Free Join

Unifying WCO and Traditional Joins

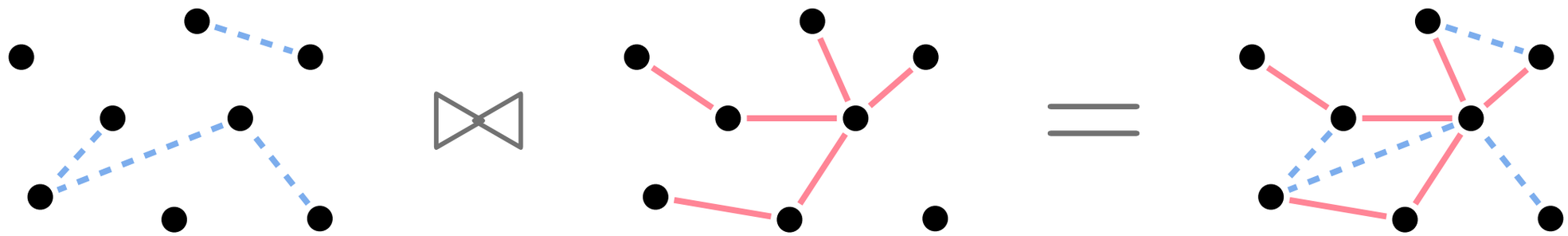
Remy Wang Max Willsey Dan Suciu

University of Washington

May 2023 @ NWDS

✨ Wake-up questions ✨

Join is the **compositional** operator



compose data

Join is the **compositional** operator

$$f \bowtie g = f \circ g$$

compose programs

90% of run time

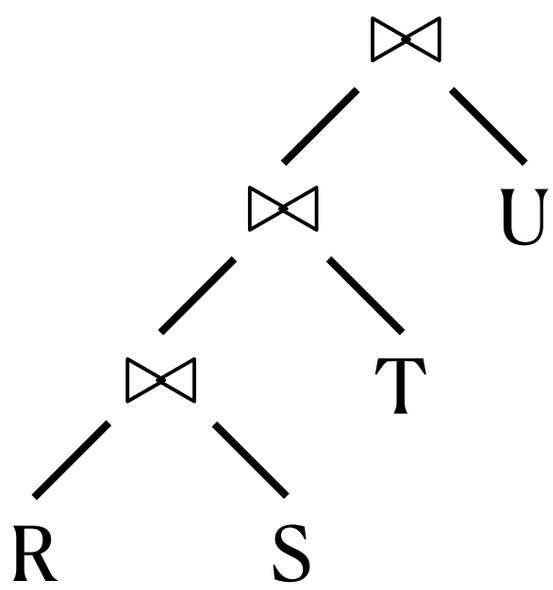
— Leis et.al. VLDB'15

10,000's of papers

A Tale of Two Joins

Binary Join

Codd'70



2 relation at a time

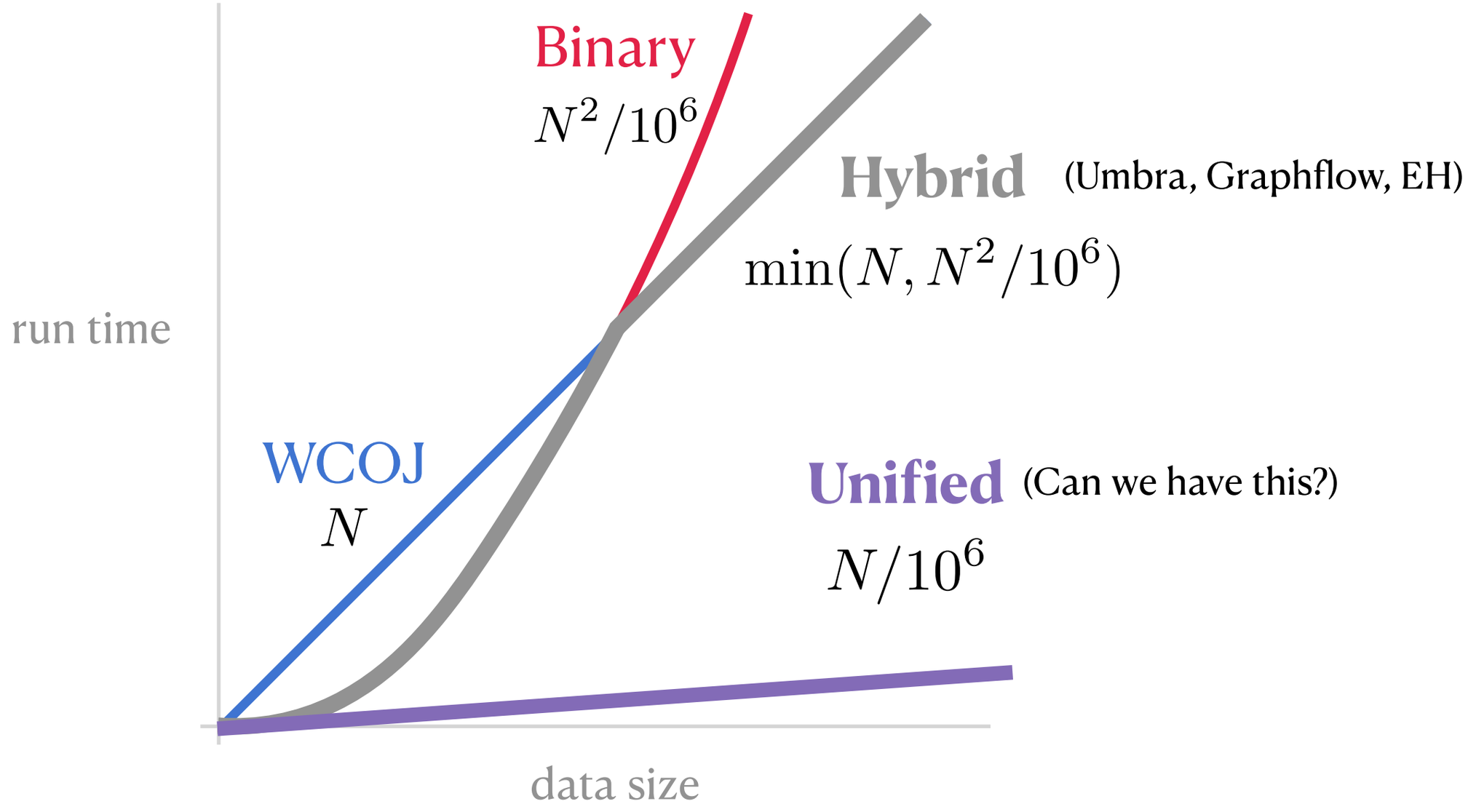
v.s.

Worst-Case Optimal Join

(WCOJ) NNPR'12, V'12

1 attribute at a time

A Tale of Two Joins



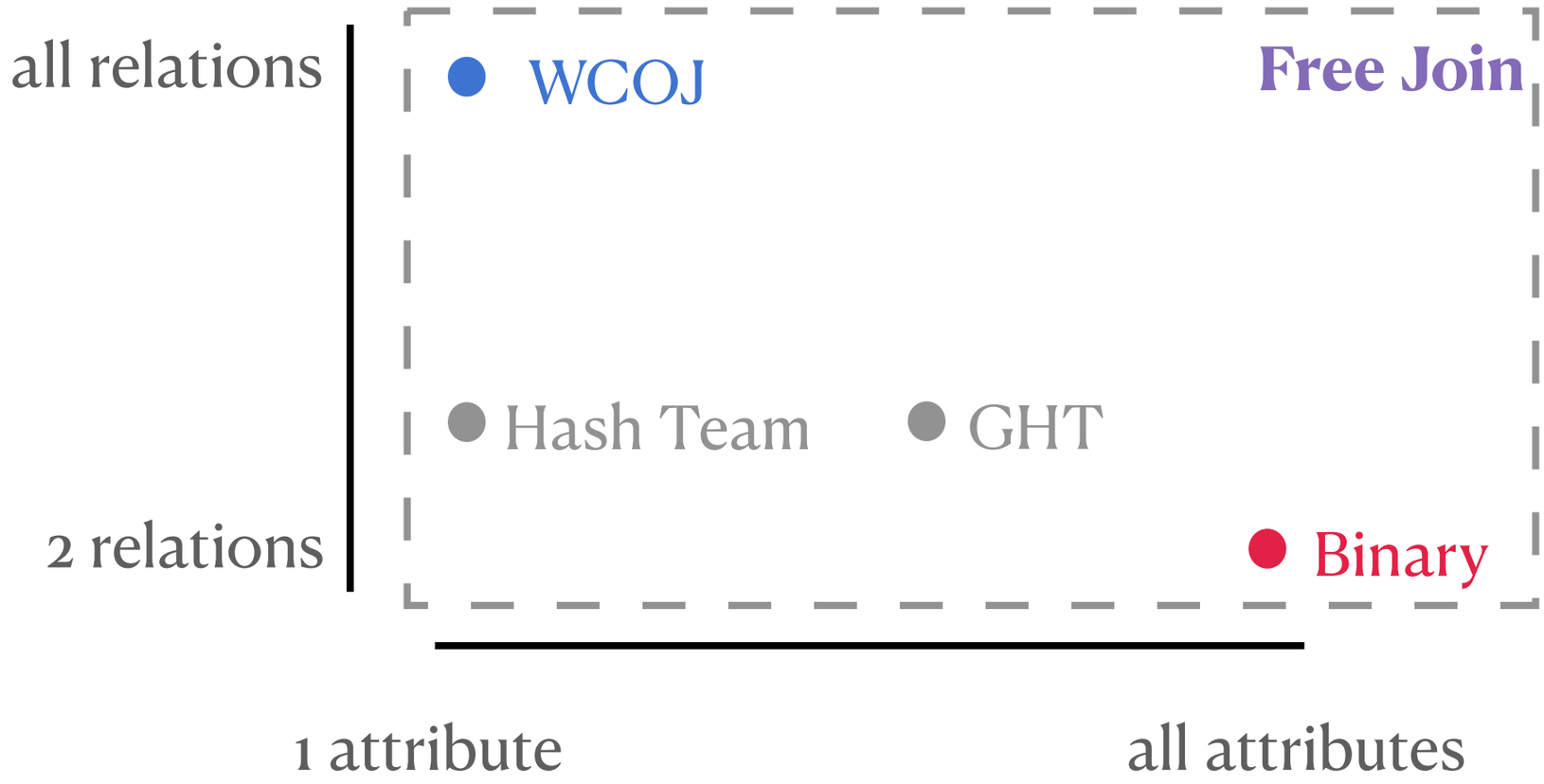
Problem

Can we **unify** WCOJ & binary join
into a new algorithm
w/ the best of both worlds?

Free Join = WCOJ + Binary Join

SIGMOD'23, **W**WS

Key idea: joins any # of relations & attributes



Step 1: partition schema into **subatoms**

schema	subatoms	
`R(x,y,z)`	`R(x,y)`	`R(z)`
`S(y,z,w)`	`S(y)`	`S(z,w)`
`T(z,w,x)`	`T(z,w)`	`T(x)`
`U(w,x,y)`	`U(w)`	`U(x,y)`

Step 1: partition schema into **subatoms**

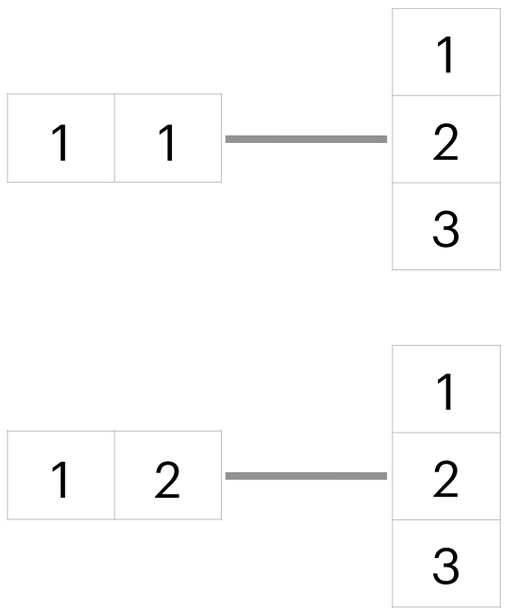
schema

`R(x,y,z)`

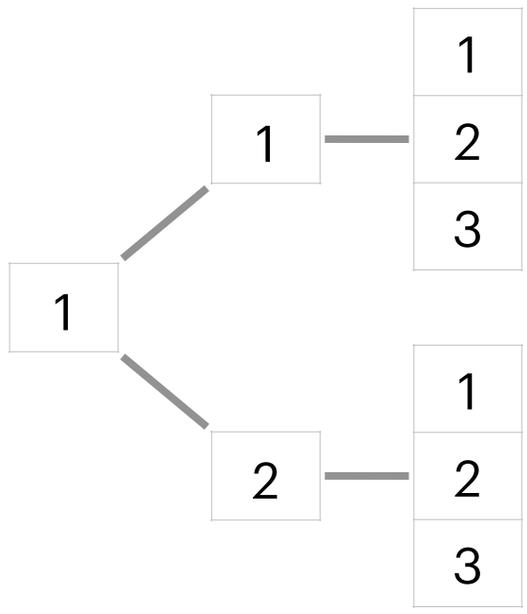
x	y	z
1	1	1
1	1	2
1	1	3
1	2	1
1	2	2
1	2	3

subatoms

`R(x,y)` `R(z)`



`R(x)` `R(y)` `R(z)`



1 subatom = 1 trie level

Step 1: partition schema into **subatoms**

schema	subatoms	
`R(x,y,z)`	`R(x,y)`	`R(z)`
`S(y,z,w)`	`S(y)`	`S(z,w)`
`T(z,w,x)`	`T(z,w)`	`T(x)`
`U(w,x,y)`	`U(w)`	`U(x,y)`

Step 1: partition schema into **subatoms**

Step 2: organize subatoms into groups

schema	subatoms	
R(x,y,z)	R(x,y)	R(z)
S(y,z,w)	S(y)	S(z,w)
T(z,w,x)	T(z,w)	T(x)
U(w,x,y)	U(w)	U(x,y)

Step 2: organize subatoms into groups

group	subatoms			
1	R(x,y)	U(x,y)	T(x)	S(y)
2	S(z,w)	T(z,w)	R(z)	U(w)

Step 2: organize subatoms into groups

Step 3: compile to nested loops

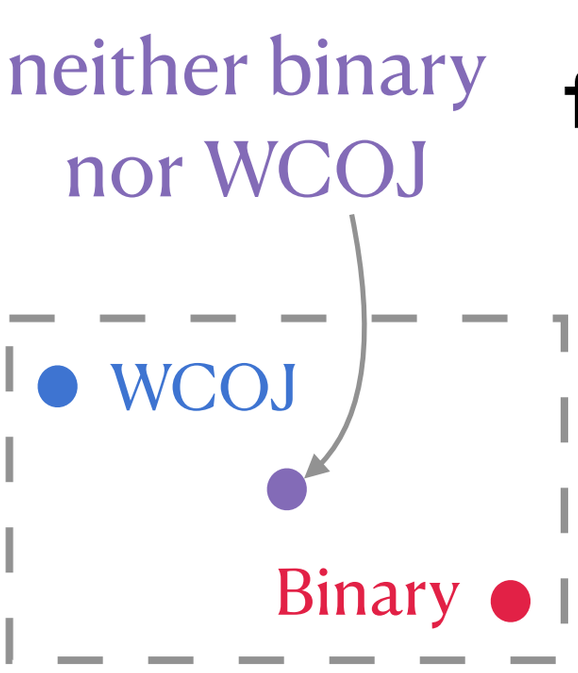
group	subatoms			
1	R(x,y)	U(x,y)	T(x)	S(y)
2	S(z,w)	T(z,w)	R(z)	U(w)

Step 3: compile to nested loops

```
for (x,y) in R:
  u=U[(x,y)]?; t=T[x]?; s=S[y]?
    for (z,w) in s:
      t[(z,w)]?; r[z]?; u[w]?
      output(x,y,z,w)
```

1	**R(x,y)**	U(x,y)	T(x)	S(y)
2	**S(z,w)**	T(z,w)	R(z)	U(w)

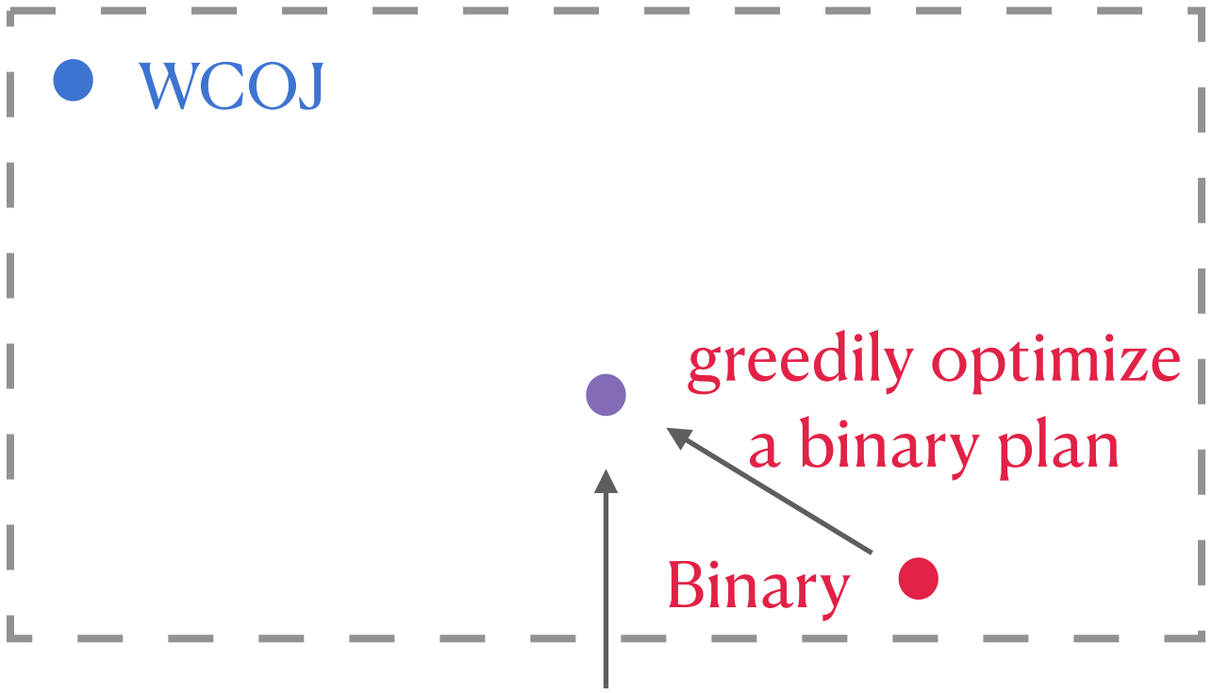
Step 3: compile to nested loops




```
for (x,y) in R:
  u=U[(x,y)]?; t=T[x]?; s=S[y]?
    for (z,w) in s:
      t[(z,w)]?; r[z]?; u[w]?
      output(x,y,z,w)
```

continue
on failure

1	**R(x,y)**	U(x,y)	T(x)	S(y)
2	**S(z,w)**	T(z,w)	R(z)	U(w)



How to pick the “sweet spot”?

`q.sql` →

UMBRA

PostgreSQL

DuckDB

As fast, or faster

→ Binary plan

↓ optimize

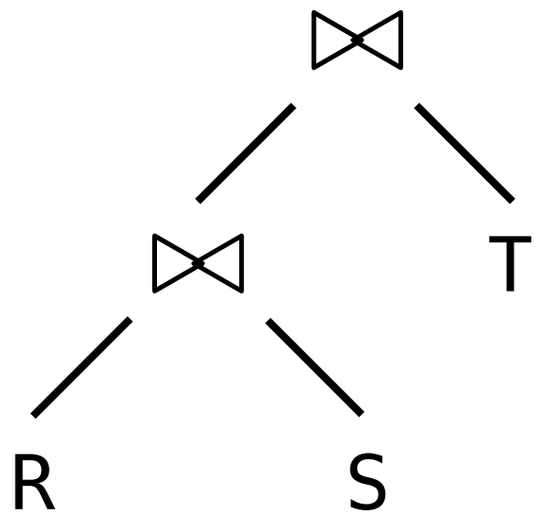
Free Join plan

↓ run

result

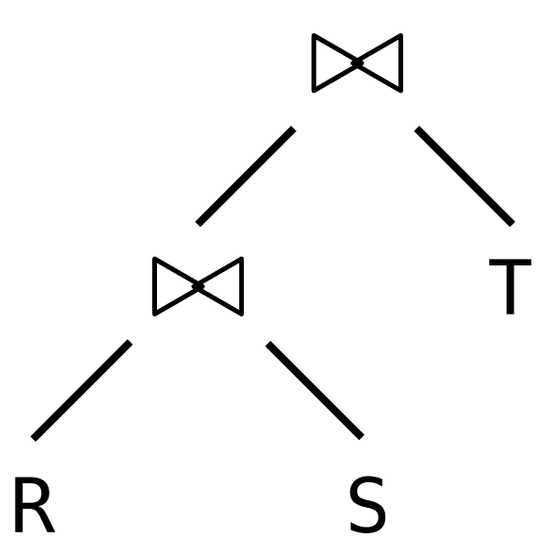
Already pretty fast!

```
SELECT *
  FROM R(x,a),S(x,b),T(x,c)
 WHERE R.x = S.x = T.x
```



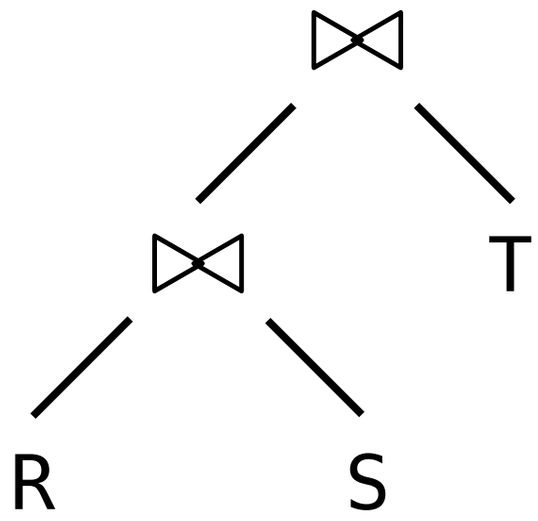

```
for (x,a) in R
  s = S[x]?
  for b in s
    t = T[x]?
    for c in t
      output(x,a,b,c)
```

```
SELECT *
  FROM R(x,a),S(x,b),T(x,c)
 WHERE R.x = S.x = T.x
```




```
for (x,a) in R
  s = S[x]?
  for b in s
    t = T[x]?
    for c in t
      output(x,a,b,c)
```

```
SELECT *
  FROM R(x,a),S(x,b),T(x,c)
 WHERE R.x = S.x = T.x
```




```
for (x,a) in R
  s = S[x]?
  t = T[x]?
  for b in s
    for c in t
      output(x,a,b,c)
```

Greedily pull out lookups

w/o reordering

or building extra hash levels

Best of Both Worlds

WCOJ

Asymptotic Complexity

Adaptive Planning

Binary Join

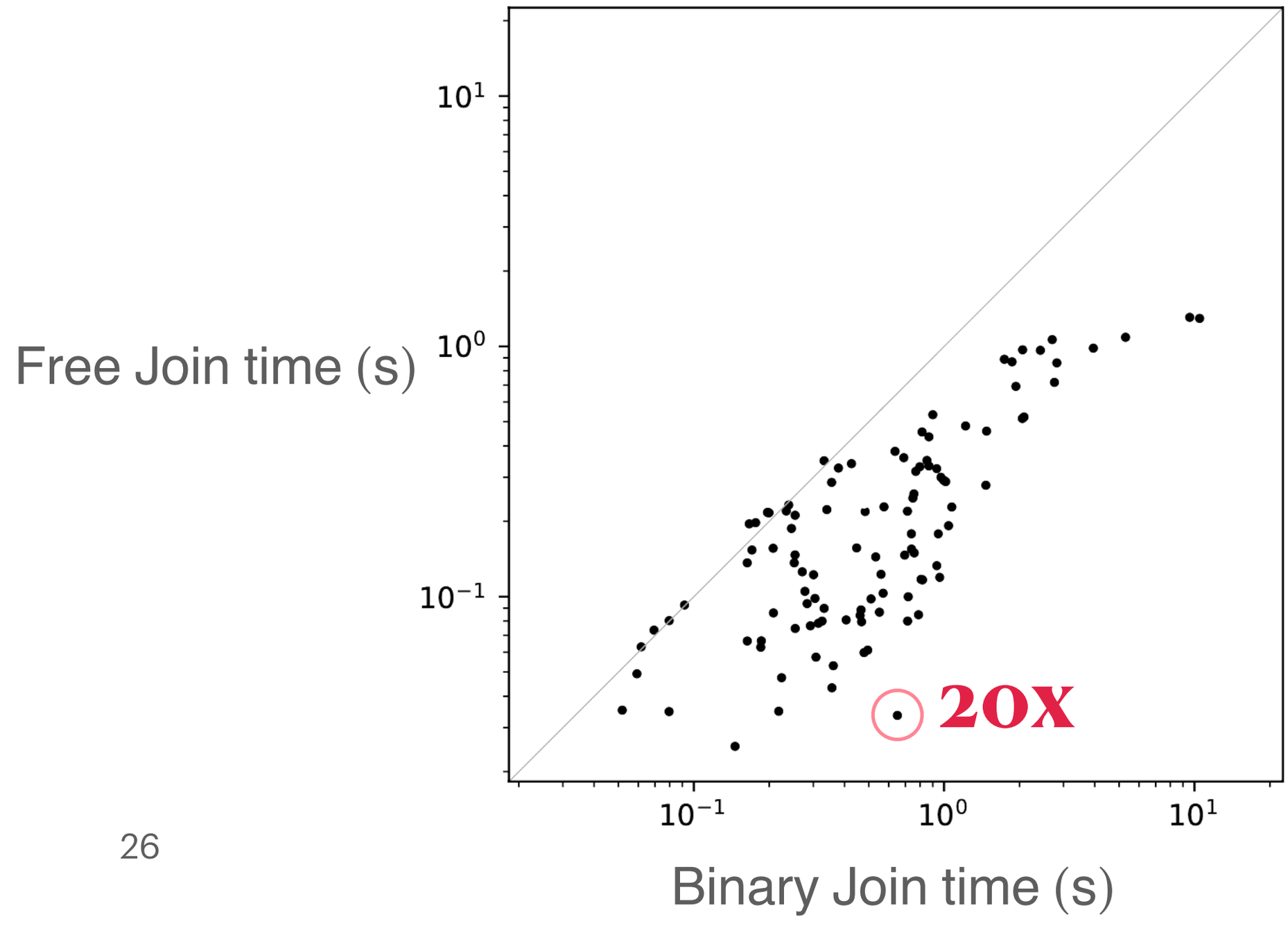
Constant Factor

Column-wise Layout

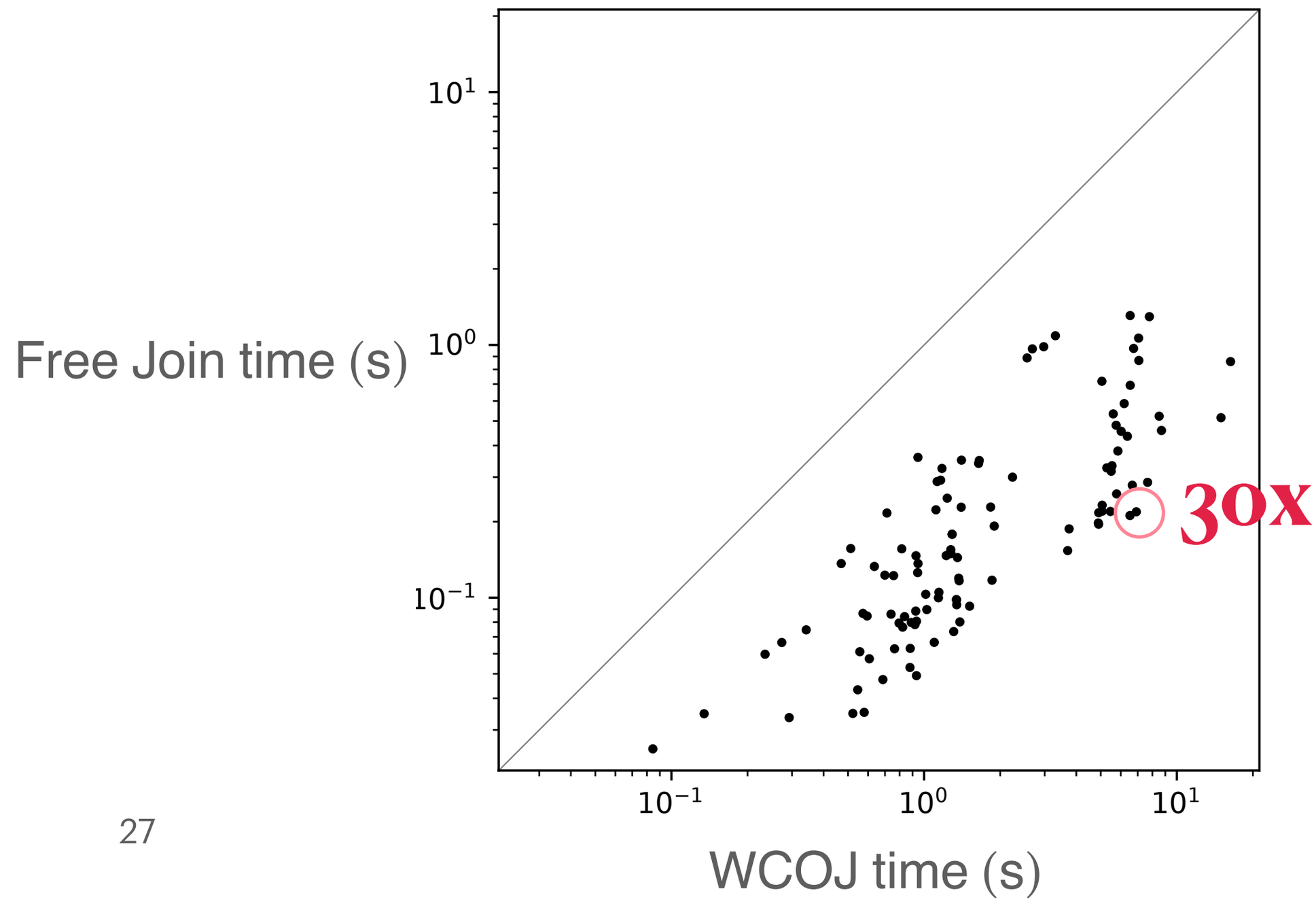
Vectorization

Query optimization

Free Join v.s. Binary Join



Free Join v.s. WCOJ



Next: how to **Free** your DB