

Turing Machine II

Zeyu Mi

2019-12-2



Adapted From:

IALC 8.3.1, 8.3.2, 8.4.1, 8.4.2, 8.6.1, 8.6.2

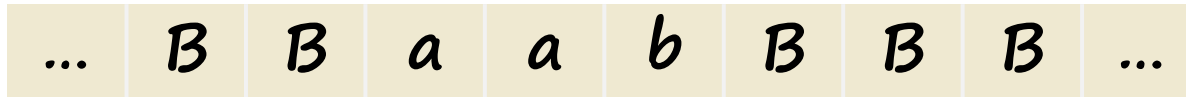
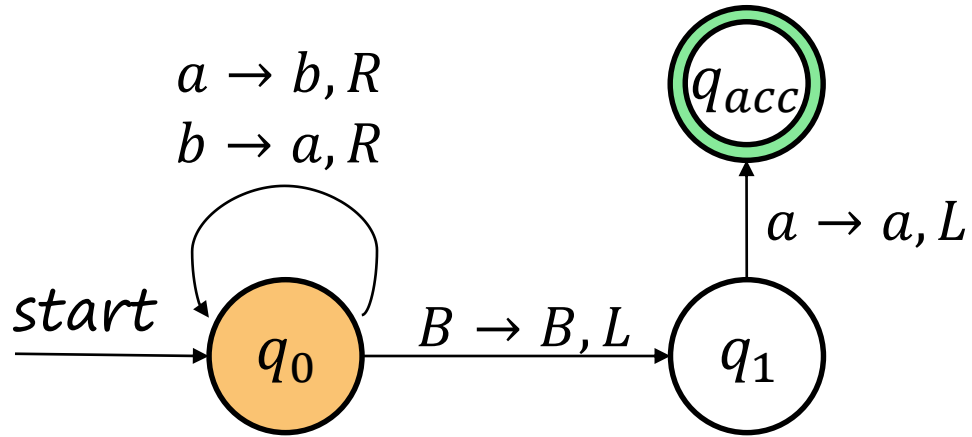
Review

- Computers are complex.
- Automata is a mathematical model of a computing device
 - Clean and simple
 - DFA: Five tuples $(Q, \Sigma, \delta, q_0, F)$
 - Turing Machine: Seven tuples, $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Review

- A **Turing Machine** consists of:
 - Q : A finite set of states
 - Σ : A finite set of input symbols
 - Γ : The complete set of tape symbols. $\Sigma \subset \Gamma$
 - δ : A transition function. $Q \times \Sigma \rightarrow Q \times \Gamma \times \{L, R\}$
 - q_0 : The start state. $q_0 \in Q$
 - B : The blank symbol. $B \in \Gamma \wedge B \notin \Sigma$.
 - F : A set of final or accepting states. $F \subseteq Q$

Review



The Language of a TM

- Similar to DFA, the language of a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is set of strings that it accepts, or:

$$L(M) = \{w | w \in \Sigma^* \wedge p \in F \wedge q_0 w \vdash^* \alpha p \beta\}$$

- For a certain language L , if there exists a TM M such that $L = L(M)$, then we call L is a **recursively enumerable language(递归可枚举语言)**, or **RE**.

Example: TM for $\{0^n 1^n\}$

- Count the number of 0 and the number of 1, and compare them.
 - But we have no idea how to store the number and compare them.
- Or when we see a 0, we try to find an 1, and repeat the process until we count every symbol.

Example: TM for $\{0^n 1^n\}$

- Our solution:
 - When we see a **0** at the beginning, we change this **0** to **X** to indicate we have counted it, and try to find an **1** by moving tape head to right.
 - When we find an **1**, we change this **1** to **Y**, and turn back to find another **0** at the beginning (right after an **X**).
 - Repeat the above process when there are only **X** and **Y** left on the tape, then we accept the input. Otherwise we reject the input.

Example: TM for $\{0^n 1^n\}$

We've found a *0*



Example: TM for $\{0^n 1^n\}$

Change it to X



Example: TM for $\{0^n 1^n\}$

Move right to find an **1**



Example: TM for $\{0^n 1^n\}$

We've found an **1**



Example: TM for $\{0^n 1^n\}$

Change it to **Y**



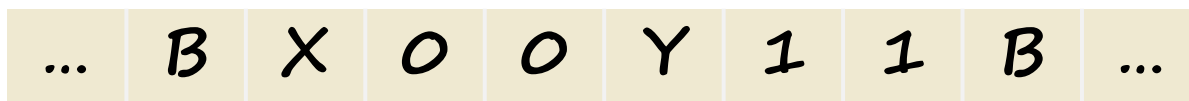
Example: TM for $\{0^n 1^n\}$

Turn back to find *o* at the beginning



Example: TM for $\{0^n 1^n\}$

Stop move left when we meet X.



Example: TM for $\{0^n 1^n\}$

Take a right move. We found another *0*.



Example: TM for $\{0^n 1^n\}$

Change it to **X** and continue.



Example: TM for $\{0^n 1^n\}$

Another 1.



Example: TM for $\{0^n 1^n\}$

Change it to Y and turn back again.



Example: TM for $\{0^n 1^n\}$

Meet **X**, stop moving left.



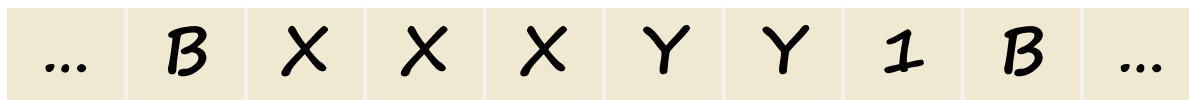
Example: TM for $\{0^n 1^n\}$

Another *O*.



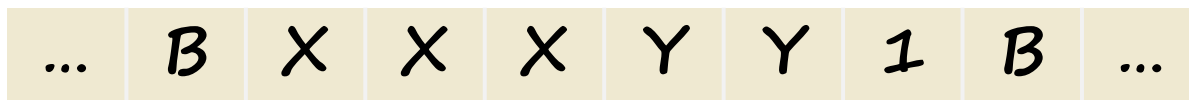
Example: TM for $\{0^n 1^n\}$

Change it to X.



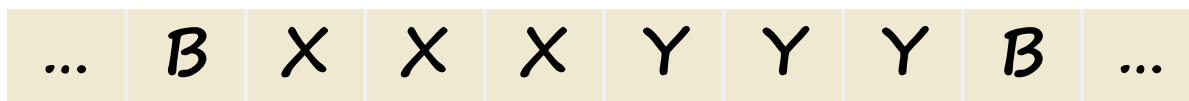
Example: TM for $\{0^n 1^n\}$

Another 1.



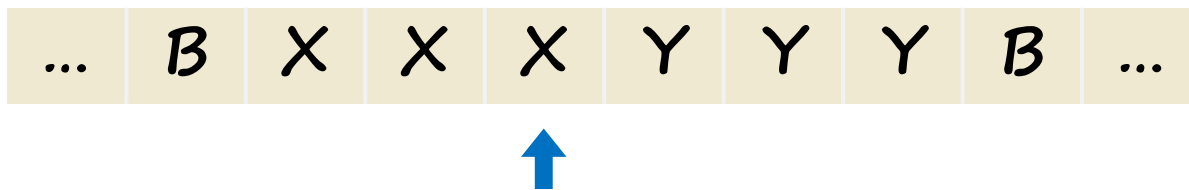
Example: TM for $\{0^n 1^n\}$

Change it and turn back again.



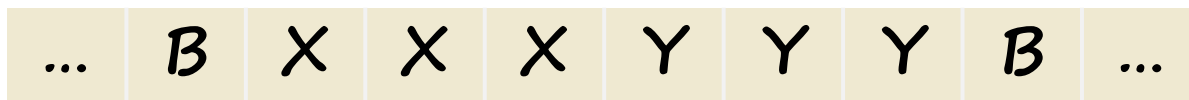
Example: TM for $\{0^n 1^n\}$

Meet **X**, take a right move.



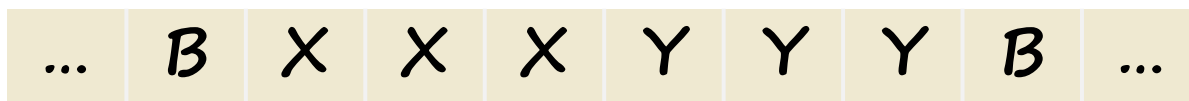
Example: TM for $\{0^n 1^n\}$

There's no **O** left. We need to check if there's only **X** and **Y** left on the tape.



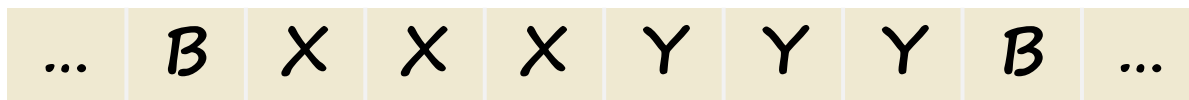
Example: TM for $\{0^n 1^n\}$

We only need to check towards the right direction as we know left if all $\times$.



Example: TM for $\{0^n 1^n\}$

We stop and accept the result until we meet a **B**.



Example: TM for $\{0^n 1^n\}$

	0	1	X	Y	B
$\rightarrow q_0$	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
$* q_4$	—	—	—	—	—

Example: TM for $\{0^n 1^n\}$

1. Find a 0, change it to X and move right

2. Move right to find an 1

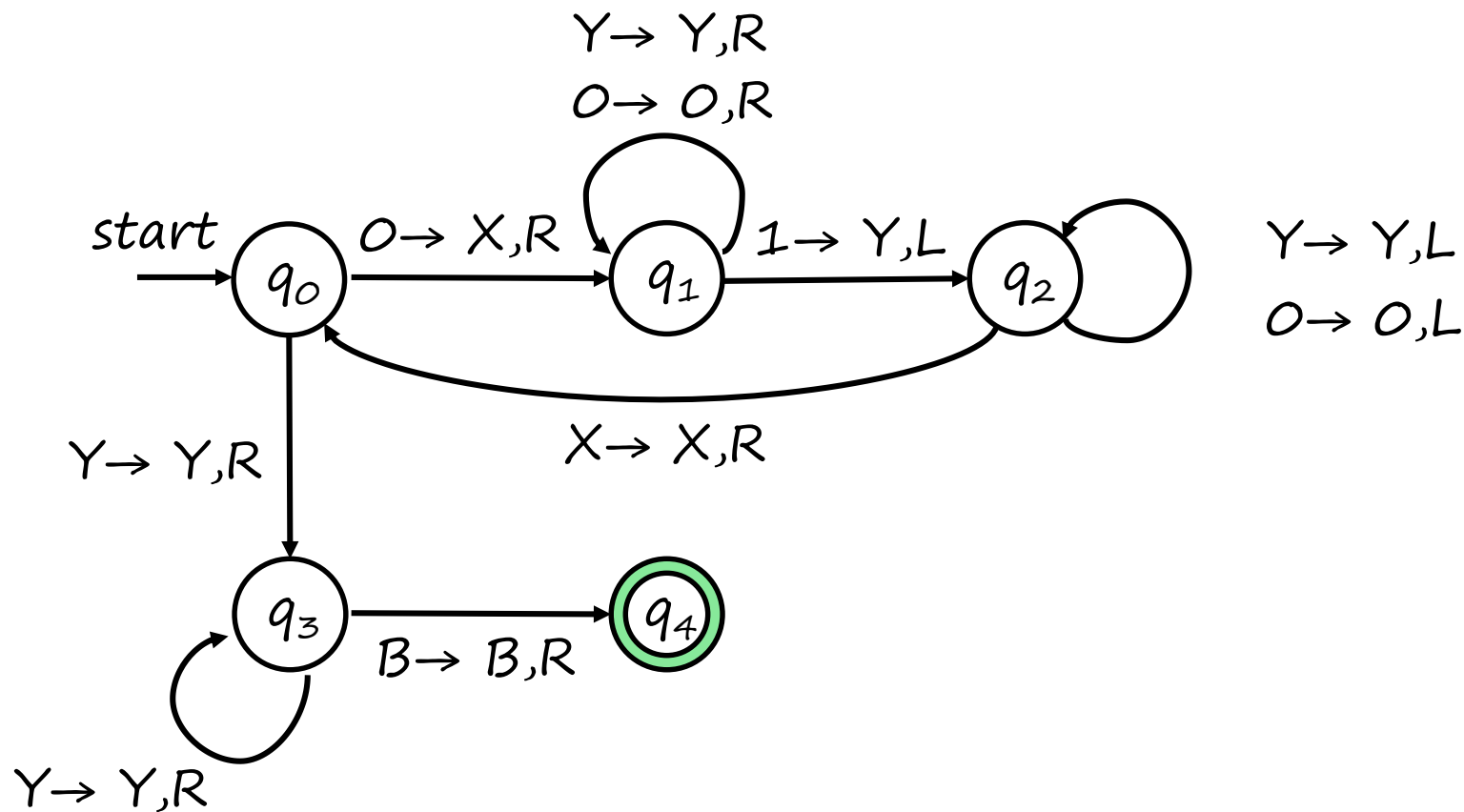
4. No 0 left, check if all symbol are Y

	0	1	X	Y	B
→ q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
* q_4	—	—	—	—	—

3. Move left to find an X

5. Accept when we meet a blank.

Example: TM for $\{0^n 1^n\}$



How we investigate computability?

2.1 Discussion on Problems

Part1. Intro & Set theory(I) : Basics & Formal Language.

Part2. Set Theory(II): Axiom system & Cardinality.

Part3. Capture Structures: Binary Relation & Function

2.2 Discussion on Computation

Part1. Turing Machine Basics.

Part2. Variants of Turing Machine. Church-Turing Thesis.

2.3 Discussion on computability

Part1. The Language of Turing Machine. R & RE

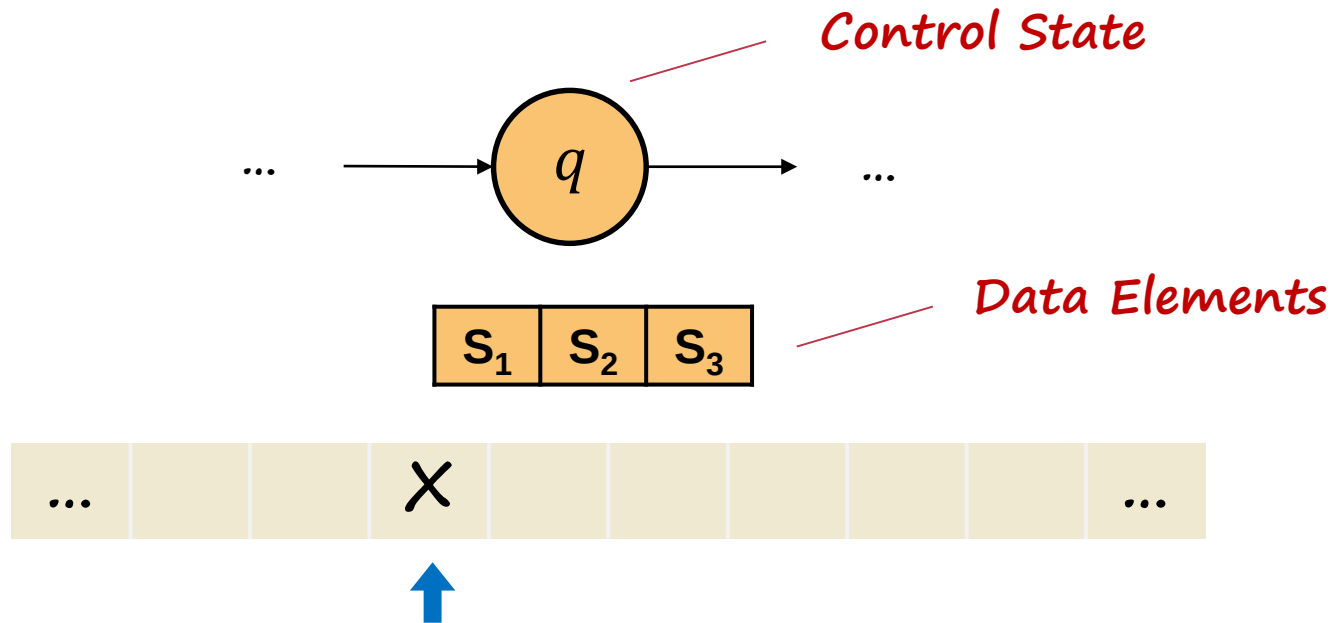
Part2. Undecidability.

Remember the Execution State

- **It's common in our daily programming that using some variables to remember the execution state.**
 - A certain char we've seen in processing a string.
 - The biggest number we've found in finding the maximum.
 - Did we end a loop normally or by break.
 - Or many states combined together.

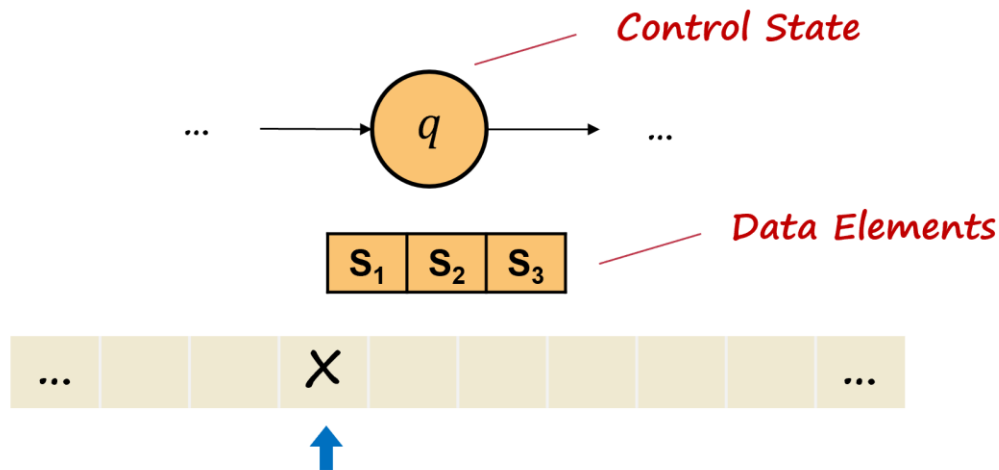
Can we simply apply this programming techniques into TM?

Remember the Execution State



TM with storage

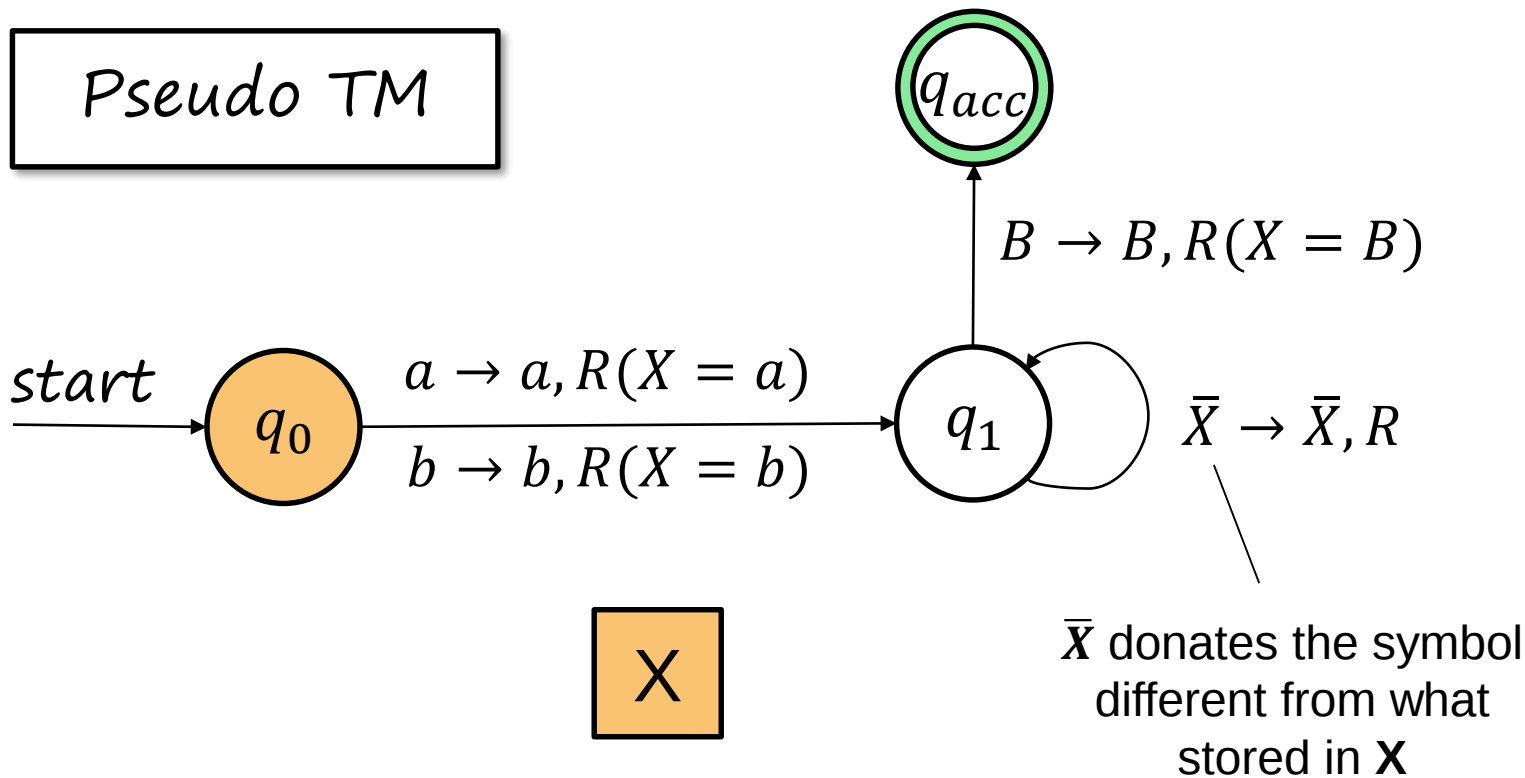
- We can use the state to hold a **finite** amount of data.
- The technique requires no extension to the TM model, we just think the state as tuple now.
 - $[q, s_1, s_2, s_3]$ is the current state.
- Regarding states this way allows us to describe transitions in a more systematic way



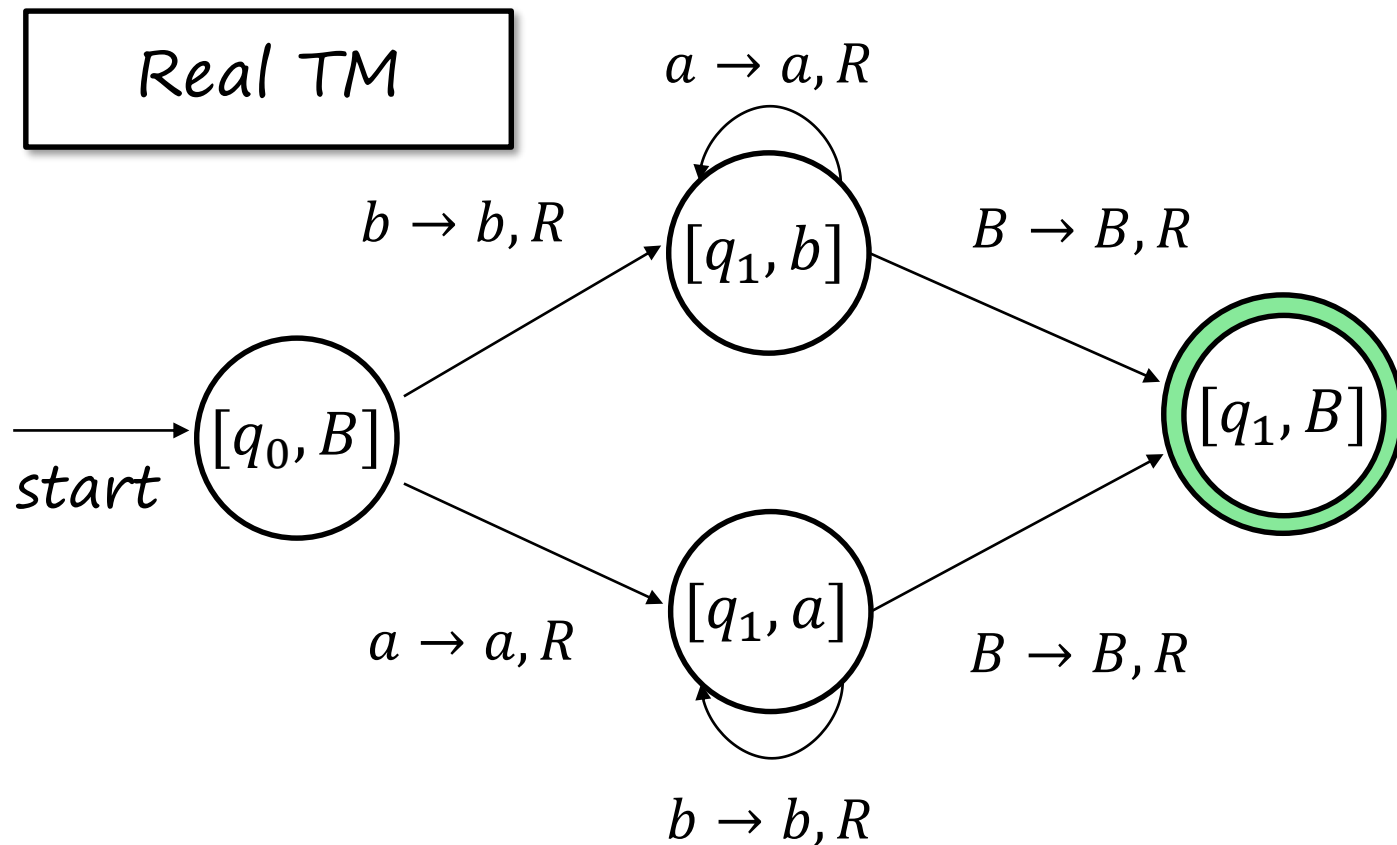
Example: $\{ab^* + ba^*\}$

- We can remember the first symbol we've seen, and check whether the remaining symbols are different from it.
 - q_0 : Read the first symbol, save it into data elements.
 - q_1 : Then scan the remaining symbols, we continue this process if every symbol is different from the symbol stored, and stop and accept when we meet a blank symbol.
- The data elements can be a, b or B , so the actual states of this TM are $\{q_0, q_1\} \times \{a, b, B\}$

Example: $\{ab^* + ba^*\}$



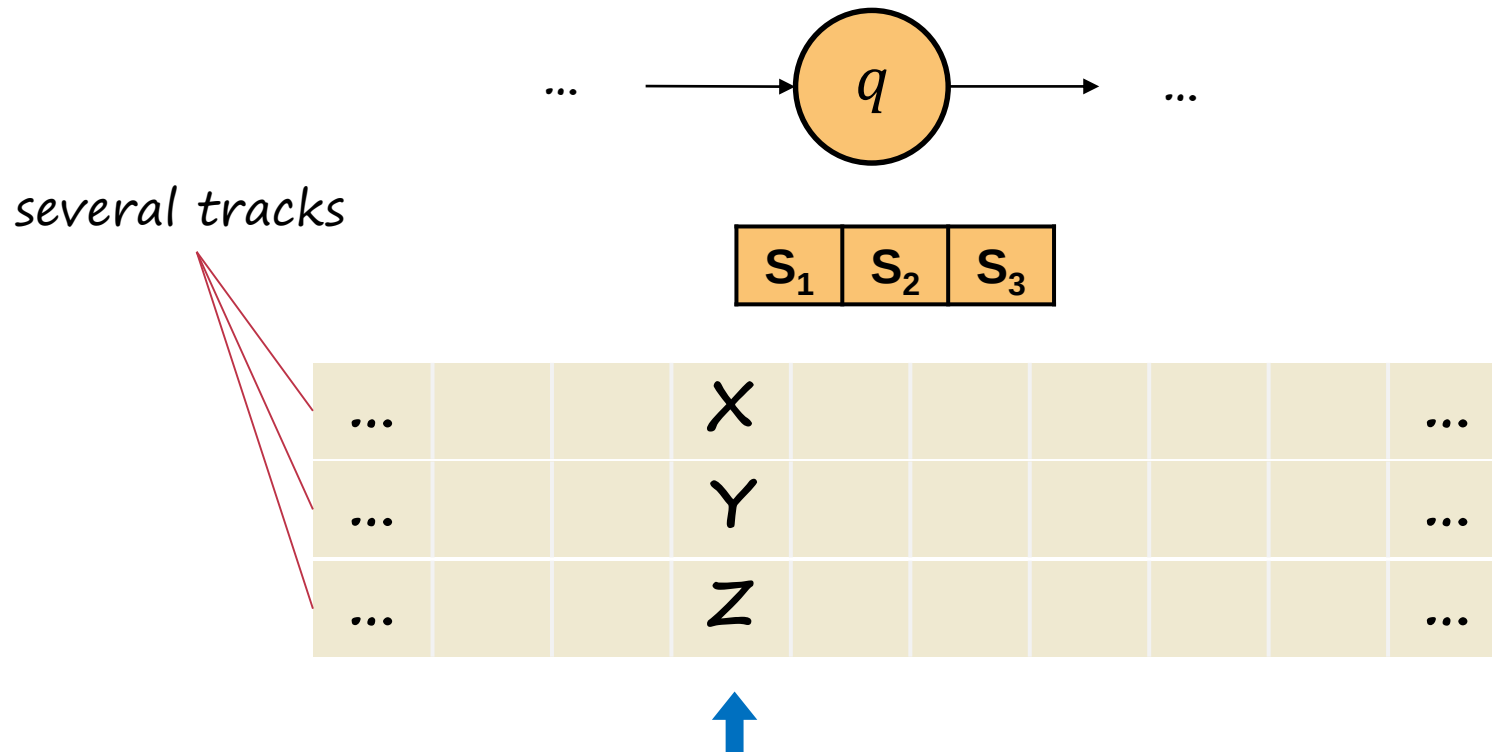
Example: $\{ab^* + ba^*\}$



Remember the Execution State

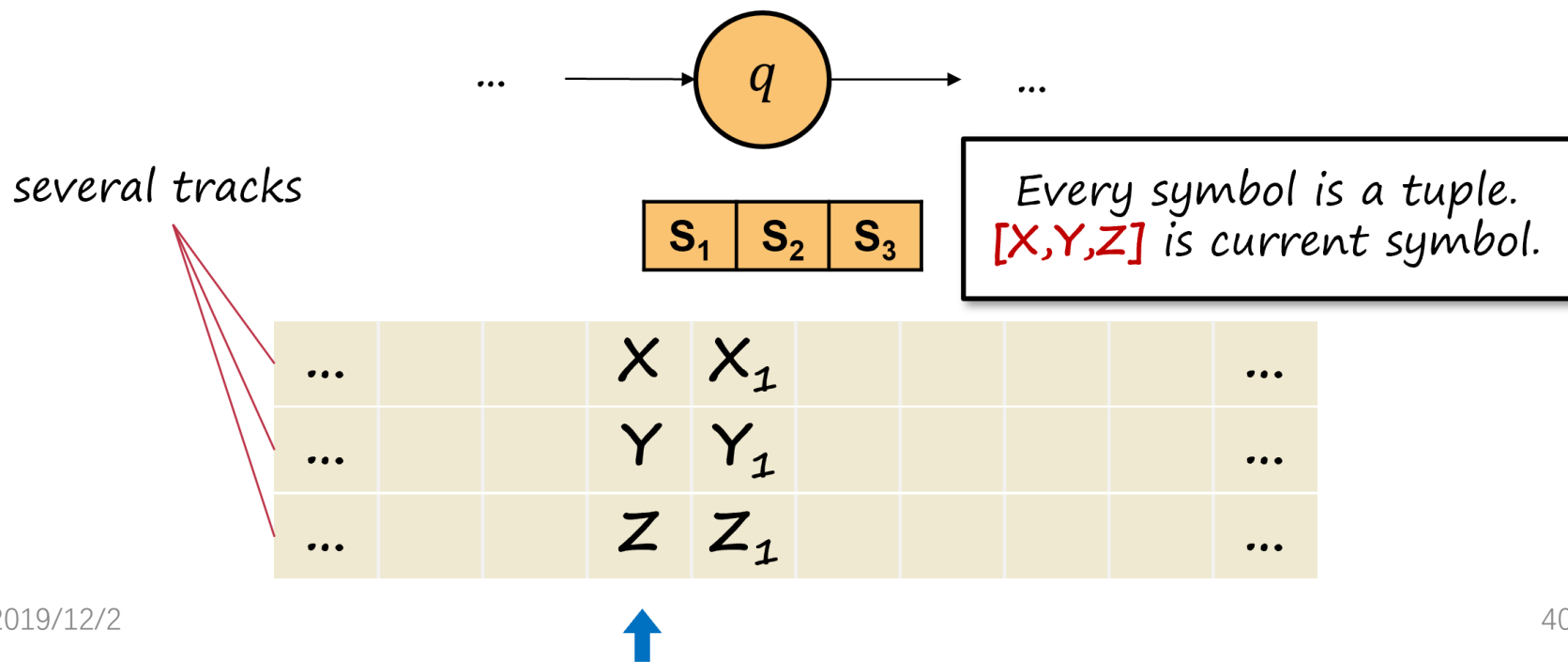
- We may need to remember the state for every single elements in an array, which means we may need to remember infinite states.
 - A bit array denotes whether an element has certain property.
- But the technique above can only hold finite many data in states

Remember the Execution State

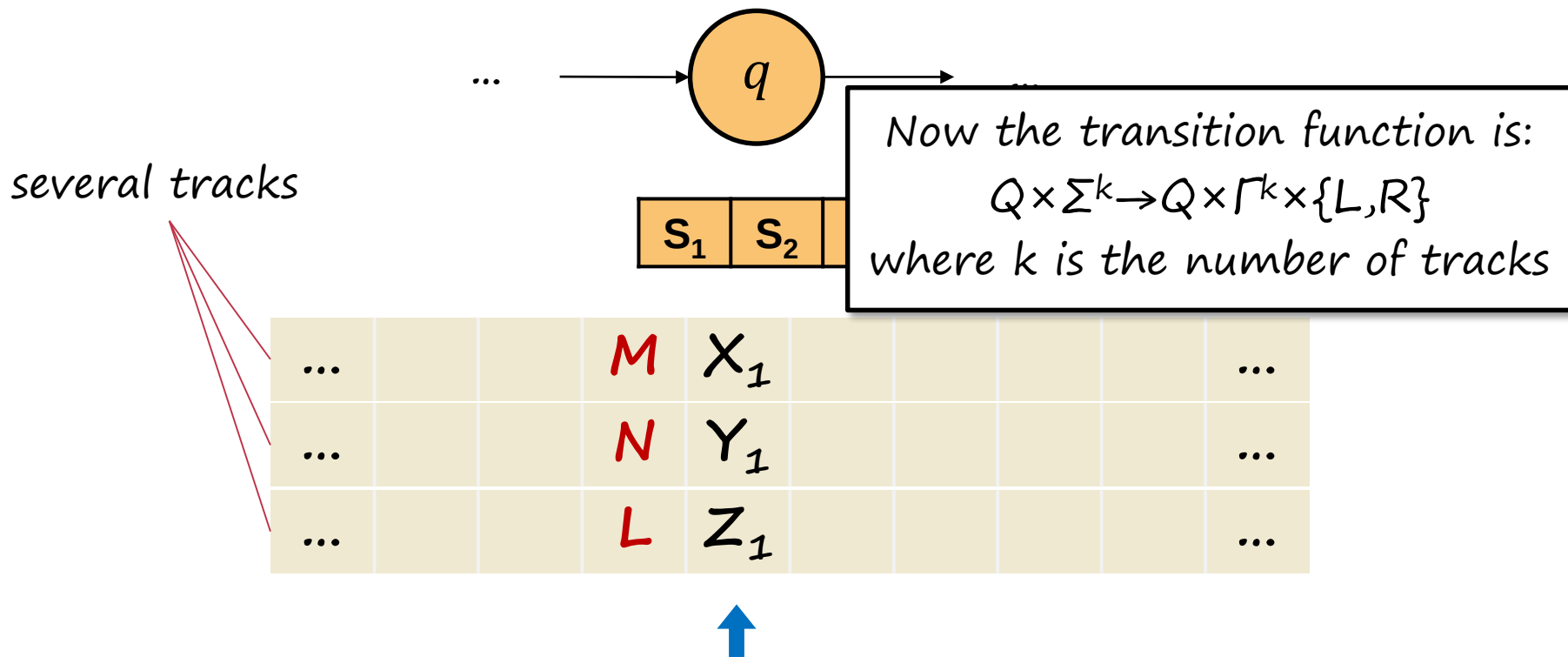


Multitrack

- We can think of the tape of a Turing machine as composed of several tracks. Each track can hold one symbol, and the tape alphabet of the TM consists of tuples.



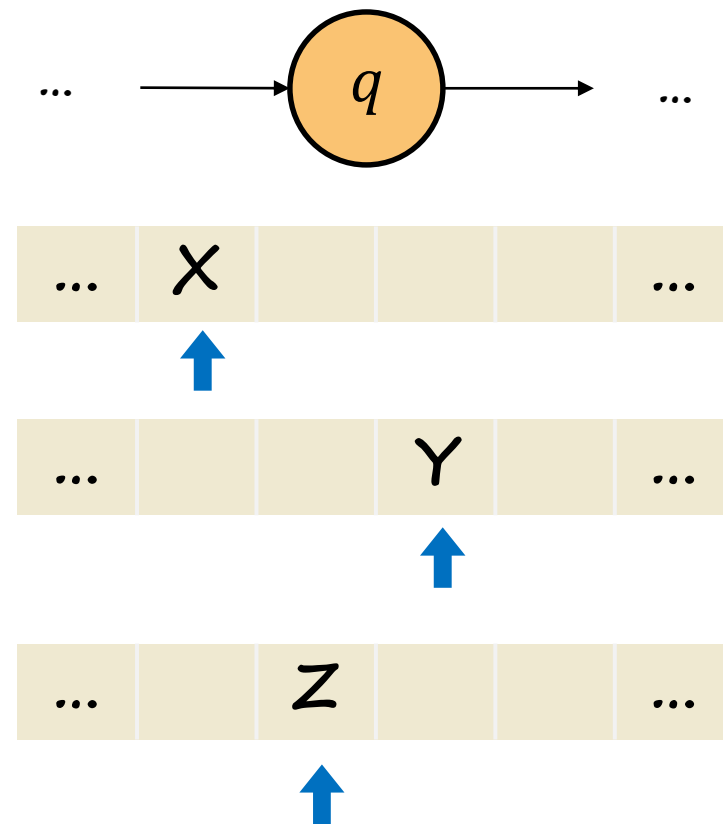
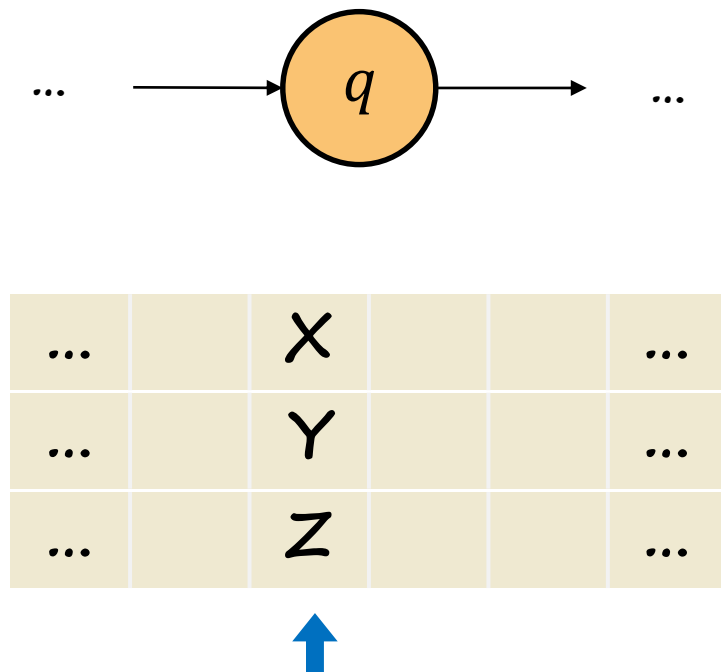
Multitrack



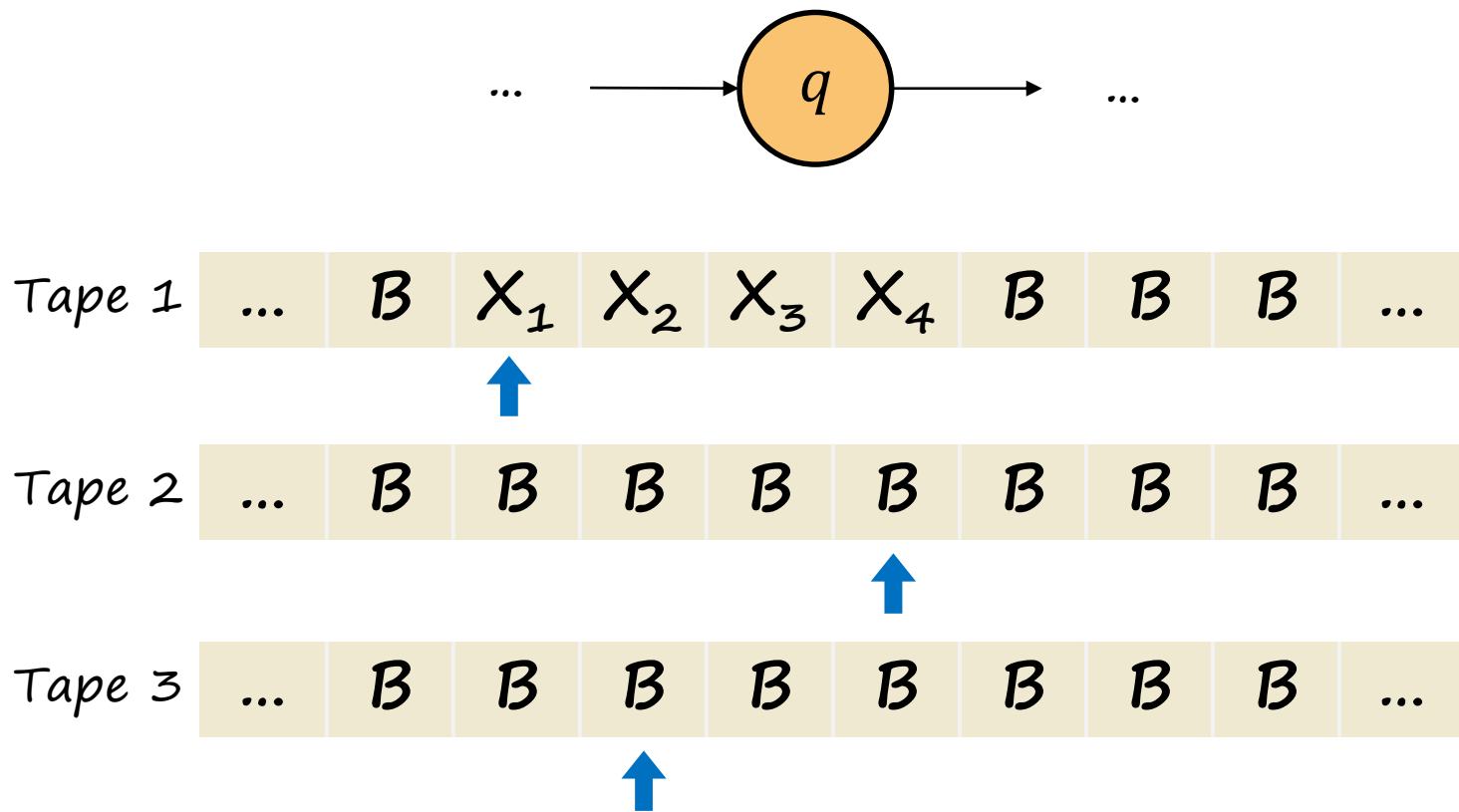
Multitrack

- Multitrack also **does not extend** what the Turing machine can do. It is simply a way to view tape symbols and to imagine that they have a useful structure.
 - Actually, we can use a single alphabet Γ' such that there's a bijection between Γ' and $\Gamma \times \Gamma \dots \times \Gamma$. So we can replace the tuple in $\Gamma \times \Gamma \dots \times \Gamma$ with a symbol in Γ' in all places.

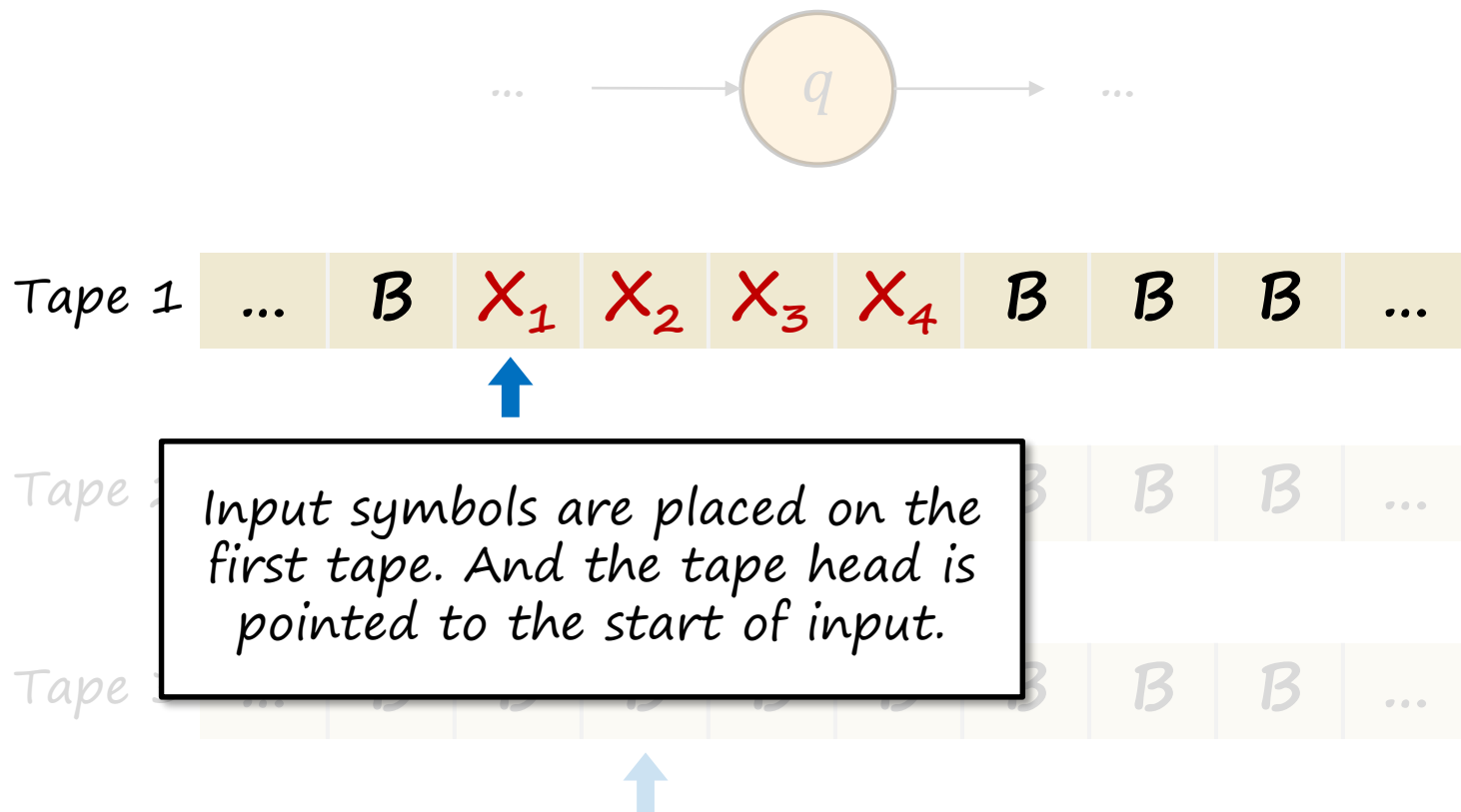
Multitrack and Multi-tape



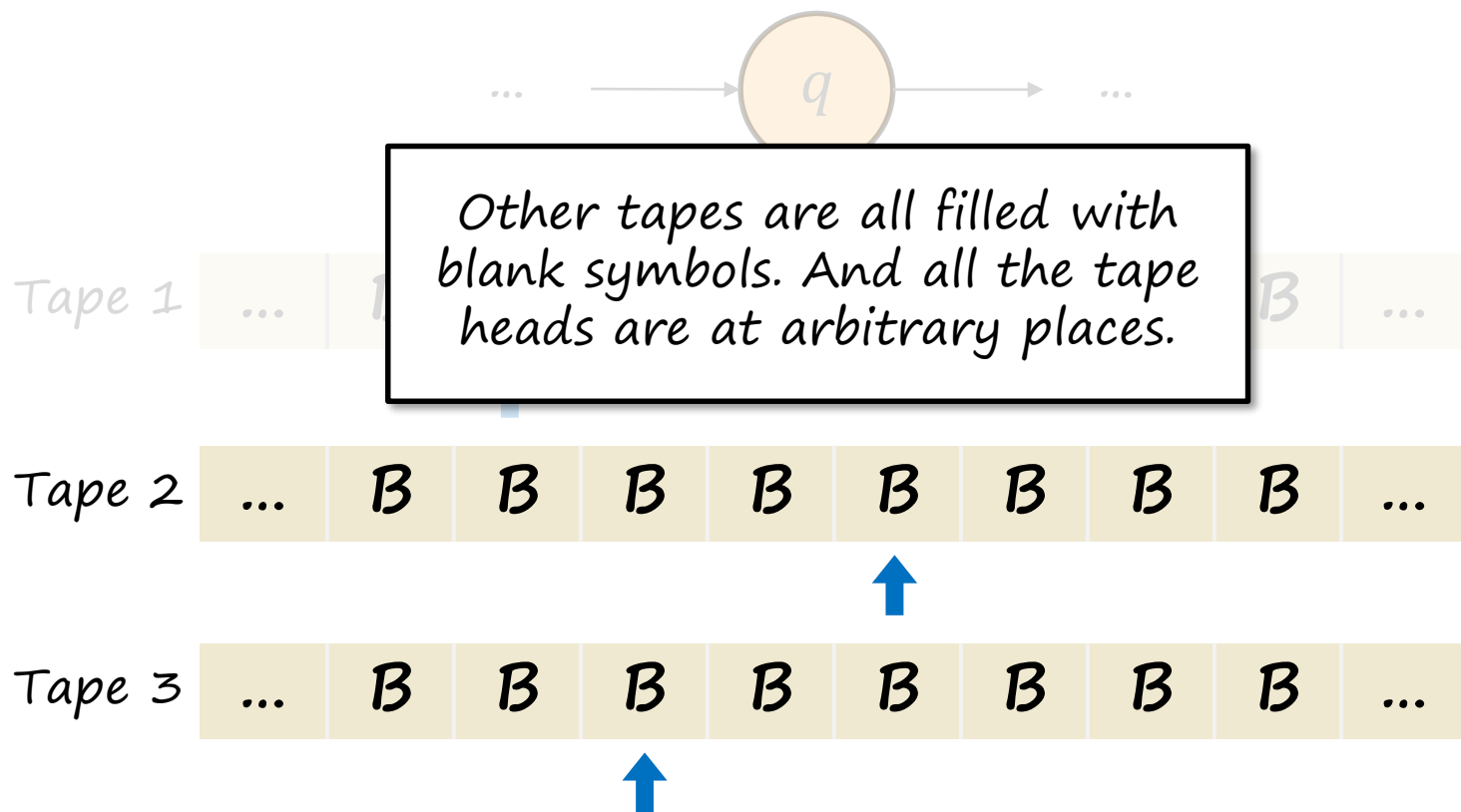
Multi-tape Turing Machine



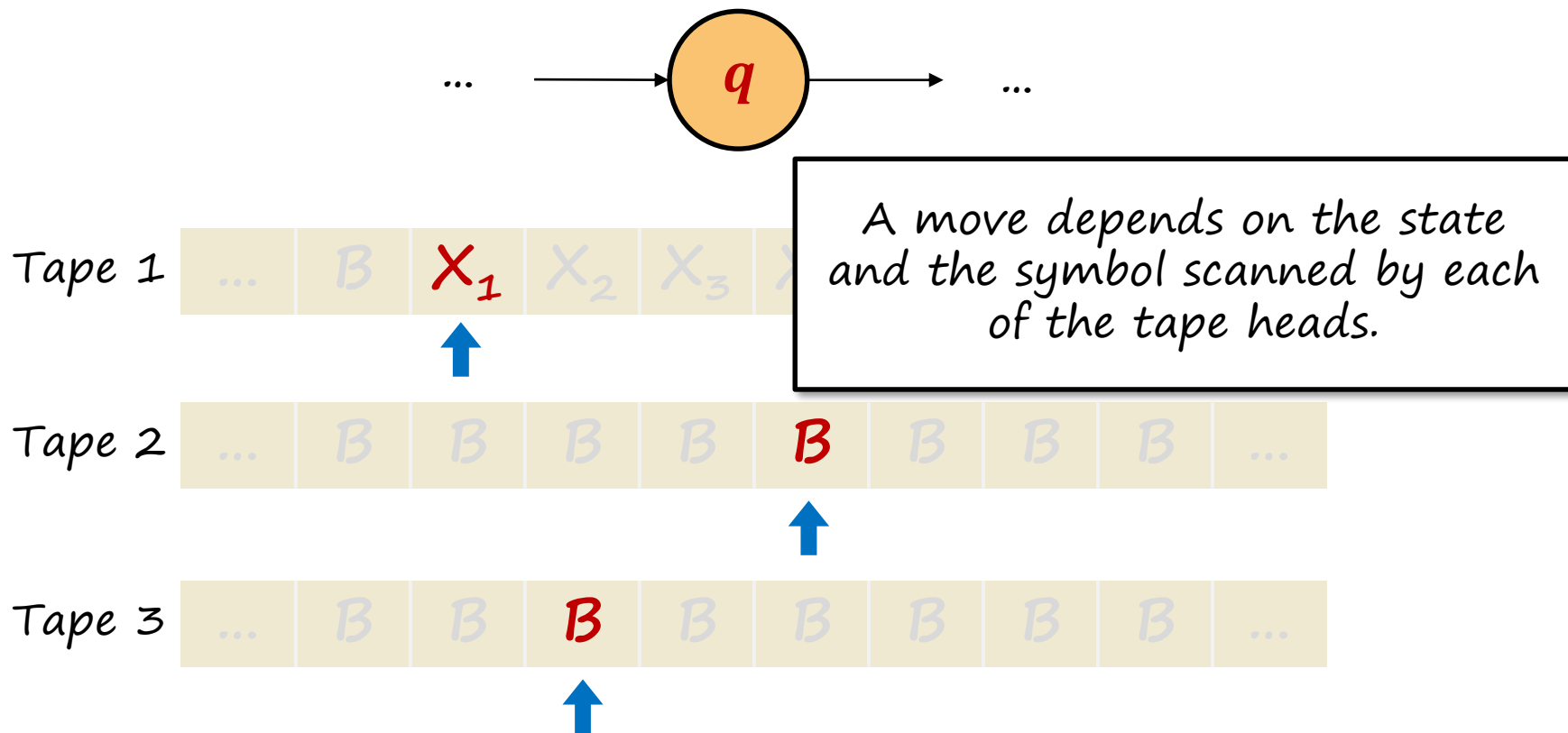
Multi-tape Turing Machine



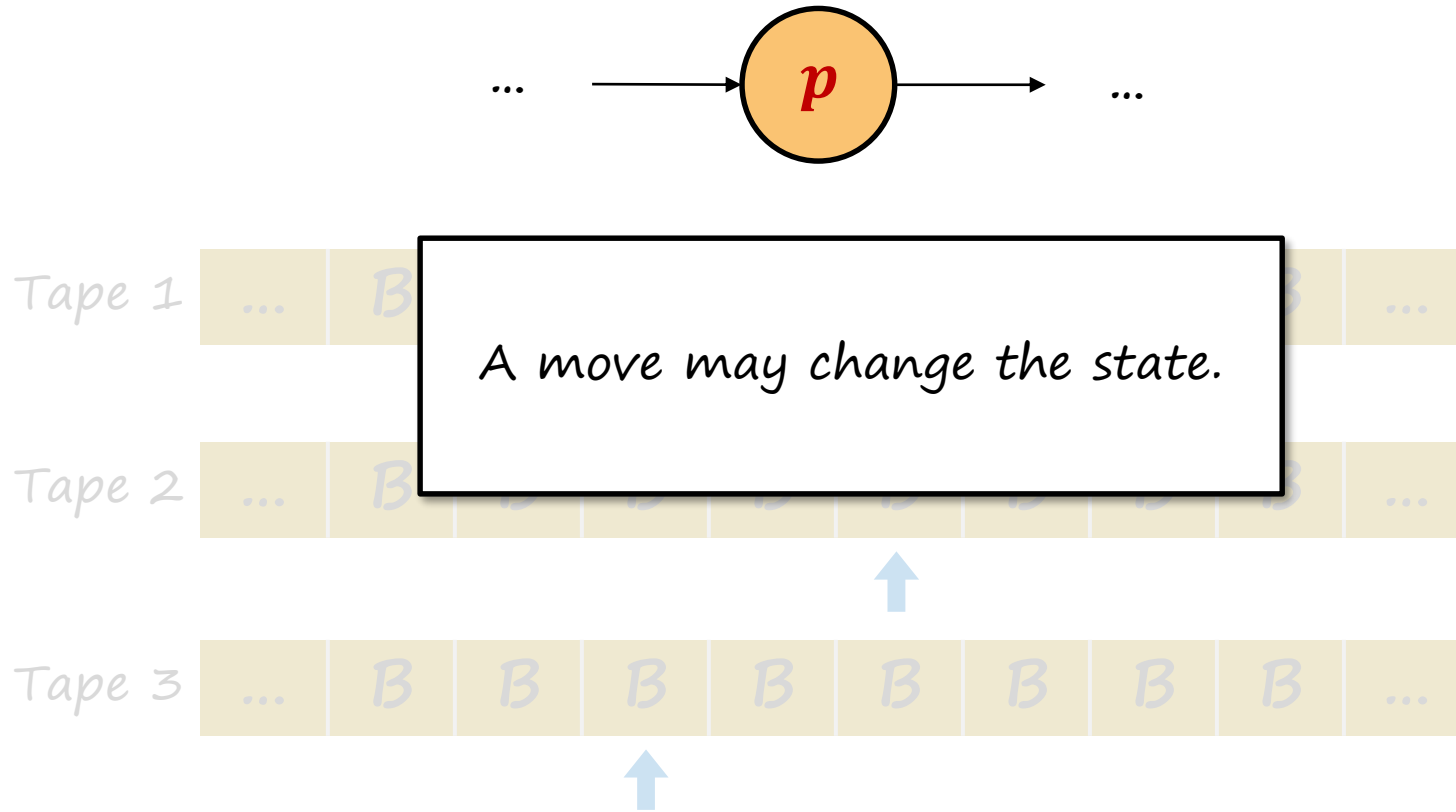
Multi-tape Turing Machine



Multi-tape Turing Machine

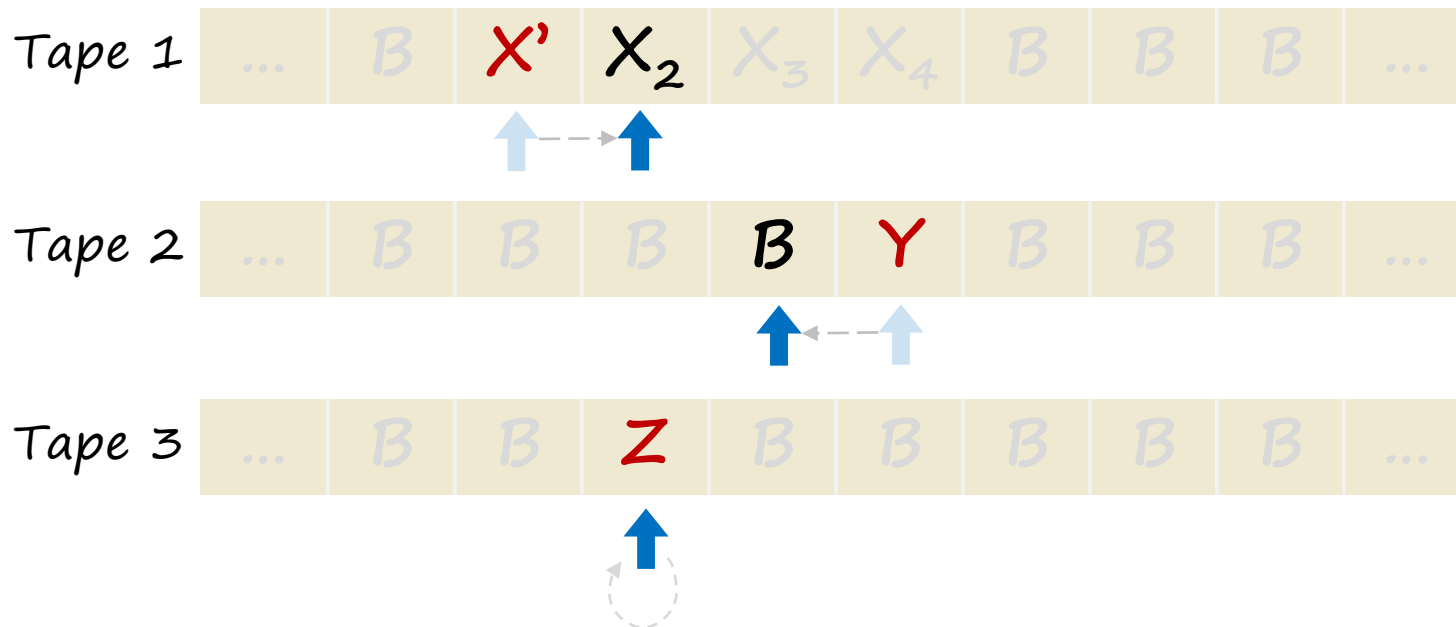


Multi-tape Turing Machine



Multi-tape Turing Machine

On each tape, a symbol may be written on the current cell, and the tape head move to left, right or stationary.



Multi-tape Turing Machine

- A ***k*-tape Turing Machine** can also be denoted as a seven tuple $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$
- But now the transition function is:

$$\delta: Q \times \Sigma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

k symbols scanned
by all tapes

k symbols written
into current cells
of all tapes

k moving directions
for all tape heads.
L, *R* and *S* denote left,
right and stationary.

Computation Capability

- For any certain type of automata (e.g. DFA, TM), the computation capability of it can be regarded as all the languages the type of automata can accept.

$$P(T) = \{L(A) | A \text{ is a } T\},$$

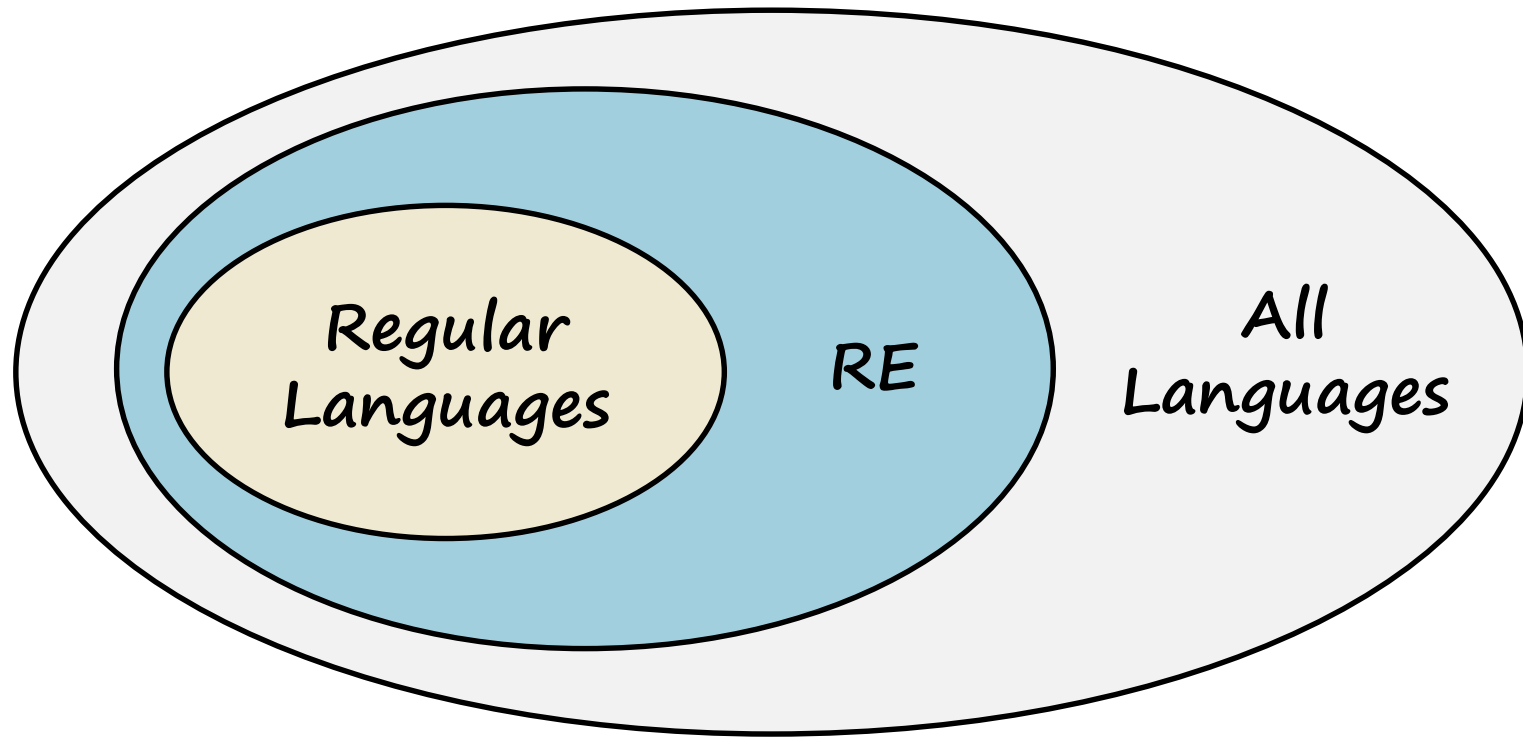
where T can be DFA , Turing Machine, etc.

- For DFA, the capability is all the regular languages. For Turing Machine, it is all the *RE* languages.
 - $P(DFA) = \{L | L \text{ is a Regular Language}\}$
 - $P(TM) = \{L | L \text{ is a RE Language}\}$

Computation Capability

- For two types of automata T_1 and T_2
 - If $P(T_1) = P(T_2)$, we say T_1 and T_2 have same capability.
 - If $P(T_1) \subset P(T_2)$, we say T_2 have greater capability than T_1 .
- DFA and TM:
 - For any language L accepted by a DFA ($L \in P(DFA)$), there's a TM M such that $L(M) = L$ ($L \in P(TM)$), so it means that $P(DFA) \subseteq P(TM)$
 - And we know there's some languages can be accepted by TM but can not be accepted by DFA (like $\{1^n 2^n\}$), so we have $P(DFA) \subset P(TM)$, which means **TM have greater computation capability than DFA**

Computation Capability



Multi-tape TM and Single-tape TM

- A multi-tape TM (**MT-TM**) can easily simulate single-tape TM (**ST-TM**), which means for any language L accepted by a single-tape TM, there always exists a multi-tape TM M such that $L(M) = L$, so we have:

$$P(ST-TM) \subseteq P(MT-TM)$$

- But whether $P(ST-TM) \subset P(MT-TM)$?

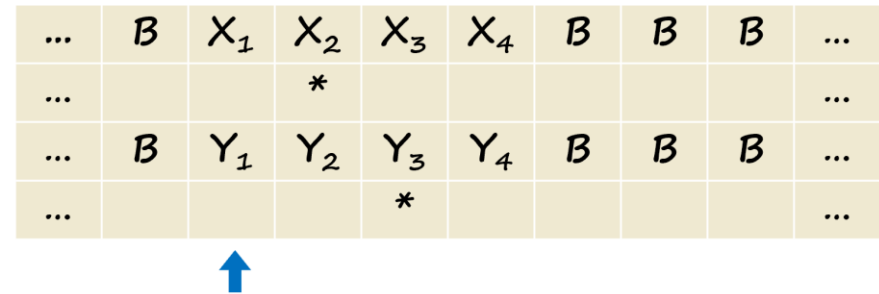
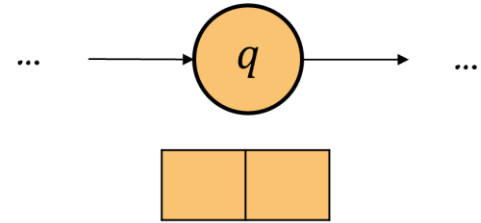
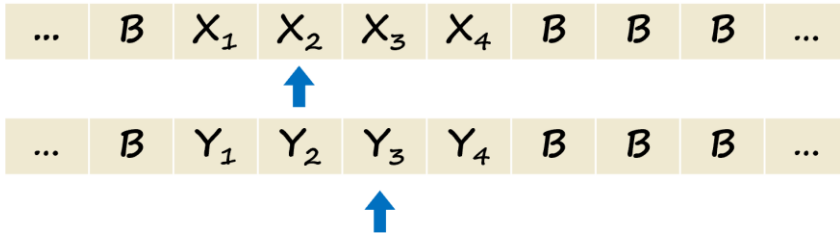
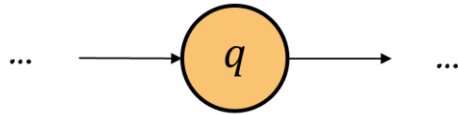
Multi-tape TM and Single-tape TM

Every language accepted by a multi-tape TM is RE, or

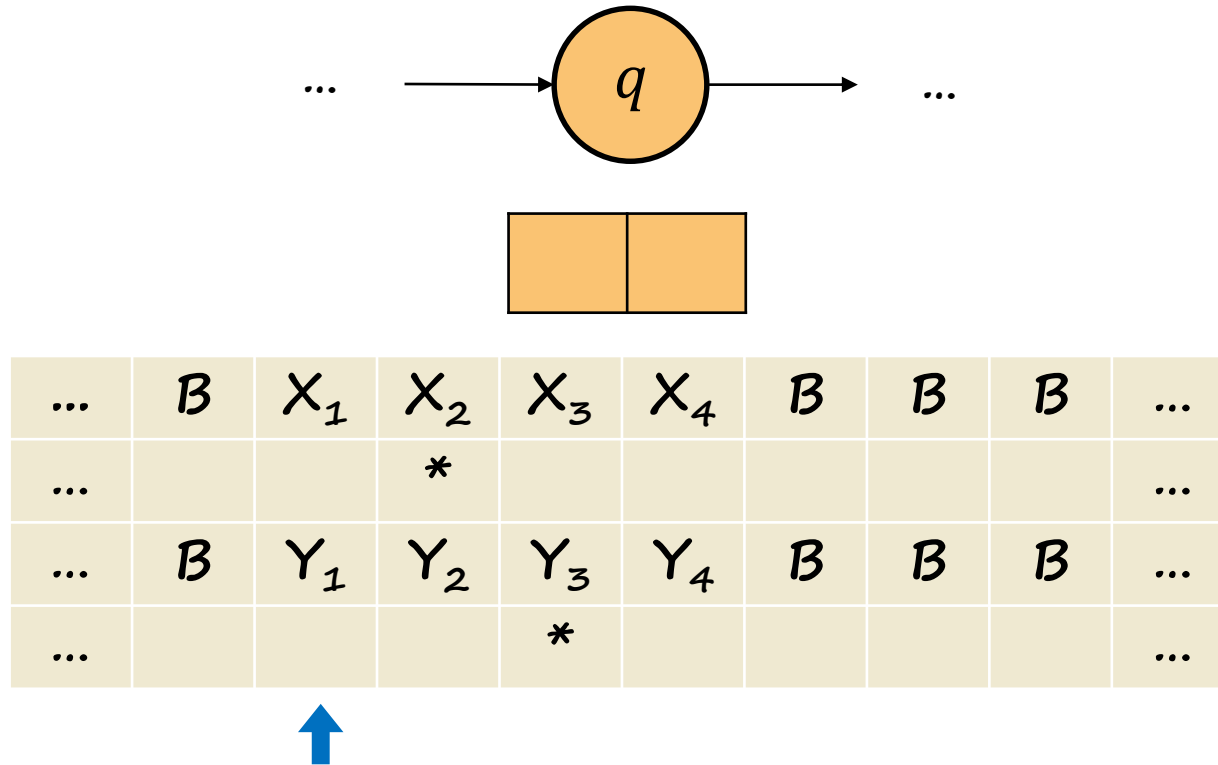
$$P(ST-TM) = P(MT-TM)$$

- For a k -tape TM, we use a single-tape TM with k extra storage and $2k$ tracks to simulate it.
 - Storage and multitrack will not extend what TM can do

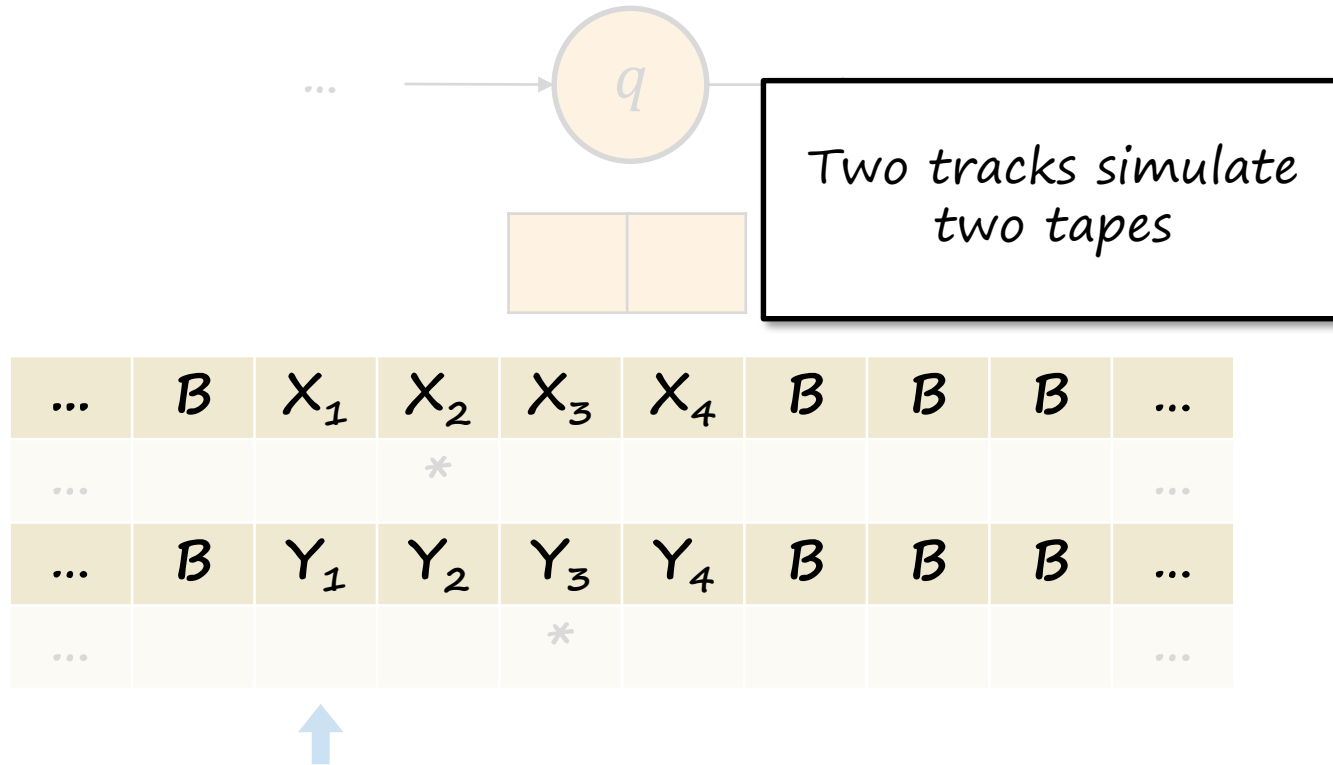
Equivalence of ST-TM and MT-TM (k=2)



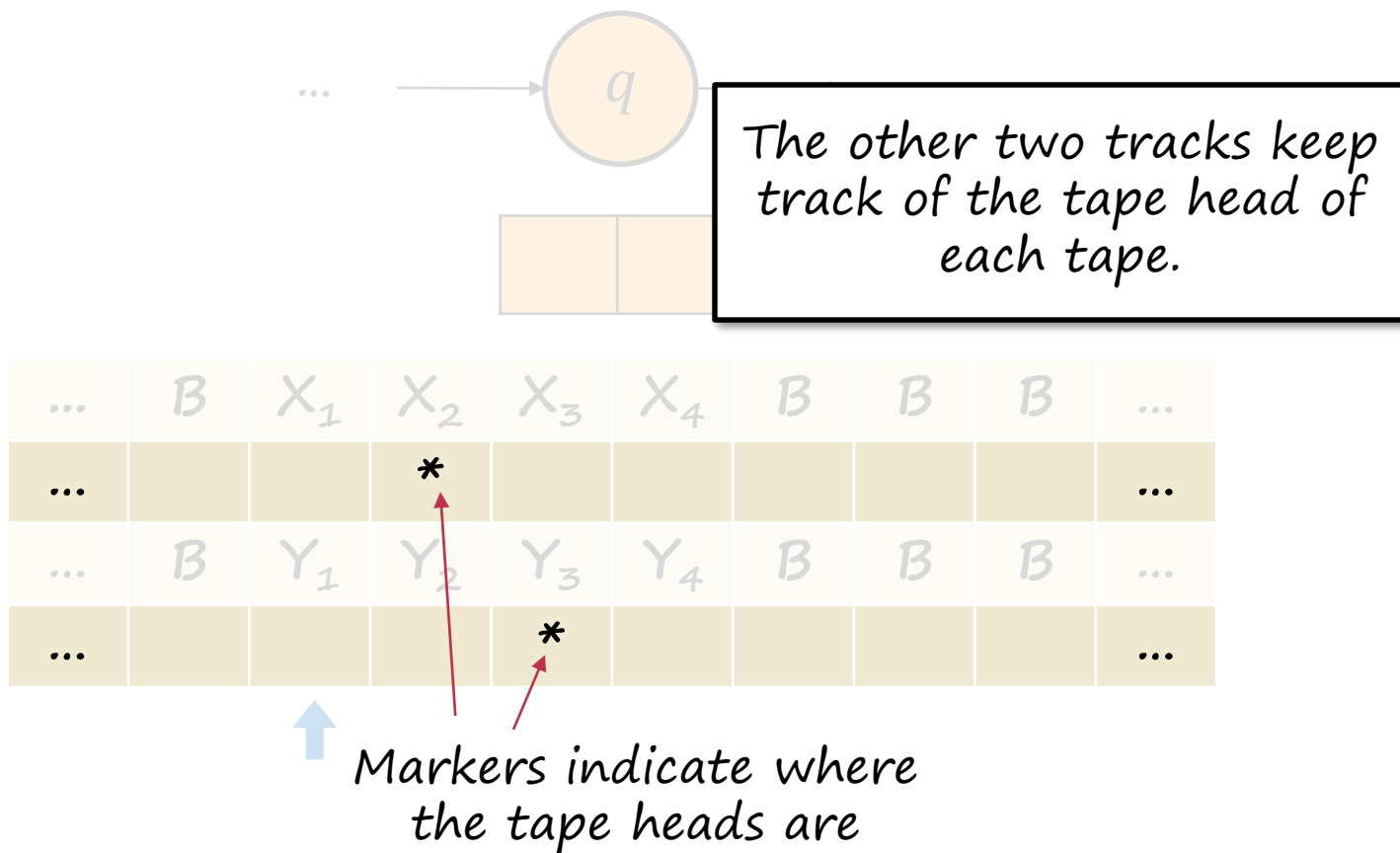
Equivalence of ST-TM and MT-TM (k=2)



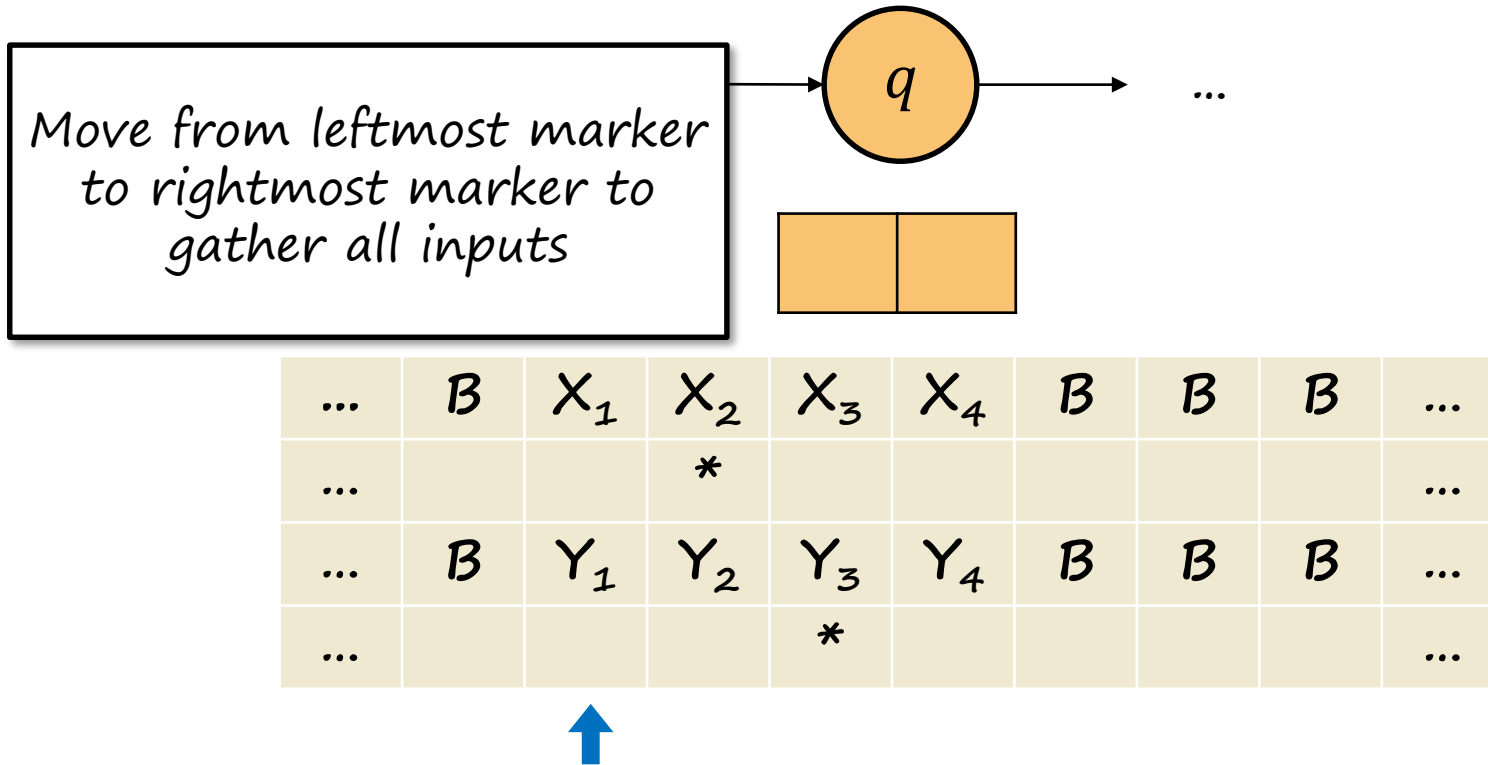
Equivalence of ST-TM and MT-TM (k=2)



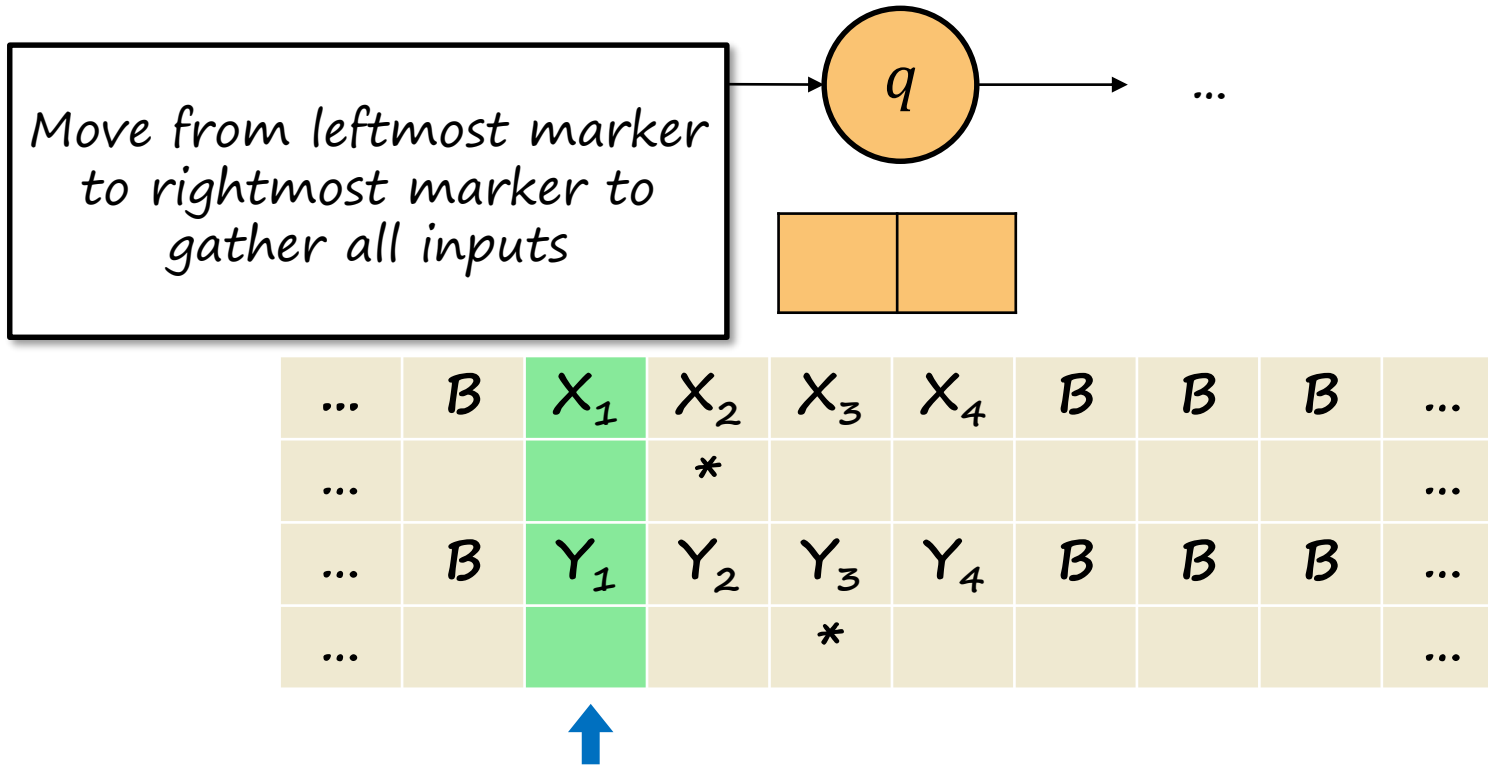
Equivalence of ST-TM and MT-TM ($k=2$)



Equivalence of ST-TM and MT-TM ($k=2$)

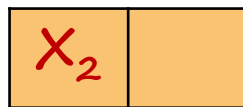
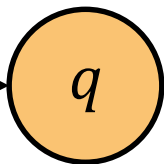


Equivalence of ST-TM and MT-TM ($k=2$)



Equivalence of ST-TM and MT-TM ($k=2$)

Find a marker, store the symbol into its state.

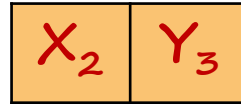
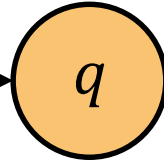


...	B	X_1	X_2	X_3	X_4	B	B	B	...
...			*						...
...	B	Y_1	Y_2	Y_3	Y_4	B	B	B	...
...				*					...



Equivalence of ST-TM and MT-TM (k=2)

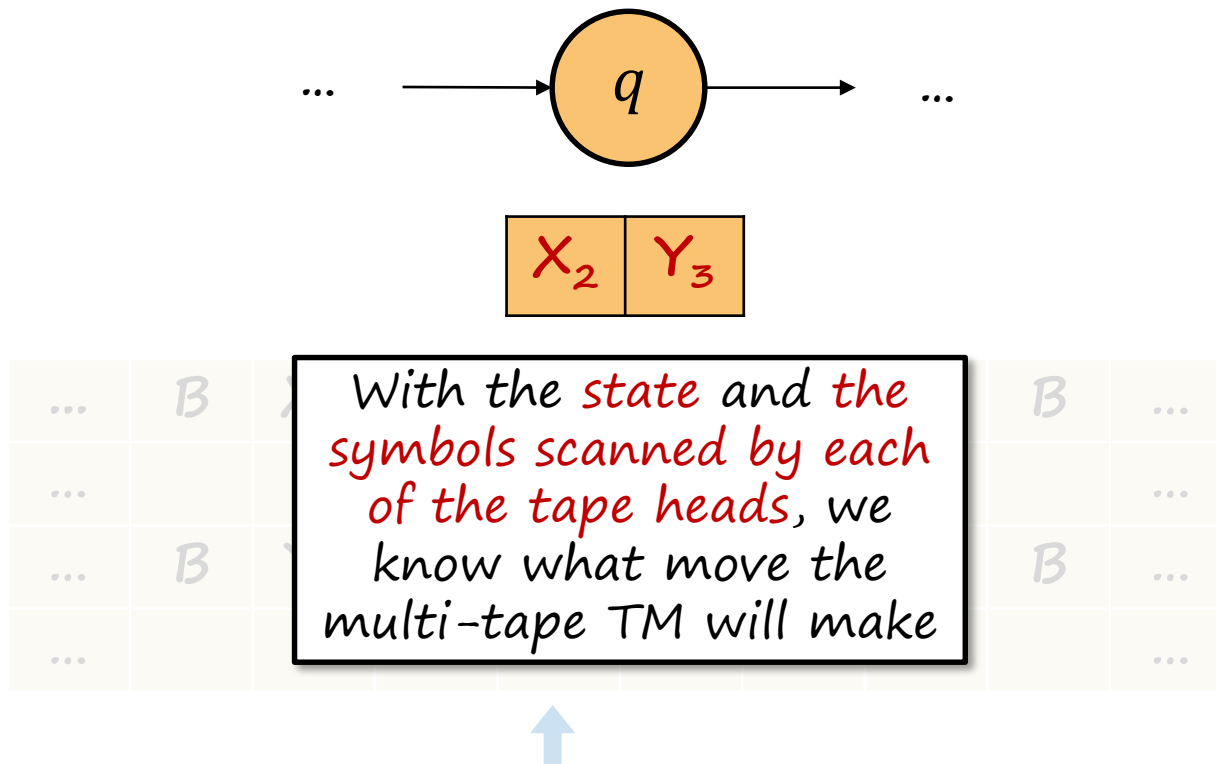
Find a marker, store the symbol into its state.



...	B	X_1	X_2	X_3	X_4	B	B	B	...
...			*						...
...	B	Y_1	Y_2	Y_3	Y_4	B	B	B	...
...				*					...



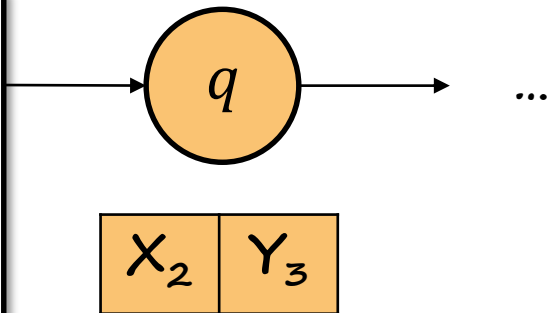
Equivalence of ST-TM and MT-TM (k=2)



Equivalence of ST-TM and MT-TM (k=2)

Take the action same as done by the k-tape TM

1. Rewrite the symbols
2. Move the tape-head
3. Change the state



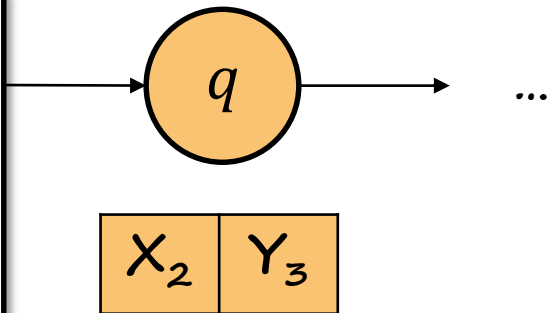
...	B	X_1	X_2	X_3	X_4	B	B	B	...
...			*						...
...	B	Y_1	Y_2	Y_3	Y_4	B	B	B	...
...				*					...



Equivalence of ST-TM and MT-TM ($k=2$)

Take the action same as done by the k -tape TM

1. Rewrite the symbols
2. Move the tape-head
3. Change the state



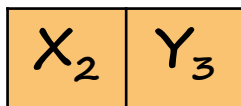
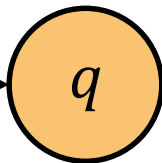
...	B	X_1	X_2	X_3	X_4	B	B	B	...
...			*						...
...	B	Y_1	Y_2	M	Y_4	B	B	B	...
...									...



Equivalence of ST-TM and MT-TM (k=2)

Take the action same as done by the k-tape TM

1. Rewrite the symbols
2. Move the tape-head
3. Change the state



...	B	X_1	X_2	X_3	X_4	B	B	B	...
...			*						...
...	B	Y_1	Y_2	M	Y_4	B	B	B	...
...					*				...

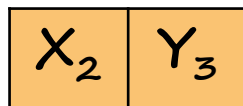
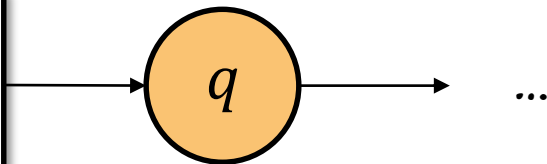


Move to right.

Equivalence of ST-TM and MT-TM (k=2)

Take the action same as done by the k-tape TM

1. Rewrite the symbols
2. Move the tape-head
3. Change the state



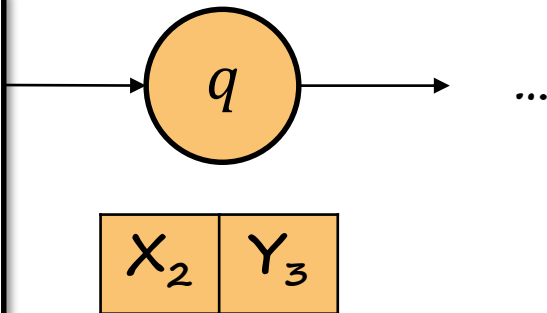
...	B	X ₁	X ₂	X ₃	X ₄	B	B	B	...
...			*						...
...	B	Y ₁	Y ₂	M	Y ₄	B	B	B	...
...					*				...



Equivalence of ST-TM and MT-TM (k=2)

Take the action same as done by the k-tape TM

1. Rewrite the symbols
2. Move the tape-head
3. Change the state



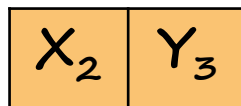
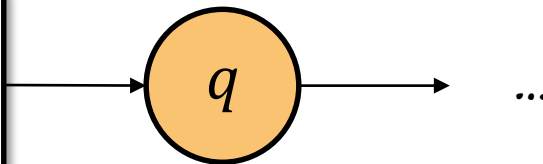
...	B	X_1	N	X_3	X_4	B	B	B	...
...									...
...	B	Y_1	Y_2	M	Y_4	B	B	B	...
...					*				...



Equivalence of ST-TM and MT-TM ($k=2$)

Take the action same as done by the k -tape TM

1. Rewrite the symbols
2. Move the tape-head
3. Change the state



...	B	X_1	N	X_3	X_4	B	B	B	...
...		*							...
...	B	Y_1	Y_2	M	Y_4	B	B	B	...
...					*				...

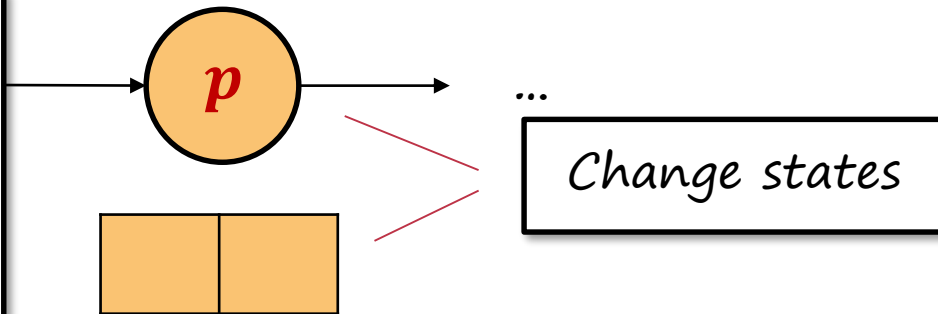


Move to left.

Equivalence of ST-TM and MT-TM (k=2)

Take the action same as done by the k-tape TM

1. Rewrite the symbols
2. Move the tape-head
3. Change the state

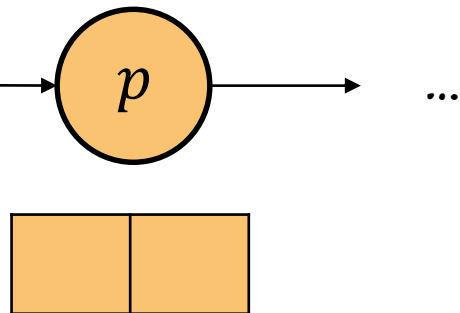


...	B	X_1	N	X_3	X_4	B	B	B	...
...		*							...
...	B	Y_1	Y_2	M	Y_4	B	B	B	...
...					*				...



Equivalence of ST-TM and MT-TM ($k=2$)

We've simulated a move of multi-tape TM. So we can continue this process to simulate the k -tape TM.



...	B	X_1	N	X_3	X_4	B	B	B	...
...		*							...
...	B	Y_1	Y_2	M	Y_4	B	B	B	...
...					*				...



Multi-tape TM and Single-tape TM

Every language accepted by a multi-tape TM is RE, or

$$P(ST-TM) = P(MT-TM)$$



Computer and Turing Machine

- What we care about most is that whether Turing Machine can complete what a computer can do.
 - We even hope TM has greater capability than a computer.
- A real computer has **memory limitations**: you have a finite amount of RAM, a finite amount of disk space, etc.
- But we'll focus on an **idealized computer** which is like a regular computer, but with unlimited RAM and disk space. It functions just like a regular computer, but never runs out of memory.

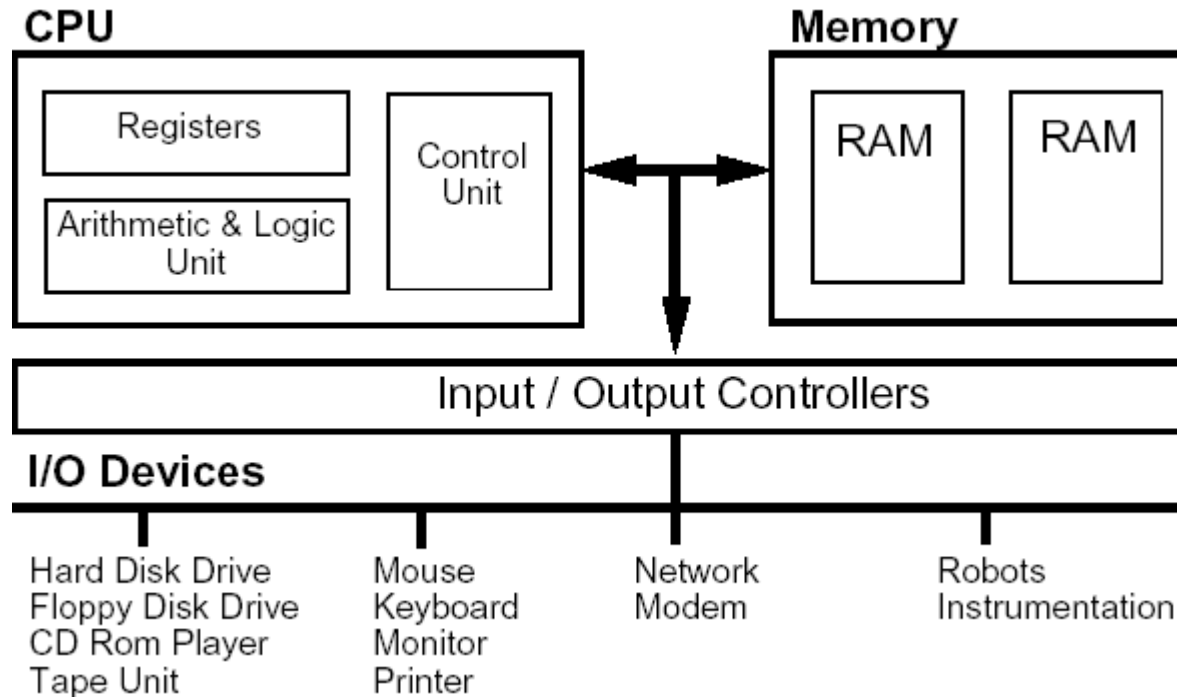
Computers Simulate TMs

- It's obvious that we can simulate a TM with a computer since we can describe it in a programmable way.
 - How about a programming lab for writing a TM?
- But the tape of TM is infinite, we can only simulate all TMs in an idealized computer with infinite memory
 - Actually, if a computer has finite memory, there's no chance to simulate all TMs.

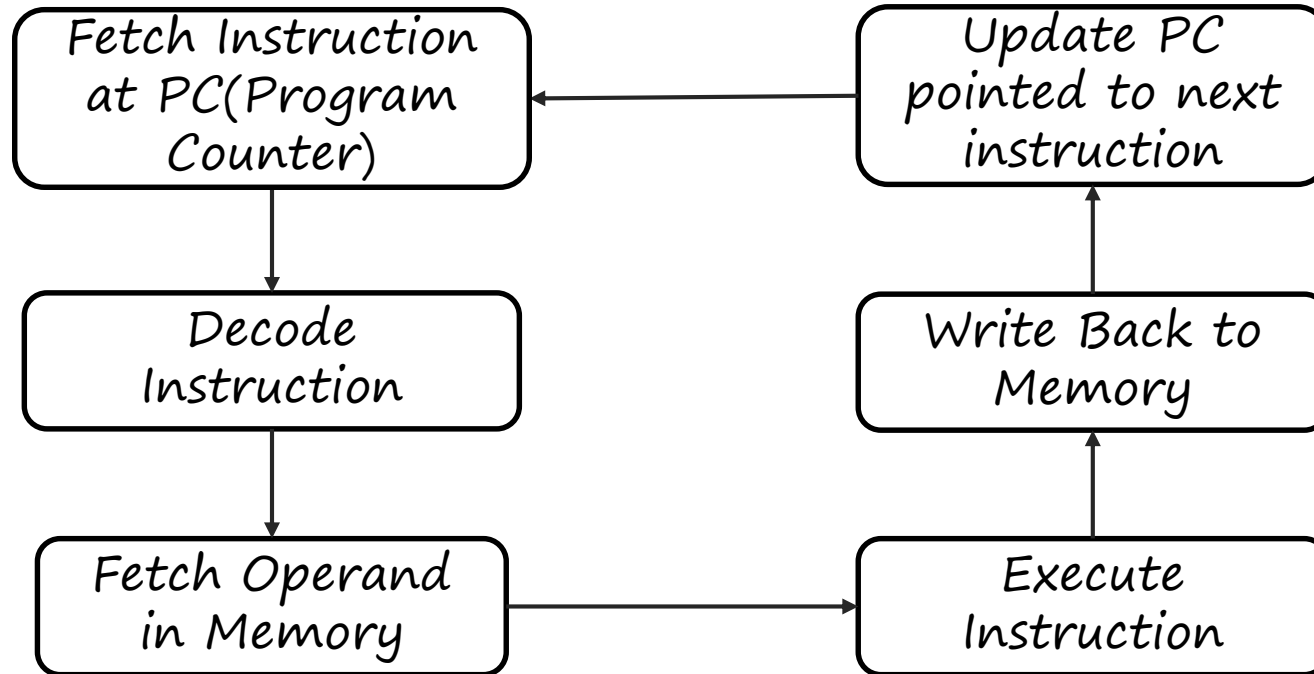
TMs Simulate Computers

- The more interesting question is that whether a TM can simulate an idealized computer.
- But what is the specific definition of an idealized computer or even a general computer?
 - No rigorous mathematical definition, we can only define it by ourselves.

Computer Architecture



Perform by Executing Instructions



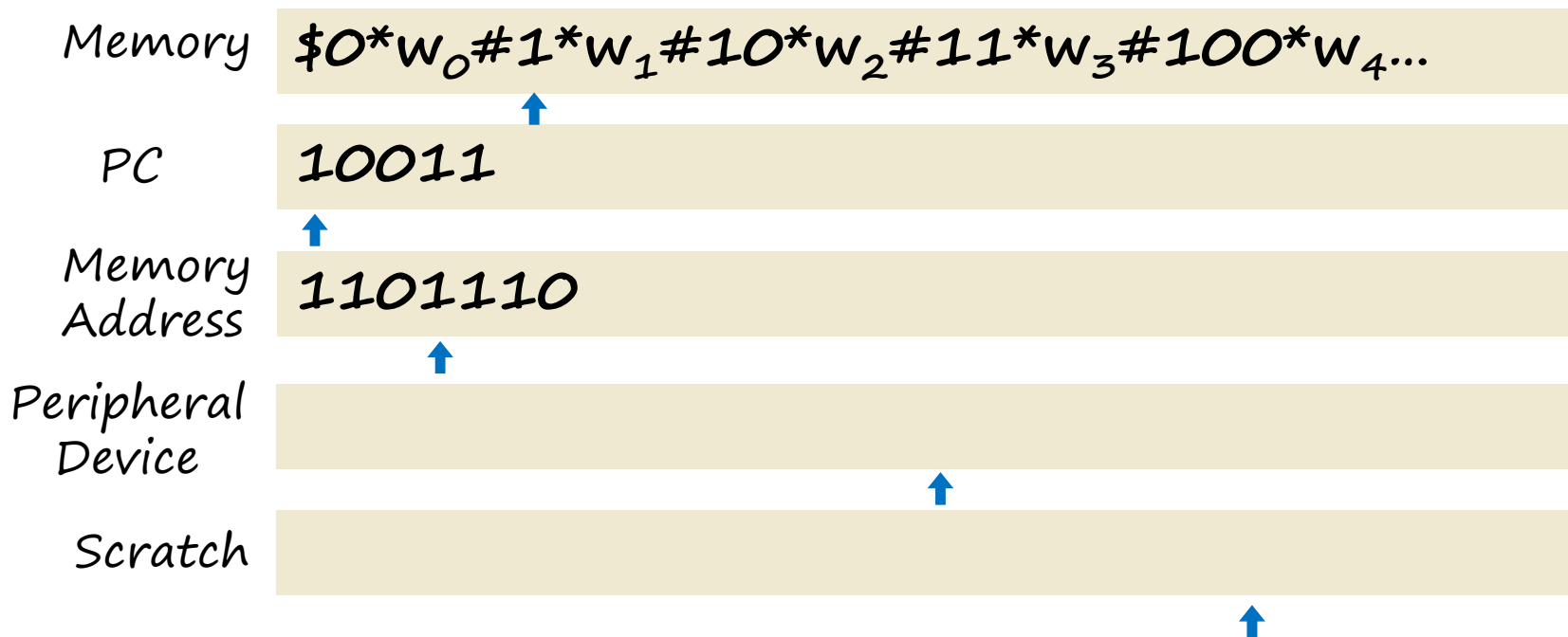
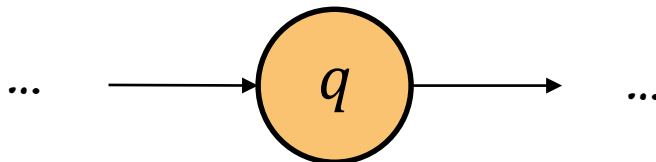
CSAPP 4.3.1 Instruction Cycle

Assumptions

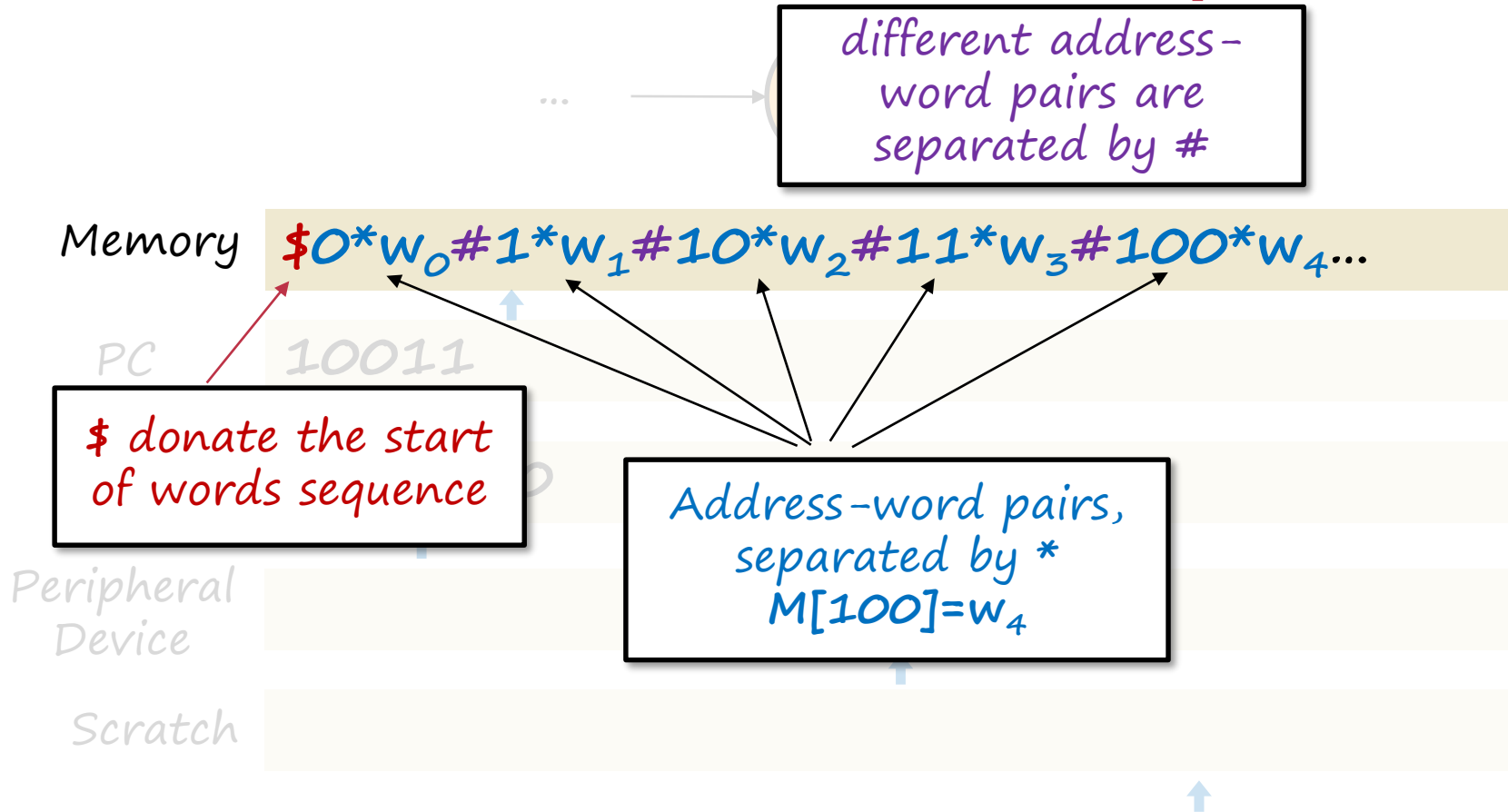
- The storage of a computer consists of a sequence of **words**, each with an **address**. A word can be of any length. Addresses will be assumed to be integers 0,1,2 and so on instead of align them to 32 or 64 bit. We use $M[n]$ to denote the n th word in memory.
- There's no other registers except a PC.
- Program(code) of the computer is stored in some of the words of memory. Each word represents a simple instruction like:
 - Zero(n): $M[n] = 0$
 - Successor(n): $M[n] = M[n] + 1$
 - Mov(m, n): $M[n] = M[m]$
 - Jump(m,n,q): if $M[m] == M[n]$, then set PC to q .

Actually these four instructions are enough for all computation!

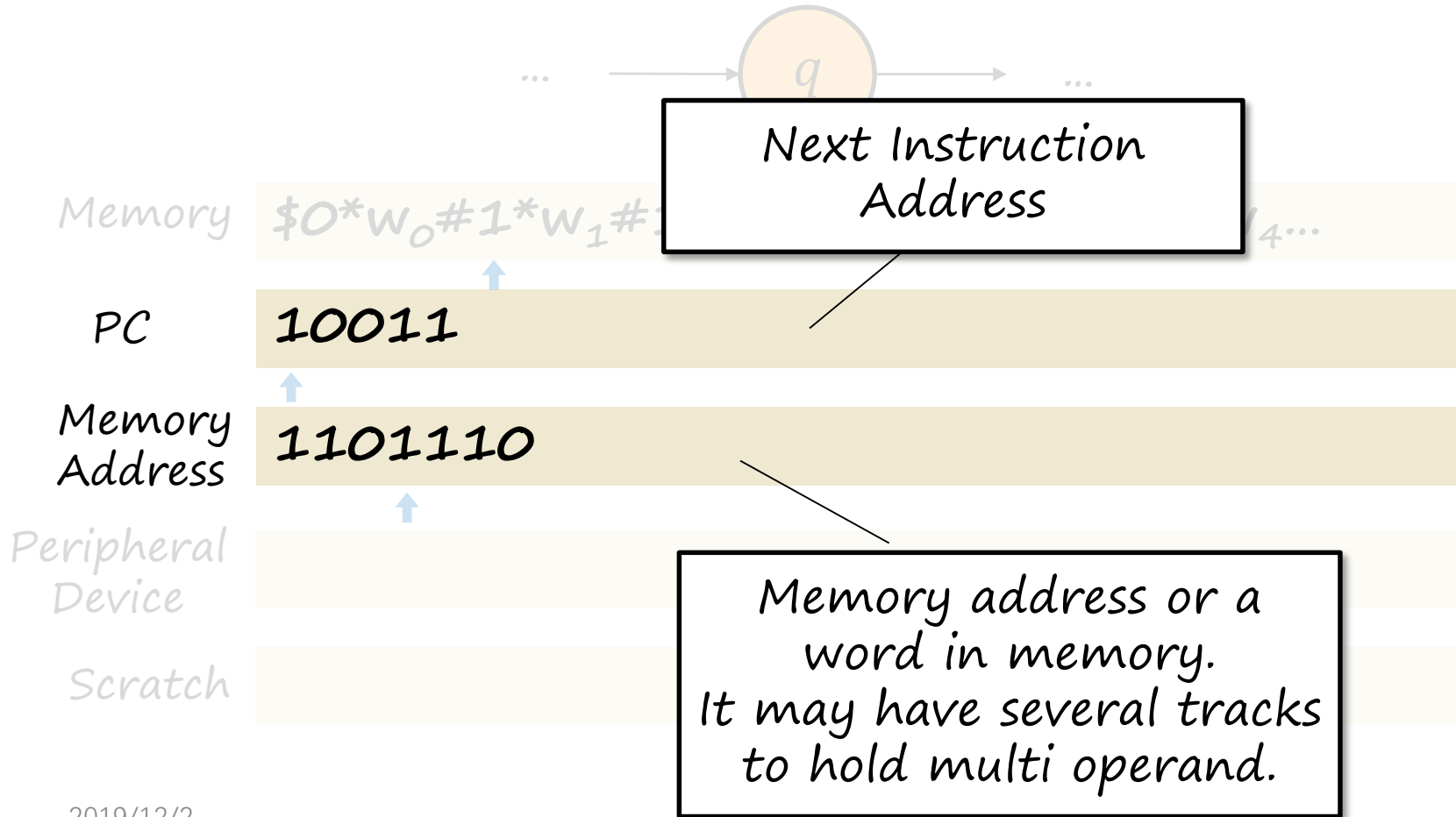
A TM Simulates Idealized Computers.



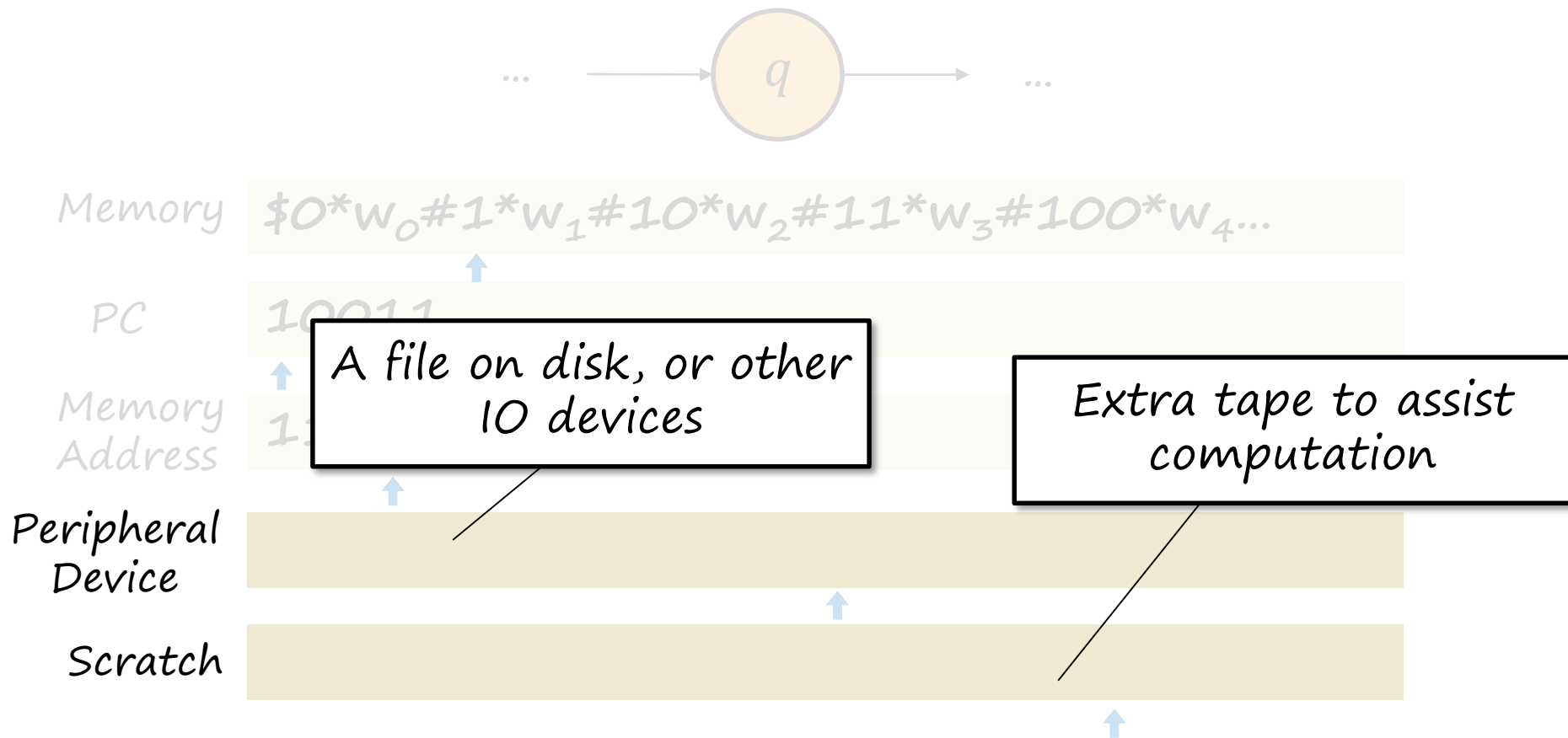
A TM Simulates Idealized Computers.



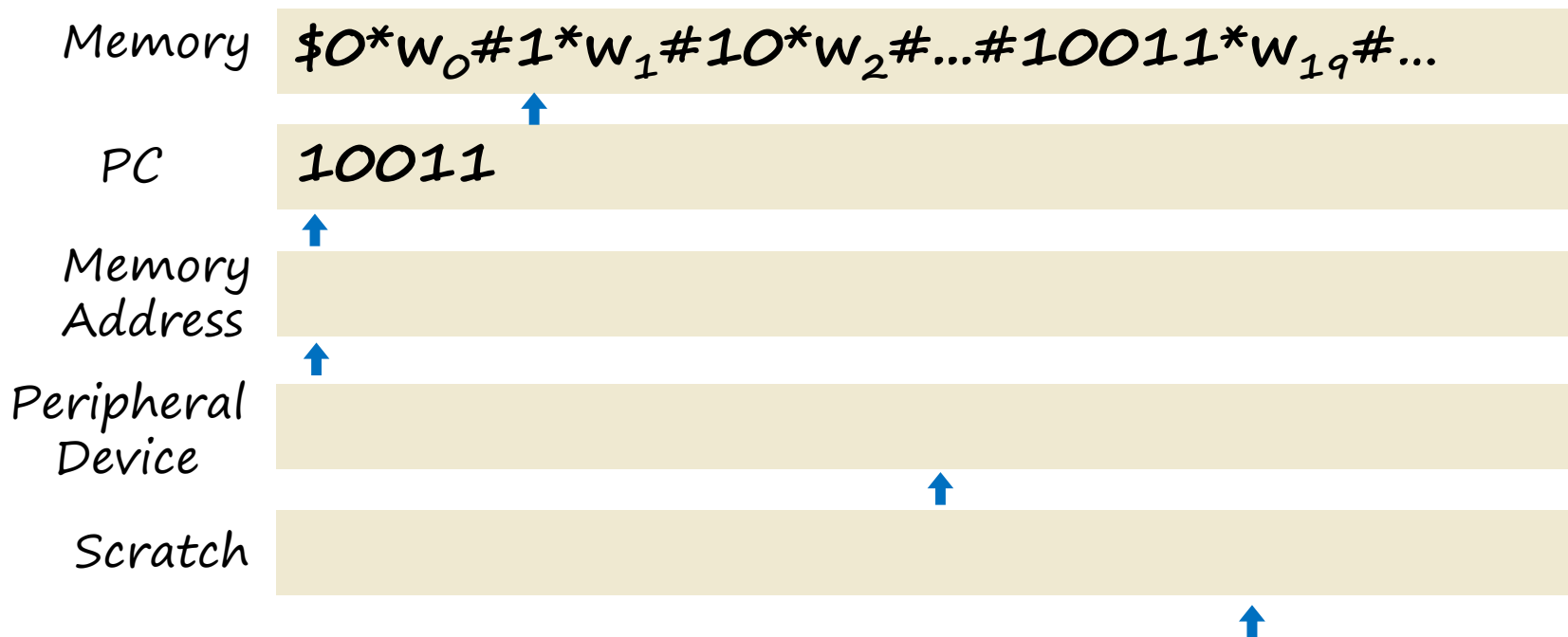
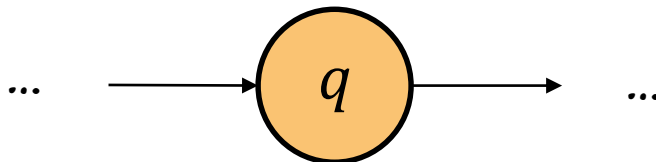
A TM Simulates Idealized Computers.



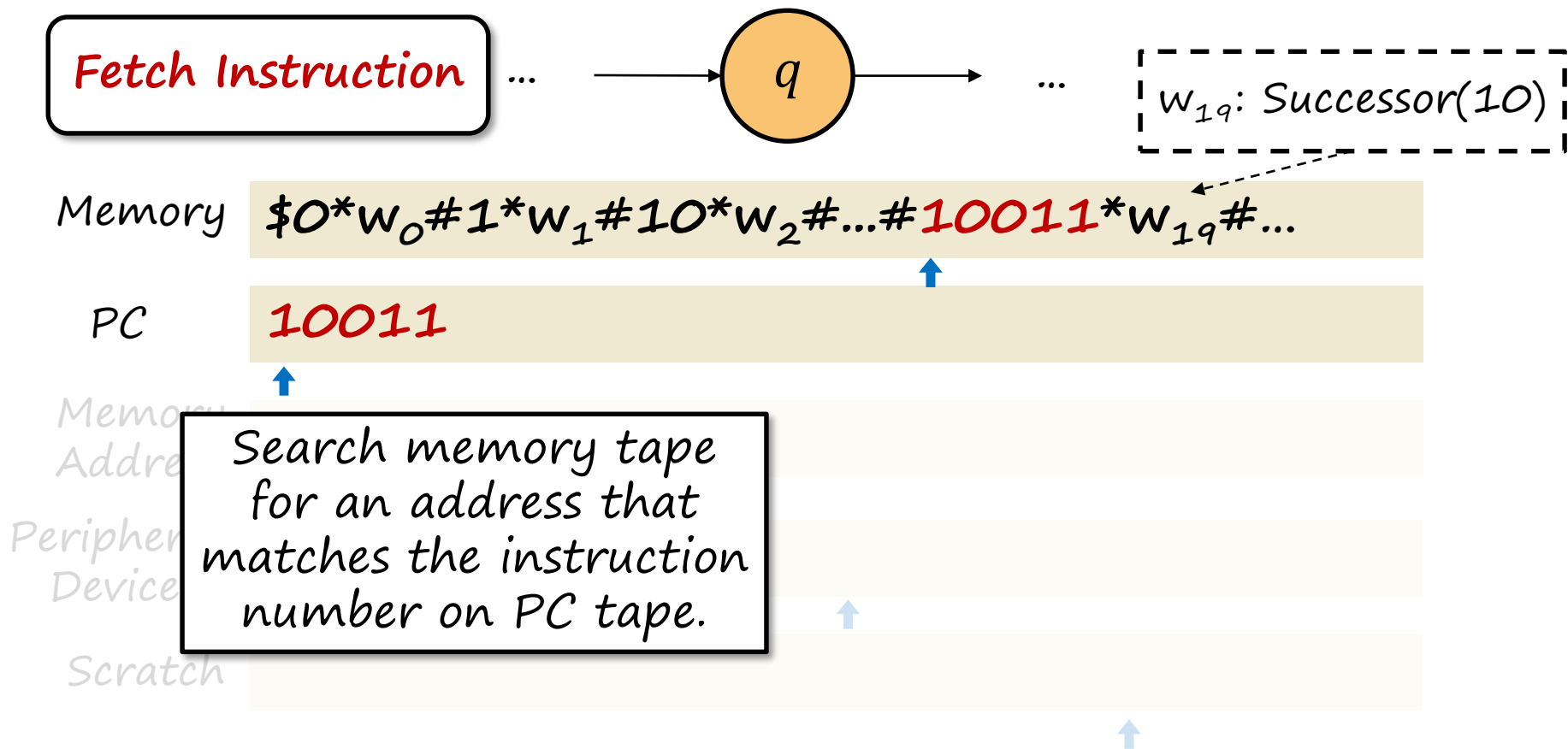
A TM Simulates Idealized Computers.



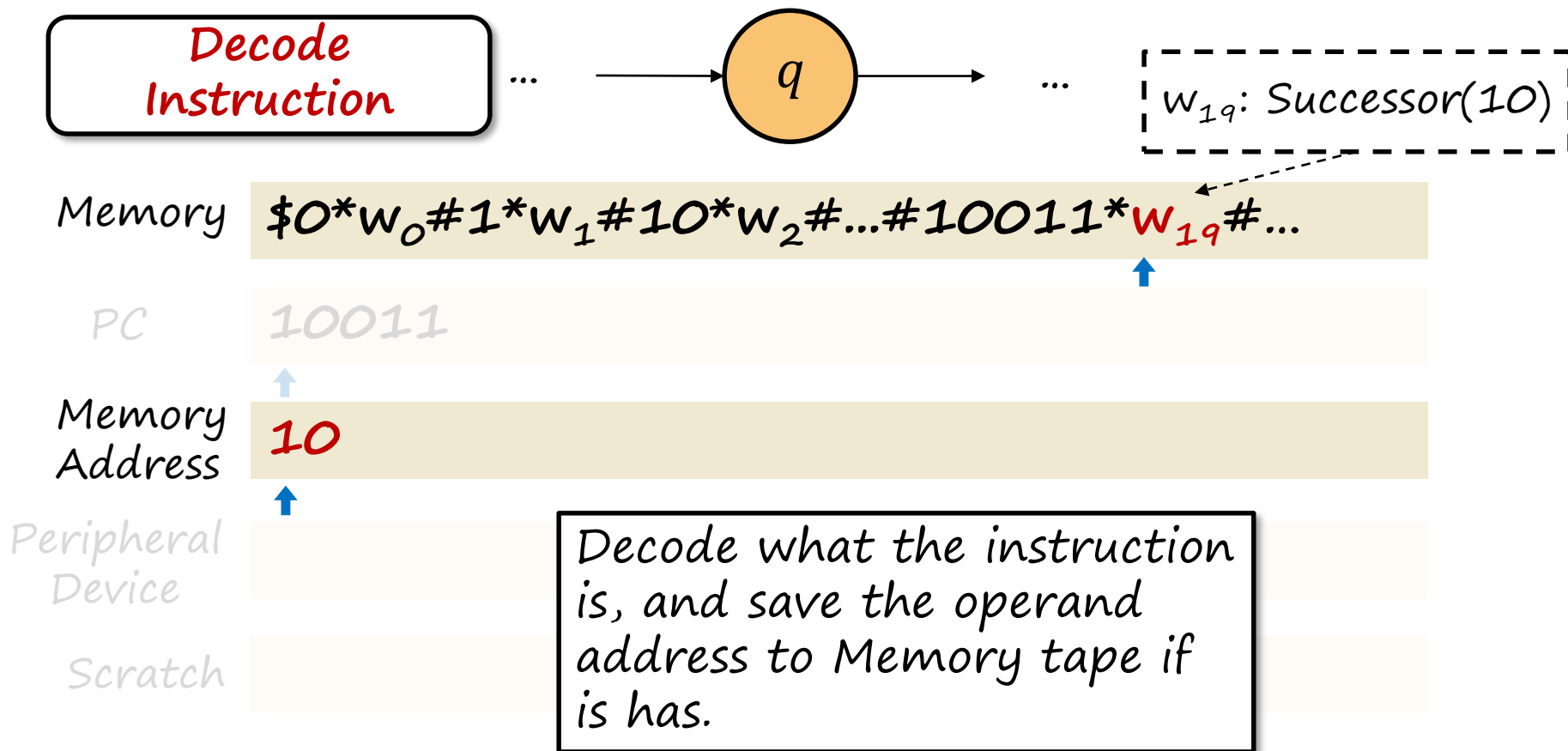
A TM Simulates Idealized Computers.



A TM Simulates Idealized Computers.

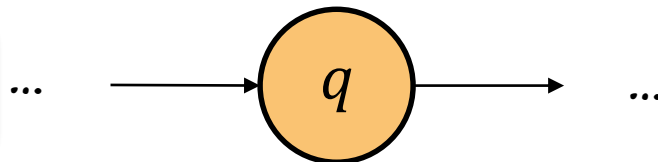


A TM Simulates Idealized Computers.



A TM Simulates Idealized Computers.

Fetch Operands



Memory $\$0^*w_0\#1^*w_1\#10^*w_2\#\dots\#10011^*w_{1q}\#\dots$



PC 10011



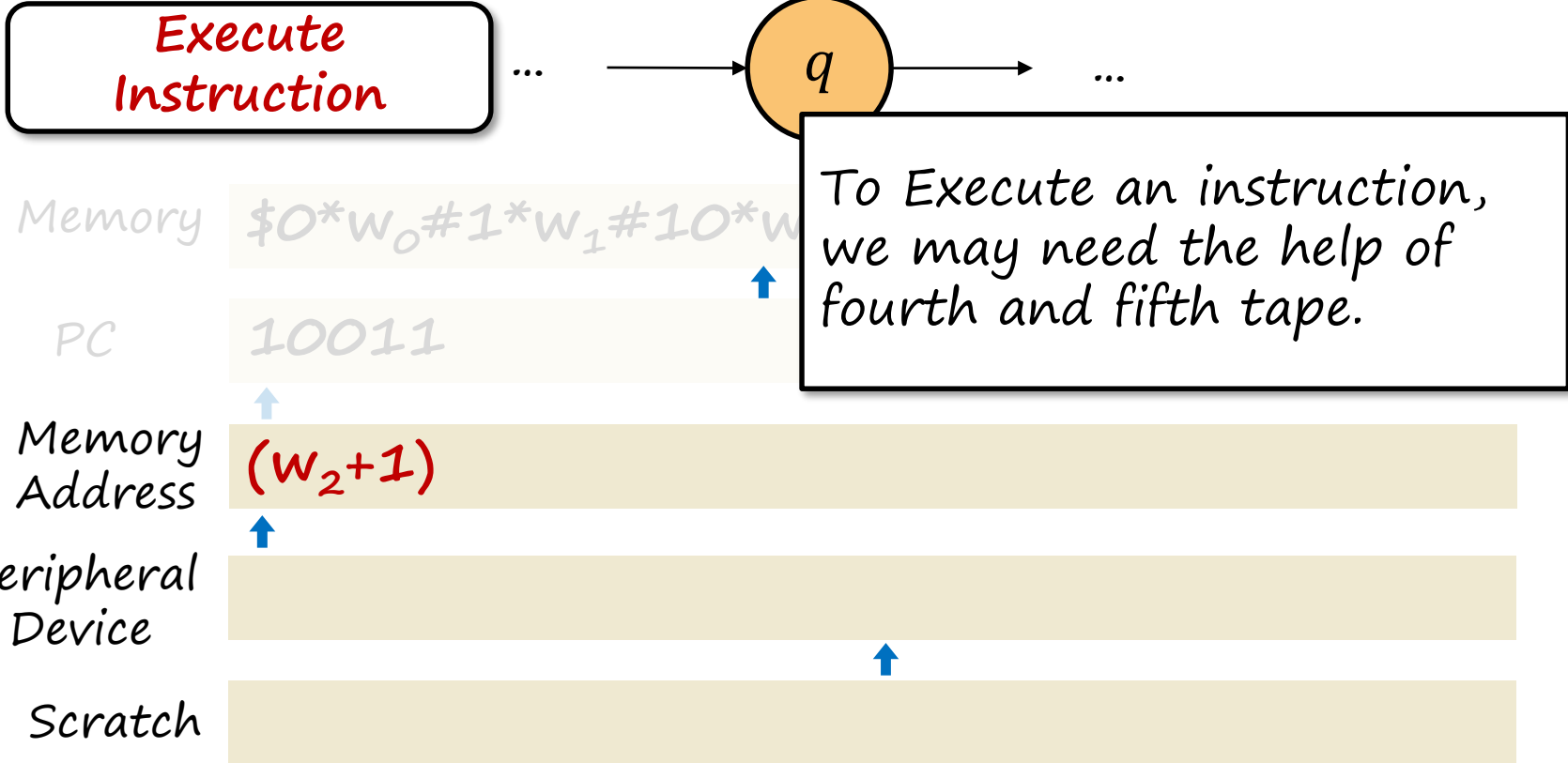
Memory Address w_2



Peripheral Device

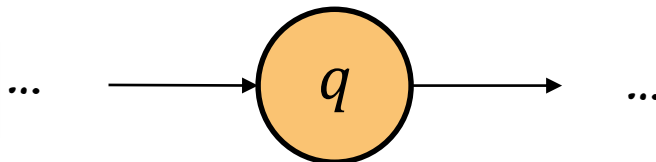
Scratch

Search memory for address that matches the address in third tape, copy its word into third tape.



A TM Simulates Idealized Computers.

Write Back



Memory $\$0^*w_0\#1^*w_1\#10^*(w_2+1)\#\dots\#10011^*w_{1q}\#\dots$



PC 10011



Memory Address (w_2+1)

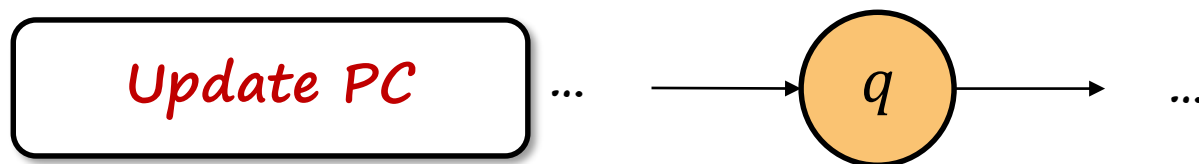


Peripheral Device

Scratch

Write back to memory if needed.

A TM Simulates Idealized Computers.



Memory $\$0^*w_0\#1^*w_1\#10^*w_2\#\dots\#10011^*w_{1q}\#\dots$



PC **10100**



Memory
Address



Peripheral
Device

If it is not jump instruction,
update PC pointed to next
instruction.

Scratch



TMs and Computers

- We've informally proved that a TM can simulate computers by simulating its instruction cycle.
 - It may require a leap of faith to believe since we do not prove it rigorously.
- So we know TM and computer can simulate each other, which means that TM has same computation capability with computer.

$TM \approx Computer$

Effective Computation

- An *effective method of computation* is a form of computation with the following properties:
 - The computation consists of a set of steps.
 - There are fixed rules governing how one step leads to the next.
 - Any computation that yields an answer does so in finitely many steps.
 - Any computation that yields an answer always yields the correct answer.
- This is not a formal definition. Rather, it's a set of properties we expect out of a computational system.

Church-Turing Thesis

Every effective method of computation is either equivalent to or weaker than a Turing machine.

- This is not a **theorem** but a falsifiable scientific hypo**thesis**. But it has been thoroughly tested! So we have strong faith in its correctness.
- This means Turing Machine can model any “computation”.

~~What problems can a computer solve?~~



~~What languages can a computer recognize?~~



What languages can TM recognize?



Next

Effective Computation on TM

- Decidable, recognizable. R & RE language
- The properties of R & RE languages.
- The universal TM.