In-Memory Computing SUMMIT EUROPE 2019

How to Test the Ability of Large-Scale, Distributed Software Systems to Cope with Failures

Pavel Lipsky
Dell Technologies
06/04/2019

DELL Technologies

Who am I?

https://github.com/leapsky

Pavel Lipsky

Before 2005
Building scalable web sites

From 2005 to 2014
Test automation and DevOps

From 2014
Performance and reliability of large-scale, distributed systems

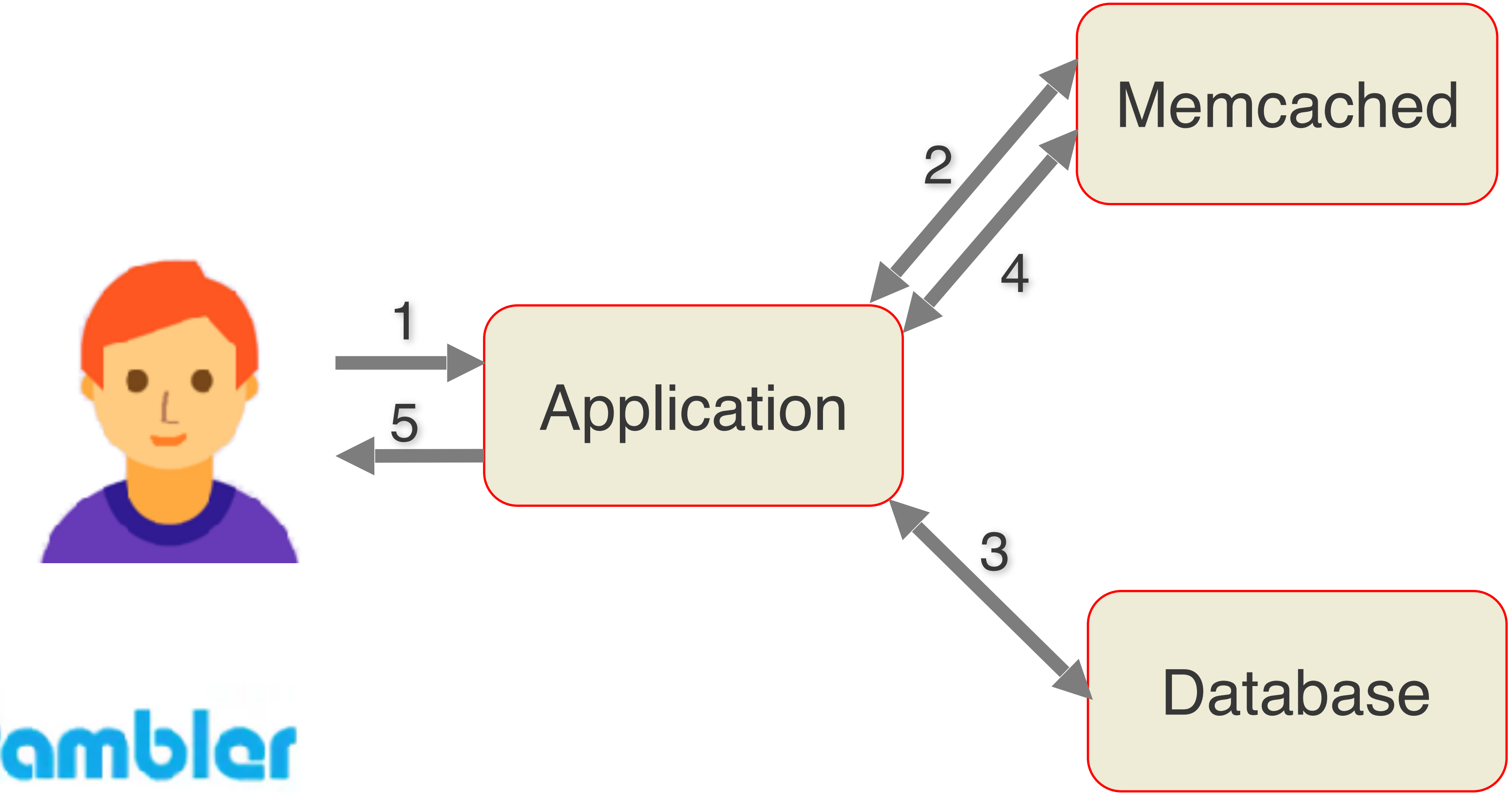
Agenda

- What is Fault Injection?
- Test Object
- Stories & Demos - https://github.com/leapsky
- Tools & Frameworks

Story 1

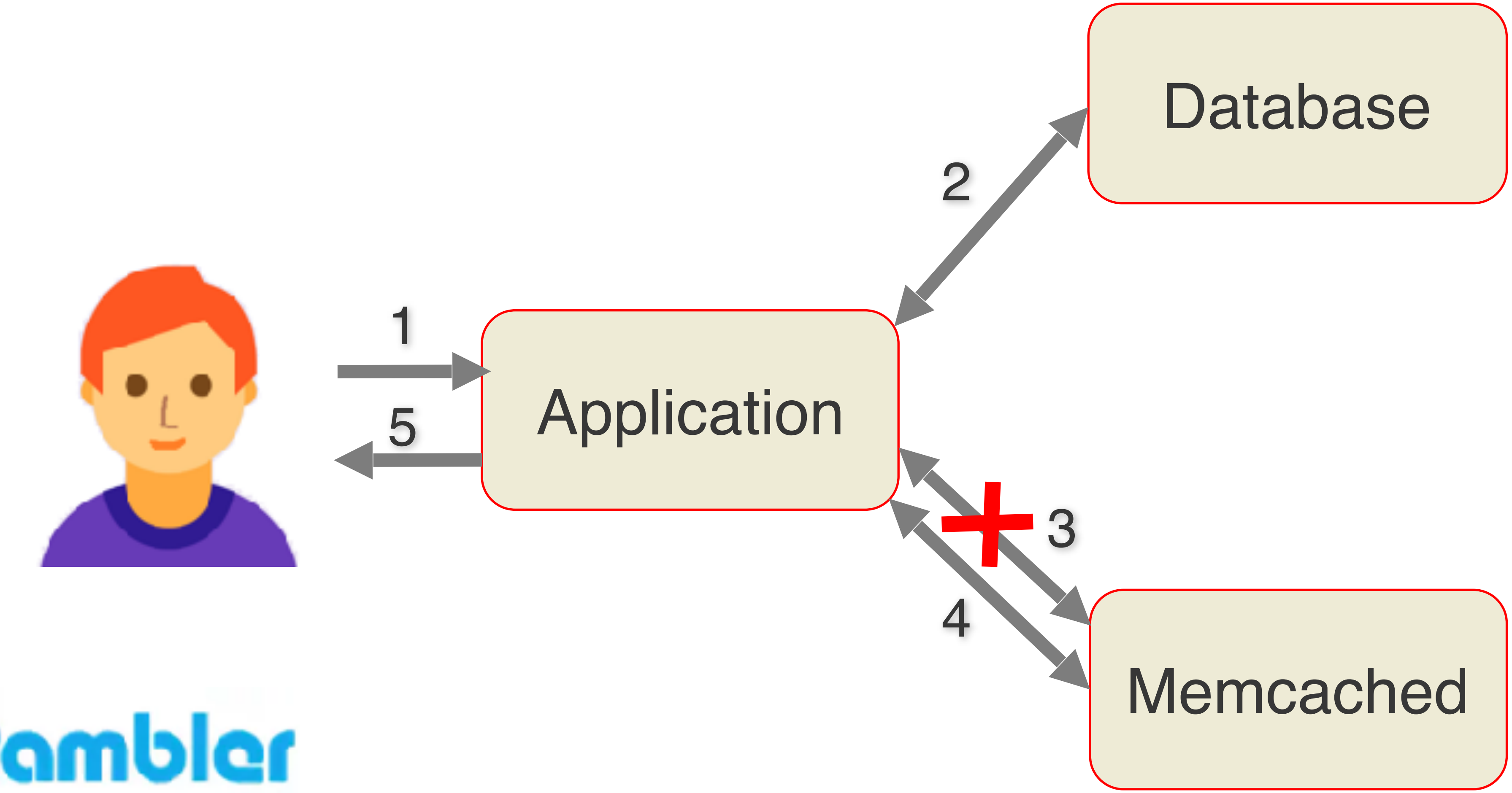
Memcached

Fetching Data from Memcached



Rambler

Changing Data in Memcached



Rambler

Types of Software Testing

Load testing

Functional testing

Fault Injection

Security testing

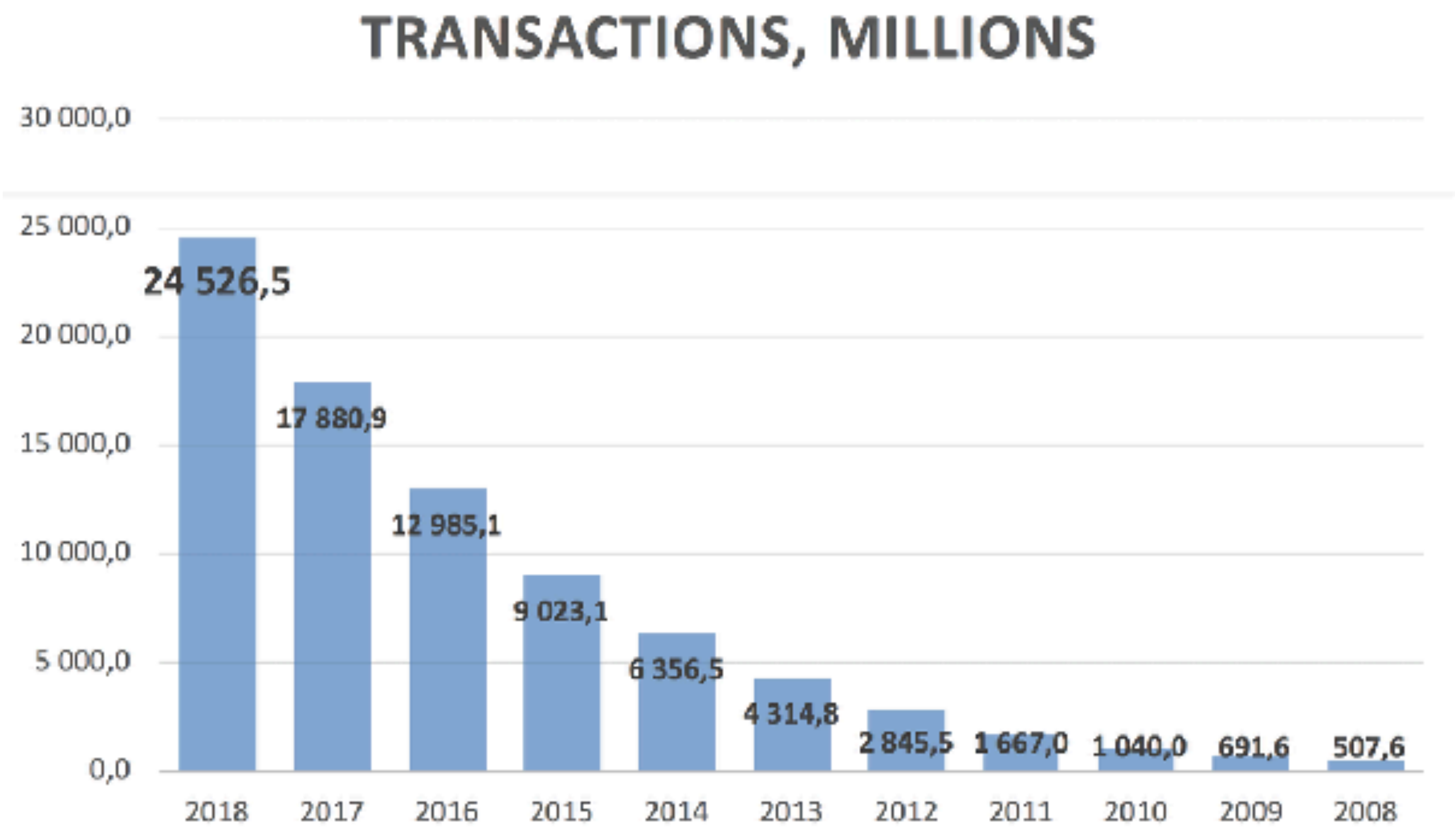
Usability testing

Story 2

GridGain

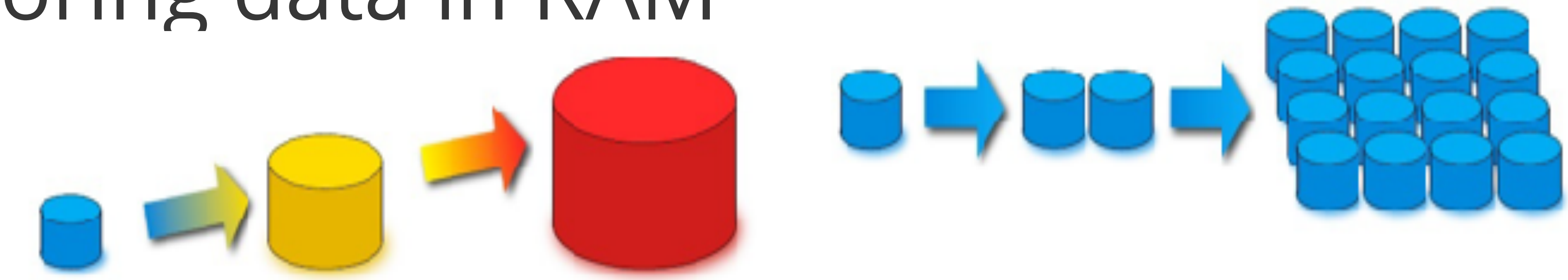
In-Memory Computing SUMMIT EUROPE 2019

Payments for Goods with Payment Cards Issued by Russian Banks


TRANSACTIONS, MILLIONS

New IT Platform

- Horizontal scaling
- Using open-source software
- Affordable low-end hardware
- Reliability
- Storing data in RAM



GridGain Enterprise

- SQL support
- Quick access to objects by key
- In-memory computing
- Persistent Data Store
- Strong consistency
- Failure resistance
- Horizontal scalability
- …

Forcing a System to Fail

“Without explicitly forcing a system to fail, it is unreasonable to have any confidence it will operate correctly in failure modes.”Caitie McCaffrey (Backed Brat & Distributed Systems Diva),
The Verification of a Distributed System

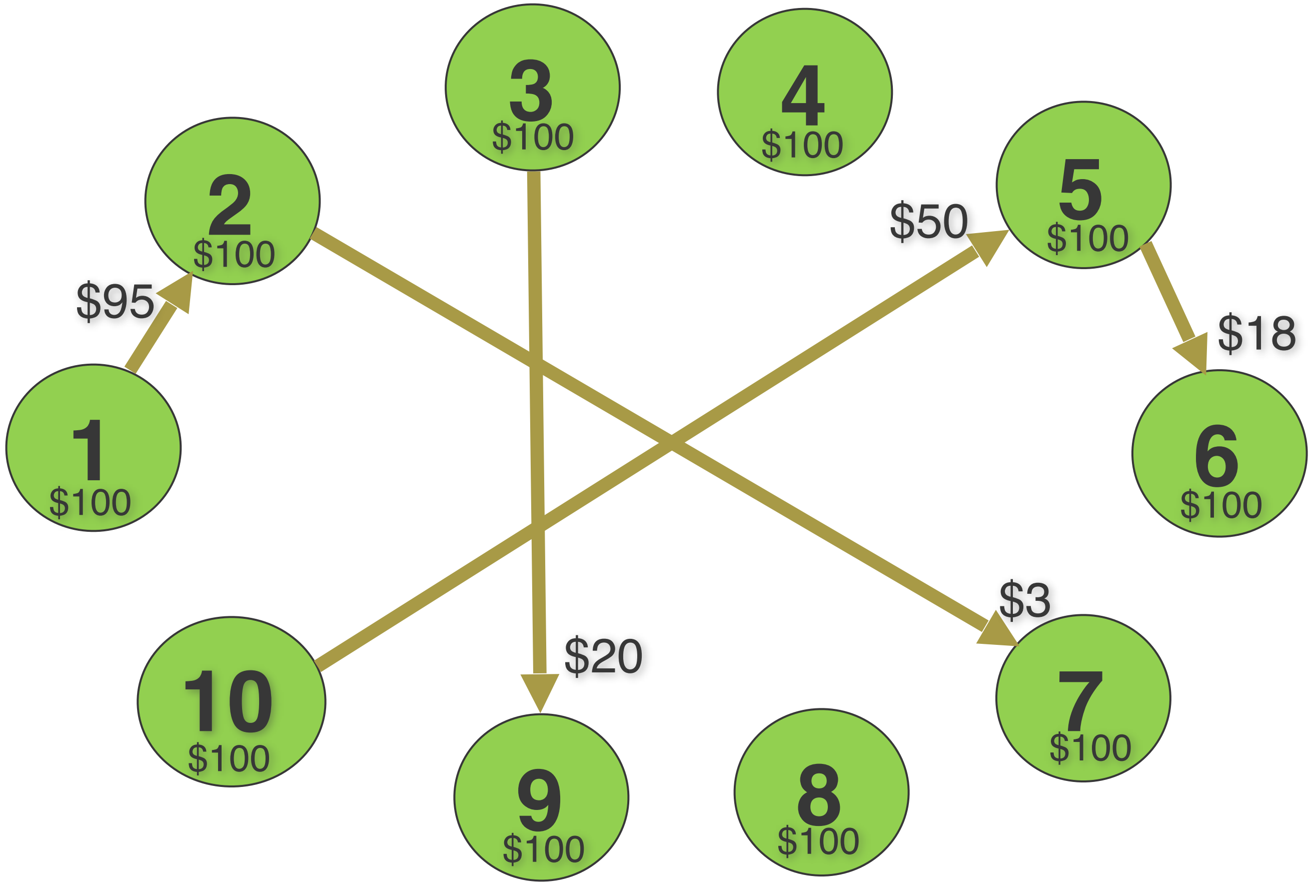
Story 3

Lost Updates

Example of Fund Transfer

```
1. read(A)
2. A := A - 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

Fund Transfers Between Bank Accounts



Demo Time

Lost Updates

Lost Updates

A := $50

	Task 1	Task 2
T1	`read(A)`	`read(A)`
T2	`A:= A - 50`	`A := A - 50`
T3	`write(A)`	
T4		`write(A)`
T5	…	…

Expected value of A is $50

Real value of A is $0

Story 4

ACID

ACID Properties

- **Atomicity**
- **Consistency**
- **Isolation**
- **Durability**

```
1. read(A)
2. A := A - 50
3. write(A)
4. read(B)
5. B := B + 50
6. write(B)
```

Isolation Levels and the ANSI/ISO SQL Standard

Isolation Levels	Dirty Read	Non-Repeatable Read	Phantom Read
READ UNCOMMITTED	Permitted	Permitted	Permitted
READ COMMITTED	--	Permitted	Permitted
REPEATABLE READ	--	--	Permitted
SERIALIZABLE	--	--	--

READ_COMMITTED

A := $50

	Transaction	Transaction 2
T1	read(A)	read(A)
T2	A:= A - 50	A := A + 50
T3	write(A)	
T4	commit	write(A)
T5	…	commit

Expected value of A is $50

Real value of A is $100

Apache Ignite Concurrency Modes and Isolation Levels

Concurrency Modes

- PESSIMISTIC
- OPTIMISTIC

Isolation Levels

- READ_COMMITTED
- REPEATABLE_READ
- SERIALIZABLE

Apache Ignite Documentation: Concurrency Modes and Isolation Levels

PESSIMISTIC REPEATABLE_READ - Entry lock is acquired and data is fetched from the primary node on the first read or write access and stored in the local transactional map. All consecutive access to the same data is local and will return the last read or updated transaction value. This means no other concurrent transactions can make changes to the locked data, and you are getting Repeatable Reads for your transaction.

OPTIMISTIC SERIALIZABLE - Stores an entry version upon first read access. Ignite will fail a transaction at the commit stage if the Ignite engine detects that at least one of the entries used as part of the initiated transaction has been modified.

Demo Time

Transactions

.txStart(CONCURRENCY_MODE, ISOLATION_LEVEL)

```
try (Transaction tx = ignite.transactions().txStart(OPTIMISTIC, SERIALIZABLE)) {
    Account fromAccount = cache.get(fromAccountId);
    Account toAccount = cache.get(toAccountId);
    ...
    tx.commit();
}
```

Story 5

Testing Under Load

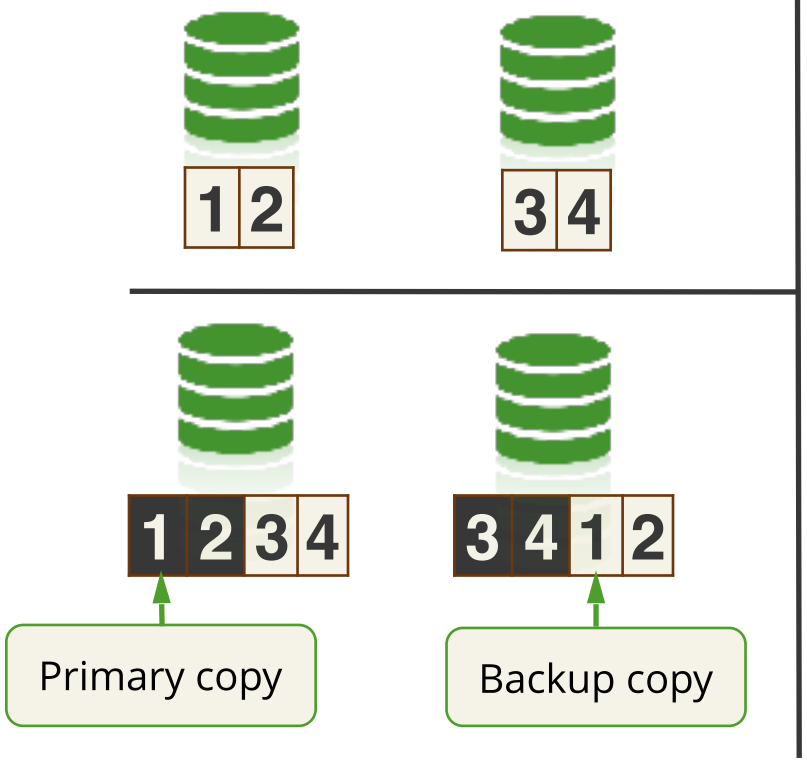
Performance Testing Tools

Demo Time

APACHE JMeter™

What cache mode to choose?

PARTIONED



1 2

3 4

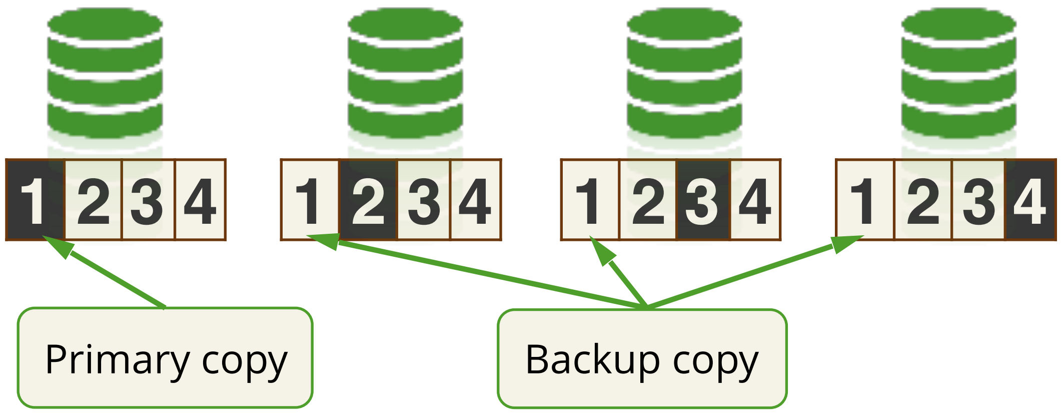
1 2 3 4

3 4 1 2

Primary copy

Backup copy

REPLICATED



1 2 3 4

1 2 3 4

1 2 3 4

1 2 3 4

Primary copy

Backup copy

.txStart(CONCURRENCY_MODE, ISOLATION_LEVEL)

```
CacheConfiguration<Integer, Account> cfg = new CacheConfiguration<>(CACHE_NAME);
cfg.setAtomicityMode(CacheAtomicityMode.TRANSACTIONAL);
cfg.setCacheMode(CacheMode.PARTITIONED);
cfg.setBackups(2);
```

Demo Time

In-Memory Computing SUMMIT EUROPE 2019

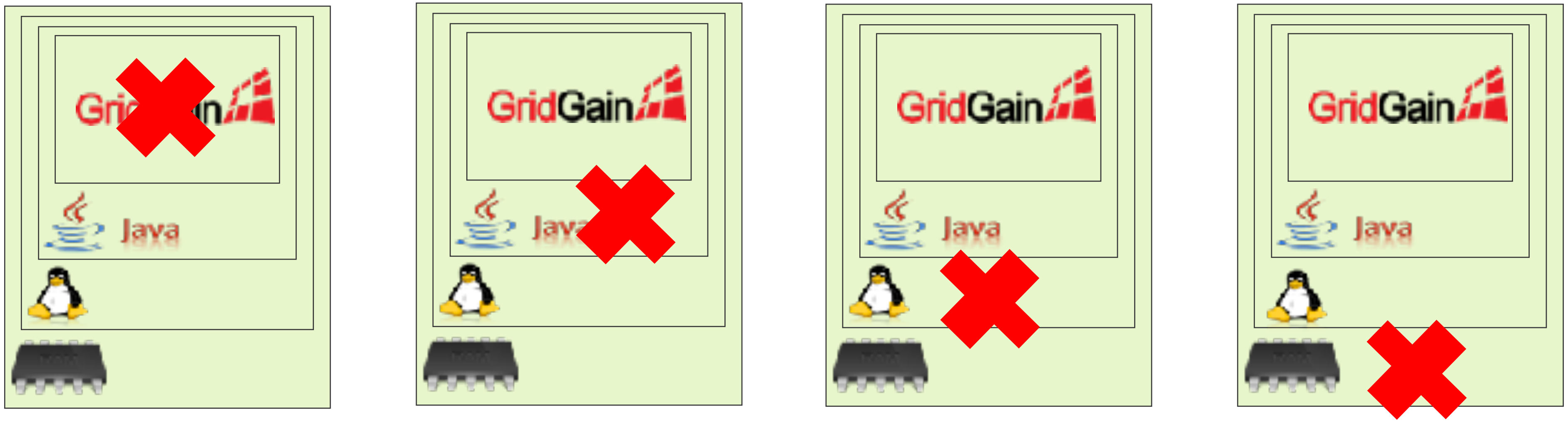
Jepsen Test

```
lein run test \
  --test bank \
  --time-limit 60 \
  --concurrency 5 \
  --nodes-file nodes \
  --username root \
  --password root \
  --cache-mode PARTITIONED \
  --cache-atomicity-mode TRANSACTIONAL \
  --cache-write-sync-mode FULL_SYNC \
  --read-from-backup YES \
  --transaction-concurrency PESSIMISTIC \
  --transaction-isolation REPEATABLE_READ \
  --backups 2 \
  --pds true \
  --version 2.7.0 \
  --os debian \
  --nemesis kill-node
```

Story 6

Disruptive Scenarios

Node failure

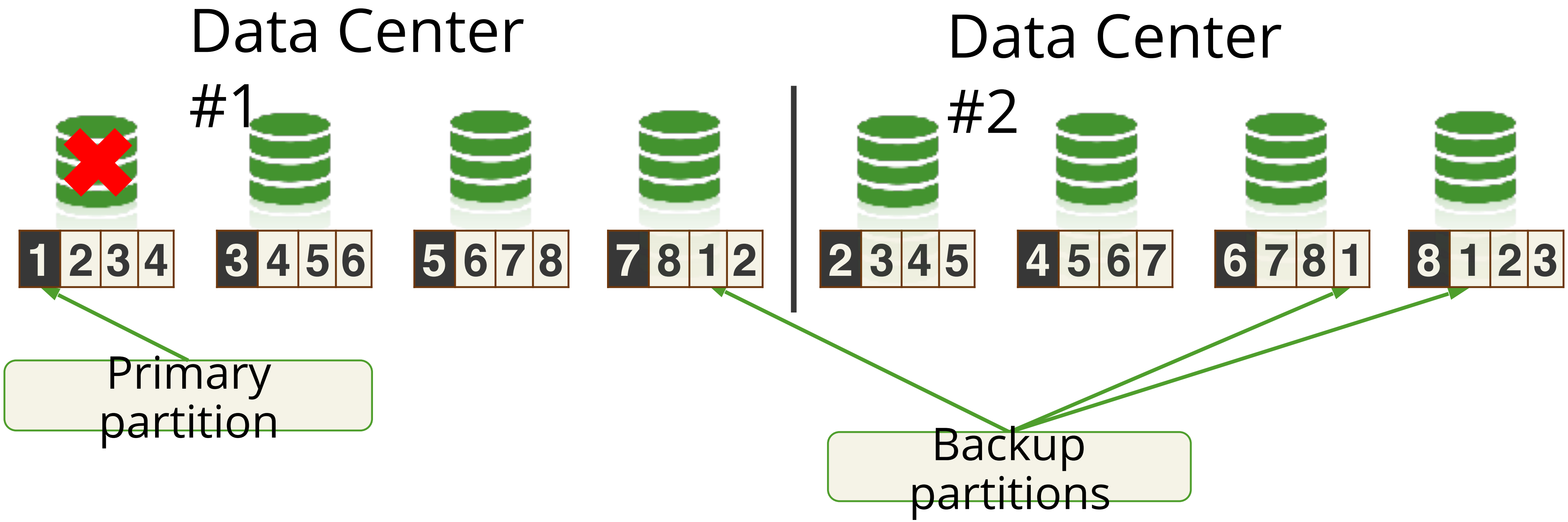


Application crash JVM crash OS crash Hardware crash

Disruptive Scenarios

- Hardware
- Network
- Application
- Other scenarios

Disruptive Scenarios: Hardware



Data Center #1

Data Center #2

1 2 3 4

3 4 5 6

5 6 7 8

7 8 1 2

2 3 4 5

4 5 6 7

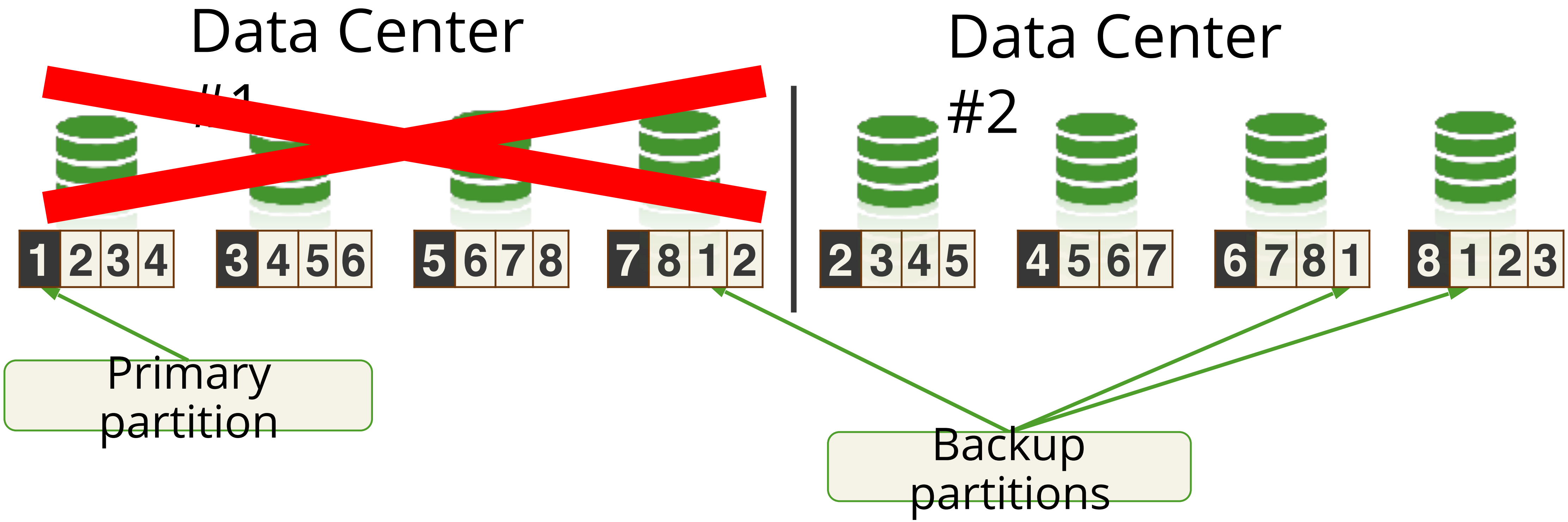
6 7 8 1

8 1 2 3

Primary partition

Backup partitions

Disruptive Scenarios: Hardware

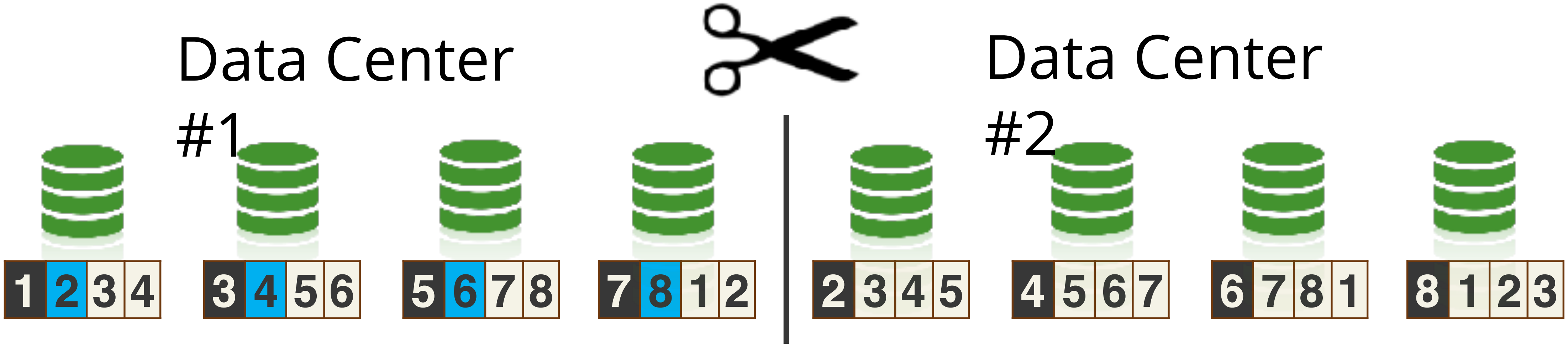


Disruptive Scenarios: Network

- iptables

- NetEm emulates:
  - network delays with different distribution functions
  - packet loss
  - repeat packets
  - reordering of packets
  - packet distortion

Disruptive Scenarios: Network



Data Center #1

Data Center #2

1 2 3 4

3 4 5 6

5 6 7 8

7 8 1 2

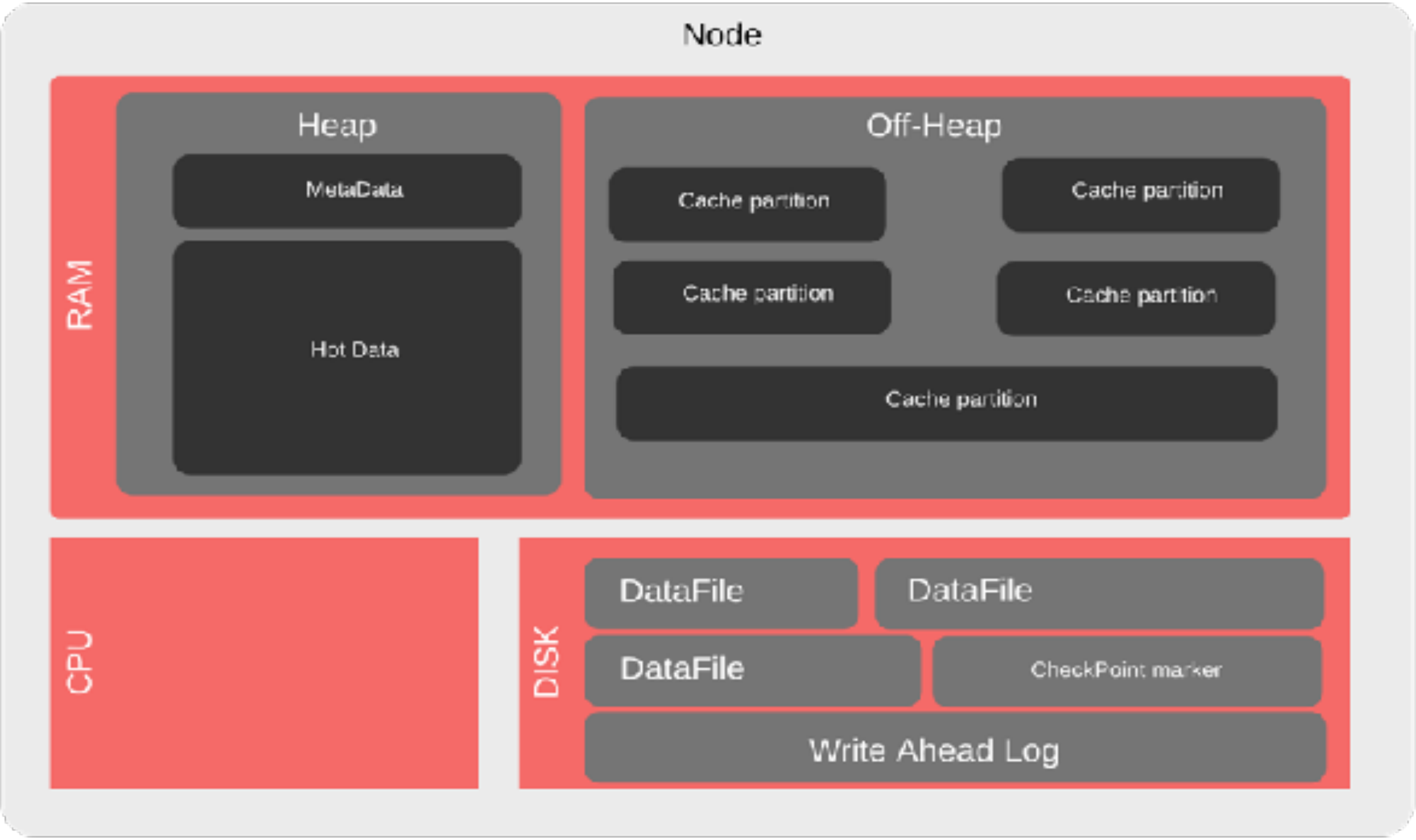
2 3 4 5

4 5 6 7

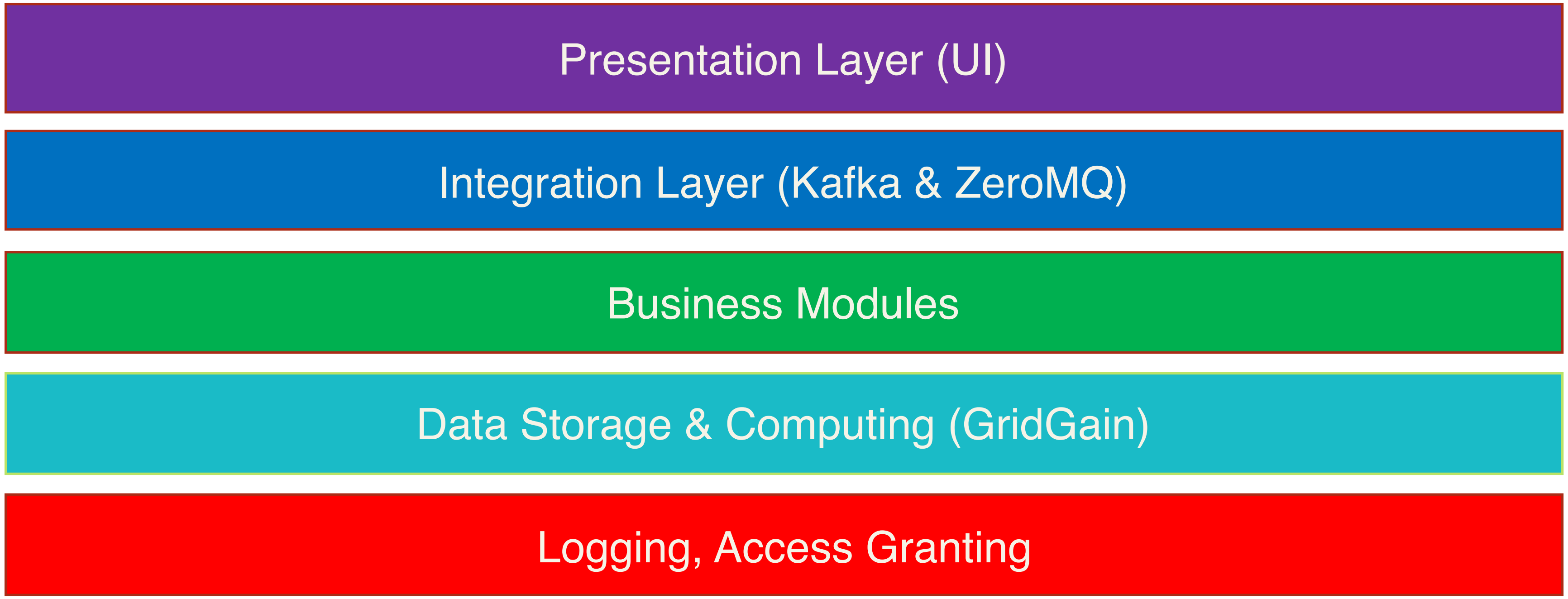
6 7 8 1

8 1 2 3

Disruptive Scenarios: Application



Disruptive Scenarios: Application



Presentation Layer (UI)

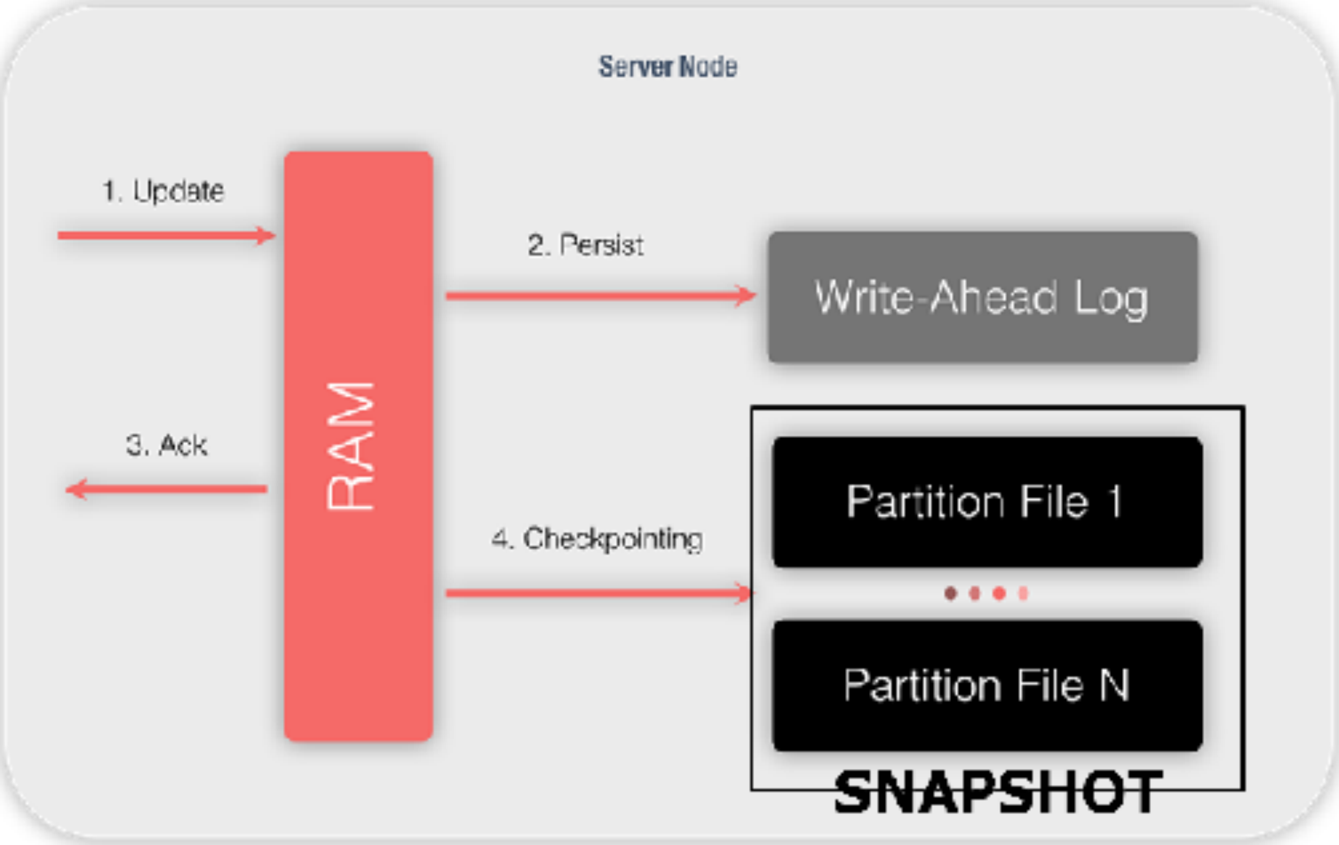
Integration Layer (Kafka & ZeroMQ)

Business Modules

Data Storage & Computing (GridGain)

Logging, Access Granting

Disruptive Scenarios: Other Scenarios



Tools to start using Fault Injection

Code examples

https://github.com/leapsky/FaultInjectionExamples

Frameworks

Jepsen - https://github.com/jepsen-io/jepsen
Chaos Monkey - https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey

Linux Utilities

NetEm (tc) - https://wiki.linuxfoundation.org/networking/netem
stress-ng - https://manned.org/stress-ng/fd34c972
Iperf - https://iperf.fr
kill -9

Load testing tools

JMeter - https://jmeter.apache.org

Configuration Management

Ansible - https://docs.ansible.com
Puppet - https://puppet.com

Lessons Learned

- **Fault Injection** is the art of explicitly forcing a system to fail to make sure that it will operate correctly in failure modes.
- **No risk** - **no test!**
- Test results must be **clear** and **unambiguous**.
- The closer your **test environments match** your **production environments**, the more accurate your testing will be.

Thank you! Questions?

Pavel Lipsky

pavel.lipsky@gmail.com

https://github.com/jepsen-io/jepsen/tree/master/ignite

https://github.com/leapsky/