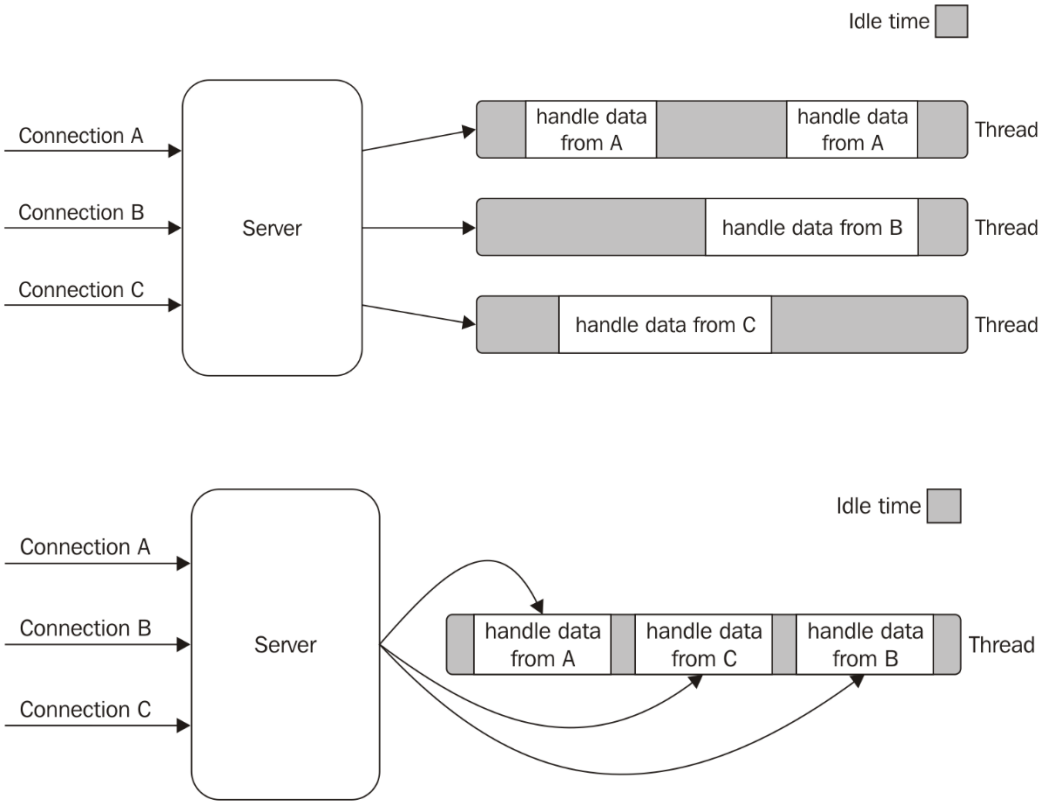
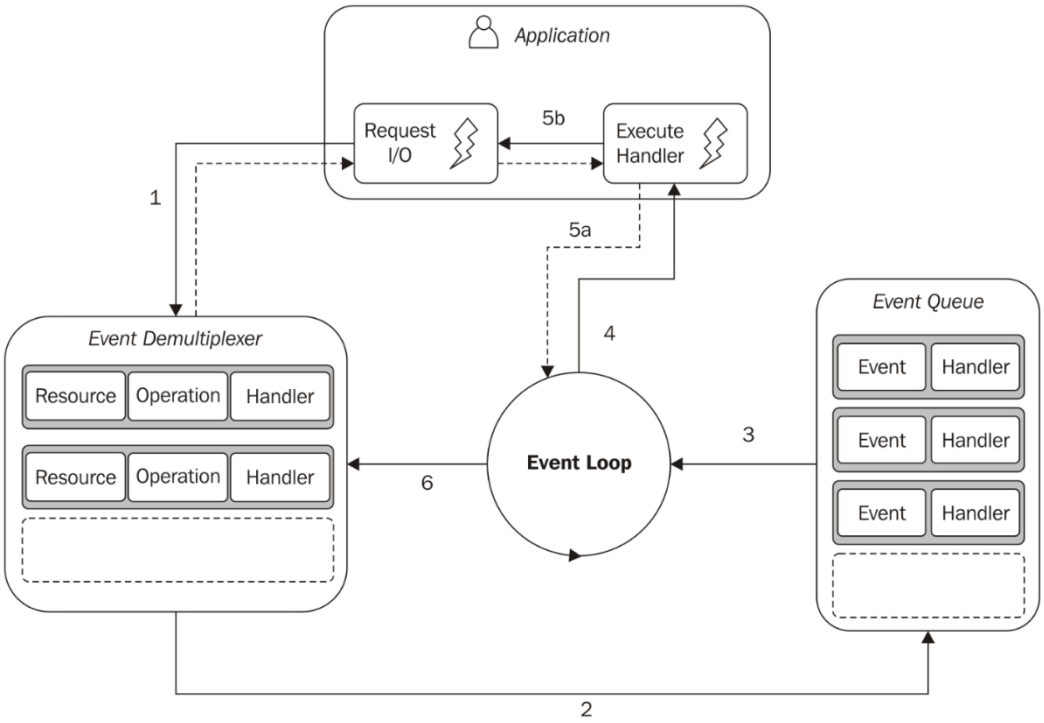
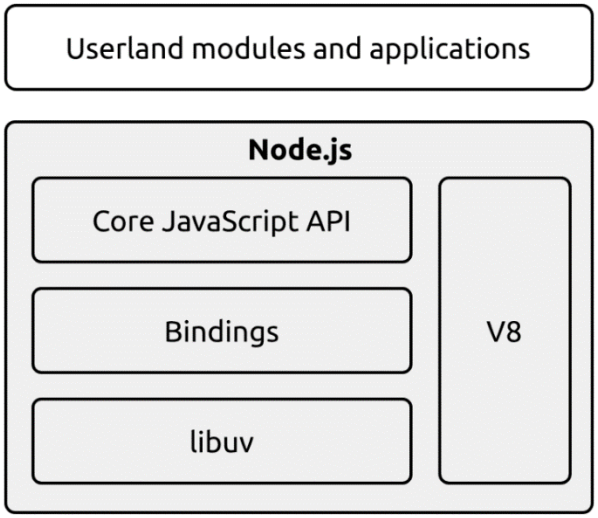
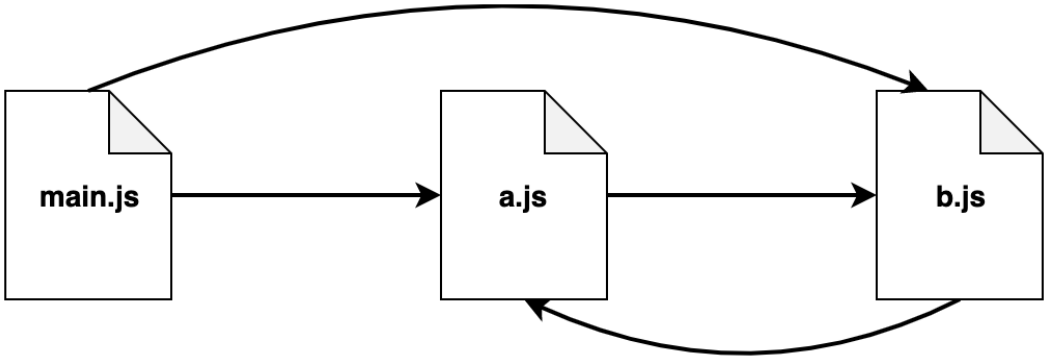
Chapter 01: The Node.js Platform

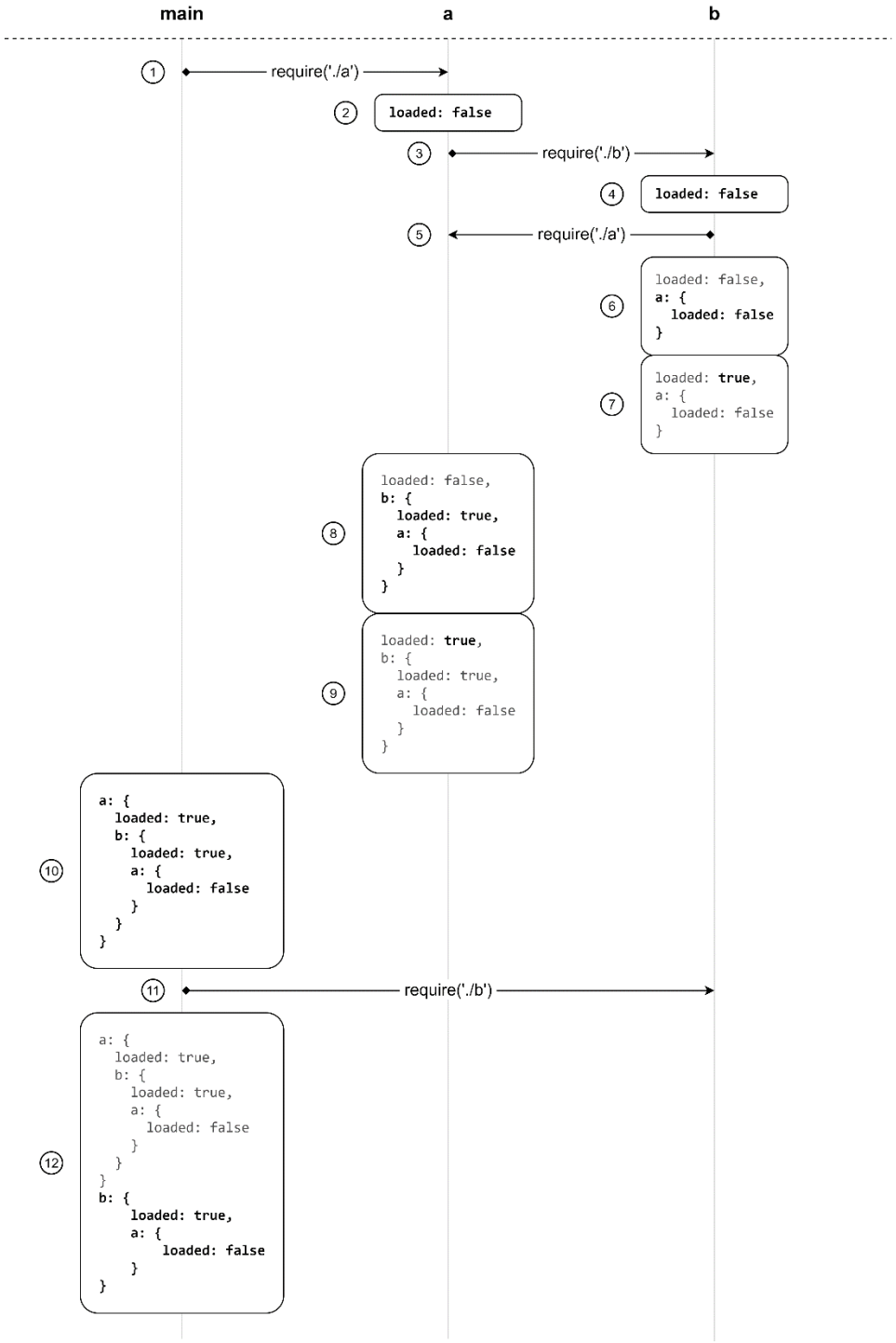


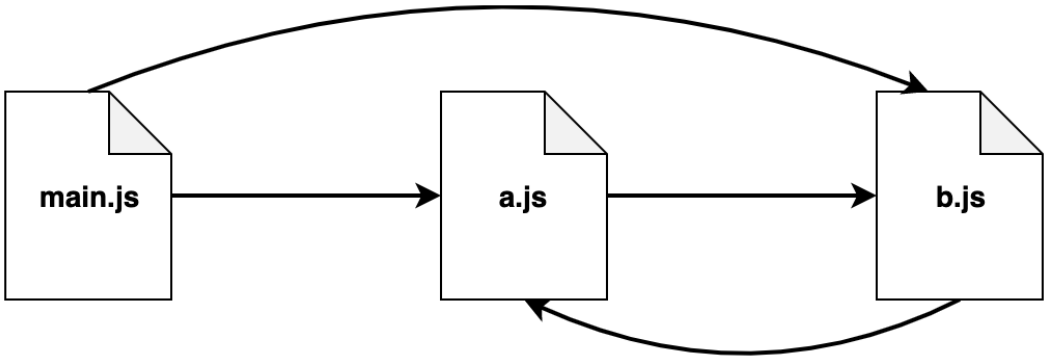

Application
Request I/O
5b
Execute Handler
1
5a
4
Event Demultiplexer
Resource
Operation
Handler
Resource
Operation
Handler
Event Loop
6
3
Event Queue
Event
Handler
Event
Handler
Event
Handler
2

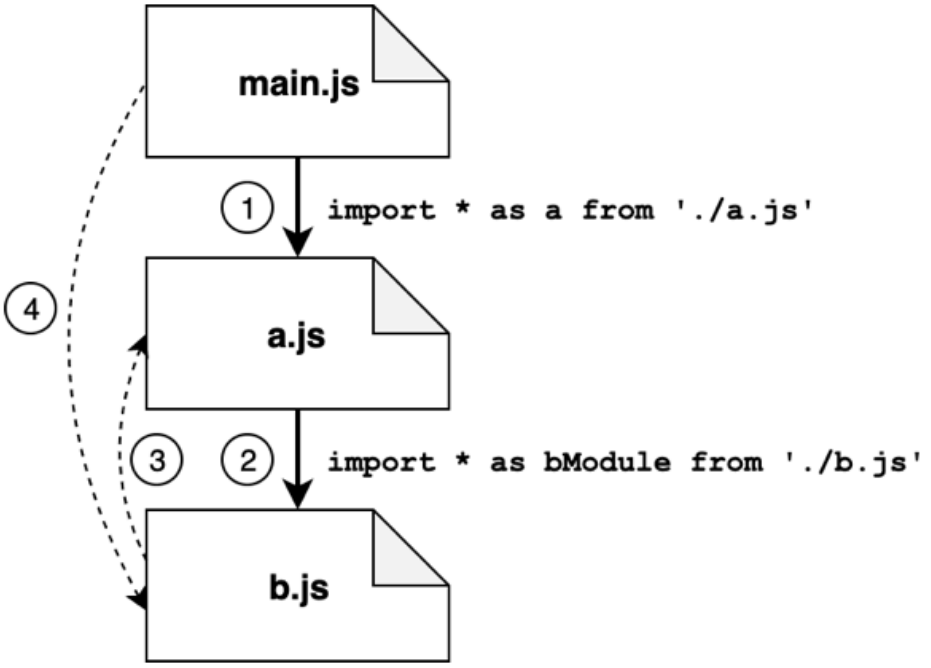

Userland modules and applications
Node.js
Core JavaScript API
Bindings
libuv
V8

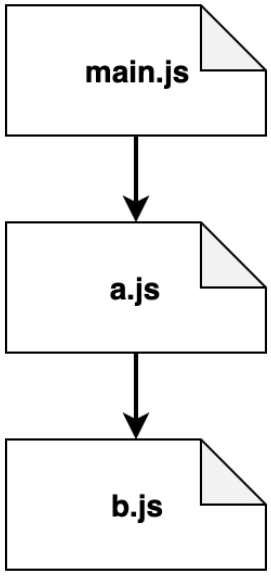
Chapter 02: The Module System

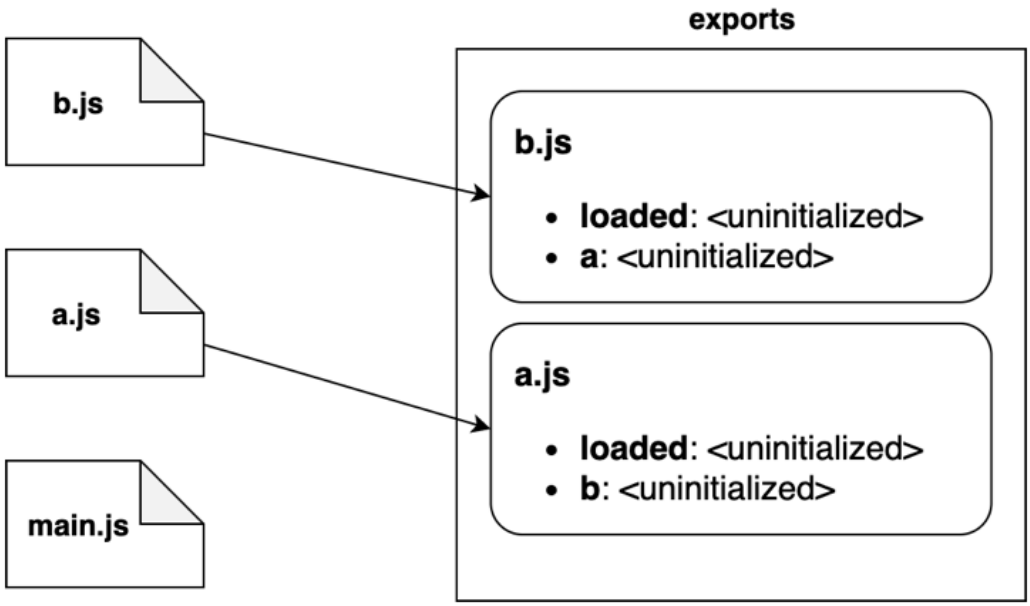


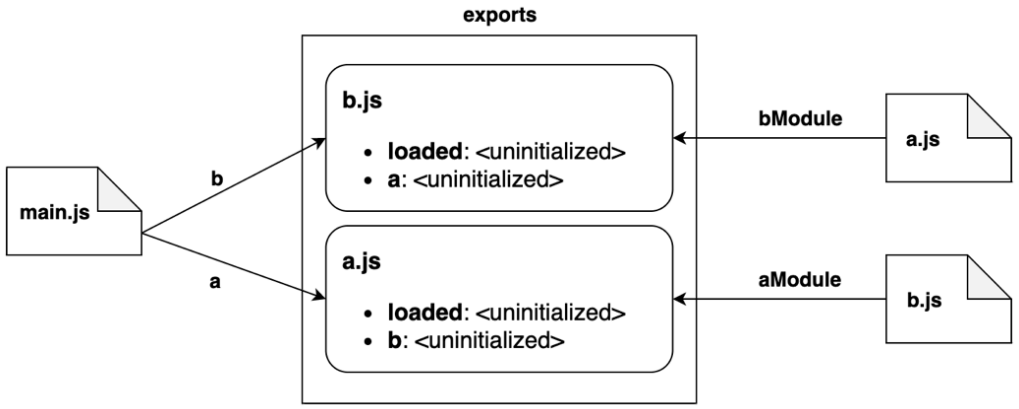

main
a
b
1
require('./a')
2
loaded: false
3
require('./b')
4
loaded: false
5
require('./a')
6
loaded: false,
a: {
loaded: false
}
7
loaded: true,
a: {
loaded: false
}
8
loaded: false,
b: {
loaded: true,
a: {
loaded: false
}
}
9
loaded: true,
b: {
loaded: true,
a: {
loaded: false
}
}
10
a: {
loaded: true,
b: {
loaded: true,
a: {
loaded: false
}
}
}
11
require('./b')
12
a: {
loaded: true,
b: {
loaded: true,
a: {
loaded: false
}
}
}
b: {
loaded: true,
a: {
loaded: false
}
}

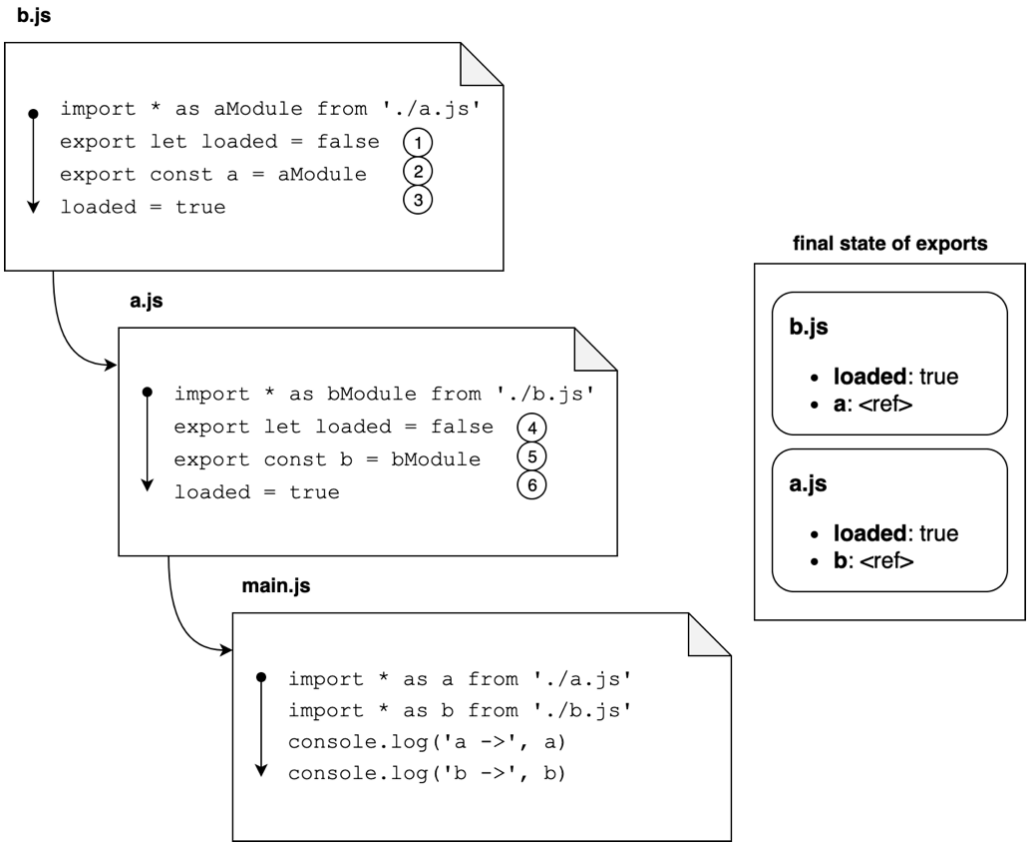

main.js
a.js
b.js


main.js
1
import * as a from './a.js'
a.js
4
3
2
import * as bModule from './b.js'
b.js

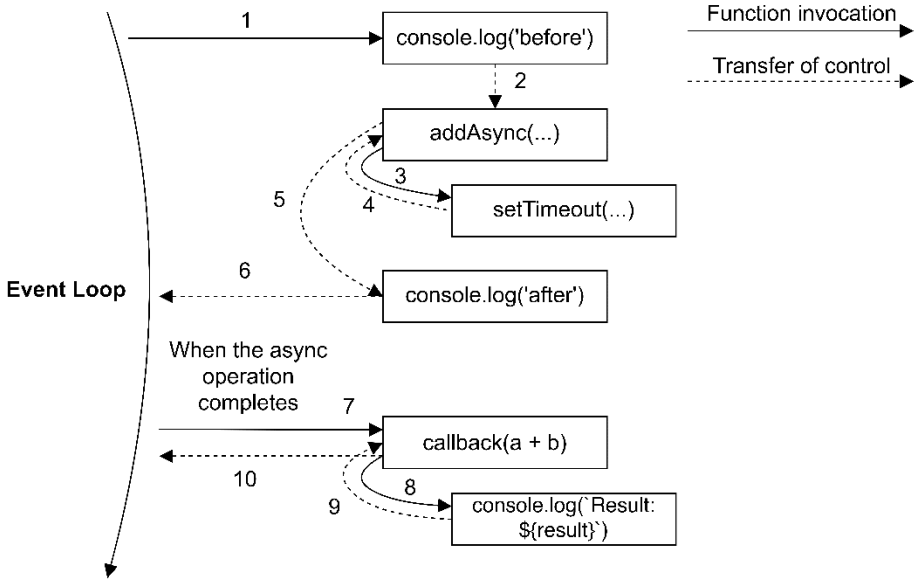

main.js
a.js
b.js

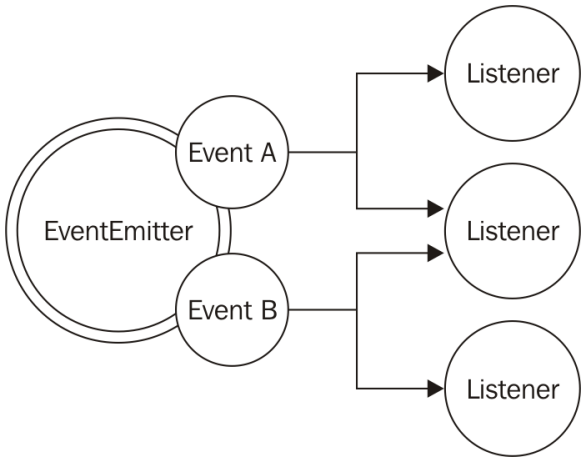

exports
b.js
a.js
main.js
b.js
loaded: <uninitialized>
a: <uninitialized>
a.js
loaded: <uninitialized>
b: <uninitialized>


exports
b.js
loaded: <uninitialized>
a: <uninitialized>
a.js
loaded: <uninitialized>
b: <uninitialized>
bModule
a.js
aModule
b.js
main.js
b
a

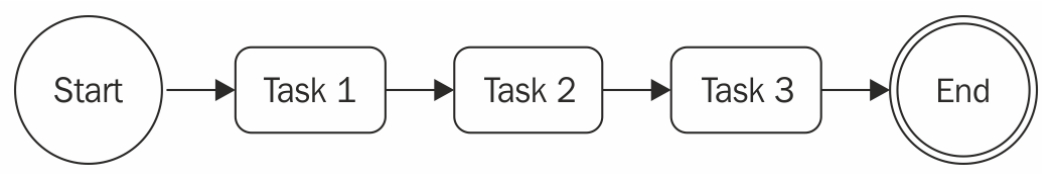

b.js
import * as aModule from './a.js'
export let loaded = false
export const a = aModule
loaded = true
1
2
3
a.js
import * as bModule from './b.js'
export let loaded = false
export const b = bModule
loaded = true
4
5
6
main.js
import * as a from './a.js'
import * as b from './b.js'
console.log('a ->', a)
console.log('b ->', b)
final state of exports
b.js
loaded: true
a: <ref>
a.js
loaded: true
b: <ref>

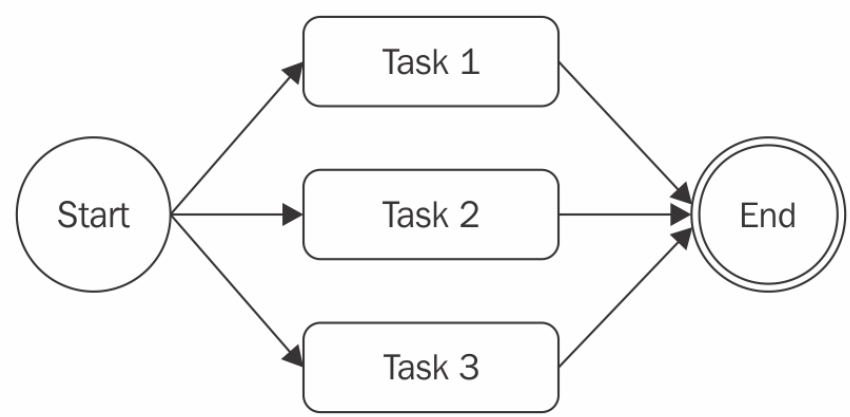
Chapter 03: Callbacks and Events

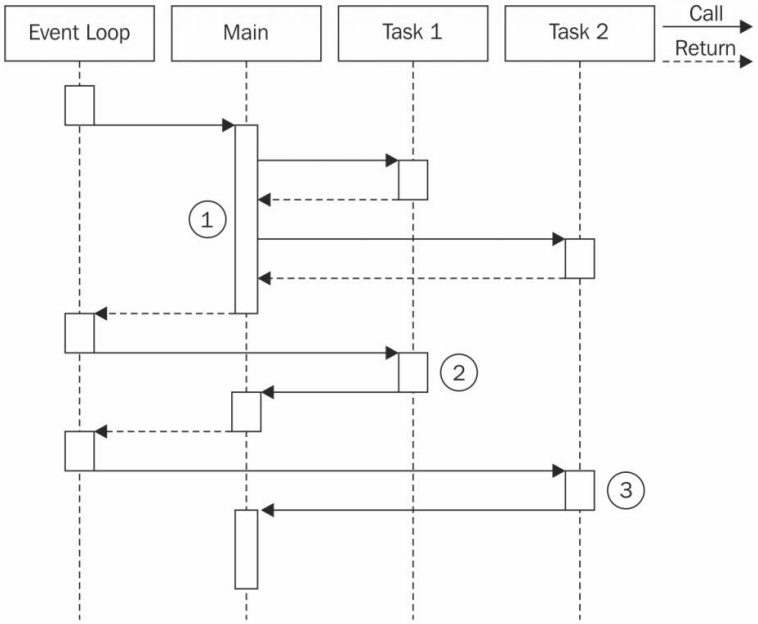


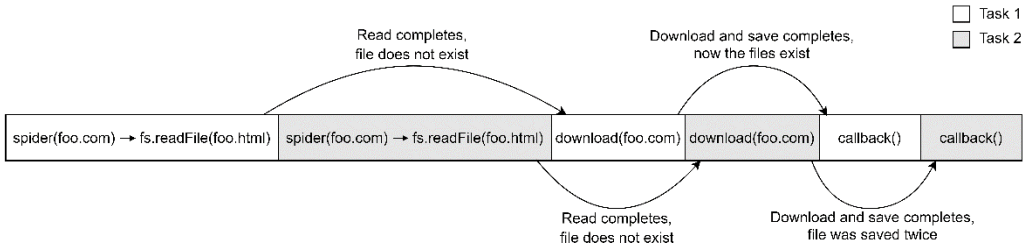


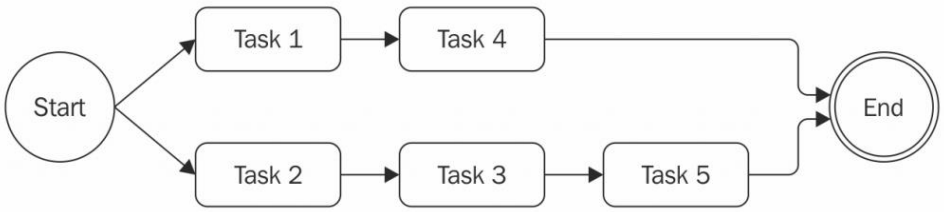
Chapter 04: Asynchronous Control Flow Patterns with Callbacks



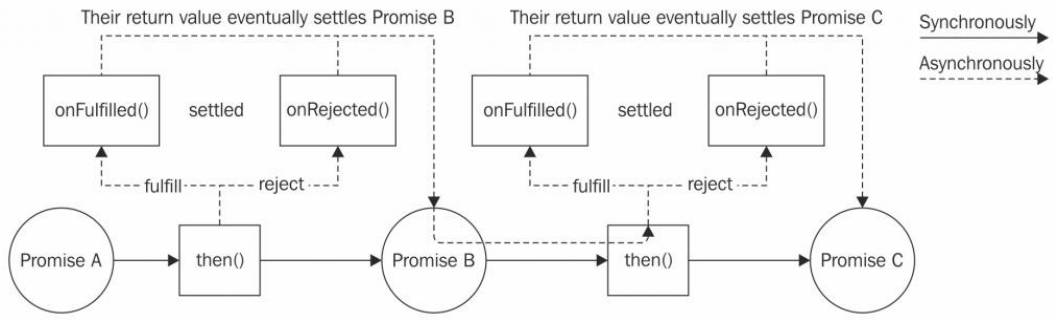


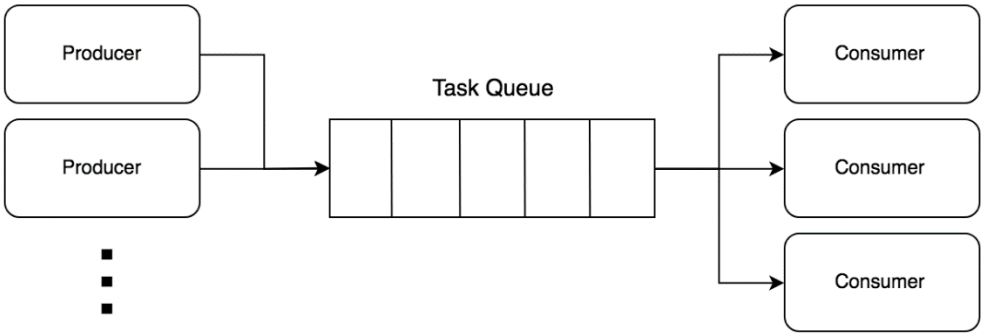

Event Loop
Main
Task 1
Task 2
Call
Return
1
2
3


Task 1
Task 2
Read completes, file does not exist
Download and save completes, now the files exist
spider(foo.com) → fs.readFile(foo.html)
spider(foo.com) → fs.readFile(foo.html)
download(foo.com)
download(foo.com)
callback()
callback()
Read completes, file does not exist
Download and save completes, file was saved twice

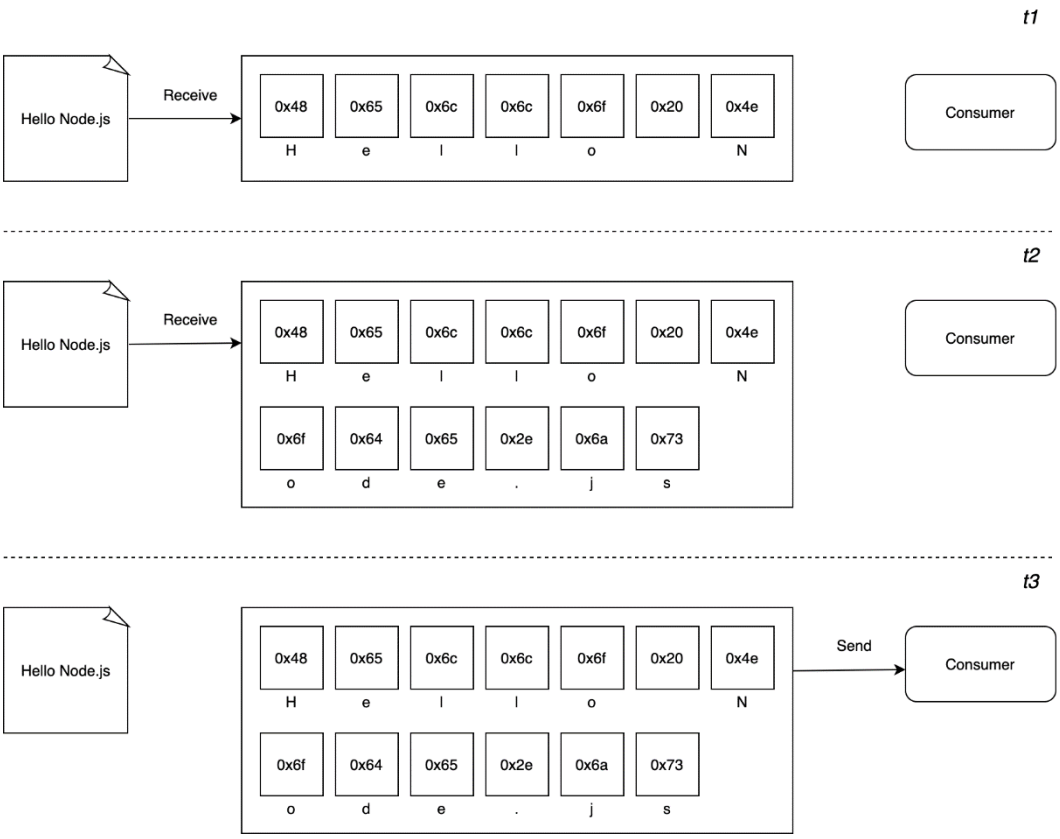

Start
Task 1
Task 4
Task 2
Task 3
Task 5
End

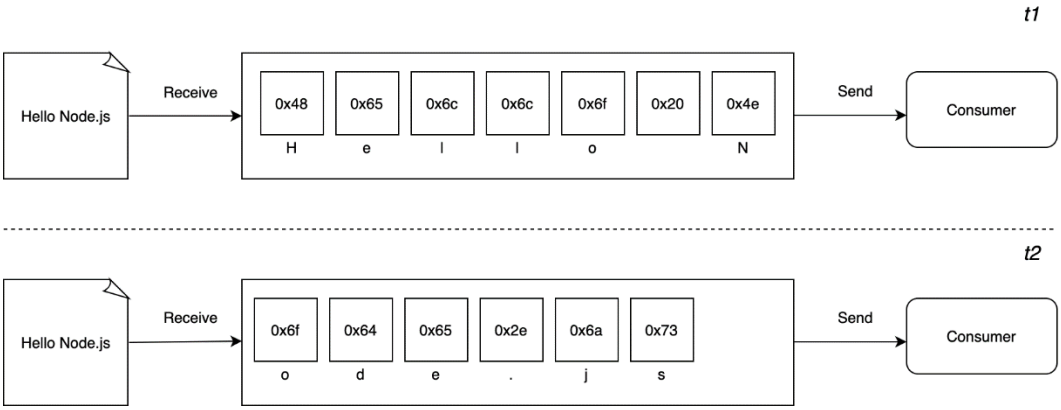
Chapter 05: Asynchronous Control Flow Patterns with Promises and Async/Await

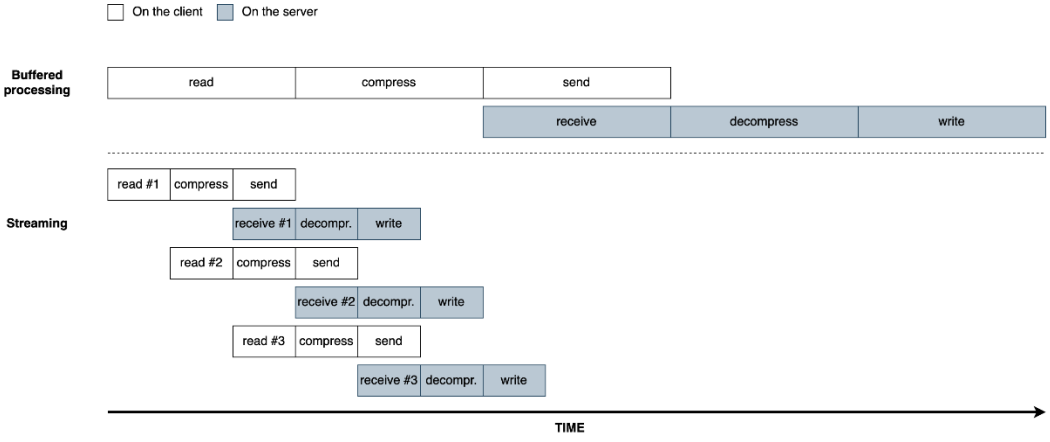


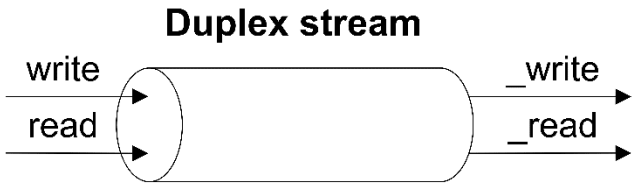


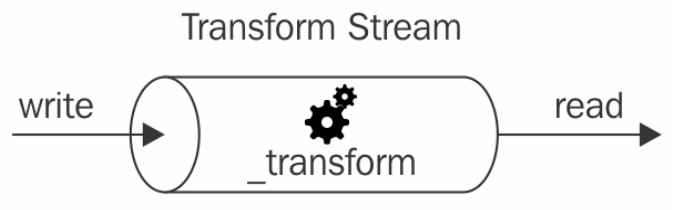
Chapter 06: Coding with Streams

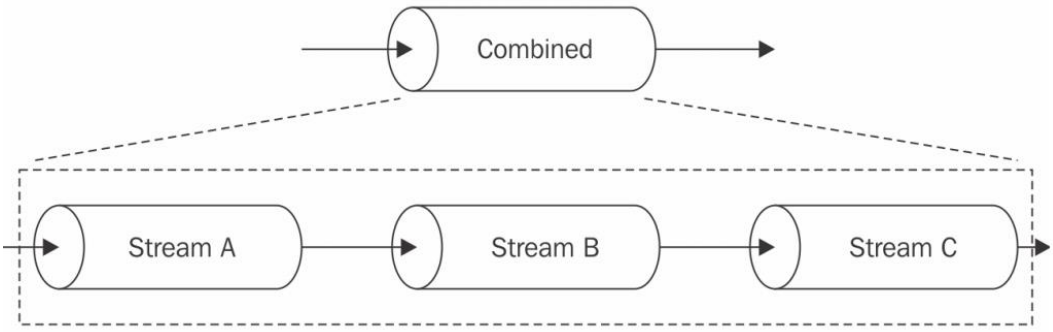


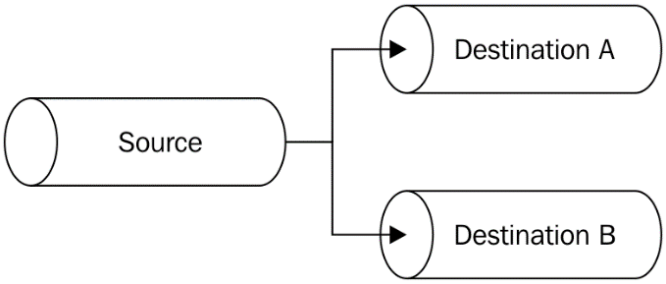

t1
Hello Node.js
Receive
0x48
0x65
0x6c
0x6c
0x6f
0x20
0x4e
H
e
l
l
o
N
Send
Consumer
t2
Hello Node.js
Receive
0x6f
0x64
0x65
0x2e
0x6a
0x73
o
d
e
.
j
s
Send
Consumer

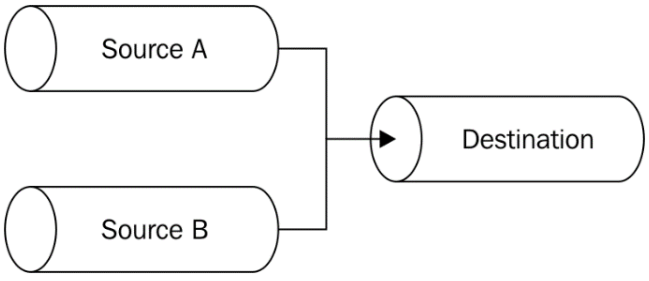

On the client
On the server
Buffered processing
read
compress
send
receive
decompress
write
Streaming
read #1
compress
send
receive #1
decompr.
write
read #2
compress
send
receive #2
decompr.
write
read #3
compress
send
receive #3
decompr.
write
TIME

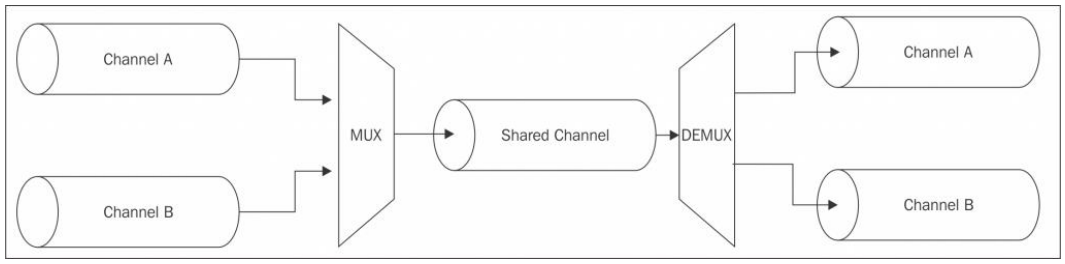

Duplex stream
write
read
_write
_read

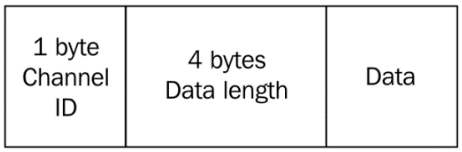

Transform Stream
write
_transform
read

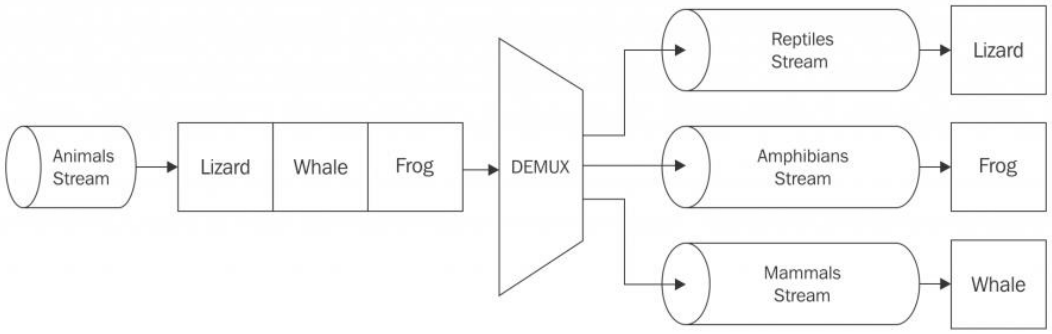

Combined
Stream A
Stream B
Stream C


Source
Destination A
Destination B

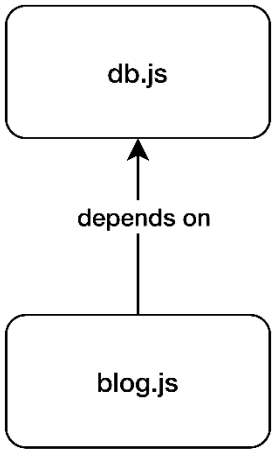

Source A
Source B
Destination

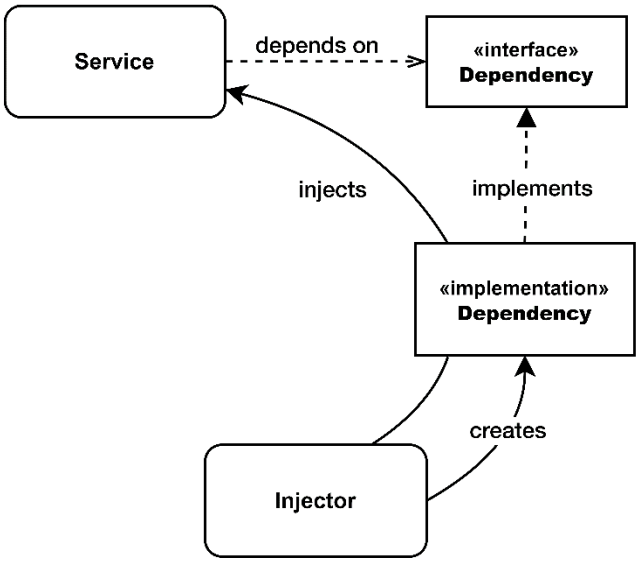

Channel A
Channel B
MUX
Shared Channel
DEMUX
Channel A
Channel B


1 byte
Channel
ID
4 bytes
Data length
Data

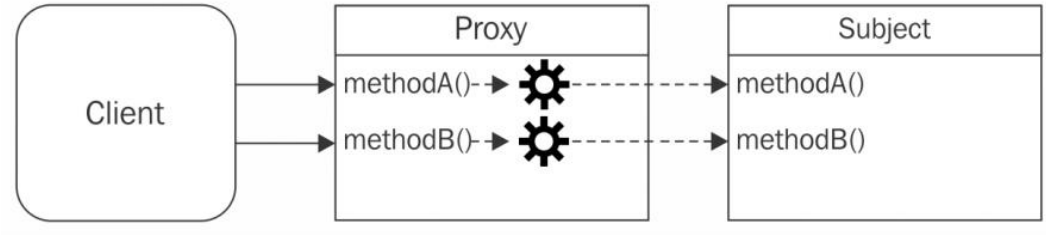

Animals
Stream
Lizard
Whale
Frog
DEMUX
Reptiles
Stream
Amphibians
Stream
Mammals
Stream
Lizard
Frog
Whale

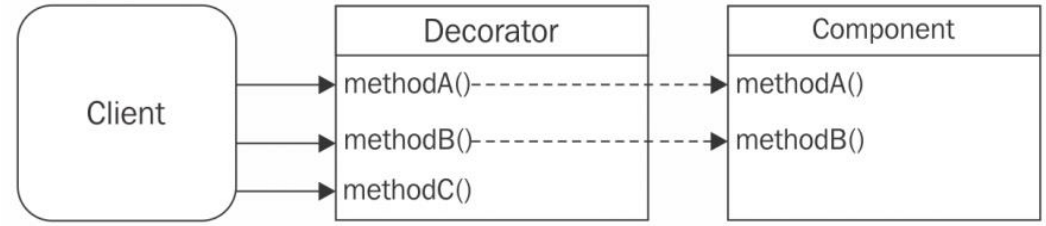
Chapter 07: Creational Design Patterns

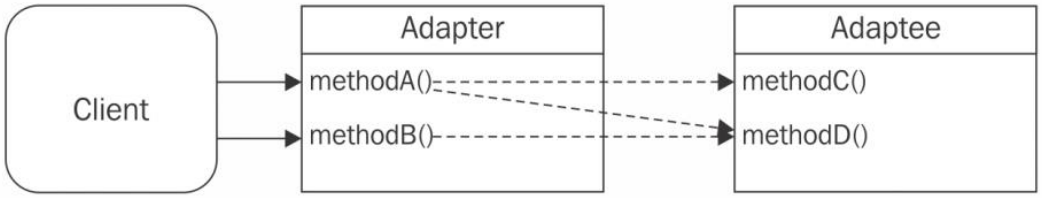




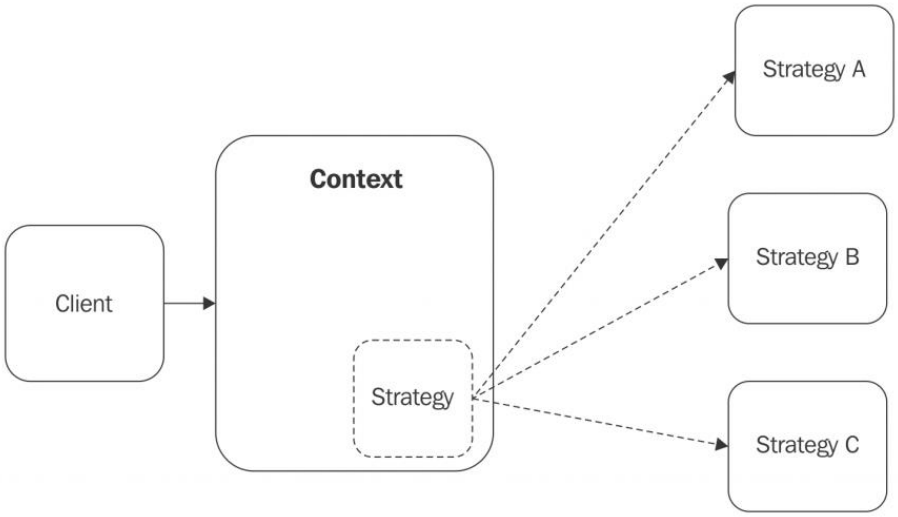
Chapter 08: Structural Design Patterns

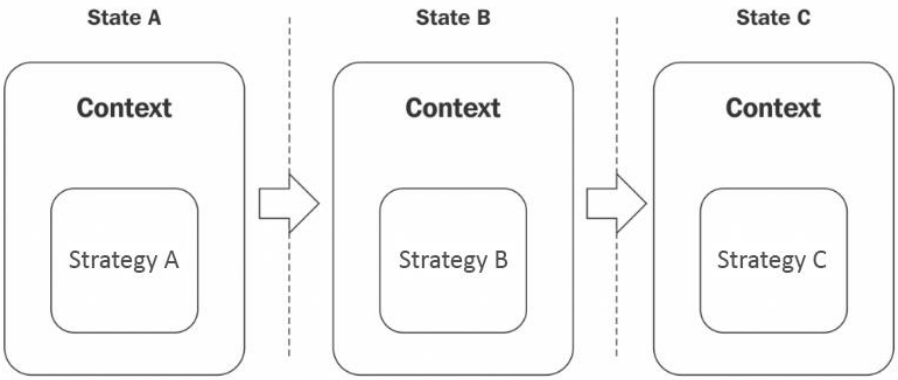


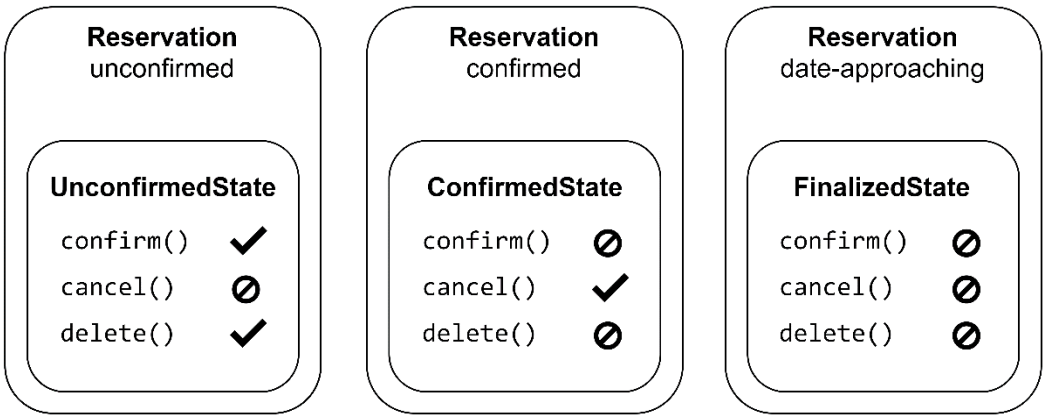


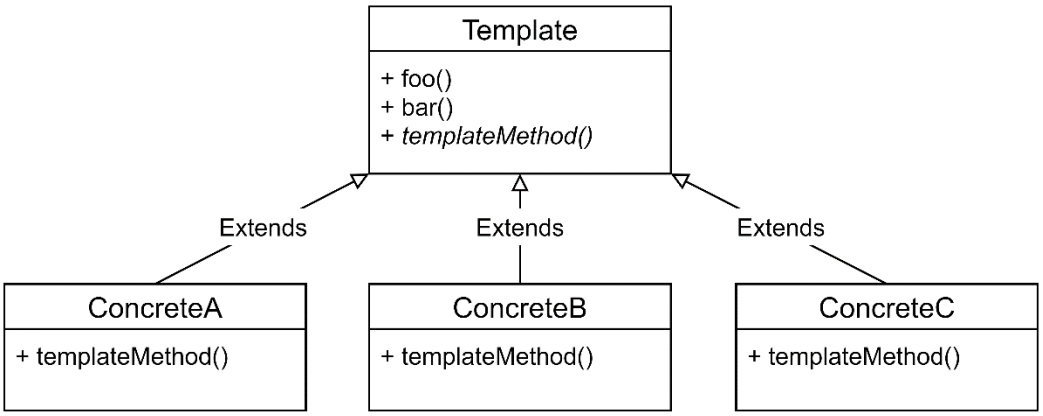


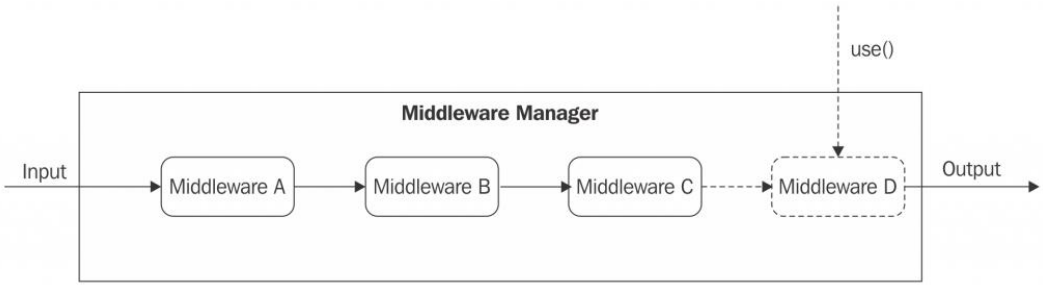
Chapter 09: Behavioral Design Patterns

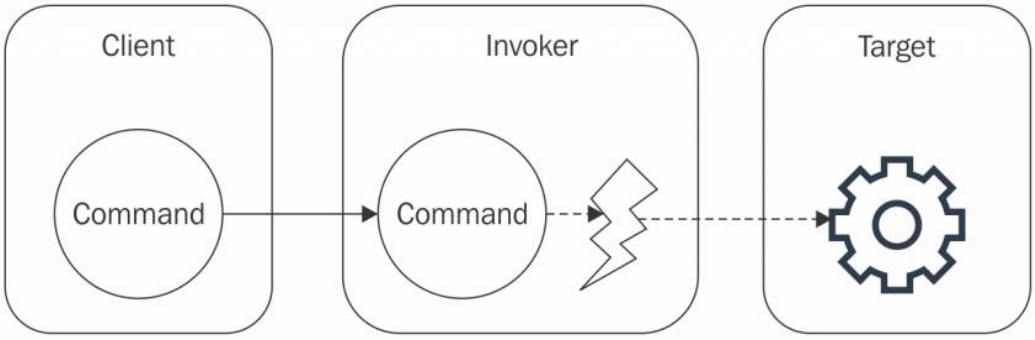




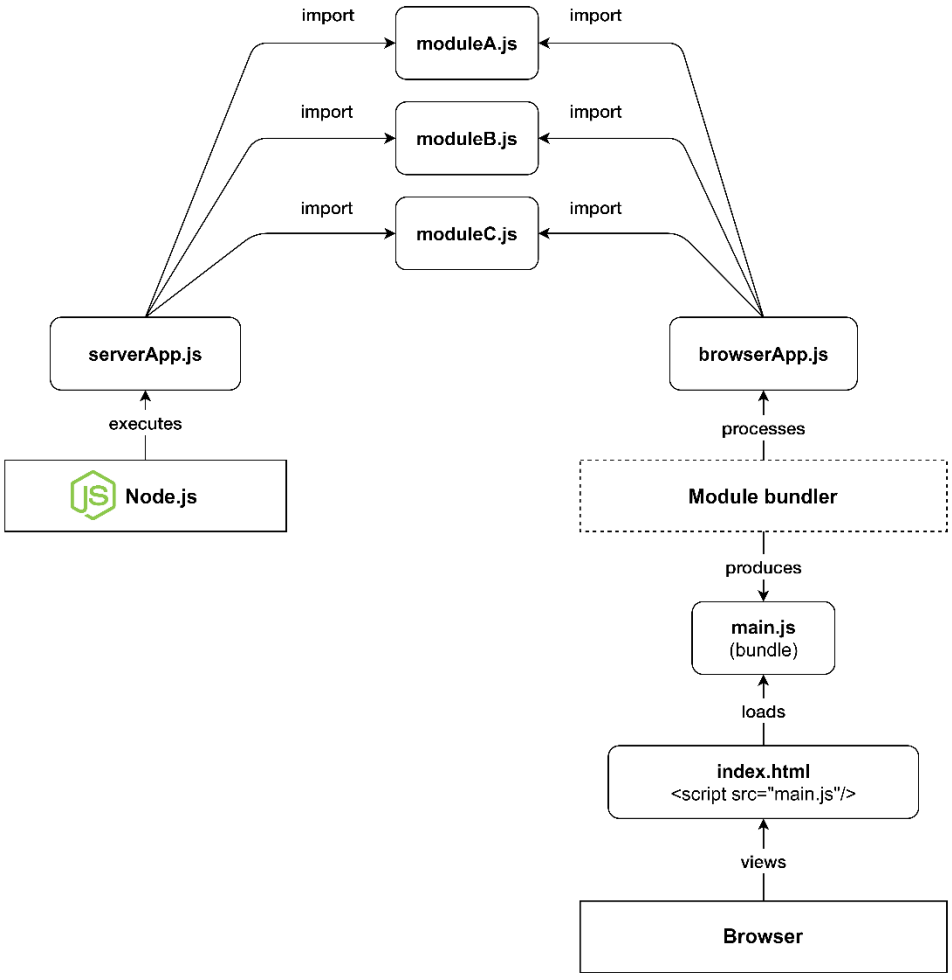

Reservation
unconfirmed
UnconfirmedState
confirm()
cancel()
delete()
Reservation
confirmed
ConfirmedState
confirm()
cancel()
delete()
Reservation
date-approaching
FinalizedState
confirm()
cancel()
delete()

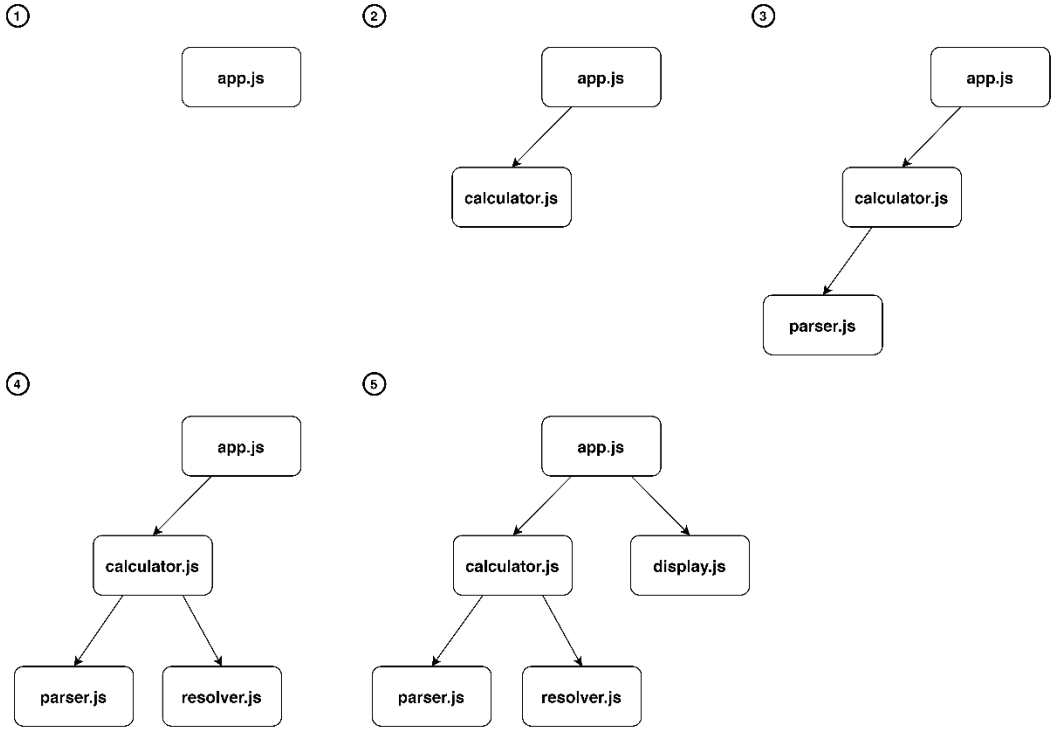

Template
+ foo()
+ bar()
+ templateMethod()
Extends
Extends
Extends
ConcreteA
+ templateMethod()
ConcreteB
+ templateMethod()
ConcreteC
+ templateMethod()

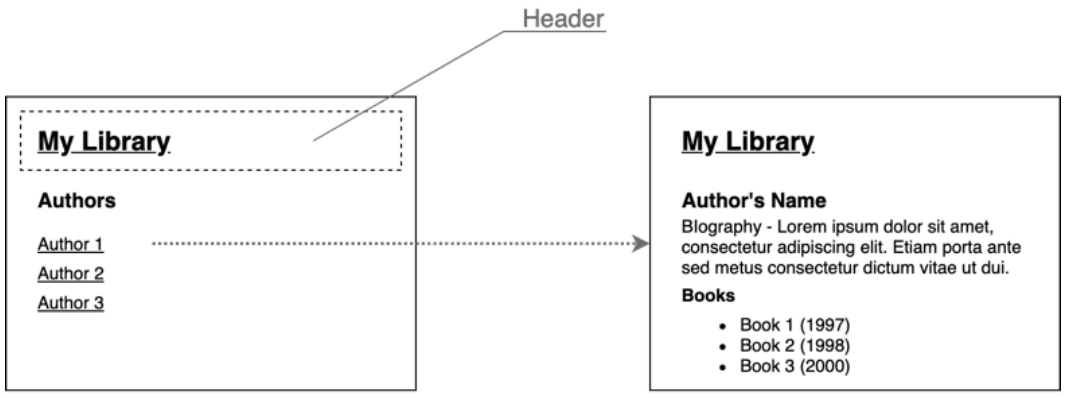

use()
Middleware Manager
Input
Middleware A
Middleware B
Middleware C
Middleware D
Output


Client
Command
Invoker
Command
Target

Chapter 10: Universal JavaScript for Web Applications

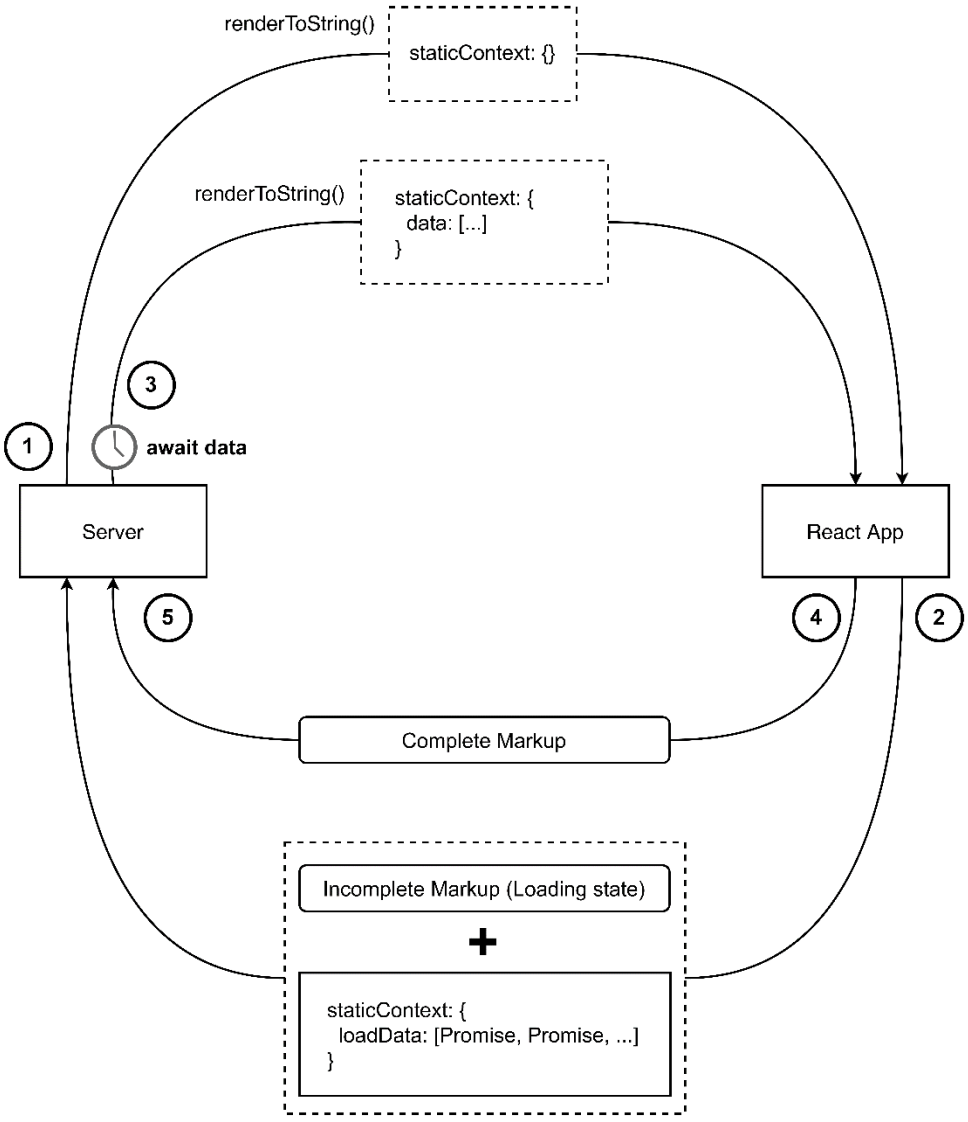


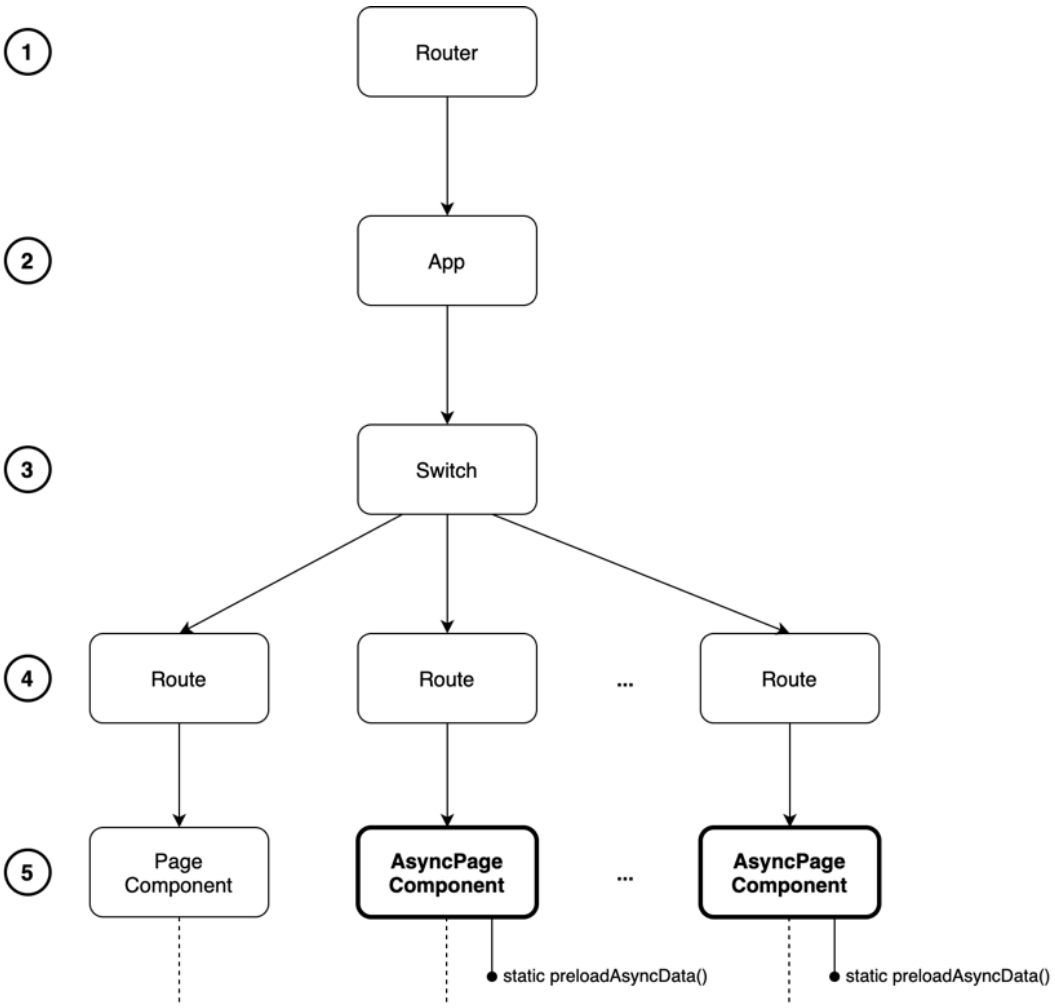




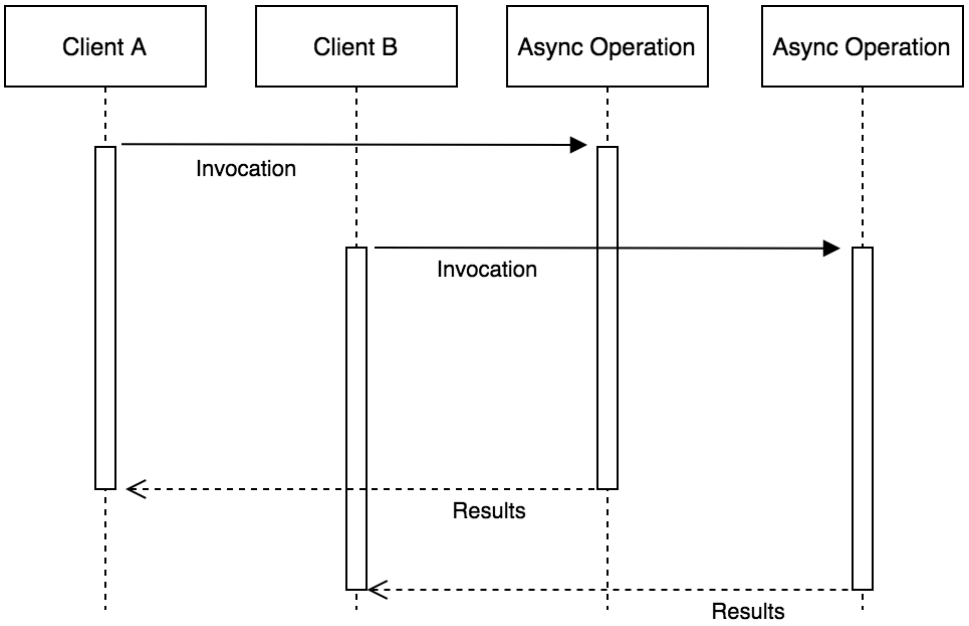
Authors Index Page

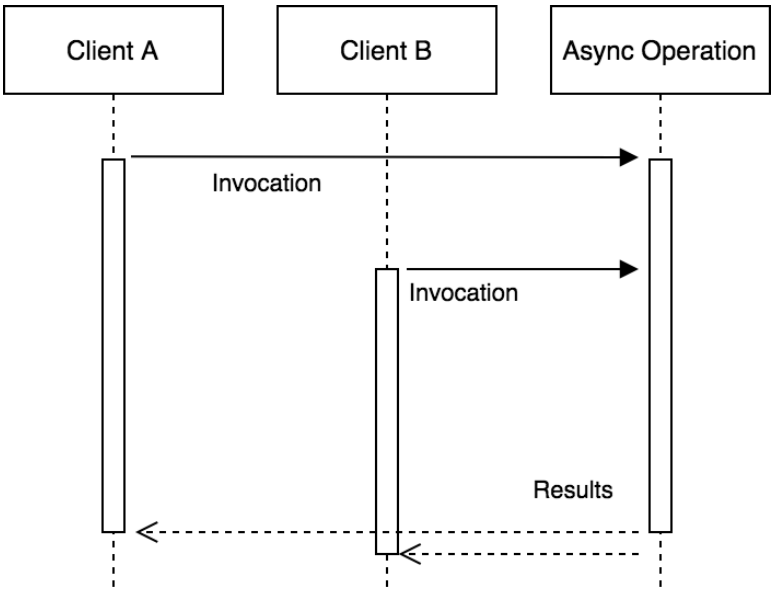
Author's Page

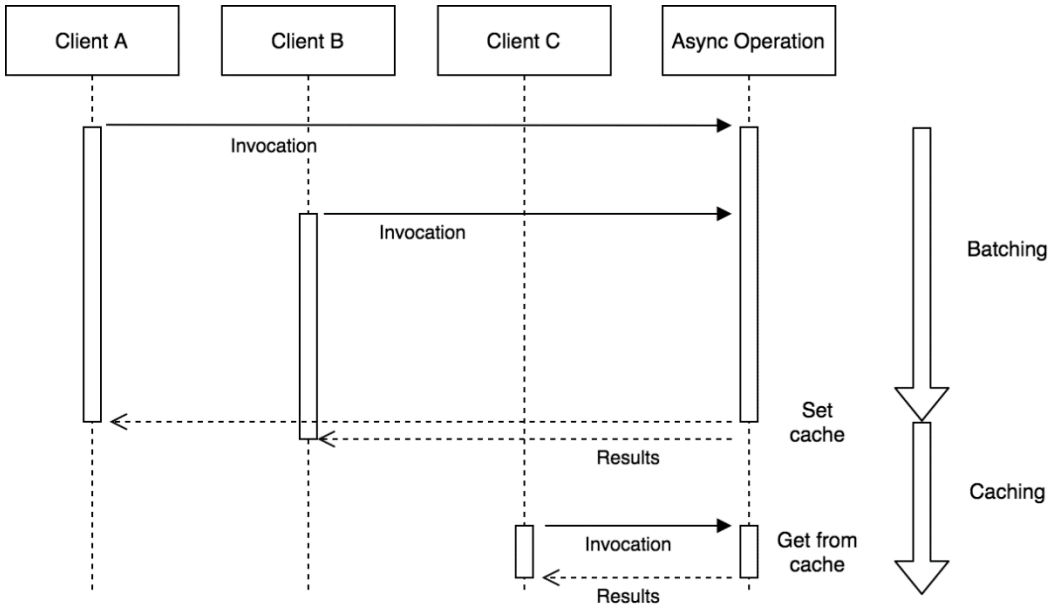

renderToString()
staticContext: {}
renderToString()
staticContext: {
data: [...]
}
3
1
await data
Server
React App
5
4
2
Complete Markup
Incomplete Markup (Loading state)
+
staticContext: {
loadData: [Promise, Promise, ...]
}


1
Router
2
App
3
Switch
4
Route
Route
...
Route
5
Page Component
AsyncPage Component
...
AsyncPage Component
static preloadAsyncData()
static preloadAsyncData()

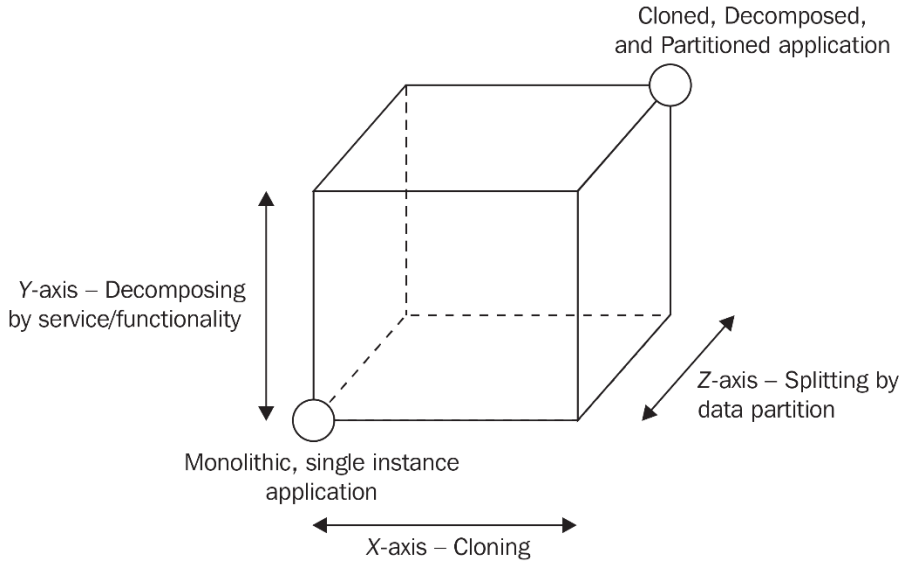
Chapter 11: Advanced Recipes

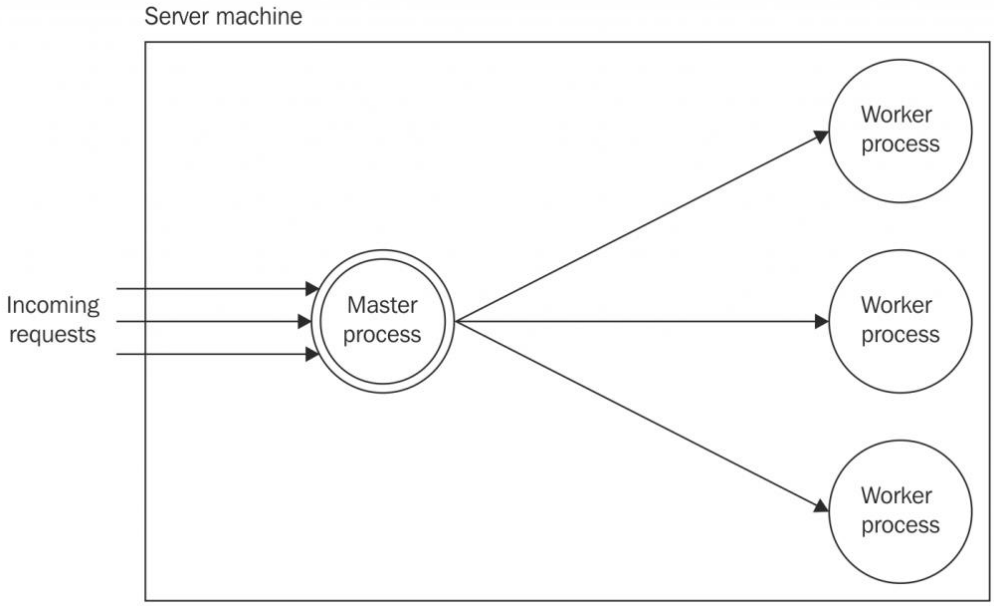


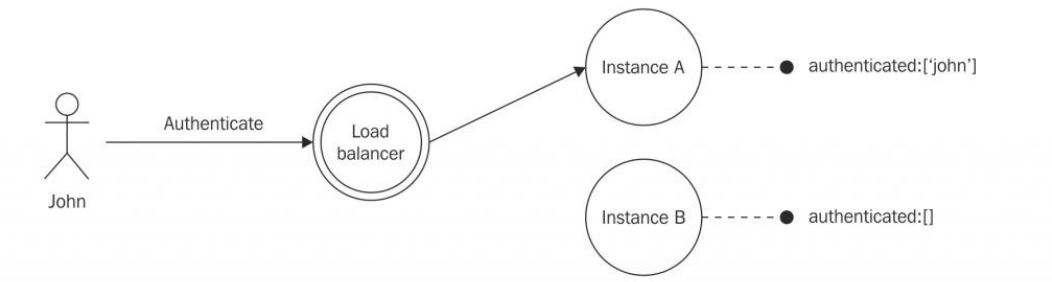


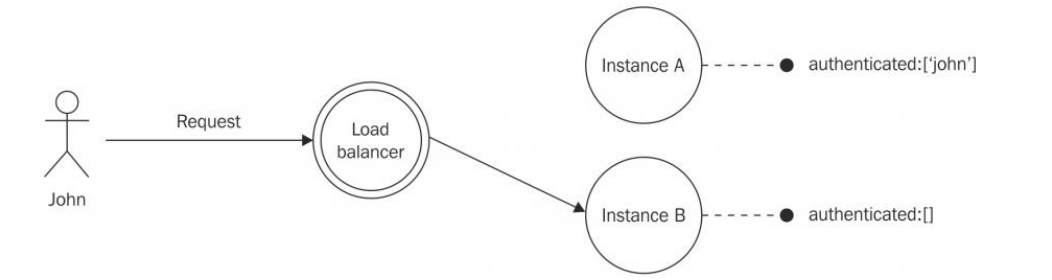

Client A
Client B
Client C
Async Operation
Invocation
Invocation
Batching
Set cache
Results
Caching
Invocation
Get from cache
Results

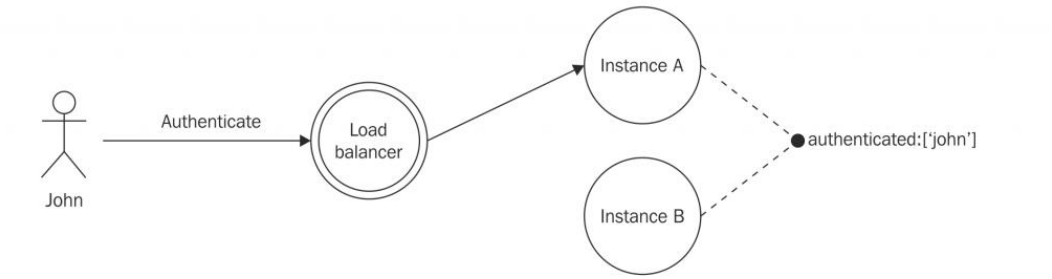
Chapter 12: Scalability and Architectural Patterns

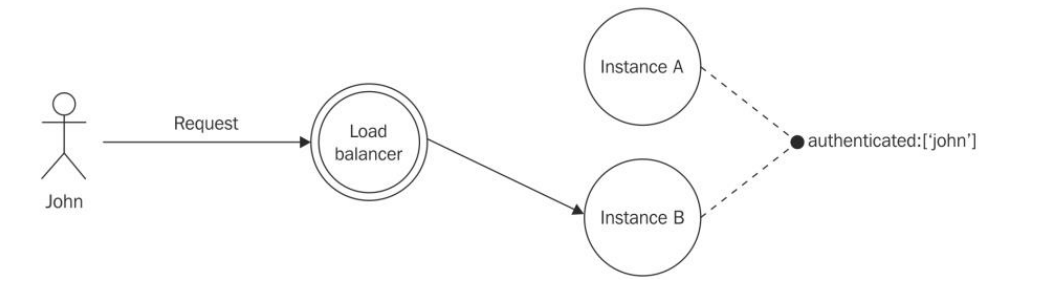


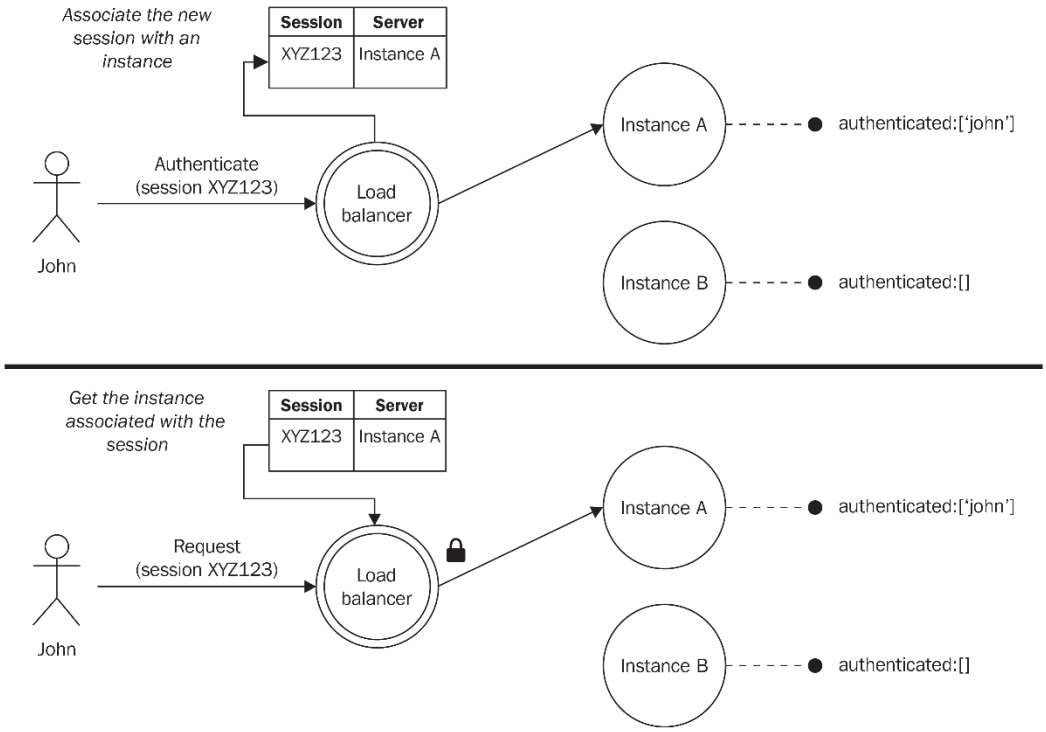


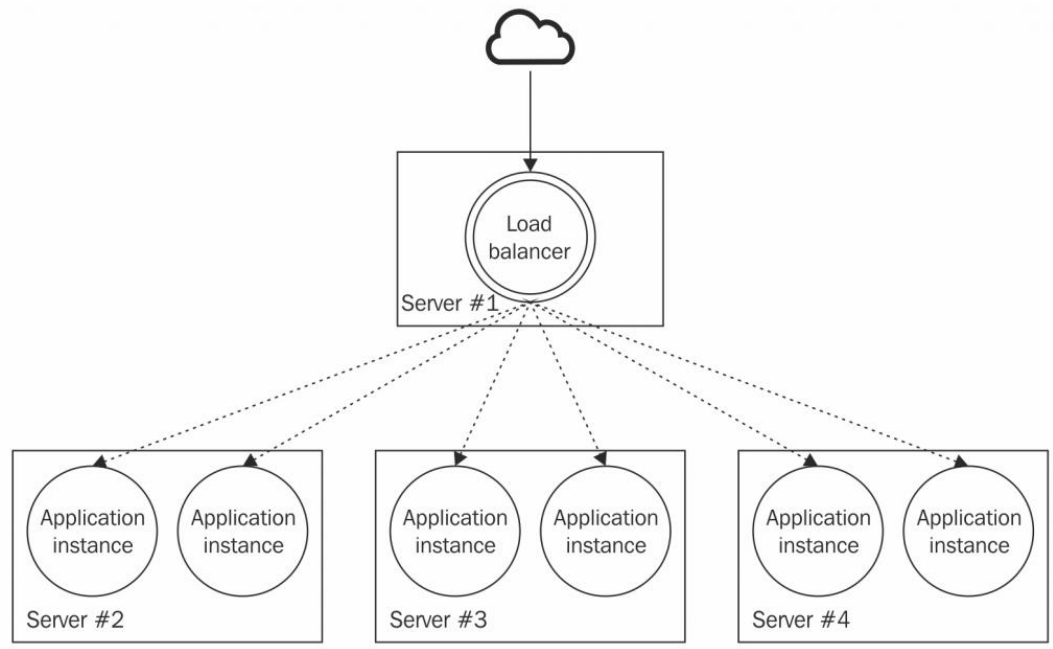

Authenticate
Load balancer
John
Instance A
authenticated:['john']
Instance B
authenticated:[]

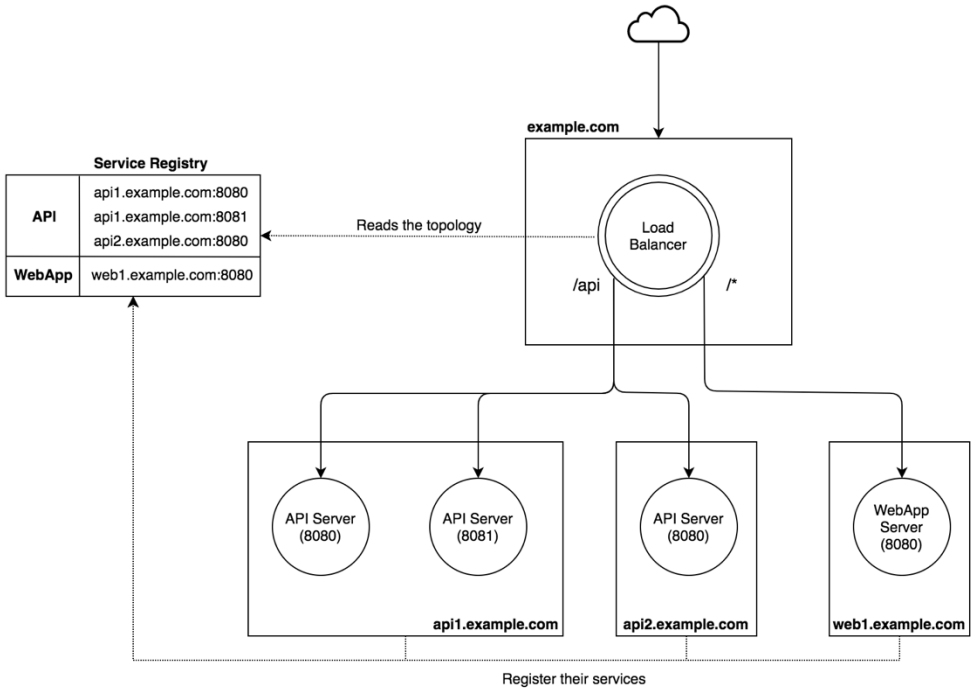

Request
Load balancer
John
Instance A
authenticated:['john']
Instance B
authenticated:[]

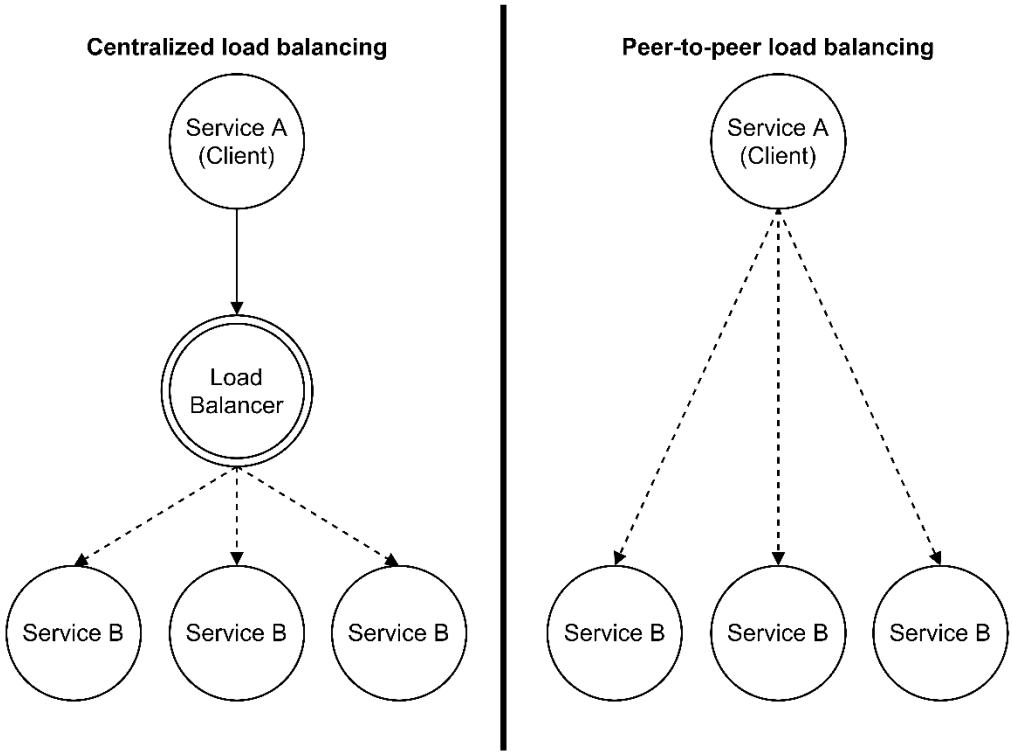

Authenticate
Load balancer
John
Instance A
Instance B
authenticated:['john']

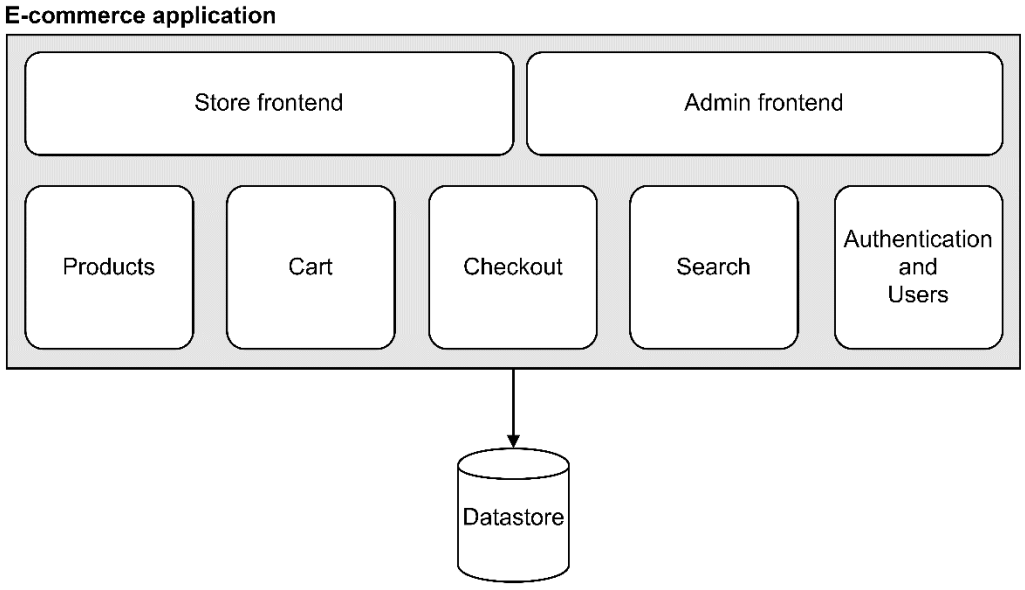

Request
Load balancer
John
Instance A
Instance B
authenticated:['john']

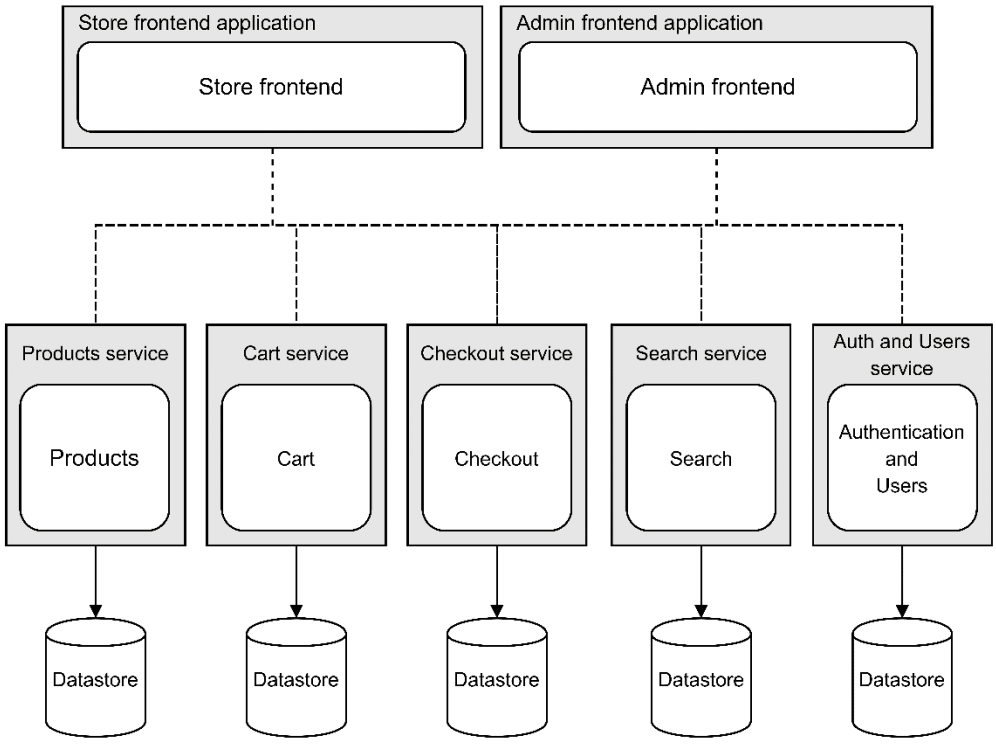

Associate the new session with an instance
Session
Server
XYZ123
Instance A
Authenticate (session XYZ123)
Load balancer
John
Instance A
authenticated:['john']
Instance B
authenticated:[]
Get the instance associated with the session
Session
Server
XYZ123
Instance A
Request (session XYZ123)
Load balancer
John
Instance A
authenticated:['john']
Instance B
authenticated:[]

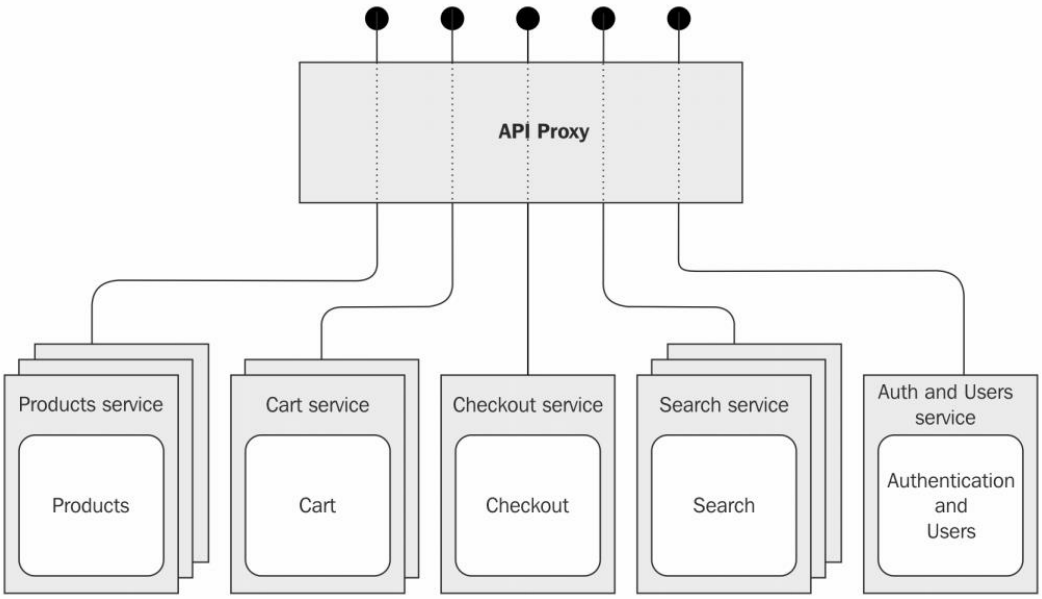

Load
balancer
Server #1
Application
instance
Application
instance
Server #2
Application
instance
Application
instance
Server #3
Application
instance
Application
instance
Server #4

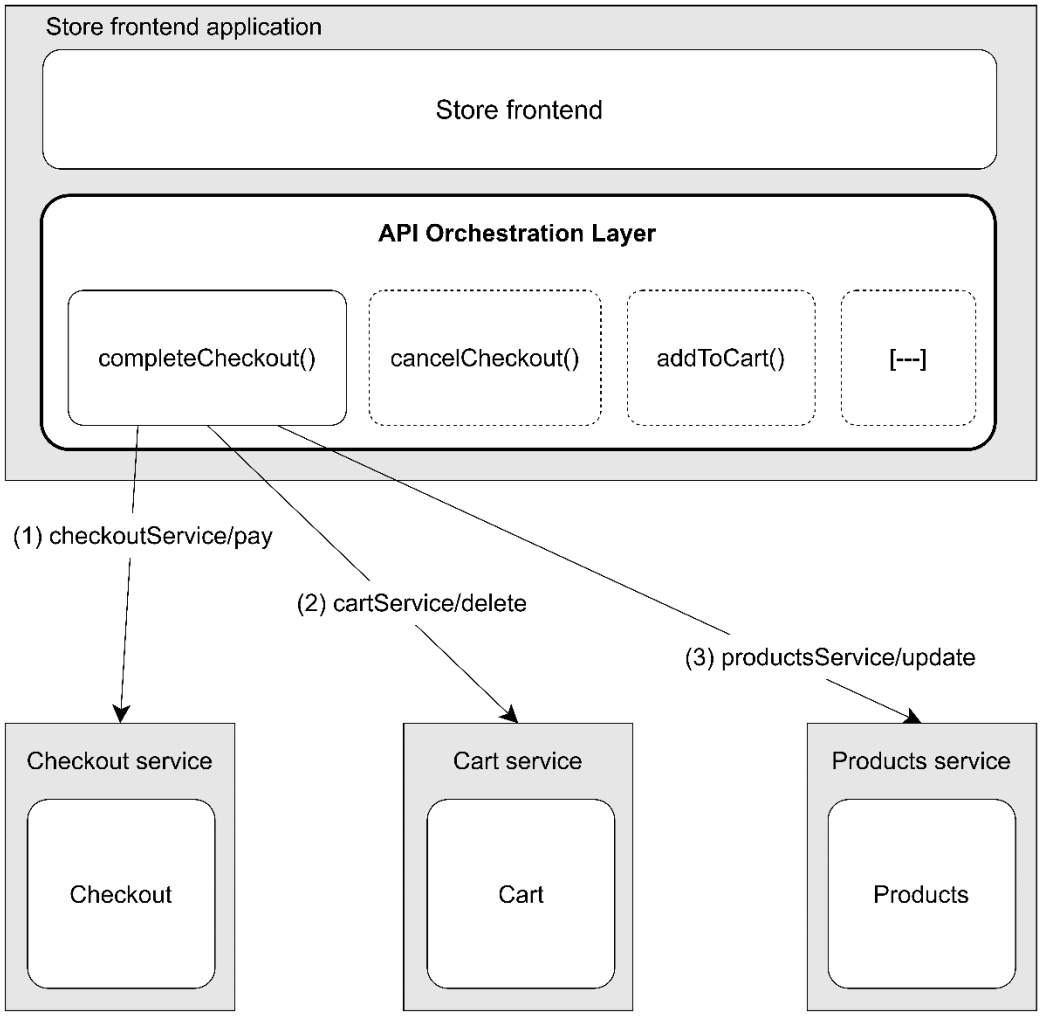

example.com
Service Registry
API
api1.example.com:8080
api1.example.com:8081
api2.example.com:8080
WebApp
web1.example.com:8080
Reads the topology
Load Balancer
/api
/*
API Server (8080)
API Server (8081)
API Server (8080)
WebApp Server (8080)
api1.example.com
api2.example.com
web1.example.com
Register their services

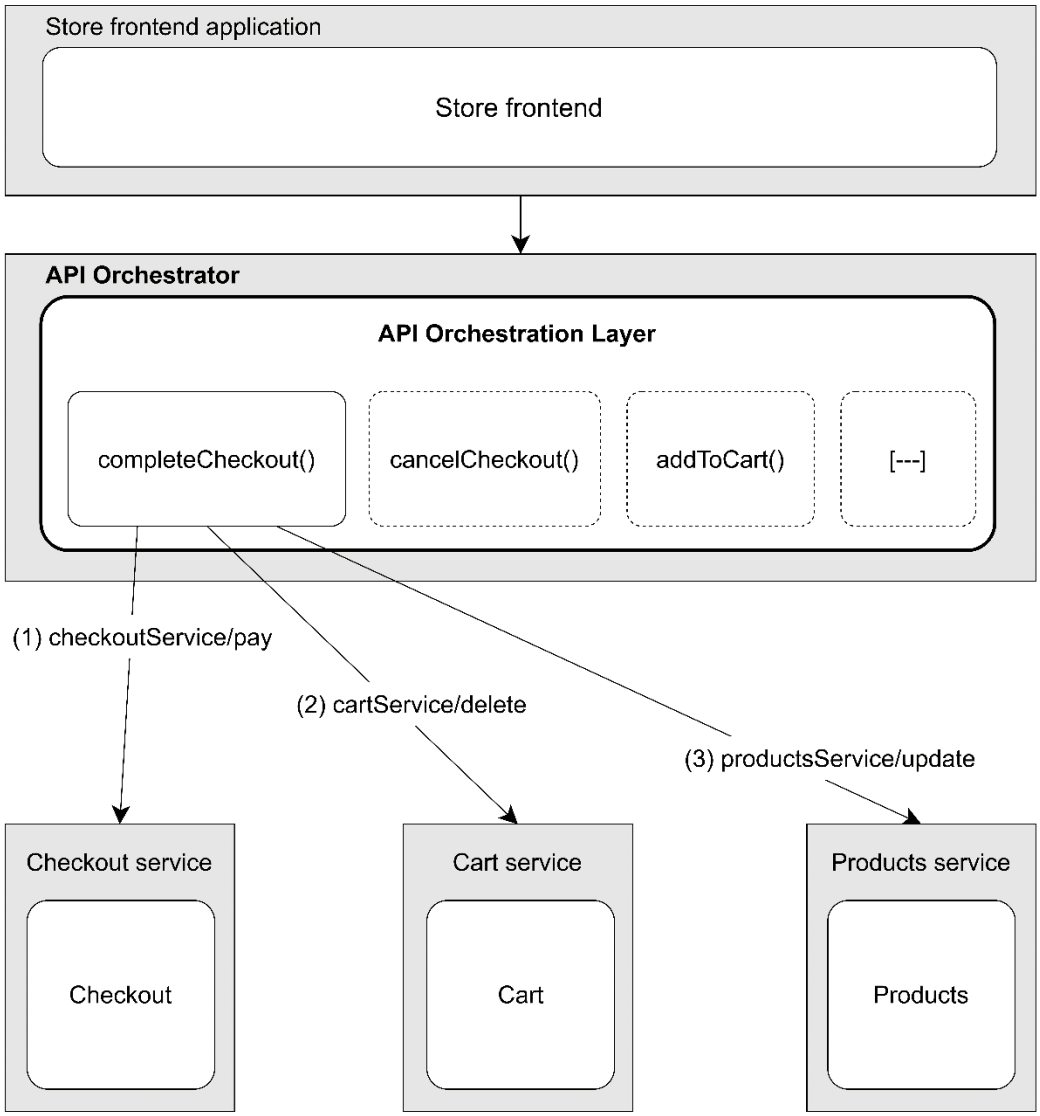

Centralized load balancing
Service A
(Client)
Load
Balancer
Service B
Service B
Service B
Peer-to-peer load balancing
Service A
(Client)
Service B
Service B
Service B

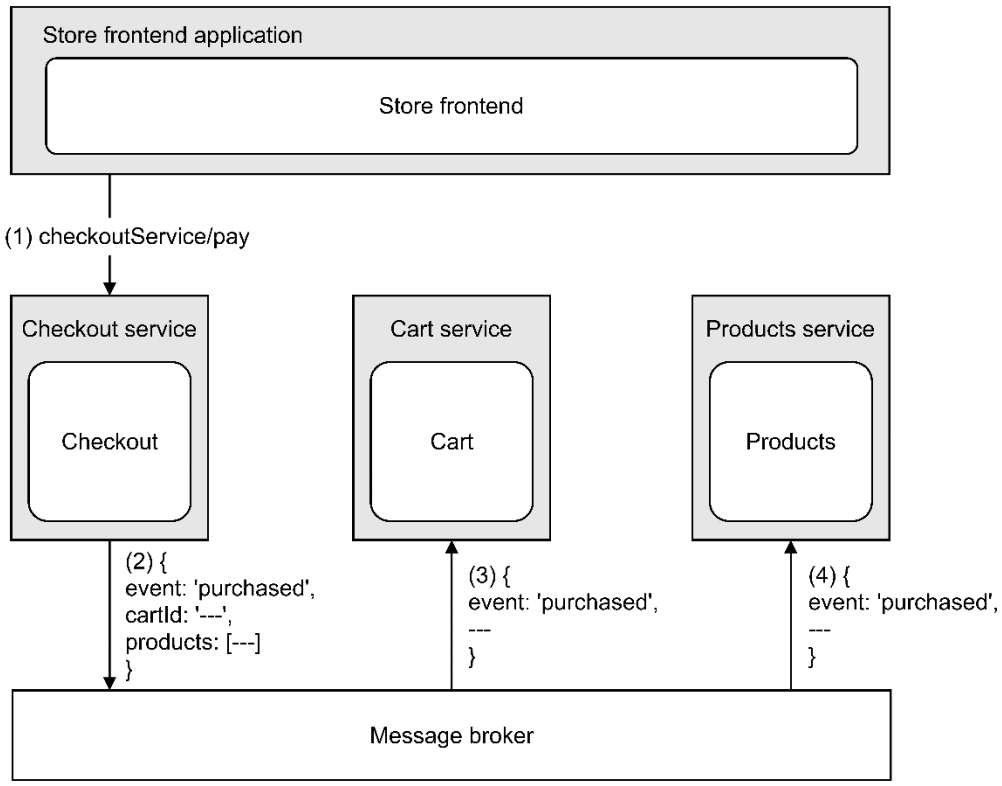

E-commerce application
Store frontend
Admin frontend
Products
Cart
Checkout
Search
Authentication and Users
Datastore


Store frontend application
Store frontend
Admin frontend application
Admin frontend
Products service
Products
Cart service
Cart
Checkout service
Checkout
Search service
Search
Auth and Users service
Authentication and Users
Datastore
Datastore
Datastore
Datastore
Datastore


API Proxy
Products service
Products
Cart service
Cart
Checkout service
Checkout
Search service
Search
Auth and Users service
Authentication and Users


Store frontend application
Store frontend
API Orchestration Layer
completeCheckout()
cancelCheckout()
addToCart()
[---]
(1) checkoutService/pay
(2) cartService/delete
(3) productsService/update
Checkout service
Checkout
Cart service
Cart
Products service
Products

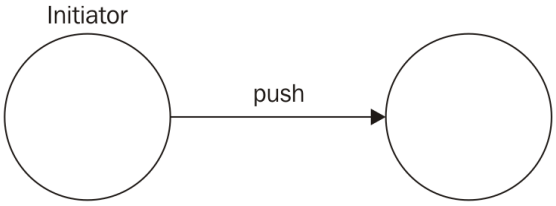

Store frontend application
Store frontend
API Orchestrator
API Orchestration Layer
completeCheckout()
cancelCheckout()
addToCart()
[---]
(1) checkoutService/pay
(2) cartService/delete
(3) productsService/update
Checkout service
Checkout
Cart service
Cart
Products service
Products

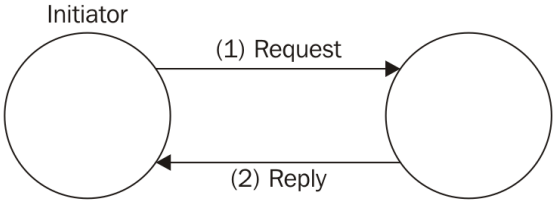

Store frontend application
Store frontend
(1) checkoutService/pay
Checkout service
Checkout
Cart service
Cart
Products service
Products
(2) {
event: 'purchased',
cartId: '---',
products: [---]
}
(3) {
event: 'purchased',

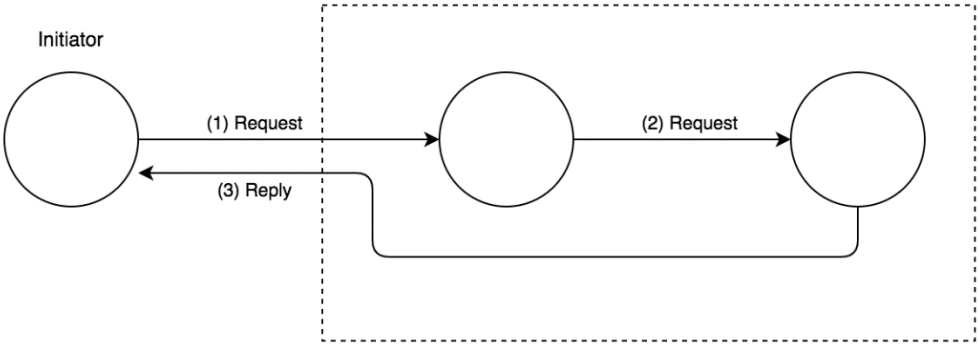
}
(4) {
event: 'purchased',

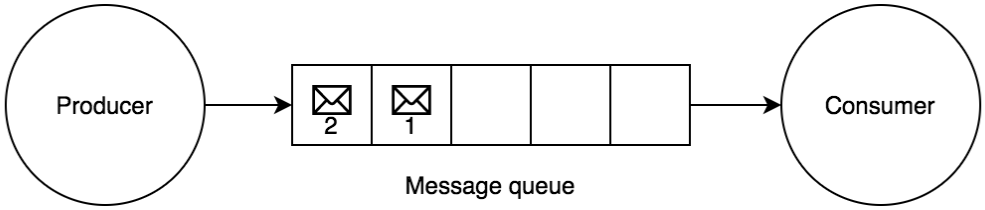
}
Message broker

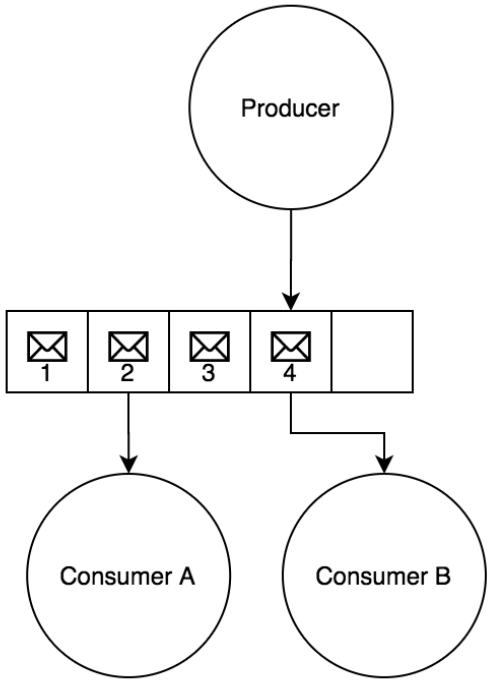
Chapter 13: Messaging and Integration Patterns

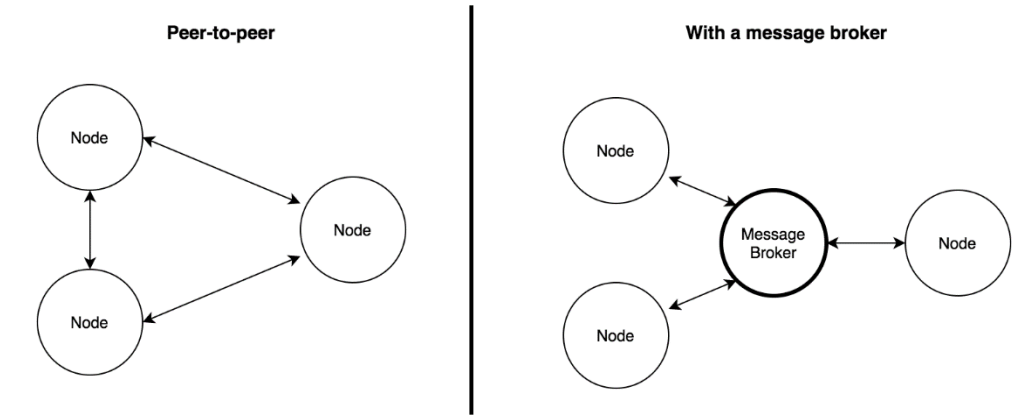


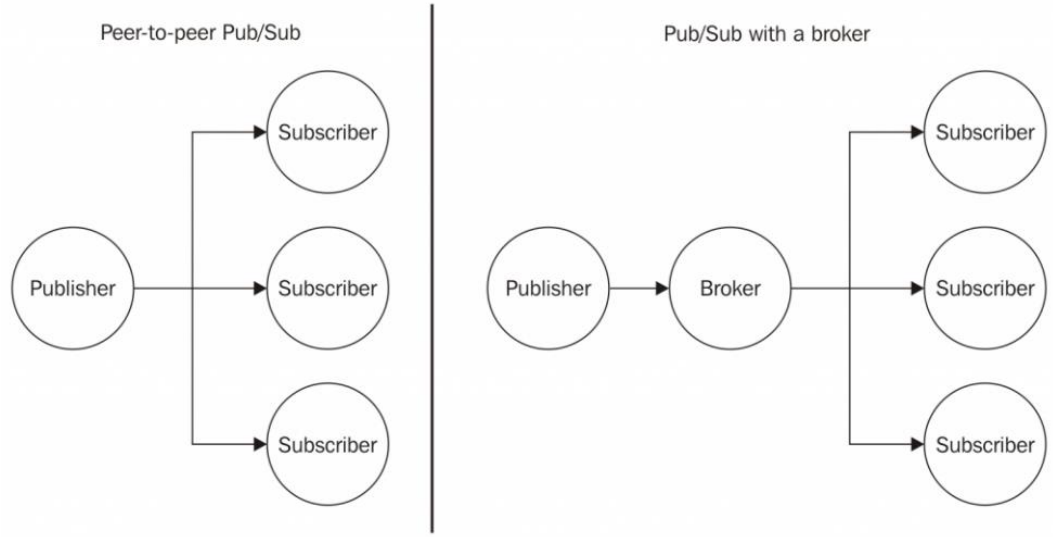


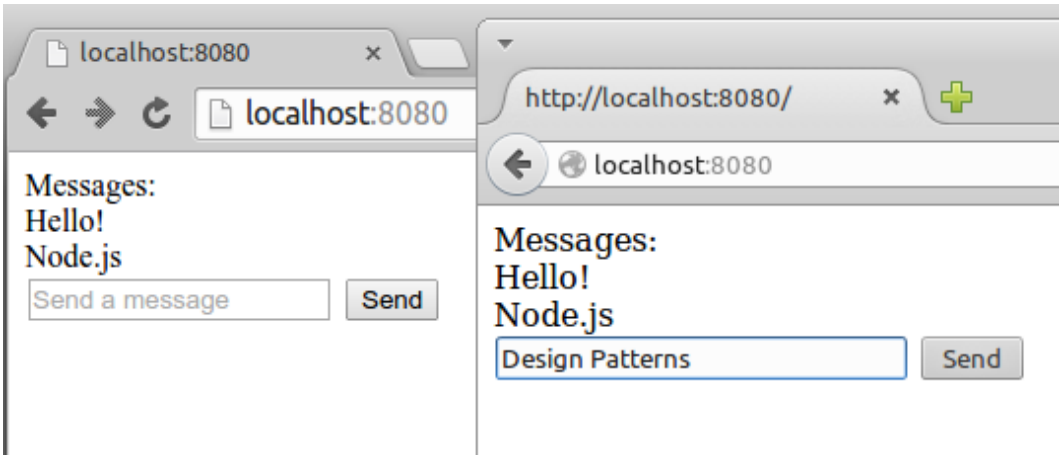


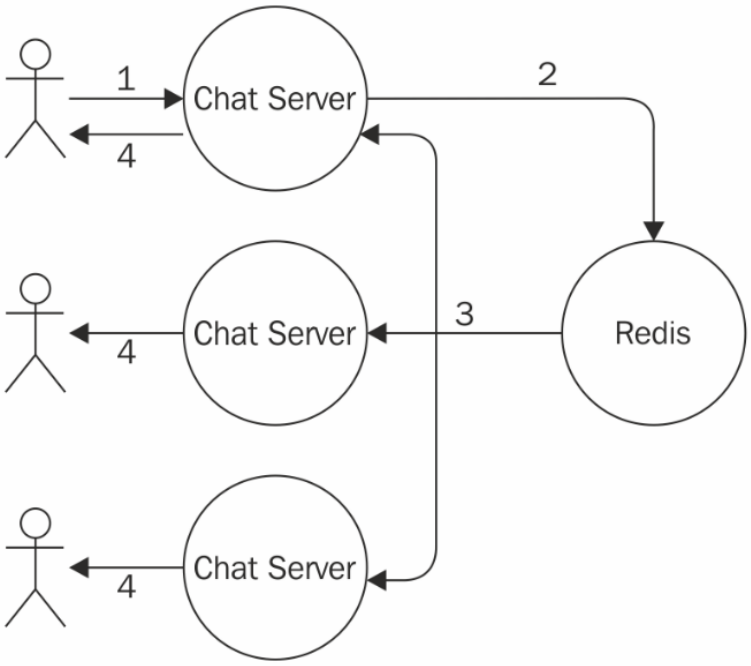

Producer
2
1
Consumer
Message queue

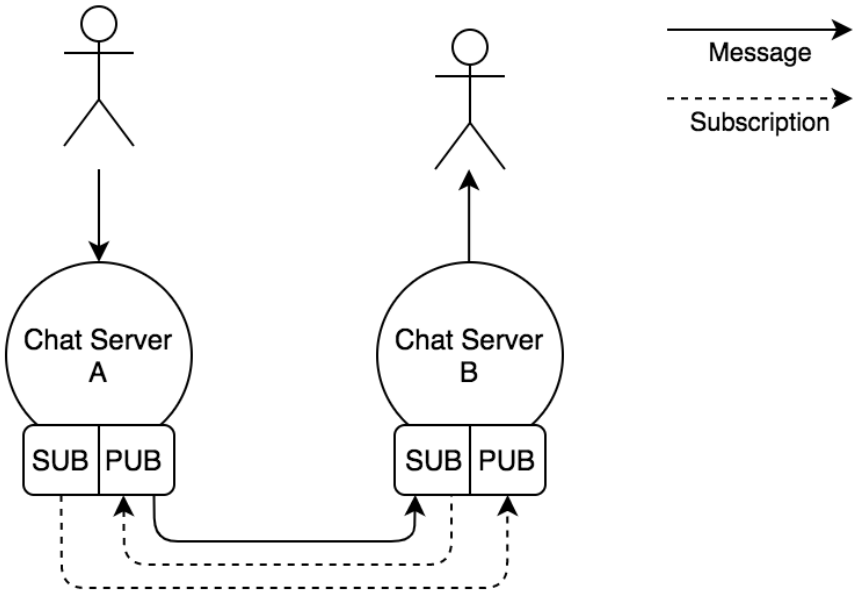

Producer
1
2
3
4
Consumer A
Consumer B

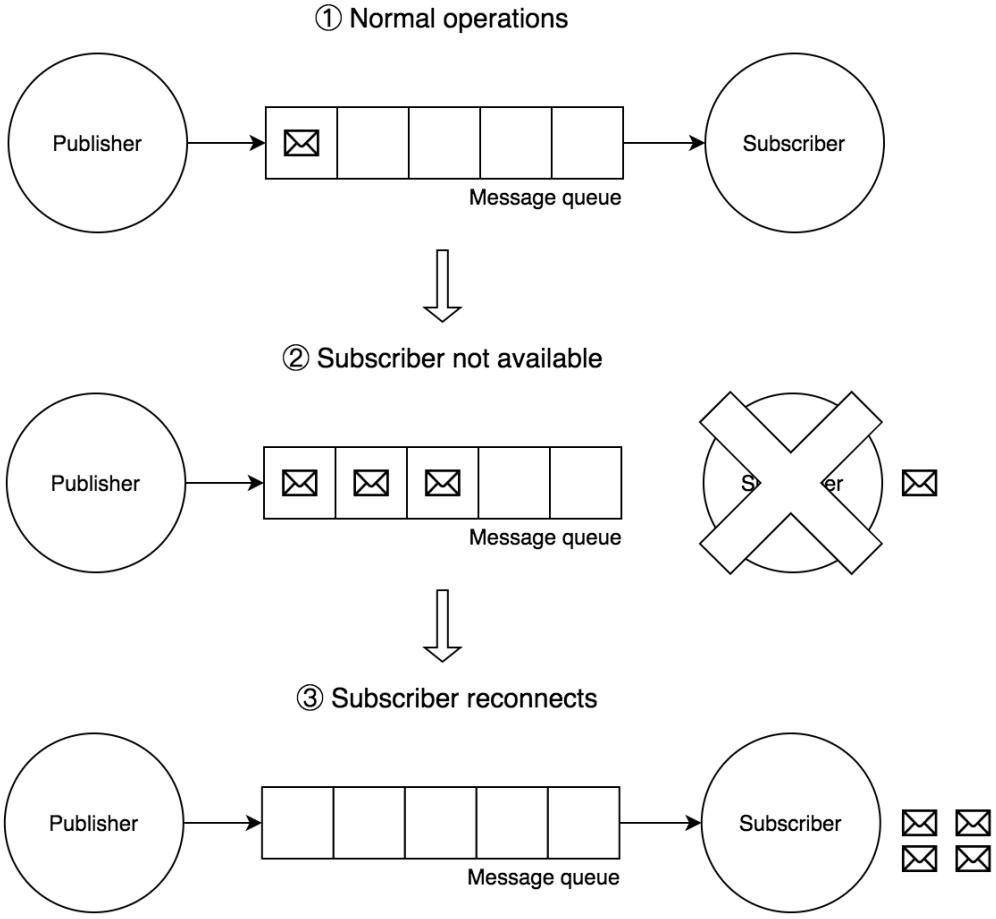

Peer-to-peer
Node
Node
Node
With a message broker
Node
Message Broker
Node
Node

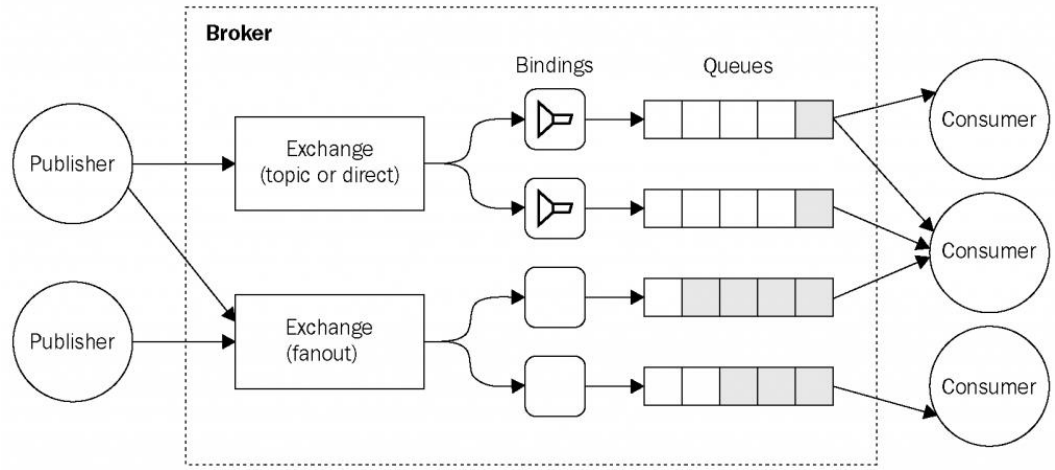

Peer-to-peer Pub/Sub
Publisher
Subscriber
Subscriber
Subscriber
Pub/Sub with a broker
Publisher
Broker
Subscriber
Subscriber
Subscriber

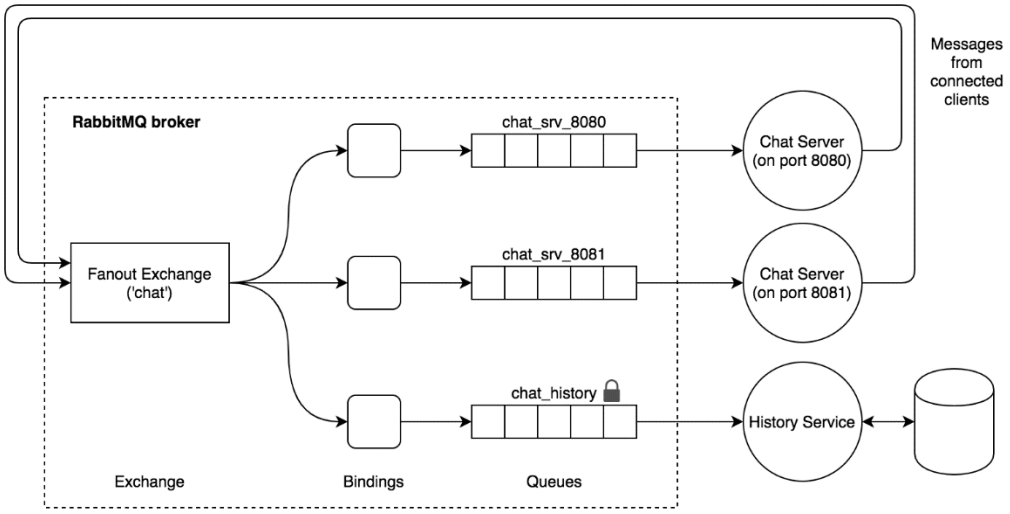

localhost:8080
localhost:8080
Messages:
Hello!
Node.js
Send a message
Send
http://localhost:8080/
localhost:8080
Messages:
Hello!
Node.js
Design Patterns
Send

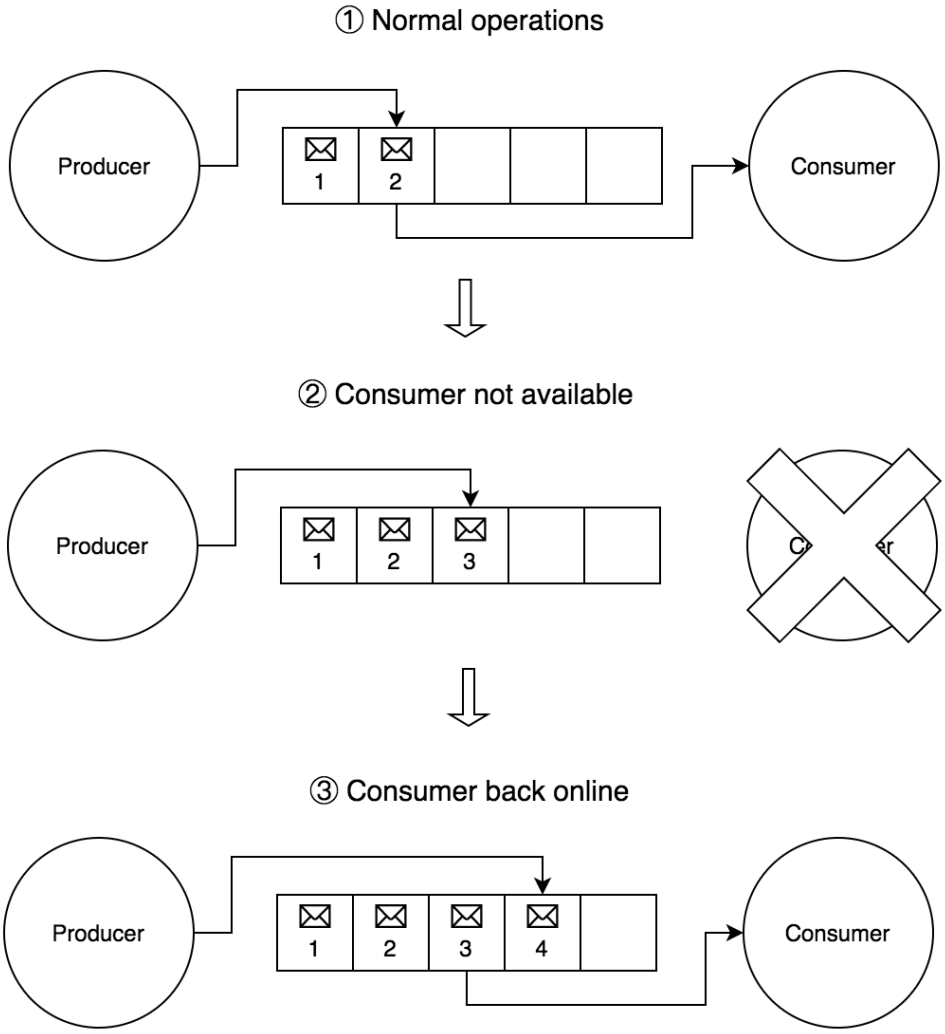

1
Chat Server
2
4
Chat Server
3
Redis
4
Chat Server
4

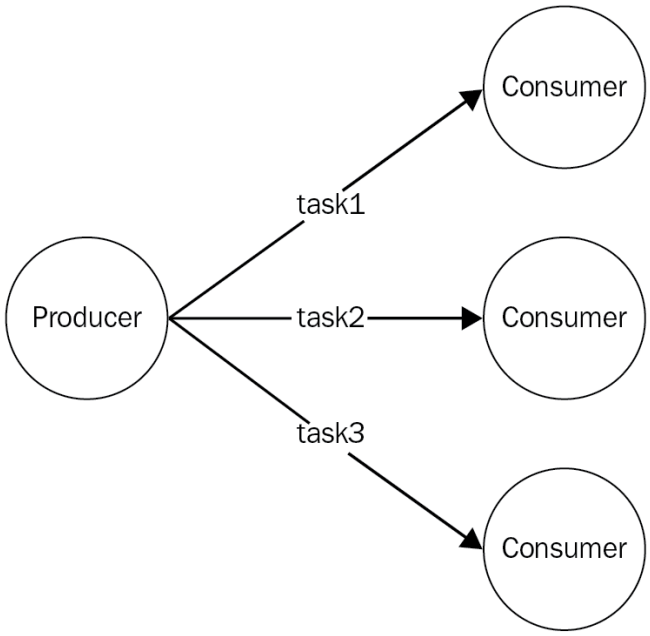

Message
Subscription
Chat Server A
SUB
PUB
Chat Server B
SUB
PUB

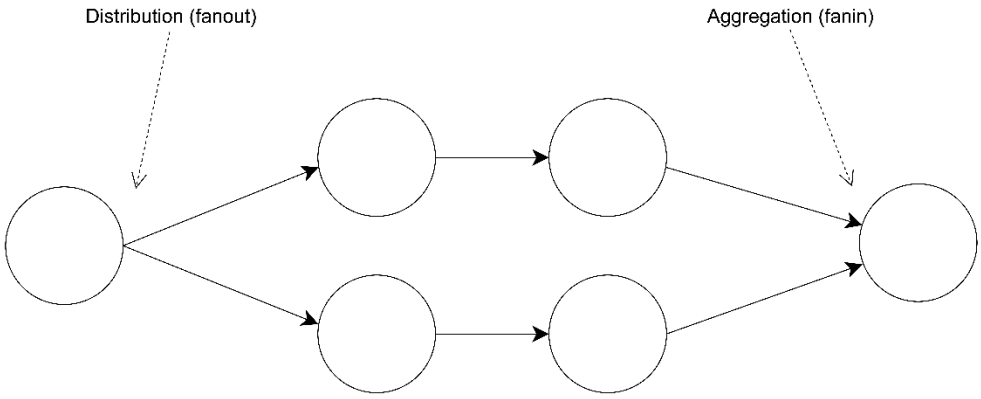

① Normal operations
Publisher
Subscriber
Message queue
② Subscriber not available
Publisher
Message queue
③ Subscriber reconnects
Publisher
Subscriber
Message queue

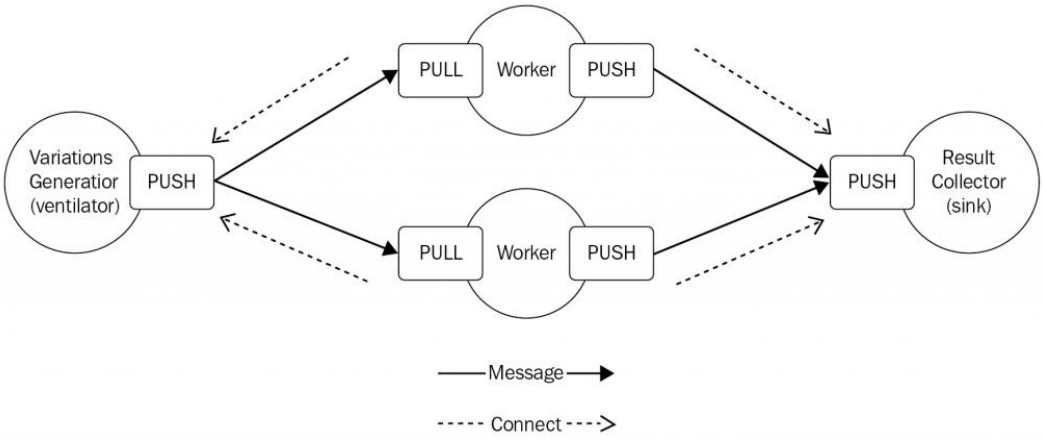

Broker
Bindings
Queues
Publisher
Exchange
(topic or direct)
Publisher
Exchange
(fanout)
Consumer
Consumer
Consumer

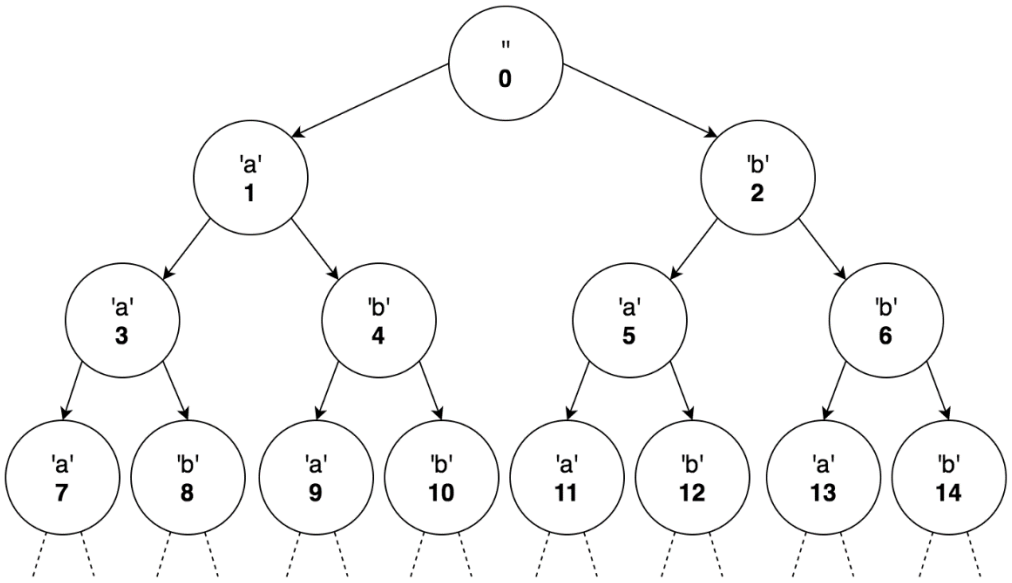

Messages
from
connected
clients
RabbitMQ broker
chat_srv_8080
Chat Server
(on port 8080)
Fanout Exchange
('chat')
chat_srv_8081
Chat Server
(on port 8081)
chat_history
History Service
Exchange
Bindings
Queues

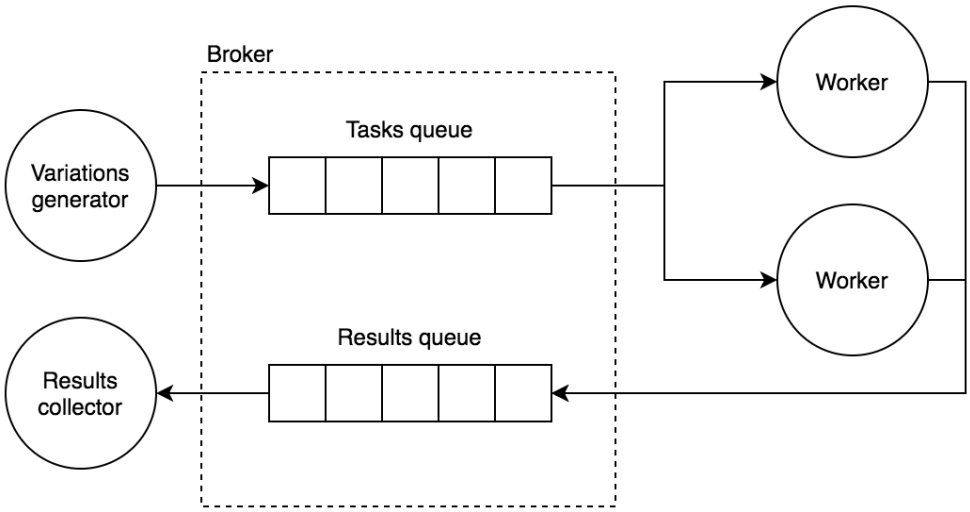

① Normal operations
Producer
1
2
Consumer
② Consumer not available
Producer
1
2
3
③ Consumer back online
Producer
1
2
3
4
Consumer

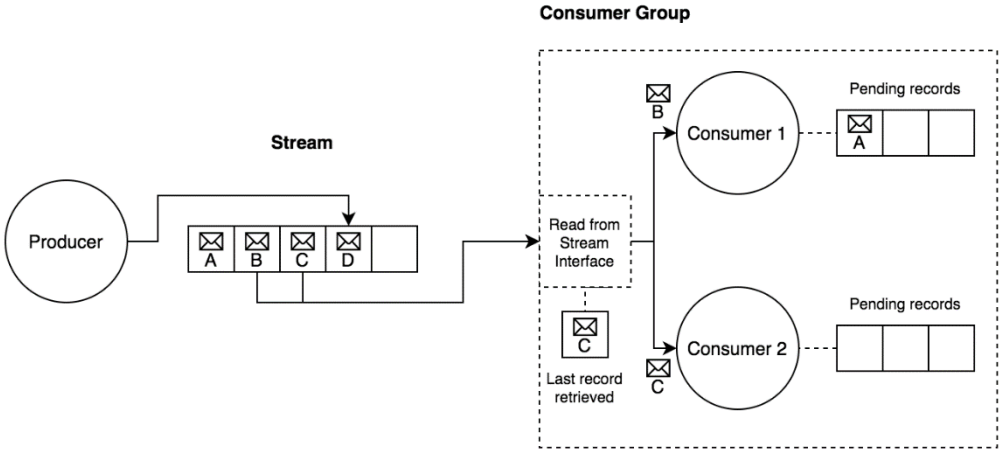

Consumer
task1
Producer
task2
Consumer
task3
Consumer

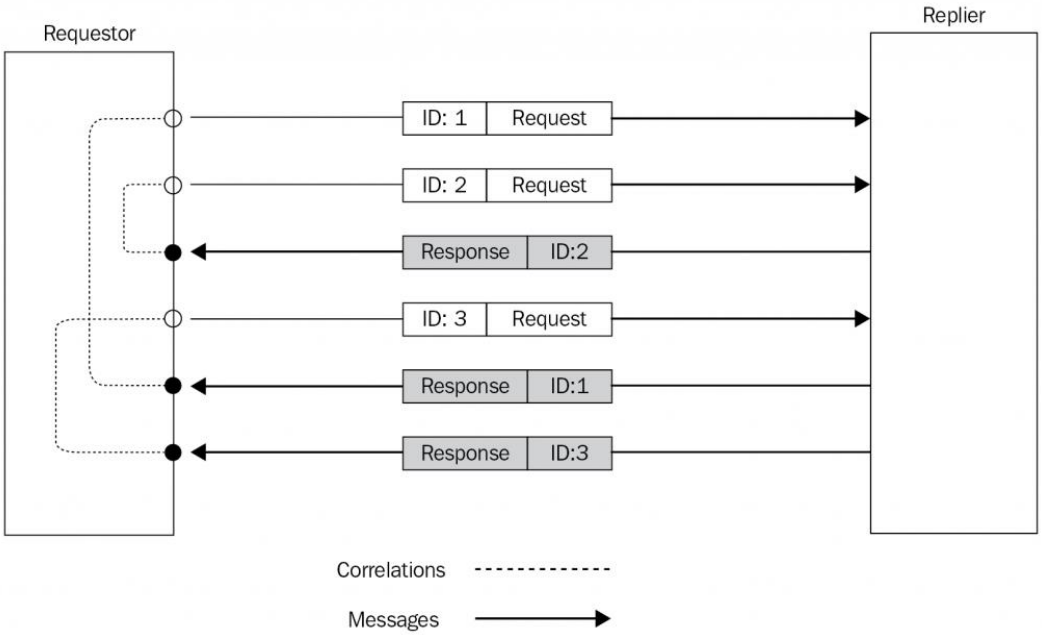

Distribution (fanout)
Aggregation (fanin)

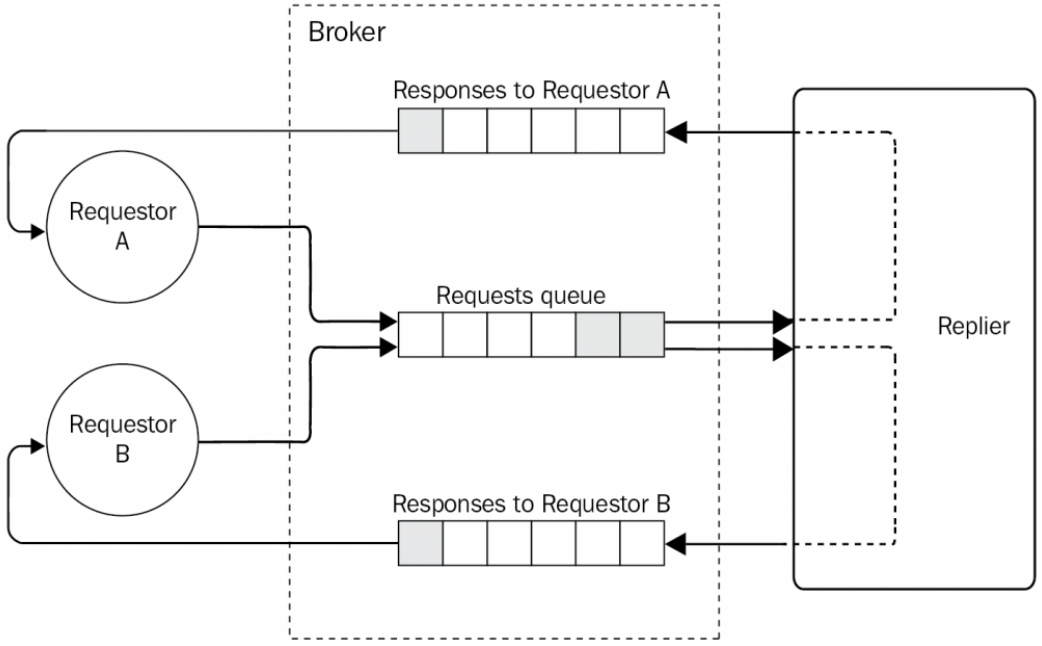

PULL
Worker
PUSH
Variations
Generatior
(ventilator)
PUSH
PUSH
Result
Collector
(sink)
PULL
Worker
PUSH
Message
Connect


''
0
'a'
1
'b'
2
'a'
3
'b'
4
'a'
5
'b'
6
'a'
7
'b'
8
'a'
9
'b'
10
'a'
11
'b'
12
'a'
13
'b'
14


Broker
Variations generator
Tasks queue
Worker
Worker
Results queue
Results collector


Consumer Group
Stream
Producer
A
B
C
D
Read from Stream Interface
B
Consumer 1
Pending records
A
C
Last record retrieved
Consumer 2
Pending records
C


Requestor
Replier
ID: 1
Request
ID: 2
Request
Response
ID:2
ID: 3
Request
Response
ID:1
Response
ID:3
Correlations
Messages


Broker
Responses to Requestor A
Requestor
A
Requests queue
Replier
Requestor
B
Responses to Requestor B