Vagif Abilov

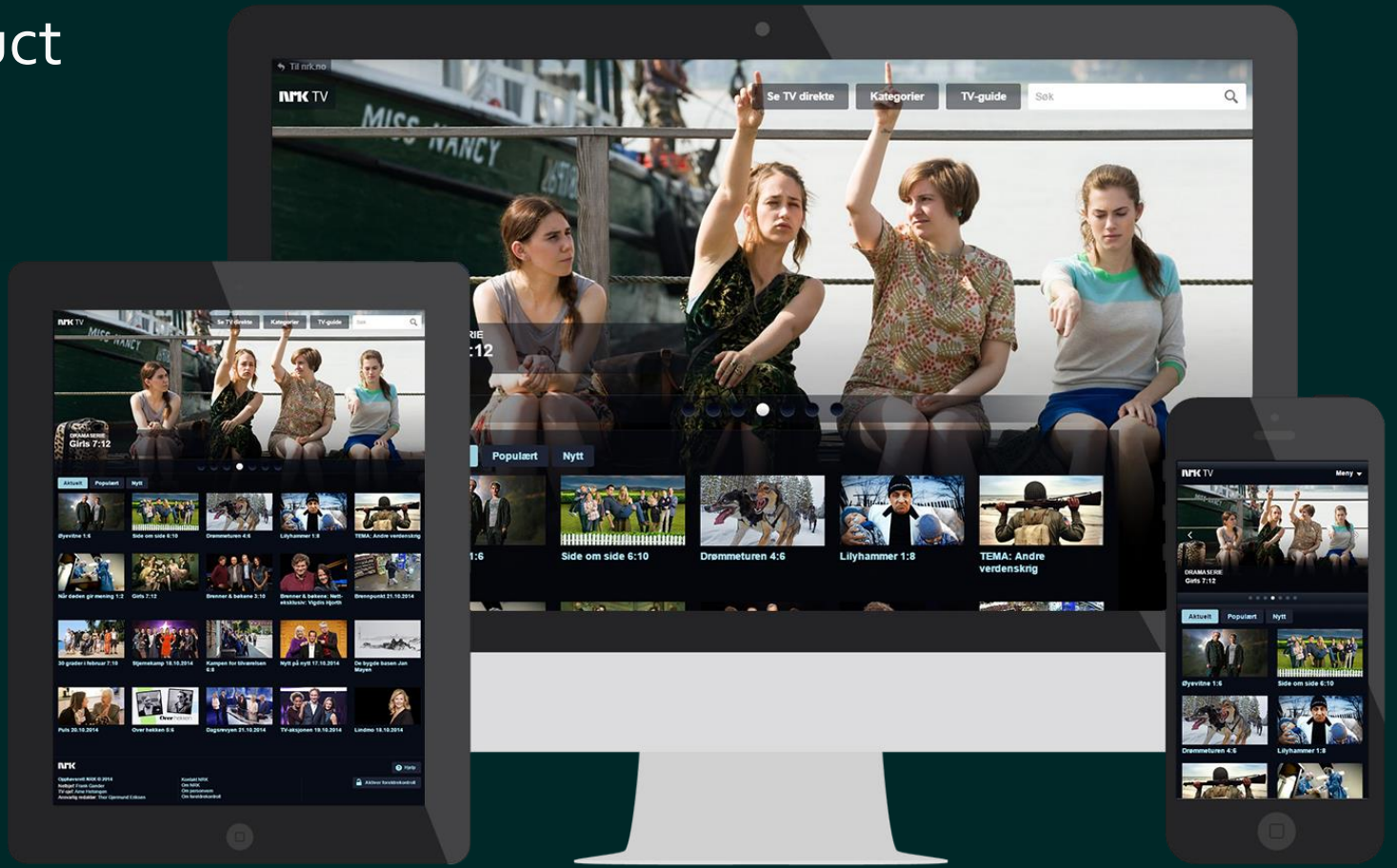
Reliable Messaging in the World of Actors

Vagif Abilov
Consultant in Miles
Oslo, Norway

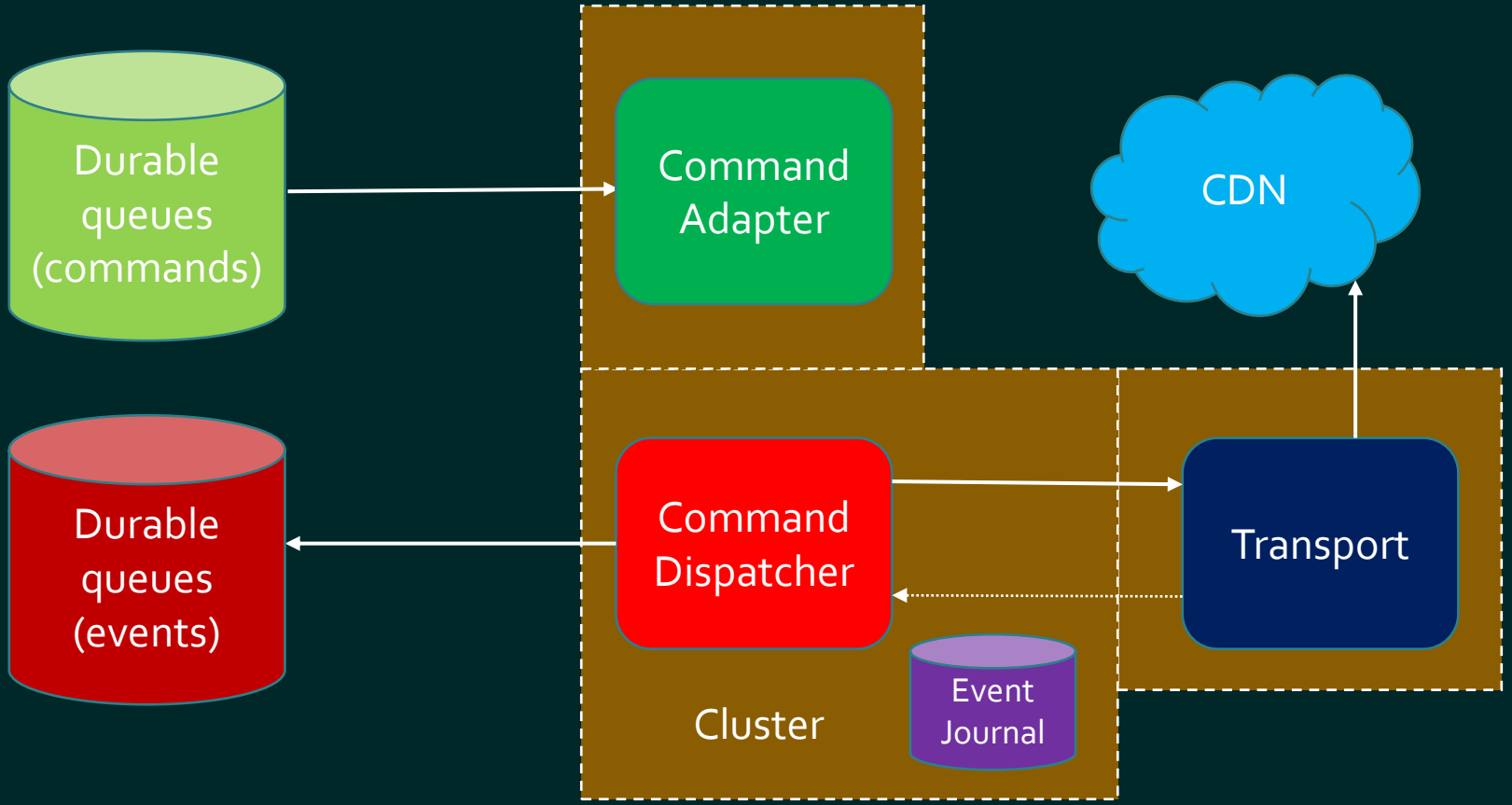
Work with F# and C#

@ooobject
vagif.abilov@mail.com

Our product


Til nrk.no
NRK TV
Se TV direkte
Kategorier
TV-guide
Søk
Populært
Nytt
Side om side 6:10
Drømmeturen 4:6
Lilyhammer 1:8
TEMA: Andre verdenskrig
Girls 7:12
Aktuelt
Populært
Nytt
Meny

Our architecture at large



Our technical stack

- Actor model (using Akka.NET cluster)
- F# as the main programming language
- RabbitMQ and Azure Service Bus as durable queues
- Both SQL and NoSQL databases to store persistent data

This talk is a retrospective of changes in our approach to message handling guarantee

How do we provide
message handling guarantee?

Semantics of delivery guarantees

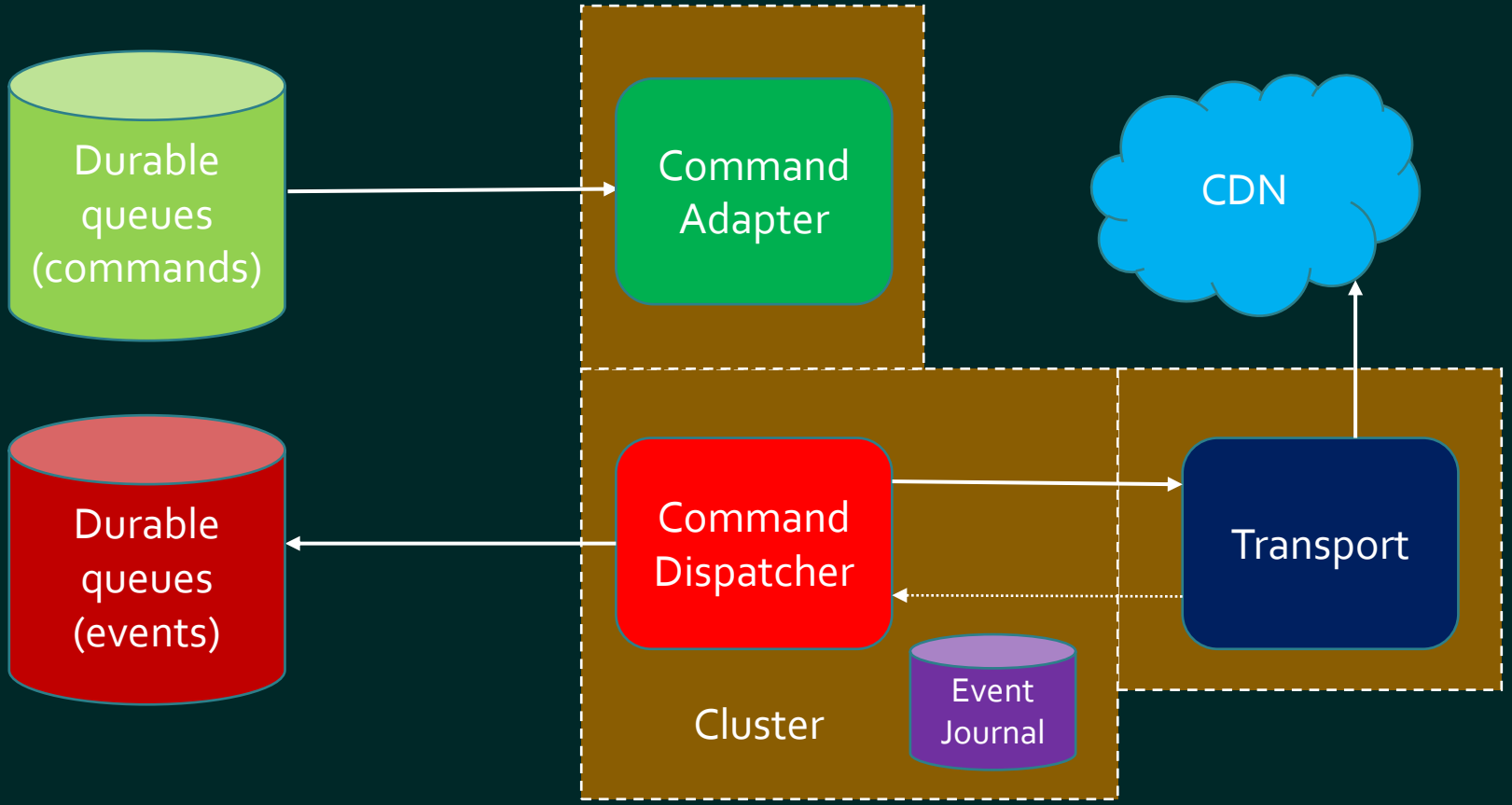
- At-most-once delivery: each message handed to the system is delivered once or not at all (i.e. messages may be lost)
- At-least-once delivery: each message handed to the system may potentially be attempted to be delivered multiple times (i.e. messages may be duplicated but not lost)
- Exactly-once delivery: for each message handed to the system exactly one delivery is made to the recipient (i.e. the message can neither be lost nor duplicated)

Message delivery rules in actor systems

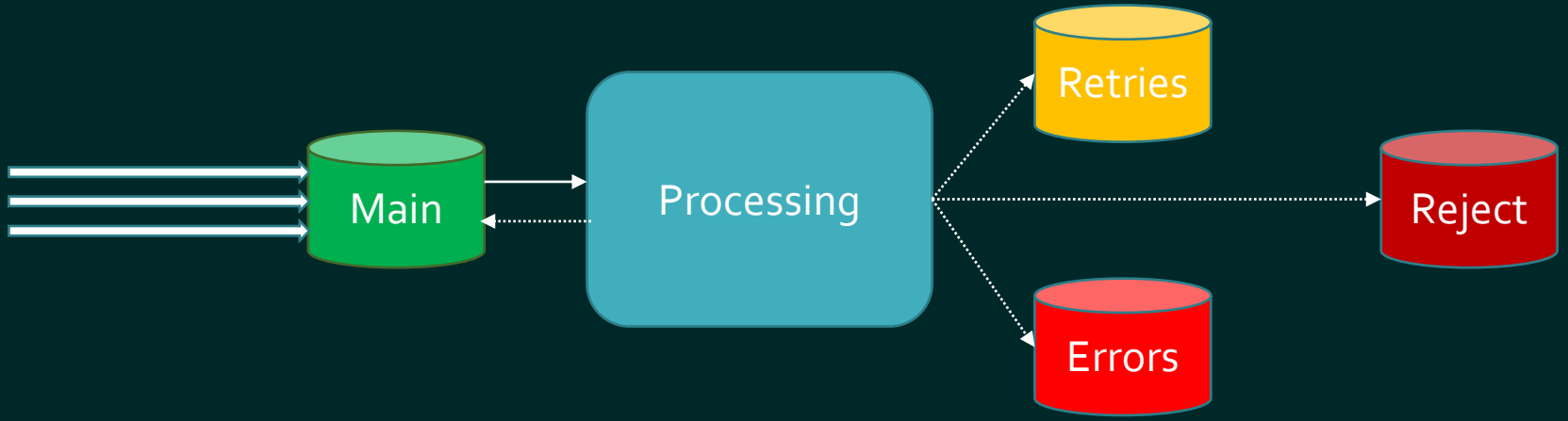
- At-most-once delivery- all major actor system implementations
- Message ordering per sender-receive pair (e.g. Akka)
- Some systems may be configured to achieve at-least-once delivery using infinite retries (Orleans)

Can we run a reilable system
with at-most-once message delivery?

Our architecture at large



Message queue configuration



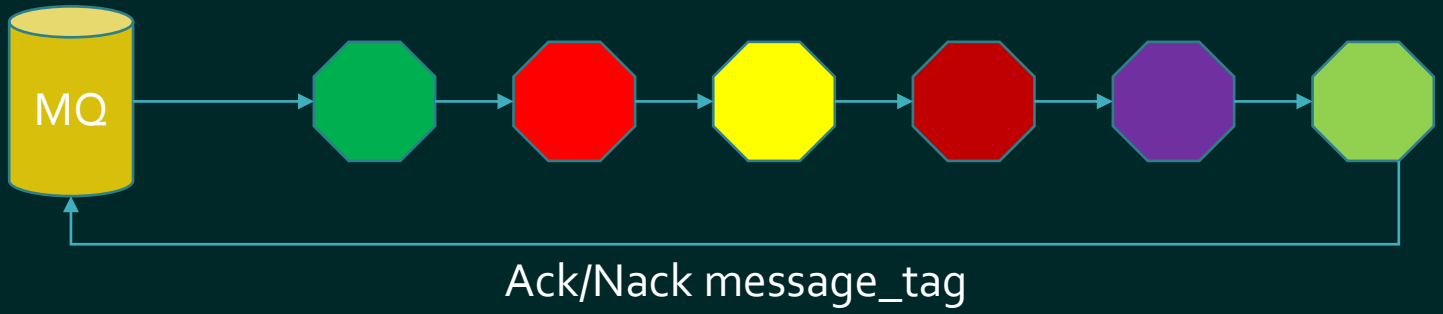
dm.ps.subtitles.prod	rabbit@malxodaramq01.felles.ds.nrk.no +2	classic	D DLX Pri **all**	idle
dm.ps.subtitles.prod.errors	rabbit@malxodaramq01.felles.ds.nrk.no +2	classic	D TTL DLX Pri **all**	idle
dm.ps.subtitles.prod.rejected	rabbit@malxodaramq01.felles.ds.nrk.no +2	classic	D Pri **all**	idle
dm.ps.subtitles.prod.retries	rabbit@malxodaramq01.felles.ds.nrk.no +2	classic	D TTL DLX Pri **all**	idle

Version 1

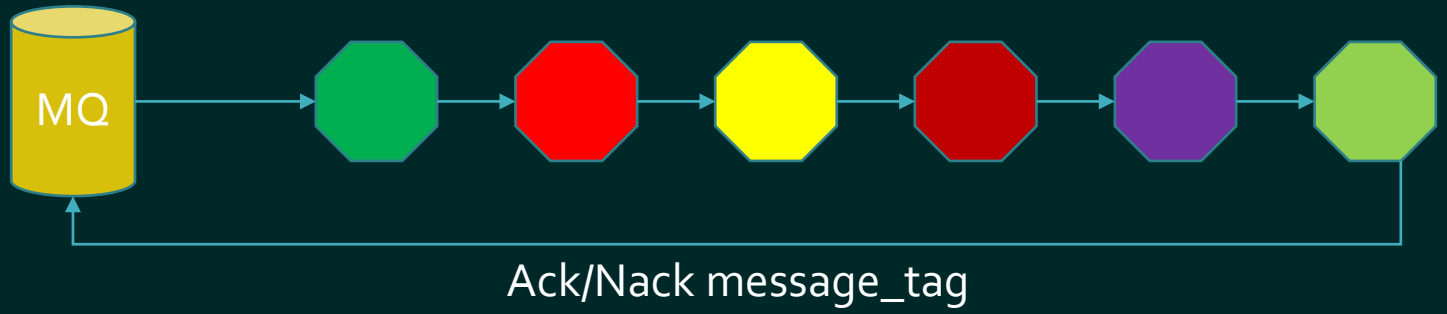
Don’t pay the ferryman

.. until he gets you to the other side

Acknowleding queue messages

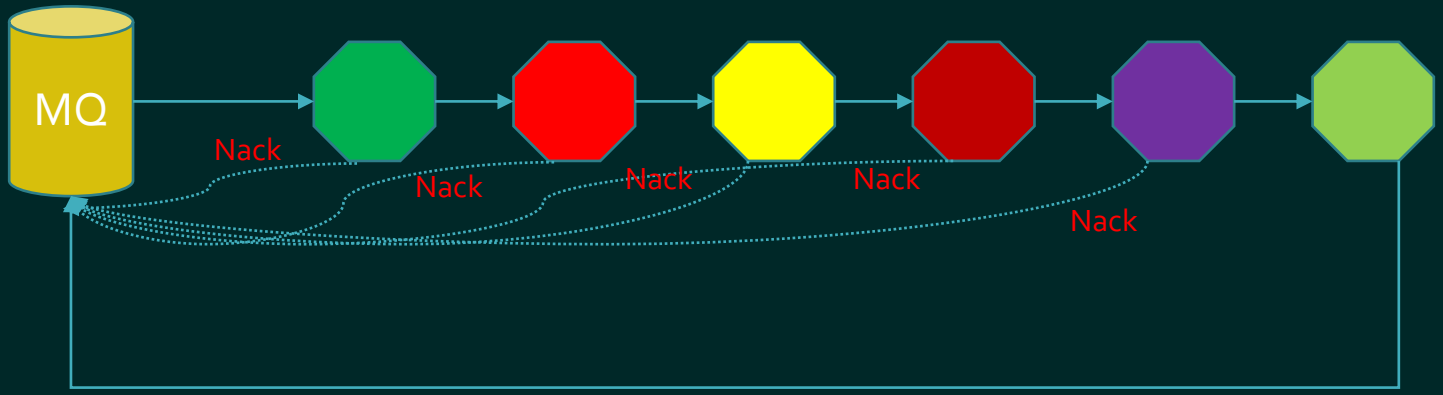


Acknowleding queue messages



NB! This is not a transactional semantics, this is at-least once delivery

Acknowleding queue messages



Return Address pattern

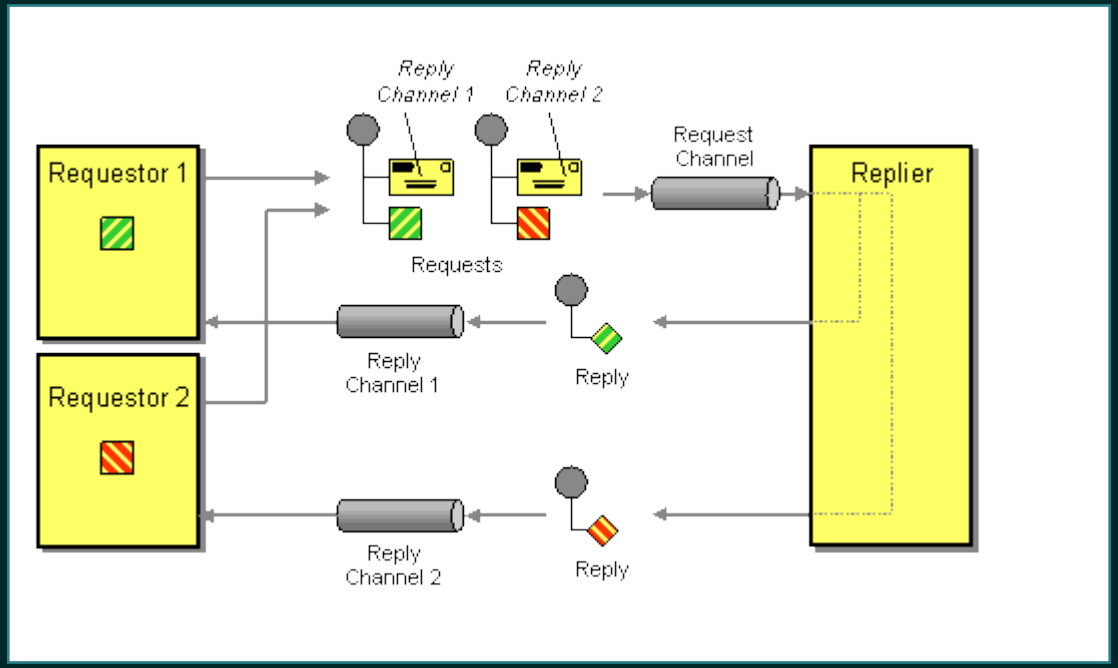


Illustration from «Enterprise Integation Patterns» book by Gregor Hohpe & Bobby Woolf

Message envelope

Drawbacks of using queue acknowledgement to acknowledge workflow completion

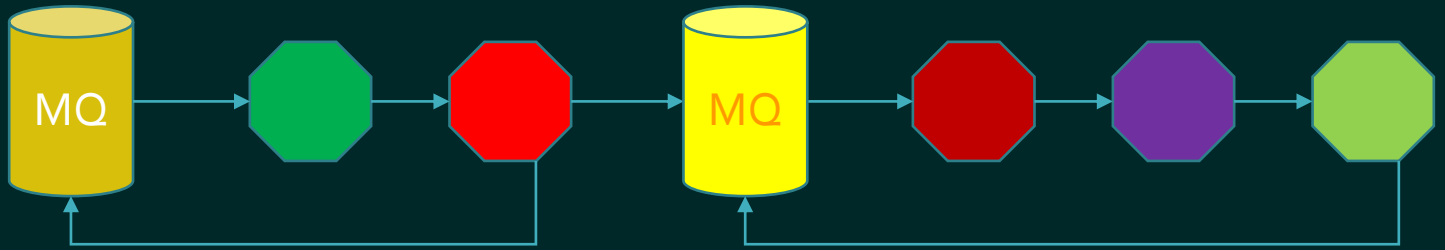
- Messages must be wrapped into envelopes that contain acknowledgement Id
- Last actor in chain is responsible for ack but every actor in chain can issue nack
- Actors in the middle of the chain must not issue ack
- Operations can be long running, queues may be configured with TTL policies
- Message processing workflow can contain forks and joins

How long should a postman wait?

Version 2

Durable queues as persistent bookmarks

Durable queues for delivery guarantees



At-least-once

At-least-once

Drawbacks of sending actor-to-actor messages using message queues

- Messages must be wrapped into envelopes that contain acknowledgement Id
- An actor system gets partitioned
- Some units of work may still be long running and exceed TTL values configured for message queues

Again: how long should a postman wait?

Version 3

At-Least-Once Delivery
Akka extensions

Actors with at-least-one delivery semantics

- Based on persistent actors
- Messages include DeliveryId
- Each delivery requires confirmation from a recipient
- Behavior configuration
  - Redelivery burst limit
  - Warnings on unconfirmed delivery attempts
  - Maximum unconfirmed messages
- After working with durable queues (RabbitMQ, Azure ServiceBus) feels like poor man durable queues

Don’t implement at-least-once delivery semantics using actors just to put actors everywhere
Use technology that is built for it
(durable message queues)

Version 4 (current)

It’s all about fulfillment of a Desired State

«Nobody Needs Reliable Messaging»

Marc de Graauw
https://www.infoq.com/articles/no-reliable-messaging/

“Nobody Needs Reliable Messaging”

“If reliability is important on the business level, do it on the business level”

Transactions and Fiefdoms

In a system that cannot count on distributed transactions, the management of uncertainty must be implemented in the business logic

Pat Helland
https://pathelland.substack.com/p/autonomous-computing-short-version

Work happens with a sequence of related messages over time to perform cooperative work. This is how it was done centuries ago and it's how it's done today.

Pat Helland
https://pathelland.substack.com/p/autonomous-computing-short-version

Reliable collaborations with unreliable messages

Reliable collaborations with unreliable messages

1. Receive an incoming request
2. Evaluate the desired state of your aggregate root
   - Desired state must include information about expected outgoing messages (use Outbox pattern)
3. Persist the desired state
4. Acknowledge the received message
5. Proceed with the request execution

Outbox pattern

Outbox Pattern ensures that the application state (stored in the application database) and its respective domain event (forwarded to the external consumers) are consistent and durable under a single transaction

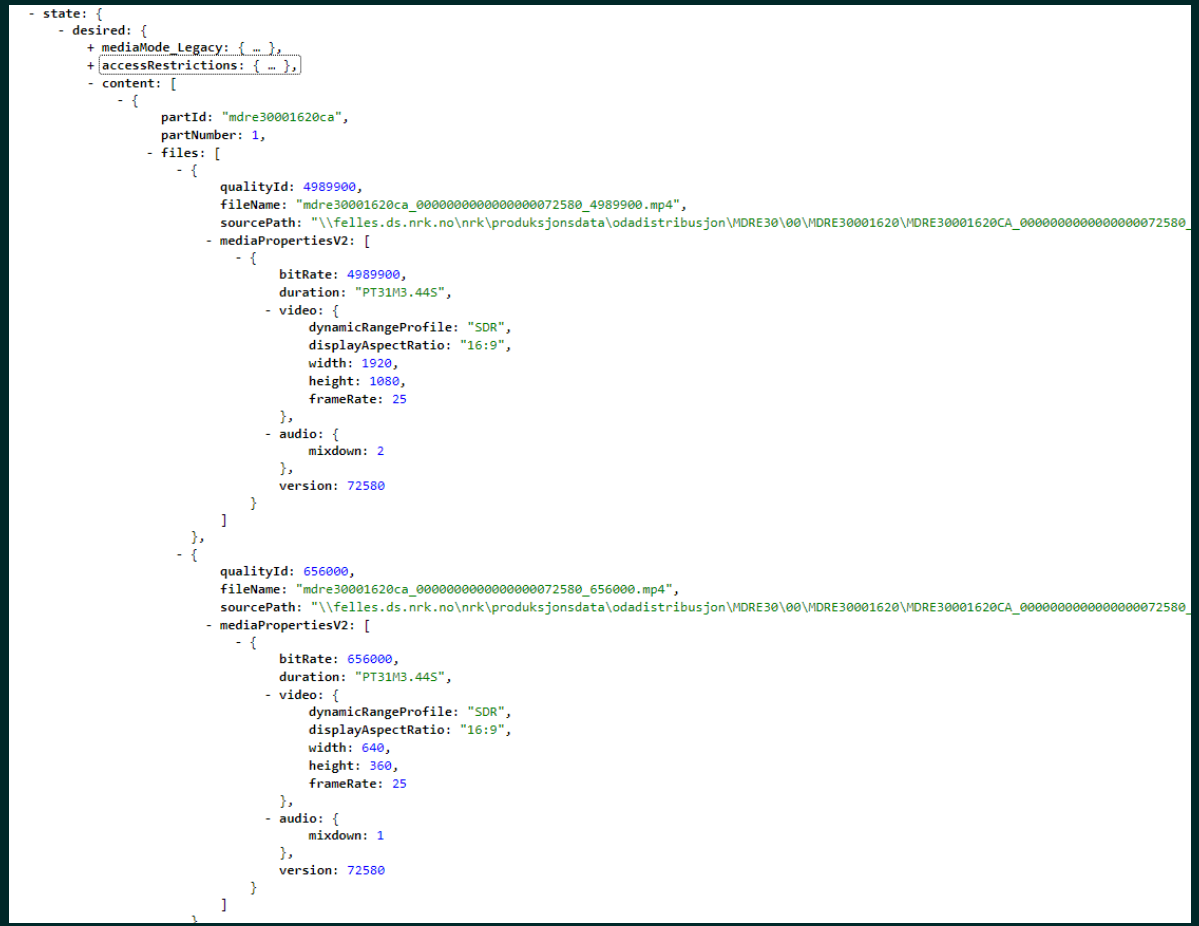
Request execution

1. Evaluate the desired state
2. Evaluate the current state
3. Work = Current state – Desired state

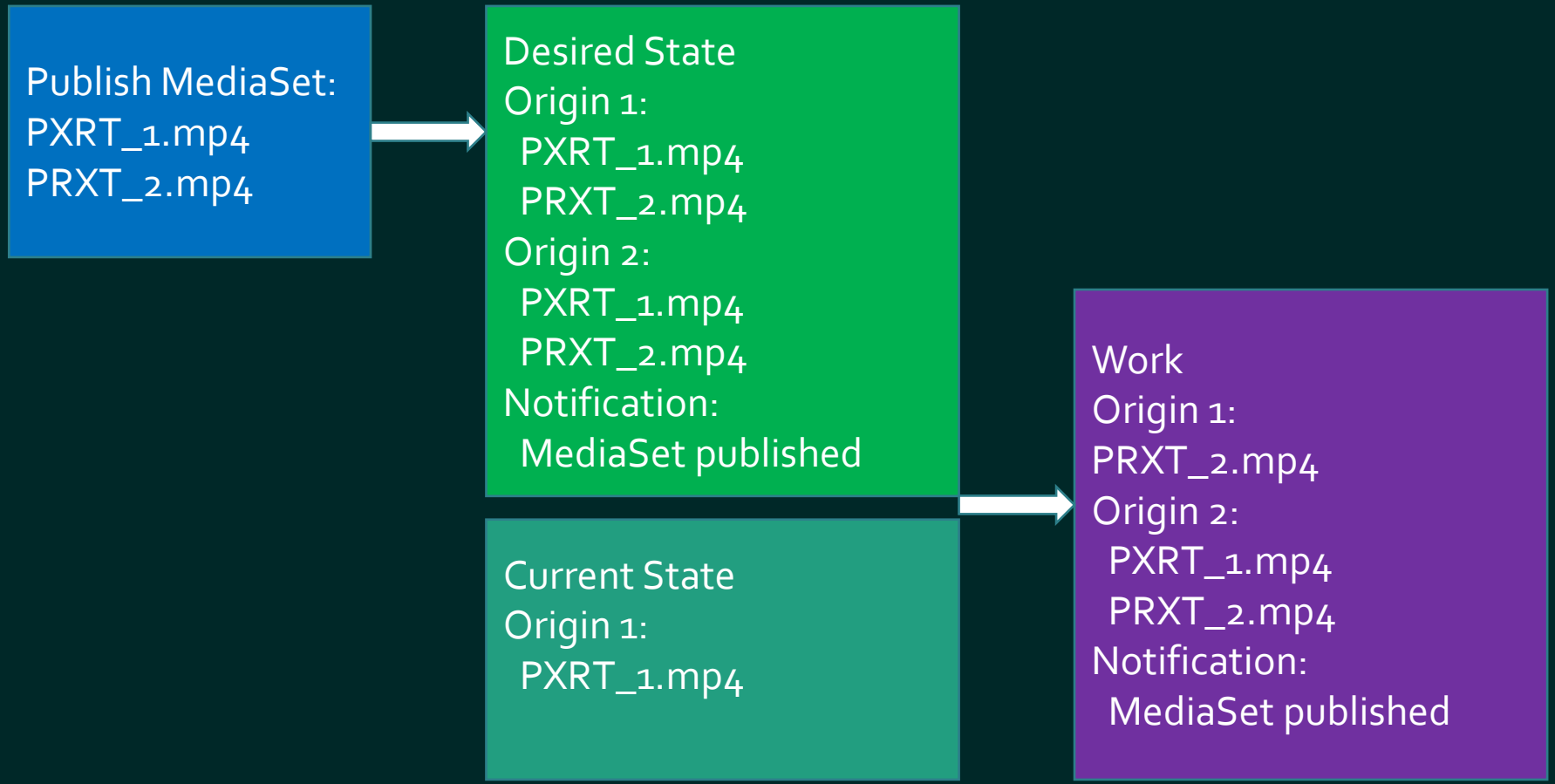
Desired state concept

- One of the core concepts of Kubernetes
  - You describe the state of the objects that will run the containers
  - Kubernetes are in charge of regulating the state of the system
- PowerShell Desired State Configuration is a configuration management platform
  - Decrease the complexity of scripting
  - Increase the speed of iteration

Important assumption: idempotency


- state: {
- desired: {
+ mediaMode_Legacy: { … },
+ accessRestrictions: { … },
- content: [
- {
partId: "mdre30001620ca",
partNumber: 1,
- files: [
- {
qualityId: 4989900,
fileName: "mdre30001620ca_00000000000000000072580_4989900.mp4",
sourcePath: "\\felles.ds.nrk.no\nrk\produksjonsdata\odadistribusjon\MDRE30\00\MDRE30001620\MDRE30001620CA_00000000000000000072580_
- mediaPropertiesV2: [
- {
bitRate: 4989900,
duration: "PT31M3.44S",
- video: {
dynamicRangeProfile: "SDR",
displayAspectRatio: "16:9",
width: 1920,
height: 1080,
frameRate: 25
},
- audio: {
mixdown: 2
},
version: 72580
}
]
},
- {
qualityId: 656000,
fileName: "mdre30001620ca_00000000000000000072580_656000.mp4",
sourcePath: "\\felles.ds.nrk.no\nrk\produksjonsdata\odadistribusjon\MDRE30\00\MDRE30001620\MDRE30001620CA_00000000000000000072580_
- mediaPropertiesV2: [
- {
bitRate: 656000,
duration: "PT31M3.44S",
- video: {
dynamicRangeProfile: "SDR",
displayAspectRatio: "16:9",
width: 640,
height: 360,
frameRate: 25
},
- audio: {
mixdown: 1
},
version: 72580
}
]

```
- current: {
    - akamaiStorage: {
          volumeId: "13",
          edgeChar: "c",
          timestamp: "2021-03-02T18:08:34.8035764+01:00"
      },
    - akamaiFiles: [
        - {
            partId: "mdre30001620ca",
            qualityId: 208000,
          - file: {
              sourcePath: "\\manas01\odadistribusjon$\MDRE30\00\MDRE30001620\MDRE30001620CA_00000000000000072580_ID180.mp4",
              directoryPath: "mdre30001620~mdre30001620ca",
              cdnPath: "http://nordond13c-f.akamaihd.net/z/no/open/ps/md/mdre30001620/mdre30001620ca/mdre30001620ca_208000.mp4",
              version: 72580
            },
            state: 5,
          - lastResult: {
              removed_ResultCode: 0,
              resultCode: 0
            },
            timestamp: "2021-11-23T08:58:10.1554439+01:00"
        },
        - {
            partId: "mdre30001620ca",
            qualityId: 381000,
          - file: {
              sourcePath: "\\manas01\odadistribusjon$\MDRE30\00\MDRE30001620\MDRE30001620CA_00000000000000072580_ID270.mp4",
              directoryPath: "mdre30001620~mdre30001620ca",
              cdnPath: "http://nordond13c-f.akamaihd.net/z/no/open/ps/md/mdre30001620/mdre30001620ca/mdre30001620ca_381000.mp4",
              version: 72580
            },
            state: 5,
          - lastResult: {
              removed_ResultCode: 0,
              resultCode: 0
            },
            timestamp: "2021-11-23T08:58:11.7731315+01:00"
        },
```


Publish MediaSet:
PXRT_1.mp4
PRXT_2.mp4
Desired State
Origin 1:
PXRT_1.mp4
PRXT_2.mp4
Origin 2:
PXRT_1.mp4
PRXT_2.mp4
Notification:
MediaSet published
Current State
Origin 1:
PXRT_1.mp4
Work
Origin 1:
PRXT_2.mp4
Origin 2:
PXRT_1.mp4
PRXT_2.mp4
Notification:
MediaSet published

But actors are reactive
How do they ensure all work is fulfilled?
What wakes them up on a system crash?

Extending processing workflow

1. Receive an incoming request
2. Evaluate the desired state
3. Schedule a repeating reminder
4. Persist the desired state
5. Acknowledge the received message
6. Proceed with the request execution

Reminder message may be simply a trigger to wake up the actor

What was I supposed to do today?

When an aggregate root actor wakes up

1. Replay actor state from the event journal (state recovery)
2. Evaluate remaining work (Current state – Desired state)
3. Remaining work = Nothing?
   - Yes -> Cancel the repeating reminder
   - No -> Proceed with the request execution

Main lesson learned when modeling actors' behavior

Consider using patterns established in real world collaborations

The real world is solid

Rant about workflow engines and saga managers

You don’t need them

Check out talk «Events, Workflows, Sagas?» by Lutz Huenhnken at kanDDDinsky.de

Conclusion

- Actors use at-most-once delivery for good reasons, let them stay quick and responsive
- Don't use durable message queues as transaction guards
  - Acknowledge messages on receiving, not on completion of the request they imply
- Persist the intention and record the triggered operations outcome, these form Desired and Current state, then you can always evaluate the remaining work

Thank you!

Vagif Abilov
Consultant in Miles

Github: object
Twitter: @ooobject
vagif.abilov@mail.com