Non-deterministic Finite Automata

Lecture 4

Today

What is *non-determinism*?

NFAs

NFAs *vs.* DFAs

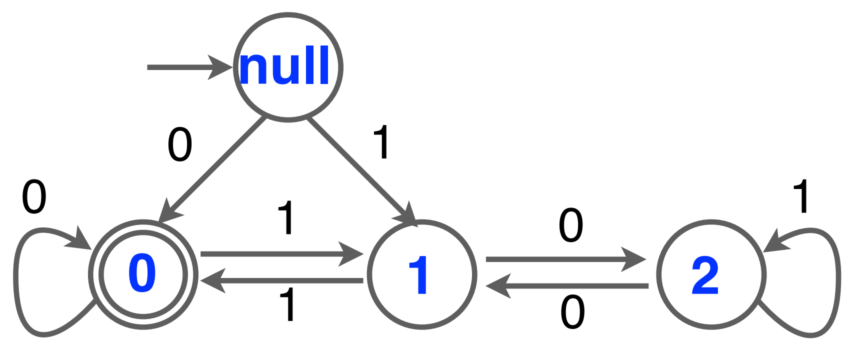
NFAs with ε-moves

Closure Properties of
class of languages accepted by NFAs/DFAs

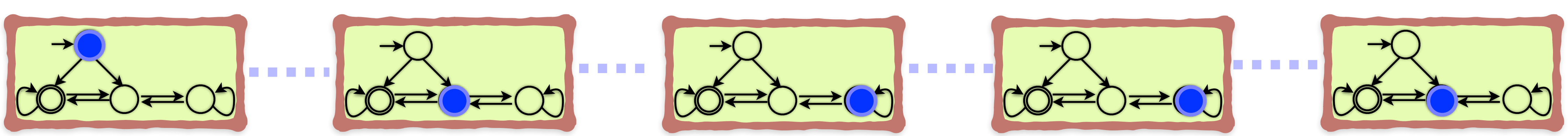
Tracking Computation

current state and
remaining input

A computation's *configuration* evolves in each time-step



on input 1010



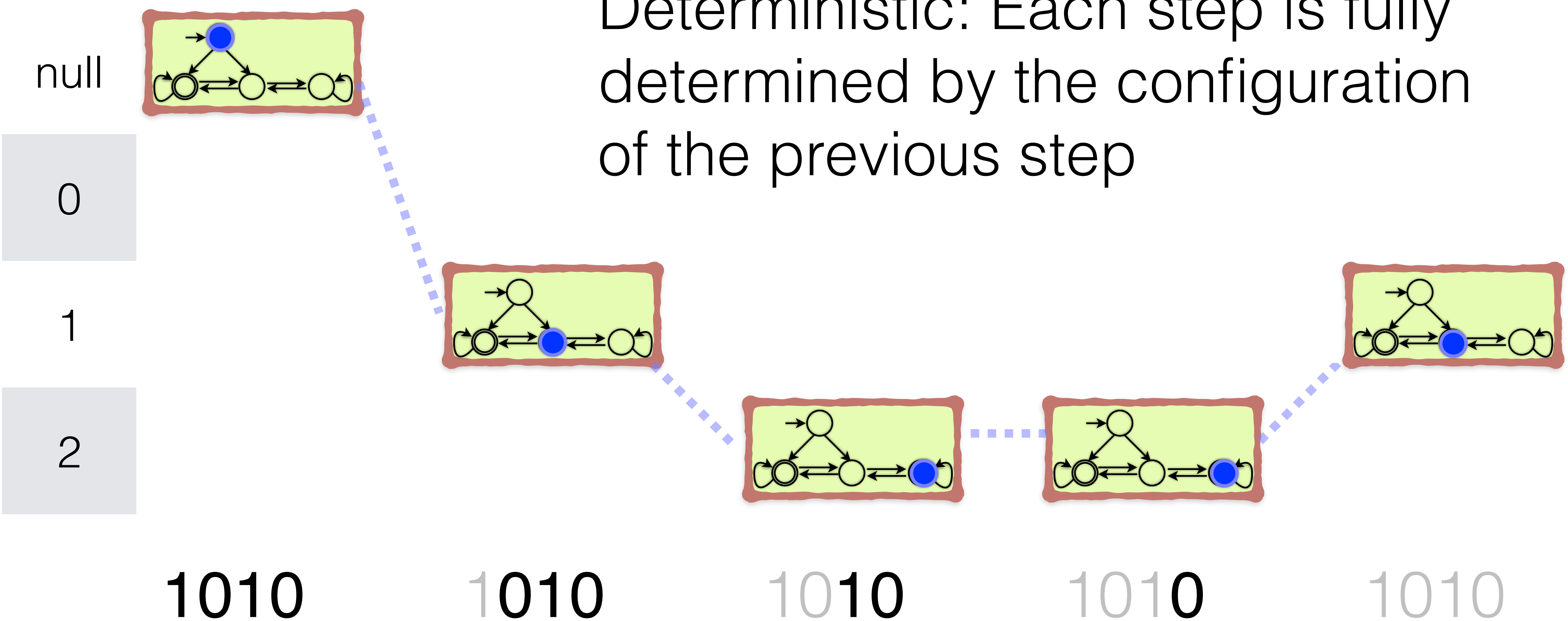
1010 1010 1010 1010 1010

Tracking Computation

current state and remaining input

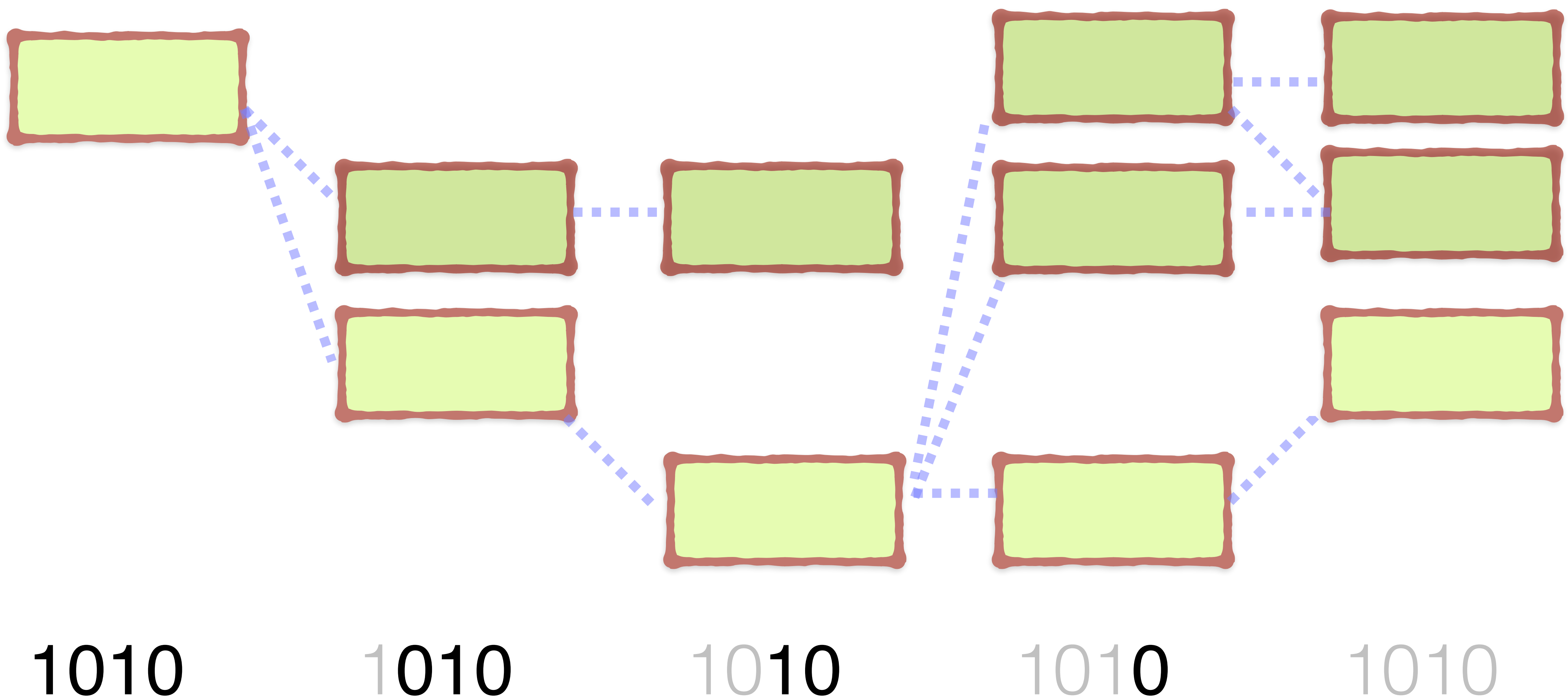
A computation's *configuration* evolves in each time-step

Deterministic: Each step is fully determined by the configuration of the previous step



Non-Determinism

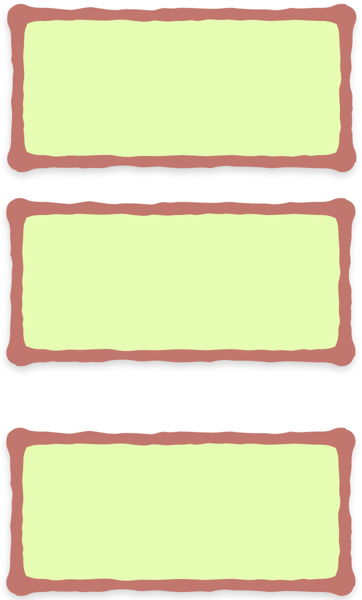
At each step the computation is allowed to proceed in multiple ways (zero, one or more)



Non-Determinism

At each step the computation is allowed to proceed in multiple ways (zero, one or more)

At the end, left with zero or more possible configurations (each with its own output)



What is the outcome of the entire computation?

Accept iff at least one configuration accepts

How about other rules — e.g., accept iff all the configurations accept, accept iff a majority accepts etc.?

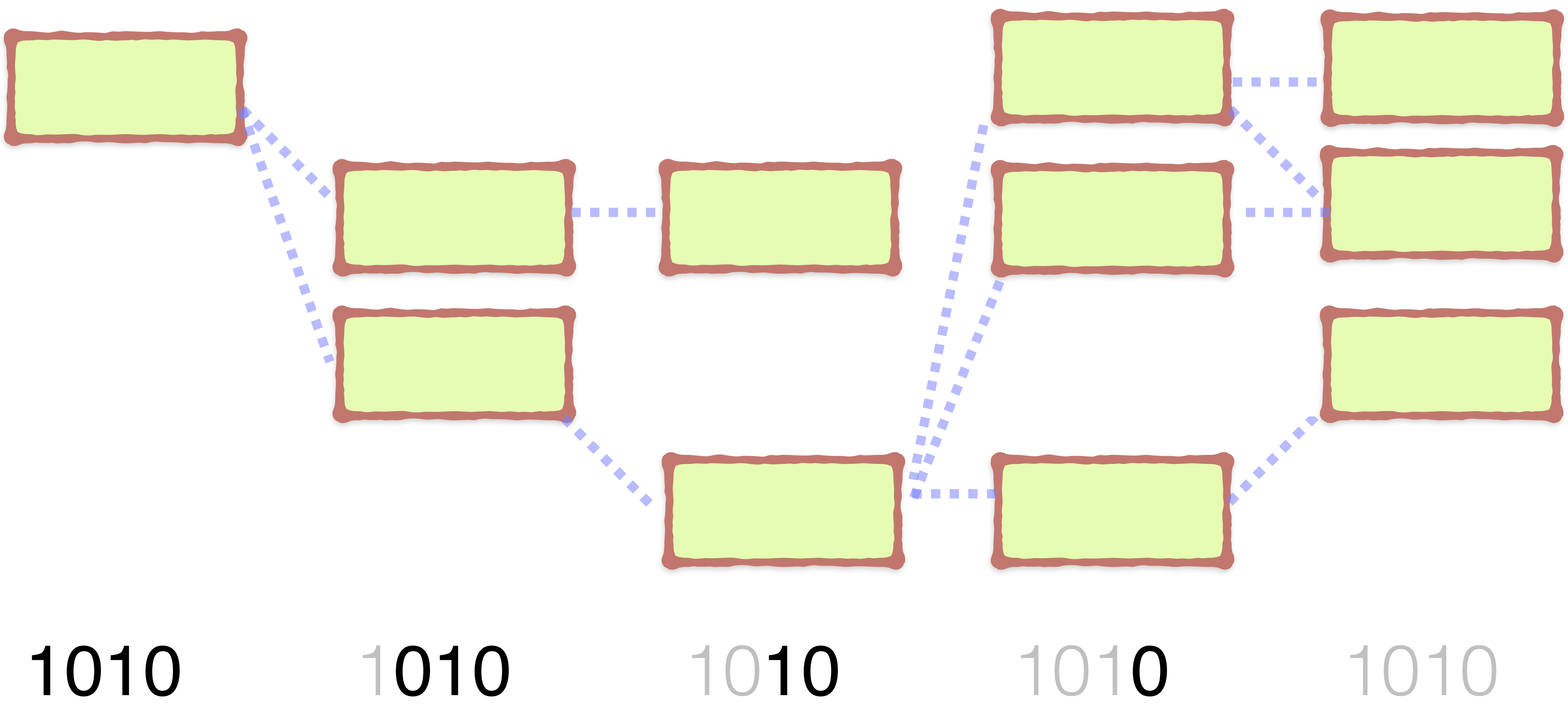
Sure, but they have other names — e.g., co-non-deterministic computation, probabilistic computation etc.

Non-Determinism

Unrealistic model of computation!

will see some uses today

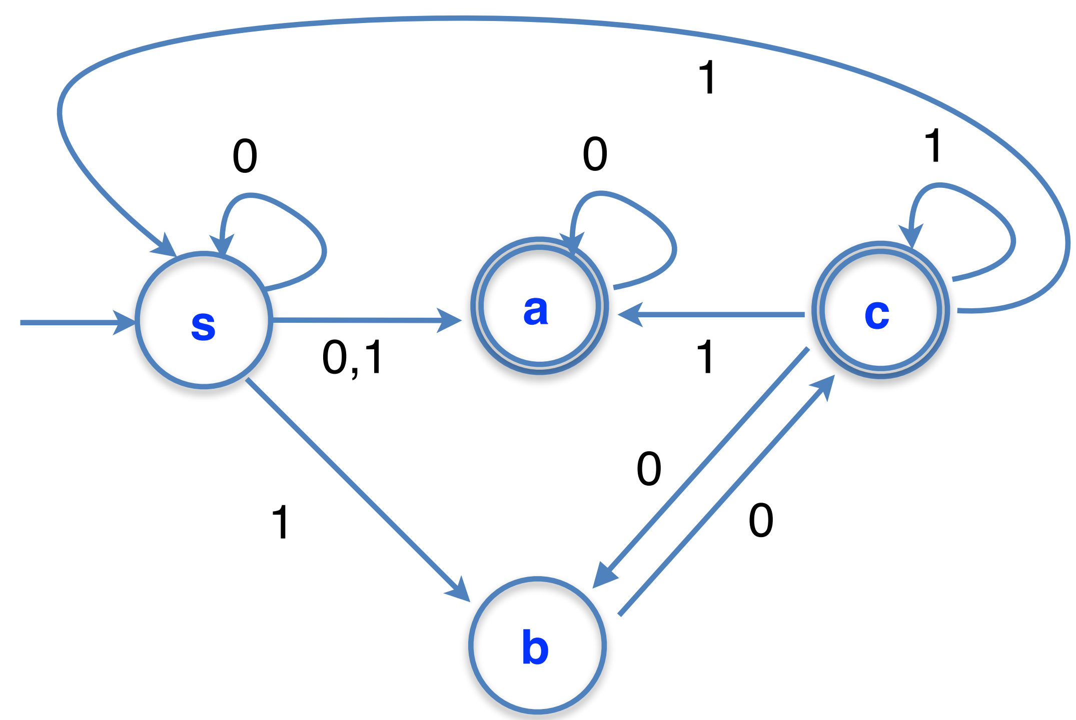
A very powerful “high-level” programming language



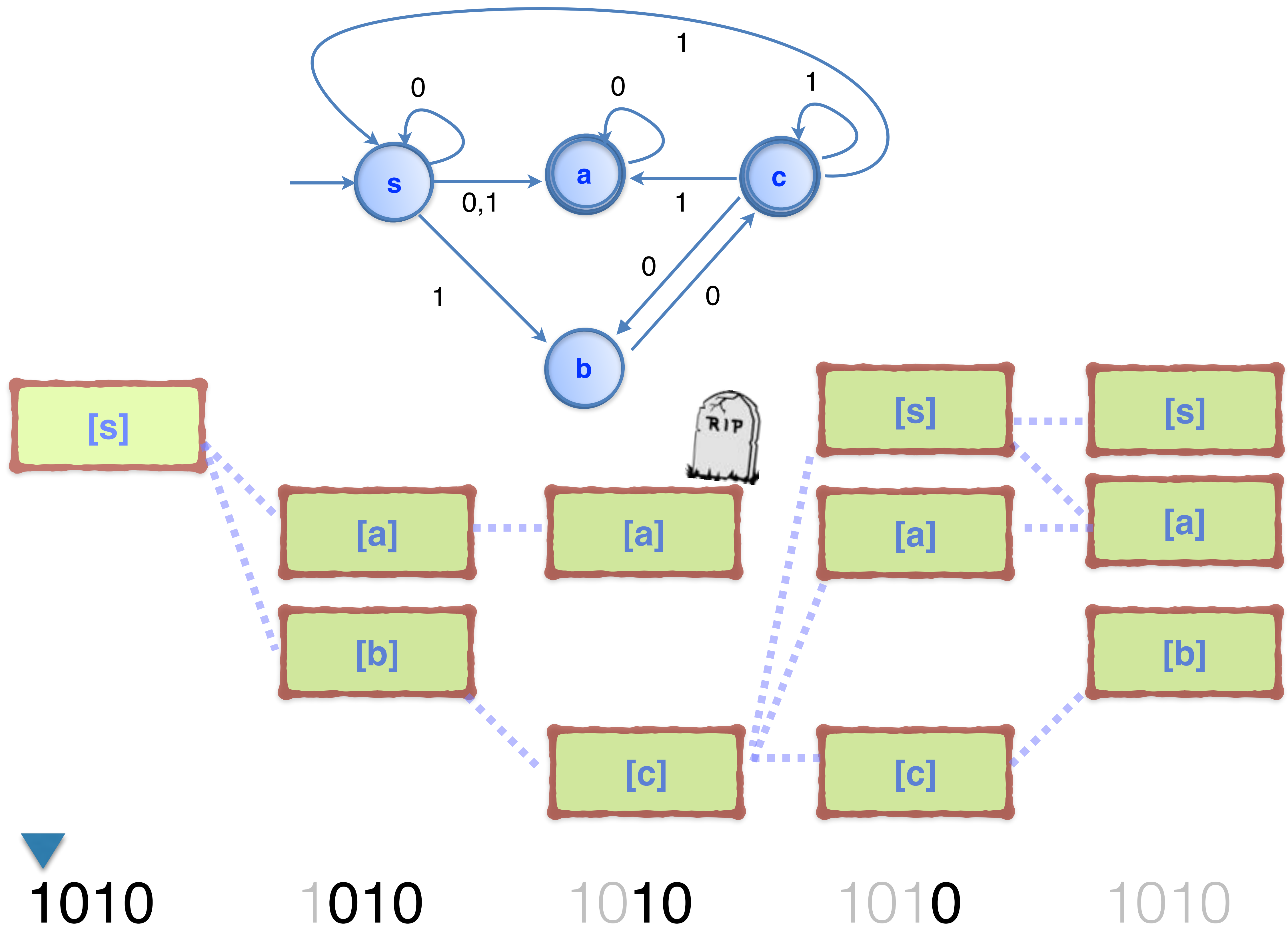
Non-Deterministic FA

What can be non-deterministic about an FA?

At a given state, on a given input, multiple “next-states”

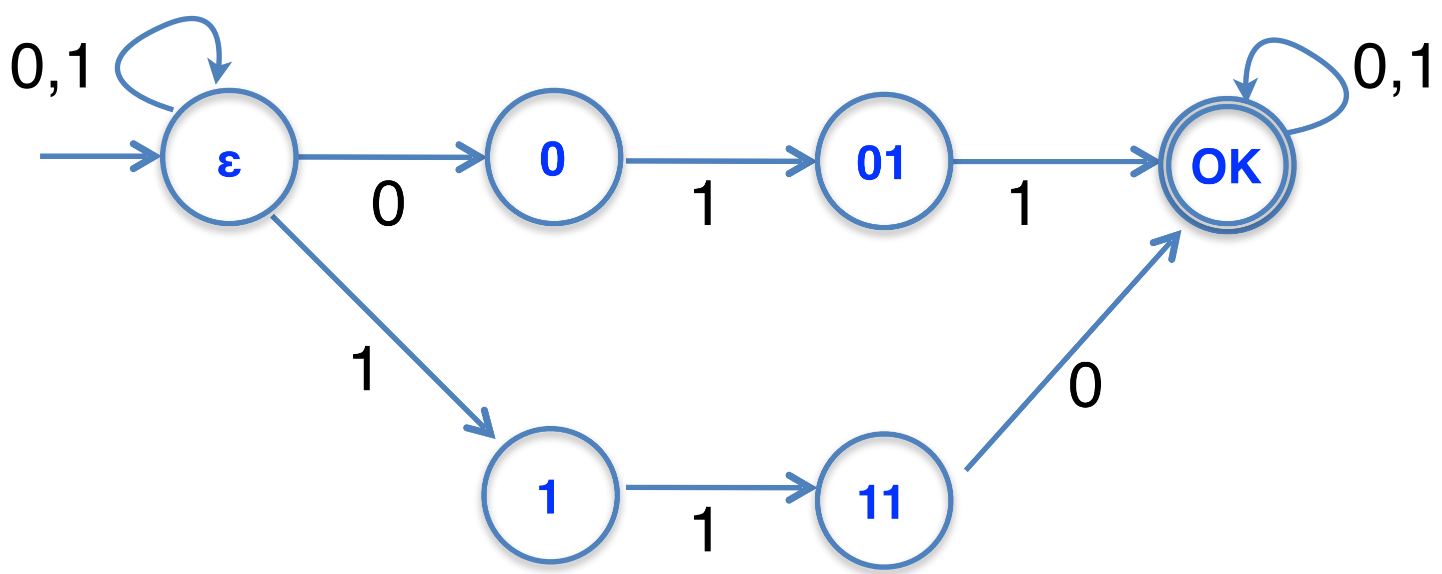


Non-Deterministic FA



NFA : Examples

Design an NFA to recognize
$L(M) = \{w \mid w$ contains 011 or 110 $\}$

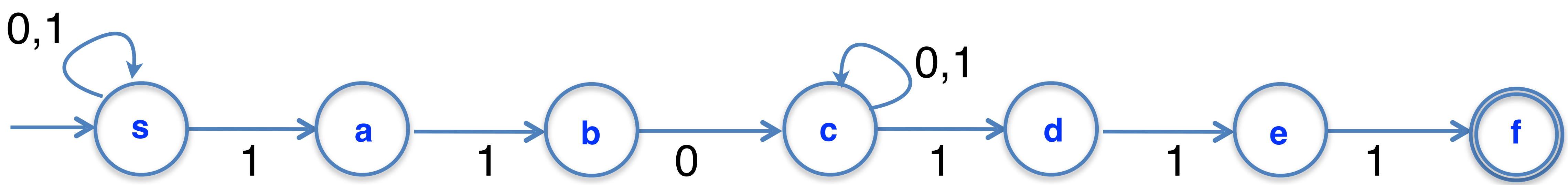


For any input string, if it contains 011 or 110, then there is *some* computation path, that ends in the final state

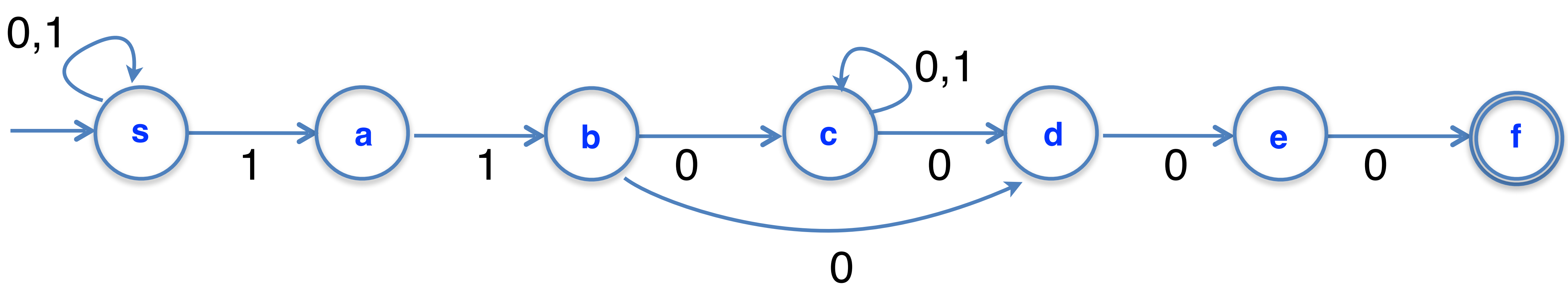
And vice versa

NFA : Examples

Design an NFA to recognize
$L(M) = \{w \mid w$ has the substring 110 and ends in 111 $\}$



Design an NFA to recognize
$L(M) = \{w \mid w$ has the substring 110 and ends in 000 $\}$



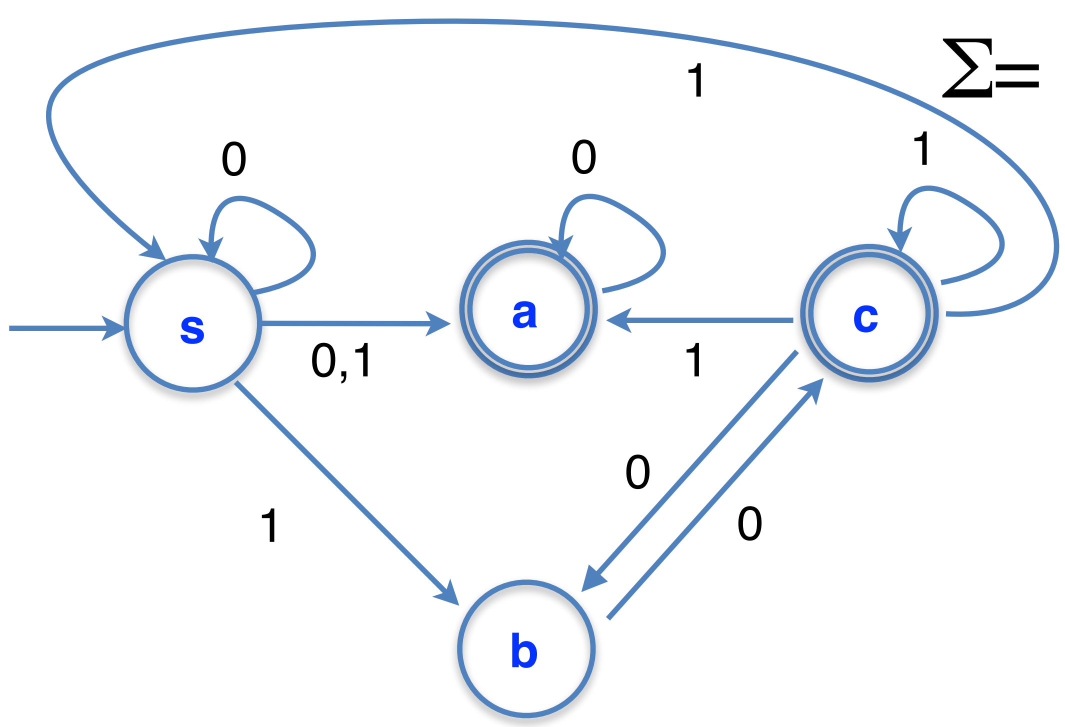
NFA : Formally

Similar to a DFA : $N = (\Sigma, Q, \delta, s, F)$

Σ: alphabet Q: state space s: start state F: set of accepting states

$$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$$

$\delta(q, a)$ = <u>*set*</u> of states N could move to from q, on input a



$\Sigma = \{0,1\}$. $Q = \{$ **s**,**a**,**b**,**c** $\}$. $s =$ **s**. $F = \{$ **a**,**c** $\}$

$\delta($**s**$,0) = \{$**s**,**a**$\}$, $\delta($**s**$,1) = \{$**a**,**b**$\}$

$\delta($**a**$,0) = \{$**a**$\}$, $\delta($**a**$,1) = \emptyset$

$\delta($**b**$,0) = \{$**c**$\}$, $\delta($**b**$,1) = \emptyset$

$\delta($**c**$,0) = \{$**b**$\}$, $\delta($**c**$,1) = \{$**s**,**a**,**b**$\}$

NFA : Formally

Similar to a DFA : $N = (\Sigma, Q, \delta, s, F)$

Σ: alphabet Q: state space s: start state F: set of accepting states

$$\delta : Q \times \Sigma \to \mathcal{P}(Q)$$

$\delta(q, a)$ = set of states N could move to from q, on input a

$q \overset{w}{\rightsquigarrow} p$ for $w = a_1 \ldots a_t$ if $\exists\ q_1, \ldots, q_{t+1}$, such that

$q_1 = q$, $q_{t+1} = p$, and $\forall\ i \in [\ 1, t\ ]$, $q_{i+1} \in \delta(q_i, a_i)$

N **accepts** w if $s \overset{w}{\rightsquigarrow} p$ for a $p \in F$

$L(N) = \{\ w \mid N$ accepts $w\ \}$

Same definition for DFAs, but with "=" here

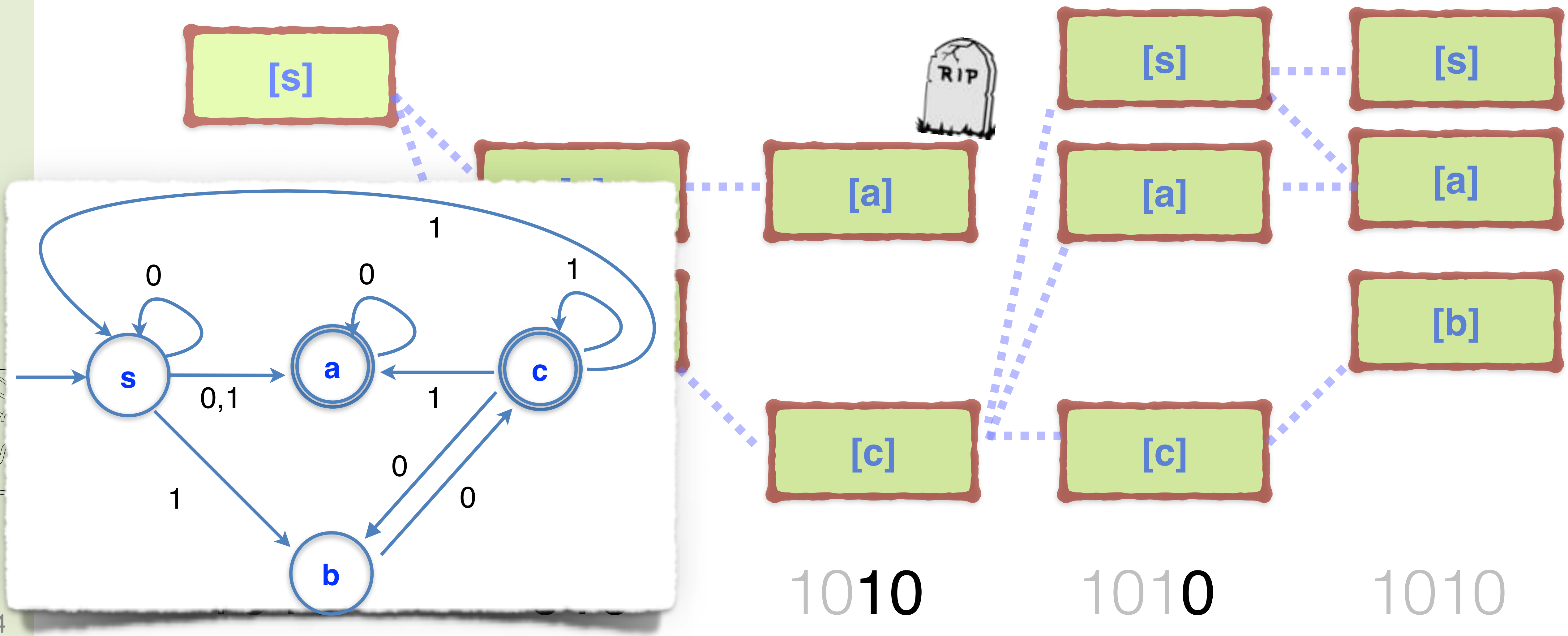
Keeping Track of an NFA

Given a *current set* of states, the next *set* of states:

$$\delta^\dagger : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$$

e.g., $\delta^\dagger(\{\mathbf{a},\mathbf{c}\},1) = \{\mathbf{s},\mathbf{a},\mathbf{c}\}$

$\delta^\dagger(\{\mathbf{s},\mathbf{a},\mathbf{c}\},0) = \{\mathbf{s},\mathbf{a},\mathbf{b}\}$



Keeping Track of an NFA

Given a *current set* of states, the next *set* of states:

$$\delta^\dagger : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$$

Formally: $\delta^\dagger(T, a) = \cup_{q \in T}\ \delta(q,a)$

How about the set of states that a *string* leads to?

$$\boxed{\begin{array}{c} \delta^{\dagger *}(T, \varepsilon) = T \\ \delta^{\dagger *}(T, au) = \delta^{\dagger *}(\delta^\dagger(T, a), u) \end{array}}$$

Exercise:
Prove $\delta^{\dagger *}(T, w)$
$= \{ p \mid q \overset{w}{\rightsquigarrow} p, q \in T\}$

$$s^\dagger = \{s\}, F^\dagger = \{ T \mid T \cap F \neq \emptyset \}$$

$$L(N) = \{ w \mid \delta^{\dagger *}(s^\dagger, w) \in F^\dagger\}$$

Keeping Track of an NFA Using a DFA!

NFA: $N = (\Sigma, Q, \delta, s, F)$ DFA: $M_N = (\Sigma, \mathcal{P}(Q), \delta^\dagger, s^\dagger, F^\dagger)$

$\delta^\dagger : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$: transition for set of states on a symbol

$\delta^{\dagger *} : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$: transition for set of states on a string

$$\boxed{\begin{aligned} \delta^{\dagger *}(T, \varepsilon) &= T \\ \delta^{\dagger *}(T, au) &= \delta^{\dagger *}(\delta^\dagger(T, a), u) \end{aligned}}$$

$$s^\dagger = \{s\}, F^\dagger = \{\, T \mid T \cap F \neq \emptyset \,\}$$

$$L(N) = \{\, w \mid \delta^{\dagger *}(s^\dagger, w) \in F^\dagger \} = L(M_N)$$

NFAs & DFAs

NFA is a more general model than DFA

Any DFA can be trivially converted to an equivalent NFA (i.e., accepting the same language)

by treating the output of the DFA's transition function as a *singleton set*

Any NFA can be converted to an equivalent DFA

but this *may* exponentially increase the number of states

So, not a good way to construct a DFA in practice!

NFAs & DFAs

Equivalence with NFAs is still very useful to understand DFAs!

e.g., We claimed: Every regular language has a DFA

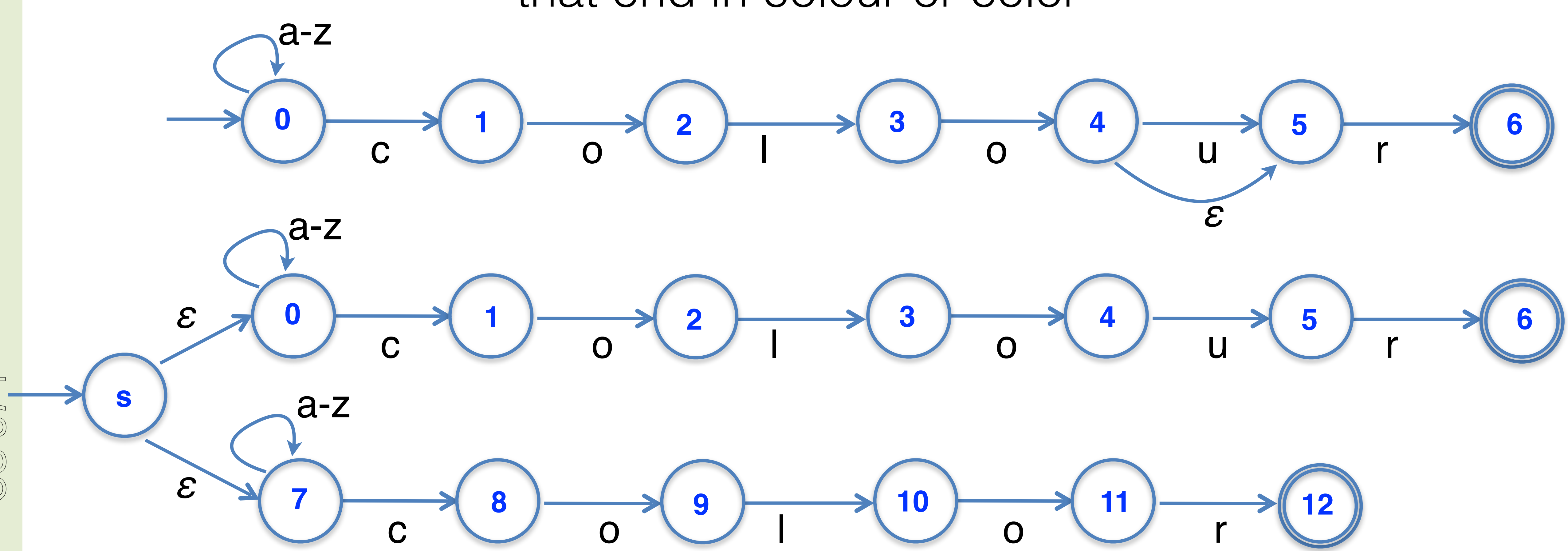
We'll prove this (next time) by showing that every regular language has an NFA

We need one more additional feature in an NFA for that (and other applications)

NFAs with ε-Moves

In an ε-move an NFA changes its state by following an arc labeled ε, *without consuming an input symbol*

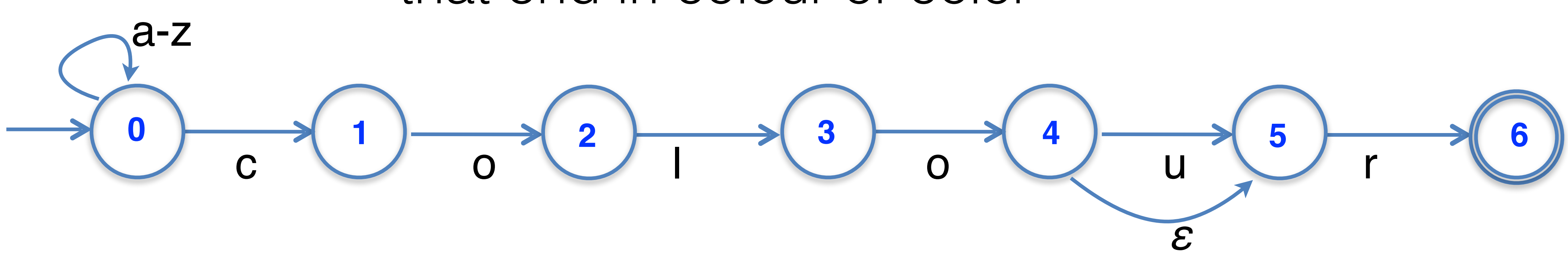
e.g., an NFA that accepts the language of all strings over {a,..,z} that end in colour or color



NFAs with ε-Moves

In an ε-move an NFA changes its state by following an arc labeled ε, *without consuming an input symbol*

e.g., an NFA that accepts the language of all strings over {a,..,z} that end in colour or color



For an NFA with ε-moves, $q \overset{w}{\rightsquigarrow} p$

if $\exists\, a_1,\ldots, a_t \in \Sigma \cup \{\varepsilon\}$ and $\exists\, q_1,\ldots,q_{t+1}$, such that

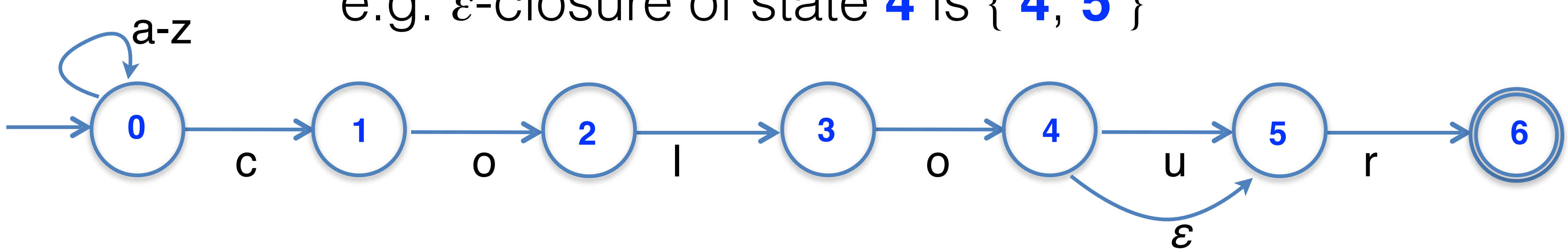
$w = a_1\ldots a_t$, $q_1 = q$, $q_{t+1} = p$, and $\forall\, i \in [\,1, t\,]$, $q_{i+1} \in \delta(q_i,a_i)$

NFAs with ε-Moves

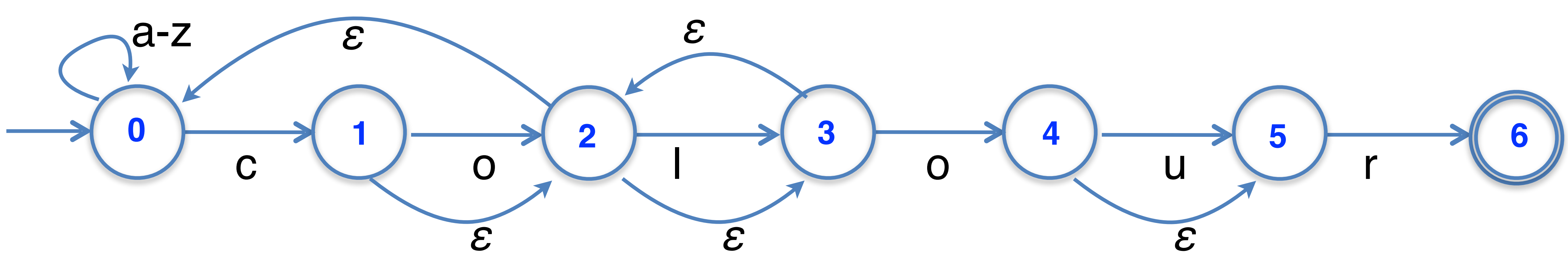
$$\delta : Q \times (\, \Sigma \cup \{\varepsilon\} \,) \to \mathcal{P}(Q)$$

ε-closure of a state q: all states reachable from q without consuming any input

e.g. ε-closure of state **4** is { **4**, **5** }



e.g., ε-closure of state **1** is { **1**, **2**, **3**, **0** }



NFAs with ε-Moves

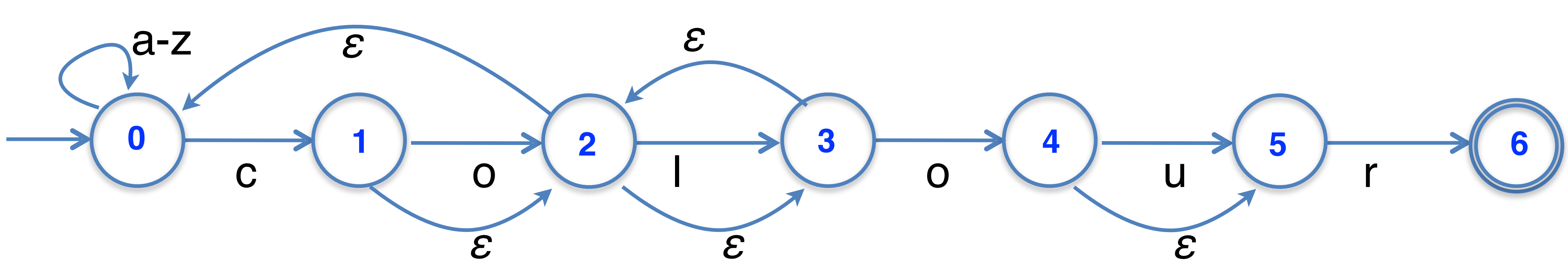
$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q)$$

ε-closure of a set of states T: all states reachable from *some* state T in without consuming any input

e.g. Below, ε-closure of the set {**3**,**4**} is { **3**,**2**,**0**,**4**,**5** }

$$C_\varepsilon : \mathcal{P}(Q) \rightarrow \mathcal{P}(Q)$$

$$C_\varepsilon(S) = \{ \, p \mid q \overset{\varepsilon}{\rightsquigarrow} p \text{ for some } q \in S \, \}$$



ε-Moves is Syntactic Sugar

Can easily modify an NFA with ε-moves N,
to get an NFA N_{new} without ε-moves

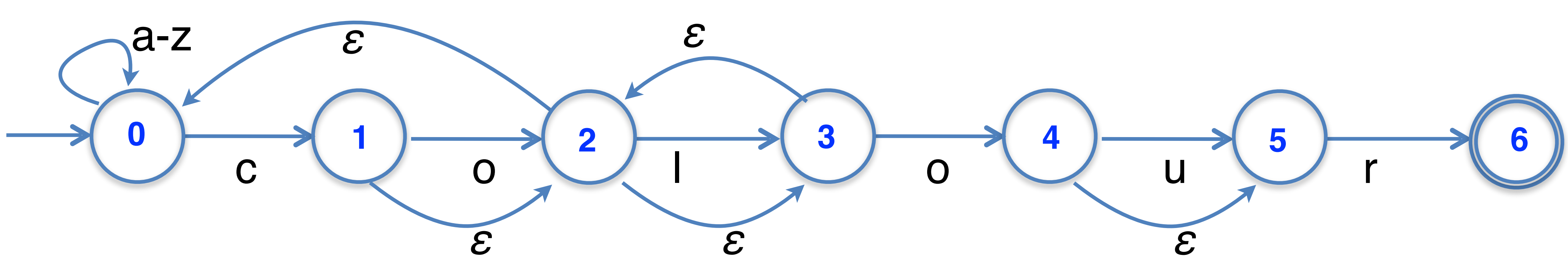
$$\delta_{\text{new}}(q, a) = C_\varepsilon(\ \delta(\ C_\varepsilon(\{q\}), a))$$

e.g.: $\delta_{\text{new}}(\mathbf{1},\text{o}) = \{\mathbf{0},\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5}\}$

Prove by induction:
for $|w| \geq 1$,
$$q \overset{w}{\rightsquigarrow}_N p \Leftrightarrow q \overset{w}{\rightsquigarrow}_{N\text{new}} p$$

$$F_{\text{new}} = F, \quad \text{if } C_\varepsilon(\{s\}) \cap F = \emptyset$$
$$F_{\text{new}} = F \cup \{s\}, \quad \text{otherwise.}$$

Theorem: $L(N) = L(N_{\text{new}})$



NFAs & DFAs

3 "equivalent" computational models: DFAs, NFAs w/o ε-moves, NFAs (w/ ε-moves)

Equivalent: the class of languages that can be computed in each model is the same

Because a "program" in one model can be "compiled" into one in any other model

There may be an "efficiency loss":

NFAs (w/ ε-moves) $\rightarrow$ NFAs w/o ε-moves :
Number of transitions can increase (polynomially)

NFAs w/o ε-moves $\rightarrow$ DFAs :
Number of state can increase (exponentially)