# 15-251

# Great Theoretical Ideas in Computer Science

# Lecture 21 - Dan Kilgallin

# Turing Machines

# Goals

- Describe the nature of computation
- Mathematically formalize the behavior of a program running on a computer system
- Compare the capabilities of different programming languages

# Turing Machines

- What can we do to make a DFA better?
- Idea: Give it some memory!
- How much memory?
- Infinite!
- What kind of memory? Sequential access (e. g. CD, hard drive) or RAM?
- An infinite amount of RAM requires remembering really big numbers and needlessly complicates proofs

# Turing Machines: Definition

- A Turing Machine consists of
  - A DFA to store the machine state, called the controller
  - An array that is infinitely long in both directions, called the tape
  - A pointer to some particular cell in the array, called the head

# Operation

- The head begins at cell 0 on the tape
- The DFA begins in some initial state
- Symbols from some alphabet are written on a finite portion of the tape, beginning at cell 1

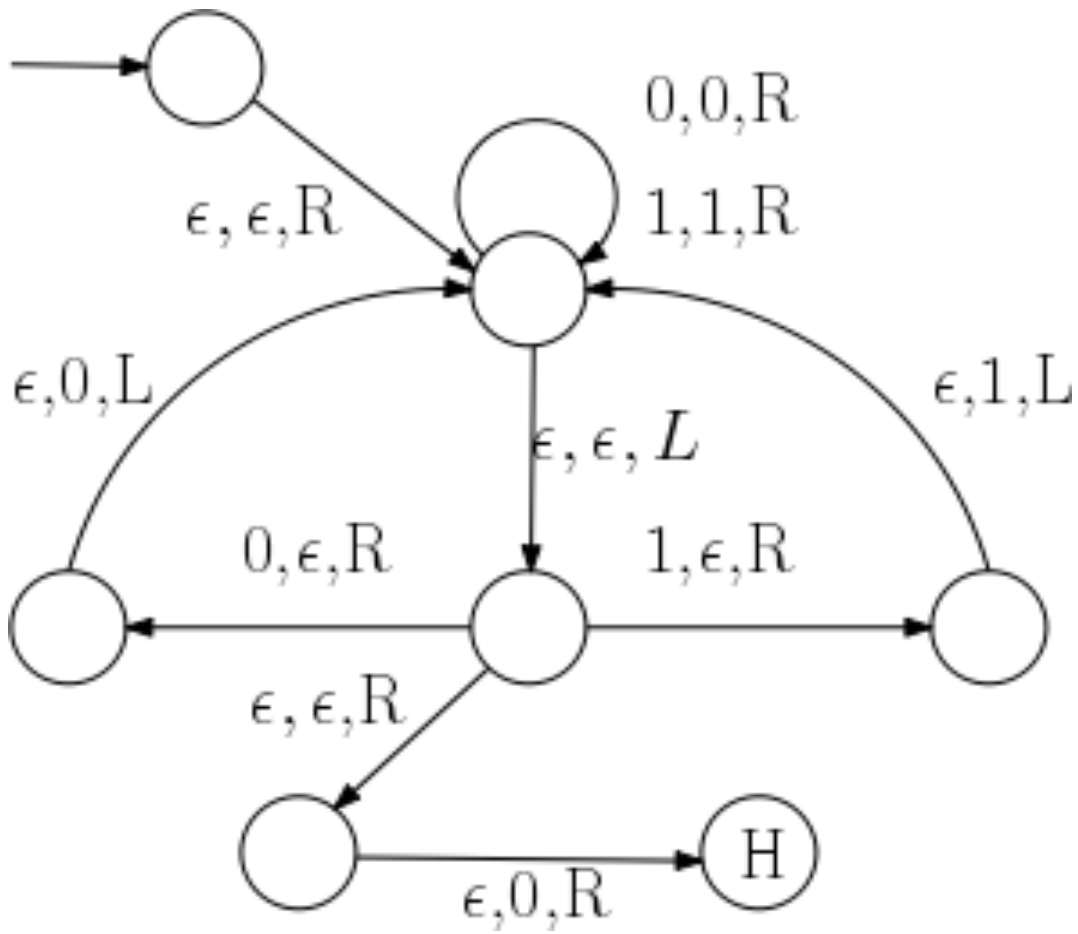
# Operation

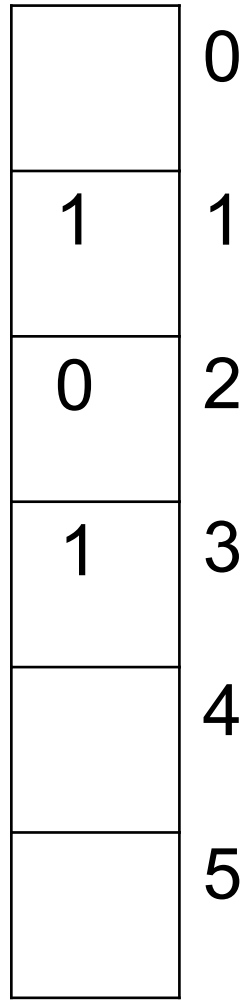
- At each step, the machine uses as input:
  - The state of the controller
  - The symbol written on the cell which the head is at
- Using this, the machine will:
  - Transition to some state in the DFA (possibly the same state)
  - Write some symbol (possibly the same symbol) on the tape
  - Move the head left or right one cell

# Two Flavors!

- The machine will stop computation if the DFA enters one of a pre-defined set of "halting" states
- A "decision" machine will, if it reaches a halting state, output either "Yes" or "No" (i.e. there are "accepting" and "rejecting" halt states)
- A "function" machine will, if it reaches a halting state, have some string of symbols written on the tape. This string of symbols is the output of the function

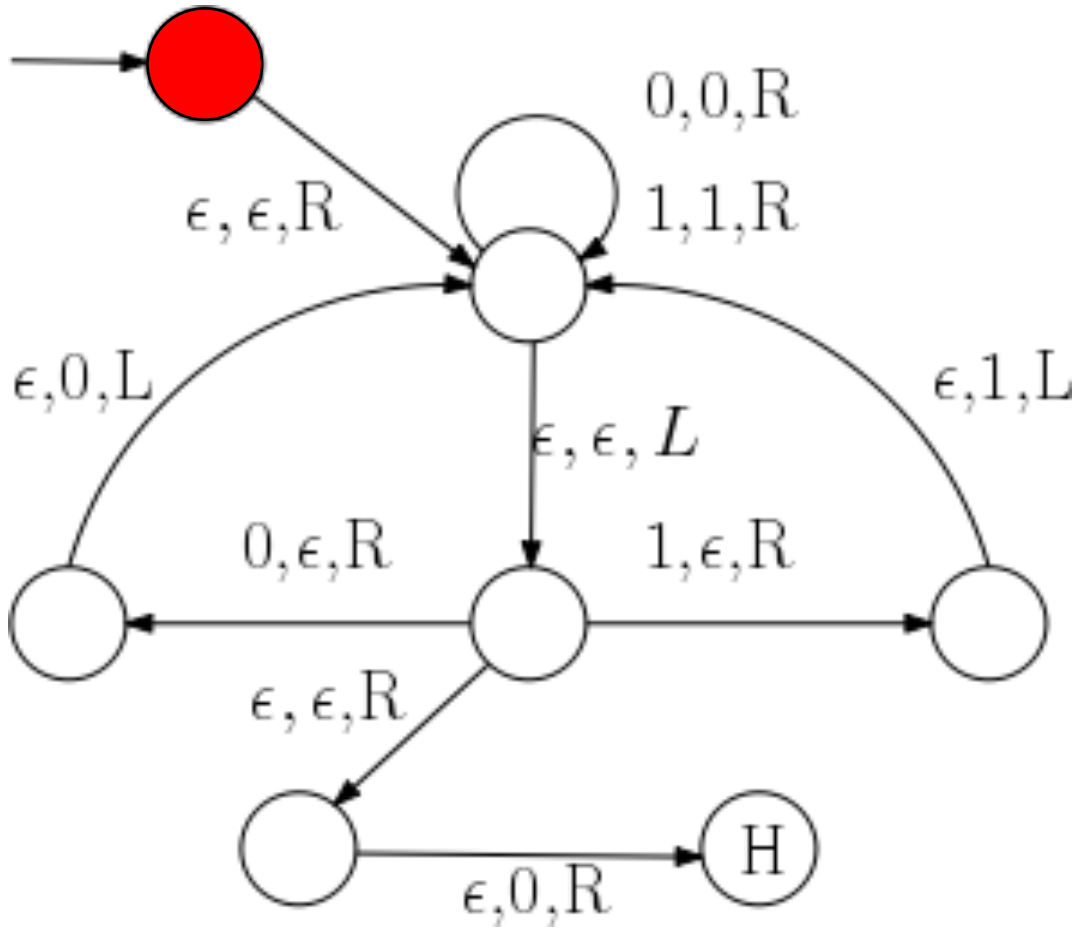
# Example: Doubling a Number

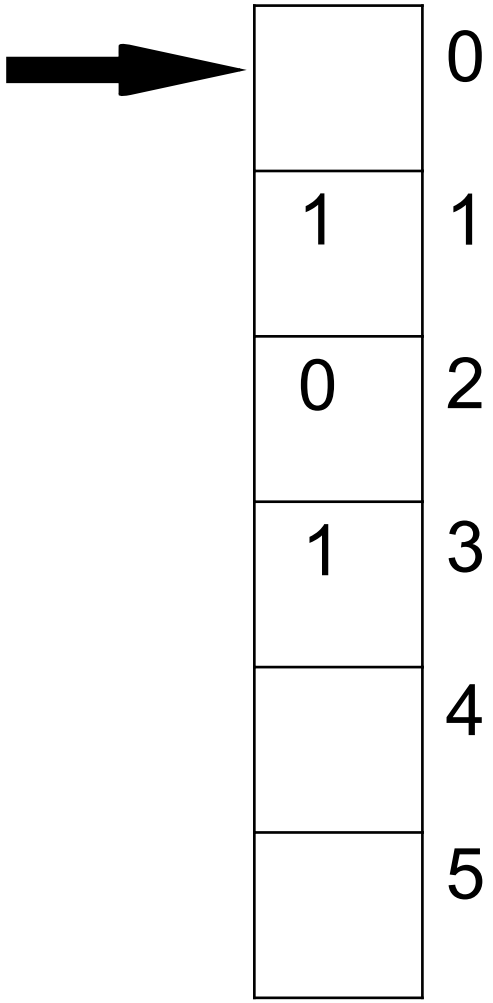


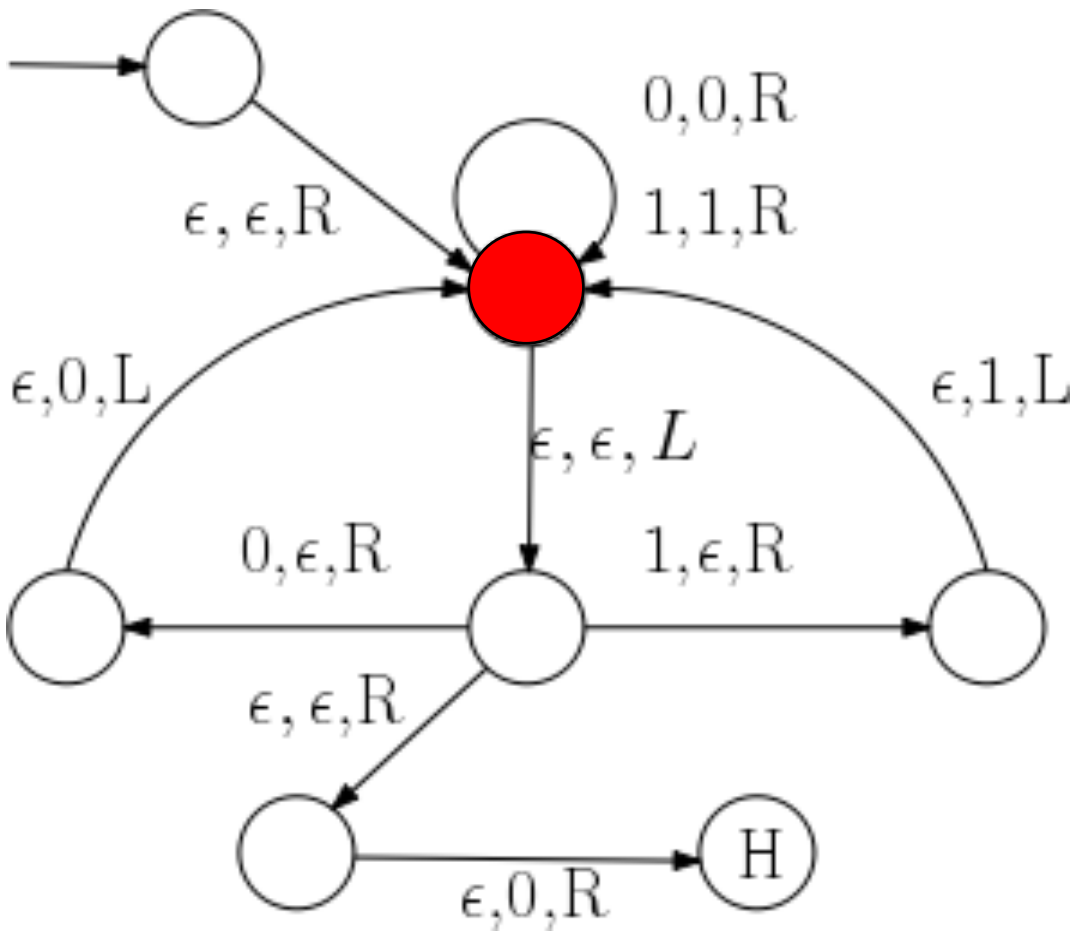

Notation: (old symbol, new symbol, direction to move)
"epsilon" denotes empty cell, "H" denotes halting state(s)

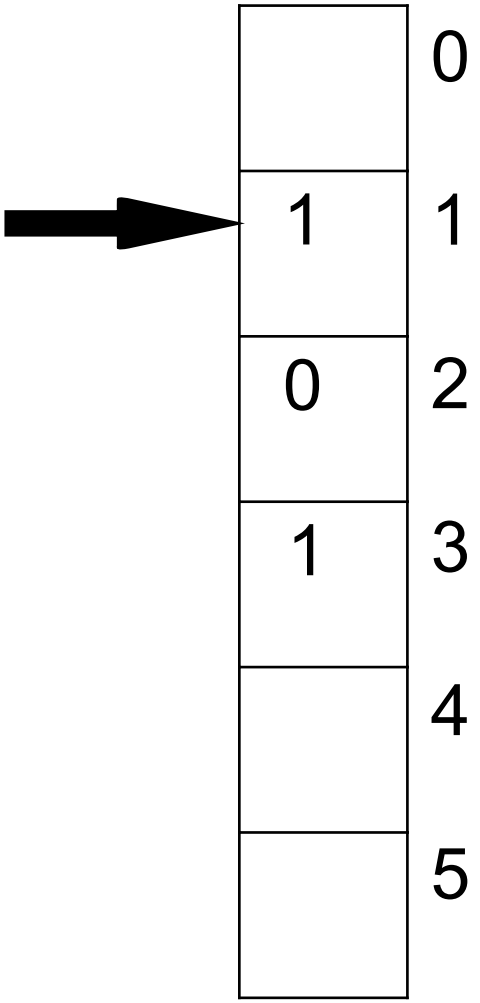
# Example: Doubling a Number



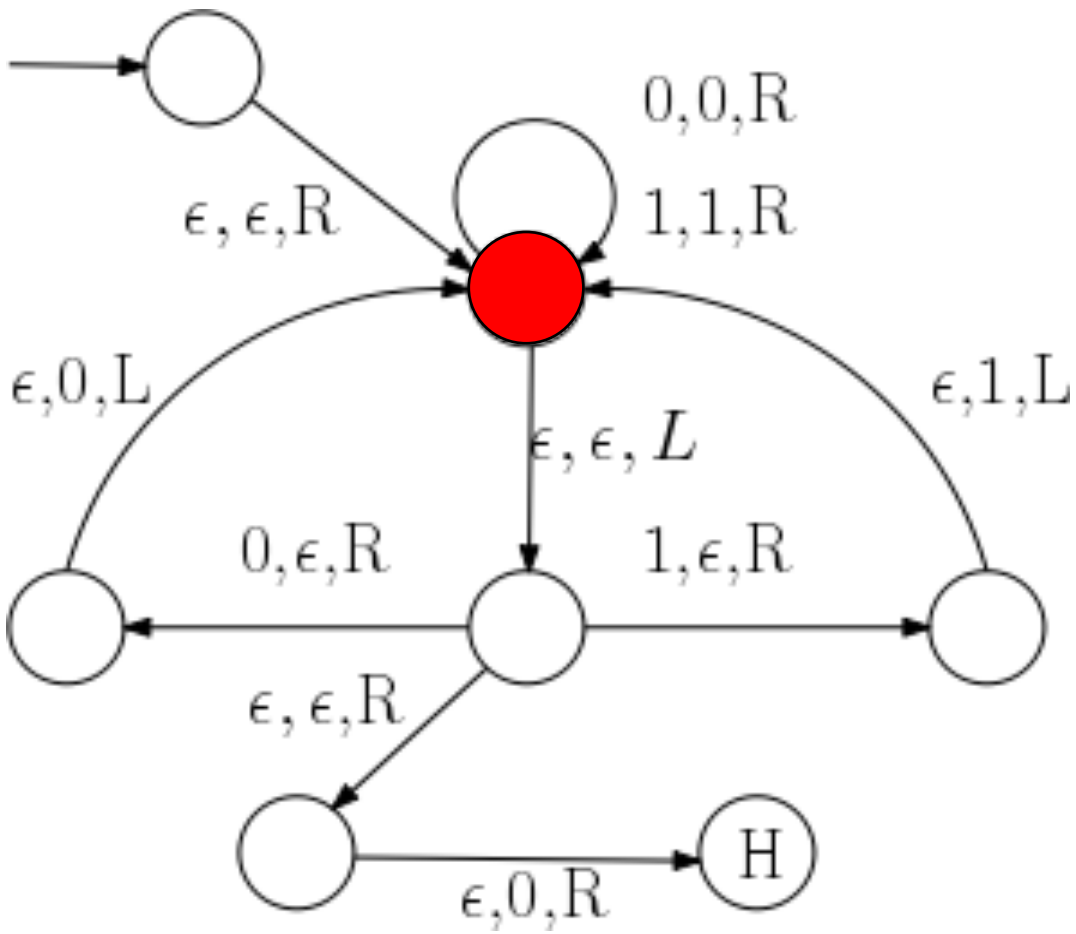



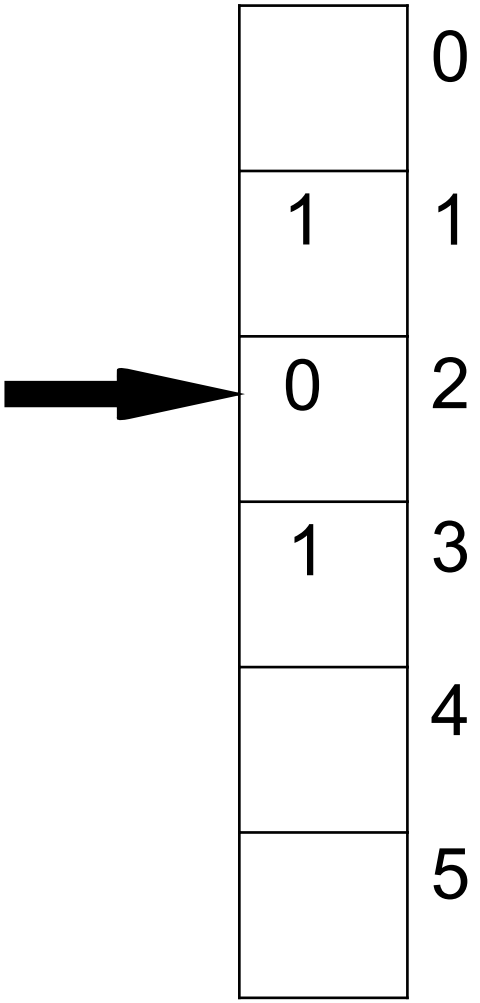
# Example: Doubling a Number



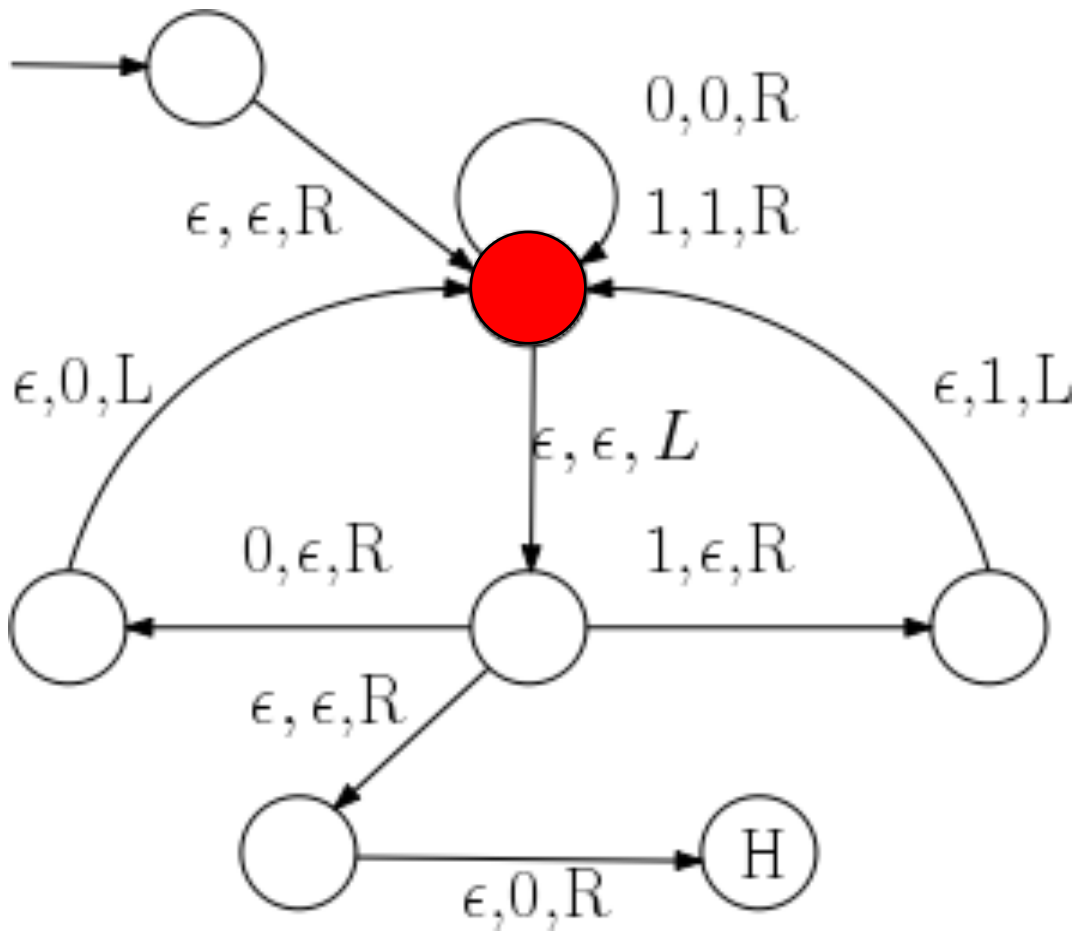



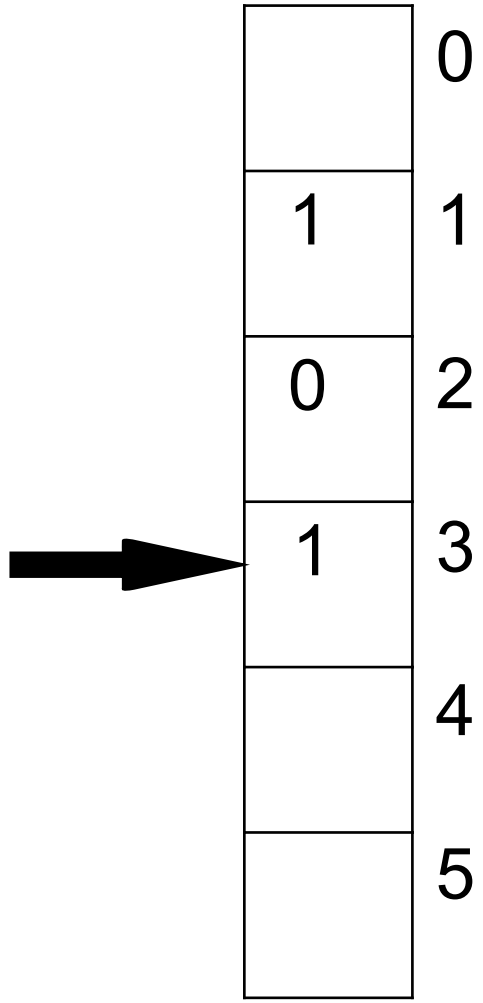
# Example: Doubling a Number



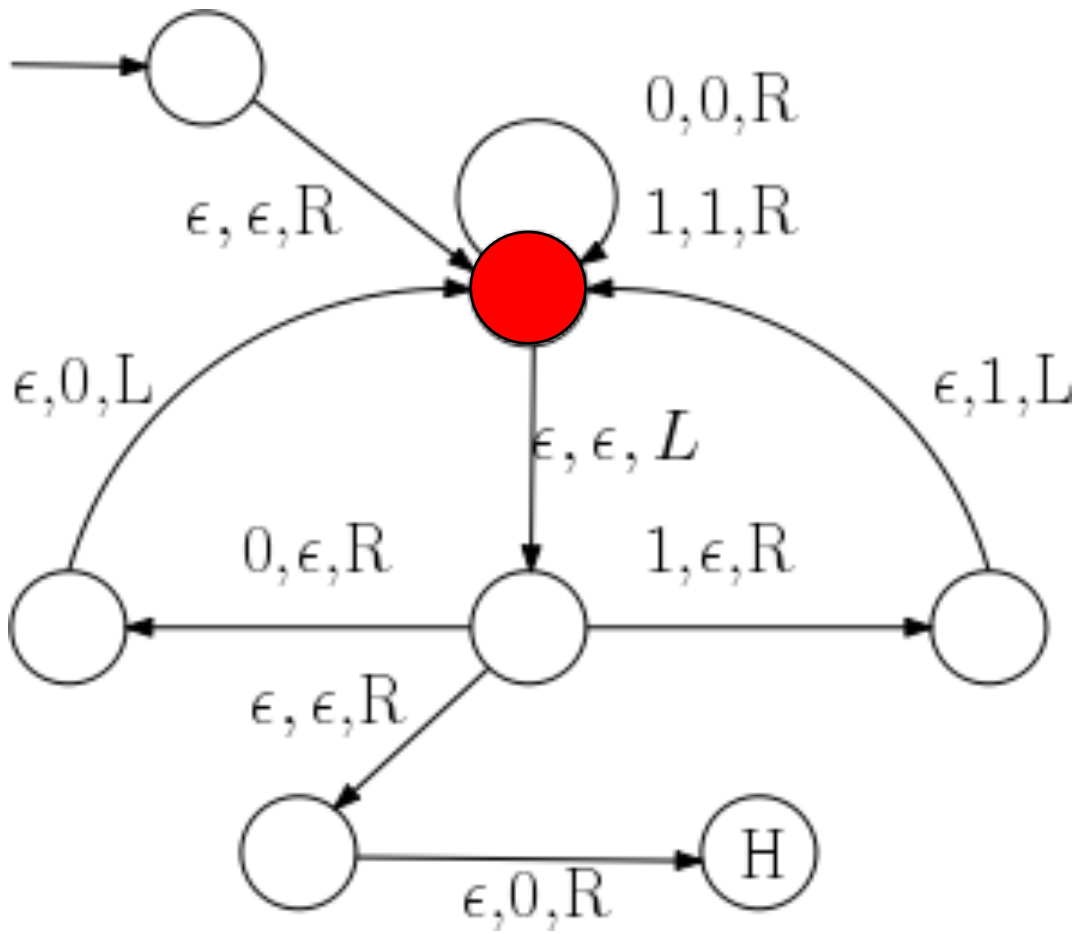



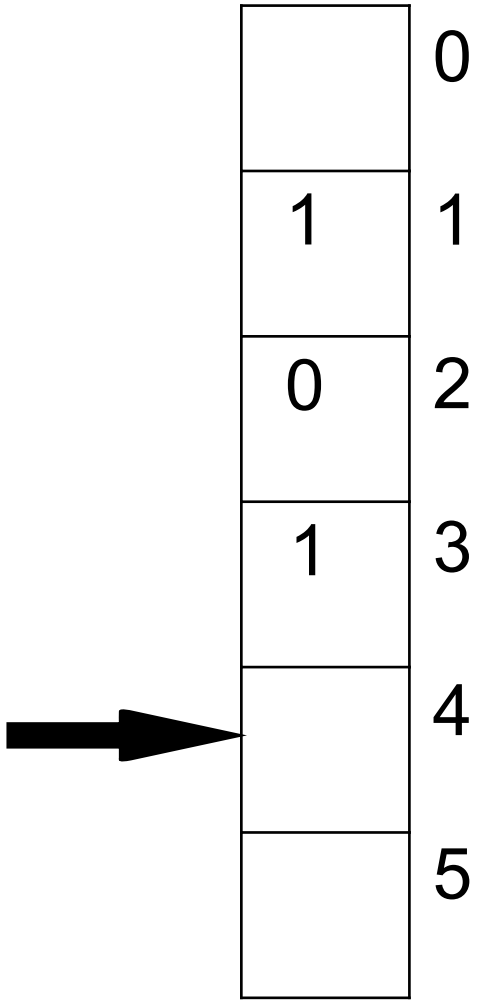
# Example: Doubling a Number



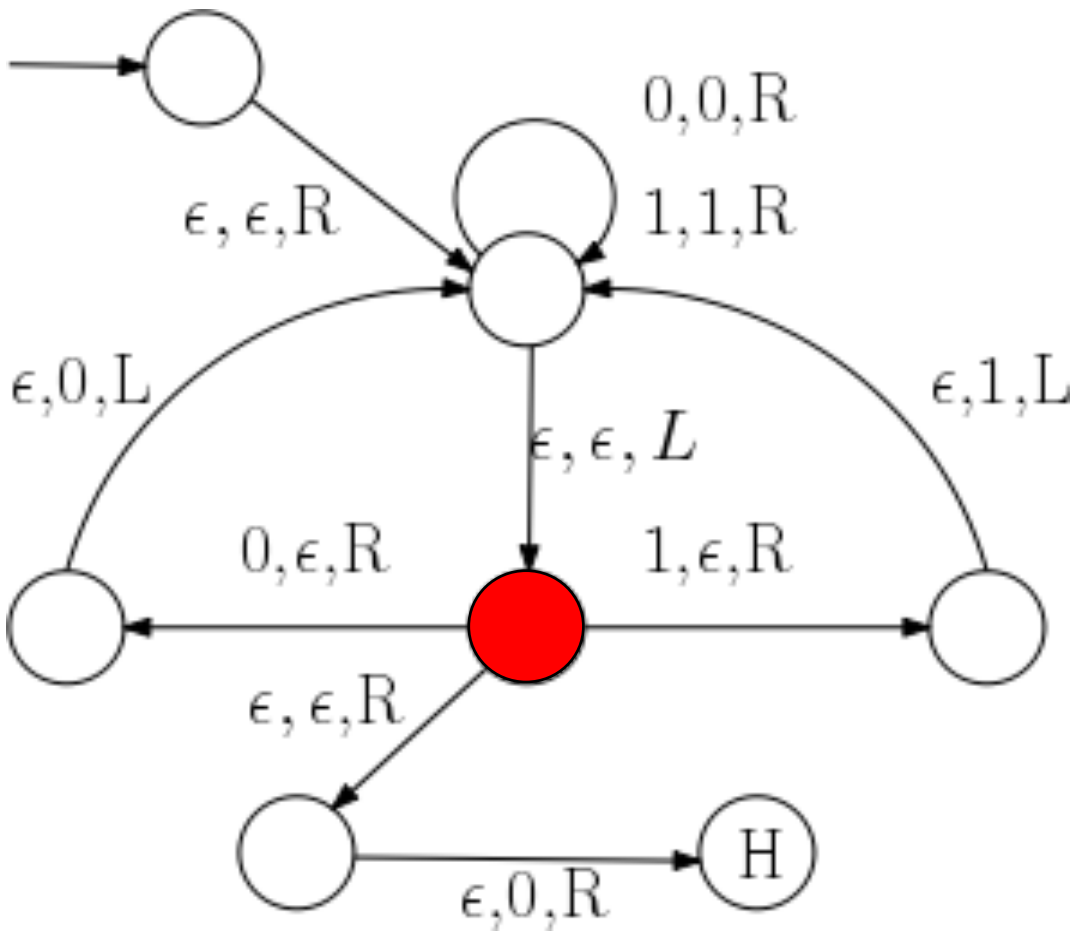



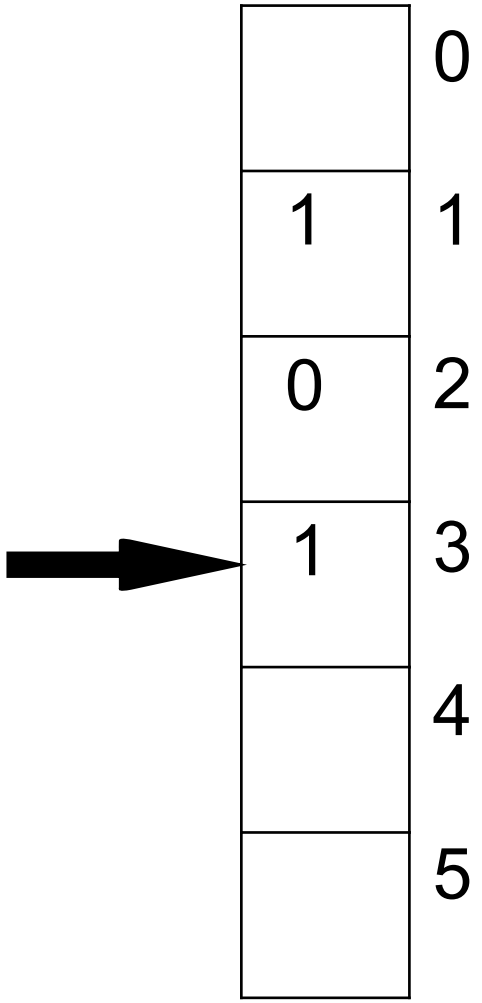
# Example: Doubling a Number



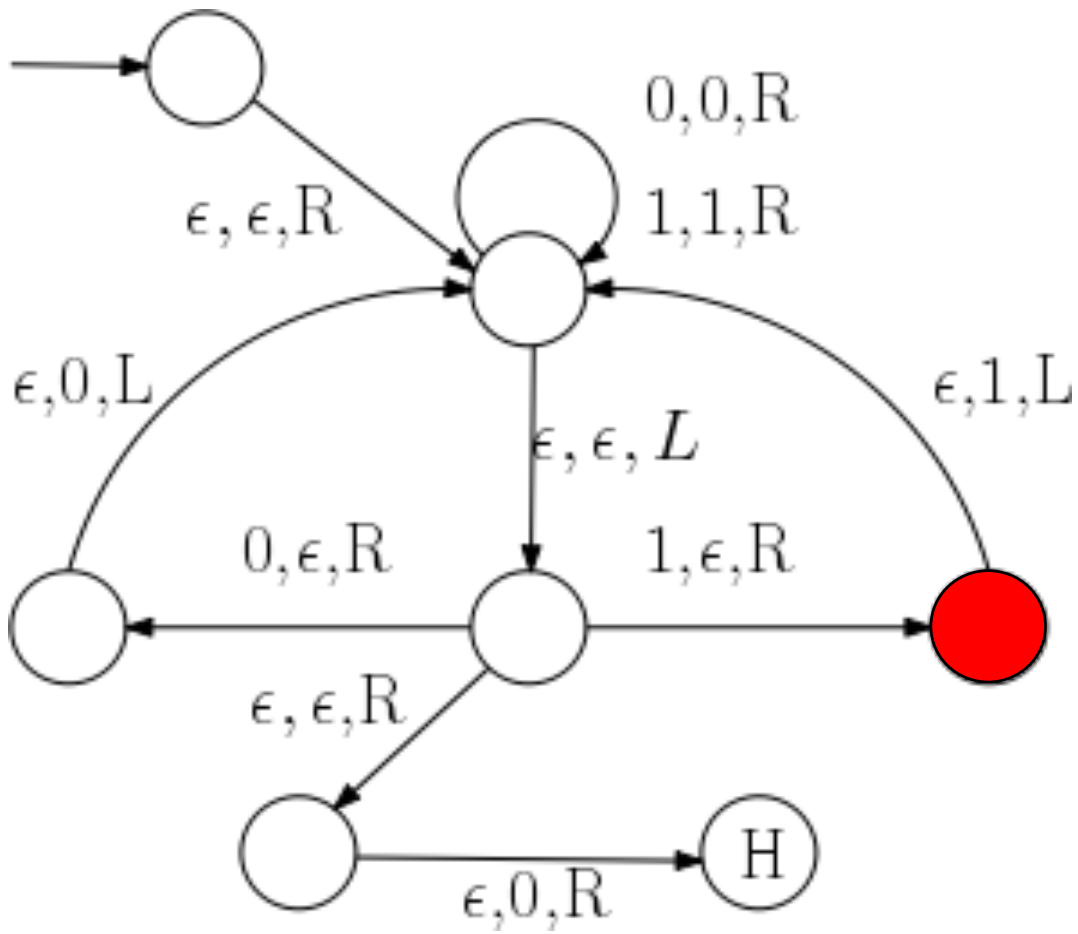



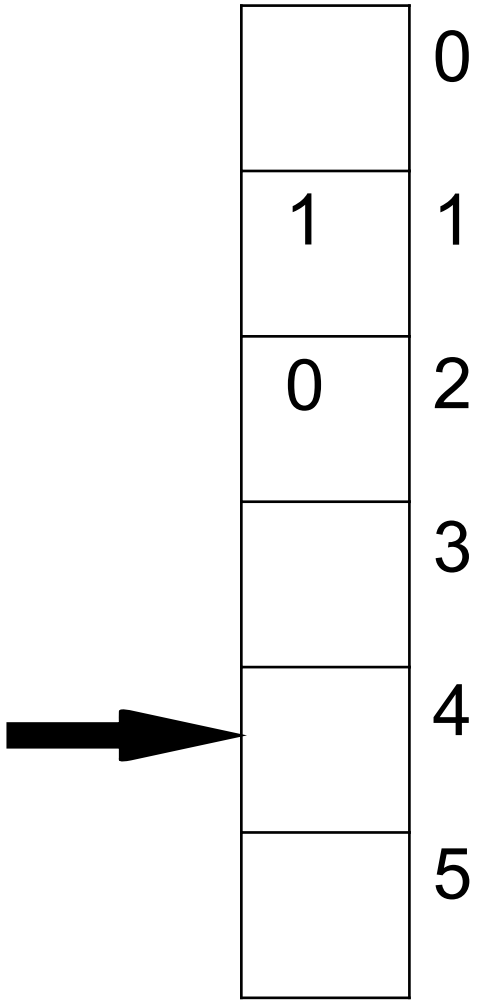
# Example: Doubling a Number



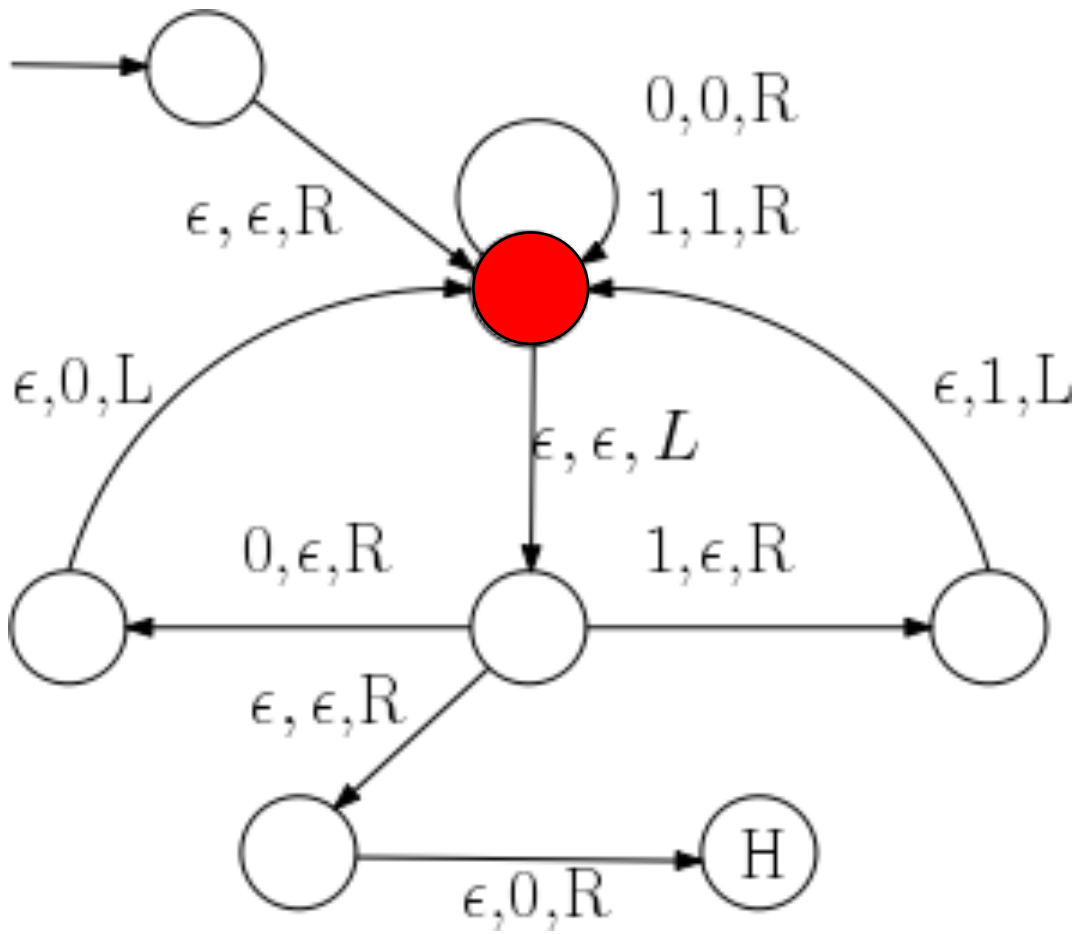



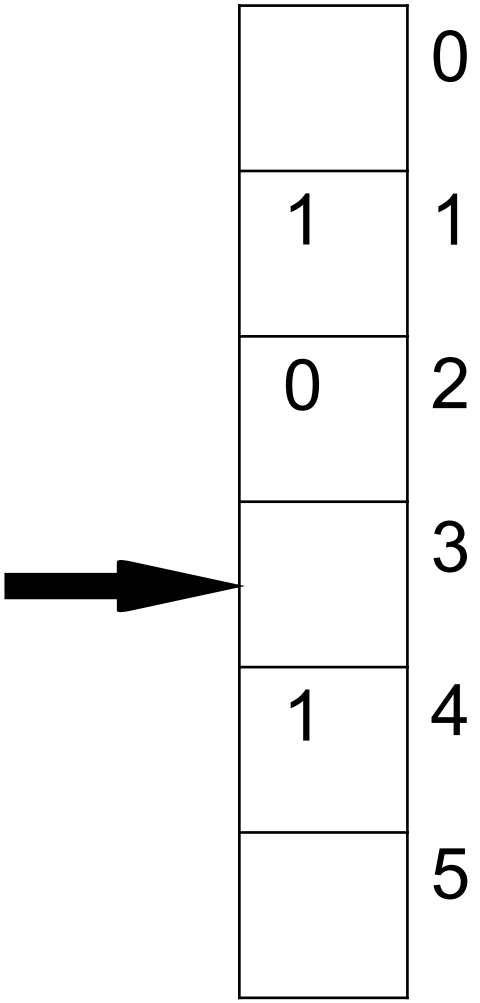
# Example: Doubling a Number



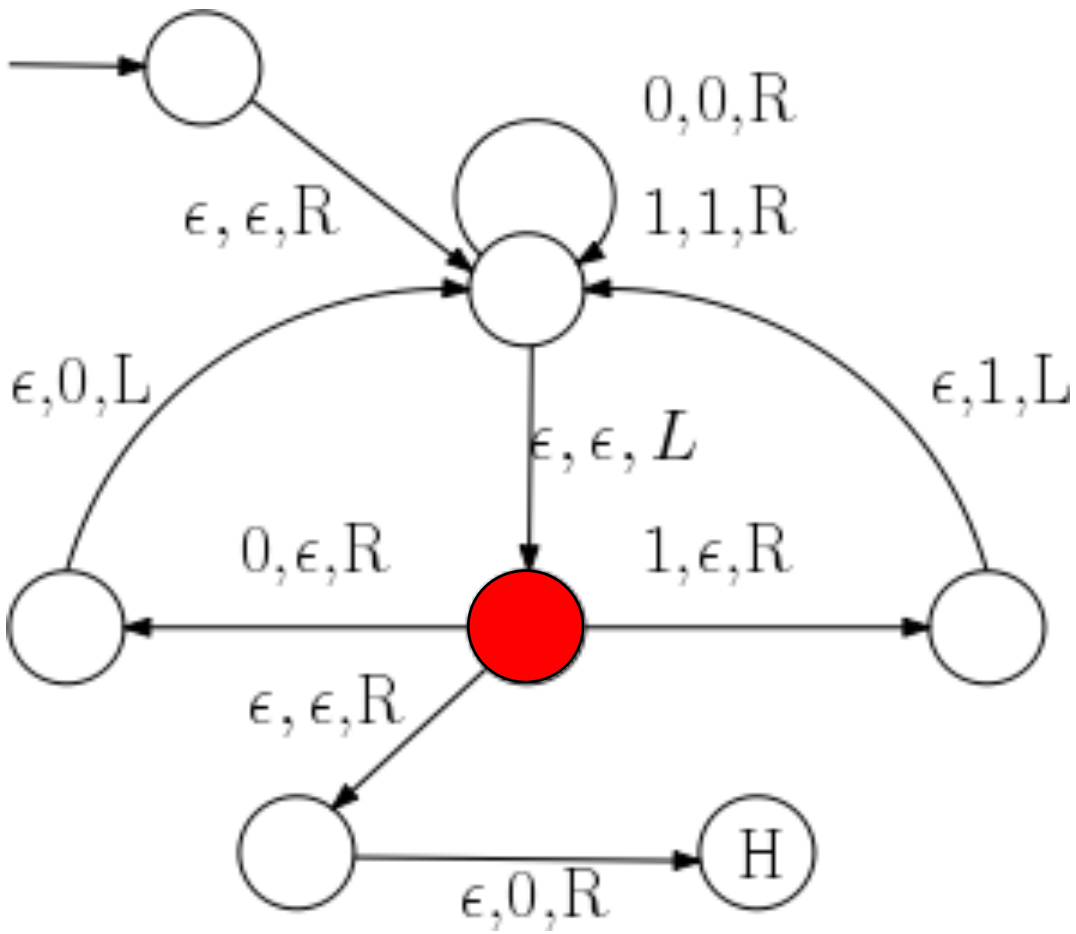



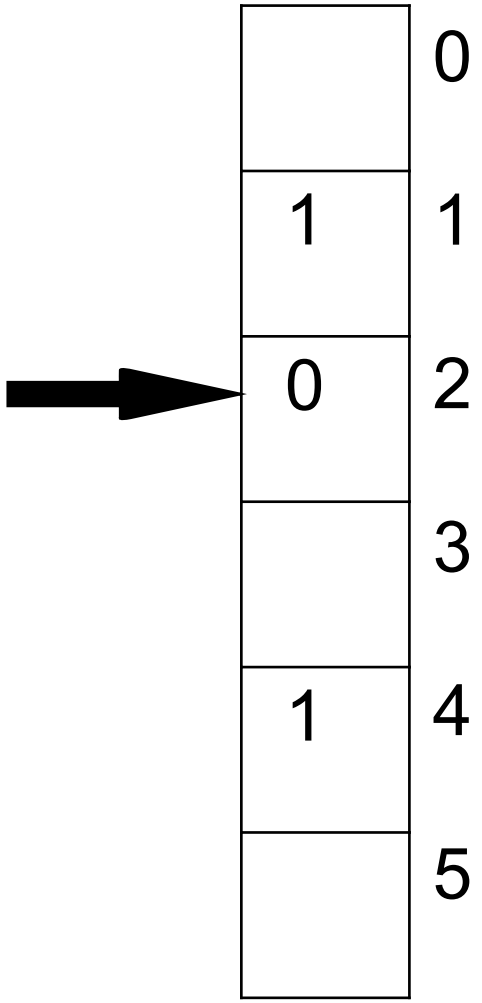
# Example: Doubling a Number



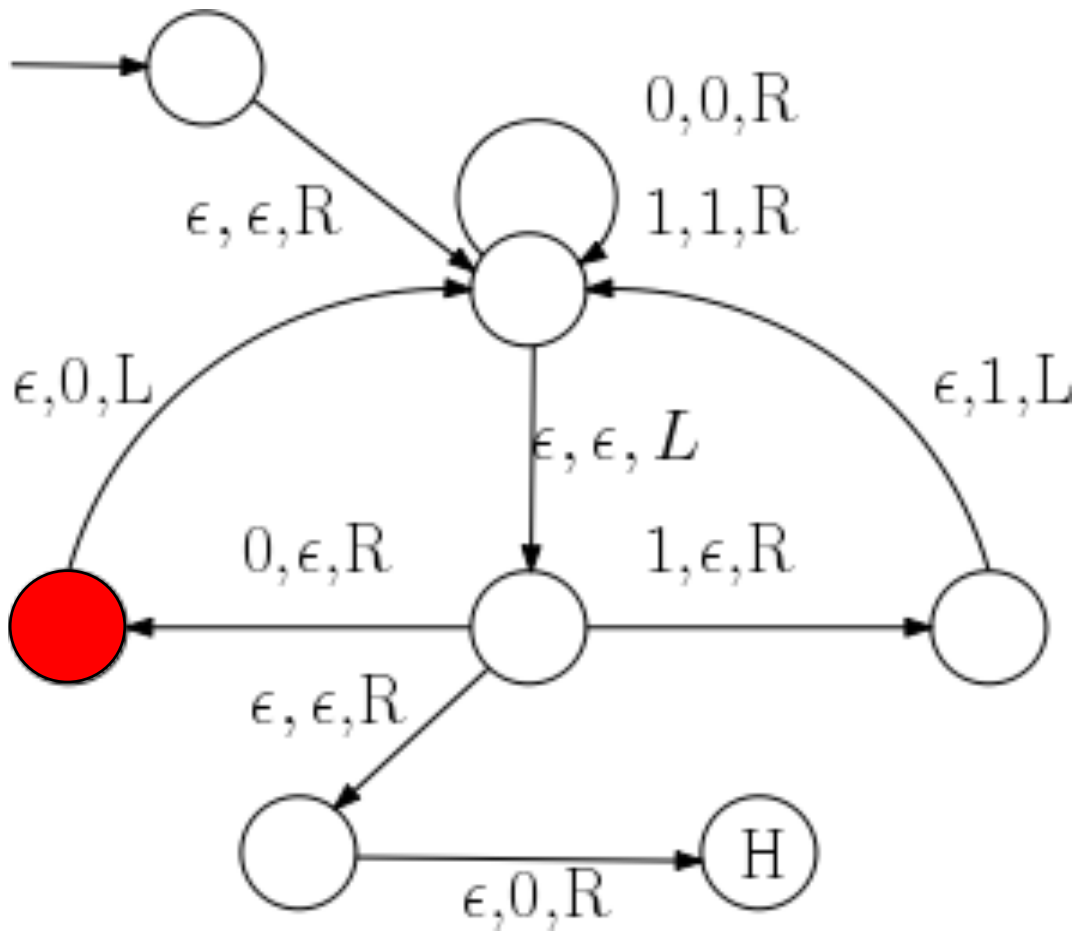



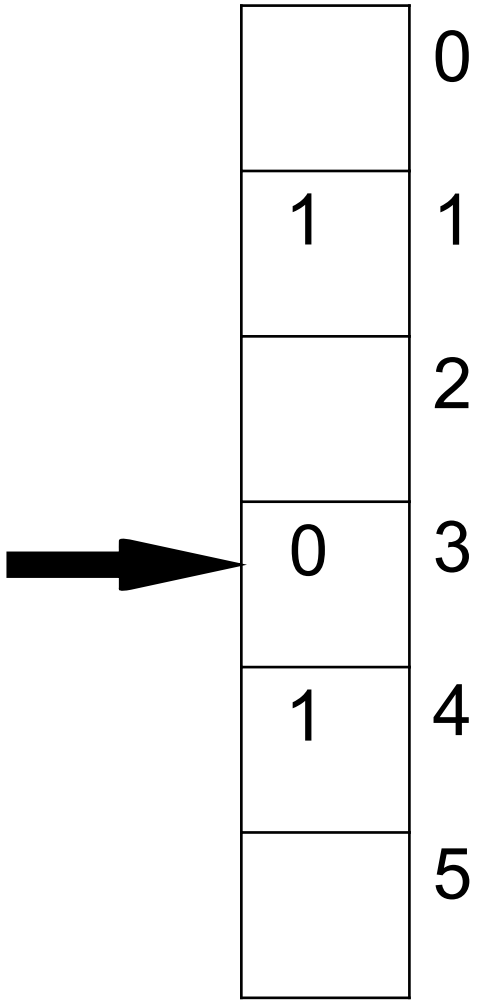
# Example: Doubling a Number



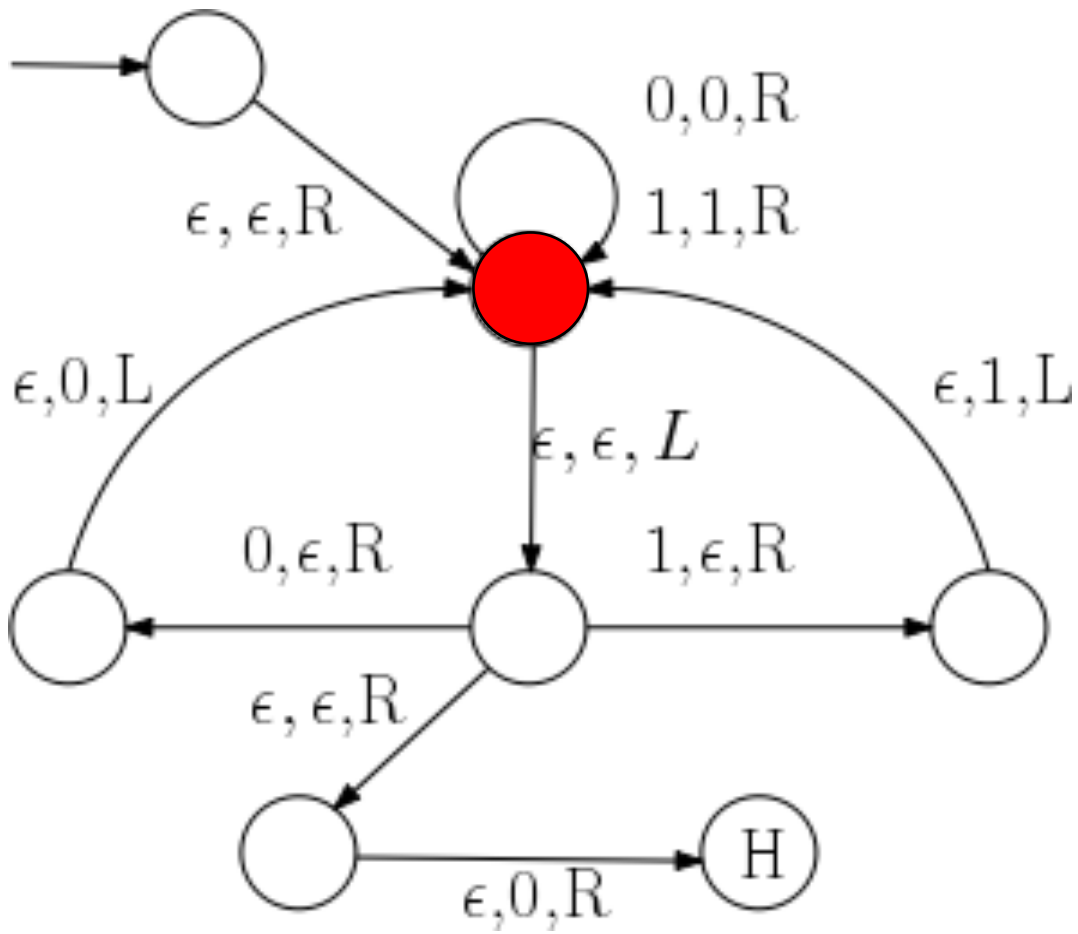



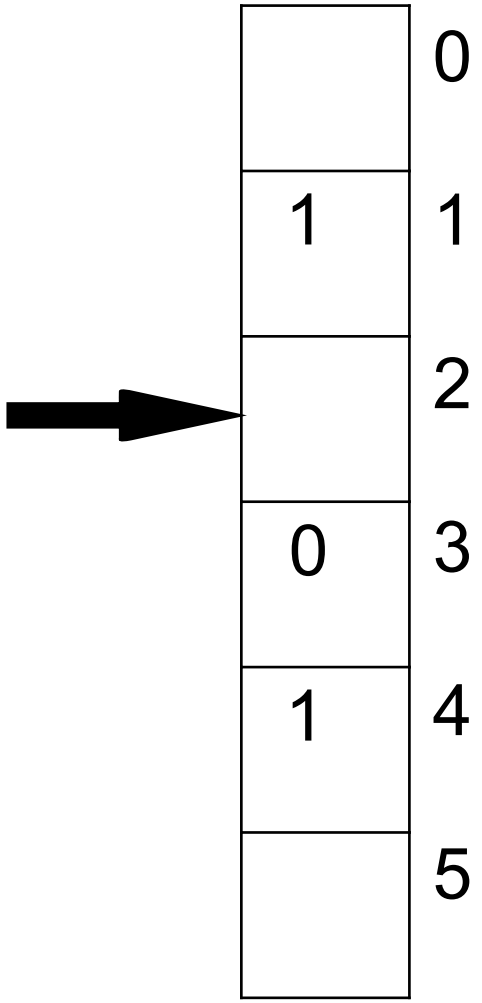
# Example: Doubling a Number



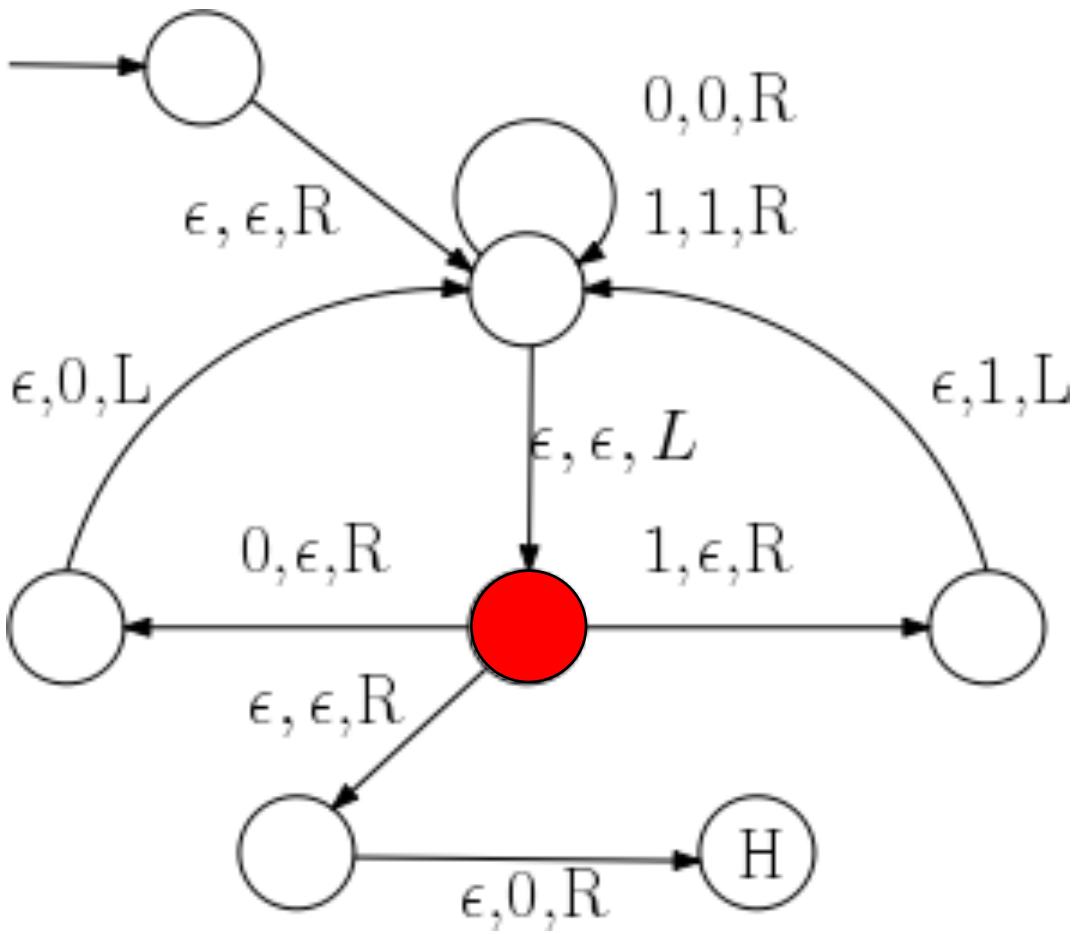



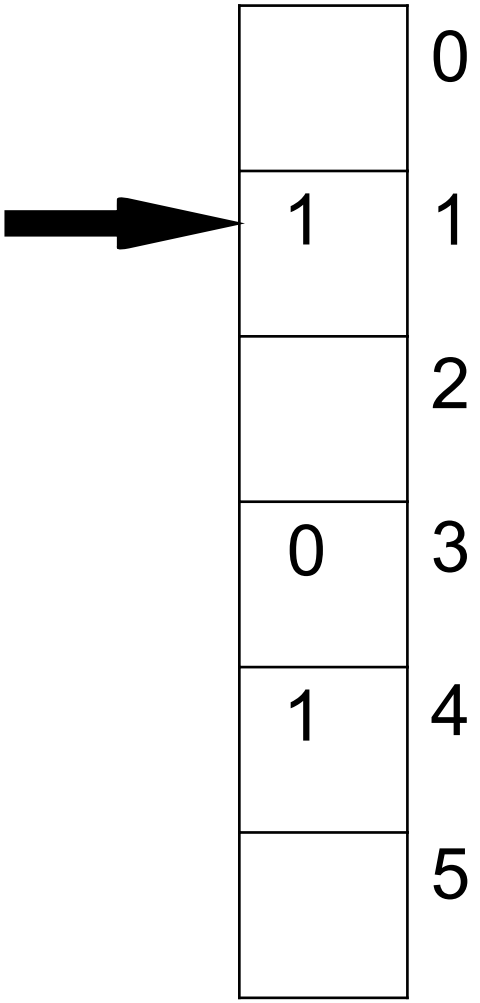
# Example: Doubling a Number



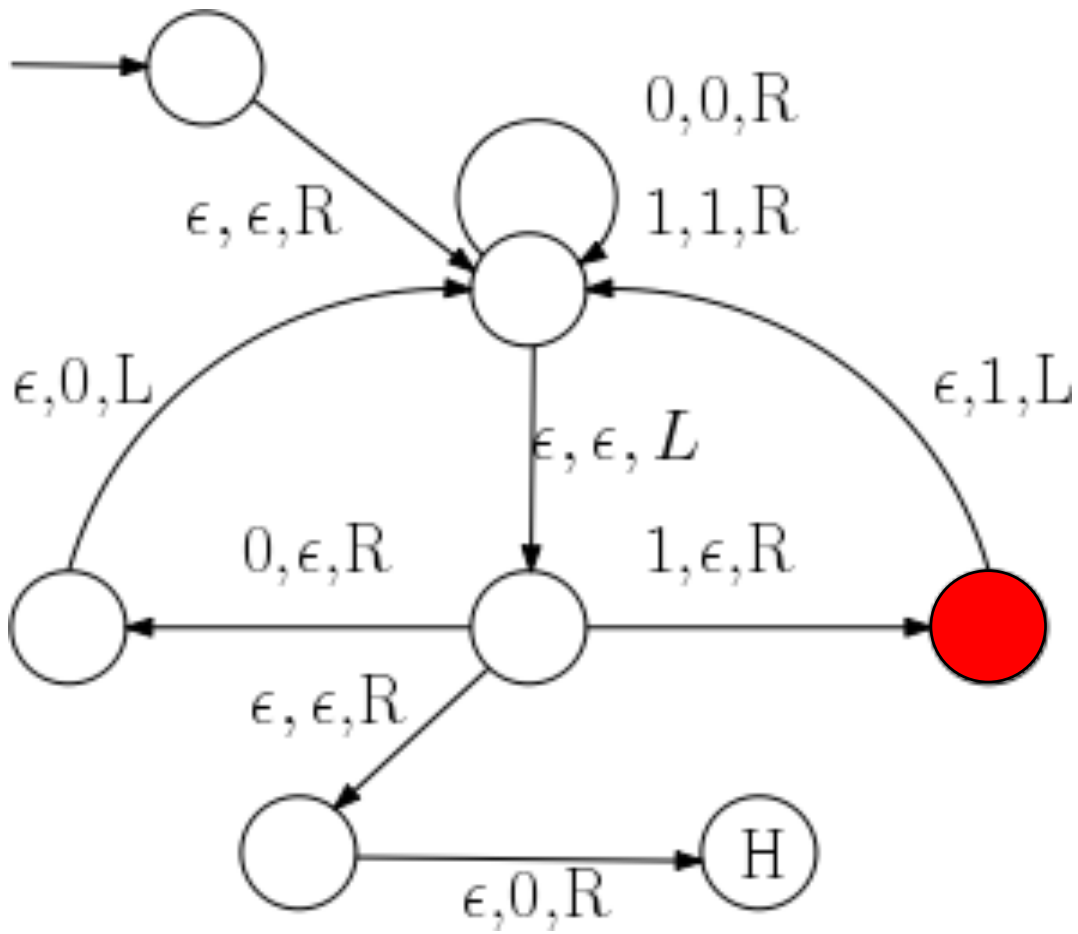



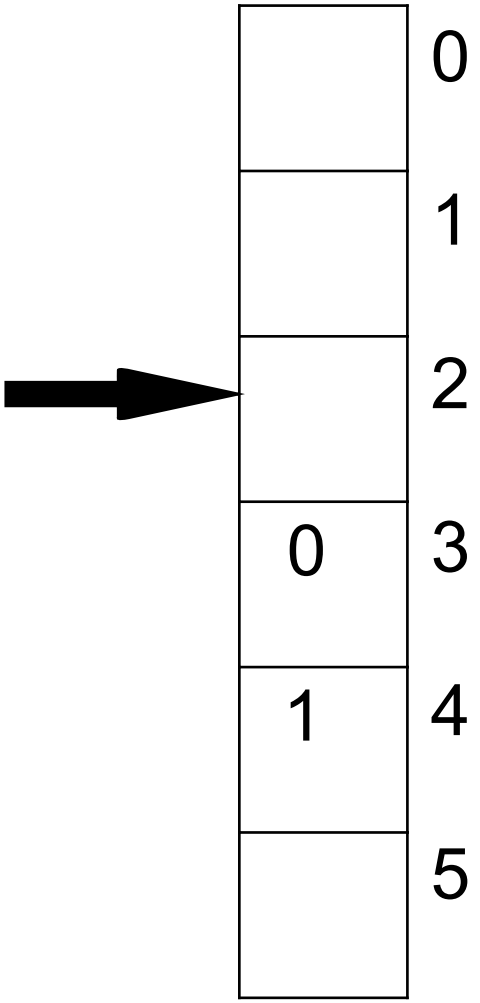
# Example: Doubling a Number



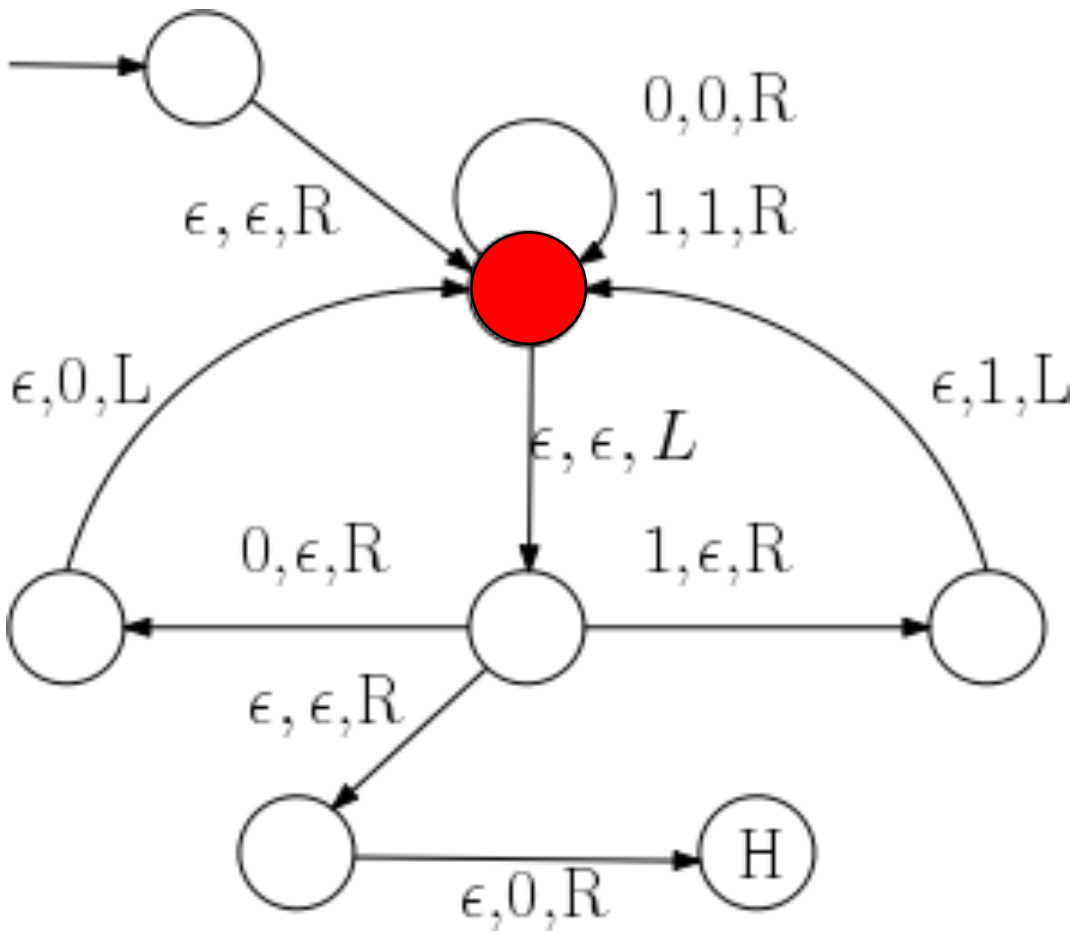



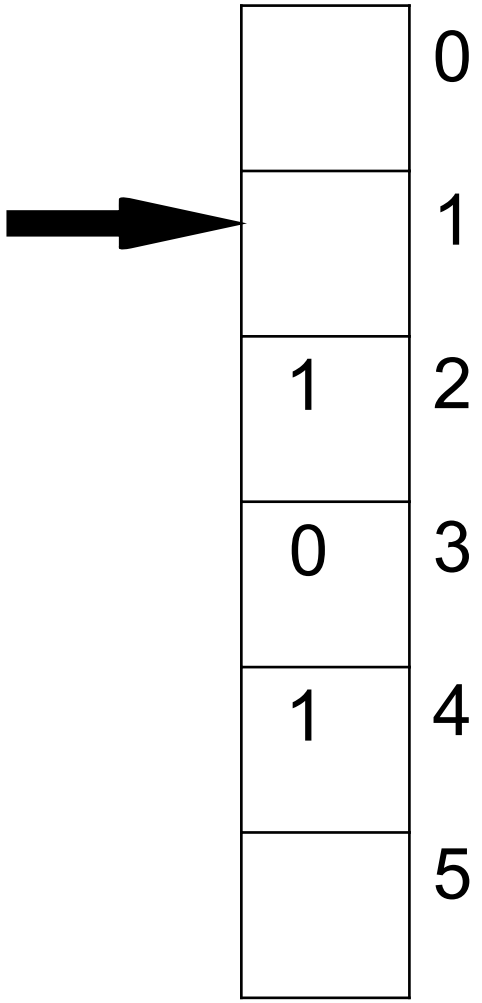
# Example: Doubling a Number



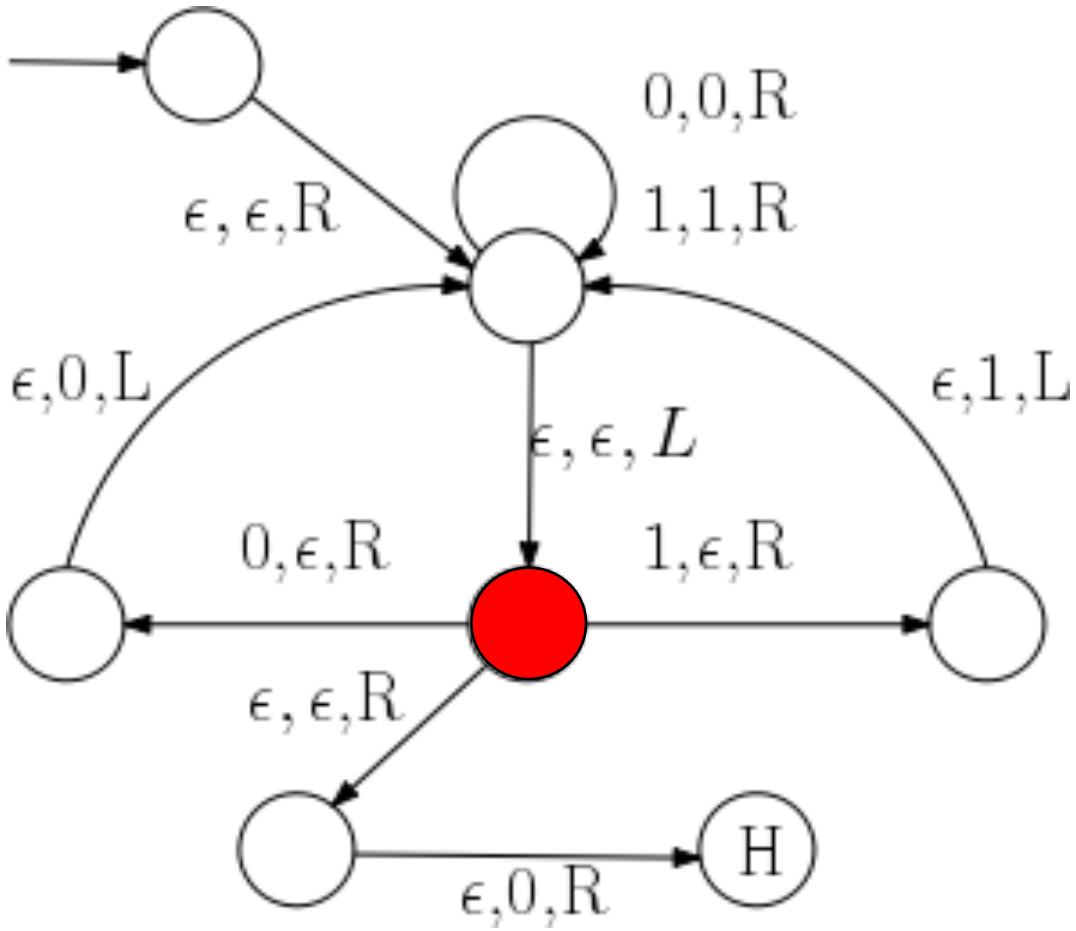



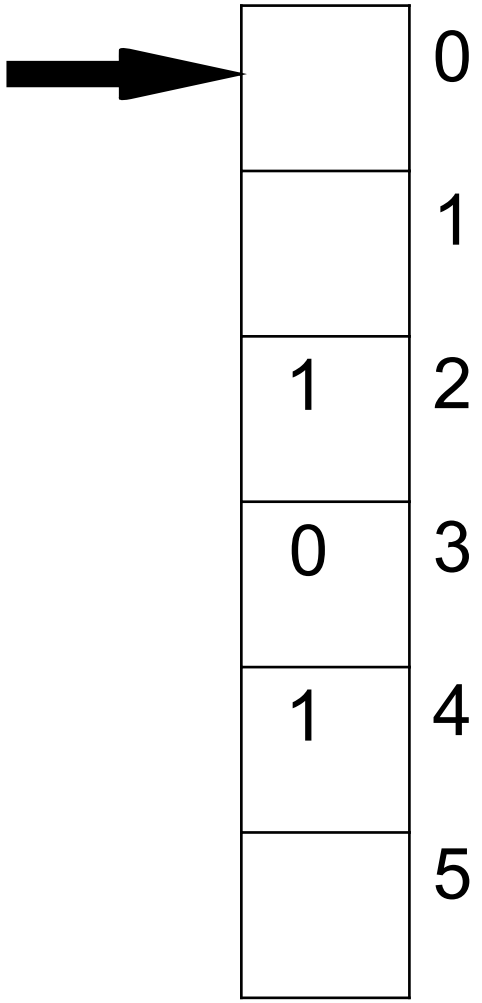
# Example: Doubling a Number



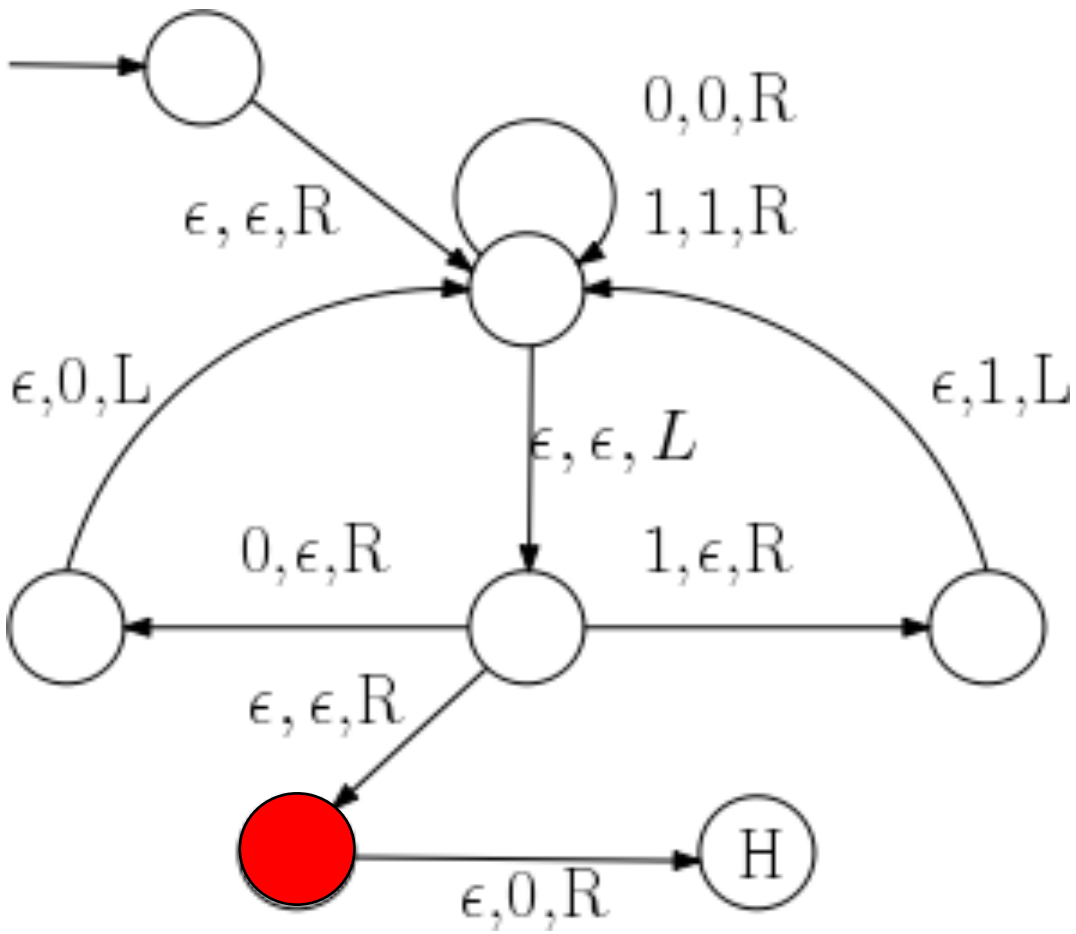



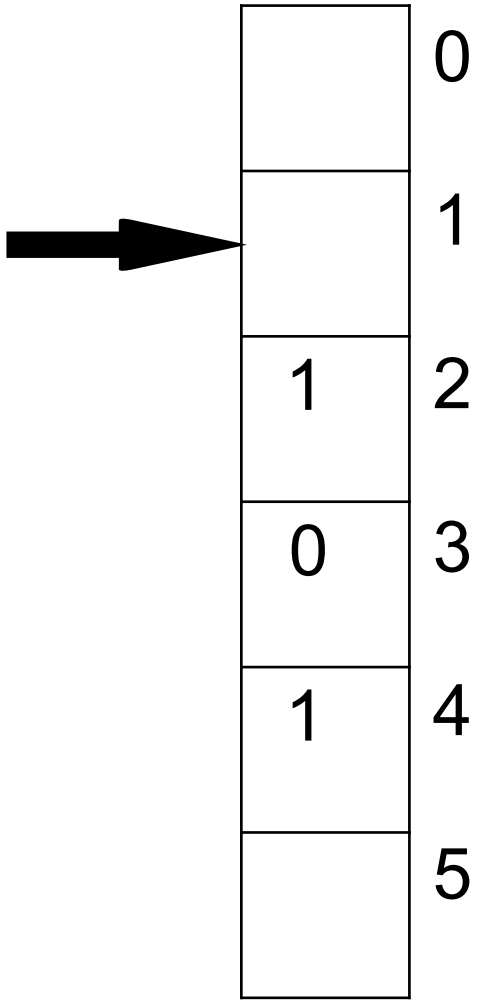
# Example: Doubling a Number



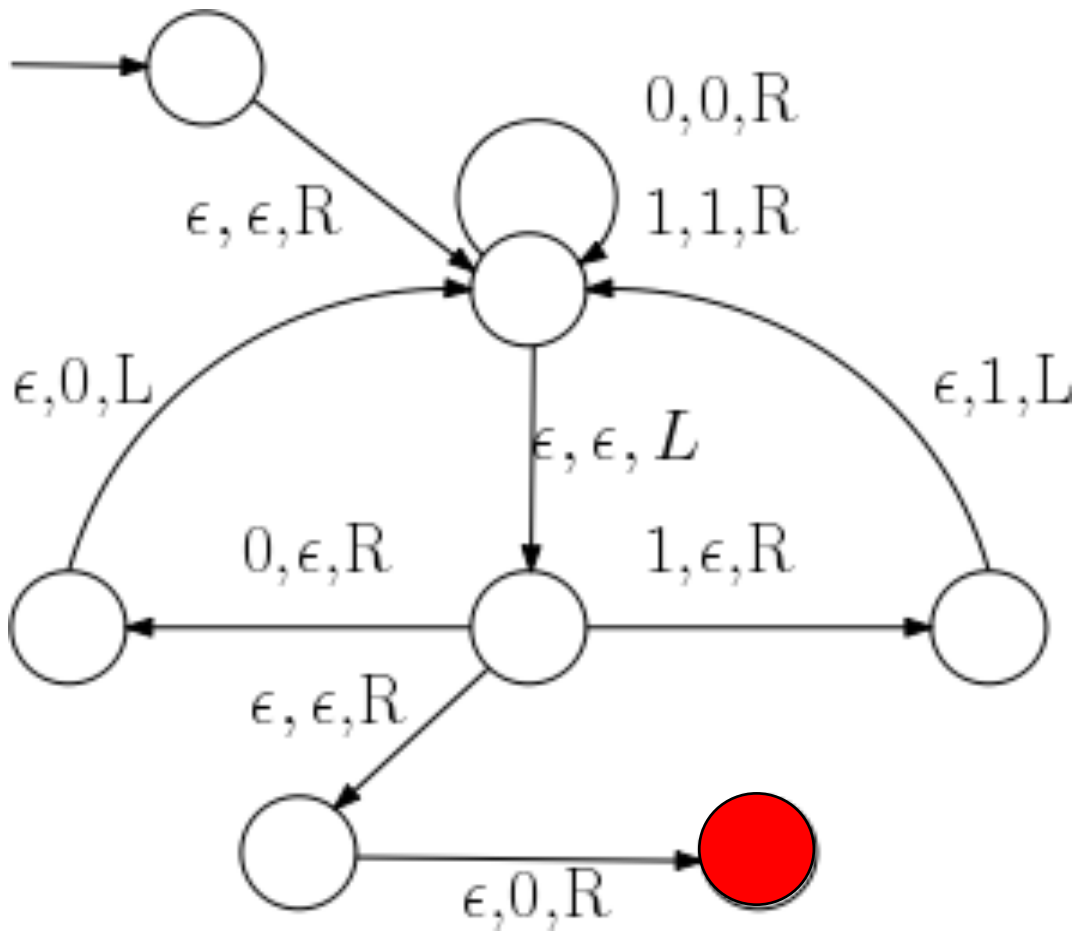



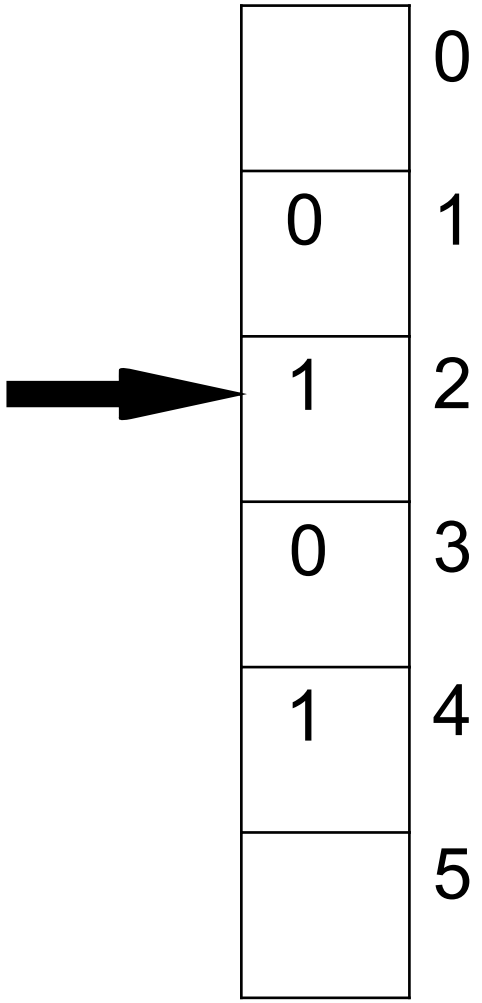
# Example: Doubling a Number



# Example: Doubling a Number

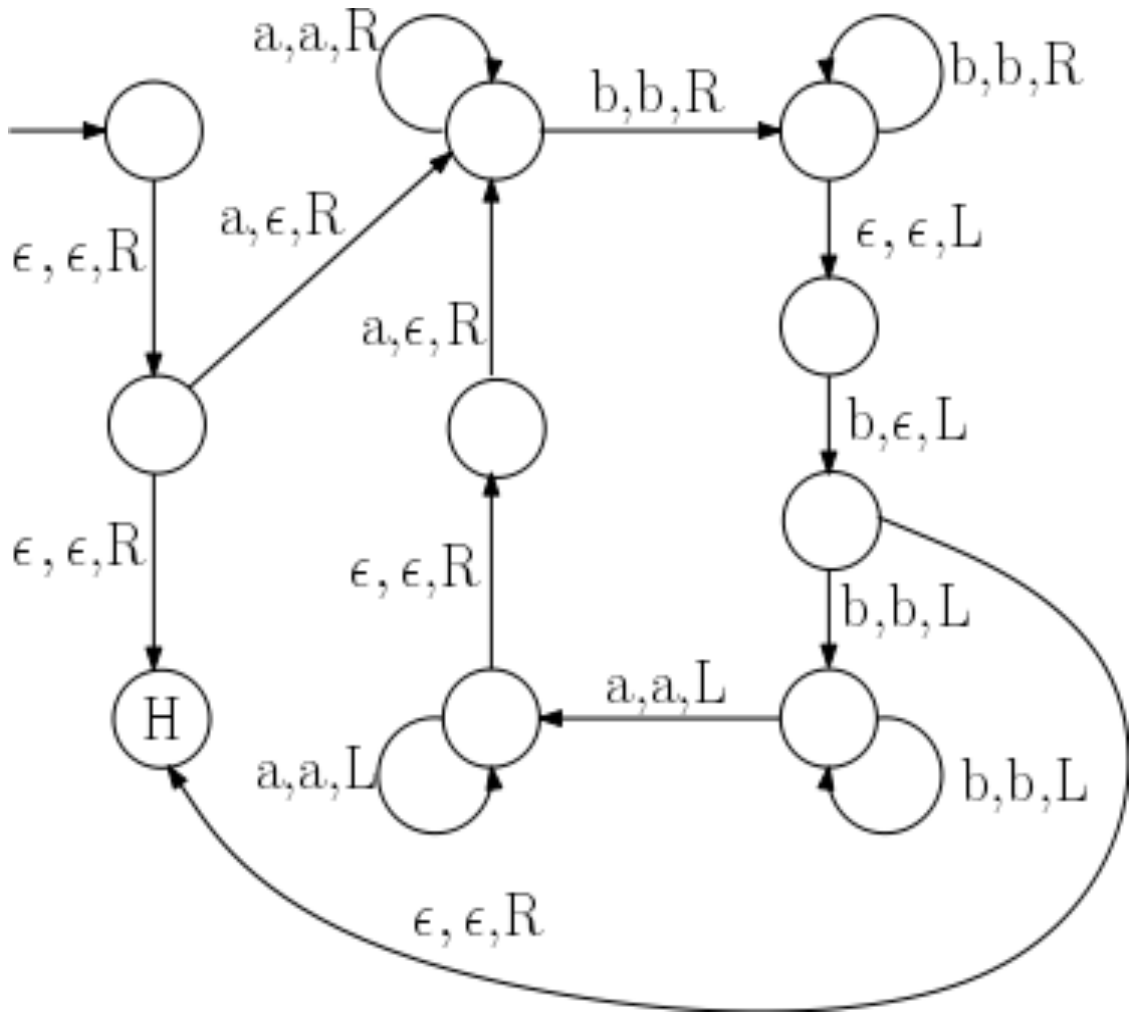


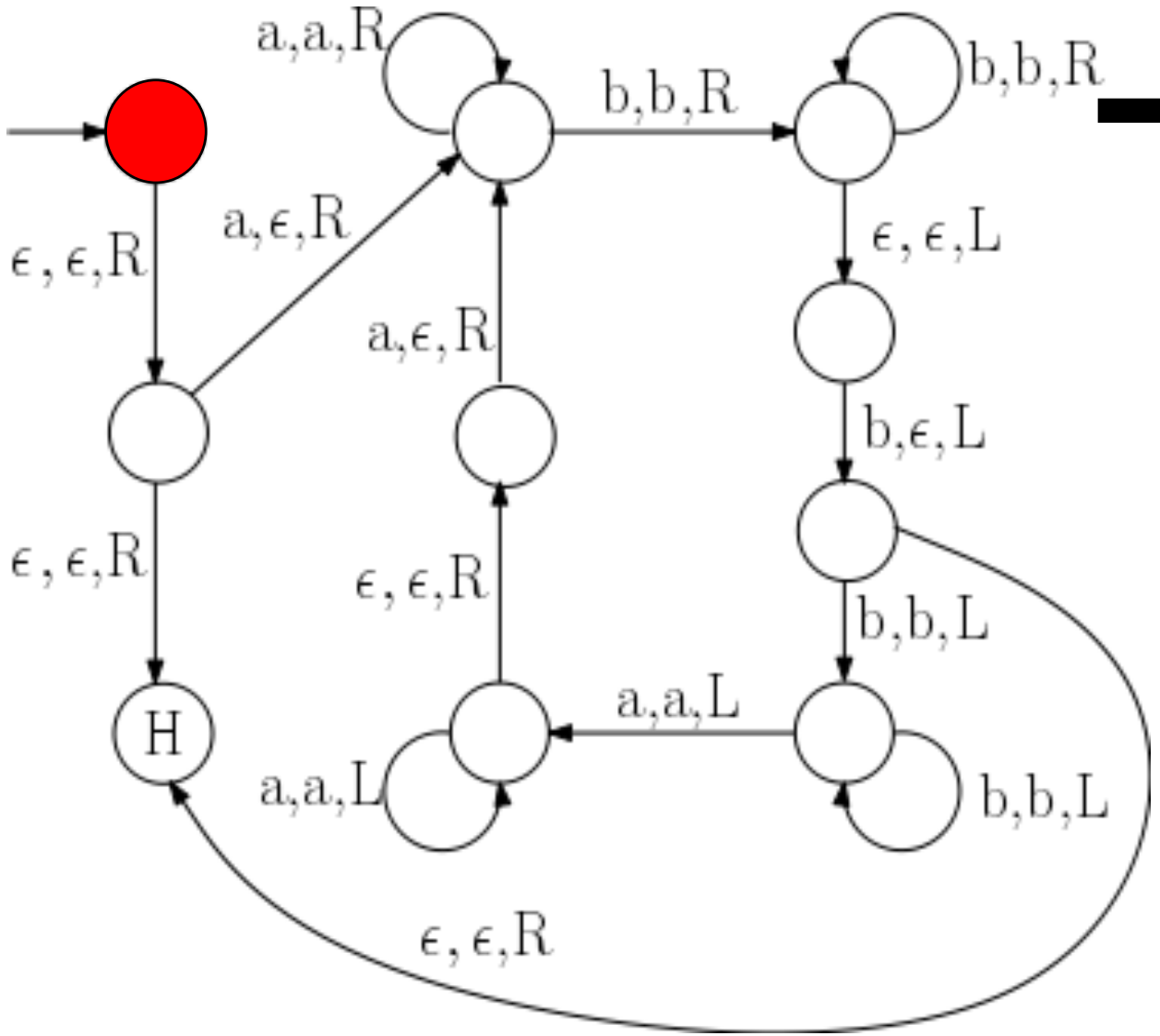
# TM vs DFA

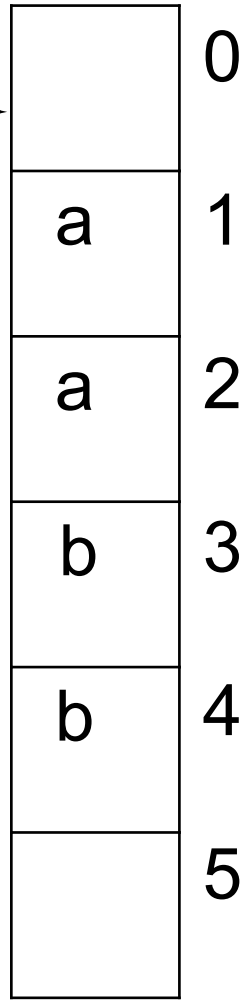
- Is a decision machine more powerful than a DFA?
- Yes!
- Claim: A decision machine can recognize the set $\{a^n b^n \mid n >= 0\}$

# Example: $a^n b^n$

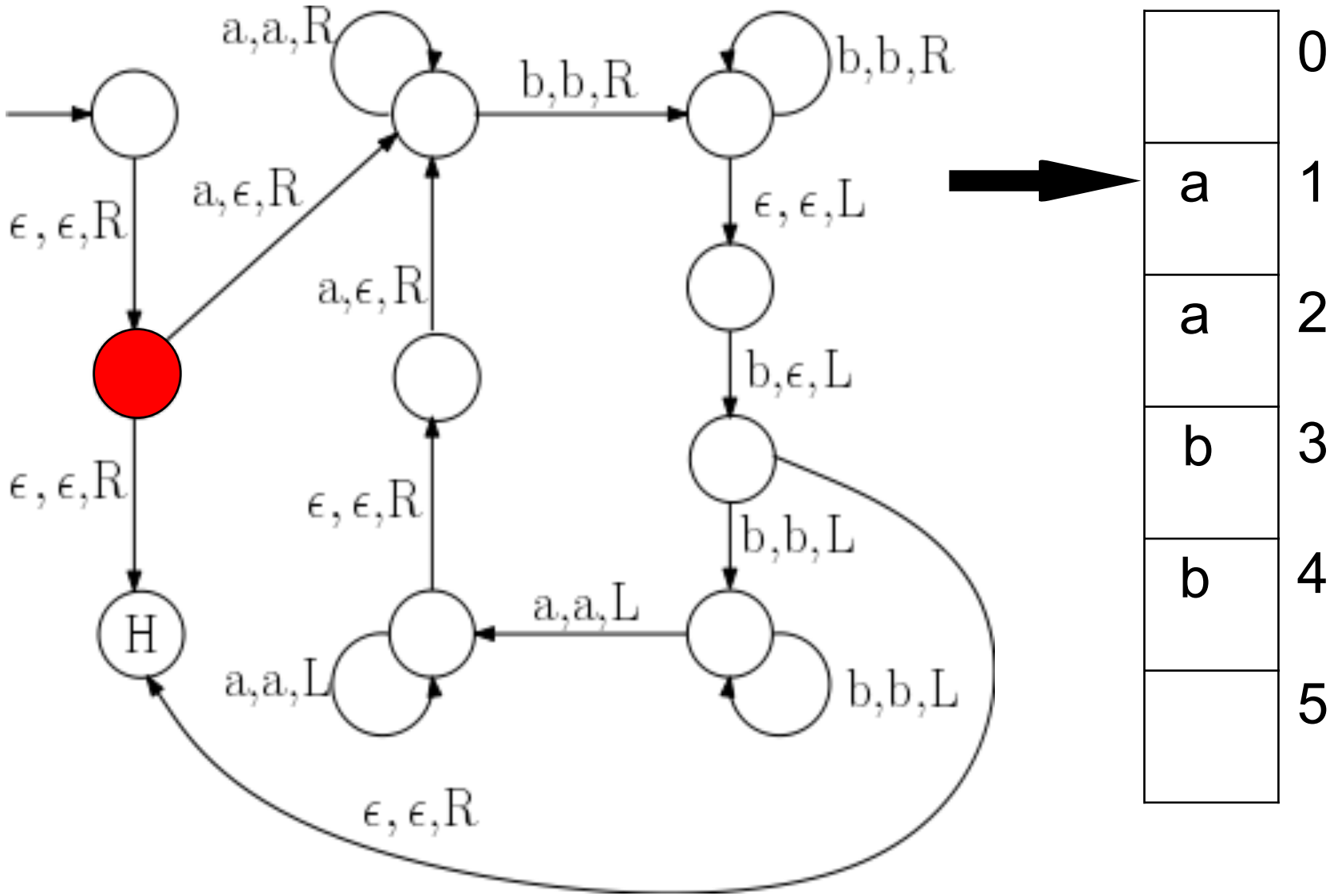


# Example: $a^n b^n$

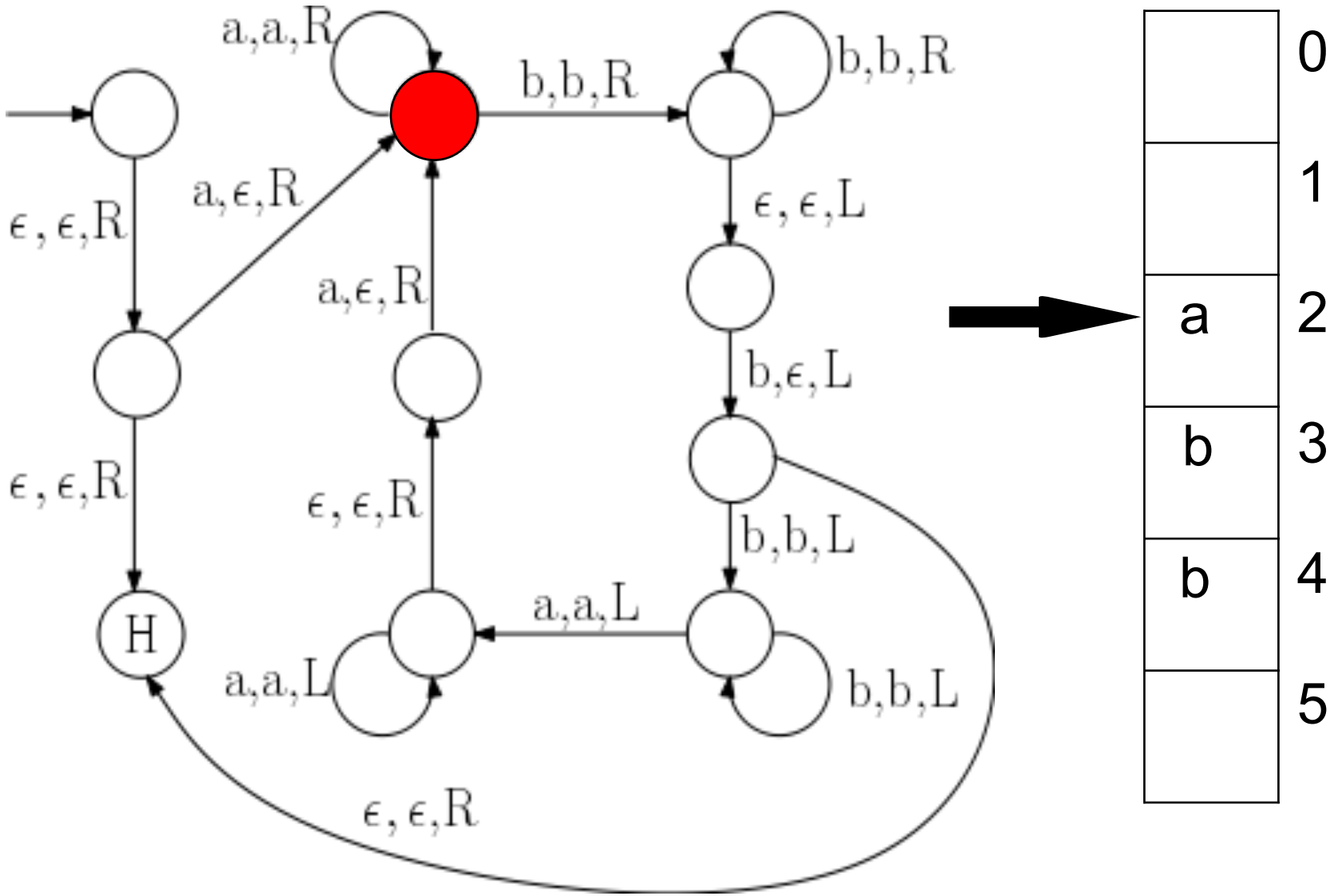


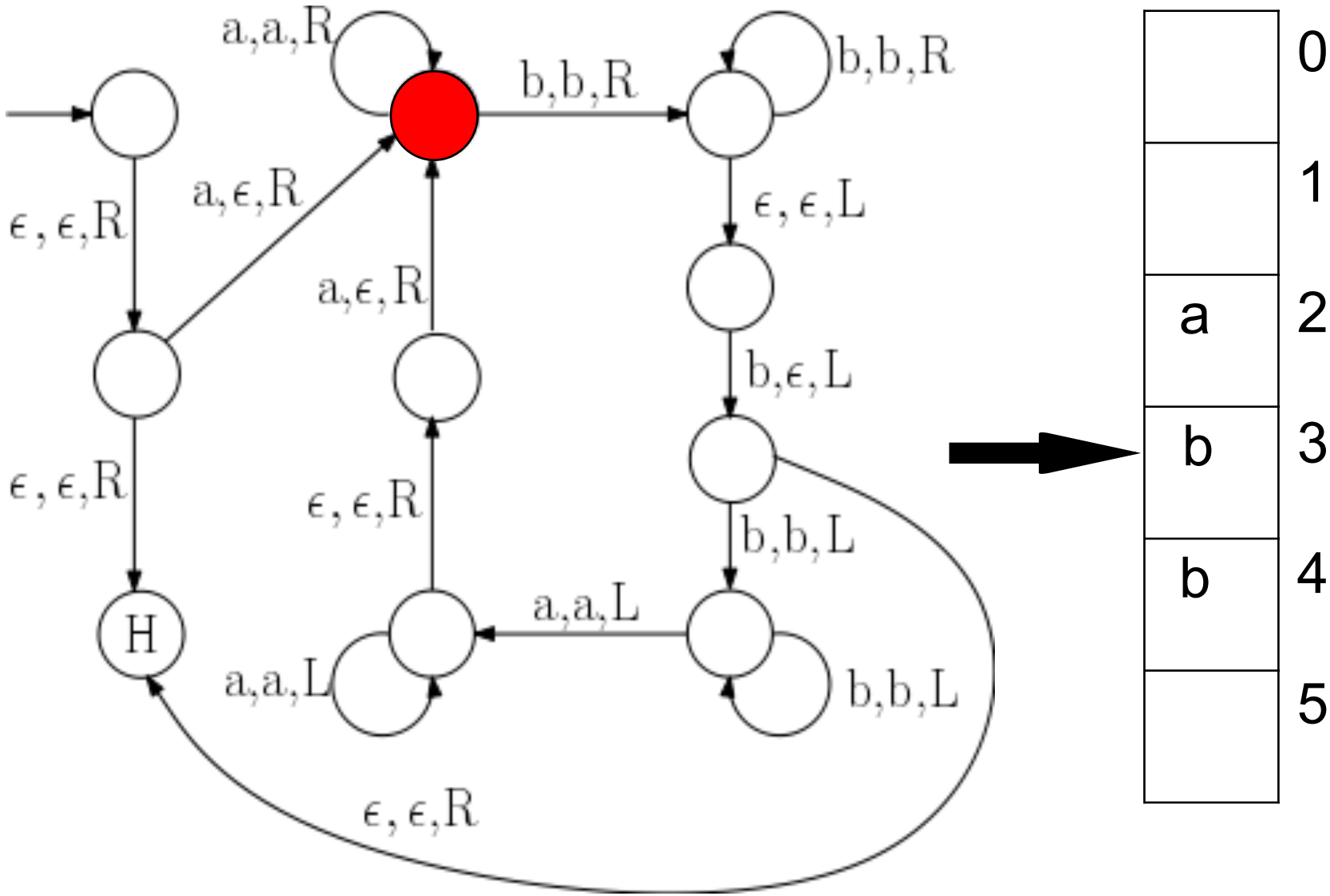
# Example: $a^n b^n$



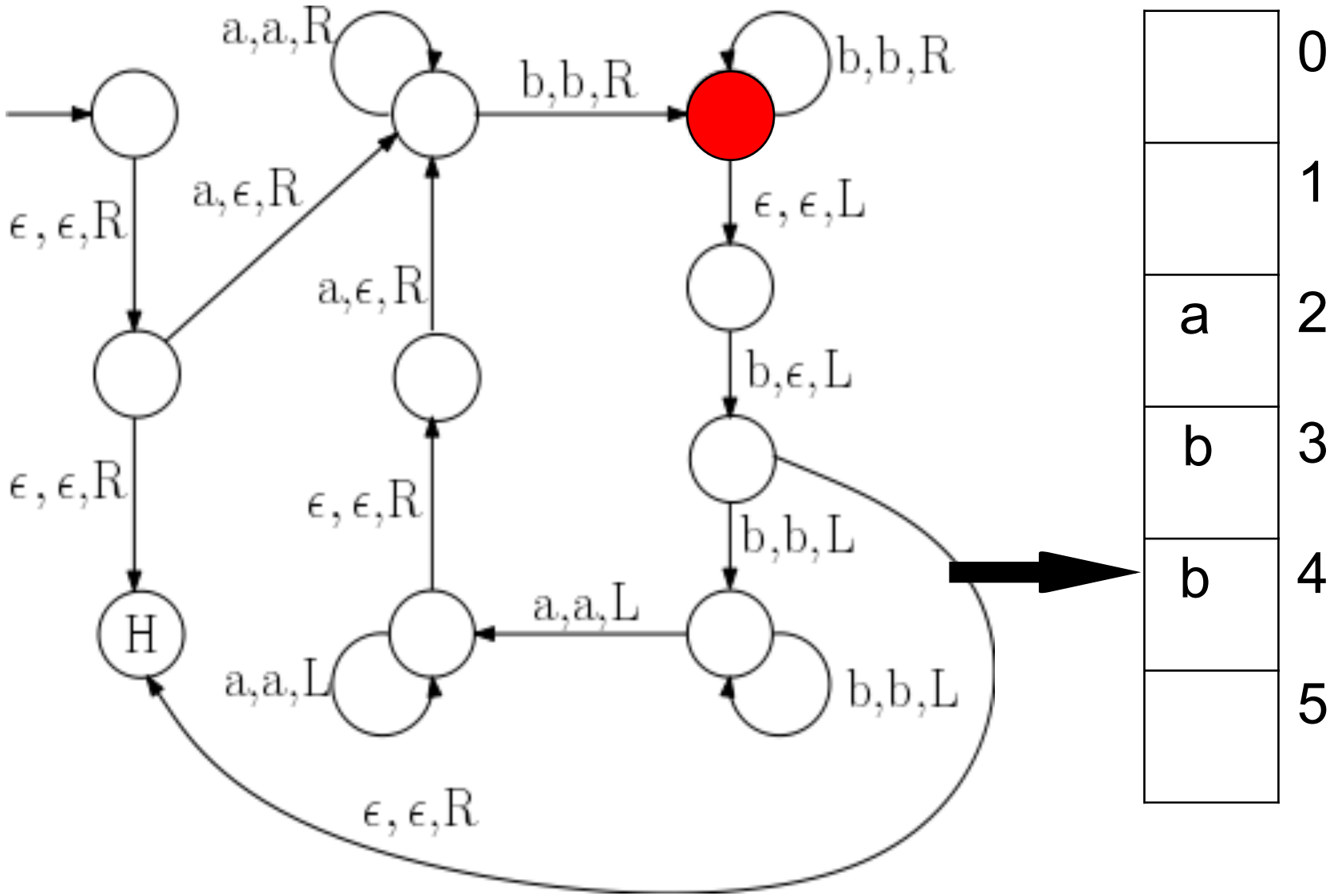
# Example: $a^n b^n$



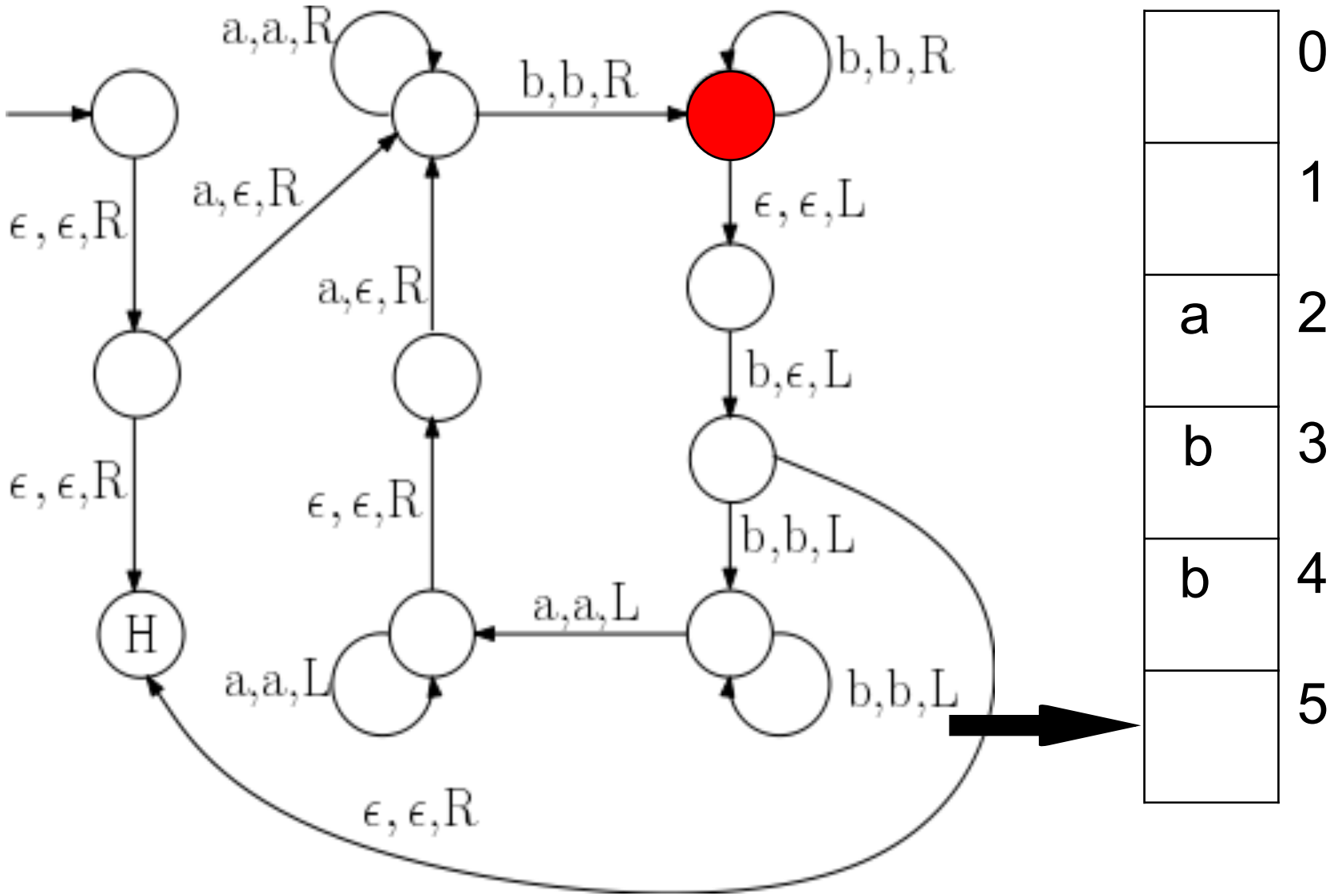
# Example: $a^n b^n$



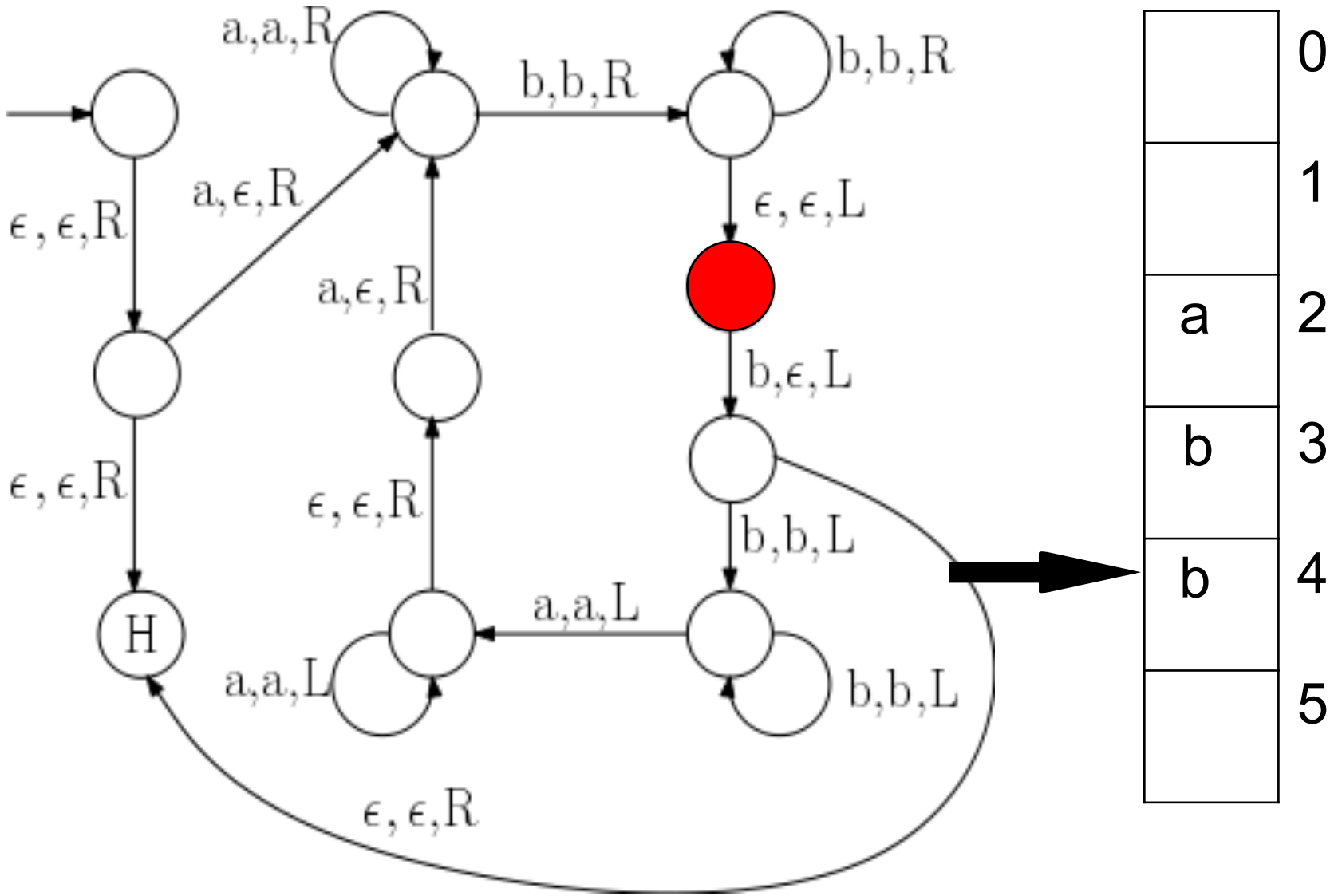
# Example: $a^n b^n$



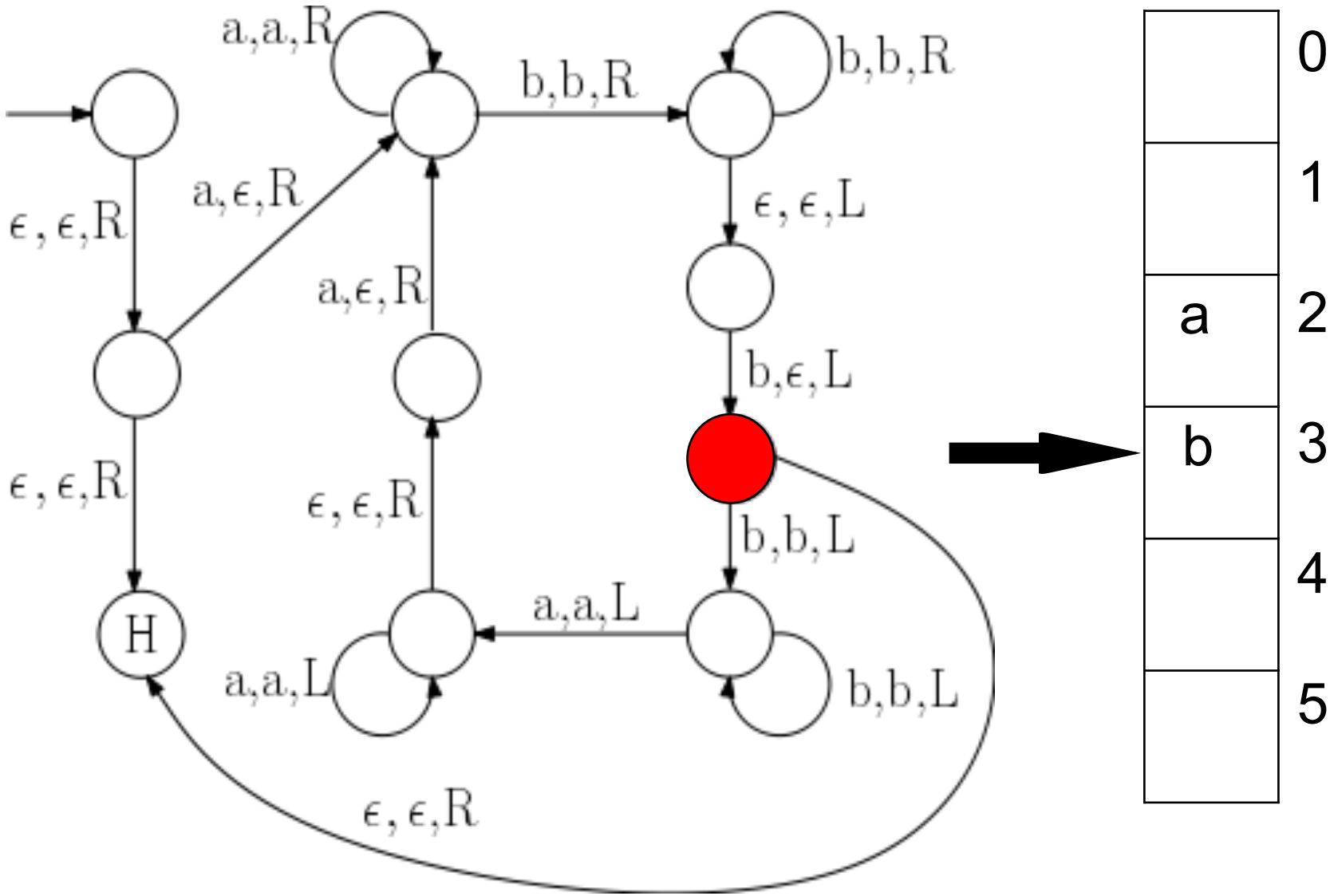
# Example: $a^n b^n$



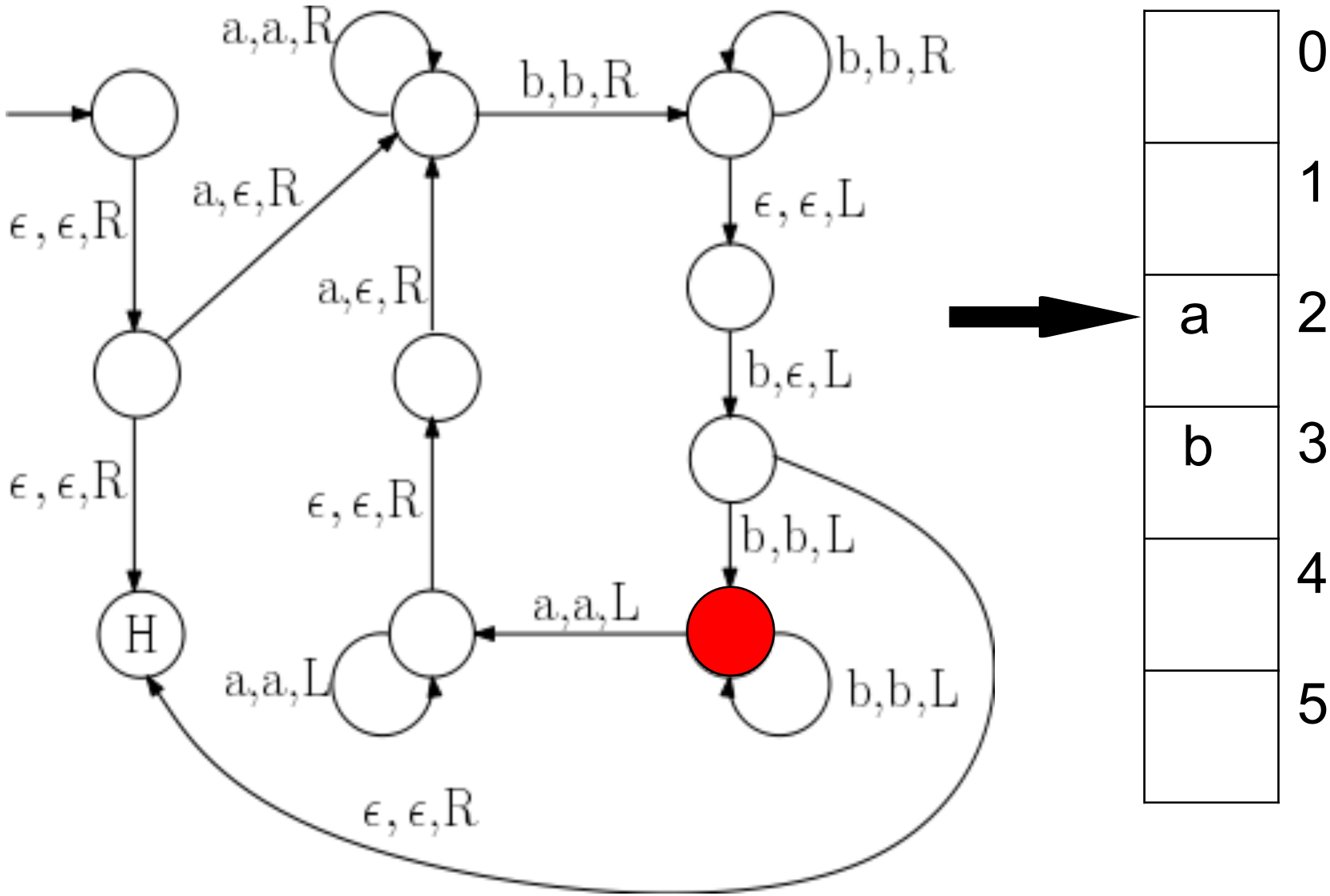
# Example: $a^n b^n$



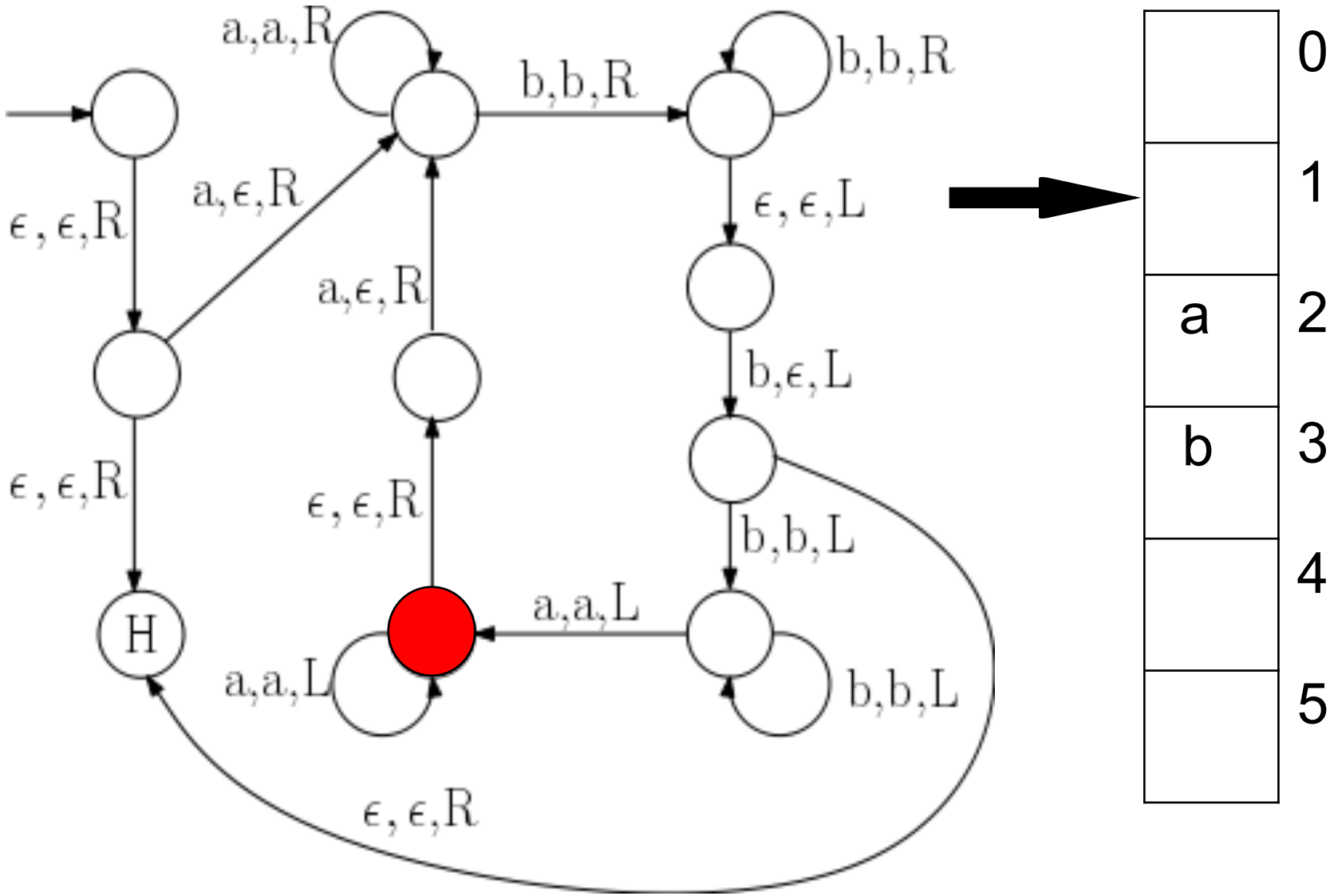
# Example: $a^n b^n$



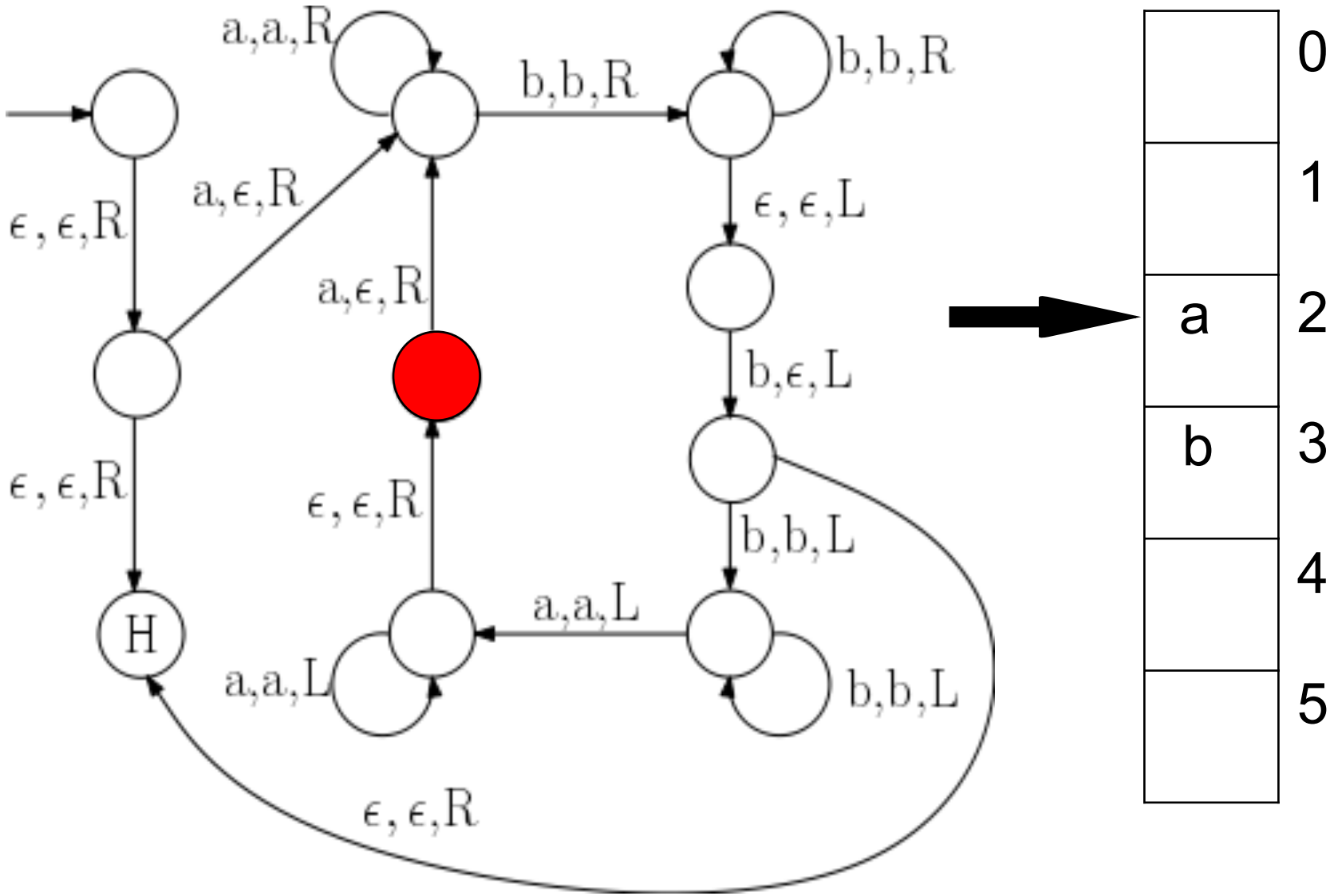
# Example: $a^n b^n$



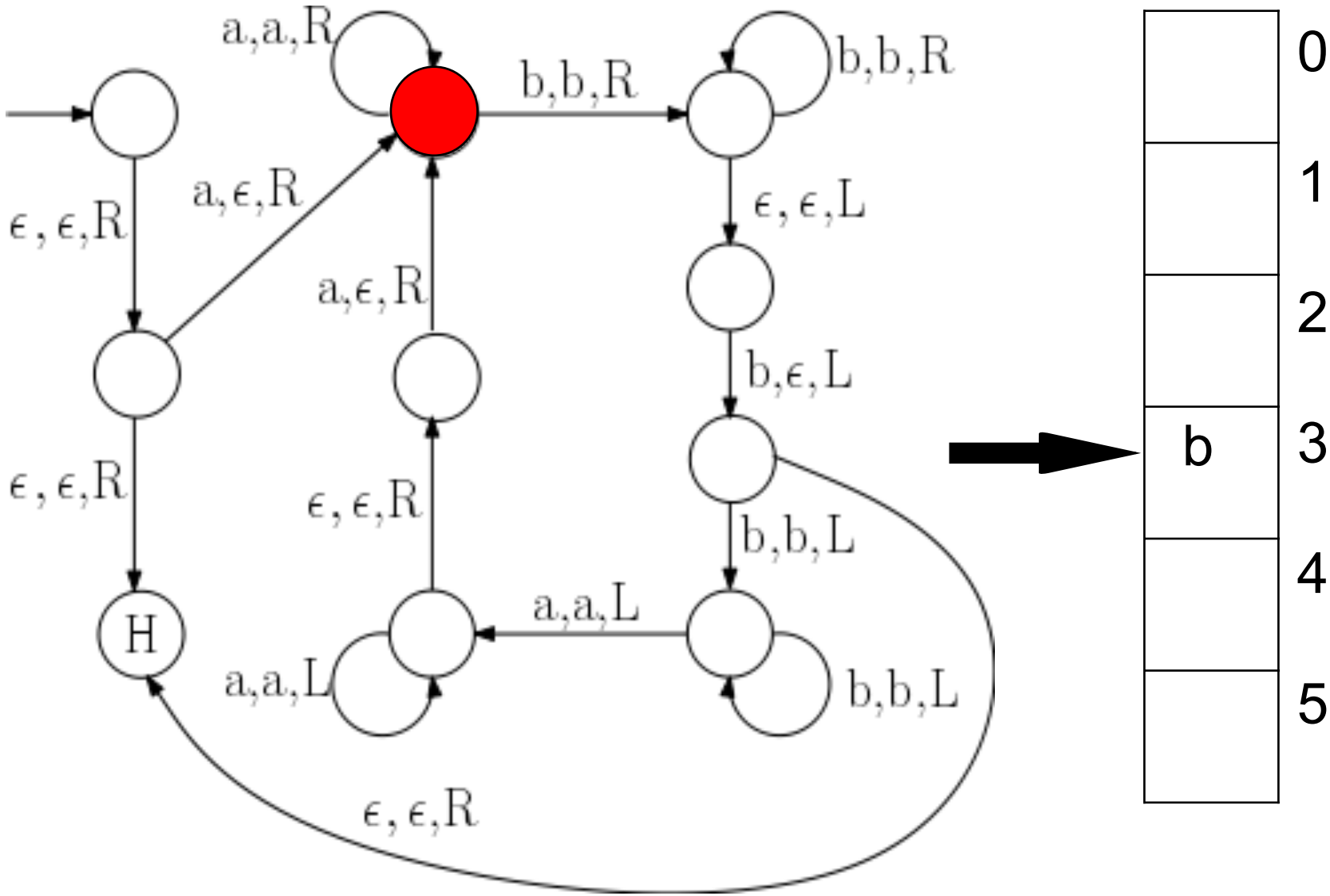
# Example: $a^n b^n$



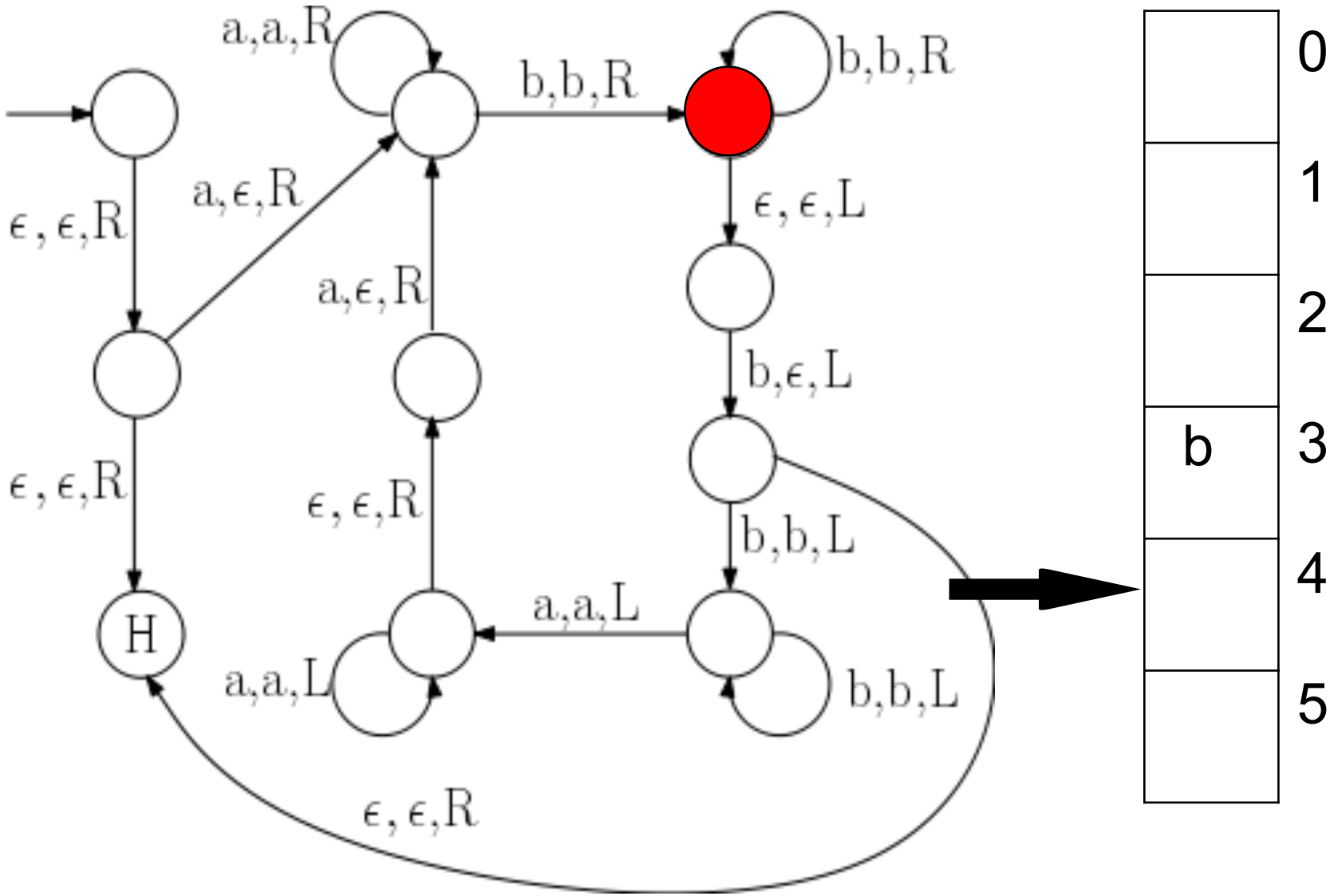
# Example: $a^n b^n$



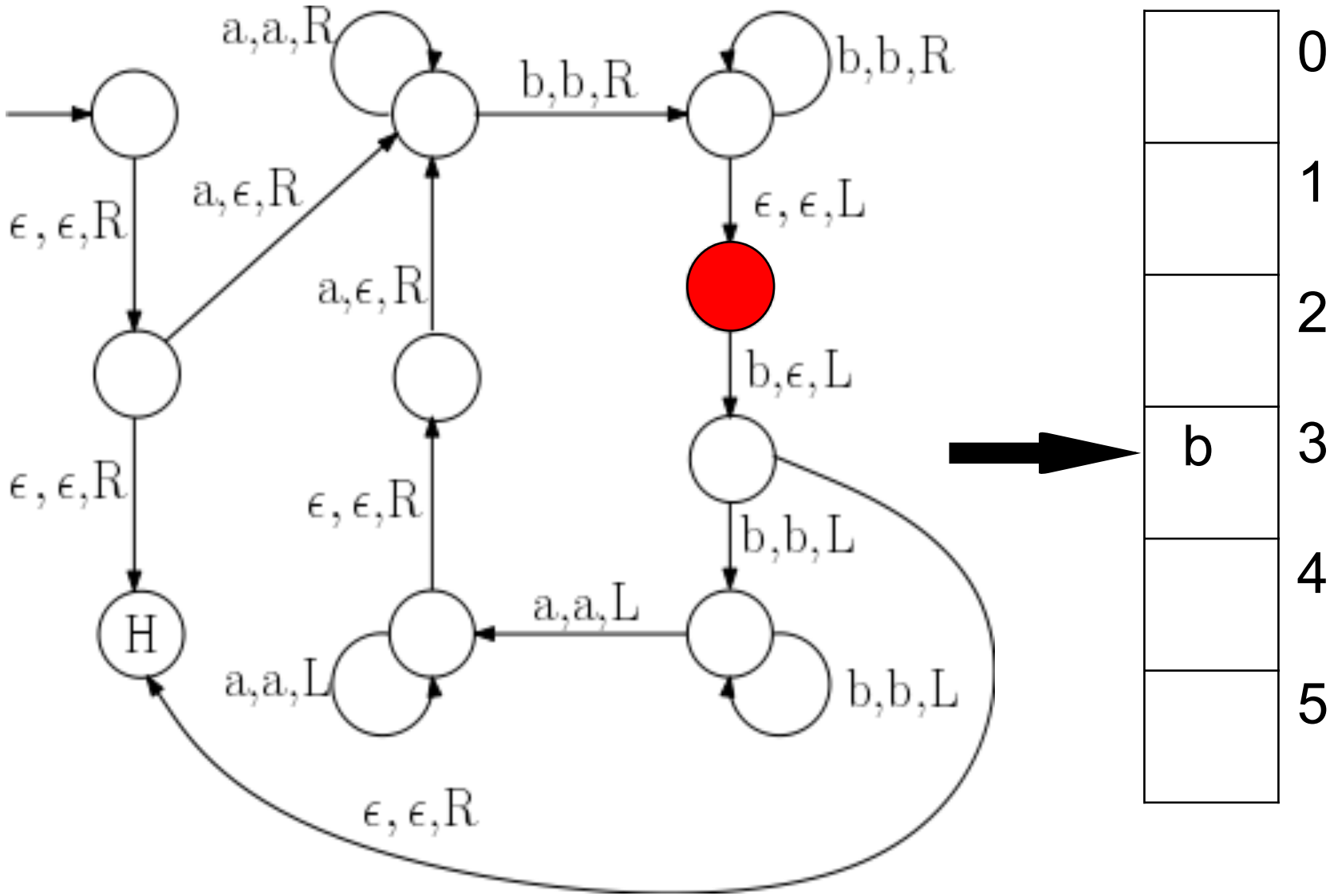
# Example: $a^n b^n$



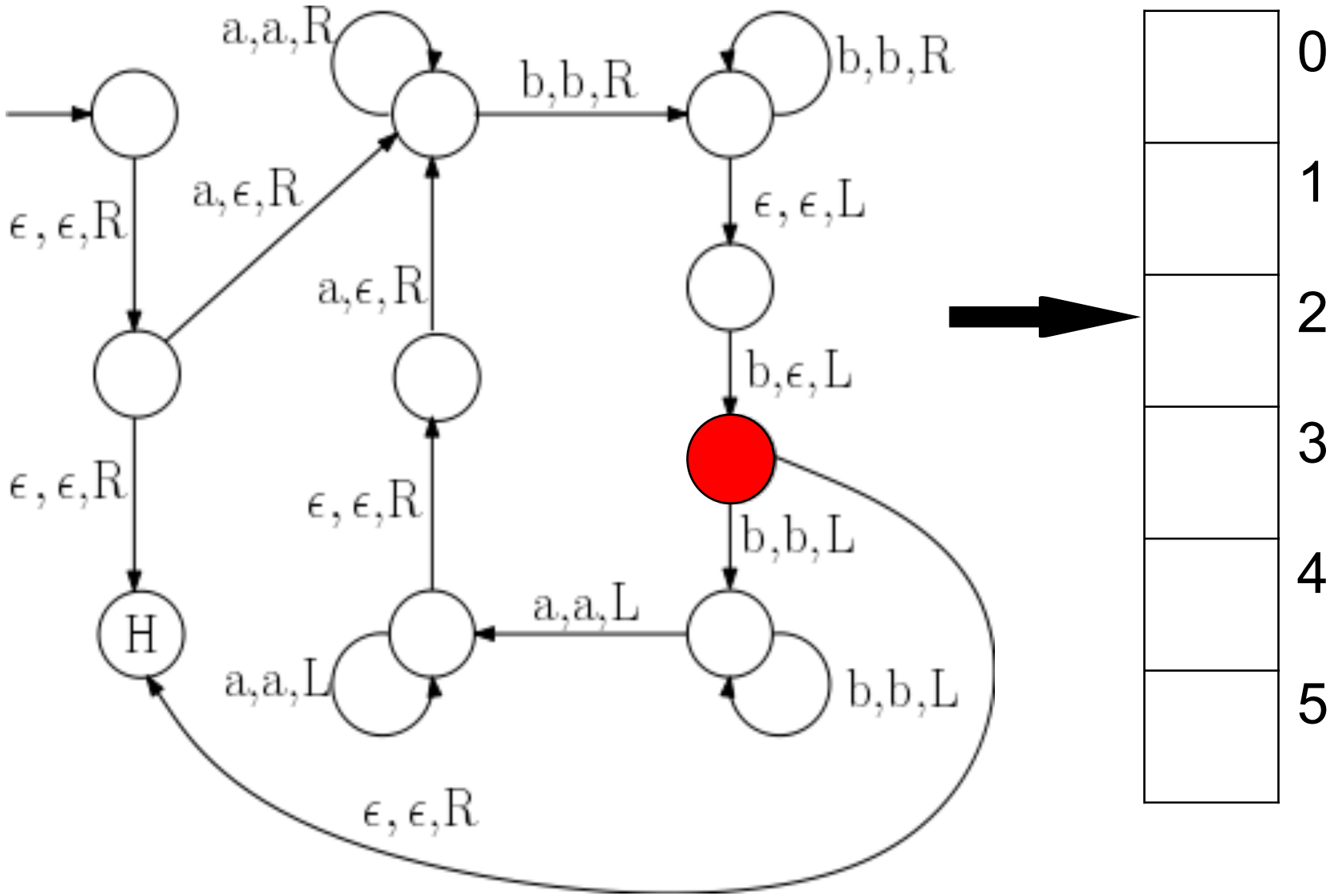
# Example: $a^n b^n$



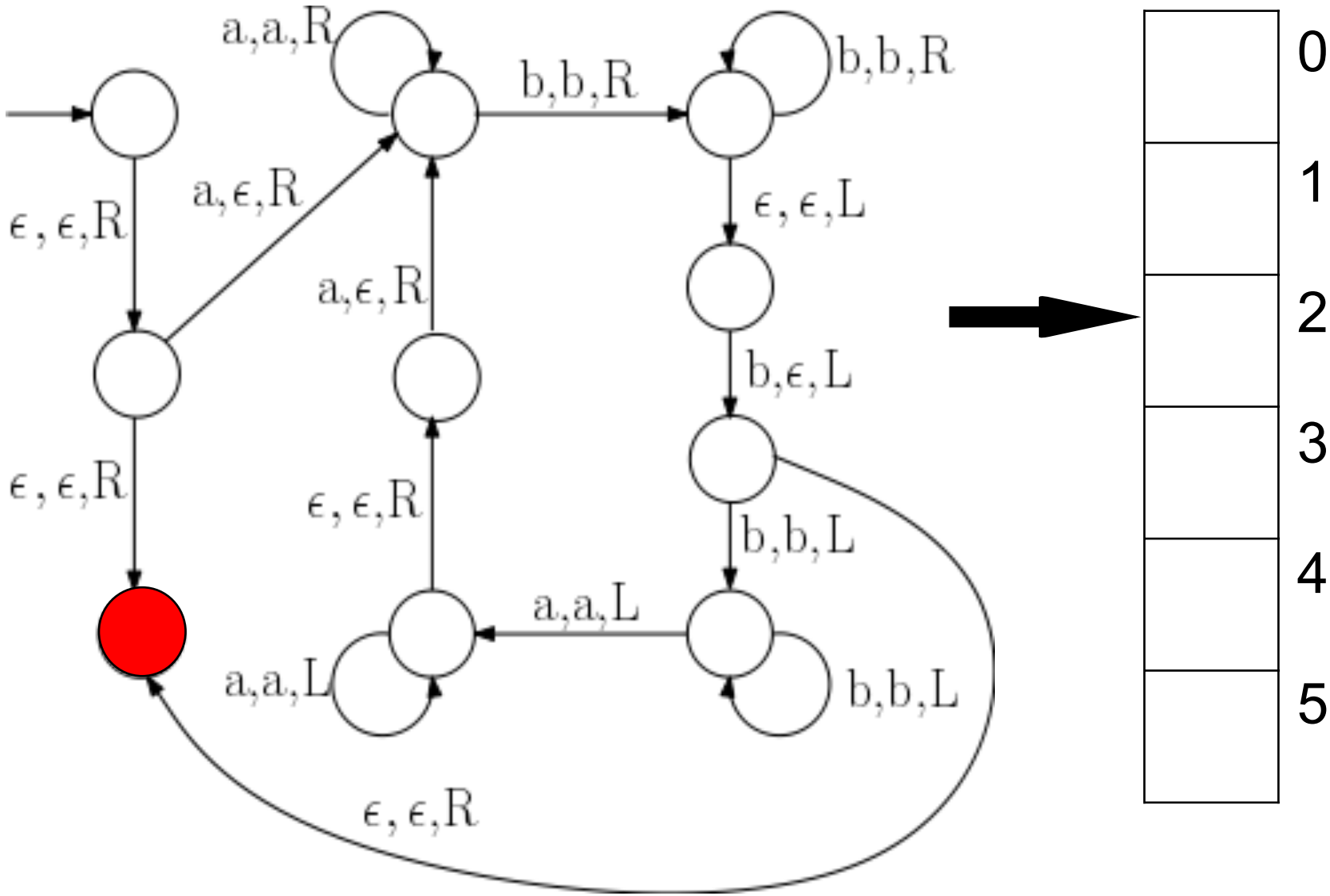
# Example: $a^n b^n$



# Example: $a^n b^n$



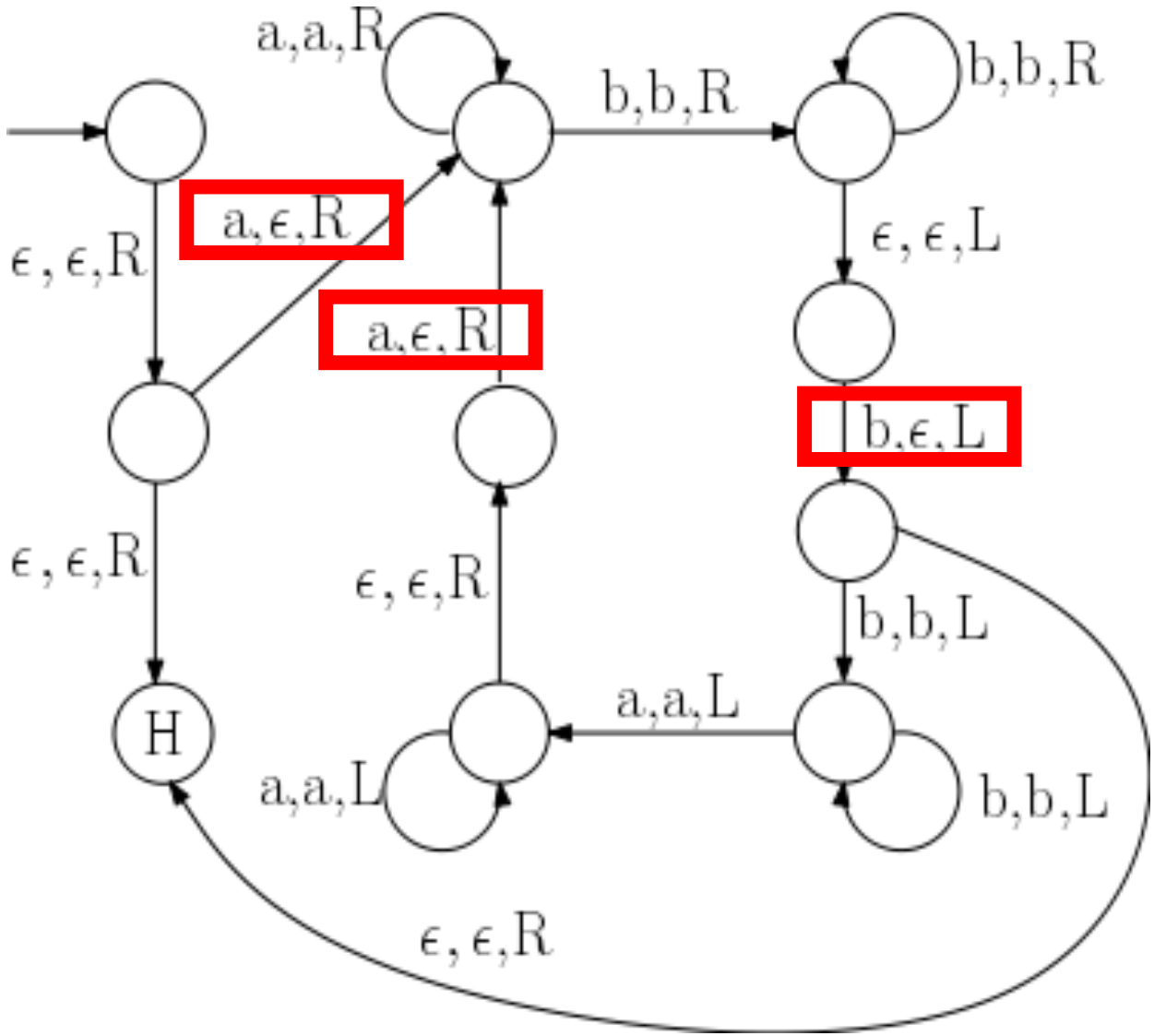
# Example: $a^n b^n$

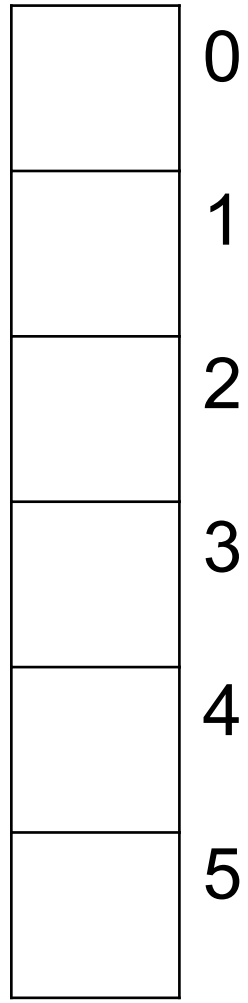


# Correctness

- Every time we reach the bottom-right state, the number of "b"s deleted equals the number of "a"s

# Correctness

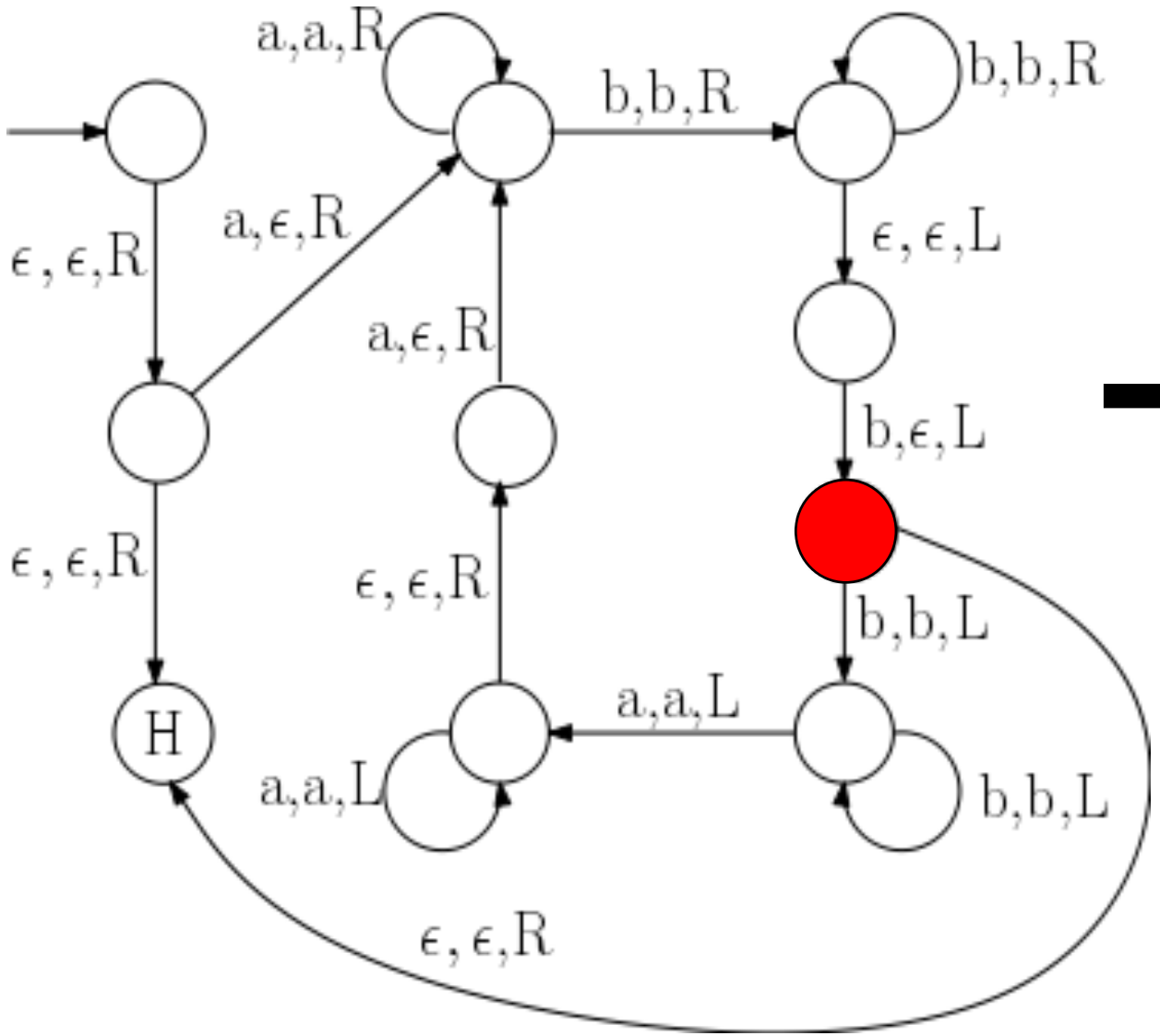


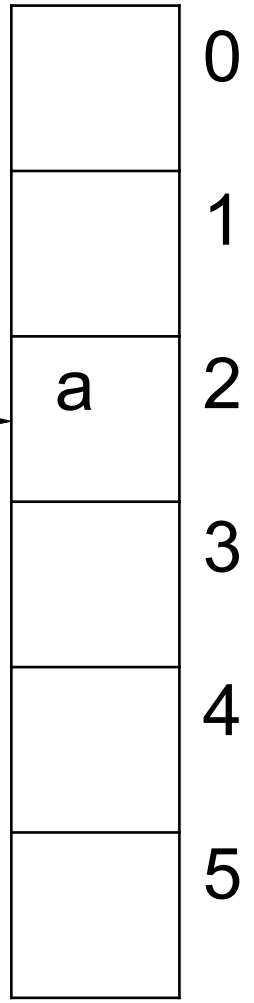



# Correctness

- Every time we reach the bottom-right state, the number of "b"s deleted equals the number of "a"s
- If there are more "a"s, it crashes right after deleting the last "b"

# Correctness



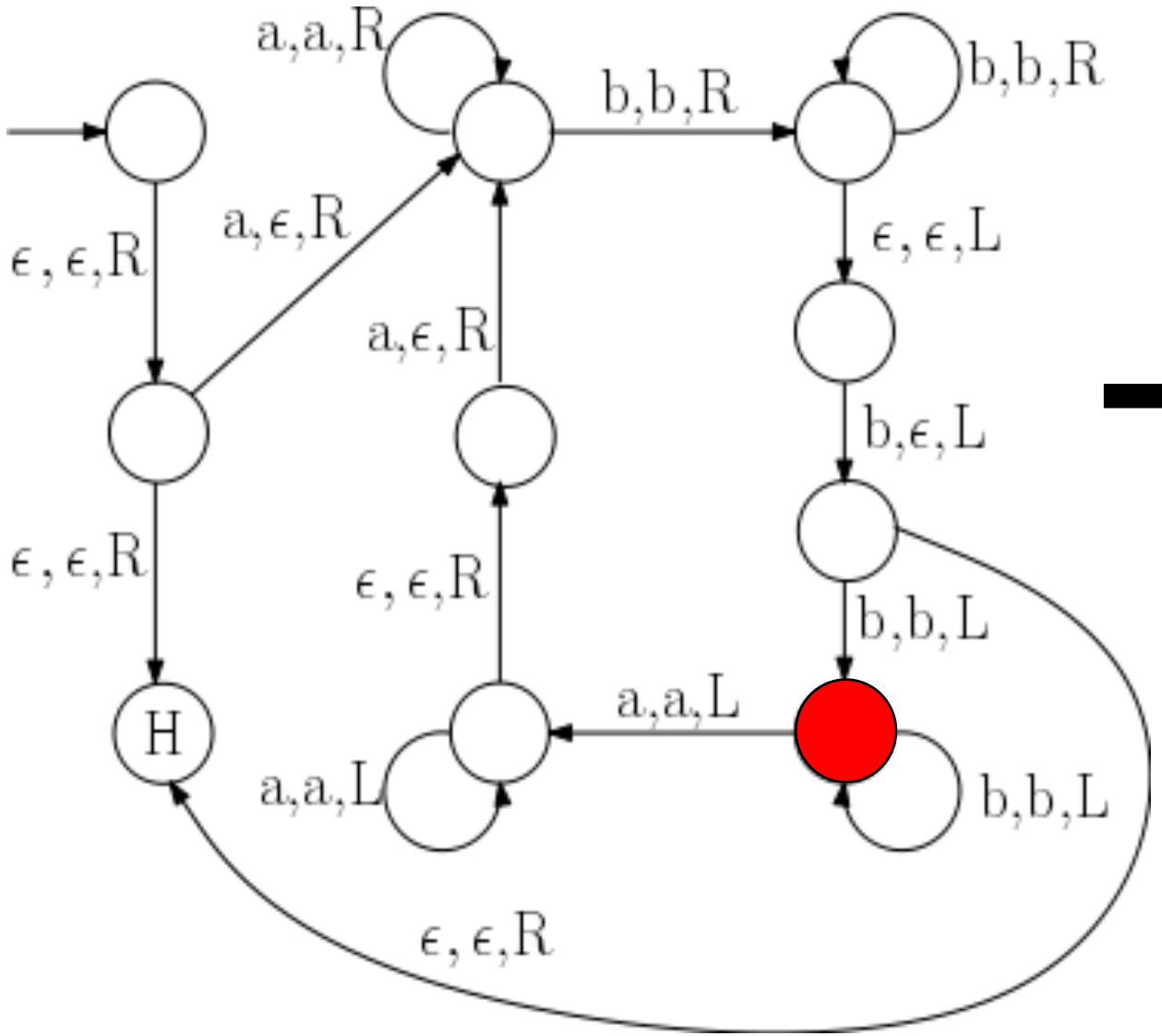



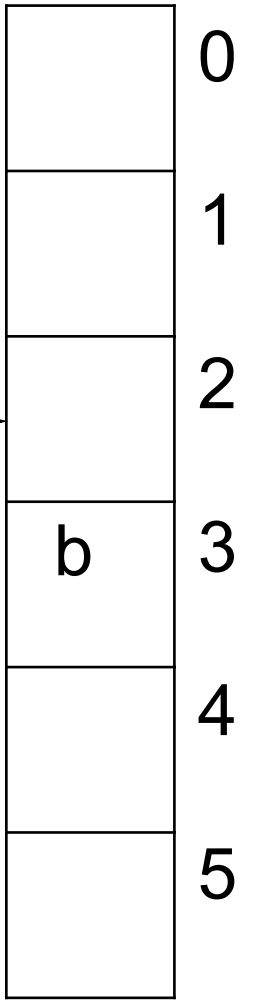

No "a" transition!

# Correctness

- Every time we reach the bottom-right state, the number of "b"s deleted equals the number of "a"s
- If there are more "a"s, it crashes right after deleting the last "b"
- If there are more "b"s, it crashes in the bottom-right state after scanning to the left of the "b"s

# Correctness



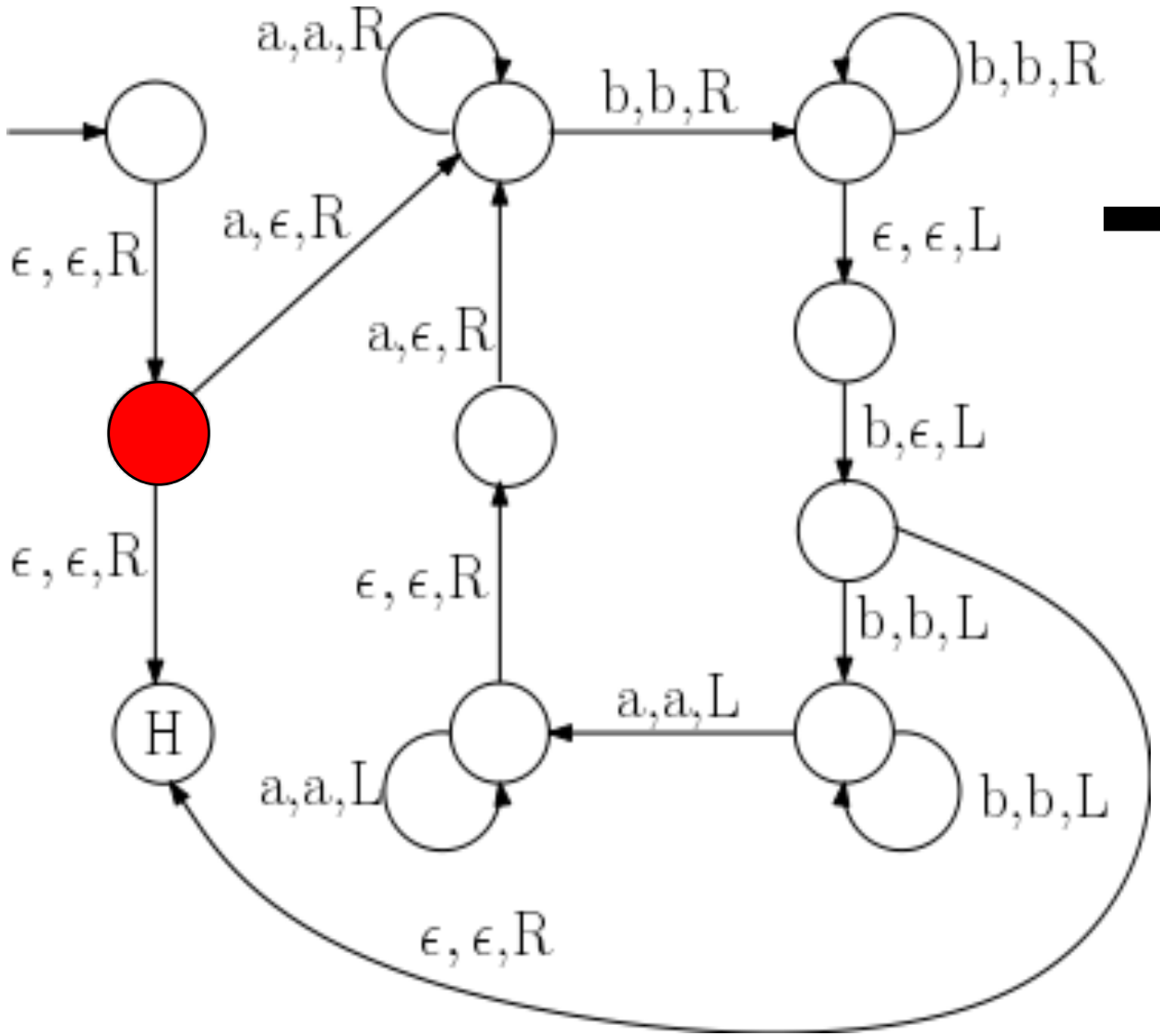



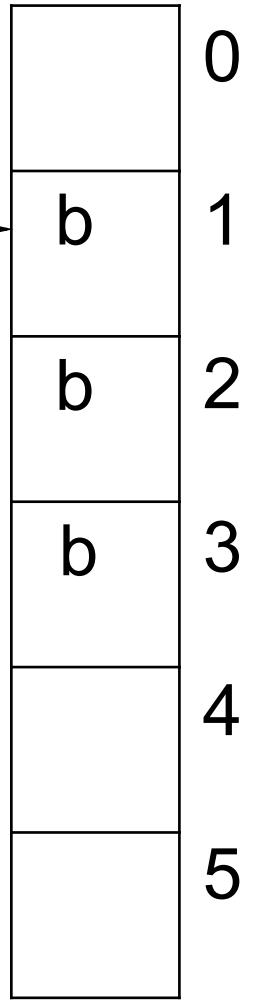

No epsilon transition!

# Correctness

- Every time we reach the bottom-right state, the number of "b"s deleted equals the number of "a"s
- If there are more "a"s, it crashes right after deleting the last "b"
- If there are more "b"s, it crashes in the bottom-right state after scanning to the left of the "b"s
- If NO "a"s, crashes immediately

# Correctness



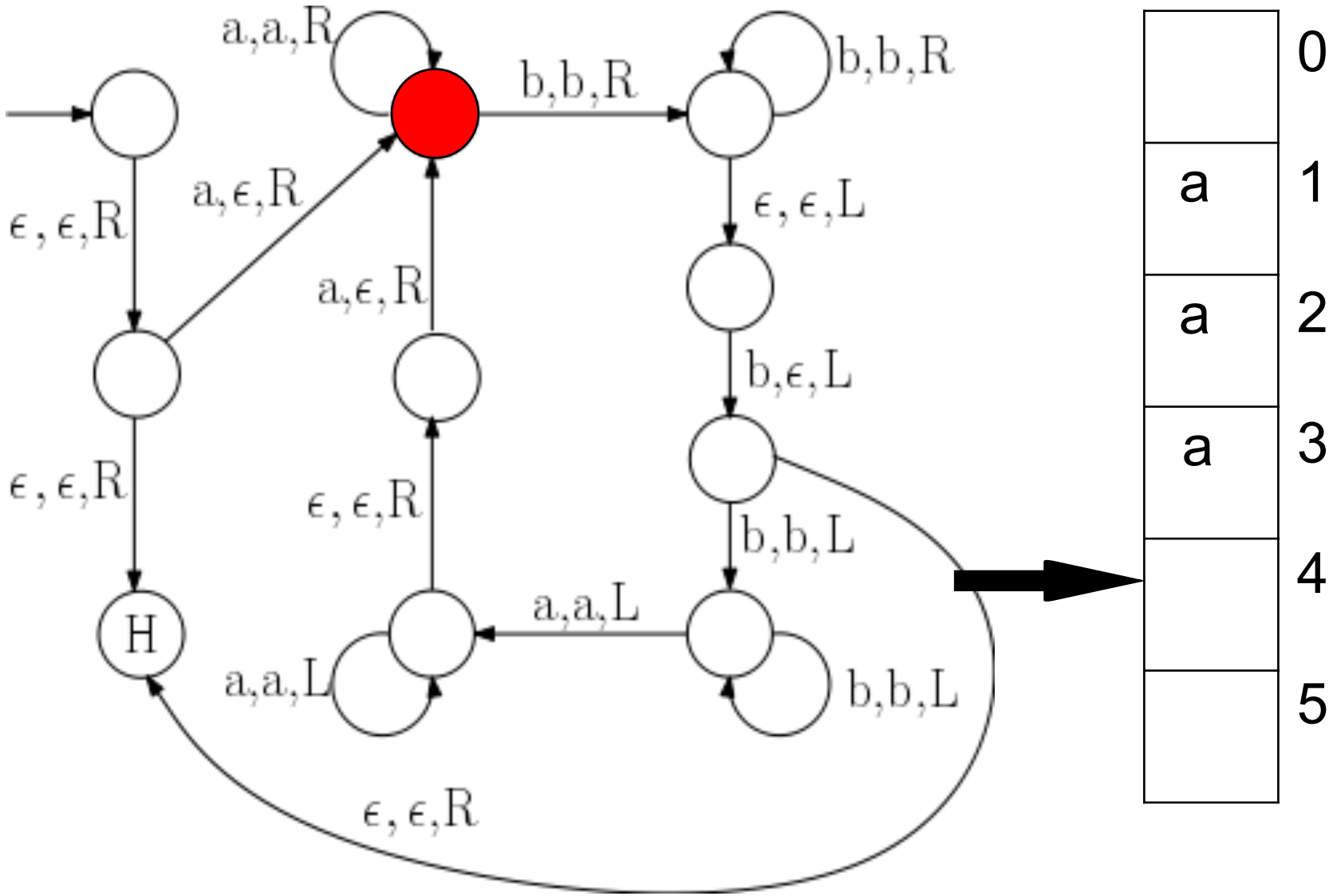




No "b" transition!

# Correctness

- Every time we reach the bottom-right state, the number of "b"s deleted equals the number of "a"s
- If there are more "a"s, it crashes right after deleting the last "b"
- If there are more "b"s, it crashes in the bottom-right state after scanning to the left of the "b"s
- If NO "a"s, crashes immediately
- If no "b"s, crashes after reading the "a"s

# Correctness




No epsilon transition!

# Definitions

- A set is *recursively enumerable* if there is a decision machine which answers "yes" if and only if the input belongs to the set
- A function f is *computable* or *recursive* if there is a function machine that, when given input x, produces output f(x)

# Interlude: What is an "algorithm"?

- "A precise step-by-step plan for a computational procedure that begins with an input value and yields an output value in a finite number of steps."

  -Corbin, Leiserson, Rivest. "Introduction to Algorithms"

- Turing machines can capture this notion precisely!

# Algorithms on a TM

- Given a human-comprehensible goal, encode the input into some alphabet
- Define the steps of the algorithm as a DFA
- Put the input on a tape
- Let the Turing Machine run
- Translate the output

# Objection

- Why not just use a programming language, such as C?
- As it turns out, writing a C program is just using a computer to build a Turing Machine for you

# Algorithms on a TM

- Given a human-comprehensible goal, encode the input into some alphabet
- Define the steps of the algorithm as a DFA
- Put the input on a tape
- Let the Turing Machine run
- Translate the output

# Algorithms in C

- Given a human-comprehensible goal, encode the input into some alphabet
- Define the steps of the algorithm as a DFA
- Put the input on a tape
- Let the Turing Machine run
- Translate the output

# Algorithms in C

- Given a human-comprehensible goal, encode the input into binary
- Define the steps of the algorithm as a DFA
- Put the input on a tape
- Let the Turing Machine run
- Translate the output

# Algorithms in C

- Given a human-comprehensible goal, encode the input into binary
- Define the steps of the algorithm as a sequence of processor instructions
- Put the input on a tape
- Let the Turing Machine run
- Translate the output

# Algorithms in C

- Given a human-comprehensible goal, encode the input into binary
- Define the steps of the algorithm as a sequence of processor instructions
- Put the input in computer memory
- Let the Turing Machine run
- Translate the output

# Algorithms in C

- Given a human-comprehensible goal, encode the input into binary
- Define the steps of the algorithm as a sequence of processor instructions
- Put the input in computer memory
- Let the program run
- Translate the output

# Algorithms in C

- Given a human-comprehensible goal, encode the input into binary
- Define the steps of the algorithm as a sequence of processor instructions
- Put the input in computer memory
- Let the program run
- Translate the output into radiation beamed at the user's face, or ink on a paper

# Extending the parallel: 15-213 in 60 seconds

- String of 1's and 0's in memory representing the arguments to a function (tape)
- Handful of registers that point to locations in memory (head(s))
- Block of memory describing instructions (controller transition function)
- Special register, %eip, pointing to current instruction (controller state)

# More Objections

- Can't computer programs do more than Turing Machines can?
  - Jump to arbitrary memory address
  - Wait for input that wasn't given at the program's start (e.g. keyboard)

# Jumping to an arbitrary address: Turing Machines can do that!

- PC: access the memory at byte with address *0xdeadface*
- TM: Encode address in unary (i.e. block of "0xdeadface" ones)
- Shift the unary number so it starts at 0
- Scan through the segment to find the right endpoint. The head will be at cell 0+0xdeadface, as desired

# Sub-objection?

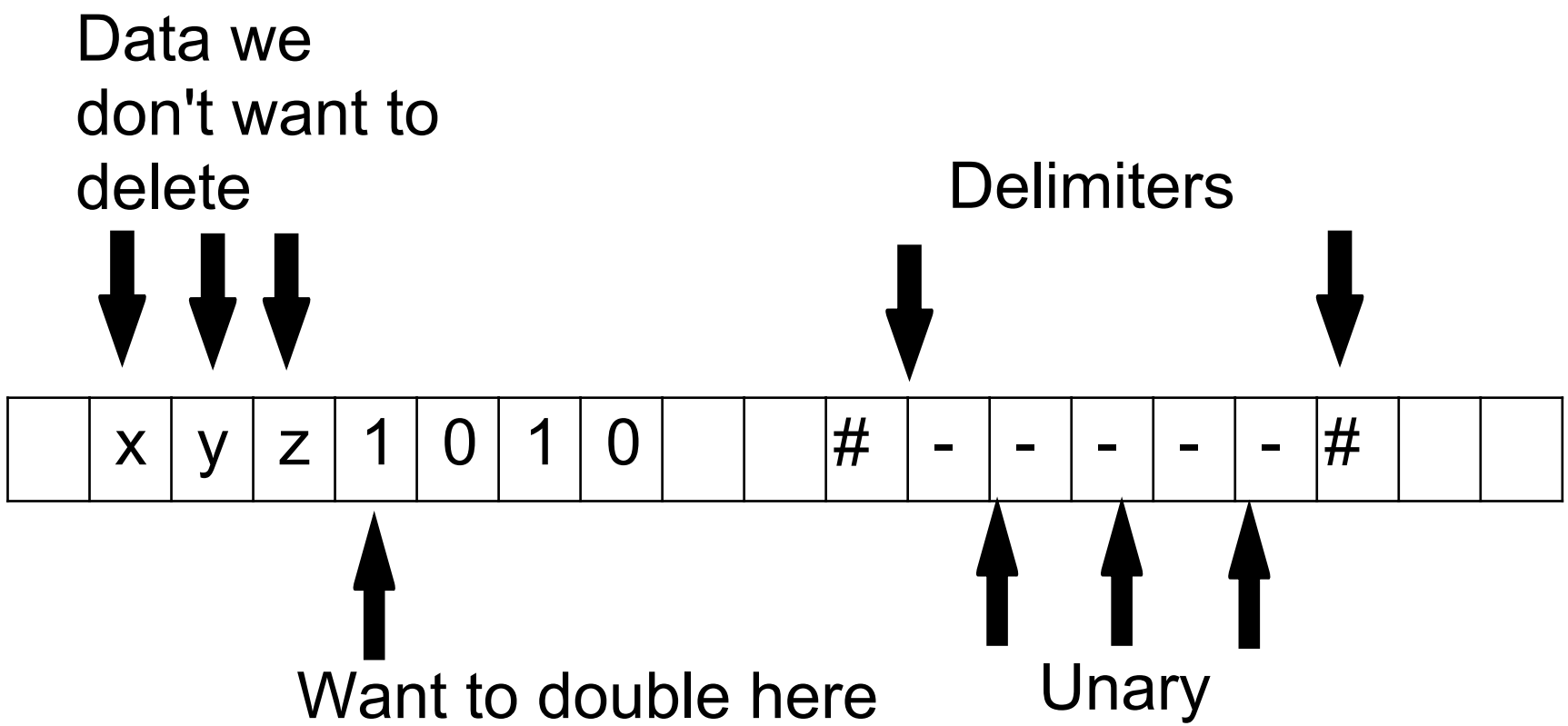
- How do you mark a block of cells as representing a single variable?
  - Define more symbols. Make a special "marker" symbol to indicate the endpoint of any region you want
- How do you shift the variable without overwriting other data?
  - Define an extra bit for each symbol, used exclusively to store the information "is this cell part of the unary sequence?"

# Example: Doubling a Number

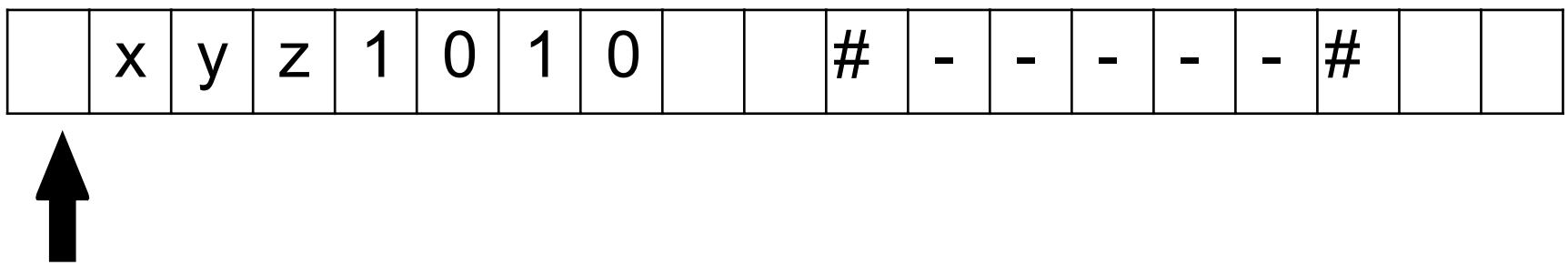
```
byte x, y, z; //compiler stores at addresses 1,2,3
int n=5; //compiler stores at bytes 4-7
int* charmander = &n; //contains value "4",
    //stored at location 11

void double(){
    int** metapod = &charmander; //metapod = 11
    **metapod *= 2;  //Finds value 4 at memory
            //location 11, goes to the memory location
            //referenced, and doubles the value found
            //there
}
```

# Example: Doubling a Number

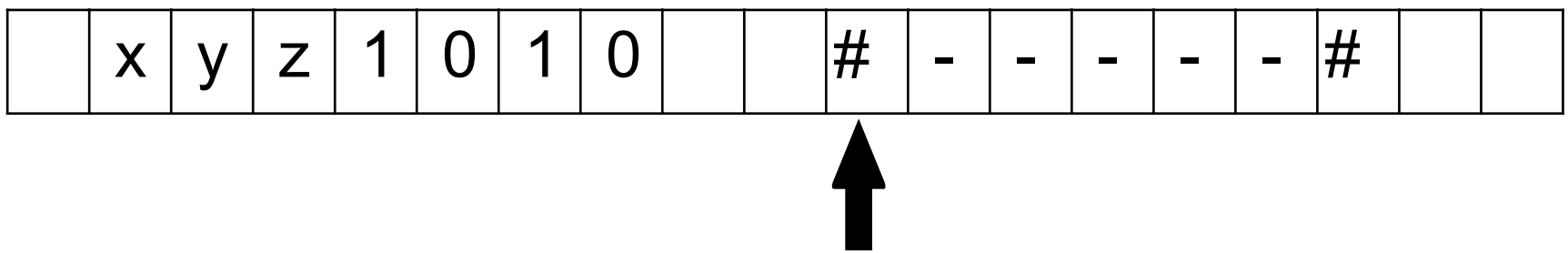


# Example: Doubling a Number



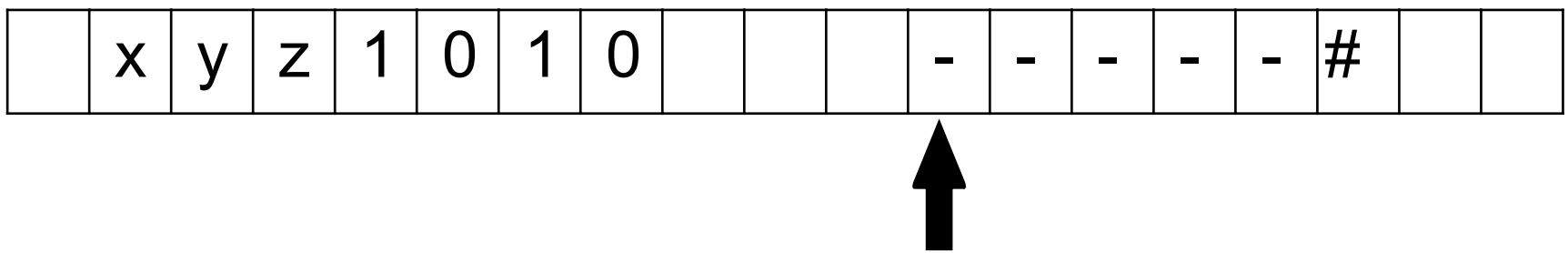
# Example: Doubling a Number

- Copy "-" over



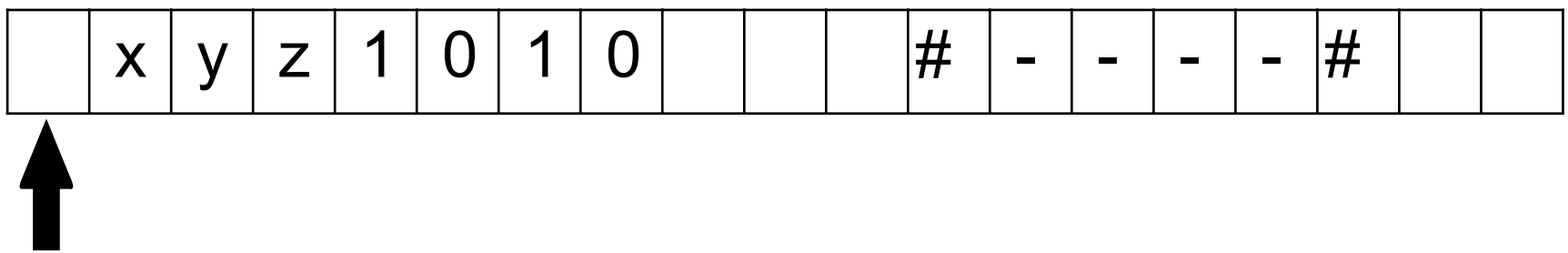
# Example: Doubling a Number

- Copy "-" over



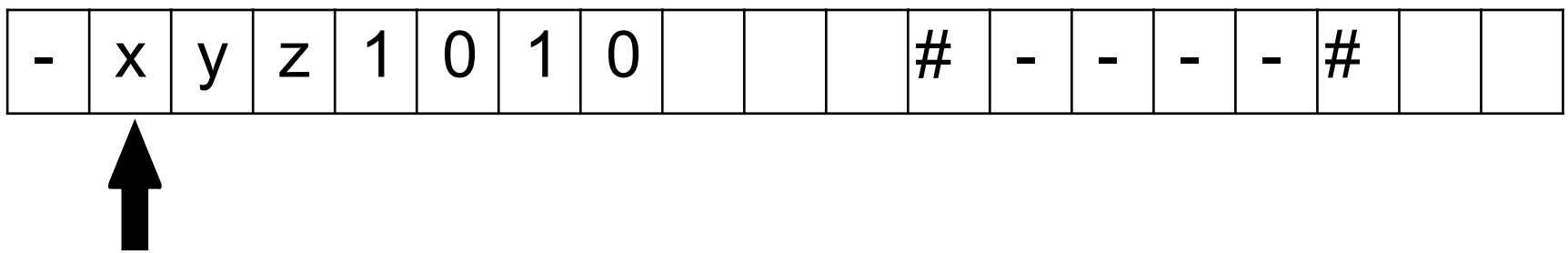
# Example: Doubling a Number

- Copy "-" over



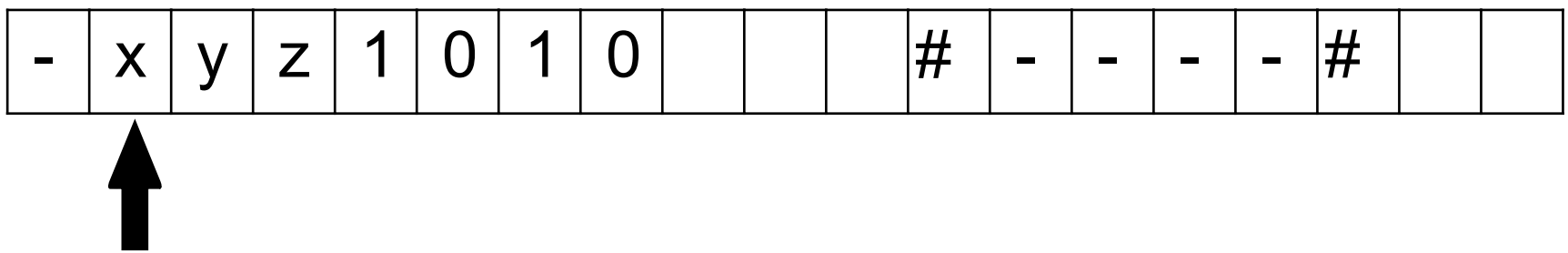
# Example: Doubling a Number

- Copy "-" over



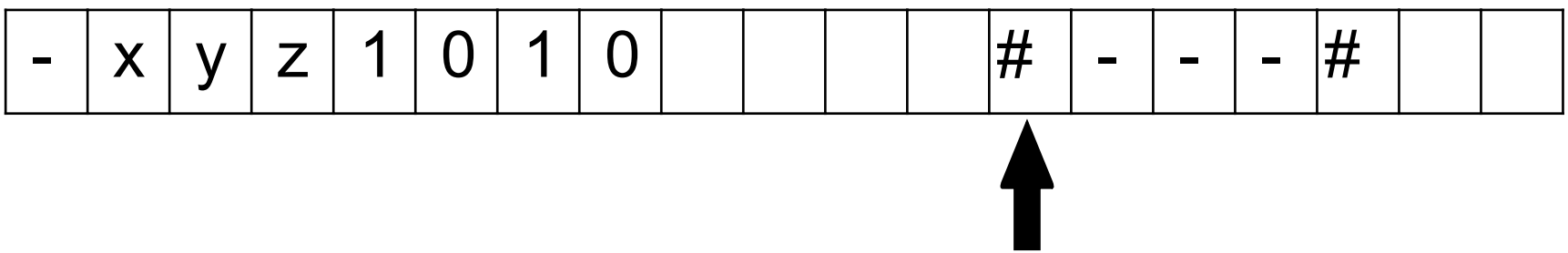
# Example: Doubling a Number

- Copy "-" over
- Repeat, without deleting old data



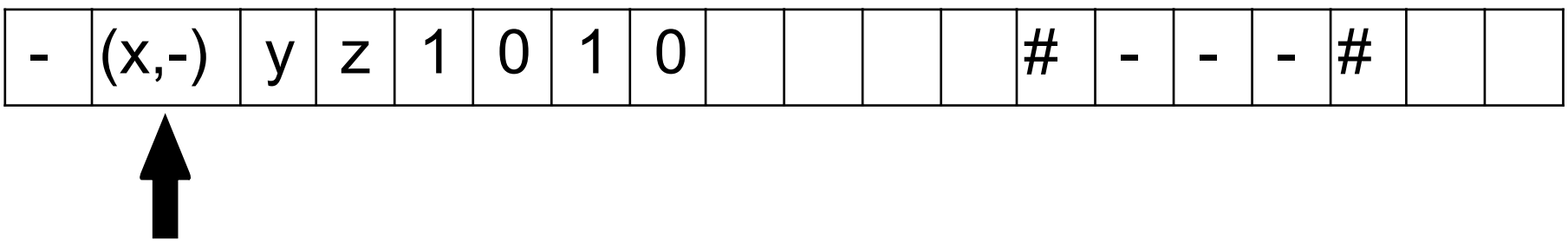
# Example: Doubling a Number

- Copy "-" over
- Repeat, without deleting old data



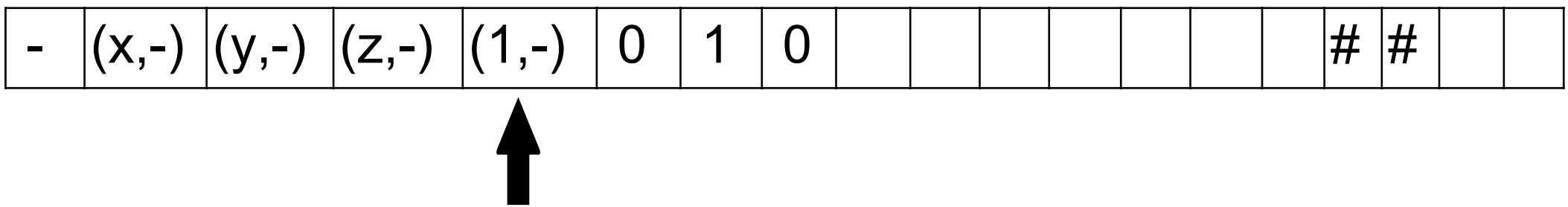
# Example: Doubling a Number

- Copy "-" over
- Repeat, without deleting old data



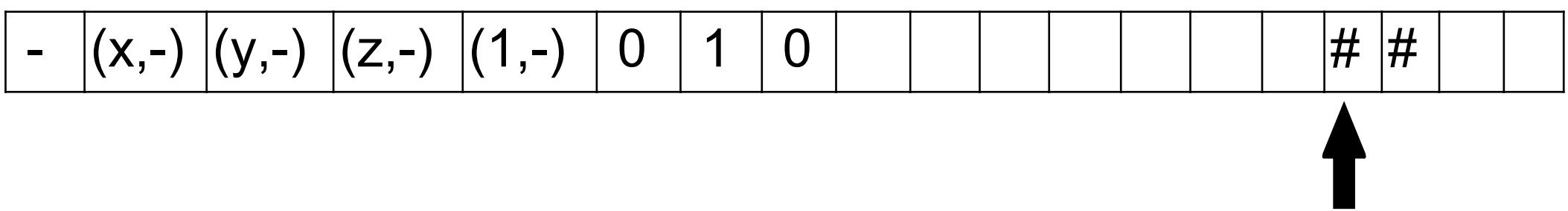
# Example: Doubling a Number

- Copy "-" over
- Repeat, without deleting old data



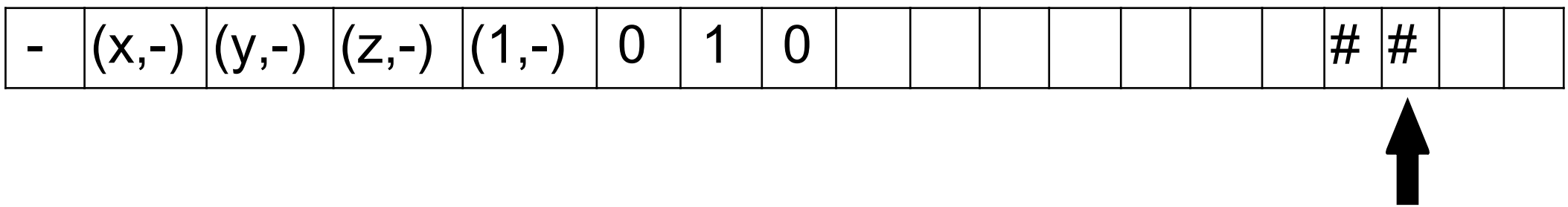
# Example: Doubling a Number

- Copy "-" over
- Repeat, without deleting old data
- Two "#" in a row indicate shift complete



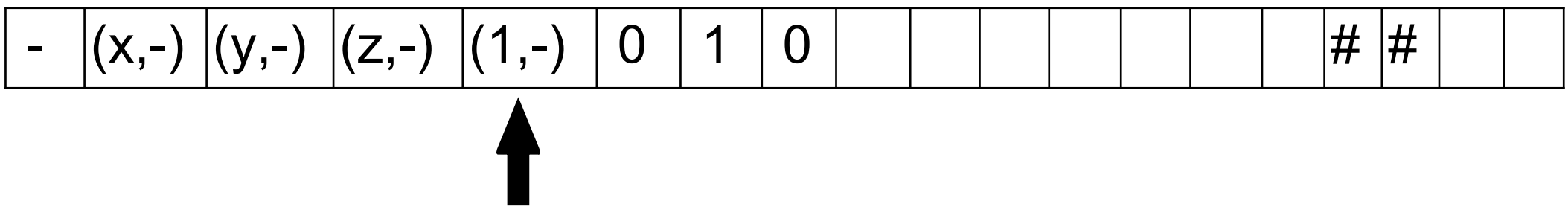
# Example: Doubling a Number

- Copy "-" over
- Repeat, without deleting old data
- Two "#" in a row indicate shift complete



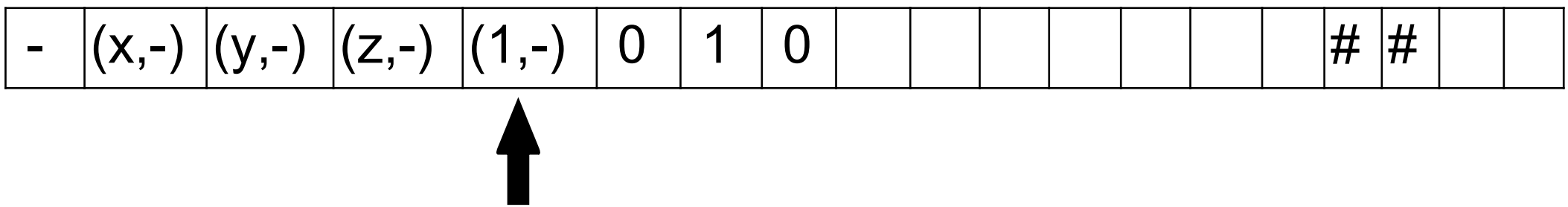
# Example: Doubling a Number

- Copy "-" over
- Repeat, without deleting old data
- Two "#" in a row indicate shift complete
- Scan left till you find a "-"



# Example: Doubling a Number

- Copy "-" over
- Repeat, without deleting old data
- Two "#" in a row indicate shift complete
- Scan left till you find a "-"
- Double as before



# Waiting for input: Cheat

- Allow user to edit symbols on the tape while the TM is running
- Consider a state that does nothing on an empty symbol, moves right on anything else
- When it sees a blank symbol, it's stuck until the user changes it
- Note: For doing this, it's convenient to allow the head to stay put on a transition, but not necessary

# C vs Turing Machine

- Any algorithm that can be done in C can be done with a Turing Machine
- Any algorithm that can be done on a Turing Machine can be done in C (easy to write a TM simulator)
- Thus, C and Turing Machines have the same computational power

# How about Java?

- It is also easy to simulate a TM in Java
- Thus, it can simulate a Turing Machine that simulates C!
- Java has at least as much computational power as C
- Java can also be simulated on a TM
- Thus, C can simulate a Turing Machine simulating Java
- Therefore, C and Java have the same computational power!

# Generalizing

- In fact, a Turing Machine can simulate programs from ANY language that will run on a PC
- Just like with C; nothing but RAM and registers
- Thus, any programming language in which it is possible to simulate a Turing Machine has the same power as any other language that can simulate a Turing Machine

# Definition

- A computational system is said to be *Turing-complete* if it has at least as much computational power as a Turing Machine

# Turing-completeness

- Many, many models of computation are Turing-complete:
  - Several other theoretical models (e.g. Post productions, Lambda Calculus)
  - Any mainstream programming language
  - LaTeX
  - The C pre-processor
  - Legos
  - The Starcraft Map Editor

# Meta-simulations

- Any Turing Machine can be simulated in some C program
- Any C program can be simulated in some Turing Machine
- Thus, you can have a Turing Machine simulating a C program simulating a Turing Machine...
- Or, a C program simulating a Turing Machine simulating a C program

# Definition

- A *Universal Turing Machine* is a TM that can simulate the behavior of any other Turing Machine
- Analogous to a Virtual Machine, Interpreter, or Operating System
- "Yo dawg, I heard you liek Turing Machines, so we put a Turing Machine in your Turing Machine so you can recurse while you recurse."

# Universal Turing Machine

- Universal Machines can be as small as 4 states, and have an alphabet as small as 6 symbols. Stores nearly all data on tape
- However, It takes much longer to simulate the TM than to run it directly

# How Universal is it?

- So, can a Universal Turing Machine compute every possible function?
- No!
  - Halting problem
  - Busy Beaver function
  - Rice's Theorem
- Wait until next week for proof

# Interlude: History Lesson

- In fact, Turing Machines were formulated to prove there were undecidable mathematical problems
- Developed as an answer to Hilbert's "Entscheidungsproblem"
- This, along with Alonzo Church's Lambda Calculus, established the field of modern theoretical computer science

# Can we do better?

- Can we modify our definition so that it can compute more functions?
  - More tape heads?
    - can simulate multi-threading with just one head
  - More tapes?
    - already have infinite space
  - Non-deterministic controller?
    - NFA's don't give any more computational power

# Can anything else do better?

- Church-Turing thesis: "Any function that is computable in the intuitive sense is computable by a Universal Turing Machine"
  -Alonzo Church and Alan Turing

# Can anything else do better?

- The*sis* , not theo*rem*
- Implication: The universe is a giant Turing Machine
- Theological question, not mathematical!
- Would solve issues such as
  - Free will
  - Possibility of an omnipotent god
  - Possibility of an omniscient god
  - Absolute truth

# What you should know:

- Two types of Turing Machines
- Computable function, R.E. set
- Turing-completeness
- Turing Machine/Computer Program Relationship
- Universal Turing Machine
- Church-Turing Thesis