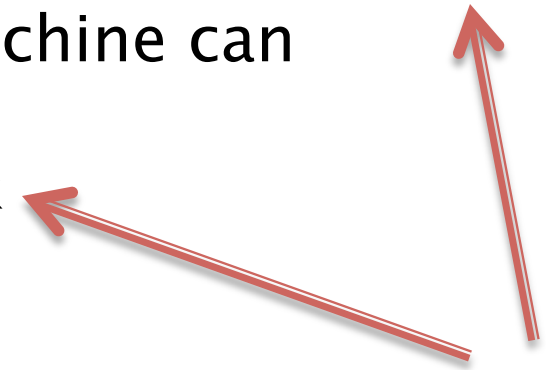
Pushdown Automata

Pushdown Automata

- In Chapter 1 we saw that regular languages are exactly the languages accepted by DFAs
- Is there a type of machine such that context-free languages are exactly the languages accepted by these machines?
  - Yes!
  - NFAs plus a little something extra
    - More than just a counter (helps for $\{0^i1^i \mid i \geq 0\}$ but not others)
    - Don't want randomly accessible memory for the entire string (would include too much $\{ ww \mid w \in \{a,b\}^* \}$)
  - A stack!

Pushdown Automata

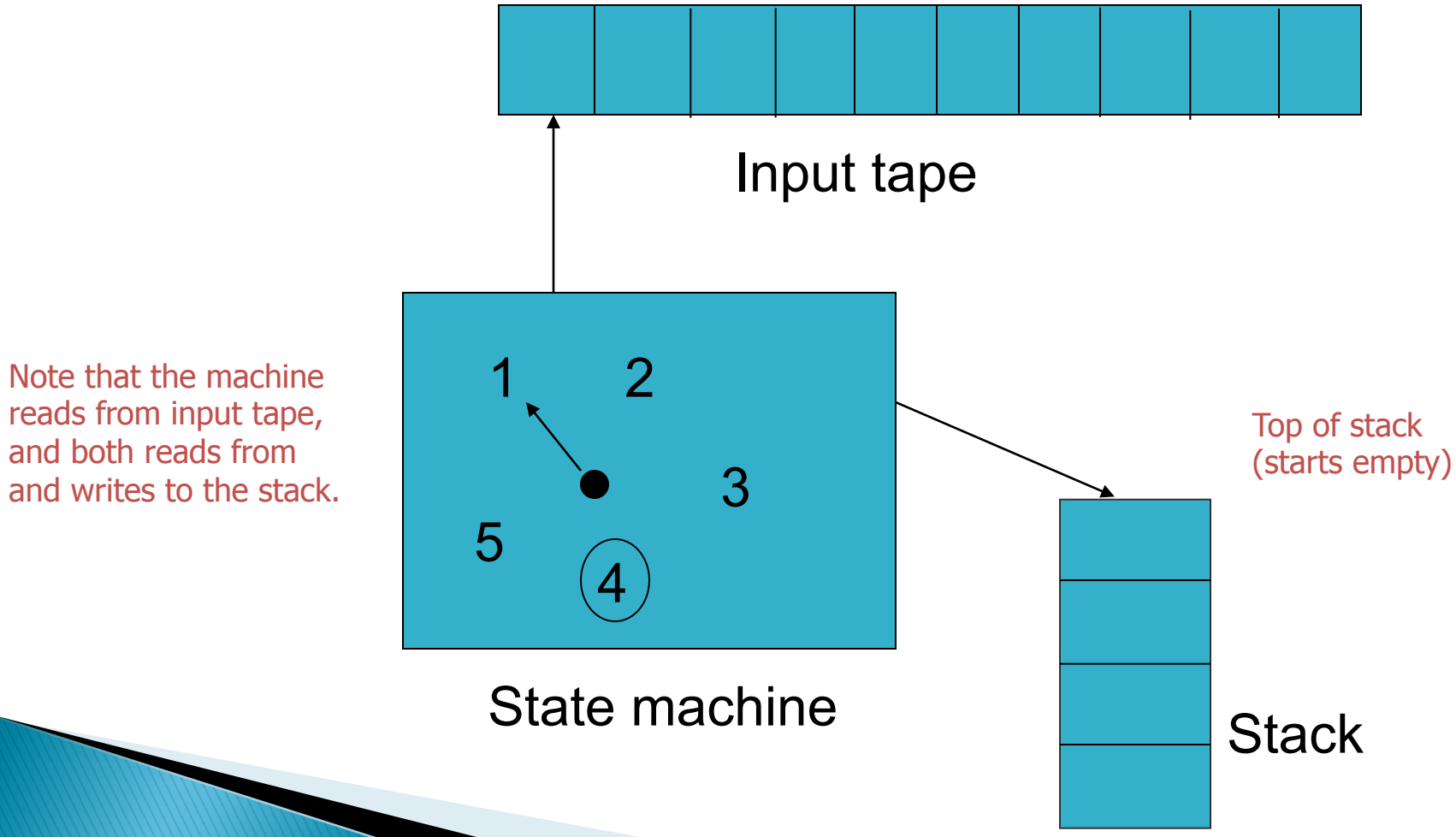
- A pushdown automata (PDA) is essentially:
  - An NFA with a stack
  - A "move" of a PDA will depend upon
    - Current state of the machine
    - Current symbol being read in
    - Current symbol popped off the top of the stack
  - With each "move", the machine can
    - Move into a new state
    - Push a symbol onto the stack



Note that since the PDA is based on the NFA, there can be more than one legal move for a given set of inputs

These are the new pieces for the pushdown automata. The other steps are the same as for an NFA.

Pushdown Automata



Pushdown Automata

- The stack
  - The stack has its own alphabet.
    - May overlap with input alphabet
    - May have additional symbols
  - Included in this alphabet is a special symbol used to indicate an empty stack. ($)

Pushdown Automata

- Let's formalize this:
  - A pushdown automata (PDA) is a 6-tuple:
    - $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where
      - Q = finite set of states
      - Σ = input tape alphabet
      - Γ = stack alphabet (may have symbols in common with Σ)
      - $q_0 \in Q$ = start state
      - $F \subseteq Q$ = set of accepting states
      - δ = transition function

Pushdown Automata

- About this transition function δ:
  - During a move of a PDA:
    - At most one character is read from the input tape
      - ϵ transitions are okay
    - At most one character is popped from the top of the stack
      - ϵ transitions are okay
    - The machine will move to a new state based on:
      - The character read from the tape
      - The character popped off the stack
      - The current state of the machine
    - At most one character from the stack alphabet is pushed onto the stack.
      - ϵ transitions are okay

Pushdown Automata

- Formally:
  - δ: Q x ($\Sigma \cup \{\epsilon\}$) x ($\Gamma \cup \{\epsilon\}$) $\rightarrow$ P(Q x ($\Gamma \cup \{\epsilon\}$))
  - Domain:
    - Q = state
    - ($\Sigma \cup \{\epsilon\}$) = symbol read off tape
    - ($\Gamma \cup \{\epsilon\}$) = symbol popped off stack
  - Range
    - Q = new state
    - ($\Gamma \cup \{\epsilon\}$) = symbol pushed onto the stack
    - P( ) is the power set – the PDA includes non-determinism, so any given input 3-tuple can branch to zero or more output 2-tuples

Pushdown Automata

- Example:
  - δ (q, a, b) = {(p, c)}
    - Meaning:
      - When in state q,
      - Reading in a from the tape
      - With b popped off the stack
    - The machine will:
      - Go into state p
      - Push c onto the stack

In the state diagram this corresponds to an arrow from state q to state p, labeled a,b → c

Pushdown Automata

- Example:
  - δ (q, ϵ, a) = {(p, b)}
    - Meaning:
      - When in state q,
      - Don't read anything from the tape
      - With a popped off the stack
    - The machine will:
      - Go into state p
      - Push b onto the stack

In the state diagram this corresponds to an arrow from state q to state p, labeled ϵ,a $\rightarrow$ b

Pushdown Automata

- Example:
  - δ (q, a, ϵ) = {(p, b)}
    - Meaning:
      - When in state q,
      - Reading in a from the tape
      - Don't pop anything off the stack
    - The machine will:
      - Go into state p
      - Push b onto the stack

In the state diagram this corresponds to an arrow from state q to state p, labeled a,$\epsilon \rightarrow$ b

Language Accepted by a PDA

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a PDA

- A string $w = y_1 y_2 \ldots y_m$ is <u>accepted</u> by M if
  - A sequence of states $r_0 r_1 \ldots r_m$ exists
  - And a sequence of strings $s_0 s_1 \ldots s_m$ exists
  - with the conditions:
    - $r_0 = q_0$ and $s_0 = \epsilon$
      - Start at start state with empty stack
    - $(r_{i+1}, b) \in \delta (r_i, y_{i+1}, a)$ for $i = 0, \ldots, m-1$
      - where $s_i = at$, $s_{i+1} = bt$ for $a,b \in (\Gamma \cup \{\epsilon\})$, $t \in \Gamma^*$
      - The machine is run on string w
    - $r_m \in F$
      - The machine ends in a final state.

$y_i \in \Sigma \cup \{\epsilon\}$

$r_i \in Q$

$s_i \in \Gamma^*$

s_i represents the contents of the stack at any given time

Language Accepted by a PDA

- Note that there can be more than one sequence of states and stack contents that exist and satisfy the conditions necessary to accept the string
- Note that for string $w = y_1y_2...y_m$, y_i can be ϵ
  - This is how to represent moves in which no symbol is read, but the stack contents may be modified
- Note that the end-state contents of the stack do not matter with respect to acceptance
  - Often, however, the contents of the stack are used to determine whether to enter an accepting state

Pushdown Automata

- Let's look at an example:
  - $L = \{ a^n \# b^n \mid n \geq 0 \}$

- Basic idea…
  - As you read a's you "count" them by placing on the stack.
  - When you encounter #, a's should be done.
  - When you read b's, you match them against the a's
  - If you have an empty stack at the end of reading the string, all a's have been matched with b's, thus, machine should accept.

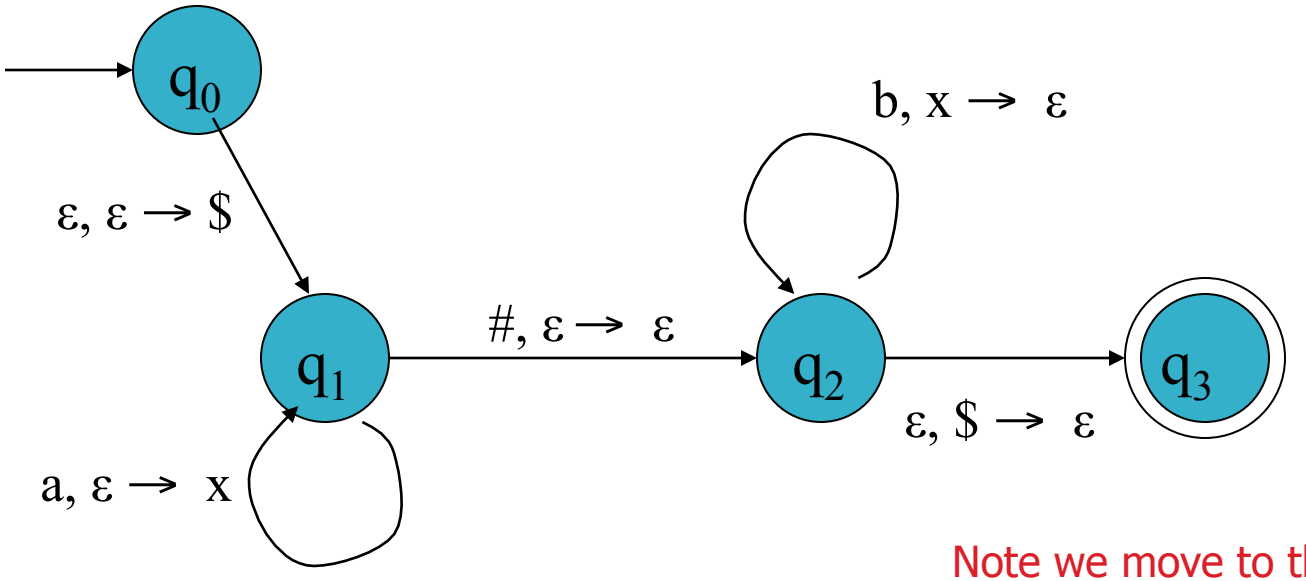
  - Note: must be able to detect the empty stack.

Pushdown Automata

- Let's look at an example:
  - $L = \{ a^n \# b^n \mid n \geq 0 \}$
  - The PDA will have 4 states
    - State 0 (initial): push empty stack marker
    - State 1: reading a's
    - State 2: reading and matching b's
    - State 3 (accepting): move to only if the stack is empty

Pushdown Automata

- Let's look at an example:
  - $L = \{ a^n \# b^n \mid n \geq 0 \}$



Note we move to the accept state only when stack is empty, and there are no transitions from the accept state, so we will stay in an accept state only if there are no more symbols to be read

Pushdown Automata

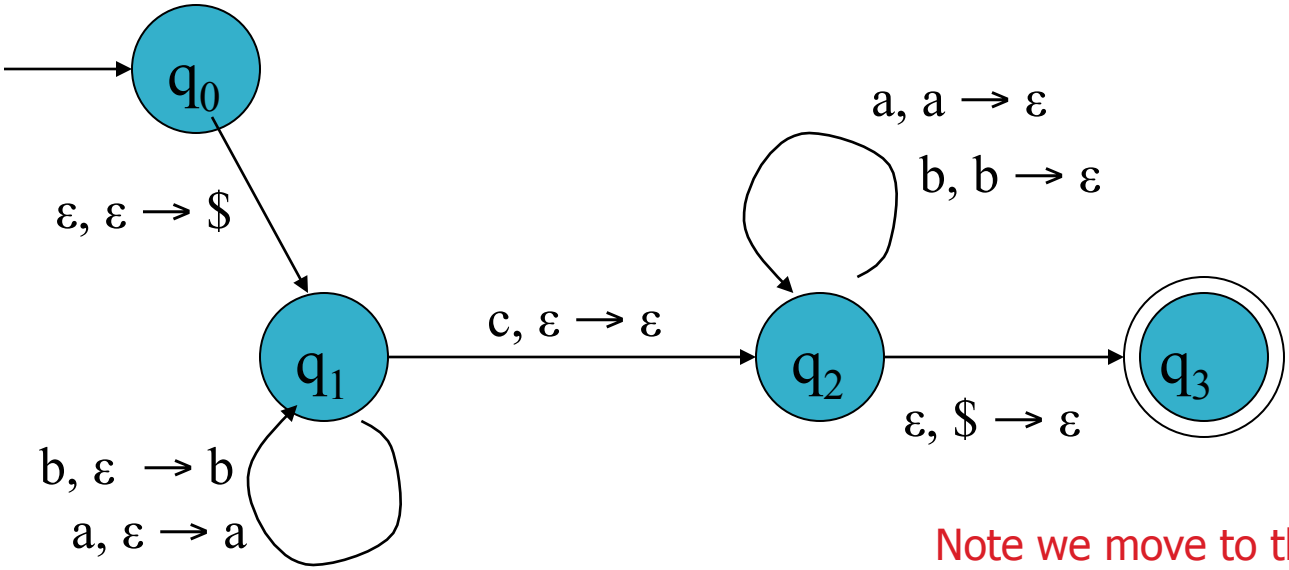
- Let's look at an example:
  - $L = \{ xcx^r \mid x \in \{ a,b \}^* \}$
  - Basic idea for building a PDA
    - Read symbols off the tape until you reach the c.
    - As you read symbols push them on the stack
    - After reading the c, begin matching symbols being read with symbols popped off the stack until all symbols are read
    - If at any point the symbol read does not match the symbol popped, the machine "crashes"

Pushdown Automata

- Let's look at an example:
  - $L = \{ xcx^r \mid x \in \{ a,b \}^* \}$
  - The PDA will have 4 states
    - State 0 (initial): push empty stack marker
    - State 1: reading before the c
    - State 2: read after c, comparing symbols
    - State 3 (accepting): move to only if stack is empty

Pushdown Automata

- Let's look at an example:
  - $L = \{ xcx^r \mid x \in \{ a,b \}^* \}$



Note we move to the accept state only when stack is empty, and there are no transitions from the accept state, so we will stay in an accept state only if there are no more symbols to be read

Pushdown Automata

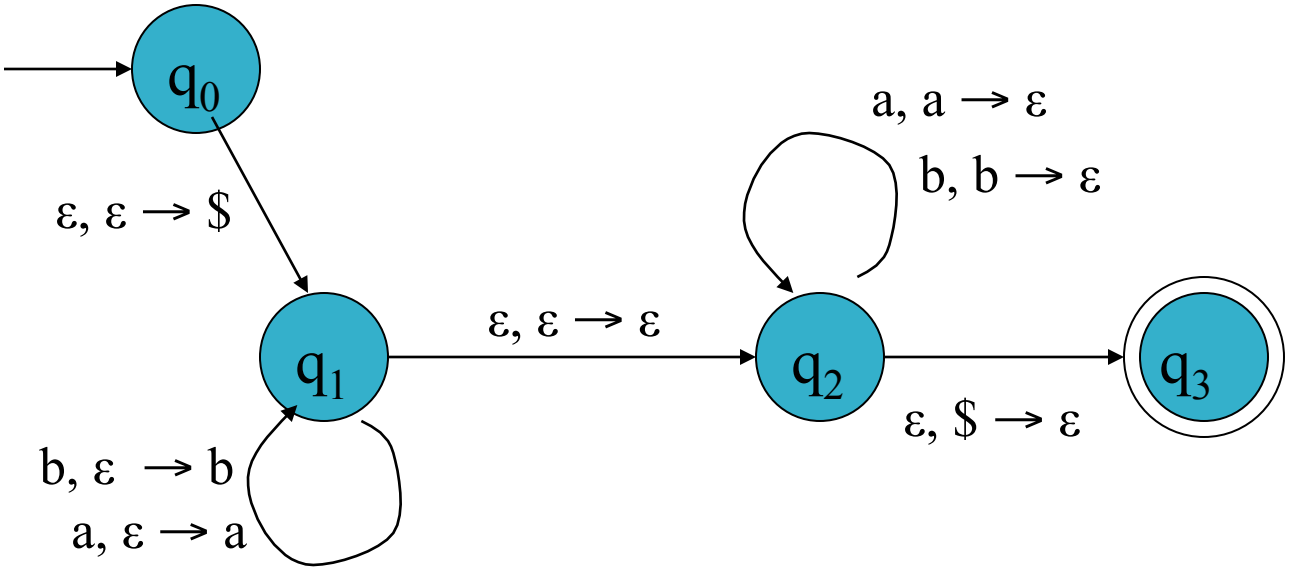
- Let's look at another example:
  - $L = \{ xx^r \mid x \in \{ a,b \}^* \}$
  - Basic idea for building a PDA
    - Much like last example, except
      - This time we don't know when to start popping and comparing
      - Since PDAs are non-deterministic, this is not a problem

Pushdown Automata

- Let's look at another example:
  - $L = \{ xx^r \mid x \in \{ a,b \}^* \}$
  - The PDA will have 4 states
    - State 0 (initial): push empty stack marker
    - State 1: reading before the center of string
    - State 2: read after center of string, comparing symbols
    - State 3 (accepting): move to only if stack is empty
  - The machine can choose to go from state 1 to state 2 at any time:
    - Will result in many "wrong" sets of moves
    - All you need is one "right" set of moves for a string to be accepted.

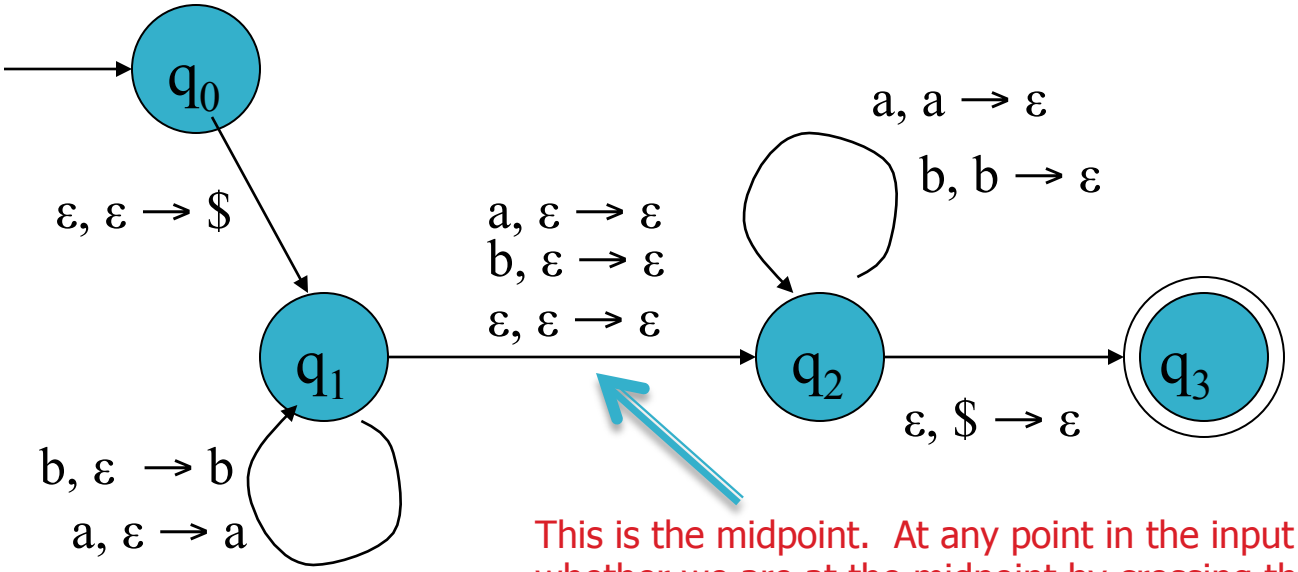
Pushdown Automata

- Let's look at an example:
  - $L = \{ xx^r \mid x \in \{ a,b \}^* \}$



Pushdown Automata

- By the previous two examples, we have effectively created a PDA that accepts all palindromes over {a,b}*



This is the midpoint. At any point in the input string we can probe whether we are at the midpoint by crossing this path. If we're probing for an even length string, we'll use the epsilon transition. Otherwise, we'll probe assuming our current input symbol (a or b) is the middle symbol of the palindrome. We can send out lots of probes, and obviously most of them will be wrong and the path will die out, but we only need one to accept…