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?

poly-time

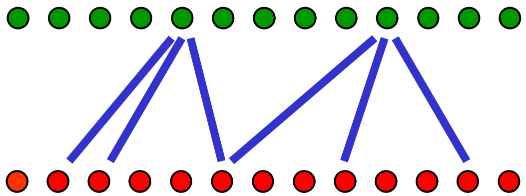
NC

log-space

NC^1

AC^0

NC^0



Sufficient Assumptions for Crypto in NC^0

Caveats:

- We g... PRG ... with sub-linear st...
- decry... verification not in ...
  - In ... ssible to decrypt/ver...
  - ... commit in NC^0 with ... in NC^0[AND]

Assuming min-PRG in NC^1

PRG
Hash
Sym-Enc
PK-Enc
Signature
Commit
NIZK

exist NC^1 [AIK04] [AIK05] $\Rightarrow$

OWF
PRG
Hash
Sym-Enc
PK-Enc
Signature
Commit
NIZK

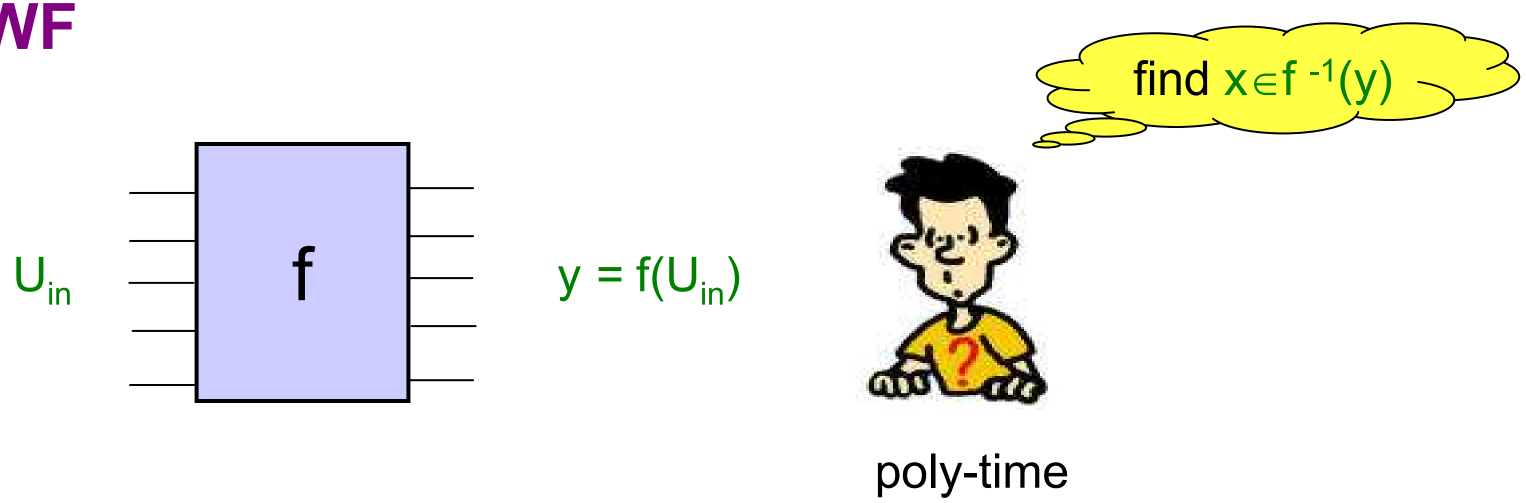
$\in NC^0$

factoring, discrete-log/DDH, lattices, …

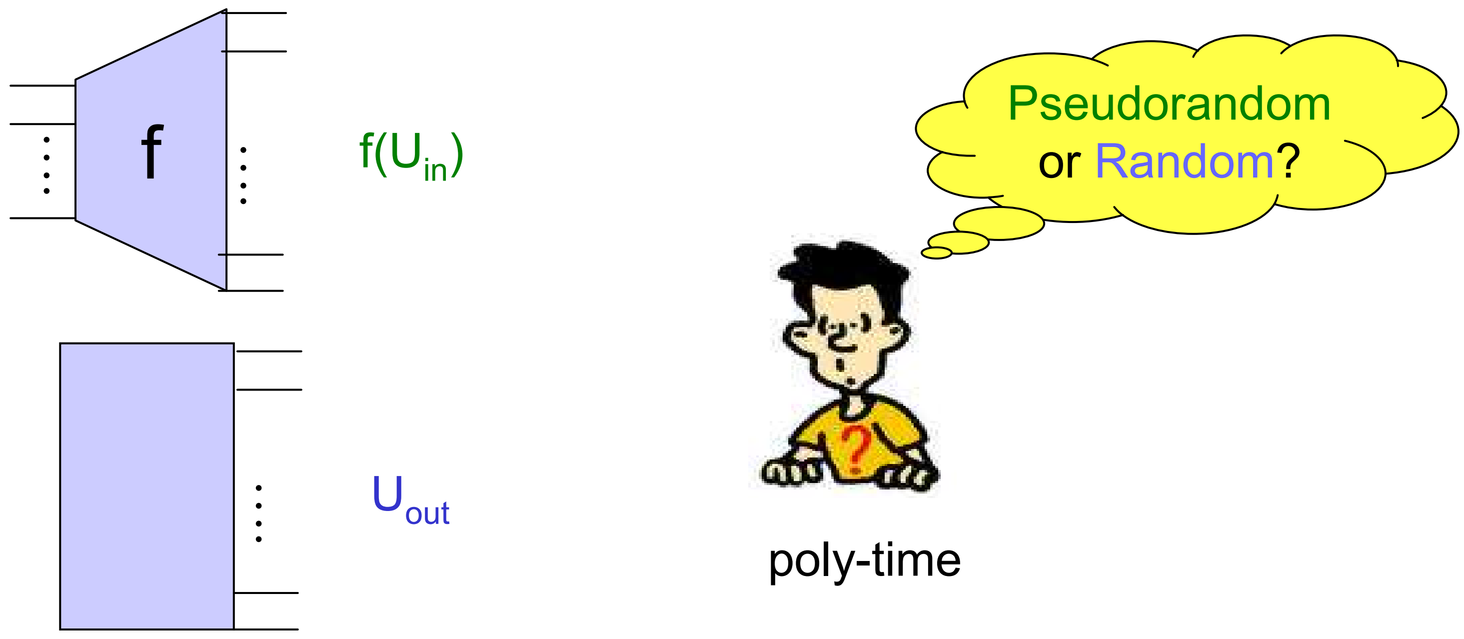
	OWF	PRG	Hash	Sym-Enc	PK-Enc	NI-Com	Sign	NIZK
P								factoring
NC^1								factoring
NC^0_4								factoring

Main Primitives

OWF



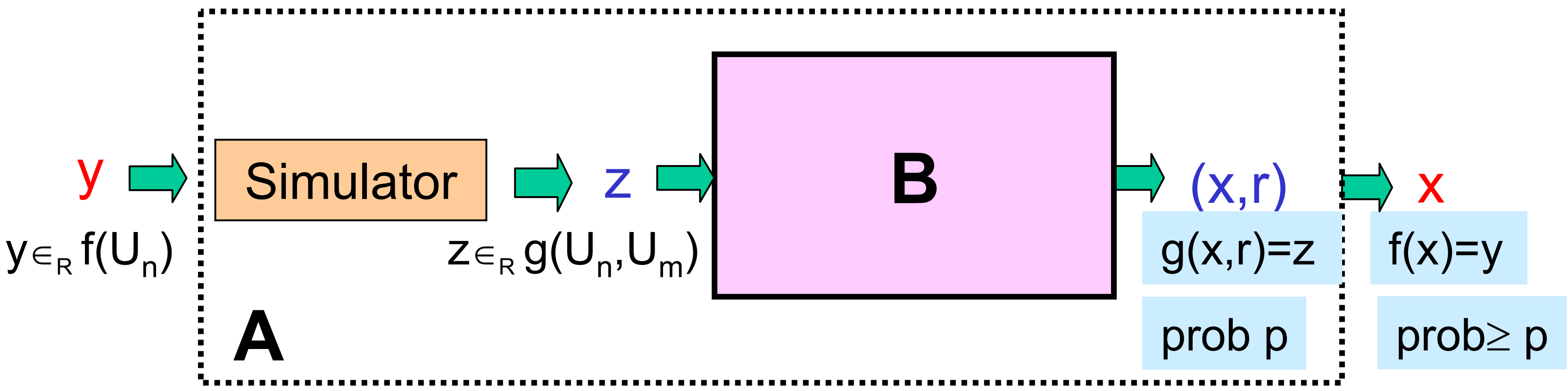
PRG



Encoding a OWF

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

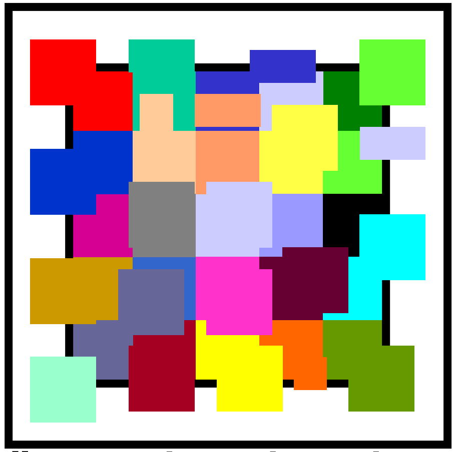
Proof: inverter B for g $\Rightarrow$ inverter A for f



- 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 ⇒ locality ~~5~~ 4 rand. encoding

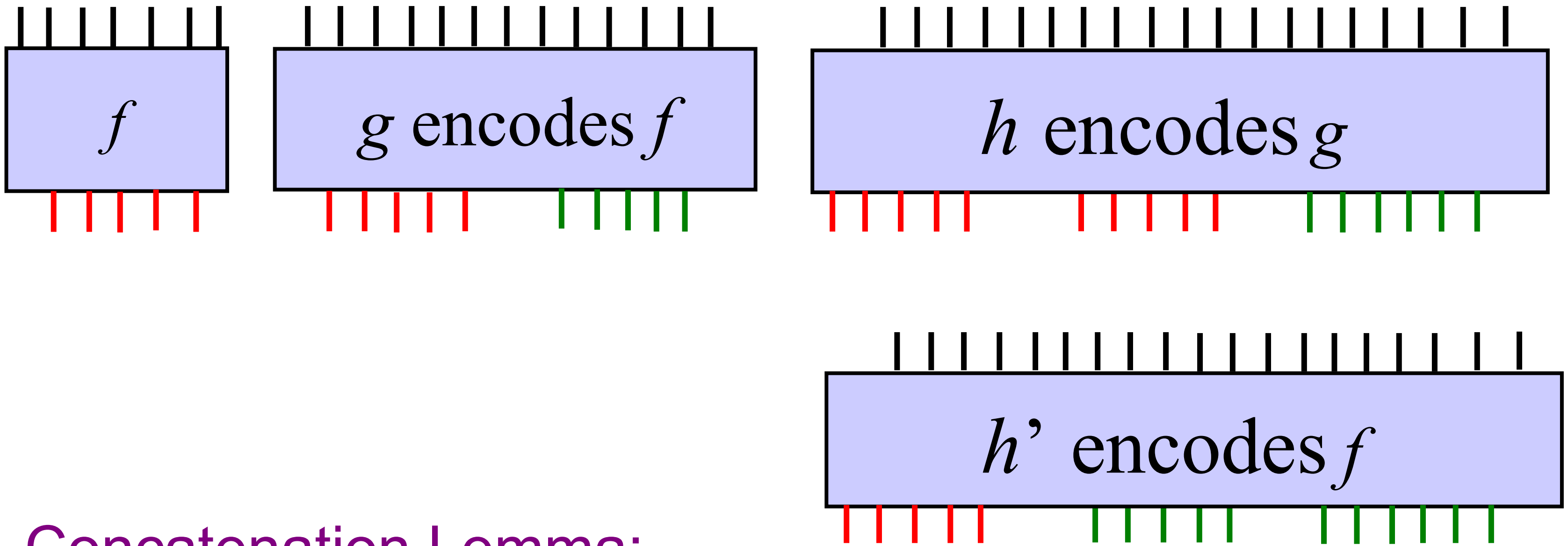
$$f(x) \quad = \quad T_1(x) \quad + \quad T_2(x) \quad + \quad \dots \quad + \quad T_k(x)$$

$$g(x,r,s) = \quad \boxed{T_1(x)+r_1} \quad \boxed{-r_1+T_2(x)+r_2} \quad \cdots \quad \boxed{-r_{k-1}+T_k(x)}$$

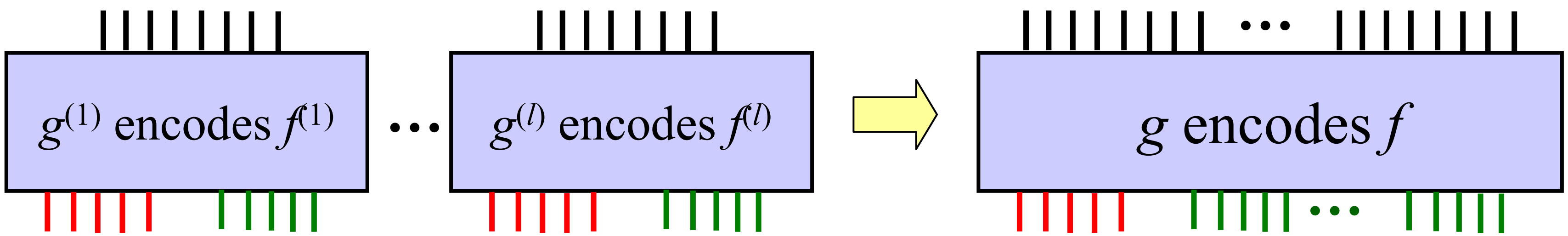
$$\boxed{-r_1+s_1} \quad \boxed{-s_1-r_2+s_2} \quad \cdots \quad \boxed{-s_{k-1}-r_k}$$

Wrapping Up

Composition Lemma:



Concatenation Lemma:



From Branching Programs to Locality 4