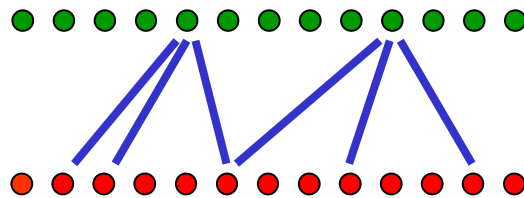
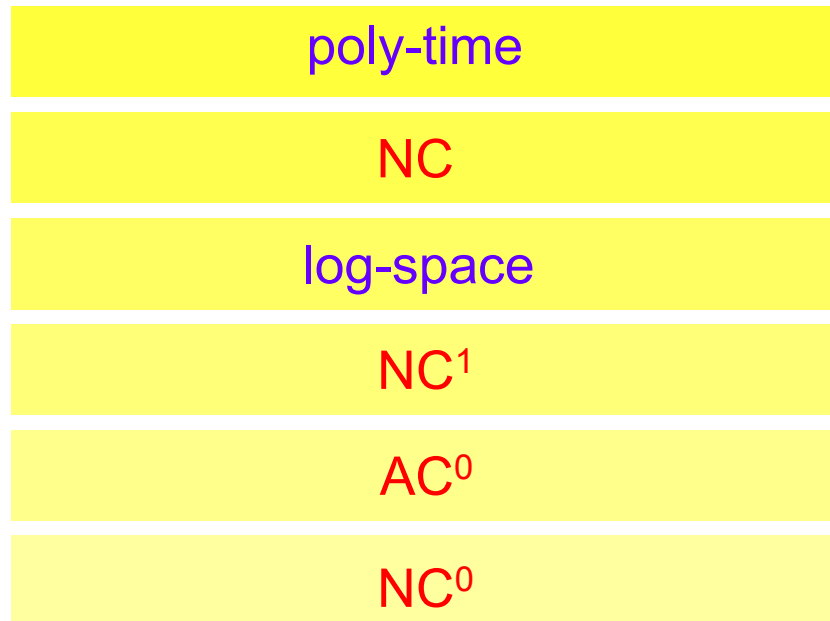


Outline

1. (Long) Introduction
2. Randomized Polynomials (w/applications to round-efficient MPC)
3. Randomized Encodings w/applications to NC^0 Cryptography
4. Constant Input Locality
5. Computational Randomized Encodings (w/applications)
6. NC^0 Linear Stretch PRG (w/applications)

Parallel Cryptography

How low can we get?



Sufficient Assumptions for Crypto in NC^0

Caveats:

- We get PRG with sub-linear stretch
- decryption verification not in NC^0
 - Infeasible to decrypt/verify [AKK05]
 - ... commit in NC^0 with constant commitment in NC^0 [AND]

Assuming min-PRG in NC^1

Hash

Sym-Enc

PK-Enc

Signature

Commit

NIZK

OWF

PRG

Hash

Sym-Enc

PK-Enc

Signature

Commit

NIZK

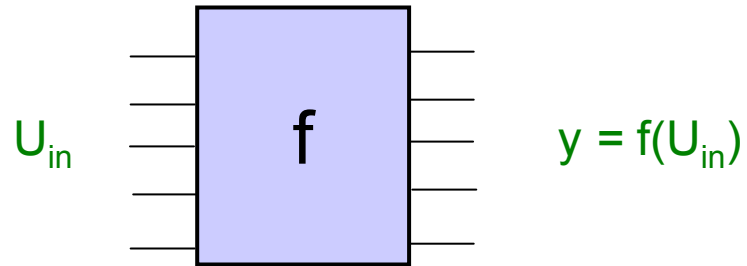
$\in NC^0$
[AND]

factoring, discrete-log/DDH, lattices, ...

P								factoring
NC^1								factoring
NC^0_4								factoring
	OWF	PRG	Hash	Sym-Enc	PK-Enc	NI-Com	Sign	NIZK

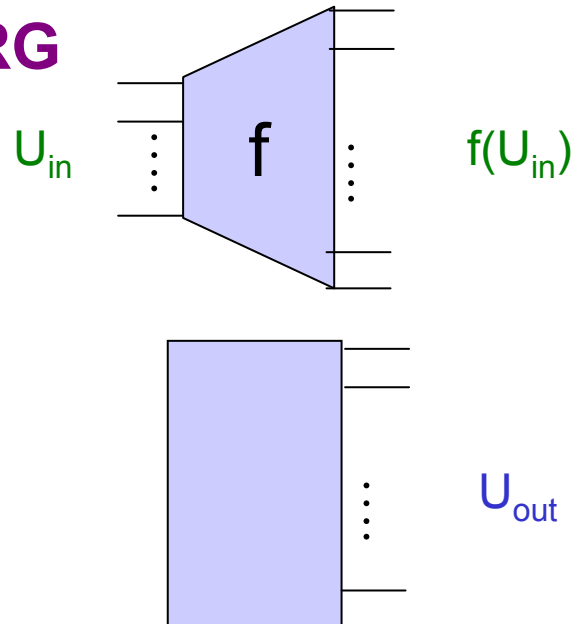
Main Primitives

OWF



poly-time

PRG

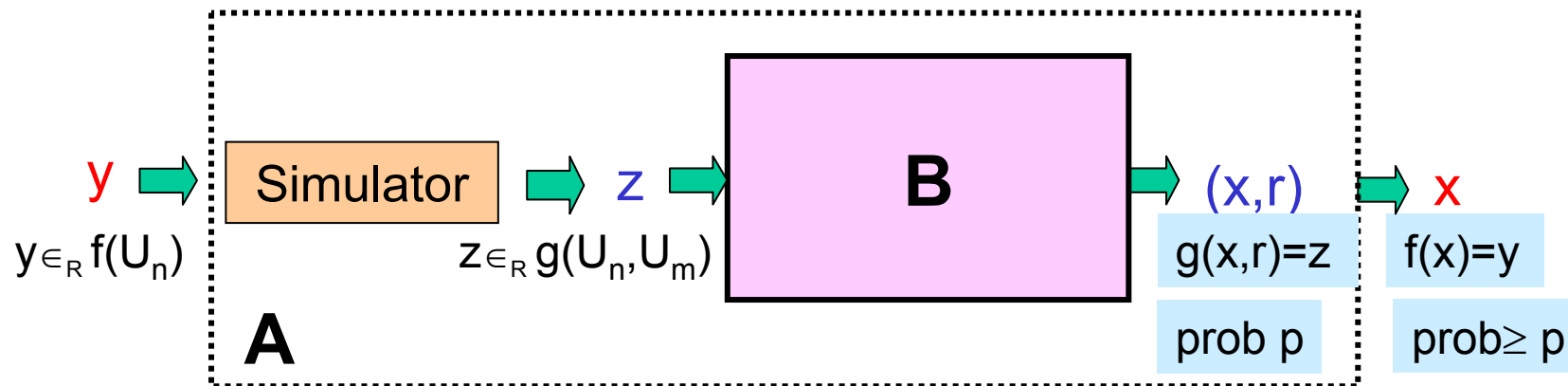


poly-time

Encoding a OWF

Thm. $f(x)$ is a OWF $\Rightarrow g(x,r)$ is a OWF

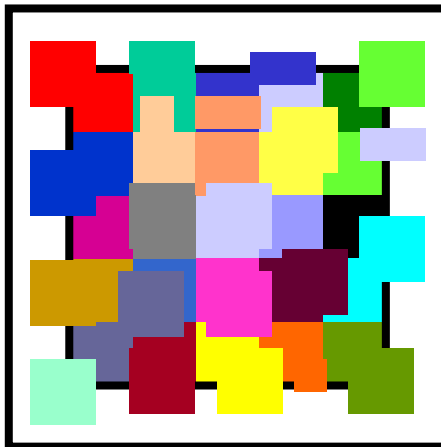
Proof: inverter B for $g \Rightarrow$ inverter A for f



- A succeeds whenever B succeeds
 - Follows from **perfect** correctness of decoder
- A generates a correct input distribution for B
 - Follows from correctness of simulator

Encoding a PRG

- **Want:** $f(x)$ is a PRG \Rightarrow $g(x,r)$ is a PRG
- **Problems:**
 - output of g may not be pseudorandom
 - g may *shrink* its input



- **Solution:** “perfect” randomized encoding
 - respects pseudorandomness, additive stretch, ...
 - stretch of g is typically sublinear even if that of f is superlinear
 - most (not all) known constructions give perfectness for free

From Low Degree to Low Locality

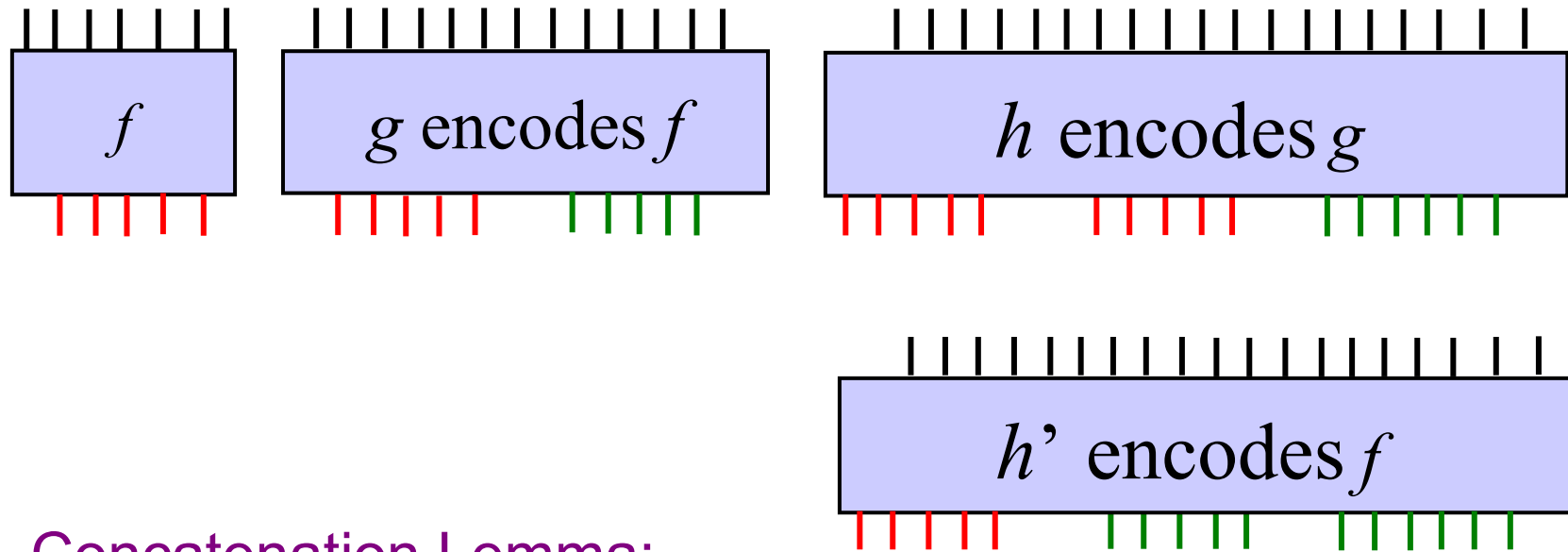
- **Locality Reduction:**
 degree 3 boolean function \Rightarrow locality ~~3~~⁴ rand. encoding

$$f(x) = T_1(x) + T_2(x) + \dots + T_k(x)$$

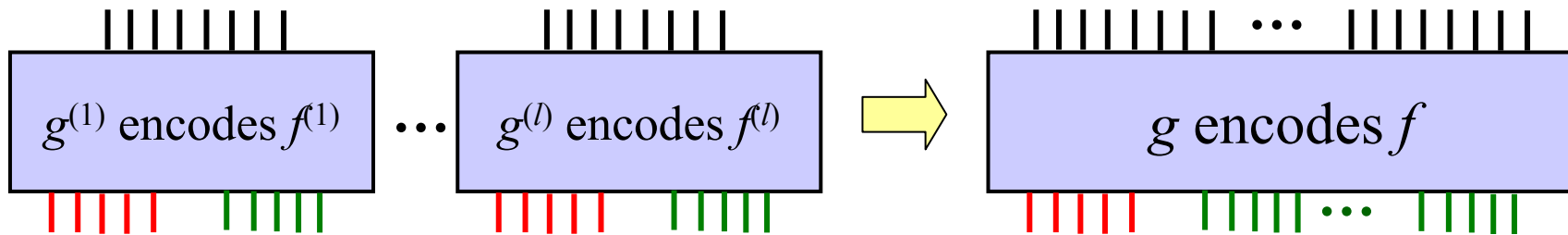
$$g(x, r) = \begin{array}{cccc} T_1(x) + r_1 & -r_1 + T_2(x) + r_2 & \dots & -r_{k-1} + T_k(x) \\ -r_1 + s_1 & -s_1 - r_2 + s_2 & \dots & -s_{k-1} - r_k \end{array}$$

Wrapping Up

Composition Lemma:



Concatenation Lemma:



From Branching Programs to Locality 4

