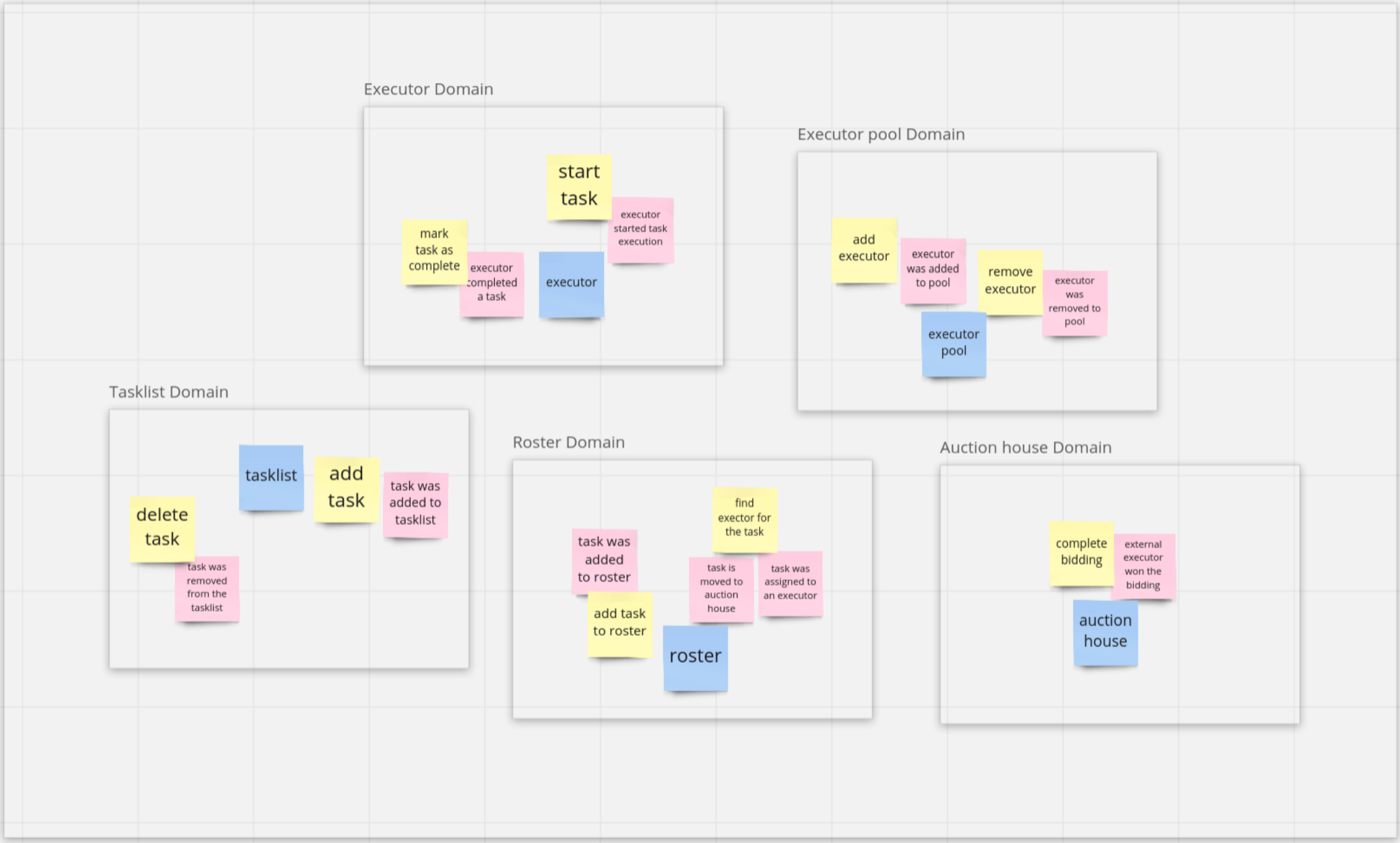
Group 3


Executor Domain
start task
executor started task execution
mark task as complete
executor completed a task
executor
Executor pool Domain
add executor
executor was added to pool
remove executor
executor was removed to pool
executor pool
Tasklist Domain
tasklist
add task
task was added to tasklist
delete task
task was removed from the tasklist
Roster Domain
find exector for the task
task was added to roster
task is moved to auction house
task was assigned to an executor
add task to roster
roster
Auction house Domain
complete bidding
external executor won the bidding
auction house

Client request

Client request

Client request

API Layer

Tasklist

module

module

Roster

module

module

Executor

module

module

Executor Pool

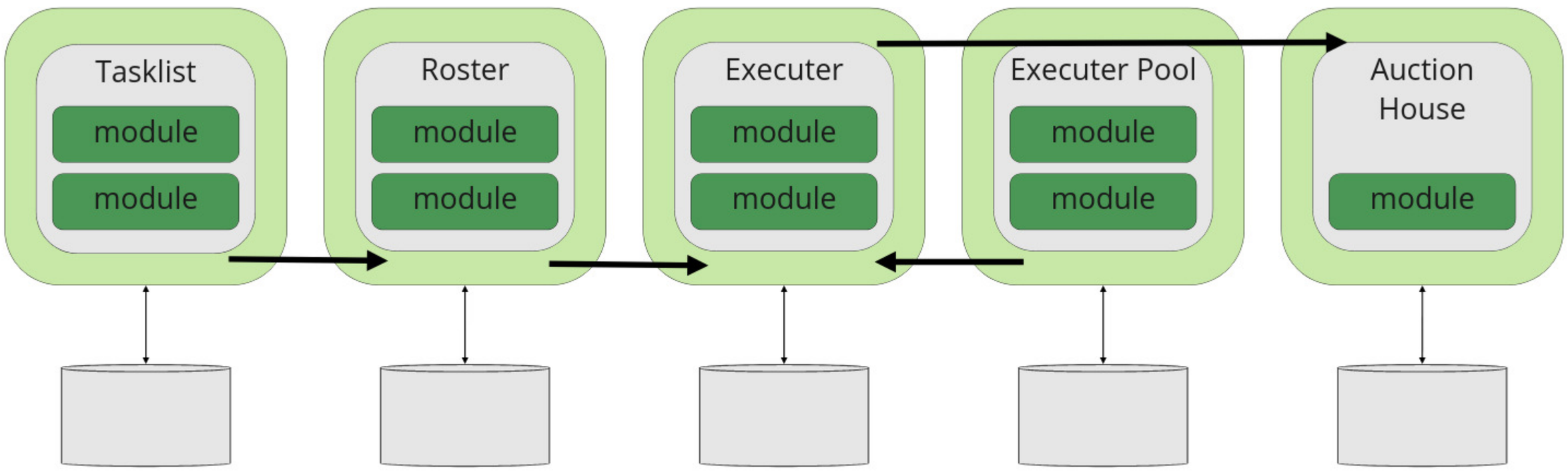
module

module

Auction House

module

Workflow and Choreography



1. Use Microservices as software architecture

Date: 2022-09-26

Status

Accepted

Context

We need a distributed architecture because we need to have a collection of independently deployable components.
Based on our chosen architecture characteristics interoperability, fault-tolerance and scalability we evaluated an event-driven system or microservices.
We chose microservices because of lower complexity which leads to less developement overhead, additionally microservices satisfy ACID properties.

Decision

We propose to user a microservice architecture

Consequences

Our system gets more agile and scalable and is easier to extend.
The workflow will be more complex and remote method calls will impact performance.

2. Service granularity

Date: 2022-09-26

Status

Accepted

Context

Based on our five domains we checked against the integrators and disintegrators to see if we can divide or join the domains into microservices.

Service functionality:
The domains are already divided into the different functionalities therefore it's suitable to create one microservice for each domain.

Code Volatility:
Inside the different microservices components have the same code volatility.

Scalability and Throughput:
Different scaled services are already divided into their own microservices, e.g. The Executor Pool need to scale differently than the Tasklist service based on calls per minute.

Fault Tolerance:
The services are independent if one service fails the others will still work, e.g. if the roster fails tasks and executors can still be added.

Data Security:
At this point no services need special security. In the future this could change based on data ownership requirements.

Extensibility:
The parts of the systems that need to be extended are separated into microservices, e.g. the auction house

Database transaction:
There is no action that requires a single data transaction that need to be executed in multiple services therefore we don't need to join any services.

Workflow and choreography:
There are no services that need to execute tasks in synchronization, services work independently.

Shared code:
As of now there are no shared libraries that are used in multiple services

Data relationships:
Each service handles their own data. Roster, executor and tasklist services could use the same database but can also be decoupled with ease. Based on advantages listed before we decided to decouple the services.

Decision

5 microservices are created: Tasklist, Roster, ExecutorPool, Executor, AuctionHouse

Consequences

Decoupling the services into 5 services provides flexibility but also adds more complexity and overhead.