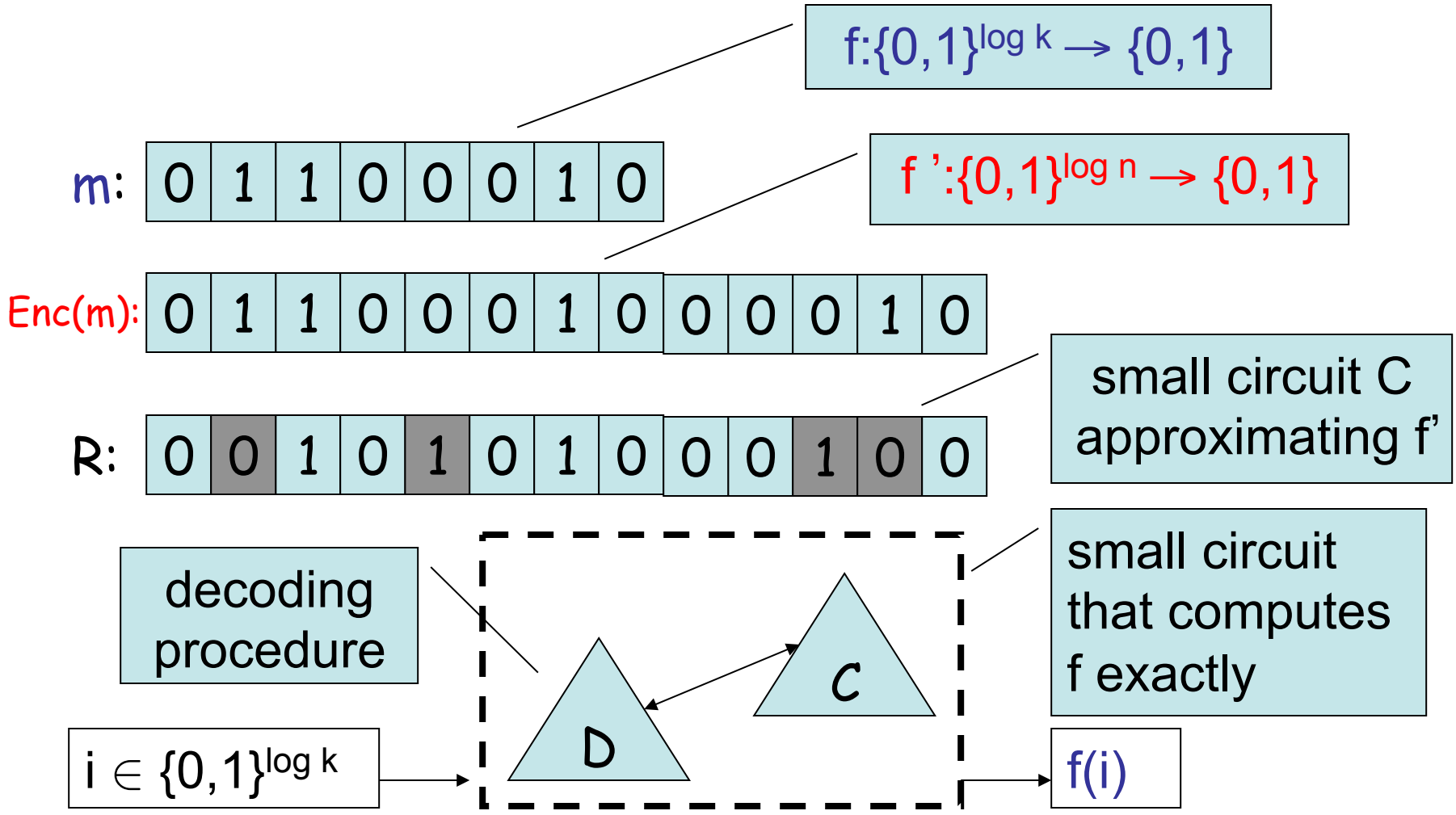
CS151
Complexity Theory

Lecture 10

May 3, 2017

Codes and Hardness



Encoding

- use a (variant of) Reed-Muller code *concatenated* with the Hadamard code
  - q (field size), t (dimension), h (degree)
- **encoding procedure**:
  - message $m \in \{0,1\}^k$
  - subset $S \subseteq F_q$ of size h

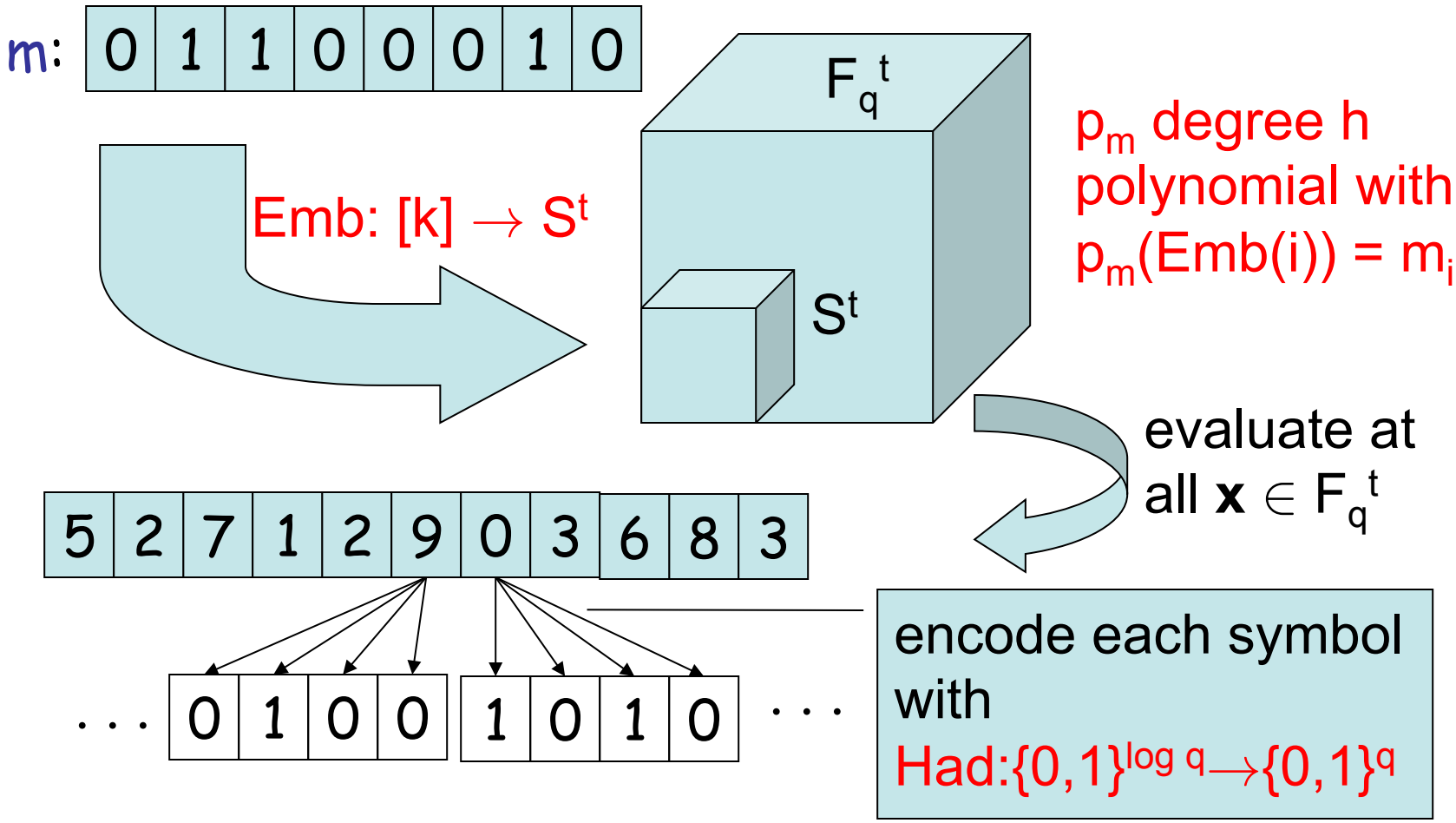
so, need $h^t \geq k$

  - efficient 1-1 function Emb: $[k] \rightarrow S^t$
  - find coeffs of degree h polynomial $p_m: \mathbf{F}_q^t \rightarrow \mathbf{F}_q$ for which $p_m(\text{Emb}(i)) = m_i$ for all i (linear algebra)

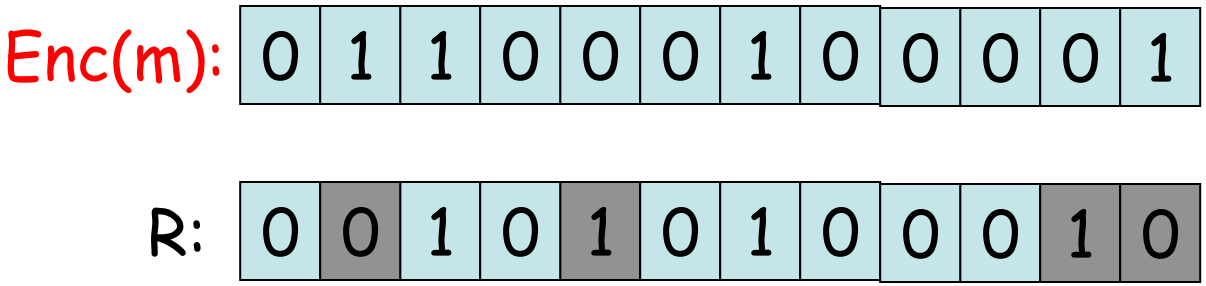
Encoding

- **encoding procedure** (continued):
  - Hadamard code $\mathrm{Had}:\{0,1\}^{\log q} \to \{0,1\}^{q}$
    - = Reed-Muller with field size 2, dim. log q, deg. 1
    - distance ½ by Schwartz-Zippel
  - final codeword: $(\mathrm{Had}(p_m(\mathbf{x})))_{x \in F_q^t}$
    - evaluate p_m at all points, and encode each evaluation with the Hadamard code

Encoding


m:
0 1 1 0 0 0 1 0
Emb: [k] → S^t
F_q^t
S^t
p_m degree h polynomial with p_m(Emb(i)) = m_i
evaluate at all x ∈ F_q^t
5 2 7 1 2 9 0 3 6 8 3
. . .
0 1 0 0 1 0 1 0
. . .
encode each symbol with
Had:{0,1}^log q → {0,1}^q

Decoding

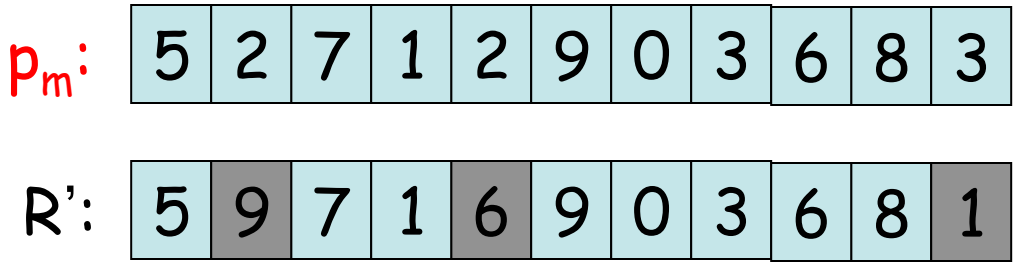


- small circuit C computing R, agreement ½ + δ
- **Decoding step 1**
  - produce circuit C' from C
    - given $\mathbf{x} \in F_q^t$ outputs "guess" for $p_m(\mathbf{x})$
    - C' computes {z : Had(z) has agreement ½ + δ/2 with x-th block}, outputs random z in this set

Decoding

- **Decoding step 1** (continued):
  - for at least $\delta/2$ of blocks, agreement in block is at least $\frac{1}{2} + \delta/2$
  - Johnson Bound: when this happens, list size is $S = O(1/\delta^2)$, so probability C' correct is $1/S$
  - altogether:
    - $\Pr_x[C'(x) = p_m(x)] \geq \Omega(\delta^3)$
    - C' makes q queries to C
    - C' runs in time poly(q)

Decoding



- small circuit C' computing R', agreement $\delta' = \Omega(\delta^3)$
- **Decoding step 2**
  - produce circuit C'' from C'
    - given $\mathbf{x} \in$ emb(1,2,…,k) outputs $p_m(\mathbf{x})$
    - idea: restrict p_m to a random curve; apply efficient R-S list-decoding; fix "good" random choices

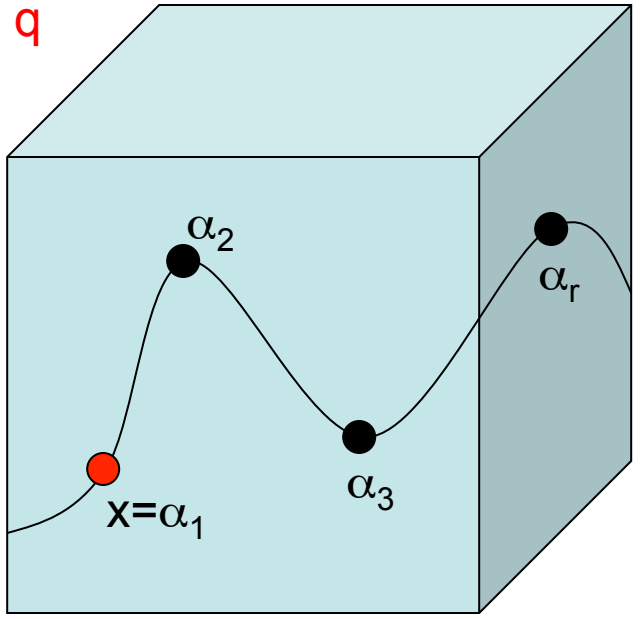
Restricting to a curve

– points $x=\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_r \in F_q^t$ specify a degree r curve $L : F_q \rightarrow F_q^t$

- $w_1, w_2, \ldots, w_r$ are distinct elements of F_q
- for each i, $L_i : F_q \rightarrow F_q$

is the degree r poly for which $L_i(w_j) = (\alpha_j)_i$ for all j

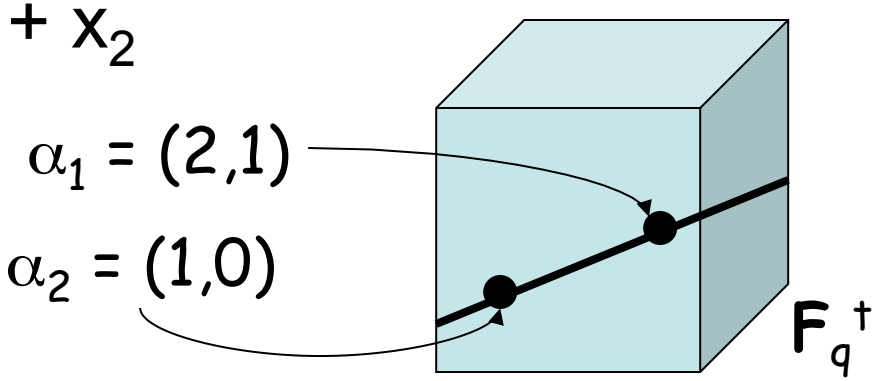
- Write $p_m(L(z))$ to mean $p_m(L_1(z), L_2(z), \ldots, L_t(z))$
- $p_m(L(w_1)) = p_m(x)$



degree r·h·t *univariate* poly

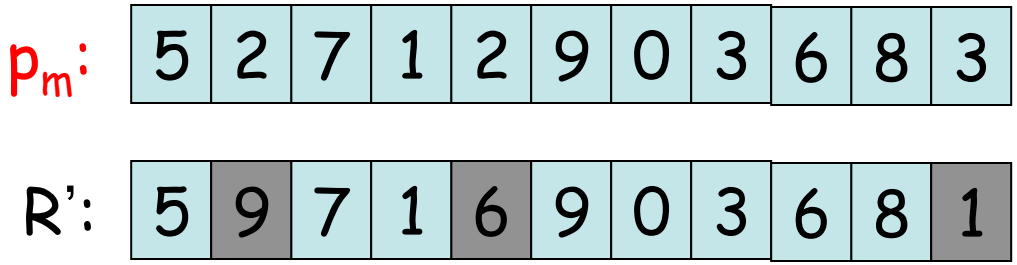
Restricting to a curve

- Example:
  - $p_m(x_1, x_2) = x_1^2 x_2^2 + x_2$
  - $w_1 = 1, w_2 = 0$



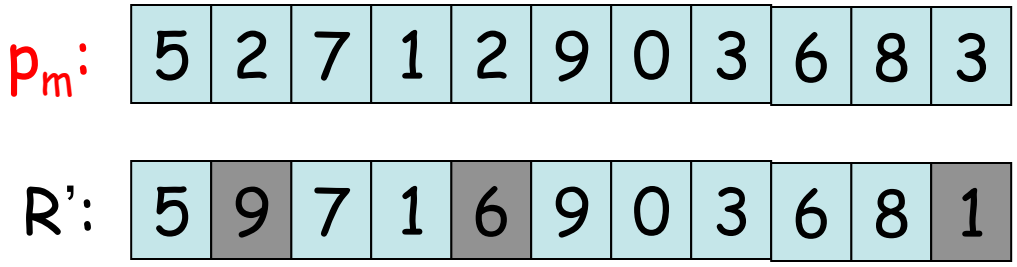
  - $L_1(z) = 2z + 1(1-z) = z + 1$
  - $L_2(z) = 1z + 0(1-z) = z$
  - $p_m(L(z)) = (z+1)^2 z^2 + z = z^4 + 2z^3 + z^2 + z$

Decoding



- small circuit C' computing R', agreement $\delta' = \Omega(\delta^3)$
- **Decoding step 2** (continued):
  - pick random $w_1, w_2, \ldots, w_r; \alpha_2, \alpha_3, \ldots, \alpha_r$ to determine curve L
  - points on L are (r-1)-wise independent
  - random variable: $Agr = |\{z : C'(L(z)) = p_m(L(z))\}|$
  - $E[Agr] = \delta' q$ and $Pr[Agr < (\delta' q)/2] < O(1/(\delta' q))^{(r-1)/2}$

Decoding



- small circuit C' computing R', agreement $\delta' = \Omega(\delta^3)$
- **Decoding step 2** (continued):
  - agr = $|\{z : C'(L(z)) = p_m(L(z))\}|$ is $\geq (\delta' q)/2$ with very high probability
  - compute using Reed-Solomon list-decoding:

    $\{q(z) : \deg(q) \leq r \cdot h \cdot t, \Pr_z[C'(L(z)) = q(z)] \geq (\delta' q)/2\}$
  - if agr $\geq (\delta' q)/2$ then $p_m(L(\cdot))$ is in this set!

Decoding

- **Decoding step 2** (continued):
  - assuming $(\delta' q)/2 > (2r \cdot h \cdot t \cdot q)^{1/2}$
  - Reed-Solomon list-decoding step:
    - running time = poly(q)
    - list size $S \leq 4/\delta'$
  - probability list fails to contain $p_m(L(\cdot))$ is $O(1/(\delta q))^{(r-1)/2}$

Decoding

- **Decoding step 2** (continued):
  - Tricky:
    - functions in list are determined by the set $L(\cdot)$, independent of parameterization of the curve
    - Regard $w_2, w_3, \ldots, w_r$ as random points on curve L
    - for $q \neq p_m(L(\cdot))$

$$\Pr[q(w_i) = p_m(L(w_i))] \leq (rht)/q$$

$$\Pr[\forall\, i,\ q(w_i) = p_m(L(w_i))] \leq [(rht)/q]^{r-1}$$

$$\Pr[\exists\ q \text{ in list s.t. } \forall\ i\ q(w_i) = p_m(L(w_i))] \leq (4/\delta')[(rht)/q]^{r-1}$$

Decoding

- **Decoding step 2** (continued):
  - with probability $\geq$ 1 - $O(1/(\delta q))^{(r-1)/2}$ - $(4/\delta)[(rht)/q]^{r-1}$
    - list contains $q^* = p_m(L(\cdot))$
    - q^* is the *unique* q in the list for which

      $q(w_i) = p_m(L(w_i))$ ($=p_m(\alpha_i)$) for i = 2,3,…,r
  - circuit C'':
    - hardwire $w_1, w_2, \ldots, w_r; \alpha_2, \alpha_3, \ldots, \alpha_r$ so that $\forall$ x $\in$ emb(1,2,…,k) both events occur
    - hardwire $p_m(\alpha_i)$ for i = 2,…r
    - on input x, find q^*, output $q^*(w_1)$ (= $p_m(x)$)

Decoding

- Putting it all together:
  - C approximating f' used to construct C'
    - C' makes q queries to C
    - C' runs in time poly(q)
  - C' used to construct C'' computing f exactly
    - C'' makes q queries to C'
    - C'' has r-1 elts of F_q^t and 2r-1 elts of F_q hardwired
    - C'' runs in time poly(q)
  - C'' has size poly(q, r, t, size of C)

Picking parameters

- k truth table size of f, hard for circuits of size s
- **q** field size, **h** R-M degree, **t** R-M dimension
- **r** degree of curve used in decoding

– $h^t \geq k$ (to accomodate message of length k)

– $\delta^6 q^2 > \Omega(rhtq)$ (for R-S list-decoding)

– $k[O(1/(\delta q))^{(r-1)/2} + (4/\delta')[(rht)/q]^{r-1}] < 1$

(so there is a "good" fixing of random bits)

– Pick: $h = s$, $t = (\log k)/(\log s)$

– Pick: $r = \Theta(\log k)$, $q = \Theta(rht\delta^{-6})$

Picking parameters

- k truth table size of f, hard for circuits of size s
- q field size, h R-M degree, t R-M dimension
- r degree of curve used in decoding
- $h = s$, $t = (\log k)/(\log s)$
- $r = \Theta(\log k)$, $q = \Theta(rht\delta^{-6})$

$\log k, \delta^{-1} < s$

Claim: truth table of f' computable in time poly(k)

(so f' $\in$ **E** if f $\in$ **E**).

- poly(q^t) for R-M encoding
- poly(q)·q^t for Hadamard encoding
- $q \leq$ poly(s), so $q^t \leq$ poly(s)^t = poly(h)^t = poly(k)

Picking parameters

- k truth table size of f, hard for circuits of size s
- q field size, h R-M degree, t R-M dimension
- r degree of curve used in decoding
- h = s, t = (log k)/(log s)
- r = $\Theta(\log k)$, q = $\Theta(rht\delta^{-6})$

$\log k, \delta^{-1} < s$

Claim: f' s'-approximable by C implies f computable exactly in size s by C'', for $s' = s^{\Omega(1)}$

- C has size s' and agreement $\delta = 1/s'$ with f'
- C'' has size poly(q, r, t, size of C) = s

Putting it all together

Theorem 1 (IW, STV): If **E** contains functions that require size $2^{\Omega(n)}$ circuits, then **E** contains $2^{\Omega(n)}$-unapproximable functions.

(proof on next slide)

Theorem (NW): if **E** contains $2^{\Omega(n)}$-unapproximable functions then **BPP** = **P**.

Theorem (IW): **E** requires exponential size circuits $\Rightarrow$ **BPP** = **P**.

Putting it all together

- Proof of Theorem 1:
  - let $f = \{f_n\}$ be hard for size $s(n) = 2^{\delta n}$ circuits
  - define $f' = \{f_n'\}$ to be just-described encoding of (the truth tables of) $f = \{f_n\}$
  - two claims we just showed:
    - f' is in **E** since f is.
    - if f ' is $s'(n) = 2^{\delta' n}$-approximable, then f is computable exactly by size $s(n) = 2^{\delta n}$ circuits.
  - contradiction.

Extractors

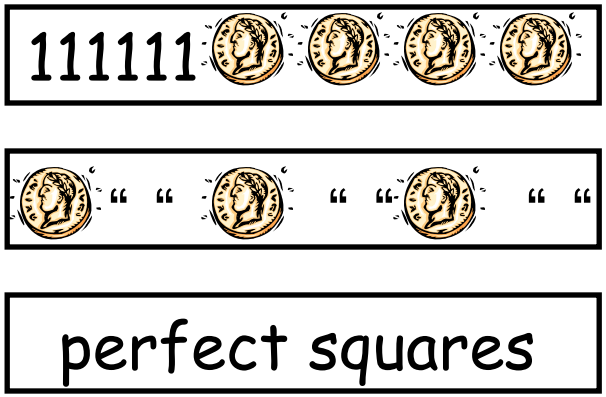
- PRGs: can remove randomness from algorithms
  - based on unproven assumption
  - polynomial slow-down
  - not applicable in other settings
- Question: can we use "real" randomness?
  - physical source
  - imperfect – biased, correlated

Extractors

- "Hardware" side
  - what physical source?
  - ask the physicists…
- "Software" side
  - what is the minimum we need from the physical source?

Extractors

- imperfect sources:
  - "stuck bits":
  - "correlation":
  - "more insidious correlation":



- there are specific ways to get independent unbiased random bits from specific imperfect physical sources

Extractors

- want to assume we don't know details of physical source

- **general model** capturing all of these?
  - yes: "min-entropy"
- **universal procedure** for all imperfect sources?
  - yes: "extractors"

Min-entropy

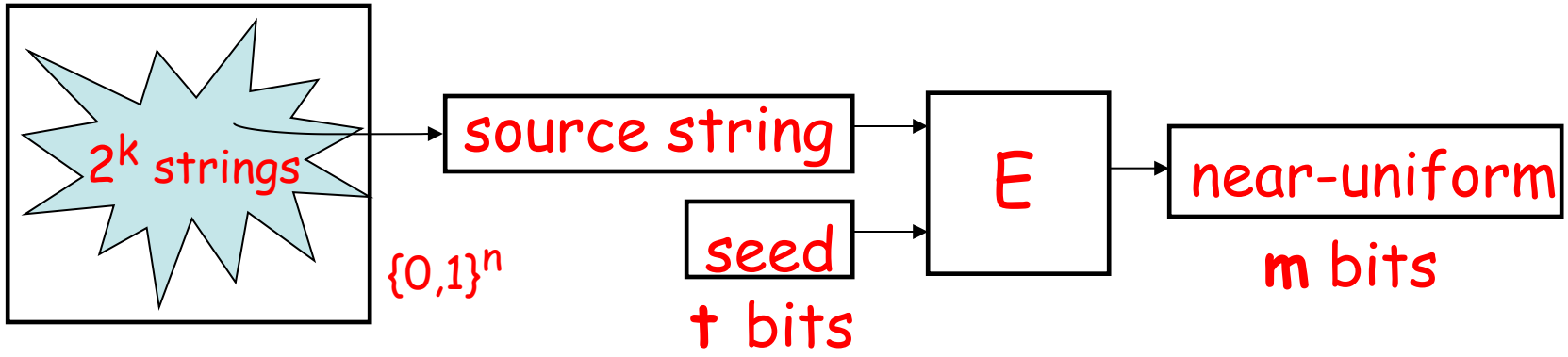
- General model of physical source w/ k < n bits of hidden randomness

Definition: random variable X on $\{0,1\}^n$ has **min-entropy** $\min_x -\log(\Pr[X = x])$

– min-entropy k implies no string has weight more than 2^{-k}

Extractor

- Extractor: universal procedure for "purifying" imperfect source:



  - E is efficiently computable
  - truly random seed as "catalyst"

Extractor

"(k, ε)-extractor" ⇒ for all X with min-entropy k:

– output fools **all** circuits C:

$$|Pr_z[C(z) = 1] - Pr_{y,\, x \leftarrow X}[C(E(x, y)) = 1]| \leq \boldsymbol{\varepsilon}$$

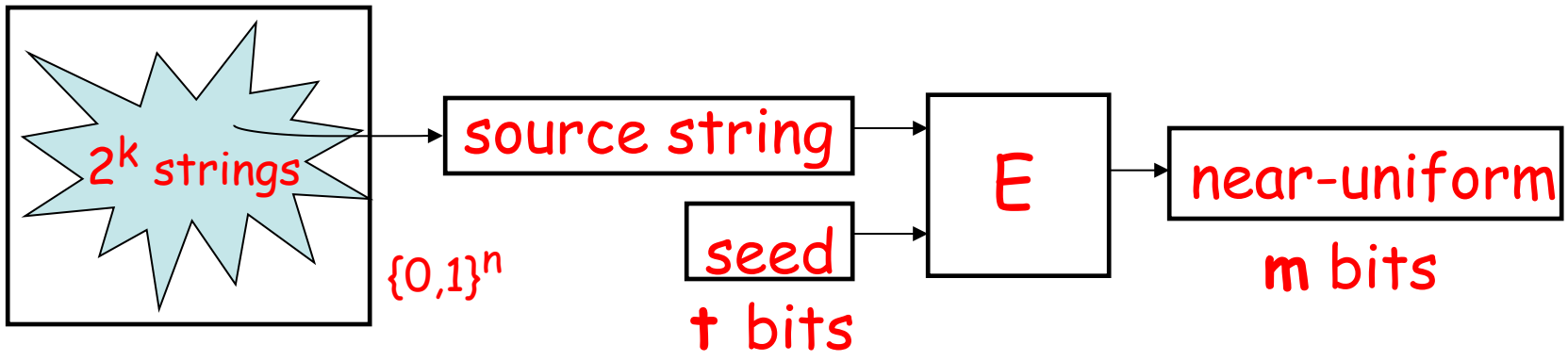
– distributions $E(X, U_t)$, U_m "**ε**-close" (L_1 dist ≤ 2**ε**)

- Notice similarity to PRGs
  - output of PRG fools all **efficient** tests
  - output of extractor fools **all** tests

Extractors

- Using extractors
  - use output in place of randomness in any application
  - alters probability of **any** outcome by at most ε
- Main motivating application:
  - use output in place of randomness in algorithm
  - how to get truly random seed?
  - enumerate all seeds, take majority

Extractors



- Goals:

	good:	best:
short seed	$O(\log n)$	$\log n+O(1)$
long output	$m = k^{\Omega(1)}$	$m = k+t-O(1)$
many k's	$k = n^{\Omega(1)}$	any $k = k(n)$

Extractors

- random function for E achieves best !
  - but we need **explicit** constructions
  - many known; often complex + technical
  - optimal extractors still open
- Trevisan Extractor:
  - insight: use NW generator with source string in place of hard function
  - this works (!!)
  - proof slightly different than NW, easier