Testing of Large Scale Distributed Software

Gerson Sunyé
Lina - Université de Nantes
http://sunye.free.fr

Agenda

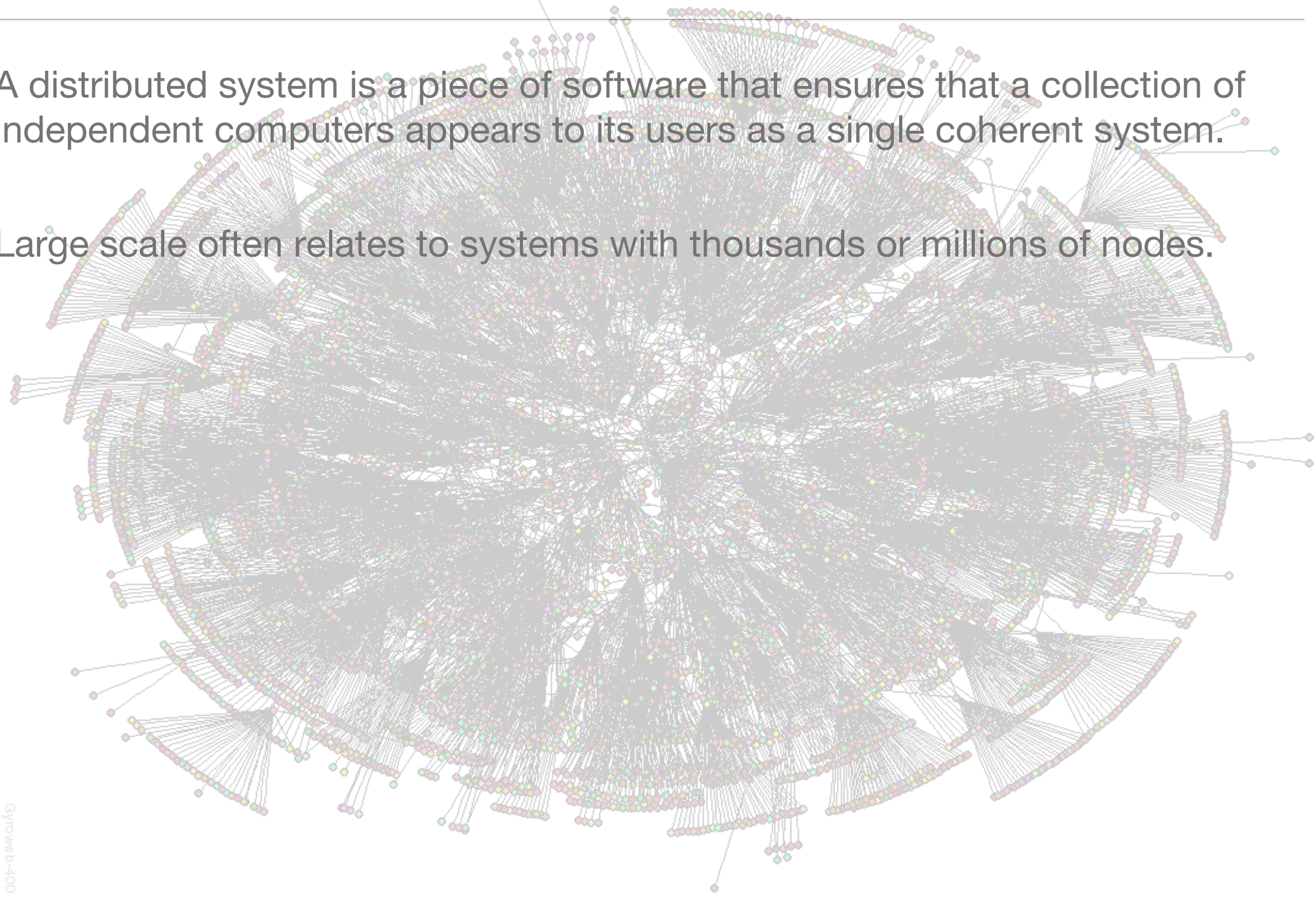
- Large-Scale Distributed Systems.
- Characteristics of Distributed Software
- Testing of Distributed Systems
- Web Applications and Web Services.
- Peer-to-Peer Systems
- A Framework for Testing Distributed Systems
- Conclusion

Large-Scale Distributed Systems

Large-Scale Distributed (LSD) Systems

- A distributed system is a piece of software that ensures that a collection of independent computers appears to its users as a single coherent system.

- Large scale often relates to systems with thousands or millions of nodes.

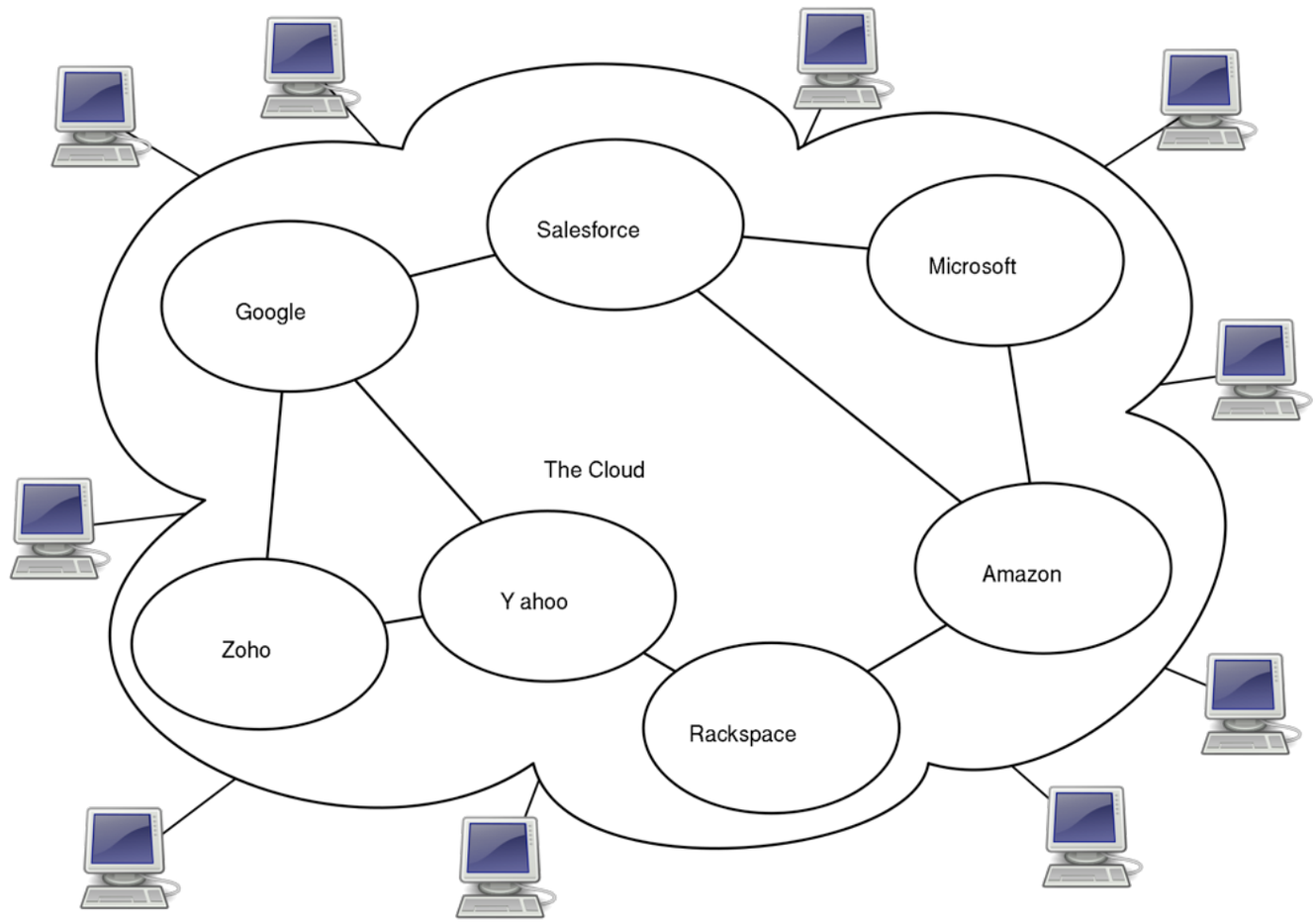


LSD Systems Examples

- Peer-to-Peer Systems: Gnutella, Napster, Skype, Joost, Distributed Hash Tables, etc.
- Cloud Computing: GoogleDocs, ThinkFree Online, etc.
- Grid Computing: Berkeley Open Infrastructure for Network Computing (BOINC), etc.
- MapReduce: Hadoop, Google, Oracle, etc.

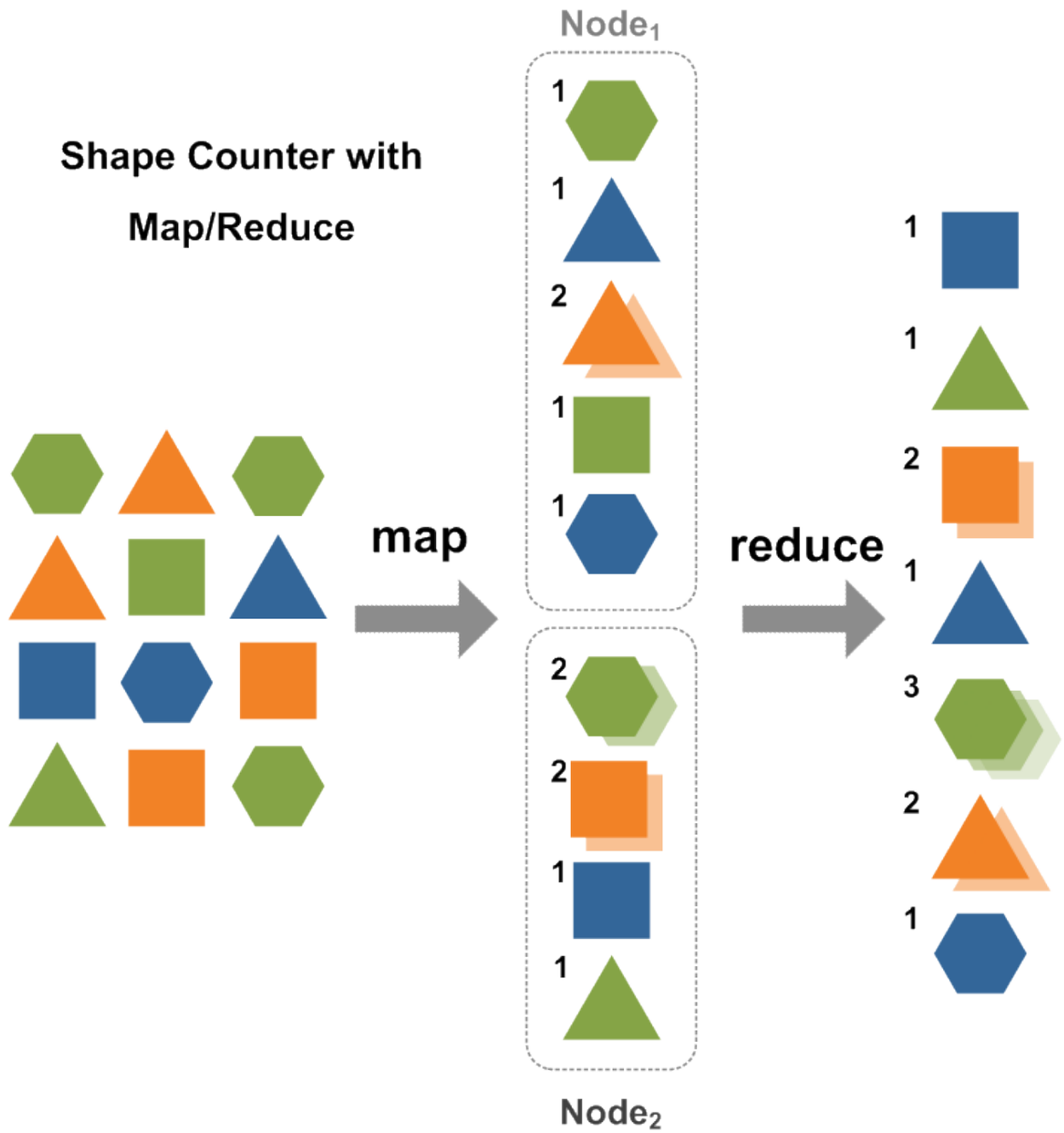
Cloud Computing

- Main concept: the computing is "in the cloud".
- Cloud service providers: Amazon, Microsoft, Google.
- Infrastructure as a service.
- Software as a service.
- Examples: Eucalyptus, OpenNebula, etc.



MapReduce

- Framework introduced by Google.
- Supports distributed computing on large data sets on clusters of computers.
- Exemples: Apache Hadoop, Greenplum’s, Aster Data’s, etc.
- Users: Google Index, Facebook,...



Distributed Hash Table (DHT)

- Second generation P2P overlay network (Structured).
- Self-organizing, load balanced, fault tolerant.
- Scalable: guarantees the number of hops to answer a query.
- Simple interface:
    - store(key, value)
    - retrieve(key) : value

DHT Principles

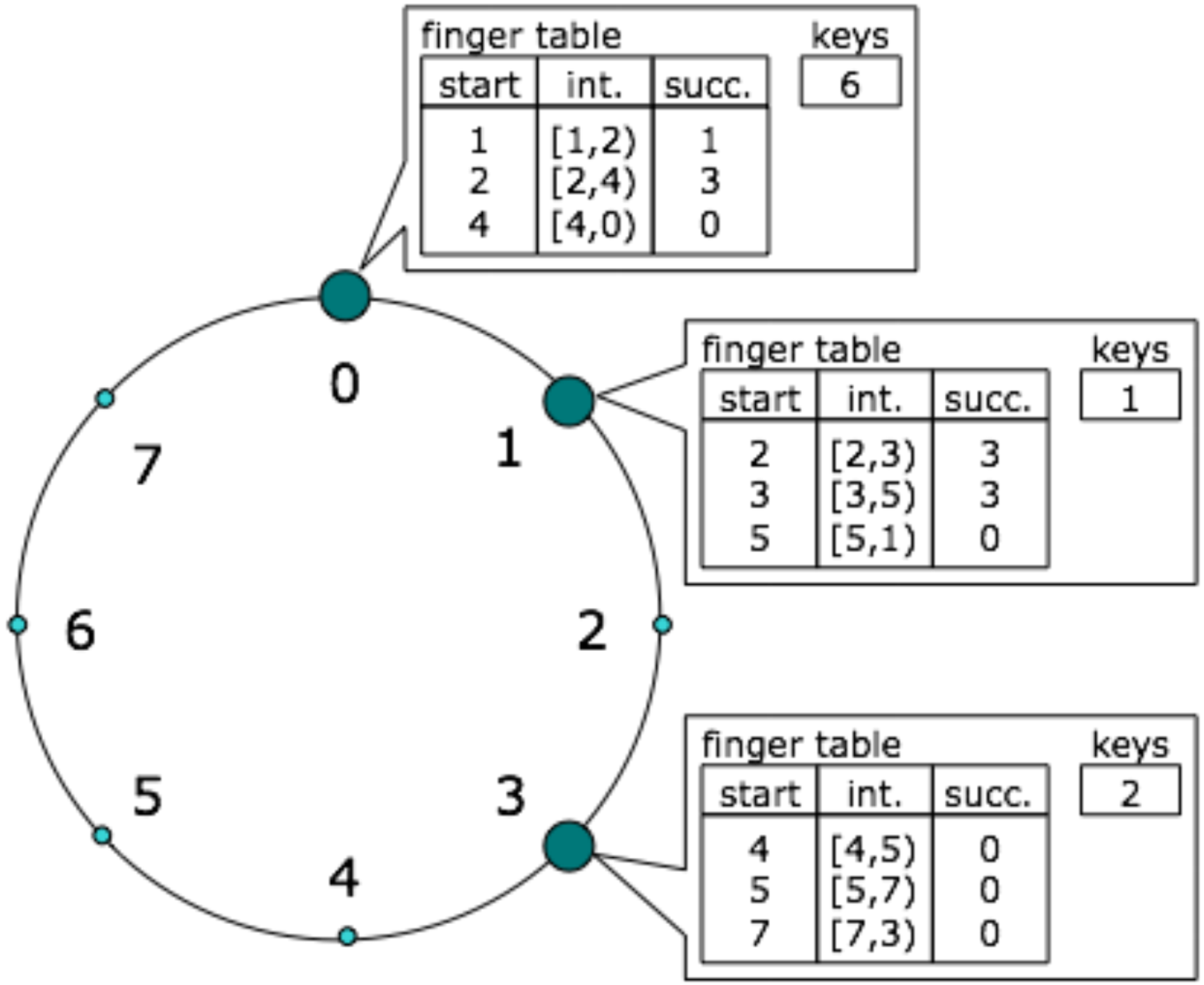
- Each node is responsible for a range of keys.

- Core operation: find the node responsible for a given key.

DHT

- Node X Id association:
    - 1 node knows all other nodes (centralized).
    - All nodes know all other nodes (replication).
    - All nodes have a partial view of the whole system.

DHT Example: Chord



Why do we need DHTs ?

- Abstraction:
    - Persistence, volatility, message routing, etc.
- Applications:
    - File systems.
    - Data sharing (Gnutella index).

Characteristics of Distributed Software

How is distributed software different?

- Non-determinism;
- Third-party infrastructure;
- Partial failures;
- Time-outs;
- Dynamic nature of the structure.

Non-determinism

- Difficult to reproduce a test execution.
- The thread execution order may be affected by external programs.
- Some defects do not appear on all executions.

Third-party infrastructure

- Distributed software often rely on third-party middleware: RPC, Message Exchanging, Brokers, SOA, etc.

- Infrastructure may be self-organizing.

- Infrastructure (services) may belong to other entities.

Partial Failures

- A failure in a particular node prevents part of the system from achieving its behavior.

Time-Outs

- Time-outs are used to avoid deadlocks.
- When a response is not received within a certain time, the request is aborted.
- This is not an error.

Dynamic nature of the structure.

- Often, the architecture changes.

- Specific requests may be redirected dynamically to different nodes, depending on the load on various machines

- The number of participating nodes may vary.

Distributed Software Testing

How is Distributed Testing Different?

- Test harness deployment.
- Results retrieval.
- Execution synchronization.
- Node failure simulation.
- Number of nodes variation.

How is Distributed Testing Different?

- Level: System Testing.

- Growing need of non-functional testing: security, load, stress.

Test harness deployment

- Test harness must be deployed to every node that composes the SUT.
  - Input data.
  - Test cases.
  - Software under test.
  - Libraries.

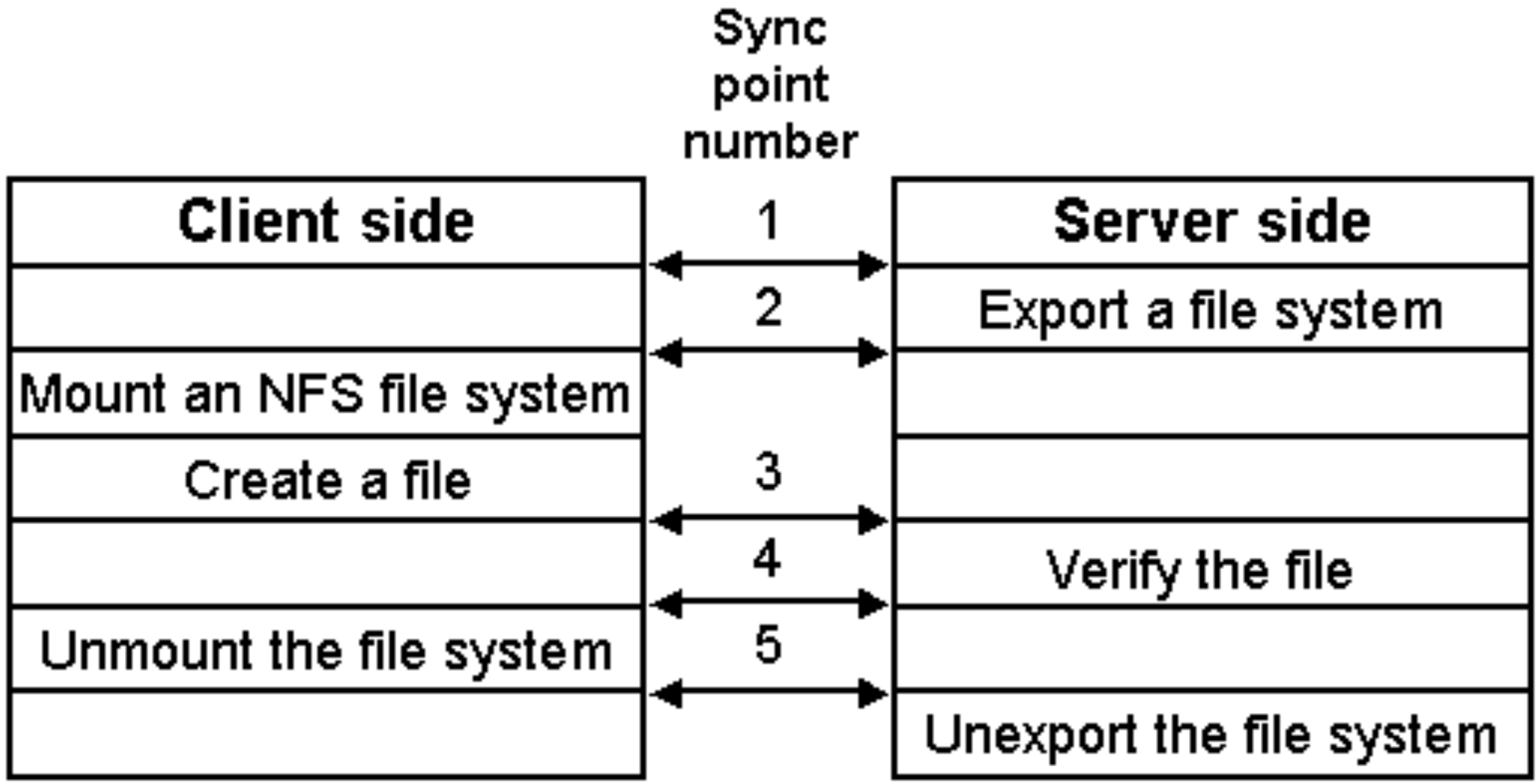
Results retrieval

- All results must be retrieved.

- A timeline must be constructed.

Synchronization Service

- The test architecture must ensure synchronization among nodes.

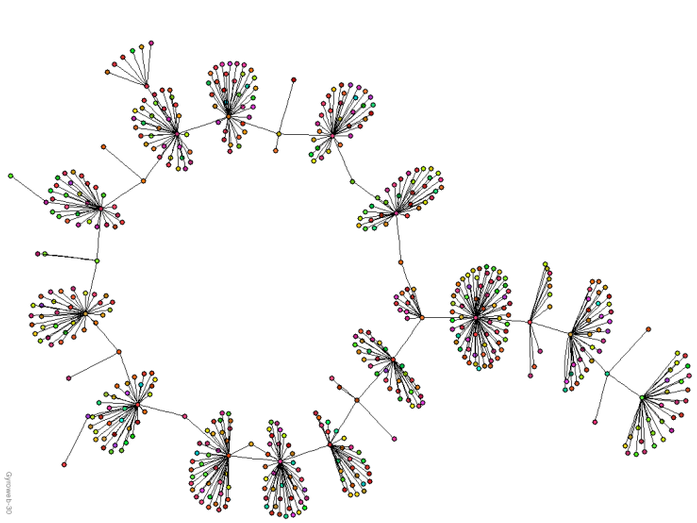


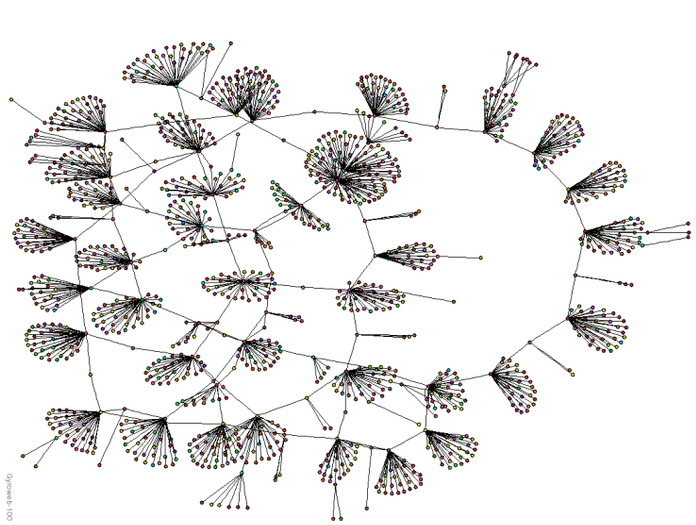
Node Failure Simulation

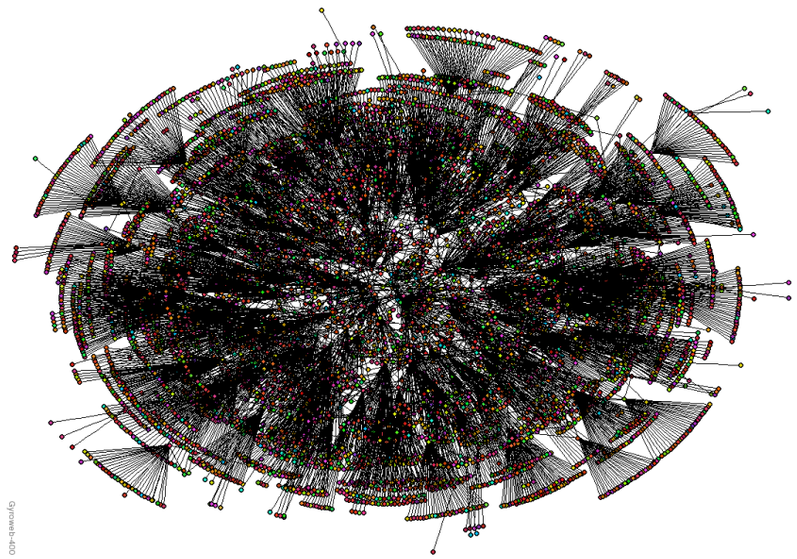
- Removing network connections.
- Shutting down a node.
- Blocking all threads of a node.
- Testing must ensure that software behaves correctly when time-outs are reached.

Number of Nodes Variation

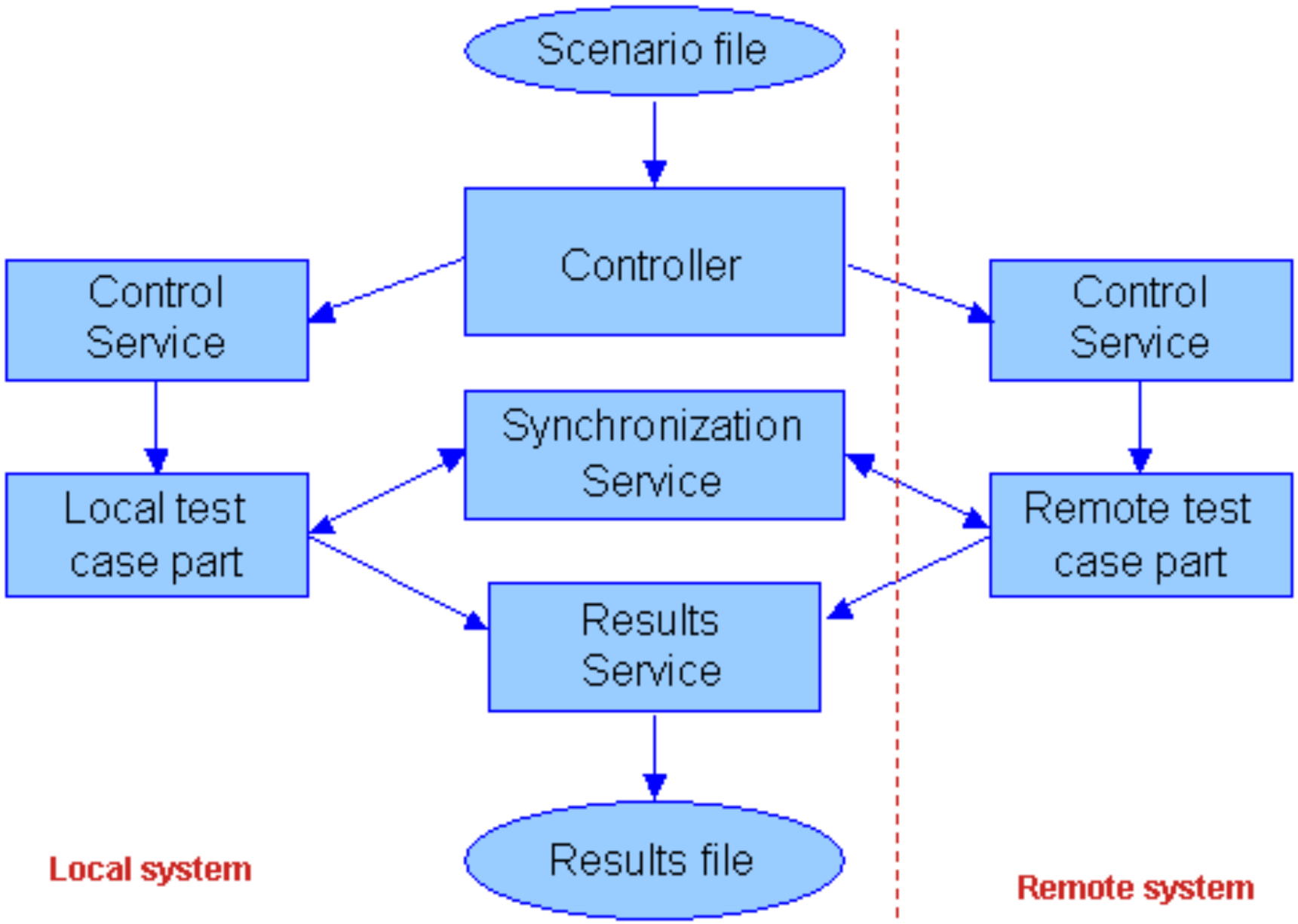
- Tests must be executed on different configurations.



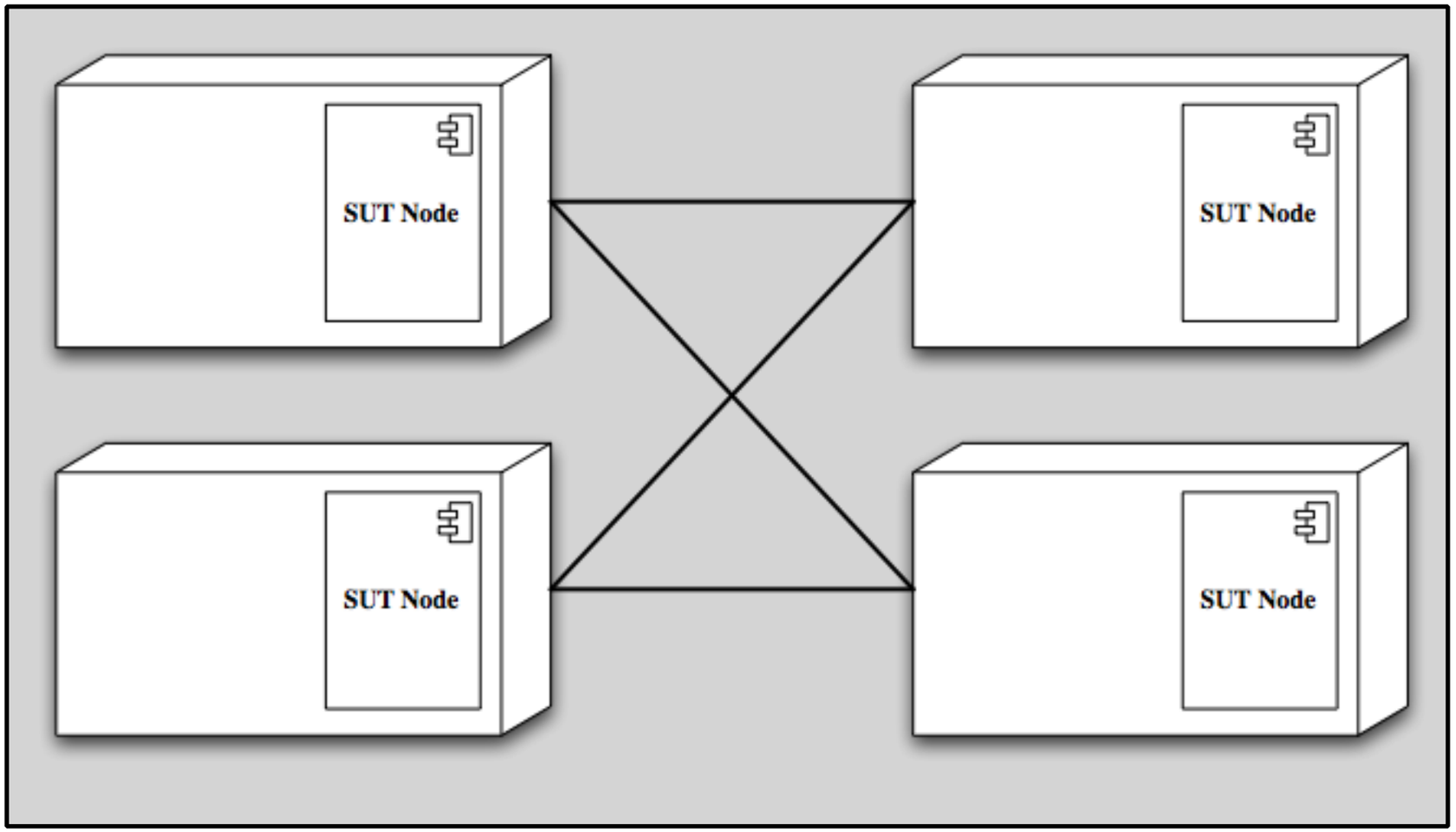




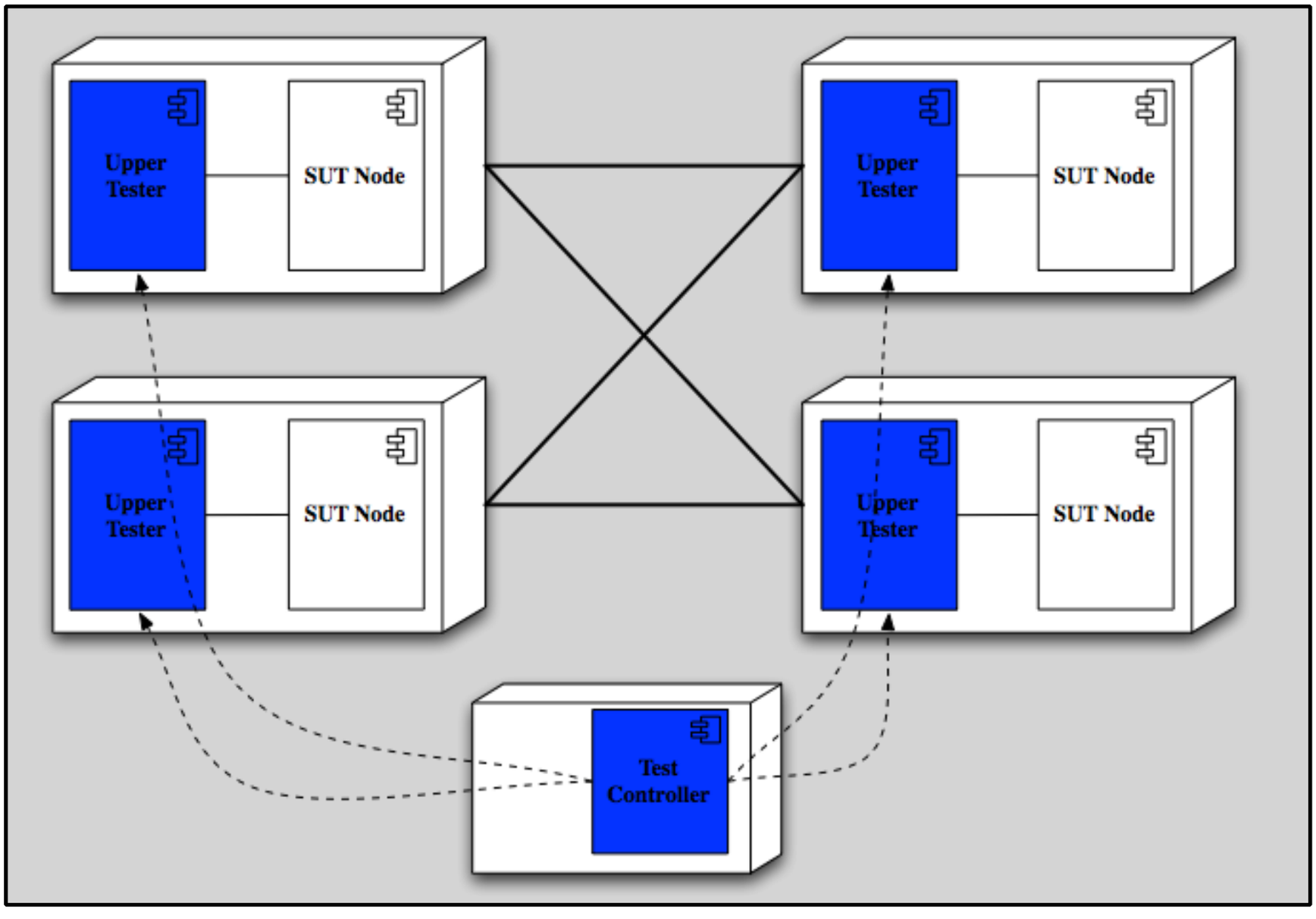
Distributed Test Architecture



A Distributed System



Distributed Test Architecture

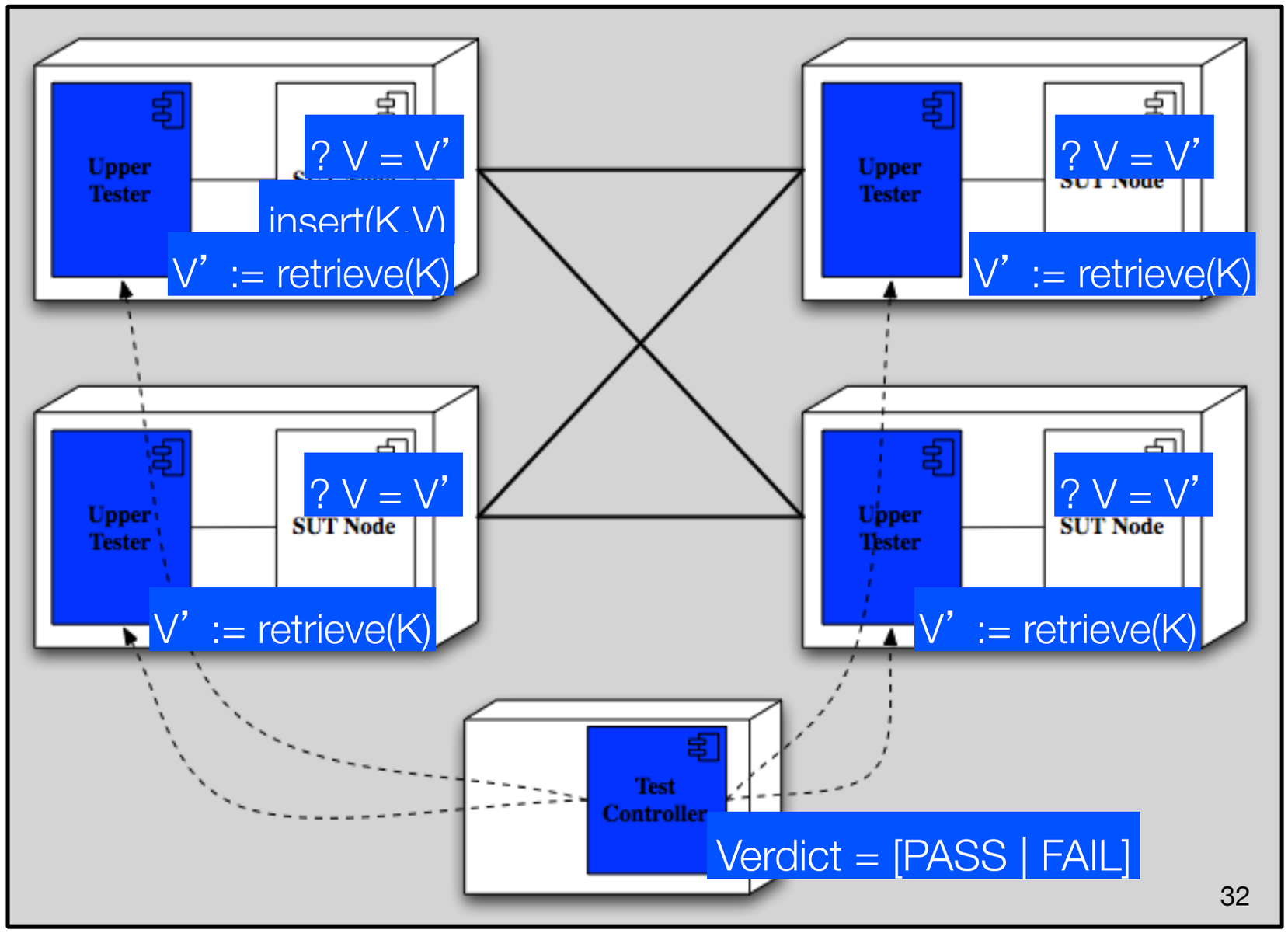


A Distributed Test Case

Step	Action	Nodes
1	insert(K,V)	1
2	V’ := retrieve(K)	all
3	? V = V’	all

Test Case Execution

Step	Action	Nodes
1	insert(K,V)	1
2	V’ := retrieve(K)	all
3	? V = V’	all



Sample Defects

- Link failure between two nodes.
- A single node failure.
- Wrong shared data access.

Web Software

Web Software

- Composed of two or more components:
  - Loosely coupled.
  - Communicate by messages.
  - Almost no shared memory.
- Concurrent and distributed.
- Technological jungle.

Web Applications

- A software deployed on the web.
- HTML User Interface.
- International (available worldwide).
- HTTP as underlying protocol.

How is Web Application Testing Different?

- Traditional graphs (control flow, call) do not apply.
- State behavior is hard to model and to describe (HTTP is stateless).
- All inputs go through the HTML UI – low controllability.
- Stress testing is crucial.

Common difficulties

- Stateless protocol: requests are independent from each other.

- Different hardware devices, programming languages, versions, etc.

New Problems

- The user interface is created dynamically.
- The flow of control may be changed:
  - back/forward, refresh
  - HTML may be rewritten!
- Input validation.
- Dynamic integration: modules can be loaded dynamically.

- Hard to get access to server-side state (memory, files, database) – low observability
- Not clear what logic predicates can be effectively used
- No model for mutation operators on web software

Web Services

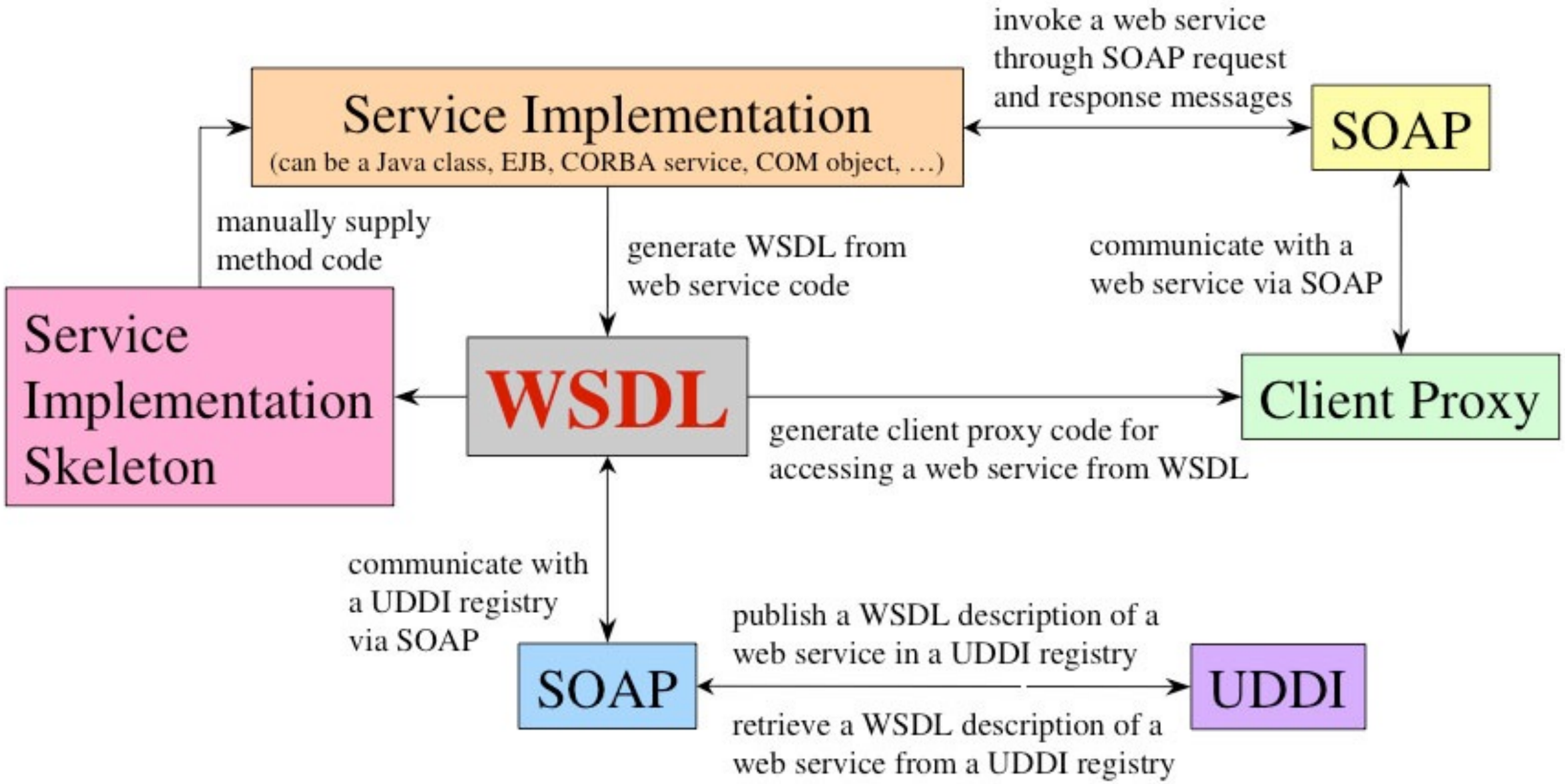
Web Services

- A software deployed on the web that accepts remote procedure call (RPC): SOAP, XML-RPC, RESTful.

- No human interface.

- Must be published to be discovered.

Service Publication



How is Web Services Testing Different?

- Communications are often between services published by different organizations.
- The owner of a service can not control the external services it uses.
- Reconfiguration: a service can be replace by another one.

Peer-to-Peer Systems

P2P Applications

- Huge number of nodes.
- No central point of failure.
- Node volatility is a common behavior.
- Nodes are autonomous.

How is P2P Testing Different?

- Volatility must be simulated and controlled.
- Centralized services are a bottleneck:
  - Synchronization.
  - Results retrieval.

Tools

Test Tools

- SeleniumHQ
- HttpUnit
- Apache JMeter
- PeerUnit

SeleniumHQ

- Selenium IDE: Firefox plugin
- Selenium Remote Control: test runner
- Selenium Grid: RC extension for parallel testing.

HttpUnit

- Server testing.

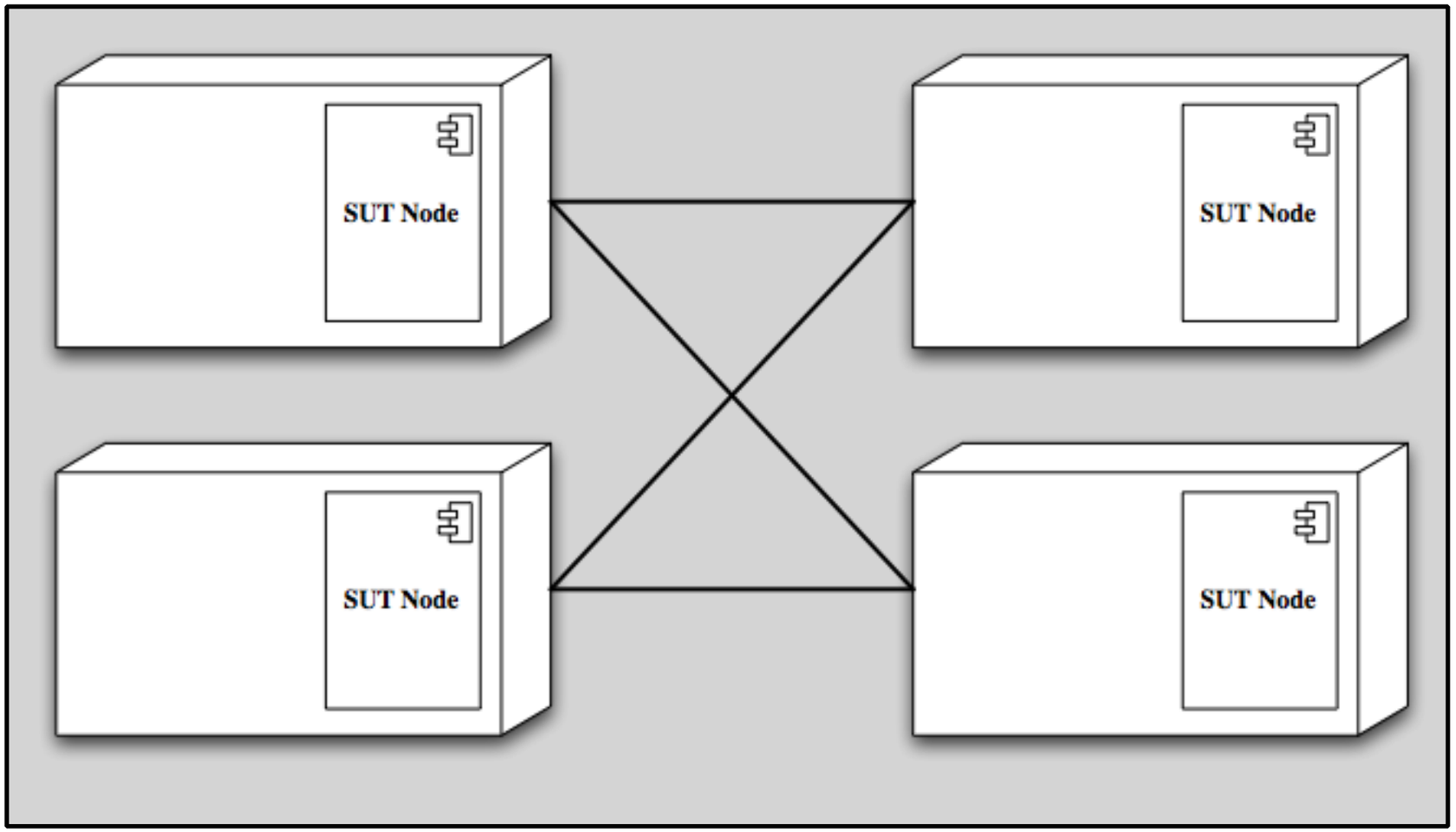
- Emulates browser behavior.

Apache JMeter

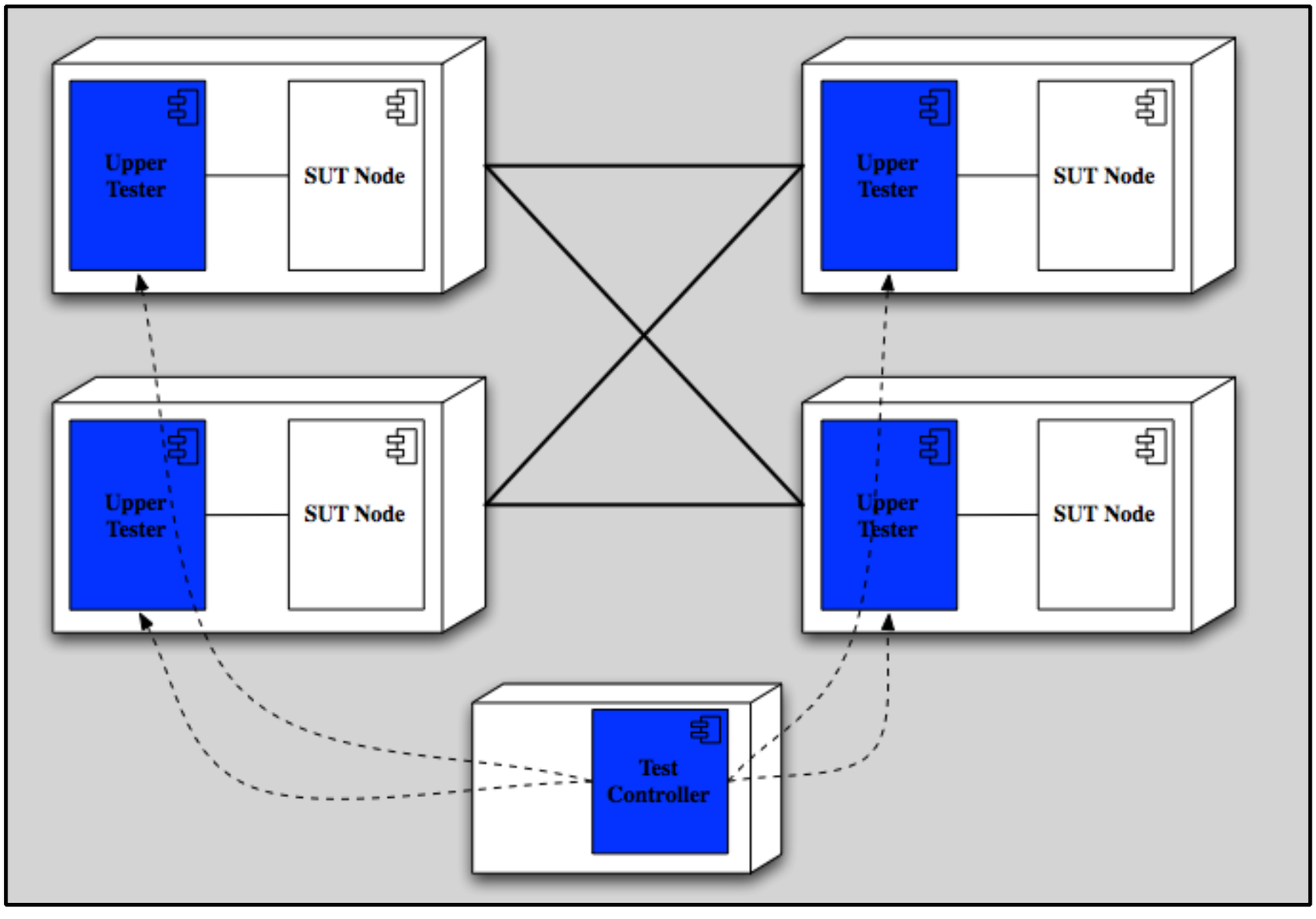
- Performance testing.
- Not restricted to web software.
  - DBMS, LDAP, JMS, SMTP, etc.

Distributed Test Architecture

A Distributed System



Distributed Testing Architecture

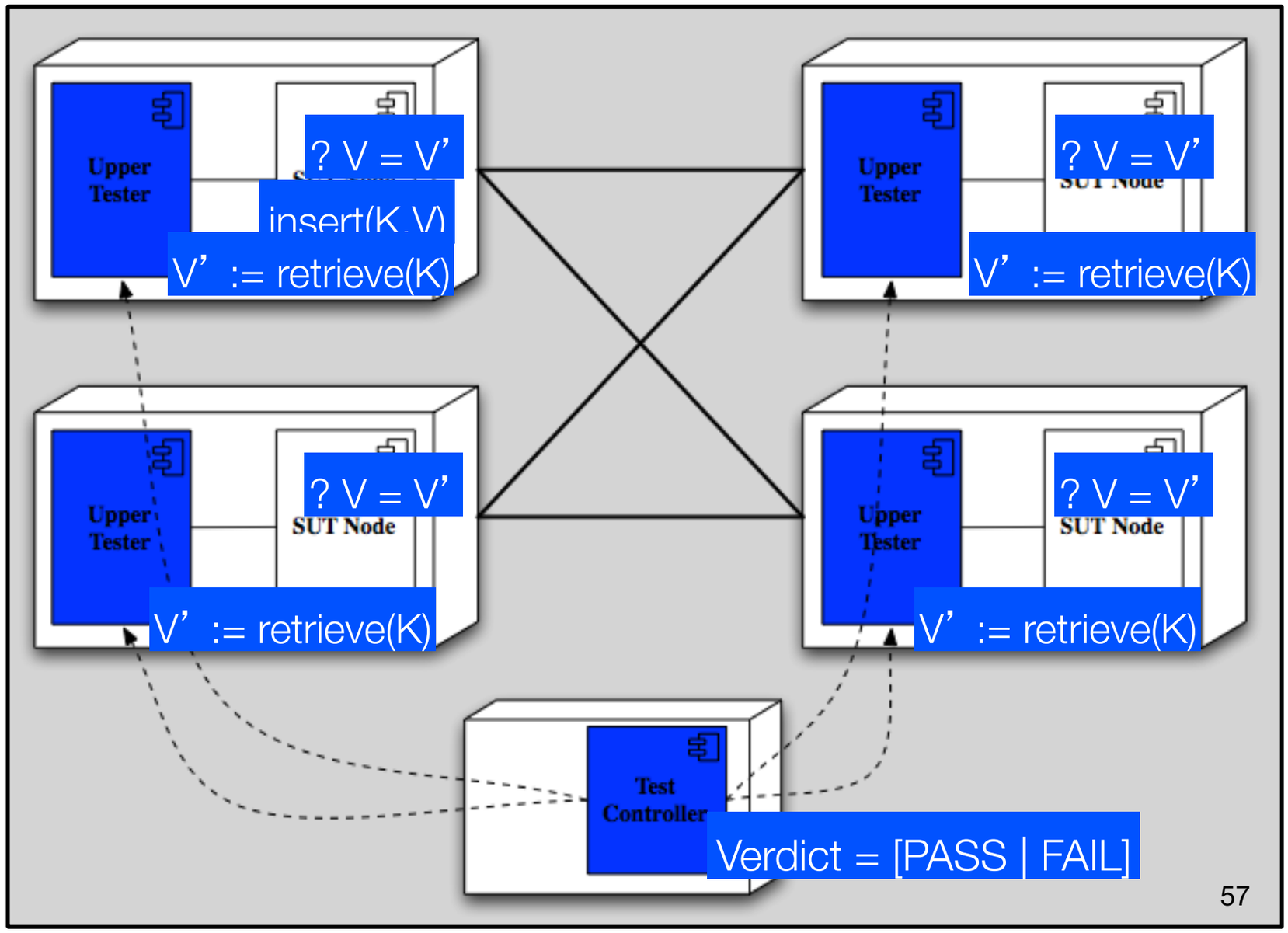


A Distributed Test Case

Step	Action	Nodes
1	insert(K,V)	1
2	V’ := retrieve(K)	all
3	? V = V’	all

Test Case Execution

Step	Action	Nodes
1	insert(K,V)	1
2	V’ := retrieve(K)	all
3	? V = V’	all



A Framework for Testing Distributed Systems

PeerUnit

- A framework for testing P2P systems;
- Written in Java 1.5;
- Open source, GPL license.

Properties

- Uses Java annotations.
- Synchronization (for > 1,000 testers).
- Volatility simulation.
- Shared variables.
- Distributed Architecture.

Basic Principles

Test	PeerUnit
Test Case	Java Class
Test Step	Java Method

A Simple Test Case

```
public class SimpleTest {
  @TestStep(range = "1", timeout = 100, order = 1)
  public void insert() {...}

  @TestStep(range = "*", timeout = 100, order = 2)
  public void retrieve() {...}

  @TestStep(range = "*", timeout = 100, order = 3)
  public void compare() {...}
}
```

Test Execution

Step 1 - Start the Test Controller:

java fr.inria.peerunit.CoordinatorRunner &

Step 2 - Run 1 or more Testers:

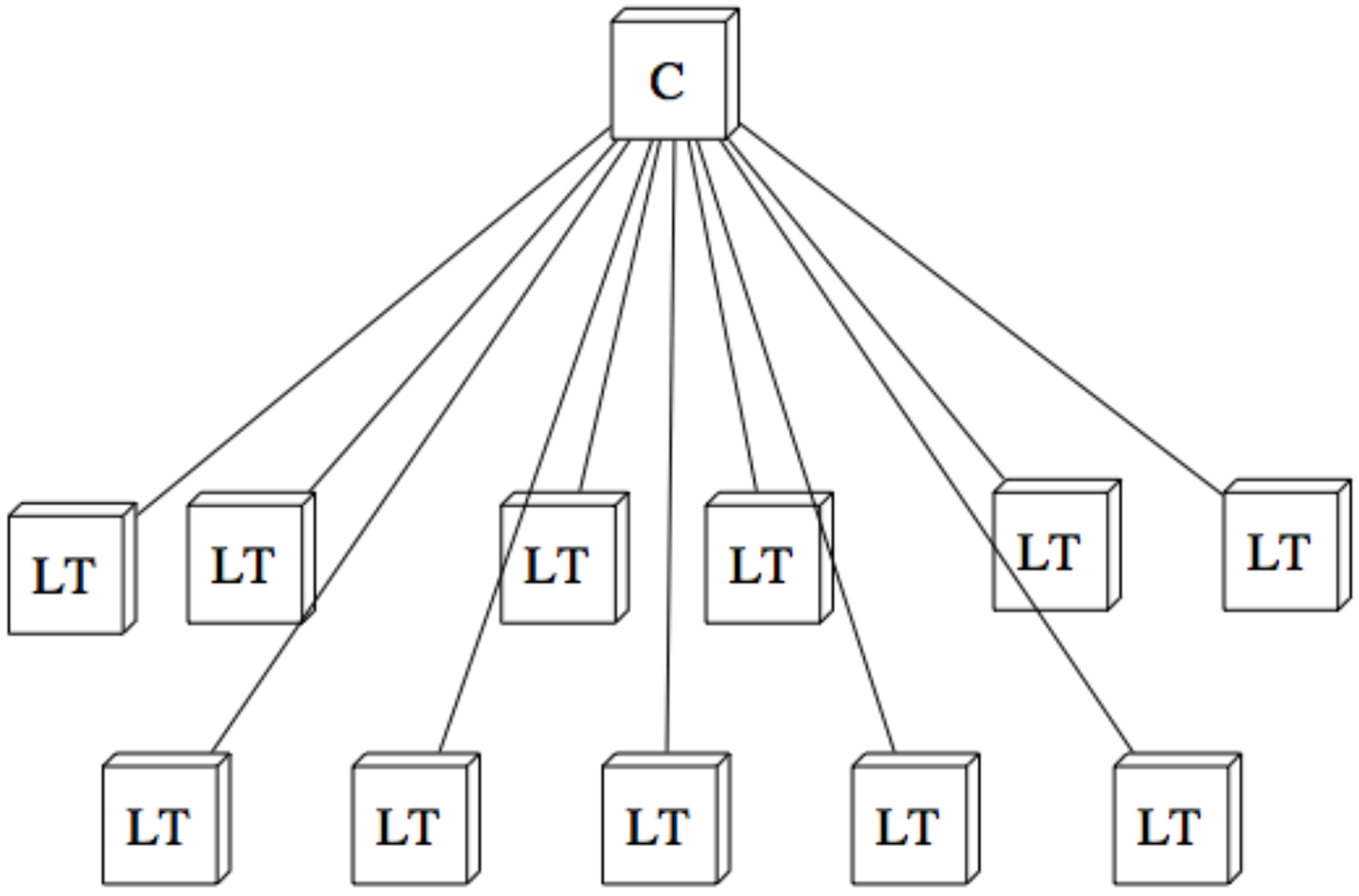
java fr.inria.peerunit.TestRunner SimpleTest &

Test Results

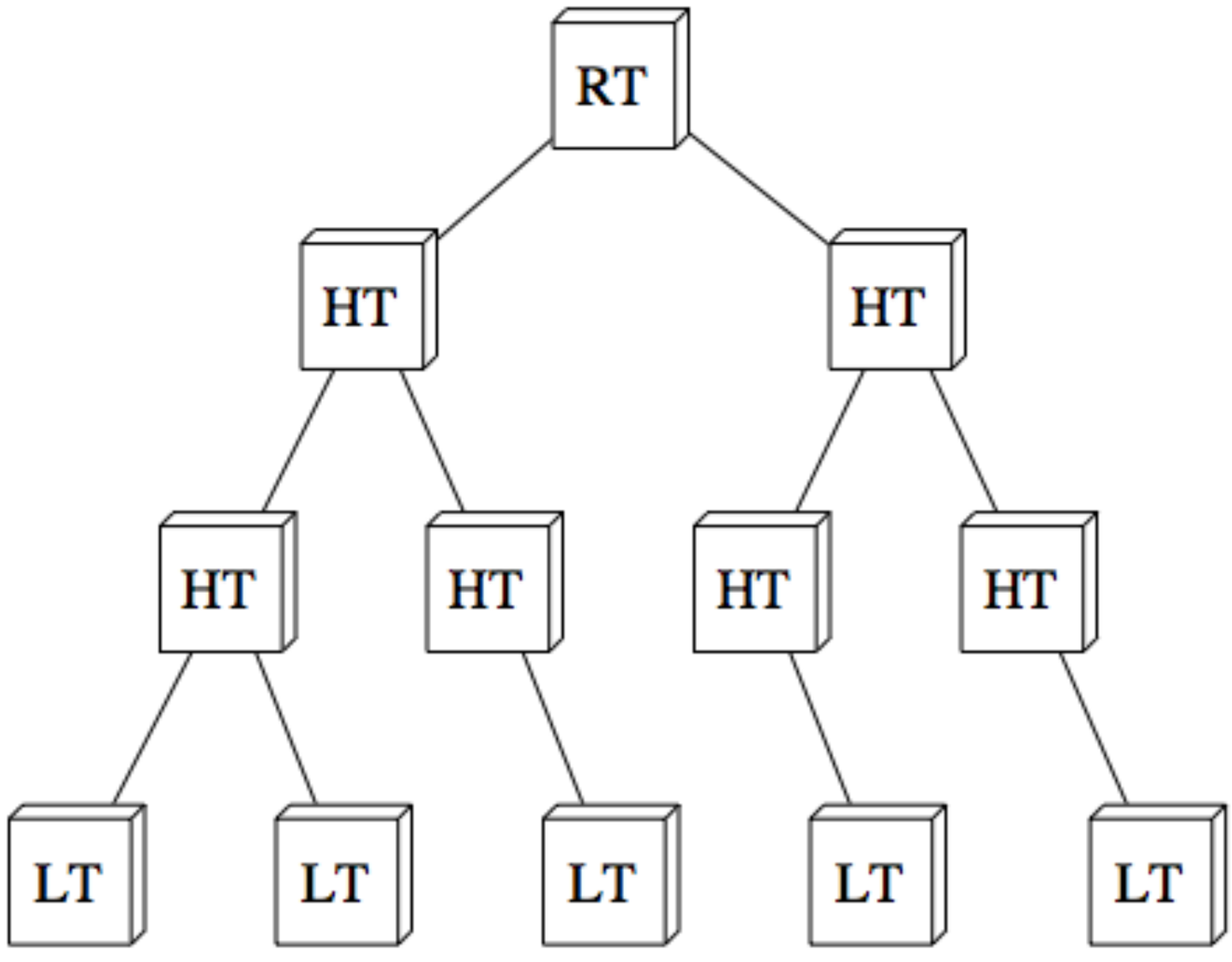
```
[java] ------------------------------
[java] Test Case Verdict:

[java] Step: 1. Pass: 1. Fails: 0. Errors: 0. Inconclusive: 0.  Time
elapsed: 23 msec. Average: 12 msec.     Method: insert

[java] Step: 2. Pass: 7. Fails: 0. Errors: 0. Inconclusive: 0.  Time
elapsed: 57 msec. Average: 11 msec.     Method: retrieve

[java] Step: 3. Pass: 7. Fails: 0. Errors: 0. Inconclusive: 0.  Time
elapsed: 5 msec. Average: 1 msec.       Method: compare

[java] ------------------------------
```

PeerUnit Architecture

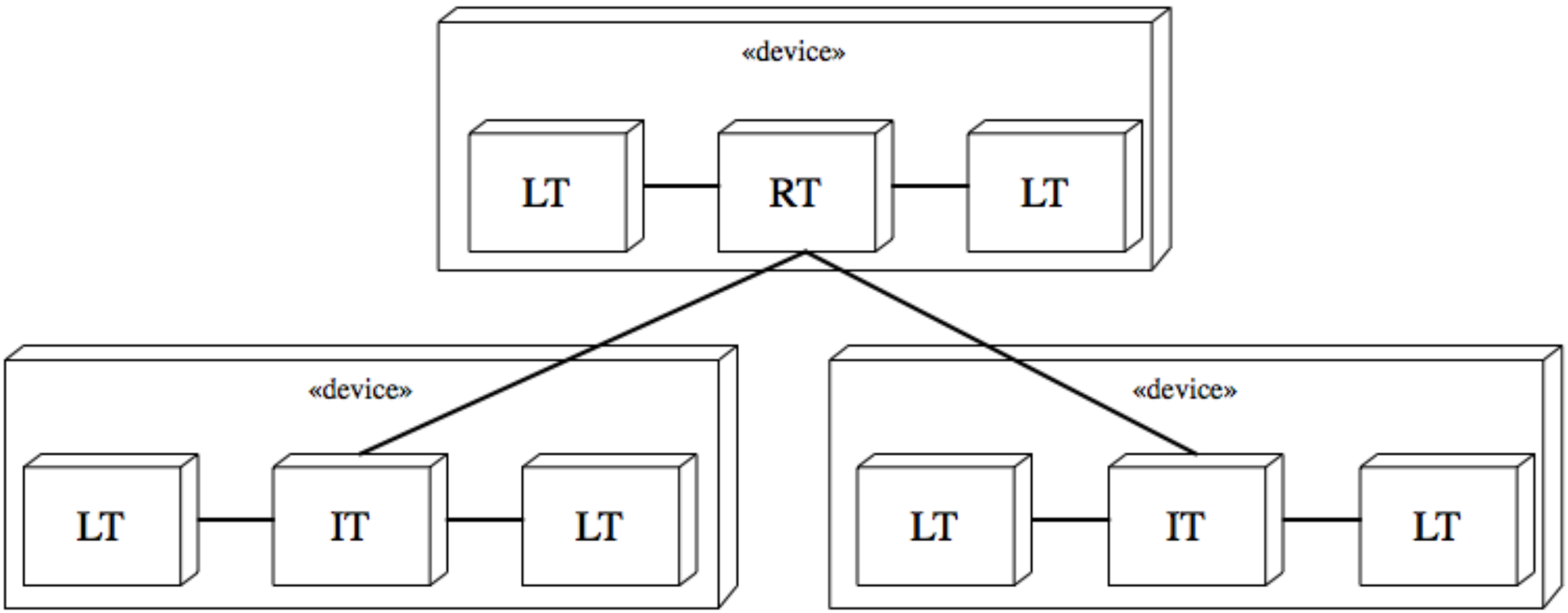
Architecture 1 : Centralized



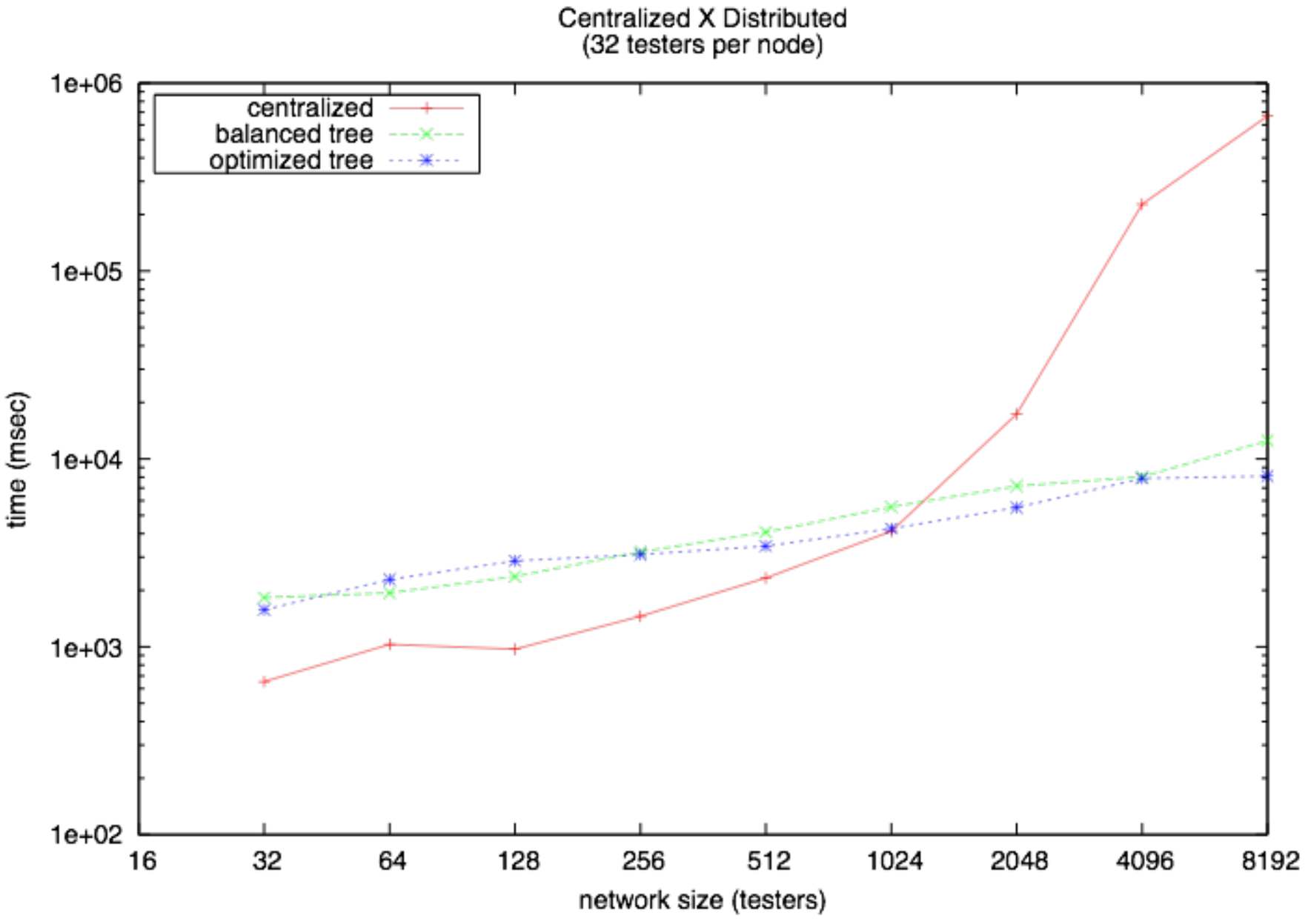
Architecture 2: Tree



Architecture 3: Optimized Tree



Synchronization Performance



Systems already tested

- OpenChord
- FreePastry
- Kademlia
- PostgreSQL

Testing a DHT

The SUT Interface

```
public interface RemoteDHT extends Remote {
        public void put(String k, Serializable v)
                        throws RemoteException;

        public Serializable get(String k)
                        throws RemoteException;
}
```

Test Case Summary

Step	Action	Testers
1	Create the (mock) DHT	1
2	Connect to DHT	*
3	Insert Data	2
4	Retrieve and Compare	*
5	Insert More Data	1
6	Retrieve and Compare	*

Test Case Implementation

(RemoteDHTTest.java)

Configuration

```
tester.peers=7
tester.server=127.0.0.1
tester.port=8181

# Log level order SEVERE,WARNING,INFO,CONFIG,FINE,FINER,FINEST
tester.log.level=INFO
tester.logfolder=.

# Tree Parameters
test.treeOrder=1

# Coordination Type (0) centralized, (1) tree
test.coordination=1
```

Test Case Execution

- Create a build file.
- Deploy the test bed.
- Execute.
- Demo.

Conclusion and References

Conclusion

- Large Scale Distributed Systems provide a new way to build software.
  - Several new technologies introducing fascinating new problems.
- Very active research area.

Open Issues

- Distributed architecture optimization.

- Lack of tools for: deployment, log analysis, graphical interface, Maven plugin, etc.

- Few users.

Bibliography

- « Testing Object-Oriented Systems: Models, Patterns, and Tools».
- Robert V. Binder.
- Addison-Wesley Object Technology Series.
- «Introduction to Software Testing».
- Paul Ammann and Jeff Offutt.
- Cambridge University Press.

« If debugging is the process of removing bugs, then programming must be the process of putting them in»

Edsger W. Dijkstra

« There’s always one more bug»

Lubarsky’s Law of Cybernetic Entomology