Computational Complexity

Lecture 2
in which we talk about
NP-completeness
(reductions, reductions)

Recap

Recap

- Languages in NP are of the form:

Recap

- Languages in NP are of the form:
  - $L = \{ x \mid \exists w, |w| < poly(|x|) \text{ s.t. } (x,w) \in L' \}$, where L' is in P

Recap

- Languages in NP are of the form:
  - L= { x | $\exists$w, |w| < poly(|x|) s.t. (x,w) $\in$ L′ }, where L′ is in P
- Today: Hardest problems in NP

Reductions

Reductions

- At the heart of today's complexity theory

Reductions

- At the heart of today's complexity theory
- $L_1 \leq L_2$ if problem of deciding L_1 "reduces to that of deciding" L_2

Reductions

- At the heart of today's complexity theory
- $L_1 \leq L_2$ if problem of deciding L_1 "reduces to that of deciding" L_2
  - if can decide L_2, can decide L_1

Turing and Many-One

Turing and Many-One

- Turing reduction:

Turing and Many-One

- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1

Turing and Many-One

- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1
  - M_{L1} may query O_{L2} many times (with different inputs)

Turing and Many-One

- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1
  - M_{L1} may query O_{L2} many times (with different inputs)
- Many-One:

Turing and Many-One

- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1
  - M_{L1} may query O_{L2} many times (with different inputs)
- Many-One:
  - M_{L1} can query O_{L2} only once, and must output what O_{L2} outputs

Turing and Many-One

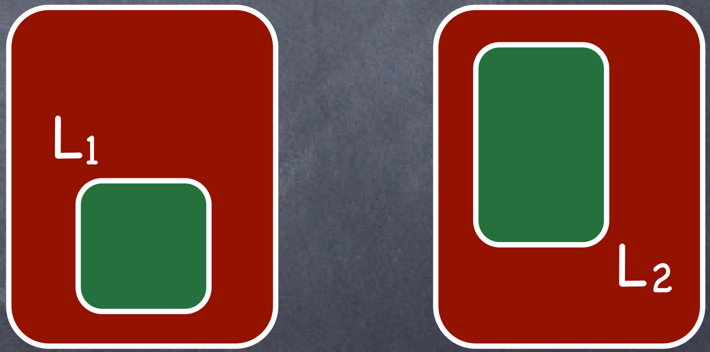
- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1
  - M_{L1} may query O_{L2} many times (with different inputs)
- Many-One:
  - M_{L1} can query O_{L2} only once, and must output what O_{L2} outputs
  - M_{L1} maps its input x to an input f(x) for O_{L2}

Turing and Many-One

- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1
  - M_{L1} may query O_{L2} many times (with different inputs)
- Many-One:
  - M_{L1} can query O_{L2} only once, and must output what O_{L2} outputs
  - M_{L1} maps its input x to an input f(x) for O_{L2}
    - $x \in L_1 \Rightarrow f(x) \in L_2$ and $x \notin L_1 \Rightarrow f(x) \notin L_2$

Turing and Many-One

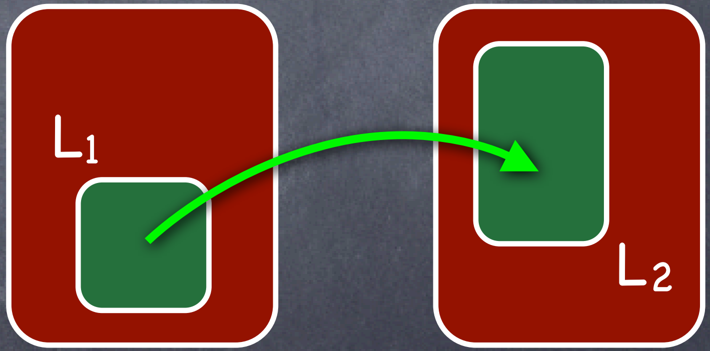
- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1
  - M_{L1} may query O_{L2} many times (with different inputs)
- Many-One:
  - M_{L1} can query O_{L2} only once, and must output what O_{L2} outputs



  - M_{L1} maps its input x to an input f(x) for O_{L2}
    - $x \in L_1 \Rightarrow f(x) \in L_2$ and $x \notin L_1 \Rightarrow f(x) \notin L_2$

Turing and Many-One

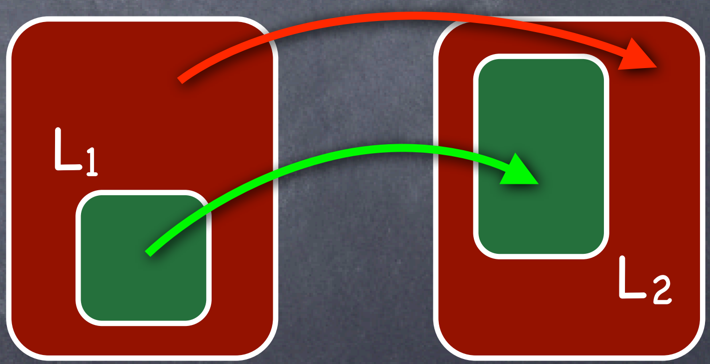
- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1
  - M_{L1} may query O_{L2} many times (with different inputs)
- Many-One:
  - M_{L1} can query O_{L2} only once, and must output what O_{L2} outputs



  - M_{L1} maps its input x to an input f(x) for O_{L2}
    - $x \in L_1 \Rightarrow f(x) \in L_2$ and $x \notin L_1 \Rightarrow f(x) \notin L_2$

Turing and Many-One

- Turing reduction:
  - Build a TM (oracle machine) M_{L1}, s.t. using the oracle O_{L2} which decides L_2, M_{L1}^O_{L2} decides L_1
  - M_{L1} may query O_{L2} many times (with different inputs)
- Many-One:



  - M_{L1} can query O_{L2} only once, and must output what O_{L2} outputs
  - M_{L1} maps its input x to an input f(x) for O_{L2}
    - $x \in L_1 \Rightarrow f(x) \in L_2$ and $x \notin L_1 \Rightarrow f(x) \notin L_2$

Polynomial-Time Reduction

Polynomial-Time Reduction

- Many-one reduction, where M_{L1} runs in polynomial time

Polynomial-Time Reduction

- Many-one reduction, where M_{L1} runs in polynomial time
- $L_1 \leq_p L_2$

Polynomial-Time Reduction

- Many-one reduction, where M_{L1} runs in polynomial time
- $L_1 \leq_p L_2$
- L_2 is "computationally (almost) as hard or harder" compared to L_1

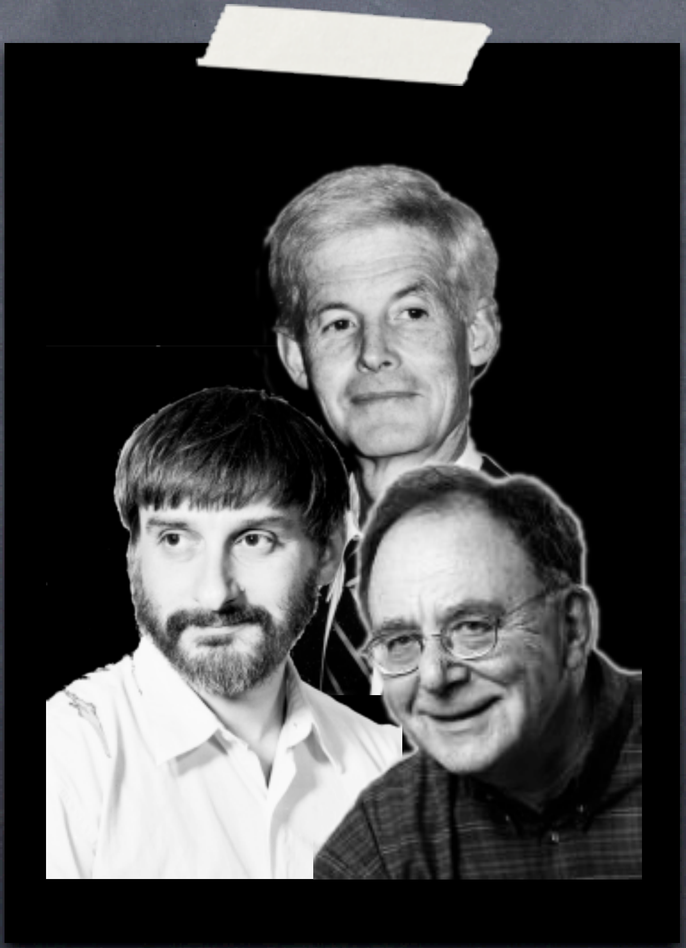
Polynomial-Time Reduction

- Many-one reduction, where M_{L1} runs in polynomial time
- $L_1 \leq_p L_2$
- L_2 is "computationally (almost) as hard or harder" compared to L_1
  - "almost": reduction overheads (reduction time, size blow-up)

Polynomial-Time Reduction

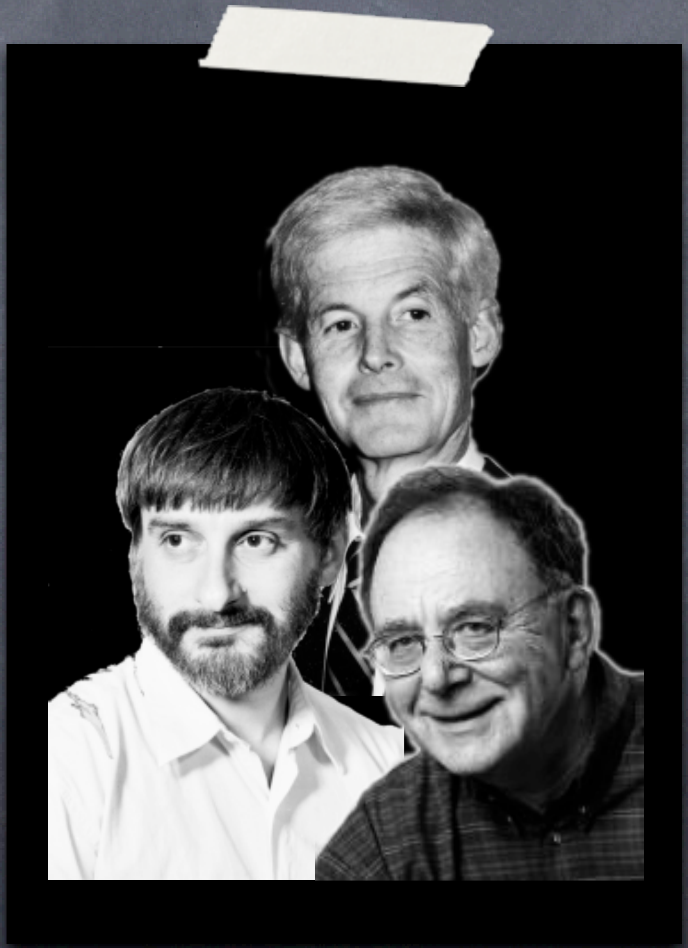
- Many-one reduction, where M_{L1} runs in polynomial time
- $L_1 \leq_p L_2$
- L_2 is "computationally (almost) as hard or harder" compared to L_1
  - "almost": reduction overheads (reduction time, size blow-up)
  - L_2 may be way harder

Cook, Karp, Levin



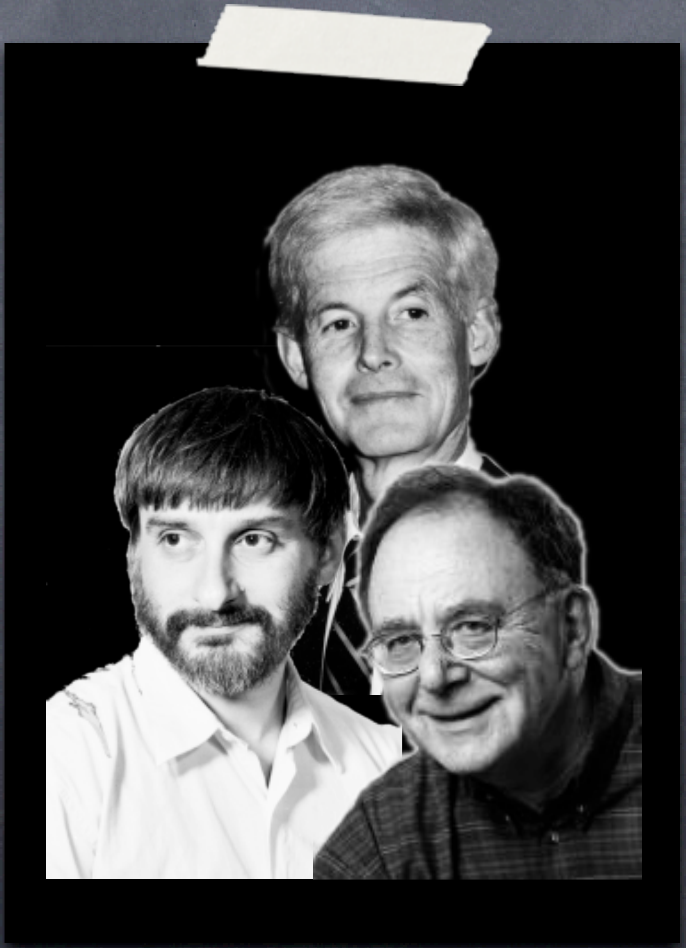
Cook, Karp, Levin

- Polynomial-time reduction



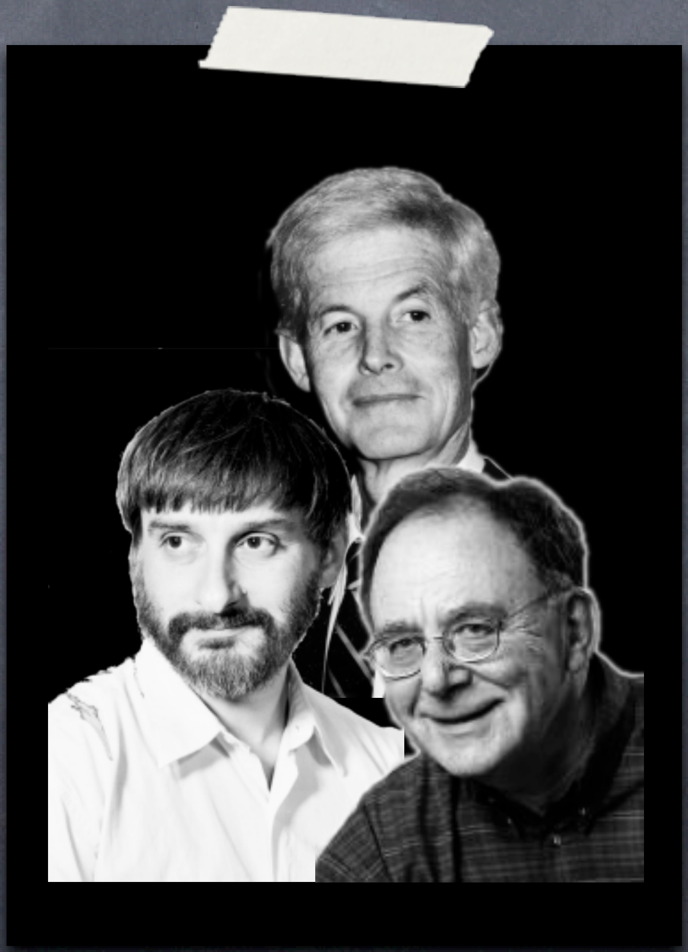
Cook, Karp, Levin

- Polynomial-time reduction
  - Cook: Turing reduction



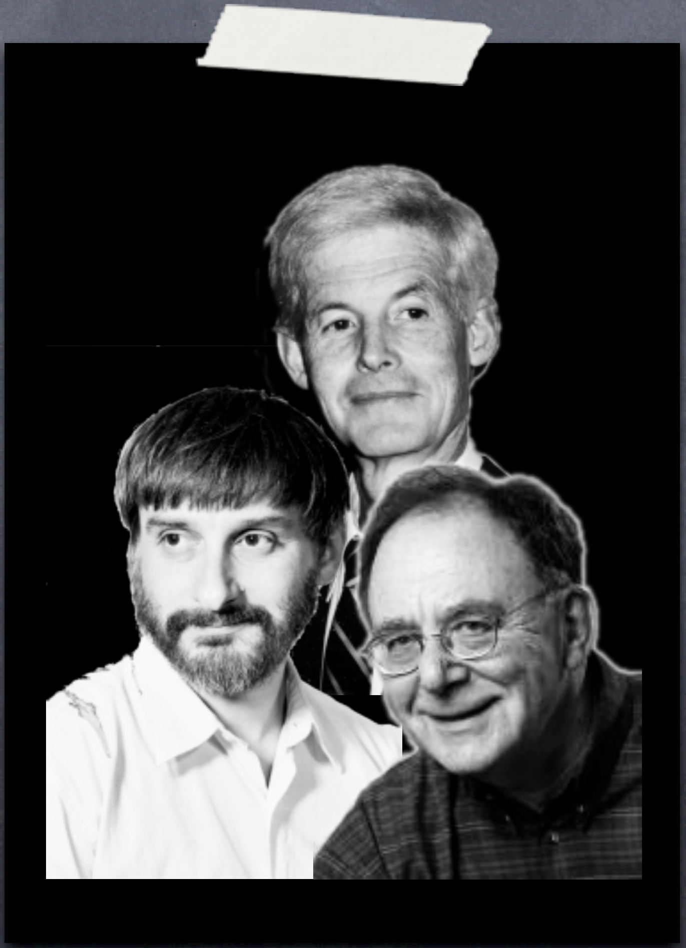
Cook, Karp, Levin

- Polynomial-time reduction
  - Cook: Turing reduction
  - Karp: Many-one reduction



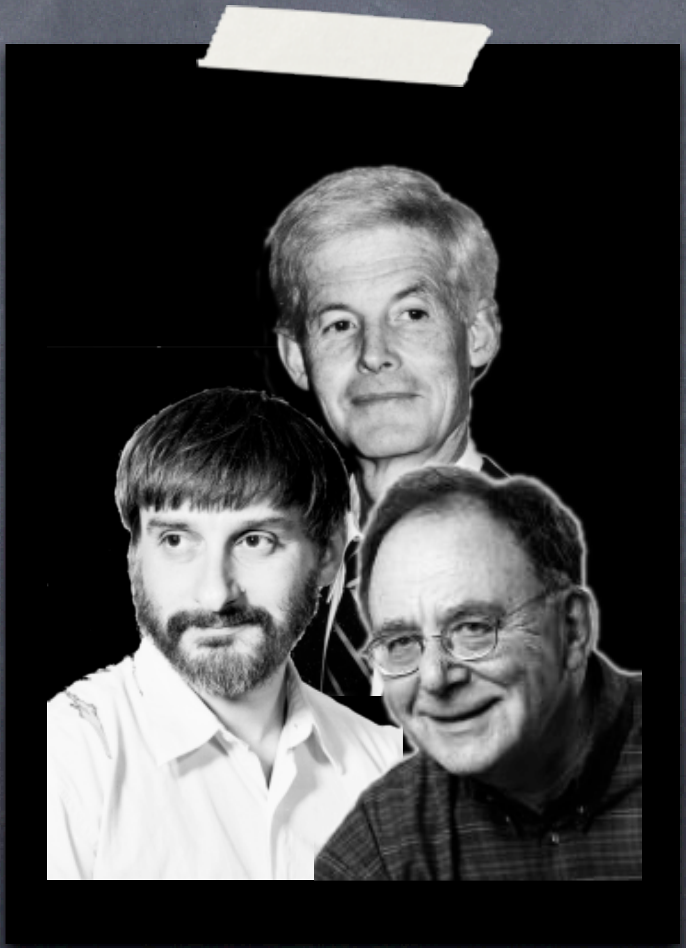
Cook, Karp, Levin

- Polynomial-time reduction
  - Cook: Turing reduction
  - Karp: Many-one reduction
    - We use this for $\leq_p$



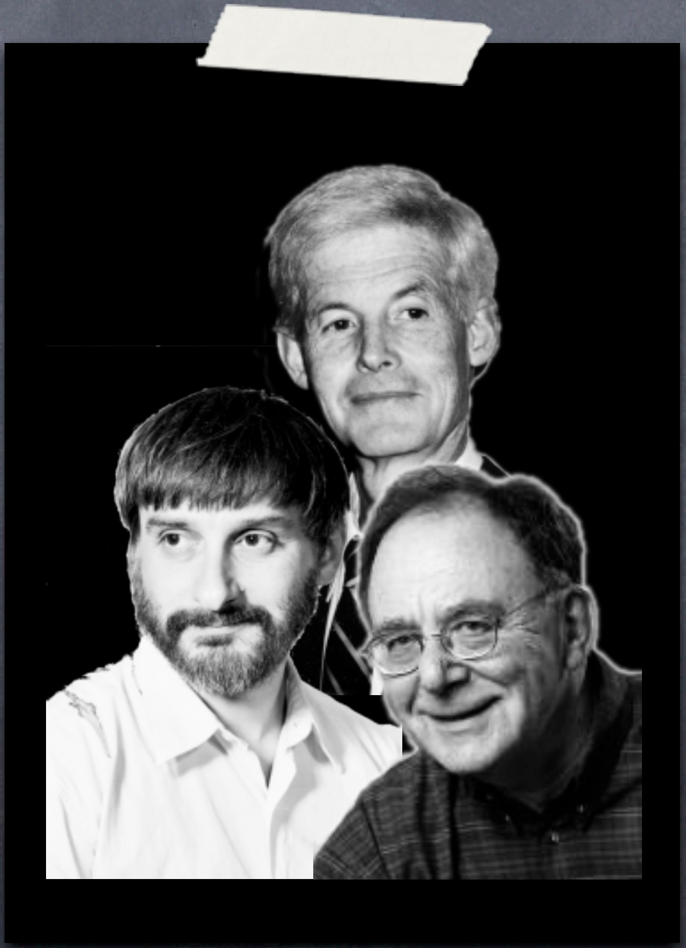
Cook, Karp, Levin

- Polynomial-time reduction
  - Cook: Turing reduction
  - Karp: Many-one reduction
    - We use this for $\leq_p$
- Between NP languages



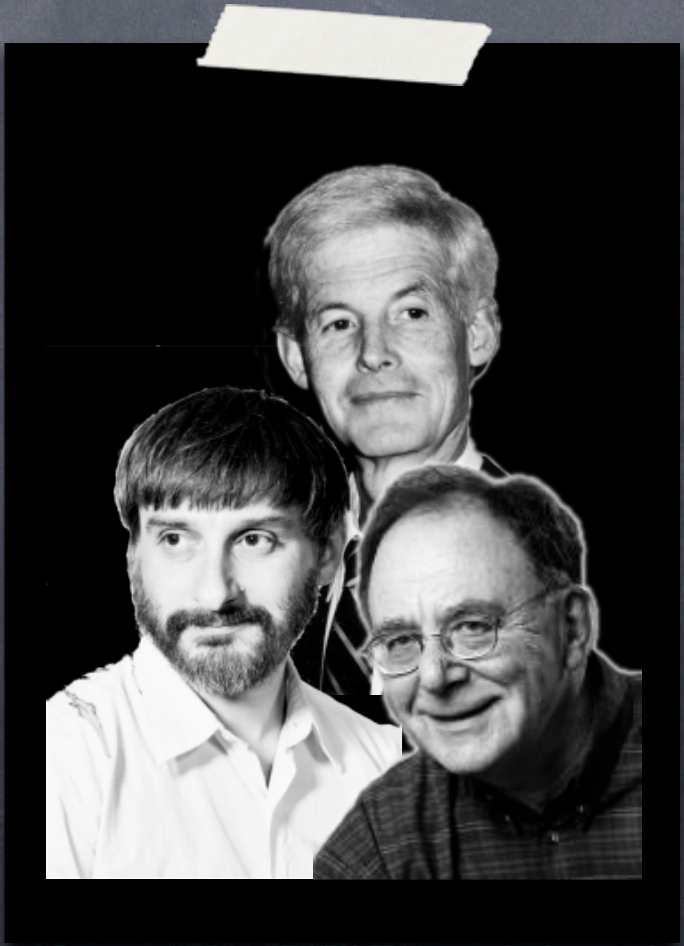
Cook, Karp, Levin

- Polynomial-time reduction
  - Cook: Turing reduction
  - Karp: Many-one reduction
    - We use this for $\leq_p$
- Between NP languages
  - Levin: Karp + witnesses easily transformed back and forth



Cook, Karp, Levin

- Polynomial-time reduction
  - Cook: Turing reduction
  - Karp: Many-one reduction
    - We use this for $\leq_p$
- Between NP languages
  - Levin: Karp + witnesses easily transformed back and forth
  - Parsimonious: Karp + number of witnesses doesn't change



NP-completeness

NP-completeness

- A language L is NP-Hard if for all L′ in NP, $L' \leq_p L$

NP-completeness

- A language L is NP-Hard if for all L' in NP, $L' \leq_p L$
- A language L is NP-Complete if it is NP-Hard and is in NP

NP-completeness

- A language L is NP-Hard if for all L′ in NP, $L' \leq_p L$
- A language L is NP-Complete if it is NP-Hard and is in NP
  - To efficiently solve all problems in NP, you need to efficiently solve L and nothing more

A simple NPC language

A simple NPC language

- TMSAT = { $(M,z,1^n,1^t)$ | $\exists w$, $|w|<n$, s.t. TM represented by M accepts (z,w) within time t }

A simple NPC language

- TMSAT = { $(M,z,1^n,1^t)$ | $\exists w$, $|w|<n$, s.t. TM represented by M accepts (z,w) within time t }
  - TMSAT is in NP: TMVAL = { $(M,z,1^n,1^t,w)$ | $|w|<n$ and M accepts (z,w) within time t } is in P

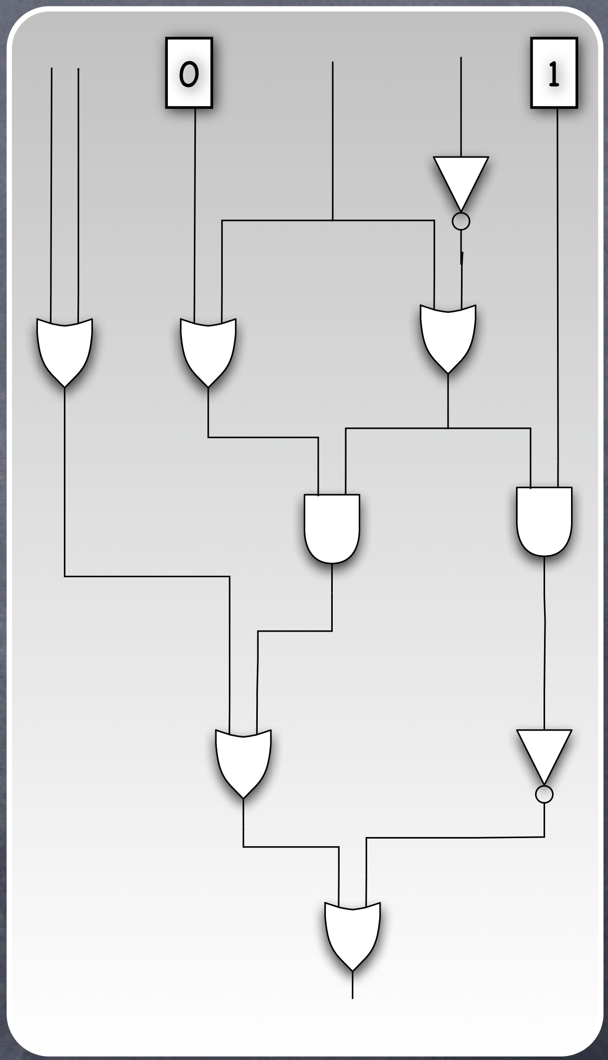
A simple NPC language

- TMSAT = { $(M,z,1^n,1^t)$ | ∃w, |w|<n, s.t. TM represented by M accepts (z,w) within time t }
  - TMSAT is in NP: TMVAL = { $(M,z,1^n,1^t,w)$ | |w|<n and M accepts (z,w) within time t } is in P
  - TMSAT is NP-hard: Given a language L in NP defined as L = { x | ∃w, |w|<n s.t. $M_{L'}$ accepts (x,w) } and $M_{L'}$ runs within time t, (where n,t are poly(|x|)), let the Karp reduction be $f(x) = (M_{L'},x,1^n,1^t)$

A simple NPC language

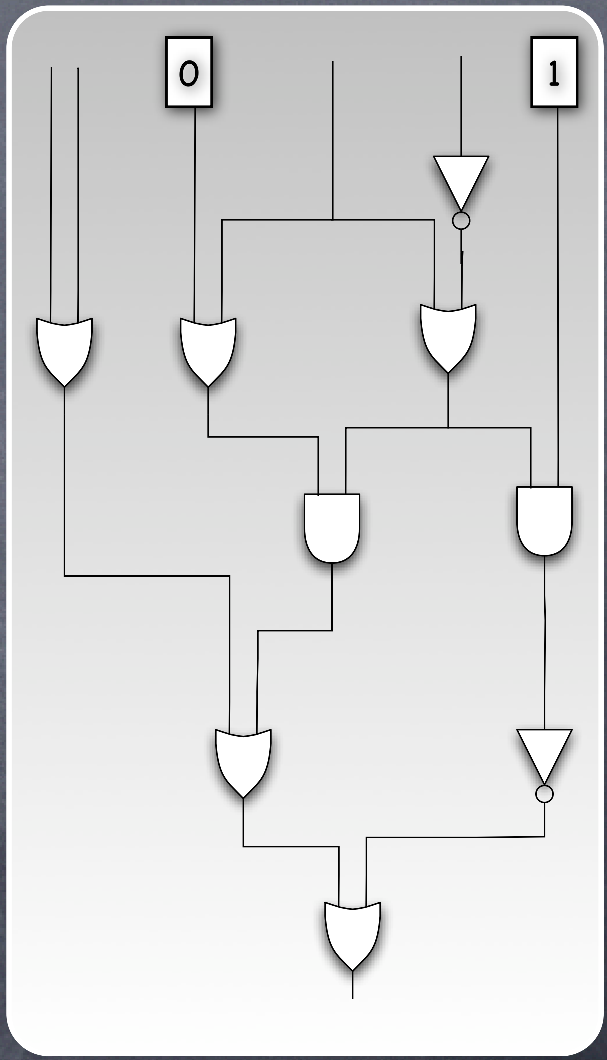
- TMSAT = { $(M,z,1^n,1^t)$ | $\exists w$, $|w|<n$, s.t. TM represented by M accepts (z,w) within time t }
  - TMSAT is in NP: TMVAL = { $(M,z,1^n,1^t,w)$ | $|w|<n$ and M accepts (z,w) within time t } is in P
  - TMSAT is NP-hard: Given a language L in NP defined as L = { x | $\exists w$, $|w|<n$ s.t. $M_{L'}$ accepts (x,w) } and $M_{L'}$ runs within time t, (where n,t are poly(|x|)), let the Karp reduction be $f(x) = (M_{L'},x,1^n,1^t)$
- Any "natural" NPC language?

Boolean Circuits



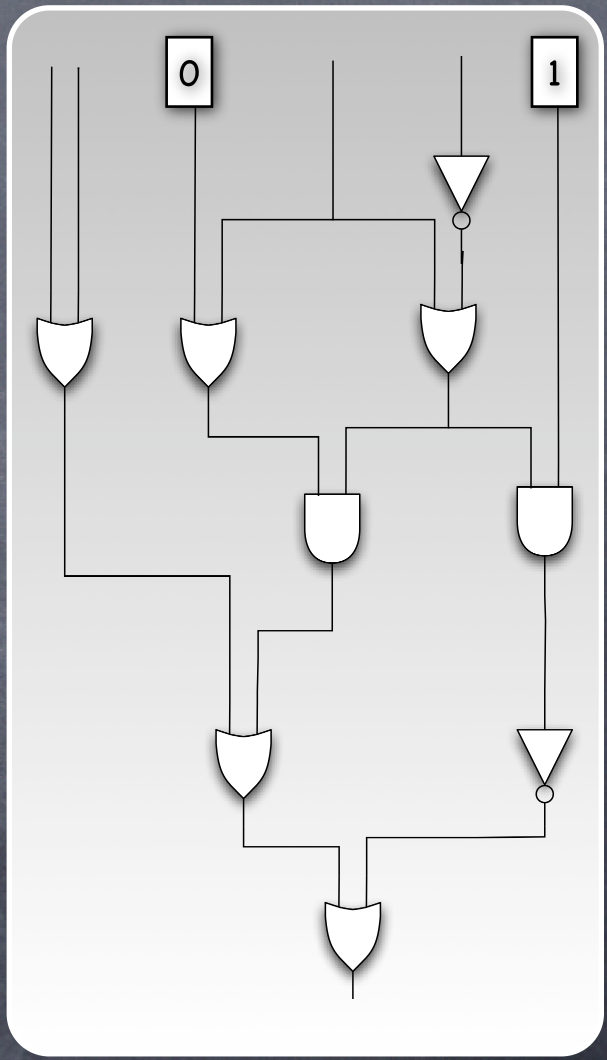
Boolean Circuits

- Boolean valued wires, AND, OR, NOT, CONST gates, inputs, output, directed acyclic graph



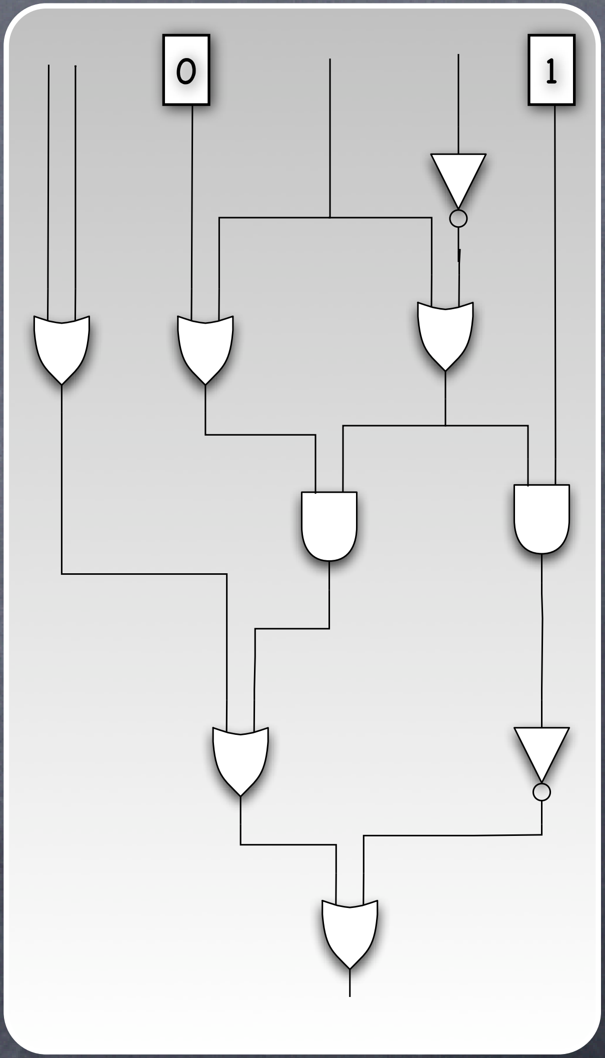
Boolean Circuits

- Boolean valued wires, AND, OR, NOT, CONST gates, inputs, output, directed acyclic graph
  - Circuit evaluation CKT-VAL: given (ckt,inputs) find ckt's boolean output value



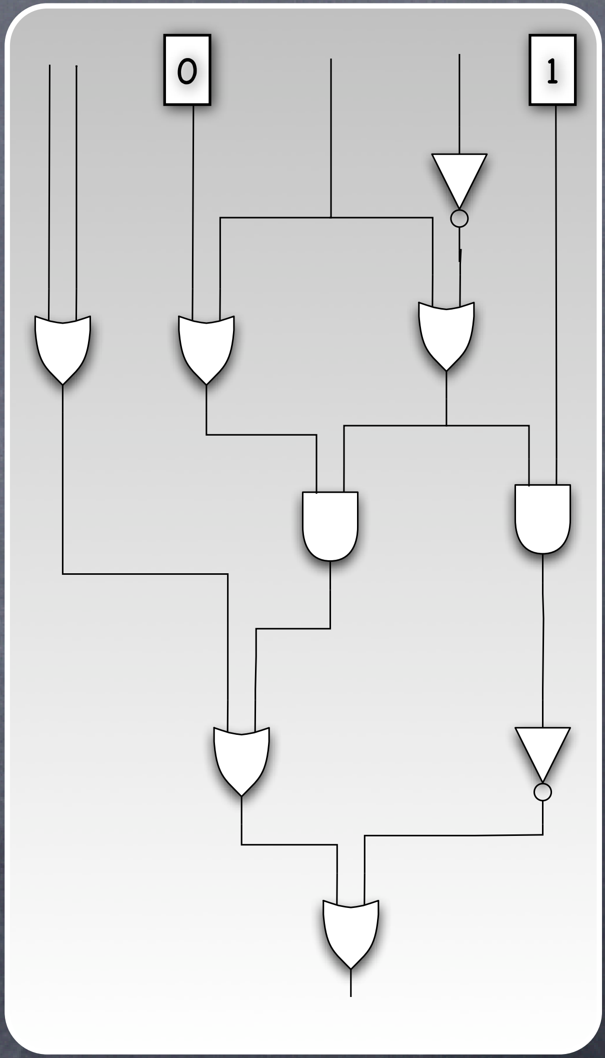
Boolean Circuits

- Boolean valued wires, AND, OR, NOT, CONST gates, inputs, output, directed acyclic graph
  - Circuit evaluation CKT-VAL: given (ckt,inputs) find ckt's boolean output value
  - Can be done very efficiently: CKT-VAL is in P



Boolean Circuits

- Boolean valued wires, AND, OR, NOT, CONST gates, inputs, output, directed acyclic graph
  - Circuit evaluation CKT-VAL: given (ckt,inputs) find ckt's boolean output value
  - Can be done very efficiently: CKT-VAL is in P
- CKT-SAT: given ckt, is there a "satisfying" input (output=1). In NP.



CKT-SAT is NP-Complete

CKT-SAT is NP-Complete

- Reduce any NP language L to CKT-SAT

CKT-SAT is NP-Complete

- Reduce any NP language L to CKT-SAT
  - Let's start from the TM for verifying membership in L, with time bound T

CKT-SAT is NP-Complete

- Reduce any NP language L to CKT-SAT
  - Let's start from the TM for verifying membership in L, with time bound T
  - Build a circuit which on input w outputs what the TM outputs on (x,w), within T steps

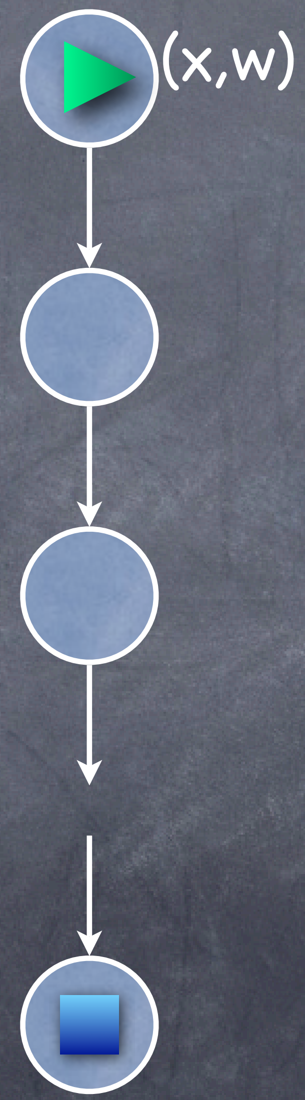
CKT-SAT is NP-Complete

- Reduce any NP language L to CKT-SAT
  - Let's start from the TM for verifying membership in L, with time bound T
  - Build a circuit which on input w outputs what the TM outputs on (x,w), within T steps
  - This circuit is an instance of CKT-SAT

CKT-SAT is NP-Complete

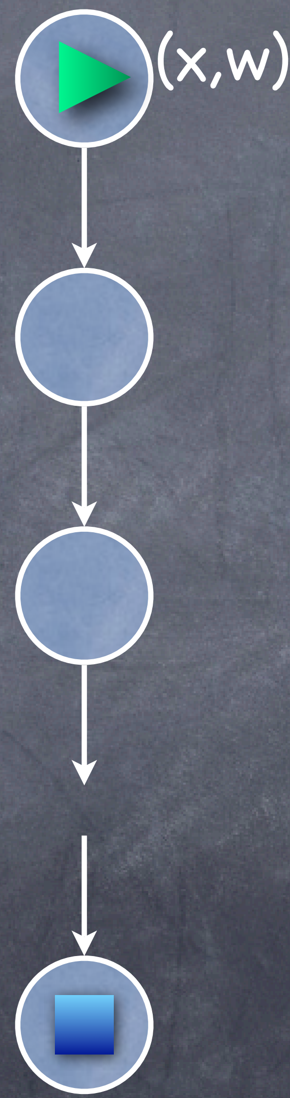
- Reduce any NP language L to CKT-SAT
  - Let's start from the TM for verifying membership in L, with time bound T
  - Build a circuit which on input w outputs what the TM outputs on (x,w), within T steps
  - This circuit is an instance of CKT-SAT
  - Ensure reduction is poly-time

TM to Circuit



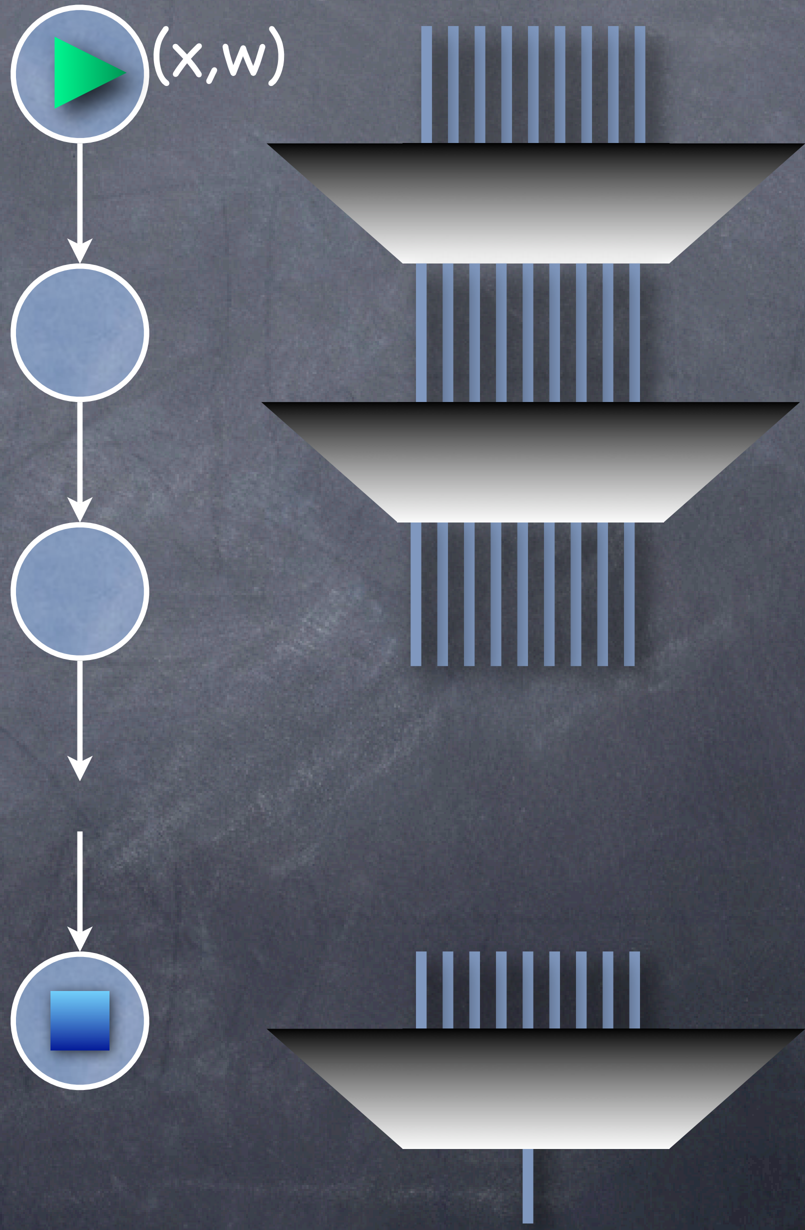
TM to Circuit

- Wires for configurations: a bundle for each tape cell, encoding (content,state), where state is encoded in the cell with the head



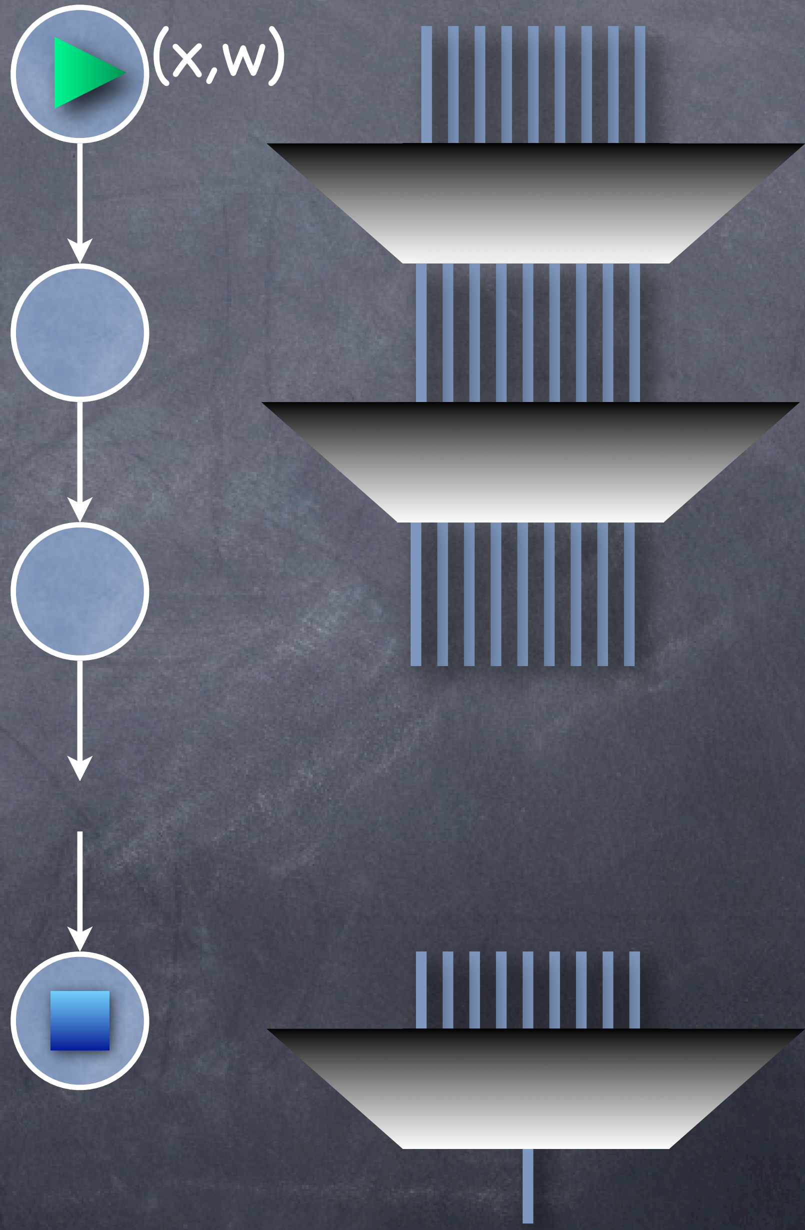
TM to Circuit

- Wires for configurations: a bundle for each tape cell, encoding (content,state), where state is encoded in the cell with the head



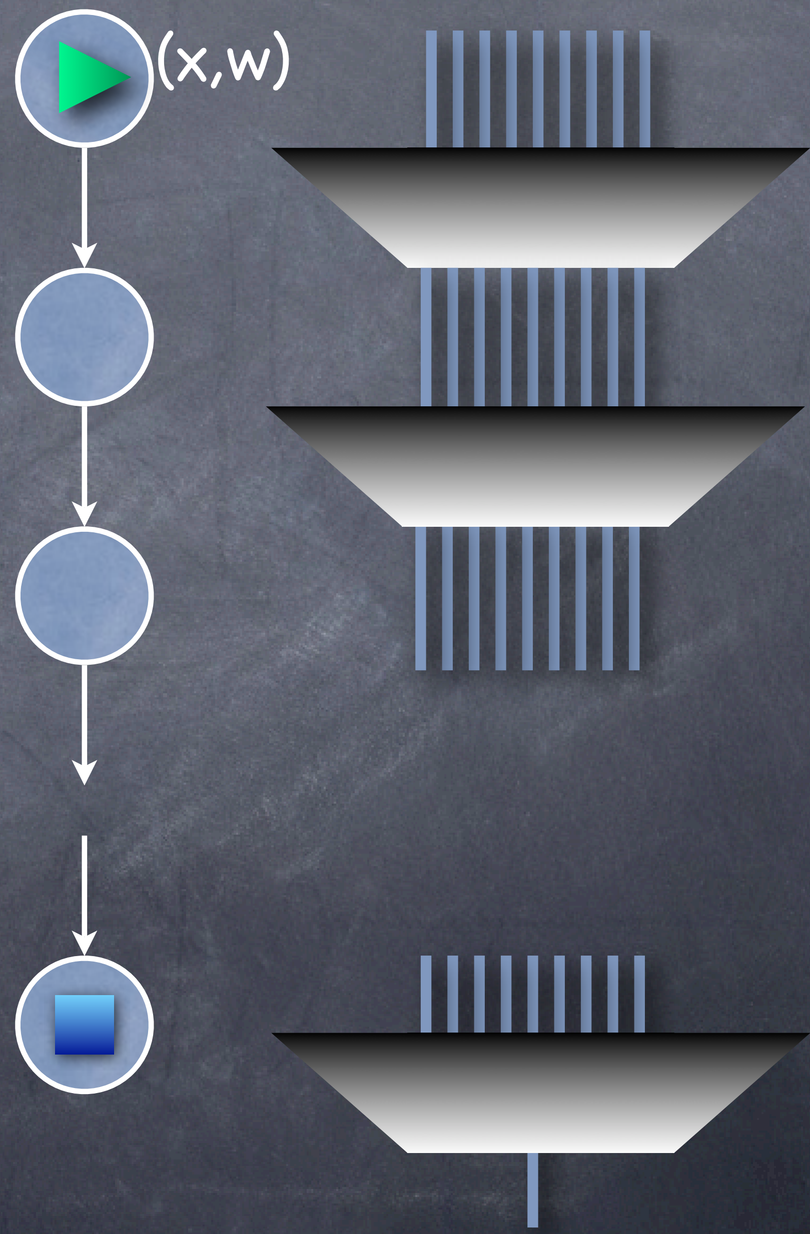
TM to Circuit

- Wires for configurations: a bundle for each tape cell, encoding (content,state), where state is encoded in the cell with the head
- Circuitry for evolution: each bundle depends only on 3 previous bundles



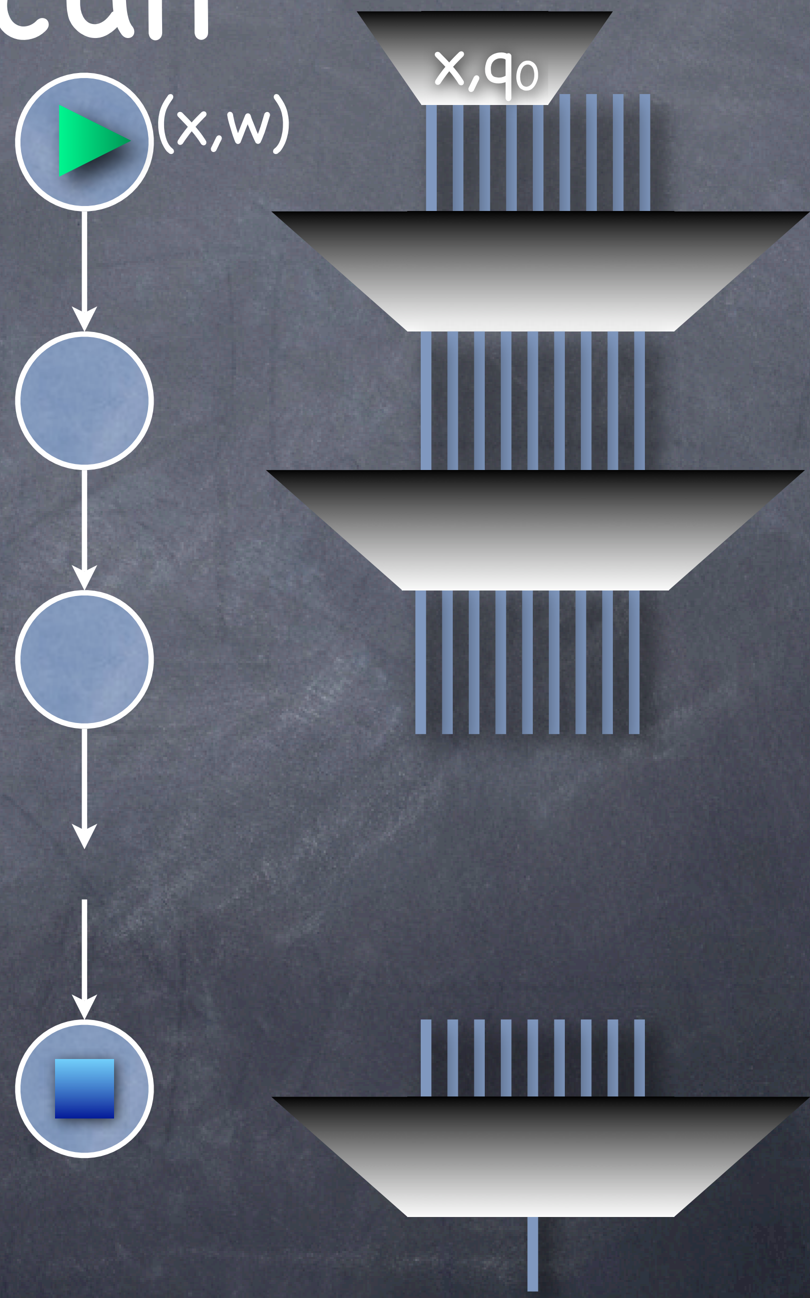
TM to Circuit

- Wires for configurations: a bundle for each tape cell, encoding (content,state), where state is encoded in the cell with the head
- Circuitry for evolution: each bundle depends only on 3 previous bundles
- (Part of) initial configuration, namely w, to be plugged in as input



TM to Circuit

- Wires for configurations: a bundle for each tape cell, encoding (content,state), where state is encoded in the cell with the head
- Circuitry for evolution: each bundle depends only on 3 previous bundles
- (Part of) initial configuration, namely w, to be plugged in as input

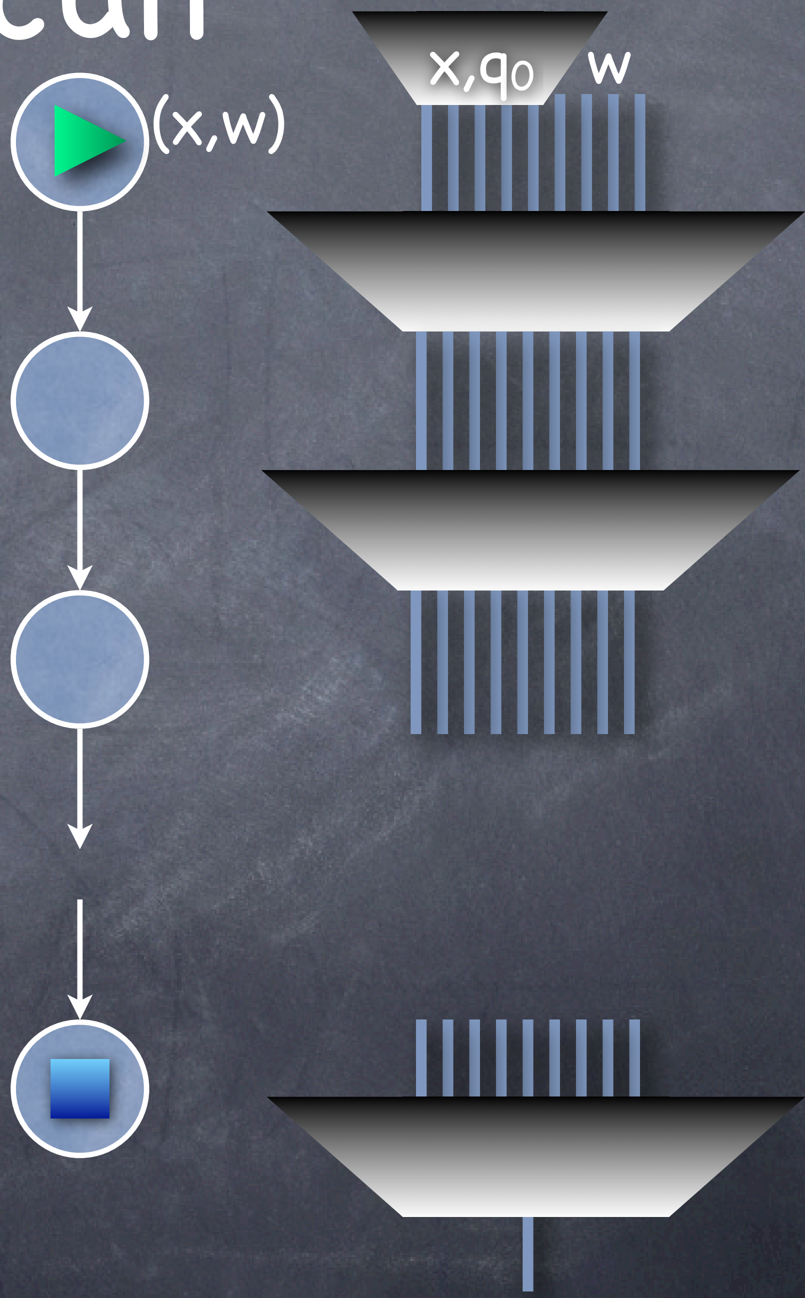


TM to Circuit

- Wires for configurations: a bundle for each tape cell, encoding (content,state), where state is encoded in the cell with the head

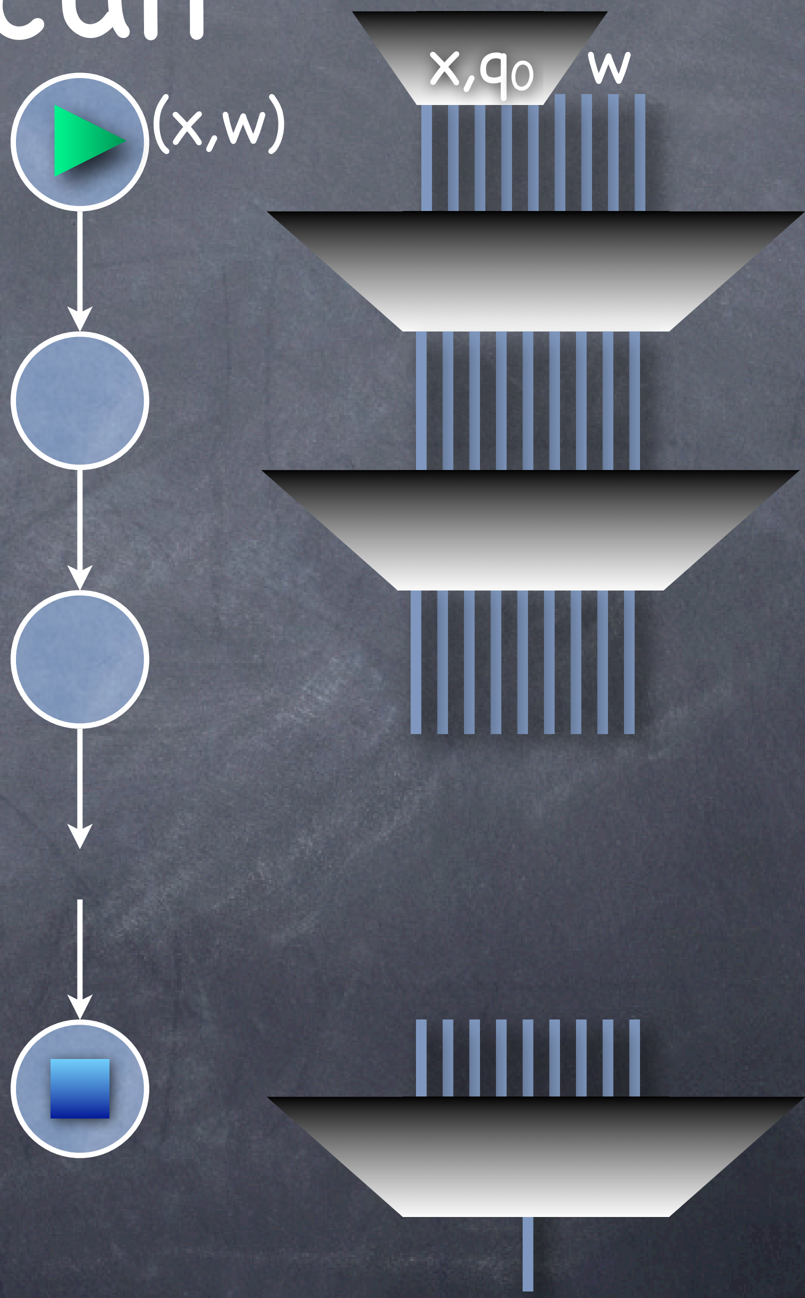
- Circuitry for evolution: each bundle depends only on 3 previous bundles

- (Part of) initial configuration, namely w, to be plugged in as input



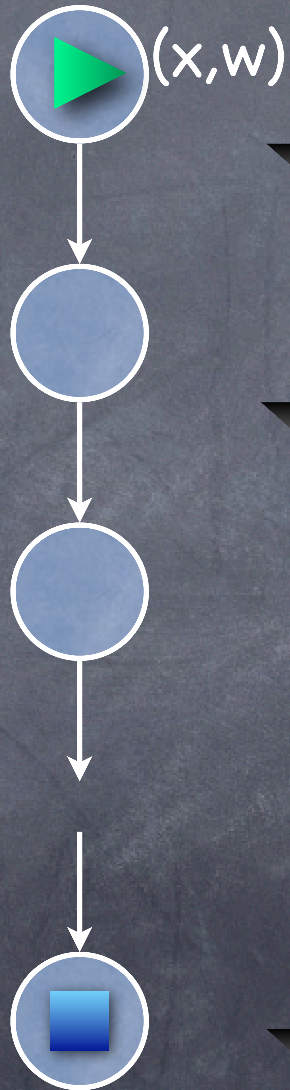
TM to Circuit

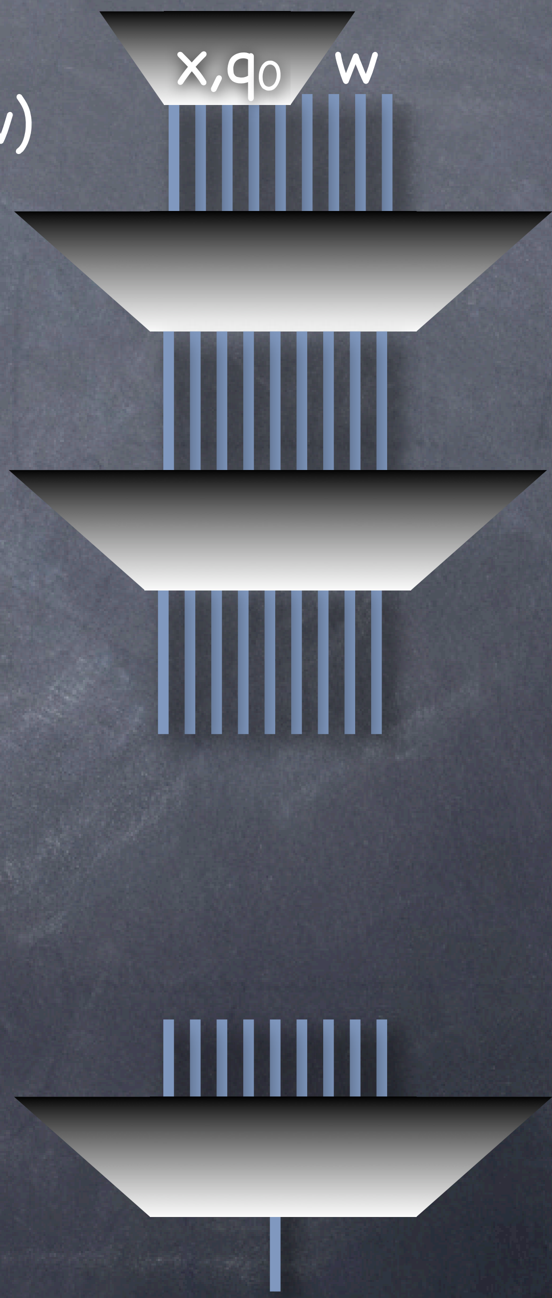
- Wires for configurations: a bundle for each tape cell, encoding (content,state), where state is encoded in the cell with the head
- Circuitry for evolution: each bundle depends only on 3 previous bundles
- (Part of) initial configuration, namely w, to be plugged in as input
- T configurations, T bundles each



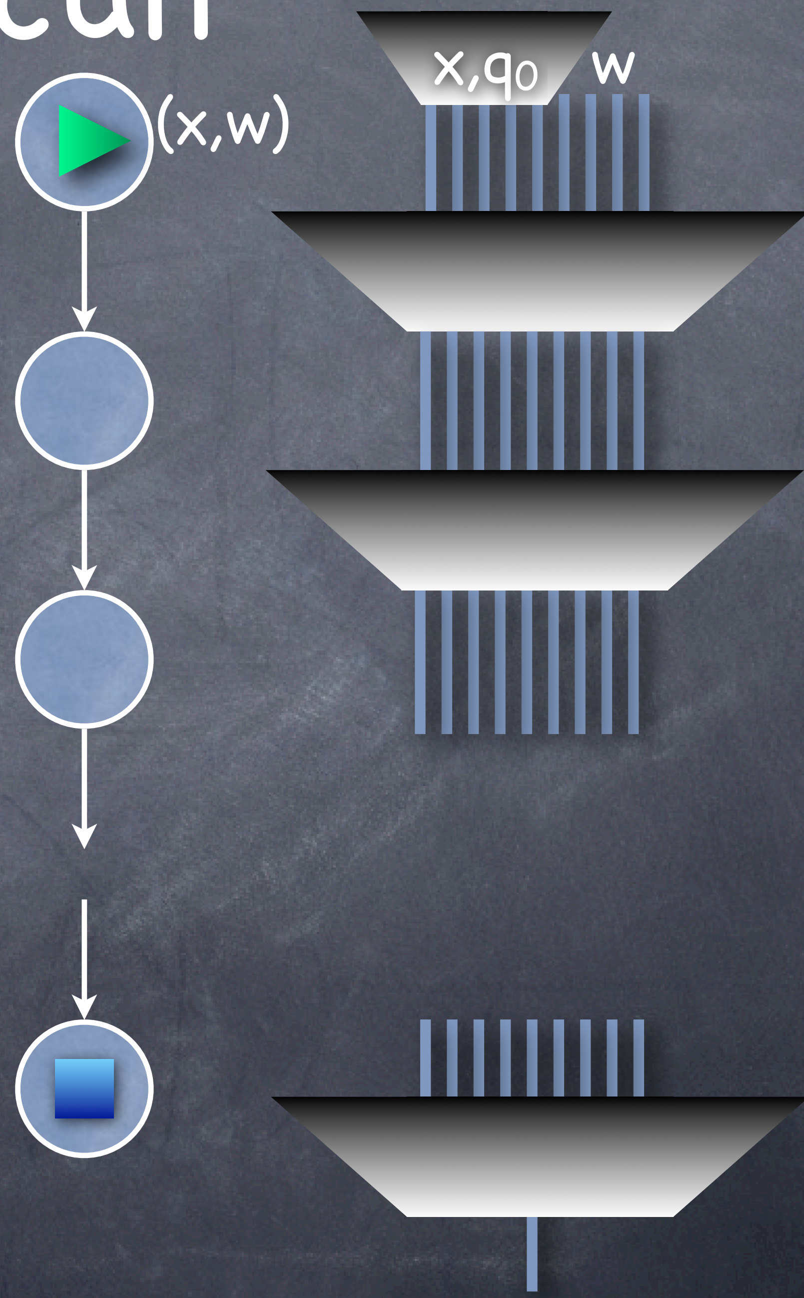
TM to Circuit

- Wires for configurations: a bundle for each tape cell, encoding (content,state), where state is encoded in the cell with the head
- Circuitry for evolution: each bundle depends only on 3 previous bundles
- (Part of) initial configuration, namely w, to be plugged in as input
- T configurations, T bundles each
- Circuit size = $O(T^2)$



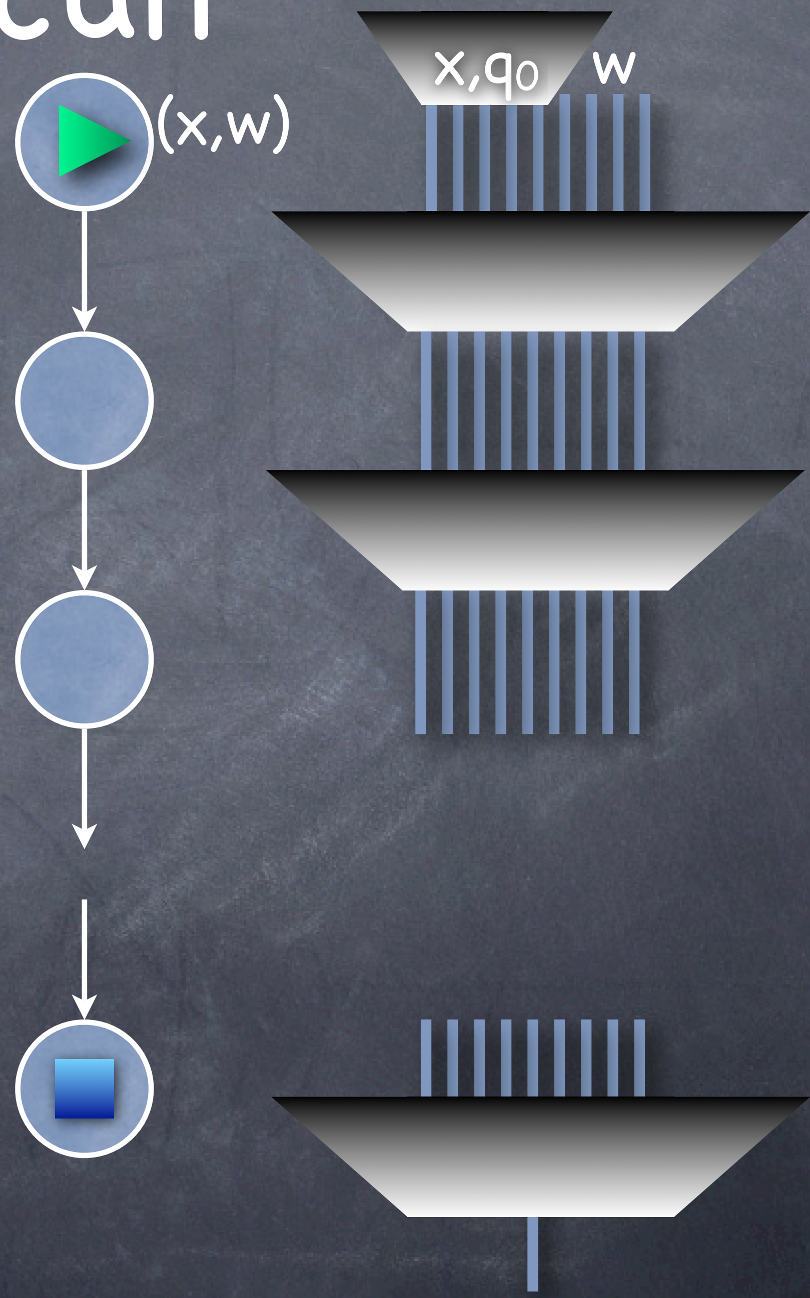


TM to Circuit



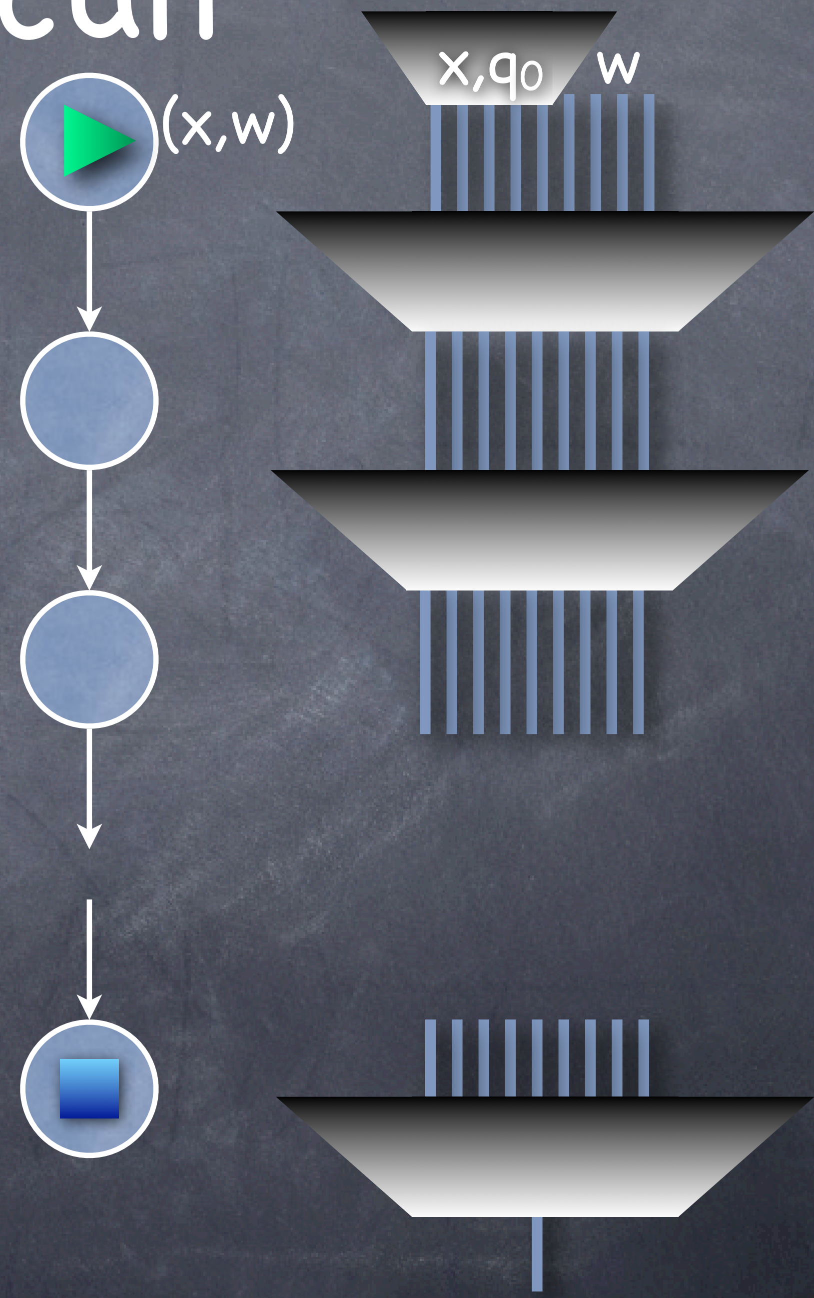
TM to Circuit

- Reducing any NP language L to CKT-SAT



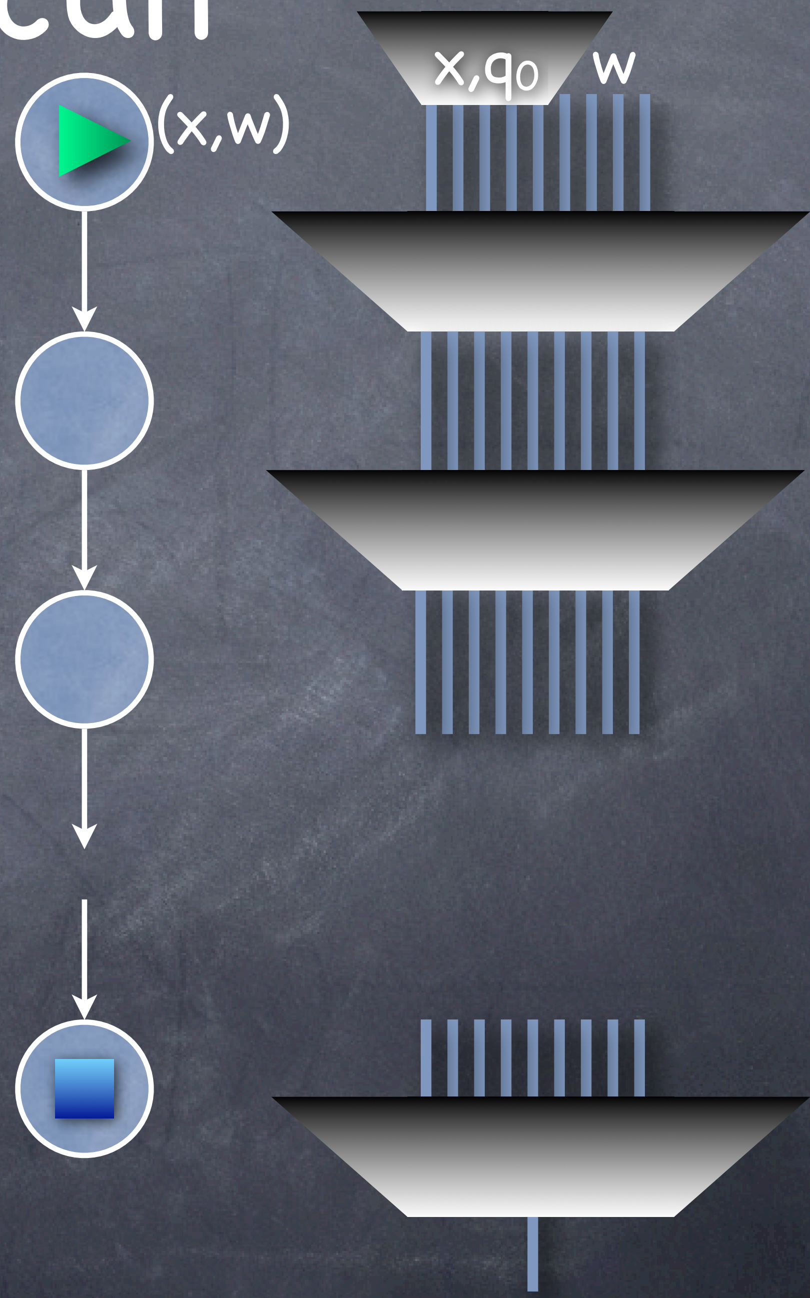
TM to Circuit

- Reducing any NP language L to CKT-SAT
  - TM for verifying membership in L, time-bound T, and input x → A circuit which on input w outputs what the TM outputs on (x,w) within T steps



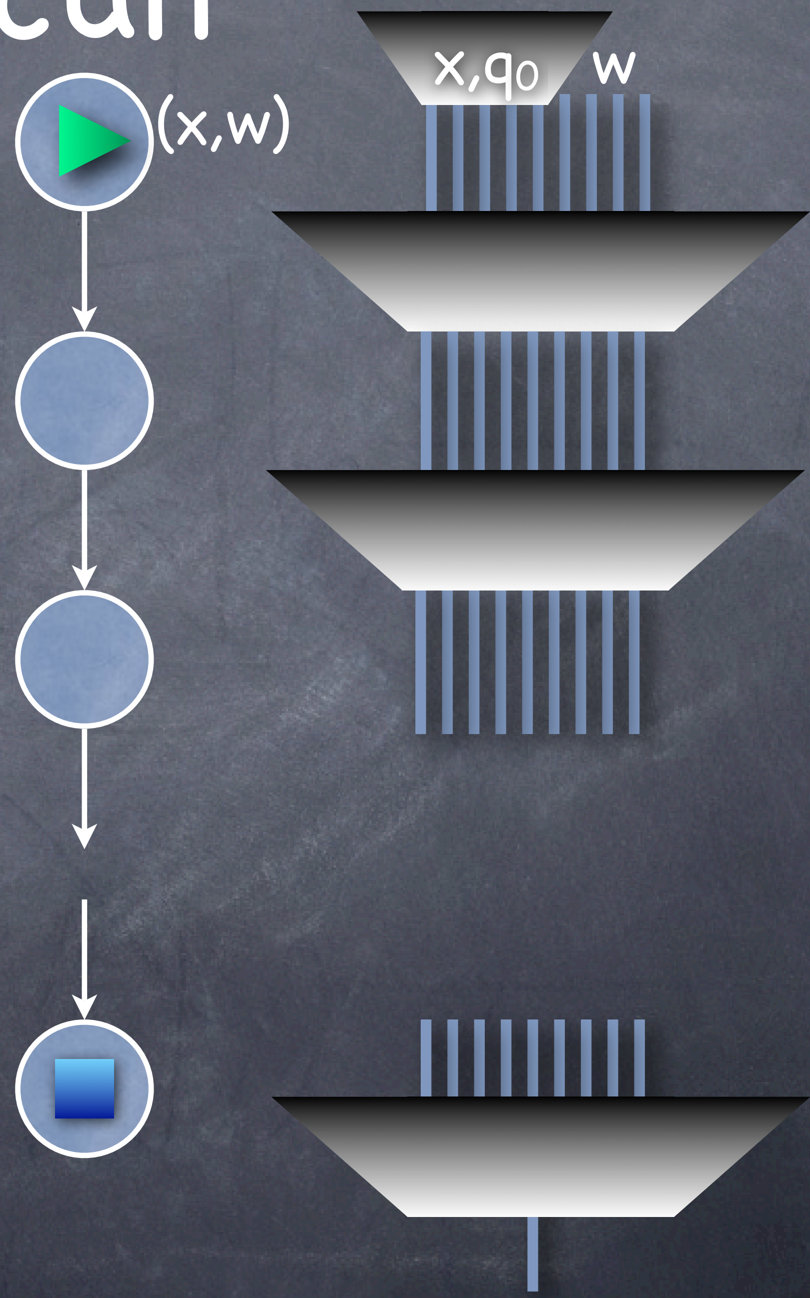
TM to Circuit

- Reducing any NP language L to CKT-SAT
  - TM for verifying membership in L, time-bound T, and input x → A circuit which on input w outputs what the TM outputs on (x,w) within T steps
  - Poly-time reduction



TM to Circuit

- Reducing any NP language L to CKT-SAT
  - TM for verifying membership in L, time-bound T, and input x → A circuit which on input w outputs what the TM outputs on (x,w) within T steps
  - Poly-time reduction
  - CKT-SAT is NP-complete



Other NP-complete problems

Other NP-complete problems

- SAT and 3SAT

Other NP-complete problems

- SAT and 3SAT
  - SAT: Are all given “clauses” simultaneously satisfiable? (Conjunctive Normal Form)

Other NP-complete problems

- SAT and 3SAT
  - SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
  - 3SAT: Each clause has at most 3 literals

Other NP-complete problems

- SAT and 3SAT
  - SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
  - 3SAT: Each clause has at most 3 literals
- CLIQUE, INDEP-SET, VERTEX-COVER

Other NP-complete problems

- SAT and 3SAT
  - SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
  - 3SAT: Each clause has at most 3 literals
- CLIQUE, INDEP-SET, VERTEX-COVER
- Hundreds (thousands?) more

Other NP-complete problems

- SAT and 3SAT
  - SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
  - 3SAT: Each clause has at most 3 literals
- CLIQUE, INDEP-SET, VERTEX-COVER
- Hundreds (thousands?) more
- Shown using already known ones:

Other NP-complete problems

- SAT and 3SAT
  - SAT: Are all given "clauses" simultaneously satisfiable? (Conjunctive Normal Form)
  - 3SAT: Each clause has at most 3 literals
- CLIQUE, INDEP-SET, VERTEX-COVER
- Hundreds (thousands?) more
- Shown using already known ones:
  - If $L \leq_p L_1$ and $L_1 \leq_p L_2$, then $L \leq_p L_2$

CKT-SAT $\leq_p$ SAT

CKT-SAT $\leq_p$ SAT

- Converting a circuit to a collection of clauses:

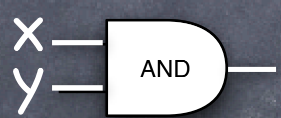
CKT-SAT $\leq_p$ SAT

- Converting a circuit to a collection of clauses:
  - For each wire (connected component), add a variable

CKT-SAT $\leq_p$ SAT

- Converting a circuit to a collection of clauses:
  - For each wire (connected component), add a variable
  - For each gate, add a clause involving variables for wires connected to the gate:

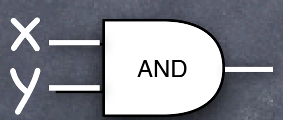
CKT-SAT $\leq_p$ SAT

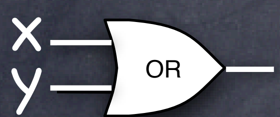
- Converting a circuit to a collection of clauses:
  - For each wire (connected component), add a variable
  - For each gate, add a clause involving variables for wires connected to the gate:
    - e.g. 

      z: $(z \Rightarrow x)$, $(z \Rightarrow y)$, $(\neg z \Rightarrow \neg x \vee \neg y)$.

      i.e., $(\neg z \vee x)$, $(\neg z \vee y)$, $(z \vee \neg x \vee y)$.

CKT-SAT $\leq_p$ SAT

- Converting a circuit to a collection of clauses:
  - For each wire (connected component), add a variable
  - For each gate, add a clause involving variables for wires connected to the gate:
    - e.g. 

z: $(z \Rightarrow x)$, $(z \Rightarrow y)$, $(\neg z \Rightarrow \neg x \vee \neg y)$.
i.e., $(\neg z \vee x)$, $(\neg z \vee y)$, $(z \vee \neg x \vee y)$.
    - and 

z: $(z \Rightarrow x \vee y)$, $(\neg z \Rightarrow \neg x)$, $(\neg z \Rightarrow \neg y)$.

SAT $\leq_p$ 3SAT

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.
- More directly:

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.
- More directly:
  - $(a \vee b \vee c \vee d \vee e) \rightarrow (a \vee b \vee x), (\neg x \vee c \vee d \vee e)$

    $\rightarrow (a \vee b \vee x), (\neg x \vee c \vee y), (\neg y \vee d \vee e)$

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.
- More directly:
  - $(a \vee b \vee c \vee d \vee e) \rightarrow (a \vee b \vee x), (\neg x \vee c \vee d \vee e)$

    $\rightarrow (a \vee b \vee x), (\neg x \vee c \vee y), (\neg y \vee d \vee e)$
- Reduction needs 3SAT

$SAT \leq_p 3SAT$

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So $SAT \leq_p 3SAT$.
- More directly:
  - $(a \vee b \vee c \vee d \vee e) \rightarrow (a \vee b \vee x), (\neg x \vee c \vee d \vee e)$

    $\rightarrow (a \vee b \vee x), (\neg x \vee c \vee y), (\neg y \vee d \vee e)$
- Reduction needs 3SAT
  - 2SAT is in fact in P! [Exercise]

SAT $\leq_p$ 3SAT

- Previous reduction was to 3SAT, so 3SAT is NP-complete. And SAT is in NP. So SAT $\leq_p$ 3SAT.
- More directly:
  - $(a \lor b \lor c \lor d \lor e) \rightarrow (a \lor b \lor x), (\neg x \lor c \lor d \lor e)$
    $\rightarrow (a \lor b \lor x), (\neg x \lor c \lor y), (\neg y \lor d \lor e)$
- Reduction needs 3SAT
  - 2SAT is in fact in P! [Exercise]
- Reduction not parsimonious (can you make it? [Exercise])

3SAT $\leq_p$ CLIQUE

3SAT $\leq_p$ CLIQUE

- Clauses → Graph

3SAT $\leq_p$ CLIQUE

- Clauses → Graph

$(x \lor \neg y \lor \neg z)$

$(w \lor y)$

$(w \lor x \lor \neg z)$

3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)

$(x \lor \neg y \lor \neg z)$

$(w \lor y)$

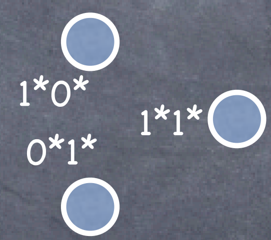
$(w \lor x \lor \neg z)$

3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)

(x ∨ ¬y ∨ ¬z)

(w ∨ y)



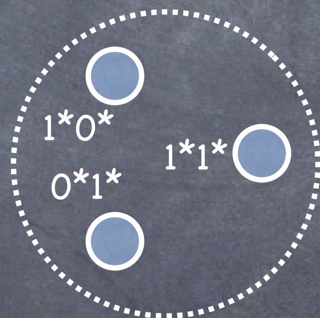
(w ∨ x ∨ ¬z)

3SAT $\leq_p$ CLIQUE

(x ∨ ¬y ∨ ¬z)

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)

(w ∨ y)

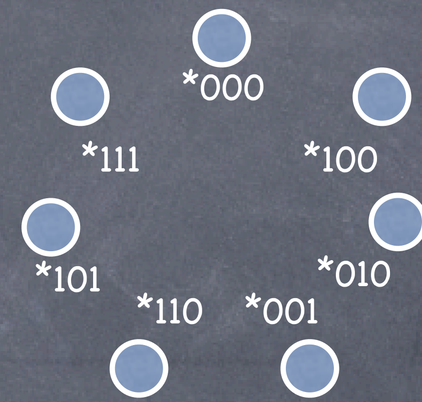


(w ∨ x ∨ ¬z)

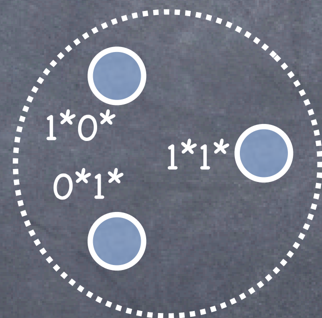
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)

$(x \lor \neg y \lor \neg z)$



$(w \lor y)$

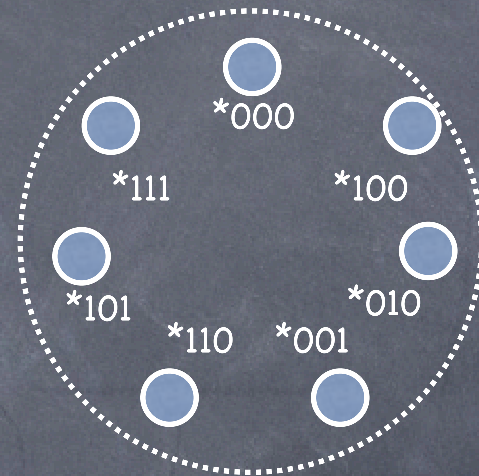


$(w \lor x \lor \neg z)$

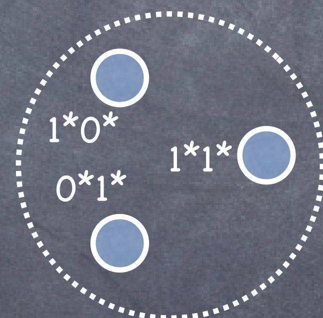
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)

$(x \lor \neg y \lor \neg z)$



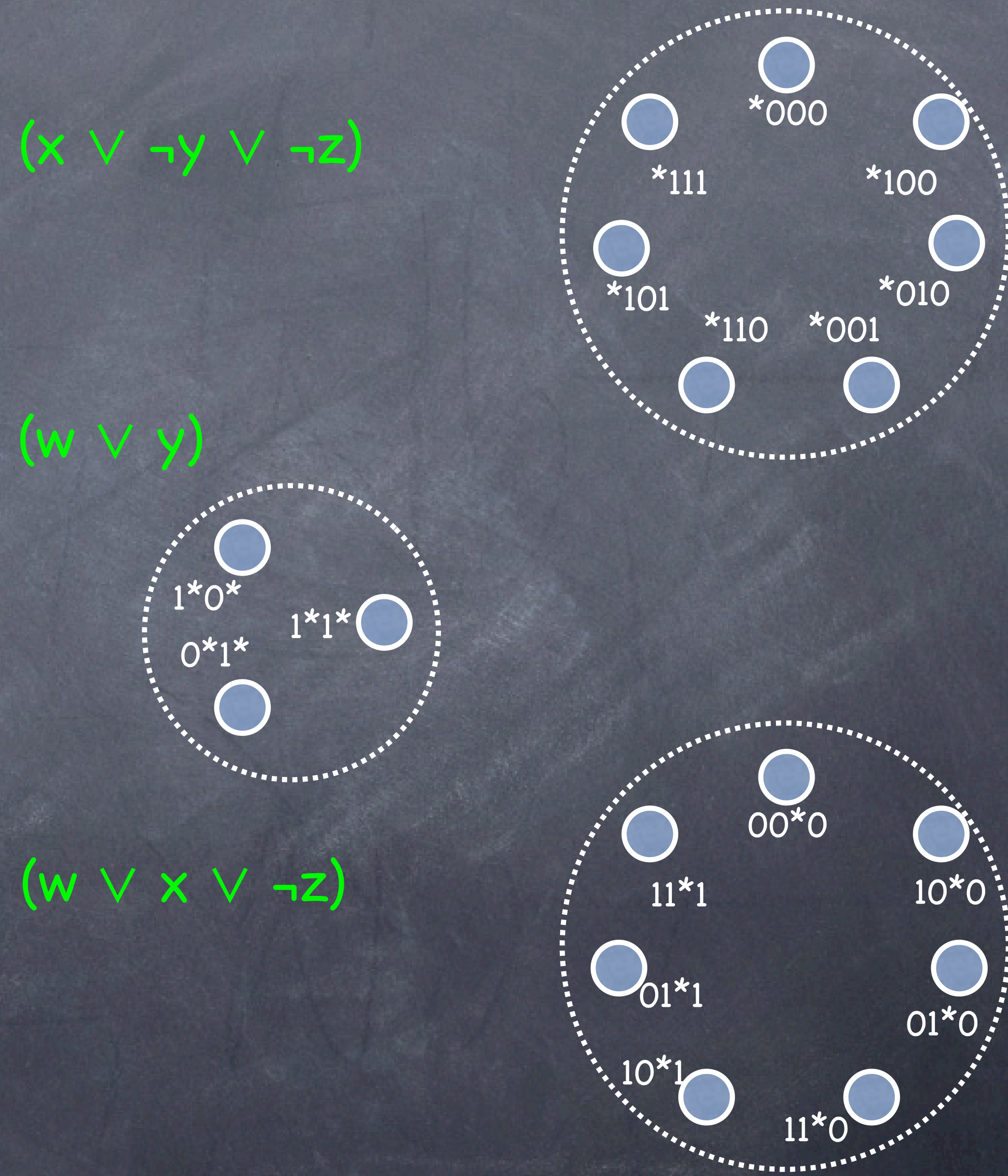
$(w \lor y)$



$(w \lor x \lor \neg z)$

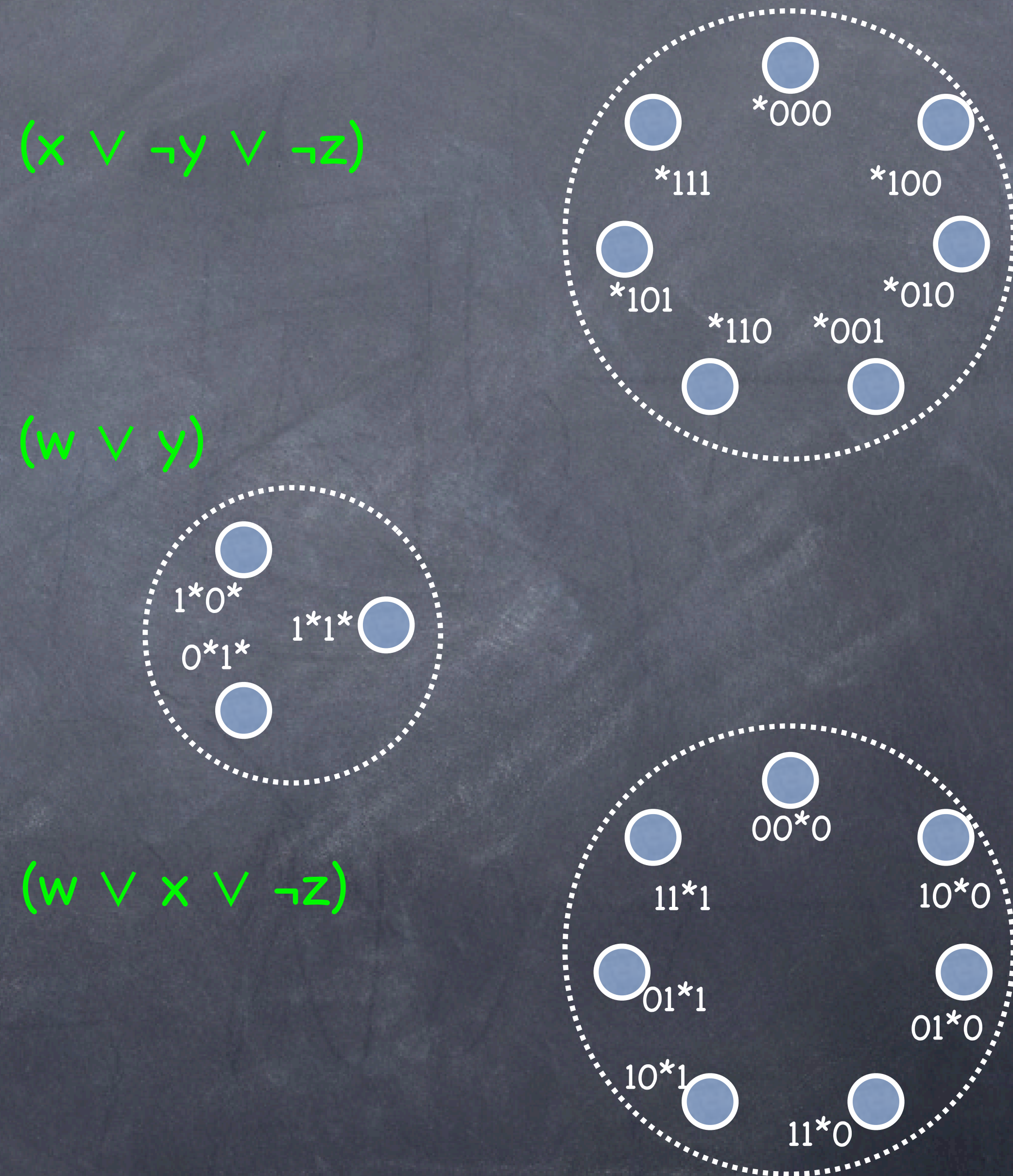
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)



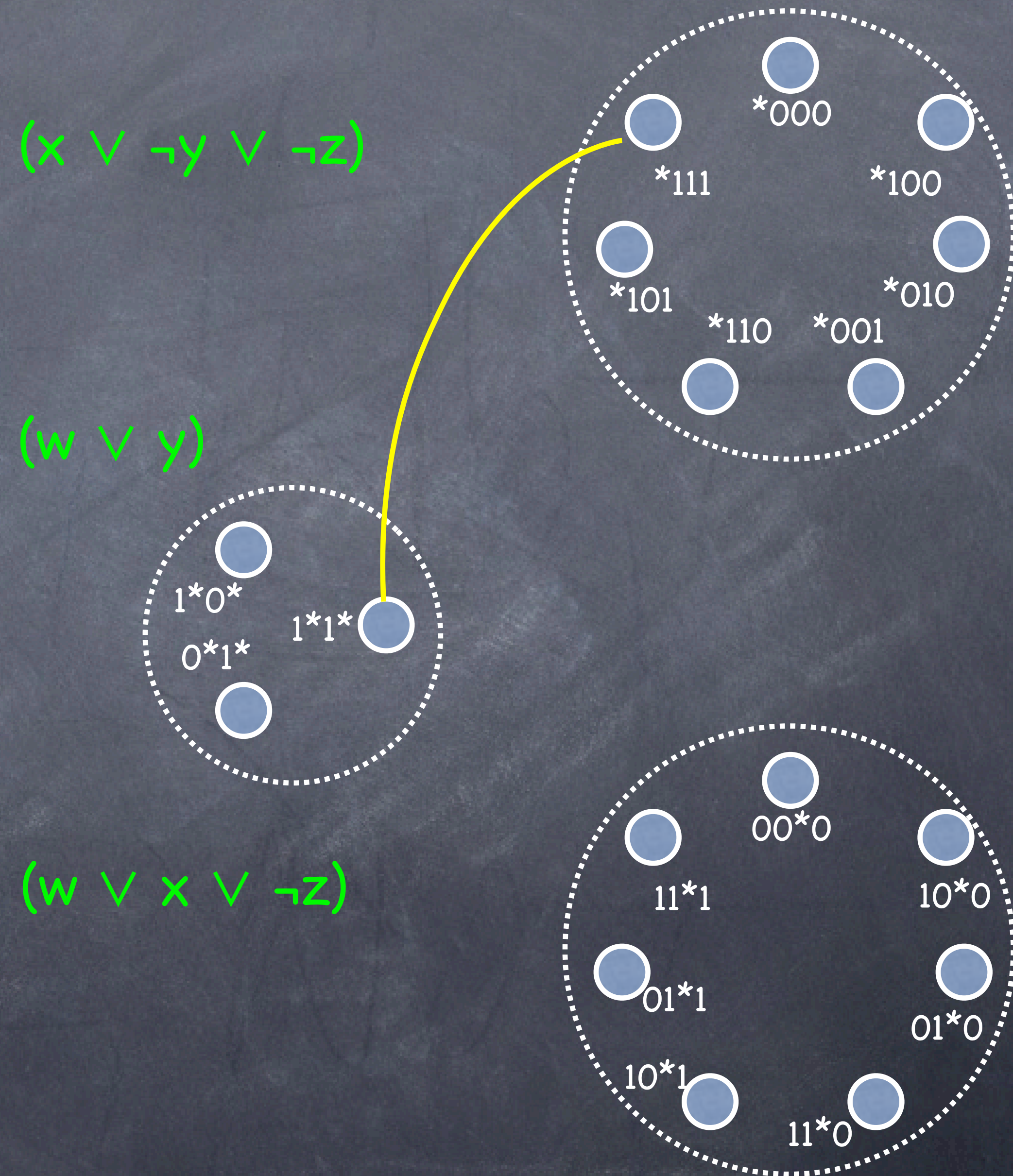
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments



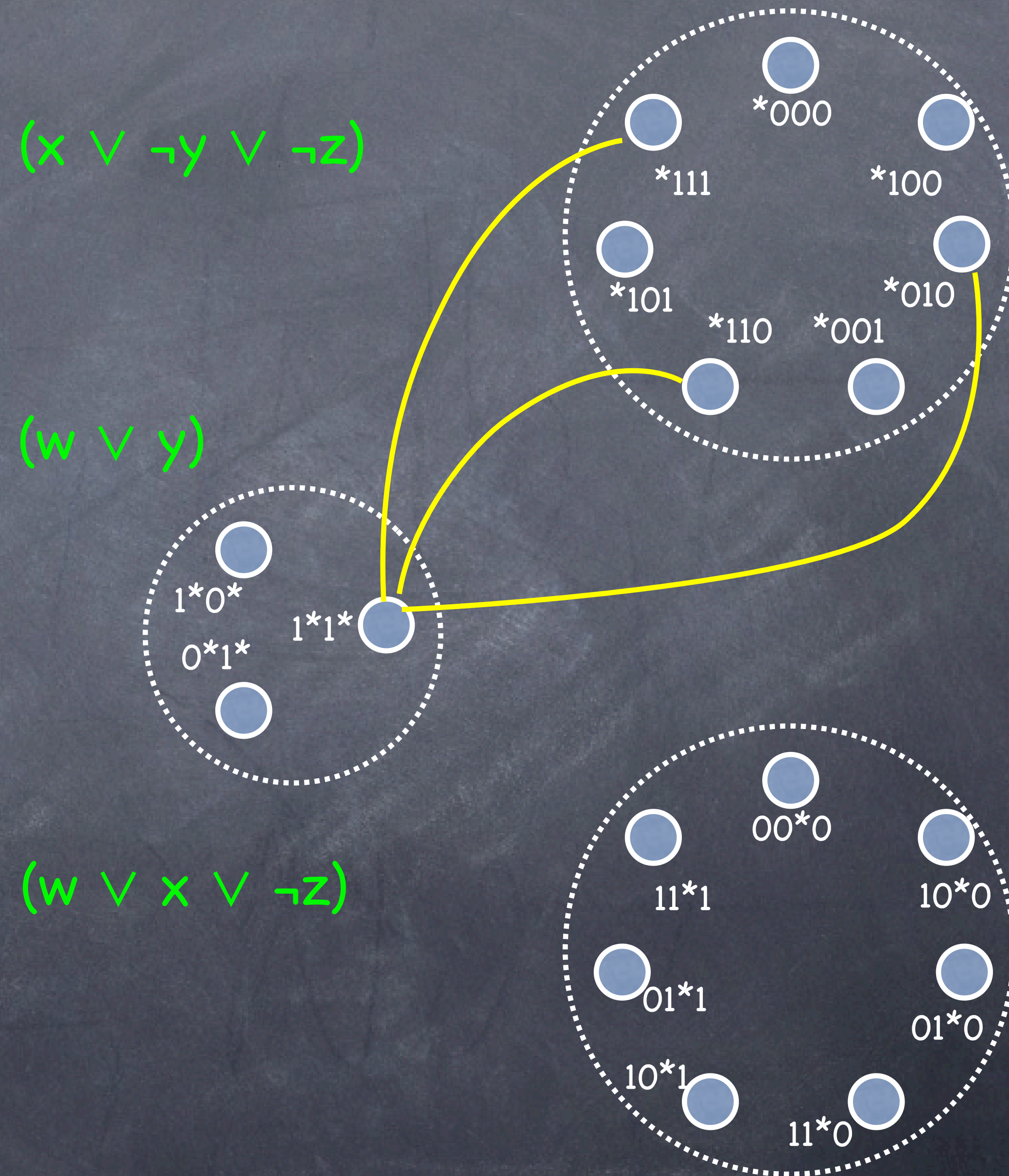
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments



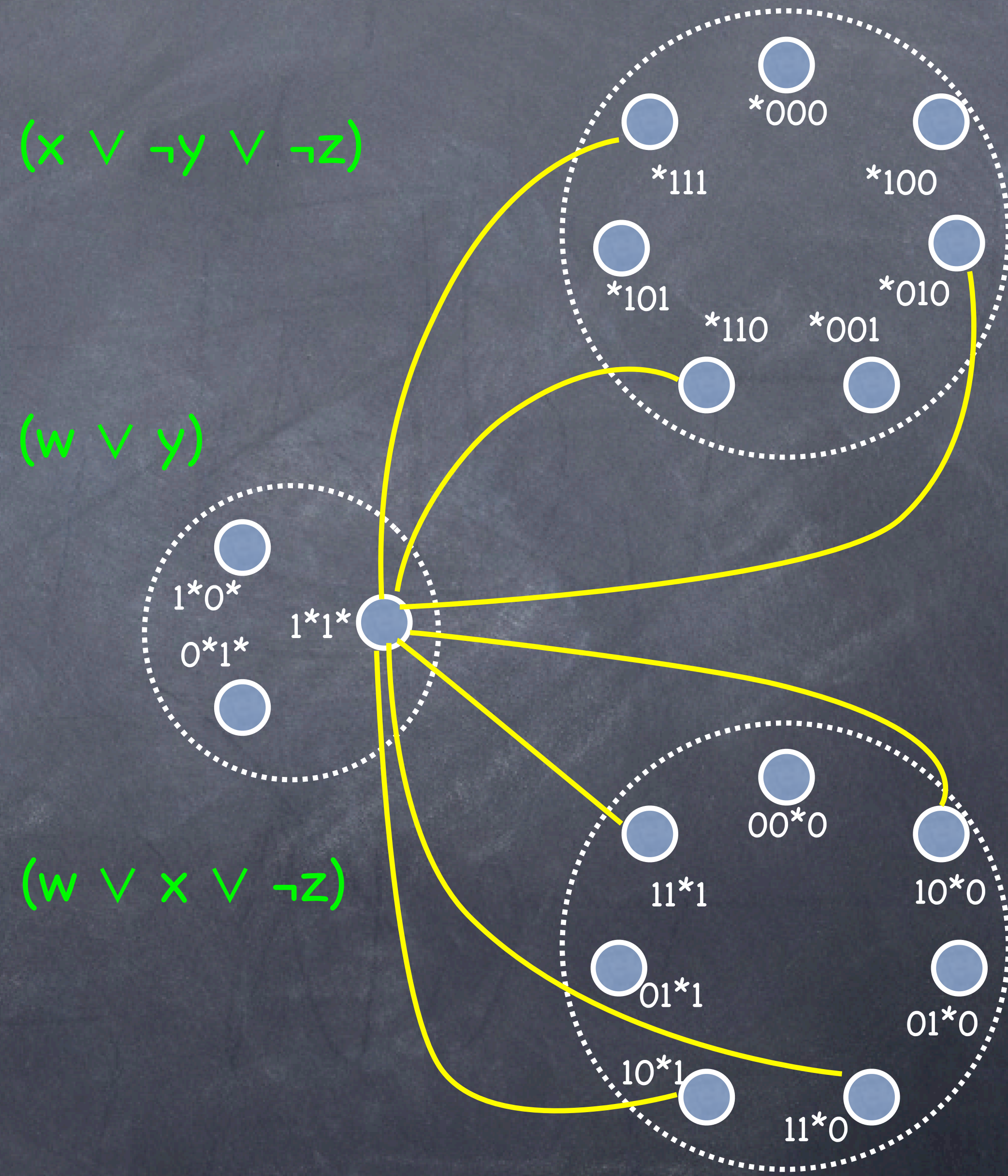
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments



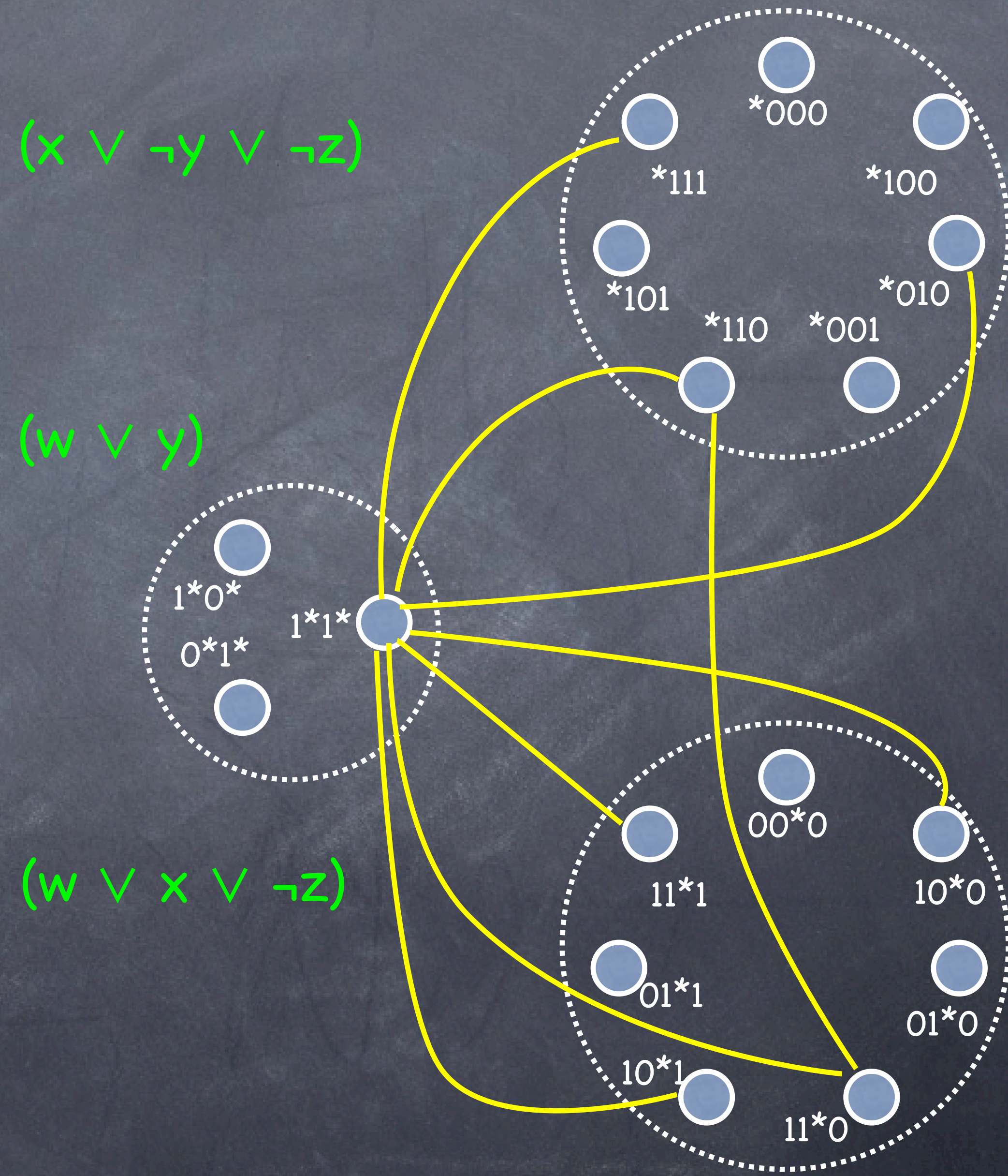
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments



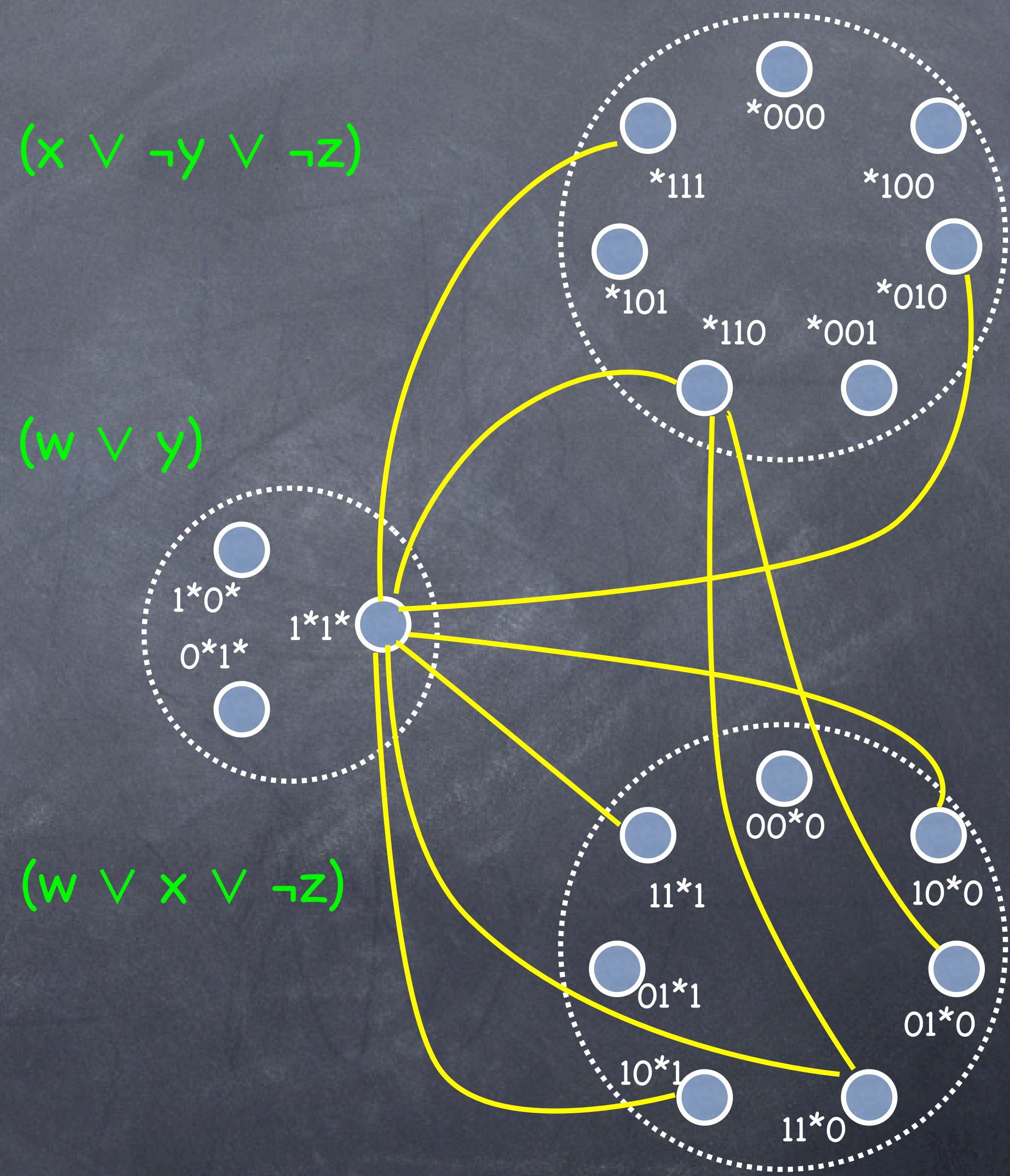
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments



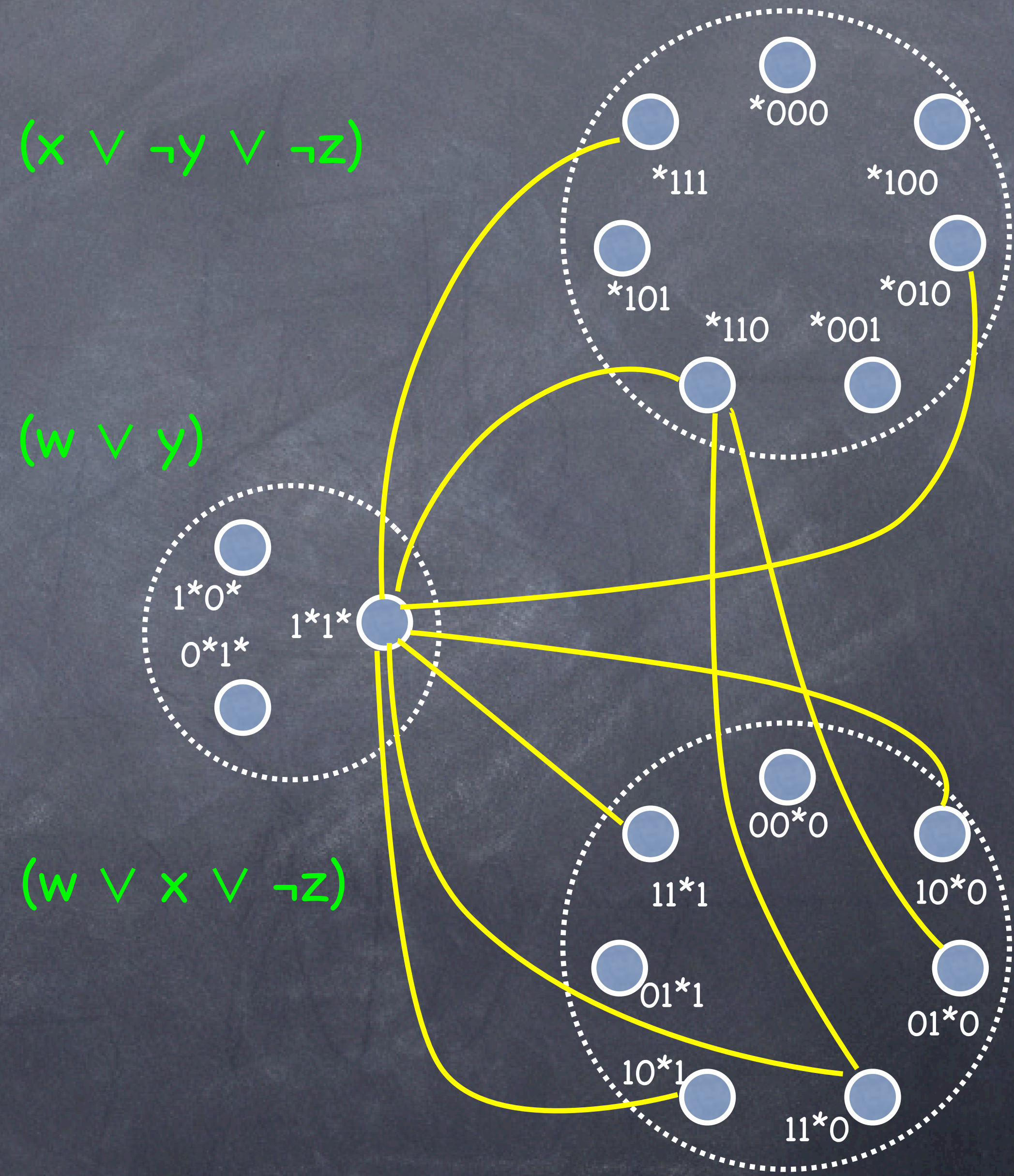
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments



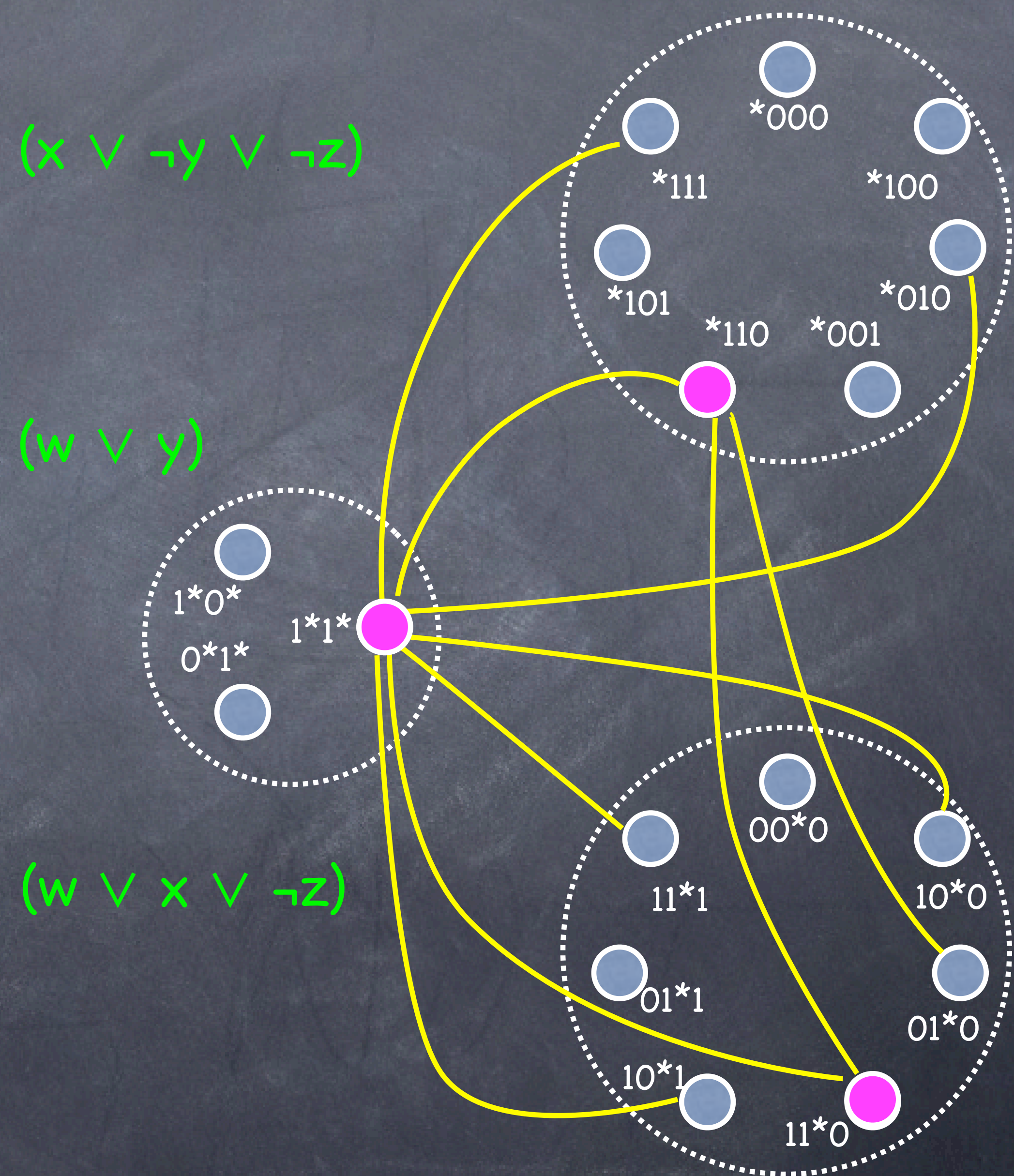
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments
  - m-clique iff all m clauses satisfiable



3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments
  - m-clique iff all m clauses satisfiable



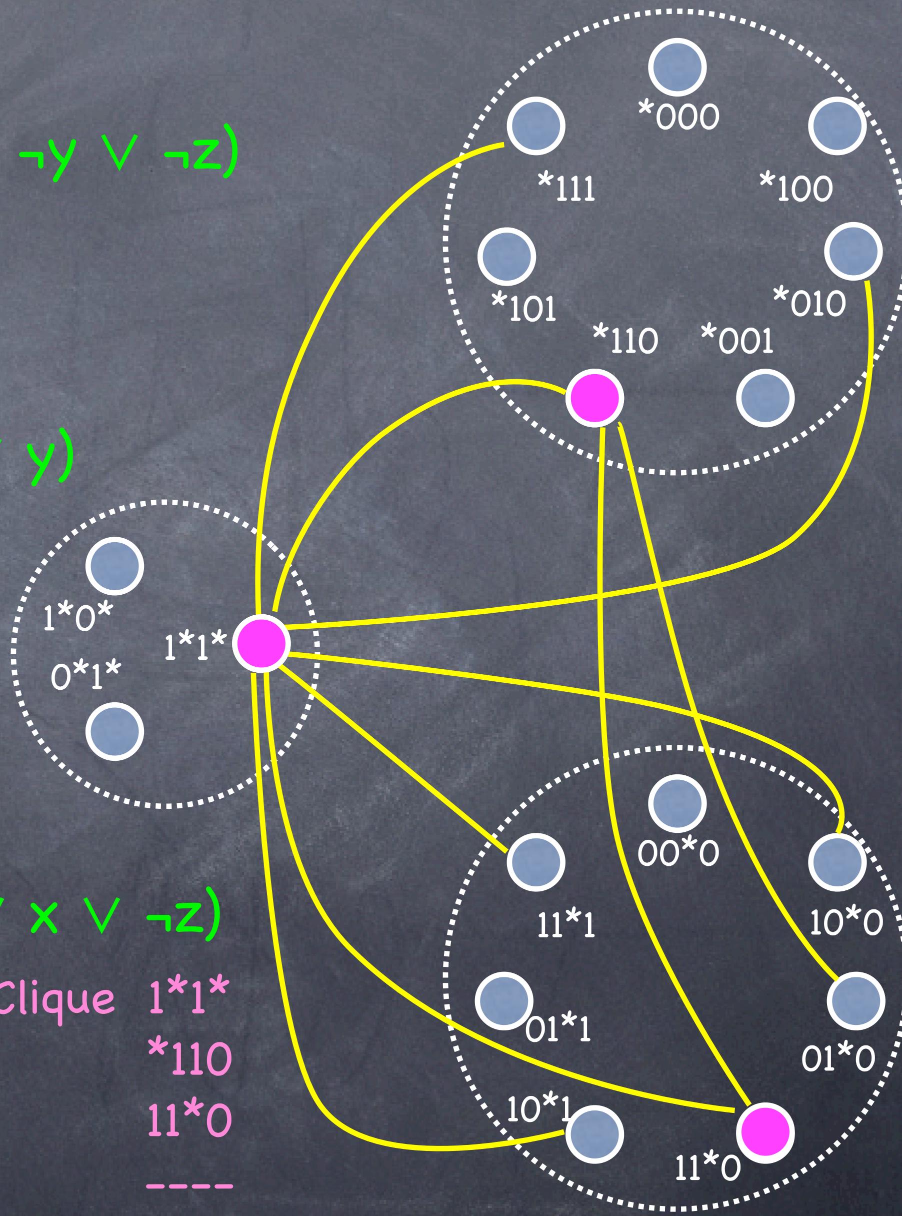
3SAT $\leq_p$ CLIQUE

- Clauses → Graph
  - vertices: each clause's satisfying assignments (for its variables)
  - edges between consistent assignments
  - m-clique iff all m clauses satisfiable

(x ∨ ¬y ∨ ¬z)

(w ∨ y)

(w ∨ x ∨ ¬z)



3-Clique 1*1*
*110
11*0
----
sat assignment 1110

INDEP-SET and VERTEX-COVER

INDEP-SET and VERTEX-COVER

- CLIQUE $\leq_p$ INDEP-SET

INDEP-SET and VERTEX-COVER

- CLIQUE $\leq_p$ INDEP-SET
  - G has an m-clique iff G' has an m-independent-set

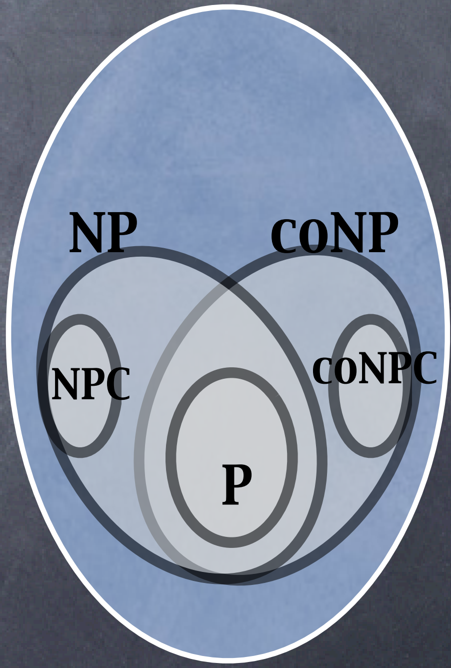
INDEP-SET and VERTEX-COVER

- CLIQUE $\leq_p$ INDEP-SET
  - G has an m-clique iff G' has an m-independent-set
- INDEP-SET $\leq_p$ VERTEX-COVER

INDEP-SET and VERTEX-COVER

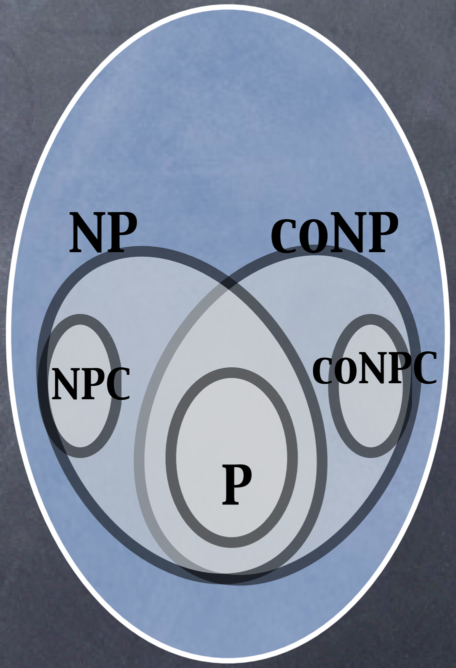
- CLIQUE $\leq_p$ INDEP-SET
  - G has an m-clique iff G' has an m-independent-set
- INDEP-SET $\leq_p$ VERTEX-COVER
  - G has an m-indep-set iff G has an (n-m)-vertex-cover

NP, P, co-NP and NPC



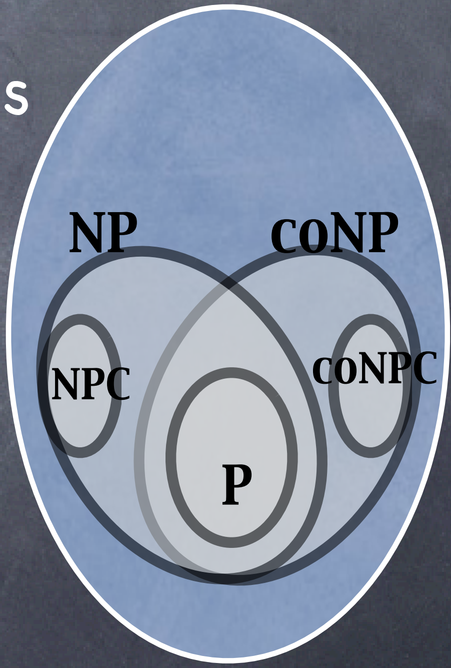
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X



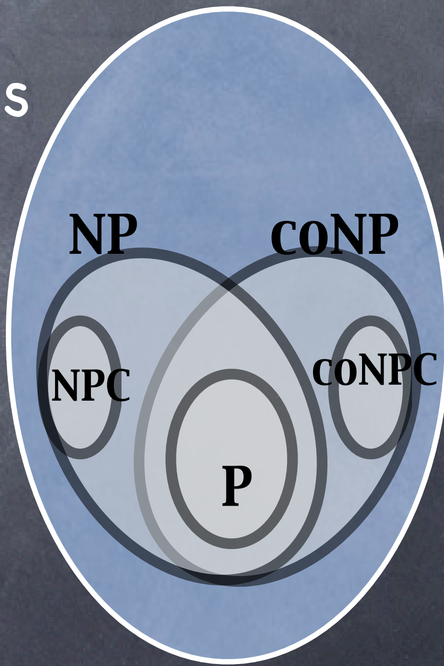
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions



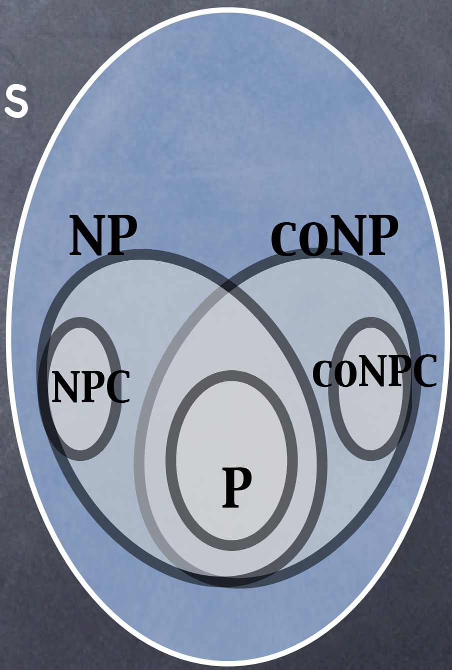
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)



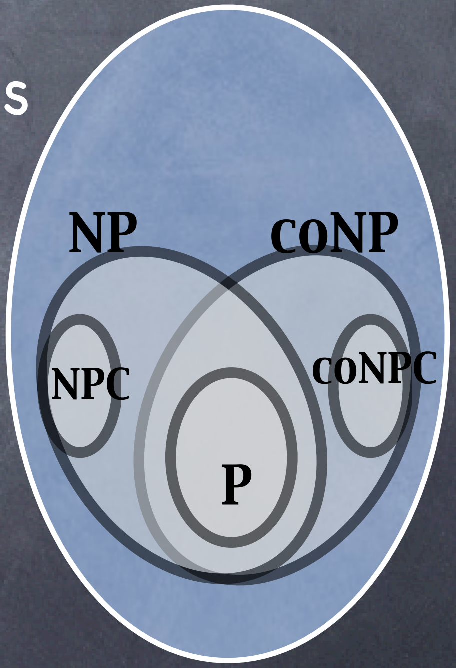
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)
- If any NPC language is in P, then NP = P



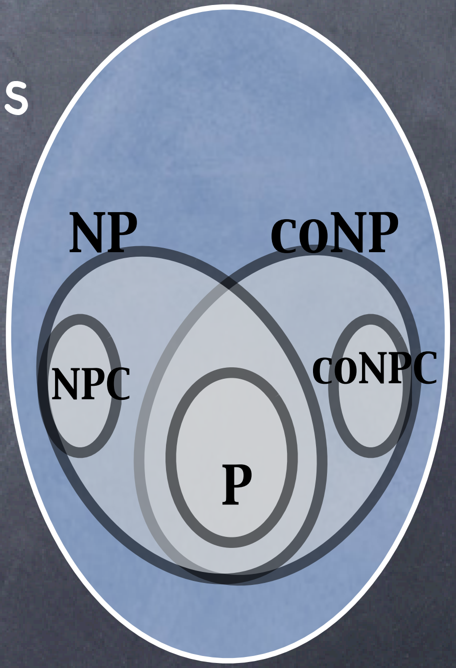
NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)
- If any NPC language is in P, then NP = P
- If any NPC language is in co-NP, NP=co-NP



NP, P, co-NP and NPC

- We say class X is "closed under polynomial reductions" if ($L_1 \leq_p L_2$ and L_2 in class X) implies L_1 in X
  - e.g. P, NP are closed under polynomial reductions
    - So is co-NP (If X is closed, so is co-X. Why?)
- If any NPC language is in P, then NP = P
- If any NPC language is in co-NP, NP=co-NP
  - Note: if L in NPC, L^c is in co-NPC



Today

Today

- Polynomial-time reductions

Today

- Polynomial-time reductions
- NP-completeness (using Karp reductions)

Today

- Polynomial-time reductions
- NP-completeness (using Karp reductions)
  - Trivially, TMSAT

Today

- Polynomial-time reductions
- NP-completeness (using Karp reductions)
  - Trivially, TMSAT
  - Interestingly, CKT-SAT, SAT, 3SAT, CLIQUE, INDEP-SET, VERTEX-COVER

Today

- Polynomial-time reductions
- NP-completeness (using Karp reductions)
  - Trivially, TMSAT
  - Interestingly, CKT-SAT, SAT, 3SAT, CLIQUE, INDEP-SET, VERTEX-COVER
  - If any NPC language in P, then P=NP

Next Time

Next Time

- Ladner's Theorem: If NP $\neq$ P, then non-P, non-NPC languages

Next Time

- Ladner's Theorem: If NP ≠ P, then non-P, non-NPC languages
- Time hierarchy theorems: More time, more power, strictly!