Red Hat

TRANSACTIONAL SOLUTION FOR MICROSERVICES

ADAM RŮŽIČKA A ONDŘEJ CHALOUPKA

ABOUT US

ADAM RŮŽIČKA

- Red Hat
- developer of Foreman project
- mainly Ruby
- github: github.com/adamruzicka

ONDŘEJ CHALOUPKA

- Red Hat
- developer at WildFly project
- mainly Java
- working on Narayana transaction manager
- github: github.com/ochaloup
- twitter: @_chalda

AGENDA

AGENDA

- what is transaction management

AGENDA

- what is transaction management
- microservices and transactions

AGENDA

- what is transaction management
- microservices and transactions
- introduction to saga pattern

AGENDA

- what is transaction management
- microservices and transactions
- introduction to saga pattern
- Long Running Actions for MicroProfile

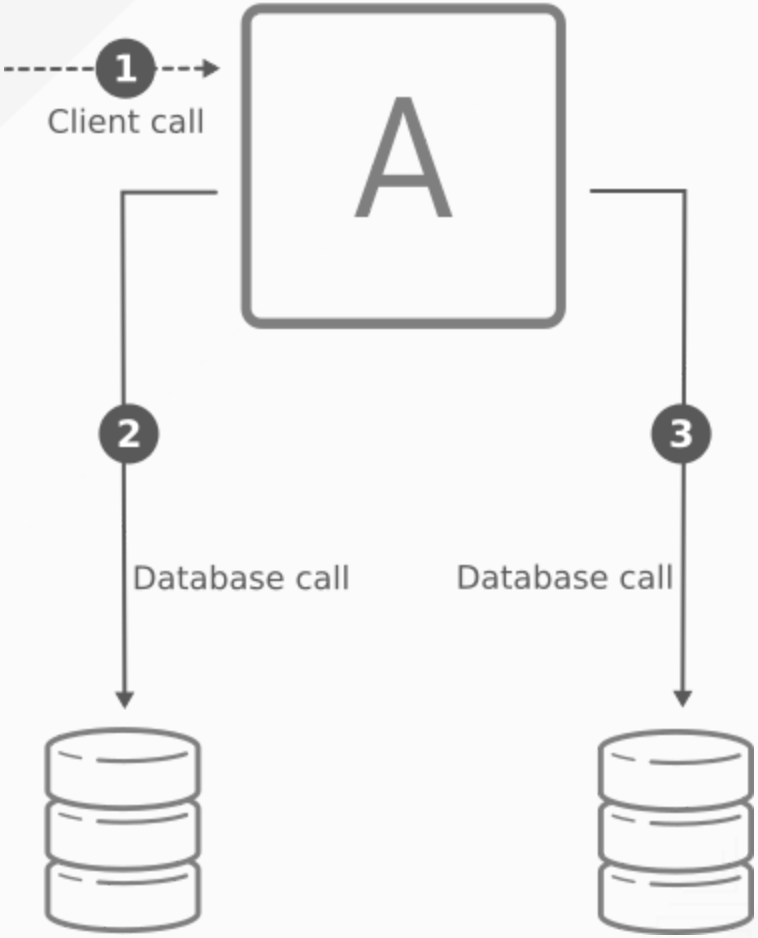
AGENDA

- what is transaction management
- microservices and transactions
- introduction to saga pattern
- Long Running Actions for MicroProfile
- DynFlow framework

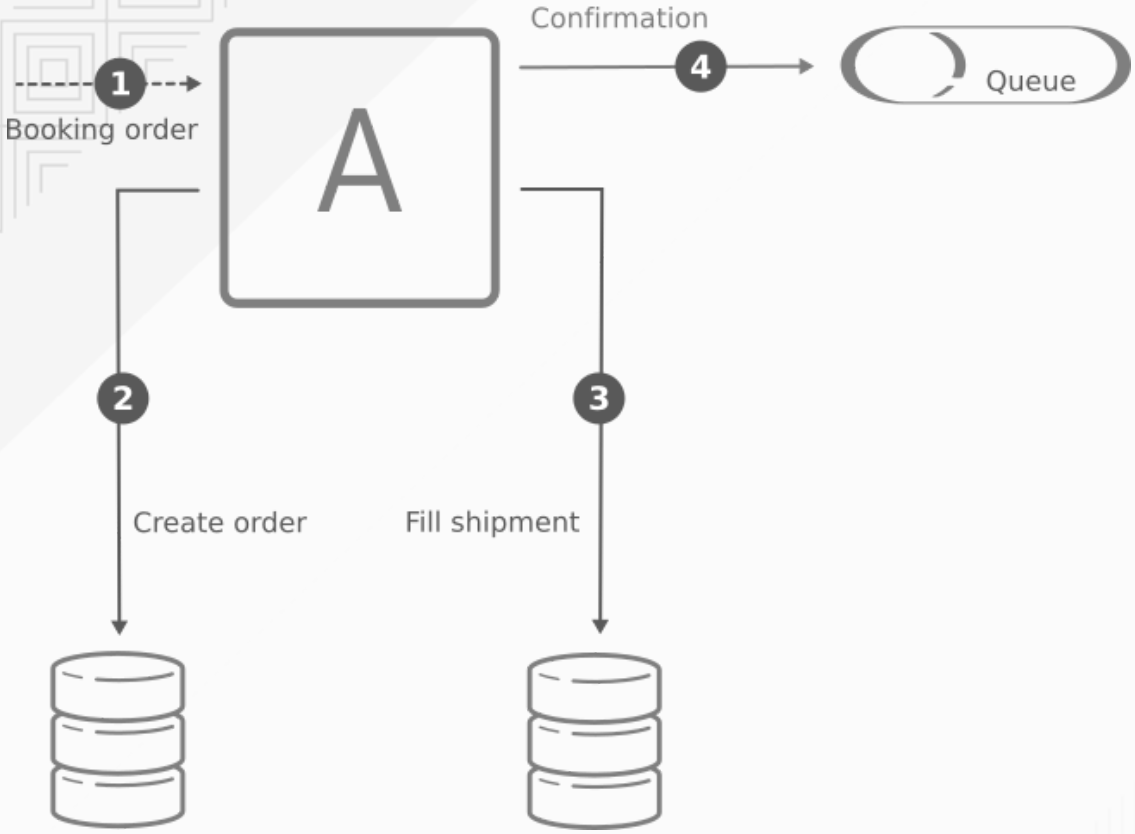
TRANSACTION

An atomic unit of work where all parts either finish with success or fail.

MONOLITHIC APPLICATION

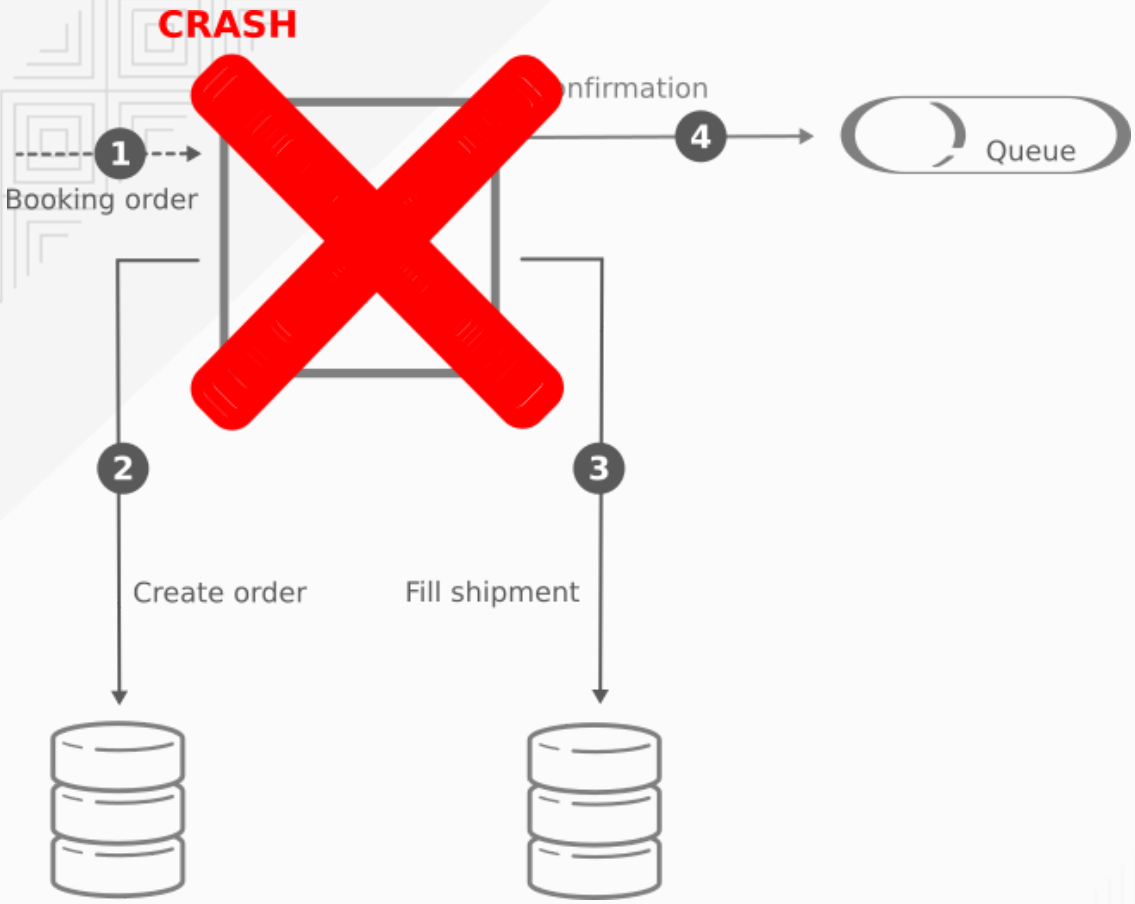


LET'S CREATE A BOOKING



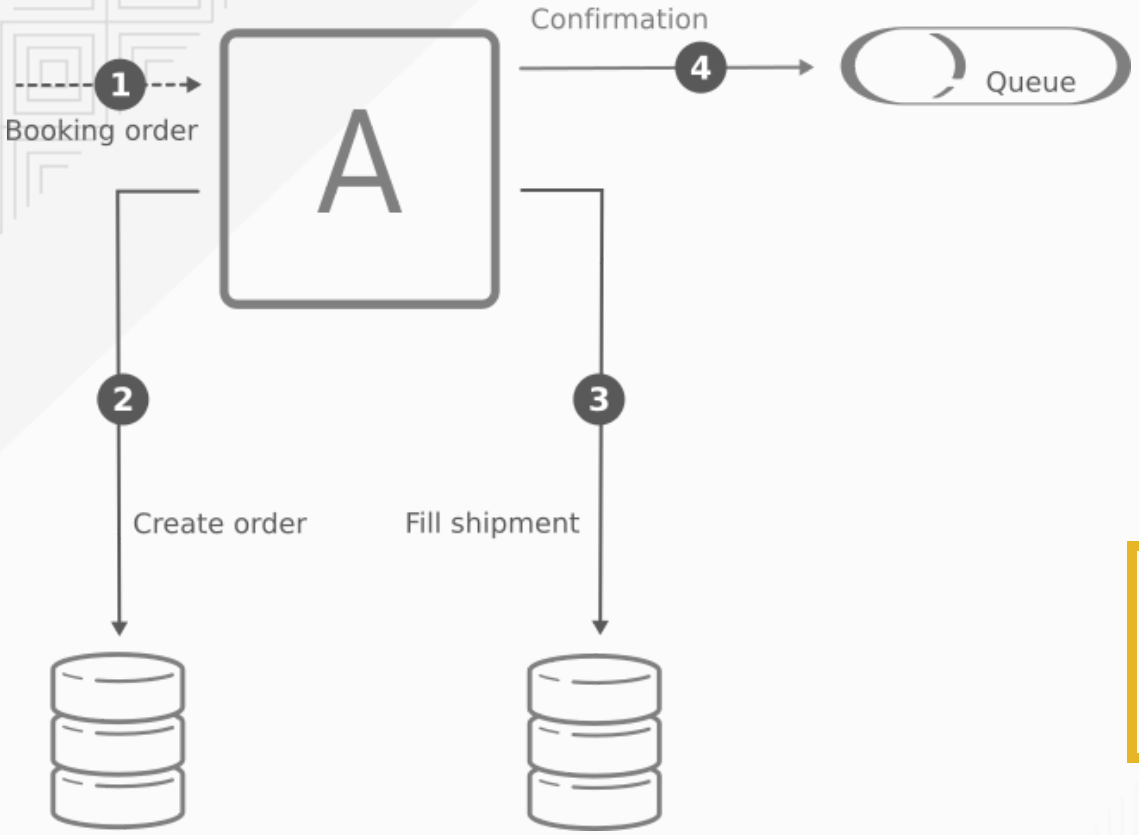
- Creating order
- Filling shipment
- Sending confirmation

...AND NOW WHAT?



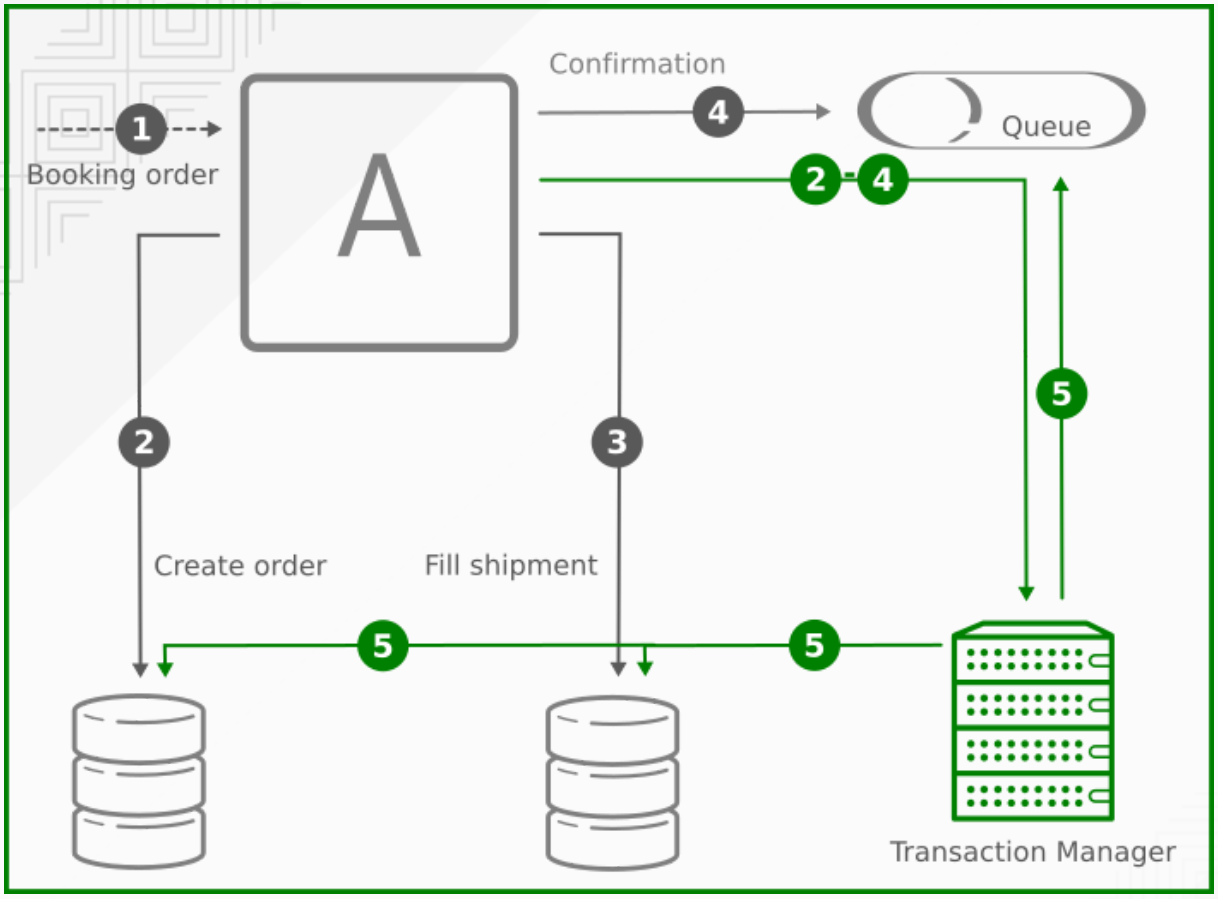
- Creating order
- Filling shipment
- Sending confirmation

A SINGLE UNIT OF WORK



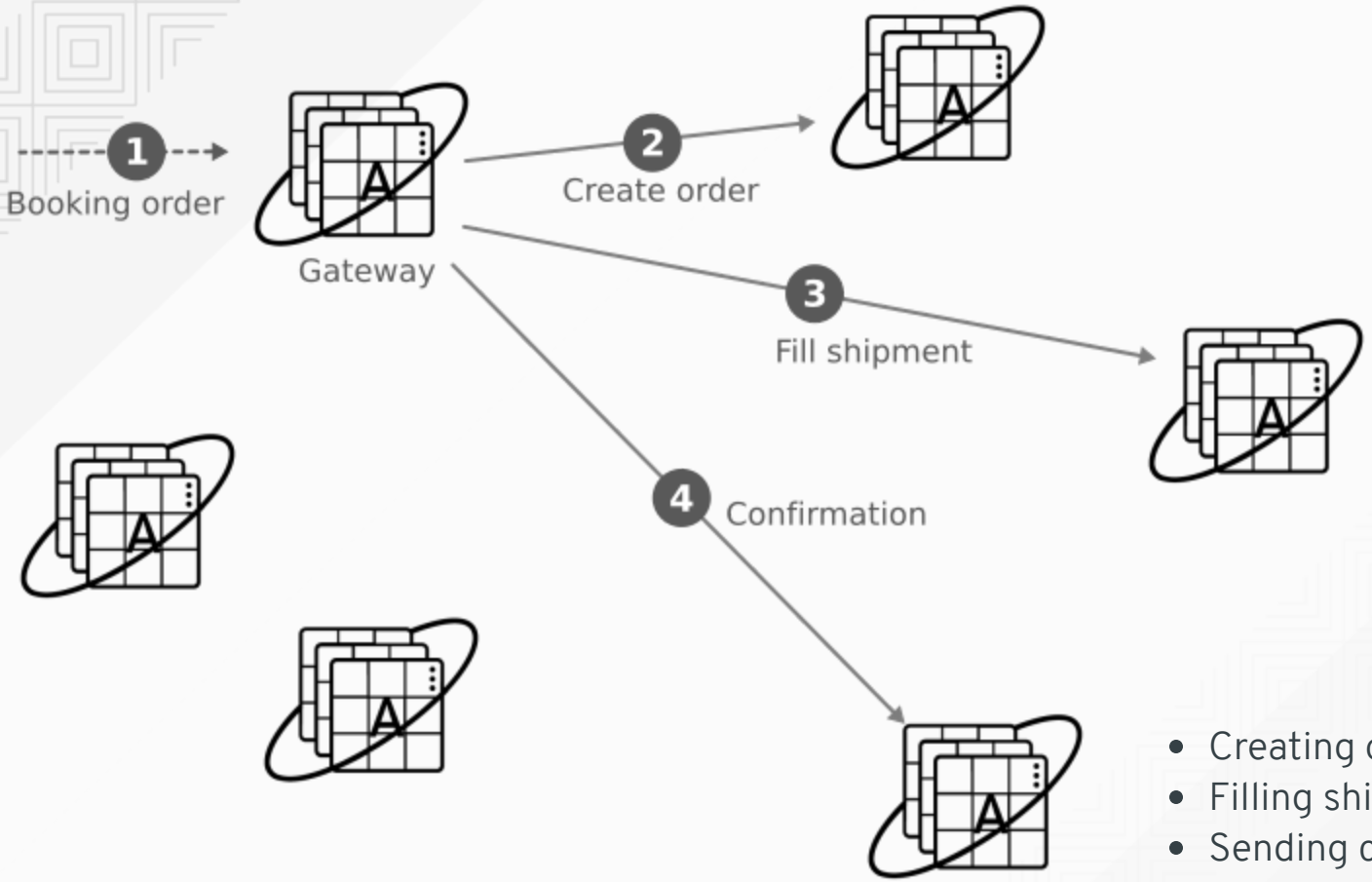
- Creating order
- Filling shipment
- Sending confirmation

ACID TRANSACTIONS TO RESCUE



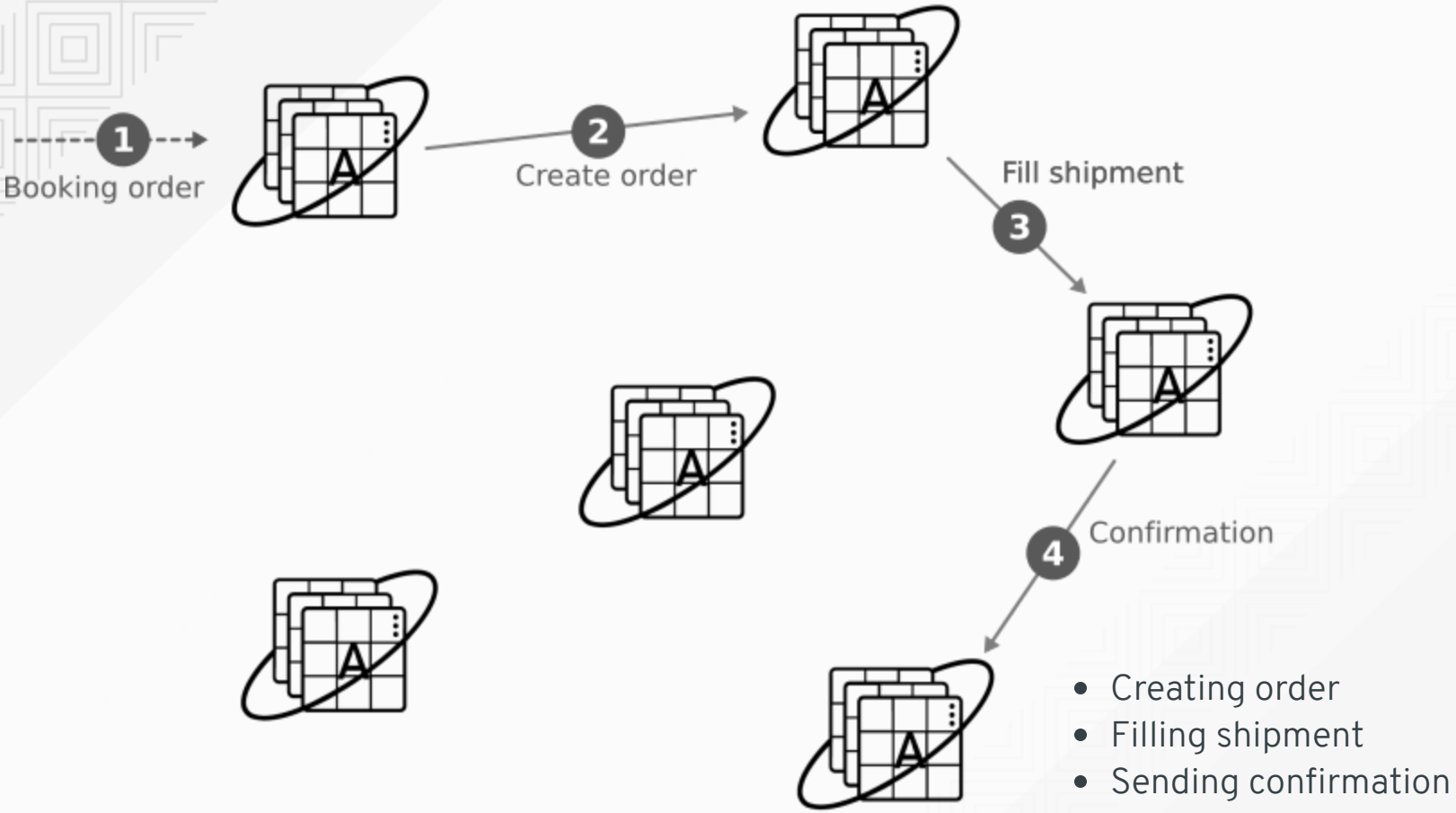
- Creating order
- Filling shipment
- Sending confirmation

WORLD OF MICROSERVICES



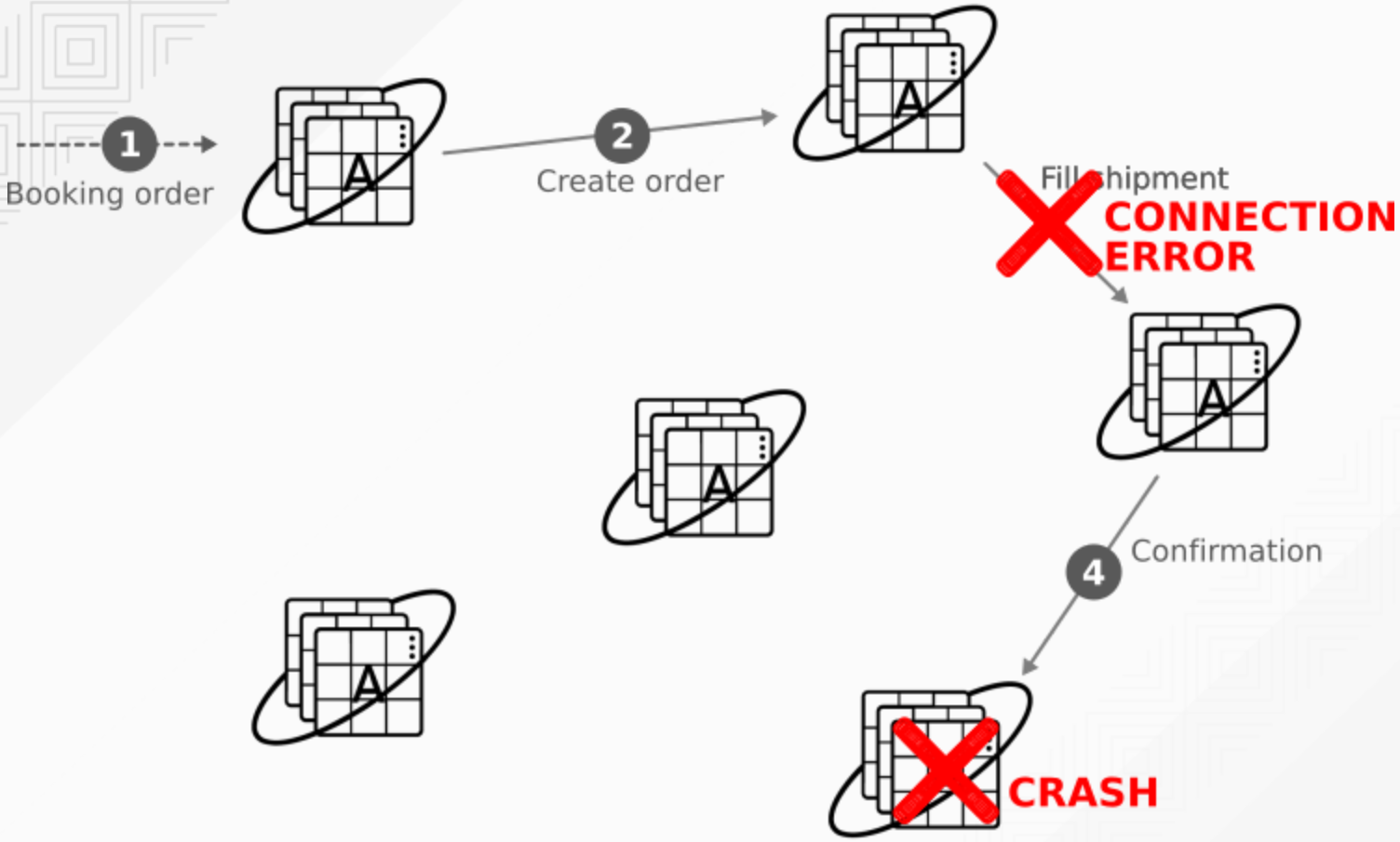
- Creating order
- Filling shipment
- Sending confirmation

WORLD OF MICROSERVICES



- Creating order
- Filling shipment
- Sending confirmation

FAILURES HAPPENS



Martin Kleppmann: Transactions: myths, surprises and opportunities

...AND NOW WHAT?

- just use ACID transactions

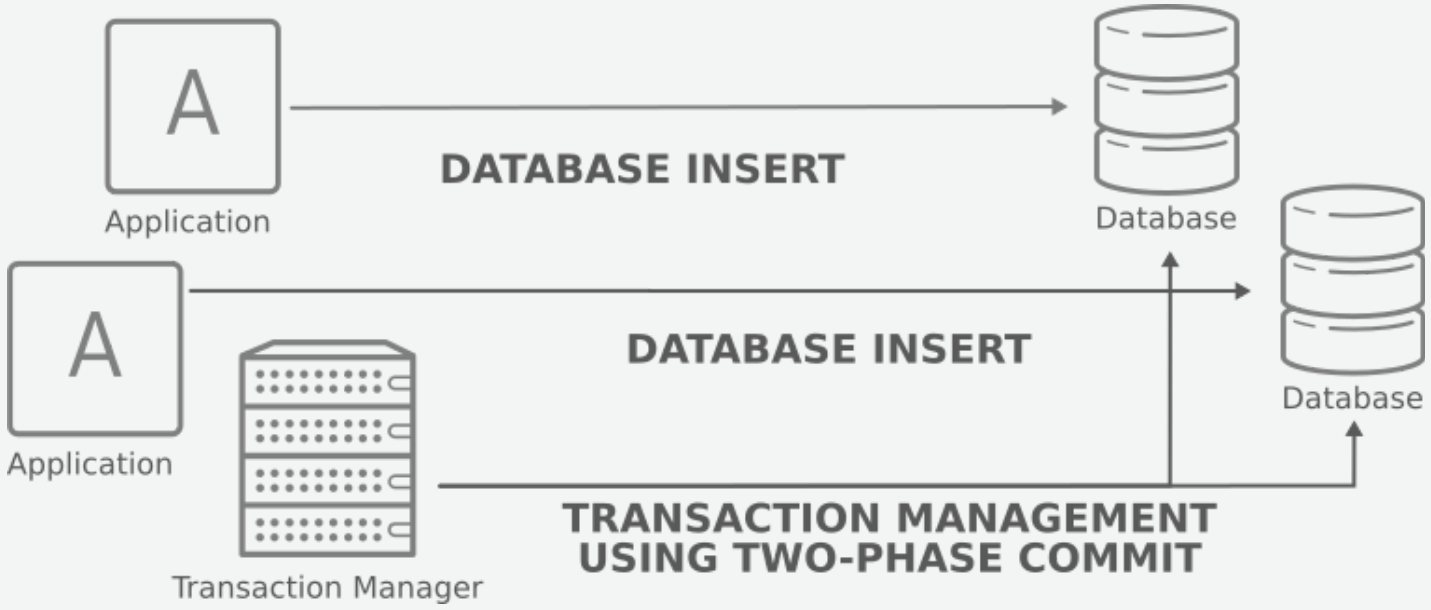
...AND NOW WHAT?

- just use ACID transactions
- but:
  - using locks

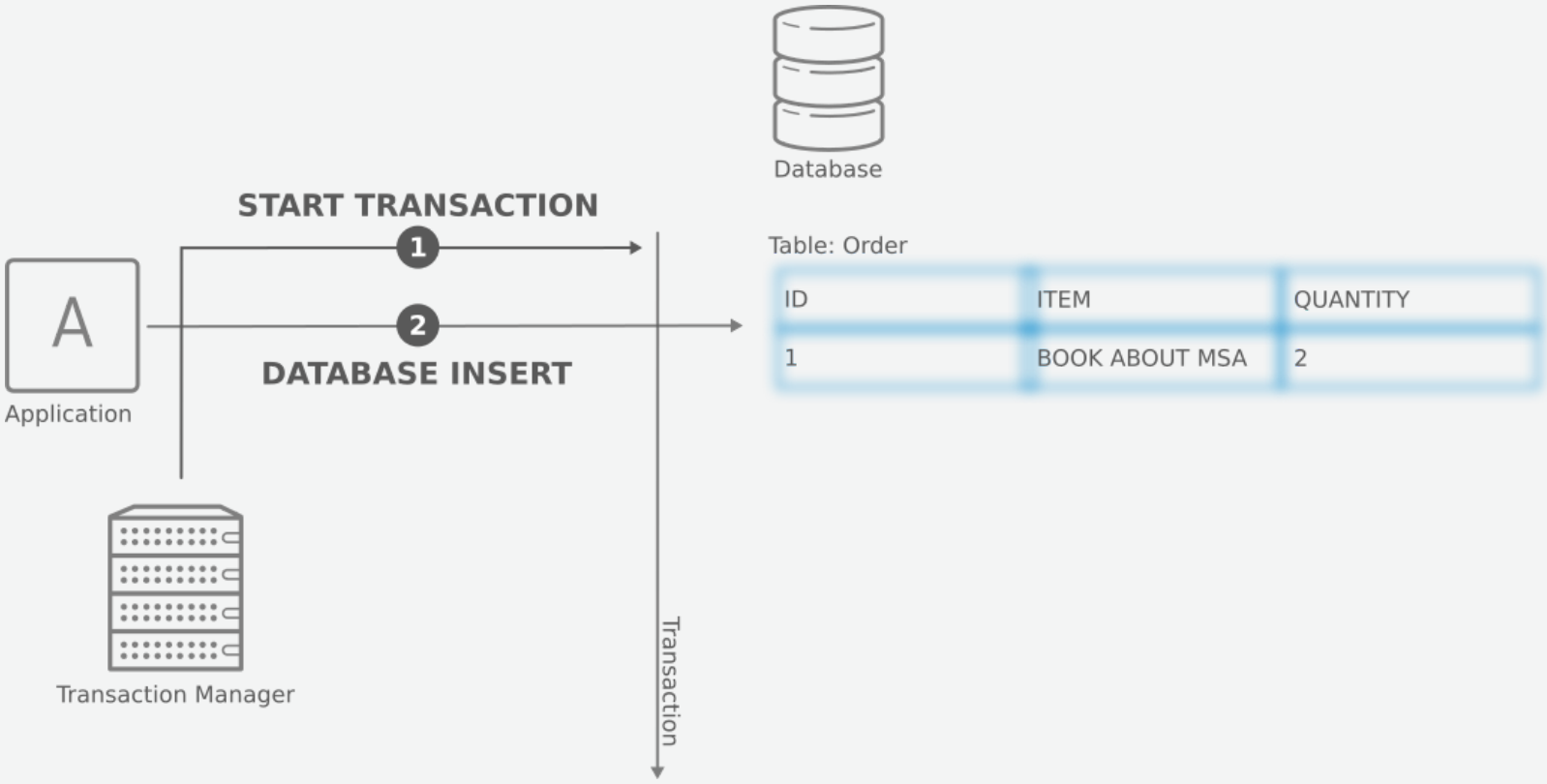
...AND NOW WHAT?

- just use ACID transactions
- but:
  - using locks
  - coupling microservices together

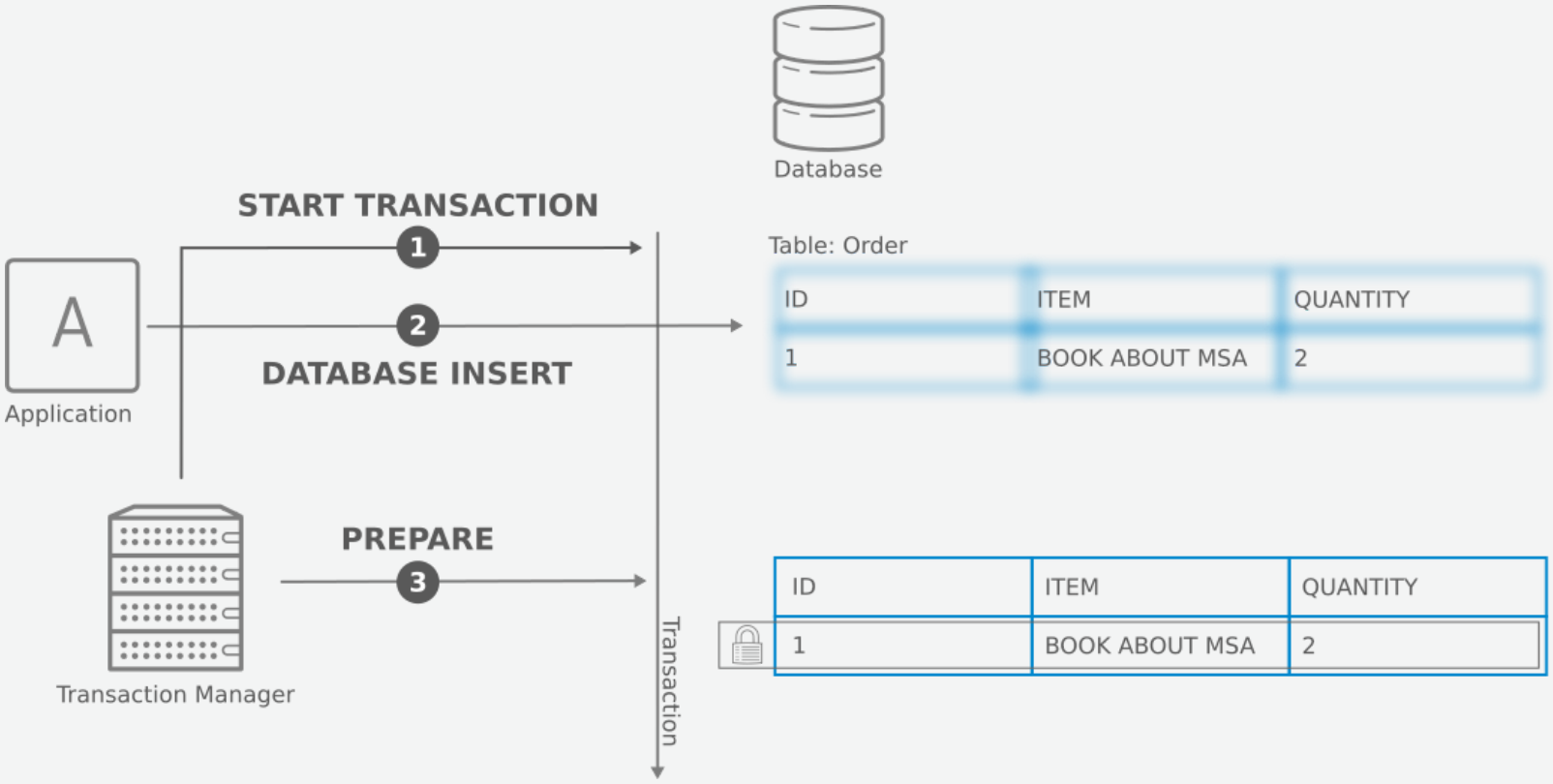
DISTRIBUTED XA TRANSACTION



ACID AND TWO-PHASE COMMIT



ACID AND TWO-PHASE COMMIT



ACID AND TWO-PHASE COMMIT

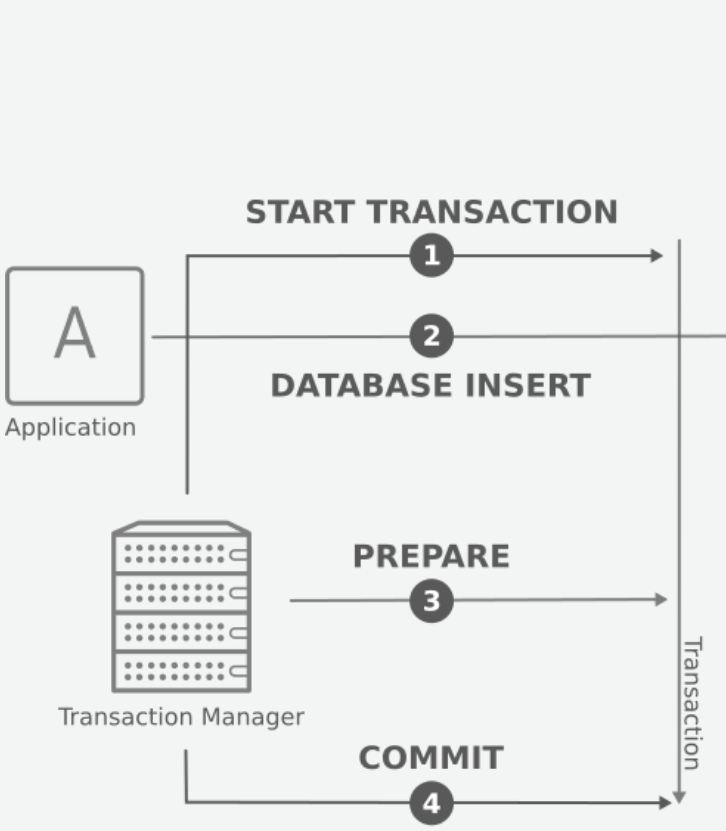


Table: Order

ID	ITEM	QUANTITY
1	BOOK ABOUT MSA	2

ID	ITEM	QUANTITY
1	BOOK ABOUT MSA	2

ID	ITEM	QUANTITY
1	BOOK ABOUT MSA	2

...AND NOW WHAT?

...AND NOW WHAT?

- rollback to monolithic approach

...AND NOW WHAT?

- rollback to monolithic approach
- but:

...AND NOW WHAT?

- rollback to monolithic approach
- but:
  - agility

...AND NOW WHAT?

- rollback to monolithic approach
- but:
  - agility
  - independence

...AND NOW WHAT?

- rollback to monolithic approach
- but:
  - agility
  - independence
  - scalability

...AND NOW WHAT?

- rollback to monolithic approach
- but:
  - agility
  - independence
  - scalability
  - easy to understand

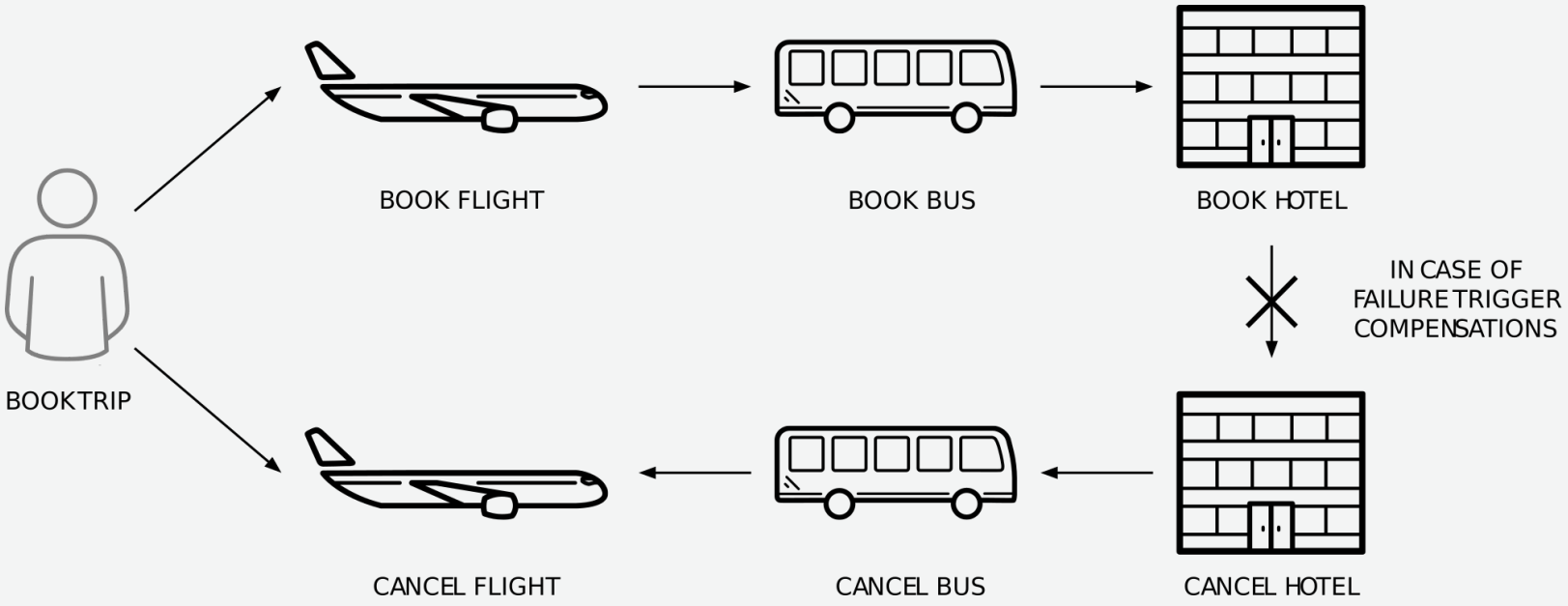
...AND NOW WHAT?

- rollback to monolithic approach
- but:
  - agility
  - independence
  - scalability
  - easy to understand
  - fault isolation

SAGA PATTERN

a distributed domain transaction

SAGA PATTERN



SAGA PATTERN - THE BASIC IDEA

SAGA PATTERN - THE BASIC IDEA

- break overall transaction into **smaller steps**

SAGA PATTERN - THE BASIC IDEA

- break overall transaction into **smaller steps**
- steps can be performed in atomic transactions internally

SAGA PATTERN - THE BASIC IDEA

- break overall transaction into **smaller steps**
- steps can be performed in atomic transactions internally
- saga ensures that either the overall transaction is **fully completed** or the changes are undone

SAGA PATTERN

SAGA PATTERN

- first published in 1987

SAGA PATTERN

- first published in 1987
- intended for long running transactions in databases

SAGA PATTERN

- first published in 1987
- intended for long running transactions in databases
- good fit for microservices nowadays

SAGA PATTERN

- first published in 1987
- intended for long running transactions in databases
- good fit for microservices nowadays
- two main approaches to saga

SAGA PATTERN

- first published in 1987
- intended for long running transactions in databases
- good fit for microservices nowadays
- two main approaches to saga
  - orchestration

SAGA PATTERN

- first published in 1987
- intended for long running transactions in databases
- good fit for microservices nowadays
- two main approaches to saga
  - orchestration
    - provides a good way of controlling the flow

SAGA PATTERN

- first published in 1987
- intended for long running transactions in databases
- good fit for microservices nowadays
- two main approaches to saga
  - orchestration
    - provides a good way of controlling the flow
    - an orchestrator tells participants what local transactions to execute

SAGA PATTERN

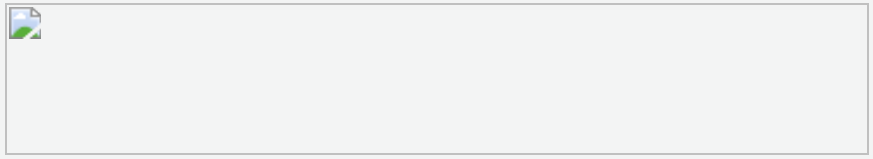
- first published in 1987
- intended for long running transactions in databases
- good fit for microservices nowadays
- two main approaches to saga
  - orchestration
    - provides a good way of controlling the flow
    - an orchestrator tells participants what local transactions to execute
  - choreography

SAGA PATTERN

- first published in 1987
- intended for long running transactions in databases
- good fit for microservices nowadays
- two main approaches to saga
  - orchestration
    - provides a good way of controlling the flow
    - an orchestrator tells participants what local transactions to execute
  - choreography
    - each local transaction publishes events that trigger local transaction in other services

LRA: LONG RUNNING ACTIONS

- Java based
- specification proposal for long running activities under Eclipse MicroProfile umbrella
  - https://github.com/eclipse/microprofile-lra
- defines LRA coordinator
- over HTTP, LRA context is passed in HTTP headers
- definition for REST style endpoints
- implementation in project Narayana.io



narayana

DYNFLOW

DYNFLOW

- workflow engine written in Ruby

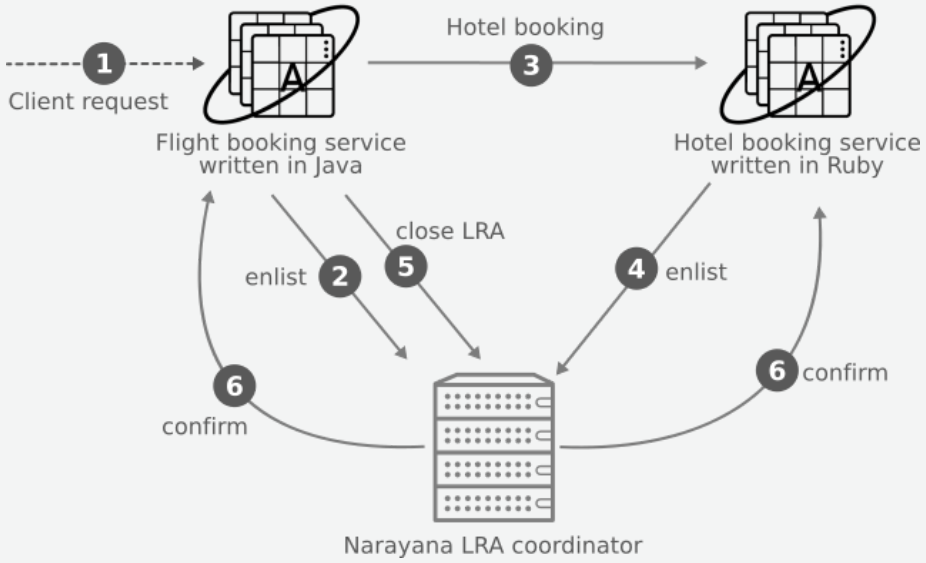
DYNFLOW

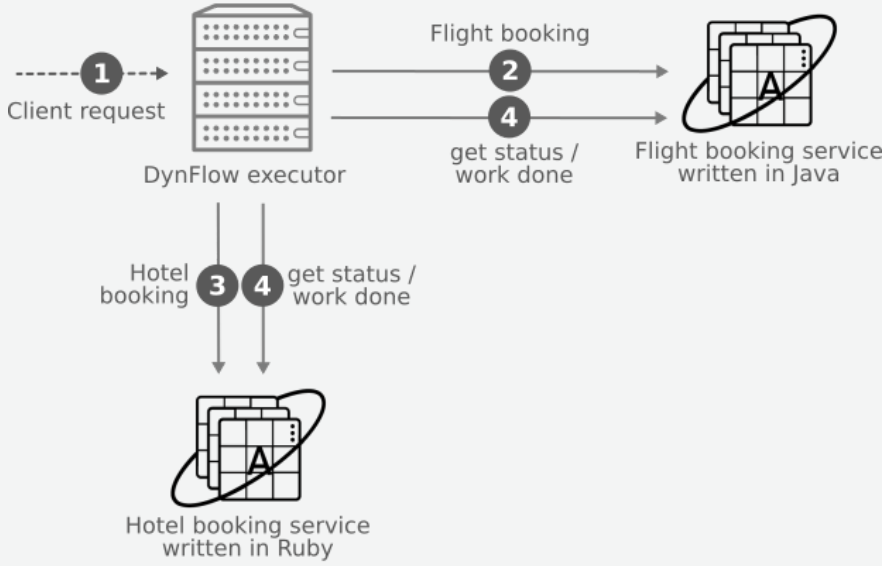
- workflow engine written in Ruby
- currently in use by the Foreman project

DYNFLOW

- workflow engine written in Ruby
- currently in use by the Foreman project
- can do all sorts of stuff out of scope of this talk
  - running independent steps concurrently
  - polling external tasks
  - and much more
- support for Sagas in the form of rescue strategy

LRA VS. DYNFLOW

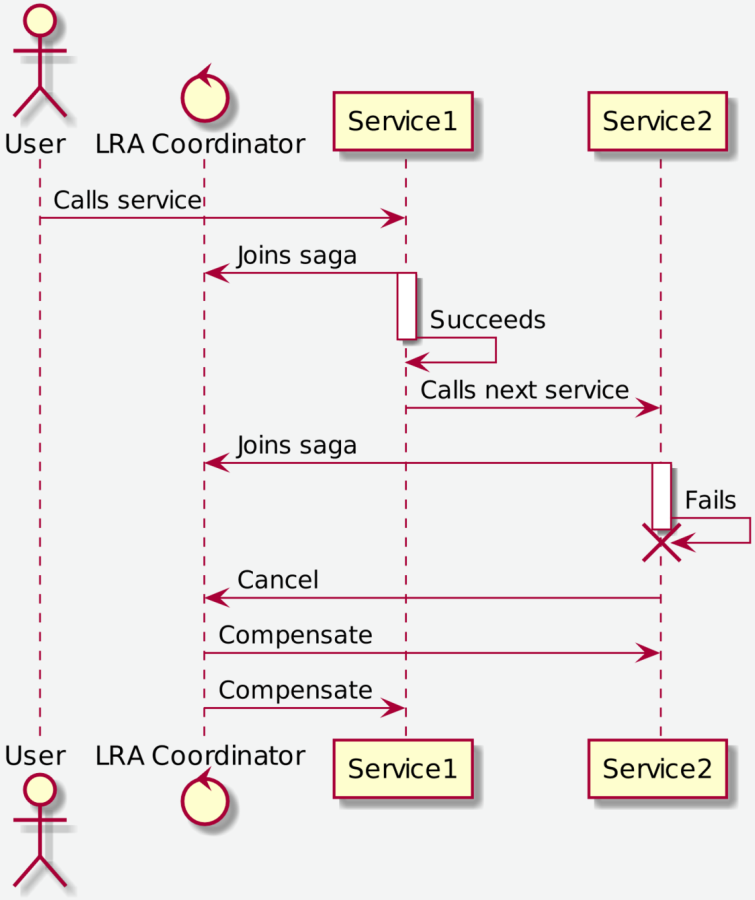




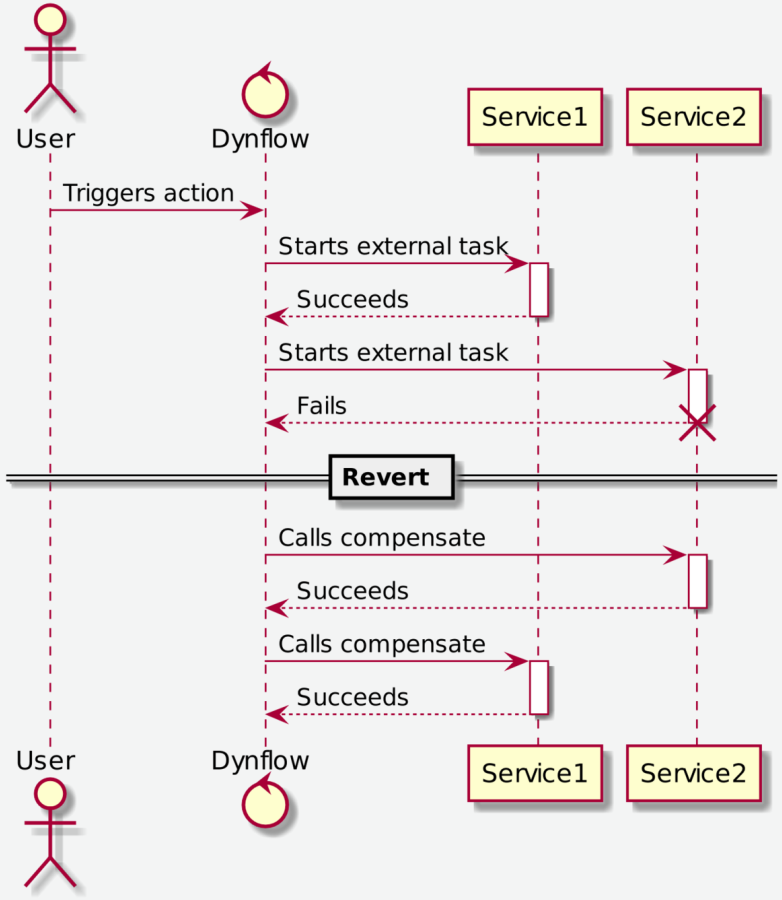
Long Running Actions

DynFlow

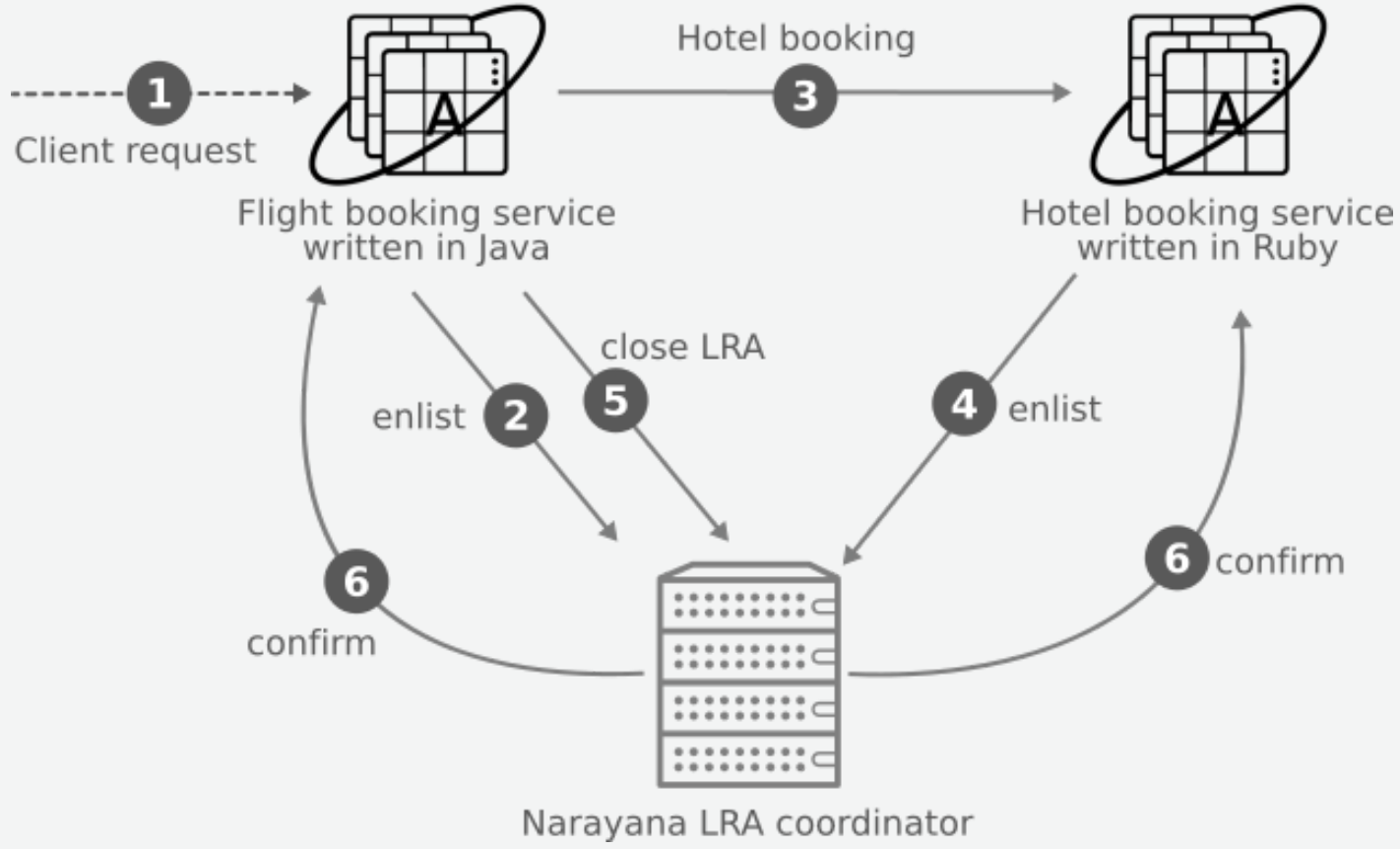
LRA VS DYNFLOW



VS.



LONG RUNNING ACTIONS



LONG RUNNING ACTIONS

```
@LRA
@NestedLRA

@Complete
@Compensate

@Leave
@Status
```

```
org.eclipse.microprofile.lra.client.LRAClien
```

```
startLRA()


closeLRA()
cancelLRA()

leaveLRA()
getStatus()

getAllLRAs()
getActiveLRAs()
getRecoveringLRAs()
```

DYNFLOW BUILDING BLOCKS

DYNFLOW BUILDING BLOCKS

- Actions
  - have three phases - plan, run and finalize
  - can be composed

DYNFLOW BUILDING BLOCKS

- Actions
  - have three phases - plan, run and finalize
  - can be composed
- Execution plans
  - are generated by planning actions
  - in our case a scope for transaction

DYNFLOW BUILDING BLOCKS

- Actions
  - have three phases - plan, run and finalize
  - can be composed
- Execution plans
  - are generated by planning actions
  - in our case a scope for transaction
- Steps
  - units of work

ACTION EXAMPLE

```ruby
class BookHotel < ::Dynflow::Action
  include REST

  def run
    output[:response] = post_rest(input[:url])
  end
end

class BookTrip < ::Dynflow::Action
  def plan
    5.times { plan_action BookHotel, :url => 'http://hotel.california/book' }
  end
end
```

SAGAS IN DYNFLOW

SAGAS IN DYNFLOW

- For an execution plan we know how all its steps finished

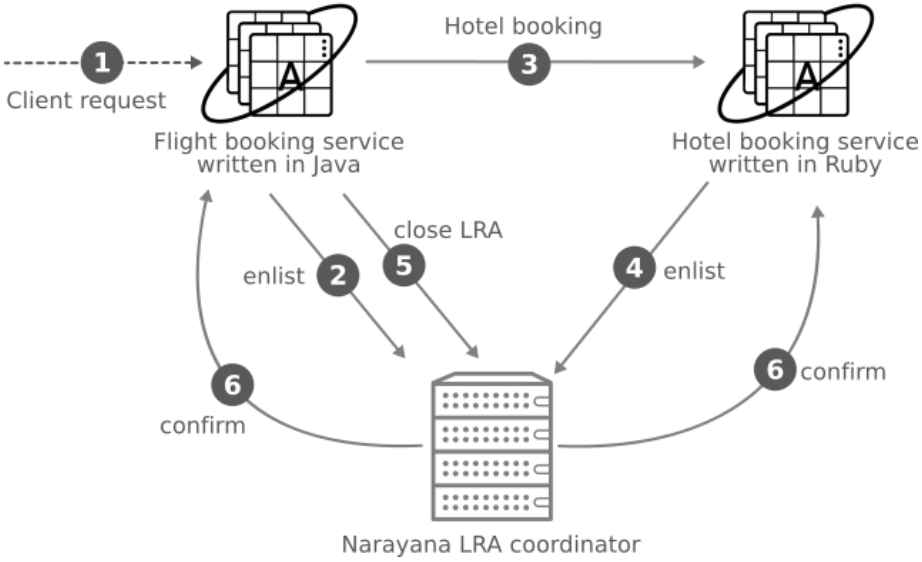
SAGAS IN DYNFLOW

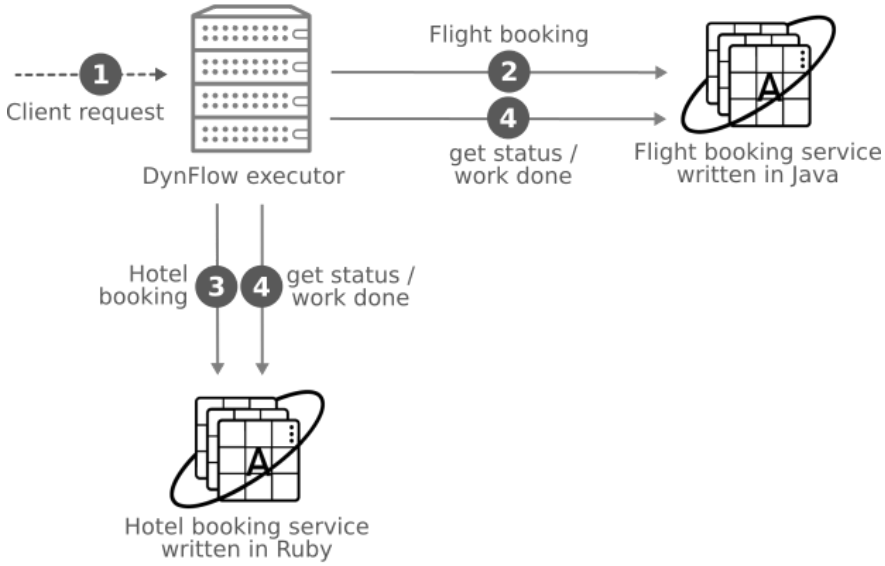
- For an execution plan we know how all its steps finished
- If we know how to undo every single step, we can undo the entire execution plan

ROLLBACKS IN DYNFLOW

```ruby
class BookHotel < ::Dynflow::Action
  include ::Dynflow::Action::Revertible
  include REST

  def run
    output[:response] = post_rest(input[:url], :parse_json => true)
  end

  def revert_run
    id = original_output.fetch(:response, {})[:id]
    post_rest(original_input[:url] + "/#{id}/compensate", :parse_json => false) if id
  end
end

class BookTrip < ::Dynflow::Action
  include ::Dynflow::Action::Revertible

  def plan
    5.times { plan_action BookHotel, :url => 'http://hotel.california/book' }
  end
end
```

DEMO


1
Client request
Flight booking service
written in Java
Hotel booking
3
Hotel booking service
written in Ruby
enlist
2
close LRA
5
4
enlist
6
confirm
6
confirm
Narayana LRA coordinator


1
Client request
DynFlow executor
Flight booking
2
4
get status /
work done
Flight booking service
written in Java
Hotel
booking
3
4
get status /
work done
Hotel booking service
written in Ruby

SUMMARY

SUMMARY

- Sagas are great solution for transactions in microservice deployments

SUMMARY

- Sagas are great solution for transactions in microservice deployments
  - if you're willing to loosen your requirements and go from strict atomicity to eventual consistency

QUESTIONS

LINKS

LINKS

- MicroProfile LRA specification: https://github.com/eclipse/microprofile-lra
- Community gitter: https://gitter.im/eclipse/microprofile-lra
- Blog posts: Narayana LRA: implementation of saga transactions, Saga implementations comparison

LINKS

- MicroProfile LRA specification: https://github.com/eclipse/microprofile-lra
- Community gitter: https://gitter.im/eclipse/microprofile-lra
- Blog posts: Narayana LRA: implementation of saga transactions, Saga implementations comparison
- Link to LRA demo: https://github.com/ochaloup/devconf2019-lra

- Dynflow: https://github.com/dynflow/dynflow
- Dynflow documentation: https://dynflow.github.io

- Saga paper: https://www.cs.cornell.edu/andru/cs711/2002fa/reading/sagas.pdf

Red Hat

THANK YOU!