# Backtracking

Lecture12

# Recursion

- We have seen divide and conquer:

— split into subproblems of size n/c (some c).

— Analyze running time with recursion trees.

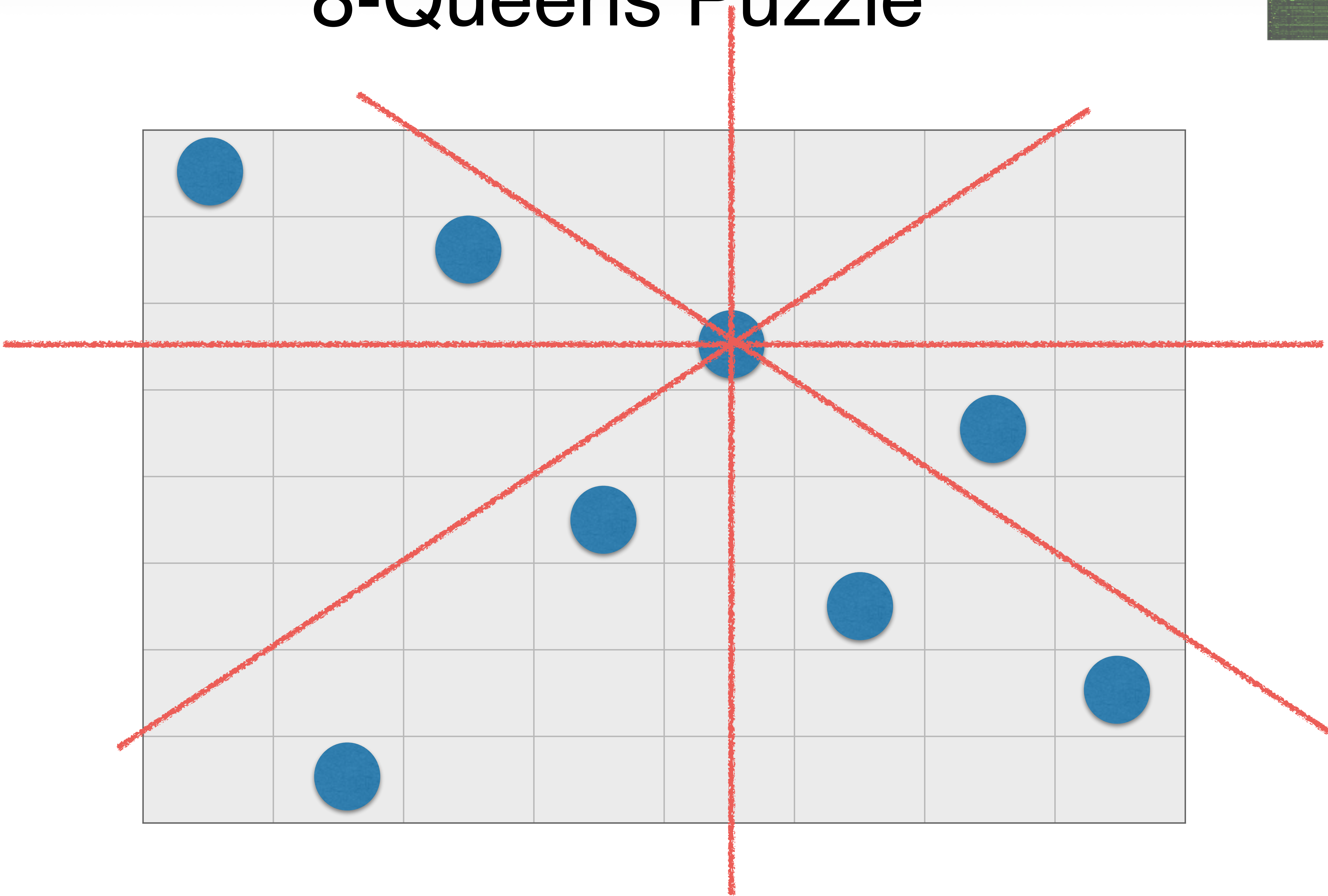
- Different style of recursion: Backtracking

— reduce to subproblems of smaller size n-c (some c).

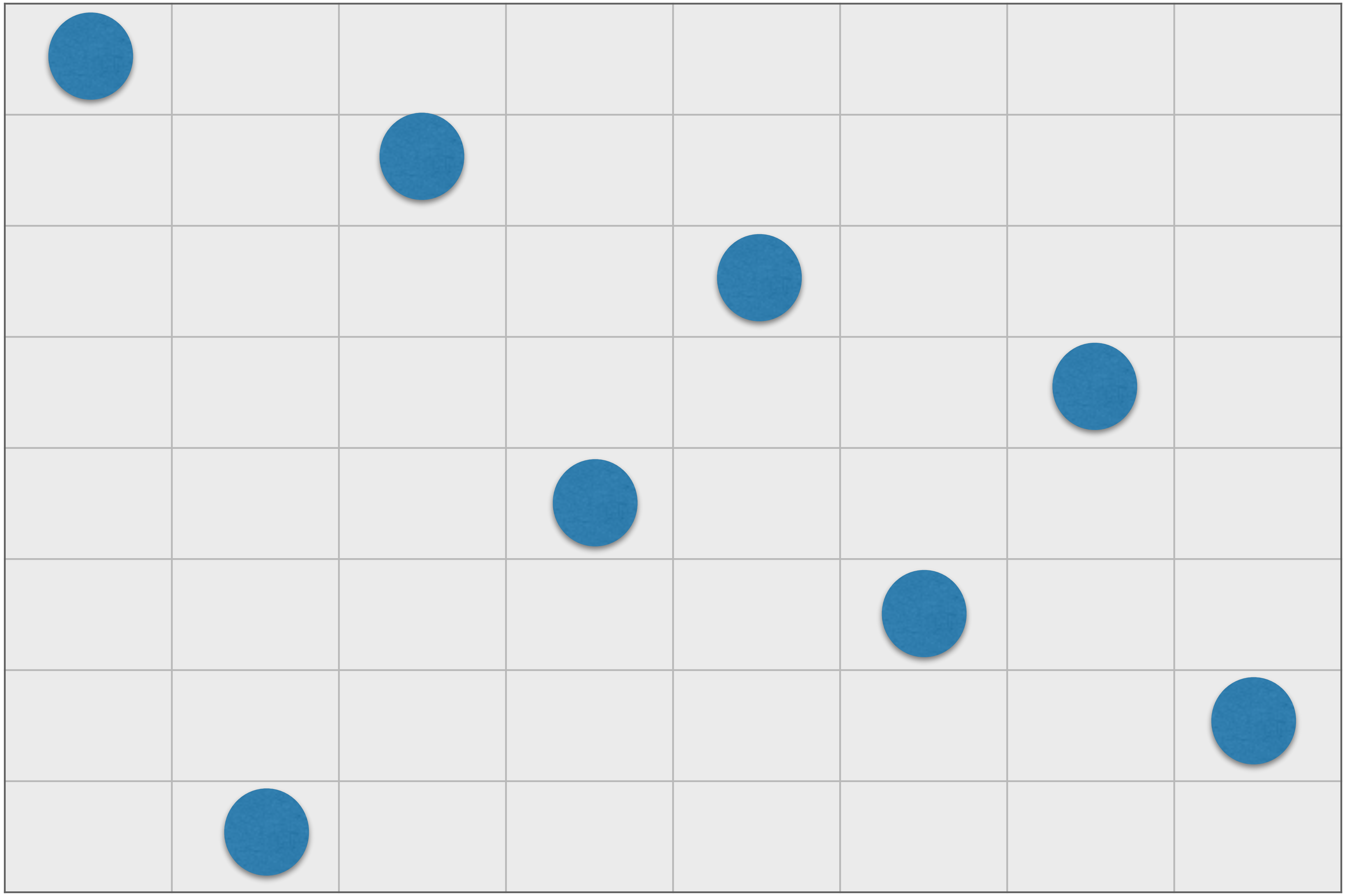
— Usually exponential time

— Way of developing correct recursive algorithms, won't deal with running time often.

# 8-Queens Puzzle

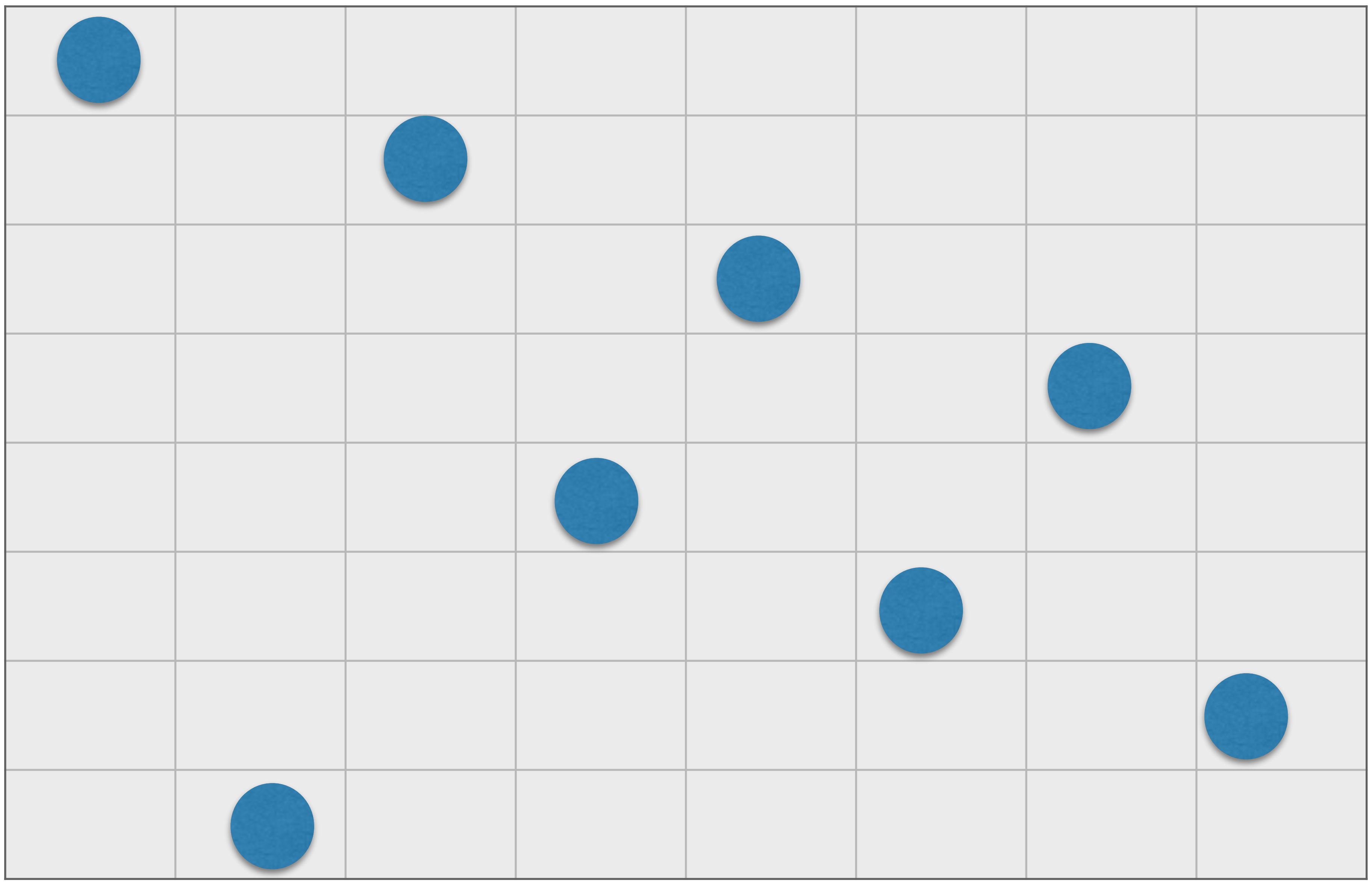


# 8-Queens Puzzle



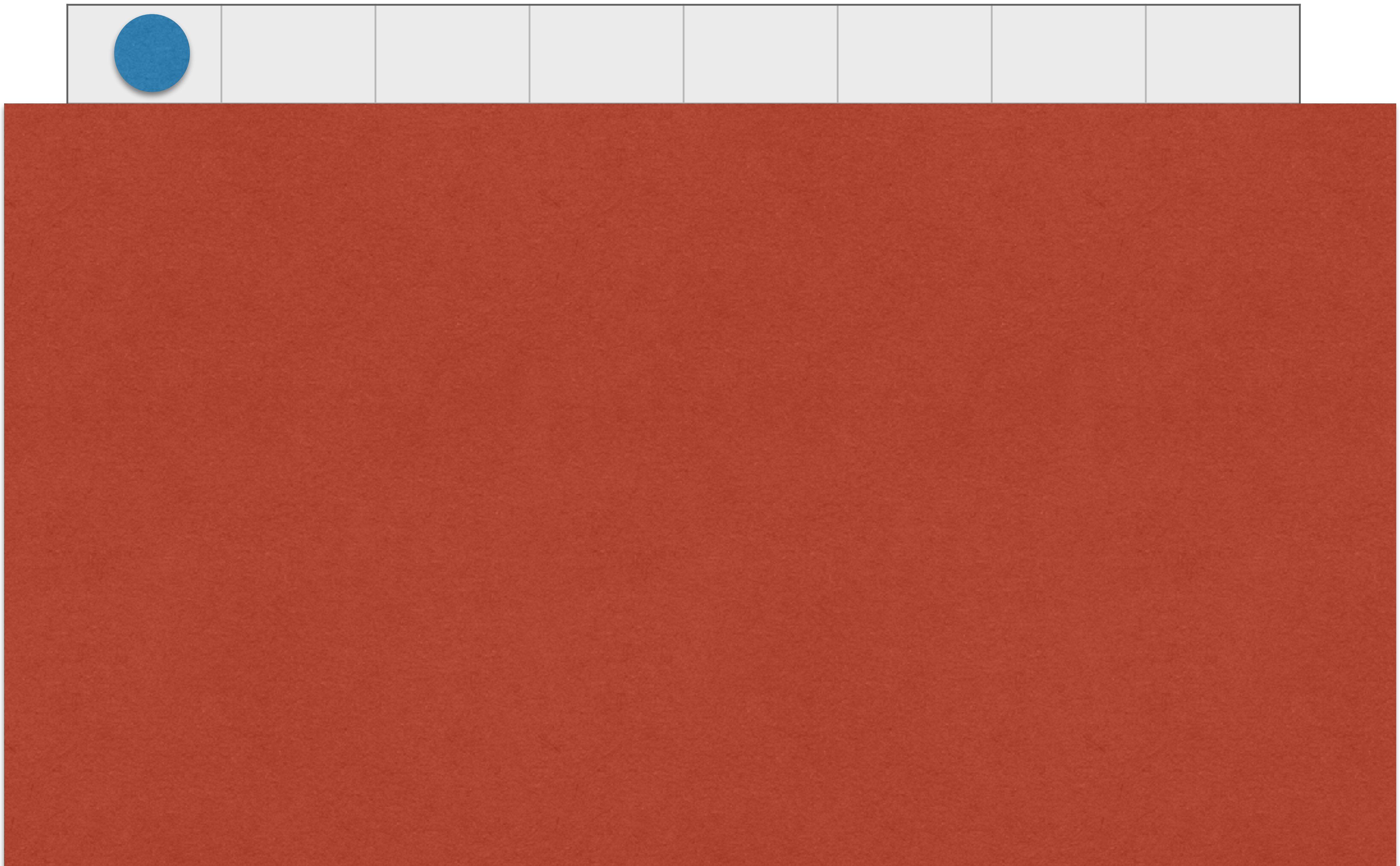
How  long does it take to solve it from scratch?

# n-Queens Puzzle



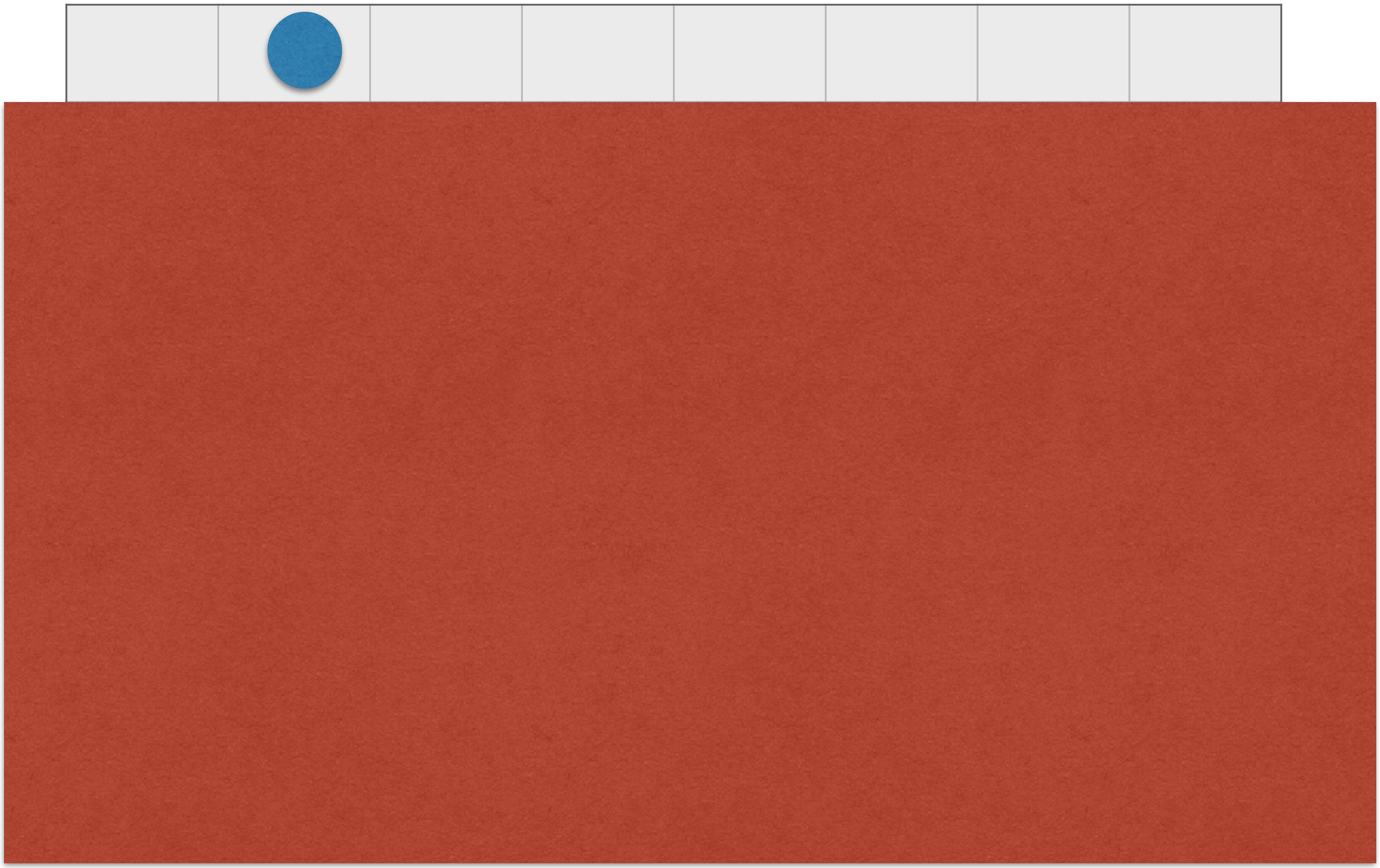
Represent by array Q[1…n].
Q[i] = which square in row i has a queen

# n-Queens Puzzle



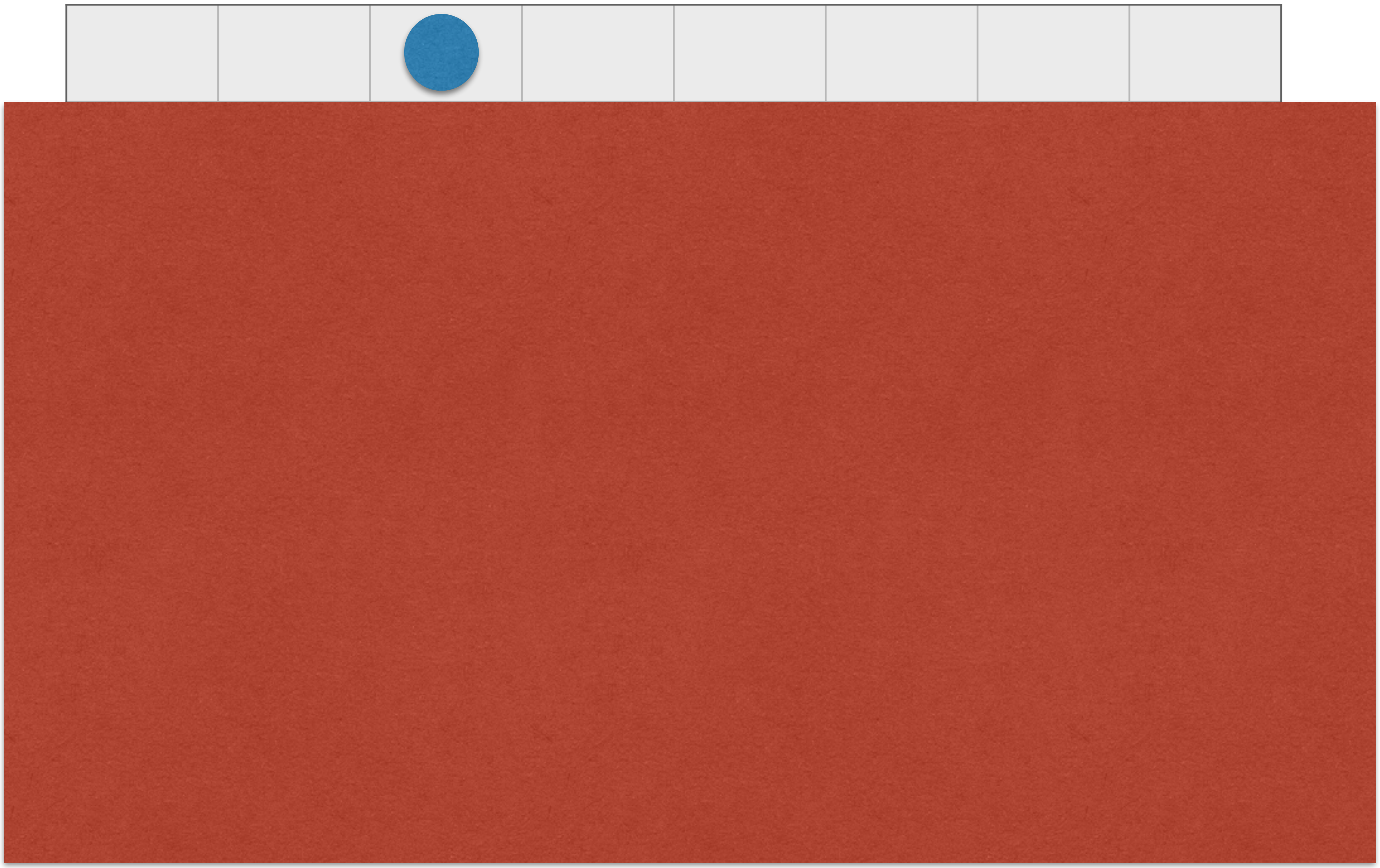
Place a queen at the first empty row-try all possible places

# n-Queens Puzzle



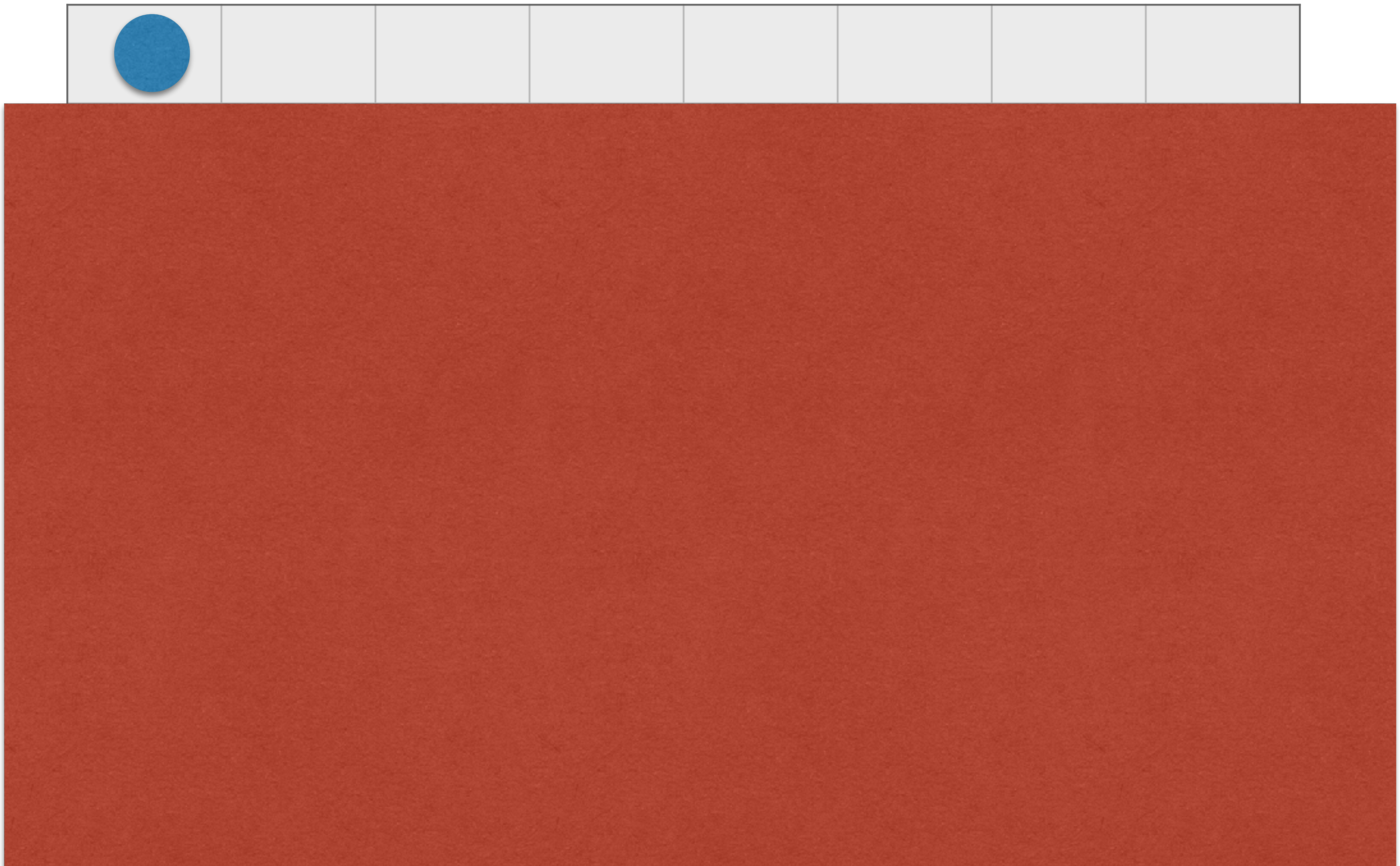
Place a queen at the first empty row-try all possible places

# n-Queens Puzzle



Place a queen at the first empty row-try all possible places

# n-Queens Puzzle

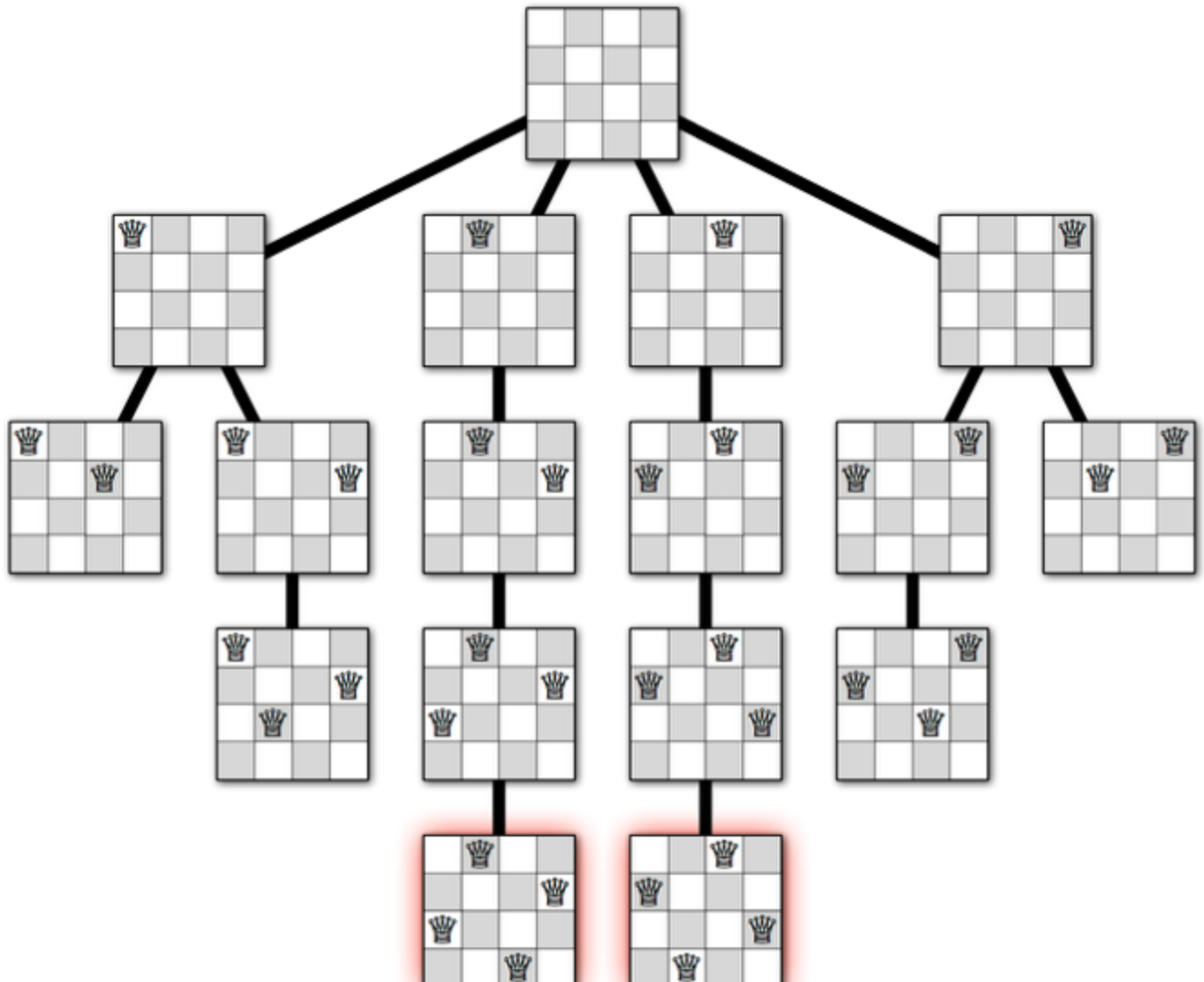


Place a queen at the first empty row-try all possible places

# n-Queens Puzzle

```
RecursiveNQueens(Q[1..n], r):
  if r = n + 1
      print Q
  else
      for j ← 1 to n
          legal ← True
          for i ← 1 to r − 1
              if (Q[i] = j) or (Q[i] = j + r − i) or (Q[i] = j − r + i)
                  legal ← False
          if legal
              Q[r] ← j
              RecursiveNQueens(Q[1..n], r + 1)
```

# n-Queens Puzzle



# Subset sum

- Given a set X of positive integers and a target positive integer t, is there a subset of elements in X that add up to t?
- Given X, find A subset of X, so that $\sum A=t$?
- What is the first element to go into A?
- Try them all!
- If there is an element equal to t, done
- If t is zero, we are done! (why?)
- If t negative, no!

# Subset sum

- Given a set X of positive integers and a target positive integer t, is there a subset of elements in X that add up to t?
- Given X, find A subset of X, so that $\sum A = t$?
- Assume t is positive and no element bigger than t.

# Subset sum

- Given a set X of positive integers and a target positive integer t, is there a subset of elements in X that add up to t?
- Given X, find A subset of X, so that ∑A=t?
- Example: X={3,2,4,6,9}, t = 7
- What element to try first?
- Say x= 6. Then is there subset of {3,2,4,9} that adds to 1? NO

# Subset sum

- Given a set X of positive integers and a target positive integer t, is there a subset of elements in X that add up to t?
- Given X, find A subset of X, so that $\sum A=t$?
- Example: X={3,2,4,6,9}, t = 7
- What element to try first?
- Say x= 6. Then is there subset of {3,2,4,9} that adds to 1? NO
- Two cases: x in A or x not in A.

# Subset sum

- If there is a subset A with $\sum A=t$ then either

- x in A, call SubsetSum(X-{x},t-x)

- or x not in A call SubsetSum(X-{x},t)

# Subset sum

$$
\begin{array}{l}
\underline{\textsc{SubsetSum}(X[1..n], T):} \\
\quad \text{if } T = 0 \\
\qquad \text{return } \textsc{True} \\
\quad \text{else if } T < 0 \text{ or } n = 0 \\
\qquad \text{return } \textsc{False} \\
\quad \text{else} \\
\qquad \text{return } \big(\textsc{SubsetSum}(X[1..n-1], T) \vee \textsc{SubsetSum}(X[1..n-1], T - X[n])\big)
\end{array}
$$

Call the algorithm with i=n

Canonical order to choose elements in the subset

# Subset sum

- Running time?

- $T(n) \leq O(1) + 2T(n-1)$
- Tower of Hanoi! exponential time $2^n$
- Brute force!
- NP-Hard!

# NFA acceptance

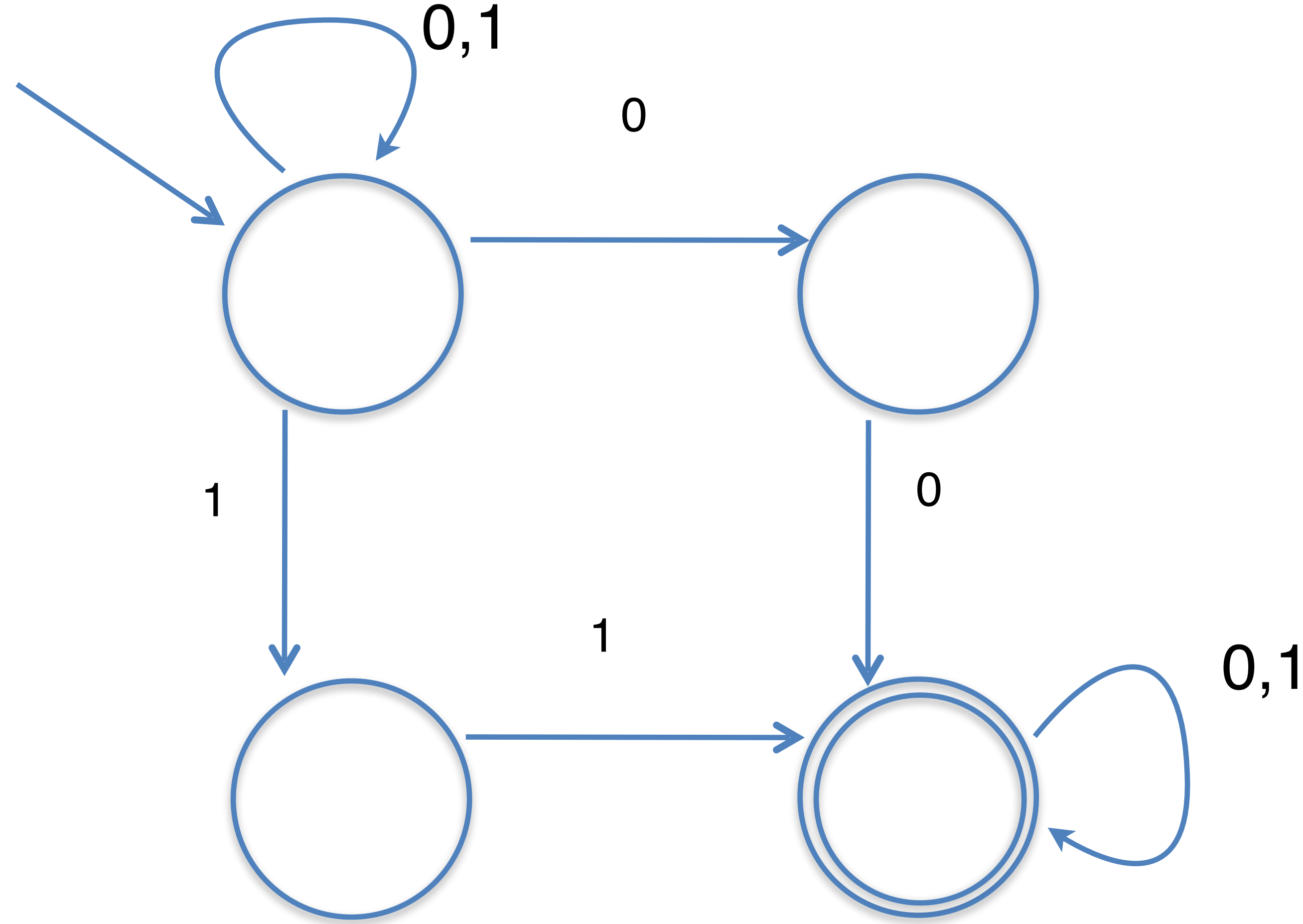
- Given NFA : $N = (\Sigma, Q, \delta, s, A)$ and $w \in \Sigma^*$

  is $\delta^*(s, w) \cap A \neq \emptyset$

- Is there a walk in N from s to an accepting state labeled w?

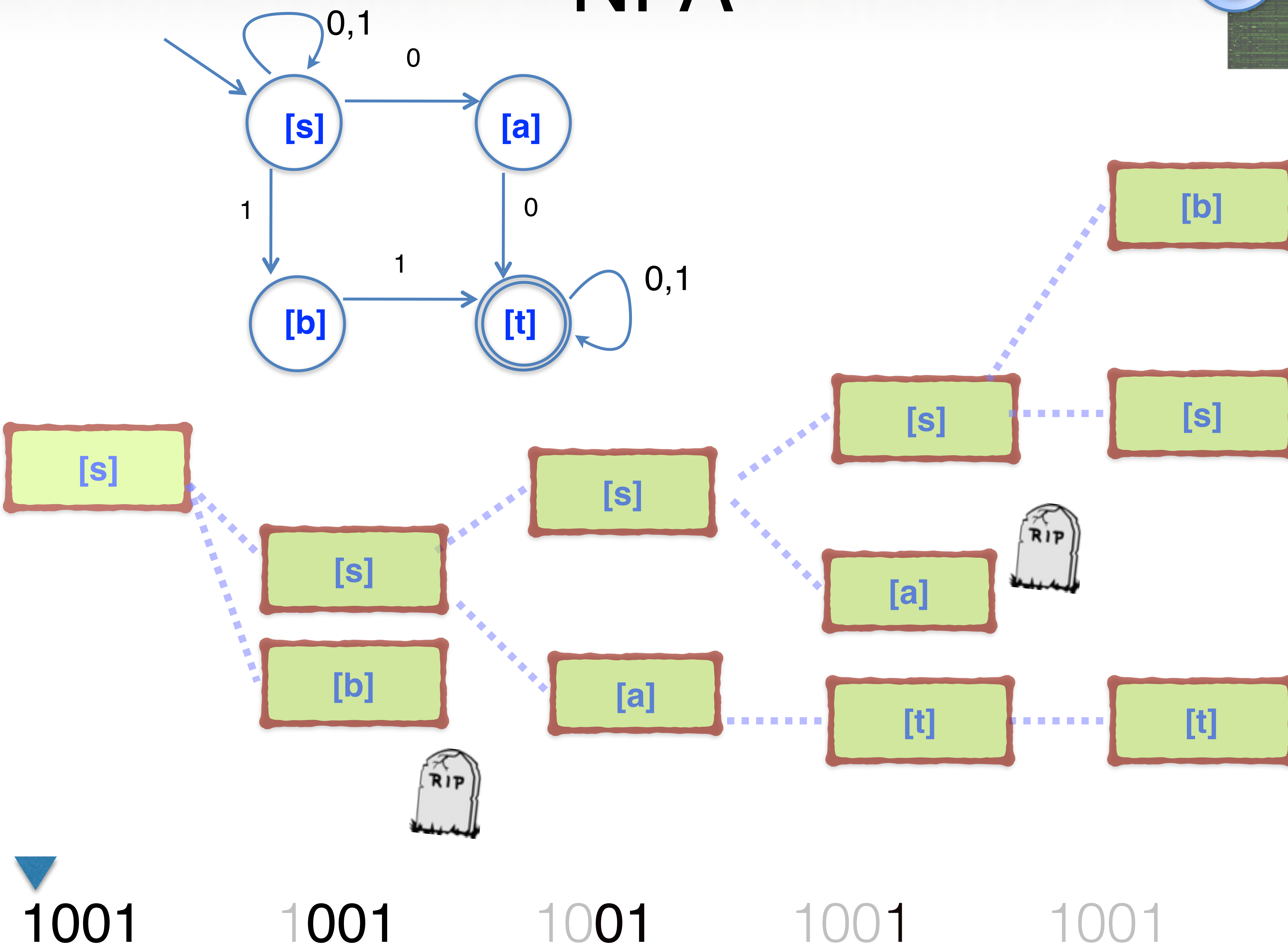
# NFA acceptance

- Input = 01001

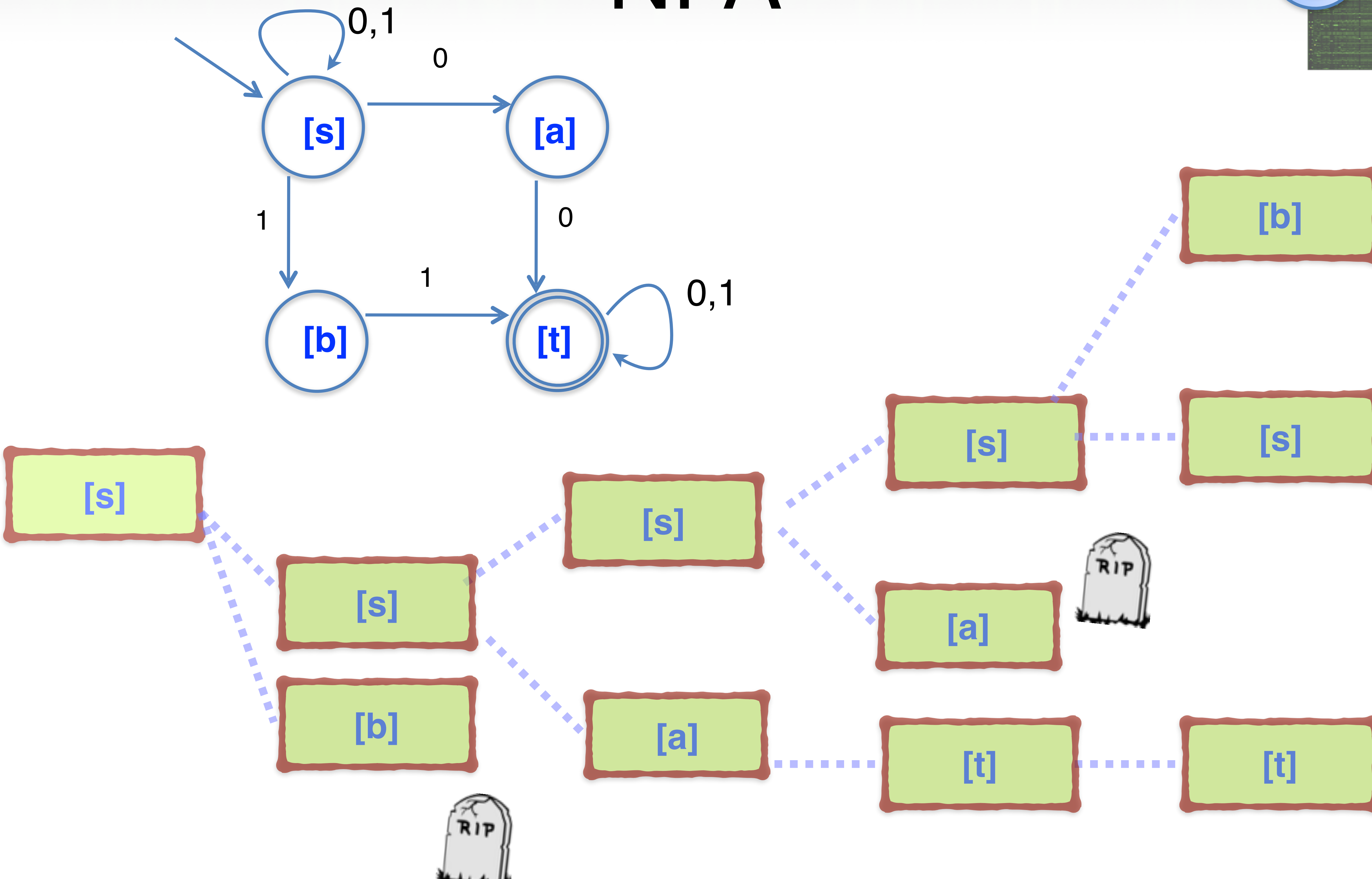



- L ={contains either 00 or 11}

# NFA

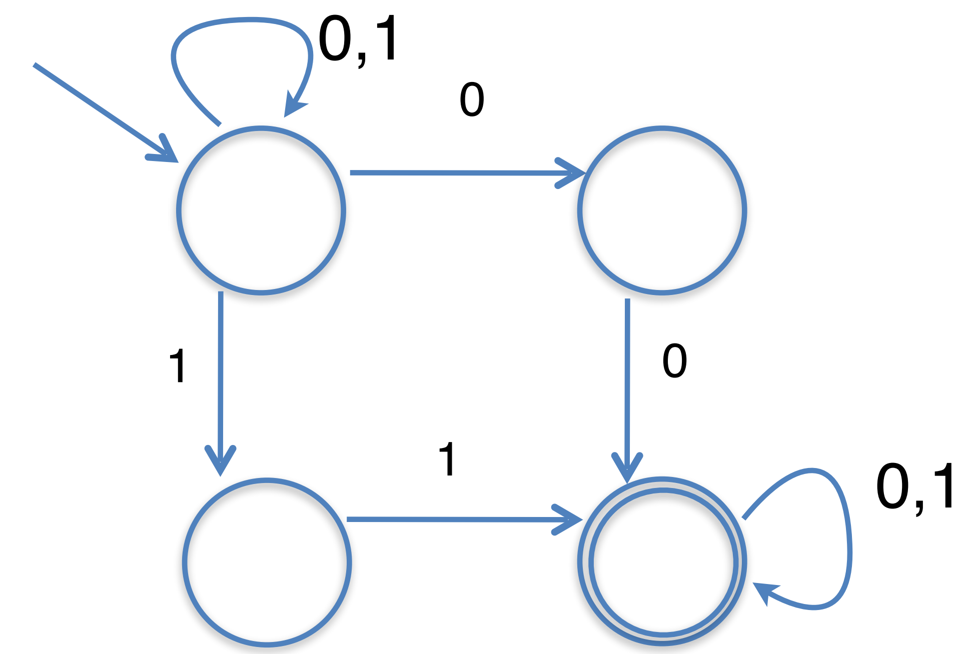


# NFA




One of the states are accepting. There needs to be AT LEAST one accepting state

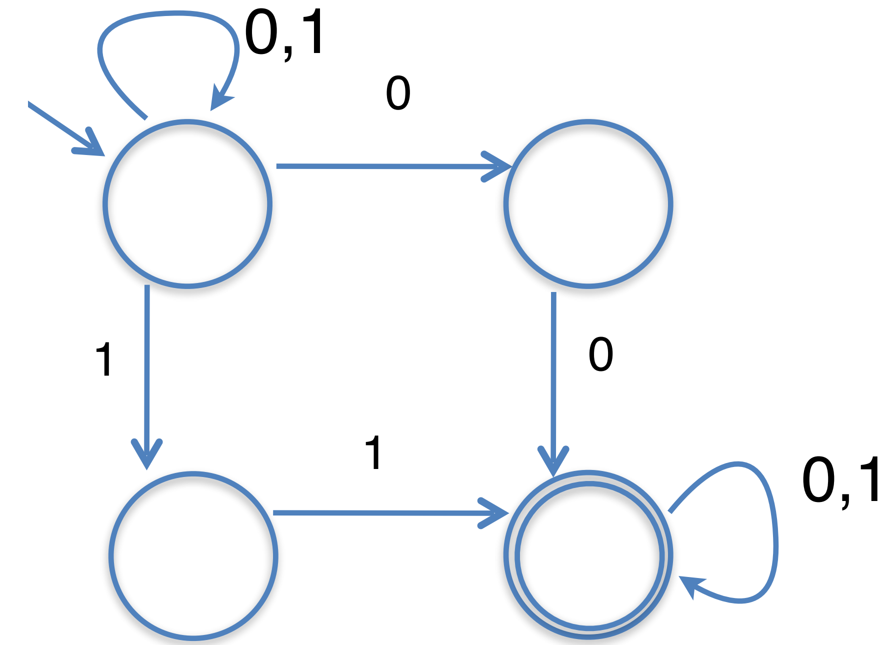
# NFA acceptance




- Input = 01001
- How do I decide what to do once I read the first 0?
- Try both! maybe one of them will work.
- Smaller subproblem, when we need to figure out if the NFA accepts a smaller input.
- Need to specify what state the NFA is in and what string is left to read.
- Accept (q,w)

# NFA acceptance

```
ACCEPTS?(q, w[1..n]):
    if n = 0
        return A[q]
    for all states r
        if δ[q, w[1], r] and ACCEPTS?(r, w[2..n])
            return TRUE
    return FALSE
```




- A[i] is 1 iff i is an accepting state.
- δ[q,w[1],r] =1 iff r∈δ(q,w[1])
- Every time the recursion branches, there are at most Q states
- $Q^n$ upper bound on running time!!!

# Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8
- Subsequence different than substring.
- Increasing = in an order.
- Recursion?

# Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8
- Look at first element. Keep or ditch?

What went wrong?
I didn't use
INCREASING

- LIS(A[1…n])

If n< $10^{10}$, brute force

keep: 1+LIS(A[2…n])

ditch: LIS(A[2…n])

# Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8

- LIS(A[1…n])

If n< $10^{10}$, brute force

keep: 1+ ?

ditch: LIS(A[2…n])

- What is the correct subproblem?
- LIS where every number is larger than the number p I keep
- Not the same problem anymore!

# Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8

- LIS(A[1…n], p)

If n< $10^{10}$, brute force

keep:

ditch:

- What are the new cases?
- Either use A[1] or not.
- Anything else?

# Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8
  - LIS(A[1…n],p)

If n< $10^{10}$, brute force

If A[1] ≤ p,

RETURN LIS(A[2…n],p)

else

RETURN MAX: LIS(A[2…n],p)
1+LIS(A[2…n],A[1])

# Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8

  - LIS(A[1…n],p)

If n< $10^{10}$, brute force

If A[1] ≤ p,

RETURN LIS(A[2…n],p)

else

RETURN MAX: LIS(A[2…n],p) / 1+LIS(A[2…n],A[1])

- LIS(A[1…n],−∞) to find LIS
- Running time?
- $2^n$