Persistent Pointer Information

Xiao Xiao, Qirun Zhang, Jinguo Zhou, Charles Zhang

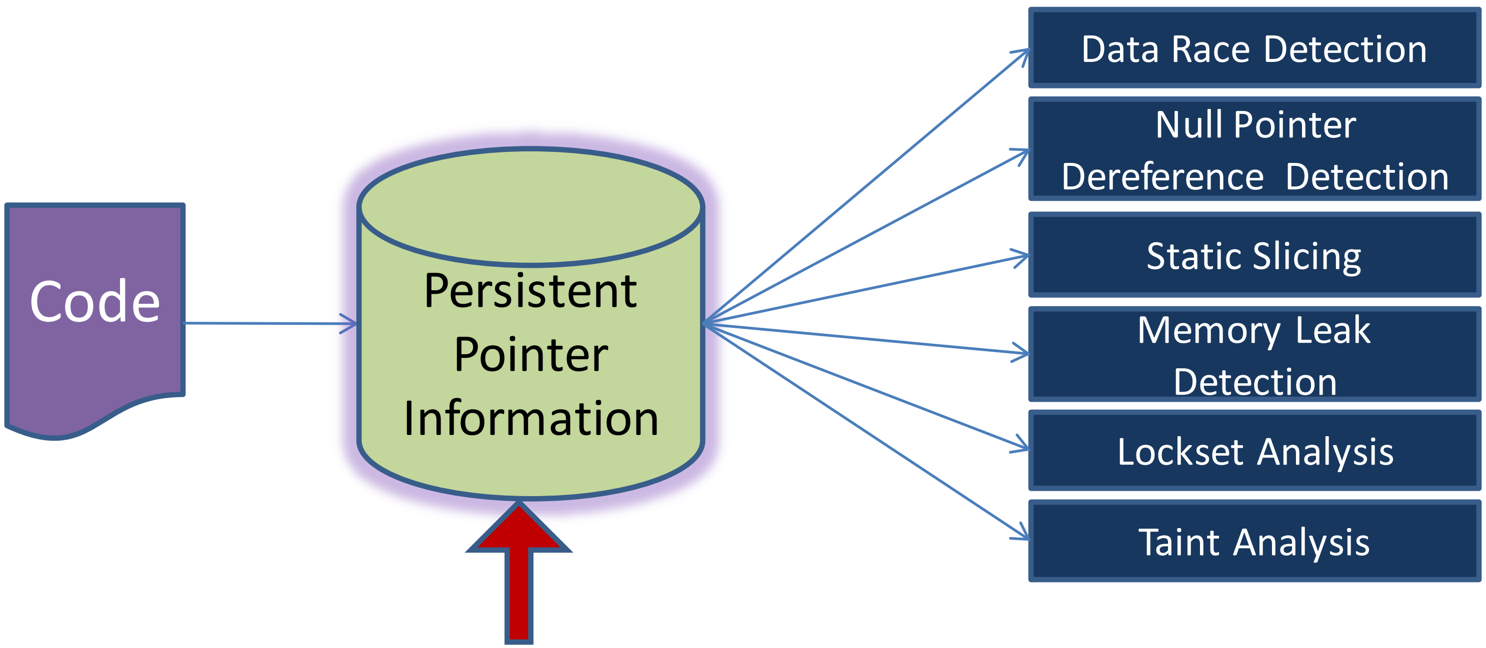
The Hong Kong University of Science & Technology

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

THE DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
計算機科學及工程學系

Motivation

Computing once, using multiple times!



Storing the pointer information in a database!

1. How to store?
2. What to store?

How to store?

- Using general database?

Not enough!

Not domain-aware!

- Storage size is large

How to store?

Program	Language	LOC	#Pointers	#Objects
samba	C	2112.7K	1004880	237201
gs	C	1508.1K	711082	150009
php	C	1312.4K	673156	146760
postgreSQL	C	1189.2K	584774	131886
antlr	Java	75.4K	302560	76970
luindex	Java	67.4K	269878	70426
bloat	Java	188.4K	625056	129471
chart	Java	375.1K	890971	234811
batik	Java	404.5K	766238	137488
sunflow	Java	326.2K	552974	106456
tomcat	Java	357.5K	657394	103627
fop	Java	415.1K	1173406	201122

Group 1

Group 2
Dacapo-2006

Group 3
Dacapo-9.12

- Group 1: flow-sensitive, POPL '11
- Group 2: 1-object + *1*-heap sensitive + JDK 1.4, ISSTA '02
- Group 3: 1-callsite + *1*-heap sensitive + JDK 1.6, ISSTA '11

How to store?

We compress points-to information by bzip:

	Compressed Size	Compression Time
Samba	3.2 MB	2 min
gs	45 MB	11 min
php	141 MB	37 min
postgreSQL	572 MB	117 min
antlr	9.2 MB	1 min
luindex	7.9 MB	1 min
bloat	33 MB	4 min
chart	380 MB	27 min
batik	5,300 MB	260 min
sunflow	2,200 MB	139 min
tomcat	1,900 MB	146 min
fop	15,000 MB	353 min

Information format:

p_1 o_1 o_2 o_3 ......
p_2 o_1 o_2 o_3 ....
......
p_n o_1 o_2 o_3 ......

What to store?

- Persisting points-to information only:
  - Computing aliasing information on-demand
  - A common strategy for existing tools

Not enough!

Pointed-by and aliasing related queries are not efficient.

What to store?

Information flow analysis:

- Given ***p.f = b***, the value of ***b*** can flow to all ***a*** iff:
  - ***a = q.f*** and ***(p, q)*** is an alias pair

Using IsAlias(p, q):

- Querying if (p, q) is an alias pair

Using ListAliases(p):

- Computing all the aliased pointers of *p*

Statements ***p.f = b*** and ***a = q.f*** are also distinguished by program points or contexts.

What to store?

Querying with points-to information:

	IsAlias (s)	ListAliases (s)
samba	103.7	55.3
gs	146.6	81.1
php	745.1	350.5
postgreSQL	843.2	365.3
antlr	35.1	26.7
luindex	28.7	22
bloat	134.2	105
chart	207.2	147.9
batik	117.6	30.3
sunflow	68.5	26.5
tomcat	71.3	29.6
fop	205.9	57.5

Not so fast!

Underlying data structure is bitmap.

What to store?

Query	Description
ListPointsTo(p)	Output the points-to set for pointer p
ListPointedBy(o)	Output the pointers that point to memory o
IsAlias(p, q)	Decide if the pointer p is an alias of q
ListAliases(p)	Output the pointers that are aliased to pointer p

- Efficiently supporting all common queries
  - Points-to information is not enough!
- Storing points-to + pointed-by + alias information!

Solution I: Sparse Bitmap

- Using sparse bitmap:
  - Representing all information by matrix
  - Efficient for manipulating sparse boolean matrix

- Points-to matrix: PM
- Pointed-by matrix : PM^T
- Alias matrix: $PM \times PM^T$

Solution I: Sparse Bitmap

- Computing alias matrix is inefficient:

	Computing Time	Storage Size
antlr	97.3 s	1.6 G
luindex	67.2 s	1.3 G
bloat	1448.7 s	5.1 G

Try BDD?

BDD is much more compact than bitmap.

Solution II: BDD

- Slow for generating alias matrix:
  - Same variable ordering does not work well for both PM and PM^T
  - Cannot terminate in 1 hour

- Slow for querying:

	Storage Size (PM only)	IsAlias (Bitmap)	IsAlias (BDD)
antlr	45M	28.3s	6752.6s
luindex	40M	23.7s	5146.2s
bloat	92M	101.2s	32907s

Solution II: BDD

- BDD is compact because:
  - BDD merges all *equivalent* points-to sets;
  - BDD merges *similar* points-to sets (with shared prefix);

- BDD is NOT query-efficient because:
  - BDD does not support set operations for individual elements
  - IsAlias(p, q): We should first take out the points-to sets of p and q, and intersect them.

Solution II: BDD

- Can we design a data structure that retains:
  - The compactness of BDD;
  - The querying efficiency of sparse bitmap;

Merging Equivalent Sets

- Are there still many equivalent pointers *after* the points-to analysis?

	Average (For all subjects)
Non-equivalent Pointers	18.5%
Non-equivalent Objects	82.9%

Merging Equivalent Sets

- Compressed points-to matrix:
  - Size can be reduced by 71.5%!

Faster enough for computing alias matrix?

Yes!

samba	gs	php	postgreSQL	antlr	luindex	bloat	chart	batik	sunflow	tomcat	fop
0.7	5.3	12.6	16.9	0.8	0.7	4.4	6.5	344.4	228.7	545	615.4

Unit: second

Merging Equivalent Sets

- Storage size (MB):

samba	gs	php	postgreSQL	antlr	luindex	bloat	chart	batik	sunflow	tomcat	fop
20.4	30.1	46.7	54.5	13	11.9	46.6	58.3	172.7	113.4	146.3	255.7

Small enough?

Depends!

1. Aggregation effect can quickly take over;
2. Compressing them with bzip increases decoding time, may not be tolerable for some applications

Merging Similar Sets

- Using similarity property for compression:
  - p is aliased to: a1, a3, a4, a5, a7, a8
  - q is aliased to: a1, a2, a3, a4, a7, a8

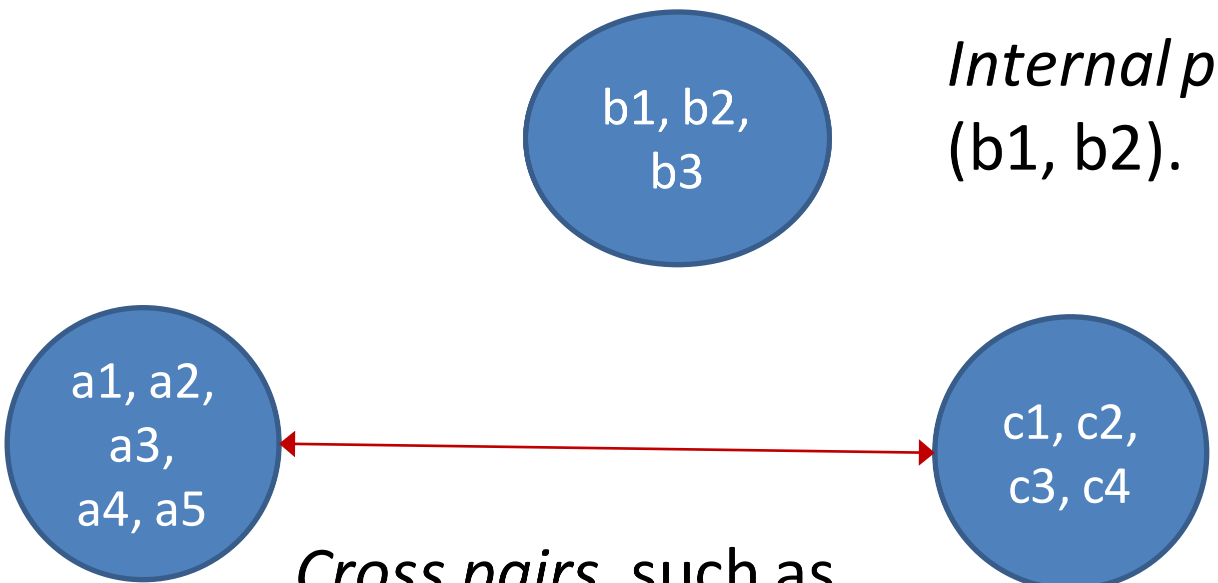
- Merging their common aliases:
  - (p, q): a1, a3, a4, a7, a8
  - p: a5
  - q: a2

Merging Similar Sets

- The aliasing relationships are unknown
- Intuitively, pointers with similar points-to sets may have similar aliasing relationships
- Grouping pointers with similar points-to sets instead

Merging Similar Sets

A hypothetical approach:



Internal pairs, such as (b1, b2).

Cross pairs, such as (a1, c2) .

Question

Challenges:

- What is the best way of partitioning?
- How to compute and encode cross pairs?

Pestrie Encoding

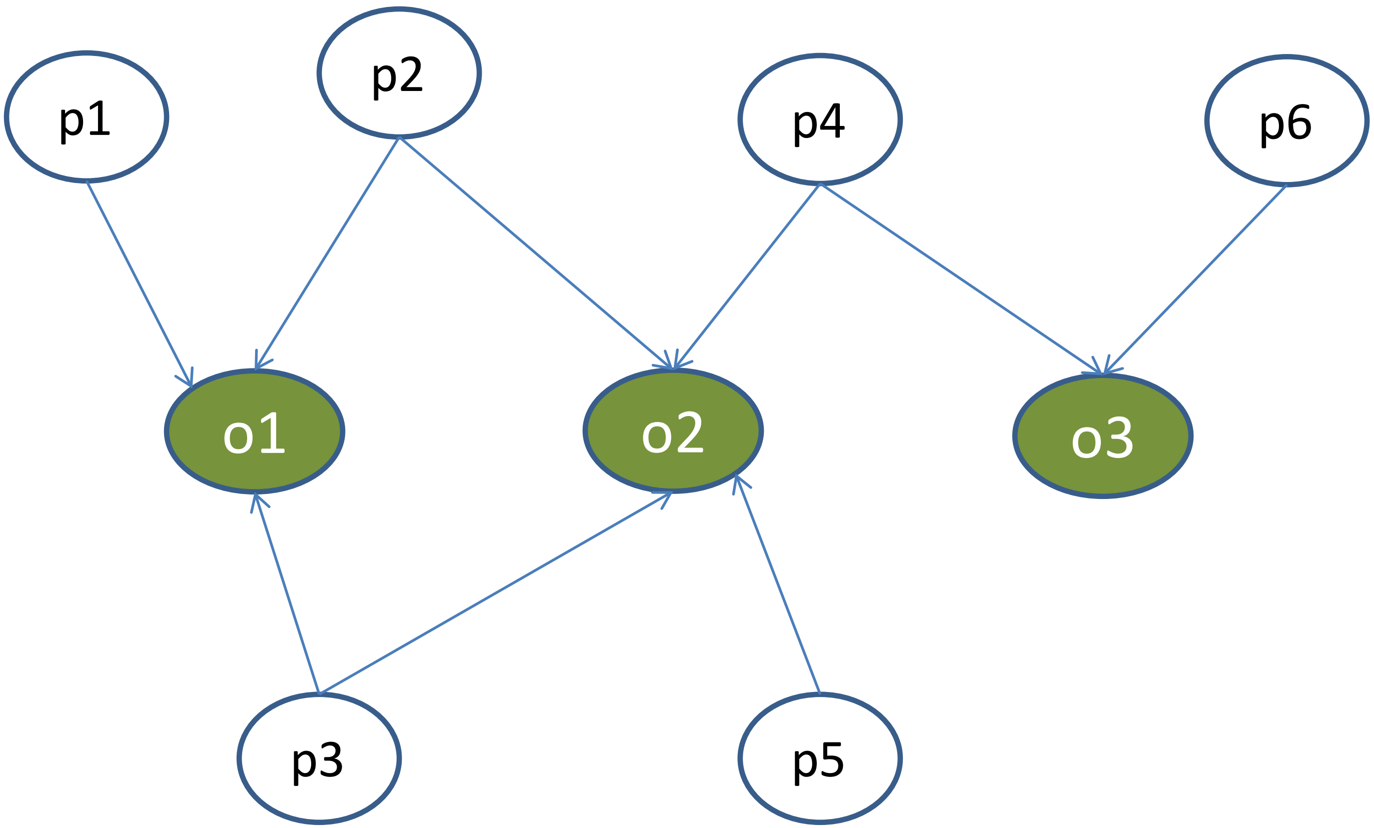
Solution sketch:

- Partitioning and structuring the pointers in the same group as a ***tree***
- Encoding trees as ***intervals***
- Representing cross pairs as ***rectangles***

Pestrie Encoding

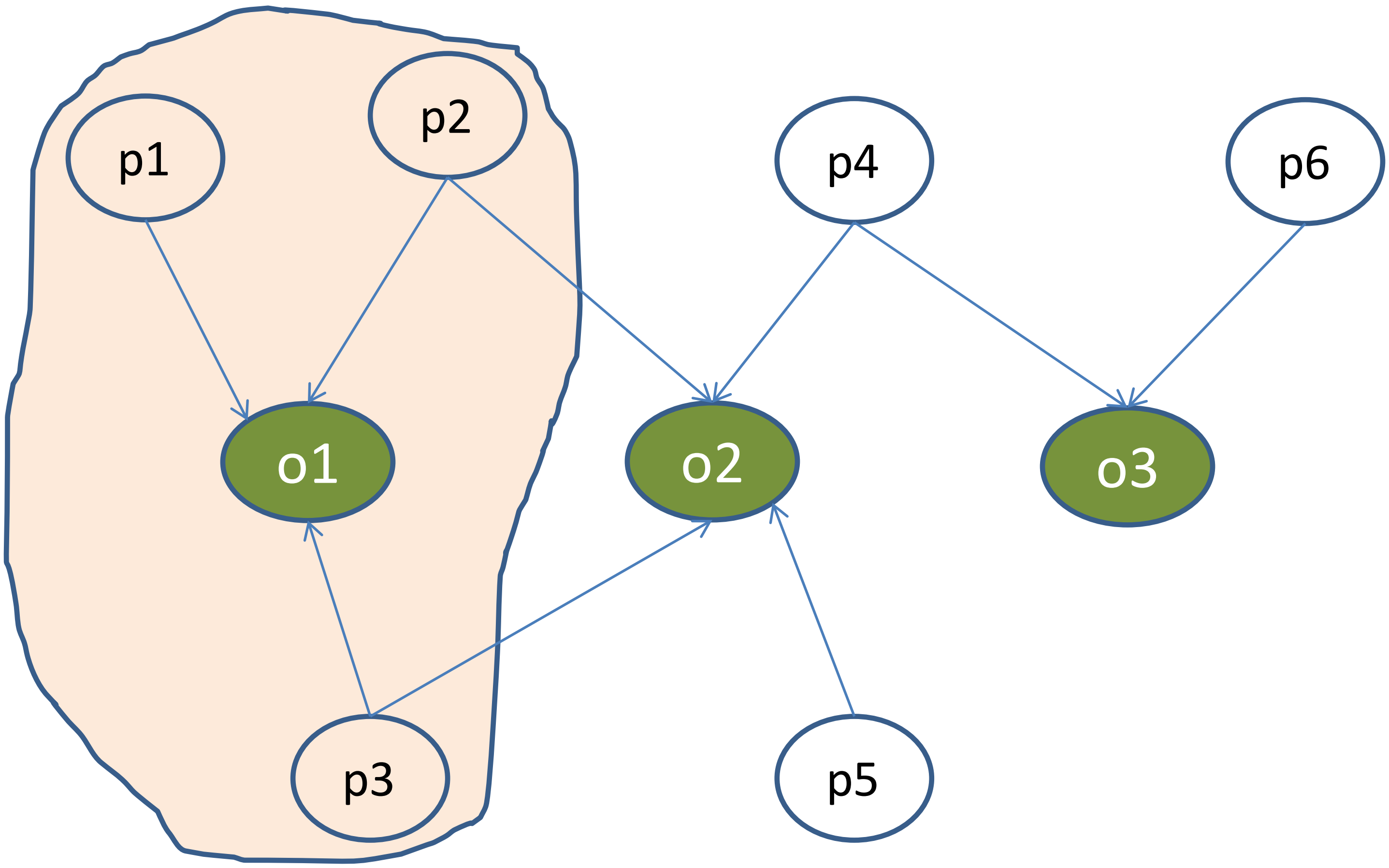
Step 1: Partitioning and structuring the pointers

- Pointers that point to the same object can be put into a partition



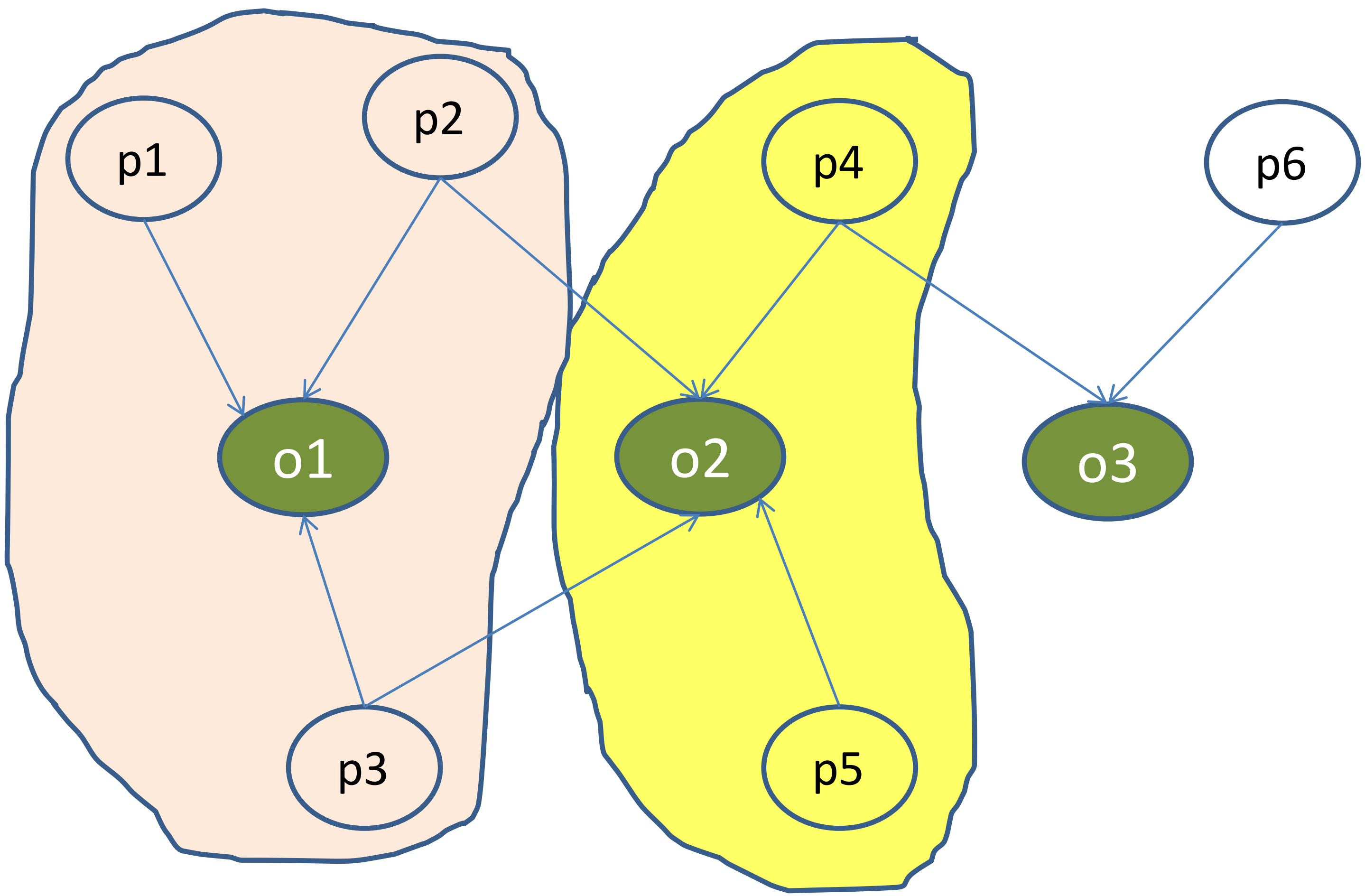
Pestrie Encoding

Using o1 for partition



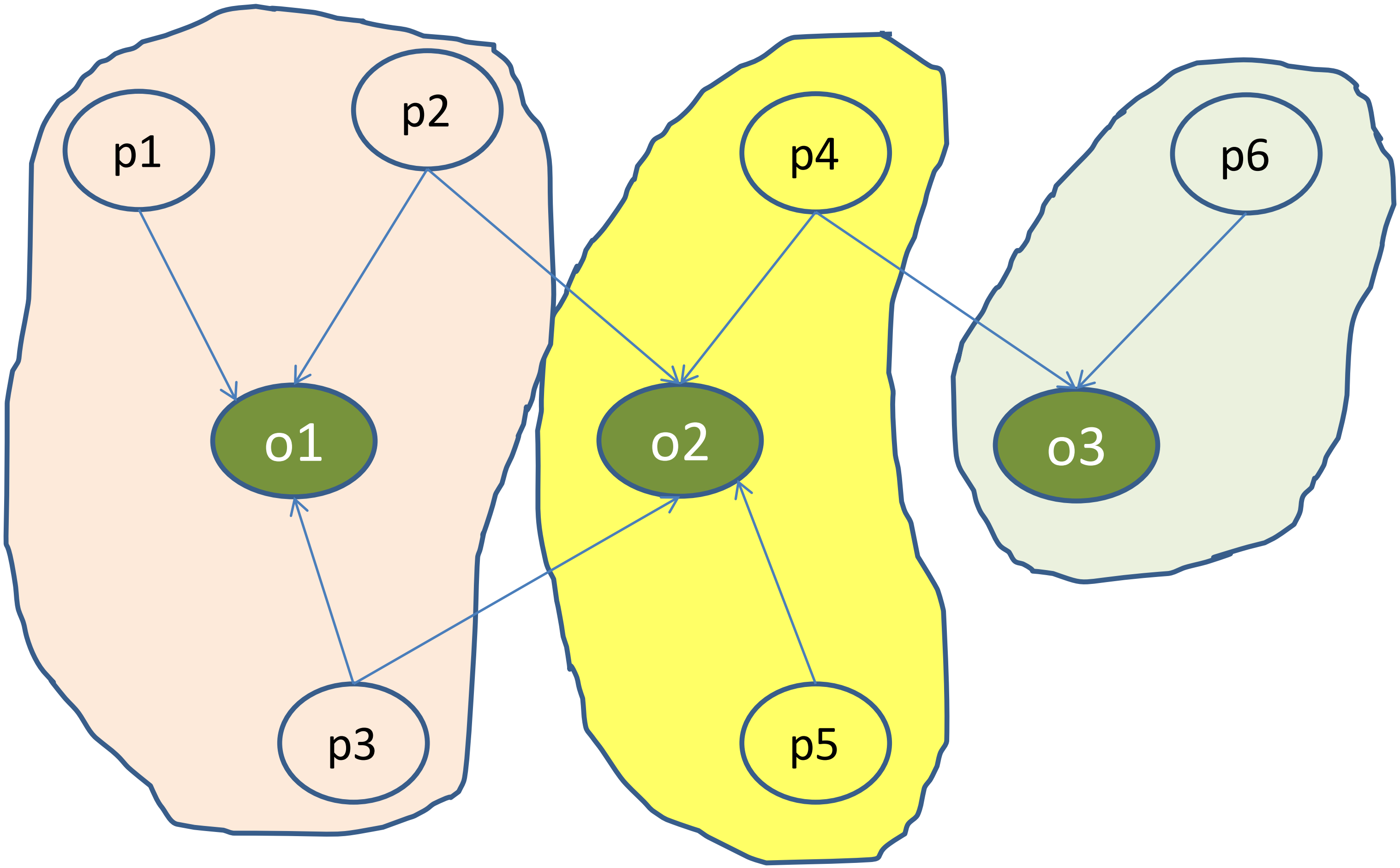
Pestrie Encoding

Using o2 for partition



Pestrie Encoding

Using o3 for partition



Pestrie Encoding

- Does the partitioning order of objects matter?

Yes!

Very similar to BDD, different ordering results in different compression ratio.

NP-hard problem for finding the best order.

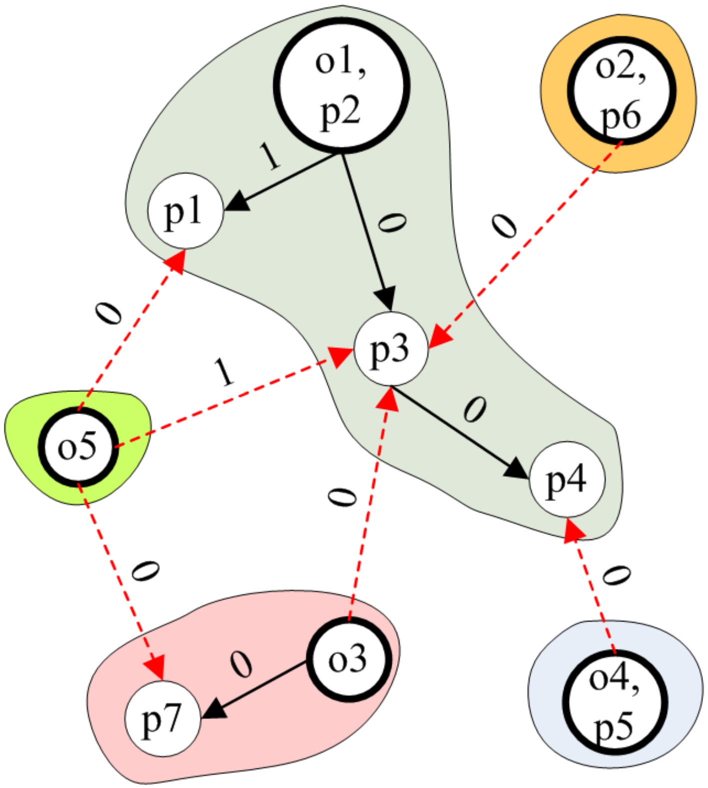
We give a good heuristic in the paper.

Pestrie Encoding

- A running example:
  - Using the order o1, o2, o3, o4, o5 for partition

PM	PM^T
$\begin{array}{cc} & \begin{array}{ccccc} o_1 & o_2 & o_3 & o_4 & o_5 \end{array} \\ \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{array} & \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$	$\begin{array}{cc} & \begin{array}{ccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 \end{array} \\ \begin{array}{c} o_1 \\ o_2 \\ o_3 \\ o_4 \\ o_5 \end{array} & \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$

Pestrie Encoding



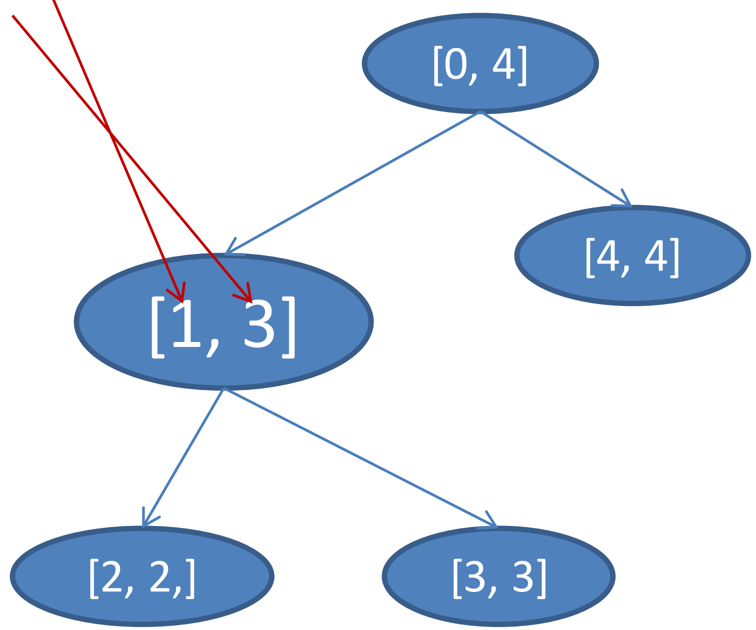
Pestrie: A trie variant

- Each shadowed area:
  - *P*artial *E*quivalent *S*et
  - A tree structure
- Red edges
  - Cross edges
- Black edge:
  - Tree edges
- Edge numbers:
  - Ignore them at the moment

Interval Encoding

Step 2: Encoding tree by interval label

- Interval label for node *v*: $[I_v , E_v]$
- I_v: the pre-order of *v* in DFS traversal
- E_v: the largest pre-order in the sub-tree of *v*

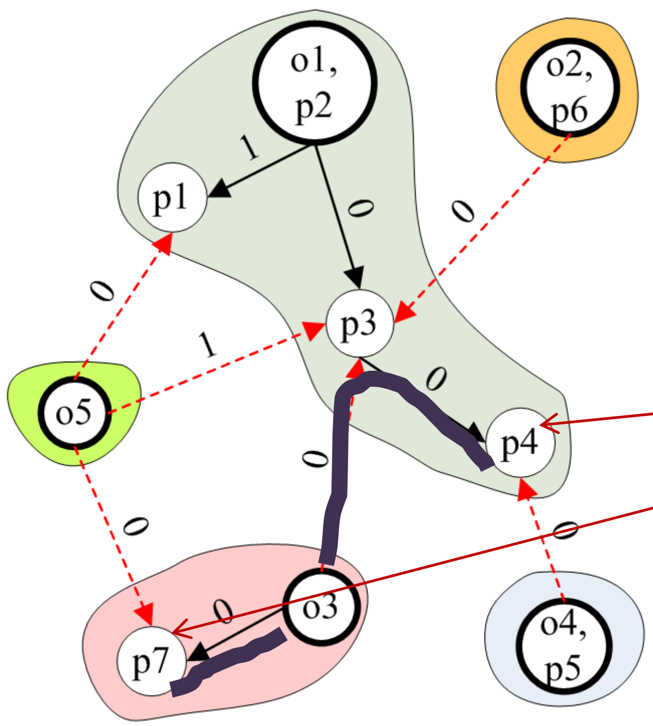


Pestrie Decomposition

Step 3: Encoding cross pairs

(p, q) is a cross pair iff:

– p and q are reachable from $u \rightarrow x$ and $u \rightarrow y$;



Two edges of root u

For example,
(p4, p7) is an alias

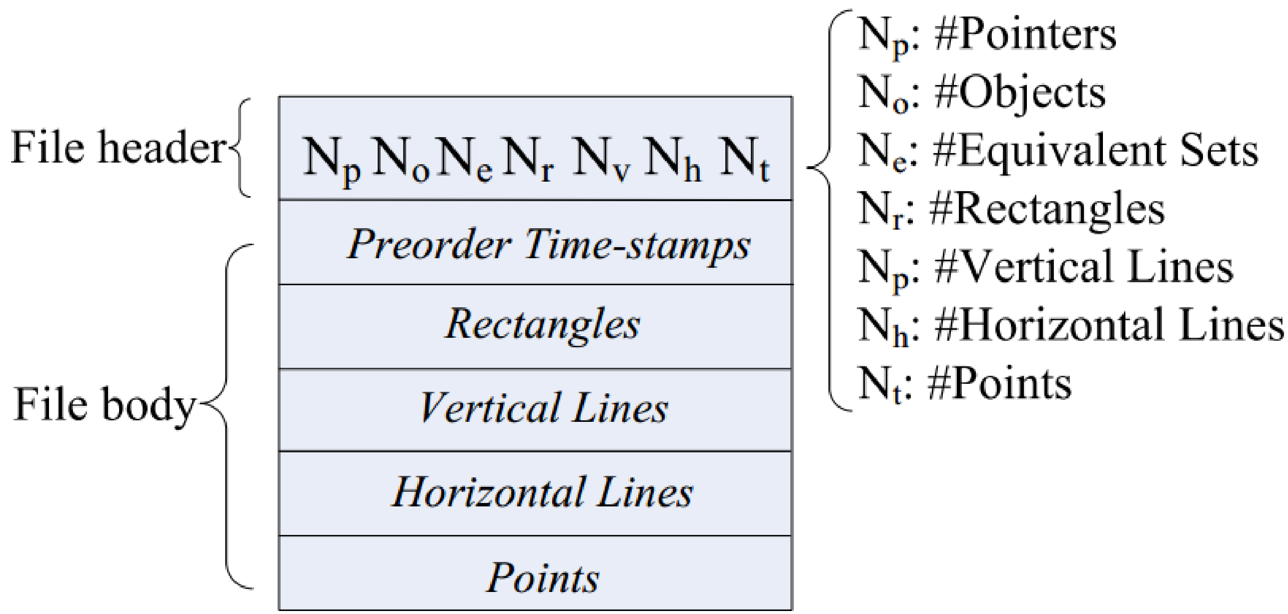
Pestrie Decomposition

- Encoding all cross pairs from the cross edges $u \rightarrow x$ and $u \rightarrow y$

- S1 (Nodes reachable from $u \rightarrow x$): [I_1, E_1]
- S2 (Nodes reachable from $u \rightarrow y$): [I_2, E_2]
- All cross pairs for $S1 \times S2$: [I_1, E_1, I_2, E_2]

Rectangle label

Generating Persistence file



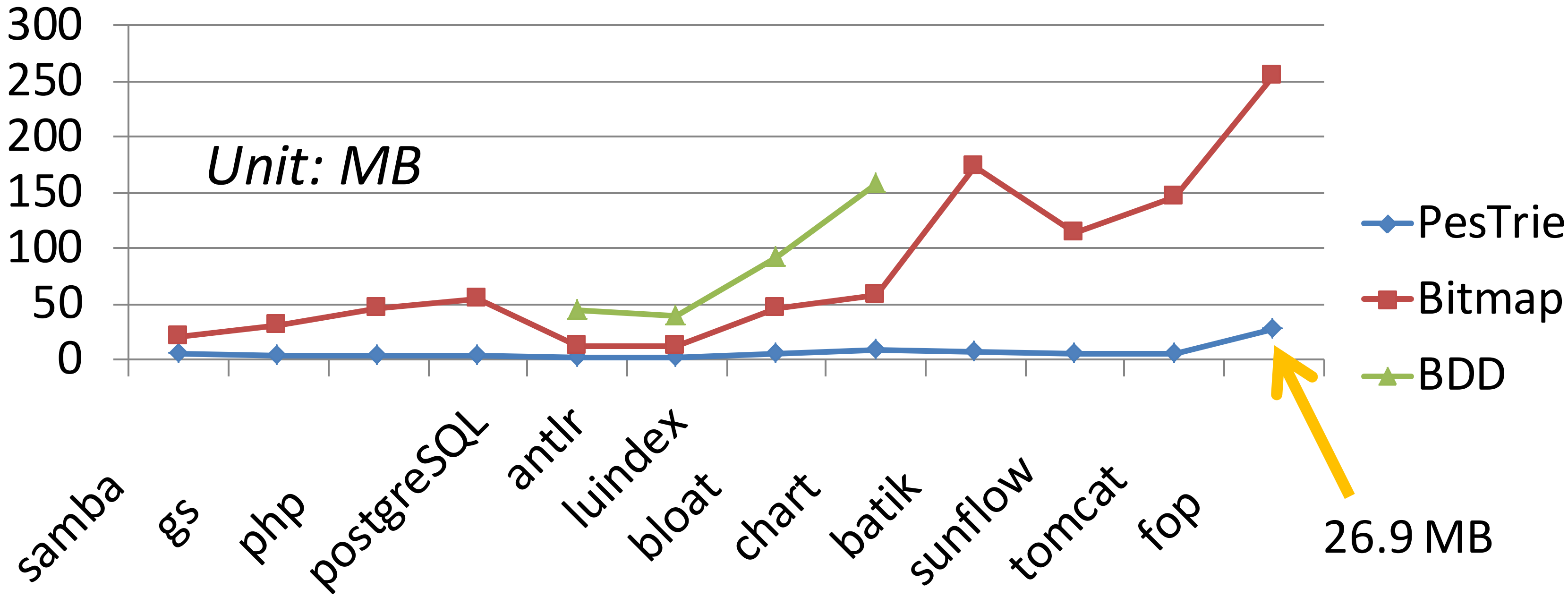
- Generating persistent file:
  - Including *points-to, pointed-by, alias* information
  - Rectangles are stored as points, lines, and rectangles

Decoding & Querying

- Decoding and constructing query structure:
  - Time: O((n+R)lgn)
  - R is #rectangles, n is #pointers

- Querying performance:
  - IsAlias: O(lgn)
  - ListAliases: O(K), *K* is the size of answer set
  - ListPointsTo: O(K)
  - ListPointedTo: O(K)

Storage Size



Bitmap:

Storing points-to and alias matrices

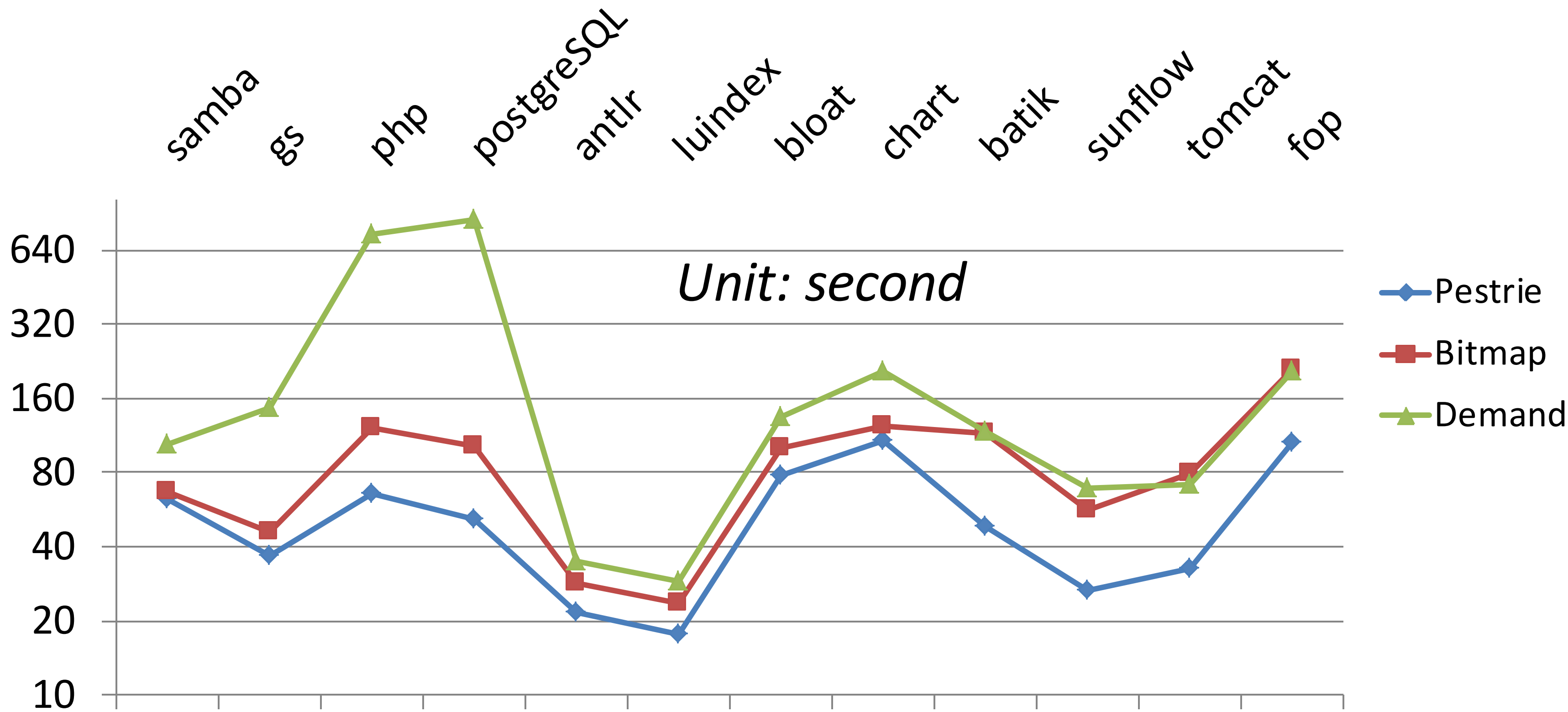
BDD:

Storing points-to matrix

Pestrie:

- Points-to, pointed-to, and alias matrices
- 10.5X smaller than bitmap
- 17.5X smaller than BDD

IsAlias Query

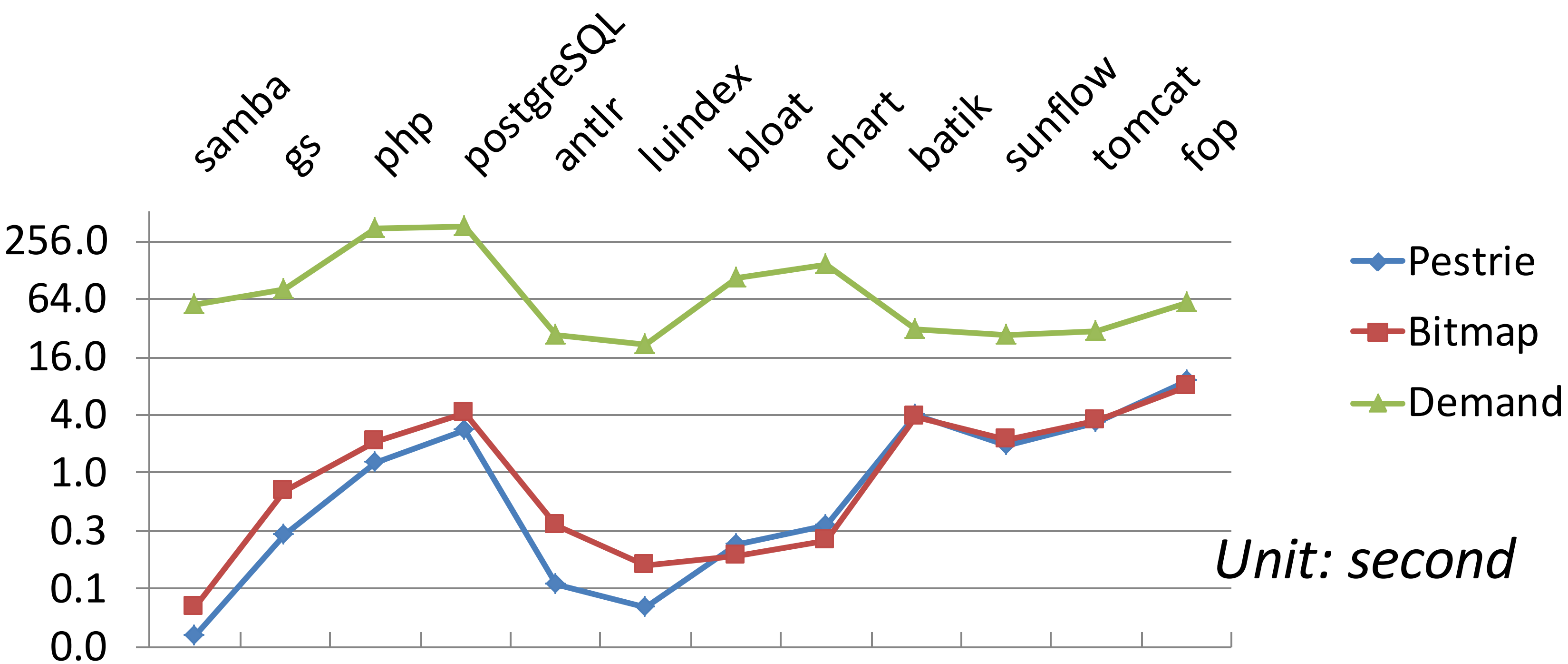


Pestrie is:

1.6X faster than bitmap (with alias information)

2.8X faster than bitmap (on-demand)

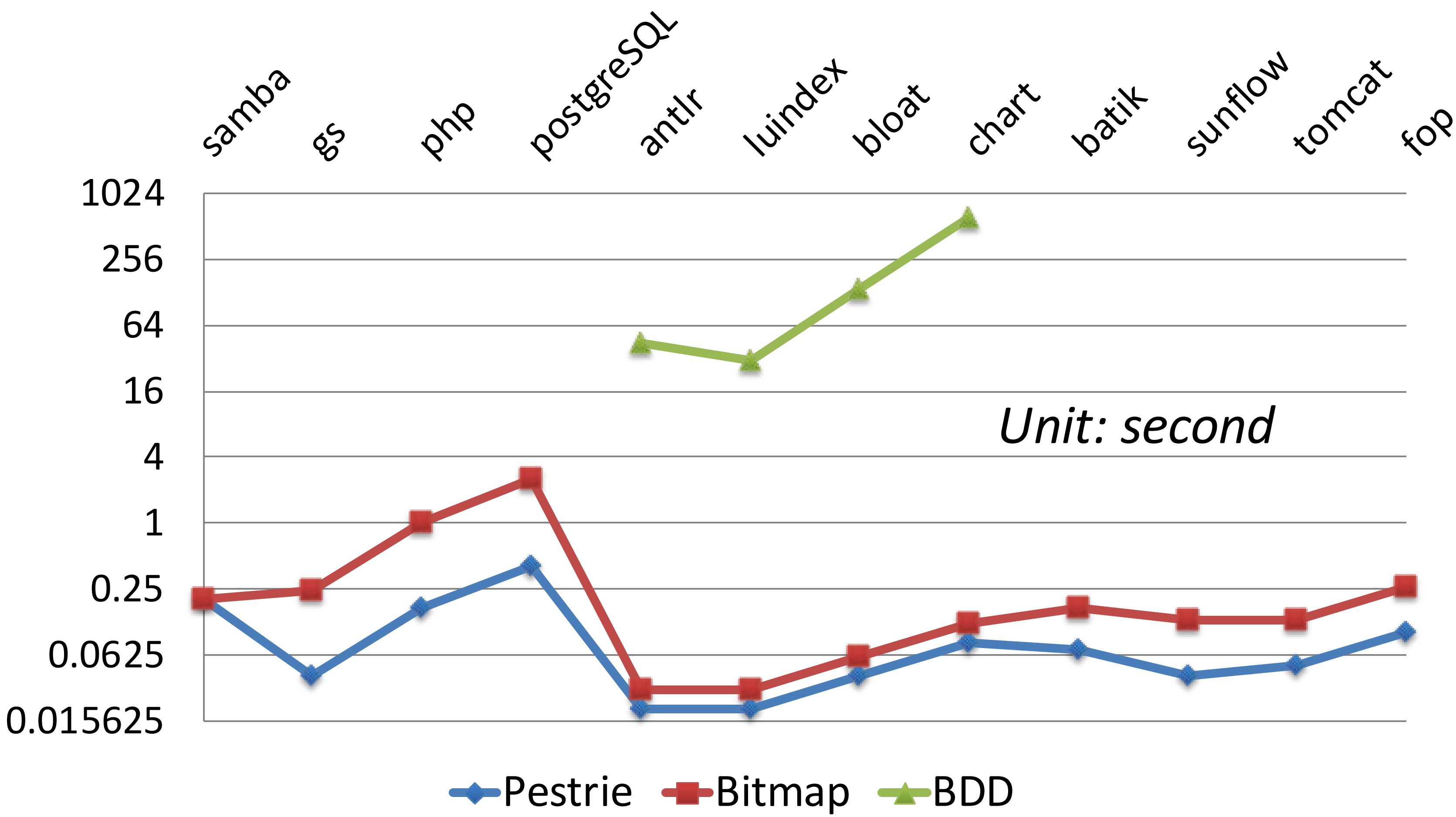
ListAliases Query



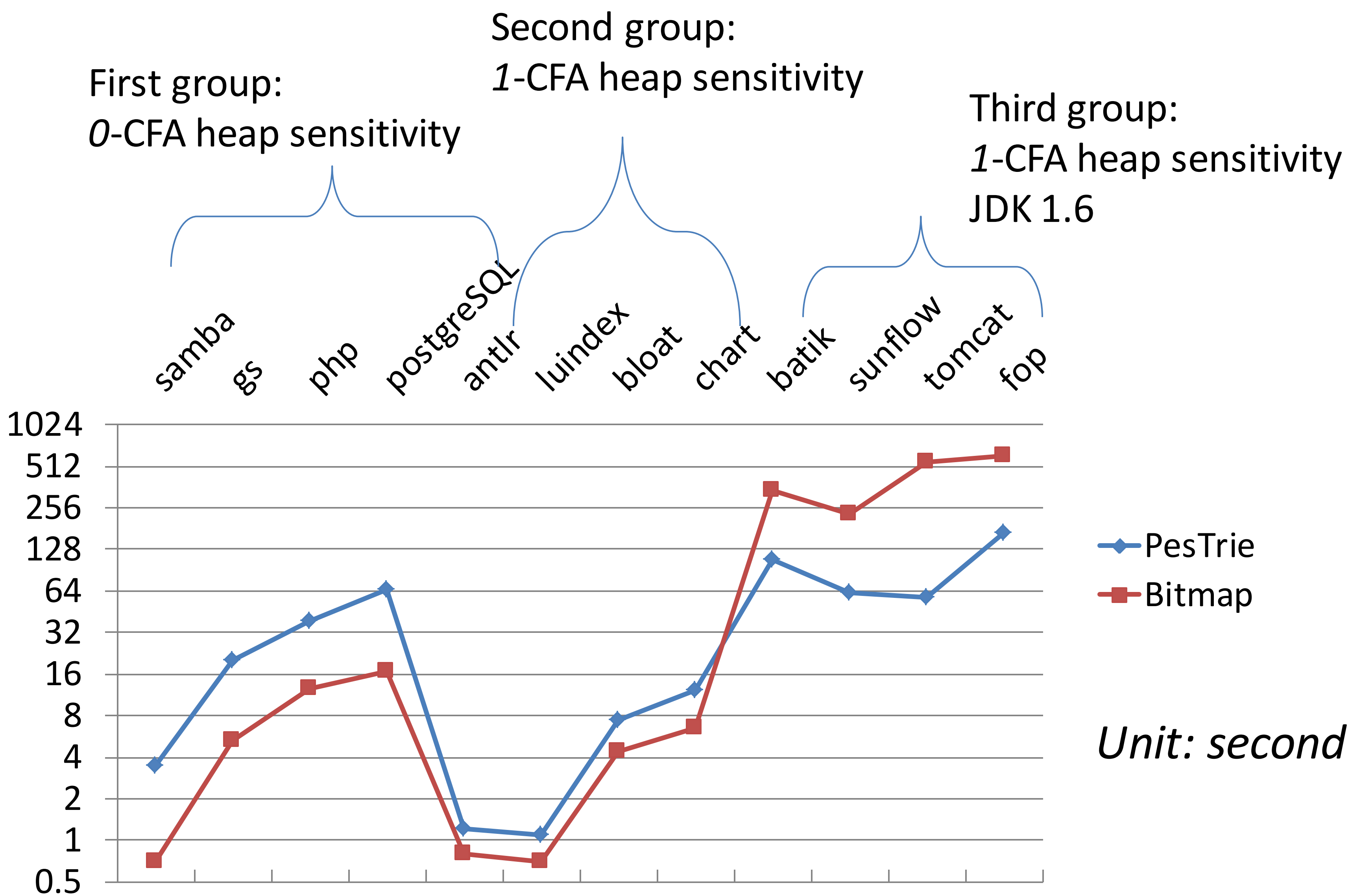
Pestrie and bitmap (with alias information) are almost equally fast.

Pestrie is 123.6X faster than bitmap (on-demand)

ListPointsTo Query

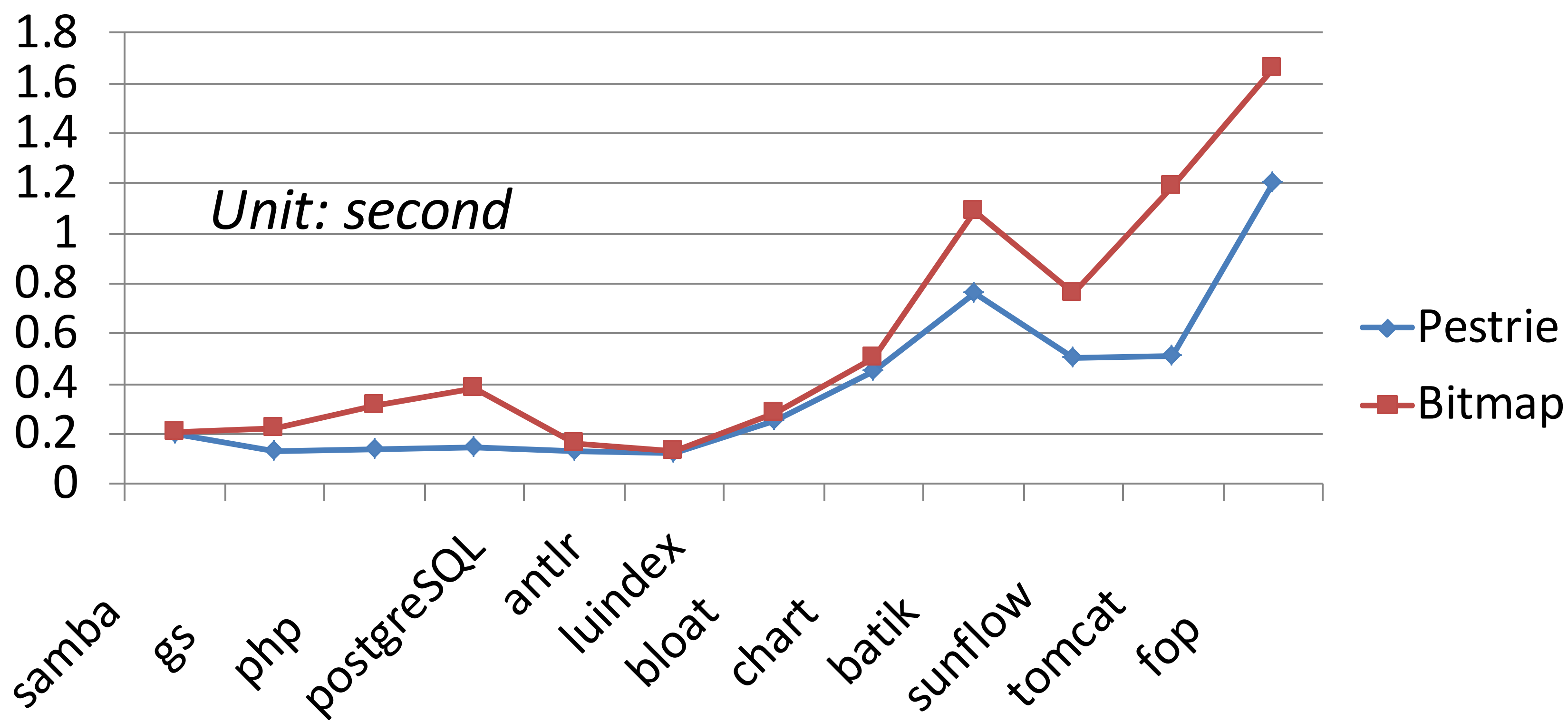


Construction Time



Unit: second

Decoding Time

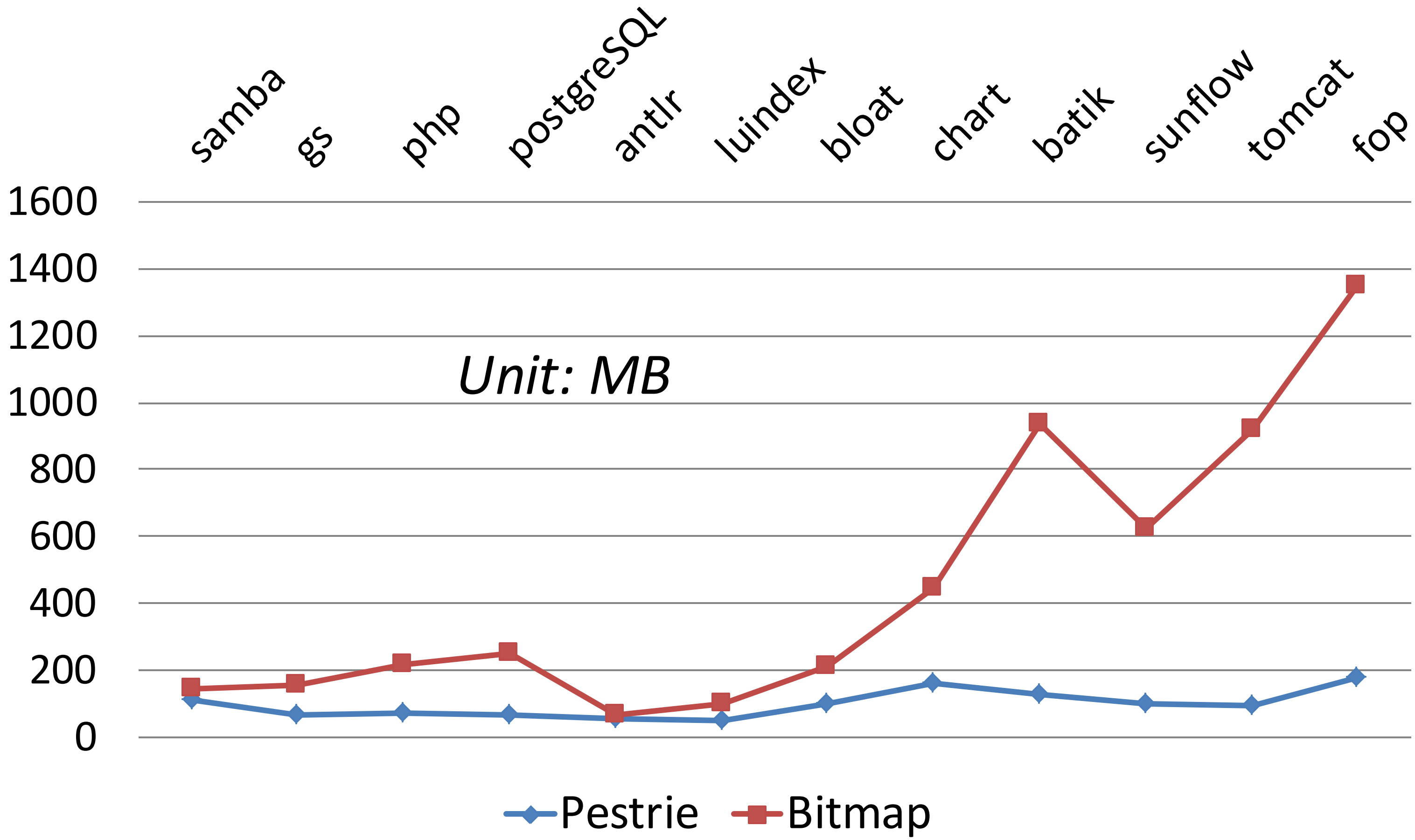


Bitmap decoding:

– Loading points-to and alias matrices;

– Constructing pointed-to matrix.

Querying Memory



More Details in Our Paper

- Complete construction algorithm;
- Pruning strategies;
- Proofs;
- Querying algorithms;
- Optimization theory;
- Preparing points-to matrix;
- More experiments and explanations.

Summary

- We study the problem of persisting points-to, pointed-to, and aliasing information.
- We design Pestrie persistence scheme:
  - Compact size
  - Fast querying
  - Efficient construction and decoding

Q & A

Thank You

Additional Slides

Partitioning Order

- Recall our aim:
  - Grouping the pointers with similar points-to sets

- Recall the HITS algorithm:
  - High quality hub has links to many authority pages
  - Authority page has many links to high quality hubs

Heuristic ===> Pages with similar authority values point to hubs with similar qualities

Partitioning Order

- Authority pages are analogy of pointers
- Hub pages are analogy of objects

- Using hub values to rank the objects

$$H_o = \sqrt{\sum_{p \in pt^T(o)} \left| pt(p) \right|^2}$$