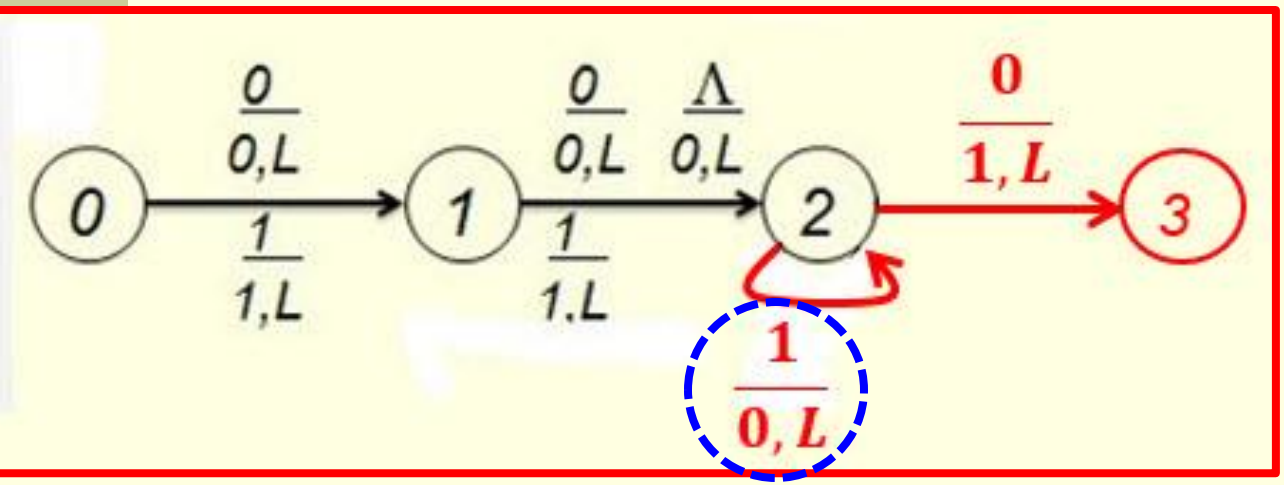
No carry from this point on

Add 1 Case 2-2:

Add 1:	
(2, 0, 1, *L*, 3)	Move left
(2, 1, 0, *L*, 2)	Carry
(2, Λ, 1, *S*, halt)	Done





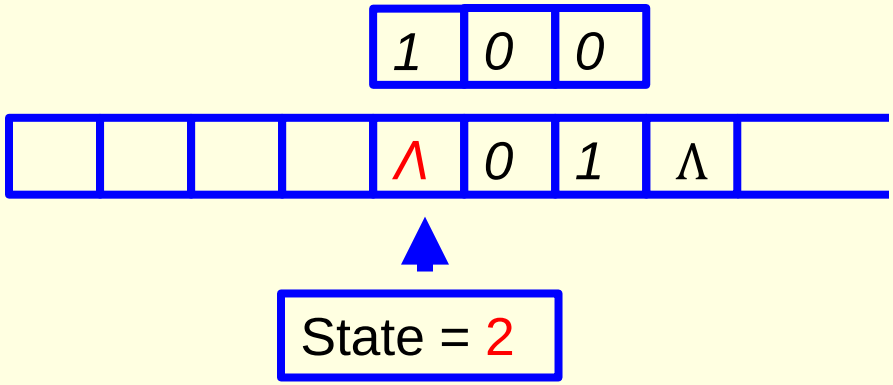


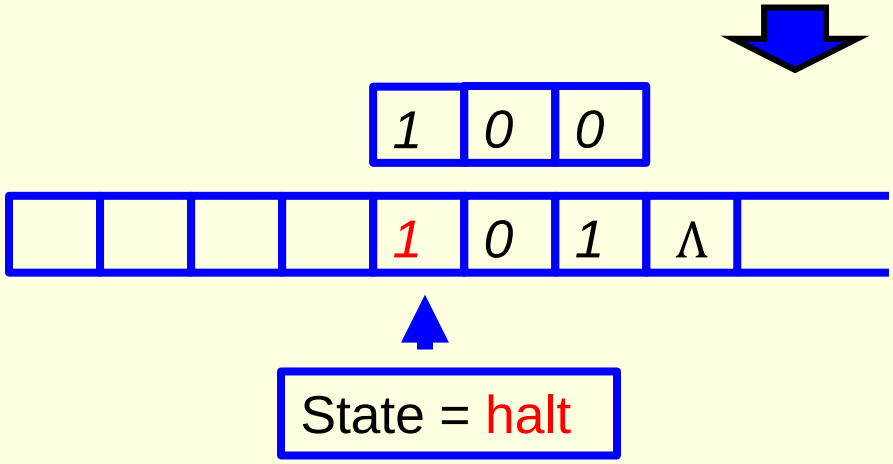
There is a carry for the next point

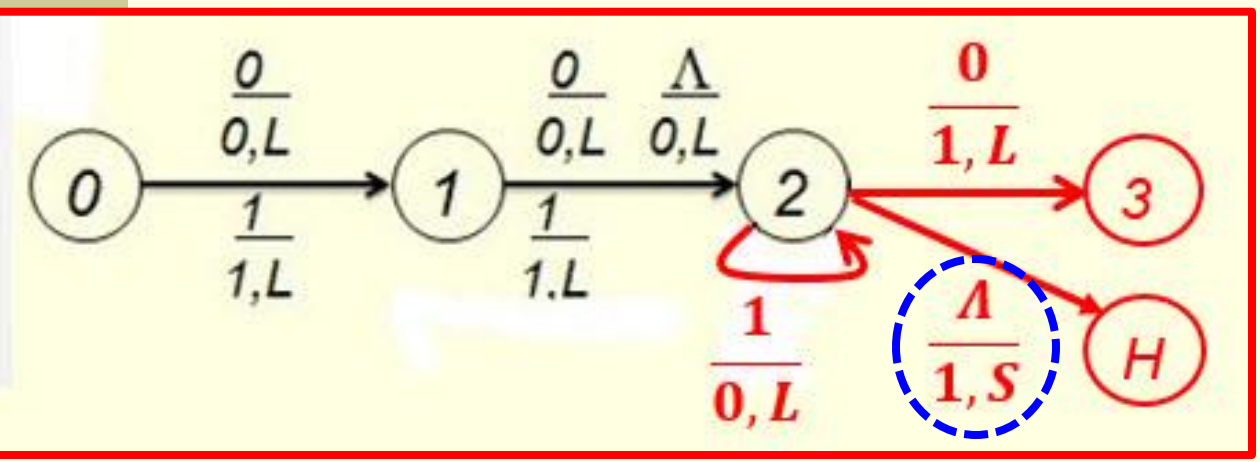
Add 1 *Case 2-3:*

Add 1:

(2, 0, 1, *L*, 3)	Move left
(2, 1, 0, *L*, 2)	Carry
(2, Λ, 1, *S*, halt)	Done







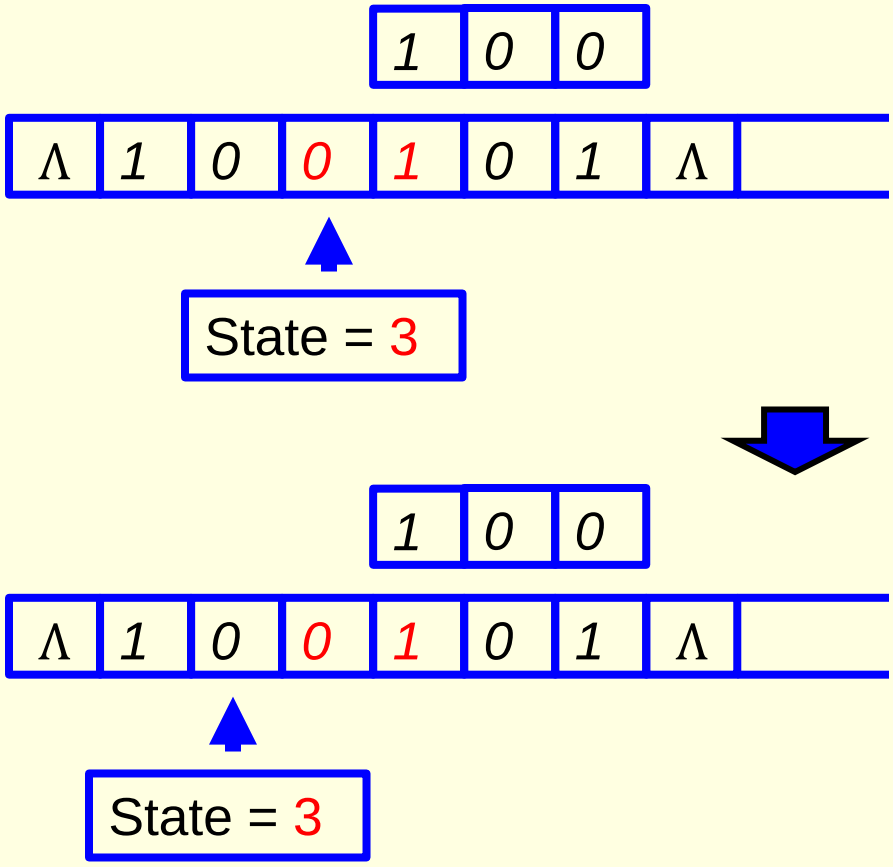
Find left end of the string
Case 3-1:

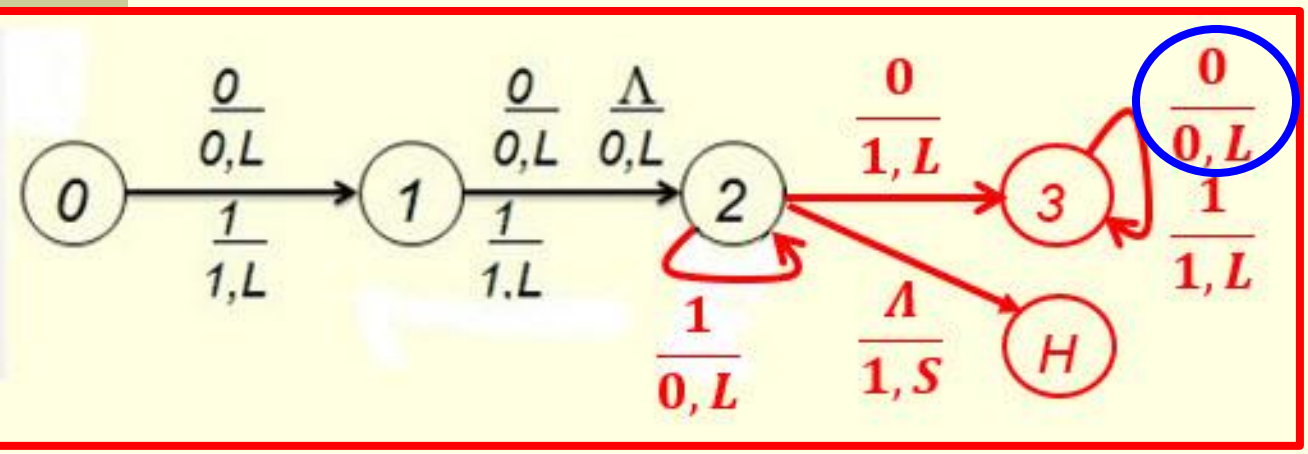
Find left end of the string:

(3, 0, 0, *L*, 3)

(3, 1, 1, *L*, 3)

(3, Λ, Λ, *R*, halt) Done





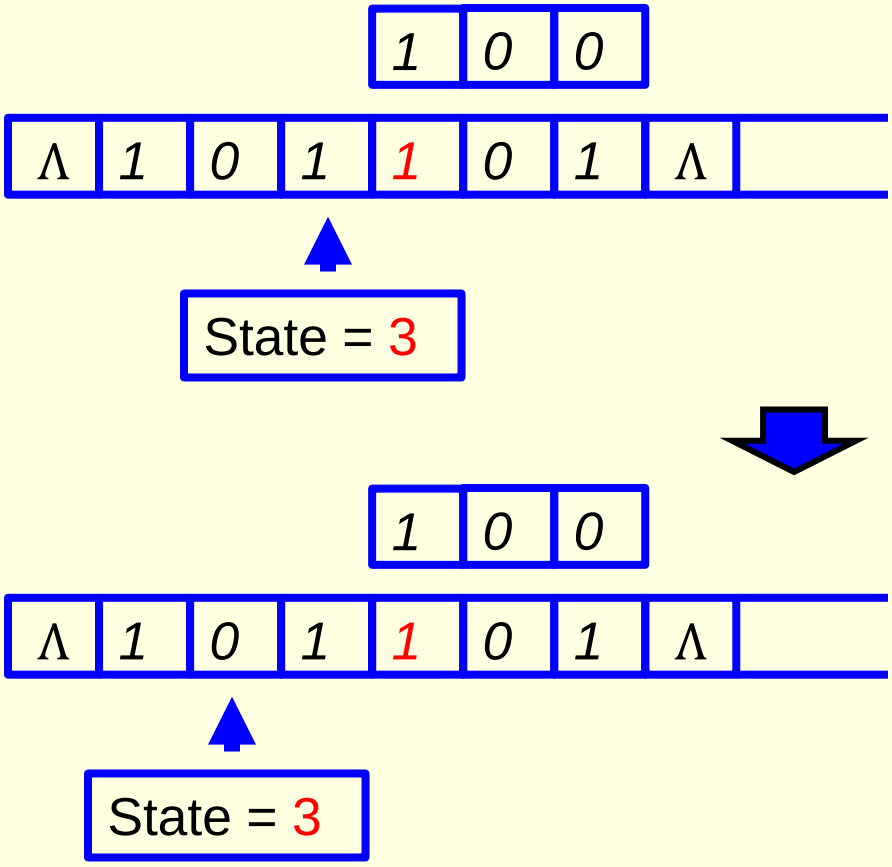
Find left end of the string
Case 3-2:

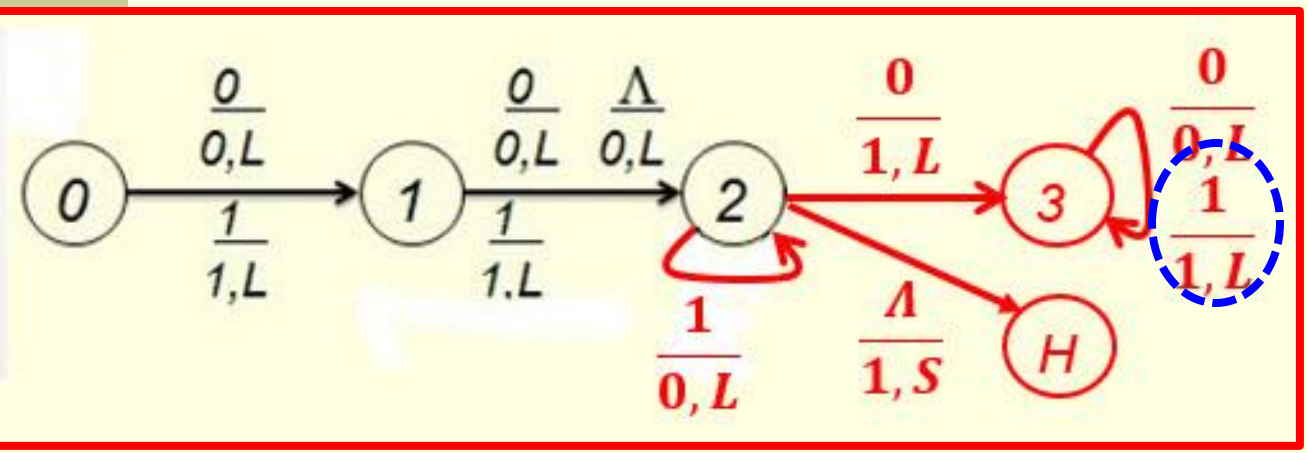
Find left end of the string:

(3, 0, 0, *L*, 3)

(3, 1, 1, *L*, 3)

(3, Λ, Λ, *R*, halt) Done





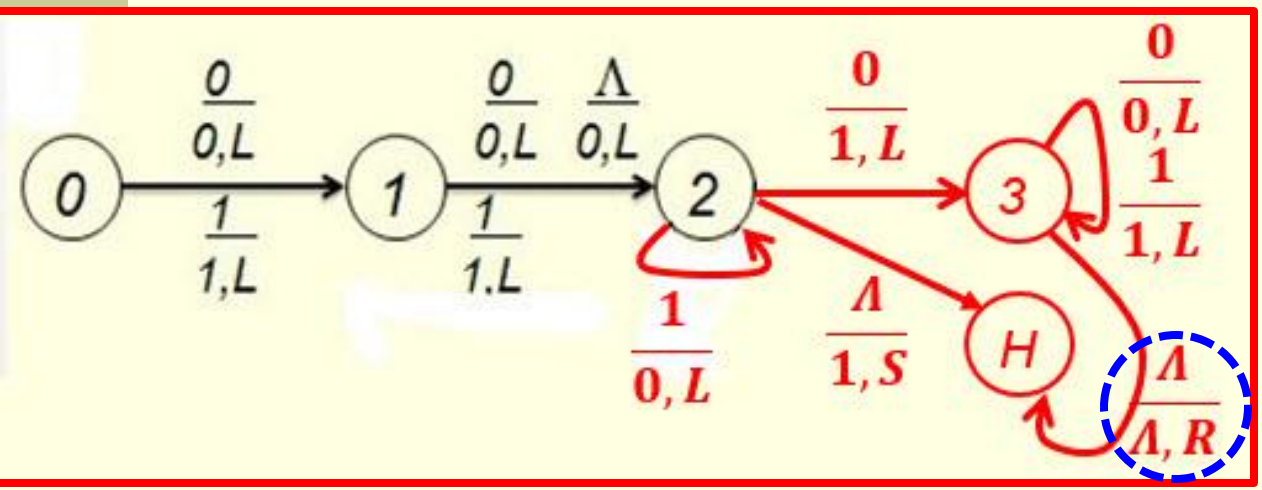
Find left end of the string
Case 3-3:

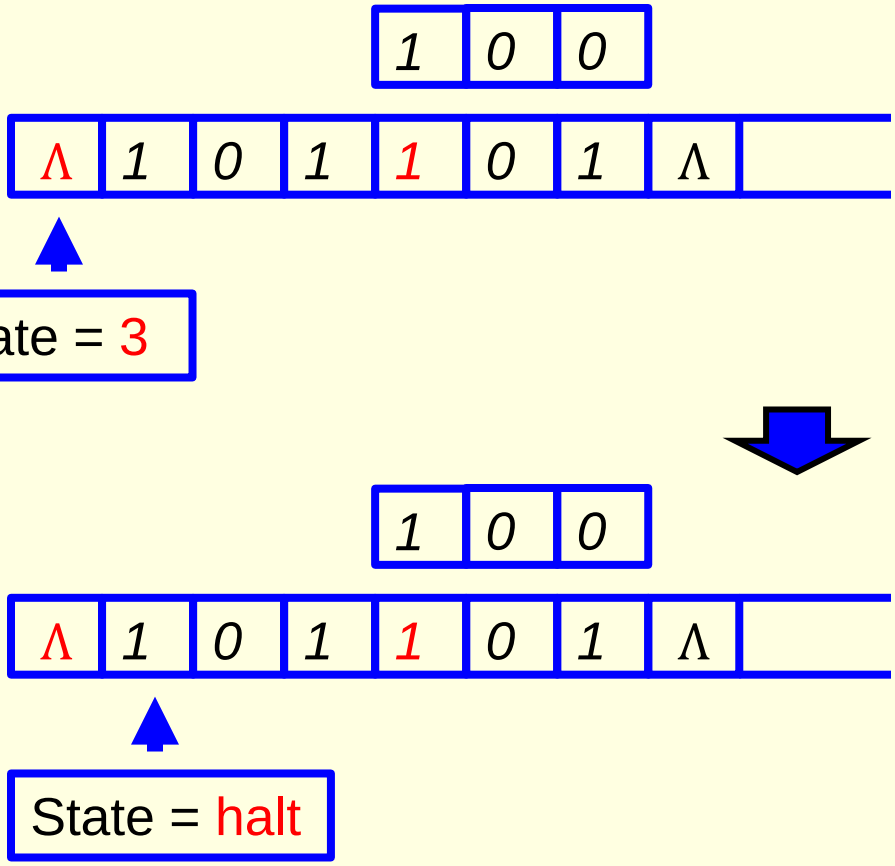
Find left end of the string:

(3, 0, 0, *L*, 3)

(3, 1, 1, *L*, 3)

(3, Λ, Λ, *R*, halt) Done





Hence, we have

11 instructions (5 states)

Move two cells left:

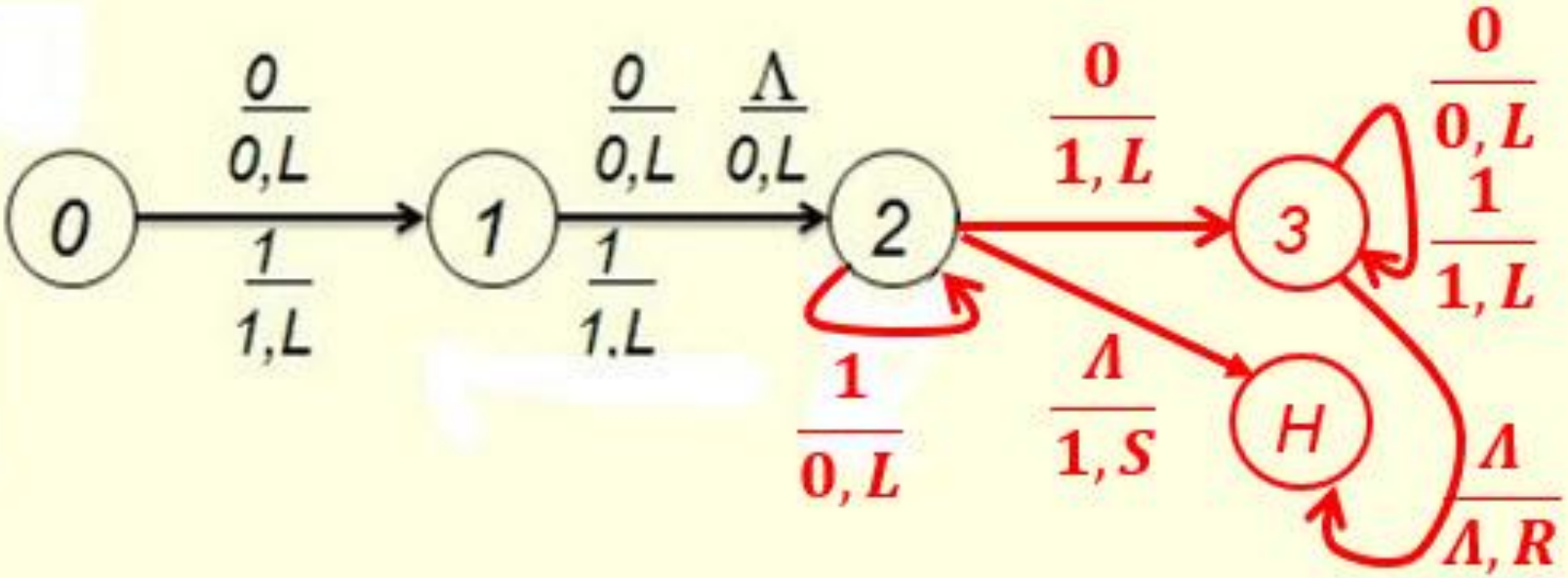
$(0, 0, 0, L, 1)$
$(0, 1, 1, L, 1)$
$(1, 0, 0, L, 2)$
$(1, 1, 1, L, 2)$
$(1, \Lambda, 0, L, 2)$

Add 1:

$(2, 0, 1, L, 3)$	Move left
$(2, 1, 0, L, 2)$	Carry
$(2, \Lambda, 1, S, \text{halt})$	Done

Find left end of the string:

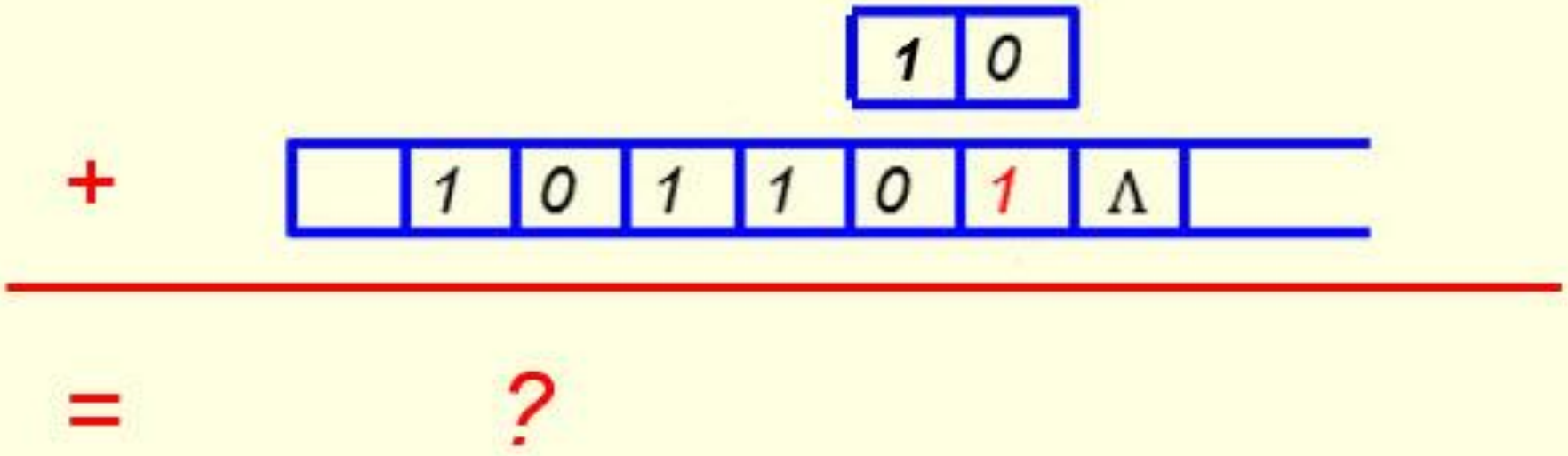
$(3, 0, 0, L, 3)$	
$(3, 1, 1, L, 3)$	
$(3, \Lambda, \Lambda, R, \text{halt})$	Done



8. Turing Machines

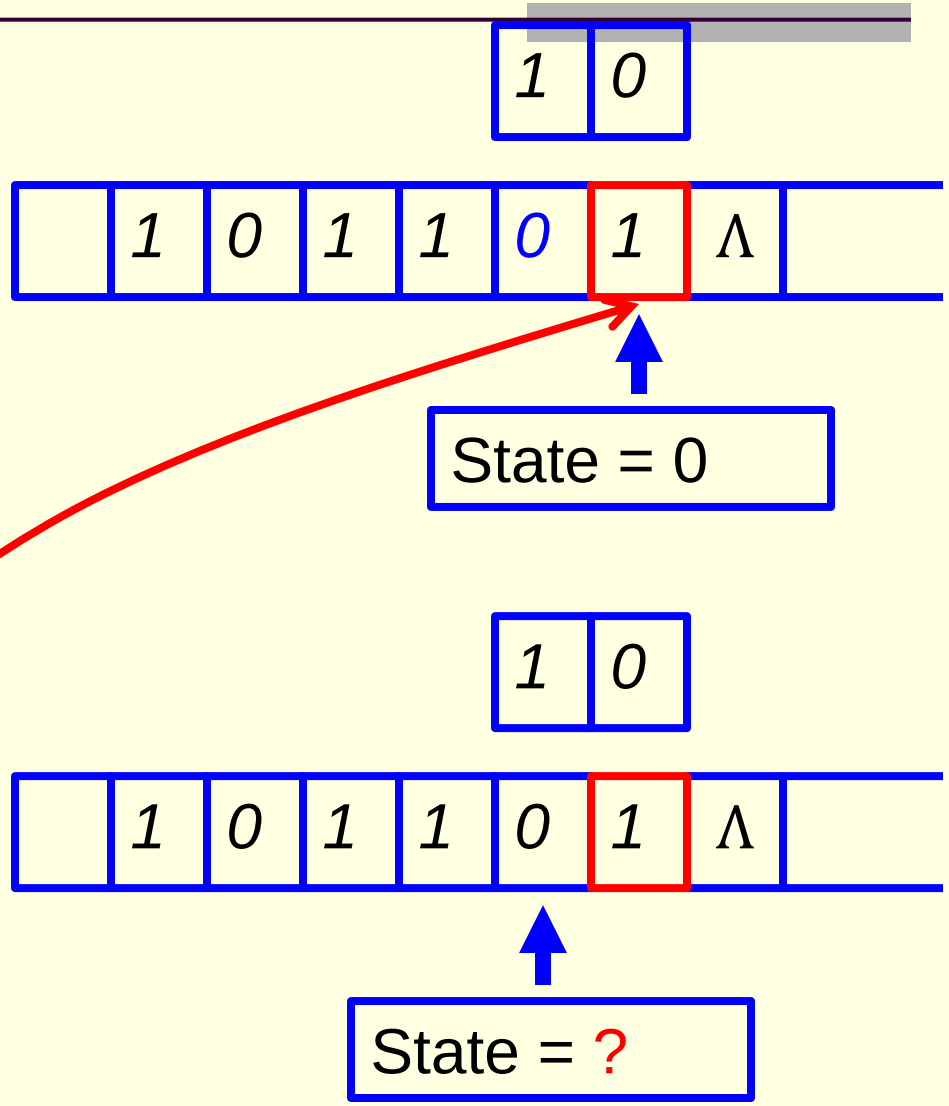
Question. 2 + 45 = ?

$$2 = 10_2$$
$$45 = 101101_2$$



Solution.

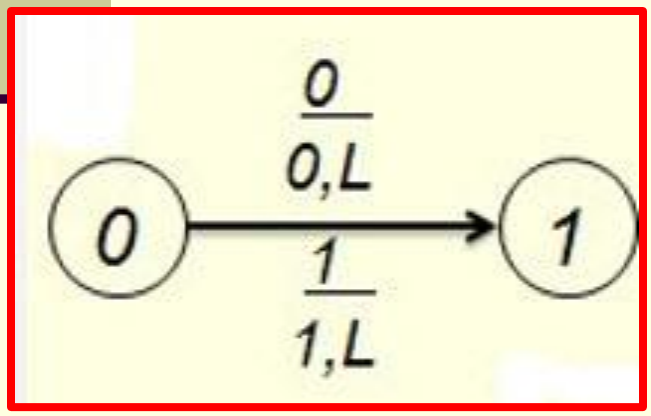
Initially,
(start state = 0)

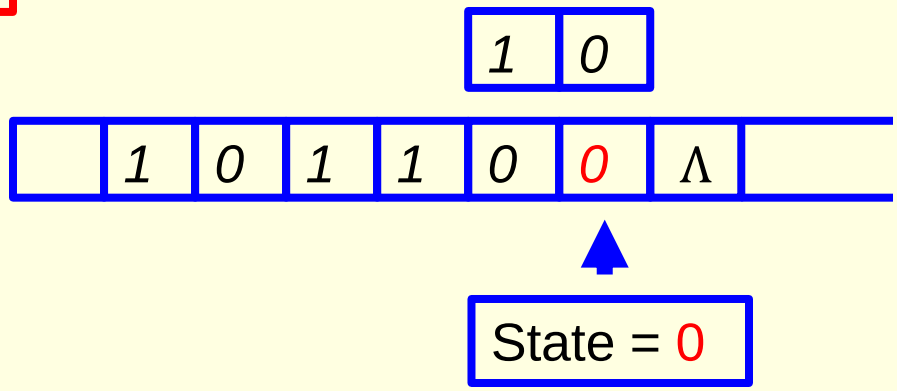


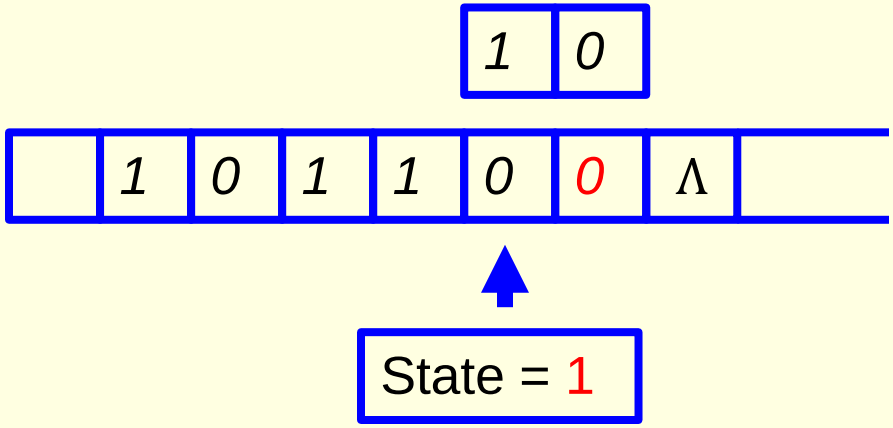
Move one cell left: case 0-1

$(0, 0, 0, L, 1)$

$(0, 1, 1, L, 1)$



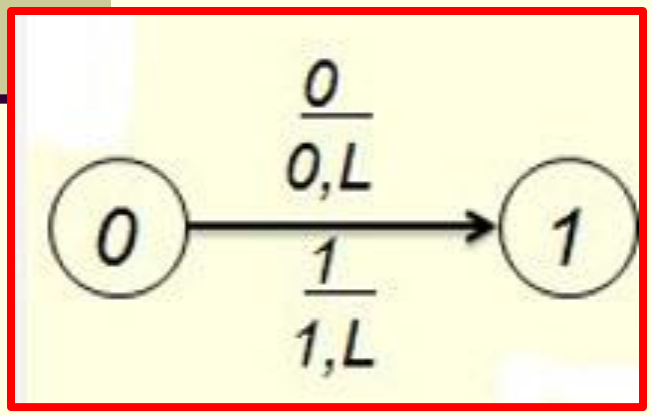


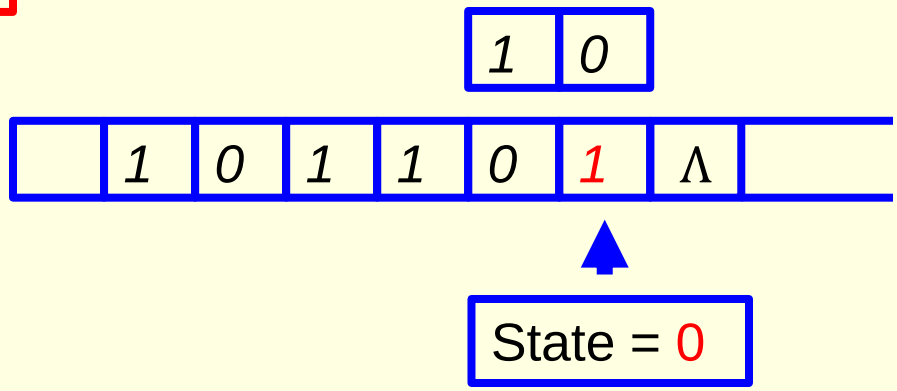


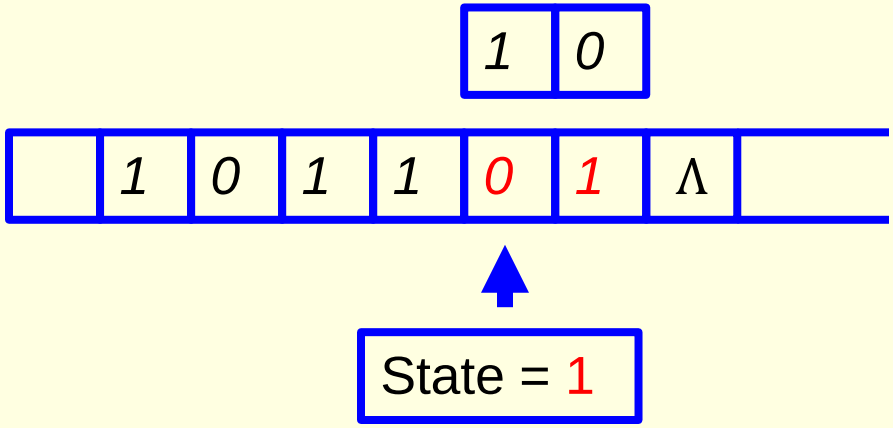
Move one cell left: case 0-2

$(0, 0, 0, L, 1)$

$(0, 1, 1, L, 1)$

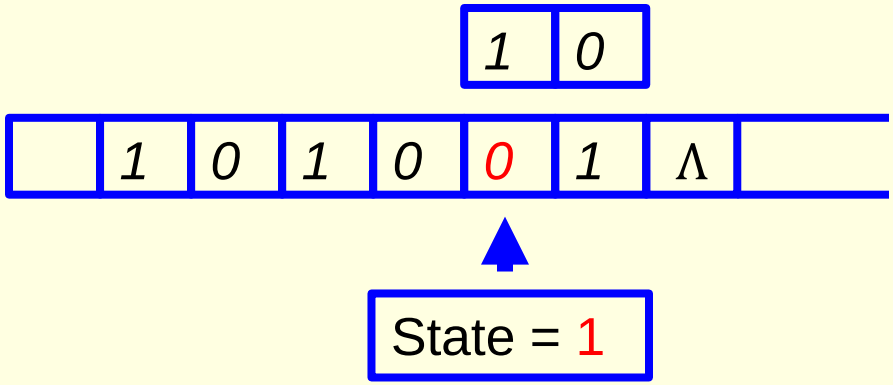


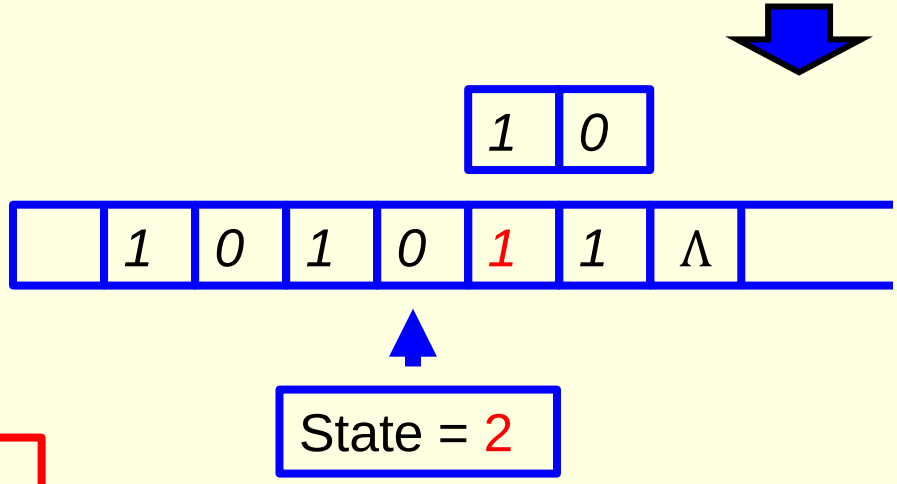


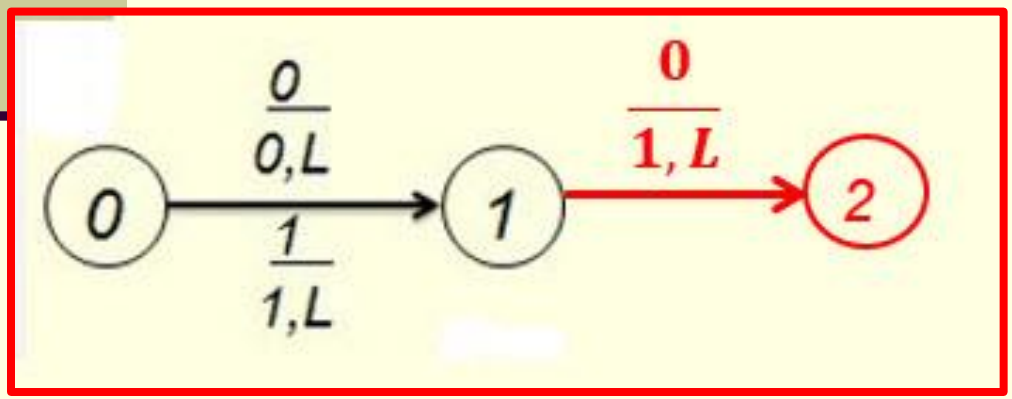


Add 1 *Case 1-1:*

(1, 0, 1, L, 2)	Move left
(1, 1, 0, L, 1)	Carry
(1, Λ, 1, S, halt)	Done





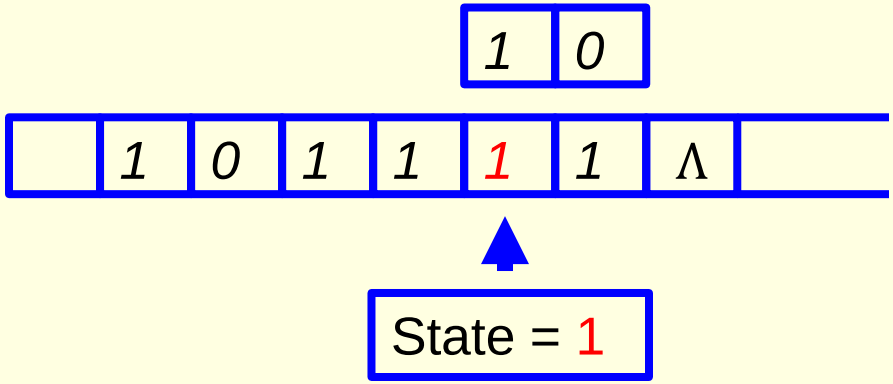


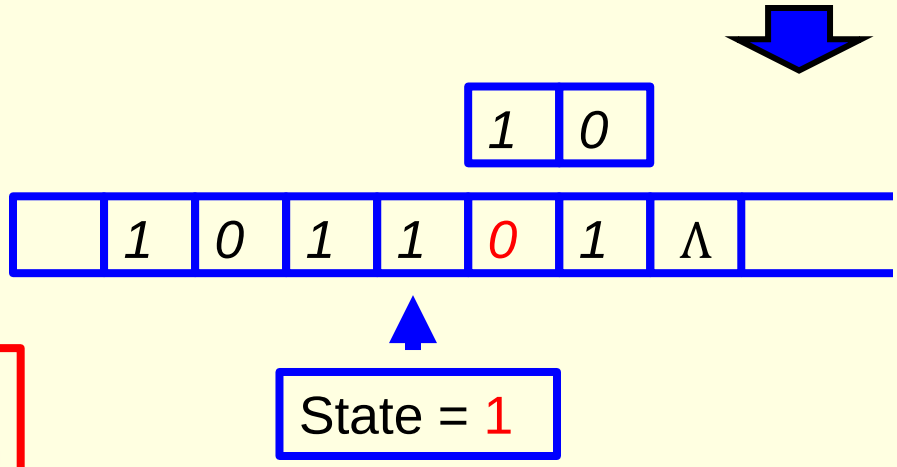
Add 1 *Case 1-2:*

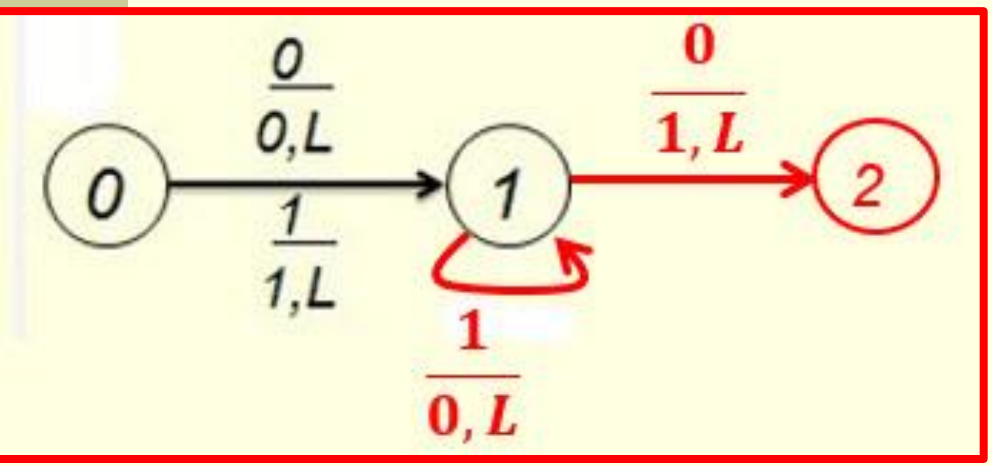
(1, 0, 1, L, 2) Move left

(1, 1, 0, L, 1) Carry

(1, Λ, 1, S, halt) Done

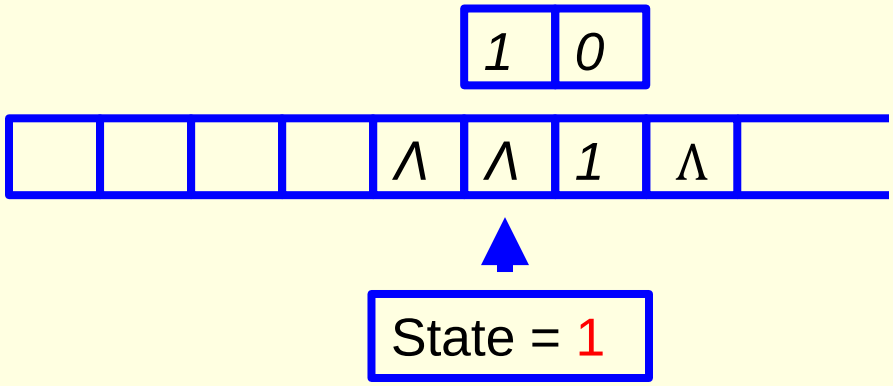


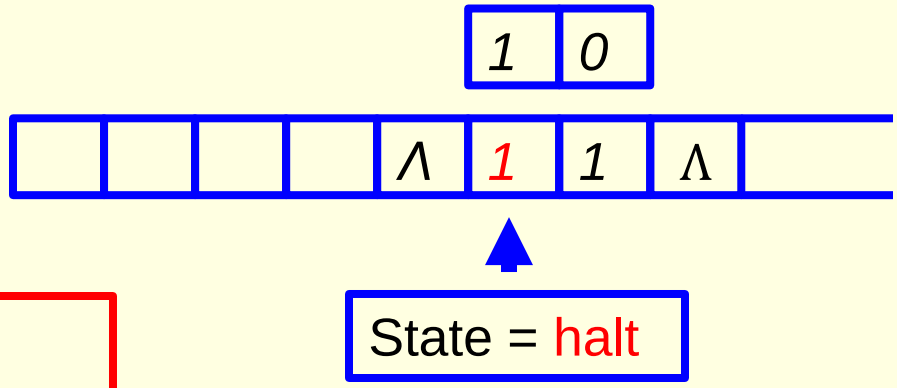


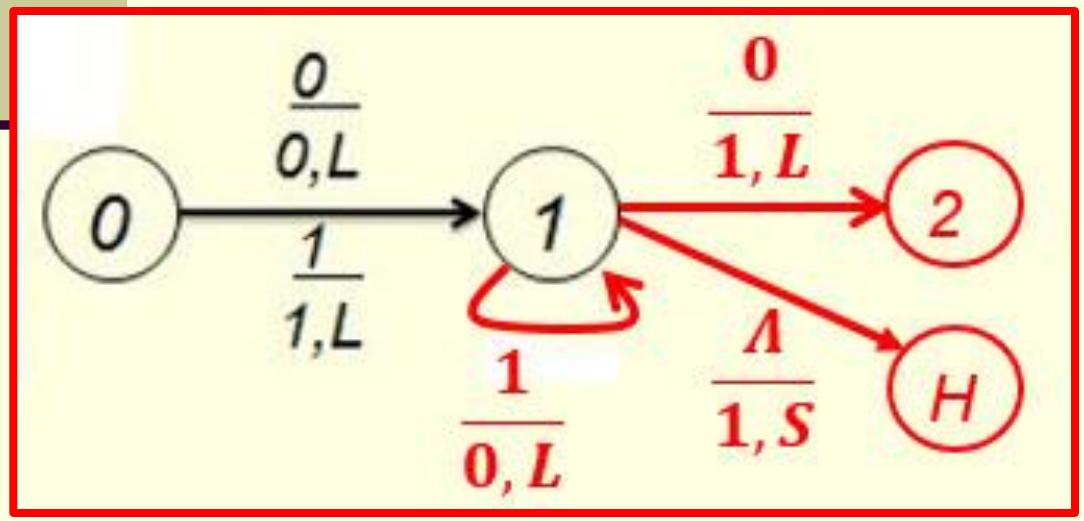


Add 1 *Case 1-3:*

(1, 0, 1, L, 2)	Move left
(1, 1, 0, L, 1)	Carry
(1, Λ, 1, S, halt)	Done





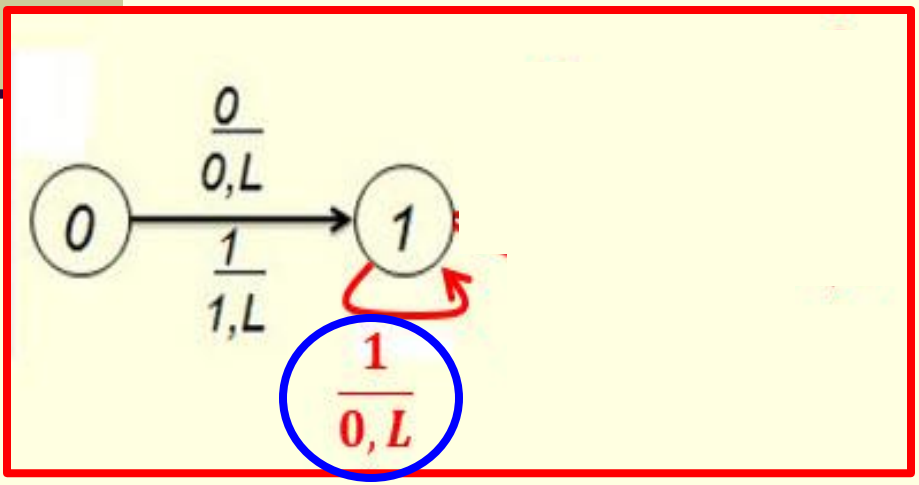


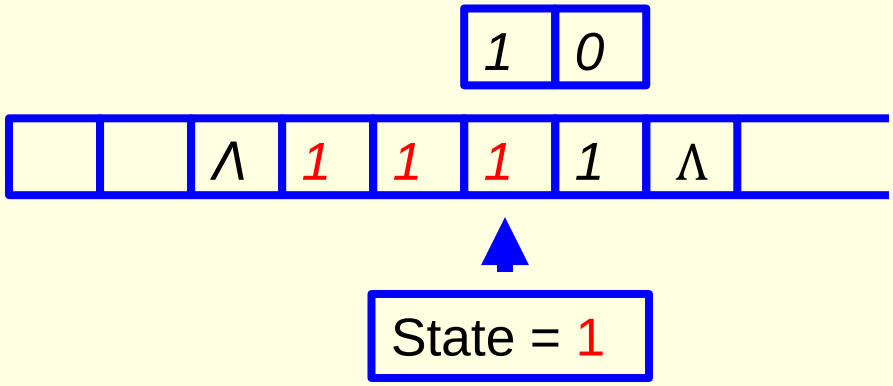
Add 1 *Case 1-4:*

(1, 0, 1, L, 2) Move left

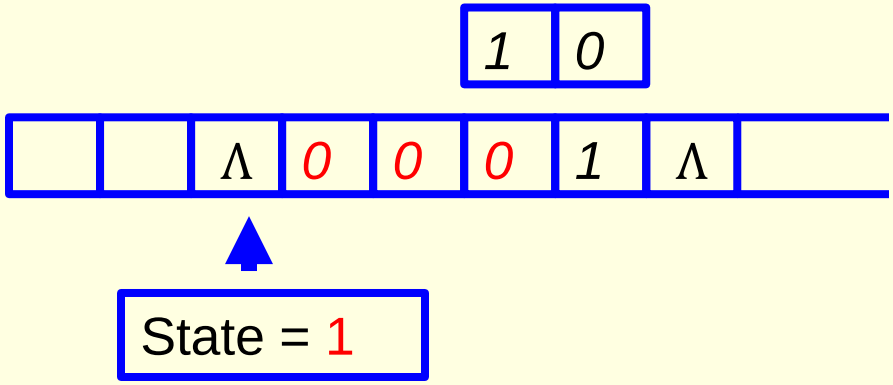
(1, 1, 0, L, 1) Carry

(1, Λ, 1, S, halt) Done



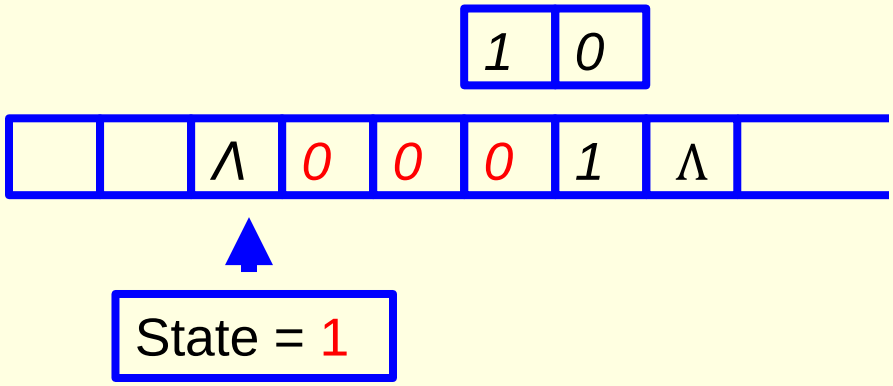


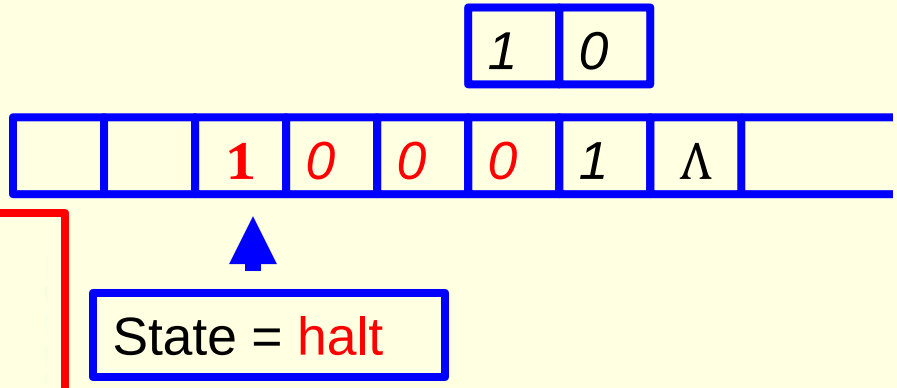
(*After 3 steps*)

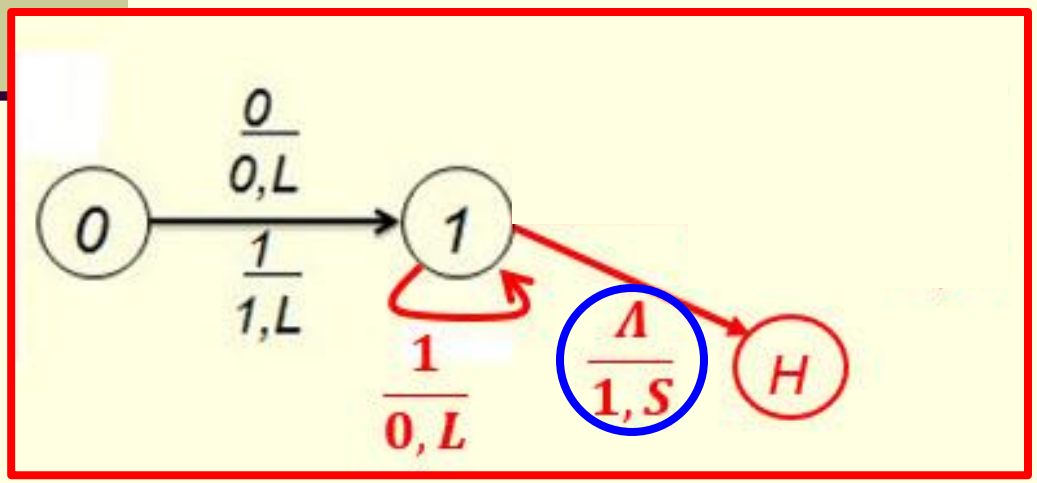


Add 1 *Case 1-4:*

(1, 0, 1, L, 2) Move left
(1, 1, 0, L, 1) Carry
(1, Λ, 1, S, halt) Done





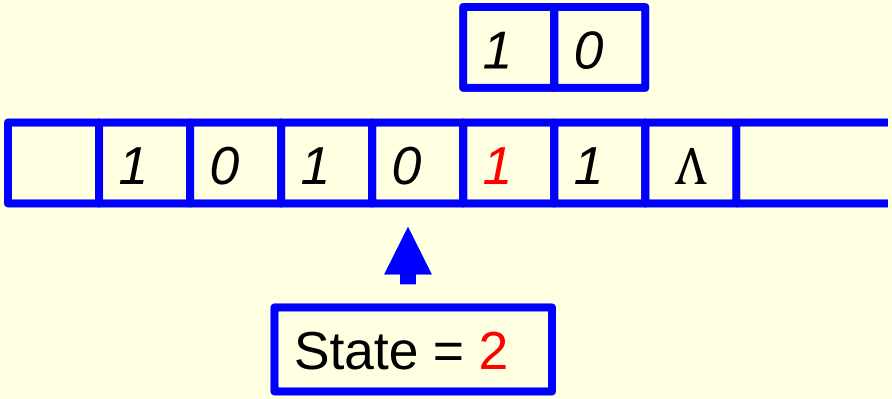


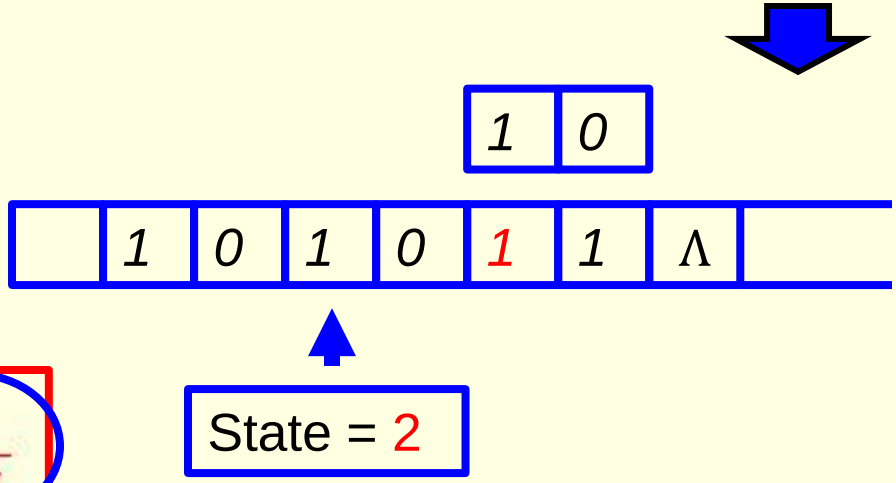
Find left end of the string
Case 2-1:

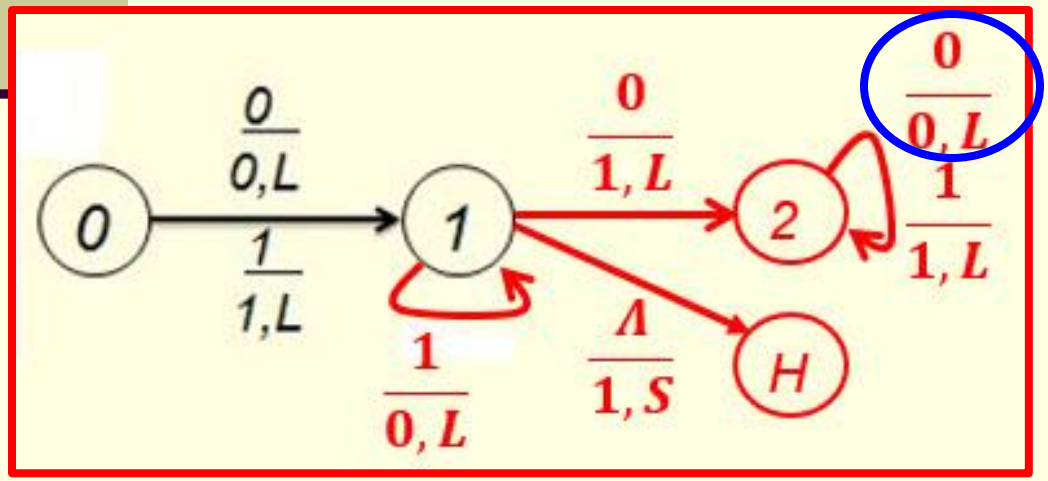
(2, 0, 0, L, 2)

(2, 1, 1, L, 2)

(2, Λ, Λ, R, halt) Done





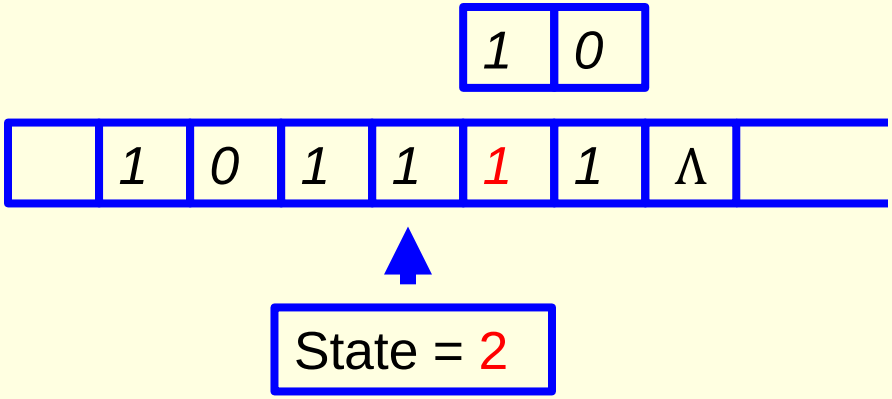


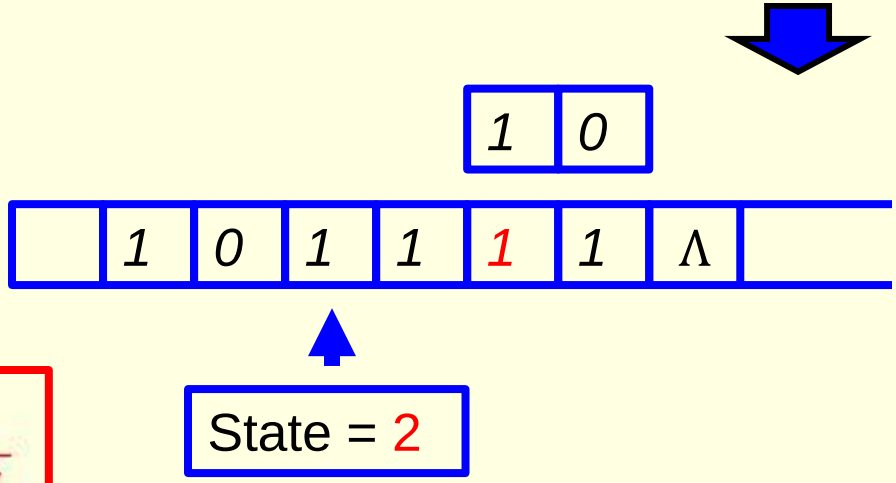
Find left end of the string
Case 2-2:

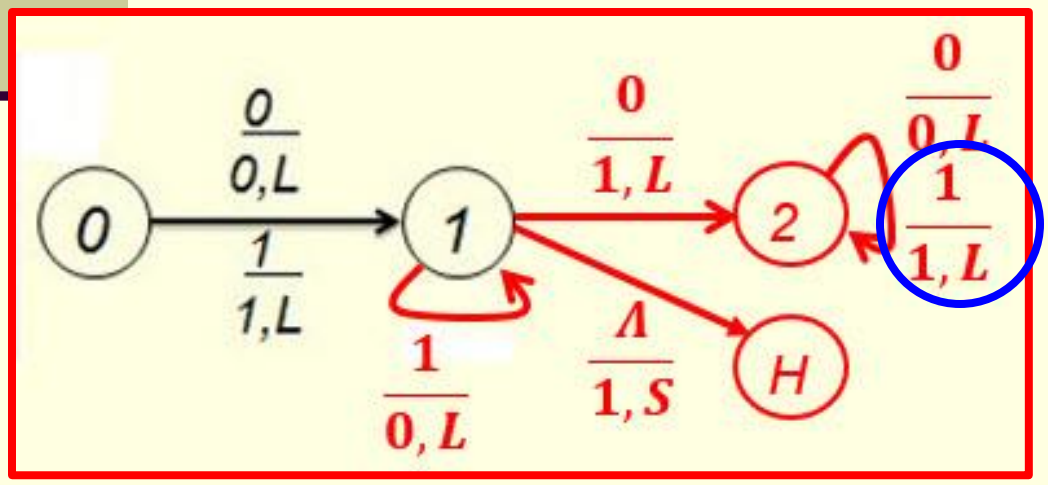
(2, 0, 0, L, 2)

(2, 1, 1, L, 2)

(2, Λ, Λ, R, halt) Done





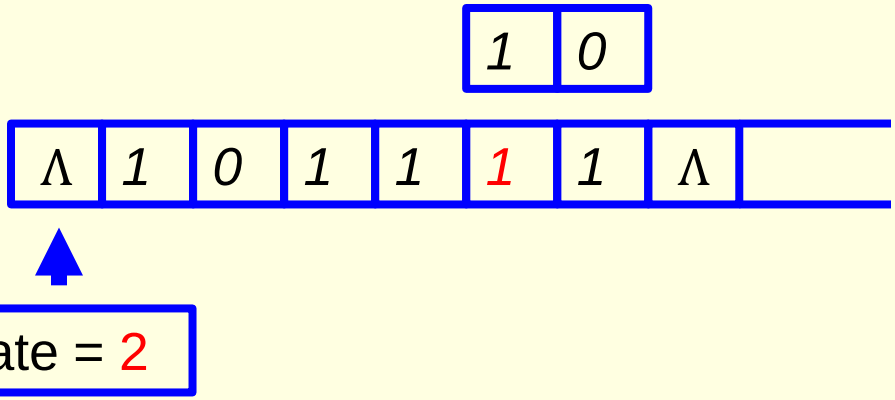


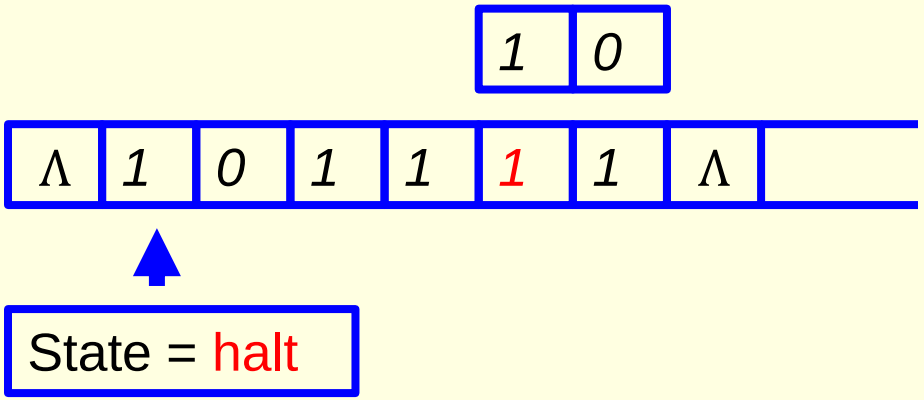
Find left end of the string
Case 2-3:

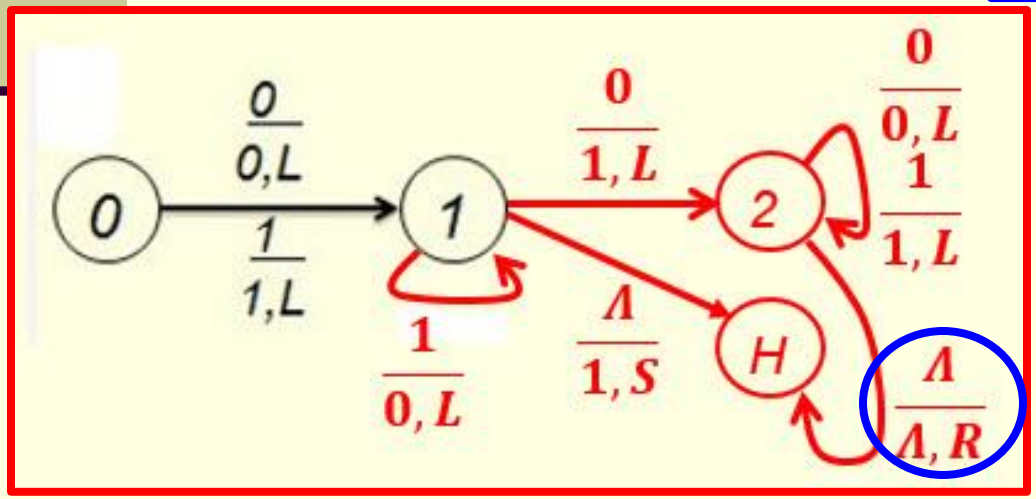
(2, 0, 0, L, 2)

(2, 1, 1, L, 2)

(2, Λ, Λ, R, halt) Done







Hence we have: *8 instructions (4 states)*

Move one cell left:

(0, 0, 0, L, 1)

(0, 1, 1, L, 1)

Add 1

(1, 0, 1, L, 2) **Move left**

(1, 1, 0, L, 1) **Carry**

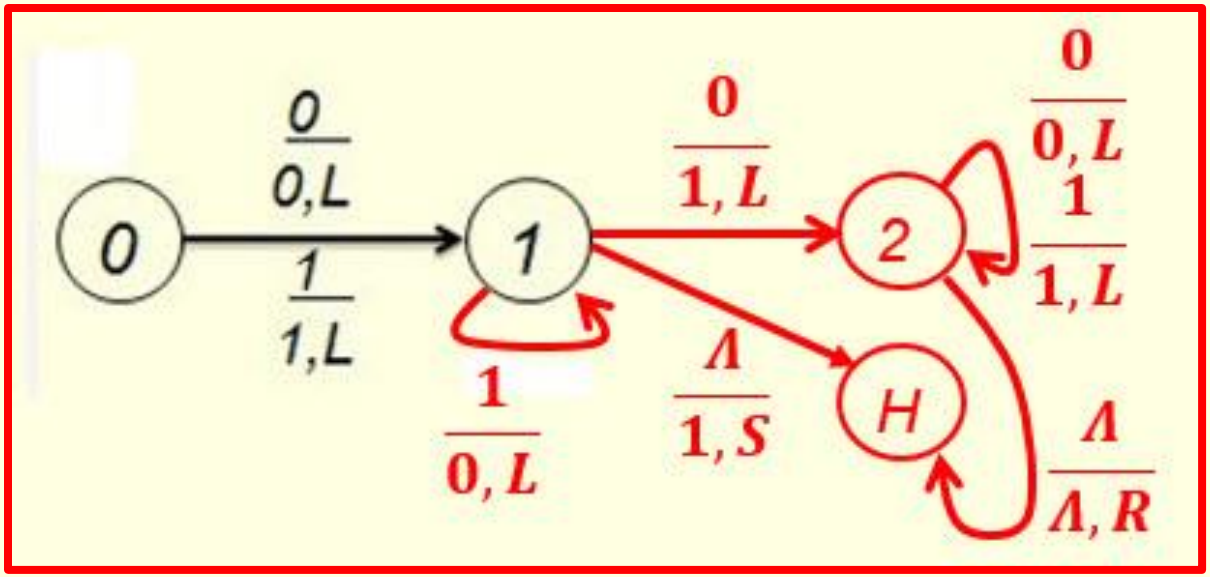
(1, Λ, 1, S, halt) **Done**

Find left end of the string

(2, 0, 0, L, 2)

(2, 1, 1, L, 2)

(2, Λ, Λ, R, halt) **Done**



Quiz. *construct a TM to add 5 to a binary number*

One Solution: Add 1, move back to right end, and then use the preceding solution.

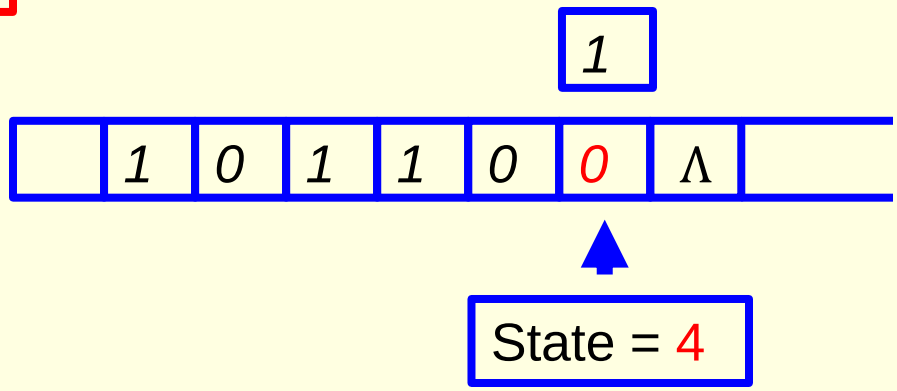
For this purpose, let the start state be 4.

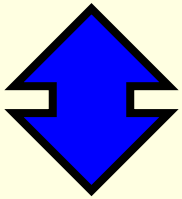
Add 1:	
(4, 0, 1, R, 5)	Move right
(4, 1, 0, L, 4)	Carry
(4, Λ, 1, R, 5)	Move right

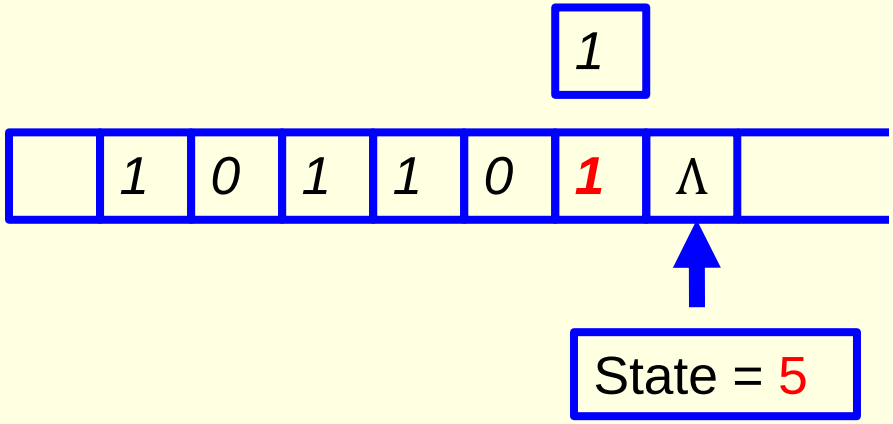
Find right end of the string:	
(5, 0, 0, R, 5)	
(5, 1, 1, R, 5)	
(5, Λ, Λ, L, 0)	Go add 4

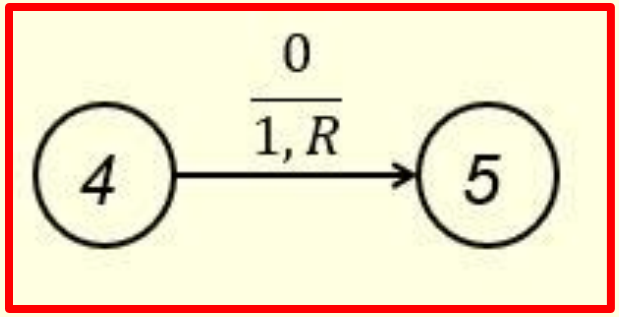
Add 1: case 0-1

(4, 0, 1, R, 5)	Move right
(4, 1, 0, L, 4)	Carry
(4, Λ, 1, R. 5)	Move right



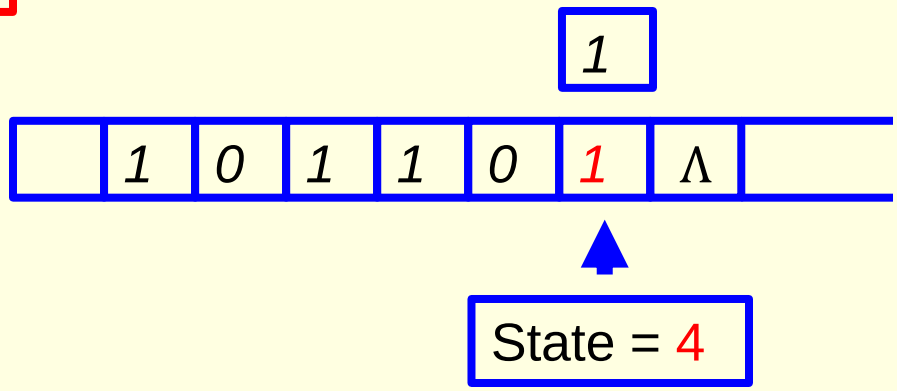


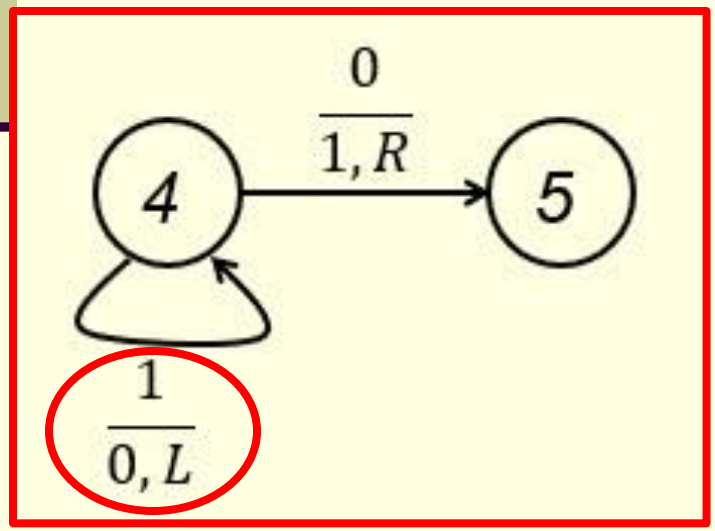


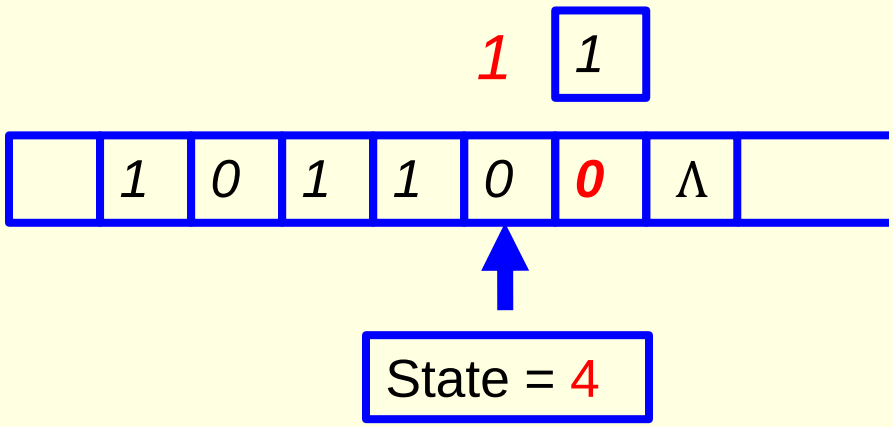


Add 1: case 0-2

(4, 0, 1, R, 5)	Move right
(4, 1, 0, L, 4)	Carry
(4, Λ, 1, R. 5)	Move right

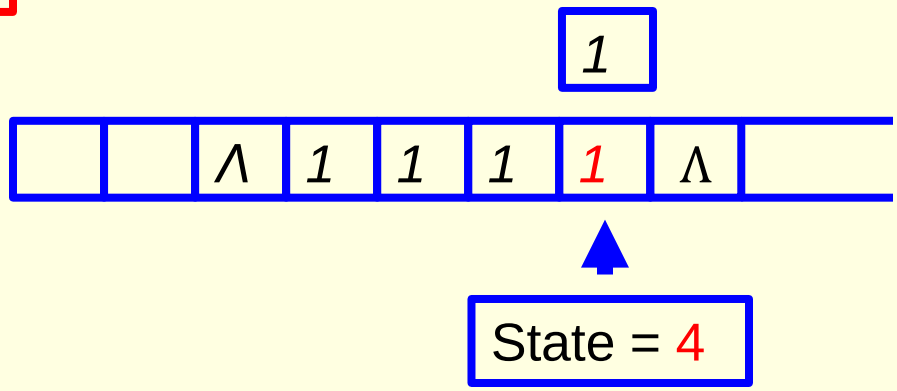


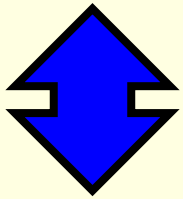




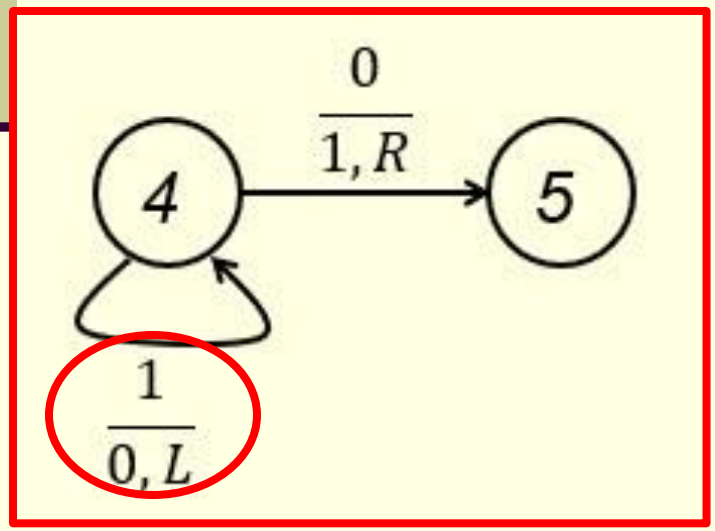
Add 1: case 0-3

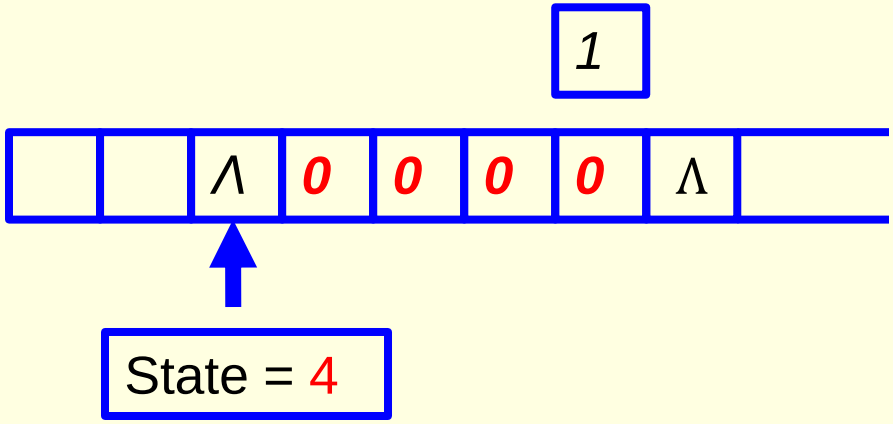
(4, 0, 1, R, 5)	Move right
(4, 1, 0, L, 4)	Carry
(4, Λ, 1, R. 5)	Move right





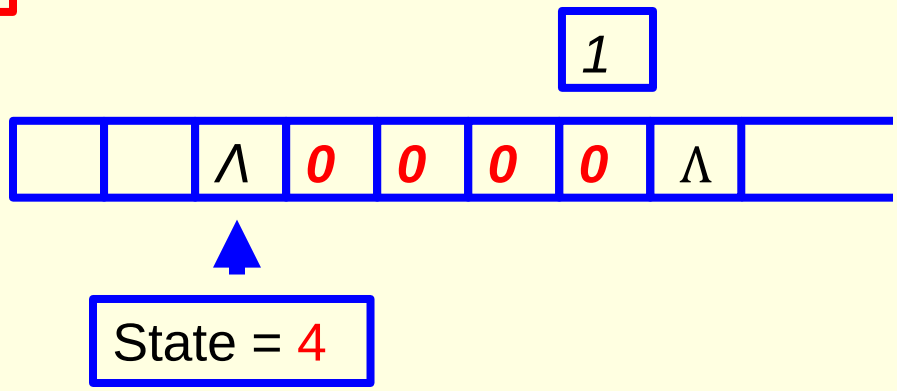
(after 4 steps)

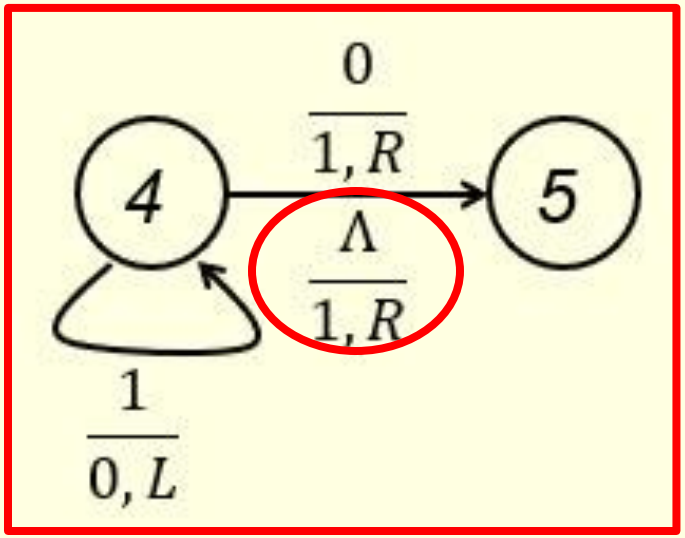


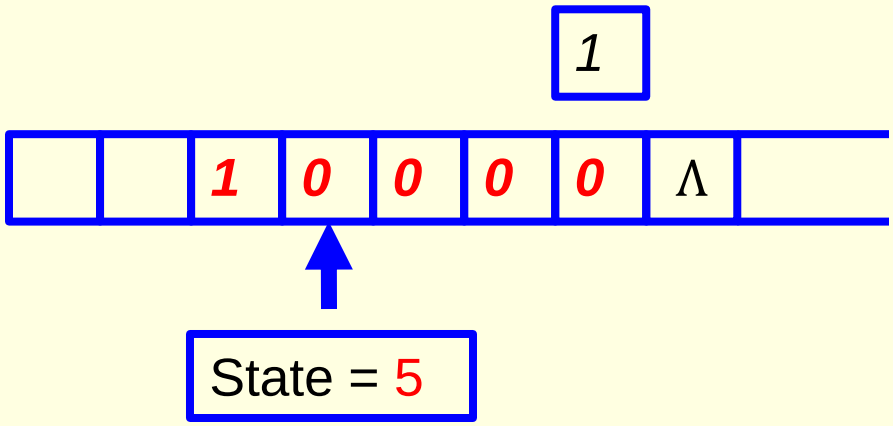


Add 1: case 0-3 (conti)

(4, 0, 1, R, 5)	Move right
(4, 1, 0, L, 4)	Carry
(4, Λ, 1, R. 5)	Move right

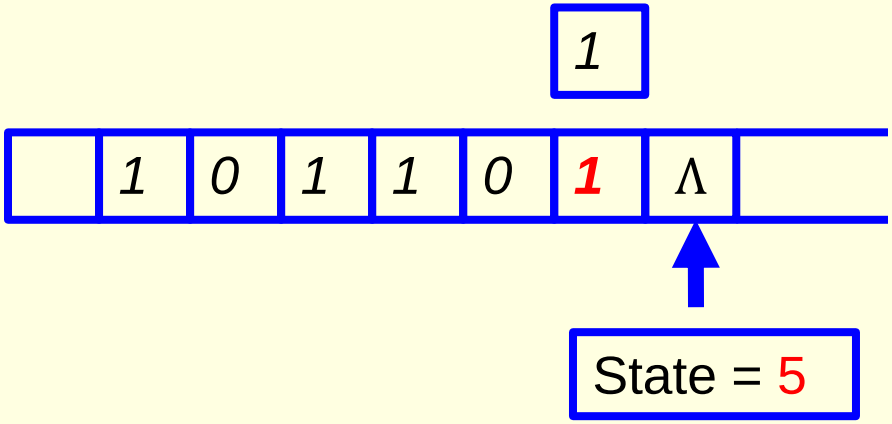


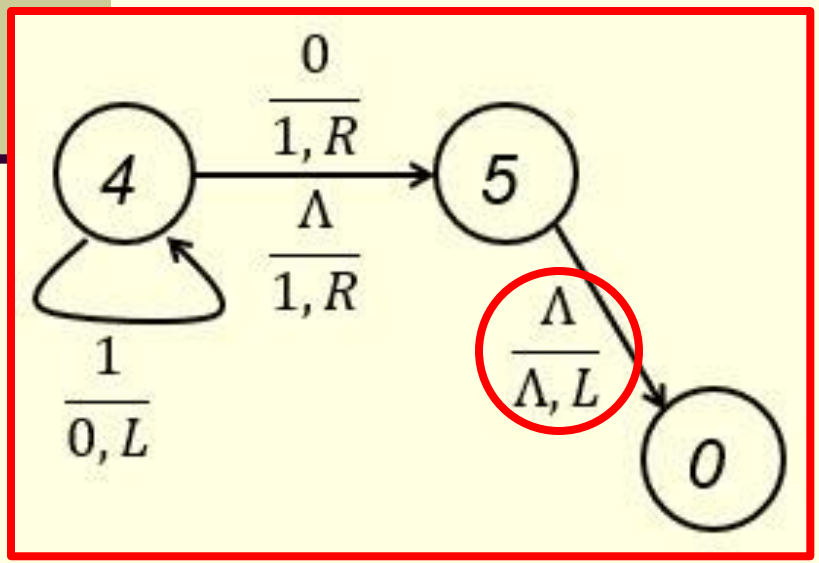


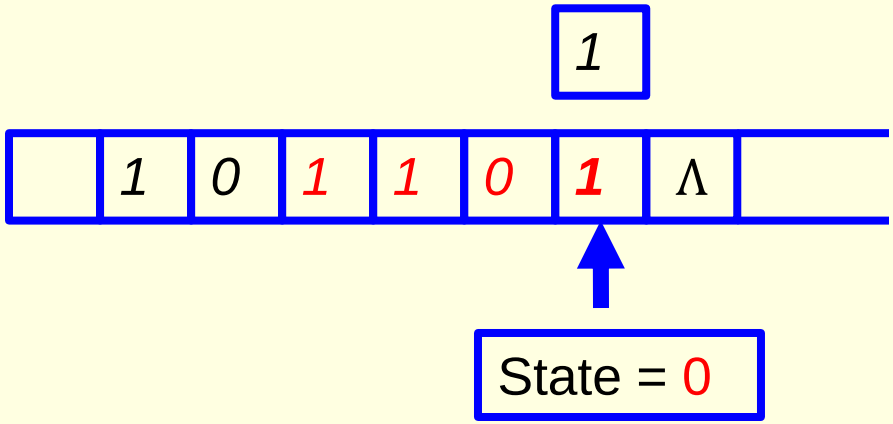


Move back to right end:
Case 0-1

(5, 0, 0, R, 5)	Find right end
(5, 1, 1, R, 5)	Find right end
(5, Λ, Λ, L, 0)	Go add 4

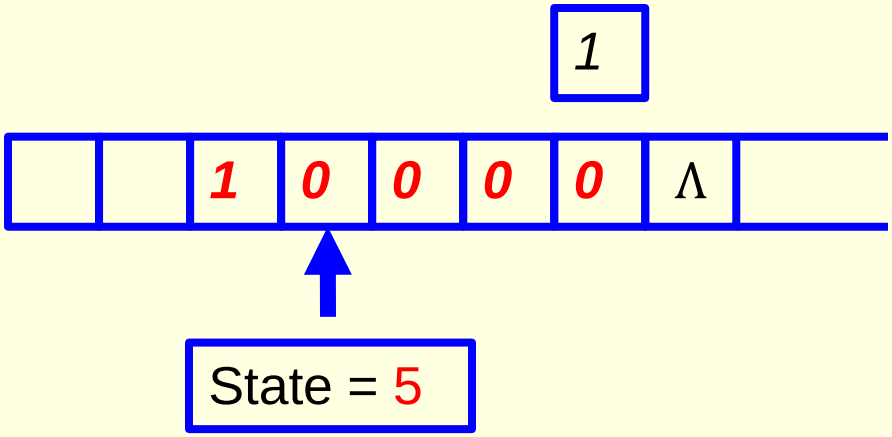


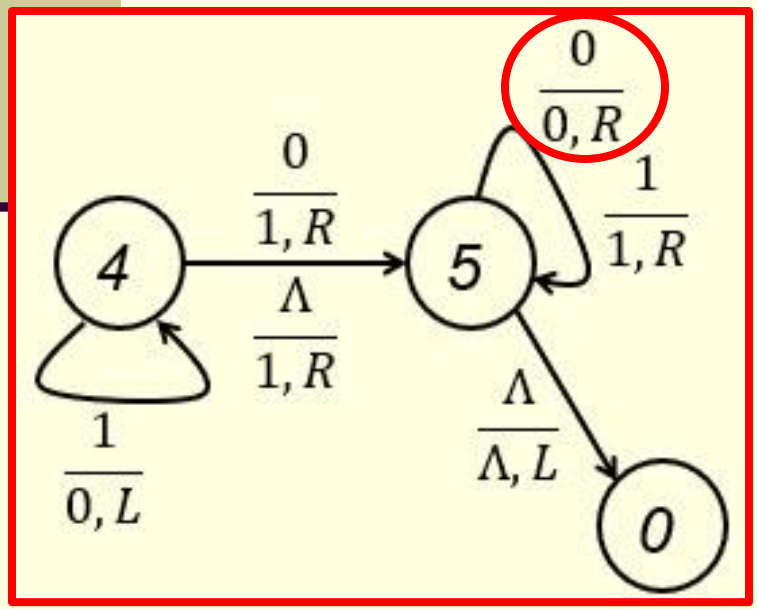


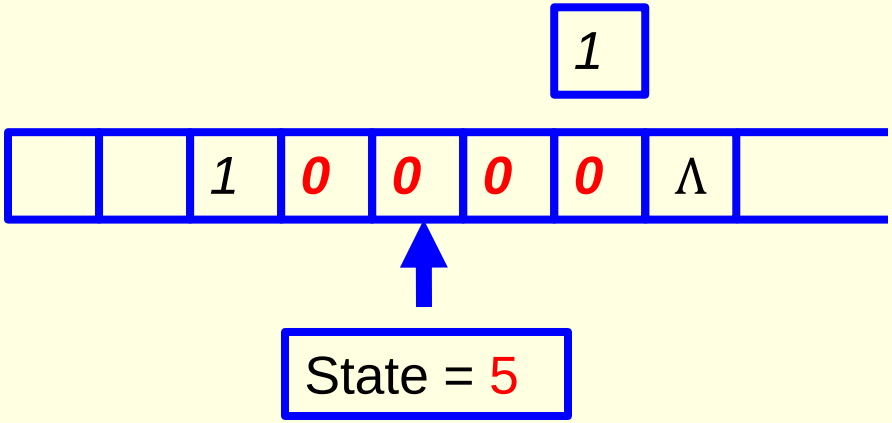


Move back to right end:
Case 0-2

(5, 0, 0, R, 5)	Find right end
(5, 1, 1, R, 5)	Find right end
(5, Λ, Λ, L, 0)	Go add 4

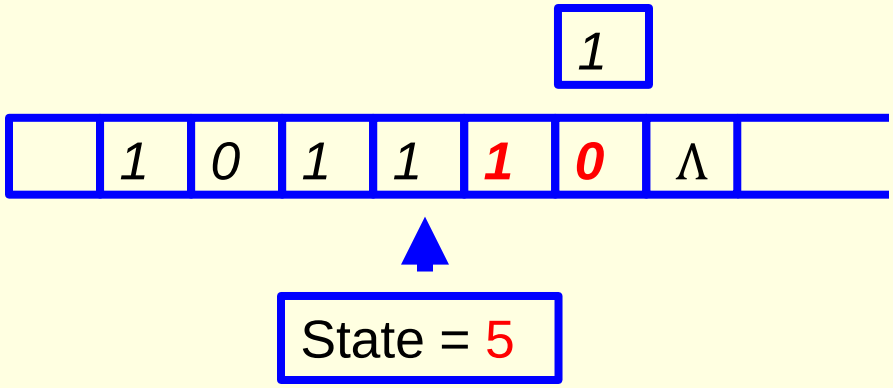


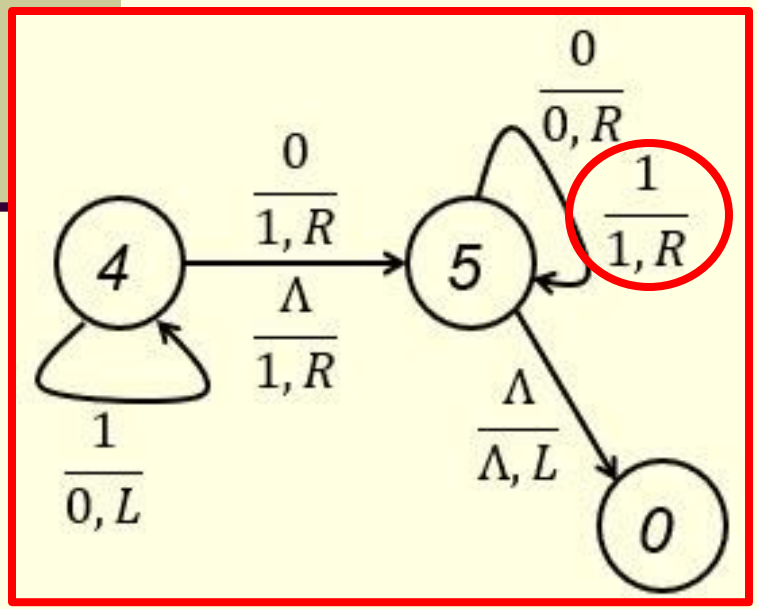


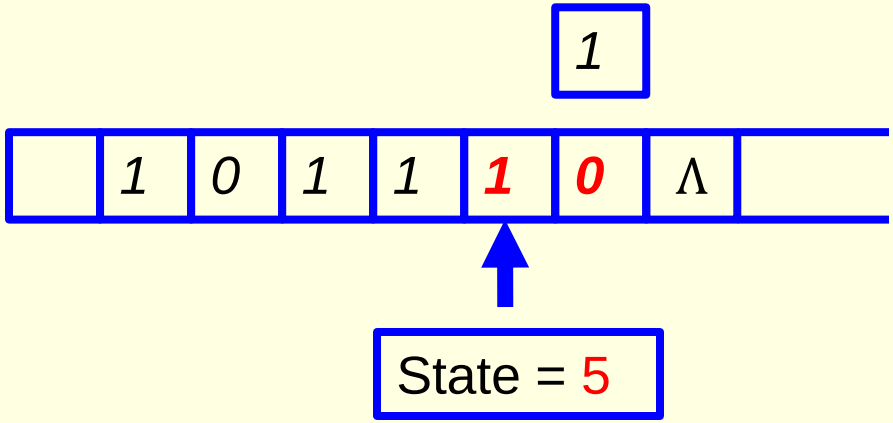


Move back to right end:
Case 0-3

(5, 0, 0, R, 5)	Find right end
(5, 1, 1, R, 5)	Find right end
(5, Λ, Λ, L, 0)	Go add 4







Question: *How do you compute the sum of two given integers x and y, say 43 and 54?*

```
    1 0 1 0 1 1
+   1 1 0 1 1 0
-----------------------
                ?
```

1. *Numbers will be represented in unary format (all zeroes or all ones)*
2. *….*

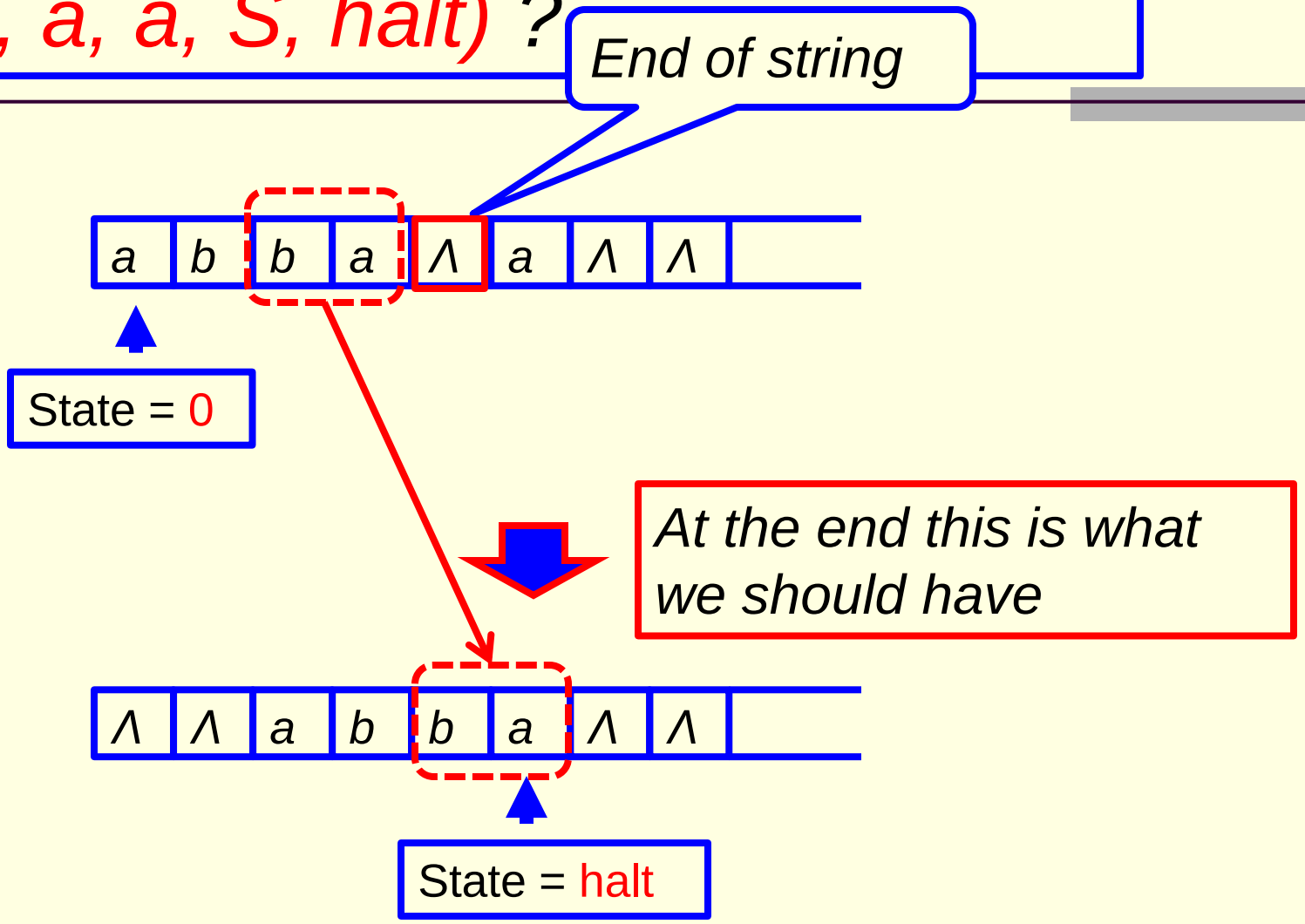
A TM moves a string to the right two cells position

(0, a, Λ, R, 1)	found an a
(0, b, Λ, R, 2)	found a b
(0, Λ, Λ, S, halt)	Done
(1, a, Λ, R, 11)	found aa
(1, b, Λ, R, 12)	found ab
(1, Λ, Λ, R, 10)	found aΛ
(2, a, Λ, R, 21)	found ba
(2, b, Λ, R, 22)	found bb
(2, Λ, Λ, R, 20)	found bΛ

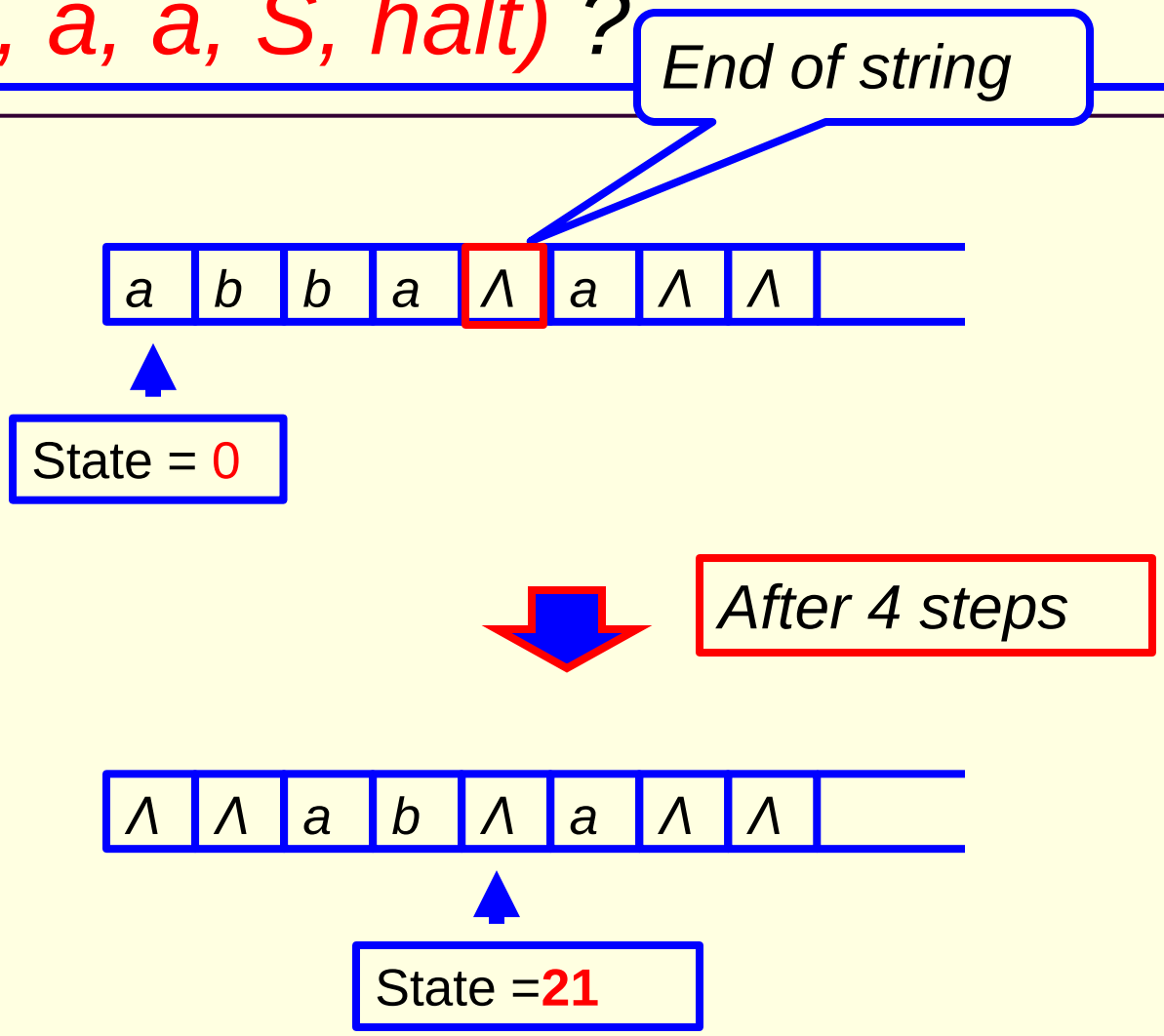
A TM moves a string to the right two cells position **(conti)**

(11, a, a, R, 11)	write an a, aa to go
(11, b, a, R, 12)	write an a, ab to go
(11, Λ, a, R, 10)	write an a, aΛ to go
(12, a, a, R, 21)	write an a, ba to go
(12, b, a, R, 22)	write an a, bb to to
(12, Λ, a, R, 20)	write an a, bΛ to go
(10, Λ, a, S, halt)	write an a, then halt
(10, a, a, S, halt)	write an a, then halt
(10, b, a, S, halt)	write an a, then halt

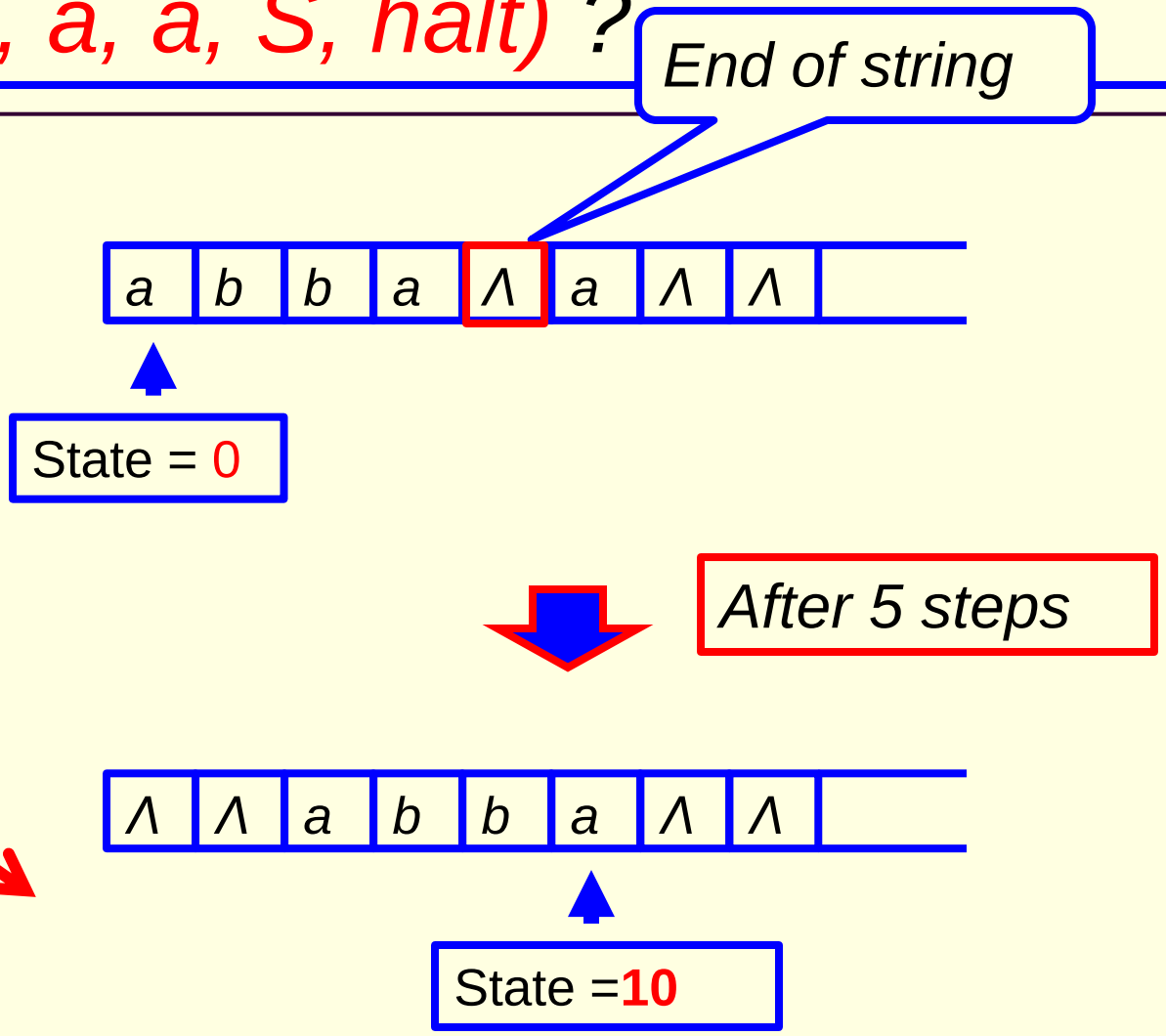
Why do we need this instruction (10, a, a, S, halt) ?



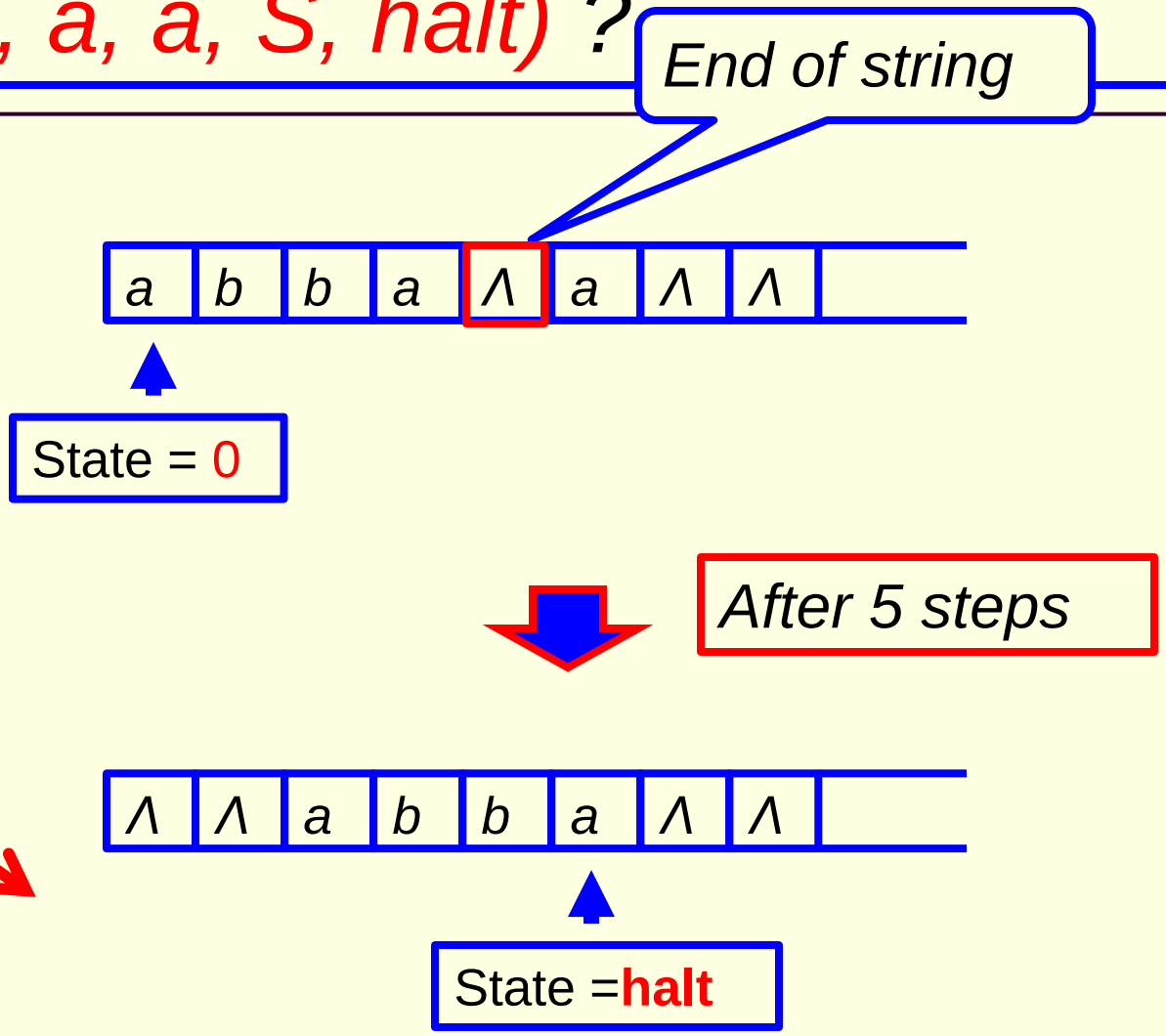
Why do we need this instruction (10, a, a, S, halt) ?



Why do we need this instruction (10, a, a, S, halt) ?



Why do we need this instruction (10, a, a, S, halt) ?

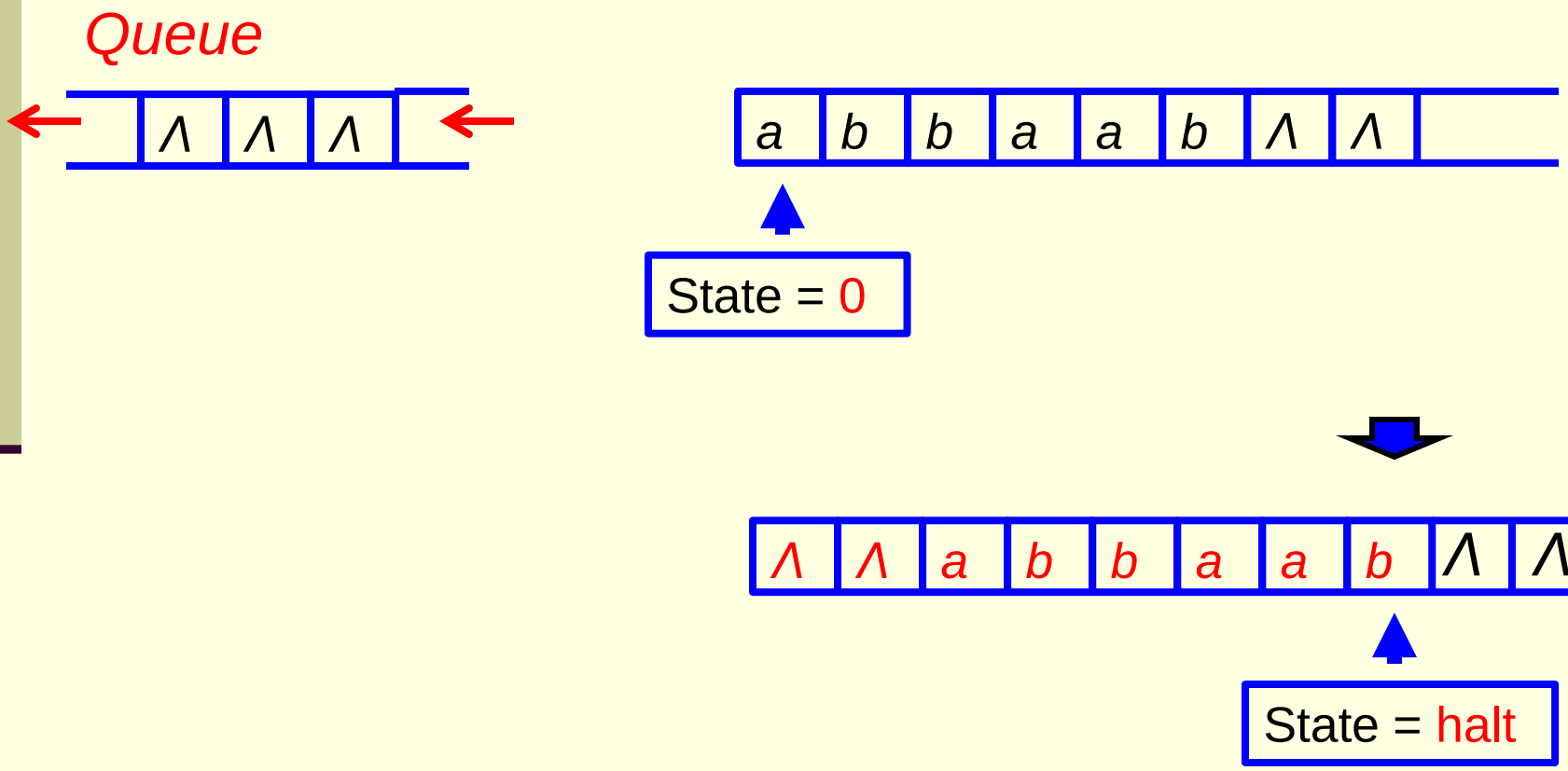


A TM moves a string to the right two cells position **(conti)**

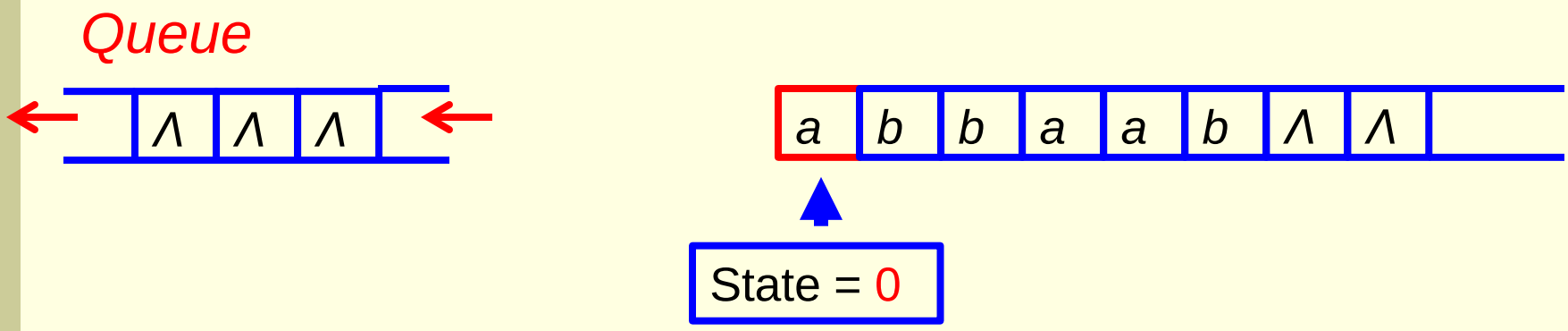
(21, a, b, R, 11)	write a b, aa to go
(21, b, b, R, 12)	write a b, ab to go
(21, Λ, b, R, 10)	write a b, aΛ to go
(22, a, b, R, 21)	write a b, ba to go
(22, b, b, R, 22)	write a b, bb to to
(22, Λ, b, R, 20)	write a b, bΛ to go
(20, Λ, b, S, halt)	write a b, then halt
(20, a, b, S, halt)	write a b, then halt
(20, b, b, S, halt)	write a b, then halt

Question: How are these instructions designed?

First, using a queue with 3 entries to move a string 2 units to the right



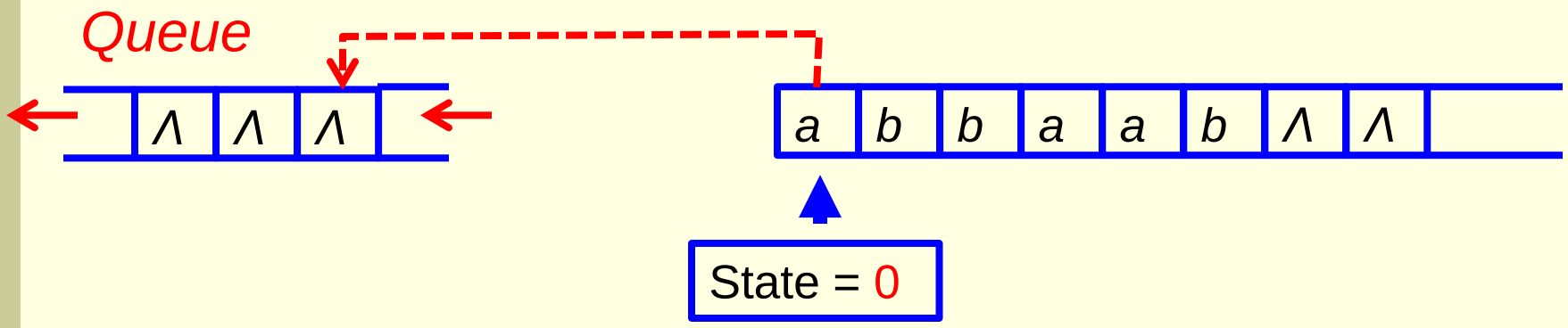
Using a queue with 3 entries to move a string 2 units to the right

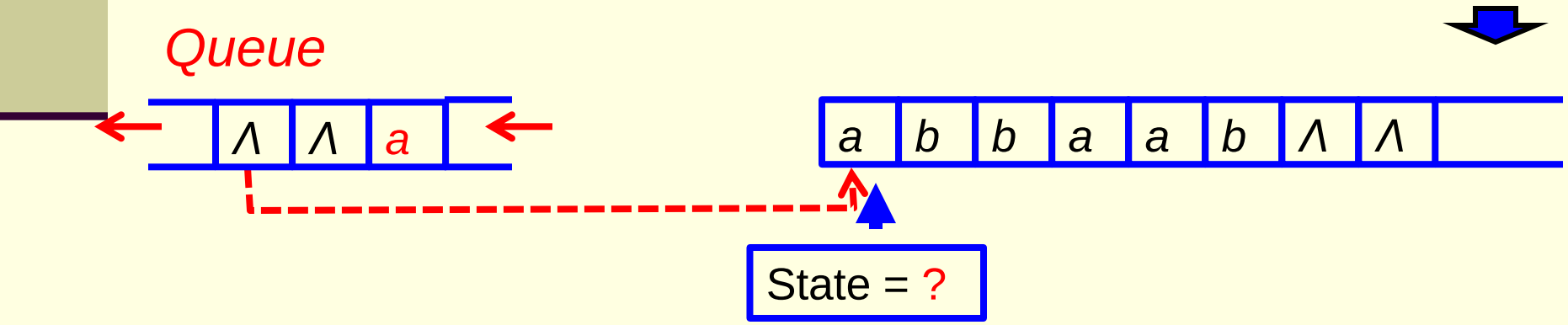


For the cell pointed at by the read/write head, do:

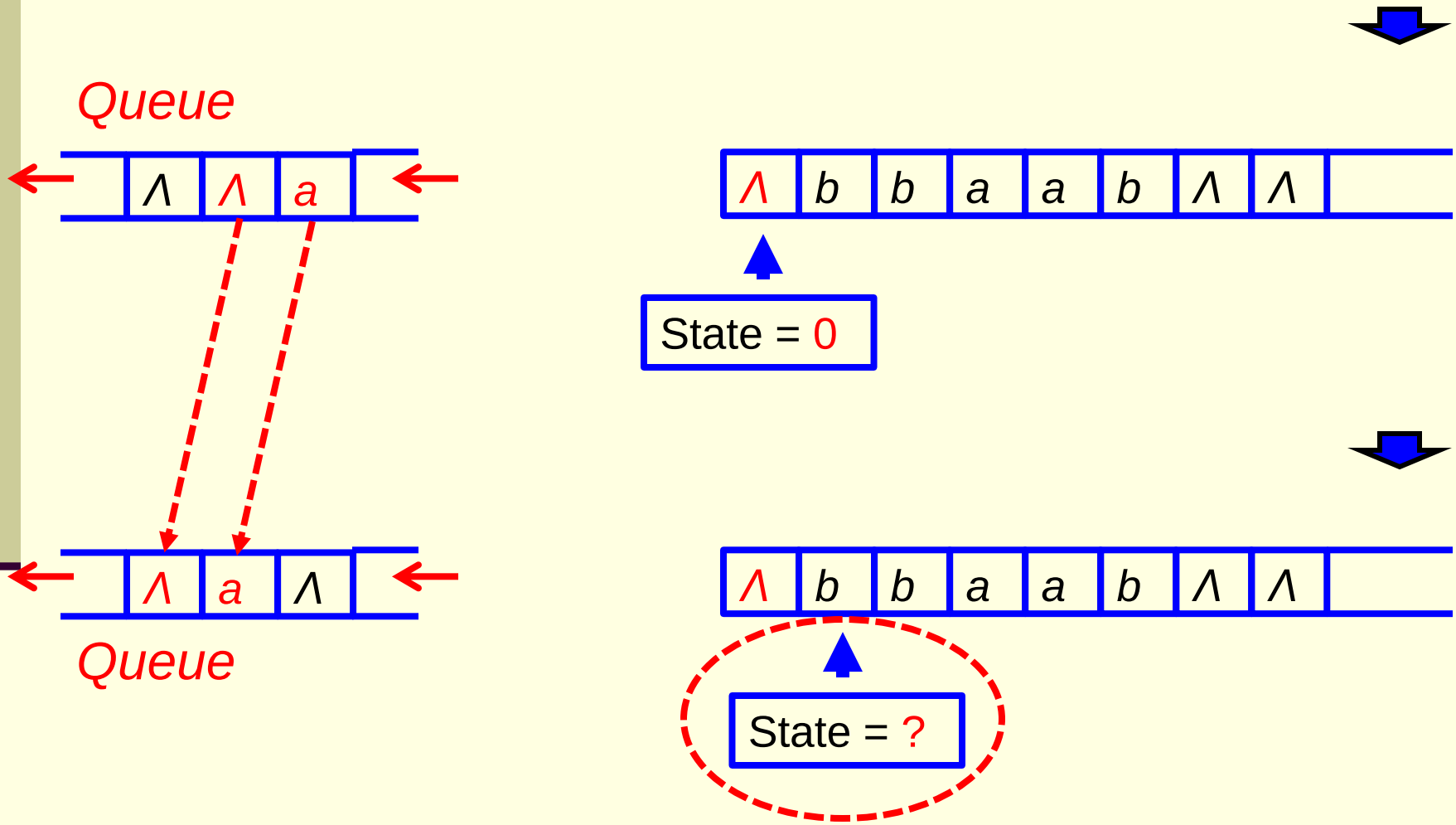
1. *Write content of the cell into 3rd entry of the queue*
2. *Write content of 1st entry of the queue into the cell*
3. *Move 2nd and 3rd entries of the queue one unit to the left*
4. *Move the R/W head one unit to the right*

The procedure: step by step (1)

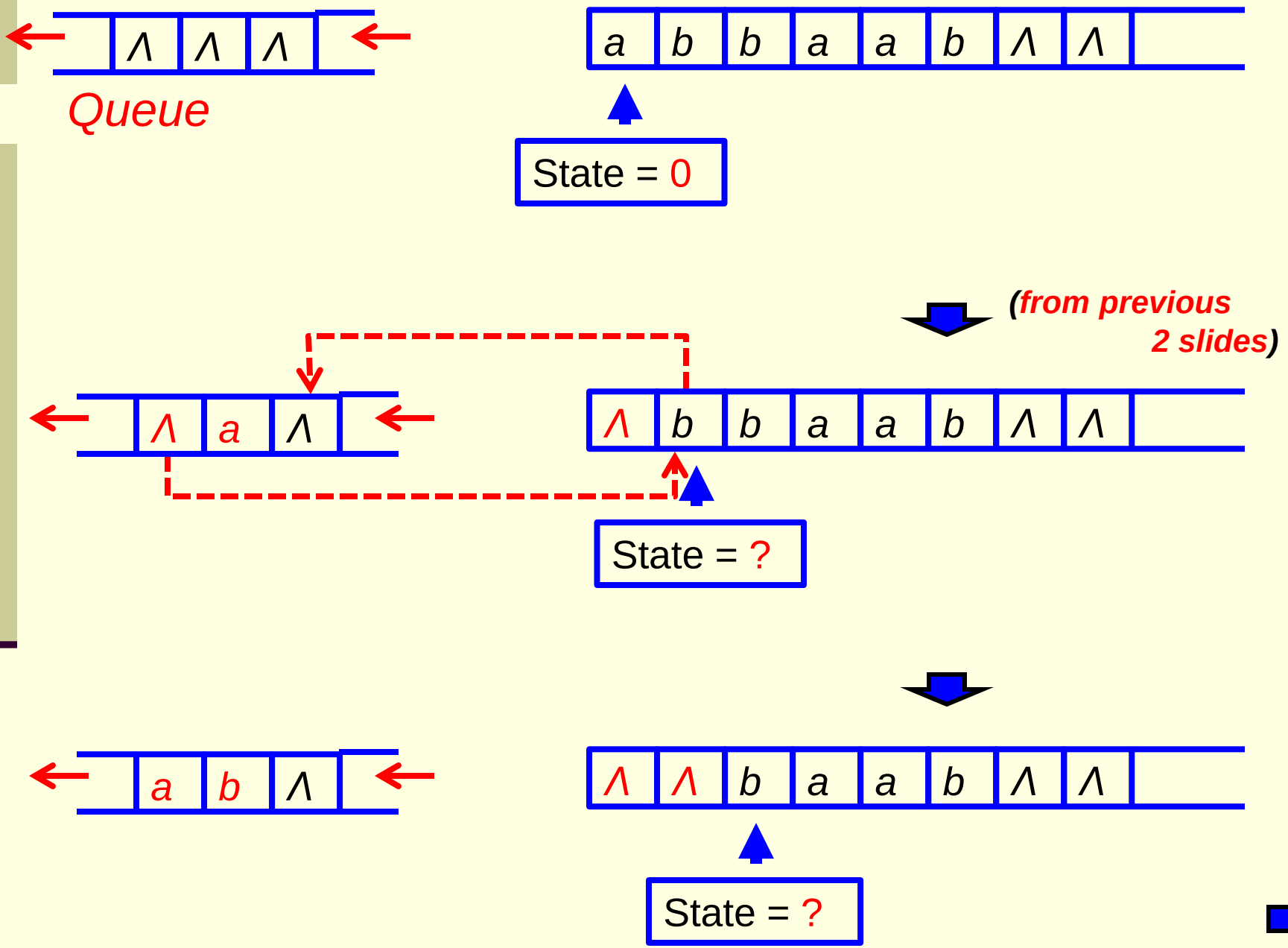




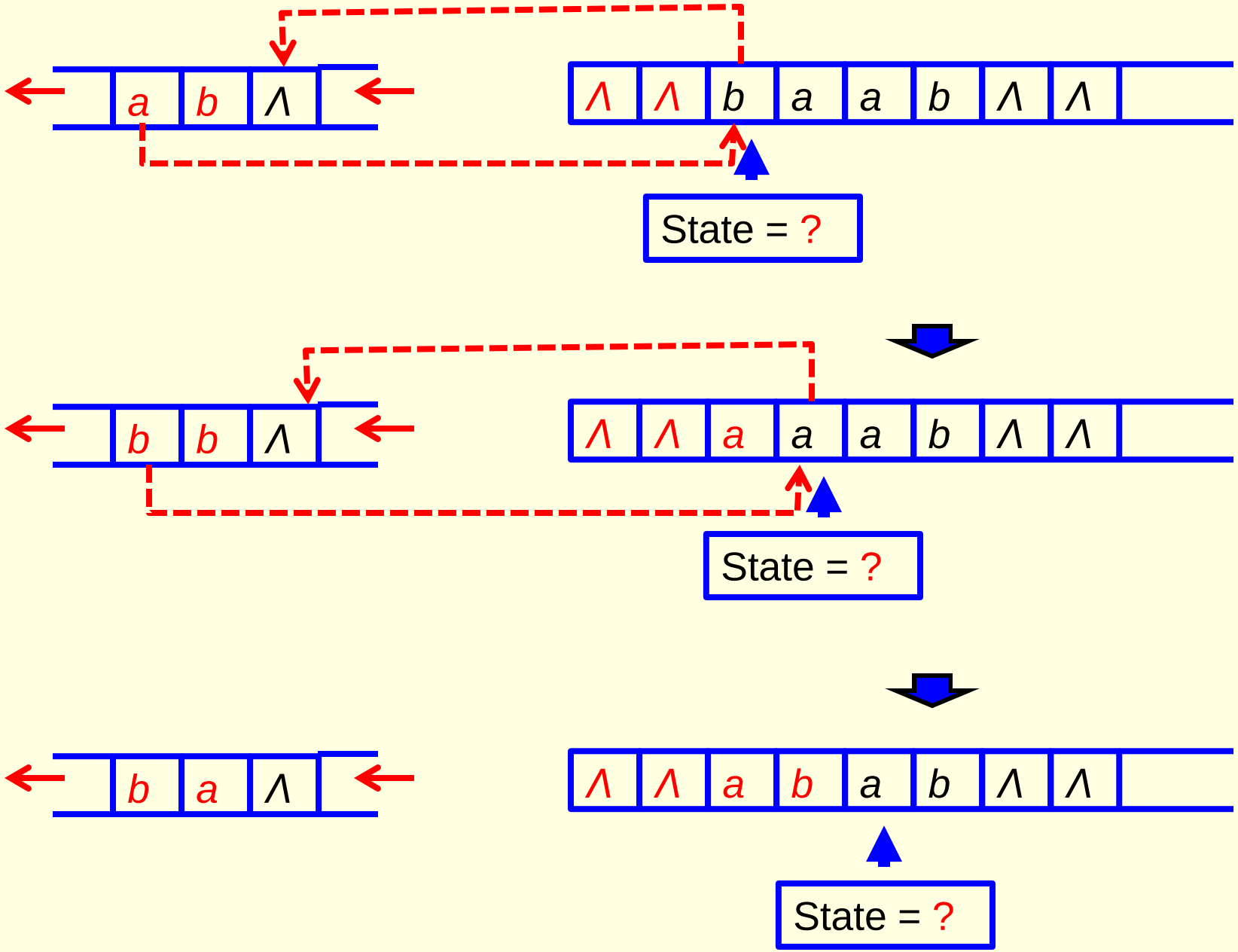
The procedure: step by step (2)



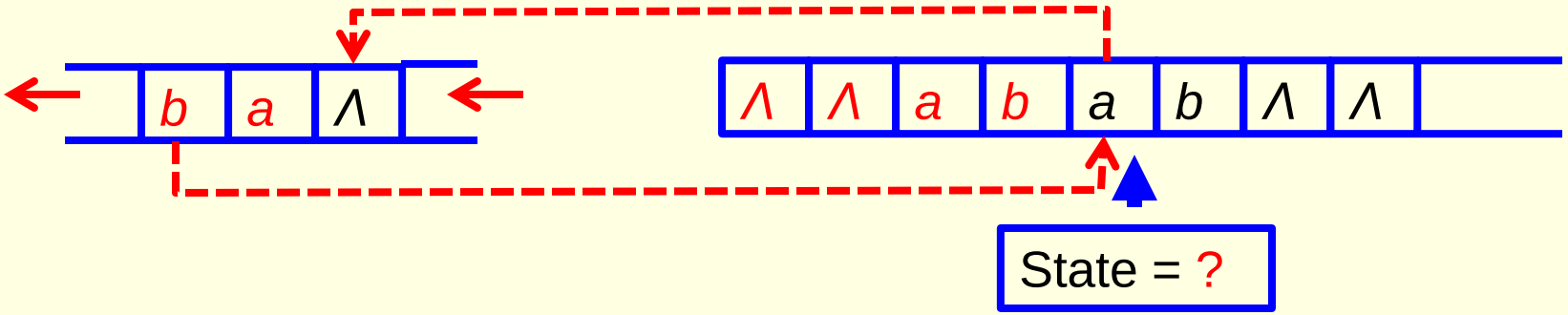
Example (using a queue): (1)

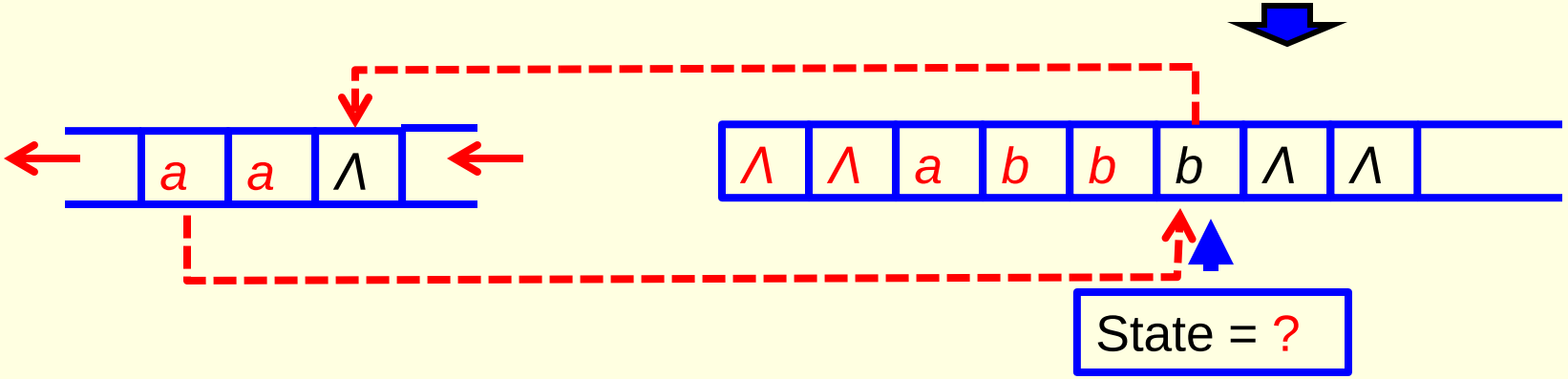


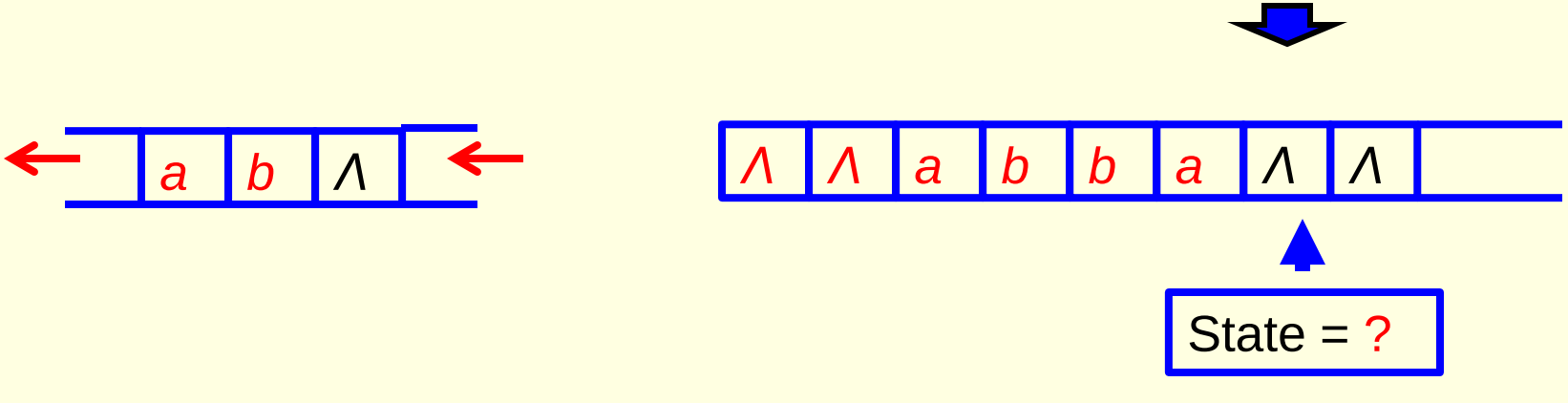
Example (using a queue): (2)



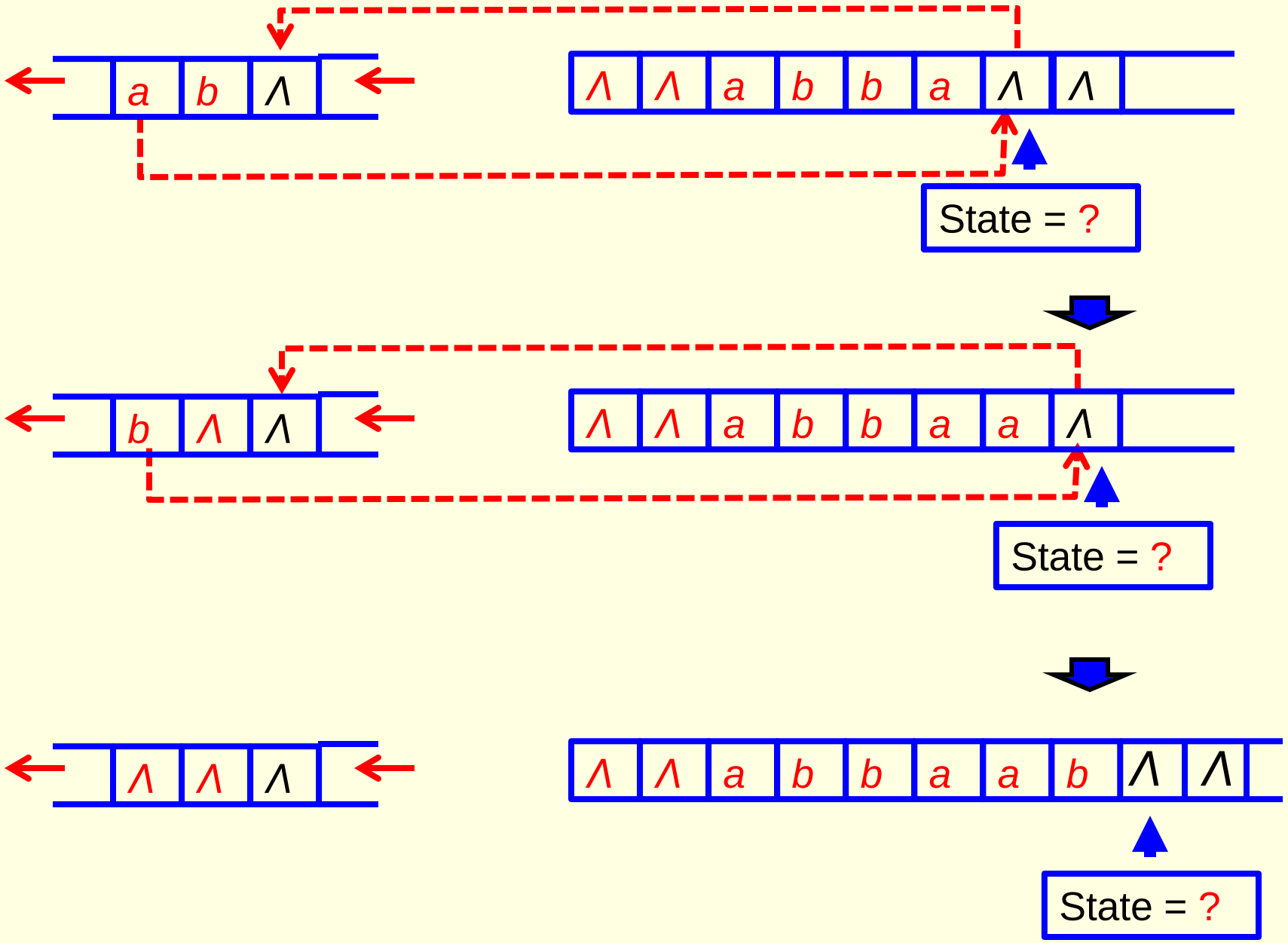
Example (using a queue): (3)





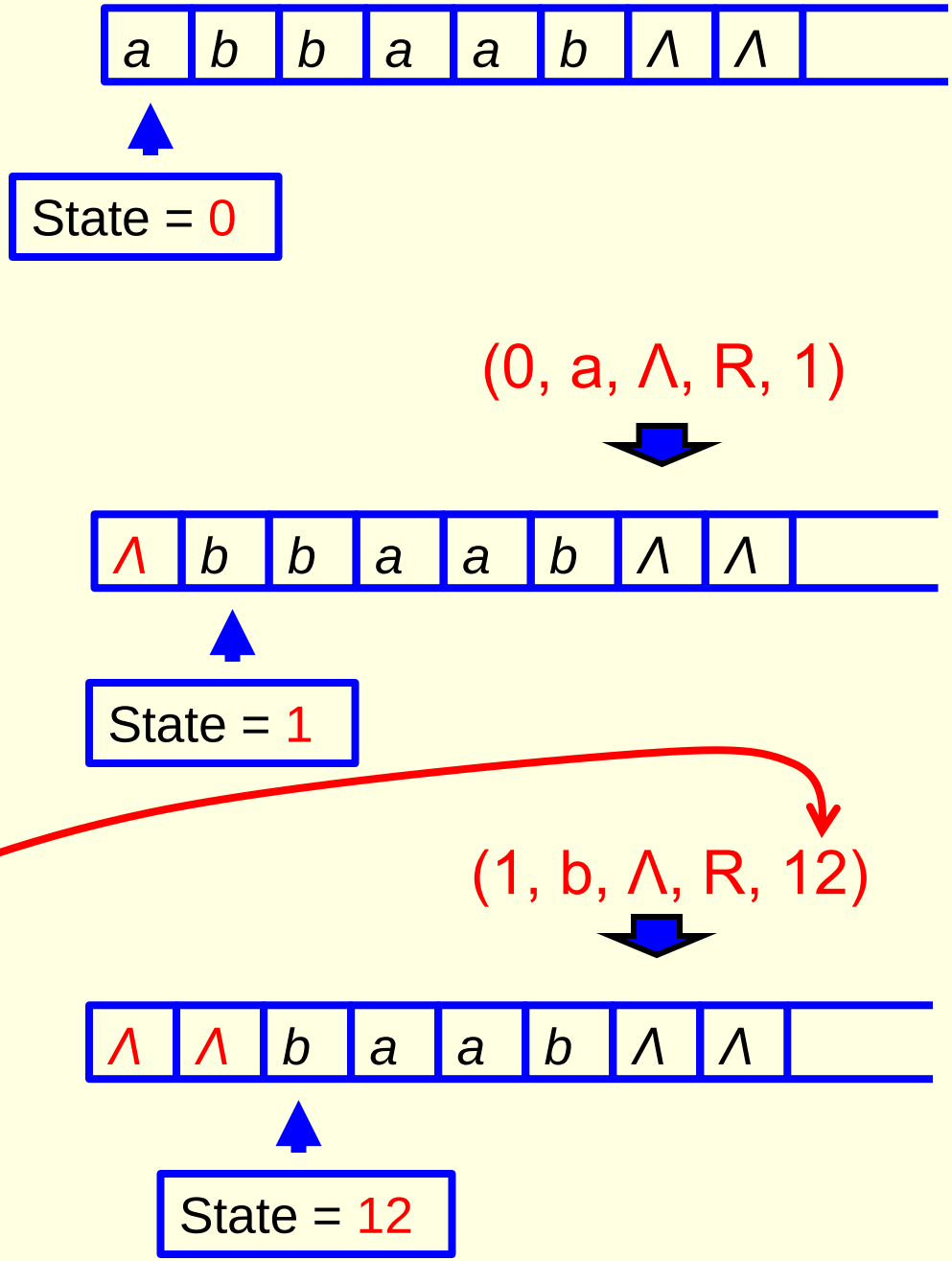


Example (using a queue): (4)

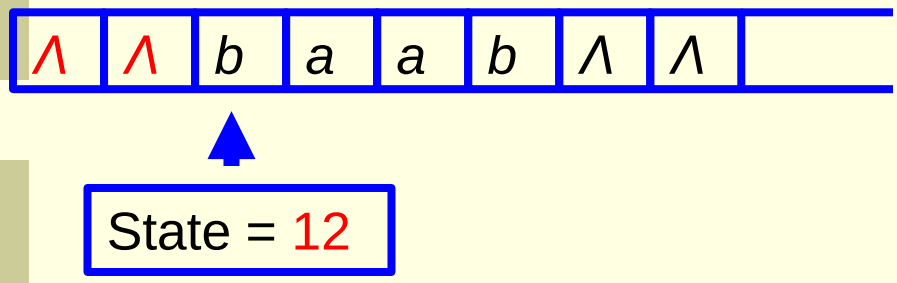


Example (w/o a queue): (1)

The key is to design states of the TM that reflect the contents of the queue at each stage.

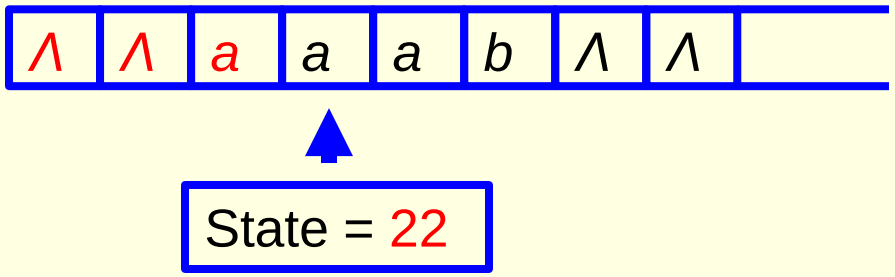


Example (w/o a queue): (2)

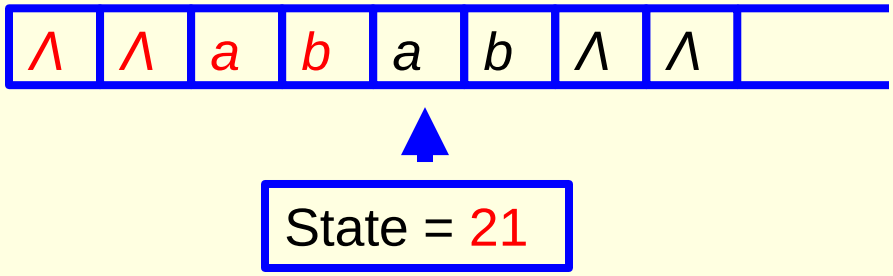


Write an *a*, *bb* to go

(12, b, a, R, 22)

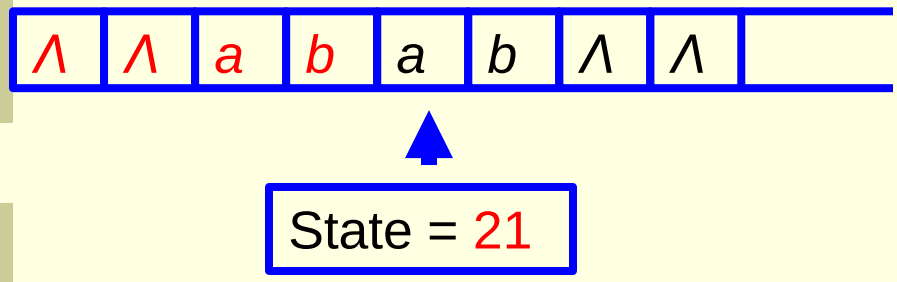


(22, a, b, R, 21)



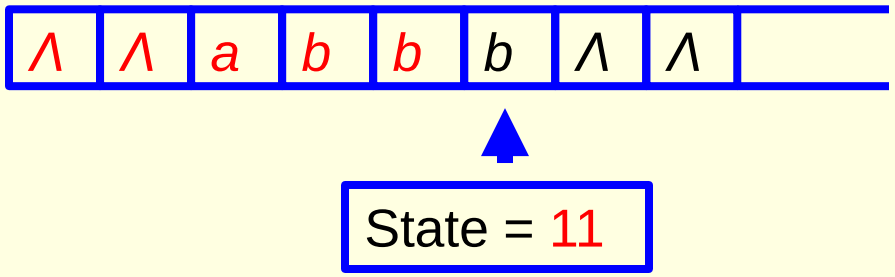
Write a *b*, *ba* to go

Example (w/o a queue): (3)

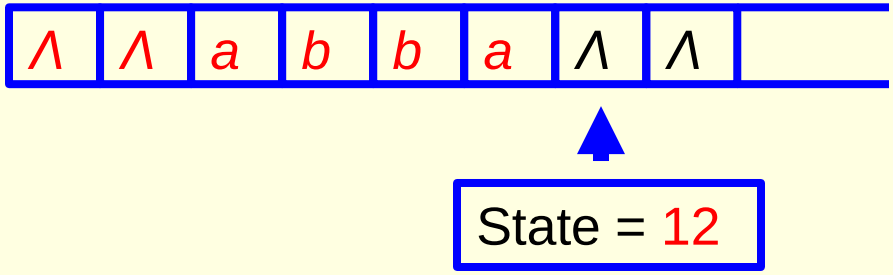


Write a b, aa to go

(21, a, b, R, 11)

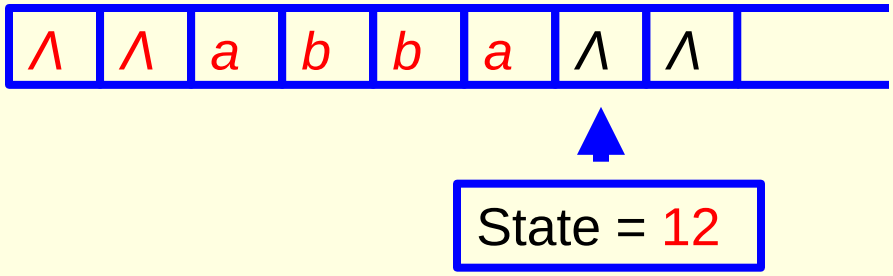


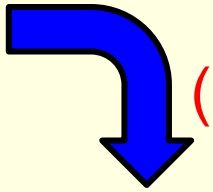
(11, b, a, R, 12)



Write an a, ab to go

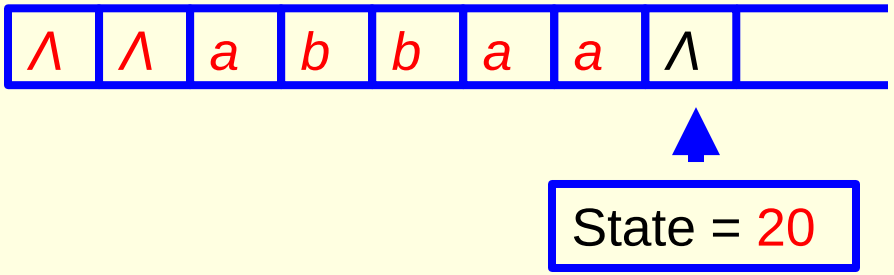
Example (w/o a queue): (4)

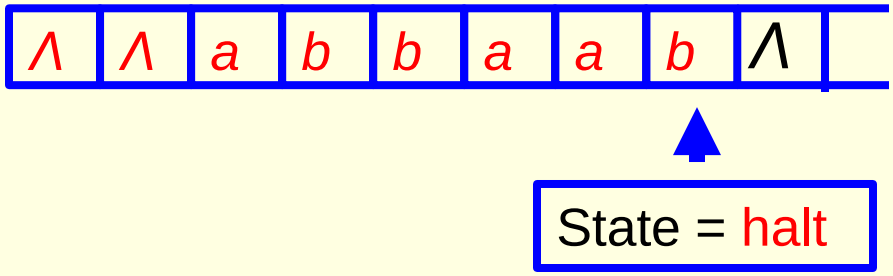


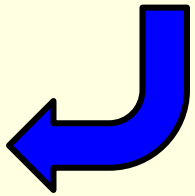


(12, Λ, a, R, 20)

Write an a, bΛ to go



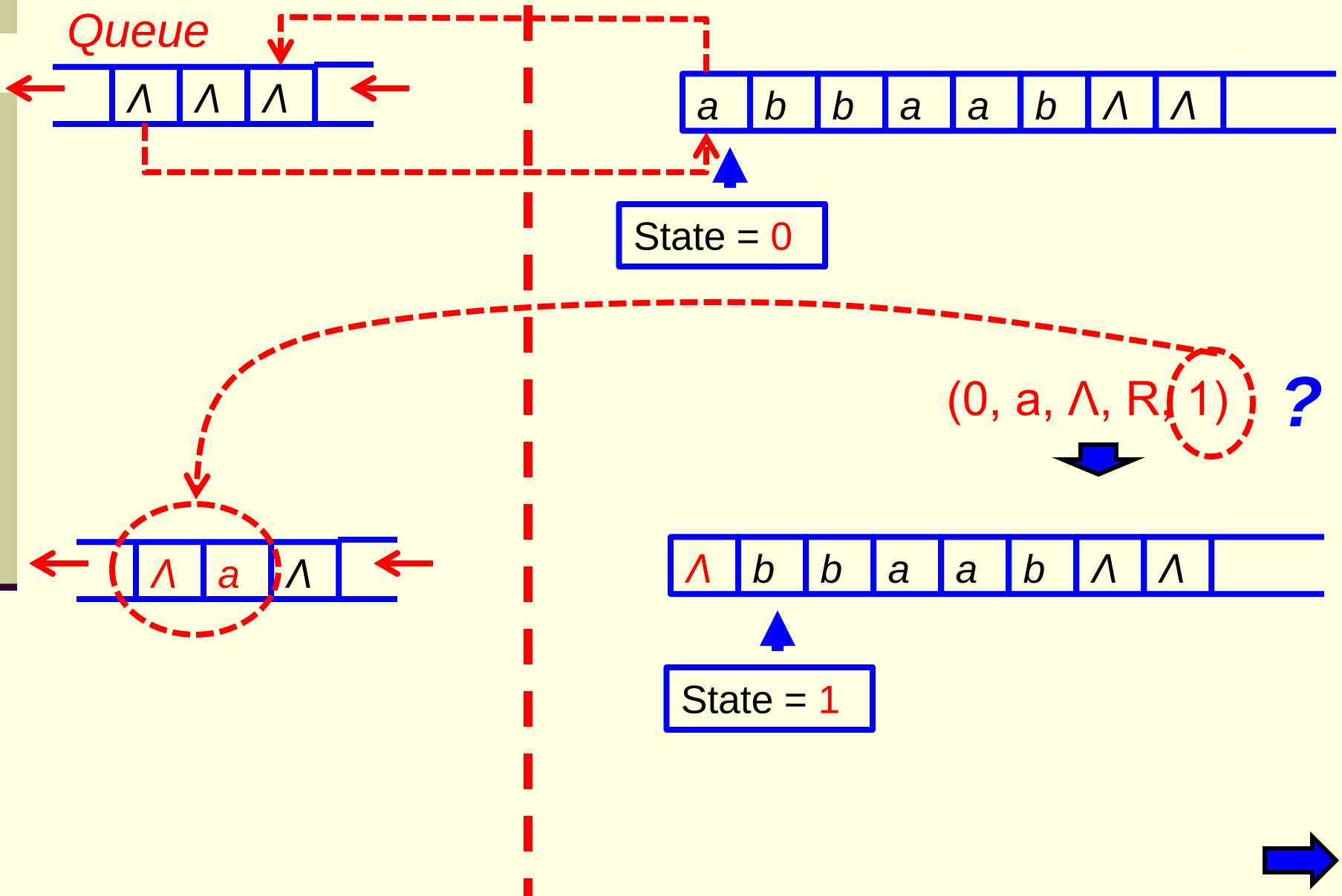




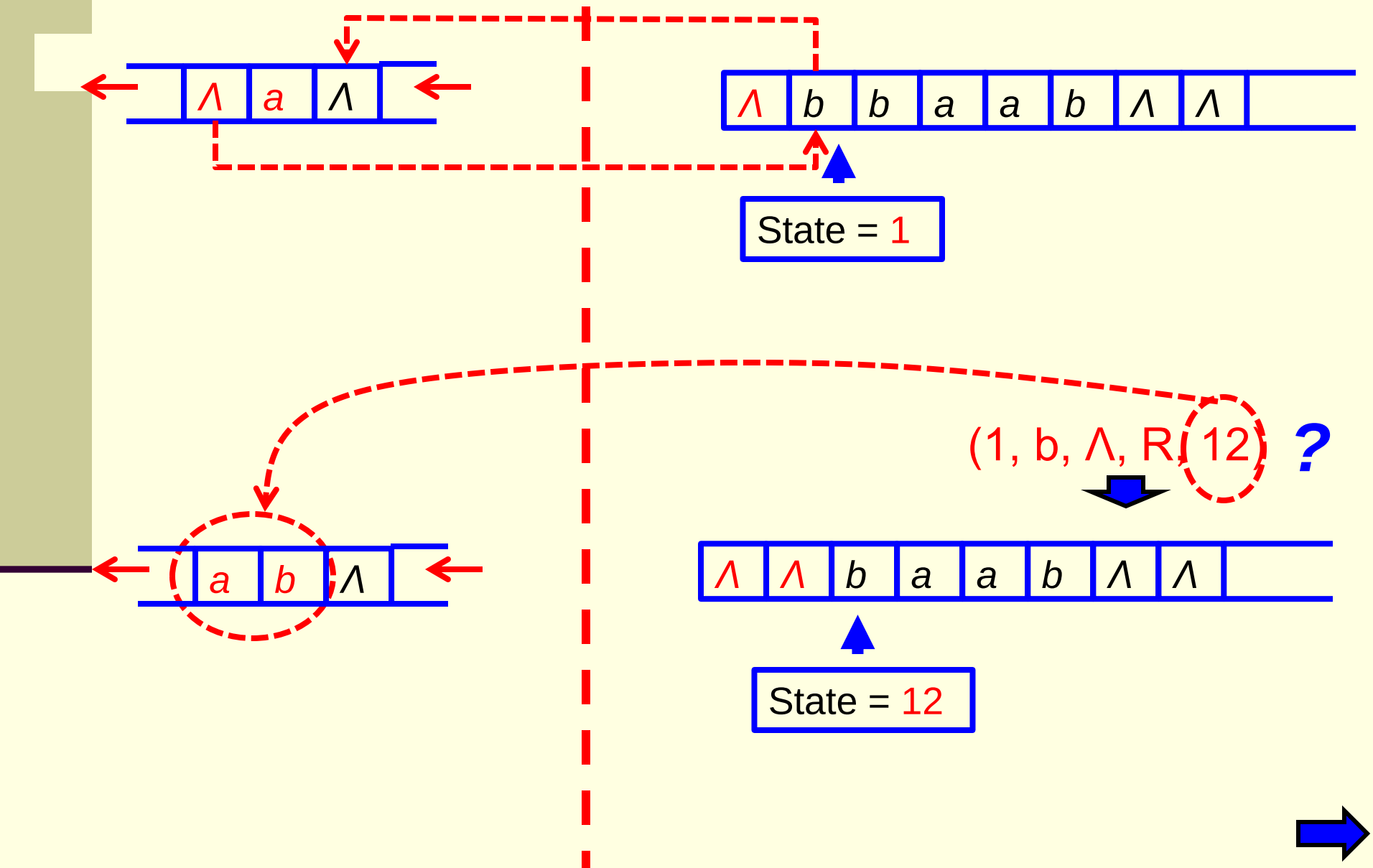
(20, Λ, b, S, halt)

Write a b, then halt

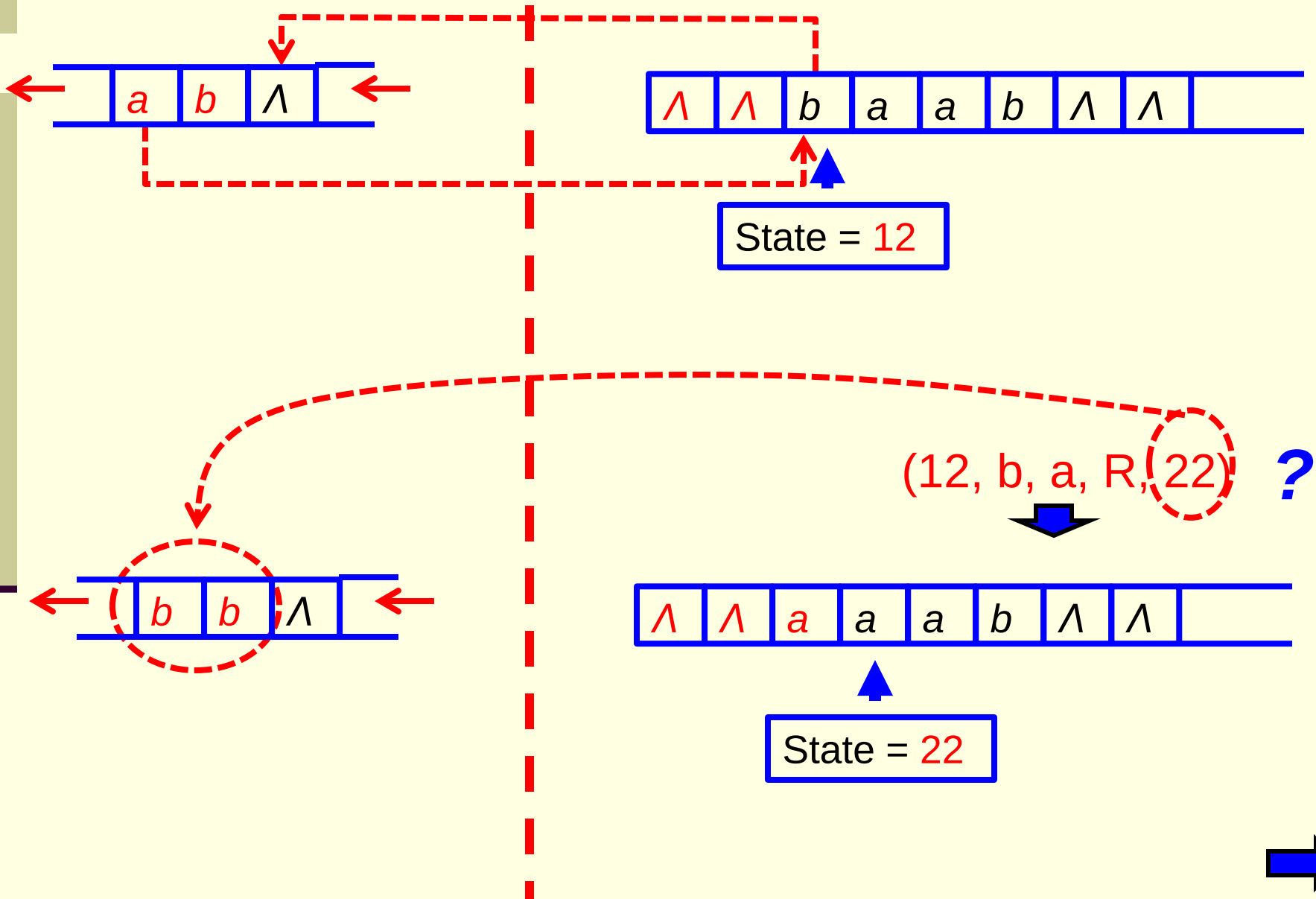
More specifically, why are the instructions (especially the states) designed this way? (1)



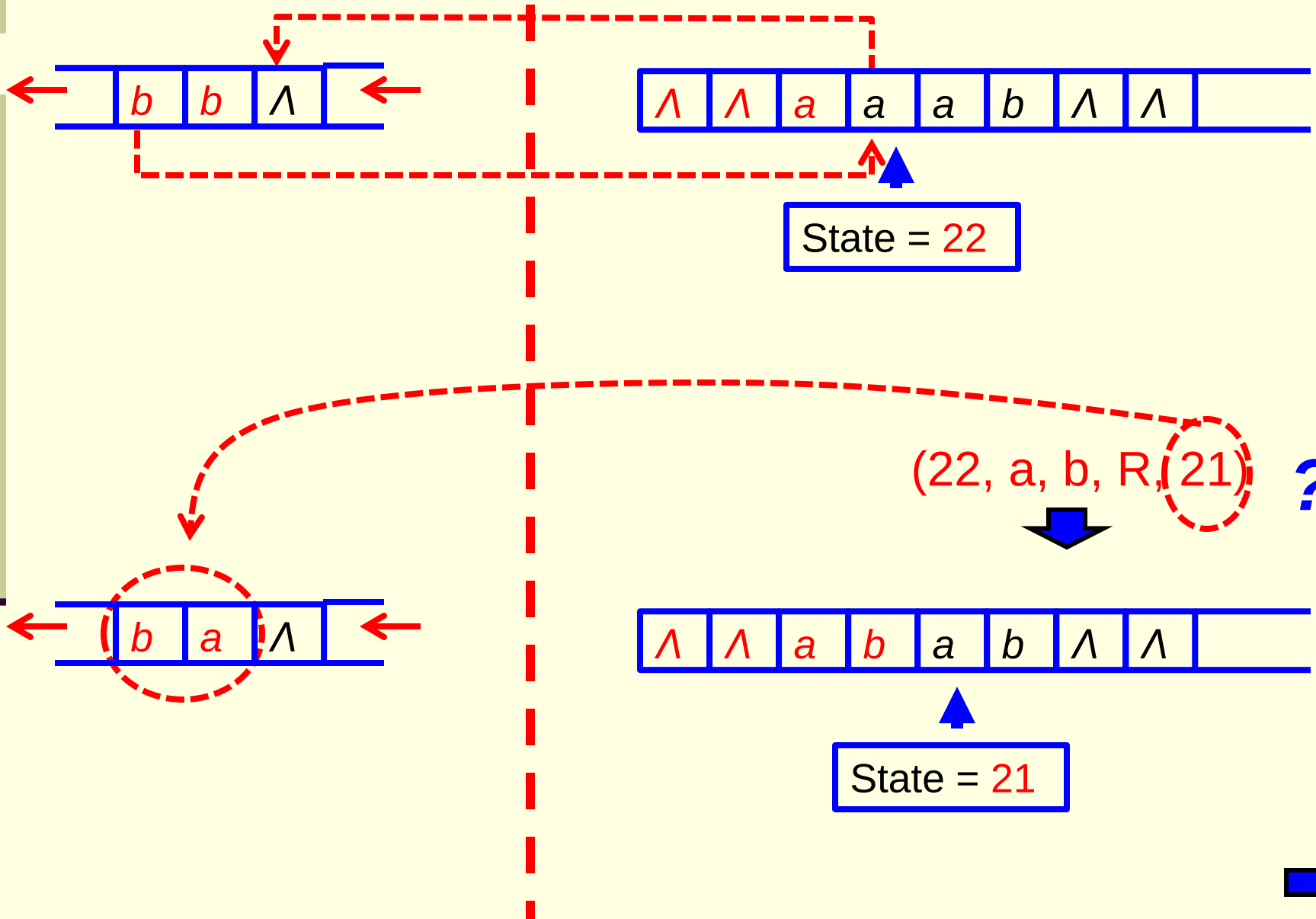
More specifically, why are the instructions (especially the states) designed this way? (2)



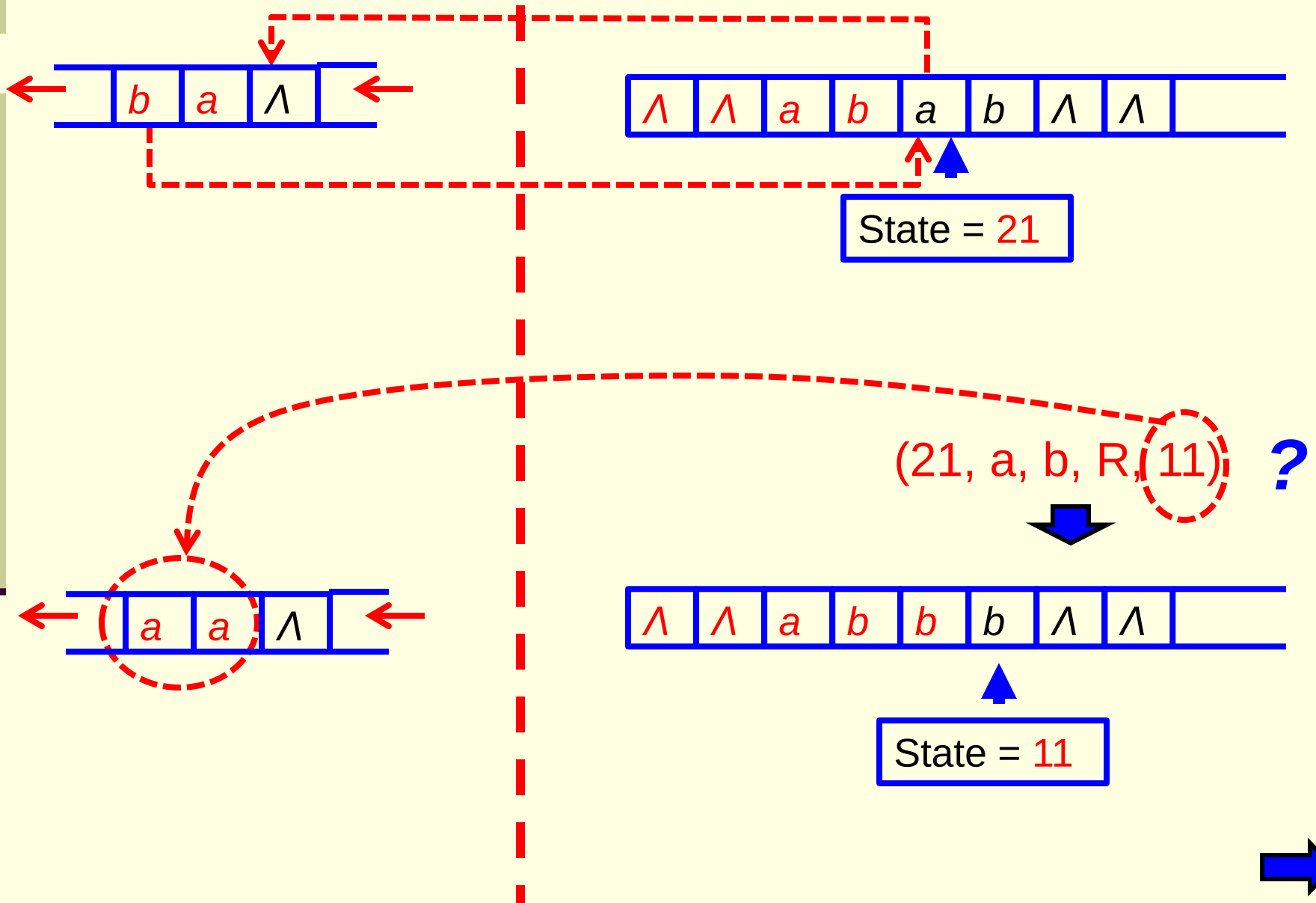
More specifically, why are the instructions (especially the states) designed this way? (3)



More specifically, why are the instructions (especially the states) designed this way? (4)



More specifically, why are the instructions (especially the states) designed this way? (5)



More specifically, why are the instructions (especially the states) designed this way? (5)

So it should be clear now why the instructions, especially the states, are designed the way shown in the instruction set in slides 123-129

More specifically, why are the instructions (especially the states) designed this way? (5),

A question for you:

Why do we need the following instructions In the instruction set of this TM?

(10, a, a, S, halt)
(10, b, a, S, halt)
(20, a, b, S, halt)
(20, b, b, S, halt)

8. Turing Machines - some more history

In the midst of many groundbreaking works, Alan Turing was found dead by his housekeeper in his bed room one morning in 1954.

There was a bitten apple on the nightstand. 

8. Turing Machines - some more history

The autopsy showed that he was poisoned by cyanide and the apple was soaked in cyanide solvent.

He was 42, only 42, that year.

Four people attended his funeral. One of them was his mom.

8. Turing Machines - some more history

Four years after Alan Turing's death, the British government abolished the law convicting homosexuality.

8. Turing Machines - some more history

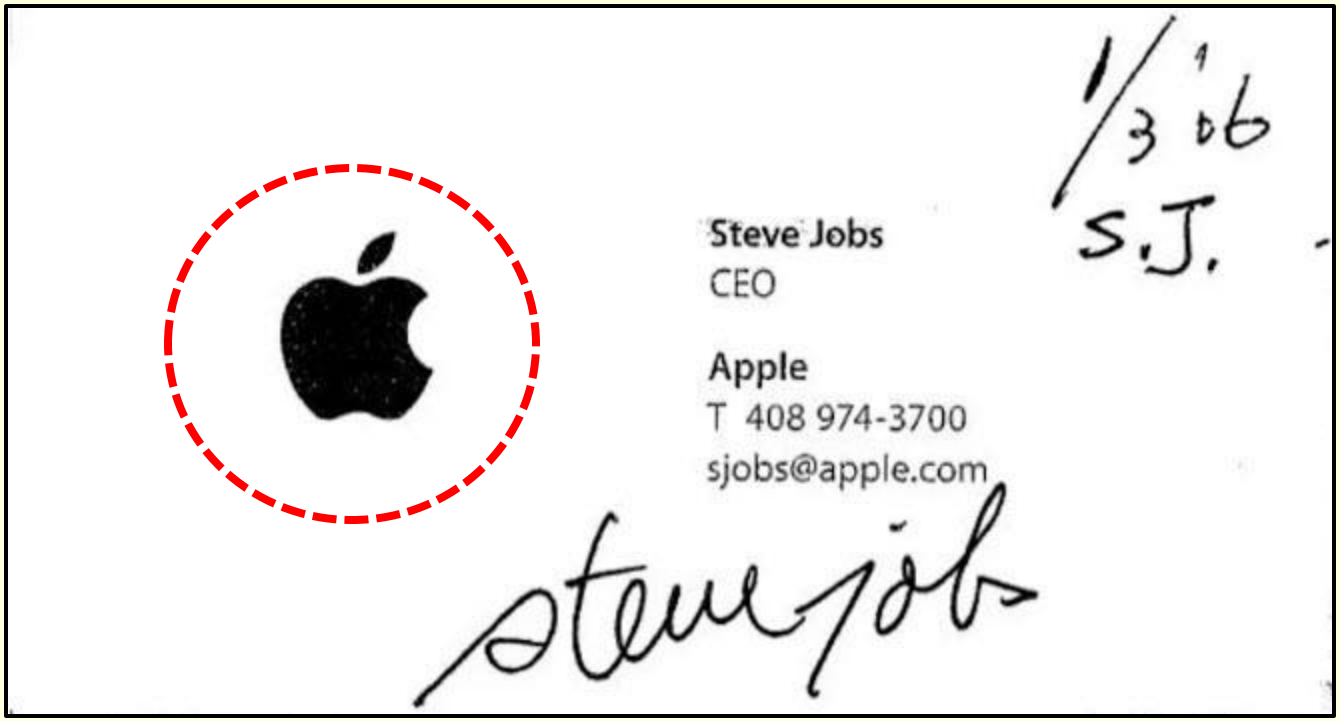
Twenty two years after his death (1976),

a young man named Steve Jobs,

a fan of Alan Turing,

co-founded a company called Apple, with a bitten apple as its logo.

8. Turing Machines - some more history



8. Turing Machines - some more history

By the early 21st century Turing's prosecution for being gay had become infamous.

In 2009 British Prime Minister Gordon Brown, speaking on behalf of the British government, publicly apologized for Turing's "utterly unfair" treatment.

2013

Four years later Queen Elizabeth II granted Turing a royal pardon.

Skip slides 155-162

Millennium Prize Problems

P versus NP problem

Hodge conjecture

Poincaré conjecture (solved)

Riemann hypothesis

Yang–Mills existence and mass gap

Navier–Stokes existence and smoothness

Birch and Swinnerton-Dyer conjecture

The P versus NP problem:

- a major unsolved problem in computer science
- It asks whether every problem whose solution can be quickly verified can also be solved quickly.
- *quickly* means the existence of an algorithm solving the task that runs in **polynomial time**
- The class of questions for which some algorithm can provide an answer in polynomial time is called "class **P**" or just "**P**"

The P versus NP problem:

An example of a "**P-problem**"

Given a list of n integers and an integer k, is there an integer in the list greater than k?

O(n)

The P versus NP problem:

An example of a "**P-problem**"

Given a list of n integers and an integer k, is there an integer m ($m \leq n$) such that the sum of m consecutive integers in the list is greater than or equal to k?

$$O(n^2)$$

The P versus NP problem:

$\{d_1, d_2, d_3, \cdots, d_n\}$, k, $d = max\{d_1, d_2, d_3, \cdots, d_n\}$

$$
\begin{array}{lll}
d_1 & & \geq k \\
d_1 + d_2 & & \geq k \\
d_1 + d_2 + d_3 & & \geq k \\
d_1 + d_2 + d_3 + d_4 & & \geq k \\
\cdots & & \vdots \\
\cdots & & \vdots \\
d_1 + d_2 + d_3 + d_4 \cdots + d_n & & \geq k \\
\hline
\leq \quad nd + nd + nd + nd \cdots + nd & = (n^2)d &
\end{array}
$$

$O(n^2)$

The P versus NP problem:

- For some questions, there is no known way to find an answer quickly, but if one is provided with information showing what the answer is, it is possible to verify the answer quickly.
- The class of questions for which an answer can be *verified* in polynomial time is called **NP**, which stands for "nondeterministic polynomial time"

Difference between verifying an answer and finding an answer

For instance, it can be done quickly to check if 1 is a root of the following equation

$$3x^5 + 7x^4 - 2x^2 + 11 = 0$$

But it is not so easy to solve this equation to find its roots.
We don't know if we can solve it in minutes, in hours, or even in days.

The P versus NP problem:

Polynomial time (an example)

Data size: n

Execution time: $O(3n^2 + 4n + 5)$

$= \mathbf{O(n^2)}$

The P versus NP problem:

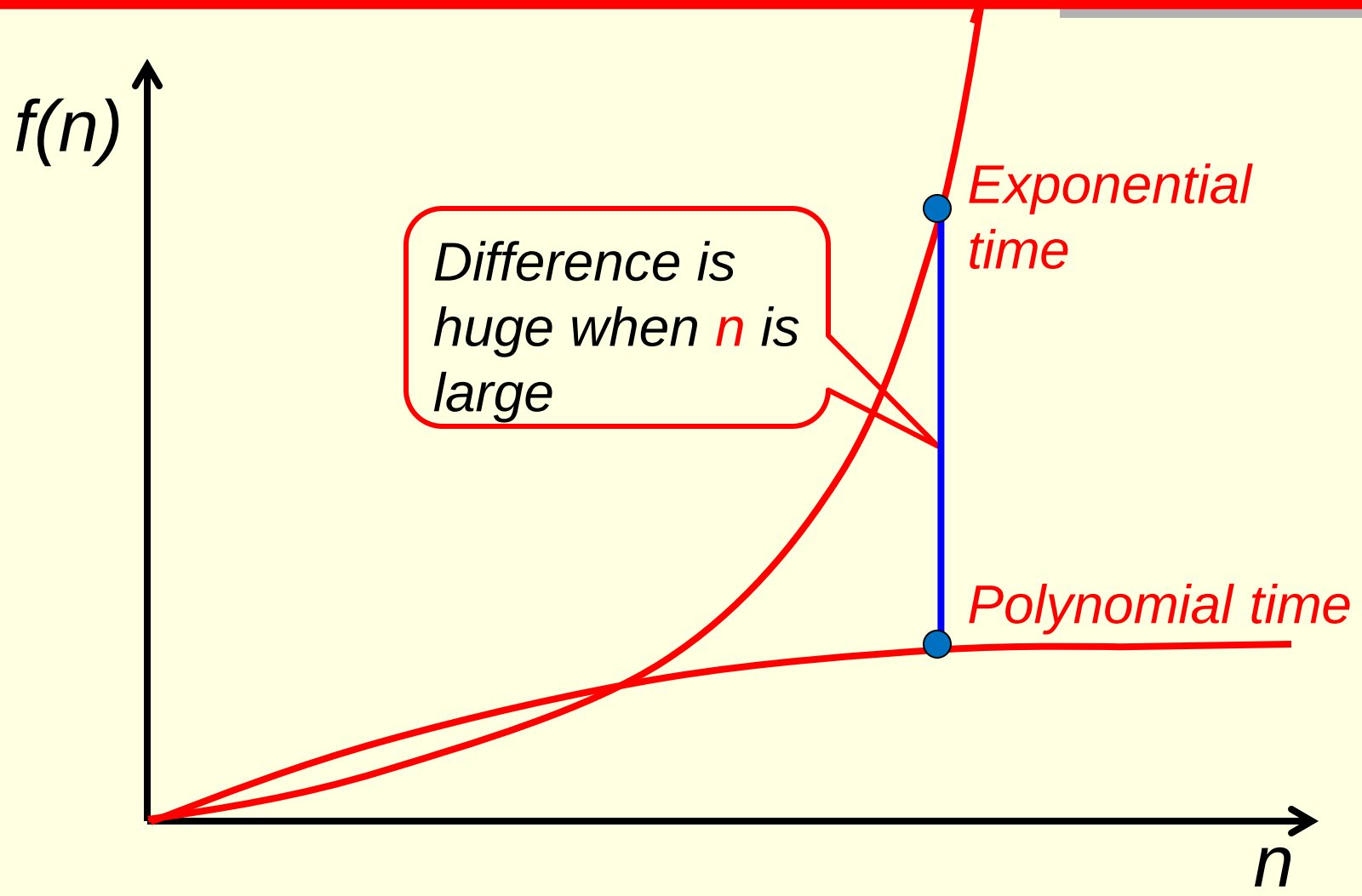
Exponential time (an example)

Data size: n

Execution time: $O(3 * 2^n + 4n + 5)$

$= \mathbf{O(2^n)}$

Polynomial time vs Exponential time:



The P versus NP problem:

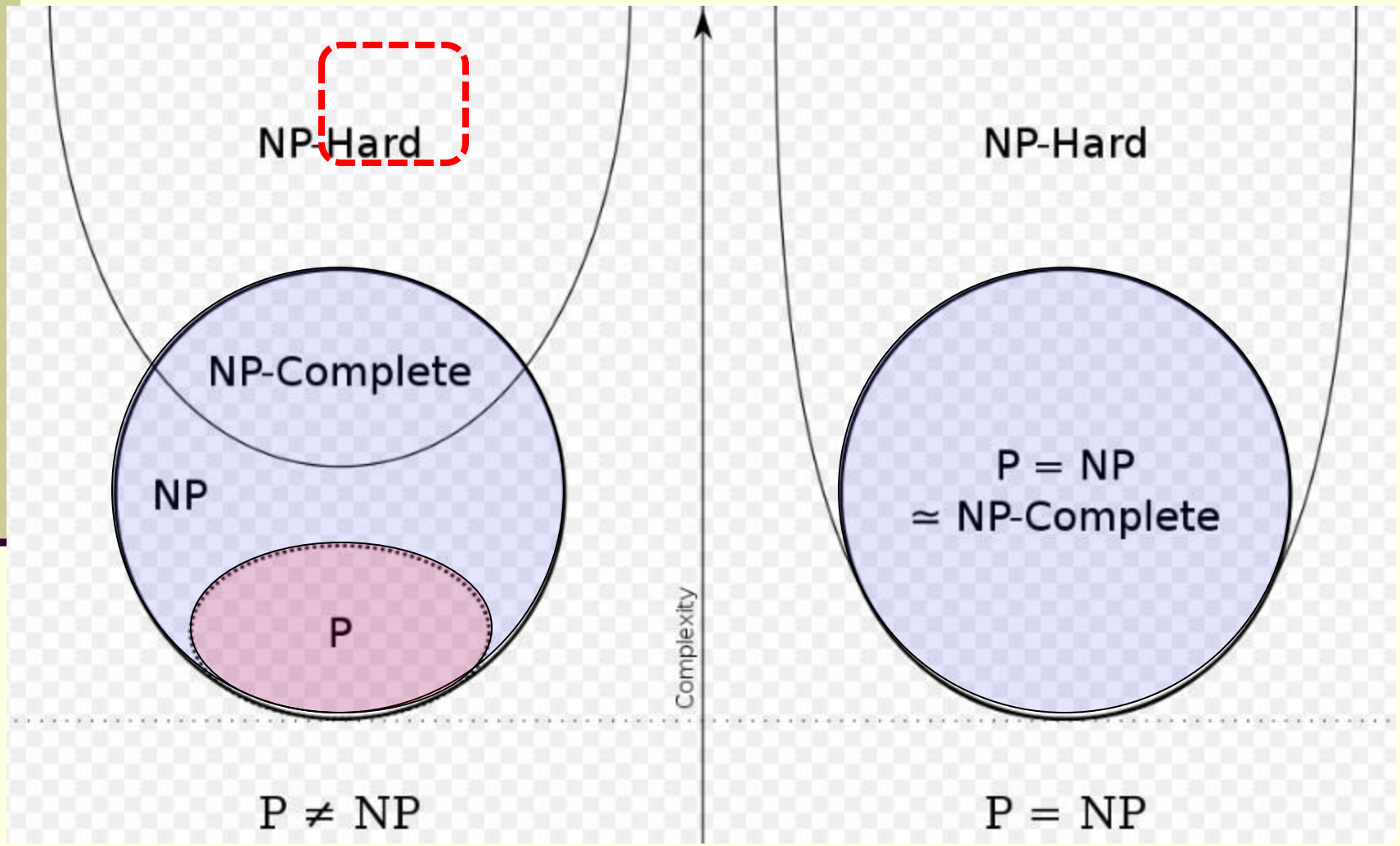
A few **NP problems** not known to be in **P** or to be **NP**-complete:

Can be solved in polynomial time only using a non-deterministic method

- The **graph isomorphism problem** (determine if two finite graphs are isomorphic)
- The **integer factorization problem** (is n a prime?)
- The **discrete logarithm problem**

(given a group G, a generator g of the group and an element h of G, to find **the discrete logarithm** to the base g of h in the group G)

The P versus NP problem:



The P versus NP problem:

- It is widely believed **P ≠ NP**
- a proof either way would have profound implications for mathematics, cryptography, algorithm research, artificial intelligence, game theory, multimedia processing, philosophy, economics and many other fields

The P versus NP problem:

What if **P = NP** ?

then it wouldn’t be worth it to build quantum computers, because standard computers would be fast enough (for all questions), and easier to build.

8. Turing Machines - some more history

The world owes a big thanks to Alan Turing :

without him, we wouldn't be able to enjoy most of the conveniences we have today -

everything that has a CPU built into it.

8. Turing Machines - some more history

So, who is the most important scientist from the last century?

Not someone who built an atomic bomb or someone who discovered an important natural phenomenon,

but the person who changed our life the most – Alan Turing.

Or, look at it this way: nobody would carry an atomic bomb with him/her everyday, but would definitely carry a cell phone and/or a laptop.

Of course the first portable computer was created by the Osborne company in 1981 and the first mobile phone was made by Motorola employee Martin Cooper in 1973. But without a CPU, none of these products would function properly.

And, remember, a CPU was built on the concept/theory of a Turing Machine.

Remember this name

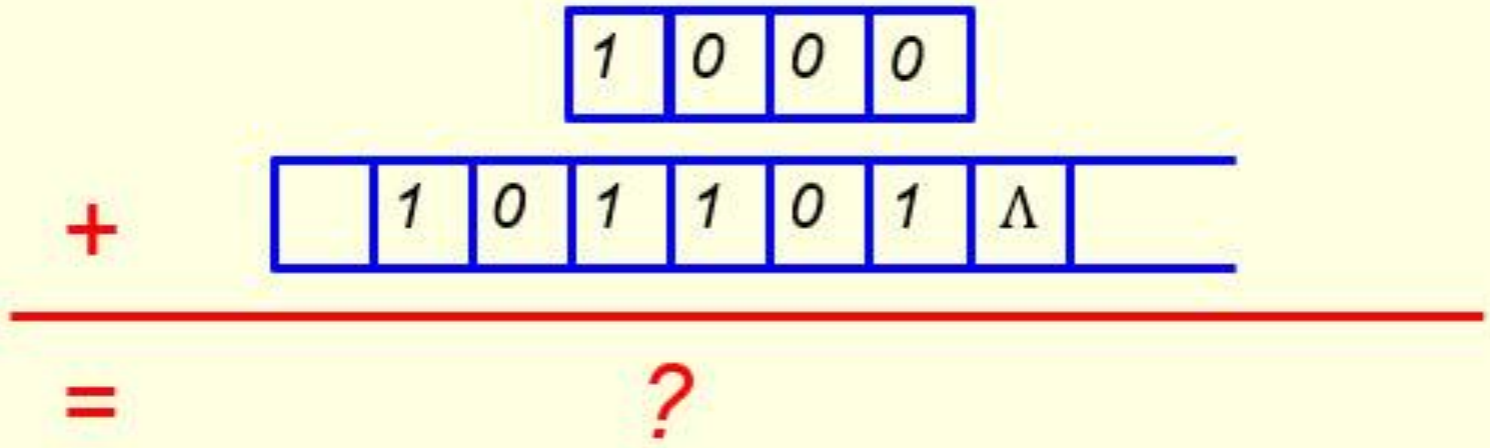
and

Honor this name

8. Turing Machines

Example. $8 + 45 = ?$

$$8 = 1000_2$$
$$45 = 101101_2$$

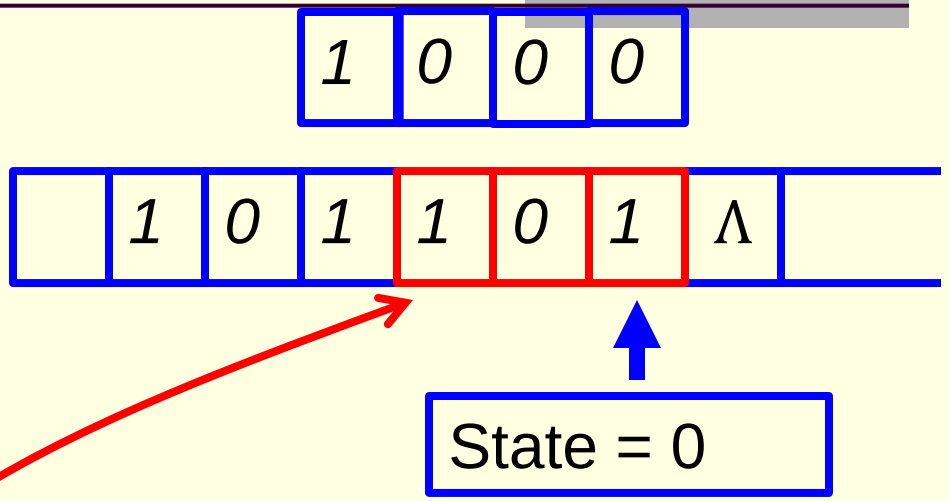


One Solution: use the TM shown in slides 87-97 twice.

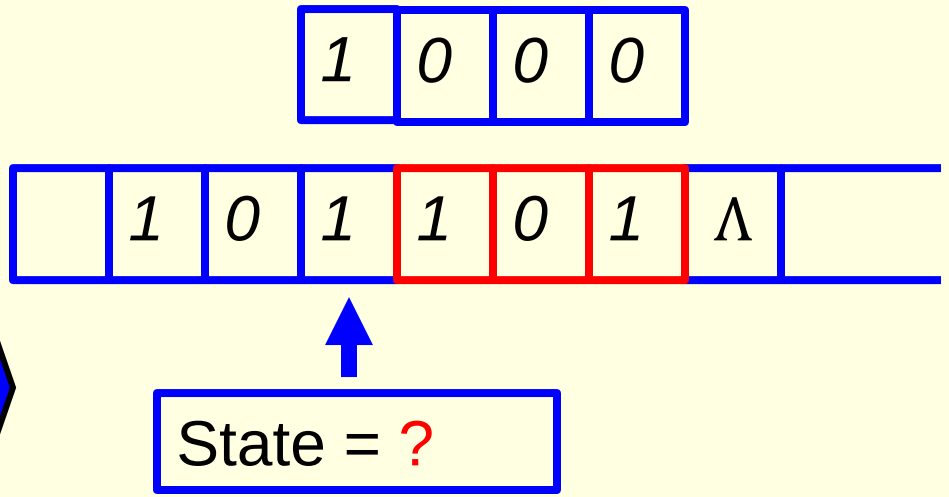
Second Solution: use a direct approach.

Direct approach - 14 instructions + 6 states

Initially,
(start state = 0)



These three digits are of no concern to us, can move immediately to the fourth digit



Direct approach - 14 instructions + 6 states

Move three cells left

(0, 0, 0, L, 1)
(0, 1, 1, L, 1)
(1, 0, 0, L, 2)
(1, 1, 1, L, 2)
(1, Λ, 0, L, 2)
(2, 0, 0, L, 3)
(2, 1, 1, L, 3)
(2, Λ, 0, L, 3)

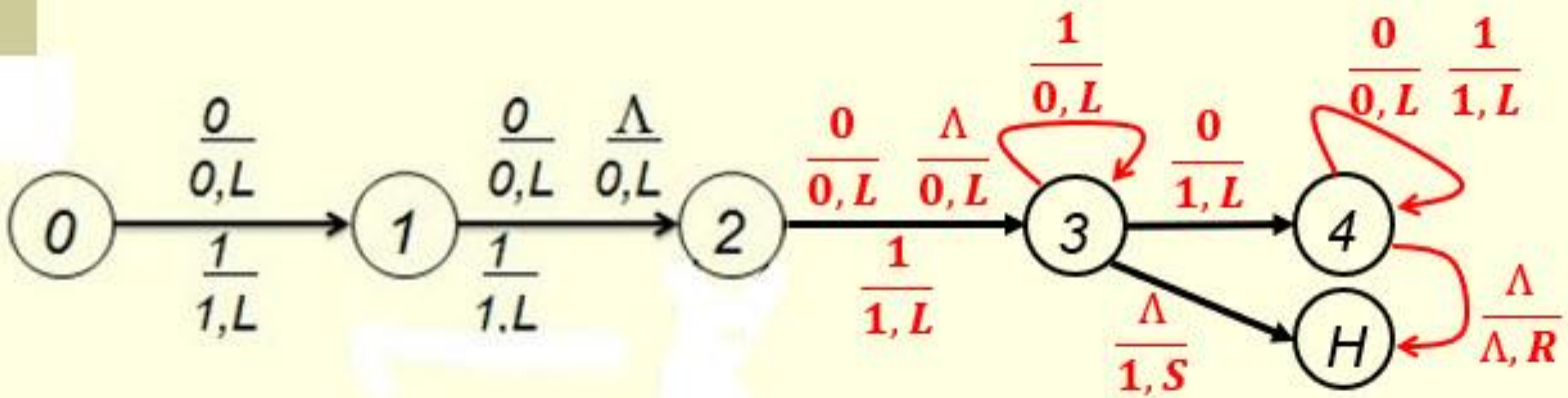
Add 1:

(3, 0, 1, L, 4) Move left
(3, 1, 0, L, 3) Carry
(3, Λ, 1, S, halt) Done

Find left end of the string:

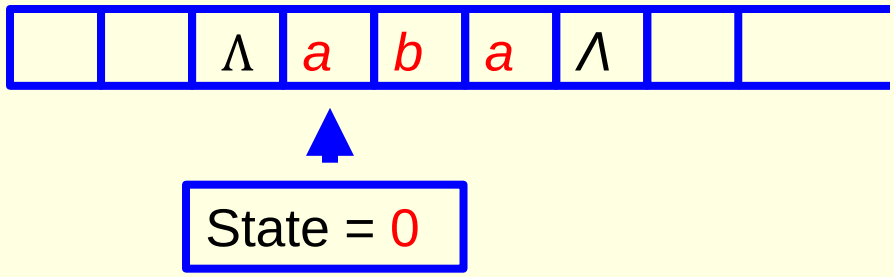
(4, 0, 0, L, 4)
(4, 1, 1, L, 4)
(4, Λ, Λ, R, halt) Done

Direct approach - 14 instructions + 6 states

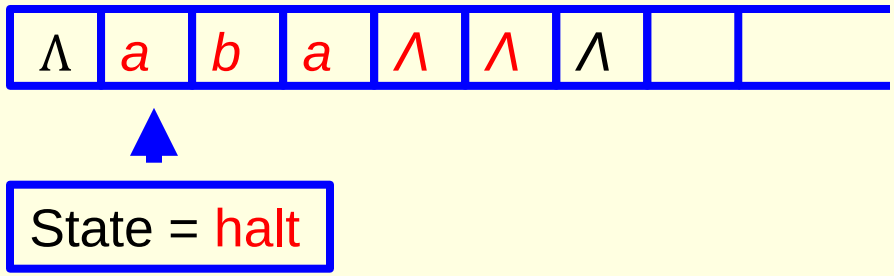


Example. *Find a TM to move any string over {a, b} to the left two cells position. Assume the tape head ends at the left end of any nonempty output string.*

Initially,



At the end,



One Solution: use the TM shown in slides 44-51 twice.

Second Solution: use a direct approach (move one letter a time).

Third Solution: use a stack (move two letters each time).

Direct Approach – 14 instructions + 11 states

Find a or b to move:

(0, a, Λ, L, 11)	found a
(0, b, Λ, L, 21)	found b
(0, Λ, Λ, L, 41)	no more a's or b's

Move to left end of output:

	skip Λ
	skip Λ
(5, a, a, L, 5)	skip a
(5, b, b, L, 5)	skip b
(5, Λ, Λ, R, halt)	Done

Write a or b:

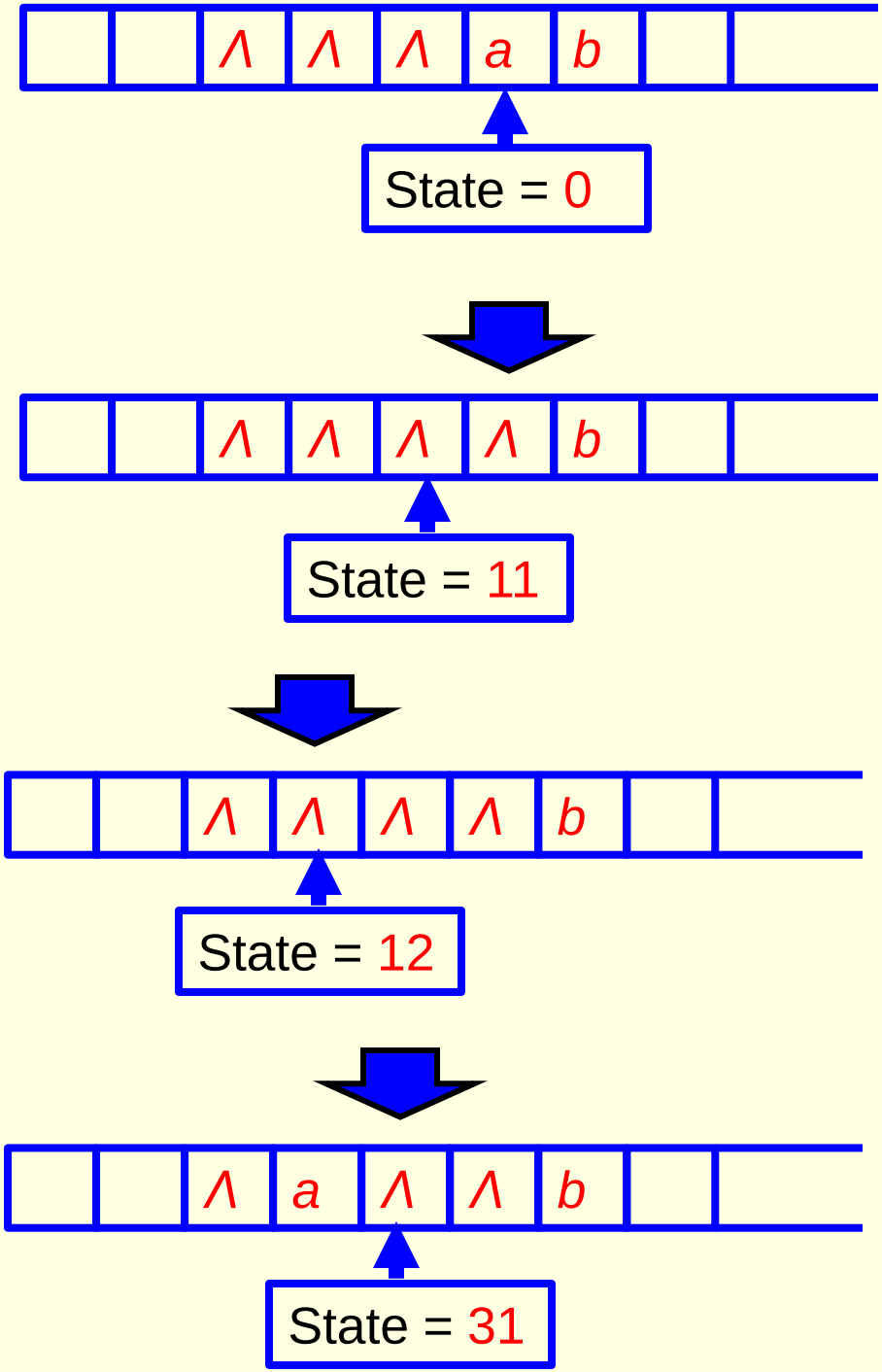
(11, Λ, Λ, L, 12)	skip Λ
	write a
(21, Λ, Λ, L, 22)	skip Λ
	write b
(31, Λ, Λ, R, 32)	skip Λ
	skip Λ

Find a or b to move:

(0, a, Λ, L, 11)	found a
(0, b, Λ, L, 21)	found b
(0, Λ, Λ, L, 41)	no more a's or b's

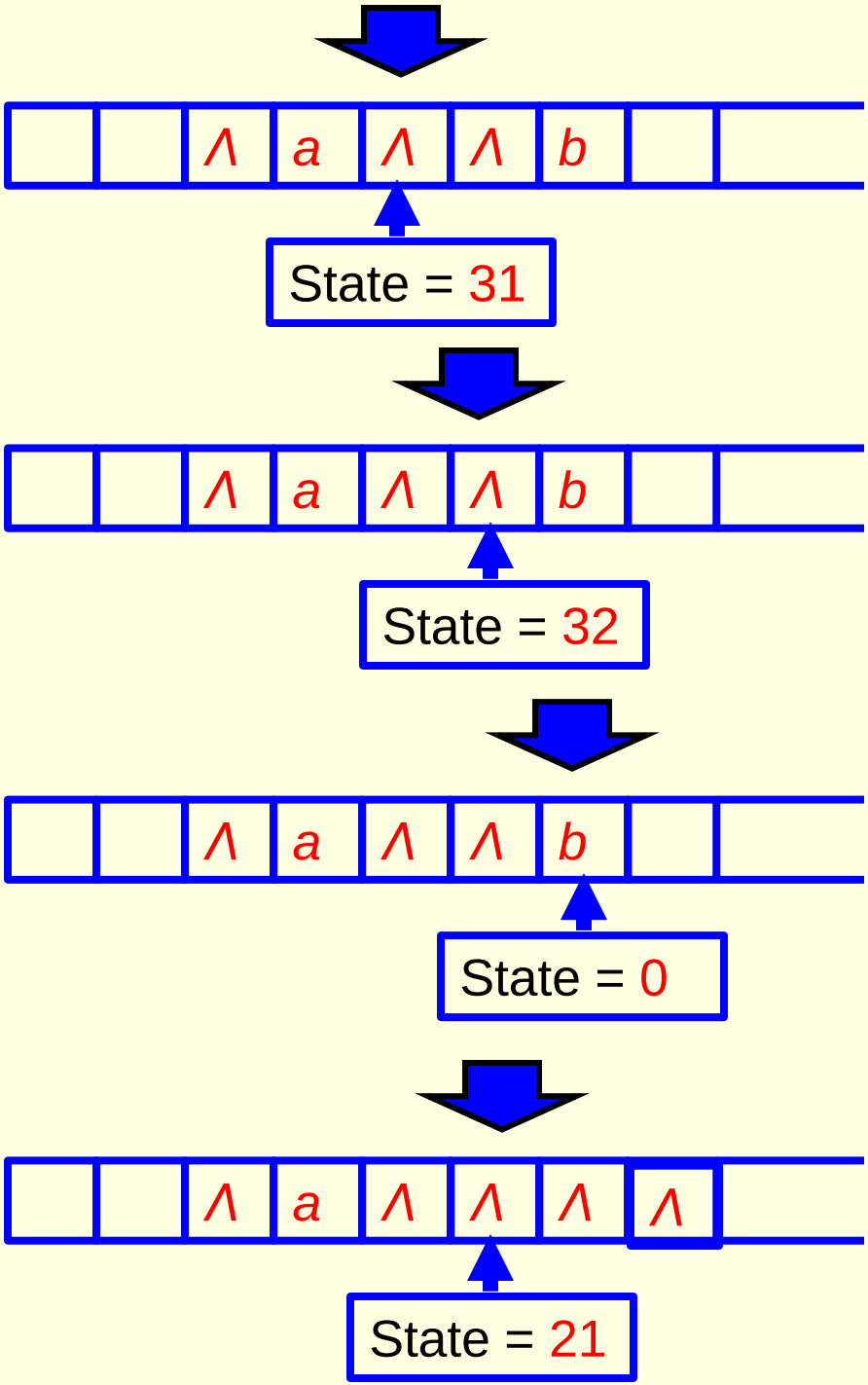
Write a or b:

(11, Λ, Λ, L, 12)	skip Λ
(12, Λ, a, R, 31)	write a
(21, Λ, Λ, L, 22)	skip Λ
(22, Λ, b, R, 31)	write b
(31, Λ, Λ, R, 32)	skip Λ
(32, Λ, Λ, R, 0)	skip Λ



Find a or b to move:	
(0, a, Λ, L, 11)	found a
(0, b, Λ, L, 21)	found b
(0, Λ, Λ, L, 41)	no more a's or b's

Write a or b:	
(11, Λ, Λ, L, 12)	skip Λ
(12, Λ, a, R, 31)	write a
(21, Λ, Λ, L, 22)	skip Λ
(22, Λ, b, R, 31)	write b
(31, Λ, Λ, R, 32)	skip Λ
(32, Λ, Λ, R, 0)	skip Λ

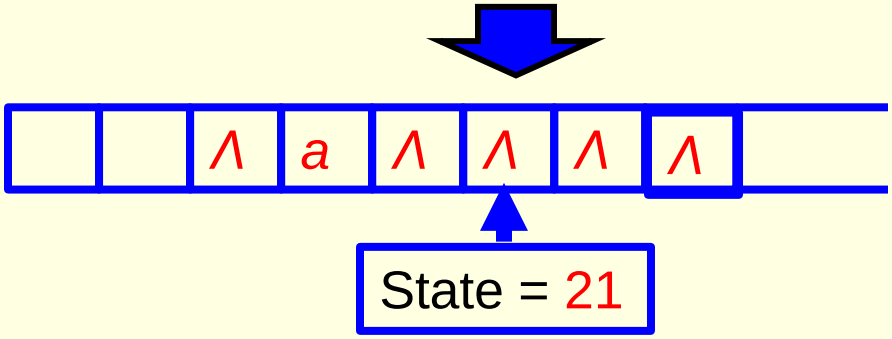


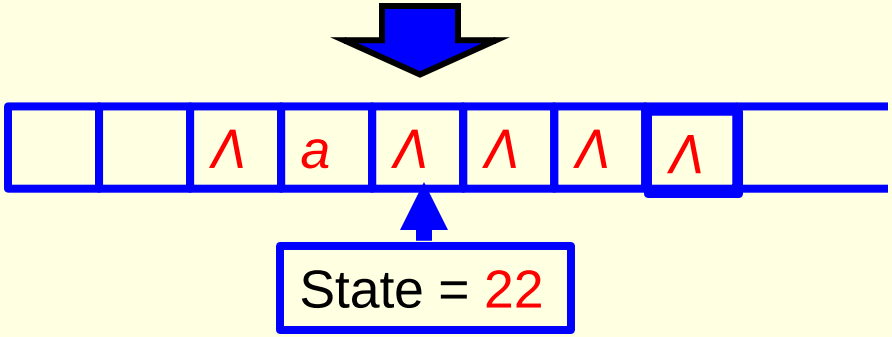
Find a or b to move:

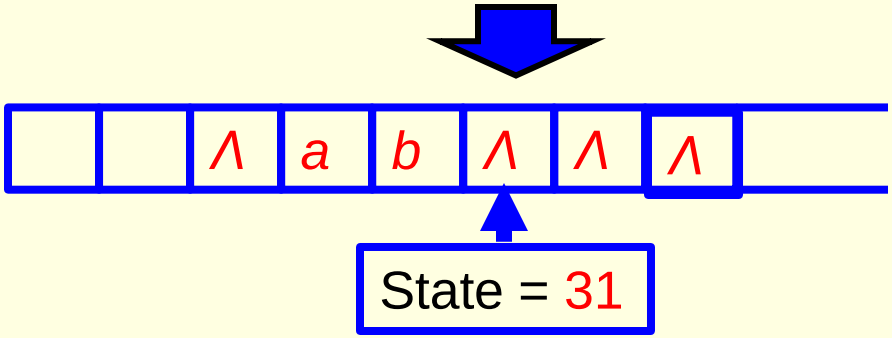
(0, a, Λ, L, 11)	found a
(0, b, Λ, L, 21)	found b
(0, Λ, Λ, L, 41)	no more a's or b's

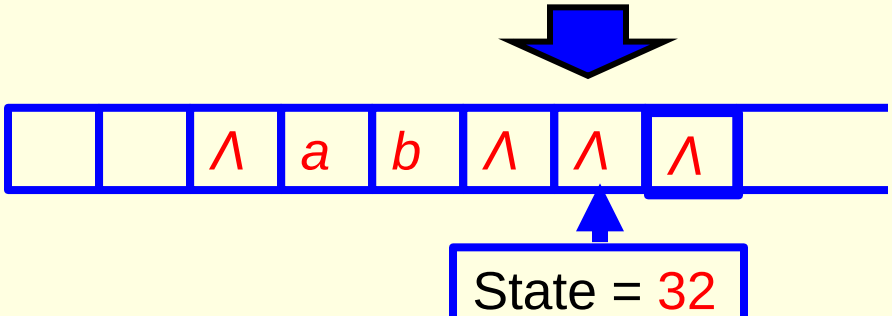
Write a or b:

(11, Λ, Λ, L, 12)	skip Λ
(12, Λ, a, R, 31)	write a
(21, Λ, Λ, L, 22)	skip Λ
(22, Λ, b, R, 31)	write b
(31, Λ, Λ, R, 32)	skip Λ
(32, Λ, Λ, R, 0)	skip Λ







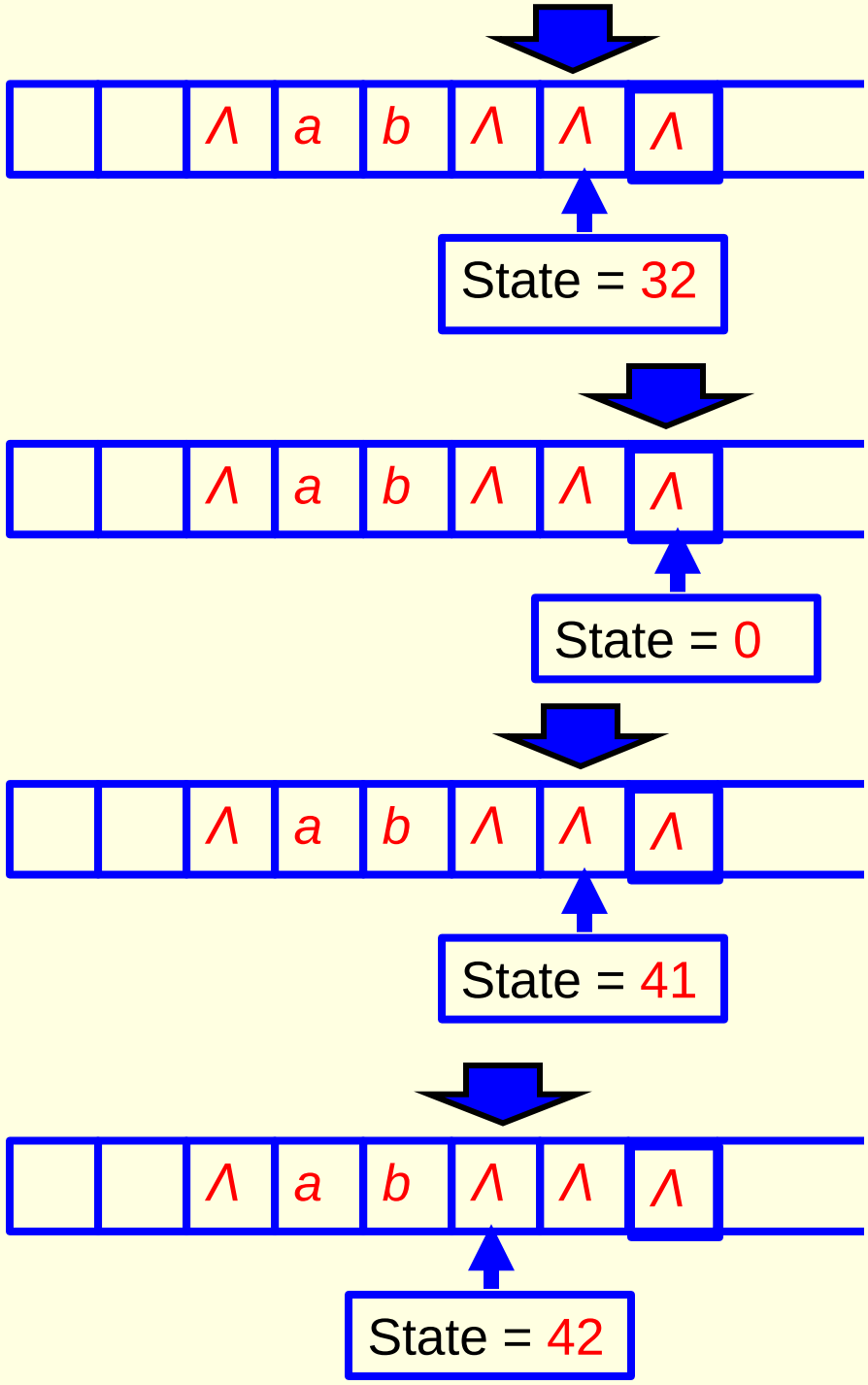


Find a or b to move:

(0, a, Λ, L, 11)	found a
(0, b, Λ, L, 21)	found b
(0, Λ, Λ, L, 41)	no more a's or b's

Move to left end of output:

(41, Λ, Λ, L, 42)	skip Λ
(42, Λ, Λ, L, 5)	skip Λ
(5, a, a, L, 5)	skip a
(5, b, b, L, 5)	skip b
(5, Λ, Λ, R, halt)	Done

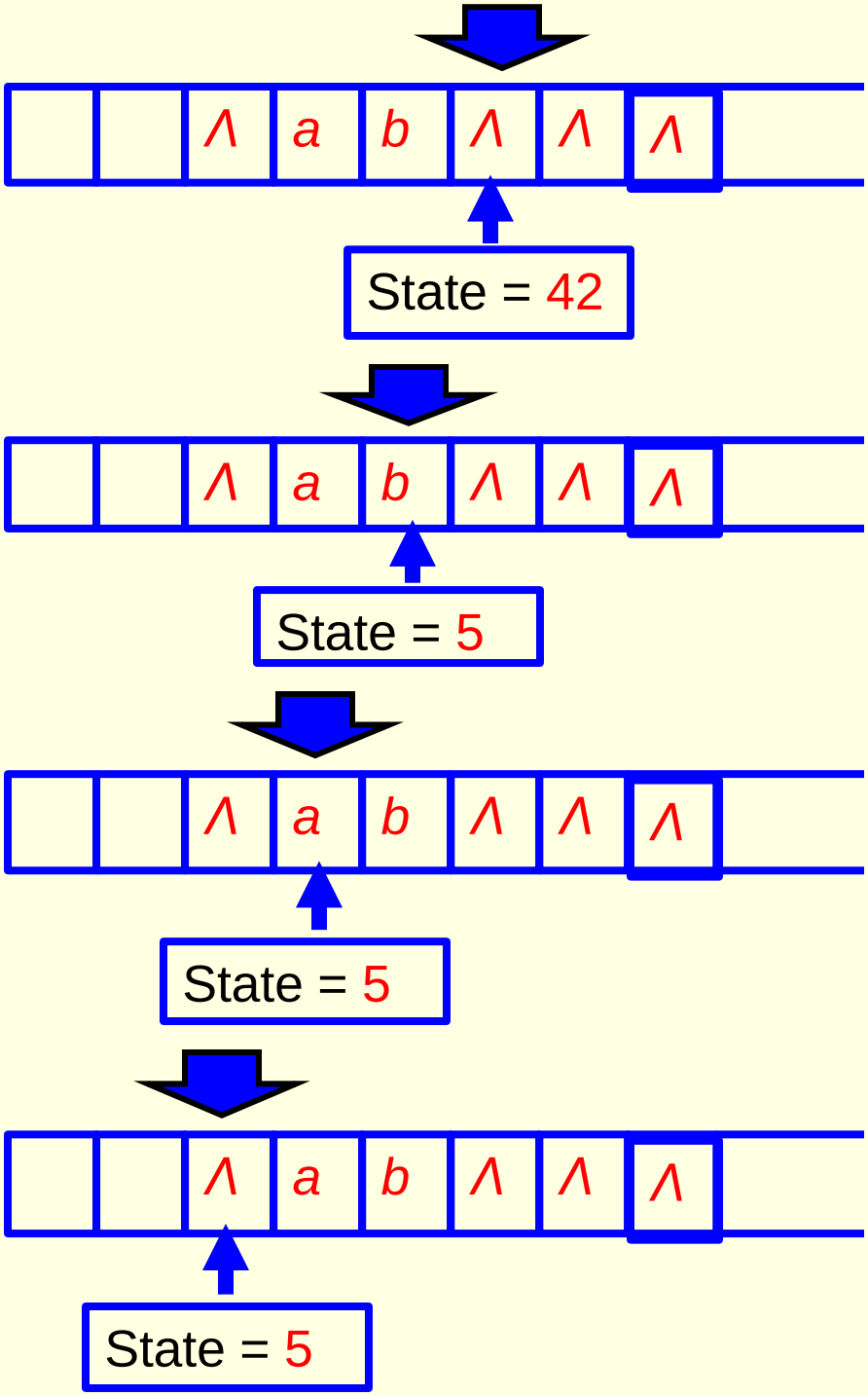


Find a or b to move:

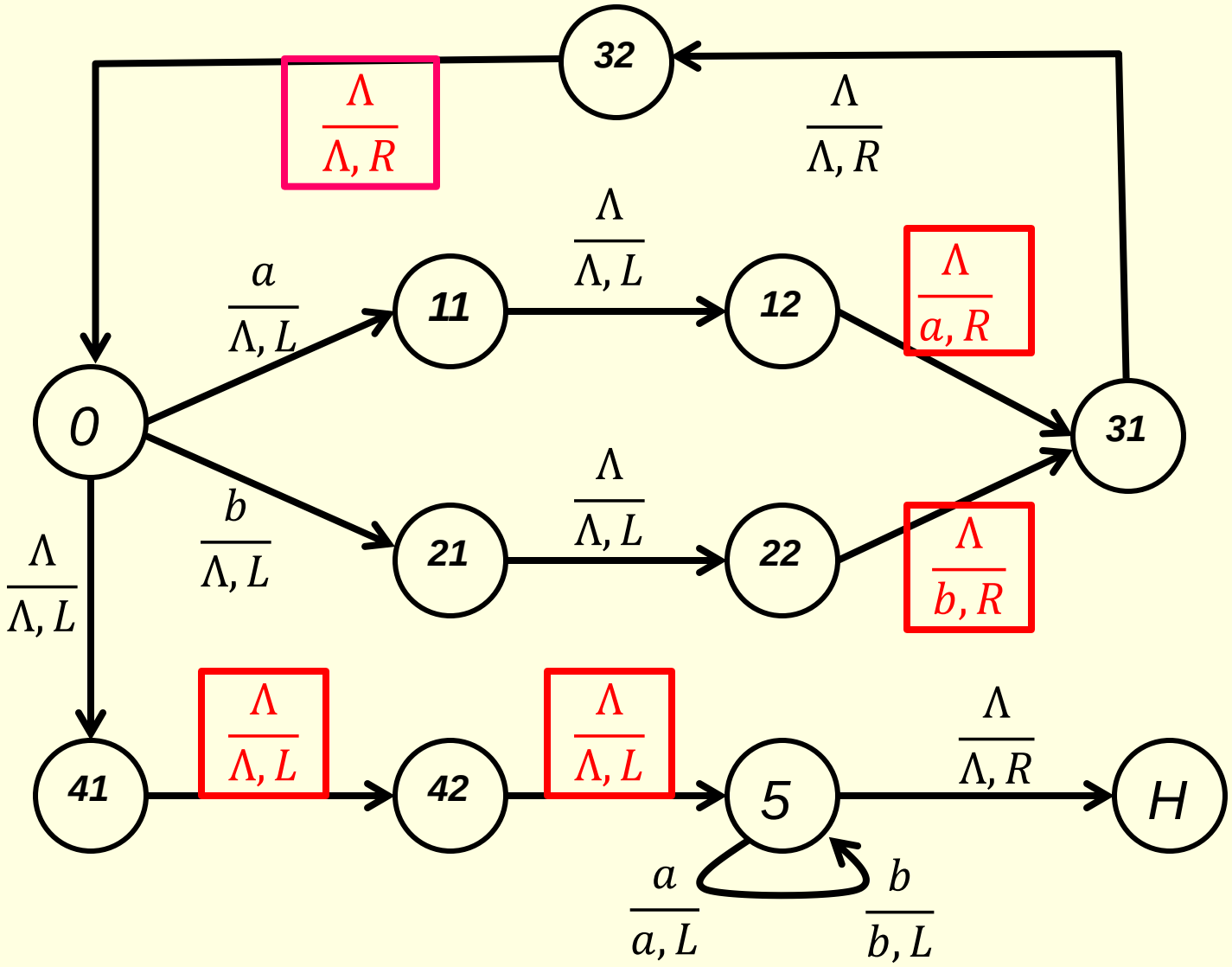
(0, a, Λ, L, 11)	found a
(0, b, Λ, L, 21)	found b
(0, Λ, Λ, L, 41)	no more a's or b's

Move to left end of output:

(41, Λ, Λ, L, 42)	skip Λ
(42, Λ, Λ, L, 5)	skip Λ
(5, a, a, L, 5)	skip a
(5, b, b, L, 5)	skip b
(5, Λ, Λ, R, halt)	Done

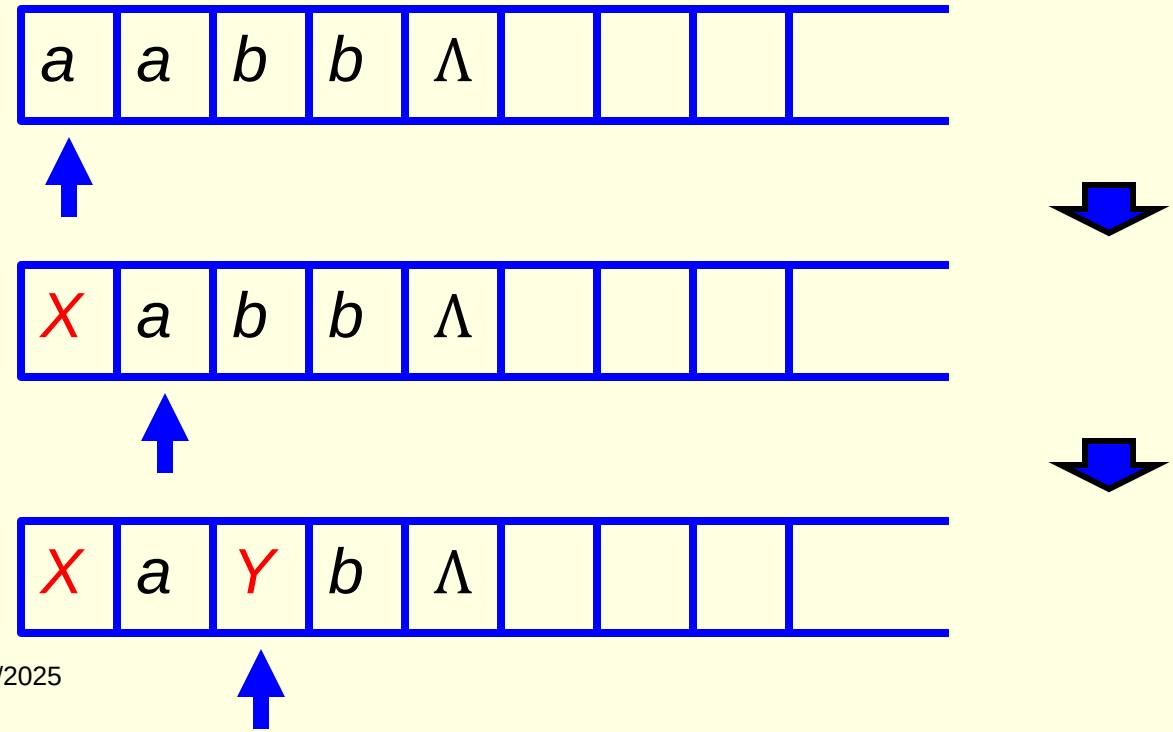


Direct Approach – 14 instructions + 11 states



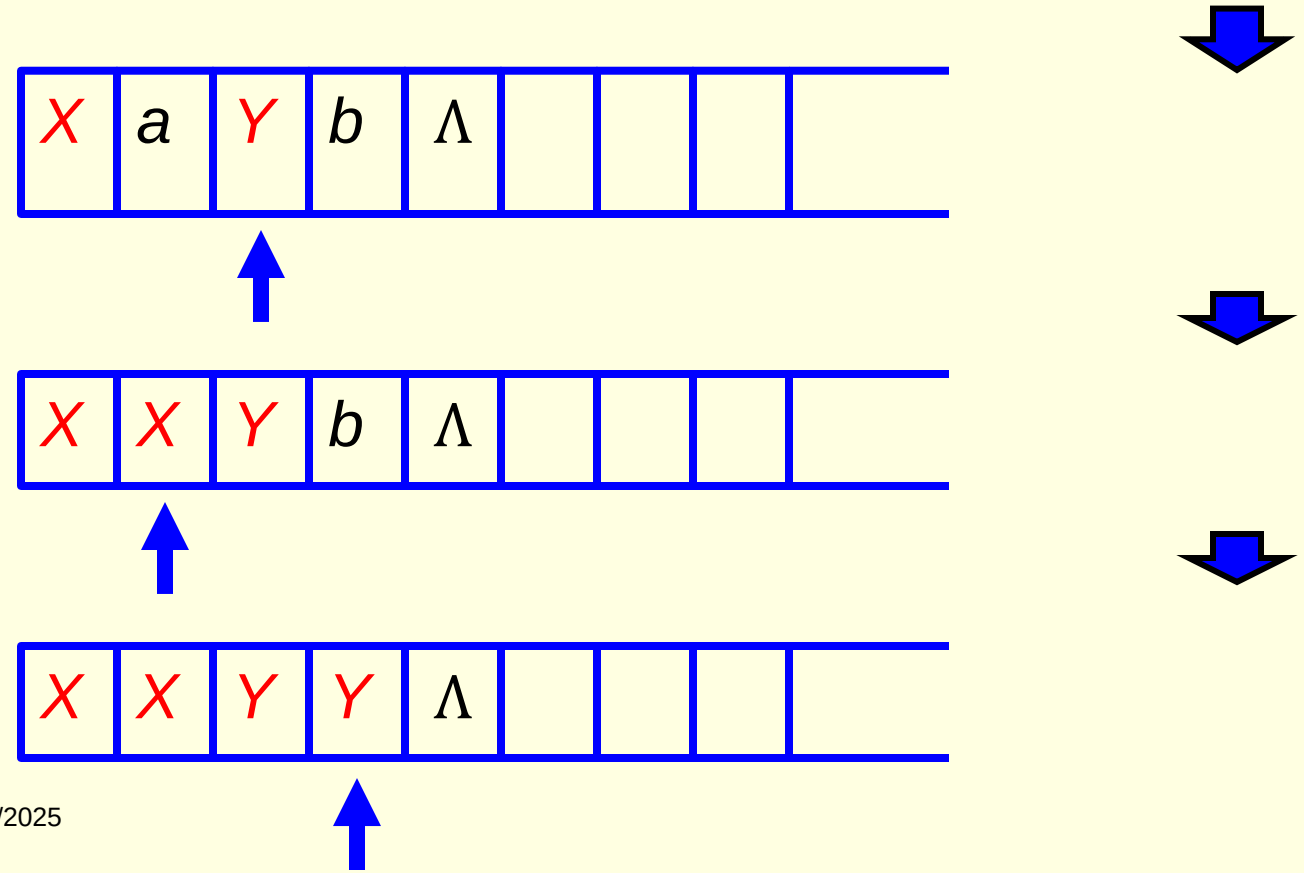
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

How the TM that accepts $\{ a^n b^n \mid n \in \mathbf{N} \}$ works?



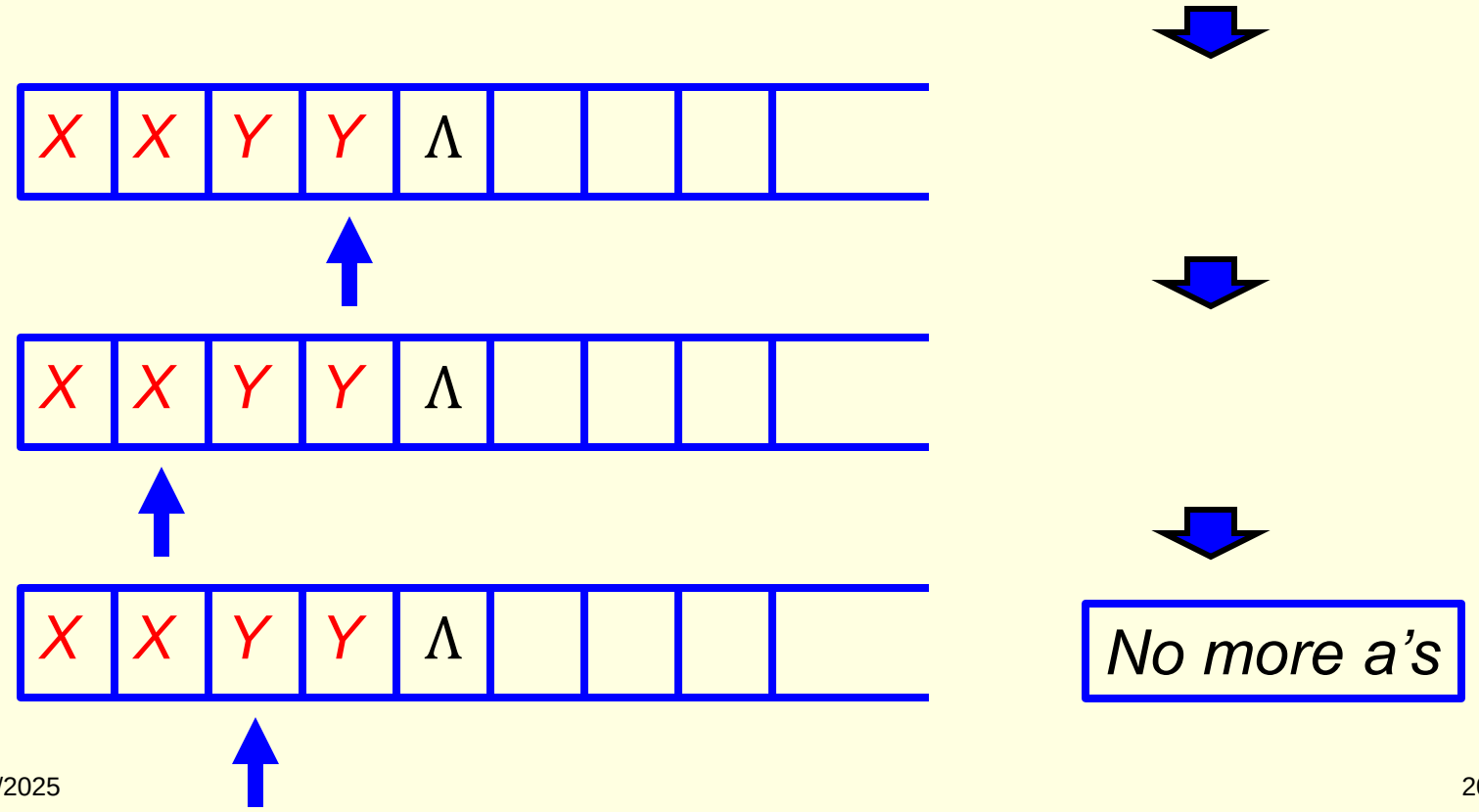
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

How the TM that accepts $\{ a^n b^n \mid n \in \mathbf{N} \}$ works?
(conti)



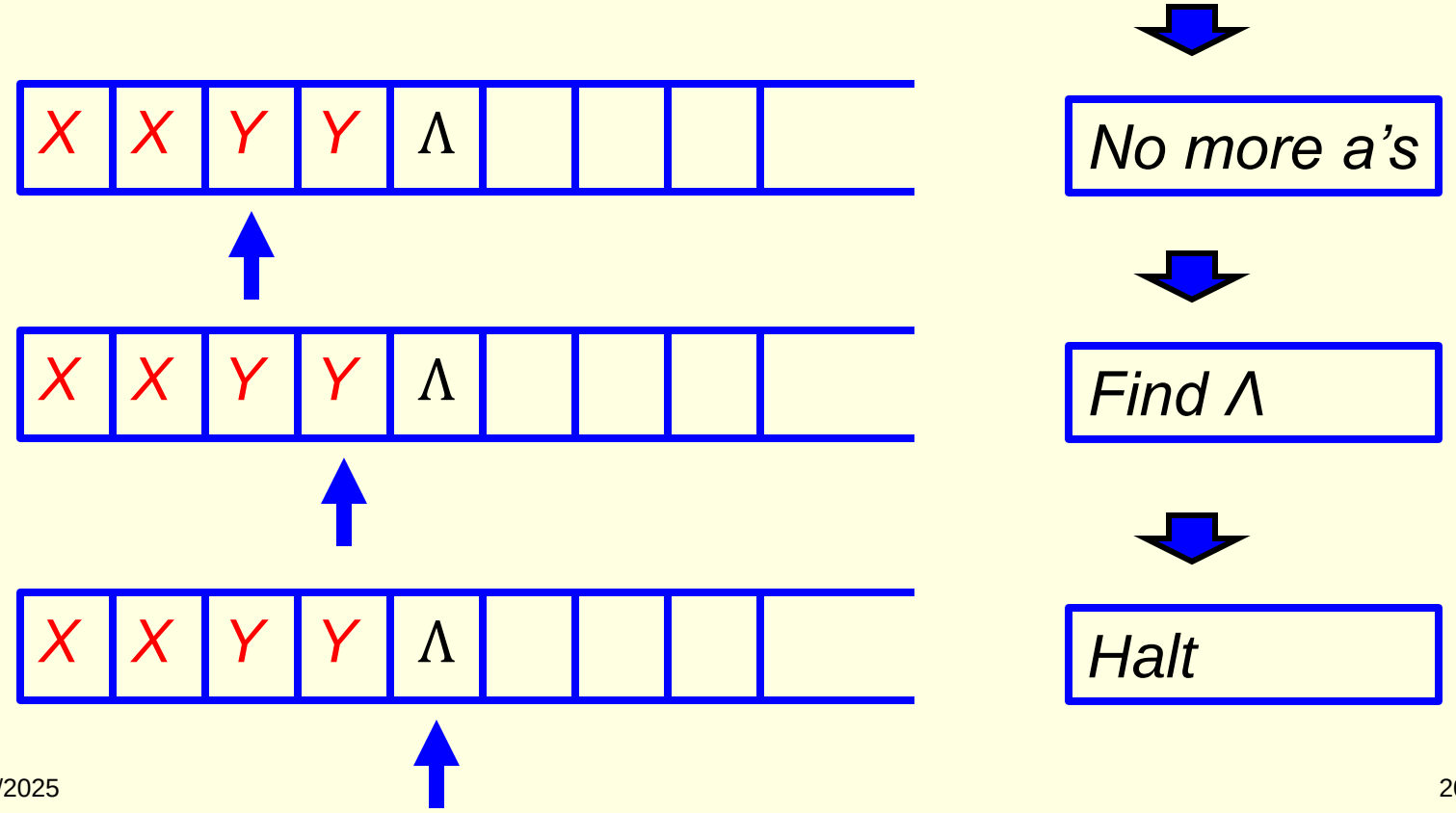
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

How the TM that accepts $\{ a^n b^n \mid n \in \mathbf{N} \}$ works?
(conti)



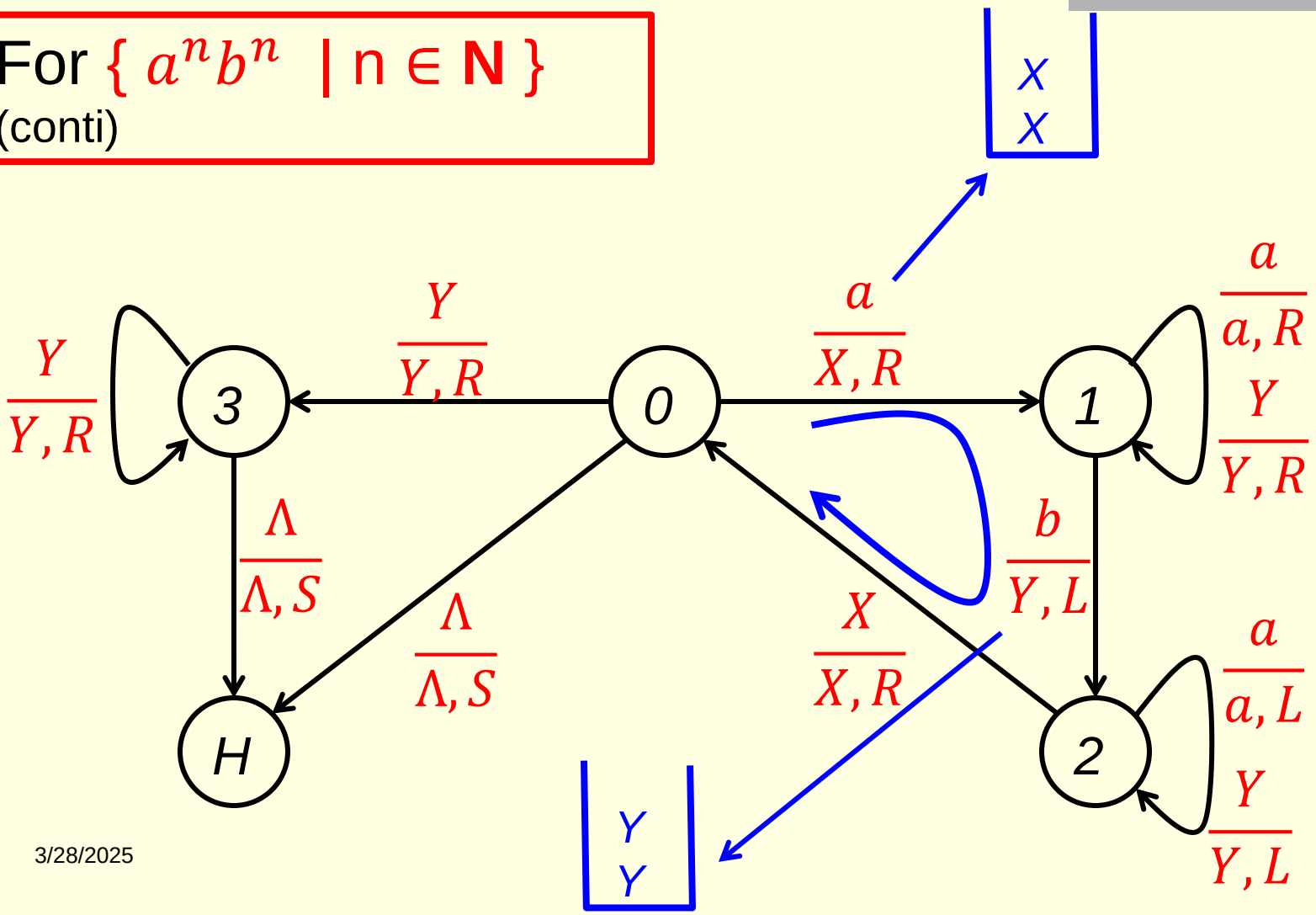
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

How the TM that accepts $\{ a^n b^n \mid n \in \mathbf{N} \}$ works?
(conti)



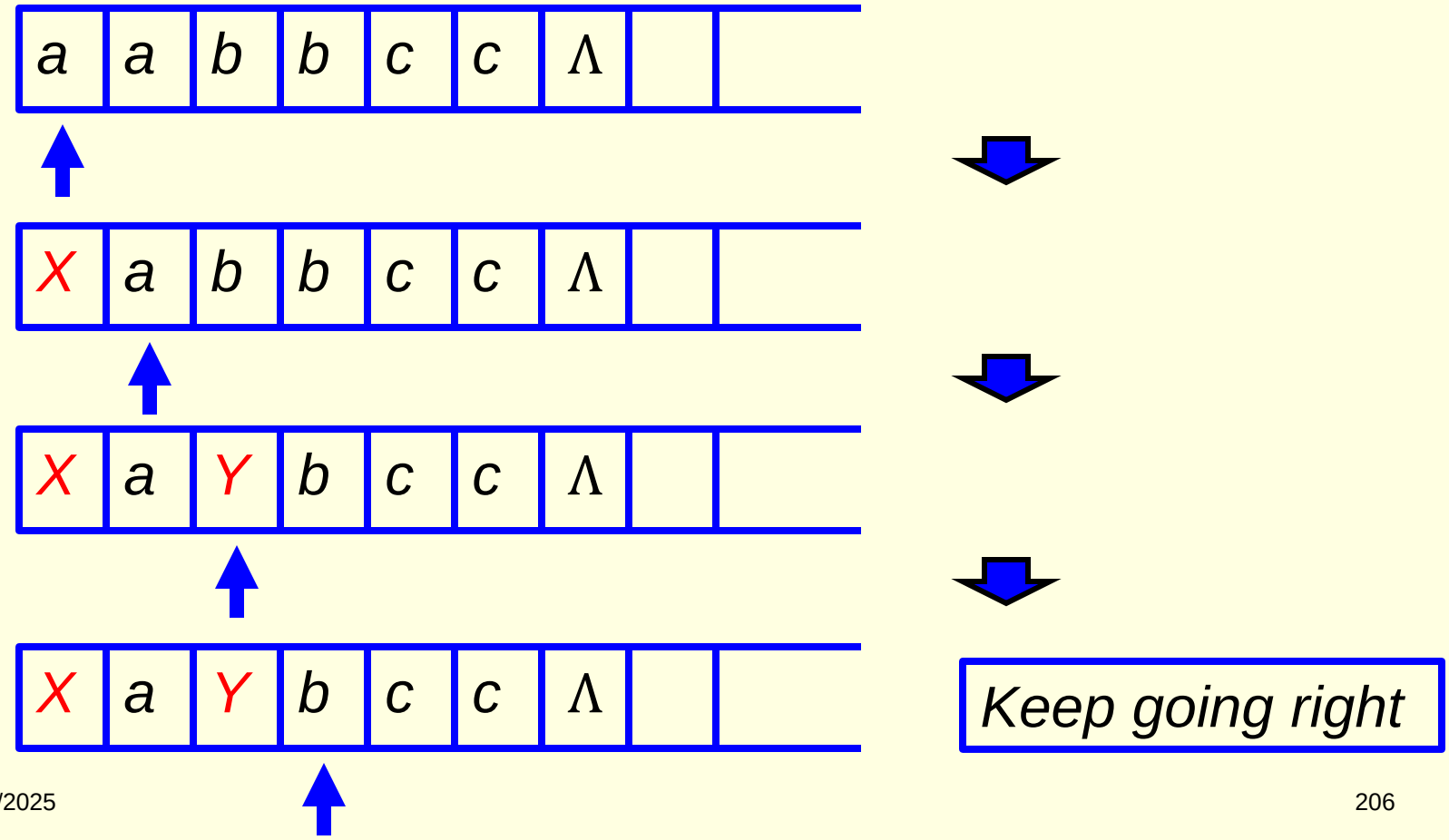
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

For $\{ a^n b^n \mid n \in \mathbf{N} \}$
(conti)



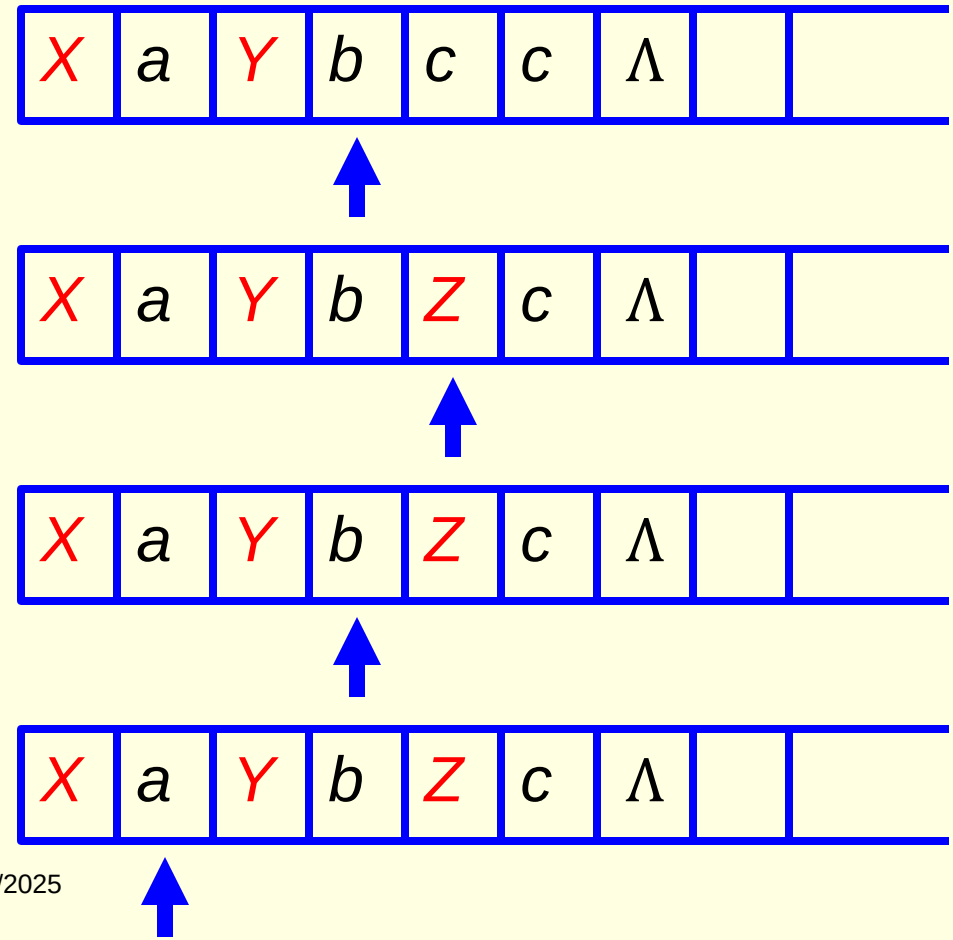
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

Basic idea:



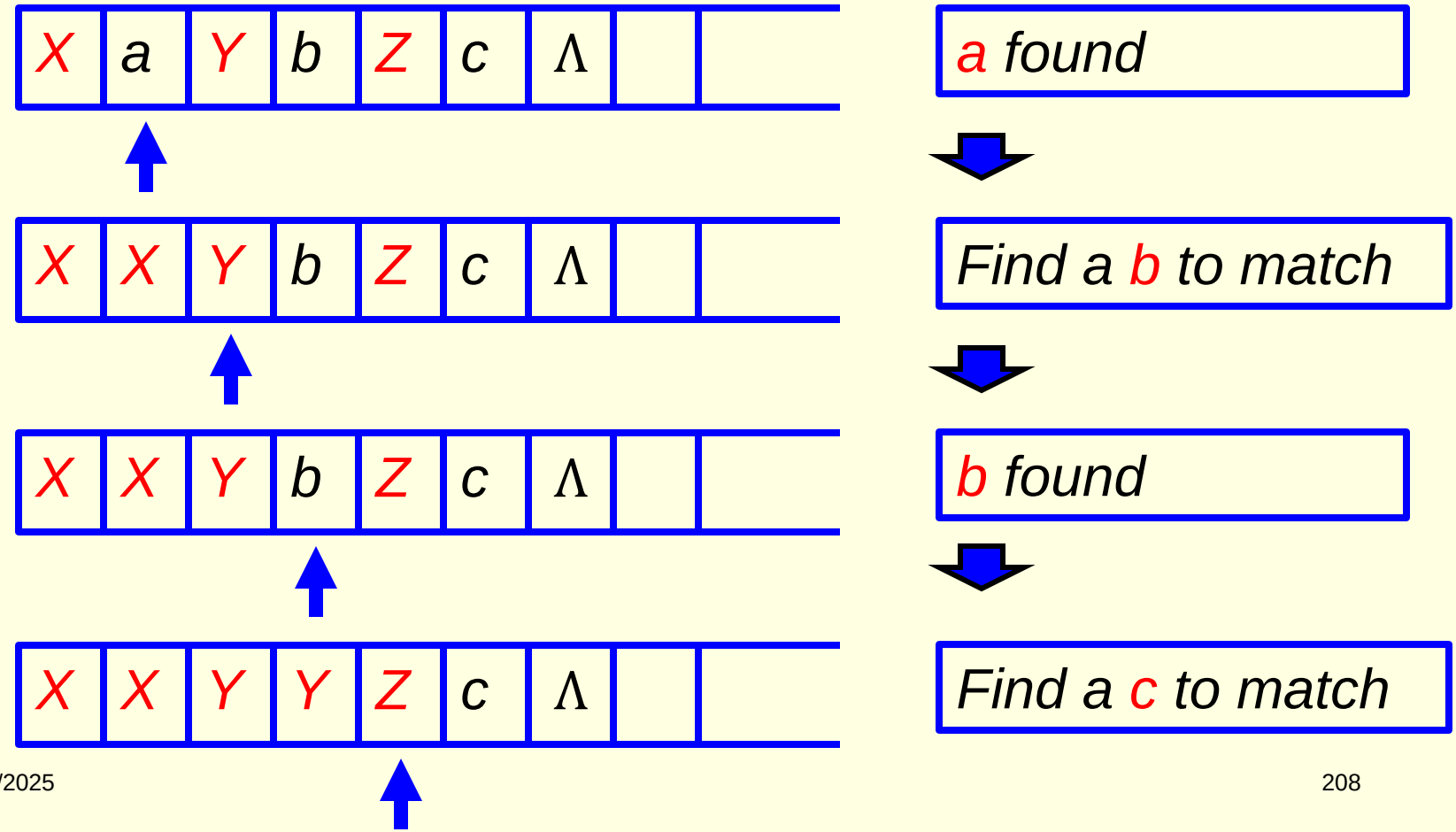
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

Basic idea (conti):



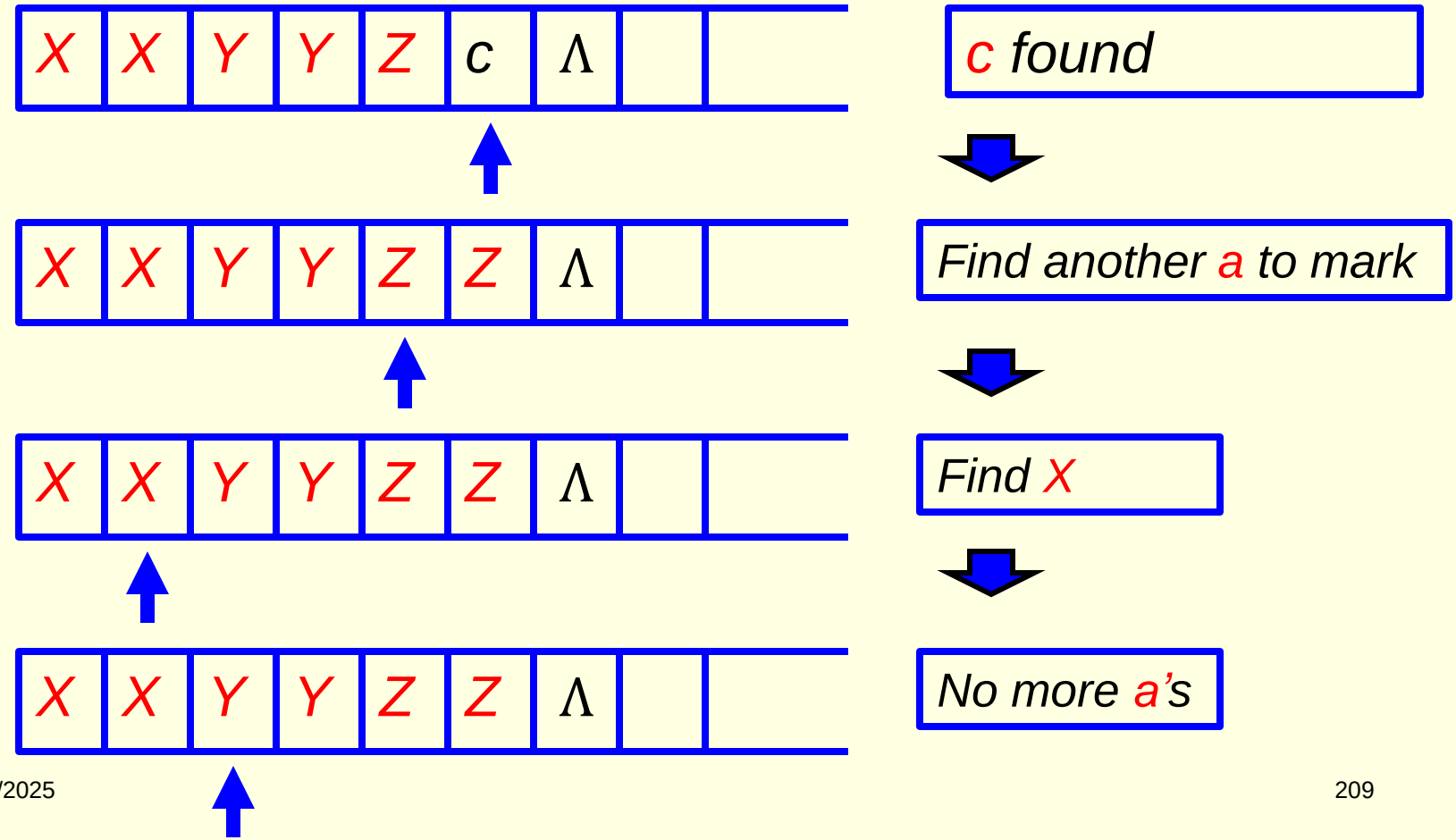
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

Basic idea (conti):



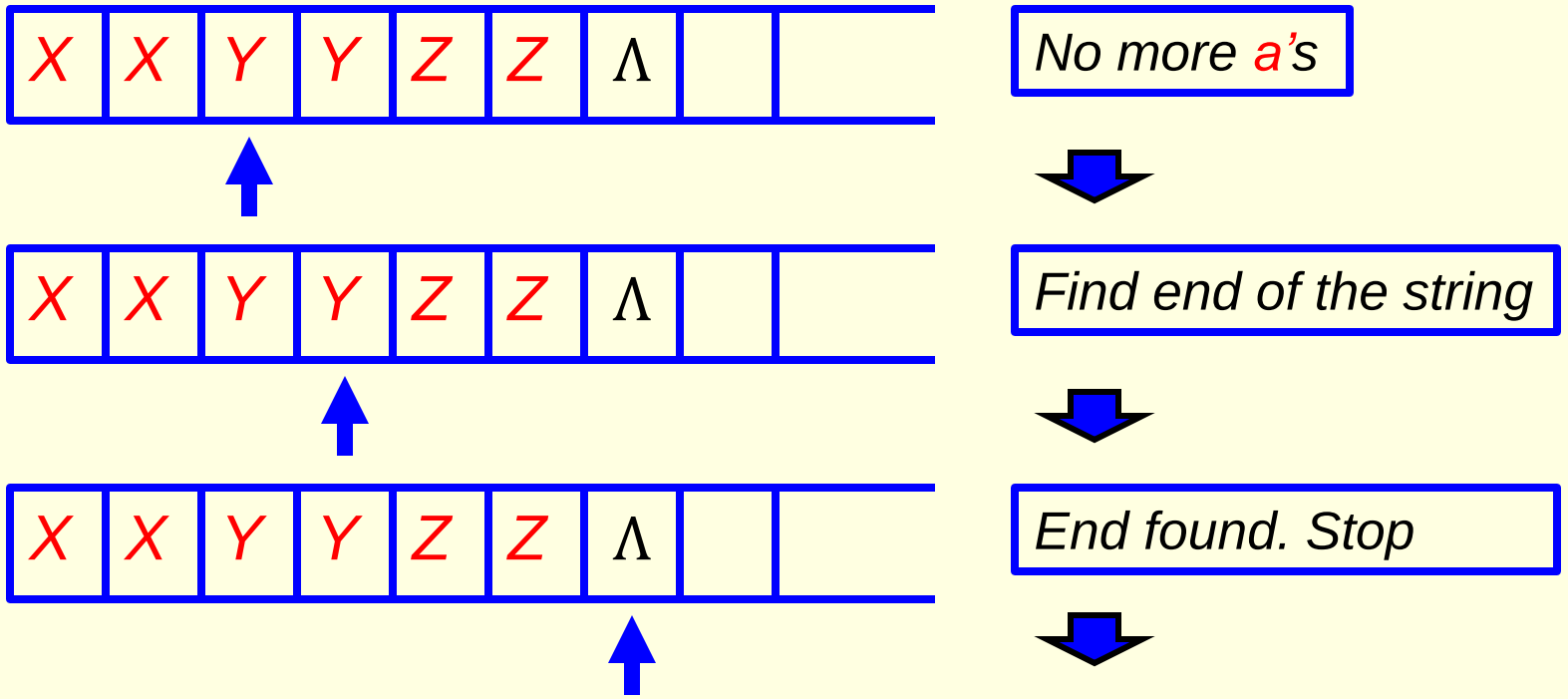
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

Basic idea (conti):



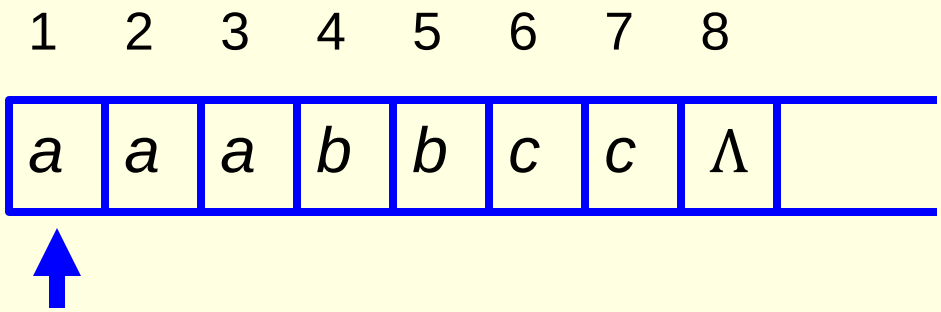
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

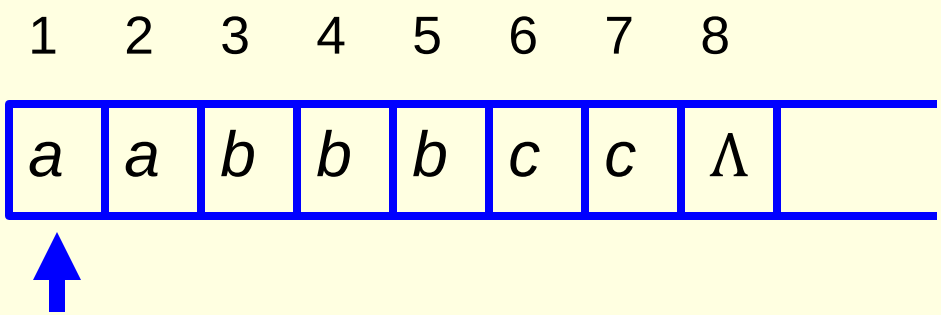
Basic idea (conti):

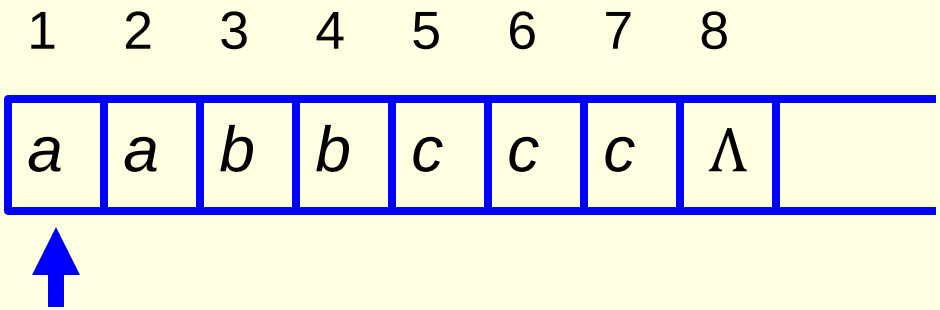


Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

Strings not accepted by this TM:

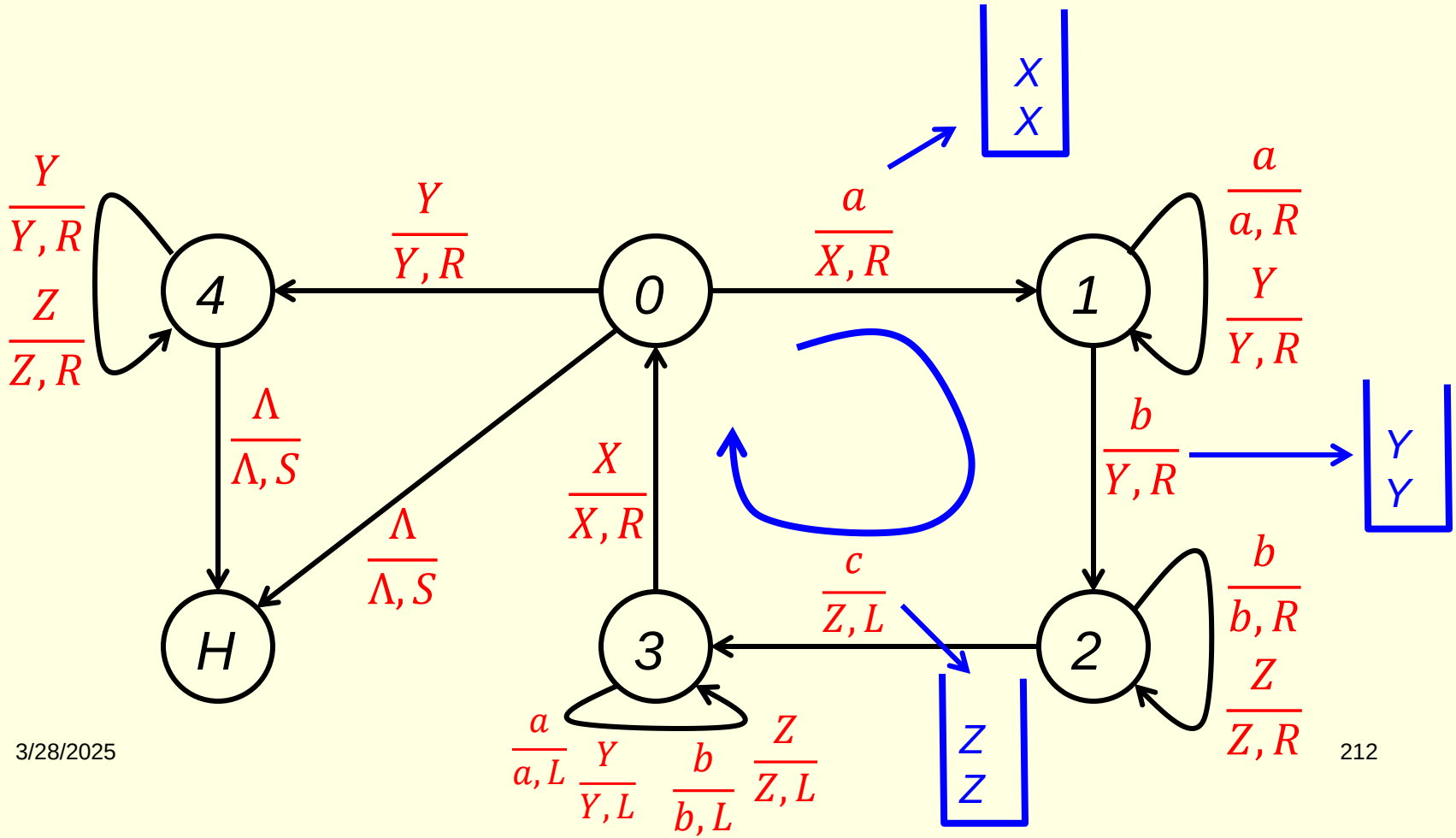






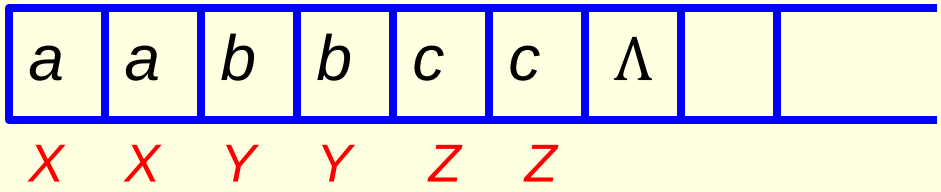
Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

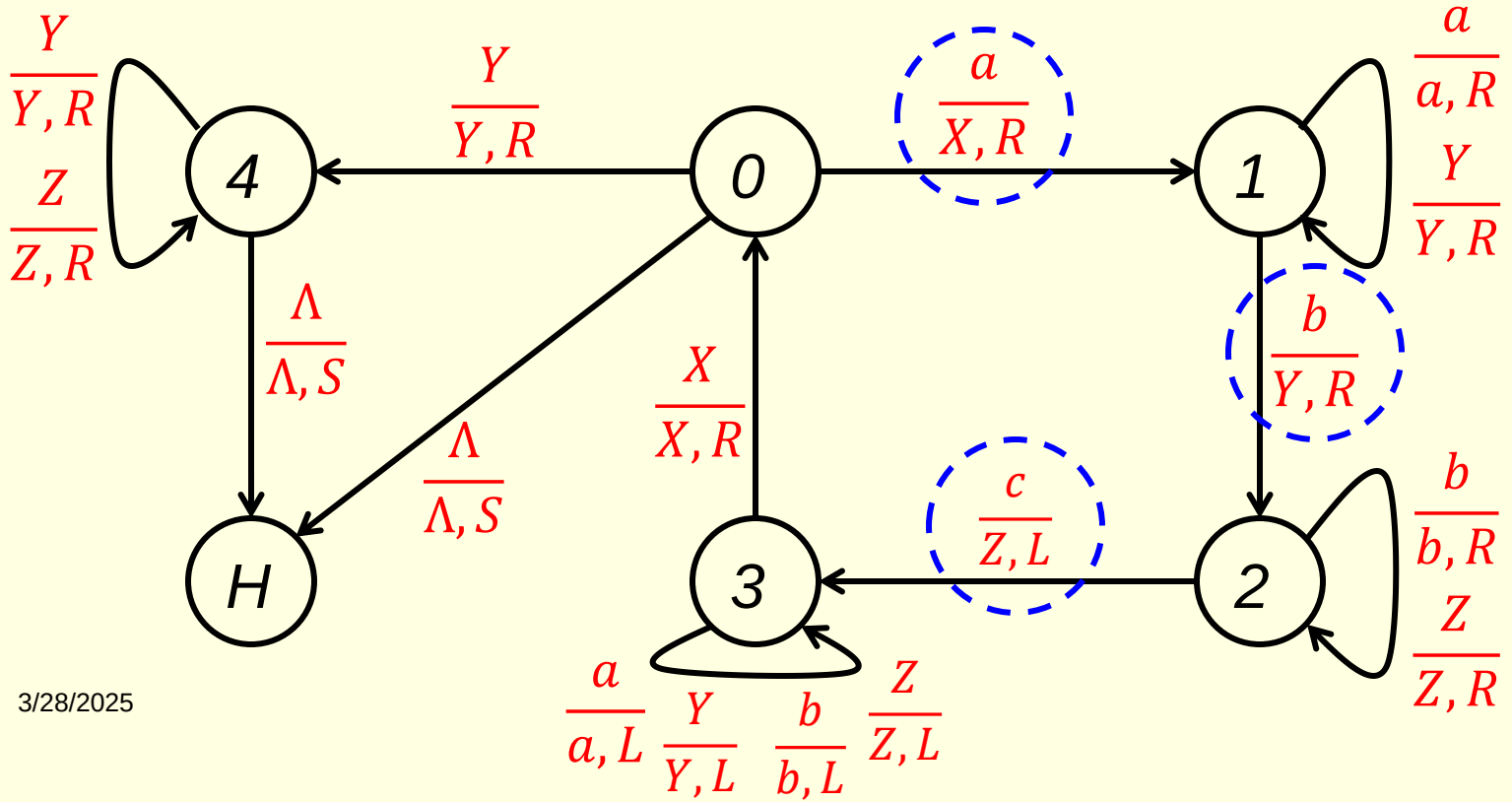
For $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$



Example. Find a TM to accept $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$.

For $\{ a^n b^n c^n \mid n \in \mathbf{N} \}$





Skip slides 214-231

End of Turing Machines I