From Learning Algorithms to Differentially Private Algorithms

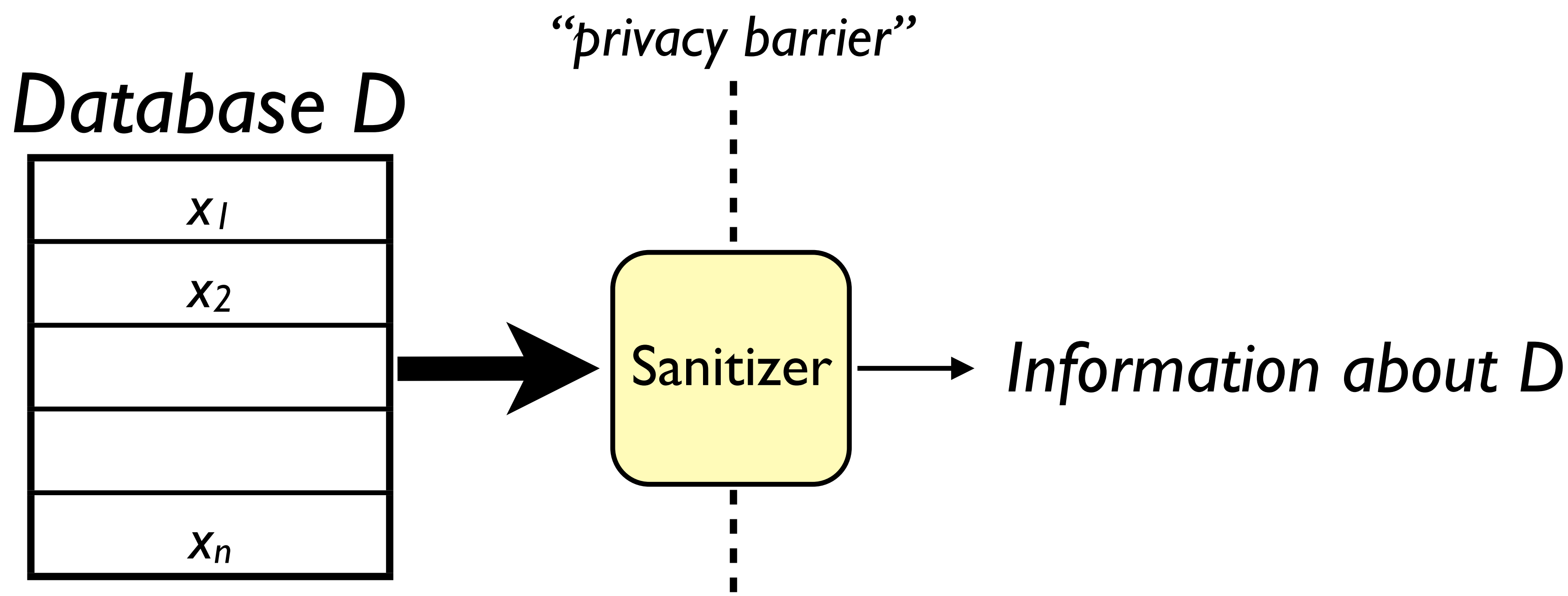
Jonathan Ullman, Harvard University

Big Data and Differential Privacy Workshop
December 12, 2013

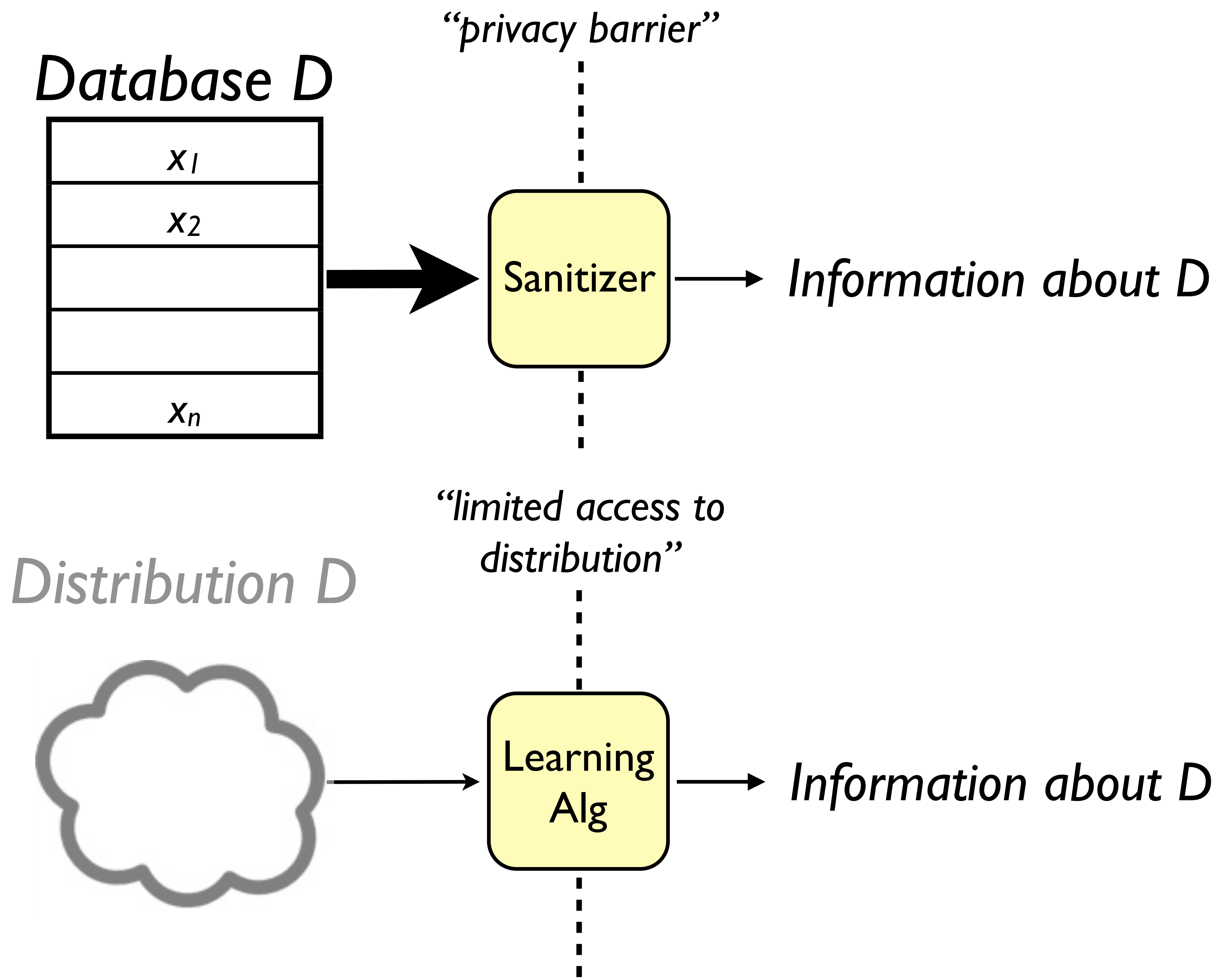
What is this tutorial about?

- Using powerful techniques from learning theory to design differentially private algorithms

Why would we want to do that?



Why would we want to do that?



Why would we want to do that?

- Connections between learning and DP algorithm design first(?) introduced in [BDMN,KLNRS]

Why would we want to do that?

- Clean, qualitatively strong guarantees brought out the potential of differentially private data analysis
- For these strong guarantees, learning-theoretic techniques yield nearly-optimal algorithms

Private Counting Query Release

Counting query: *What fraction of records satisfy property q?*

$D \in (\{0,1\}^d)^n$

GiveYouUp?	LetYouDown?	RunAround?	DesertYou?
0	0	1	1
1	1	1	1
1	0	0	0
1	1	0	0

d attributes per record

Private Counting Query Release

Counting query: *What fraction of records satisfy property q? e.g.*
q(x) = GiveYouUp ∨ LetYouDown?

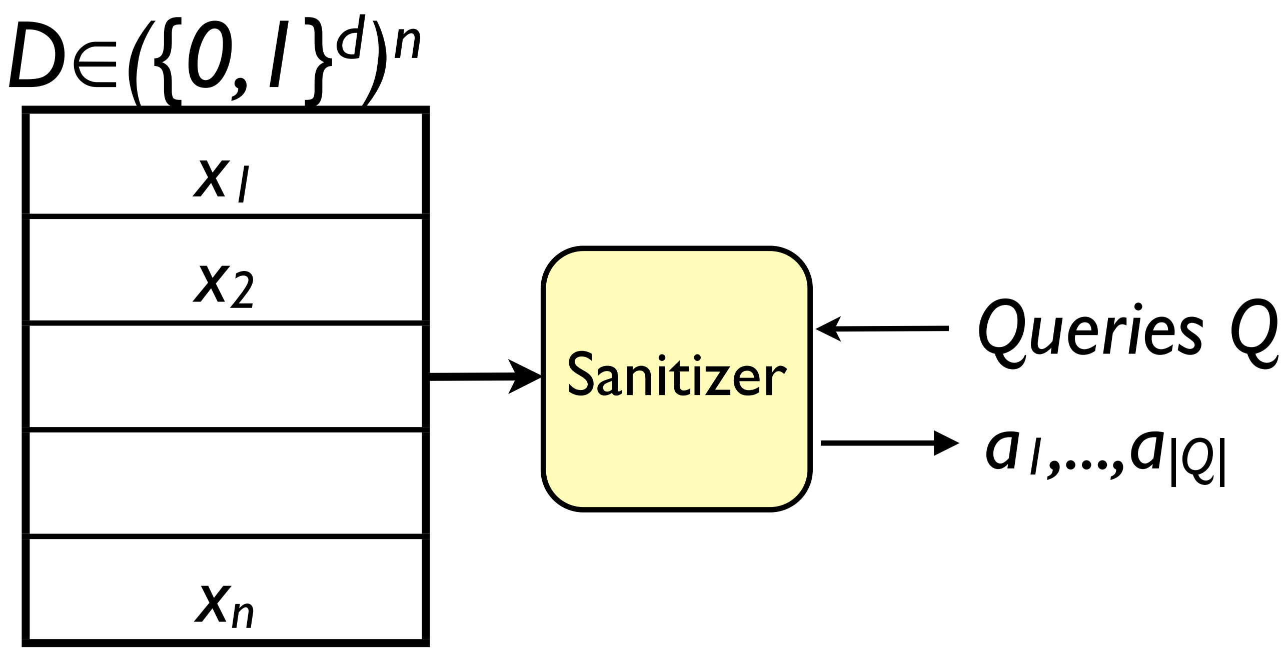
$D \in (\{0,1\}^d)^n$

GiveYouUp?	LetYouDown?	RunAround?	DesertYou?	
0	0	1	1	$q(x_1) = 0$
1	1	1	1	$q(x_1) = 1$
1	0	0	0	$q(x_1) = 1$
1	1	0	0	$q(x_1) = 1$

d attributes per record

$q(D) = 3/4$

Private Counting Query Release

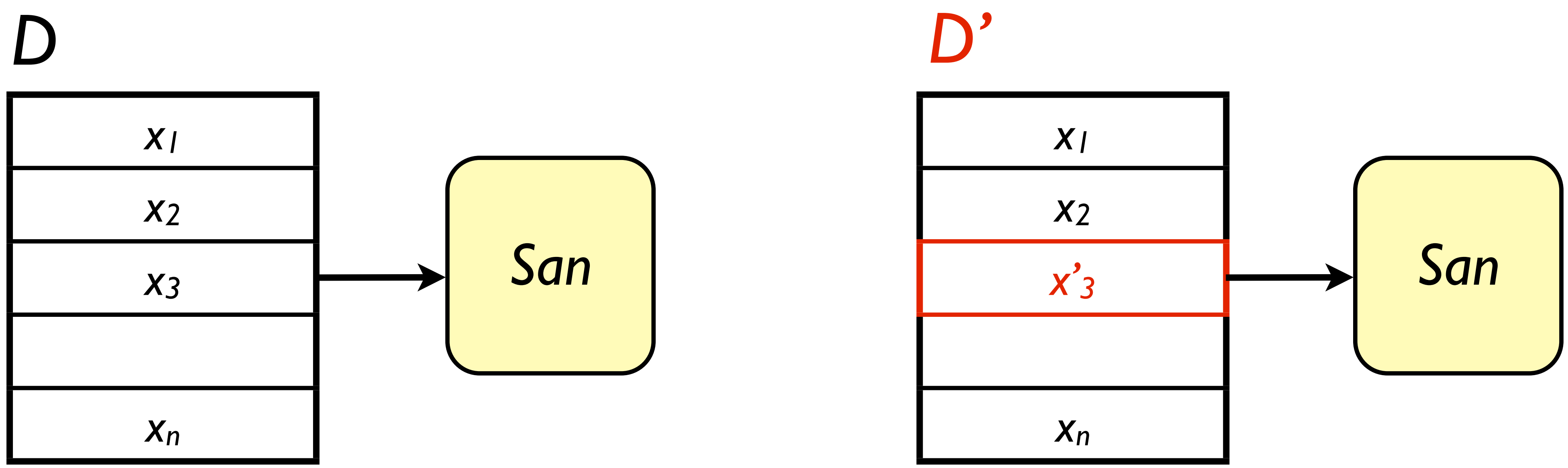


Accurate if
$|a_q - q(D)| \leq \alpha$
for every $q \in Q$

- Want to design a sanitizer that is simultaneously differentially private and accurate

Differential Privacy

[DN,DN,BDMN,DMNS,D]

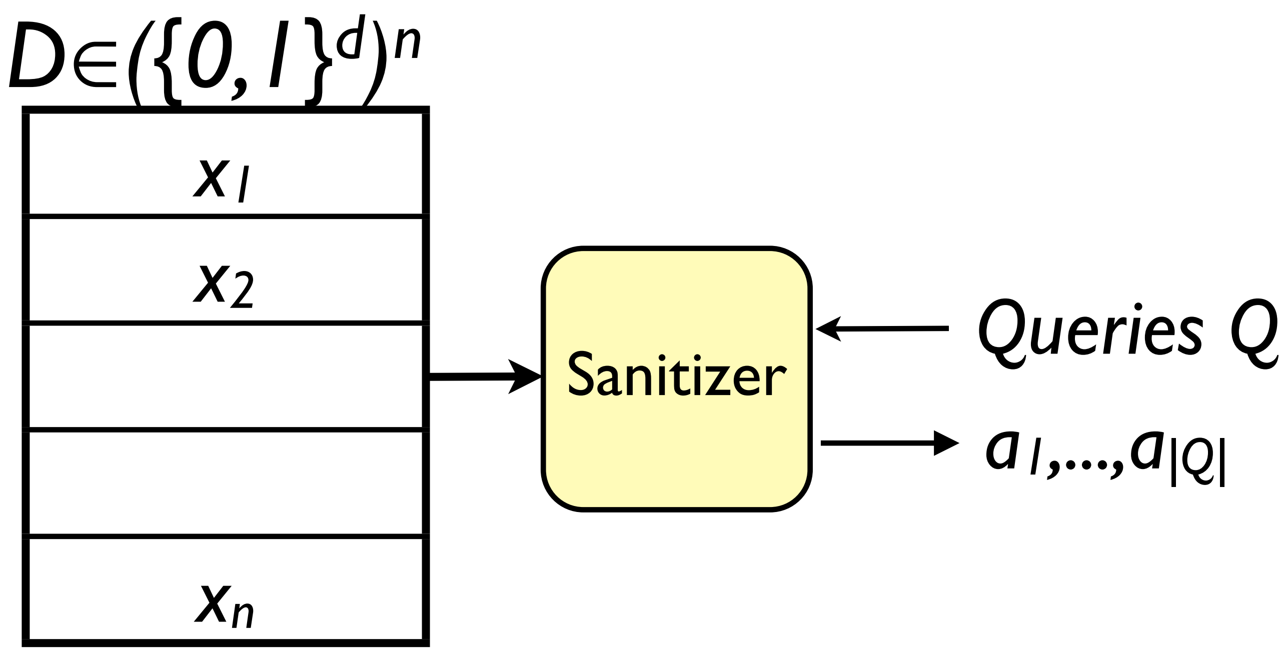


D and *D'* are neighbors if they differ only on one user's data

Definition: A (randomized) *San* is (ε,δ)-differentially private if for all neighbors *D*, *D'* and every $S \subseteq Range(San)$

$$Pr[San(D) \in S] \leq e^{\varepsilon} Pr[San(D') \in S] + \delta$$

Private Counting Query Release



Accurate if
$|a_q - q(D)| \leq \alpha$
for every $q \in Q$

- Want to design a sanitizer that is simultaneously differentially private and accurate
- Want to minimize
  - Amount of data required, n for a given Q,d,α
  - Running time of the sanitizer

Private Query Release: An Abridged History

- Adding independent noise (Laplace mechanism) requires $n \gtrsim |Q|^{1/2}/\alpha$

Private Query Release: An Abridged History

- Adding independent noise (Laplace mechanism) requires $n \gtrsim |Q|^{1/2}/\alpha$
- [BLR] gave a sanitizer that requires only $n \gtrsim d \log|Q|/\alpha^3$
  - Several important improvements by [DNRRV,DRV,RR]

Private Query Release: An Abridged History

- [HR] introduced the private multiplicative weights algorithm, requires only $n \gtrsim d^{1/2} \log|Q|/\alpha^2$

Private Query Release: An Abridged History

- [HR] introduced the private multiplicative weights algorithm, requires only $n \gtrsim d^{1/2} log|Q|/\alpha^2$
- Put in a general framework, with tight analysis by [GHRU,GRU,HLM]
- Several improvements for special cases of private query release followed [GRU,JT,BR,HR,HRS,TUV,CTUW,...]

Talk Outline

- Differentially private query release
- A blueprint for private query release
  - No-regret algorithms / MW
- Query Release Algorithms
  - Offline MW
  - Online MW
  - Variants
  - Faster algorithms for disjunctions via polynomial approx.

A Blueprint for Query Release

Sanitized (DP) Output

Raw Data

D

A Blueprint for Query Release

Sanitized (DP) Output

Raw Data

D_1

D

A Blueprint for Query Release

Sanitized (DP) Output

Raw Data

D_1

D

Is D_1
good for *Q?*

A Blueprint for Query Release

Sanitized (DP) Output

Raw Data

D_1 *(hint₁)*

D

Is D_1 *good for Q?*

Here's hint $_1$

A Blueprint for Query Release

Sanitized (DP) Output

D_1 ($query_1$)

Raw Data

D

Is D_1 *good for* Q*?* →

← *Here's* $query_1$

$query_1$ is a query D_1 answers incorrectly

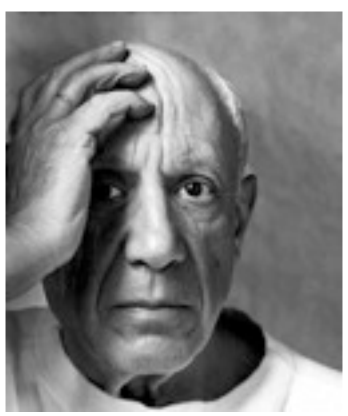
A Blueprint for Query Release

Sanitized (DP) Output

D_1 *(query$_1$)*

Raw Data

D



Update Alg: *U*

A Blueprint for Query Release

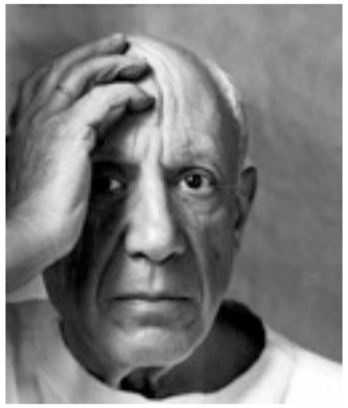
Sanitized (DP) Output

Raw Data

D_1 *(query$_1$)*

D_2

D



Update Alg: *U*

A Blueprint for Query Release

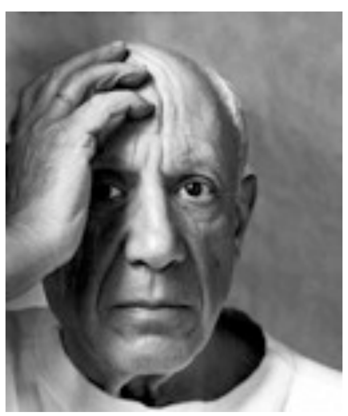
Sanitized (DP) Output

Raw Data

D_1 *(query$_1$)*

D_2  *(query$_2$)*

D



Update Alg: *U*

Is D_2
good for *Q?*

Here's query$_2$

query$_2$ is a query D_2 answers incorrectly

A Blueprint for Query Release

Sanitized (DP) Output

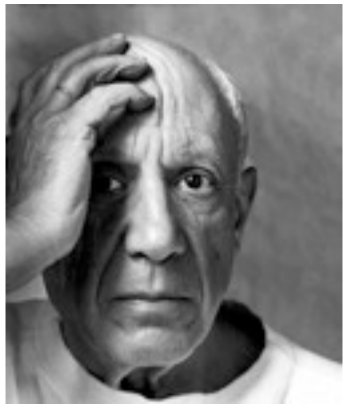
Raw Data

D_1 *(query₁)*

D_2  *(query₂)*

D_3 

D



Update Alg: *U*

A Blueprint for Query Release

Sanitized (DP) Output | Raw Data

D_1 *(query$_1$)*

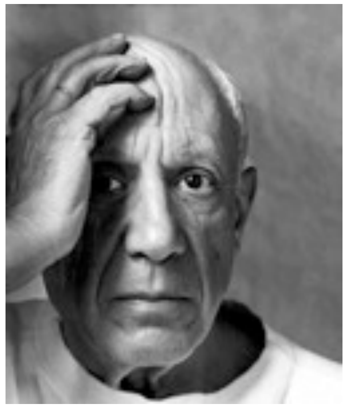
D_2 *(query$_2$)*

D_3 *(query$_3$)*

D_4 *(query$_4$)*

⋮ ⋮

D_T *(OK)*



Update Alg: *U*

Is D_T good for Q?

D

→

←

Yes, approximately!

A Blueprint for Query Release

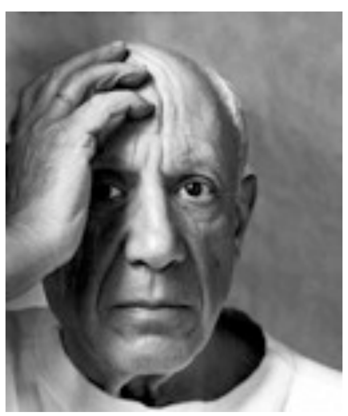
Sanitized (DP) Output

Raw Data

D

D_1 *(query$_1$)*

D_2  *(query$_2$)*



Update Alg: U

D_3  *(query$_3$)*

Is D_T good for Q? →

← *Yes, approximately!*

D_4  *(query$_4$)*

⋮

D_T  *(OK)*

A Blueprint for Query Release

LET D *be the real database*

LET D_1 *be an "initial guess"*

FOR $t = 1,\ldots,T$

  LET $query_t = argmax_{q \in Q}\ q(D_t) - q(D)$

  LET $D_{t+1} = Update(D_t, q_t)$

Why did we do this?

- Decomposed the problem into smaller problems
  - Fortunately, DP has nice composition properties
- We've separated privacy (finding q_t) from the task of learning the database (updating D_t)
  - Means we can choose any update algorithm

A Blueprint for Query Release

Sanitized (DP) Output

Raw Data

D

D_1 *(query$_1$)*

D_2  *(query$_2$)*

D_3  *(query$_3$)*

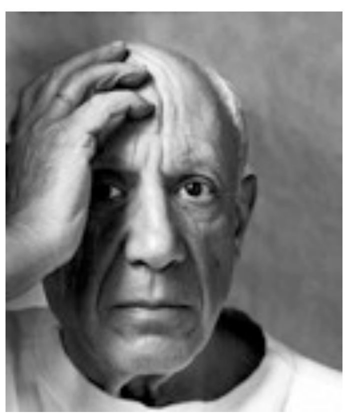
D_4  *(query$_4$)*

⋮ ⋮

$D_{|Q|}$  *(OK)*

Is $D_{|Q|}$ good for Q? →

← *Yes, approximately!*



Update Alg: U

Why did we do this?

- (Hopefully) decomposed the problem into $T \ll |Q|$ smaller problems
  - Fortunately, DP has nice composition properties
- We've separated privacy (finding q_t) from the task of learning the database (updating D_t)
  - Means we can choose any update algorithm

Talk Outline

- Differentially private query release
- A blueprint for private query release
  - No-regret algorithms / MW
- Query Release Algorithms
  - Offline MW
  - Online MW
  - Variants
  - Faster algorithms for disjunctions via polynomial approx.

No-Regret Learning Algorithms

Set of experts X

Losses for each expert

No-Regret Learning Algorithms

Set of experts *X*

Distribution over *X*

D_1 *.25*   *.25*   *.25*   *.25*

Losses for each expert

No-Regret Learning Algorithms

Set of experts X

Distribution over X

D_1 .25 .25 .25 .25

Loss is $\langle D_1, L_1 \rangle$

Losses for each expert

$[0,1]^X$

1 0 1 0 L_1

No-Regret Learning Algorithms

Set of experts X

Distribution over X

D_1	.25	.25	.25	.25
D_2	.20	.30	.20	.30

Losses for each expert

$[0,1]^X$

Loss is $\langle D_1, L_1 \rangle$ 1   0   1   0 L_1

Multiplicative Weights Update [LW]

$D_2 = MWU(D_1, L_1)$:

$$D_2'(x) = (1 - \eta L_1(x)) D_1(x)$$

$$D_2(x) = D_2'(x) / \sum_{x \in X} D_2'(x)$$

No-Regret Learning Algorithms

Set of experts X

Distribution over X

Losses for each expert

$[0,1]^X$

D_1	.25	.25	.25	.25	Loss is $<D_1,L_1>$	1	0	1	0	L_1
D_2	.20	.30	.20	.30	Loss is $<D_2,L_2>$	0	0	1	0	L_2

No-Regret Learning Algorithms

Set of experts X

Losses for each expert

Distribution over X

$[0,1]^X$

D_1	.25	.25	.25	.25	Loss is $\langle D_1,L_1 \rangle$	1	0	1	0	L_1
D_2	.20	.30	.20	.30	Loss is $\langle D_2,L_2 \rangle$	0	0	1	0	L_2
					⋮					
D_T	.23	.32	.15	.32	Loss is $\langle D_T,L_T \rangle$	0	0	0	1	L_T

No-Regret Learning Algorithms

Set of experts X										Losses for each expert

	Distribution over X					$[0,1]^X$				
D_1	.25	.25	.25	.25	Loss is $<D_1,L_1>$	1	0	1	0	L_1
D_2	.20	.30	.20	.30	Loss is $<D_2,L_2>$	0	0	1	0	L_2
					⋮					
D_T	.23	.32	.15	.32	Loss is $<D_T,L_T>$	0	0	0	1	L_T

For any distribution D, sequence $L_1, \ldots, L_T$,

$$\sum_{t=1}^{T} \langle D_t - D, L_t \rangle \leq \sqrt{T \log |X|}$$

Counting Queries

Counting query: *What fraction of records satisfy property q? e.g.*
q(x) = GiveYouUp ∨ LetYouDown

$D \in (\{0,1\}^d)^n$

GiveYouUp?	LetYouDown?	RunAround?	DesertYou?	
0	0	1	1	$q(x_1)=0$
1	1	1	1	$q(x_2)=1$
1	0	0	0	$q(x_3)=1$
1	1	0	0	$q(x_4)=1$

d attributes per record

$q(D)=3/4$

Counting Queries

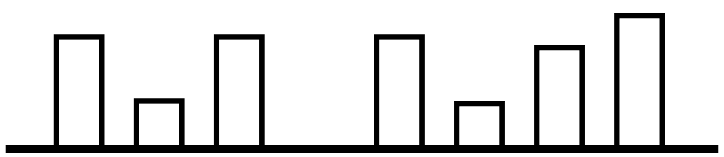
Counting query: *What fraction of records satisfy property q? e.g.*
$q(x)$ = GiveYouUp ∨ LetYouDown

$D \in (\{0,1\}^d)^n$

GiveYouUp?	LetYouDown?	RunAround?	DesertYou?	
0	0	1	1	$q(x_1)=0$
1	1	1	1	$q(x_2)=1$
1	0	0	0	$q(x_3)=1$
1	1	0	0	$q(x_4)=1$

d attributes per record

$q(D)=3/4$



D is a distribution on $\{0,1\}^d$

Counting Queries

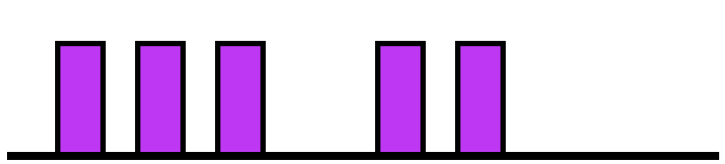
Counting query: *What fraction of records satisfy property q? e.g.*
$q(x) = GiveYouUp \vee LetYouDown$

$D \in (\{0,1\}^d)^n$

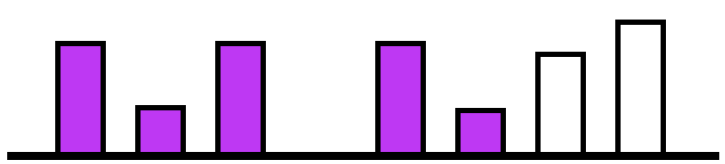
GiveYouUp?	LetYouDown?	RunAround?	DesertYou?	
0	0	1	1	$q(x_1)=0$
1	1	1	1	$q(x_2)=1$
1	0	0	0	$q(x_3)=1$
1	1	0	0	$q(x_4)=1$

d attributes per record

$q(D)=3/4$



q is an indicator vector



D is a distribution on $\{0,1\}^d$

Linear query: $q(D) = \langle D, q \rangle$

No-Regret Learning Algorithms

Set of experts X		Losses for each expert
		
Distribution over X		$[0,1]^X$

D_1	.25	.25	.25	.25	Loss is $<D_1,L_1>$	1	0	1	0	L_1
D_2	.20	.30	.20	.30	Loss is $<D_2,L_2>$	0	0	1	0	L_2
					⋮					
D_T	.23	.32	.15	.32	Loss is $<D_T,L_T>$	0	0	0	1	L_T

For any distribution D, sequence $L_1, \ldots, L_T$,

$$\sum_{t=1}^{T} \langle D_t - D, L_t \rangle \leq \sqrt{T \log |X|}$$

Multiplicative Weights for Query Release

Set of experts $X=\{0,1\}^d$ | Losses for each expert

Distribution over $X=\{0,1\}^d$ | Truth table of q in $[0,1]^X$

D_1	.25	.25	.25	.25	Loss is <D_1,q_1>	1	0	1	0	q_1
D_2	.20	.30	.20	.30	Loss is <D_2,q_2>	0	0	1	0	q_2
					⋮					
D_T	.23	.32	.15	.32	Loss is <D_T,q_T>	0	0	0	1	q_T

For any database D, sequence $q_1, \ldots, q_T$,

$$\sum_{t=1}^{T} \langle D_t - D, q_t \rangle \leq \sqrt{Td}$$

A Blueprint for Query Release

LET D *be the real database, viewed as a dist over* $\{0,1\}^d$
LET D_1 *be the uniform dist on* $\{0,1\}^d$
FOR $t = 1,\ldots,T$
    LET $q_t = \text{argmax}_{q\in Q} \langle D_t - D, q\rangle$
    LET $D_{t+1} = \text{MWU}(D_t, q_t)$

$$D'_{t+1}(x) = (1 - \eta q_t(x))D_t(x)$$

$$D_{t+1}(x) = \frac{D'_{t+1}(x)}{\sum_{x\in\{0,1\}^d} D'_{t+1}(x)}$$

Query Release via MW

- Thm: For any database *D* sequence $q_1,\ldots,q_T$,

$$\sqrt{Td} \geq \sum_{t=1}^{T} \langle D_t - D, q_t \rangle$$

Query Release via MW

- Thm: For any database *D* sequence $q_1,\ldots,q_T$,

$$\sqrt{Td} \geq \sum_{t=1}^{T} \langle D_t - D, q_t \rangle$$

- If $q_1,\ldots,q_T$ all satisfy $\langle D_t - D, q_t \rangle \geq \alpha$, then we have

$$\sqrt{Td} \geq \sum_{t=1}^{T} \langle D_t - D, q_t \rangle \geq \alpha T$$

Query Release via MW

- Thm: For any database *D* sequence $q_1,\ldots,q_T$,

$$\sqrt{Td} \geq \sum_{t=1}^{T} \langle D_t - D, q_t \rangle$$

- If $q_1,\ldots,q_T$ all satisfy $\langle D_t - D, q_t \rangle \geq \alpha$, then we have

$$\sqrt{Td} \geq \sum_{t=1}^{T} \langle D_t - D, q_t \rangle \geq \alpha T$$

- If $T \gtrsim d/\alpha^2$, then $\langle D_T - D, q \rangle \leq \alpha$ for all of Q

Query Release via MW

- Thm: For any database D sequence $q_1,...,q_T$,

$$\sqrt{Td} \geq \sum_{t=1}^{T} \langle D_t - D, q_t \rangle$$

- If $q_1,...,q_T$ all satisfy $<D_t - D, q_t> \geq \alpha$, then we have

$$\sqrt{Td} \geq \sum_{t=1}^{T} \langle D_t - D, q_t \rangle \geq \alpha T$$

- If $T \gtrsim d/\alpha^2$, then $|<D_T - D, q>| \leq \alpha$ for all of Q

Q is closed under neg.

A Blueprint for Query Release

LET D *be the real database, viewed as a dist over* $\{0,1\}^d$

LET D_1 *be the uniform dist on* $\{0,1\}^d$

FOR $t = 1,\ldots,T=O(d/\alpha^2)$

LET $q_t = \mathrm{argmax}_{q \in Q} \langle D_t - D, q \rangle$

LET $D_{t+1} = MWU(D_t, q_t)$

A Blueprint for Query Release

LET D be the real database, viewed as

LET D_1 be the uniform dist on $\{0,1\}^d$

FOR $t = 1,\ldots,T=O(d/\alpha^2)$

LET $q_t = \mathrm{argmax}_{q\in Q} <D_t - D, q>$

LET $D_{t+1} = MWU(D_t, q_t)$

Have to make this DP

Finding the "Bad" Queries

- How do I find $argmax_{q \text{ in } Q} \langle D_t - D, q \rangle$ privately? Use the exponential mechanism!
- Output q wp proportional to $exp(\varepsilon_0 n \langle D_t - D, q \rangle)$

If $n \gtrsim log|Q|/\alpha\varepsilon_0$ then whp EM outputs q_t s.t.
$\langle D_t - D, q_t \rangle \geq max_{q \in Q} \langle D_t - D, q \rangle - \alpha/2$

A Blueprint for Query Release

LET D be the real database, viewed as a dist on $\{0,1\}^d$

LET D_1 *be the uniform distribution on* $\{0,1\}^d$

FOR $t = 1,\ldots,T = O(d/\alpha^2)$

LET q_t = *q wp proportional to* $\exp(\varepsilon_0 n\langle D_t - D, q\rangle)$

LET $D_{t+1} = MWU(D_t, q_t)$

A Blueprint for Query Release

Need $n \geq \log|Q|/\alpha\varepsilon_0$

LET D be the real database, viewed as a dist on $\{0,1\}^d$

LET D_1 *be the uniform distribution on* $\{0,1\}^d$

FOR $t = 1,\ldots,T = O(d/\alpha^2)$

LET $q_t = q$ *wp proportional to* $\exp(\varepsilon_0 n<D_t - D, q>)$

LET $D_{t+1} = MWU(D_t, q_t)$

A Blueprint for Query Release

Need $n \geq \log|Q|/\alpha\varepsilon_0$

LET D *be the real database, viewed as a dist on* $\{0,1\}^d$

LET D_1 *be the* uniform distribution *on* $\{0,1\}^d$

FOR $t = 1,\ldots,T = O(d/\alpha^2)$

  LET $q_t = q$ *wp proportional to* $\exp(\varepsilon_0 n\langle D_t - D, q\rangle)$

  LET D_{t+1} = *MWU*(D_t, q_t)

Thm [DRV]: If $\varepsilon_0 \leq \varepsilon/(8T\log(1/\delta))^{1/2} \approx \varepsilon/T^{1/2}$, then running T (adaptively chosen) ε_0-DP algorithms satisfies (ε,δ)-DP.

A Blueprint for Query Release

Need
$n \gtrsim d^{1/2} \log|Q| / \alpha^2 \varepsilon$

LET D be the real database, viewed as a dist on $\{0,1\}^d$
LET D_1 *be the uniform distribution on* $\{0,1\}^d$
FOR $t = 1,\ldots,T = O(d/\alpha^2)$
  LET $q_t = q$ *wp proportional to* $\exp(\varepsilon \alpha n \langle D_t - D, q \rangle / d^{1/2})$
  LET $D_{t+1} = MWU(D_t, q_t)$

Thm [DRV]: If $\varepsilon_0 \approx \varepsilon / T^{1/2} \approx \varepsilon\alpha / d^{1/2}$, then running T (adaptively chosen) ε_0-DP algorithms satisfies (ε,δ)-DP.

Recap

Thm: PMW takes a database $D \in (\{0,1\}^d)^n$ and a set of counting queries Q, satisfies (ε,δ)-DP and, if

$$n \gtrsim d^{1/2} \log|Q|/\alpha^2\varepsilon,$$

it outputs D_T such that for every $q \in Q$,

$$|q(D) - q(D_T)| \leq \alpha$$

Optimality?

- PMW achieves a nearly-optimal data requirement for this level of generality
  - Thm [BUV]: for every sufficiently large *s*, there is a family of *s* queries *Q* such that any (ε,δ)-DP algorithm that is α-accurate for *Q* requires $n \gtrsim d^{1/2} \log|Q| / \alpha^2 \varepsilon$

Recap

Thm: PMW takes a database $D \in (\{0,1\}^d)^n$ and a set of counting queries Q, satisfies (ε,δ)-DP and, if
$n \gtrsim O(d^{1/2} log|Q|/\alpha^2 \varepsilon)$,
it outputs D_T such that for every $q \in Q$,
$|q(D) - q(D_T)| \leq \alpha$

Thm: PMW runs in time $poly(n, 2^d, |q_1| + ... + |q_{|Q|}|)$

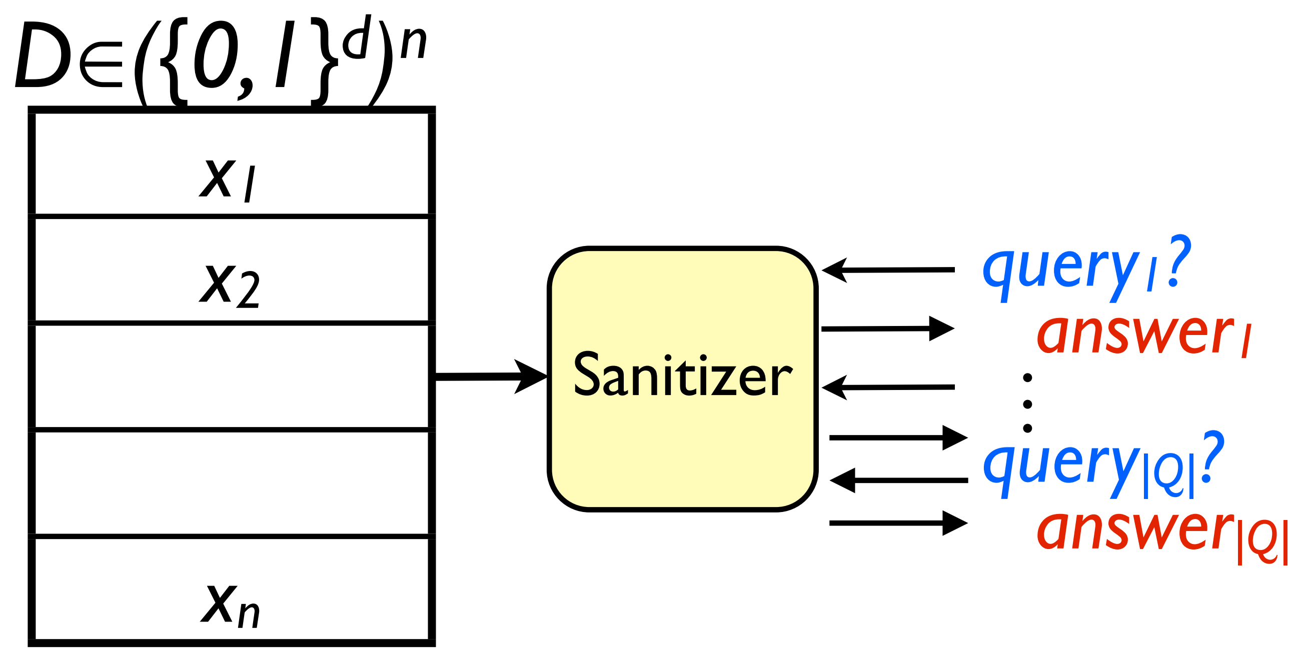
Optimality?

- Private multiplicative weights achieves nearly-optimal running time for this level of generality
  - Thm [U]: any DP algorithm that takes a database $D \in (\{0,1\}^d)^n$ and a set of counting queries Q, runs in time $poly(n,d,|q_1|+...+|q_{|Q|}|)$, and accurately answers Q requires $n \gtrsim |Q|^{1/2}$ (assuming secure crypto exists)
- But PMW can be practical! [HLM]

Talk Outline

- Differentially private query release
- A blueprint for private query release
  - No-regret algorithms / MW
- Query Release Algorithms
  - Offline MW
  - Online MW
  - Variants
  - Faster algorithms for disjunctions via polynomial approx.

Online Counting Query Release



Accurate if
$|a_q - q(D)| \leq \alpha$
for every $q \in Q$

- Want to design an online sanitizer that is simultaneously differentially private and accurate
- Want to minimize
  - Amount of data required, n as a function of $|Q|, d, \alpha$
  - Running time of the sanitizer per query

A Blueprint for Query Release

Sanitized (DP) Output

Raw Data

Family of queries *Q*? →

D_1 *(query₁)*

D_2 *(query₂)*

D_3 *(query₃)*

D_4 *(query₄)*

⋮

D_T *(OK)*

Is D_1 good for *Q?* →

D

← *Here's query₁*

query₁ is a query D_1 answers incorrectly

A Blueprint for Online Query Release

Sanitized (DP) Output

Raw Data

Query q_1? → D_1

D

Is D_1 good for q_1? →

← Yes

A Blueprint for Online Query Release

Sanitized (DP) Output

Raw Data

Query q_1? →

$q_1(D_1)$ ←

D_1

(OK)

D

Is D_1 good for q_1? →

← Yes

A Blueprint for Online Query Release

Sanitized (DP) Output

Raw Data

Query q_1? →
$q_1(D_1)$ ←

D_1

(OK)

Query q_2? →
$q_2(D)$ ←

D_1

$(q_2(D))$

D

Is D_1 good for q_2? →

← No, $q_2(D_2)$ should be $q_2(D)$

A Blueprint for Online Query Release

Sanitized (DP) Output

Raw Data

Query q_1? →

$q_1(D_1)$ ←

D_1 (OK)

Query q_2? →

$q_2(D)$ ←

D_1 $(q_2(D))$

D_2

Is D_1 good for q_2? →

← No, $q_2(D_2)$ is too low

D

A Blueprint for Online Query Release

Sanitized (DP) Output | Raw Data

Query q_1? → ← $q_1(D_1)$ | D_1 | (OK)

Query q_2? → ← $q_2(D)$ | D_1 | $(q_2(D))$

Query q_3? → ← $q_3(D_2)$ | D_2  | (OK)

Query q_4? → ← $q_4(D)$ | D_3  | $(q_4(D))$

⋮ ⋮ ⋮

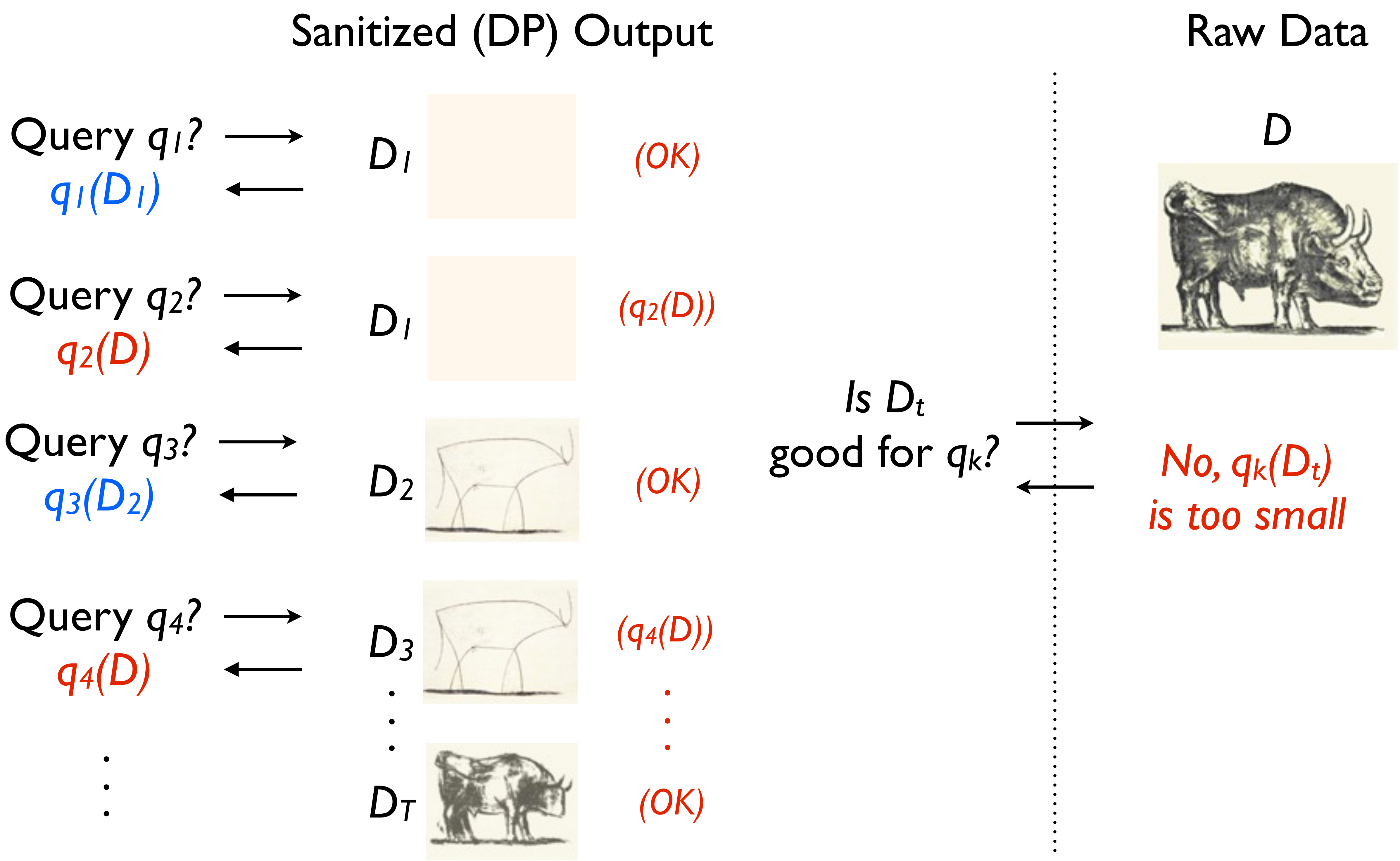
D_T  | (OK)

Is D_t good for q_k? →

← No, $q_k(D_t)$ is too small

D

A Blueprint for Online Query Release



A Blueprint for Query Release

LET D be the real database, viewed as a dist over $\{0,1\}^d$

LET D_1 *be the uniform dist on* $\{0,1\}^d$

FOR $k = 1,\ldots,|Q|$

    IF $|\langle D_t - D, q_k\rangle| \leq \alpha$ *THEN answer* $\langle D_t, q_k\rangle$

    ELSE

        answer $\langle D, q_k\rangle$, $D_{t+1} = MWU(D_t, q_k)$

        LET $t=t+1$

$T \leq d/\alpha^2$

“Threshold” Algorithm

- Suppose we have a stream of queries $q_1,...,q_k$ and promise that there is only a single q_i s.t. $q_i(D) \geq \alpha/2$
- Then there is an ε_0-DP algorithm that whp answers every query with accuracy α as long as $n \gtrsim \log(k)/\alpha\varepsilon_0$

A Blueprint for Online Query Release

Query q_1? → D_1 *(OK)*

$q_1(D_1)$ ←

Query q_2? → D_1 *($q_2(D)$)*

$q_2(D)$ ←

Instance of threshold algorithm

Query q_3? → D_2 *(OK)*

$q_3(D_2)$ ←

Query q_4? → D_3 *($q_4(D)$)*

$q_4(D)$ ←

Instance of threshold algorithm

⋮

D_T *(OK)*

Recap

Thm: Online PMW takes a database $D \in (\{0,1\}^d)^n$ and an online stream of counting queries Q, satisfies (ε,δ)-DP and, if

$n \gtrsim d^{1/2} \log|Q|/\alpha^2\varepsilon$,

is α-accurate for all of Q

Thm: Runs in time $poly(n,2^d,|q|)$ for each query q

Talk Outline

- Differentially private query release
- A blueprint for private query release
  - No-regret algorithms / MW
- Query Release Algorithms
  - Offline MW
  - Online MW
  - Variants
  - Faster algorithms for disjunctions via polynomial approx.

Other Applications

- PMW has optimal data requirement and running time in the worst case, but better algorithms are known for special cases
- Modular design makes it easy to construct new algorithms by swapping in different no-regret algorithms

Graph Cuts

[GRU]

- G *in* $(V \times V)^{|E|}$. *Cut query* $q_{S,T}(G)$ asks "What fraction of edges cross from S to T?"
  - Counting queries on a database D *in* $(\{0,1\}^{2\log|V|})^{|E|}$
- Can reduce the data requirement for some settings of parameters by replacing MW with an algorithm based on the "cut-decomposition" [FK]

Mirror Descent

[JT]

- Replace MW with algorithms from the mirror descent family
  - Reduces the data requirement when the L_p norm of the database and L_q norm of the queries satisfy certain relationships
    - For PMW, we view the database as a distribution over $X=\{0,1\}^d$ (L_1 norm = 1), we view the query as a vector in $[0,1]^X$ (L_∞ norm = 1)
  - Applications to cut queries, matrix queries

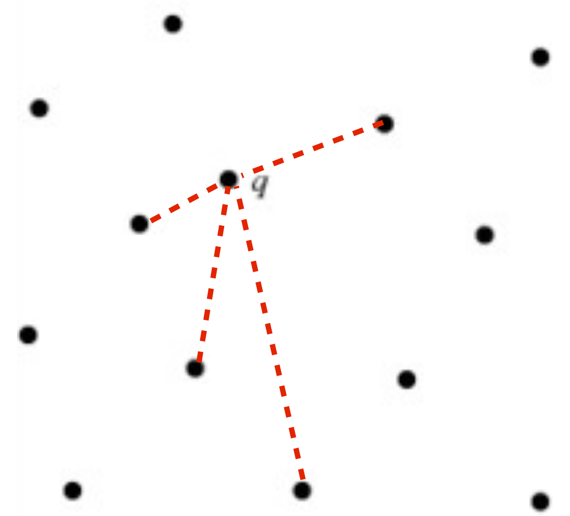
Sparse Queries

[BR]

- Query is sparse if it only accepts $S \ll 2^d$ elements from $\{0,1\}^d$
- Can design an "implicit" implementation of MW that keeps track of $\sim S$ weights instead of 2^d
  - Improves running time per query from 2^d to $\sim S$
  - Also improves the data requirement slightly

Distance Queries

[HR]



- D *in* $([0,1]^d)^n$. Query q_x is a point x *in* $[0,1]^d$ and asks "What is the average distance between points in D and x?"
- Can answer in time *poly(n,d)* per query using a specialized no-regret algorithm for distance queries
  - Improves data requirement in some cases too

Talk Outline

- Differentially private query release
- A blueprint for private query release
  - No-regret algorithms / MW
- Query Release Algorithms
  - Offline MW
  - Online MW
  - Variants
  - Faster algorithms for disjunctions via polynomial approx.

Private Counting Query Release

Counting query: *What fraction of records satisfy property q? e.g.*

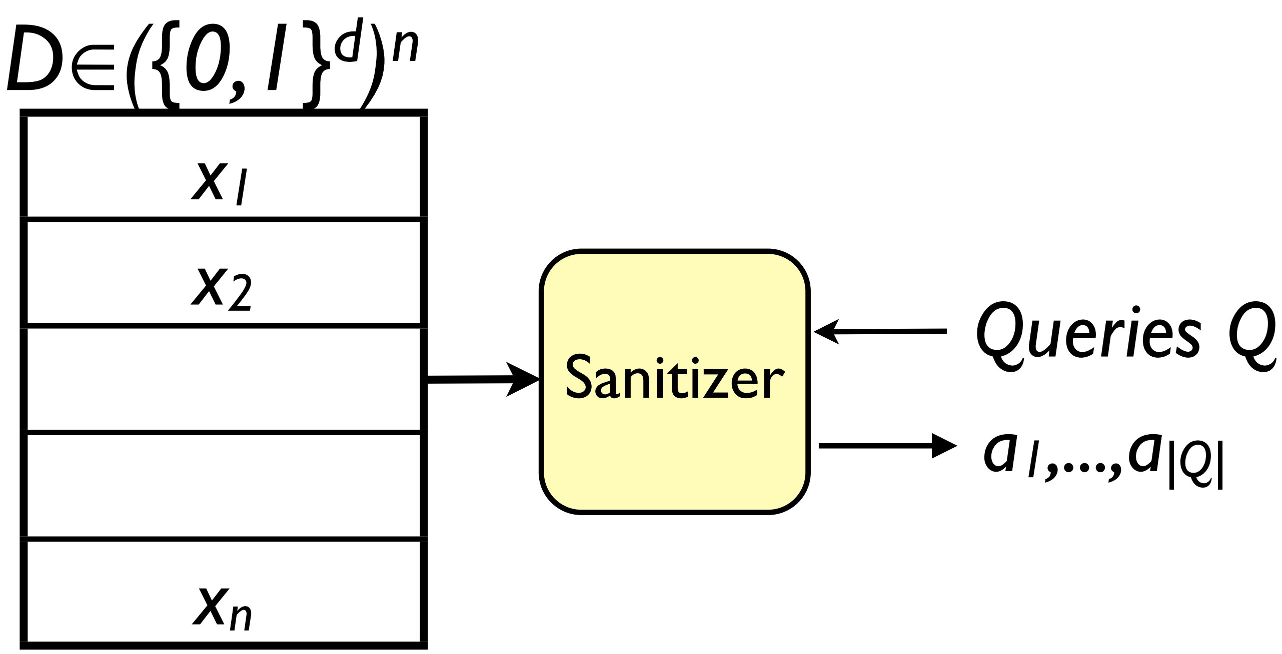
$q(x)$ = GiveYouUp $\vee$ LetYouDown

$D \in (\{0,1\}^d)^n$

GiveYouUp?	LetYouDown?	RunAround?	DesertYou?	
0	0	1	1	$q(x_1)=0$
1	1	1	1	$q(x_2)=1$
1	0	0	0	$q(x_3)=1$
1	1	0	0	$q(x_4)=1$

d attributes per record

$q(D)=3/4$



Accurate if

$|a_q - q(D)| < \alpha$

for every $q \in Q$

Private Counting Query Release

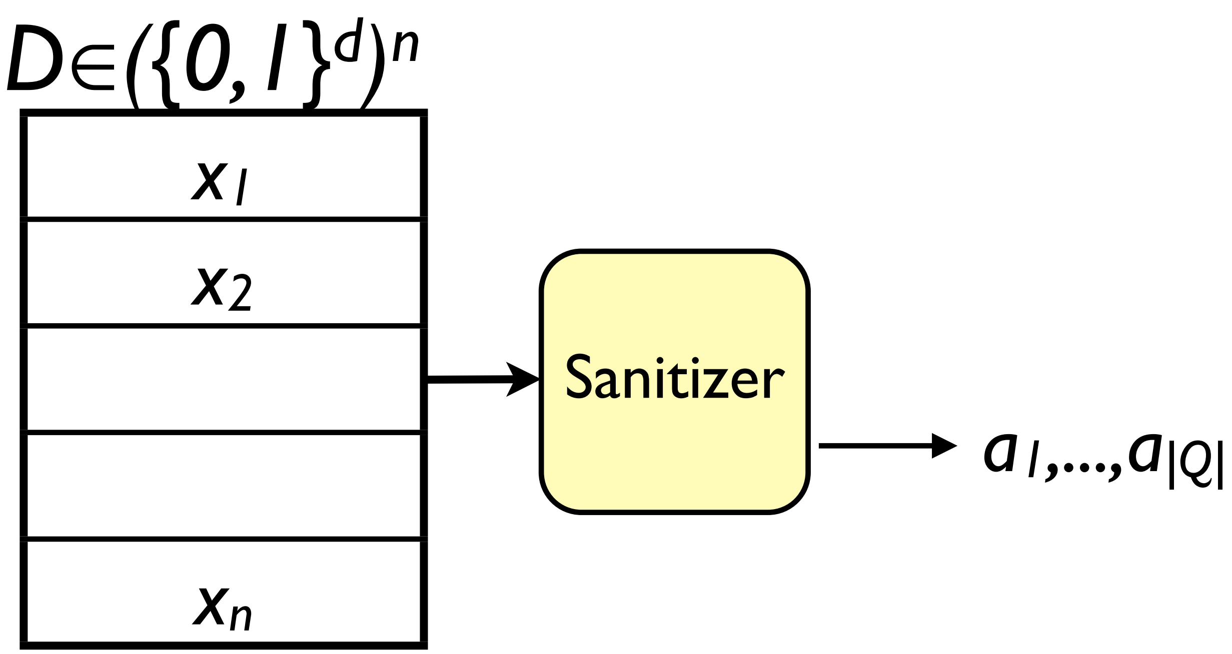
Disjunction query: *What fraction of records satisfy a given monotone k-way disjunction q_S, $|S| \leq k$?*

$q_S(x) = \vee_{i \in S} x_i$

$D \in (\{0,1\}^d)^n$

GiveYouUp?	LetYouDown?	RunAround?	DesertYou?
0	0	1	1
1	1	1	1
1	0	0	0
1	1	0	0

d attributes per record



Accurate if
$|a_q - q(D)| < .01$
for every $q \in Q$

Private Counting Query Release

Disjunction query: *What fraction of records satisfy a given monotone k-way disjunction q_S, $|S| \leq k$?*

$q_S(x) = \vee_{i \in S} x_i$

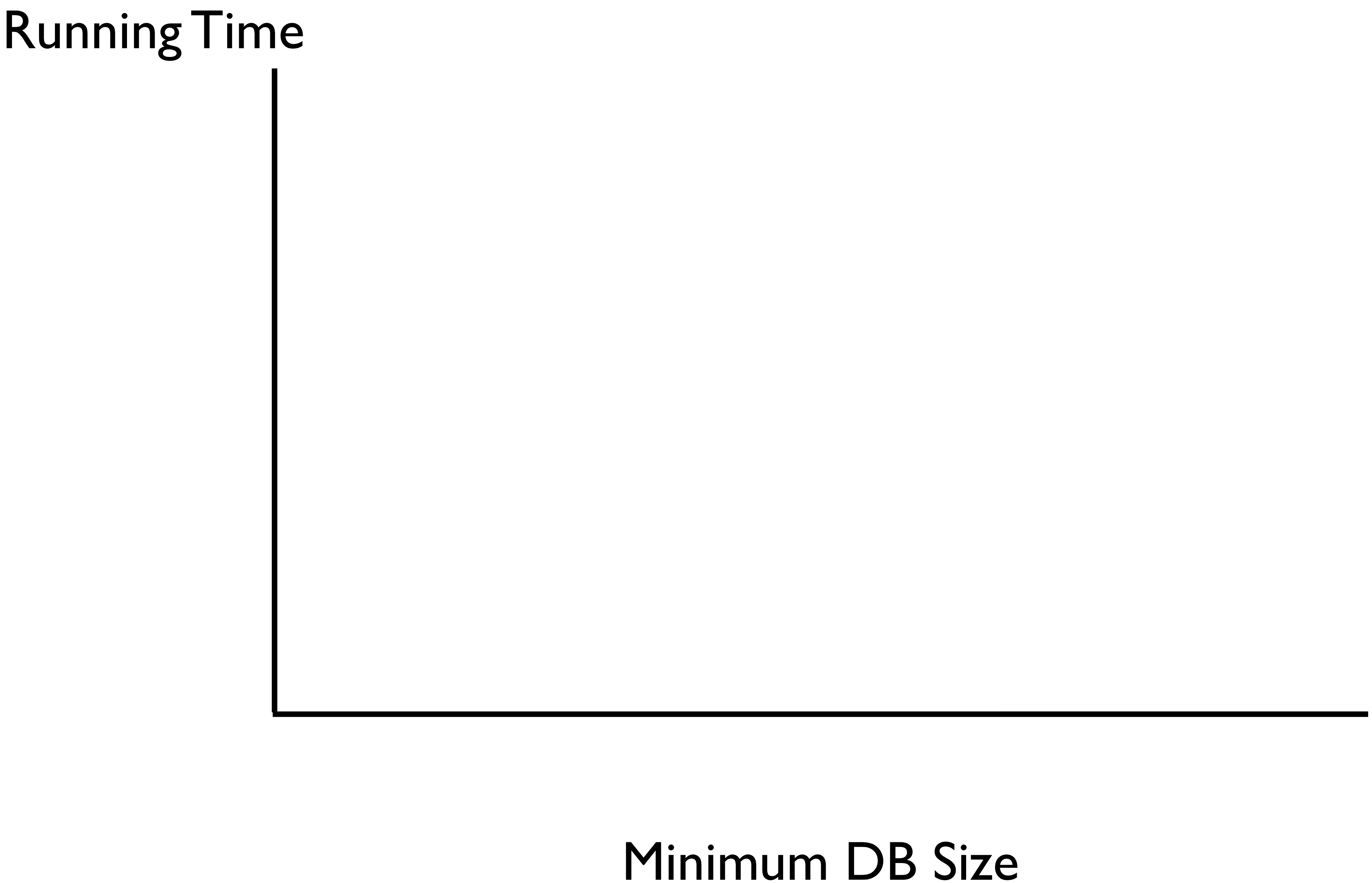
$D \in (\{0,1\}^d)^n$

GiveYouUp?	LetYouDown?	RunAround?	DesertYou?
0	0	1	1
1	1	1	1
1	0	0	0
1	1	0	0

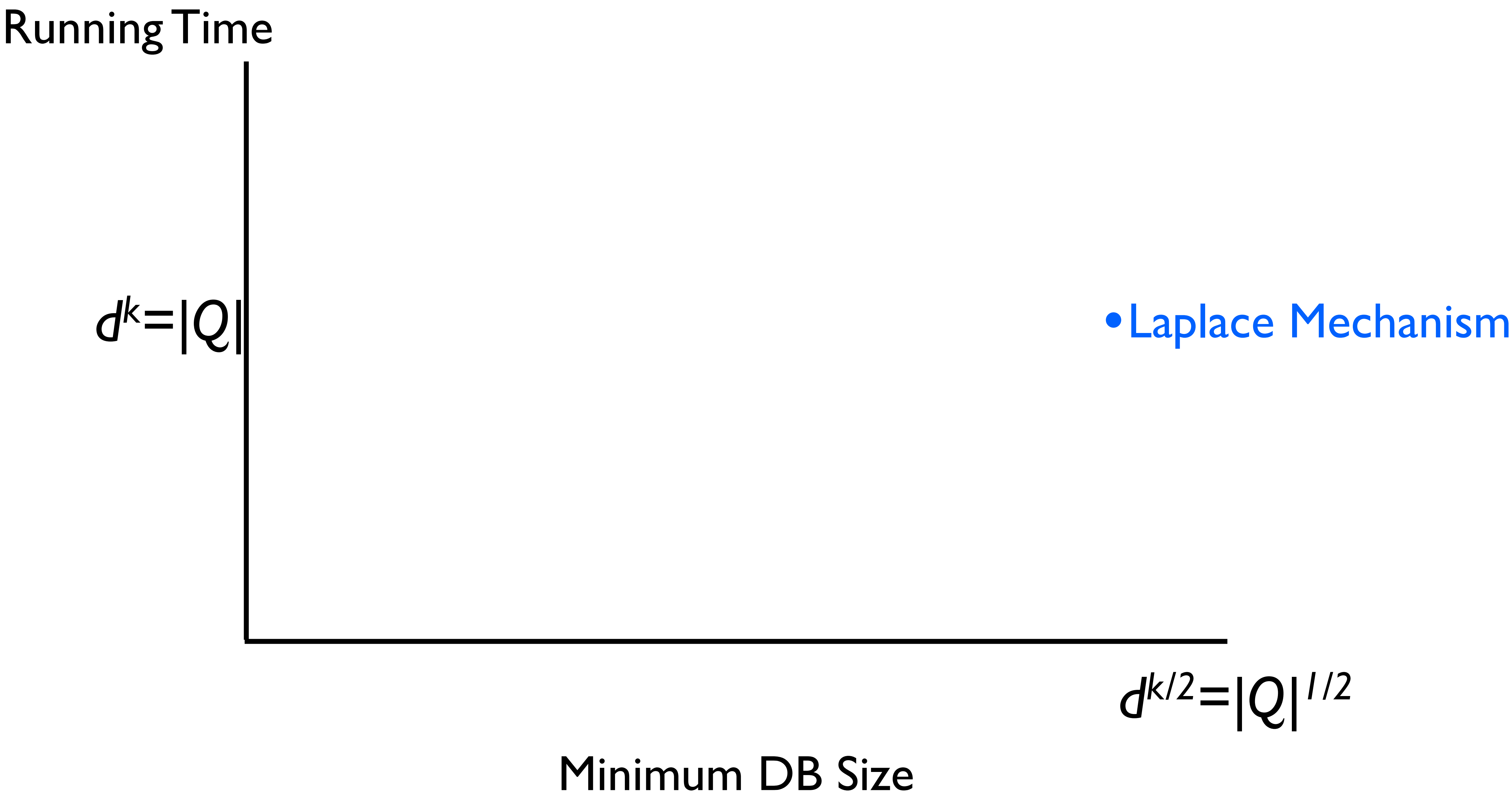
d attributes per record

- Useful facts:
  - Number of *k*-way disj's is *d-choose-k* $\sim d^k$
  - Equivalent to conjunctions / marginal queries / contingency tables

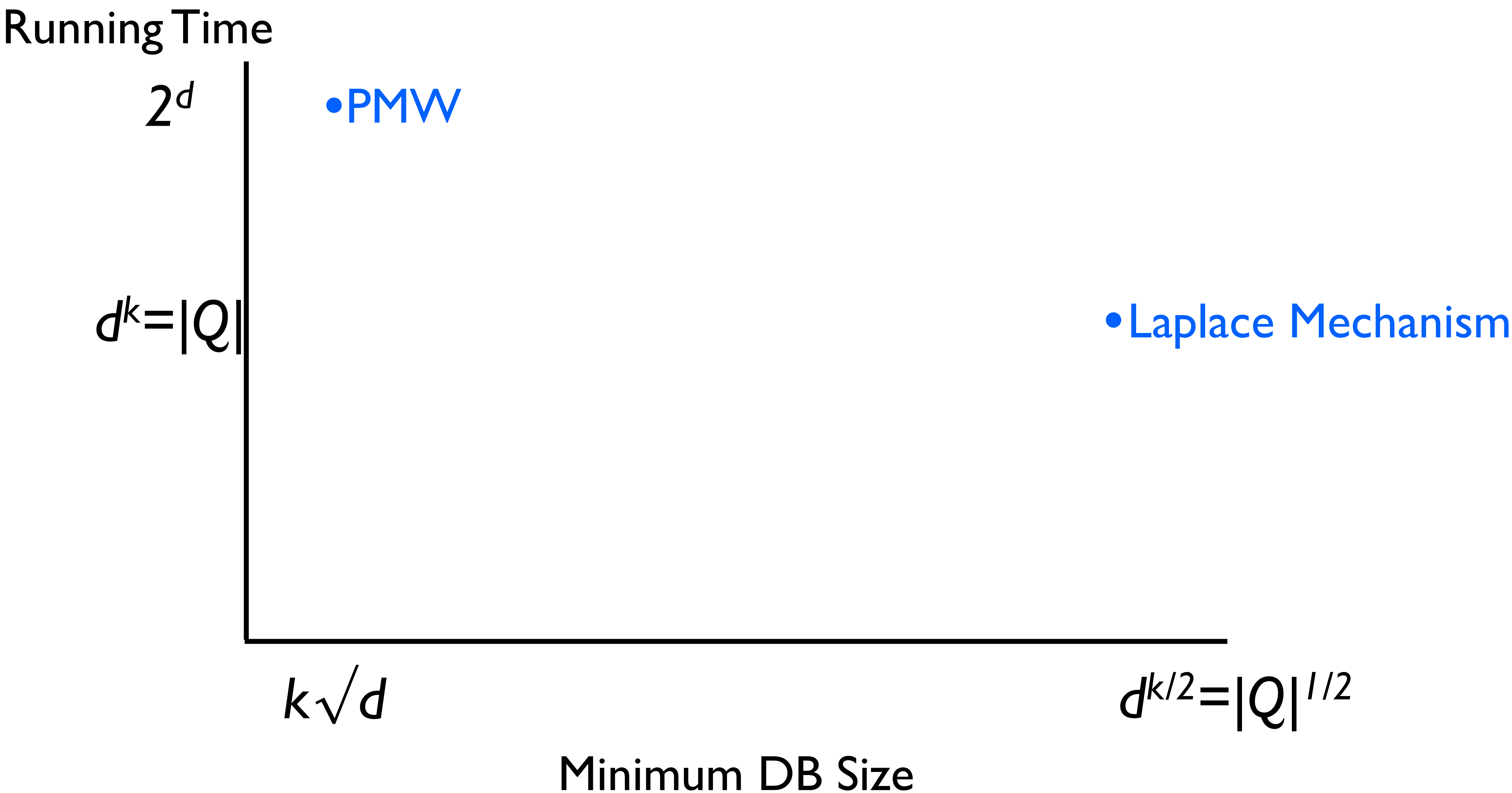
Algorithms for Disjunctions



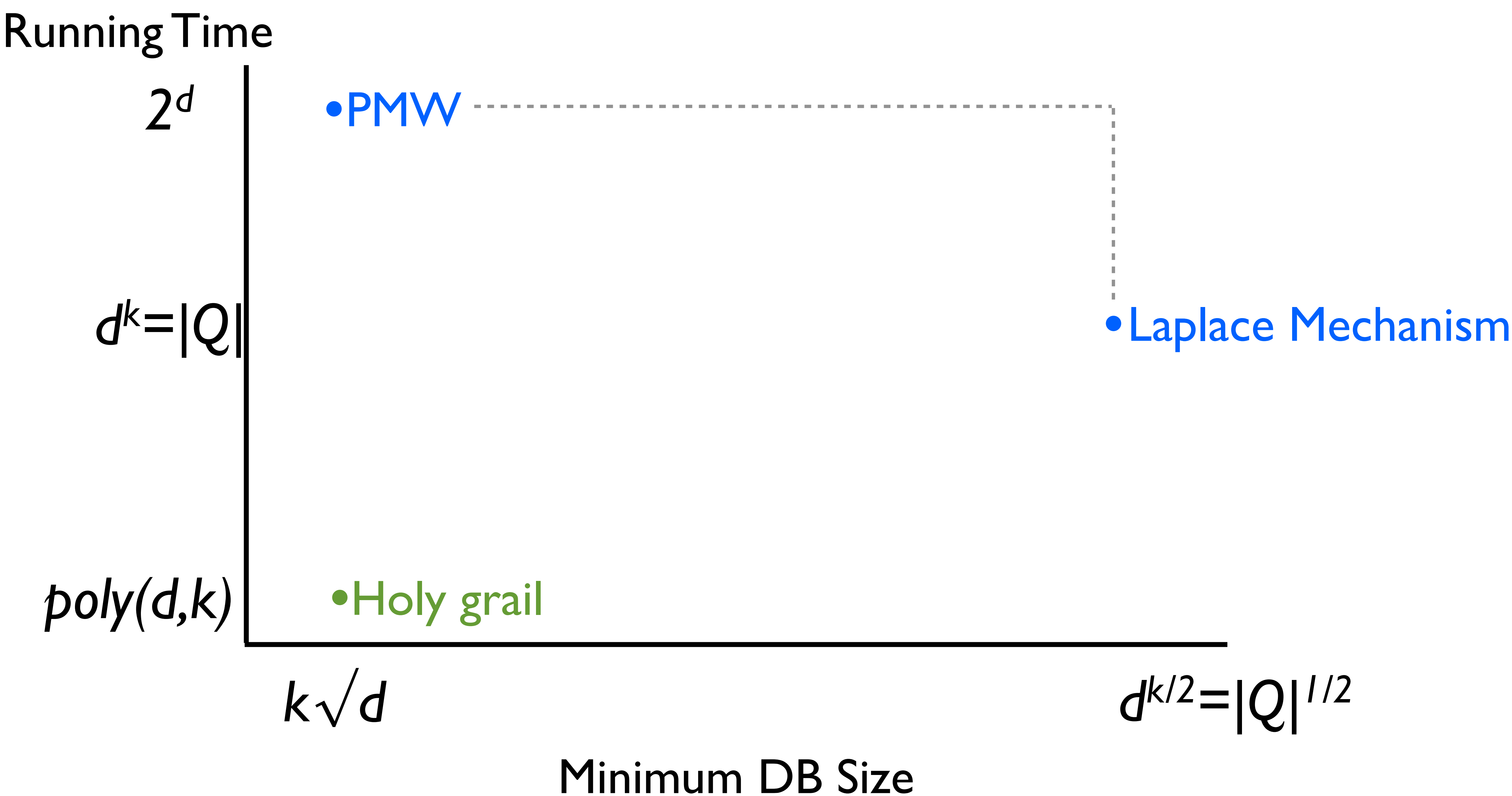
Algorithms for Disjunctions



Algorithms for Disjunctions



Algorithms for Disjunctions



Efficient Reduction to Learning

- The bottleneck in PMW is viewing the database as a distribution over $\{0,1\}^d$

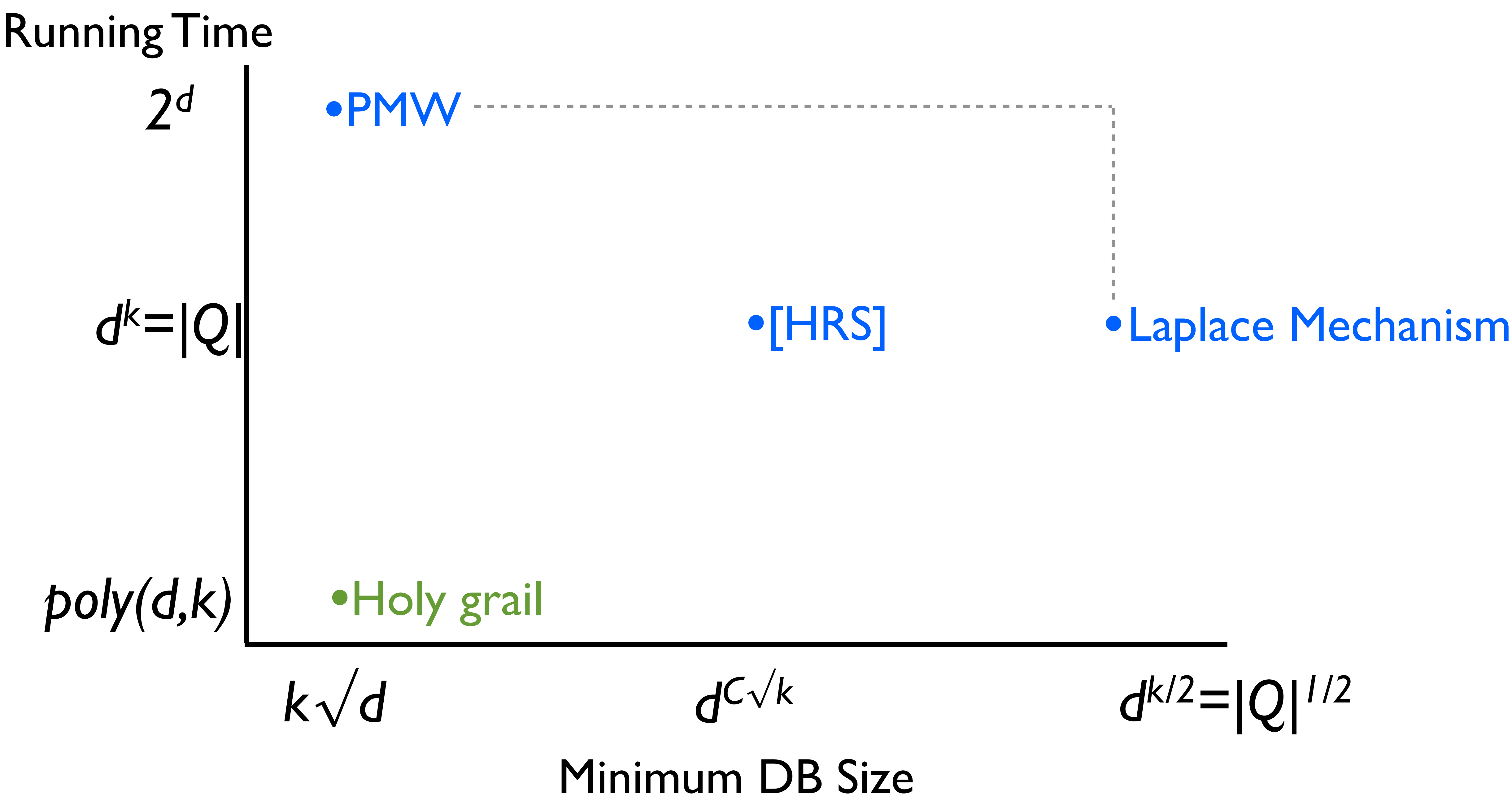
Efficient Reduction to Learning

- The bottleneck in PMW is viewing the database as a distribution over $\{0,1\}^d$
- Instead, view the database as a map $f_D: Q \rightarrow [0,1]$
  - If Q is "simple", this map might have a nice structure that leads to more efficient algorithms
  - Doesn't even need to be defined for queries outside Q

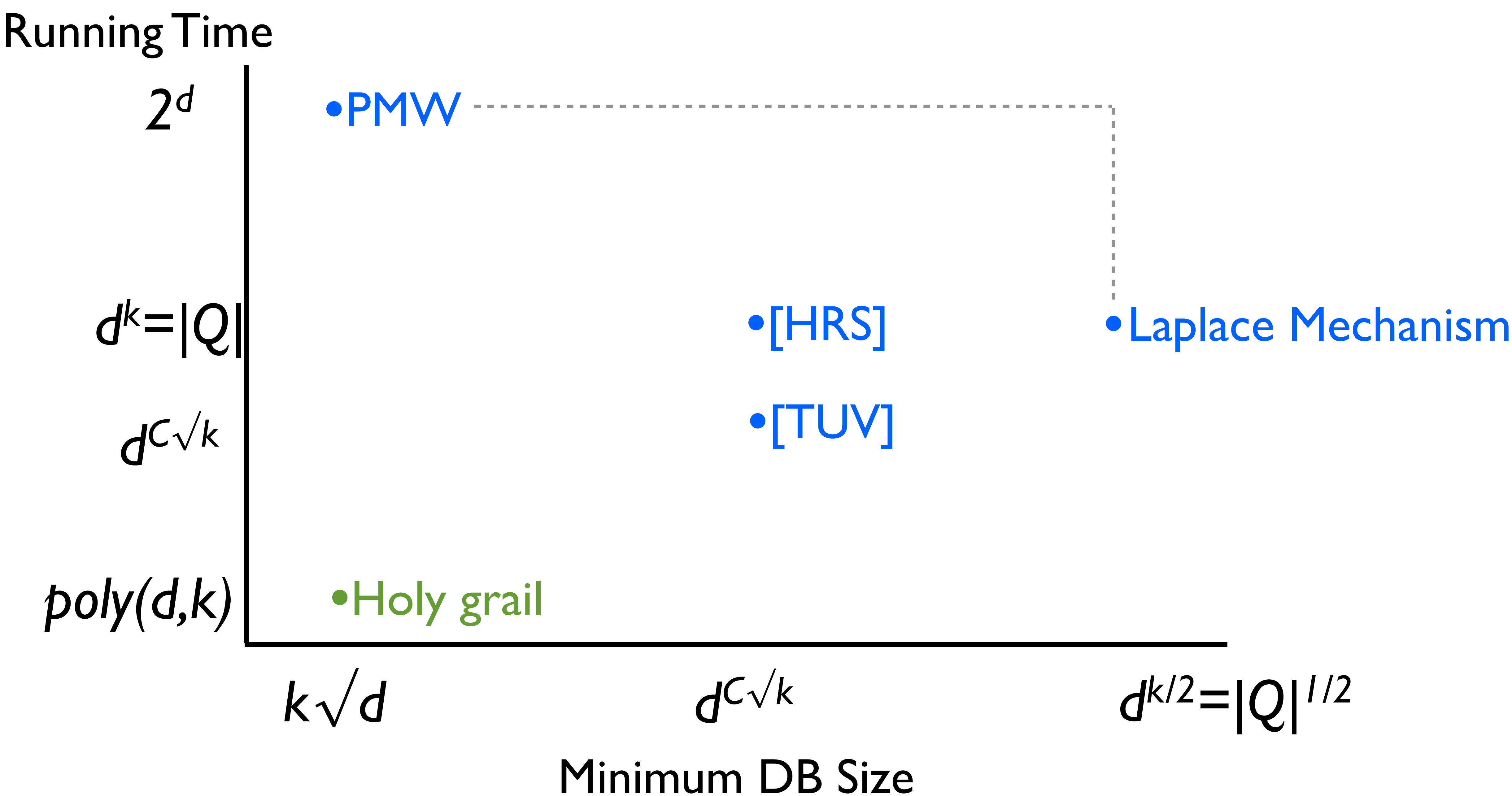
Efficient Reduction to Learning

- View the database as a map $f_D: Q \to [0,1]$
- Thm (Approximately) [HRS]: There is an efficient reduction from answering a family of queries Q to "learning" the family $\{f_D: Q \to [0,1]\}_D$
  - Approach was implicit in [GHRU,CKKL]
- Using the learning techniques, without going through the reduction, gives simpler algorithms and stronger guarantees [TUV, CTUW]

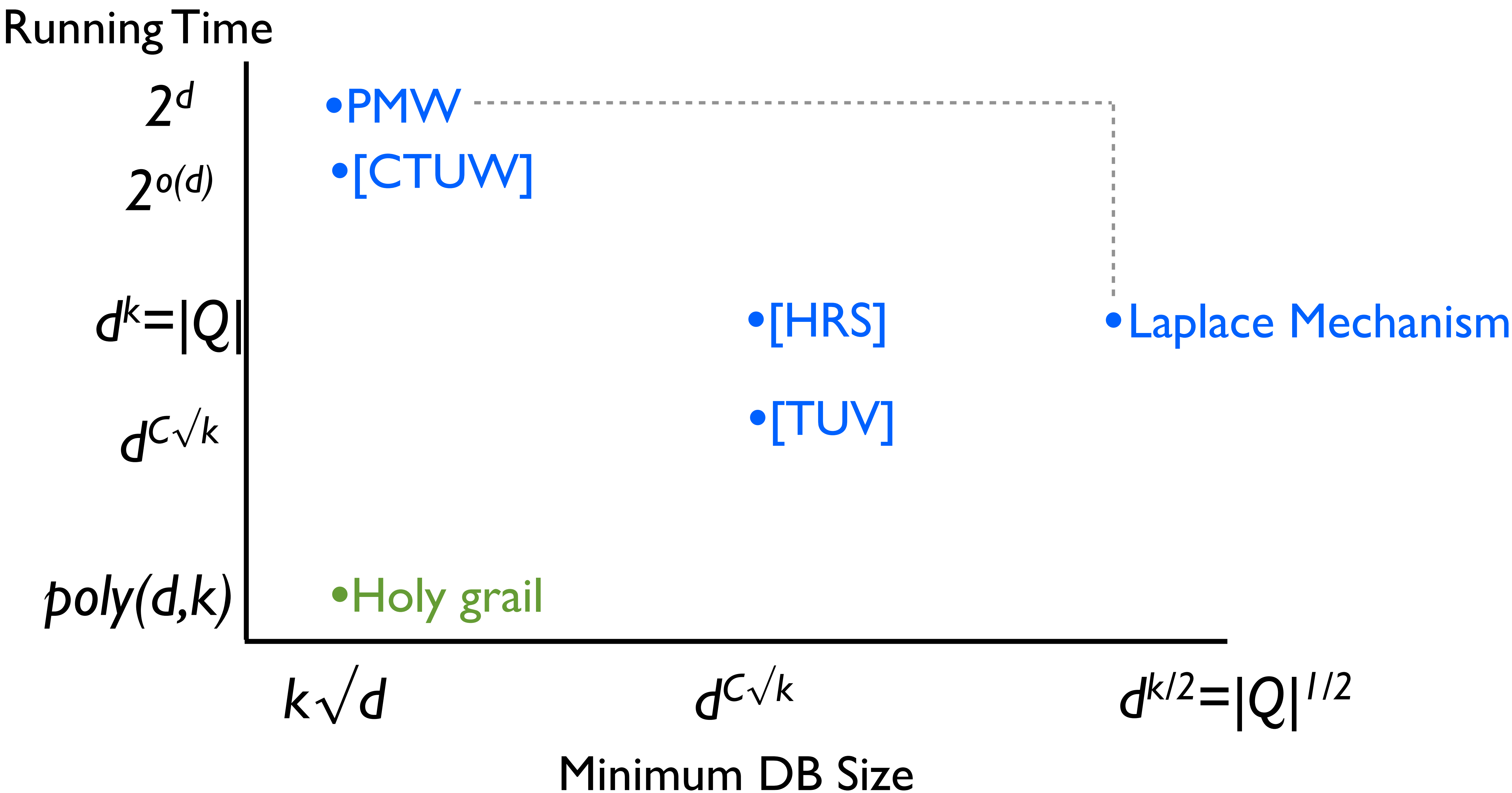
Algorithms for Disjunctions



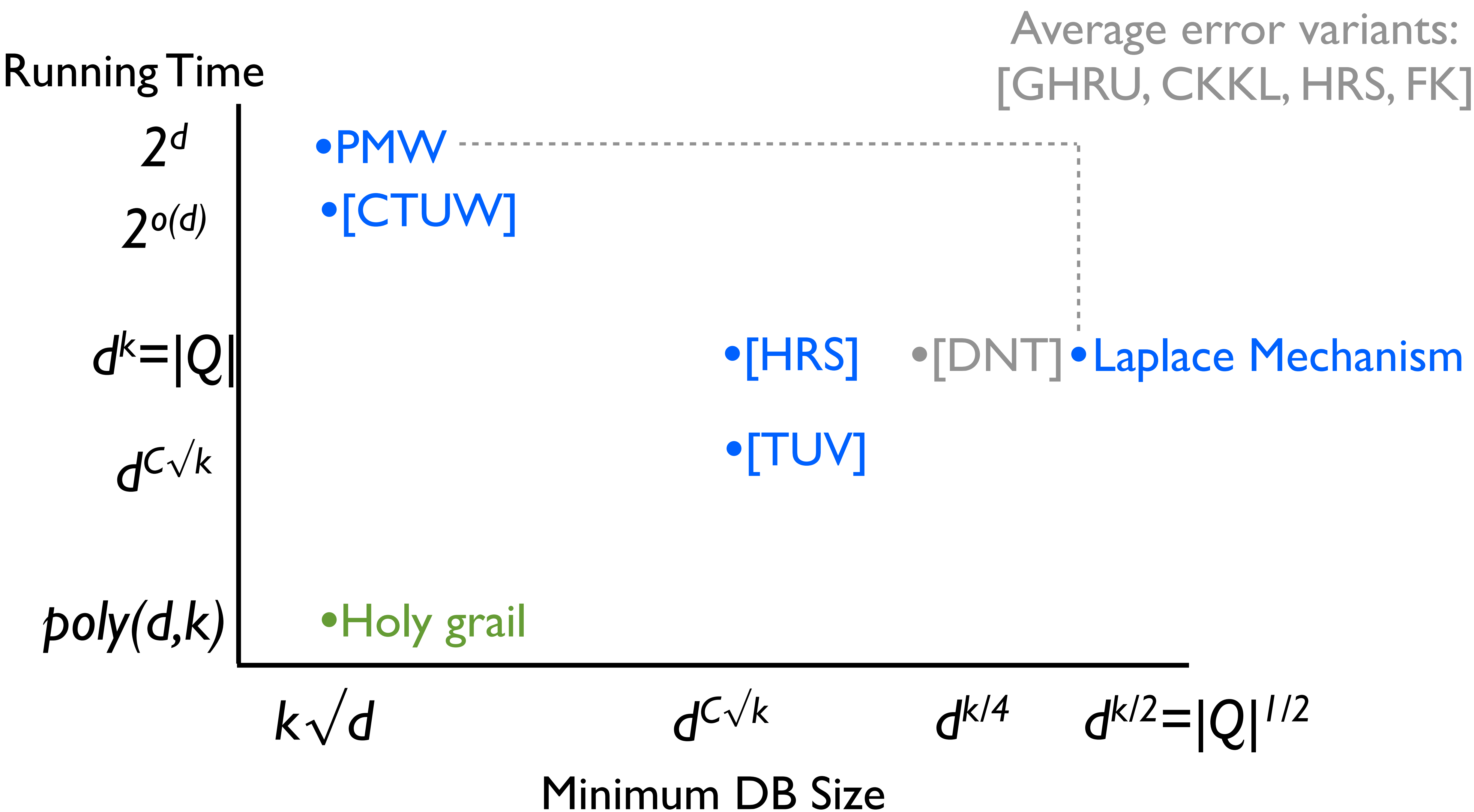
Algorithms for Disjunctions



Algorithms for Disjunctions



Algorithms for Disjunctions



Low-Weight Bases

- Instead, view the database as a map $f_D: Q \rightarrow [0,1]$
  - If Q is "simple", this map might have a nice structure that leads to more efficient algorithms
  - For disjunctions, f_D will be a "low-weight" linear combination of a small number of "basis functions"

Multiplicative Weights

Set of experts $X=\{0,1\}^d$

Distribution over $X=\{0,1\}^d$

D .25 .25 .25 .25

$q(D) = \langle D,q \rangle$

Losses for each expert

Truth table of q in $[0,1]^X$

1 0 1 0 q

$q_x = 1$ *iff* $q(x) = 1$

Multiplicative Weights

Basis of functions $\{f_x\}$, x in $\{0,1\}^d$

Weight 1 linear comb of fns in $\{f_x\}$

D .25 .25 .25 .25 $q(D) = \langle D,q \rangle$

Losses for each expert

Truth table of q in $[0,1]^X$

1 0 1 0 q

Query function on a row:

$f_x(q) = q(x)$

Query function on a DB:

$f_D(q) = (1/n)\Sigma_i f_{xi}(q)$

Losses for an expert x:

$f_x(q) = q(x)$

No-Regret Learning Algorithms

Set of experts X

Losses for each expert

	Distribution over X					$[0,1]^X$				
D_1	.25	.25	.25	.25	Loss is <D_1,L_1>	1	0	1	0	L_1
D_2	.20	.30	.20	.30	Loss is <D_2,L_2>	0	0	1	0	L_2
					⋮					
D_T	.23	.32	.15	.32	Loss is <D_T,L_T>	0	0	0	1	L_T

For any distribution D, sequence $L_1, \ldots, L_T$,

$$\sum_{t=1}^{T} \langle D_t - D, L_t \rangle \leq \sqrt{T \log |X|}$$

No-Regret Learning Algorithms

Set of experts $X = F$

Losses for each expert

Weight W linear comb over $X = F$ $\qquad [0,1]^X$

D_1	1	1	1	1	Loss is <D_1,L_1>	1	0	1	0	L_1
D_2	.80	1.20	.80	1.20	Loss is <D_2,L_2>	0	0	1	0	L_2
					⋮					
D_T	.92	1.28	.60	1.28	Loss is <D_T,L_T>	0	0	0	1	L_T

For any weight W linear combination D, sequence $L_1, \ldots, L_T$,

$$\sum_{t=1}^{T} \langle D_t - D, L_t \rangle \leq W \sqrt{T \log |X|}$$

Multiplicative Weights

Basis of functions $\{f_x\}$, x in $\{0,1\}^d$

Weight 1 linear comb of fns in $\{f_x\}$

D .25 .25 .25 .25 $q(D) = \langle D,q \rangle$

Losses for each expert

Truth table of q in $[0,1]^X$

1 0 1 0 q

Query function on a row:

$f_x(q) = q(x)$

Query function on a DB:

$f_D(q) = (1/n)\Sigma_i f_{x_i}(q)$

Losses for an expert x:

$f_x(q) = q(x)$

> The Private MW algorithm treats the database as a weight 1 linear comb. of a set of 2^d functions f_x: $\{All\ Queries\} \rightarrow \{0,1\}$

Multiplicative Weights

Basis of functions $\{f_x\}$, x in $\{0,1\}^d$

Weight 1 linear comb of fns in $\{f_x\}$

D .25   .25   .25   .25 $q(D) = \langle D,q \rangle$

Losses for each expert

Truth table of q in $[0,1]^X$

1 0 1 0 q

Query function on a row:

$f_x(q) = q(x)$

Query function on a DB:

$f_D(q) = (1/n)\Sigma_i f_{xi}(q)$

Losses for an expert x:

$f_x(q) = q(x)$

Improved algs for disj's treat the database as a weight W linear comb. of a set of S functions f: $\{k\text{-way disj's}\} \rightarrow \{0,1\}$

Low-Weight Bases

- View the database as a map $f_D: Q \rightarrow [0,1]$
- Let $F = \{f: Q \rightarrow \{0,1\}\}$ be a set of functions
- Def: F is a weight-W approximate basis wrt Q if for every database D, there exists a weight-W linear combination of functions in F, p_D, such that for every $q \in Q$, $|f_D(q) - p_D(q)| \leq .001$

No-Regret Learning Algorithms

Set of experts X | Losses for each expert

Distribution over X | $[0,1]^X$

D_1	.25	.25	.25	.25	Loss is $\langle D_1,L_1 \rangle$	1	0	1	0	L_1
D_2	.20	.30	.20	.30	Loss is $\langle D_2,L_2 \rangle$	0	0	1	0	L_2
					⋮					
D_T	.23	.32	.15	.32	Loss is $\langle D_T,L_T \rangle$	0	0	0	1	L_T

For any distribution D, sequence $L_1, \ldots, L_T$,

$$\sum_{t=1}^{T} \langle D_t - D, L_t \rangle \leq \sqrt{T \log |X|}$$

No-Regret Learning Algorithms

Set of experts $X = F$

Losses for each expert

Weight W linear comb over $X = F$

$[0,1]^X$

D_1	1	1	1	1	Loss is $\langle D_1, L_1 \rangle$	1	0	1	0	L_1
D_2	.80	1.20	.80	1.20	Loss is $\langle D_2, L_2 \rangle$	0	0	1	0	L_2
					⋮					
D_T	.92	1.28	.60	1.28	Loss is $\langle D_T, L_T \rangle$	0	0	0	1	L_T

For any weight W linear combination D, sequence $L_1, \ldots, L_T$,

$$\sum_{t=1}^{T} \langle D_t - D, L_t \rangle \leq W \sqrt{T \log |X|}$$

Recap

Thm: PMW takes a database $D \in (\{0,1\}^d)^n$ and a set of counting queries Q, satisfies (ε,δ)-DP and, if

$$n \gtrsim d^{1/2} log|Q|/\alpha^2\varepsilon,$$

it outputs D_T such that for every $q \in Q$,

$$|q(D) - q(D_T)| \leq \alpha$$

Thm: PMW runs in time $poly(n, 2^d, |q_1| + ... + |q_{|Q|}|)$

Recap

Thm [CTUW]: PMW (run with F, a weight-W approximate basis wrt Q) takes a database $D \in (\{0,1\}^d)^n$, satisfies (ε,δ)-DP and, if

$$n \gtrsim W d^{1/2} \log|Q| / \alpha^2 \varepsilon,$$

it outputs D_T such that for every $q \in Q$,

$$|q(D) - q(D_T)| \leq .01$$

Thm: PMW runs in time $poly(n, |F|, |q_1| + \ldots + |q_{|Q|}|)$

Low-Weight Bases

- But where do these low-weight bases come from?
- Polynomial approximations!
  - Extremely prevalent in PAC/agnostic learning. Underlies the most-efficient learning algorithms.
  - First used for disjunctions by [CKKL],[HRS]

Low-Weight Bases

Query on a row:

$q(x) = x_1 \vee x_2$

Query on a DB:

$q(D) = (1/n)\Sigma_i\, q(x_i)$

$D \in (\{0,1\}^d)^n$

x_1?	x_2?	x_3?	x_4?
1	1	1	0
1	1	0	0
0	0	1	1
0	0	0	1

Low-Weight Bases

$D \in (\{0,1\}^d)^n$

x_1?	x_2?	x_3?	x_4?
1	1	1	0
1	1	0	0
0	0	1	1
0	0	0	1

Query on a row:

$q_y(x) = x_1 \vee x_2$

Query on a DB:

$q_y(D) = (1/n)\Sigma_i\, q_y(x_i)$

Each query described by a d-bit string $y \in \{0,1\}^d$

Low-Weight Bases

$D \in (\{0,1\}^d)^n$

x_1?	x_2?	x_3?	x_4?
1	1	1	0
1	1	0	0
0	0	1	1
0	0	0	1

Query on a row:

$q_y(x) = x_1 \vee x_2$

Query on a DB:

$q_y(D) = (1/n)\Sigma_i \, q_y(x_i)$

Each query described by a d-bit string $y \in \{0,1\}^d$

Query function on a row:

$f_x(y) = q_y(x)$

Query function on a DB:

$f_D(y) = (1/n)\Sigma_i \, f_{x_i}(y)$

Low-Weight Bases

$D \in (\{0,1\}^d)^n$

x_1?	x_2?	x_3?	x_4?
1	1	1	0
1	1	0	0
0	0	1	1
0	0	0	1

Query on a row:

$q_y(x) = x_1 \vee x_2$

Query on a DB:

$q_y(D) = (1/n)\Sigma_i\, q_y(x_i)$

Each query described by a d-bit string $y \in \{0,1\}^d$

Query function on a row:

$f_x(y) = q_y(x)$

Query function on a DB:

$f_D(y) = (1/n)\Sigma_i\, f_{x_i}(y)$

Approximation: For every x, want $p_x(y)$ s.t.

- p_x has degree T
- p_x has weight W
- for every y corresponding to a k-way disj. $|p_x(y) - f_x(y)| \leq .001$

Low-Weight Bases

$D \in (\{0,1\}^d)^n$

x_1?	x_2?	x_3?	x_4?
1	1	1	0
1	1	0	0
0	0	1	1
0	0	0	1

Query on a row:

$q_y(x) = x_1 \vee x_2$

Query on a DB:

$q_y(D) = (1/n)\Sigma_i\, q_y(x_i)$

Each query described by a d-bit string $y \in \{0,1\}^d$

Query function on a row:

$f_x(y) = q_y(x)$

Query function on a DB:

$f_D(y) = (1/n)\Sigma_i\, f_{xi}(y)$

$f_{(1,1,1,0)}(y_1,\ldots,y_d) =$
$y_1 \vee y_2 \vee y_3$

Disjunction on y
(Coincidentally)

Low-Weight Bases

$D \in (\{0,1\}^d)^n$

x_1?	x_2?	x_3?	x_4?
1	1	1	0
1	1	0	0
0	0	1	1
0	0	0	1

Query on a row:

$q_y(x) = x_1 \vee x_2$

Query on a DB:

$q_y(D) = (1/n)\Sigma_i\, q_y(x_i)$

Each query described by a d-bit string $y \in \{0,1\}^d$

Query function on a row:

$f_x(y) = q_y(x)$

Query function on a DB:

$f_D(y) = (1/n)\Sigma_i\, f_{x_i}(y)$

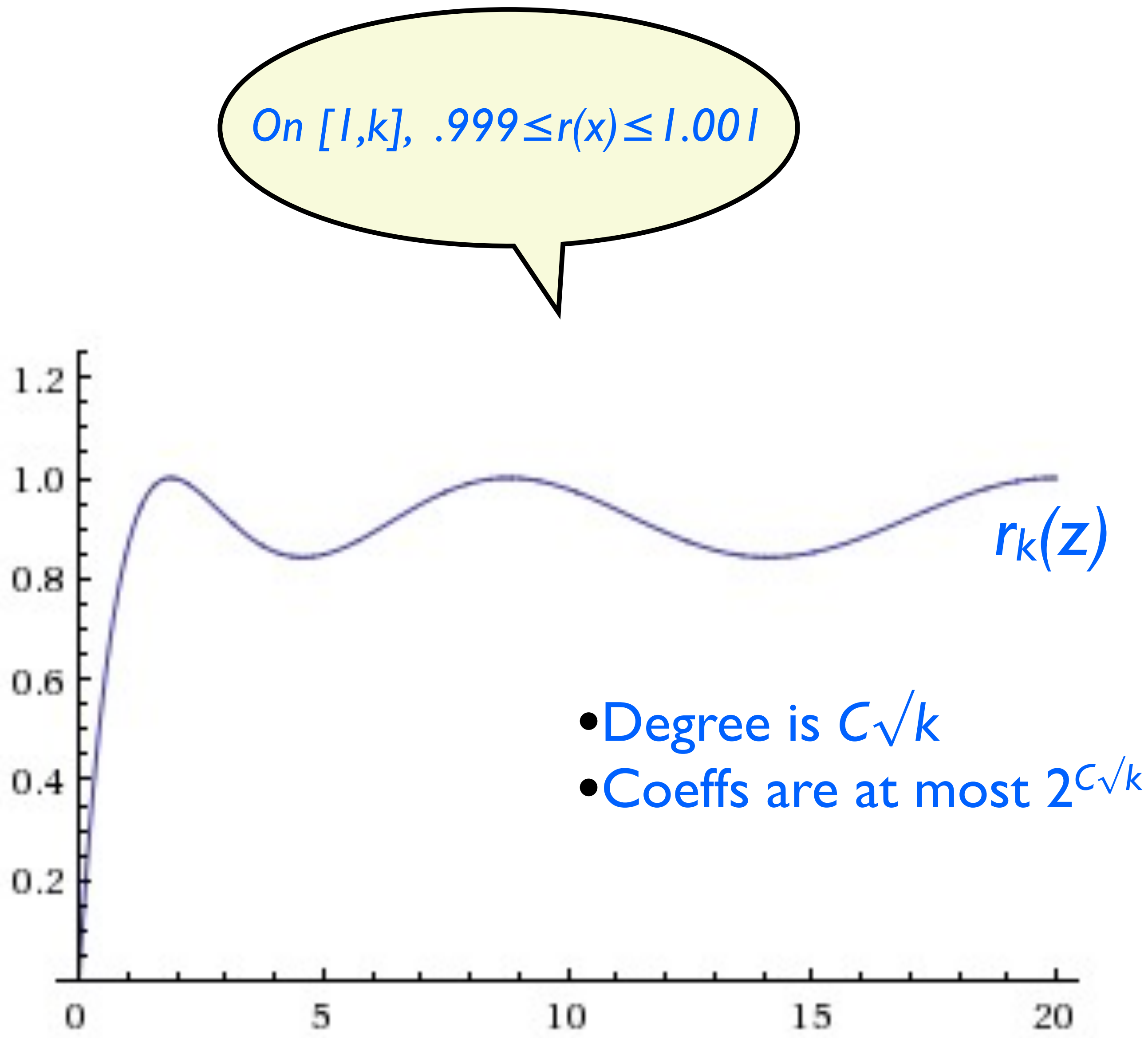
$f(y_1,\ldots,y_d) = OR(y_1,\ldots,y_d)$

Sufficient to approx. d-variate OR function on inputs with at most k non-zeros

Recap

- Suppose there is a *d*-variate polynomial *p* of deg *T* and weight *W* such that for every *y in* $\{0,1\}^d$ with at most *k* non-zeroes $|OR(y) - p(y)| \leq .001$.
- Then there is a weight-*W* approximate basis wrt *k*-way disj's of size roughly *d-choose-T*
  - *F = {all d-variate monomials of degree at most T}*

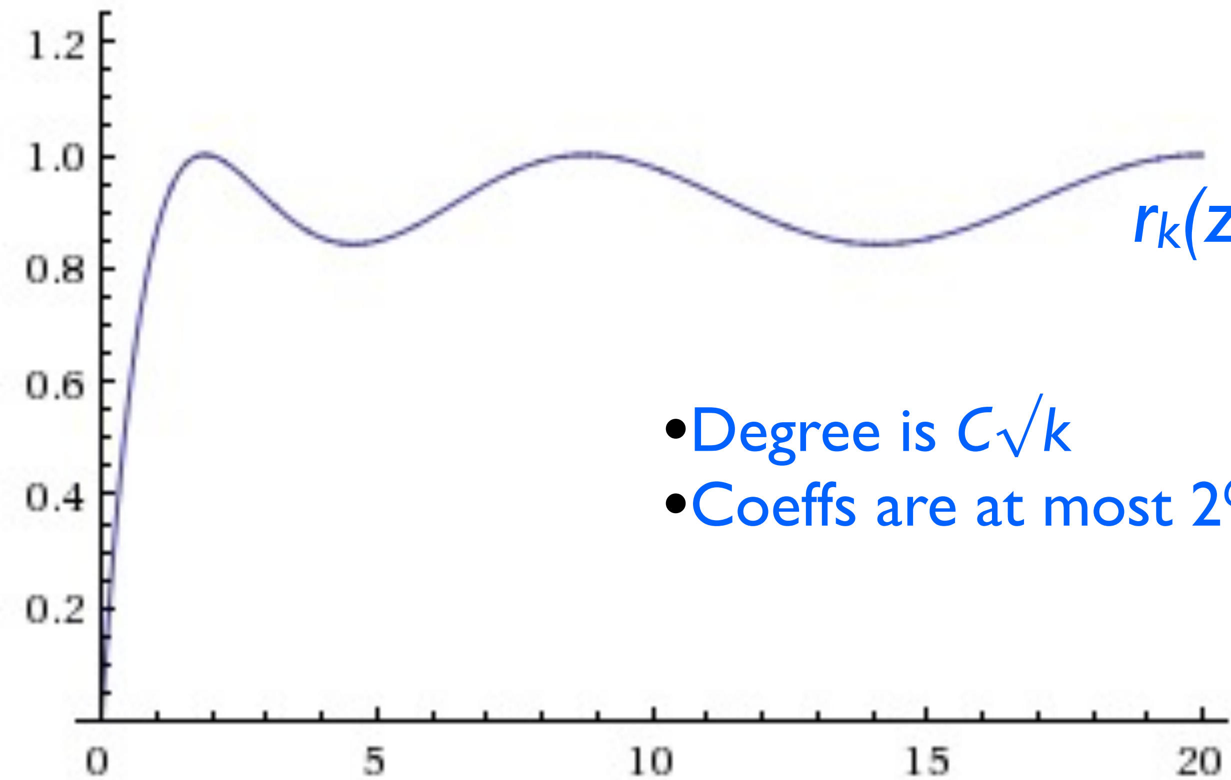
Approximating OR (Low Weight)



- Want to approx $OR(y_1,\ldots,y_d)$ on inputs with k non-zeros

Approximating OR (Low Weight)

On $[1,k]$, $.999 \leq r(x) \leq 1.001$



$r_k(z)$

- Degree is $C\sqrt{k}$
- Coeffs are at most $2^{C\sqrt{k}}$

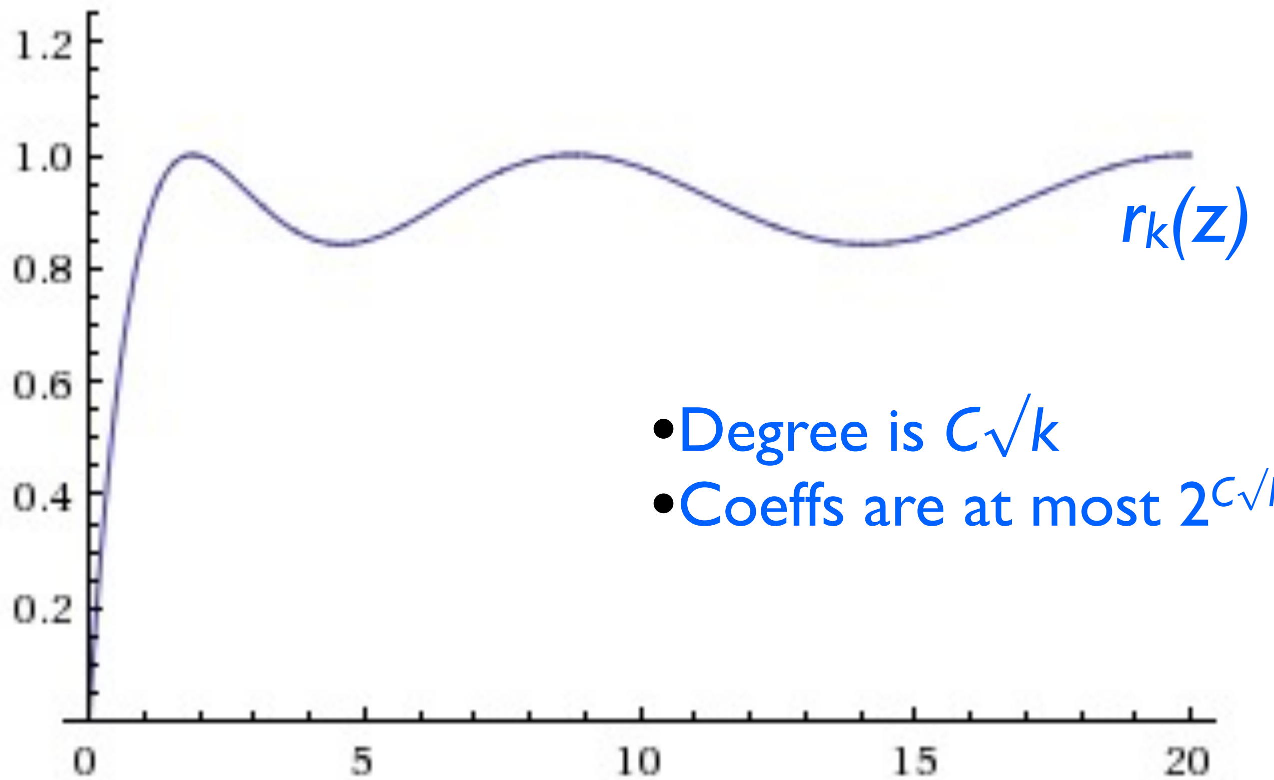
- Want to approx $OR(y_1,...,y_d)$ on inputs with k non-zeros

- Set

$$p(y_1,...,y_d) = r_k(y_1 + \ldots + y_d)$$

Approximating OR (High Weight)

On $[1,k]$, $.999 \le r(x) \le 1.001$



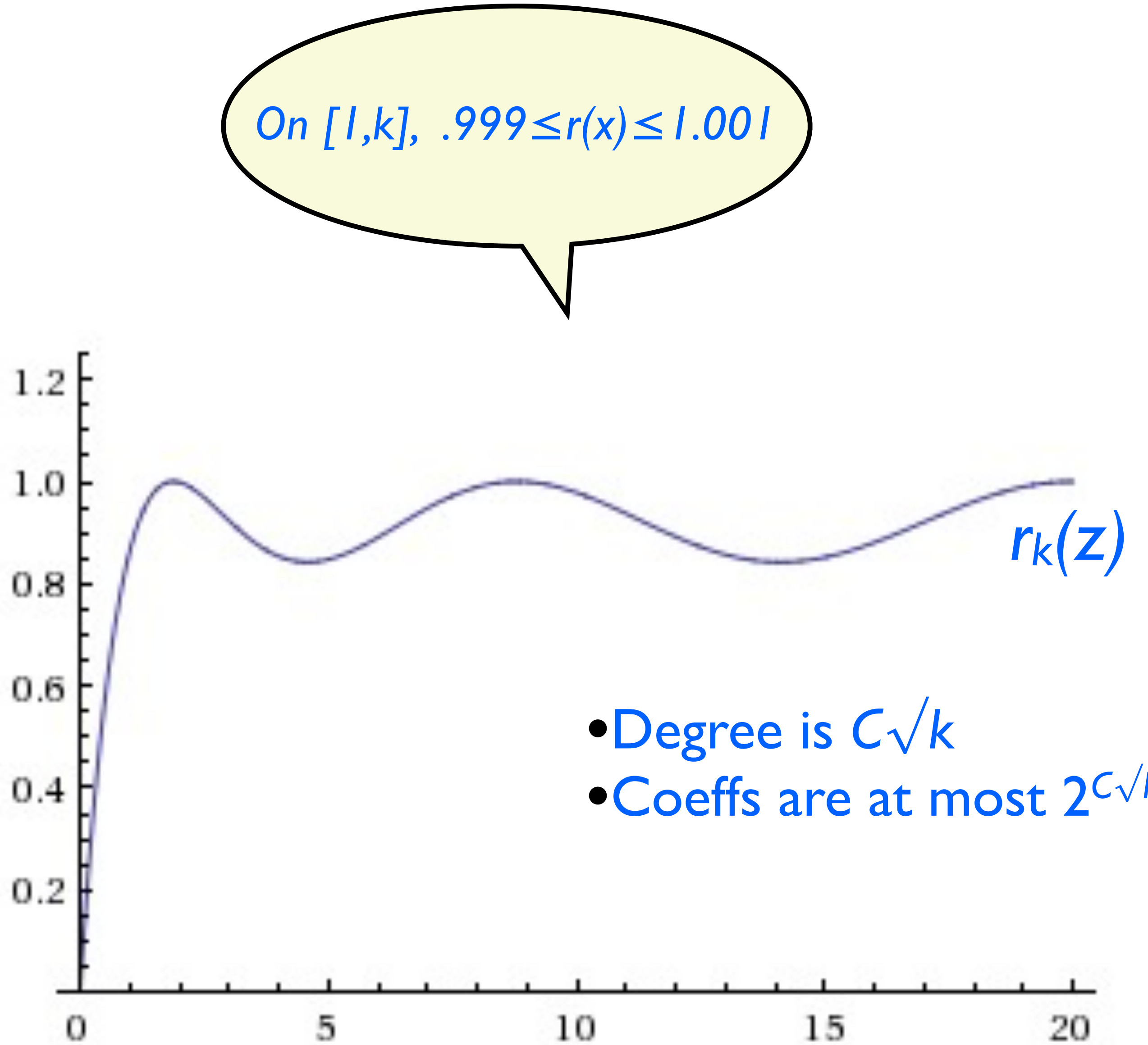
- Degree is $C\sqrt{k}$
- Coeffs are at most $2^{C\sqrt{k}}$

- Want to approx $OR(y_1,\ldots,y_d)$ on inputs with k non-zeros

- Set
$$p(y_1,\ldots,y_d) = r_k(y_1 + \ldots + y_d)$$

- If $OR(y_1,\ldots,y_d)=0$, then
$$p(y_1,\ldots y_d) = r_k(0) = 0$$
- If $OR(y_1,\ldots,y_d)=1$, then $1 \le y_1+\ldots+y_d \le k$
$$p(y_1,\ldots,y_d) = r_k(y_1+\ldots+y_d) \approx 1$$

Approximating OR (High Weight)



- Degree is $C\sqrt{k}$
- Coeffs are at most $2^{C\sqrt{k}}$

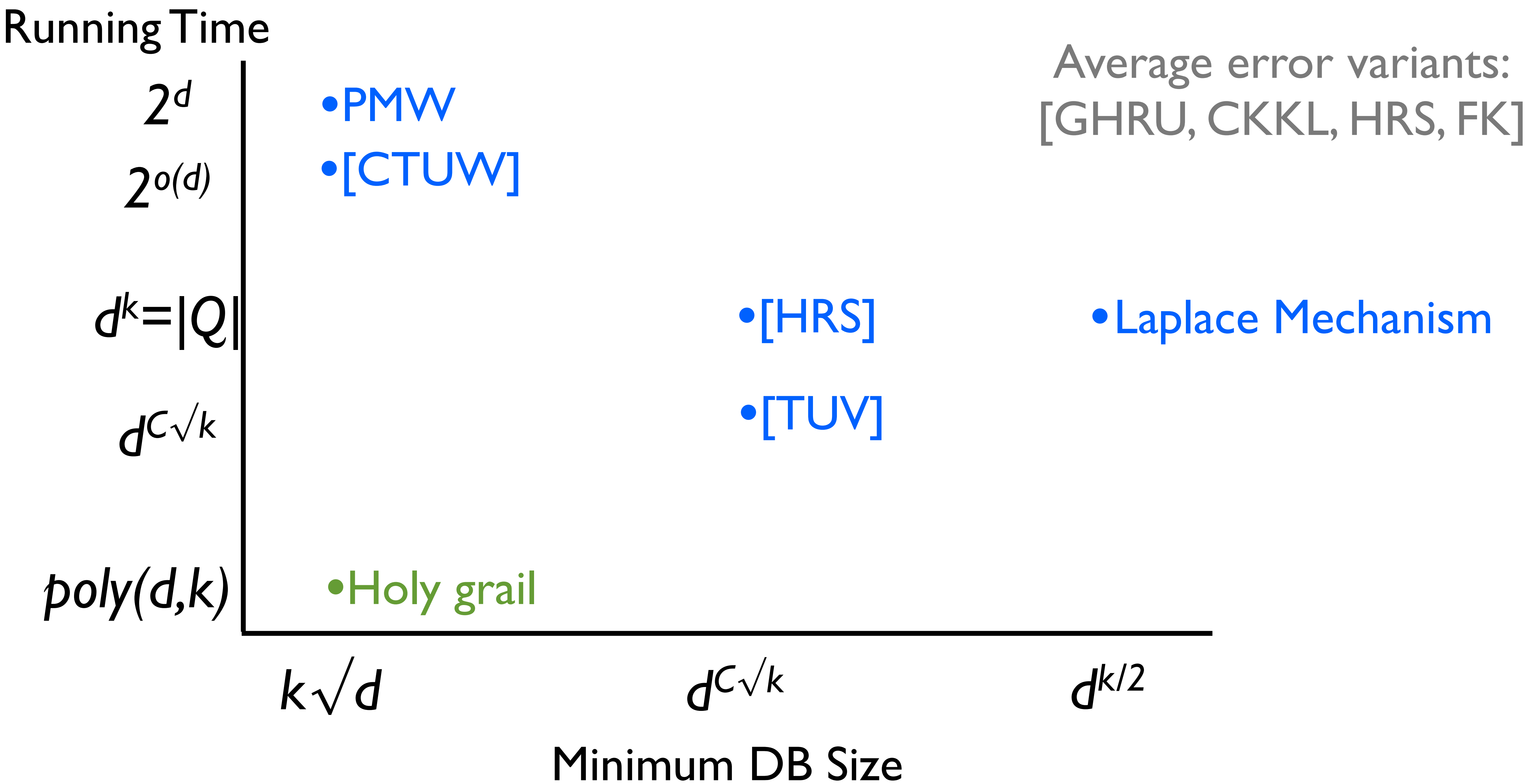
- Want to approx $OR(y_1,\ldots,y_d)$ on inputs with k non-zeros

- Set
  $p(y_1,\ldots,y_d) = r_k(y_1 + \ldots + y_d)$

- If $OR(y_1,\ldots,y_d)=0$, then
  $p(y_1,\ldots y_d) = r_k(0) = 0$
- If $OR(y_1,\ldots,y_d)=1$, then $1 \leq y_1+\ldots+y_d \leq k$
  $p(y_1,\ldots,y_d) = r_k(y_1+\ldots+y_d) \approx 1$

Polynomial has degree $C\sqrt{k}$, weight $d^{C\sqrt{k}}$

Algorithms for Disjunctions



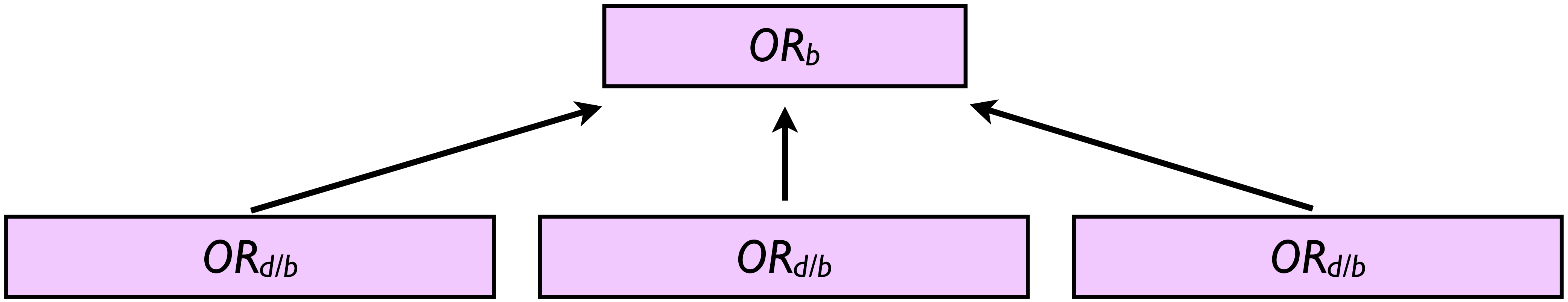
Approximating OR (Low Weight)

- Have an approximation with degree $C\sqrt{k}$ and weight $d^{C\sqrt{k}}$
- The "trivial" exact polynomial has degree d and weight 1

OR_d

Approximating OR (Low Weight)

- Have an approximation with degree $C\sqrt{k}$ and weight $d^{C\sqrt{k}}$
- The "trivial" exact polynomial has degree d and weight 1

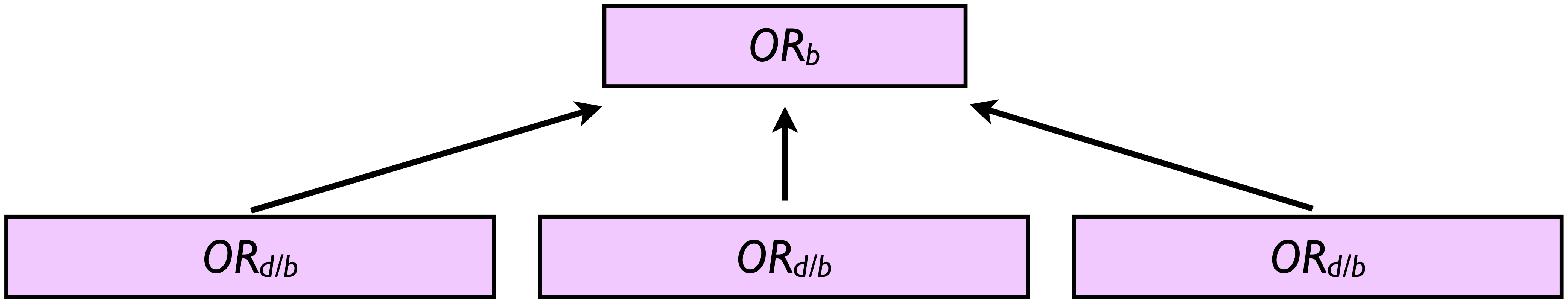


Final polynomial has degree $C(d/b)\sqrt{k}$, weight $b^{C\sqrt{k}}$

Approximating OR (Low Weight)

- Have an approximation with degree $C\sqrt{k}$ and weight $d^{C\sqrt{k}}$
- The "trivial" exact polynomial has degree d and weight 1

Use $T = C\sqrt{k}, W = b^{C\sqrt{k}}$

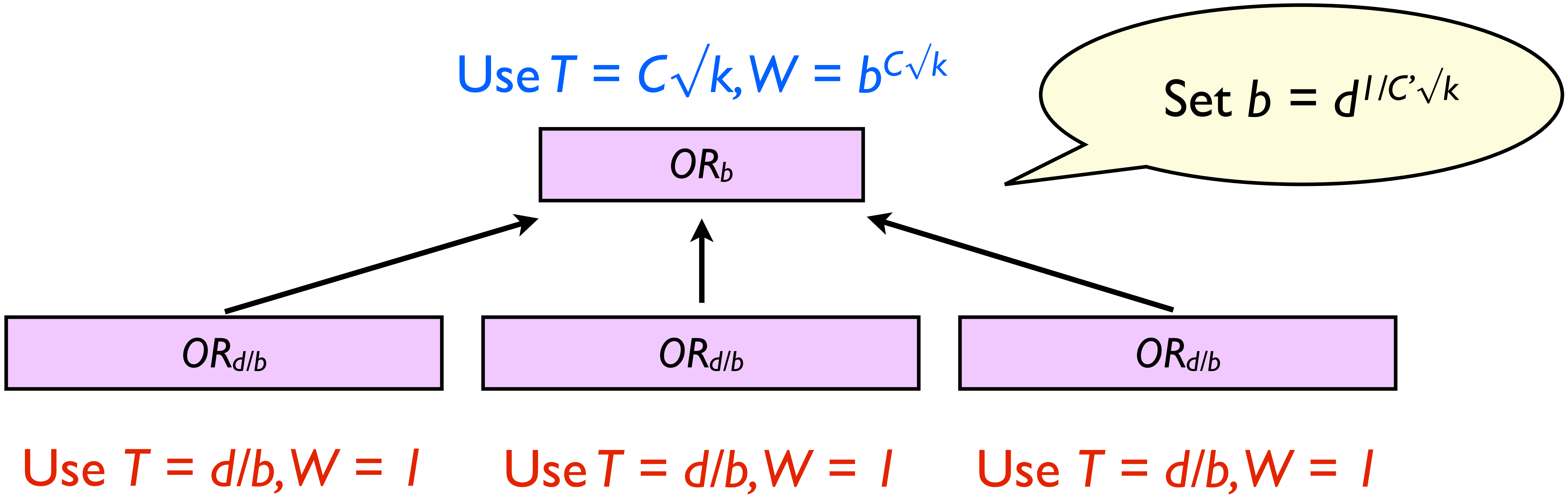


Use $T = d/b, W = 1$ Use $T = d/b, W = 1$ Use $T = d/b, W = 1$

Final polynomial has degree $C(d/b)\sqrt{k}$, weight $b^{C\sqrt{k}}$

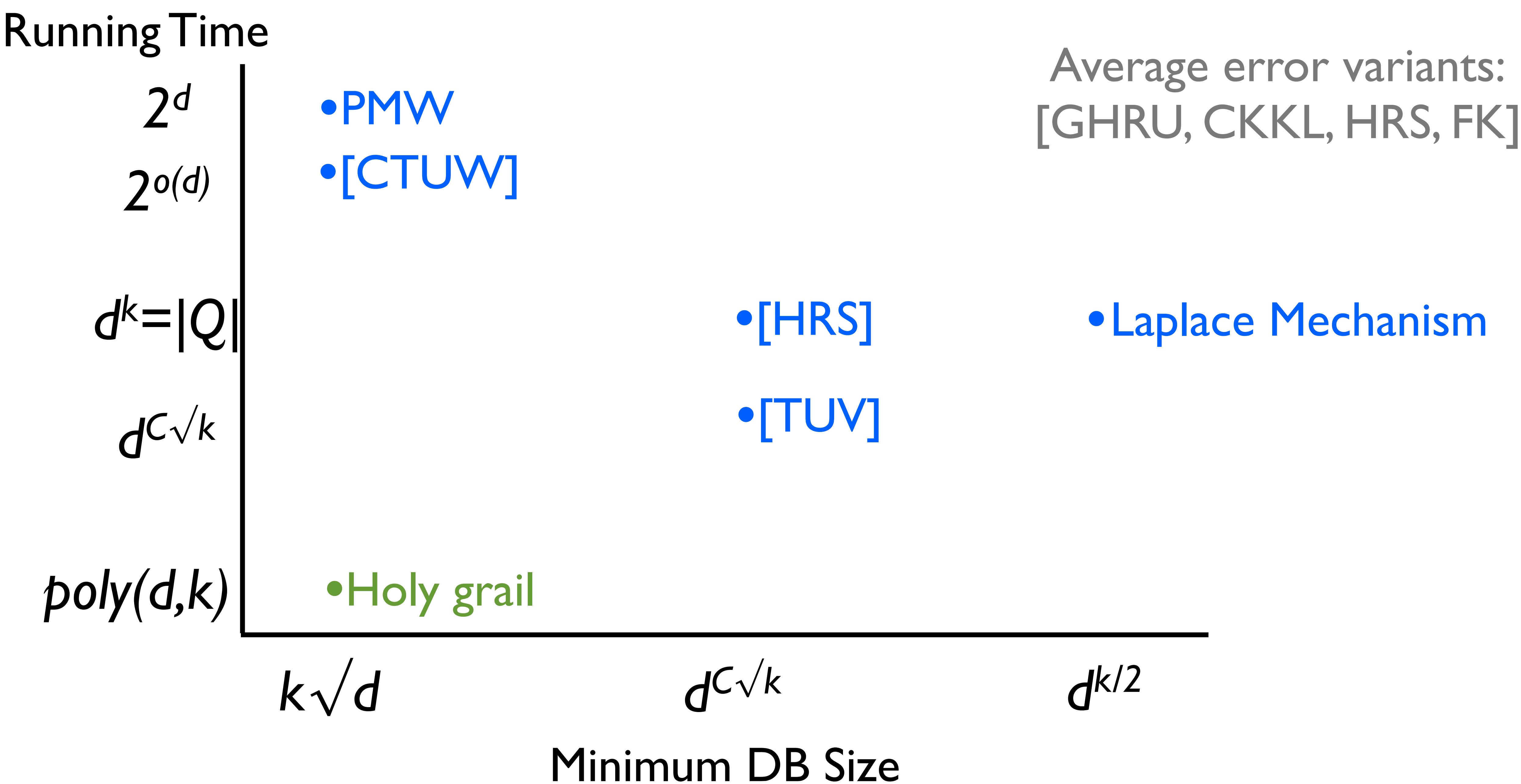
Approximating OR (Low Weight)

- Have an approximation with degree $C\sqrt{k}$ and weight $d^{C\sqrt{k}}$
- The "trivial" exact polynomial has degree d and weight 1



Final polynomial has degree $\sim d^{1-1/C'\sqrt{k}}$, weight $\sim d^{.01}$

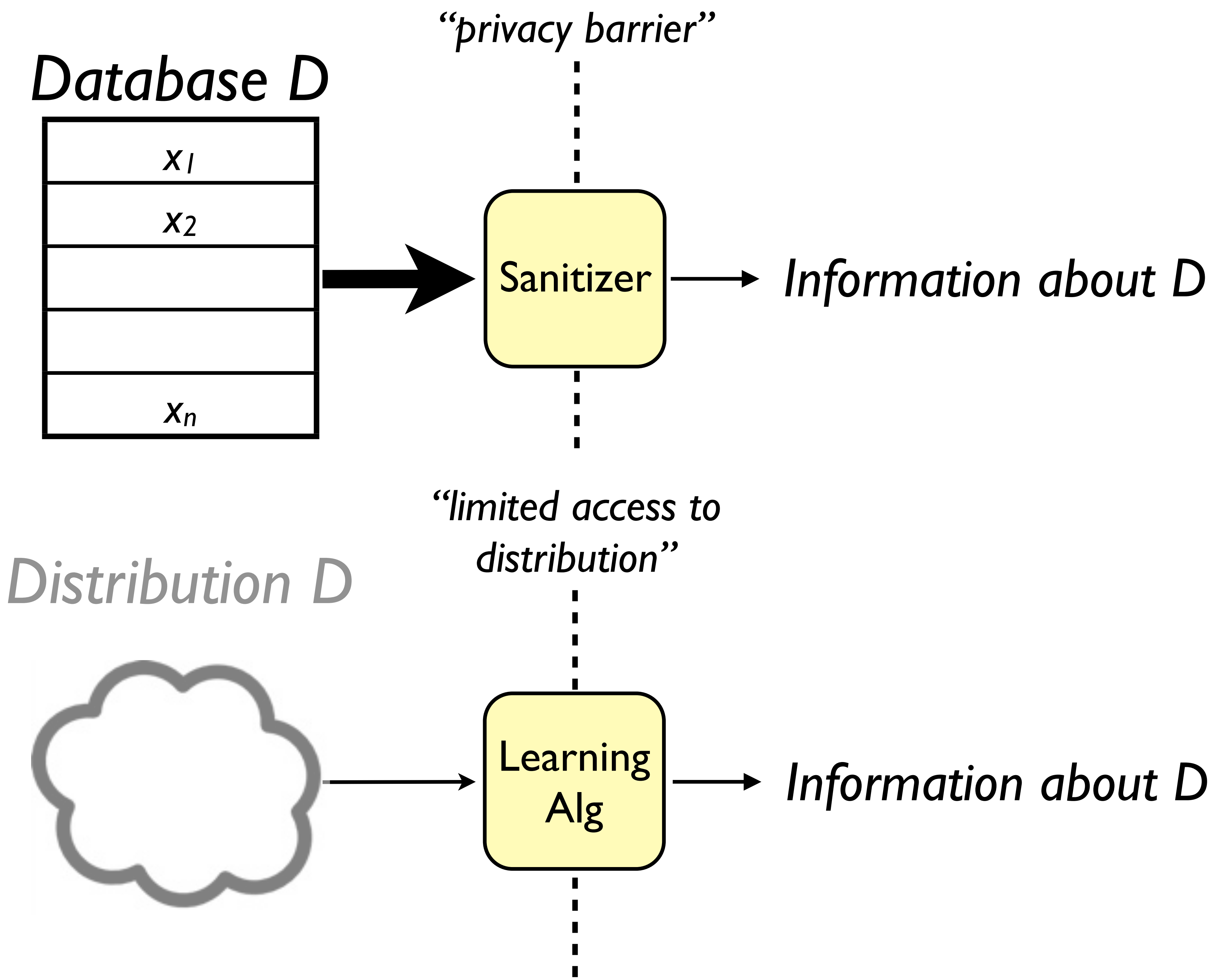
Algorithms for Disjunctions



Can these results be improved?

- Not using polynomials! [CTUW]
- In the high-weight setting, there is no approximate basis smaller than $d^{C\sqrt{k}}$ [S]
- Open question: What is the smallest weight-poly(d) basis wrt to {*k*-way disj}?

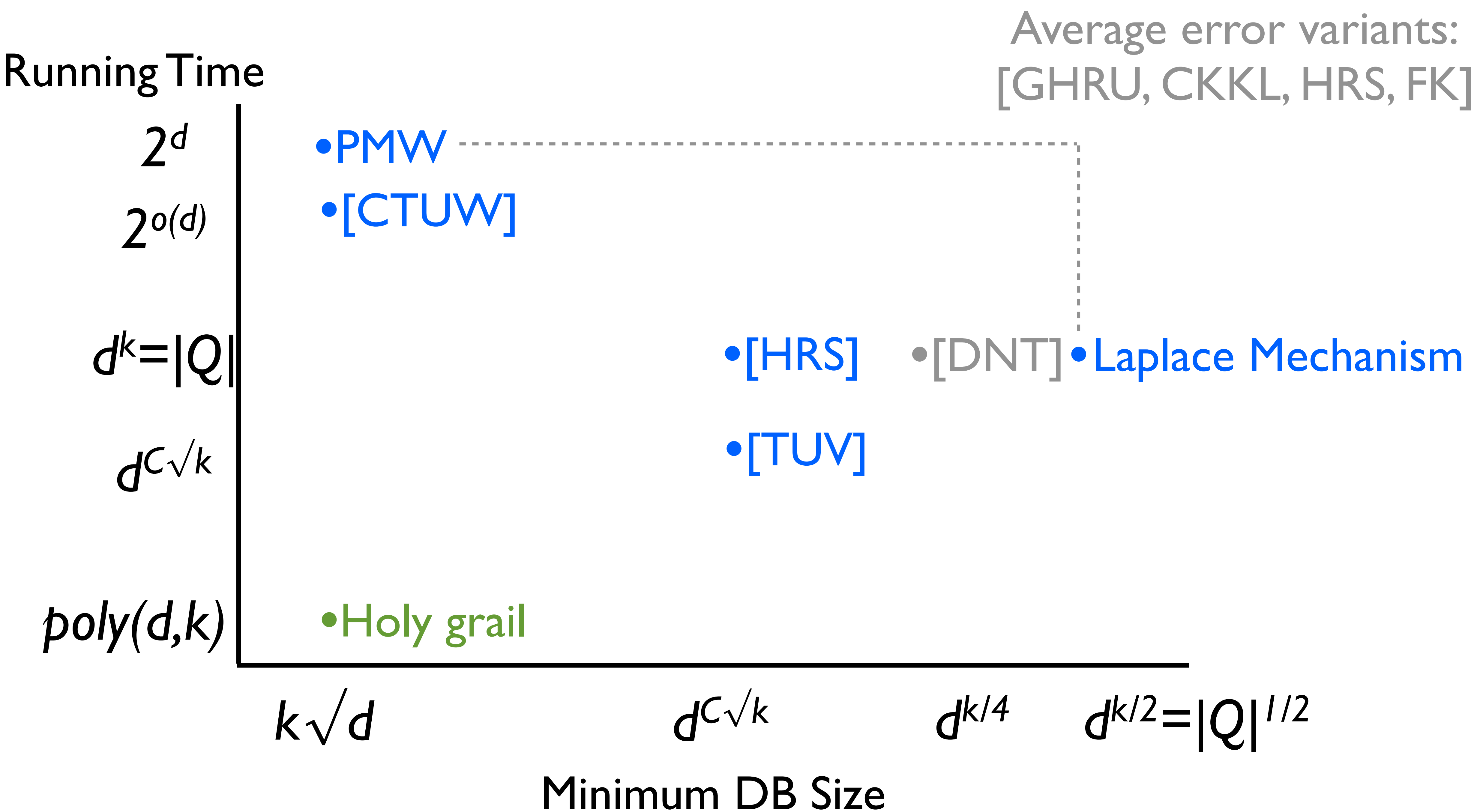
What about using different techniqes?



Can these results be improved?

- Sometimes we can improve running time by avoiding learning algorithms altogether.

Algorithms for Disjunctions



Wrap-Up

- There is a flexible, modular framework for deriving differentially private algorithms from learning-theoretic techniques
- For the general private counting query release problem, these techniques (PMW) give optimal accuracy and running time guarantees
- For natural, special cases of query release, learning techniques (often) give best-known algorithms
  - But is this the right approach?

Thanks!