TURING MACHINES

Introduction to Turing Machine

Finite Automata

- Regular Languages

Pushdown Automata

- Context-Free Languages

Finite Automata

- Regular Languages

Pushdown Automata

- Context-Free Languages

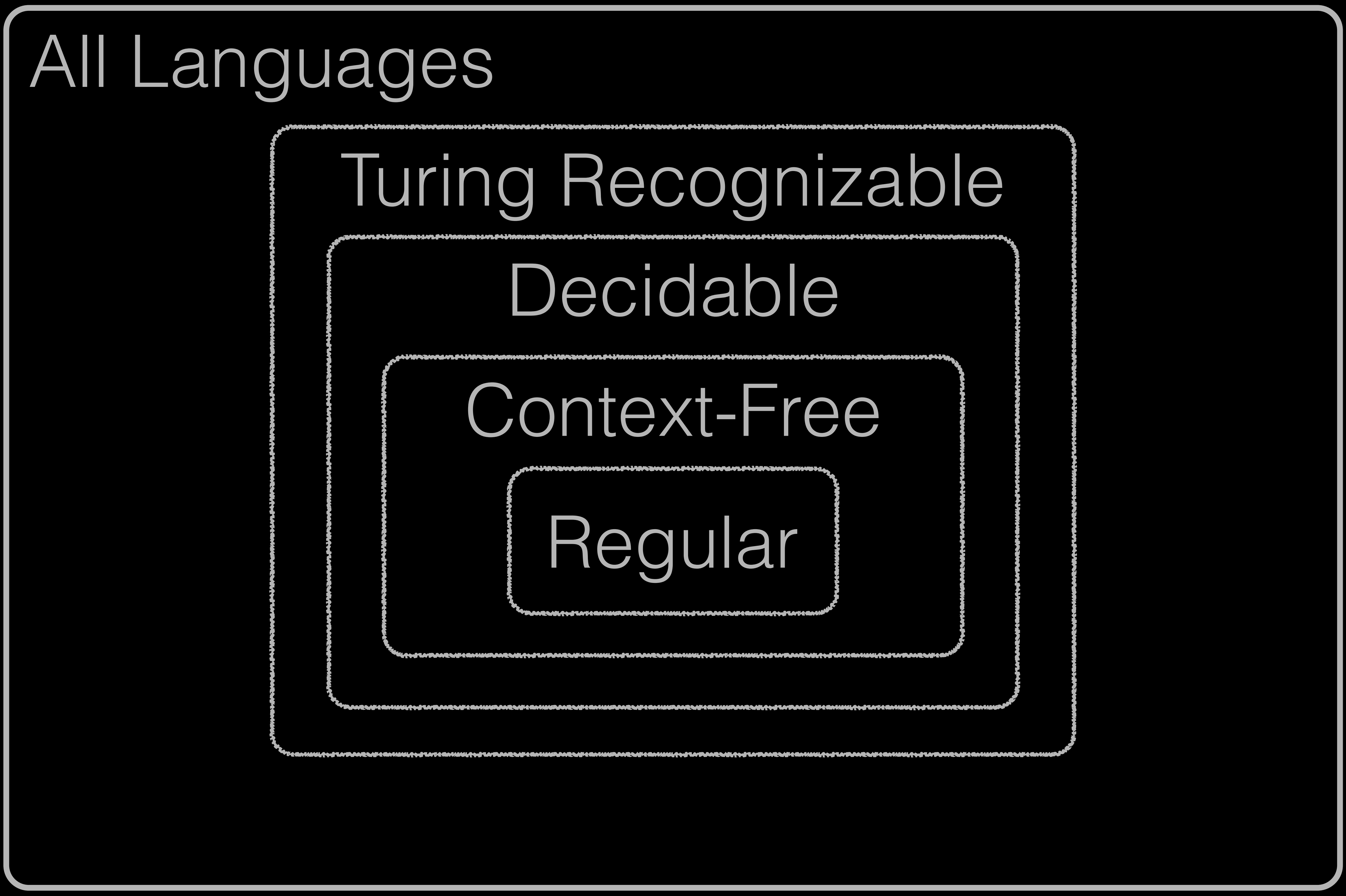
TM: Turing Machines

- A new model of computation
- Not much more elaborate
- A “model” for all computers

TM: Turing Machines

- “decidable” languages
- “Turing recognizable” languages
- languages that are not “Turing recognizable”

CLASSES OF LANGUAGES



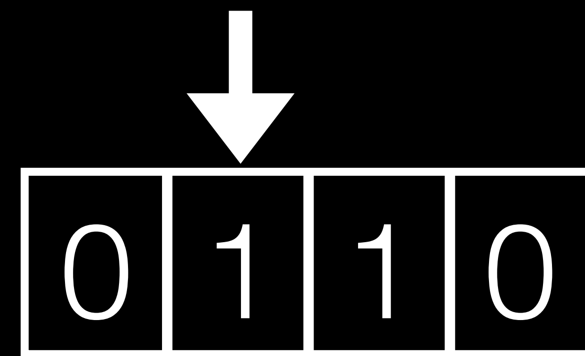
Note: There are variations in the exact definition from textbook to textbook.

All variations are **equivalent.**

DATA STRUCTURES

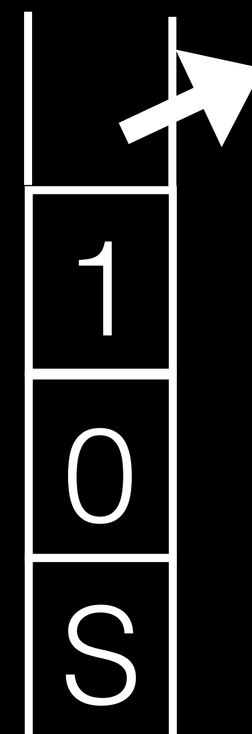
FSM

- the input string



PDA

- the input string
- a stack



TM

- a “tape”

Typical: $\Sigma = \{0, 1\}$

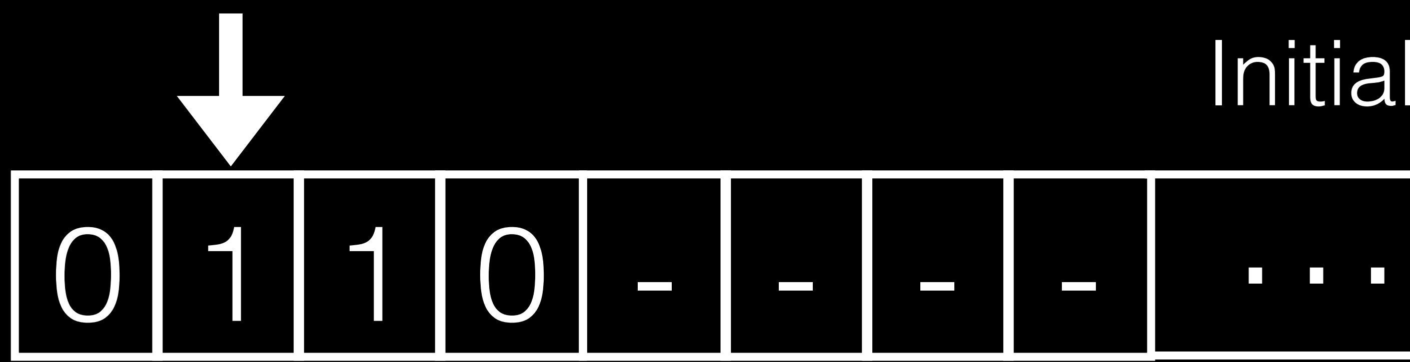
But also common: $\{0, 1, a, b, X, Y, \$\}$

The "blank" symbol is special

$$\text{\_} \notin \Sigma$$

TAPE ALPHABET

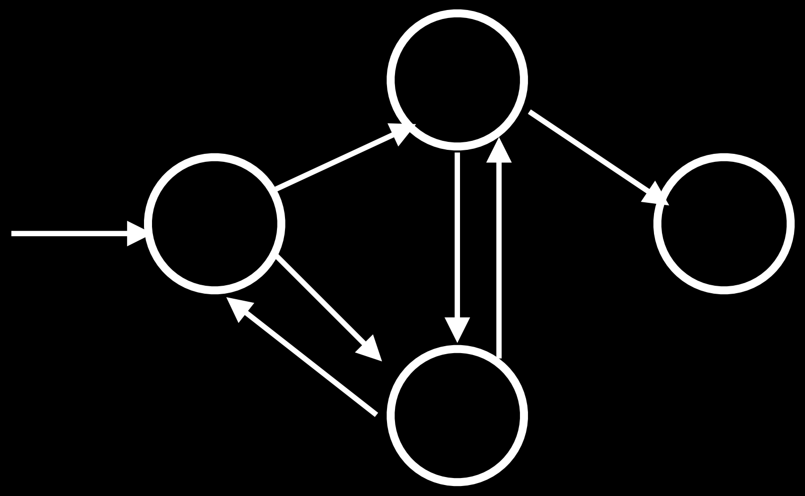
Initial configuration



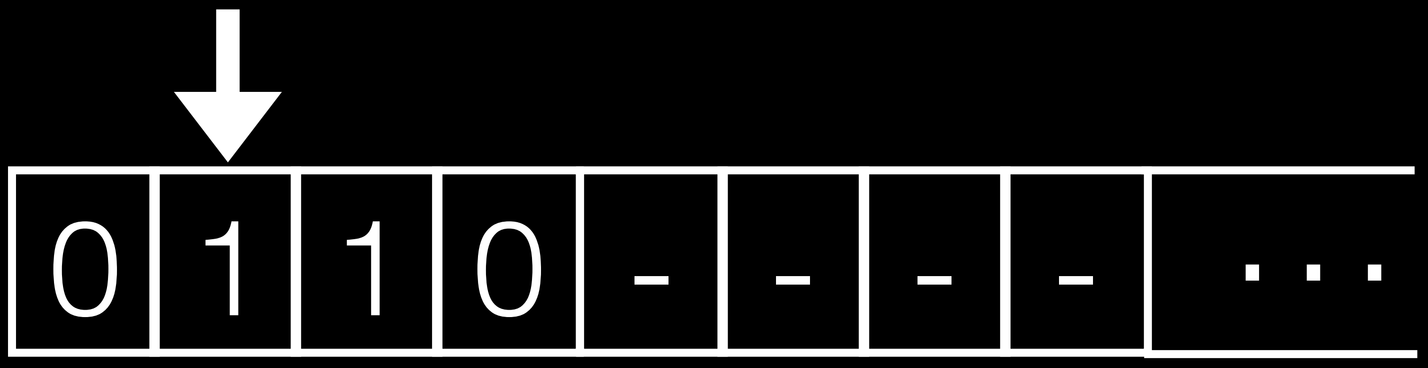
The current position (the tape head)

- initially at the leftmost cell
- can move left or right
- can read (“scan”) the current symbol
- can write the symbol to current position

TAPE ALPHABET



The control portion
Similar to a FSM or PDA
The “program”
Deterministic

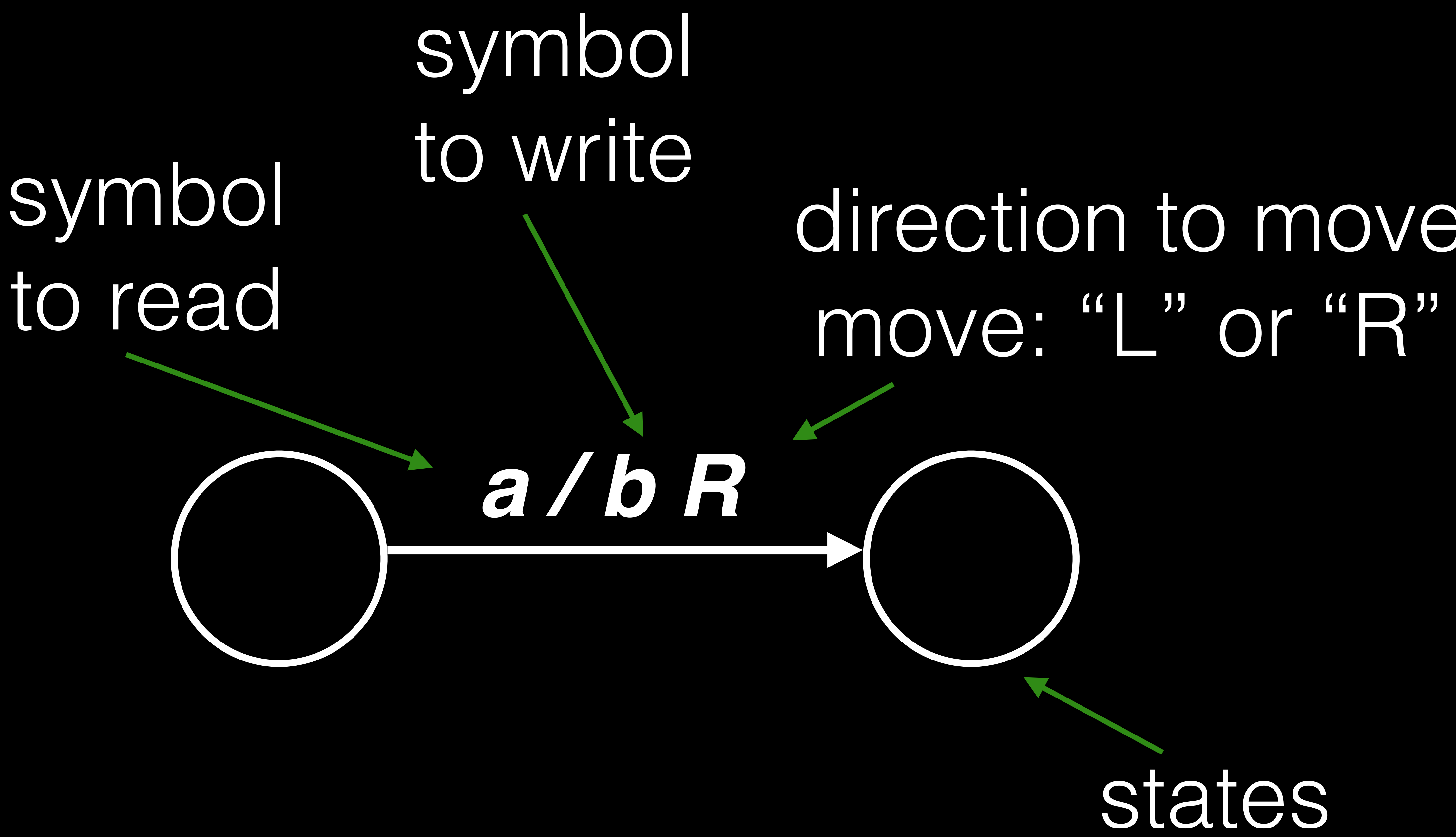


RULES OF OPERATION

At each step of the computation:

- Read the current symbol
- Update (i.e. write) the same cell
- Move exactly one cell either left or right
  - If we are at the left end of the tape and trying to move left, then do not move; stay at the left end.

RULES OF OPERATION



Don’t want to update the cell?

- Just write the same symbol.

RULES OF OPERATION

- Control is with a sort of finite state machine.
- Initial state
- Final states: exactly 2 final states
  - The “accept” state
  - The “reject” state

RULES OF OPERATION

Computation can…

- **halt and "accept"**
  - Whenever the machine enters the "accept" state, computation immediately halts
- **halt and "reject"**
  - Whenever the machine enters the "reject" state, computation immediately halts
- "**loop**"
  - The machine fails to halt

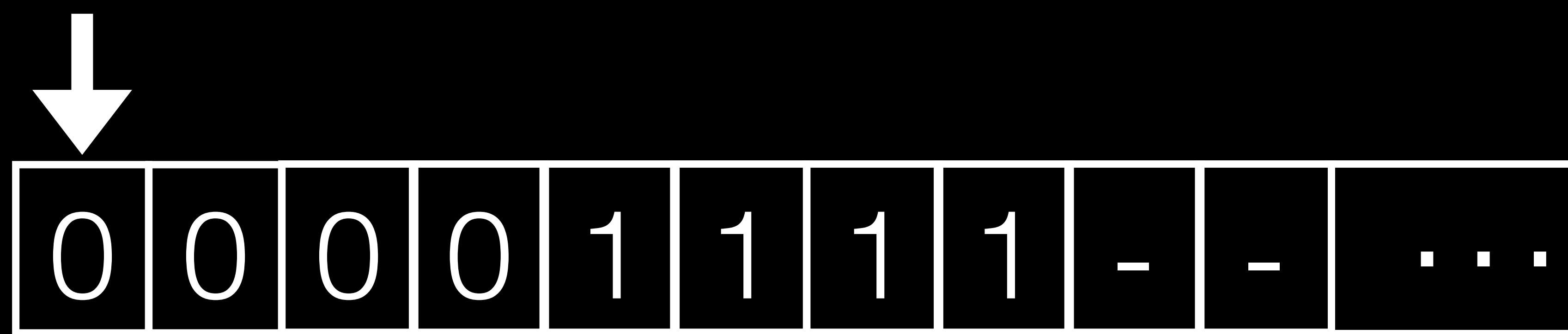
RULES OF OPERATION

The TM is **deterministic**.

Turing Machine Examples

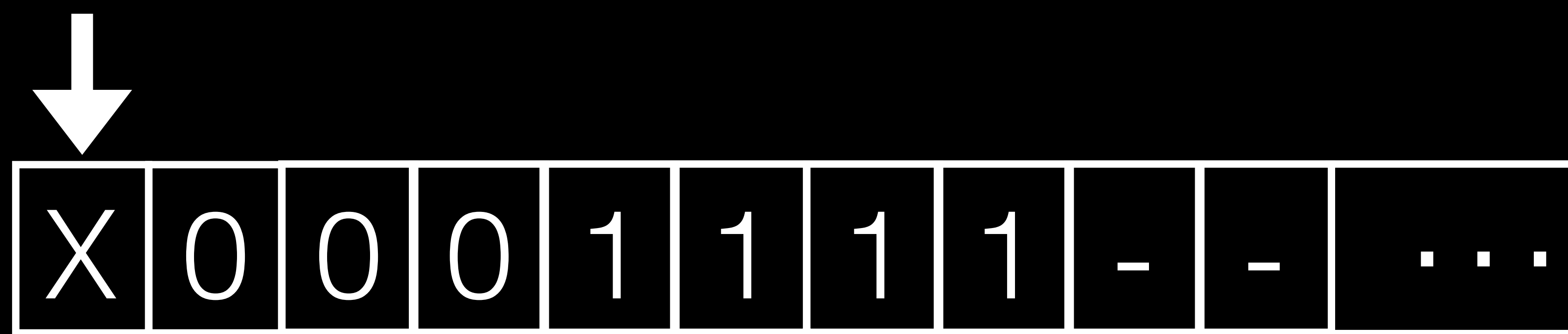
EXAMPLE

$$L = 0^n 1^n$$



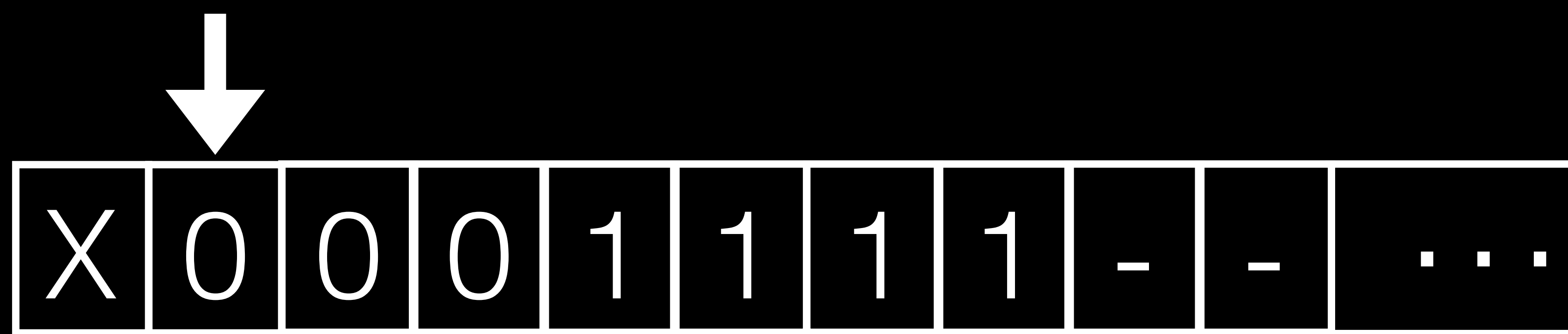
EXAMPLE

$L = 0^n 1^n$



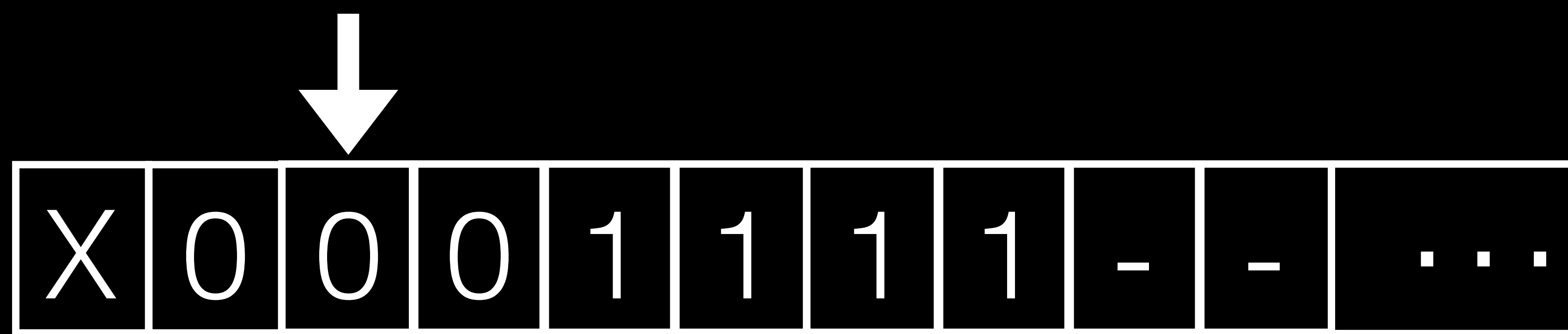
EXAMPLE

$$L = 0^n1^n$$



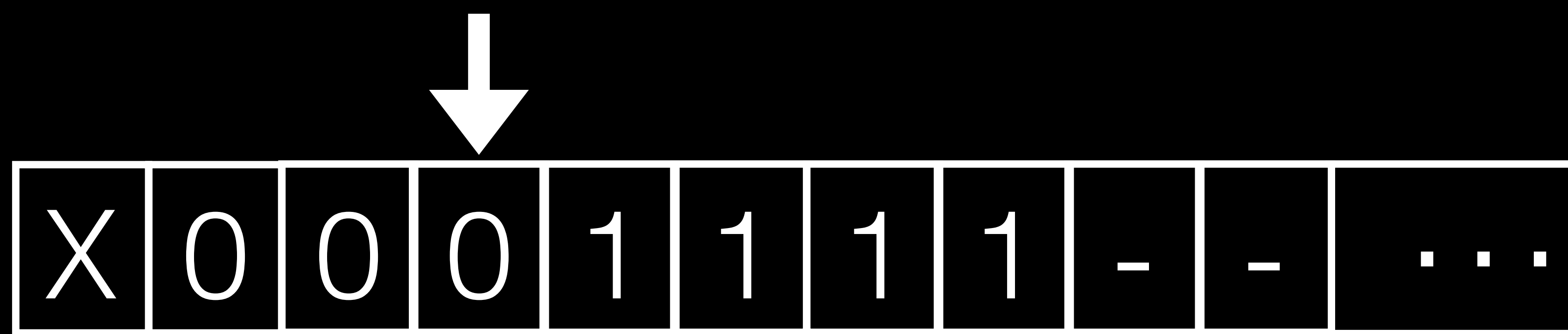
EXAMPLE

$$L = 0^n 1^n$$



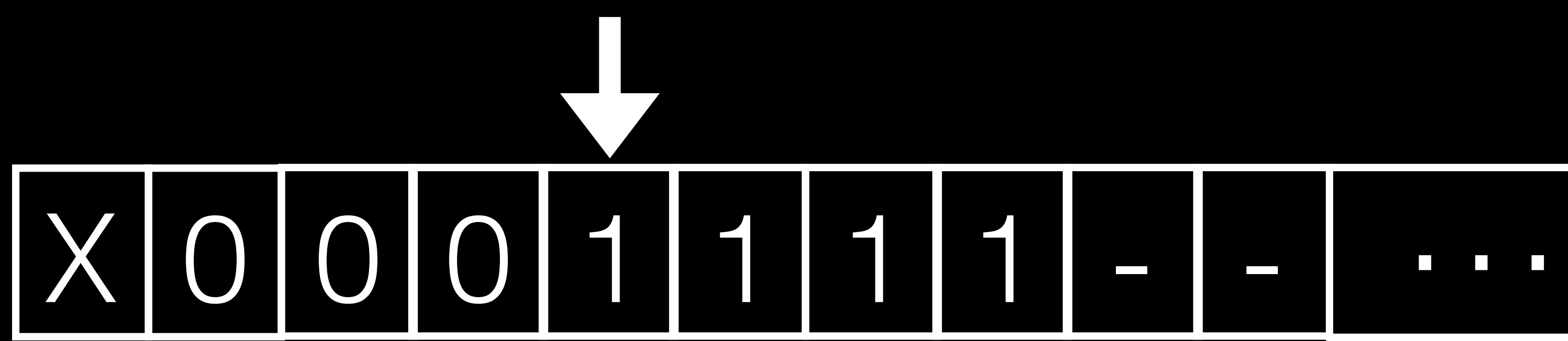
EXAMPLE

$$L = 0^n1^n$$



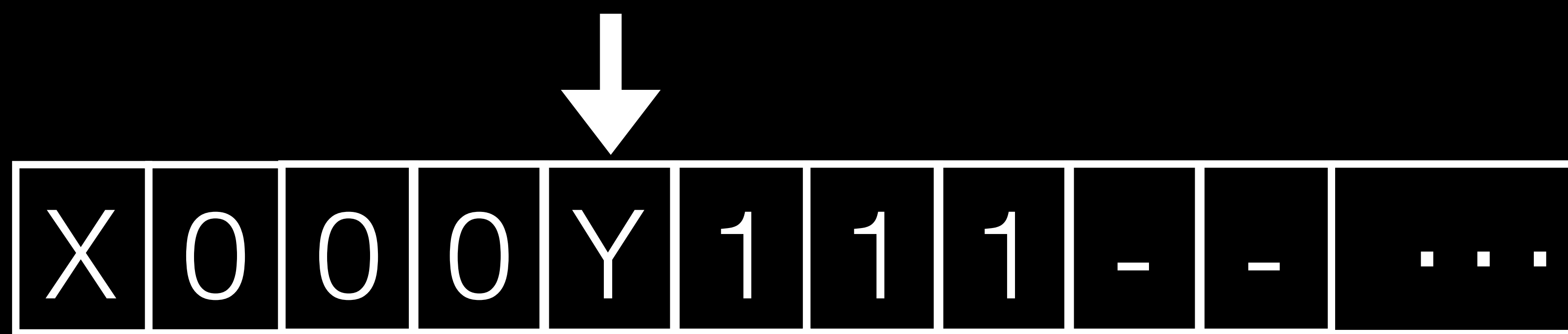
EXAMPLE

$L = 0^n 1^n$



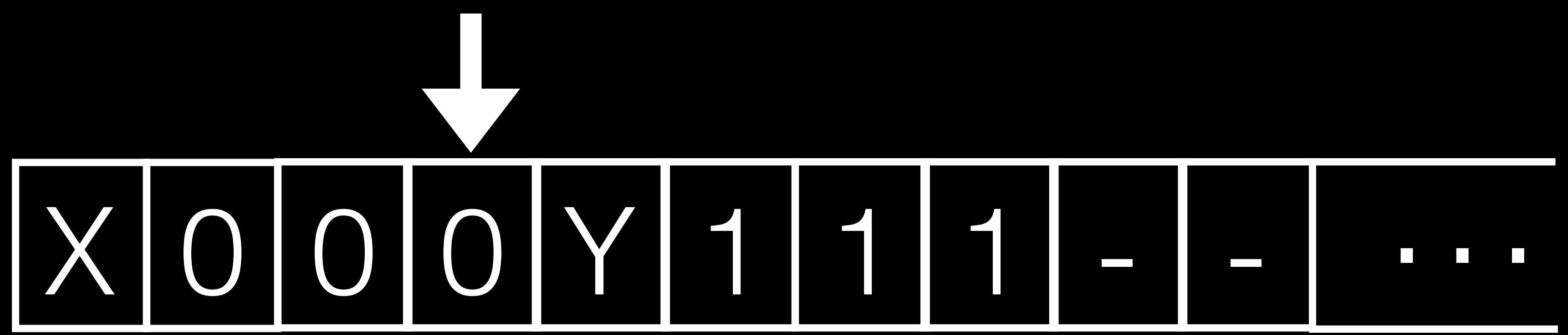
EXAMPLE

$L = 0^n 1^n$



EXAMPLE

$L = 0^n 1^n$



EXAMPLE

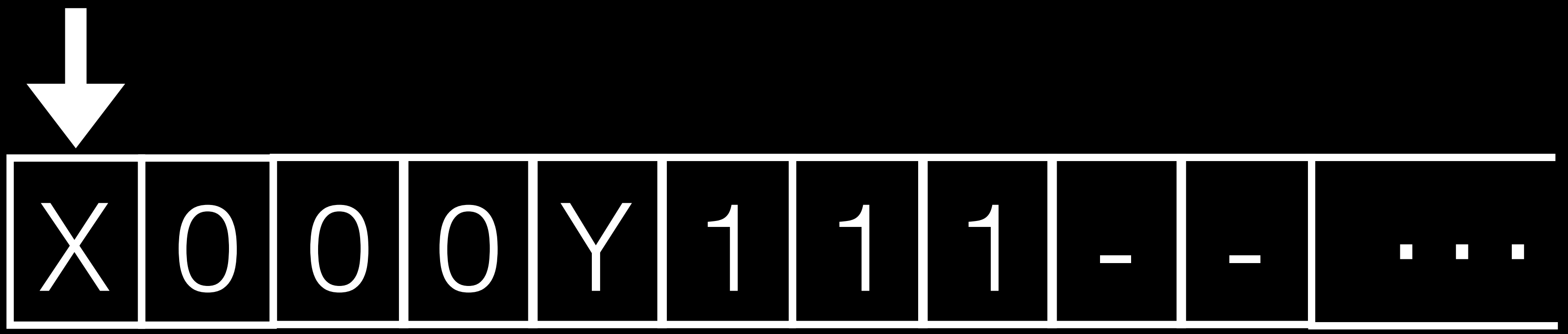
$L = 0^n 1^n$

EXAMPLE

$$L = 0^n 1^n$$

EXAMPLE

$L = 0^n 1^n$



EXAMPLE

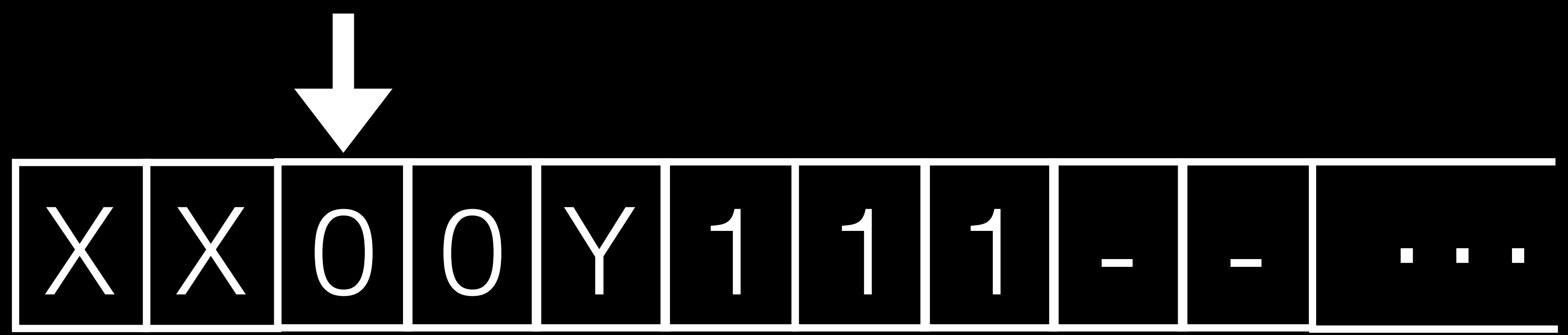
$L = 0^n 1^n$

EXAMPLE

$$L = 0^n 1^n$$

EXAMPLE

$$L = 0^n 1^n$$

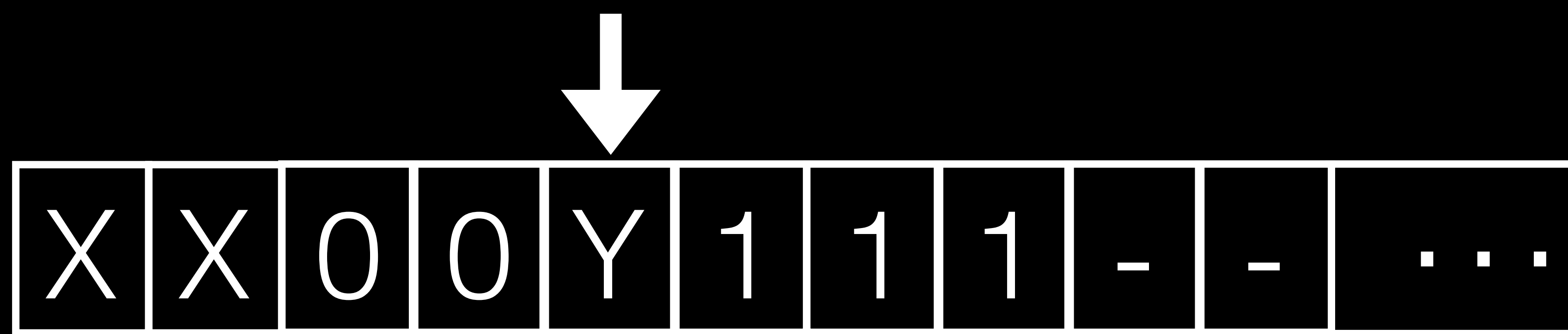


EXAMPLE

$$L = 0^n 1^n$$

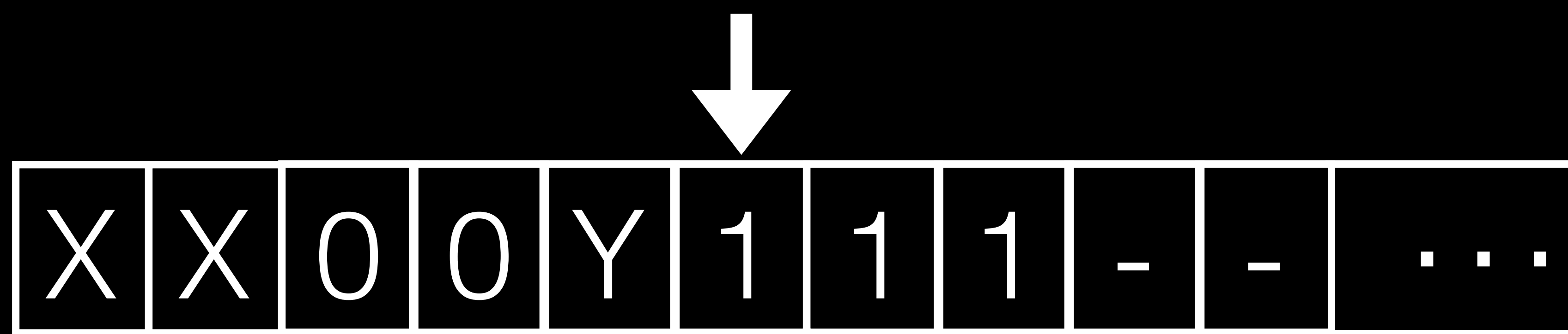
EXAMPLE

$L = 0^n1^n$



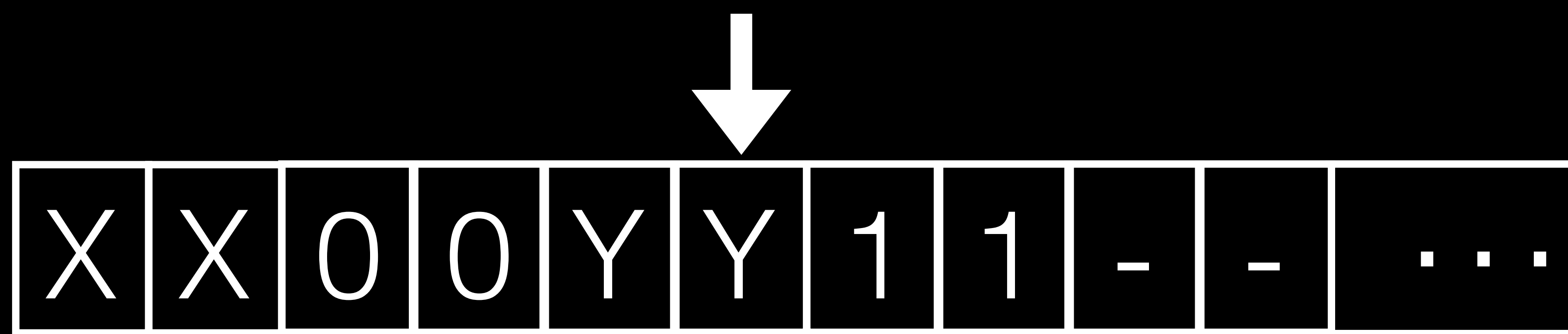
EXAMPLE

$L = 0^n 1^n$



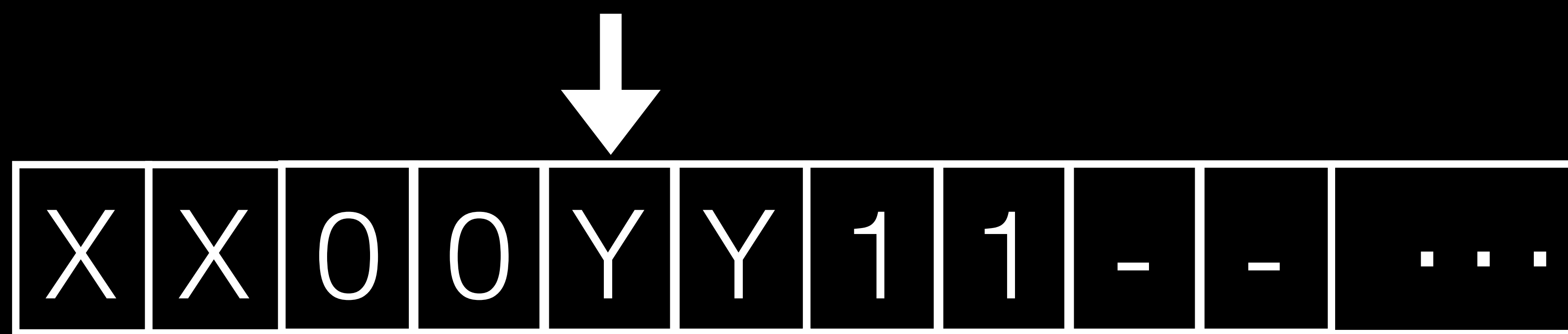
EXAMPLE

$L = 0^n 1^n$



EXAMPLE

$L = 0^n 1^n$



EXAMPLE

$$L = 0^n 1^n$$

EXAMPLE

$L = 0^n 1^n$

EXAMPLE

$$L = 0^n 1^n$$

EXAMPLE

$L = 0^n 1^n$

X	X	0	0	Y	Y	1	1	-	-	...

EXAMPLE

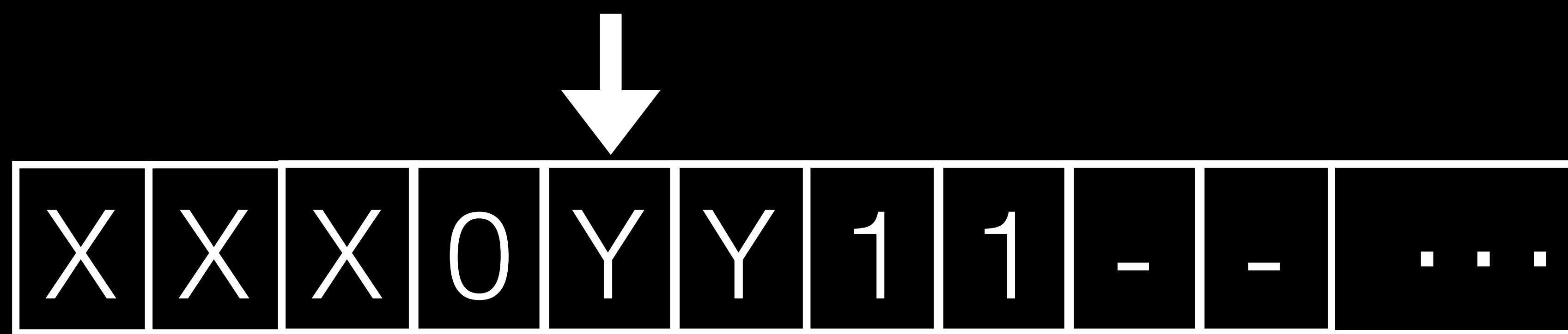
$L = 0^n 1^n$

EXAMPLE

$$L = 0^n1^n$$

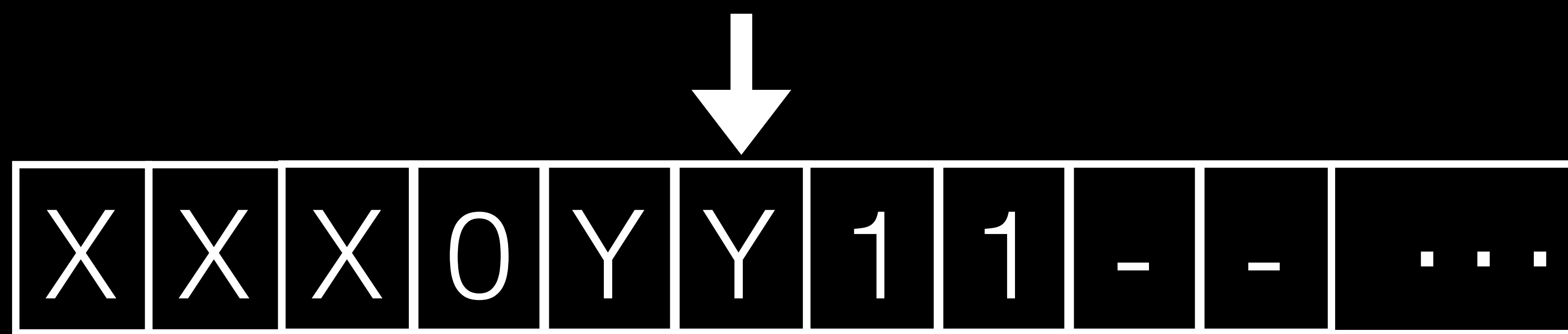
EXAMPLE

$L = 0^n 1^n$



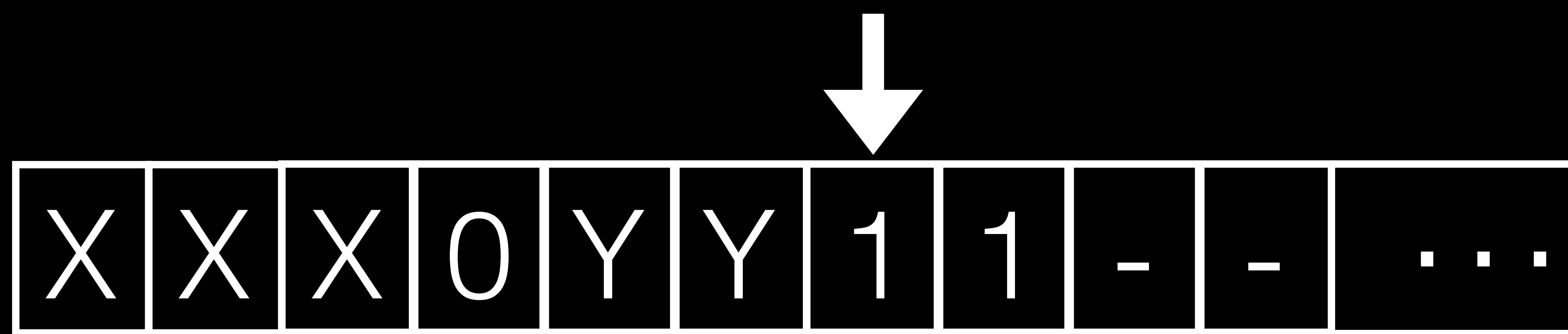
EXAMPLE

$$L = 0^n 1^n$$



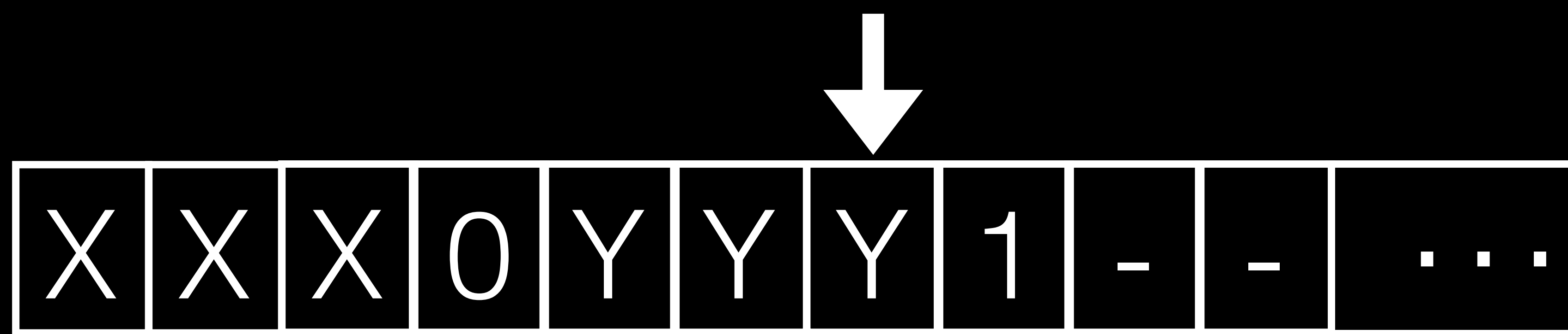
EXAMPLE

$L = 0^n 1^n$



EXAMPLE

$$L = 0^n 1^n$$



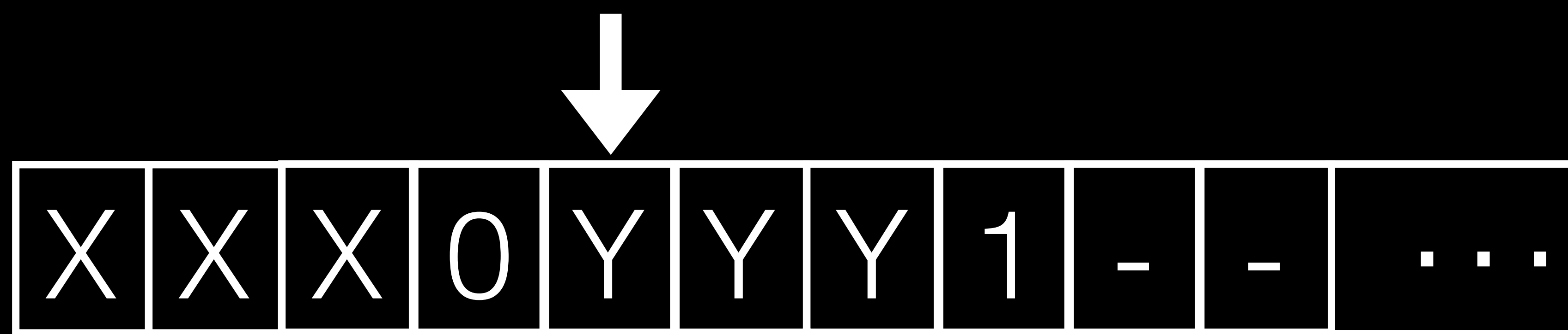
EXAMPLE

$$L = 0^n1^n$$

X	X	X	0	Y	Y	Y	1	-	-	...

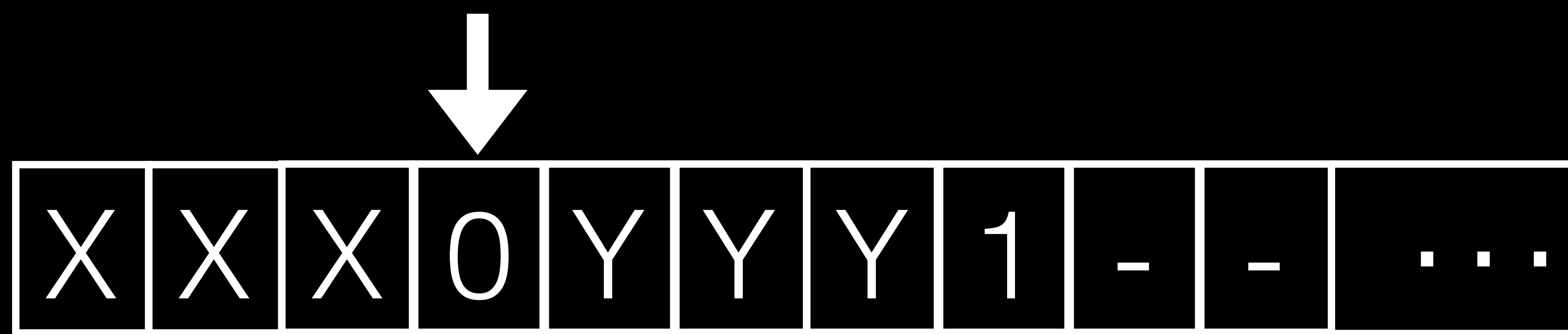
EXAMPLE

$L = 0^n 1^n$



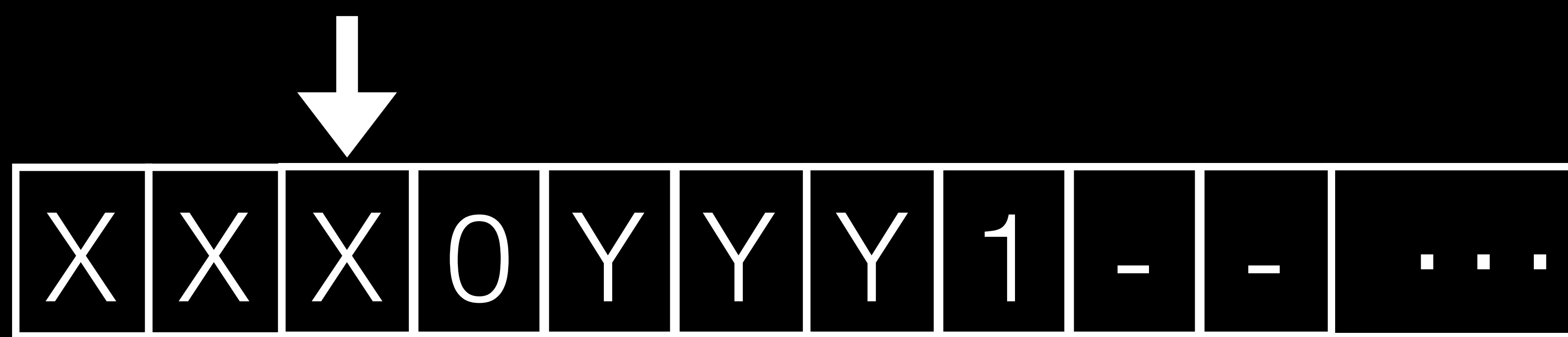
EXAMPLE

$L = 0^n 1^n$



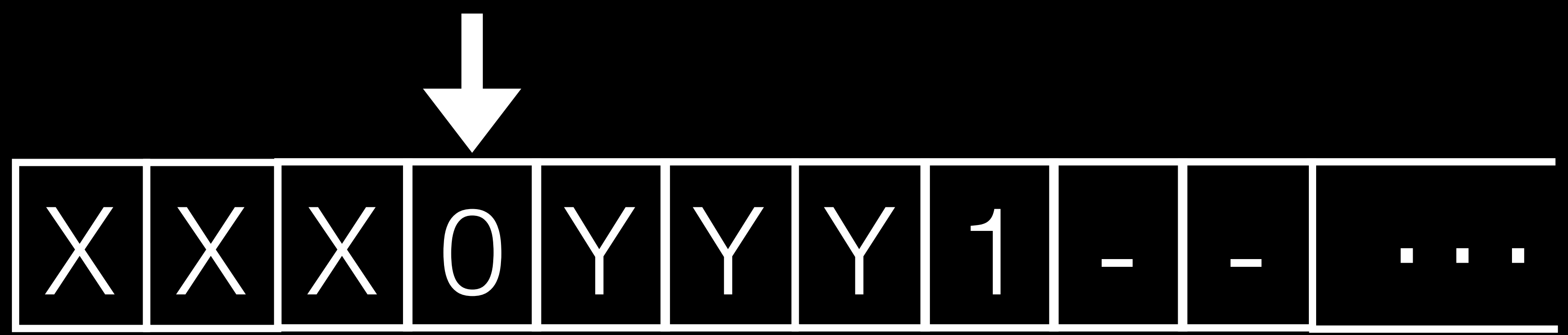
EXAMPLE

$L = 0^n 1^n$



EXAMPLE

$L = 0^n 1^n$



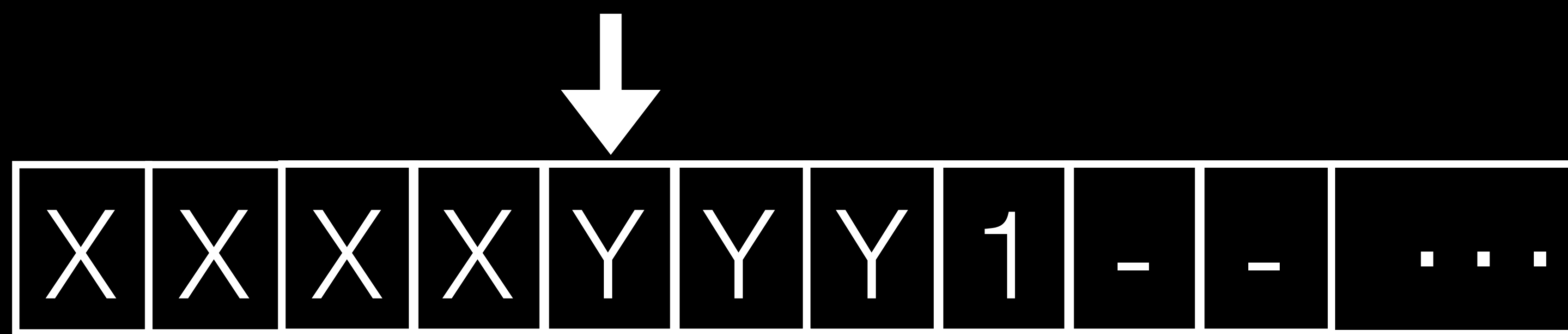
EXAMPLE

$L = 0^n 1^n$

X	X	X	X	Y	Y	Y	1	-	-	...

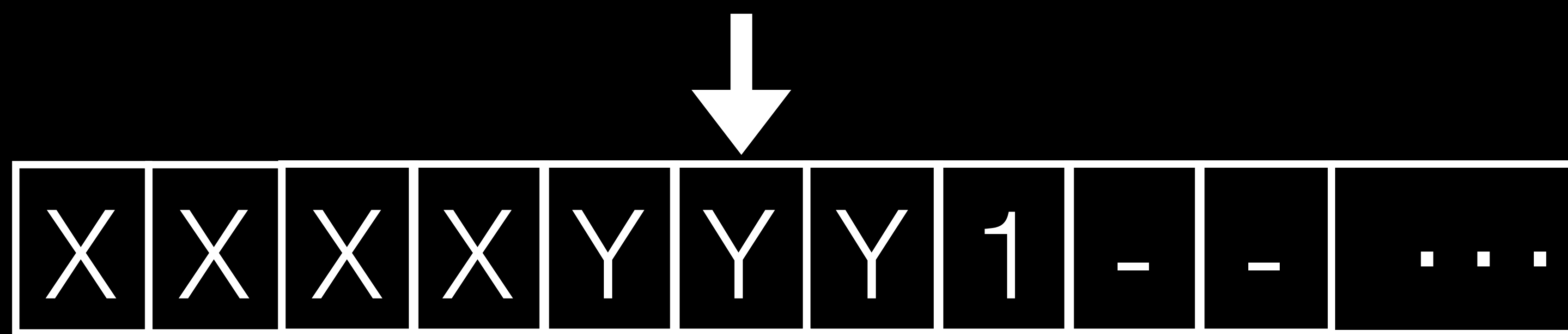
EXAMPLE

$$L = 0^n 1^n$$



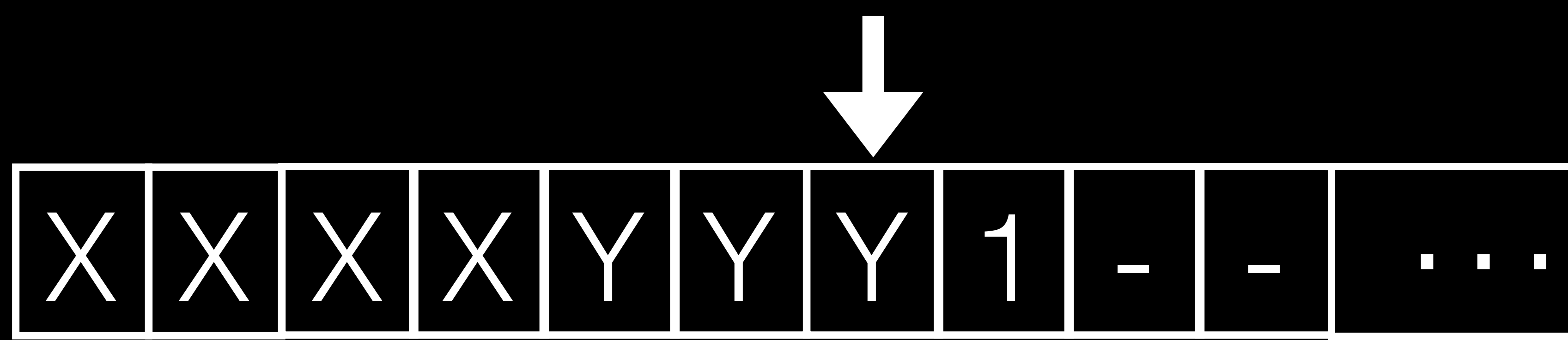
EXAMPLE

$L = 0^n1^n$



EXAMPLE

$$L = 0^n 1^n$$



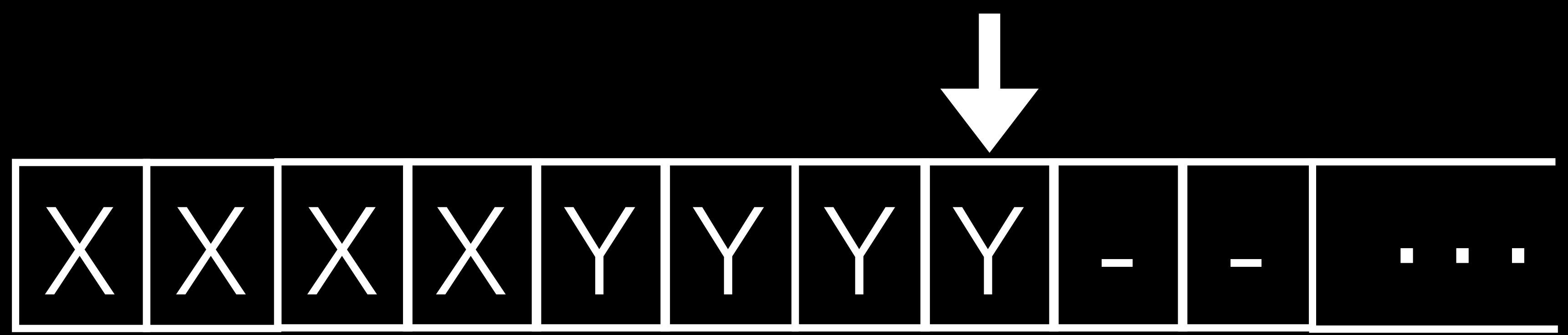
EXAMPLE

$L = 0^n 1^n$

X	X	X	X	Y	Y	Y	1	-	-	...

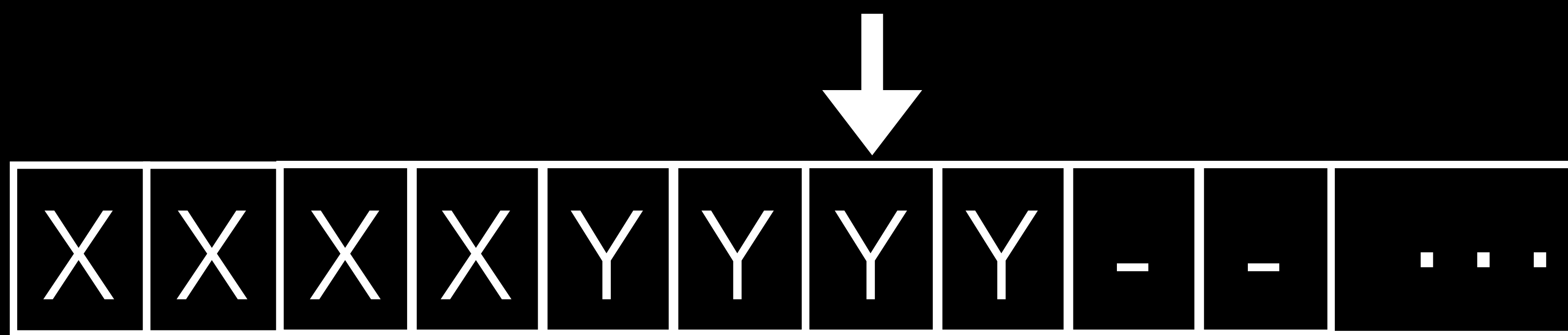
EXAMPLE

$L = 0^n 1^n$



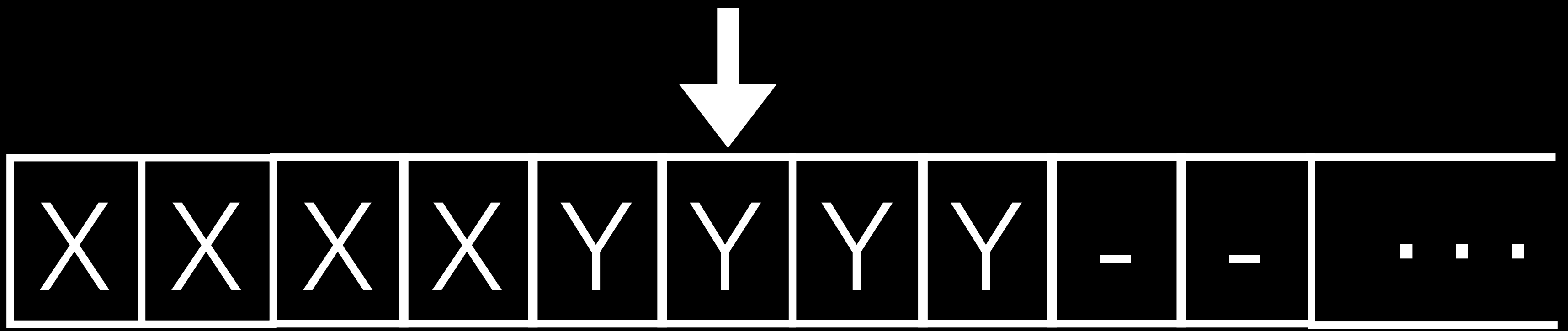
EXAMPLE

$$L = 0^n 1^n$$



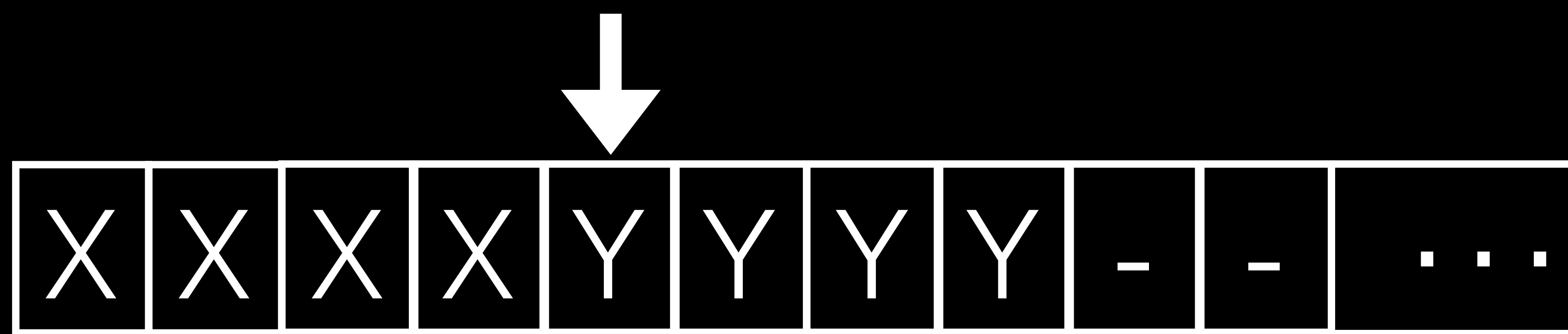
EXAMPLE

$L = 0^n 1^n$



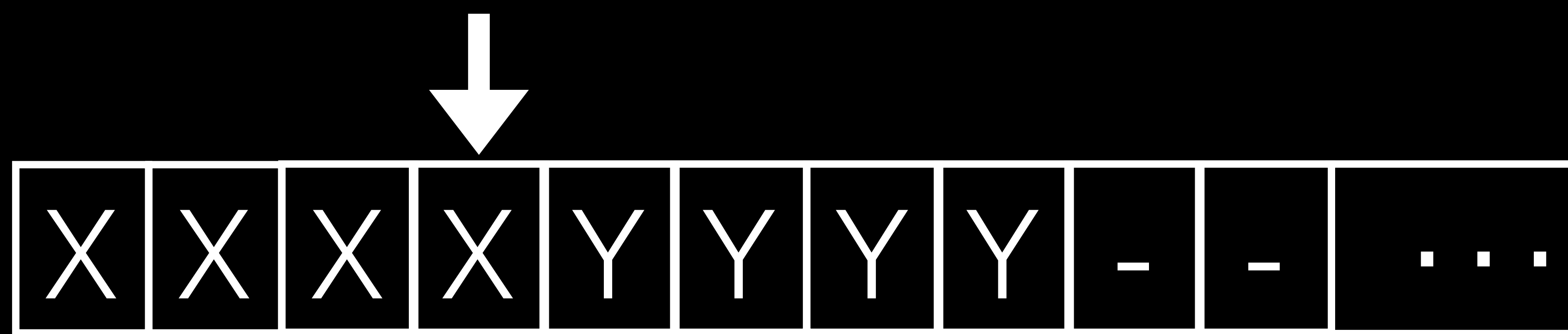
EXAMPLE

$$L = 0^n 1^n$$



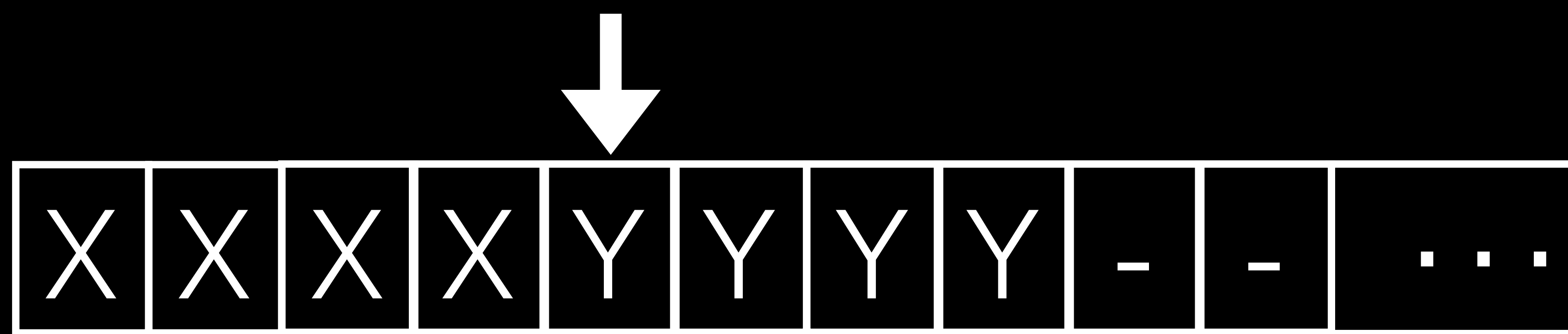
EXAMPLE

$$L = 0^n1^n$$



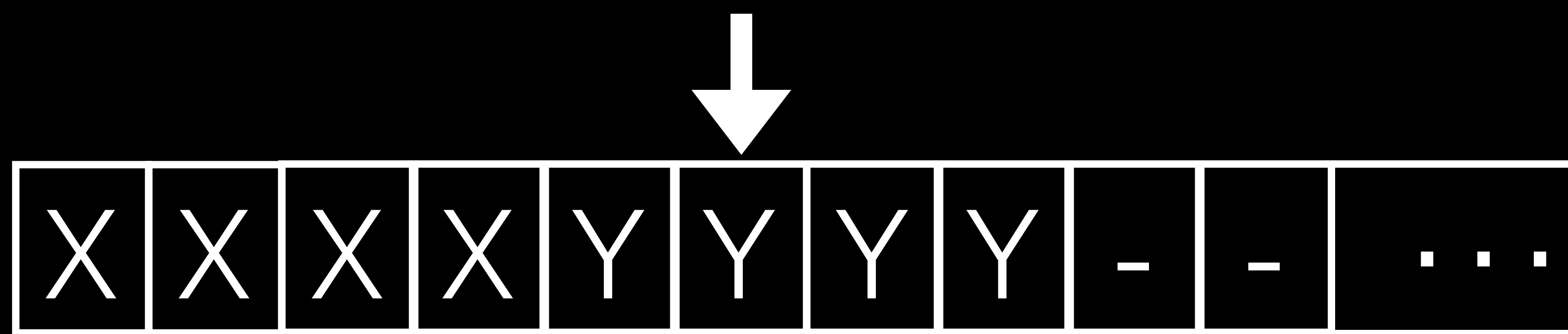
EXAMPLE

$$L = 0^n 1^n$$



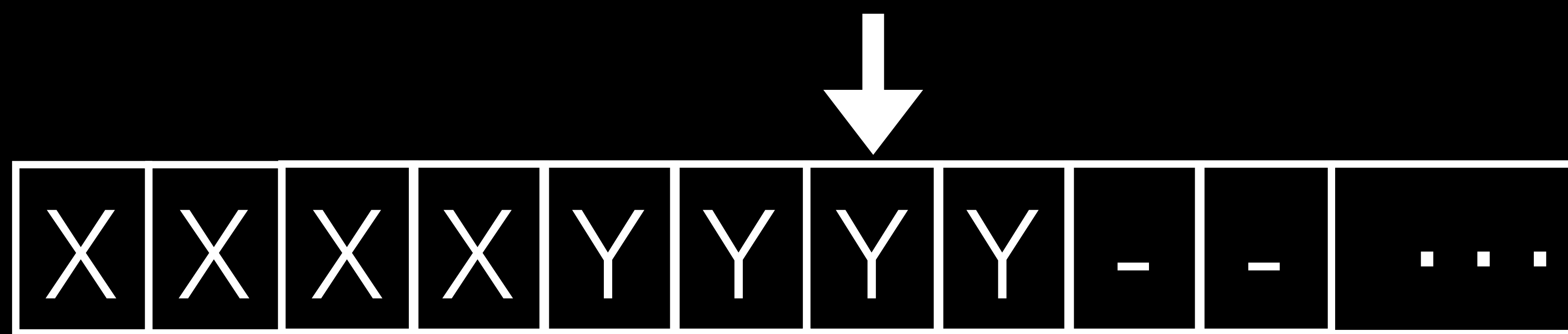
EXAMPLE

$L = 0^n 1^n$



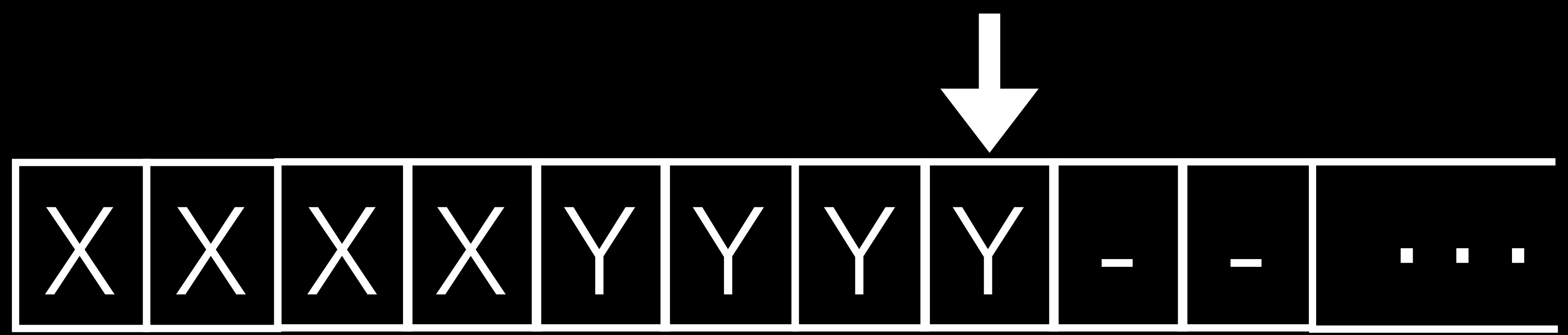
EXAMPLE

$$L = 0^n 1^n$$



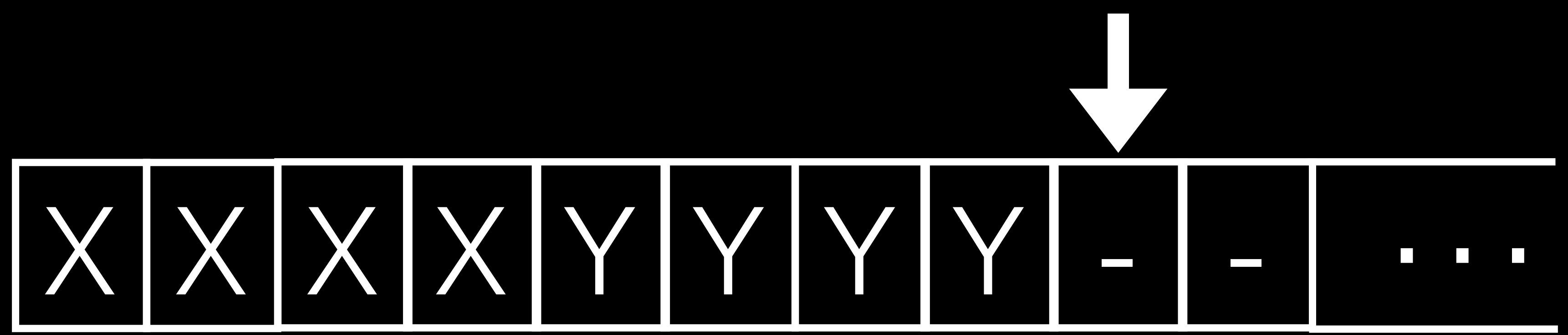
EXAMPLE

$$L = 0^n 1^n$$



EXAMPLE

$L = 0^n 1^n$



Accept!

EXAMPLE

$L = 0^n 1^n$

Algorithm:

1. Change 0 to X
2. Move right to first 1. If none, reject.
3. Change 1 into Y.
4. Move left to leftmost 0
5. Repeat (1) until no more 0s.
6. Make sure no more 1s remain.

EXAMPLE

$L = 0^n 1^n$

Computation History:

0 0 0 0 1 1 1 1

X 0 0 0 1 1 1 1

X 0 0 0 Y 1 1 1

X X 0 0 Y 1 1 1

. . .

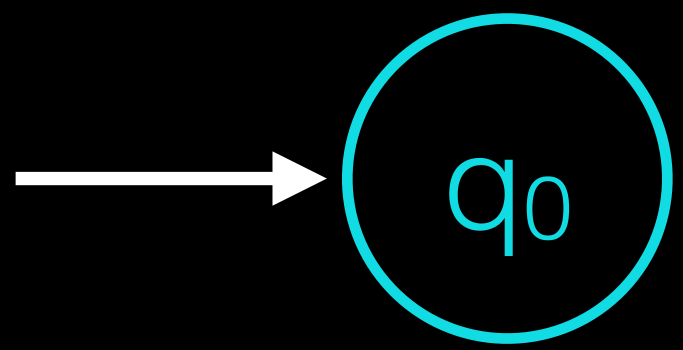
X X X X Y Y Y Y

Tape Alphabet:

$\Gamma = \{0, 1, X, Y, —\}$

EXAMPLE

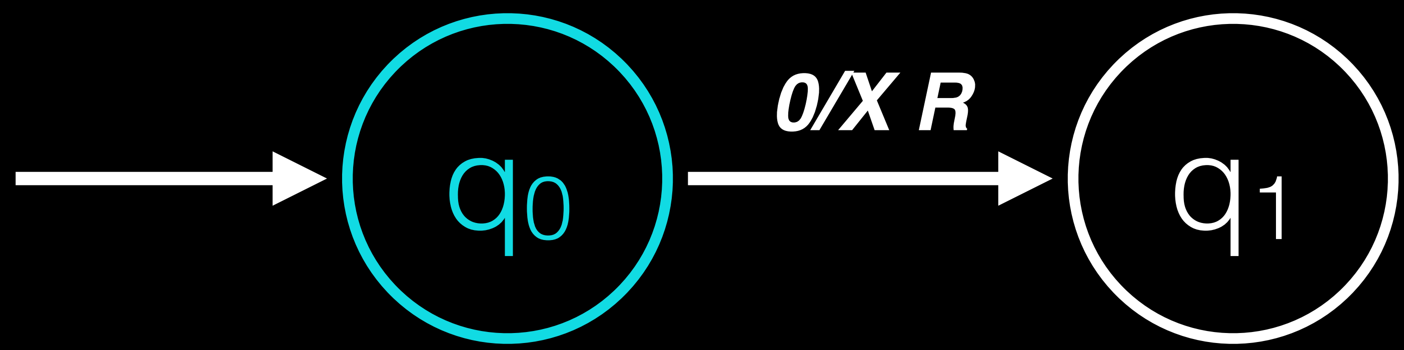
$L = 0^n 1^n$



0 0 0 1 1 1

EXAMPLE

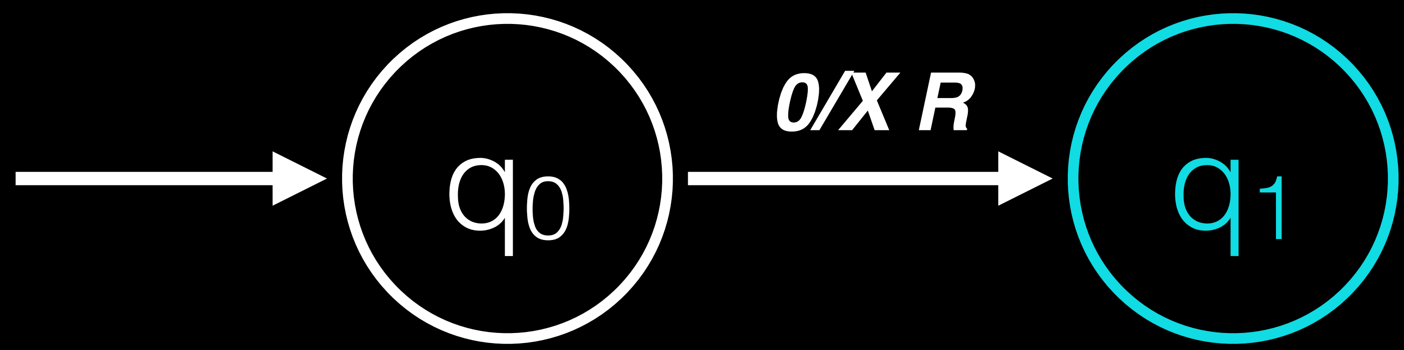
$L = 0^n 1^n$



0 0 0 1 1 1

EXAMPLE

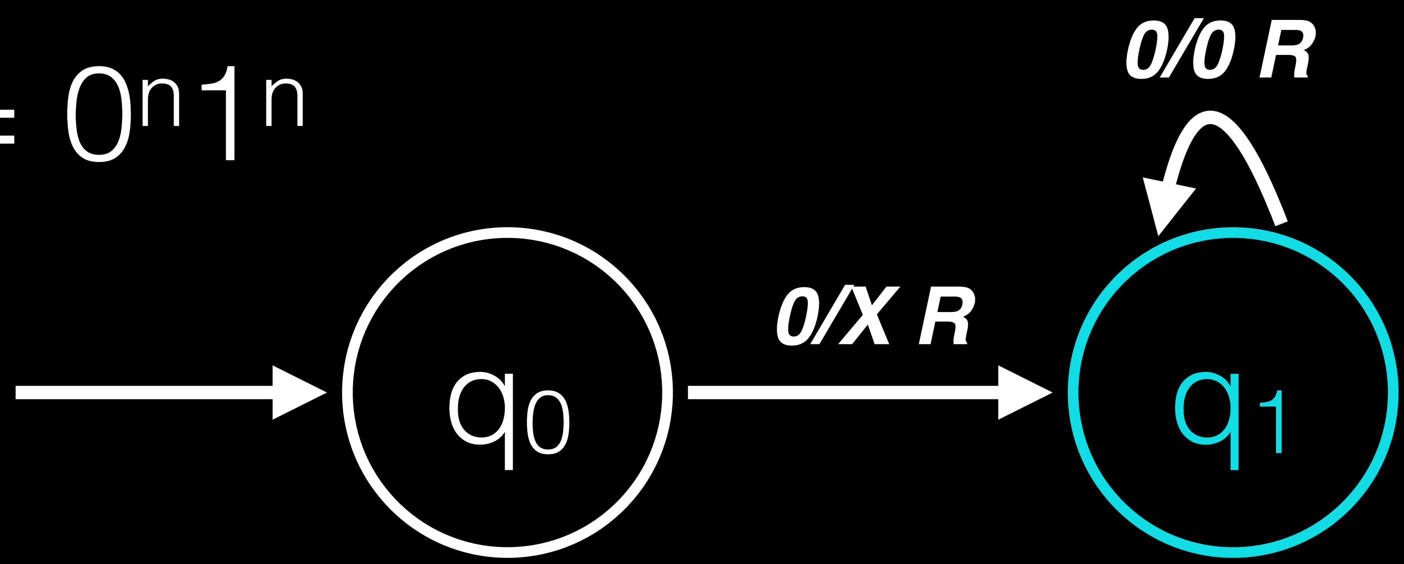
$L = 0^n 1^n$



X 0 0 1 1 1

EXAMPLE

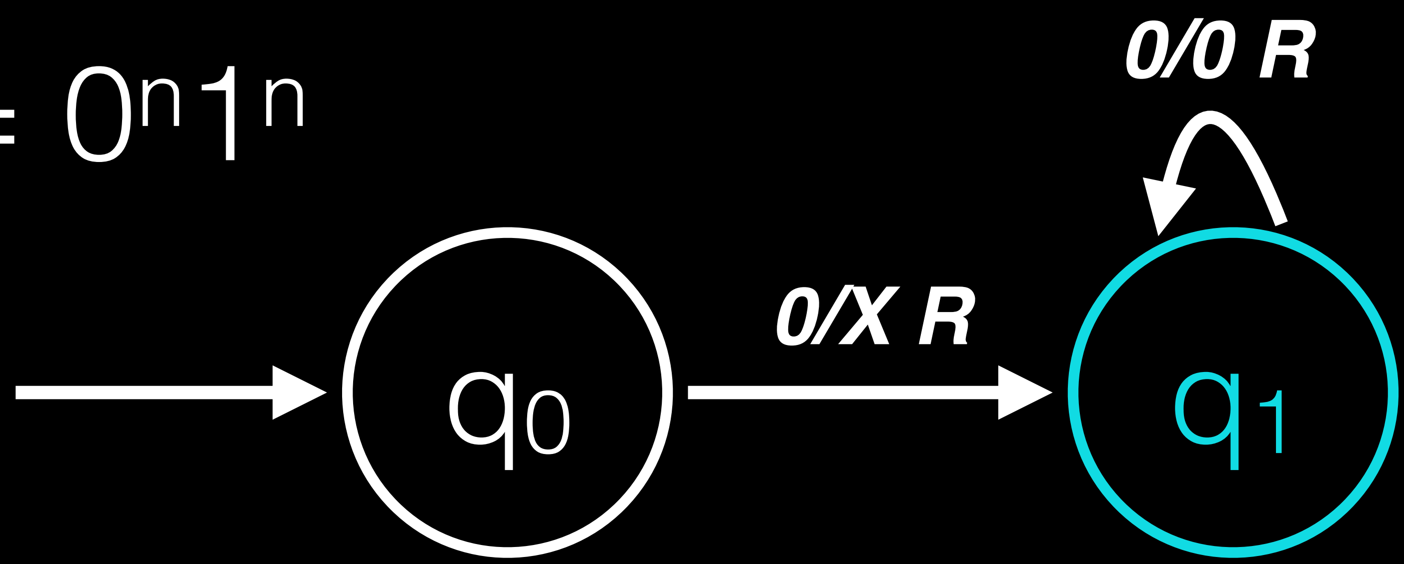
$L = 0^n 1^n$



X 0 0 1 1 1

EXAMPLE

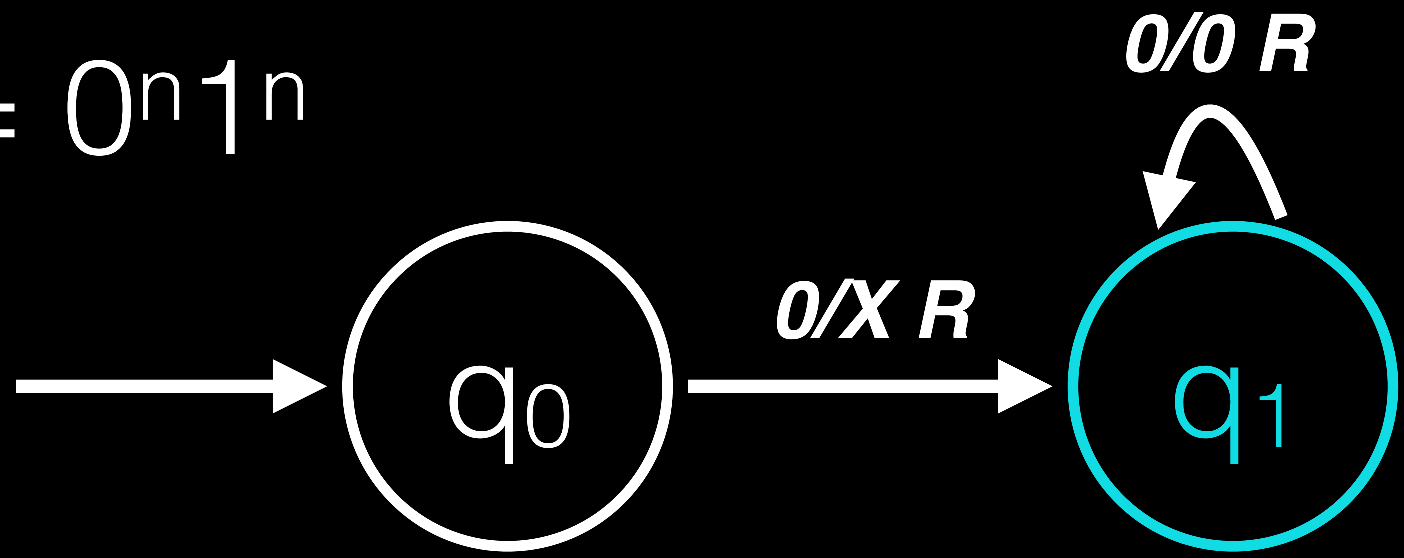
$L = 0^n 1^n$



X 0 0 1 1 1

EXAMPLE

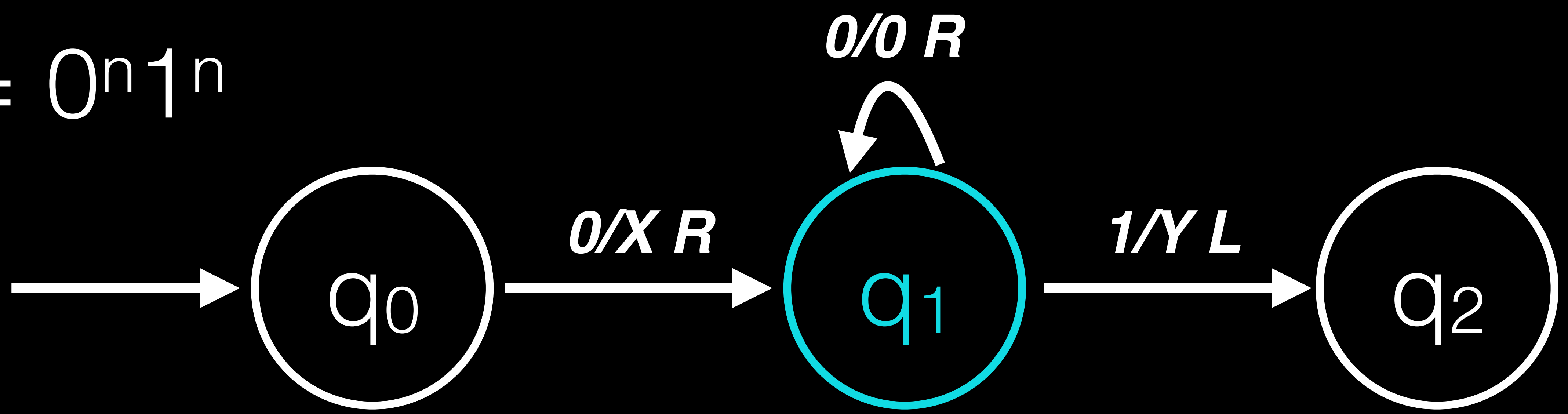
$L = 0^n1^n$



X 0 0 1 1 1

EXAMPLE

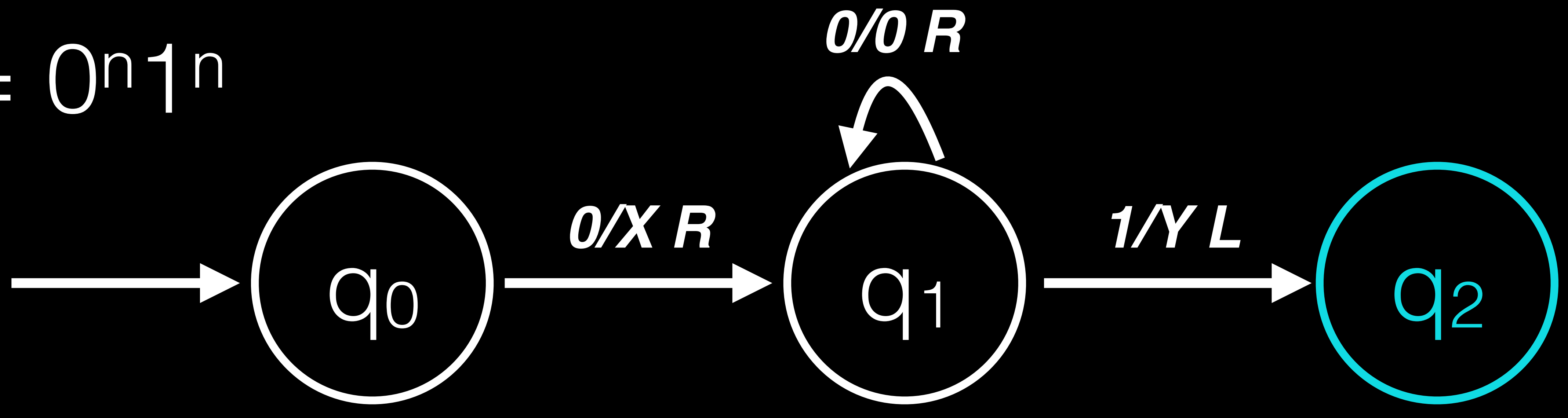
$L = 0^n 1^n$



X 0 0 1 1 1

EXAMPLE

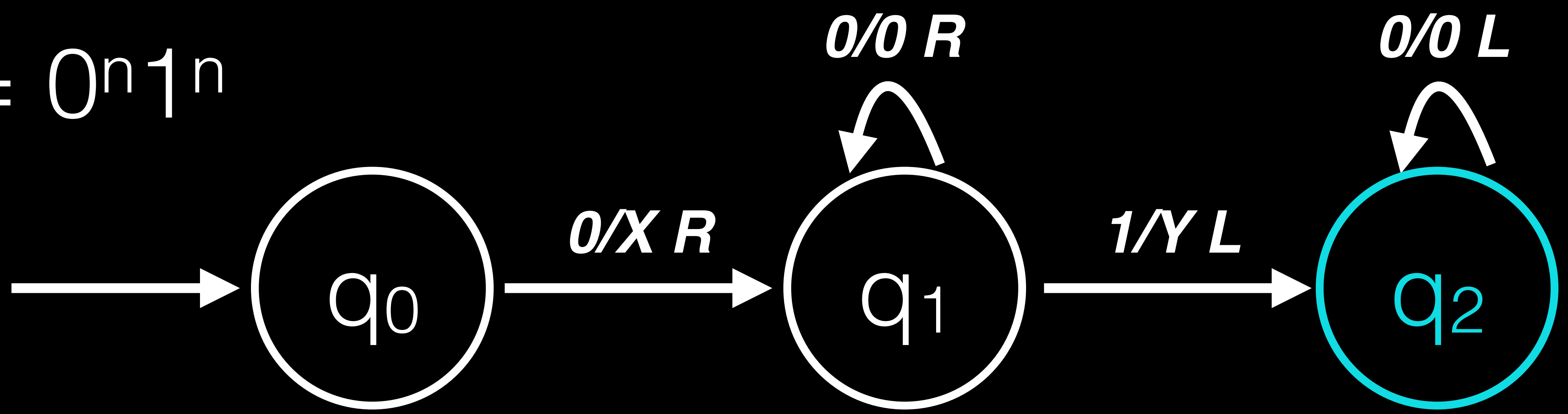
$L = 0^n 1^n$



X 0 0 Y 1 1

EXAMPLE

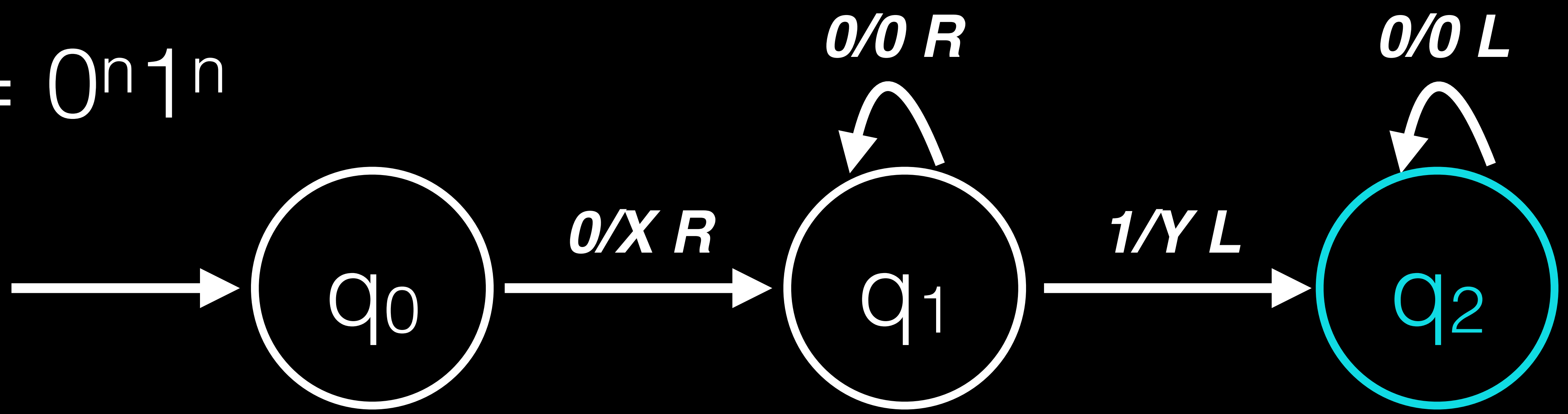
$L = 0^n 1^n$



X 0 0 Y 1 1

EXAMPLE

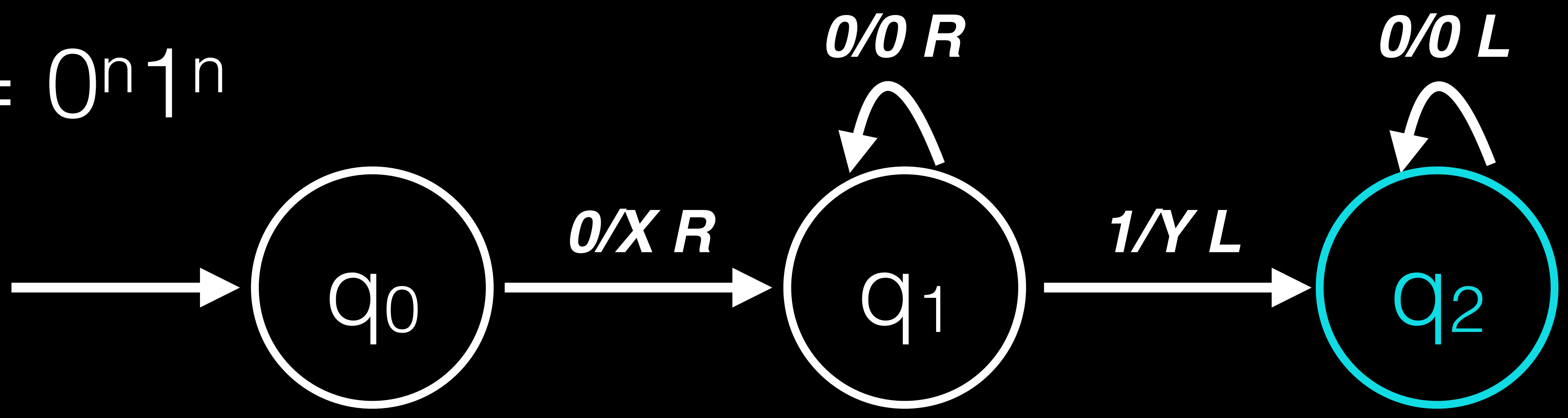
$L = 0^n1^n$



X 0 0 Y 1 1

EXAMPLE

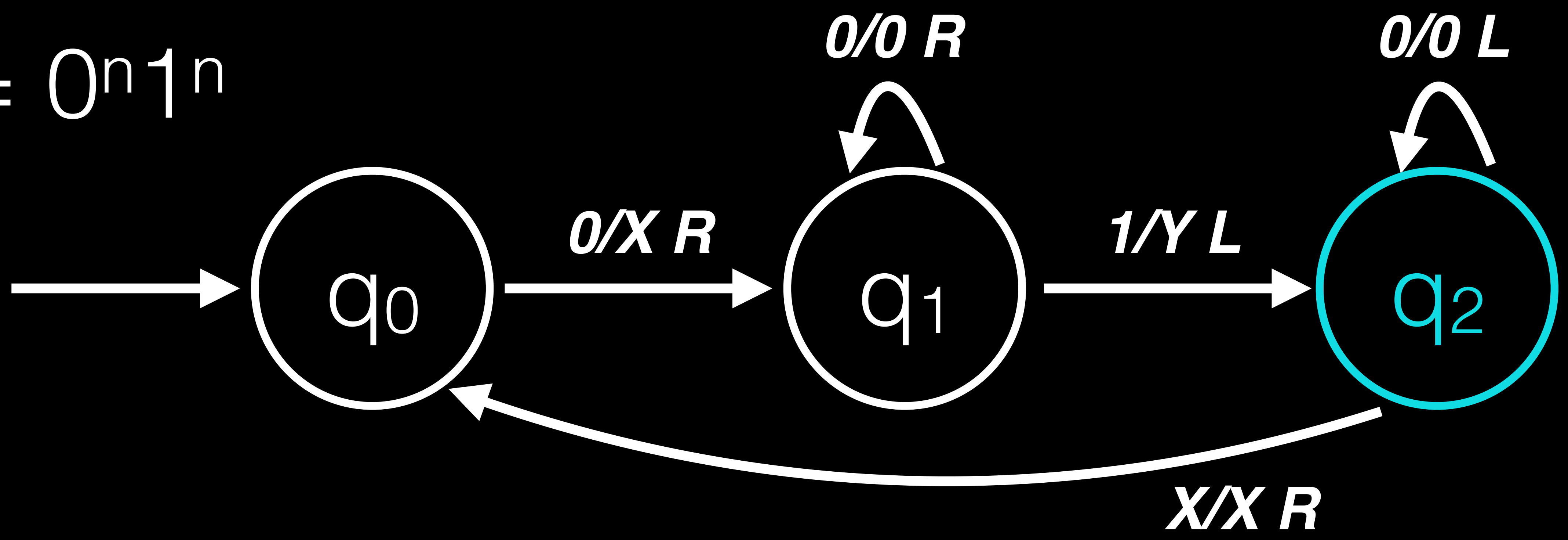
$L = 0^n 1^n$



X 0 0 Y 1 1

EXAMPLE

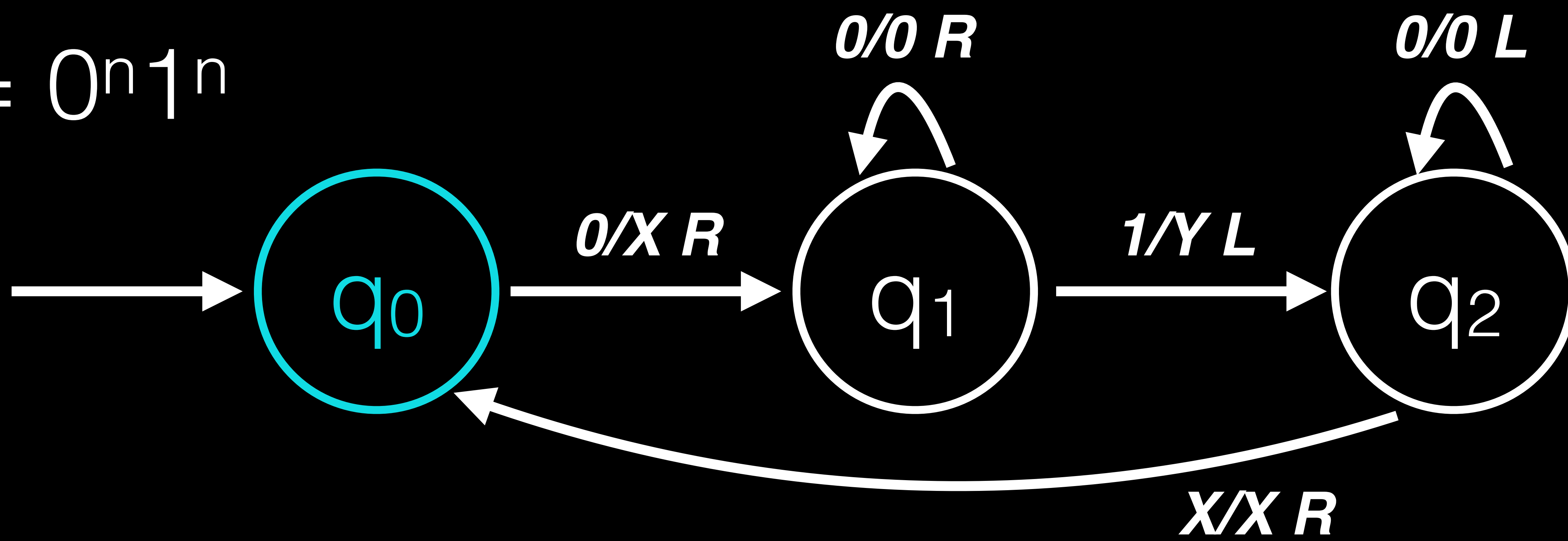
$L = 0^n 1^n$



X 0 0 Y 1 1

EXAMPLE

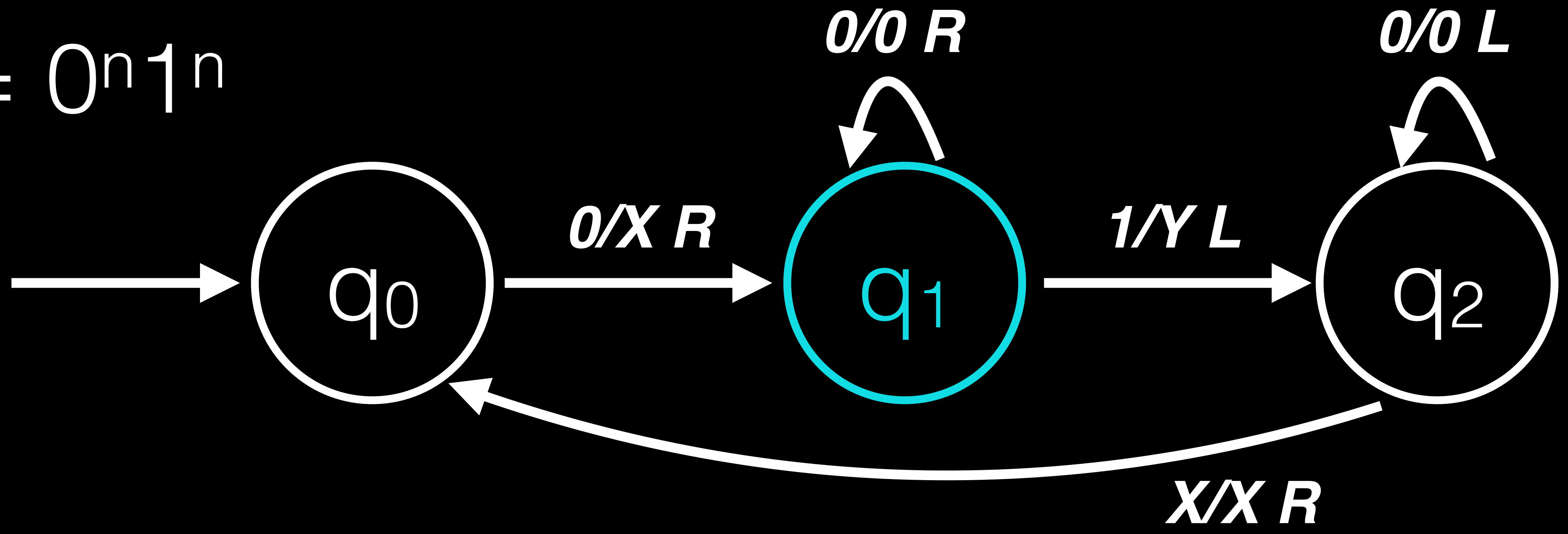
$L = 0^n 1^n$



X 0 0 Y 1 1

EXAMPLE

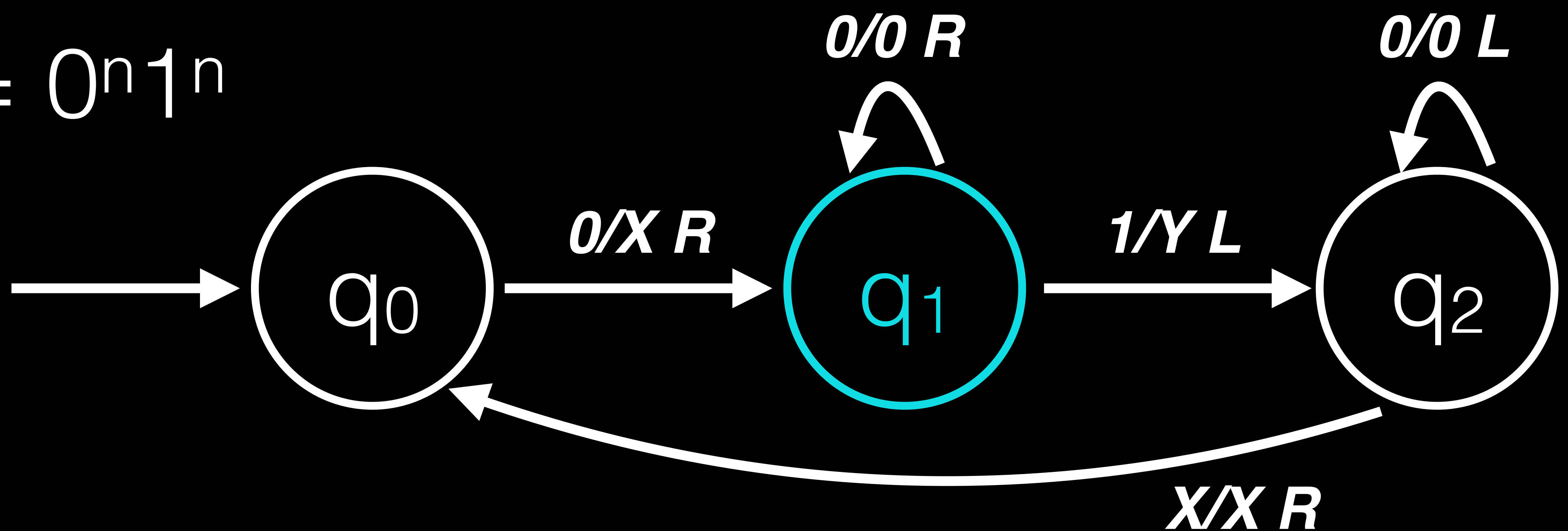
$L = 0^n 1^n$



X X 0 Y 1 1

EXAMPLE

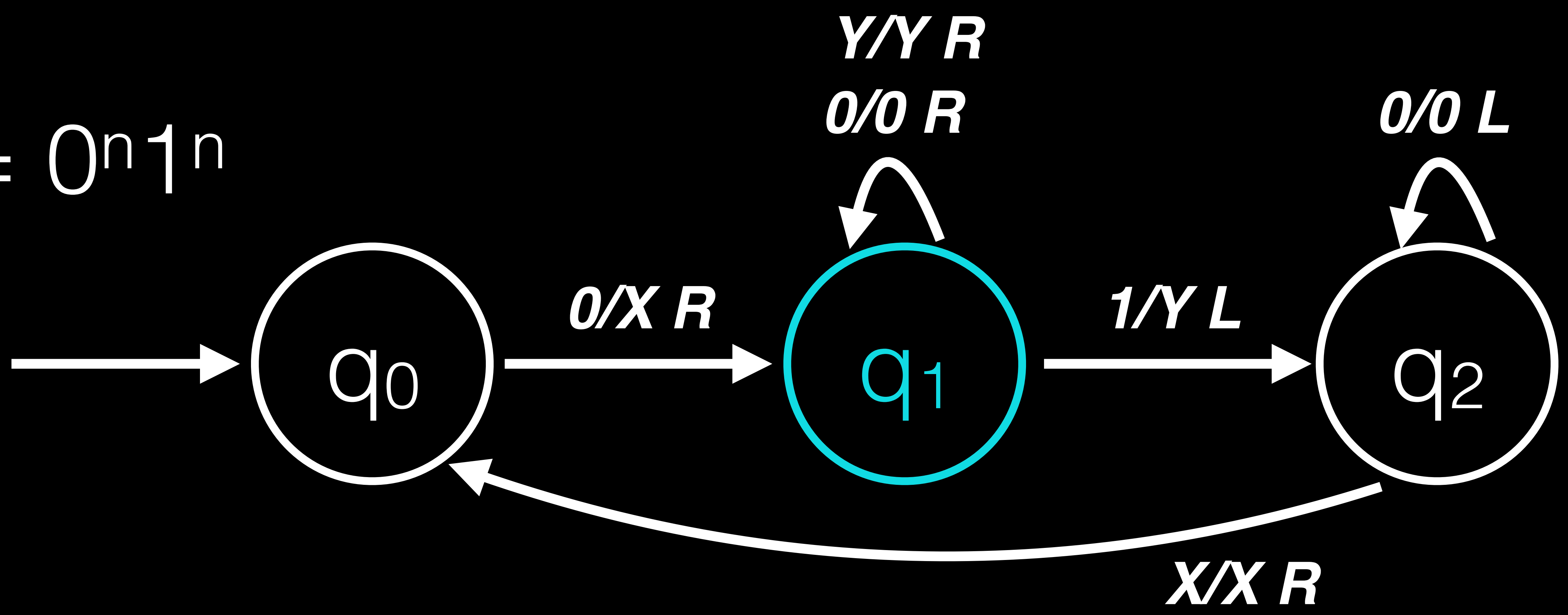
$L = 0^n 1^n$



X X 0 Y 1 1

EXAMPLE

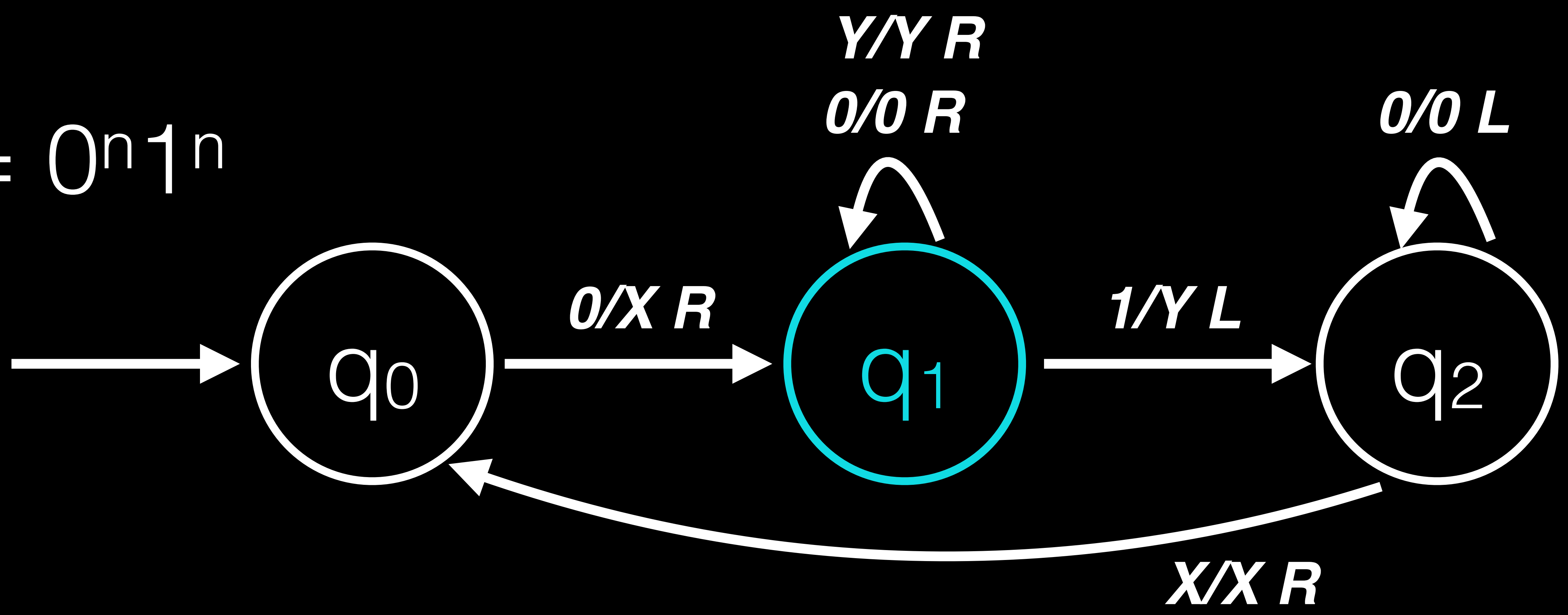
$$L = 0^n 1^n$$



X X 0 Y 1 1

EXAMPLE

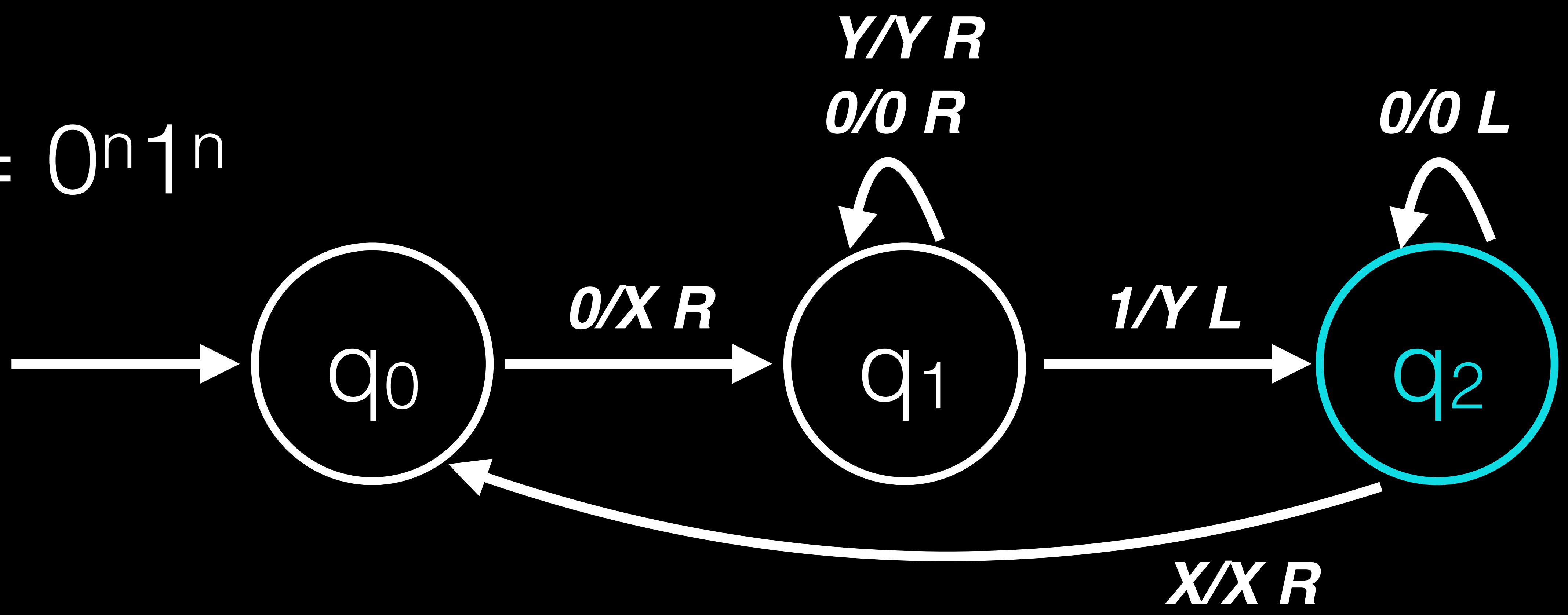
$L = 0^n 1^n$



X X 0 Y 1 1

EXAMPLE

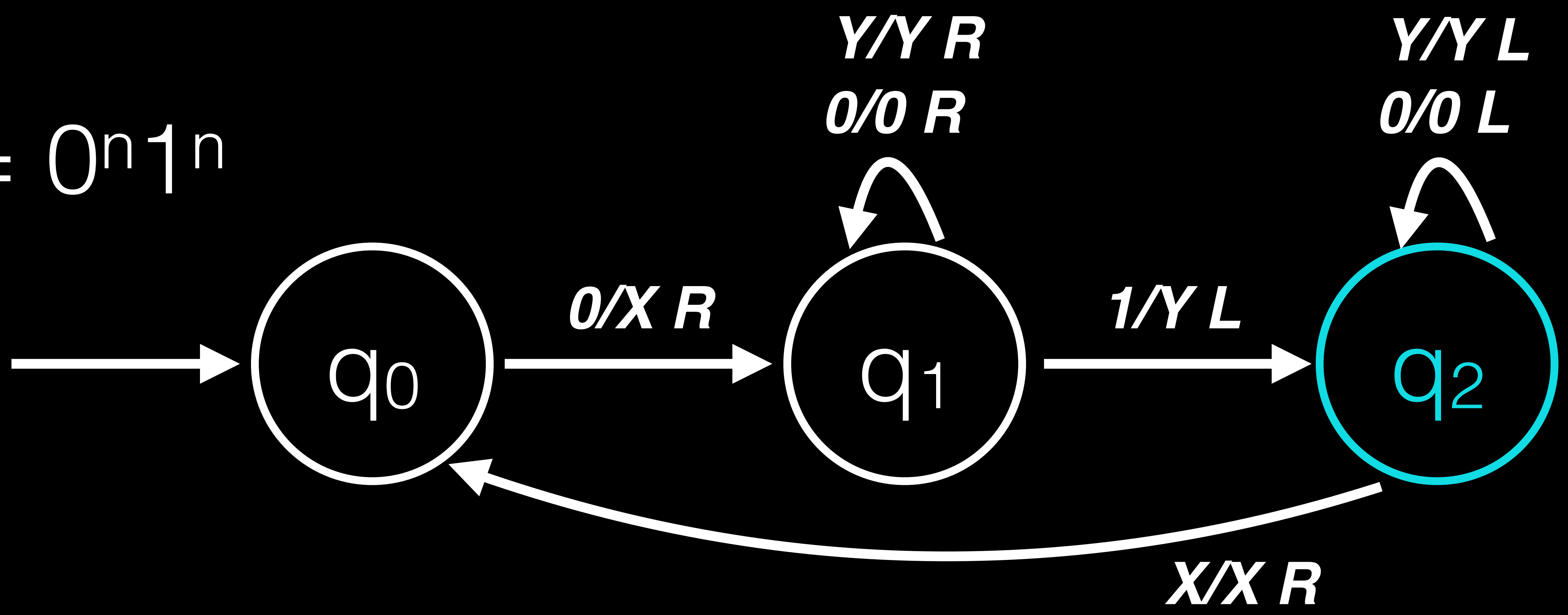
$L = 0^n 1^n$



X X 0 Y Y 1

EXAMPLE

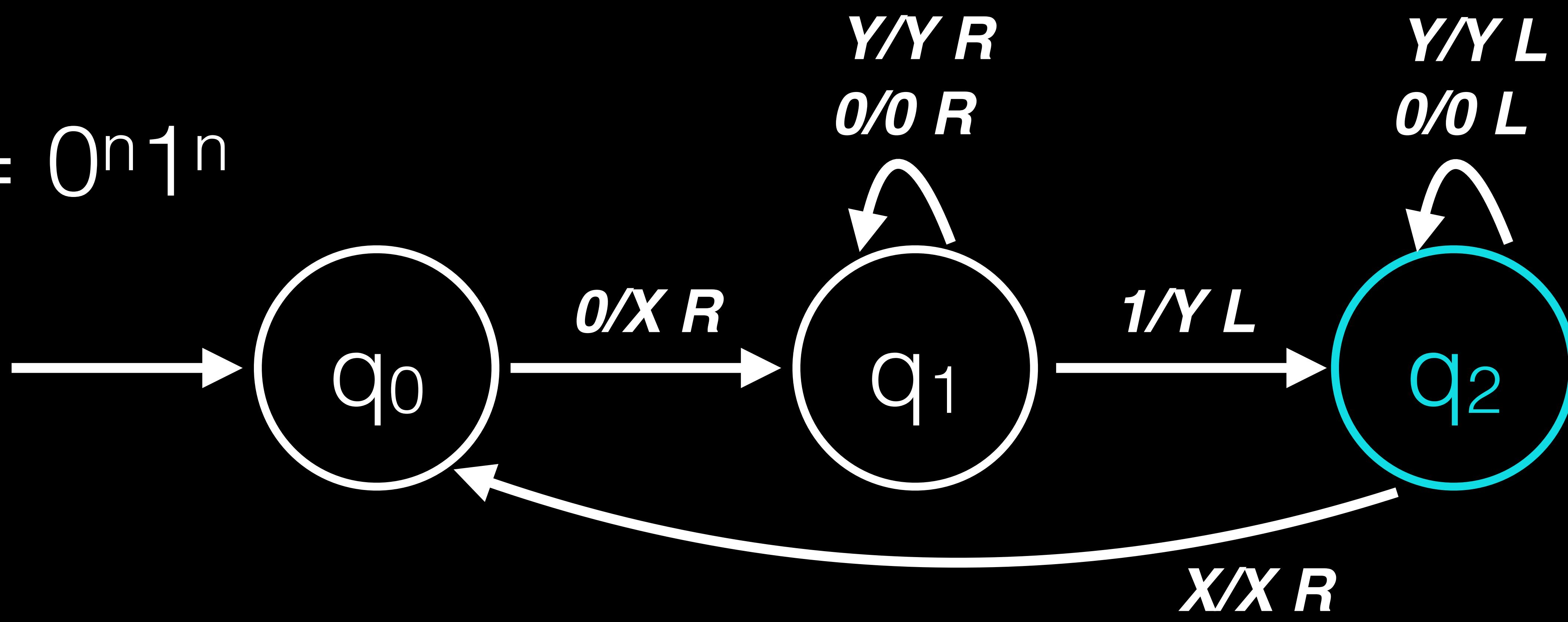
$L = 0^n 1^n$



X X 0 Y Y 1

EXAMPLE

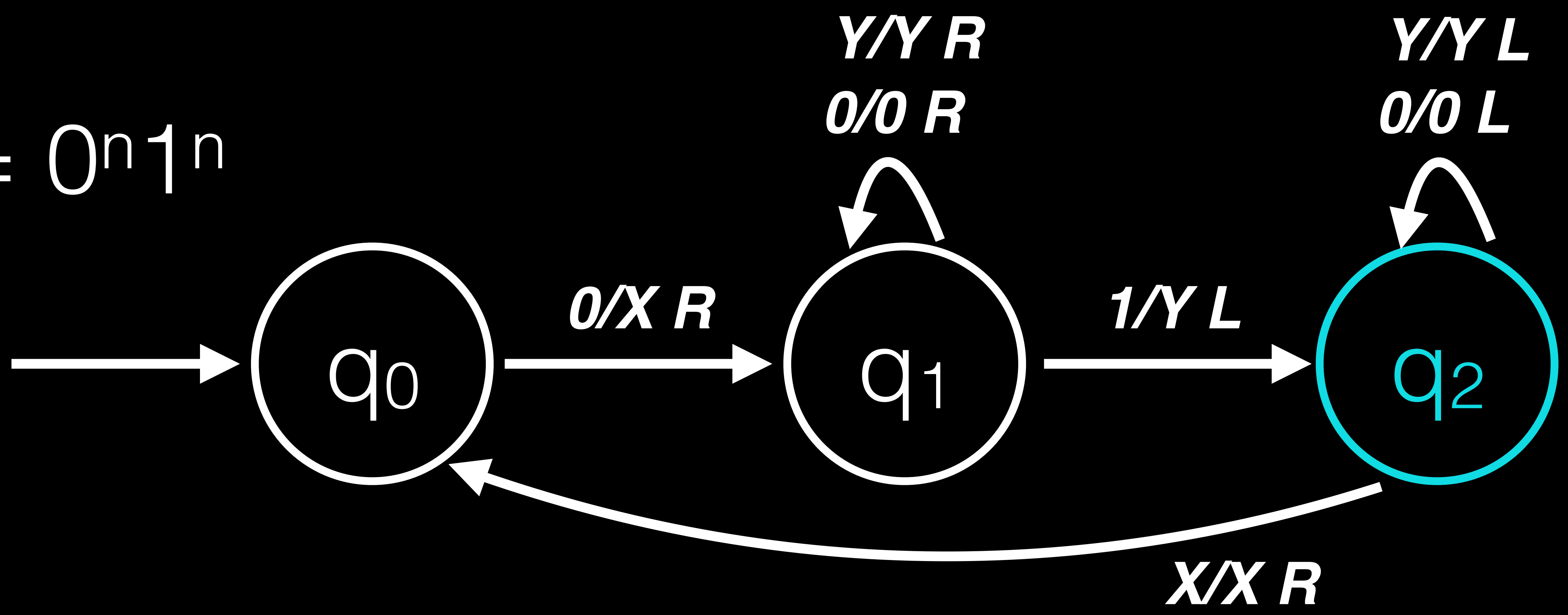
$L = 0^n 1^n$



X X 0 Y Y 1

EXAMPLE

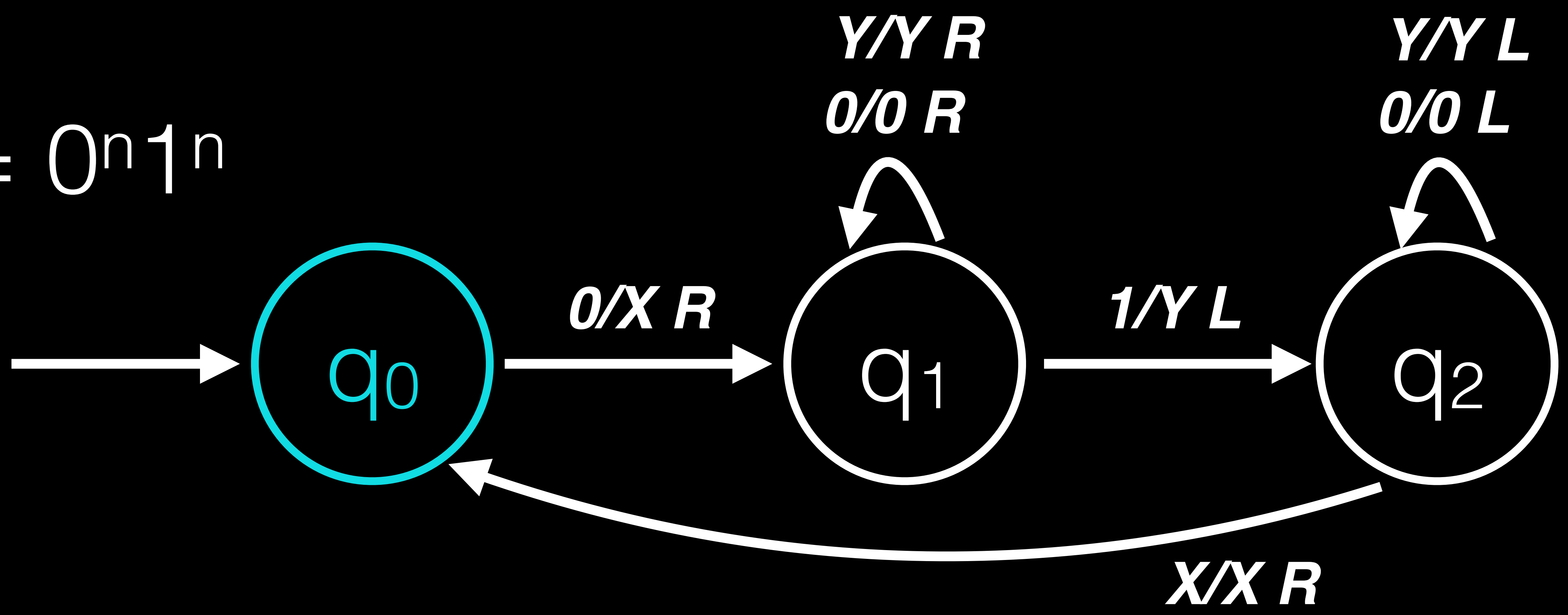
$L = 0^n 1^n$



X X 0 Y Y 1

EXAMPLE

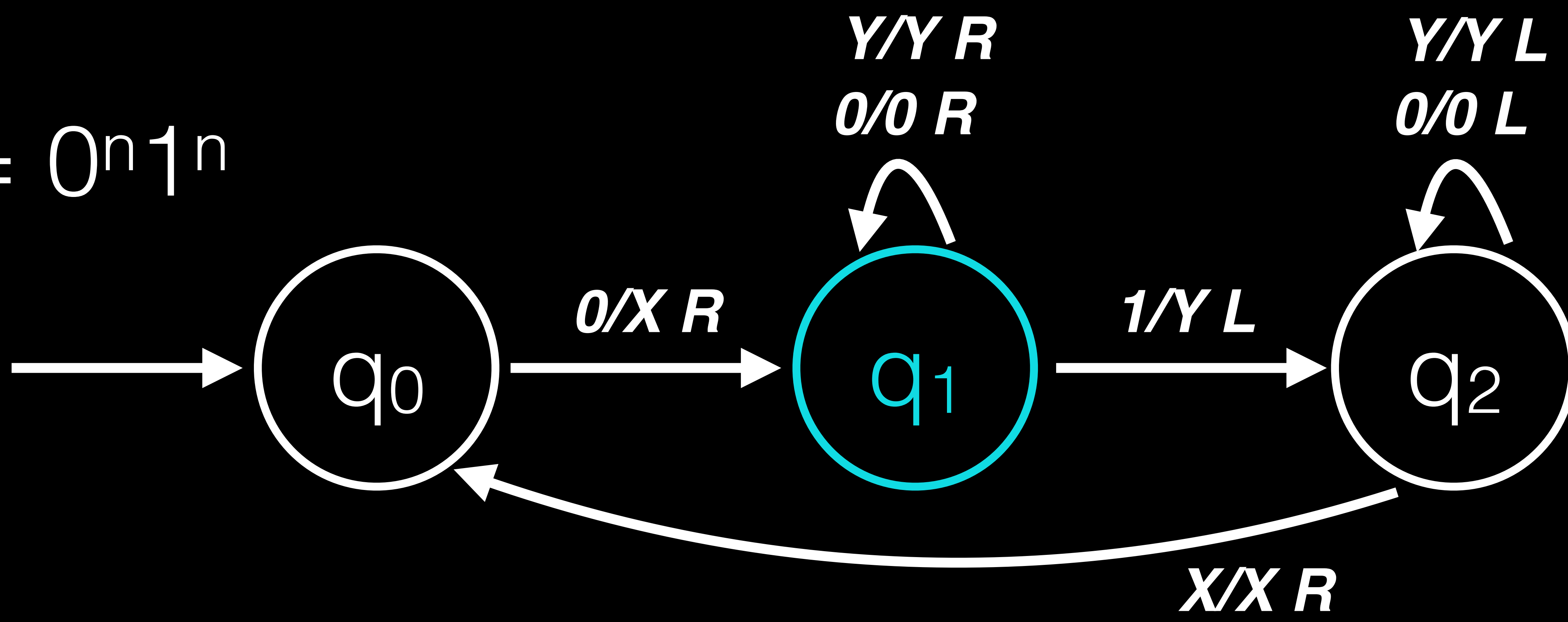
$L = 0^n 1^n$



X X 0 Y Y 1

EXAMPLE

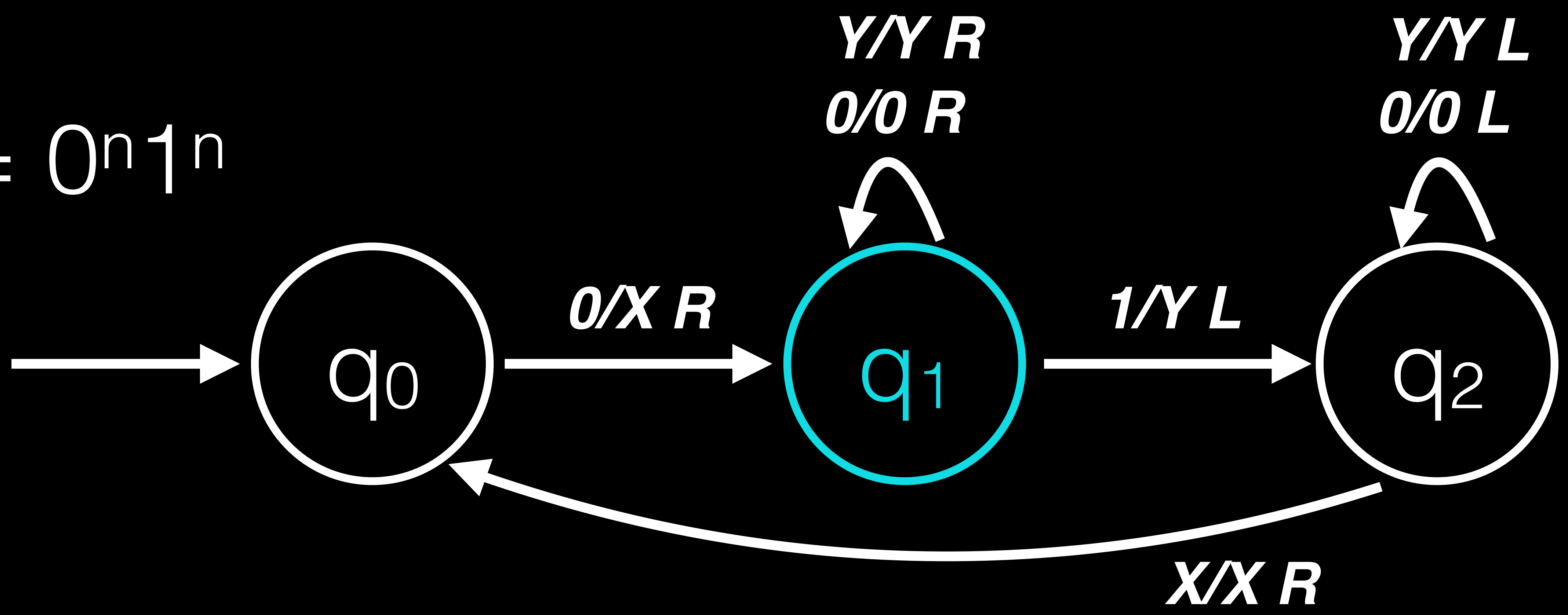
$L = 0^n 1^n$



X X X Y Y 1

EXAMPLE

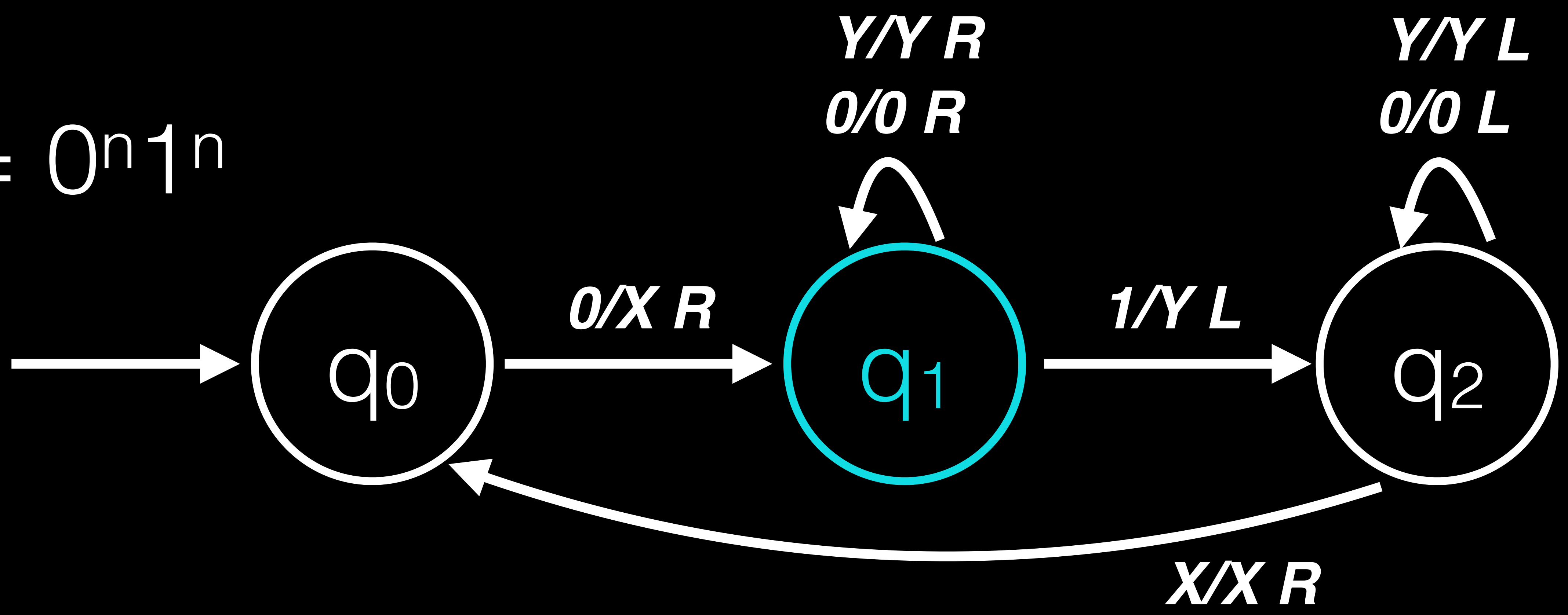
$L = 0^n 1^n$



X X X Y Y 1

EXAMPLE

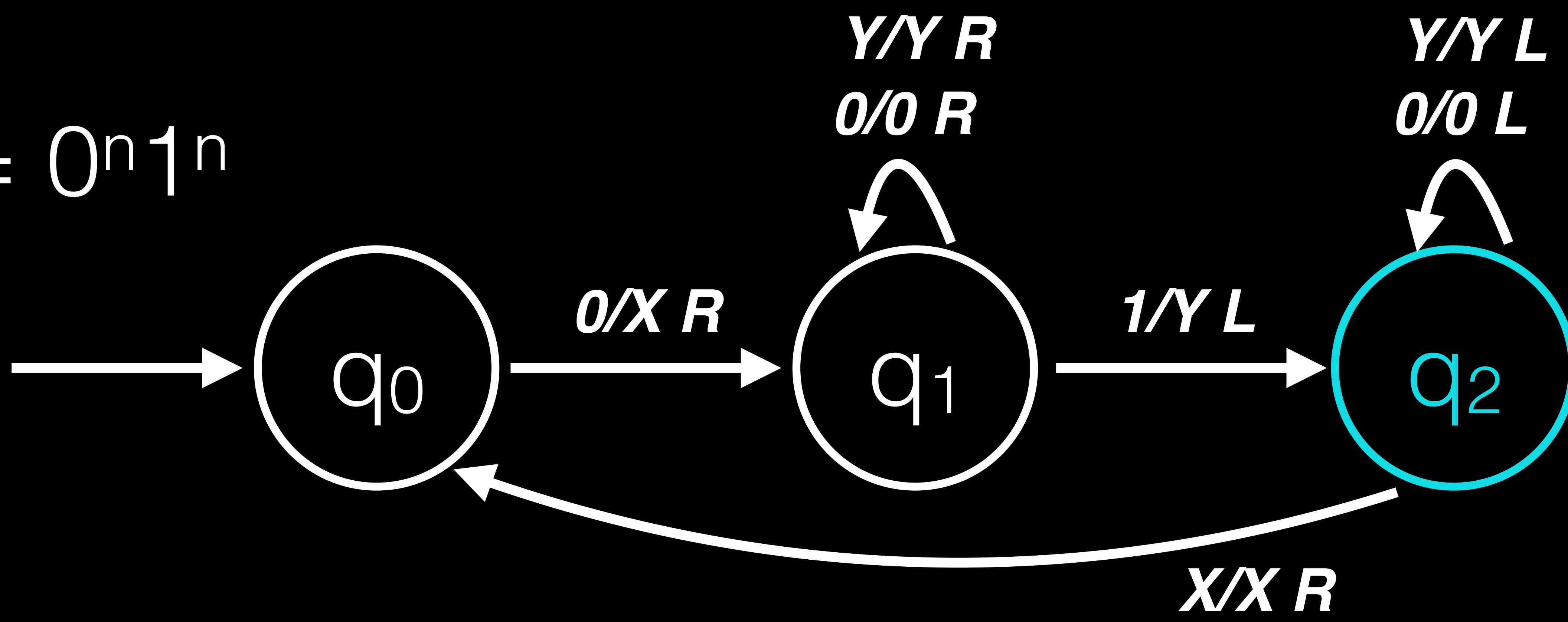
$L = 0^n 1^n$



X X X Y Y 1

EXAMPLE

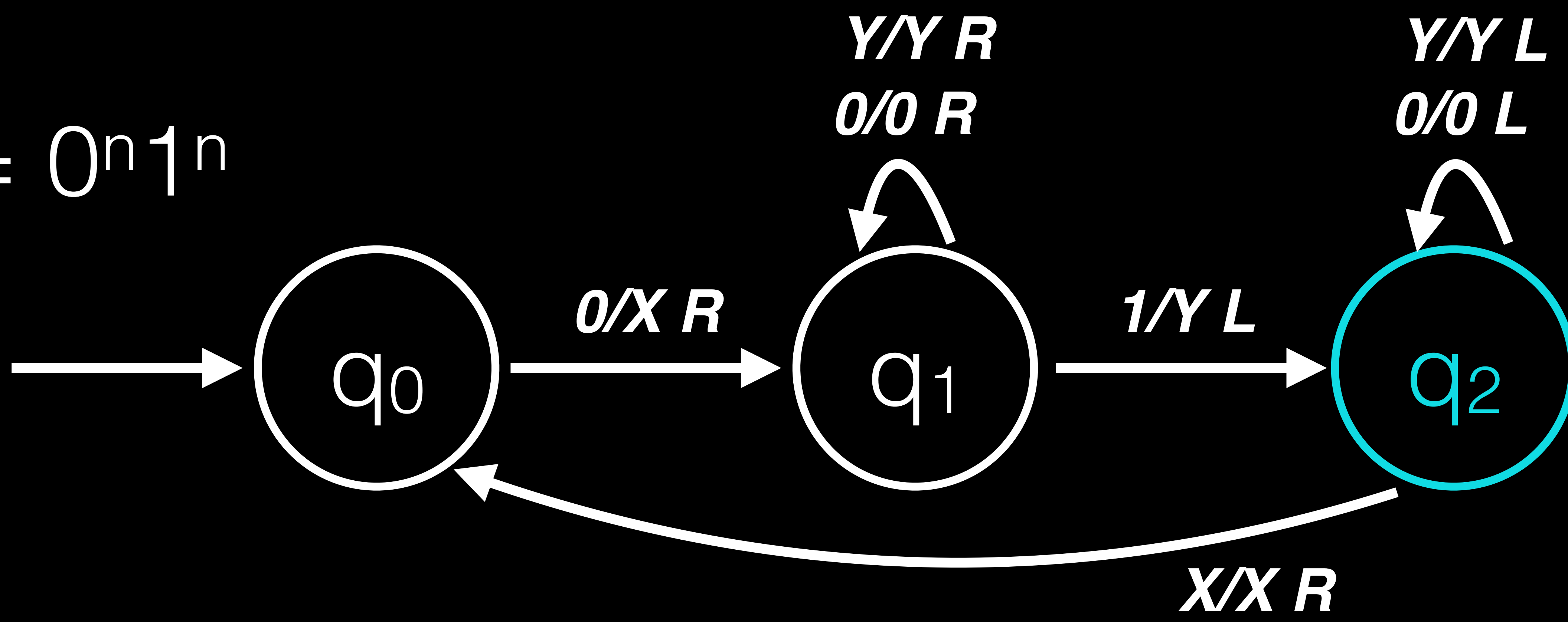
$L = 0^n 1^n$



X X X Y Y Y

EXAMPLE

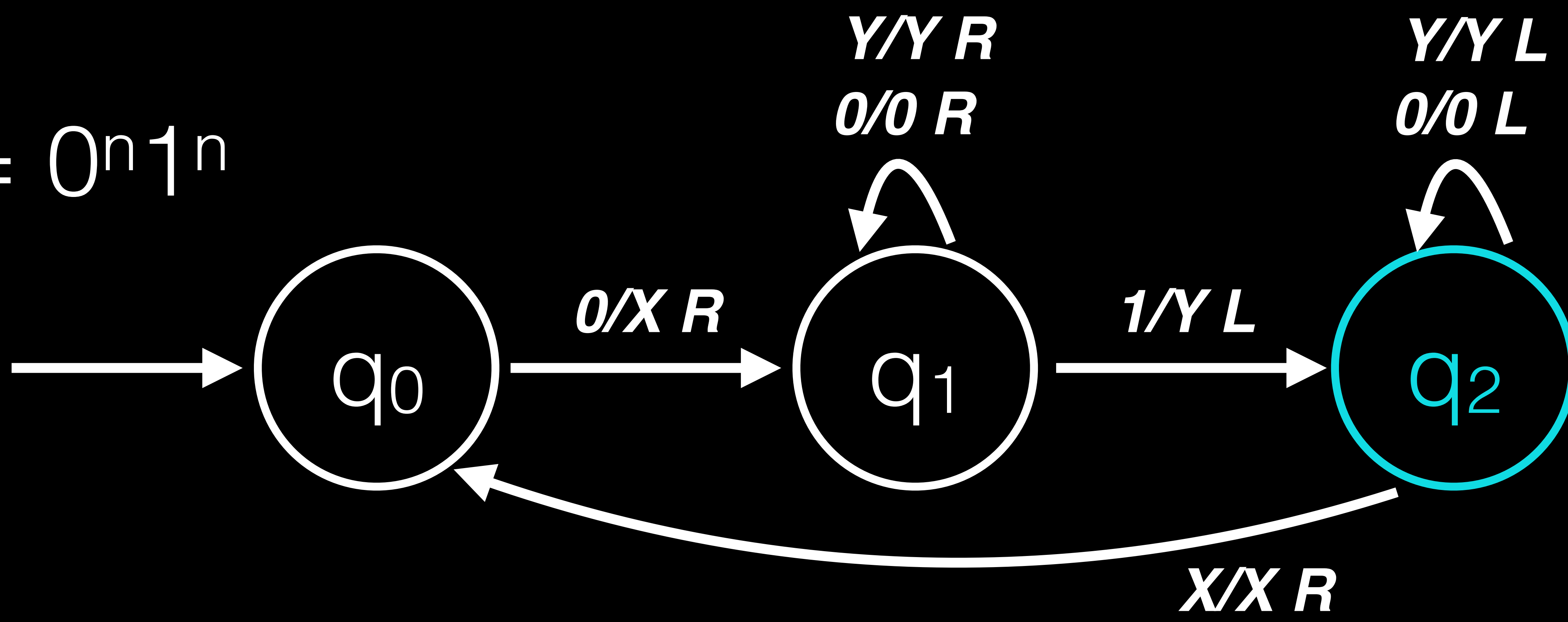
$L = 0^n 1^n$



X X X Y Y Y

EXAMPLE

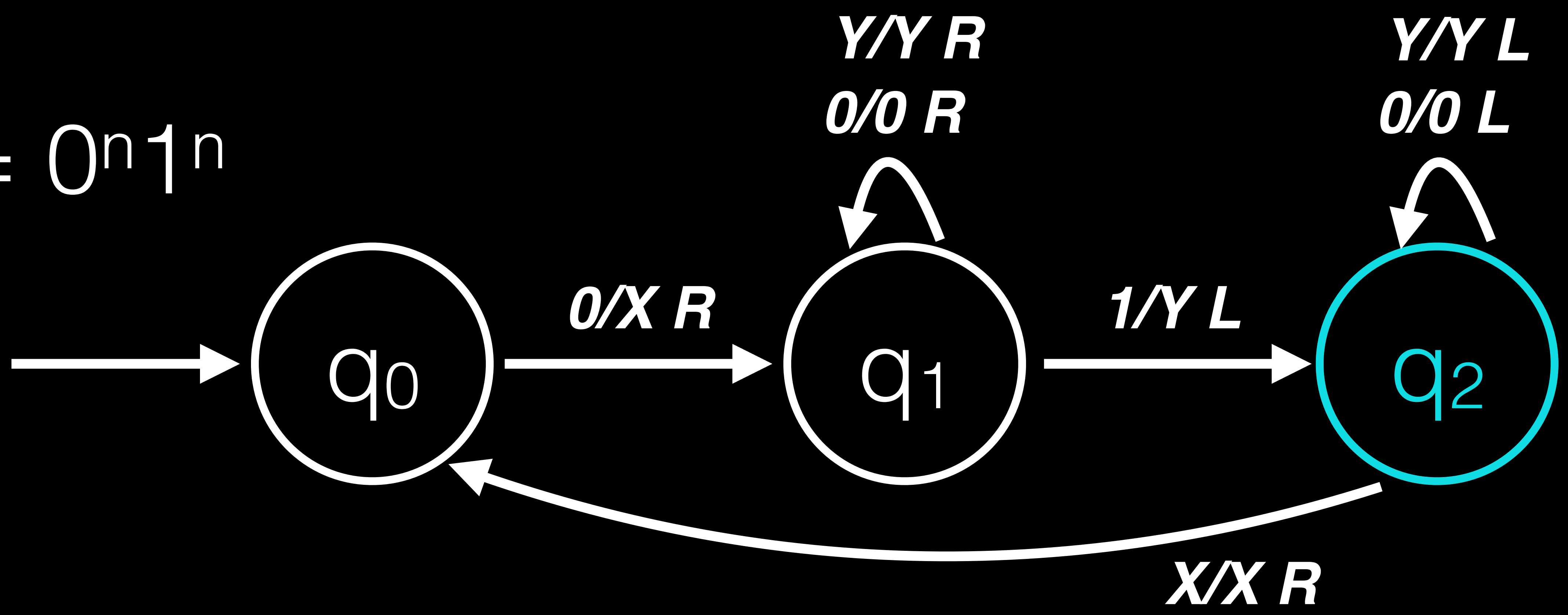
$L = 0^n 1^n$



X X X Y Y Y

EXAMPLE

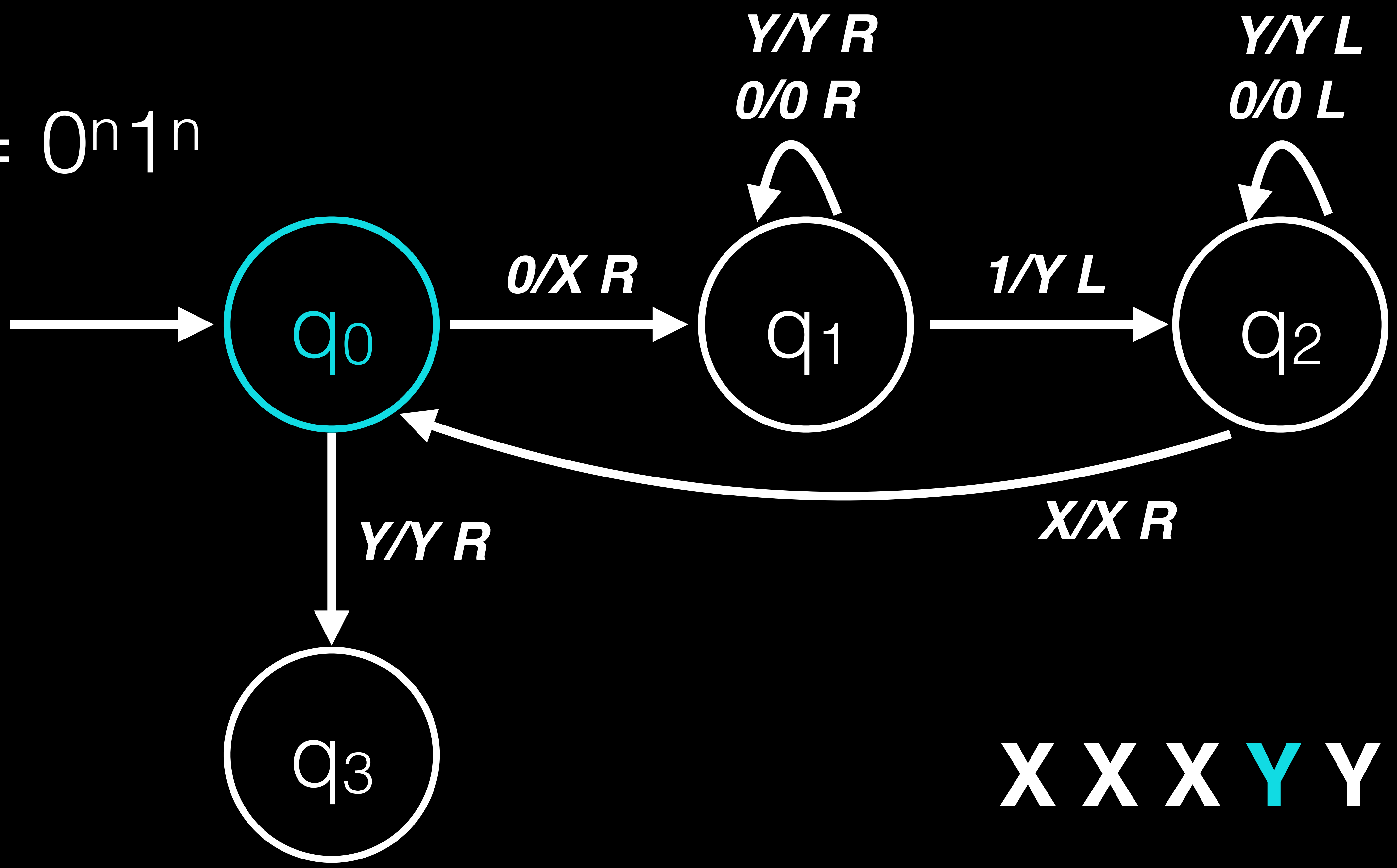
$L = 0^n 1^n$



X X X Y Y Y

EXAMPLE

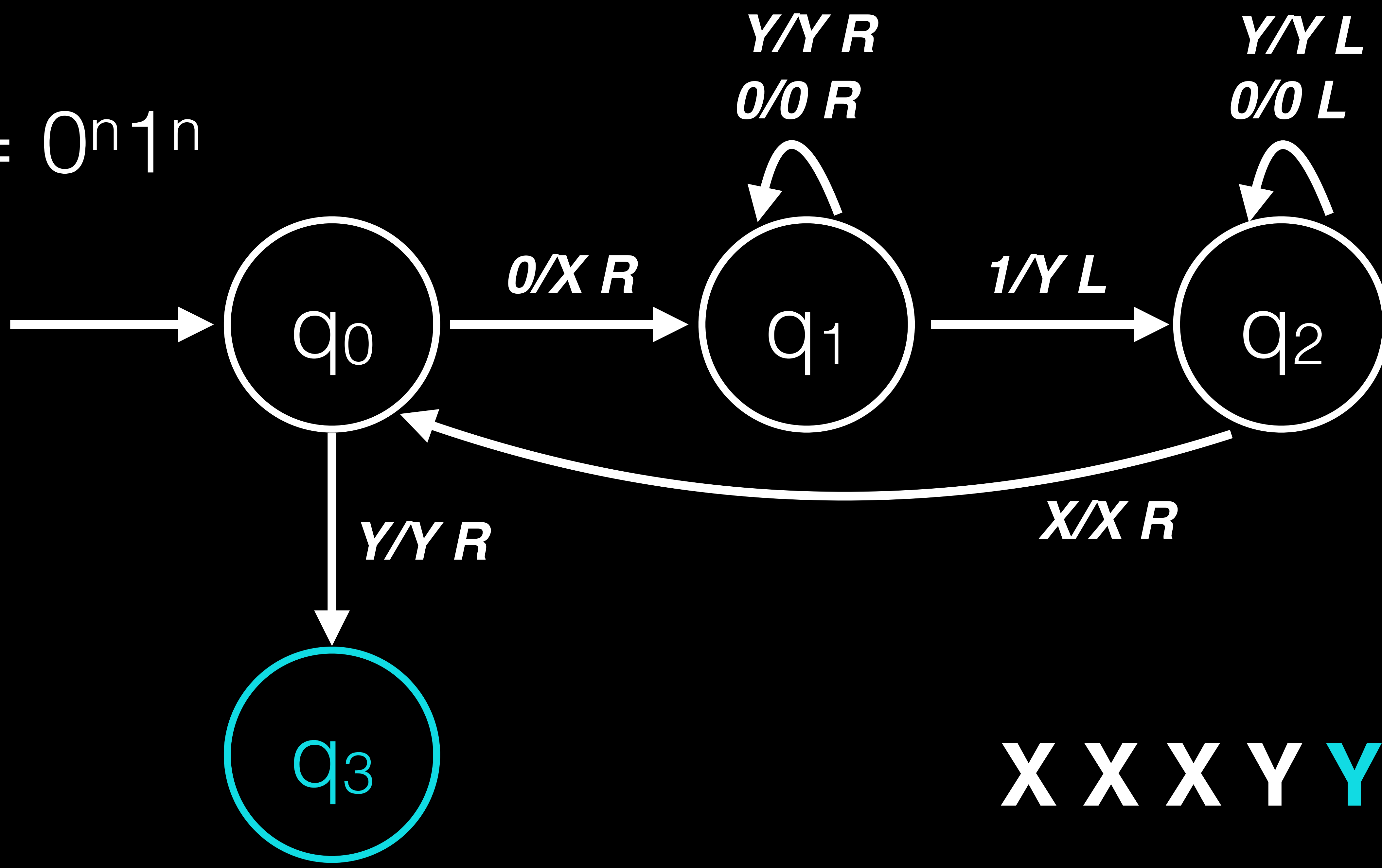
$L = 0^n 1^n$



X X X Y Y Y

EXAMPLE

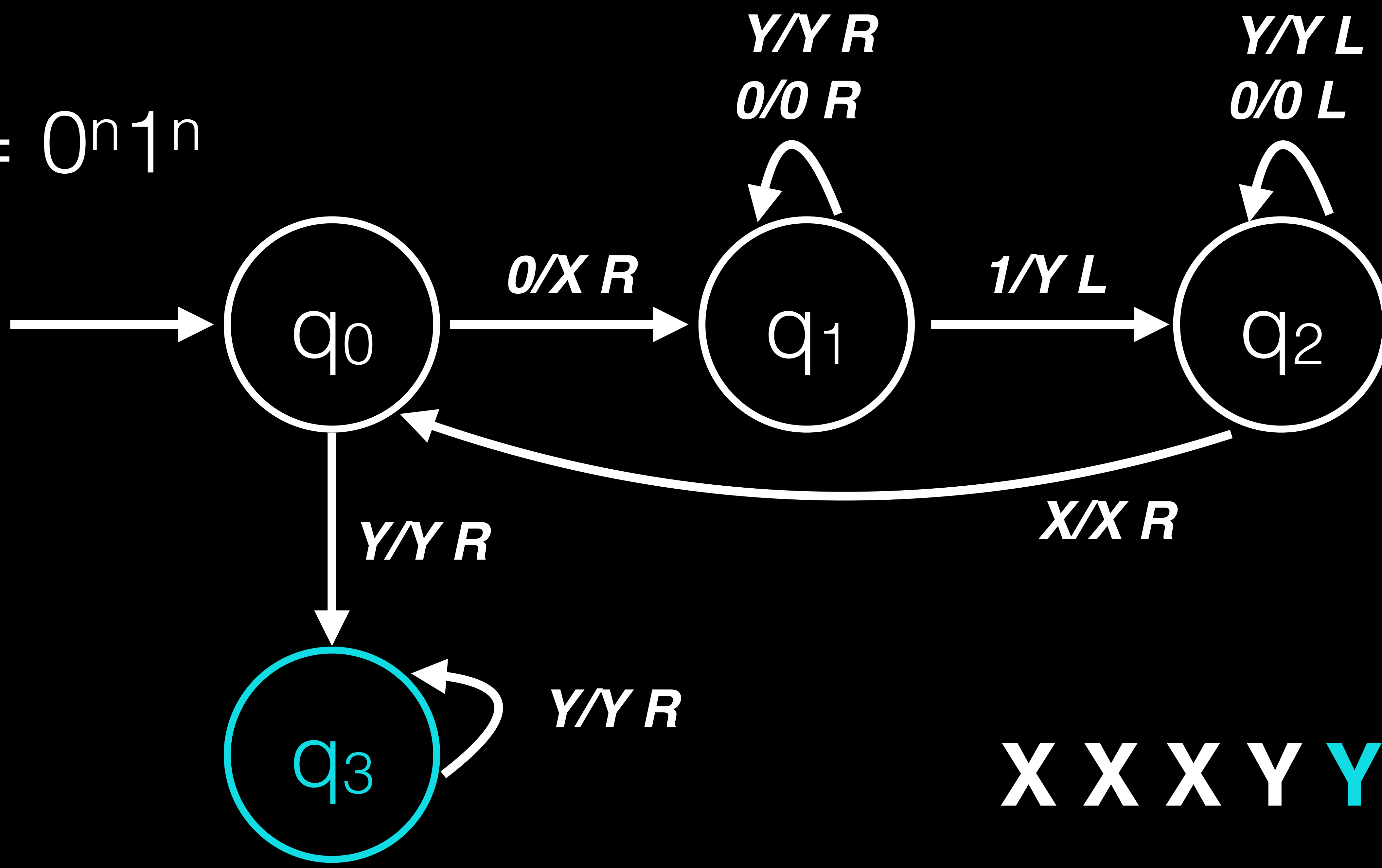
$L = 0^n 1^n$



X X X Y Y Y

EXAMPLE

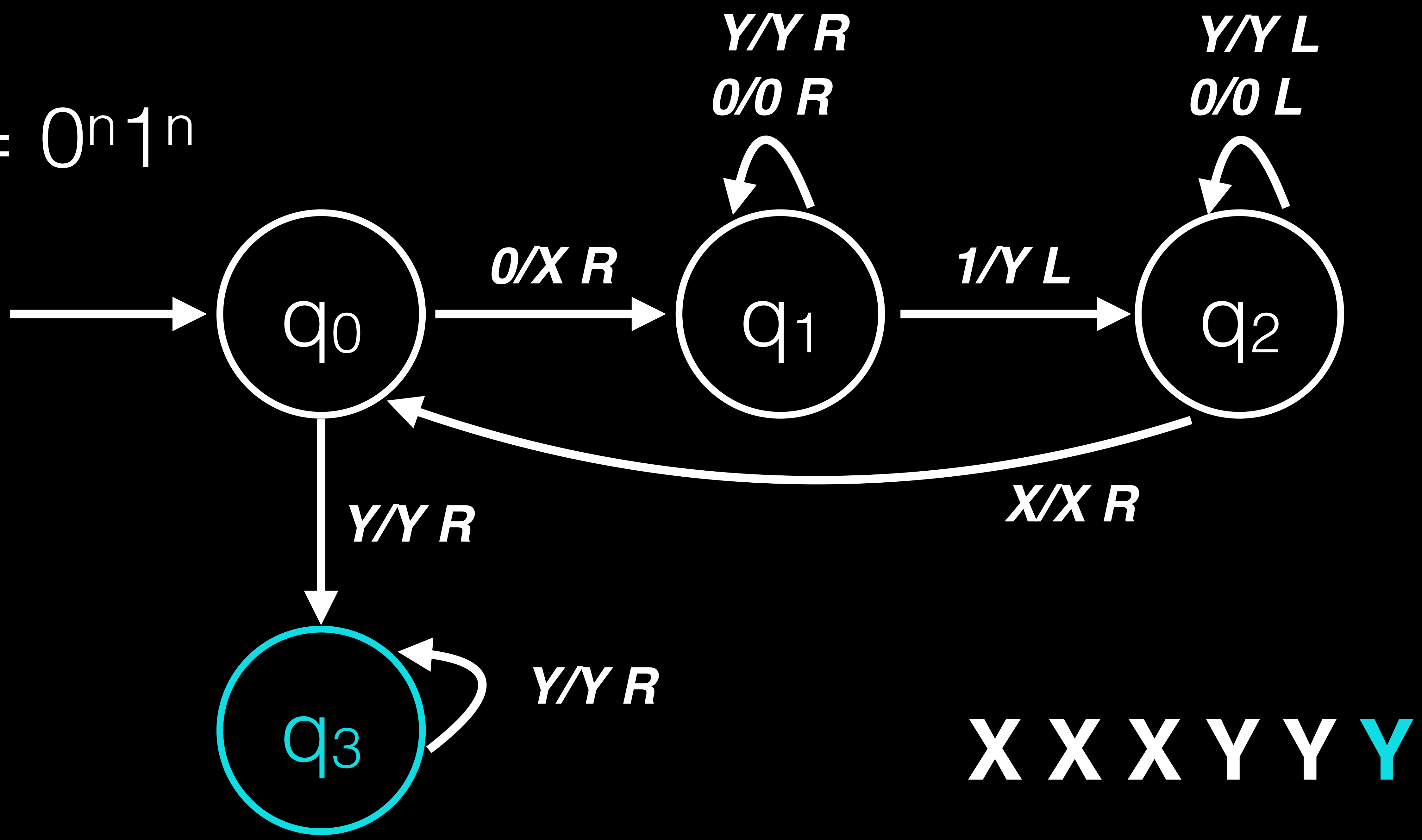
$L = 0^n 1^n$



X X X Y Y Y

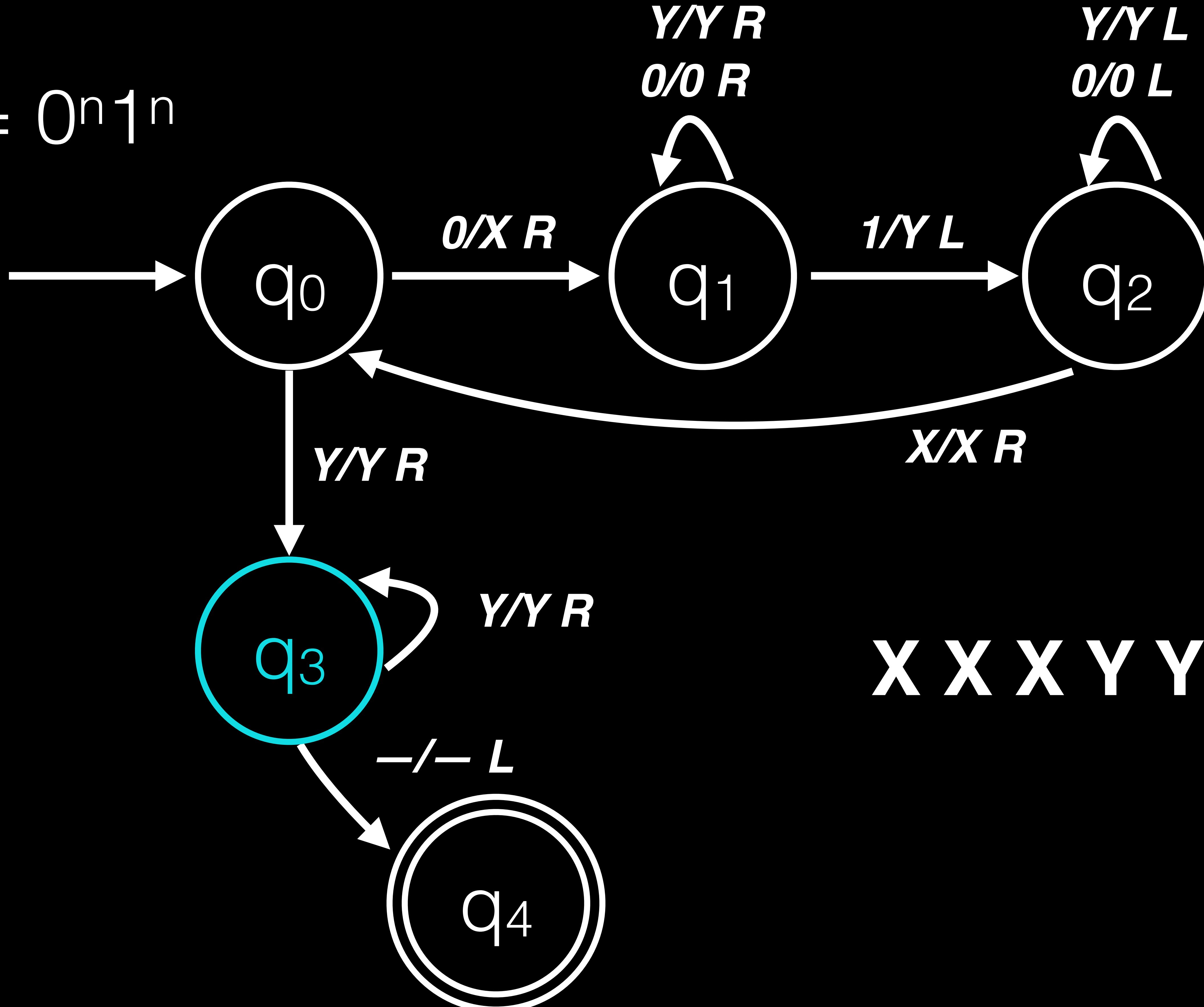
EXAMPLE

$L = 0^n 1^n$



EXAMPLE

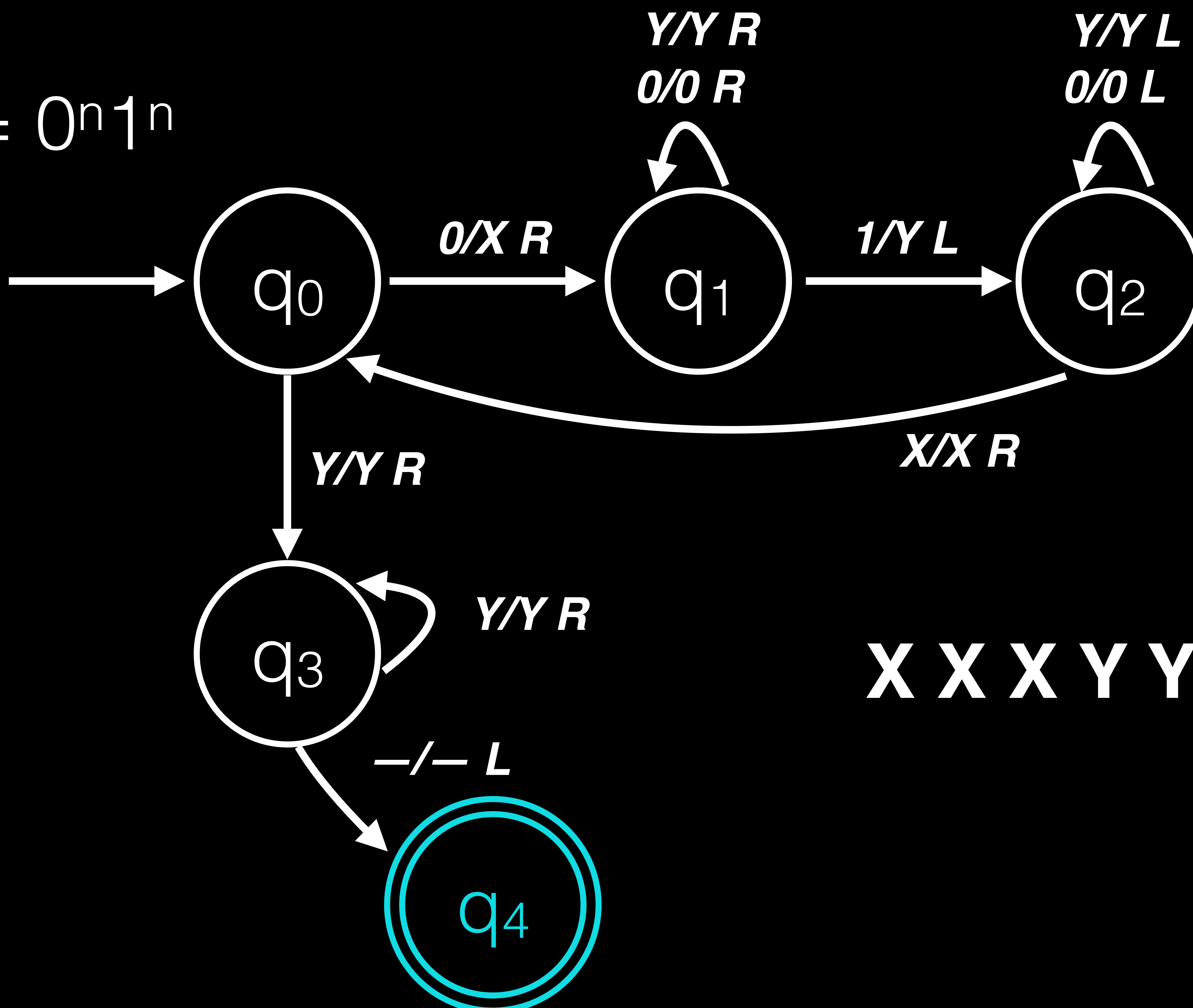
$L = 0^n 1^n$



X X X Y Y Y

EXAMPLE

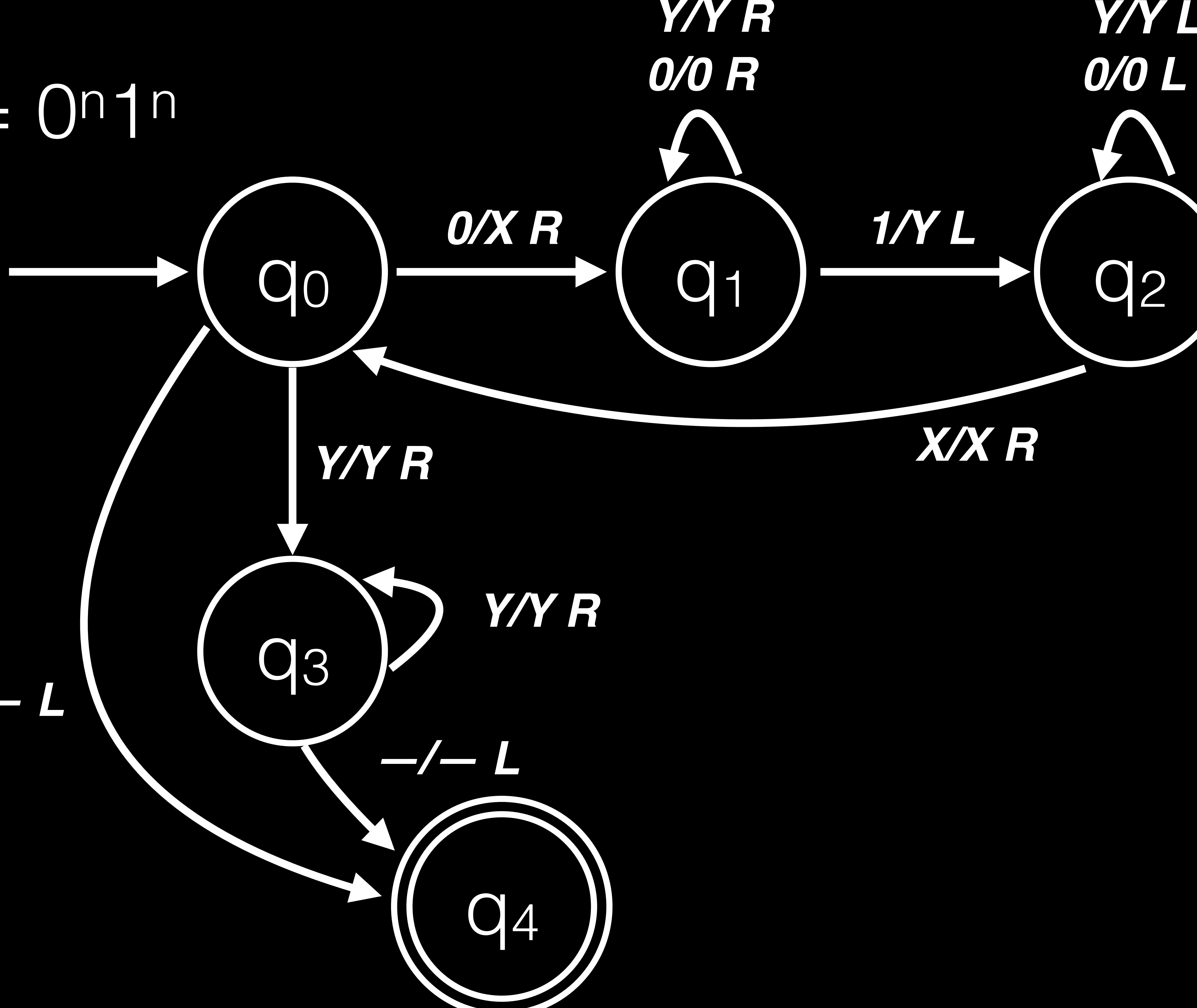
$L = 0^n 1^n$



X X X Y Y Y

EXAMPLE

$L = 0^n 1^n$



EXAMPLE

Questions:

- ⋆ Is this machine correct?
- ⋆ Does it work?
- ⋆ Does it contain bugs?
  - TMs model computers
  - In this way they are similar!

Definition of TMs and Related Language Classes

FORMAL DEFINITION

$(Q, \Sigma, \Gamma, \delta, q_0, q_{ACCEPT}, q_{REJECT})$

- Q = Set of states
- Σ = Input alphabet
- Γ = Tape Alphabet
  - Often we need a few extra symbols to make computation easier. $\Sigma \subseteq \Gamma$
  - The input cannot contain a blank. $- \notin \Sigma$ and $- \in \Gamma$
- q_0 = Initial state $q_0 \in Q$
- q_{ACCEPT} $q_{ACCEPT} \in Q$
- q_{REJECT} $q_{REJECT} \in Q$
- $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ Transition Function

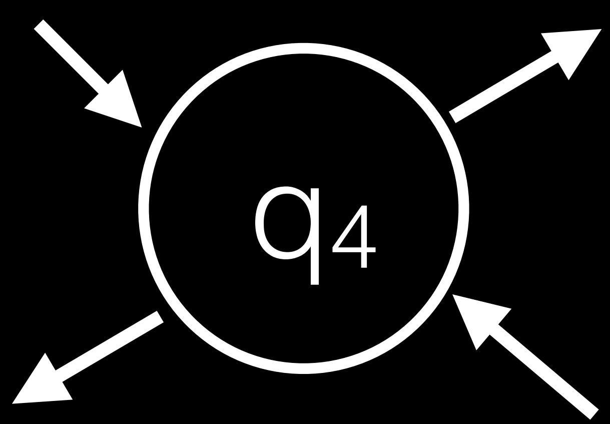
CONFIGURATION

- Gives the entire state of the machine.
- Snapshot of execution at some step.

CONFIGURATION

Need:

- Contents of the tape
- Location of the “tape head”
- Current state



A configuration is a string like this:

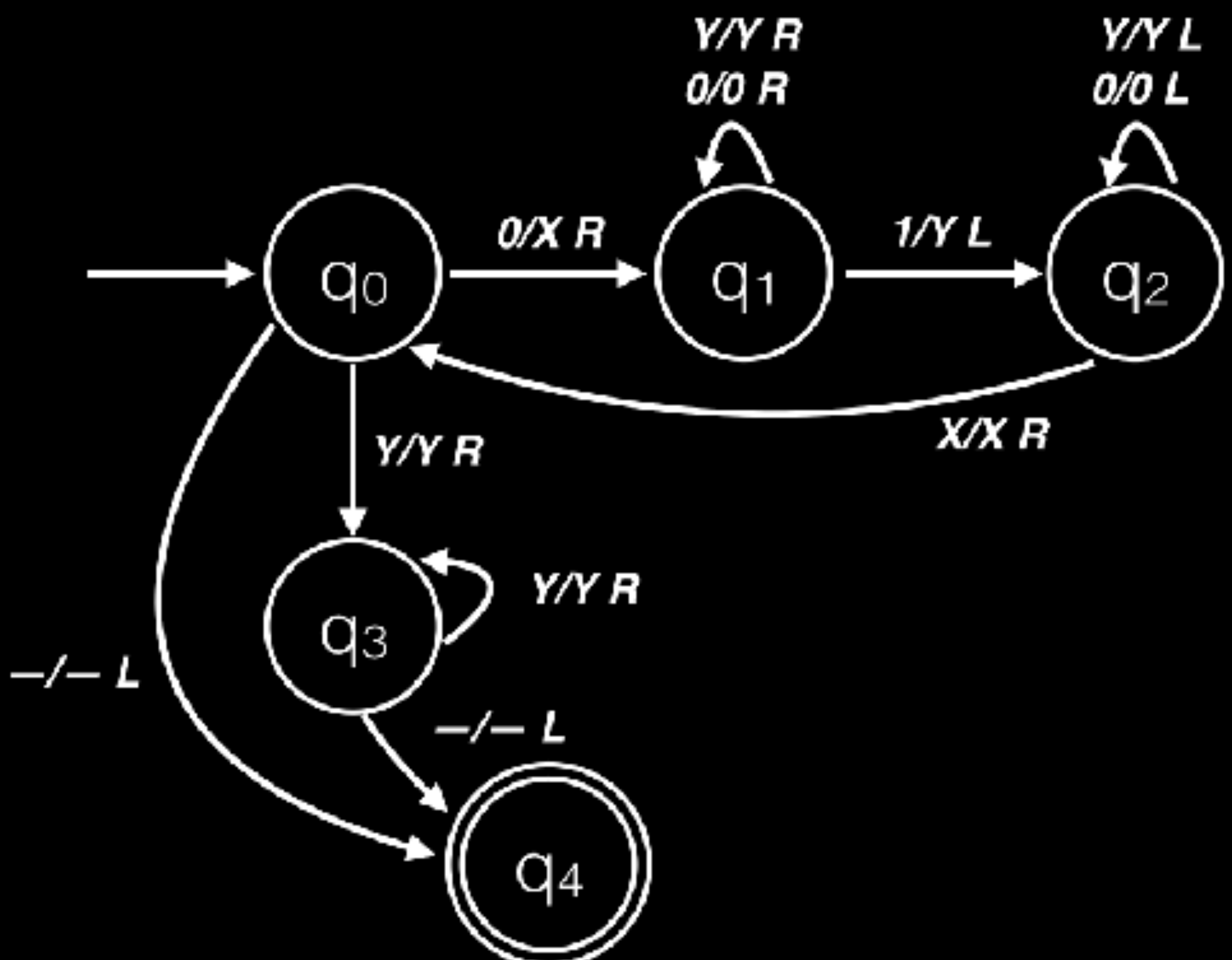
$$0\ 0\ 1\ 1\ q_4\ 0\ 1\ 1$$

A sequence of configurations, starting with the **start configuration**, and ending with an **[Accepting]* configuration** and containing only legal transitions provide a c**omputation history**.

** Rejecting*

EXAMPLE

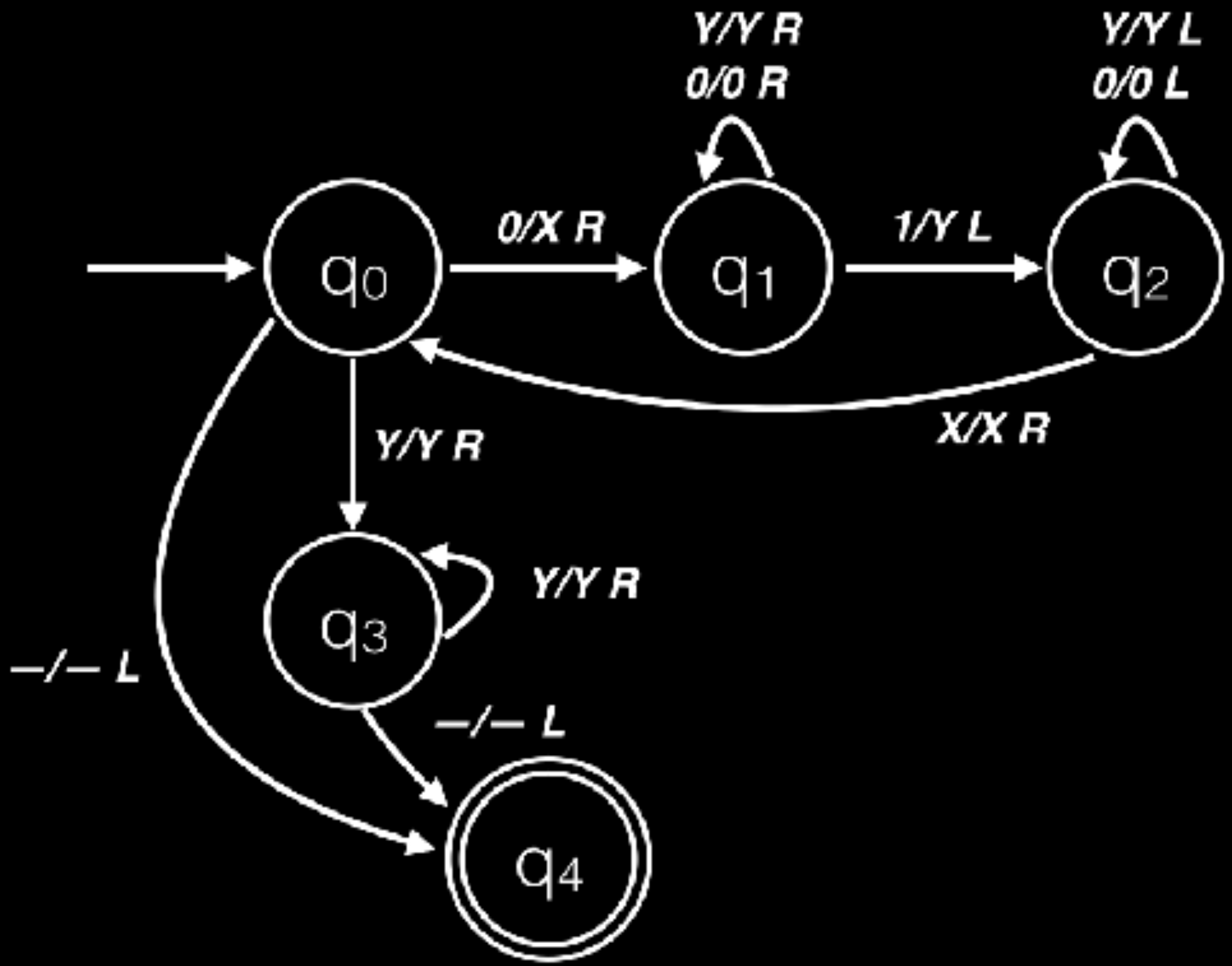
$L = 0^n1^n$, given the string 0011.



EXAMPLE

L = 0^n1^n, given the string 0011.

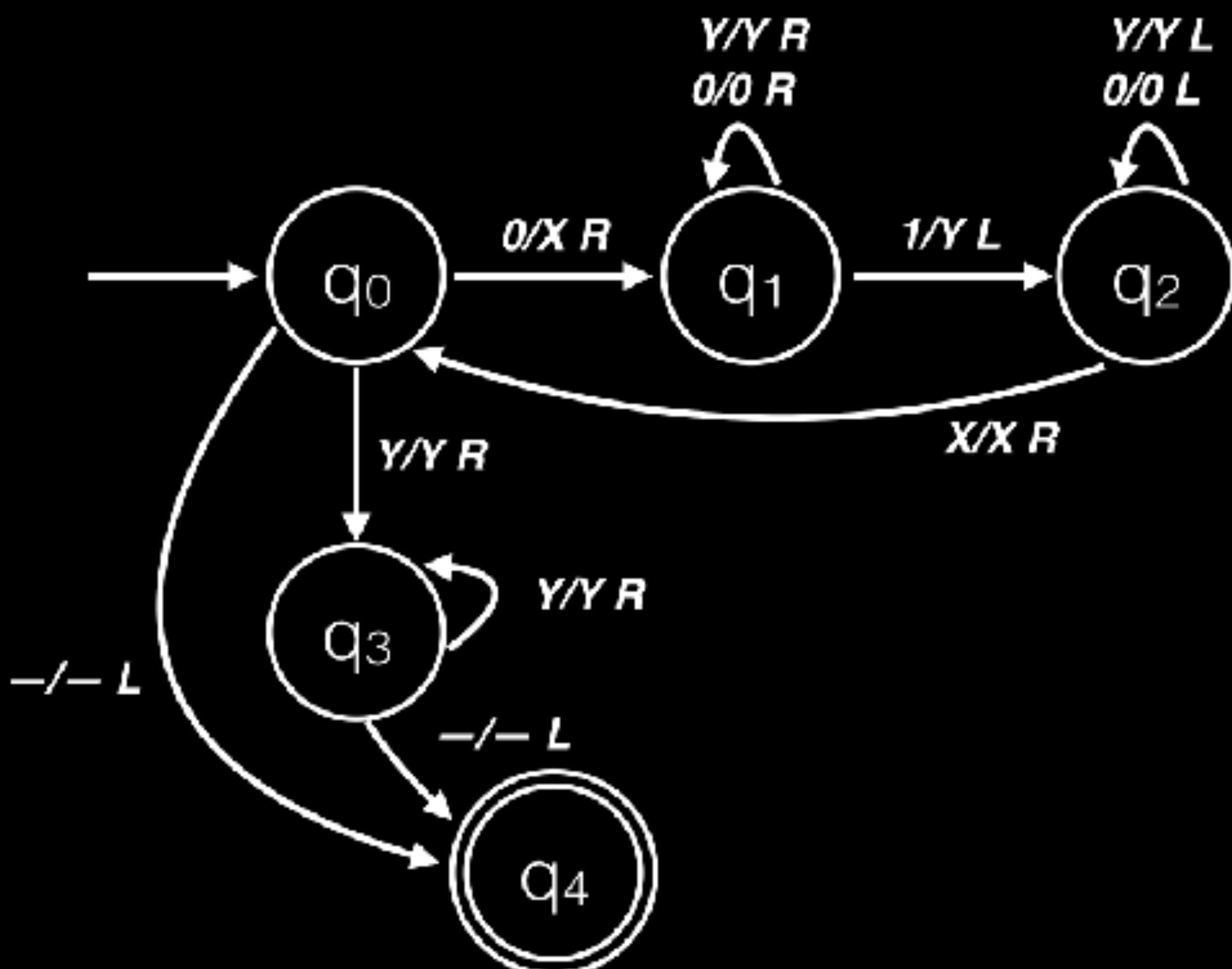
$q_0$0011



EXAMPLE

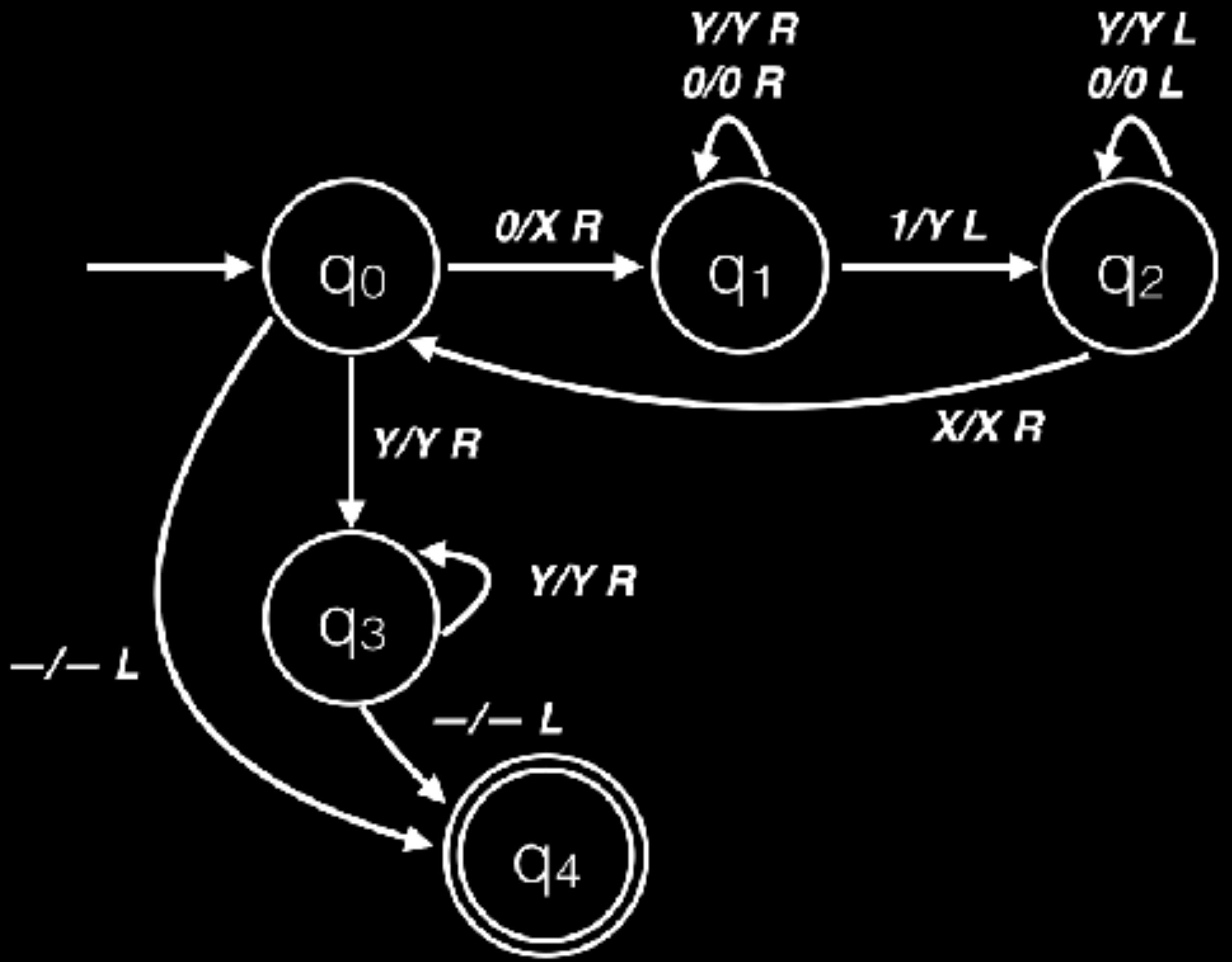
$L = 0^n1^n$, given the string 0011.

$q_0$0011
X$q_1$011



EXAMPLE

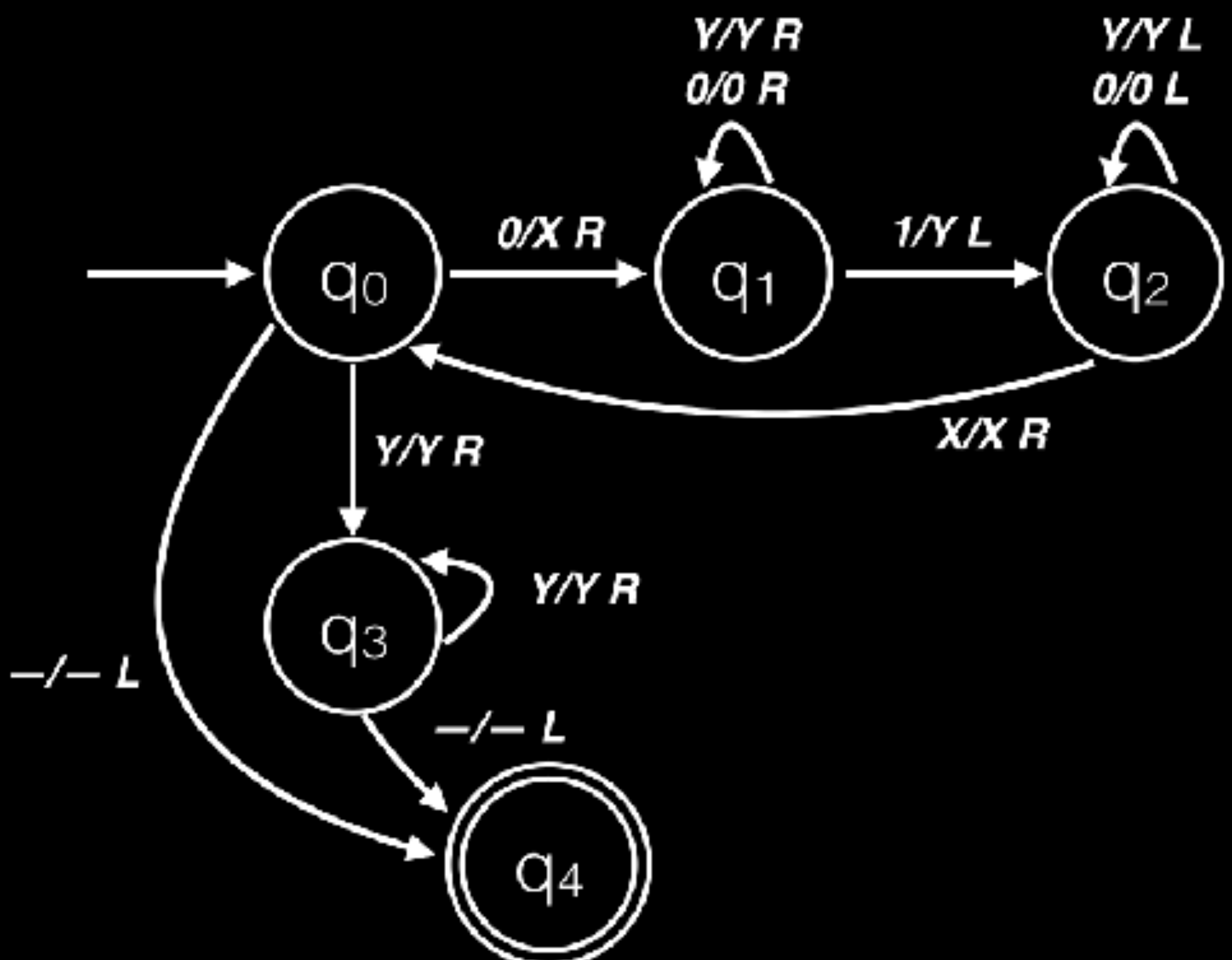
$L = 0^n1^n$, given the string 0011.



$q_0$0011
X$q_1$011
X0$q_1$11

EXAMPLE

$L = 0^n1^n$, given the string 0011.



$q_0$0011

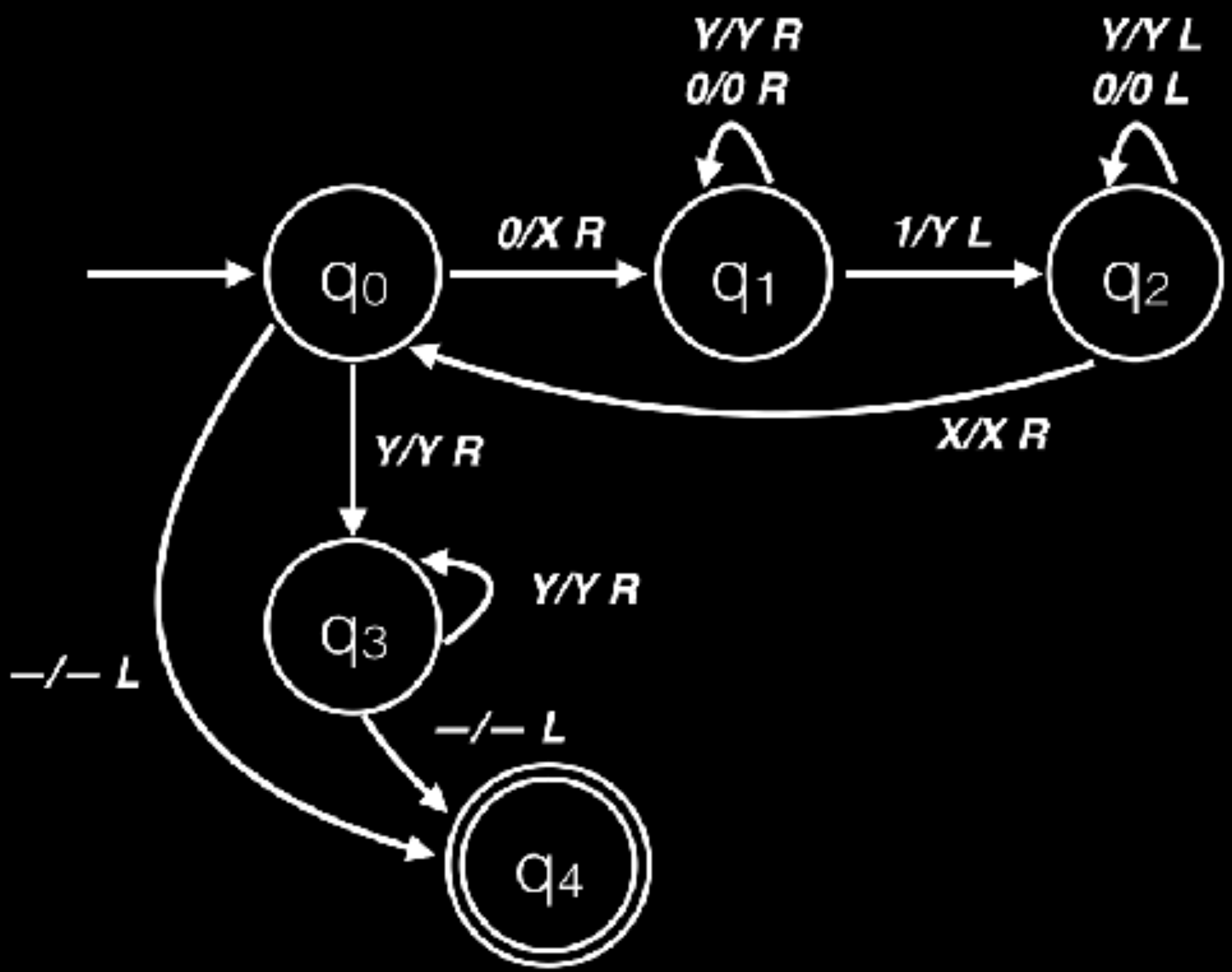
X$q_1$011

X0$q_1$11

X$q_2$0Y1

EXAMPLE

$L = 0^n1^n$, given the string 0011.



$q_0 0011$

$Xq_1 011$

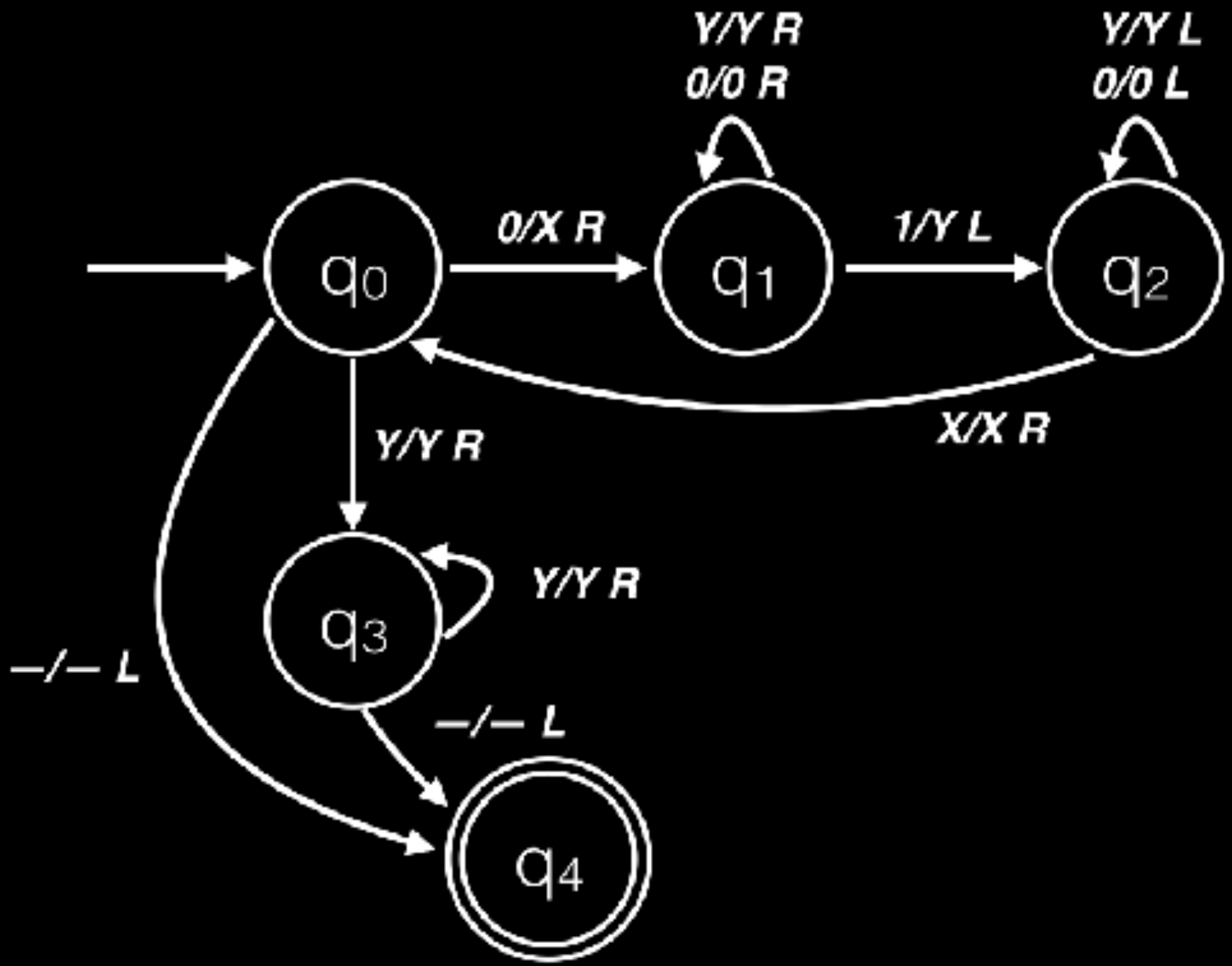
$X0q_1 11$

$Xq_2 0Y1$

$q_2 X0Y1$

EXAMPLE

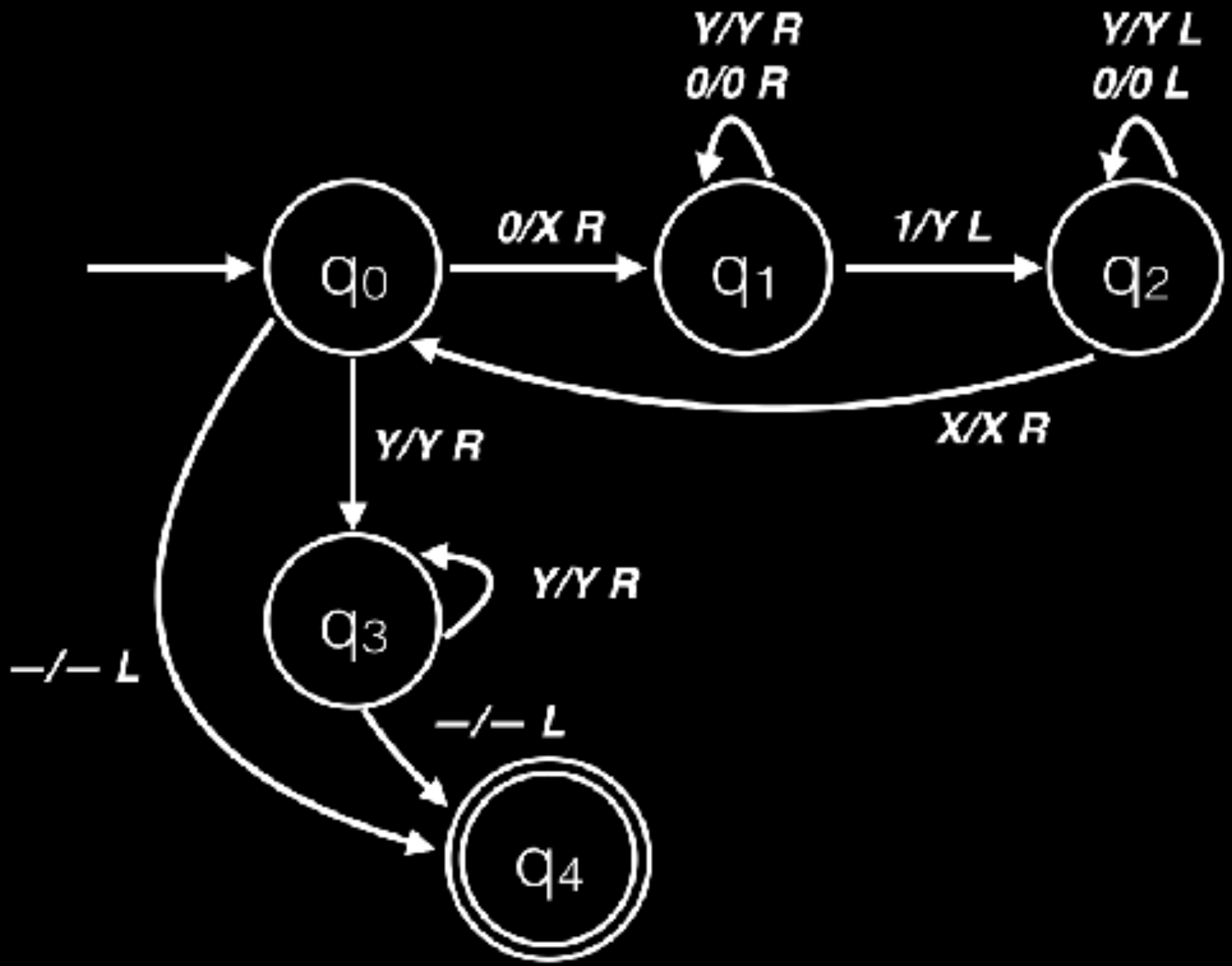
L = 0^n1^n, given the string 0011.



$q_0$0011
X$q_1$011
X0$q_1$11
X$q_2$0Y1
q_2X0Y1
X$q_0$0Y1

EXAMPLE

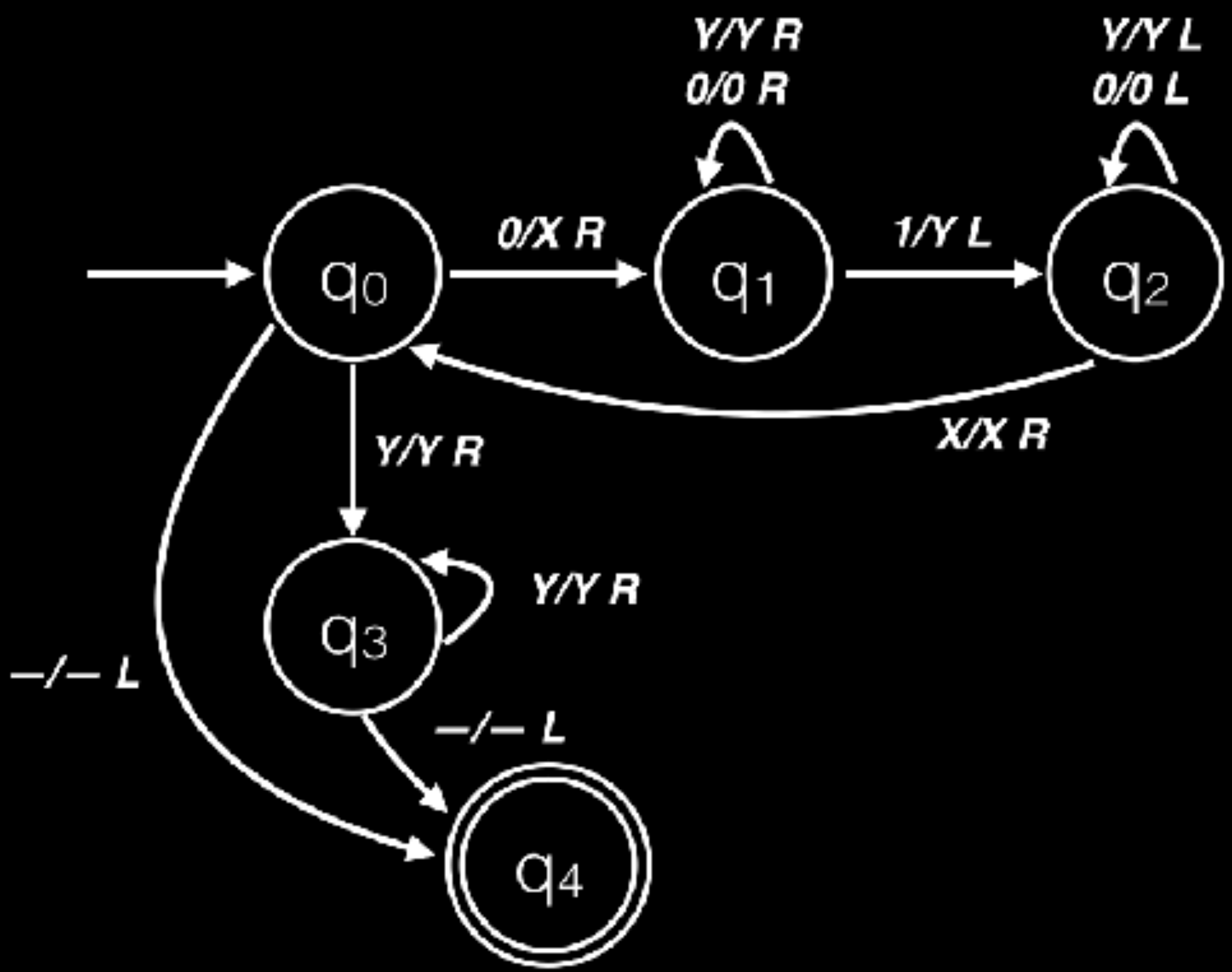
$L = 0^n 1^n$, given the string 0011.



$q_0 0011$
$Xq_1 011$
$X0q_1 11$
$Xq_2 0Y1$
$q_2 X0Y1$
$Xq_0 0Y1$
$XXq_1 Y1$

EXAMPLE

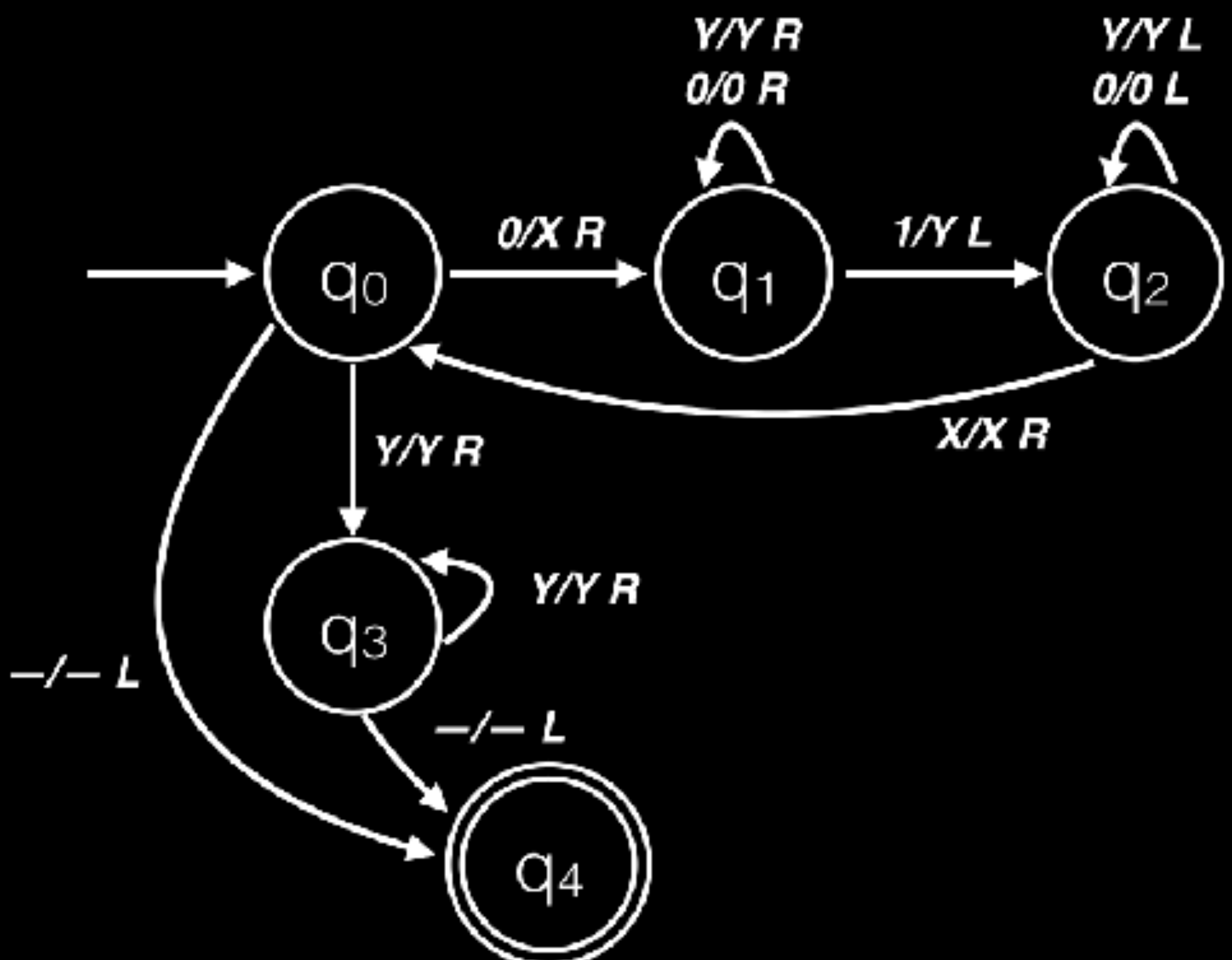
$L = 0^n1^n$, given the string 0011.



$q_0$0011
X$q_1$011
X0$q_1$11
X$q_2$0Y1
q_2X0Y1
X$q_0$0Y1
XXq_1Y1
XXY$q_1$1

EXAMPLE

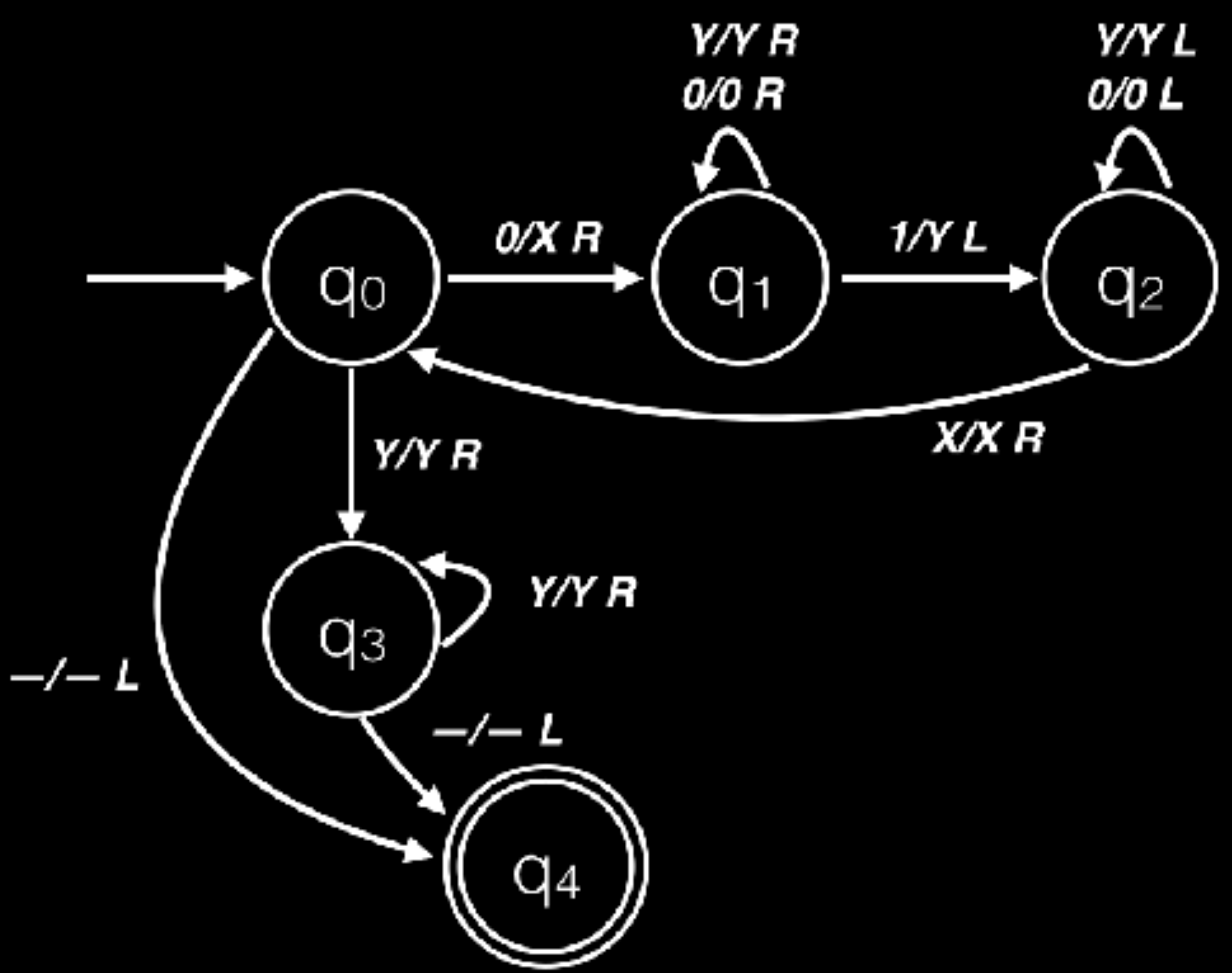
$L = 0^n1^n$, given the string 0011.



$q_0 0011$
$Xq_1 011$
$X0q_1 11$
$Xq_2 0Y1$
$q_2 X0Y1$
$Xq_0 0Y1$
$XXq_1 Y1$
$XXYq_1 1$
$XXq_2 YY$

EXAMPLE

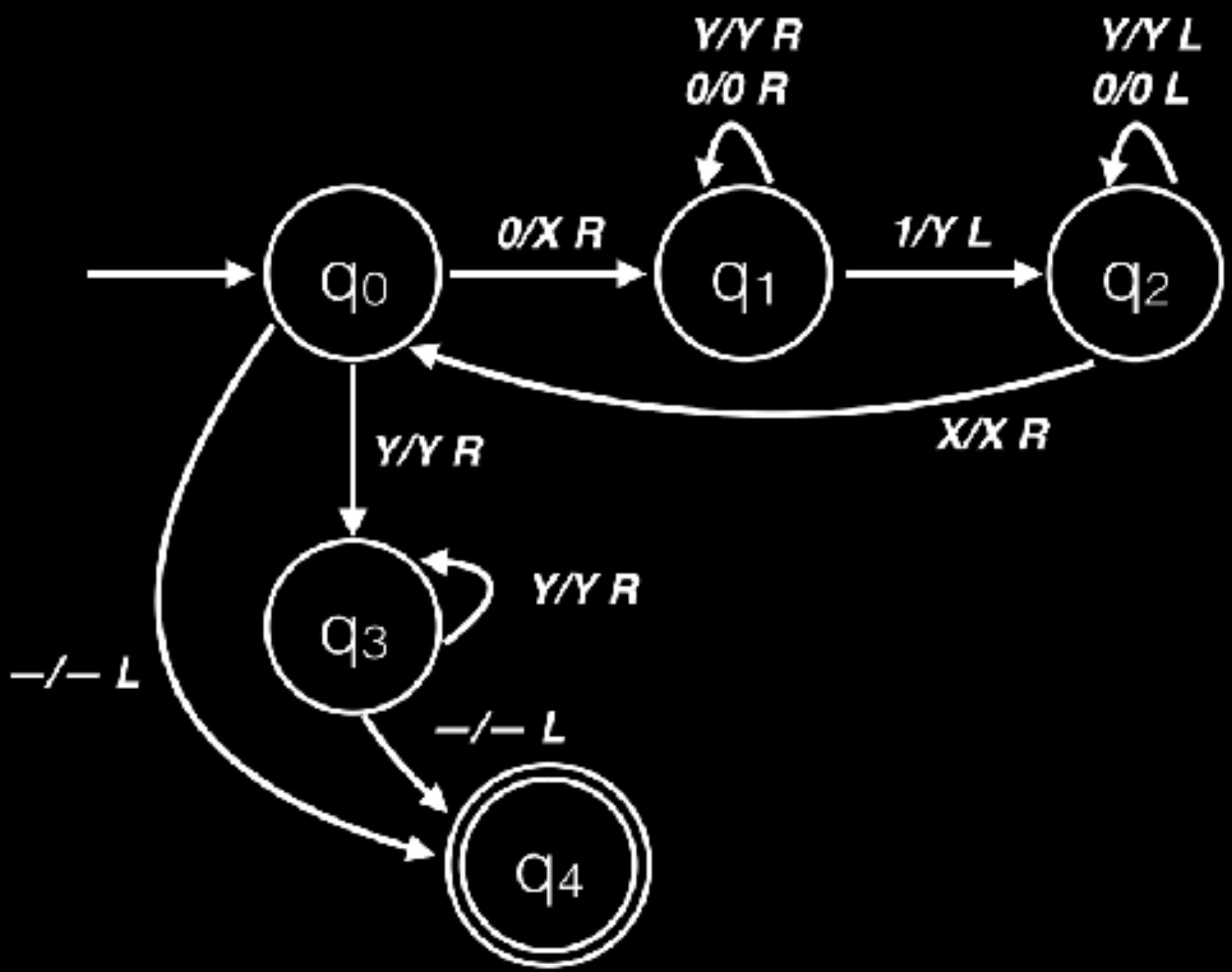
$L = 0^n1^n$, given the string 0011.



$q_0 0011$
$Xq_1 011$
$X0q_1 11$
$Xq_2 0Y1$
$q_2 X0Y1$
$Xq_0 0Y1$
$XXq_1 Y1$
$XXYq_1 1$
$XXq_2 YY$
$Xq_2 XYY$

EXAMPLE

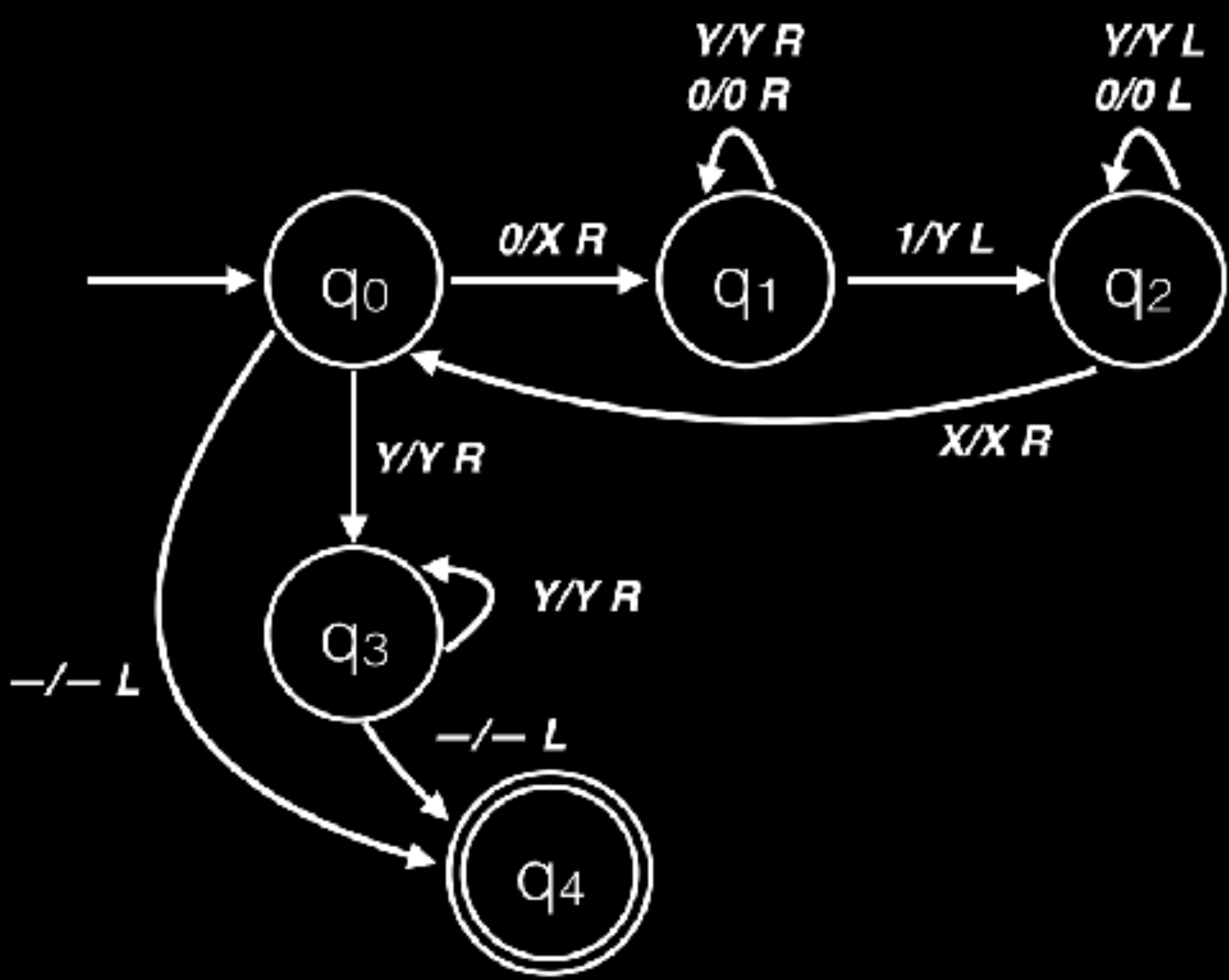
$L = 0^n1^n$, given the string 0011.



$q_0 0011$
$Xq_1 011$
$X0q_1 11$
$Xq_2 0Y1$
$q_2 X0Y1$
$Xq_0 0Y1$
$XXq_1 Y1$
$XXYq_1 1$
$XXq_2 YY$
$Xq_2 XYY$
$XXq_0 YY$

EXAMPLE

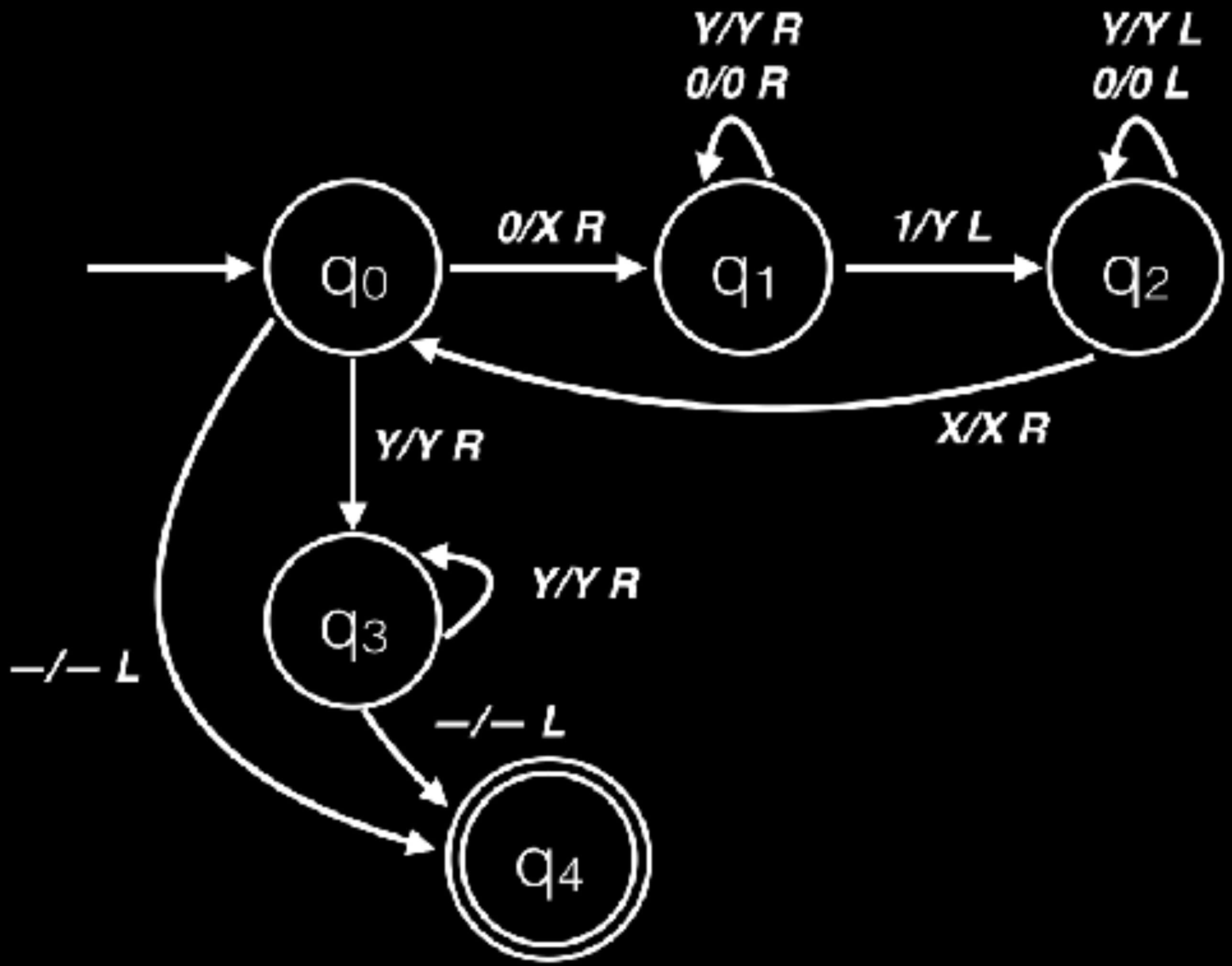
$L = 0^n 1^n$, given the string 0011.



$q_0 0011$
$X q_1 011$
$X0 q_1 11$
$X q_2 0Y1$
$q_2 X0Y1$
$X q_0 0Y1$
$XX q_1 Y1$
$XXY q_1 1$
$XX q_2 YY$
$X q_2 XYY$
$XX q_0 YY$
$XXY q_3 Y$

EXAMPLE

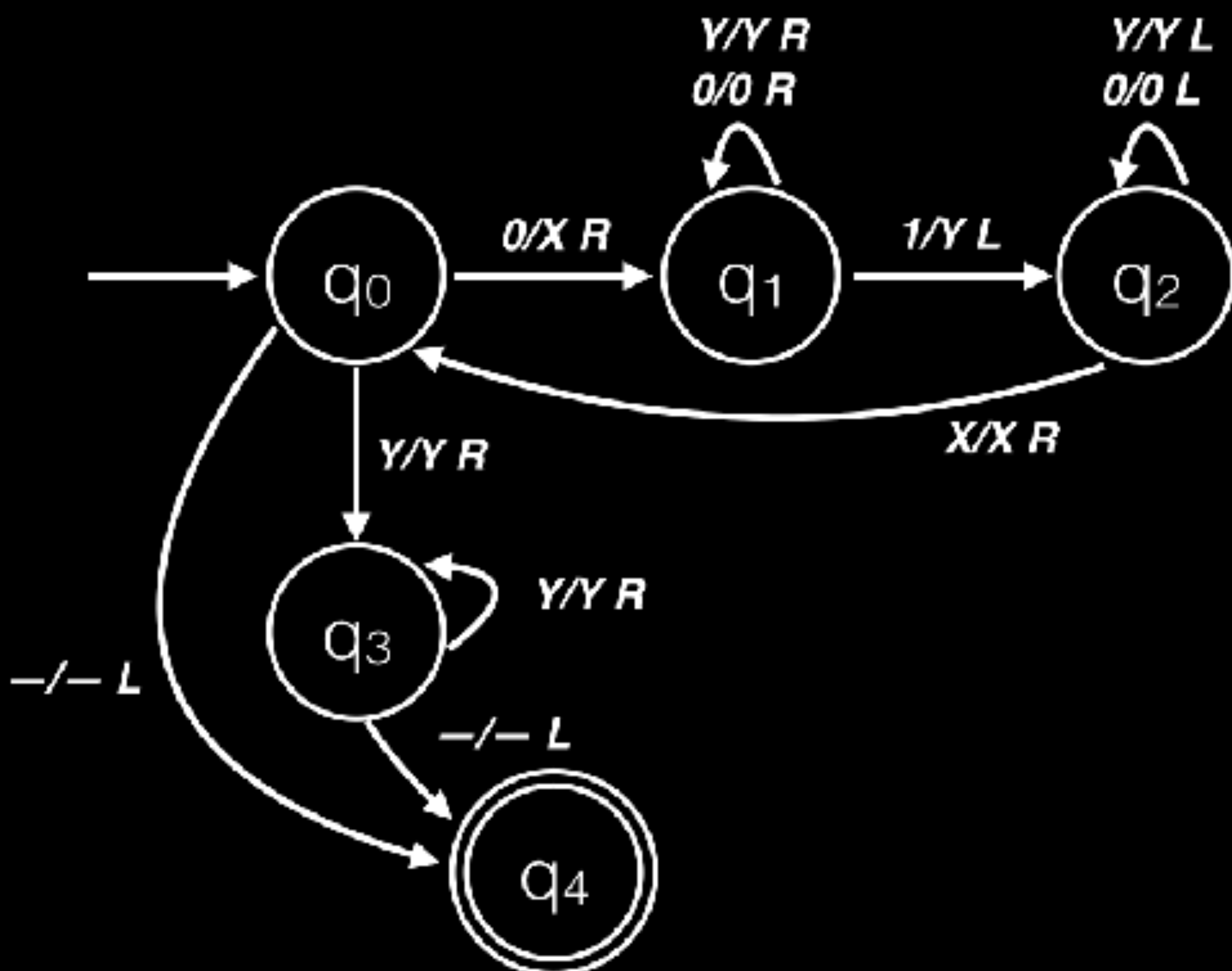
$L = 0^n1^n$, given the string 0011.



$q_0 0011$
$Xq_1 011$
$X0q_1 11$
$Xq_2 0Y1$
$q_2 X0Y1$
$Xq_0 0Y1$
$XXq_1 Y1$
$XXYq_1 1$
$XXq_2 YY$
$Xq_2 XYY$
$XXq_0 YY$
$XXYq_3 Y$
$XXYYq_3 —$

EXAMPLE

$L = 0^n1^n$, given the string 0011.



$q_0$0011
X$q_1$011
X0$q_1$11
X$q_2$0Y1
q_2X0Y1
X$q_0$0Y1
XXq_1Y1
XXY$q_1$1
XXq_2YY
Xq_2XYY
XXq_0YY
XXYq_3Y
XXYYq_3—
XXYY—q_4—

1. Decidable Languages

- When given a string as input, the TM will **always halt**.
  - TM will **accept** if it is in L.
  - TM will **reject** if it is not in L.

- also: recursive, computable, solvable

2. Turing Recognizable Languages

- When given a string that is in the language, the TM will always **halt and accept.**
- If not, the TM will either **reject or loop.**

- also: recursively enumerable, RE, partially decidable, semi-decidable

3. Not Turing Recognizable Languages

- Cannot recognize members reliably!

- also: not recursively enumerable, not RE, not partially decidable

USES OF TURING MACHINE

- To **decide** a language
- To **recognize** a language

USES OF TURING MACHINE

- To **decide** a language
- To **recognize** a language
- **To compute a function**

USES OF TURING MACHINE: TO COMPUTE A FUNCTION

1. Computable Function
    - ★ Computable ≅ Decidable
    - ★ Totally computable
    - ★ Defined on all inputs

2. Partially Computable Functions
   - Undefined on some inputs
   - Semi-decidable functions

The Church-Turing Thesis

Several variations on Turing Machines

- One tape or many?
- Infinite on both ends?
- Tiny alphabet {0, 1} or not?
- Can the head stay in the same place?
- Allow nondeterminism

Several variations on Turing Machines

- One tape or many?
- Infinite on both ends?
- Tiny alphabet {0, 1} or not?
- Can the head stay in the same place?
- Allow nondeterminism

All variations are equivalent in computing capability!

Several variations on Turing Machines

- One tape or many?
- Infinite on both ends?
- Tiny alphabet {0, 1} or not?
- Can the head stay in the same place?
- Allow nondeterminism

All variations are equivalent in computing capability!

TMs and Lambda Calculus are also equivalent in power.

CONCLUSION / DEFINITION

Algorithmically Computable

=

Computable by a TM

Turing Machine

$\neq$

Turing Test

Turing Machine Programming Techniques

COMMON PROBLEM

How can we recognize the left end of the tape?

COMMON PROBLEM

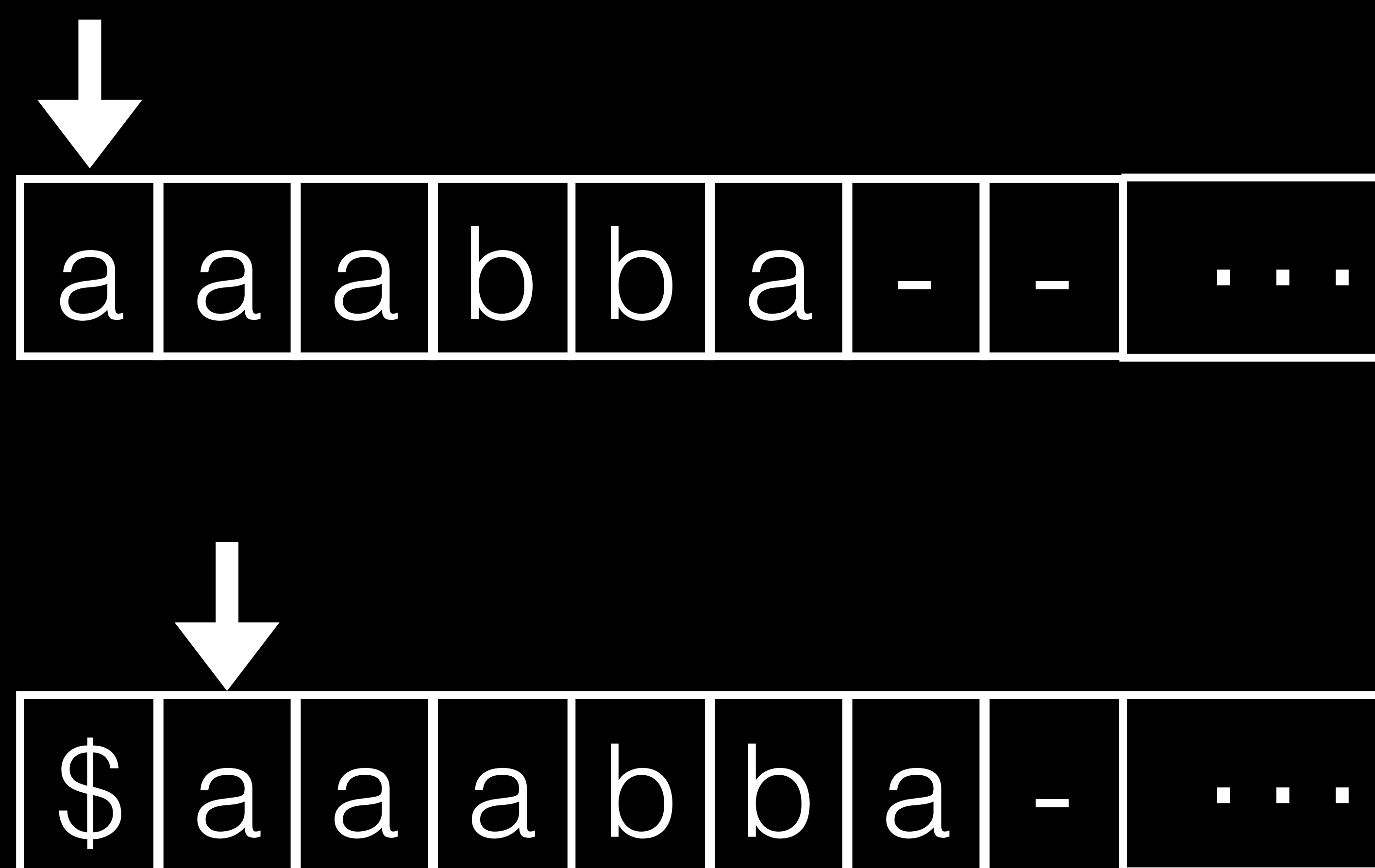
How can we recognize the left end of the tape?

GOAL: Want to put a special symbol $ on the left end and shift the input over 1 cell to the right.

COMMON PROBLEM

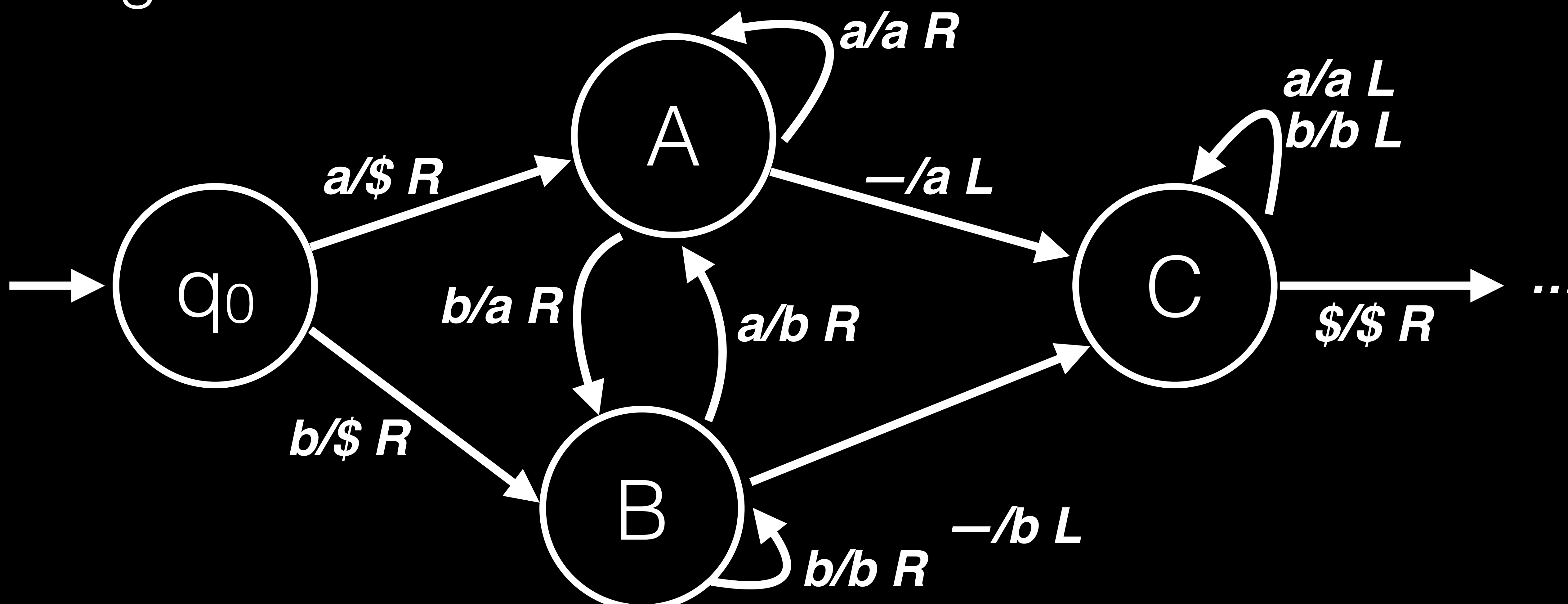
How can we recognize the left end of the tape?

GOAL: Want to put a special symbol $ on the left end and shift the input over 1 cell to the right.



How can we recognize the left end of the tape?

GOAL: Want to put a special symbol $ on the left end and shift the input over 1 cell to the right.



Q: How much TM programming do you need to do?

A: Just enough to get the idea and to convince yourself that all programs or algorithms can be implemented on a Turing Machine.

WRITING PROGRAMS

Machine Code
0110, 1100

↓

Assembly Code
ADD R1, R2, R3

↓

C Code
i = (2 + k) * n

↓

Algorithms
if $S \cap T = \emptyset$…
No implementation details

WRITING PROGRAMS IN TURING MACHINE

Machine Code
0110, 1100

↓

Assembly Code
ADD R1, R2, R3

↓

C Code
i = (2 + k) * n

↓

Algorithms
if $S \cap T = \varnothing$…
No implementation details

Turing Machine
States, Transition Function
(complete TM)

↓

Outline of Algorithm
Still talking about
tape head movement
Data representation

↓

High-level specification of algorithm
No TM specific details

Build a TM to recognize the language $0^n1^n0^n$.

Build a TM to recognize the language $0^n1^n0^n$.

Build a TM to **decide** the language.

Build a TM to recognize the language $0^n1^n0^n$.

This language is **not** context-free. So this will prove

CONTEXT-FREE LANGUAGES $\subset$ DECIDABLE LANGUAGES

proper subset

Build a TM to recognize the language $0^n1^n0^n$.

We already have a TM to turn 0^n1^n into X^nY^n and to decide that language.

PROGRAMMING TECHNIQUE: SUBROUTINE

Build a TM to recognize the language $0^n1^n0^n$.

We already have a TM to turn 0^n1^n into X^nY^n and to decide that language.

Build a TM to recognize the language $0^n1^n0^n$.

We already have a TM to turn 0^n1^n into X^nY^n and to decide that language.

Idea: Use that TM as a subroutine!

Idea: Use that TM as a subroutine!

Step 1:

0 0 0 0 1 1 1 1 0 0 0 0

X X X X Y Y Y Y 0 0 0 0

Step 2:

Build a similar TM to recognize Y^n0^n

Step 3:

Build the final TM by “gluing” these smaller TMs together into one larger TM.

Idea: Use that TM as a subroutine!

Step 1:

0 0 0 0 1 1 1 1 0 0 0 0

X X X X Y Y Y Y 0 0 0 0

Step 2:

Build a similar TM to recognize $Y^n 0^n$

Step 3:

Build the final TM by “gluing” these smaller TMs together into one larger TM.

PROGRAMMING TECHNIQUE: MARKING SYMBOL

Compare two strings. A TM to decide $\{w\#w \mid w \in \{a, b, c\}\}$.

Compare two strings. A TM to decide $\{w\#w \mid w \in \{a, b, c\}$.

Solution:

- Use a new symbol, such as "x"
- Turn each symbol into an "x" after it has been examined.

PROGRAMMING TECHNIQUE: MARKING SYMBOL

Compare two strings. A TM to decide $\{w\#w \mid w \in \{a, b, c\}\}$.

a a b a c # a a b a c

x a b a c # x a b a c

x x b a c # x x b a c

....

x x x x x # x x x x x

Compare two strings. A TM to decide $\{w\#w \mid w \in \{a, b, c\}\}$.

Problem:

Do it nondestructively, without losing the original strings. (Perhaps this task is part of a larger task.)

Compare two strings. A TM to decide $\{w\#w \mid w \in \{a, b, c\}\}$.

Problem:

Do it nondestructively, without losing the original strings. (Perhaps this task is part of a larger task.)

a = x, b = y, c = z

Compare two strings. A TM to decide $\{w\#w \mid w \in \{a, b, c\}$.

Solution:

Mark each symbol to keep track of what we've already done. Add some new symbols to help.

$a = x$, $b = y$, $c = z$

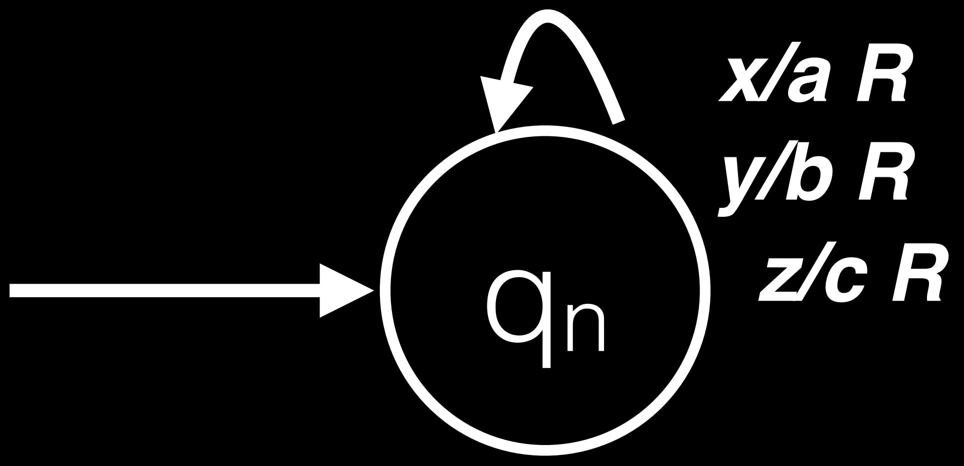
Compare two strings. A TM to decide $\{w\#w \mid w \in \{a, b, c\}$.

a a b a c # a a b a c

...

x x y x z # x x y x z

Later, restore the strings if we need to



PROGRAMMING TECHNIQUE: MARKING SYMBOL

“Mark each symbol with a dot.”

“Remember this location.”

Multitape Turing Machines

THEOREM

Every multitape Turing Machine has an equivalent single-tape Turing Machine.

Equivalent means it decides/recognizes the same languages. It’s not about speed, efficiency or ease of programming.

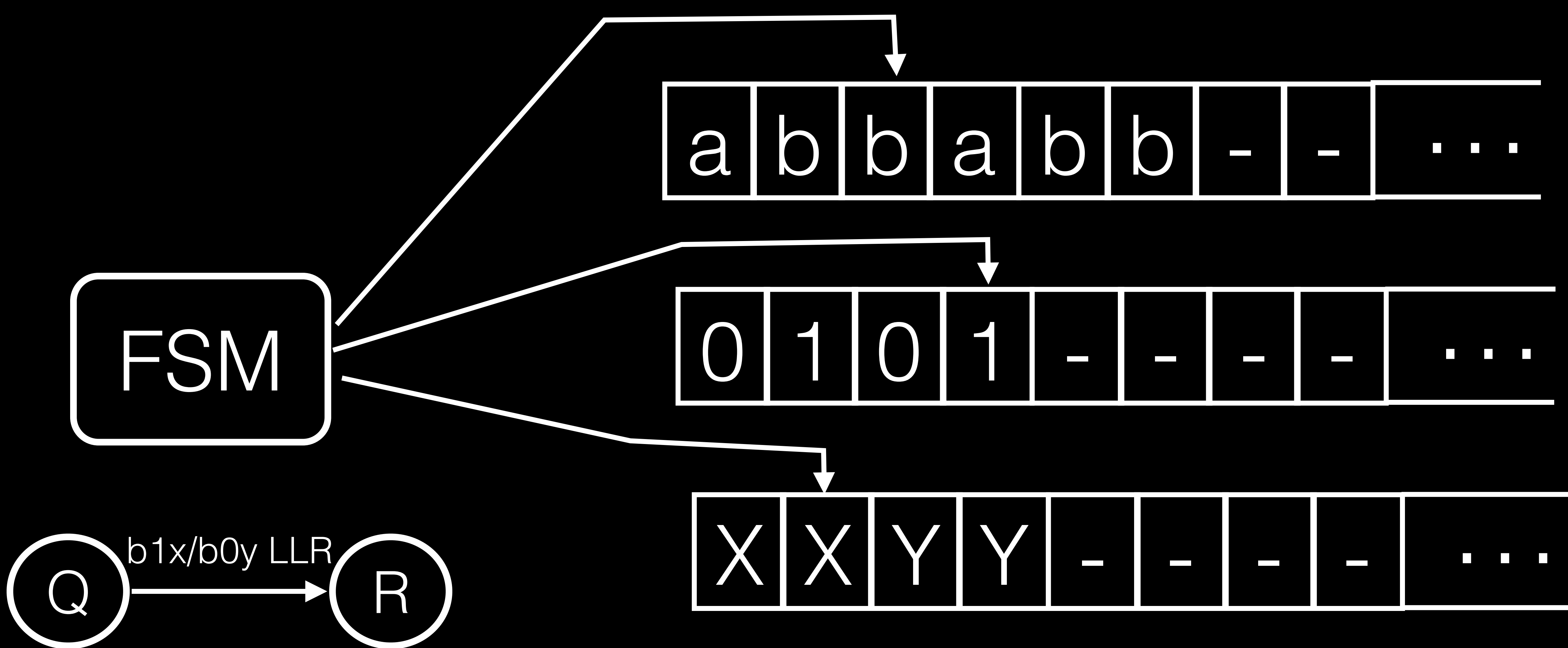
PROOF

Given a multitape TM, show how to build a single-tape TM.

- Need to store all tapes on a single tape. (Show data representation.)
- Each tape has a tape head. (Show how to store that information.)
- Need to transform a move in the multitape TM into a one or more in the single-tape TM.

PROOF

Multitape TM

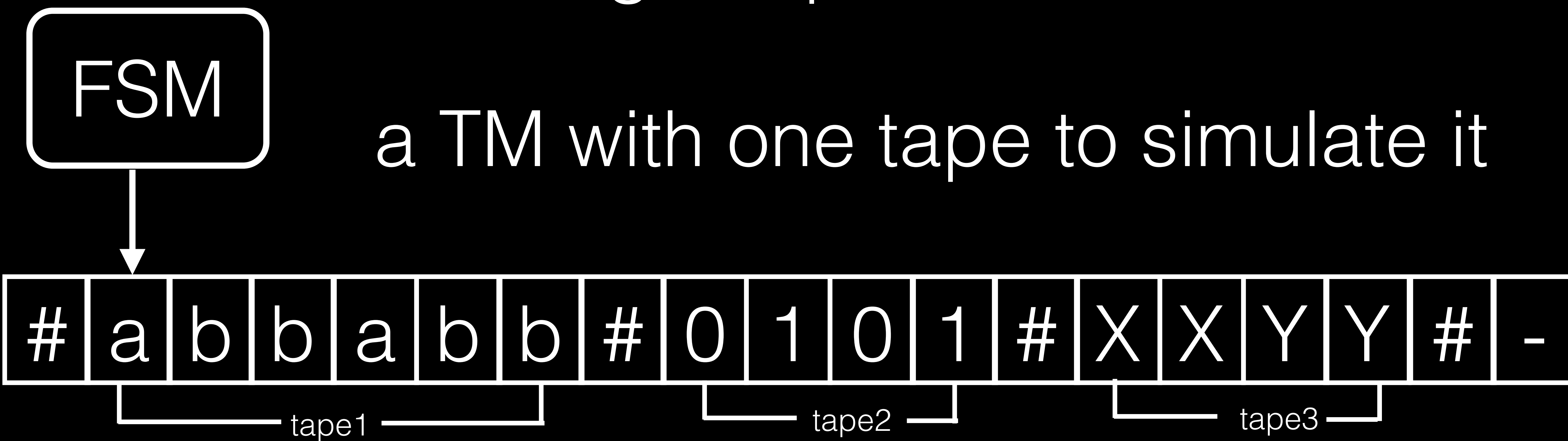


An example machine with k = 3 tapes

PROOF

Single-tape TM

a TM with one tape to simulate it



- Add “dots” to show where head k is.
- To simulate a transition from state Q, we must scan our tape to see which symbols are under the k tape heads.

Nondeterministic Turing Machines

Nondeterminism means that the TM may have more than one choice of action. As usual, a nondeterministic TM (or NTM) accepts a string if some choice of actions lead to that accept state.

Theorem. A nondeterministic TM has the same power as a standard TM.

THEOREM: TM vs NTM

Proof: We show that the NTM can be simulated by a deterministic one.

We need the concept of configuration. We view the calculations of NTM as a tree. The nodes are the configurations of the NTM, and the children of a node are the possible next steps. The NTM accepts the input if there is a branch that leads to an accepting configuration.

The simulator does breadth-first-search of tree.

Turing Machine as Problem Solvers