MACCHINE NON DETERMINISTICHE
COMPLESSITÀ TEMPORALE E SPAZIALE
LE CLASSI P E NP

INFORMATICA

MACCHINE NON DETERMINISTICHE

- Fino ad ora abbiamo studiato le macchine di Turing **deterministiche**, dove ogni configurazione aveva al più una sola configurazione successiva

- Cosa succede se invece di avere un funzione di transizione invece di avere come condominio $Q \times \Gamma \times \{ \leftarrow , \rightarrow \}$ avesse $\mathscr{P}(Q \times \Gamma \times \{ \leftarrow , \rightarrow \})$?

- Invece di avere una sola scelta di stato successivo, simbolo da scrivere e movimento ne possiamo avere un insieme

MACCHINE NON DETERMINISTICHE

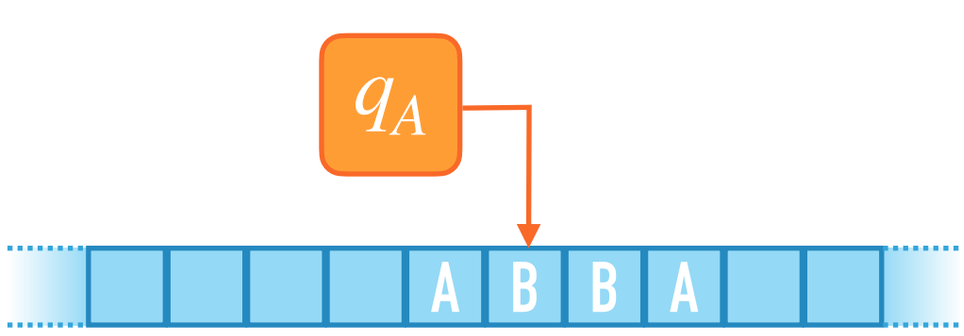
- Prendiamo la stessa definizione di macchina di Turing deterministica e cambiamo solo la definizione delle funzione di transizione

- Una macchina di Turing non deterministica (non-deterministic Turing machine o NDTM) è un settupla $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ definita come una TM tranne che la funzione di transizione ha la forma $\delta : Q \times \Gamma \to \mathscr{P}(Q \times \Gamma \times \{ \leftarrow , \rightarrow \})$

MACCHINE NON DETERMINISTICHE

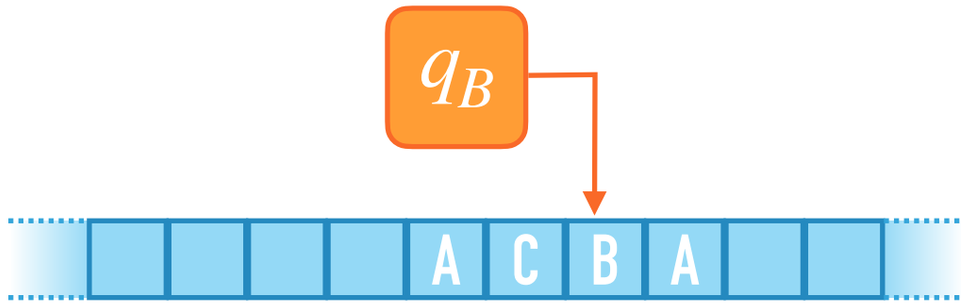
Come agisce una macchina non deterministica?

Supponiamo $\delta(q_A, A) = \{(q_B, C, \rightarrow), (q_A, B, \leftarrow)\}$

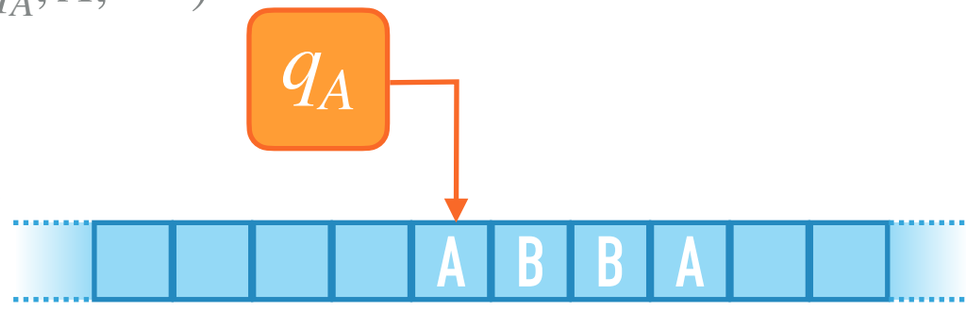


Come prosegue la computazione?

Scelta $(q_B, C, \rightarrow)$



Scelta $(q_A, A, \leftarrow)$

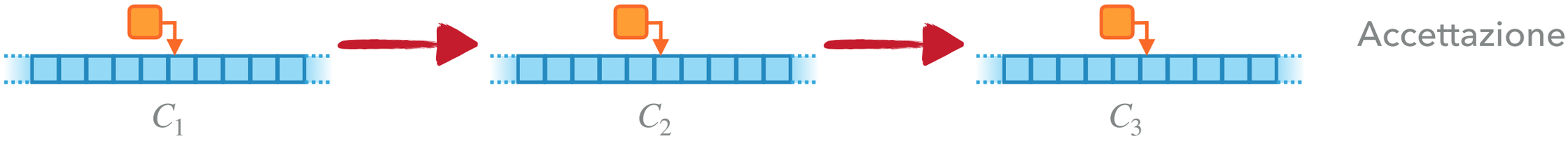


MACCHINE NON DETERMINISTICHE

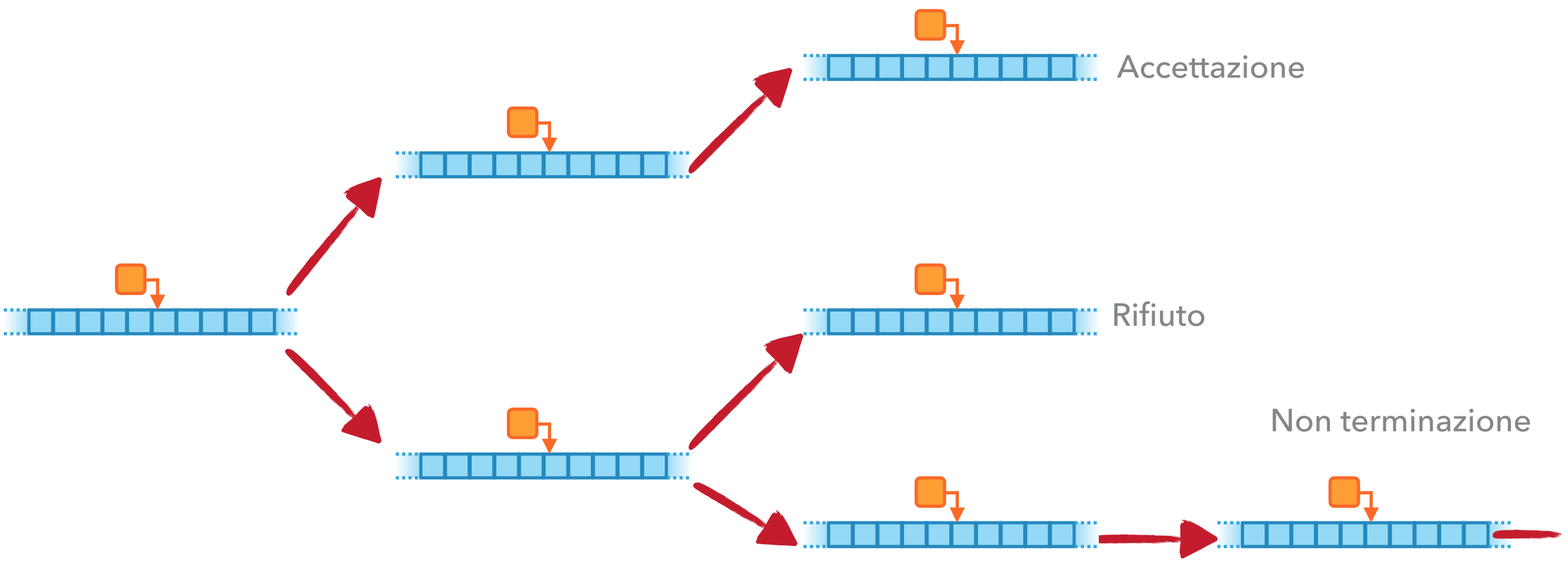
- La macchina non deterministica sceglie in modo non deterministico tra tutte le scelte possibili
- Quindi a partire dallo stesso input possono esserci più computazioni!
- Mentre per una TM deterministica parliamo della sua computazione su input w...
- ...in una TM non deterministica parliamo delle sue computazioni su input w

COMPUTAZIONI

Macchina deterministica



Macchina non deterministica



MACCHINE NON DETERMINISTICHE: ACCETTAZIONE

- Definire le condizioni di accettazione delle macchine non deterministiche non è immediato:
- Cosa succede se una computazione accetta ed una rifiuta? Accettiamo o rifiutiamo?
- Si è scelto di dire che una NDTM M accetta l'input w se **esiste** una computazione accettante
- Il quantificatore è importante

MACCHINE NON DETERMINISTICHE: COSA FANNO?

- Le macchine non deterministiche non risolvono più problemi della macchine deterministiche[1]
- Le NDTM non corrispondono a nessun modello fisico realistico — ma vedremo che sono comunque molto importanti
- Possiamo pensare alle NDTM come macchine che, davanti ad una scelta, esplorano tutte le future computazioni possibili ed accettano quando trovano una sequenza di scelta che permette di accettare

[1] Per chi vuole provare a dimostrarlo, un suggerimento è che è possibile costruire una TM deterministica che simula una non deterministica visitando in ampiezza l'albero di computazioni e accettare al primo stato accettante incontrato

PASSIAMO ALLE QUESTIONI DI COMPLESSITÀ

- Fino ad ora abbiamo visto i linguaggi che le TM possono riconoscere o decidere
- Ora passiamo ad analizzare l'efficienza con cui un linguaggio viene deciso. Data una parola w possiamo chiederci:
  - Complessità temporale: quanti passi servono per decidere se $w \in L$?
  - Complessità spaziale: quante celle di nastro ci servono per decidere se $w \in L$?

TEMPO DI CALCOLO PER TM

- Data una macchina di Turing M ed un input $w \in \Sigma^\star$ possiamo contare il numero di passi che M richiede prima di fermarsi, che indicheremo con $t(w)$
- Per tutti gli $n \in \mathbb{N}$ definiamo $t(n)$ come il massimo del numero di passi che M richiedere per riconoscere una parola $w \in \Sigma^\star$ di lunghezza n (i.e., $|w| = n$):

$$t(n) = \max_{w \in \Sigma^\star,\, |w| = n} t(w)$$

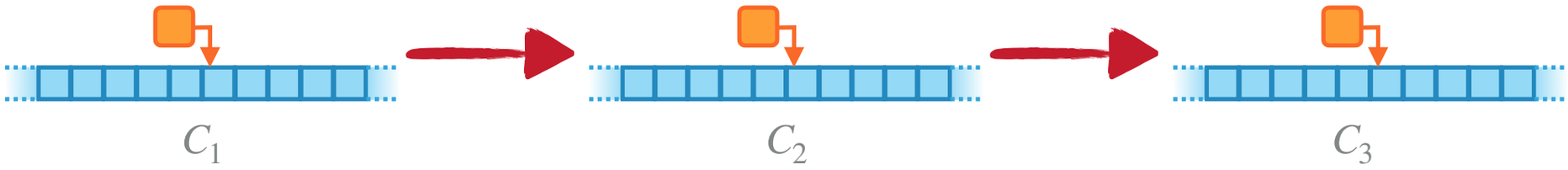
TEMPO DI CALCOLO PER TM NON DETERMINISTICHE

- Il tempo di calcolo per NDTM è lievemente più complesso perché abbiamo più di una computazione
- Su una parola w di input definiamo $t(w)$ il **massimo** numero di passi tra tutte le possibili computazioni
- Come prima, $t(n)$ rappresenta il massimo $t(w)$ tra tutti le parole w di lunghezza n

COMPLESSITÀ TEMPORALE

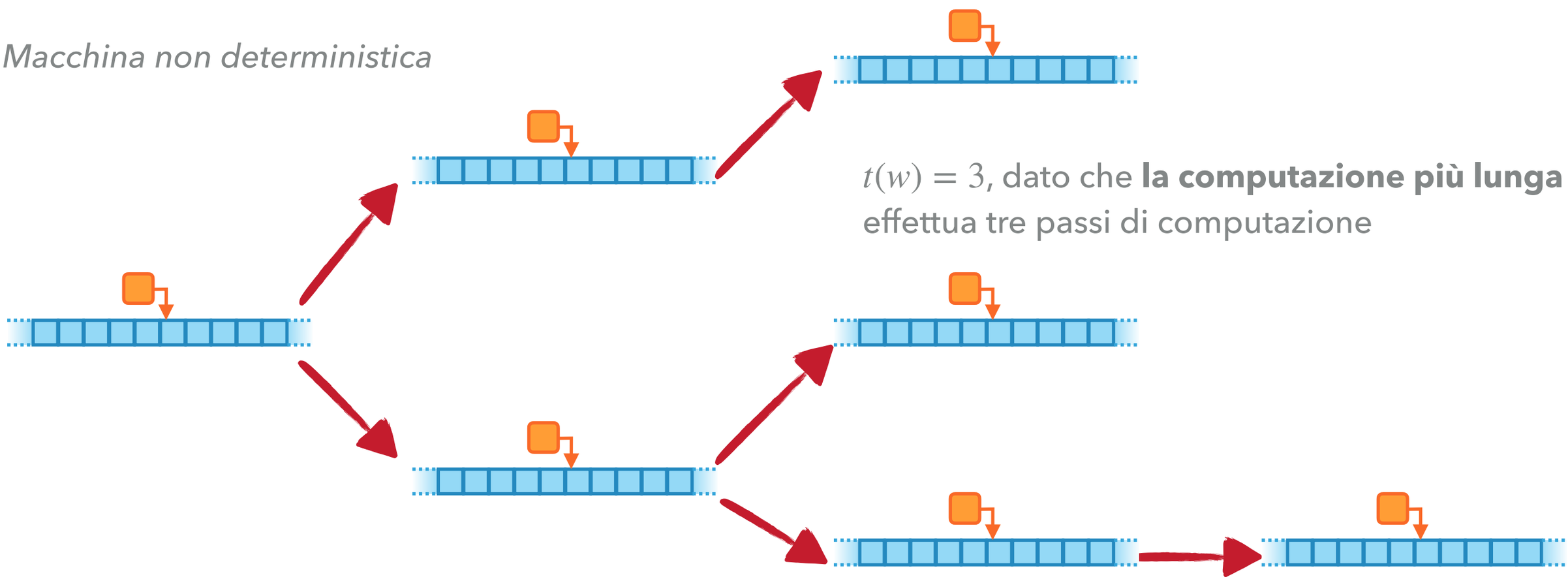
Macchina deterministica

$t(w) = 2$, dato che effettuiamo due passi di computazione



Macchina non deterministica

$t(w) = 3$, dato che **la computazione più lunga** effettua tre passi di computazione



MA LA COMPLESSITÀ TEMPORALE DEI LINGUAGGI?

- A noi però interessa non la singola TM, ma un linguaggio
- Dato un linguaggio L possiamo avere più TM che lo decidono, ognuna con la sua complessità temporale
- L'esistenza di una TM M che decide L in tempo $t(n)$ ci dice il tempo $t(n)$ è **sufficiente** per decidere L

CLASSI DI COMPLESSITÀ TEMPORALI

- Definiamo come $\mathrm{TIME}(f(n))$ l'insieme di tutti i **linguaggi** per i quali esiste una macchina di Turing deterministica che li decide in tempo $O(f(n))$
- Similmente, definiamo come $\mathrm{NTIME}(f(n))$ l'insieme di tutti i linguaggi per i quali esiste una macchina di Turing non-deterministica che li decide in tempo $O(f(n))$
- Chiaramente $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$ per qualsiasi funzione $f(n)$

CLASSI DI COMPLESSITÀ TEMPORALI

- Notate che per dimostrare che $L \notin \mathrm{TIME}(f(n))$ dobbiamo dimostrare che non esiste alcuna TM in grado di decidere L in tempo $O(f(n))$
- Questo è differente dalle analisi fatte in precedenza! Non stiamo guardando la complessità di uno specifico algoritmo (una TM) ma del problema (il linguaggio)

MODELLI DI CALCOLO DETERMINISTICI RAGIONEVOLI

- Noi usiamo TM per studiare la complessità temporale
- Una possibile critica è che una TM è dissimile da un moderno computer o dal modello di macchina RAM
- Ma, informalmente, tutti i sistemi di calcolo deterministici ragionevoli sono **polinomialmente equivalenti**
- Ovvero possono tutti simularsi a vicenda con un rallentamento al più polinomiale

CLASSI DI COMPLESSITÀ TEMPORALI

- La classe di complessità dei linguaggi riconoscibili da macchine di Turing deterministiche in tempo polinomiale è indicata con P ed è definita come

$$\mathrm{P} = \bigcup_{k \in \mathbb{N}} \mathrm{TIME}(n^k)$$

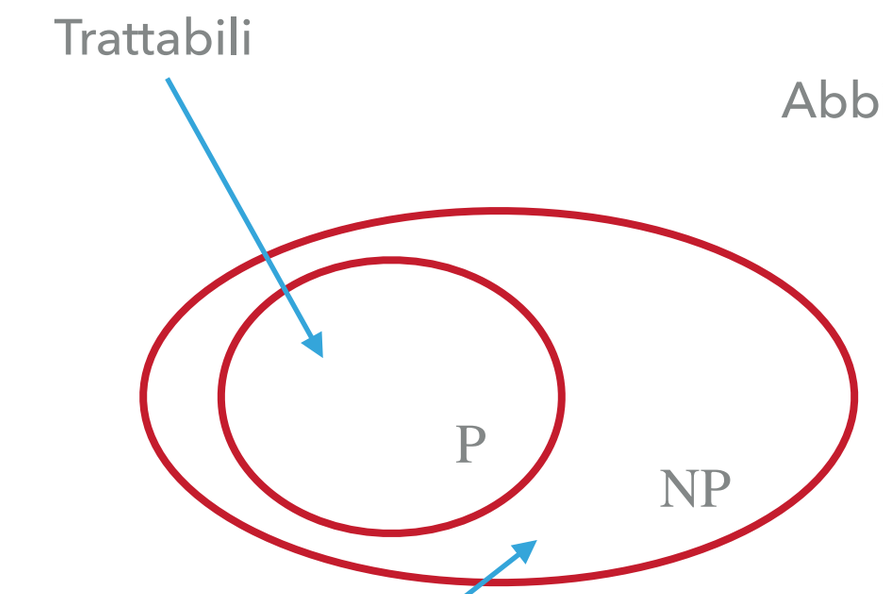
- La classe di complessità dei linguaggi riconoscibili da macchine di Turing **non** deterministiche in tempo polinomiale è indicata con NP ed è definita come

$$\mathrm{NP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(n^k)$$

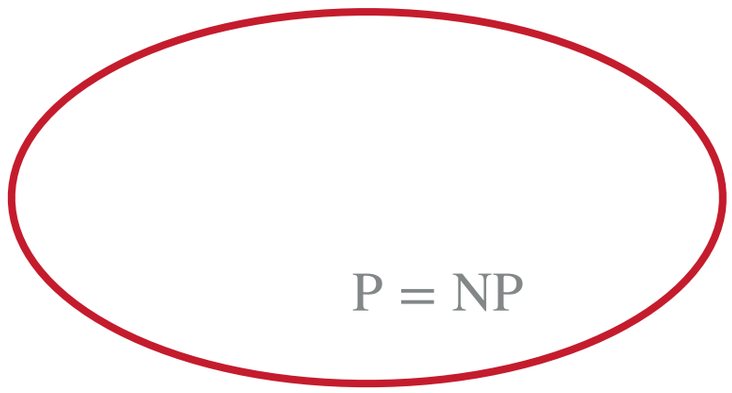
CLASSI DI COMPLESSITÀ TEMPORALI

- P rappresenta l'insieme dei problemi che sono risolvibili in modo "efficiente" (anche se un tempo n^{100} non è particolarmente efficiente)
- P è invariante rispetto ad ogni modello di calcolo deterministico ragionevole
- Quindi P è di interesse pratico. Se un linguaggio L è in P allora esiste un algoritmo che lo risolve in tempo polinomiale
- P è una classe robusta, che non varia al variare del modello di calcolo

CLASSI DI COMPLESSITÀ



Abbiamo due possibilità:



$P = NP$ o $P \neq NP$ è uno dei grandi problemi aperti dell'informatica teorica

L'impressione corrente è che $P \neq NP$

DEFINIZIONE ALTERNATIVA DI NP

- NP rappresenta una classe di linguaggi che sono **verificabili** in tempo polinomiale
- Vediamo una definizione alternativa di NP e mostriamo l'equivalenza con la definizione già data
- Verificare un linguaggio significa, in modo intuitivo, per ogni parola $w \in L$ avere una stringa aggiuntiva c detta certificato che possiamo usare per verificare che effettivamente $w \in L$

VERIFICATORI IN TEMPO POLINOMIALE

- Un **verificatore** di un linguaggio L è un algoritmo V tale per cui
  $L = \{w \ : \ V \text{ accetta } \langle w, c \rangle \text{ per qualche string } c\}$
- Se V richiede tempo polinomiale rispetto a w per accettare/rifiutare, allora V è un verificatore in tempo polinomiale per w
- Se esiste un verificatore in tempo polinomiale per L, allora L è verificabile in tempo polinomiale

VERIFICATORI IN TEMPO POLINOMIALE

- Dato che V deve eseguire in tempo polinomiale rispetto a w, ne segue che c, il certificato, deve essere di lunghezza polinomiale rispetto a w, altrimenti non avremmo il tempo di leggere tutto c
- Ma cosa è c?
- Come possiamo mostrare l'equivalenza delle due definizioni di NP?

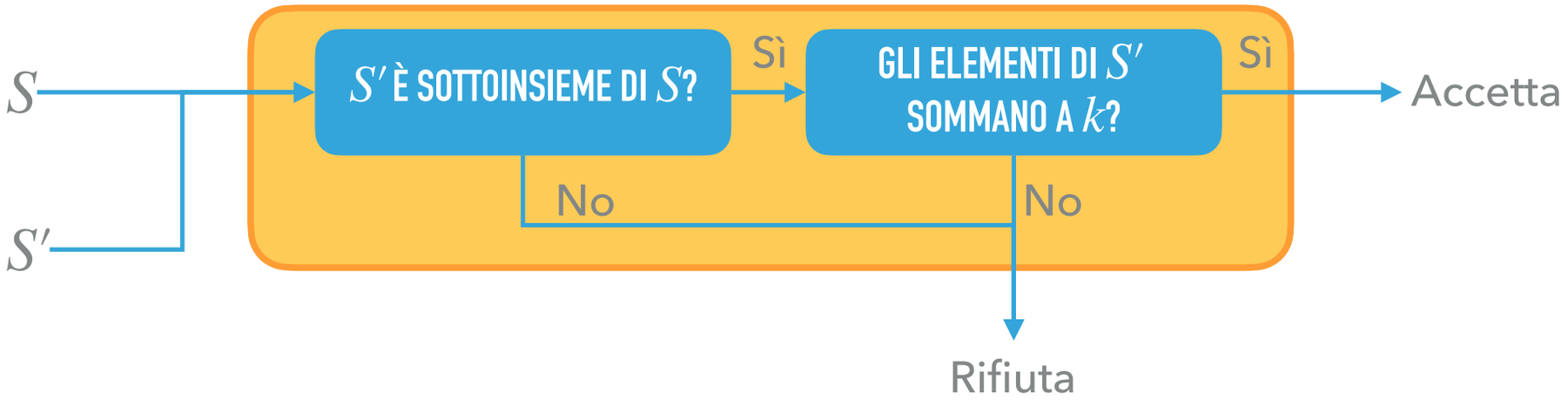
VERIFICATORI IN TEMPO POLINOMIALE: ESEMPIO

Insieme di numeri interi: $S = \{s_1, s_2, \ldots, s_m\}$

Scegliamo il seguente linguaggio: $L = \{S \mid \text{esiste } S' \subseteq S \text{ con } \sum_{s \in S'} s = k\}$

Dato un insieme S come possiamo trovare un certificato che mostri che $S \in L$?

Un buon candidato è S' i cui elementi hanno somma k



Questa verifica può essere compiuta in tempo polinomiale, quindi L è verificabile in tempo polinomiale

EQUIVALENZA TRA LE DEFINIZIONI DI NP

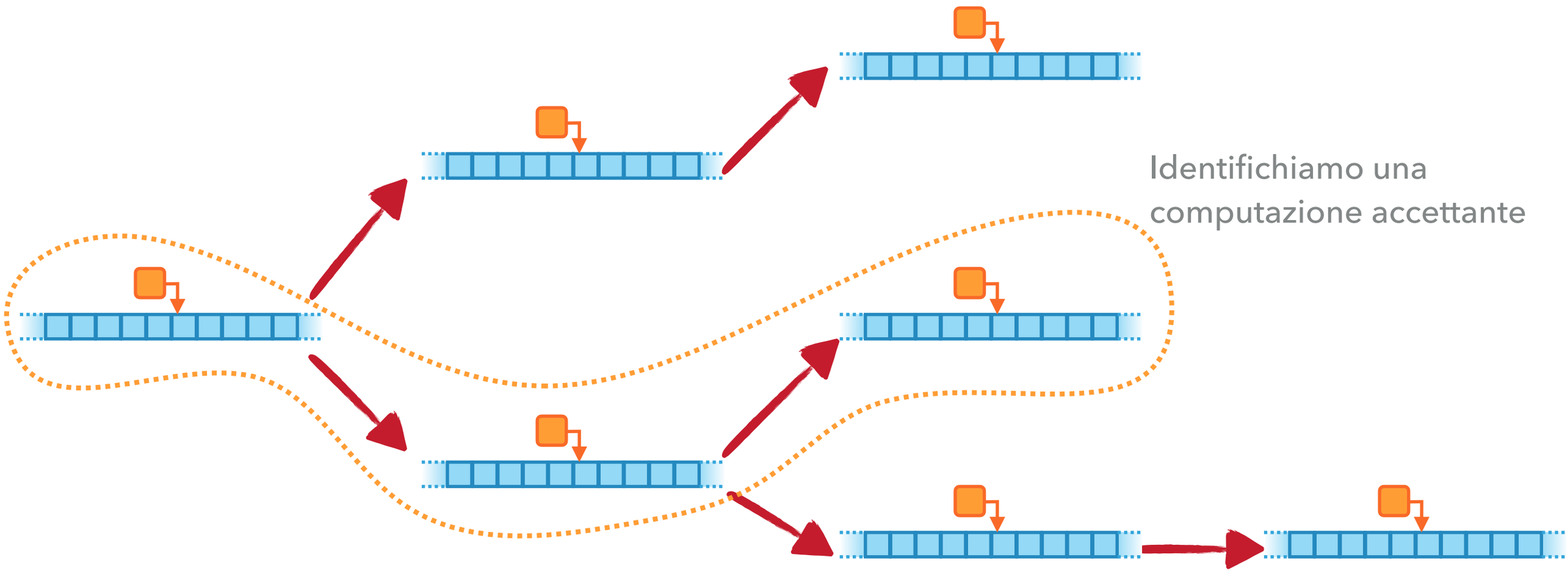
- Supponiamo di avere una NDTM M che lavora in tempo polinomiale, costruiamo un verificatore V che lavora in tempo polinomiale per lo stesso linguaggio deciso da M
- Se su input w la macchina M accetta, significa che esiste una computazione accettante $C_1, \ldots, C_k$ di lunghezza polinomiale rispetto a w
- Per ogni passaggio C_i a C_{i+1} è stata applicata una transizione nella forma (q, σ, M) con $q \in Q$, $\sigma \in \Sigma$ e $M \in \{ \leftarrow , \rightarrow \}$

EQUIVALENZA TRA LE DEFINIZIONI DI NP

- Usiamo come certificato c la sequenza di transizioni applicate lungo tutta la computazione (sono in numero polinomiale)
- Queste transizioni ci identificano una specifica computazione della NDTM M
- Il verificatore deve solo simulare quella computazione, senza fare scelte non deterministiche, dato che le transizioni da fare sono tutte in c e verificare che la computazione accetti

EQUIVALENZA TRA LE DEFINIZIONI DI NP

Macchina non deterministica



Identifichiamo una computazione accettante

Il verificatore deve solo fare un “replay” della computazione e verificare che accetti

EQUIVALENZA TRA LE DEFINIZIONI DI NP

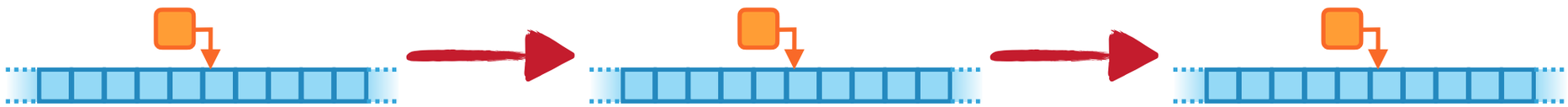
- Abbiamo mostrato che per ogni linguaggio accettato da una NDTM che lavora in tempo polinomiale possiamo costruire un verificatore polinomiale per lo stesso linguaggio
- Dobbiamo mostrare anche l'inclusione in senso inverso
- Sfruttiamo il fatto che possiamo usare una NDTM per scrivere un "certificato" sul nastro e, se esiste un certificato che ci permette di accettare, questo sarà presente in una computazione

EQUIVALENZA TRA LE DEFINIZIONI DI NP

- Sia V un verificatore in tempo polinomiale per L
- Assumiamo V esegua in tempo n^k
- Costruiamo una NDTM che su input w fa due cose:
  - Genera in modo non deterministico un certificato c di lunghezza al più n^k
  - Simula V su input $\langle w, c \rangle$ e accetta se V accetta e rifiuta se V rifiuta

EQUIVALENZA TRA LE DEFINIZIONI DI NP

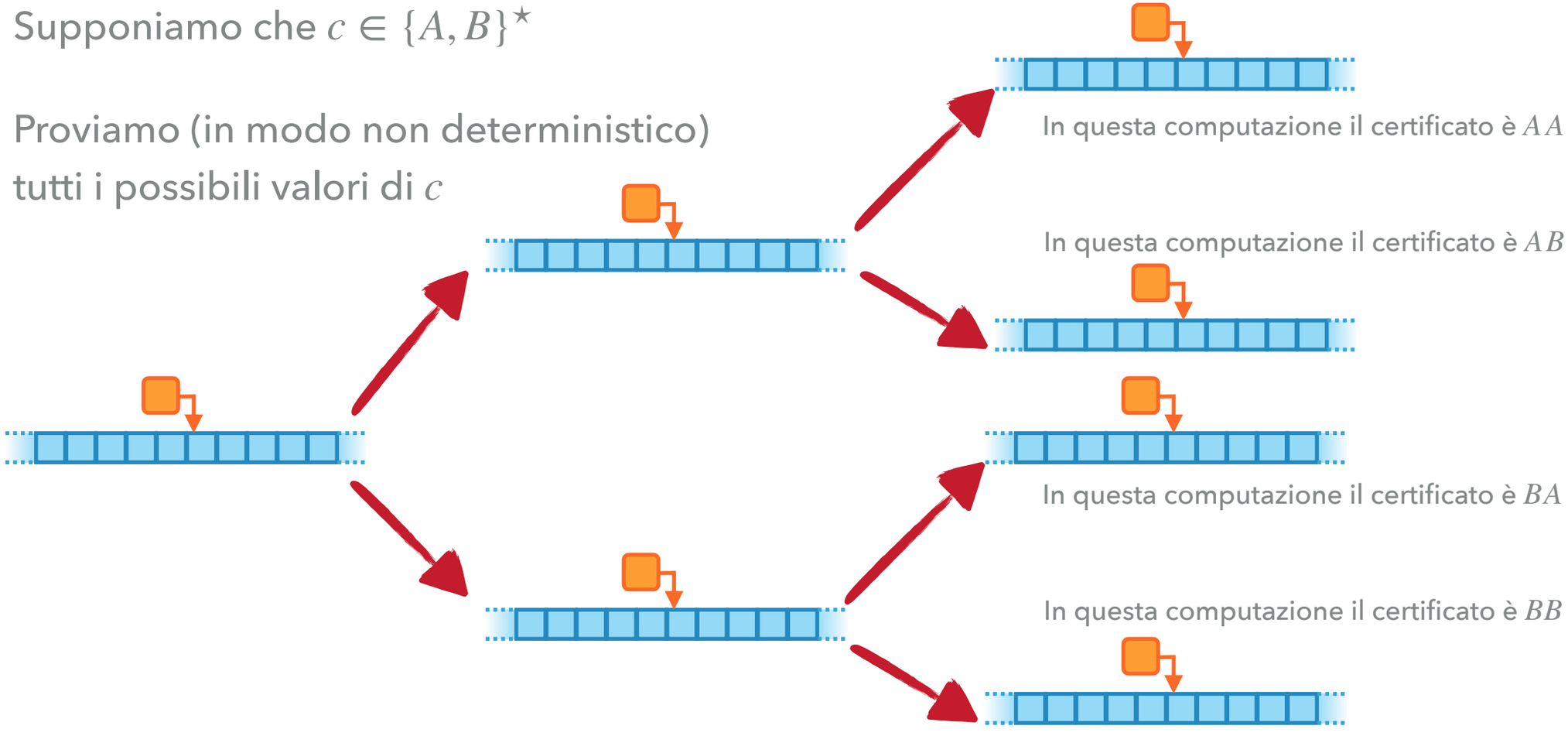
Sappiamo che V accetta su input $\langle w, c \rangle$ se $w \in L$



Ma conosciamo solo w e non c

Supponiamo che $c \in \{A, B\}^\star$

Proviamo (in modo non deterministico) tutti i possibili valori di c



EQUIVALENZA TRA LE DEFINIZIONI DI NP

- Abbiamo mostrato che per ogni linguaggi per il quale esiste un verificatore polinomiale possiamo costruire una NDTM che decide lo stesso linguaggio in tempo polinomiale
- Quindi le due definizioni di NP sono equivalenti
- Chiedersi se $\mathrm{P} = \mathrm{NP}$ può essere visto come chiedersi se il certificato sia necessario

SPAZIO DI CALCOLO PER TM

- Data una macchina di Turing M ed un input $w \in \Sigma^\star$, indichiamo con $s(w)$ il massimo numero di celle non blank durante la computazione di M su input w. Questo rappresenta la spazio richiesto da M su input w

- Per tutti gli $n \in \mathbb{N}$ definiamo $s(n)$ come il massimo dello spazio richiesto da M per riconoscere una parola $w \in \Sigma^\star$ di lunghezza n (i.e., $|w| = n$):

$$s(n) = \max_{w \in \Sigma^\star,\ |w|=n} s(w)$$

SPAZIO DI CALCOLO PER TM NON DETERMINISTICHE

- Come per il tempo di calcolo, anche per lo spazio nelle TM non deterministiche c'è da gestire la presenza di più computazioni

- Su una parola w di input definiamo $s(w)$ il **massimo** spazio utilizzato tra tutte le possibili computazioni

- Come prima, $s(n)$ rappresenta il massimo $s(w)$ tra tutti le parole w di lunghezza n

CLASSI DI COMPLESSITÀ SPAZIALI

- Definiamo come $\mathrm{SPACE}(f(n))$ l'insieme di tutti i **linguaggi** per i quali esiste una macchina di Turing deterministica che li decide in spazio $O(f(n))$
- Similmente, definiamo come $\mathrm{NSPACE}(f(n))$ l'insieme di tutti i linguaggi per i quali esiste una macchina di Turing non-deterministica che li decide in spazio $O(f(n))$
- Chiaramente $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$ per qualsiasi funzione $f(n)$

CLASSI DI COMPLESSITÀ SPAZIALI

- La classe di complessità dei linguaggi riconoscibili da macchine di Turing deterministiche in **spazio** polinomiale è indicata con PSPACE ed è definita come

$$\mathrm{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathrm{SPACE}(n^k)$$

- La classe di complessità dei linguaggi riconoscibili da macchine di Turing **non** deterministiche in **spazio** polinomiale è indicata con NPSPACE ed è definita come

$$\mathrm{NPSPACE} = \bigcup_{k \in \mathbb{N}} \mathrm{NPSPACE}(n^k)$$

RELAZIONE TRA CLASSI DI COMPLESSITÀ SPAZIALI E TEMPORALI

- Se $f(n)$ è almeno lineare possiamo trovare una relazione tra classi di complessità spaziali e temporali:
  - Per utilizzare $s(n)$ celle la TM deve scrivere su $s(n) - n$ di esse almeno una volta (n celle sono già occupate dall'input)
- Dato che una TM può scrivere al più una cella per ogni passo, abbiamo che $\mathrm{TIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$ e che $\mathrm{NTIME}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$