Deterministic Simulations and Hierarchy Theorems For Randomized Algorithms

Jeff Kinne

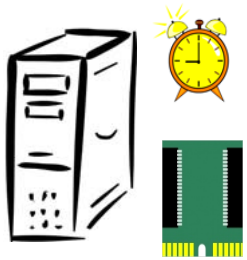
University of Wisconsin-Madison, Department of Computer Sciences

Thesis Defense, April 27, 2010

Does Randomness Add Significant Power?

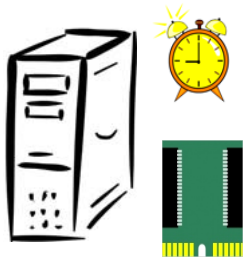
Does Randomness Add Significant Power?

Running Time /
Memory Space



Does Randomness Add Significant Power?

Running Time /
Memory Space

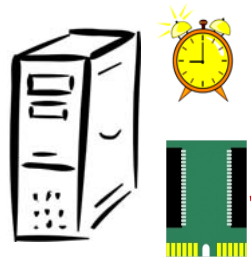


primality testing

undirected connectivity

Does Randomness Add Significant Power?

Running Time /
Memory Space

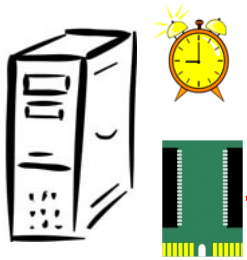


primality testing

undirected connectivity

Does Randomness Add Significant Power?

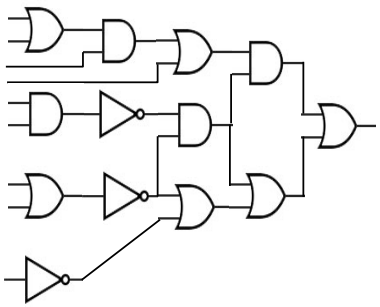
Running Time / Memory Space



~~primality testing~~

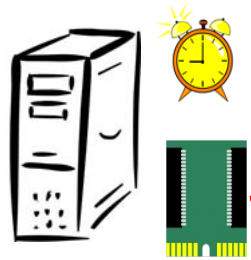
~~undirected connectivity~~

Circuits



Does Randomness Add Significant Power?

Running Time / Memory Space

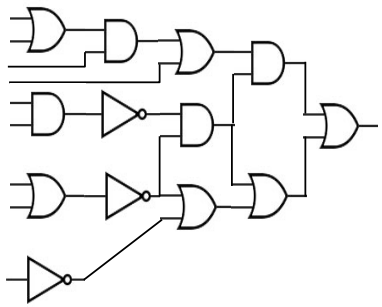


~~primality testing~~

~~undirected connectivity~~

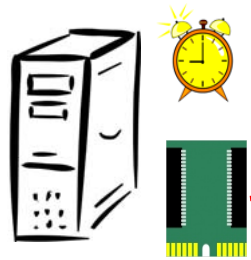
Circuits

No



Does Randomness Add Significant Power?

Running Time / Memory Space

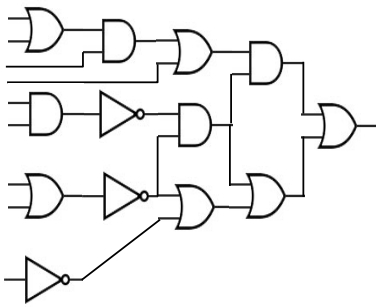


~~primality testing~~

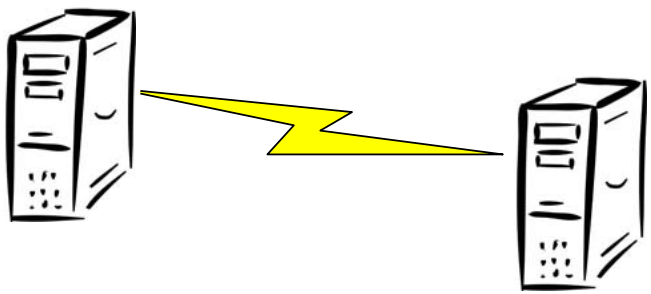
~~undirected connectivity~~

Circuits

No

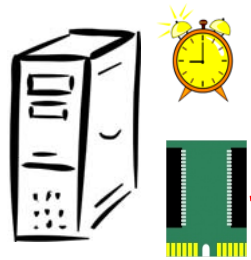


Communication complexity



Does Randomness Add Significant Power?

Running Time / Memory Space

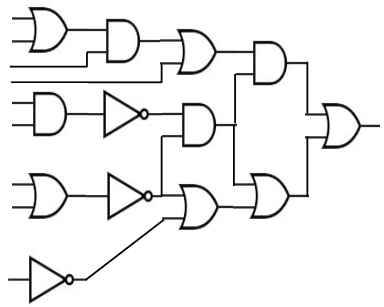


~~primality testing~~

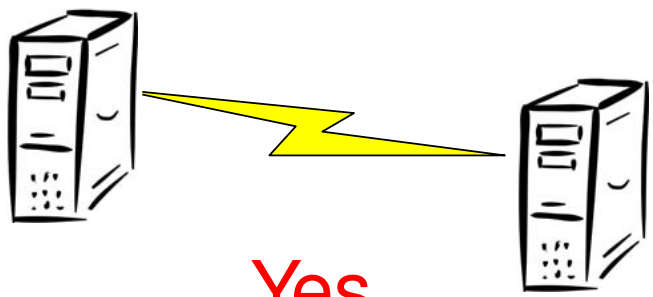
~~undirected connectivity~~

Circuits

No



Communication complexity

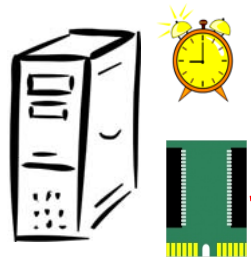


Yes

equality

Does Randomness Add Significant Power?

Running Time / Memory Space

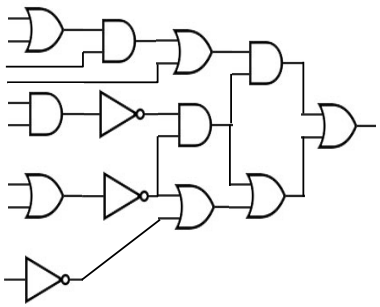


~~primality testing~~

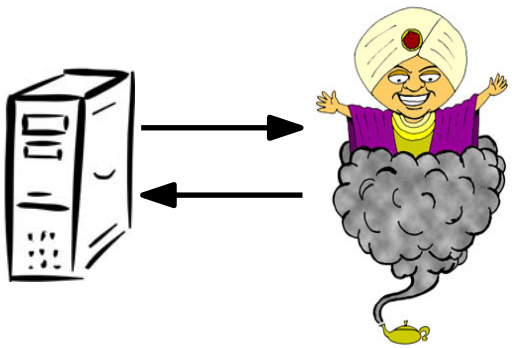
~~undirected connectivity~~

Circuits

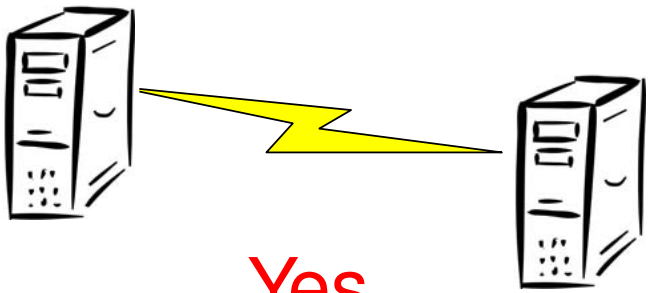
No



Interactive proofs



Communication complexity

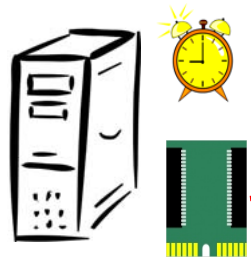


Yes

equality

Does Randomness Add Significant Power?

Running Time / Memory Space

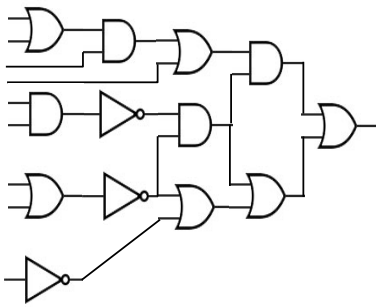


~~primality testing~~

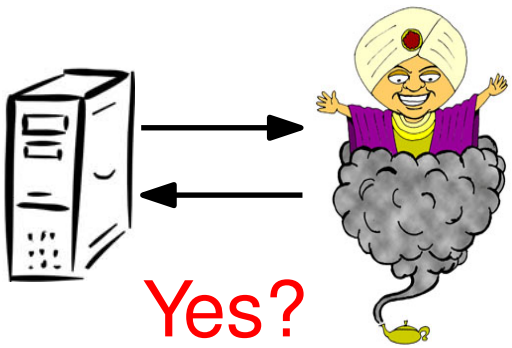
~~undirected connectivity~~

Circuits

No



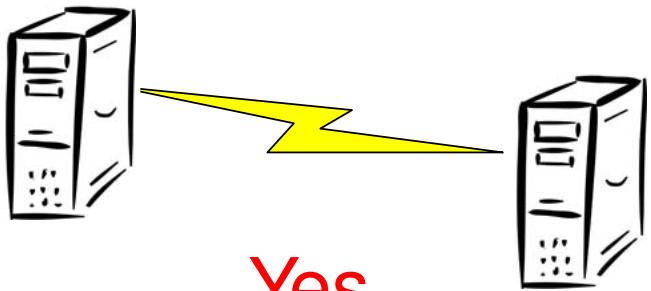
Interactive proofs



Yes?

quantified Boolean formula

Communication complexity

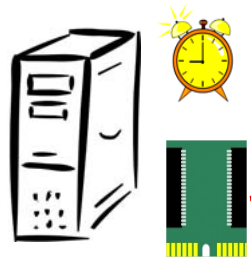


Yes

equality

Does Randomness Add Significant Power?

Running Time / Memory Space

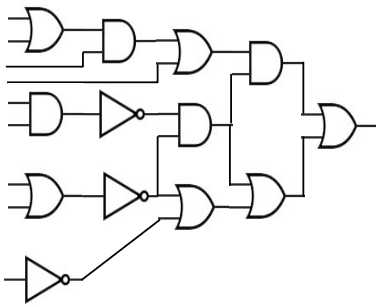


~~primality testing~~

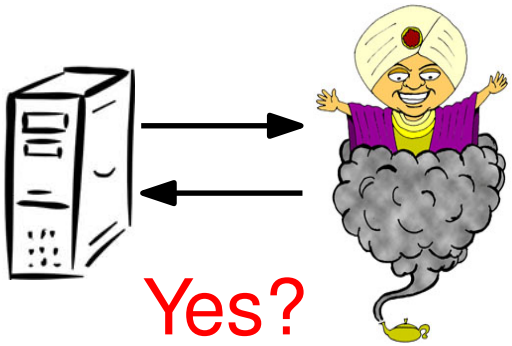
~~undirected connectivity~~

Circuits

No



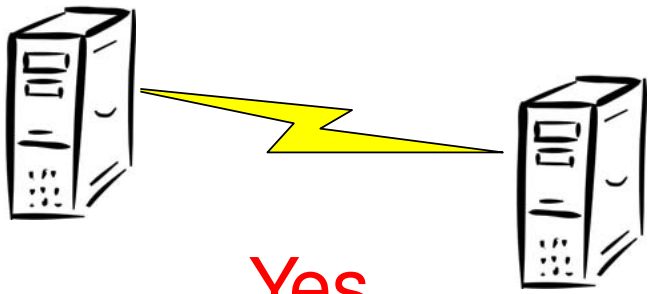
Interactive proofs



Yes?

quantified Boolean formula

Communication complexity



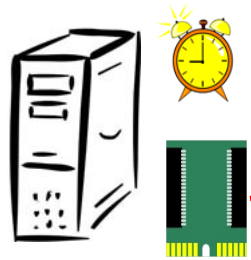
Yes

equality

Does Randomness Add Significant Power?

Running Time / Memory Space

No?

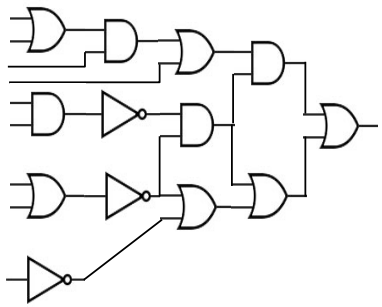


~~primality testing~~

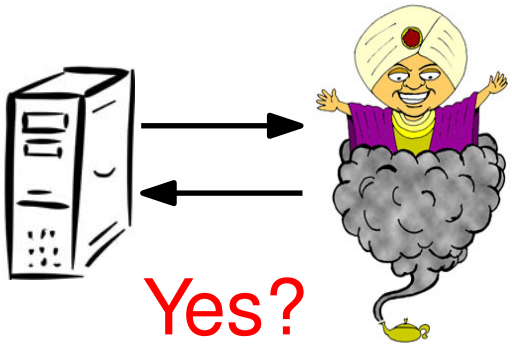
~~undirected connectivity~~

Circuits

No



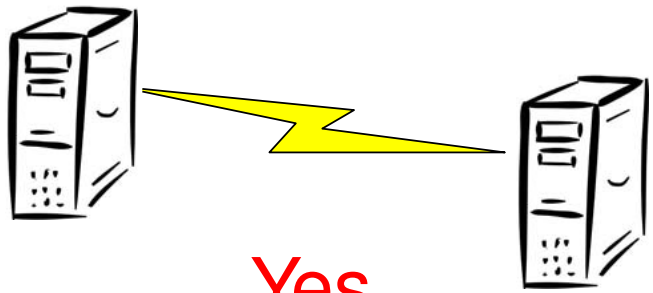
Interactive proofs



Yes?

quantified Boolean formula

Communication complexity



Yes

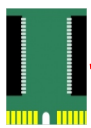
equality

Does Randomness Add Significant Power?

Running Time / Memory Space

No?

poly identity testing

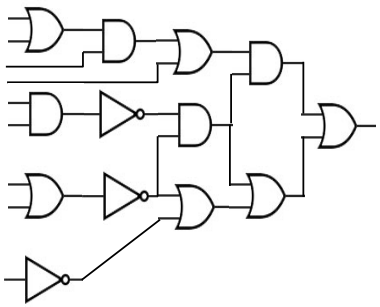


~~primality testing~~

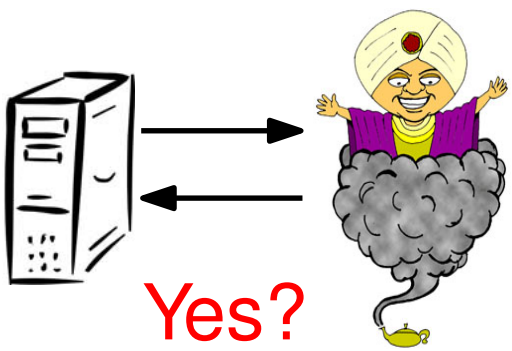
~~undirected connectivity~~

Circuits

No



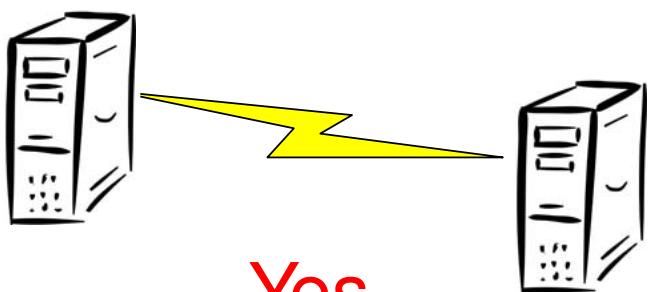
Interactive proofs



Yes?

quantified Boolean formula

Communication complexity



Yes

equality

Outline

Outline

How Powerful is Randomness?

Outline

How Powerful is Randomness?

- "Typically-correct" Derandomization

Outline

How Powerful is Randomness?

- "Typically-correct" Derandomization
- Hierarchy Theorems for Randomized Algorithms

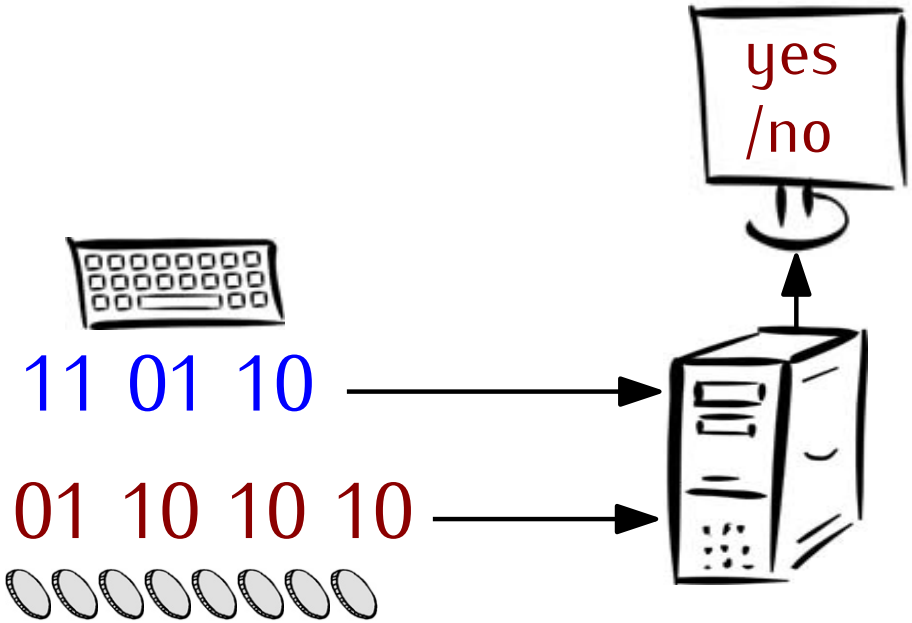
Outline

How Powerful is Randomness?

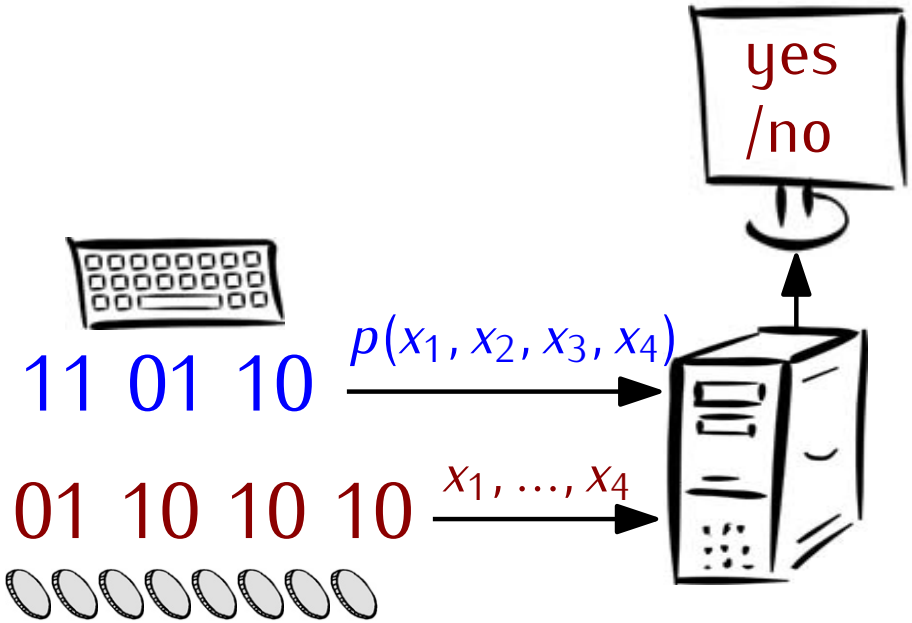
- "Typically-correct" Derandomization
- Hierarchy Theorems for Randomized Algorithms
- Derandomizing Monotone Computations

Randomized Algorithm

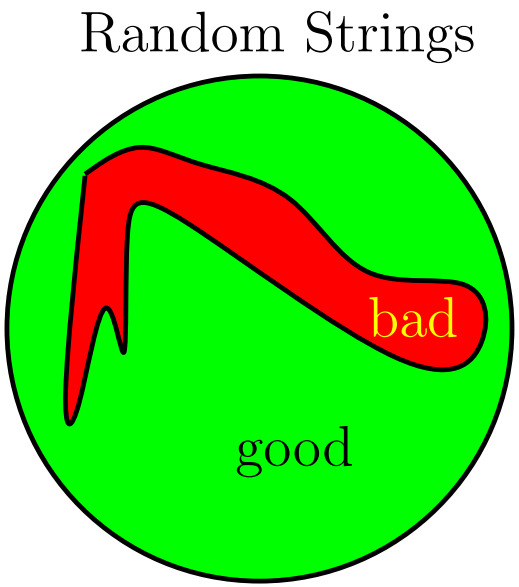
Randomized Algorithm

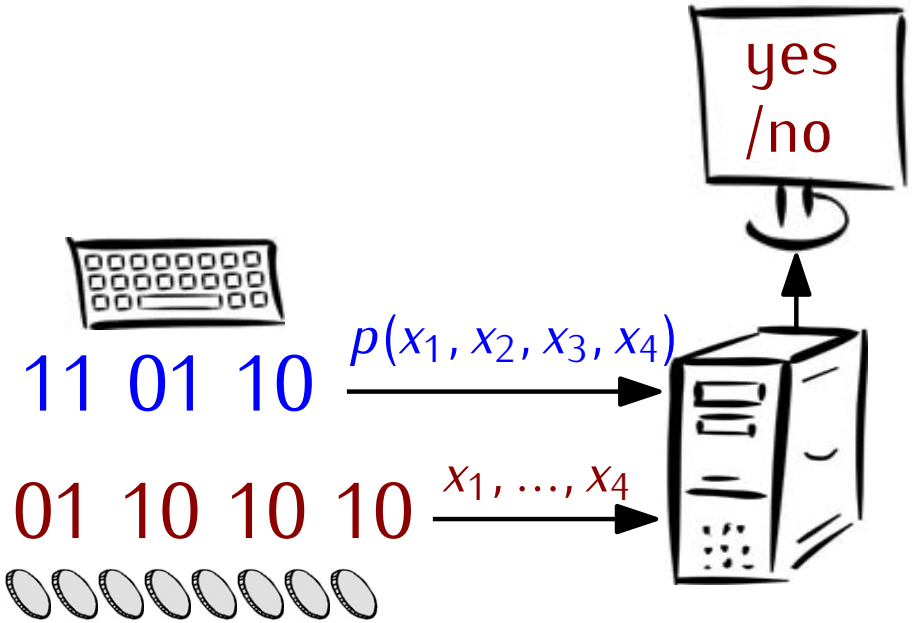


Randomized Algorithm

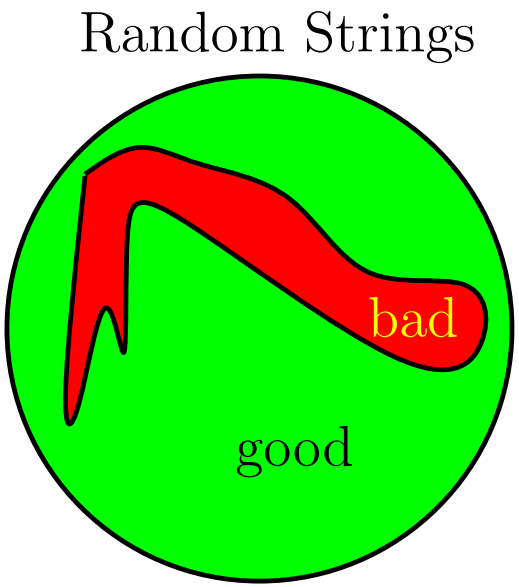


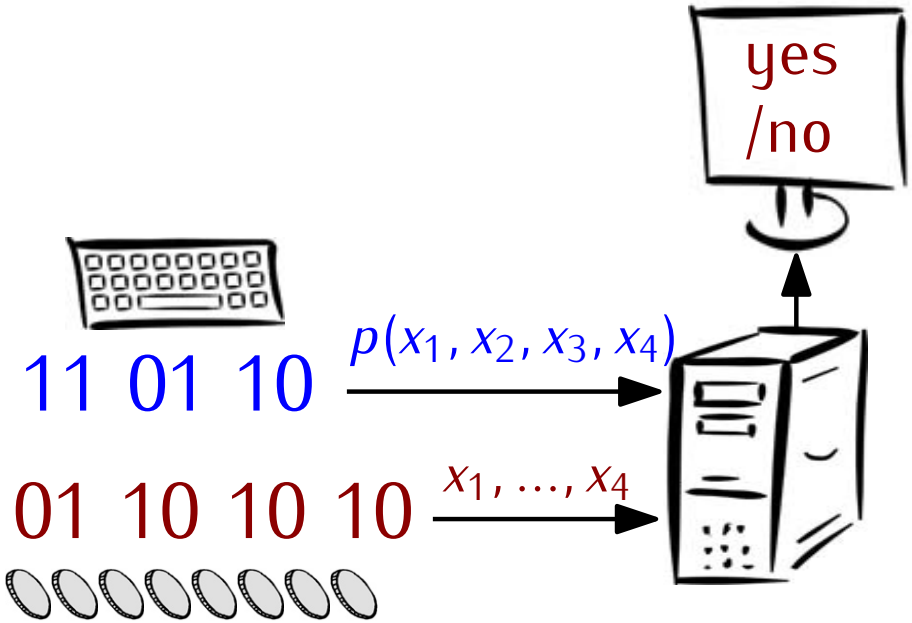
Randomized Algorithm





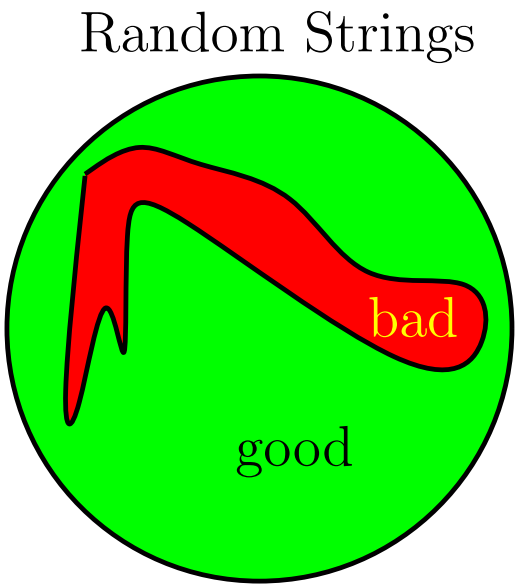
Randomized Algorithm

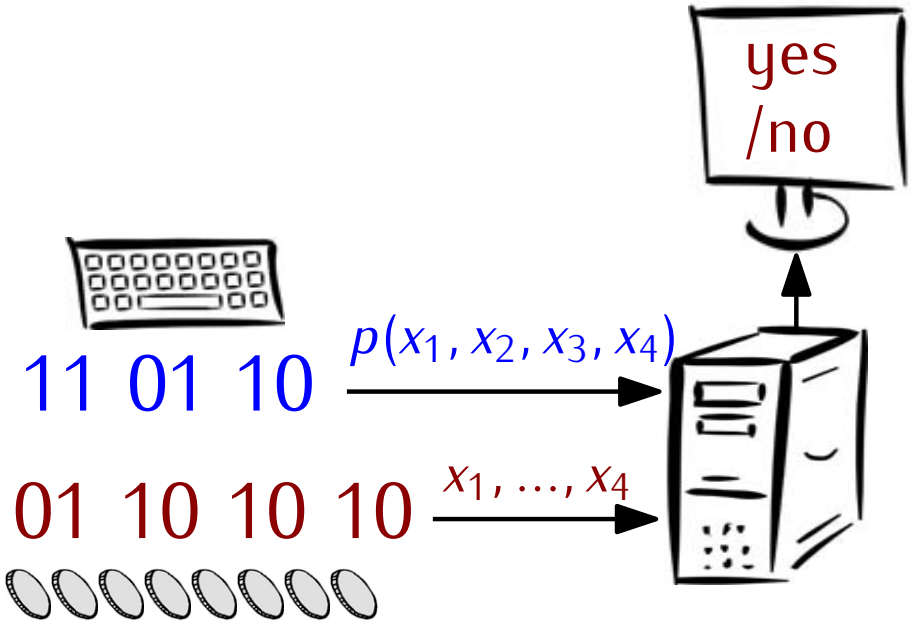




- Bounded error: For every fixed input, correct with Pr $>$ 99%

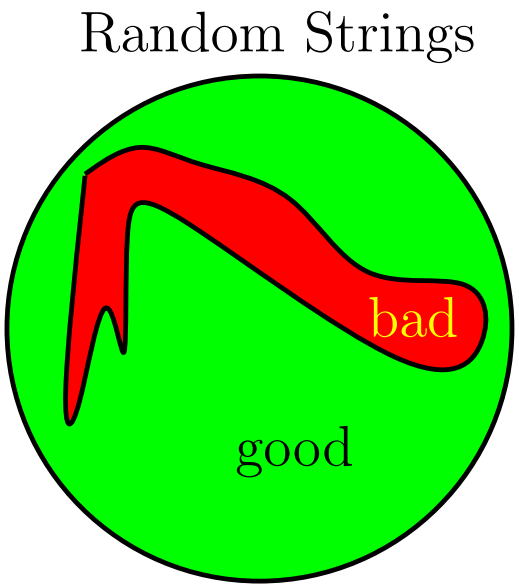
Randomized Algorithm

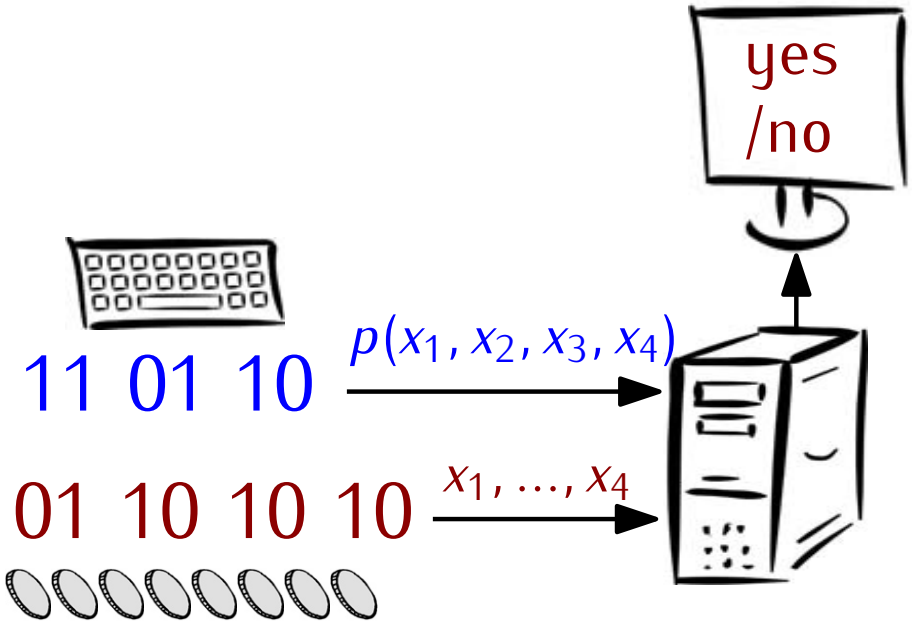




- Bounded error: For every fixed input, correct with Pr $> 99\%$
- **BPP**: Bounded-error Probabilistic Poly time

Randomized Algorithm



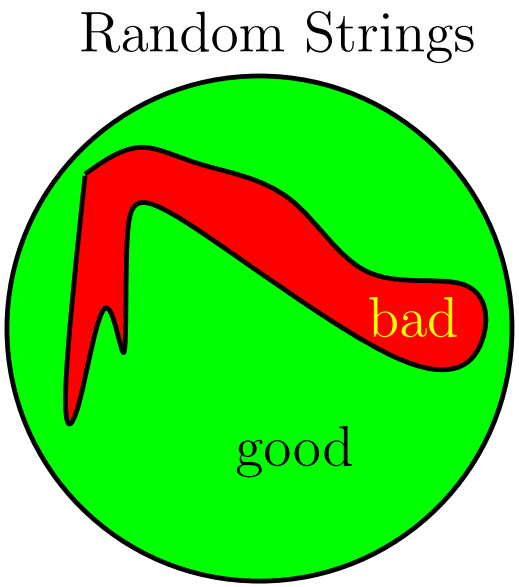


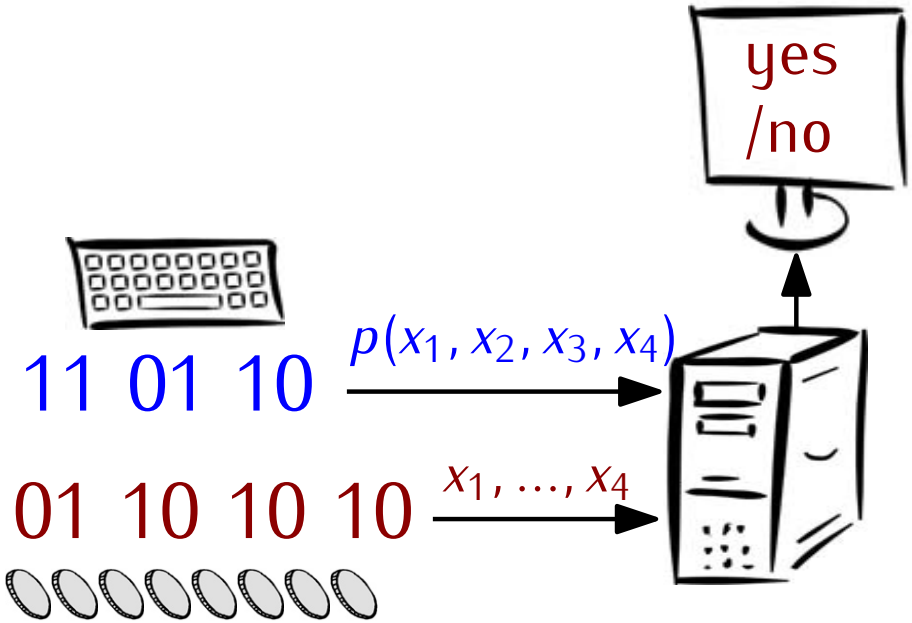
- Bounded error: For every fixed input, correct with Pr $> 99\%$
- **BPP**: Bounded-error Probabilistic Poly time **BPL**: Log space

Typically-Correct Derandomization

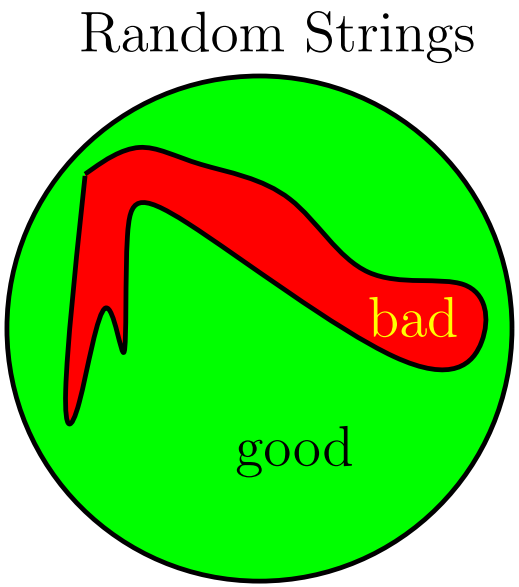
Naive Derandomization

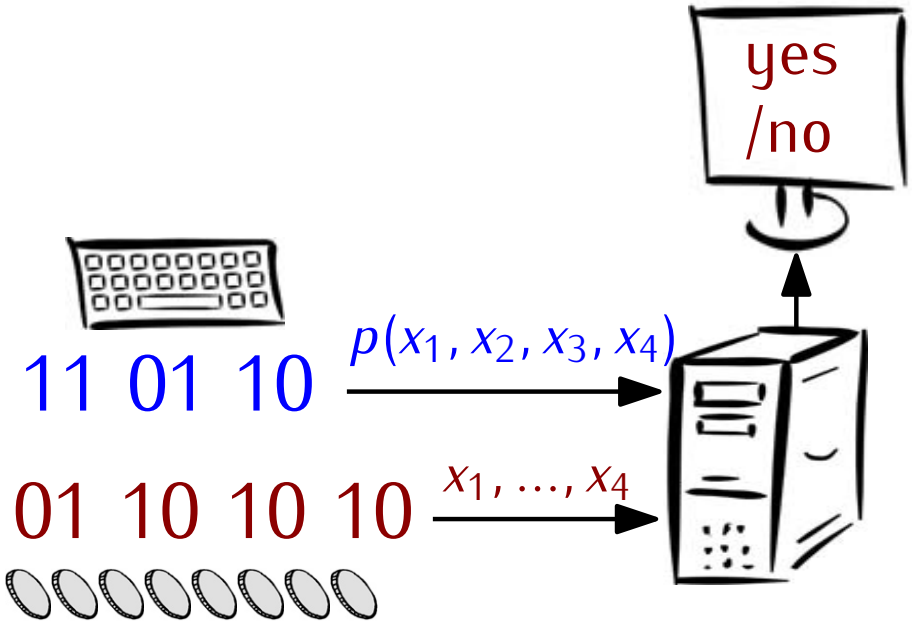
Naive Derandomization





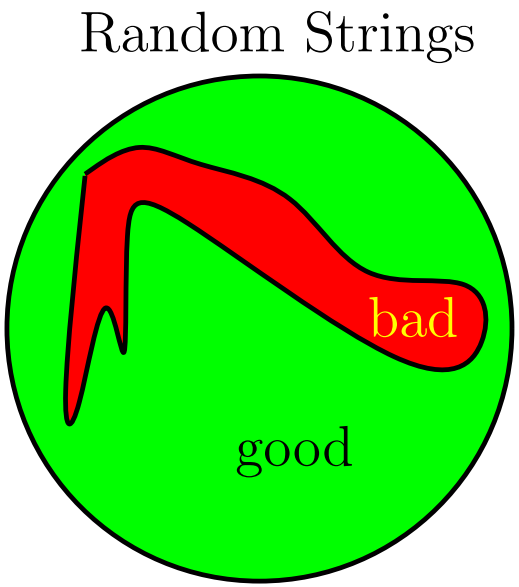
Naive Derandomization

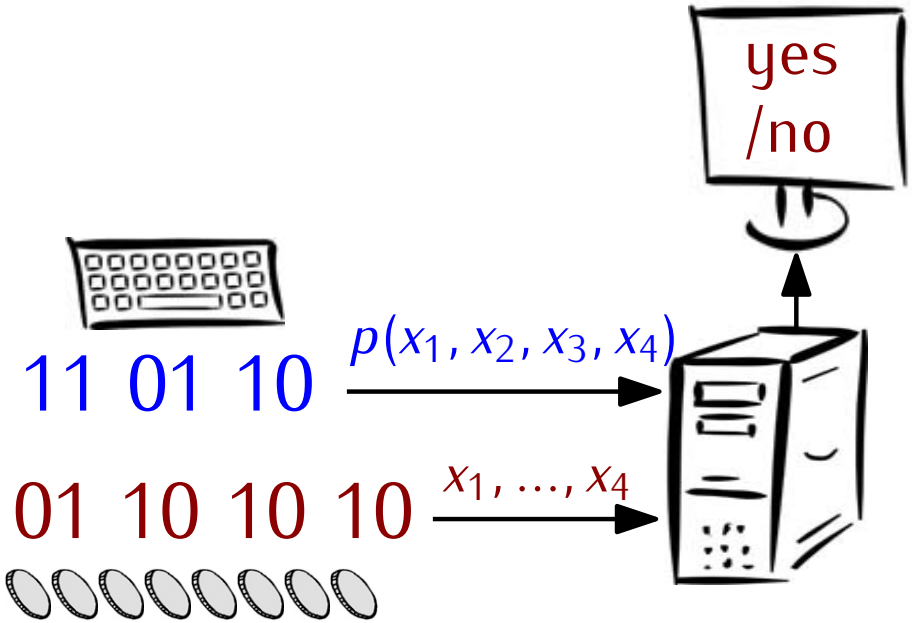




Try all possible random bit strings

Naive Derandomization

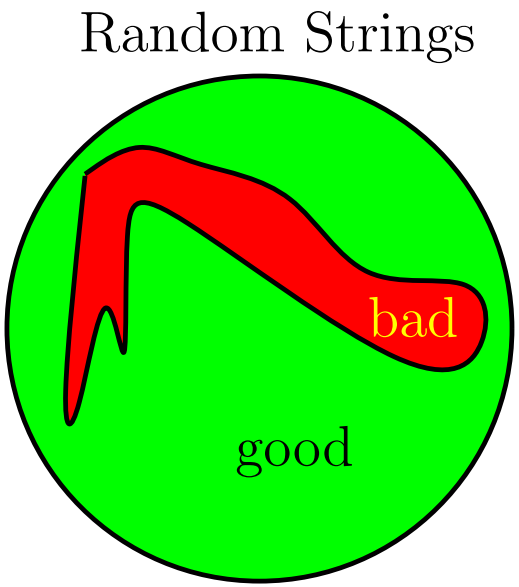


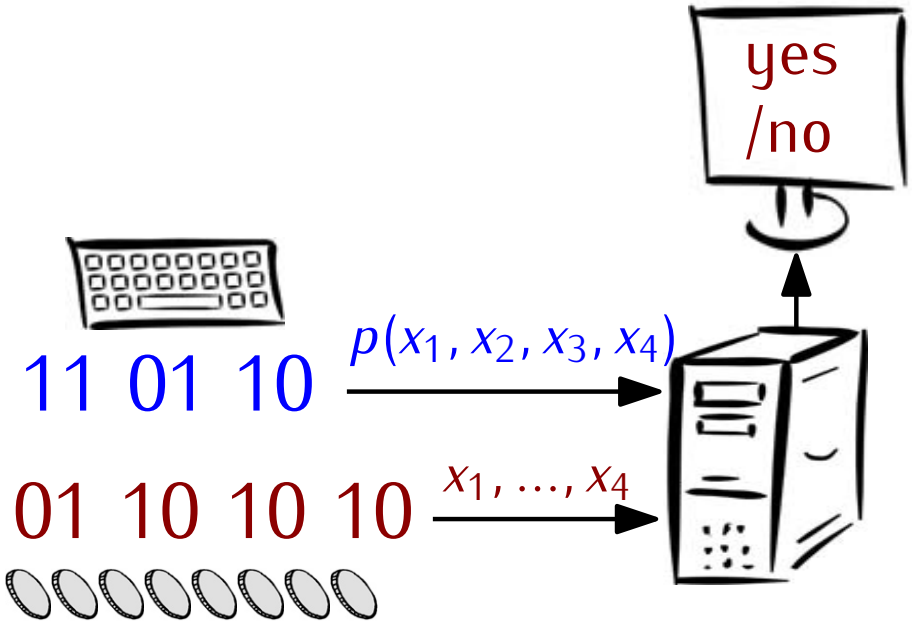


Try all possible random bit strings – exponentially many

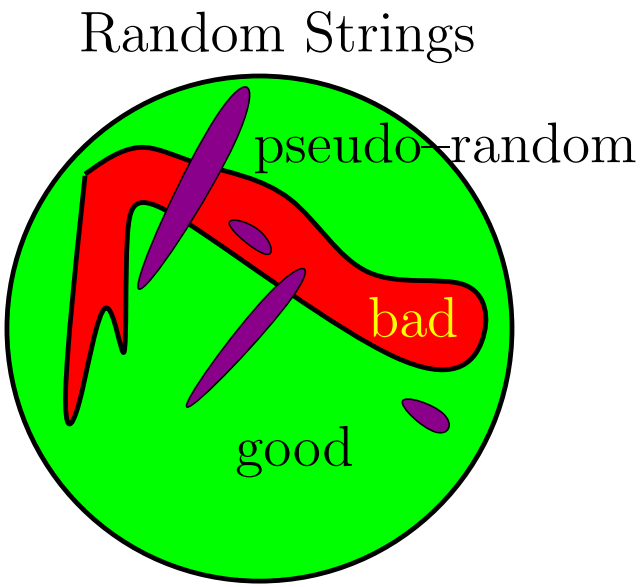
Derandomization – the Standard PRG Approach

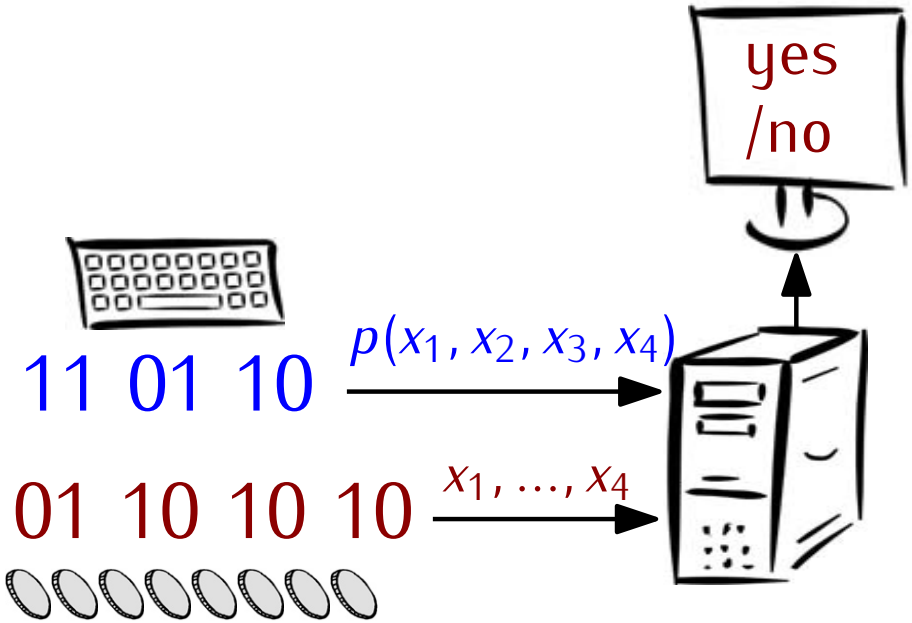
Derandomization – the Standard PRG Approach



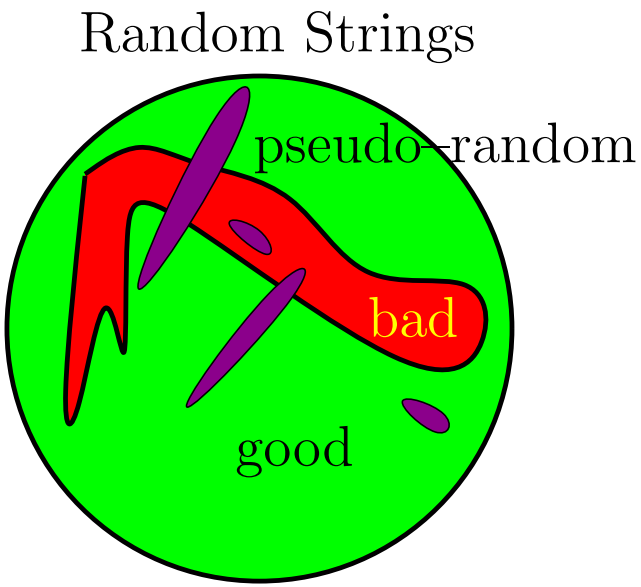


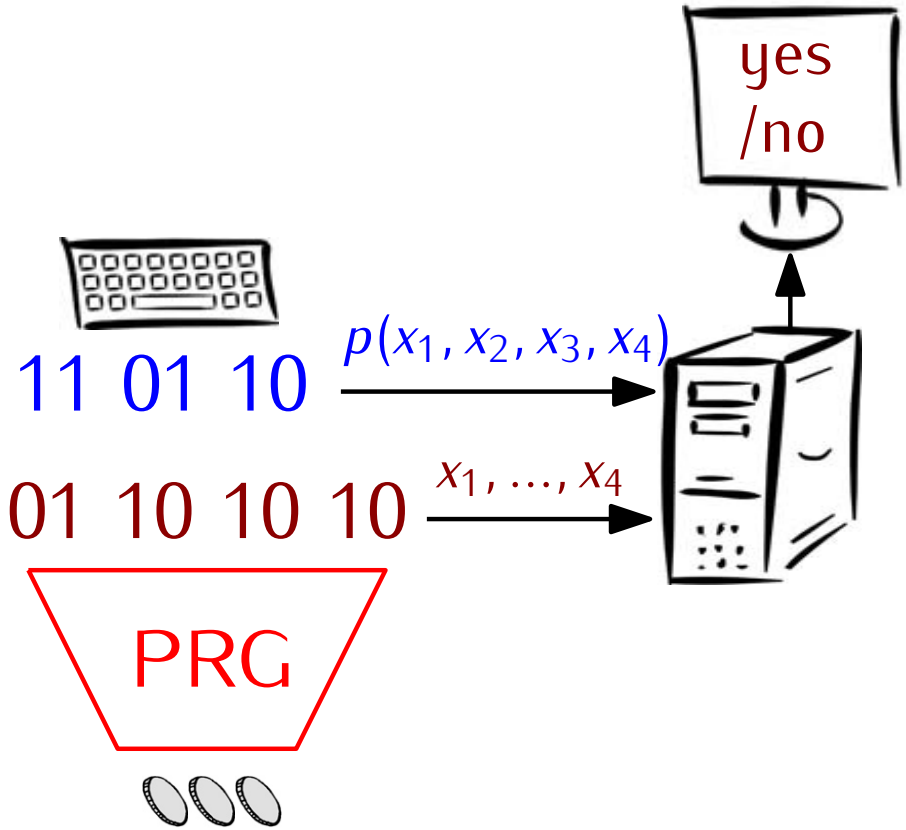
Derandomization – the Standard PRG Approach



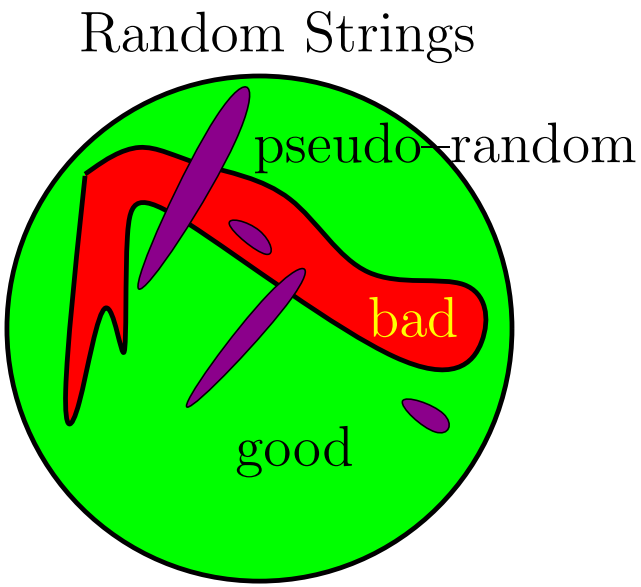


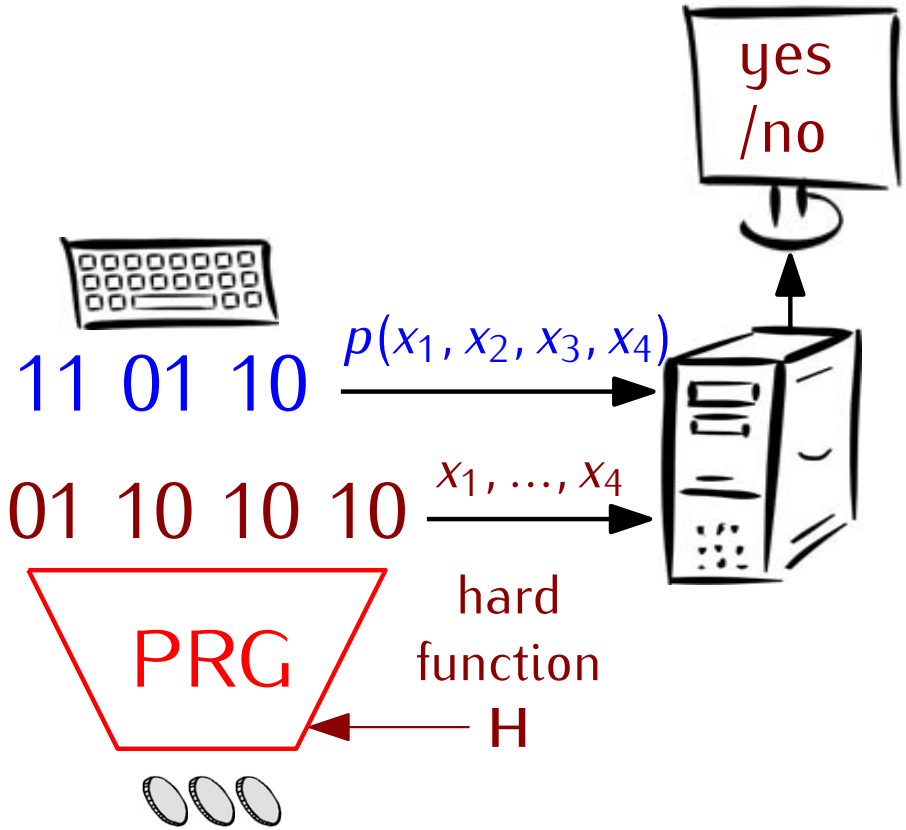
Derandomization – the Standard PRG Approach



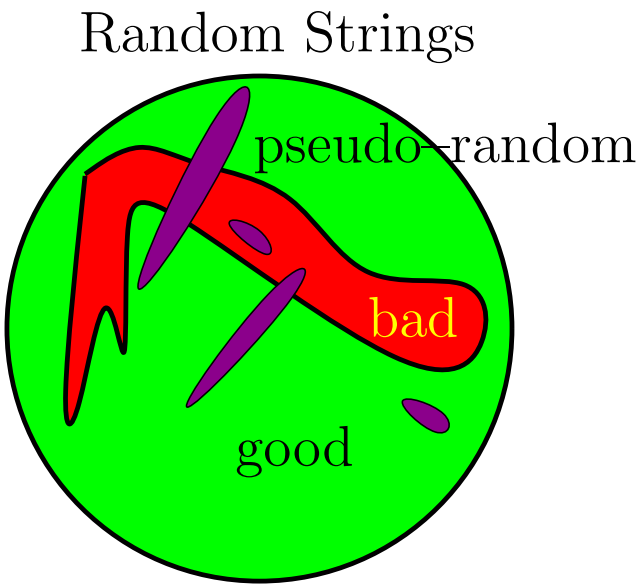


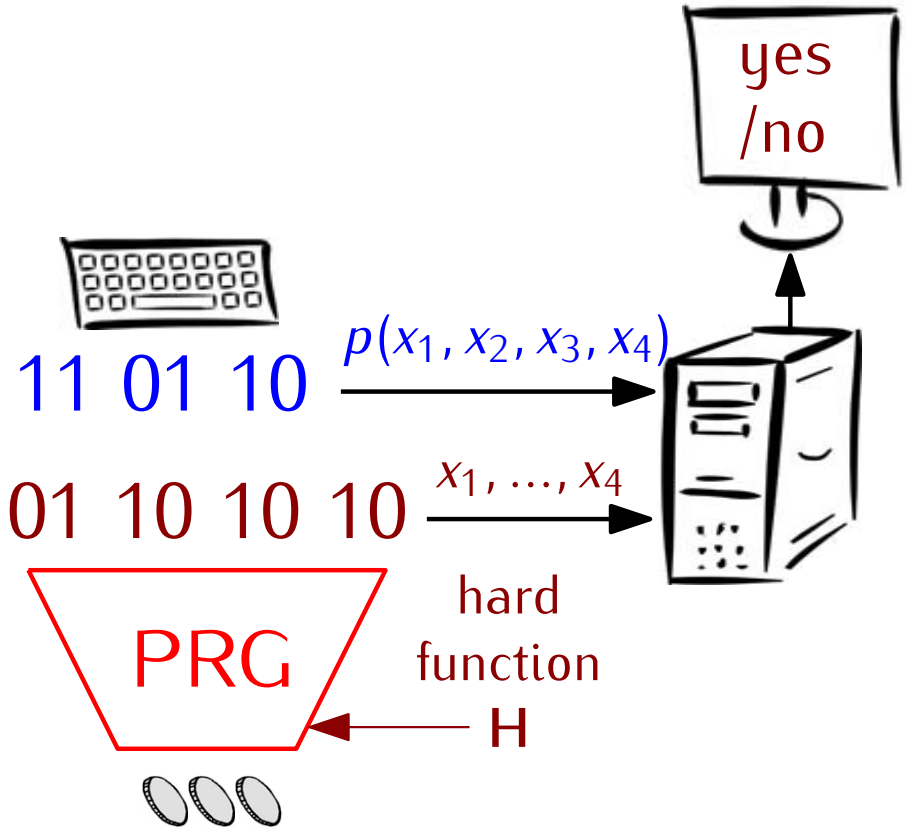
Derandomization – the Standard PRG Approach





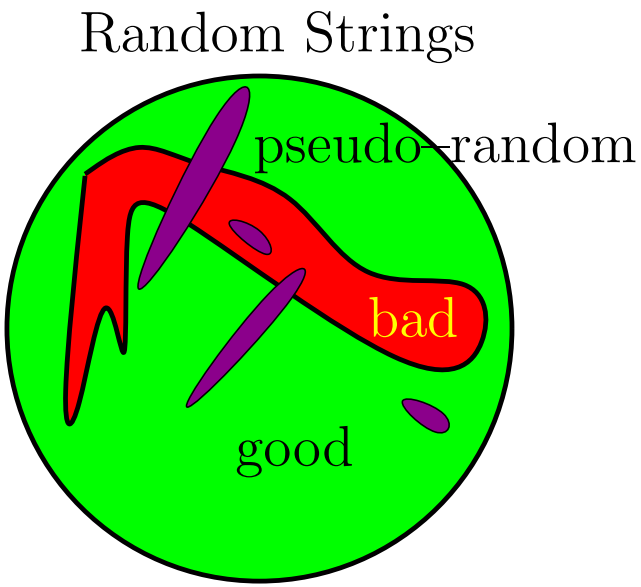
Derandomization – the Standard PRG Approach

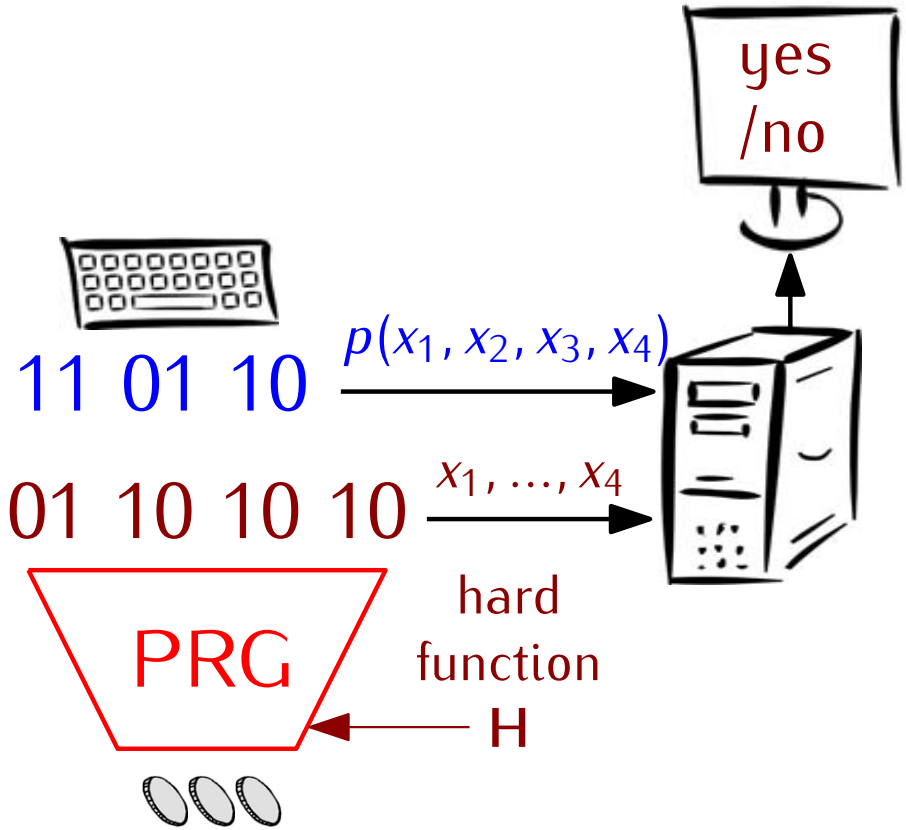




Poly many strings to try

Derandomization – the Standard PRG Approach

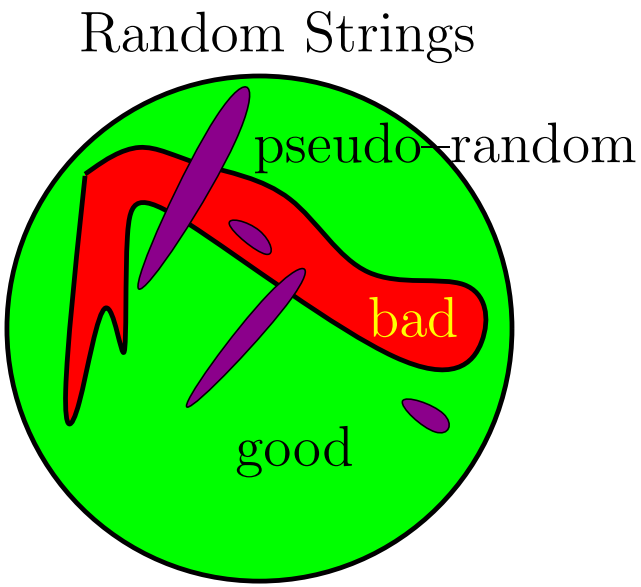


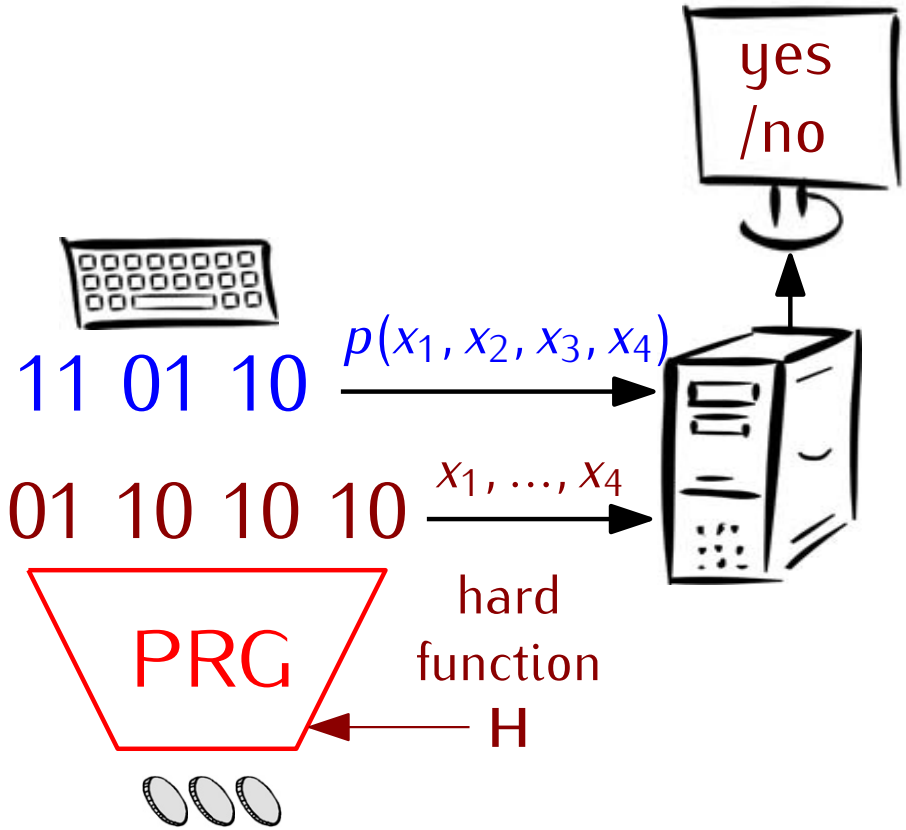


Poly many strings to try $\Rightarrow$ O(log n) seed, exp stretch

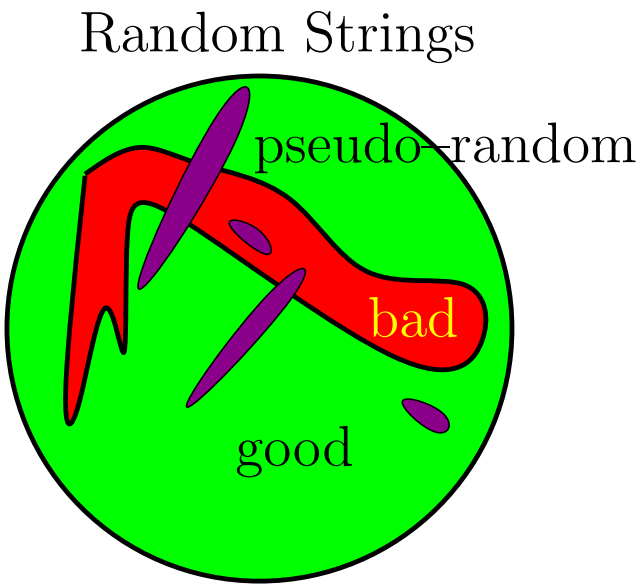
A New Approach – Typically-Correct Derandomization

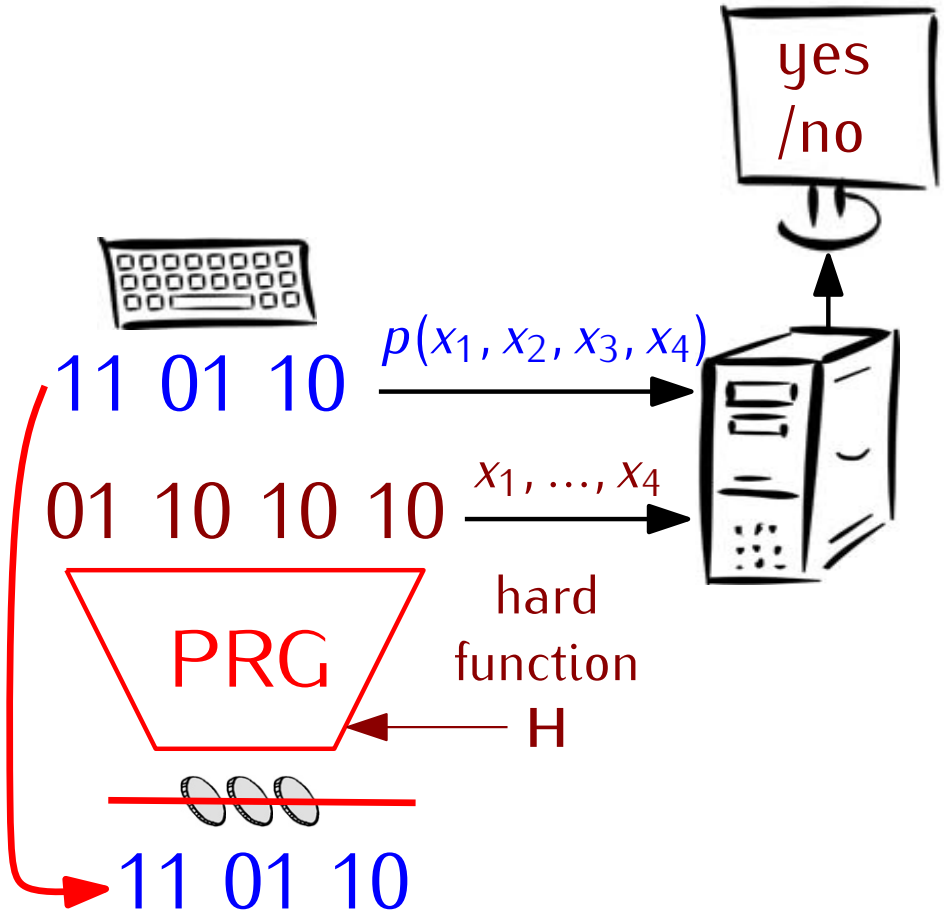
A New Approach – Typically-Correct Derandomization



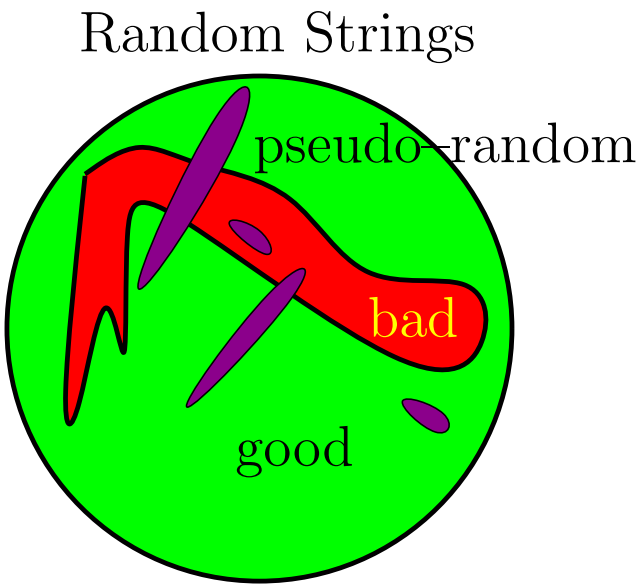


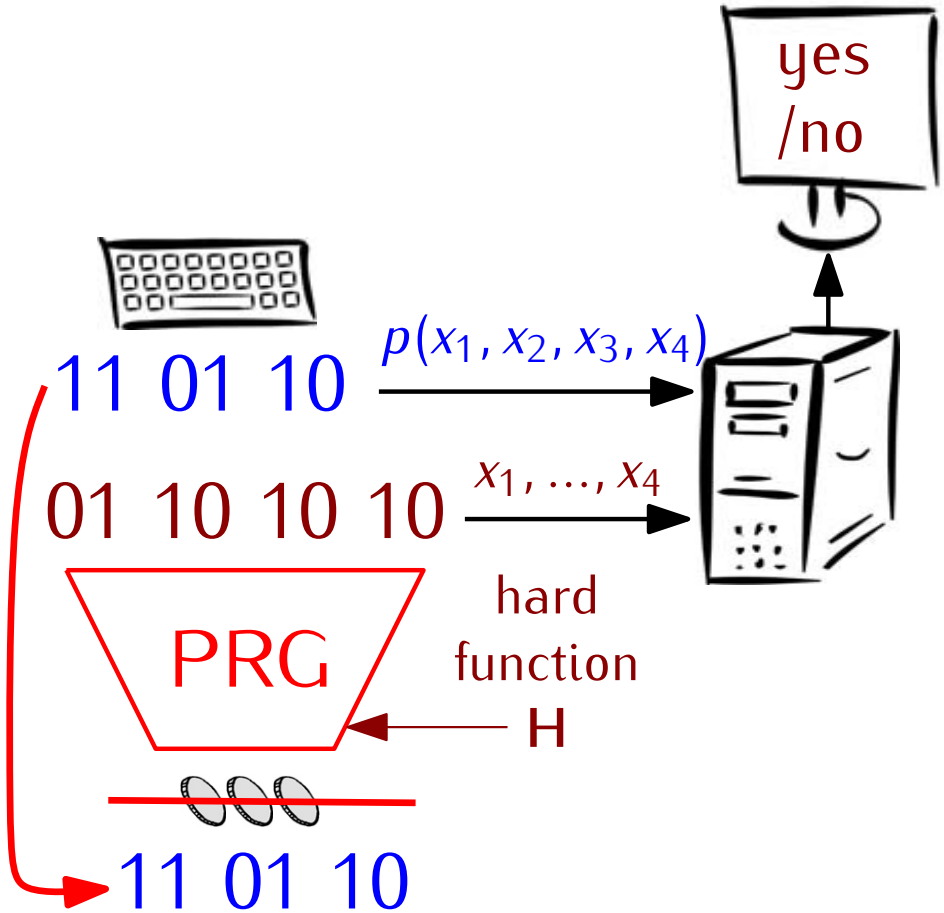
A New Approach – Typically-Correct Derandomization





A New Approach – Typically-Correct Derandomization





Seed length n, poly stretch

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L

Deterministic simulation $D(x) = M(x, E(x))$

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L
Deterministic simulation D(x) = M(x, E(x))

- E pseudorandom even with seed revealed

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L
Deterministic simulation D(x) = M(x, E(x)) **= M(G(x))**

- E pseudorandom even with seed revealed
- G "seed-extending" PRG, G(x) = (x, E(x))

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L
Deterministic simulation D(x) = M(x, E(x)) **= M(G(x))**

- E pseudorandom even with seed revealed
- G "seed-extending" PRG, G(x) = (x, E(x))

G PRG for tests checking M(x,r) = L(x)

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L
Deterministic simulation D(x) = M(x, E(x)) = **M(G(x))**

- E pseudorandom even with seed revealed
- G "seed-extending" PRG, G(x) = (x, E(x))

G PRG for tests checking M(x,r) = L(x) ⇒
$\mathbf{Pr}_X[\mathbf{M(G(X)) = L(X)}] \approx \mathbf{Pr}_{X,\,R}[\mathbf{M(X,R) = L(X)}]$

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L
Deterministic simulation D(x) = M(x, E(x)) = **M(G(x))**

- E pseudorandom even with seed revealed
- G "seed-extending" PRG, G(x) = (x, E(x))

G PRG for tests checking M(x,r) = L(x) ⇒
$\mathbf{Pr}_X[\mathbf{M(G(X)) = L(X)}] \approx \mathbf{Pr}_{X,\,R}[\mathbf{M(X,R) = L(X)}] \geq 1 - \rho$

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L
Deterministic simulation D(x) = M(x, E(x)) = **M(G(x))**

- E pseudorandom even with seed revealed
- G "seed-extending" PRG, G(x) = (x, E(x))

G PRG for tests checking M(x,r) = L(x) ⇒
$\mathbf{Pr}_X[\mathbf{M(G(X)) = L(X)}] \approx \mathbf{Pr}_{X,\,R}[\mathbf{M(X,R) = L(X)}] \geq 1 - \rho$

Seed-extending PRG?

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L
Deterministic simulation D(x) = M(x, E(x)) = **M(G(x))**

- E pseudorandom even with seed revealed
- G "seed-extending" PRG, G(x) = (x, E(x))

G PRG for tests checking M(x,r) = L(x) ⇒
$\mathbf{Pr}_X[\mathbf{M(G(X)) = L(X)}] \approx \mathbf{Pr}_{X,\,R}[\mathbf{M(X,R) = L(X)}] \geq 1 - \rho$

Seed-extending PRG?

Cryptographic – No!

A New Approach – Typically-Correct Derandomization

Randomized algorithm M(x, r) computing language L
Deterministic simulation D(x) = M(x, E(x)) = **M(G(x))**

- E pseudorandom even with seed revealed
- G "seed-extending" PRG, G(x) = (x, E(x))

G PRG for tests checking M(x,r) = L(x) ⇒
$\mathbf{Pr}_X[\mathbf{M(G(X)) = L(X)}] \approx \mathbf{Pr}_{X,\,R}[\mathbf{M(X,R) = L(X)}] \geq 1 - \rho$

Seed-extending PRG?

Cryptographic – No!
Derandomization – Yes! [NW, ...]

Standard Use of PRG's vs. Typ-Correct

Standard Use of PRG's vs. Typ-Correct

Standard Derandomization	**Typ-Correct Derandomization**
[Nisan & Wigderson, ...]	

Standard Use of PRG's vs. Typ-Correct

Standard Derandomization [Nisan & Wigderson, ...]	**Typ-Correct Derandomization**
• Always correct	• Small # mistakes

Standard Use of PRG's vs. Typ-Correct

Standard Derandomization [Nisan & Wigderson, ...]	**Typ-Correct Derandomization**
• Always correct	• Small # mistakes
• Run PRG many times	• Run PRG only once

Standard Use of PRG's vs. Typ-Correct

Standard Derandomization [Nisan & Wigderson, ...]	**Typ-Correct Derandomization**
• Always correct	• Small # mistakes
• Run PRG many times	• Run PRG only once
• Need exponential stretch	• Need only poly stretch

Our Results

Our Results

Randomized algorithm M(x, r) computing language L

Our Results

Randomized algorithm M(x, r) computing language L

Deterministic simulation: $D(x) = M(x, NW_H(x))$

NW_H based on hardness of H

Our Results

Randomized algorithm M(x, r) computing language L

Deterministic simulation: $D(x) = M(x, NW_H(x))$

NW_H based on hardness of H

- New conditional typically-correct derandomizations

Our Results

Randomized algorithm M(x, r) computing language L

Deterministic simulation: $D(x) = M(x, NW_H(x))$

NW_H based on hardness of H

- New conditional typically-correct derandomizations
- New unconditional typically-correct derandomizations

Deterministic Poly-time Simulations of BPP

Deterministic Poly-time Simulations of BPP

H δ-hard for SIZE(s)

Deterministic Poly-time Simulations of BPP

H δ-hard for SIZE(s)

Circuit C, size $\leq$ s

Deterministic Poly-time Simulations of BPP

H δ-hard for SIZE(s)

Circuit C, size $\leq$ s $\Rightarrow \Pr_{X \in \{0,1\}^n}[C(X) \neq H(X)] \geq \delta(n)$

Deterministic Poly-time Simulations of BPP

H δ-hard for SIZE(s)

Circuit C, size $\leq$ s $\Rightarrow \Pr_{X\in\{0,1\}^n}[C(X)\neq H(X)]\geq \delta(n)$

	Hardness Assumption $\rightarrow$	# mistakes
[NW, IW]	$E \not\subseteq SIZE(2^{\epsilon n})$	0
[GW]	P is 1/3-hard for $SIZE^{SAT}(n^d)$	2^{n^ϵ}
[Sha]	P is $\frac{1}{2} - 2^{-n^{\Omega(1)}}$-hard for $SIZE(n^d)$	$\frac{2^n}{2^{n^{\Omega(1)}}}$
ours	P is 1/poly-hard for $SIZE(n^d)$	$\frac{2^n}{poly}$

Deterministic Poly-time Simulations of BPP

H δ-hard for SIZE(s)

Circuit C, size $\leq$ s $\Rightarrow \Pr_{X\in\{0,1\}^n}[C(X)\neq H(X)]\geq \delta(n)$

	Hardness Assumption $\rightarrow$	# mistakes
[NW, IW]	$E \not\subseteq SIZE(2^{\epsilon n})$	0
[GW]	P is 1/3-hard for $SIZE^{SAT}(n^d)$	2^{n^ϵ}
[Sha]	P is $\frac{1}{2} - 2^{-n^{\Omega(1)}}$-hard for $SIZE(n^d)$	$\frac{2^n}{2^{n^{\Omega(1)}}}$
ours	P is 1/poly-hard for $SIZE(n^d)$	$\frac{2^n}{poly}$

- Similar conditional results for AM, BPL, ...

New Unconditional Results

New Unconditional Results

AC^0 with few symmetric gates

New Unconditional Results

AC^0 with few symmetric gates

M uses $o(\log^2 n)$ sym gates, error $\rho \leq 1/3$

New Unconditional Results

AC^0 with few symmetric gates

M uses $o(\log^2 n)$ sym gates, error $\rho \leq 1/3 \Rightarrow$
D in AC^0[sym], D(x) = L(x) for all but $\rho + n^{-\omega(1)}$ fraction of x

New Unconditional Results

AC^0 with few symmetric gates

M uses $o(\log^2 n)$ sym gates, error $\rho \leq 1/3 \Rightarrow$
D in AC^0[sym], D(x) = L(x) for all but $\rho + n^{-\omega(1)}$ fraction of x

Other Settings

Multi-party communication protocols

Circuit Lower Bounds and Derandomization

Circuit Lower Bounds and Derandomization

[Kabanets-Impagliazzo]

Circuit Lower Bounds and Derandomization

[Kabanets-Impagliazzo]

BPP $\subseteq$ NSUBEXP $\Rightarrow$

Circuit Lower Bounds and Derandomization

[Kabanets-Impagliazzo]

BPP $\subseteq$ **NSUBEXP** $\Rightarrow$
NEXP $\not\subseteq$ P/poly or PERM $\notin$ Arith-P/poly

Circuit Lower Bounds and Derandomization

[Kabanets-Impagliazzo]

BPP $\subseteq$ NSUBEXP $\Rightarrow$
NEXP $\not\subseteq$ P/poly or PERM $\notin$ Arith-P/poly

- **Does typically-correct derandomization imply circuit lower bounds?**

Circuit Lower Bounds and Derandomization

[Kabanets-Impagliazzo]

BPP $\subseteq$ NSUBEXP $\Rightarrow$
NEXP $\not\subseteq$ P/poly or PERM $\notin$ Arith-P/poly

- **Does typically-correct derandomization imply circuit lower bounds?**
- Yes for small error:

Circuit Lower Bounds and Derandomization

[Kabanets-Impagliazzo]

BPP $\subseteq$ NSUBEXP $\Rightarrow$
NEXP $\not\subseteq$ P/poly or PERM $\notin$ Arith-P/poly

- **Does typically-correct derandomization imply circuit lower bounds?**
- Yes for small error:
  NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors

Circuit Lower Bounds and Derandomization

[Kabanets-Impagliazzo]

BPP $\subseteq$ NSUBEXP $\Rightarrow$
NEXP $\not\subseteq$ P/poly or PERM $\notin$ Arith-P/poly

- **Does typically-correct derandomization imply circuit lower bounds?**
- Yes for small error:
  NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors $\Rightarrow$
  NEXP $\not\subseteq$ P/poly or PERM $\notin$ Arith-P/poly

Circuit Lower Bounds and Derandomization

[Kabanets-Impagliazzo]

BPP $\subseteq$ NSUBEXP $\Rightarrow$
NEXP $\not\subseteq$ P/poly or PERM $\notin$ Arith-P/poly

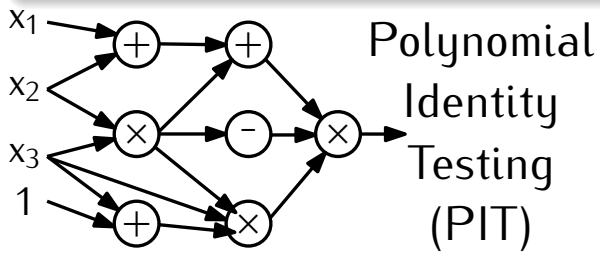
- **Does typically-correct derandomization imply circuit lower bounds?**
- Yes for small error:
  NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors $\Rightarrow$
  NEXP $\not\subseteq$ P/poly or PERM $\notin$ Arith-P/poly
- Simpler proof for everywhere-correct setting

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly

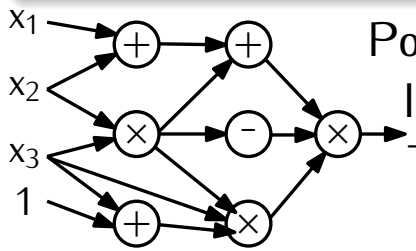
Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly



Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly

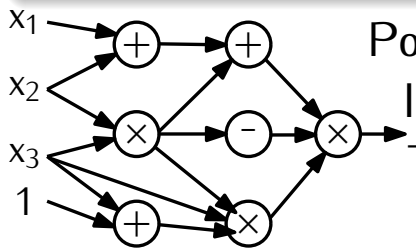


Polynomial Identity Testing (PIT)

$$\text{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly



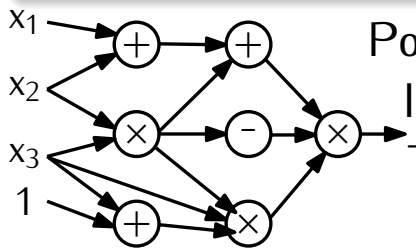
Polynomial Identity Testing (PIT)

$$\mathrm{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- $P^{\mathrm{Perm}} \subseteq$ NSUBEXP (same as [KI])

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly



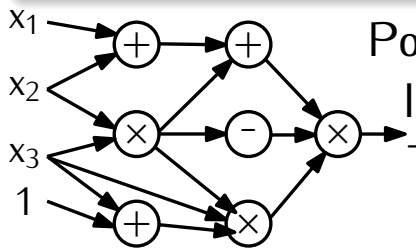
Polynomial Identity Testing (PIT)

$$\text{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- $P^{Perm} \subseteq$ NSUBEXP (same as [KI])
  - Guess arithmetic circuit C

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly



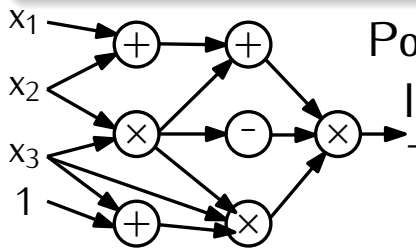
Polynomial Identity Testing (PIT)

$$\mathrm{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- $\mathrm{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP (same as [KI])
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A^*_{i,j})$ for any $i \in [n]$

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly



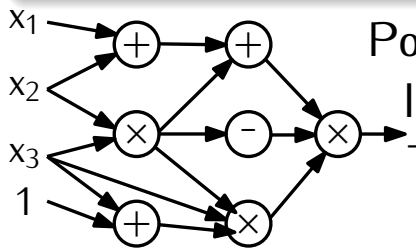
Polynomial Identity Testing (PIT)

$$\mathrm{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- $\mathrm{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP (same as [KI])
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A^*_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly



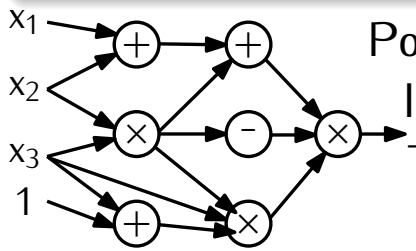
Polynomial Identity Testing (PIT)

$$\mathrm{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- $\mathrm{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP (same as [KI])
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A^*_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm
- $\mathrm{P}^{\mathrm{Perm}} \not\subseteq \mathrm{SIZE}(n^k)$ [Toda, Kannan]

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly



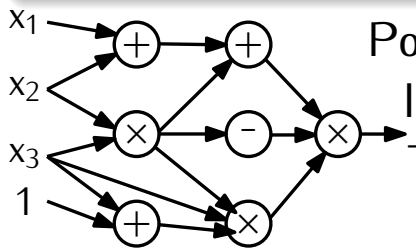
Polynomial Identity Testing (PIT)

$$\mathrm{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- $\mathrm{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP (same as [KI])
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A^*_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm
- $\mathrm{P}^{\mathrm{Perm}} \not\subseteq \mathrm{SIZE}(n^k)$ [Toda, Kannan]
- NSUBEXP $\not\subseteq \mathrm{SIZE}(n^k)$

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly



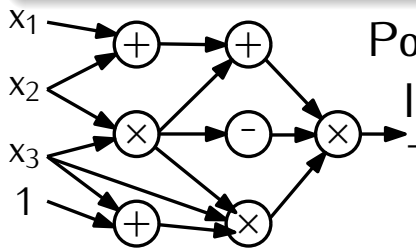
Polynomial Identity Testing (PIT)

$$\mathrm{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- $\mathrm{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP (same as [KI])
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A^*_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm
- $\mathrm{P}^{\mathrm{Perm}} \not\subseteq \mathrm{SIZE}(n^k)$ [Toda, Kannan]
- NSUBEXP $\not\subseteq \mathrm{SIZE}(n^k) \Rightarrow$ NEXP $\not\subseteq \mathrm{SIZE}(\mathrm{poly})$

Derandomization Implies Circuit Lower Bounds

BPP $\subseteq$ NSUBEXP and Perm $\in$ Arith-P/poly
$\Rightarrow$ NEXP $\not\subseteq$ P/poly



Polynomial Identity Testing (PIT)

$$\mathrm{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

- $\mathrm{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP (same as [KI])
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A^*_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm
- $\mathrm{P}^{\mathrm{Perm}} \not\subseteq \mathrm{SIZE}(n^k)$ [Toda, Kannan]
- NSUBEXP $\not\subseteq \mathrm{SIZE}(n^k) \Rightarrow$ NEXP $\not\subseteq$ SIZE(poly)

- What if assumed PIT alg is only typically-correct?

Typically-Correct Derand Implies Circuit Lower Bounds

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- Avoid false positives

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- Avoid false positives
  - Advice: # mistakes s.t. M claims $\equiv 0$ but wrong

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- Avoid false positives
  - Advice: # mistakes s.t. M claims $\equiv 0$ but wrong
  - Guess and verify set of false positives

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- Avoid false positives
  - Advice: # mistakes s.t. M claims $\equiv 0$ but wrong
  - Guess and verify set of false positives
  - Accept if M claims $\equiv 0$ and not among $\uparrow$ mistakes

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- Avoid false positives
  - Advice: # mistakes s.t. M claims $\equiv 0$ but wrong
  - Guess and verify set of false positives
  - Accept if M claims $\equiv 0$ and not among $\uparrow$ mistakes
- Avoid false negatives

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- Avoid false positives
  - Advice: # mistakes s.t. M claims $\equiv 0$ but wrong
  - Guess and verify set of false positives
  - Accept if M claims $\equiv 0$ and not among $\uparrow$ mistakes
- Avoid false negatives
  - Pad PIT language so $> 2^{n^\epsilon}$ duplicate copies of each circuit

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- Avoid false positives
  - Advice: # mistakes s.t. M claims $\equiv 0$ but wrong
  - Guess and verify set of false positives
  - Accept if M claims $\equiv 0$ and not among $\uparrow$ mistakes
- Avoid false negatives
  - Pad PIT language so $> 2^{n^\epsilon}$ duplicate copies of each circuit
  - Guess a duplicate, not all duplicates can be false negatives

Typically-Correct Derand Implies Circuit Lower Bounds

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors and Perm $\in$ Arith-P/poly

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- $\mathrm{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP **with $n^{\epsilon'}$ bits of advice**

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- $\mathrm{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP **with $n^{\epsilon'}$ bits of advice**
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- $\mathsf{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP **with $n^{\epsilon'}$ bits of advice**
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm
- $\mathsf{P}^{\mathrm{Perm}} \not\subseteq \mathrm{SIZE}(n^k)$ [Toda, Kannan]

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- $\mathsf{P}^{\mathrm{Perm}} \subseteq$ NSUBEXP **with $n^{\epsilon'}$ bits of advice**
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm
- $\mathsf{P}^{\mathrm{Perm}} \not\subseteq \mathrm{SIZE}(n^k)$ [Toda, Kannan]
- NSUBEXP **with $n^{\epsilon'}$ bits of advice** $\not\subseteq \mathrm{SIZE}(n^k)$

Typically-Correct Derand Implies Circuit Lower Bounds

NSUBEXP computes BPP with $\leq 2^{n^\epsilon}$ errors and Perm $\in$ Arith-P/poly $\Rightarrow$ NEXP $\not\subseteq$ P/poly

NSUBEXP algorithm M computes PIT with $\leq 2^{n^\epsilon}$ errors
$\Rightarrow$ PIT $\subseteq$ NSUBEXP with n^ϵ advice bits

- $\mathsf{P}^{\mathsf{Perm}} \subseteq$ NSUBEXP **with $n^{\epsilon'}$ bits of advice**
  - Guess arithmetic circuit C
  - $\mathrm{Perm}(A) = \sum_{j=1}^{n} A_{i,j} \cdot \mathrm{Perm}(A_{i,j})$ for any $i \in [n]$
  - Use C in place of Perm in $\uparrow$, check using PIT algorithm
- $\mathsf{P}^{\mathsf{Perm}} \not\subseteq \mathrm{SIZE}(n^k)$ [Toda, Kannan]
- NSUBEXP **with $n^{\epsilon'}$ bits of advice** $\not\subseteq \mathrm{SIZE}(n^k)$
  $\Rightarrow$ NEXP $\not\subseteq$ SIZE(poly)

Typically-Correct Derandomization Recap

Typically-Correct Derandomization Recap

How Powerful is Randomness?

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization
  - **For all of BPP?**

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization
  - **For all of BPP?**
    - Small error rates:

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization
  - **For all of BPP?**
    - Small error rates: implies circuit lower bounds

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization
  - **For all of BPP?**
    - Small error rates: implies circuit lower bounds
    - Larger error rates:

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization
  - **For all of BPP?**
    - Small error rates: implies circuit lower bounds
    - Larger error rates: relativizing techniques and arithmetization alone not enough

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization
  - **For all of BPP?**
    - Small error rates: implies circuit lower bounds
    - Larger error rates: relativizing techniques and arithmetization alone not enough
  - **Applications?**

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization
  - **For all of BPP?**
    - Small error rates: implies circuit lower bounds
    - Larger error rates: relativizing techniques and arithmetization alone not enough
  - **Applications?**
    - Time-space lower bounds

Typically-Correct Derandomization Recap

How Powerful is Randomness? P vs. BPP, L vs. BPL

- Conjecture: P=BPP, L=BPL
- "Typically-correct" Derandomization
  - **For all of BPP?**
    - Small error rates: implies circuit lower bounds
    - Larger error rates: relativizing techniques and arithmetization alone not enough
  - **Applications?**
    - Time-space lower bounds
    - Hierarchy theorems for randomized algorithms

Outline

How Powerful is Randomness?

- "Typically-correct" Derandomization
- Hierarchy Theorems for Randomized Algorithms
- Derandomizing Monotone Computations

Hierarchy Theorems

Hierarchy Theorems

Hierarchy Theorems

- Fix a model of computing

Hierarchy Theorems

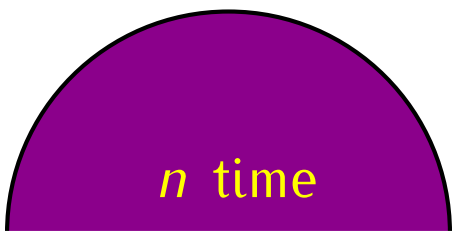
- Fix a model of computing
  (deterministic, randomized, nondeterministic)

Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**

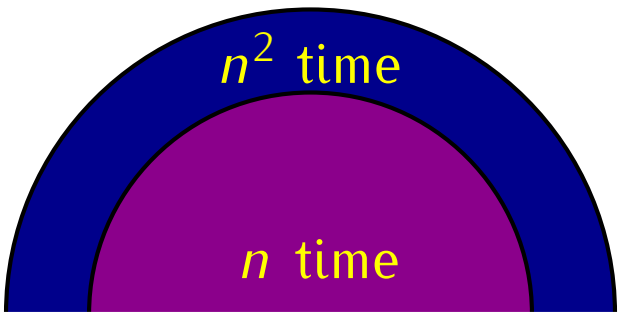
Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**



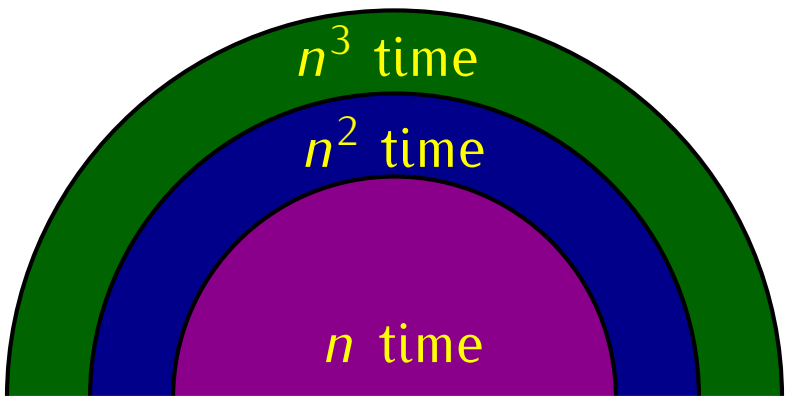
Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**



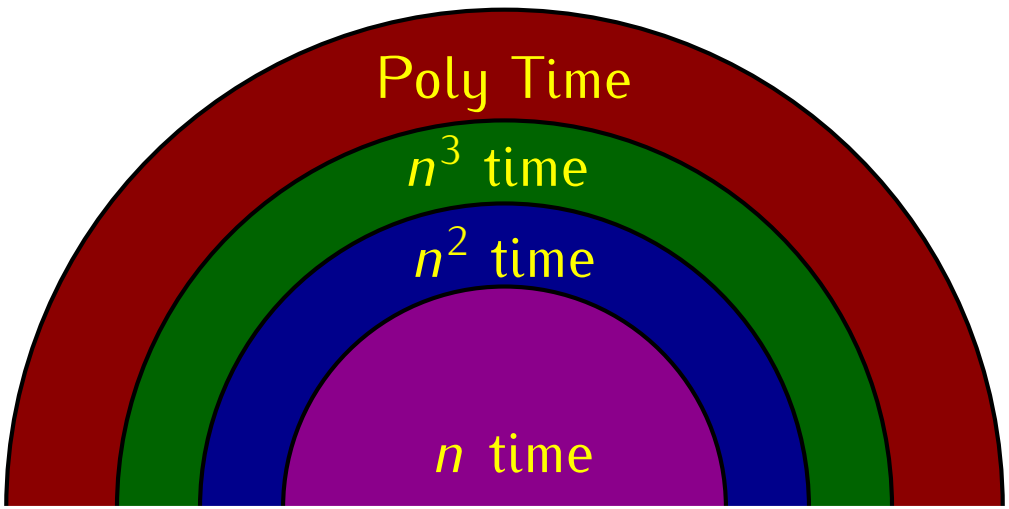
Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**



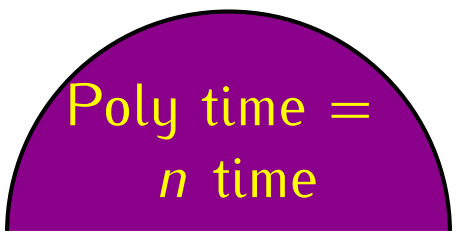
Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**



Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**

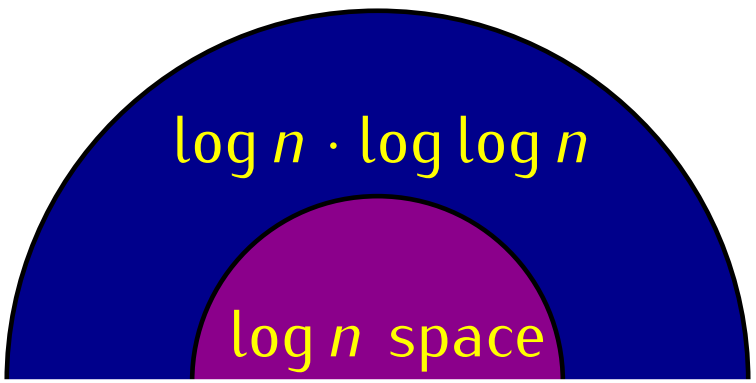


Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**

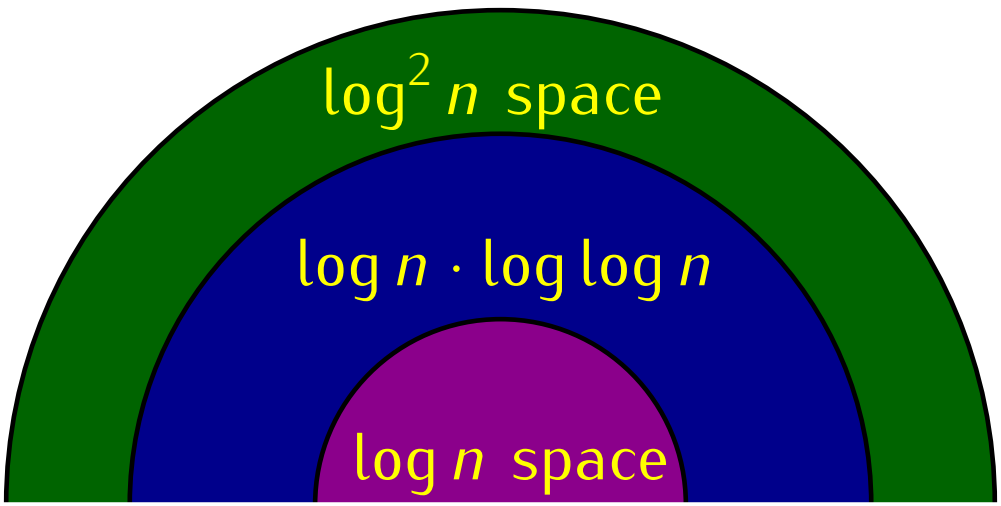
Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**



Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**



Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**

Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**

- My work: hierarchy theorems for randomized algorithms

Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**

- My work: hierarchy theorems for randomized algorithms
- If PRGs prove BPP=P, BPL=L

Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**

- My work: hierarchy theorems for randomized algorithms
- If PRGs prove BPP=P, BPL=L then hierarchies also

Hierarchy Theorems

- Fix a model of computing
  (deterministic, randomized, nondeterministic)
- **Can we achieve more given more resources?**

- My work: hierarchy theorems for randomized algorithms
- If PRGs prove BPP=P, BPL=L then hierarchies also
- Hierarchies without need for derandomization?

Hierarchy Theorems for Deterministic Algorithms

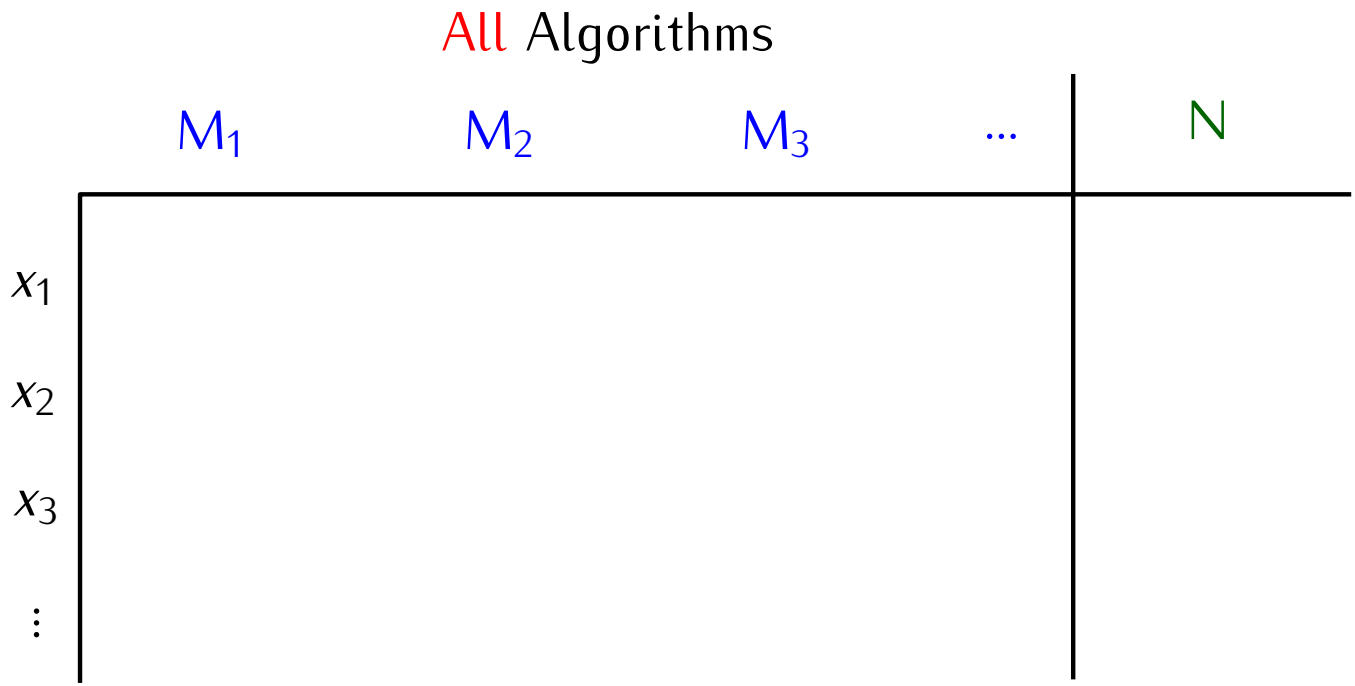
Hierarchy Theorems for Deterministic Algorithms

All Algorithms

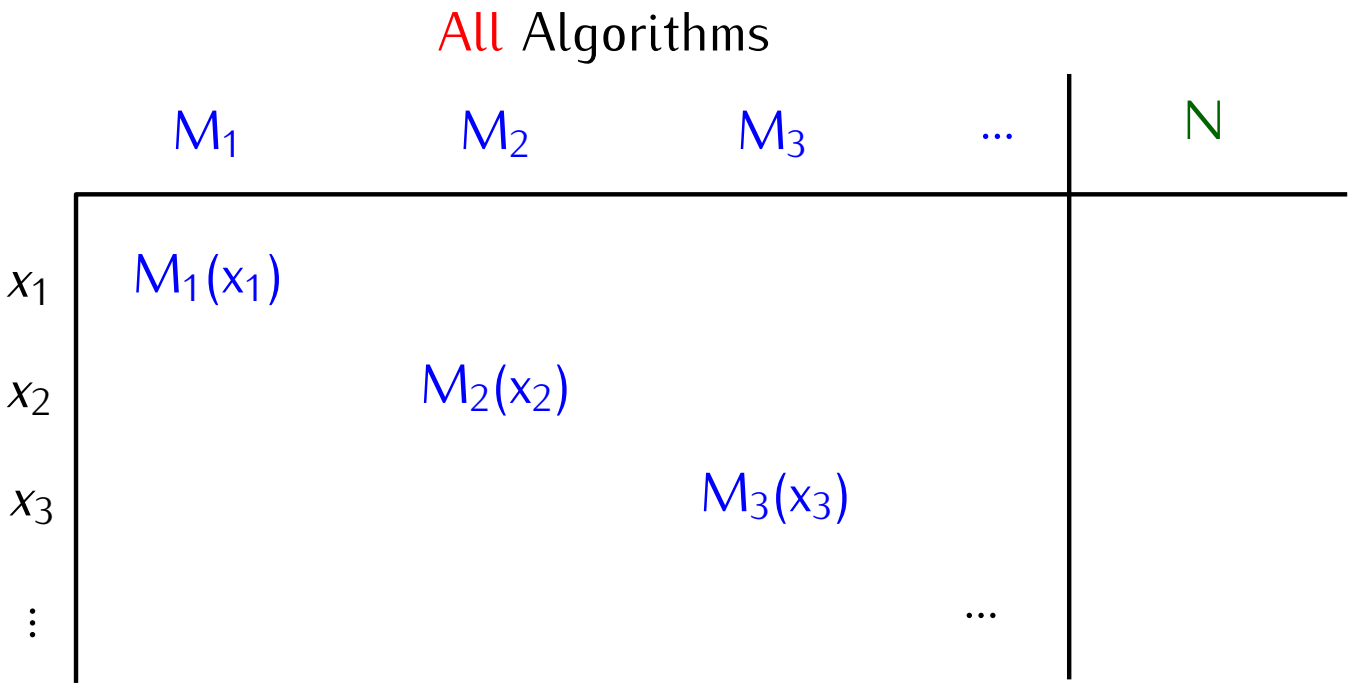
M_1 M_2 M_3 ...

N

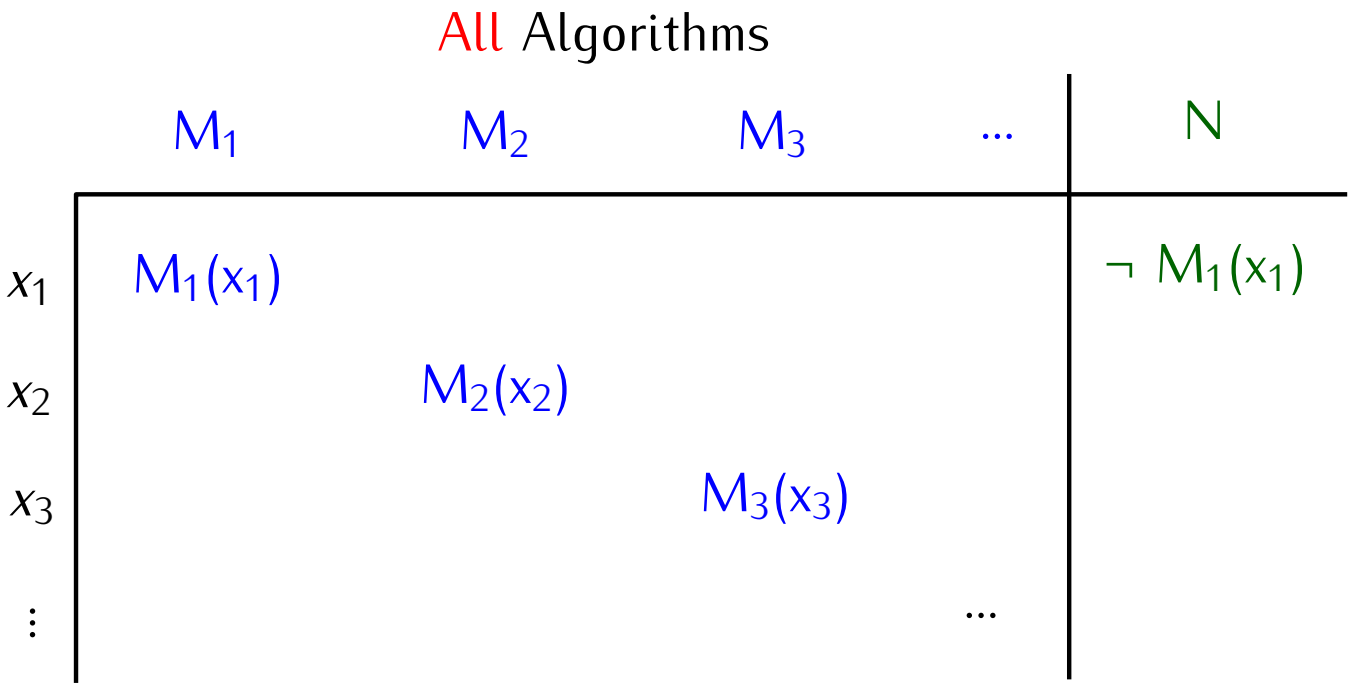
Hierarchy Theorems for Deterministic Algorithms



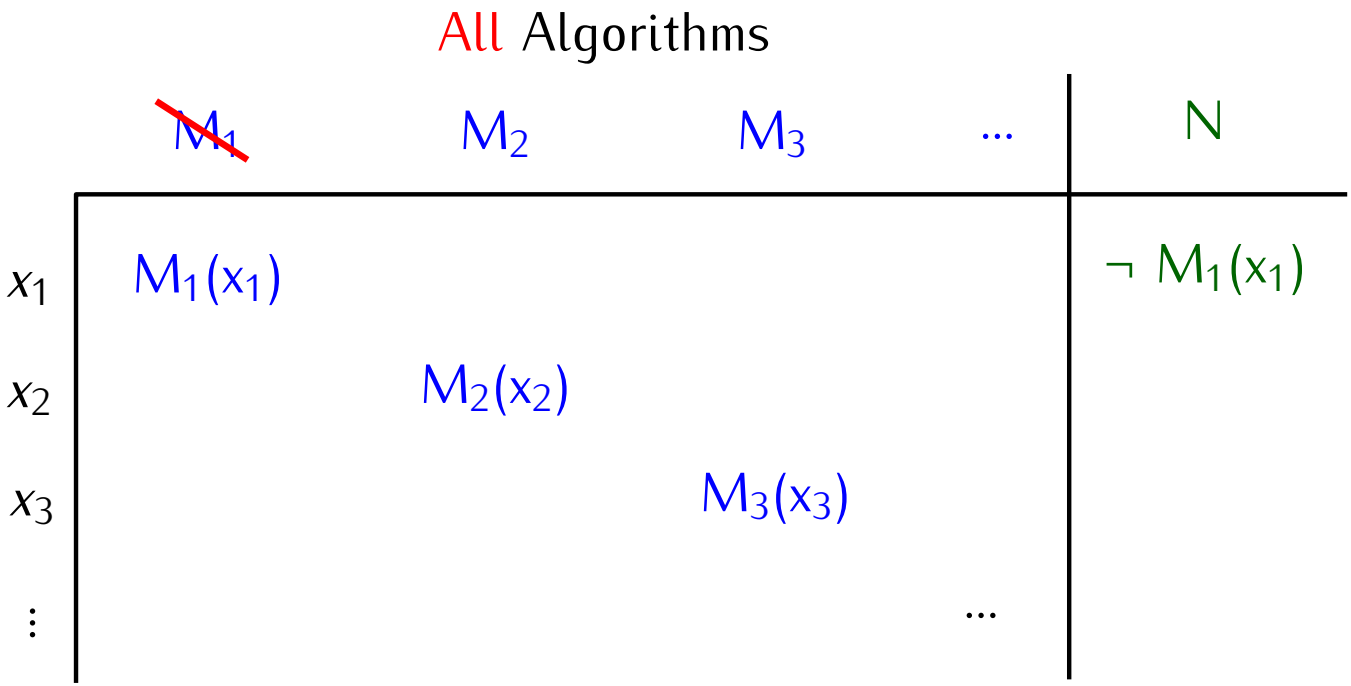
Hierarchy Theorems for Deterministic Algorithms



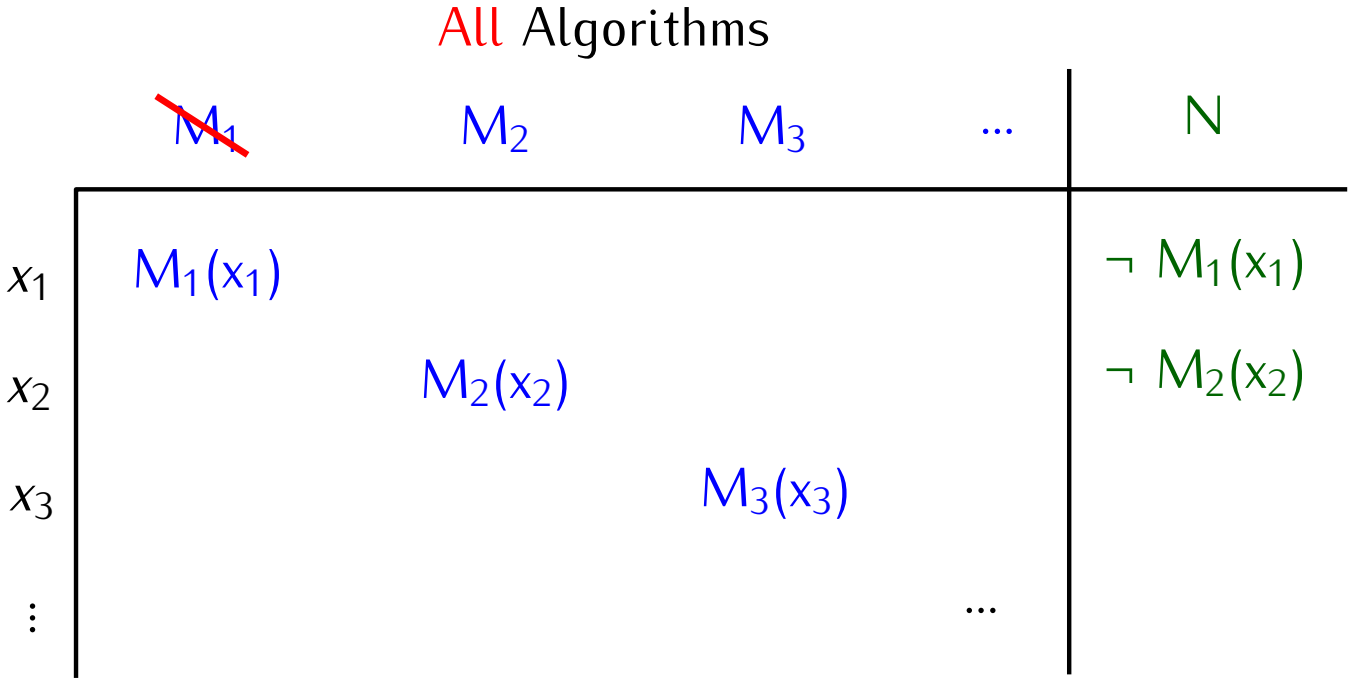
Hierarchy Theorems for Deterministic Algorithms



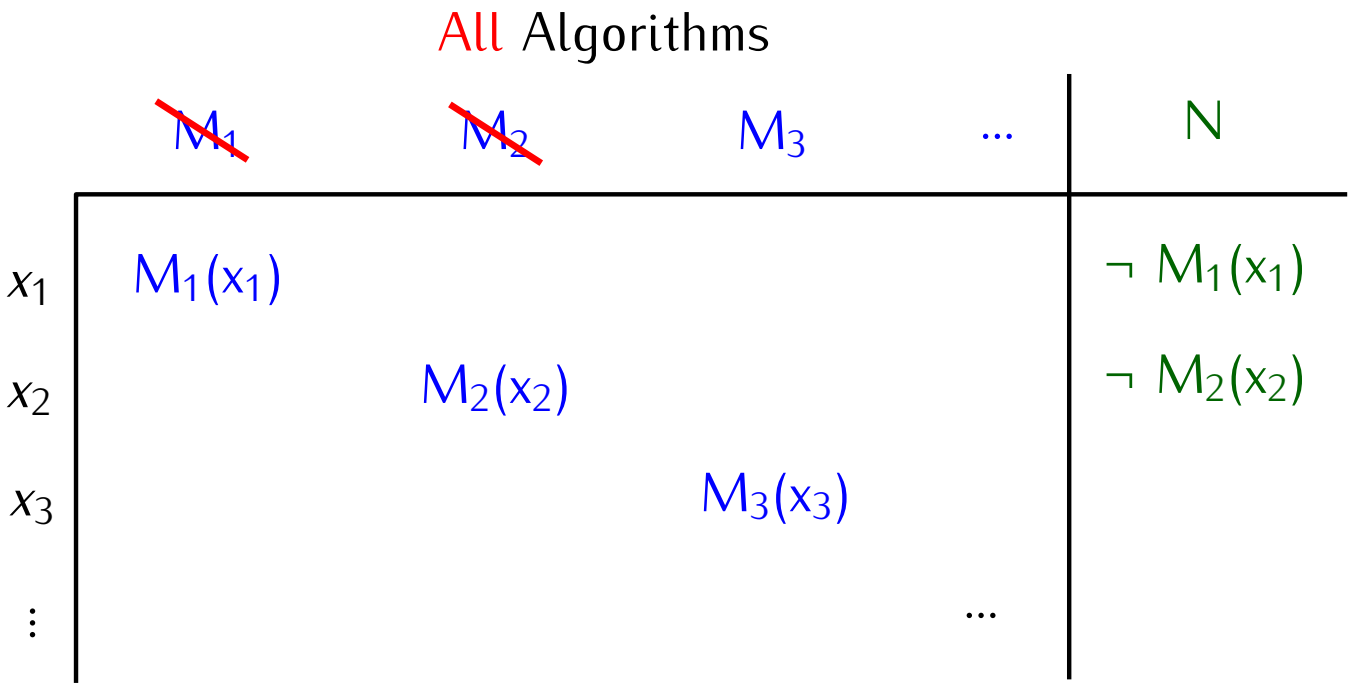
Hierarchy Theorems for Deterministic Algorithms



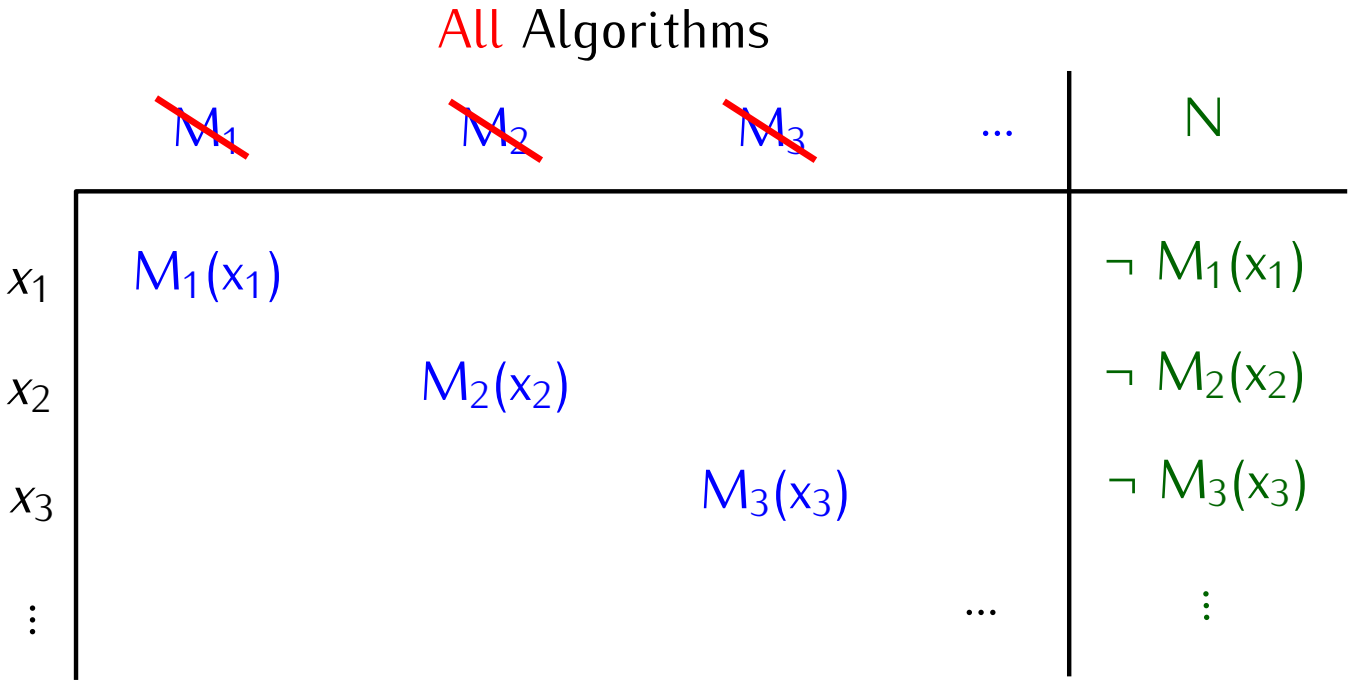
Hierarchy Theorems for Deterministic Algorithms



Hierarchy Theorems for Deterministic Algorithms



Hierarchy Theorems for Deterministic Algorithms

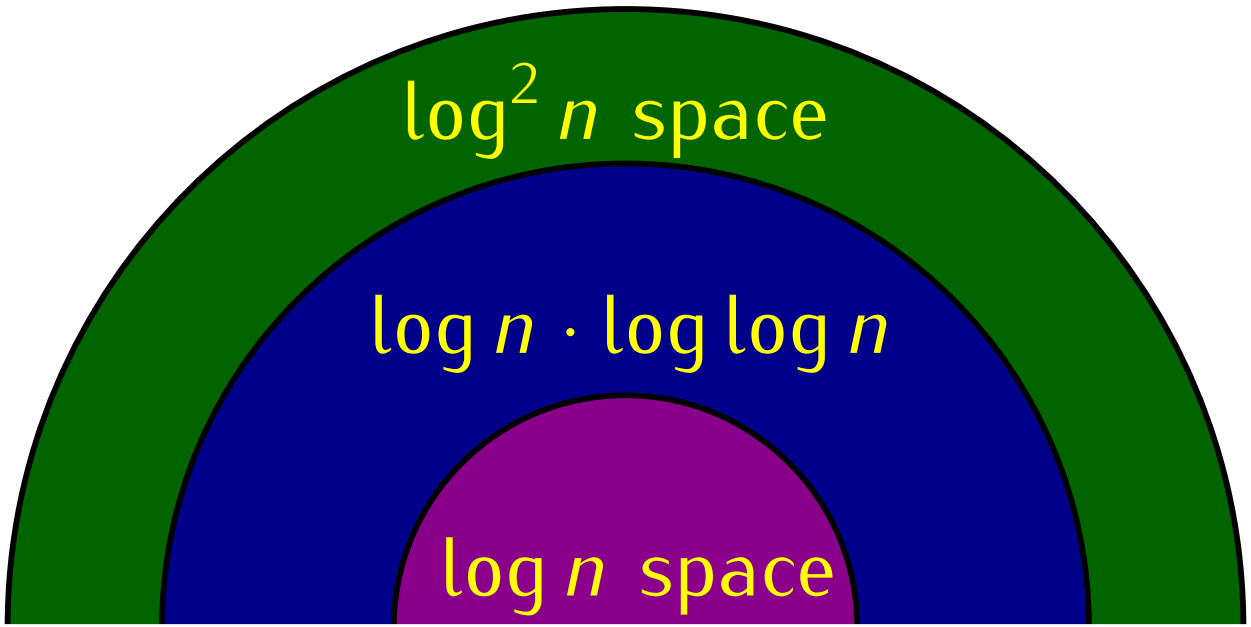


Hierarchy Theorems for Deterministic Algorithms

For any s, s'$=\omega$(s), SPACE(s') $\not\subseteq$ SPACE(s)

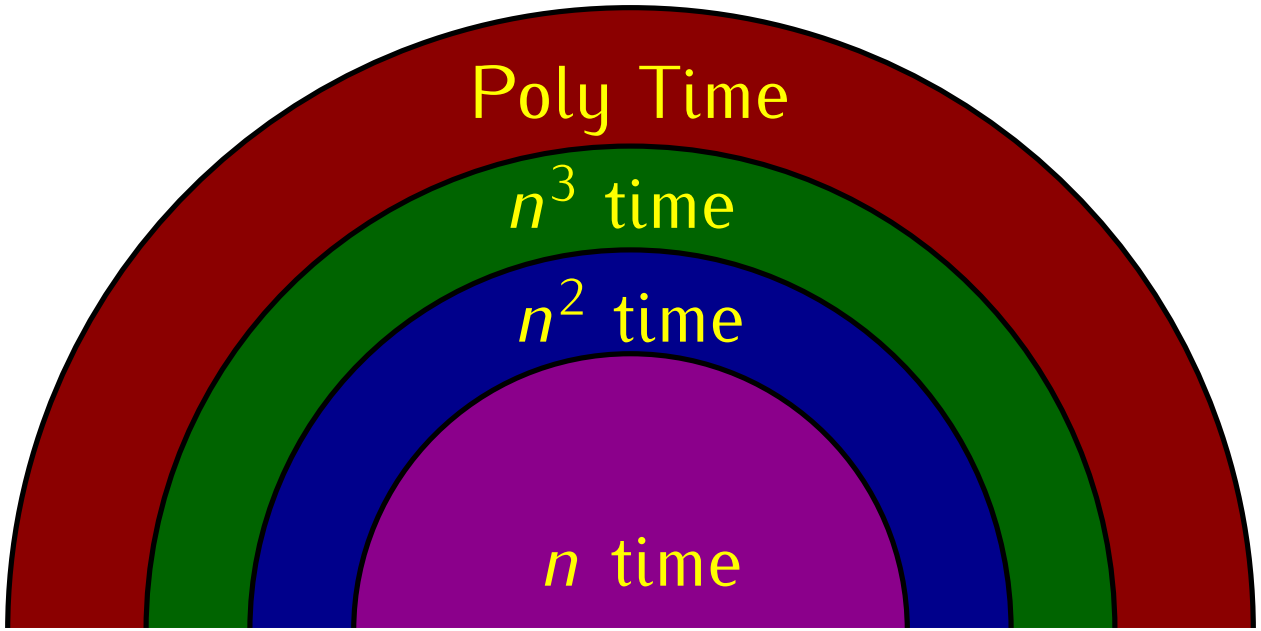
Hierarchy Theorems for Deterministic Algorithms

For any s, s'=ω(s), SPACE(s') $\not\subseteq$ SPACE(s)



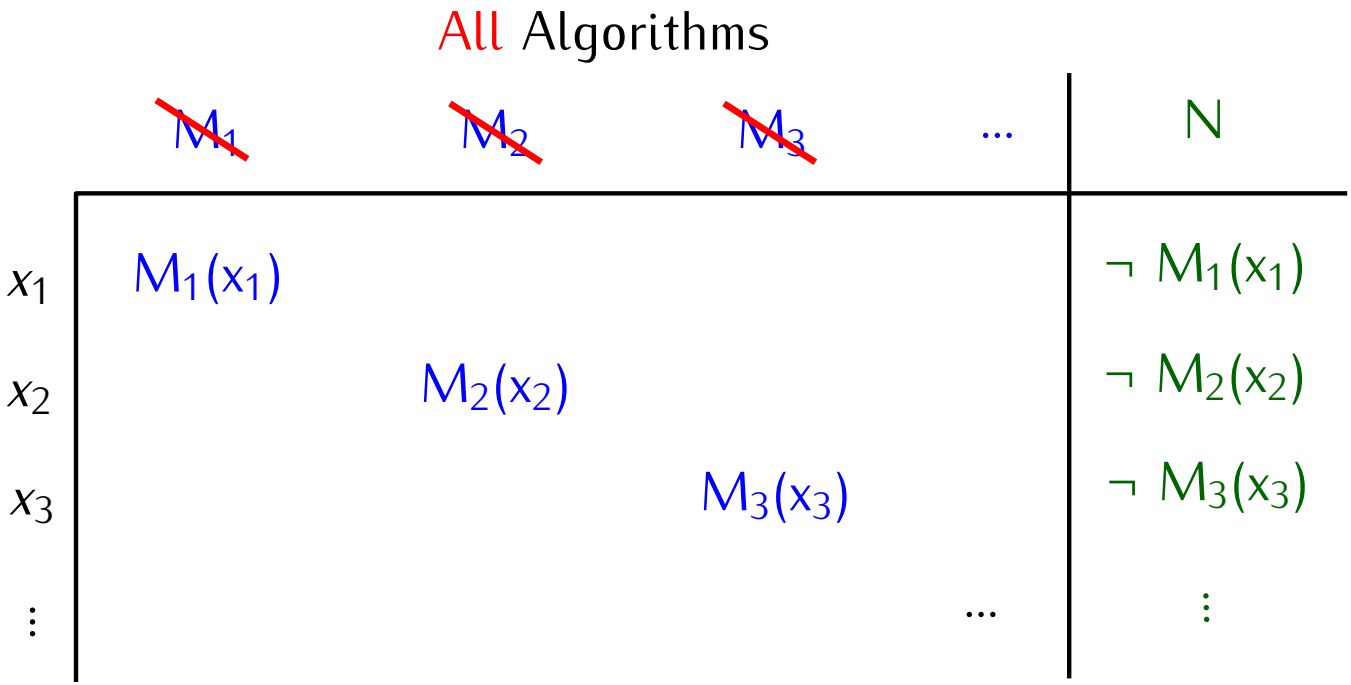
Hierarchy Theorems for Deterministic Algorithms

For any s, s'=ω(s), SPACE(s') $\not\subseteq$ SPACE(s)

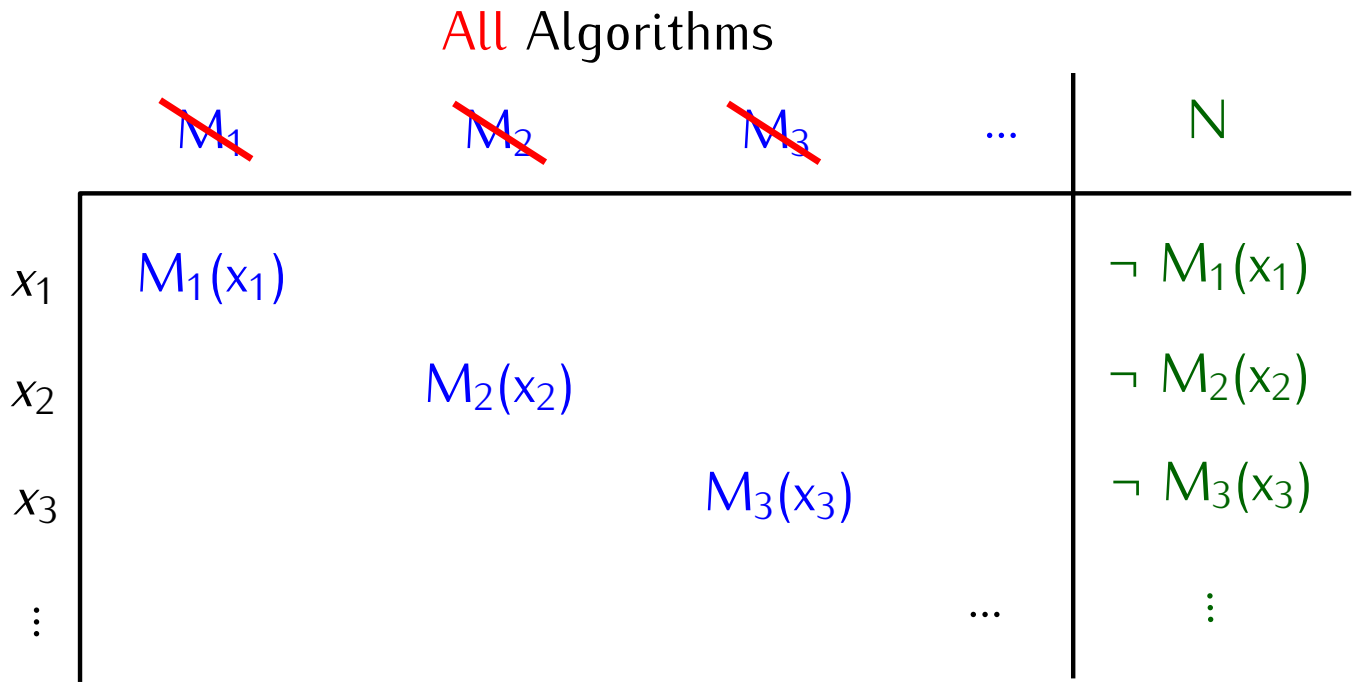


Hierarchy Theorems for **Bounded-Error** Rand Algs?

Hierarchy Theorems for **Bounded-Error** Rand Algs?

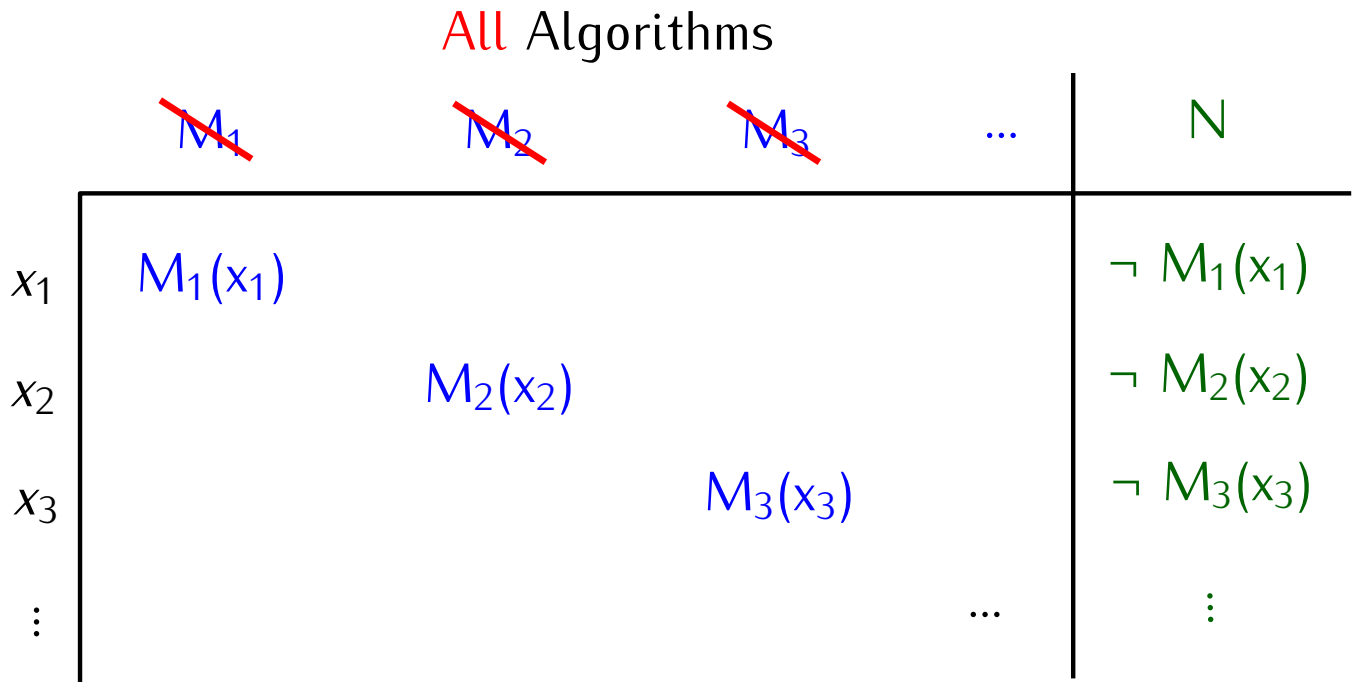


Hierarchy Theorems for **Bounded-Error** Rand Algs?



What if $\Pr[M_1(x_1) = \text{“yes”}] \approx .5$?

Hierarchy Theorems for **Bounded-Error** Rand Algs?



What if $\Pr[M_1(x_1) = \text{“yes”}] \approx .5$?$\Rightarrow$ N not *bounded error*

Best-known Hierarchies Theorems for Randomized Algs

Best-known Hierarchies Theorems for Randomized Algs

"Safe" complementation

Best-known Hierarchies Theorems for Randomized Algs

"Safe" complementation

- N bounded error, regardless behavior of M

Best-known Hierarchies Theorems for Randomized Algs

"Safe" complementation

- N bounded error, regardless behavior of M
- If M bounded error, $N(x) \neq M(x)$

Best-known Hierarchies Theorems for Randomized Algs

"Safe" complementation

- N bounded error, regardless behavior of M
- If M bounded error, $N(x) \neq M(x)$

- [Savitch, ...] Unbounded-error Randomized SPACE(s) $\subseteq$ Deterministic SPACE(s^2)

Best-known Hierarchies Theorems for Randomized Algs

"Safe" complementation

- N bounded error, regardless behavior of M
- If M bounded error, $N(x) \neq M(x)$

- [Savitch, ...] Unbounded-error Randomized SPACE(s) $\subseteq$ Deterministic SPACE(s^2)
- $\Rightarrow$ Randomized SPACE(s') $\not\subseteq$ Randomized SPACE(s), for $s' = \omega(s^2)$

Best-known Hierarchies Theorems for Randomized Algs

"Safe" complementation

- N bounded error, regardless behavior of M
- If M bounded error, $N(x) \neq M(x)$

- [Savitch, ...] Unbounded-error Randomized SPACE(s) $\subseteq$ Deterministic SPACE(s^2)
- $\Rightarrow$ Randomized SPACE(s') $\not\subseteq$ Randomized SPACE(s), for s'$=\omega(s^2)$
- Can show: Randomized SPACE(s') $\not\subseteq$ Randomized SPACE(s), for s'$=\omega(s^{1+\delta})$, any $\delta > 0$

Best-known Hierarchies Theorems for Randomized Algs

"Safe" complementation

- N bounded error, regardless behavior of M
- If M bounded error, $N(x) \neq M(x)$

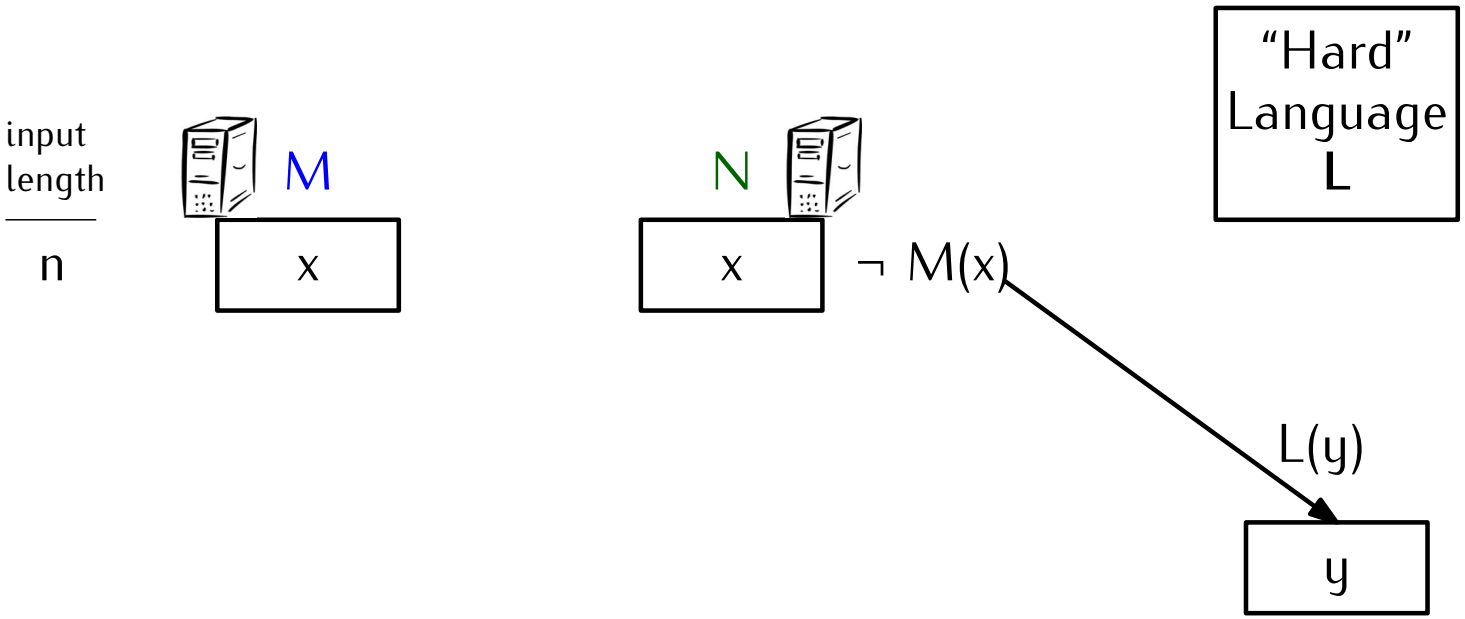
- [Savitch, ...] Unbounded-error Randomized SPACE(s) $\subseteq$ Deterministic SPACE(s^2)
- $\Rightarrow$ Randomized SPACE(s') $\not\subseteq$ Randomized SPACE(s), for s'$=\omega(s^2)$
- Can show: Randomized SPACE(s') $\not\subseteq$ Randomized SPACE(s), for s'$=\omega(s^{1+\delta})$, any $\delta > 0$
- **Best possible: SPACE(s') $\not\subseteq$ SPACE(s) for any s'$=\omega$(s)**

Tight Hierarchies with One Bit of Advice, Log Space

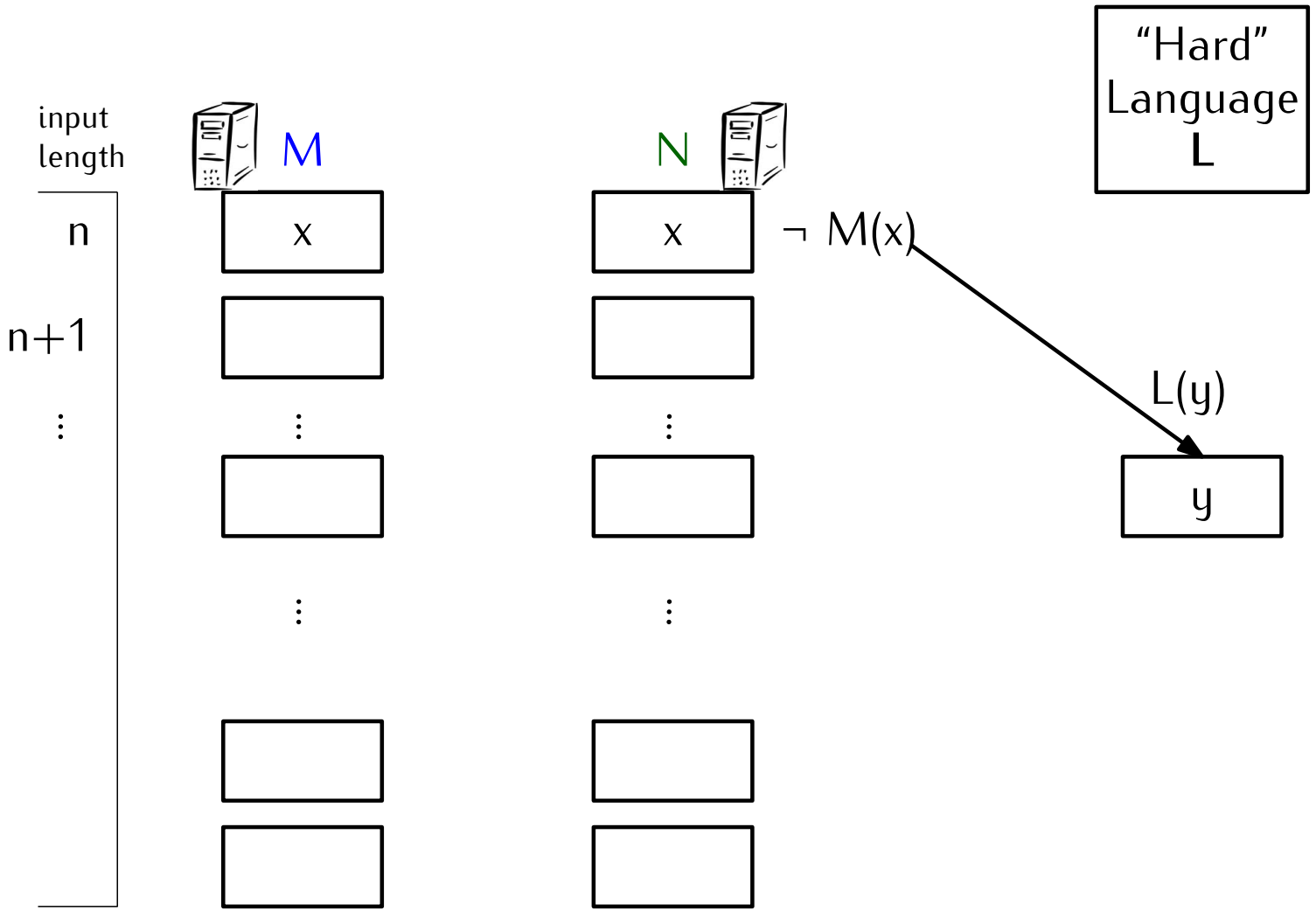
Tight Hierarchies with One Bit of Advice, Log Space

Tight Hierarchies with One Bit of Advice, Log Space

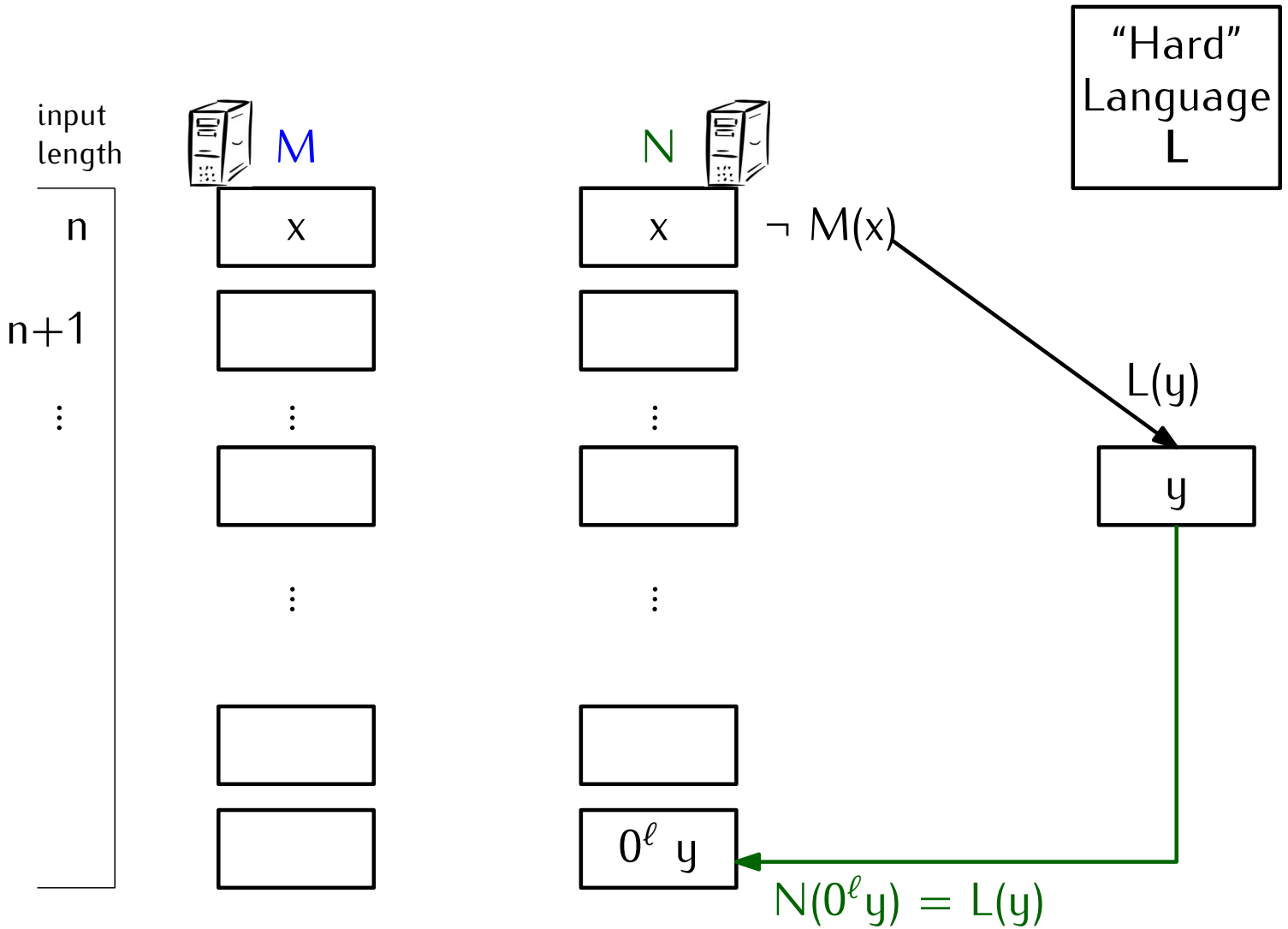
Tight Hierarchies with One Bit of Advice, Log Space



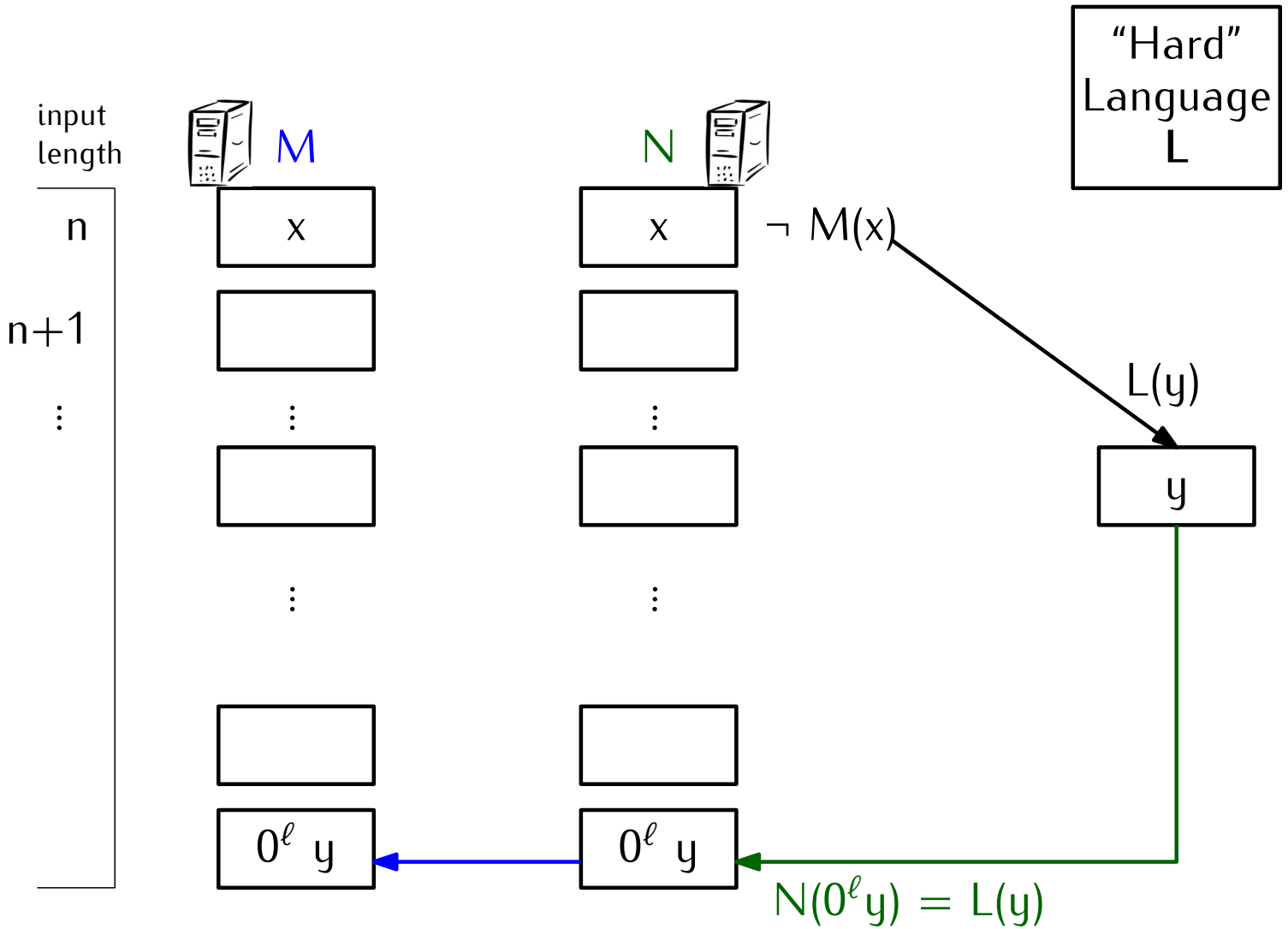
Tight Hierarchies with One Bit of Advice, Log Space



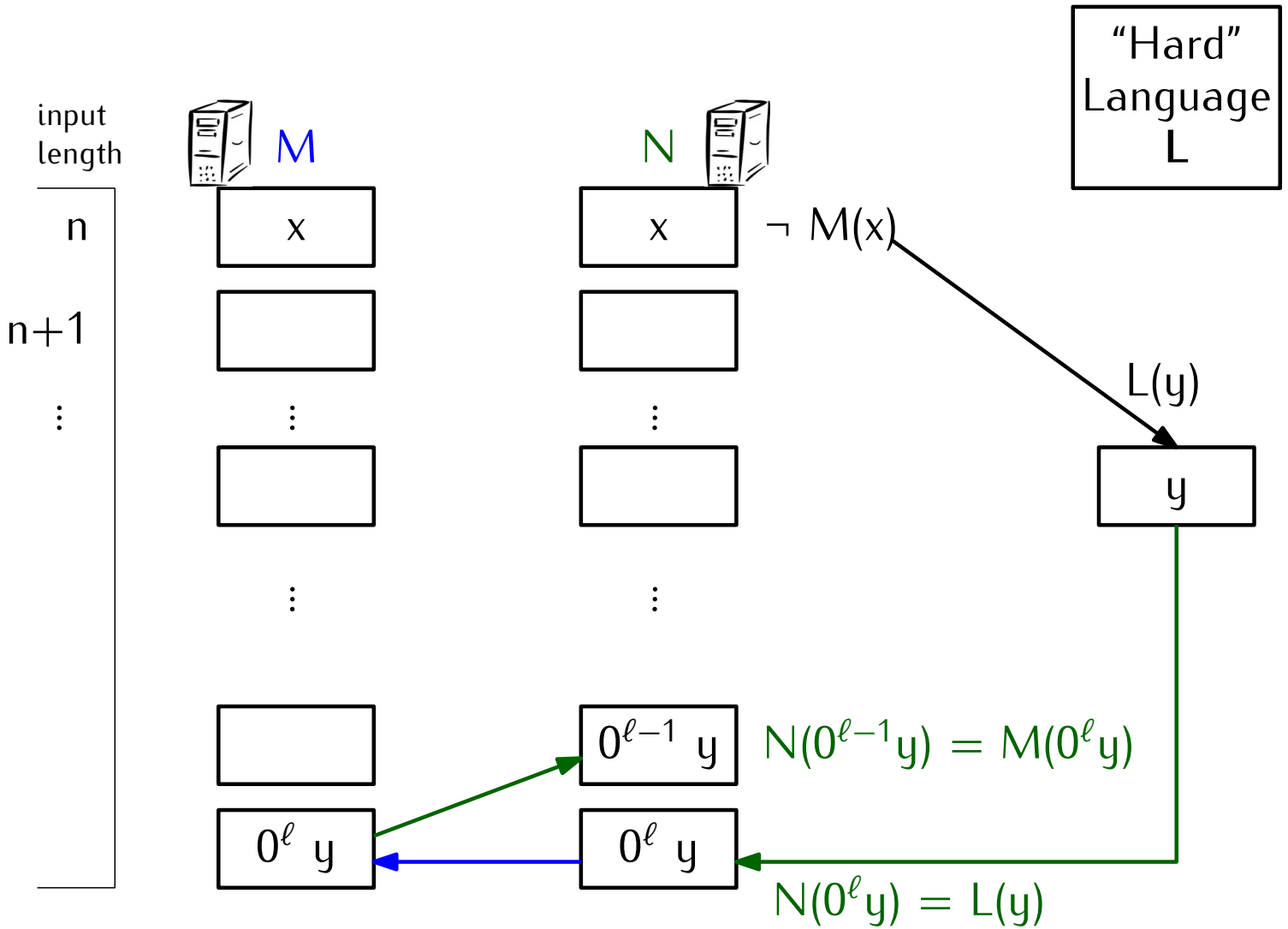
Tight Hierarchies with One Bit of Advice, Log Space



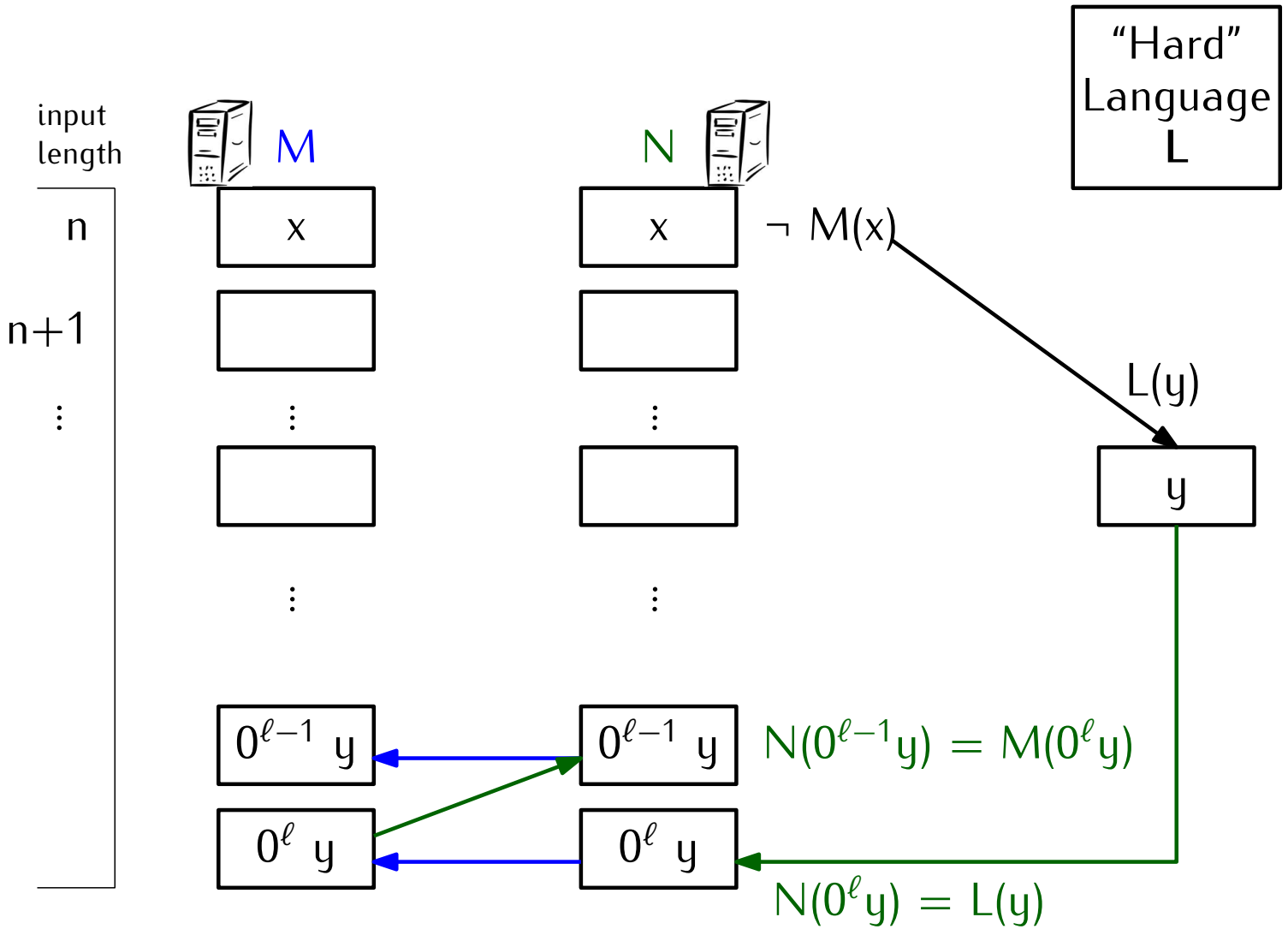
Tight Hierarchies with One Bit of Advice, Log Space



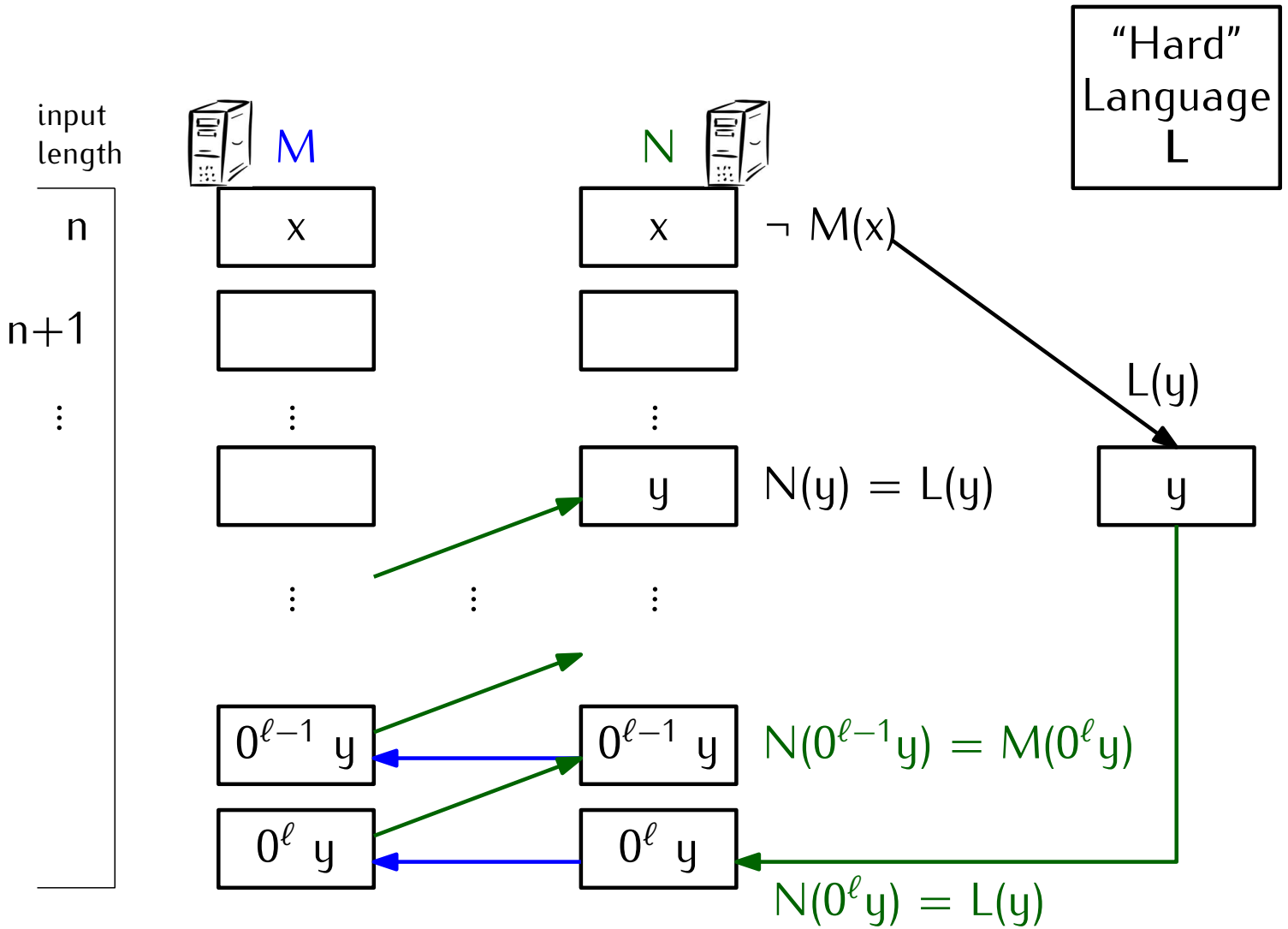
Tight Hierarchies with One Bit of Advice, Log Space



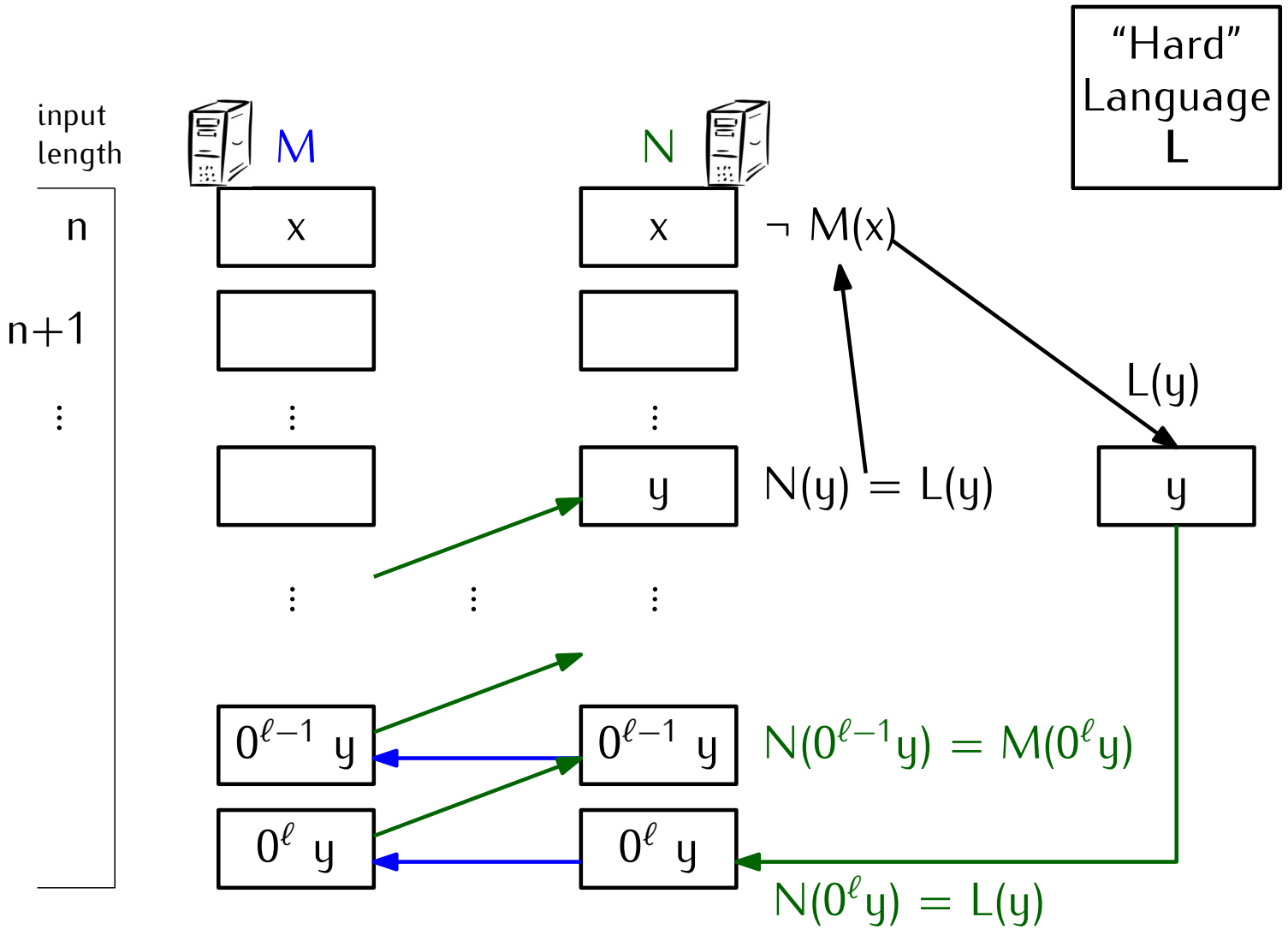
Tight Hierarchies with One Bit of Advice, Log Space



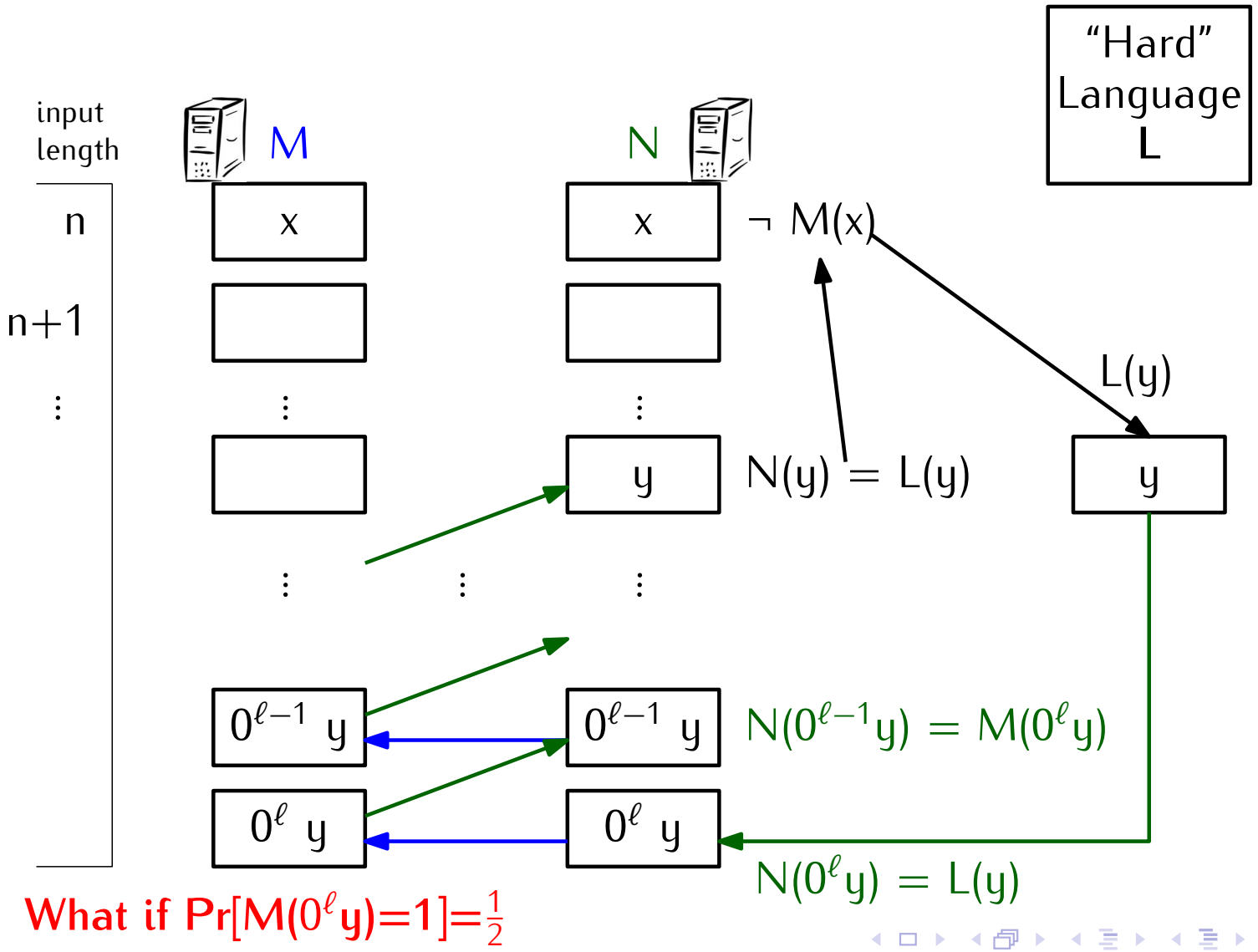
Tight Hierarchies with One Bit of Advice, Log Space



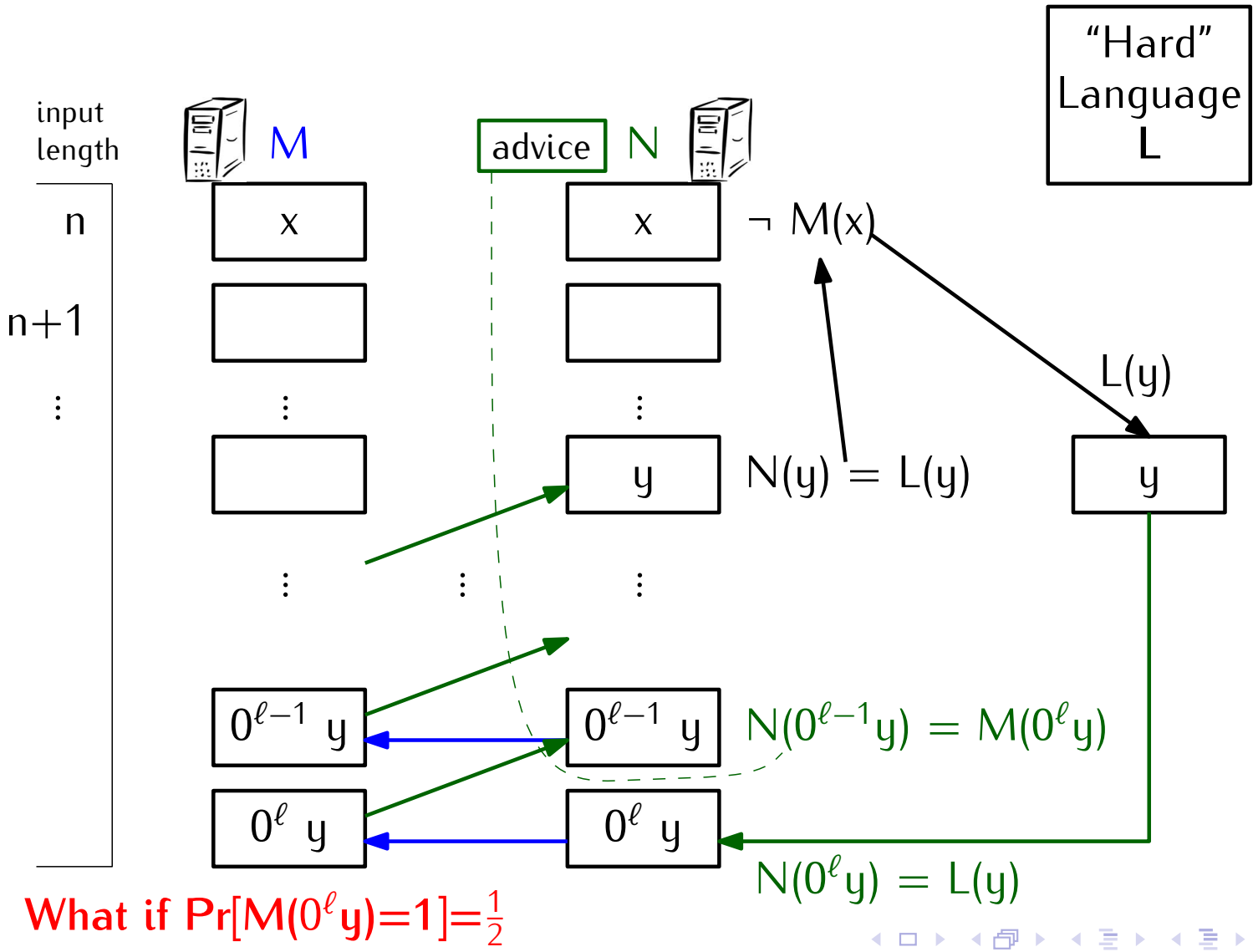
Tight Hierarchies with One Bit of Advice, Log Space



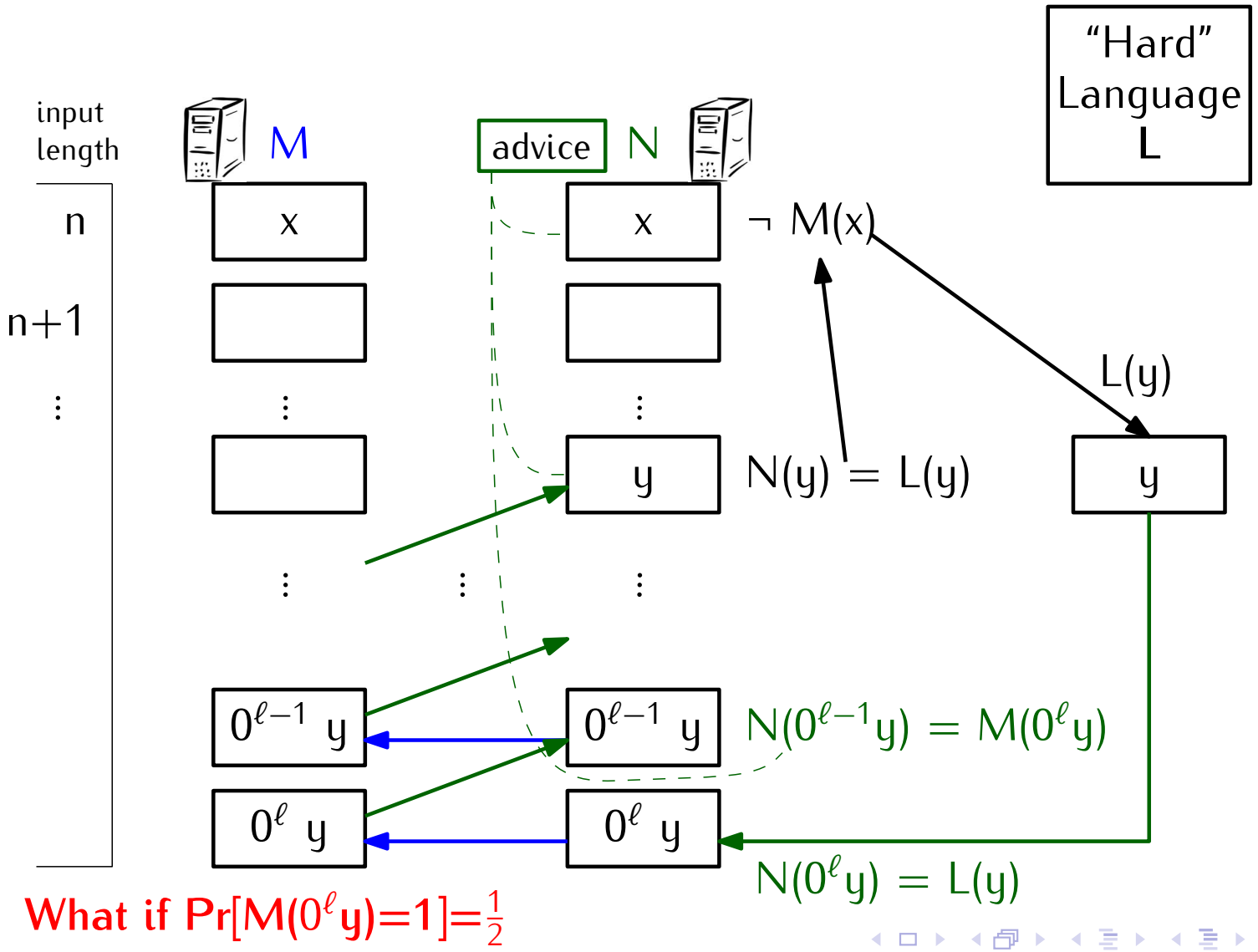
Tight Hierarchies with One Bit of Advice, Log Space



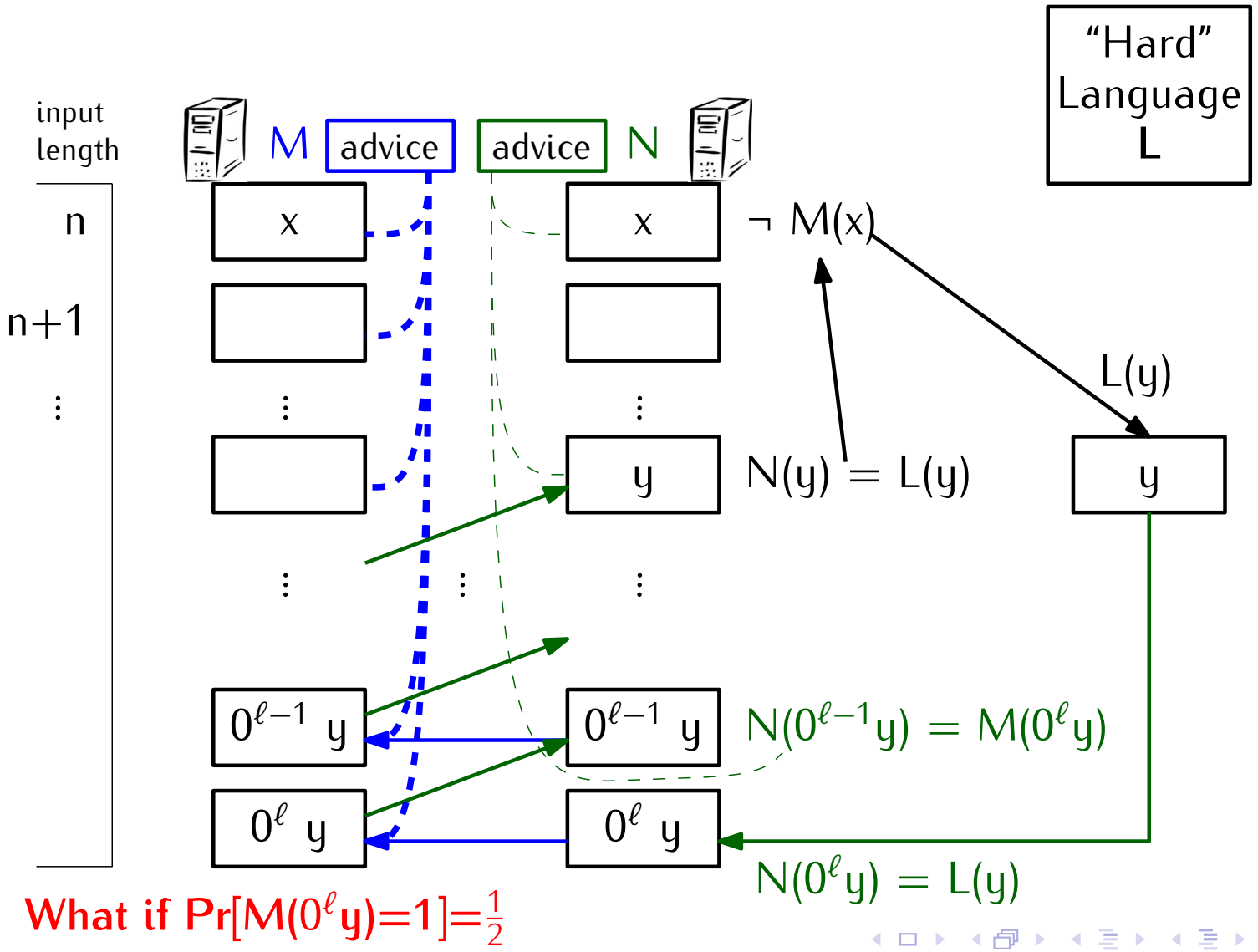
Tight Hierarchies with One Bit of Advice, Log Space



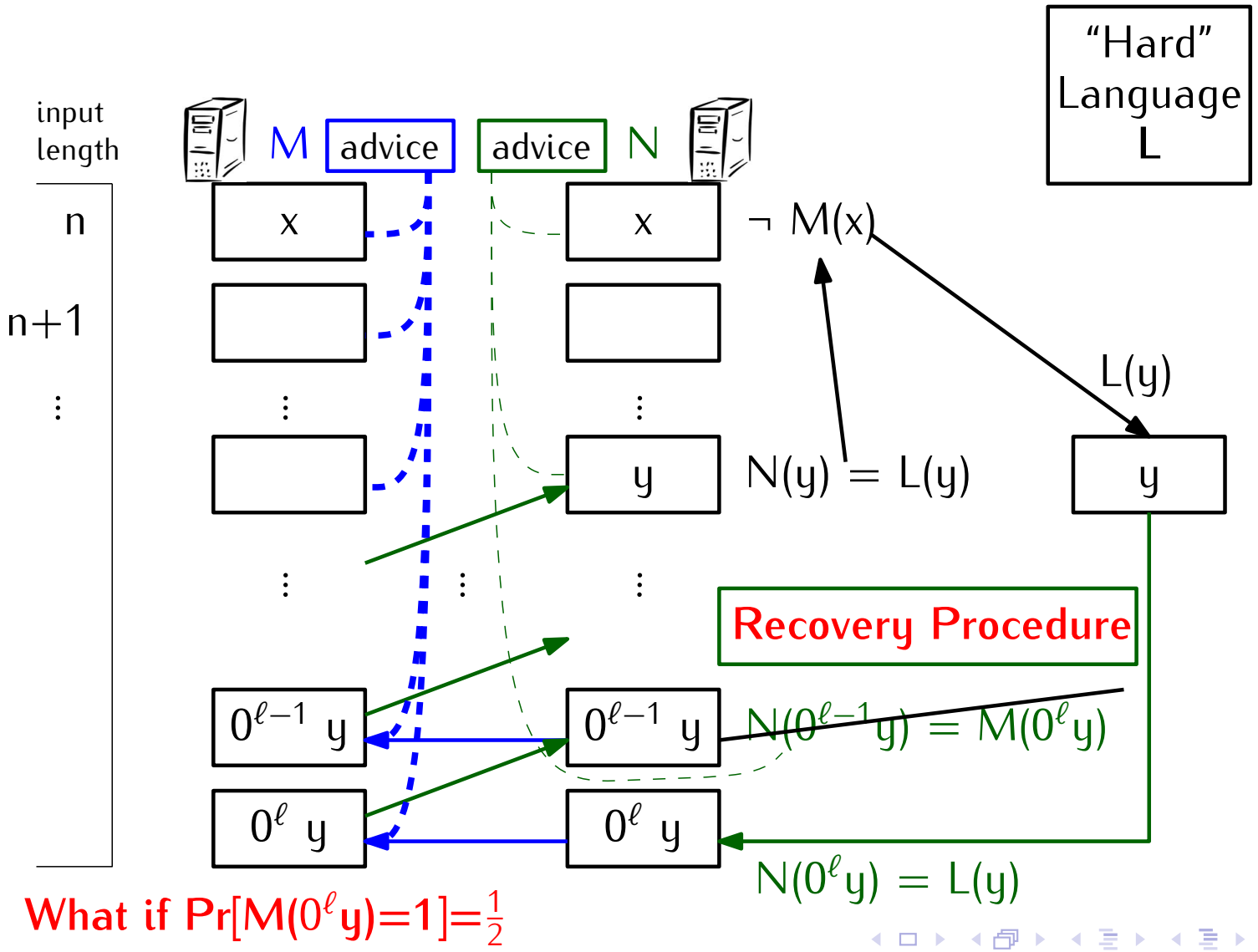
Tight Hierarchies with One Bit of Advice, Log Space



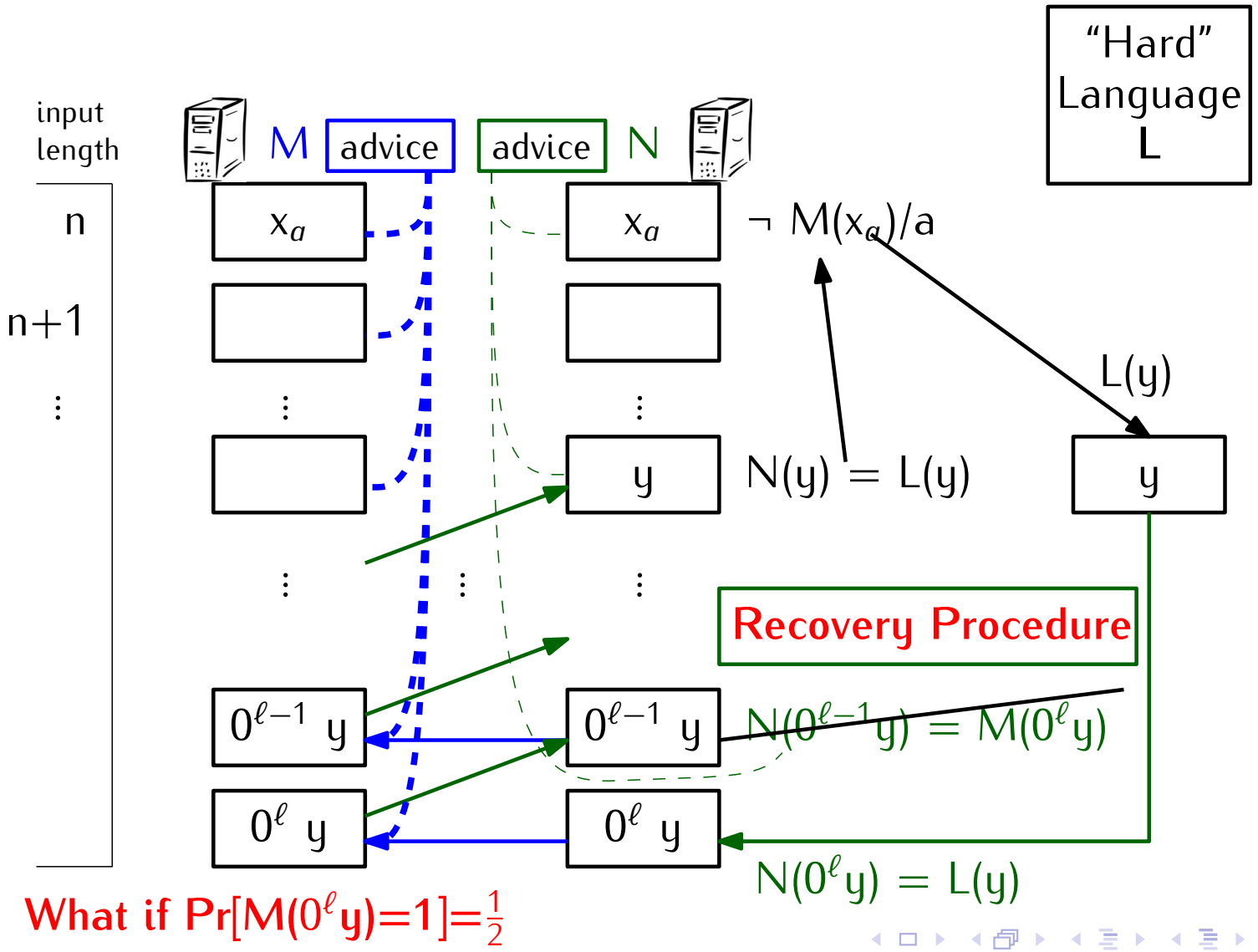
Tight Hierarchies with One Bit of Advice, Log Space



Tight Hierarchies with One Bit of Advice, Log Space



Tight Hierarchies with One Bit of Advice, Log Space



Recovery Procedure for L

Recovery Procedure for L

- **Input:** y, list of randomized machines

Recovery Procedure for L

- **Input:** y, list of randomized machines
- **Output:** L(y), using small space, with bounded error

Recovery Procedure for L

- **Input:** y, list of randomized machines
- **Output:** L(y), using small space, with bounded error
- **Pre-condition:** at least one machine in list computes L on instances of length $|y|$, using small space, with bounded error

Hard Language L – $\langle D,x,t,j \rangle$

Hard Language L – ⟨D,x,t,j⟩

Computation Tableau of machine D on input x

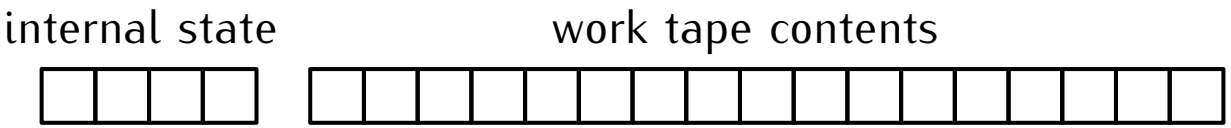
Hard Language L – ⟨D,x,t,j⟩

Computation Tableau of machine D on input x

internal state work tape contents

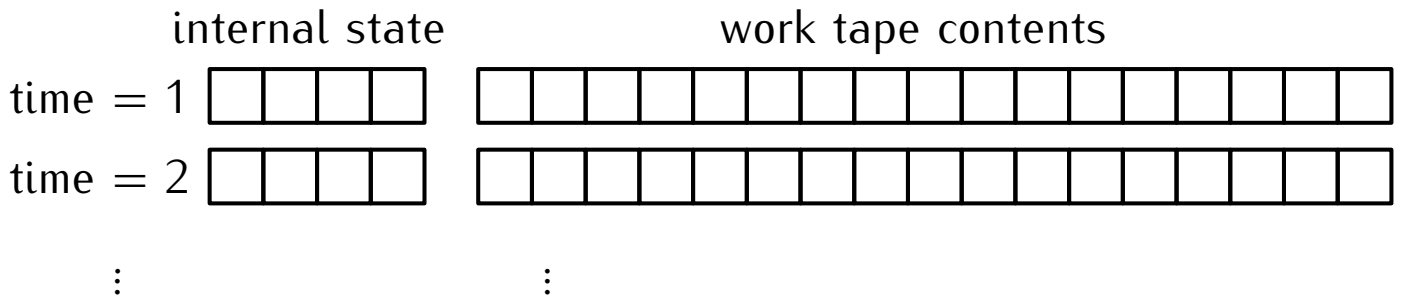
Hard Language L – ⟨D,x,t,j⟩

Computation Tableau of machine D on input x

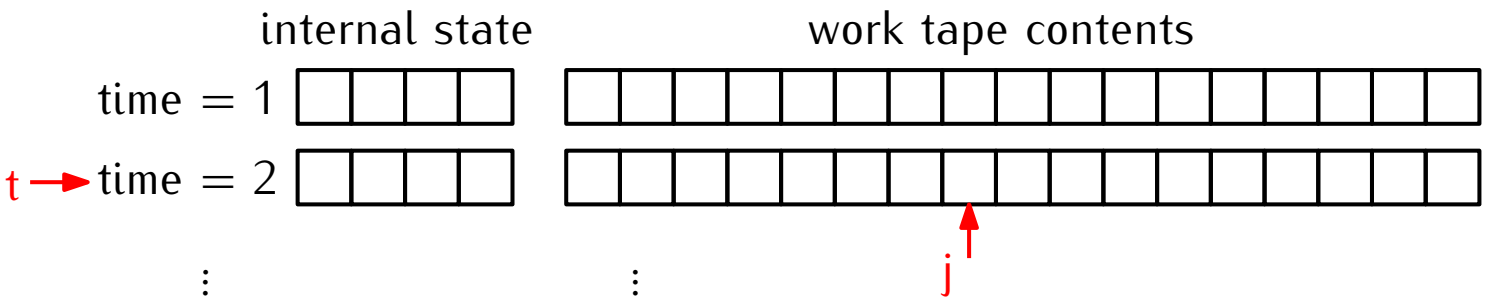


Hard Language L – ⟨D,x,t,j⟩

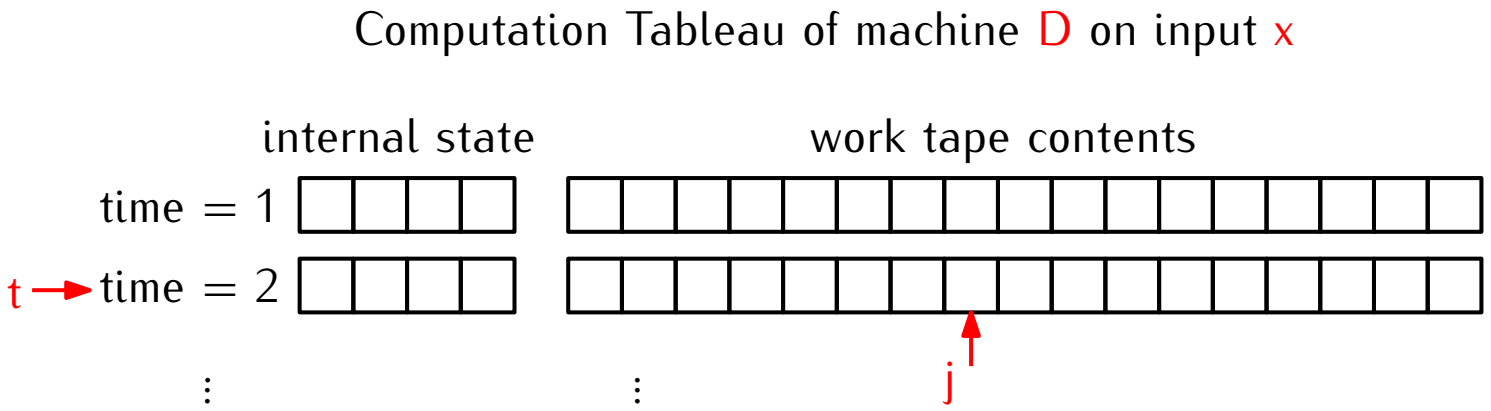
Computation Tableau of machine D on input x



Hard Language L – ⟨D,x,t,j⟩

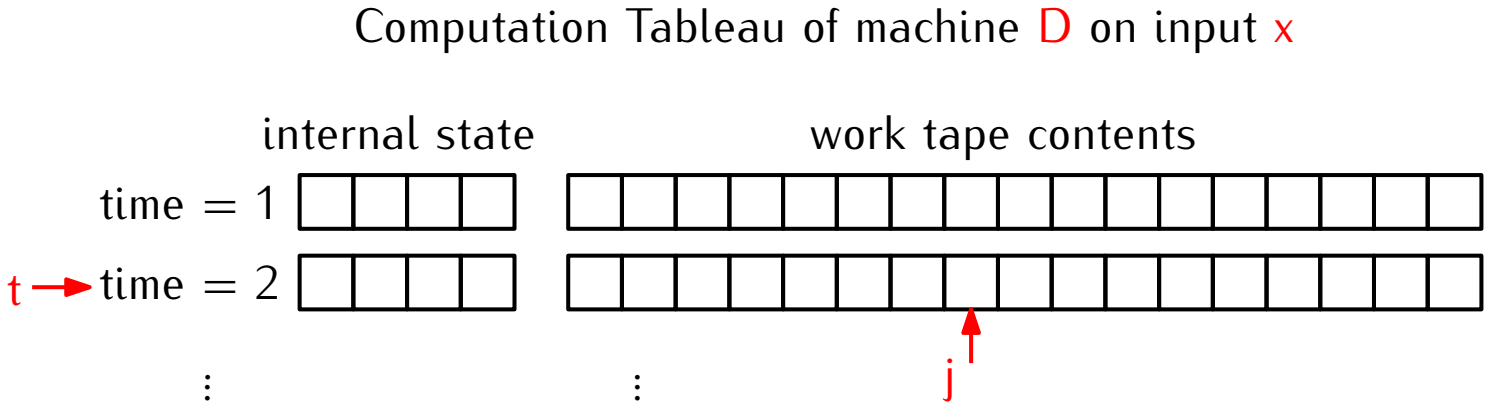


Hard Language L – ⟨D,x,t,j⟩



- L is "hard" – complete for P ⊇ BPL

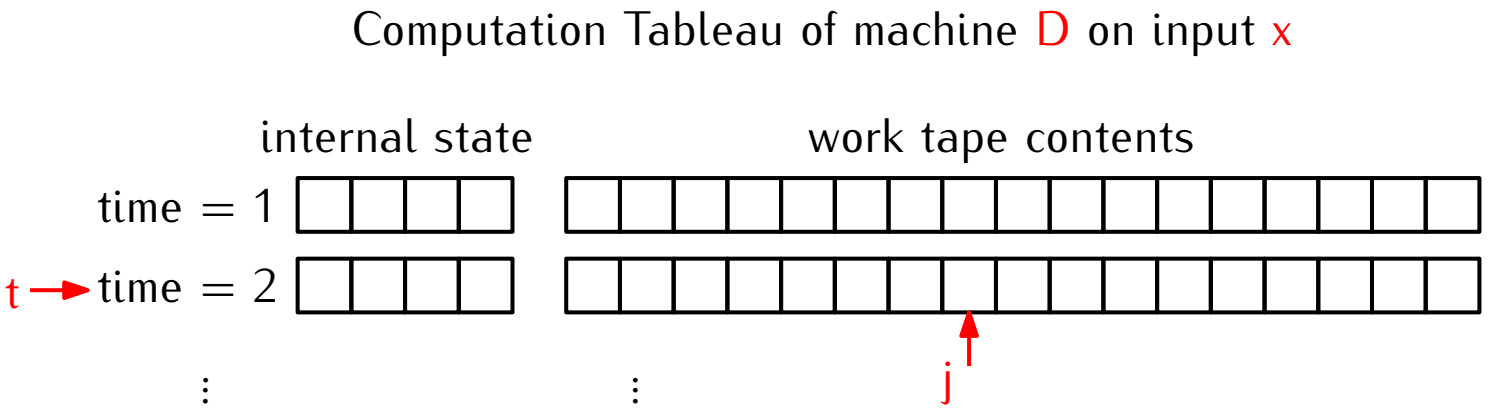
Hard Language L – ⟨D,x,t,j⟩



- L is "hard" – complete for P $\supseteq$ BPL
- Space-efficient Recovery Procedure for L

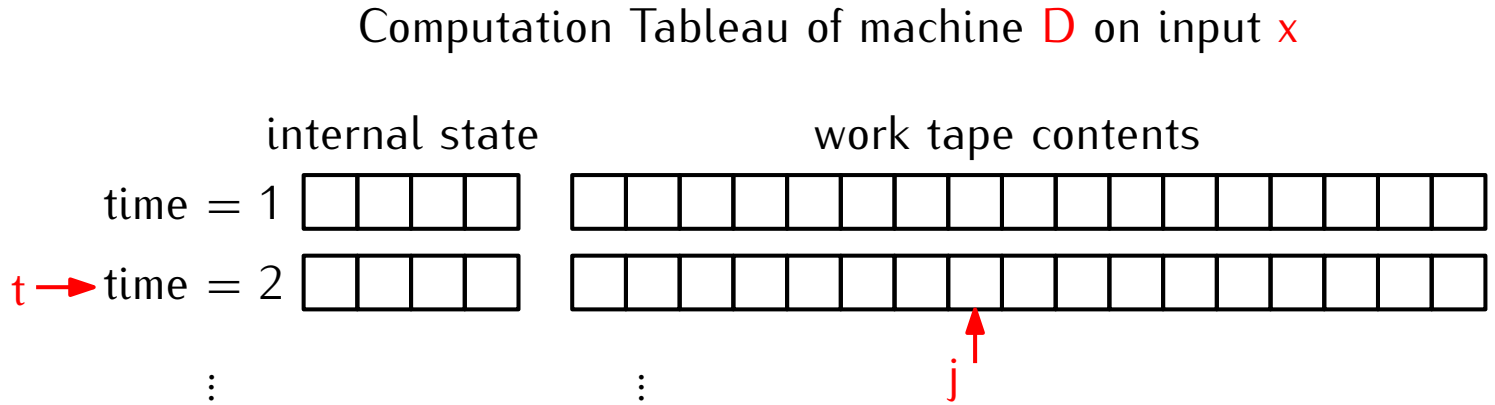
Recovery Procedure

Recovery Procedure



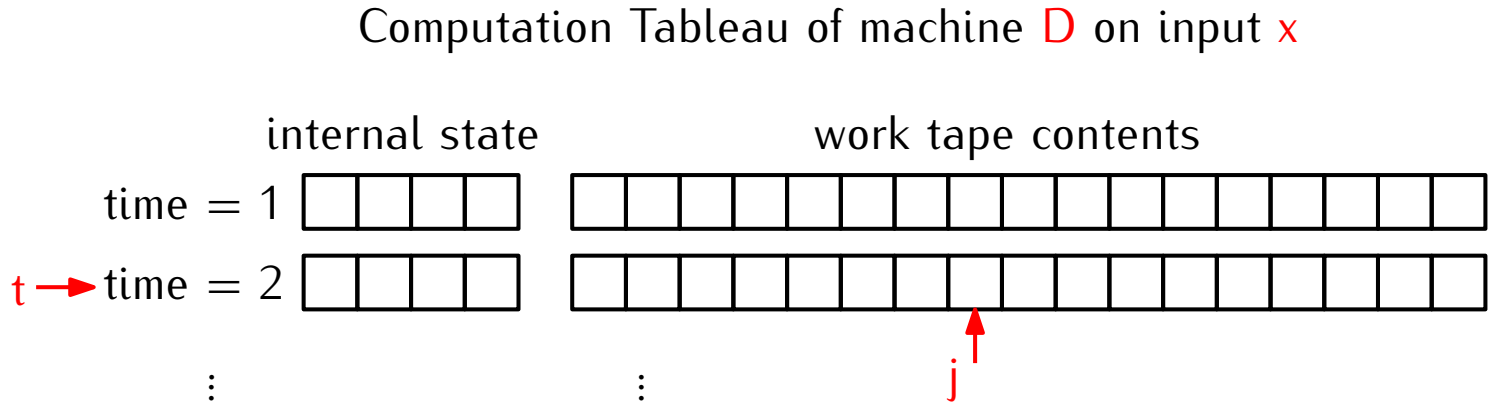
- **Input:** $\langle D,x,t,j \rangle$, $\{M_1, M_2, M_3, \ldots\}$

Recovery Procedure



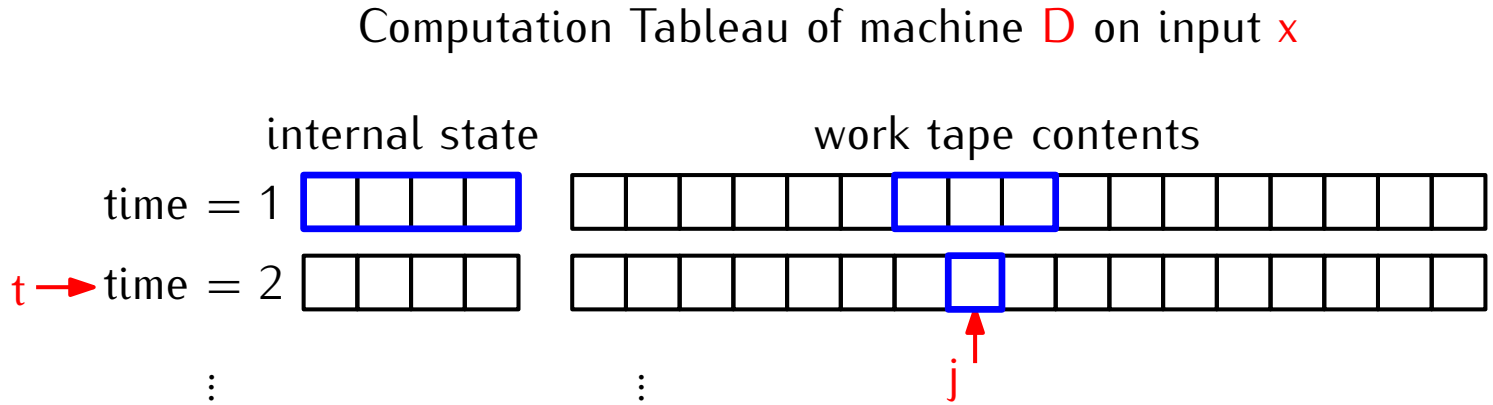
- **Input:** $\langle D,x,t,j \rangle$, $\{M_1, M_2, M_3, \ldots\}$
- For each M_i, $\langle D,x,t',j' \rangle$

Recovery Procedure



- **Input:** $\langle D,x,t,j \rangle$, $\{M_1, M_2, M_3, \ldots\}$
- For each M_i, $\langle D,x,t',j' \rangle$
  - Bounded-error test

Recovery Procedure



- **Input:** $\langle D,x,t,j \rangle$, $\{M_1, M_2, M_3, \ldots\}$
- For each M_i, $\langle D,x,t',j' \rangle$
  - Bounded-error test
  - Local consistency test

Hierarchy with One Bit of Advice

Hierarchy with One Bit of Advice

For bounded-error randomized machines,
SPACE(s')/1 $\not\subseteq$ SPACE(log n)/log n, for any s'=ω(log n)

Hierarchy with One Bit of Advice

For bounded-error randomized machines,
SPACE(s')/1 $\not\subseteq$ SPACE(s)/s, for any $s' = \omega(s)$,
typical s from log n to n

Hierarchy with One Bit of Advice

For bounded-error randomized machines,
SPACE(s')/1 $\not\subseteq$ SPACE(s)/s, for any $s' = \omega(s)$,
typical s from log n to n

- One- and Zero-sided error

Hierarchy with One Bit of Advice

For bounded-error randomized machines,
SPACE(s')/1 $\not\subseteq$ SPACE(s)/s, for any $s' = \omega(s)$,
typical s from log n to n

- One- and Zero-sided error
  - Same high level approach

Hierarchy with One Bit of Advice

For bounded-error randomized machines,
SPACE(s')/1 $\not\subseteq$ SPACE(s)/s, for any $s' = \omega(s)$,
typical s from log n to n

- One- and Zero-sided error
  - Same high level approach
  - Hard Language "L": NL-complete language similar to st-connectivity

Hierarchy with One Bit of Advice

For bounded-error randomized machines,
SPACE(s')/1 $\not\subseteq$ SPACE(s)/s, for any $s' = \omega(s)$,
typical s from log n to n

- One- and Zero-sided error
  - Same high level approach
  - Hard Language "L": NL-complete language similar to st-connectivity
  - Zero-error recovery procedure for L based on inductive counting [I88, S88]

Outline

How Powerful is Randomness?

- "Typically-correct" Derandomization
- Hierarchy Theorems for Randomized Algorithms
- Derandomizing Monotone Computations

Derandomizing Monotone Computations

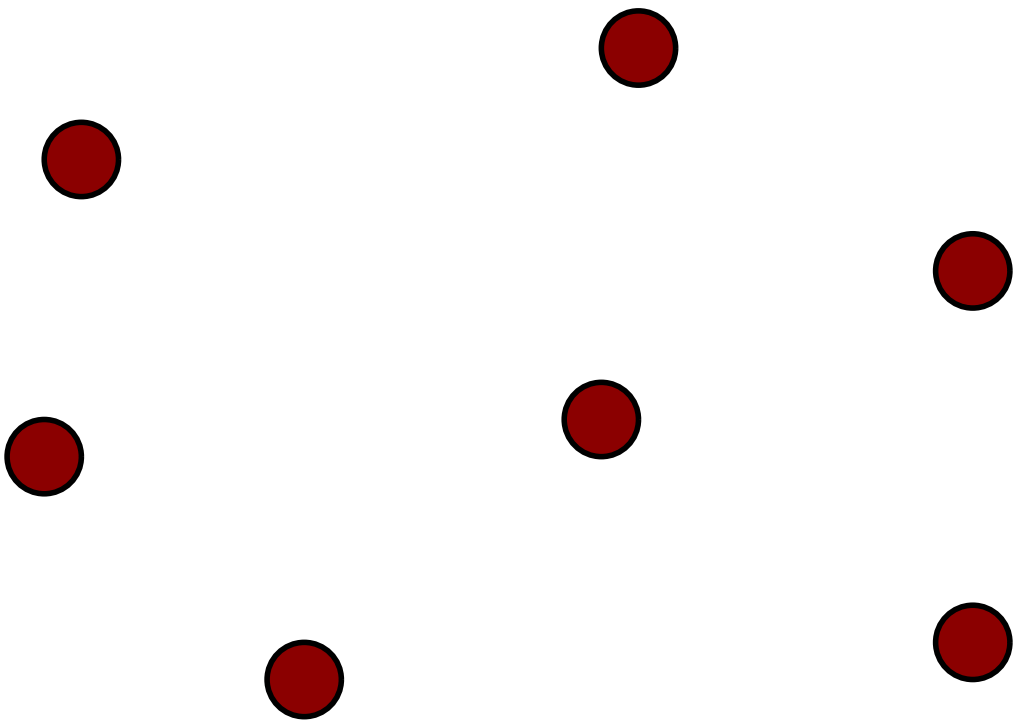
Monotone Graph Properties

Monotone Graph Properties

Does G have a 3-clique/triangle?

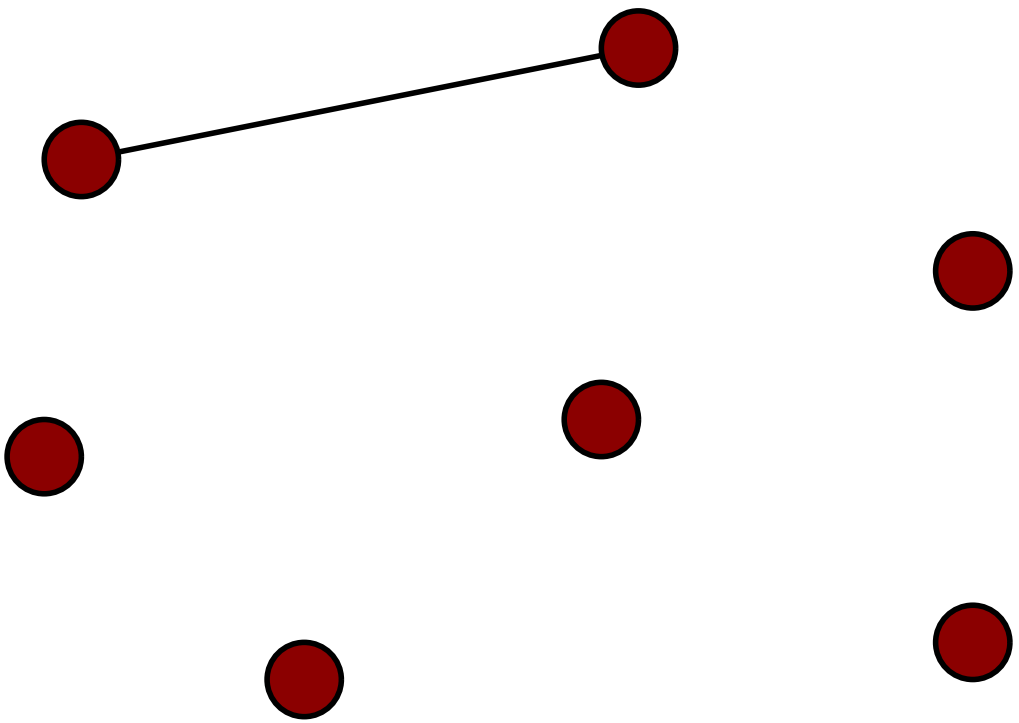
Monotone Graph Properties

Does G have a 3-clique/triangle?



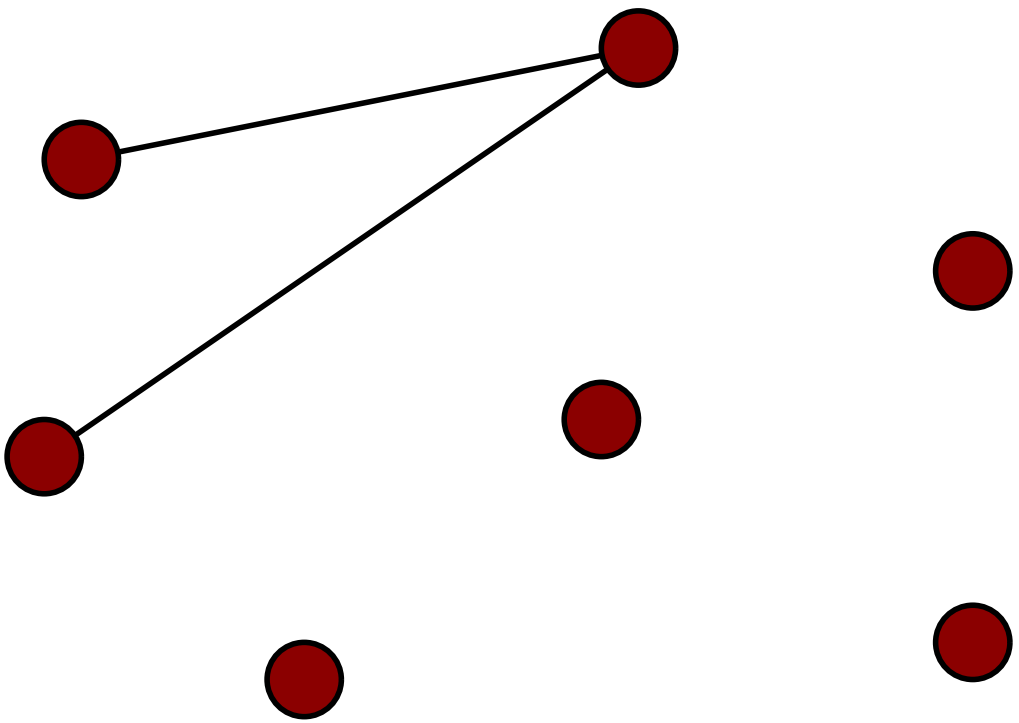
Monotone Graph Properties

Does G have a 3-clique/triangle?



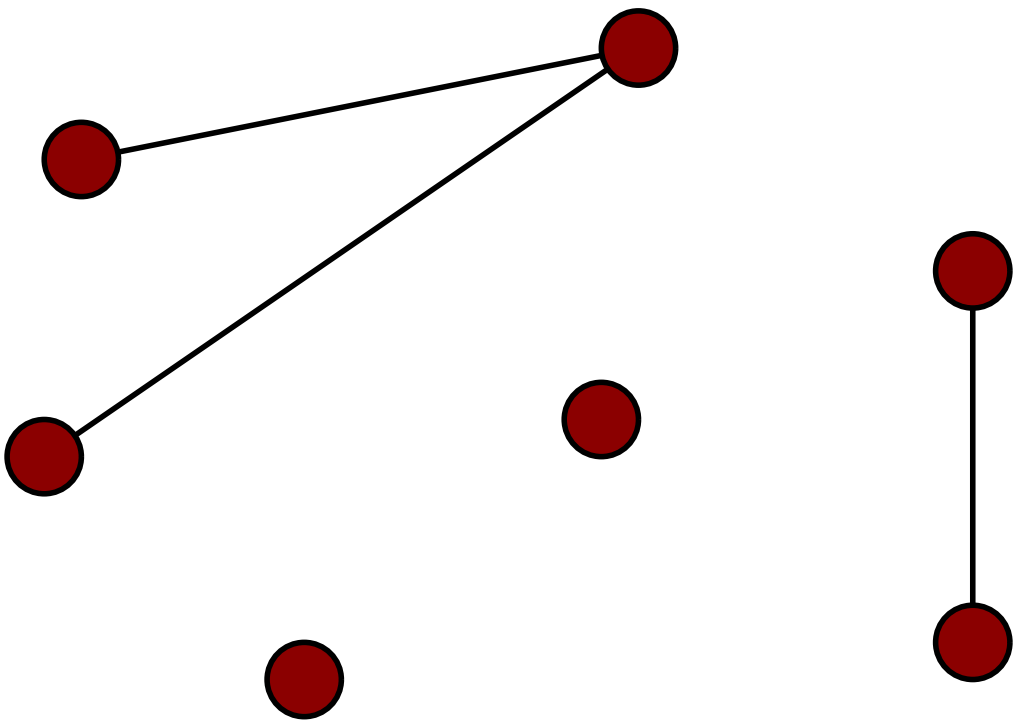
Monotone Graph Properties

Does G have a 3-clique/triangle?



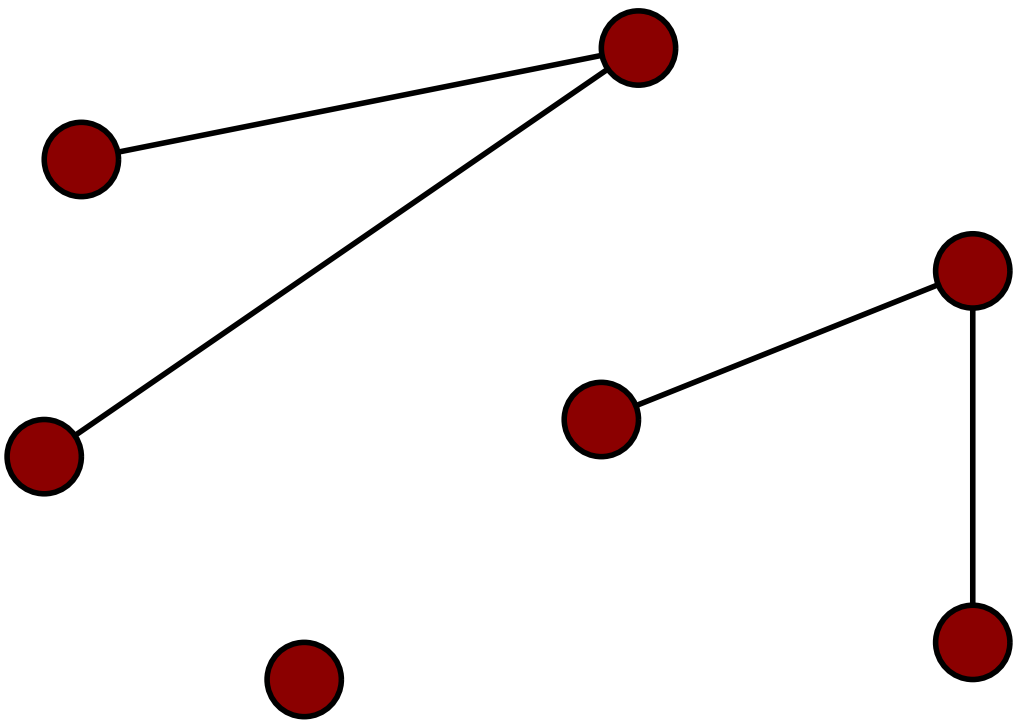
Monotone Graph Properties

Does G have a 3-clique/triangle?



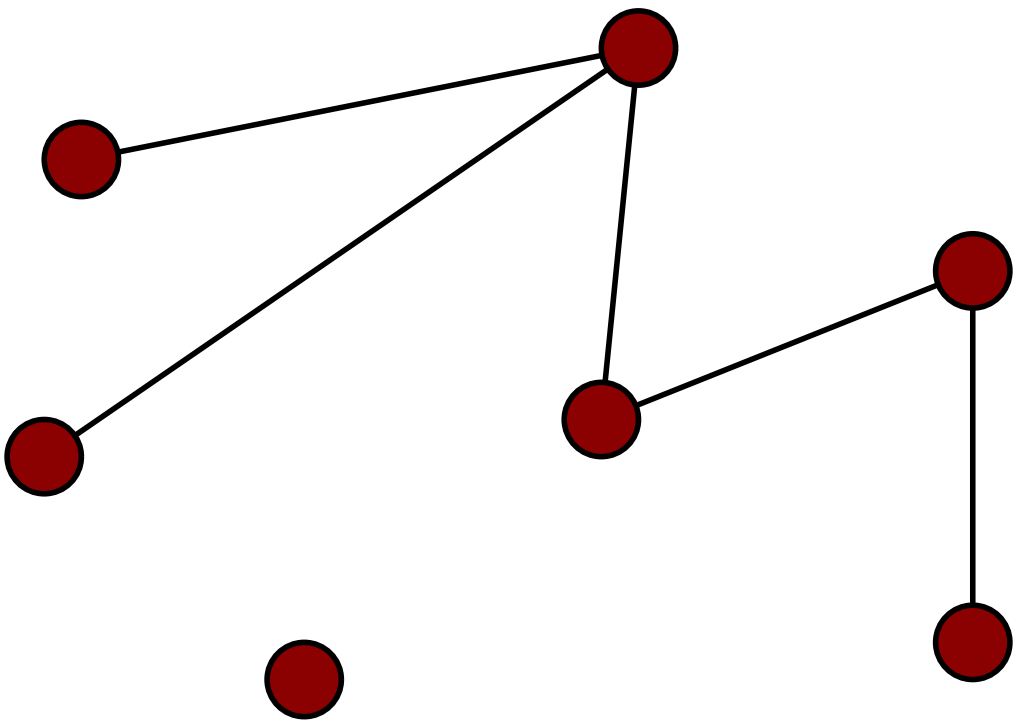
Monotone Graph Properties

Does G have a 3-clique/triangle?



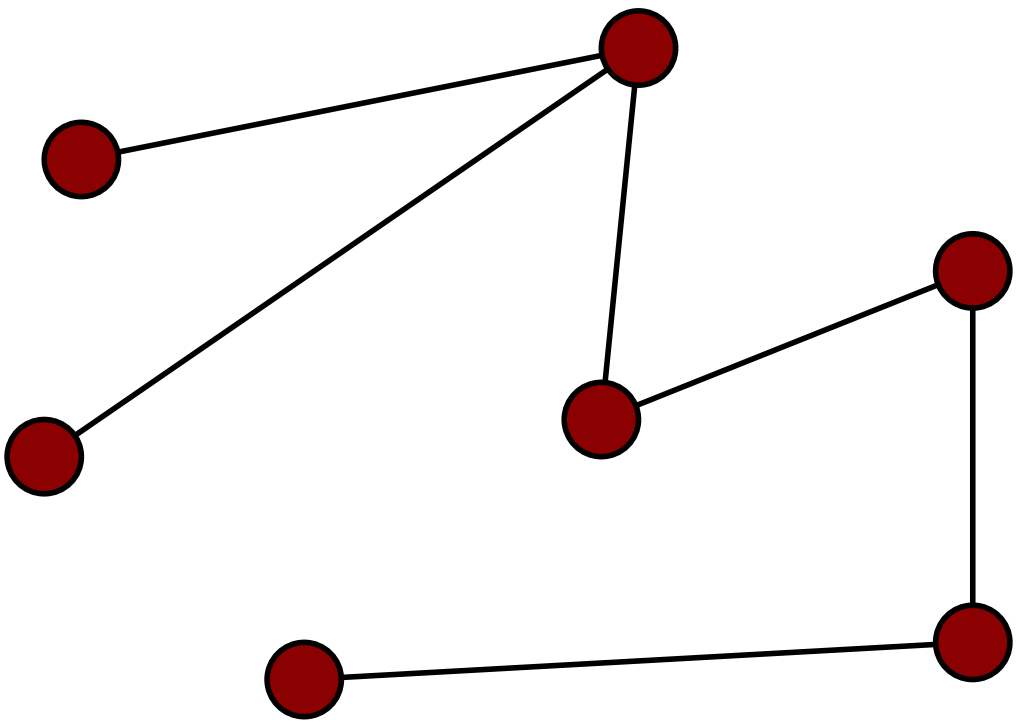
Monotone Graph Properties

Does G have a 3-clique/triangle?



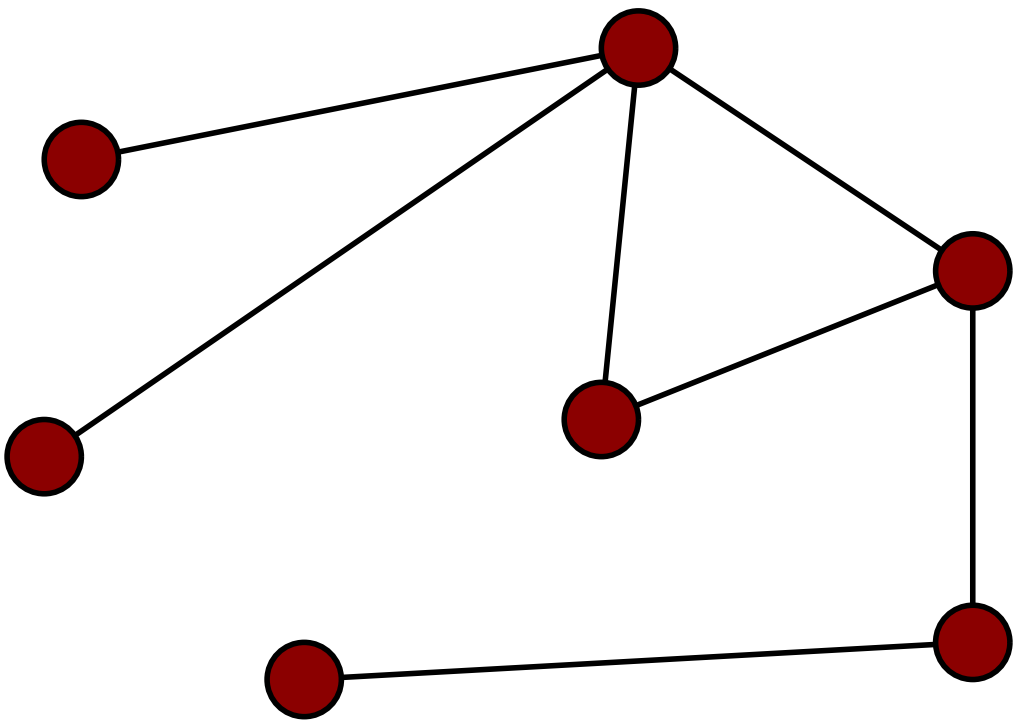
Monotone Graph Properties

Does G have a 3-clique/triangle?



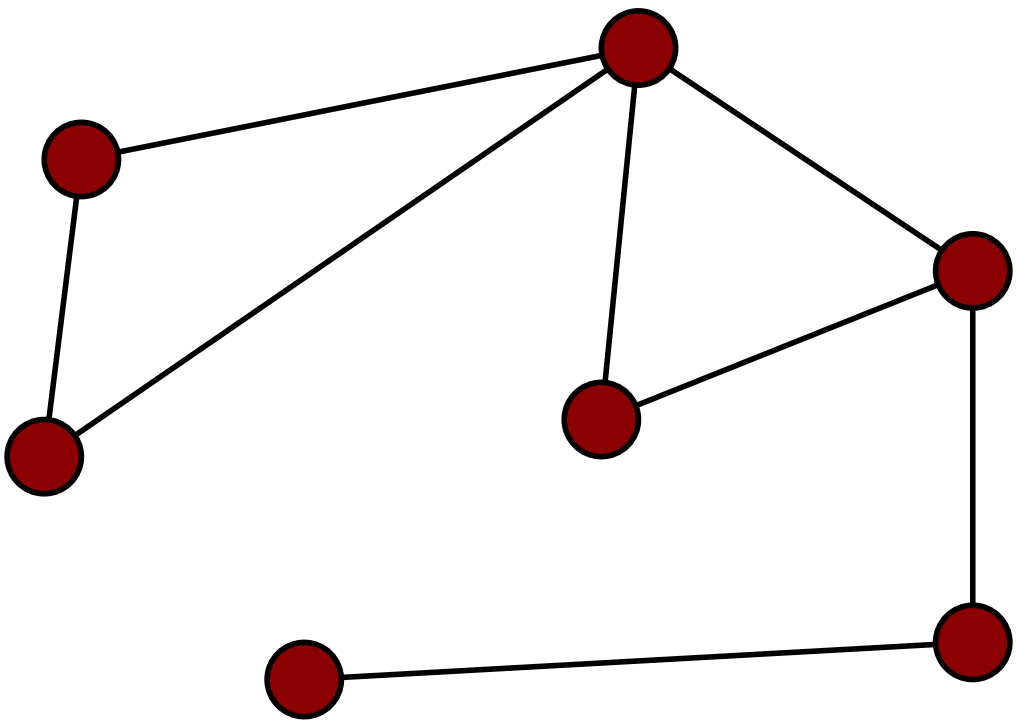
Monotone Graph Properties

Does G have a 3-clique/triangle?



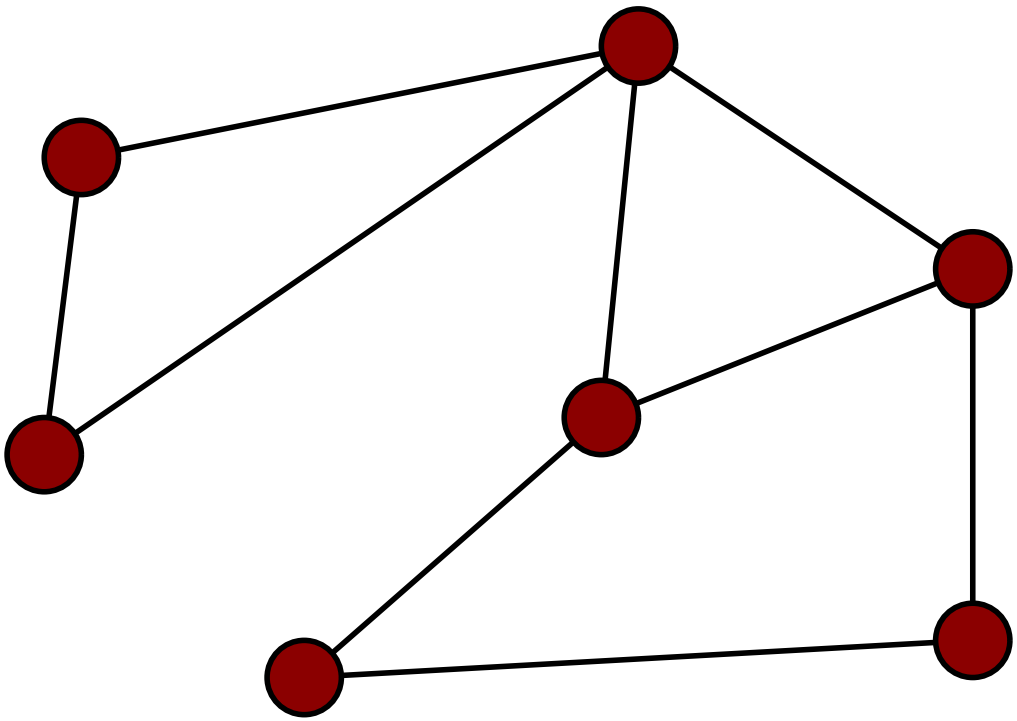
Monotone Graph Properties

Does G have a 3-clique/triangle?



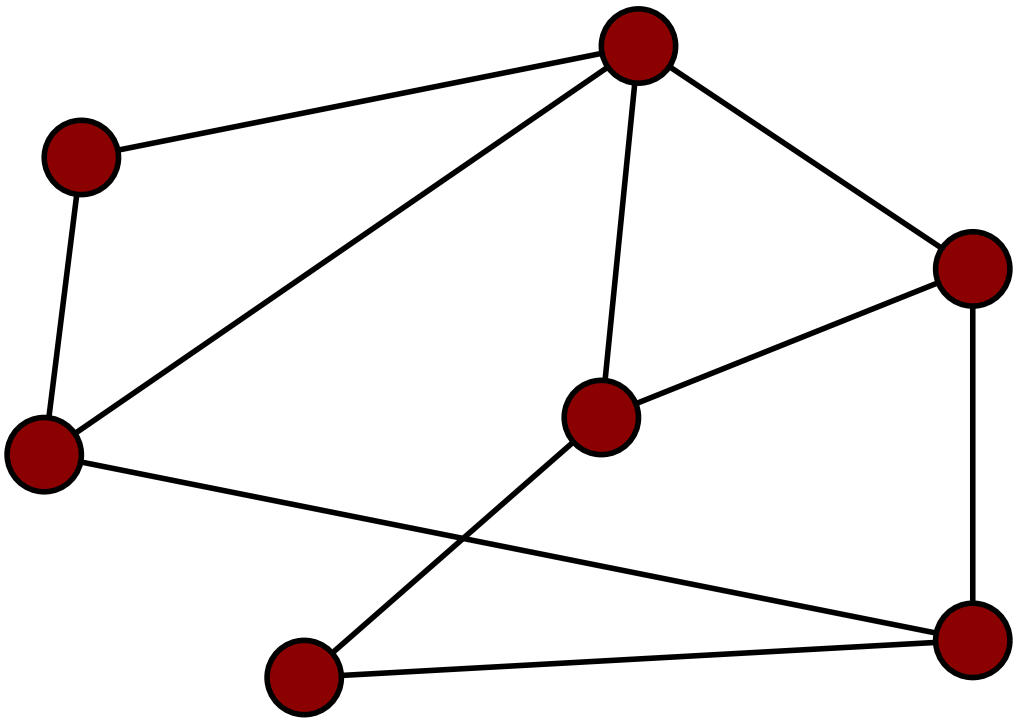
Monotone Graph Properties

Does G have a 3-clique/triangle?



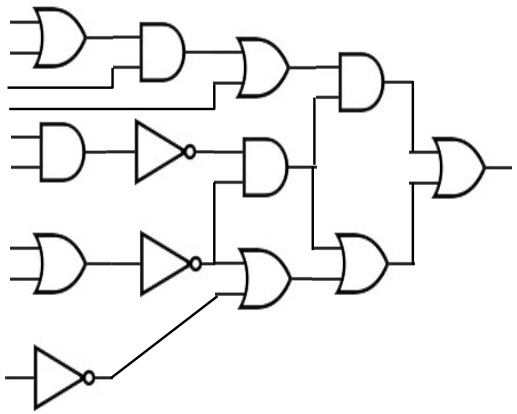
Monotone Graph Properties

Does G have a 3-clique/triangle?

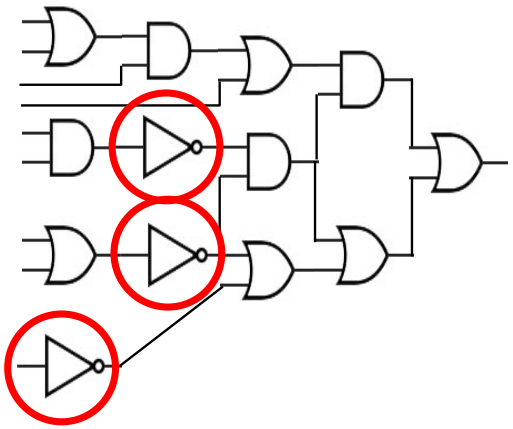


Monotone Circuits

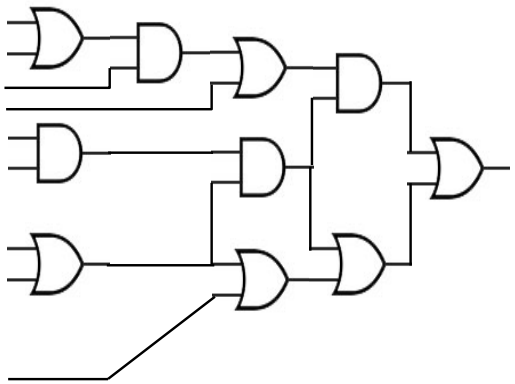
Monotone Circuits



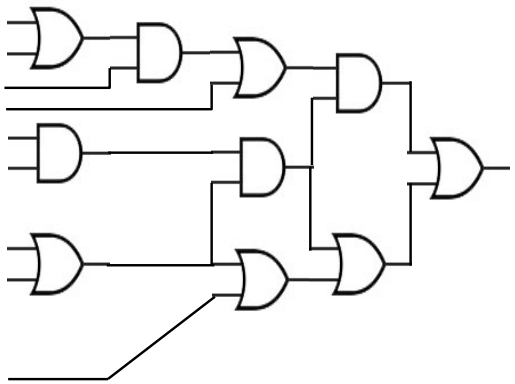
Monotone Circuits



Monotone Circuits

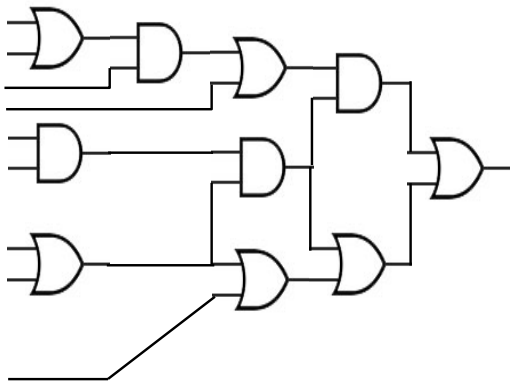


Monotone Circuits



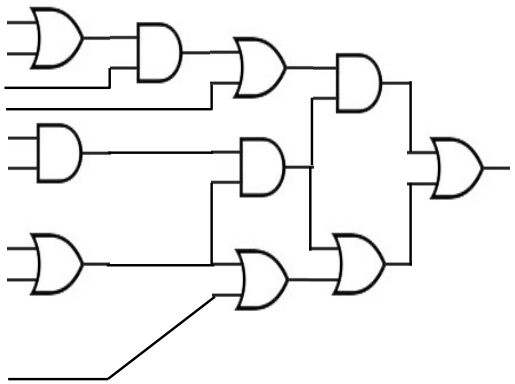
- [Razborov, ...] **Monotone circuit size of clique is super-poly**

Monotone Circuits



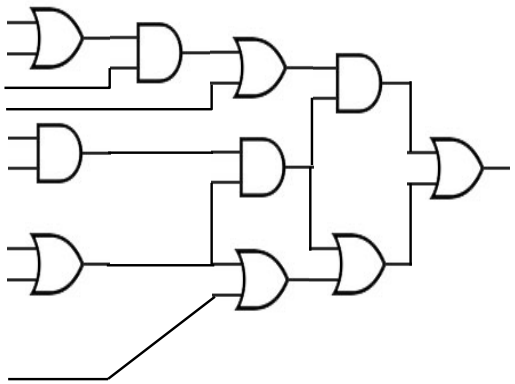
- [Razborov, ...] **Monotone circuit size of clique is super-poly**
- Average-case lower bounds, derandomization of randomized monotone circuits?

Monotone Circuits



- [Razborov, ...] **Monotone circuit size of clique is super-poly**
- Average-case lower bounds, derandomization of randomized monotone circuits?
- [Kearns-Valiant, ...] Monotone functions not highly average-case hard

Monotone Circuits



- [Razborov, ...] **Monotone circuit size of clique is super-poly**
- Average-case lower bounds, derandomization of randomized monotone circuits?
- [Kearns-Valiant, ...] Monotone functions not highly average-case hard
- Non-monotone functions?

Results

Results

Average-case hardness

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f
$\Rightarrow$ monotone C_{mon} within $(\frac{1}{2} - \epsilon')$ of f,

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f

$\Rightarrow$ monotone C_{mon} within $(\frac{1}{2} - \epsilon')$ of f, $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f
$\Rightarrow$ monotone C_{mon} within $(\frac{1}{2} - \epsilon')$ of f, $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

PRG's

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f
$\Rightarrow$ monotone C_{mon} within $(\frac{1}{2} - \epsilon')$ of f, $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

PRG's

ϵ-distinguisher for distribution (e.g. output of PRG)

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f
$\Rightarrow$ monotone C_{mon} within $(\frac{1}{2} - \epsilon')$ of f, $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

PRG's

ϵ-distinguisher for distribution (e.g. output of PRG)
$\Rightarrow$ monotone ϵ'-distinguisher, $\epsilon' = \max(\frac{\epsilon}{2(n+1)}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f
$\Rightarrow$ monotone C_{mon} within $(\frac{1}{2} - \epsilon')$ of f, $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

PRG's

ϵ-distinguisher for distribution (e.g. output of PRG)
$\Rightarrow$ monotone ϵ'-distinguisher, $\epsilon' = \max(\frac{\epsilon}{2(n+1)}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

Derandomization

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f
$\Rightarrow$ monotone C_{mon} within $(\frac{1}{2} - \epsilon')$ of f, $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

PRG's

ϵ-distinguisher for distribution (e.g. output of PRG)
$\Rightarrow$ monotone ϵ'-distinguisher, $\epsilon' = \max(\frac{\epsilon}{2(n+1)}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

Derandomization

If all languages solvable by bounded-error randomized monotone circuits are in P

Results

Average-case hardness

C within $(\frac{1}{2} - \epsilon)$ of f
$\Rightarrow$ monotone C_{mon} within $(\frac{1}{2} - \epsilon')$ of f, $\epsilon' = \max(\frac{\epsilon}{n+1}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

PRG's

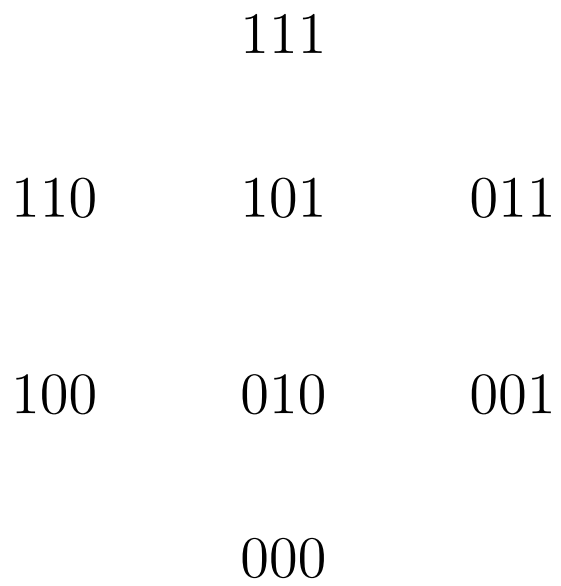
ϵ-distinguisher for distribution (e.g. output of PRG)
$\Rightarrow$ monotone ϵ'-distinguisher, $\epsilon' = \max(\frac{\epsilon}{2(n+1)}, \frac{c}{\sqrt{n \log(1/\epsilon)}})$

Derandomization

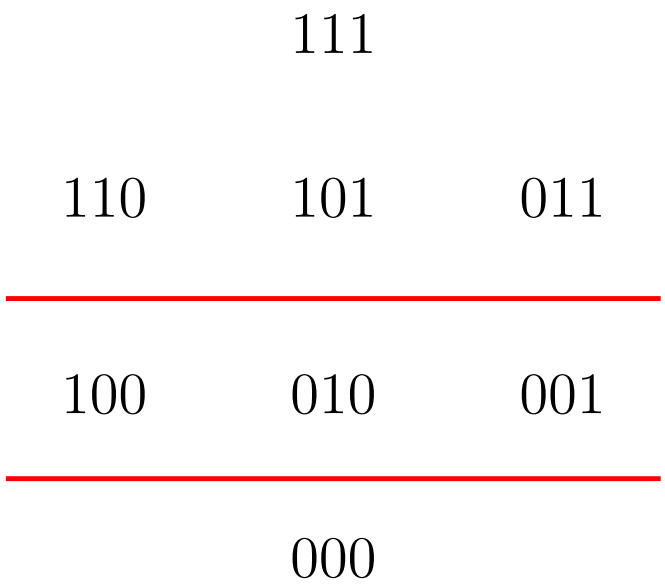
If all languages solvable by bounded-error randomized monotone circuits are in P $\Rightarrow$ BPP = P

Monotone Slice Functions

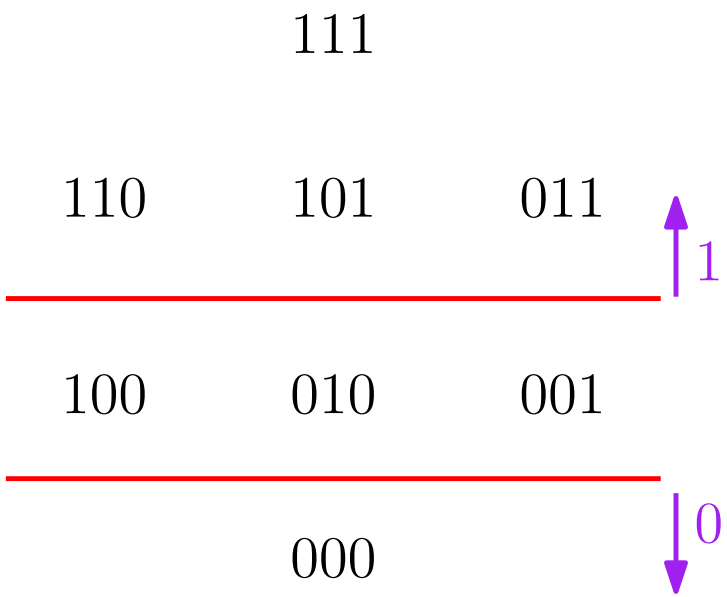
Monotone Slice Functions



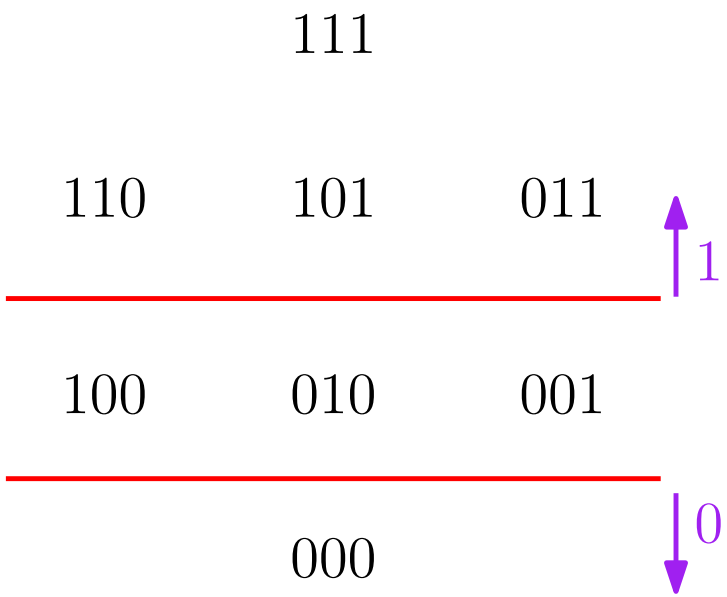
Monotone Slice Functions



Monotone Slice Functions

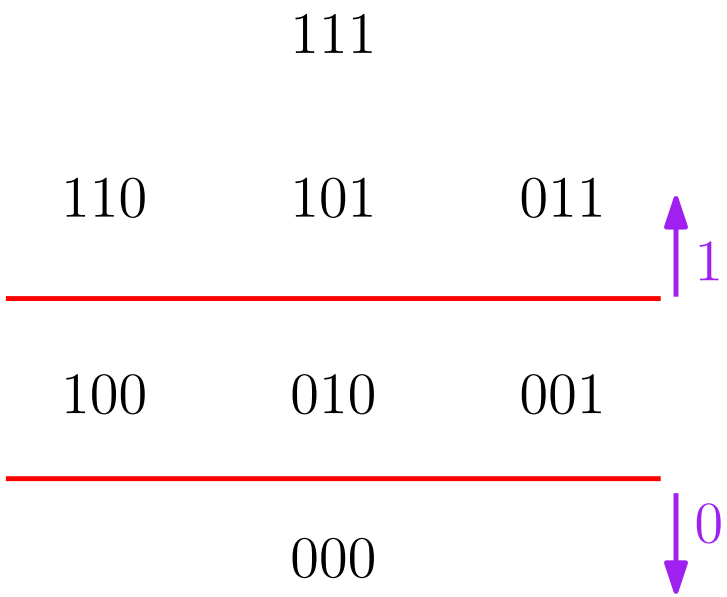


Monotone Slice Functions



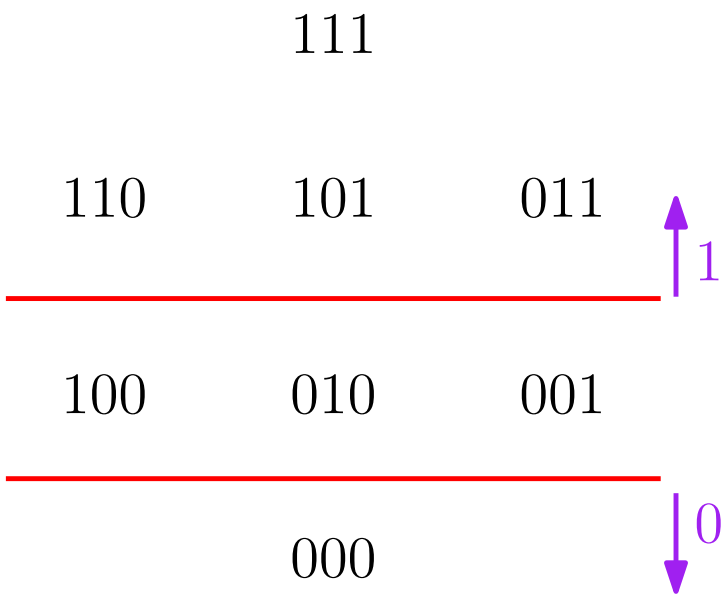
- i-th bit $= 0$

Monotone Slice Functions



- i-th bit $= 0$ iff at least one other bit is 1

Monotone Slice Functions



- i-th bit $= 0$ iff at least one other bit is 1
- **Slice function $\Rightarrow$ monotone and general circuit size nearly the same**

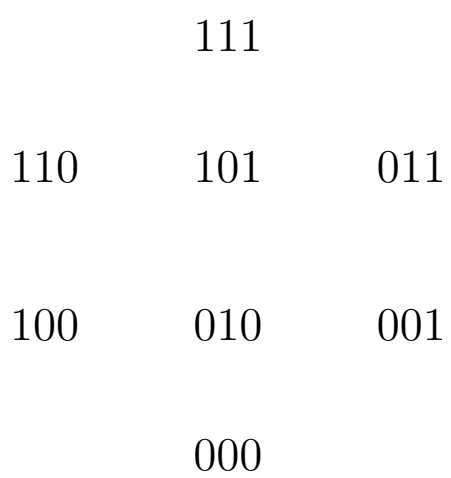
General versus Monotone Distinguishers

General versus Monotone Distinguishers

Distribution $\mathcal{D}$, general circuit C such that
$|\Pr_{X \leftarrow \mathcal{D}}[C(X) = 1] - \Pr_{X \leftarrow U_n}[C(X) = 1]| \geq \epsilon$

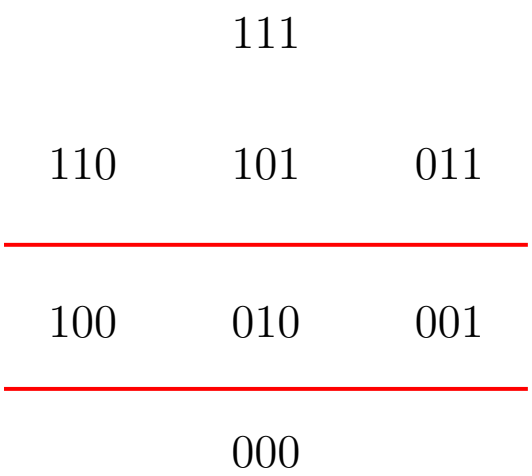
General versus Monotone Distinguishers

Distribution $\mathcal{D}$, general circuit C such that
$|\Pr_{X \leftarrow \mathcal{D}}[C(X) = 1] - \Pr_{X \leftarrow U_n}[C(X) = 1]| \geq \epsilon$



General versus Monotone Distinguishers

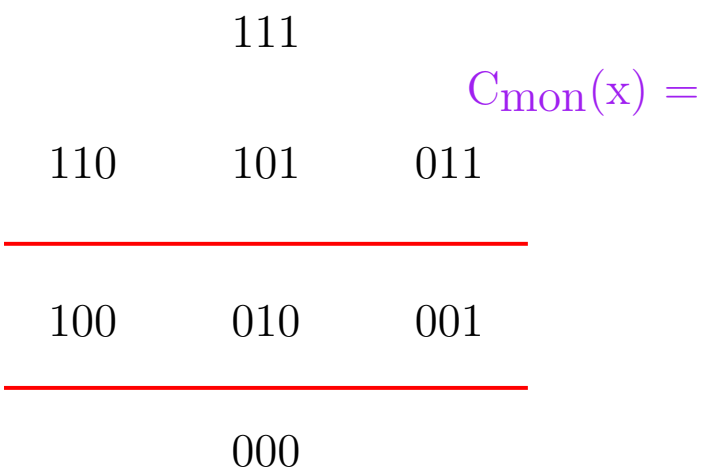
Distribution $\mathcal{D}$, general circuit C such that
$|\Pr_{X \leftarrow \mathcal{D}}[C(X) = 1] - \Pr_{X \leftarrow U_n}[C(X) = 1]| \geq \epsilon$



- C $(\frac{\epsilon}{n+1})$-distinguishes for x with |x|=k

General versus Monotone Distinguishers

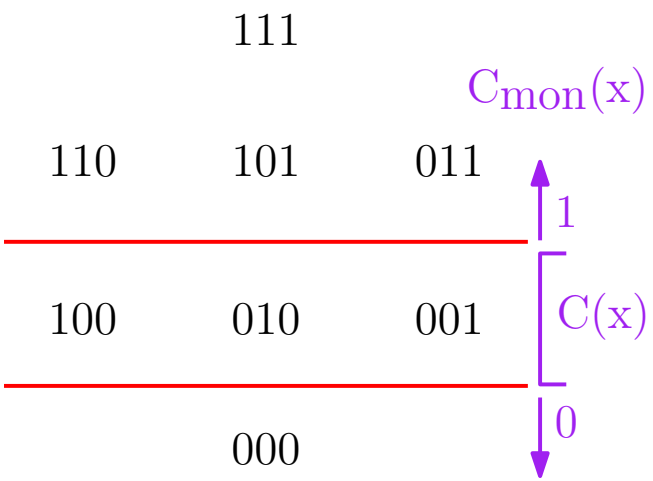
Distribution $\mathcal{D}$, general circuit C such that
$|\Pr_{X \leftarrow \mathcal{D}}[C(X) = 1] - \Pr_{X \leftarrow U_n}[C(X) = 1]| \geq \epsilon$



- C $(\frac{\epsilon}{n+1})$-distinguishes for x with |x|=k

General versus Monotone Distinguishers

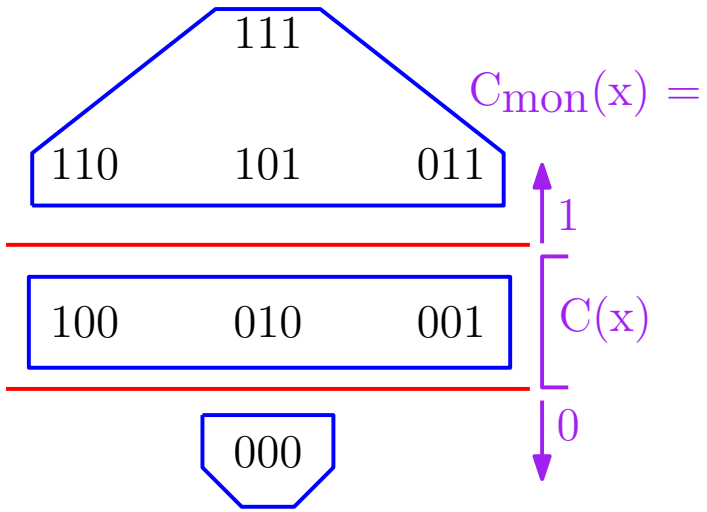
Distribution $\mathcal{D}$, general circuit C such that
$|\Pr_{X \leftarrow \mathcal{D}}[C(X) = 1] - \Pr_{X \leftarrow U_n}[C(X) = 1]| \geq \epsilon$



- C $(\frac{\epsilon}{n+1})$-distinguishes for x with |x|=k

General versus Monotone Distinguishers

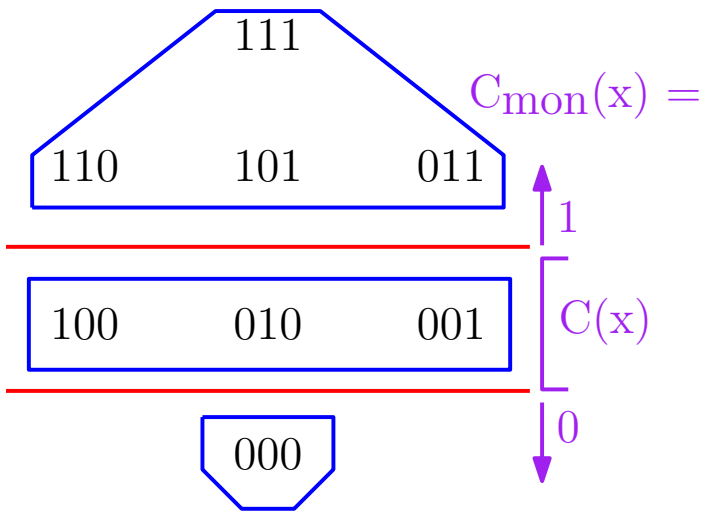
Distribution $\mathcal{D}$, general circuit C such that
$|\Pr_{X \leftarrow \mathcal{D}}[C(X) = 1] - \Pr_{X \leftarrow U_n}[C(X) = 1]| \geq \epsilon$



- C $(\frac{\epsilon}{n+1})$-distinguishes for x with |x|=k

General versus Monotone Distinguishers

Distribution $\mathcal{D}$, general circuit C such that
$|\Pr_{X \leftarrow \mathcal{D}}[C(X) = 1] - \Pr_{X \leftarrow U_n}[C(X) = 1]| \geq \epsilon$



- C $(\frac{\epsilon}{n+1})$-distinguishes for x with |x|=k
- If threshold is not $\epsilon' = (\frac{\epsilon}{2(n+1)})$-distinguisher, then C_{mon} is

Recap

How Powerful is Randomness?

- "Typically-correct" Derandomization
- Hierarchy Theorems for Randomized Algorithms
- Derandomizing Monotone Computations

The End, Thank You!

The End, Thank You!

Slides will be available at:

http://www.kinnejeff.com/ (or E-mail me)