

# Context-Sensitive Program Analysis as Database Queries

Monica Lam

Stanford University

Team: John Whaley, Ben Livshits, Michael Martin,  
Dzintars Avots, Michael Carbin, Chris Unkel

# State-of-the-Art Programming Tools

- Emacs
  - Grep
- IDE: integrated program development environment (e.g. Eclipse)
  - Smarter syntactic searches
- What programmers want:
  - Information about dynamic behavior
  - Compiler (data-flow) analysis

# PQL: Program Query Language

User: Queries on dynamic behavior of programs

PQL: Resolve with static (and dynamic) analyses

|                    |                   |
|--------------------|-------------------|
| Important problems | Database security |
|--------------------|-------------------|

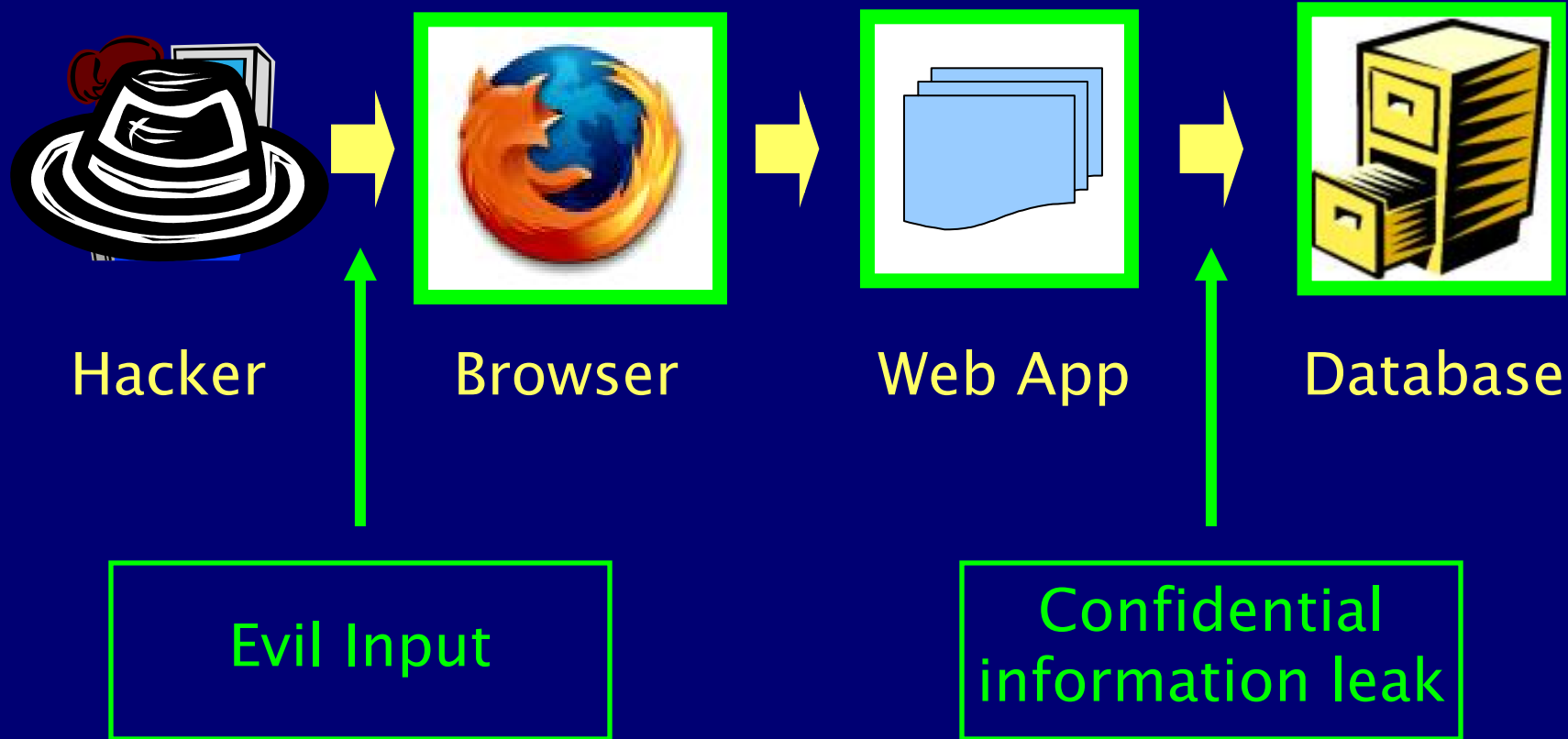
|              |                             |
|--------------|-----------------------------|
| Easy queries | PQL (declarative) → Datalog |
|--------------|-----------------------------|

|               |                      |
|---------------|----------------------|
| Deep analyses | A deductive database |
|---------------|----------------------|

|                  |                                   |
|------------------|-----------------------------------|
| Accurate answers | Sound:all errors, few false warn. |
|------------------|-----------------------------------|

Hard Important Problems

# Web Applications



# Web Application Vulnerabilities

- 48% of all vulnerabilities Q3–Q4, 2004
  - Up from 39% Q1–Q2, 04  
[Symantec May 21, 2005]
- 50% databases had a security breach  
[2002 Computer crime & security survey]

# Top Ten Security Flaws in Web Applications [OWASP]

1. Unvalidated Input
2. Broken Access Control
3. Broken Authentication and Session Management
4. Cross Site Scripting (XSS) Flaws
5. Buffer Overflows
6. Injection Flaws
7. Improper Error Handling
8. Insecure Storage
9. Denial of Service
10. Insecure Configuration Management

# Vulnerability Alerts

- SecurityFocus.com, on May 16, 2005

- 2005-05-16: JGS-Portal Multiple Cross-Site Scripting and **SQL Injection Vulnerabilities**
- 2005-05-16: WoltLab Burning Board Verify\_email Function **SQL Injection Vulnerability**
- 2005-05-16: Version Cue Local Privilege Escalation Vulnerability
- 2005-05-16: NPDS THOLD Parameter **SQL Injection Vulnerability**
- 2005-05-16: DotNetNuke User Registration Information **HTML Injection Vulnerability**
- 2005-05-16: Pserv completedPath **Remote Buffer Overflow Vulnerability**
- 2005-05-16: DotNetNuke User-Agent String Application Logs **HTML Injection Vulnerability**
- 2005-05-16: Dd
- 2005-05-16: Md
- 2005-05-16: Si
- 2005-05-16: Md
- 2005-05-16: PS
- 2005-05-16: PS
- 2005-05-16: Ps
- 2005-05-16: Me
- 2005-05-16: W
- 2005-05-16: Op
- 2005-05-16: Pc
- 2005-05-16: Me
- 2005-05-16: Me
- 2005-05-16: Sh
- 2005-05-16: Sh
- 2005-05-16: SV
- 2005-05-16: PC
- 2005-05-16: Ap
- 2005-05-16: Squid Proxy Unspecified DNS Spoofing Vulnerability
- 2005-05-16: Linux Kernel ELF Core Dump Local **Buffer Overflow Vulnerability**
- 2005-05-16: Gaim Jabber File Request Remote Denial Of Service Vulnerability
- 2005-05-16: Gaim IRC Protocol Plug-in **Markup Language Injection Vulnerability**
- 2005-05-16: Gaim Gaim\_Markup\_Strip\_HTML Remote Denial Of Service Vulnerability
- 2005-05-16: GDK-Pixbuf BMP Image Processing Double Free Remote Denial of Service Vulnerability
- 2005-05-16: Mozilla Firefox Install Method Remote Arbitrary Code Execution Vulnerability
- 2005-05-16: Multiple Vendor FTP Client Side File Overwriting Vulnerability
- 2005-05-16: PostgreSQL TSearch2 Design Error Vulnerability
- 2005-05-16: PostgreSQL Character Set Conversion Privilege Escalation Vulnerability

## Source of vulnerabilities

Input validation: 62%  
SQL injection: 26%

# SQL Injection Errors



Hacker

Browser

Web App

Database

Give me Bob's credit card #  
Delete all records

# Happy-go-lucky SQL Query

User supplies: *name*, *password*

Java program:

String query =

*"SELECT UserID, Creditcard FROM CCRec  
WHERE Name = "*

*+ name + " AND PW = "*

*+ password*

# Fun with SQL

“ — ”: “the rest are comments” in Oracle SQL

```
SELECT UserID, CreditCard FROM CCRec
```

WHERE:

```
Name = bob AND PW = foo
```

```
Name = bob— AND PW = x
```

```
Name = bob or 1=1— AND PW = x
```

```
Name = bob; DROP CCRec— AND PW = x
```

# A Simple SQL Injection

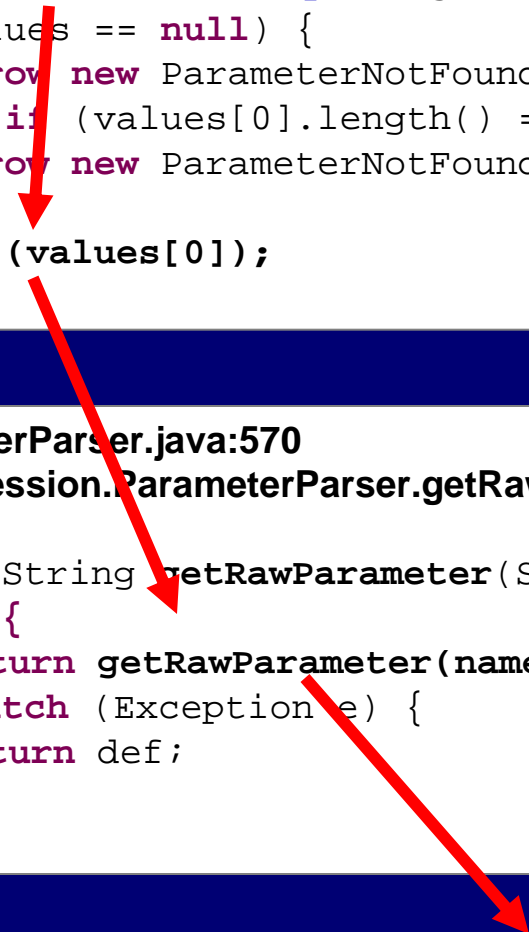
```
o = req.getParameter ( );  
stmt.executeQuery ( o );
```

# In Practice

ParameterParser.java:586

String session.ParameterParser.getRawParameter(String name)

```
public String getRawParameter(String name)
    throws ParameterNotFoundException {
    String[] values = request.getParameterValues(name);
    if (values == null) {
        throw new ParameterNotFoundException(name + " not found");
    } else if (values[0].length() == 0) {
        throw new ParameterNotFoundException(name + " was empty");
    }
    return (values[0]);
}
```



ParameterParser.java:570

String session.ParameterParser.getRawParameter(String name, String def)

```
public String getRawParameter(String name, String def) {
    try {
        return getRawParameter(name);
    } catch (Exception e) {
        return def;
    }
}
```

# In Practice (II)

ChallengeScreen.java:194

Element lessons.ChallengeScreen.doStage2(WebSession s)

```
String user = s.getParser().getRawParameter( USER, "" );
StringBuffer tmp = new StringBuffer();
tmp.append("SELECT cc_type, cc_number from user_data
WHERE useric = '");
tmp.append(user);
tmp.append("'");
query = tmp.toString();
Vector v = new Vector();
try
{
    ResultSet results = statement3.executeQuery( query );
    ...
}
```

# PQL

Dynamically:

```
o = req.getParameter ( );  
stmt.executeQuery (o);
```

Statically:

```
p1 = req.getParameter ( );  
stmt.executeQuery (p2);
```

*p*<sub>1</sub> and *p*<sub>2</sub> point to same object?

Pointer alias analysis

# SQL Injection in PQL

```
query SQLInjection()
returns object Object source, taint;
uses object HttpServletRequest req, java.sql.Statement stmt;
matches {
    source = req.getParameter ();
    tainted := derivedString(source);
    stmt.execute(tainted);
}
query derivedString(object Object x)
returns object Object y;
uses object Object temp;
matches {
    y := x
    | { temp.append(x); y := derivedString(temp); }
}
```

# Vulnerabilities in Web Applications

## **Inject**

Parameters

Hidden fields

Headers

Cookie poisoning

**X**

## **Exploit**

SQL injection

Cross-site scripting

HTTP splitting

Path traversal

# Big Picture

|                    |                              |
|--------------------|------------------------------|
| Important problems | Security Auditing, Debugging |
|--------------------|------------------------------|

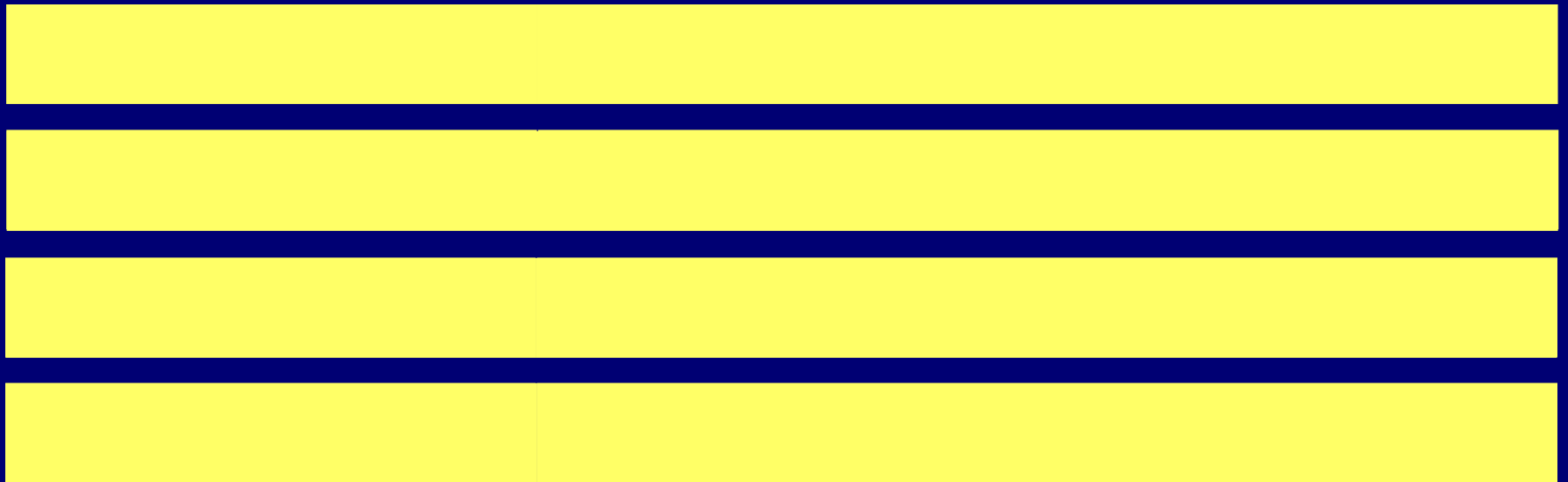
|              |     |
|--------------|-----|
| Easy queries | PQL |
|--------------|-----|

|               |  |
|---------------|--|
| Deep analyses |  |
|---------------|--|

|                  |  |
|------------------|--|
| Accurate answers |  |
|------------------|--|

# Top 4 Techniques in PQL Implementation

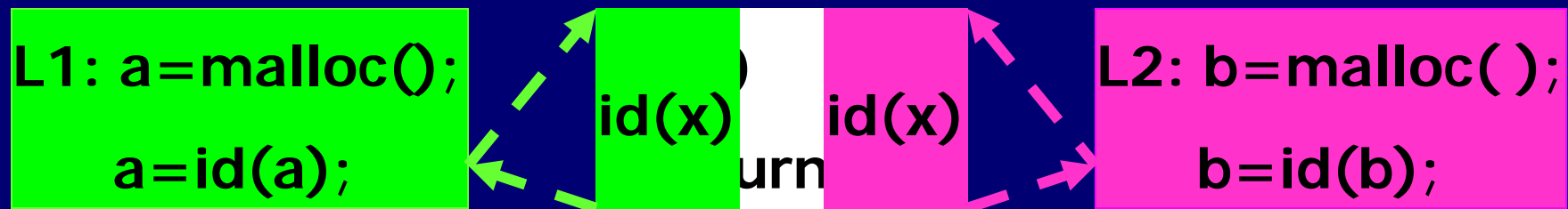
Drawn from 4 different fields



Compiler Context-sensitive pointer analysis

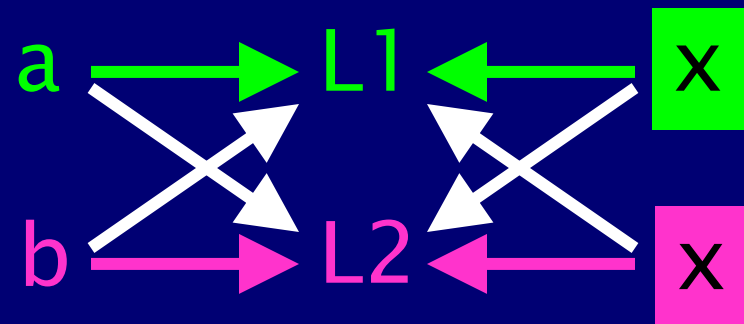
HW Verification BDD: binary decision diagrams

# Context-Sensitive Pointer Analysis

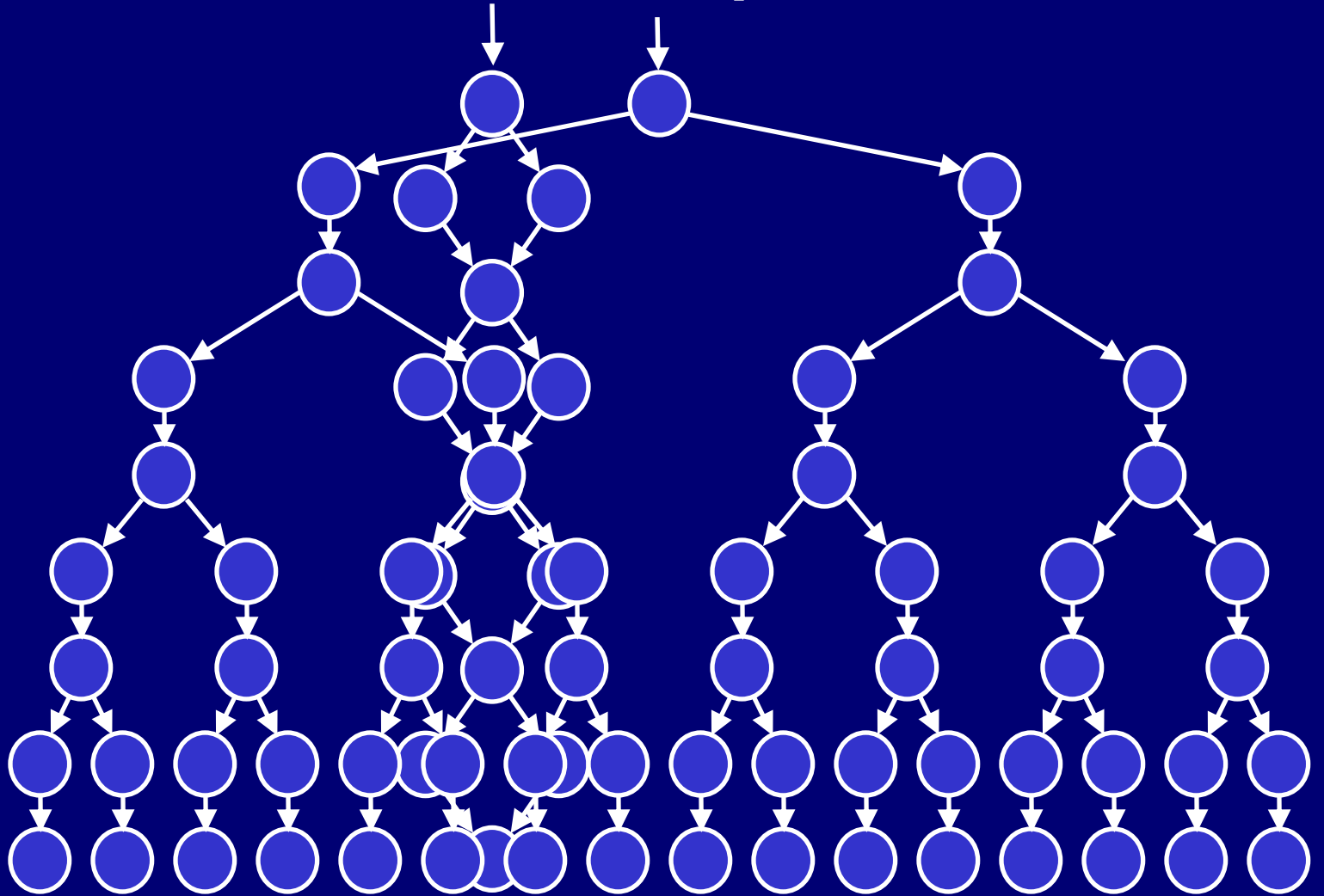


*context-sensitive*

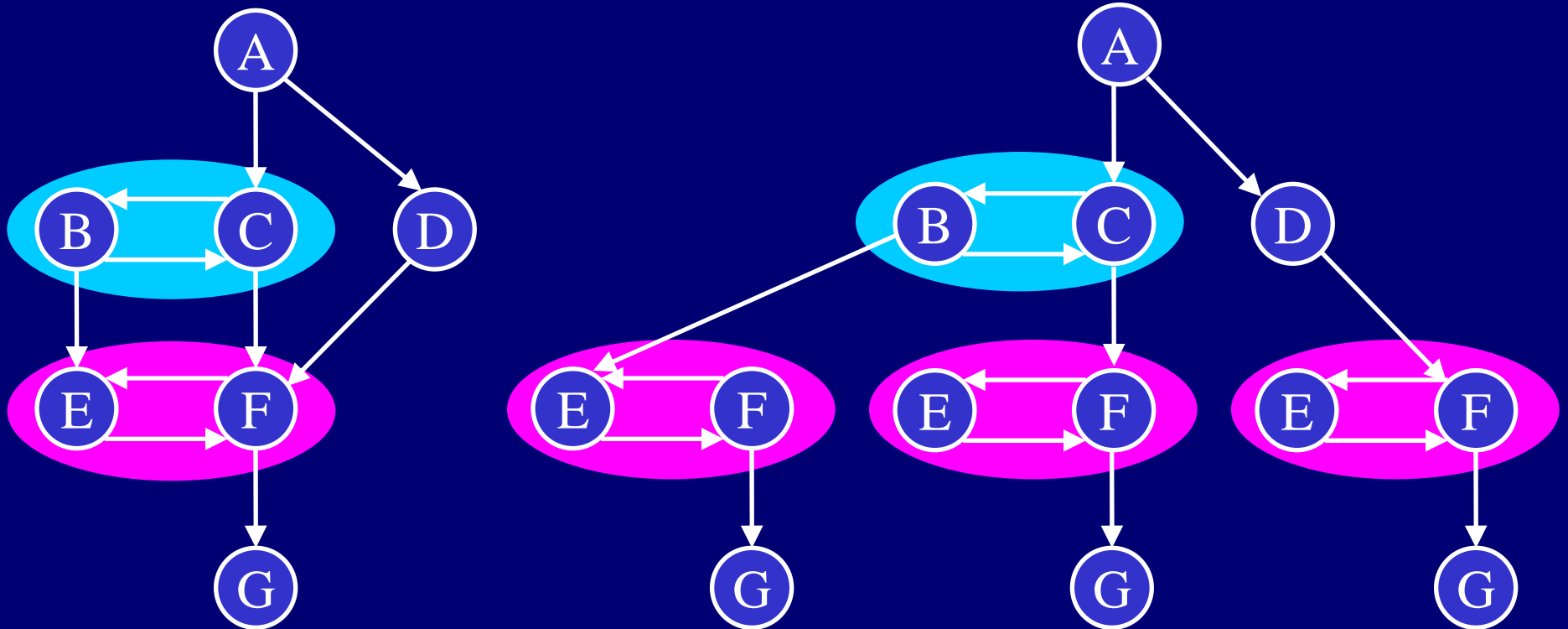
*context-insensitive*



# # of Contexts is exponential!

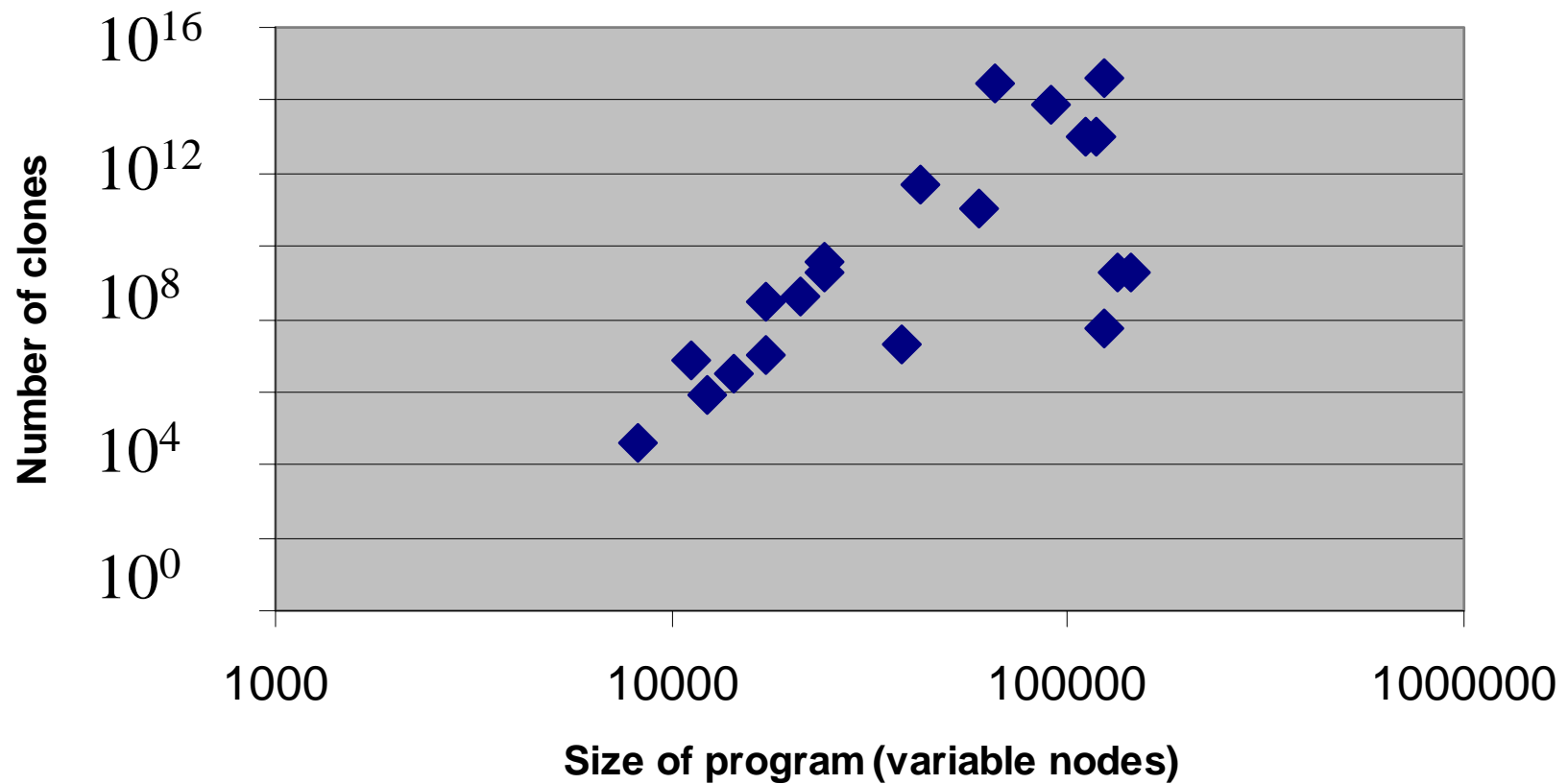


# Recursion



# Top 20 Sourceforge Java Apps

Number of Clones



# Costs of Context Sensitivity

- Typical large program has  $\sim 10^{14}$  paths
- If you need 1 byte to represent a context:
  - 256 terabytes of storage
  - > 12 times size of Library of Congress
  - 1GB DIMMs: \$98.6 million
    - Power: 96.4 kilowatts (128 homes)
  - 300 GB hard disks:  $939 \times \$250 = \$234,750$ 
    - Time to read sequential: 70.8 days

# Cloning-Based Algorithm

- Whaley&Lam, PLDI 2004 (best paper)
- Create a “clone” for every context
- Apply **context-insensitive** algorithm to cloned call graph
- Lots of redundancy in result
- Exploit redundancy by clever use of BDDs (binary decision diagrams)

# Performance of BDD Algorithm

- Direct implementation
  - Does not finish even for small programs
  - > 3000 lines of code
- Requires tuning for about 1 year
- Easy to make mistakes
  - Mistakes found months later

# Automatic Analysis Generation



Ptr analysis in 10 lines

Thousand-lines  
1 year tuning

PQL

Datalog

BDD code

**bddb**  
(**BDD**-based  
**d**eductive **datab**ase)  
with  
Active Machine Learning

Datalog

**bddb**  
(**BDD-based**  
**deductive database**)  
with  
Active Machine Learning

BDD code

# Flow-Insensitive Pointer Analysis

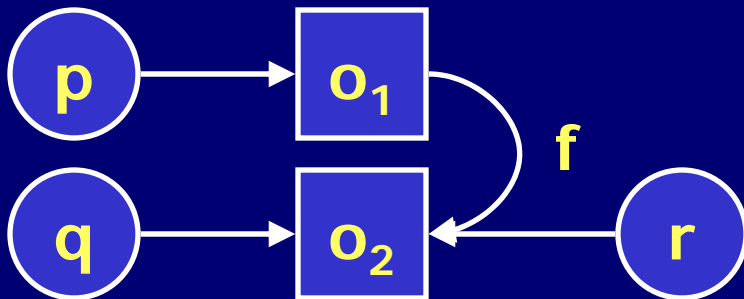


$o_1$ :  $p = \text{new Object}();$

$o_2$ :  $q = \text{new Object}();$

$p.f = q;$

$r = p.f;$



## Input Tuples

$v\text{PointsTo}(p, o_1)$

$v\text{PointsTo}(q, o_2)$

$\text{Store}(p, f, q)$

$\text{Load}(p, f, r)$

## New Tuples

$h\text{PointsTo}(o_1, f, o_2)$

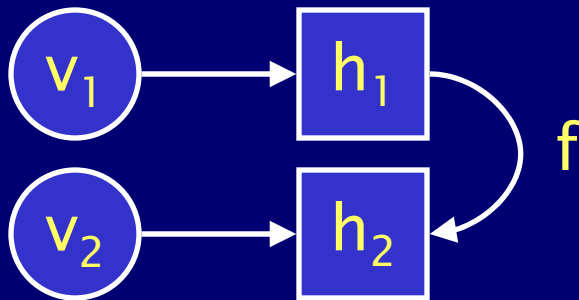
$v\text{PointsTo}(r, o_2)$

# Inference Rule in Datalog

Stores:

$\text{hPointsTo}(h_1, f, h_2) :- \text{Store}(v_1, f, v_2),$   
 $\text{vPointsTo}(v_1, h_1),$   
 $\text{vPointsTo}(v_2, h_2).$

$v_1.f = v_2;$



# Inference Rules

$\text{vPointsTo}(\mathbf{v}, \mathbf{h}) \text{ :- vPointsTo}_0(\mathbf{v}, \mathbf{h}).$

$\text{vPointsTo}(\mathbf{v}_1, \mathbf{h}_1) \text{ :- Assign}(\mathbf{v}_1, \mathbf{v}_2),$   
 $\text{vPointsTo}(\mathbf{v}_2, \mathbf{h}_1).$

$\text{hPointsTo}(\mathbf{h}_1, \mathbf{f}, \mathbf{h}_2) \text{ :- Store}(\mathbf{v}_1, \mathbf{f}, \mathbf{v}_2),$   
 $\text{vPointsTo}(\mathbf{v}_1, \mathbf{h}_1),$   
 $\text{vPointsTo}(\mathbf{v}_2, \mathbf{h}_2).$

$\text{vPointsTo}(\mathbf{v}_2, \mathbf{h}_2) \text{ :- Load}(\mathbf{v}_1, \mathbf{f}, \mathbf{v}_2),$   
 $\text{vPointsTo}(\mathbf{v}_1, \mathbf{h}_1),$   
 $\text{hPointsTo}(\mathbf{h}_1, \mathbf{f}, \mathbf{h}_2).$

# Pointer Alias Analysis

- Specified by a few Datalog rules
  - Creation sites
  - Assignments
  - Stores
  - Loads
- Apply rules until they converge

# SQL Injection Query

PQL:

SQLInjection:

```
o = req.getParameter ();  
stmt.executeQuery ( o );
```

Datalog:

SQLInjection (*o*) :-

```
calls(c1, b1, _, "getParameter"),  
ret(b1, v1), vPointsTo(c1, v1, o),  
calls(c2, b2, _, "executeQuery"),  
actual(b2, 1, v2), vPointsTo(c2, v2, o)
```

# Program Analyses in Datalog

- Context-sensitive Java pointer analysis
- C pointer analysis
- Escape analysis
- Type analysis
- External lock analysis
- Interprocedural def-use
- Interprocedural mod-ref
- Object-sensitive analysis
- Cartesian product algorithm

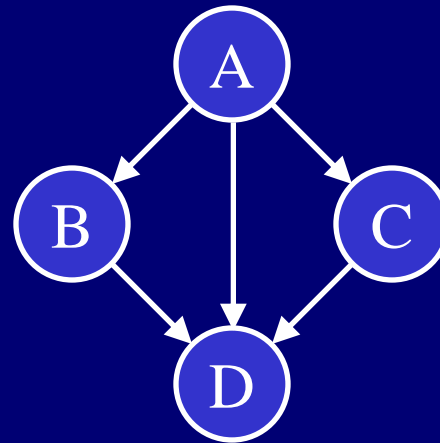
Datalog

**bddb**  
(**BDD**-based  
deductive **database**)  
with  
Active Machine Learning

BDD code

# Example: Call Graph Relation

- “Call graph” expressed as a relation.
  - Five edges:
    - $\text{calls}(A,B)$
    - $\text{calls}(A,C)$
    - $\text{calls}(A,D)$
    - $\text{calls}(B,D)$
    - $\text{calls}(C,D)$

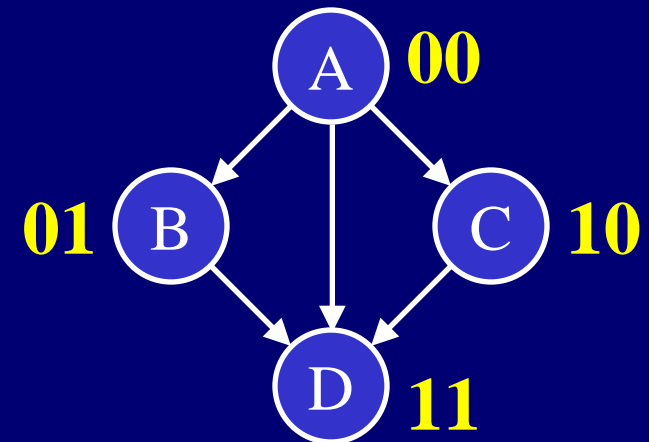


# Call Graph Relation

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $f$ |
|-------|-------|-------|-------|-----|
| 0     | 0     | 0     | 0     | 0   |
| 0     | 0     | 0     | 1     | 1   |
| 0     | 0     | 1     | 0     | 1   |
| 0     | 0     | 1     | 1     | 1   |
| 0     | 1     | 0     | 0     | 0   |
| 0     | 1     | 0     | 1     | 0   |
| 0     | 1     | 1     | 0     | 0   |
| 0     | 1     | 1     | 1     | 1   |
| 1     | 0     | 0     | 0     | 0   |
| 1     | 0     | 0     | 1     | 0   |
| 1     | 0     | 1     | 0     | 0   |
| 1     | 0     | 1     | 1     | 1   |
| 1     | 1     | 0     | 0     | 0   |
| 1     | 1     | 0     | 1     | 0   |
| 1     | 1     | 1     | 0     | 0   |
| 1     | 1     | 1     | 1     | 0   |

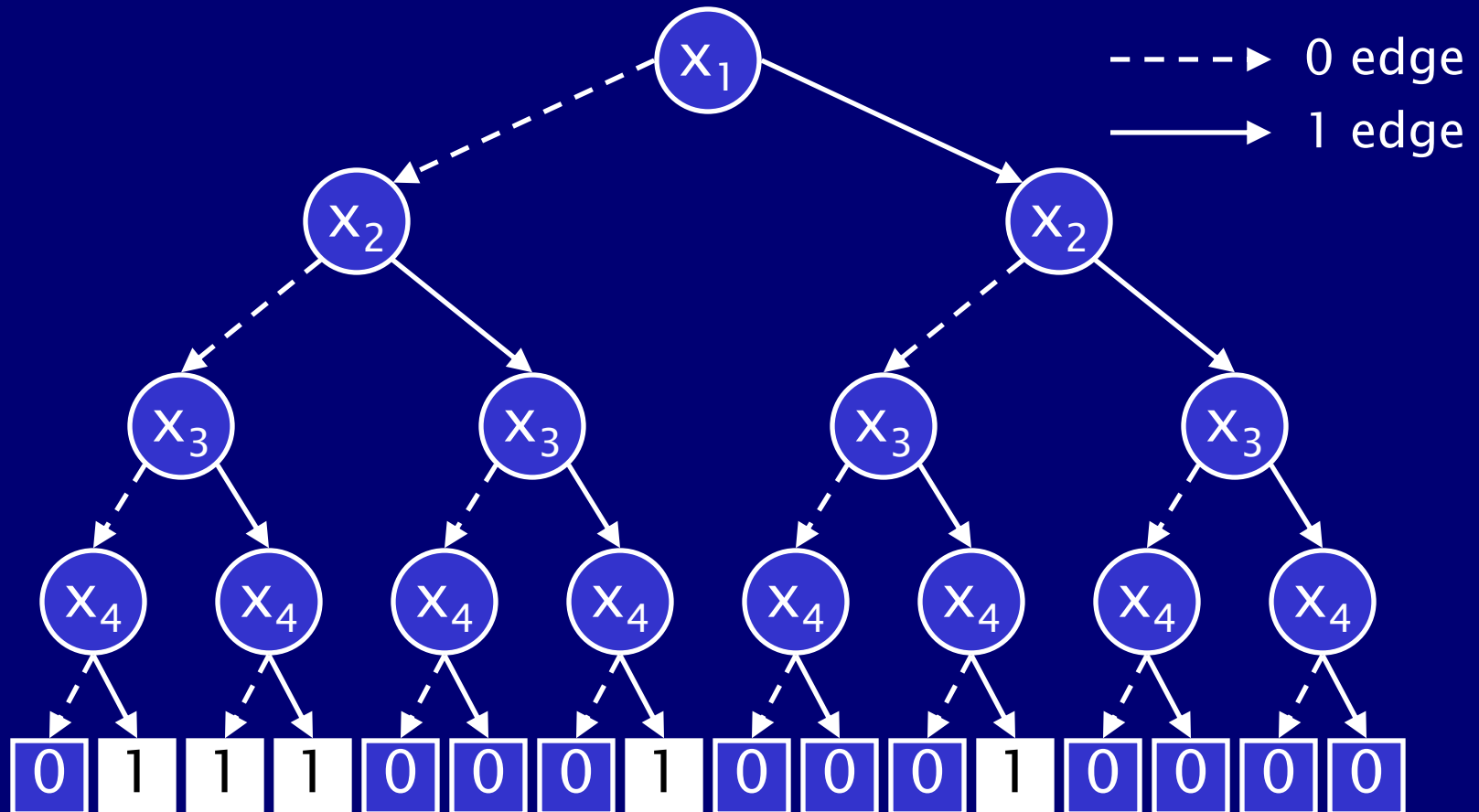
- Relation expressed as a binary function.

- A=00, B=01, C=10, D=11



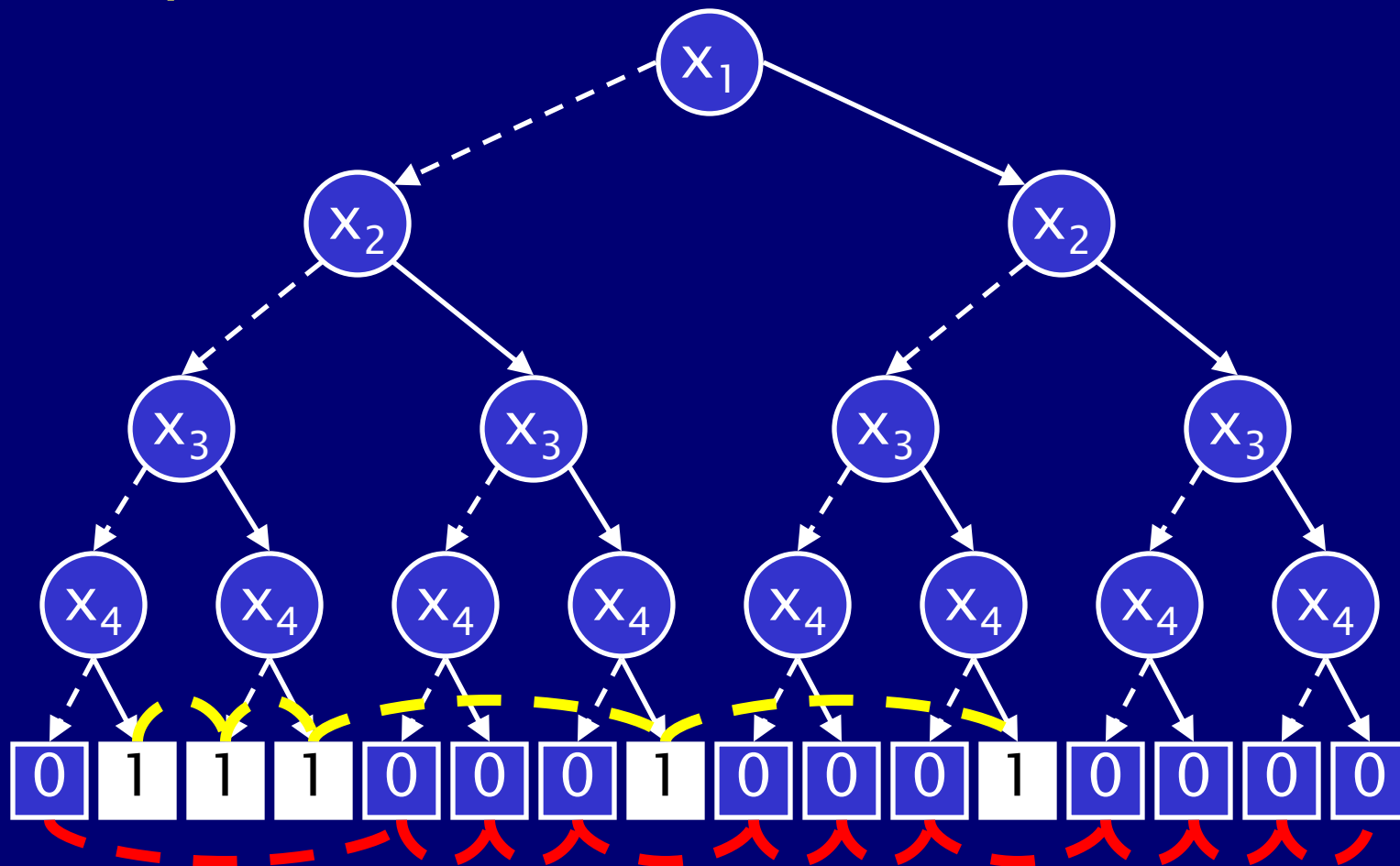
# Binary Decision Diagrams

- Graphical encoding of a truth table.



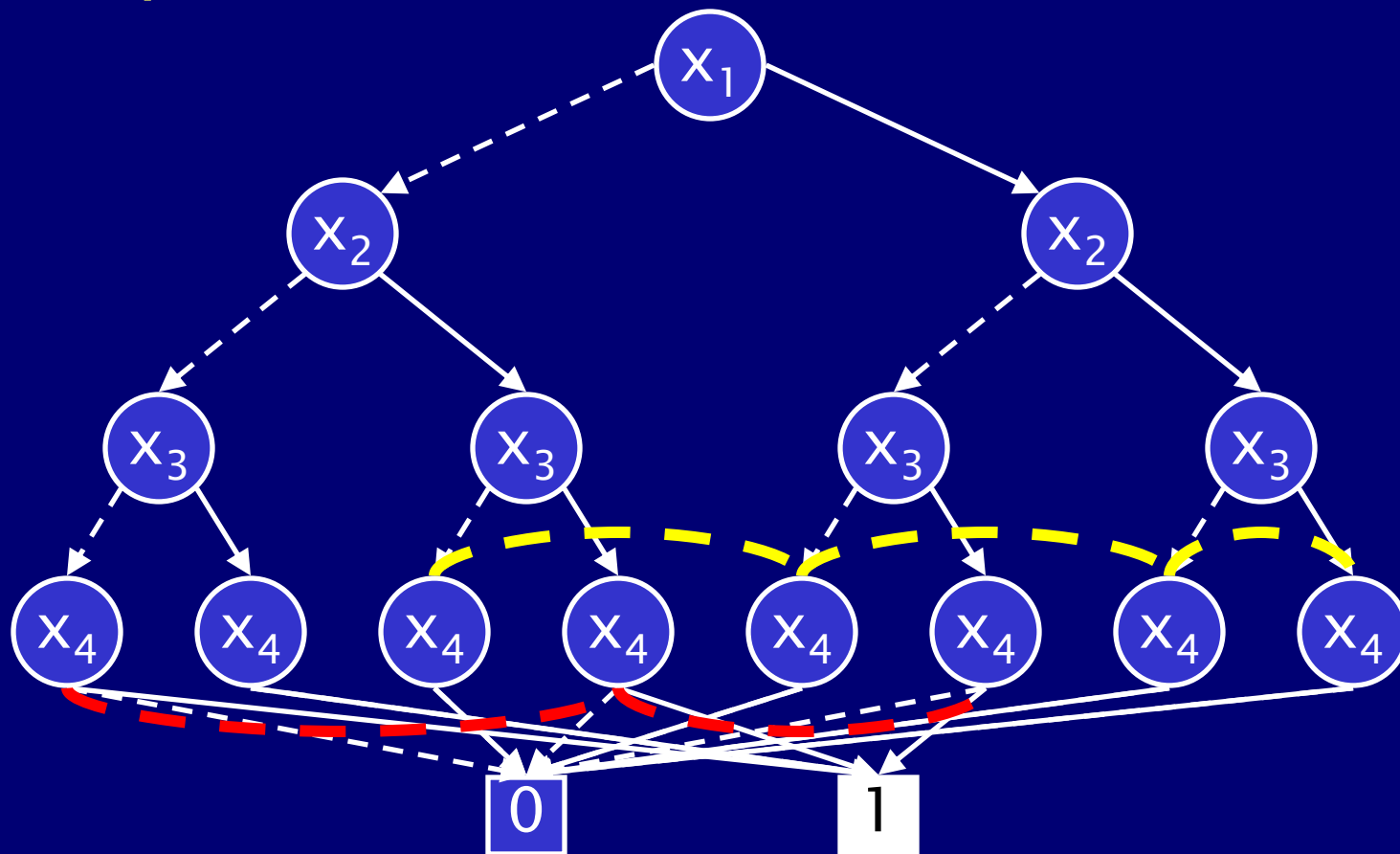
# Binary Decision Diagrams

- Collapse redundant nodes.



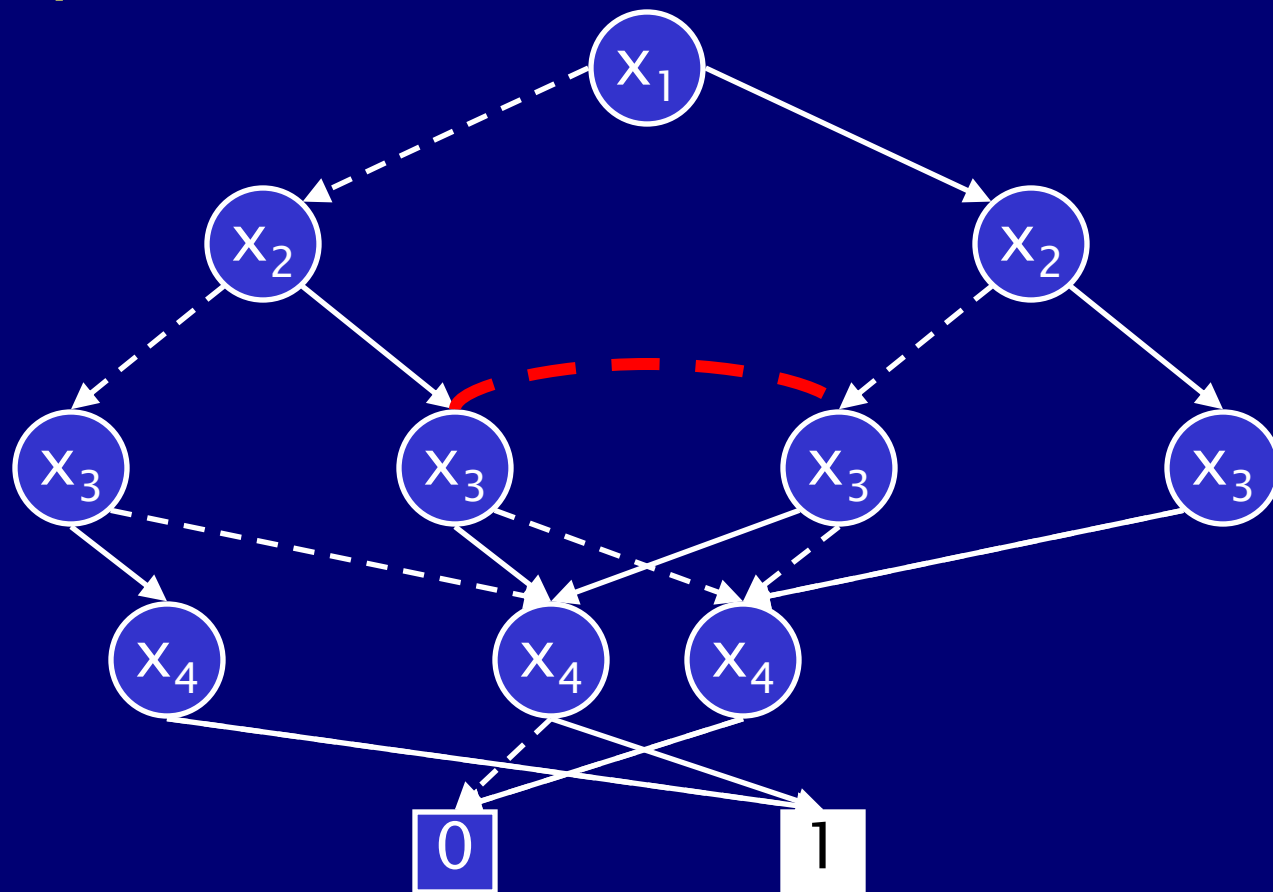
# Binary Decision Diagrams

- Collapse redundant nodes.



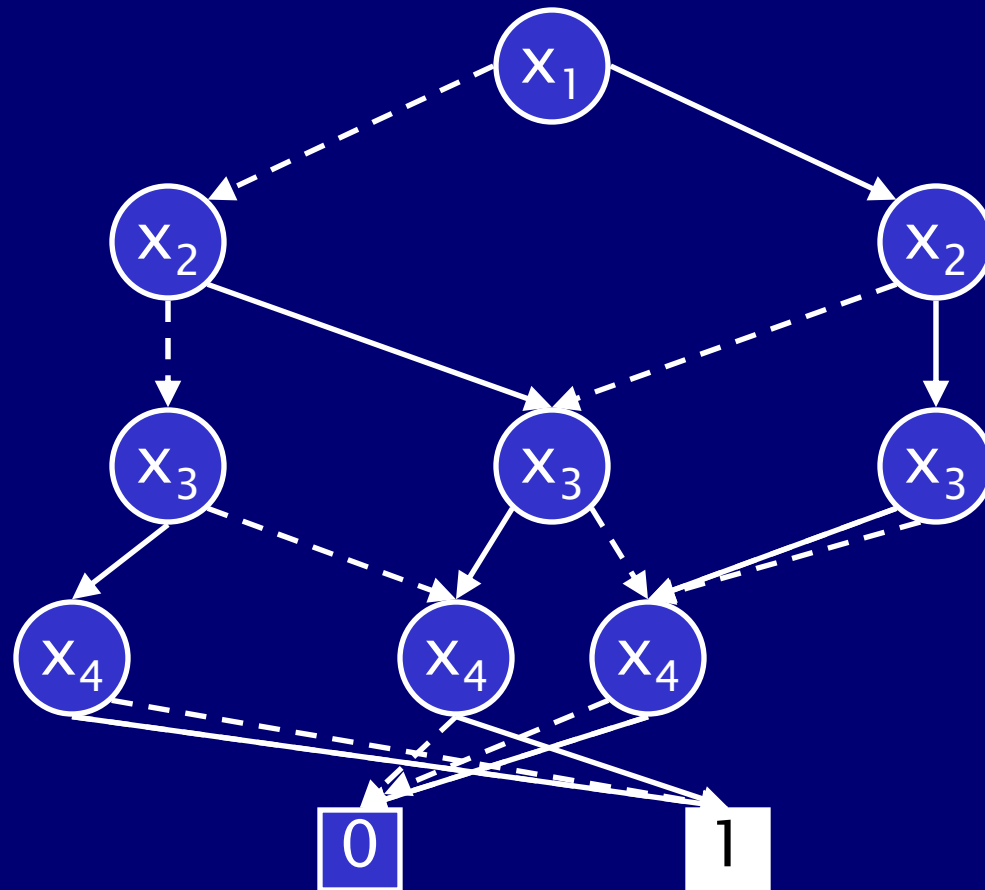
# Binary Decision Diagrams

- Collapse redundant nodes.



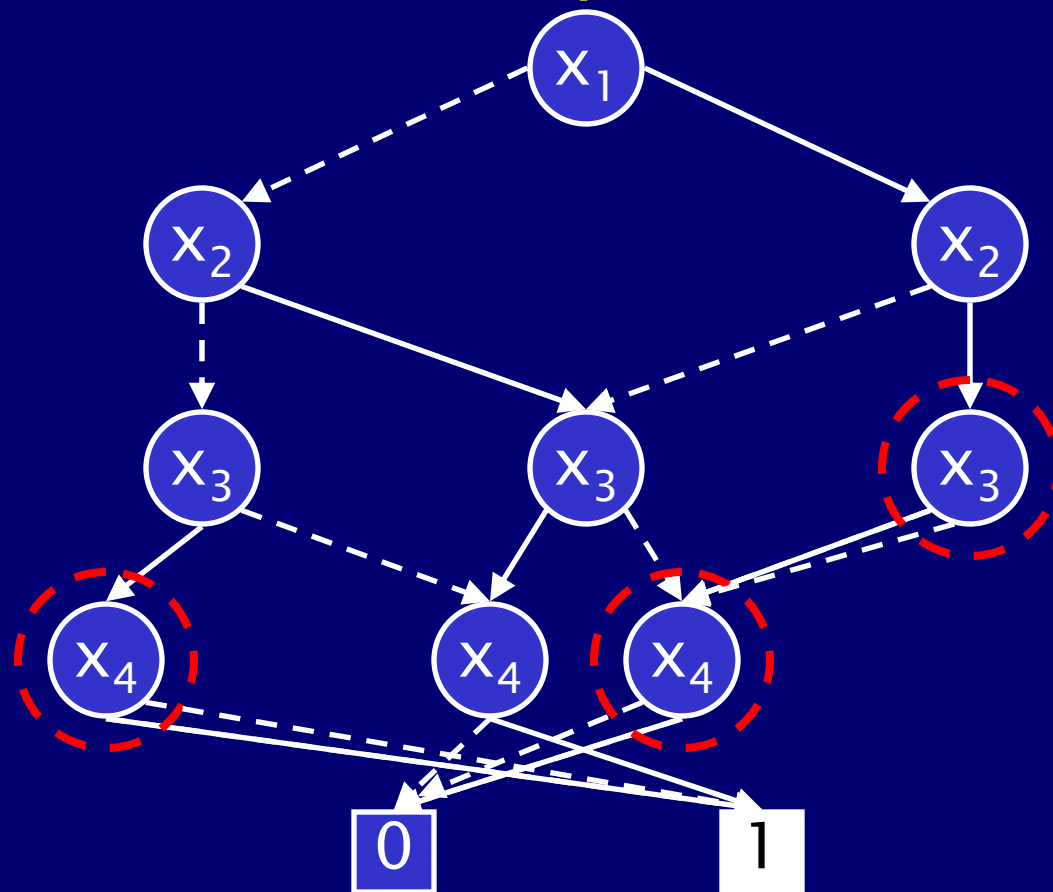
# Binary Decision Diagrams

- Collapse redundant nodes.



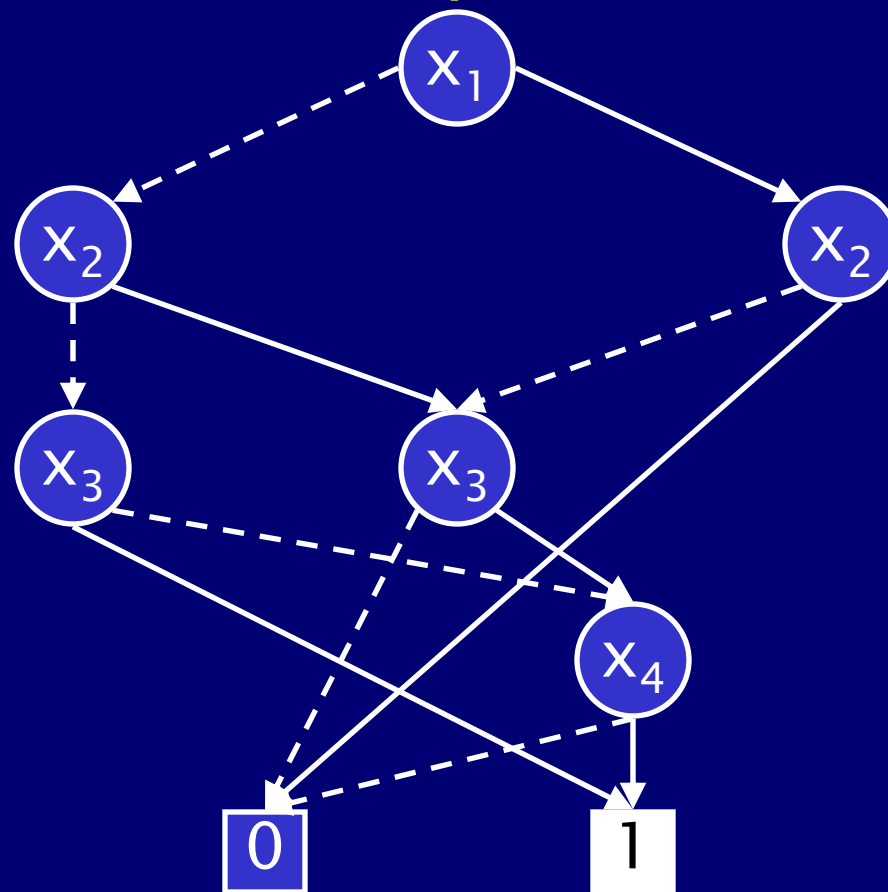
# Binary Decision Diagrams

- Eliminate unnecessary nodes.



# Binary Decision Diagrams

- Eliminate unnecessary nodes.



# Datalog $\rightarrow$ BDDs

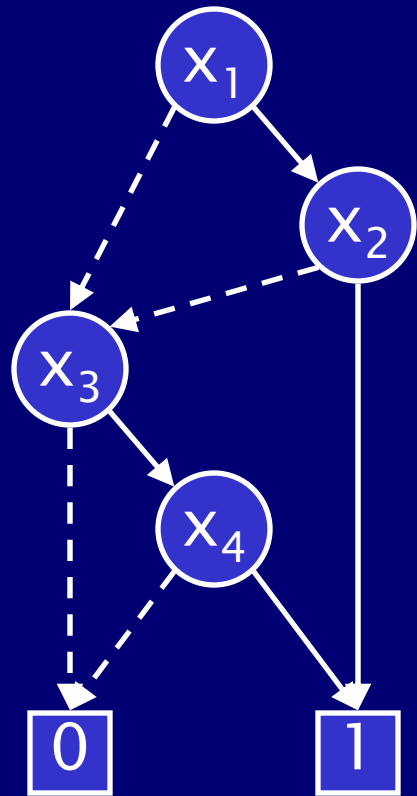
| Datalog   | BDDs   |
|---|--|
| Relations   | Boolean functions                                |
| Relation ops:<br>$\bowtie, \cup, \text{select}, \text{project}$ | Boolean function ops:<br>$\wedge, \vee, -, \sim$ |
| Relation at a time  | Function at a time                               |
| Semi-naïve evaluation   | Incrementalization                               |
| Fixed-point   | Iterate until stable                             |

# Binary Decision Diagrams

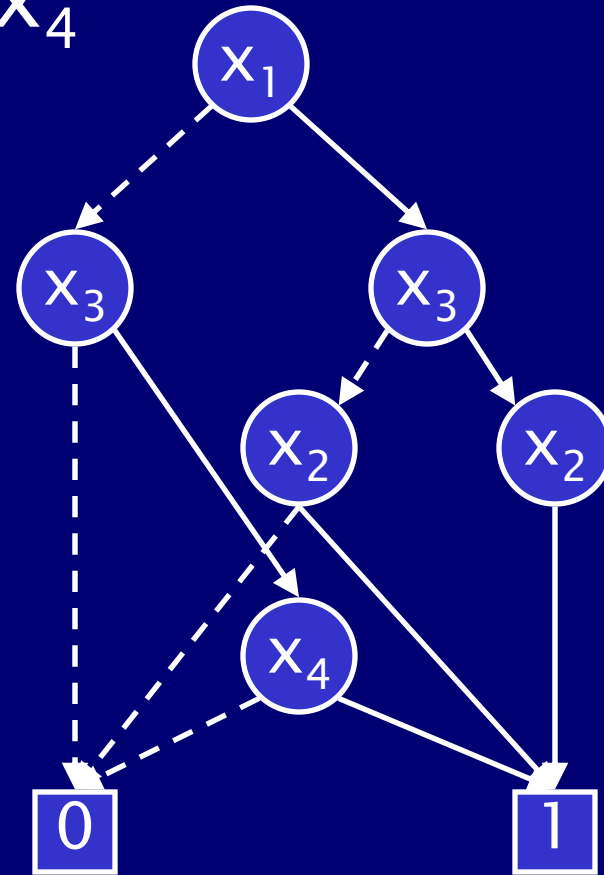
- Represent tiny and huge relations compactly
- Size depends on redundancy
  - Similar contexts have similar numberings
  - Variable ordering in BDDs

# BDD Variable Order is Important!

$$x_1x_2 + x_3x_4$$



$$x_1 < x_2 < x_3 < x_4$$



$$x_1 < x_3 < x_2 < x_4$$

# Variable Numbering: Active Machine Learning

- Must be determined dynamically
- Limit trials with properties of relations
- Each trial may take a long time
- Active learning:  
select trials based on uncertainty
- Several hours
- Comparable to exhaustive for small apps

# Optimizations in bddbddb

- **Algorithmic**
  - Clever context numbering to exploit similarities
- **Query optimizations**
  - Magic-set transformation
  - semi-naïve evaluation
- **Compiler optimizations**
  - Redundancy elimination, liveness analysis
- **BDD optimizations**
  - Active machine learning
- **BDD library extensions and turning**

# Top 4 Techniques in PQL

|          |                                    |
|----------|------------------------------------|
| Compiler | Context-sensitive pointer analysis |
|----------|------------------------------------|

|                 |                               |
|-----------------|-------------------------------|
| HW Verification | BDD: binary decision diagrams |
|-----------------|-------------------------------|

|          |         |
|----------|---------|
| Database | Datalog |
|----------|---------|

|    |                         |
|----|-------------------------|
| AI | Active machine learning |
|----|-------------------------|

# Big Picture

Important problems Security

Easy queries PQL → Datalog → BDD (bddbdb)

Deep analyses Context-sensitive pointers

Accurate answers

# Benchmark

Nine large, widely used applications

- Blogging/bulletin board applications
- Used at a variety of sites
- Open-source Java J2EE apps
- Available from [SourceForge.net](http://SourceForge.net)

# Vulnerabilities Found

|                  | <b>SQL<br/>injection</b> | <b>HTTP<br/>splitting</b> | <b>Cross-site<br/>scripting</b> | <b>Path<br/>traversal</b> | <b>Total</b> |
|------------------|--------------------------|---------------------------|---------------------------------|---------------------------|--------------|
| <b>Header</b>    | <b>0</b>                 | <b>6</b>                  | <b>4</b>                        | <b>0</b>                  | <b>10</b>    |
| <b>Parameter</b> | <b>6</b>                 | <b>5</b>                  | <b>0</b>                        | <b>2</b>                  | <b>13</b>    |
| <b>Cookie</b>    | <b>1</b>                 | <b>0</b>                  | <b>0</b>                        | <b>0</b>                  | <b>1</b>     |
| <b>Non-Web</b>   | <b>2</b>                 | <b>0</b>                  | <b>0</b>                        | <b>3</b>                  | <b>5</b>     |
| <b>Total</b>     | <b>9</b>                 | <b>11</b>                 | <b>4</b>                        | <b>5</b>                  | <b>29</b>    |

# Accuracy

| Benchmark      | Classes | Context insensitive | Context sensitive | False |
|----------------|---------|---------------------|-------------------|-------|
| jboard         | 264     | 0                   | 0                 | 0     |
| blueblog       | 306     | 1                   | 1                 | 0     |
| webgoat        | 349     | 51                  | 6                 | 0     |
| blojsom        | 428     | 48                  | 2                 | 0     |
| personalblog   | 611     | 460                 | 2                 | 0     |
| snipsnap       | 653     | 732                 | 27                | 12    |
| road2hibernate | 867     | 18                  | 1                 | 0     |
| pebble         | 889     | 427                 | 1                 | 0     |
| roller         | 989     | 378                 | 1                 | 0     |
| Total          | 5356    | 2115                | 41                | 12    |

# Related Work

- Program analysis as deductive queries
  - Ullman, Principles of Database and Knowledge-Base Systems, 1989

| <b>Reps, 1994</b>                                     | <b>bddbddb, 2005</b>                        |
|---|---|
| Problem<br>data-flow analysis<br>reaching def/slicing | software security<br>pointer alias analysis |
| Coral   | Custom BDD based                            |
| Demand Driven   | Exhaustive                                  |
| Implementation ease<br>magic set xform                | BDD tuning                                  |
| Slower  | Faster<br>solved open problem               |
| < 1000 lines of code                                  | 800,000 byte codes                          |

# References

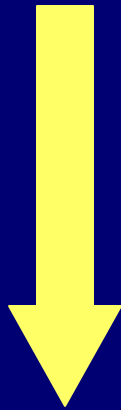
Pointers: Whaley, Lam, PLDI 04

C pointers: Avots, Dalton, Livshits, Lam, ICSE 05

PQL: Martin, Livshits, Lam, OOPSLA 05

Java Security: Livshits, Lam, Usenix security 05

# Easy Context-Sensitive Analysis



Sophisticated  
Context-sensitive  
Analysis

PQL

Datalog

**bddbddb**

(**BDD**-based  
**d**eductive **d**atabase)

with

Active Machine Learning

BDD code