Space Complexity

Motivation

Complexity classes correspond to bounds on resources

One such resource is space: the number of tape cells a TM uses when solving a problem

Introduction

- Objectives:
  - To define space complexity classes
- Overview:
  - Space complexity classes
  - Low space classes: L, NL
  - Savitch's Theorem
  - Immerman's Theorem
  - TQBF

Space Complexity Classes

For any function $f: N \rightarrow R^+$, we define:

SPACE(f(n))={ L : L is decidable by a deterministic O(f(n)) space TM}

NSPACE(f(n))={ L : L is decidable by a non-deterministic O(f(n)) space TM}

Low Space Classes

Definitions (logarithmic space classes):

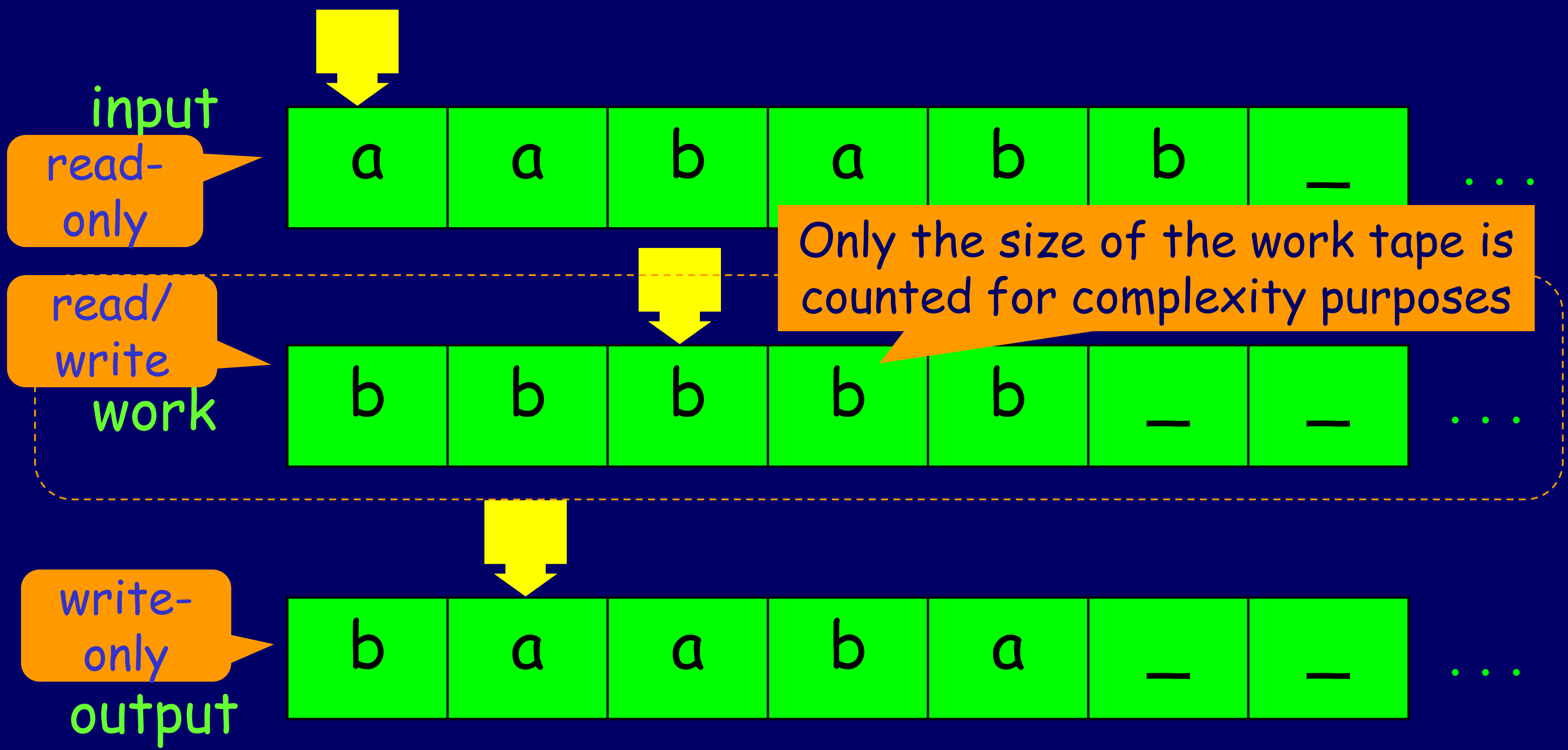
- L = SPACE(logn)
- NL = NSPACE(logn)

Problem!

How can a TM use only logn space if the input itself takes n cells?!

!?

3Tape Machines



Example

Question: How much space would a TM that decides $\{a^n b^n \mid n>0\}$ require?

Note: to count up to n, we need $\log n$ bits

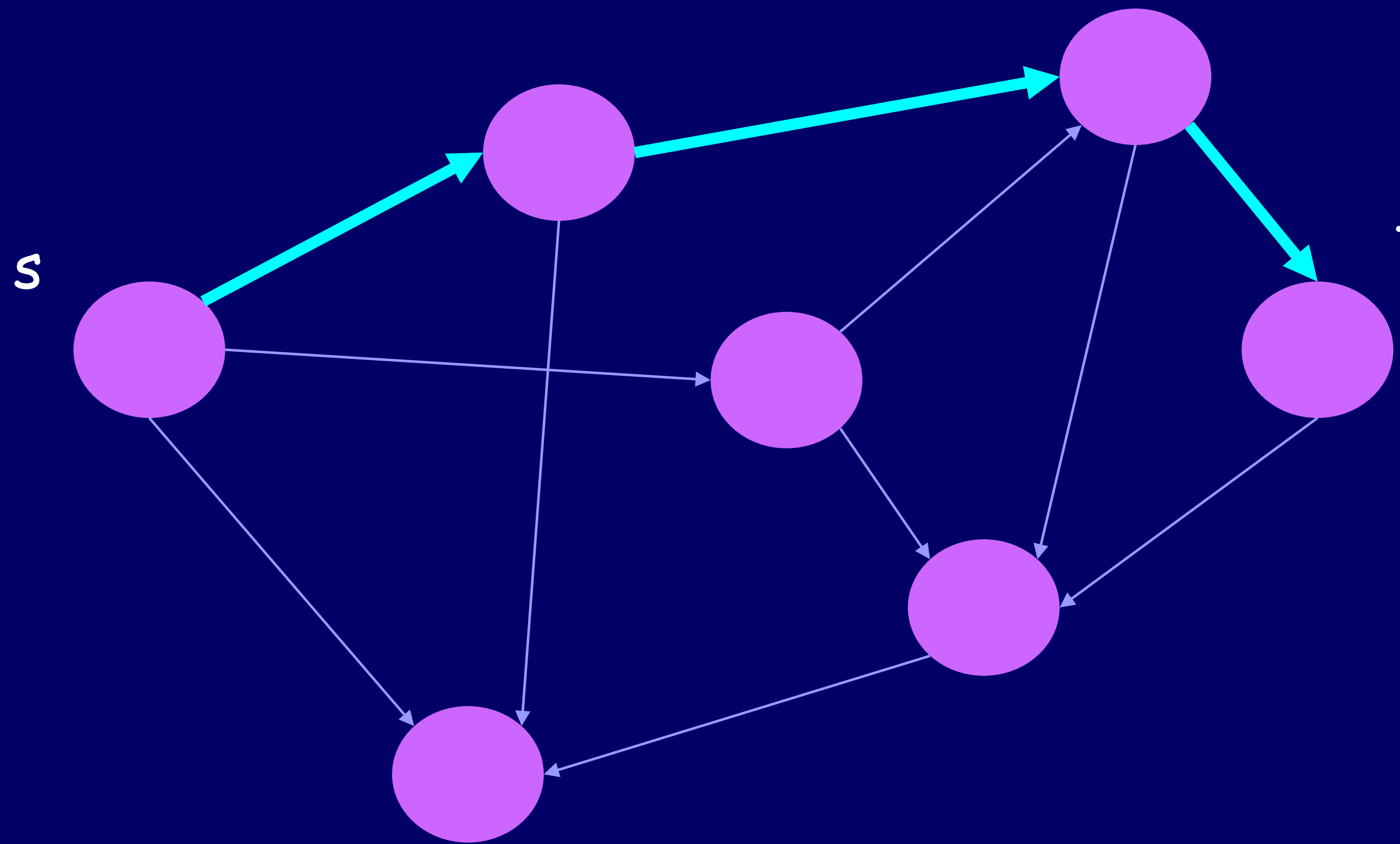
Graph Connectivity

An undirected version is also worth considering

CONN

- <u>Instance</u>: a directed graph $G=(V,E)$ and two vertices $s,t \in V$
- <u>Problem</u>: To decide if there is a path from s to t in G?

Graph Connectivity



CONN is in NL

- Start at s
- For $i = 1, .., |V|$ {
  - Non-deterministically choose a neighbor and jump to it
  - Accept if you get to t

  }
- If you got here – reject!

> • Counting up to $|V|$ requires $\log|V|$ space
>
> • Storing the current position requires $\log|V|$ space

Log-Space Reductions

Definition:

A is log-space reducible to B, written $A \leq_L B$,

if there exists a log space TM M that, given input w, outputs f(w) s.t.

$$w \in A \text{ iff } f(w) \in B$$

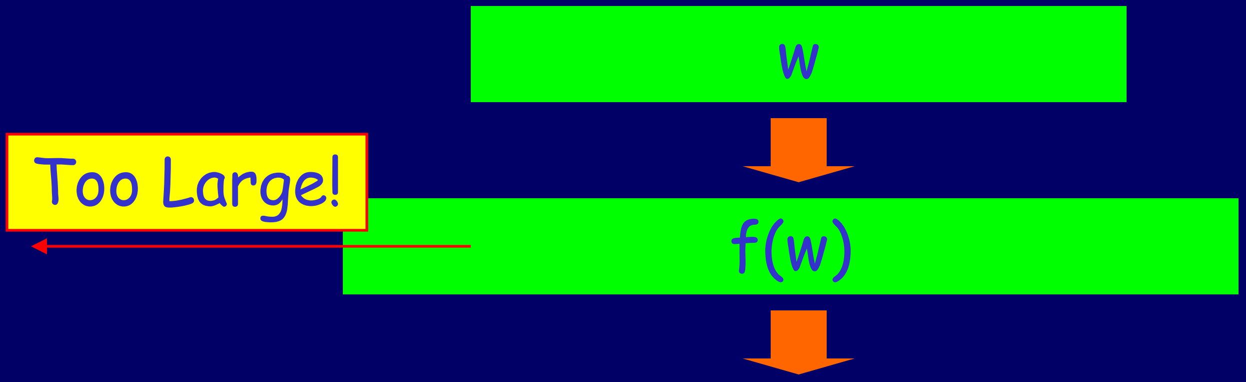
the reduction

- L, NL, P, NP, PSPACE and EXPTIME are closed under log-space reductions.

Do Log-Space Reductions Imply what they should?

Suppose $A_1 \leq_L A_2$ and $A_2 \in L$; how to construct a log space TM which decides A_1?

Wrong Solution:



Use the TM for A_2 to decide if $f(w) \in A_2$

Log-Space reductions

Claim: if

1. $A_1 \leq_L A_2$ – f is the log-space reduction
2. $A_2 \in L$ – M is a log-space machine for A_2

Then, A_1 is in L

Proof: on input x, in or not-in A_1:

Simulate M and

whenever M reads the i^{th} symbol of its input tape

run f on x and wait for the i^{th} bit to be outputted

NL Completeness

Definition:

A language B is NL-Complete if

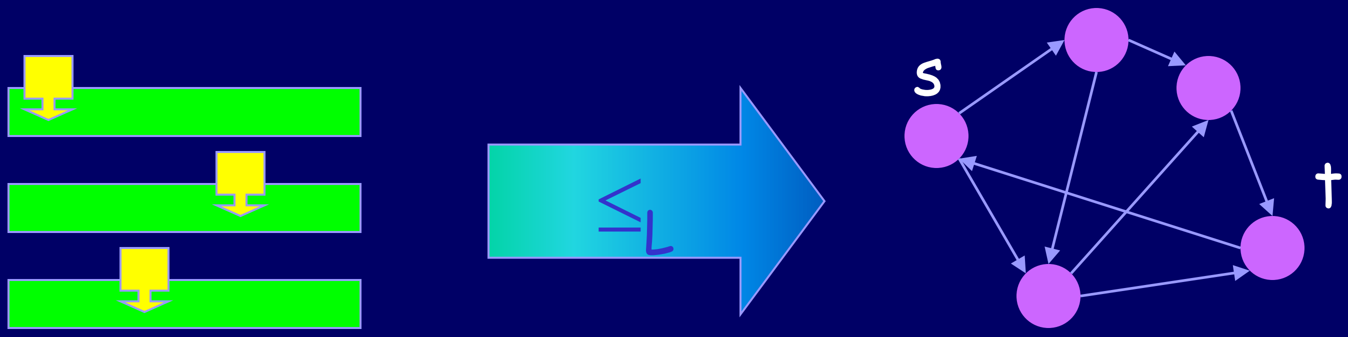
1. $B \in NL$
2. For every $A \in NL$, $A \leq_L B$.

If (2) holds, B is NL-hard

CONN is NL-Complete

Theorem: CONN is NL-Complete

Proof: by the following reduction:



"Does M accept x?"

"Is there a path from s to t?"

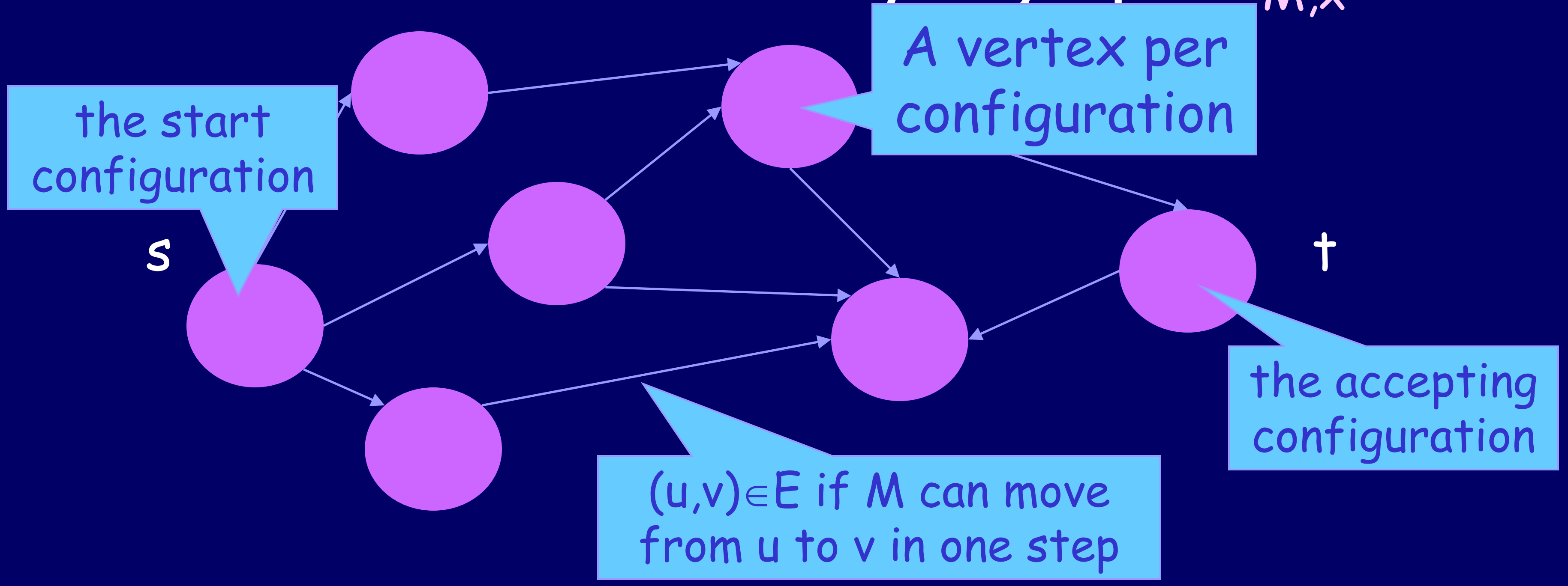
Technicality

<u>Observation:</u>

Without loss of generality, we can assume all NTM's have exactly one accepting configuration.

Configurations Graph

A Computation of a NTM M on an input x can be described by a graph $G_{M,x}$:



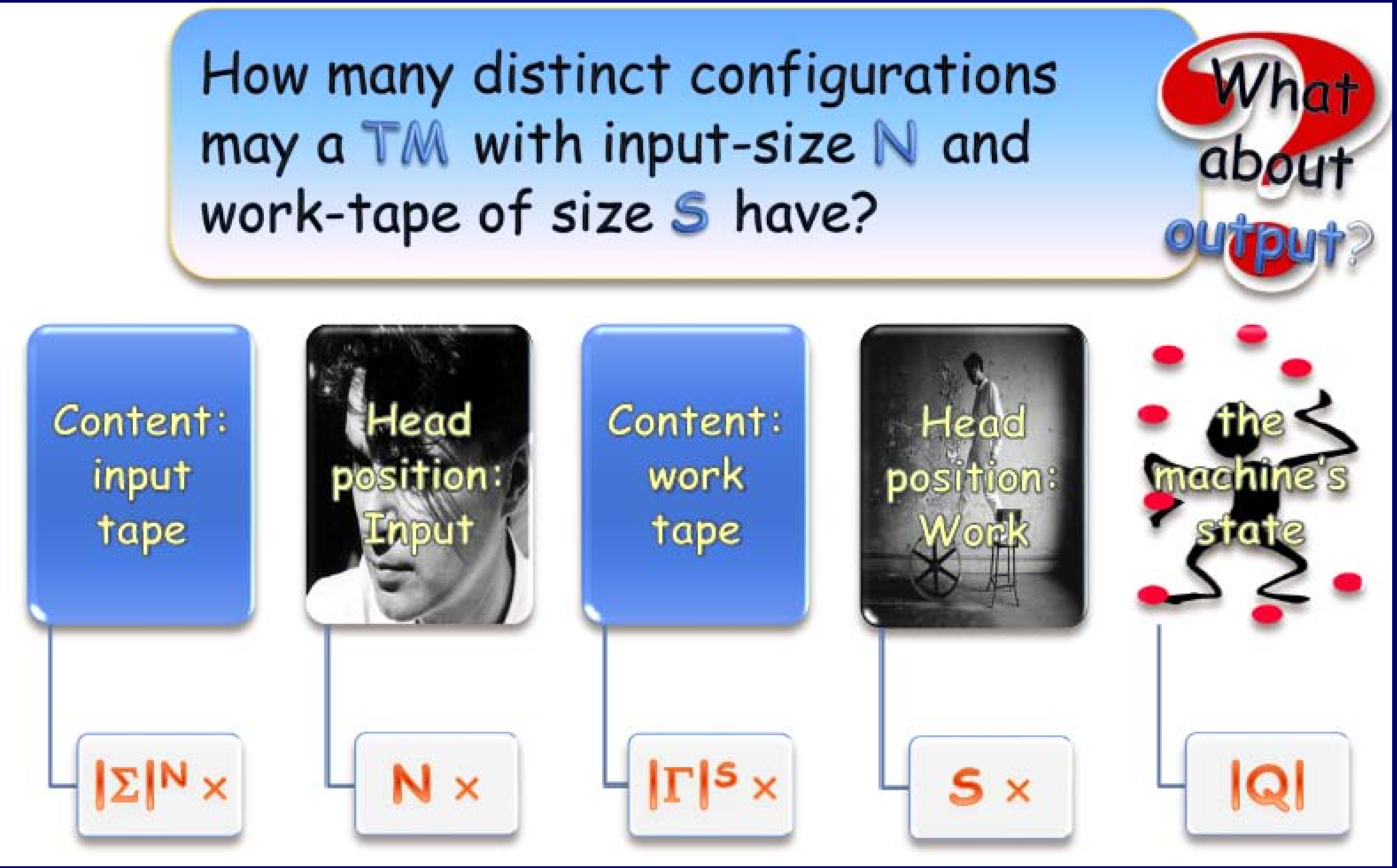
Configurations

Which objects determine the configuration of a TM of the new type?

- The content of the work tape
- The machine's state
- The head position on the input tape
- The head position on the work tape
- The head position on the output tape

If the TM uses logarithmic space, there are polynomially many configurations

Configurations



Correctness

Claim: For every non-deterministic log-space Turing machine M and every input x,

M accepts x iff there is a path from s to t in $G_{M,x}$

CONN is NL-Complete

Corollary: CONN is NL-Complete

Proof: We've shown CONN is in NL. We've also presented a reduction from any NL language to CONN which is computable in log space (Why?) ■

A Byproduct

Claim: $NL \subseteq P$

Proof:

- Any NL language is log-space reducible to CONN
- Thus, any NL language is poly-time reducible to CONN
- CONN is in P
- Thus any NL language is in P. ■

Savitch's Theorem

<u>Theorem:</u>

$\forall S(n) \geq \log(n)$

$NSPACE(S(n)) \subseteq SPACE(S(n)^2)$

<u>Proof:</u>

First we'll prove $NL \subseteq SPACE(\log^2 n)$ then, show this implies the general case

Savitch's Theorem

Theorem:

$$\mathrm{NSPACE}(\log n) \subseteq \mathrm{SPACE}(\log^2 n)$$

Proof:

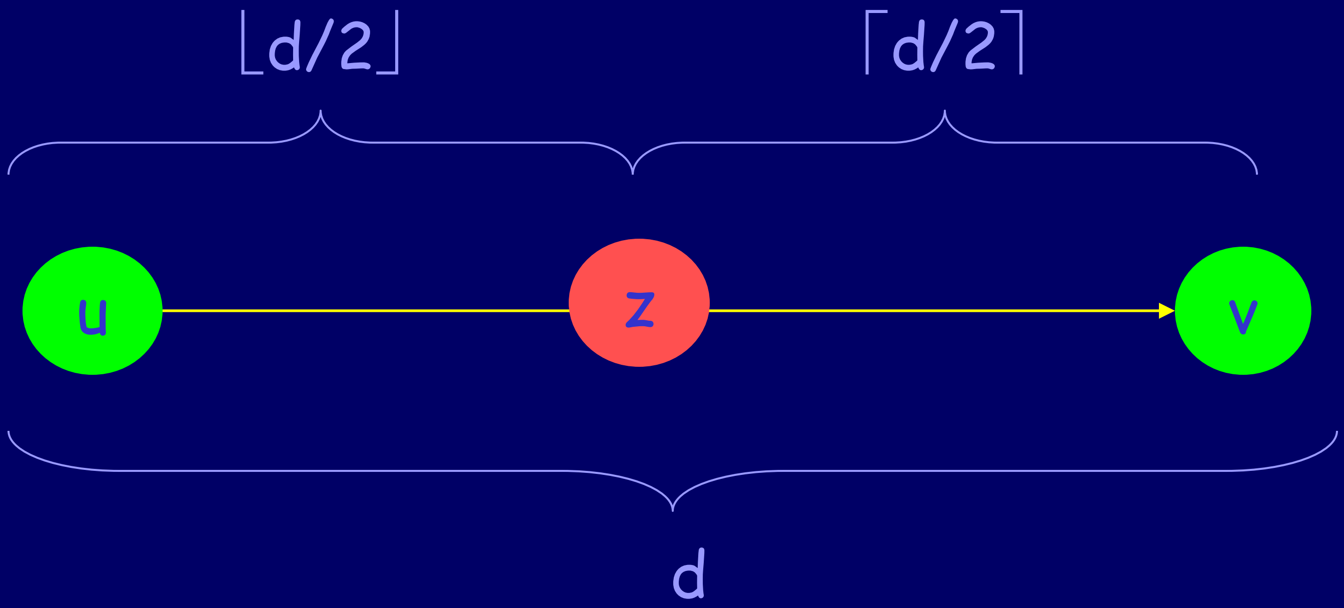
1. First prove CONN is NL-complete (under log-space reductions)
2. Then show an algorithm for CONN that uses $\log^2 n$ space

What Next?

We need to show CONN can be decided by a deterministic TM in $O(\log^2 n)$ space.

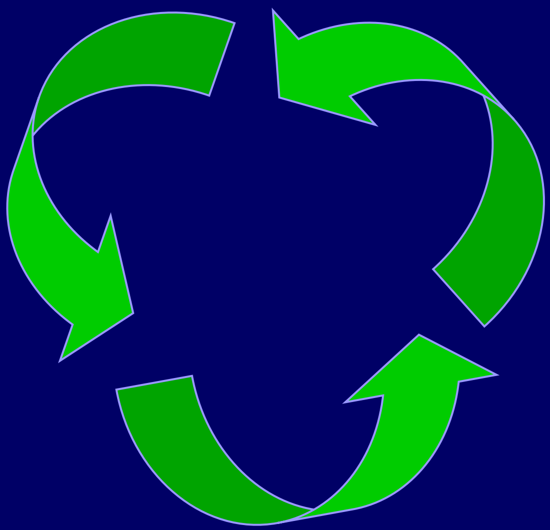
The Trick

"Is there a vertex z, so there is a path from u to z of size $\lfloor d/2 \rfloor$ and one from z to v of size $\lceil d/2 \rceil$?"



Recycling Space

The two recursive invocations can use the same space



The Algorithm

```
Boolean PATH(a,b,d) {
    if there is an edge from a to b then
        return TRUE
    else {
      if d=1 return FALSE
      for every vertex v {
            if PATH(a,v,⌈d/2⌉) and PATH(v,b,⌊d/2⌋)
            then return TRUE
      }
      return FALSE
    }
}
```

$O(\log^2 n)$ Space DTM

Claim: There is a deterministic TM which decides CONN in $O(\log^2 n)$ space.

Proof:

To solve CONN, we invoke PATH(s,t,|V|)

The space complexity:

$S(n)=S(n/2)+O(\log n)=O(\log^2 n)$ ■

Conclusion

<u>Theorem:</u>

$$\mathrm{NSPACE}(\log n) \subseteq \mathrm{SPACE}(\log^2 n)$$

How about the general case $\mathrm{NSPACE}(S(n)) \subseteq \mathrm{SPACE}(S^2(n))$?

The Padding Argument

Motivation: Scaling-Up Complexity Claims

We have:

space $\xrightarrow{\text{can be simulated by...}}$ space

+ non-determinism   + determinism

We want:

space $\xrightarrow{\text{can be simulated by...}}$ space

+ non-determinism   + determinism

Formally

$s_i(n)$ can be computed with space $s_i(n)$

Claim: For any two space constructible functions $s_1(n), s_2(n) \geq \log n$, $f(n) \geq n$:

simulation overhead

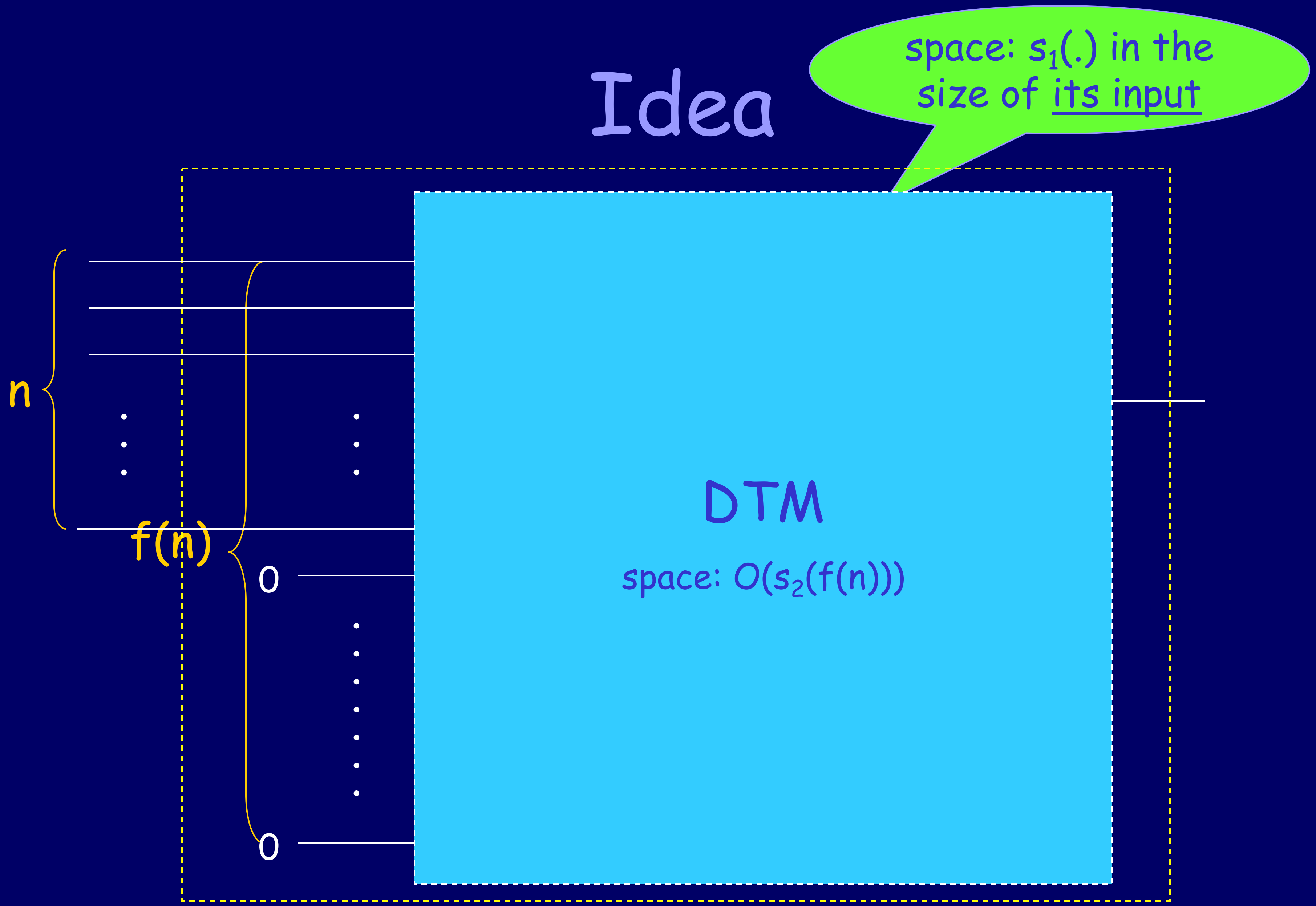
$$NSPACE(s_1(n)) \subseteq SPACE(s_2(n))$$

$$\Downarrow$$

$$NSPACE(s_1(f(n))) \subseteq SPACE(s_2(f(n)))$$

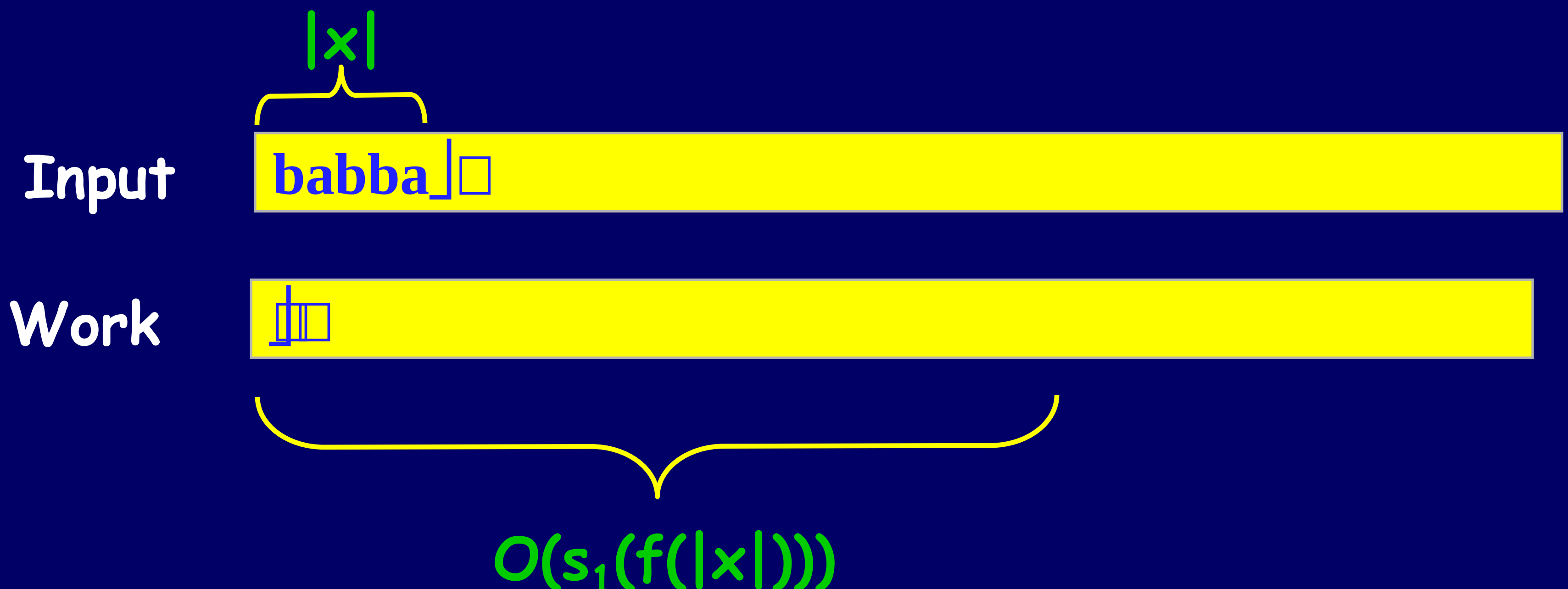
E.g $NSPACE(n) \subseteq SPACE(n^2) \Rightarrow NSPACE(n^2) \subseteq SPACE(n^4)$

Idea



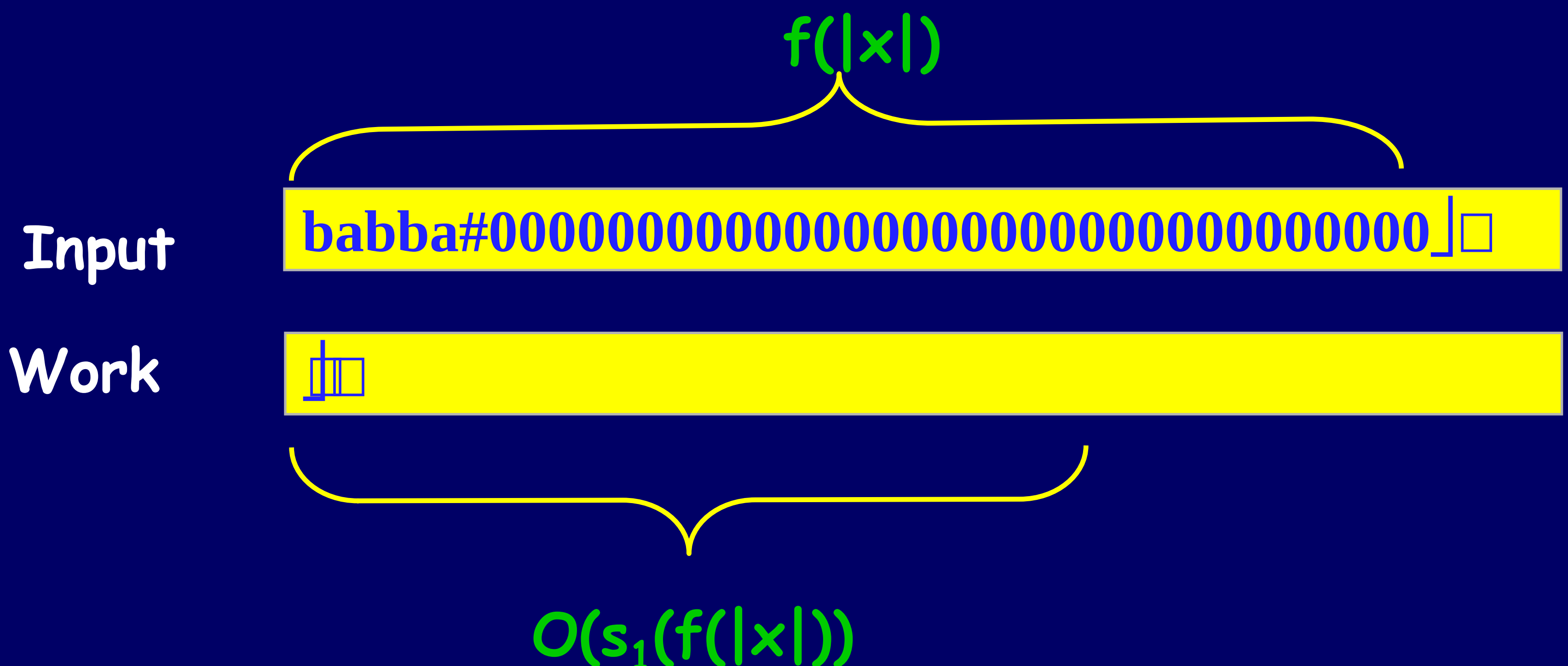
Padding argument

- Let $L \in NPSPACE(s_1(f(n)))$
- There is a 3-Tape-NTM M_L:



Padding argument

- Let $L' = \{\ x0^{f(|x|)-|x|} \mid x \in L\ \}$
- We'll show a NTM $M_{L'}$ which decides L' in the same number of cells as M_L.



Padding argument - $M_{L'}$

In $O(\log(f(|x|))$ space

1. Count backwards the number of 0's and check there are $f(|x|)-|x|$ such.
2. Run M_L on x.

in $O(s_1(f(|x|)))$ space

$f(|x|)$

Input babba00000000000000000000000000000000

Work

$O(s_1(f(|x|)))$

Padding argument

Total space: $O(s_1(f(|x|)))$

$f(|x|)$

Input babba00000000000000000000000000000000⌋□

Work ⌋□□

$O(s_1(f(|x|)))$

Padding Argument

- We started with $L \in NSPACE(s_1(f(n)))$
- We showed: $L' \in NSPACE(s_1(n))$
- Thus, $L' \in SPACE(s_2(n))$
- Using the DTM for L' we'll construct a DTM for L, which will work in $O(s_2(f(n)))$ space.

Padding Argument

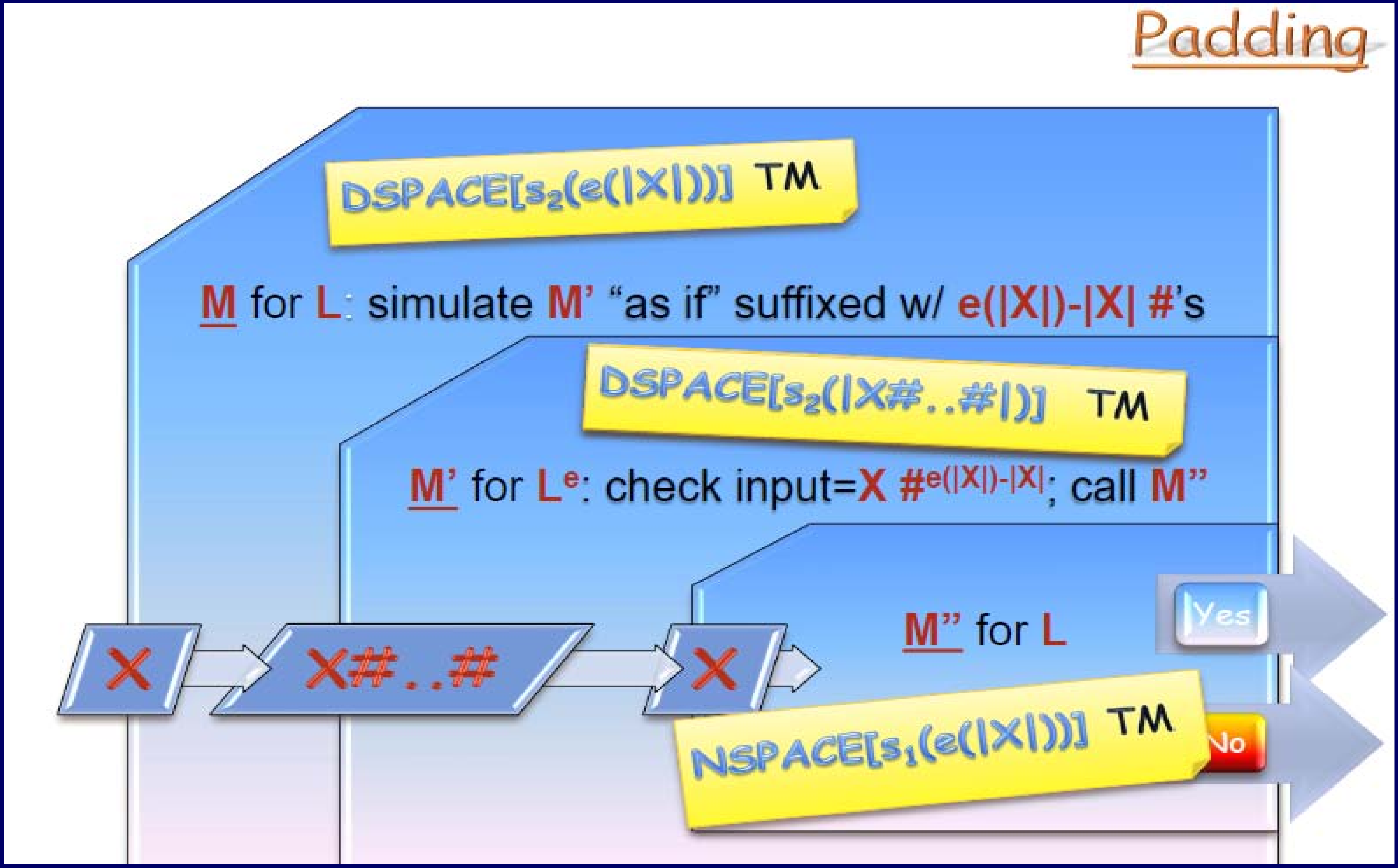
- The DTM for L will simulate the DTM for L′ when working on its input concatenated with zeros

Input

Padding Argument

- When the input head leaves the input part, just pretend it encounters 0s.
- maintaining the simulated position (on the imaginary part of the tape) takes $O(\log(f(|x|)))$ space.
- Thus our machine uses $O(s_2(f(|x|)))$ space.
- $\Rightarrow$ $NSPACE(s_1(f(n)))\subseteq SPACE(s_2(f(n)))$

Padding Argument



Savitch: Generalized Version

Theorem (Savitch):

$\forall S(n) \geq \log(n)$

$$NSPACE(S(n)) \subseteq SPACE(S(n)^2)$$

Proof: We proved $NL \subseteq SPACE(\log^2 n)$. The theorem follows from the padding argument. ■

Corollary

<u>Corollary:</u> PSPACE = NPSPACE

<u>Proof:</u>

Clearly, PSPACE$\subseteq$NPSPACE.

By Savitch's theorem, NPSPACE$\subseteq$PSPACE. ■

Space Vs. Time

- We've seen space complexity probably doesn't resemble time complexity:
  - Non-determinism doesn't decrease the space complexity drastically (Savitch's theorem).
- We'll next see another difference:
  - Non-deterministic space complexity classes are closed under completion (Immerman's theorem).

NON-CONN

NON-CONN

- Instance: A directed graph G and two vertices $s,t \in V$.
- Problem: To decide if there is no path from s to t.

NON-CONN

- Clearly, NON-CONN is coNL-Complete.

(Because CONN is NL-Complete. See the coNP lecture)

- If we'll show it is also in NL, then NL=coNL.

(Again, see the coNP lecture)

An Algorithm for NON-CONN

We'll see a log space algorithm for counting reachability

1. Count how many vertices are reachable from s.
2. Take out t and count again.
3. Accept if the two numbers are the same.

N.D. Algorithm for $reach_s(v, l)$

$reach_s(v, l)$

1. length = l; u = s

2. **while** (length > 0) {
   3. **if** u = v return 'YES'
   4. **else, for all** (u' $\in$ V) {
      5. **if** (u, u') $\in$E **nondeterministic switch**:
         5.1 u = u'; --length; **break**
         5.2 **continue**
   }
}
6. return 'NO'

Takes up logarithmic space

This N.D. algorithm might never stop

N.D. Algorithm for CR_s

CR_s (d)

1. count = 0
2. **for all** $u \in V$ {
   3. $count_{d-1} = 0$
   4. **for all** $v \in V$ {
      5. **nondeterministic switch**:
         5.1 **if** reach(v, d - 1) then $++count_{d-1}$ <u>**else** *fail*</u>
         if $(v,u) \in E$ then ++count; **break**
         5.2 **continue**

   }

   6. **if** $count_{d-1} < CR_s\,(d-1)$ fail

}

7. return count

Assume $(v,v) \in E$

Recursive call!

N.D. Algorithm for CR_s

CR_s (d $\boxed{, C)}$

1. count = 0
2. for all $u \in V$ {
   3. $count_{d-1} = 0$
   4. for all $v \in V$ {
      5. nondeterministic switch:
         5.1 if reach(v, d - 1) then $++count_{d-1}$ <u>else *fail*</u>
         if $(v,u) \in E$ then ++count; break
         5.2 continue
   }
   6. if $count_{d-1} < \boxed{C}$ fail
}
7.return count

Main Algorithm:

CR_s

C = 1

for d = 1..|V|

C = CR(d, C)

return C

parameter

Efficiency

<u>Lemma</u>: The algorithm uses $O(\log(n))$ space.

<u>Proof:</u>

There is a constant number of variables (d, count, u, v, $count_{d-1}$). Each requires $O(\log(n))$ space (range $|V|$). ■

Immerman's Theorem

Theorem[Immerman/Szelepcsenyi]: NL=coNL

Proof:

(1) NON-CONN is NL-Complete

(2) NON-CONN$\in$NL

Hence, NL=coNL. ■

Corollary

<u>Corollary:</u>

$\forall s(n) \geq \log(n)$,

$NSPACE(s(n)) = coNSPACE(s(n))$

<u>Proof:</u> By a padding argument.

TQBF

- We can use the insight of Savich's proof to show a language which is complete for PSPACE.
- We present TQBF, which is the quantified version of SAT.

TQBF

- Instance: a fully quantified Boolean formula ϕ
- Problem: to decide if ϕ is true

Example: a fully quantified Boolean formula

$$\forall x \exists y \forall z [(x \vee \neg y \vee z) \wedge (\neg x \vee y)]$$

Variables` range is $\{0,1\}$

TQBF is in PSPACE

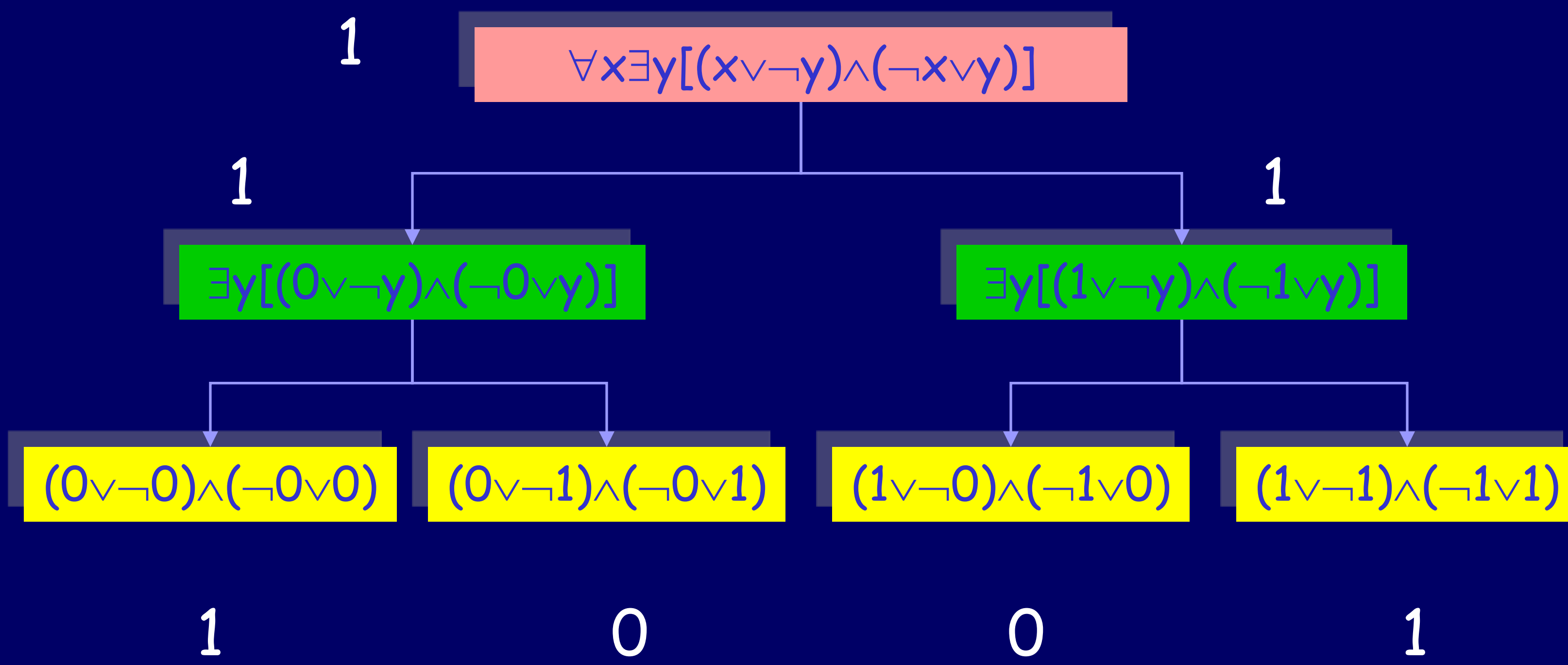
Theorem: TQBF$\in$PSPACE

Proof: We'll describe a poly-space algorithm A for evaluating ϕ:

in poly time

- If ϕ has no quantifiers: evaluate it
- If $\phi=\forall x(\psi(x))$ call A on $\psi(0)$ and on $\psi(1)$; Accept if *both* are true.
- If $\phi=\exists x(\psi(x))$ call A on $\psi(0)$ and on $\psi(1)$; Accept if *either* is true.

Algorithm for TQBF



Efficiency

- Since both recursive calls use the same space,
- the total space needed is polynomial in the number of variables (the depth of the recursion)

$\Rightarrow$ TQBF is polynomial-space decidable ■

PSAPCE Completeness

Definition:

A language B is PSPACE-Complete if

1. $B \in PSPACE$
2. For every $A \in PSAPCE$, $A \leq_p B$.

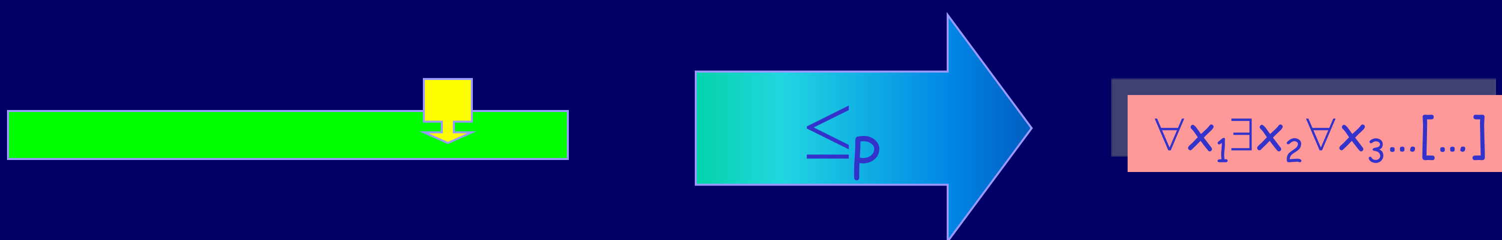
standard Karp reduction

If (2) holds, then B is PSPACE-hard

TQBF is PSPACE-Complete

Theorem: TQBF is PSAPCE-Complete

Proof: It remains to show TQBF is PSAPCE-hard:



"Will the poly-space M accept x?"

"Is the formula true?"

TQBF is PSPACE-Hard

Given a TM M for a language $L \in$ PSPACE, and an input x, let $f_{M,x}(u, v)$, for any two configurations u and v, be the function evaluating to TRUE iff M on input x moves from configuration u to configuration v

$f_{M,x}(u, v)$ is efficiently computable

Formulating Connectivity

The following formula, over variables $u,v \in V$ and path's length d, is TRUE iff G has a path from u to v of length $\leq d$

$\phi(u,v,1) \equiv f_{M,x}(u, v) \vee u=v$

$\phi(u,v,d) \equiv$

$\exists w \forall x \forall y[((x=u \wedge y=w) \vee (x=w \wedge y=v)) \rightarrow \phi(x,y,d/2)]$

w is reachable from u in $\lceil d/2 \rceil$ steps. v is reachable from w in $\lceil d/2 \rceil$ steps.

simulates AND of $\phi(u,w,d/2)$ and $\phi(w,v,d/2)$

TQBF is PSPACE-Complete

Claim: TQBF is PSPACE-Complete

Proof:

$\phi \equiv \phi(s,t,|V|)$ is TRUE iff there is a path from s to t.

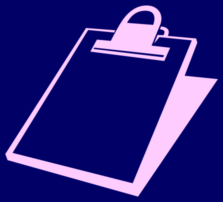
ϕ is constructible in poly-time.

Thus, any PSPACE language is poly-time reducible to TQBF, i.e – TQBF is PSAPCE-hard.

Since TQBF$\in$PSPACE, it's PSAPCE-Complete.

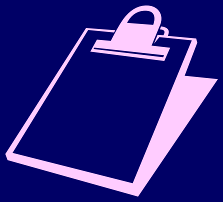
■

Summary



- We introduced a new way to classify problems: according to the space needed for their computation.
- We defined several complexity classes: L, NL, PSPACE.

Summary



- Our main results were:
  - Connectivity is NL-Complete
  - TQBF is PSPACE-Complete

    By reducing decidability to reachability
  - Savitch's theorem ($NL \subseteq SPACE(\log^2)$)
  - The padding argument (extending results for space complexity)
  - Immerman's theorem ($NL=coNL$)