



FACULTY  
OF INFORMATICS

Masaryk University

# Use of Transactions within a Reactive Microservices Environment

---

Bc. Martin Štefanko

# Assignment

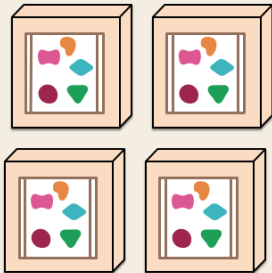
- Review the state of the art, in terms of problems of synchronous/blocking approaches for transaction management and other approaches/patterns available - taking into account the microservices context
- Propose a proof-of-concept implementation, using the Narayana transaction manager and prepare a service capable to manage transactions in the context of reactive microservices
- Prepare an example/quickstart showing the whole issues in more practical terms, proving that the transaction manager can work in an asynchronous environment

# Microservices

*A monolithic application puts all its functionality into a single process...*



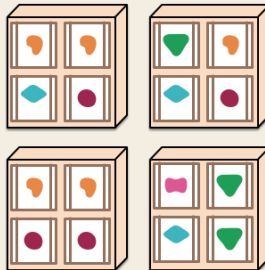
*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*

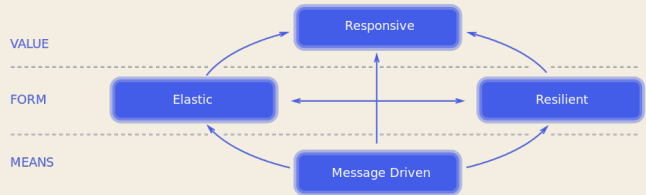


*... and scales by distributing these services across servers, replicating as needed.*



# Reactive microservices

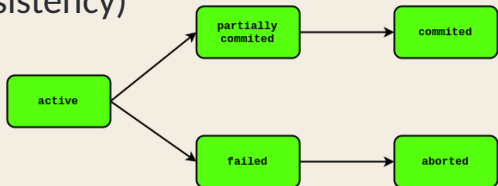
- reactive systems
- reactive programming
- reactive streams



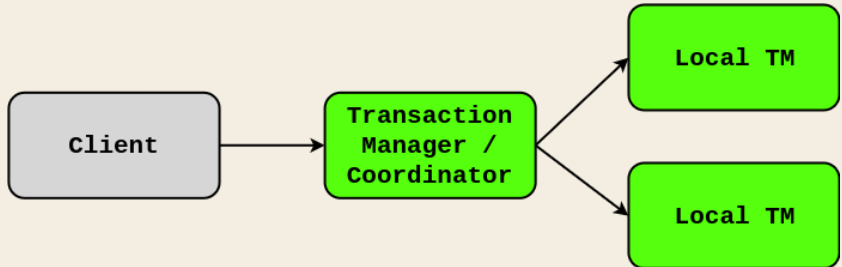
# Transactions

*"A transaction is a unit of processing that provides all-or-nothing property to the work that is conducted within its scope, also ensuring that shared resources are protected from multiple users" [1].*

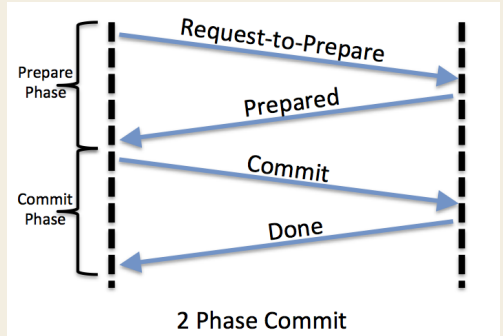
- sequence of operations
- commit or rollback
- ACID (values consistency)



## Distributed transactions



## Two phase commit protocol



- $O(n^2)$  messages
- blocking
- coordinator - single point of failure

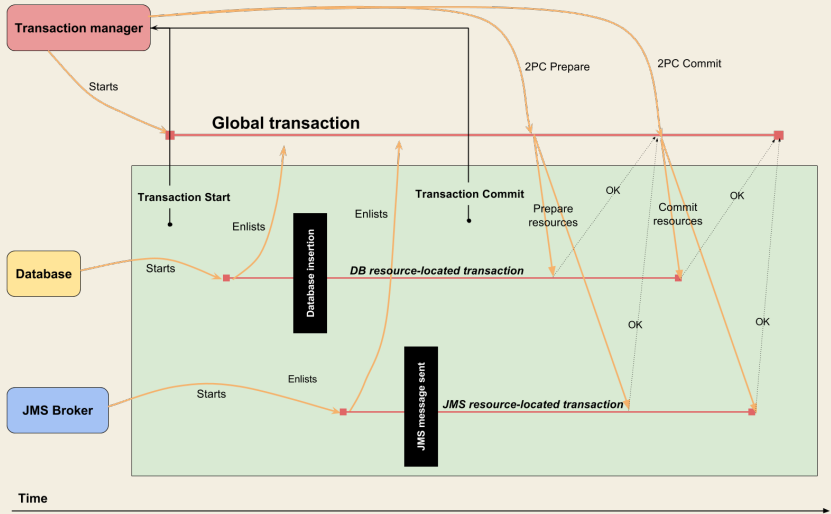
# Saga pattern

*Hector Garcia-Molina and Kenneth Salem, Princeton University, 1987*

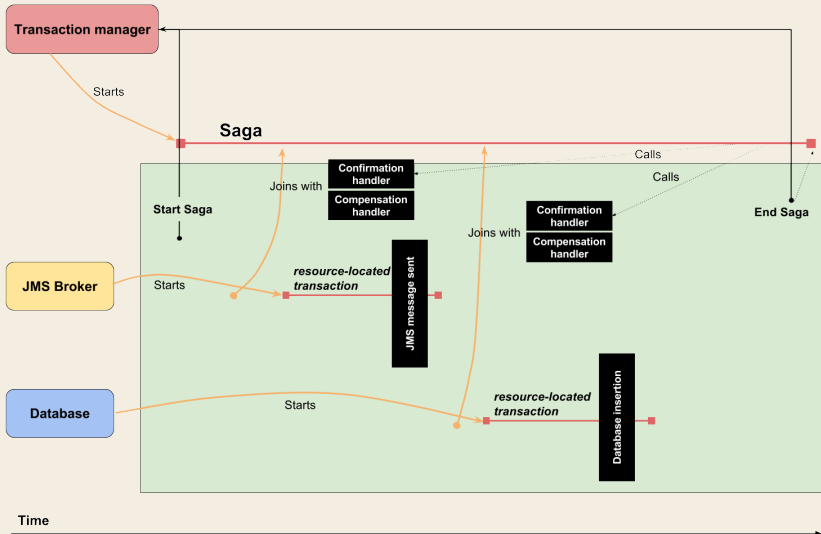
- long lived transactions
- compensations
- all-or-nothing property
  - 2PC -  $T_1, T_2, \dots, T_n$  (in a single step)
  - Saga
    - » success -  $T_1, T_2, \dots, T_n$
    - » failure -  $T_1, T_2, \dots, T_k, C_k, C_{k-1}, \dots, C_1$
- BASE (values availability)
  - Basically Available, Soft state, Eventual consistency



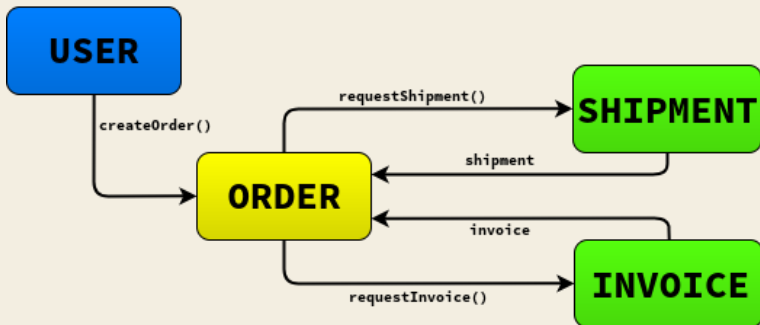
# Two phase commit protocol



# Saga pattern



## Saga implementations comparison scenario



## Saga implementations investigation

- Axon framework
- Eventuate Event Sourcing (ES)
- Eventuate Tram
- Narayana Long Running Actions (LRA)

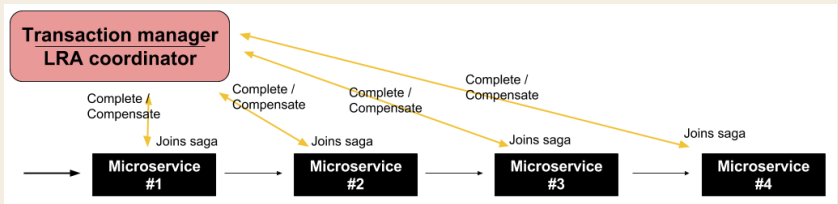
## Saga implementations comparison

Problem	Axon	Eventuate ES	Eventuate Tram	LRA
CQRS restriction	Yes	Yes	Optional	No
Asynchronous by default	Yes	Yes	No	No
Saga tracking and definition	No	No	Yes	No
Single point of failure	No	Yes	Yes	Yes*
Communication restrictions	Yes	Yes	Yes	No
Distributed by default	No	Yes	Yes	Yes

## Saga implementations performance testing

- Axon - 2 reported issues
- Eventuate ES - 1 reported issue
- Eventuate Tram - 1 feature request
- Narayana LRA

## LRA executor motivation



## LRA executor extension

- proof of concept / prototype
- asynchronous
- extensible and flexible design
- protocol / platform independent
  - further future extensions are expected
- two modules
  - LRA definitions
  - LRA executor



# LRA Definitions

- LRADefinition
- Action
- fluent API

```
RESTLraBuilder.lra()  
  .name("testLRA")  
  .withAction(RESTAction  
    .post(new URL("http://example.com/request"))  
    .callbackUrl(new URL("http://example.com/callback"))  
    .build())  
  .data(42)  
  .callback("http://local.org")  
  .build();
```

```
{  
  "name": "testLRA",  
  "actions": [{  
    "target": "http://example.com/request",  
    "callbackUrl": "http://example.com/callback"  
  }],  
  "data": 42,  
  "parentLRA": null,  
  "clientId": "",  
  "timelimit": 0,  
  "callbackUrl": "http://local.org",  
  "nestedLRAs": []  
}
```

# LRA executor

- LRAExecutor
- synchronous and asynchronous executions
- AbstractLRAExecutor – default implementation
  - actions are invoked in the declared order
- LRA manipulation methods
  - startLRA, completeLRA, compensateLRA
- integrated and tested (quickstart) with Narayana 5.8.1.Final

## Future work

- integration in the Narayana codebase
- communication methods
- definition representations
- processing strategies

# Questions

## Bibliography

- [1] M. Little, J. Maron, and G. Pavlik. *Java transaction processing*. Prentice Hall, 2004.
- [2] Leslie Lamport. *TEX: A Document Preparation System*. Addison-Wesley, 1986.
- [3] M. Goossens, F. Mittelbach, and A. Samarin. *The TEX Companion*. Addison-Wesley, 1994.
- [4] Till Tantau. *User's Guide to the Beamer Class Version 3.01*. Available at <http://latex-beamer.sourceforge.net>.
- [5] M. Fowler. *Microservices* Available at <https://www.martinfowler.com/articles/microservices.html>
- [6] Team Encrypt. *2 – Phase Commit Protocol* <https://encrypt.co.in/2-phase-commit-protocol/>

## Opponent's review

- transaction heuristic outcomes
  - heuristic commit, rollback, mixed
  - non-atomic outcome
  - requires semantic knowledge
- LRA service performance test
  - REST requests queuing
- recovery capabilities of the executor
  - main concern - failure after the marking of the participant invocation
  - idempotent requests (may be too restrictive)
  - timeouts

## Supervisor's review

- performance testing
- LRA specification relations
  - still in the draft form
  - focusing only on the coordination capabilities
  - currently only providing the REST reference implementation