NC STATE UNIVERSITY Computer Science

Substra: a Framework for Automatic Generation of Integration Tests

Hai Yuan **Tao Xie**

Department of Computer Science
North Carolina State University, USA

http://ase.csc.ncsu.edu/

Motivation

- Software components interact with each other through component interfaces
  - Integration testing to verify correctness of component interactions
- Specifications can help integration testing: test-input generation and behavior checking
  - But specifications often don't exist in practice

Substra: automatic integration-test generation based on dynamic inference

Example: ATM Integration Tests

```
ATM thisATM = new ATM(42, "NCSU","Wachovia",
  null);

Session thisSession = new Session(thisATM);
Card thisCard = new Card(1);

Transaction thisTrans =
  Transaction.makeTransaction(thisATM,
 thisSession, thisCard, 42, 0);

thisTrans.performTransaction();
```

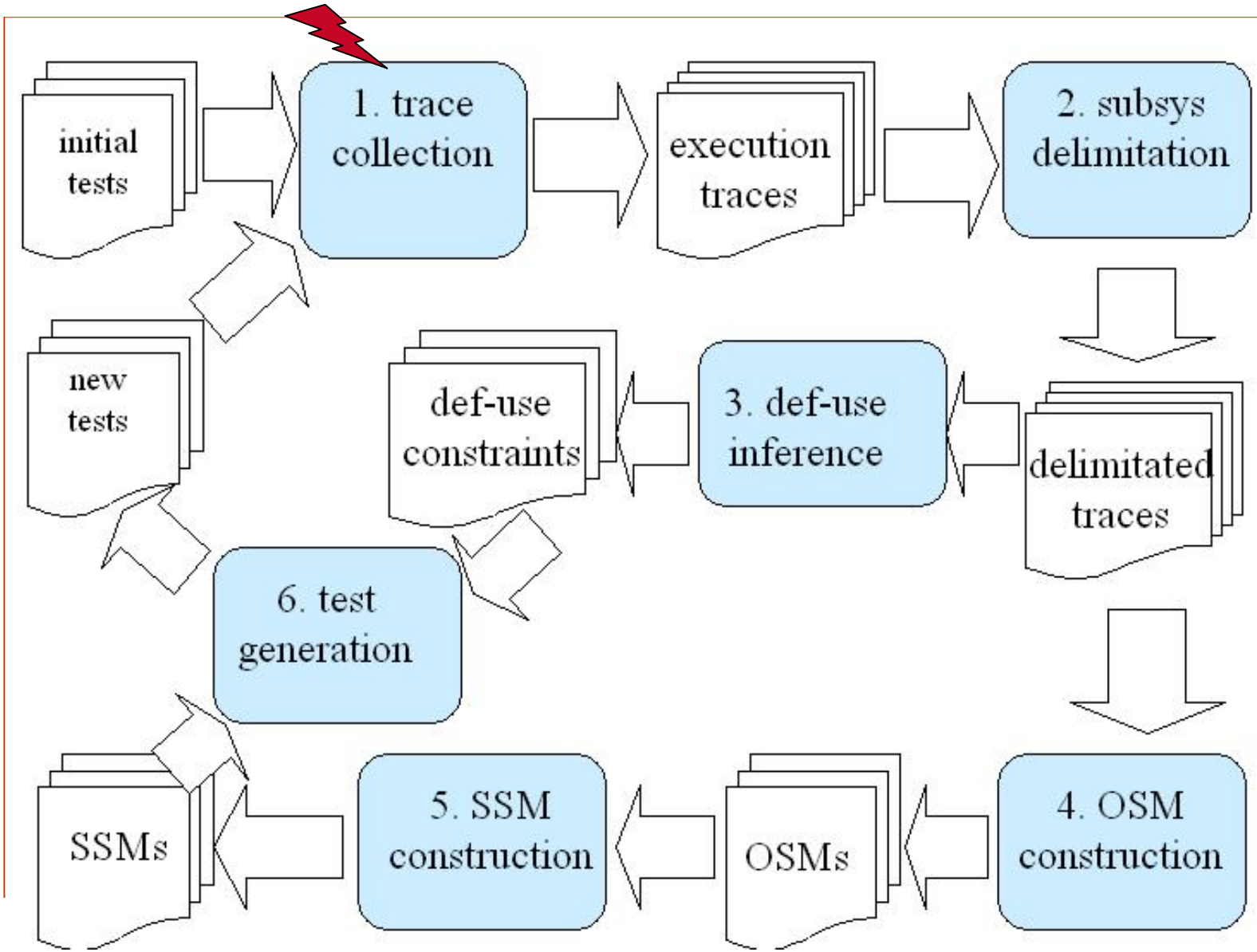
- def-use constraints
- sequencing constraints

New Approach

- Infer specification-like *constraints*, e.g.,
  - def-use constraints
  - sequencing constraints

  on a *subsystem* interface
  from existing *runs*, e.g.,
  - manually written (system/integration) tests
  - normal operations
- Generate new tests based on inferred constraints
  - abstract away primitive method arguments

Substra Framework

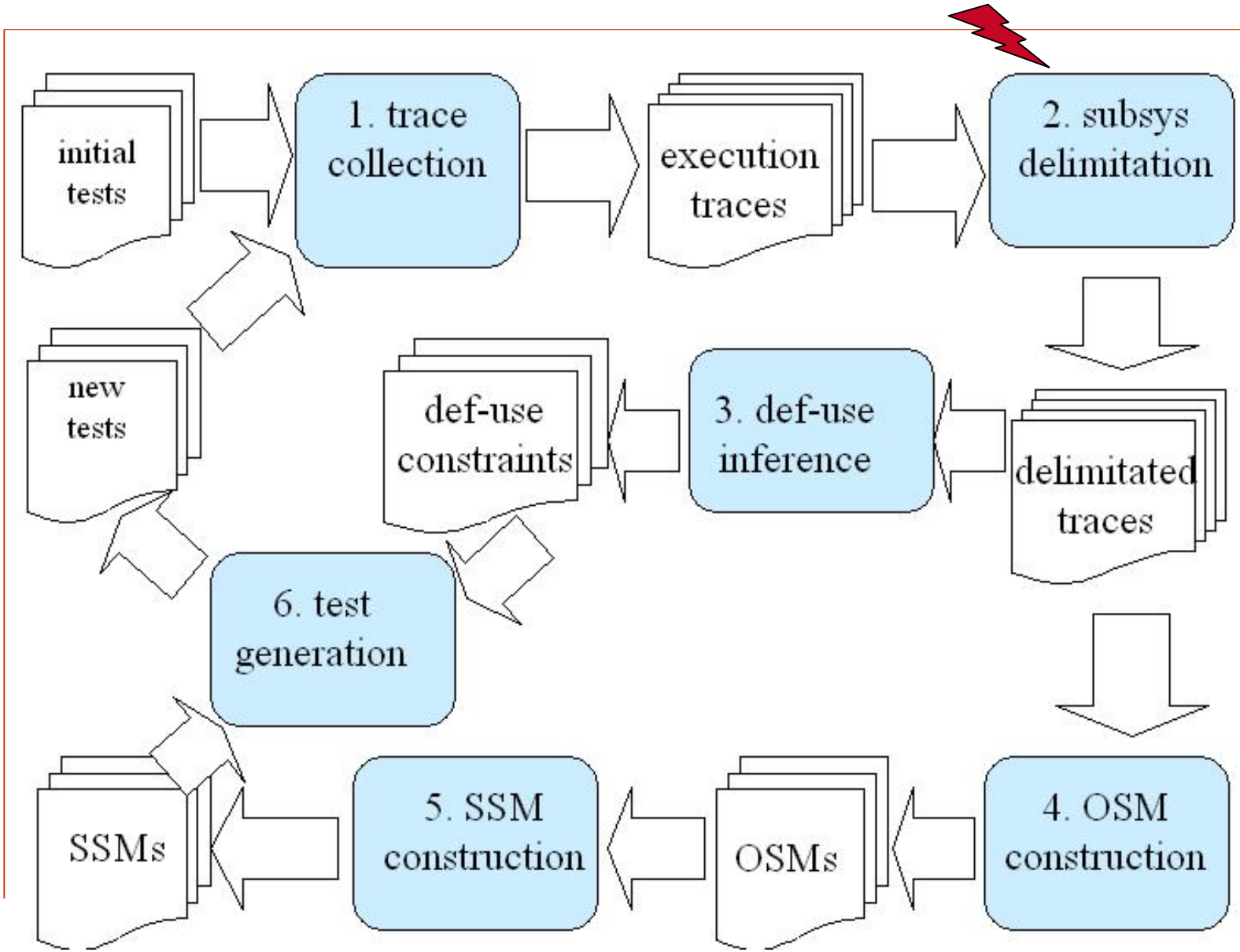


Trace Collection

- Inputs: initial tests/normal operations
- Outputs: execution traces

- Developed based on Daikon Java front-end [Ernst et al. 01]
  - Collect states at method entries and exits
  - State of an object: values of fields transitively reachable from the object

Substra Framework

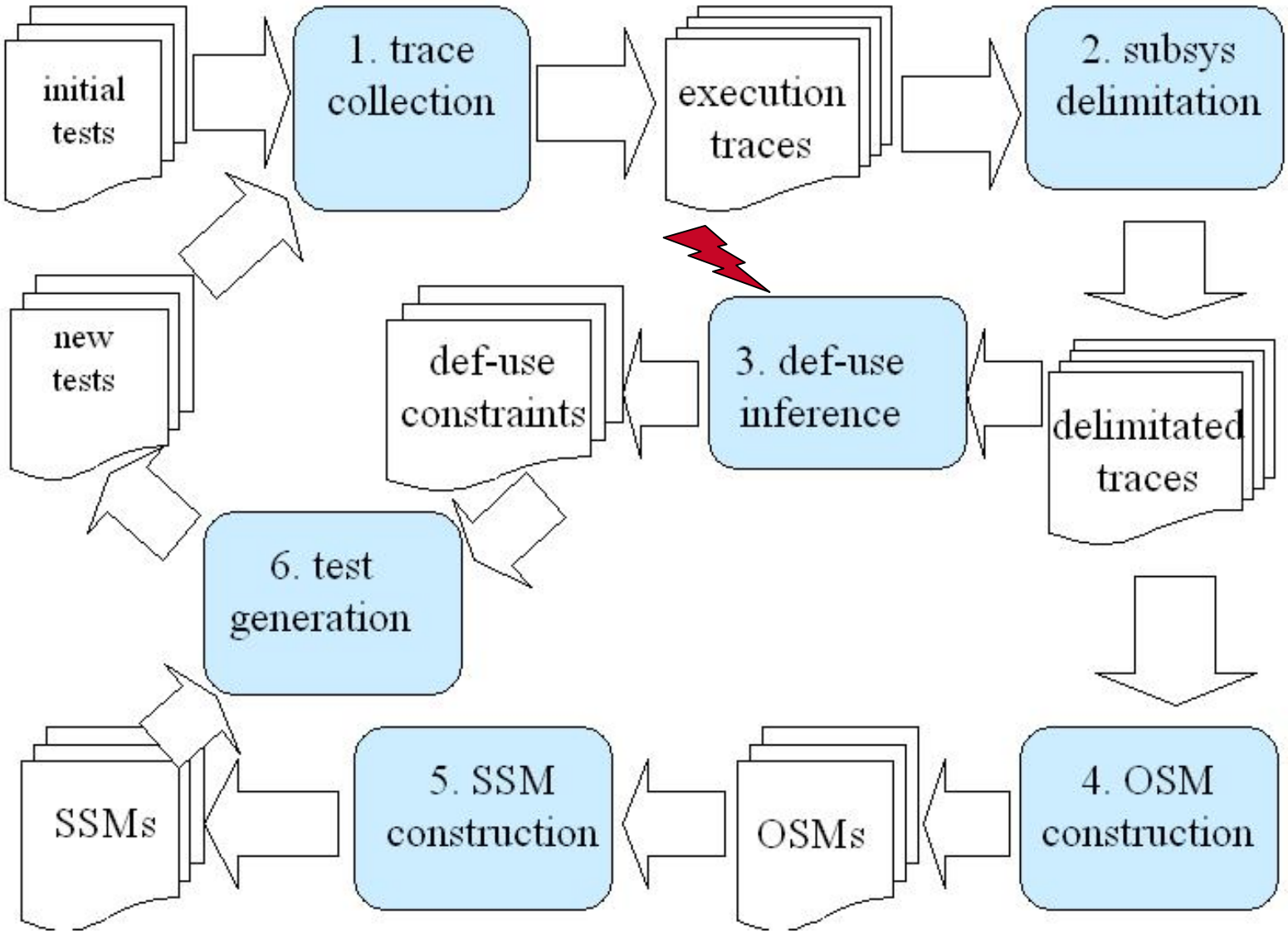


Subsystem Delimitation

- Inputs: execution traces
  + user-defined scope of subsystem e.g.,
    a package or several classes
- Outputs: delimited traces

- Boundary method call
  - a method call within the subsystem scope
  - whose caller is not within the subsystem scope

Substra Framework

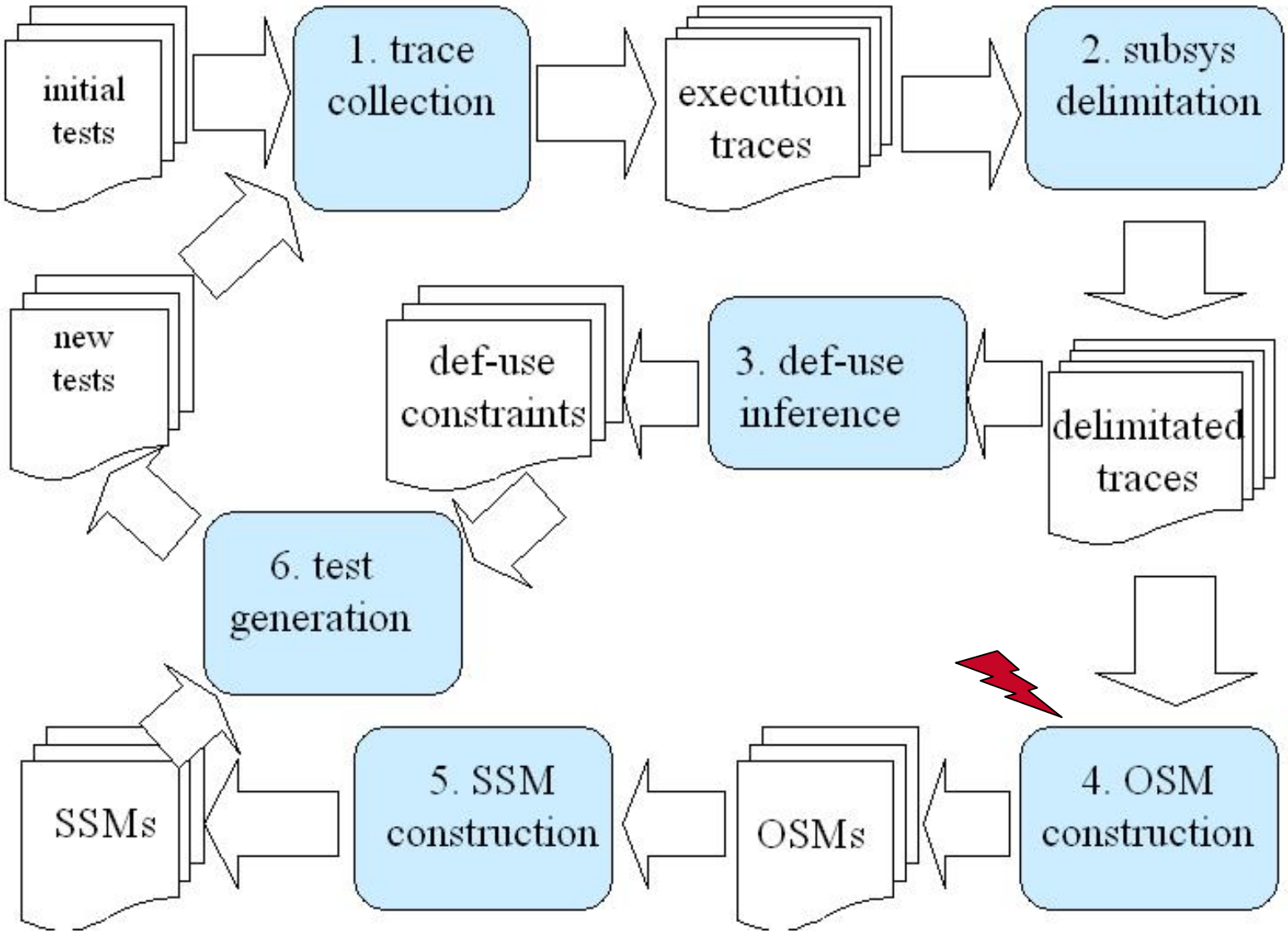


Def-Use Inference

- Inputs: delimited traces
- Outputs: def-use constraints

- Keep track of each boundary method call info
  - reference of return object (def)
  - reference of method-argument object (use)
  - reference of receiver object (use)
- Use object-reference equivalence to infer def-use constraints
  - stored as guard condition for the method call

Substra Framework

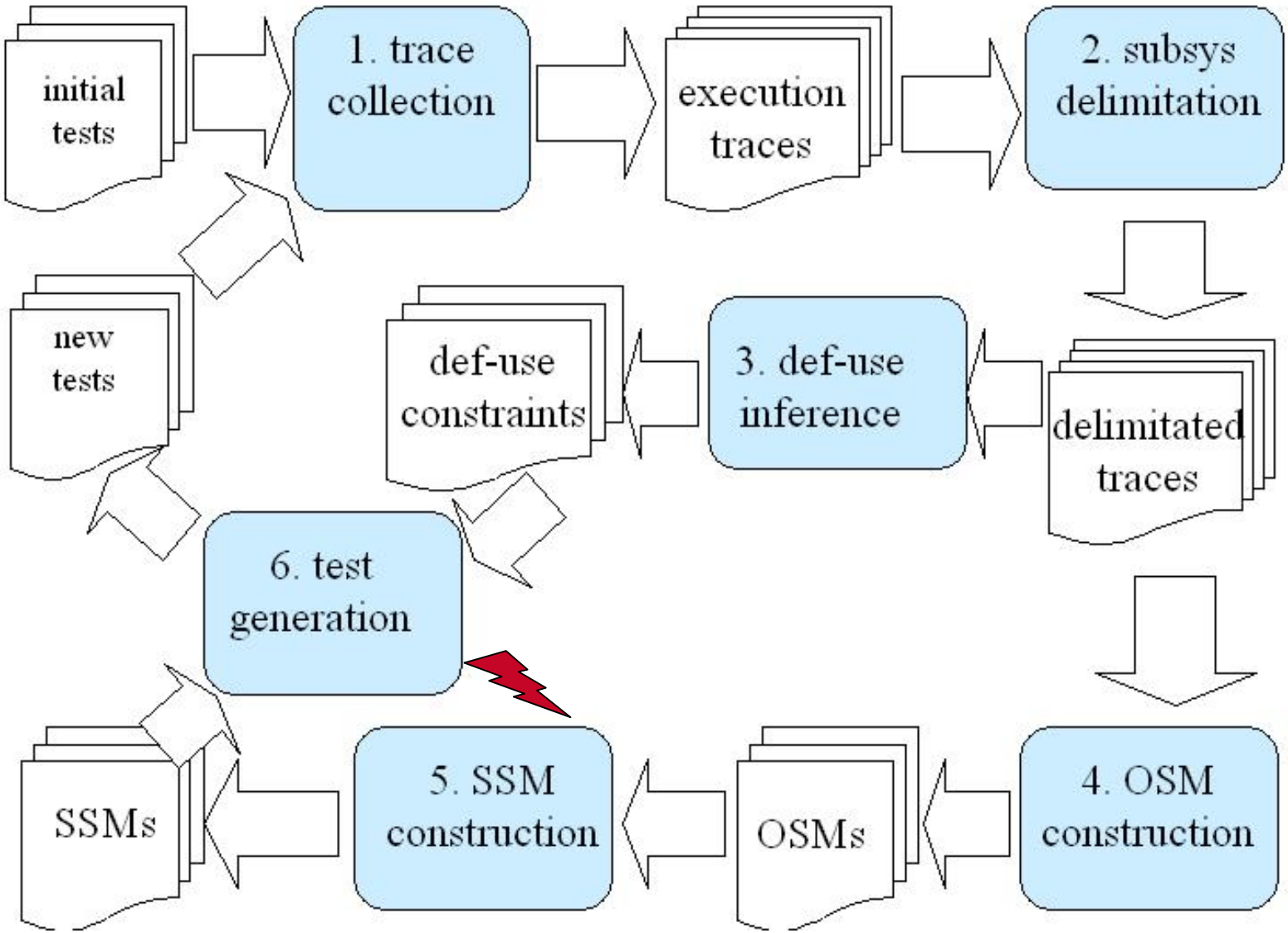


Object State Machine (OSM) Construction

- Inputs: delimited traces
- Outputs: OSMs

- OSM
  - States: receiver object states
  - Transitions: method calls

Substra Framework

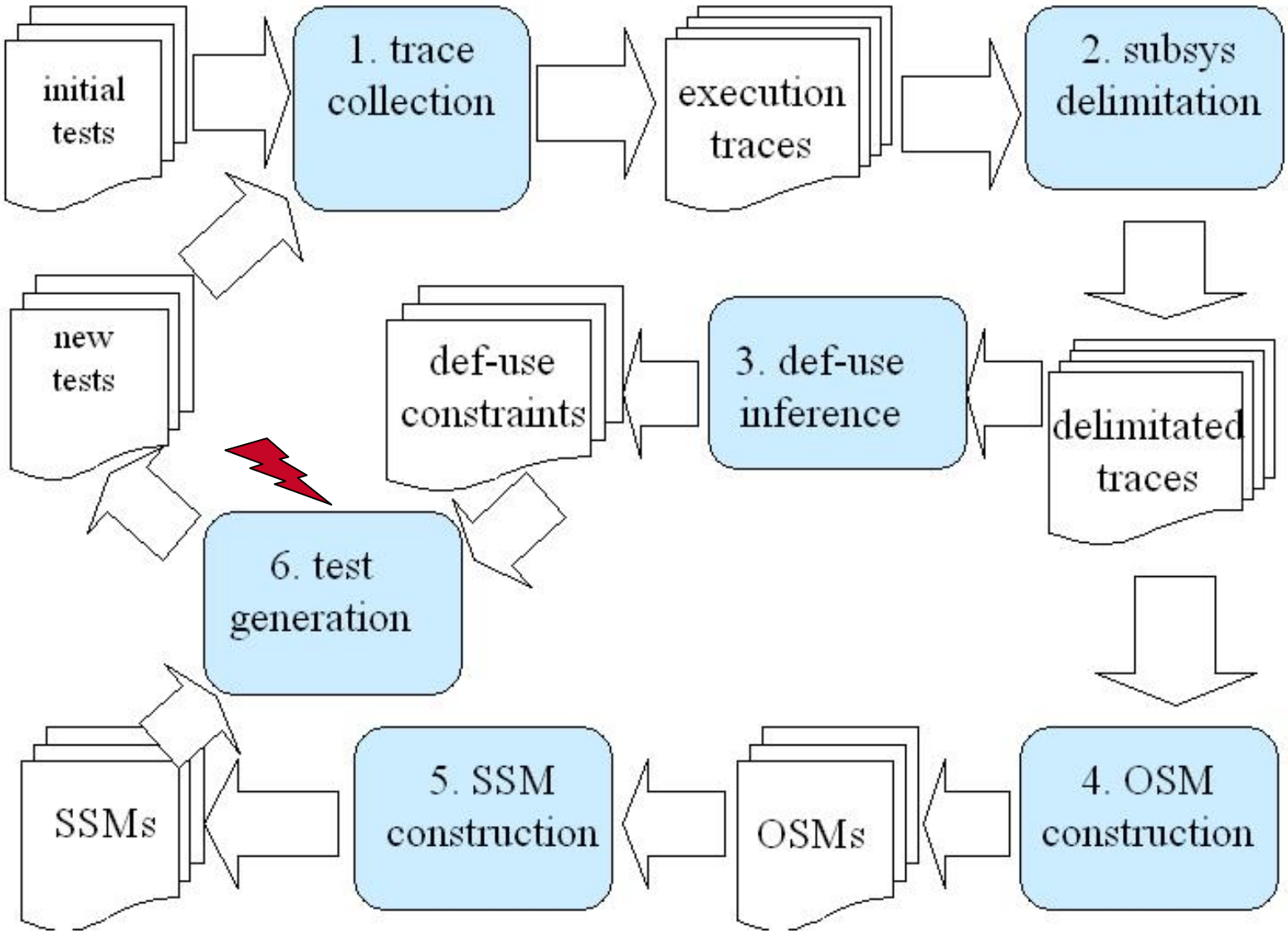


Subsystem-State Machine (SSM) Construction

- Inputs: OSMs
- Outputs: SSMs

- SSM
  - States: subsystem object states
    - Aggregated object states since the program start
    - Exclude temp object (created inside a boundary method call *c* but never referred to after *c*)
  - Transitions: boundary method calls
    - annotated with def-use constraints as guard conditions

Substra Framework



Test Generation

- Inputs: SSMs equipped with def-use constraints
- Outputs: new tests

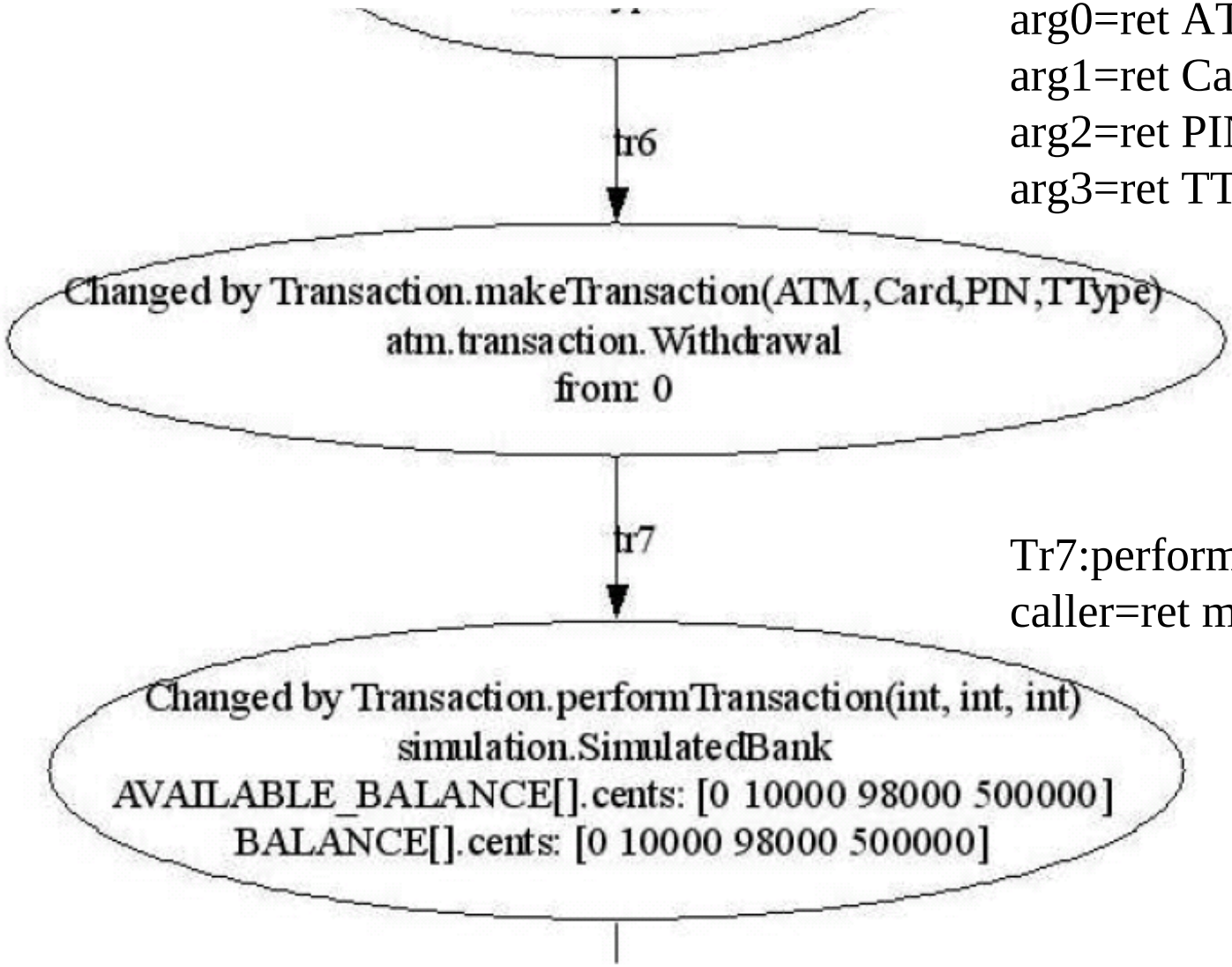
- Abstract primitive arguments in transitions
- Depth-first traverse SSMs to generate method-sequence skeletons
- Use jCUTE [Sen&Agha 06] or random techniques to generate primitive arguments in method-sequence skeletons

Preliminary Results– ATM Example

- Originally developed by Bjork; include 4 packages: banking, simulation, atm.physical, atm.transaction
- Subsystem scope: atm.transaction
- Initial test: withdrawal from an existing account with a correct ATM card number and PIN.

http://courses.knox.edu/cs292/ATMExample/Intro.html

SSM w/ Def-Use – ATM Example



Tr6:makeTrasaction
arg0=ret ATM.ATM(…);
arg1=ret Card.Card(…);
arg2=ret PIN.PIN(…);
arg3=ret TType.TType(…);

Tr7:performTrasaction
caller=ret makeTrasaction;

New Generated Tests – ATM Example

- Random test generation for primitive values (also allow user-defined primitive values optionally)
- New program behavior exercised by new tests
  - withdrawal with incorrect account information
  - deposit with incorrect account information
  - transfer with incorrect account information
  - incorrect transaction types
  - incorrect account types
  - …

Related Work

- Unit-test generation based on inferred specification [Xie&Notkin 03][Pacheco&Ernst 05]
- Infer protocol specifications from program executions [Whaley et al. 02][Ammons et al. 02]
- Reverse engineer UML sequence diagrams statically [Rountev et al. 05] or dynamically [Briand et al. 04]
- Integration test generation from UML diagrams [Ali et al. 05][Basanieri et al. 02]…
- Automatic test factoring [Saff et al. 05] and selective capture&replay [Orso&Kennedy 05]

Conclusion

- Specifications can help integration testing: test-input generation and behavior checking
  - But specifications often don't exist in practice
- Substra infers integration constraints from existing runs
  - Def-use constraints
  - Sequencing constraints
- Generate new tests based on inferred constraints
  - Abstract away primitive method arguments
- New tests can expose new useful program behavior

Questions?