Transactions in your micro-services architecture

Dawn Parzych (CatchPoint)

Rudy De Busscher

payara®

What?

- Transaction and Micro-Services
- MicroProfile LRA (Long Running Actions)
- Demo

payara®

Rudy De Busscher

- Payara
  - Service team
- Involved in
  - Committer of MicroProfile
  - Committer in Eclipse EE4J groups
  - Java EE Security API Expert group member

@rdebusscher

https://blog.payara.fish/
https://www.atbash.be

payara®

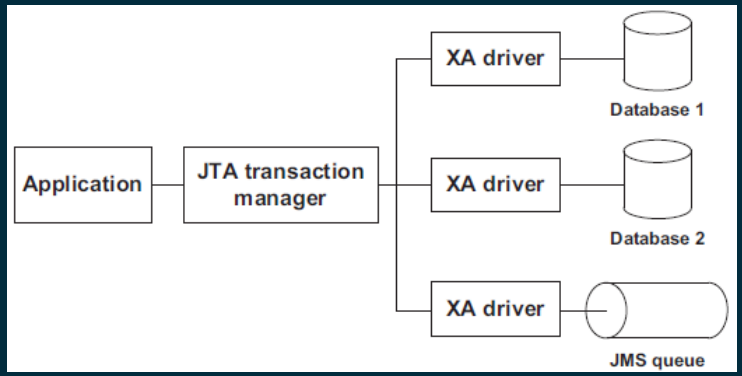
Transaction

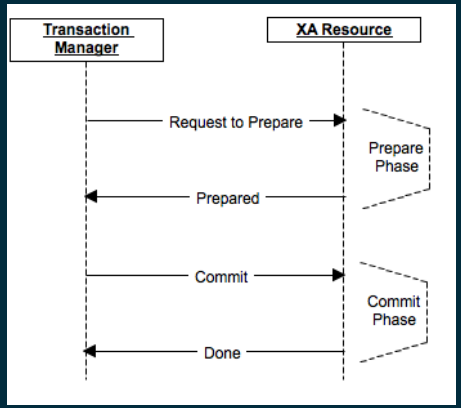
- Typical concept from the ‘Monolith’
- Start - End
- Typical short - to avoid locks

payara®

Distributed Transaction

- Multiple datasources - Distributed
- XA transactions
- 2 Phase Commit
- Not scalable
  - Locks





To be ACID or not

- *Atomicity*: All or Nothing
- *Consistency*: All values are aligned
- *Isolation*: Different transactions can't see 'temporal' values
- *Durable*: Reliably Stored

- **<u>CAP Theorem</u>**
  - Consistency - Availability - Partitioning

payara®

Long Running Transactions

- Multiple Datasource
- Several steps over long period
- Single Unit of Work?

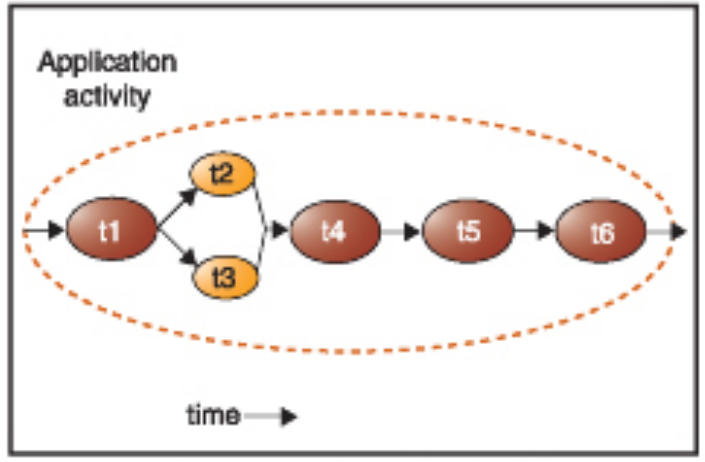


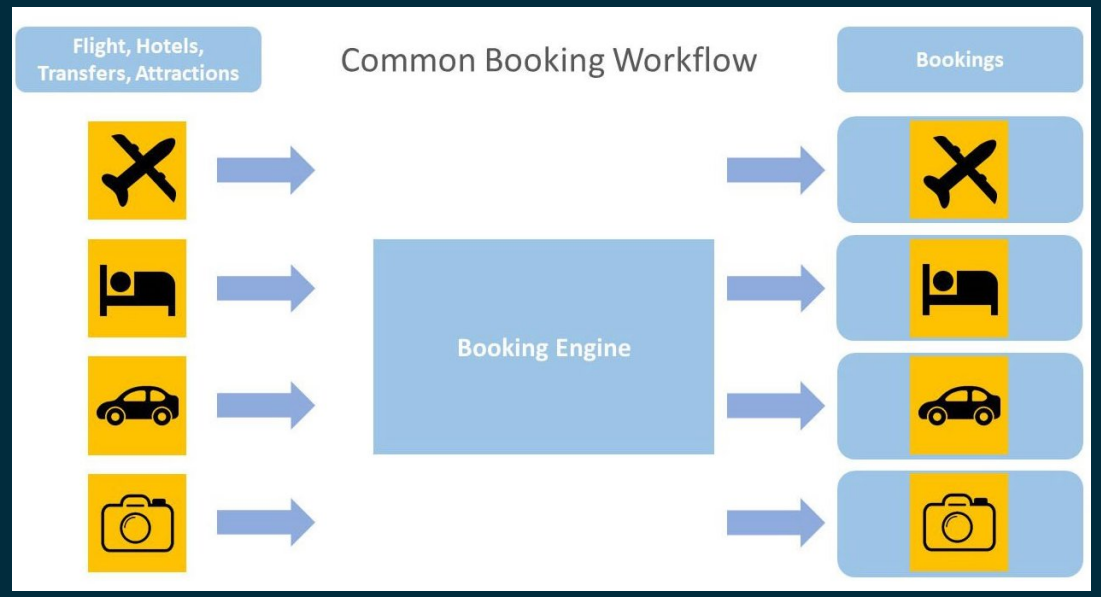
FIGURE 2 An example of a logical long-running "transaction," without failure

Web Services Journal

payara®

Example

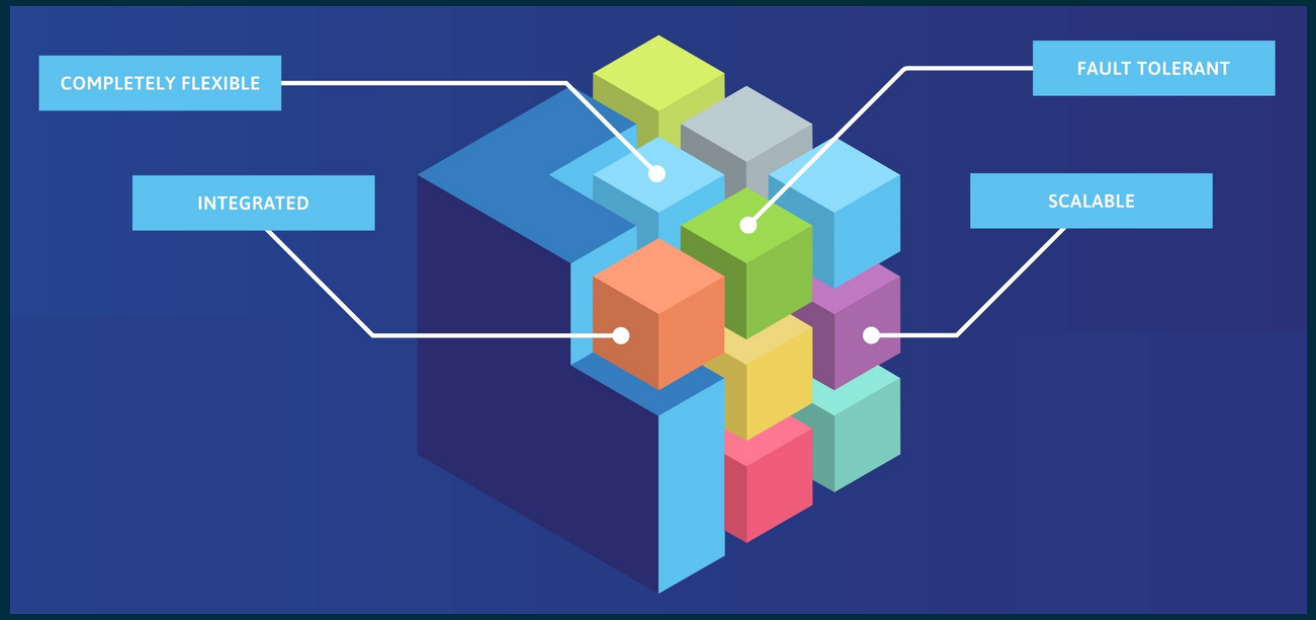
- Book Flight
- Book Hotel
- Payment



AltexSoft

Microservice world

- No Atomicity
- Eventual Consistency
- No Isolation
- Durable

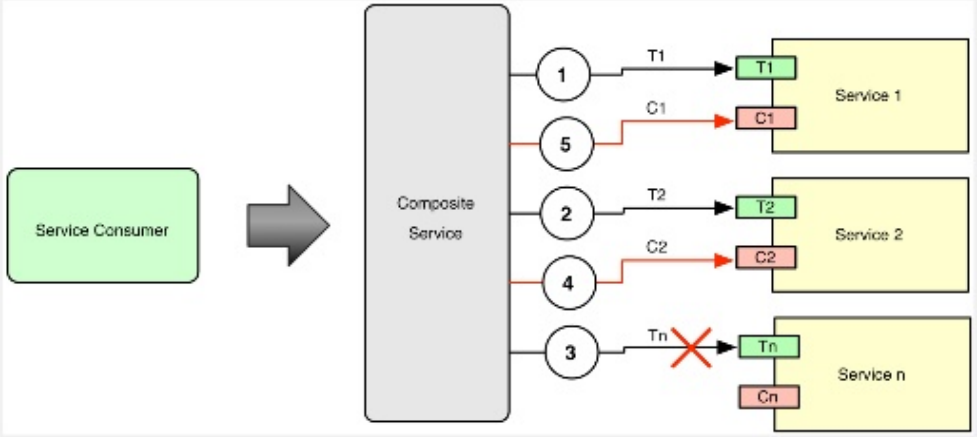


KumuluzEE

SAGA Pattern

Saga Pattern

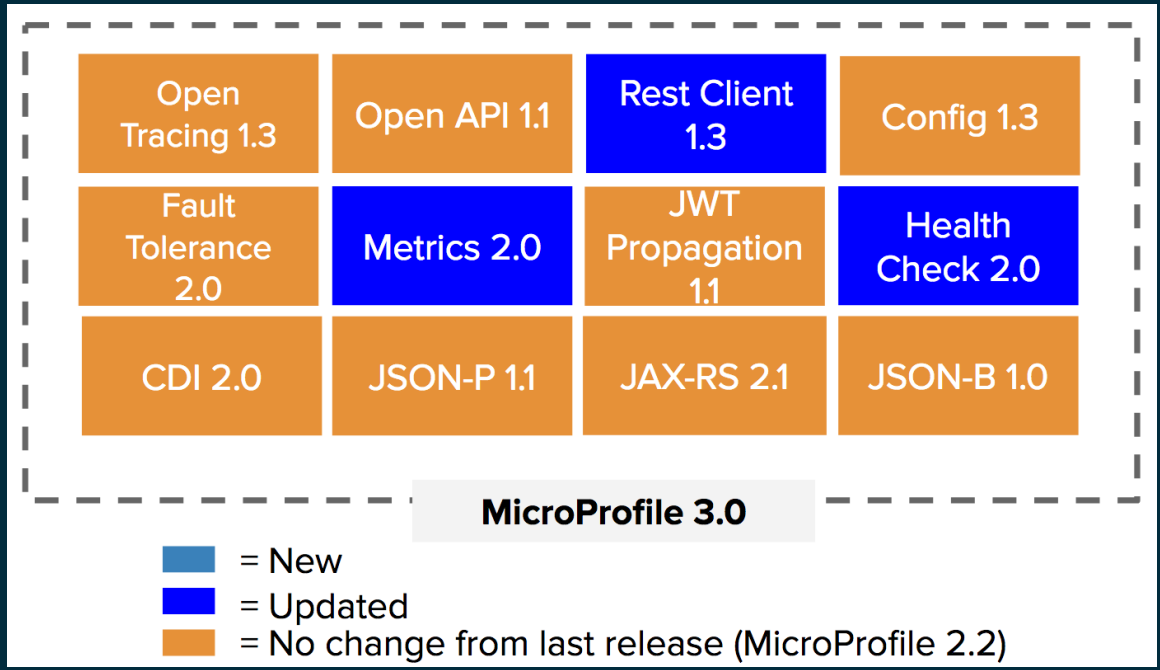
Ensures that each step of the business process has a compensating action to undo the work completed in the case of partial failures.



redhat.

payara®

- Optimizing Enterprise Java for a Micro-services Architecture
- Based on some Java EE (Jakarta EE) specs



payara®

MP Long Running Action

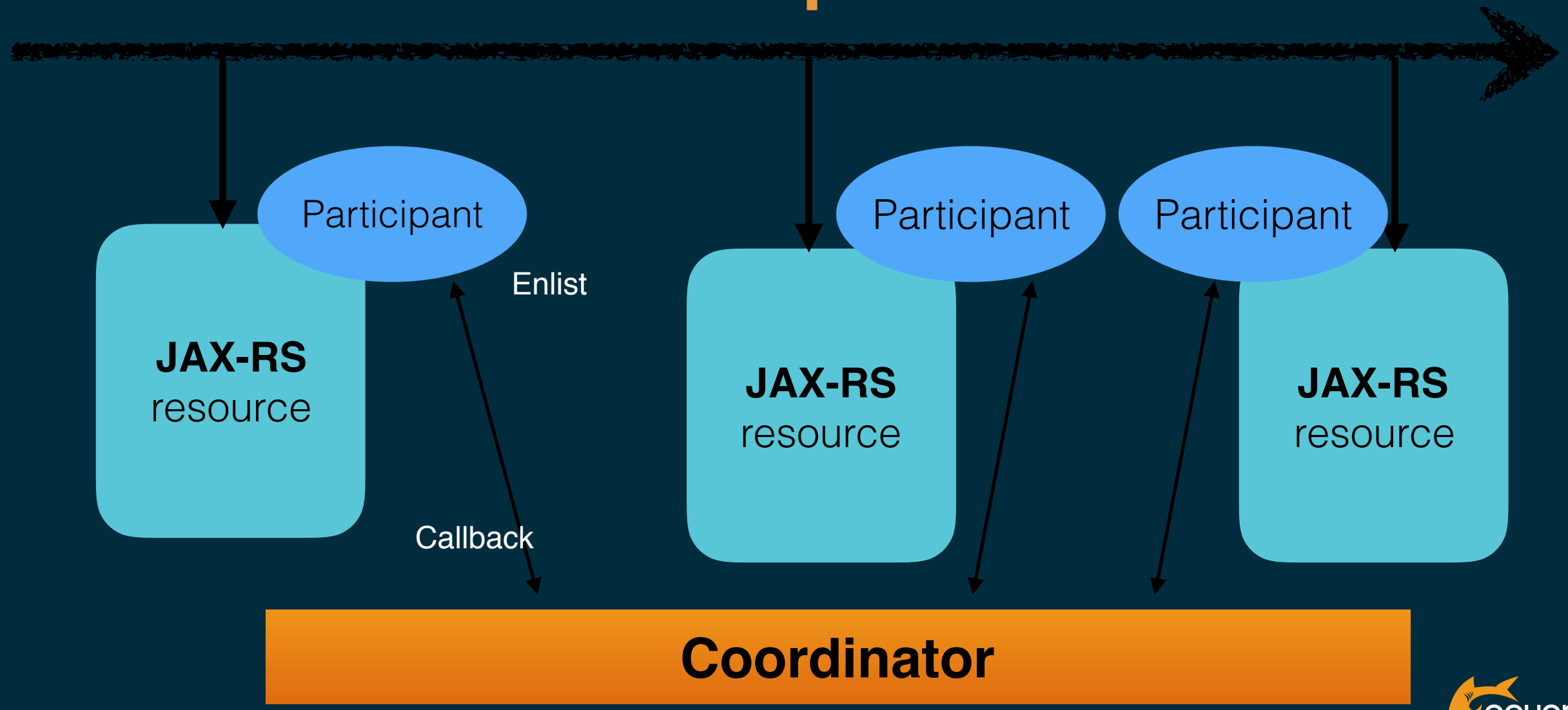
- First release candidate available.
- Finalised in coming months.

MicroProfile LRA

- Long Running Actions
- Features
  - Loose coupling
  - Guaranteed a globally consistent outcome
  - Compensatable actions (SAGAs)

payara®

LRA Components



DEMO

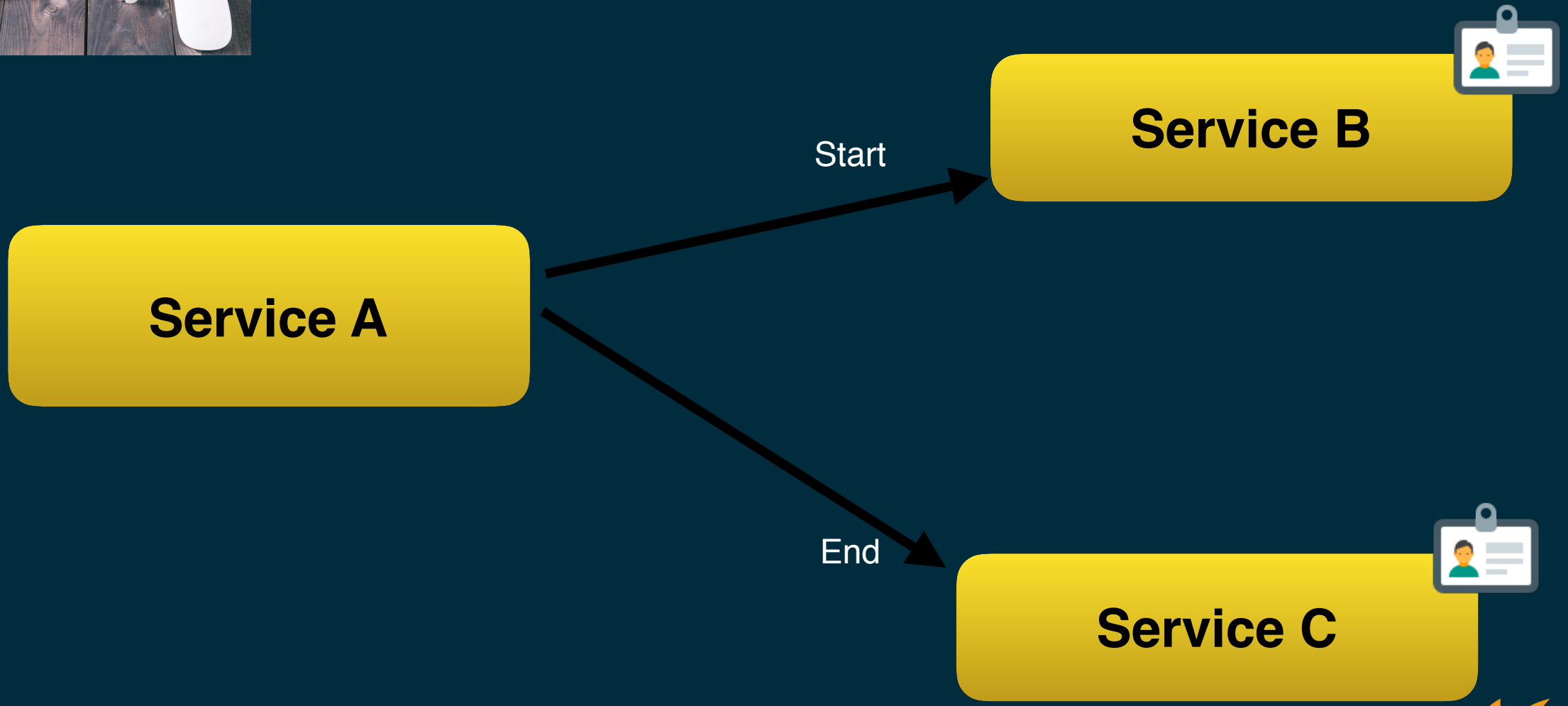
payara®

Some Key Code concepts

- @LRA
  - LRA Level / defines transaction
- @Compensate
  - Participant level: Required
- @Complete
  - Participant level
- LRA ID (= URI)
  - Uniquely defines the Long Running Action (transaction)
  - Propagated through Header

payara®

Distributed



Returning Participant Status

- Immediate
  - Return type `ParticipantStatus, void`
- Asynchronous return
  - Return `CompletableFuture<ParticipantStatus>`
- Idempotent `@Complete/@Compensate`
  - Return type `ParticipantStatus`
  - `ParticipantStatus.Completing / ParticipantStatus.Compensating`
- Through `@Status` method
  - First call to `@Complete/@Compensate`
  - Following calls to `@Status`

payara®

TimeOut

- By Default, LRA runs forever
- But also by Default, immediately closed

- Closed explicitly (end=true)
- Define a timeout -> cancelled / all participants compensated.

payara®

Remove Participant

- @Leave
- Remove Participant from LRA
- Developer responsible for cleanup
  - No @Compensate/@Complete/@Status called

payara®

Feedback Final status

- @AfterLRA
  - What was the final outcome

payara®

Store LRA info

- LRA only performs orchestration
- Participant needs to keep track of LRA
- Store LRA Id as part of your business data
- Extension
  - LRAData
  - ParticipantData

payara®

Takeaways

- Classic approach for Transactions and Long Running Action not applicable for micro-services
- MicroProfile LRA uses Compensatable actions
- Loosely coupled on top of JAX-RS resources
- Specification : Almost final

payara®

Code

- Project https://github.com/eclipse/microprofile-lra
- Demo code
  - https://github.com/rdebusscher/mp-lra-demo

payara®

Q & A

payara®

Thank You

Download the open source software:
https://payara.fish/downloads

Need support for the Payara Platform?
https://payara.fish/support

payara®