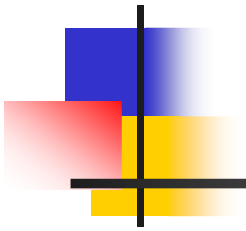
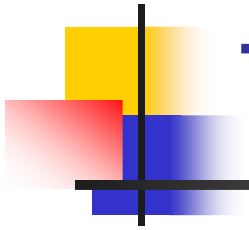


Turing Machines

Reading: Chapter 8



Turing Machines are…

- Very powerful (abstract) machines that could simulate any modern day computer

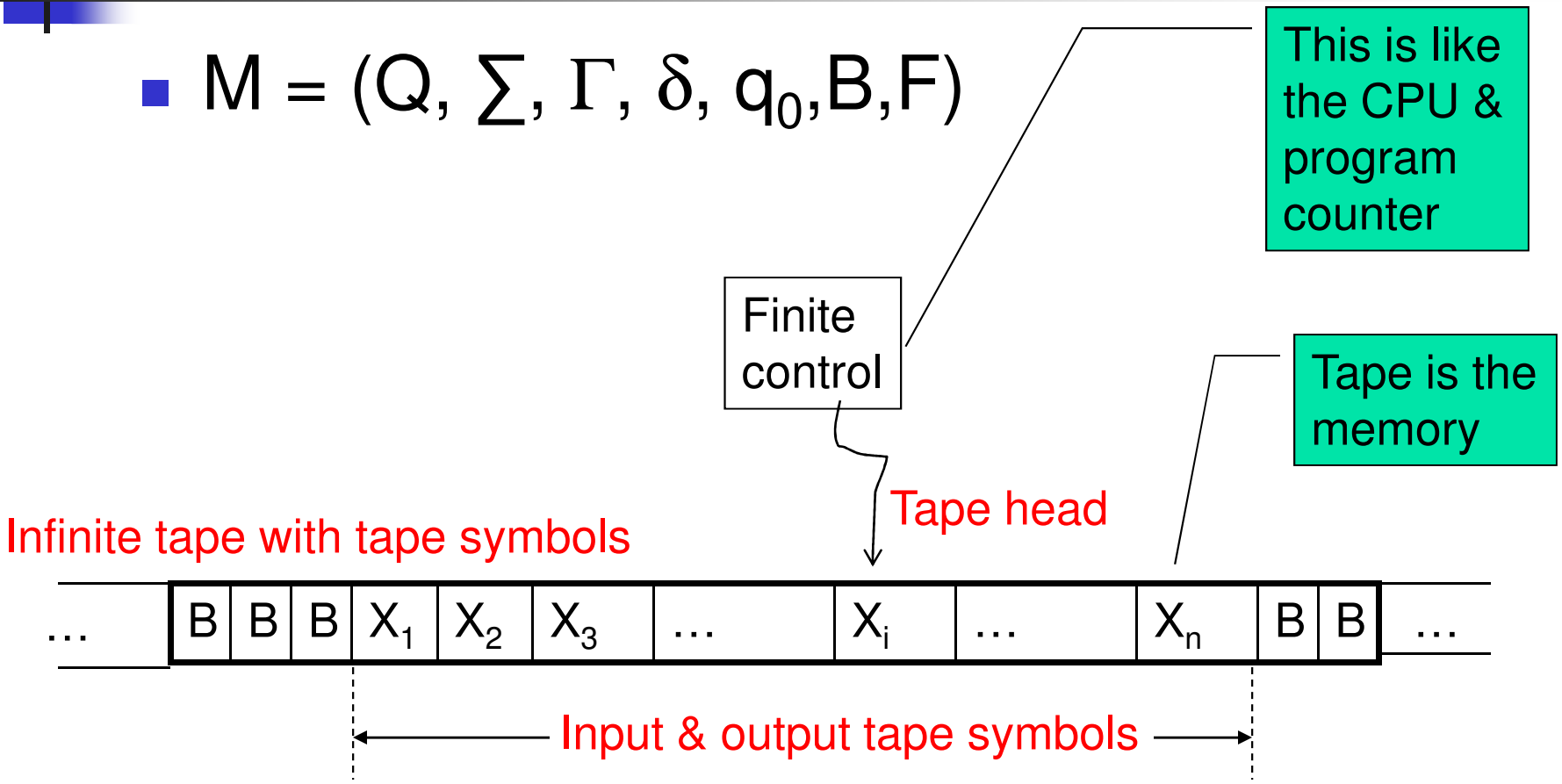
For every input,
answer YES or NO

- Why design such a machine?
  - If a problem can be "solved" using a TM, then it implies that the problem is ***decidable***

- Computability vs. Decidability

A Turing Machine (TM)

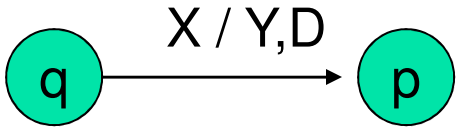
- $M = (Q, \sum, \Gamma, \delta, q_0, B, F)$

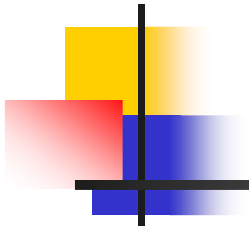


Transition function

You can also use:
-> for R
<- for L

- One move (denoted by |---) in a TM does the following:
  - ***δ(q,X) = (p,Y,D)***
    - q is the current state
    - X is the current tape symbol pointed by tape head
    - State changes from q to p
    - <u>After the move:</u>
      - X is replaced with symbol Y
      - If D="L", the tape head moves "left" by one position.
        Alternatively, if D="R" the tape head moves "right" by one position.

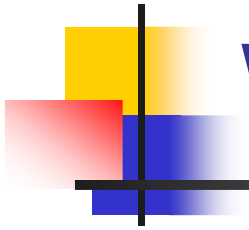




ID of a TM

- Instantaneous Description or ID :
  - $X_1X_2\ldots X_{i-1}qX_iX_{i+1}\ldots X_n$
    means:
    - q is the current state
    - Tape head is pointing to X_i
    - $X_1X_2\ldots X_{i-1}X_iX_{i+1}\ldots X_n$ are the current tape symbols

- $\delta(q,X_i) = (p,Y,R)$ is same as:
  $X_1\ldots X_{i-1}qX_i\ldots X_n$ |---- $X_1\ldots X_{i-1}YpX_{i+1}\ldots X_n$
- $\delta(q,X_i) = (p,Y,L)$ is same as:
  $X_1\ldots X_{i-1}qX_i\ldots X_n$ |---- $X_1\ldots pX_{i-1}YX_{i+1}\ldots X_n$



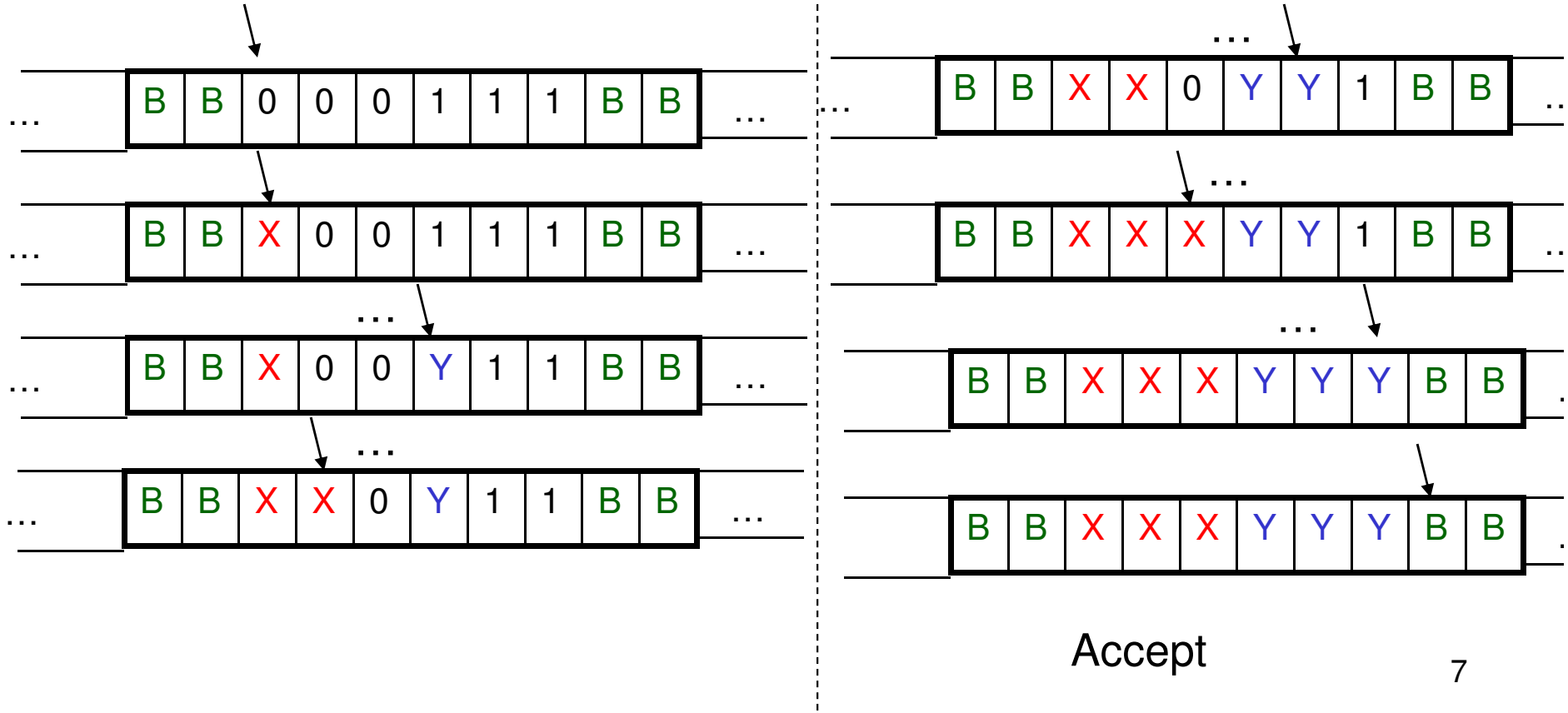
Way to check for Membership

- Is a string *w* accepted by a TM?

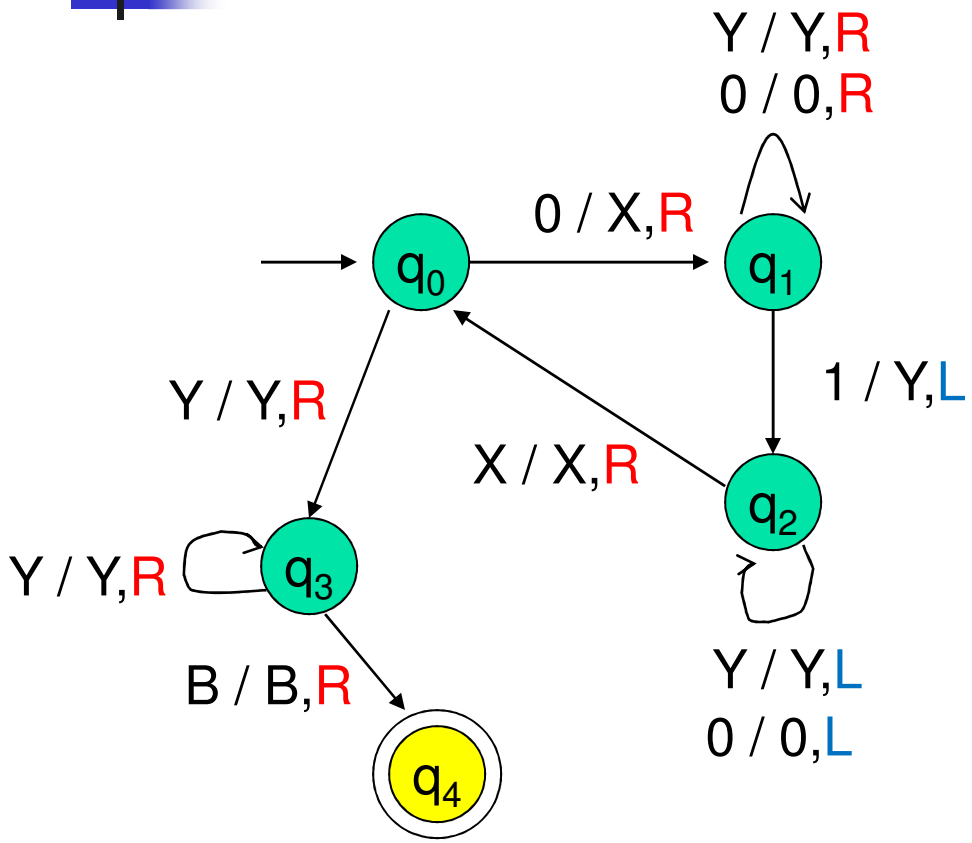
- Initial condition:
  - The input string *w* is present in TM, preceded and followed by infinite blank symbols
- Final acceptance:
  - Accept *w* if TM enters final state and halts
  - If TM halts and not final state, then reject

Example: L = {0^n1^n | n≥1}

- Strategy: w = 000111



TM for $\{0^n1^n \mid n \geq 1\}$

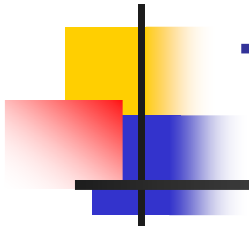


1. Mark next unread 0 with X and move right
2. Move to the right all the way to the first unread 1, and mark it with Y
3. Move back (to the left) all the way to the last marked X, and then move one position to the right
4. If the next position is 0, then goto step 1.
   Else move all the way to the right to ensure there are no excess 1s. If not move right to the next blank symbol and stop & accept.

TM for $\{0^n1^n \mid n \geq 1\}$

	Next Tape Symbol				
Curr. State	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)
$^*q_4$	-	--	-	-	-

Table representation of the state diagram



TMs for calculations

- TMs can also be used for calculating values
  - Like arithmetic computations
  - Eg., addition, subtraction, multiplication, etc.

Example 2: monus subtraction

"m -- n" = max{m-n,0}

0^m10^n (*input*) 0^{m-n} or ...BB…B.. (*output*)

Give state diagram

1. For every 0 on the left (mark X), mark off a 0 on the right (mark Y)
2. Repeat process, until one of the following happens:
    1. // No more 0s remaining on the left of 1
       Answer is 0, so flip all excess 0s on the right of 1 to Bs (and the 1 itself) and halt
    2. //No more 0s remaining on the right of 1
       Answer is m-n, so simply halt after making 1 to B

- Some correction moves may be needed towards the end.

Example 3: Multiplication

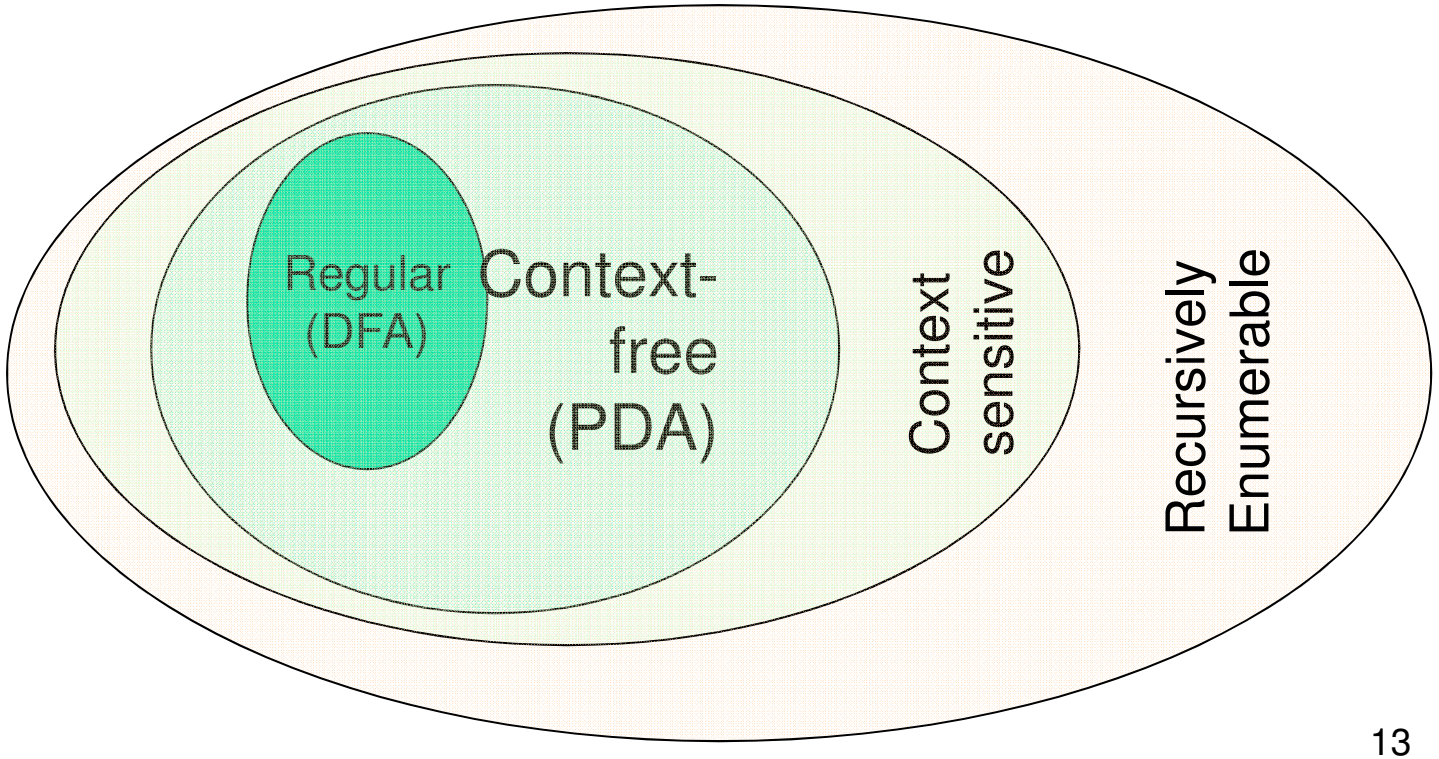
- 0^m10^n1 (input), $0^{mn}1$ (output)

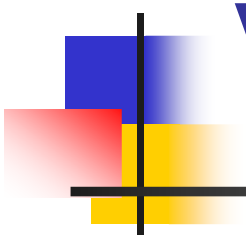
- Pseudocode:
  1. Move tape head back & forth such that for every 0 seen in 0^m, write n 0s to the right of the last delimiting 1
  2. Once written, that zero is changed to B to get marked as finished
  3. After completing on all m 0s, make the remaining n 0s and 1s also as Bs

Give state diagram

Language of the Turing Machines

- *Recursive Enumerable (RE) language*



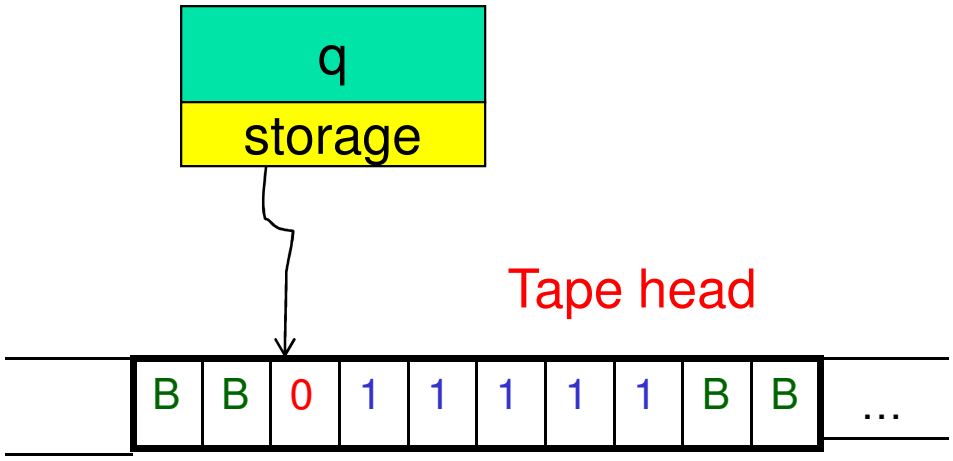


Variations of Turing Machines

Generic description
Will work for both
a=0 and a=1

TMs with *storage*

- E.g., TM for 01* + 10*



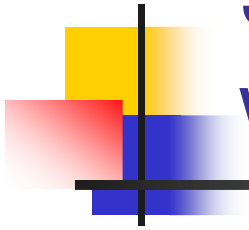
Transition function δ:

- $\delta([q_0,B],a) = ([q_1,a], a, R)$
- $\delta([q_1,a],\overline{a}) = ([q_1,a], \overline{a}, R)$
- $\delta([q_1,a],B) = ([q_2,B], B, R)$

[q,a]: where q is current state, a is the symbol in storage

Are the standard TMs equivalent to TMs with storage?

Yes



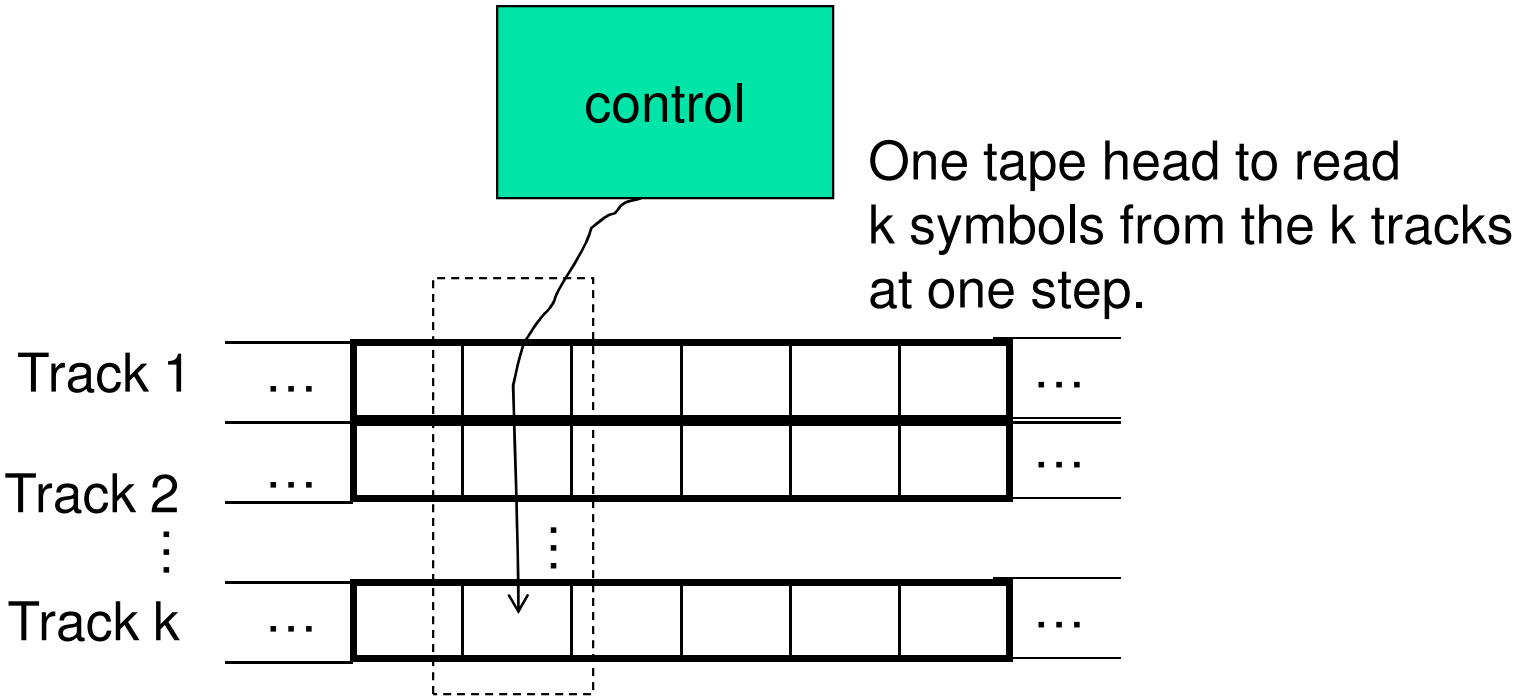
Standard TMs are equivalent to TMs with storage - Proof

- Every TM w/ storage can be simulated by a TM w/o storage as follows:
  - For every [state, symbol] combination in the TM w/ storage:
    - Create a new state in the TM w/o storage
    - Define transitions induced by TM w/ storage

Since there are only finite number of states and symbols in the TM with storage, the number of states in the TM without storage will also be finite

Multi-track Turing Machines

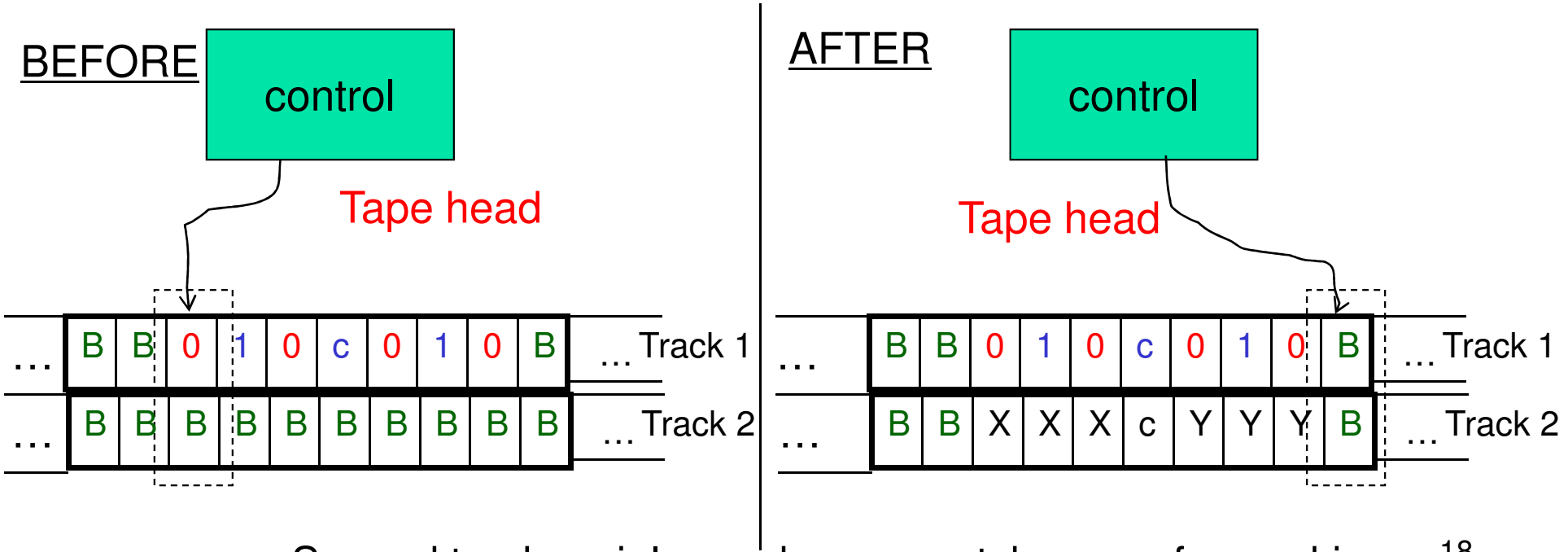
- TM with multiple tracks,
  but just one unified tape head



Multi-Track TMs

- TM with multiple “tracks” but just one head

E.g., TM for {wcw | w∈ {0,1}* }
but w/o modifying original input string



Second track mainly used as a scratch space for marking

Multi-track TMs equivalency to TMs

- Let M be a single-track TM
  - $M = (Q, \sum, \Gamma, \delta, q_0, B, F)$
- Let M' be a multi-track TM (*k* tracks)
  - $M' = (Q', \sum', \Gamma', \delta', q'_0, B, F')$
  - $\delta'(q_i, \langle a_1, a_2, \ldots a_k \rangle) = (q_j, \langle b_1, b_2, \ldots b_k \rangle, L/R)$
- Claims:
  - *For every M, there is an M' s.t. L(M)=L(M').*
    - (proof trivial here)

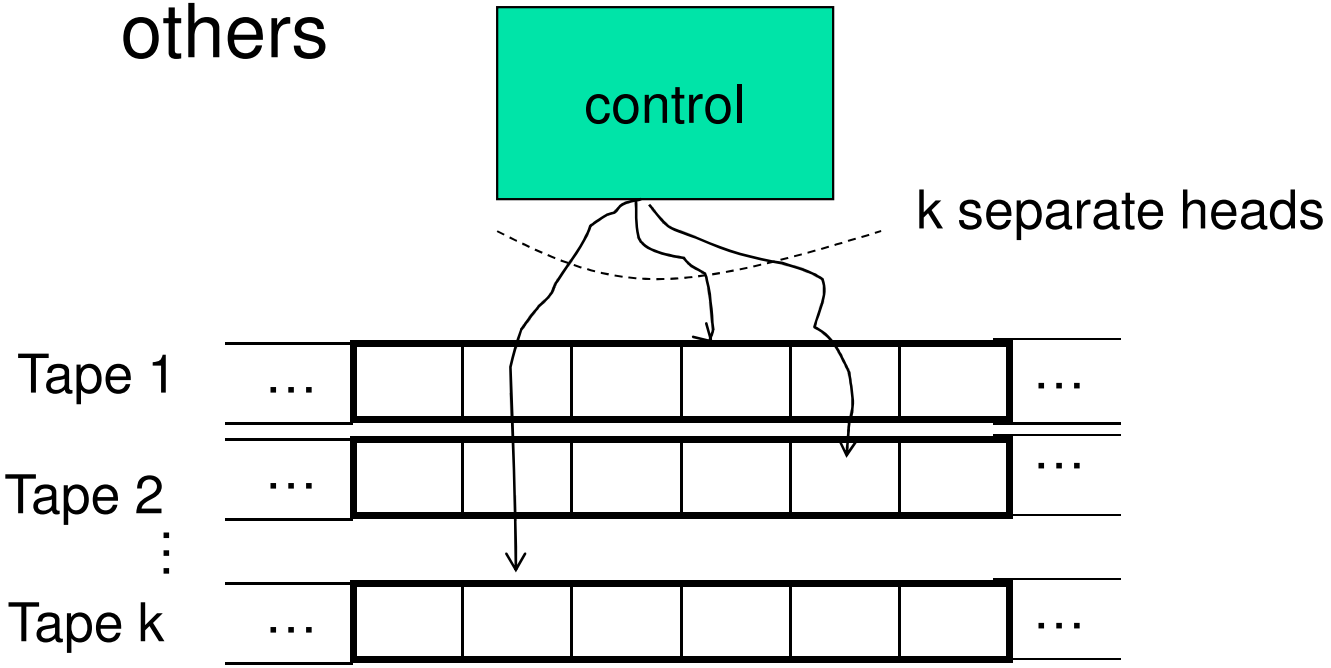
Multi-track TM ==> TM (proof)

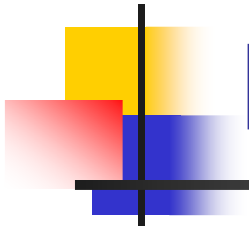
- *For every M', there is an M s.t. L(M')=L(M).*
  - M = (Q, $\sum$, Γ, δ, q_0,[B,B,…],F)
  - Where:
    - Q = Q'
    - $\sum = \sum' \times \sum' \times \ldots$ (k times for k-track)
    - $\Gamma = \Gamma' \times \Gamma' \times \ldots$ (k times for k-track)
    - $q_0 = q'_0$
    - F = F'
    - $\delta(q_i,[a_1,a_2,\ldots a_k]) = \delta'(q_i, <a_1,a_2,\ldots a_k>)$
- Multi-track TMs are just a different way to represent single-track TMs, and is a matter of design convenience.

Main idea:
Create one composite symbol to represent every combination of k symbols

Multi-tape Turing Machines

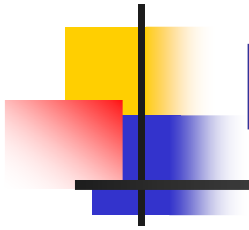
- TM with multiple tapes, *each tape with a separate head*
  - Each head can move independently of the others





Multi-tape TM

- Initially:
  - The input is in tape #1
  - All other cells in tape #1 are blanks
  - All other tapes contain only blanks
  - The tape head for tape #1 points is at the left end of the input
  - The heads for all other tapes point at an arbitrary cell (don't care since all are blanks anyway)
- A move:
  - Is a function of the current state and the symbols pointed by all the heads
  - After each move, each tape head can move independently (left or right) of one another

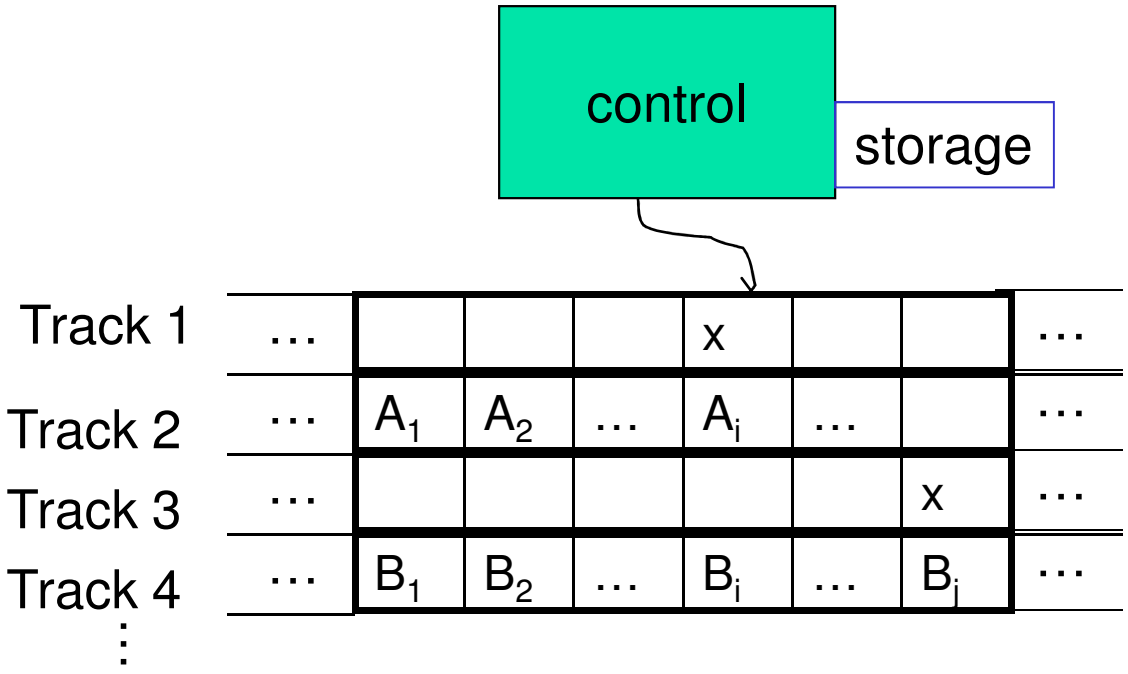


Multitape TMs ≡ Basic TMs

- Theorem: Every language accepted by a k-tape TM is also accepted by a single-tape TM
- Proof by construction:
  - Construct a single-tape TM with 2k tracks, where each tape of the k-tape TM is simulated by 2 tracks of basic TM
  - k out the 2k tracks simulate the k input tapes
  - The other k out of the 2k tracks keep track of the k tape head positions

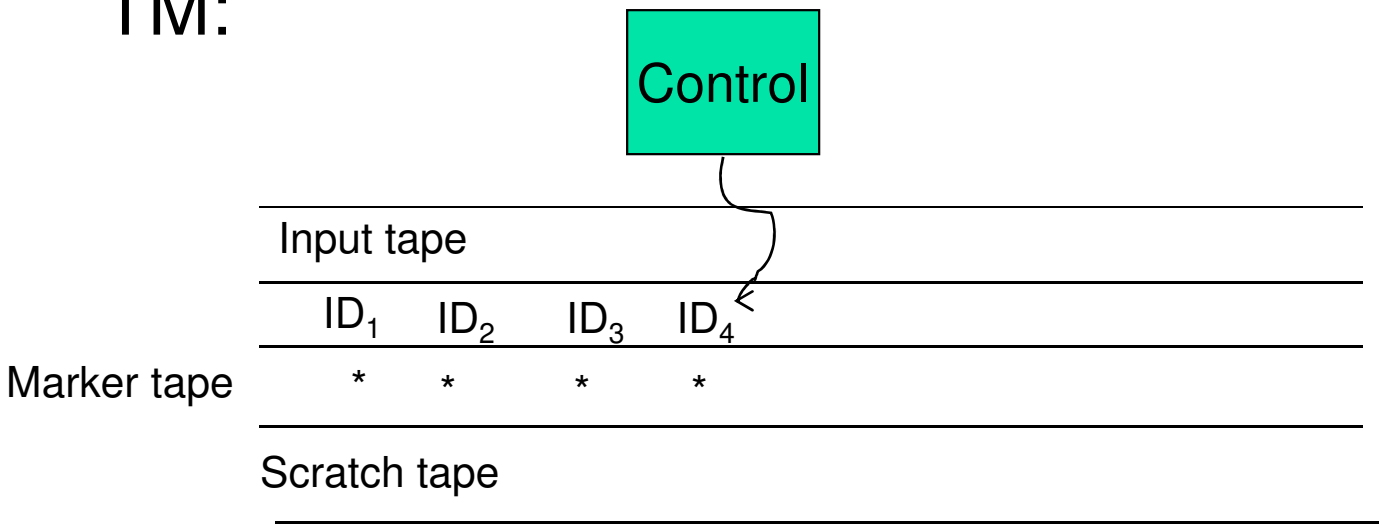
Multitape TMs ≡ Basic TMs …

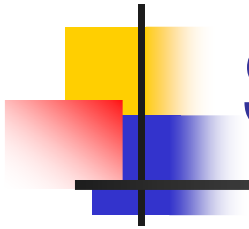
- To simulate one move of the k-tape TM:
  - Move from the leftmost marker to the rightmost marker (k markers) and in the process, gather all the input symbols into storage
  - Then, take the action same as done by the k-tape TM (rewrite tape symbols & move L/R using the markers)



Non-deterministic TMs

- A TM can have non-deterministic moves:
  - $\delta(q,X) = \{ (q_1,Y_1,D_1), (q_2,Y_2,D_2), \ldots \}$
- Simulation using a multitape deterministic TM:





Summary

- TMs == Recursively Enumerable languages
- TMs can be used as both:
  - Language recognizers
  - Calculators/computers
- Basic TM is equivalent to:
  - TM + storage
  - Multi-track TM
  - Multi-tape TM
  - Non-deterministic TM
- TMs are like universal computing machines with unbounded storage