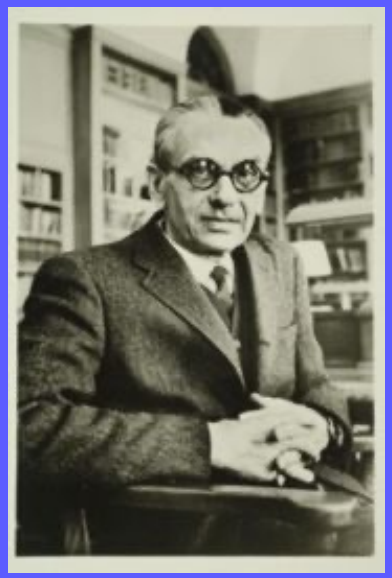
Algorithms

NP-Completeness

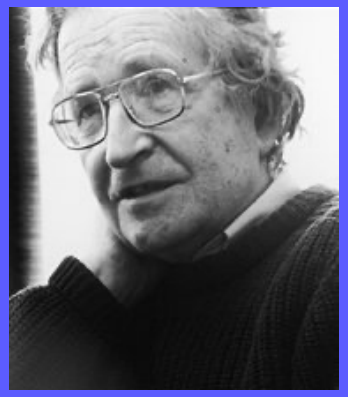
(pp. 341~357)



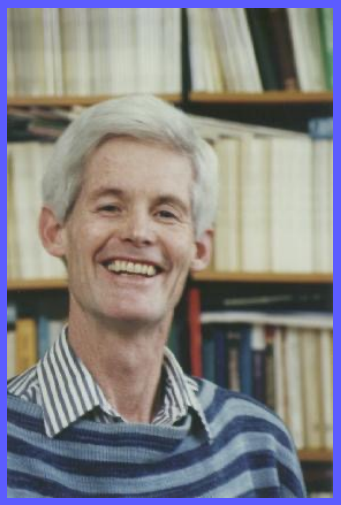
Kurt Gödel
Apr. 28, 1906

Alonzo Church
Jun. 14, 1903

Alan Turing
Jun. 23, 1912



Noam Chomsky
Dec. 7, 1928



Stephen Cook
Dec. 14, 1939

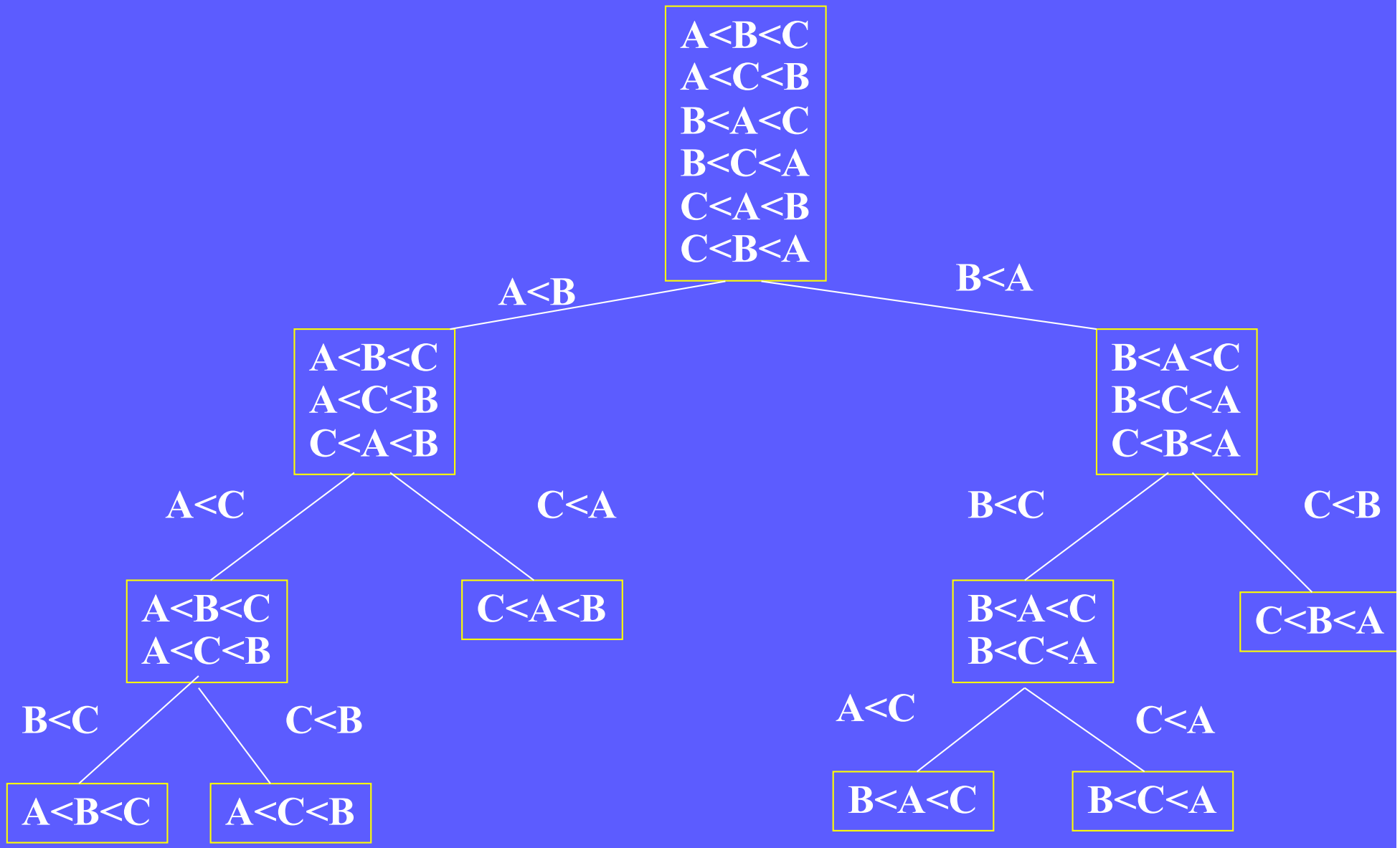
Leonid Levin
Nov. 2, 1948

How Difficult of the Sorting Problem ?

Difficulty of Problems

- **How difficult is sorting problem ?**
- **What is the lower bound of time complexity of sorting ?**
  - **古人: the most efficient algorithm has been found**
  - **來者 ?**
- **How to prove ?**
  - **sorting**
    - decision tree with n! leaves
    - depth O(log n!)
    - O(nlogn) comparisons

Decision Tree for Sorting Three Element



Proof of Lower Bound of Sorting

- A binary tree T of depth *d* has at most 2^d leaves
- A binary tree with *L* leaves have depth at least $\lceil \log L \rceil$
- Any comparison-based sorting algorithm requires at least $\lceil \log(N!) \rceil$ comparisons in the worst case.
- Any comparison-based sorting algorithm requires $\Omega(N \log N)$ comparisons.

  (log(N!) >= (N/2)*log(N/2))

How Difficult of

Any Problem ?

Difficulty of Problems

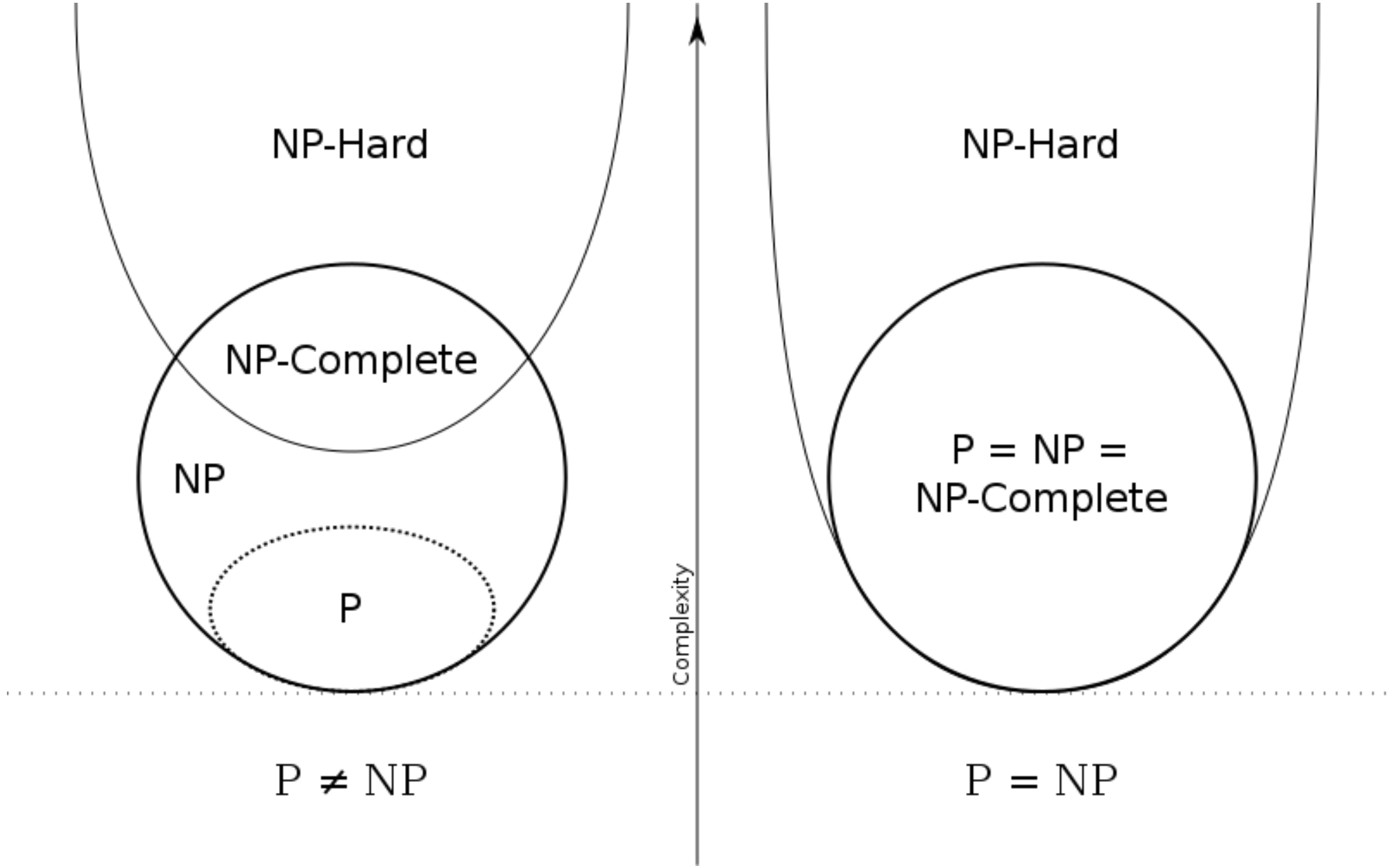
- **Tractable problems**
  - **Problems that can be solved by efficient algorithms**
  - **Efficient algorithms**
    - Running time is O(P(n)), where P(n) is a polynomial in the size of input n.
  - **P (Polynomial time)**
- **Intractable problems**
  - **Problems not to be solvable in polynomial time**
  - **NP-Hard**

Difficulty of Problems (cont.)

Until now, three types of problems

1. **Problems for which polynomial-time algorithms have been found (P)**
2. **Problems that have been proven to be intractable (NP-Hard)**
3. **Problems that have not been proven to be intractable, but for which polynomial-time algorithms have never been found (NP)**

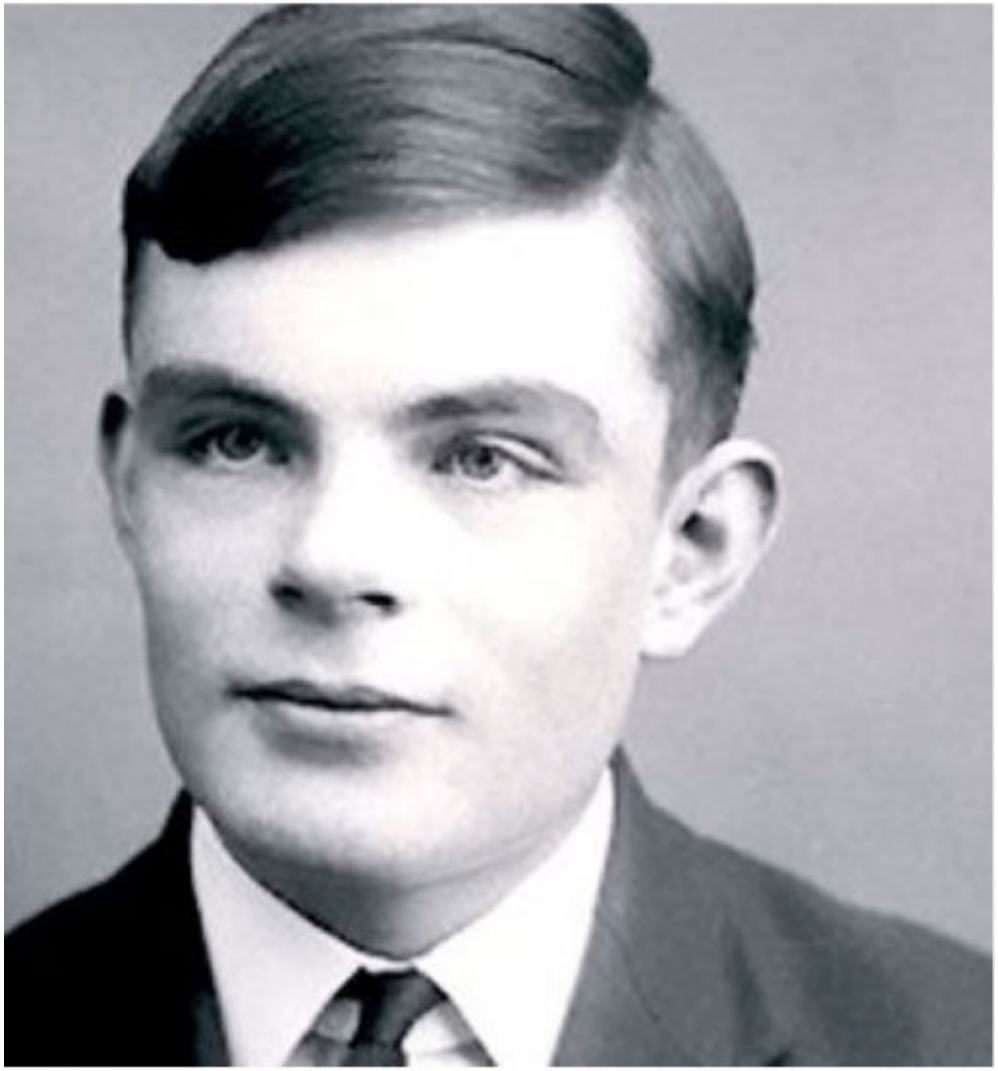
*** NP-Complete: NP Problem that is NP-Hard**


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP

What is a computer ?

Universal Computer Model ?

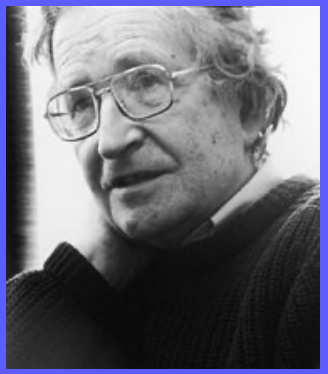
Turing Machine



Language, Grammar, Machine & Problem

- **Language: accepted by a machine**
- **Grammar: rules to generate a language**
- **Machine: recognize a language**

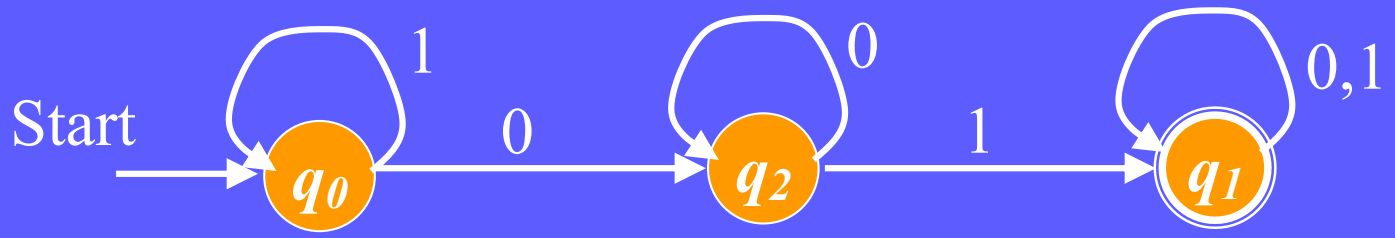
Chomsky Hierarchy



	Language	Machine	Grammar
Type-0	**Recursively Enumerable Language (RE)**	**Turing Machine**	$\alpha \rightarrow \beta$
Type-1	**Context Sensitive Language (CSL)**	**Linear-bounded Non-deterministic Turing Machine**	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	**Context Free Language (CFL)**	**Push-Down Automata (PDA)**	$A \rightarrow \alpha$
Type-3	**Regular Language (RL)**	**Finite State Machine (FSM)**	$A \rightarrow \alpha B$

Finite State Machine

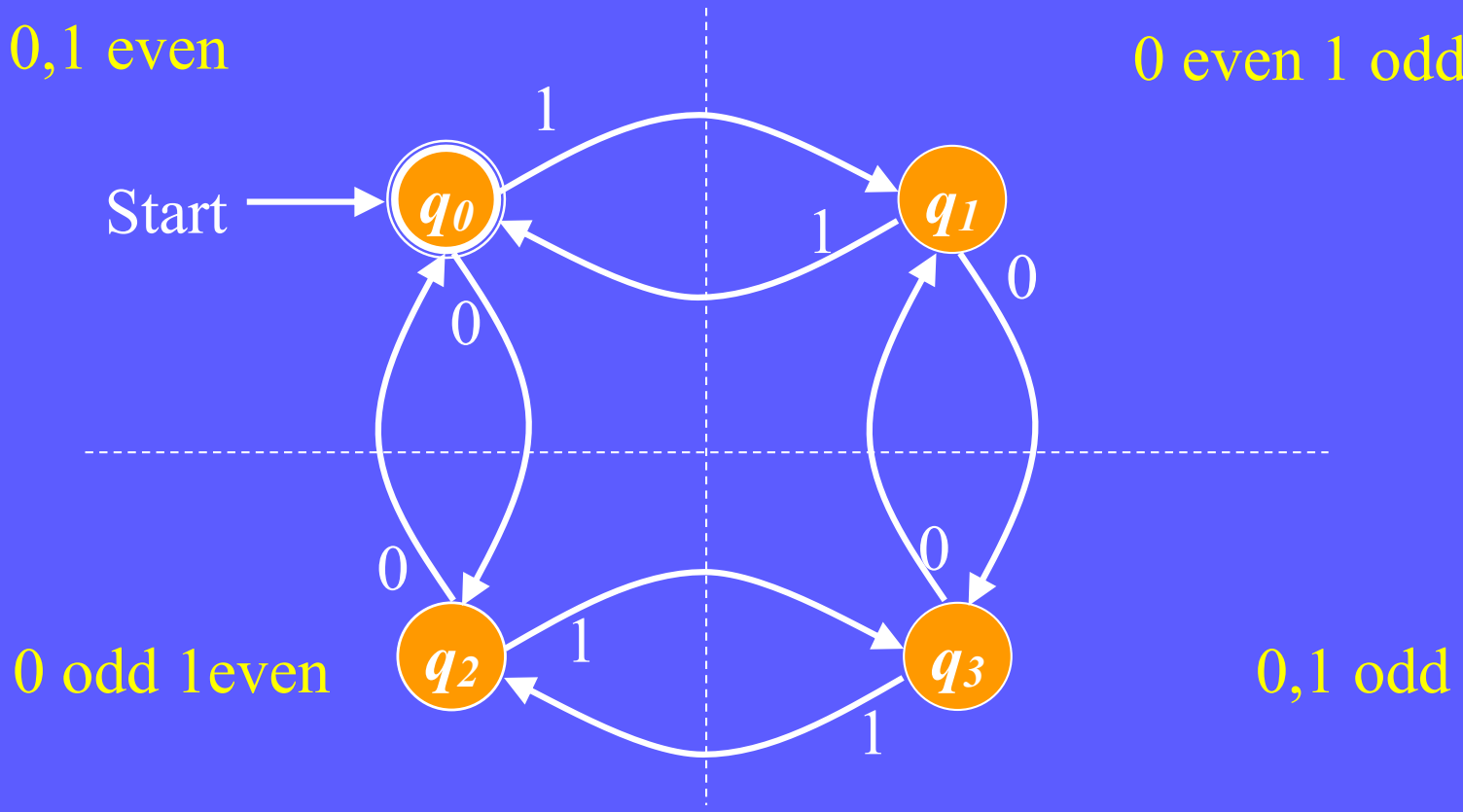
- **A Finite State Machine** $A = (Q, \Sigma, \delta, q_0, F)$
  1. **A finite set of states, *Q***
  2. **A finite set of input symbols, $\sum$**
  3. **A transition function δ: *Q* x $\sum \rightarrow Q$, e.g. $\delta(q_2, 1) = q_1$**
  4. **A start state q_0**
  5. **A set of final states *F***



$$A = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_1\})$$

- **regular language {x01y | x, y are strings of 0's and 1'}**
  = {01, 0010, 11010, 100011, 0011101, …}

Finite State Machine (cont.)

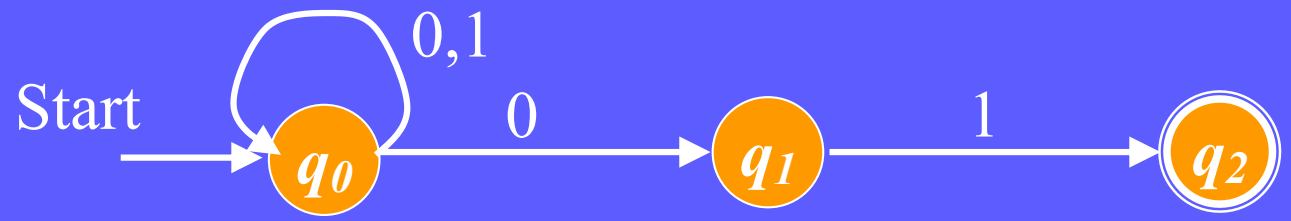


$$A = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_0\})$$

- **{w|w has both an even number of 0's & an even number of 1's} = {00, 0101, 100010 …}**

Nondeterministic Finite State Machine

- **A Nondeterministic FSM** $A = (Q, \Sigma, \delta, q_0, F)$
  1. **a finite set of states, *Q***
  2. **a finite set of input symbols, Σ**
  3. **a transition function δ: $Q \times \Sigma \rightarrow 2^Q$, e.g. $\delta(q_0, 0) = (q_0, q_1)$**
  4. **a start state q_0**
  5. **a set of final states *F***

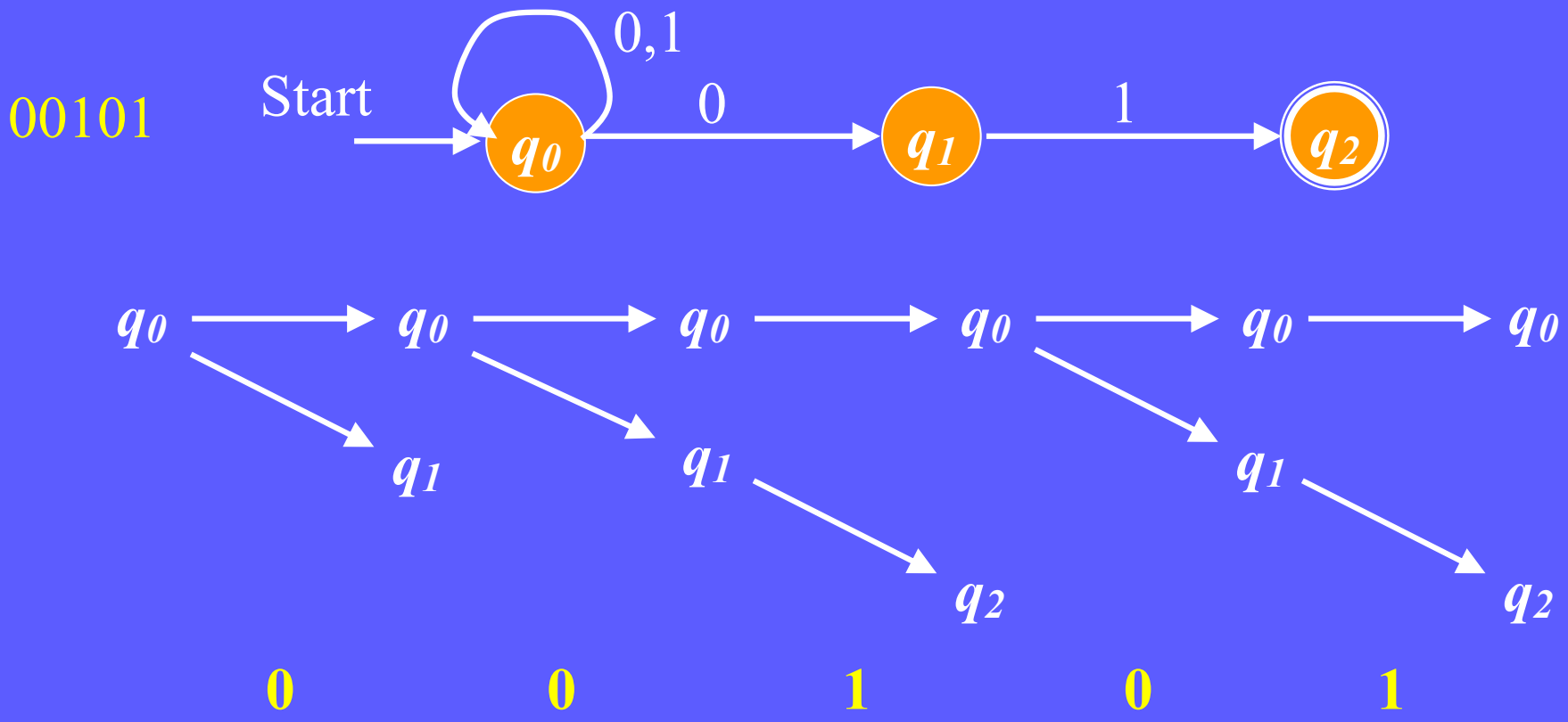


- $A = (Q, \Sigma, \delta, q_0, F) = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_2\})$

  • A regular language {xy01 | x, y are strings of 0's and 1'}
  = {01, 0001, 1001, 11011001, 10111011101, …}

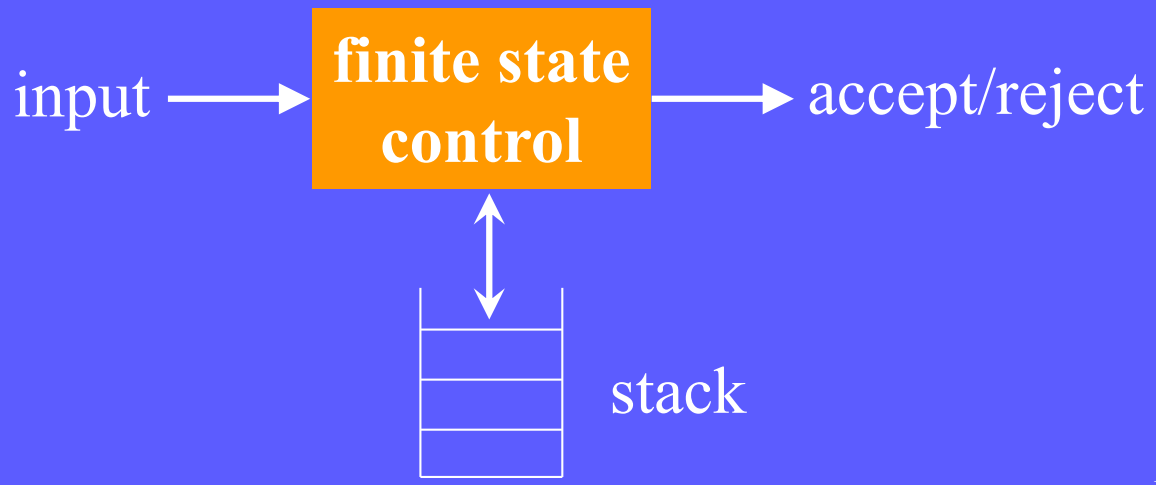
Nondeterministic Finite State Machine (cont.)

- **Nondeterministic finite state machine**
  - **has the power to be in several states at once**
  - **an ability to "guess" something about the input**



Pushdown Automata

- **Pushdown automata**
  - **a nondeterministic finite automata & a stack on which it can store a string of stack symbols**
  - **unlike finite automata, pushdown automata can remember an infinite amount of information**
  - **however, pushdown automata can only access the information on stack in a last-in-first-out way**
  - **pushdown automata recognize all & only the context-free language**



Pushdown Automata (cont.)

- **A PDA** $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

  Q : *a* finite set of states,

  $\sum$: a finite set of input symbols,

  Γ: a finite stack alphabet

  δ: a transition function δ: $Q \times \sum \times \Gamma \rightarrow Q \times \Gamma^*$

  q_0 : the start state

  Z_0: the start symbol

  F : the set of final states

Pushdown Automata (cont.)

- **e.g. A PDA**

$$P = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

$0, Z_0 \rightarrow 0Z_0$
$1, Z_0 \rightarrow 1Z_0$
$0, 0 \rightarrow 00$
$0, 1 \rightarrow 01$
$1, 0 \rightarrow 10$
$1, 1 \rightarrow 11$

e.g. $\delta(q_0, 0, Z_0) = \{q_0, 0Z_0\}$
$\delta(q_0, \varepsilon, 1) = \{q_1, 1\}$
$\delta(q_1, 0, 0) = \{q_1, \varepsilon\}$

$0, 0 \rightarrow \varepsilon$
$1, 1 \rightarrow \varepsilon$

$\varepsilon, Z_0 \rightarrow Z_0$
$\varepsilon, 0 \rightarrow 0$
$\varepsilon, 1 \rightarrow 1$

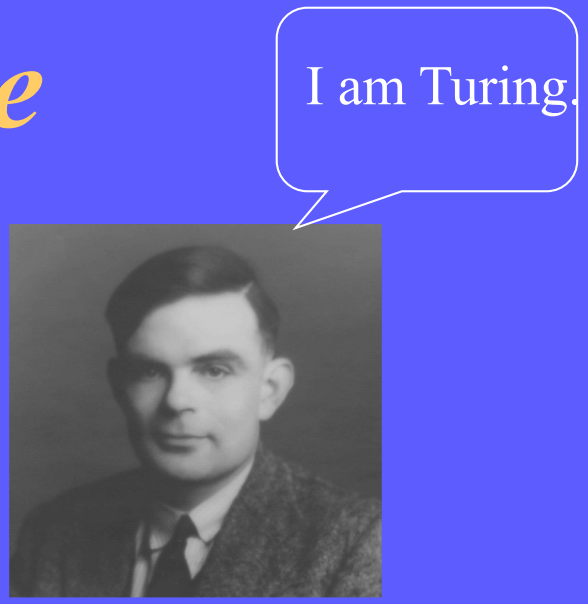
$\varepsilon, Z_0 \rightarrow Z_0$

L = $\{ww^R | w \text{ is in } (0+1)^*\}$ = {11, 1001, 110011, …}

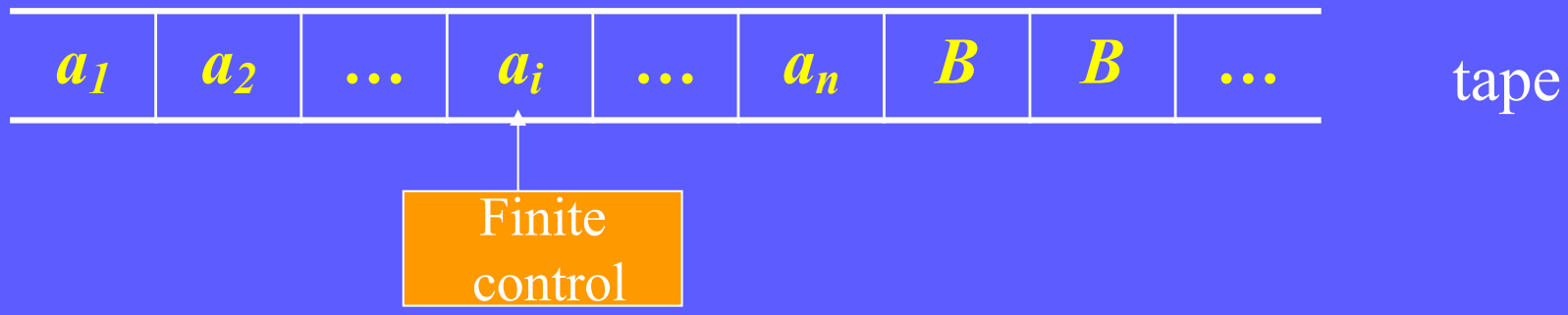
Context Free Language

- **The language accepted by final state by some PDA is a context free language**
- **Examples of Context Free Language**
  - $L = \{ww^R | w \text{ is in } (0+1)^*\}$
  - $L = \{a^n b^m c^{2(n+m)} \mid n \geq 0, m \geq 0\}$
  - $L = \{a^i b^j c^k \mid i = 2j \text{ or } j = 2k\}$
  - $L = \{0^n 1^m \mid n \leq m \leq 2n\}$
- **Examples of Non-context Free Language**
  - $L = \{ww \mid w \text{ is in } (0+1)^*\}$
  - $L = \{a^n b^n c^n \mid n \geq 1\}$
  - $L = \{a^n b^n c^i \mid i \leq n\}$
  - $L = \{a^i b^j c^k \mid i < j < k\}$

Turing Machine



- **Introduced by Alan Turing in 1936**
- **M=(Q, Γ, B, Σ, δ, q_0, F)**
  - **Q: the finite set of states**
  - **Γ: finite set of allowable tape symbols**
  - **B: a symbol of Γ, blank**
  - **Σ: the set of input symbols, a subset of Γ not including B**
  - **δ: next move function, a mapping from Q×Γ to Q×Γ×{L, R}**
  - **q_0: start state**
  - **F: the set of final states**



Example of Turing Machine

- **Subtraction m-n, input: 0^m10^n, output: 0^{m-n}**
- **Turing machine M works as follows**
  - **Repeat**

    M replaces its leading 0 by blank

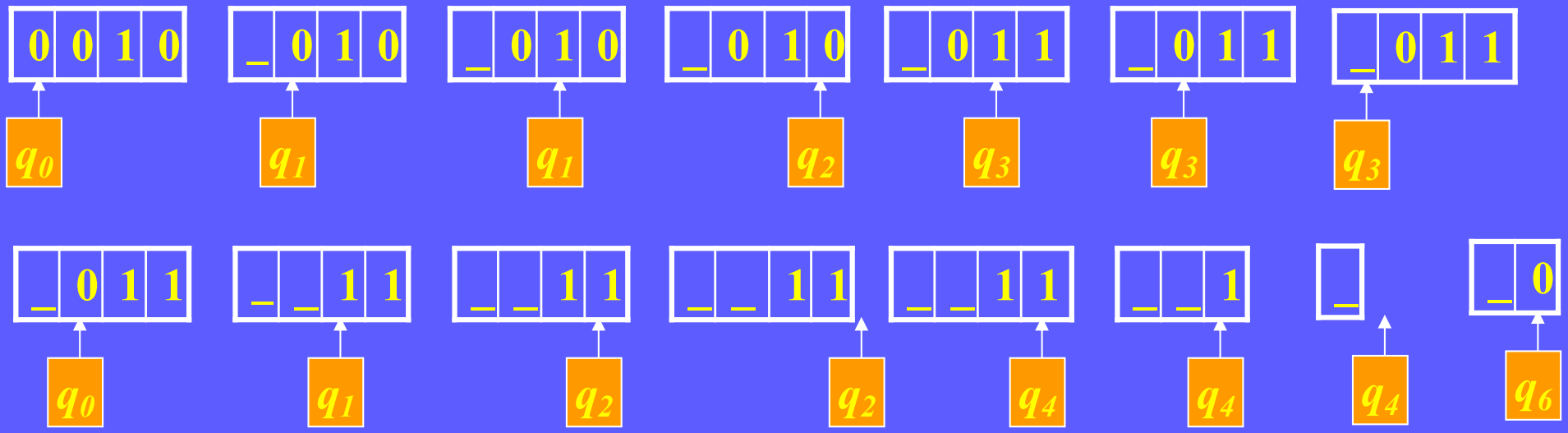
    searches right for a 1 followed by a 0 and changes the 0 to 1

    M moves left until it encounters a blank

  Until (when M searches right, encounters a blank)

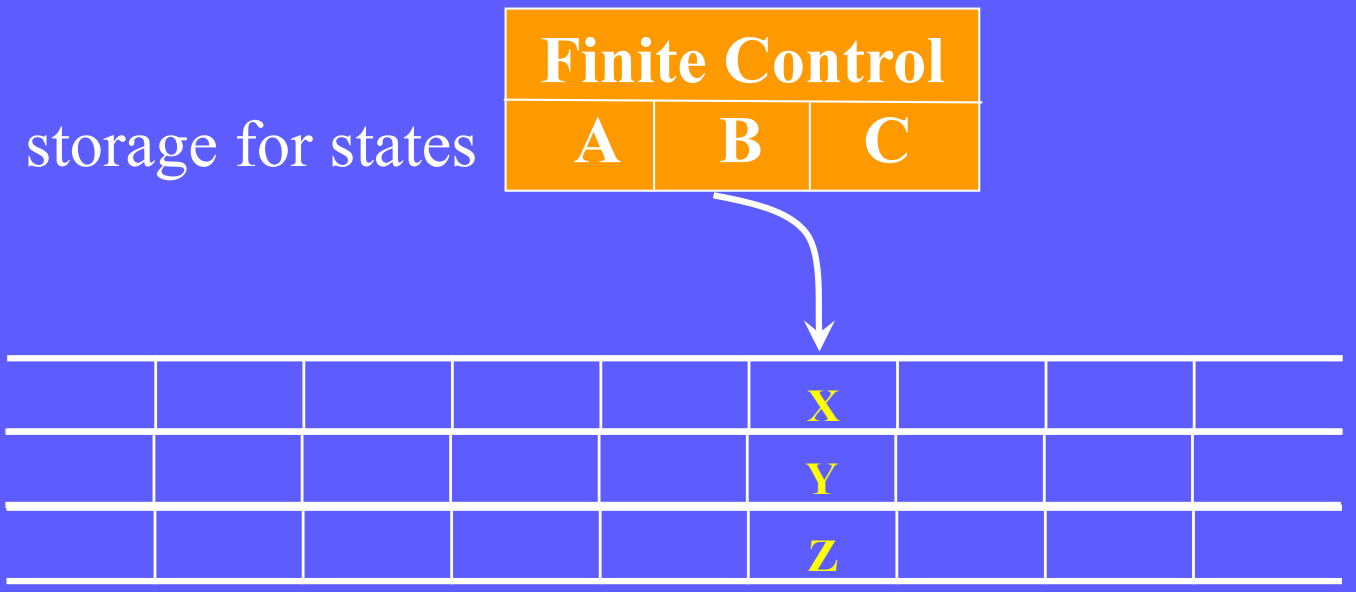
Example of Turing Machine (cont.)

- $\delta(q_0, 0)=(q_1, \_, R)$
- $\delta(q_1, 0)=(q_1, 0, R)$, $\delta(q_1, 1)=(q_2, 1, R)$
- $\delta(q_2, 1)=(q_2, 1, R)$, $\delta(q_2, 0)=(q_3, 1, L)$
- $\delta(q_3, 0)=(q_3, 0, L)$, $\delta(q_3, 1)=(q_3, 1, L)$, $\delta(q_3, \_)=(q_0, \_, R)$
- $\delta(q_2, \_)=(q_4, \_, L)$, $\delta(q_4, 1)=(q_4, \_, L)$, $\delta(q_4, 0)=(q_4, 0, L)$, $\delta(q_4, \_)=(q_6, 0, R)$
- $\delta(q_0, 1)=(q_5, \_, R)$, $\delta(q_5, 0)=(q_5, \_, R)$, $\delta(q_5, 1)=(q_5, \_, R)$, $\delta(q_5, \_)=(q_6, \_, R)$
- Example input 0010, output: 0



Extensions of the Basic Turing Machine

- **Multiple track Turing machine**
- **Multiple tape Turing machine**
- **Non-deterministic Turing machine**



Non-deterministic Turing Machine

- **A <u>non-deterministic</u> Turing machine**
  - $\delta(q, X)=\{ (q_1,Y_1, D_1), (q_2,Y_2, D_2),\ldots, (q_k,Y_k, D_k),\}$
  - **has the power to be in several states at once**
  - **an ability to "guess" something about the input**

*** NP: Non-deterministic Polynomial**

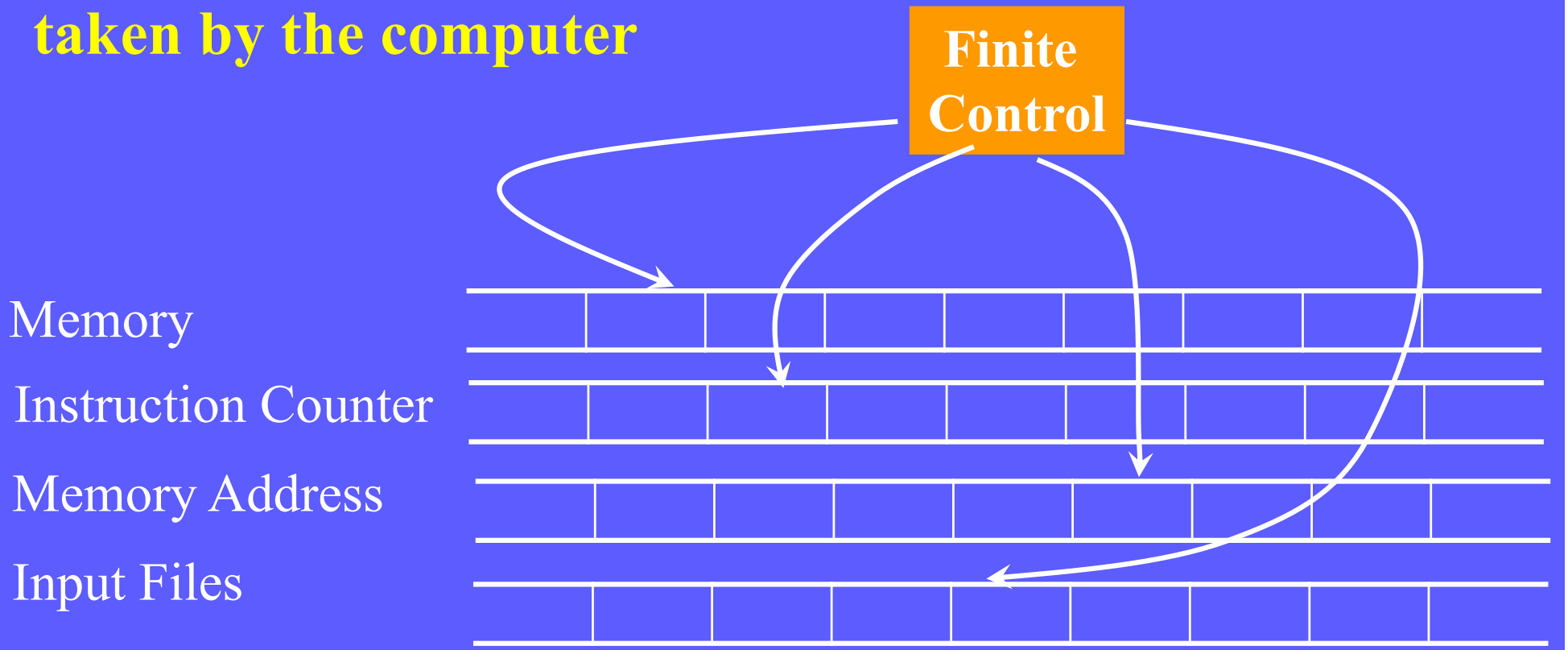
Is there any problem that Turing Machine never halt ?

Undecidable Problem

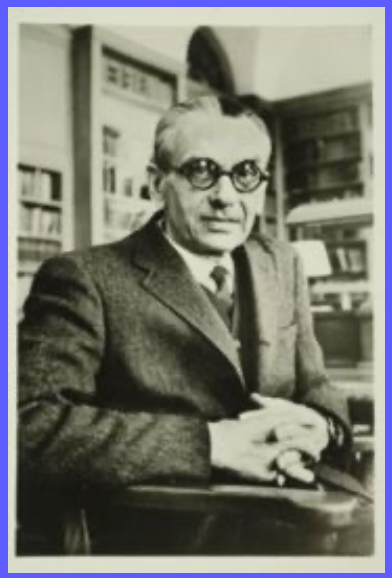
- **Undecidable problem**
  - **a problem that cannot be solved by computer**
  - **a problem that Turing machine may run forever (never halts)**
- **Is there any undecidable problem ?**
  - **yes**
  - **proof by contradiction**

Turing Machine & Computers

- **A computer can simulate a Turing machine**
- **A Turing machine can simulate a computer & can do so in at most polynomial steps taken by the computer**



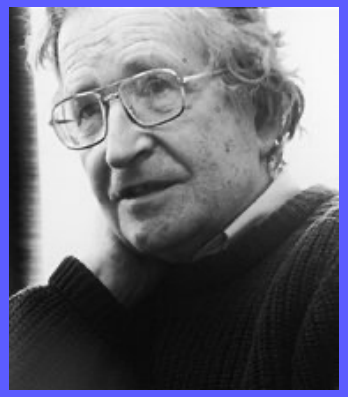
NP-Completeness



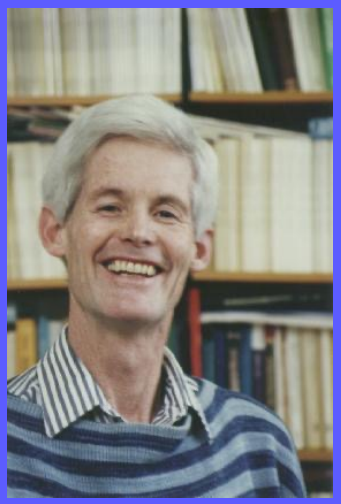
Kurt Gödel
Apr. 28, 1906

Alonzo Church
Jun. 14, 1903

Alan Turing
Jun. 23, 1912



Noam Chomsky
Dec. 7, 1928



Stephen Cook
1939

Leonid Levin
Nov. 2, 1948

1982 ACM Turing Award Recipients: Stephen A. Cook

For his advancement of our understanding of the *complexity of computation* in a significant and profound way.

His seminal paper,

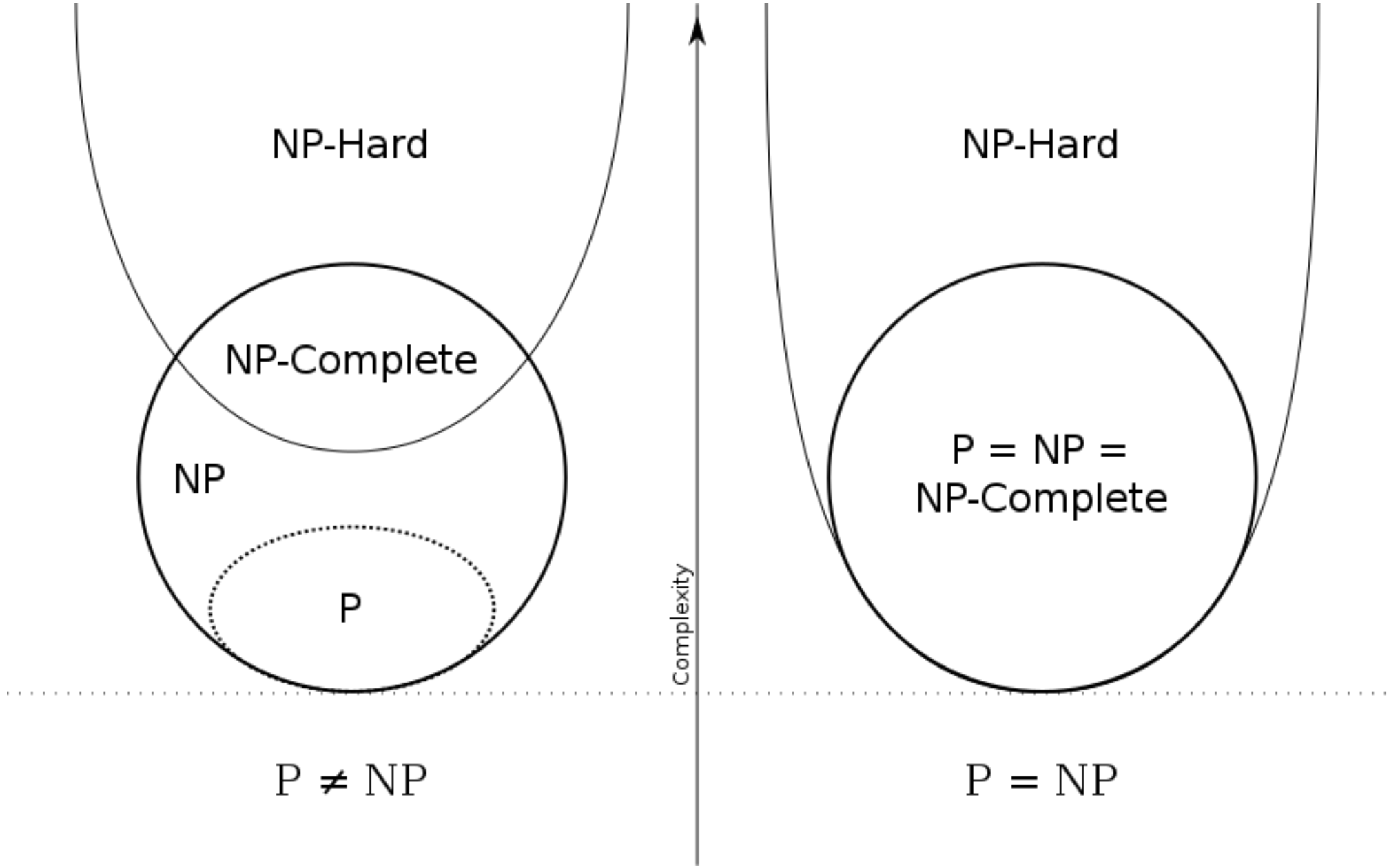
"The Complexity of Theorem Proving Procedures," presented at the 1971 ACM SIGACT Symposium on the Theory of Computing, laid the foundations for the theory of *NP-Completeness*.

The ensuing exploration of

the boundaries and nature of NP-complete class of problems

has been one of the most active and important research activities

in computer science for the last decade.


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP

Decision Problem
vs.
Optimization Problem

Decision Problems

- **Two categories of problems**
  - **optimization problems: maximal matching**
  - **decision problems: matching of size >= *k***

  ***e.g.* the minimum spanning tree problem**
    - optimization problem: given a graph *G*, find the spanning tree with minimum total edge weight
    - decision problem: given a graph *G* and a integer *W*, does G have a spanning tree of weight *W* or less

  ***e.g.* the weighted interval scheduling problem**
    - optimization problem: given a set of intervals *S*, find the maximum sized subset of non-overlapping intervals
    - decision problem: given a set of intervals *S* and a integer *W*, does *S* have a subset of non-overlapping intervals such that the subset size is *W* or more.

Weighted Interval Scheduling Problem

- optimization problem

  given a set of intervals *S*,

  find the maximum sized subset of

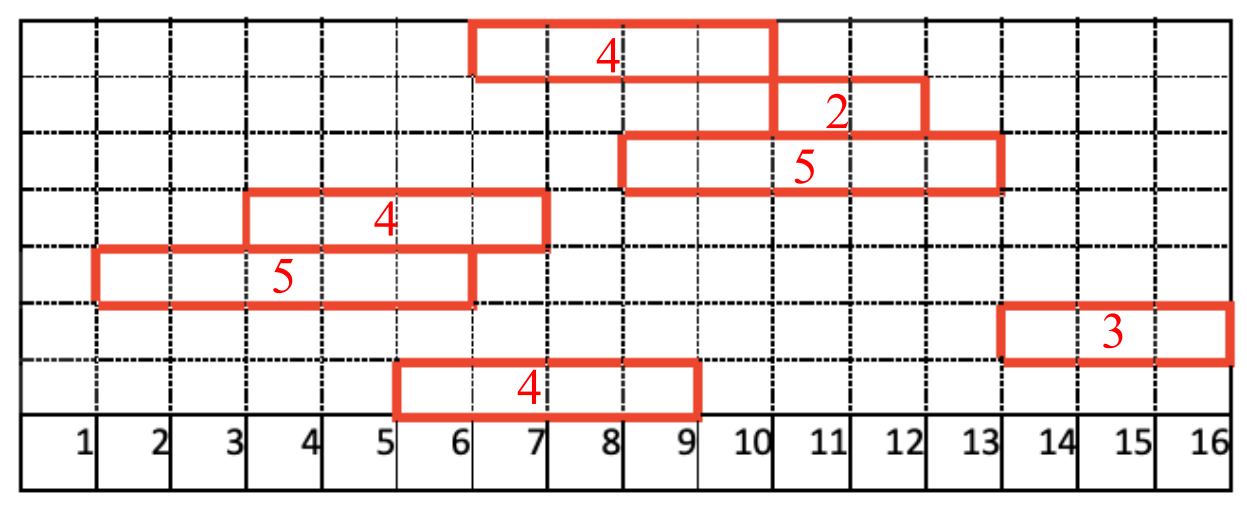
  non-overlapping intervals

- decision problem

  given a set of intervals *S* & a integer *W*,

  does *S* have a subset of non-overlapping intervals

  such that the subset size is W or more.



Decision Problems (cont.)

- **Optimization problem is harder than decision problem**
- **In discussing NP problems,**

  we shall discuss decision problems

Decision Problems (cont.)

- **Decision problems can be viewed as**

  <u>language recognition problem</u>

  - **let U be the set of all possible inputs to the decision problem**
  - **let $L \subseteq U$ be the set of all inputs for which the answer to the problem is yes**
  - **L is the language corresponding to the problem**
  - **decision problem is to recognize whether or not a given input belongs to L**

  e.g. the weighted interval scheduling

  given 3 intervals [2, 5], [3, 8], [7, 12] and W=8

  encode as 2, 5, 3, 8, 7, 10 ; 8 Yes

  encode as 2, 5, 3, 8, 7, 10 ; 20 No

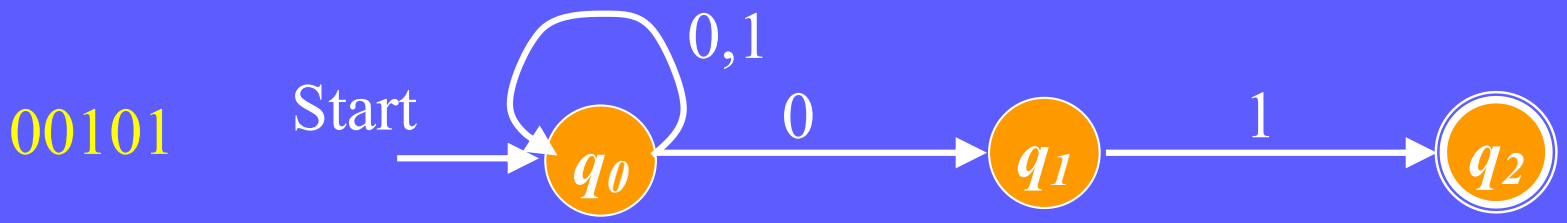
Deterministic vs. Non-Deterministic Algorithm

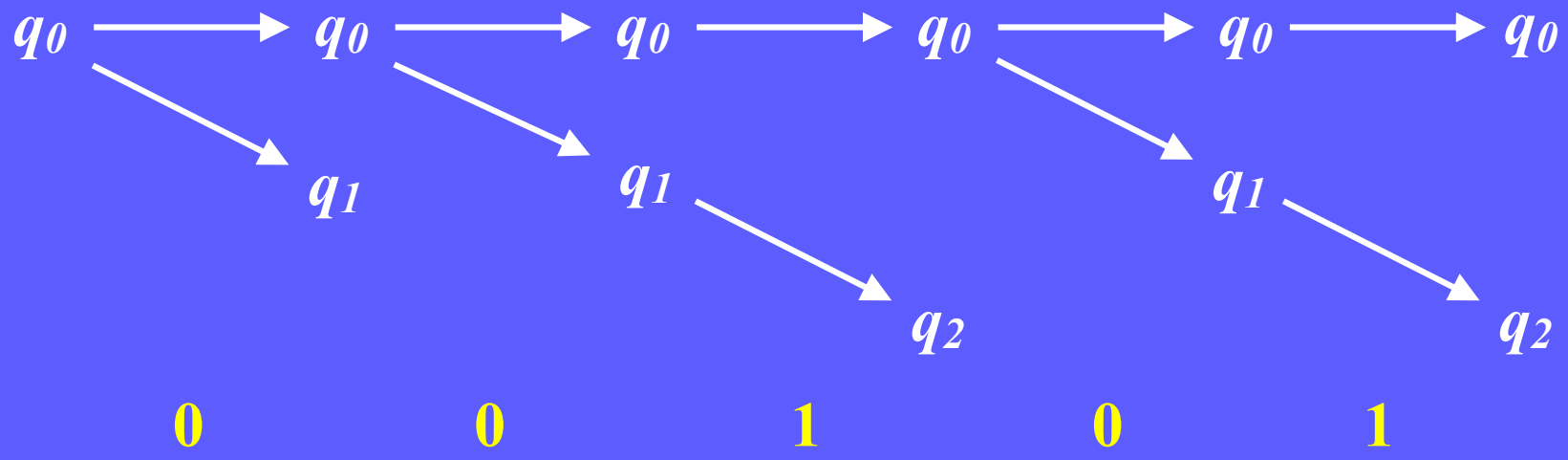
Non-Deterministic Algorithms

- **Model of algorithms**
  - **Turing machine(Cook's theorem)**
  - **Random access machine**
- **Deterministic algorithm**
- **Non-deterministic algorithm**
  - **two phase: guessing & checking**
  - **it is assumed that a non-deterministic algorithm always make a good guessing**
  - **actually, non-deterministic algorithm do not exist and they would never exist**
  - **not all problems can be solved efficiently by a non-deterministic algorithm, e.g. undecidable problems**

Nondeterministic Finite Automata (cont.)

- **Nondeterministic FSM** $A = (Q, \Sigma, \delta, q_0, F)$
  - **has the power to be in several states at once**
  - **an ability to "guess" something about the input**



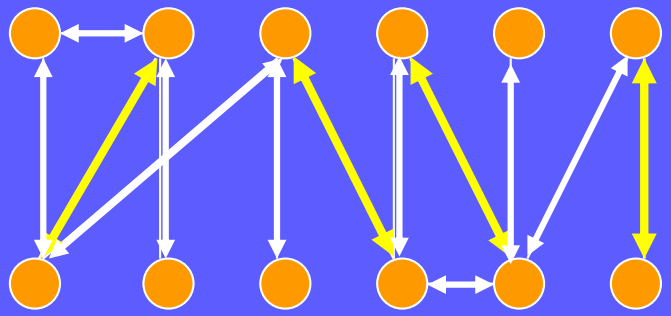


Non-deterministic Perfect Matching

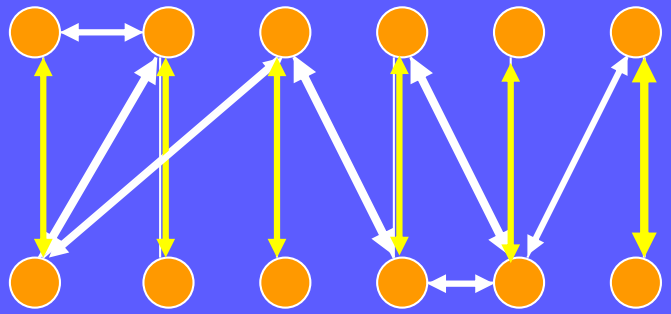
- **Decision Problem: whether a graph G=(V, E) has a perfect matching**
- **Algorithm**
  1. **M={ }**
  2. **For each edge *e***

     non-deterministic choice *e* to be include in M
  3. **Check whether M is perfect matching (| V | /2 edges)**

Perfect Matching



Check
Not Perfect Matching



Check
Perfect Matching !

Non-deterministic Bubble Sort

```
for i = 1 to n
    for j = 1 to n – 1
        if A[j] < A[i+1] then
            Either exchange A[j] and A[i+1] or do nothing
```

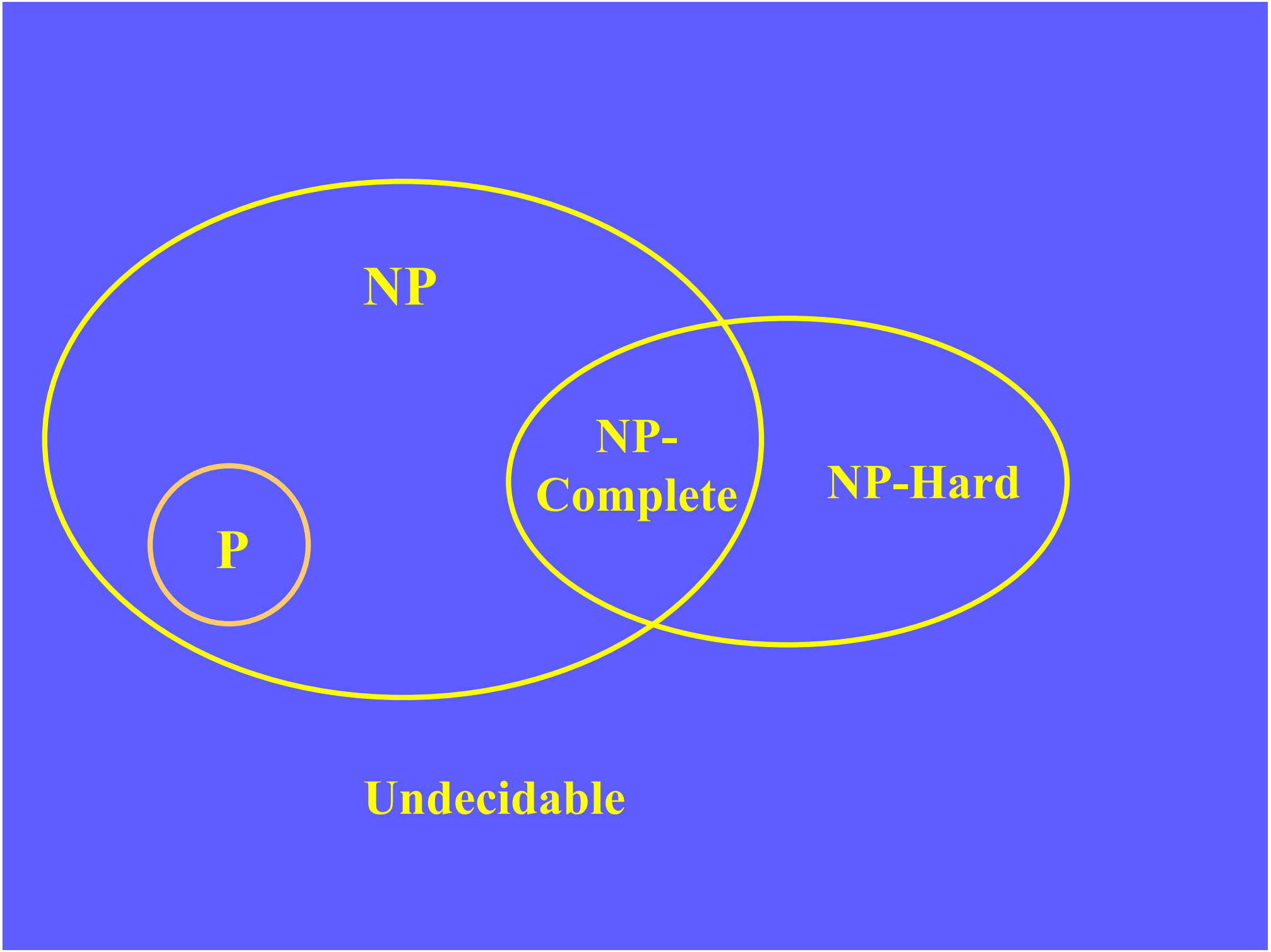
Non-deterministic Algorithm

- **Non-deterministic algorithm is very powerful, but the power is not unlimited.**
- **Not all problems can be solved efficiently by non-deterministic algorithms**

  e.g. To determine whether the maximal matching of a given graph is of size *k*

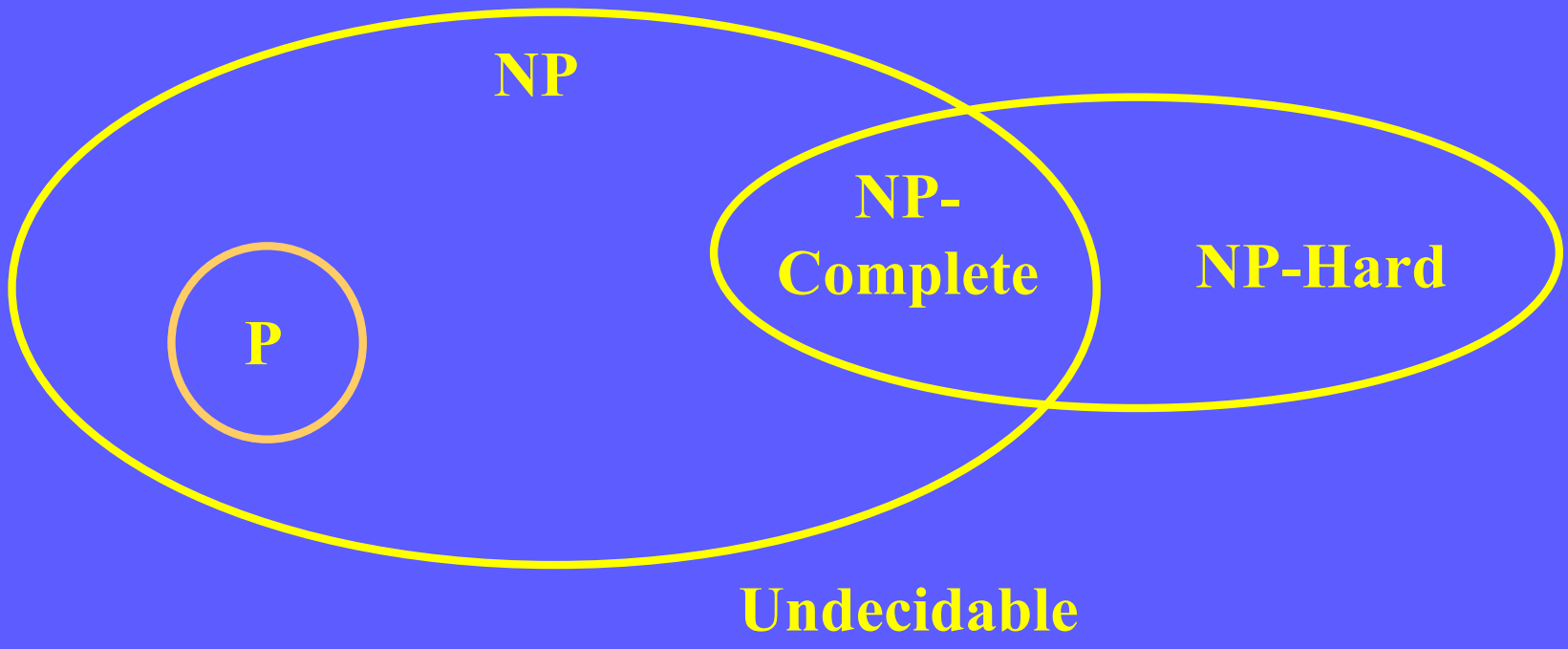
  - **we can use non-deterministic algorithm to find a matching of size *k* if it exists**
  - **we can not easily determine there is no matching of a larger size**

P, NP, NP Hard, NP Complete Problems


NP
NP-
Complete
NP-Hard
P
Undecidable

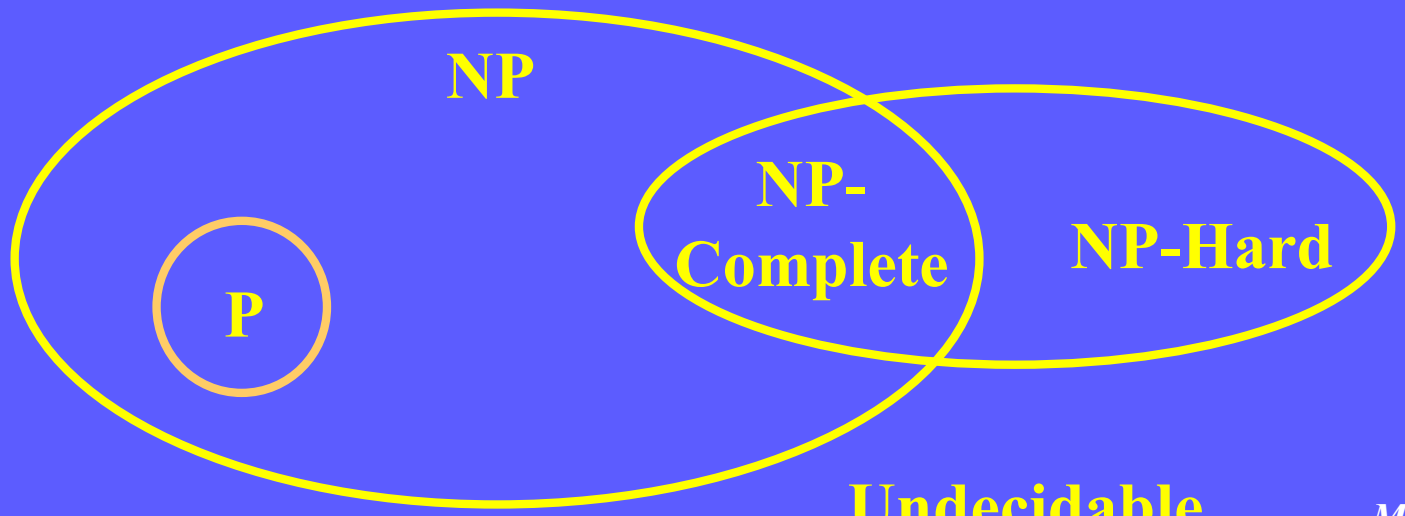
P Problems

- **Deterministic algorithm**
- **P Problem**
  - **if a problem can be solved by a deterministic algorithm in polynomial time.**



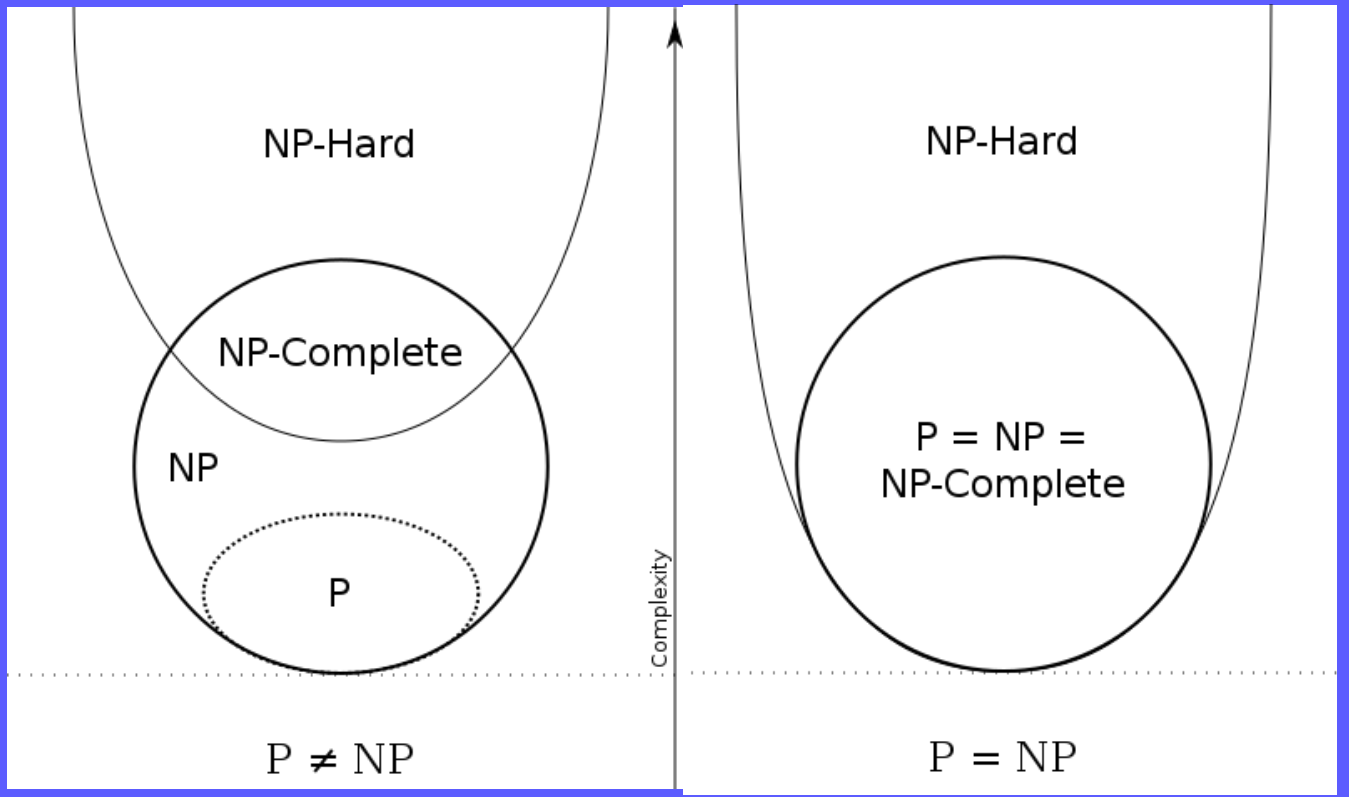
NP Problems

- Non-deterministic polynomial algorithm
  - if the checking stage is of polynomial time complexity
- NP problems (Non-deterministic Polynomial)
  - if a decision problem can be solved by a non-deterministic polynomial algorithm
- Every problem which can be solved in polynomial time by deterministic algorithms (P) must be NP problems



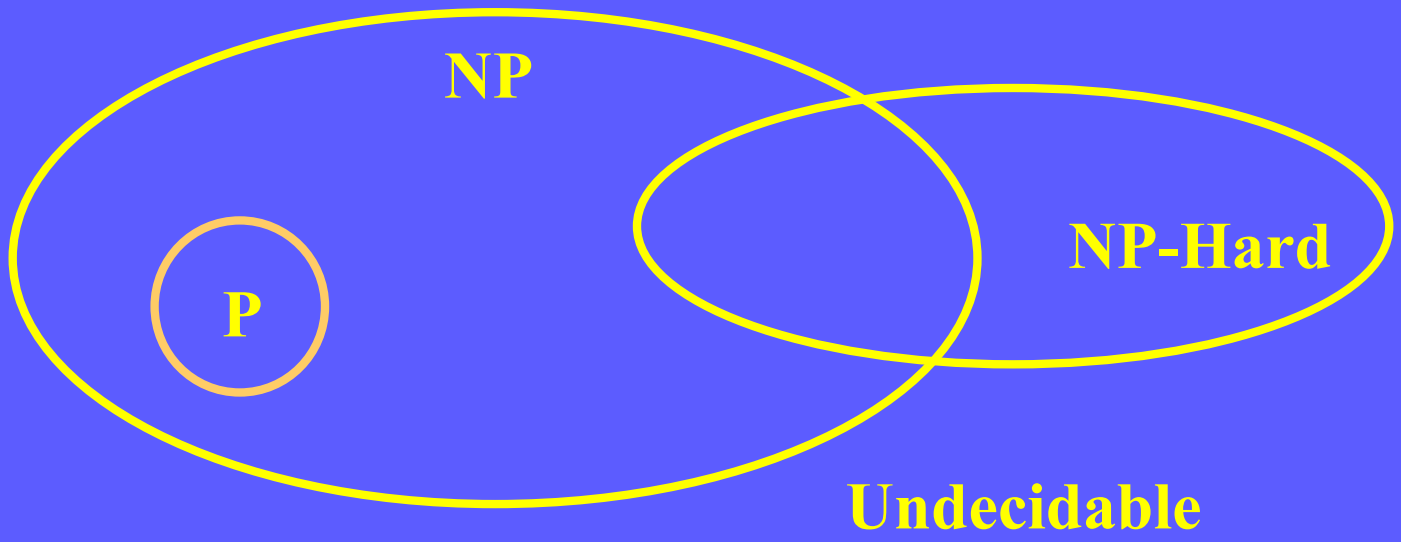
NP Problems (cont.)

- **Until now, nobody has been able to**
  - **exhibit an NP problem that is not in P (P != NP)**
  - **prove that every problem that belongs to NP can be solved by a polynomial-time deterministic algorithm (P = NP)**
  - **P = NP ? problem**



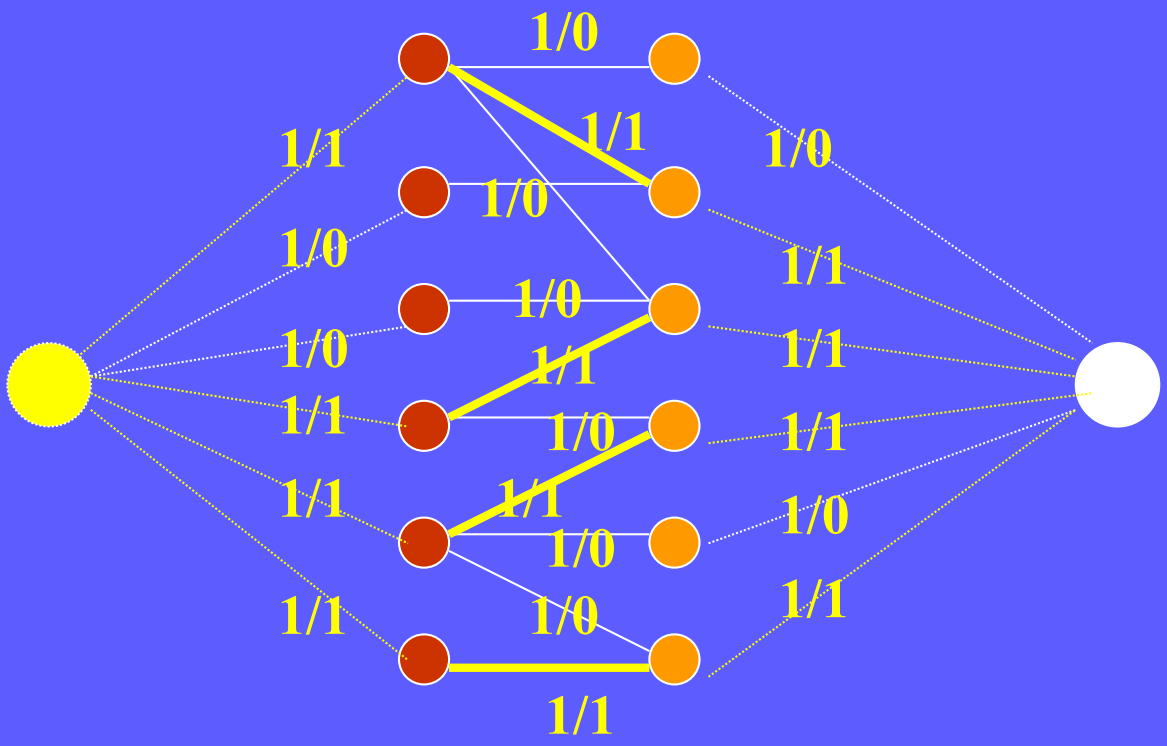
NP-Hard Problems

- **NP-Hard problems**
  - **A problem X is an NP-hard problem if every problem in NP is polynomial reducible to X**



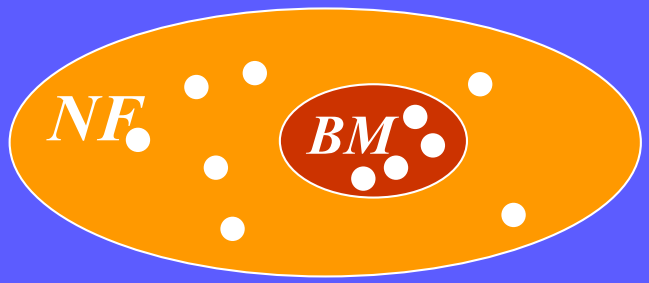
Reducing Bipartite Matching to Network Flow

- **M is a maximum matching**
  iff the corresponding flow is a maximum flow in G'

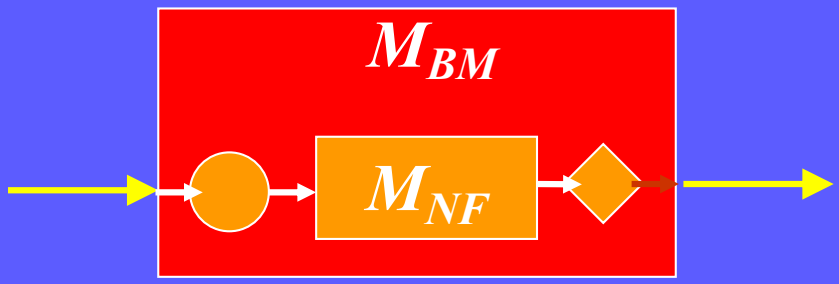


Reductions

- **Reductions**
  - **Bipartite Matching is a special case of Network Flow**
  - **A solution of problem *Bipartite Matching* using a black box that solves Network Flow**
  - **Goals**
    - **If *Bipartite Matching* is known to be a hard problem (or if we know its lower bound), then the same lower bound may be applied to *Network Flow***
    - **Solving *Bipartite Matching* cannot be harder than solving *Network Flow***



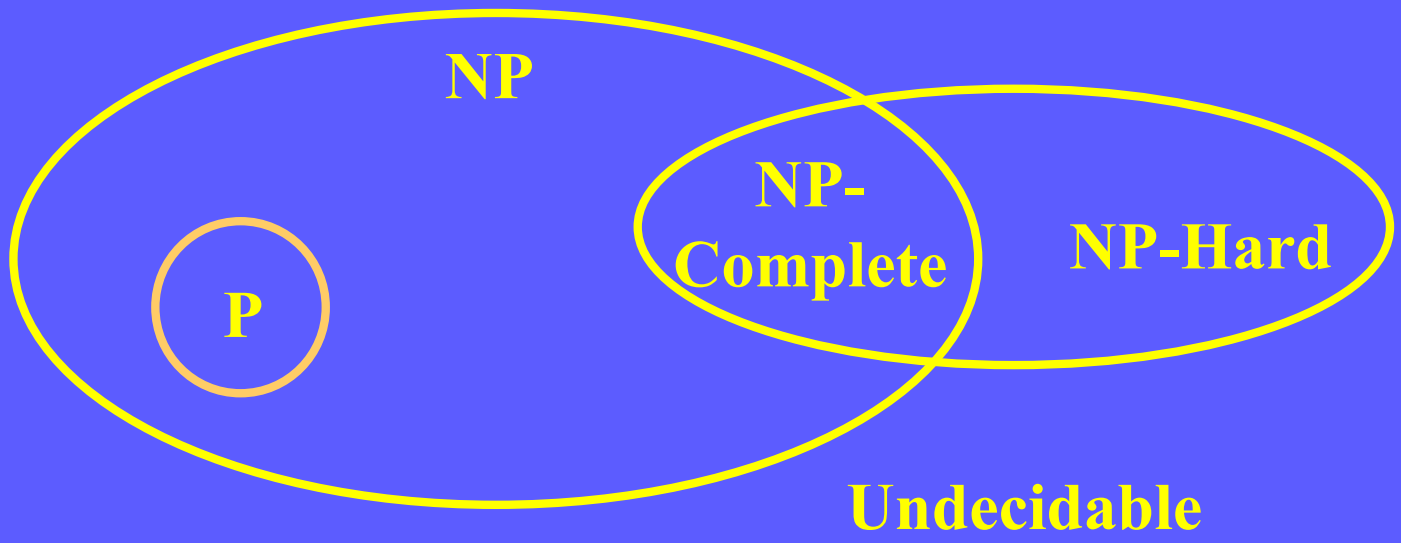
Problem Instances



Algorithm

NP-Complete Problems

- **NP-Complete problems**
  - **A problem X is called an NP-complete problem**
    if (1) X belongs to NP
    (2) X is NP-Hard
  - **An optimization problem is NP-hard if its corresponding decision problem is NP-complete**



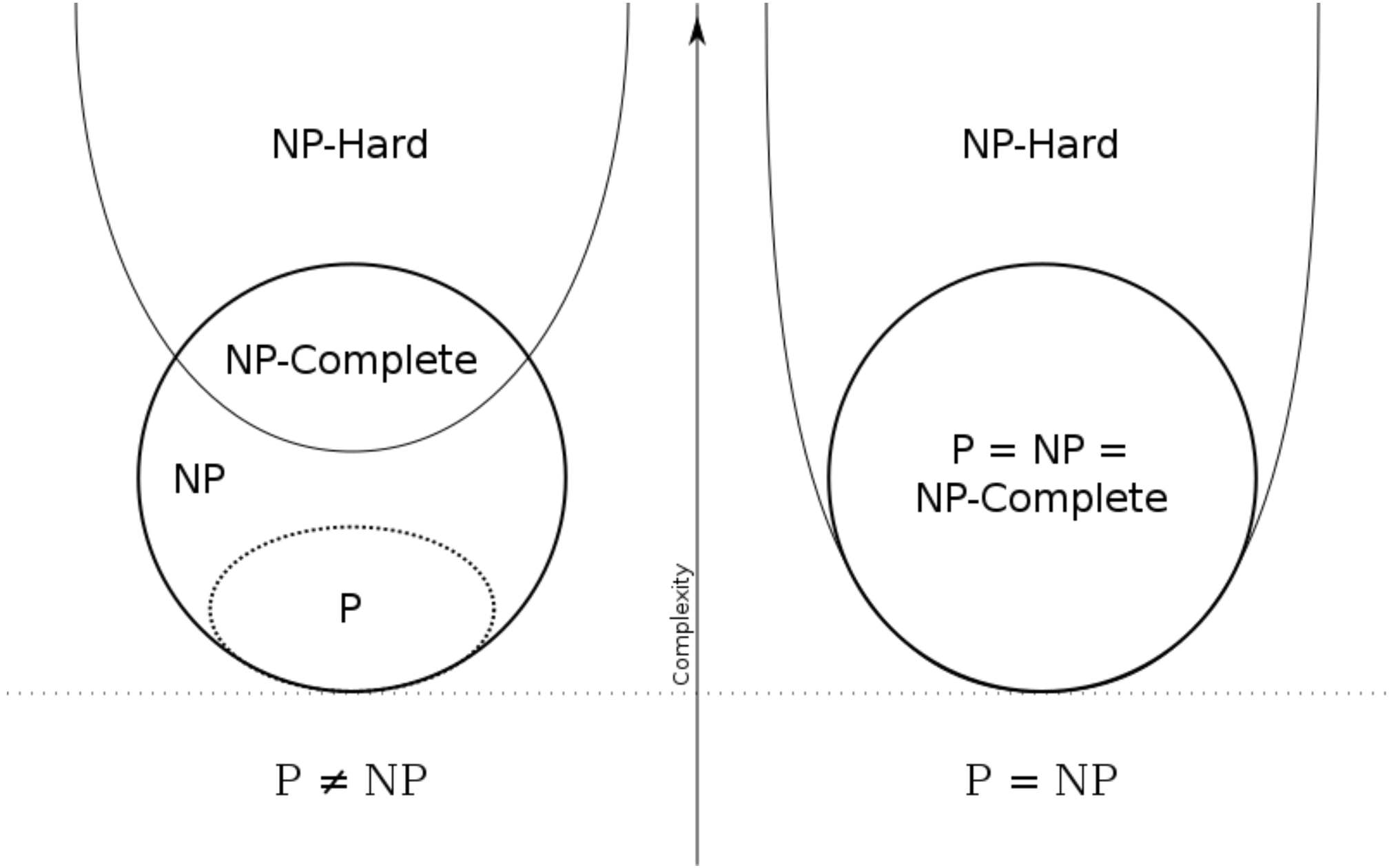
Undecidable Problems

- **Undecidable**
  - **not all problems can be solved efficiently by a non-deterministic algorithm**
  - **undeciable problems are decision problems, they cannot be solved by exhaustively examining the solution space**
  - **no upper bound for time complexity of undecidable problem**

  e.g.

  *** halting problem: given an arbitrary program with an arbitrary input data, will the program terminate or not?**

  *** first-order predicate calculus satisfibility**
    - **propositional logic: p ^ q -> s**
    - **first order logic**
      - **every man is mortal, Confucius is a man, Confucius is mortal**
      - **∀ x (p(x) -> q(x))**


NP-Hard
NP-Complete
NP
P
Complexity
NP-Hard
P = NP =
NP-Complete
P ≠ NP
P = NP

Informal Discussions of Theory of NP-Completeness

Informal Discussions of Theory of NP-Completeness

- **Theory of NP-Completeness identifies a large class of difficult problems**
- **Difficult problem: a problem whose lower bound seems to be in the order of an exponential function**
- **Not all NP problems are difficult: P**
- **Up to now, none of the NP-Complete problems can be solved by any polynomial algorithm in worst case**
- **Up to now, the best algorithm to solve any NP-Complete problem has exponential time complexity in worst case**

Informal Discussions of Theory of NP-Completeness (cont.)

- **If any NP-Complete problem can be solved in polynomial time, then all NP problems can be solved in polynomial time**
- **If any NP-Complete problem can be solved in polynomial time, then NP=P**
- **The theory of NP-Completeness has not claimed that NP-Complete problem can never be solved by polynomial algorithm**

Conclusions

- **Problem difficulty**
- **Computer Model: Turing Machine**
  - **Non-deterministic Turing Machine**
- **Problem**
  - **optimization problem**
  - **decision problem**
- **Deterministic vs. Non-deterministic Algorithm**
- **P, NP, NP-Hard, NP-Complete, Undecisible Problems**
- **P = NP ?**
- **Proof of NP-Complete by Reduction**