PSPACE et applications

Frédéric Giroire

Jcalm – 10 mars 2015

Un plan

I. COMPLEXITÉ EN ESPACE.
   Au prix du m^2 sur la côte, ça peut être utile.

II. PSPACE et problèmes PSPACE-complets classiques

III. PSPACE et JEUX : où un surfeur réunionnais inconscient doit être sauvé.

Un plan

I. COMPLEXITÉ EN ESPACE.

- **Définitions.**
- Low space Classes: L and NL.
- Connectivity.
- Théorème de Savitch
- Théorème d'Immerman

Motivation

Complexity classes correspond to bounds on resources

We have seen: Time

Now Space: number of tape cells a Turing Machine uses when solving a problem

Space Complexity Classes

For any function $f: N \rightarrow N$, 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$\}$

Example: Low (logarithmic) space classes

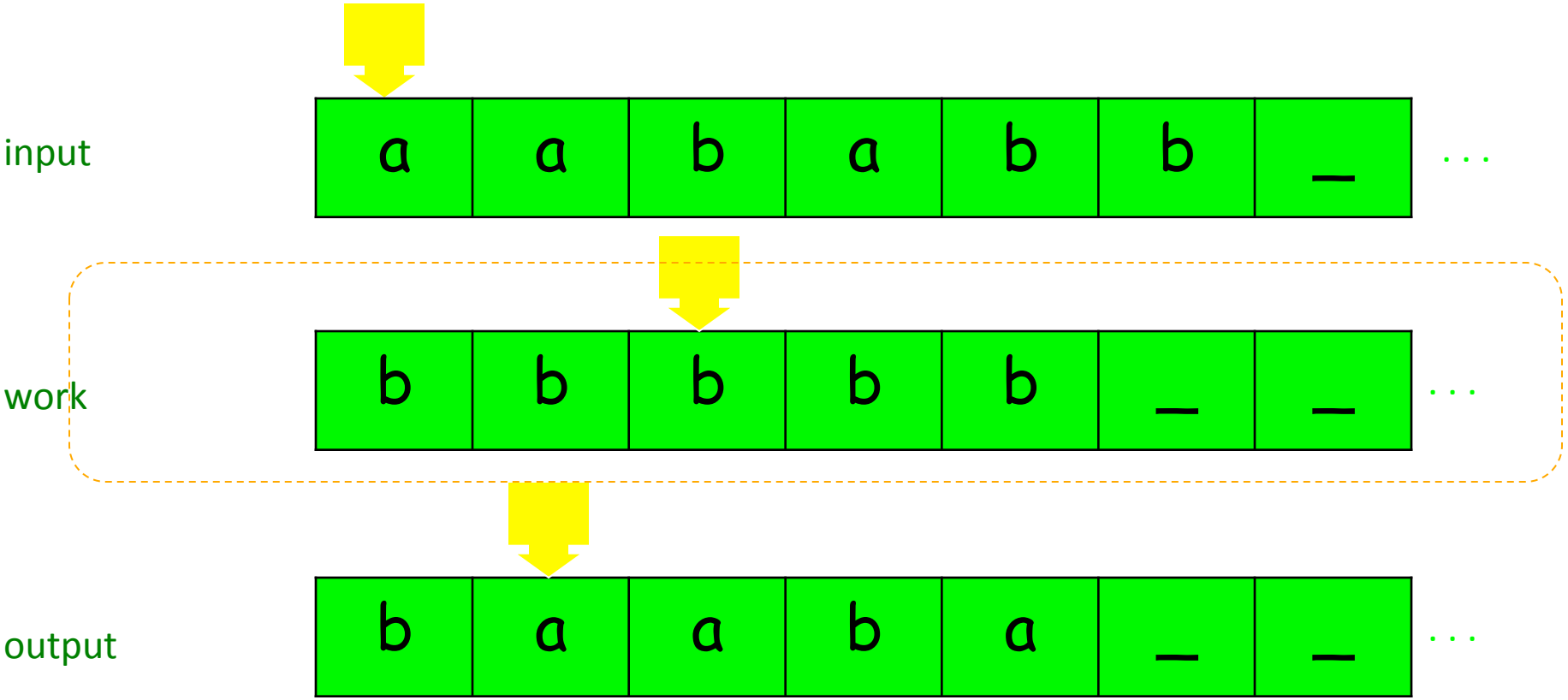
- $L = SPACE(\log n)$
- $NL = NSPACE(\log n)$

Problem!

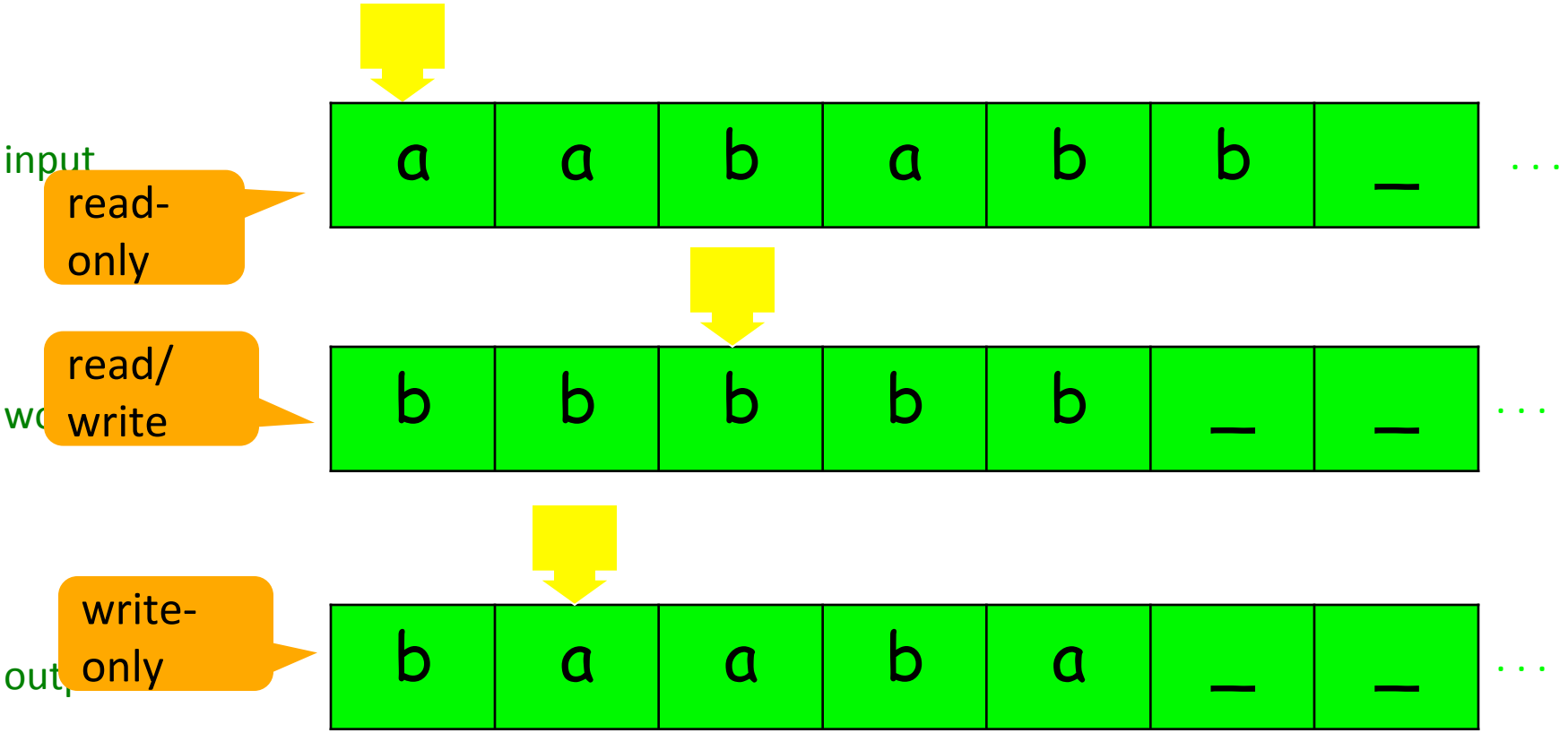
How can a TM use only $\log n$ space if the input itself takes n cells?!

!?

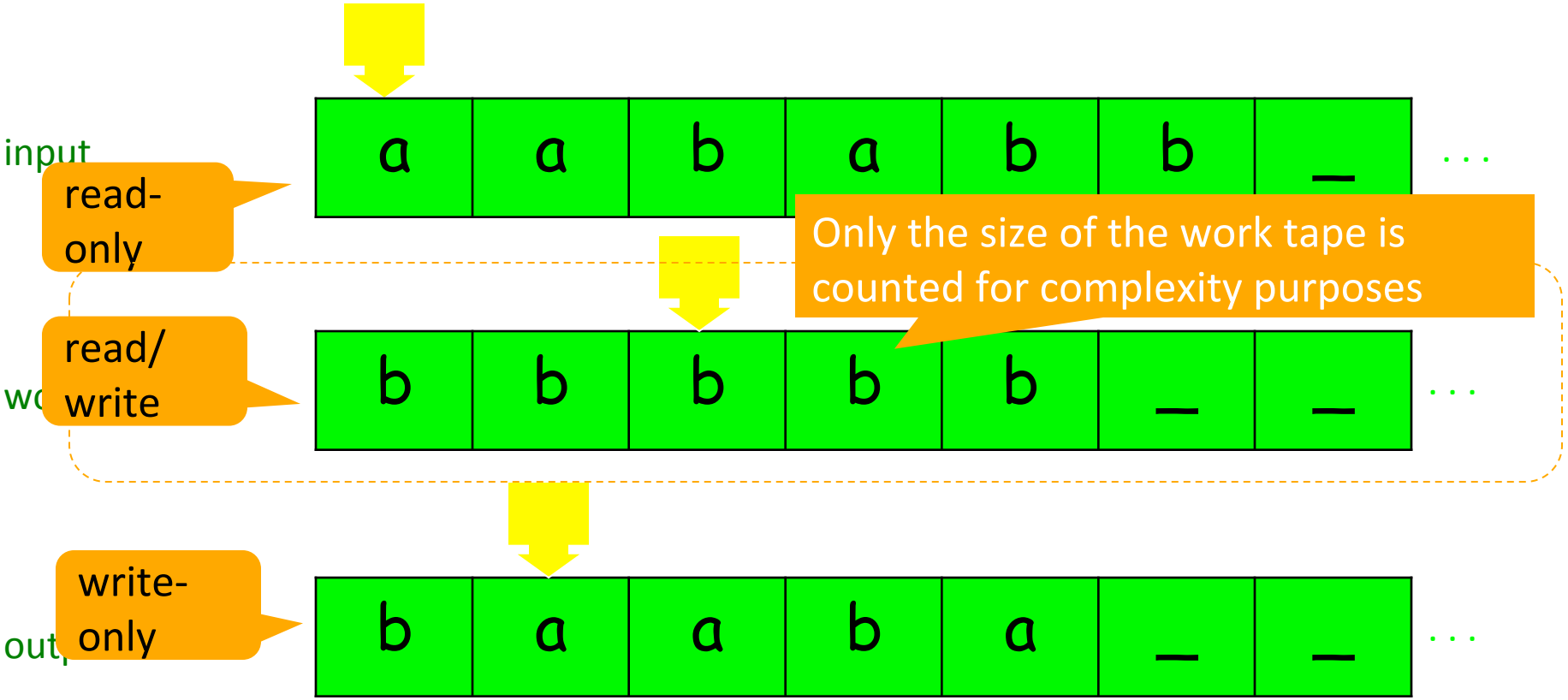
3Tape Machines



3Tape Machines

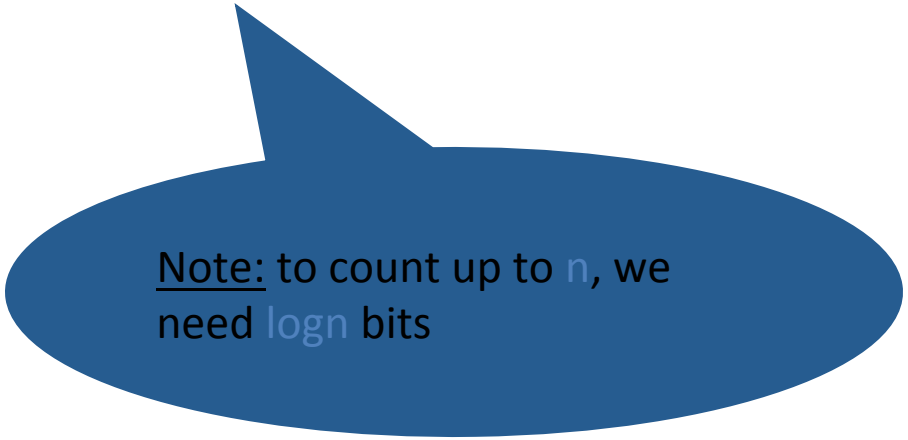


3Tape Machines



Example

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

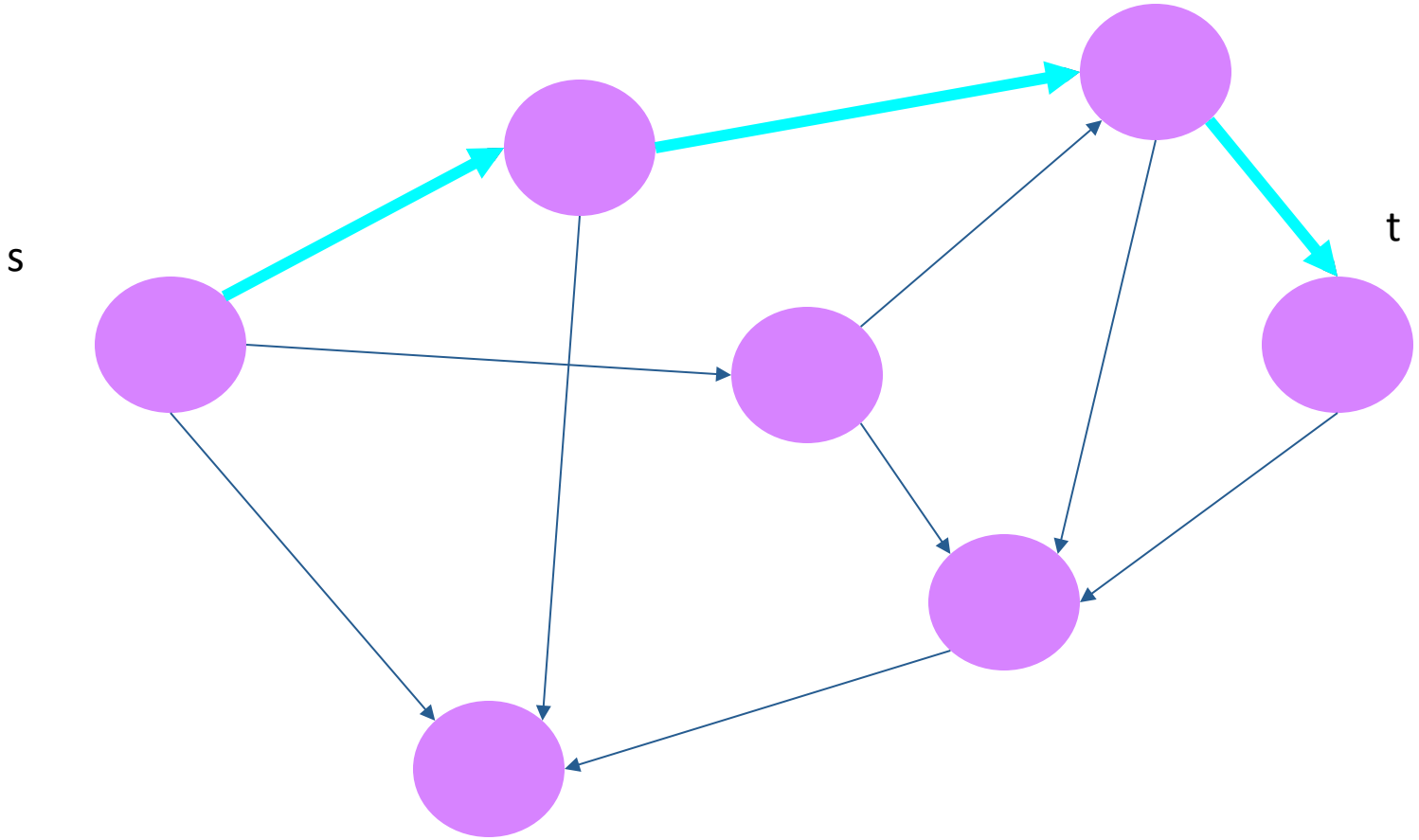


Graph Connectivity

CONN (REACH)

- Instance: a directed graph $G=(V,E)$ and two vertices $s,t \in V$
- Problem: 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

Savitch's Theorem

Theorem:

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

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

Proof:

First we'll prove $NL \subseteq SPACE(\log^2 n)$

then, show this implies the general case

Savitch's Theorem

Theorem:

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

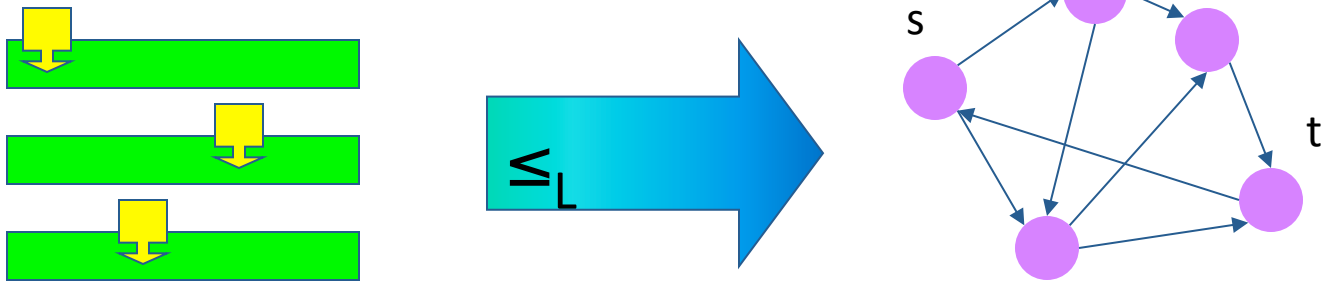
Proof:

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

CONN is NL-Complete

Theorem: CONN is NL-Complete

Proof: by the following reduction:

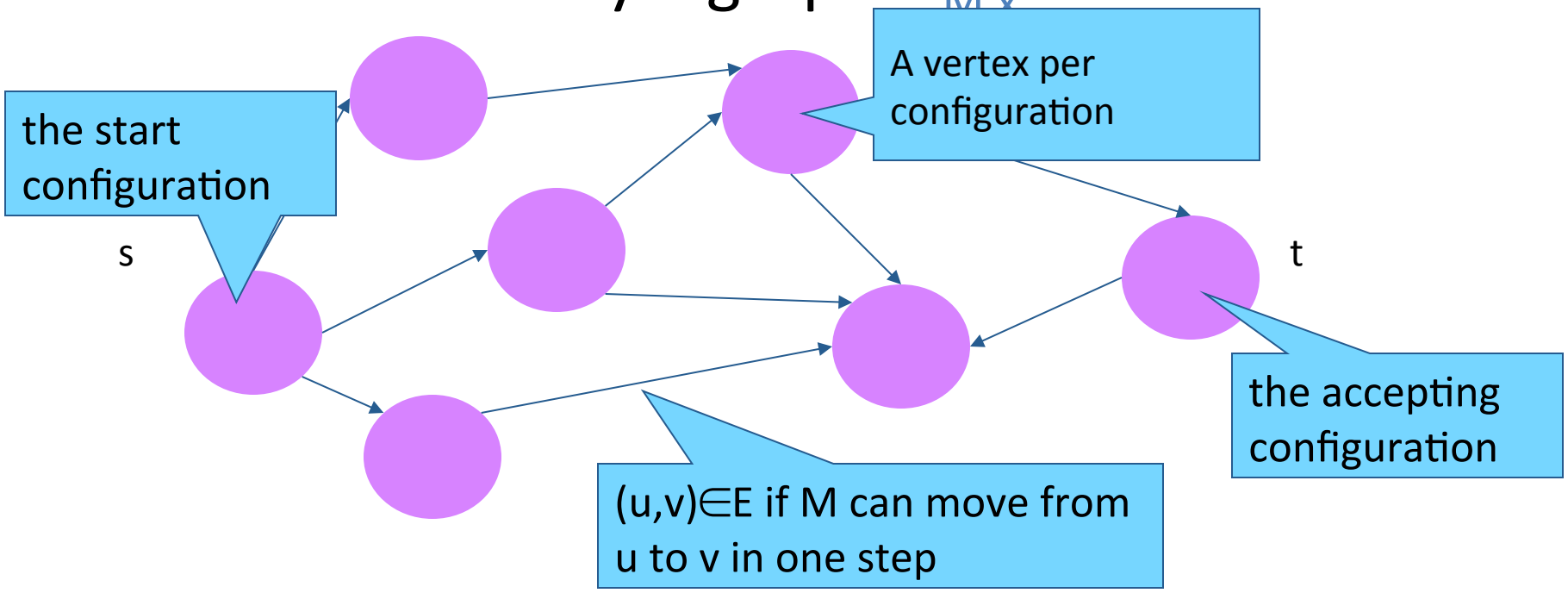


"Does M accept x?"

"Is there a path from s to t?"

Configuration Graph

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



CONN is NL-Complete

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 $G_{M,x}$

Corollary: CONN is NL-Complete

Proof:

- We've shown CONN is in NL.
- Reduction from any NL language to CONN which is computable in log space ■

A Byproduct

Claim: NL⊆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. ■

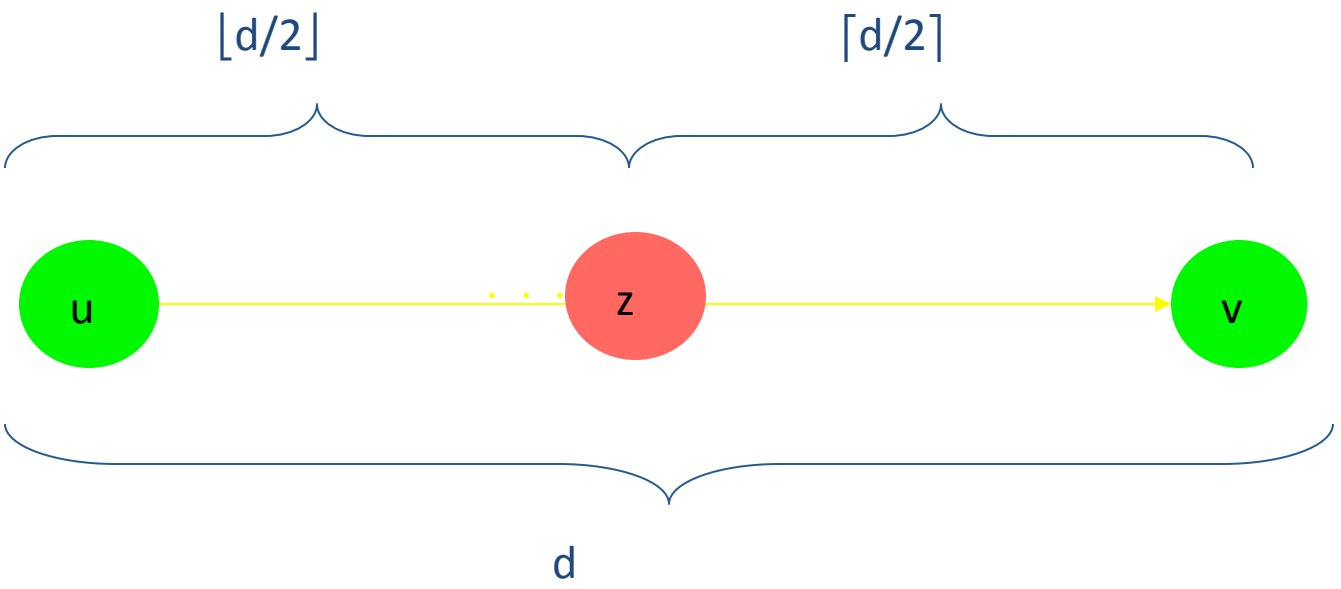
What Next?

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

The Trick

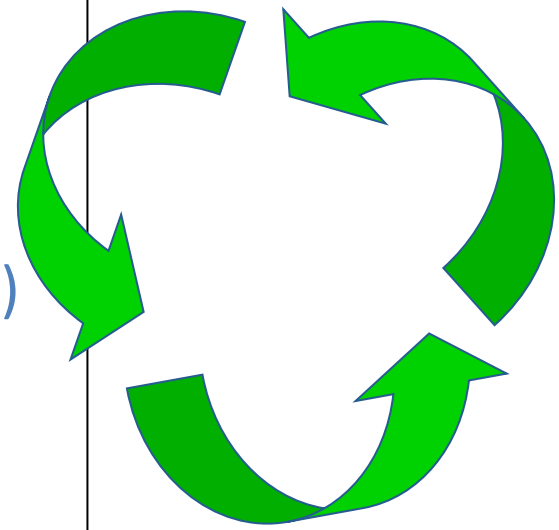
"Is there a path from u to v of length d?"

"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 $\lceil d/2 \rceil$?"



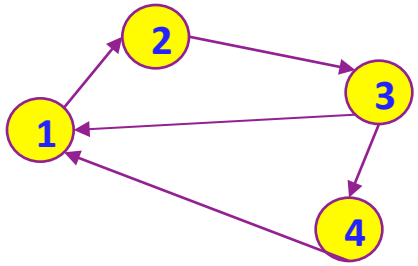
The Algorithm

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



The two recursive invocations can use the same space

Example of Savitch's 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 (not a,b)  {
         if PATH(a,v, ⌈d/2⌉) and
            PATH(v,b, ⌊d/2⌋) then
            return TRUE
      }
      return FALSE
   }
}
```

(a,b,c)=Is there a path from a to b, that takes no more than c steps.

(1,4,3) TRUE

$3\text{Log}_2(d)$

$O(\log^2 n)$ 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

Theorem:

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

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

The Padding Argument

Motivation: Scaling-Up Complexity Claims

We have:

+ non-determinism

+ determinism

We want:

+ non-determinism

+ determinism

Formally

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

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

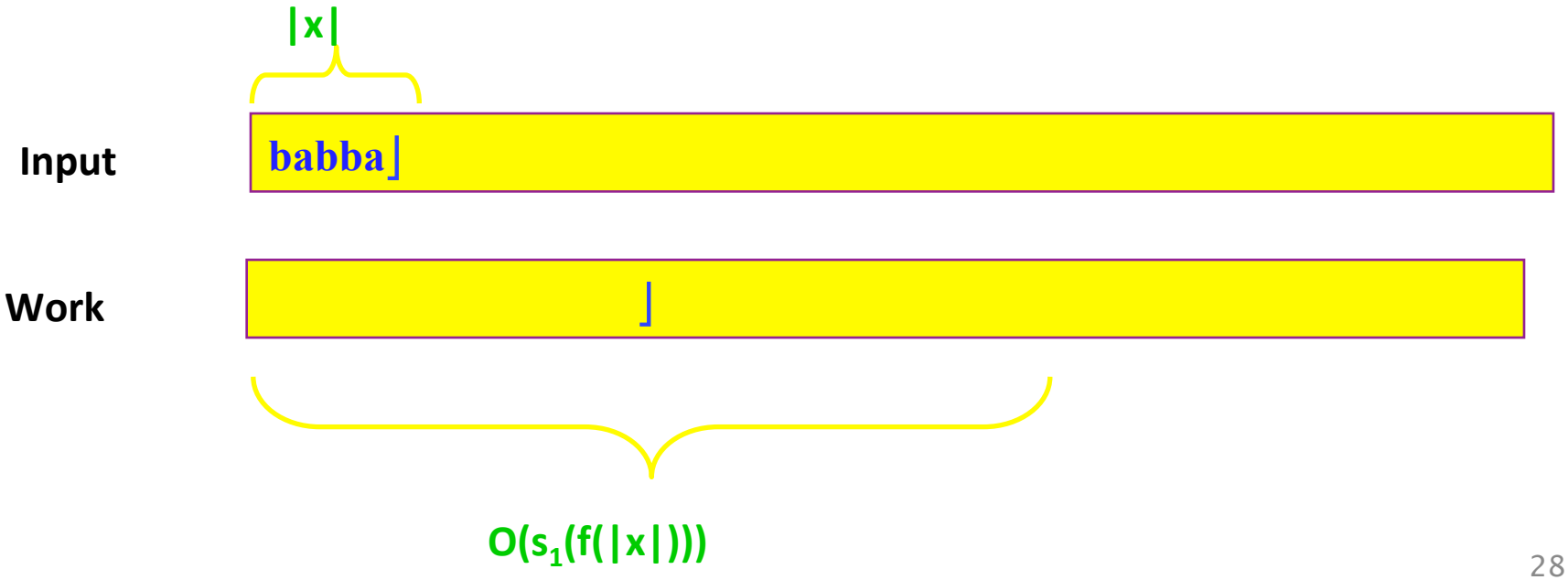
$$\Downarrow$$

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

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

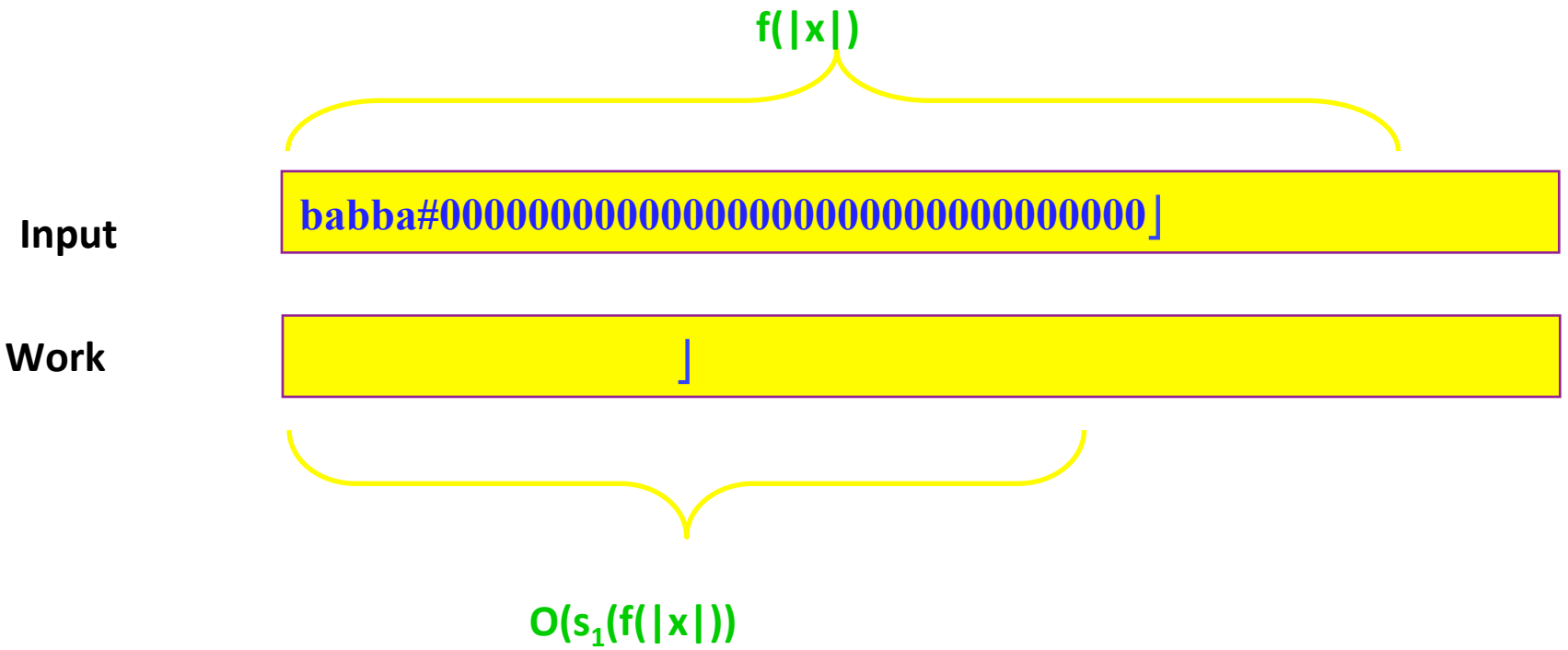
Padding argument

- 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.

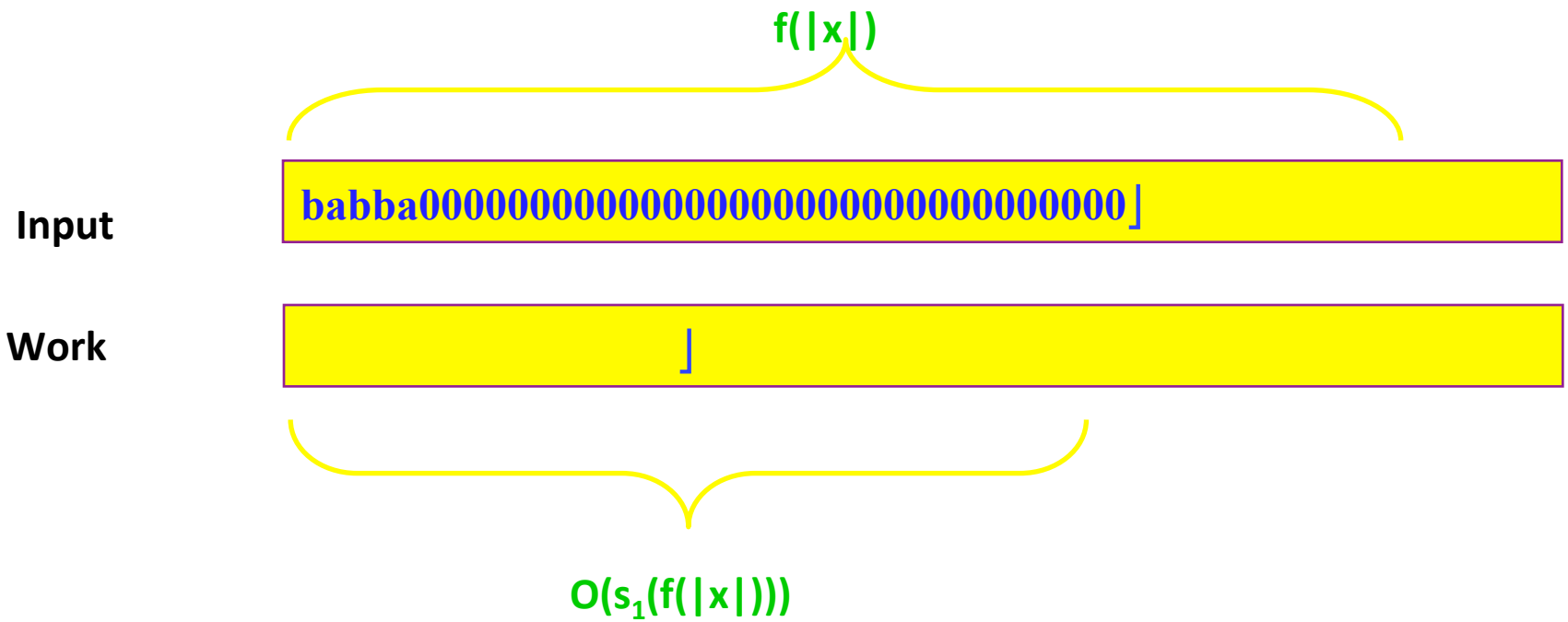


Padding argument – M_L'

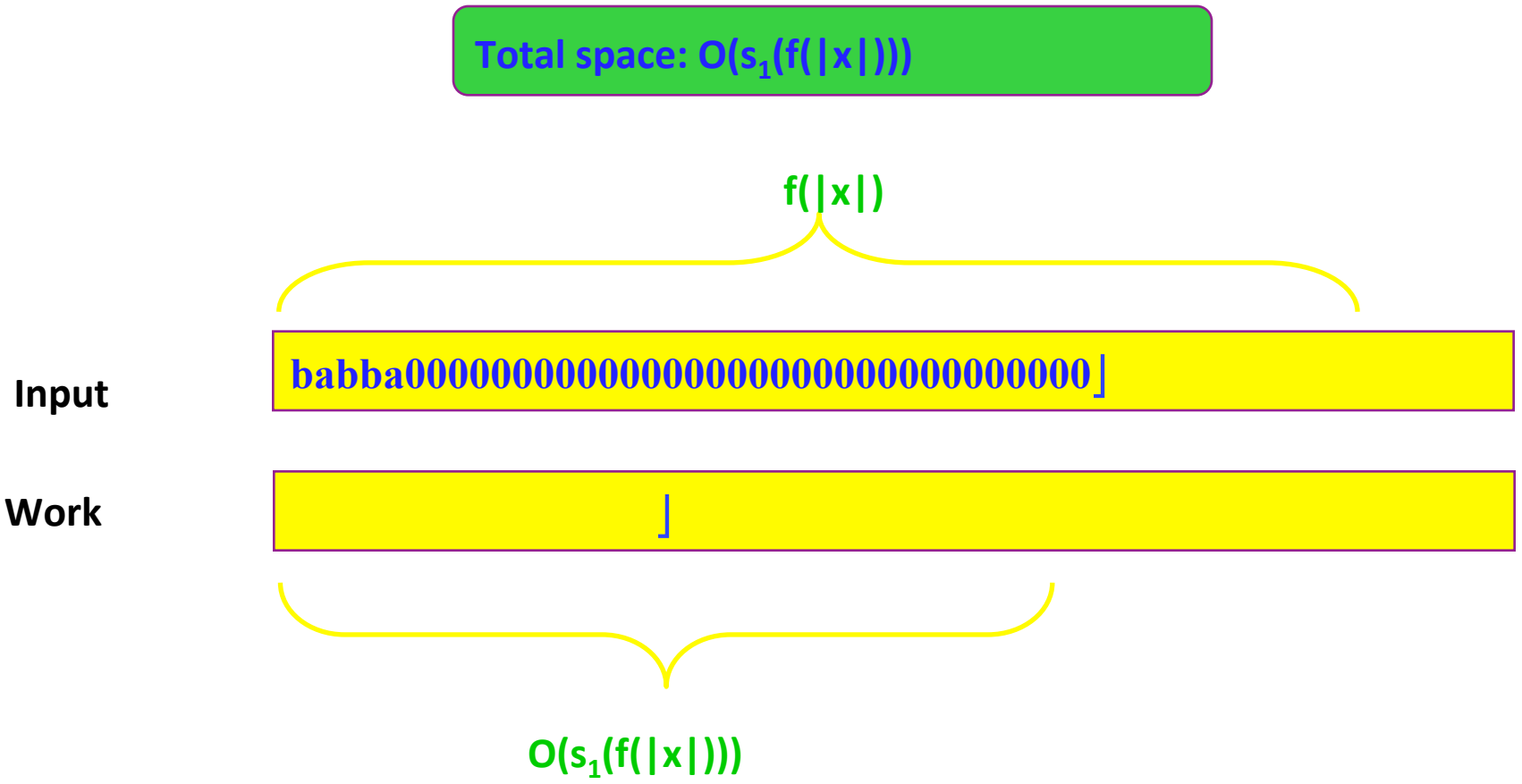
In O(log(f(|x|)) space

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



Padding argument

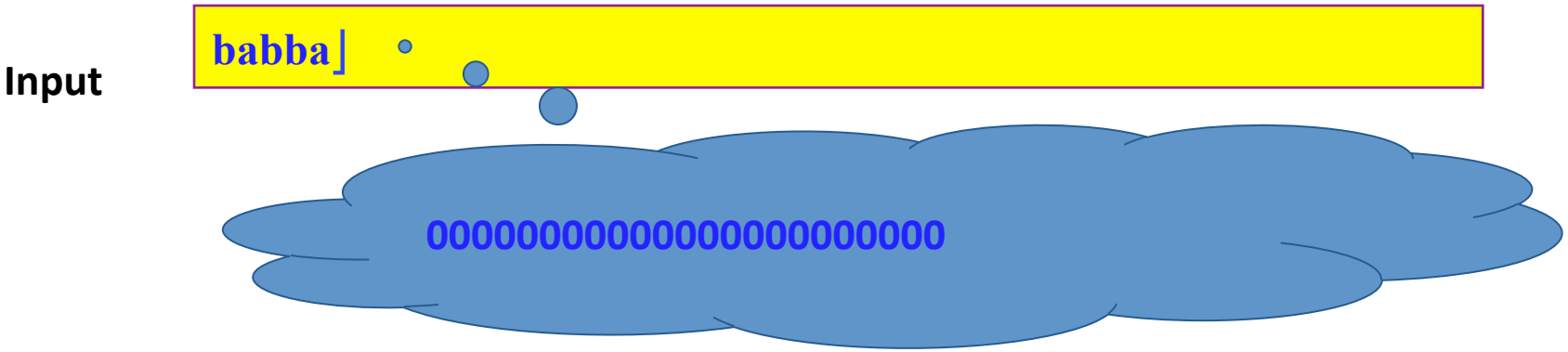


Padding Argument

- 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



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.
- $\Rightarrow \mathrm{NSPACE}(s_1(f(n))) \subseteq \mathrm{SPACE}(s_2(f(n)))$

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

Corollary: PSPACE = NPSPACE

Proof:

Clearly, PSPACE⊆NPSPACE.

By Savitch's theorem, NPSPACE⊆PSPACE. ■

Important differences in terms of complexity:

- **Non-determinism** does not decrease the space complexity drastically (Savitch's theorem).

Immerman's Theorem

Non-deterministic space complexity classes are **closed under completion** (Immerman's theorem).

Theorem[Immerman/Szelepcsenyi]: NL=coNL

Proof:

(1) NON-CONN is NL-Complete

(2) NON-CONN$\in$NL

Hence, NL=coNL. ■

Corollary

Corollary:

$\forall s(n) \geq \log(n)$, $NSPACE(s(n)) = coNSPACE(s(n))$

Proof: By a padding argument.

Take Aways

I. COMPLEXITÉ EN ESPACE.

- **Low space Classes:** L and NL.
- **Connectivity is NL-complete.**
- ***NL* ⊆ *P***
- **Théorème de Savitch** et
  - **NL ⊆ SPACE($\log^2$)**
  - **PSPACE = NPSPACE** (padding argument)
- **Théorème d'Immerman** et

  -> PSPACE = co-PSPACE

Un plan

II. PSPACE et problèmes PSPACE-complets

i. **An introduction**

ii. Quantified satisfability

iii. Machines de Turing alternantes

Geography Game

- **Geography Game**.
  - Alice names capital city *c* of country she is in.
  - Bob names a capital city starting with letter on which *c* ends.
  - Repeat this game until one player is unable to continue.
    Does Alice have a forced win?

- **Ex.** Budapest → Tokyo → Ottawa → Ankara → Amsterdam → Moscow → Washington → Nairobi → ...

- **Geography on graphs**. Given a directed graph *G* = (V, *E)* and a start node *s,* two players alternate turns
  - follow, if possible, an edge out of the current node to an unvisited node.
  - Alternate turns

  Can first player guarantee to make the last legal move?

Geography Games

- **Geography Games**.
- **Ex.** Budapest → Tokyo → Ottawa → Ankara → Amsterdam → Moscow → Washington → Nairobi → ...

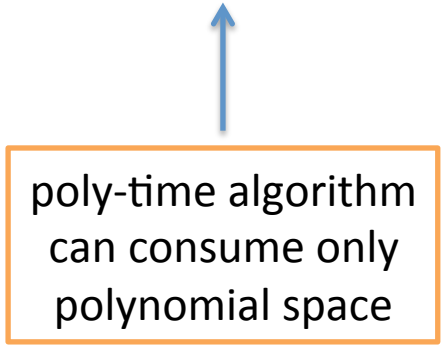
- **Complexity:**
  - P? NP? EXPTIME? NP-complete?
  - In EXPTIME, but uses a polynomial space.

- **Remark**. Some problems (especially involving 2-player games and AI) **defy classification** according to *NP, EXPTIME, NP,* and *NP-Complete.*

PSPACE

- **P** Decision problems solvable in polynomial time.
- **PSPACE** Decision problems solvable in polynomial space.

- **Observation.** ***P*** $\subseteq$ ***PSPACE***.



PSPACE

- Binary counter. Count from 0 to 2^{n-1} in binary.
  Algorithm: Use *n* bit odometer.

- Claim. 3-SAT $\in$ *PSPACE.* Pf.
  - Enumerate all 2^n possible truth assignments using counter.
  - Check each assignment to see if it satisfies all clauses.

- **Theorem. *NP* $\subseteq$ *PSPACE.***
  Pf. Consider arbitrary problem $Y \in NP$.
  - Since $Y \leq_P$ 3-SAT, there exists algorithm that solves *Y* in poly-time plus polynomial number of calls to 3-SAT black box.
  - Can implement black box in poly-space. ▪

Remarques préliminaires

Remark: ***PSPACE*** $\subseteq$ **EXP.**

Preuve:

- Une machine de Turing utilisant espace P(n) n'a que $O(2^{P(n)})$ configurations possibles.
  - Textes possibles sur le ruban de travail + position
- Ensuite boucle.

Complexity Class Hierarchy

- **Summary.** ***P*** $\subseteq$ ***NP*** $\subseteq$ ***PSPACE*** $\subseteq$ ***EXPTIME***.

- it is known that P $\neq$ EXPTIME,
- but unknown which inclusion is strict; conjectured that all are.

Un plan

II. PSPACE et problèmes PSPACE-complets
    i. An introduction
    ii. Quantified satisfability
    iii. Machines de Turing alternantes

Quantified Satisfiability

QSAT problem (ou TQBF ou QBF for True quantified Boolean formula)

Input:

A set of boolean variables. $\{x_1, ..., x_n\}$

A logical formula $\Phi(x_1, ..., x_n)$.

Question:

Is the following propositional formula true?

$$\exists\, x_1\ \forall\, x_2\ \exists\, x_3\ \forall\, x_4 \ldots \forall\, x_{n-1}\ \exists\, x_n\ \Phi(x_1, ..., x_n)$$

Remark: canonical forms

- Φ in **conjonctive normal form** (conjunction of clauses)
- **alternating** quantificators

Quantified Satisfiability

Intuition relationship QSAT and Games. Amy picks truth value for x_1, then Bob for x_2, then Amy for x_3, and so on.
Can Amy satisfy Φ (∃) whatever Bob does (∀)?

Ex. $(x_1 \vee x_2) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3)$

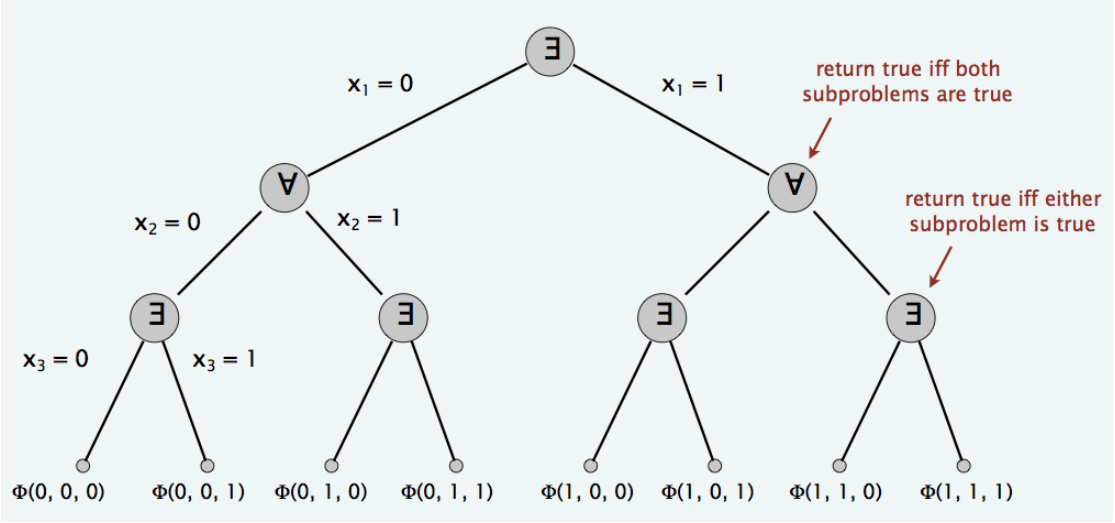
Yes. Amy sets *x1* true; Bob sets *x2;* Amy sets *x3* to be same as *x2.*

Ex. $(x_1 \vee x_2) \wedge (x_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3)$

No. If Amy sets *x1* false; Bob sets *x2* false; Amy loses;
No. if Amy sets *x1* true; Bob sets *x2* true; Amy loses.

Quantified Satisfiability is in PSPACE

Theorem. Q-SAT ∈ *PSPACE*



Pf. Recursively try all possibilities.

- Only need one bit of information from each subproblem.
- Amount of space proportional to depth of function call stack.

PSPACE Completeness

Definition:

A language B is **PSPACE-Complete** if

1. $B \in PSPACE$
2. For every $A \in SPACE$, $A \leq_P B$.

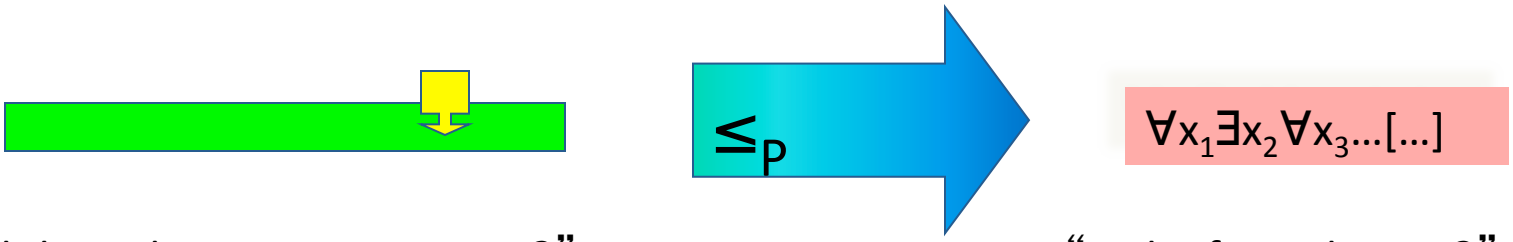
standard Karp reduction

TQBF is PSPACE-Complete

Theorem. Q-SAT is PSPACE-Complete

Proof: It remains to show TQBF is PSPACE-hard.

essentially a more technical restatement of Savitch's theorem proof.



"Will the poly-space M accept x?"

"Is the formula true?"

"Is there a path between two configurations of M?"

Un plan

II. PSPACE et problèmes PSPACE-complets
    i. An introduction
    ii. Quantified satisfability
    iii. Machines de Turing alternantes

Machine de Turing Alternantes - Definitions

- **Machine de Turing alternante** : espace des etats non finaux partitionné
  - en **etats existentiels** (MT fonctionne comme une machine non déterministe qui va deviner l'état suivant pour tenter d'atteindre un état acceptant) et
  - **les etats universels** (MT va vérifier que quel que soit le choix de l'état suivant, elle peut atteindre un état acceptant)

Classes de complexité alternante :

- ATIME(f(n)) la classe des languages en temps f(n) sur machine alternante.

- AP (temps polynomial alternant)

Machine de Turing Alternantes – Résultat principal

- **Theorem PSPACE = AP.**
  [Chandra, Kozen, Stockmeyer]

« Le temps alternant, c'est de l'espace»

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:

- ***NL ⊆ P ⊆ NP ⊆ PSPACE ⊆ EXPTIME.***

- **Complete Problems**
  - **Connectivity is NL-Complete**
  - **QSAT is PSPACE-Complete**

By reducing decidability to reachability

More PSPACE-complete Problems

- Given a memory restricted Turing machine, does it terminate in at most *k* steps?
- Do two regular expressions describe different languages?
- Is a deadlock state possible within a system of communicating processors?
- Natural **generalizations of games.**
  - Othello, Hex, Geography, Rush-Hour, Instant Insanity
  - Shanghai, go-moku, Sokoban

Un plan

III. PSPACE et jeux

i. **Une remarque intéressante**

ii. Jeu à un joueur : Sokoban

iii. Jeu à deux joueurs : sauver un surfeur inconscient

Remarque intéressante

La complexité des jeux dépend de longueur (temps) des parties :

	Complexité des Jeux	
Longueur des parties	**Polynomiale**	**Exponentielle**
1 joueur	NP(-c)	PSPACE(-c)
2 joueurs	PSPACE(-c)	EXPTIME(-c)

Remarque intéressante

TABLE 2 – Différentes Complexités pour quelques jeux et puzzles populaires. Les complexités de l'espace d'états et de l'arbre de jeu sont exprimées en logarithme à base 10.

Nombre de joueurs	Longueur des parties	Jeu ou puzzle (taille standard)	Complexité Espace d'états	Complexité Arbre de jeu	Computationelle
1 joueur		Rubik's cube (3×3×3)	19	20	?
	Polynomiale	Taquin (5×5)	25	>37	NP-complet
		SameGame (15×15)	159	85	
	Exponentielle	Sokoban (20×20)	98	280	PSPACE-complet
2 joueurs	Polynomiale	Tic-tac-toe (3×3)	3	5	PSPACE-complet
		Othello (8×8)	28	58	
		Hex (11×11)	57	98	
		Gomoku (15×15)	105	70	
		Havannah (10×10×10)	127	157	
		Amazons (10×10)	40	212	
		Bridge (4×13)	<17	<40	
	Exponentielle	Checkers (8×8)	18	31	EXPTIME-complet
		Dames (10×10)	30	54	
		Échecs (8×8)	47	123	
		Shogi (9×9)	71	226	
		Go (19×19)	171	360	

Un plan

III. PSPACE et jeux

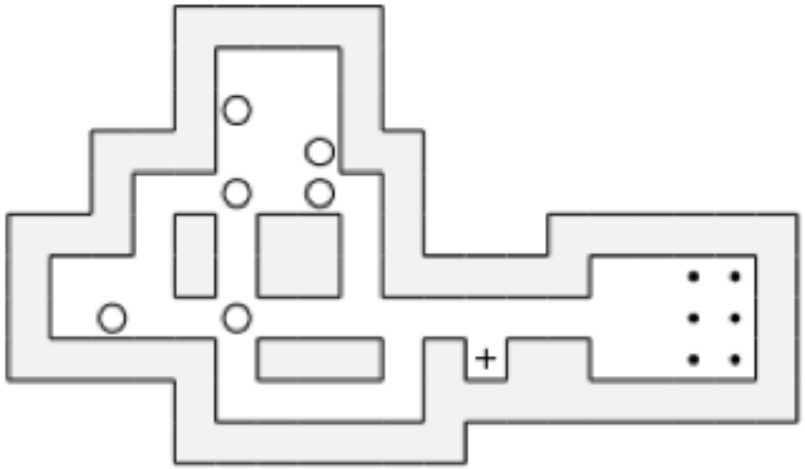
i. Une remarque intéressante

ii. Jeu à un joueur: Sokoban

iii. Jeu à deux joueurs: sauver un surfeur inconscient

Le jeu de Sokoban

- Règles du jeux
  - Le joueur (magasinier) doit pousser toutes les caisses sur des cases spéciales dites objectifs.
  - Il est impossible de tirer une caisse ou d'en pousser deux d'un même mouvement.



Le Sokoban est dans PSPACE

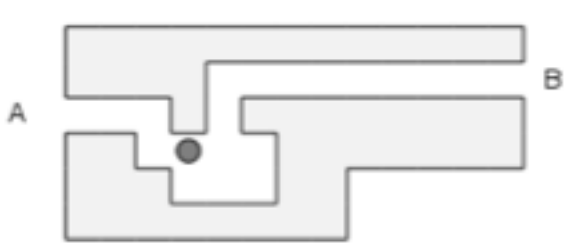
- A un niveau on peut associer son **graphe des configurations** :
  - **Configuration** = répartition de caisses + position du magasinier.
  - **Arête** = transition réalisable (mouvement autorisé)

-> Sokoban se ramène à un **problème d'accessibilité dans un graphe**, pour lequel on dispose d'un algorithme économe en espace.

[Culberson, J. (1999). Sokoban is PSPACE-complete. In *Proceedings in Informatics* (Vol. 4, pp. 65-76).]

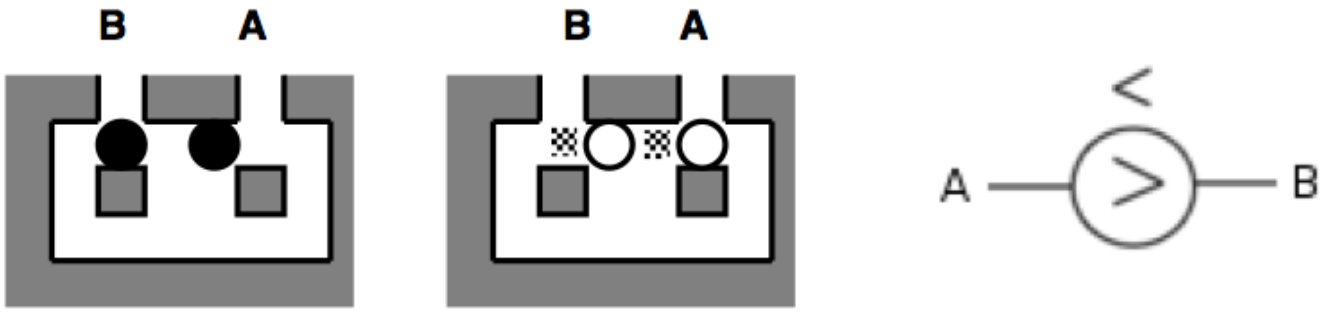
Sokoban est PSPACE-complet

- Construction de Gadgets :



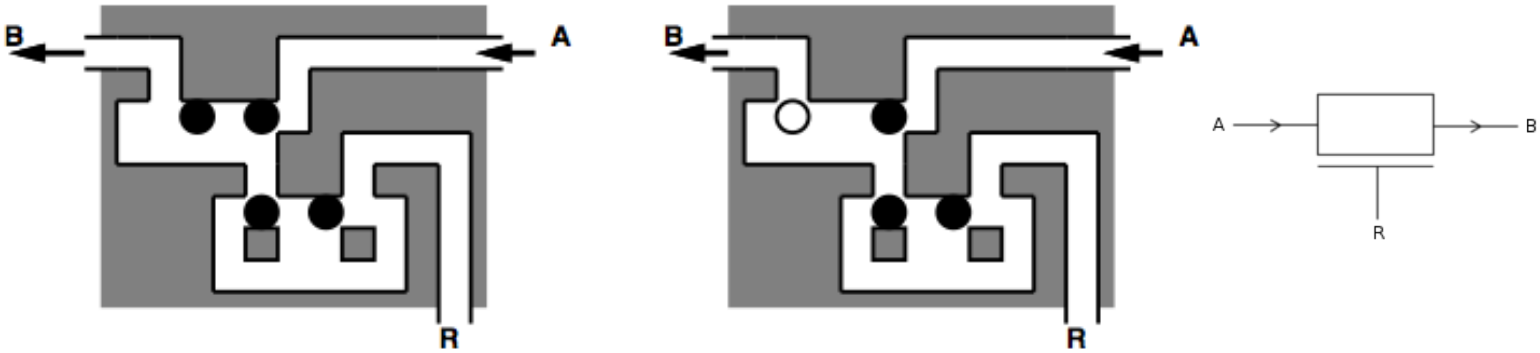
- **La diode.** Passage que de A vers B et n'altère pas l'état du *gadget,* qui reste résolu.

Sokoban est PSPACE-complet



- **L'inverseur** Ce gadget dispose de deux états.
  - Etat résolu : passasge de A vers B, passant le gadget dans l'état irrésolu
  - Etat irrésolu, passage de B vers A.

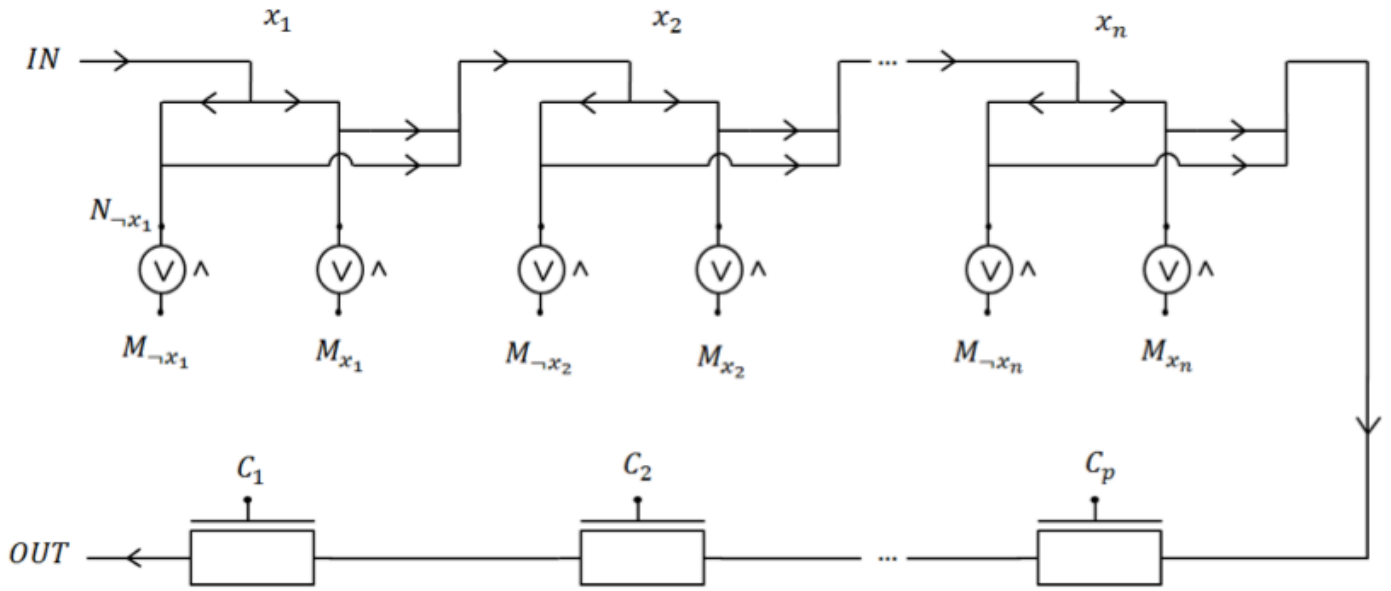
Sokoban est PSPACE-complet



- **Le transistor**
  - deux diodes associées afin d'empêcher l'entrée en B et la sortie en A.
  - Dans l'état résolu, le passage de A à B est bloqué.
  - Possible de le déverrouiller pour un unique passage en allant au point R, passant ainsi le gadget dans l'état irrésolu.

Sokoban est PSPACE-complet

- Gadget Gsat : possible de traverser si et seulement si ψ est satisfiable.



Pour l'instant, on a effectué réduction de SAT à Sokoban : NP-difficile

Sokoban est PSPACE-complet

Idée : On intègre les quantificateurs 1 à 1.

pour k tel que $0 \leqslant k \leqslant n$,

$$F^k(x_{k+1}, \cdots, x_n) = Q_k x_k \, Q_{k-1} x_{k-1} \cdots Q_1 x_1 \, \psi(x_1, \cdots, x_n)$$

On a

$$F^k(x_{k+1}, \cdots, x_n) = \begin{cases} F^{k-1}(0, x_{k+1}, \cdots, x_n) \vee F^{k-1}(1, x_{k+1}, \cdots, x_n) & \text{si } Q_k = \exists \\ F^{k-1}(0, x_{k+1}, \cdots, x_n) \wedge F^{k-1}(1, x_{k+1}, \cdots, x_n) & \text{si } Q_k = \forall \end{cases}$$

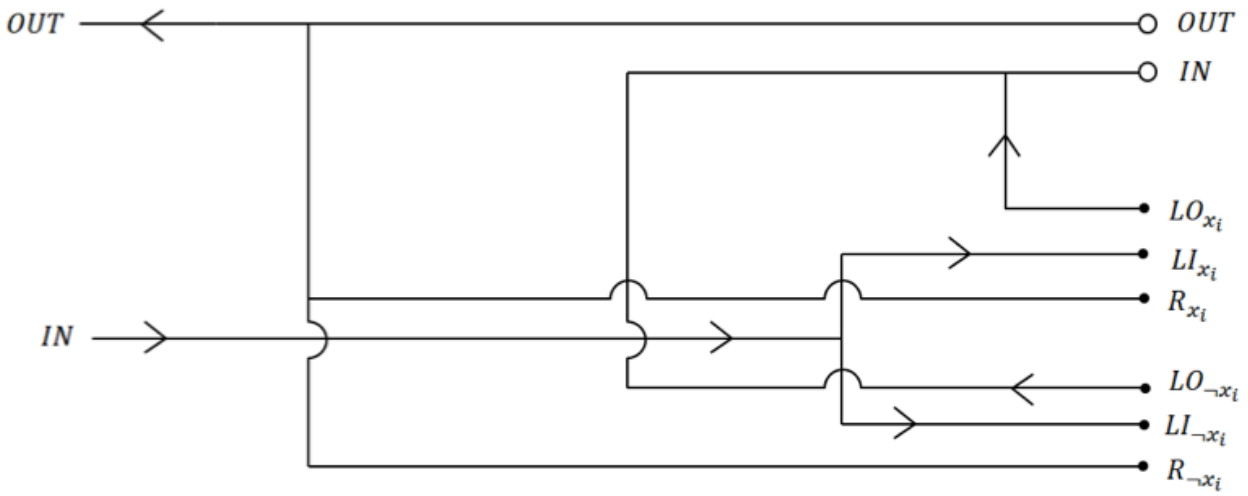
Pour tout k, on construit par **récurrence** un gadget :

G^k traversable ssi F_k est satisfiable.

Sokoban est PSPACE-complet

Si $Q_k = \exists$:

- Choisir la valeur de x_k
- Altèrer G^{k-1} en bloquant le point M_{xk} ou M_{nonxk}
- Tenter de relier IN à OUT (possible ssi F^k)

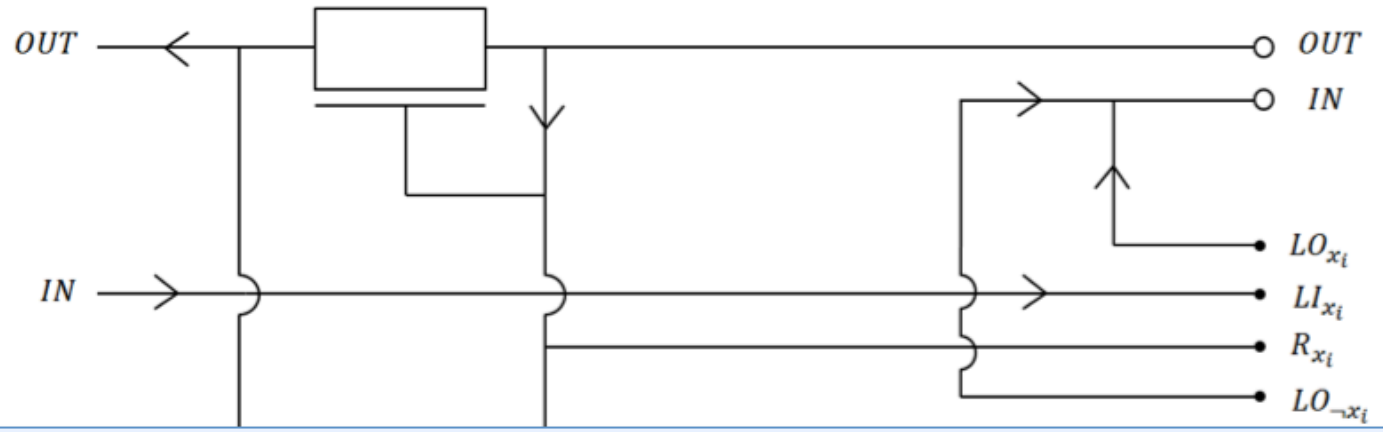


Sokoban est PSPACE-complet

Si $Q_k = \forall$:

On impose deux passages dans G_{k-1}

- 1. avec blocage de M_{xk}
- 2. avec blocage de M_{nonxk}



Theorem. Sokoban is PSPACE-Complete

Un plan

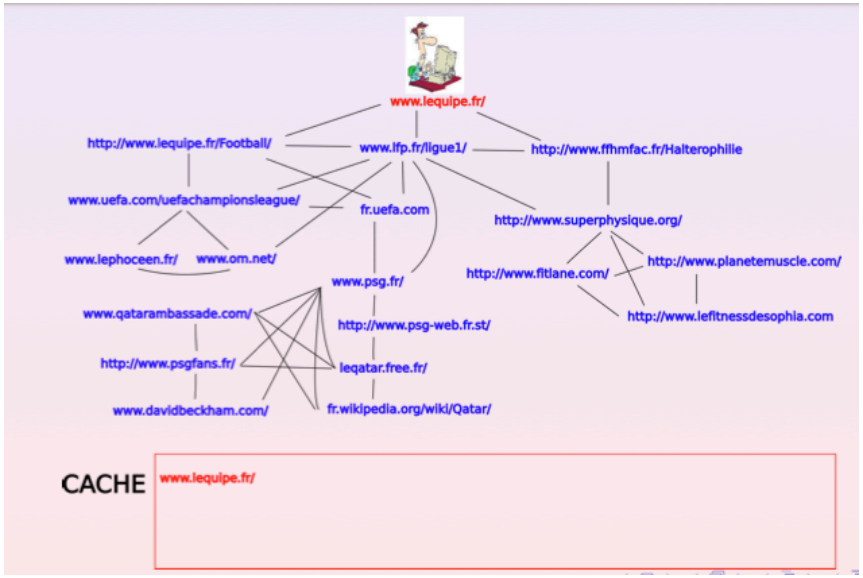
III. PSPACE et jeux

i. Une remarque intéressante

ii. Jeu à un joueur : Sokoban

iii. Jeu à deux joueurs : sauver un surfeur inconscient

Le jeu du surfeur inconscient

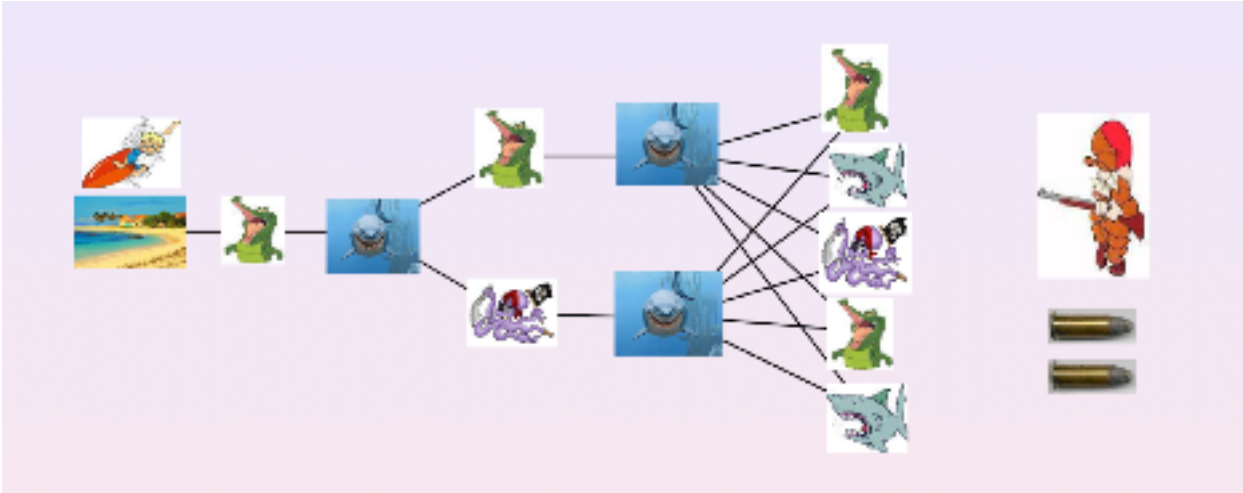


Problem motivation:

- **Idea: pre-loading web pages** before the **web Surfer** accesses it
- **Constraint: bandwidth**, takes time to load webpages in cache
- **Goal:** Avoid the web Surfer to wait

[To satisfy Impatient Web Surfers is hard, Fomin, Giroire, Jean-Marie, Mazauric, Nisse]

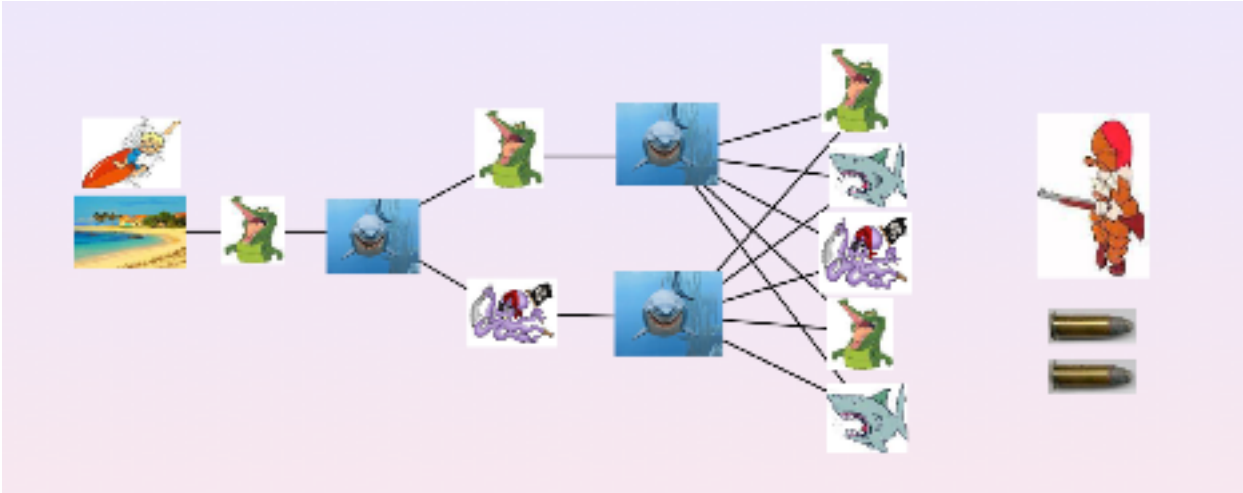
Model: a 2-player game



Players:

- A **surfer** starts from a safe homebase v_0 in G
- A **guard** with **k bullets**

Model: a 2-player game



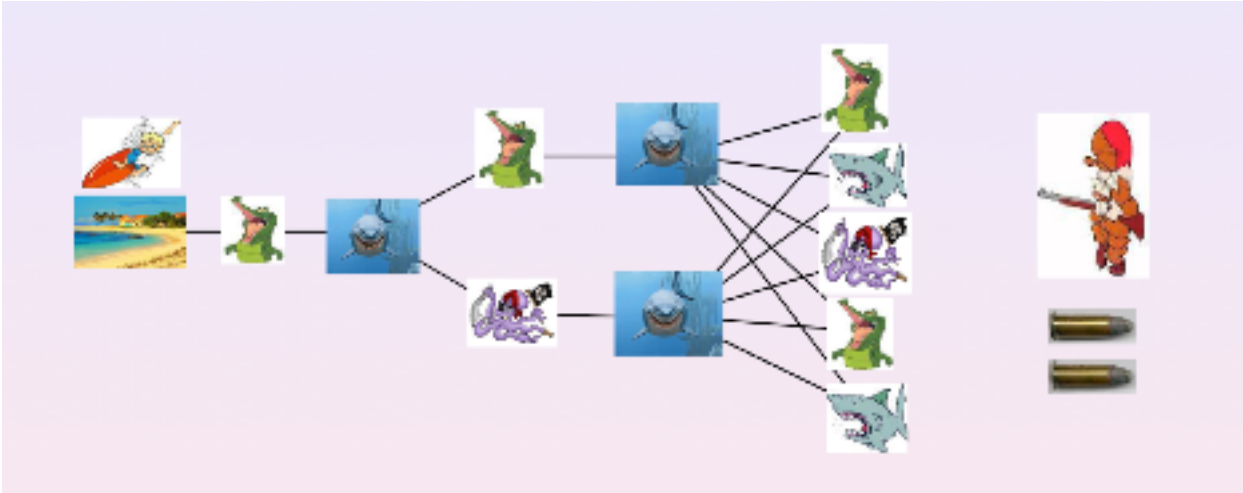
Turn-by-turn game:

- 1. The guard secures <= k nodes
- 2. Then, the surfer may move to an adjacent node.

Defeat: surfer in an unsafe node.

Victory: graph safe

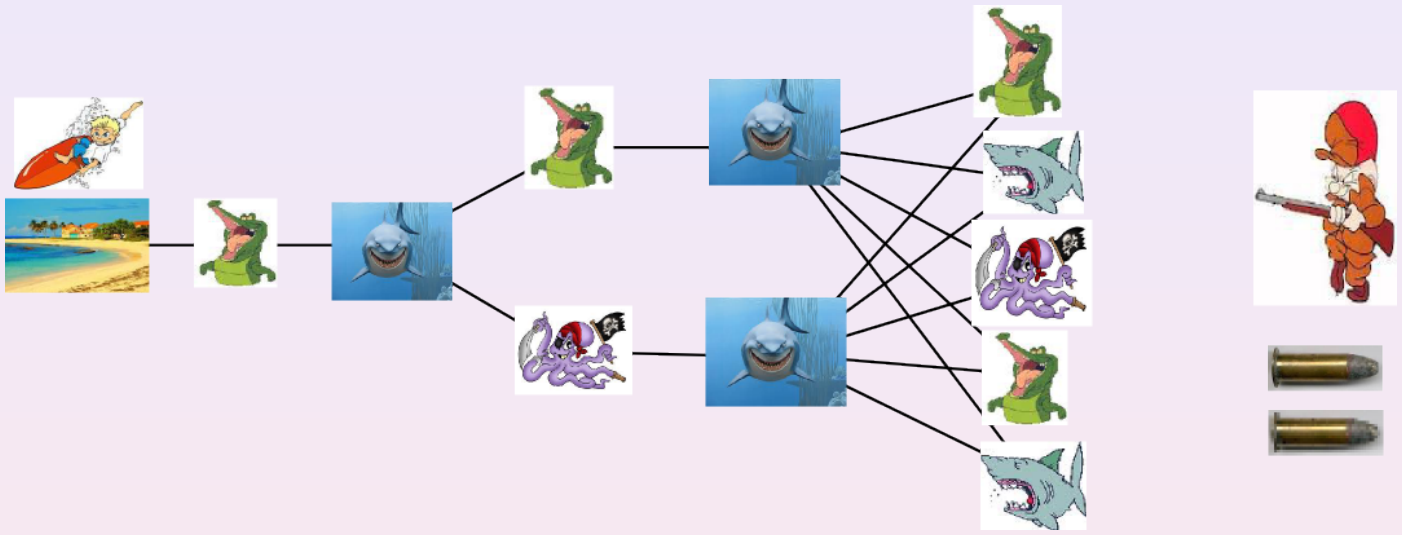
Model: a 2-player game



Problem: minimize the number of bullets to win for any surfer trajectories

Surveillance number of G from v_0: **$sn(G,v_0)$**

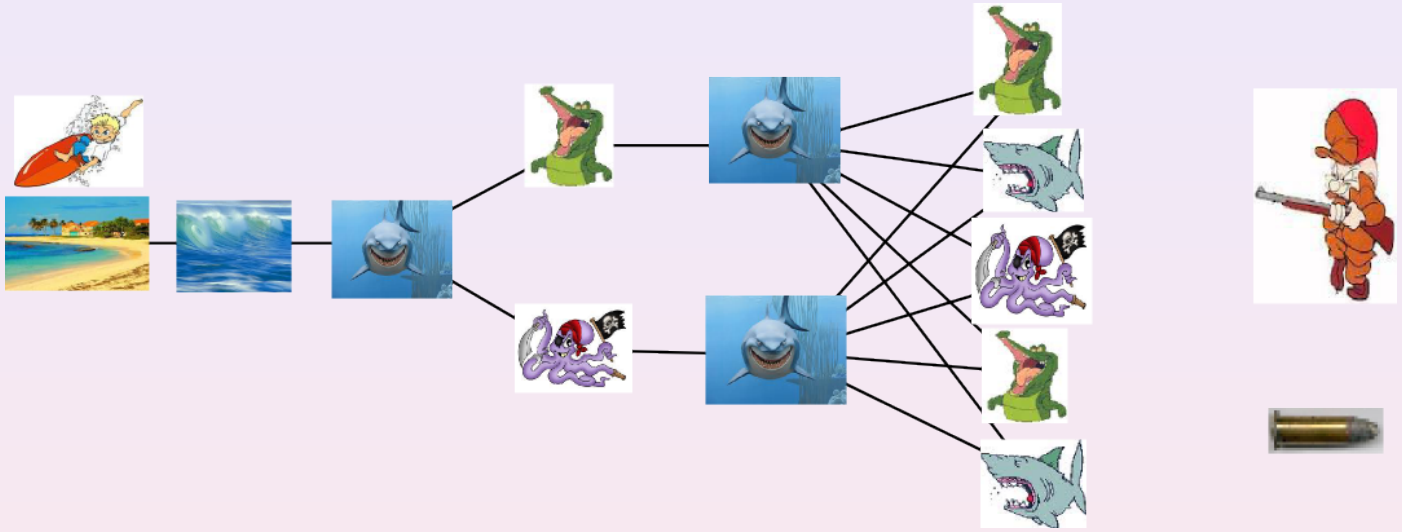
First Example



with 1 bullet: after 2 steps, Surfer faces 2 dangerous nodes!!

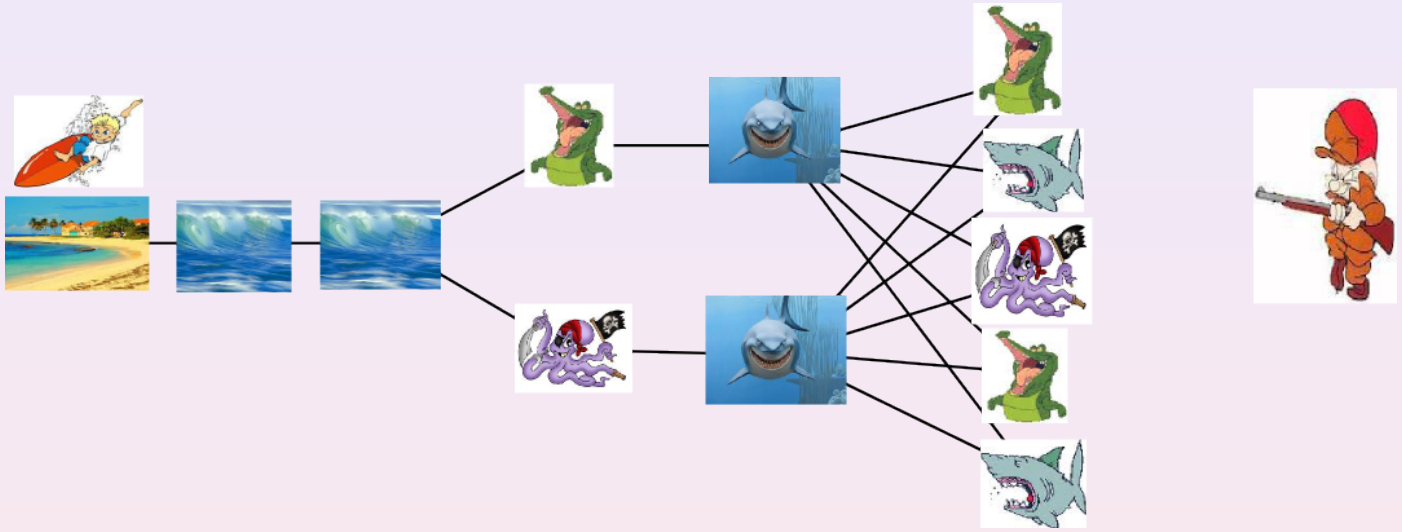
$$sn(G, v_0) > 1$$

First Example



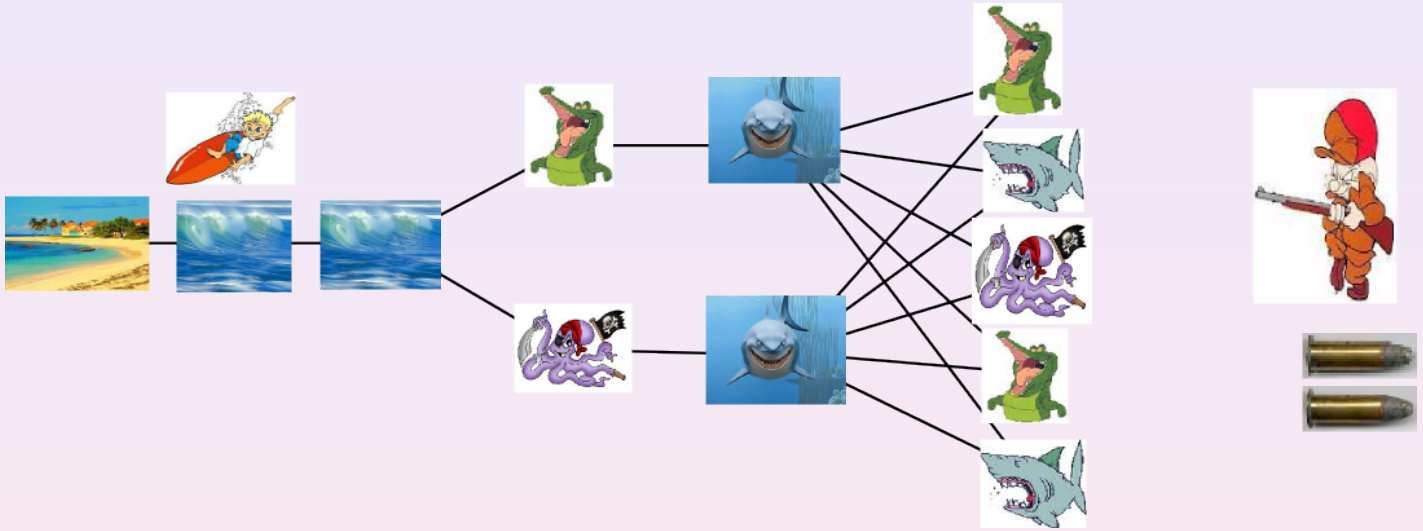
Guard uses his bullets

First Example



Guard uses (all) his bullets

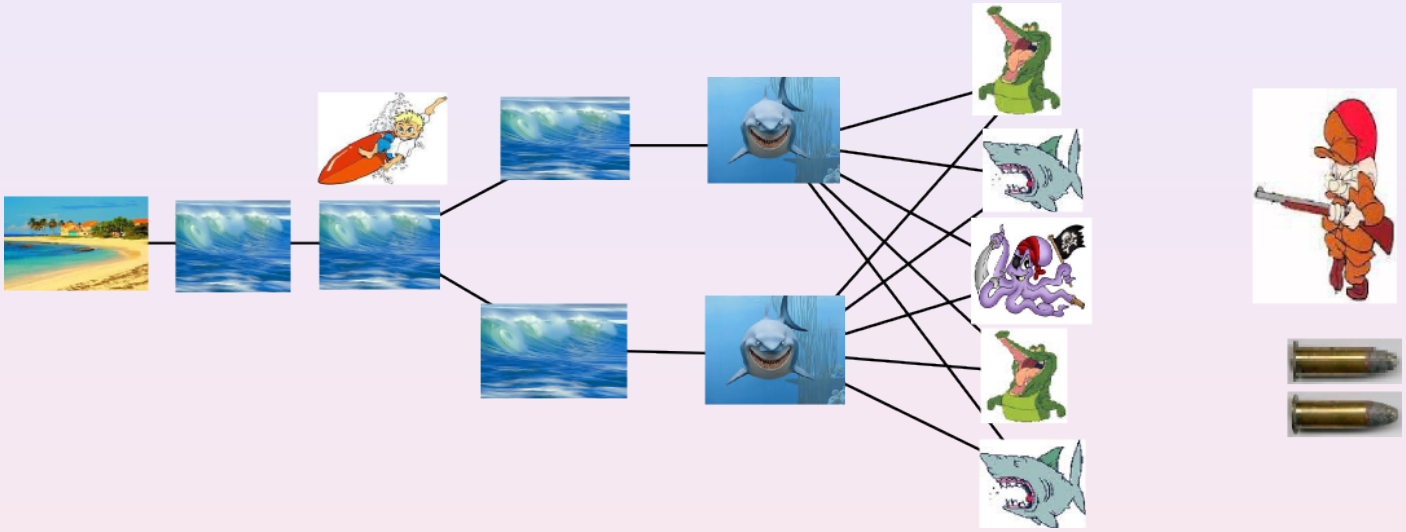
First Example



Guard uses (all) his bullets, **then** Surfer may move

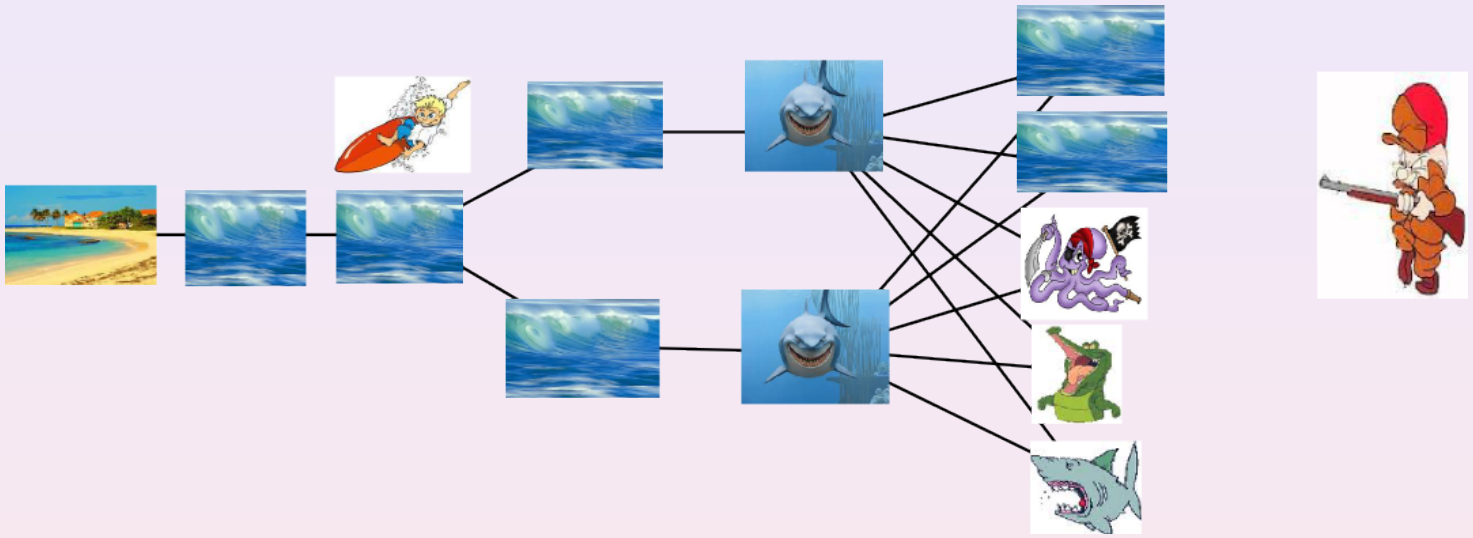
Clearly: worst case if Surfer always move

First Example



Guard uses (all) his bullets, **then** Surfer moves

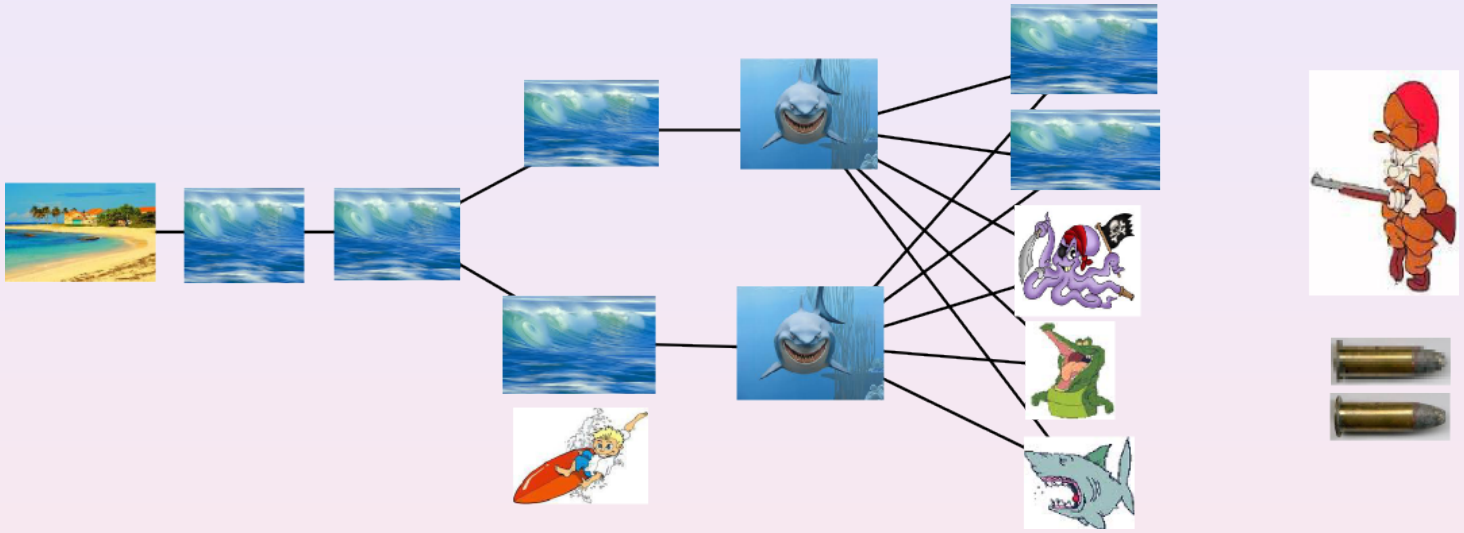
First Example



Guard uses (all) his bullets, **then** Surfer moves

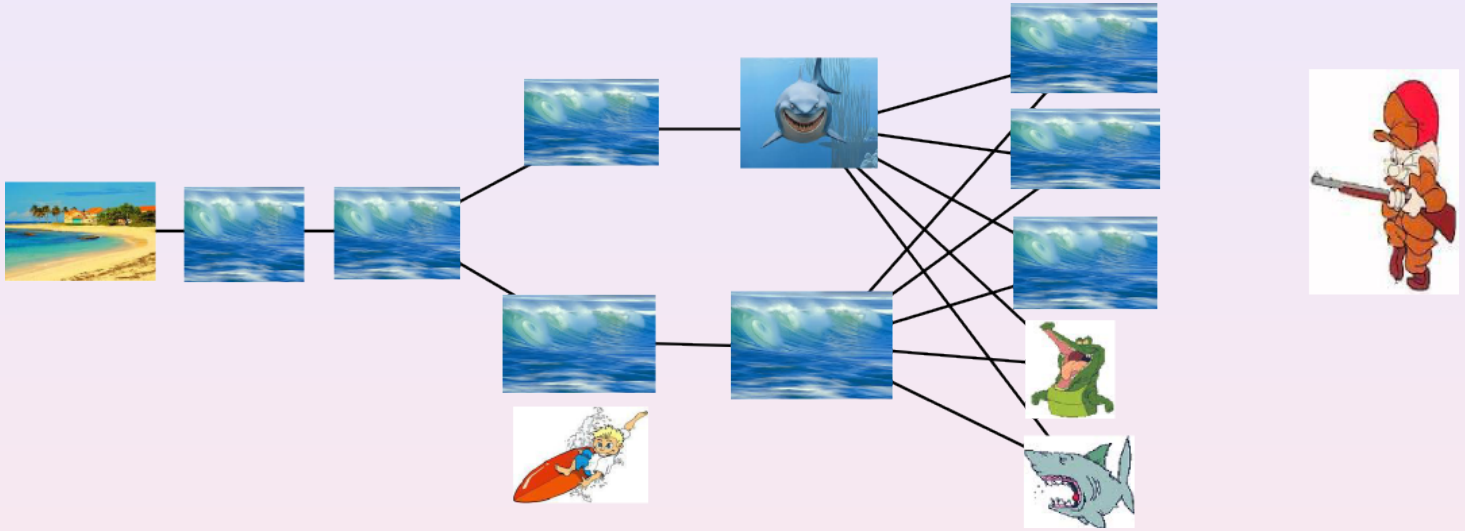
Guard may secure any node in the graph

First Example



Guard uses (all) his bullets, **then** Surfer moves

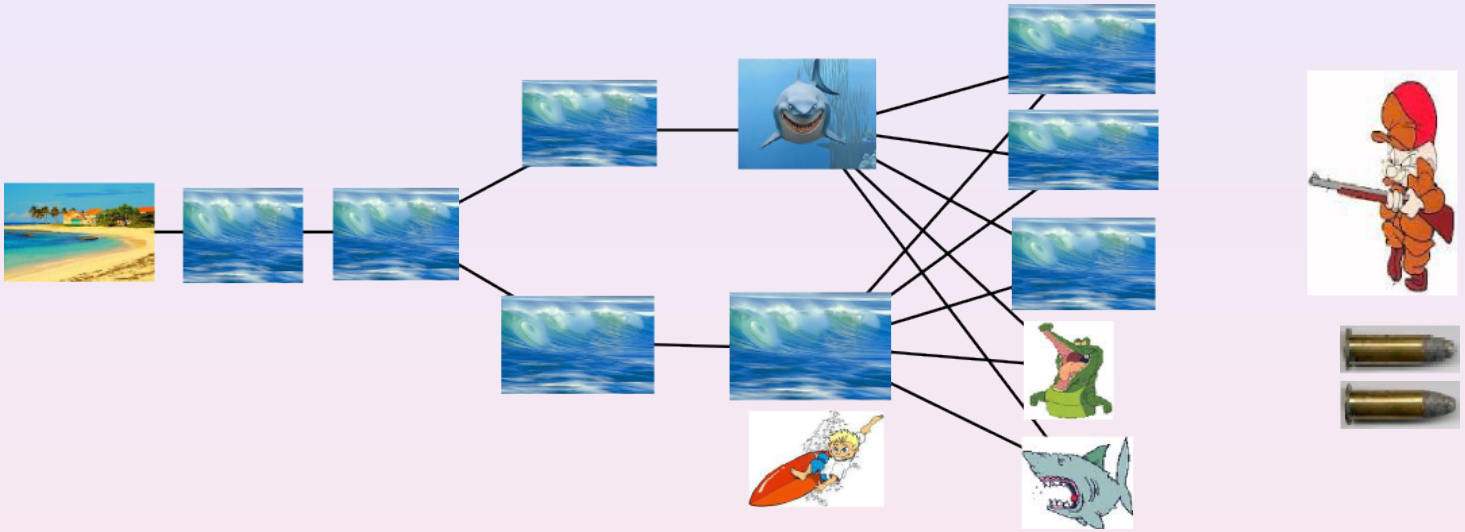
First Example



Guard uses (all) his bullets, **then** Surfer moves

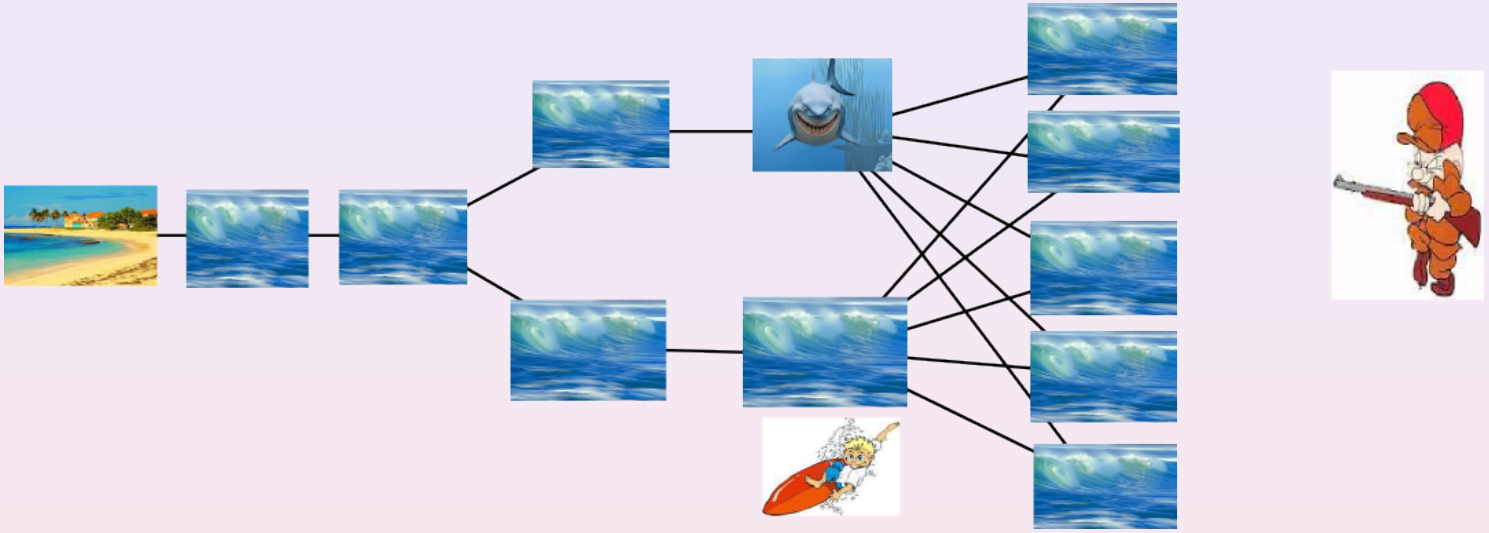
strategy: safe nodes + Surfer's node $\Rightarrow \leq k$ nodes to secure

First Example



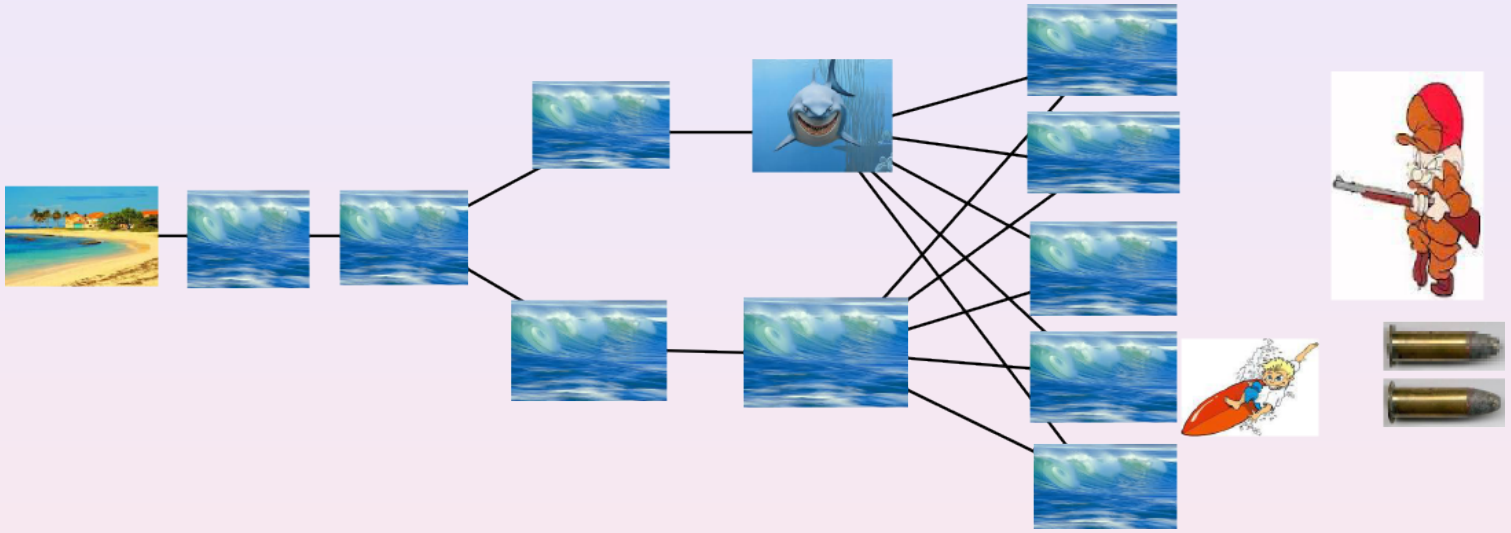
Guard uses (all) his bullets, **then** Surfer moves

First Example



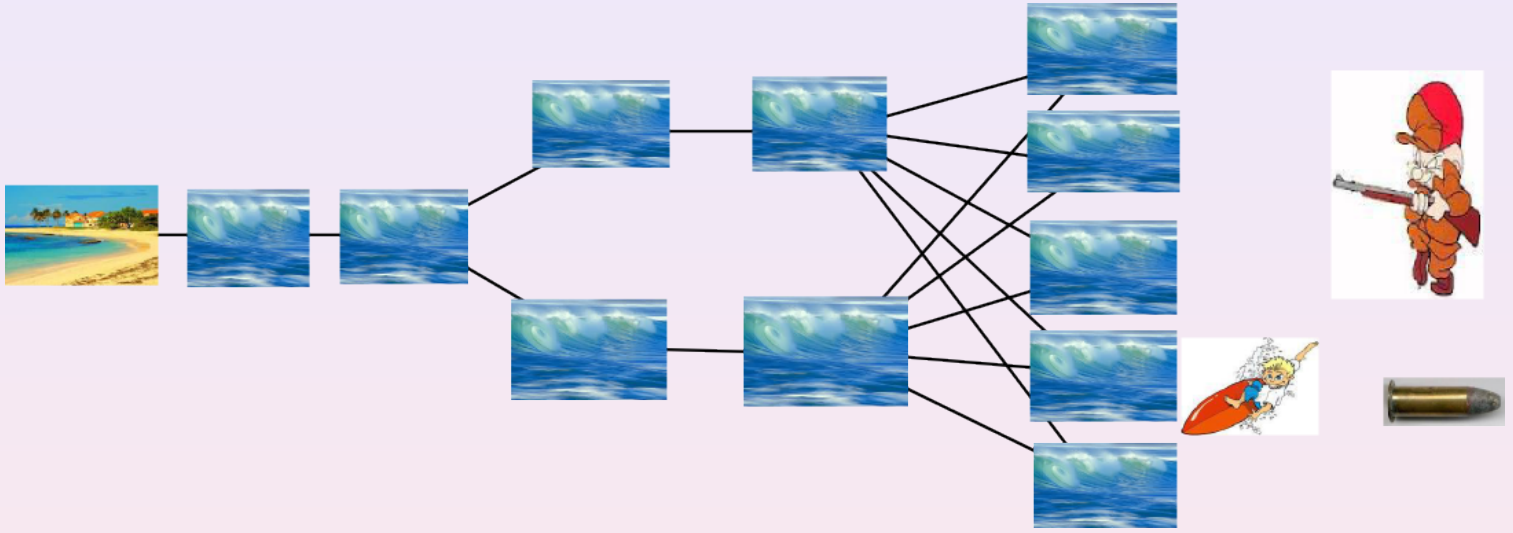
Guard uses (all) his bullets, **then** Surfer moves

First Example



Guard uses (all) his bullets, **then** Surfer moves

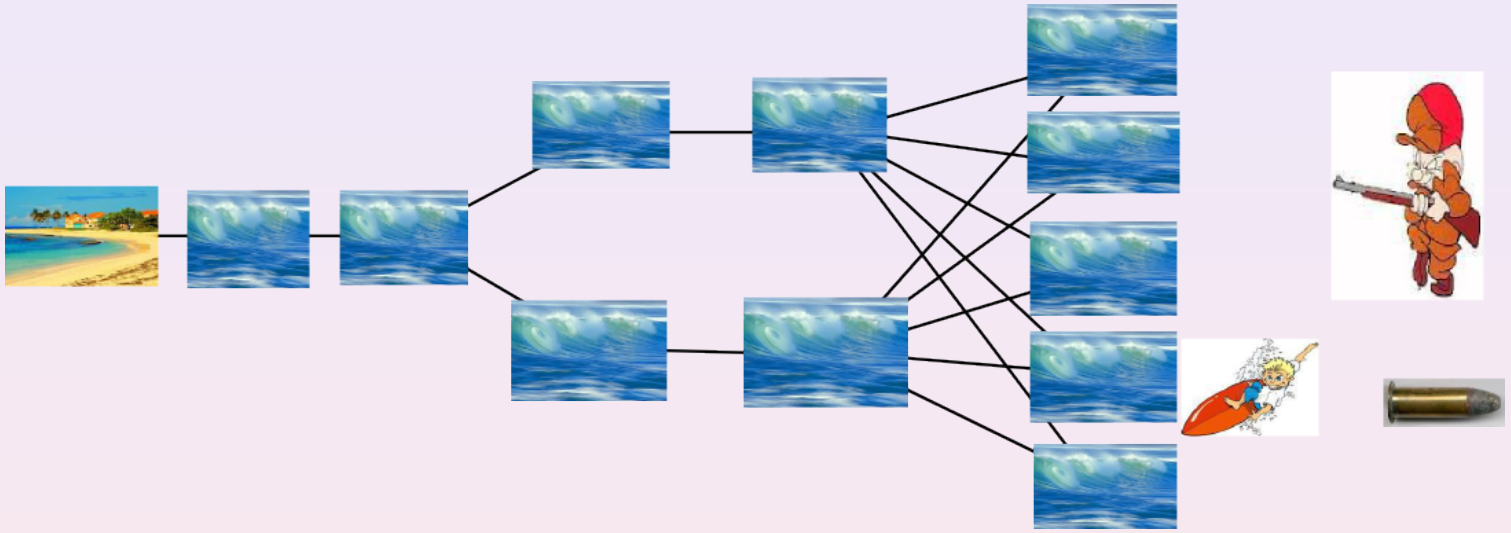
First Example



Guard uses (all) his bullets, **then** Surfer moves

All nodes safe: Victory against this trajectory of the Surfer

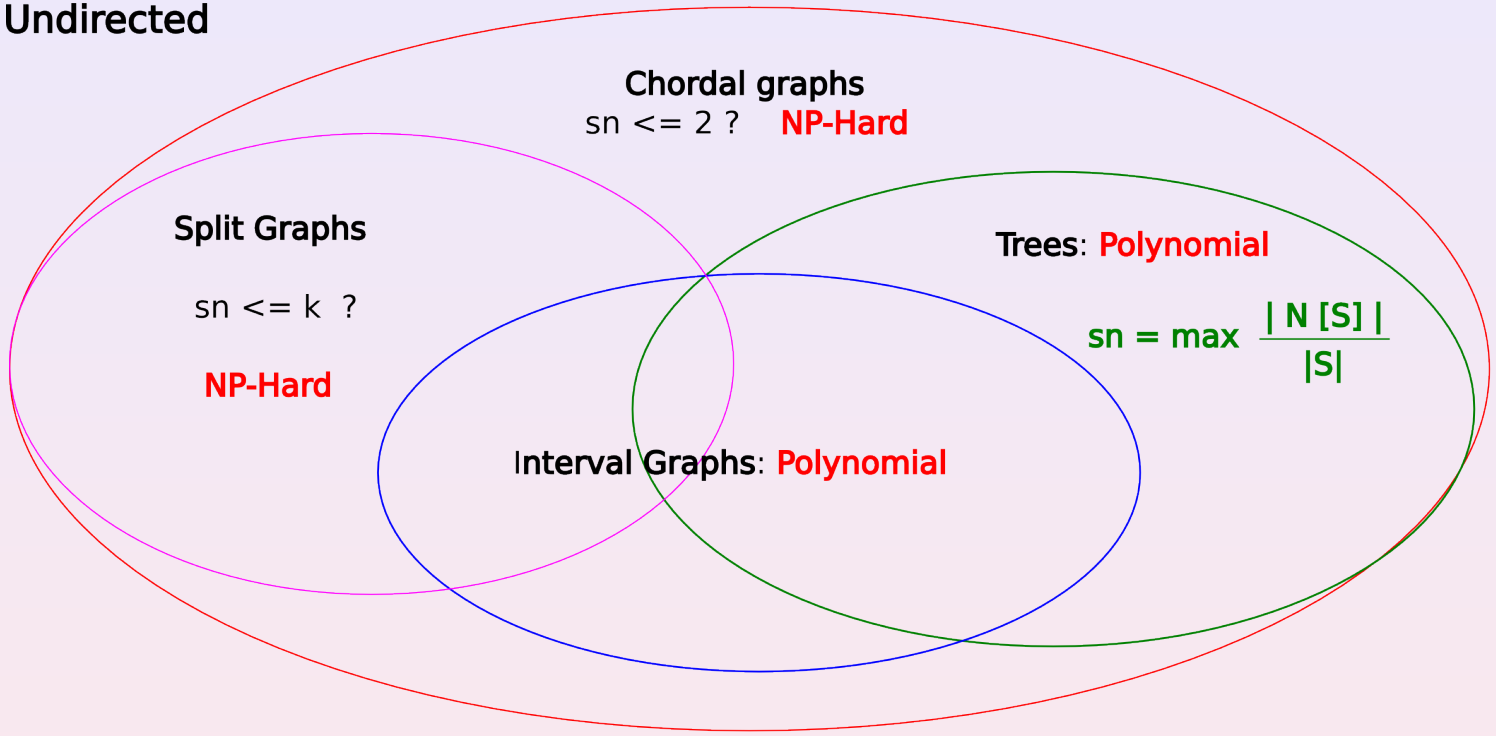
First Example



In this example, all Surfer's trajectory similar (by symmetry)

Victory whatever Surfer's trajectory $\Rightarrow sn(G, v_0) = 2$

Results: Complexity, Algorithms and Combinatoric



Directed

DAGs: sn <=4 ? **P-SPACE-Complete**

DAGs: sn <=2 ? **NP-Hard**

General

(out)-**degree (v0)** $\leq$ **sn** $\leq$ $\max \begin{pmatrix} \text{degree (v0)} \\ \textbf{max degree} \text{ -1} \\ \text{max out-degree} \end{pmatrix}$

$O(2^n)$ exact algorithm

DAG: Directed Acyclic Graph.

reduction from

3-QSAT (PSPACE-complete)

Inputs:

- set of variables $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$
- logical formula $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$

Question: is it true?

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \forall x_n \exists y_n \ \ \Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$$

PSPACE-hardness in DAGs $sn \leq 4$?

DAG: Directed Acyclic Graph.

reduction from

3-QSAT (PSPACE-complete)

Inputs:

- set of variables $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$
- logical formula $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$

Question: is it true?

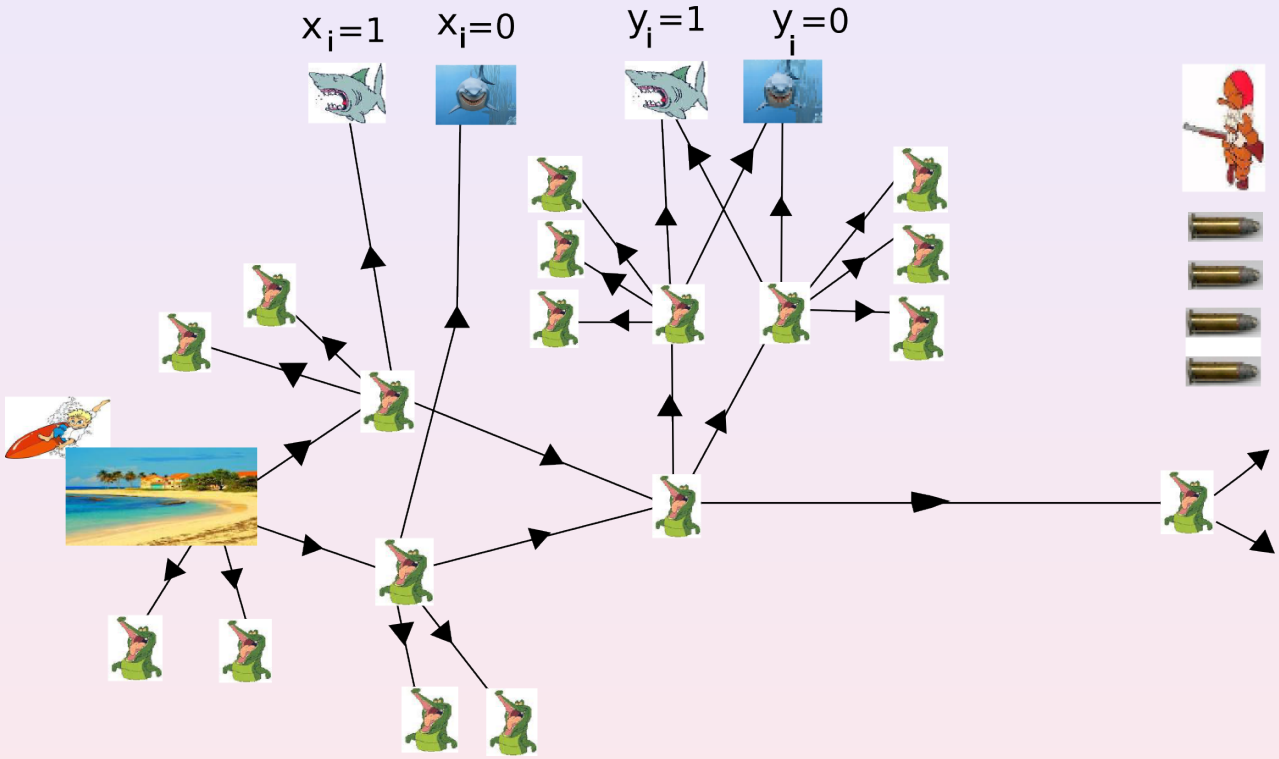
$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \forall x_n \exists y_n \ \ \Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$$

- "$\forall x_i$" depends on Surfer's path
- "$\exists y_i$" depends on Guard's strategy

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$



Gadget for x_i and y_i:

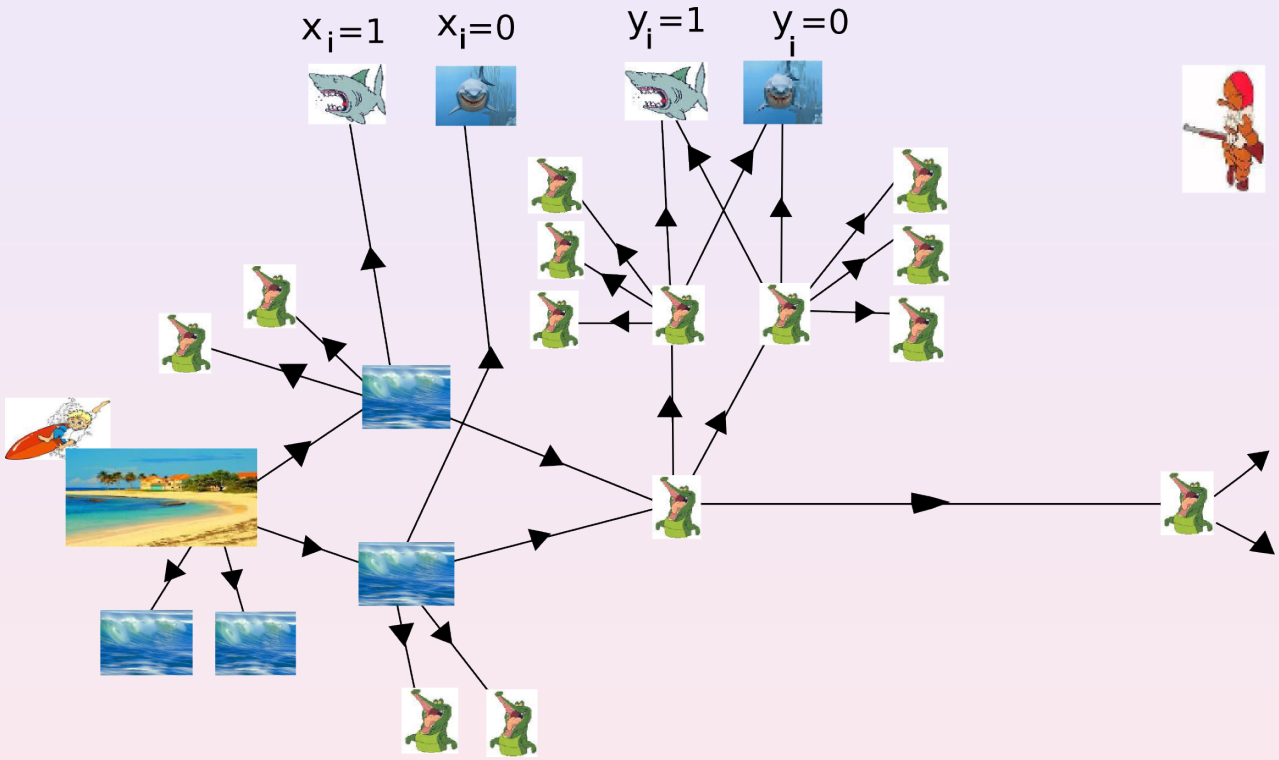
When Surfer reaches the right, 2 "sharks" have left

$\Rightarrow$ assignment of x_i (Surfer choice) and y_i (Guard choice)

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$



Gadget for x_i and y_i:

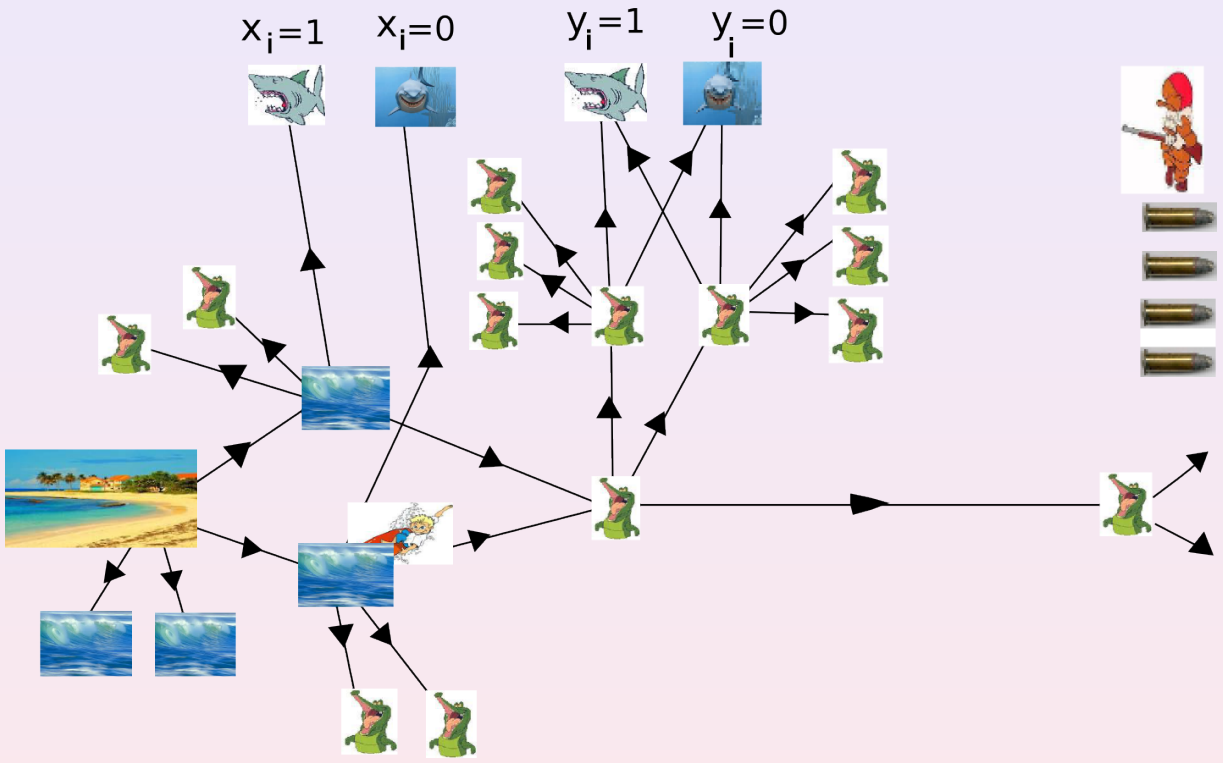
When Surfer reaches the right, 2 "sharks" have left

$\Rightarrow$ assignment of x_i (Surfer choice) and y_i (Guard choice)

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$



Gadget for x_i and y_i:

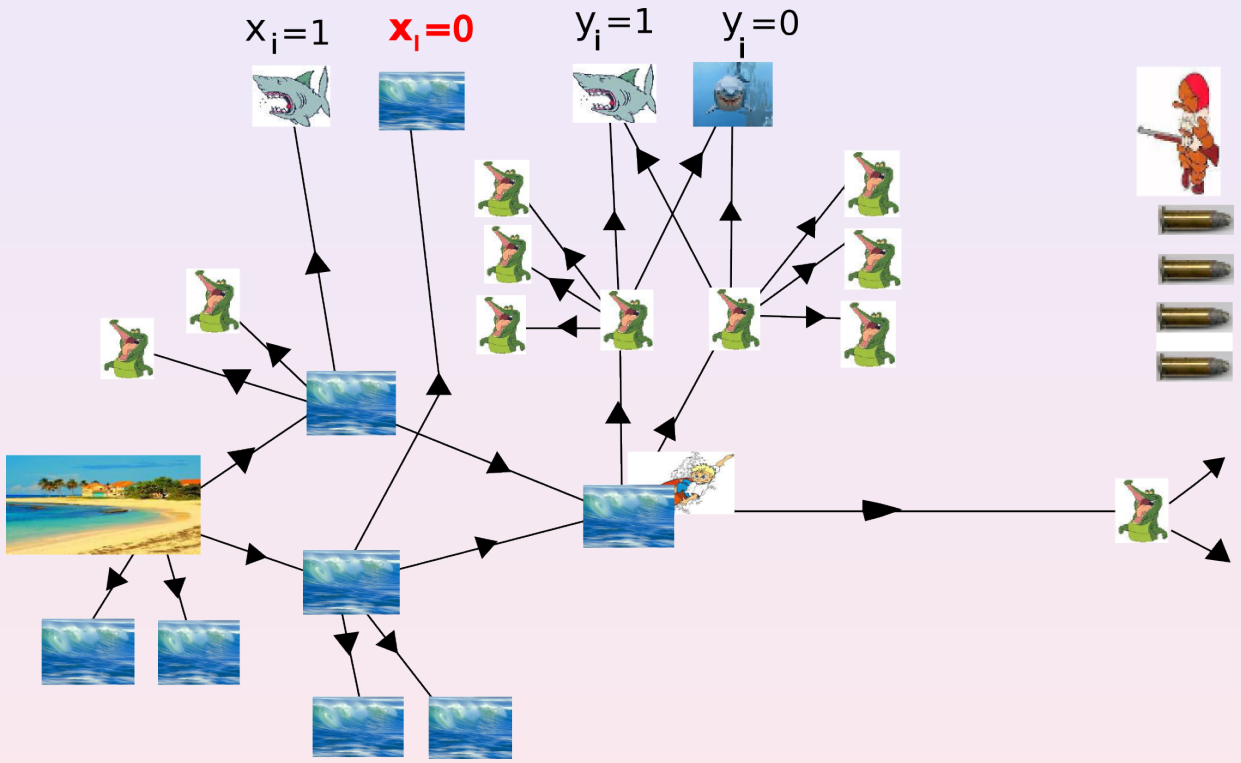
When Surfer reaches the right, 2 "sharks" have left

$\Rightarrow$ assignment of x_i (Surfer choice) and y_i (Guard choice)

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$



Gadget for x_i and y_i:

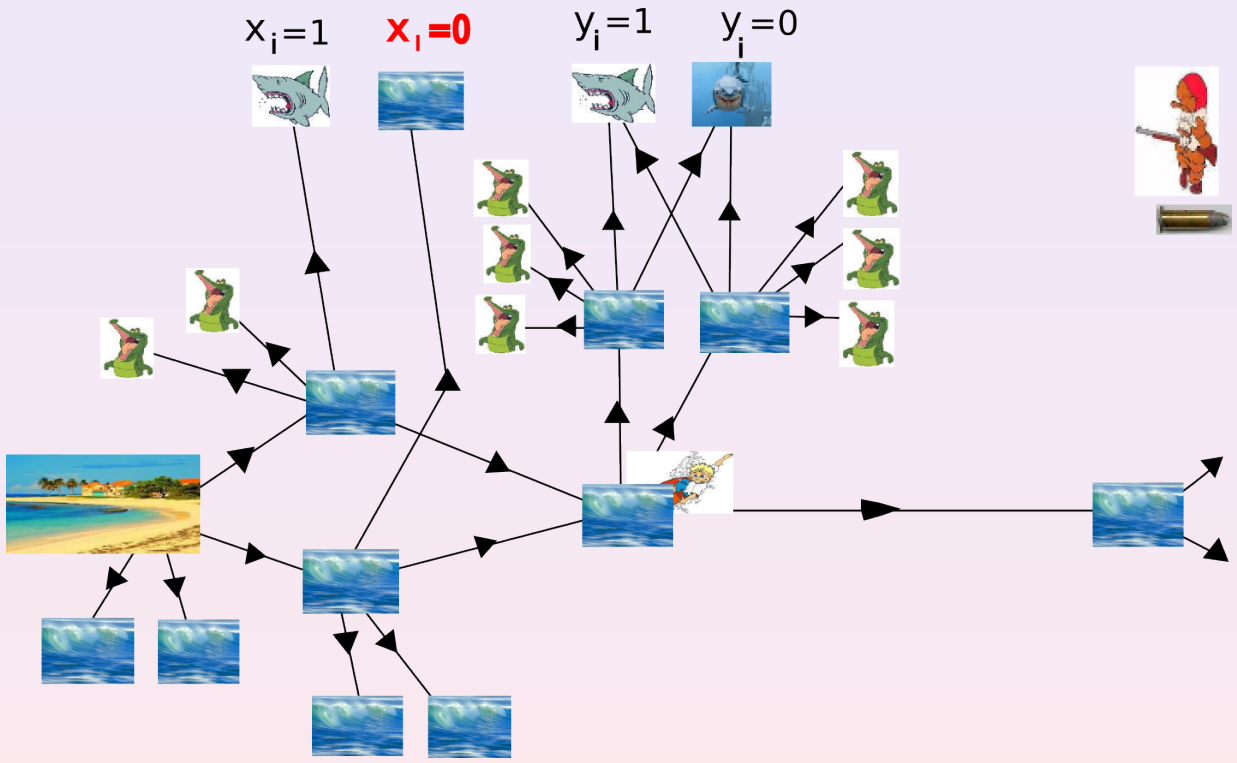
When Surfer reaches the right, 2 "sharks" have left

$\Rightarrow$ assignment of x_i (Surfer choice) and y_i (Guard choice)

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$



Gadget for x_i and y_i:

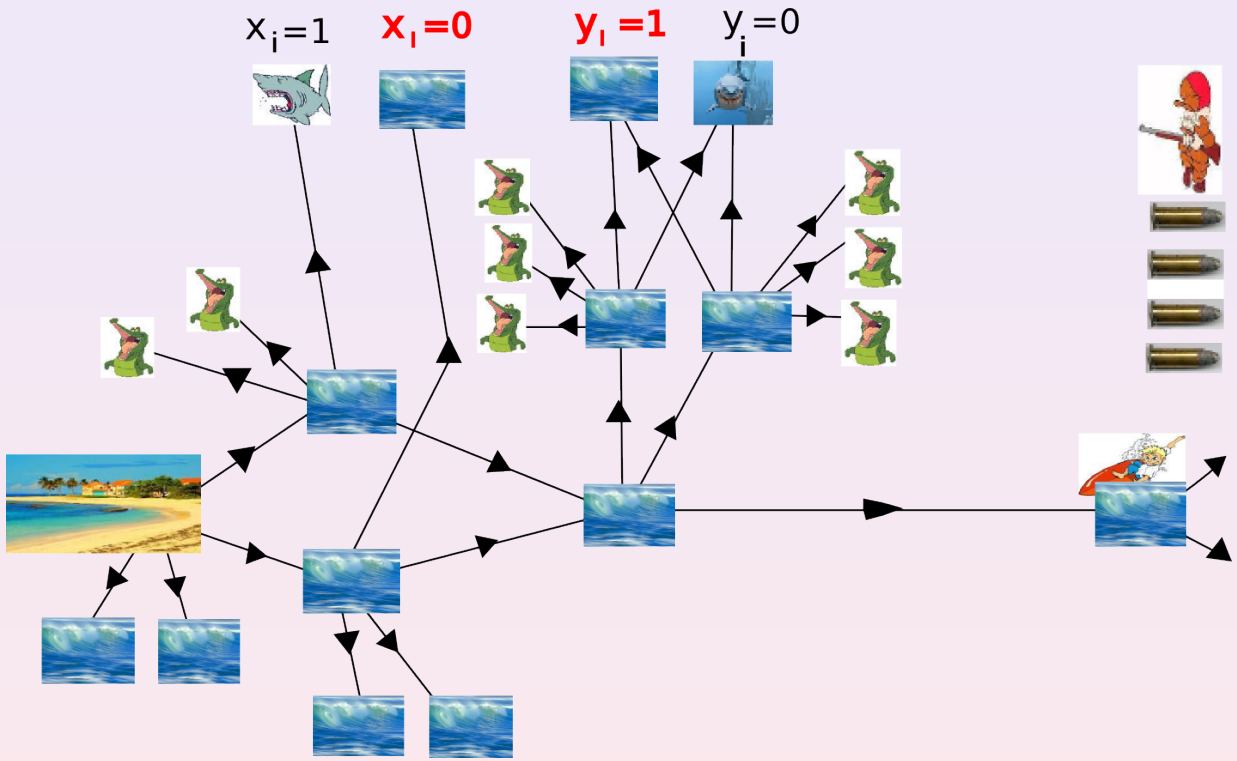
When Surfer reaches the right, 2 "sharks" have left

$\Rightarrow$ assignment of x_i (Surfer choice) and y_i (Guard choice)

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$



Gadget for x_i and y_i:

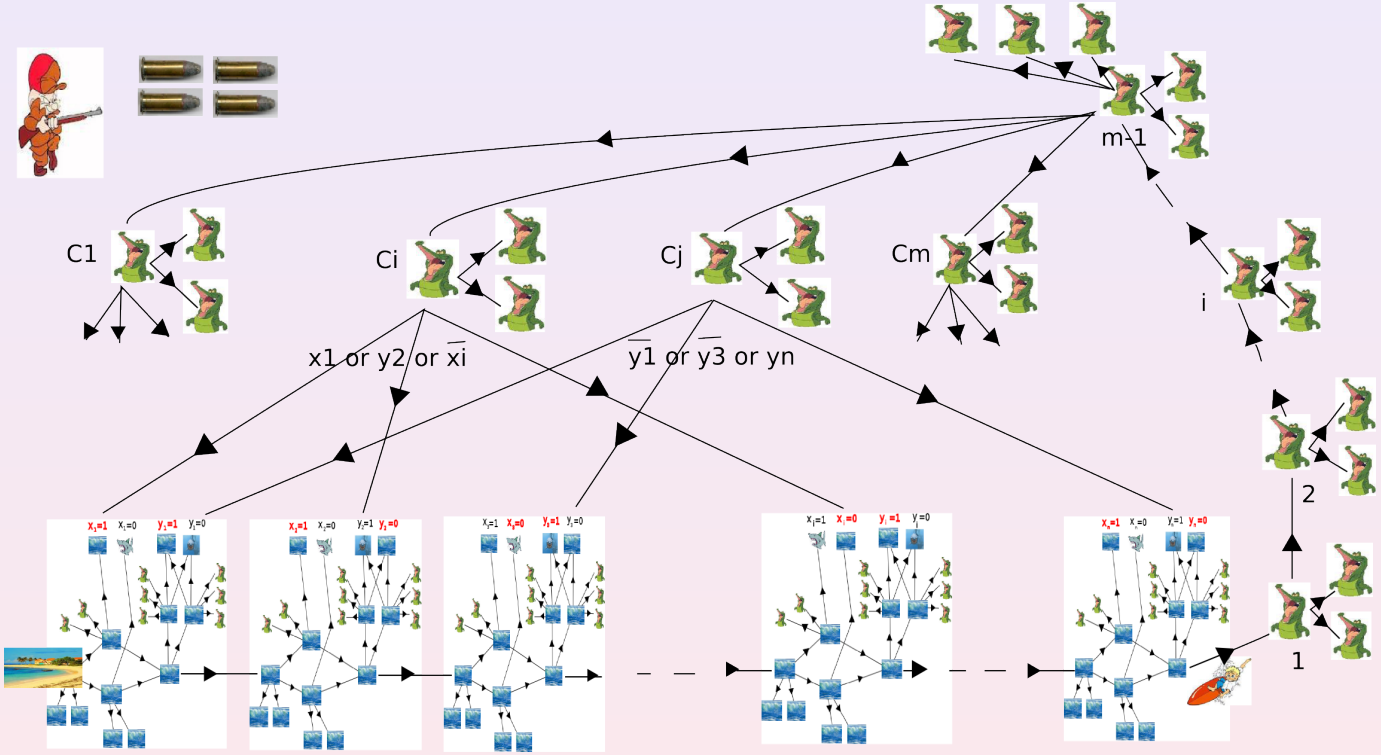
When Surfer reaches the right, 2 "sharks" have left

$\Rightarrow$ assignment of x_i (Surfer choice) and y_i (Guard choice)

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$

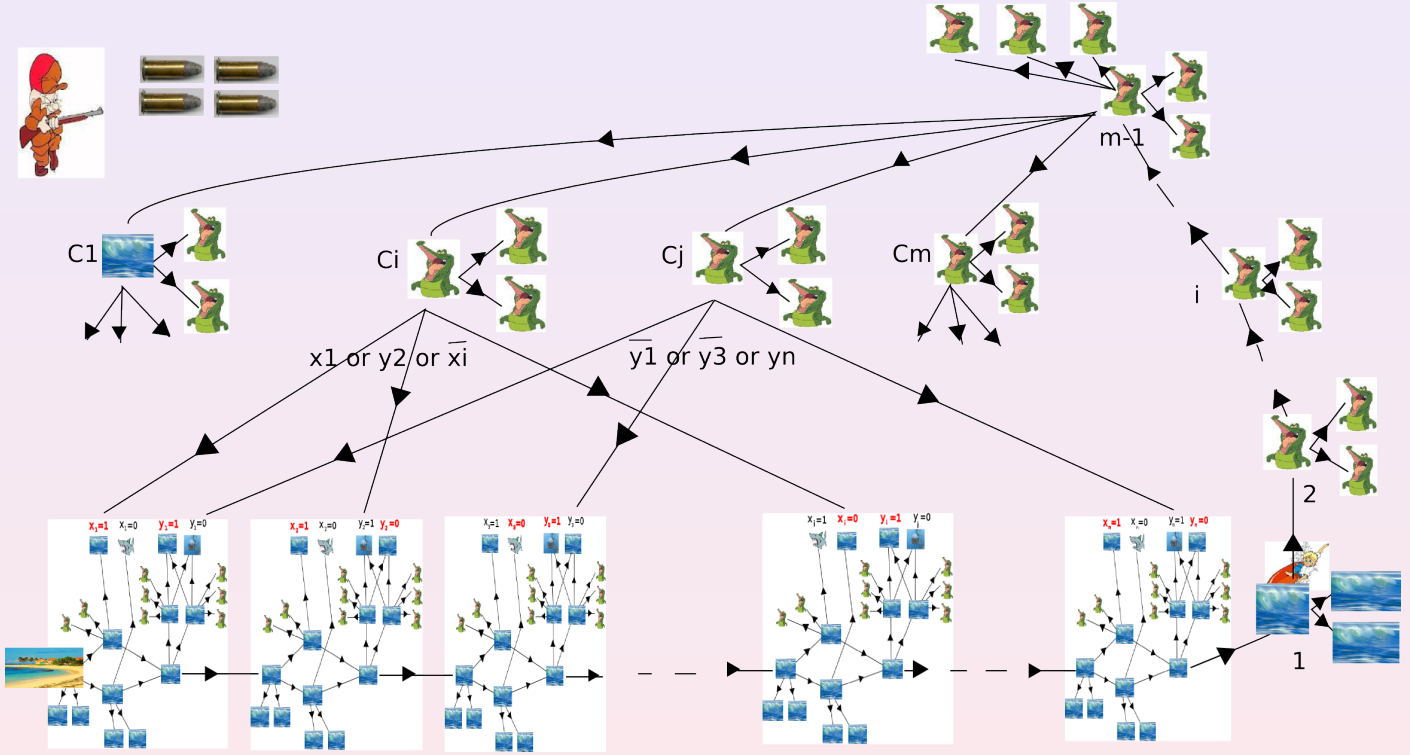


When Surfer reaches the bottom-right, all variables have been assigned: x_i's by Surfer, y_i's by Guard

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$

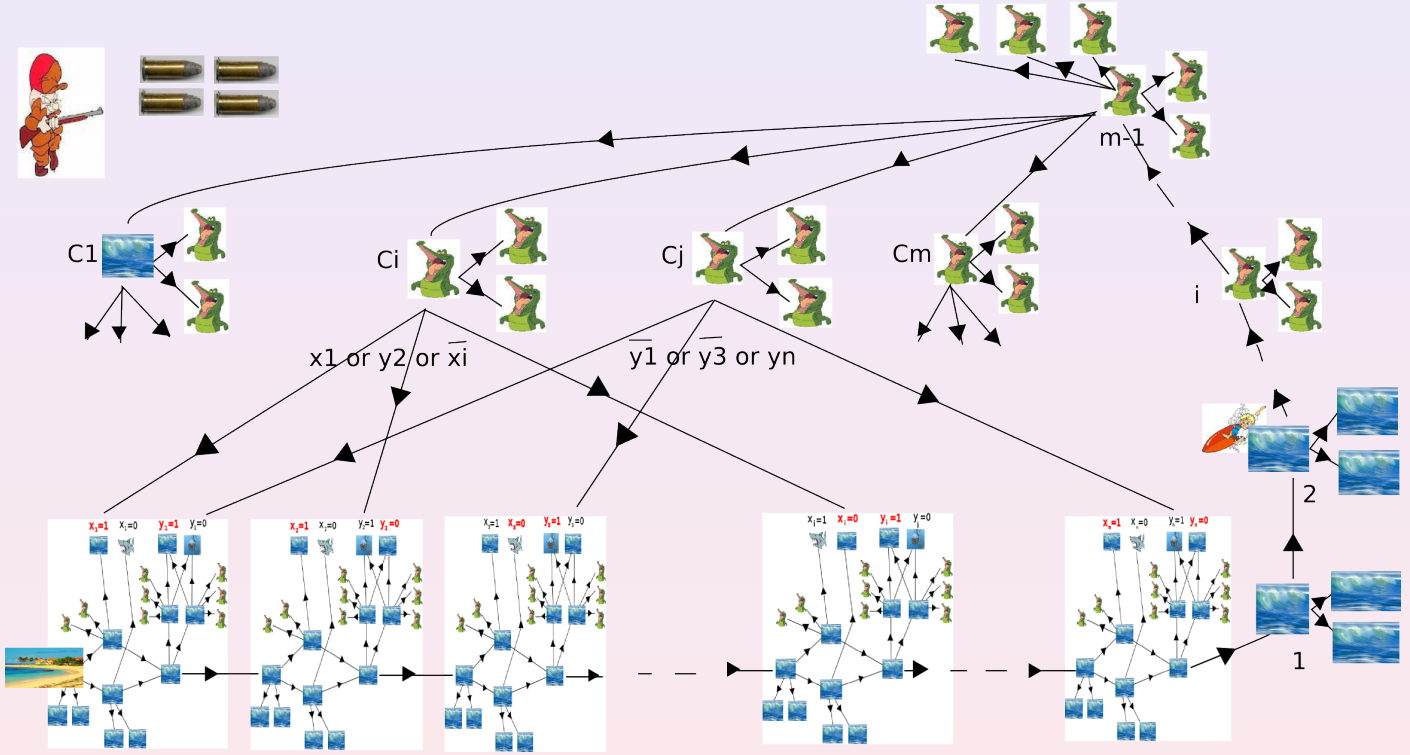


When Surfer reaches the bottom-right, all variables have been assigned: x_i's by Surfer, y_i's by Guard

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$

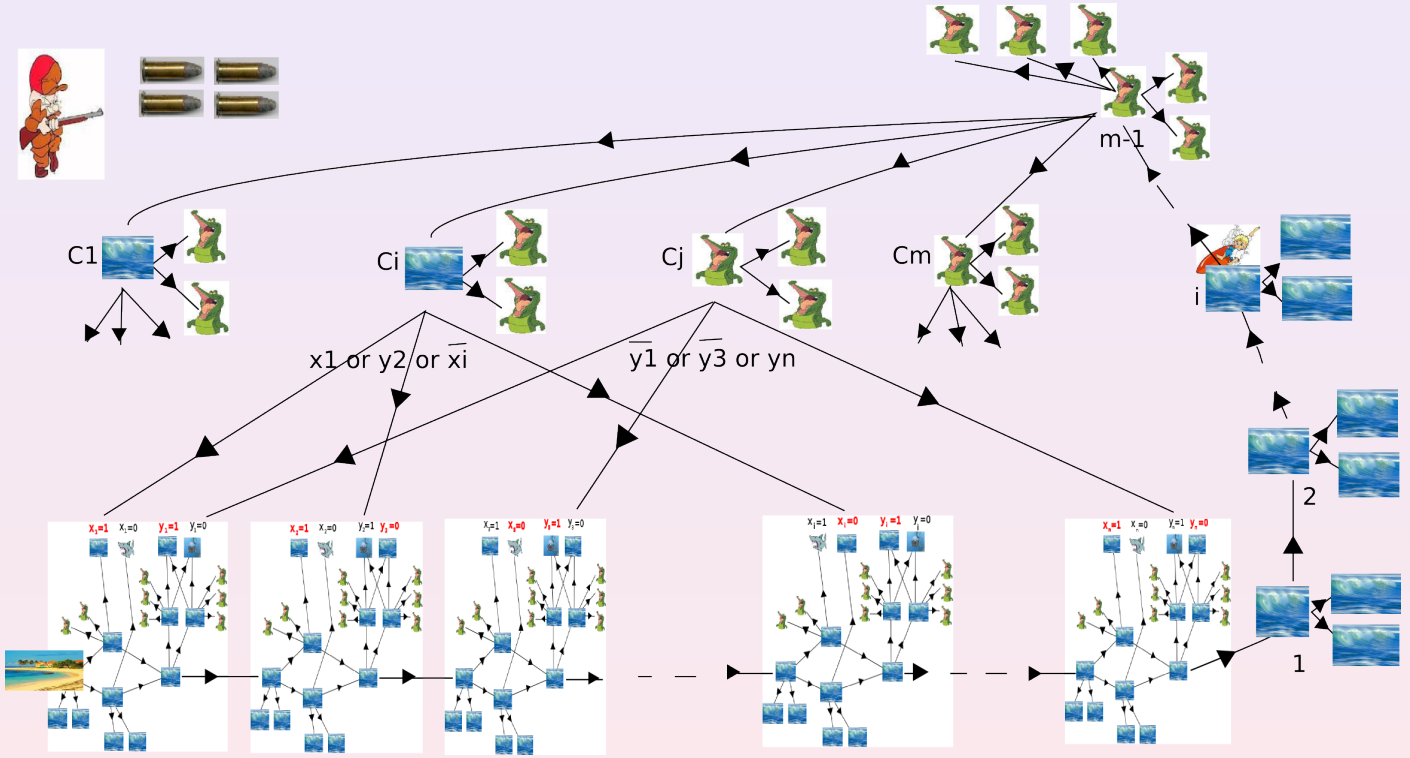


When Surfer reaches the bottom-right, all variables have been assigned: x_i's by Surfer, y_i's by Guard

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$

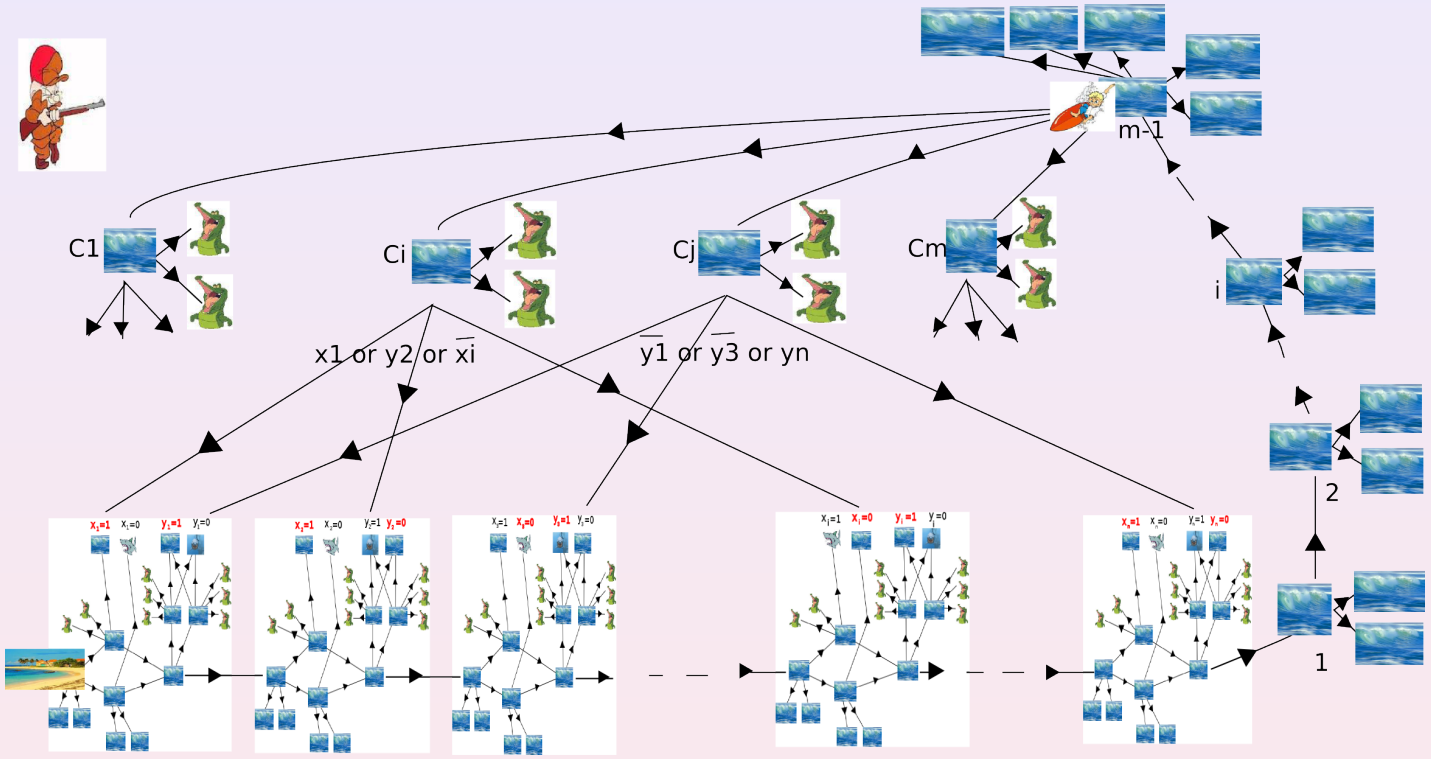


When Surfer reaches the bottom-right, all variables have been assigned: x_i's by Surfer, y_i's by Guard

PSPACE-hardness in DAGs

$sn \leq 4$?

Input: $\{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ and $\Phi(x_1, \cdots, x_n, y_1, \cdots, y_n)$



$$sn(D, v_0) \leq 4 \Leftrightarrow \forall x_1 \cdots \exists y_n \Phi \ true$$

Take-Aways

- **Space Complexity:**
  - Non-determinism not very strong (Savitch)
    - **NL $\subseteq$ SPACE($\log^2$)**
    - **PSPACE = NPSPACE** (padding argument)
- **Complete problems:**
  - **CONNECTIVITY is NL-complete**
  - **QSAT is PSPACE-complete**
- **Hierarchy:**

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME.$$

Take-Aways

- **PSPACE utile pour les jeux :**

	Complexité des Jeux	
Longueur des parties	**Polynomiale**	**Exponentielle**
1 joueur	NP(-c)	PSPACE(-c)
2 joueurs	PSPACE(-c)	EXPTIME(-c)

- Exple: Sokoban et Surfeur.