Event Sourcing as the Foundation of Traceability

DDD Europe 2020

Thomas Bøgh Fangel @tbfangel

Me

Thomas Bøgh Fangel

@tbfangel

Architect at Lunar since 2016

Distributed systems since 2004

Java, Scala, Typescript, Go

Event Sourcing for about a year

Agenda

- Context
  - Lunar and a bit of history
- Building a bank from scratch
  - Tech vision
- Event Sourcing
  - Why & How
  - Patterns
  - Challenges & Learnings

Lunar at a glance

- Founded as Lunar Way 2015
- Smartphone only challenger bank
- Originally built on top of existing bank
- Live 2016
- Best in class UX and support
- Present in DK, NO and SE

Facts

100+
Employees

150k
Users

~100
μ services

25%
Engineers

1M+
Tx pr month

3
K8S clusters

Native iOS and Android app's

Backend

RAILS

Bank Integration

DK Bank

NemID Integration

NemID

PostgreSQL

amazon web services

docker

LUNAR

Core

Auth

User

Product

RAILS

Signup

Feature

Transfer

Adapter

LUNAR

Features

Lunar Feature A

Lunar Feature B

Lunar Feature C

Adapter

DK Clearing

DK Identity

SE/NO Identity

Partner A

Partner B

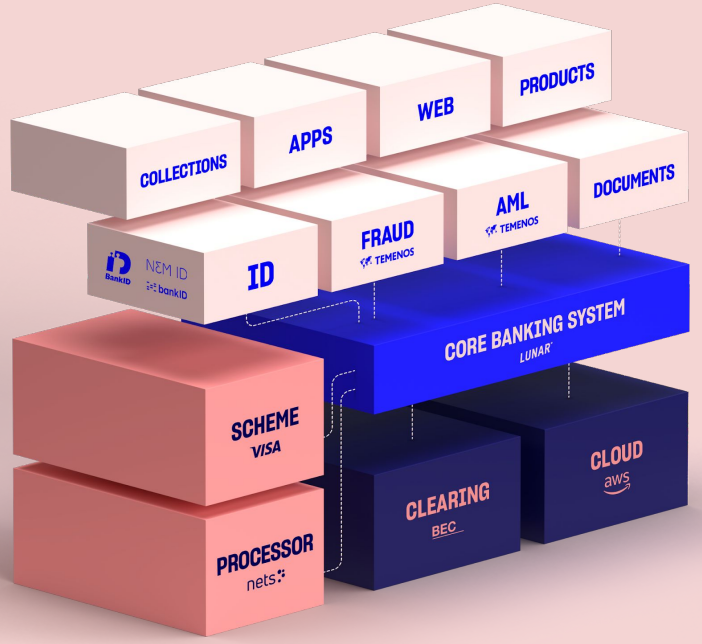
The good

- Autonomy
- Speed
- Maintainability
- Messaging

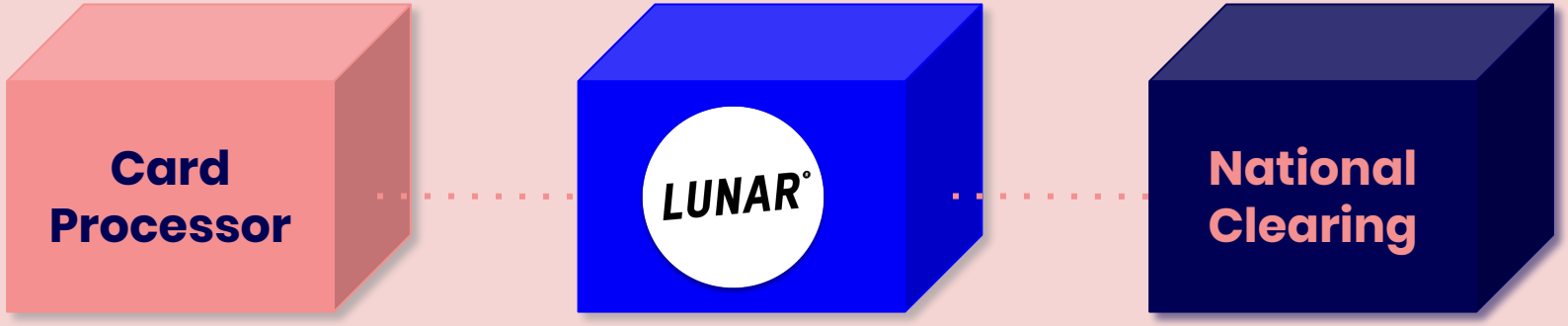
The bad

- Messaging
- Consistency
- Traceability

Building a bank from scratch



What does it mean to be a bank?



Pay with card

Keep money safe

Transfer money

Tech Vision

Trustworthy

Secure

Correct

Tech Vision

Traceability at all levels

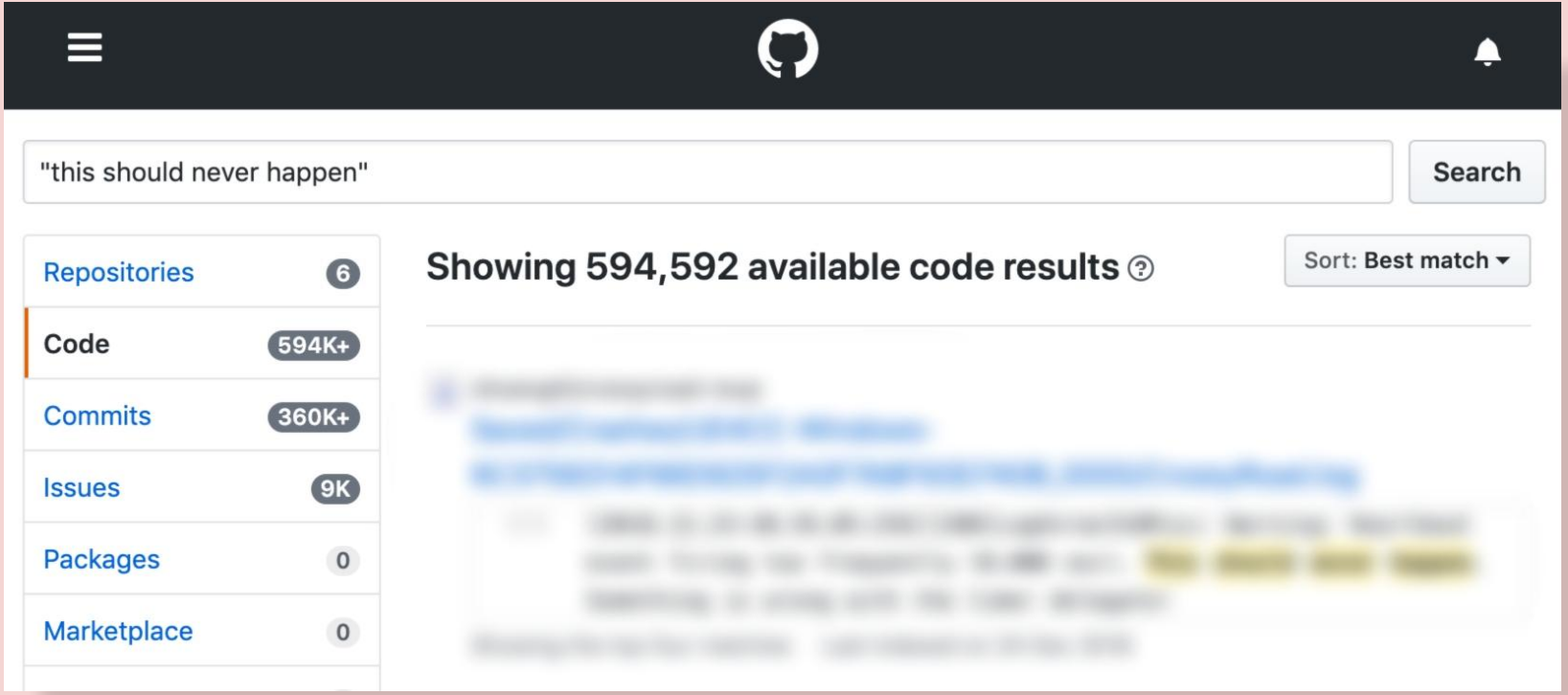
nothing should happen without us knowing
and our system should never be in a state
we cannot explain

Distributed Tracing

≠

Traceability

```
func DoSomeBusiness(s *State) (Result, error)
{
   if s.InImpossibleState() {
      //this should never happen - what to do?
      panic(“current state is impossible”)
   }
   //happy cases below
}
```


"this should never happen"
Search
Repositories 6
Code 594K+
Commits 360K+
Issues 9K
Packages 0
Marketplace 0
Showing 594,592 available code results
Sort: Best match

Production

=

The place where the impossible happens

Traceability

=

Explain the impossible

“Something really awful happened to your money – we really don’t know what happened, but we’re trying to figure it out”

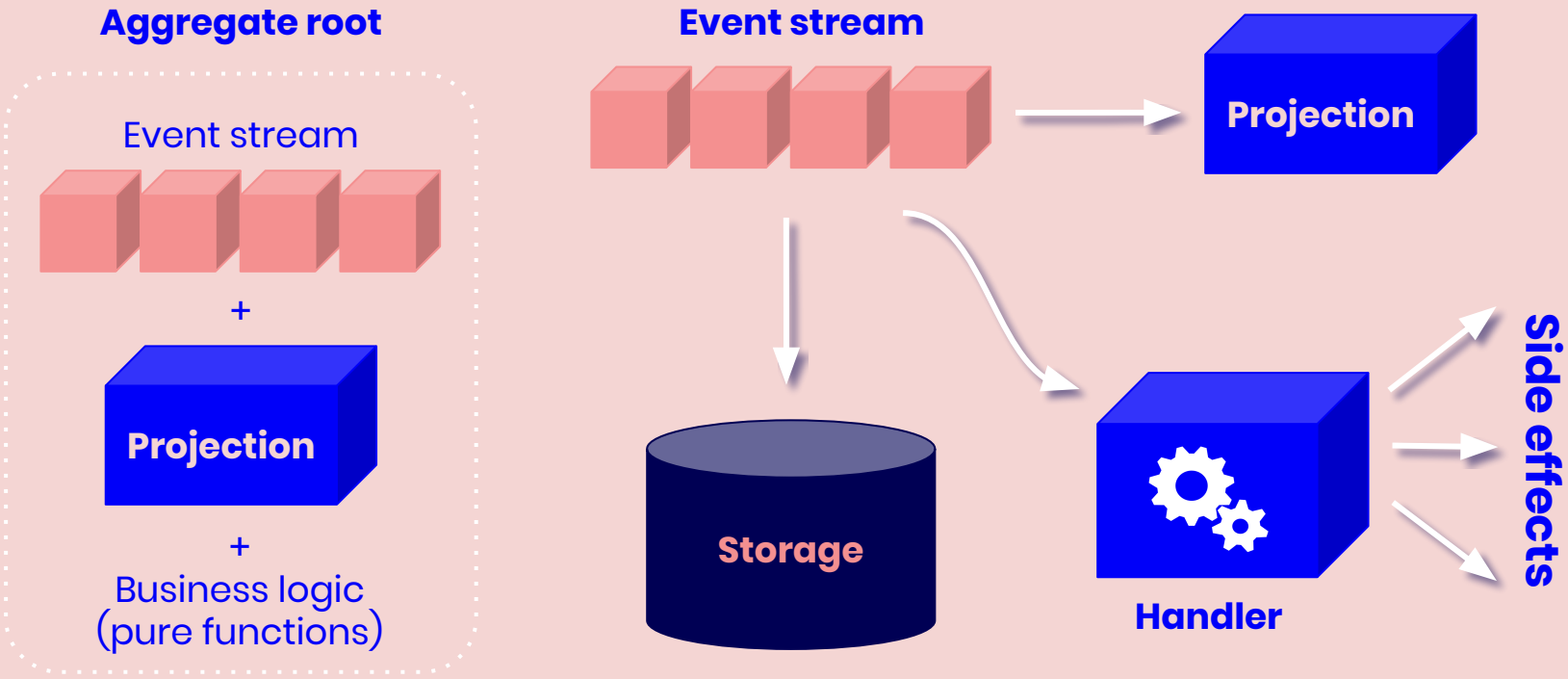
“Something really awful happened to your money - but we know exactly where the money is and we will fix it”

Event Sourcing

Shift of Focus

From state...
...to events

Event Sourcing Components



Implementation

- Apogee: ES & CQRS* library in Go (Open Source in 2020)
- Postgres as event storage
- In-memory views for AR
- SQL backed views - later

* CQRS: Command Query Responsibility Segregation

```go
//TodoList defines the state of the todo list aggregate root
type TodoList struct {
  //CreatedAt contains the timestamp of when the list was created
  CreatedAt *time.Time
  //Items is a map of the items on the list to their state
  Items         map[types.ItemID]itemstate
}

//ItemChecked is the event published when an item is checked
type ItemChecked struct {
  HappenedAt time.Time
  ID                types.ItemID
}

func (t *TodoList) ApplyItemChecked(event ItemChecked) {
  s := t.Items[event.ID]
  s.Checked = true
  t.Items[event.ID] = s
}
```

```go
func (t *TodoList) HandleCheckItem(cmd CheckItem, uow aggregateroot.UnitOfWork) {
  if cmd.ID.IsEmpty() {
    uow.Fail(FailureCode_InvalidCommand, "empty item id")
  }
  state, ok := t.Items[cmd.ID]
  if !ok {
    uow.Fail(FailureCode_ItemNotFound, "item with id %s not found", cmd.ID)
  }
  if state.Checked {
    //nothing to do
    return
  }
  uow.Publish(ItemChecked{
    ID: cmd.ID,
    HappenedAt: cmd.Time,
  })
}
```

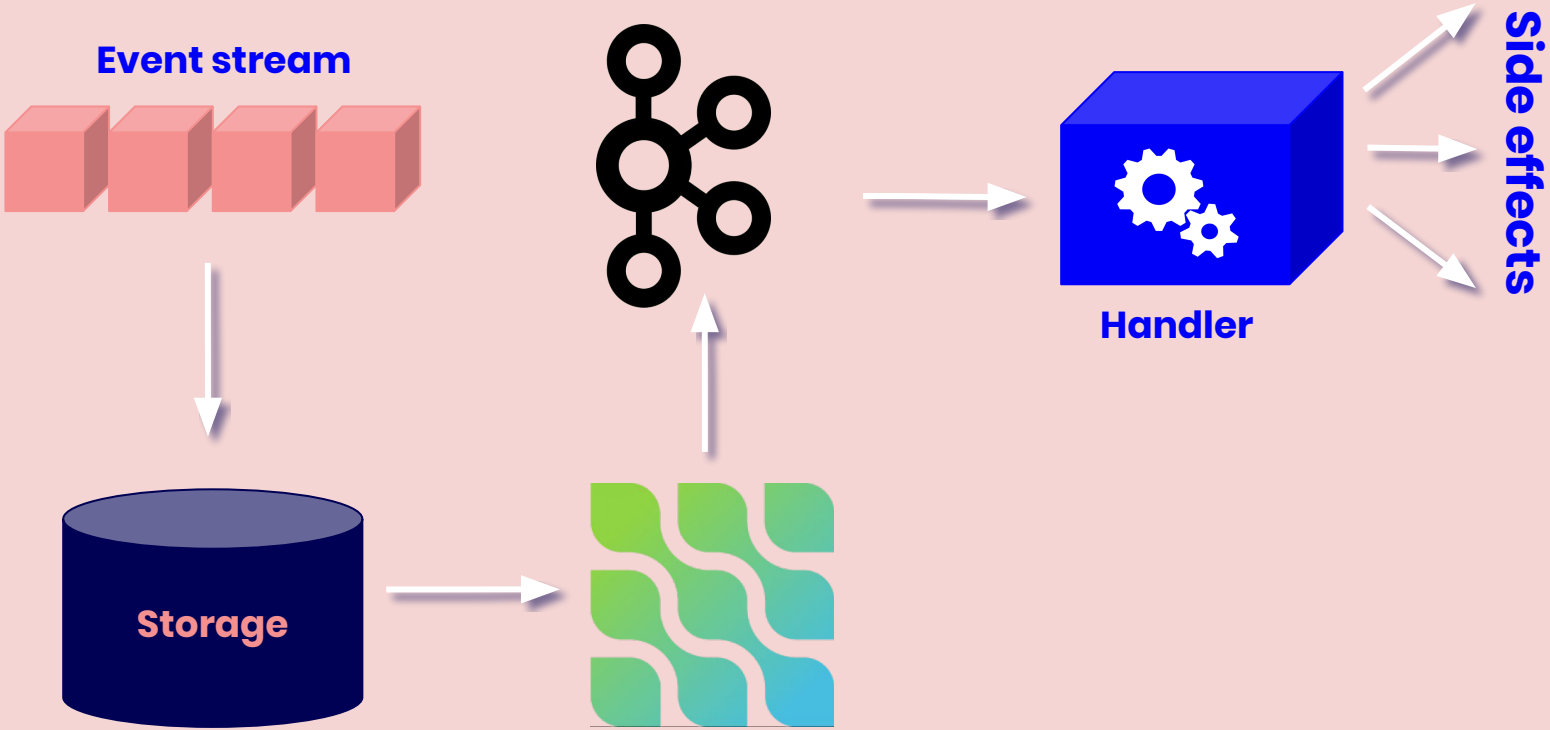
Single Crucial Property

Guaranteed Event Handling

Implementation

- Roll-your-own?
- Outbox pattern?
- Debezium with Kafka

Debezium & Kafka



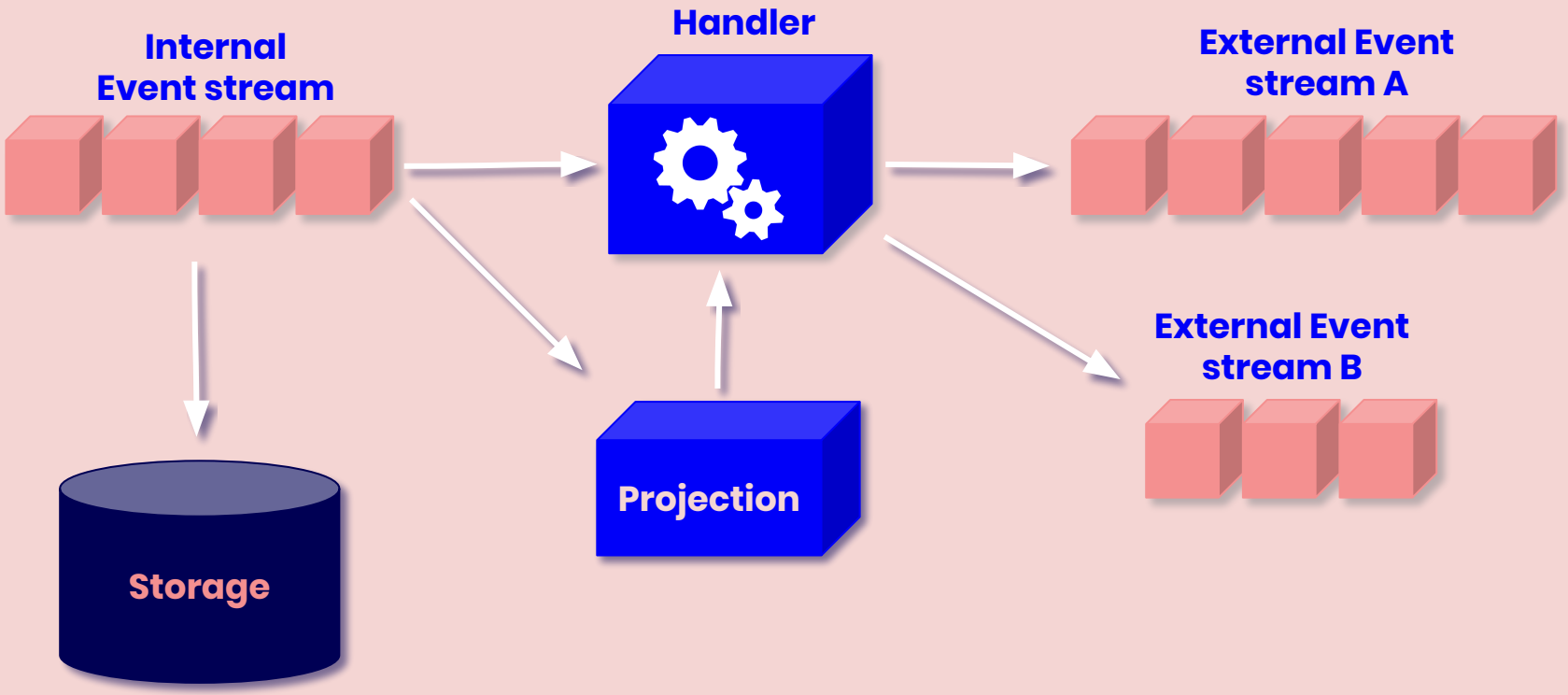
Patterns

Public APIs

External Event Streams

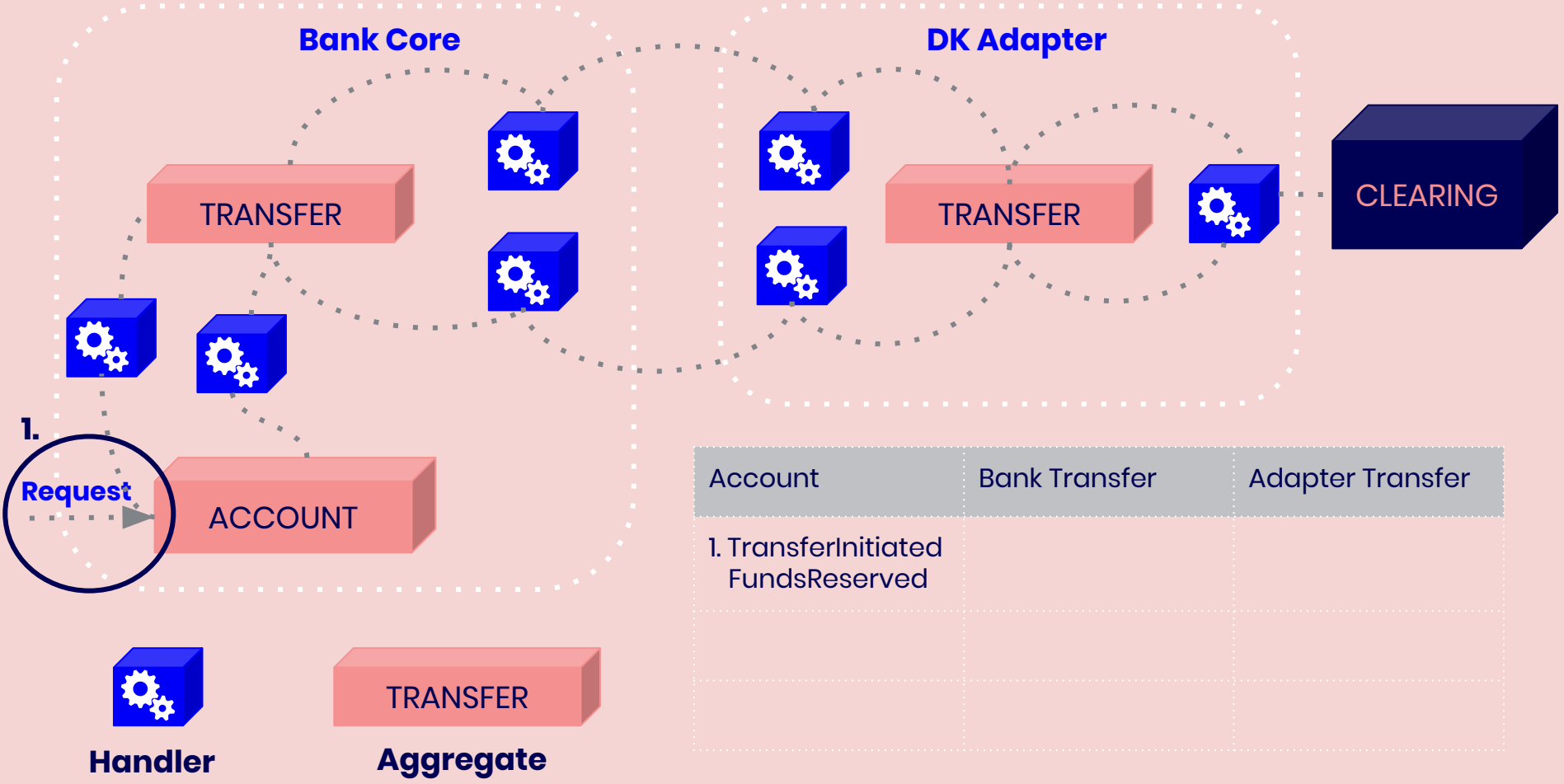
- Public API
- Same guarantees and properties as internal event stream
- Derived from internal event stream
- Essentially a handler writing to an event stream
- May keep track of source events

External Event Streams

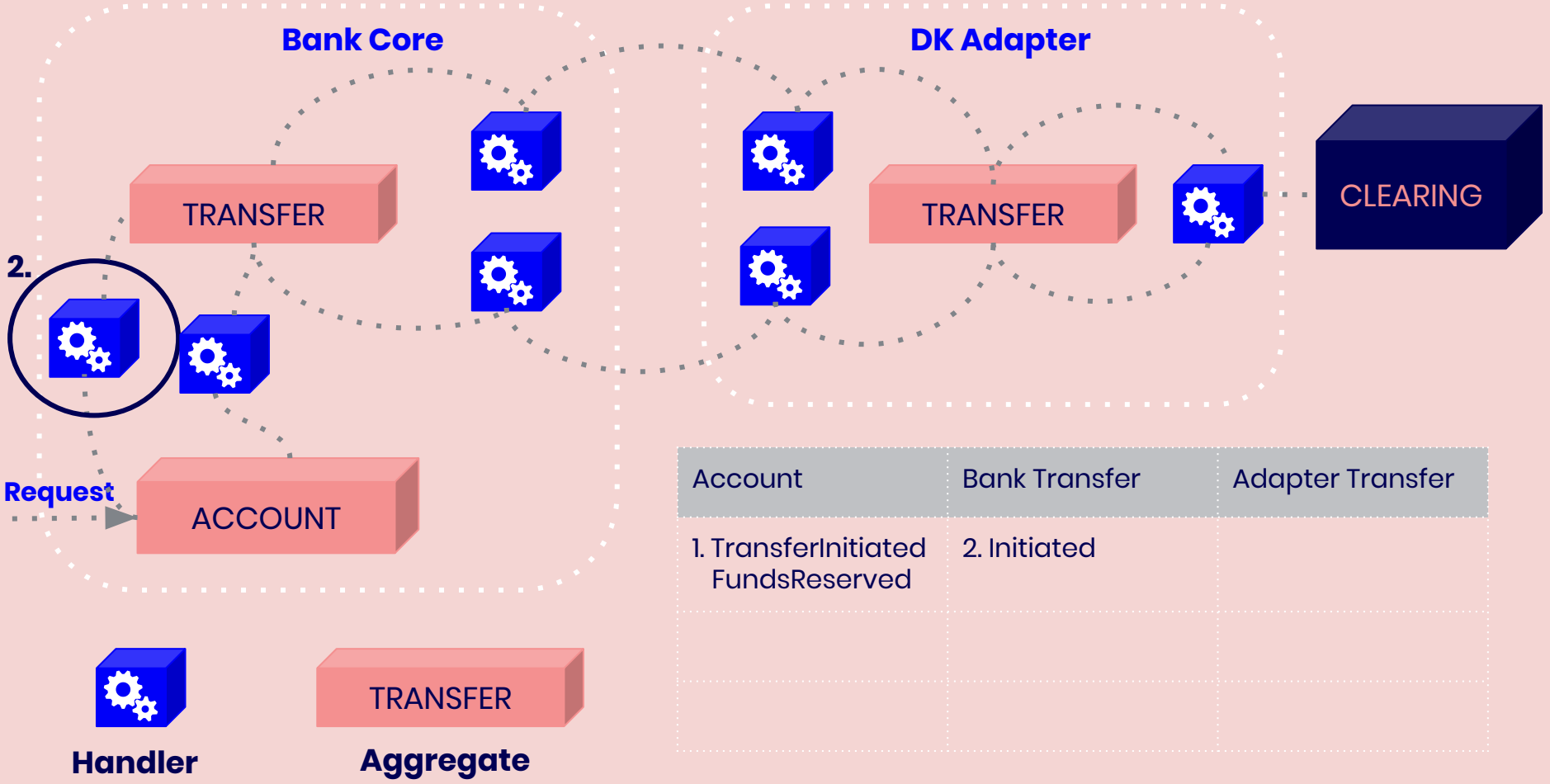


Distributed flows

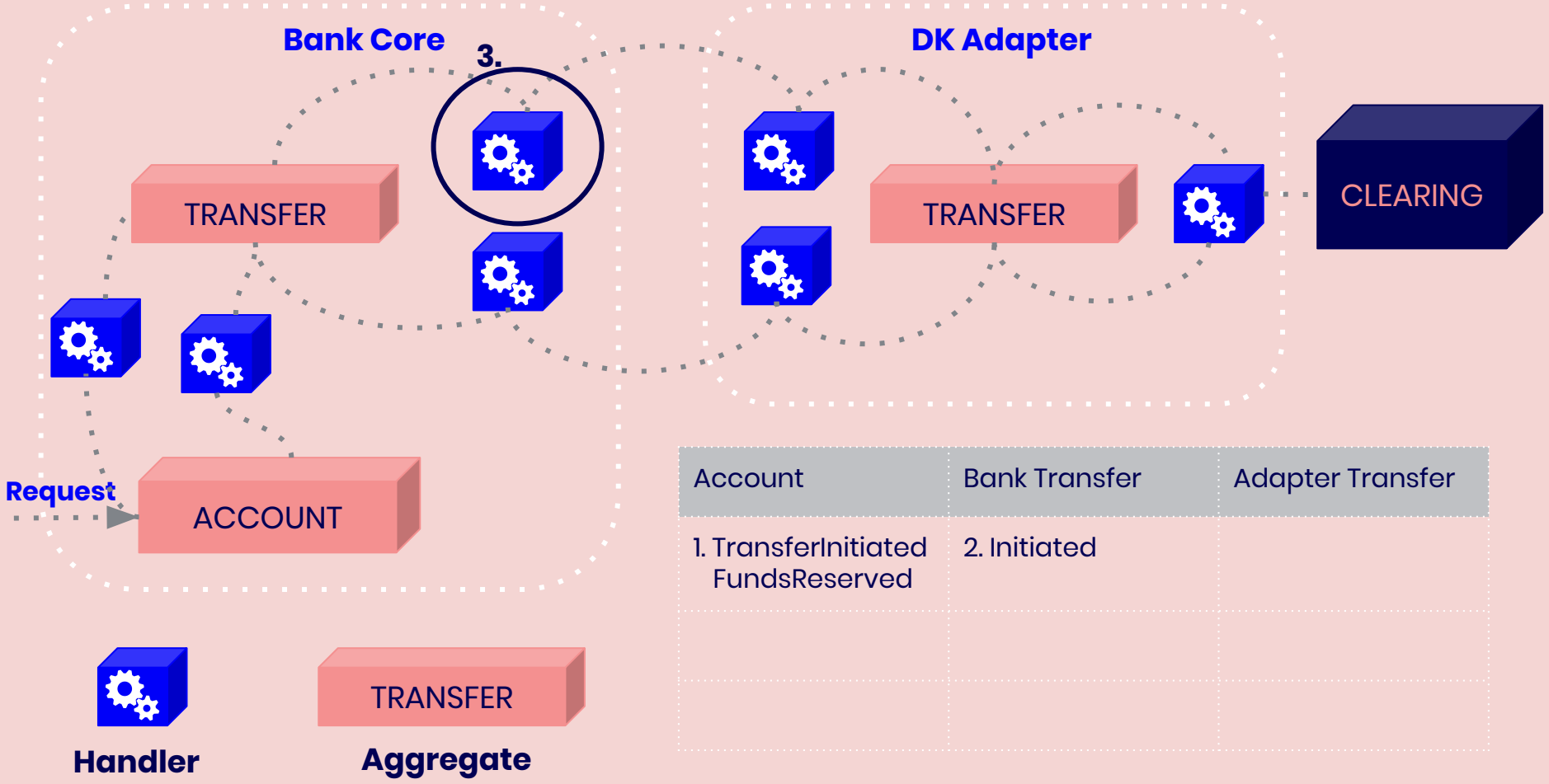
- Distributed transactions
- Represented as aggregate roots (state machines)
- State determines handler action
- Side effects in handlers
- Public event streams as API



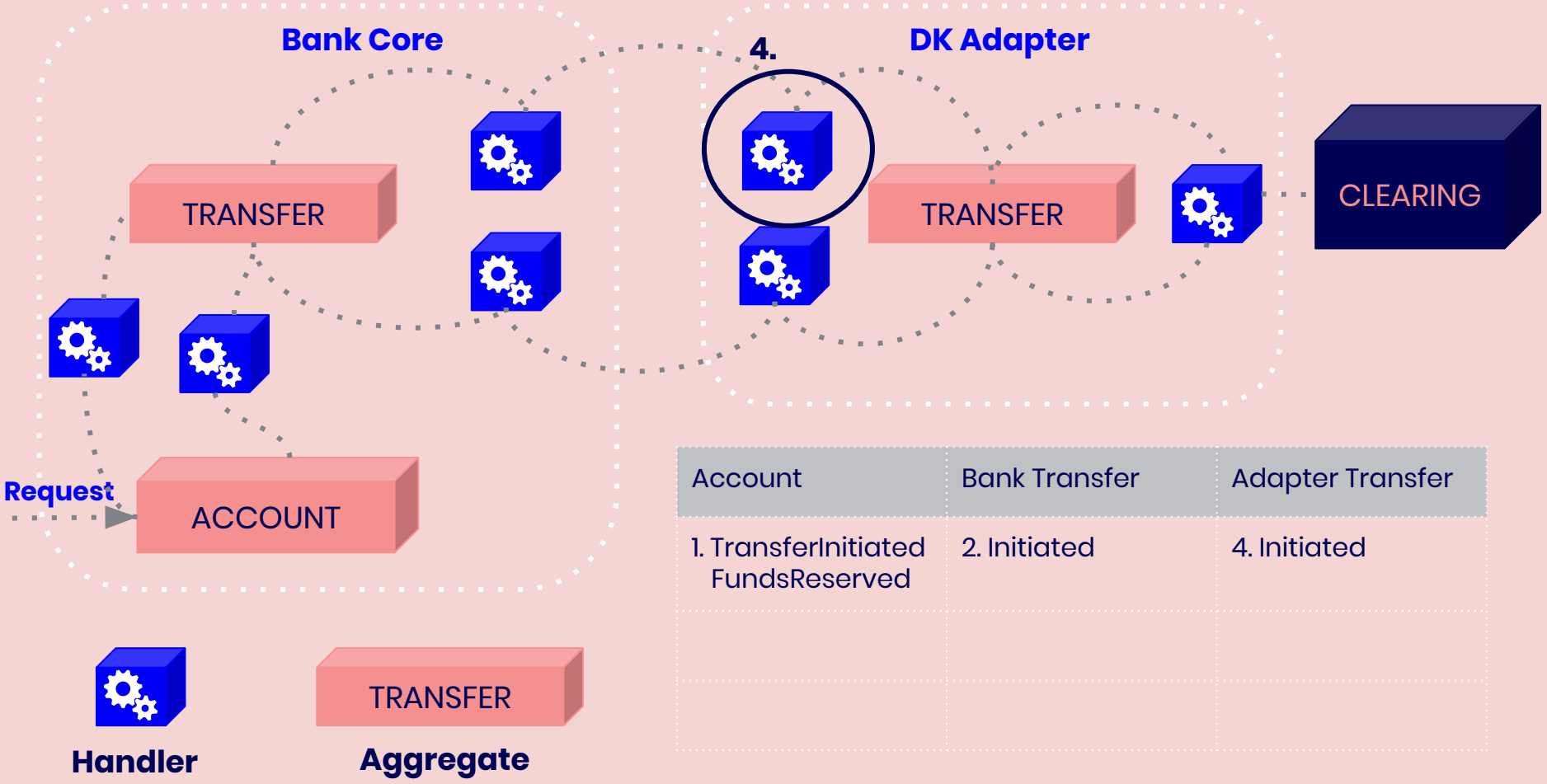
Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved		



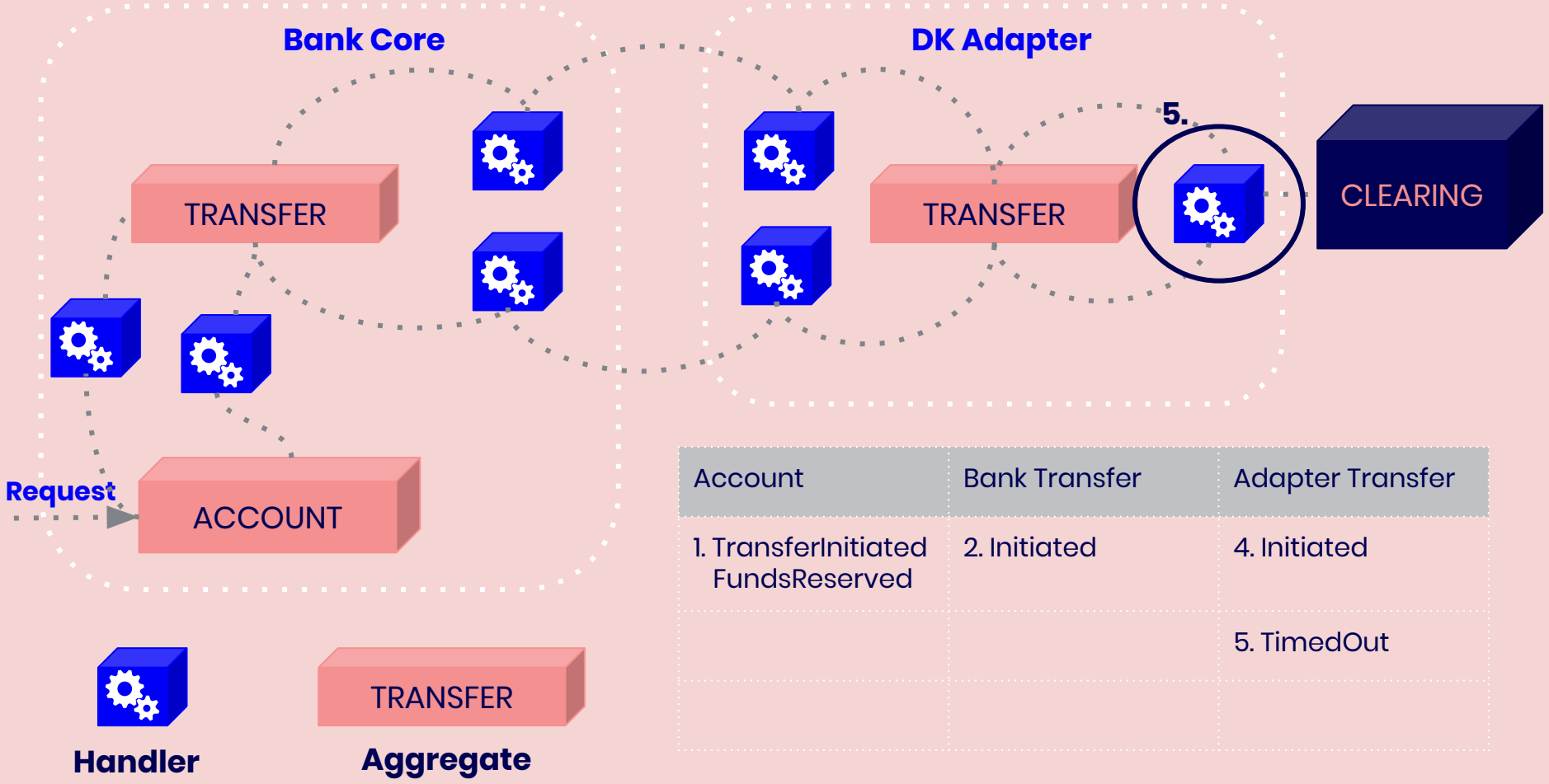
Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	



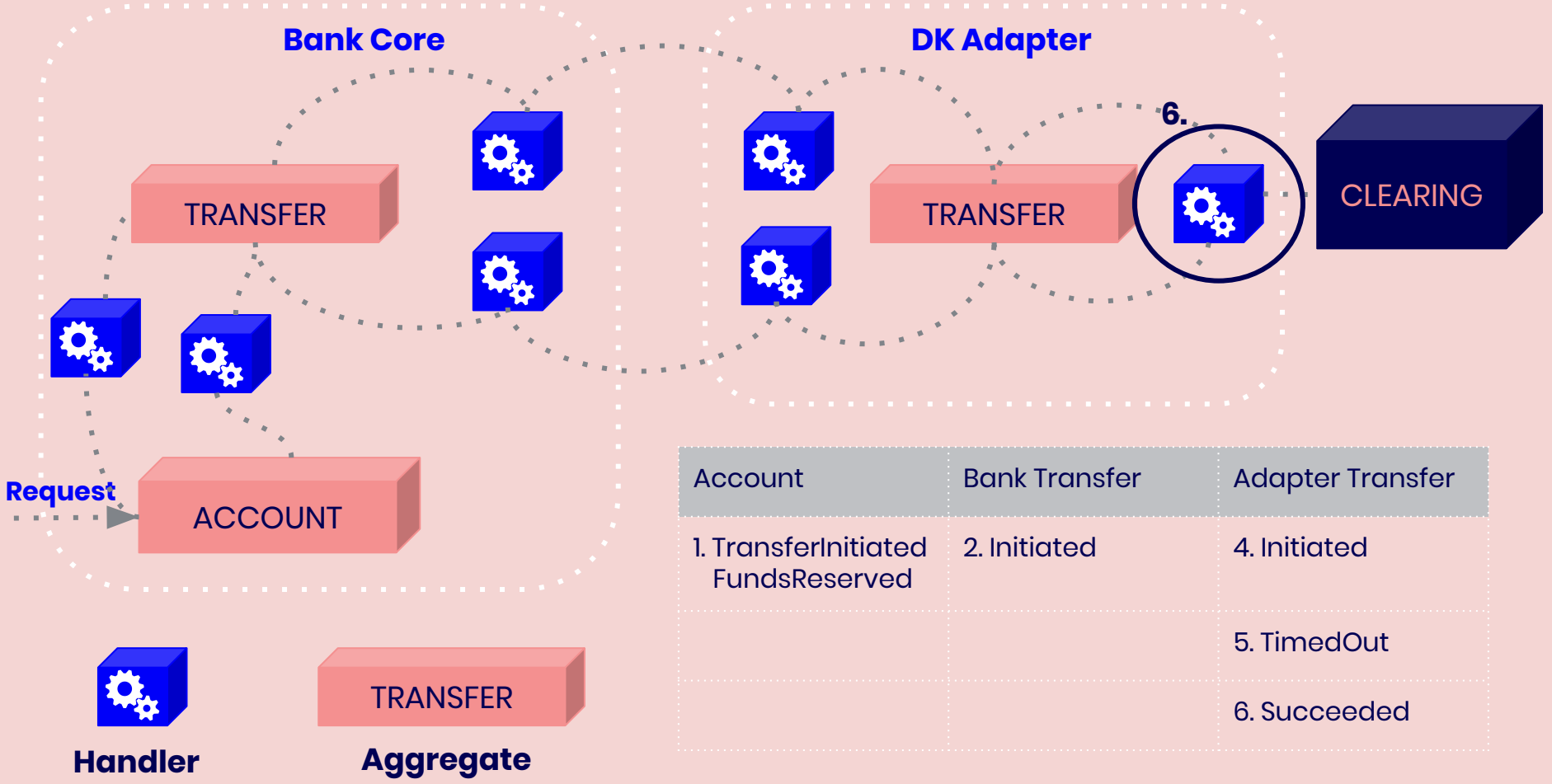
Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	



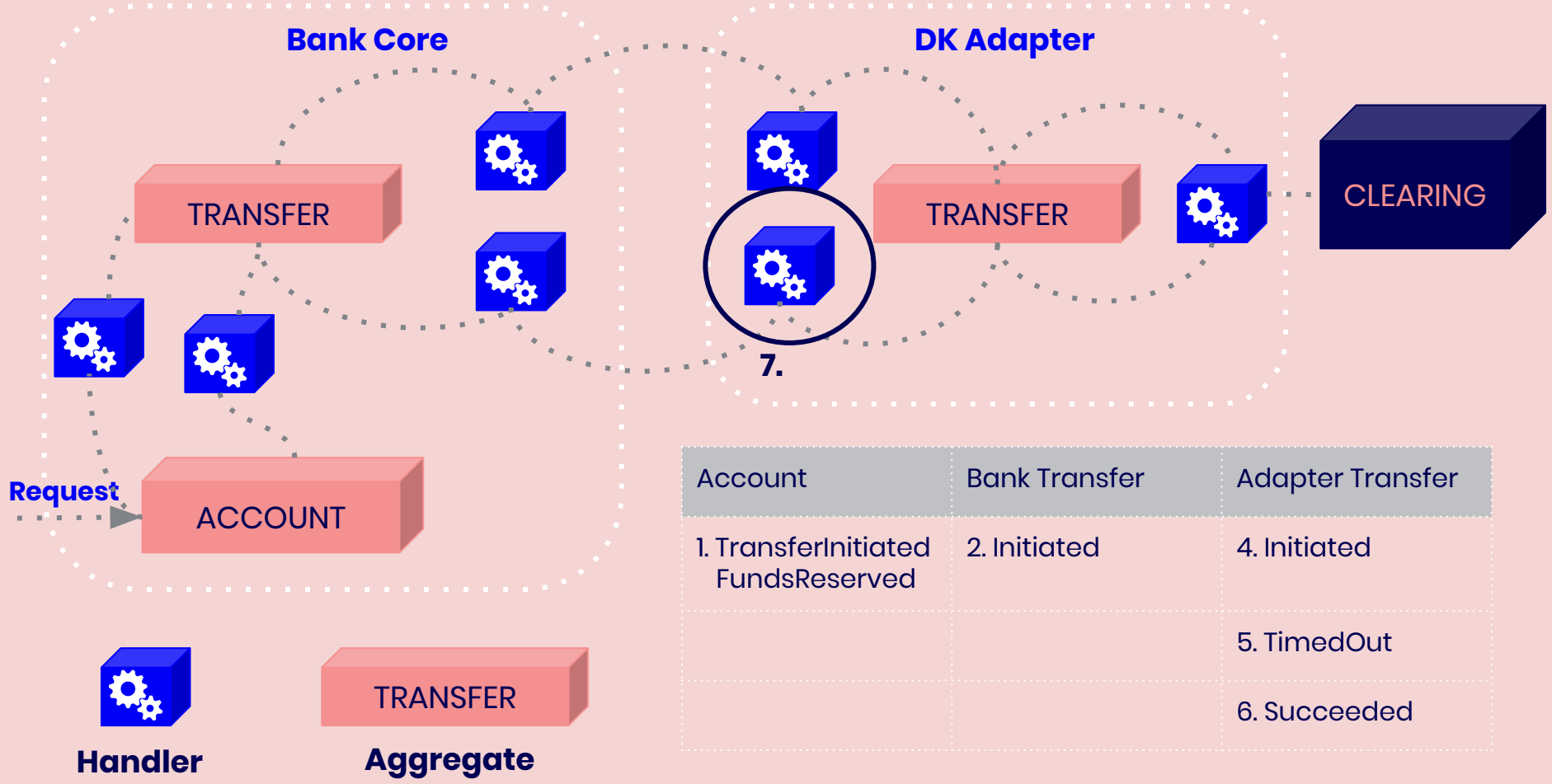
Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	4. Initiated



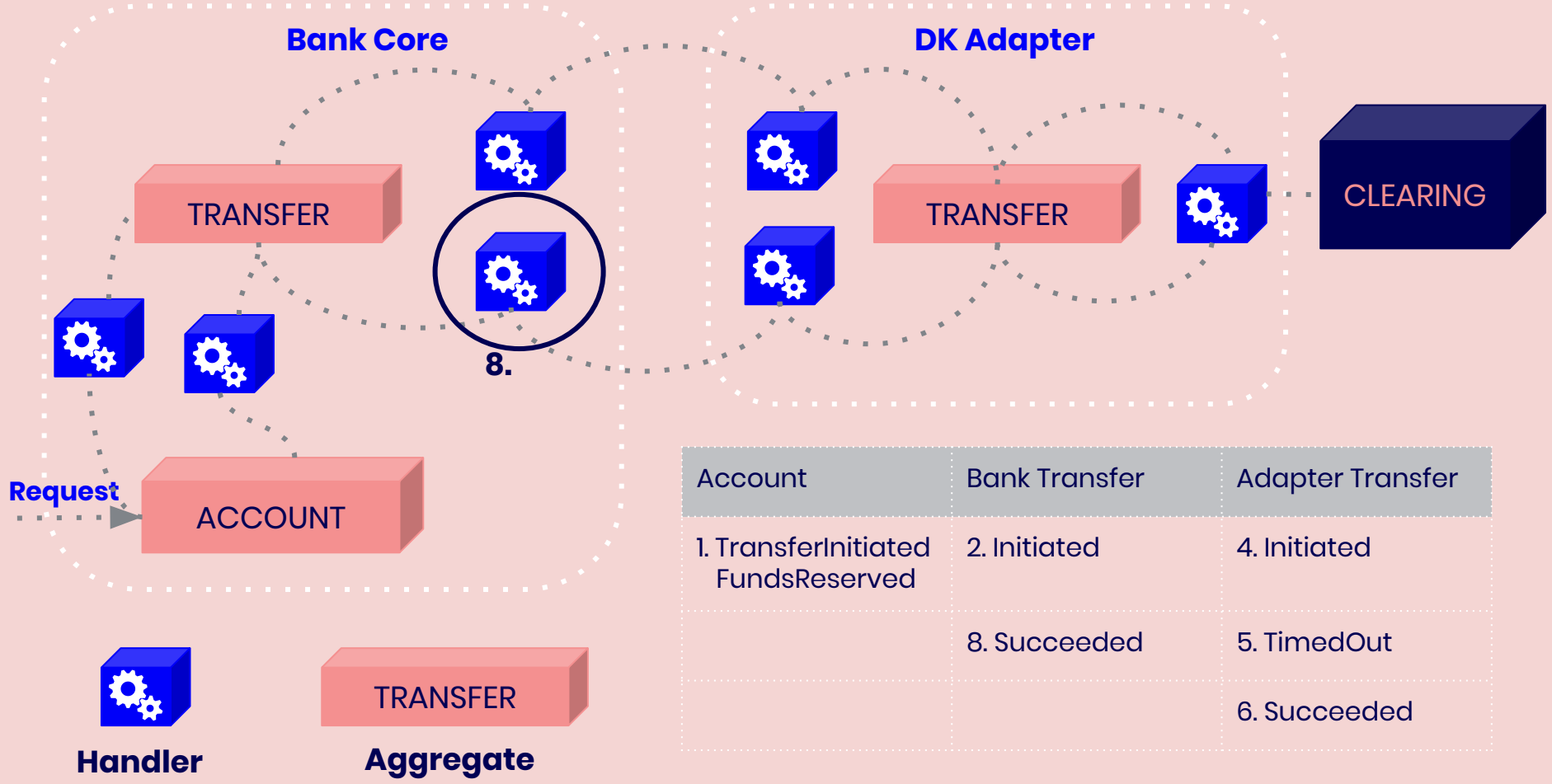
Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	4. Initiated
		5. TimedOut



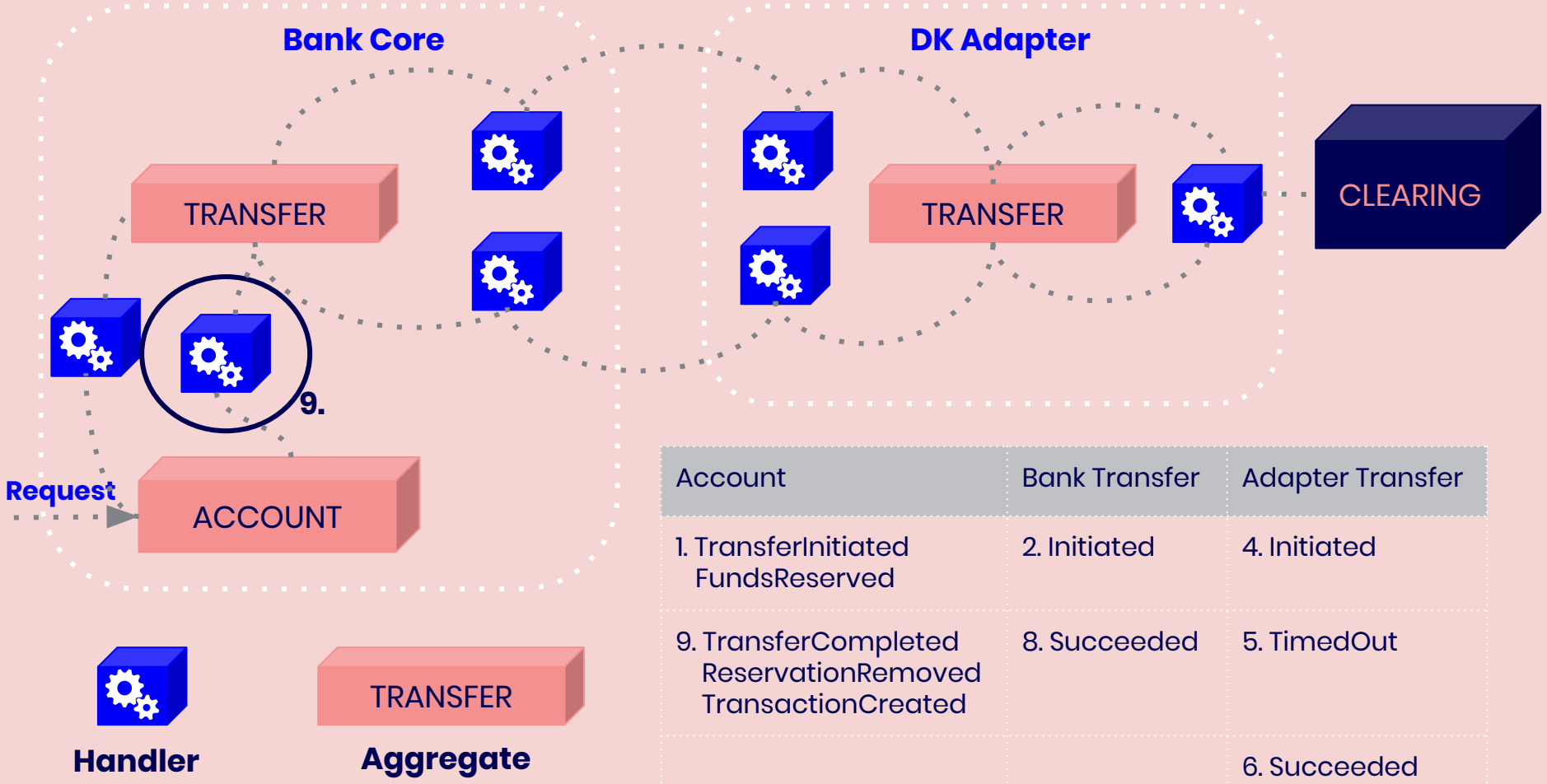
Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	4. Initiated
		5. TimedOut
		6. Succeeded



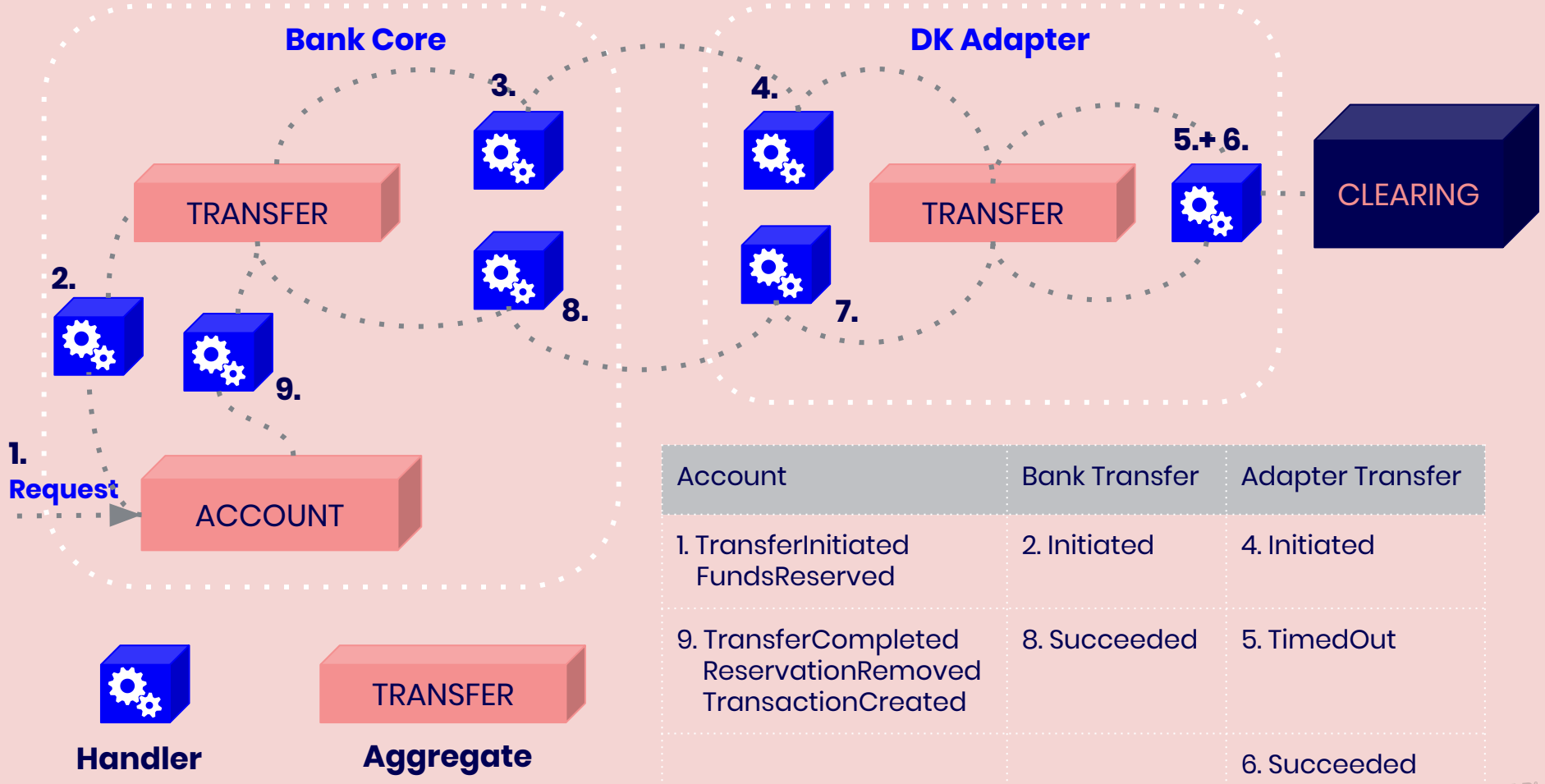
Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	4. Initiated
		5. TimedOut
		6. Succeeded



Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	4. Initiated
	8. Succeeded	5. TimedOut
		6. Succeeded



Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	4. Initiated
9. TransferCompleted ReservationRemoved TransactionCreated	8. Succeeded	5. TimedOut
		6. Succeeded



Account	Bank Transfer	Adapter Transfer
1. TransferInitiated FundsReserved	2. Initiated	4. Initiated
9. TransferCompleted ReservationRemoved TransactionCreated	8. Succeeded	5. TimedOut
		6. Succeeded

Key points

- Guaranteed event handling is paramount
- Maximum traceability

Error handling

- Failures are 1st class domain citizens
- Idempotency* crucial both internally and externally
- Only measure against timeouts and crashes

* Idempotency: system state remains the same after one or several calls

Tech Vision
Revisited

What about Correctness...?

- 100% correctness is impossible
- Understand your errors, then fix
- Traceability leads to correctness by explaining errors
- Error correction is just another event

… and Consistency?

- Ordered event streams
- Empowers consumers
- Eventually is better than maybe

Challenges
& Learnings

Event sourcing is a perfect fit with DDD... but

- It's a different mind set
- May cause mental overflow
- Leave room for experiments and failures

Event sourcing delivers on the promise of traceability… but

- It’s not necessarily mainstream tech
- Guaranteed event handling is challenging

There's tremendous power in immutable event streams and CQRS...

- But immutable data is also a challenge:
  - GDPR
  - Migrations
  - Compensating event

Thank You!

https://tech.lunarway.com/blog/

https://tech.lunarway.com/talks/

https://tech.lunarway.com/opensource/

LUNAR®