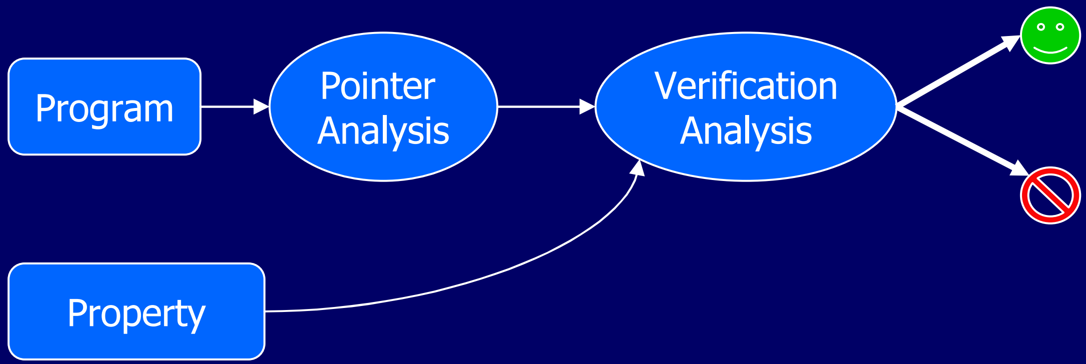
A Common Approach to Pointer Analysis in Software Verification



- Break verification problem into two phases
  - Preprocessing phase – separate points-to analysis
    - typically no distinction between objects allocated at same site
  - Verification phase – verification using points-to results
- May lose precision
  - May lose ability to perform strong updates
  - May produce false alarms

Loss of Precision in Two-Phase Approach

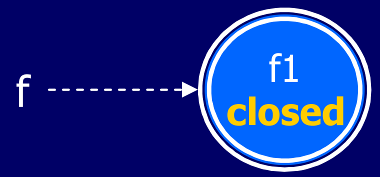
```
f = new InputStream();
f.read();
f.close();
```

Verify f is not read after it is closed

Straightforward ...

Loss of Precision in Two-Phase Approach

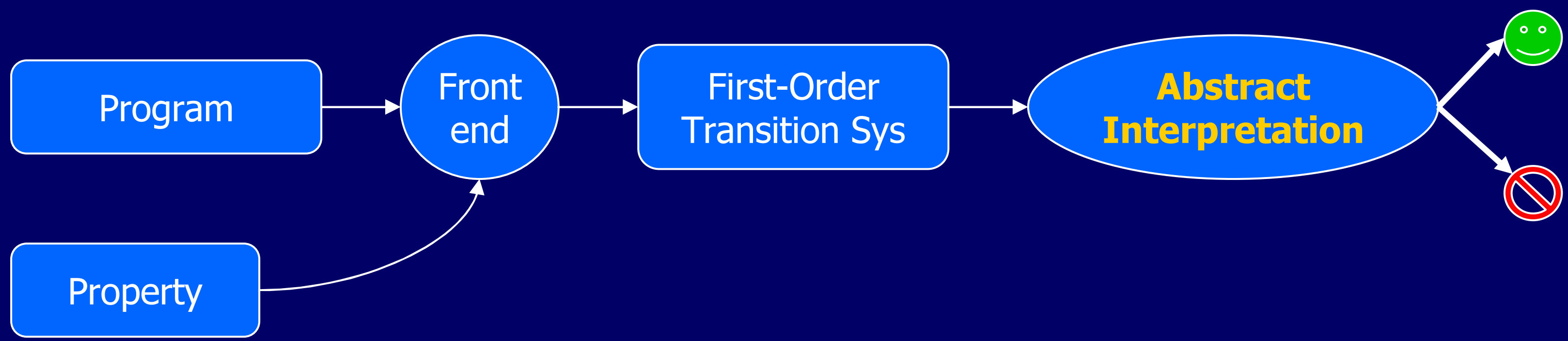
```
while (?) {
  f = new InputStream();
  f.read();
  f.close();
}
```



False alarm!
"read may be **erroneous**"

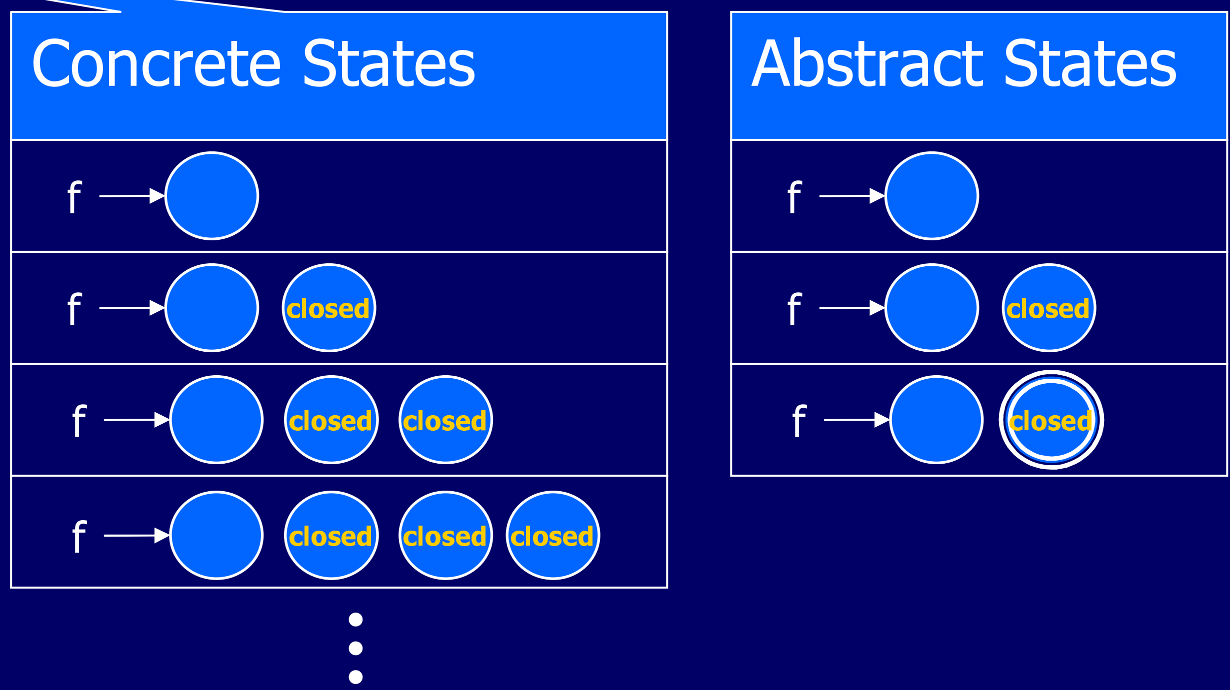
The TVLA Approach

- TVLA is a flexible (parametric) system for abstract interpretation and verification
  1. parametric heap (i.e., pointer) analysis
     - user can specify criterion for merging heap-allocated objects
     - user can specify criterion for merging shape graphs
  2. verification integrated with heap analysis
     - heap analysis (merging criterion) can adapt to verification



The TVLA Approach: An Example

```
while (?) {
  f = new InputStream();
  f.read();
  f.close();
}
```



TVLA: 3-Valued Logic Analyzer
“A Yacc For Static Analysis”

Tal Lev-Ami and Roman Manevich
http://www.cs.tau.ac.il/~tvla

Alexey Loginov

G. Ramalingam

Eran Yahav

Outline

- Simple examples
  - Cleanness of linked lists
  - Recursive procedures
  - Correctness of sorting
  - MarkAndSweep
- Projects
  - Compile time garbage collection
  - CANVAS
- Ongoing Work

Cleanness Analysis of Linked Lists (Nurit Dor, SAS 2000)

- Static analysis of C programs manipulating linked lists
- Defects checked
  - References to freed memory
  - Null dereferences
  - Memory leaks
- Existing algorithms are inadequate
  - Miss errors
  - Report too many false alarms

Null Dereferences

```
typedef struct element
{
   int value;
   struct element *n;
} Element
```

```
bool search(  int value,
              Element *x)
{
Element * c = x
   while (  x != NULL )
   {
      if (c→ val  == value)
            return TRUE;
      c = c → n;
   }
   return FALSE; }
```

Demo

TVLA inputs

- TVP - **T**hree **V**alued **P**rogram
  - Predicate declaration
  - Action definitions SOS
  - Control flow graph
- TVS - **T**hree **V**alued **S**tructure

Program independent

Demo

Challenge 1

- Write a C procedure on which TVLA reports false null dereference

Proving Correctness of Sorting Implementations (Lev-Ami, Reps, S, Wilhelm ISSTA 2000)

- Partial correctness
  - The elements are sorted
  - The list is a permutation of the original list
- Termination
  - At every loop iterations the set of elements reachable from the head is decreased

Example: InsertSort

```
typedef struct list_cell {
    int data;
    struct list_cell *n;
} *List;
```

pred.tvp

actions.tvp

Run Demo

```
List InsertSort(List x) {
List r, pr, rn, l, pl; r = x; pr = NULL;
  while (r != NULL) {
    l = x; rn = r → n; pl = NULL;
    while (l != r) {
      if (l → data > r → data) {
        pr → n = rn; r → n = l;
        if (pl == NULL) x = r;
        else pl → n = r;
        r = pr;
        break;
      }
      pl = l; l = l → n;
    }
    pr = r; r = rn;
  }
  return x;
}
```

Example: InsertSort

```
typedef struct list_cell {
    int data;
    struct list_cell *n;
} *List;
```

```
List InsertSort(List x) {
 if (x == NULL)    return NULL
 pr = x;  r = x->n;
 while (r != NULL) {
         pl = x; rn = r->n; l = x->n;
         while (l != r) {
                 pr->n = rn ;
                 r->n = l;
                 pl->n = r;
                 r = pr;
                 break;
                     }
                pl = l;
                l = l->n;
                }
         pr = r;
         r = rn;
  }
```

Run Demo

14

Example: Reverse

```
typedef struct list_cell {
    int data;
    struct list_cell *n;
} *List;
```

```
List reverse (List x) {
    List y, t;
    y = NULL;
    while (x != NULL) {
        t = y;
        y = x;
        x = x → next;
        y → next = t;
    }
    return y;
}
```

Run Demo

Challenge 2

- Write a sorting C procedure on which TVLA fails to prove sortedness or permutation

Interprocedural Analysis (Noam Rinetzky)

- Model the stack as a linked list (CC 2001)
  - Observe alias patterns
  - Handles recursion with pointers from the stack to the heap (but rather slow)
- Exploit referential transparency
  - The part of the store modified by a procedure is limited
  - Summarize irrelevant calling contexts
  - Pre-analyze Abstract Data Types
    - Analyzed parts of LEDA linked lists

Example: Mark and Sweep

```
void Mark(Node root) {
  if (root != NULL) {
    pending = ∅
    pending = pending ∪ {root}
    marked = ∅
    while (pending ≠ ∅) {
      x = SelectAndRemove(pending)
      marked = marked ∪ {x}
      t = x → left
      if (t ≠ NULL)
        if (t ∉ marked)
          pending = pending ∪ {t}
      t = x → right
      if (t ≠ NULL)
        if (t ∉ marked)
          pending = pending ∪ {t}
    }
  }
  assert(marked == Reachset(root))
```

```
void Sweep() {
  unexplored = Universe
  collected = ∅
  while (unexplored ≠ ∅) {
    x = SelectAndRemove(unexplored)
    if (x ∉ marked)
      collected = collected ∪ {x}
  }
  assert(collected ==
           Universe – Reachset(root)
        )
```

pred.tvp

Run Demo

Challenge 3

- Use TVLA to show termination of markAndSweep

Mobile Ambients [Nielson' ESOP'00]

- Algorithm for analyzing safety properties of mobile ambients
- Example properties in a routing protocol:
  - uniqueness of packet
  - mutual exclusion
- Code the tree and the program logical structures
- Code the operational semantics using first order logic
- Let TVLA do the rest

Establishing Local Temporal Heap Safety Properties with Applications to Compile-Time Memory Management [Ran Shaham SAS'03, SCP]

Memory deallocation in a timely manner is a hard problem

- Undecidable
- Premature deallocation → Program errors
- Late deallocation → Memory leaks
  Inefficient use of memory

(Old) Idea Compile-Time GC

- The compiler can issue free when objects are no longer needed
- Zero cost
- No more memory leaks
- Difficult for imperative heap-manipulating programs
  - No static names for locations
  - Destructive updates (mutations) x.field=null

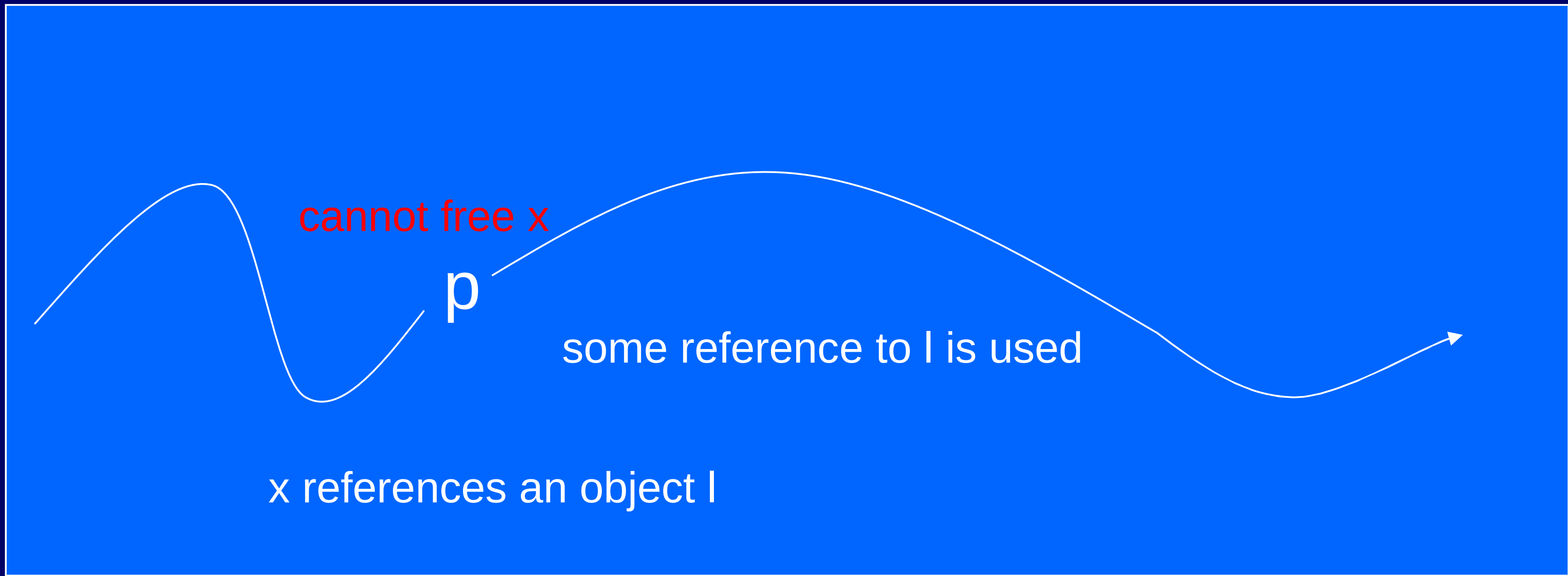
Results

- A framework for developing static algorithms for memory management
  - Free analysis
  - Assign-null (combined with GC)
- “reference” static algorithms
  - Compile-time GC which handles destructive heap updates

Free Analysis

- Free unneeded objects
- Insert “free x” after program point p

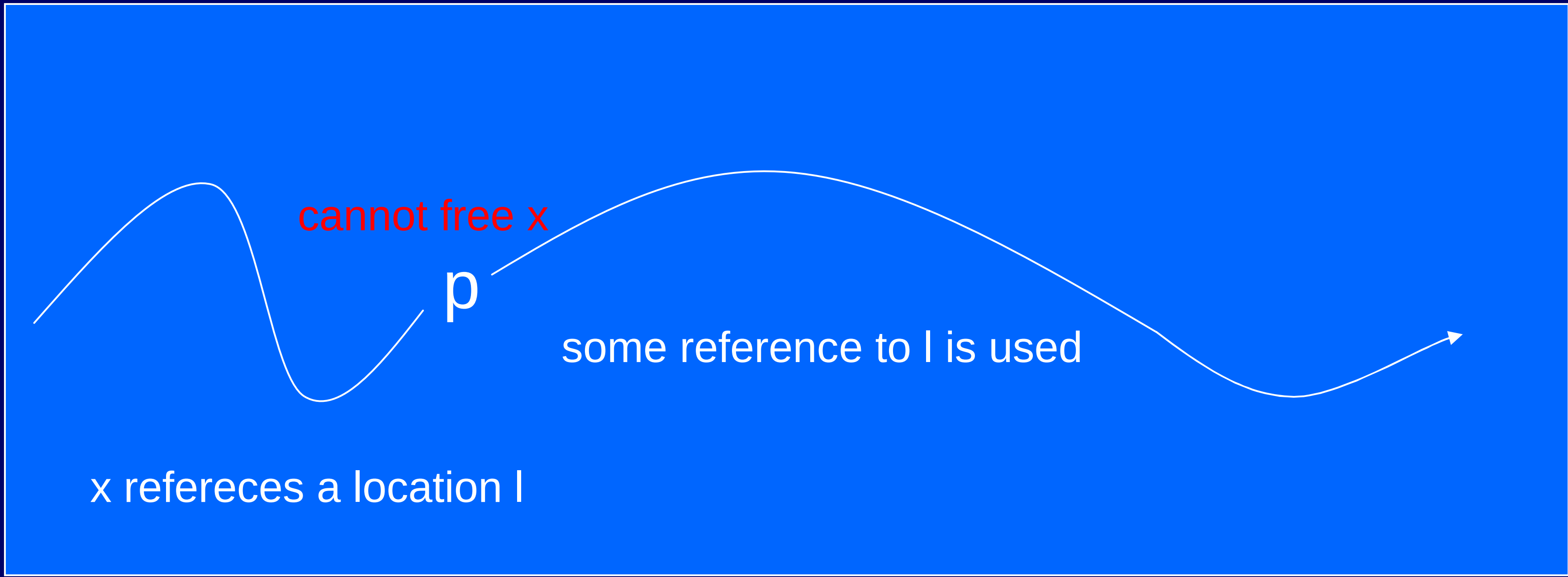
When can free x be inserted after p?

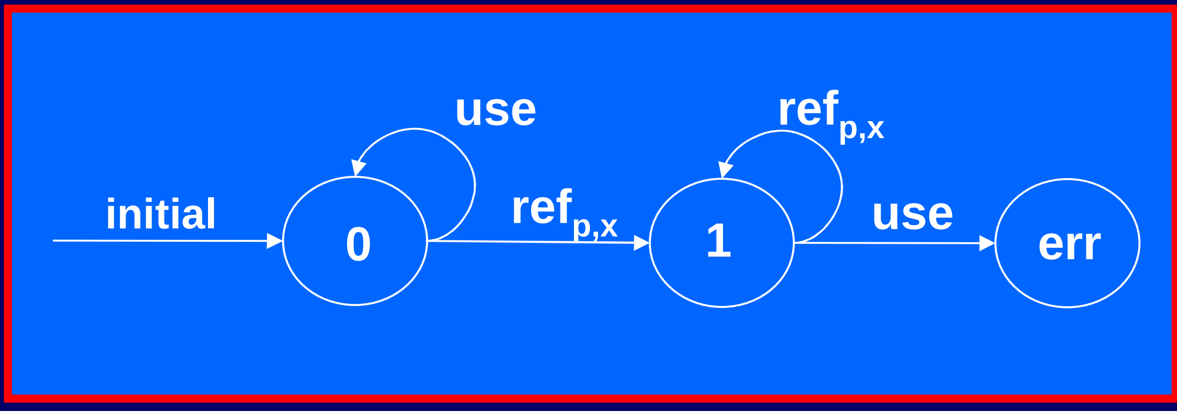


On all execution paths after p there are no uses of references to the object referenced by x ➔

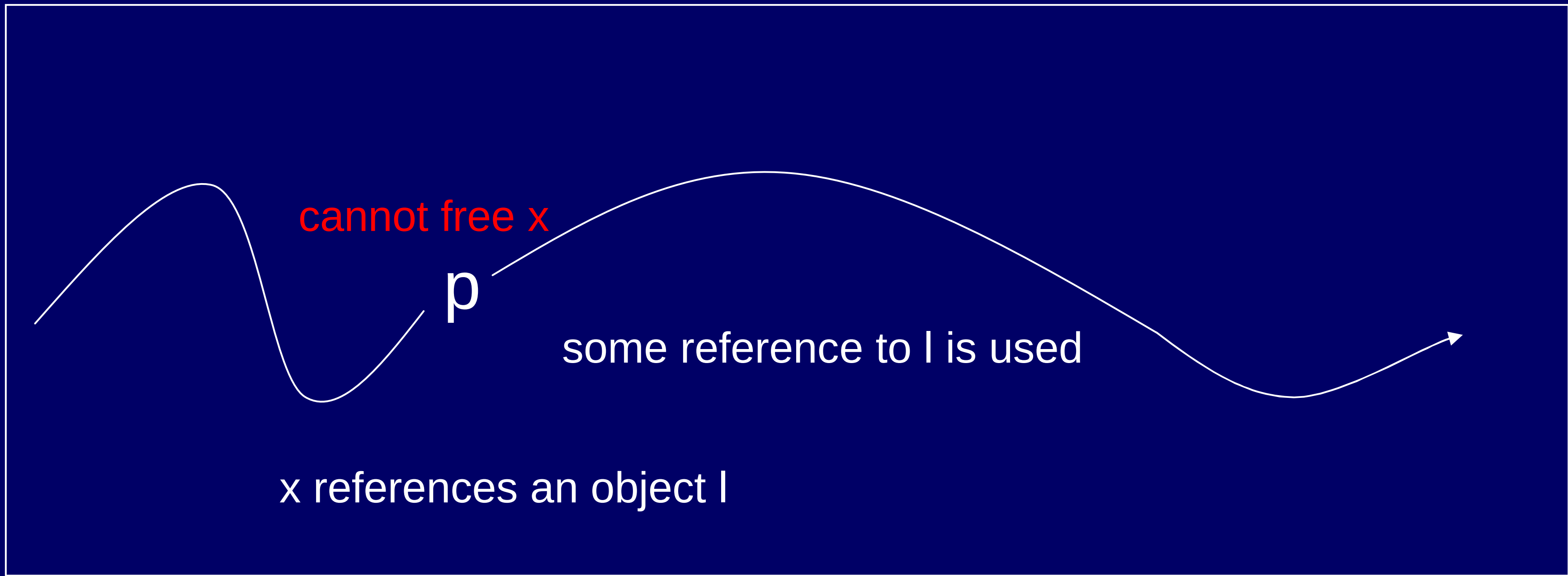
inserting free x after p is valid

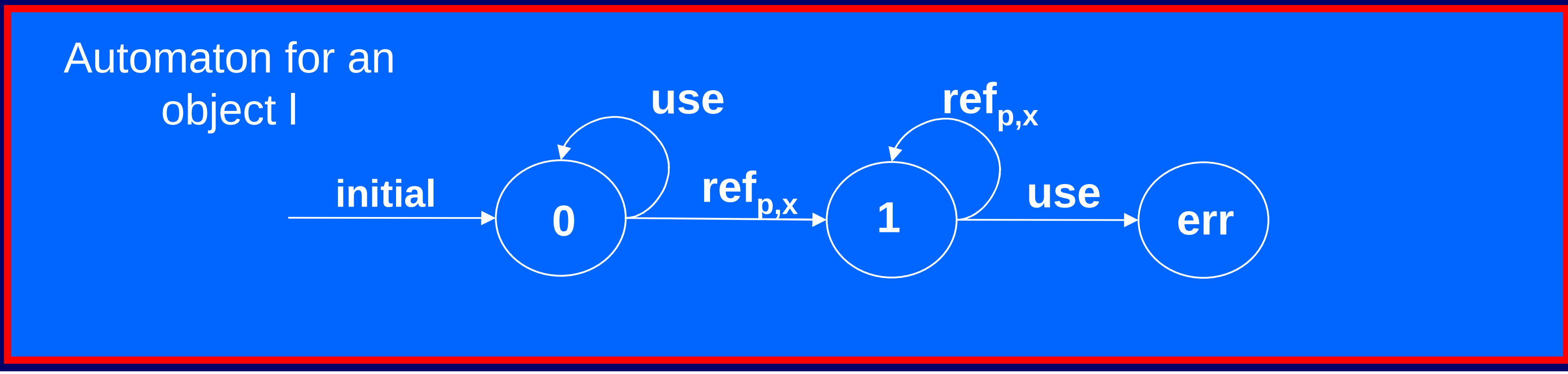
When does inserting free x after p is not valid?





free x after p automaton





A Concrete Semantics for Deallocating Space

- Program state
  - "Usual heap information"
    - Variable values, Field Values
  - Free x after p automaton state
    - For every object l
- Program statement effect
  - "Usual semantics"
  - Trigger automaton events

Prototype Implementation

- Analysis of Java/JavaCard programs
- Generic Java Bytecode Frontend
- Precise analysis

- Scalability issues due to procedure calls
- Model library code
- Number of automata

More Scalable Solution [Gilad Arnold]

- Combine backward and forward analysis
  - Forward analysis determine the shape
  - Backward analysis locates heap liveness
  - Use $\sqcap$

Verification of Safety Properties

The *Canvas* Project (IBM Watson and Tel Aviv)
(***C***omponent ***A***nnotation, ***V***erification ***a***nd ***S***tuff)

Component
a library with cleanly encapsulated state

Client
a program that uses the library

Lightweight Specification
- "correct usage" rules a client must follow
- "call open() before read()"

Certification
does the client program satisfy the lightweight specification?

Verifying Safety Properties using Separation and Heterogeneous Abstractions PLDI'04

E. Yahav

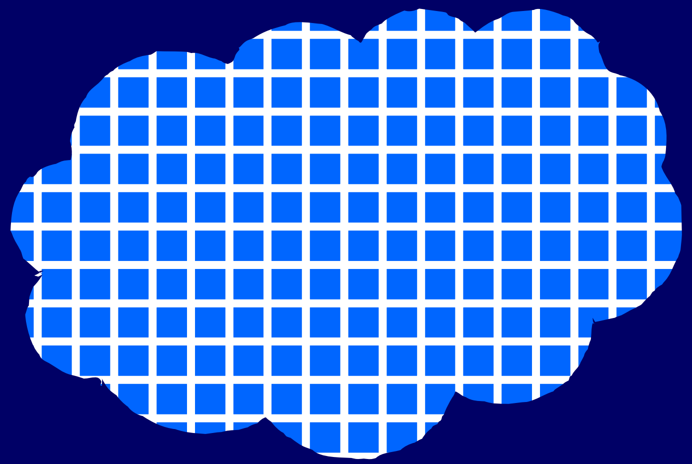
School of Computer Science

Tel-Aviv University

G. Ramalingam

IBM T.J. Watson Research Center

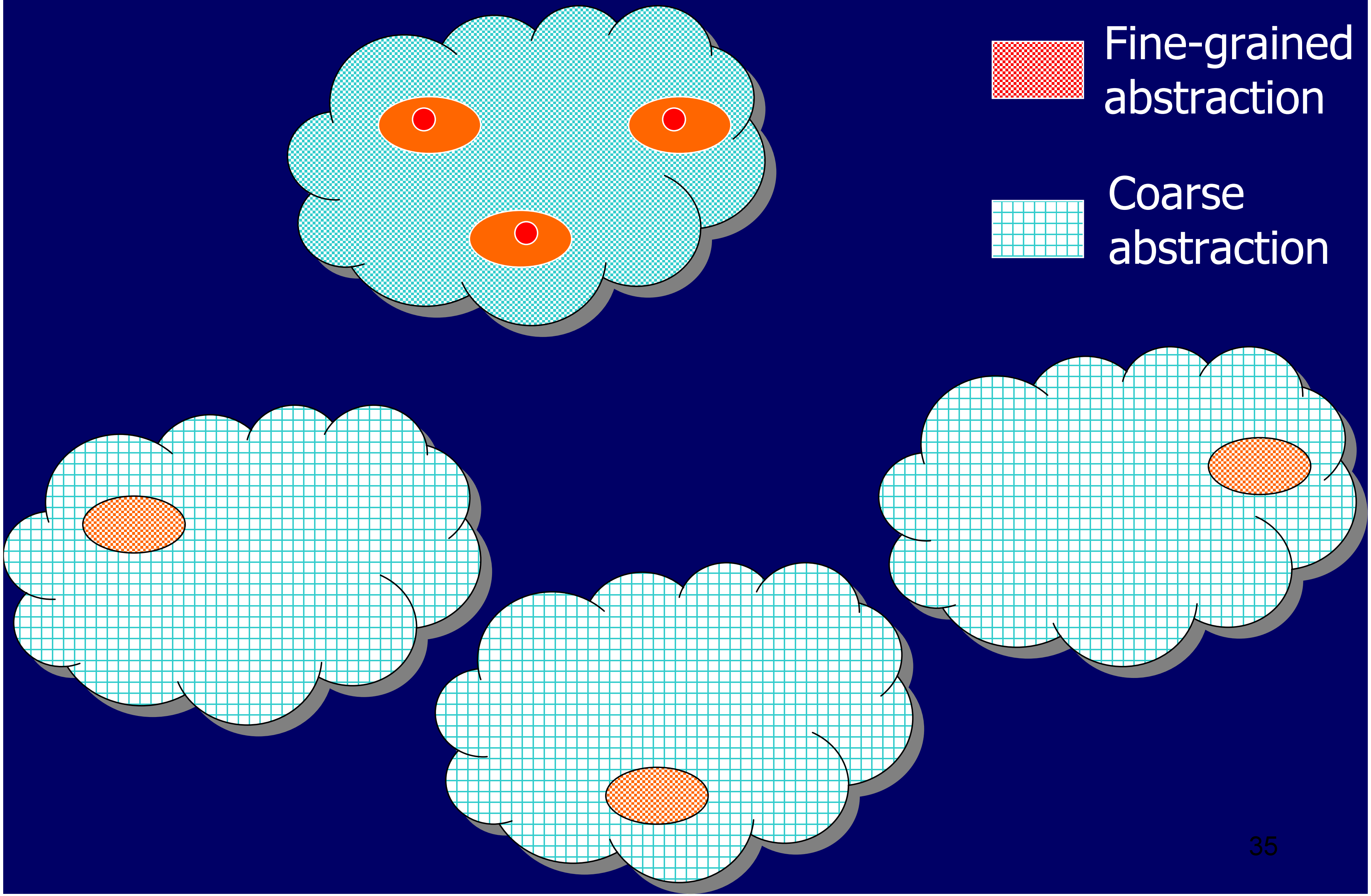
Quick Overview:
Fine Grained Heap Abstraction



Precise ...

... but often too expensive

Separation & Heterogeneous Abstraction



Outline of Seperation

❶ Decompose verification problem into a set of subproblems

❷ Adapt abstraction to each subproblem

Outline of Seperation

❶ Decompose verification problem into a set of subproblems

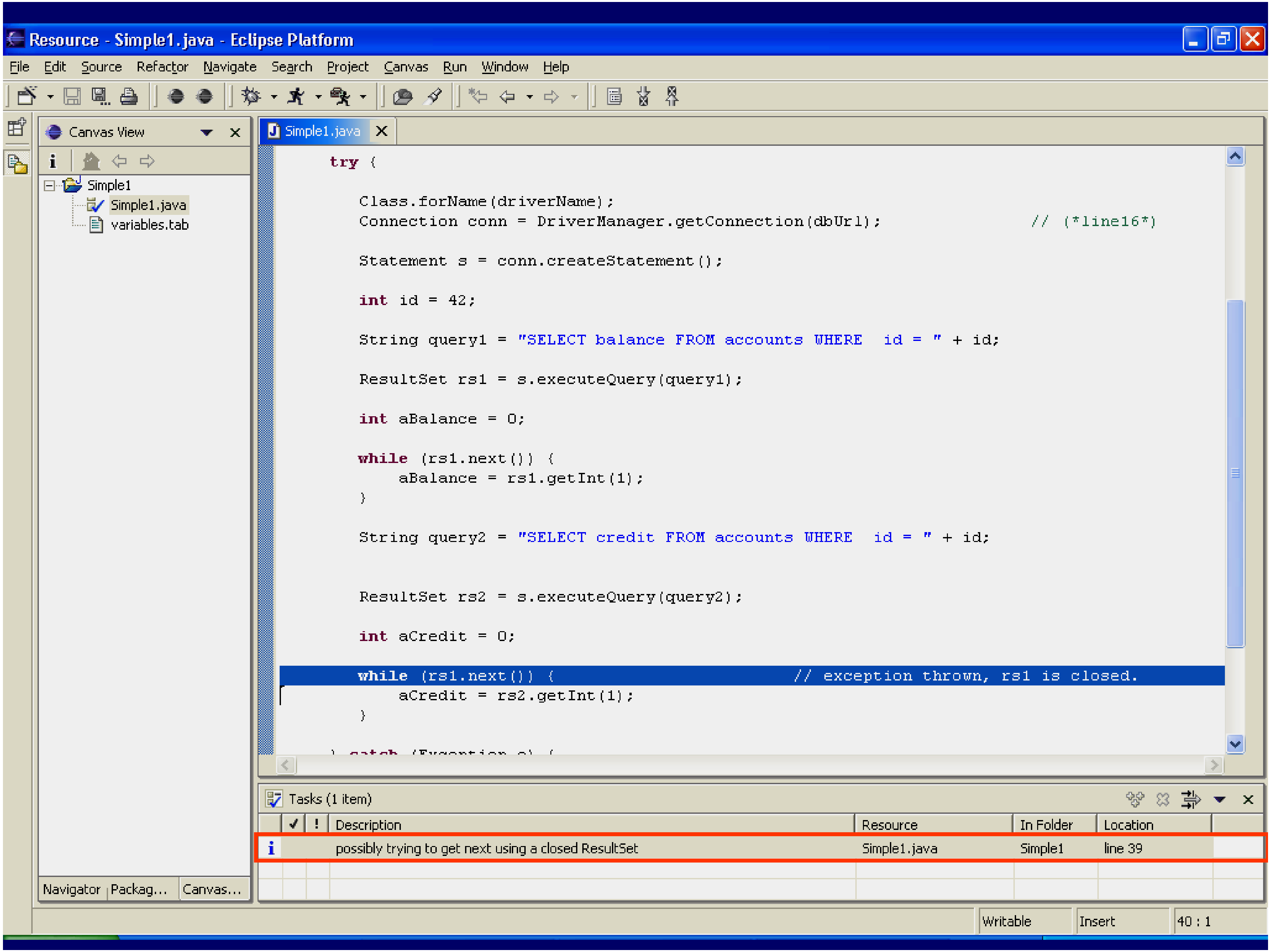
- Analysis-user specifies a separation strategy

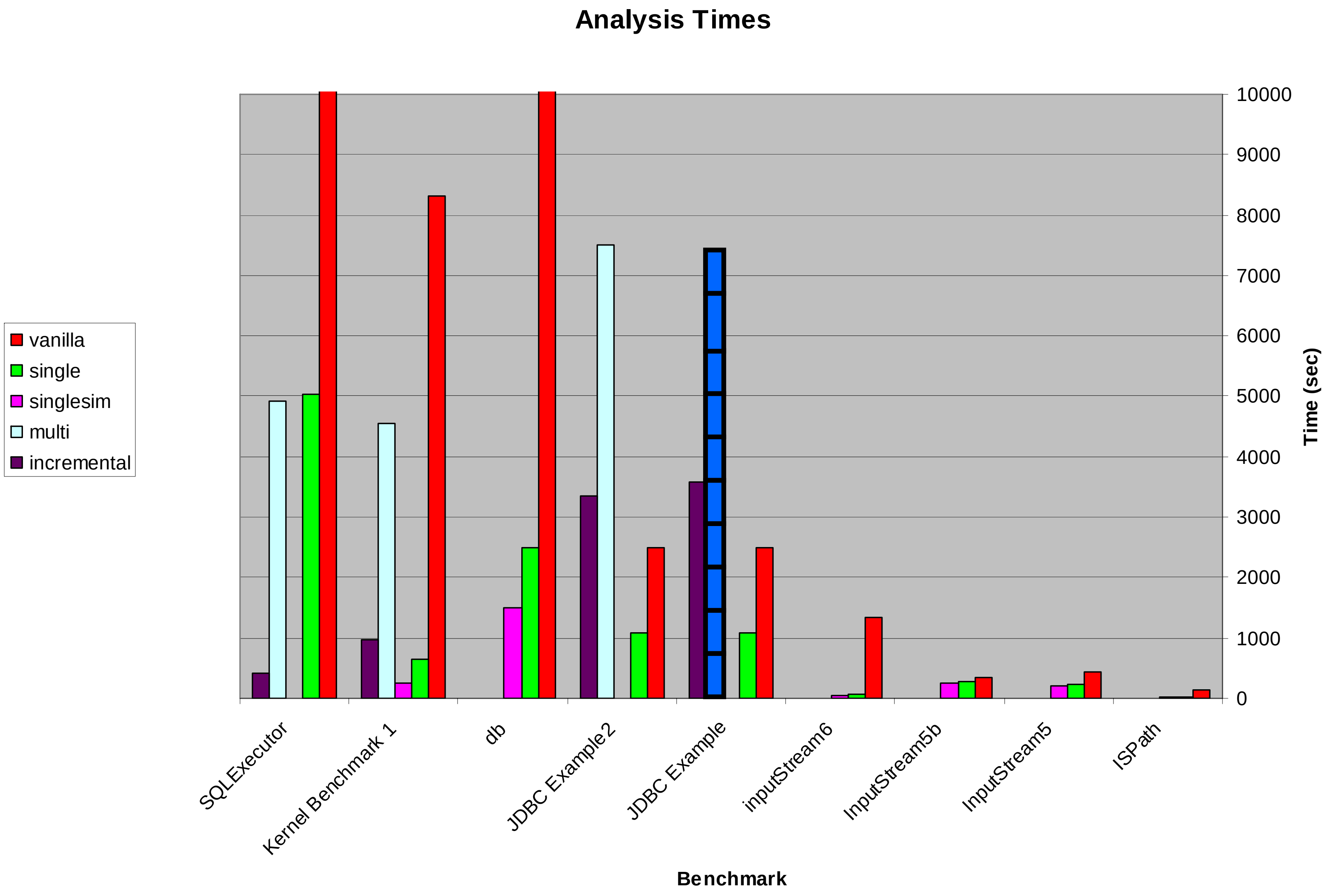
❷ Adapt abstraction to each subproblem

- Heterogeneous abstraction

Prototype Implementation

- Implemented over TVLA
- Correctness comes from the embedding theorem
- Applied to several example programs
  - Up to 5000 lines of Java
- Used to verify
  - Absence of concurrent modification exception (CME)
  - JDBC API conformance
  - IOStreams API conformance
- Improved performance
- In some cases improved precision


Resource - Simple1.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Canvas Run Window Help
Canvas View
Simple1
Simple1.java
variables.tab
Simple1.java
try {
Class.forName(driverName);
Connection conn = DriverManager.getConnection(dbUrl); // (*line16*)
Statement s = conn.createStatement();
int id = 42;
String query1 = "SELECT balance FROM accounts WHERE id = " + id;
ResultSet rs1 = s.executeQuery(query1);
int aBalance = 0;
while (rs1.next()) {
aBalance = rs1.getInt(1);
}
String query2 = "SELECT credit FROM accounts WHERE id = " + id;
ResultSet rs2 = s.executeQuery(query2);
int aCredit = 0;
while (rs1.next()) { // exception thrown, rs1 is closed.
aCredit = rs2.getInt(1);
}
Tasks (1 item)
Description
Resource
In Folder
Location
possibly trying to get next using a closed ResultSet
Simple1.java
Simple1
line 39
Navigator
Packag...
Canvas...
Writable
Insert
40 : 1


Analysis Times
vanilla
single
singlesim
multi
incremental
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Time (sec)
SQLExecutor
Kernel Benchmark 1
db
JDBC Example2
JDBC Example
inputStream6
InputStream5b
InputStream5
ISPath
Benchmark

Ongoing Work

- Assume guarantee reasoning [Yorsh]
- Correctness of collection implementation [Livshits]
- Refinement [Loginov]
- Sclalablity [Manevich, Lev-Ami]
- Heap modularity [Bauer & Rinetzky]

TVLA Design Mistakes

- The operational semantics is written in too low level language
- TVP can be a high level language
- "instrumentation" = "derived"
- "Consistency Rules" = "Integrity Rules"
- Focus = Partial Concretization
- TVLA$\Rightarrow$3VLA

Conclusion

- FO^{TC} is expressive for defining semantics
- TVLA is a very useful research tool
  - Try before you publish
  - Easy to try new algorithms
- Sometimes faster than existing shape analyzers
- But has high interpretation overhead
  - Currently improved

The End

http://www.cs.tau.ac.il/~tvla