Department of Computer Science

THEORY OF COMPUTATION

23MT2014

Topic:

COMPLEXITY CLASSES: P & NP

Session - 24

AIM OF THE SESSION

To make students explain the fundamental concepts of complexity classes P and NP, their significance in computational theory, and the implications of problems classified within these classes.

INSTRUCTIONAL OBJECTIVES

By the end of this course, students will be able to:

1. Explain the definitions of the complexity classes P and NP.
2. Illustrate examples of problems in P and NP, highlighting the differences between them.
3. Discuss the importance of the P vs NP question and its implications for computer science and other fields.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define and distinguish between the complexity classes P and NP.
2. Identify and categorize problems as belonging to P, NP, or NP-complete.
3. Evaluate the significance of the P vs NP problem and its potential impact on computational theory and practical applications.

CLASSIFYING PROBLEMS BY TIME COMPLEXITY

- **Efficiency Analysis:** **How does complexity change with input size?**
- **Resource Management:** **What resources are needed?**
- **Comparative Benchmarking:** **Which algorithm performs best?**
- **Scalability Prediction :** **How does performance change with size?**
- **Theoretical Foundation:** **What is the problem's complexity class?**
- **Optimization guidance:** **Where are the bottlenecks?**

CLASS OF LANGUAGES

- Let $t: \mathbb{N} \rightarrow \mathbb{R}^+$ be a function. Define the *time complexity class*, $\boldsymbol{TIME(t(n))}$, to be the collection of all languages that are decided by an $O(t(n))$ time Turing Machine.

POLYNOMIAL TIME

- The time taken by an algorithm (t) increases proportionally to the input size (n) raised to a power, k.
- Ex: n^2, n^3, n^4 etc.
- The time complexity is a polynomial function of the input size.

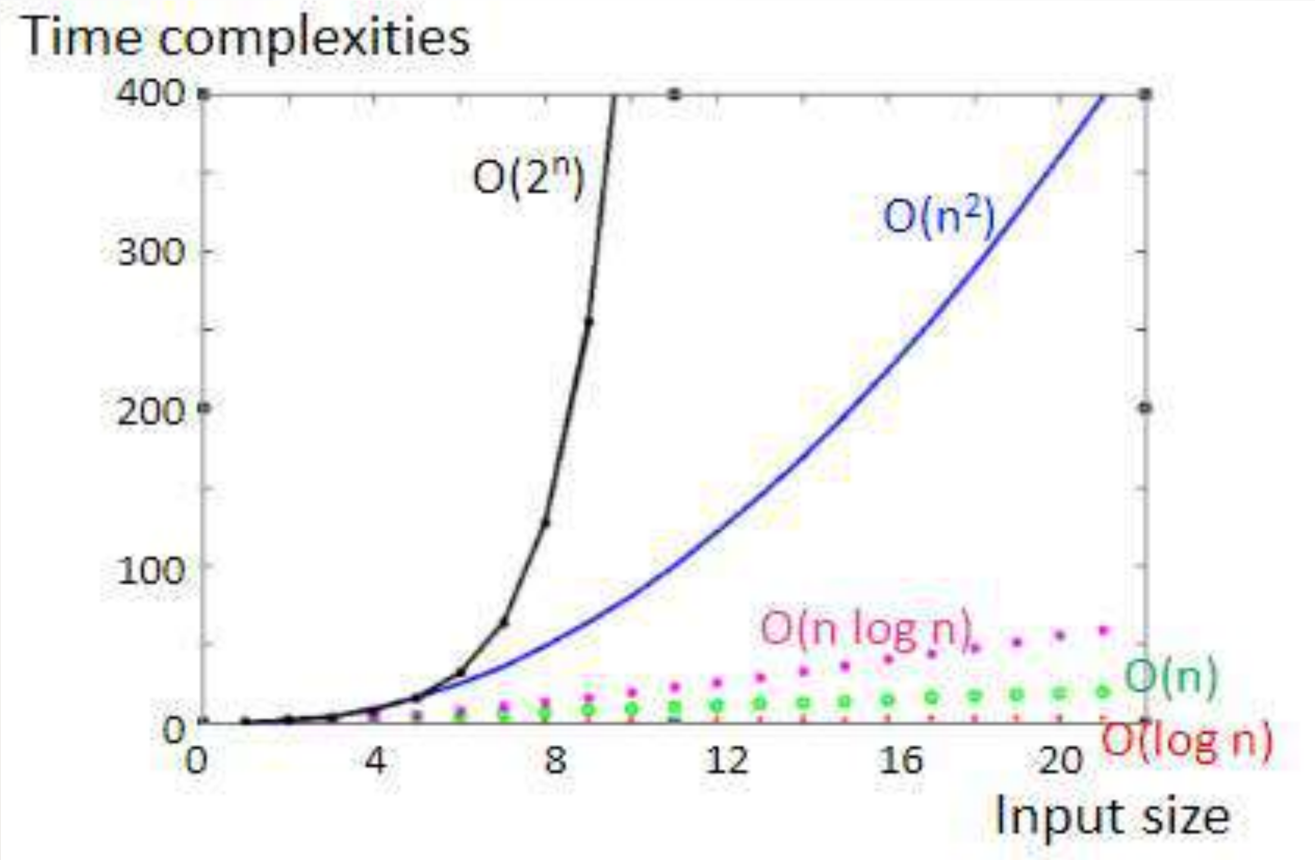
Ex: Quicksort, Merge sort

EXPONENTIAL TIME

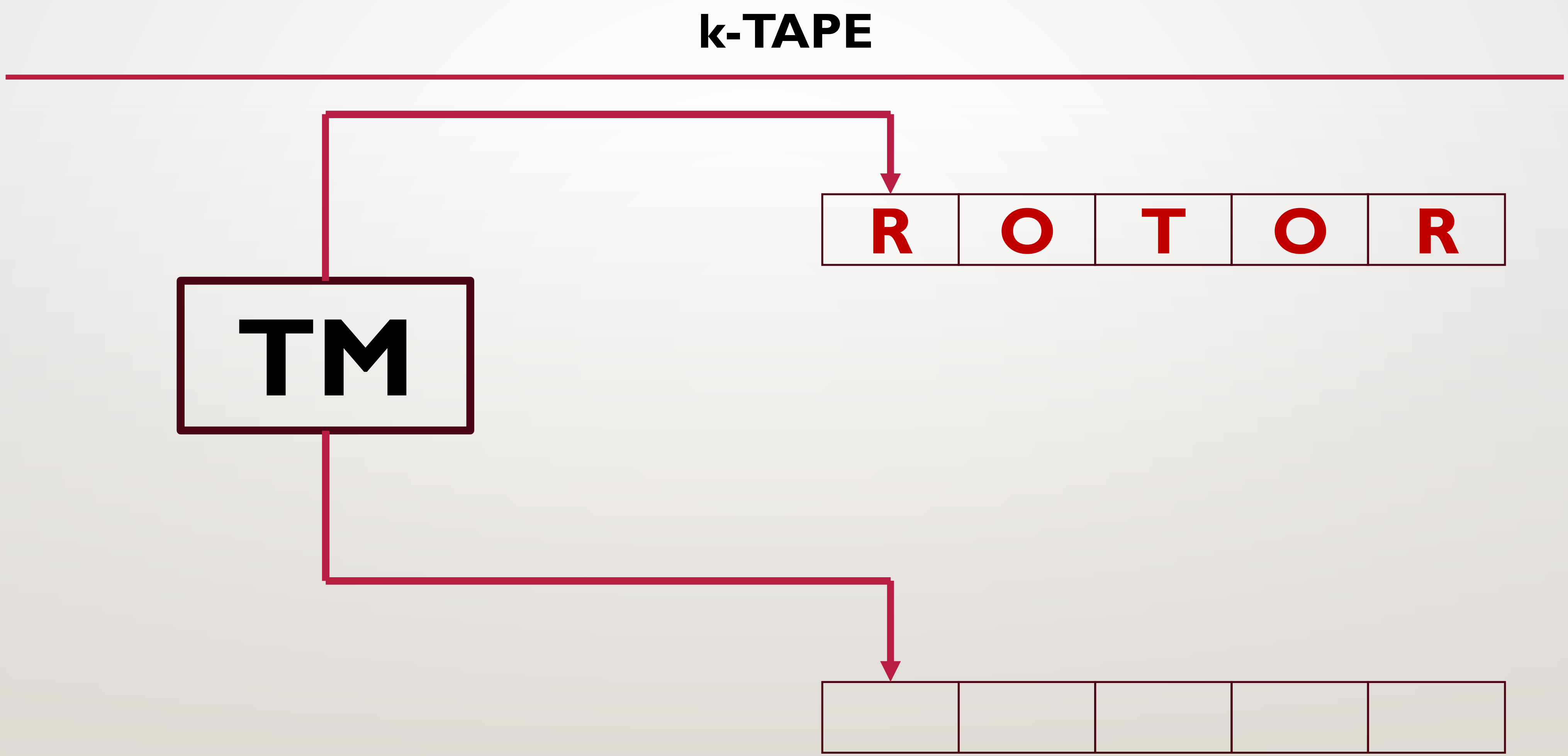
- The time taken by the algorithm (t) increases exponentially with the input size (n).
- Ex: 2^n, 3^n, 5^n etc.
- The time complexity is an exponential function of the input size.
- Example:

1. Brute-force algorithm for solving Travelling salesperson problem
2. Solving the knapsack problem using dynamic programming has a time complexity of $O(2^n)$.

POLYNOMIAL VS. EXPONENTIAL TIME

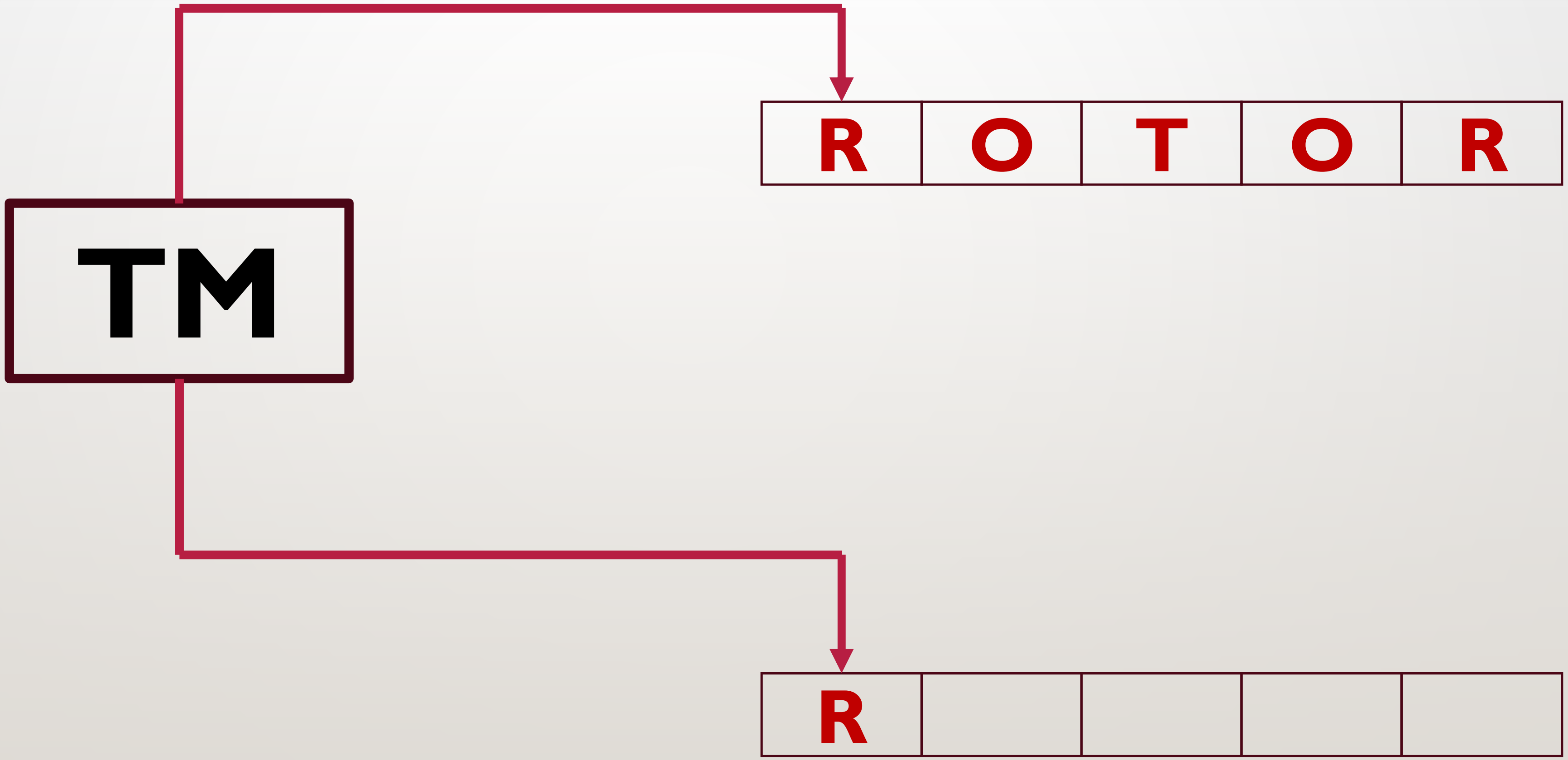


PALINDROME STRING VERIFICATION



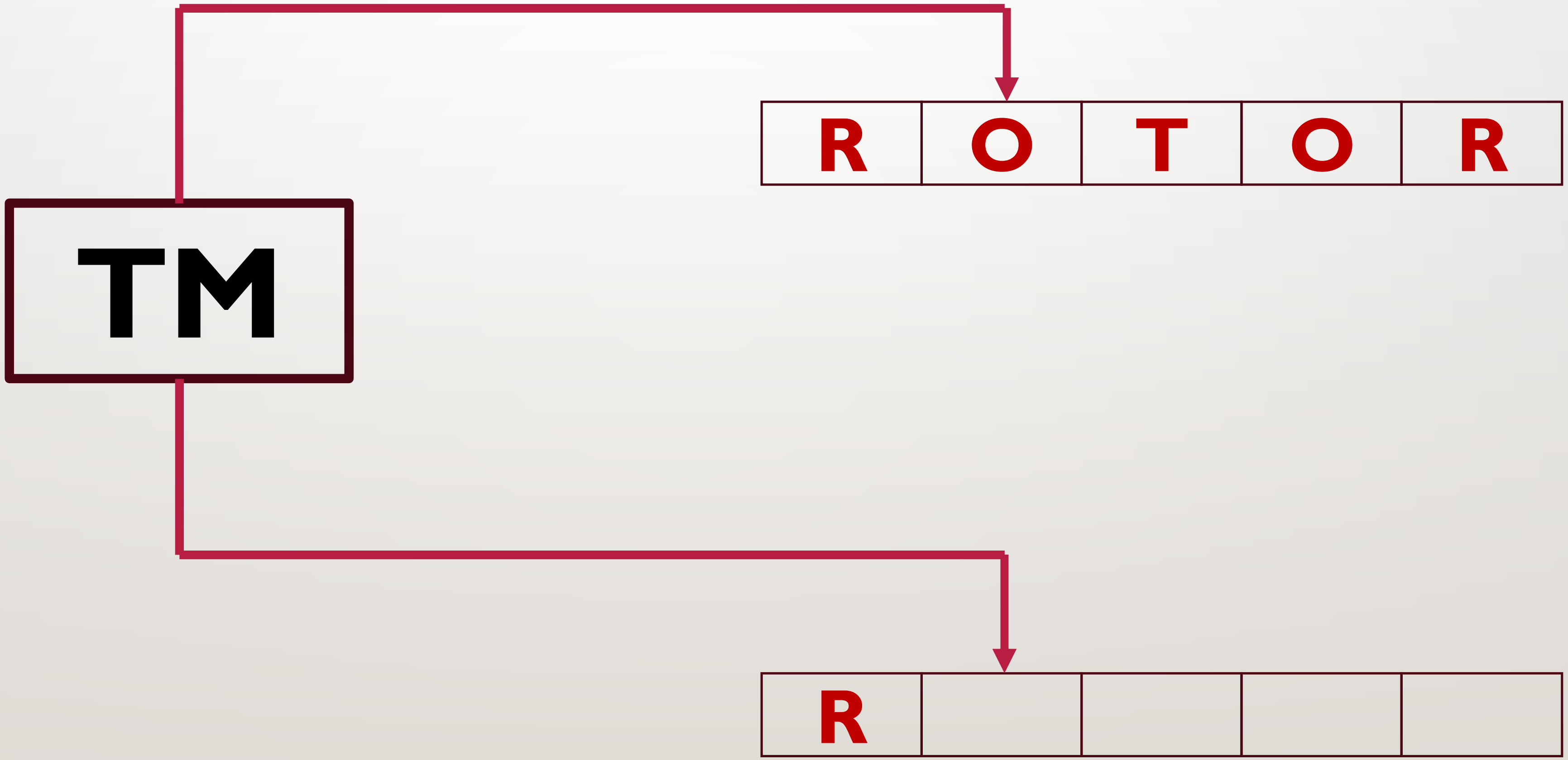
PALINDROME STRING VERIFICATION

k-TAPE



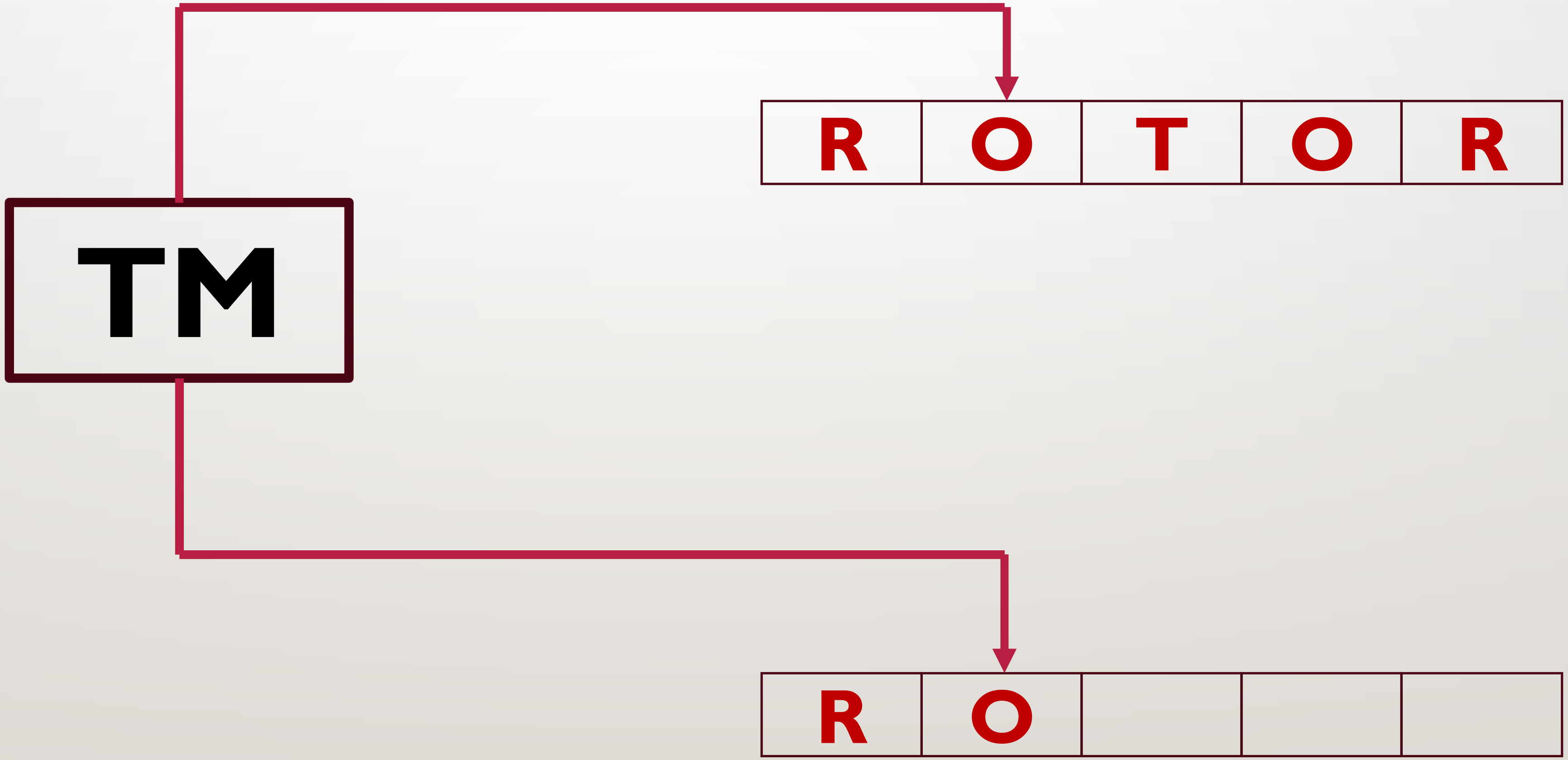
PALINDROME STRING VERIFICATION

k-TAPE



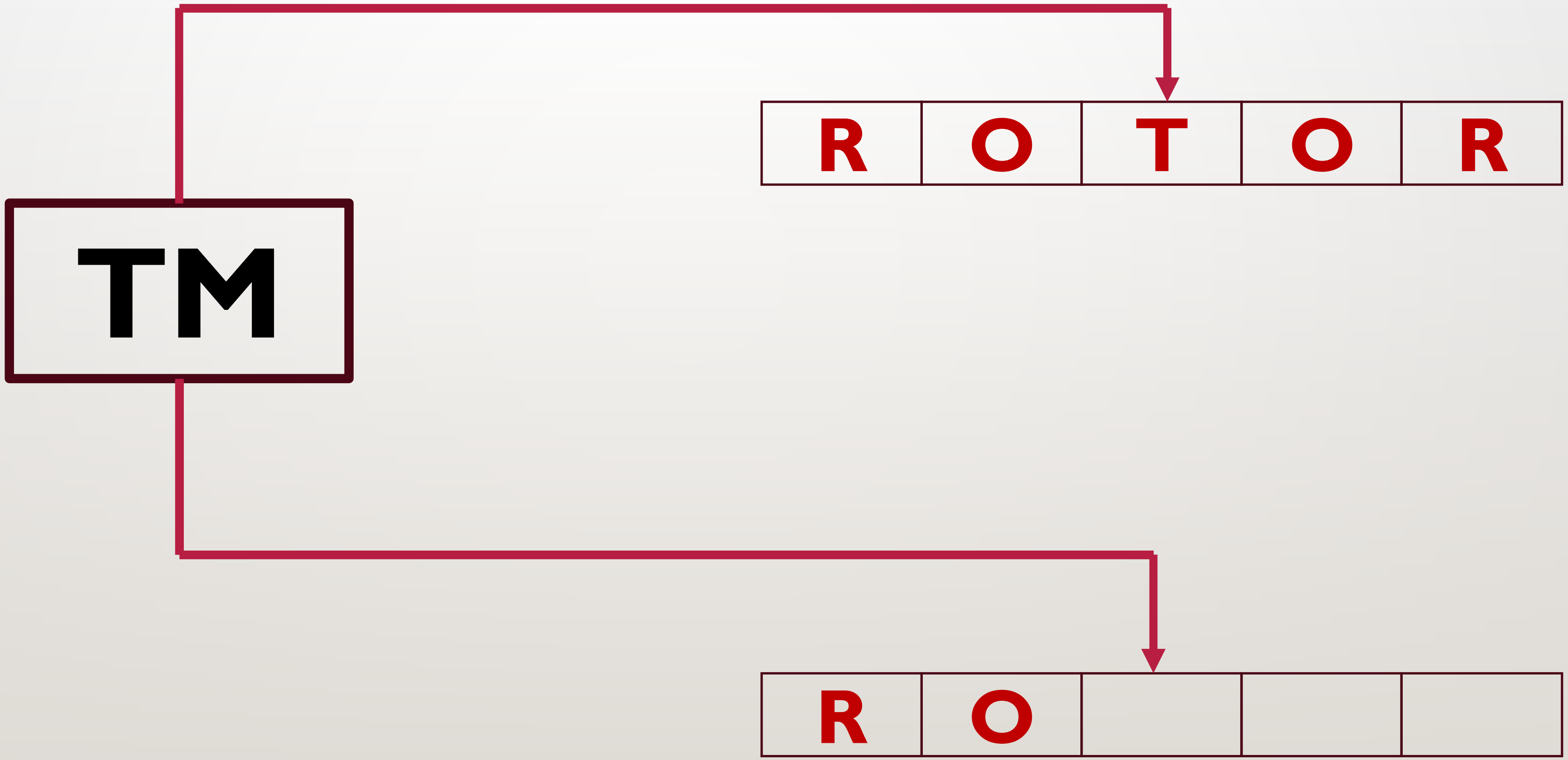
PALINDROME STRING VERIFICATION

k-TAPE



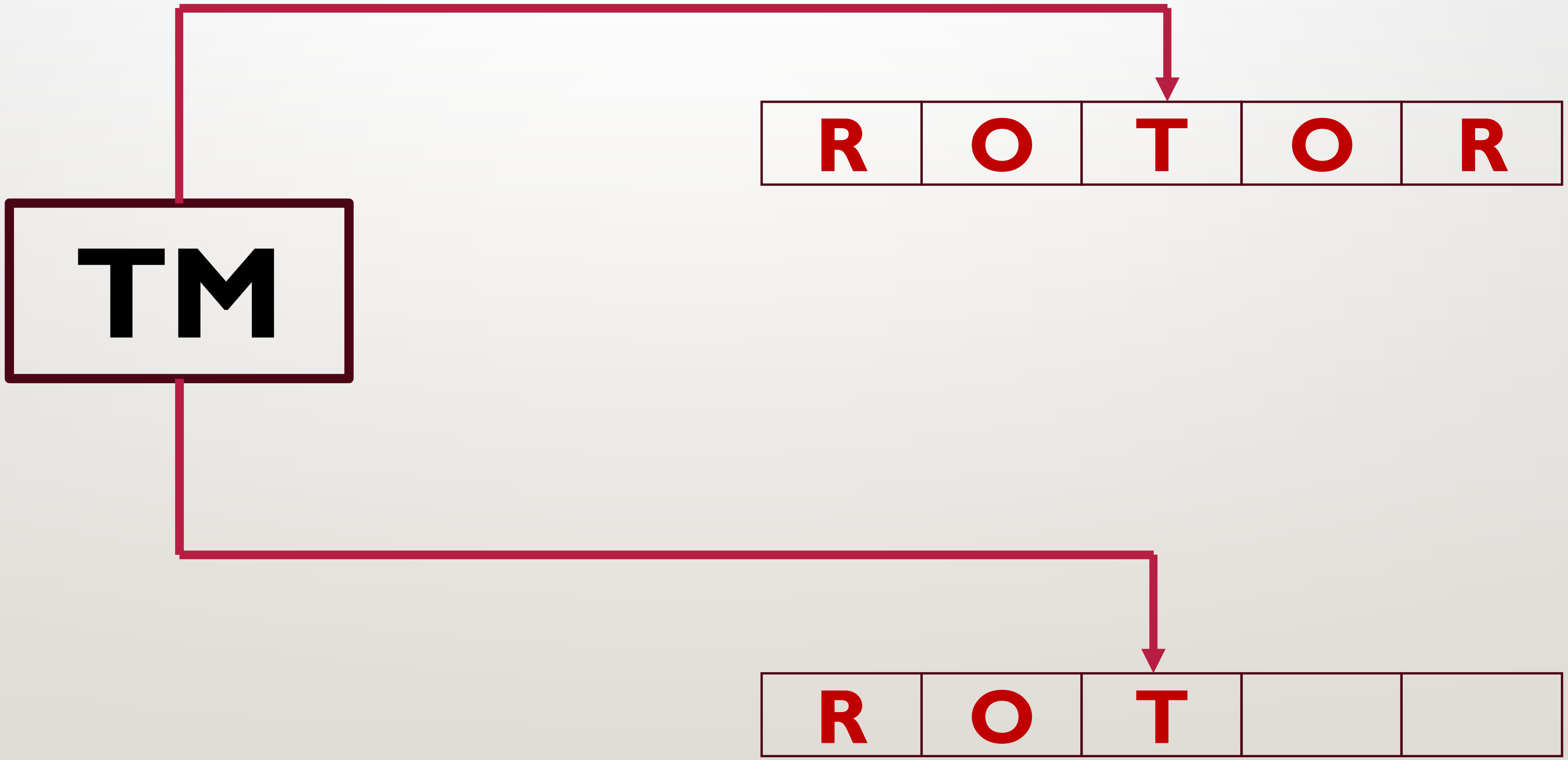
PALINDROME STRING VERIFICATION

k-TAPE



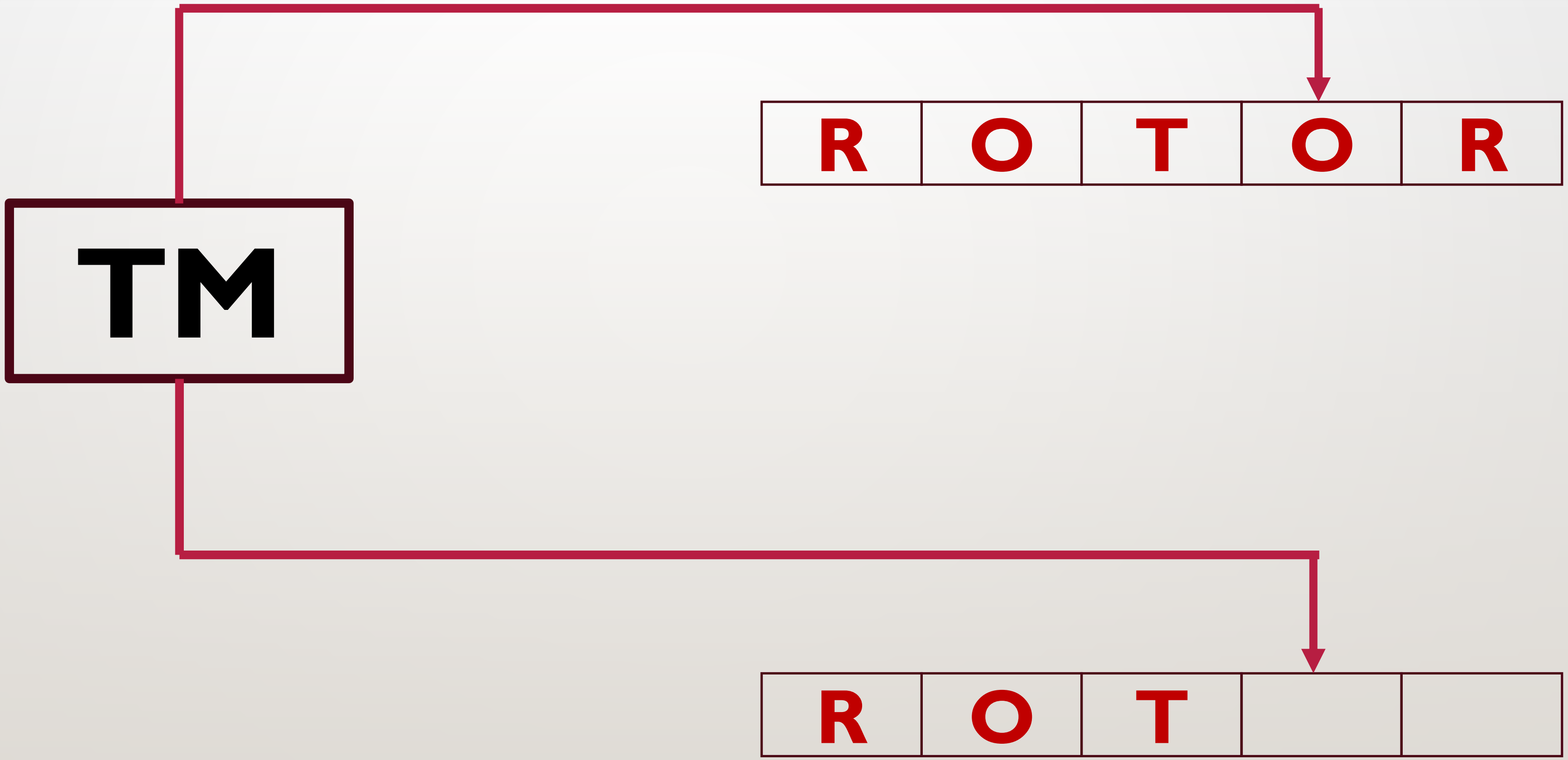
PALINDROME STRING VERIFICATION

k-TAPE



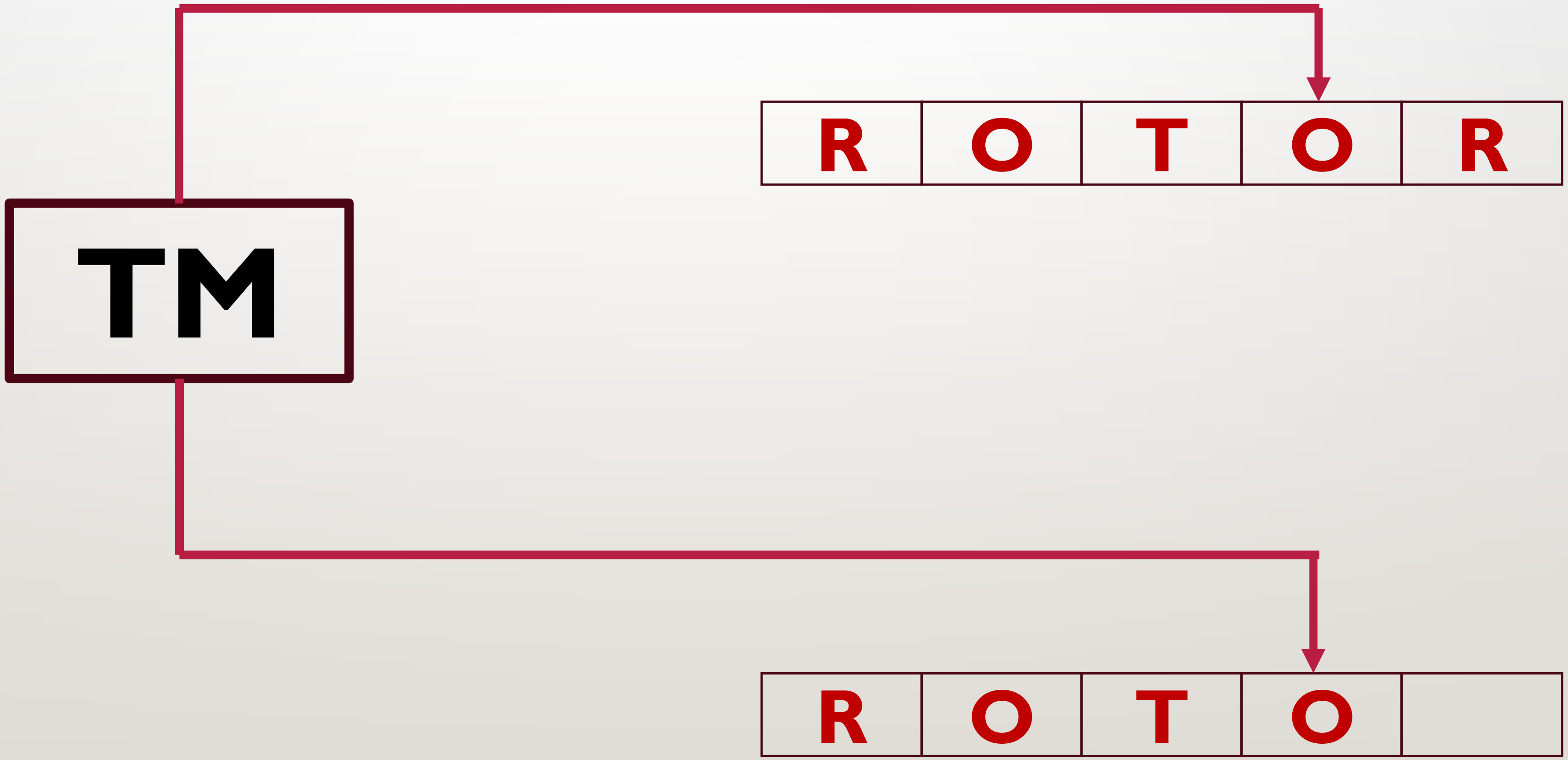
PALINDROME STRING VERIFICATION

k-TAPE



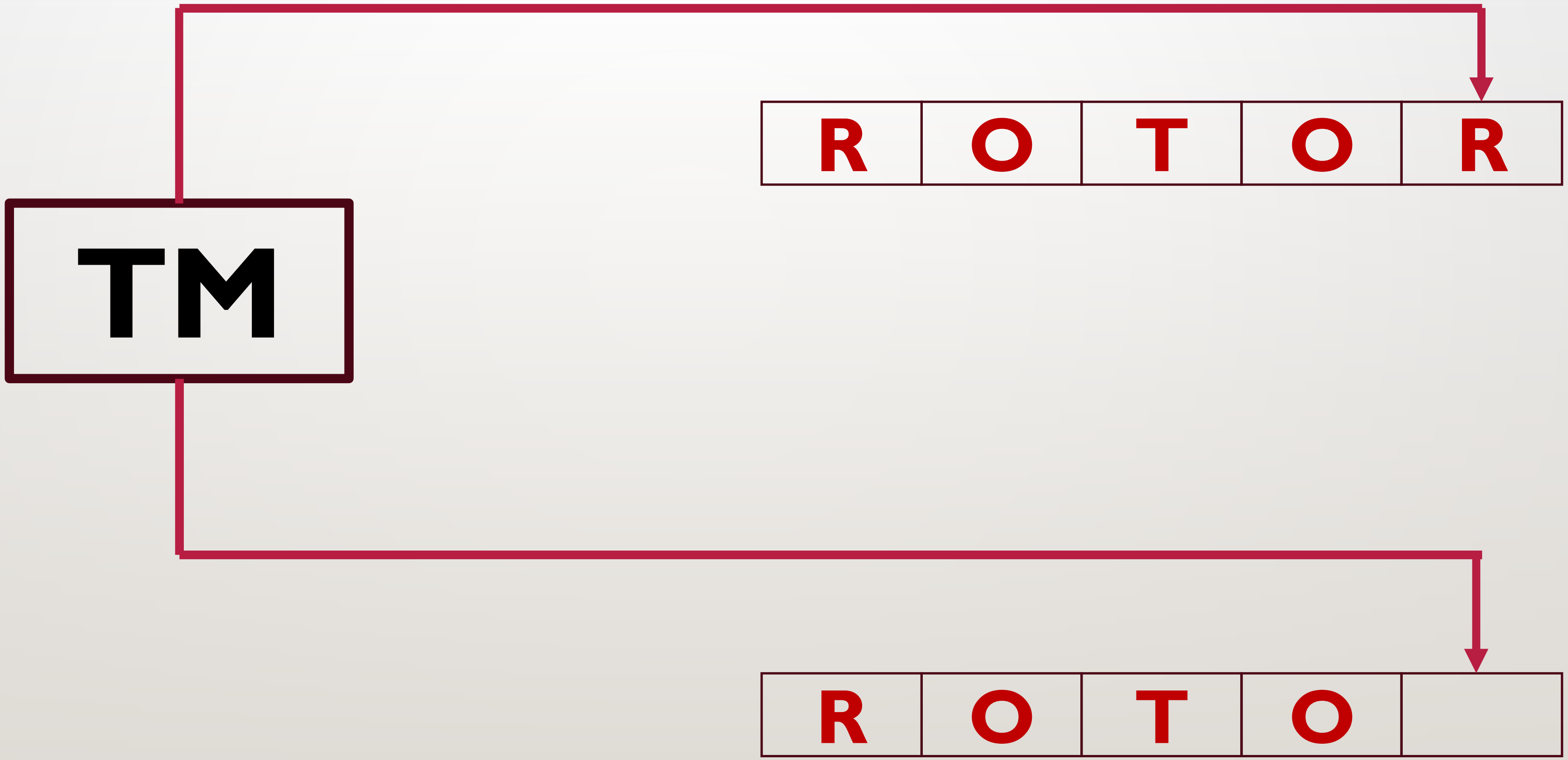
PALINDROME STRING VERIFICATION

k-TAPE

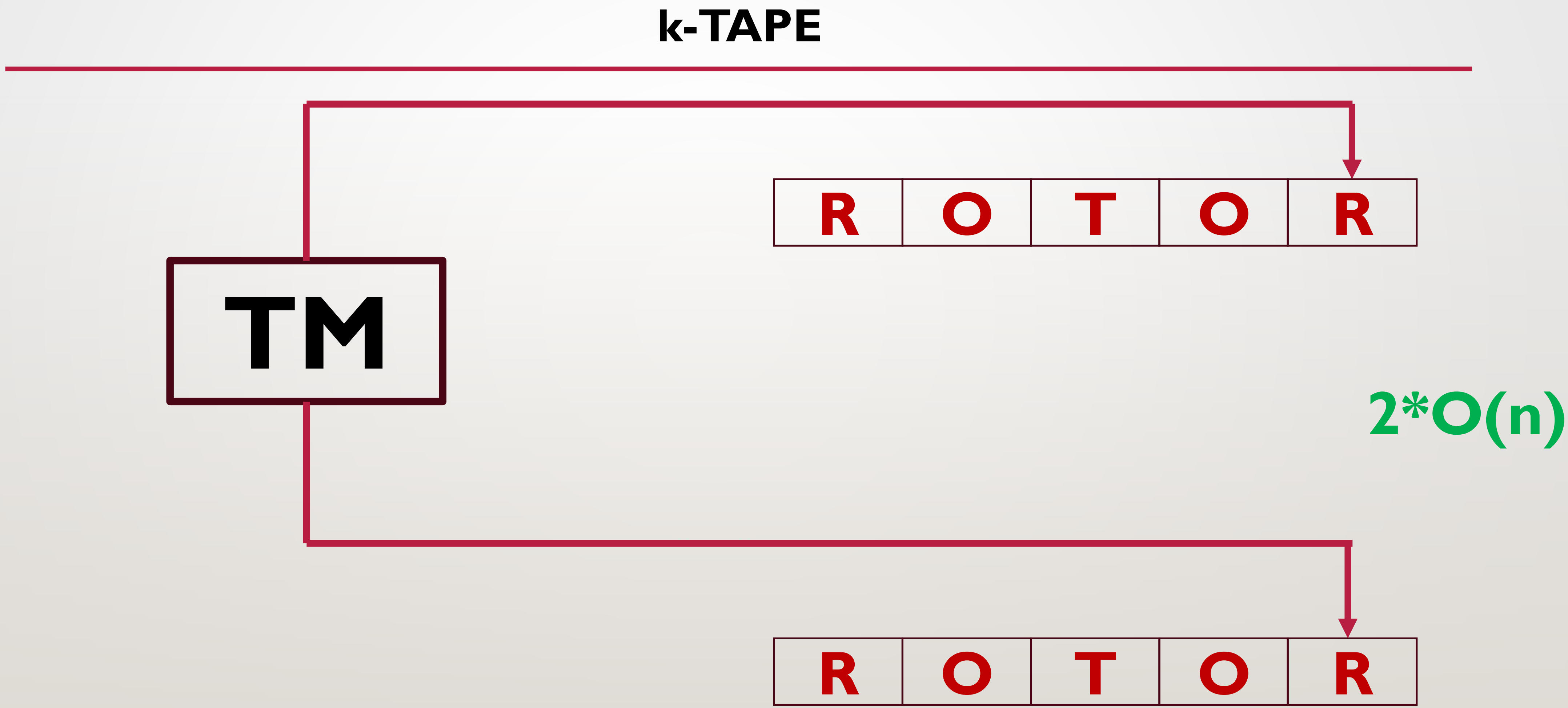


PALINDROME STRING VERIFICATION

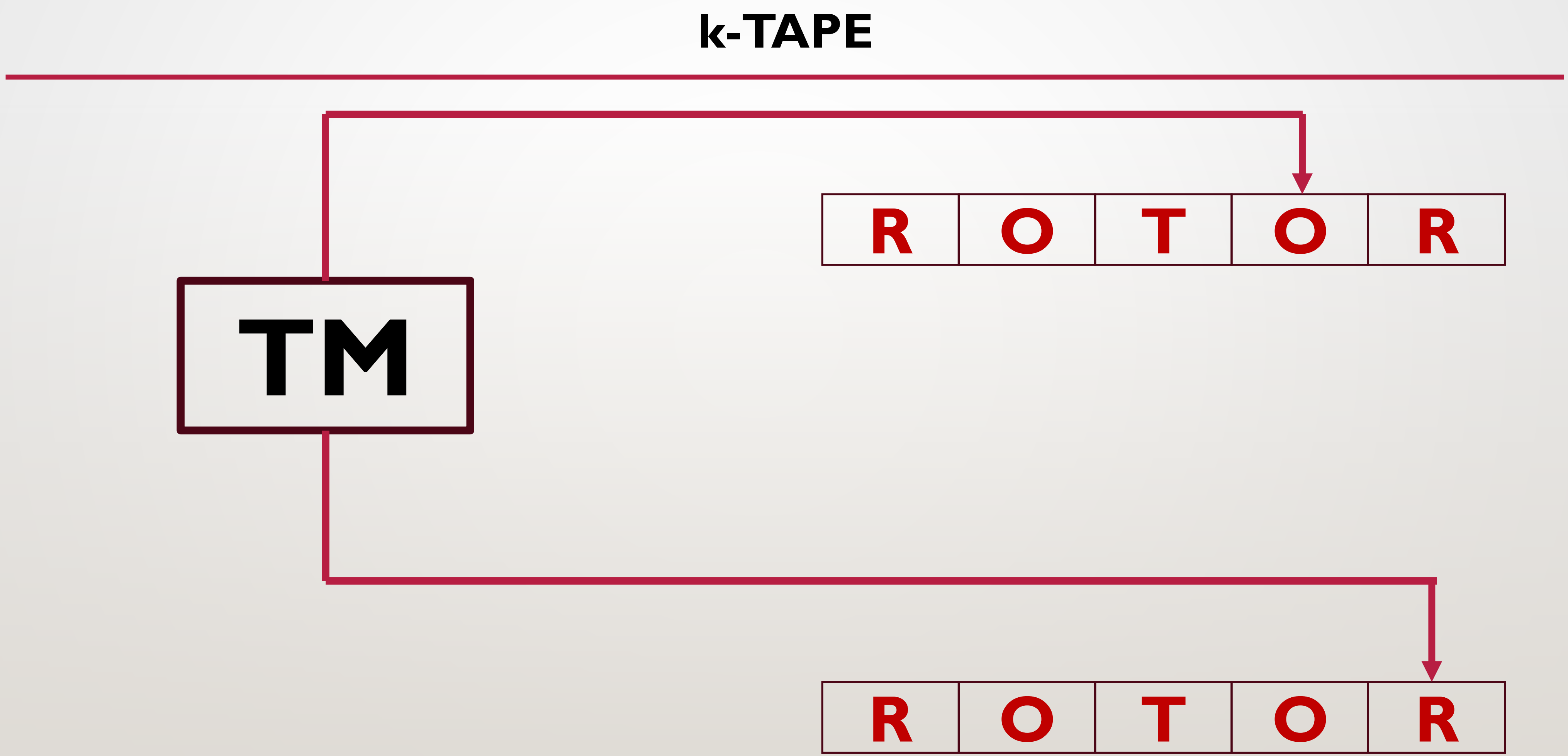
k-TAPE



PALINDROME STRING VERIFICATION

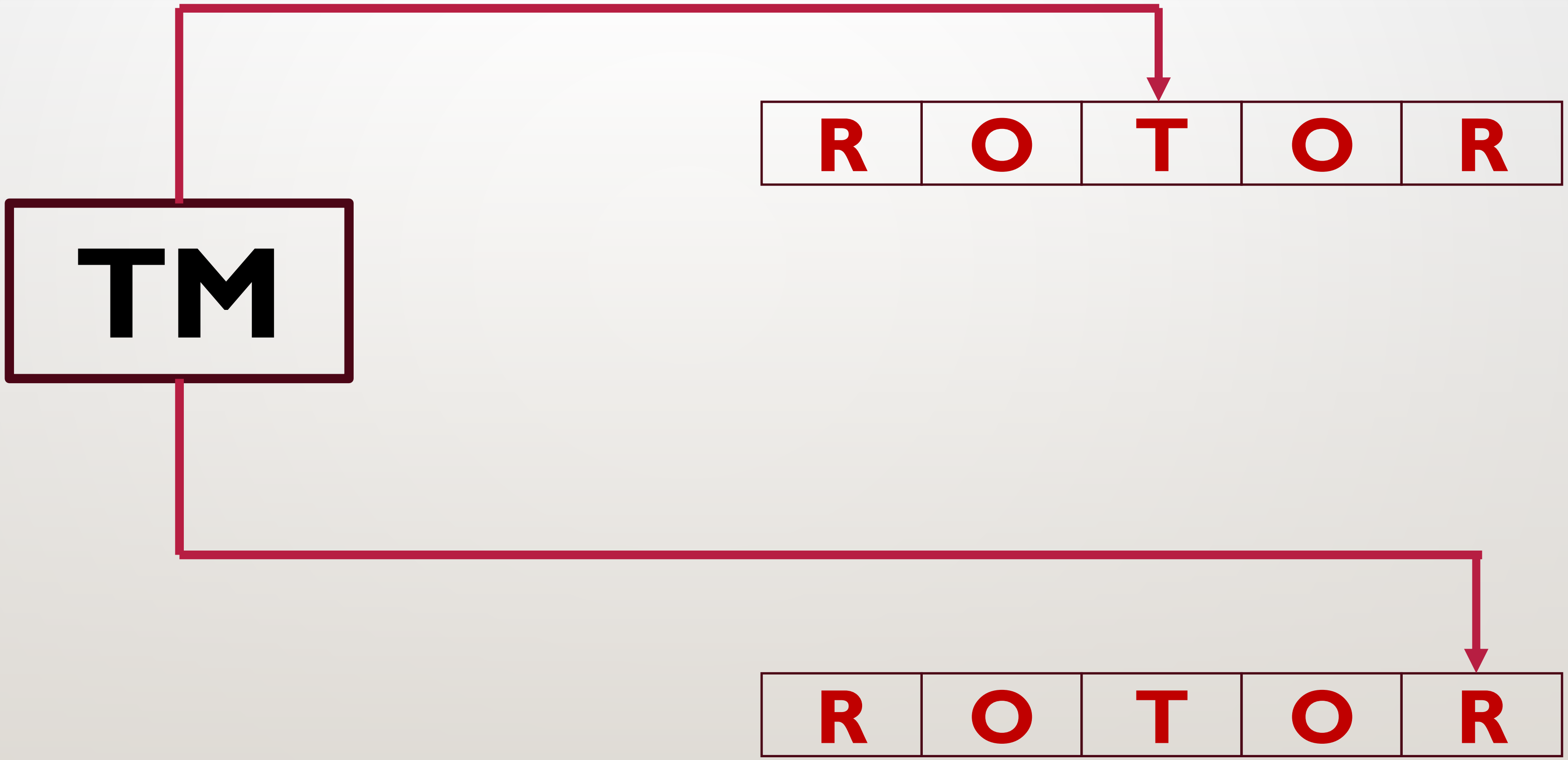


PALINDROME STRING VERIFICATION



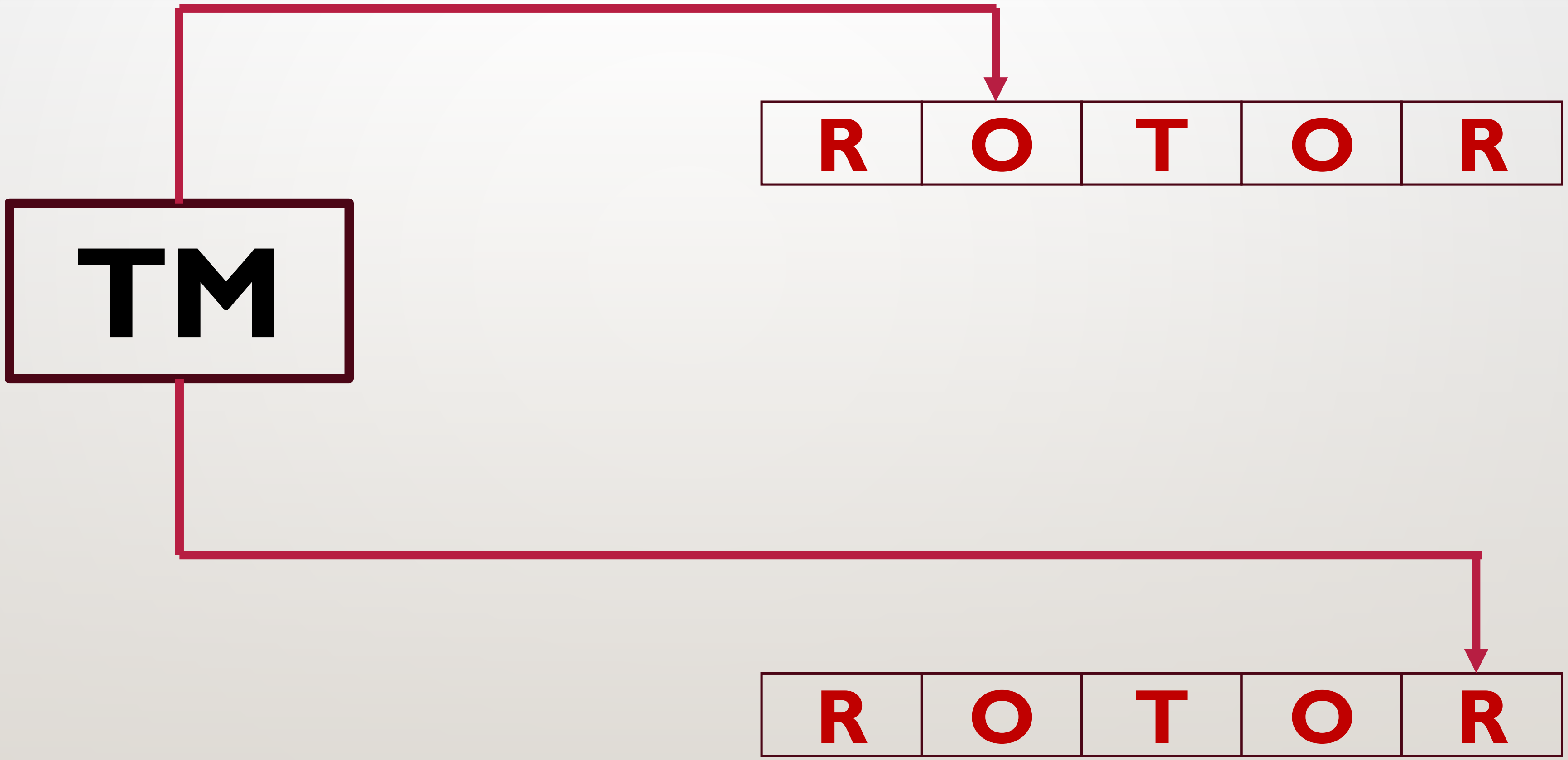
PALINDROME STRING VERIFICATION

k-TAPE



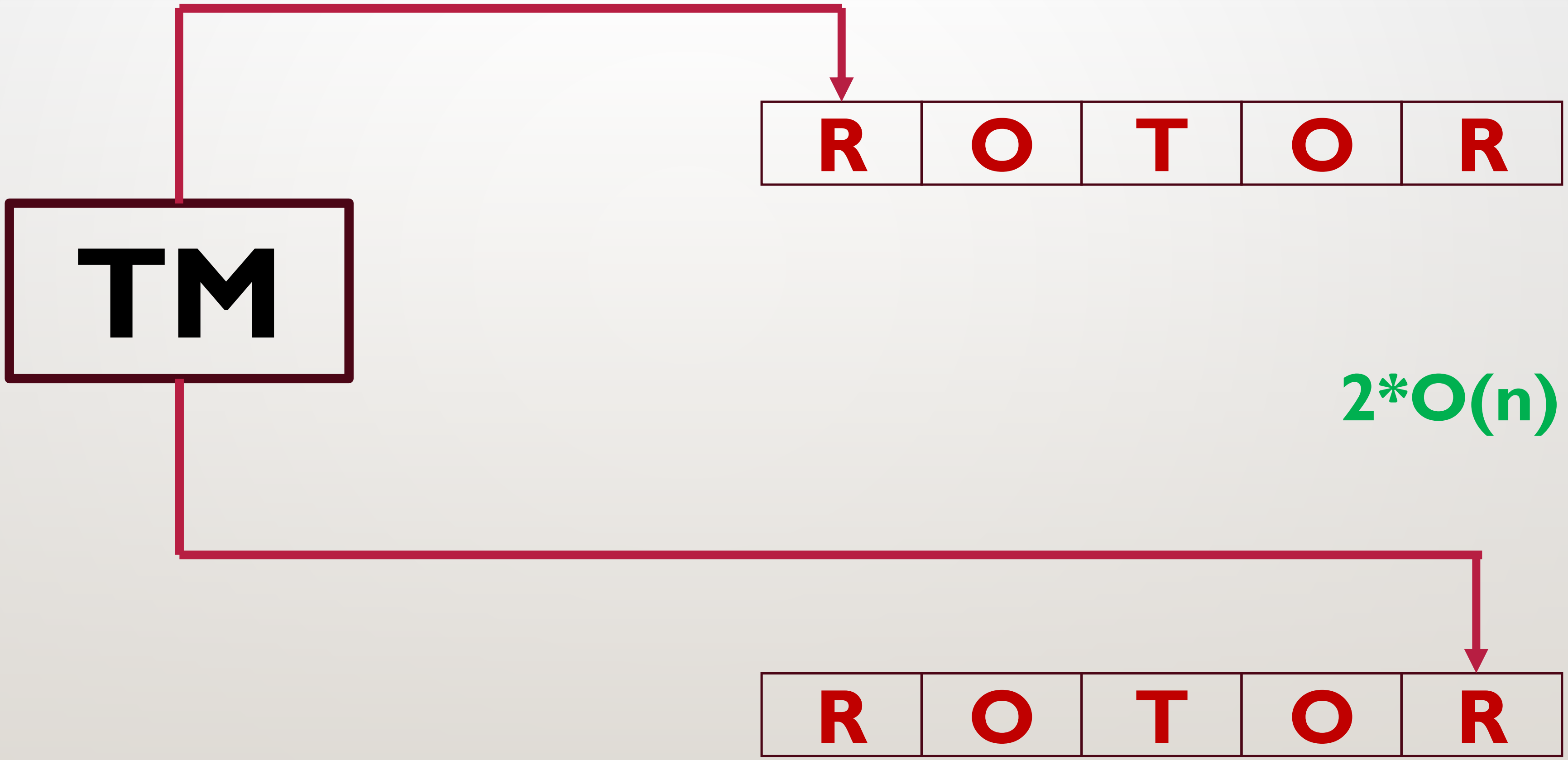
PALINDROME STRING VERIFICATION

k-TAPE



PALINDROME STRING VERIFICATION

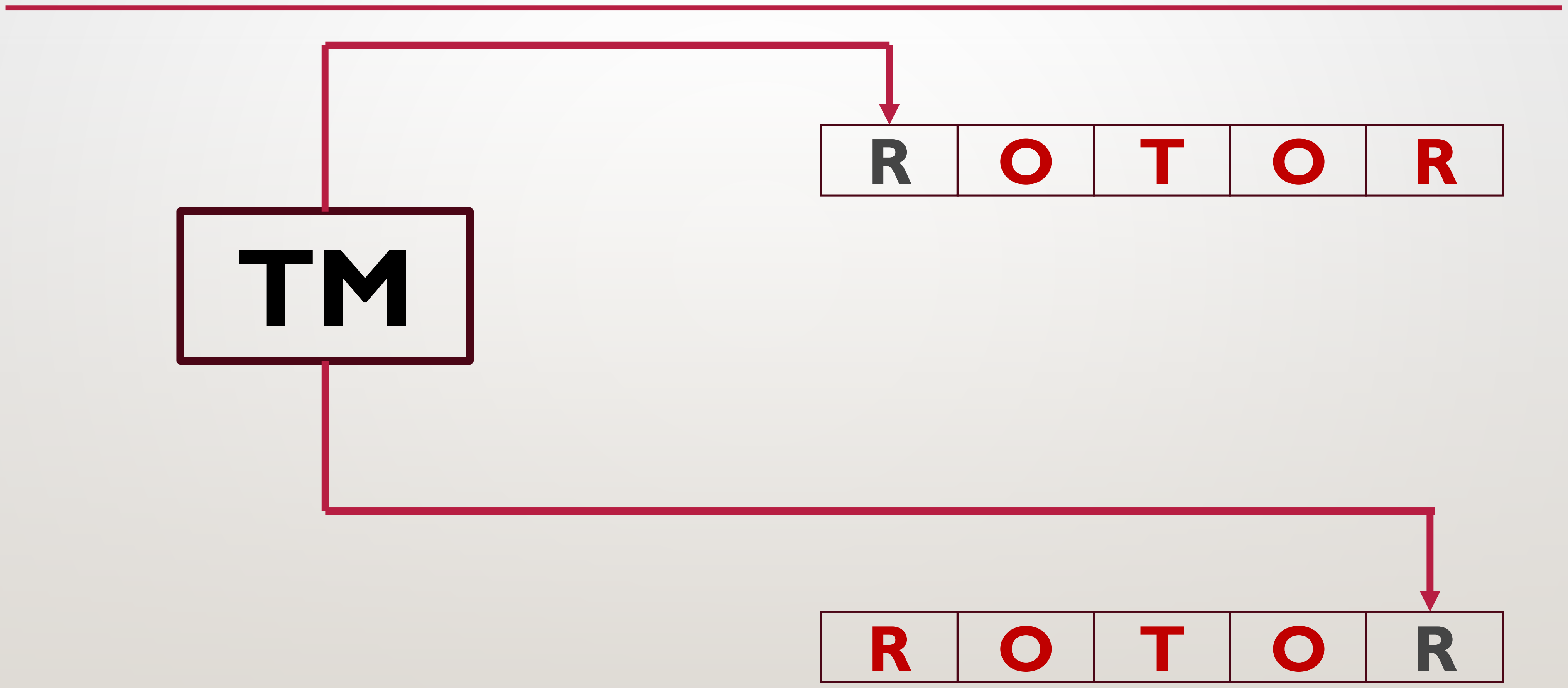
k-TAPE



2*O(n) + O(n)

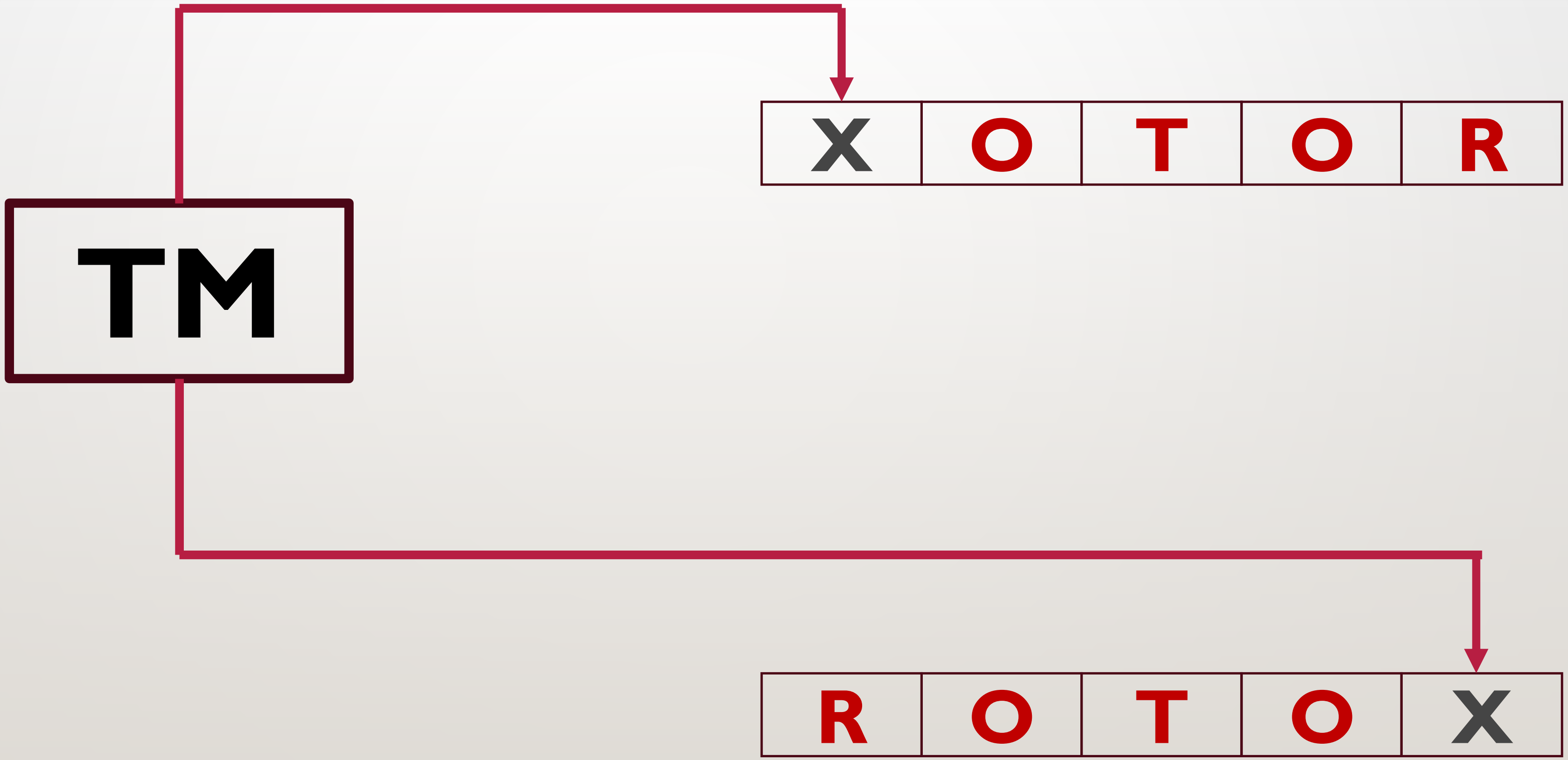
PALINDROME STRING VERIFICATION

k-TAPE



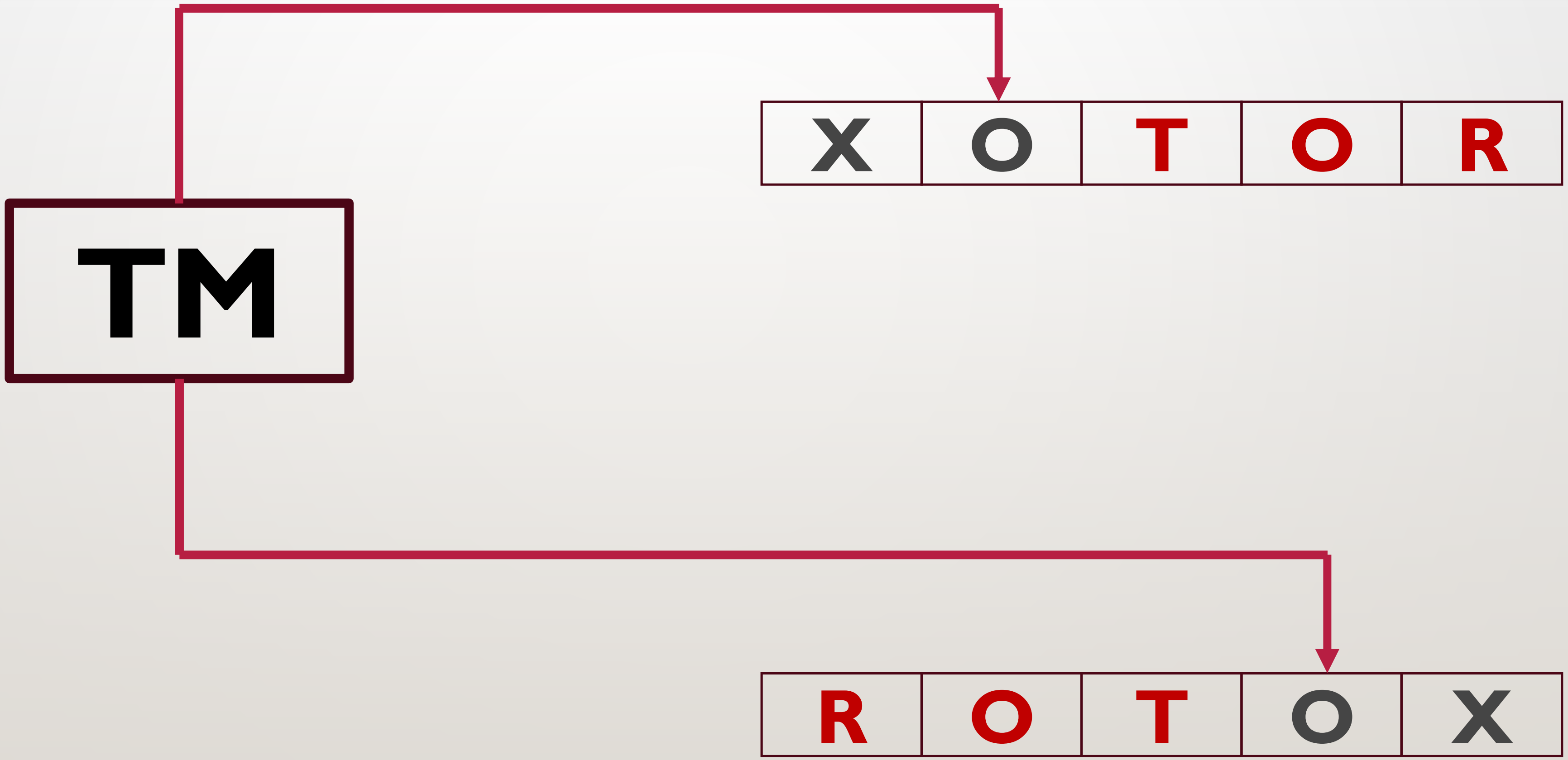
PALINDROME STRING VERIFICATION

k-TAPE



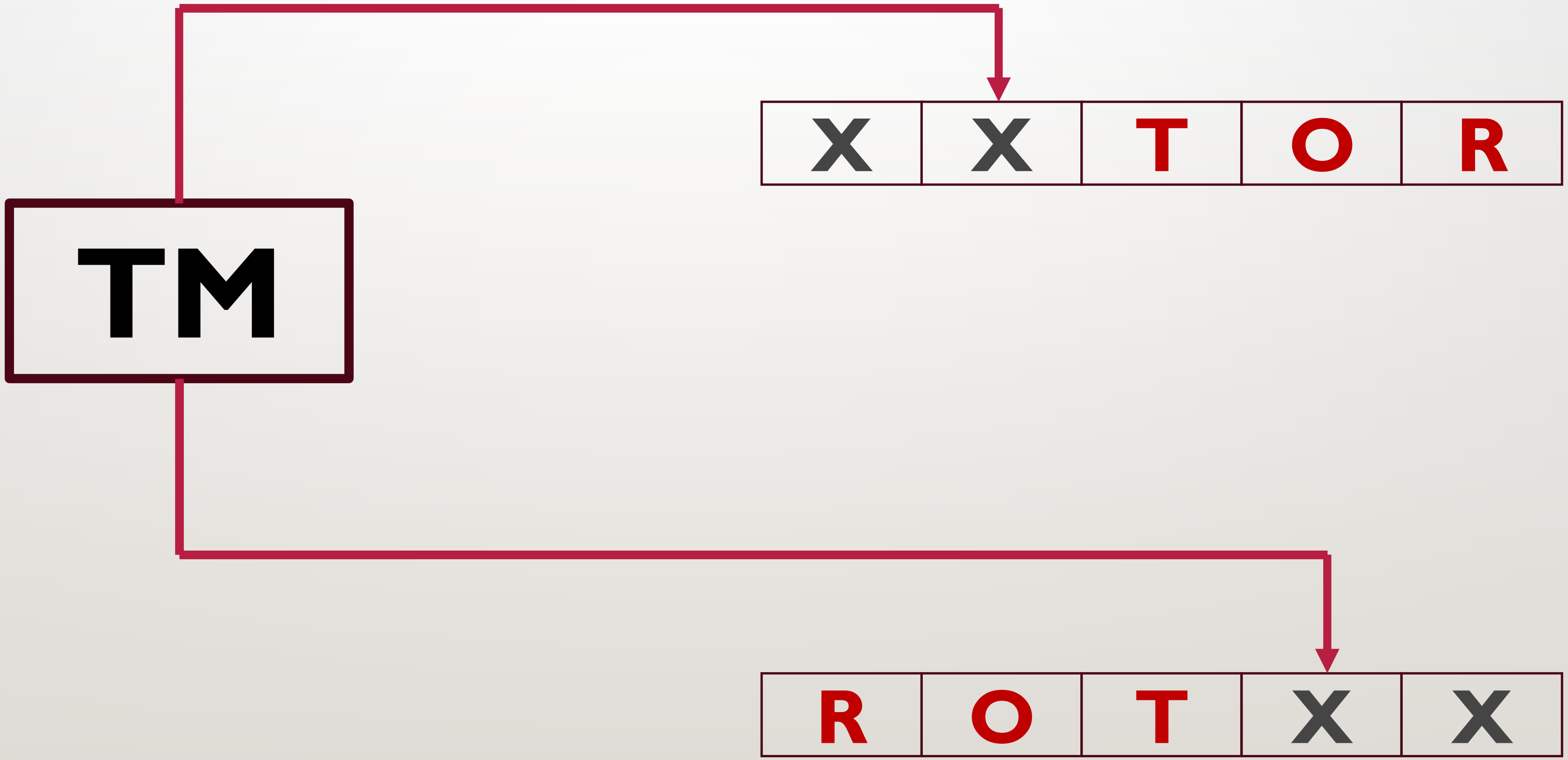
PALINDROME STRING VERIFICATION

k-TAPE



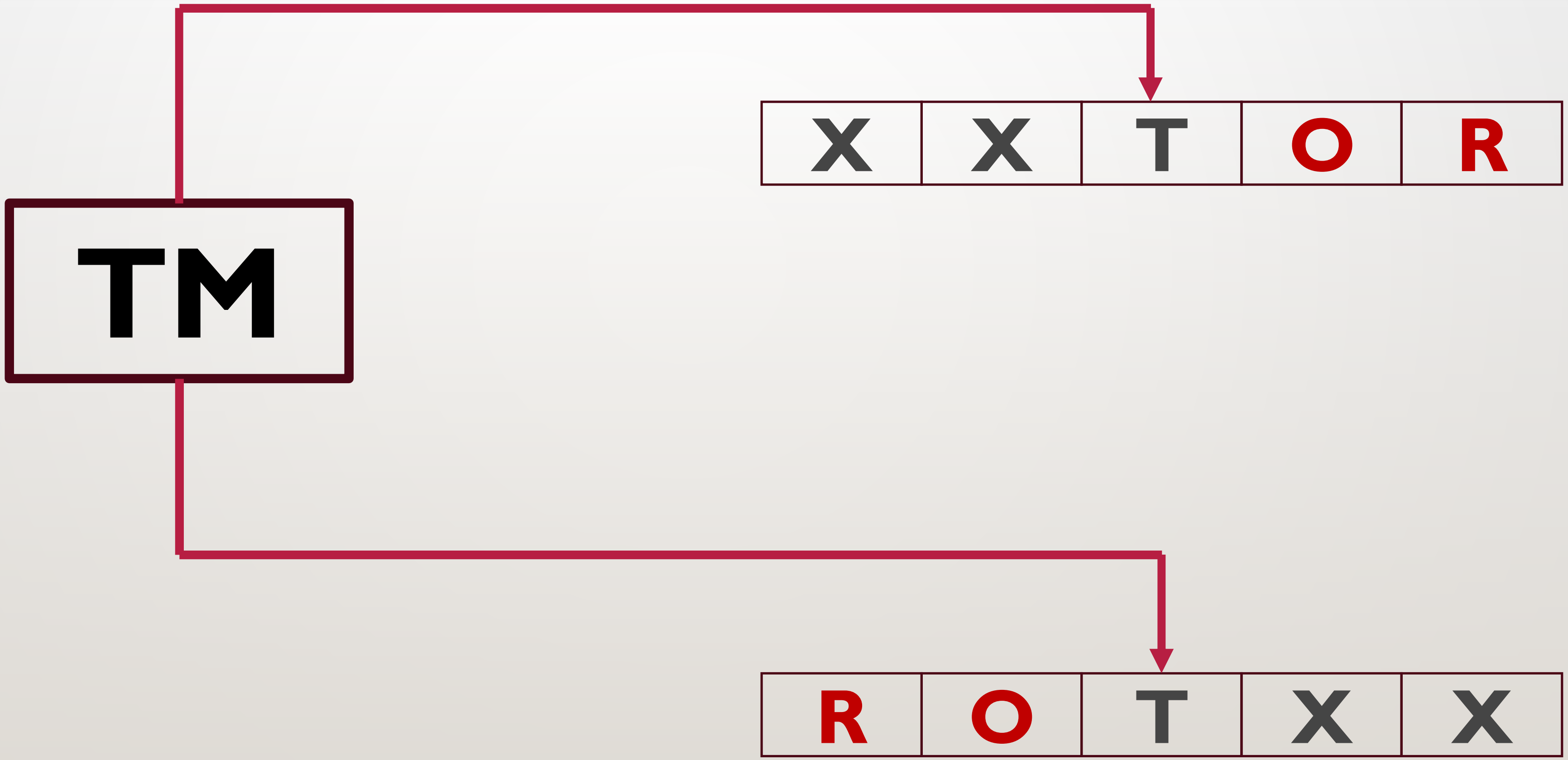
PALINDROME STRING VERIFICATION

k-TAPE



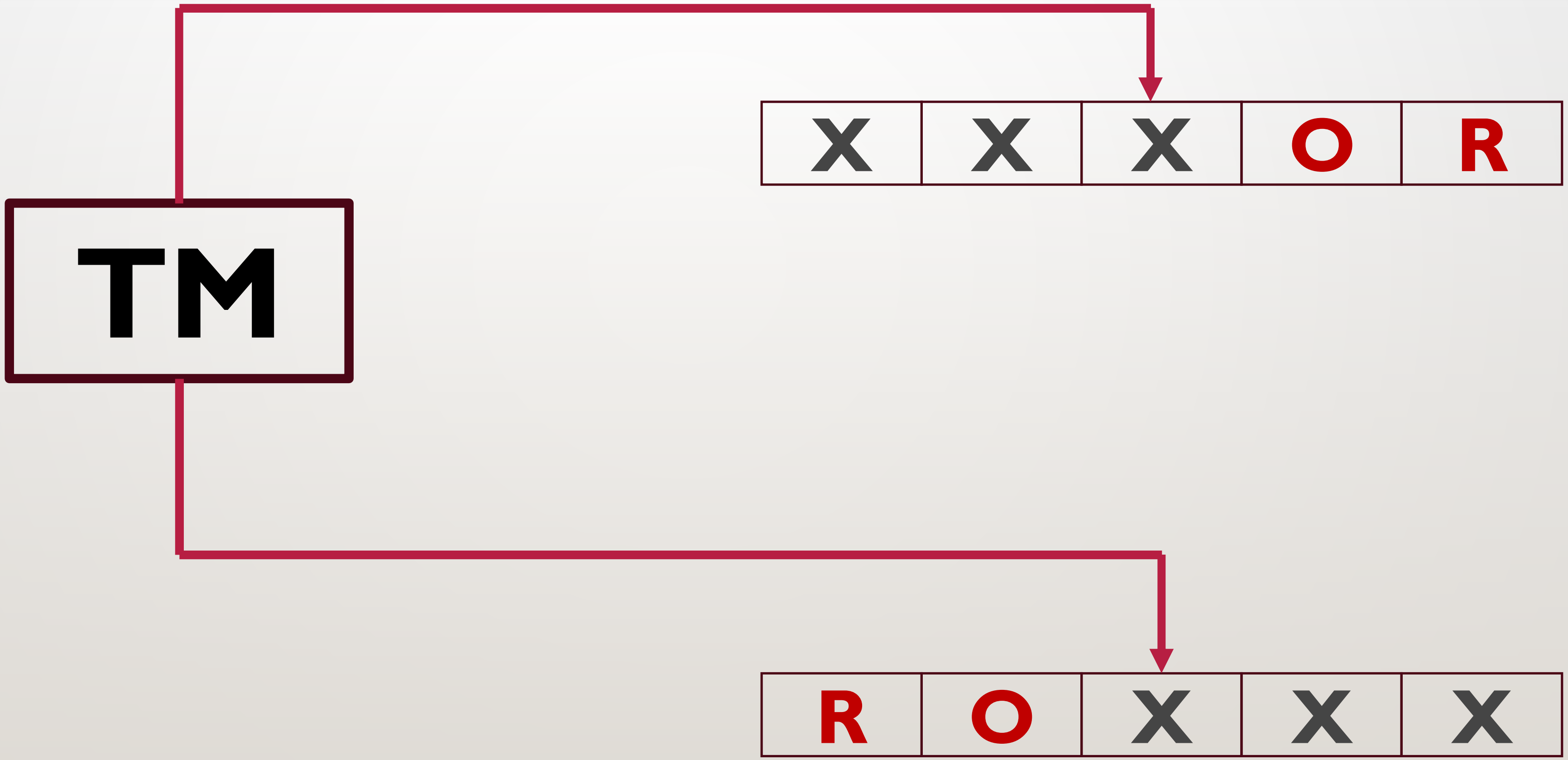
PALINDROME STRING VERIFICATION

k-TAPE



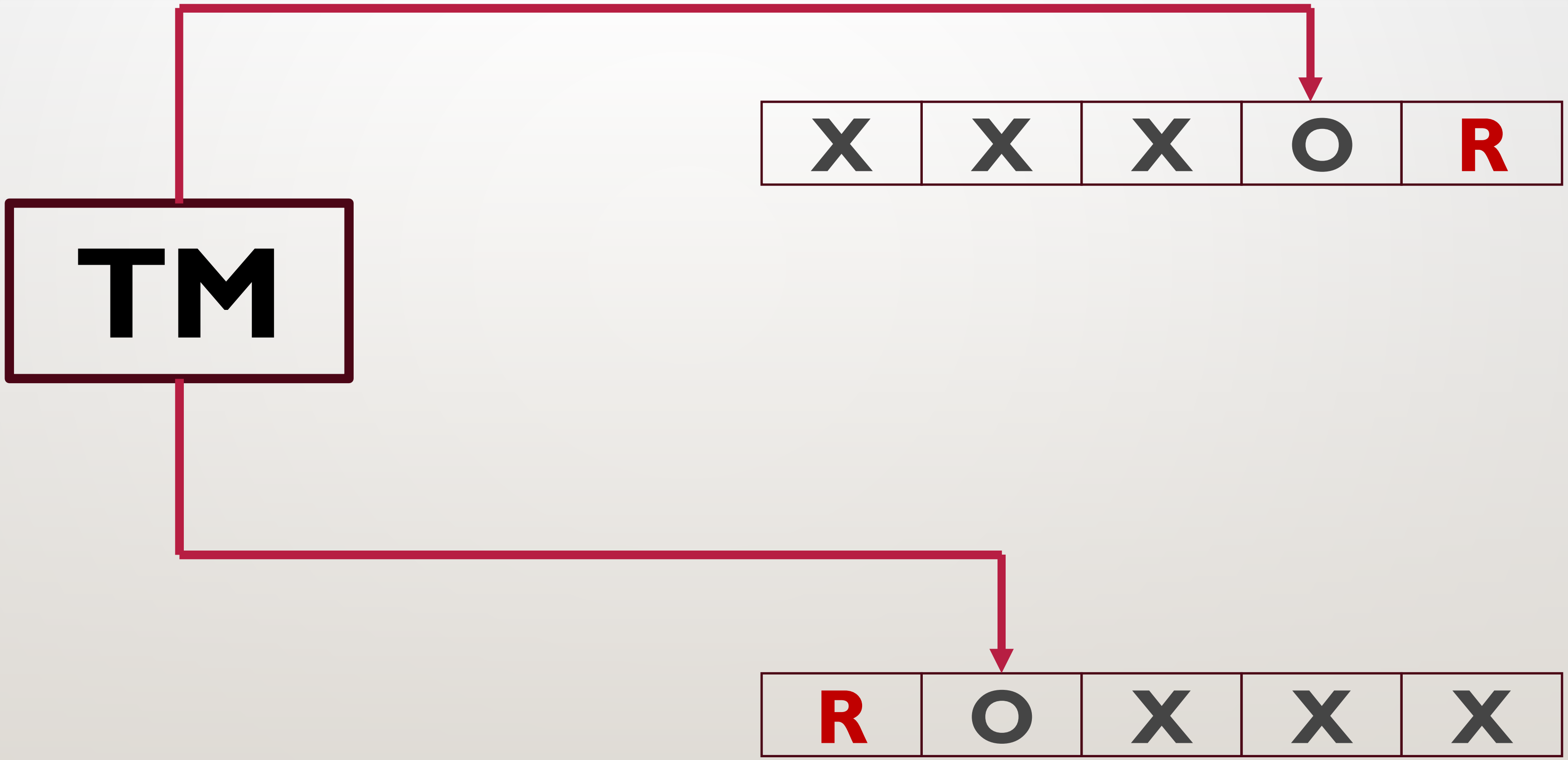
PALINDROME STRING VERIFICATION

k-TAPE



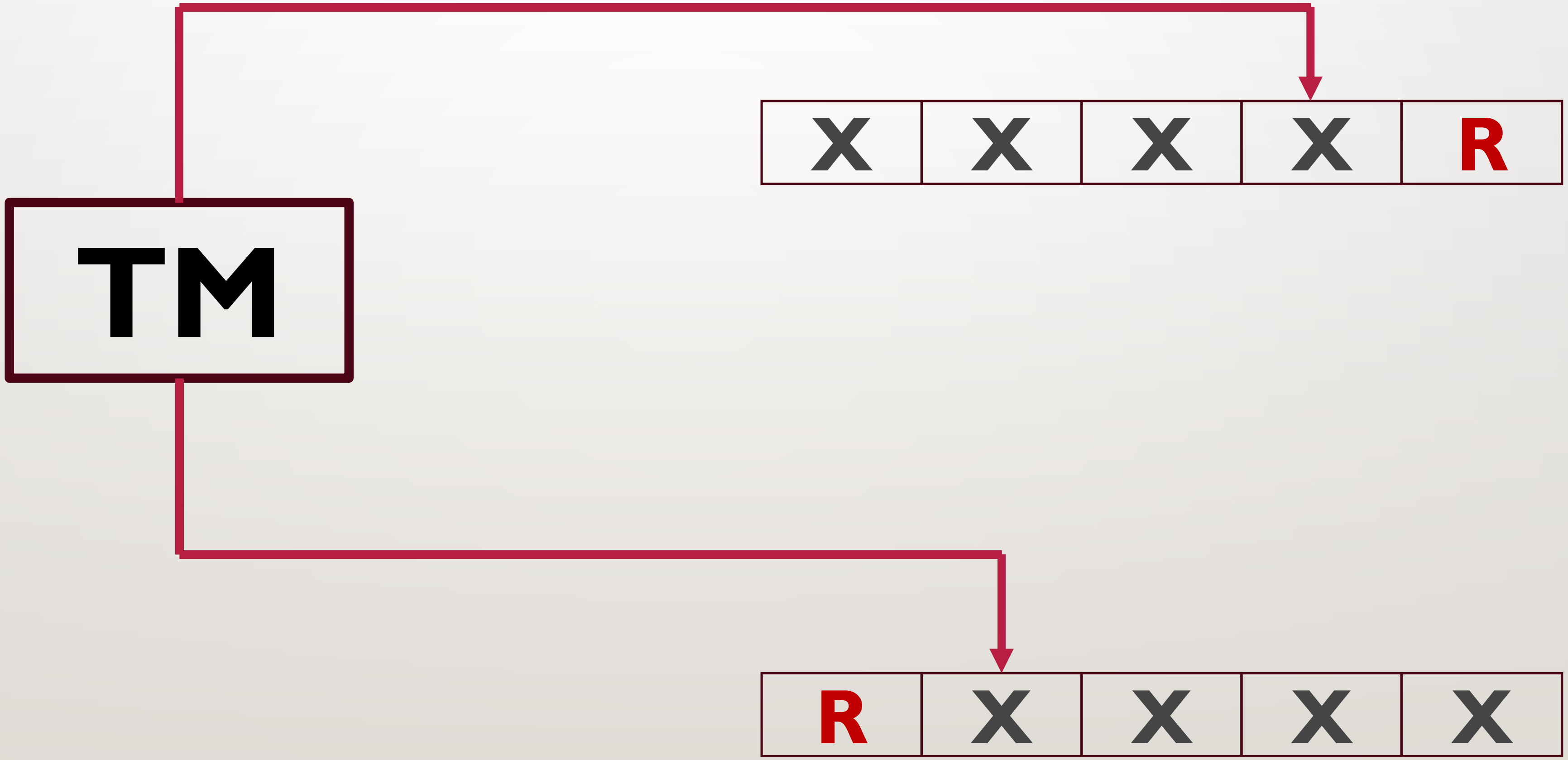
PALINDROME STRING VERIFICATION

k-TAPE



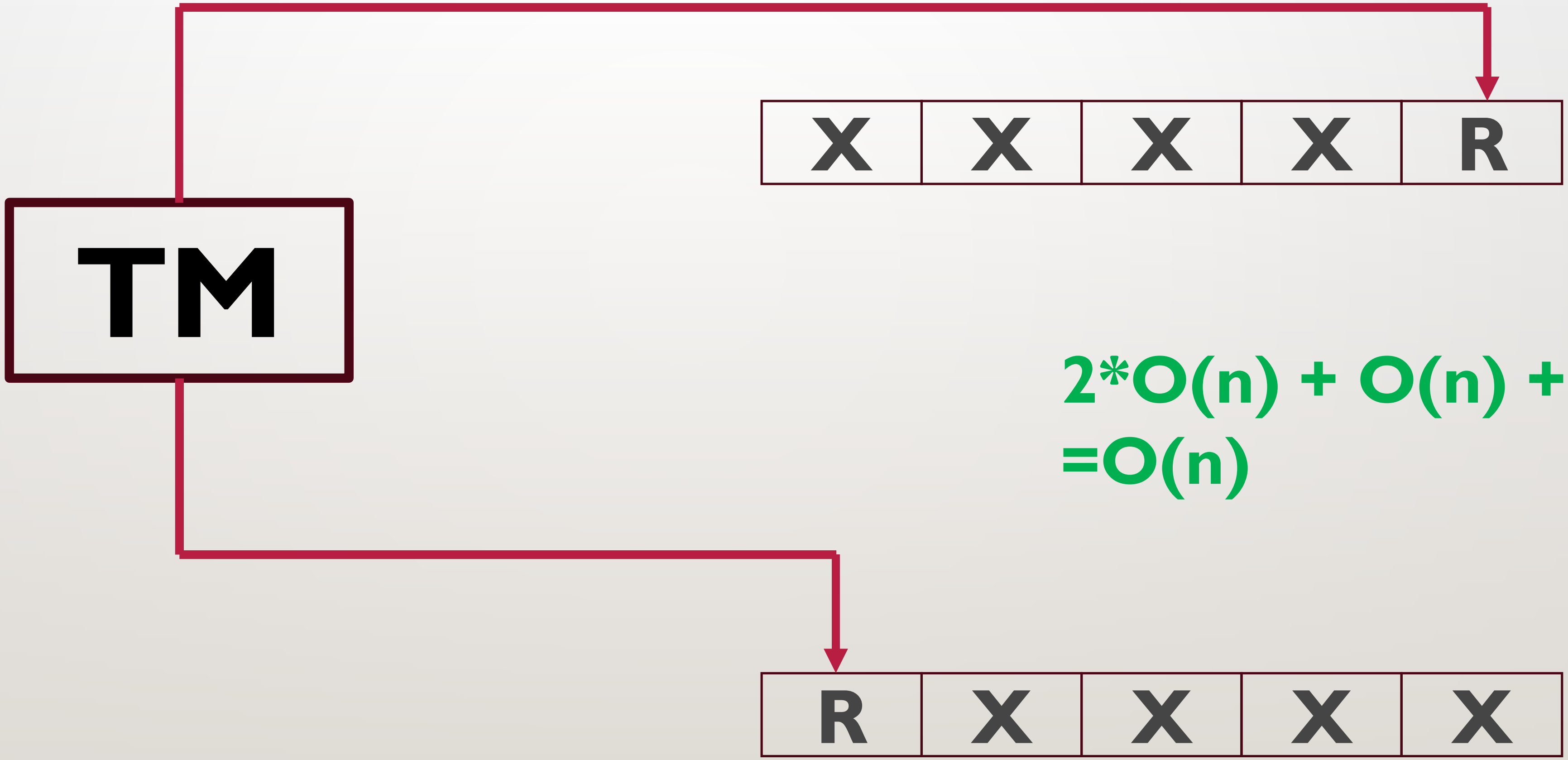
PALINDROME STRING VERIFICATION

k-TAPE



PALINDROME STRING VERIFICATION

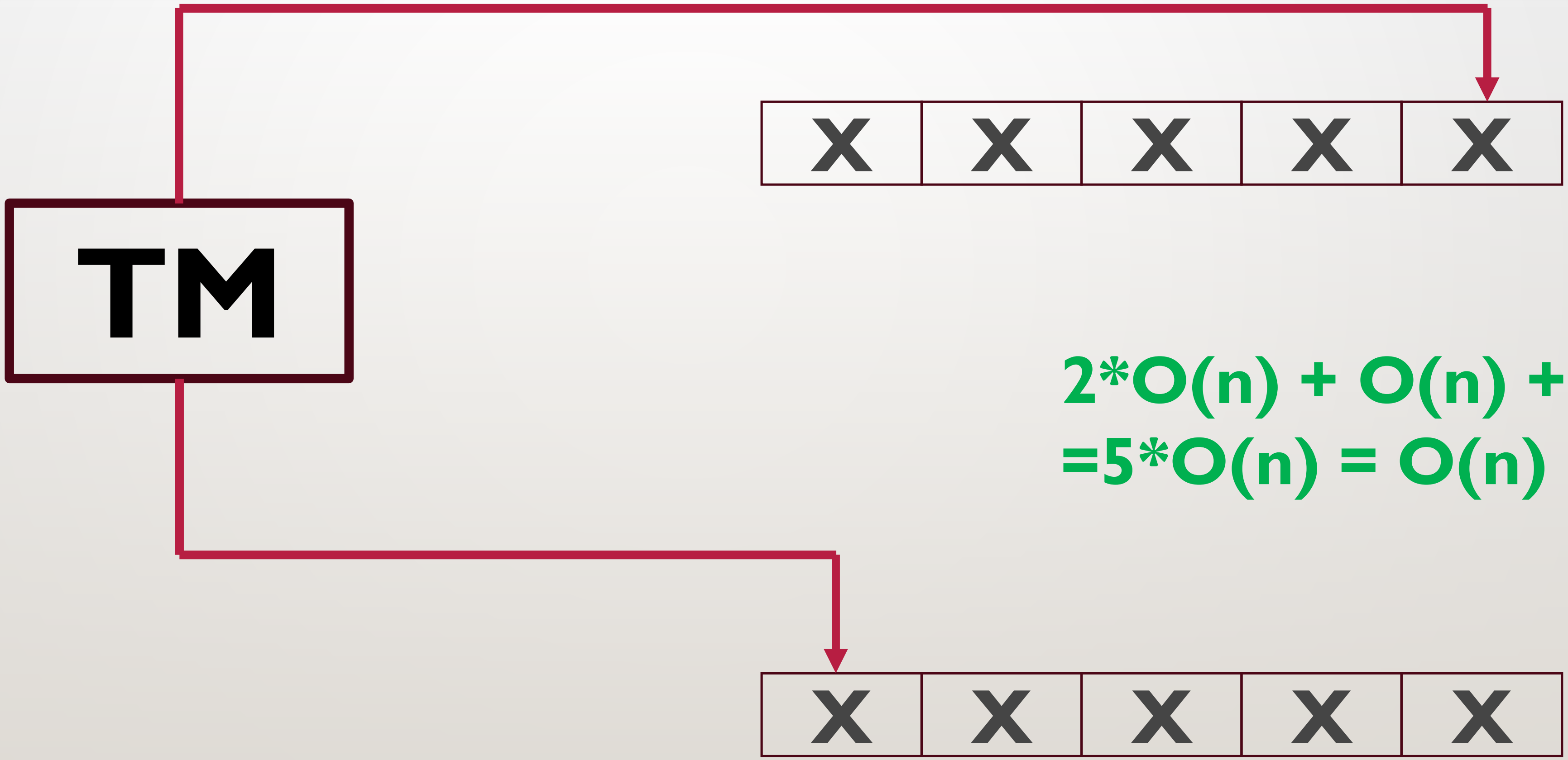
k-TAPE



$$2*O(n) + O(n) + 2*O(n)$$
$$=O(n)$$

PALINDROME STRING VERIFICATION

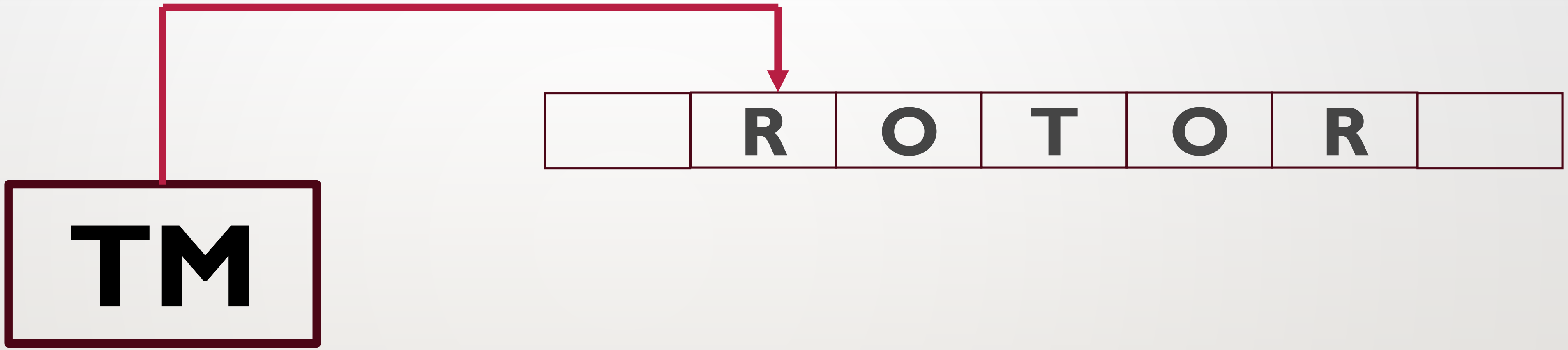
k-TAPE



$$2*O(n) + O(n) + 2*O(n)$$
$$=5*O(n) = O(n)$$

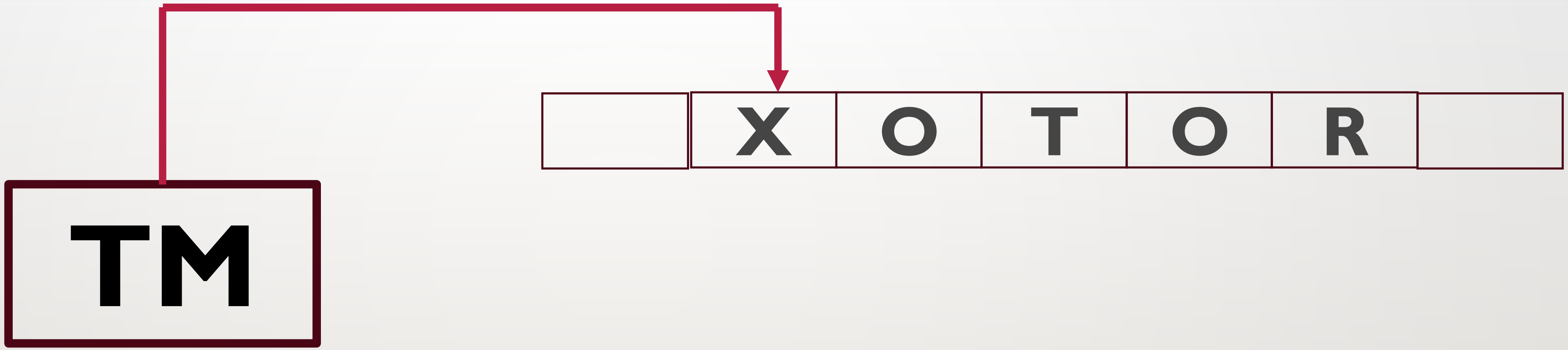
PALINDROME STRING VERIFICATION

SINGLE TAPE



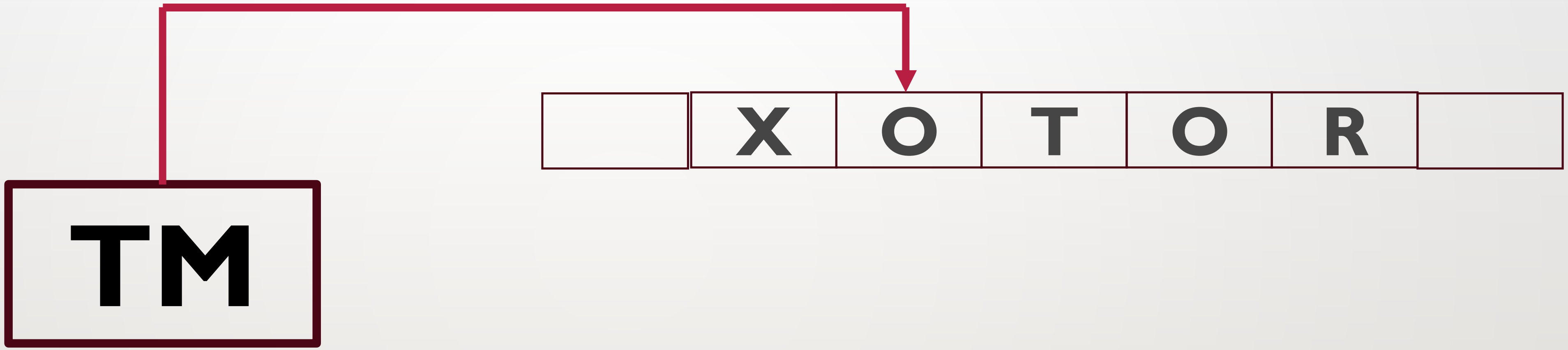
PALINDROME STRING VERIFICATION

SINGLE TAPE



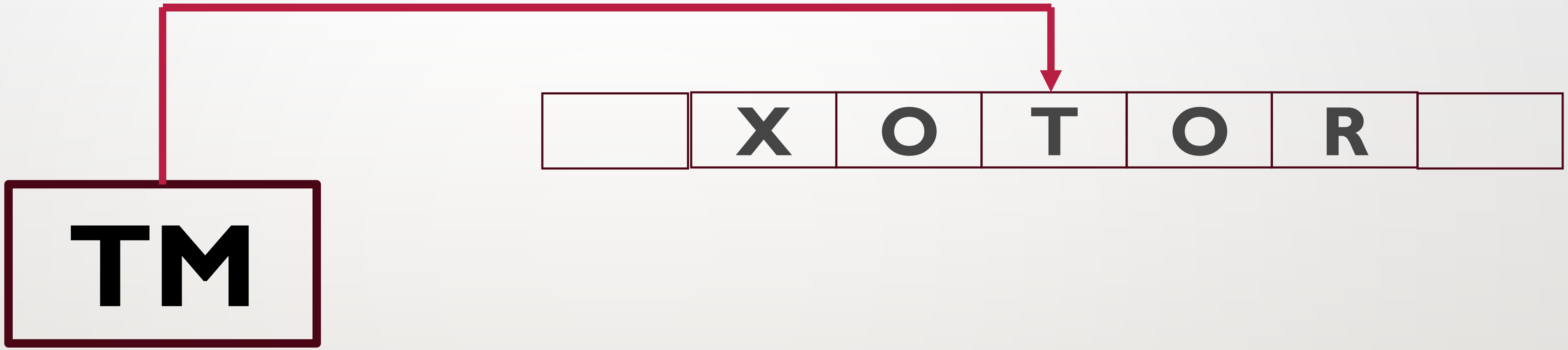
PALINDROME STRING VERIFICATION

SINGLE TAPE



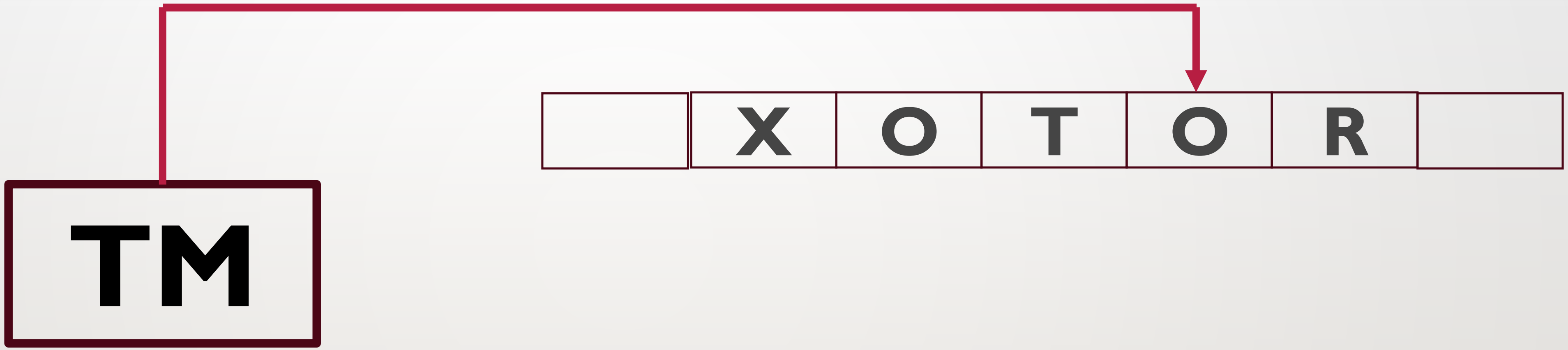
PALINDROME STRING VERIFICATION

SINGLE TAPE



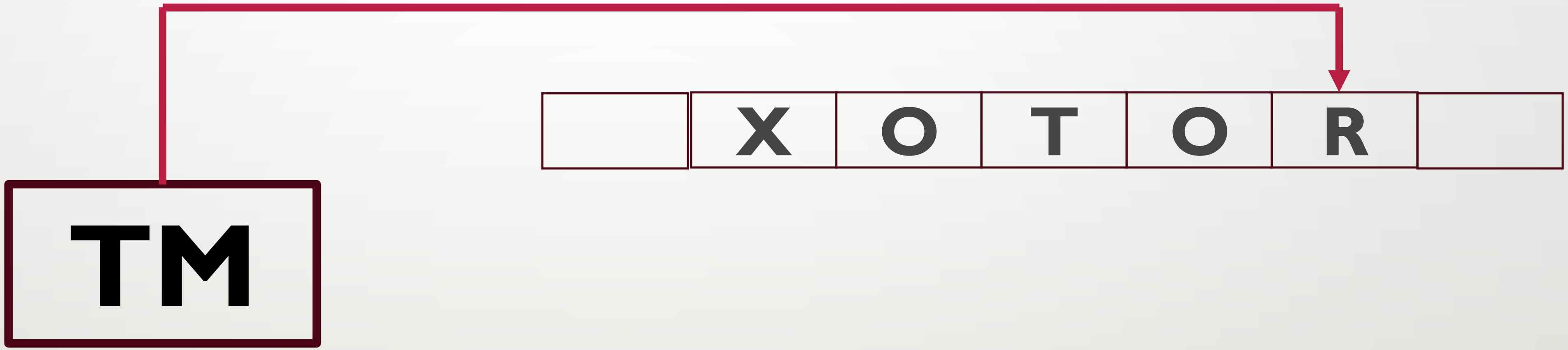
PALINDROME STRING VERIFICATION

SINGLE TAPE



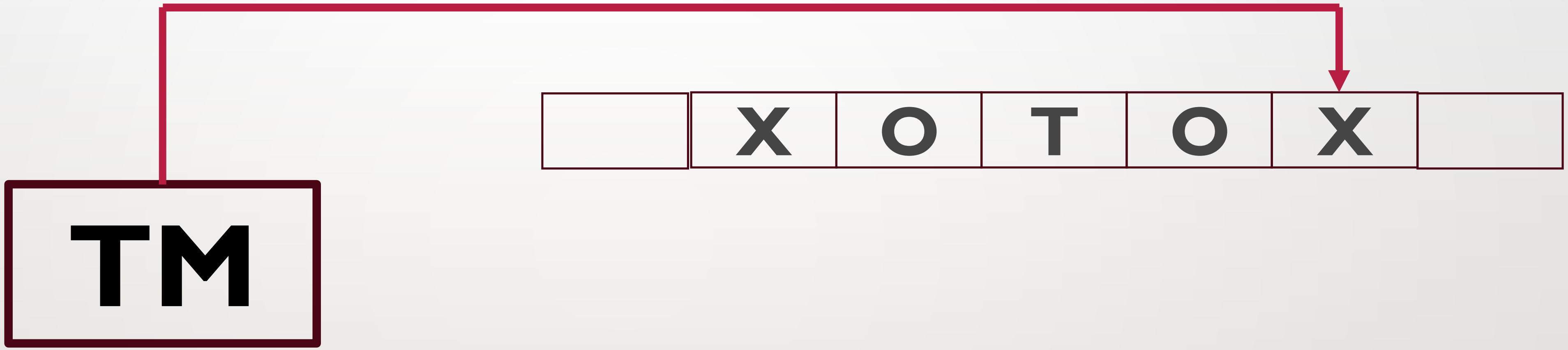
PALINDROME STRING VERIFICATION

SINGLE TAPE



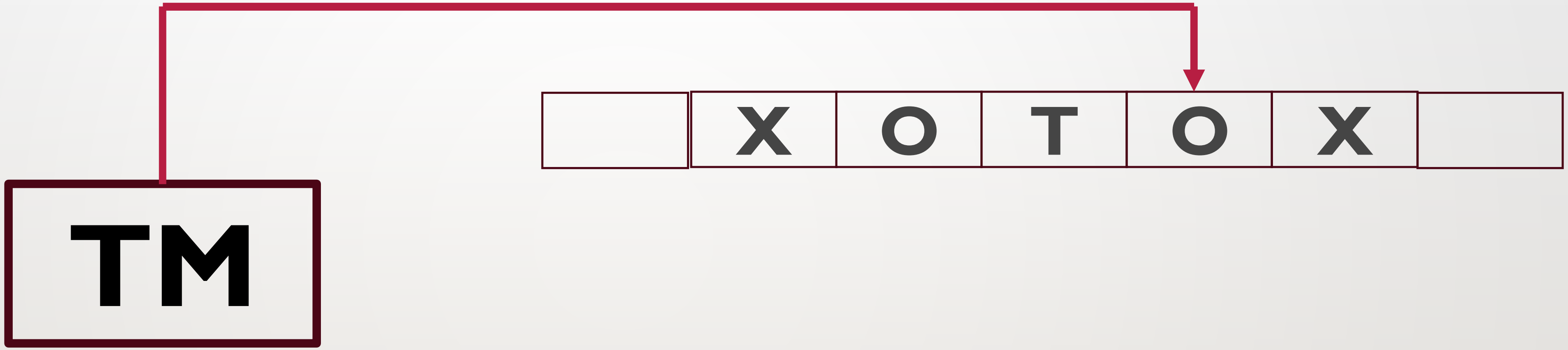
PALINDROME STRING VERIFICATION

SINGLE TAPE



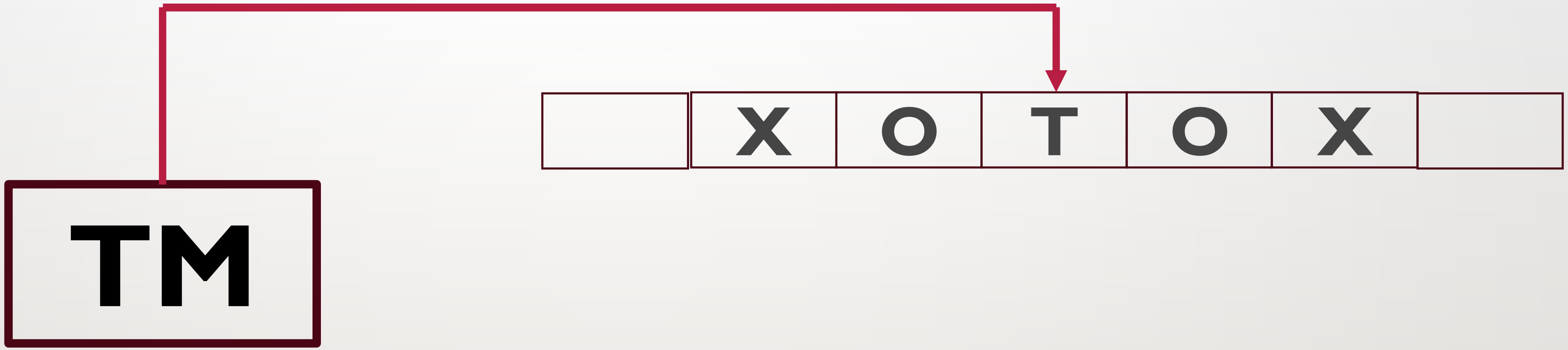
PALINDROME STRING VERIFICATION

SINGLE TAPE

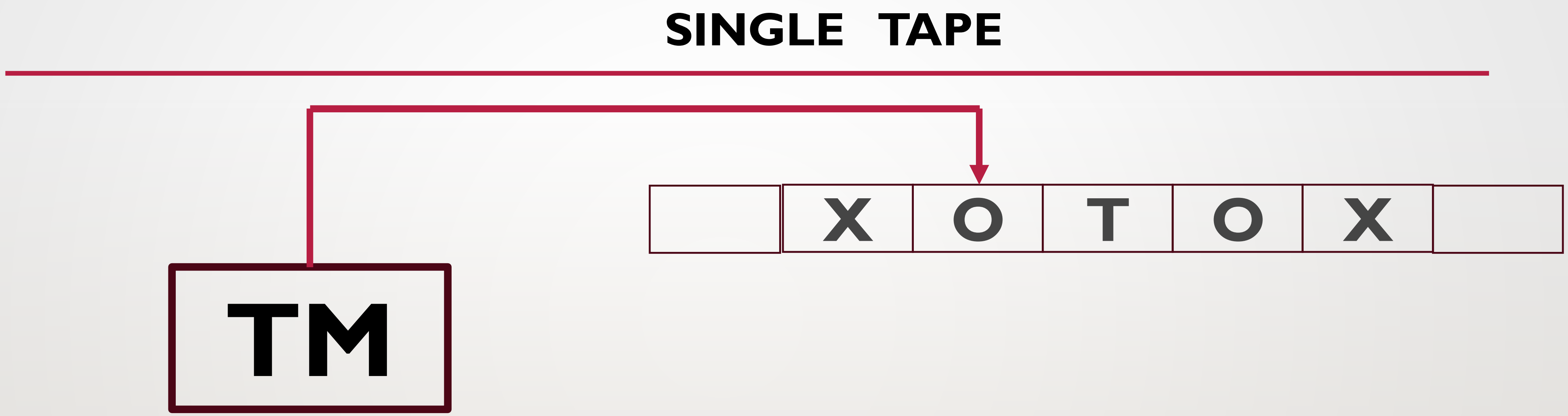


PALINDROME STRING VERIFICATION

SINGLE TAPE

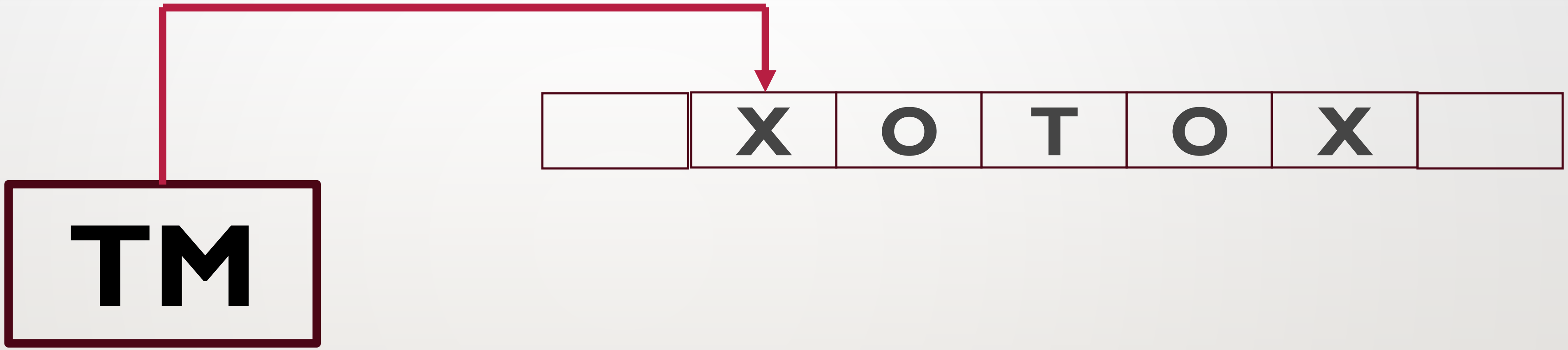


PALINDROME STRING VERIFICATION



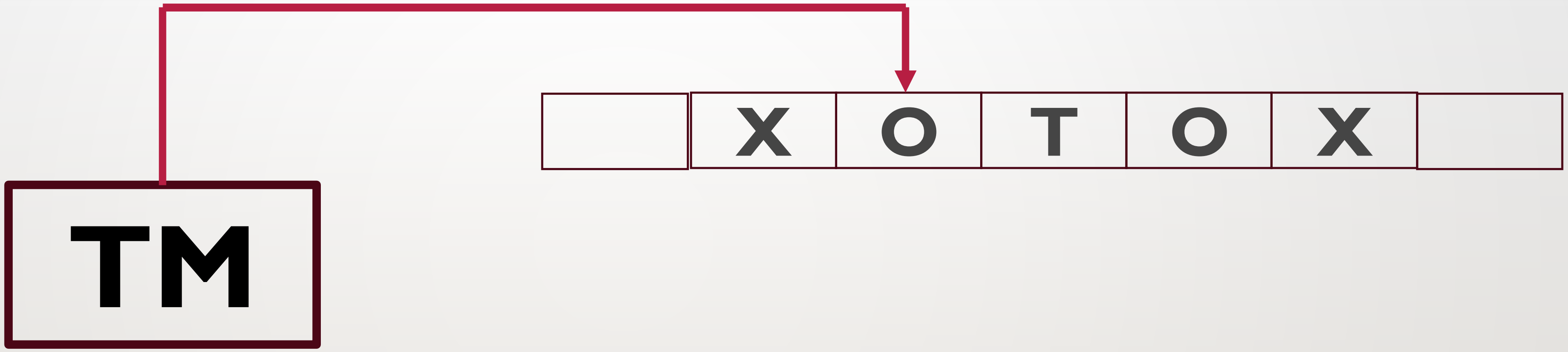
PALINDROME STRING VERIFICATION

SINGLE TAPE



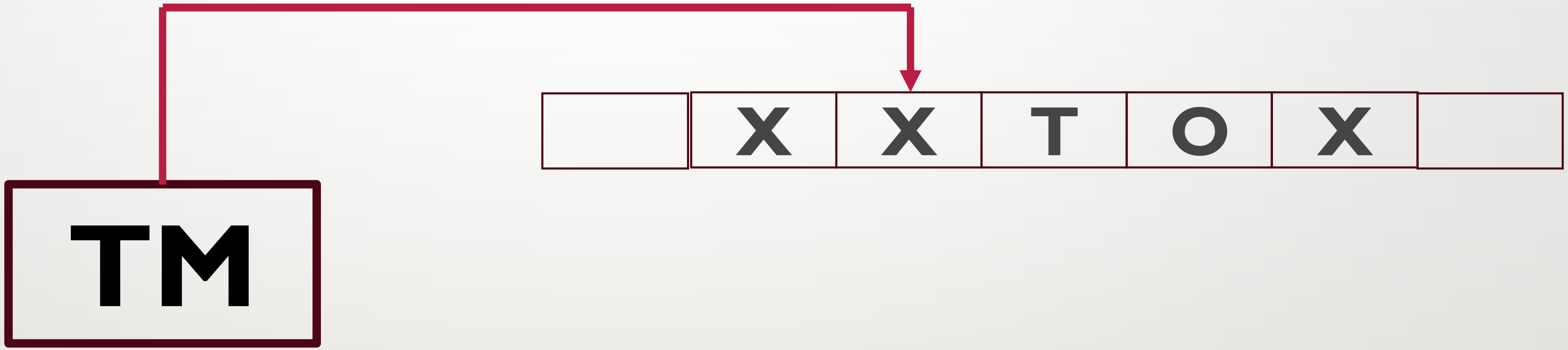
PALINDROME STRING VERIFICATION

SINGLE TAPE



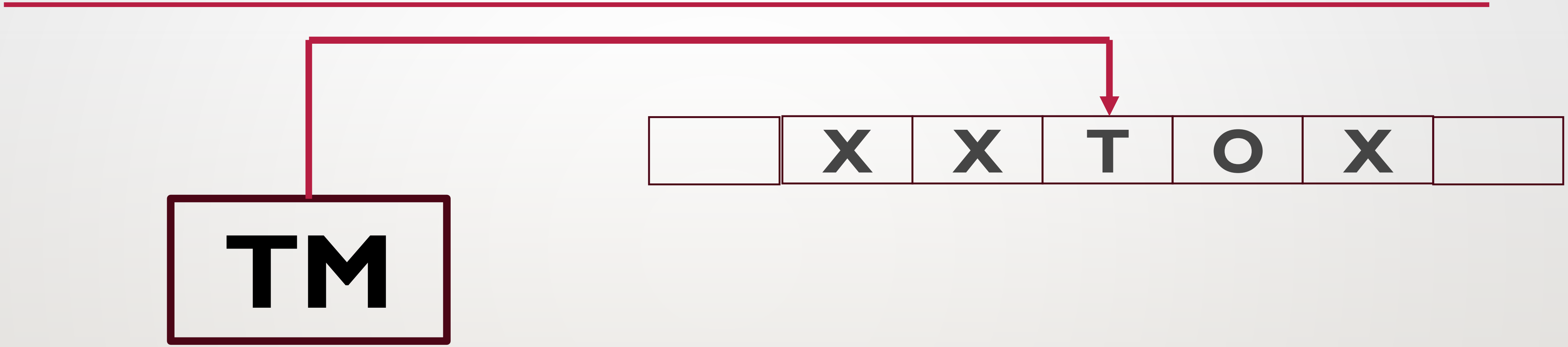
PALINDROME STRING VERIFICATION

SINGLE TAPE



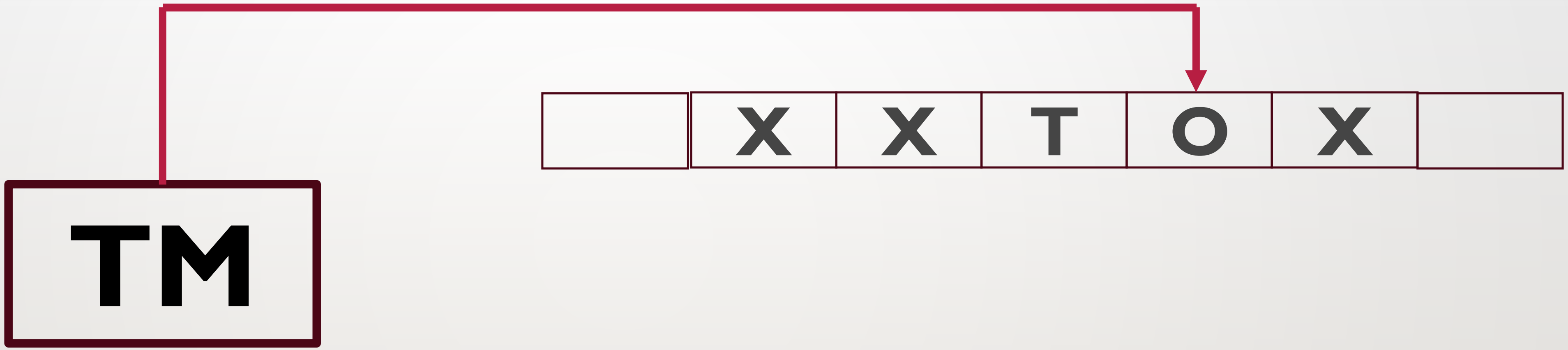
PALINDROME STRING VERIFICATION

SINGLE TAPE



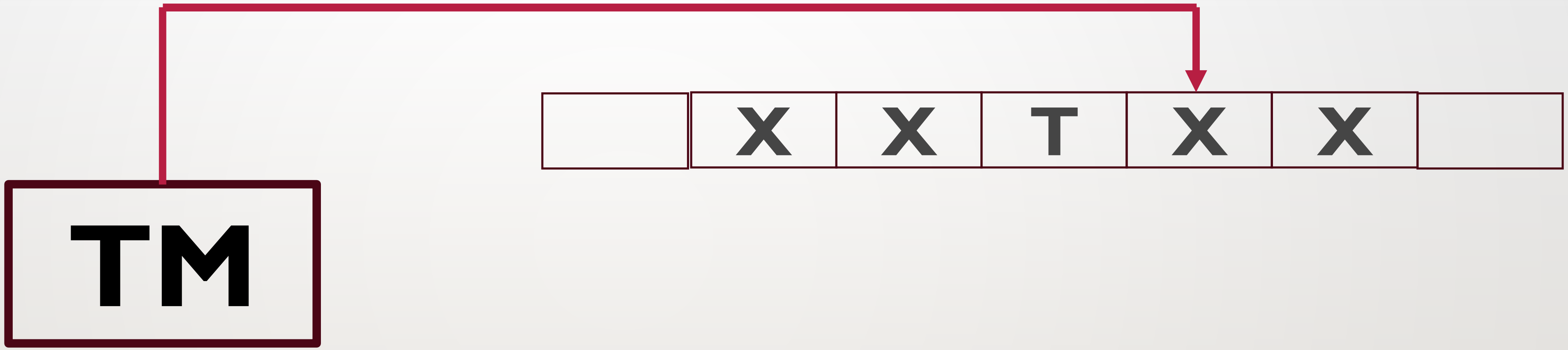
PALINDROME STRING VERIFICATION

SINGLE TAPE



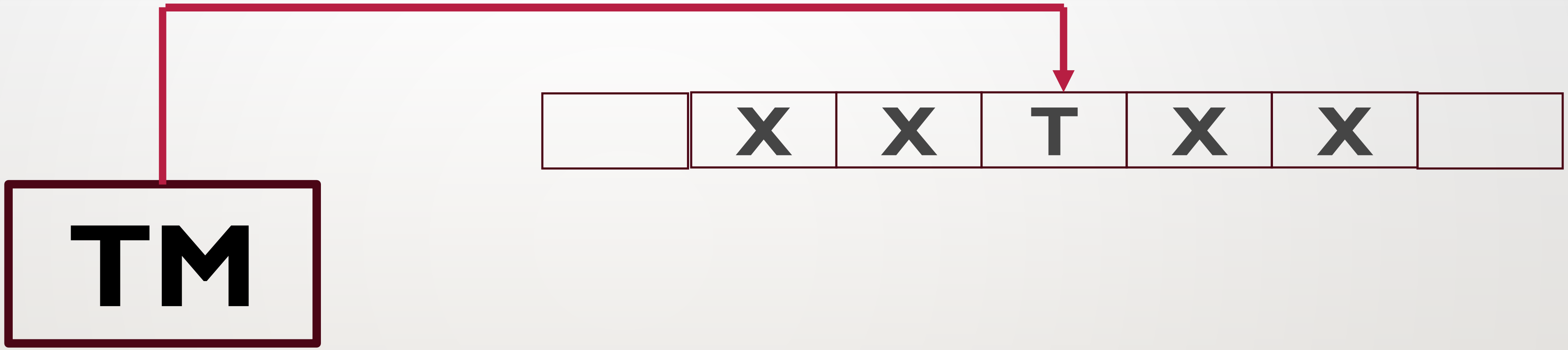
PALINDROME STRING VERIFICATION

SINGLE TAPE



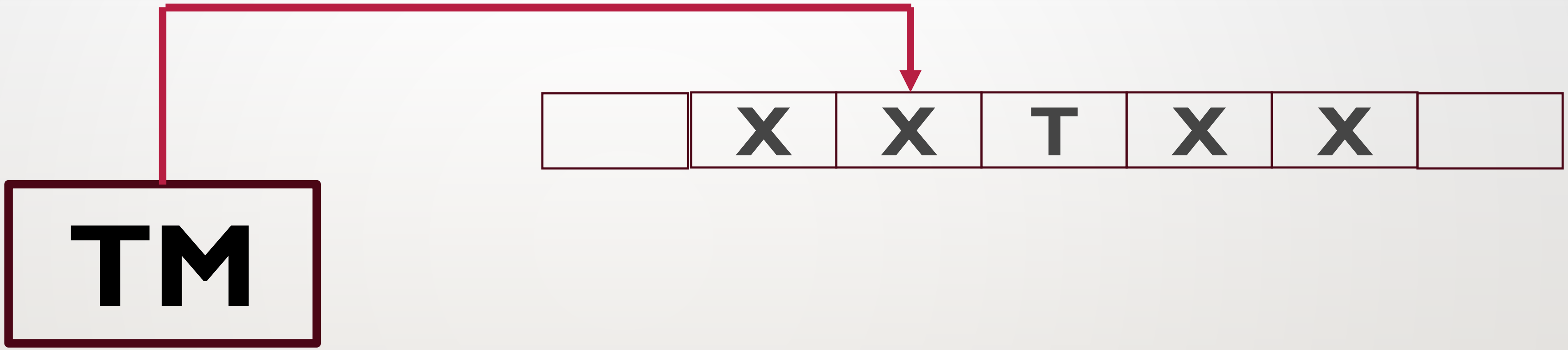
PALINDROME STRING VERIFICATION

SINGLE TAPE



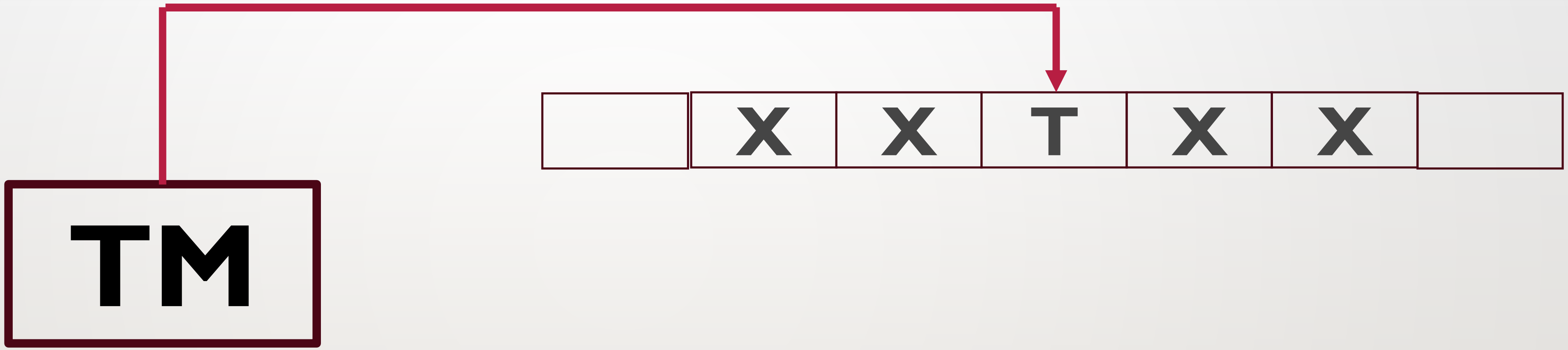
PALINDROME STRING VERIFICATION

SINGLE TAPE



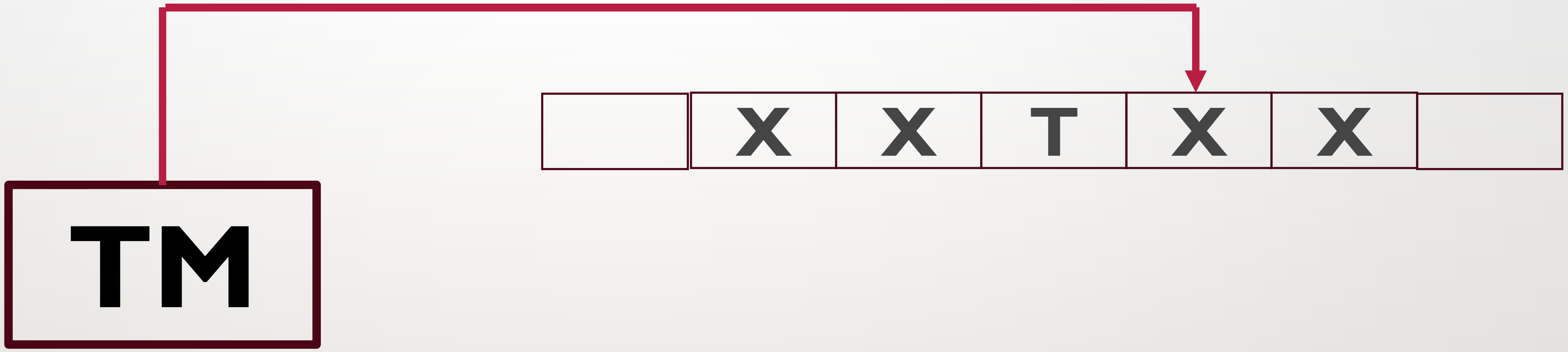
PALINDROME STRING VERIFICATION

SINGLE TAPE



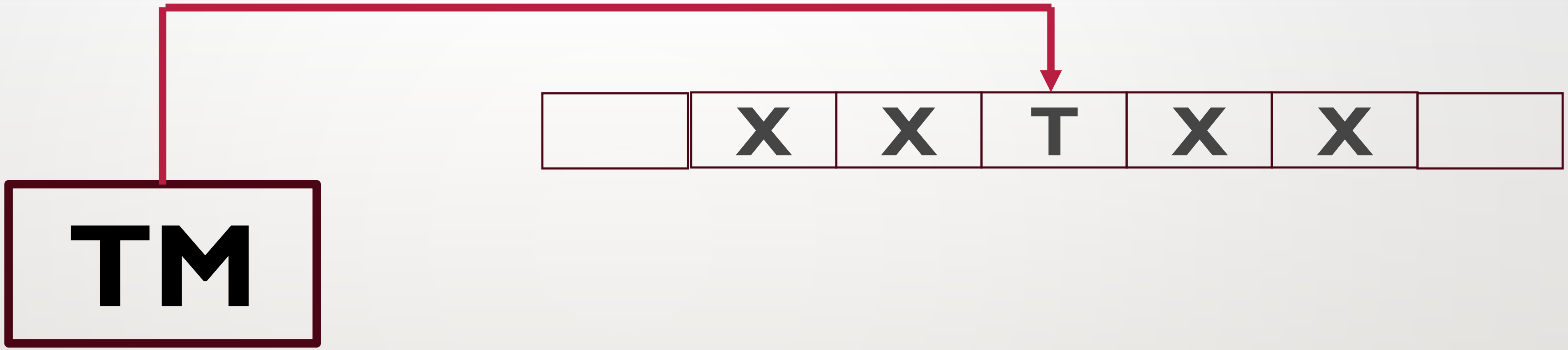
PALINDROME STRING VERIFICATION

SINGLE TAPE

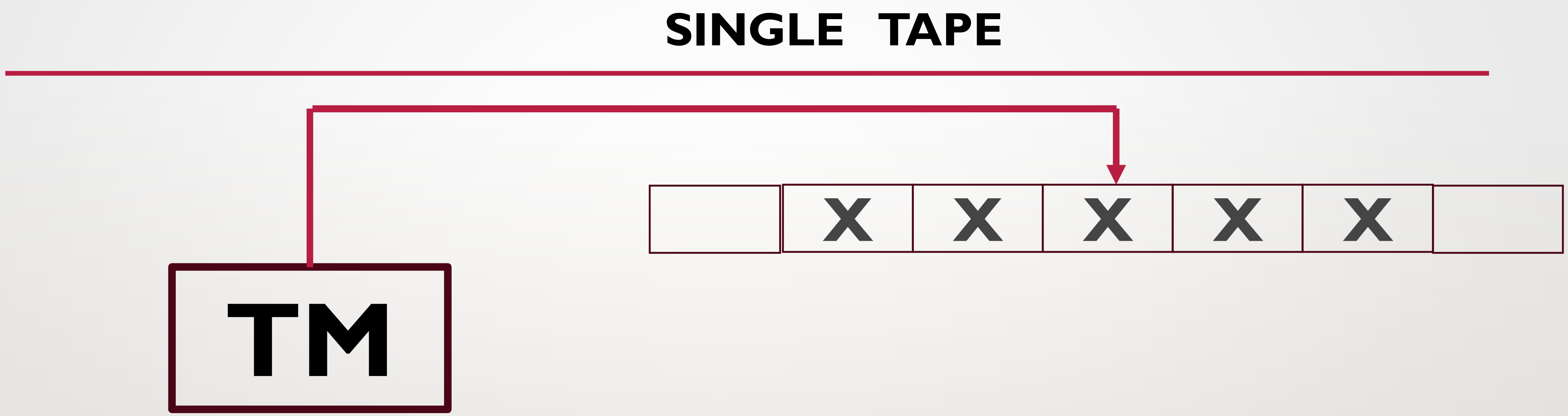


PALINDROME STRING VERIFICATION

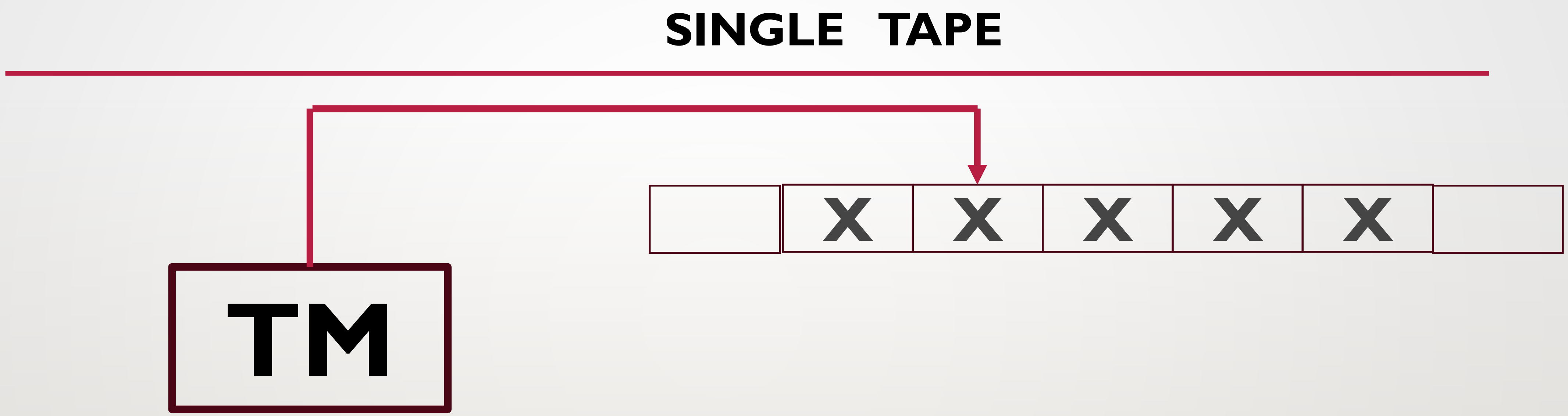
SINGLE TAPE



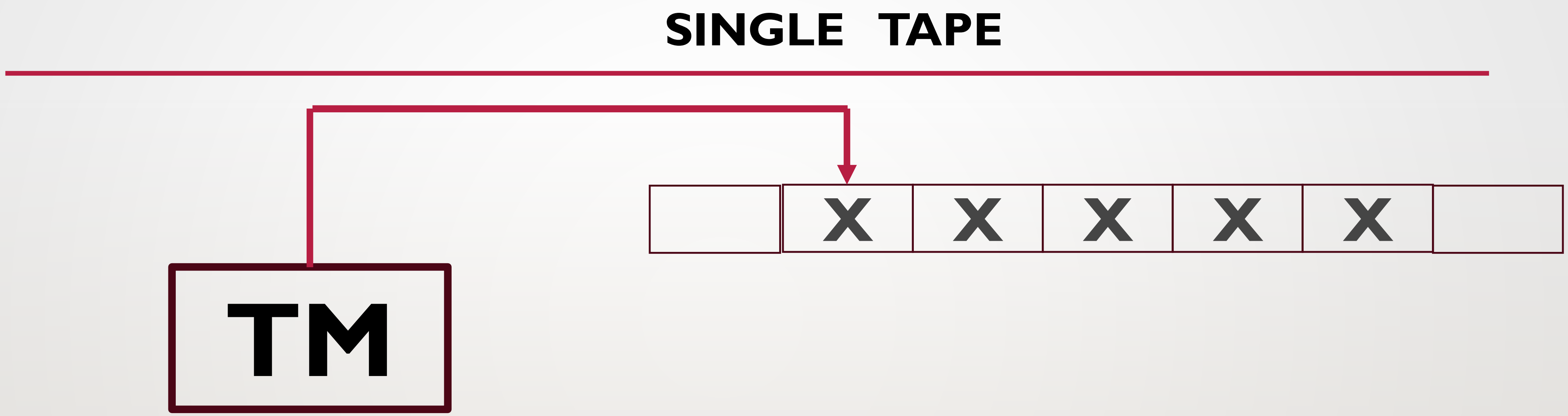
PALINDROME STRING VERIFICATION



PALINDROME STRING VERIFICATION

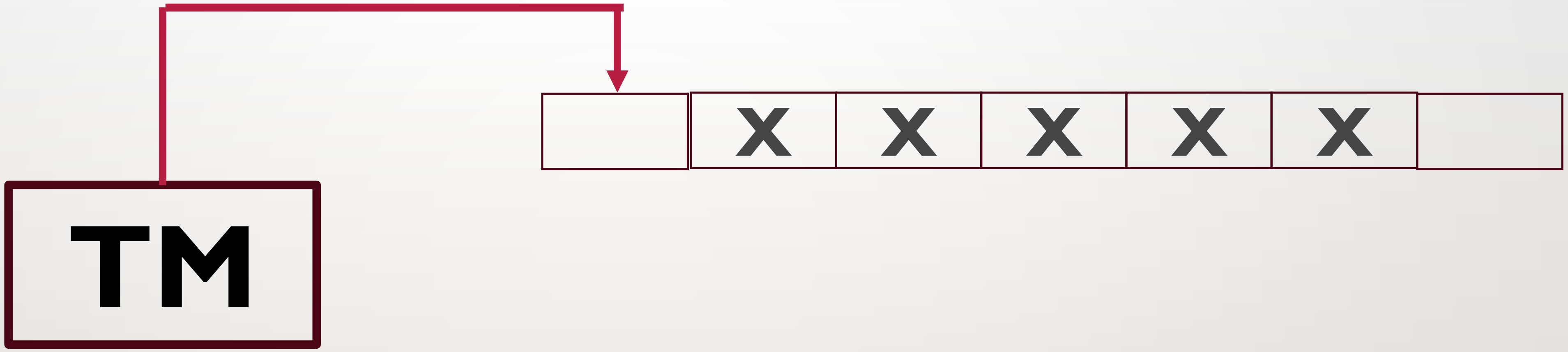


PALINDROME STRING VERIFICATION



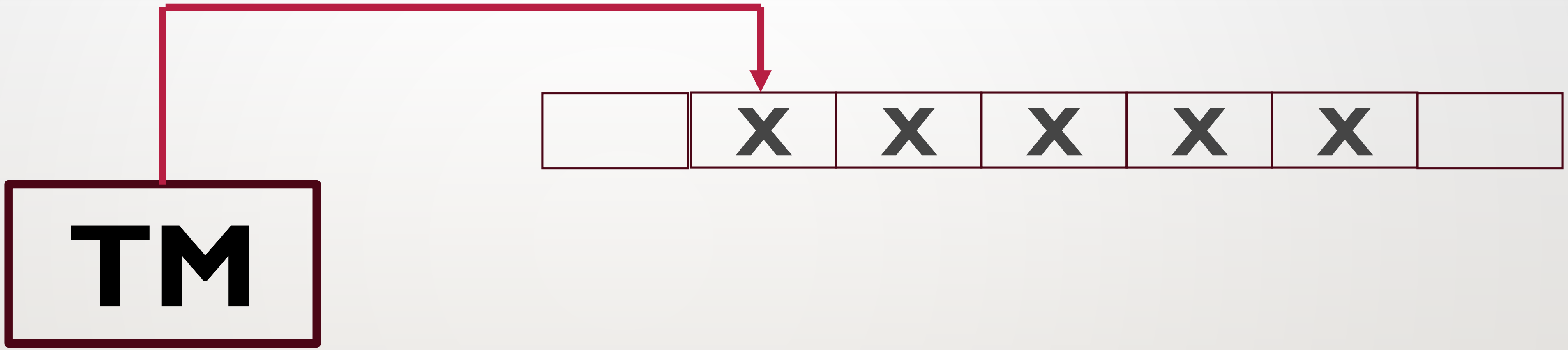
PALINDROME STRING VERIFICATION

SINGLE TAPE



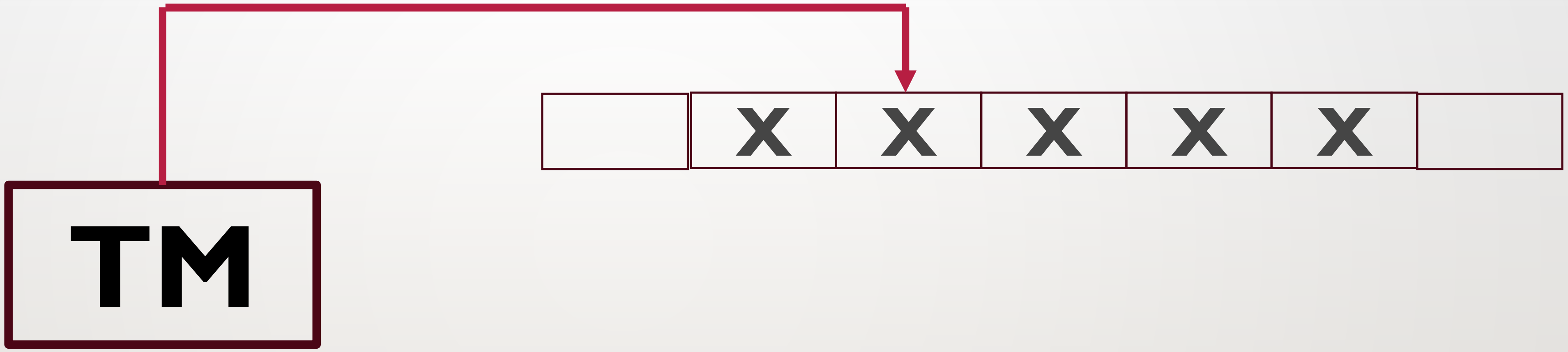
PALINDROME STRING VERIFICATION

SINGLE TAPE



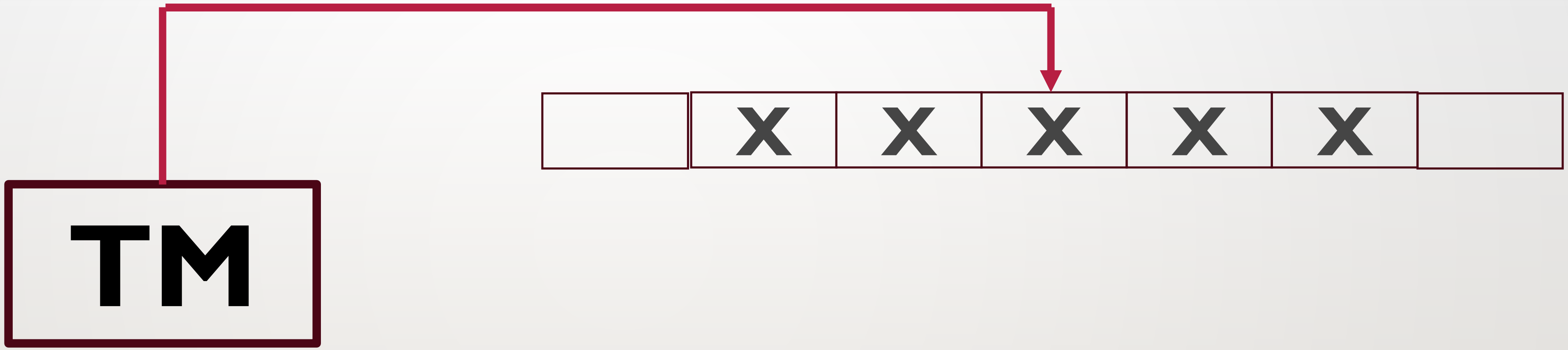
PALINDROME STRING VERIFICATION

SINGLE TAPE



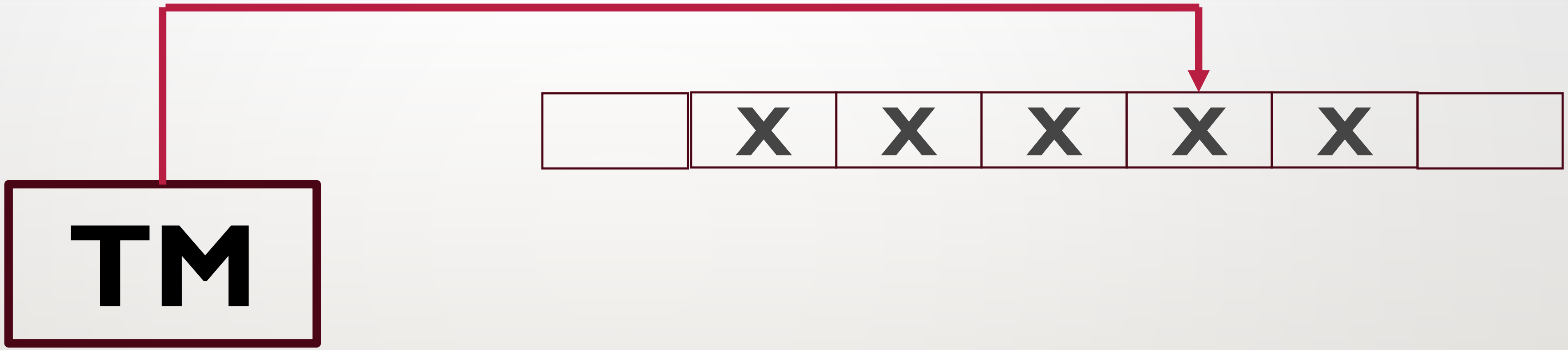
PALINDROME STRING VERIFICATION

SINGLE TAPE



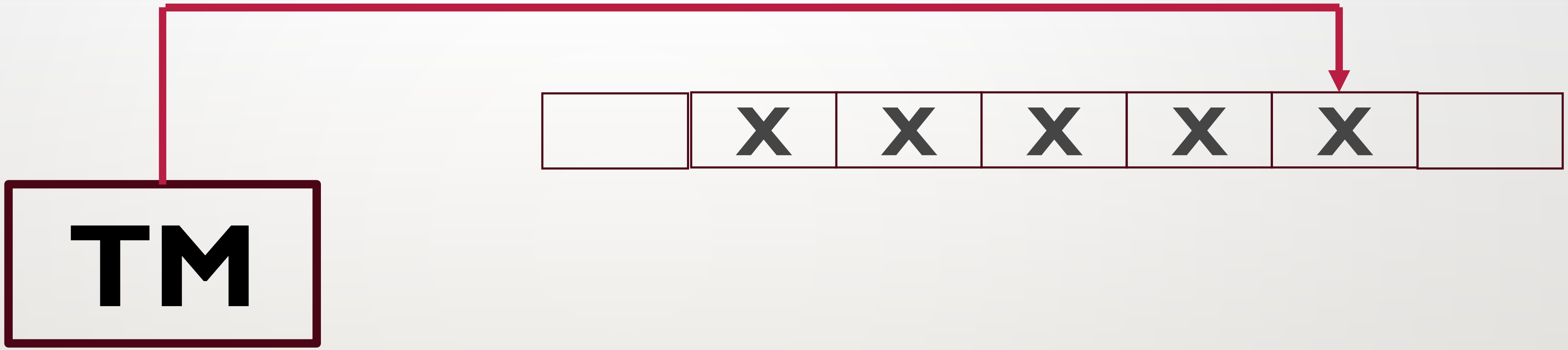
PALINDROME STRING VERIFICATION

SINGLE TAPE

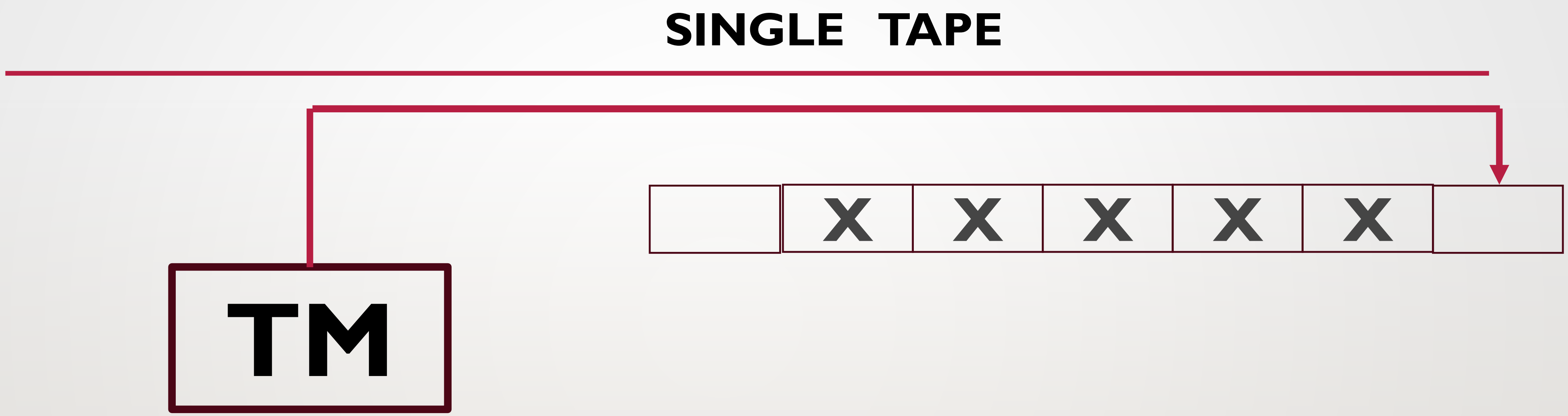


PALINDROME STRING VERIFICATION

SINGLE TAPE

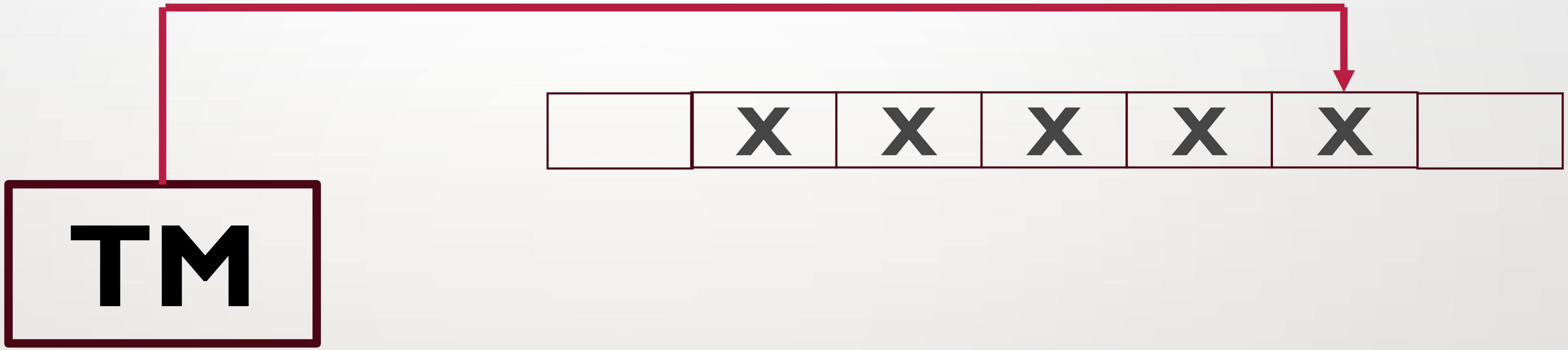


PALINDROME STRING VERIFICATION



PALINDROME STRING VERIFICATION

SINGLE TAPE



$O(n^2)$

CLASS OF LANGUAGES

- Let $t: \mathbb{N} \rightarrow \mathbb{R}^{+}$ be a function. Define the *time complexity class*, $\boldsymbol{TIME(t(n))}$, to be the collection of all languages that are decided by an $O(t(n))$ time Turing Machine.

CHALLENGES FACED

- Classifying problems by time complexity is challenging when grouping those requiring quadratic (n^2) or cubic (n^3) time into one set.
- This challenge stems from the dependency of classification on the underlying execution architecture.
- A classification system is needed to accommodate such problems within a polynomial time complexity class, regardless of execution architecture.

CLASS 'P'

- P is the class of languages decidable in polynomial time on a deterministic single-tape Turing Machine

$$P = \bigcup_{k=1}^{\infty} TIME(n^k)$$

P is class of problems realistically solvable on a computer.

Properties

- ***Robust*** i.e. independent of exact model of computation.
- ***Efficient*** i.e. It constitutes all efficient algorithms

CLASS 'P'

- *To show an algorithm belongs to P, we need to:*
  - **Provide polynomial upper bound on number of stages**
  - **Examine each stage to ensure it can run in polynomial time**

CLASS 'P'

Examples

Path problem

- Is there a path from s to t in a given graph G?
- PATH={<G, s, t> | G is directed graph with directed path from s to t}

Example: *Path problem*

- **Brute force approach**
- If G has m nodes, path cannot be more than m
- Upper bound on possible paths m^m
- Try each “path” one by one for legality and for linking s-to-t

Example: *Path problem*

- Breadth first search
- Place mark on node s
- Repeat until no new nodes marked
    - Scan all edges; If edge (a, b) found from marked node a to unmarked b, mark b
- If t is marked, accept; Otherwise, reject

CLASS 'P'

Breadth First Search Time Complexity

- Measure complexity based on number of nodes n
- Place mark on node s ← n steps (find node s in list of m nodes)
- Repeat until no new nodes marked ← n steps (if mark 1 new node/ loop)
- For each edge (a, b) with 'a' already marked, mark 'b' also ← $O(n^3)$

(n^2 max total edges, for each edge, search for 'a' to see if marked (n steps), then mark b in list if needed (n steps)

- In total: $n^2 \times (n + n) = 2n^3$
- If t marked, accept; else, reject ← n steps (find node t in list of n nodes)
- In total: $f(n)= n + n \times 2n^3 + n = 2m^4 + 2n = O(n^4)$ It's POLYNOMIAL!

CLASS 'P'

Example: RELPRIME

- Two numbers are relatively prime if **1** is the largest number that evenly divides them both
- **10** and **21** are relatively prime
- **10** and **22** are not relatively prime
- Solution: search all divisors from 2 until min(x, y) / 2
- min(x, y) / 2 numbers tried, min(x, y) /2 steps
- size of input n = length of binary encoding = $\log_2(\max(x, y))$
- 2^n steps – exponential complexity!

RELPRIME: Faster Solution

Euclidean algorithm

- E = On input <x,y>
- Repeat until y = 0
  - Assign x = x mod y
  - Exchange x and y
- Output x
- R = On input <x, y>, Run E on <x, y>
- If result is 1, accept; Otherwise, reject

RELPRIME: Simulating the Euclidean algorithm

x=10	y=21	Perform x % y
x=10	y = 21	x=y and y = (x MOD y)
x=21	y=10	Perform x % y
x=10	y=1	x=y, and y=(x MOD y)
x=0	y=1	

y=1 when x=0, so original numbers relatively prime

CLASS 'P'

RELPRIME: Euclidean Algorithm – complexity

x = x mod y ← new x always less than y

- **If old x is twice y or more, new x will be cut at least in half**
- **If old x between y and 2y, new x will be cut at least in half**
    - **new x = x – y**

Number of loops: $2*\log_2(\max(x, y))$

Length of input (in binary): $\log_2(x) + \log_2(y) = O(\log_2(\max(x, y)))$

Number of loops: $O(n)$

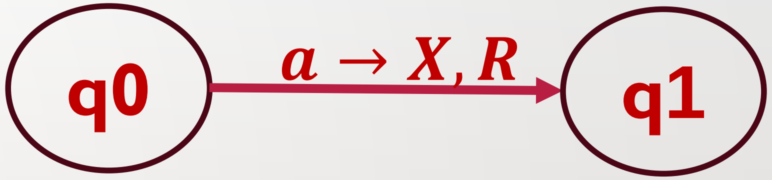
TURING MACHINE

A Decider for the language $L = \{a^n b^n c^n : n > 0\}$

Input Tape

a	a	b	b	c	c

q0aabbcc



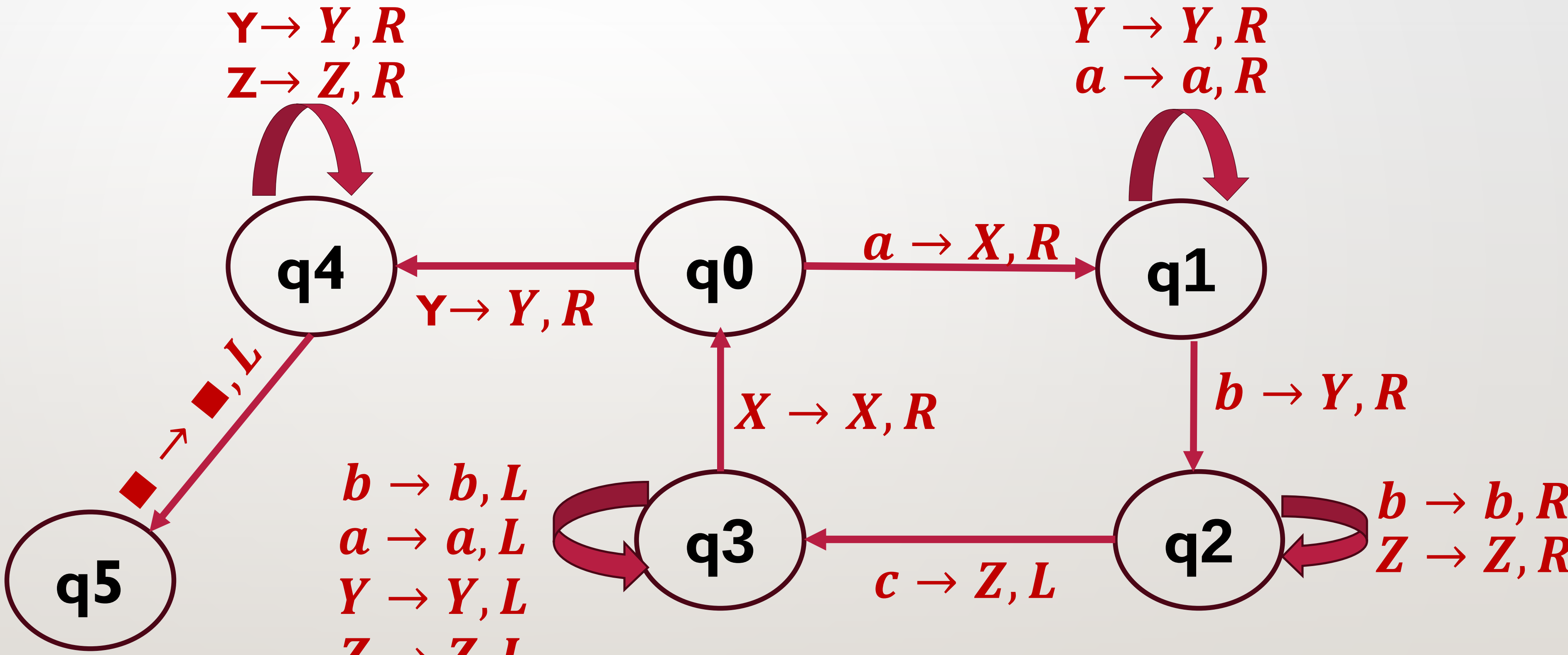
X	a	b	b	c	c

Configuration → xq1abbcc

A Decider for the language $L = \{a^n b^n c^n : n > 0\}$

A partial Computing Path

q0aabbcc
↓
xq1abbcc
↓
xaq1bbcc



TURING MACHINE

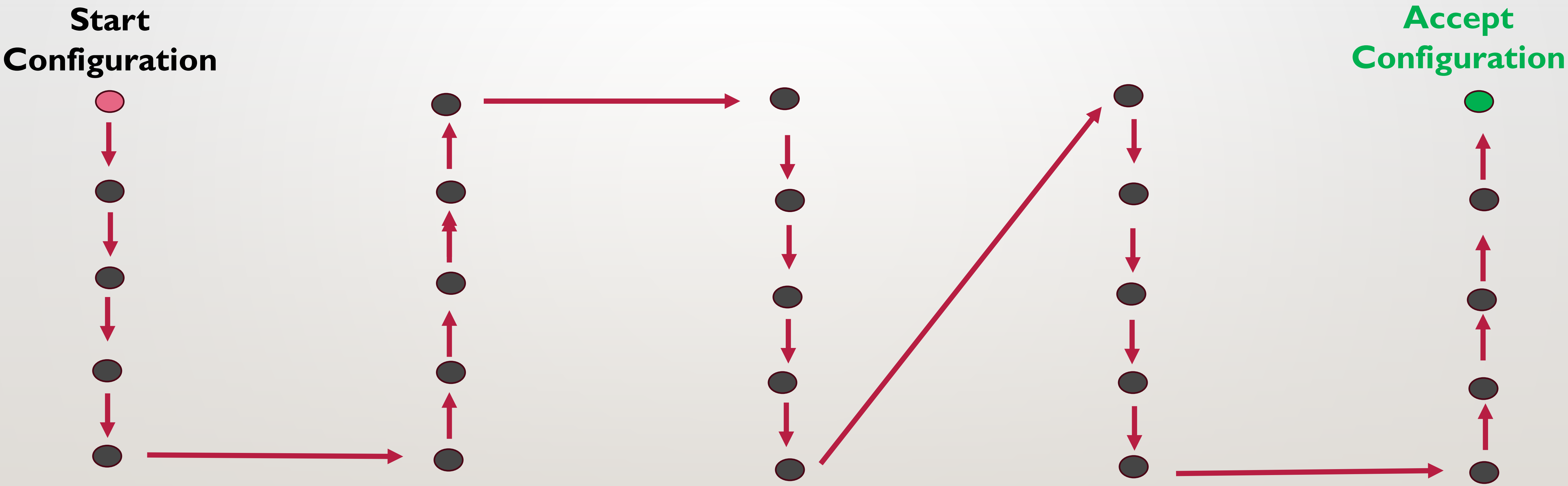
A Decider for the language $L = \{a^n b^n c^n : n > 0\}$

Start configuration | **Accept configuration**

q0aabbcc → xq1abbcc → xaq1bbcc → xayq2bcc → xaybq2cc → xaybq3zc → xayq3bzc → xaq3ybzc → xq3aybzc → q3xaybzc → xq0aybzc → xxq1ybzc → xxyq1bzc → xxyyq2zc → xxyyzq2c → xxyyq3zz → xxyq3yzz → xxq3yyzz → xq3xyyzz → xxq4yyzz → xxyq4yzz → xxyyq4zz → xxyyzq4z → xxyyzzq4■ → xxyyzq4z■

TURING MACHINE

A Decider for the language $L = \{a^n b^n c^n : n > 0\}$

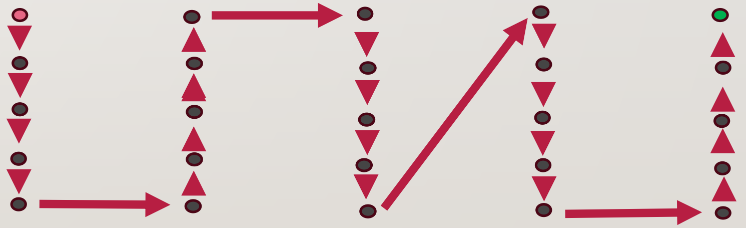


A Decider for the language $L = \{a^n b^n c^n : n > 0\}$

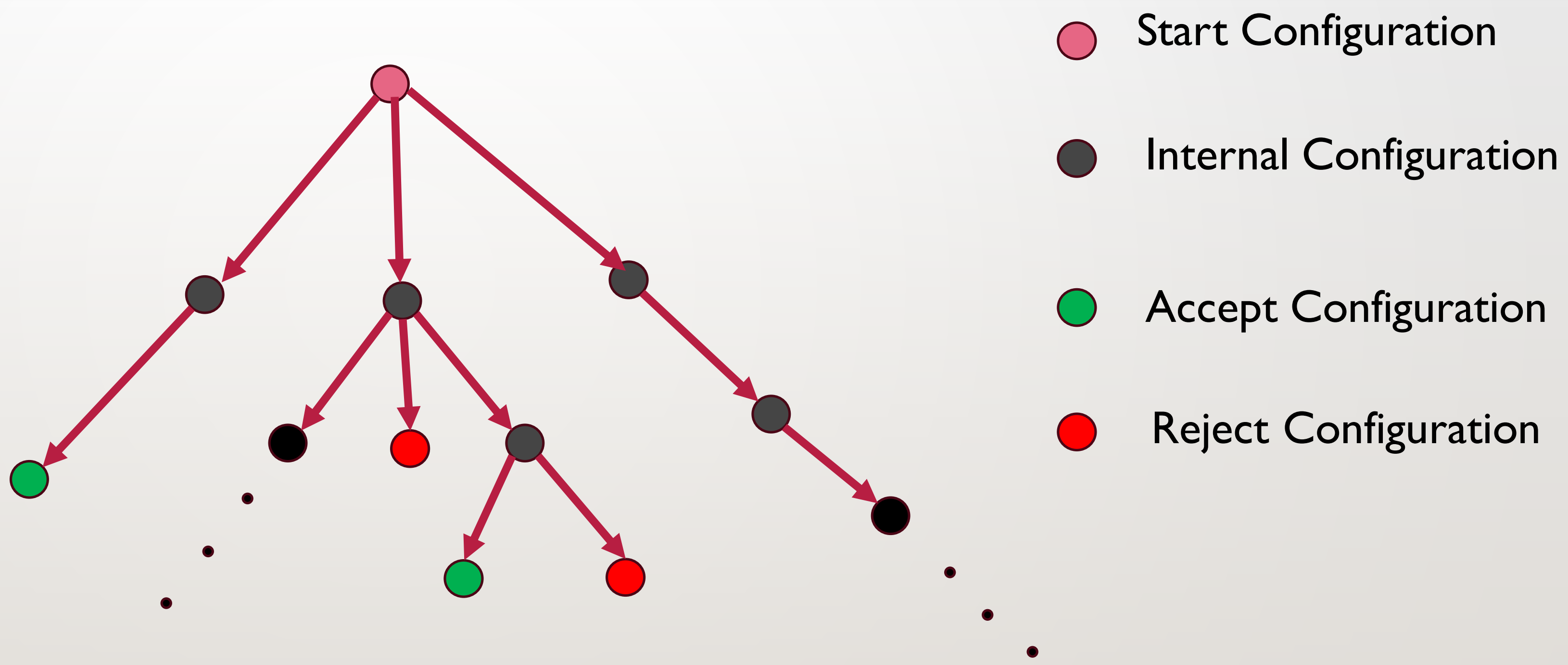
There is a Single Computing Path

TM halts on every input with either q_{accept} or q_{reject} Configuration.

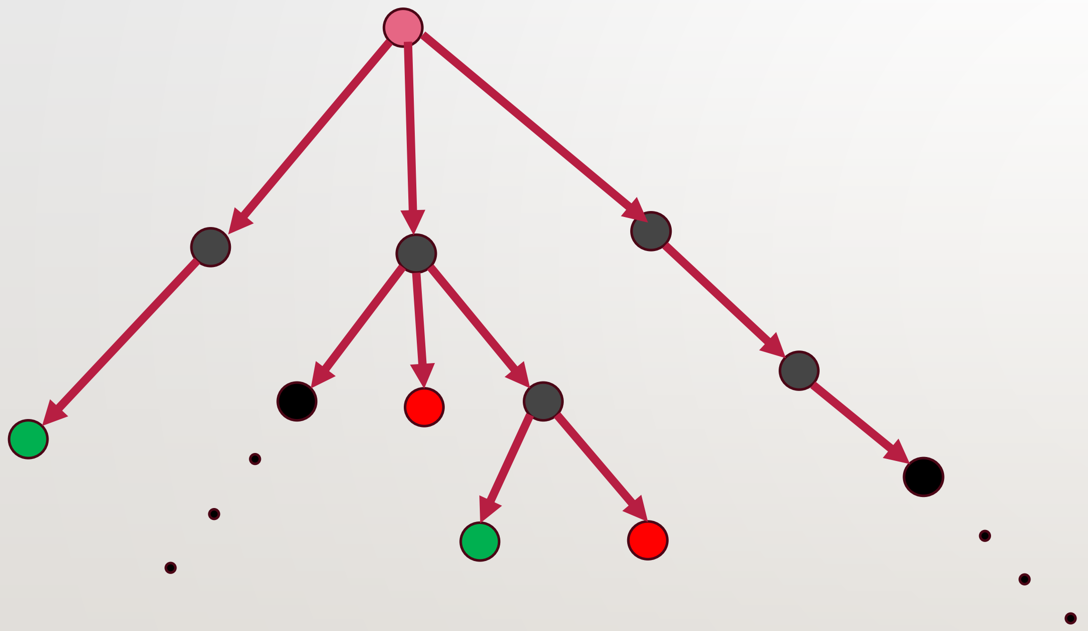
The TM we discussed is a **DETERMINISTIC** Turing Machine.



A Non-Deterministic TM



A Non-Deterministic TM

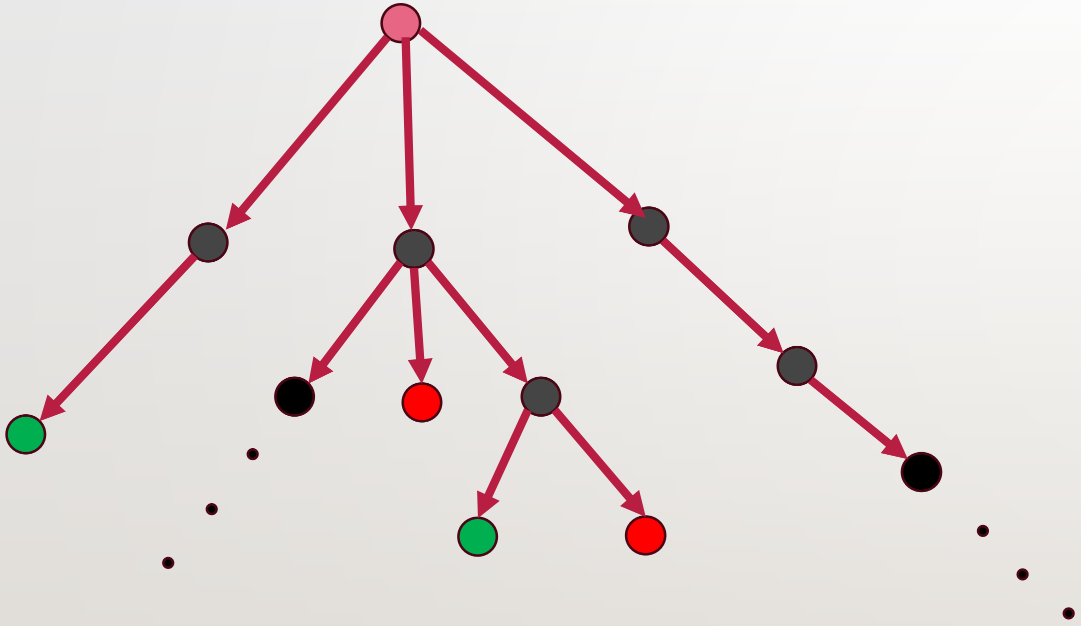


Multiple Computational Paths

Each path may lead to either q_{accept} or q_{reject} Configuration

Many of the computing paths may not be terminating

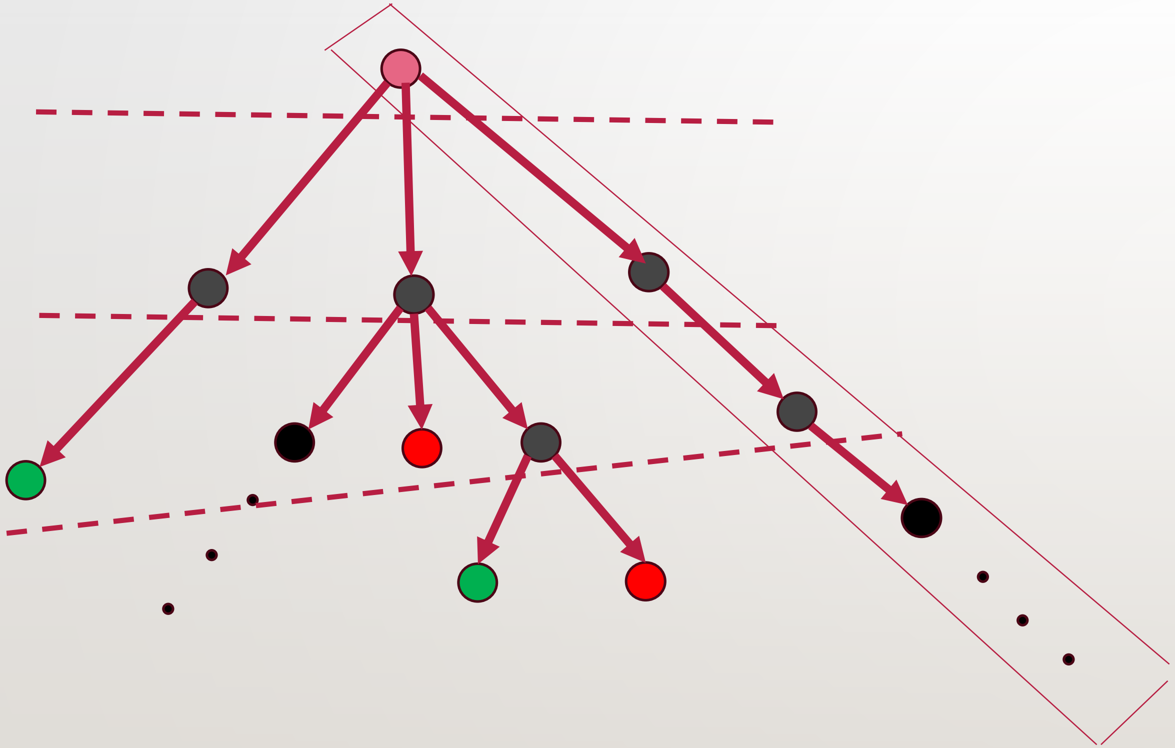
A Non-Deterministic TM



A string is accepted by a NDTM if any of the computing path reaches a q_{accept} configuration

we move from a uniquely determined sequence of computation steps to several possible sequences.

A Non-Deterministic TM



An NDTM has a magical parallelism

We can traverse the tree structure either using BFS or DFS method.

A Non-Deterministic TM

Given a language $L \subseteq \sum^*$, we determine the time $t(x)$ needed by a non-deterministic machine to process input $x \in \sum^*$ in the following way.

1. If $x \in L$ then $t(x)$ is defined as the length of the shortest path in the computation tree accepting x.
2. If $x \notin L$, the value of $t(x)$ is defined as the length of the shortest path in the computation tree.

A Non-Deterministic TM

Time $t(n)$ needed by T for processing input $x \in \sum^*$ with the length $|x| = n \in N$ is defined as the maximum of all finite $t(x)$ for $x \in \sum^*$ with $|x| = n$.

CLASS 'NP'

Non-Deterministic Polynomial Time

$$NTIME(t(n)) = \{L: L \text{ is a decided by a } NDTM \text{ in } O(t(n))\}$$

$$NP = \bigcup_{k=1}^{\infty} NTIME(n^k)$$

It is also known as given and verify model

Example: SUBSET-SUM

$$SUBSET - SUM =$$

$$\{< s, t >: s = \{x_1, x_2, \dots, x_n\} \; \exists \, t \; \subseteq \{1, 2, 3, \dots, k\} \; such \; that \sum_{i \in t} x_i = t$$

Example: SUBSET-SUM

$$SUBSET - SUM =$$

$$\{< s, t >: s = \{x_1, x_2, \ldots, x_n\} \exists\, t \subseteq \{1, 2, 3, \ldots, k\}\ such\ that \sum_{i \in t} x_i = t$$