Somebody Put a Database in My Software Testing Problem

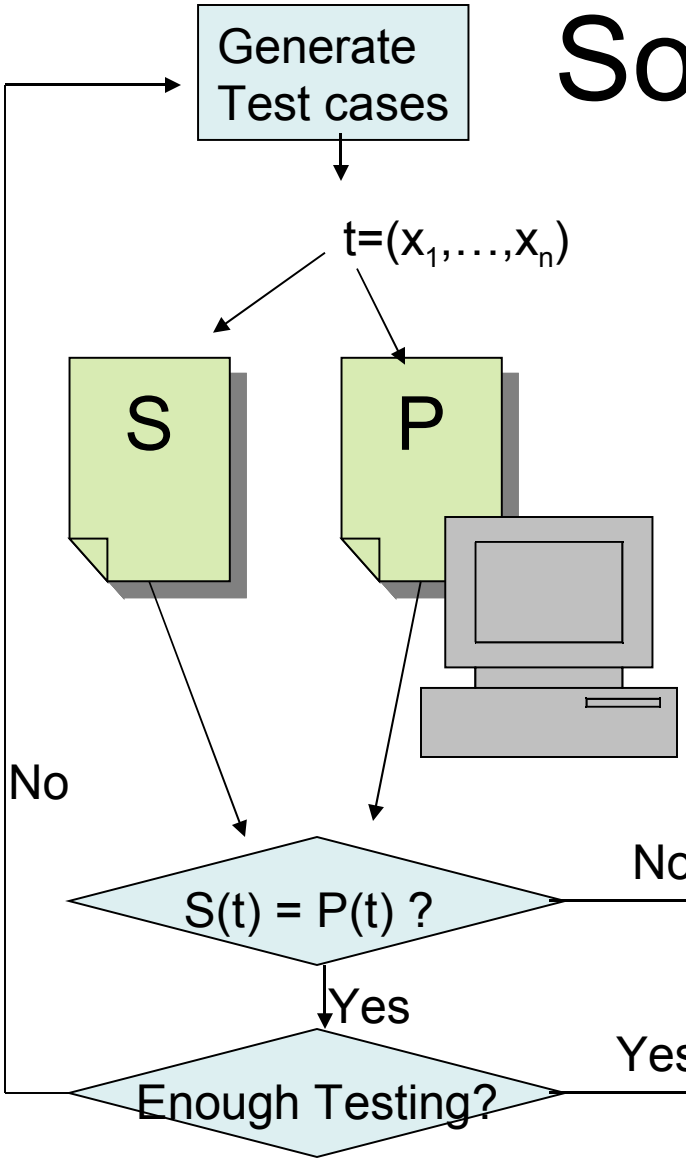
Phyllis Frankl

Polytechnic Institute of NYU

DBTest Keynote, June 7, 2010

NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Software Testing 101



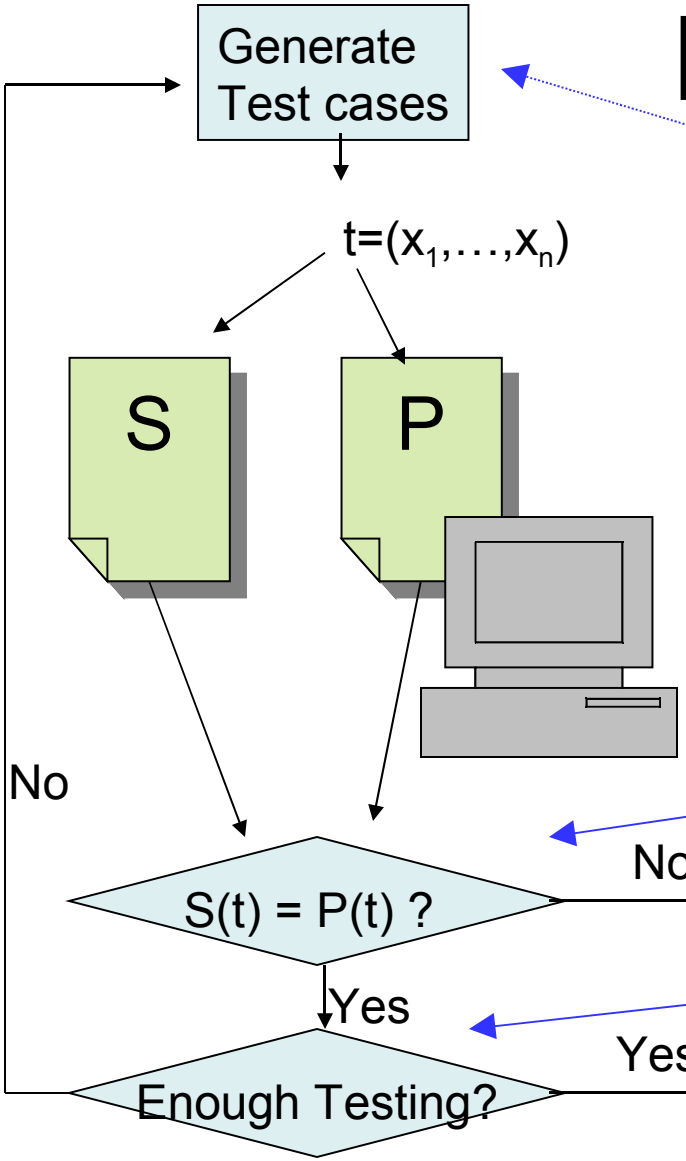
Given application program P intended to satisfy Specification S, we'd like to know whether P conforms to S.

Test P: generate inputs
execute P
check results

(and pray)

NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Research Issues



Test generation techniques

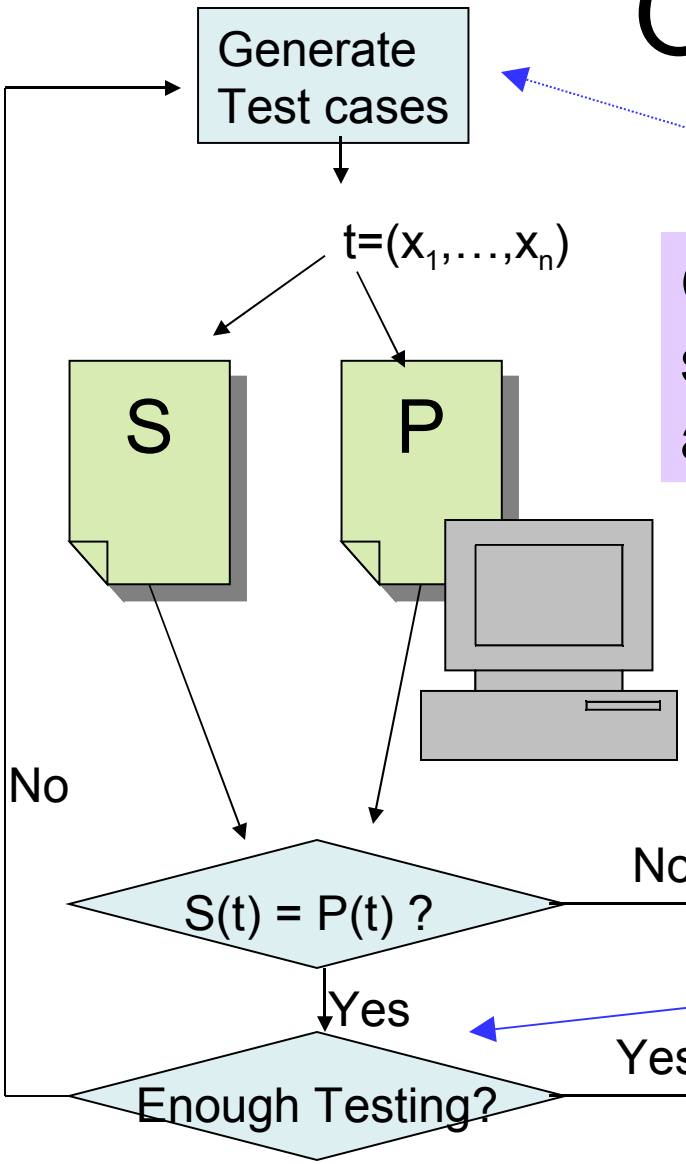
Many others:
- test management
- regression testing
- test prioritization
- dealing with more general programming paradigms: OO, concurrency, …
- …

Test oracles

Test data adequacy criteria

NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Overarching Goal



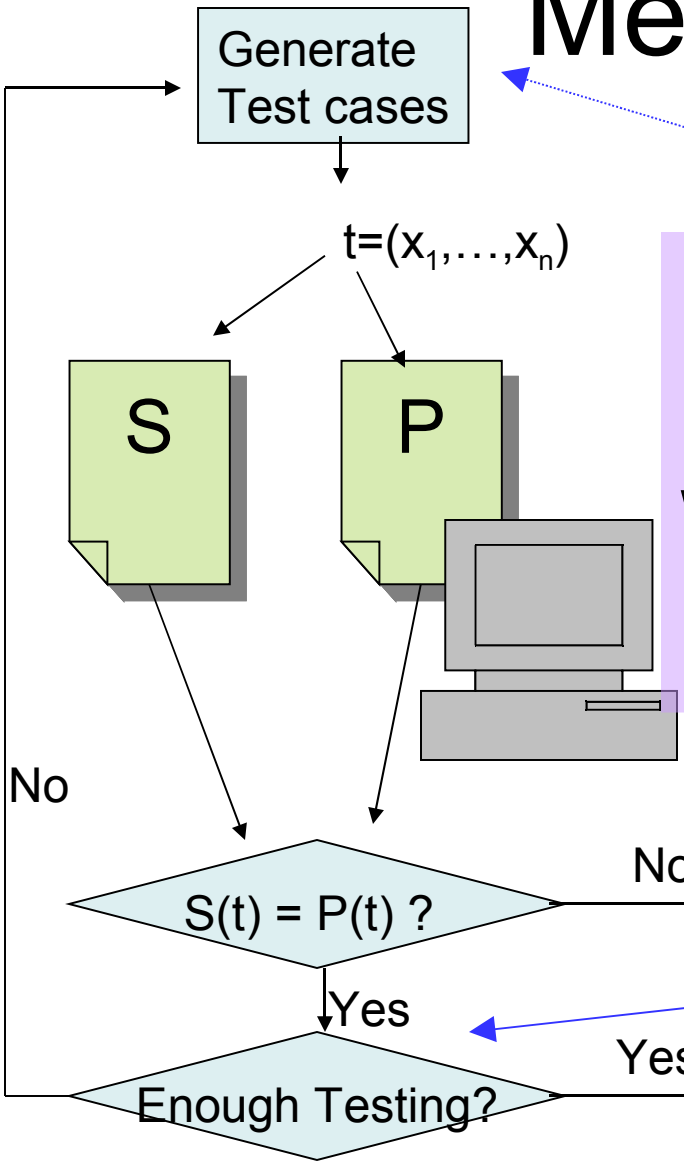
Test generation techniques

Generate/Accept test sets that are small enough to be usable in practice and are likely to detect faults.

Sources of information:
S: Black box
P: White box
Other: common errors, design, …

Test data adequacy criteria

“Meta”Research Issues



Test generation techniques

How costly are these techinques?
How likely are they to find faults?

Which technique is “best”?

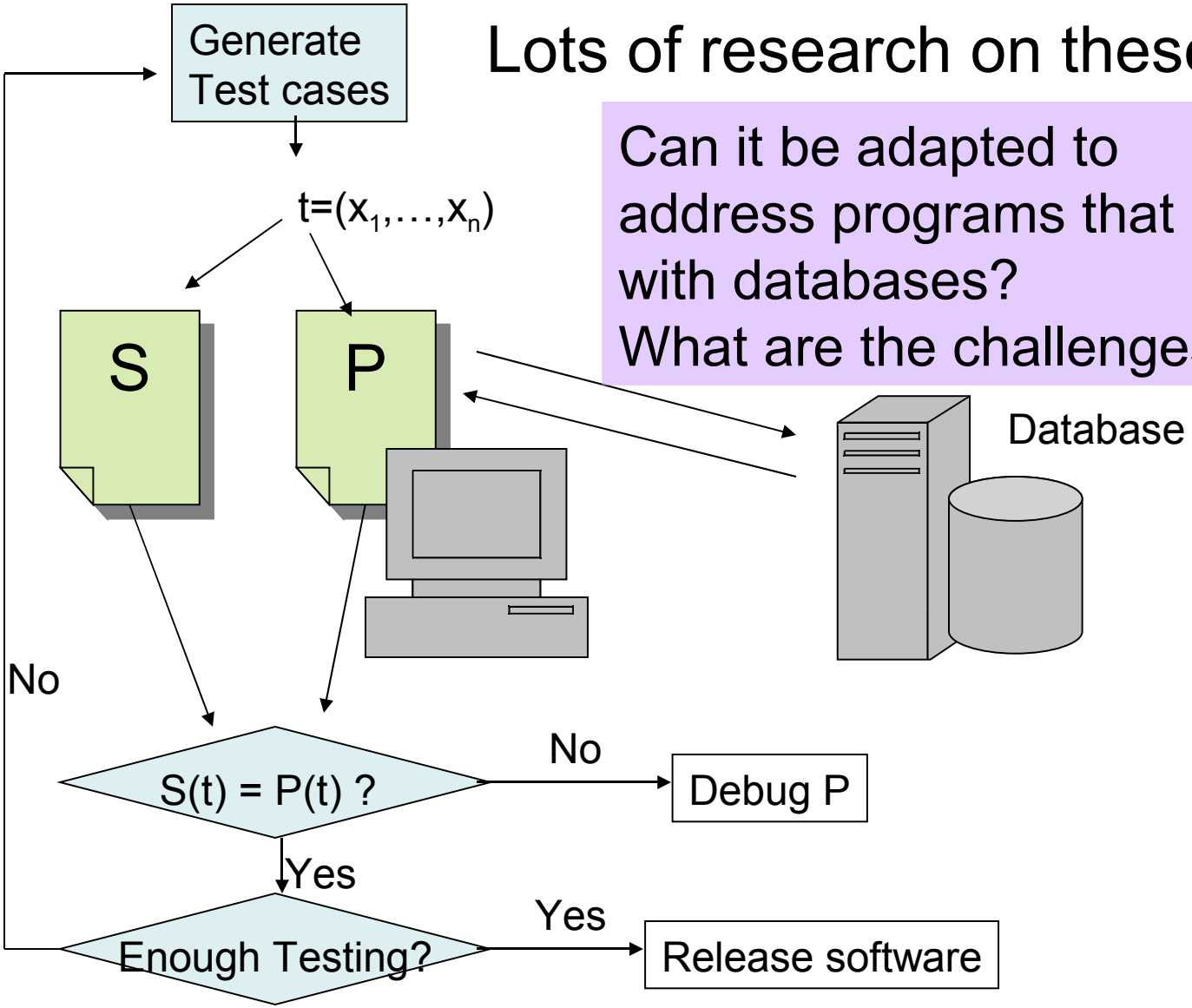
How do we address these questions?

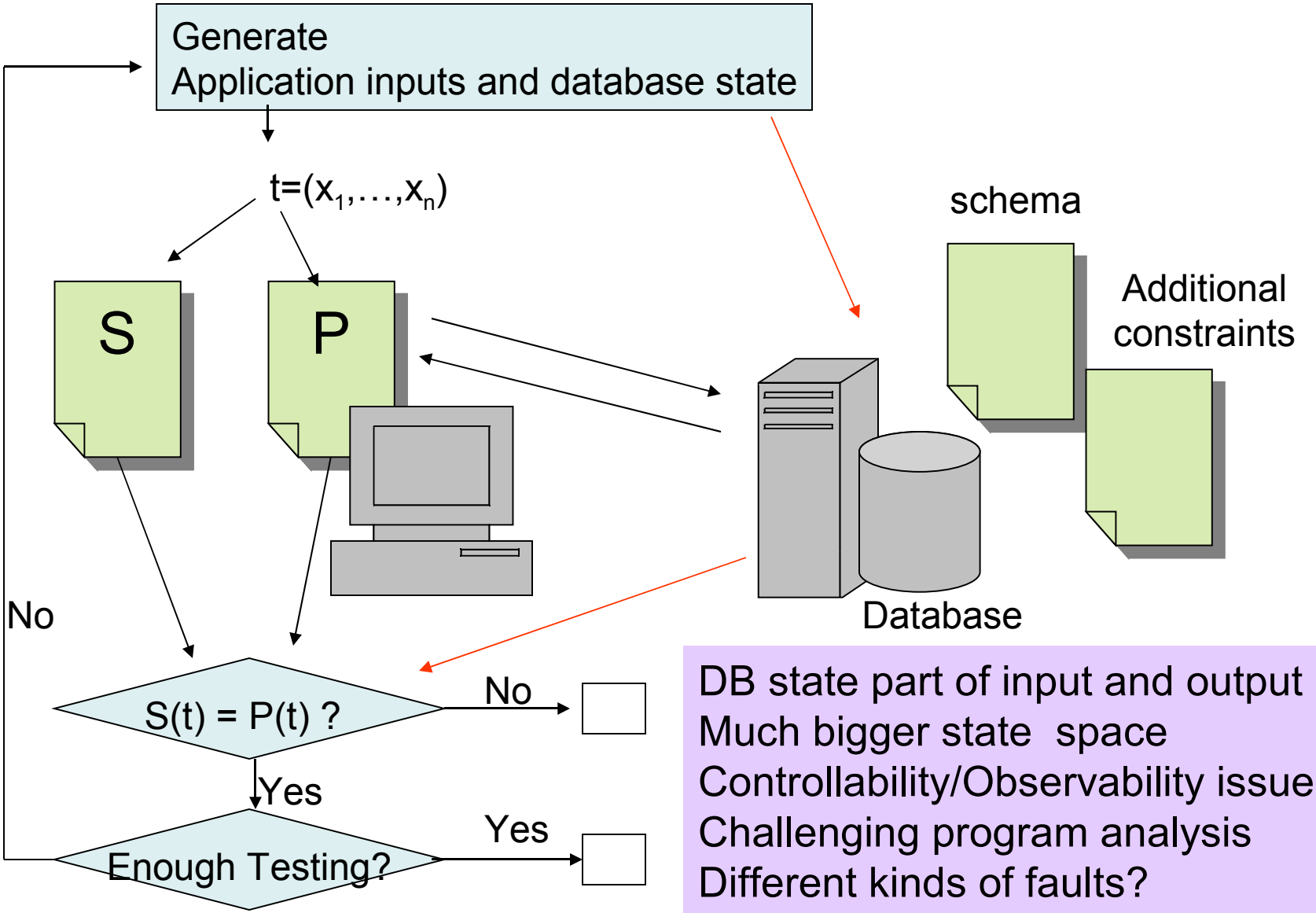
Test data adequacy criteria

NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Lots of research on these issues

Can it be adapted to address programs that interact with databases?
What are the challenges?



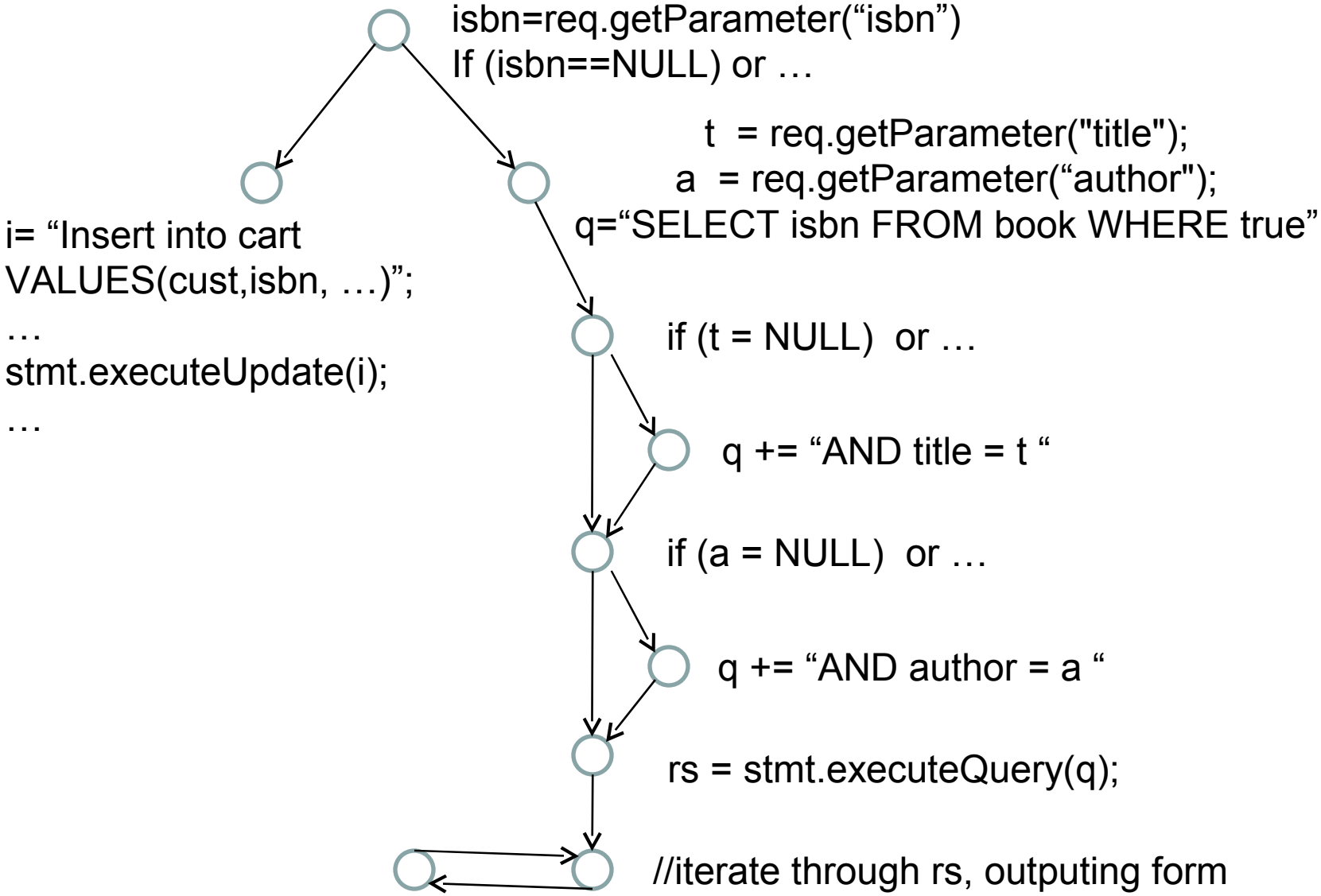


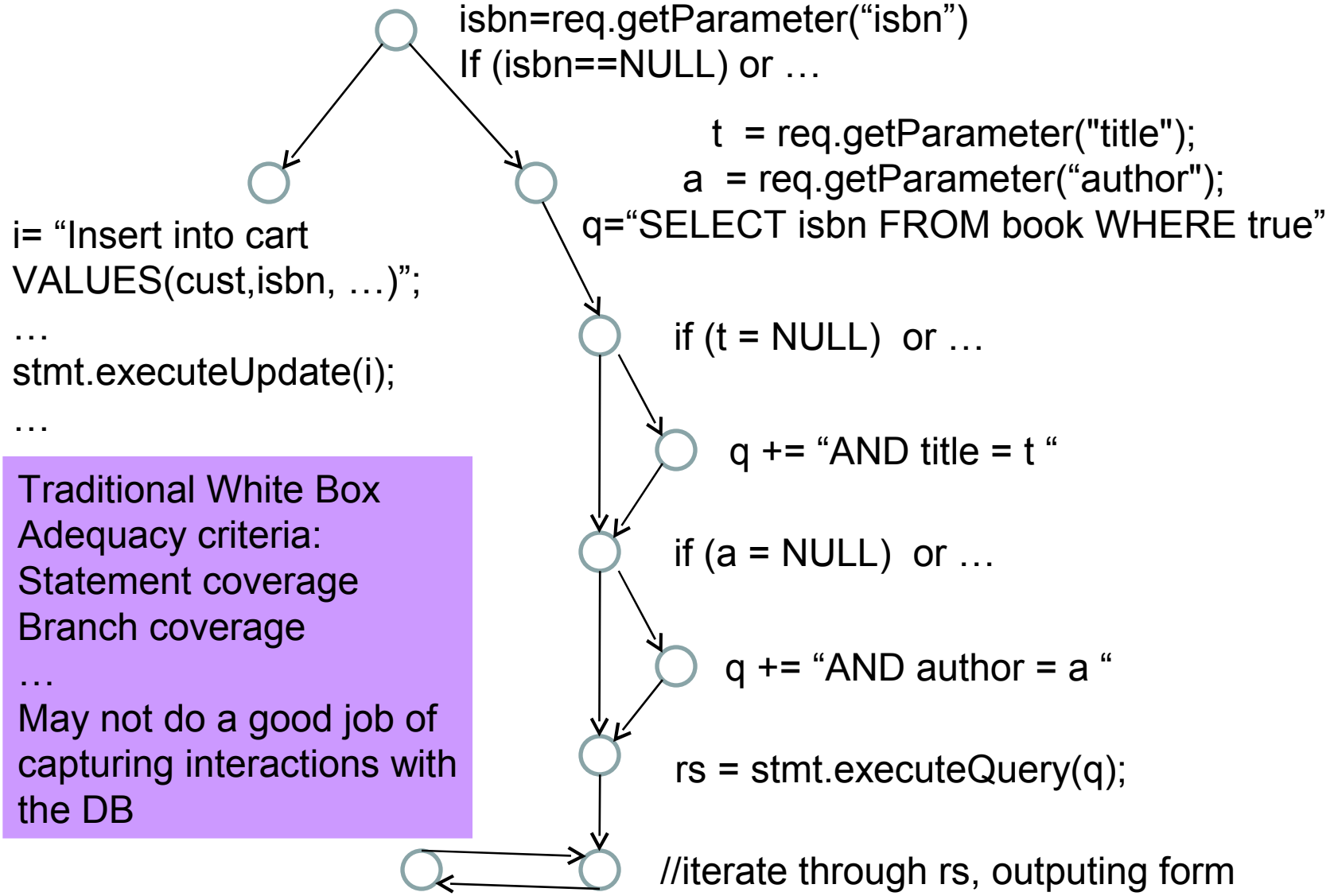
DB state part of input and output
Much bigger state space
Controllability/Observability issues
Challenging program analysis
Different kinds of faults?

NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Talk Outline

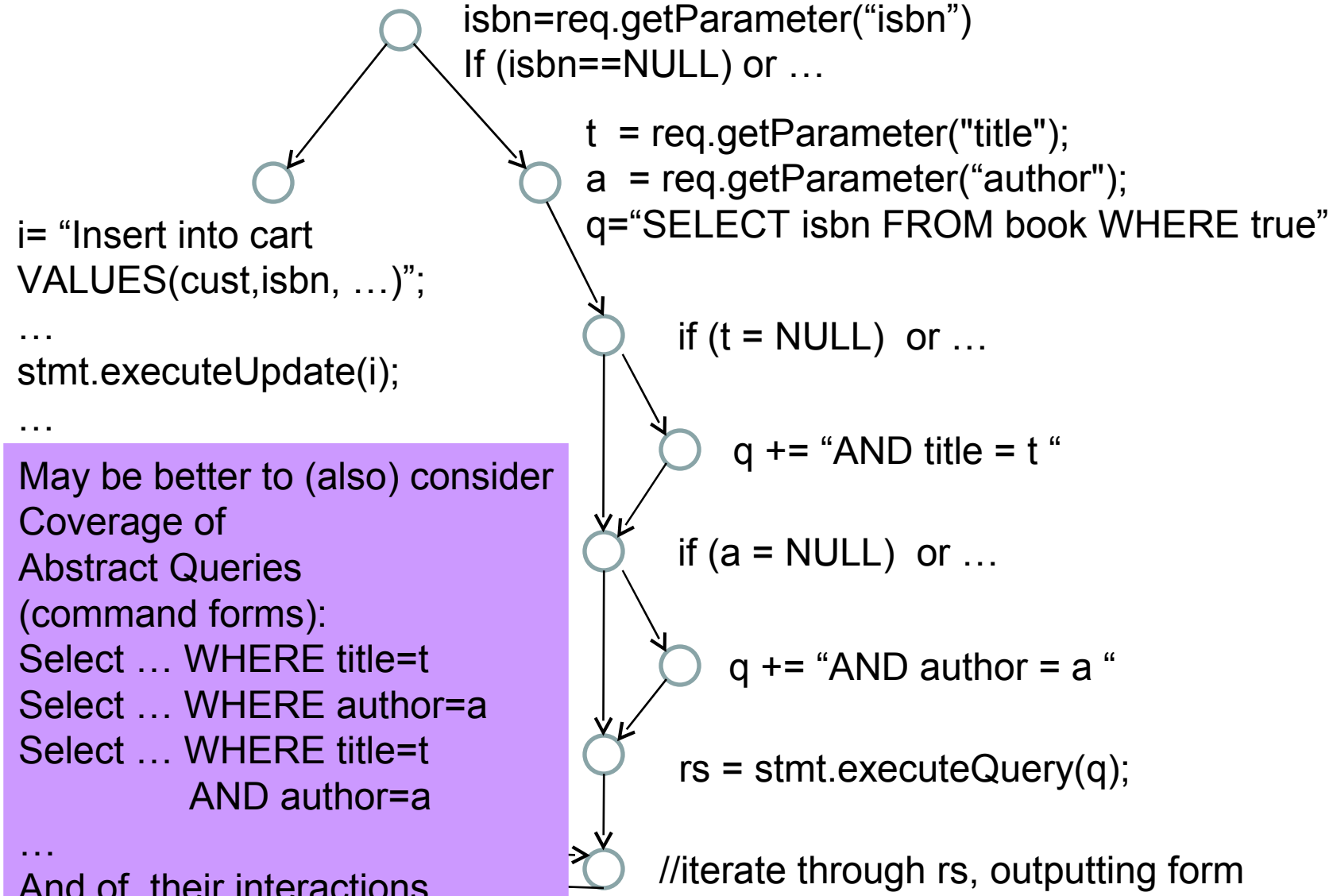
- ✓ Introduction
- Problems and Prospects for testing database applications
  - Test data adequacy criteria
  - Program analysis
  - Test data generation
  - Oracle Problem
  - Evaluation of testing techniques


isbn=req.getParameter(“isbn”)
If (isbn==NULL) or …
t = req.getParameter("title");
a = req.getParameter(“author");
q=“SELECT isbn FROM book WHERE true”
i= “Insert into cart
VALUES(cust,isbn, …)”;
…
stmt.executeUpdate(i);
…
if (t = NULL) or …
q += “AND title = t “
if (a = NULL) or …
q += “AND author = a “
rs = stmt.executeQuery(q);
//iterate through rs, outputing form



Traditional White Box Adequacy criteria:
Statement coverage
Branch coverage
…
May not do a good job of capturing interactions with the DB

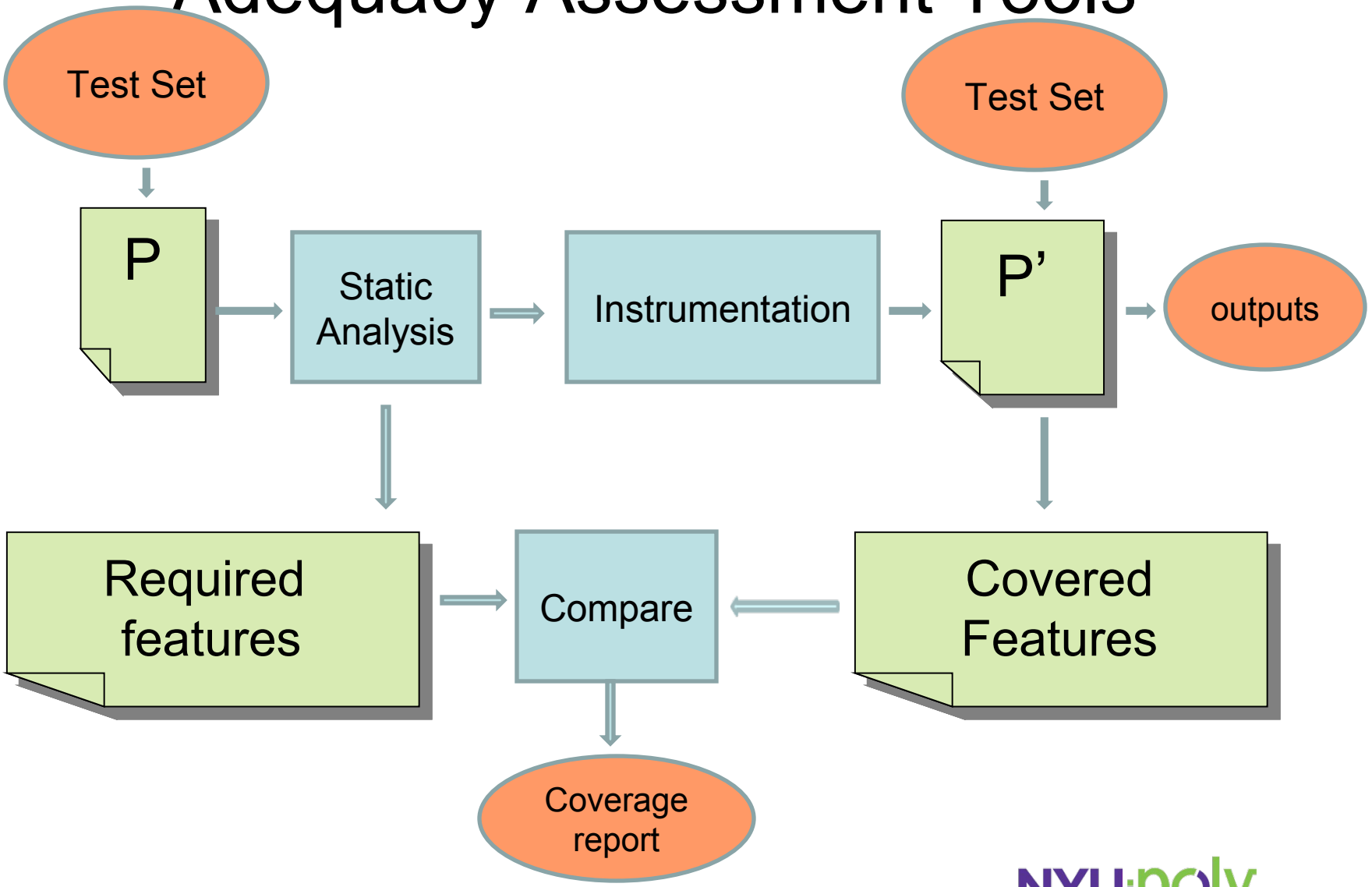
NYU·poly
POLYTECHNIC INSTITUTE OF NYU



May be better to (also) consider Coverage of Abstract Queries (command forms):
Select … WHERE title=t
Select … WHERE author=a
Select … WHERE title=t
AND author=a
…
And of their interactions
[Tang MASPLAS 06, Halfond ASE 06]

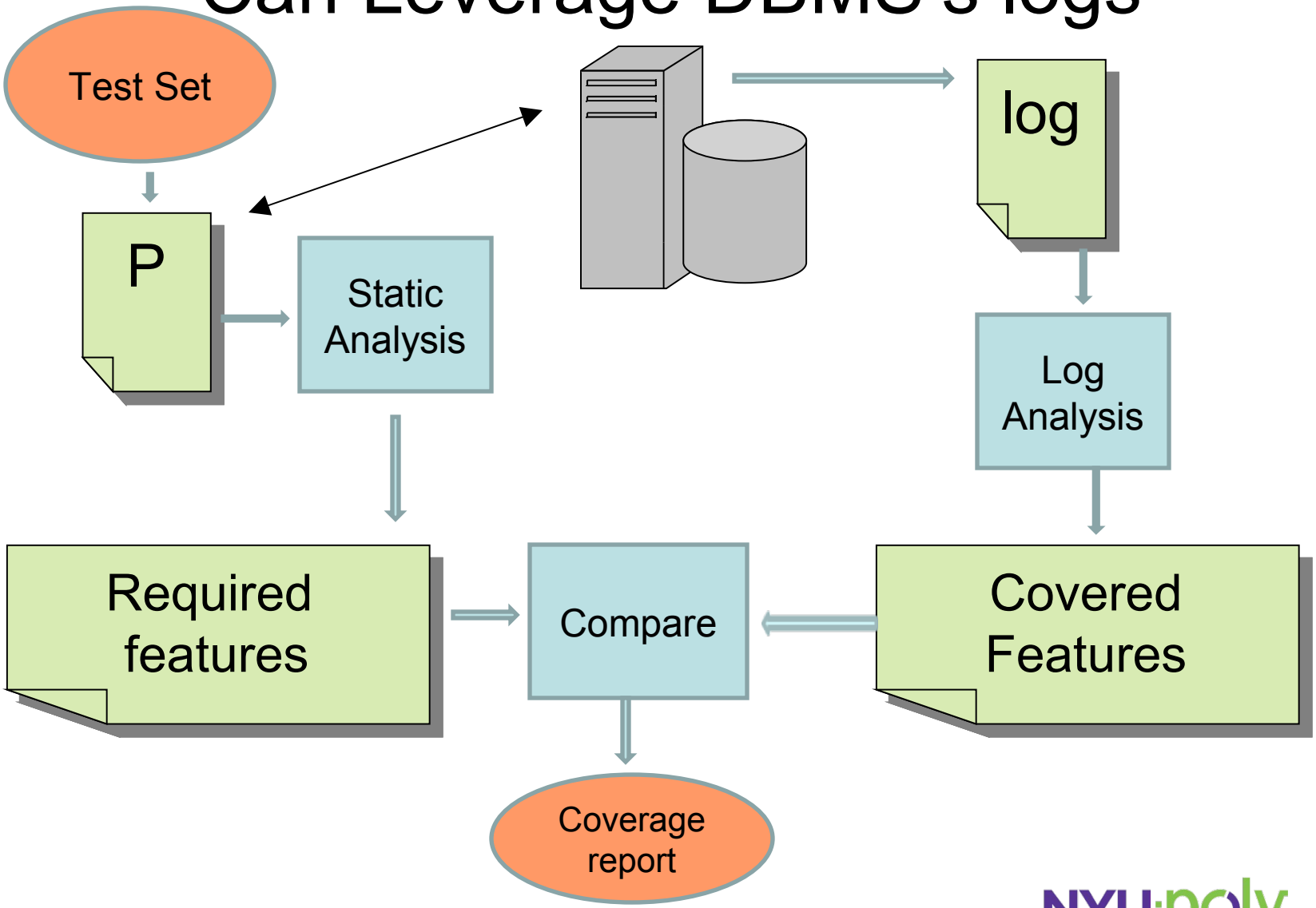
NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Adequacy Assessment Tools



NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Can Leverage DBMS's logs



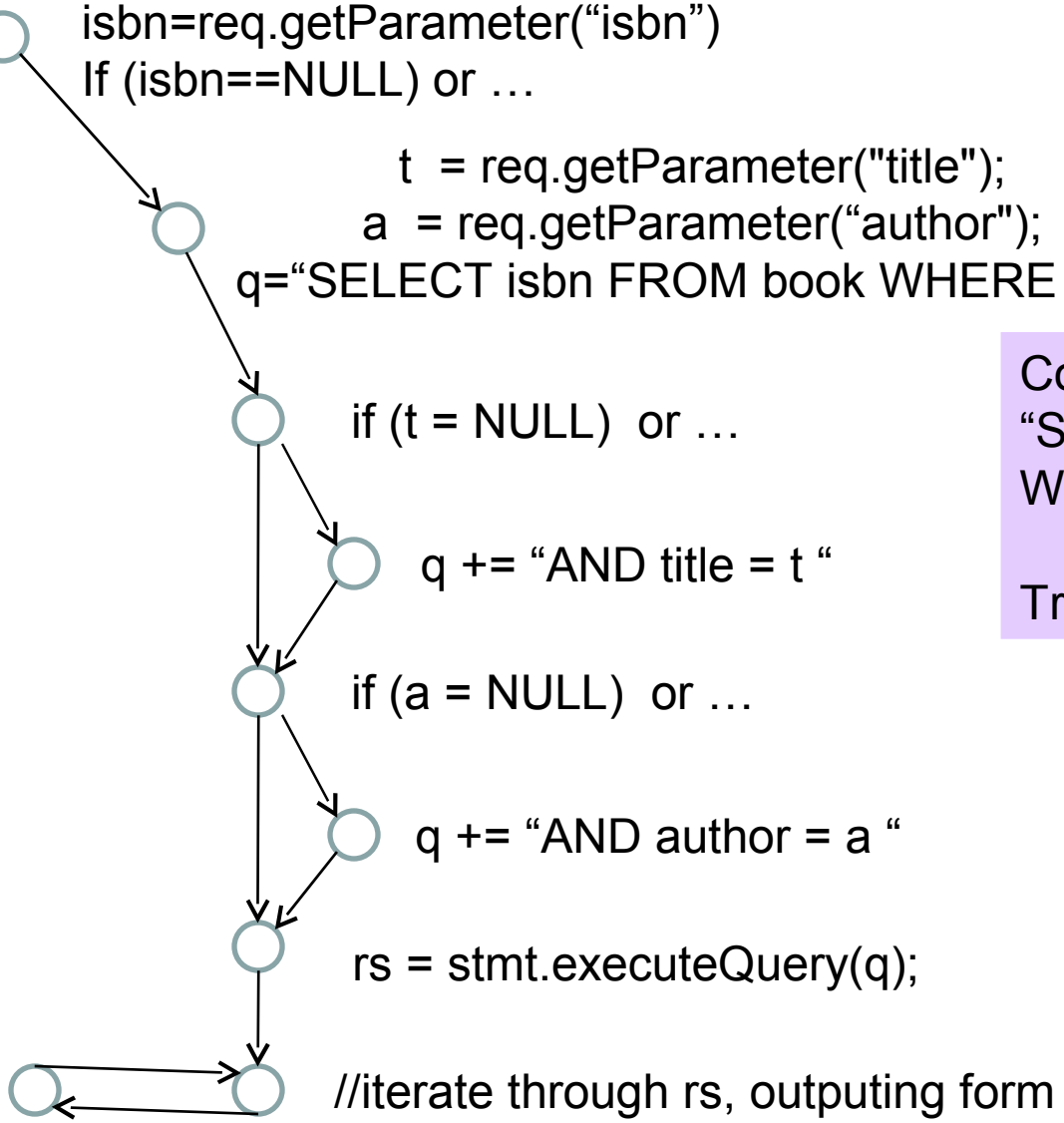
NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Program Analysis Problem

- In dynamic SQL, query strings are assembled at run-time
- To use abstract queries as basis of test adequacy (or test generation) criterion, need to determine set of possibilities statically
- Undecidable in general
- Set of possibilities at a given execution point can be approximated using data flow analysis (e.g. Java String Analyzer [Christensen et al SAS 2003])
  - Over-approximation safe, but leads to infeasible requirements
  - Analogous problems exist for traditional adequacy criteria
- Can also form the basis for statically detecting some faults (syntax errors, some type incompatibilities, unjustified cardinality assumptions, …) [Gould et al, ICSE 2004; DasGupta et al ICDE 2009]

Test Generation: Harder!

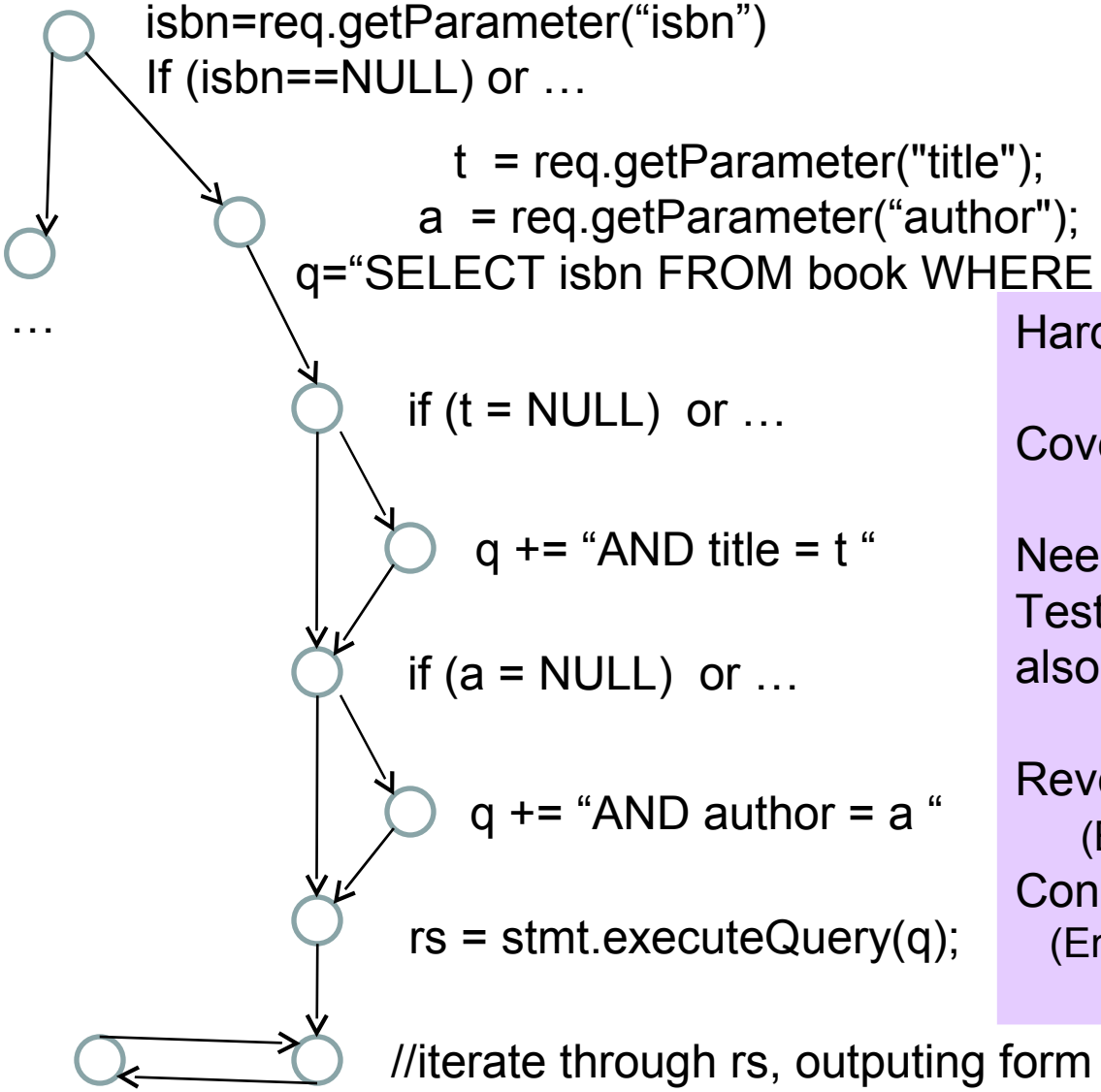
- Given some white box or black box adequacy criterion, want to generate tests that will cover given requirements
- Unsolvable in general for interesting white box criteria, but some approaches moderately successful in practice
- Some approaches for traditional programs:
  - Symbolic execution
  - Dynamic symbolic execution (concolic)
  - Search-based testing
  - Category partition testing (black box)



Cover
“SELECT isbn FROM book WHERE true AND author= ?”

Traditional approaches OK

NYU·poly
POLYTECHNIC INSTITUTE OF NYU



Harder test generation problem:

Cover back edge in loop

Need non-empty result set
Test generation technique must also consider DB state

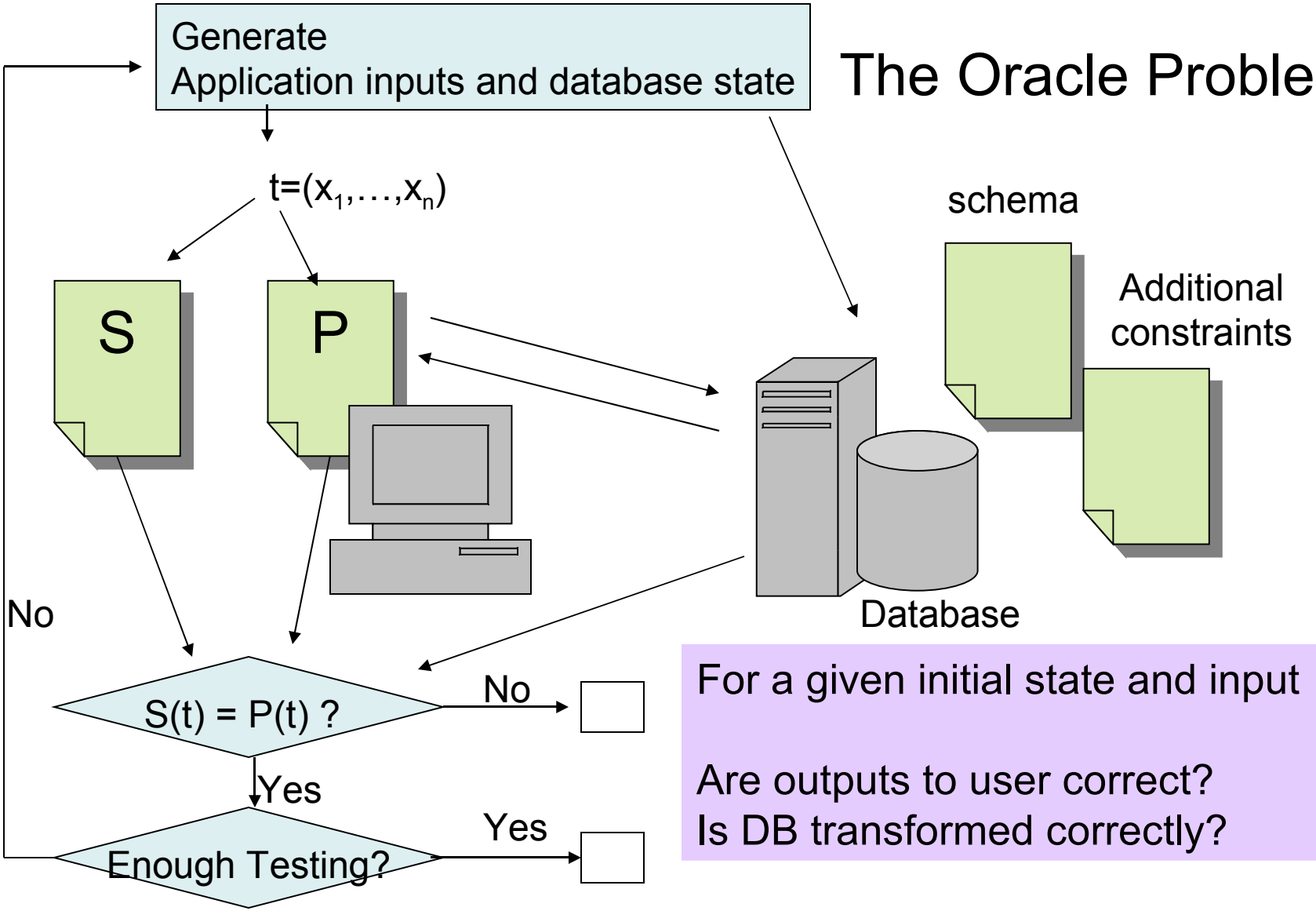
Reverse Query Processing
(Binnig, Kossmann, Lo; ICDE-07)
Concolic testing including DB state
(Emmi, Majumdar, Sen; ISSTA =07)

NYU·poly
POLYTECHNIC INSTITUTE OF NYU

AGENDA Approach

- Inspired by Category-Partition Testing (black box approach)
- Inputs
  - database schema
  - Info about how program inputs are used in application's SQL
  - Sample values, divided into data groups
  - test heuristics
  - info about expected behavior of test cases
  - Constraints on expected database state and outputs
- Goal: generate tests involving
  - various combinations of data groups
  - NULLs where possible
  - Duplicates where possible
  - …
- Outputs
  - Initial database state [Chays et al, ISSTA 00, STVR 04]
  - Application inputs [Chays, Shahid, Frankl DBtest08]
    - Generated by generating and executing queries
  - Hooks for validation of resulting DB state [Deng et al ICSE 05]

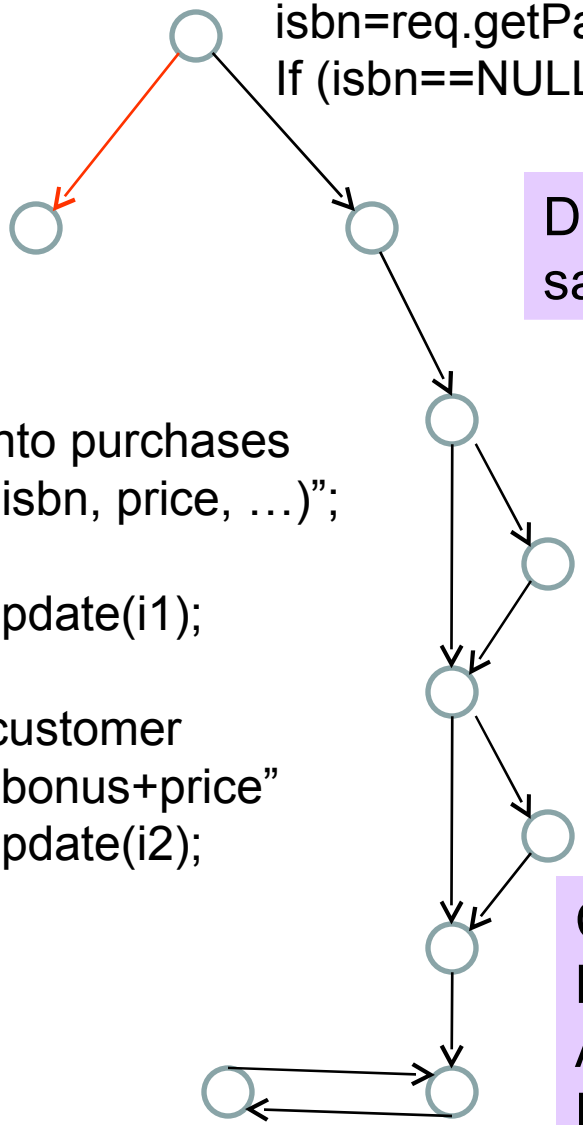
The Oracle Problem



For a given initial state and input

Are outputs to user correct?
Is DB transformed correctly?

NYU·poly
POLYTECHNIC INSTITUTE OF NYU



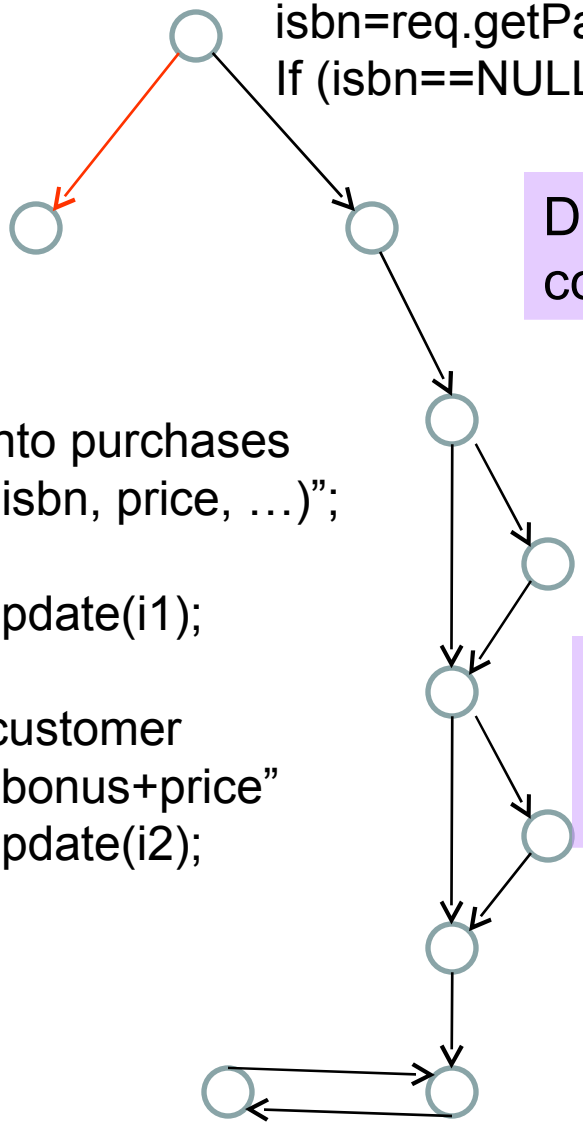

```
cust= …
price = …
i1= "INSERT into purchases
VALUES(cust,isbn, price, …)";
…
stmt.executeUpdate(i1);
…
i2="UPDATE customer
   SET bonus=bonus+price"
stmt.executeUpdate(i2);
```

Does DB state after execution satisfy constraints?

(SELECT cust, bonus FROM customer)
==
(SELECT cust, SUM(price)
FROM purchases
GROUP BY cust)

Complex constraint involving
Multiple tables
Aggregation
Not enforced by DBMS

NYU·poly
POLYTECHNIC INSTITUTE OF NYU




```
cust= …
price = …
i1= "INSERT into purchases
VALUES(cust,isbn, price, …)";
…
stmt.executeUpdate(i1);
…
i2="UPDATE customer
   SET bonus=bonus+price"
stmt.executeUpdate(i2);
```

Does application transform DB correctly when test is executed?

New row inserted with correct data; Bonus updated correctly

Checking these involves examining multiple DB states (before and after)

NYU·poly
POLYTECHNIC INSTITUTE OF NYU

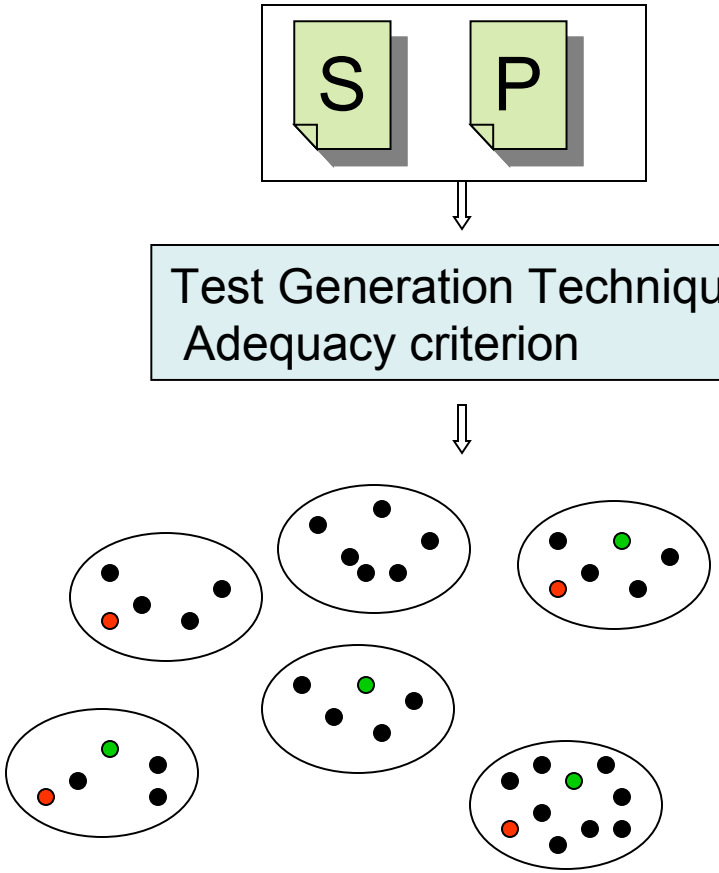
Agenda State Checker

- Based on analysis of application and schema, automatically create
  - new tables whose which
    - Collect data from multiple tables and aggregates
    - track attribute values before/after execution
    - Check constraints on single rows of the auxilliary tables
  - Triggers to insert rows into auxilliary tables when base tables are updated
  - Row level constraints on auxilliary tables, which are checked at end of transaction
- State generation revisited: Must also consider these complex constraints during state generation

Issues in checking outputs

- Need good spec
- What aspects of output are to be considered?
  - E.g., exact html produced vs contents of tables, ignoring tags
- May be more feasible to analyze result sets than to analyze output
  - Instrument code, inserting checkers after query execution points
- Some bugs can be caught with partial specs that only consider some of the data or metadata

Evaluation of Testing Techniques



Non-determinism:
Lots of different test sets can be generated / accepted
Test sets have different properties:
- size
- faults detected

Evaluation must
- sample space of test sets
- (control for test set size)

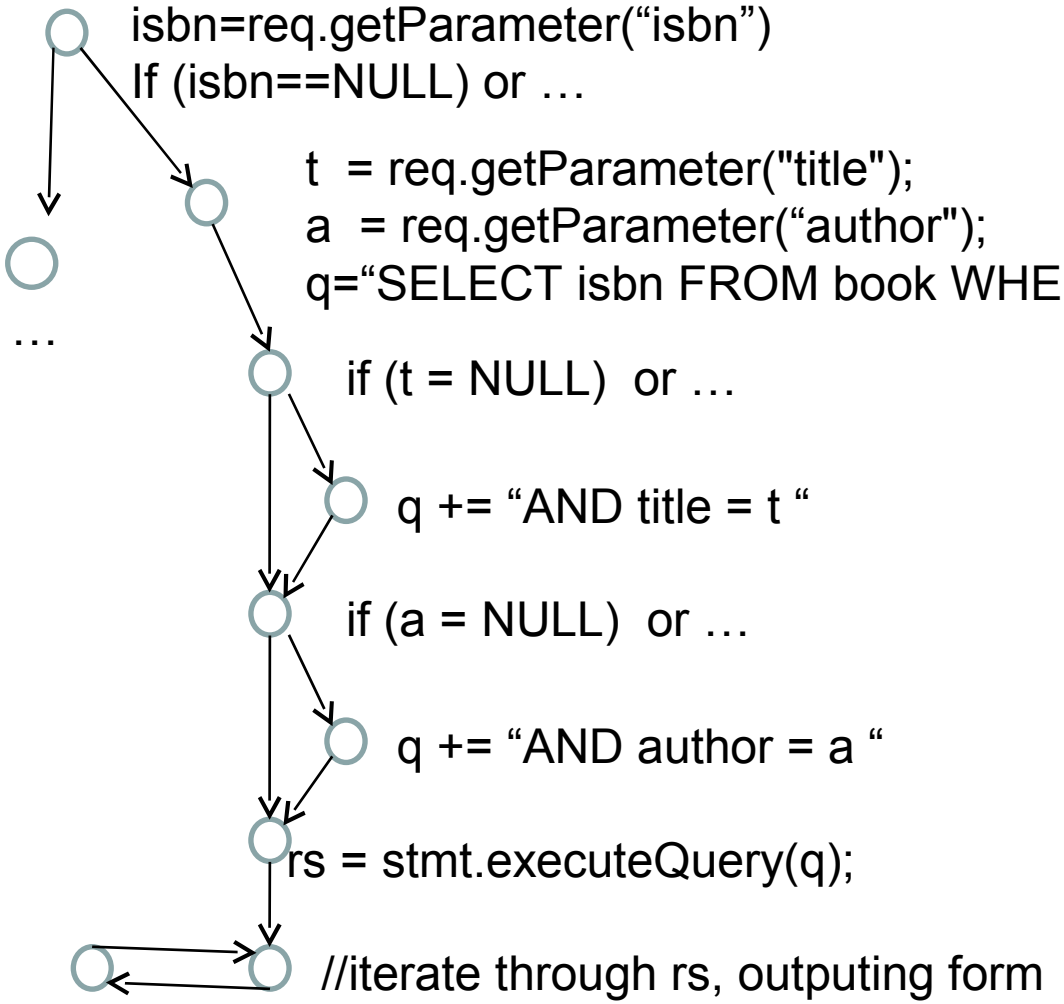
DB application
- state space part of input
- larger space of test sets

NYU·poly
POLYTECHNIC INSTITUTE OF NYU

Subject Programs

- Ideally
  - (Lots of) Real programs with (lots of) real faults
- Alternatively
  - Real programs with artificial faults
  - Often generated using program mutation
  - Some evidence that this is OK [Andrews et al, ICSE-06]
- DB Application
  - What kind of artificial faults?
  - One option is mutants of SQL statements

    [Tuya et al, Mutation Analysis 2006, IST 2007]



Abstract Query:
SELECT isbn FROM book
WHERE true AND title = ?
AND author = ?

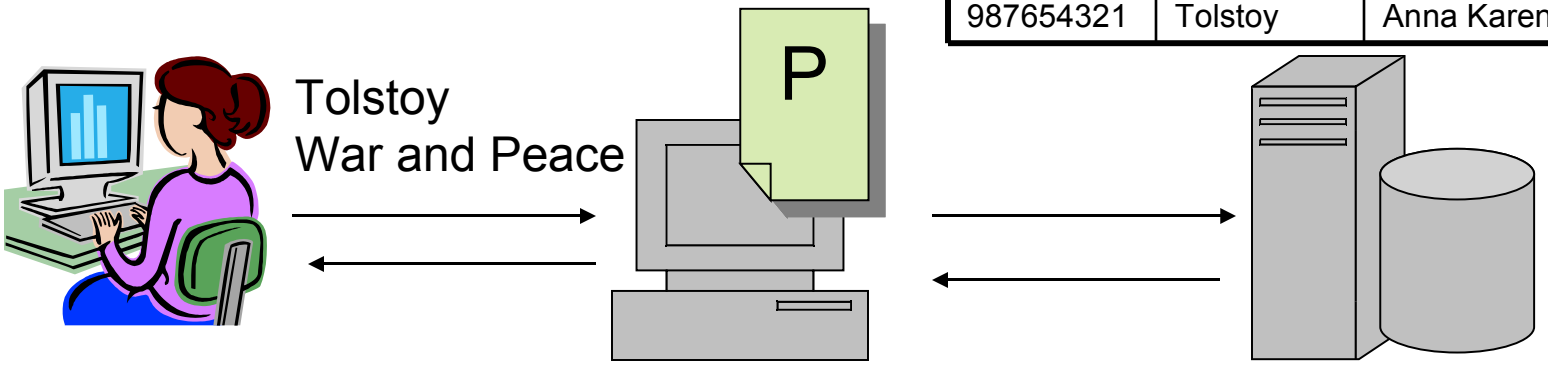
Some Abstract Mutants:
SELECT isbn FROM book
WHERE true AND title = ?
OR author = ?

SELECT isbn FROM book
WHERE true AND title = ?
AND title = ?

SELECT isbn FROM book
WHERE true AND title = ?
AND author = NULL
…

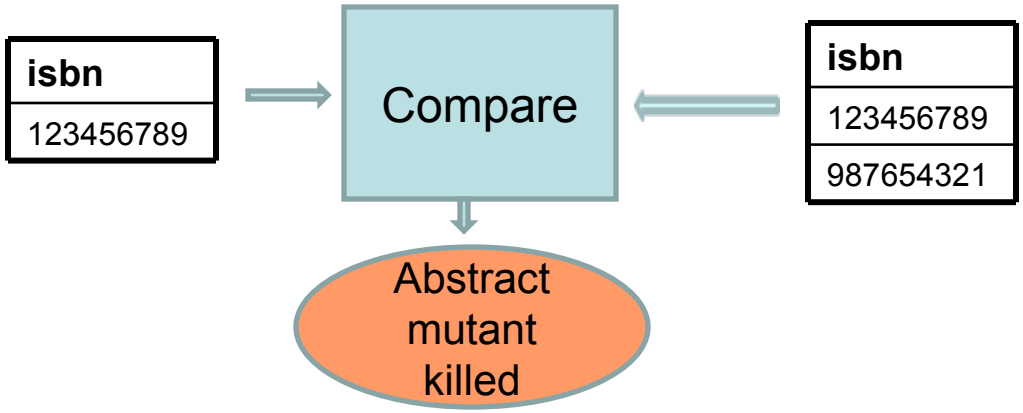
NYU·poly
POLYTECHNIC INSTITUTE OF NYU

isbn	author	title
123456789	Tolstoy	War and Peace
987654321	Tolstoy	Anna Karenina



QUERY:
SELECT isbn FROM book
WHERE true AND title = 'War and Peace'
AND author = 'Tolstoy'

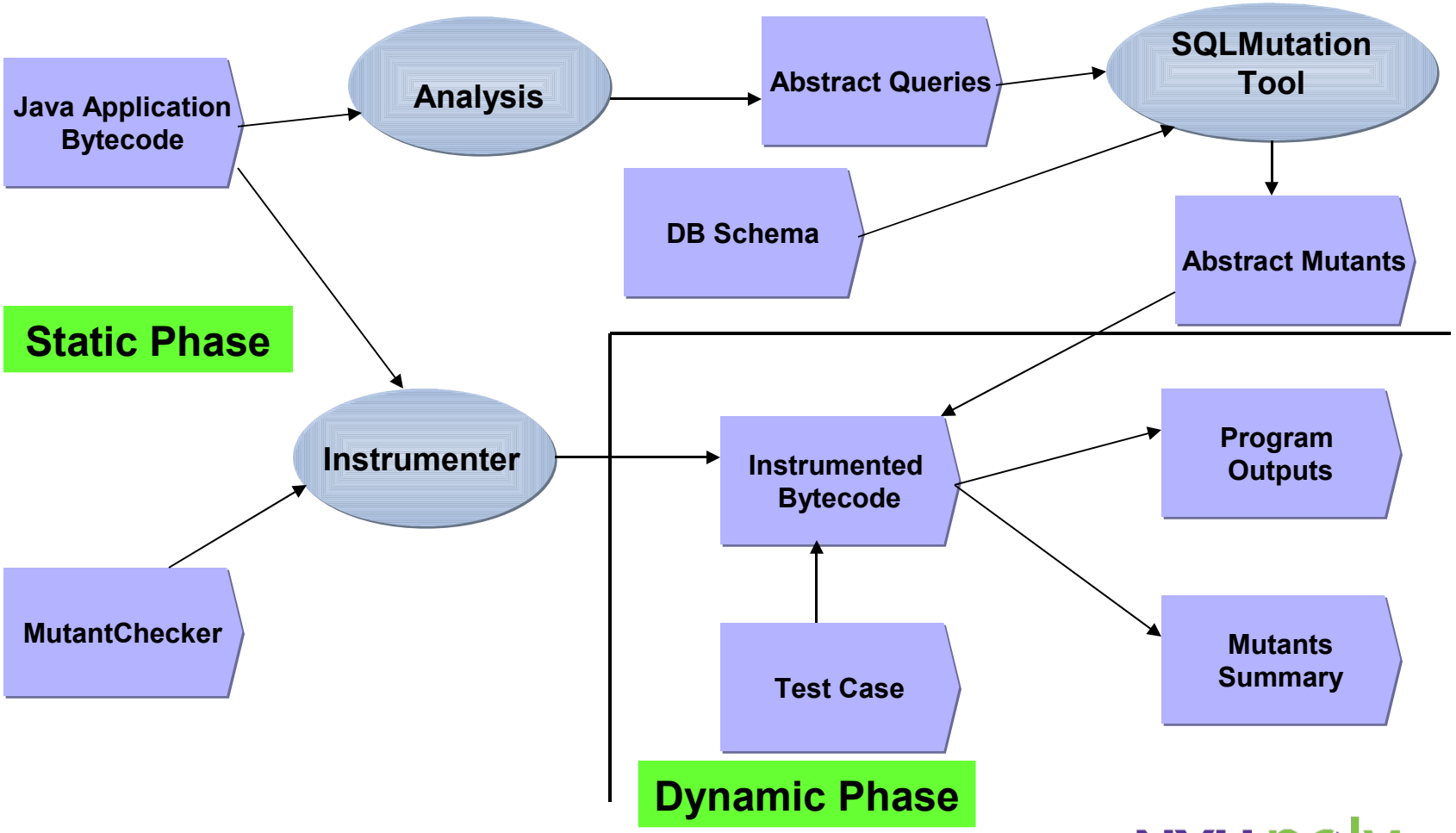
MUTANT:
SELECT isbn FROM book
WHERE true AND title = 'War and Peace'
OR author = 'Tolstoy'



NYU·poly
POLYTECHNIC INSTITUTE OF NYU

JDAMA: Java Database Application Mutation Analyzer

[Zhou, Frankl ICST 2009]



NYU·poly
POLYTECHNIC INSTITUTE OF NYU

JDAMA dynamic phase

- Instrumented code
  - Initializes mutant map
  - Calls CheckMutant at each execution point
    - Matches concrete query to abstract query
    - Finds and concretizes each abstract mutant
    - Executes each abstract mutant and compares result sets to those of original query
    - Mutant killed if results sets differ (weak mutation)

Handling DB modifications

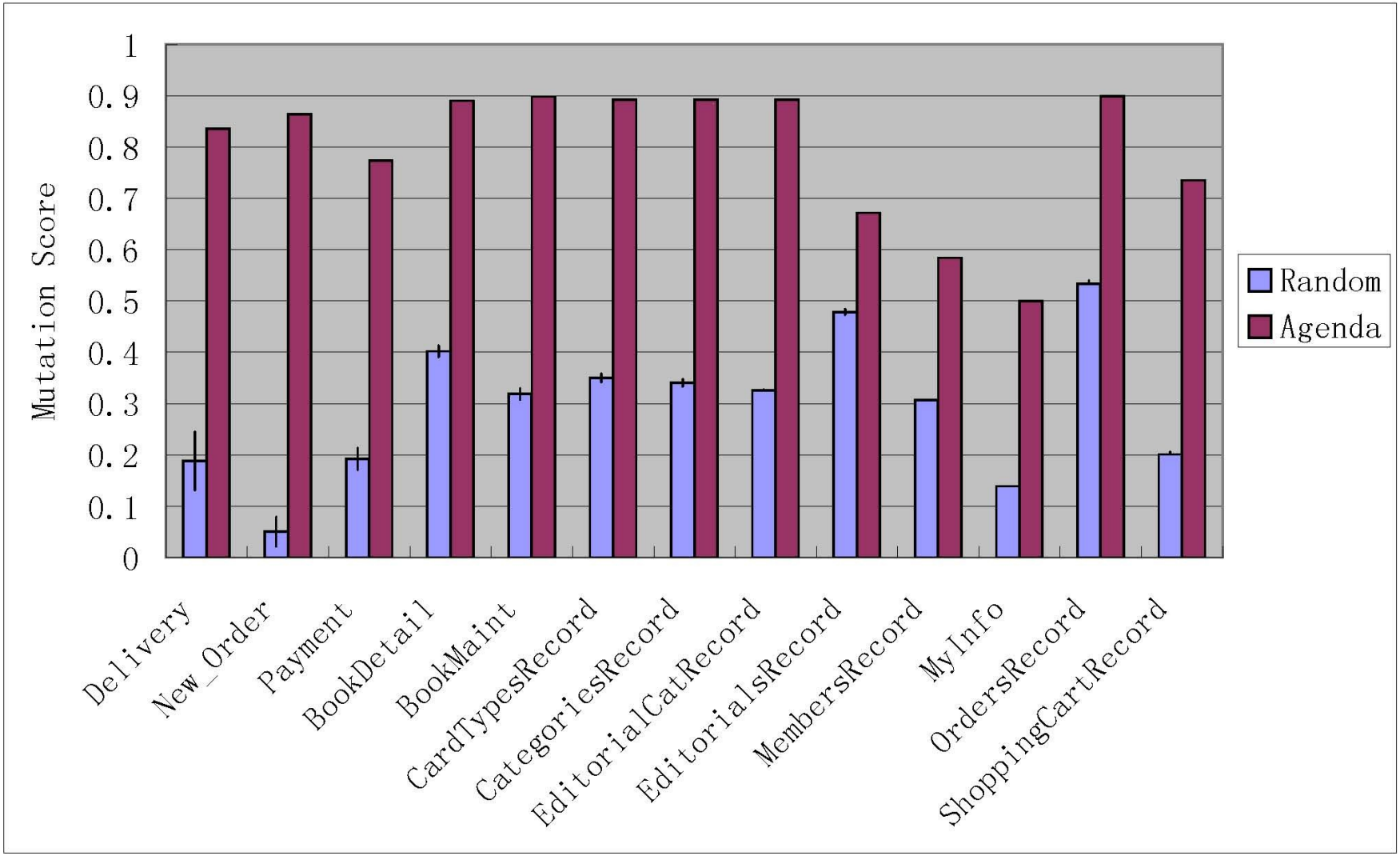
- Executing insert, delete, update statement or its mutants changes DB state, which interferes with execution of subsequent SQL statements (and/or mutants)
- Inferential checking:
  - Instead of executing mutants, infer whether they would change DB differently than original statement
  - Derive and execute queries that characterize how original statement and mutant would modify DB state, execute them, compare results

Experiment Subjects

name	LOC	selects	Deletes updates	mutants
TPC-C	4471	20	10	2782
Bookstore	19402	23	18	603
Classifieds	10702	5	4	135
Empl Dir	5529	5	4	145
Events	7164	6	6	163
Portal	16089	15	12	431

* Used one initial DB state and moderately small test case universe for each subject

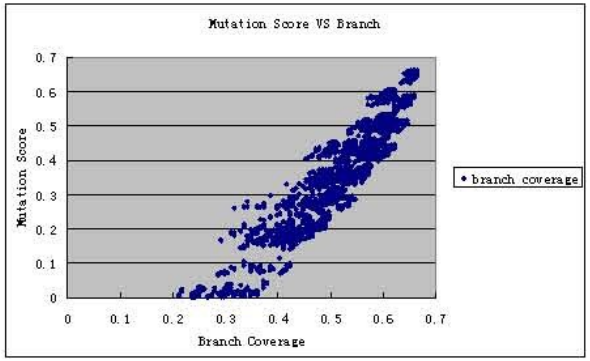
Agenda vs Random Generation



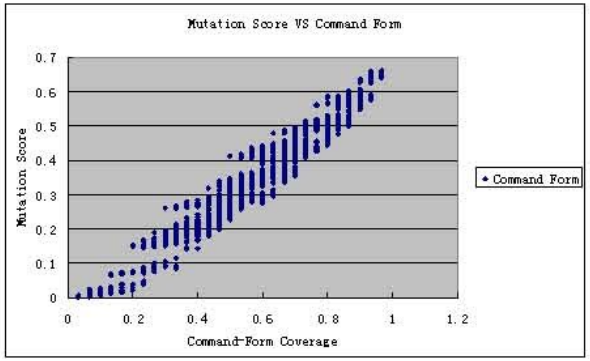
Comparing Adequacy Criteria

- Generate universe of initial DB states and test cases.
- Execute test cases and determine which features each covers and which mutants each kills
- Select large number of test sets from this universe
  - Optionally, control test set size
- For each test set determine
  - Size
  - Statements, branches, command forms (or other features) covered
  - Percentage of mutants killed (as measure of effectiveness)

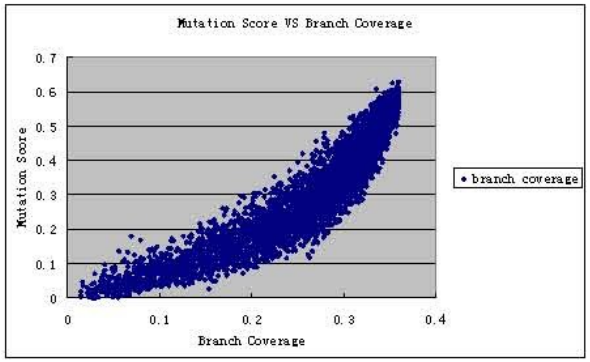
Adequacy Criteria Comparisons



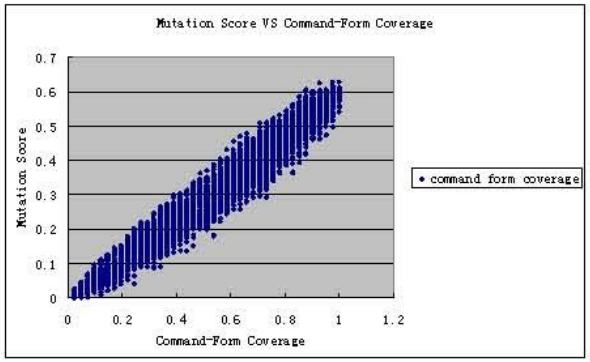
(a) TPC-C:Branch Coverage



(b) TPC-C:Command-Form Coverage



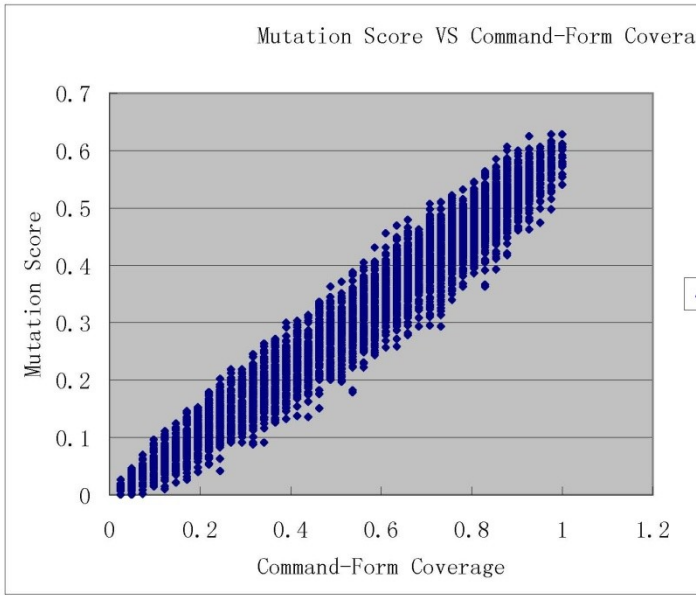
(c) Bookstore:Branch Coverage

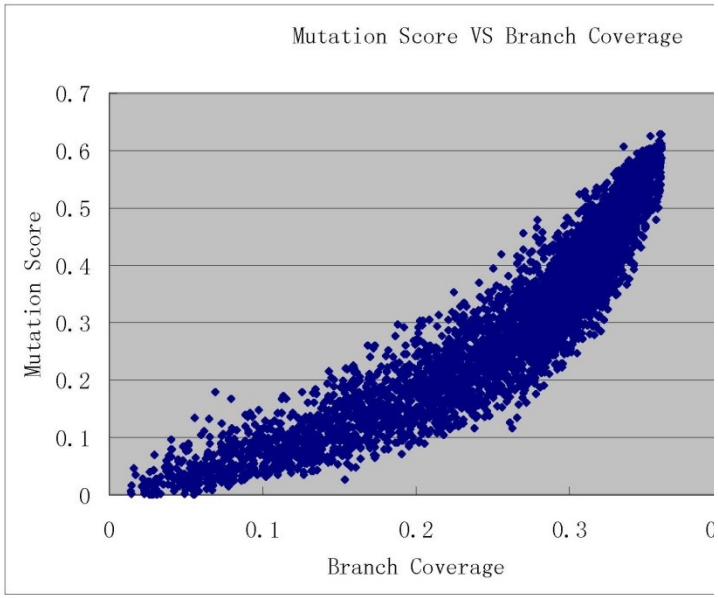


(d) Bookstore:Command-Form Coverage

Figure 1: Relationship between coverage and effectiveness

Adequacy Criteria: Bookstore





NYU·poly
POLYTECHNIC INSTITUTE OF NYU

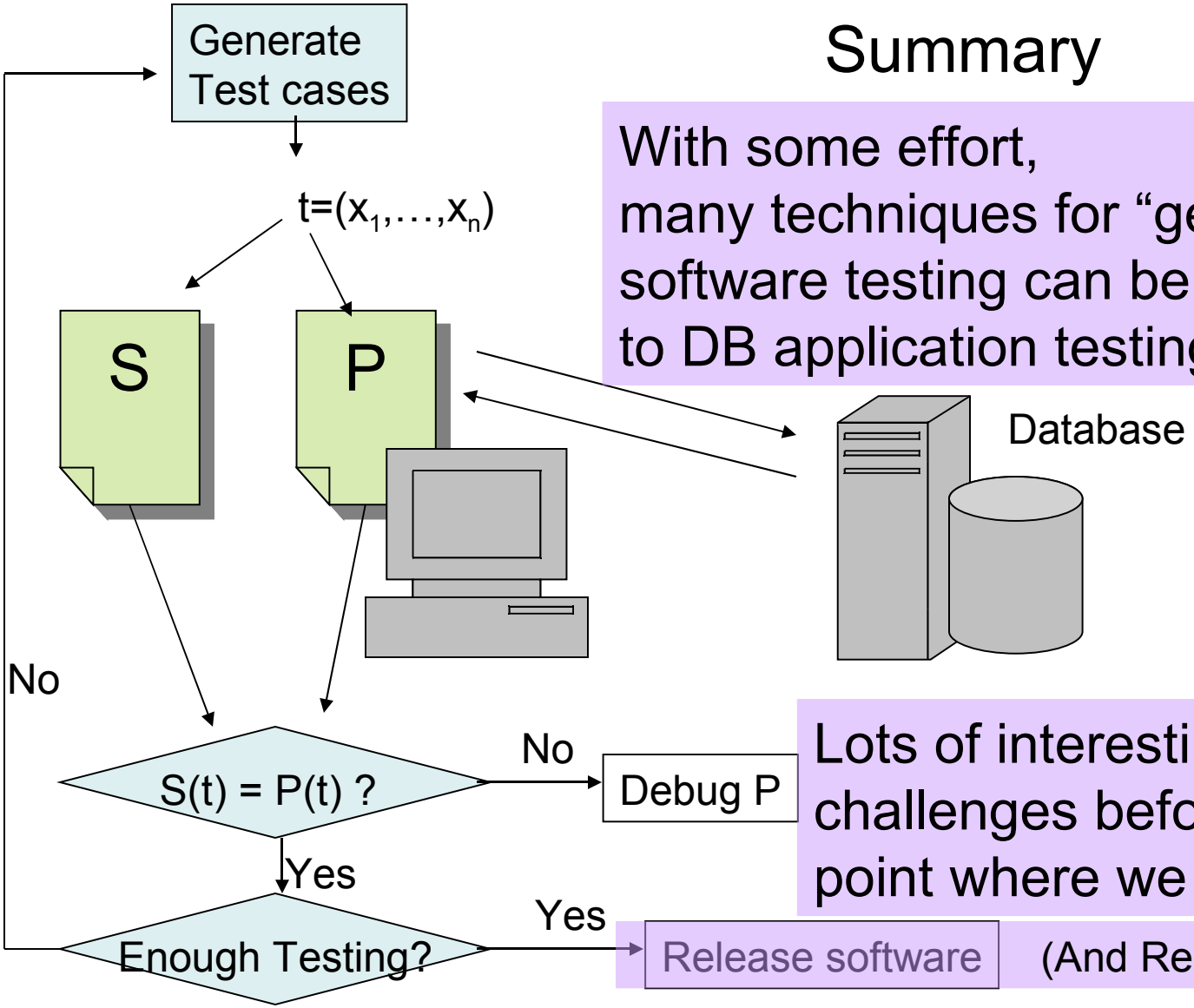
Experiment Conclusions

- On these subjects
  - AGENDA generated test sets were much more effective than random test sets of the same sizes
  - Test sets achieving high branch coverage were a little more effective than those reaching high command form coverage levels (!)
- Many caveats:
  - Small test universe
  - Only one initial DB state for each subject
  - Small number of subjects

How can DBMS vendors help application testing researchers?

- More access to SQL parser
  - E.g. lex, yacc files or similar precise high level descriptions
- This would make it easier for testing tool developers to create accurate SQL analysis components

Summary



With some effort,
many techniques for "general" software testing can be adapted to DB application testing

Lots of interesting research challenges before we reach point where we can

(And Relax)

NYU·poly
POLYTECHNIC INSTITUTE OF NYU