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

Fall 2019

Agenda of Day 13

- Solution of HW2
- Solution and Feedback of Quiz 4
- Summary of Lecture 12
- Lecture 13: Teaching ...
  - Pushdown Automata (part 1)

Solution and Feedback of Quiz 4 (Out of 22)

Section	Average	High Score	Low Score
01 (TR 3:00 PM)	18.6	22	14
02 (TR 4:30 PM)	19.1	22	12
03 (TR 6:00 PM)	19.6	22	14.5

Summary of Lecture 12: We learned ...

Regular Languages

- We could NOT construct a DAF/NFA for a^nb^n. Why?
- Because we need to count the number of a's and store it!
- And we cannot implement counter by DFAs/NFAs.
- So, we realized that languages are different and need to be categorized if we want to understand them better.
- We categorized the languages as ...
  - ... regular and non-regular.

- A language is called regular if ...
  - ... there exists a DFA/NFA to recognize it.

Finite Languages

- We proved that ...
  - All finite languages are regular.
- Formally speaking ...
  - If L is finite, then L is regular.
- The contrapositive of this theorem is true to. It means ...
  - If L is non-regular, then L is infinite.
  - All non-regular languages are infinite.

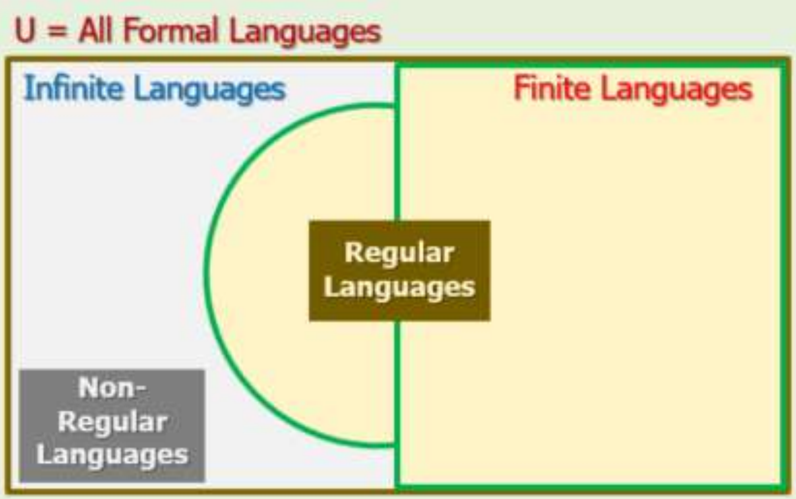
Any question?

Summary of Lecture 12: We learned ...

Languages Categorization

- So far, we categorized formal languages as:

1. Finite and Infinite
2. Regular and Non-Regular



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

- We learned some operations on regular languages that produce regular languages.
- It means, the family of regular languages is closed under those operations.

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

Any Question?

Pushdown Automata

Template for Introducing a New Class of Automata

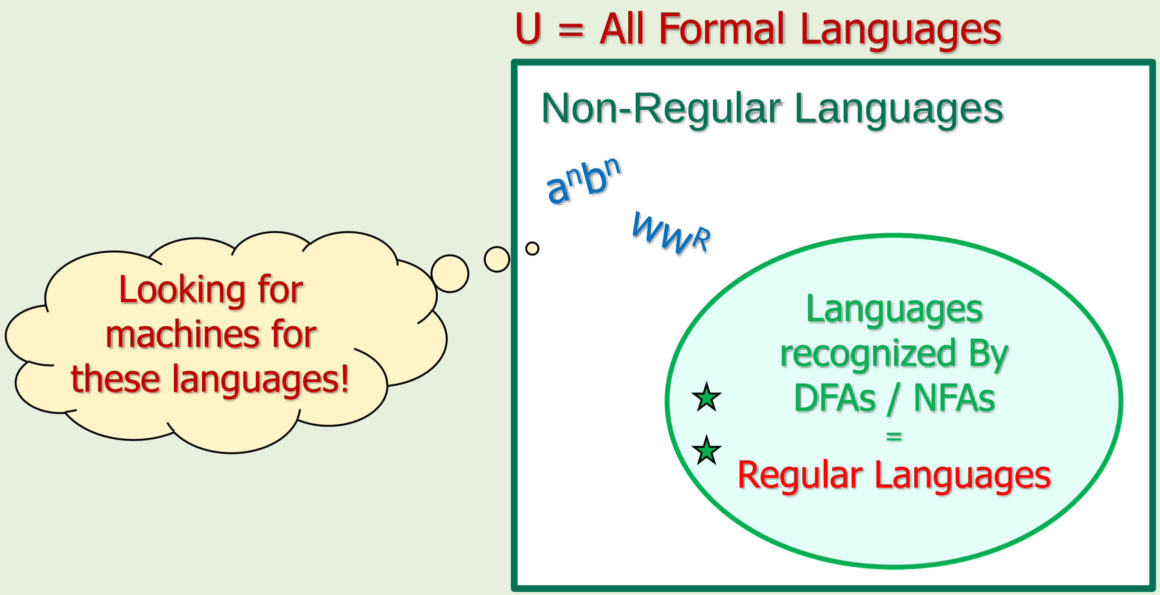
Repeated

- To construct a new class of automata, we'll follow the following steps:

1. Why do we need this new class? (Justification)
2. Name of this new class
3. Building blocks of this new class
4. How do they work?
   - 4.1. What is the starting configuration?
   - 4.2. What would happen during a timeframe?
   - 4.3. When would the machine halt?
   - 4.4. How would a string be Accepted/Rejected?
5. The automata in action
6. Formal definition
7. Their power: this class versus previous class
8. What would be the next possible class?

1. Why do We Need a New Class?

- So far, we've 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.



1. Why do We Need a New Class of Machines?

- 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 it's 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.

2. Name of the New Class

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

- So, generally we call this new class:

Pushdown Automata (PDA)

- Both deterministic and nondeterministic PDAs can be defined.
- Therefore, the new class' name specifically would be:

Deterministic Pushdown Automata (DPDA)

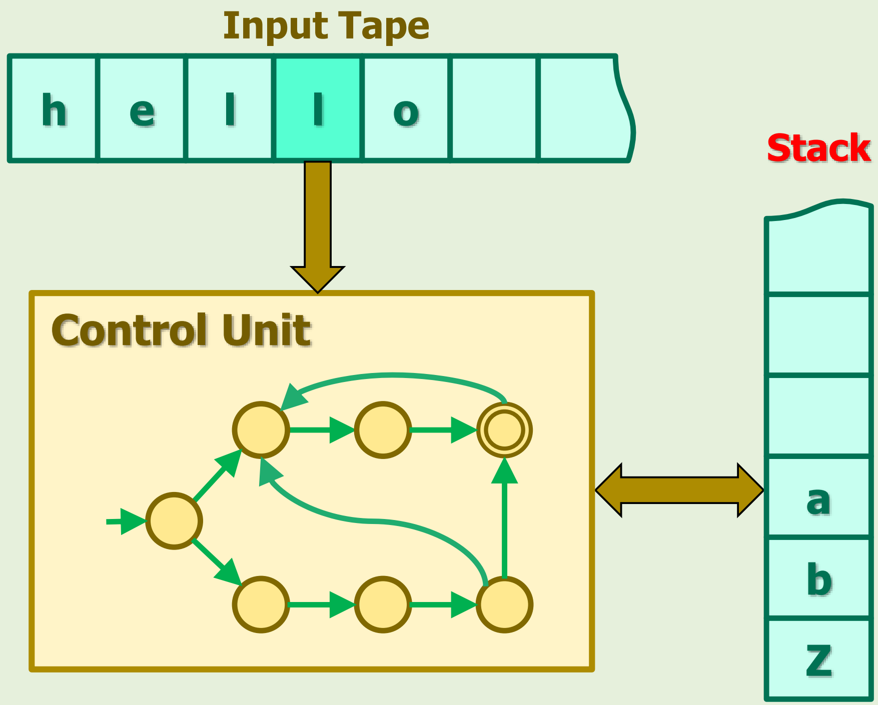
Nondeterministic Pushdown Automata (NPDA)

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

3. PDAs Building Blocks

3. PDAs Building Blocks

- PDAs have 3 main blocks:
  1. Input Tape
  2. Stack
  3. Control Unit

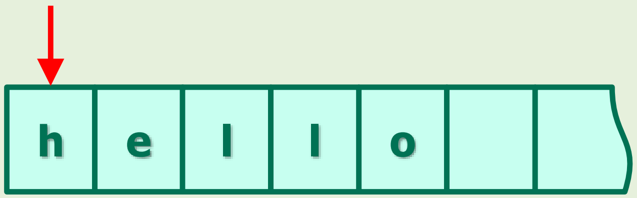


- As usual, we don't need the output part.

- Let's see each block in detail.

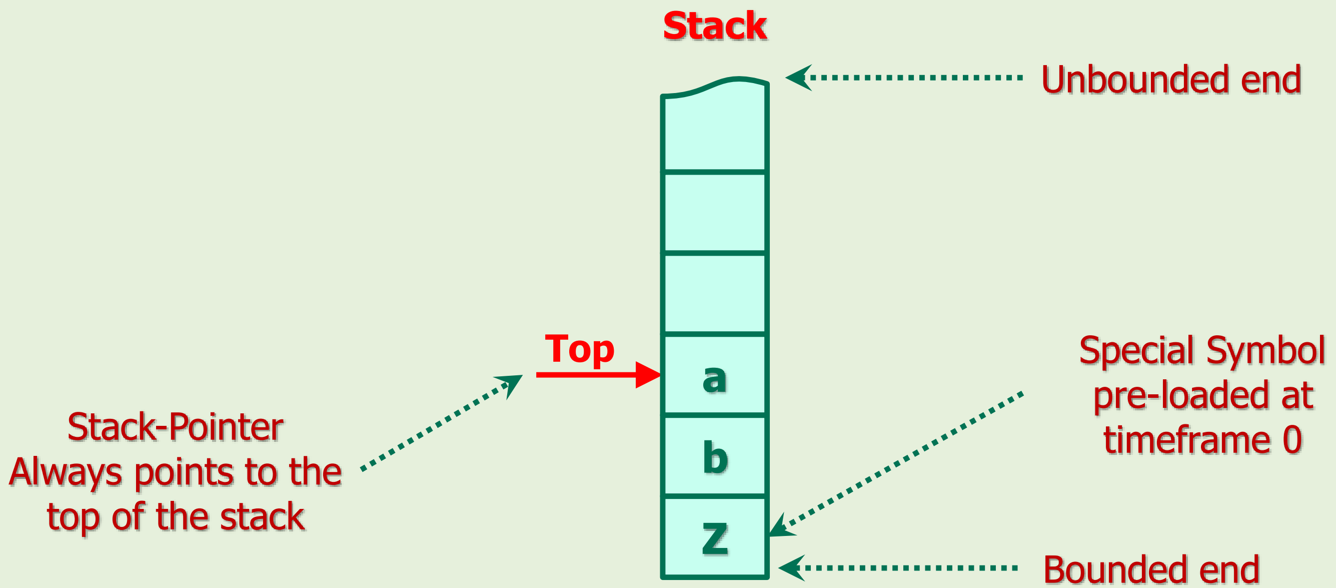
3.1. Input Tape

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



- For the detail, please refer to DFAs' input tape.

3.2. Stack: Structure

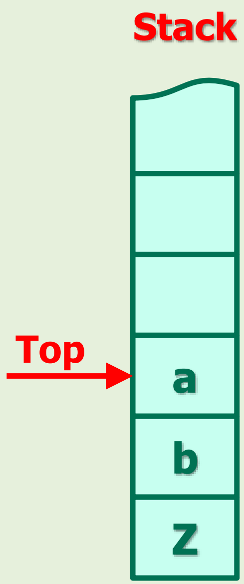


- The special symbol 'Z' is written at the bottom of the stack by the machine at timeframe 0.
- When stack pointer is pointing to 'Z', it means that the stack is empty.

3.2. Stack: How It Works

Operations on Stacks

- Stack works based on
  last in – first out (LIFO) manner.



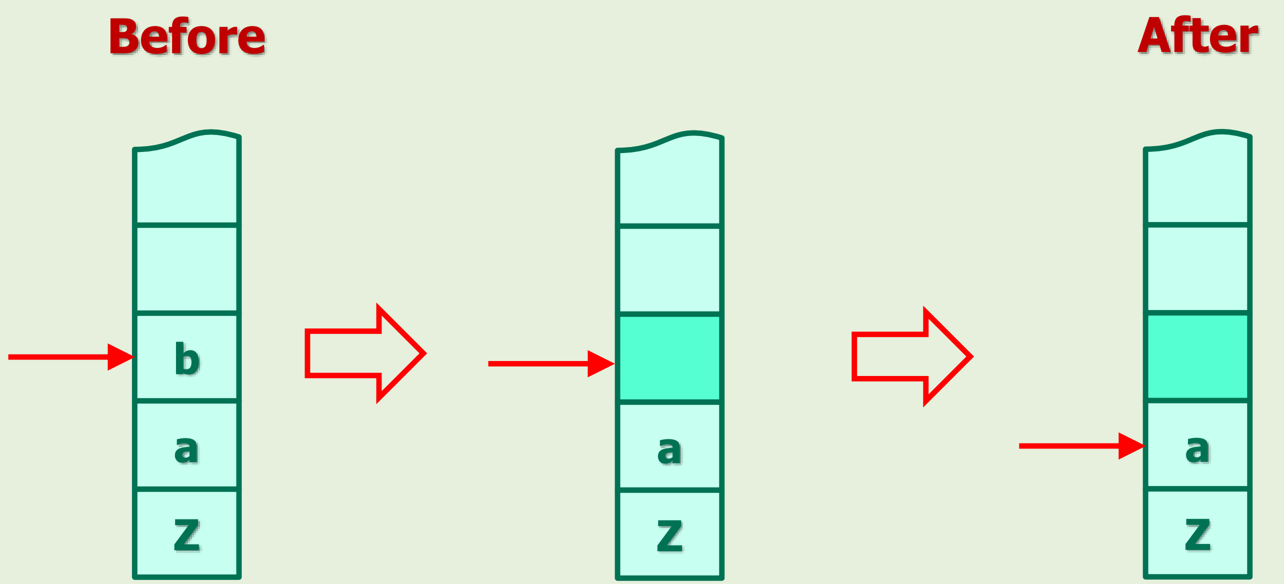
- The basic operations on stacks are "pop" and "push".
  – These operations are similar to what you've learned in data structure course.
- Nevertheless, let's have a quick review of these basic operations.

3.2. Stack: Operations on Stacks

Pop

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

- All of these phases happen during one timeframe.

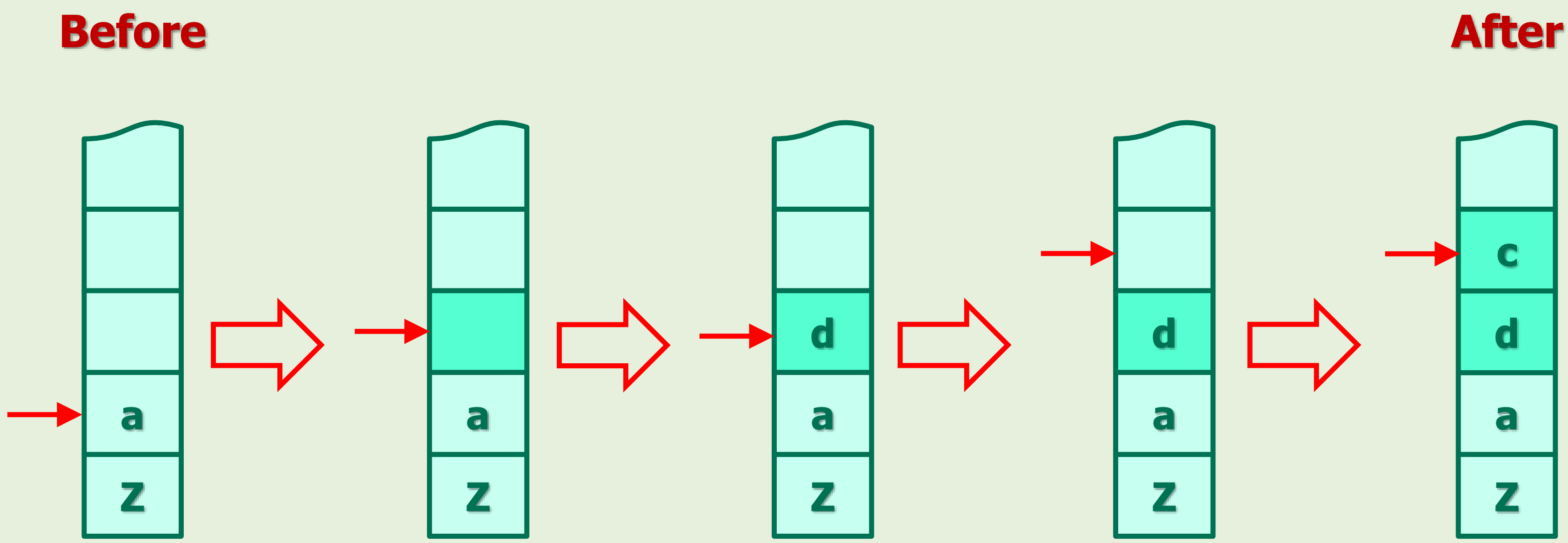


3.2. Stack: Operations on Stacks

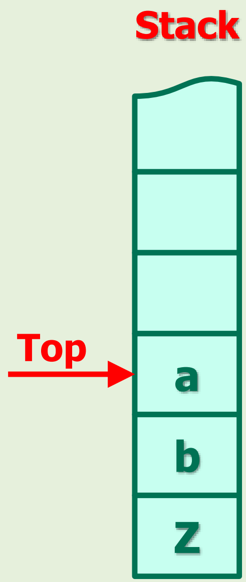
Push

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

- All of these phases happen during one timeframe.



3.2. Stack: Note



1. The "stack alphabet" and the "input tape alphabet" can be totally different. (Will be covered later.)

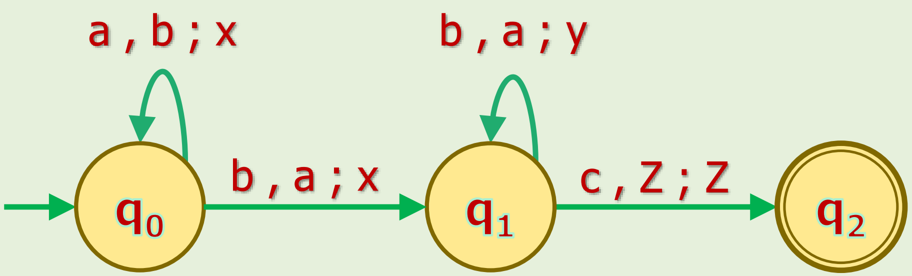
2. If your algorithm requires, you can push 'Z' as many times as you like, and you can pop even the bottom one!

   Therefore, it's designer's responsibility to take care of the contents of the stack.

3.3. Control Unit: Structure

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

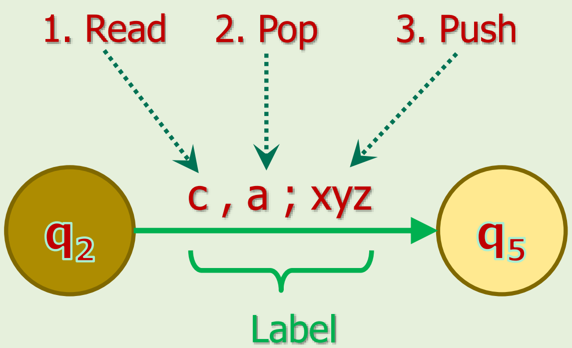
- This is an example of a PDA's transition graph.



- The only difference is how the edges are labeled.

- Let's analyze a transition in detail.

3.3. Control Unit: Labels



- The label 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. 'xyz') that should be pushed into the stack

4. How PDAs Work

4. How PDAs Work

Repeated

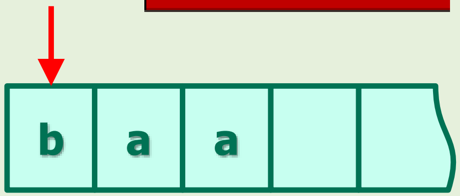
- To understand how PDAs work, we should clearly respond to the following questions:

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?

4.1. PDAs Starting Configuration

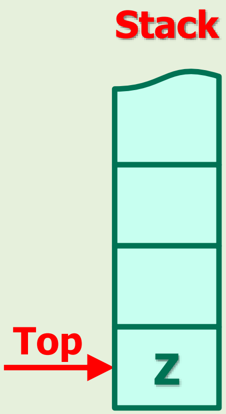
Clock

- 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.

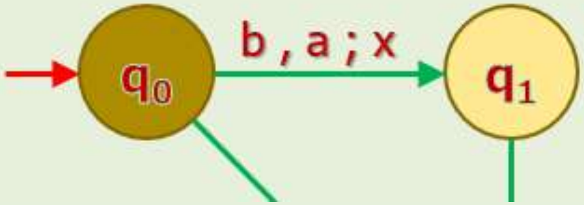


Stack

- 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.



4.2. What Happens During a Timeframe

- During a timeframe, the machine "transits" (aka "moves") from one configuration to another.

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

- Let's first visualize these tasks through some examples.
- Then, we'll summarize them in one slide.

4.2. What Happens During a Timeframe

Transition Examples

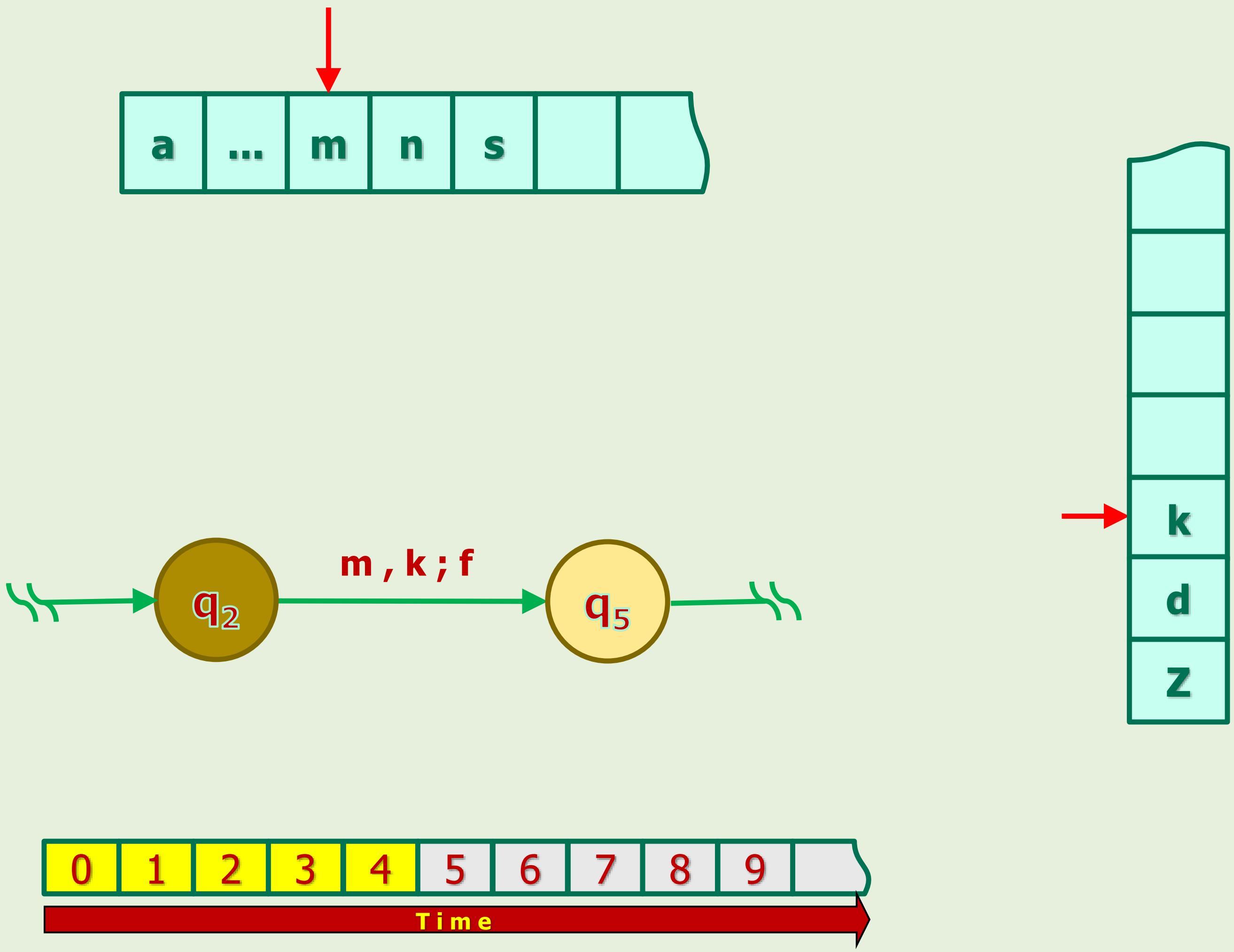
Transition Examples

- The next 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?

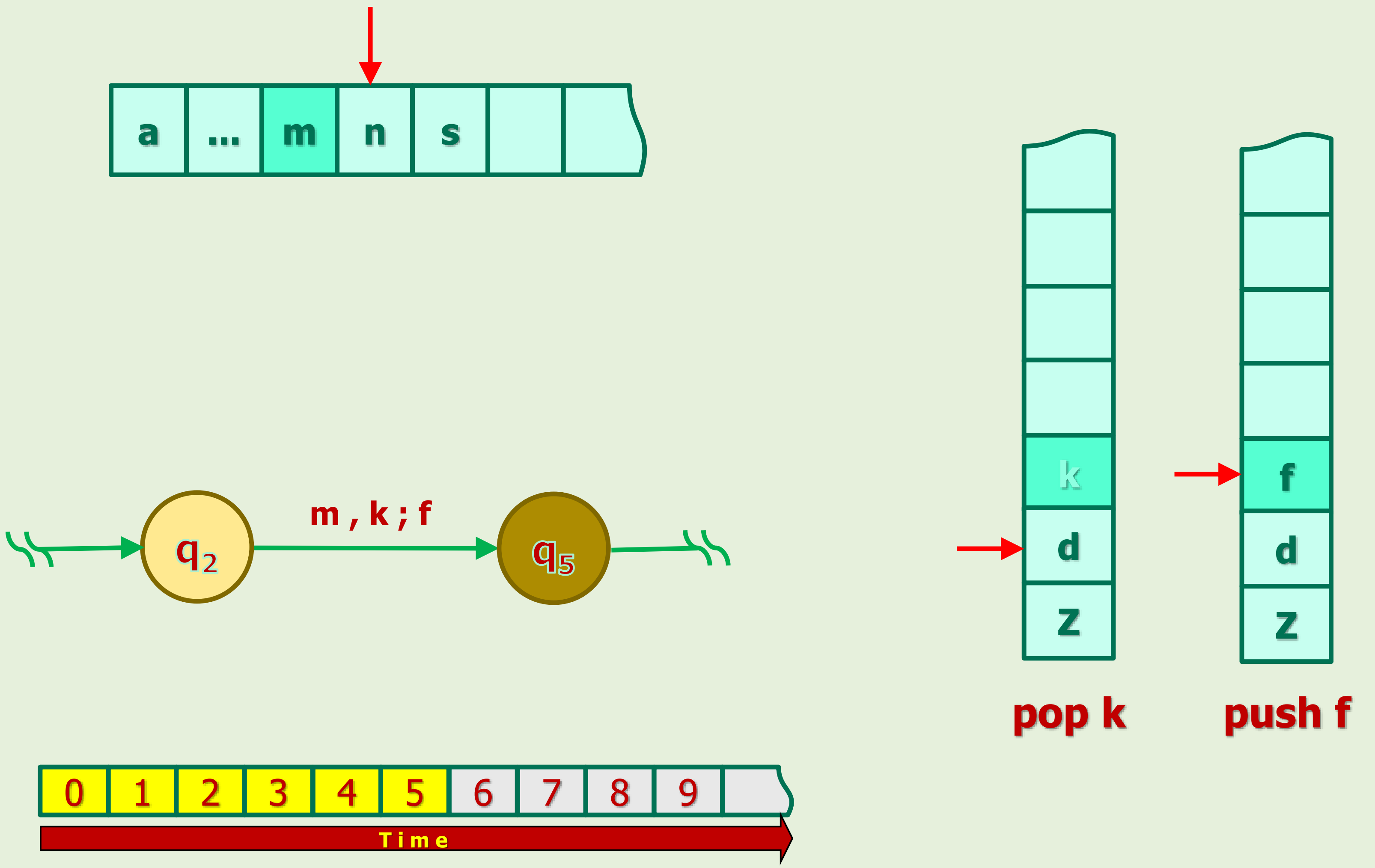
Transition Example: Altering Stack Data

Example 1



Transition Example: Altering Stack Data

Example 1 (cont'd)



4.2. What Happens During a Timeframe

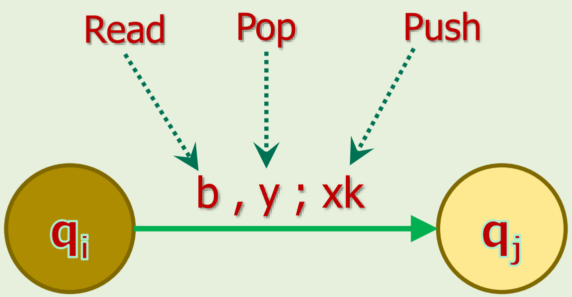
Rough Summary of Transition

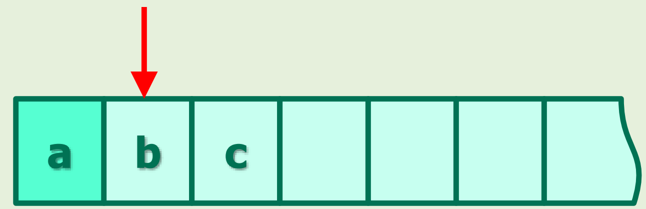
- The following tasks happen during a timeframe:
  - The precise definitions would come later.

1. A symbol at which the read-head (cursor) 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 PDAs?

PDAs' Logic of Transitions





If (Condition)

in q_i

AND

the input symbol is 'b'

AND

the top of the stack is 'y'

Then (Operation)

consume 'b'

AND

pop 'y'

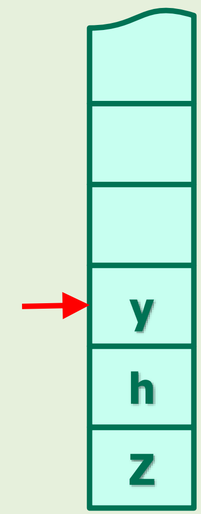
AND

push 'xk'

AND

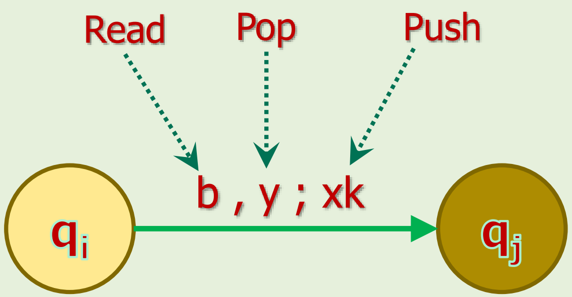
transit to q_j

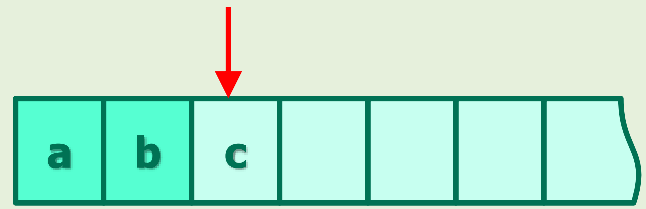
Stack

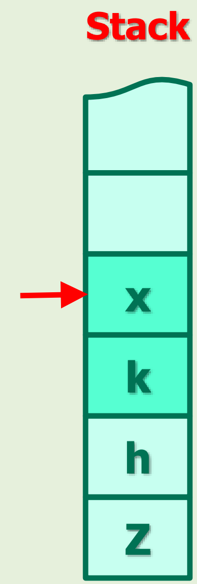


How does the machine look like after this transition?

PDAs' Logic of Transitions





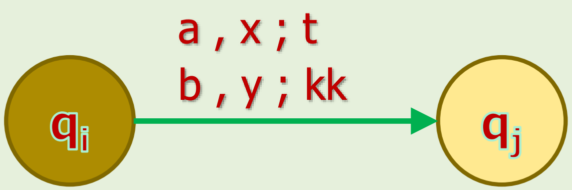


After the previous transition ...

- You might ask: what if the input is not 'b'?
- Or what if the top of the stack is not 'y'?
- Good questions! We'll get back to this question shortly.

Multiple Labels

- A transition might have multiple labels.
- In that case, we stack them over the edge.



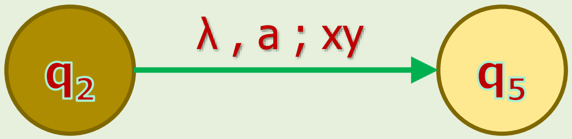
- Note that there is an OR between them.
- It means, in either condition, the machine transits and follows the appropriate label's operations.

Relaxing the Conditions and Operations by λ

- If we put λ in any part of the labels, it means "no condition" or "no action" in that part.
- So, by using 'λ', we can relax the conditions and the operations.
- Note that this is the 3rd usage of λ.
- We'll define λ-transitions when we discuss about NPDAs.
- Let's see some examples.

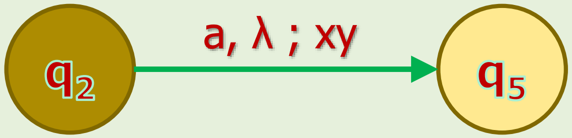
Relaxing the Conditions and Operations by λ

Example 2



- 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 input symbol!

Example 3



- 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 and Operations by λ

Example 4

- 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 5

- 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 input symbol AND do NOT push anything!

Relaxing the Conditions and Operations by λ

Example 6

- 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!

Notes

1. So far, we did not have NPDA because there was no multiple choice situations.
2. We can 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.

4.2. What Happens During a Timeframe

Summary of Transition

- The following tasks happen during a timeframe.

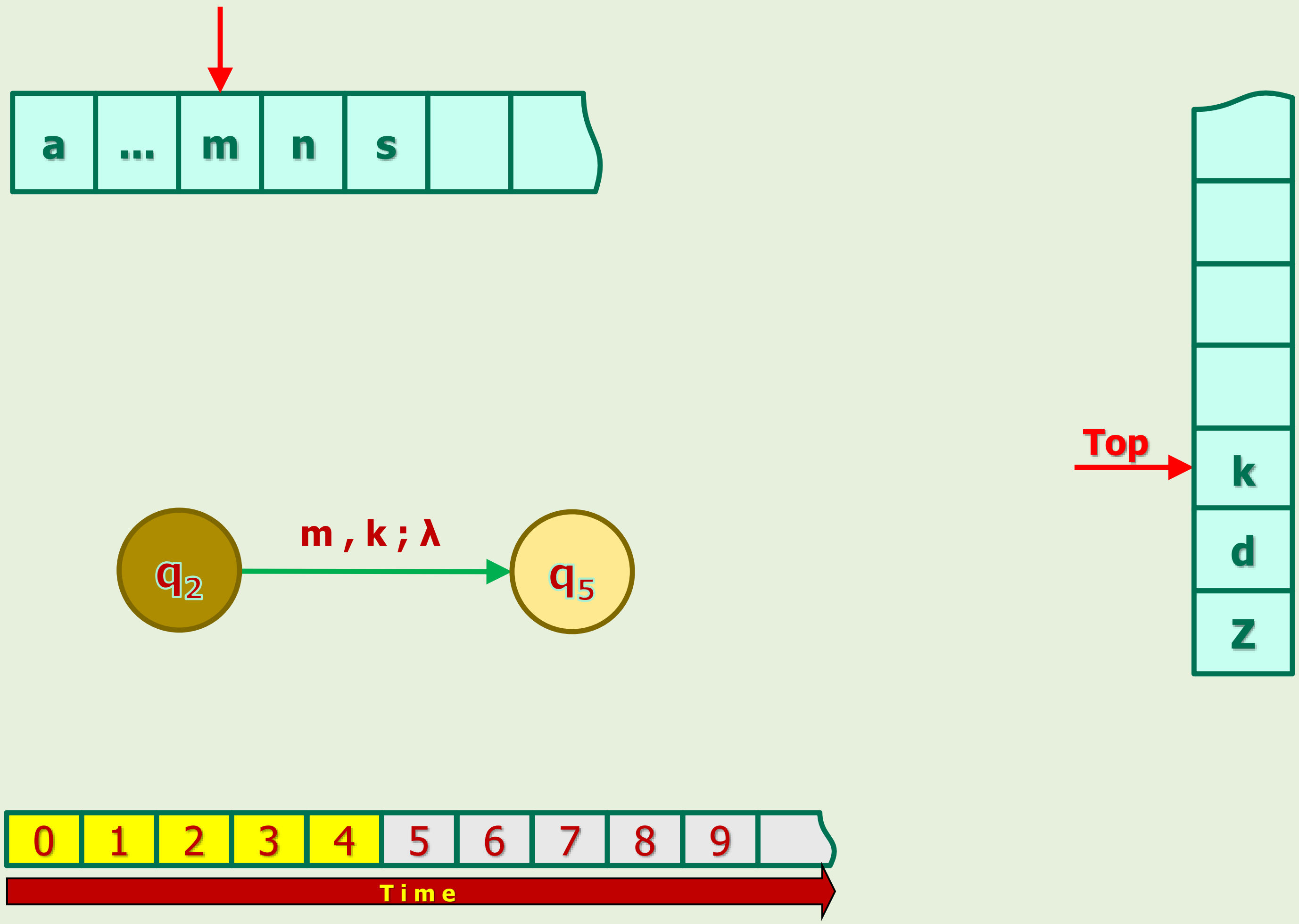
1. Zero or one symbol at which the cursor is pointing, is consumed.
2. Zero or one symbol is popped from the stack.
3. A string (could be empty) is pushed into the stack.
4. The control unit makes its move based on the "logic of the transition". (covered a few slides before)

- Now let's see some more transition examples.

Transition Examples

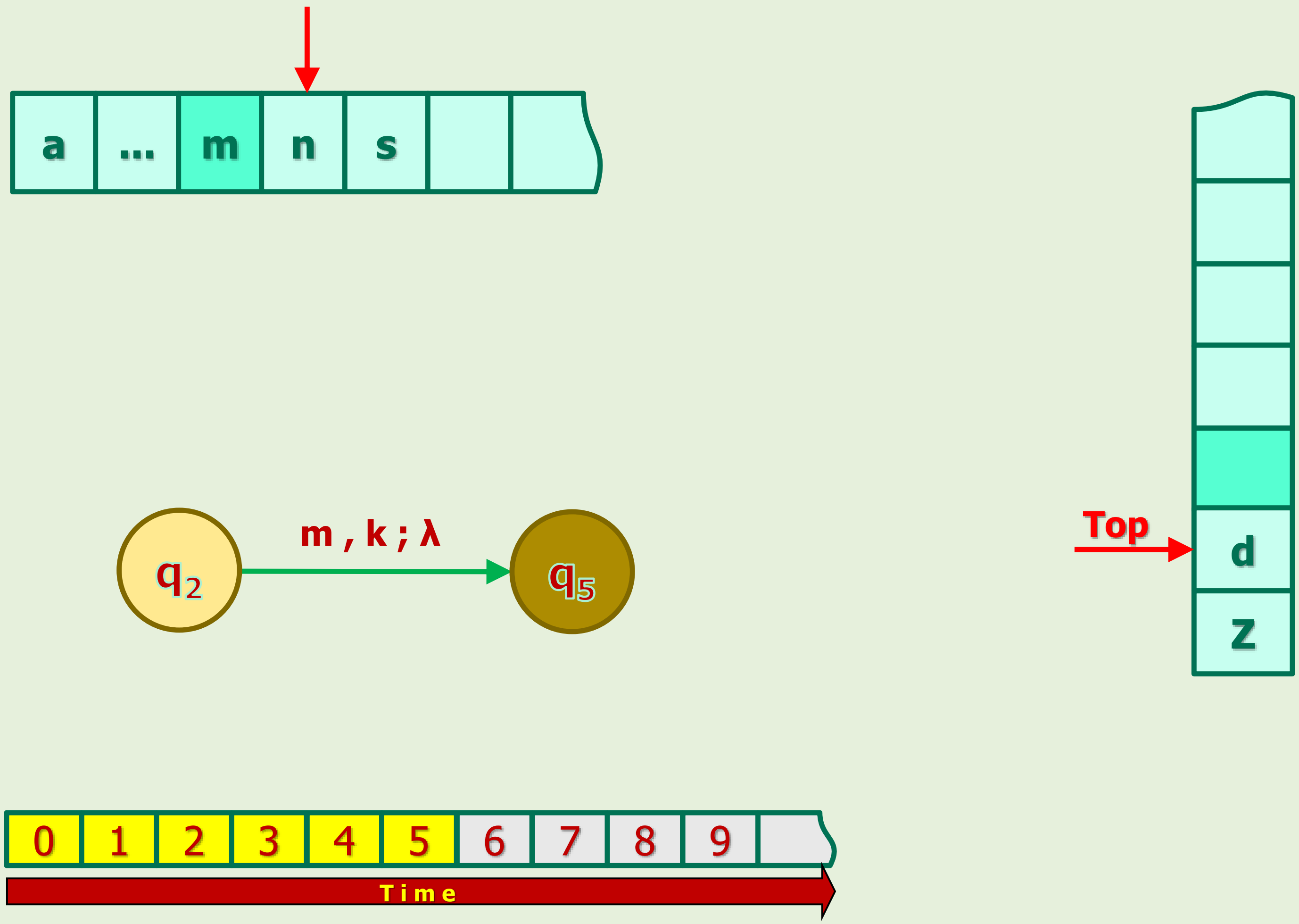
Transition Examples: Popping Stack Data

Example 7



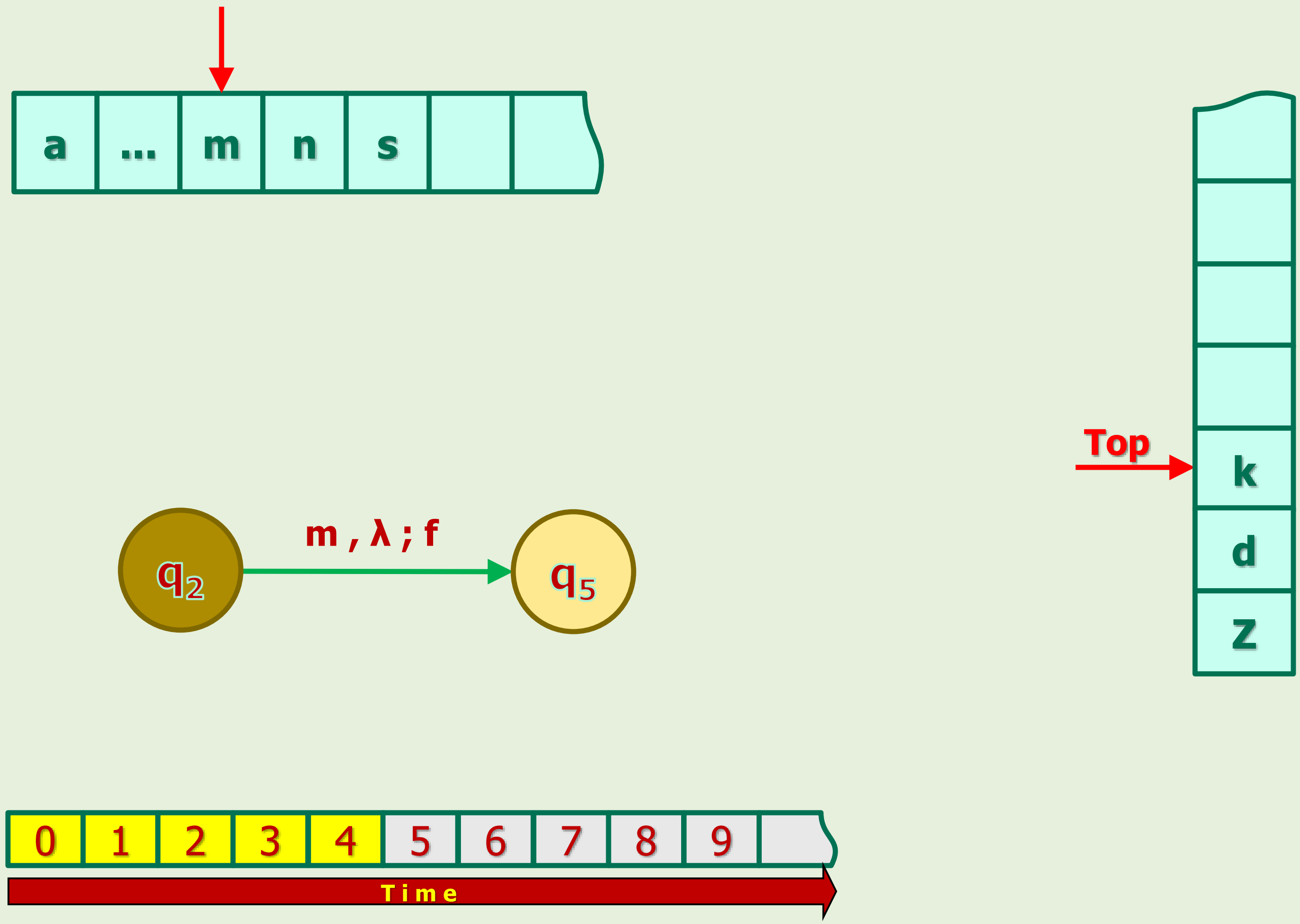
Transition Examples: Popping Stack Data

Example 7 (cont'd)



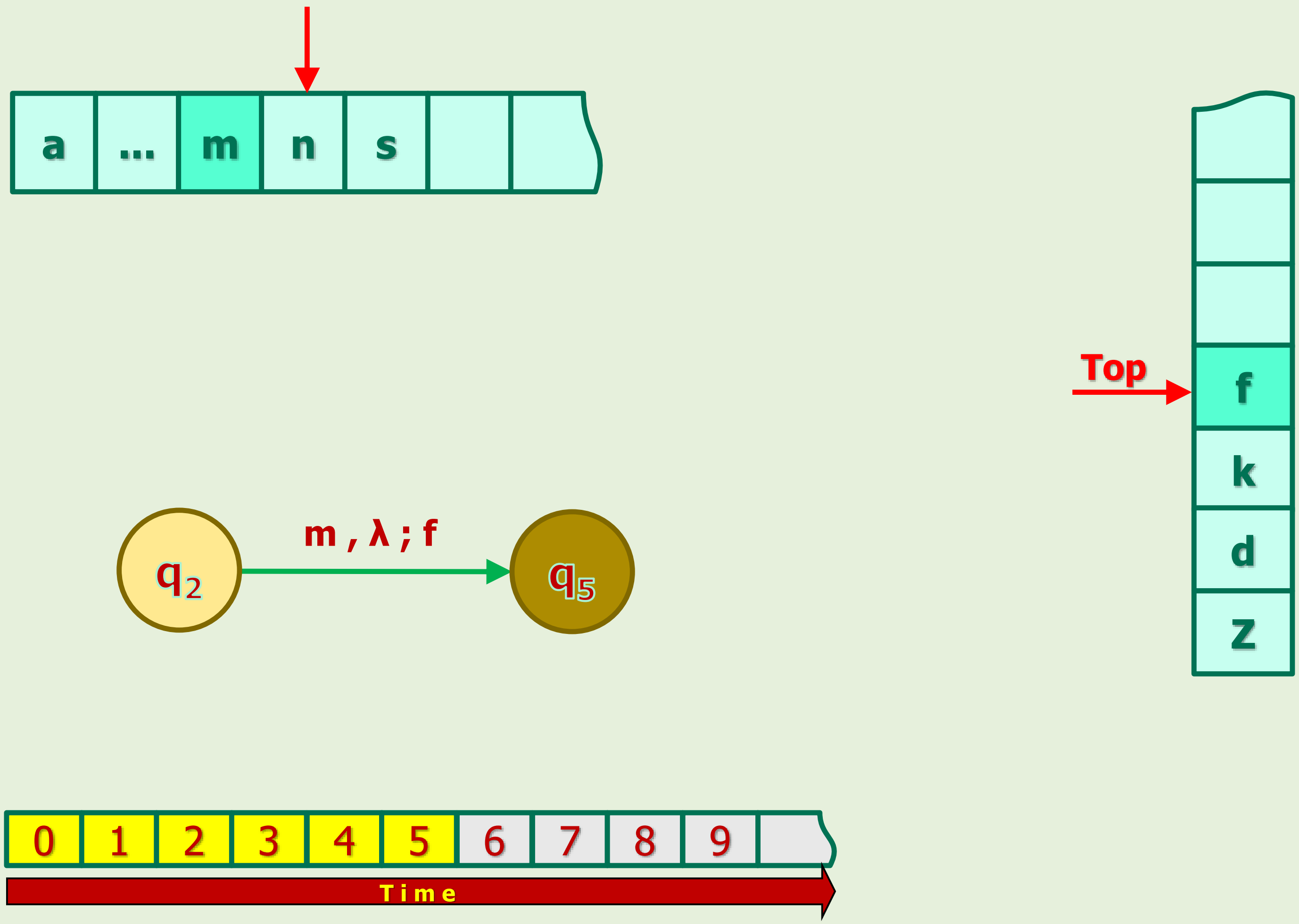
Transition Examples: Pushing Data into Stack (1)

Example 8



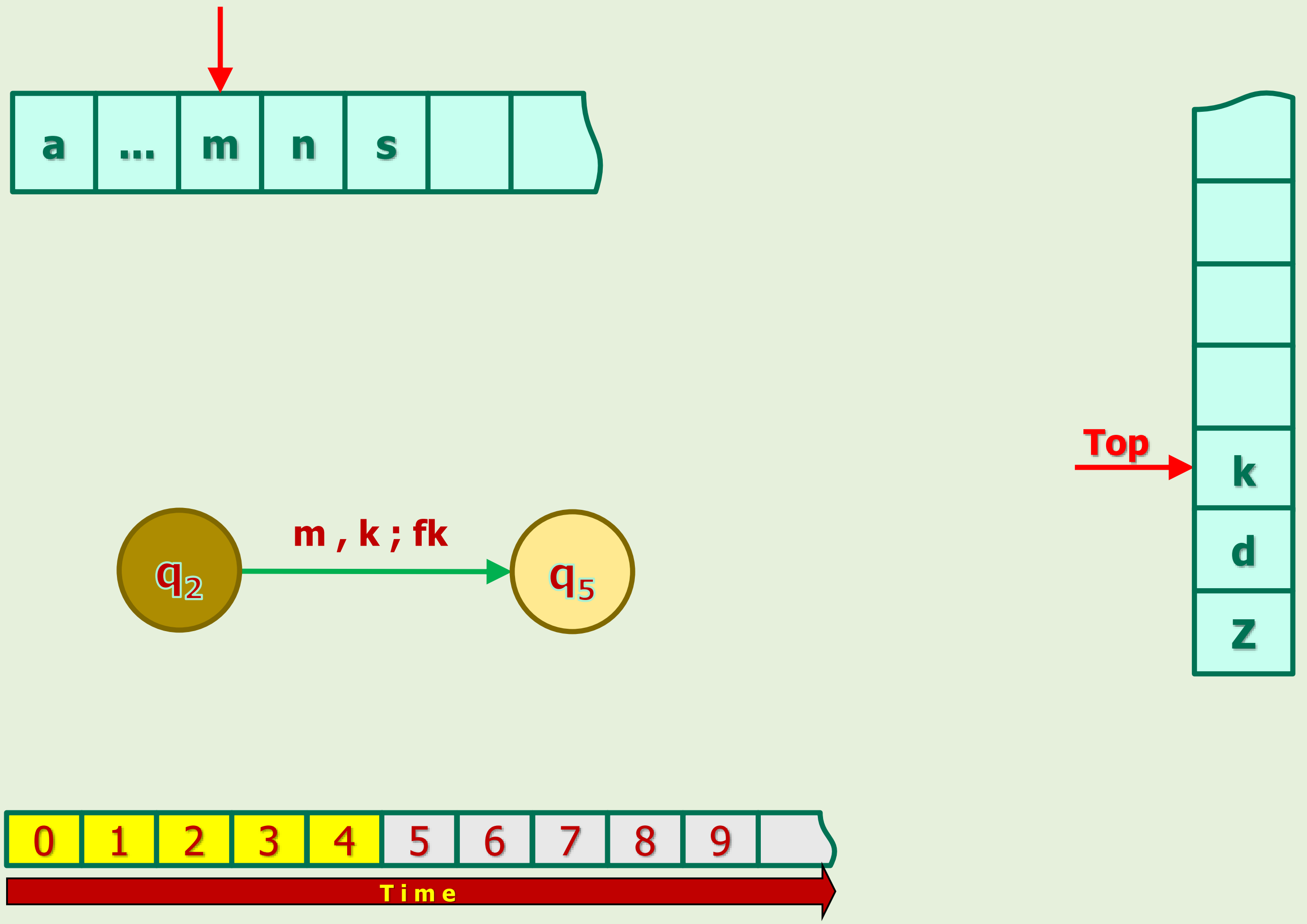
Transition Examples: Pushing Data into Stack (1)

Example 8 (cont'd)



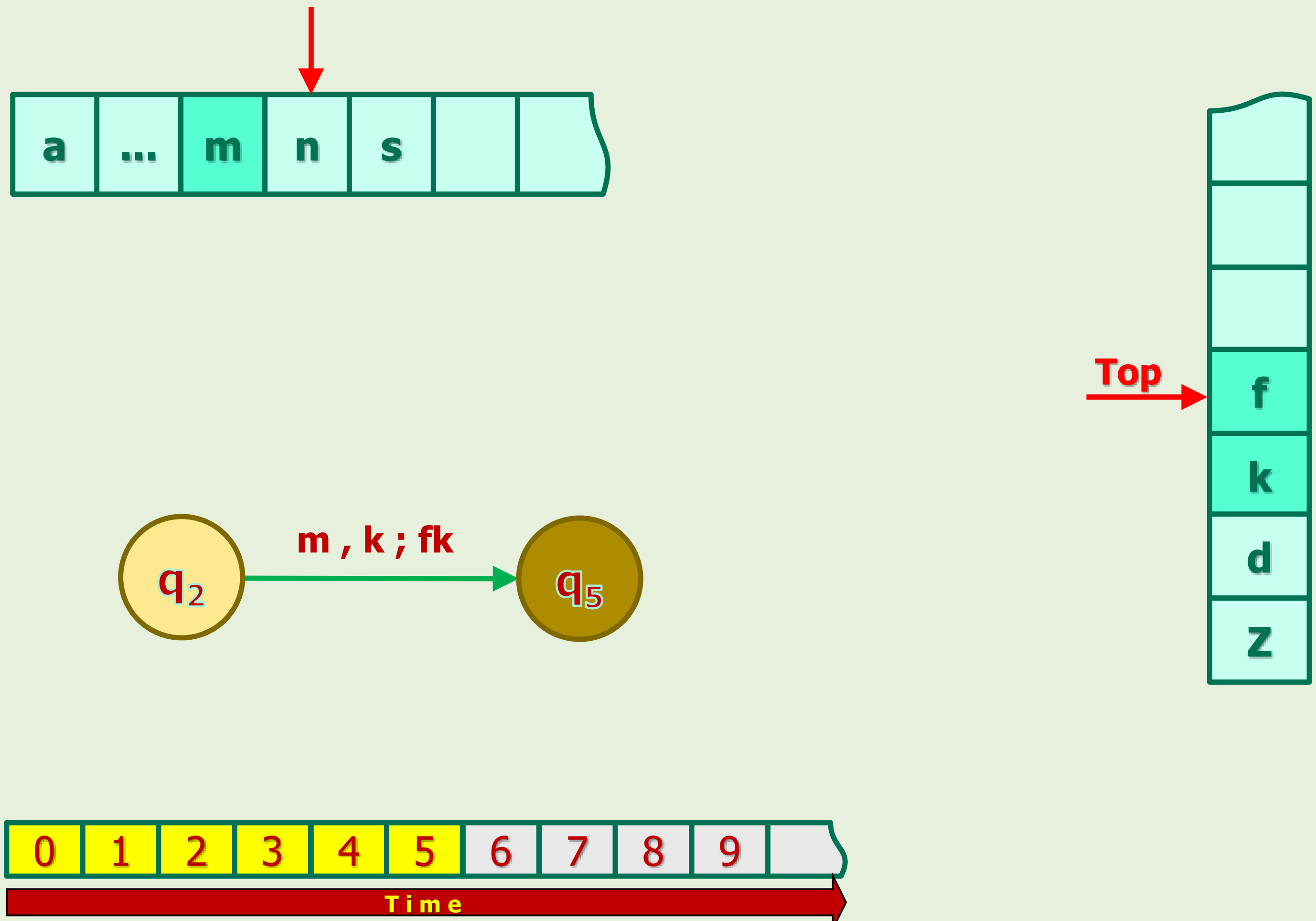
Transition Examples: Pushing Data into Stack (2)

Example 9



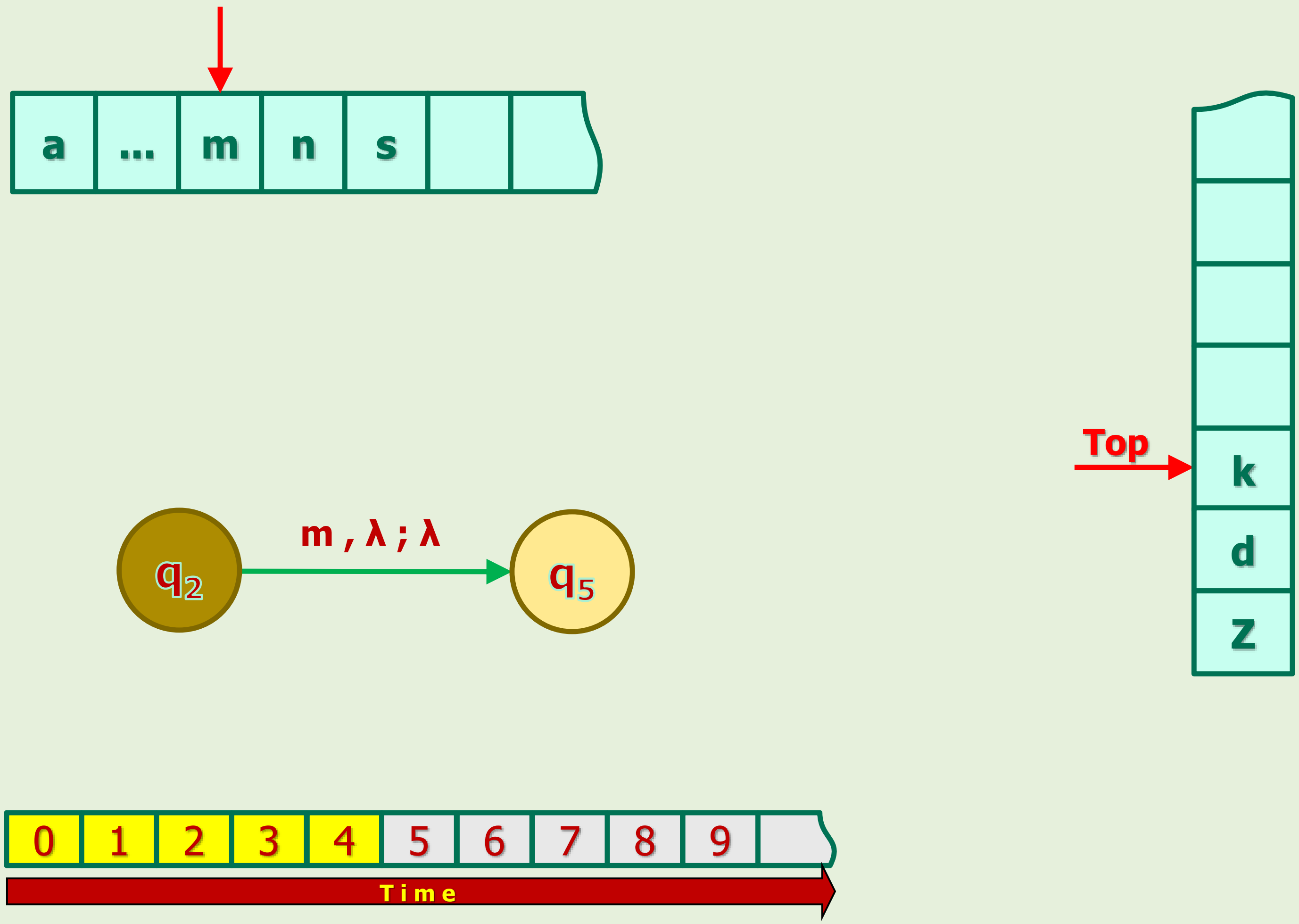
Transition Examples: Pushing Data into Stack (2)

Example 9 (cont'd)



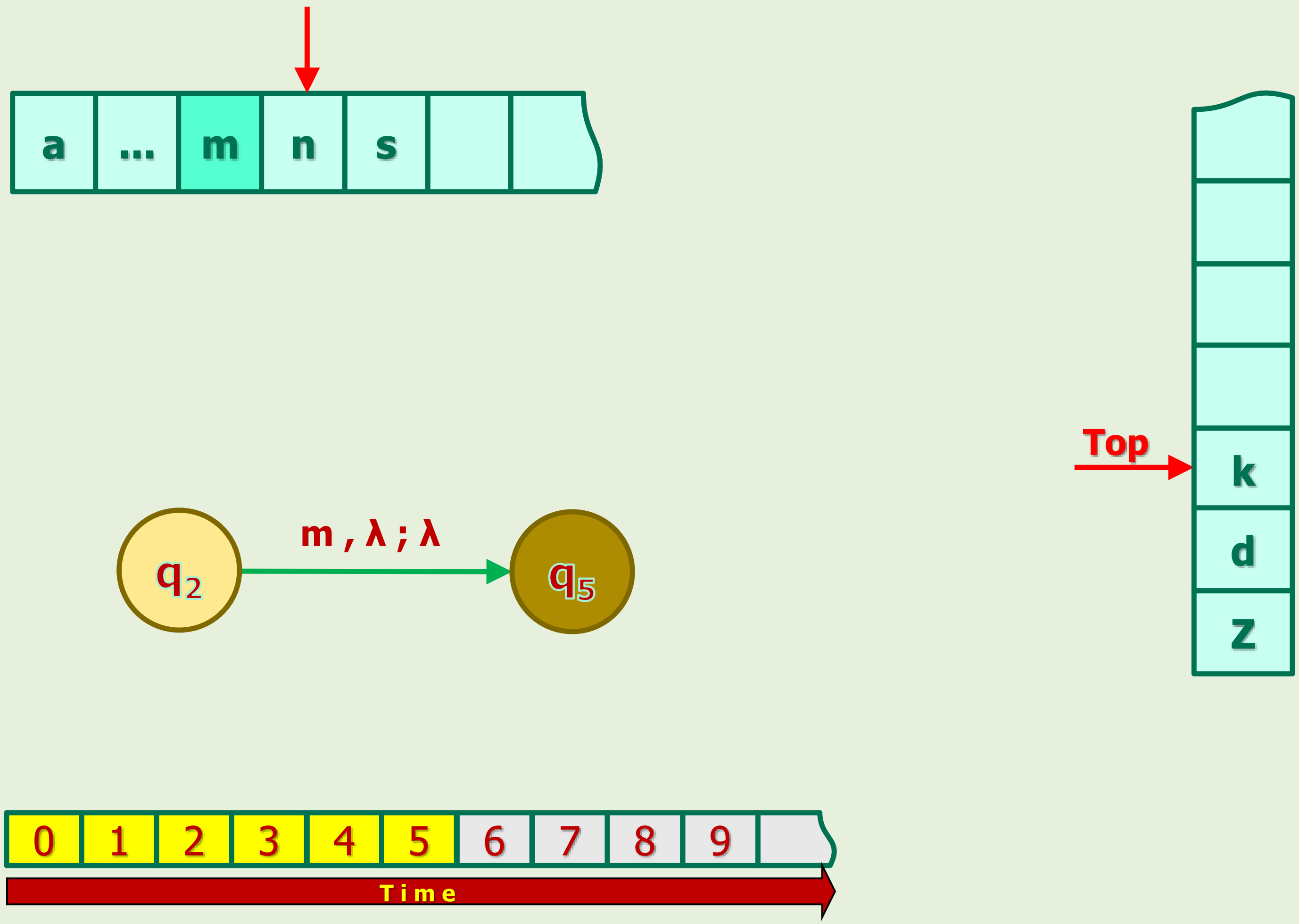
Transition Examples: No Action on Stack

Example 10



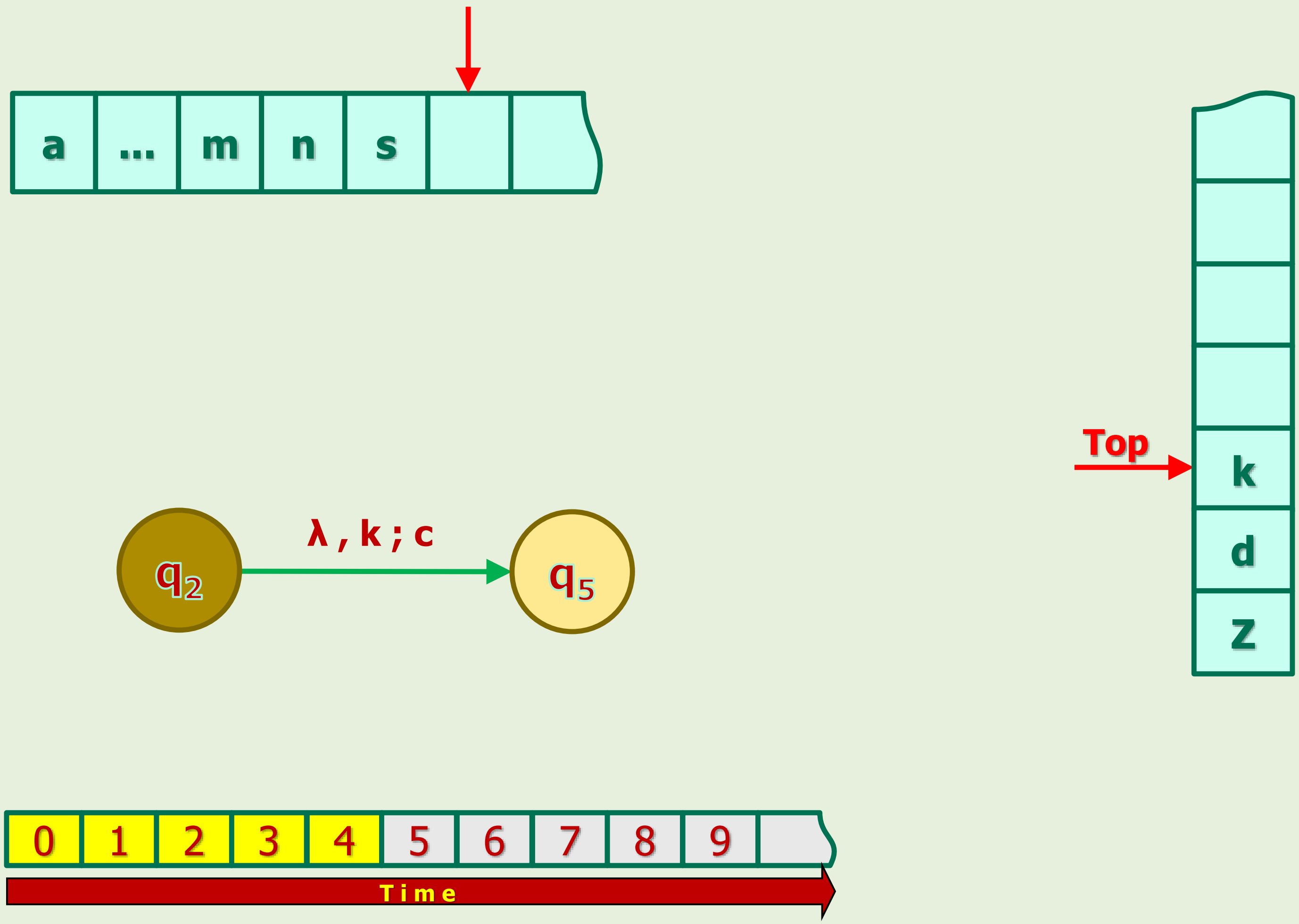
Transition Examples: No Action on Stack

Example 10 (cont'd)



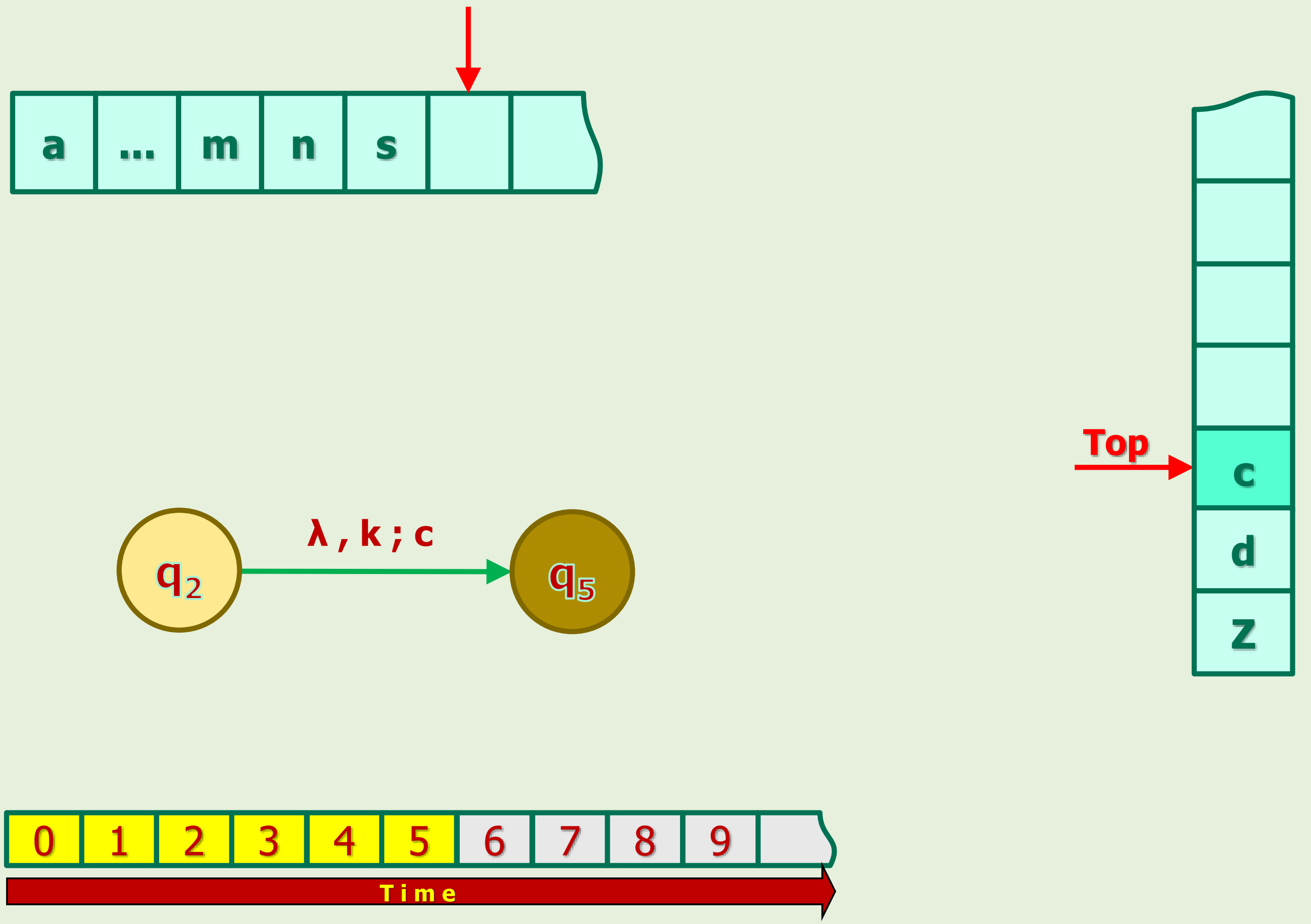
Transition Examples: No Action on Input Tape

Example 11 (Note the difference with DFAs/NFAs)



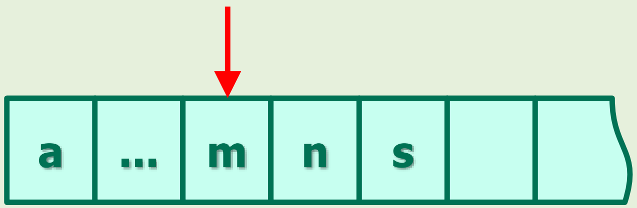
Transition Examples: No Action on Input Tape

Example 11 (cont'd)

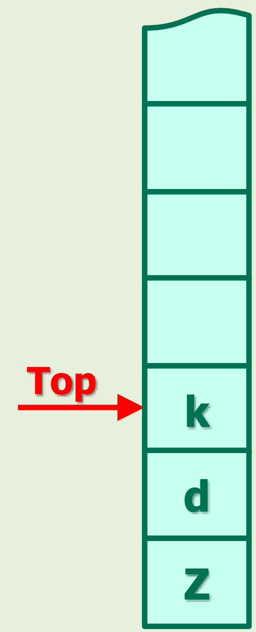


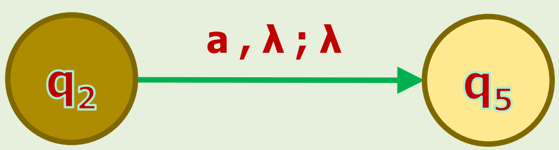
Transition Examples: No Transition

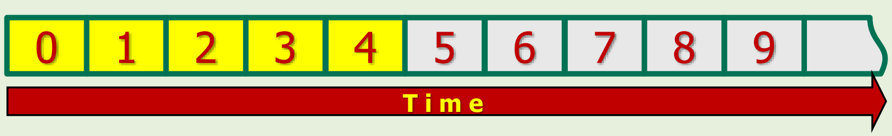
Example 12



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

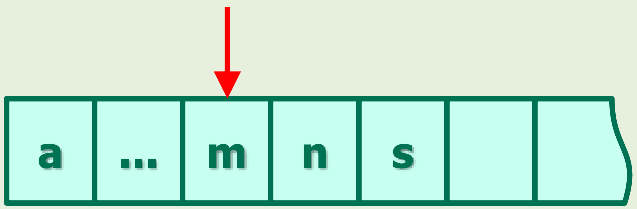




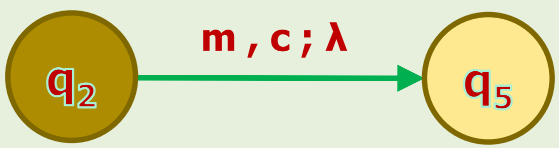


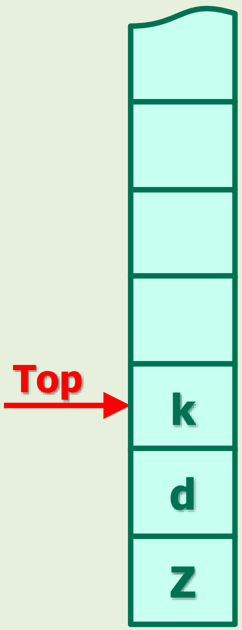
Transition Examples: No Transition

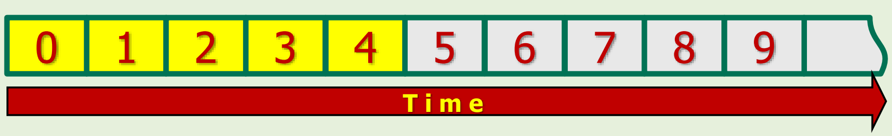
Example 13



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







4.3. When PDAs Halt

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

  "All input symbols are consumed."

- ... was NOT sufficient for PDAs to halt.

  PDAs halt when the next transition conditions are NOT satisfied.

- Transition Conditions = input symbol + top of the stack

Halt Logical Representation

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

4.4. How PDAs Accept/Reject Strings

Logical Representation of Accepting Strings By One Process

PDAs accept a string w. ≡ **a**

IFF

They halt. ≡ **h**

AND

All symbols of w are consumed. ≡ **c**

AND

They are in an accepting (final) state. ≡ **f**

$$(\mathbf{h} \land \mathbf{c} \land \mathbf{f}) \leftrightarrow \mathbf{a}$$

- Note that if we have NPDAs, that are nondeterministic., then ...

  ... they might have several processes.
- So, the above conditions are only for one process to accept a string.

4.4. How PDAs Accept/Reject Strings

Logical Representation of Rejecting Strings By One Process

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

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

Translation

PDAs reject a string w. ≡ ~**a**

IFF

They do NOT halt. ≡ ~**h**

OR

At least one symbol of w is NOT consumed. ≡ ~**c**

OR

They are NOT in an accepting (final) state. ≡ ~**f**

4.4. How PDAs Accept/Reject Strings: Notes

1. The final contents of the stack is NOT important in accepting or rejecting a string.
   - Because stack is in fact a workspace for drafting (like scratch paper).
   - It is a place to store the middle results of the computation.
2. JFLAP has an option to accept a string when the stack is empty!
   - We do NOT use this option.

- Now let's see PDAs in action!
- To show the power of PDAs, we'll design PDAs 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