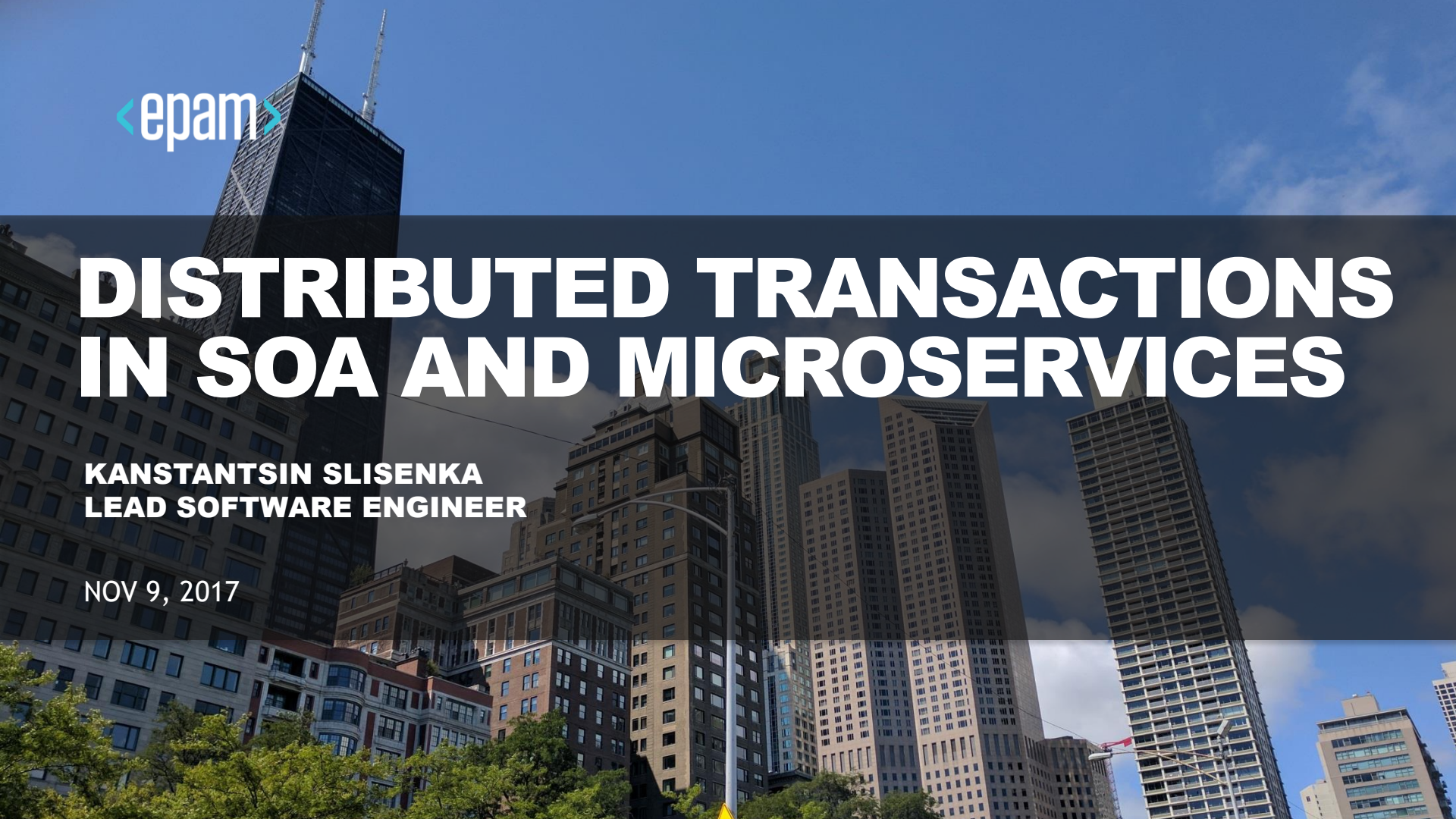


DISTRIBUTED TRANSACTIONS IN SOA AND MICROSERVICES

KANSTANTSIN SLISENKA
LEAD SOFTWARE ENGINEER

NOV 9, 2017

Kanstantsin Slisenka

Java Team Lead

Financial services, trading solutions

Speaker at: Minsk Java Tech Talks, IT Week, SEC Online, Java Professionals

github.com/kslisenko

kanstantsin_slisenka@epam.com

AGENDA

1. Local transactions

2. Distributed transactions

3. Compensations: SAGA pattern

4. Live demo

WHY DO WE NEED TRANSACTIONS?

WHY DO WE NEED TRANSACTIONS?

Data must be in known state anytime

New order created → Payment received → Product shipped

Application

Business workflows

Infrastructure

Transactions in file systems, databases,
message brokers, caches, application servers

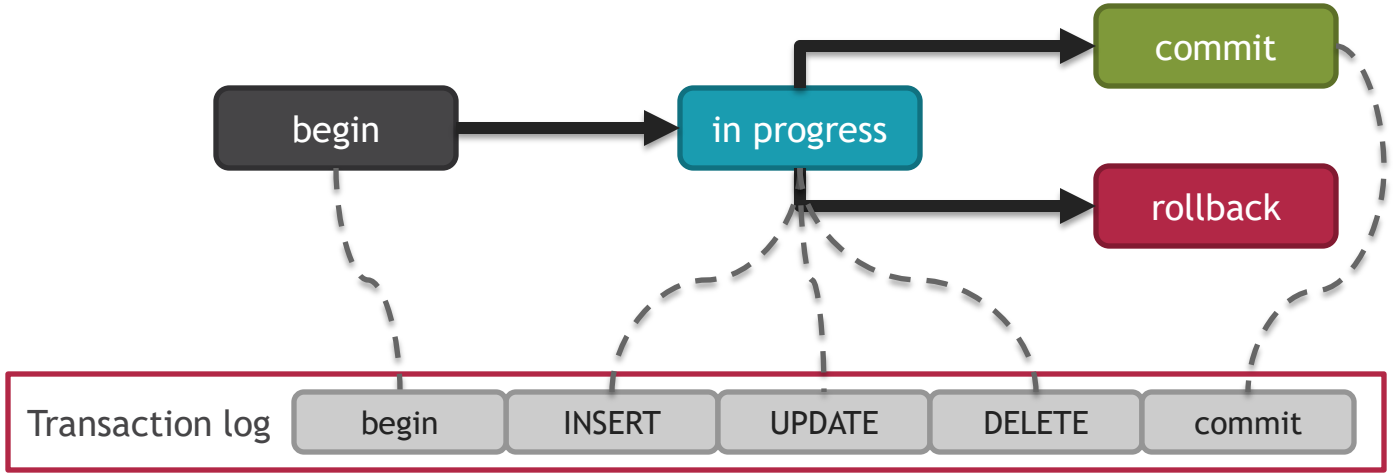
HOW IT WORKS

Transaction processing is based on **logging of**

states and

transitions

DATABASE TRANSACTION AS STATE MACHINE



- Always in known state
- States and transitions are logged
- Log can be used for recovery

ACID TRANSACTION GUARANTEES

Atomicity

All or nothing

Write-ahead log

ACID TRANSACTION GUARANTEES

Atomicity

All or nothing

Write-ahead log

Consistency

Data always in valid state

Constraints

ACID TRANSACTION GUARANTEES

Atomicity

All or nothing

Write-ahead log

Consistency

Data always in valid state

Constraints

Isolation

Visibility of concurrent actions

Locking

ACID TRANSACTION GUARANTEES

Atomicity

All or nothing

Write-ahead log

Consistency

Data always in valid state

Constraints

Isolation

Visibility of concurrent actions

Locking

Durability

Changes become permanent

Write-ahead log

Transactions guarantee the data to always be in valid state

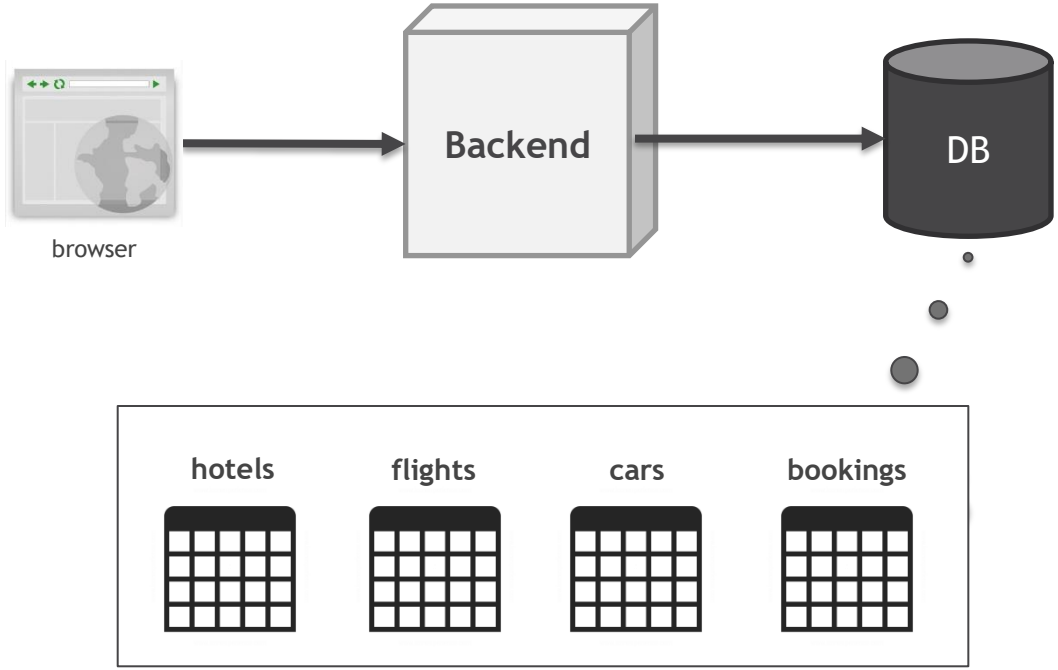
HOW IT WORKS?


EXAMPLE
TRAVEL BOOKING

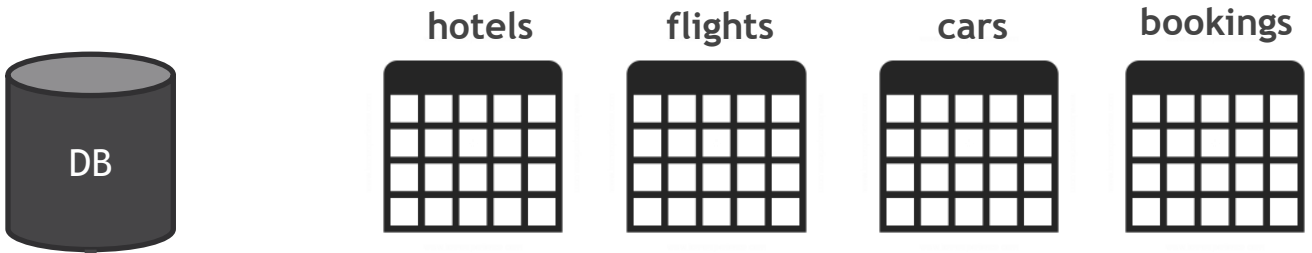
Book trip

1. Book hotel
2. Book flight
3. Hire rental car
4. Record booking

TRAVEL BOOKING SYSTEM



TRAVEL BOOKING TRANSACTION

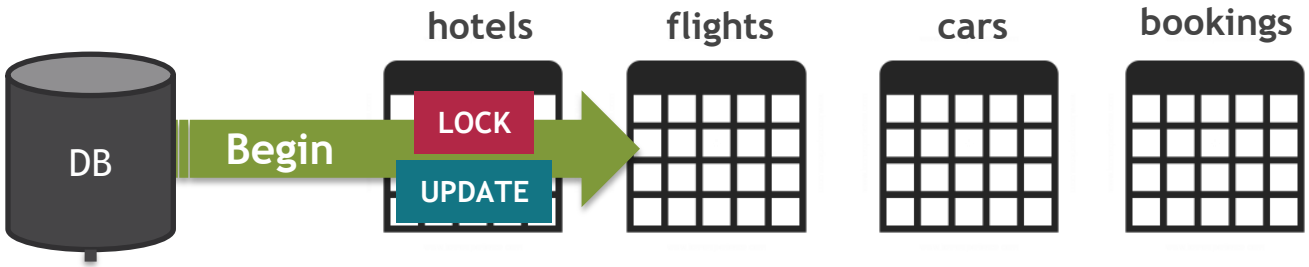


Action	Rollback action

TRAVEL BOOKING TRANSACTION

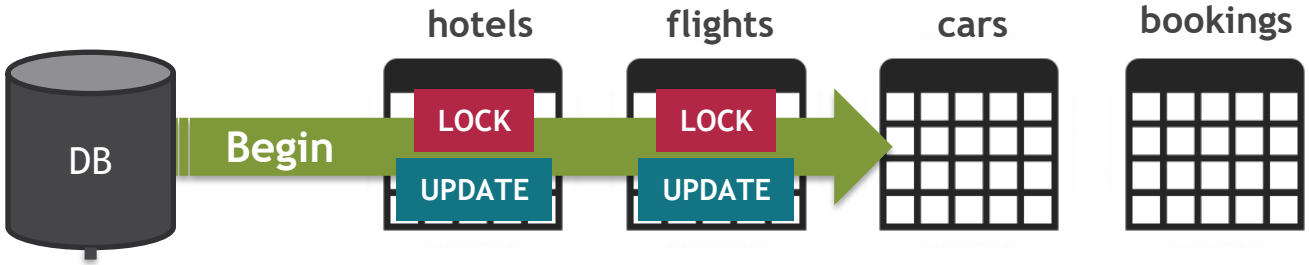
Action	Rollback action
`begin`	

TRAVEL BOOKING TRANSACTION



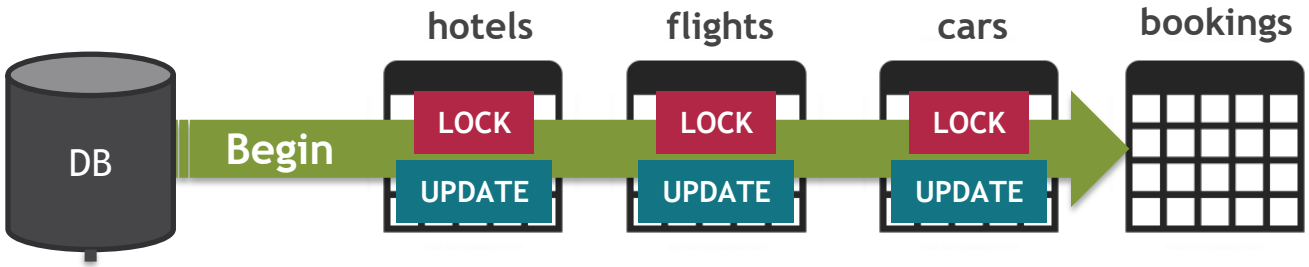
Action	Rollback action
begin	
room_id=15 status=booked, traveler=Kostia	room_id=15 status=free, traveler=null

TRAVEL BOOKING TRANSACTION



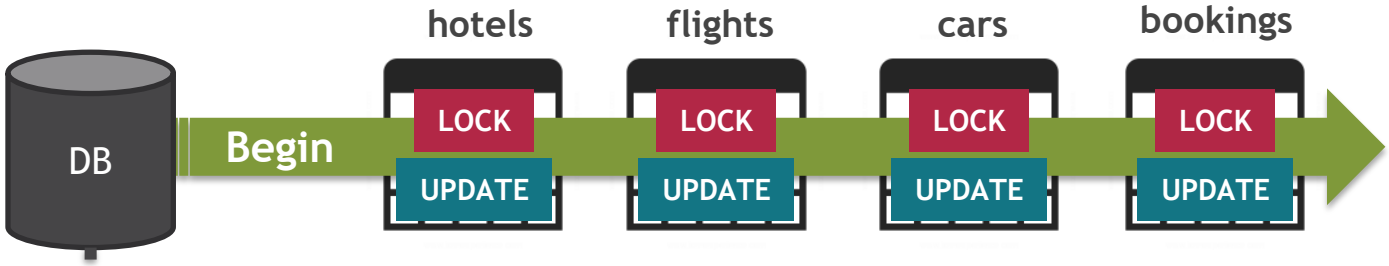
Action	Rollback action
begin	
room_id=15 status=booked, traveler=Kostia	room_id=15 status=free, traveler=null
seat_id=17c status=booked, traveler=Kostia	seat_id=17c status=free, traveler=null

TRAVEL BOOKING TRANSACTION



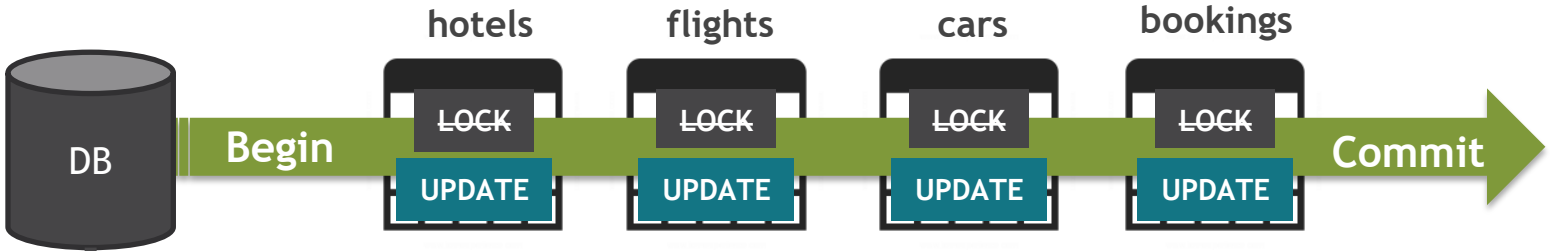
Action	Rollback action
`begin`	
`room_id=15` `status=booked, traveler=Kostia`	`room_id=15` `status=free, traveler=null`
`seat_id=17c` `status=booked, traveler=Kostia`	`seat_id=17c` `status=free, traveler=null`
`car_num=1234AA7` `status=booked, traveler=Kostia`	`car_num=1234AA7` `status=free, traveler=null`

TRAVEL BOOKING TRANSACTION



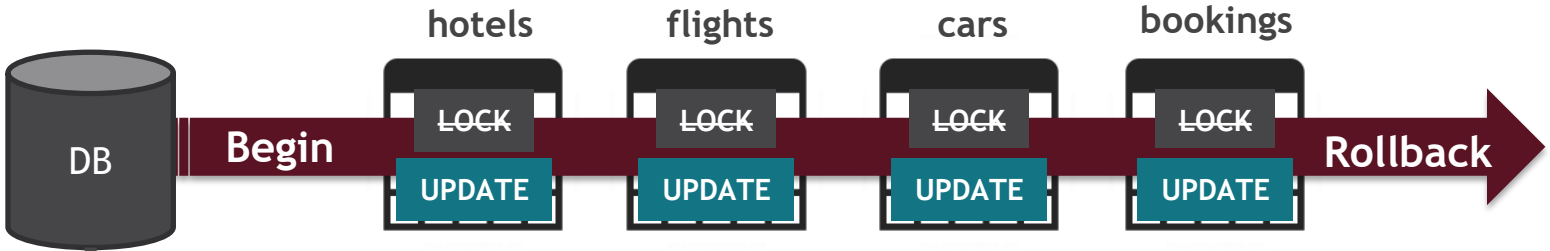
Action	Rollback action
begin	
room_id=15 status=booked, traveler=Kostia	room_id=15 status=free, traveler=null
seat_id=17c status=booked, traveler=Kostia	seat_id=17c status=free, traveler=null
car_num=1234AA7 status=booked, traveler=Kostia	car_num=1234AA7 status=free, traveler=null
INSERT traveler=Kostia, room_id=15, seat_id=17c, car_num=1234AA7	DELETE traveler=Kostia, room_id=15, seat_id=17c, car_num=1234AA7

TRAVEL BOOKING TRANSACTION



Action	~~Rollback action~~
begin	
room_id=15 status=booked, traveler=Kostia	~~room_id=15~~ ~~status=free, traveler=null~~
seat_id=17c status=booked, traveler=Kostia	~~seat_id=17c~~ ~~status=free, traveler=null~~
car_num=1234AA7 status=booked, traveler=Kostia	~~car_num=1234AA7~~ ~~status=free, traveler=null~~
INSERT traveler=Kostia, room_id=15, seat_id=17c, car_num=1234AA7	~~DELETE traveler=Kostia, room_id=15,~~ ~~seat_id=17c, car_num=1234AA7~~
commit	

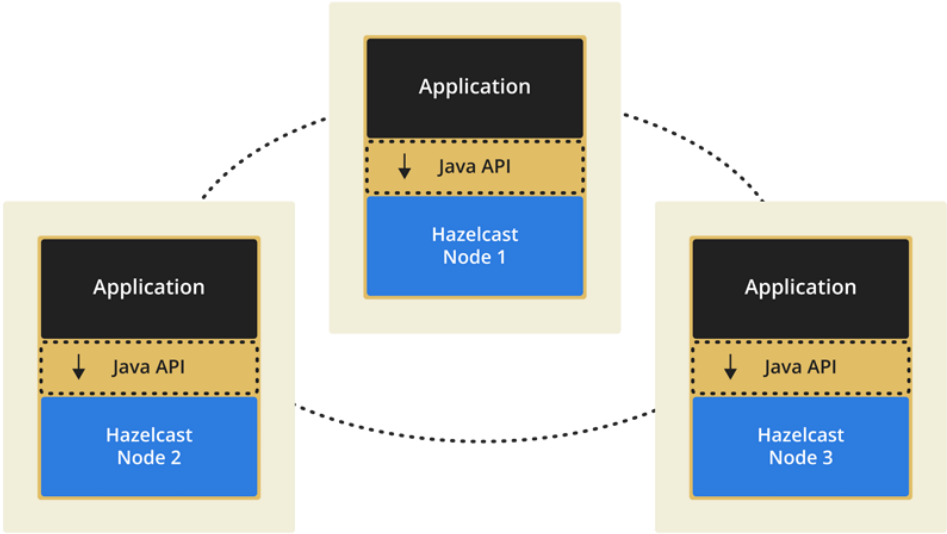
TRAVEL BOOKING TRANSACTION



Action	Rollback action
begin	
room_id=15 status=booked, traveler=Kostia	room_id=15 status=free, traveler=null
seat_id=17c status=booked, traveler=Kostia	seat_id=17c status=free, traveler=null
car_num=1234AA7 status=booked, traveler=Kostia	car_num=1234AA7 status=free, traveler=null
INSERT traveler=Kostia, room_id=15, seat_id=17c, car_num=1234AA7	DELETE traveler=Kostia, room_id=15, seat_id=17c, car_num=1234AA7
rollback	

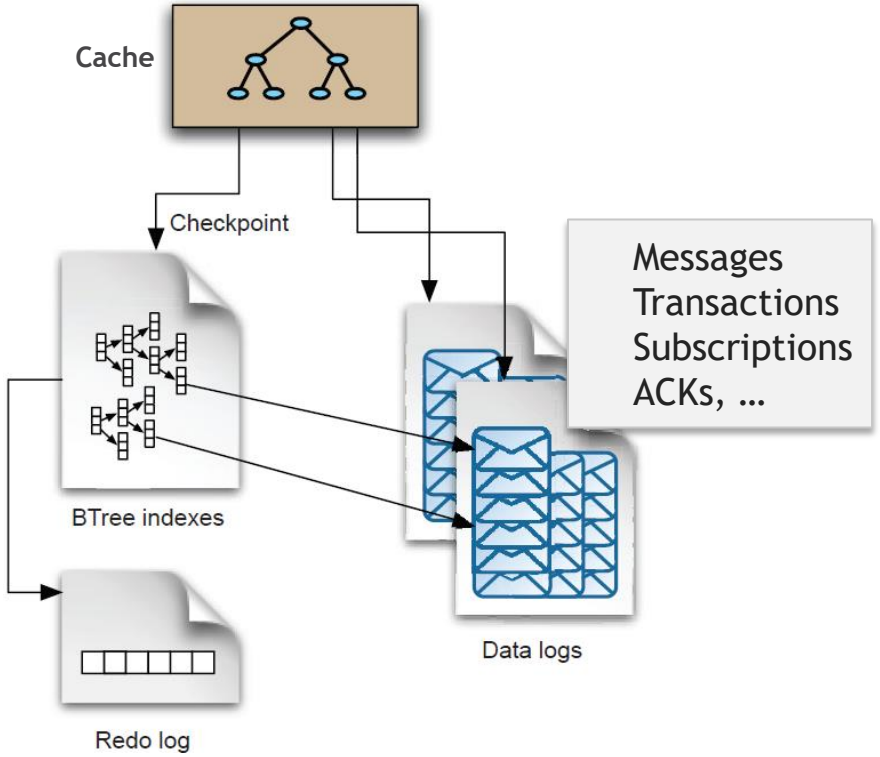
TRANSACTIONS ARE NOT ONLY IN DATABASES

- Isolation levels
- Locking
- Commit logs
- Transactions
  - *ONE and TWO phase*



ActiveMQ

- Transacted sessions
- JMS attributes
  - *JMSXConsumerTXID*
  - *JMSXProducerTXID*



AGENDA

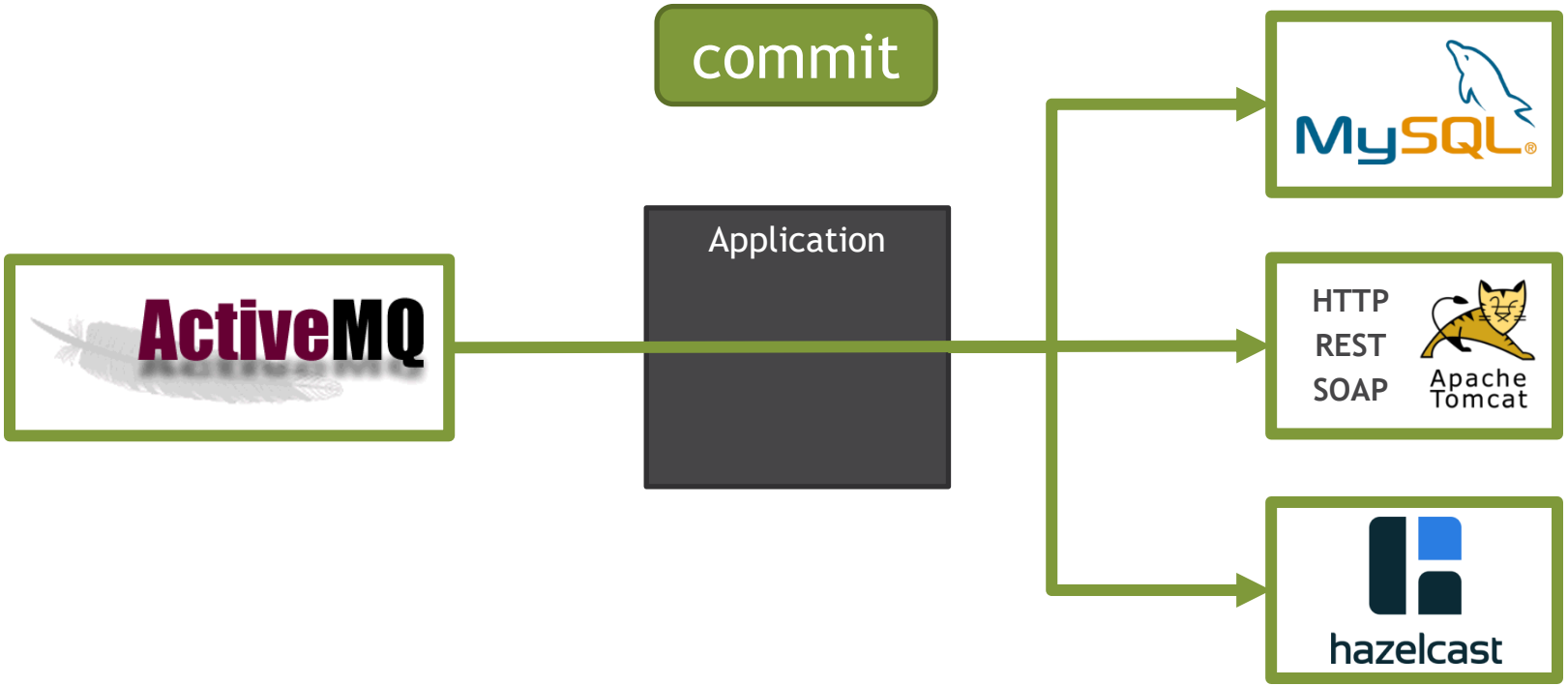
1. ~~Local transactions~~

2. Distributed transactions

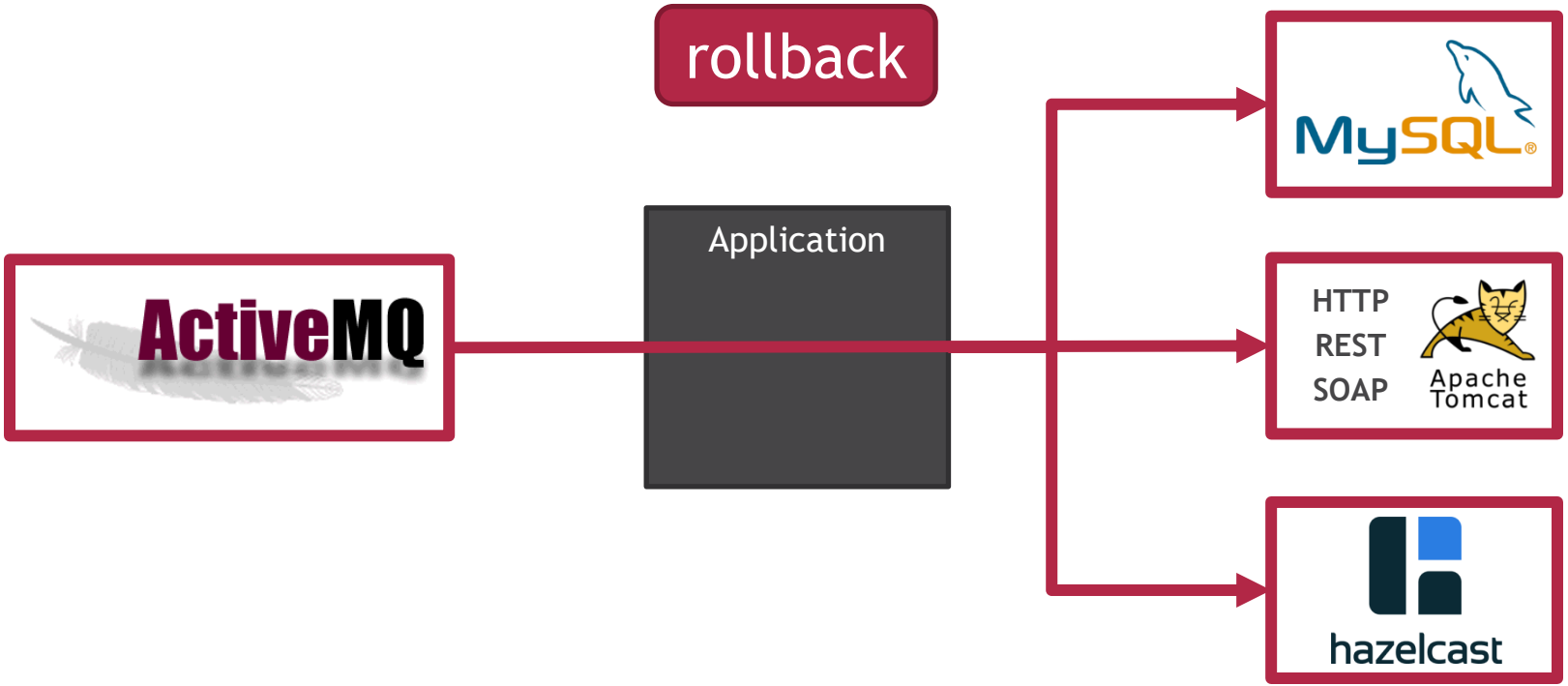
3. Compensations: SAGA pattern

4. Live demo

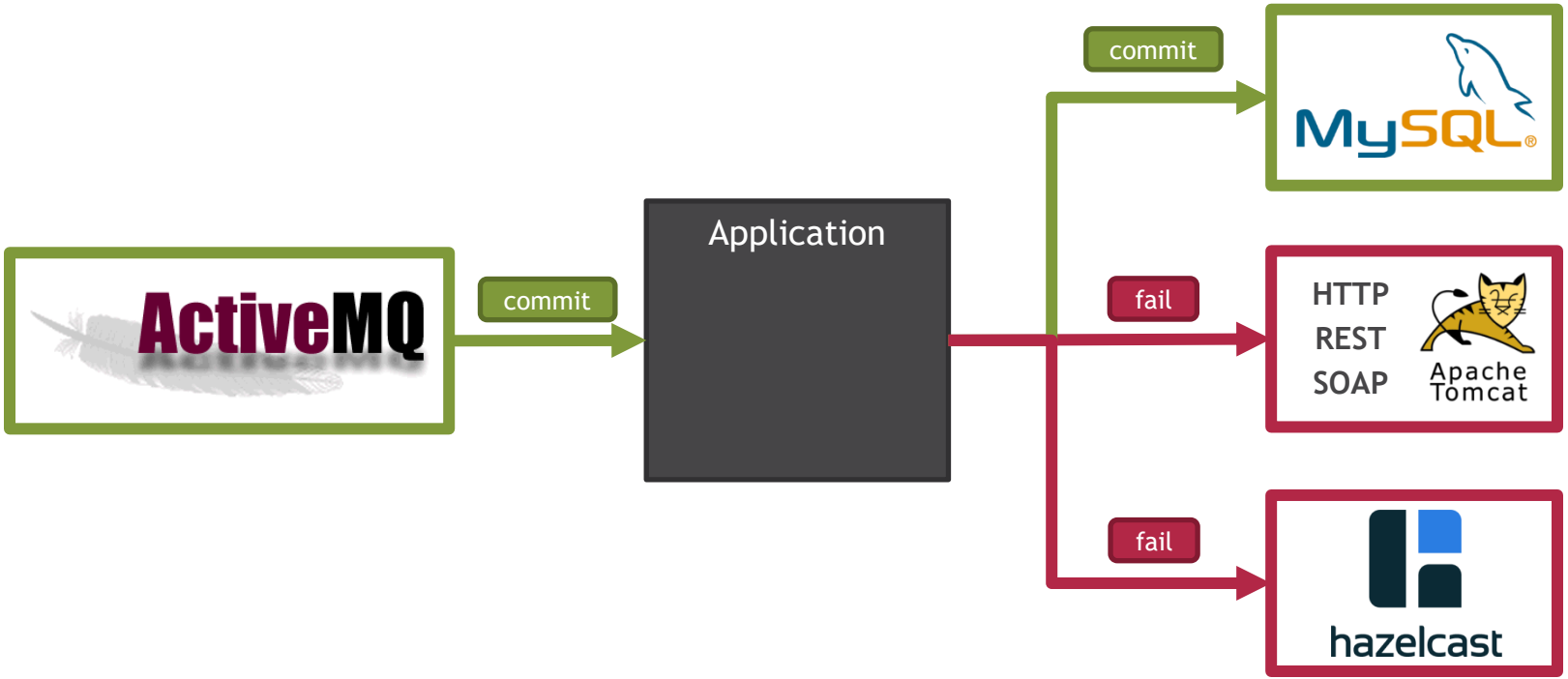
WE WANT THIS


commit
Application
ActiveMQ
MySQL
HTTP
REST
SOAP
Apache Tomcat
hazelcast

OR THIS

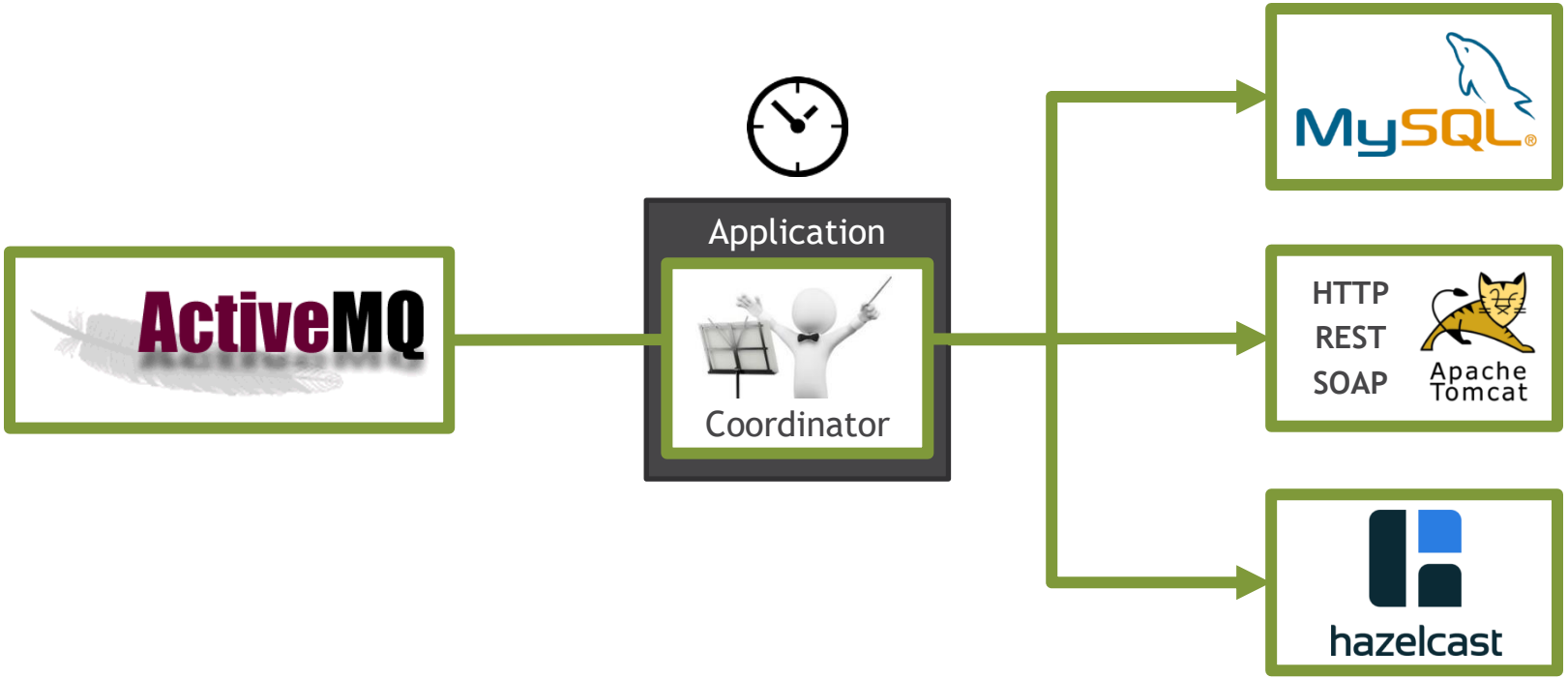

rollback
ActiveMQ
Application
MySQL
HTTP
REST
SOAP
Apache
Tomcat
hazelcast

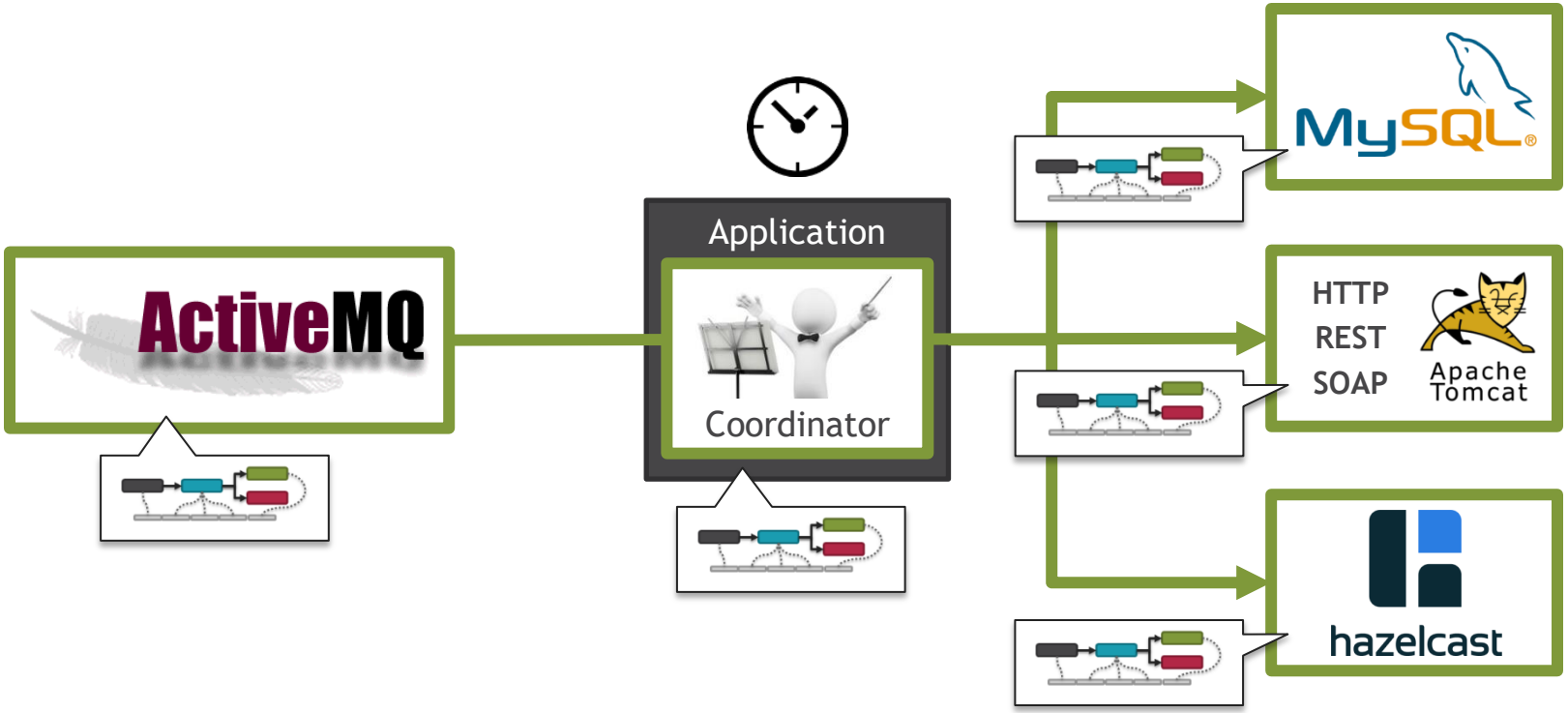
WHAT ACTUALLY HAPPENS



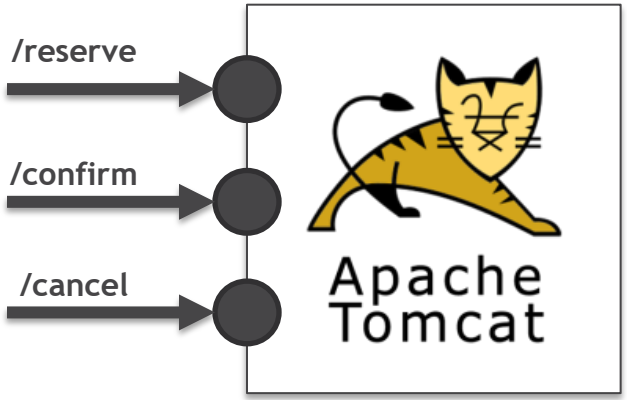

LIVE
breakyourownnews.com
BREAKING NEWS
FAILURES WITH TRANSACTIONS
18:46
CAUSED NEW WORLD ECONOMY CRISIS


STATE MACHINES
STATE MACHINES EVERYWHERE
imgflip.com

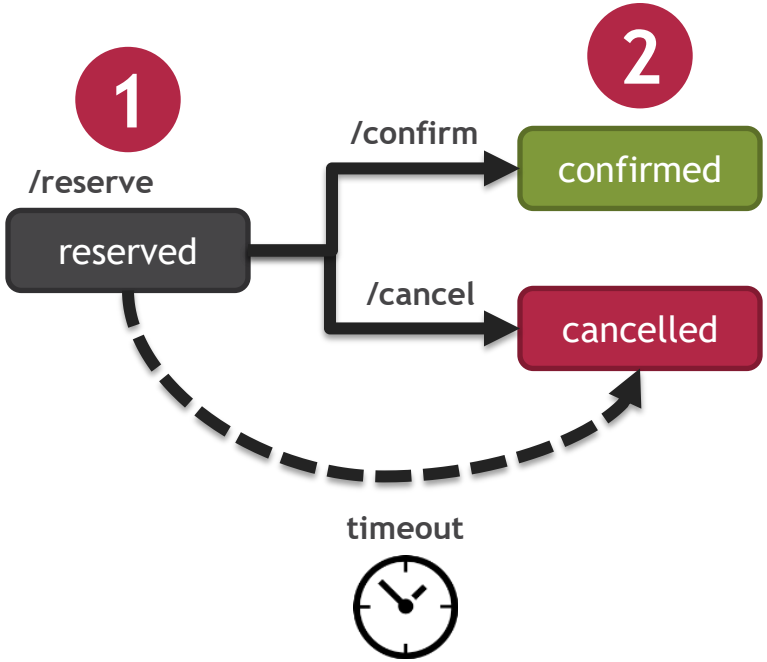

ActiveMQ
Application
Coordinator
MySQL
HTTP
REST
SOAP
Apache Tomcat
hazelcast


ActiveMQ
Application
Coordinator
MySQL
HTTP
REST
SOAP
Apache Tomcat
hazelcast

WEB-SERVICE AS STATE MACHINE



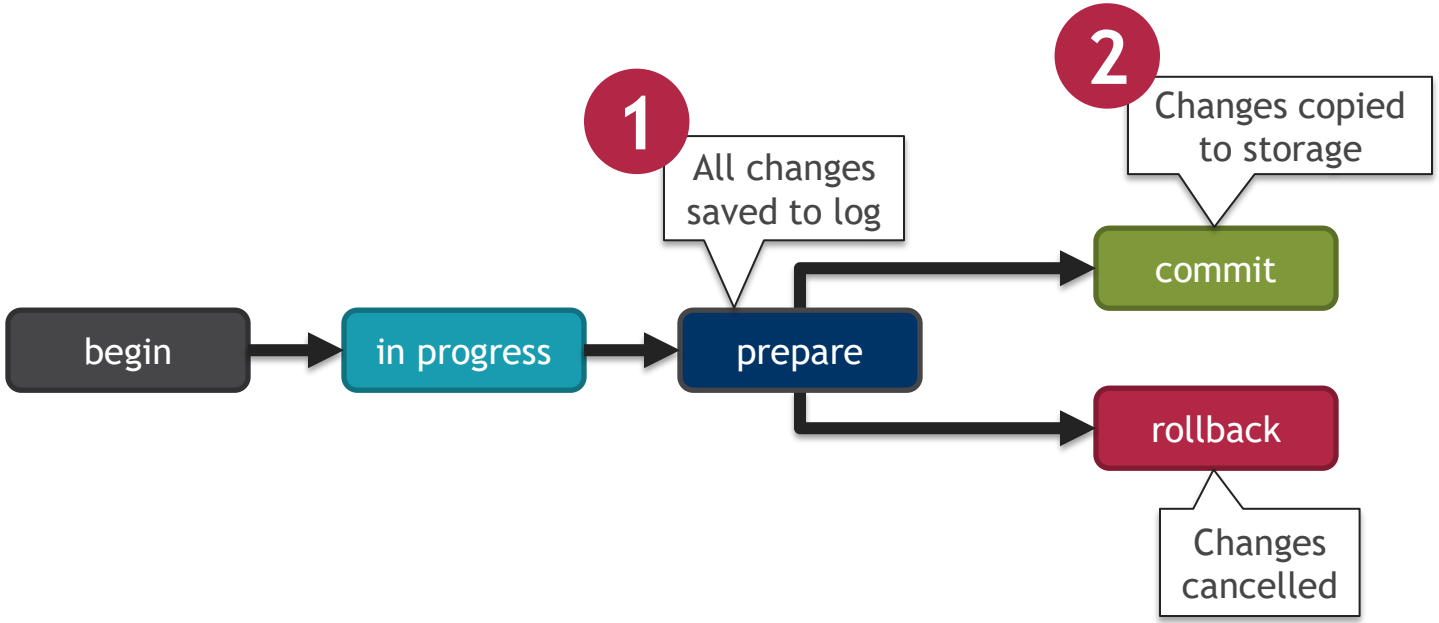
Idempotent operations



JTA/XA

2-PHASE COMMIT

2-PHASE COMMIT ALGORITHM



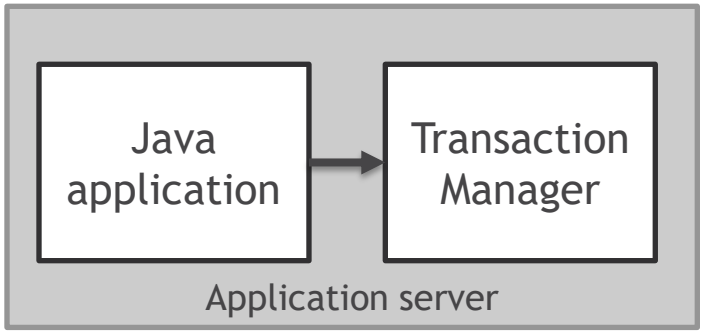
JTA AND XA SPECIFICATION

JTA AND XA SPECIFICATION

ORACLE FUSION MIDDLEWARE WEBLOGIC SERVER

IBM WebSphere

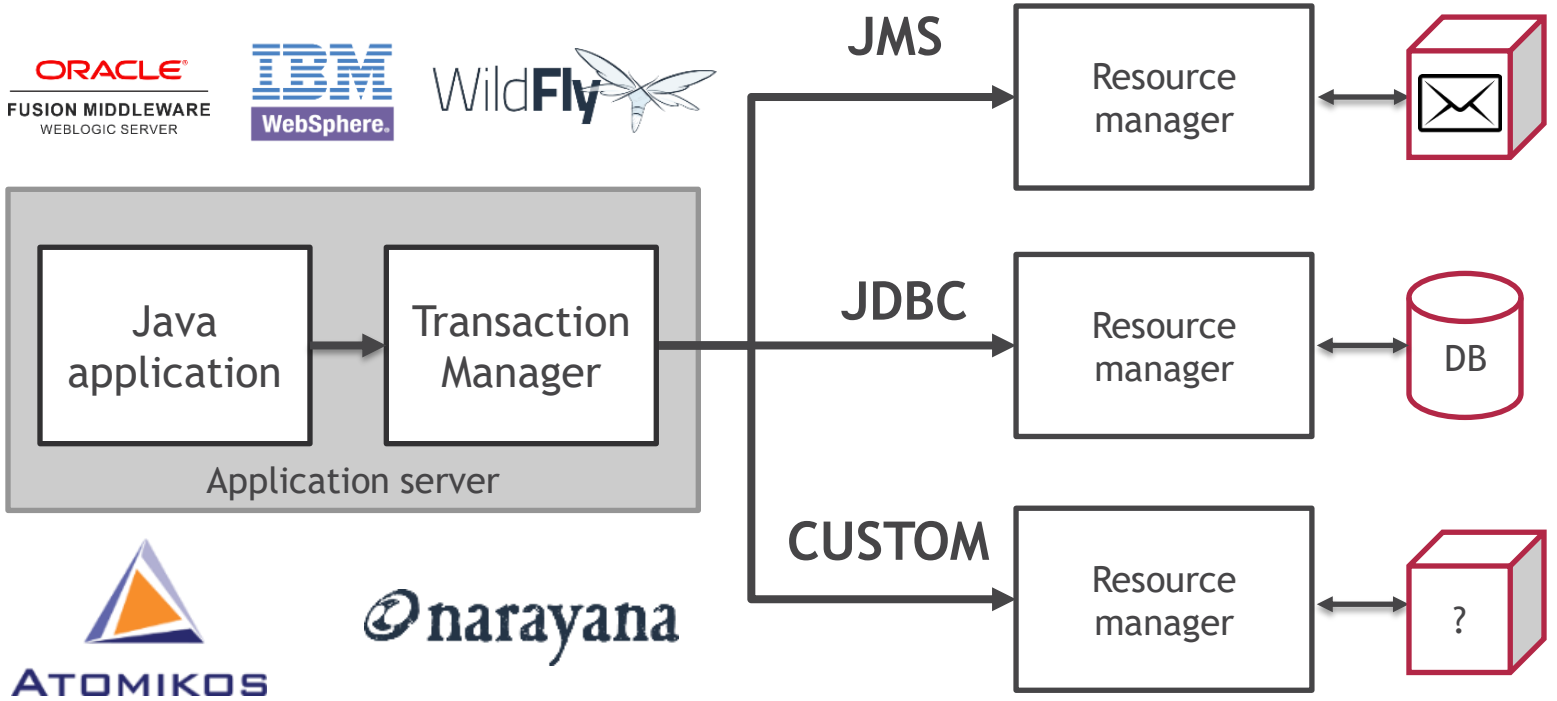
WildFly



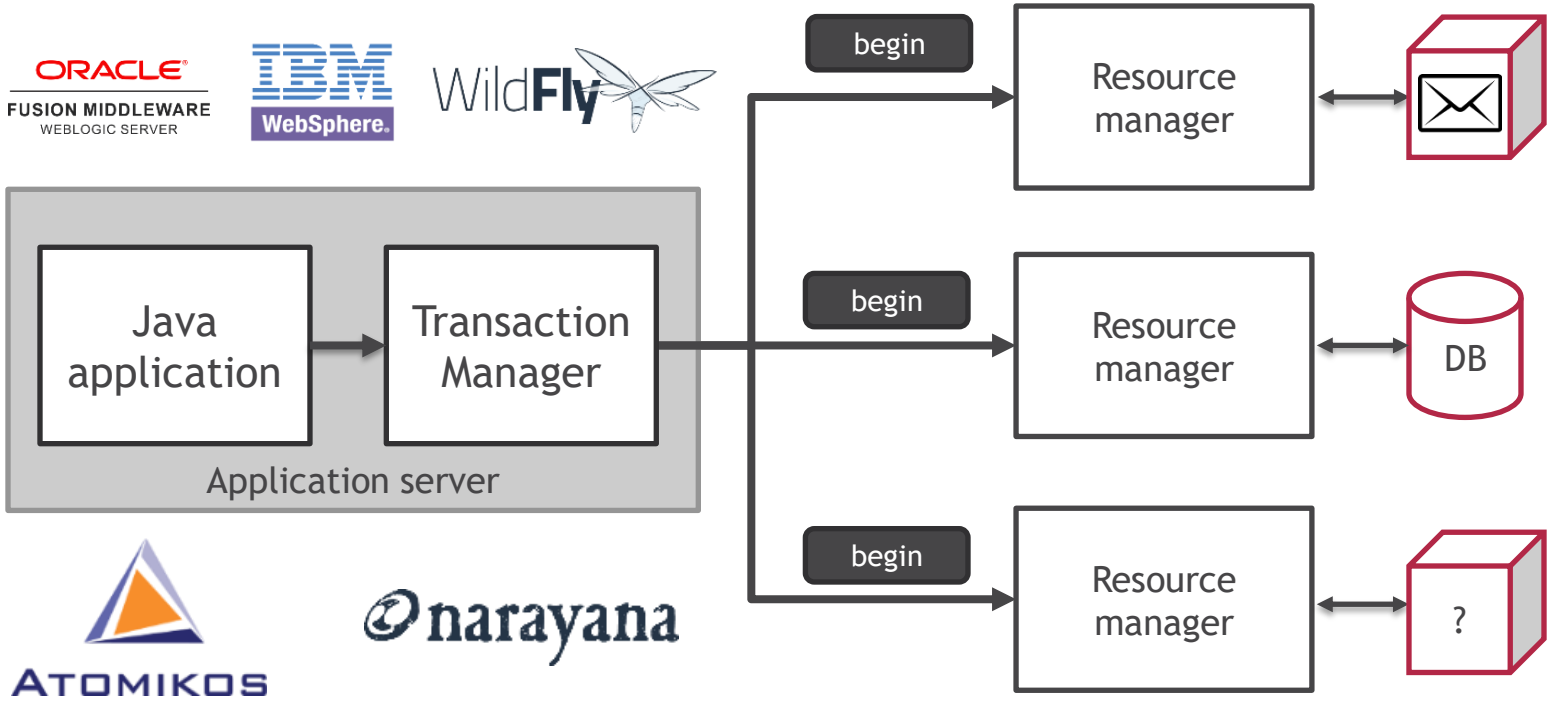
ATOMIKOS

narayana

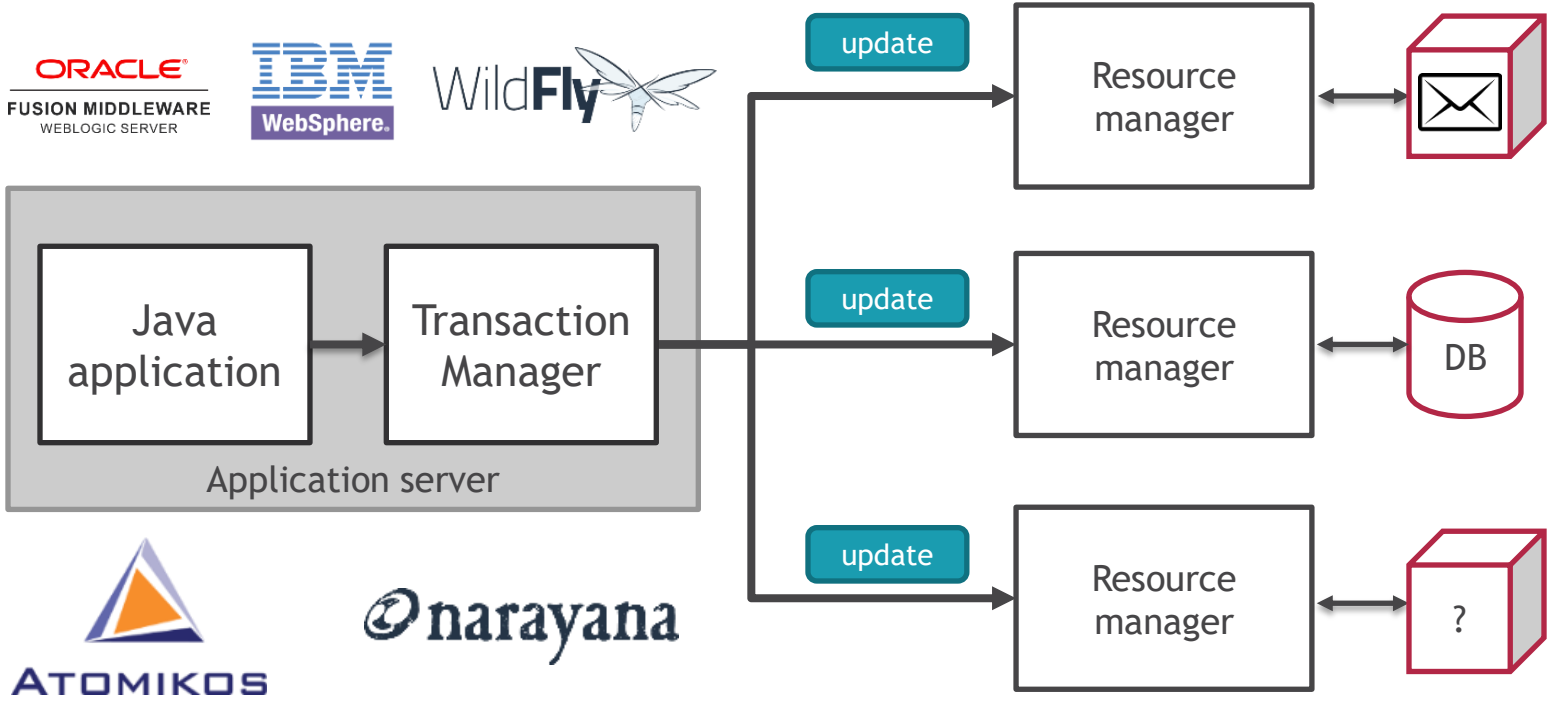
JTA AND XA SPECIFICATION



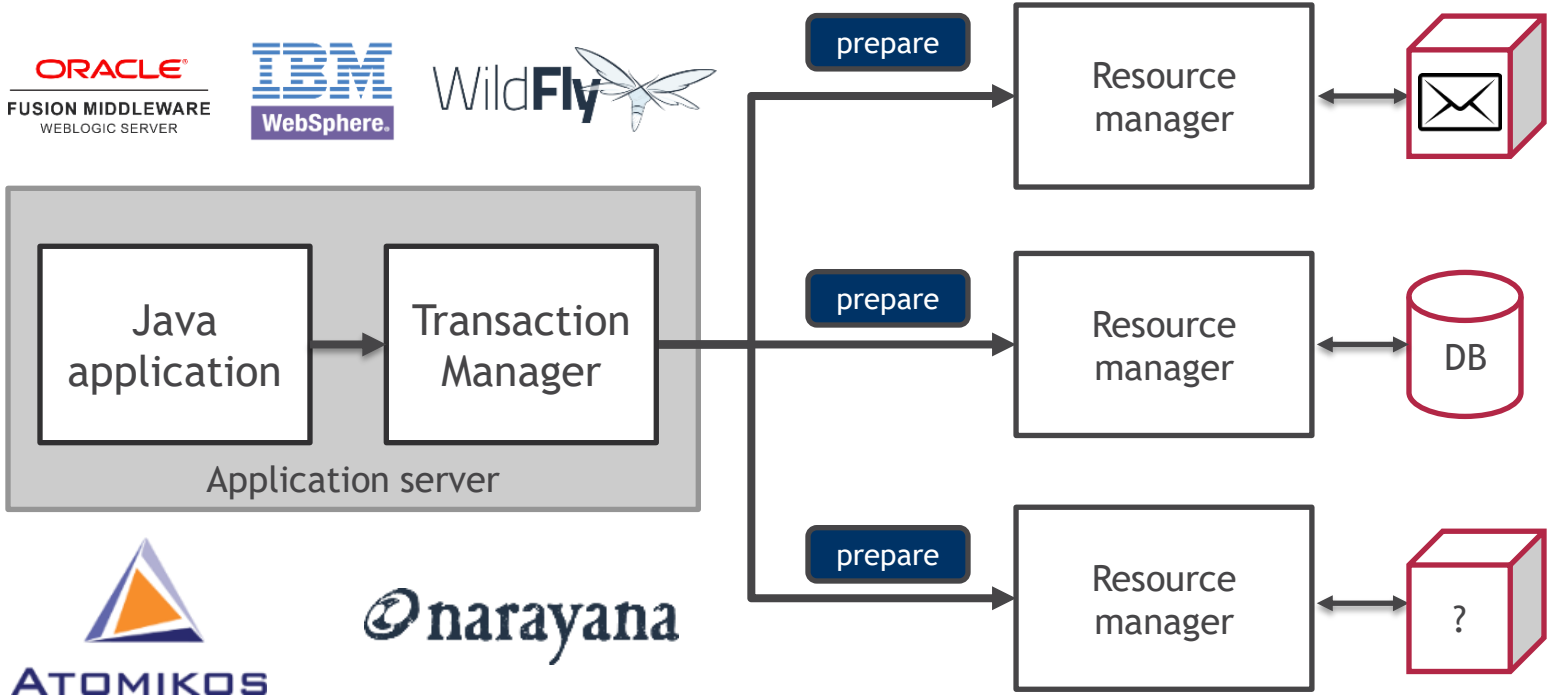
JTA AND XA SPECIFICATION



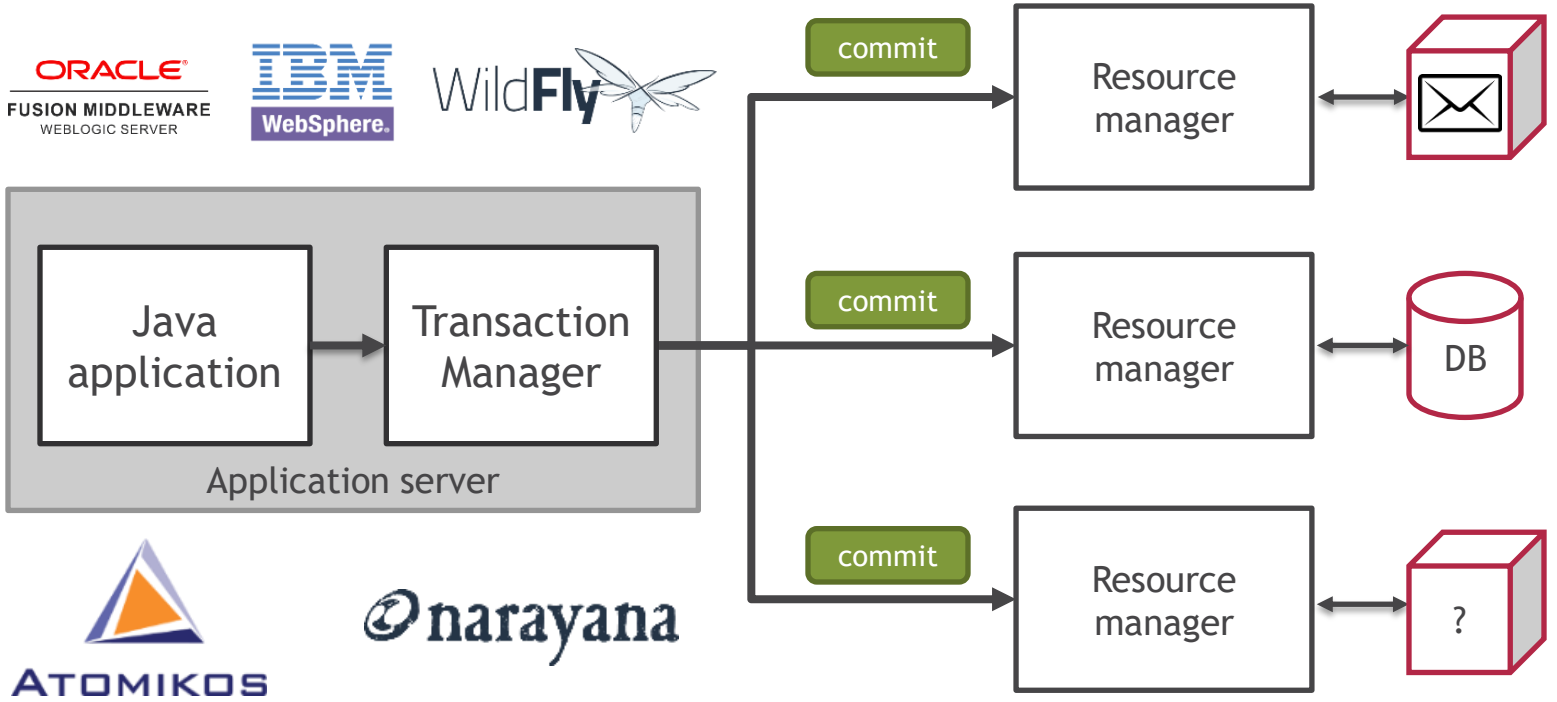
JTA AND XA SPECIFICATION



JTA AND XA SPECIFICATION



JTA AND XA SPECIFICATION



TM does all the job for us, but:

- Needs too many messages
- Doesn’t scale well
- Not all vendors support XA

AGENDA

1. ~~Local transactions~~

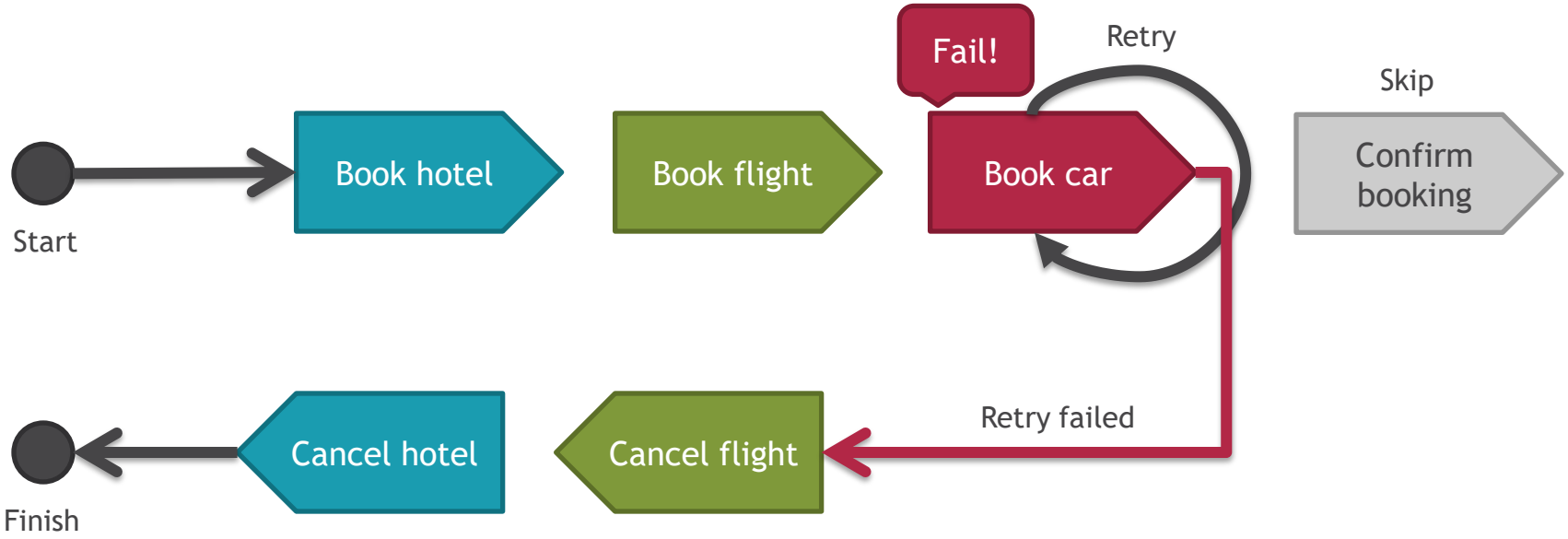
2. ~~Distributed transactions~~

3. Compensations: SAGA pattern

4. Live demo


WHAT IS SAGA?

SAGA EXAMPLE



SAGA TYPES

Central coordinator

Routing slip (peer-to-peer)

SAGA TYPES

Forward Backward

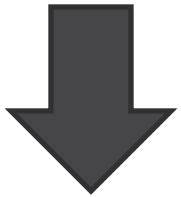
ACID

Availability

Consistency

Isolation

Durability



2-PHASE COMMIT

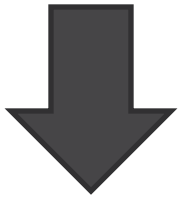
BASE

Basic Availability

Soft state

Eventual consistency

(Trading consistency for availability)



SAGA

CAP THEOREM

Consistency and Availability

System is single node

System is partitioned

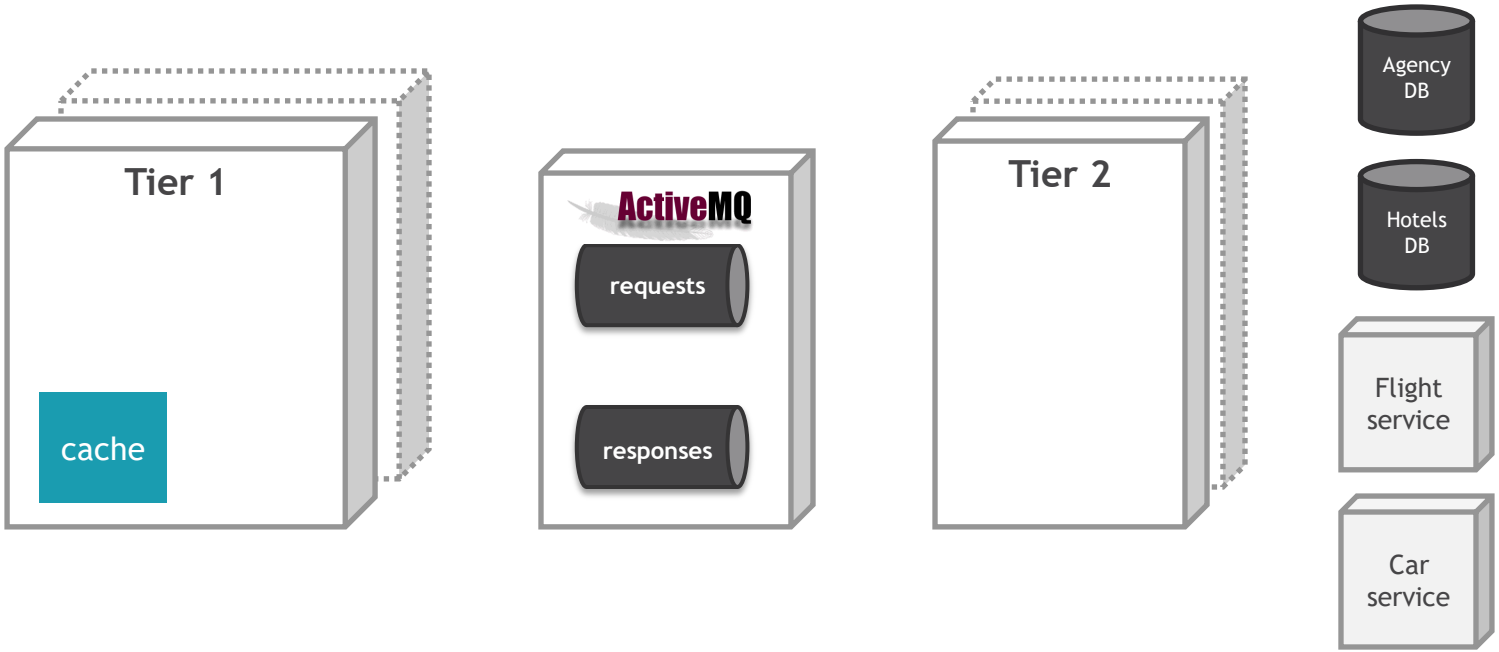
AGENDA

1. ~~Local transactions~~

2. ~~Distributed transactions~~

3. ~~Compensations: SAGA pattern~~

4. Live demo



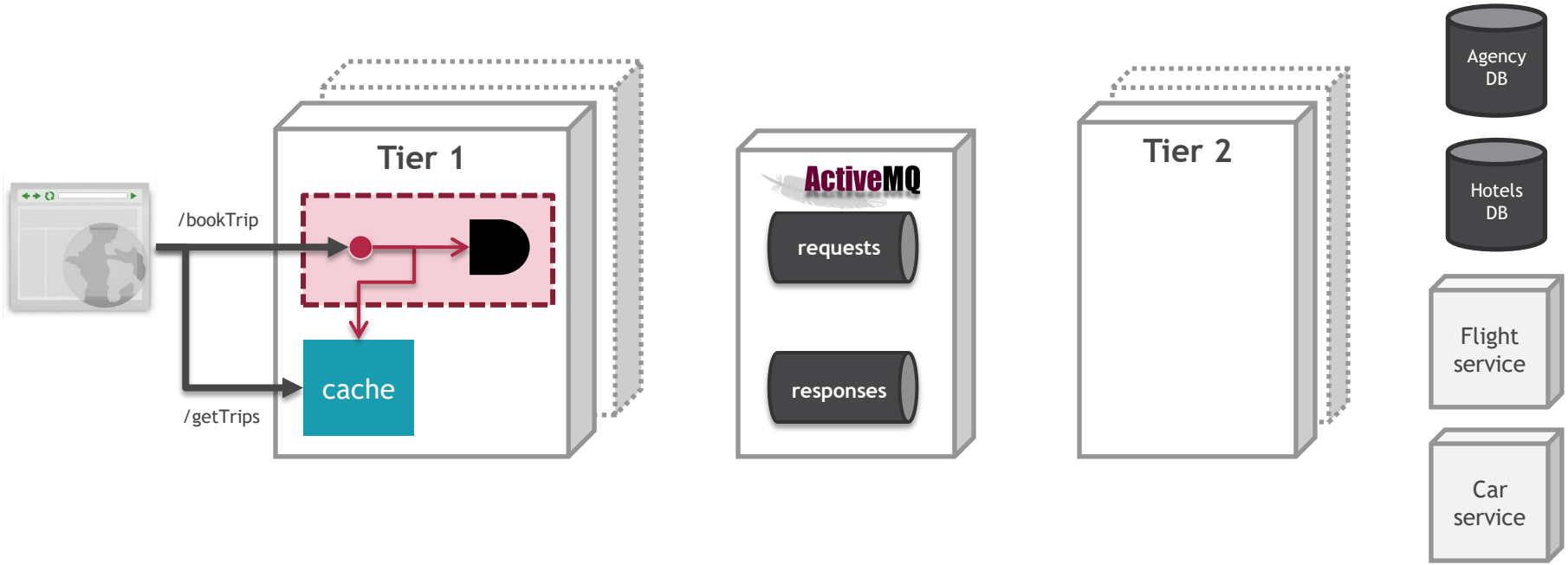
Distributed transaction

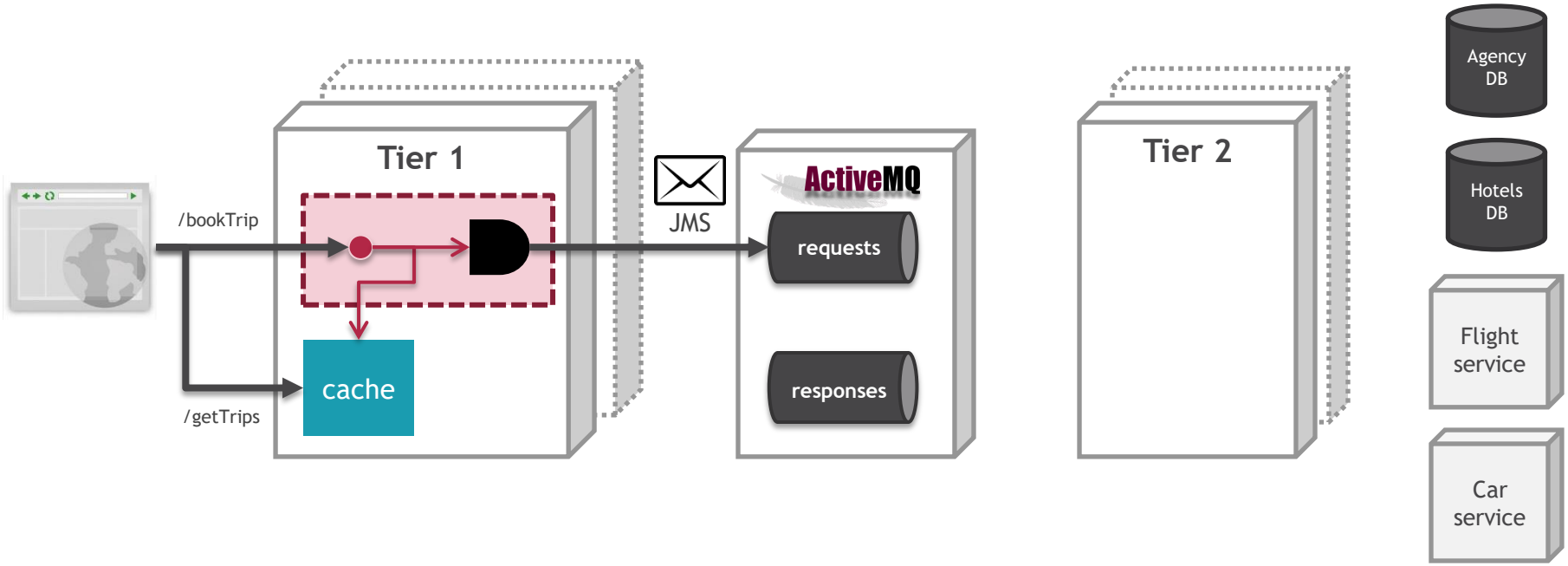
JMS endpoint

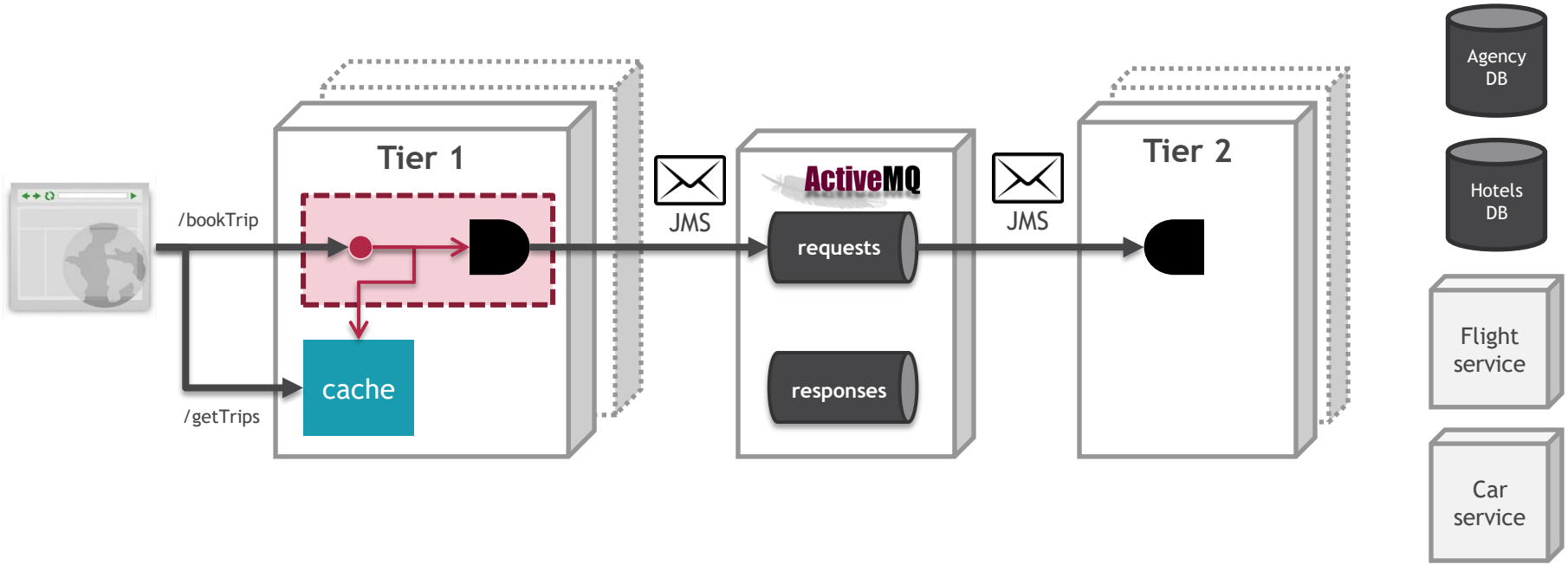
Remote call

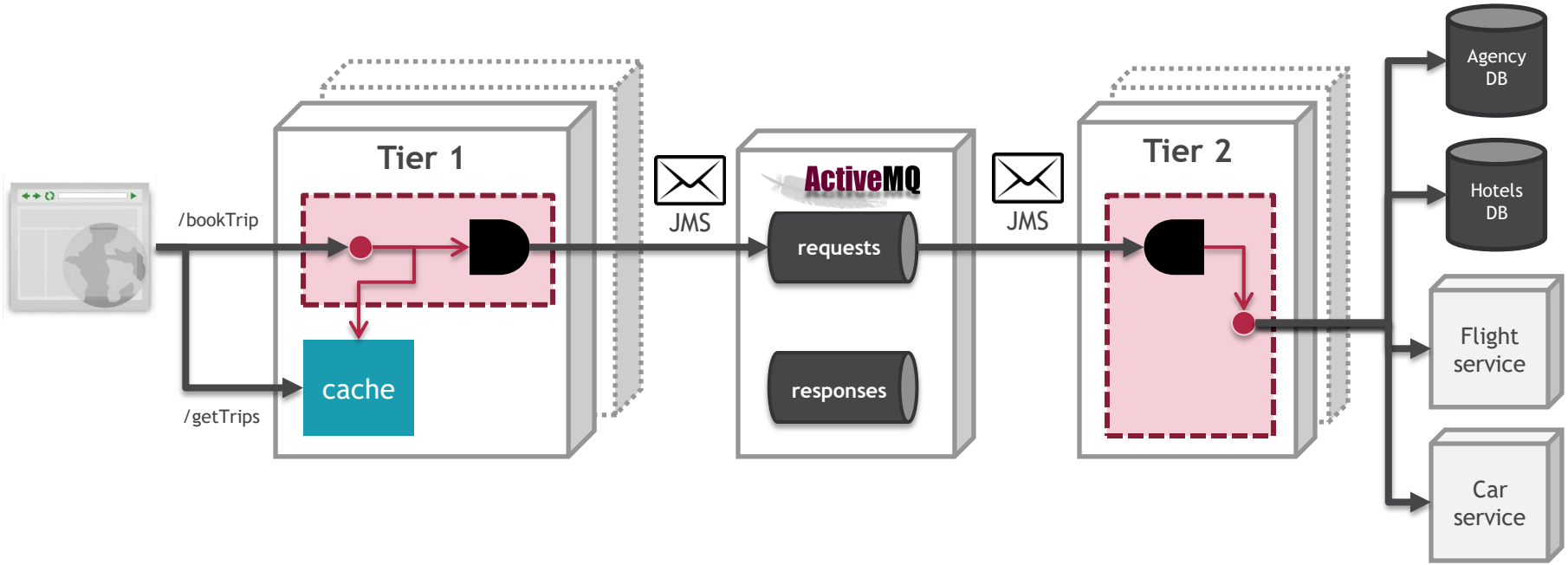
Local call

JMS queue

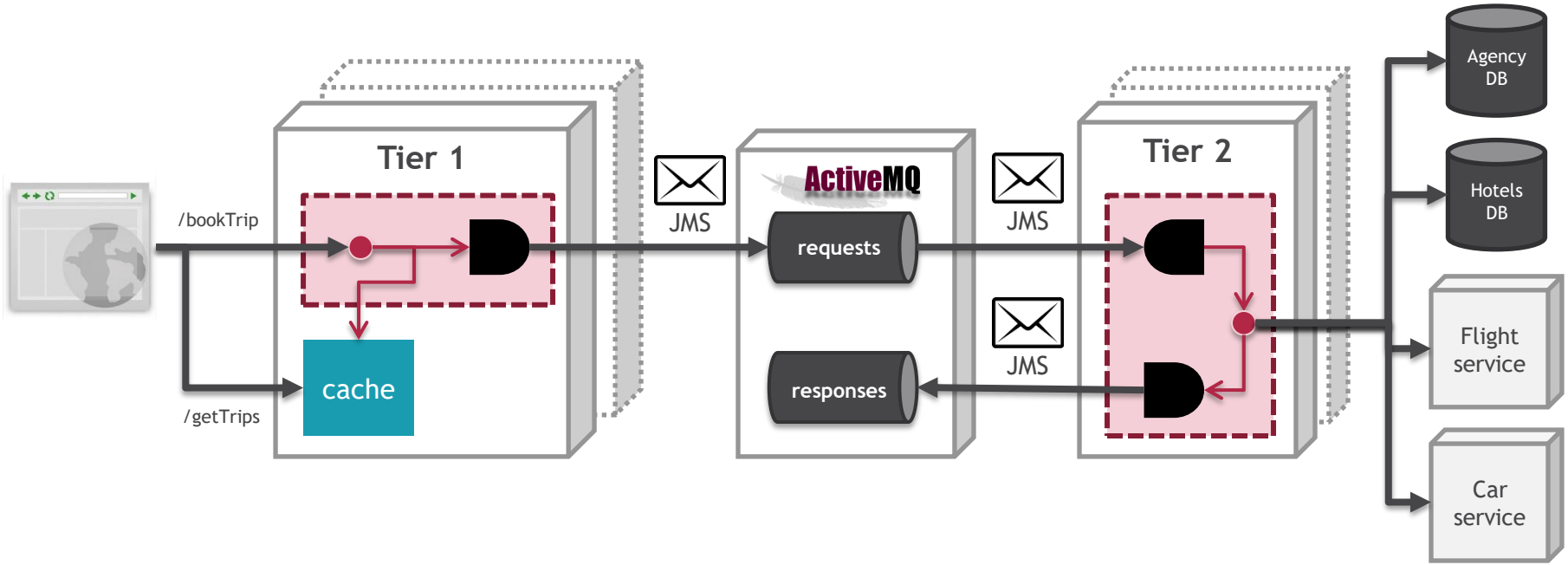

Tier 1
/bookTrip
cache
/getTrips
ActiveMQ
requests
responses
Tier 2
Agency DB
Hotels DB
Flight service
Car service
Distributed transaction
JMS endpoint
Remote call
Local call
JMS queue

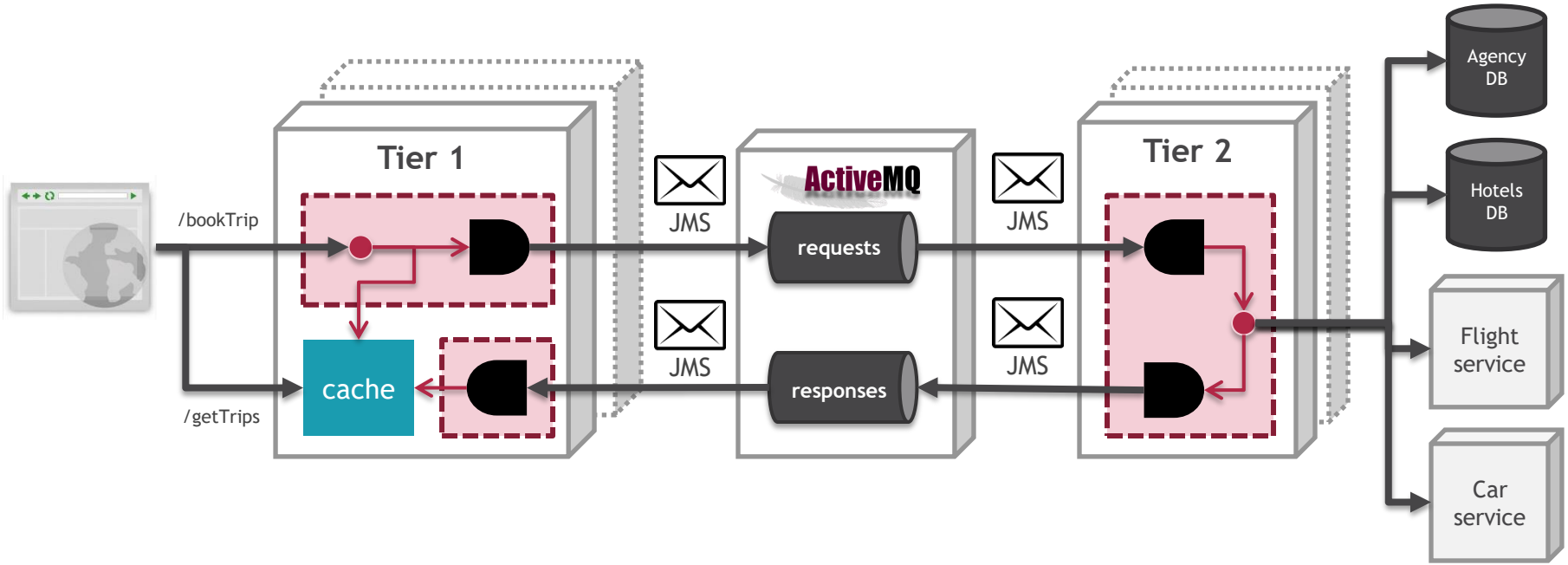

Tier 1
/bookTrip
/getTrips
cache
JMS
ActiveMQ
requests
responses
Tier 2
Agency DB
Hotels DB
Flight service
Car service
Distributed transaction
JMS endpoint
Remote call
Local call
JMS queue


Tier 1
/bookTrip
/getTrips
cache
JMS
ActiveMQ
requests
responses
JMS
Tier 2
Agency DB
Hotels DB
Flight service
Car service
Distributed transaction
JMS endpoint
Remote call
Local call
JMS queue



Distributed transaction
JMS endpoint
Remote call
Local call
JMS queue


Tier 1
/bookTrip
/getTrips
cache
JMS
ActiveMQ
requests
responses
JMS
JMS
Tier 2
Agency DB
Hotels DB
Flight service
Car service
Distributed transaction
JMS endpoint
Remote call
Local call
JMS queue



Distributed transaction

JMS endpoint

Remote call

Local call

JMS queue


LET’S GO!

CONCLUSION

1. State machines everywhere
2. Transactions everywhere
3. Design for failure
4. Rely on tools or handle by own

THANK YOU!

QUESTIONS?

KANSTANTSIN_SLISENKA@EPAM.COM