Cenni a test non funzioanle

Roberto Pietrantuono

University of Naples Federico II

Testing for improving reliability

- Analysis Techniques
  - Static Analysis/Formal Methods
- Debug (non-functional) testing
  - **Robustness Testing**
  - **Performance/Stress Testing**
  - **Performance Degradation Testing**
- Other fault avoidance (non-V&V) practices
  - Requirements Engineering
  - Design
  - Coding
  - Maintenance
  - …

Testing non-functional properties

- Examples of relevant NF requirements
  - Availability
  - Security
  - Usability
  - Performance
  - Robustness

Robustness

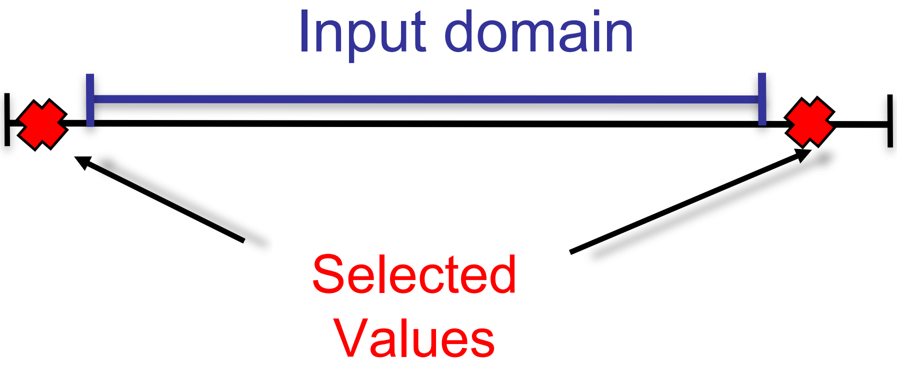
IEEE defines it as:

"The degree to which a system operates correctly in the presence of

- *exceptional inputs or*
- *stressful environmental conditions." [IEEE Std 610.12.1990]*

Simple example: *boundary value testing:*

✦ black box strategy, select out-of-range inputs

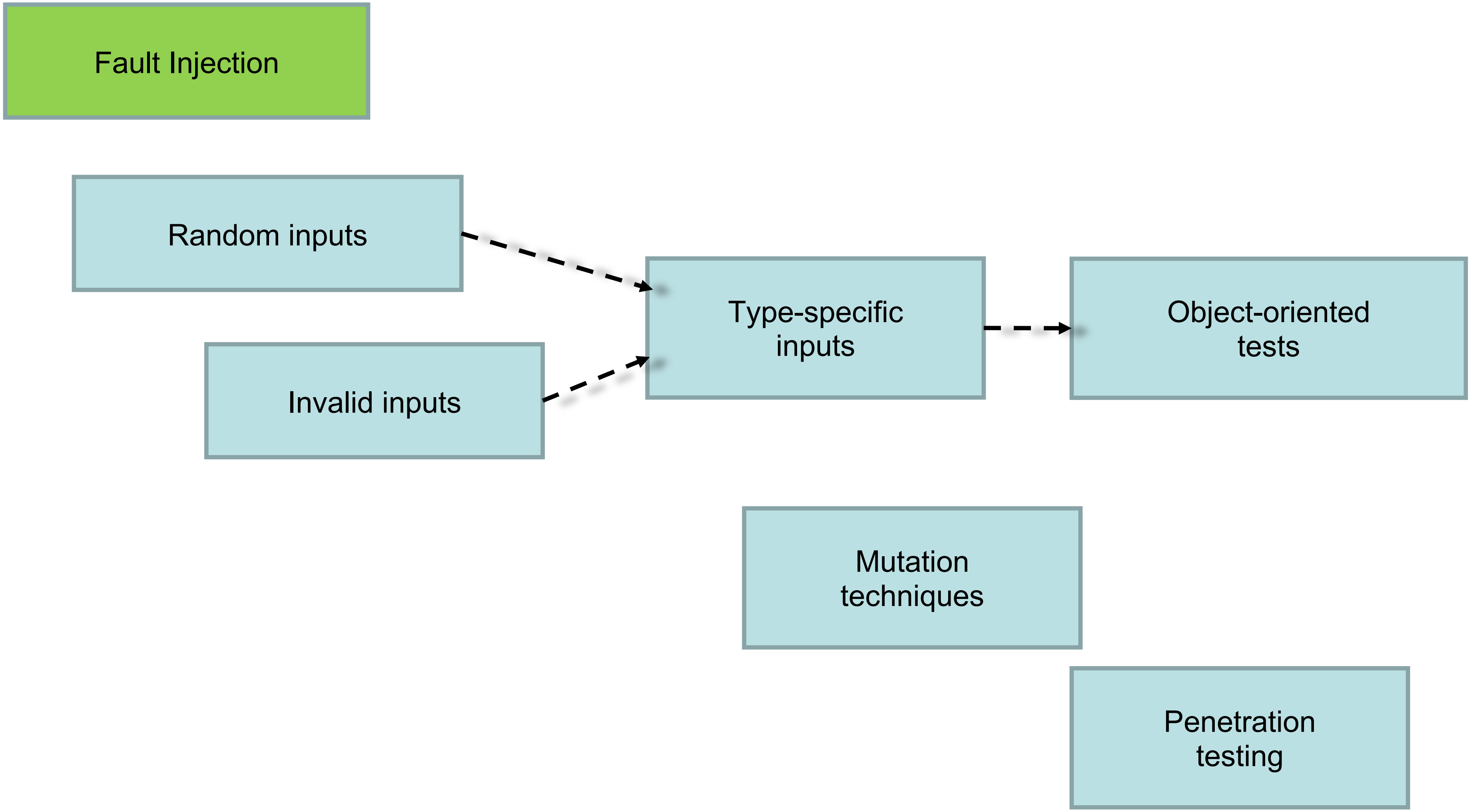


Robustness assessment

The CRASH scale

- **Catastrophic**: the system crashes
- **Restart**: the application needs restarting (e.g., hang)
- **Abort**: abnormal app termination
- **Silent**: invalid operation with no failure/error notification
- **Hindering**: error code returned not valid

Robustness Testing: techniques



Fault Injection

Random inputs

Invalid inputs

Type-specific inputs

Object-oriented tests

Mutation techniques

Penetration testing

Injecting physical faults

- Fault injection consists of the **deliberate insertion of artificial faults** in a computer system or component in order to assess its behavior in the presence of faults

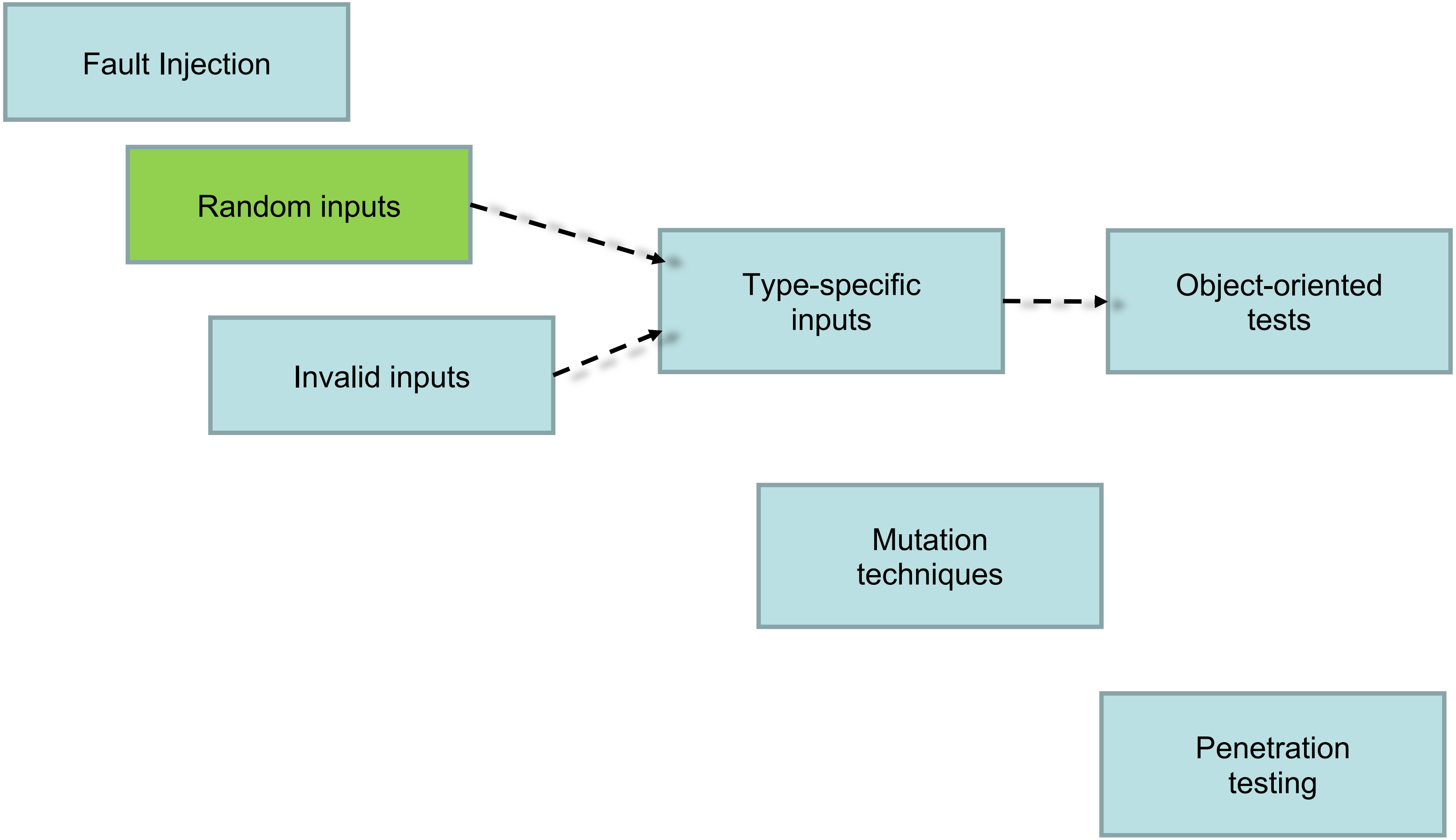
- Several fault models and injection techniques can be adopted, depending on the faults the system is supposed to tolerate
  - Hardware faults (e.g., bit-flip, stuck-at)
  - **Software faults (bugs)**

Injecting physical faults

- Software Fault Injection can be adopted for:
  - Verification of **fault-tolerance** mechanisms
  - Prediction of worst-case scenarios and risk assessment
  - Dependability benchmarking

- SFI techniques:
  - **Source-code mutations**
  - Assembly-code mutations
  - Error injection at internal variables
  - Error injection at component interfaces

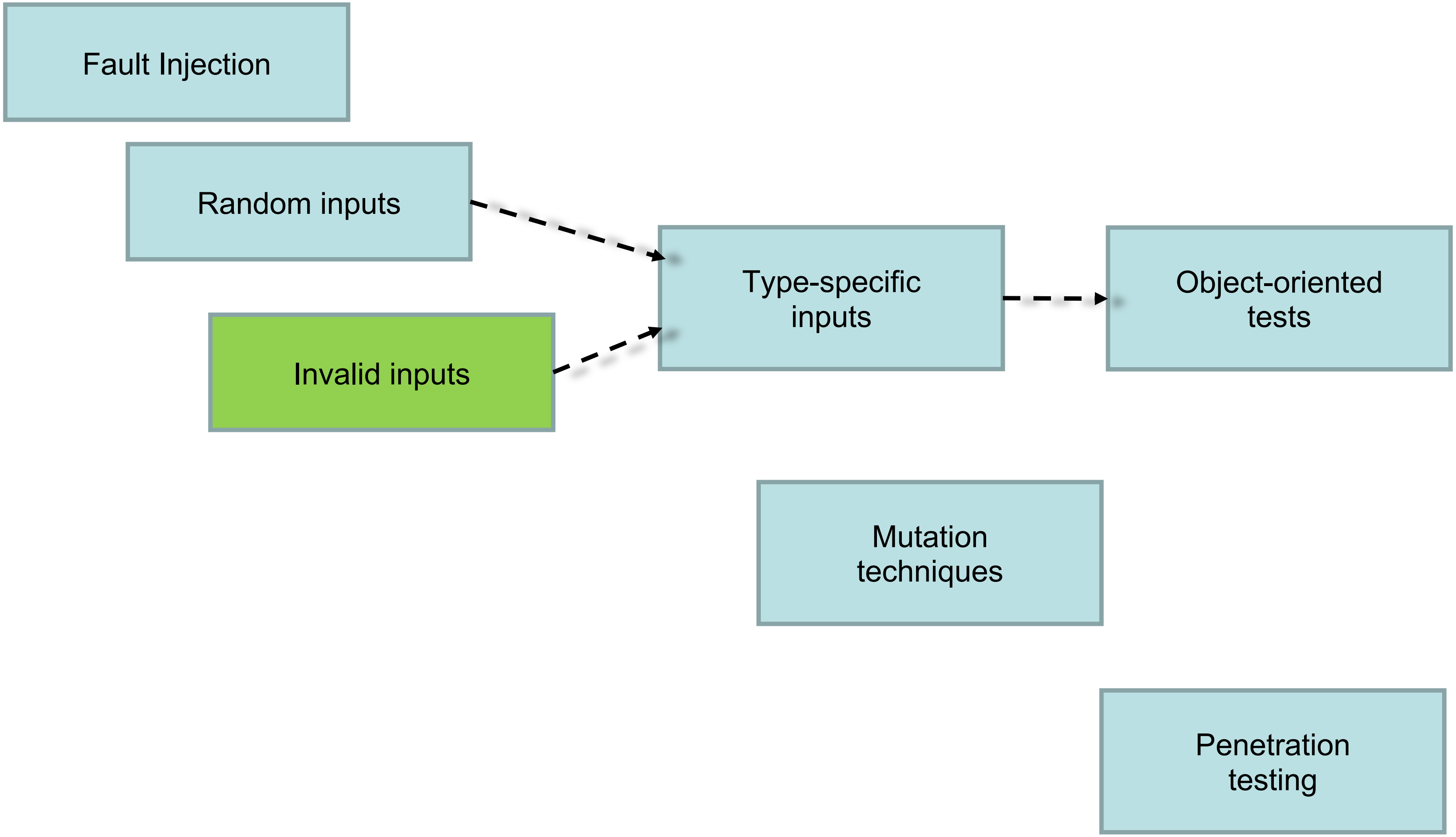
Robustness Testing: techniques



Random inputs

- Simple black box method
  - Use random input
  - Input: command line parameters, GUI messages…
  - Observe CRASH failures
- Although simple technique still useful for current SW
- Referred as „fuzzing”
  - But: fuzzing often means more advanced technique (e.g. random input guided by models)
  - Typical in protocol, file format testing
  - Extensively used in security testing

Robustness Testing: techniques



Fault Injection

Random inputs

Invalid inputs

Type-specific inputs

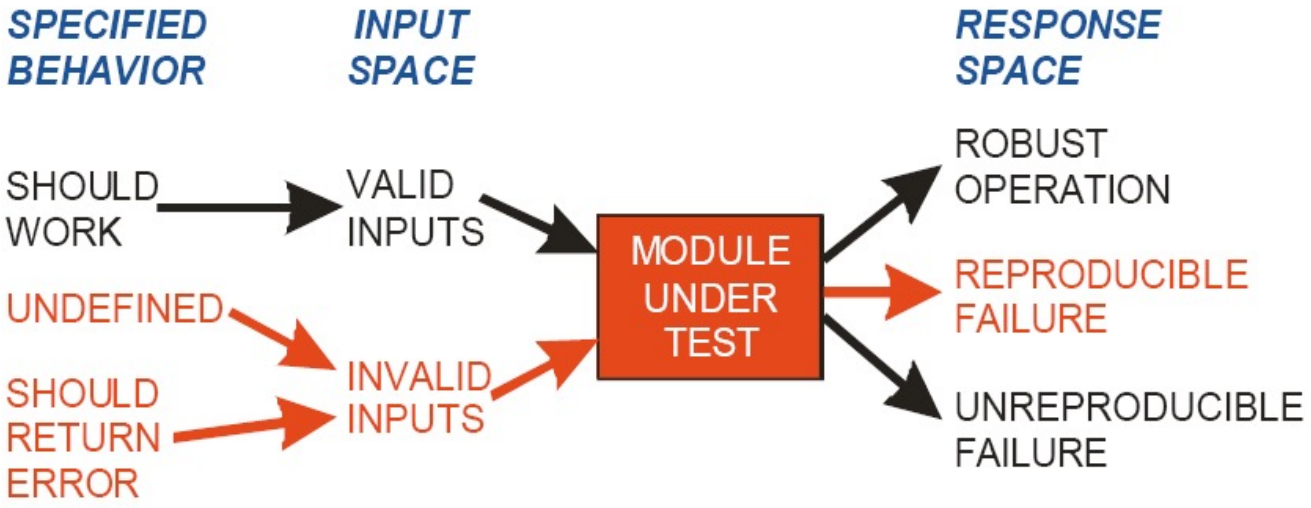
Object-oriented tests

Mutation techniques

Penetration testing

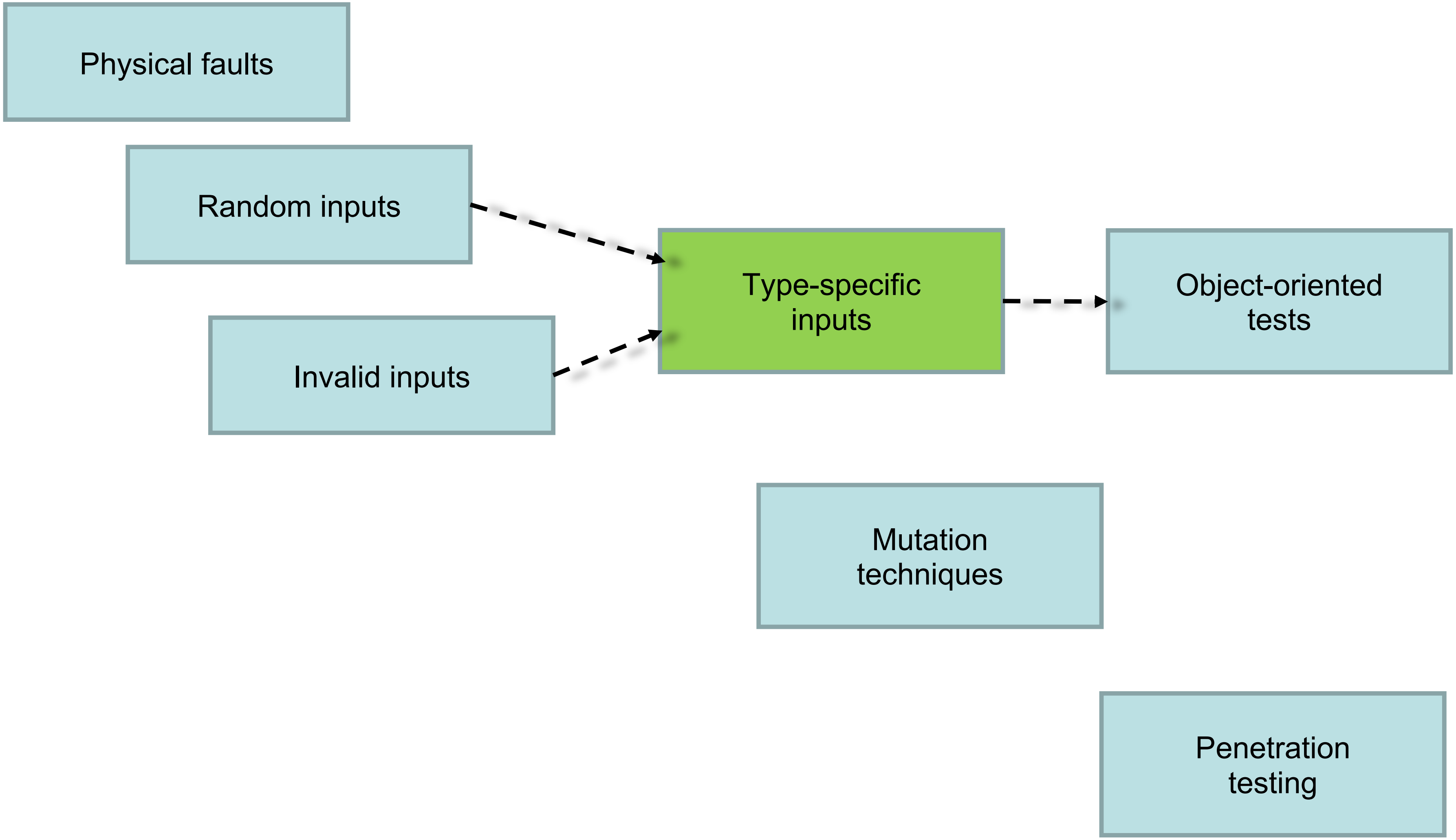
Invalid inputs

- Use the structure of the SUT
  - describe parameter types and numbers
  - e.g. grammar-based definition
- Generate legally structured input
  - filled with illegal values
  - e.g. non-printable charachters, negative numbers, long strings



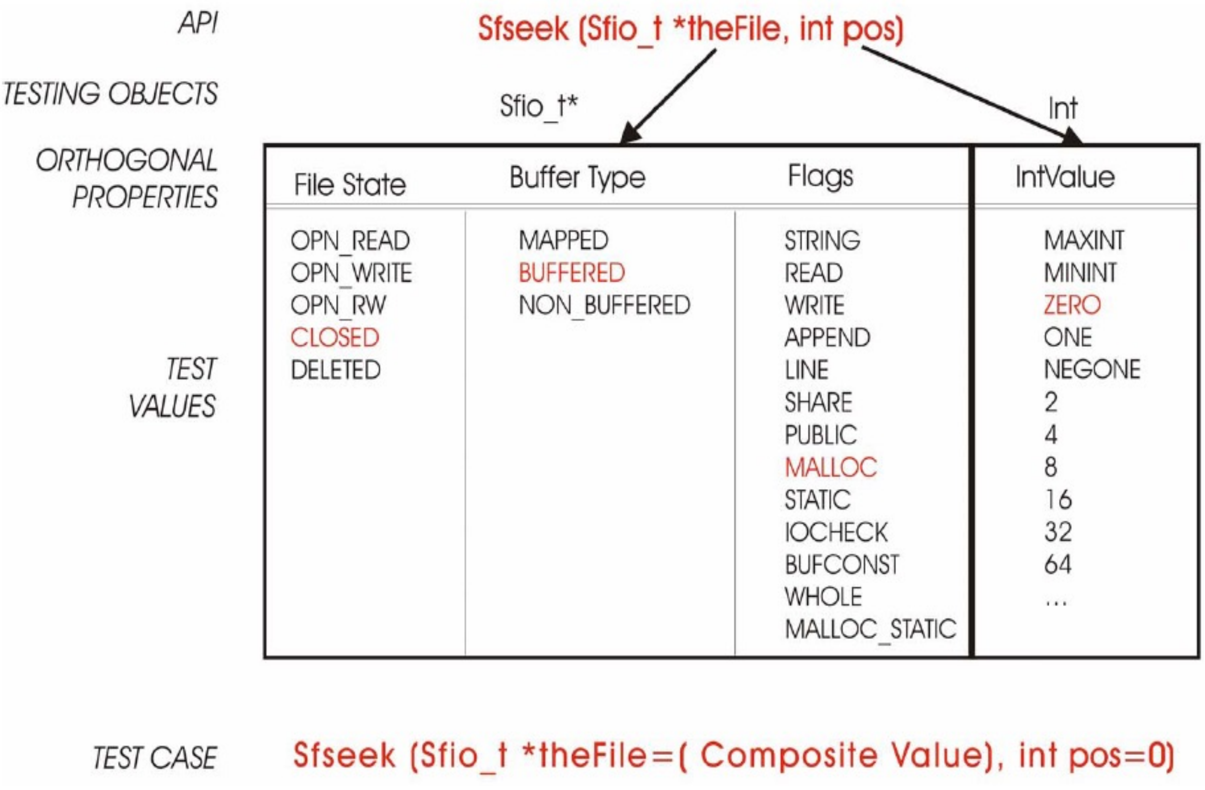
P. Koopman, J. DeVale: “The Exception Handling Effectiveness of POSIX Operating Systems”, IEEE Tran. on Software Engineering, Vol. 26, No. 9, Sept. 2000.

Robustness Testing: techniques



Type-specific tests

- Define the valid and invalid values for **parameter types**



API Sfseek (Sfio_t *theFile, int pos)

TESTING OBJECTS

ORTHOGONAL PROPERTIES

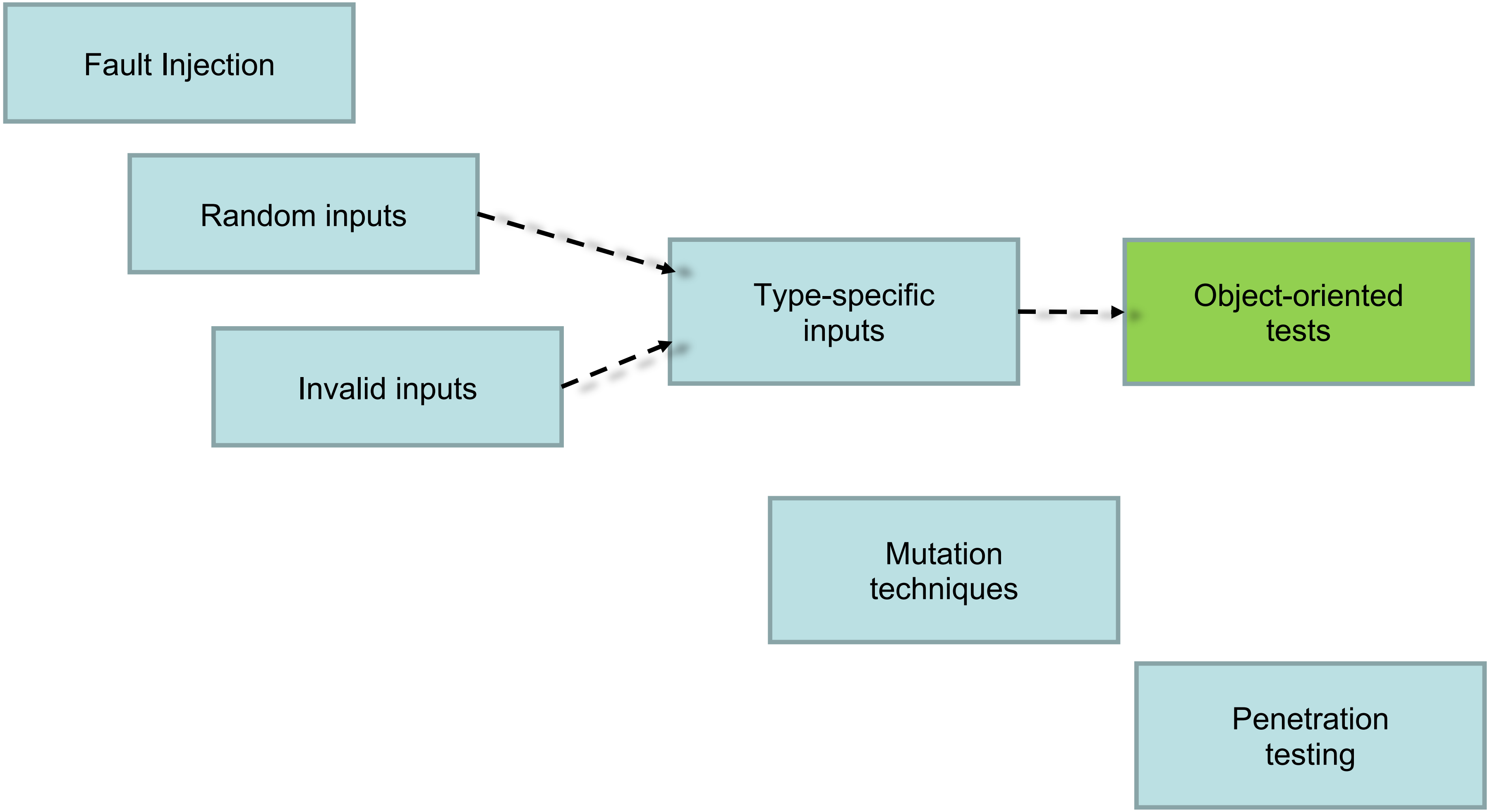
TEST VALUES

Sfio_t*			Int
File State	Buffer Type	Flags	IntValue
OPN_READ	MAPPED	STRING	MAXINT
OPN_WRITE	BUFFERED	READ	MININT
OPN_RW	NON_BUFFERED	WRITE	ZERO
CLOSED		APPEND	ONE
DELETED		LINE	NEGONE
		SHARE	2
		PUBLIC	4
		MALLOC	8
		STATIC	16
		IOCHECK	32
		BUFCONST	64
		WHOLE	...
		MALLOC_STATIC	

TEST CASE Sfseek (Sfio_t *theFile=(Composite Value), int pos=0)

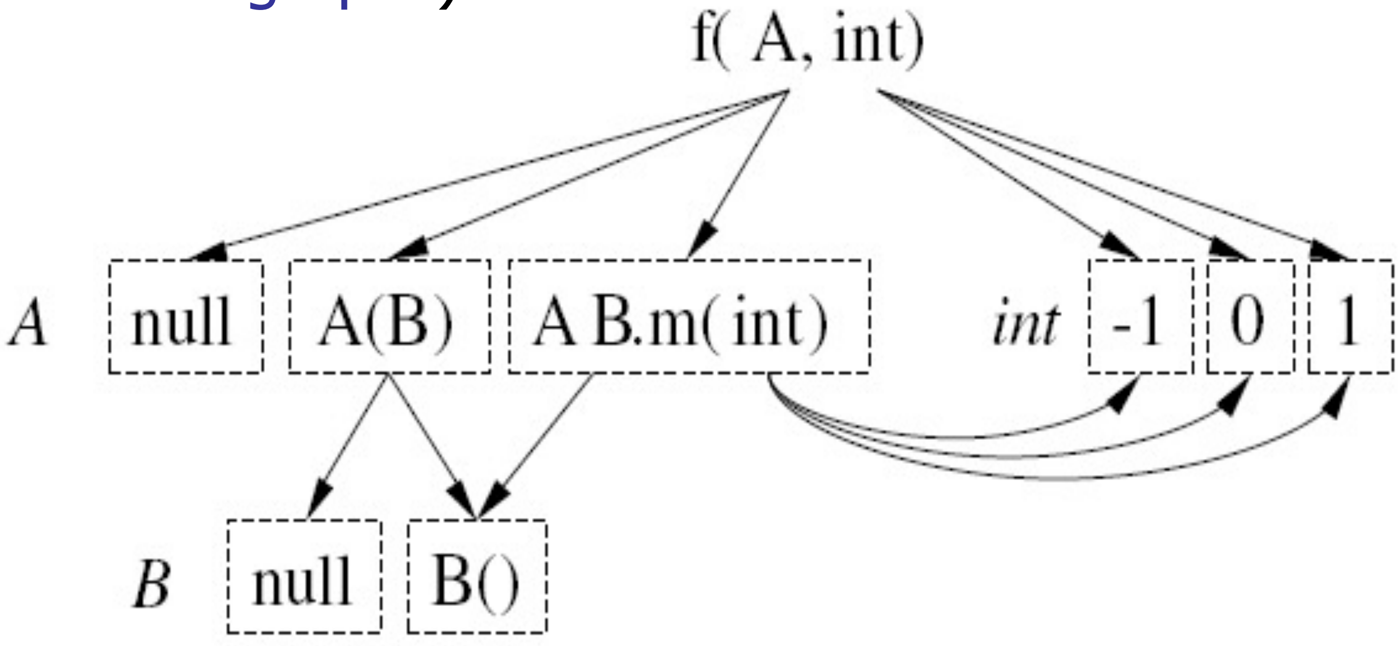
J. DeVale, High Performance Robust Computer Systems, PhD thesis, CMU, 2001

Robustness Testing: techniques



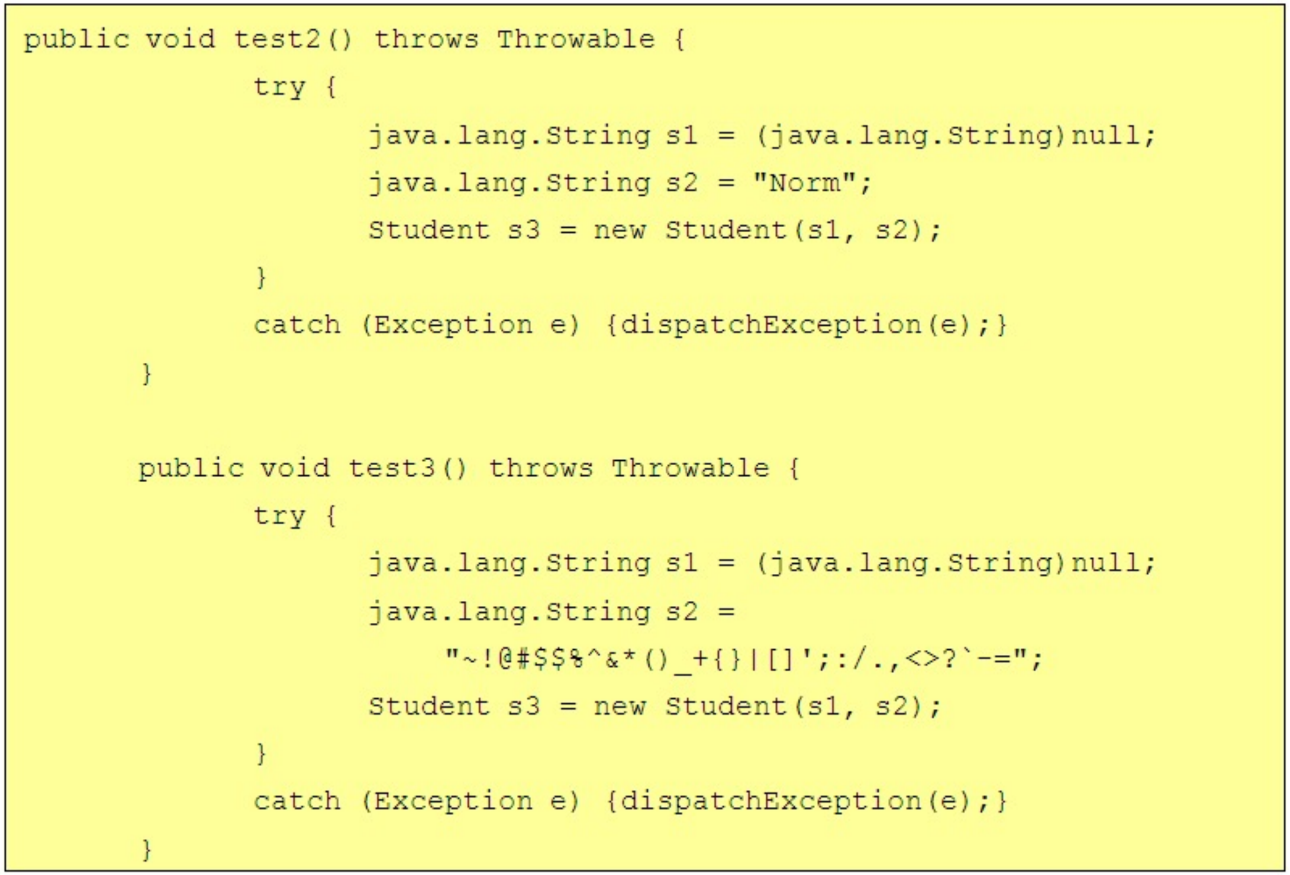
Robustness Testing: OO tests

- Extension of type-specific tests
- Create parameters using discovery of type structures (parameter graphs)



Robustness Testing: OO tests

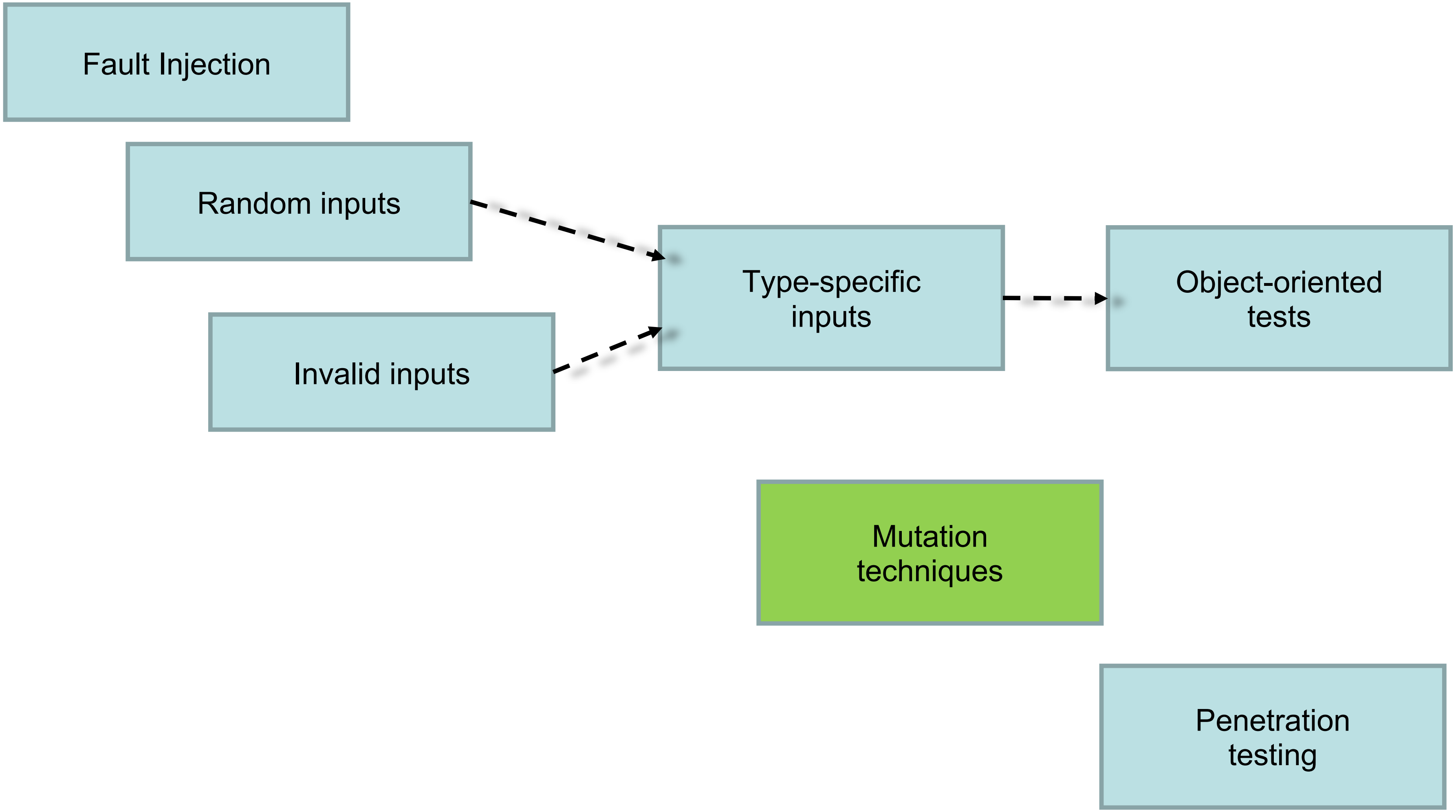
- JCrasher tool: automatic robustness tester for Java
  - Can be integrated into Eclipse
- Produces JUnit test cases



```
public void test2() throws Throwable {
          try {
               java.lang.String s1 = (java.lang.String)null;
               java.lang.String s2 = "Norm";
               Student s3 = new Student(s1, s2);
          }
          catch (Exception e) {dispatchException(e);}
     }

     public void test3() throws Throwable {
          try {
               java.lang.String s1 = (java.lang.String)null;
               java.lang.String s2 =
                   "~!@#$$%^&*()_+{}|[]';:/.,<>?`-=";
               Student s3 = new Student(s1, s2);
          }
          catch (Exception e) {dispatchException(e);}
     }
```

Robustness Testing: techniques



Fault Injection

Random inputs

Invalid inputs

Type-specific inputs

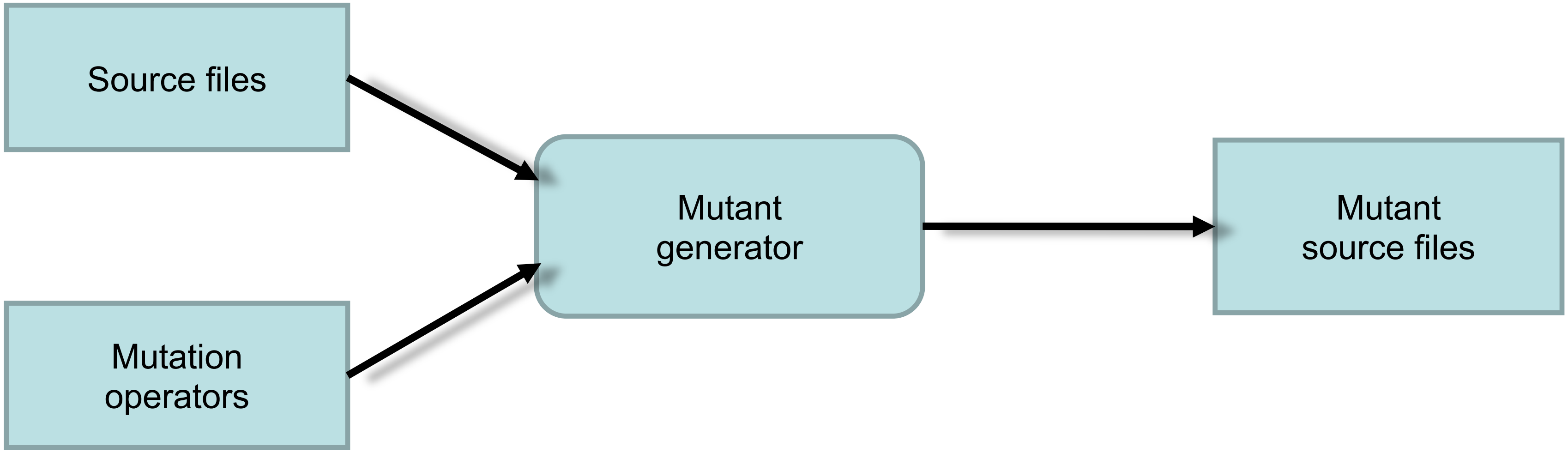
Object-oriented tests

Mutation techniques

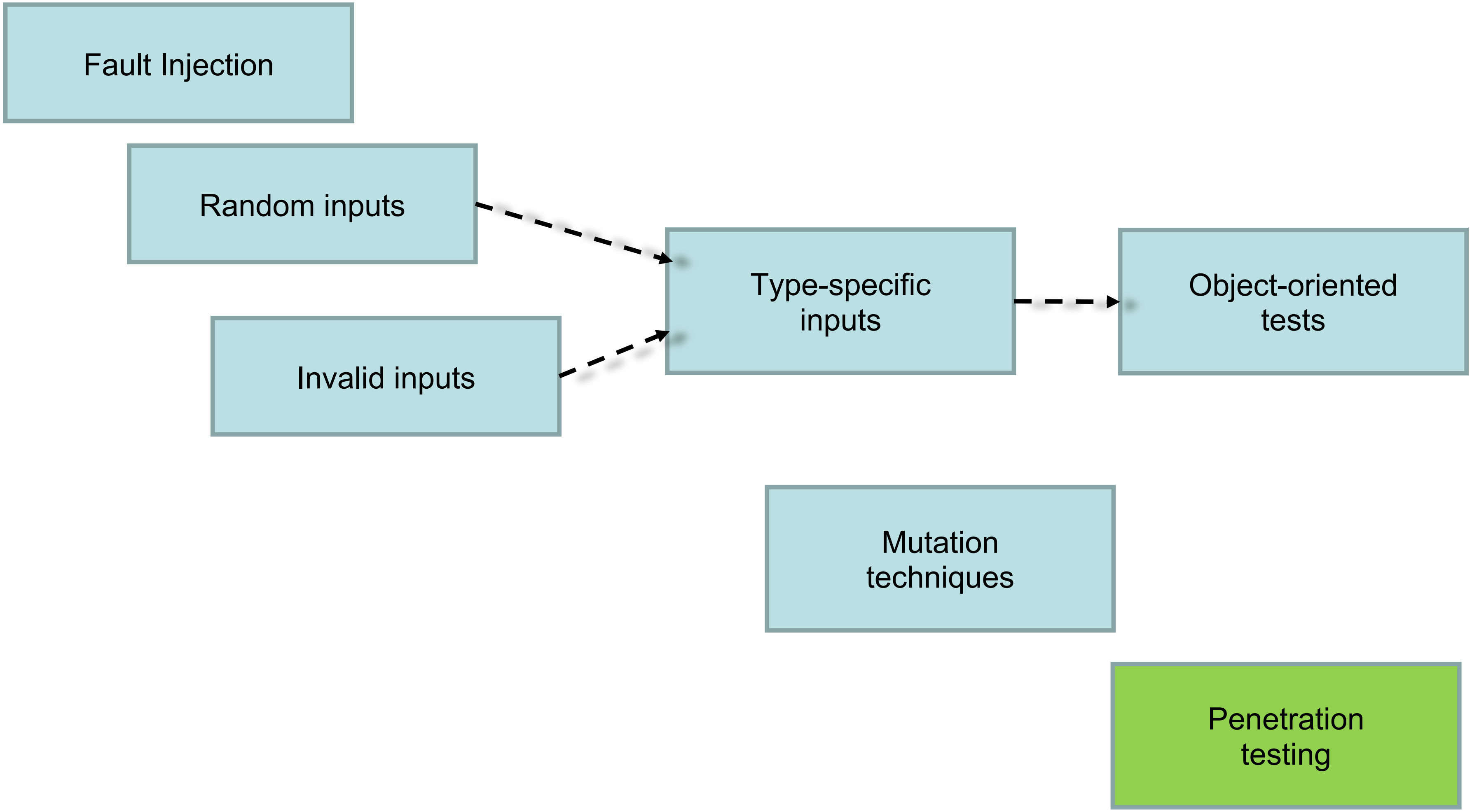
Penetration testing

Mutation testing

- Code mutation techniques
- Start from a valid code (e.g. functional test case)
  - use mutation operators resembling typical faults
  - Operators: omit call, interchange calls, replace normal values…



Robustness Testing: techniques



Fault Injection

Random inputs

Invalid inputs

Type-specific inputs

Object-oriented tests

Mutation techniques

Penetration testing

Penetration testing

- Proactive, authorized attempt to compromise security
- Active analysis
- Black/white/grey box techniques depending on knowledge of the system
- Robustness problems may open vulnerabilities
  (e.g. buffer overflow)

Performance and stress testing

- Techniques aimed at detecting failures in meeting performance requirements
  - E.g., response time, resource usage

- Performance assessment: to characterize performance under expected workload
- Stress testing:
  - Used to denote both a robustness testing technique ("assess the system under *stressful conditions*"),
  - And as performance testing, to assess performance under stressful conditions

Performance degradation testing

- **Perf. Degradation:** A phenomenon exhibited by long running applications that suffer for an increasing failure rate and/or decreased performance over time
  - due to error accumulation/activation/propagation influenced by total running time
  - e.g., memory leaks, fragmentation, round-off errors, not-terminated threads
  - caused by so-called aging-related bugs (ARB)

  - ***Software aging*** *refers to phenomena of gradual progressive degradation and resource consumption*

- **Perf. Degradation tests are long-running stress tests that try to expose aging problems at testing time**