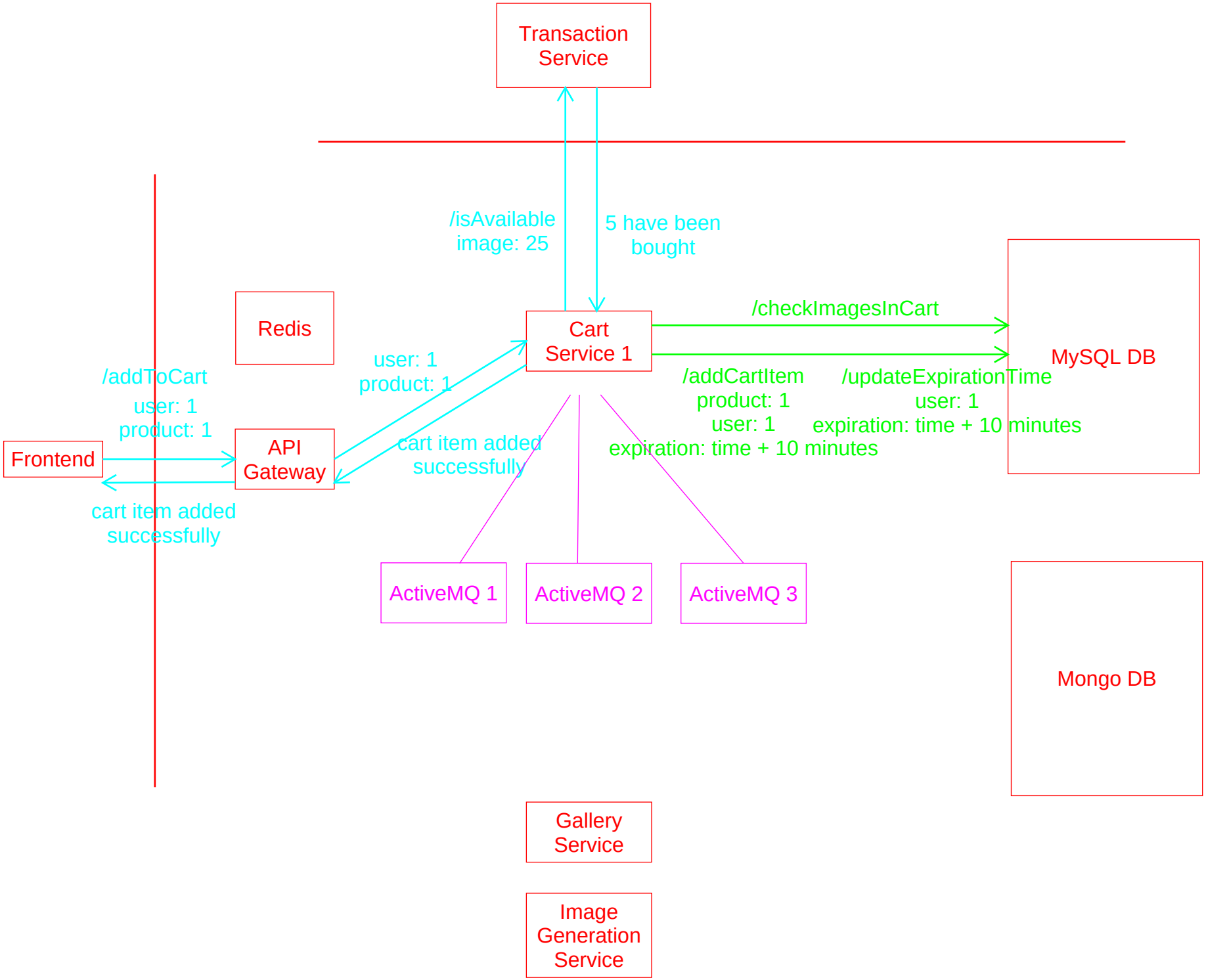
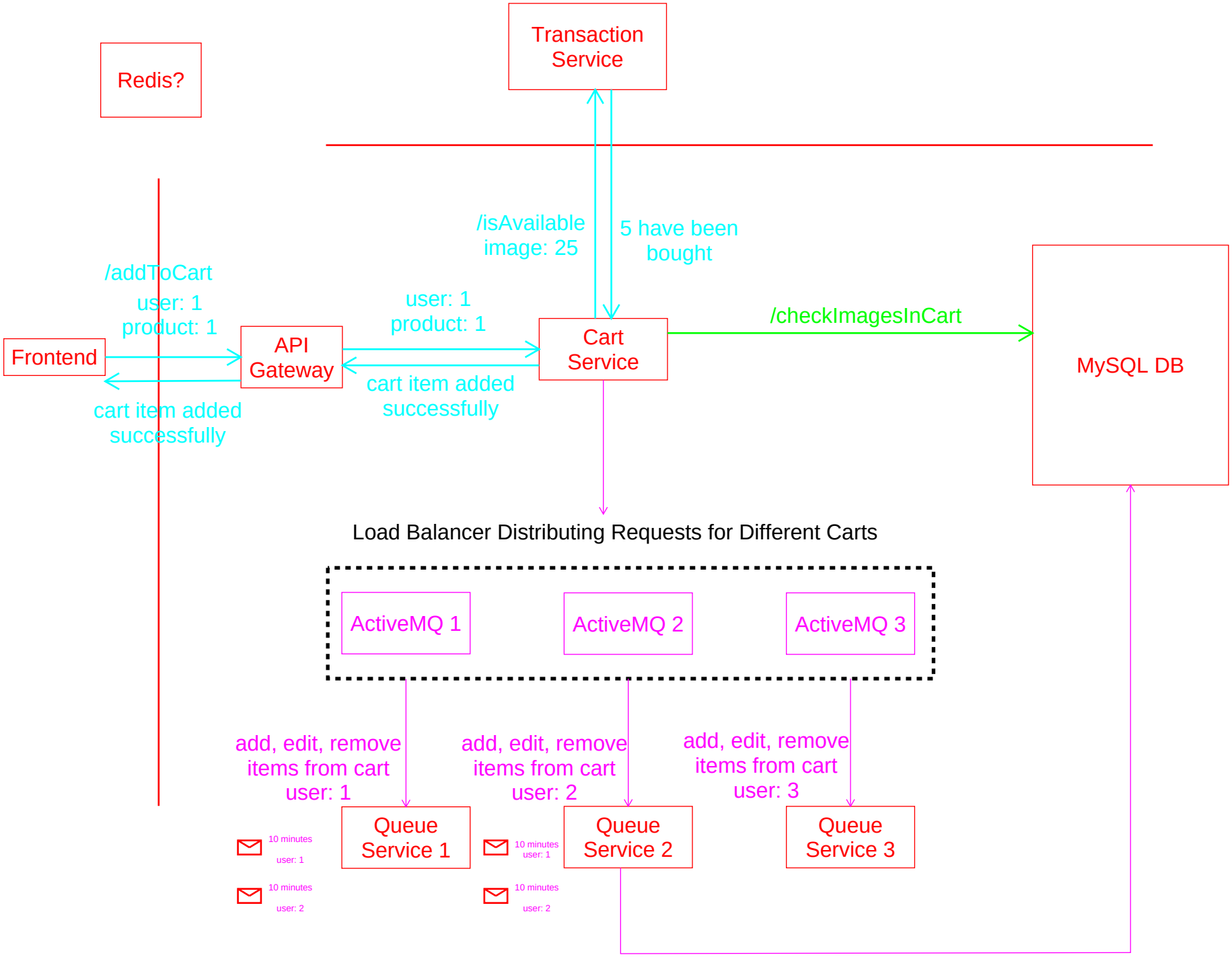
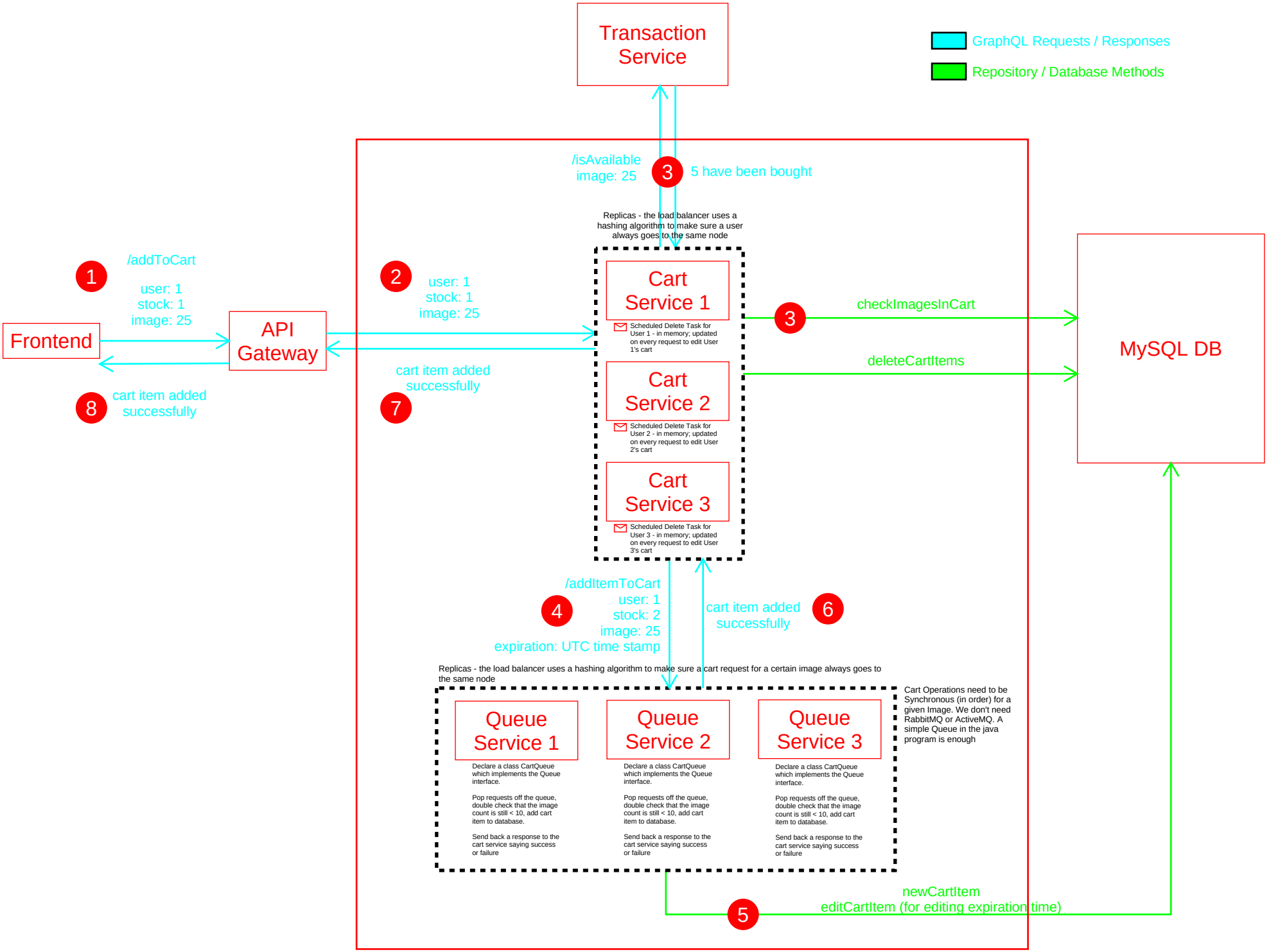
CART SERVICE DRAFTS

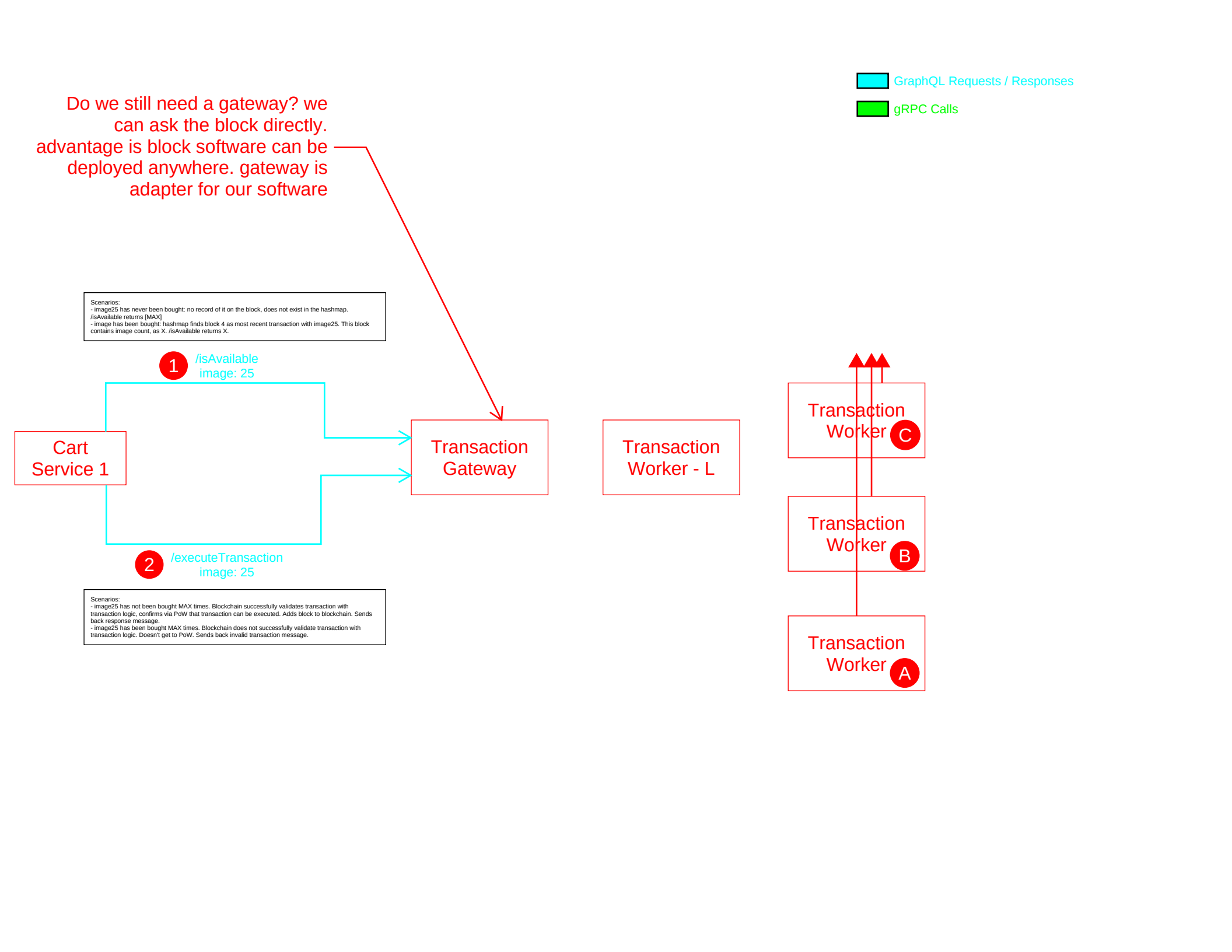

Transaction Service
/isAvailable
image: 25
5 have been bought
Redis
Cart Service 1
/checkImagesInCart
MySQL DB
/addToCart
user: 1
product: 1
user: 1
product: 1
/addCartItem
product: 1
user: 1
expiration: time + 10 minutes
/updateExpirationTime
user: 1
expiration: time + 10 minutes
Frontend
API Gateway
cart item added successfully
cart item added successfully
ActiveMQ 1
ActiveMQ 2
ActiveMQ 3
Mongo DB
Gallery Service
Image Generation Service


Redis?
Transaction Service
/isAvailable
image: 25
5 have been bought
/addToCart
user: 1
product: 1
Frontend
API Gateway
user: 1
product: 1
Cart Service
/checkImagesInCart
MySQL DB
cart item added successfully
cart item added successfully
Load Balancer Distributing Requests for Different Carts
ActiveMQ 1
ActiveMQ 2
ActiveMQ 3
add, edit, remove items from cart user: 1
add, edit, remove items from cart user: 2
add, edit, remove items from cart user: 3
Queue Service 1
Queue Service 2
Queue Service 3
10 minutes user: 1
10 minutes user: 2
10 minutes user: 1
10 minutes user: 2

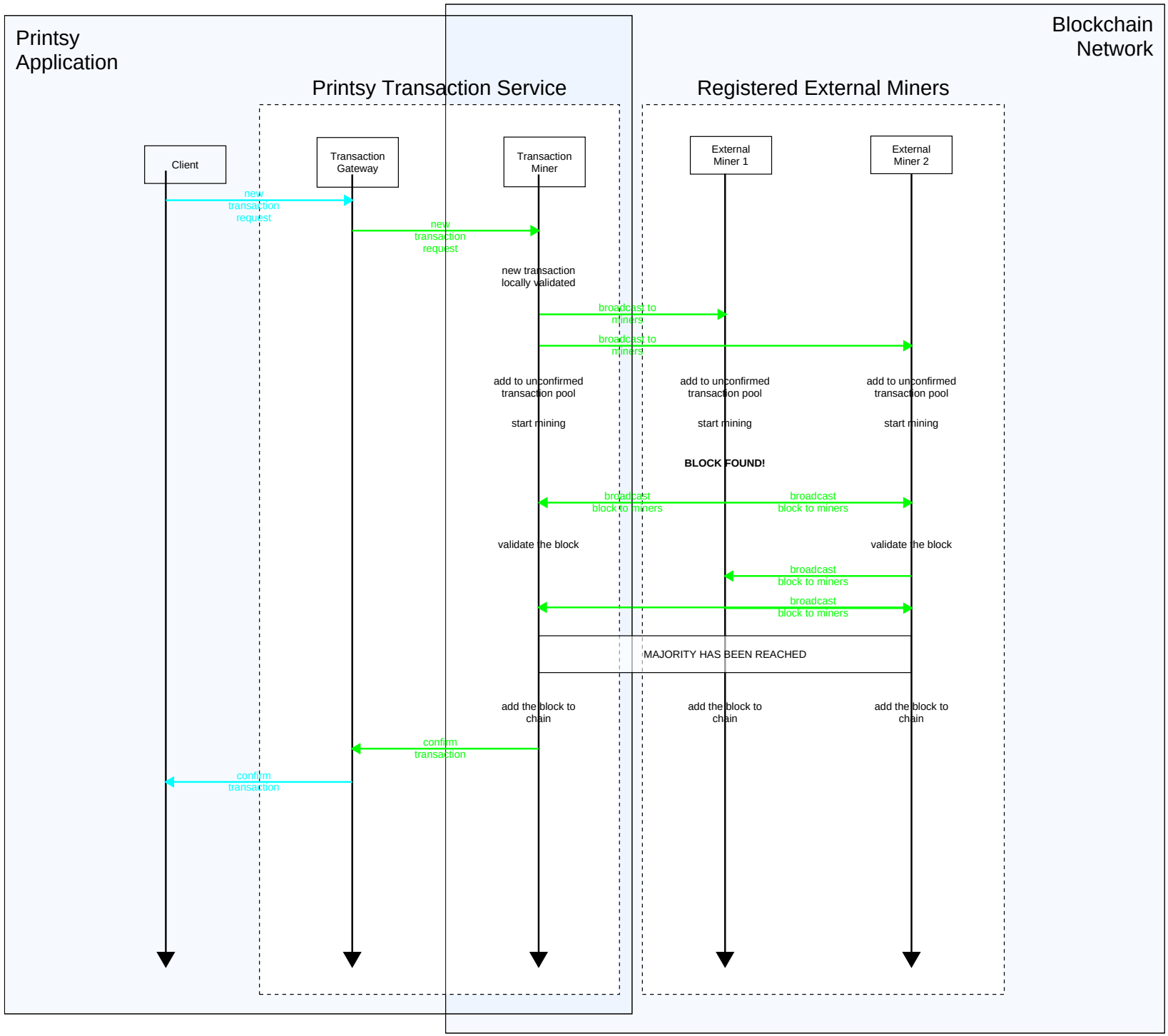
CART SERVICE FINAL

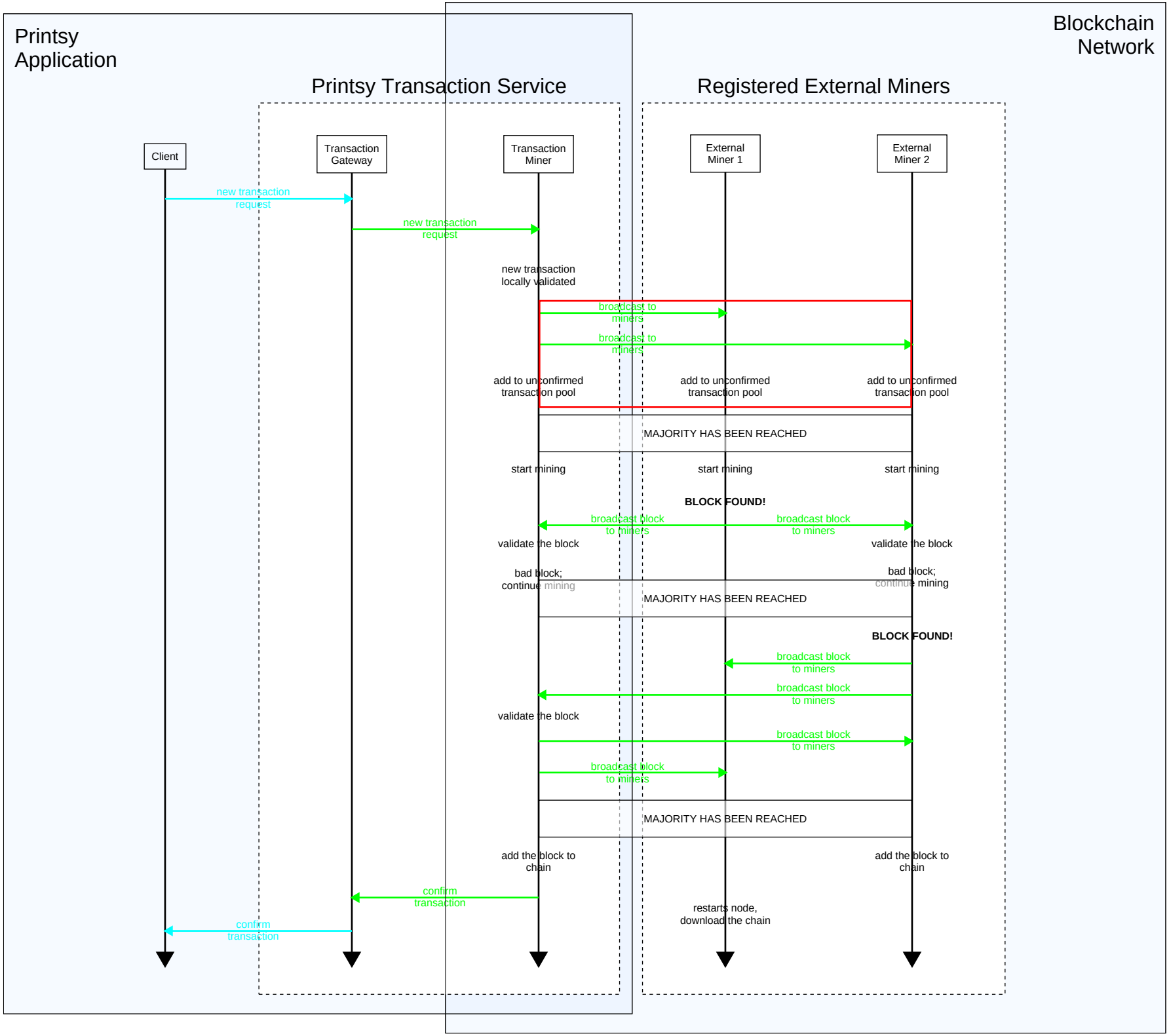

Transaction Service
GraphQL Requests / Responses
Repository / Database Methods
/isAvailable
image: 25
3
5 have been bought
Replicas - the load balancer uses a hashing algorithm to make sure a user always goes to the same node
1
/addToCart
user: 1
stock: 1
image: 25
Frontend
API Gateway
2
user: 1
stock: 1
image: 25
Cart Service 1
Scheduled Delete Task for User 1 - in memory; updated on every request to edit User 1's cart
Cart Service 2
Scheduled Delete Task for User 2 - in memory; updated on every request to edit User 2's cart
Cart Service 3
Scheduled Delete Task for User 3 - in memory; updated on every request to edit User 3's cart
3
checkImagesInCart
deleteCartItems
MySQL DB
cart item added successfully
7
8
cart item added successfully
/addItemToCart
user: 1
stock: 2
image: 25
expiration: UTC time stamp
4
cart item added successfully
6
Replicas - the load balancer uses a hashing algorithm to make sure a cart request for a certain image always goes to the same node
Queue Service 1
Declare a class CartQueue which implements the Queue interface.
Pop requests off the queue, double check that the image count is still < 10, add cart item to database.
Send back a response to the cart service saying success or failure
Queue Service 2
Declare a class CartQueue which implements the Queue interface.
Pop requests off the queue, double check that the image count is still < 10, add cart item to database.
Send back a response to the cart service saying success or failure
Queue Service 3
Declare a class CartQueue which implements the Queue interface.
Pop requests off the queue, double check that the image count is still < 10, add cart item to database.
Send back a response to the cart service saying success or failure
Cart Operations need to be Synchronous (in order) for a given Image. We don't need RabbitMQ or ActiveMQ. A simple Queue in the java program is enough
5
newCartItem
editCartItem (for editing expiration time)

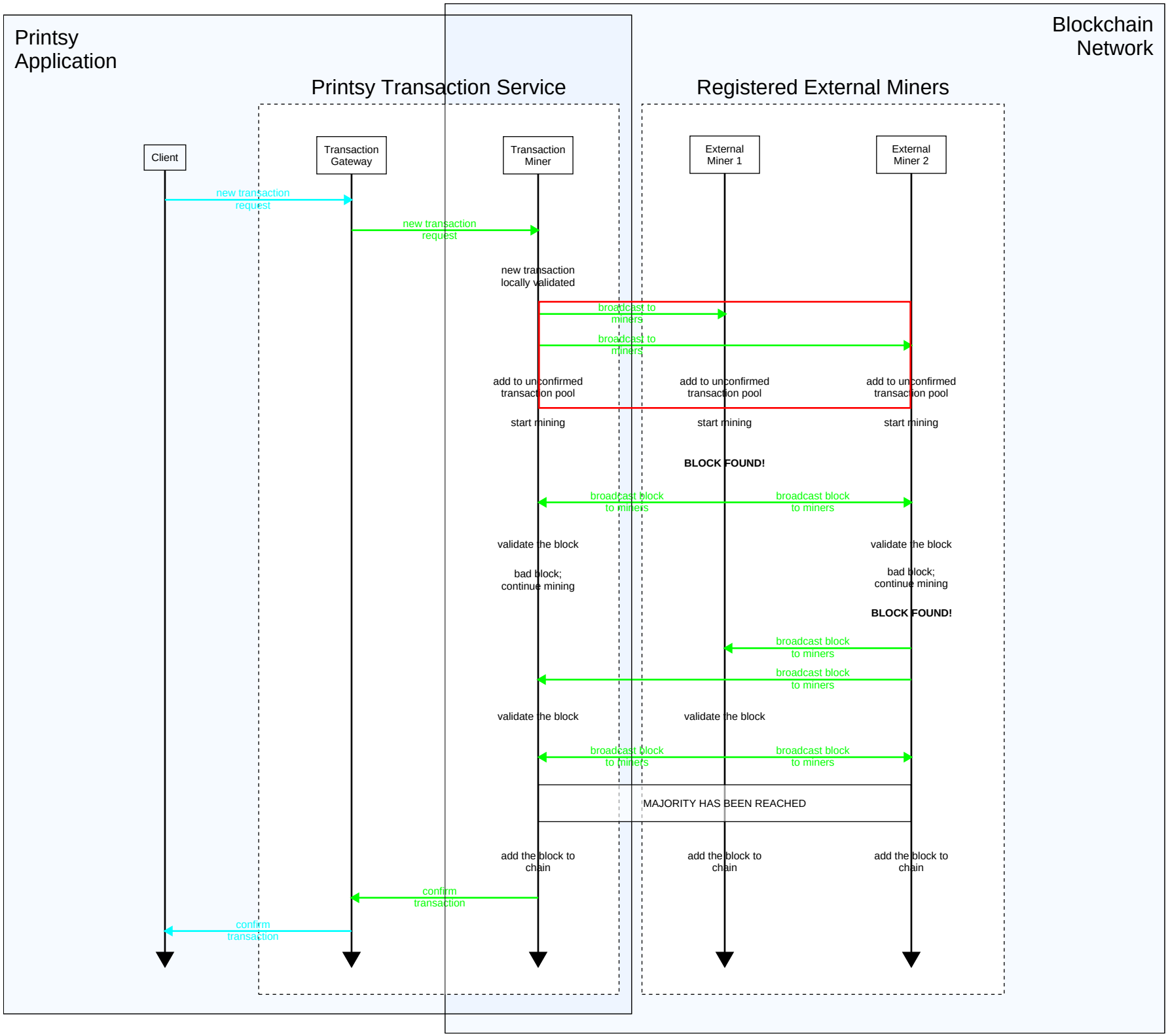
TRANSACTION SERVICE DRAFTS


GraphQL Requests / Responses
gRPC Calls
Do we still need a gateway? we can ask the block directly. advantage is block software can be deployed anywhere. gateway is adapter for our software
Scenarios:
- image25 has never been bought: no record of it on the block, does not exist in the hashmap. /isAvailable returns [MAX]
- image has been bought: hashmap finds block 4 as most recent transaction with image25. This block contains image count, as X. /isAvailable returns X.
1
/isAvailable
image: 25
Cart Service 1
Transaction Gateway
Transaction Worker - L
Transaction Worker C
Transaction Worker B
Transaction Worker A
2
/executeTransaction
image: 25
Scenarios:
- image25 has not been bought MAX times. Blockchain successfully validates transaction with transaction logic, confirms via PoW that transaction can be executed. Adds block to blockchain. Sends back response message.
- image25 has been bought MAX times. Blockchain does not successfully validate transaction with transaction logic. Doesn't get to PoW. Sends back invalid transaction message.

Scenario 1: Transaction is Valid

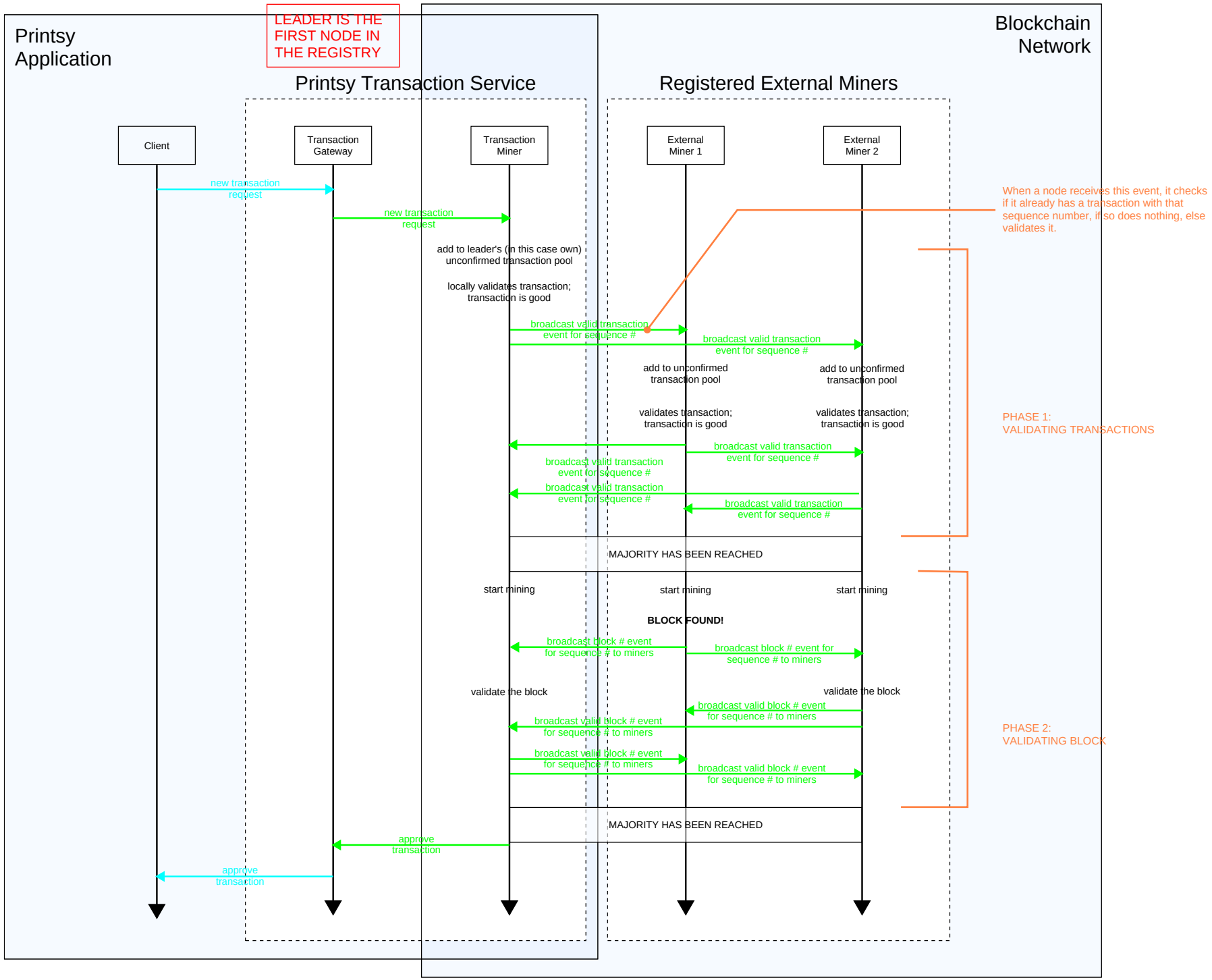



Printsy Application
Blockchain Network
Printsy Transaction Service
Registered External Miners
Client
Transaction Gateway
Transaction Miner
External Miner 1
External Miner 2
new transaction request
new transaction request
new transaction locally validated
broadcast to miners
broadcast to miners
add to unconfirmed transaction pool
add to unconfirmed transaction pool
add to unconfirmed transaction pool
MAJORITY HAS BEEN REACHED
start mining
start mining
start mining
BLOCK FOUND!
broadcast block to miners
broadcast block to miners
validate the block
validate the block
bad block; continue mining
bad block; continue mining
MAJORITY HAS BEEN REACHED
BLOCK FOUND!
broadcast block to miners
broadcast block to miners
validate the block
broadcast block to miners
broadcast block to miners
MAJORITY HAS BEEN REACHED
add the block to chain
add the block to chain
confirm transaction
restarts node, download the chain
confirm transaction

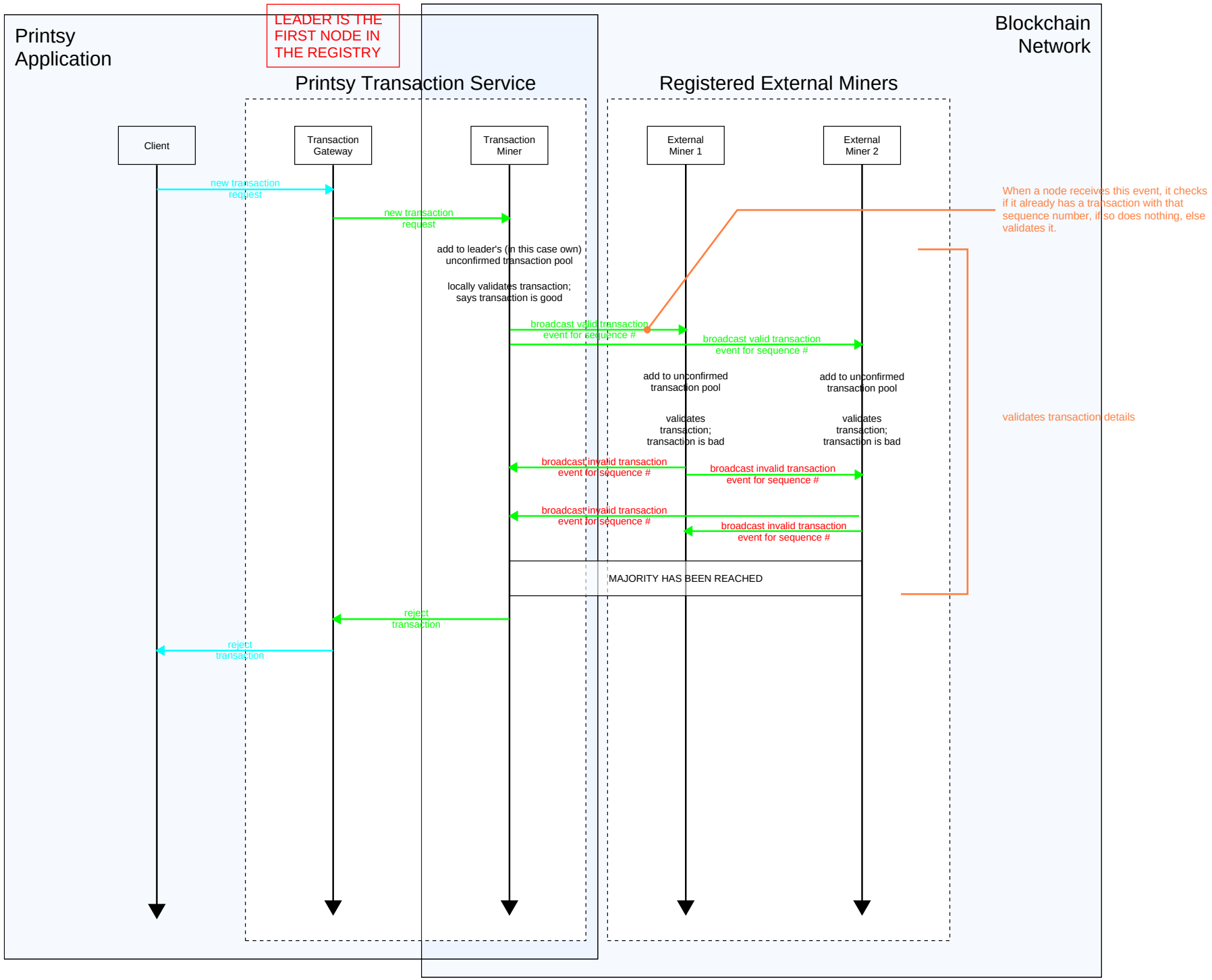
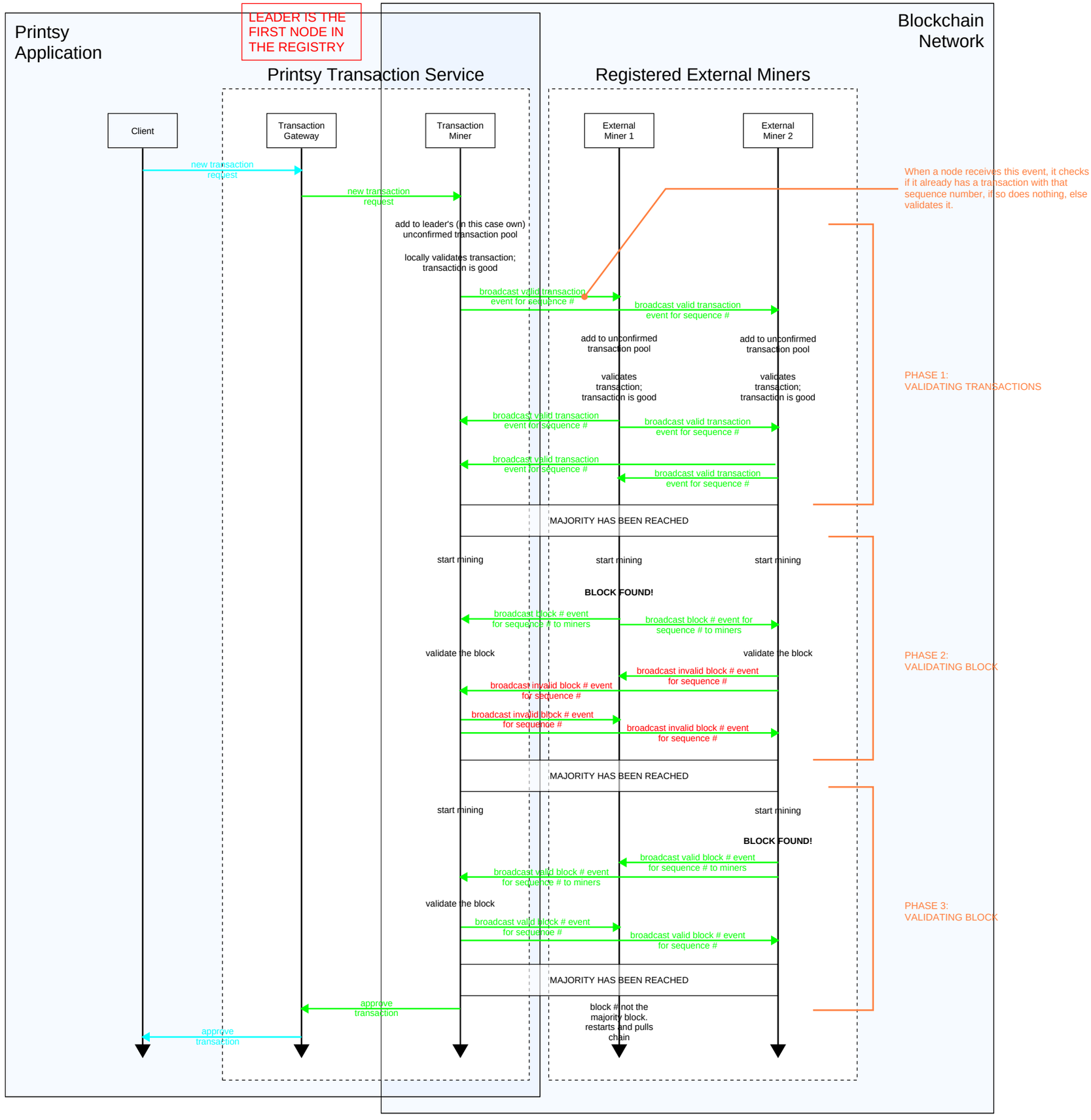

Printsy Application
Blockchain Network
Printsy Transaction Service
Registered External Miners
Client
Transaction Gateway
Transaction Miner
External Miner 1
External Miner 2
new transaction request
new transaction request
new transaction locally validated
broadcast to miners
broadcast to miners
add to unconfirmed transaction pool
add to unconfirmed transaction pool
add to unconfirmed transaction pool
start mining
start mining
start mining
BLOCK FOUND!
broadcast block to miners
broadcast block to miners
validate the block
bad block; continue mining
validate the block
bad block; continue mining
BLOCK FOUND!
broadcast block to miners
broadcast block to miners
validate the block
validate the block
broadcast block to miners
broadcast block to miners
MAJORITY HAS BEEN REACHED
add the block to chain
add the block to chain
add the block to chain
confirm transaction
confirm transaction

TRANSACTION SERVICE FINAL

Scenario 1: Transaction is Valid



Scenario 2: Transaction is Not Valid



Scenario 3: Transaction is Valid, Winning Block is Invalid