San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu
www.cs.sjsu.edu/~yazdankhah

# Pushdown Automata

# (Part 1)

## Lecture 13

## Day 14/31

## CS 154

## Formal Languages and Computability

## Spring 2018

# Agenda of Day 14

- Solution and Feedback of Quiz 4
- Solution and Feedback of HW 3
- Summary of Lecture 12
- Lecture 13: Teaching ...
  - Pushdown Automata (part 1)

# Solution and Feedback of Quiz 4 (Out of 20)

| Metrics | Section 1 | Section 2 | Section 3 |
|---|---|---|---|
| Average | 14 | 15 | 15 |
| High Score | 19 | 20 | 19 |
| Low Score | 12 | 8 | 11 |

# Solution and Feedback of HW 3 (Out of 40)

| Metrics | Section 1 | Section 2 | Section 3 |
|---|---|---|---|
| Average | 39.4 | 38.1 | 35 |
| High Score | 40 | 40 | 40 |
| Low Score | 31 | 30 | 0 |

# Summary of Lecture 12: We learned ...

## Regular Languages

- For $a^n b^n$ we could NOT construct a DAF/NFA. Why?
- Because DFAs do NOT have memory (or counter) to store the number of a's ...
- So, we realized that languages are different and need to be categorized.
- We categorized the languages as ...
  - ... regular and non-regular.
- A language is called regular iff ...
  - ... there exists a DFA/NFA to recognize it.

- We learned how to heuristically figure out whether a language is regular or not.

## Finite Languages

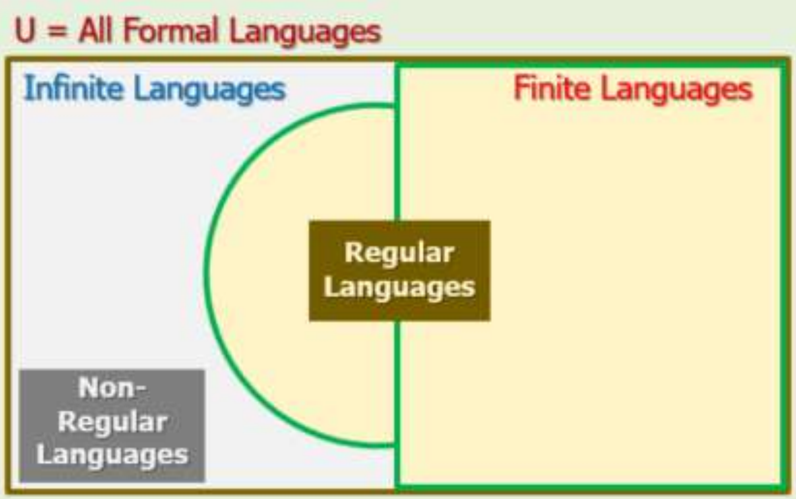
- We proved the theorem ...
  - All finite languages are regular.
- Formally speaking ...
  - If L is finite, then L is regular.
- The contrapositive of this theorem ...
  - If L is non-regular, then L is infinite.

**Any question?**

# Summary of Lecture 12: We learned ...

## Languages Categorization

- We categorized formal languages as:
- Finite and Infinite
- Regular and Non-Regular




- We need to construct more powerful machines that accepts non-regular languages.

**Any Question?**

# Pushdown Automata

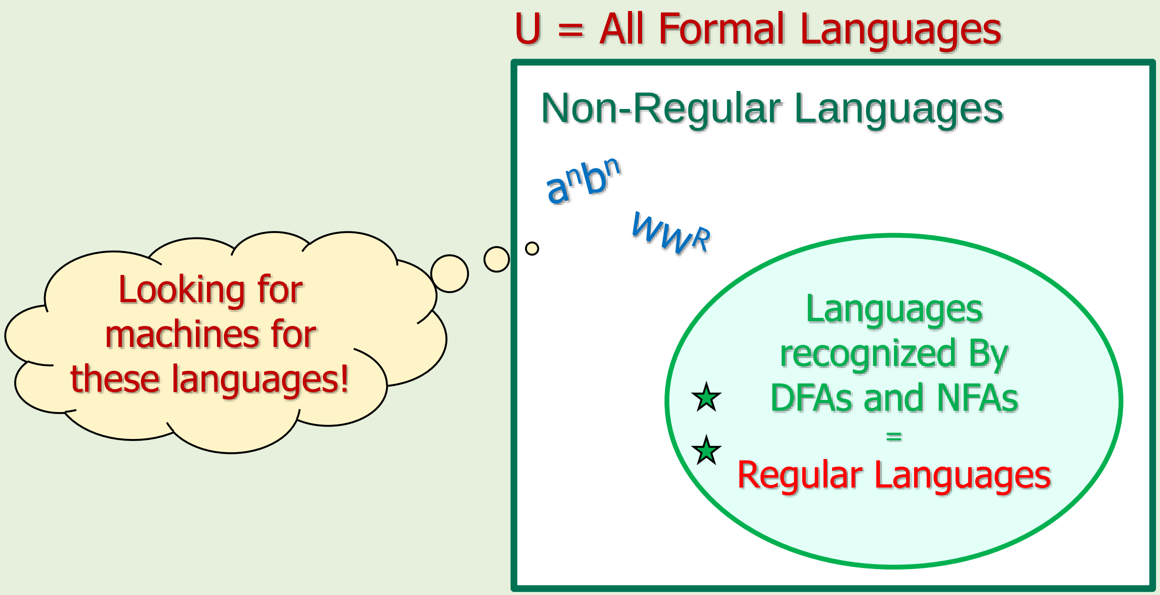
# Template for Constructing a New Class of Automata

Repeated

- To construct a new class of automata, we need to respond the following questions:
  1. Why do we need a new class of machines? (Justification)
  2. Name of the new class
  3. Building blocks of the new class
  4. How they work
     1. What is the "starting configuration"?
     2. What would happen during a timeframe?
     3. When would the machines halt (stop)?
     4. How would a string be Accepted/Rejected?
  5. Formal Definition
  6. Their power: this class versus previous class
  7. What would be the next possible class?

# Why We Need a New Class

- We learned that DFAs and NFAs have equal power.
  - Both recognize "regular languages".
- So, we are looking for a "more powerful class of automata" that can recognize all, or at least some of the "non-regular languages".



# Why We Need a New Class

- What was missing in NFAs that made them incapable of recognizing non-regular languages?

**Memory!**

- One might say, NFAs had memory:

Input Tape

- Yes, input tape is memory but read-only!
  - The machine does NOT have write capability during its operation.

- We are going to add some Read/Write memory to NFAs and construct a new class of automata.

# Name of the New Class

- The memory of this new class is structured as a "stack".

- So, we call our new class "pushdown automata (PDA)".
  - Both deterministic and nondeterministic PDAs can be defined.
  - But the nondeterministic ones are more flexible and more interesting.

- Therefore, the new class' complete name would be:

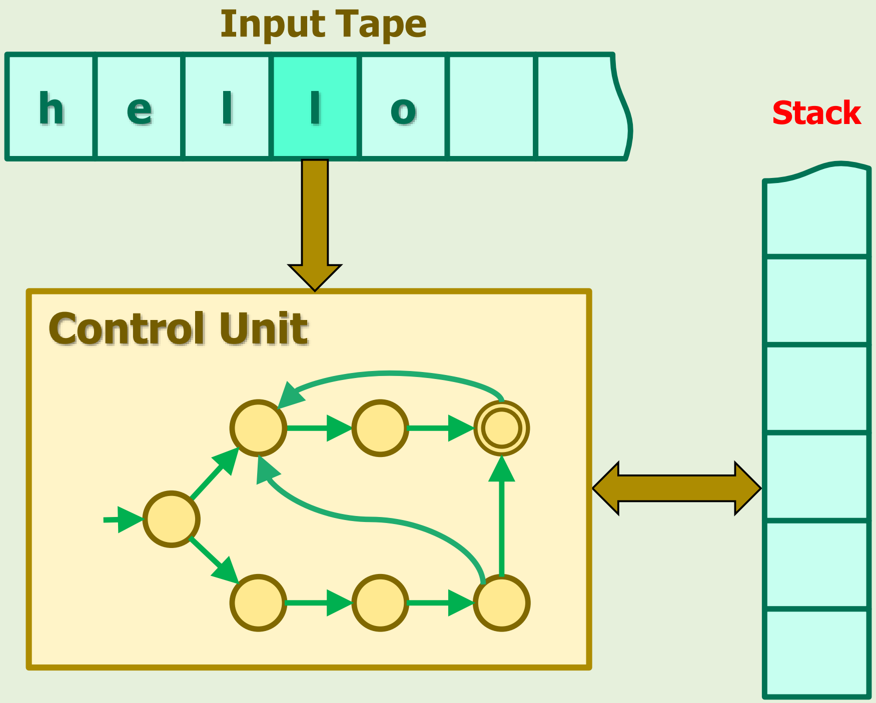
**Nondeterministic Pushdown Automata (NPDA)**

- Note that even though NPDAs are finite automata, but the "finite" is NOT mentioned in the name!

# NPDAs Building Blocks

# NPDAs Building Blocks

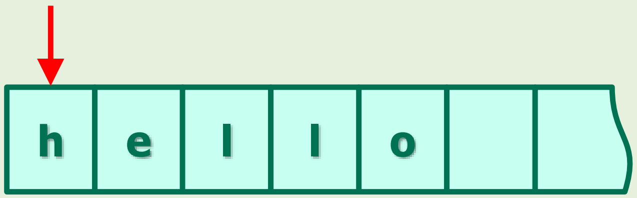
- An NPDA has 3 main blocks:
  1. Input Tape
  2. Stack
  3. Control unit




- Let's see each block in detail.

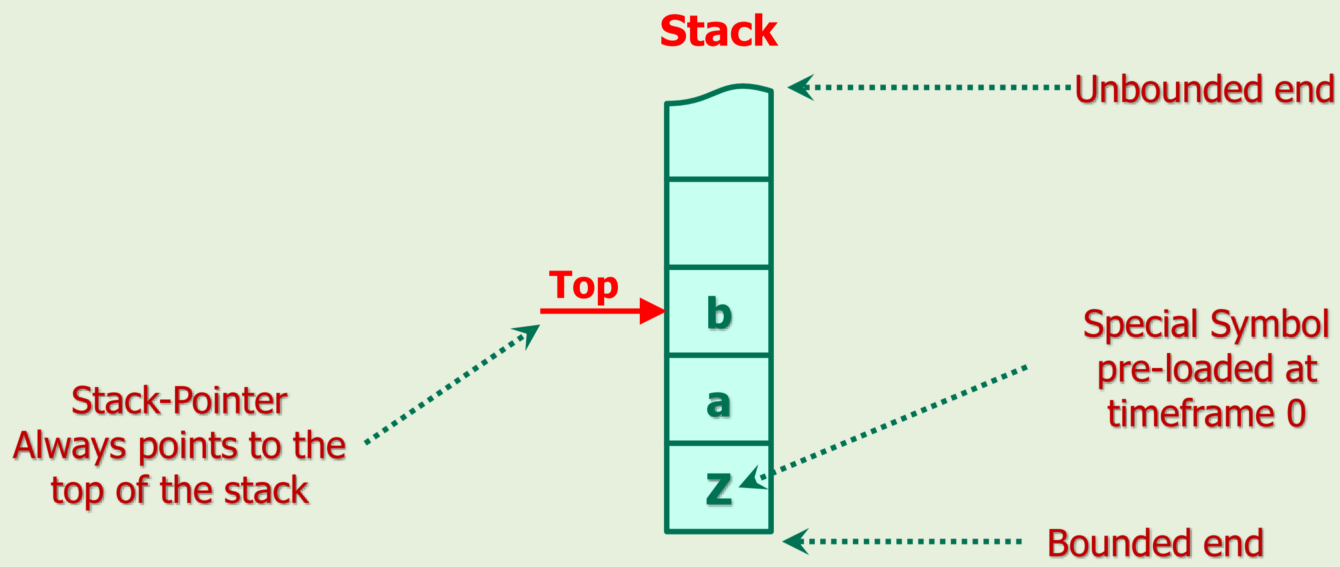
# 1. Input Tape

- The 'input tape' of NPDAs is exactly the same as DFAs'.



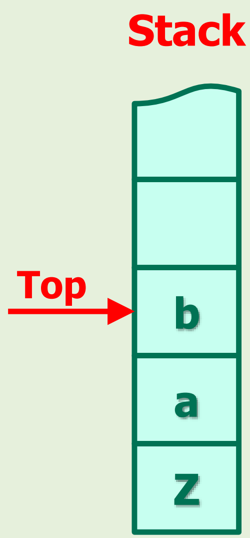

- For the detail, please review DFAs' input tape structure if you need to refresh your mind.

# 2. Stack: Structure




- The special symbol 'Z' is written at the bottom of the stack by the machine when it starts.
  - When control unit detects 'Z', it knows that the stack is empty.

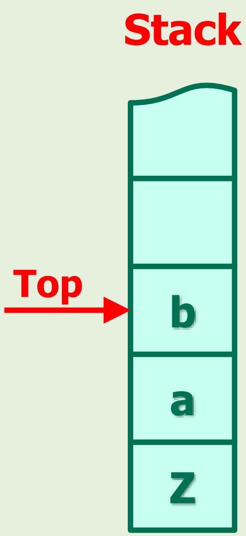
# 2. Stack: Note




- Note that the "stack alphabet" and the "input tape alphabet" can be totally different. (Will be covered later.)

# 2. Stack: How It Works

## Operations on Stacks

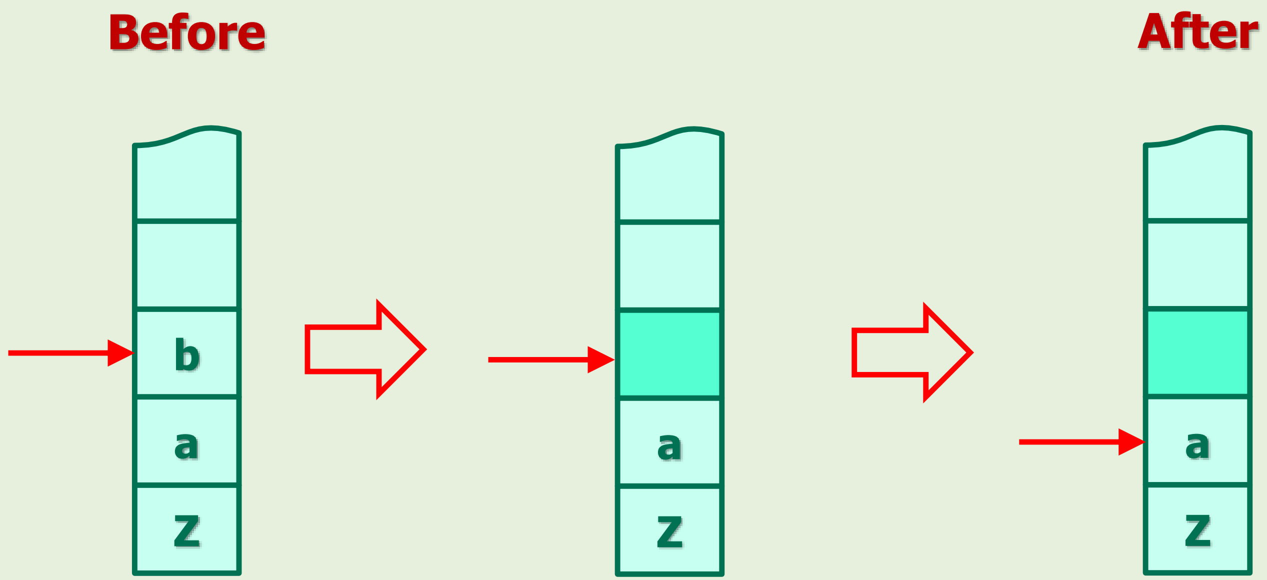



- Stack works based on last in – first out (LIFO) manner.
- The basic operations on stacks are "pop" and "push".
  - These operations are similar to what you know about the stack data structure in Java.
- Nevertheless, let's have a review of these basic operations.

## 2. Stack: Operations on Stacks

### Pop

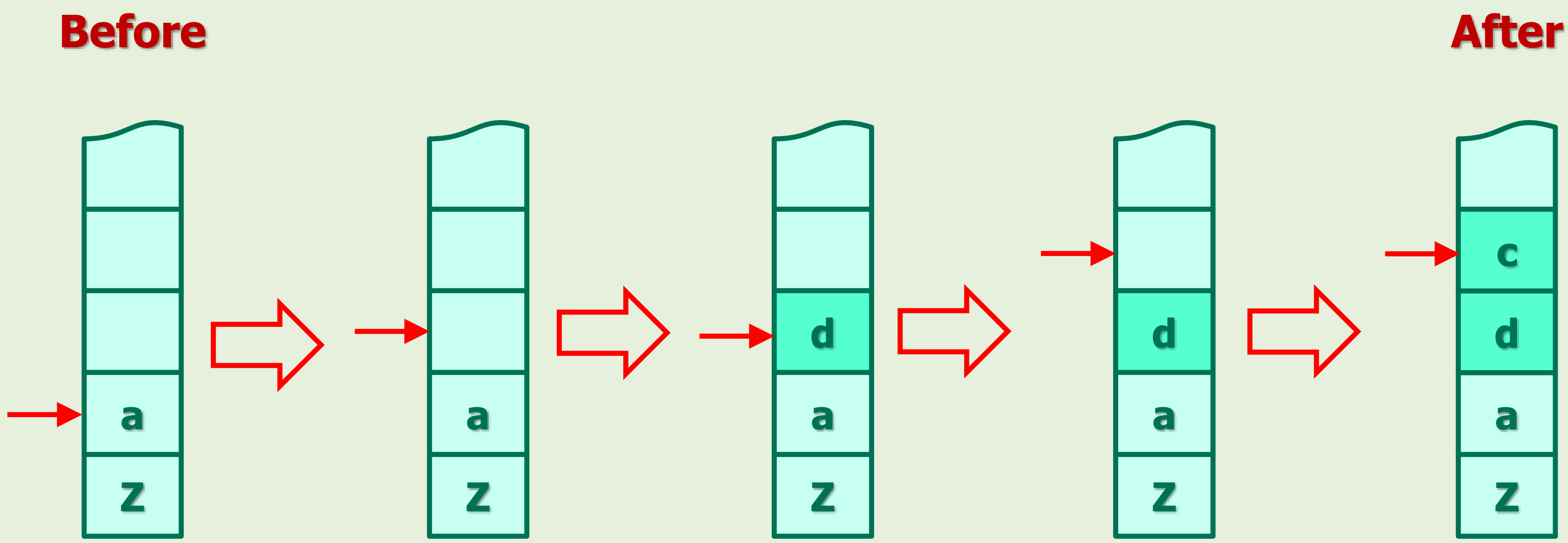
1. Remove the symbol at which the stack-pointer is pointing
2. Move the stack-pointer one cell down



# 2. Stack: Operations on Stacks

## Push

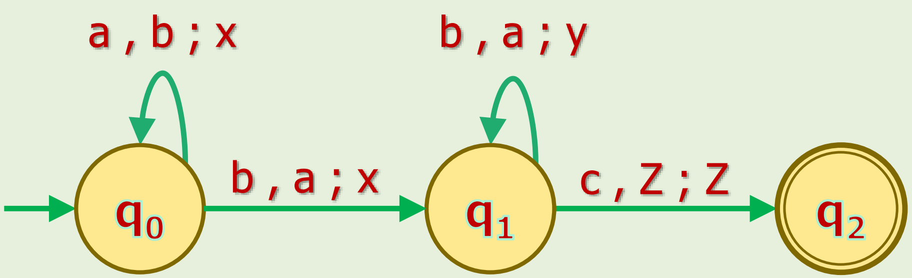
1. Move the stack-pointer one cell up
2. Put the string (e.g. 'cd') in the stack, the right symbol first (i.e. push 'd' first, then 'c')



# 3. Control Unit: Structure

- The control unit of NPDAs look pretty much like NFAs'.
  - They are represented by "transition graphs".

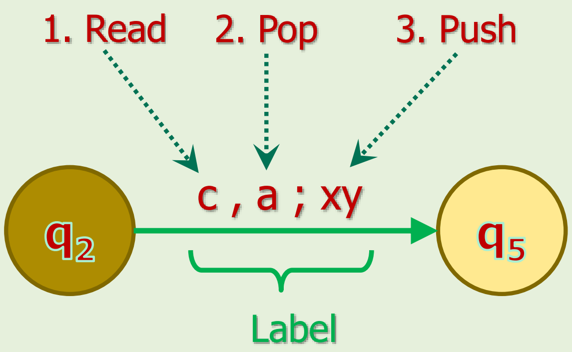
**Example 1**




- The only difference is how the edges are labeled.

- To understand the meaning of the 3 parts of the labels, let's analyze one transition.

# 3. Control Unit: Labels




- The label of an edge has 3 parts delimited by comma and semicolon:
  1. The input symbol (e.g. 'c') that should be read from the tape
  2. The symbol at the top of the stack (e.g. 'a') that should be popped
  3. The string (e.g. 'xy') that should be pushed into the stack

- We'll explain in detail what they mean shortly!

# How NPDAs Work

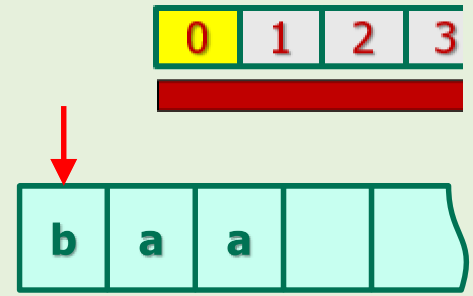
# How NPDAs Work

Repeated

- To understand the operations of NPDAs (and any other class of machines), we should clearly respond to the following questions:

1. What is the "starting configuration"?
2. What would happen during a timeframe?
3. When would the machines halt (stop)?
4. How would a string be Accepted/Rejected?

# 1. NPDAs Starting Configuration

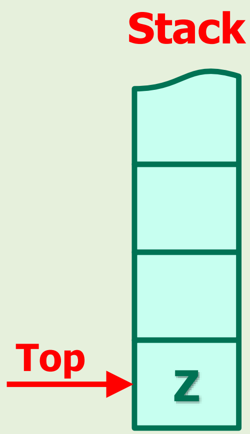



## Clock

- The clock is set at timeframe 0.

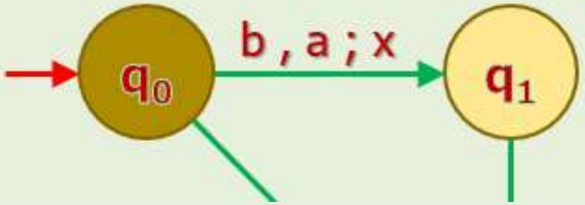
## Input Tape

- The input string has already been written on the tape.
- The read-head is pointing to the left-most symbol.




## Stack

- The stack is initialized by the symbol 'Z'.
- The stack-pointer is pointing to the 'Z'.




## Control Unit

- The control unit is set to initial state.

# 2. What Happens During a Timeframe

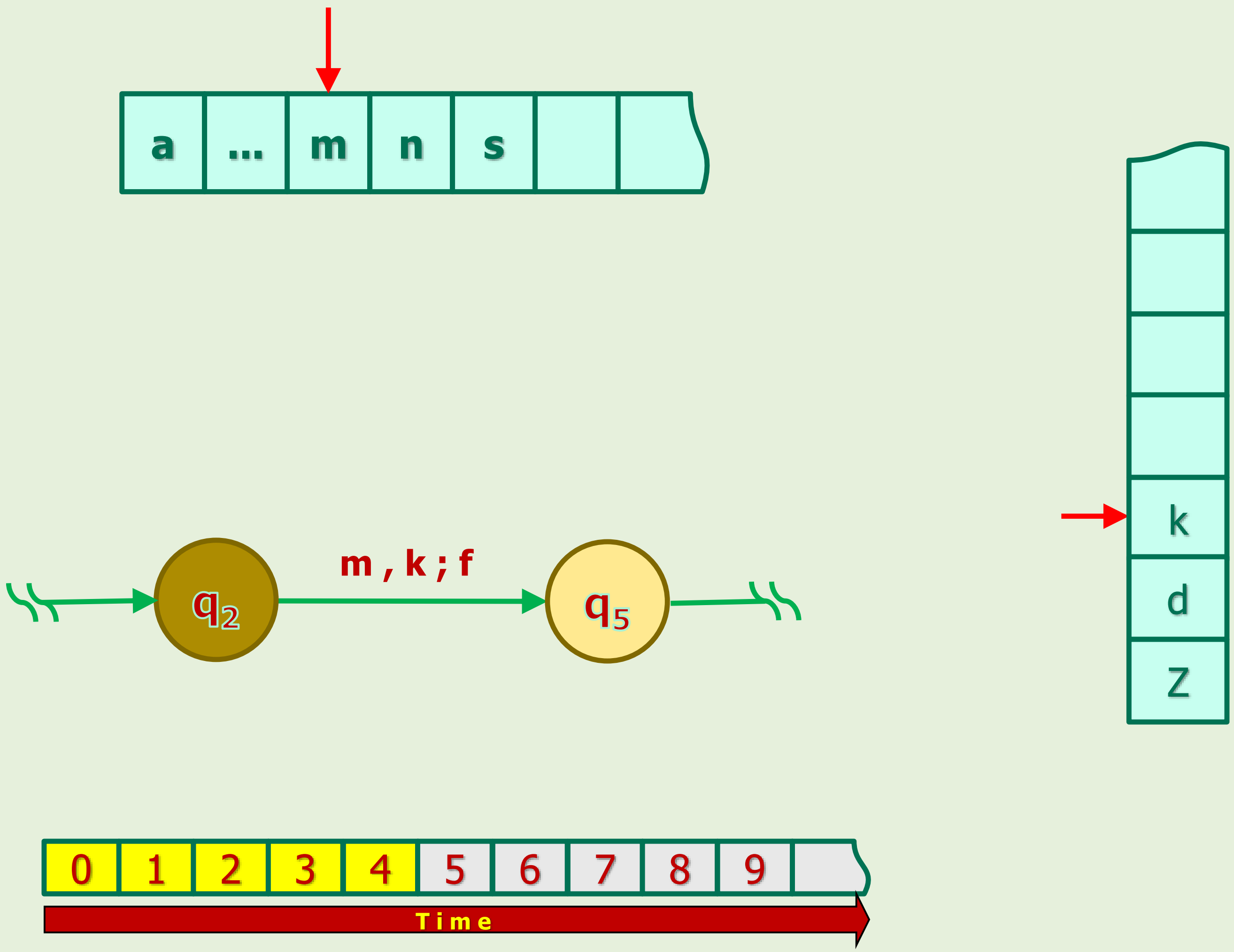
## Transition

- Several tasks happen during a timeframe.
- The combination of these tasks is called a "transition".

- Before describing them in detail, let's visualize them through an example.

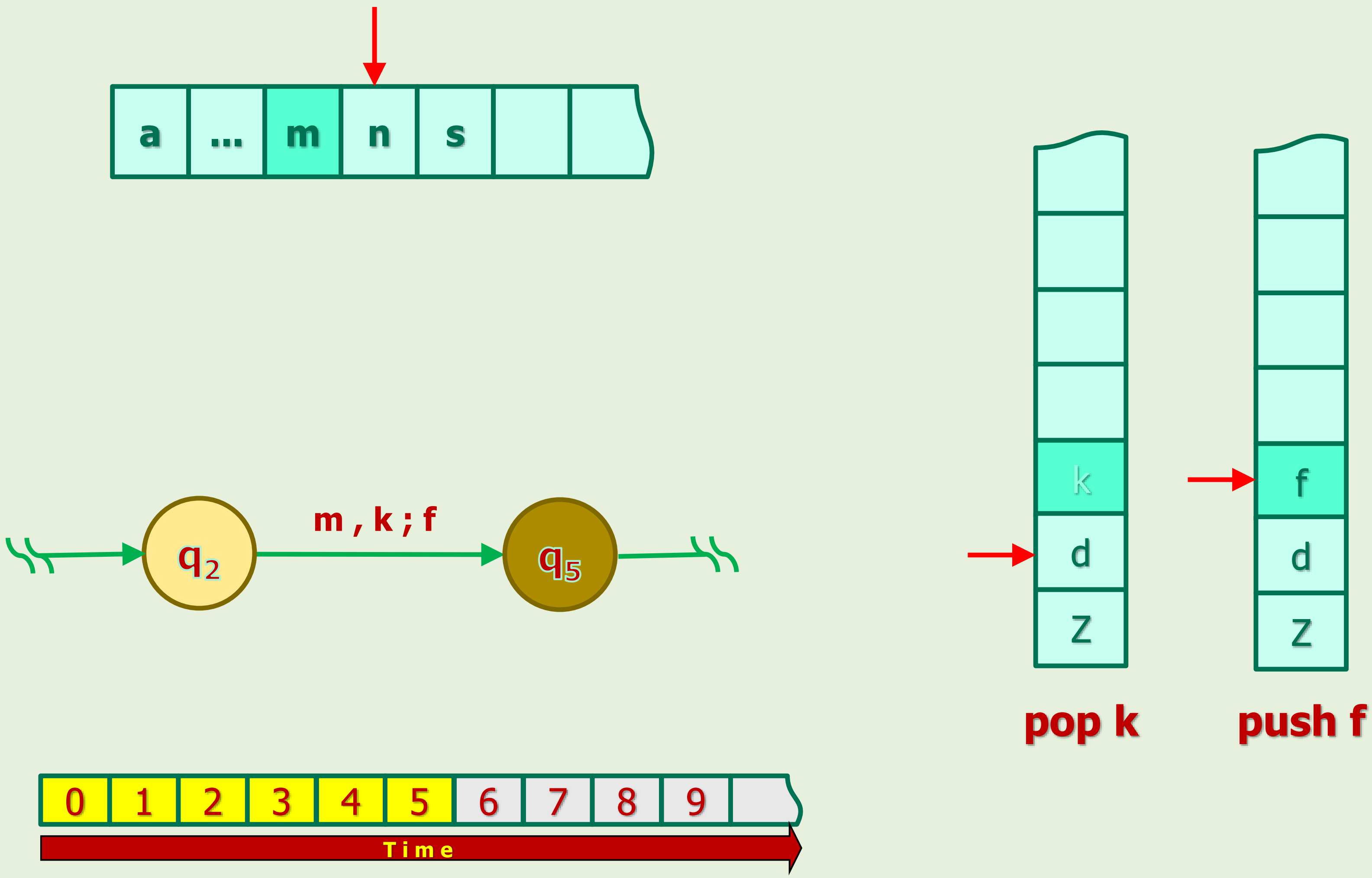
# Transition Example: Altering Stack Data

## Example 2: Timeframe 4



# Transition Example: Altering Stack Data

## Example 2: Timeframe 5



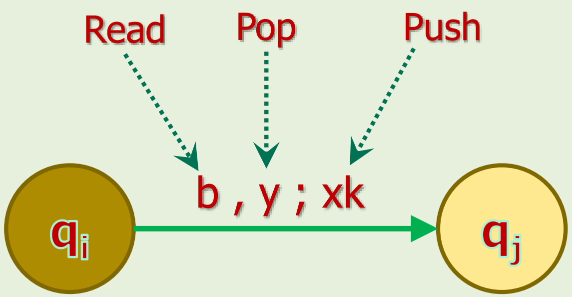
# 2. What Happens During a Timeframe

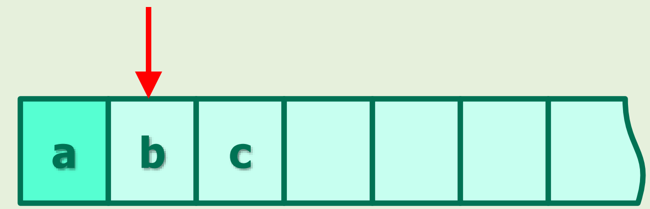
- Roughly speaking, the following tasks happen during a timeframe:

1. A symbol at which the read-head is pointing, is consumed.
2. A symbol will be popped from the stack.
3. A string will be pushed into the stack
4. The control unit makes its move based on the "logic of the transition".

- What is the "logic of the transition" of NPDAs?

# The Logic of Transitions




**If (Condition)**

– the input symbol is 'b'

AND

– the top of the stack is 'y'

**Then (Operation)**

– consume 'b'

AND

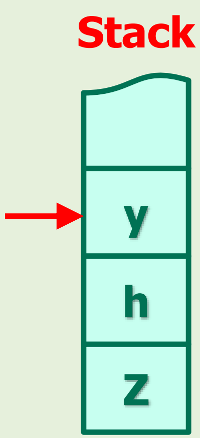
– pop 'y'

AND

– push 'xk'

AND

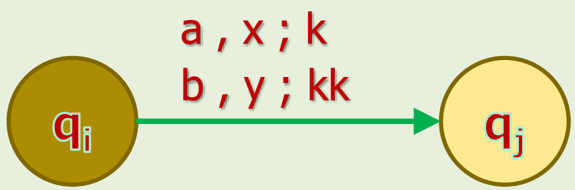
– transit to $q_j$




How does the machine look like after this transition?

# Multiple Labels

- Note that a transition might have multiple labels.
- In that case, we stack them over the edge as we did for DFAs.




**If**

- the input symbol is 'a'

AND

- the top of the stack is 'x'

**Then**

- consume 'a'

AND

- pop 'x'

AND

- push 'k'

AND

- transit to $q_j$

**OR**

**If**

- the input symbol is 'b'

AND

- the top of the stack is 'y'

**Then**

- consume 'b'

AND

- pop 'y'

AND

- push 'kk'

AND

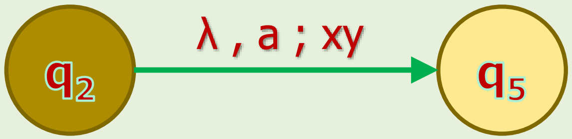
- transit to $q_j$

# Relaxing the Conditions and the Operations

- If we put λ in any part of the labels, it'd mean "no condition" or "no action" in that part.
- So, by using 'λ', we can relax the conditions and the operations.
- Let's see some examples.

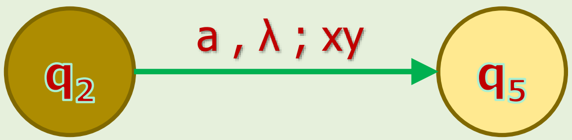
# Relaxing the Conditions by λ

## Example 3




- What does this transition mean?
  - If top of the stack is 'a', (Condition)
  - then pop 'a' AND push 'xy' AND make the move. (Operation)
  - Do NOT consume any symbol!

## Example 4




- What does this transition mean?
  - If the input symbol is 'a', (Condition)
  - then consume 'a' AND push 'xy' AND make the move. (Operation)
  - Do NOT pop anything!

# Relaxing the Conditions by λ

## Example 5


- What does this transition mean?
  - If the input symbol is 'c' AND the top of the stack is 'a', (Condition)
  - then consume 'c' AND pop 'a' AND make the move. (Operation)
  - Do NOT push anything!

## Example 6


- What does this transition mean?
  - If the top of the stack is 'a', (Condition)
  - then pop 'a' AND make the move. (Operation)
  - Do NOT consume any symbol AND do NOT push anything!

# Relaxing the Conditions by λ

## Example 7


- What does this transition mean?
  - If the input symbol is 'c', (Condition)
  - then consume 'c' AND make the move. (Operation)
  - Do NOT pop anything AND do NOT push anything!

- We can also put λ in read and pop parts or even all three parts.
  - All of these situations will be covered later.

- Now let's make the transition definition more precise.

# 2. What Happens During a Timeframe

## Transition Tasks

- The following tasks happen during a timeframe.

1. Zero or one symbol at which the read-head is pointing, is consumed.
2. Zero or one symbol is popped from the stack.
3. A string is pushed into the stack. (could be empty string.)
4. The control unit makes its move based on the "logic of the transition".

- Now let's see some visual transition examples.

# Transition Examples

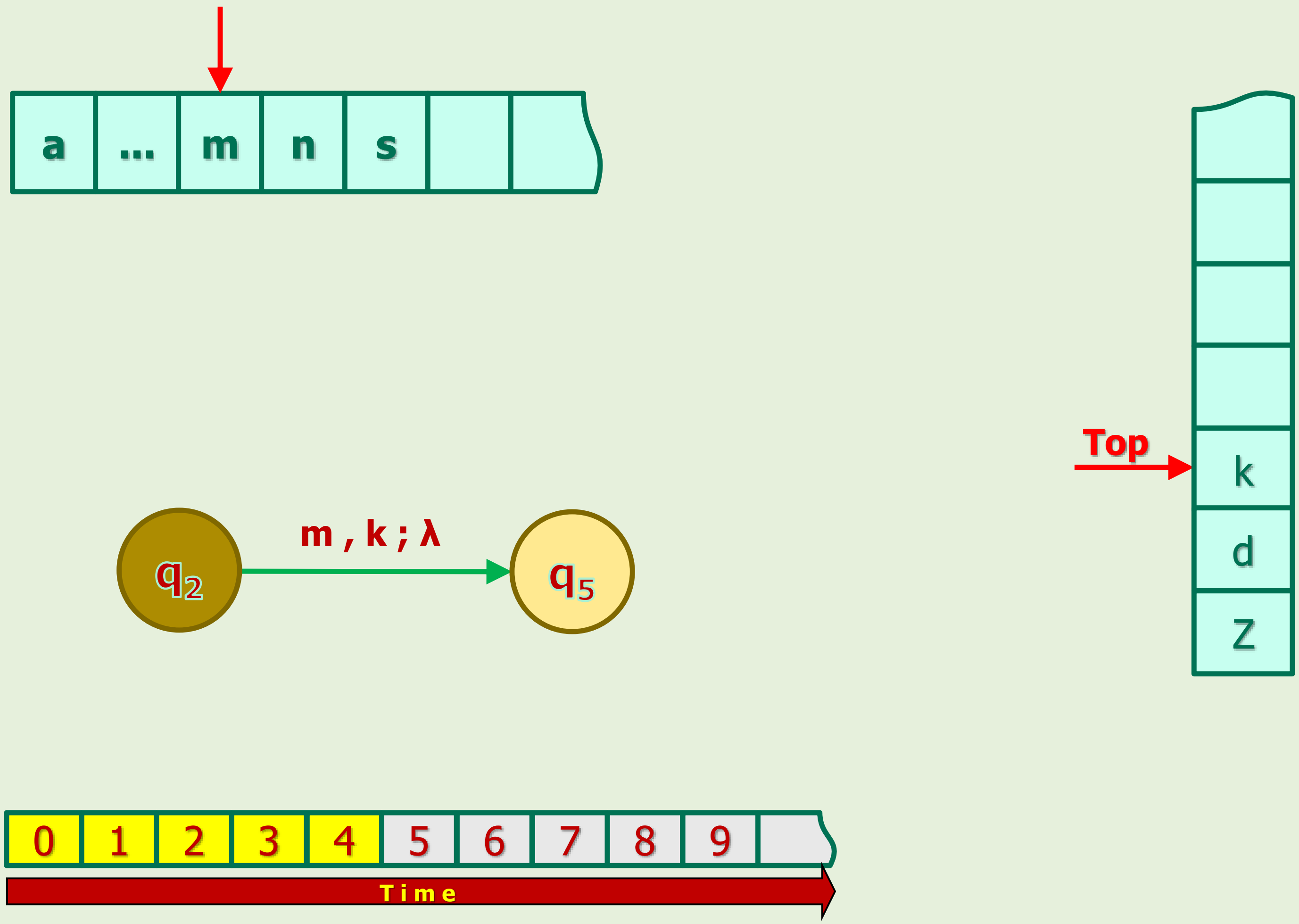
# Transition Examples

- The examples will show:
  - a partial transition graph
  - an input tape
  - a stack
  - a clock

- We assume that the machine is in the middle of its operation at timeframe n.
- The question is: in what configuration would the machine be at timeframe n+1?

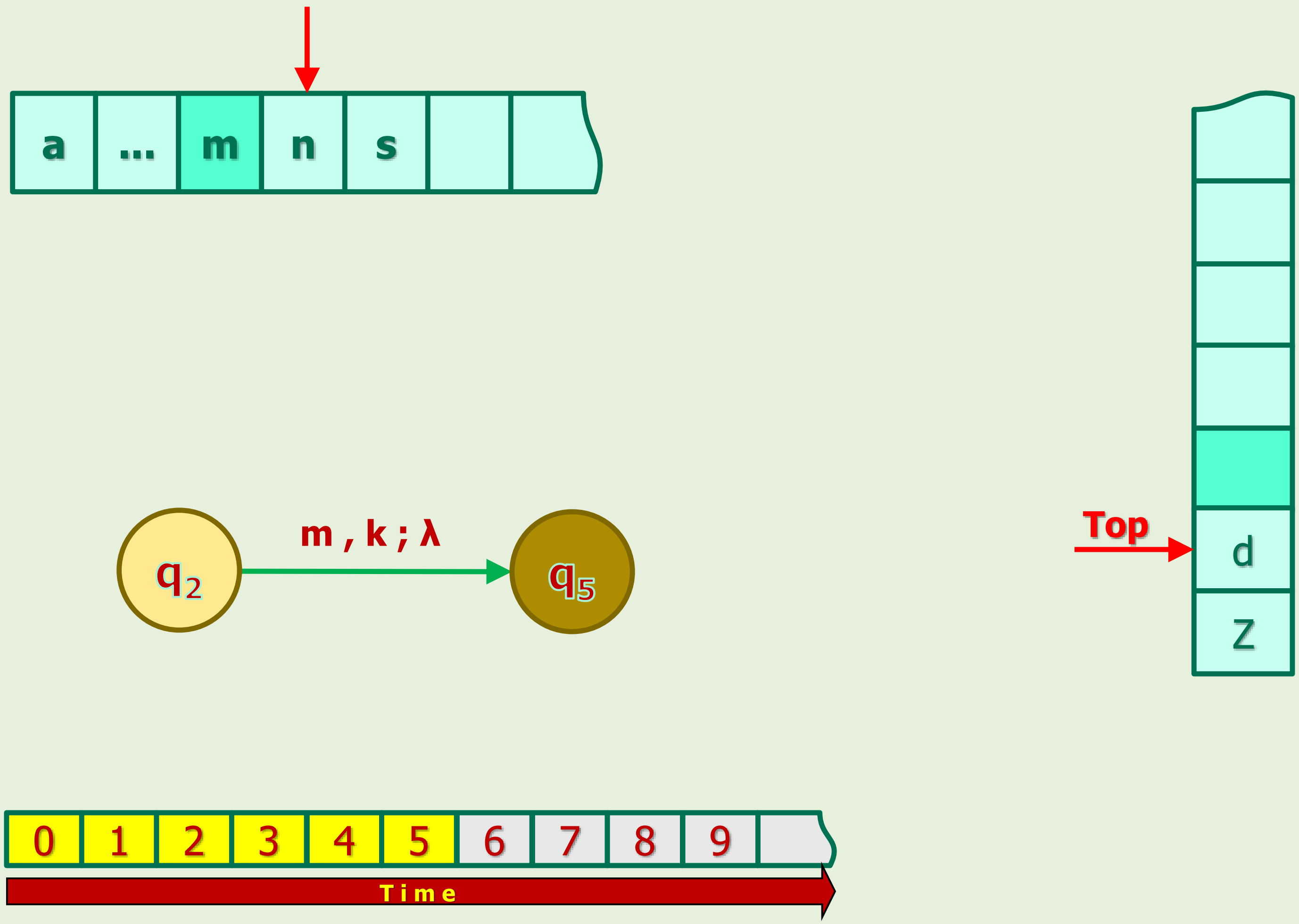
# Transition Examples: Popping Stack Data

## Example 8: Timeframe 4



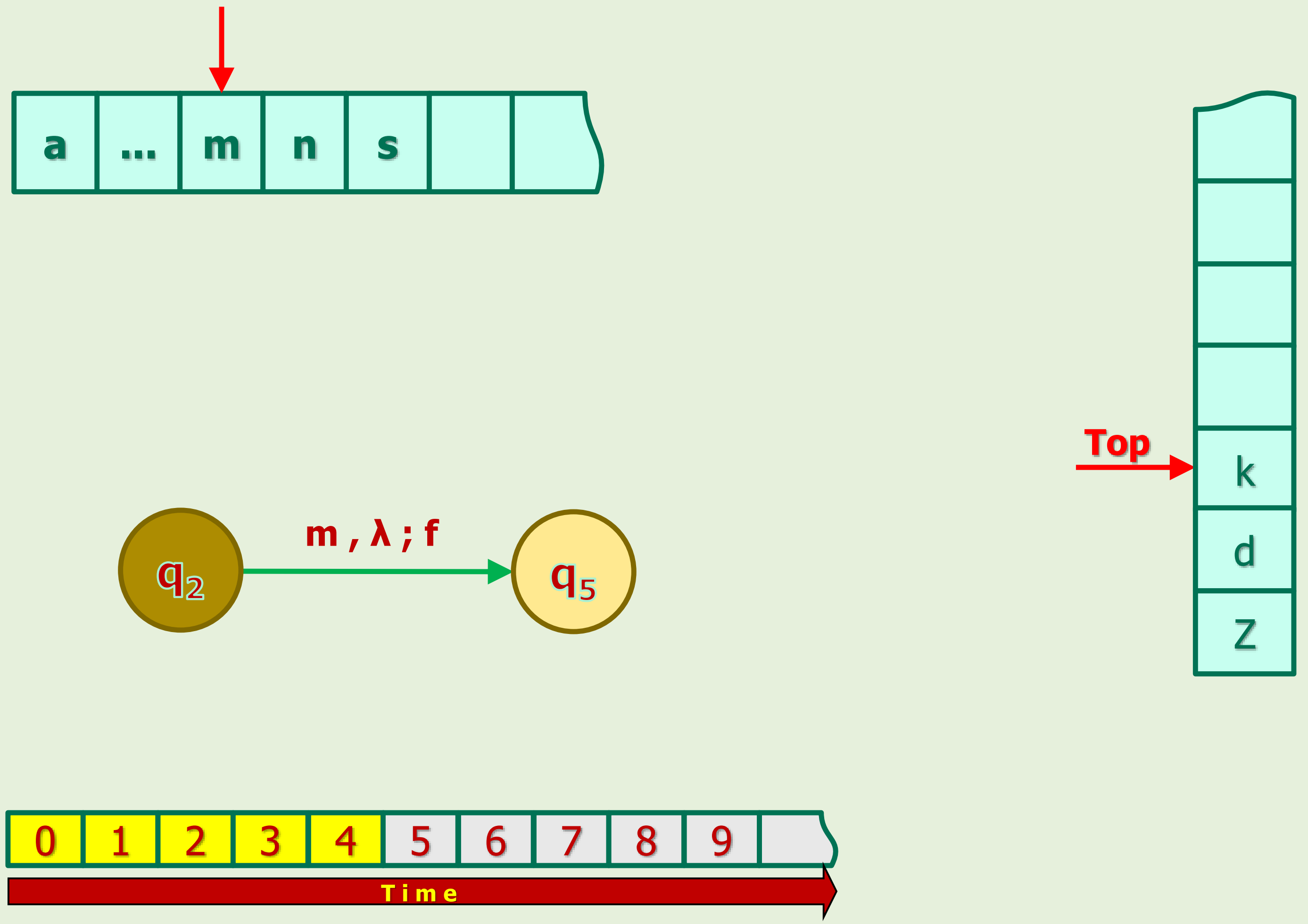
# Transition Examples: Popping Stack Data

## Example 8: Timeframe 5



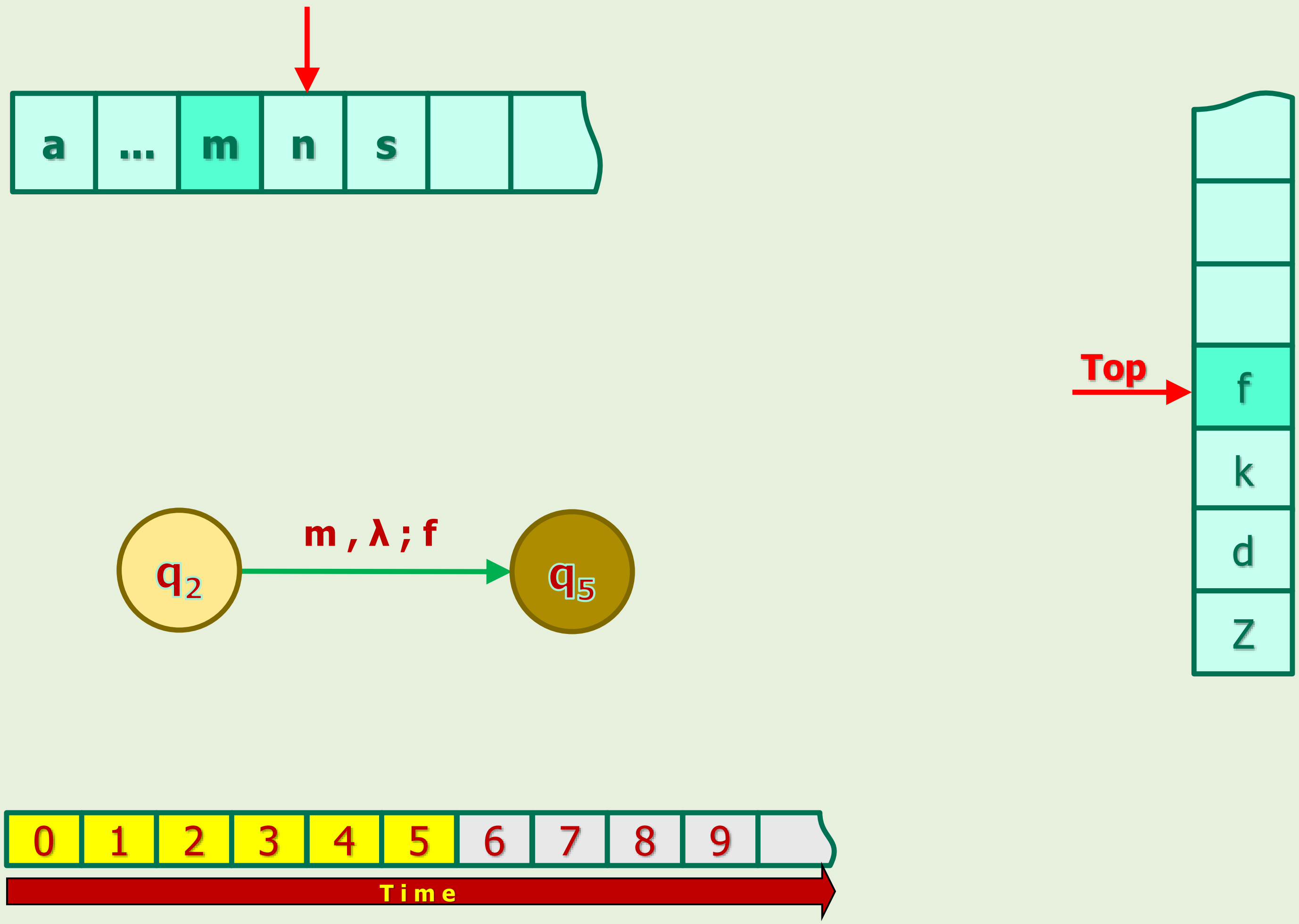
# Transition Examples: Pushing Data into Stack (1)

## Example 9: Timeframe 4



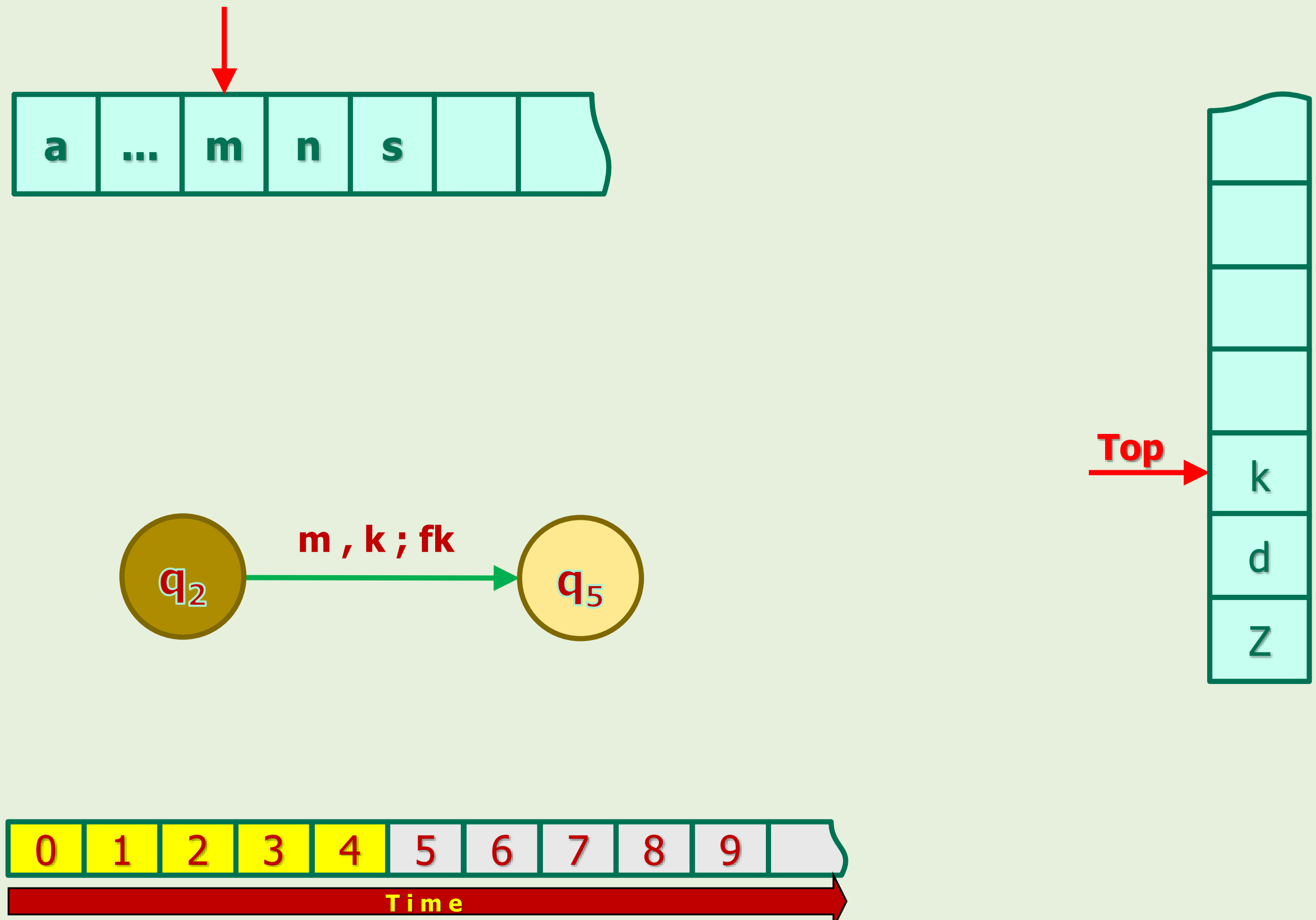
# Transition Examples: Pushing Data into Stack (1)

## Example 9: Timeframe 5



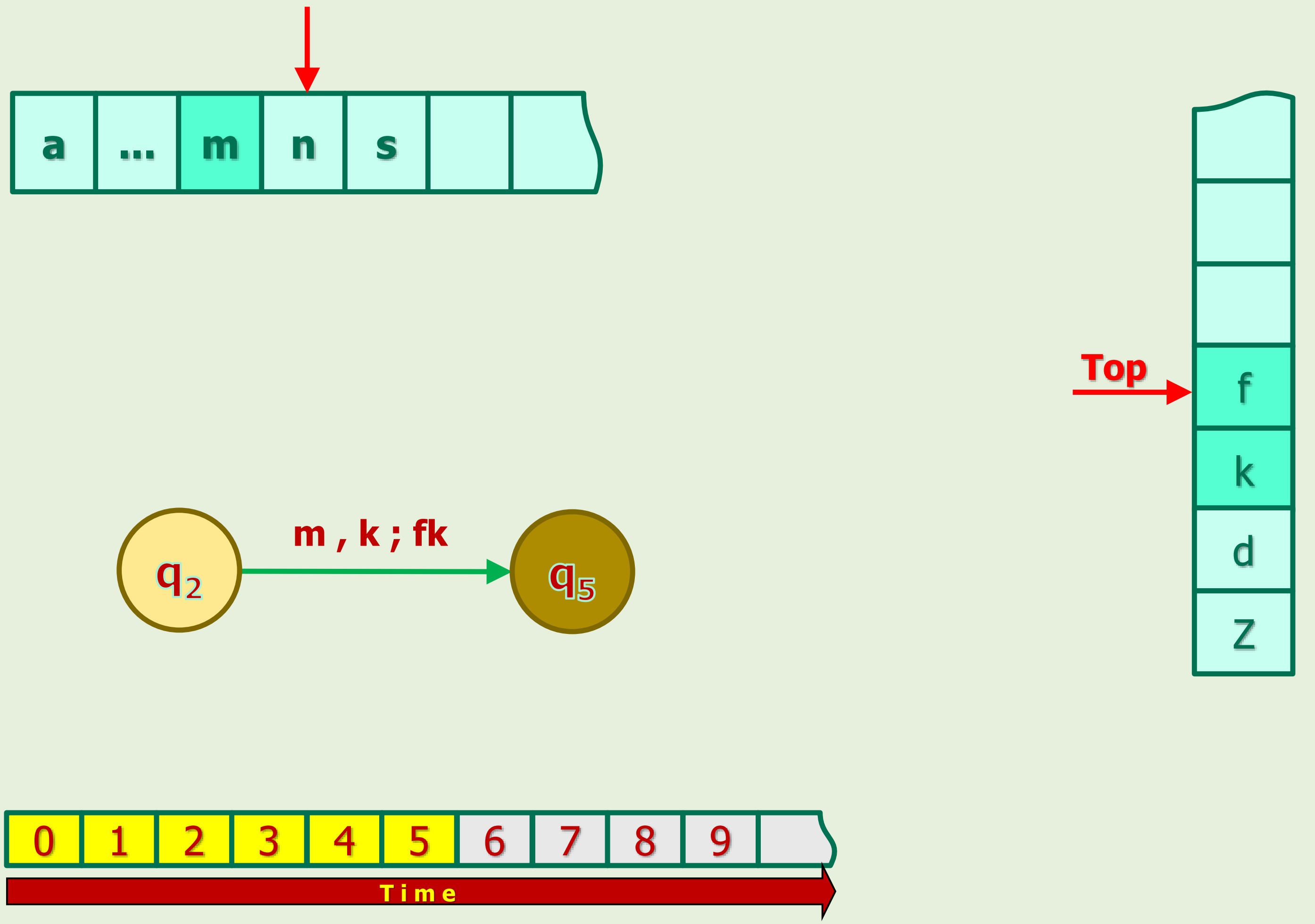
# Transition Examples: Pushing Data into Stack (2)

## Example 10: Timeframe 4



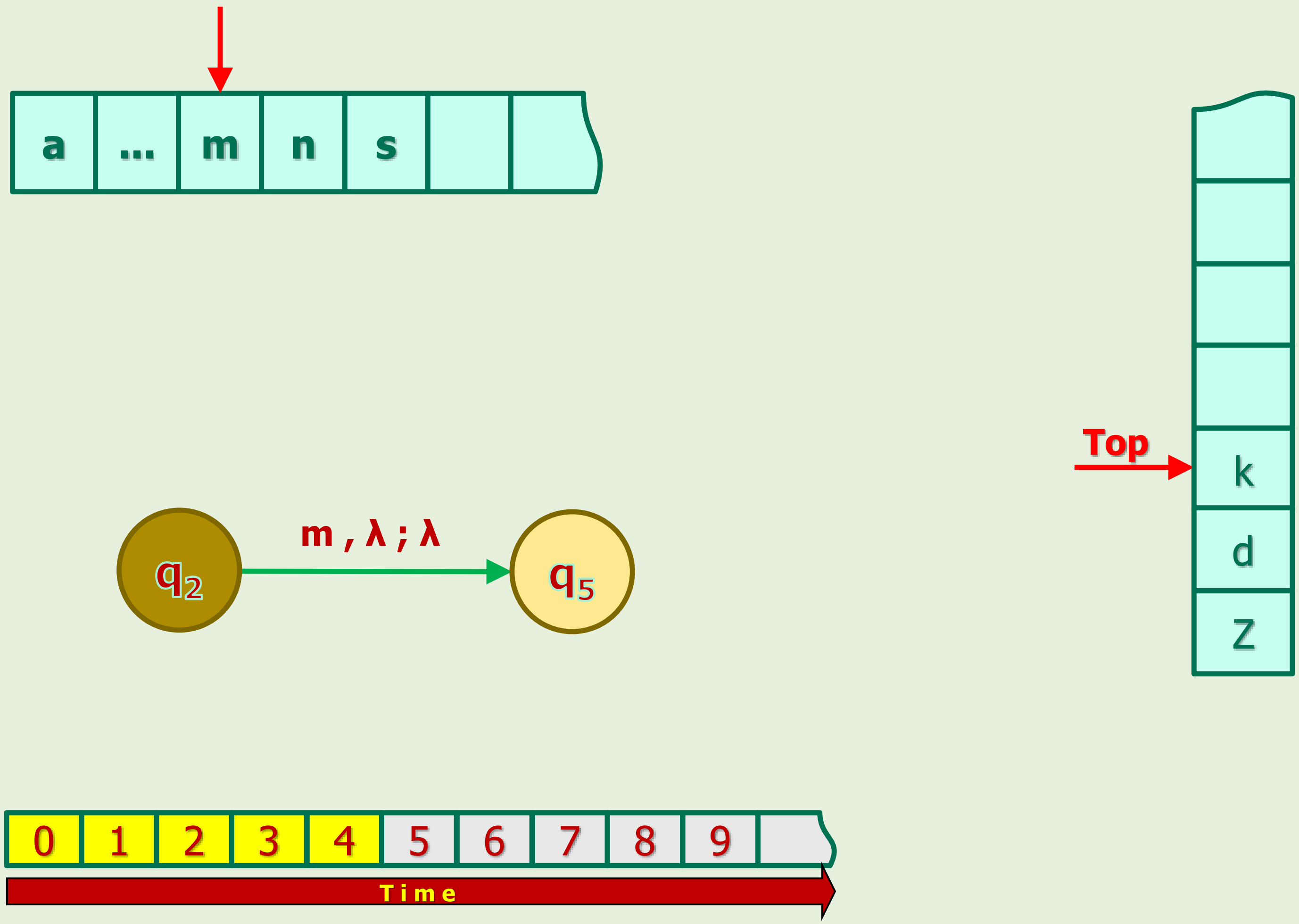
# Transition Examples: Pushing Data into Stack (2)

## Example 10: Timeframe 5



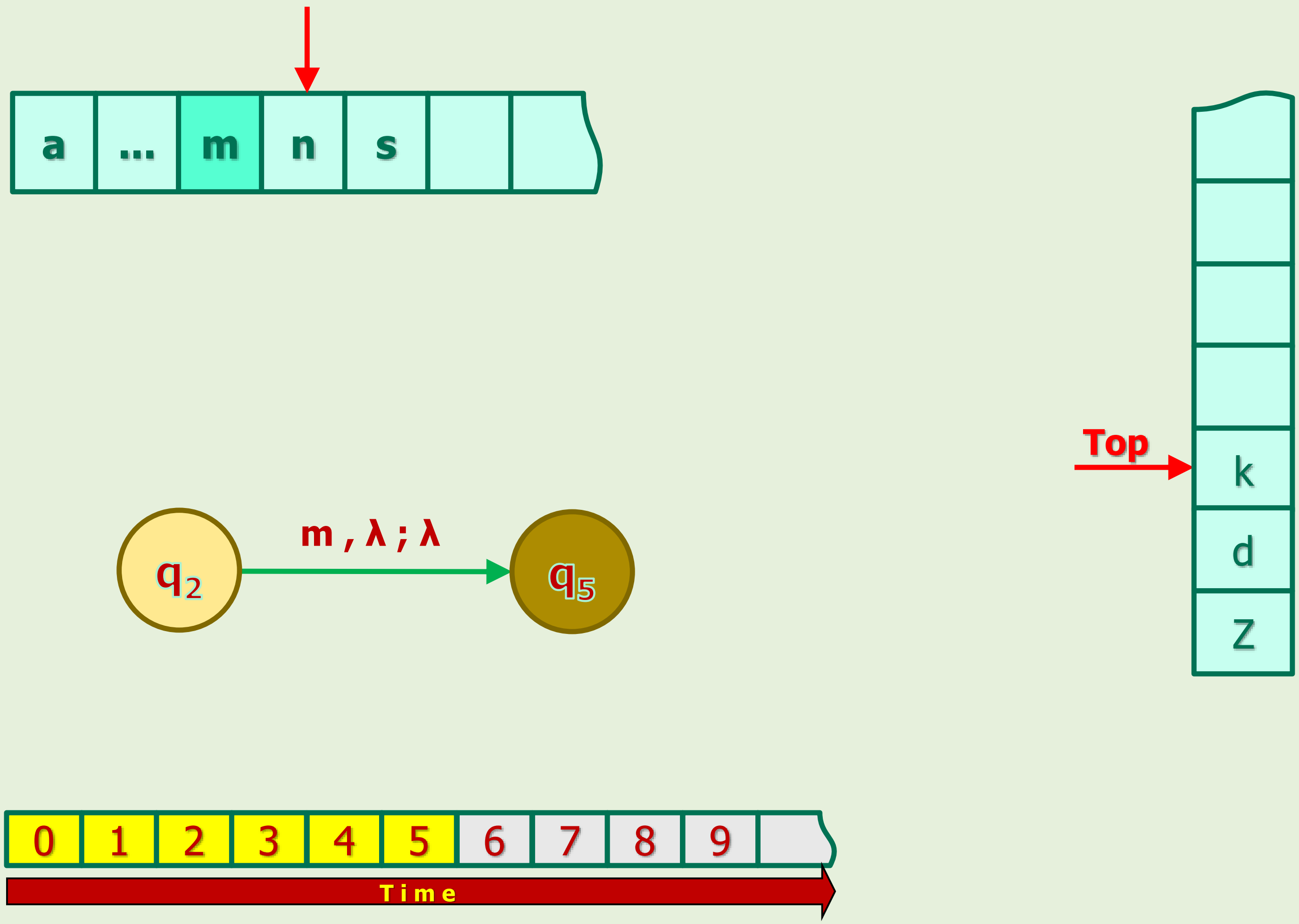
# Transition Examples: No Action on Stack

## Example 11: Timeframe 4



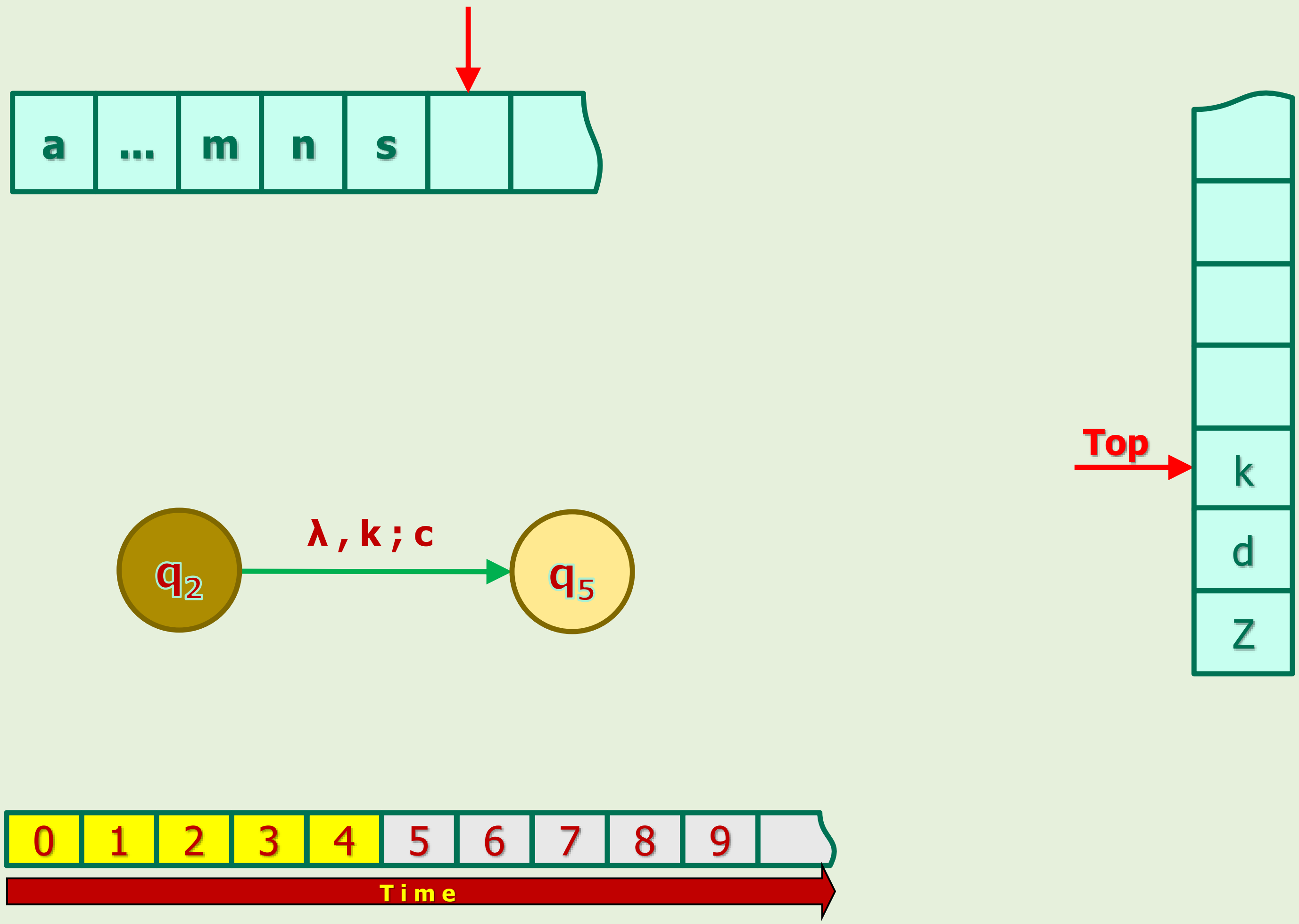
# Transition Examples: No Action on Stack

## Example 11: Timeframe 5



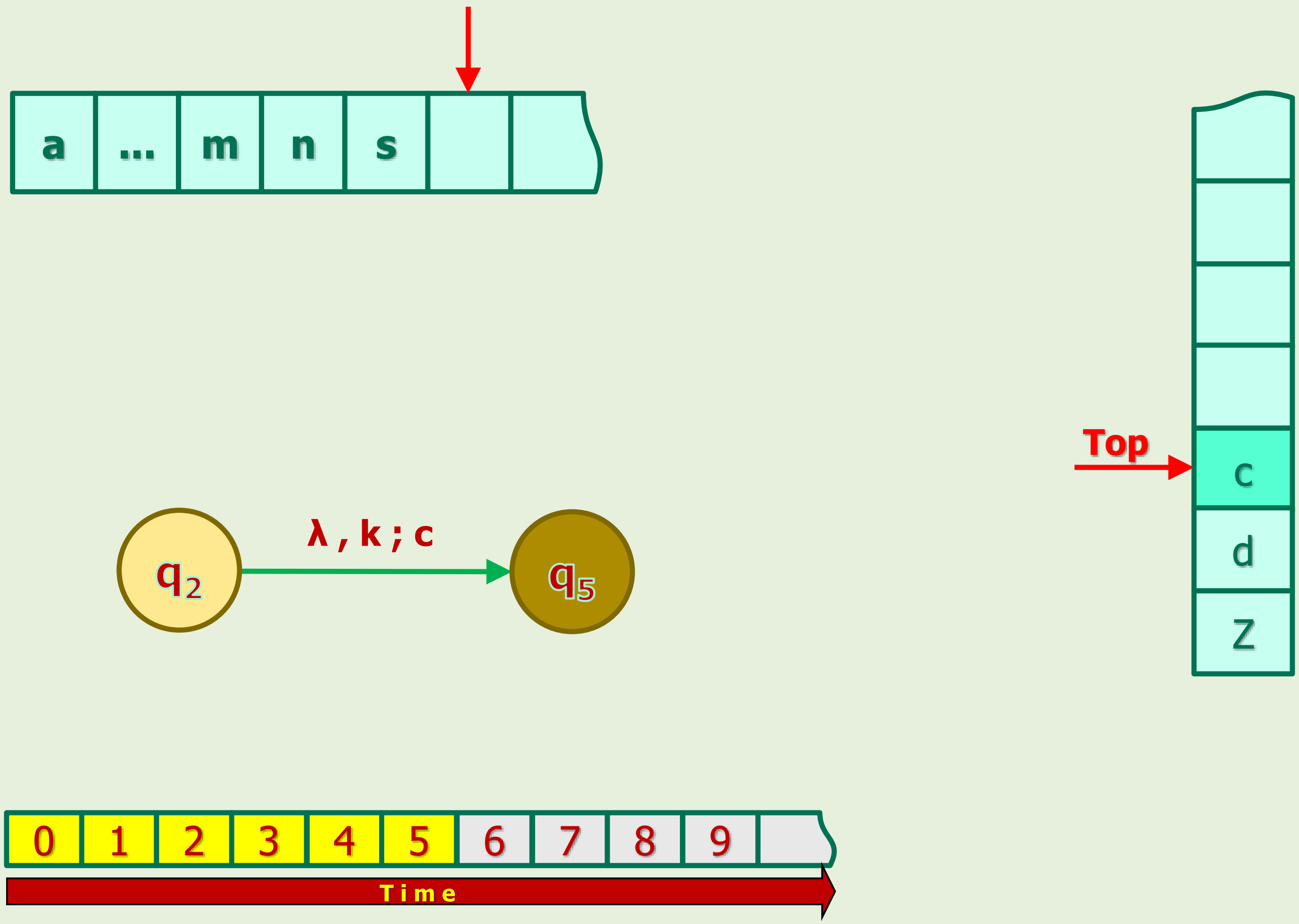
# Transition Examples: No Action on Input Tape

## Example 12: Timeframe 4 (Note the difference with DFAs)



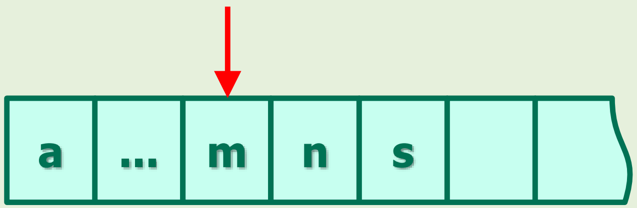
# Transition Examples: No Action on Input Tape

## Example 12: Timeframe 5 (Note the difference with DFAs)

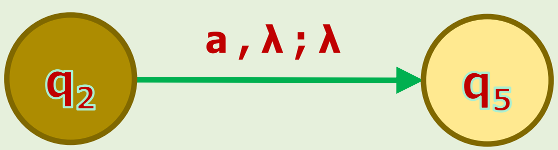


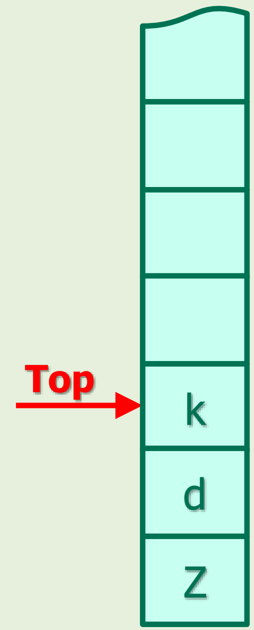
# Transition Examples: No Transition

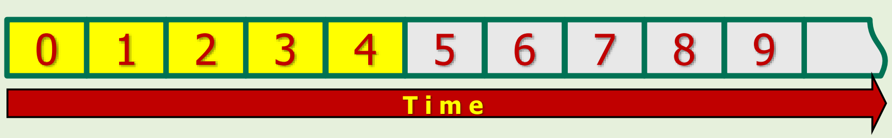
## Example 13: Timeframe 4




- No further transition because the condition (input='a') for next transition is not present. So, it "halts" in state $q_2$.

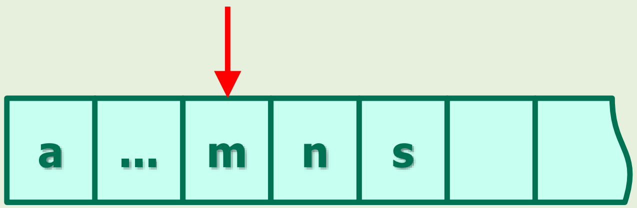


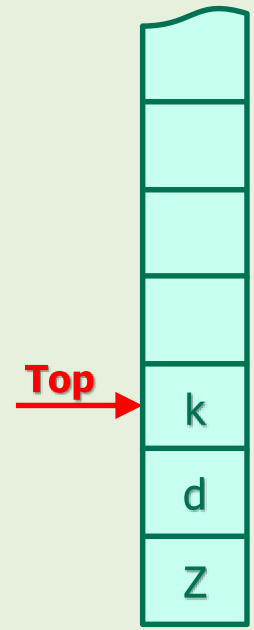



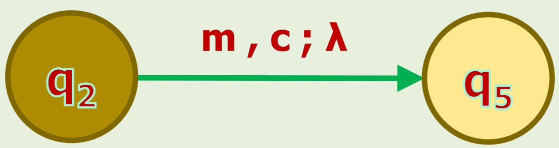
# Transition Examples: No Transition

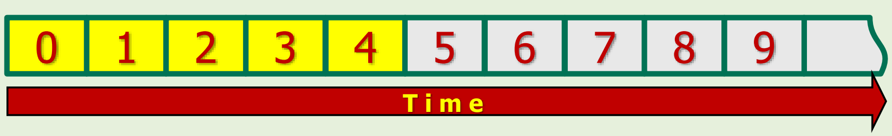
## Example 14: Timeframe 4




- No further transition because the condition (stack='c') for next transition is not present. So, it "halts" in state $q_2$.



# 3. When NPDAs Halt

- In the previous examples, we noticed that ...

  "All input symbols are consumed."

- ... was NOT sufficient for NPDAs to halt.
- NPDAs halt when the next transition conditions are NOT satisfied.

- Transition Conditions = input symbol + top of the stack

## Halt Logical Representation

$$\left.\begin{array}{l}\text{NPDAs halt.} \equiv \mathbf{h} \\ \text{IFF} \\ \text{They have zero transition.} \equiv \mathbf{z}\end{array}\right\} \mathbf{z} \leftrightarrow \mathbf{h}$$

# 4. How NPDAs Accept/Reject Strings

## Logical Representation of Accepting Strings By One Process

The string w is accepted (understood). $\equiv$ **a**

IFF

The NPDA halts. $\equiv$ **h**

AND

All symbols of w are consumed. $\equiv$ **c**

AND

The NPDA is in an accepting (final) state. $\equiv$ **f**

$$(h \land c \land f) \leftrightarrow a$$

- Note that NPDAs are nondeterministic.
  - Therefore, they might have several processes.
- The above conditions is for one process to accept a string.

# 4. How NPDAs Accept/Reject Strings

## Logical Representation of Rejecting Strings By One Process

$$(h \wedge c \wedge f) \leftrightarrow a$$

$$\sim(h \wedge c \wedge f) \leftrightarrow \sim a$$

$$(\sim h \vee \sim c \vee \sim f) \leftrightarrow \sim a$$

## Translation

The string w is rejected. ≡ $\sim$**a**

IFF

The NPDA does NOT halt. ≡ $\sim$**h**

OR

At least one symbol of w is NOT consumed. ≡ $\sim$**c**

OR

The NPDA is NOT in an accepting (final) state. ≡ $\sim$**f**

# 4. How NPDAs Accept/Reject Strings: Notes

1. Overall, NPDAs accept a string iff at least one process accept it.
2. Overall, NPDAs reject a string iff all processes reject it.
3. The final contents of the stack is NOT important in accepting or rejecting a string.
   - Because stack is in fact a workspace for rough drafting.
   - It is a place to store the middle results of the computation.
4. JFLAP has an option to accept a string when the stack is empty!
   - We do NOT use that at all.

- Now let's see NPDAs in action!
- To show the power of NPDAs, we'll design NPDAs for some of the famous non-regular languages.

# References


1. Linz, Peter, "An Introduction to Formal Languages and Automata, 5th ed.," Jones & Bartlett Learning, LLC, Canada, 2012
2. Kenneth H. Rosen, "Discrete Mathematics and Its Applications, 7th ed.," McGraw Hill, New York, United States, 2012
3. Michael Sipser, "Introduction to the Theory of Computation, 3rd ed.," CENGAGE Learning, United States, 2013
   ISBN-13: 978-1133187790