

# Automata over infinite alphabets: Investigations in Fresh-Register Automata

Nikos Tzevelekos

Queen Mary University of London

Andrzej Murawski & Steven Ramsay, University of Warwick

Radu Grigore, University of Oxford

University of Southampton, Sep 2015

*Supported by a Royal Academy of Engineering Research Fellowship*

# Why *infinite* alphabets

```
public void foo() {  
    // Create new list  
    List x = new ArrayList();  
  
    x.add(1); x.add(2);  
    Iterator i = x.iterator();  
    Iterator j = x.iterator();  
    i.next(); i.remove(); j.next();  
}
```

# Why *infinite* alphabets

```
public void foo() {  
    // Create new list  
    List x = new ArrayList();  
  
    x.add(1); x.add(2);  
    Iterator i = x.iterator();  
    Iterator j = x.iterator();  
    i.next(); i.remove(); j.next();  
}
```

Computation that uses resources can go beyond finite alphabets (cf. modelling/analysis of programs)

Lots of interest also from XML model-checking community

# What this talk is about

This talk is about **fresh register automata**

which are an automata model over infinite alphabets akin to finite-state automata:

- finite-state
- registers
- freshness

We examine the complexity of their **bisimilarity** problem and derive bounds for different subcases/variants

We also look at **pushdown-stack** extensions

# Automata theory in infinite alphabets

Let  $\Sigma = \{a_1, a_2, \dots, a_n, \dots\}$  be an infinite alphabet of names

*can only be compared for equality*

# Automata theory in infinite alphabets

Let  $\Sigma = \{a_1, a_2, \dots, a_n, \dots\}$  be an **infinite** alphabet of **names**

- examine languages over  $\Sigma^*$ 
  - or, languages over  $(F \cup \Sigma)^*$
  - or, languages over  $(F \times \Sigma)^*$ 
    - usually called *data words* (XML)
- look for notions of regularity, CFGs, etc.
- devise effective algorithms for reachability, membership, etc.

can only be compared for equality

a finite set of **constants**

# Register Automata (RA)

Let  $\Sigma = \{a_1, a_2, \dots, a_n, \dots\}$  be an **infinite** alphabet of **names**



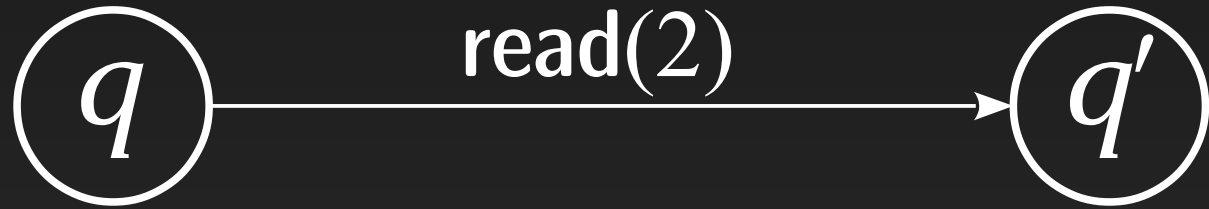
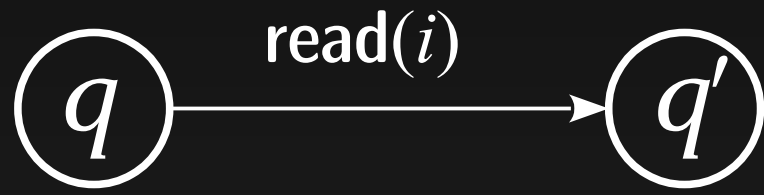
*finitely many*  
(say  $R$ ) **registers**

*registers store names*

Label  $\lambda$  of the form:

- **read**( $i$ ),  $i \in \{1, \dots, R\}$
- **fresh**( $i$ ),  $i \in \{1, \dots, R\}$

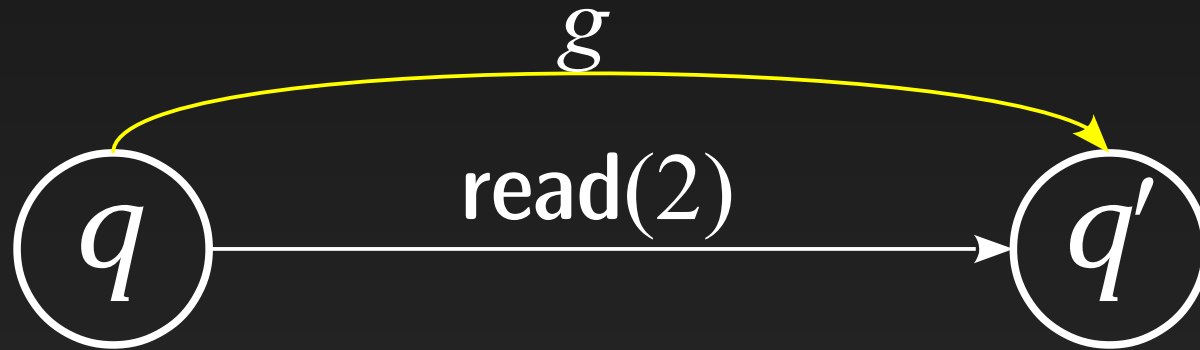
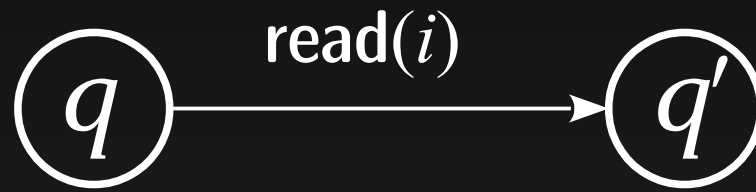
Transitions:



$a$	$g$	$b$
-----	-----	-----



Transitions:



$a$	$g$	$b$
-----	-----	-----

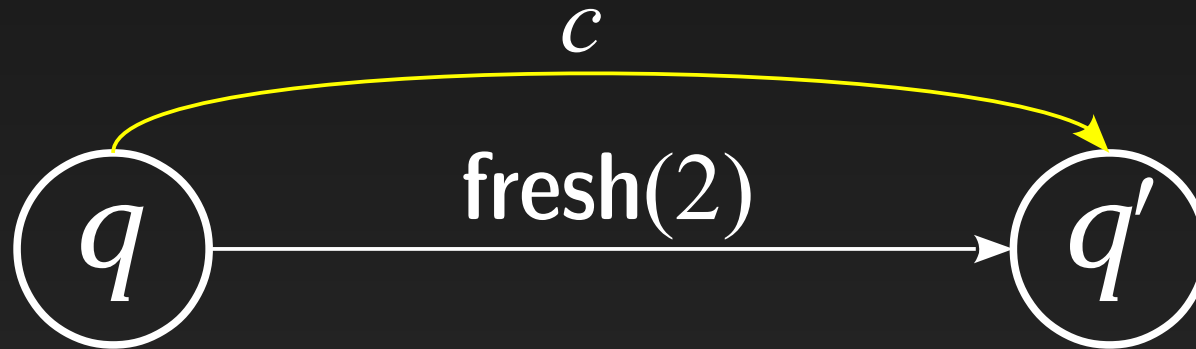
$a$	$g$	$b$
-----	-----	-----

Transitions:



$a$	$g$	$b$
-----	-----	-----

Transitions:

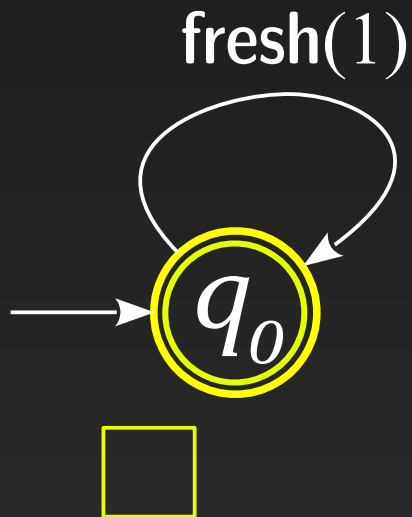


*fresh*

# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



*a*

---

# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



*ab*

---

# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



*abc*

---

# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



abca



# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



*abcd*

---

# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



*abcade*

# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



*abcadeb*

---

# Example

$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*



*abcadebagcab*

# Example

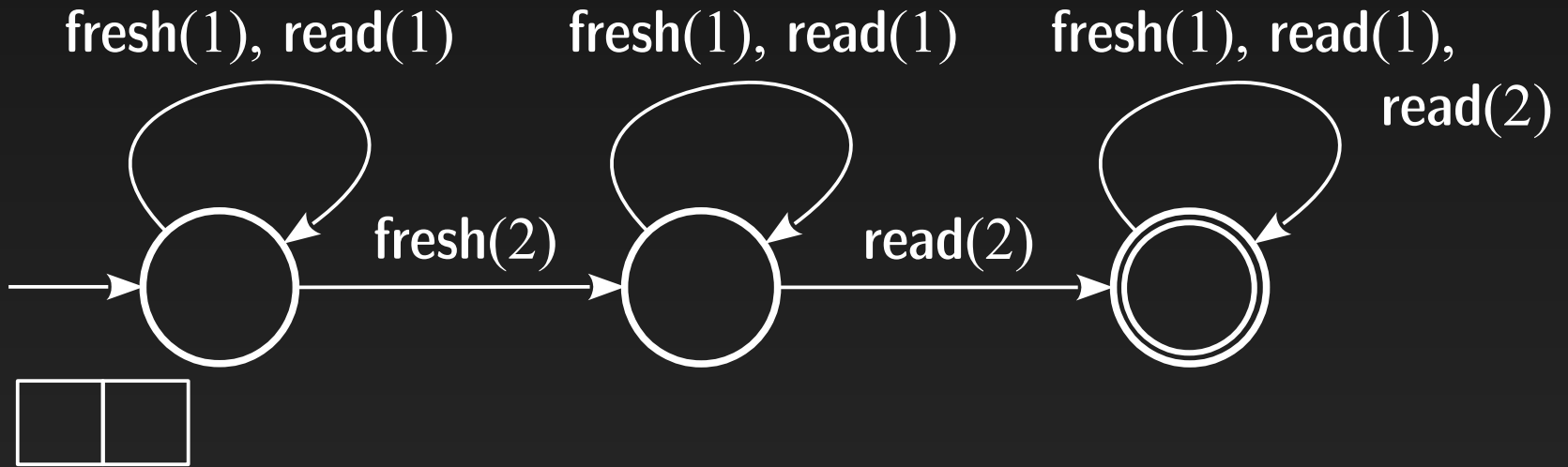
$$L_1 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*(all strings where each name is distinct from its predecessor)*

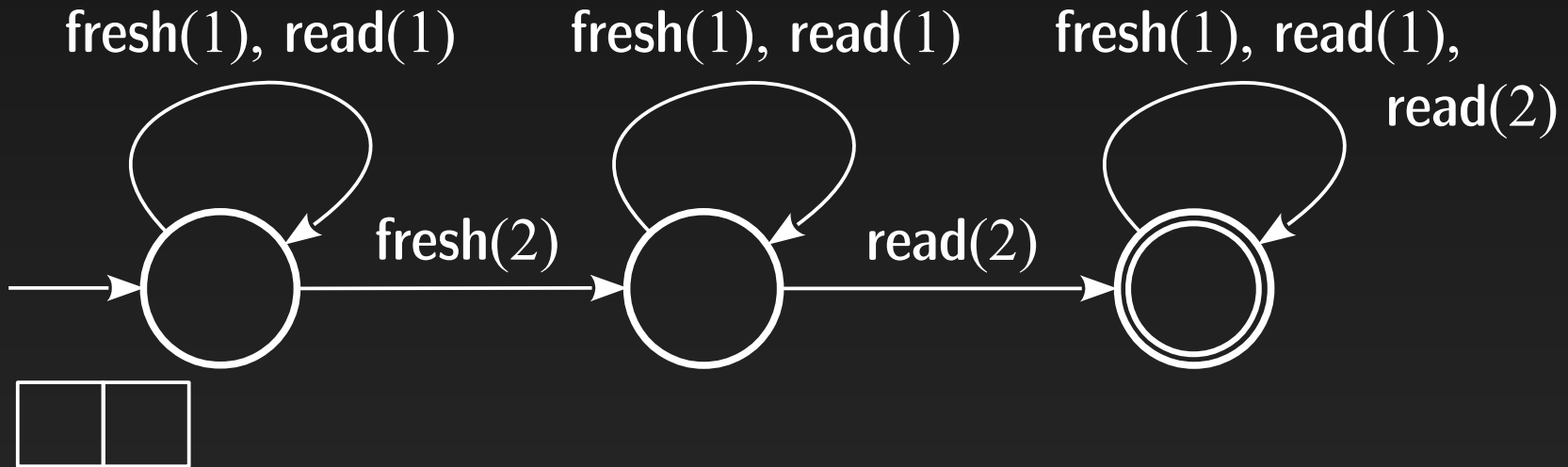


*abcadebagcab and we love cake*

# Quiz



# Quiz



$$L_2 = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \exists i \neq j. a_i = a_j \}$$

*(all strings where some name appears twice)*

$$L_{\text{fr}} = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j. a_i \neq a_j \}$$

*(all strings of pairwise distinct names)*

– what about the complement of  $L_{\text{fr}}$ ? And that of  $L_{\text{fr}} \cdot L_{\text{fr}}$ ?

# RA properties

- Capture regularity when  $\Sigma$  restricted to finite
  - Closed under  $\cup, \cap, \cdot, *$ .
  - not closed under complement & not determinisable

[Kaminski & Francez '94]

- Universality / equivalence undecidable

[Neven, Schwentick & Vianu '01]

- Decidable emptiness:
  - complexity depends on **register “mode”** (NL  $\rightarrow$  NP  $\rightarrow$  PSPACE)

[Sakamoto & Ikeda '00; Demri & Lazić '09]

- Can only truly distinguish between  $R+1$  names



# Example revisited

```
public void foo() {  
    // Create new list  
    List x = new ArrayList();  
  
    x.add(1); x.add(2);  
    Iterator i = x.iterator();  
    Iterator j = x.iterator();  
    i.next(); i.remove(); j.next();  
}
```

here is a safety property  $\varphi$ :

*if an iterator modifies its collection  $x$   
then other iterators of  $x$  become invalid*

e.g. the code on the left is bad.

We can express such “chaining”  
properties using RAs

- and dynamically verify them

[Grigore, Distefano, Petersen & T. '13]

# Example revisited

```
public void foo() {  
    // Create new list  
    List x = new ArrayList();  
  
    x.add(1); x.add(2);  
    Iterator i = x.iterator();  
    Iterator j = x.iterator();  
    i.next(); i.remove(); j.next();  
}
```

here is a safety property  $\varphi$ :

*if an iterator modifies its collection  $x$   
then other iterators of  $x$  become invalid*

e.g. the code on the left is bad.

We can express such “chaining”  
properties using RAs

- and dynamically verify them

[Grigore, Distefano, Petersen & T. '13]

However, RAs cannot capture **new**, so no good for  
modelling programs as above (e.g. for static analysis)

# Fresh-Register Automata (FRA)

Let  $\Sigma = \{a_1, a_2, \dots, a_n, \dots\}$  be an **infinite** alphabet of **names**



*finitely many*  
(say  $R$ ) **registers**

*registers store names*

Label  $\lambda$  of the form:

- **read**( $i$ ),  $i \in \{1, \dots, R\}$
- **fresh**( $i$ ),  $i \in \{1, \dots, R\}$
- **gl-fresh**( $i$ ),  $i \in \{1, \dots, R\}$

*global freshness oracle*

Transitions:



$a$	$g$	$b$
-----	-----	-----

Transitions:



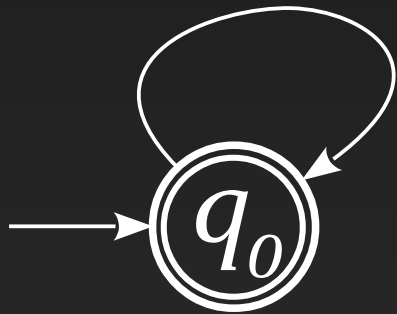
*globally fresh*

# Examples

$$L_{\text{fr}} = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j. a_i \neq a_j \}$$

*(all strings of pairwise distinct names)*

gl-fresh(1)



$$L_3 = \{ a_1 a_2 \dots a_{2n} \in \Sigma^* \mid n \geq 0, \forall i < 2n. a_i \neq a_{i+1} \\ \forall i \leq n, j < 2i. a_j \neq a_{2i} \}$$

# Application: Nominal AGS

The modelling power of FRAs can be used to model resourceful programs via **game semantics**

Program  $\rightarrow$  game model  $\rightarrow$  FRA

effectively:

two programs  
are equivalent



their FRAs are language  
equivalent / bisimilar

# Application: Nominal AGS

The modelling power of FRAs can be used to model resourceful programs via **game semantics**

Program  $\rightarrow$  game model  $\rightarrow$  FRA

effectively:

two programs  
are equivalent



their FRAs are language  
equivalent / bisimilar

*Nominal Algorithmic Game Semantics:*

- decision procedures for ML fragments
- same for Interface Middleweight Java

[Murawski & T. '11, '12]

[Murawski, Ramsay & T. '15]



# FRA properties

- Not closed under complement & not determinisable
  - Closed under  $\cup, \cap$ , but not under  $\cdot, *$
- Universality / equivalence undecidable (from RAs)
- Decidable emptiness (same as RAs):
  - complexity depends on **register "mode"** (NL  $\rightarrow$  NP  $\rightarrow$  PSPACE)
- **Bisimilarity**: decidable [T.11], complexity open

# Investigations in FRAs

## Freshness: from one to many histories

- History RA (~ Petri nets with transfer arcs)

[Grigore & T. '13]

## Bisimilarity for FRA (complexity)

- Depends on register mode (NP  $\rightarrow$  PSPACE  $\rightarrow$  EXPTIME)
  - approach uses permutation group theory

[Murawski, Ramsay & T. '15]

## Context-freeness: Pushdown FRA

[Cheng & Kaminski '98; Segoufin '06]

- Reachability EXPTIME-complete
- Global reachability via “saturation”

[Murawski & T. '12]

[Murawski, Ramsay & T. '14]

# Semantics formally: configurations

Semantics of FRAs given by **configuration graphs**:

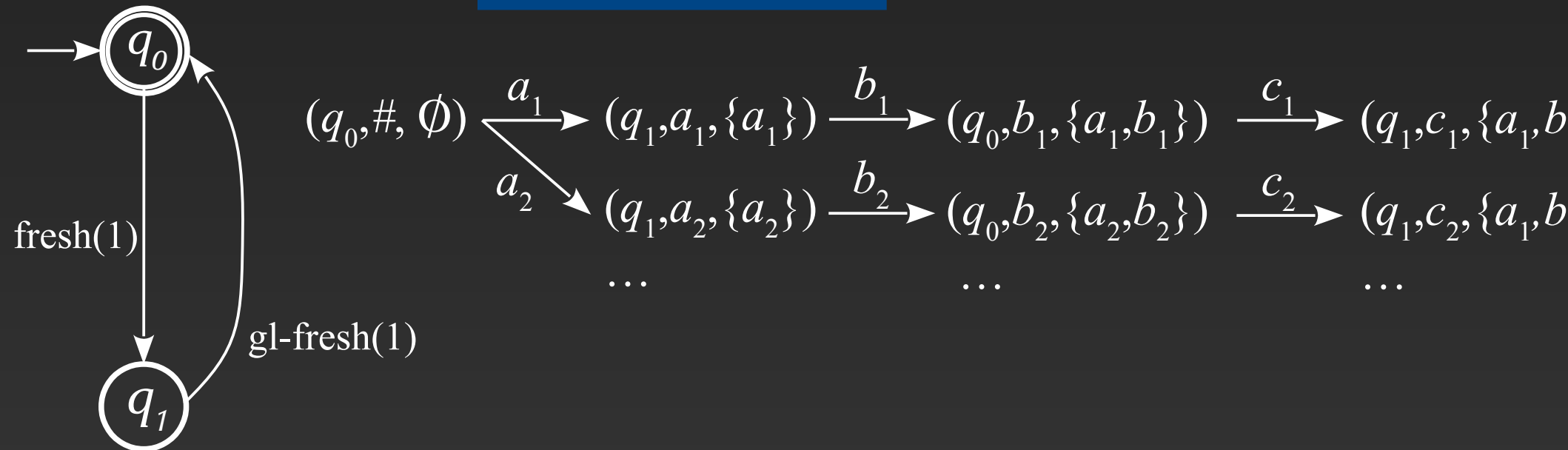
configuration

$$(q, \rho, H) \xrightarrow{a} (q', \rho', H')$$

state

register assignment:  
 $\rho : \{1, \dots, R\} \rightarrow \Sigma \cup \{\#\}$

history:  $H \subseteq_{\text{fin}} \Sigma$



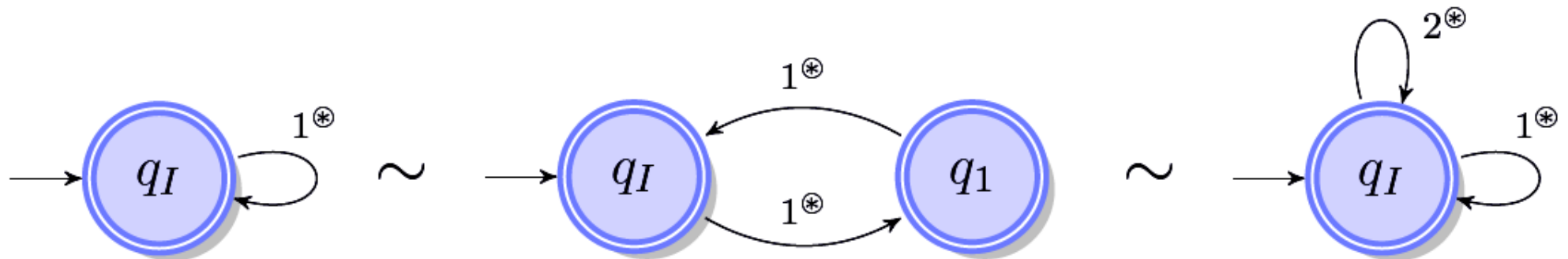
# Bisimilarity

A **behavioural** notion of equivalence:

two configurations  $\kappa_1, \kappa_2$  are bisimilar ( $\kappa_1 \sim \kappa_2$ )  
if they can simulate one another name-by-name

We say that two FRAs are bisimilar if their initial configurations are (in the combined conf. graph).

e.g. (writing  $1^*$  for  $\text{gl-fresh}(1)$ ):



# A small detail: register modes

So far we assumed: registers **initially empty**, not possible to **erase** them, or have name **duplicates**.

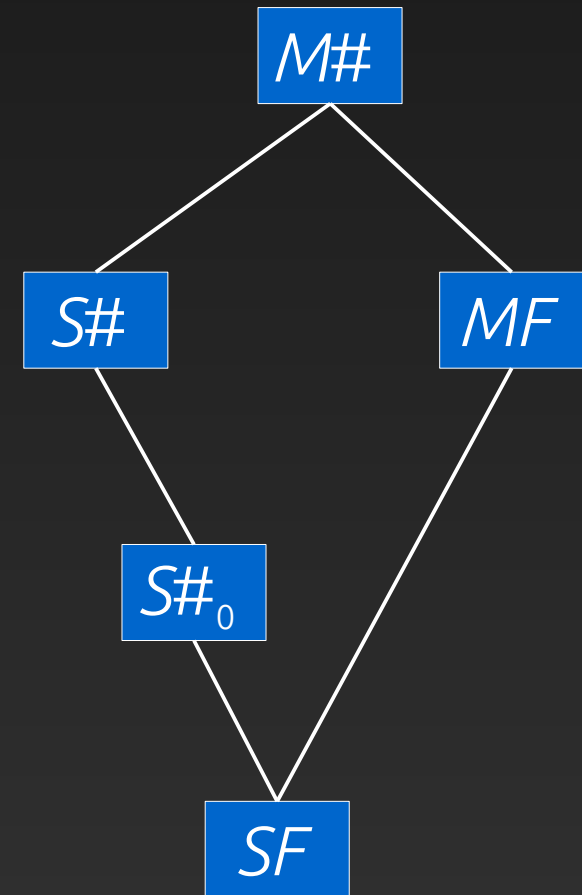
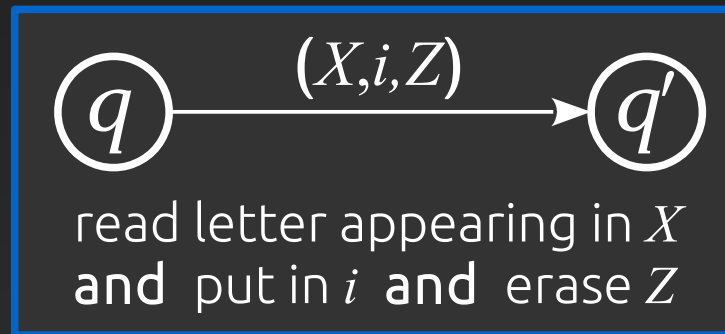
We now generalise:

*Name multiplicity:*

- ( $S$ ) single
- ( $M$ ) multiple

*Register fullness:*

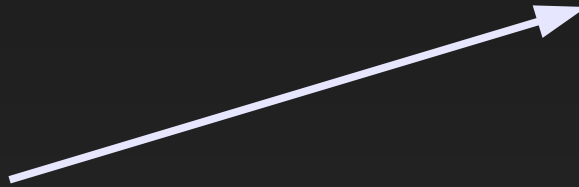
- ( $F$ ) full
- ( $\#_0$ ) initially empty
- ( $\#$ ) eraseable



remember: # means  
"empty register content"

Example:  $(S\#) \rightarrow (MF)$

<i>a</i>	<i>g</i>	#	<i>b</i>	#
----------	----------	---	----------	---



<i>z</i>	<i>a</i>	<i>g</i>	<i>z</i>	<i>b</i>	<i>z</i>
----------	----------	----------	----------	----------	----------

neat, but erasing  
gives exponentially  
large labels

# Example: $(S\#) \rightarrow (MF)$

$a \mid g \mid \# \mid b \mid \#$

$z \mid a \mid g \mid z \mid b \mid z$

neat, but erasing  
gives exponentially  
large labels

$a' \mid g' \mid c \mid b' \mid d \mid a \mid g \mid c \mid b \mid d$

concise, as each  
name appears  
at most twice

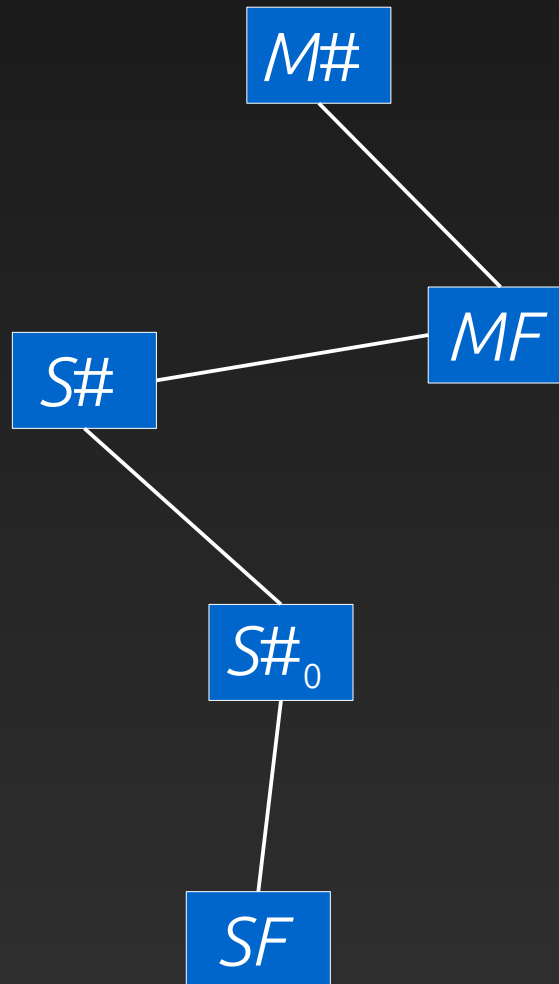
# Complexity Picture

## *Multiplicity:*

- (*S*) single
- (*M*) multiple

## *Fullness:*

- (*F*) full
- ( $\#_0$ ) initially empty
- ( $\#$ ) eraseable





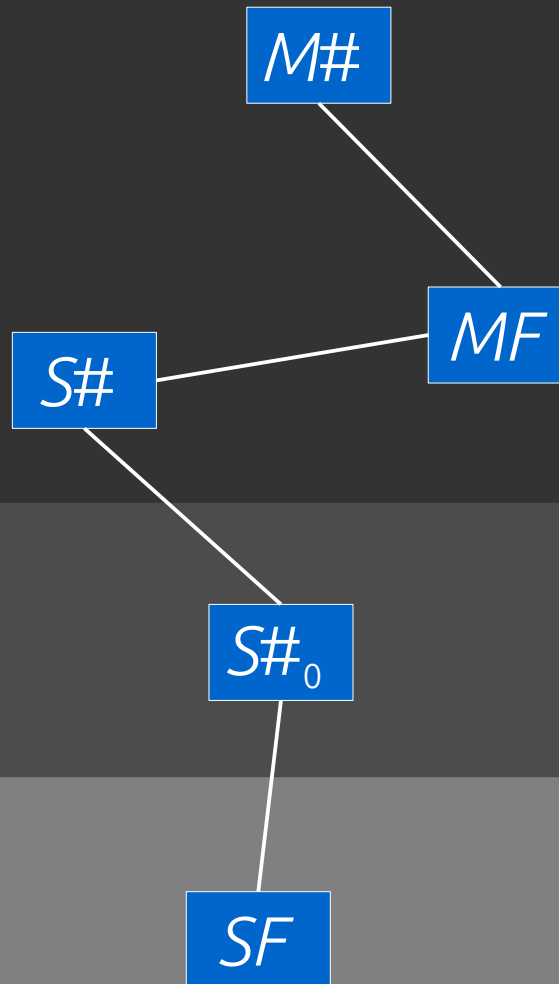
# Complexity Picture

## Multiplicity:

- (*S*) single
- (*M*) multiple

## Fullness:

- (*F*) full
- ( $\#_0$ ) initially empty
- ( $\#$ ) eraseable



EXPTIME-c

PSPACE-c

NP

# EXPTIME solvability

To decide bisimilarity of two configurations of size  $R$ :

- we need  $2R$  names to represent all possible name matchings between them
- plus one name that stands for “locally fresh”
- and another one for “globally fresh”

# EXPTIME solvability

To decide bisimilarity of two configurations of size  $R$ :

- we need  $2R$  names to represent all possible name matchings between them
- plus one name that stands for “locally fresh”
- and another one for “globally fresh”

→  $2R+2$  names, that we can encode inside states:

$$Q \longrightarrow Q \times (2R+2)^R$$

and bisimilarity for finite-state automata is in PTIME

# Complexity Picture

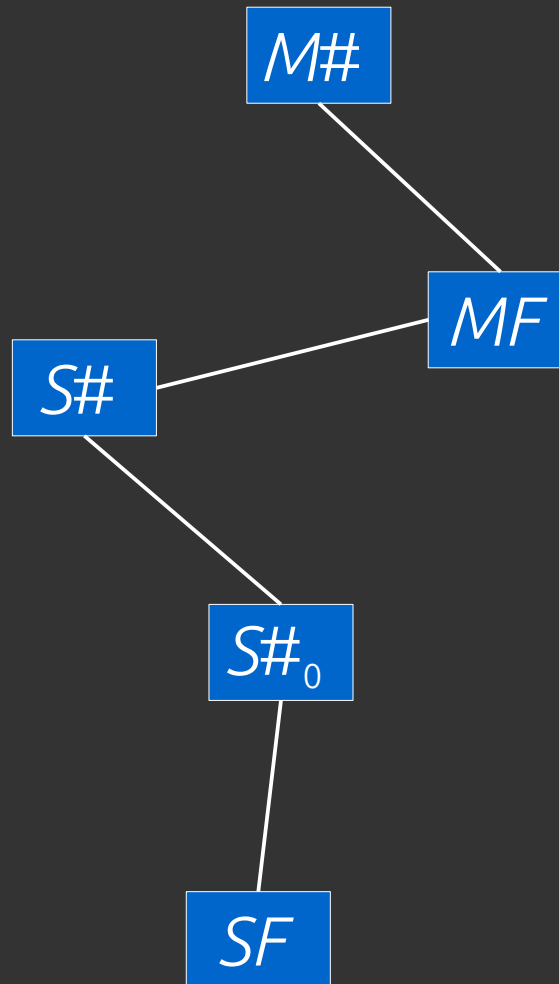
## Multiplicity:

- (*S*) single
- (*M*) multiple

## Fullness:

- (*F*) full
- ( $\#_0$ ) initially empty
- ( $\#$ ) eraseable

EXPTIME



# Bisimulation defined as a game

Two players, an **Attacker (A)** and a **Defender (D)**, play a game on a configuration graph, starting from configurations  $\kappa_1$  and  $\kappa_2$

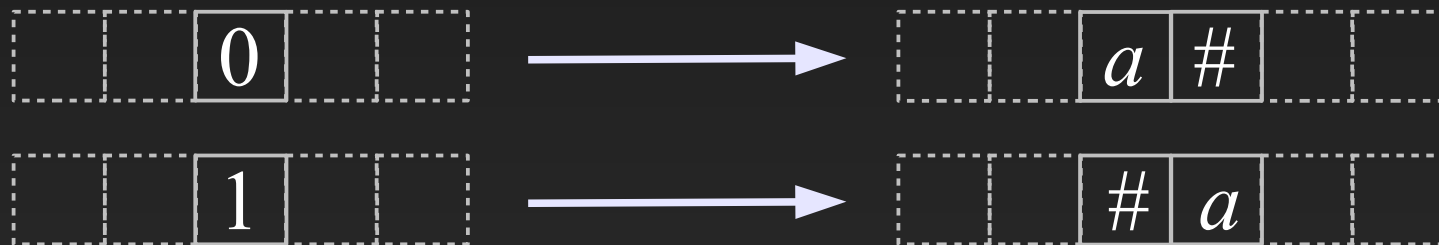
- **A** wants to show them **not equivalent**, **D** the opposite
- **A** picks a configuration, say  $\kappa_1$ , and an edge out of it, say with label  $a$
- **D** must then pick an  $a$ -edge out of  $\kappa_2$  – or he loses!
- now we are at  $\kappa_1'$  and  $\kappa_2'$  – and the game continues

$\kappa_1$  and  $\kappa_2$  are called **bisimilar**, write  $\kappa_1 \sim \kappa_2$ , if **D** has a strategy for **not losing** the game

# EXPTIME hardness

The ( $S\#$ ) case is EXPTIME-hard:

- reduce from alternating TMs with linear-size tape (ALBA)
- model each cell by two registers:



- arrange for non-bisimilarity at rejecting final states
- use Defender forcing [Jancar & Srba '08] for existential states

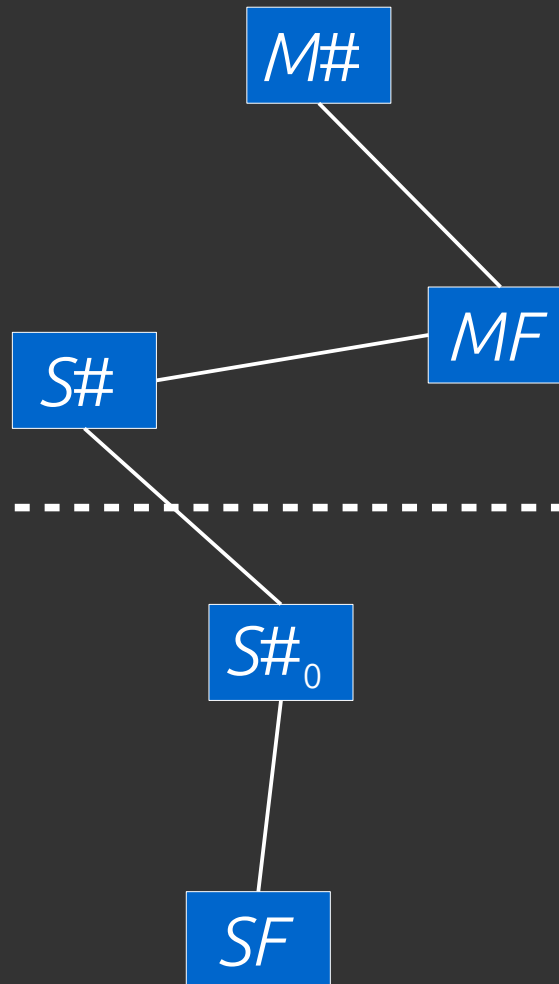
# Complexity Picture

## Multiplicity:

- (*S*) single
- (*M*) multiple

## Fullness:

- (*F*) full
- ( $\#_0$ ) initially empty
- ( $\#$ ) eraseable



EXPTIME

EXPTIME

# The original case ( $S\#_0$ )

Disallowing erasures makes impossible our modelling of a linear-size tape...

*In fact, the problem is PSPACE complete*

First, we can model boolean assignments (cf. write-once tape), which are enough for PSPACE-hardness:

- we reduce from QBF
- Attacker chooses universal variables
- Defender chooses existential ones (via forcing)



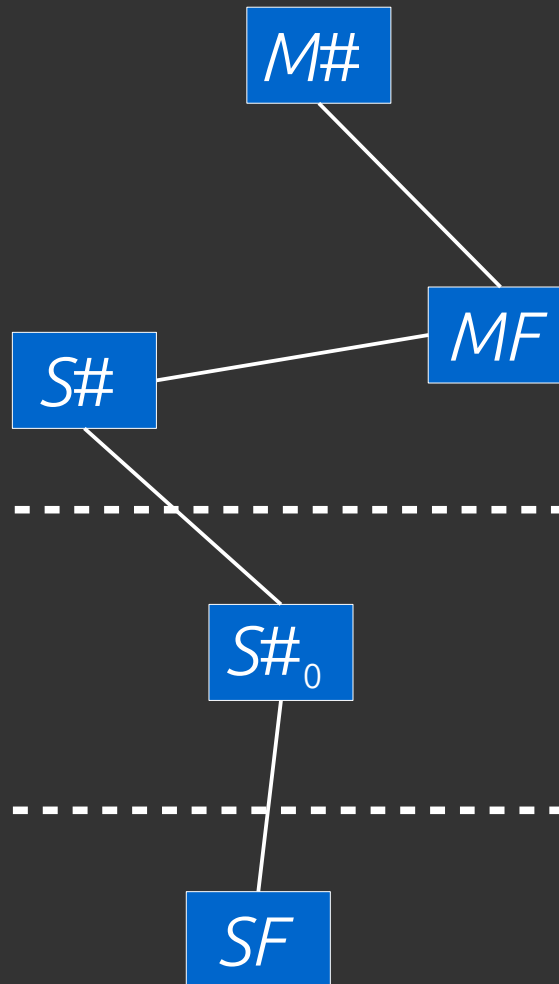
# Complexity Picture

## Multiplicity:

- (*S*) single
- (*M*) multiple

## Fullness:

- (*F*) full
- ( $\#_0$ ) initially empty
- ( $\#$ ) eraseable



EXPTIME

EXPTIME

PSPACE

# PSPACE solvability: difficult

Our best bet is  $\text{APTIME} = \text{PSPACE}$

- problem: while we cannot simulate a linear tape, we still have a lot of configurations!
  - even for RAs: exponentially many

# PSPACE solvability: difficult

Our best bet is  $\text{APTIME} = \text{PSPACE}$

- problem: while we cannot simulate a linear tape, we still have a lot of configurations!
  - even for RAs: exponentially many

We look into internal symmetries of FRAs:

- **symbolic reasoning**: we are only interested in how configurations are related, not their actual content
- **group representations**: we express these interrelations compactly via permutation groups
- **bounded history**: it suffices to consider histories of size up to  $2R$

# PSPACE solvability

$$\sim^0 \supseteq \sim^1 \supseteq \sim^2 \supseteq \dots \supseteq \sim^i \supseteq \dots \quad \text{and} \quad \sim = \bigcup_{i \in \omega} \sim^i$$

Reasoning symbolically:

- each decrease in the indexed chain can be traced back to one of polynomially many factors!

use the fact that strict subgroup chains have bounded length

This means there is a **final polynomial-size  $i$**

- gives a polynomial bound for the bisimulation game, hence APTIME

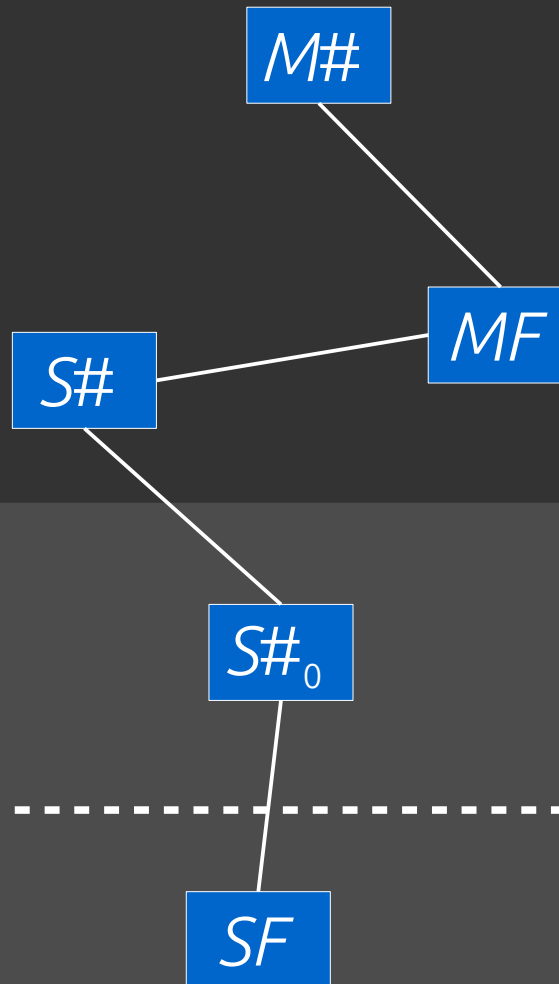
# Complexity Picture

## Multiplicity:

- (*S*) single
- (*M*) multiple

## Fullness:

- (*F*) full
- ( $\#_0$ ) initially empty
- ( $\#$ ) eraseable



# The (SF) case: NP solvability

Here, reasoning symbolically we can:

- **guess** and **verify** an actual **symbolic bisimulation**
- using compactness of group representations, achieve this in polynomial time

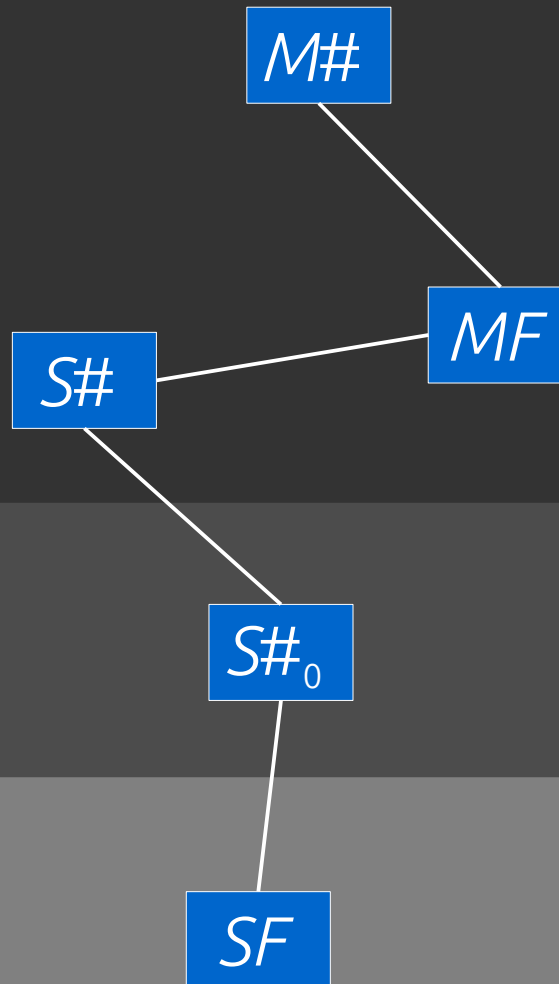
# Complexity Picture

## Multiplicity:

- (*S*) single
- (*M*) multiple

## Fullness:

- (*F*) full
- ( $\#_0$ ) initially empty
- ( $\#$ ) eraseable



EXPTIME

PSPACE

NP

# Investigations in FRAs

Freshness: from one to many histories

- History RA (~ Petri nets with transfer arcs)

[Grigore & T. '13]

Bisimilarity for FRA (complexity)

- Depends on register mode (NP  $\rightarrow$  PSPACE  $\rightarrow$  EXPTIME)
  - approach uses permutation group theory

[Murawski, Ramsay & T. '15]

Context-freeness: Pushdown FRA

[Cheng & Kaminski '98; Segoufin '06]

- Reachability EXPTIME-complete
- Global reachability via “saturation”

[Murawski & T. '12]

[Murawski, Ramsay & T. '14]

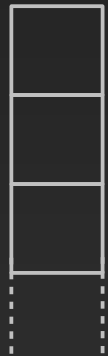


# Pushdown Register Automata (PDRA)

Let  $\Sigma = \{a_1, a_2, \dots, a_n, \dots\}$  be an **infinite** alphabet of **names**



*finitely many  
(say  $R$ ) registers*



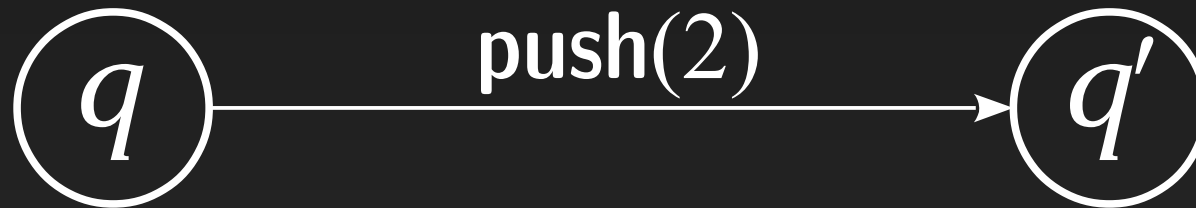
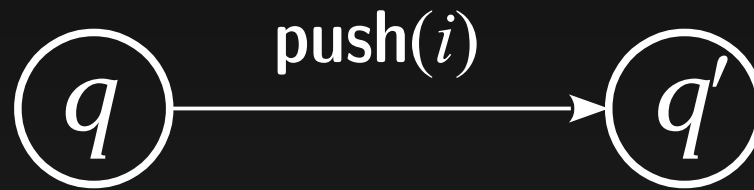
*pushdown stack*

*registers & stack store names*

Label  $\lambda$  of the form:

- **read**( $i$ ),  $i \in \{1, \dots, R\}$
- **fresh**( $i$ ),  $i \in \{1, \dots, R\}$
- **push**( $i$ ),  $i \in \{1, \dots, R\}$
- **pop**( $i$ ),  $i \in \{1, \dots, R\}$
- **pop-fresh**

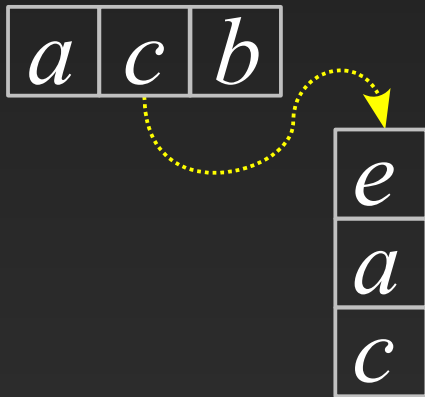
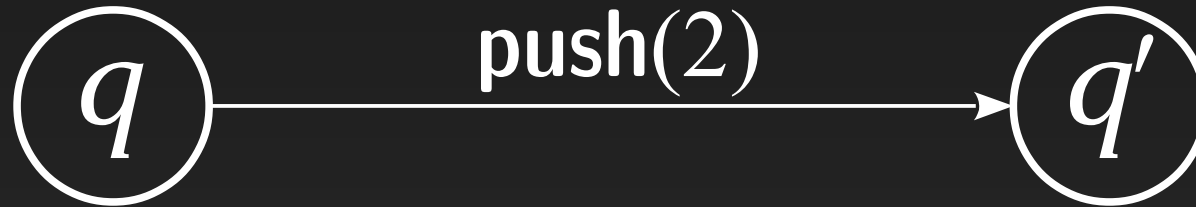
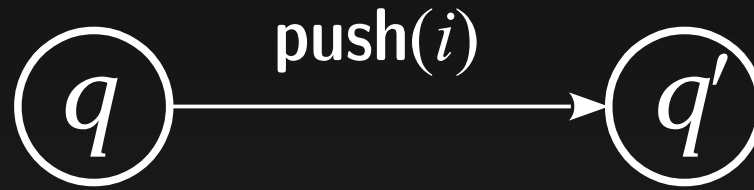
Transitions:



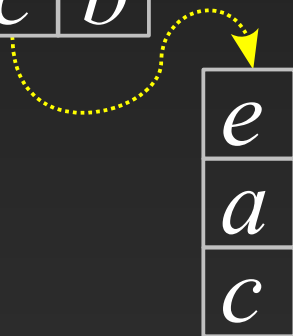
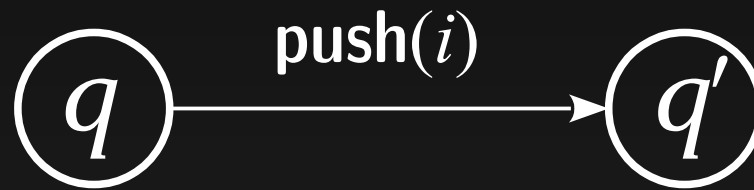
$a$	$c$	$b$
-----	-----	-----

$e$
$a$
$c$

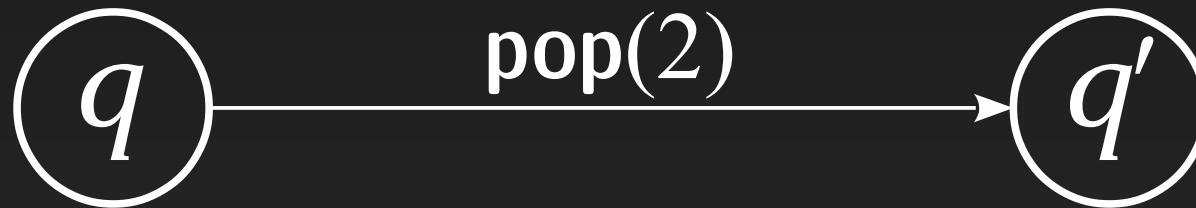
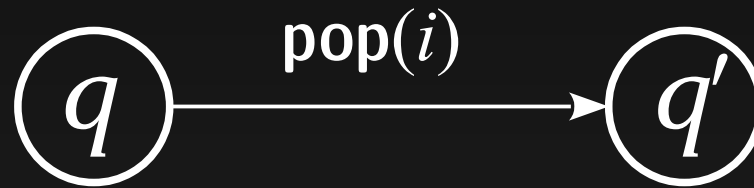
Transitions:



Transitions:



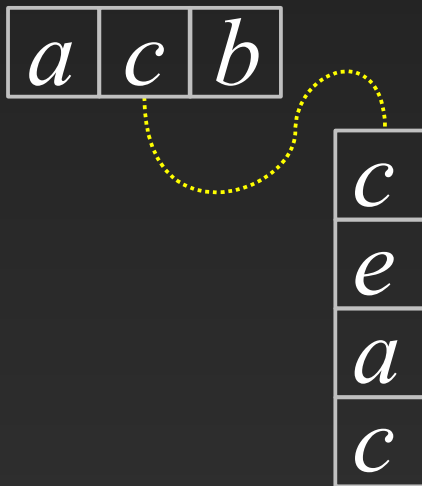
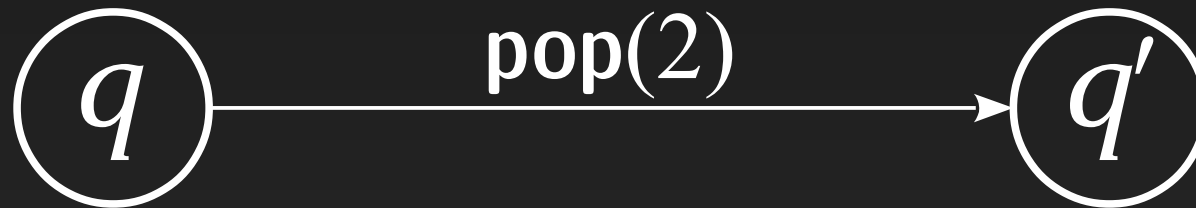
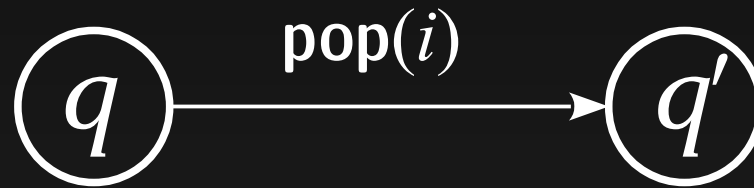
Transitions:



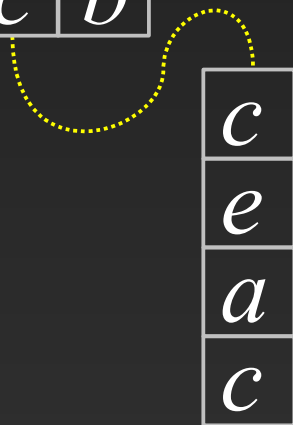
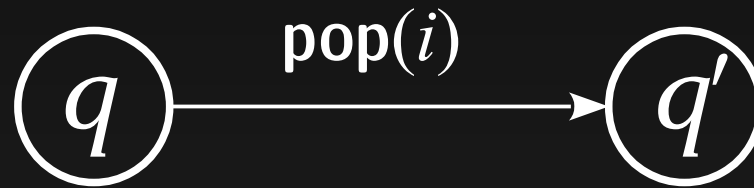
$a$	$c$	$b$
-----	-----	-----

$c$
$e$
$a$
$c$

Transitions:



Transitions:



Transitions:





Transitions:



a c b

*fresh*

e  
a  
c

Transitions:



$a$   $c$   $b$

*fresh*

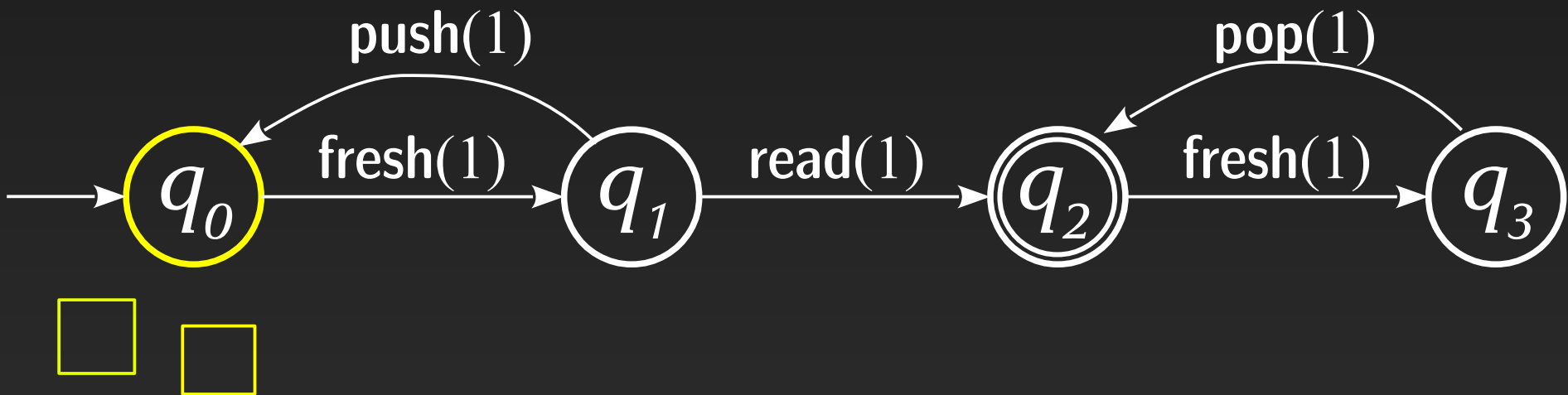
$e$   
 $a$   
 $c$

$a$   $c$   $b$

$a$   
 $c$

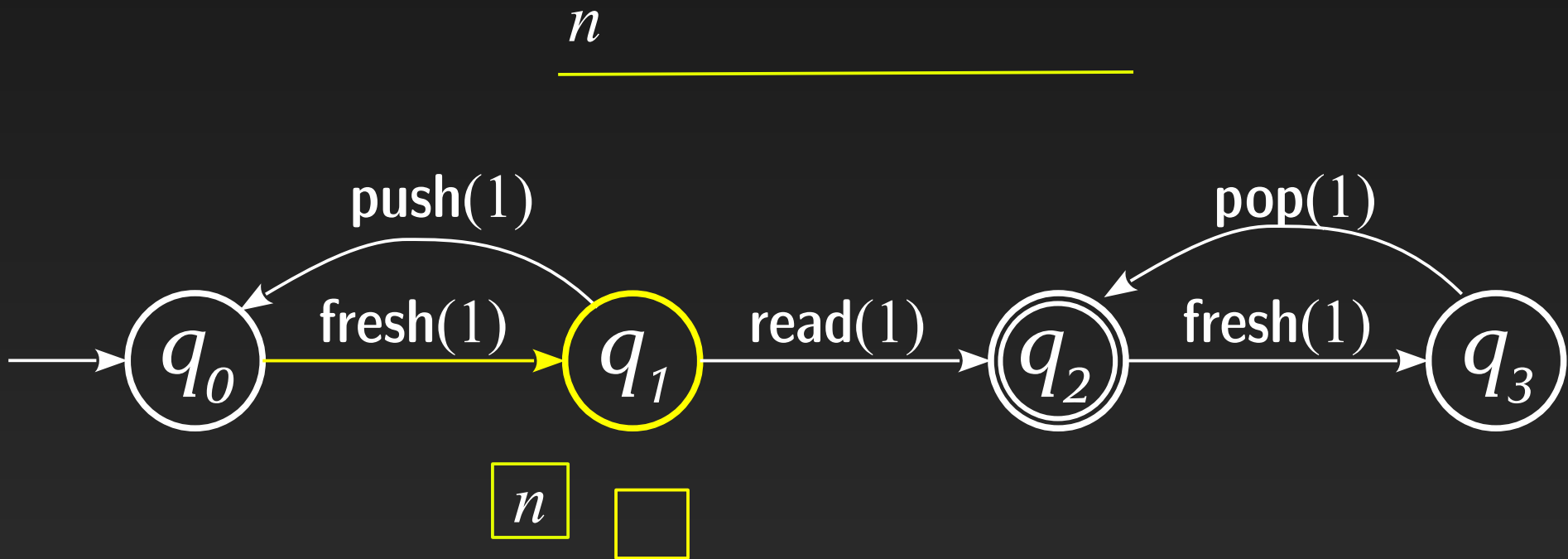
# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$



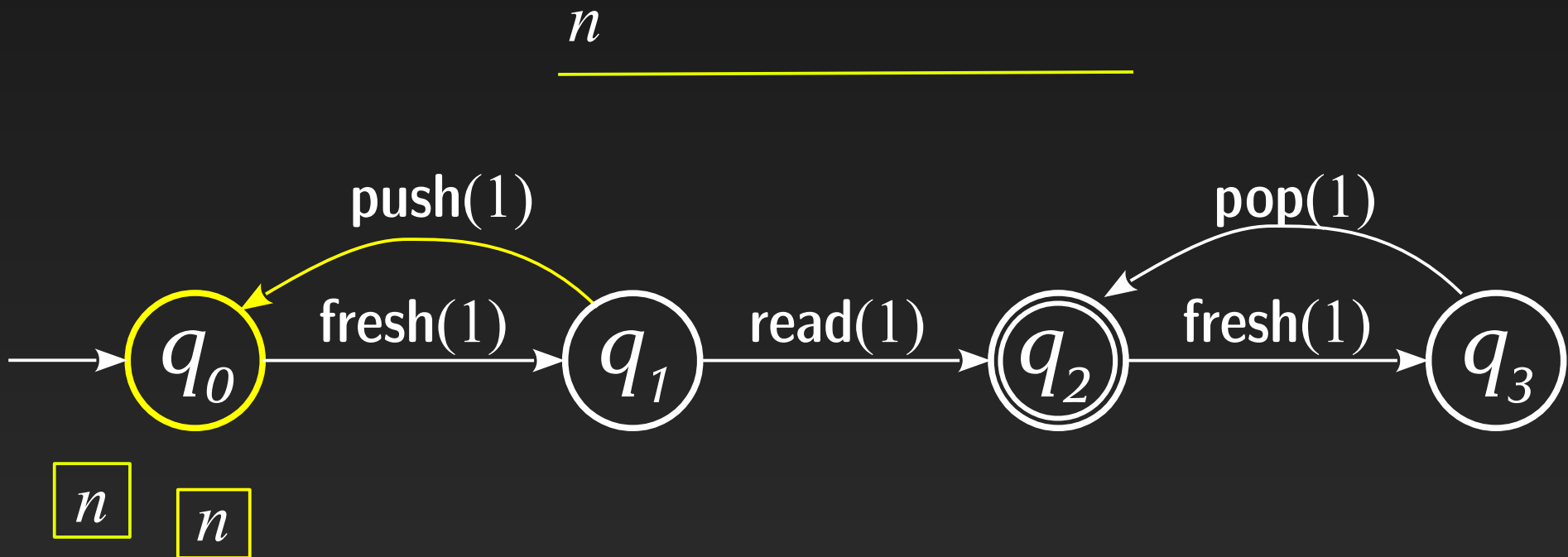
# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$



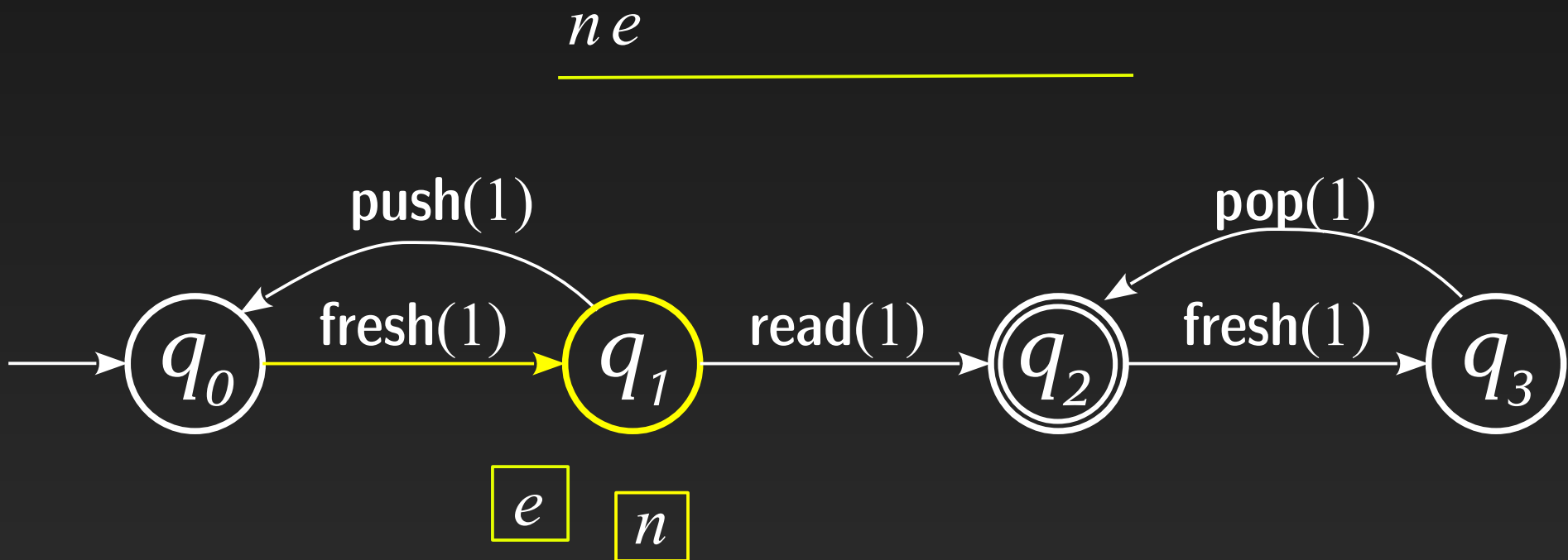
# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$



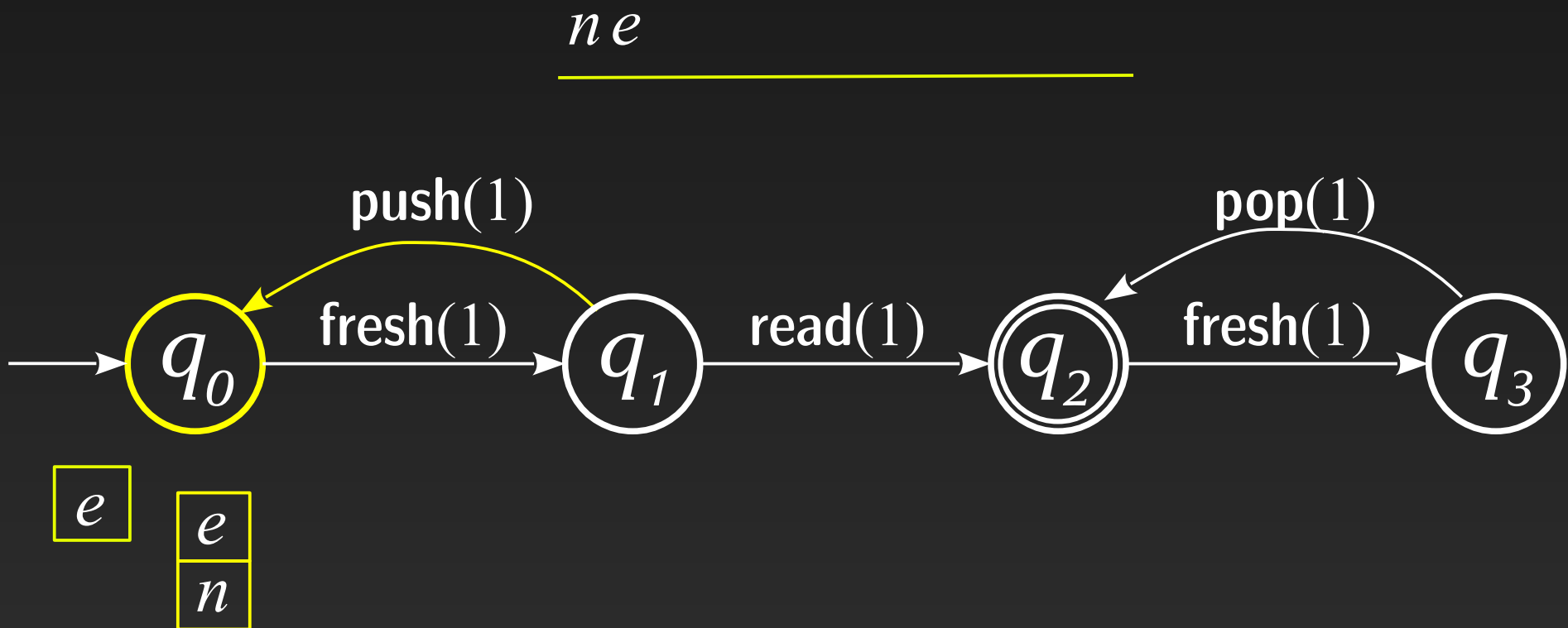
# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$



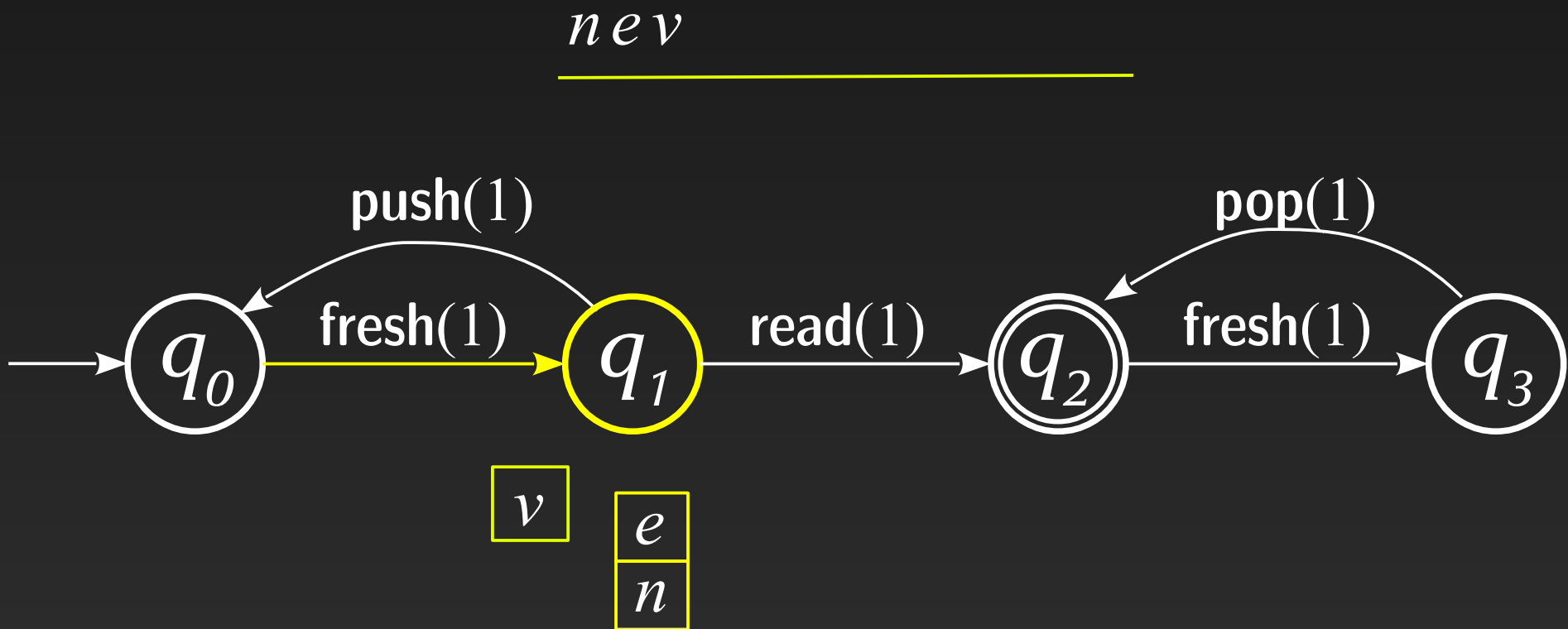
# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$



# Example

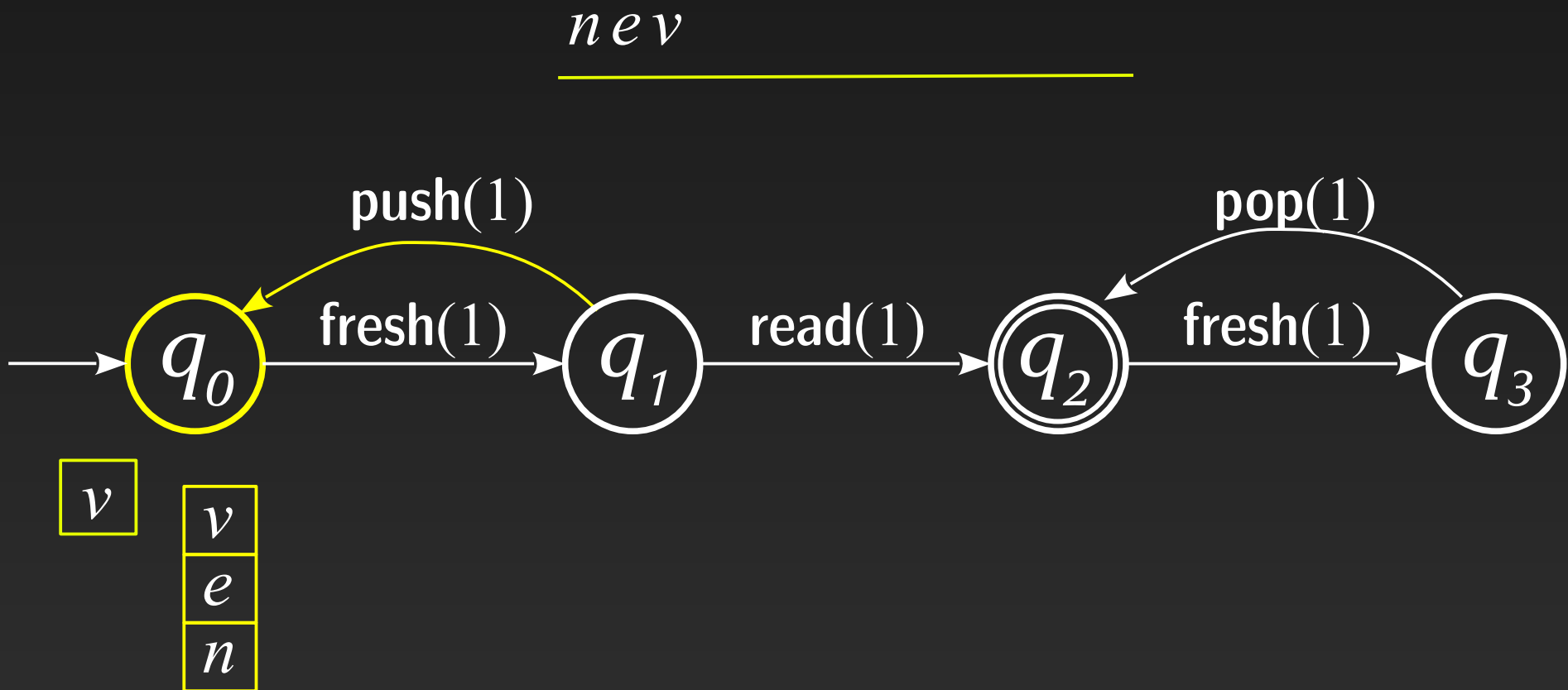
$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$





# Example

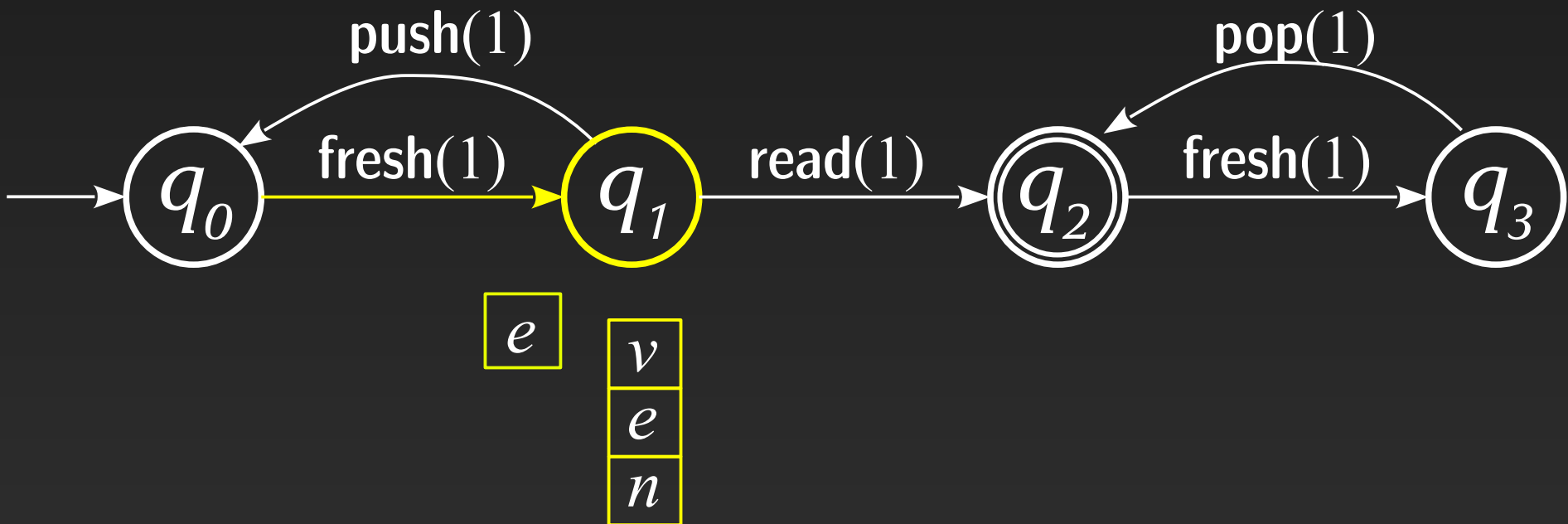
$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$



# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

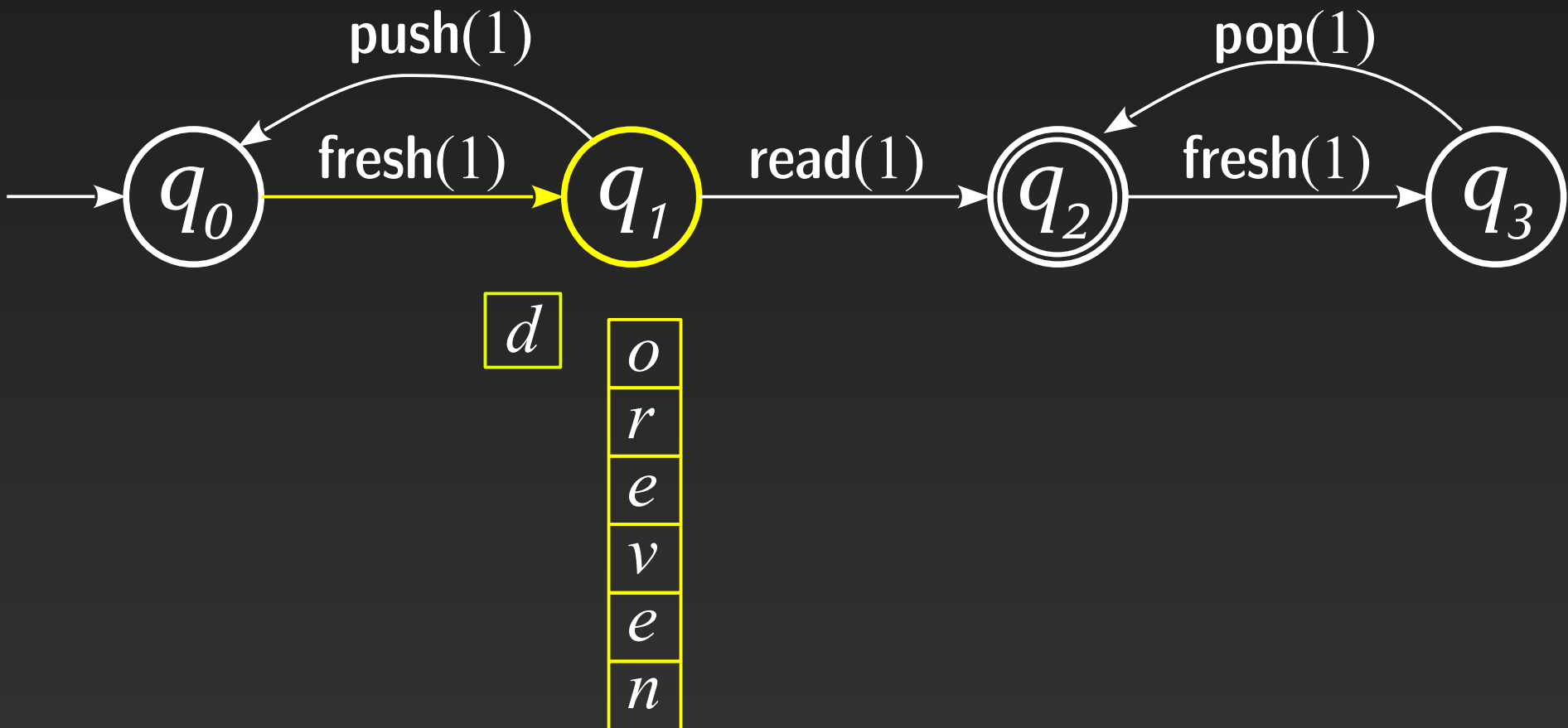
neve



# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

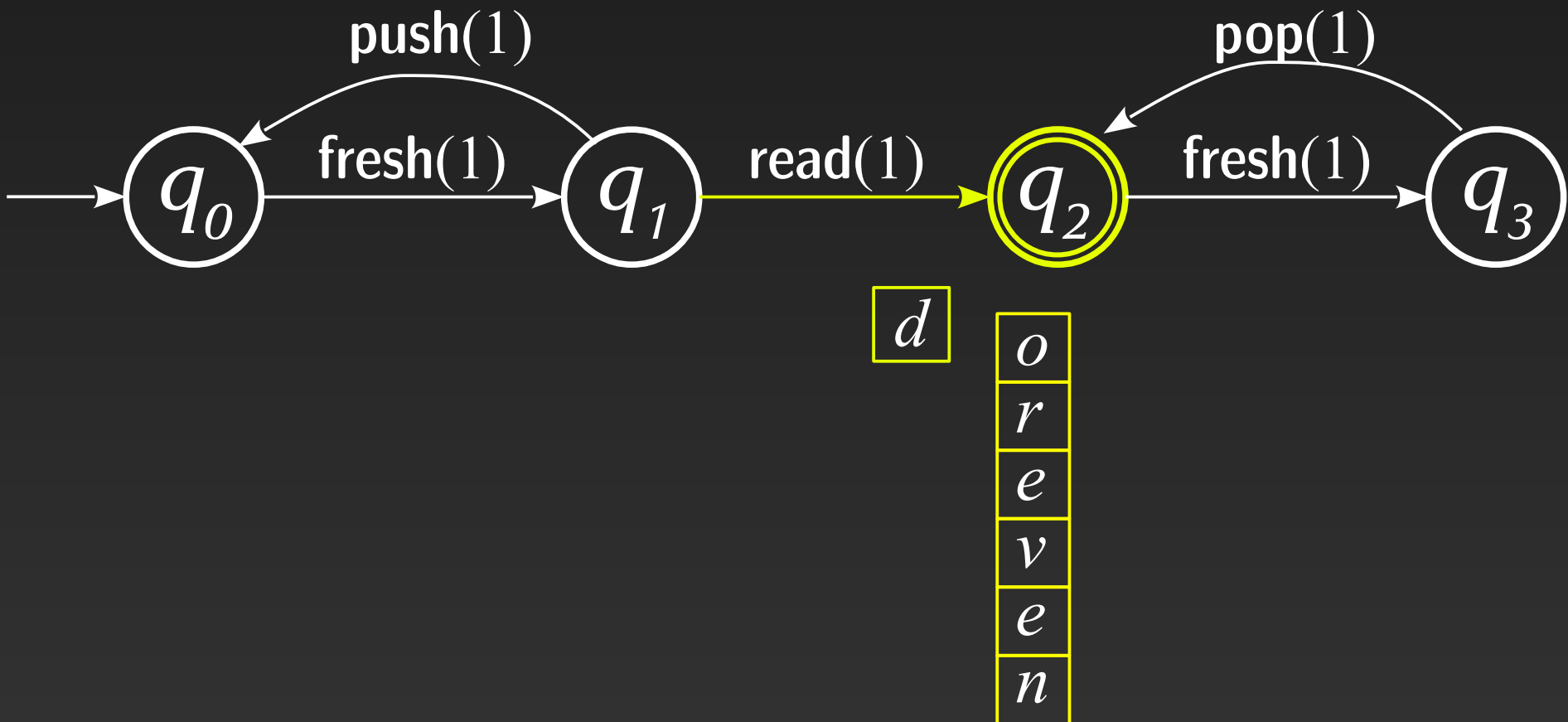
neverod



# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

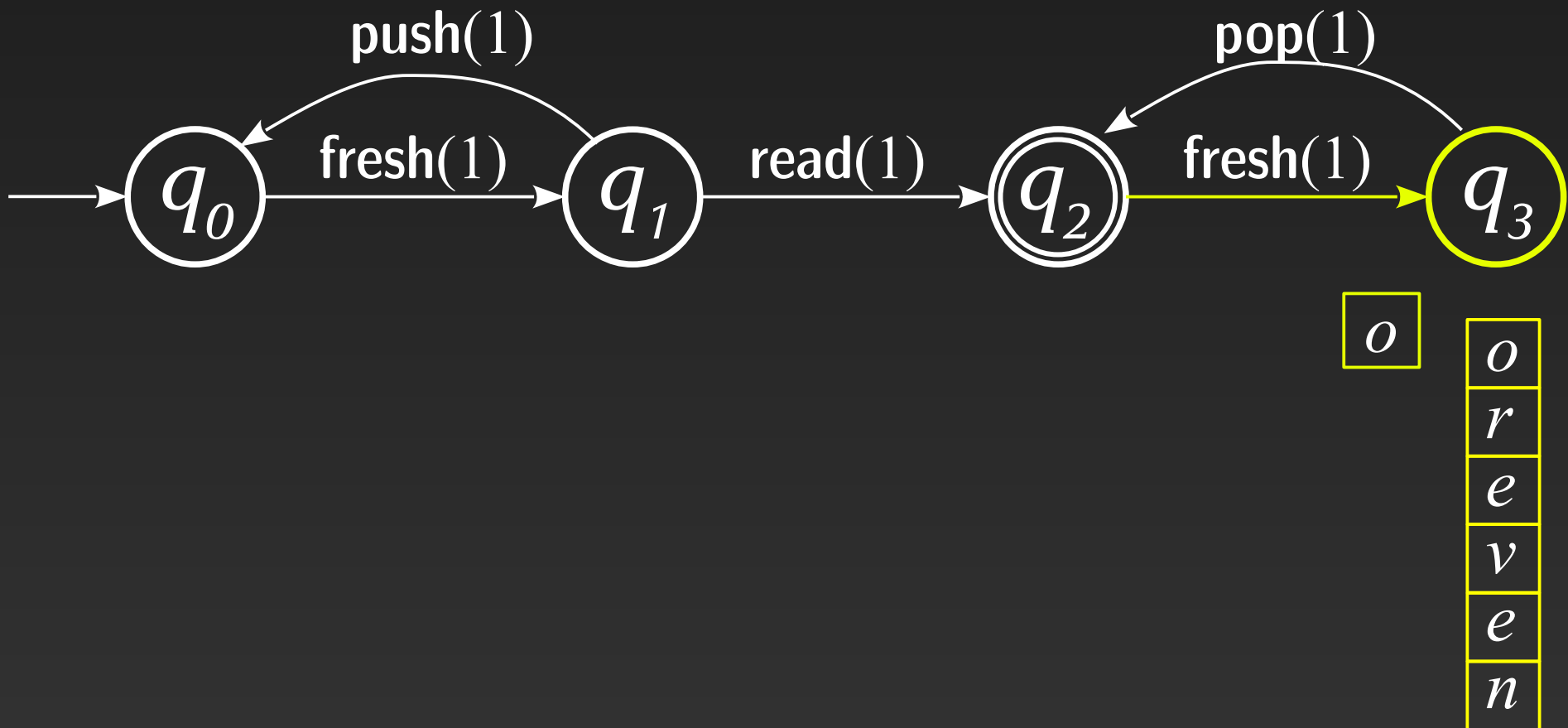
*neverodd*



# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

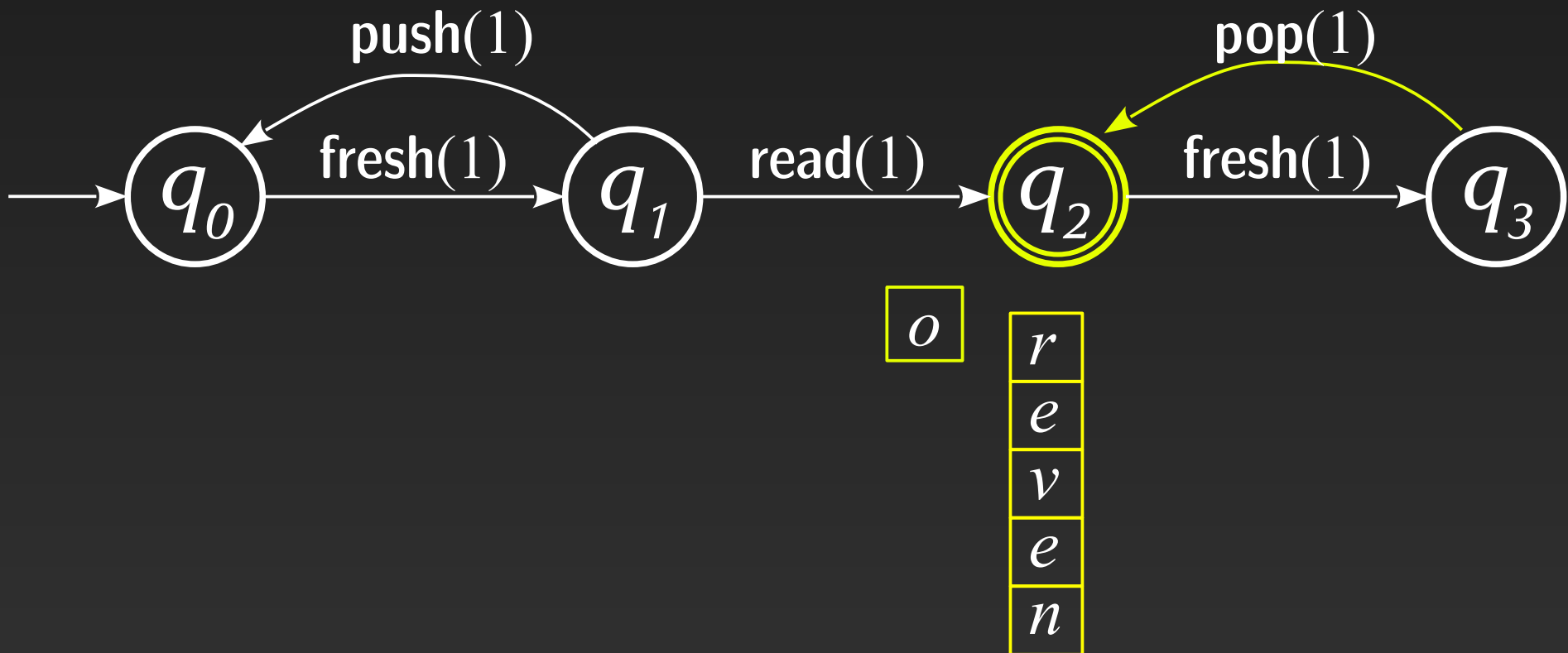
*neveroddo*



# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

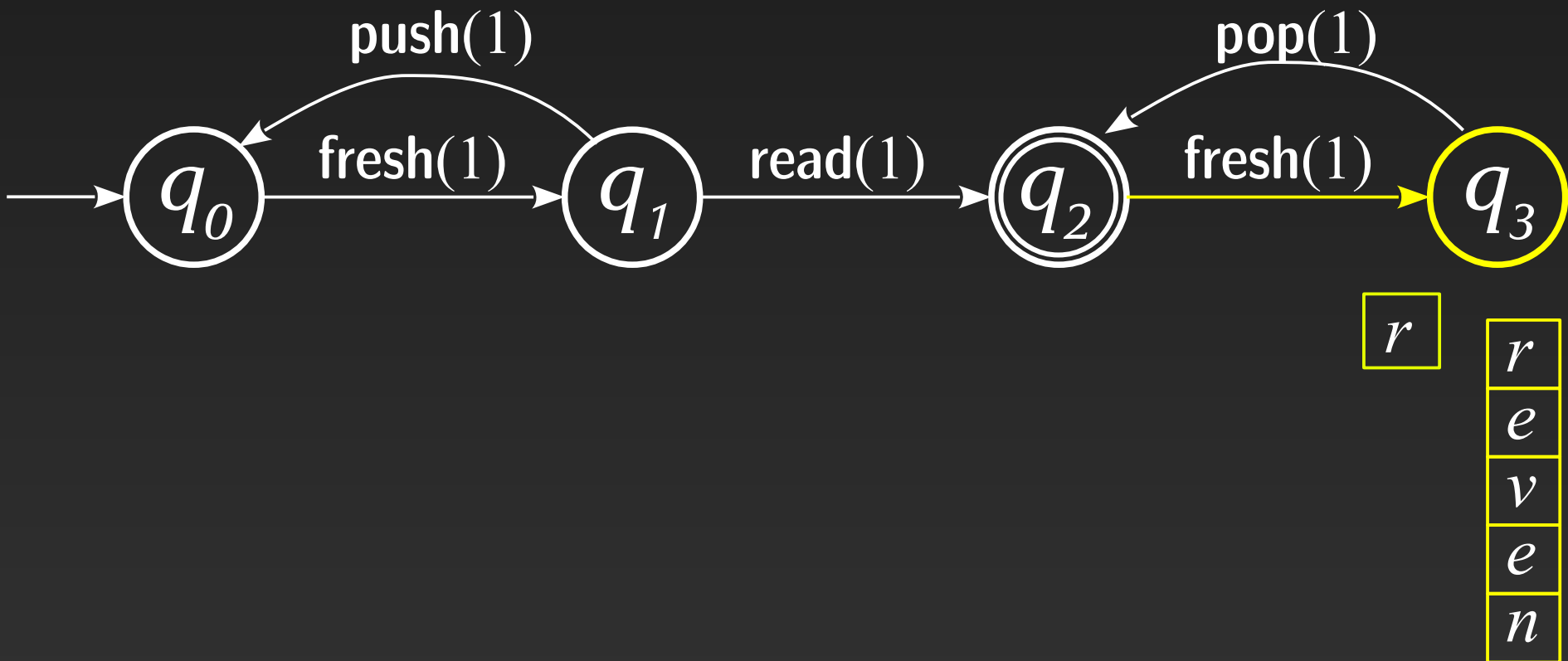
*neveroddo*



# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

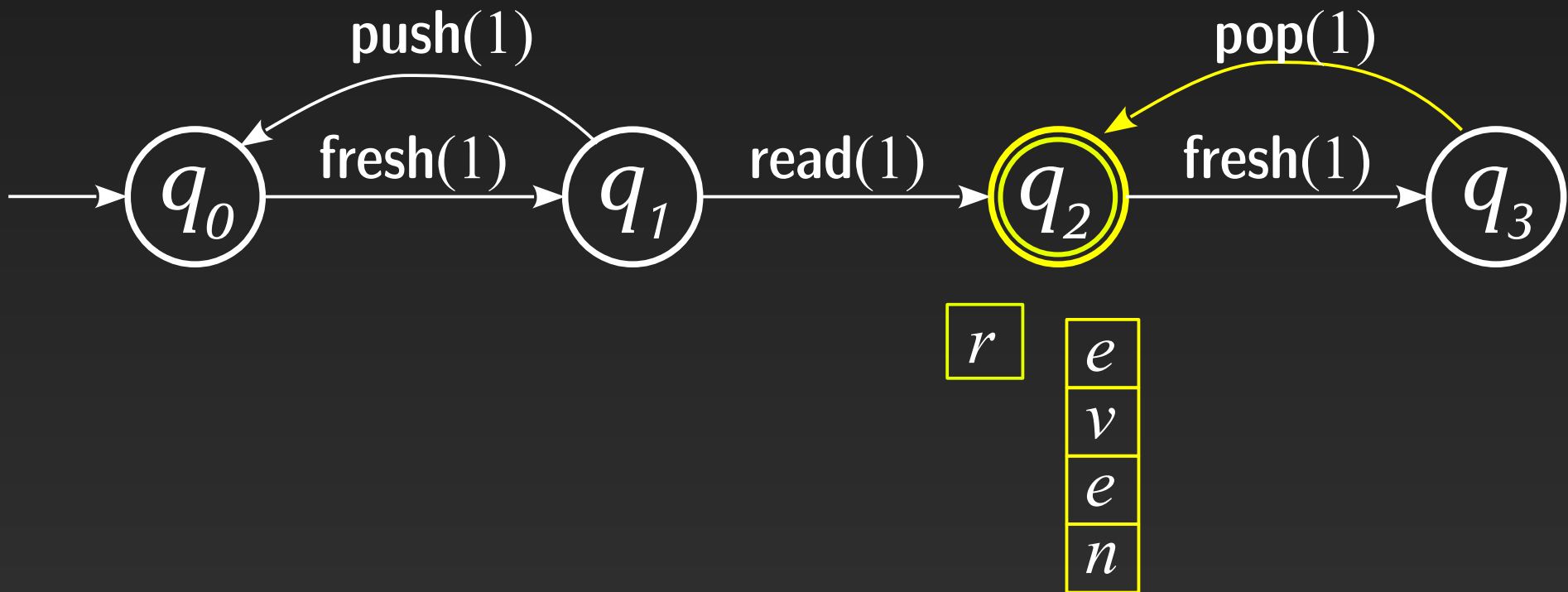
*neveroddor*



# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

*neveroddor*

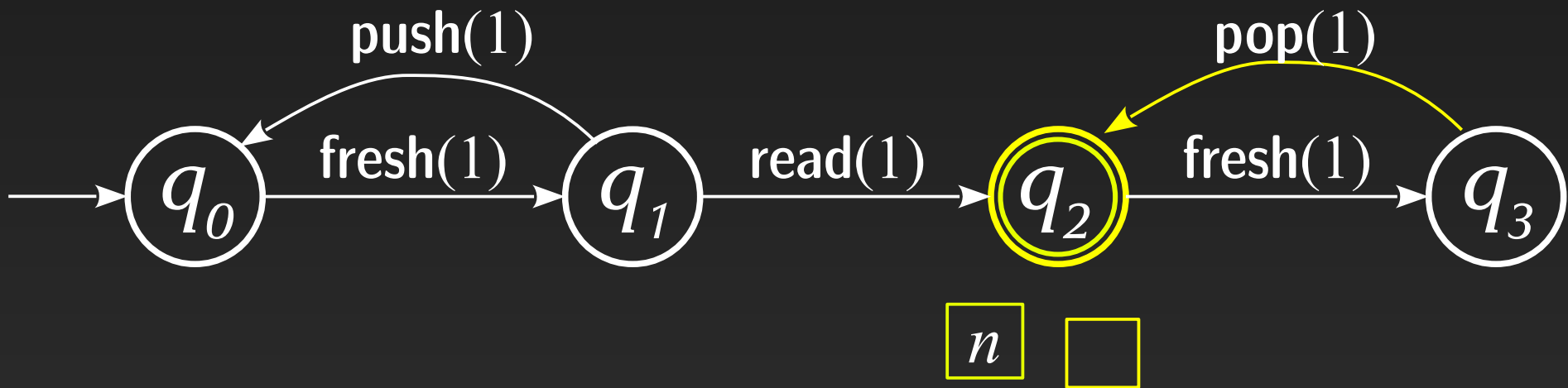




# Example

$$L_4 = \{ a_1 a_2 \dots a_n a_n \dots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$$

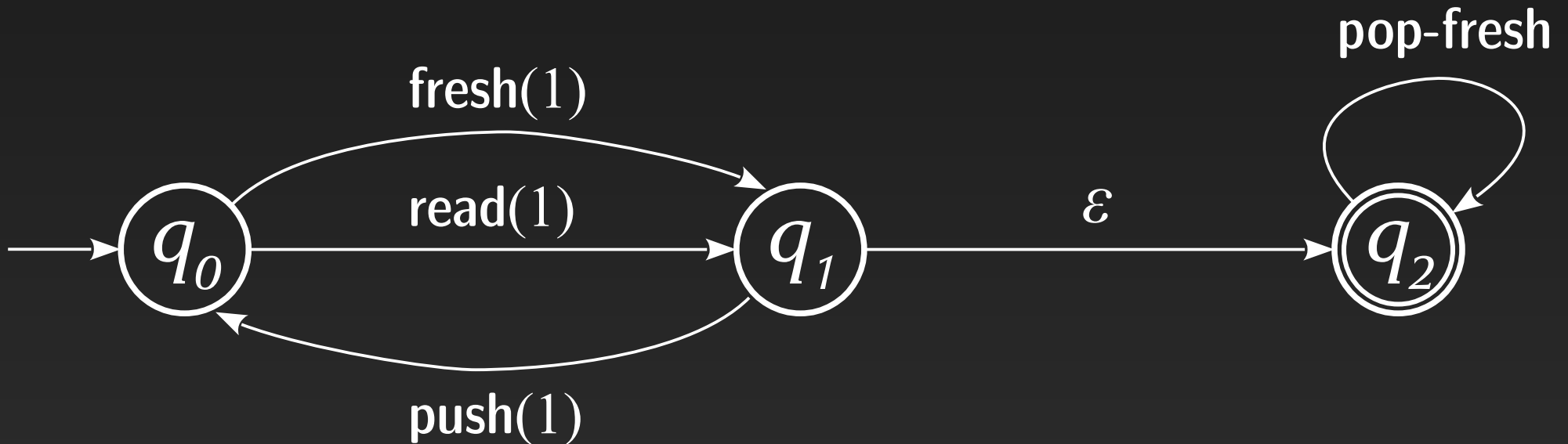
*neveroddoeven*



# Example

$$L_5 = \{ a_1 a_2 \dots a_n b \in \Sigma^* \mid n \geq 0, \forall i \leq n. a_i \neq b \}$$

*(all strings where last name is distinct from all previous ones)*



# Limited distinguish-ability

$$L_{\text{fr}} = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j. a_i \neq a_j \}$$

*(all strings of distinct names)*

# Limited distinguish-ability

$$L_{\text{fr}} = \{ a_1 a_2 \dots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j. a_i \neq a_j \}$$

*(all strings of distinct names)*

**Lemma:** Let  $A$  be a PDRA with  $R$ -many registers. For any pair of states  $q_1$  and  $q_2$ , if there is a run between them (from empty stack to empty stack) then there is one of same length involving at most  $3R$  names.

*Conversely*, there is a PDRA with  $R$  registers whose runs to a designated state involve exactly  $3R$  names.

# Reachability / non-emptiness

*R-PRDA Reach: Given a PDRA  $A$  with  $R$  registers and a state  $q$ , is there a run of  $A$  to  $q$ ?*

**Theorem:** *R-PRDA Reach is EXPTIME-complete.*

# Reachability / non-emptiness

*R-PRDA Reach: Given a PDRA  $A$  with  $R$  registers and a state  $q$ , is there a run of  $A$  to  $q$ ?*

**Theorem:** *R-PRDA Reach is EXPTIME-complete.*

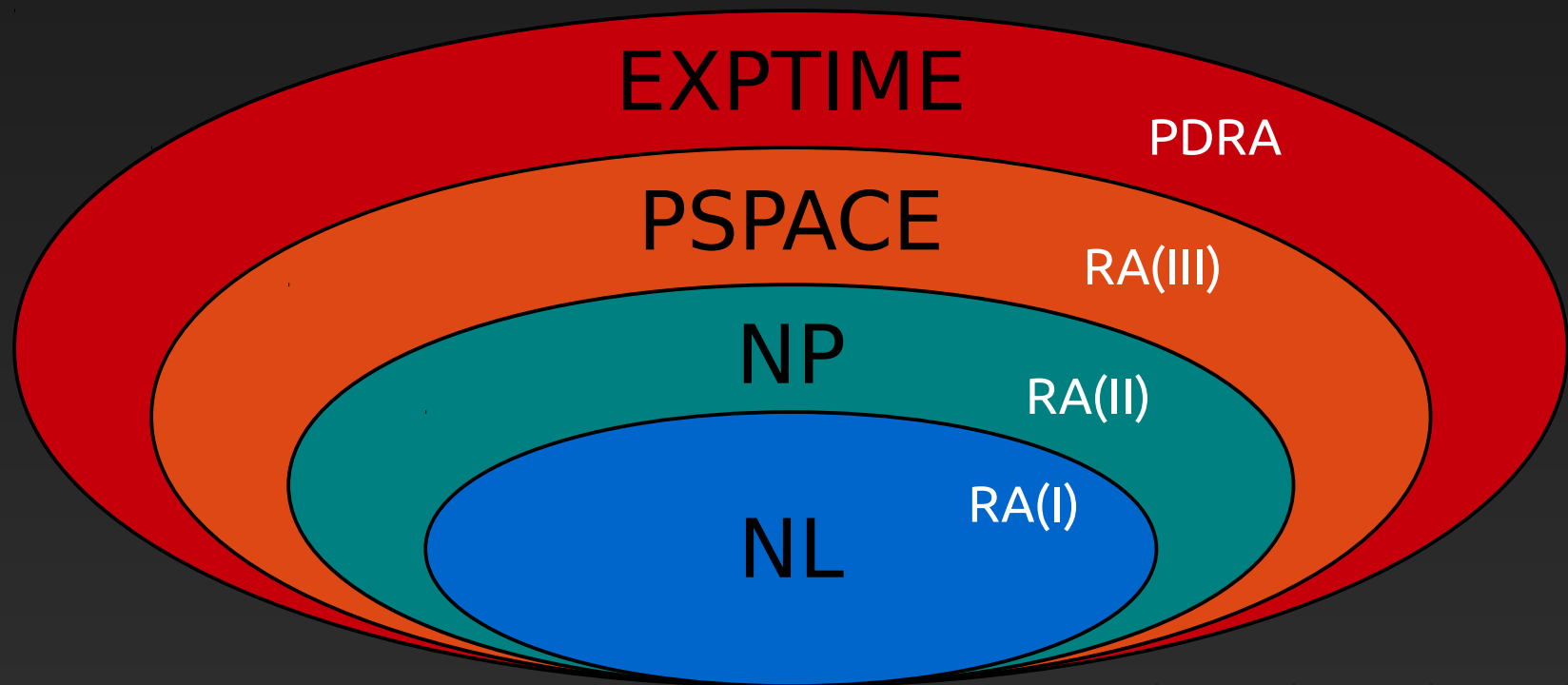
EXPTIME solvability for ( $M\#$ ):

- By previous Lemma,  $3R$  names suffice:  $\Sigma' = \{a_1, \dots, a_{3R}\}$
- so, registers can be encoded inside states:  $Q' = Q \times 3R^R$
- and we reduce to PDA reachability (PTIME)

EXPTIME hardness for ( $SF$ ):

- Reduction from TMs with stack – more difficult

# Reachability/non-emptiness picture



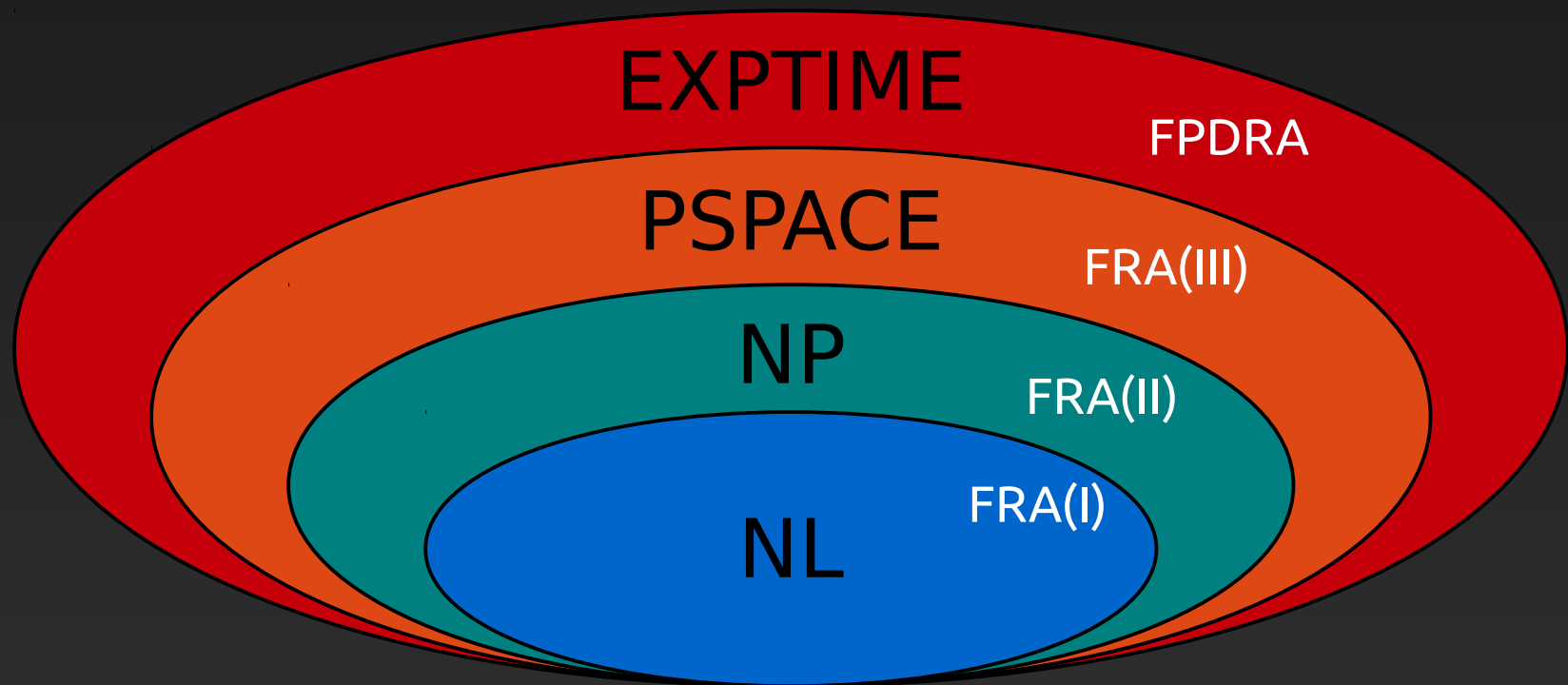
# Adding global freshness

Same as the RA  $\rightarrow$  FRA case: add **gl-fresh** transitions

- No more  $3R$  result!
- However, for reachability we can simulate global fresh:
  - by local fresh + tags on registers and stack
  - the tags ensure distinctness in every comparison
  - the simulation is exponential:  $Q' = Q \times 2^R$ 
    - summing up:  $Q'' = Q \times 2^R \times 3R^R$



# Reachability/non-emptiness for FRAs



# Concluding

## Fresh-Register Automata:

- Class of automata over infinite alphabets  
“natural” for computation with names/resources
- new landscape of algorithms and results
- applications in verification

## Things to do:

- algorithm implementations (an FRA toolkit!)
- more theory, e.g. automata learning, and applications