Approximate Query Processing

CSE 4/562: Database Systems | Lecture 13

Query rewrites are **correct** when they replace one part of the plan with another **guaranteed to compute the same result**

A little bit of error is ok sometimes...

Error-Tolerant Use Cases

- Cardinality estimation for join order selection
- Cloud resource provisioning
- Business intelligence dashboards
- Spam Filtering
- Network Monitoring
- Machine Learning (Bayes Nets, Neural Nets)

Sketches

```
SELECT approx_count_distinct(A), approx_top_k(B)
FROM BIG_TABLE
```

Lossy state encodings for “hard” (holistic) aggregates.

Trading accuracy for speed...


```
SELECT listagg(A)
FROM R
```

$$[3, 6, 10, 9, 1, 3, 9, 7, 9, 4, 7, 9, 2, 1, 2, 4, 10, 8, 9, 7]$$

Trading accuracy for speed...


```
SELECT listagg(A)
FROM R
```

$$[3, 6, 10, 9, 1, 3, 9, 7, 9, 4, 7, 9, 2, 1, 2, 4, 10, 8, 9, 7]$$

—

```
SELECT avg(A)
FROM R
```

$$6$$

Trading accuracy for speed...


```
SELECT listagg(A)
FROM R
```

$[3, 6, 10, 9, 1, 3, 9, 7, 9, 4, 7, 9, 2, 1, 2, 4, 10, 8, 9, 7]$

—

```
SELECT avg(A)
FROM R
```

6

—

```
SELECT avg(A), listagg(",", A)
FROM R
WHERE rand() < .25
```

$5.8, [3, 6, 10, 9, 1]$

Samples

- $\text{avg}(\text{sample}(A, \text{rate})) \approx \text{avg}(A)$
- $\frac{\text{sum}(\text{sample}(A,\ \text{rate}))}{\text{rate}} \approx \text{sum}(A)$

Sampling lets you approximate aggregate values with less data

Did I generate enough samples?

Estimation error

- "The correct answer is definitely in $[a, b]$"
  - Maximum safety
  - Not always possible without seeing all the data
- "I'm at least 95% sure the answer is in $[a, b]$"
  - Viable for sampling

"I'm at least $100 \cdot (1 - \delta)\%$ sure the answer is within ε."

Hoeffding's Bound

With S being n tuples sampled uniformly *with* replacement from any R:

Hoeffding's Bound

With S being n tuples sampled uniformly *with* replacement from any R:

$$|\text{avg}(S) - \text{avg}(R)| \quad \text{The absolute error}$$

Hoeffding's Bound

With S being n tuples sampled uniformly *with* replacement from any R:

$\lvert\text{avg}(S) - \text{avg}(R)\rvert$	The absolute error
$P(\lvert\text{avg}(S) - \text{avg}(R)\rvert \geq \varepsilon)$	The probability of exceeding error threshold ε

Hoeffding's Bound

With S being n tuples sampled uniformly *with* replacement from any R:

$$|\text{avg}(S) - \text{avg}(R)|$$

The absolute error

$$P(|\text{avg}(S) - \text{avg}(R)| \geq \varepsilon)$$

The probability of exceeding error threshold ε

$$P(|\text{avg}(S) - \text{avg}(R)| \geq \varepsilon) \leq 2e^{\frac{2n\varepsilon^2}{(\max(R) - \min(R))^2}}$$

... is at worst some threshold

Hoeffding's Bound

With S being n tuples sampled uniformly *with* replacement from any R:

$$|\text{avg}(S) - \text{avg}(R)|$$

The absolute error

$$P(|\text{avg}(S) - \text{avg}(R)| \geq \varepsilon)$$

The probability of exceeding error threshold ε

$$P(|\text{avg}(S) - \text{avg}(R)| \geq \varepsilon) \leq 2e^{\frac{2n\varepsilon^2}{(\max(R)-\min(R))^2}}$$

... is at worst some threshold

"Hoeffding's Bound" (Generalizable to sum, count, stddev, etc...)

Hoeffding's Bound

With S being n tuples sampled uniformly *with* replacement from any R:

$$|\text{avg}(S) - \text{avg}(R)|$$ The absolute error

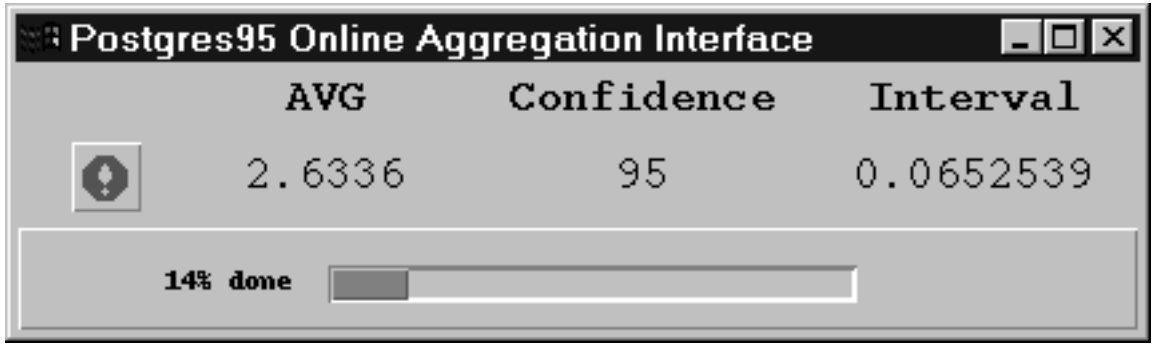
$$P(|\text{avg}(S) - \text{avg}(R)| \geq \varepsilon)$$ The probability of exceeding error threshold ε

$$P(|\text{avg}(S) - \text{avg}(R)| \geq \varepsilon) \leq 2e^{\frac{2n\varepsilon^2}{(\max(R) - \min(R))^2}}$$ ... is at worst some threshold

"Hoeffding's Bound" (Generalizable to sum, count, stddev, etc...)

See also "Chernoff's Bound", "Serfling's Bound" and others.

How do we use it?



Interfaces

- Keep adding samples until you reach a target accuracy
- Keep adding samples until you run out of time
- Keep adding samples until the user tells you to stop

Image credit: 'Online Aggregation', Hellerstein et. al.

Generating samples.

What is hard about generating samples?

What is hard?

Sampling From Disk

- Random seeks are slow

Sampling for Group By or Filtering

- Low frequency events don’t get sampled

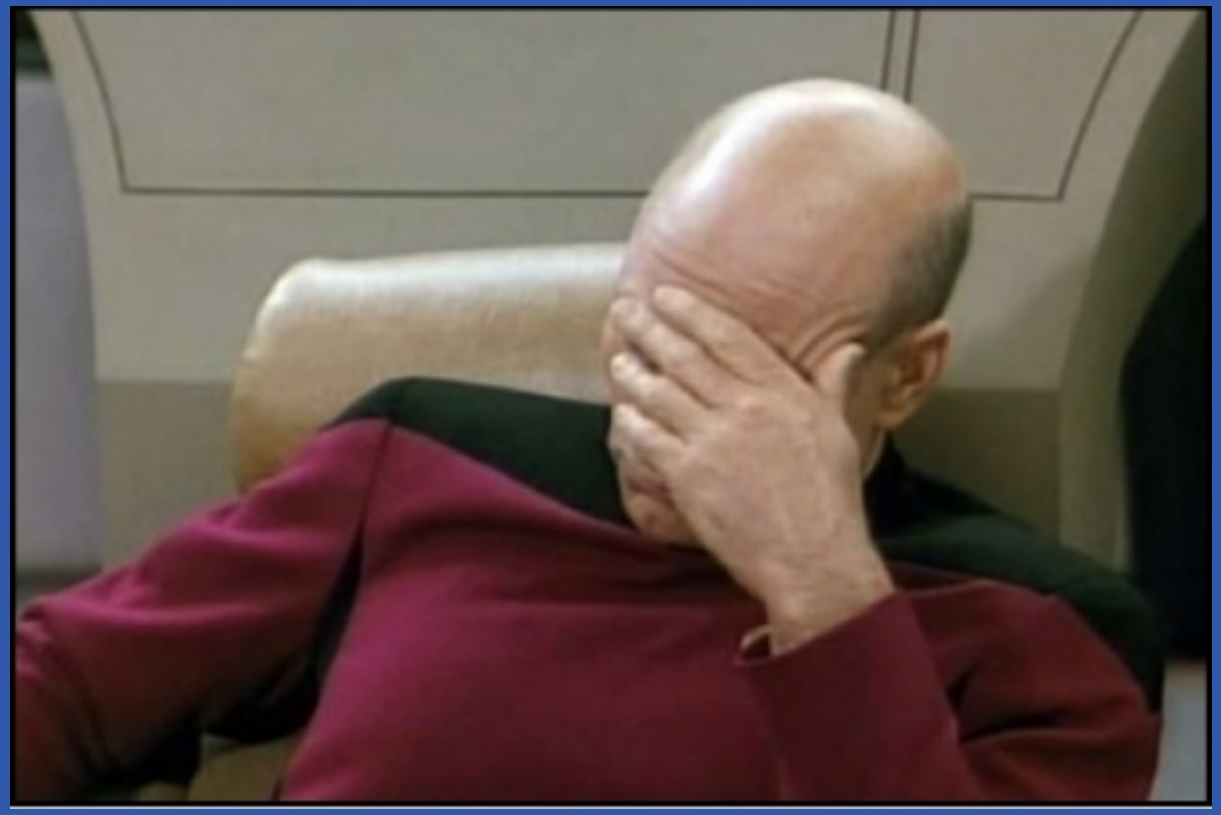
Sampling from Joins

- The Birthday Paradox

Sampling from disk

Idea 1: Pick randomly!

```
for i in range(num_samples):
      sample_id = random(0, num_records)
      samples += [ table.where( rowid = sample_id ) ]
```



Problem: Random reads have low spatial locality.

Idea 2: Sequential Scan.

```
sample_ids = [
  random(0, num_records)
  for i in range(num_samples)
]
samples = table.join(sample_ids)
```

Problem: This requires knowing how many records there are.

Idea 3: Reservoir Sampling.

```
i = 0
reservoir = []
for row in table:
  if len(reservoir) < TARGET:
    reservoir += row
  else if rand() < TARGET / len(reservoir):
    reservoir[rand_int() % TARGET] = row
```

Sampling from Group-By

Group-By Sampling

Employee	City	Salary
Alice	NYC	$120k
Bob	NYC	$110k
Carol	NYC	$115k
Dave	Syracuse	$80k

```
SELECT City, AVG(Salary) FROM NYS_Salaries;
```

Group-By Sampling

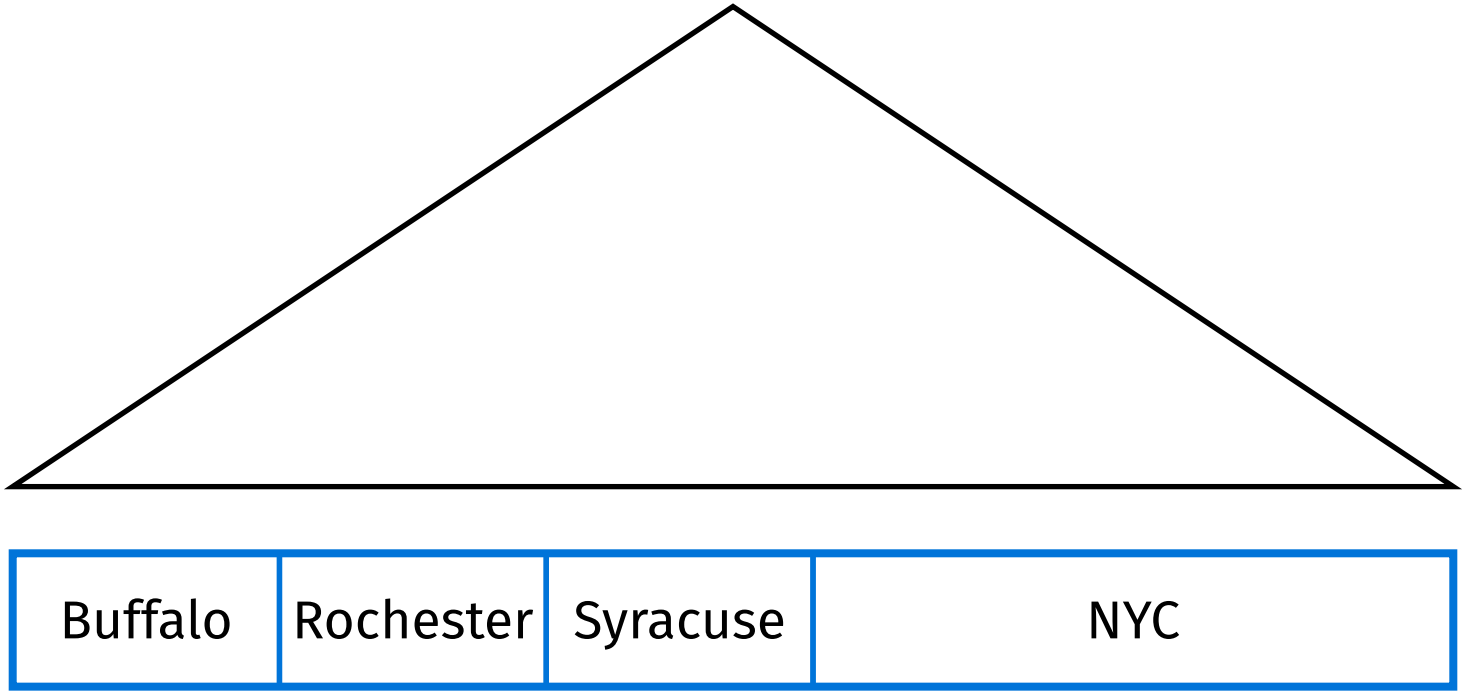
Problem: Most data is about NYC. With N samples taken uniformly, margins of error for other cities are much bigger.

Stratified Sampling

Generate $\frac{N}{\texttt{COUNT(DISTINCT City)}}$ samples for each group

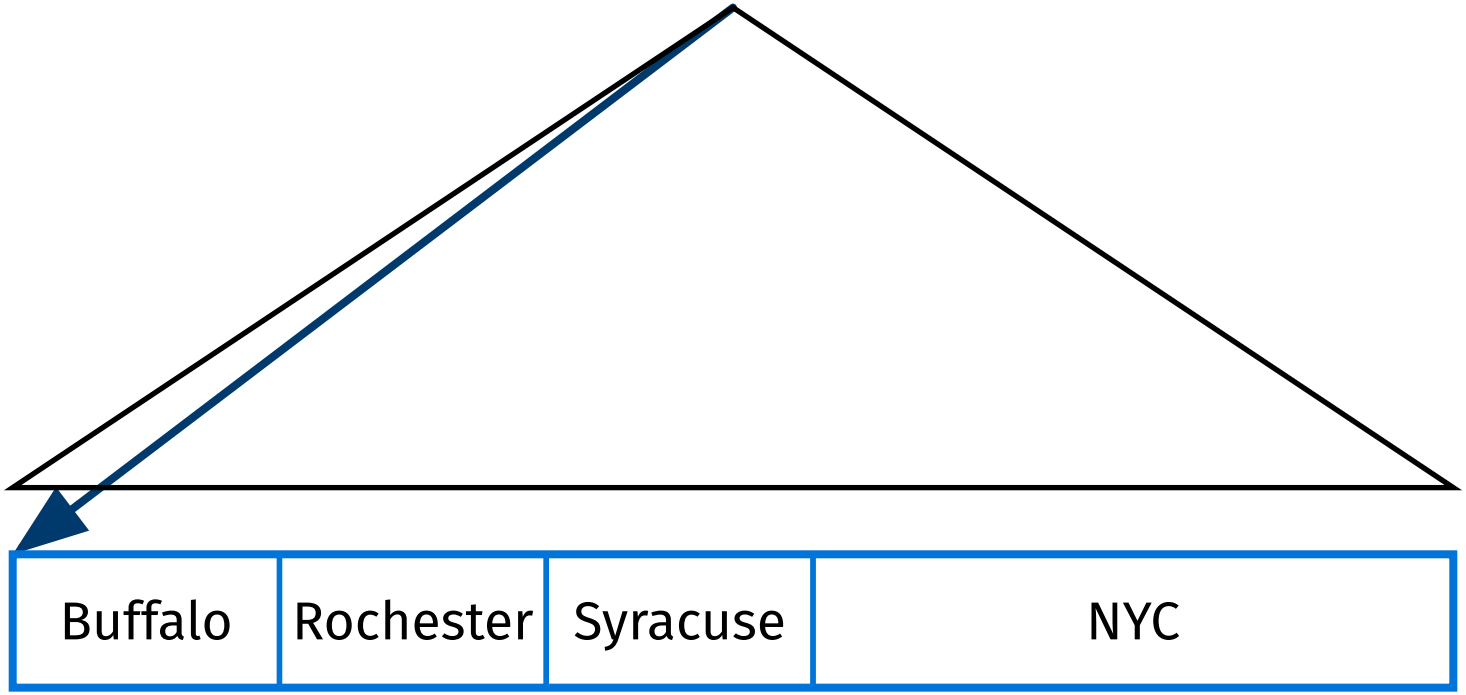
Use $\texttt{COUNT(DISTINCT City)}$ instead of $\texttt{COUNT}(*)$ as the total group size

Index Striding



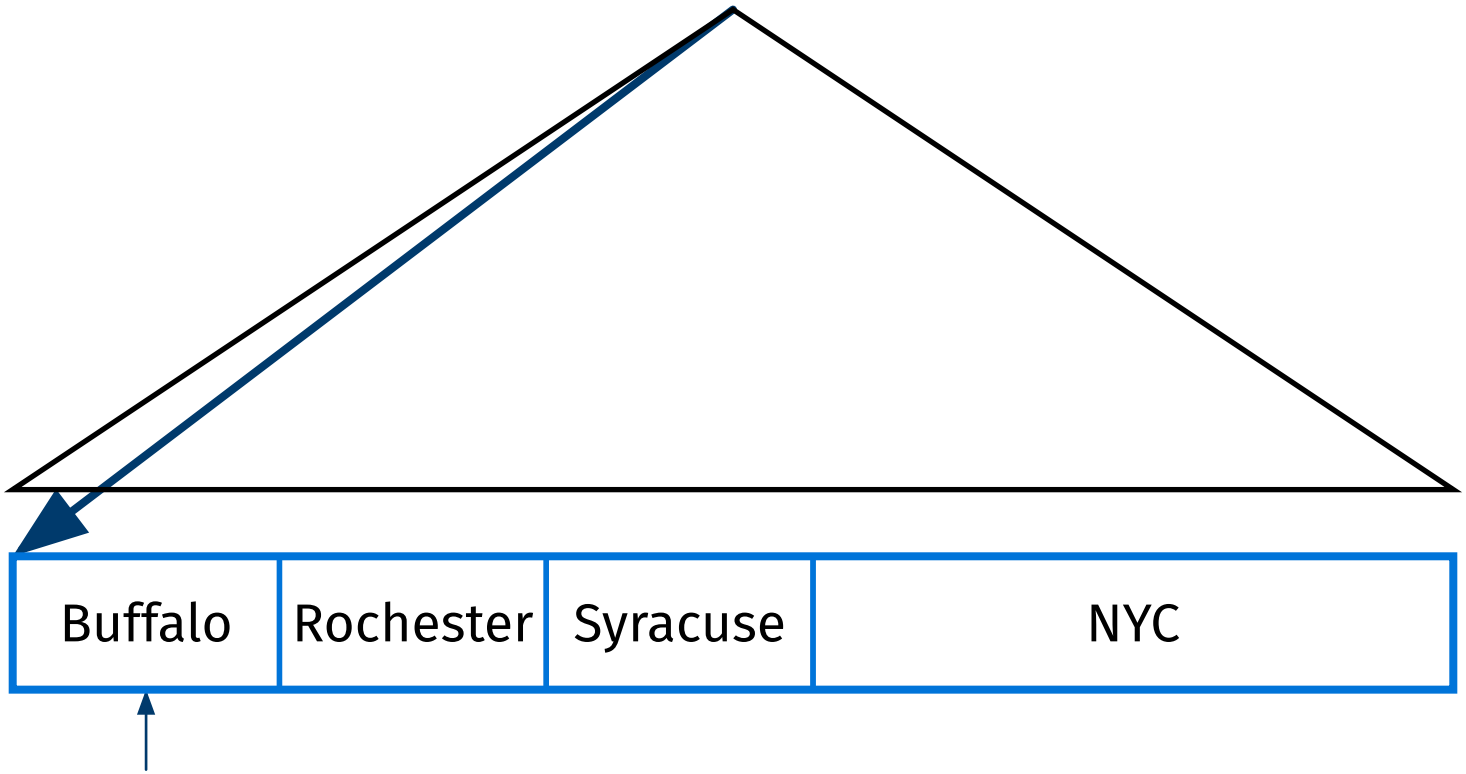
'Online Aggregation', Hellerstein et. al.

Index Striding



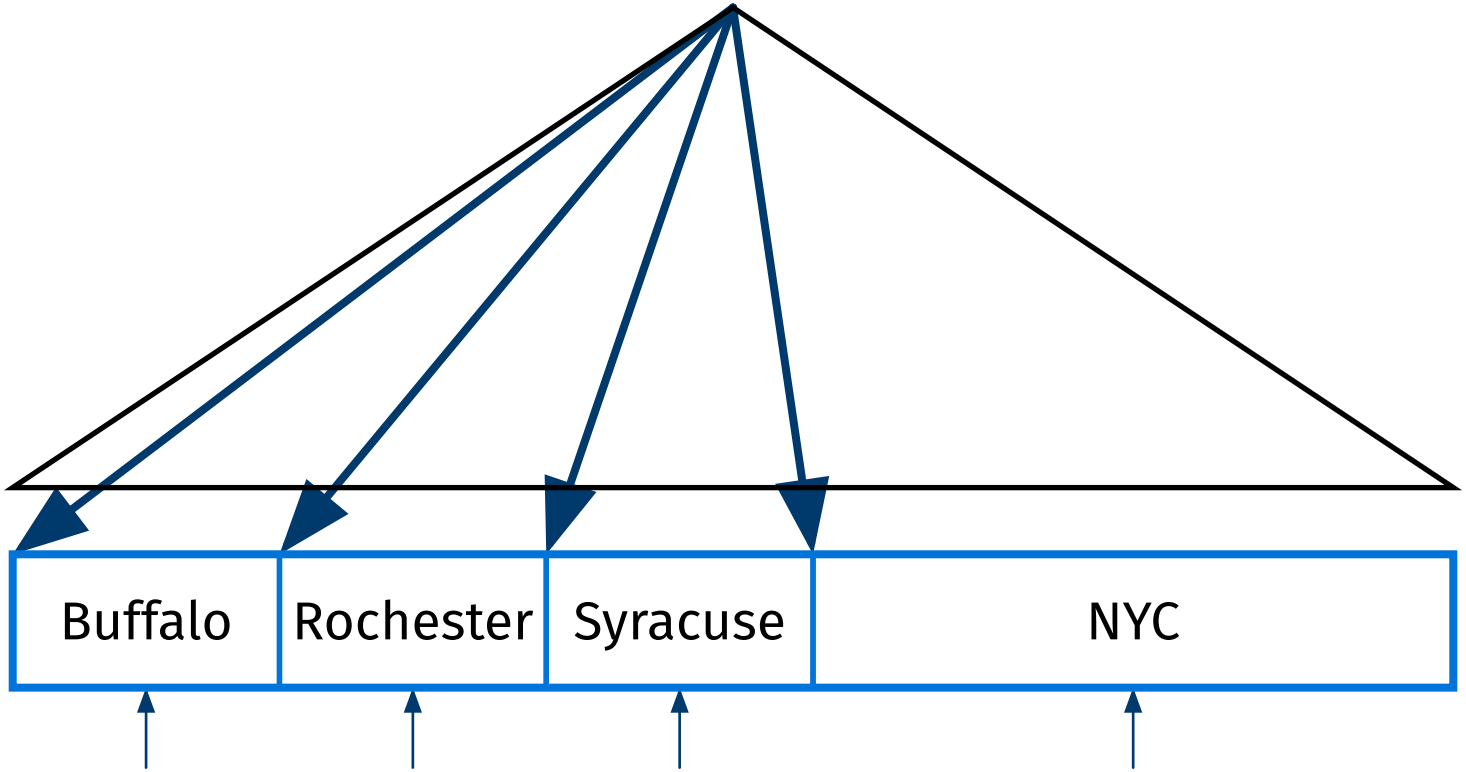
'Online Aggregation', Hellerstein et. al.

Index Striding



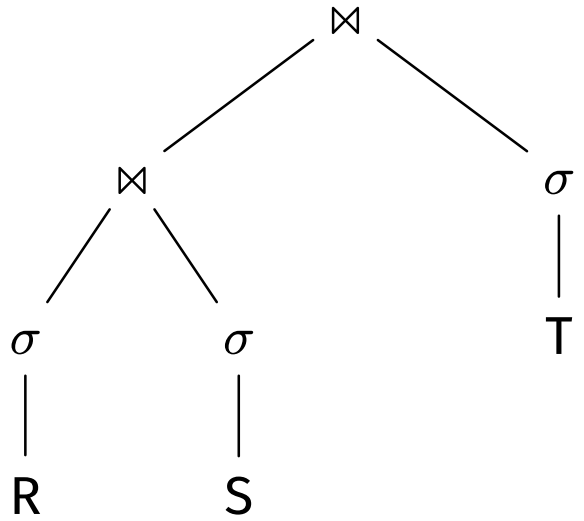
'Online Aggregation', Hellerstein et. al.

Index Striding

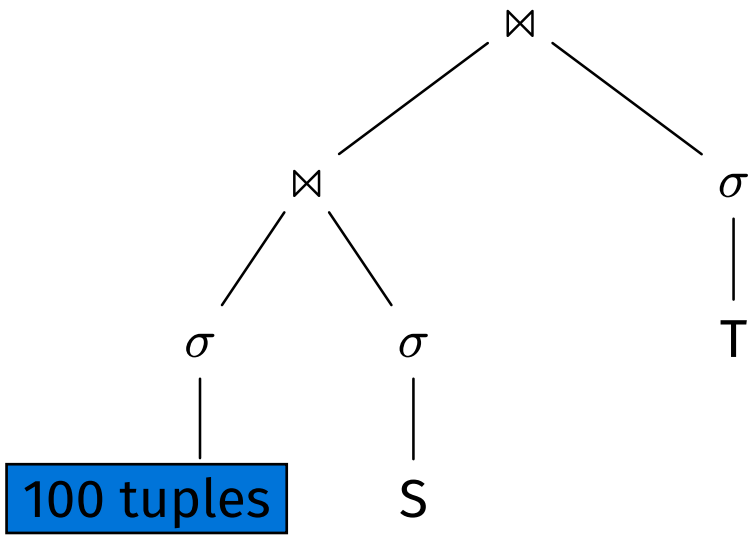


'Online Aggregation', Hellerstein et. al.

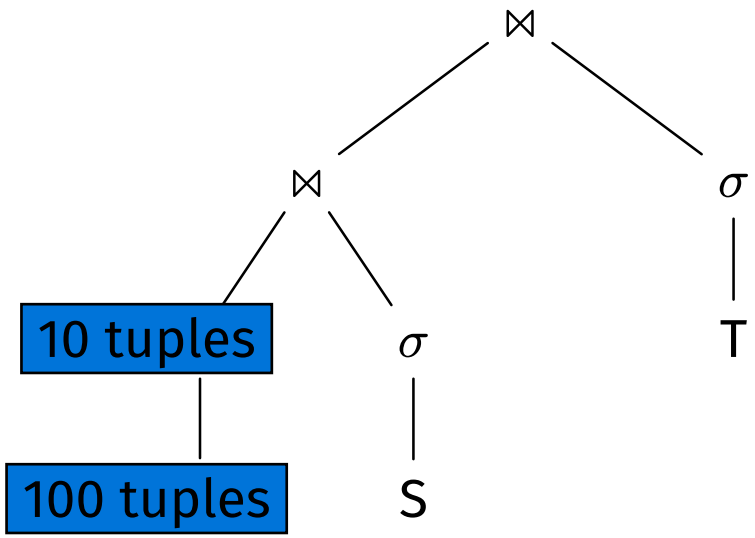
Sampling from Joins


⋈
⋈
σ
σ
σ
T
R
S

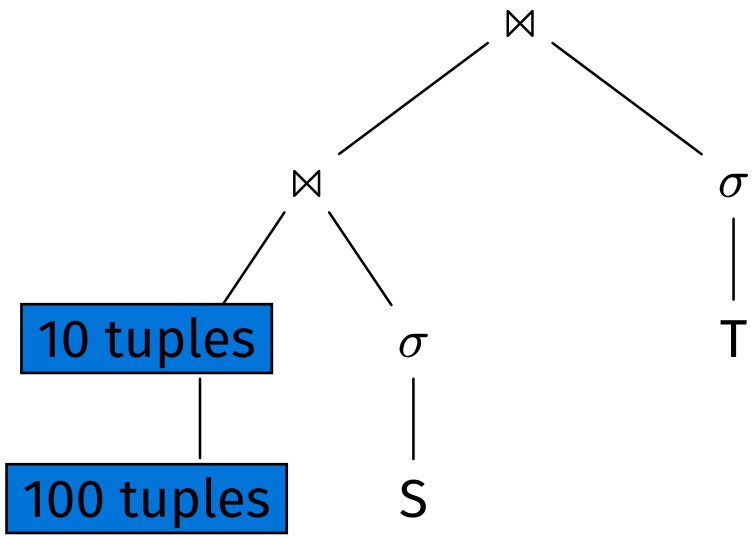
The Birthday Paradox



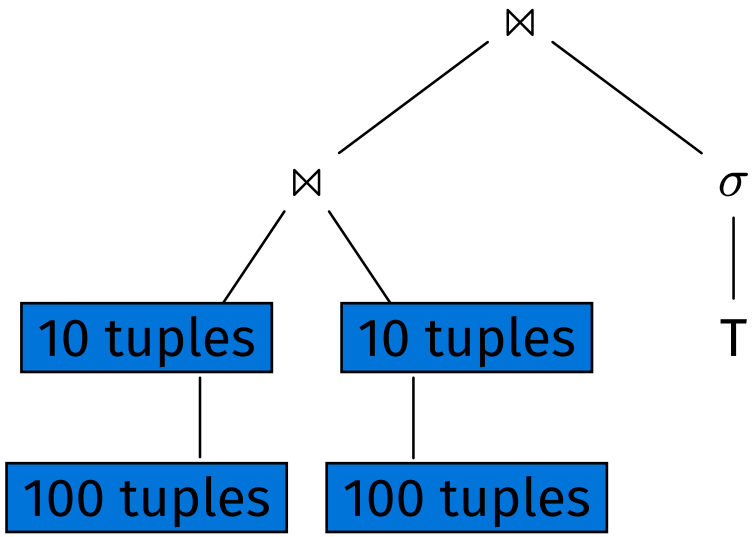
The Birthday Paradox



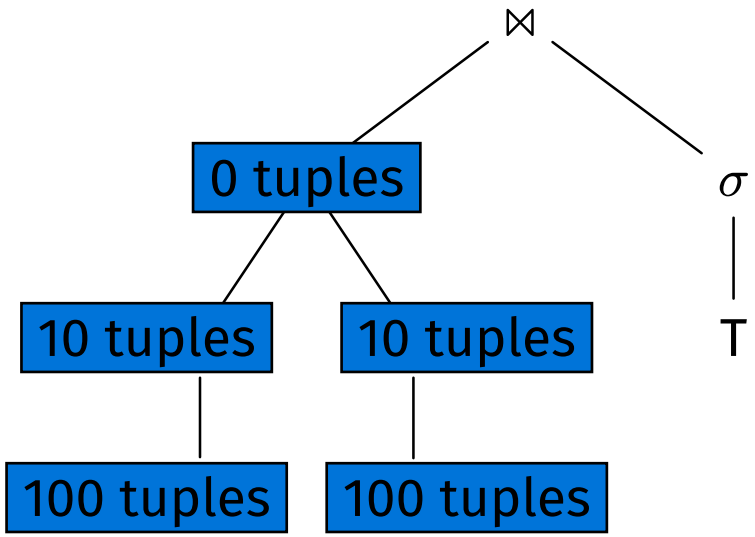
The Birthday Paradox

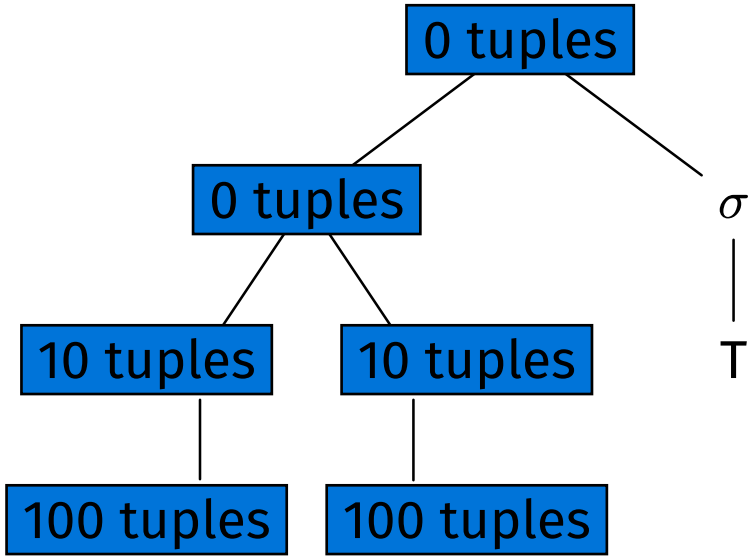


The Birthday Paradox



The Birthday Paradox




0 tuples
0 tuples
σ
10 tuples
10 tuples
T
100 tuples
100 tuples

Assume: `UNIQ(A,R) = UNIQ(A,S) =` N

The Birthday Paradox

It takes $O\left(\sqrt{N}\right)$ samples from `R` and `S` each to expect to get even one match.

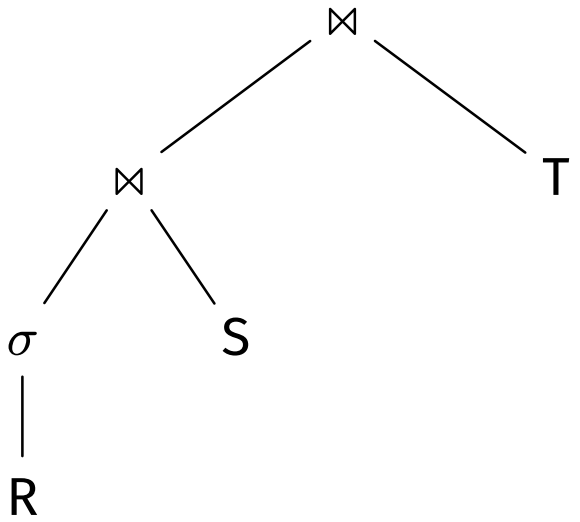
Join Synopses

Given $R \underset{B}{\bowtie} S \underset{C}{\bowtie} T$ where

- $R.B$ is a foreign key reference to $S.B$
- $S.C$ is a foreign key reference to $T.C$

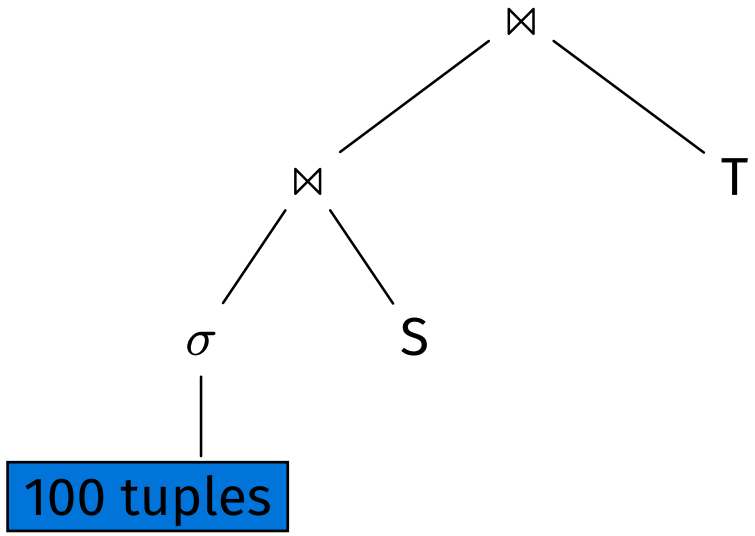
Idea: Sample from R, use all of S, T

Join Synopses



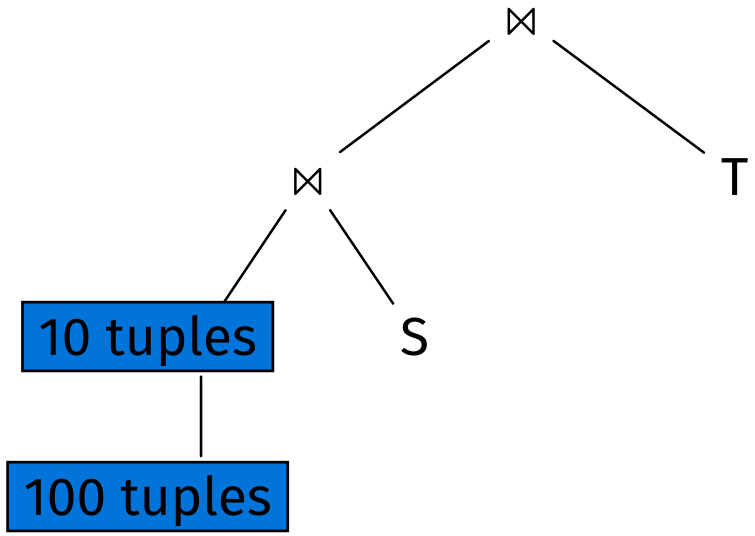
'Join synopses for approximate query answering', Acharya et. al.

Join Synopses



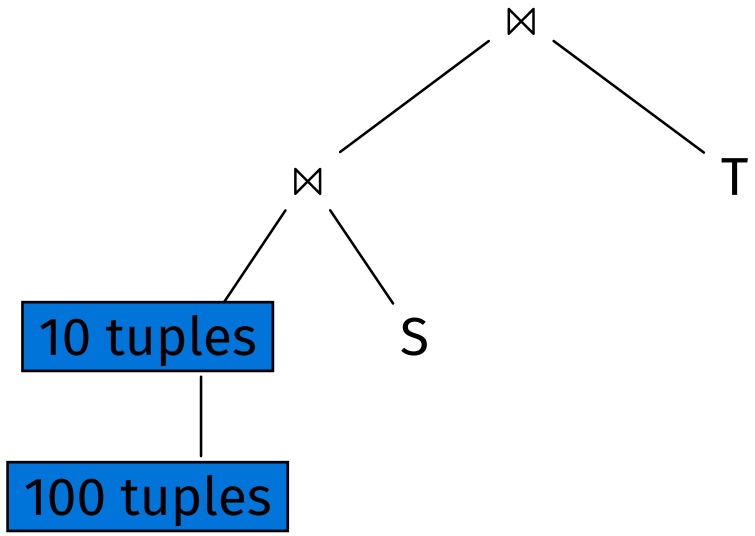
'Join synopses for approximate query answering', Acharya et. al.

Join Synopses



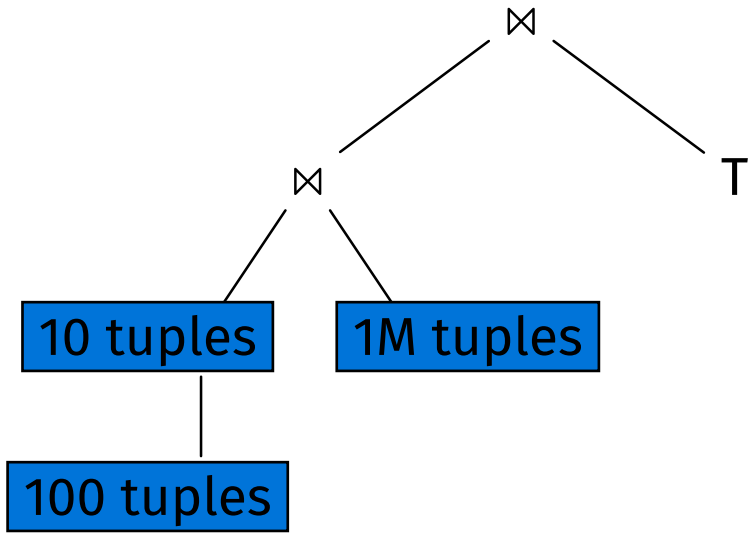
'Join synopses for approximate query answering', Acharya et. al.

Join Synopses



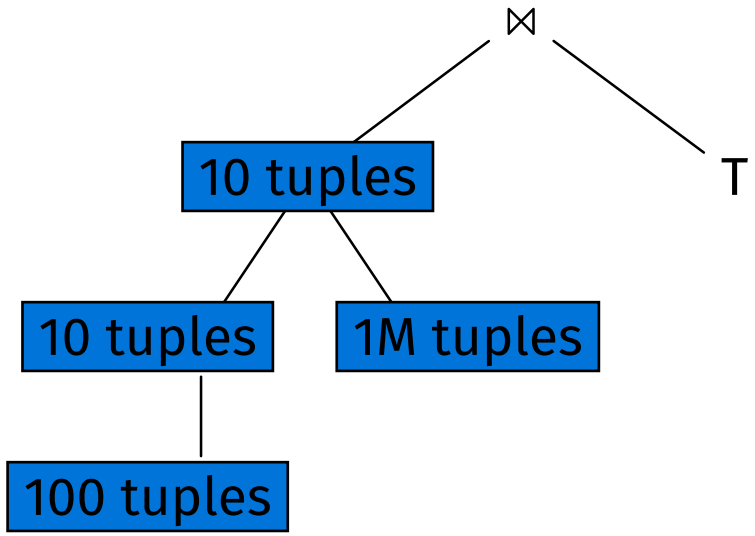
'Join synopses for approximate query answering', Acharya et. al.

Join Synopses



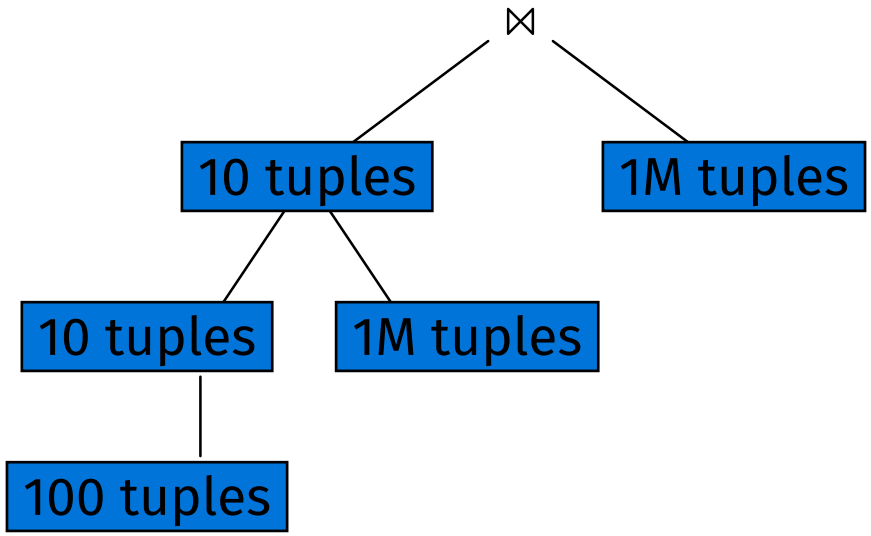
'Join synopses for approximate query answering', Acharya et. al.

Join Synopses



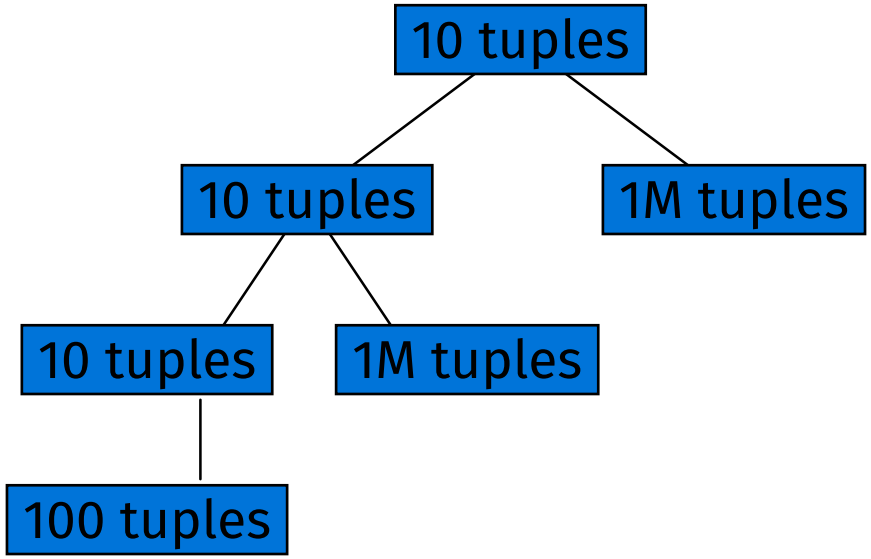
'Join synopses for approximate query answering', Acharya et. al.

Join Synopses



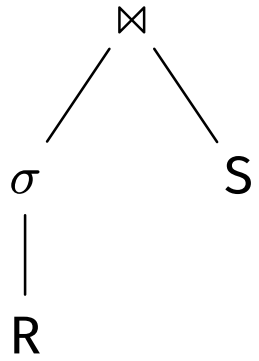
'Join synopses for approximate query answering', Acharya et. al.

Join Synopses

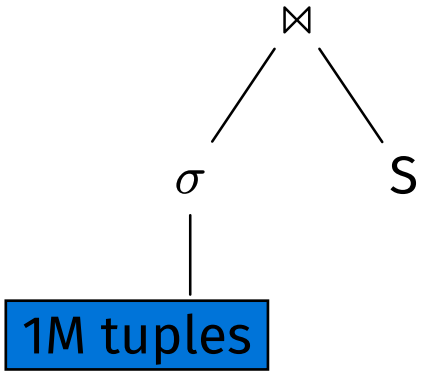


'Join synopses for approximate query answering', Acharya et. al.

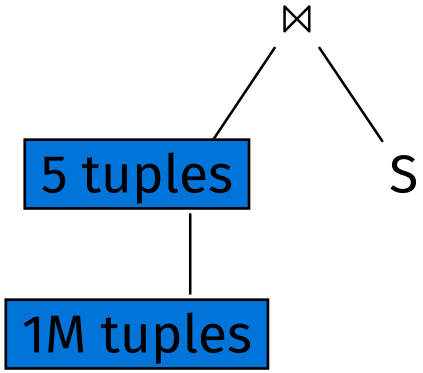
Idea 2: Use stratified sampling



Idea 2: Use stratified sampling

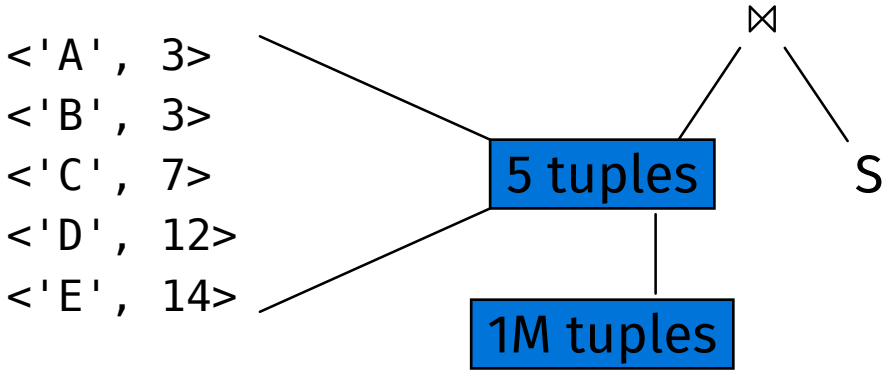


Idea 2: Use stratified sampling



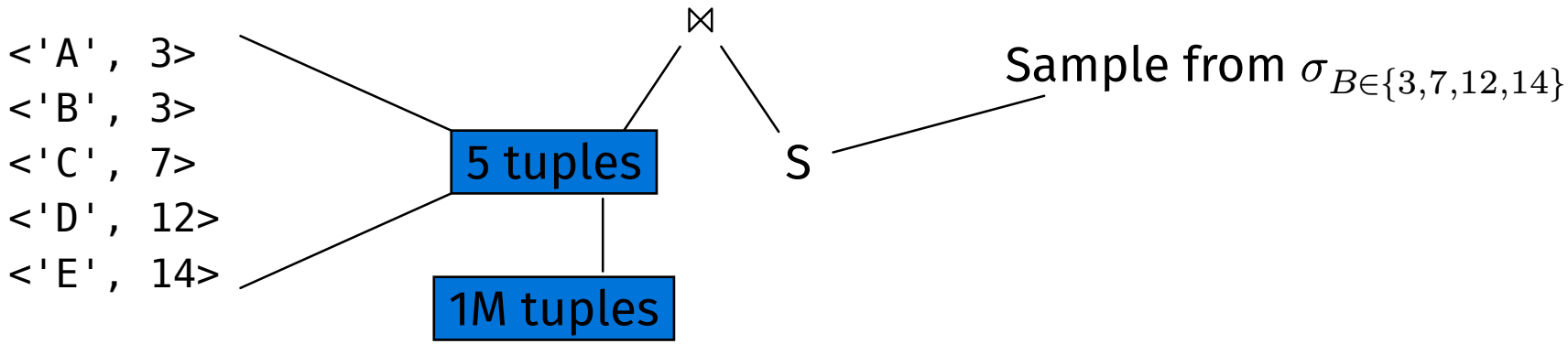
Stratified Sampling

Idea 2: Use stratified sampling



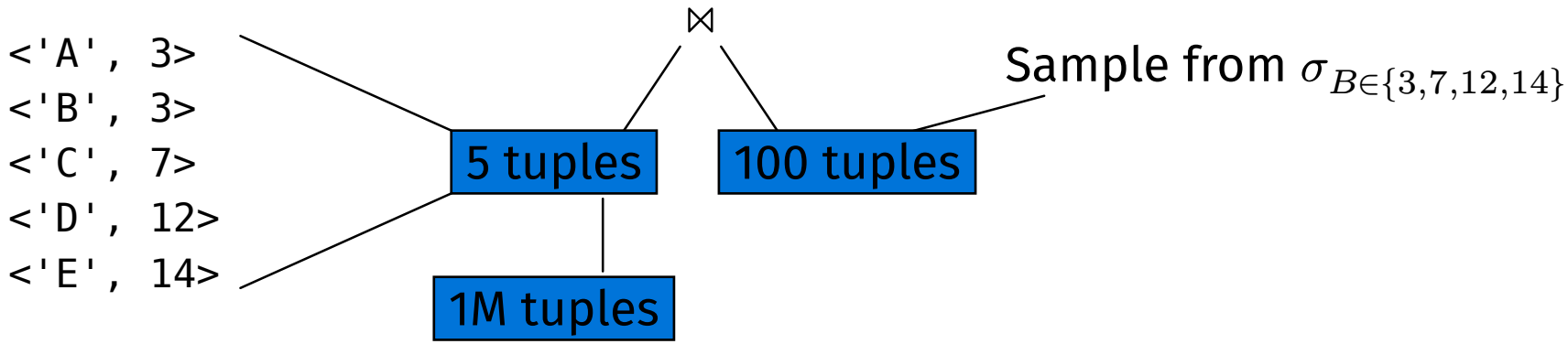
Stratified Sampling

Idea 2: Use stratified sampling



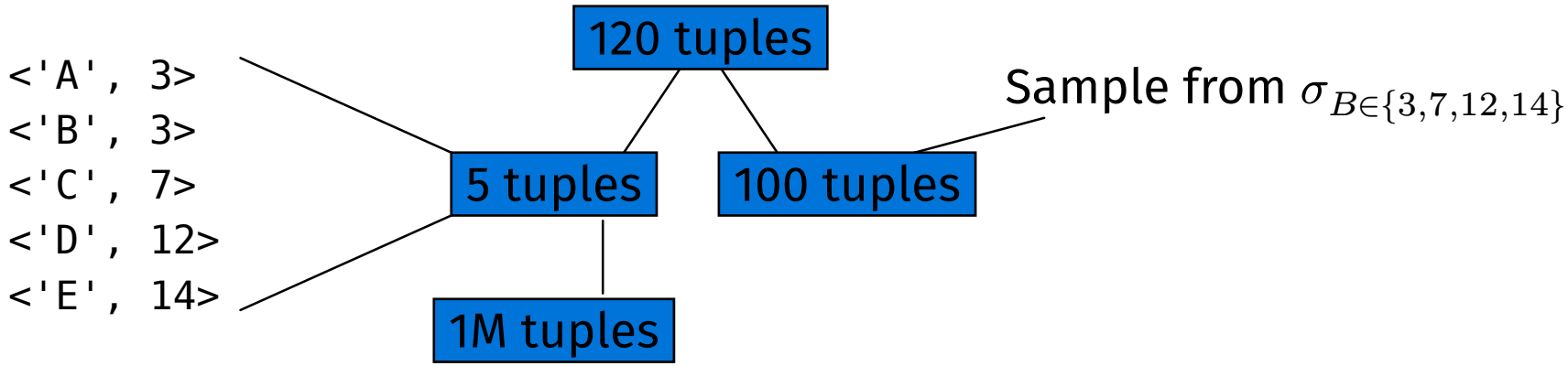
Stratified Sampling

Idea 2: Use stratified sampling



Stratified Sampling

Idea 2: Use stratified sampling



Are we biasing the sampling process?

Goal: For some $r \in R$ and $s \in S$...

Sample $r \bowtie s$ with probability $P((r \bowtie s) \in R \bowtie S)$

—

Actual: Sample a tuple from R, then sample a tuple from S based on R

$P(r \in R) \cdot P((r \bowtie s) \in R \bowtie S \mid r \in R)$

Bayes Theorem

$$P(r \in R) \cdot P((r \bowtie s) \in R \bowtie S \mid r \in R)$$

$$= P(r \in R \wedge (r \bowtie s) \in R \bowtie S)$$

$$\neq P((r \bowtie s) \in R \bowtie S)$$

Corrective Factor

Find a value C such that...

$$P(r \in R \wedge (r \bowtie s) \in R \bowtie S) \cdot C = P((r \bowtie s) \in R \bowtie S)$$

$$C = \frac{P((r \bowtie s) \in R \bowtie S)}{P(r \in R \wedge (r \bowtie s) \in R \bowtie S)}$$

$$= \frac{1}{P(r \in R \mid (r \bowtie s) \in R \bowtie S)} \approx |R| \cdot |\{s \mid r \text{ joins with } s \in S\}|$$

(i.e., the probability that R participates in **some** join)

Sampling Order

Should I sample from R or S first?

- What if R is much bigger than S?
- What if only one tuple of R joins with a tuple from S?
- What if some filtering predicate removes the tuple from S that R joins with?

'Wander Join: Online Aggregation via Random Walks'; Li et. al.

(Ask Zhuoyue Zhao if you're interested)

Ripple Join

- Incrementally increase the sample size

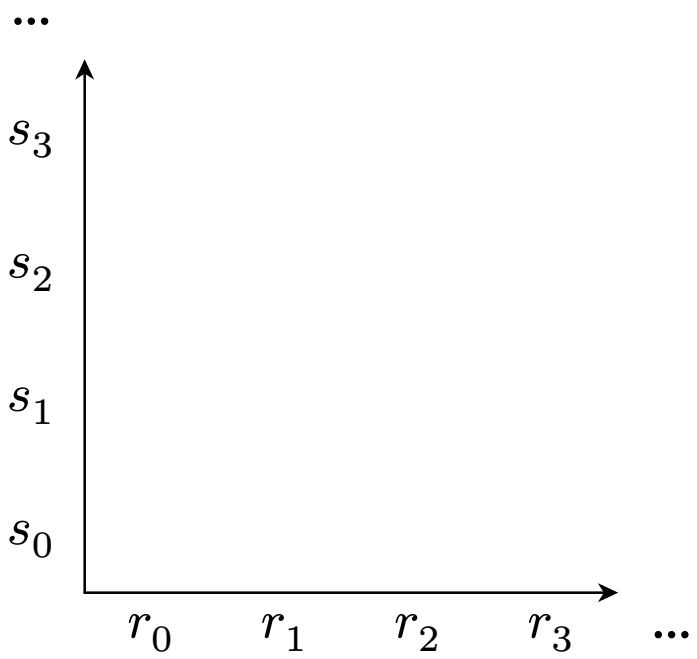
Leaky Joins

- Rotate through the space of possible join results

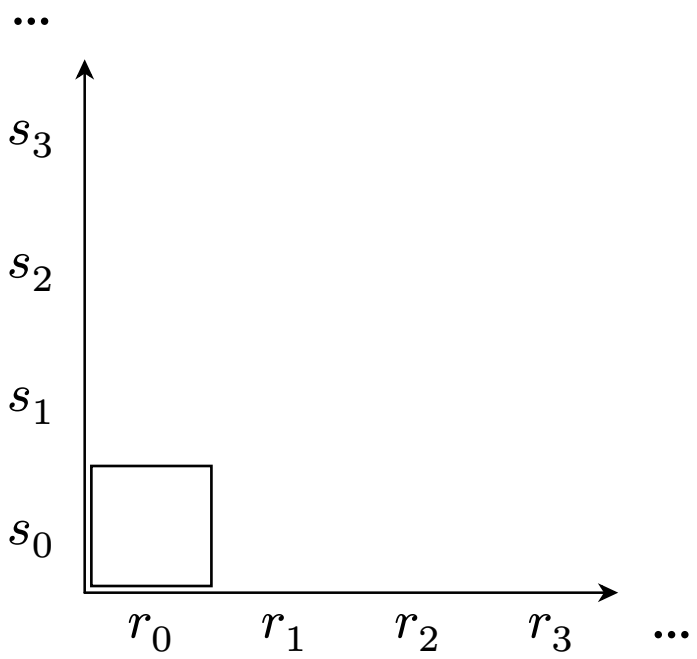
Ripple Join

$$Q = R \bowtie S$$

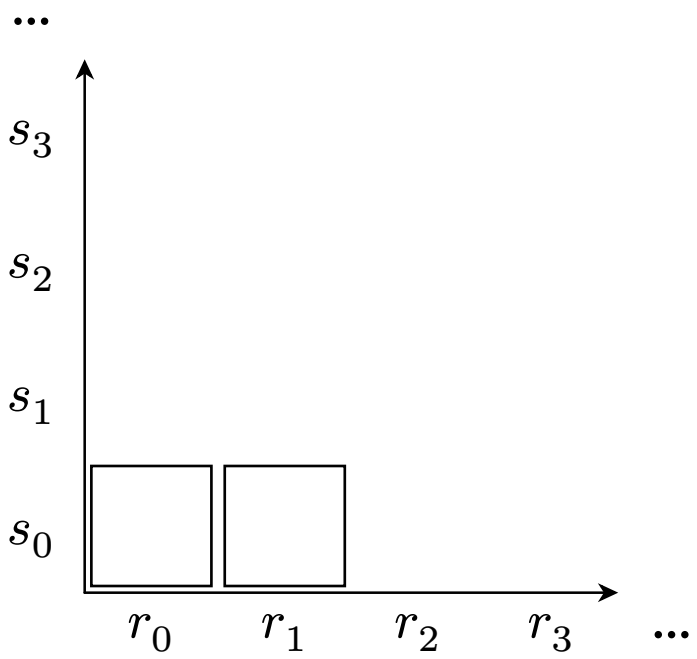
```
Q = []; r_sample = []; s_sample = []

while not r.done() and not s.done()
  if r_current := r.next_sample() is not None:
    Q += [ r_current ] ⋈ s_sample
    r_sample += [ r_current ]

  if s_current := s.next_sample() is not None:
    Q += [ s_current ] ⋈ r_sample
    s_sample += [ s_current ]
```


...
s_3
s_2
s_1
s_0
r_0
r_1
r_2
r_3
...

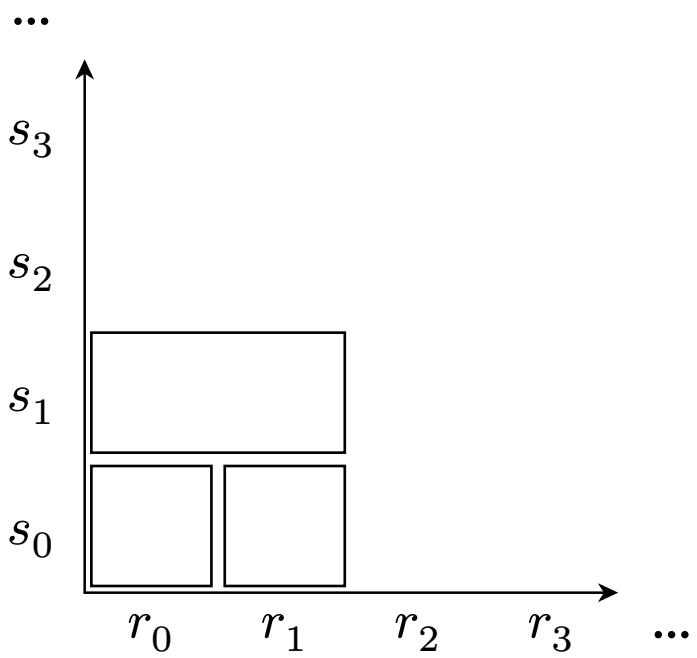
Ripple Join



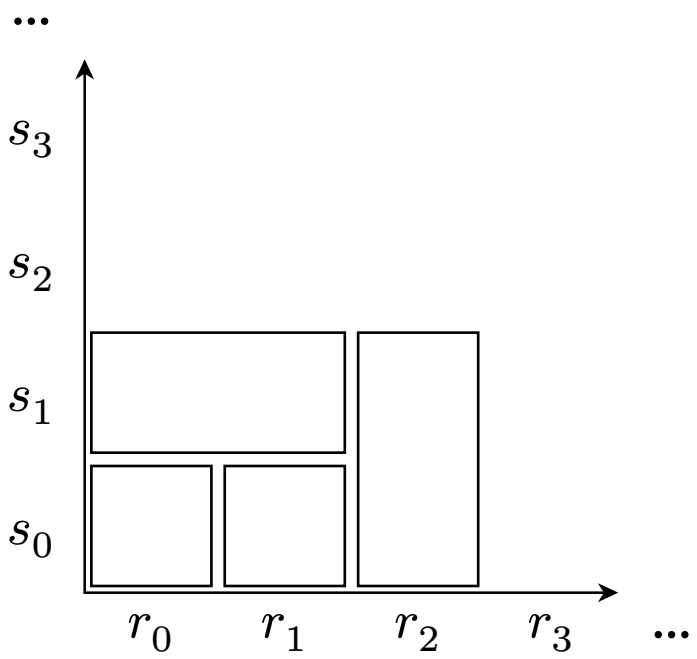
Ripple Join



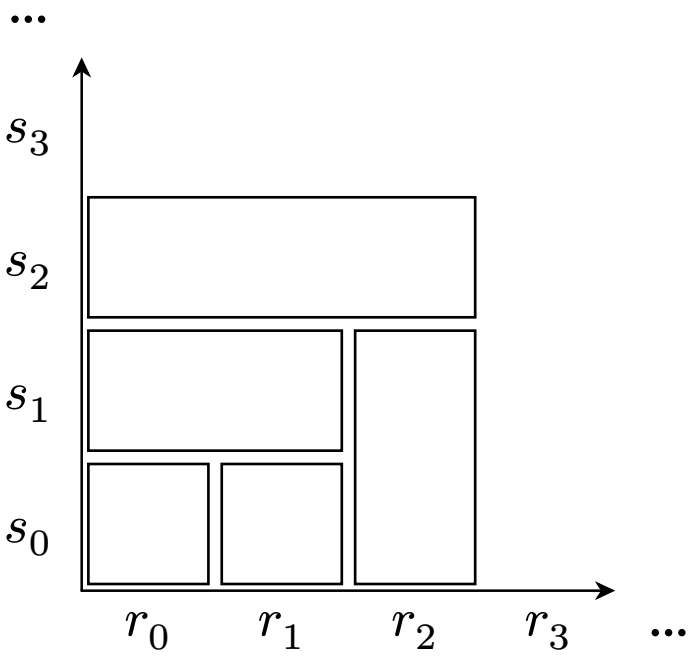
Ripple Join



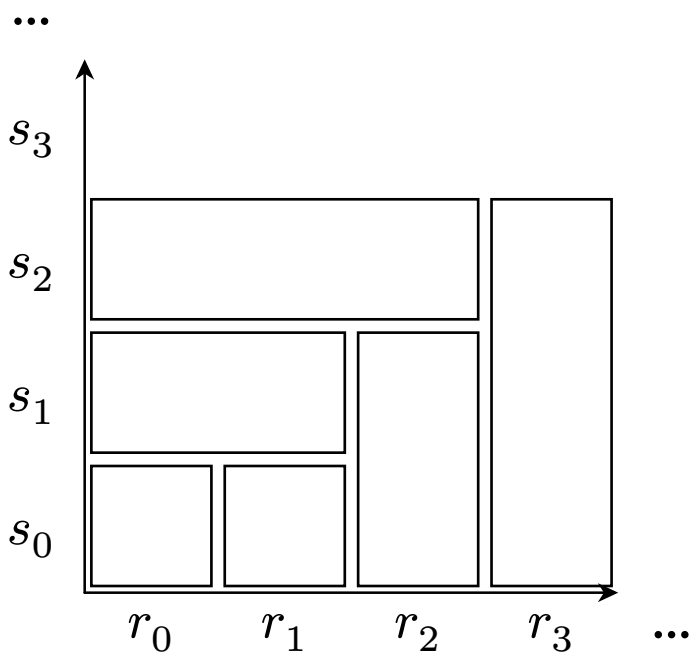
Ripple Join



Ripple Join



Ripple Join



Output-first Sampling

For $Q = \pi_{R.A,R.B,S.C}\left(R \underset{B}{\bowtie} S\right)$

1. Pick a random tuple <a, b, c>
2. Test for <a,b> in R
3. Test for <b,c> in S
4. If both are present, add <a, b, c> to the sample.
5. Repeat as needed

Dense joins.

- Inference in Bayes Nets
- (Dense) Linear Algebra

Output-first Sampling

Problem:

- Sampling without replacement will never converge
- Sampling with replacement means keeping N^k state for k attributes

Linear Congruential Generators

$X_n = (aX_{n-1} + b) \bmod m$ where...

- m and b are relatively prime
- $a - 1$ is divisible by all prime factors of m
- $a - 1$ is divisible by 4 if m is divisible by 4

Then for any $0 \leq X_0 < m$...

$\{X_0, ..., X_{m-1}\}$ is a pseudorandom permutation of [0, m)

The Leaky Join Operator


```
# m = |dom(A1)| * |dom(A2)| * ... * |dom(Ak)|
(a, b) = init_lcg(m)
x = rand_int() % m

for i in range(m):
  x = (a * x + b) % m
  t_left, t_right = extract(x)

  if left.contains(t_left):
    if right.contains(t_right):
      yield t_left + t_right;
```