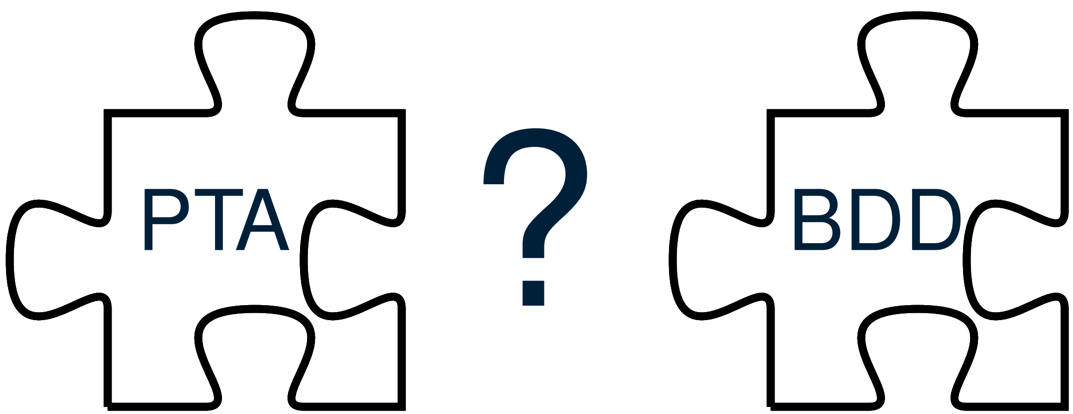
Points-to Analysis using BDDs

Marc Berndl, **Ondřej Lhoták**, Feng Qian,
Laurie Hendren, Navindra Umanee

Sable Research Group
McGill University
June 9th, 2003

Motivation

- Points-to analysis
  - requires representing many large, often similar sets
- Binary decision diagrams (BDDs)
  - provide compact representation of large sets with similarities



Background

- Points-to analysis
  - [Landi 92] [Andersen 94] [Emami 94] [Wilson 95] [Steensgaard 96] [Shapiro 97] [Aiken 98] [Fähndrich 98] [Ghiya 98] [Choi 99] [Das 00] [Hind 00] [Ruf 00] [Sundaresan 00] [Tip 00] [Heintze 01] [Liang 01] [Rountev 01] [Vivien 01] [Milanova 02] [Su 02] [Whaley 02] [Lhoták 03] and more...
- BDDs
  - [Bryant 92] [Burch 94] and many, many more...
- Program analysis using BDDs
  - [Sias 00] [Manevich 02] [Ball 03]

Talk Outline

- Introduction
  - Points-to analysis
  - BDDs
- BDD-PTA algorithm
- Performance tuning
  - Bit ordering
  - Incrementalization
- Overall performance
- Conclusions and future work

Overview

- Designed a subset-based Java points-to algorithm using BDDs
- Implemented it using BuDDy BDD library
- Compared performance of BDD-based solver with hand-tuned Spark solver on identical input constraints
  - Spark solver is very efficient compared to other Java points-to solvers [CC 03]

BuDDy: provided by Jørn Lind-Nielsen at
`http://www.itu.dk/research/buddy`

Simple points-to analysis example

```
X: a = new O();
Y: b = new O();
Z: c = new O();
a = b;
b = a;
c = b;
```

Points-to set:

{ }

Simple points-to analysis example

```
X: a = new O();
Y: b = new O();
Z: c = new O();
a = b;
b = a;
c = b;
```

Points-to set:
{ (a,X) (b,Y) (c,Z) }

Simple points-to analysis example

```
X: a = new O();
Y: b = new O();
Z: c = new O();
a = b;
b = a;
c = b;
```

Points-to set:
{ (a,X) (b,Y) (c,Z) (a,Y) }

Simple points-to analysis example

```
X: a = new O();
Y: b = new O();
Z: c = new O();
a = b;
b = a;
c = b;
```

Points-to set:

{ (a,X) (b,Y) (c,Z) (a,Y) (b,X) }

Simple points-to analysis example

```
X: a = new O();
Y: b = new O();
Z: c = new O();
a = b;
b = a;
c = b;
```

Points-to set:
{ (a,X) (b,Y) (c,Z) (a,Y) (b,X) (c,X) (c,Y) }

BDD representation

- A BDD is a compact representation of a set of bit strings
- We encode our analysis using bit strings:

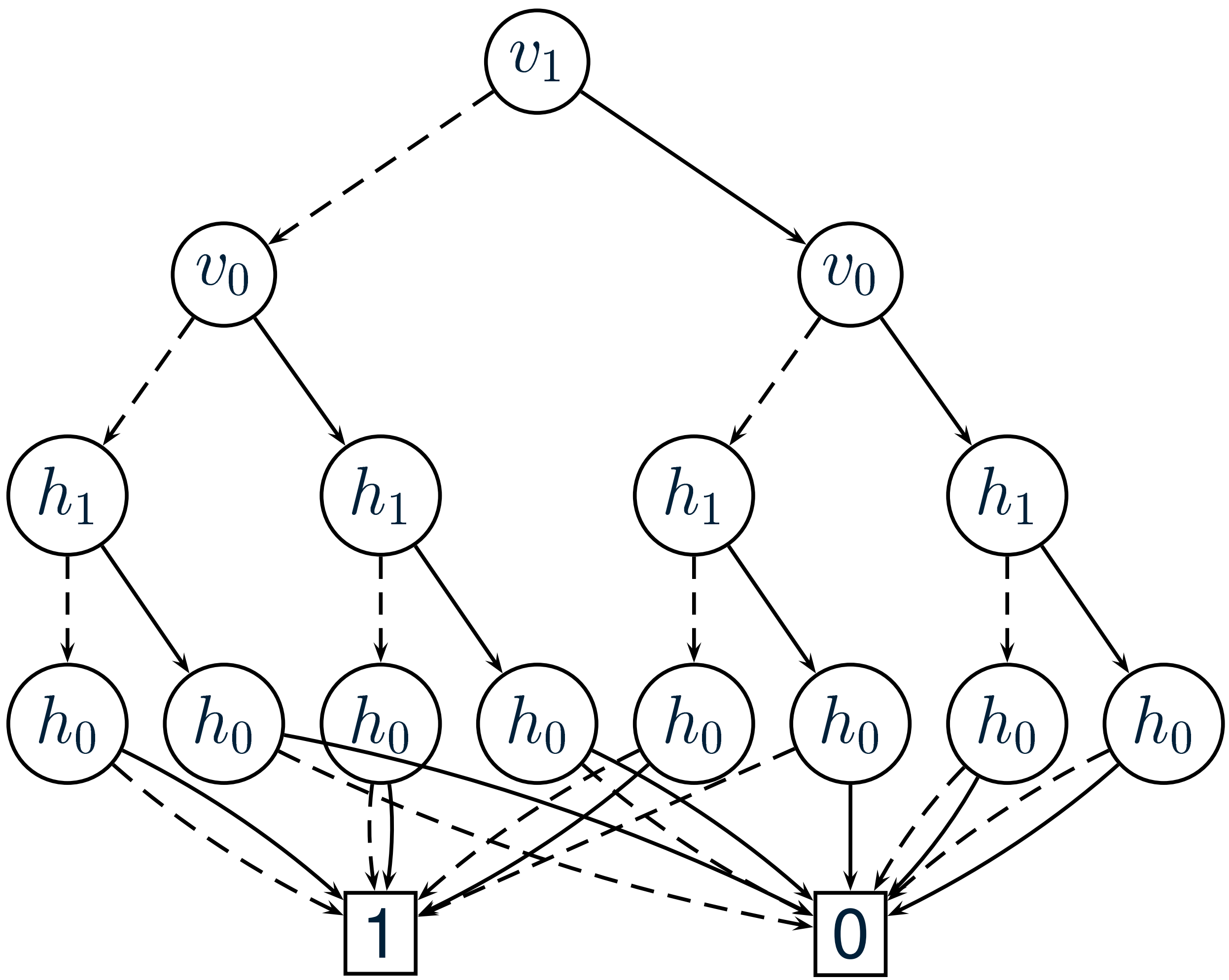
$$\begin{array}{ll} \text{a} \rightarrow 00 & \text{X} \rightarrow 00 \\ \text{b} \rightarrow 01 & \text{Y} \rightarrow 01 \\ \text{c} \rightarrow 10 & \text{Z} \rightarrow 10 \end{array}$$

Domains: V H

$v_1 v_0 h_1 h_0$

(a,Y) $\rightarrow$ 00 01

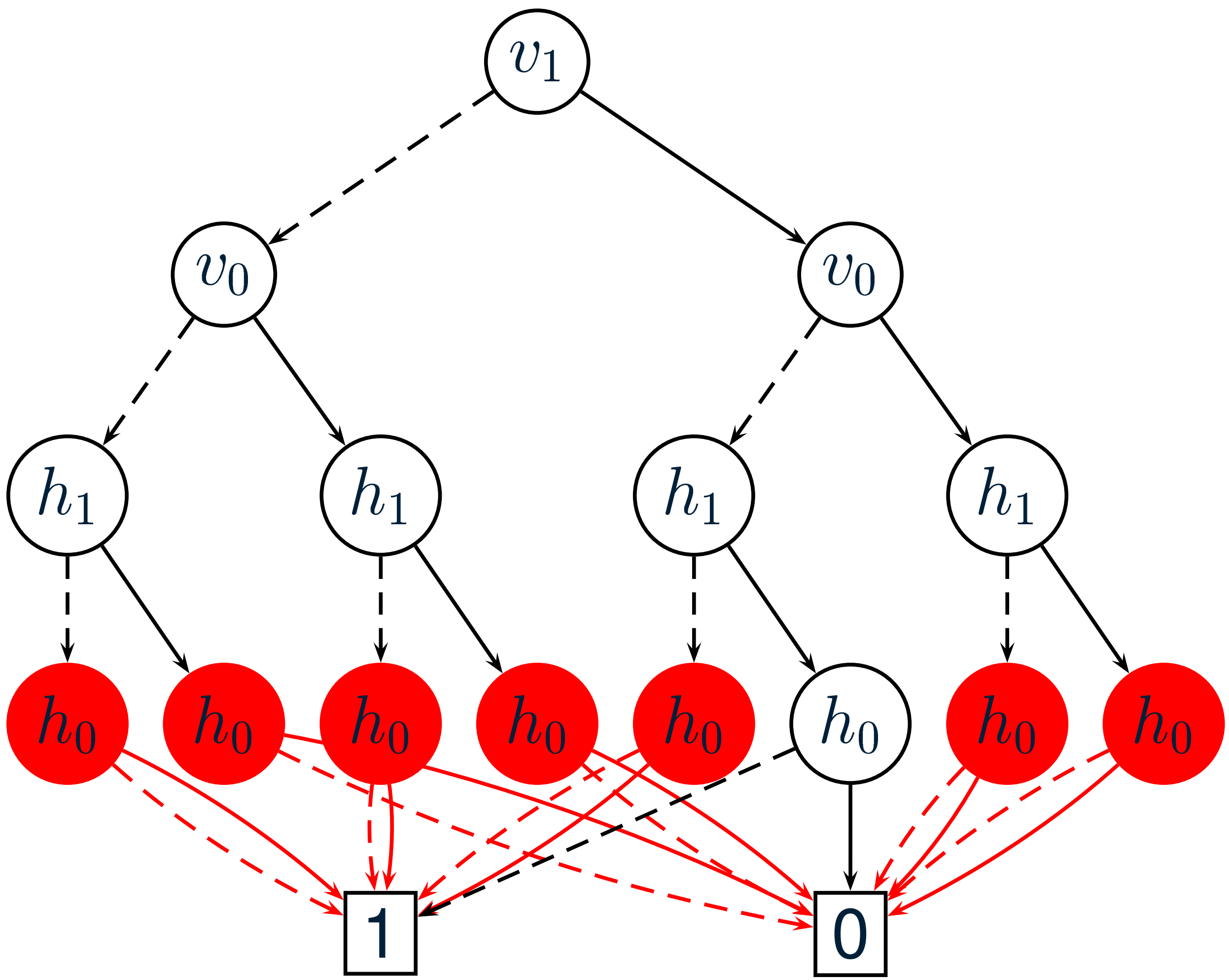
BDD representation



a/X $\rightarrow$ 00
b/Y $\rightarrow$ 01
c/Z $\rightarrow$ 10

	V	H
	$v_1 v_0$	$h_1 h_0$
(a,X)	00	00
(a,Y)	00	01
(b,X)	01	00
(b,Y)	01	01
(c,X)	10	00
(c,Y)	10	01
(c,Z)	10	10

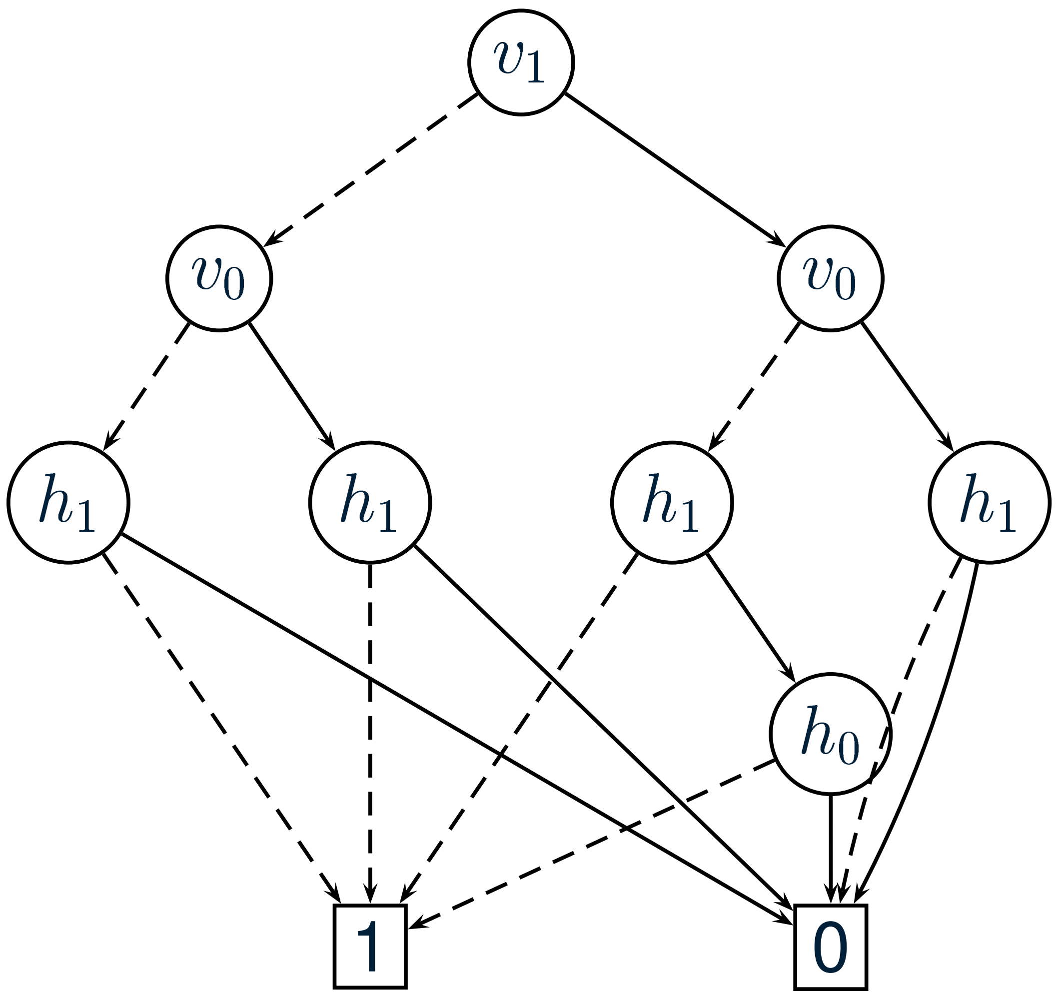
BDD representation



a/X $\rightarrow$ 00
b/Y $\rightarrow$ 01
c/Z $\rightarrow$ 10

	V	H
	$v_1 v_0$	$h_1 h_0$
(a,X)	00	00
(a,Y)	00	01
(b,X)	01	00
(b,Y)	01	01
(c,X)	10	00
(c,Y)	10	01
(c,Z)	10	10

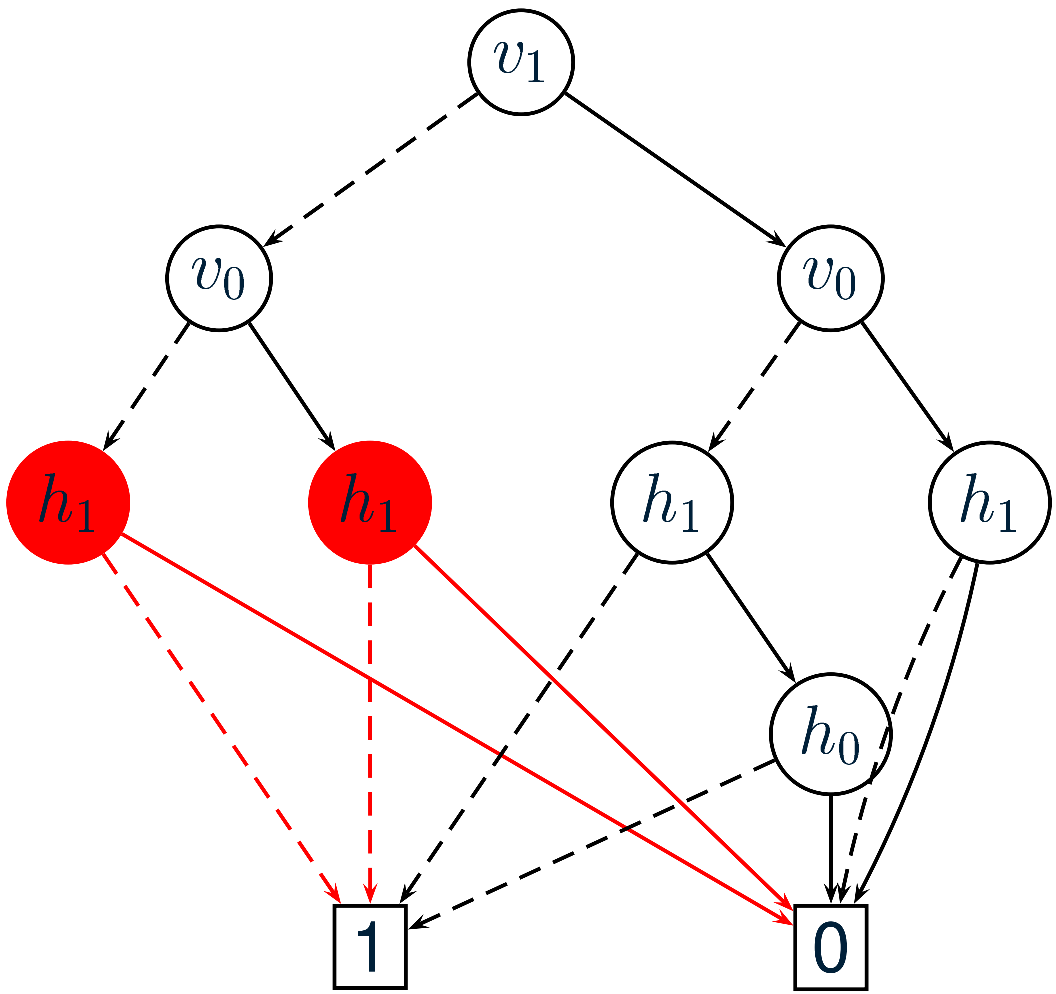
BDD representation



a/X $\rightarrow$ 00
b/Y $\rightarrow$ 01
c/Z $\rightarrow$ 10

	V	H
	$v_1 v_0$	$h_1 h_0$
(a,X)	00	00
(a,Y)	00	01
(b,X)	01	00
(b,Y)	01	01
(c,X)	10	00
(c,Y)	10	01
(c,Z)	10	10

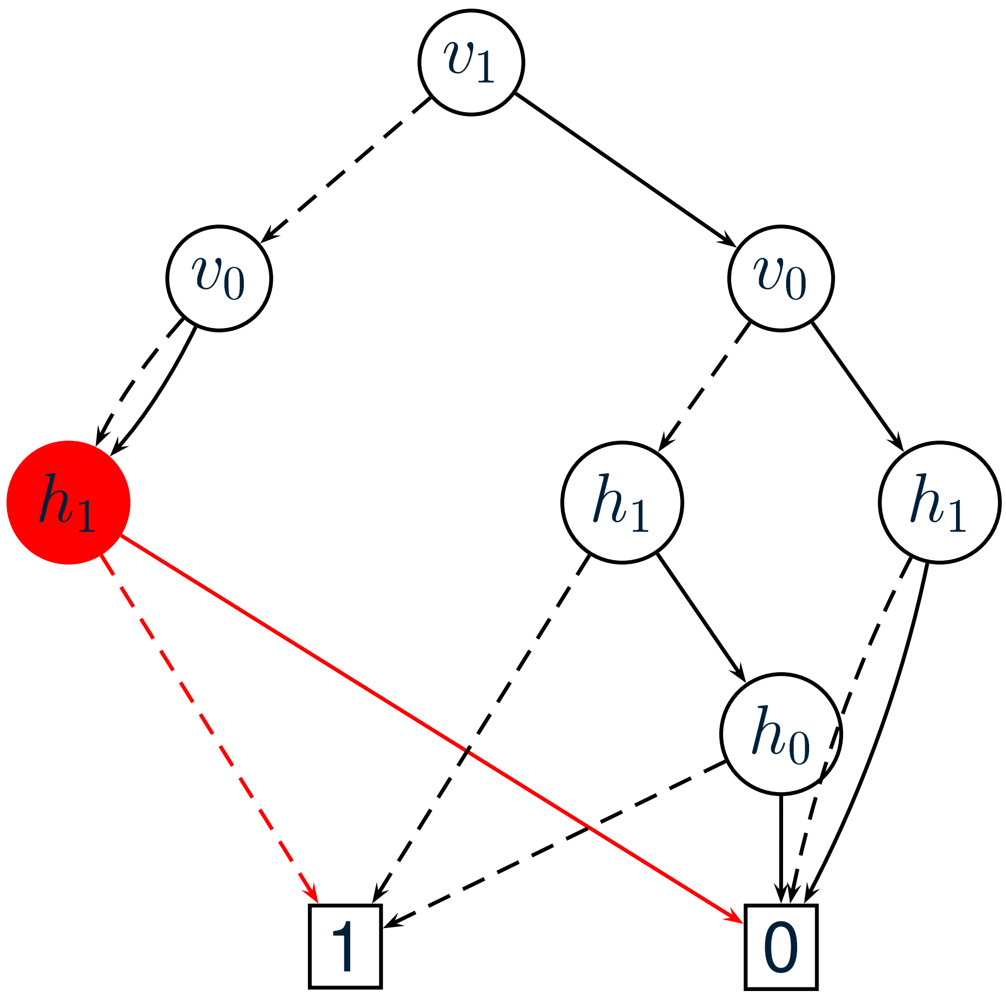
BDD representation



a/X $\rightarrow$ 00
b/Y $\rightarrow$ 01
c/Z $\rightarrow$ 10

	V	H
	$v_1 v_0$	$h_1 h_0$
(a,X)	00	00
(a,Y)	00	01
(b,X)	01	00
(b,Y)	01	01
(c,X)	10	00
(c,Y)	10	01
(c,Z)	10	10

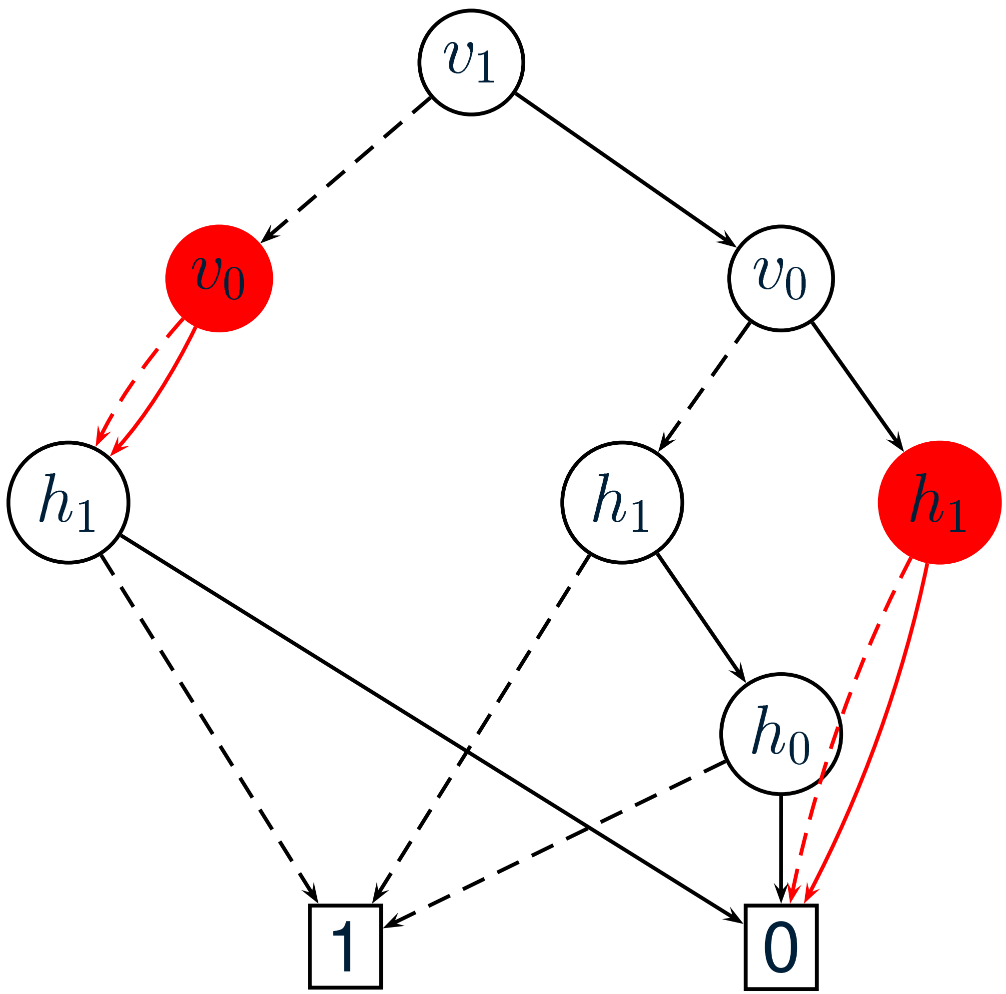
BDD representation



a/X $\rightarrow$ 00
b/Y $\rightarrow$ 01
c/Z $\rightarrow$ 10

	V	H
	$v_1 v_0$	$h_1 h_0$
(a,X)	00	00
(a,Y)	00	01
(b,X)	01	00
(b,Y)	01	01
(c,X)	10	00
(c,Y)	10	01
(c,Z)	10	10

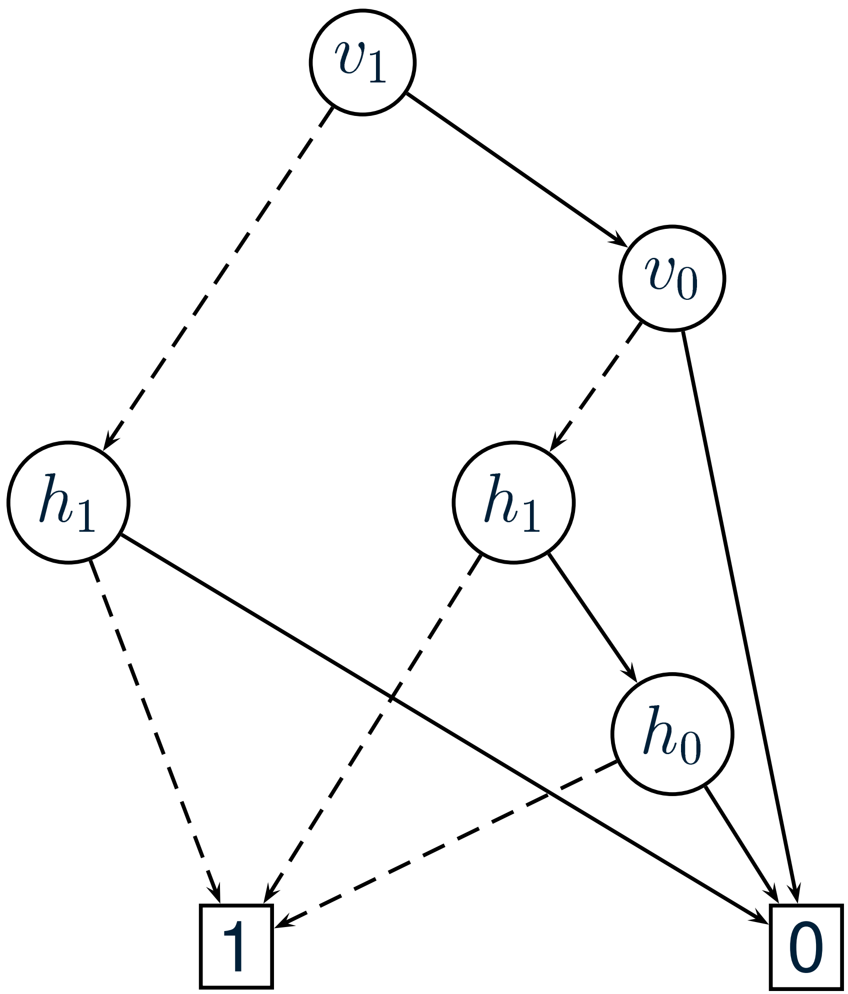
BDD representation



a/X $\rightarrow$ 00
b/Y $\rightarrow$ 01
c/Z $\rightarrow$ 10

	V H
	$v_1 v_0 h_1 h_0$
(a,X)	00 00
(a,Y)	00 01
(b,X)	01 00
(b,Y)	01 01
(c,X)	10 00
(c,Y)	10 01
(c,Z)	10 10

Reduced BDD representation

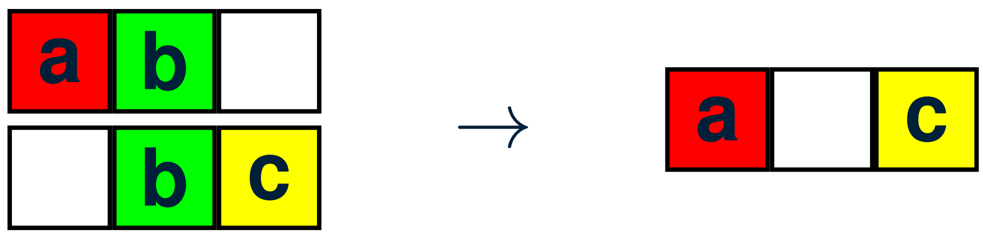


a/X $\rightarrow$ 00
b/Y $\rightarrow$ 01
c/Z $\rightarrow$ 10

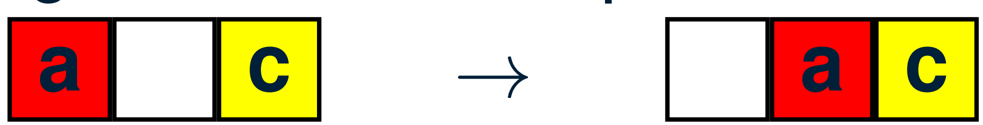
	V	H
	$v_1 v_0$	$h_1 h_0$
(a,X)	00	00
(a,Y)	00	01
(b,X)	01	00
(b,Y)	01	01
(c,X)	10	00
(c,Y)	10	01
(c,Z)	10	10

BDD operations

- Set operations ($\cup, \cap, \setminus, \ldots$)
- Relational product
  $(\{(a, c) \mid \exists b.(a, b) \in X \wedge (b, c) \in Y\})$



- Replace –
  changing bit order in a specific BDD



- Cost of operations proportional to number of nodes in BDD, not size of set represented

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b)
(c,Z) (b $\rightarrow$ c)

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	
H1	X Y Z		

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b)
(c,Z) (b $\rightarrow$ c)

relprod

Domains	Points-to			Edges			New points-to
V1	a	b	c	b	a	b	
V2				a	b	c	
H1	X	Y	Z				

Propagating points-to sets

```
X: a = new O();        a = b;
Y: b = new O();        b = a;
Z: c = new O();        c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b)
(c,Z) (b $\rightarrow$ c)

relprod

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b
H1	X Y Z		X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $(b \to a)$
(b,Y) $(a \to b)$
(c,Z) $(b \to c)$

relprod

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b
H1	X Y Z		X

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X)	(b $\rightarrow$ a)
(b,Y)	(a $\rightarrow$ b)
(c,Z)	(b $\rightarrow$ c)

relprod

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b
H1	X Y Z		X

Propagating points-to sets

```
X: a = new O();        a = b;
Y: b = new O();        b = a;
Z: c = new O();        c = b;
```

(a,X) (b $\rightarrow$ a)
(b,Y) (a $\rightarrow$ b)
(c,Z) (b $\rightarrow$ c)

relprod

Domains	Points-to			Edges			New points-to		
V1	a	b	c	b	a	b			
V2				a	b	c	b	a	c
H1	X	Y	Z				X	Y	Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X)	(b $\rightarrow$ a)
(b,Y)	(a $\rightarrow$ b)
(c,Z)	(b $\rightarrow$ c)

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b a c
H1	X Y Z		X Y Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $(b \to a)$
(b,Y) $(a \to b)$
(c,Z) $(b \to c)$

replace

Domains	Points-to	Edges	New points-to
V1	a b c	b a b	
V2		a b c	b a c
H1	X Y Z		X Y Y

Propagating points-to sets

```
X: a = new O();        a = b;
Y: b = new O();        b = a;
Z: c = new O();        c = b;
```

(a,X) $(b \to a)$
(b,Y) $(a \to b)$
(c,Z) $(b \to c)$

replace

Domains	Points-to			Edges			New points-to		
V1	a	b	c	b	a	b	b	a	c
V2				a	b	c			
H1	X	Y	Z				X	Y	Y

Propagating points-to sets

```
X: a = new O();        a = b;
Y: b = new O();        b = a;
Z: c = new O();        c = b;
```

(a,X) $(b \to a)$
(b,Y) $(a \to b)$
(c,Z) $(b \to c)$

Domains	Points-to			Edges			New points-to		
V1	a	b	c	b	a	b	b	a	c
V2				a	b	c			
H1	X	Y	Z				X	Y	Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) $(b \to a)$
(b,Y) $(a \to b)$
(c,Z) $(b \to c)$

union

Domains	Points-to			Edges			New points-to		
V1	a	b	c	b	a	b	b	a	c
V2				a	b	c			
H1	X	Y	Z				X	Y	Y

Propagating points-to sets

```
X: a = new O();          a = b;
Y: b = new O();          b = a;
Z: c = new O();          c = b;
```

(a,X) (b,Y) (c,Z)

($b \rightarrow a$) ($a \rightarrow b$) ($b \rightarrow c$)

union

Domains	Points-to						Edges			New
V1	a	b	c	b	a	c	b	a	b	
V2							a	b	c	
H1	X	Y	Z	X	Y	Y				

Talk Outline

- Introduction
  - Points-to analysis
  - BDDs
- BDD-PTA algorithm
- Performance tuning
  - Bit ordering
  - Incrementalization
- Overall performance
- Conclusions and future work

BDDs used

- $edgeset \subseteq V1 \times V2$
  simple assignments
  (l_2 := l_1)
- $stores \subseteq V1 \times (V2 \times FD)$
  field stores
  ($l_2.f$:= l_1)
- $loads \subseteq (V1 \times FD) \times V2$
  field loads
  (l_2 := $l_1.f$)
- $pointsTo \subseteq V1 \times H1$
  points-to relation for variables
  (l points to o)
- $fieldPt \subseteq (H1 \times FD) \times H2$
  points-to relation for object fields
  ($o_1.f$ points to o_2)

- 5 domains needed: $V1, V2, H1, H2, FD$

Overall algorithm

```
initialize
repeat
   repeat
      (1) process simple assignments
   until no change
   (2) process field stores
   (3) process field loads
until no change
```

Simple assignments (l_2 := l_1)

(1)
$$\frac{l_1 \to l_2 \quad o \in pt(l_1)}{o \in pt(l_2)}$$

```
newPt1:   [V2xH1] =
          relprod( edgeSet:  [V1xV2],
                   pointsTo:[V1xH1],
                            V1 );
newPt2:   [V1xH1] =
          replace( newPt1:   [V2xH1],
                            V2ToV1 );
pointsTo:[V1xH1] =
          union(   pointsTo:[V1xH1],
                   newPt2:   [V1xH1] );
```

Field stores ($q.f$:= l)

(2) $$\frac{o_2 \in pt(l) \quad l \to q.f \quad o_1 \in pt(q)}{o_2 \in pt(o_1.f)}$$

```
tmpRel1:[(V2xFD)xH1] =
    relprod( stores:   [V1x(V2xFD)],
             pointsTo:[V1xH1],
                       V1 );
tmpRel2:[(V1xFD)xH2] =
    replace( tmpRel1: [(V2xFD)xH1],
                       V2ToV1&H1ToH2);
fieldPt:[(H1xFD)xH2] =
    relprod( tmpRel2: [(V1xFD)xH2],
             pointsTo:[V1xH1],
                       V1 );
```

Field loads (l := $p.f$)

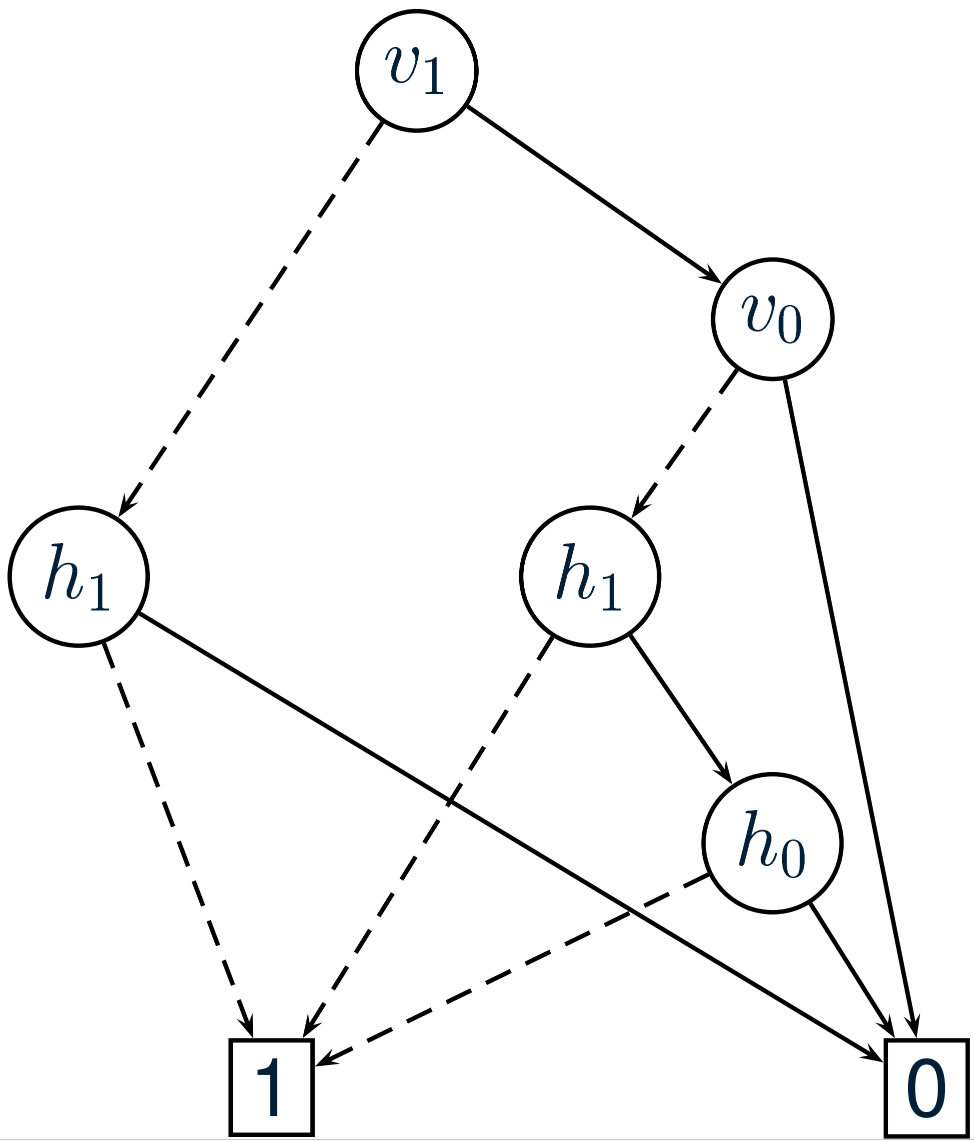
(3) $$\frac{p.f \to l \quad o_1 \in pt(p) \quad o_2 \in pt(o_1.f)}{o_2 \in pt(l)}$$

```
tmpRel3:     [(H1xFD)xV2] =
    relprod( loads:    [(V1xFD)xV2],
             pointsTo:[V1xH1],
                       V1 );
newPt4:      [V2xH2] =
    relprod( tmpRel3: [(H1xFD)xV2],
             fieldPt: [(H1xFD)xH2],
                       H1xFD );
newPt5:      [V1xH1] =
    replace( newPt4:   [V2xH2],
                       V2ToV1&H2ToH1);
```

Talk Outline

- Introduction
  - Points-to analysis
  - BDDs
- BDD-PTA algorithm
- Performance tuning
  - Bit ordering
  - Incrementalization
- Overall performance
- Conclusions and future work

Bit ordering matters

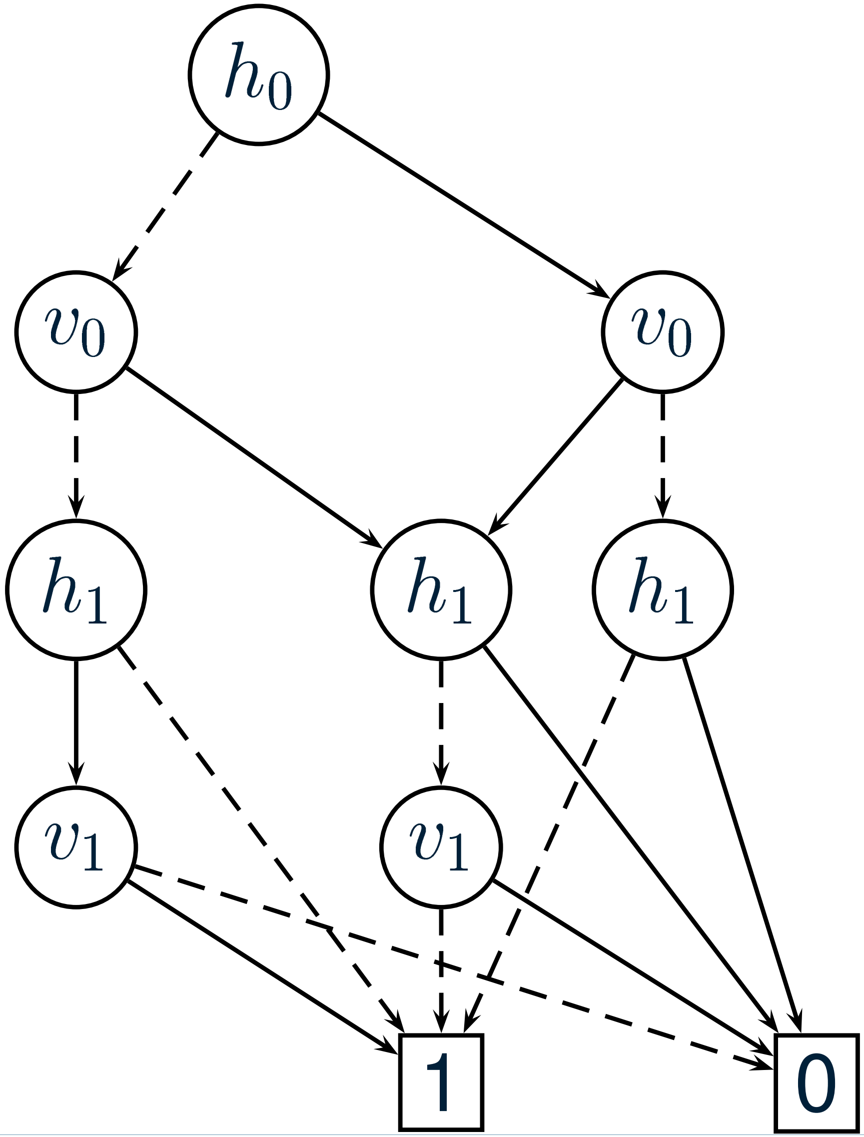


a/X $\rightarrow$ 00
b/Y $\rightarrow$ 01
c/Z $\rightarrow$ 10

$v_1 v_0 h_1 h_0$

(a,X) 0000
(a,Y) 0001
(b,X) 0100
(b,Y) 0101
(c,X) 1000
(c,Y) 1001
(c,Z) 1010

Bit ordering matters

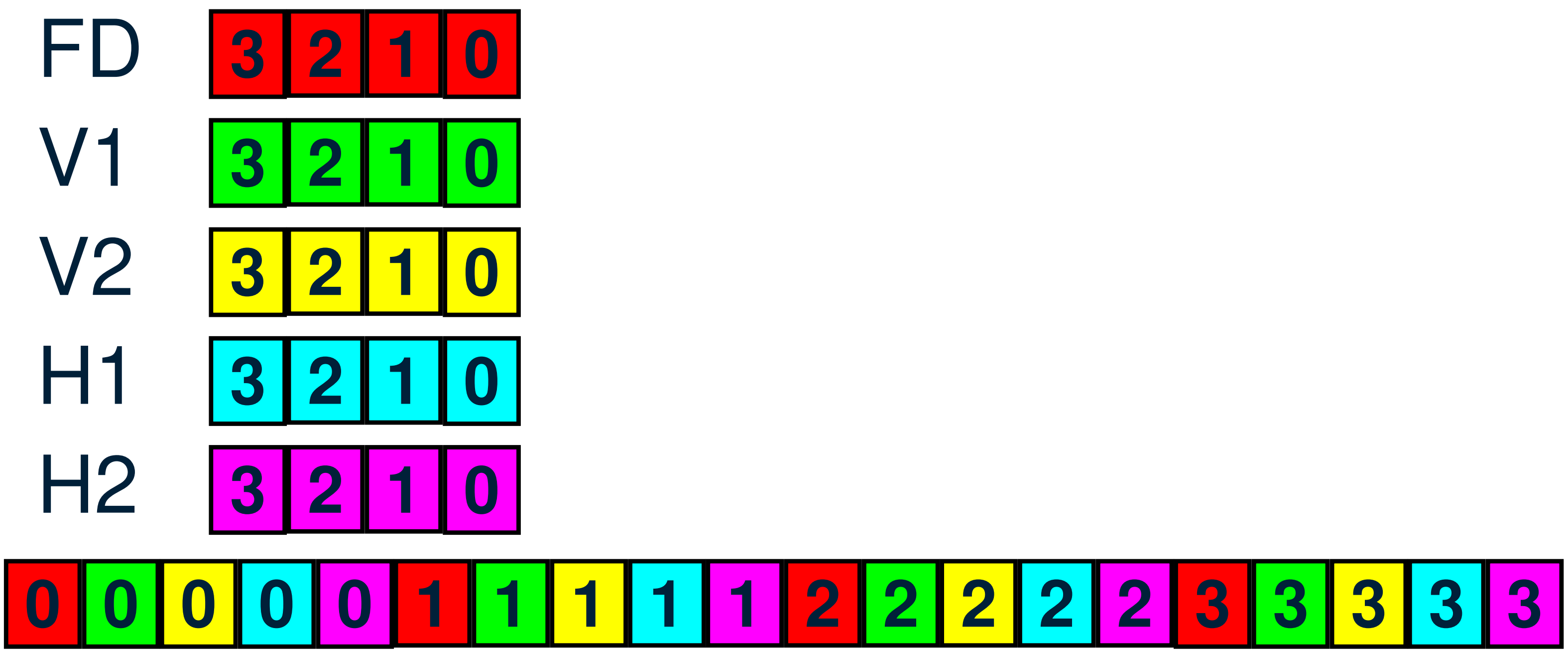


a/X $\rightarrow$ 00
b/Y $\rightarrow$ 10
c/Z $\rightarrow$ 01

	$h_0 v_0 h_1 v_1$
(a,X)	0000
(a,Y)	1000
(b,X)	0100
(b,Y)	1100
(c,X)	0001
(c,Y)	1001
(c,Z)	0011

How to find a good ordering?

- BuDDy default is to interleave bits:



- Good heuristic for state machines in model checking
- Bad for points-to analysis: much too slow!

How to find a good ordering?

Where is most of the time spent?

(1) $$\frac{l_1 \to l_2 \quad o \in pt(l_1)}{o \in pt(l_2)}$$

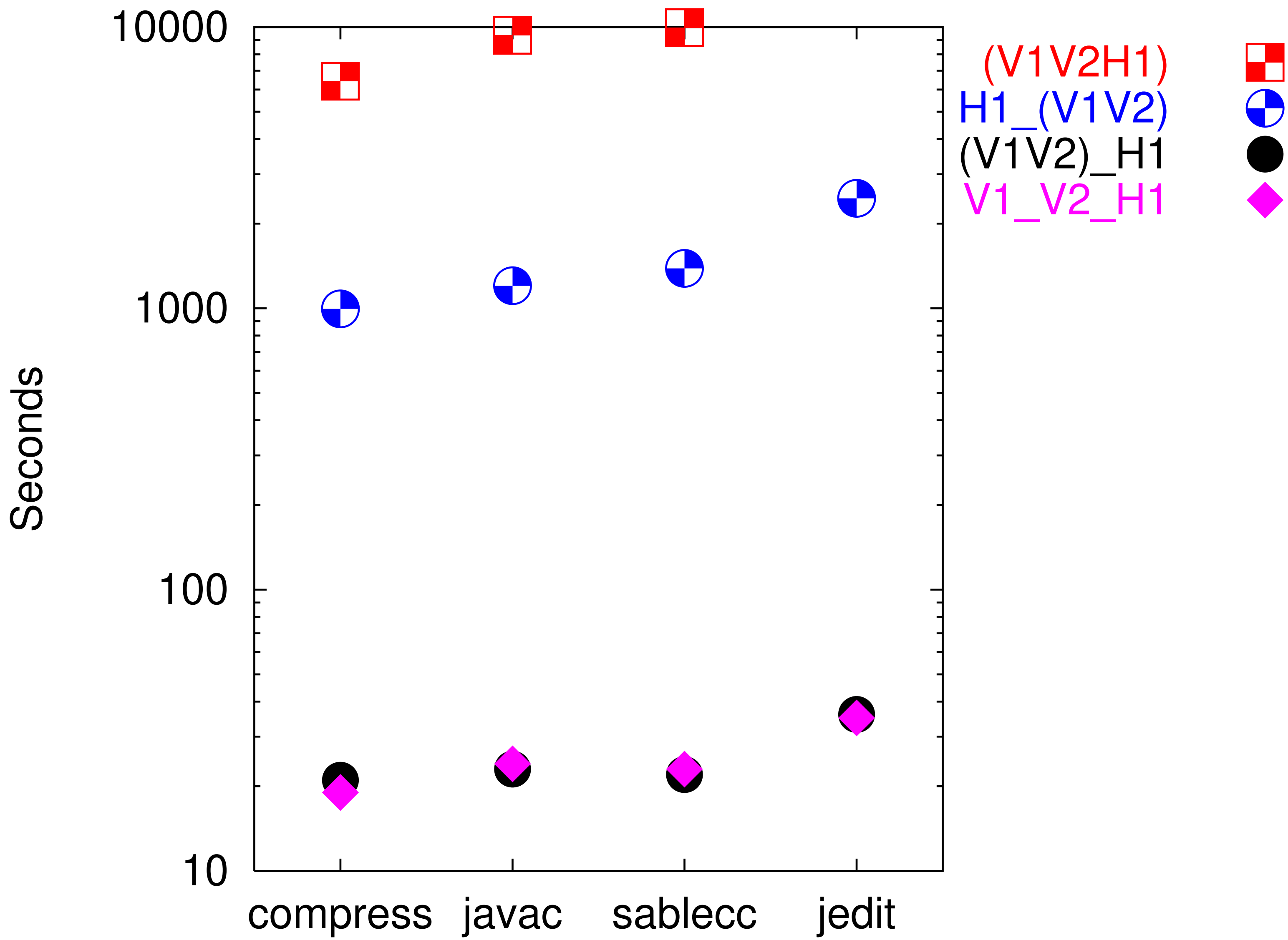
```
newPt1:  [V2xH1] =
         relprod( edgeSet: [V1xV2],
                  pointsTo:[V1xH1],
                           V1 );
newPt2:  [V1xH1] =
         replace( newPt1:  [V2xH1],
                           V2ToV1 );
```

V1, V2, H1 make a difference; H2, FD do not.

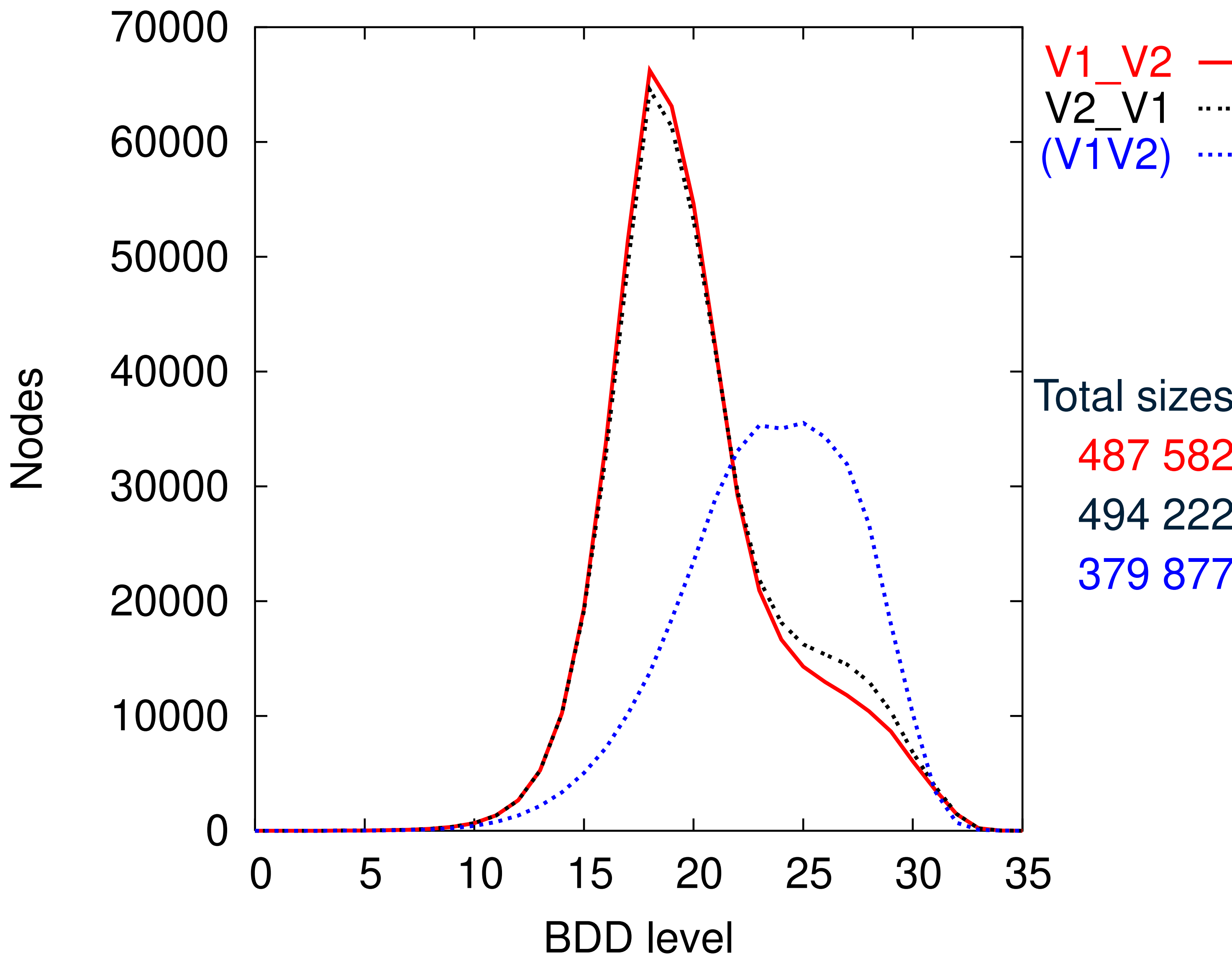
How to find a good ordering?

- Idea:
  - H1 represents points-to sets (large, regular)
  - Put it at the end $\Rightarrow$ big speedup!
- What about V1 and V2?
  - Interleaving them is usually a bit faster than one before the other

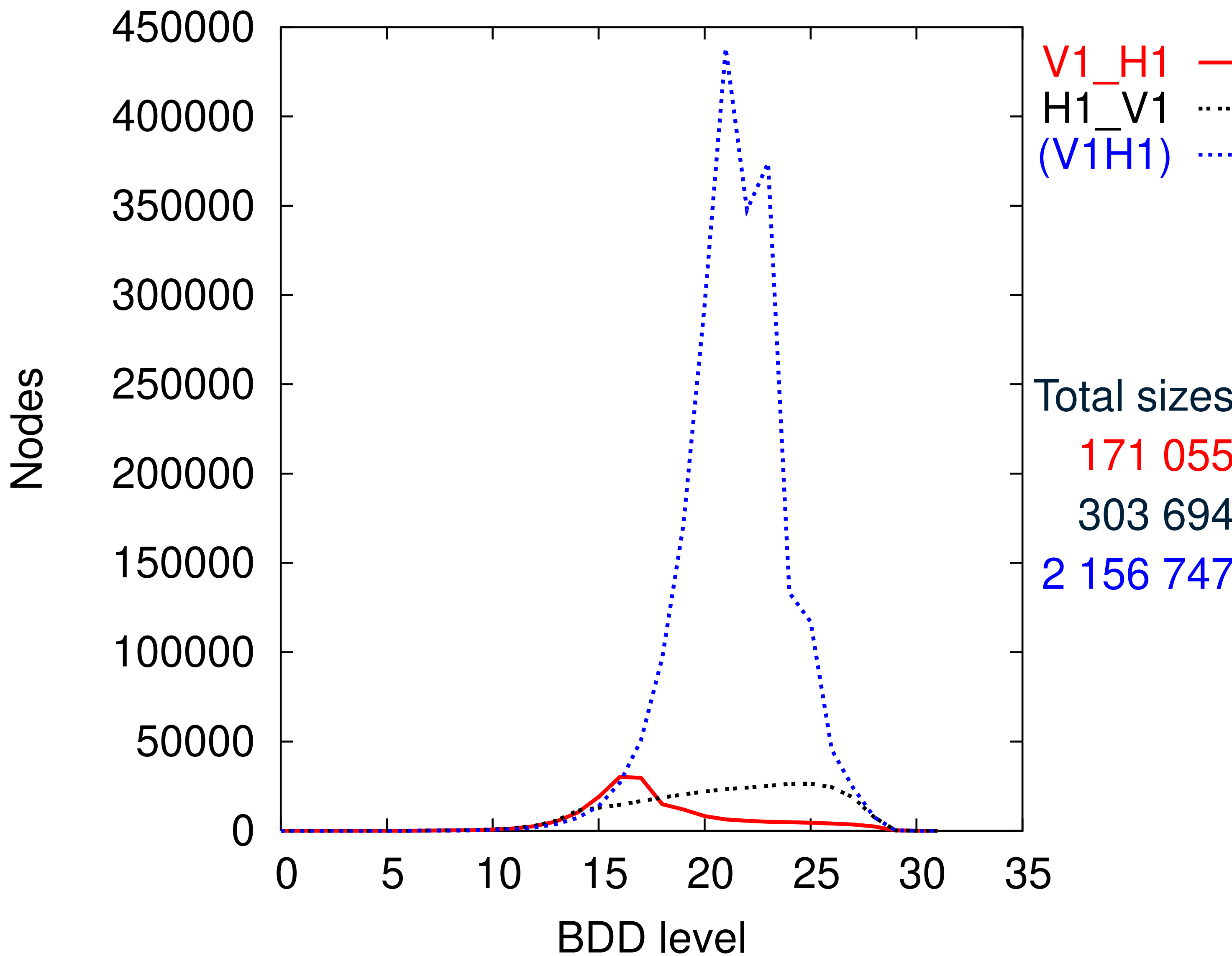
Performance of different orderings



Effect of ordering on $edgeSet$



Effect of ordering on $pointsTo$



Incrementalization

- All sets are re-propagated in each iteration
- Could we propagate only the new elements of each set?
- We found this to work well for Spark
- How well would it work in BDDs?

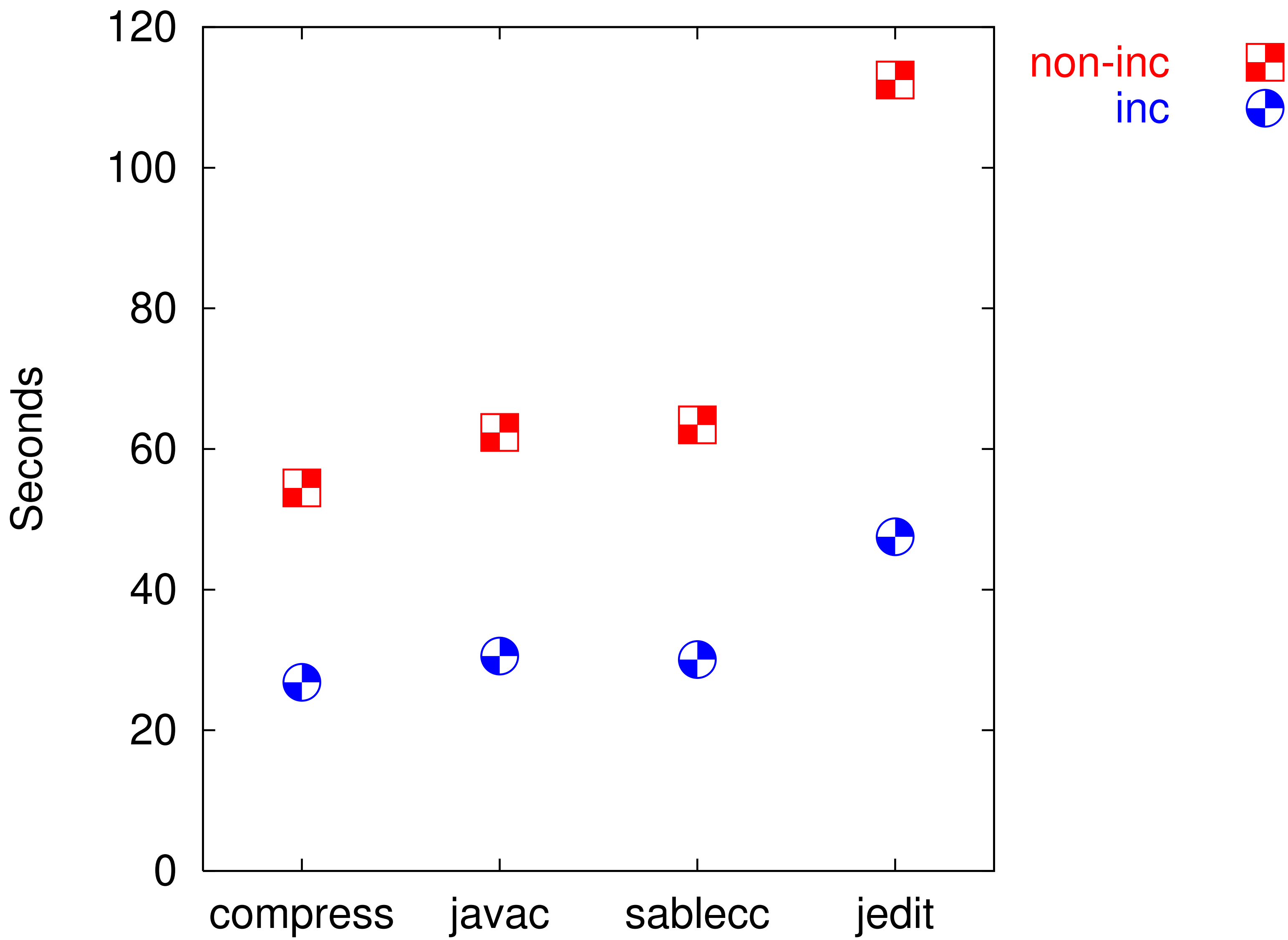
Incrementalization

```
newPt1:   [V2xH1] =
          relprod( edgeSet: [V1xV2],
                   pointsTo:[V1xH1],
                            V1 );
newPt2:   [V1xH1] =
          replace( newPt1:   [V2xH1],
                             V2ToV1 );



pointsTo:[V1xH1] =
          union(   pointsTo:[V1xH1],
                   newPt2:   [V1xH1] );
```

Incrementalization

```
newPt1:   [V2xH1] =
          relprod( edgeSet:  [V1xV2],
                   newPoint:[V1xH1],
                            V1 );
newPt2:   [V1xH1] =
          replace( newPt1:   [V2xH1],
                             V2ToV1 );
newPoint:[V1xH1] =
          setminus( newPt2:  [V1xH1],
                    pointsTo:[V1xH1] );
pointsTo:[V1xH1] =
          union(   pointsTo:[V1xH1],
                   newPoint:[V1xH1] );
```

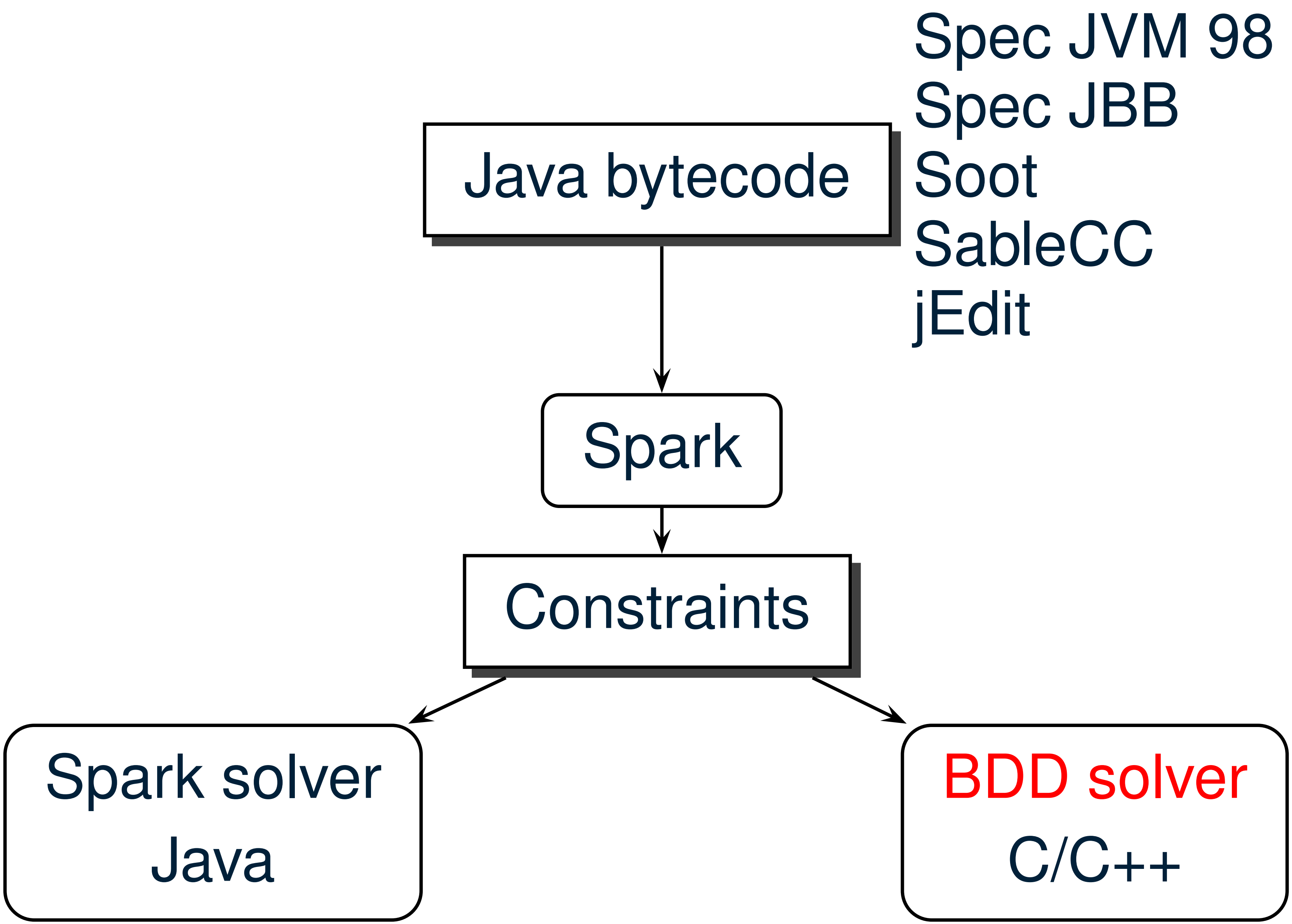
Incrementalization



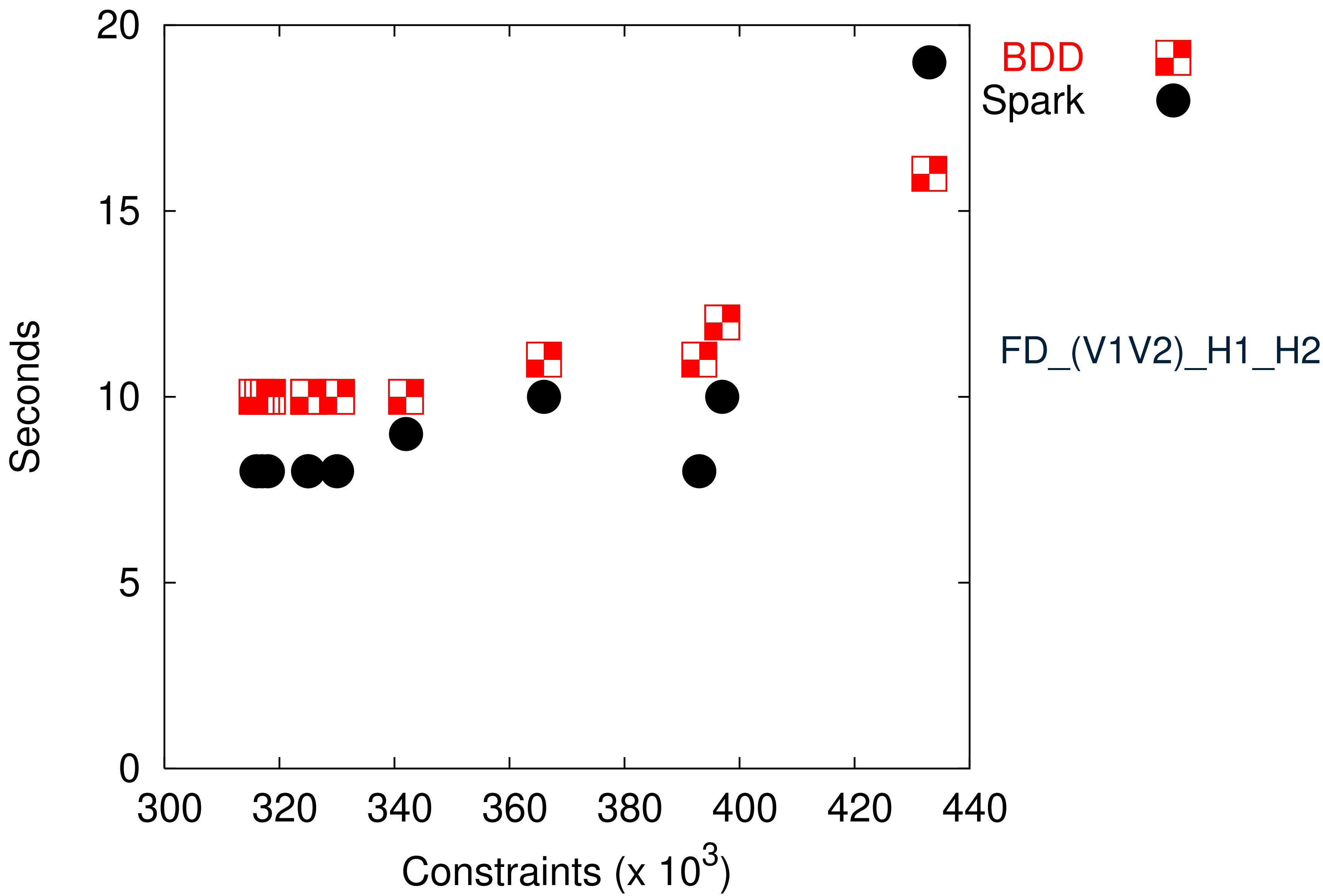
Talk Outline

- Introduction
  - Points-to analysis
  - BDDs
- BDD-PTA algorithm
- Performance tuning
  - Bit ordering
  - Incrementalization
- Overall performance
- Conclusions and future work

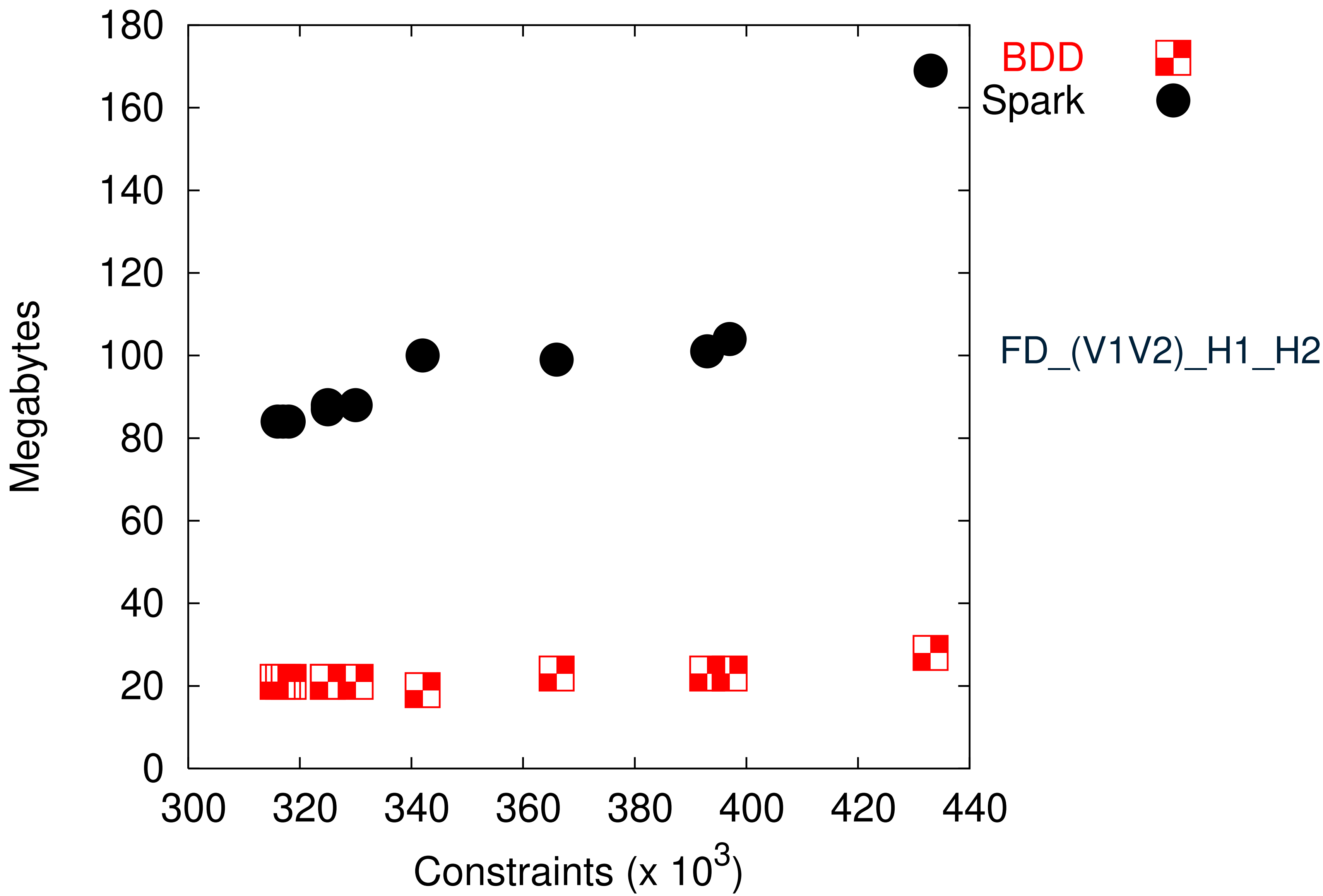
Experiment setup



Overall performance (time)



Overall performance (space)



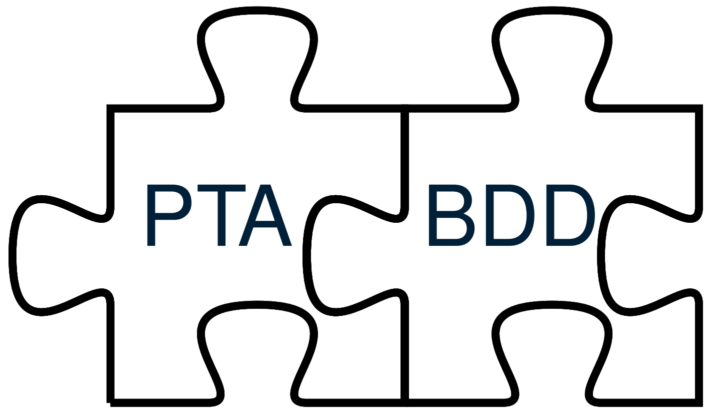
Solving without declared types

- In Java, use declared types of variables to keep points-to sets small
- Without declared types, large sets, traditional solvers do not scale
- May not have declared types (IR does not support them; language dynamically typed)
- Surprisingly, BDD-based solver scales well even without declared types

eg. javac:

	Set size	BDD size
with types	21M	31MB
no types	366M	40MB

Conclusions



- BDDs are a good fit for points-to analysis
- BDDs give reasonably efficient solvers with relatively little effort
- BDDs make it easy to experiment with variations of set-based problems
- Bit ordering is crucial (and we found a good one for points-to analysis)

Future Work

- More heuristics for BDD program analysis
- Library for program analysis using BDDs
- Variations on the points-to analysis
  - Context-sensitivity
- Compute other whole-program information
  - Call graph
  - Interprocedural side-effect analysis
  - . . . (suggestions?)