From Event Driven to Event Sourcing

by

Thomas Bøgh Fangel &
Emil Krog Ingerslev

Who are we?

Emil Krog Ingerslev 🐦 @emilingerslev

Site reliability engineer & architect @lunarway

Works with Thomas on backend of the future

❤️ reliable efficient automated software

coffee geek ☕

Talks... a lot

Thomas Bøgh Fangel @tbfangel

Architect @lunarway

Works with Emil on the backend of the future

Always looking for a better design

❤️ to write and talk about what we do

Talks... less

lunar way®

?

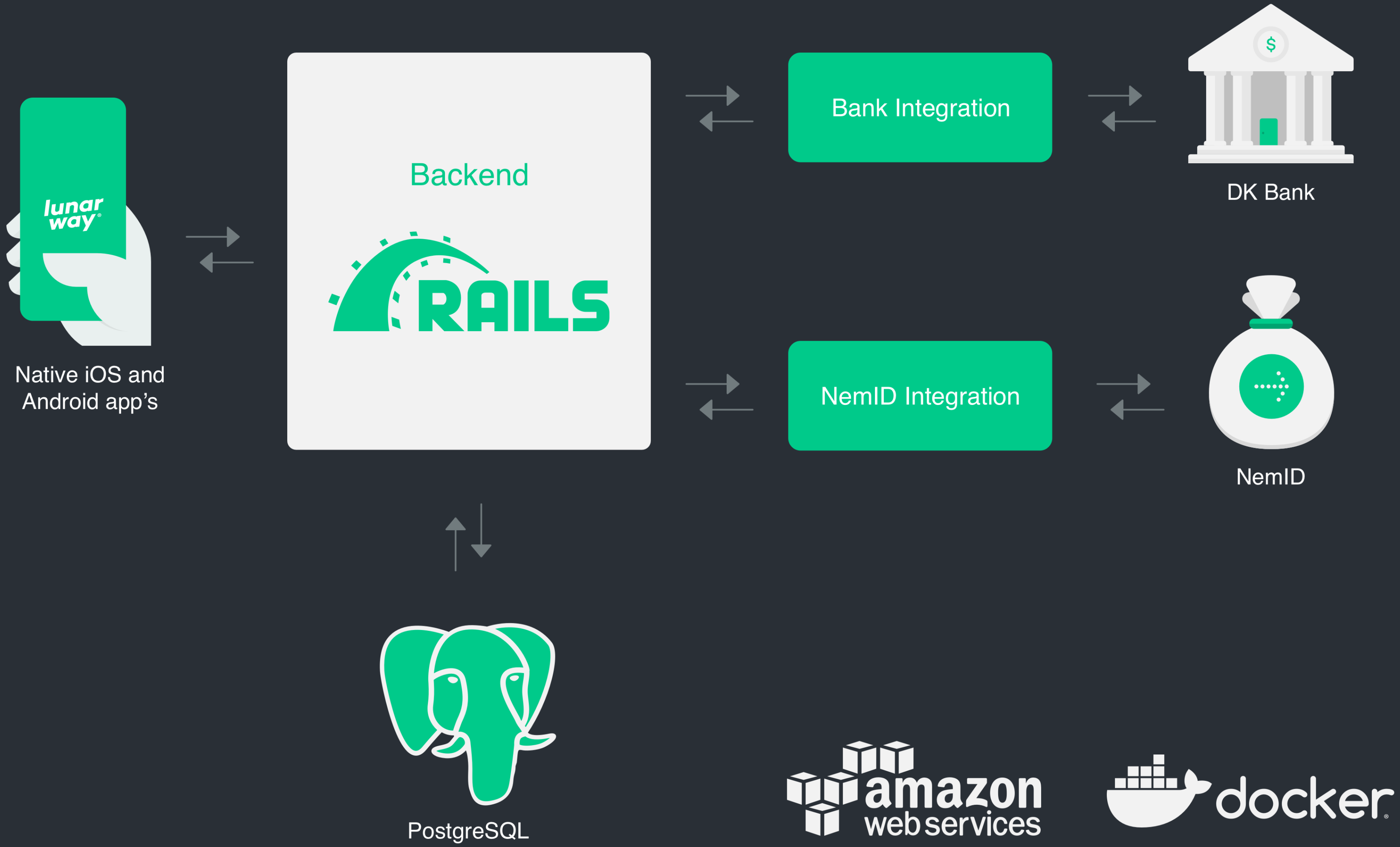
lunar way® in numbers

80.000+	10M+	+75
Users	Transcations	Microservices
80+	1.000M+	3
Employees	€ Volume	K8S clusters

Platform

Evolution

From monolith...


lunar way
Native iOS and Android app's
Backend
RAILS
Bank Integration
DK Bank
NemID Integration
NemID
PostgreSQL
amazon web services
docker

… to event driven microservices …

75+
microservices

async messages
communication pattern

10+ integrations
to 3rd parties

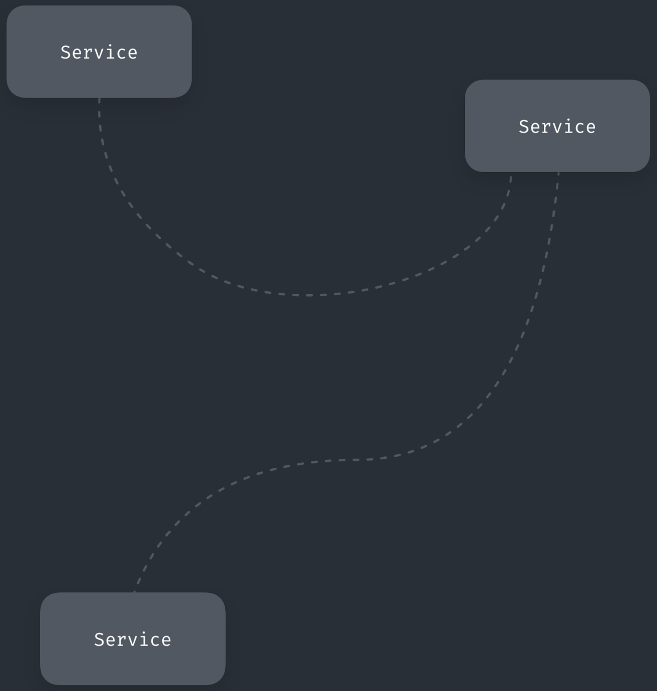
50+
daily deployments

… to event sourcing with consistency guarantees

Identified some problems

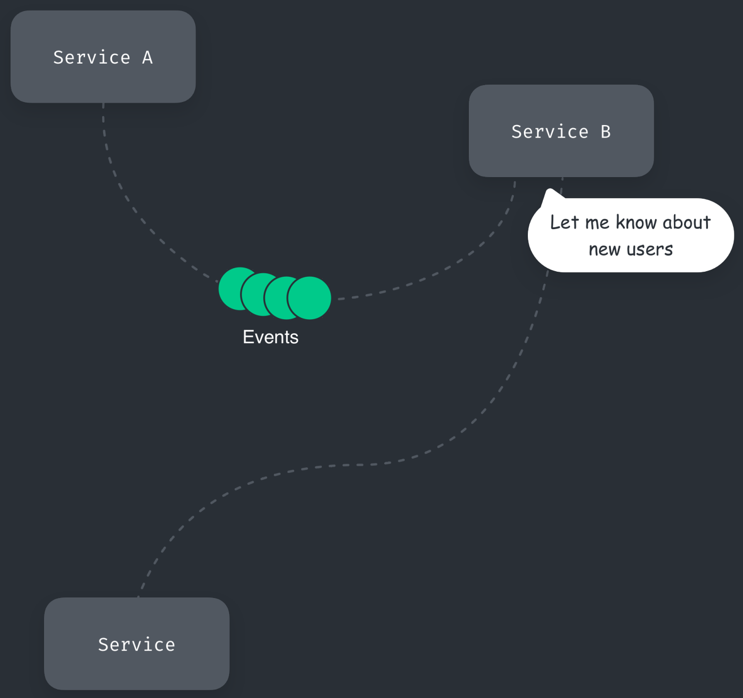
found desired characteristics

how we implemented them



a fictive overview

not showing
message queues
databases
etc...


Service A
Service B
Let me know about new users
Events
Service

We built services CRUD like

Behavior:
Step 1 - do change in DB
Step 2 - publish message

Consistency problem

do change in DB
service fails
event NOT published

Consistency problem

relied on broker
receiver in dark
publisher in dark

Consequence

receiver never gets event
zero-or-once delivery
actually zero-or-more!

services state drift
weird support cases 😲
hours of logs scrolling 😒
synthetic symptom fixes

Imagine some other characteristics 🤔

“Atomic state change and message publication”

+

“At-least-once delivery”

Event sourcing as a solution

Every event IS the state change
= Atomic event generation & state change

Guaranteed event publishing

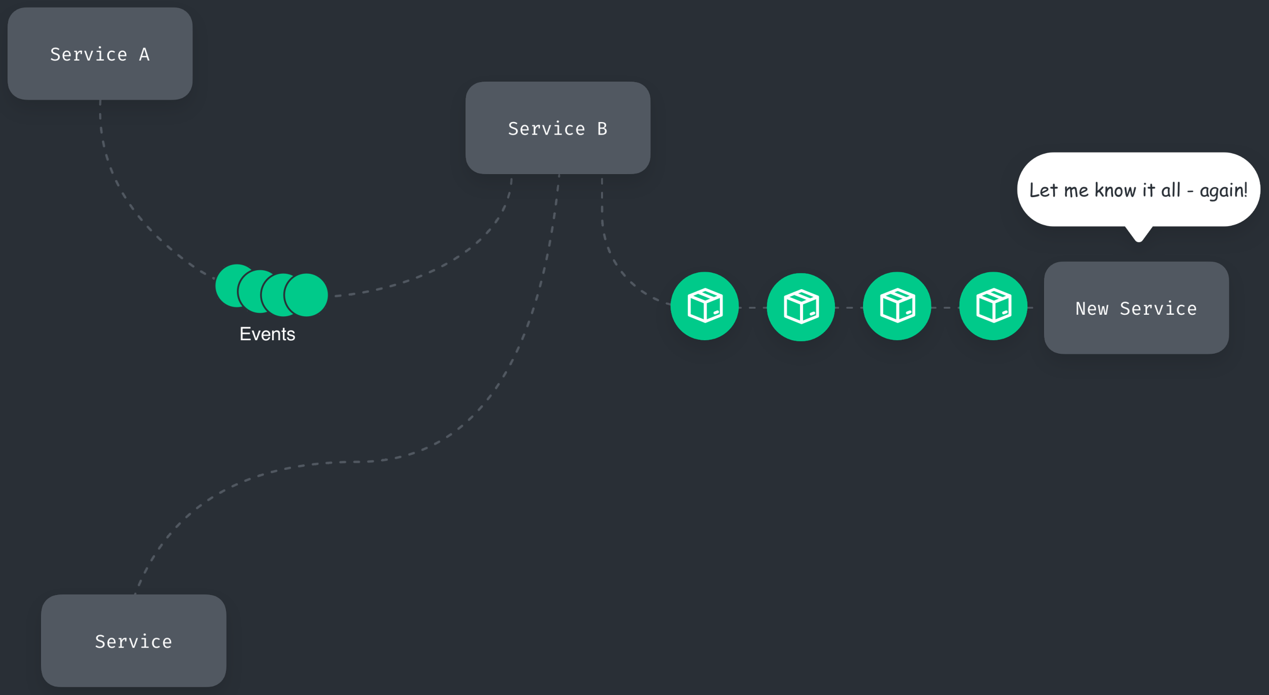
Read stream
Got event #1
Publish event #1
Save cursor #1
continue to #2
crash

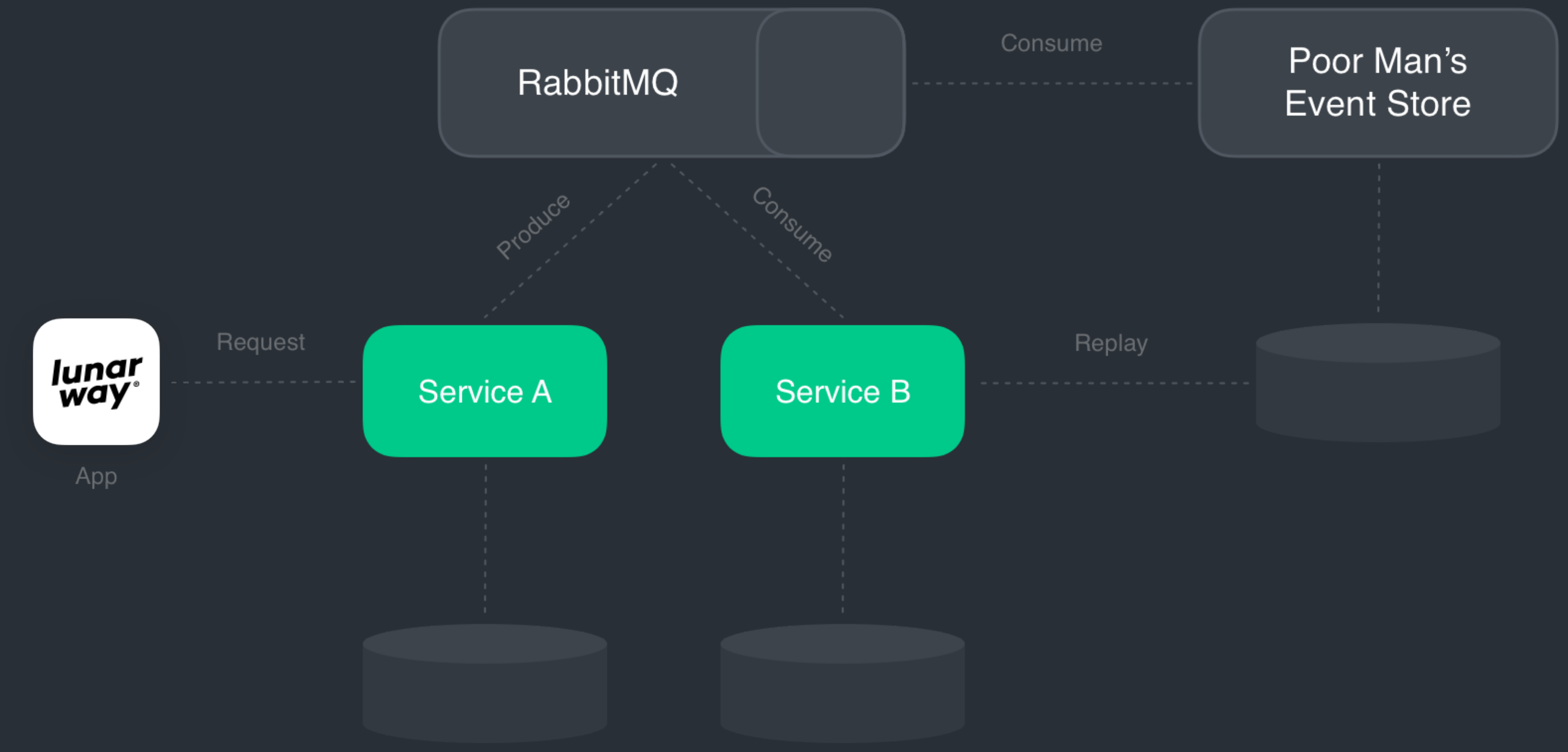
Guaranteed event publishing

Starting up
Read cursor #1
Read stream from > #1
Got event #2
Publish event #2
Save cursor #2

Guaranteed

atomic state change + event publishing
at least once event publishing


Service A
Service B
Events
Let me know it all - again!
New Service
Service


RabbitMQ
Consume
Poor Man's Event Store
Produce
Consume
lunar way
App
Request
Service A
Service B
Replay

Bootstrapping
the old way

replay from Poor Man
synthetic events

Bootstrapping problems

manual process
availability of events
consistency
handling load

🤔 Can we do better?

“Events as first class citizens”

+

“Event streams with possibility of redelivery”

=

Bootstrap galore

Event sourcing as a solution

"Every event IS the state change"

+

API on top of event stream

=

Events as first class citizens
with
event streams with redelivery

Event Sourcing Patterns

Bootstrap on-demand

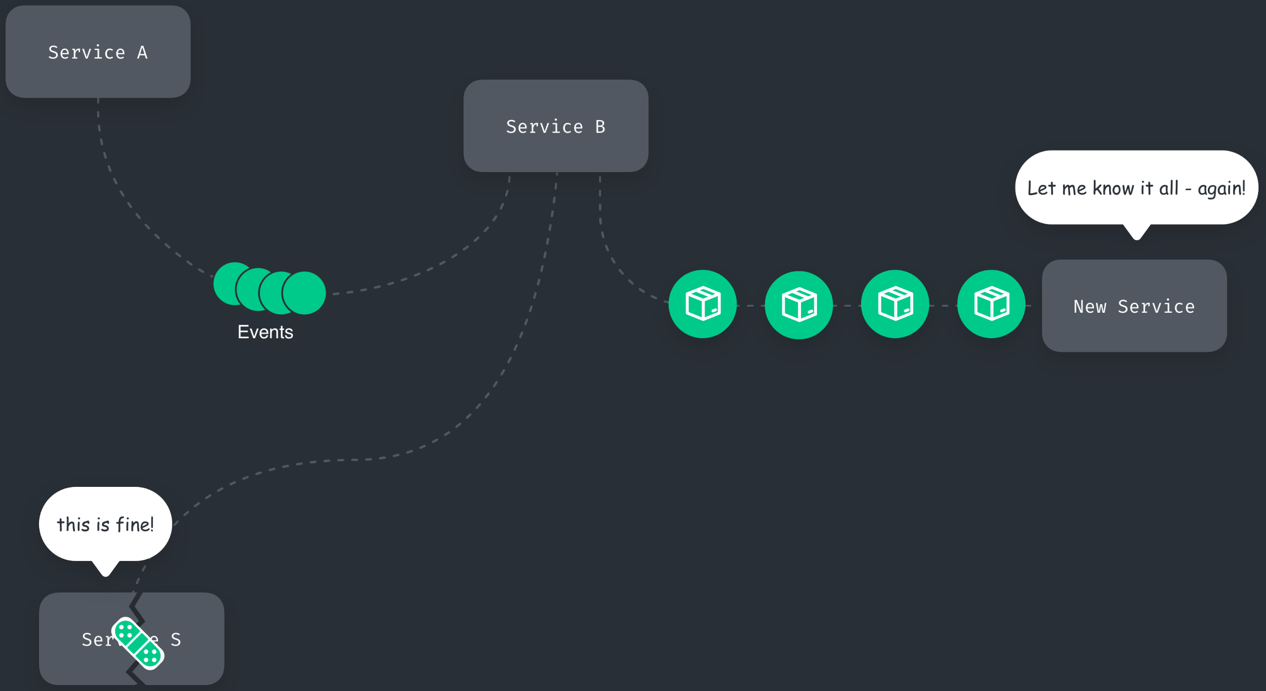
Integration events on the outside

Integration Events are
a projection of internal events
with same characteristics

so whats the purpose? 🤓

Less coupling
Producer ↔ Consumer

Producer	New Consumer
...	Gets request from **User #1**
...	Asks for events for **User #1**
Finds event stream for **User #1**	...
Hydrates integration events 💦	...
Sends events back	...
...	Hydrates **User #1** view
...	Responds to request


Service A
Service B
Events
Let me know it all - again!
New Service
this is fine!
Ser   e S

Healing problem

Service listens for events

Receives event #1

event #2 is sent, but is lost

Receives event #3

State has drifted

Consequences

Drifting state

Consumer left in dark

Support cases

Sync logic to fix problems

Desired characteristics

Heal our broken state

Know if events are missing

Redelivery of missing events

"Event streams with possibility of redelivery"

Take 2 🤓

"Ordered event streams with possibility of redelivery"

Revisiting the event sourcing patterns

Bootstrapping was about redelivery from scratch
Redelivery from any event

Self healing

Service listens for events

Receives event #1

Moves cursor to #1

Receives event #3

Request events since #1

Get event #2

Moves cursor to #2

Continue on event #3

Pitfalls

many streams
missing events detected when new event arrives
lost last event
~~eventually~~ never consistent
drifting state

Reconstitution 💦

Like syncing state, but generic

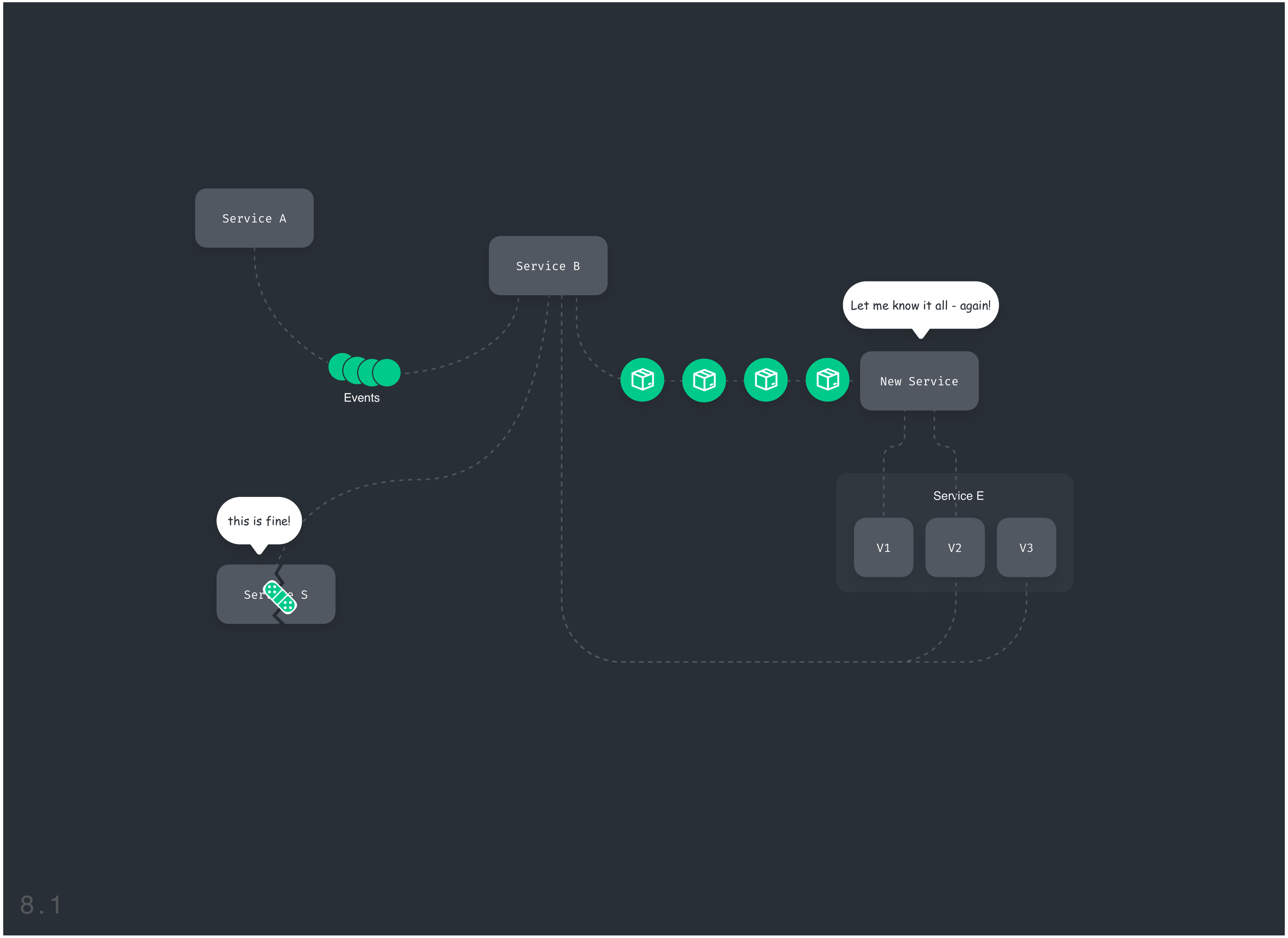
Walk over all known event streams

Ask upstream service for events since "internal cursor"

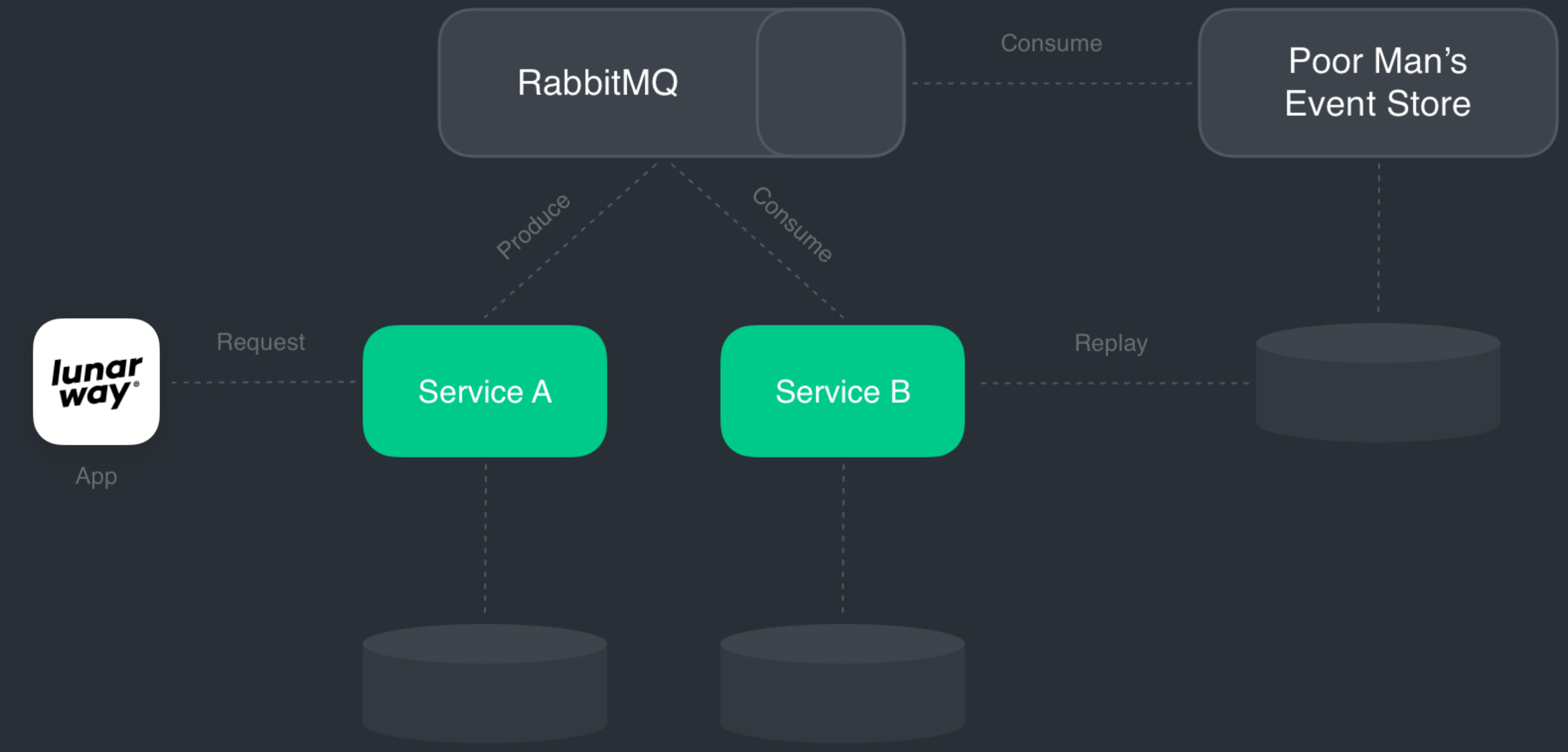
Either

we receive nothing	we receive events
✅ up to date	✅ get up to date

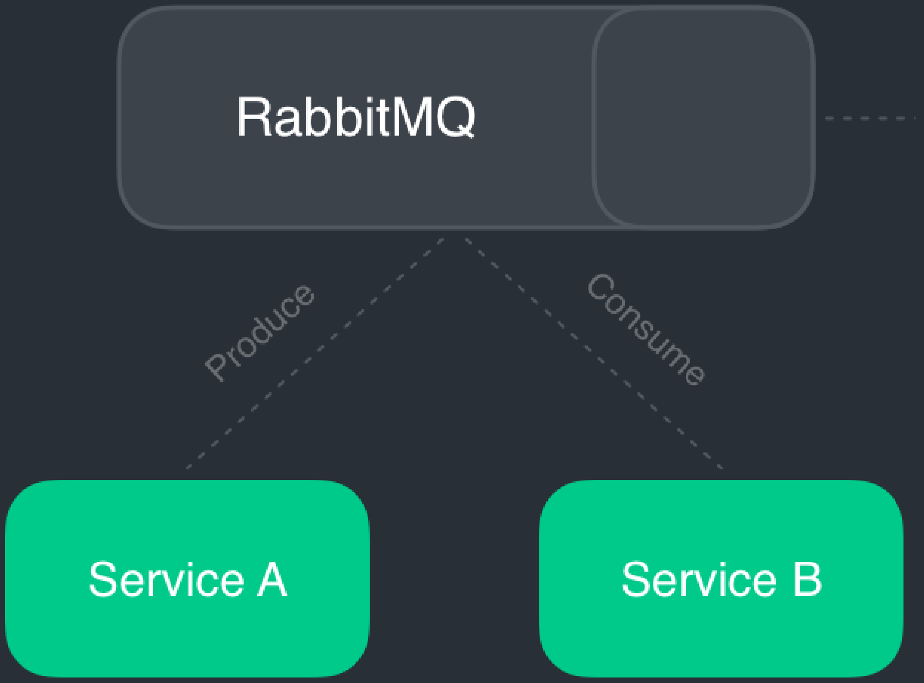
Problem solved. Thanks event sourcing 💙


Service A
Service B
Events
Let me know it all - again!
New Service
Service E
V1
V2
V3
this is fine!
Ser S

deprecate existing data
add new data
modify existing data

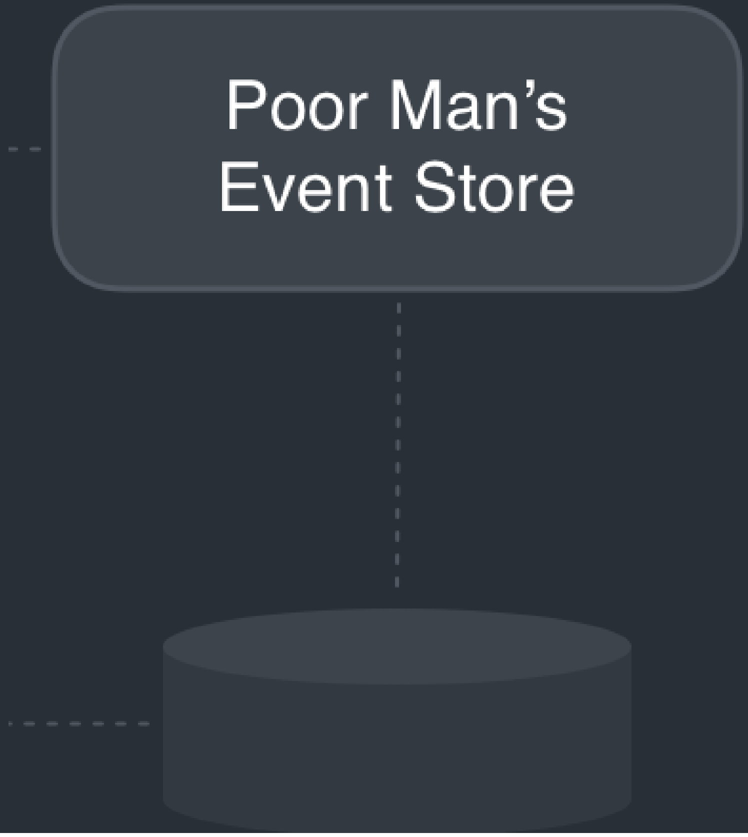


RabbitMQ
Consume
Poor Man's Event Store
Produce
Consume
lunar way
Request
Service A
Service B
Replay
App



hard coupling via events
no versioning
only additive changes
coordinated migrations

events detached from producer
events cannot be updated
consumers must adapt



🤔 Evolution as an ordinary, daily thing?

"If it hurts, do it more often"

producer owns events
ability to map events to new models
controlled, step-wise migrations

Event sourcing as a solution

"Ordered, persisted event streams with easy re-delivery"

+

integration events on the outside
versioning of projections
walkers for hydration

about integration events...

internal events can
be used for multiple integration event streams

🤯

move to new integration events

through a non-blocking migration

Producer

add new projection 🐣

...

...

...

...

🗑️ old projection

Consumers

...

extend consumer 1

extend consumer 2

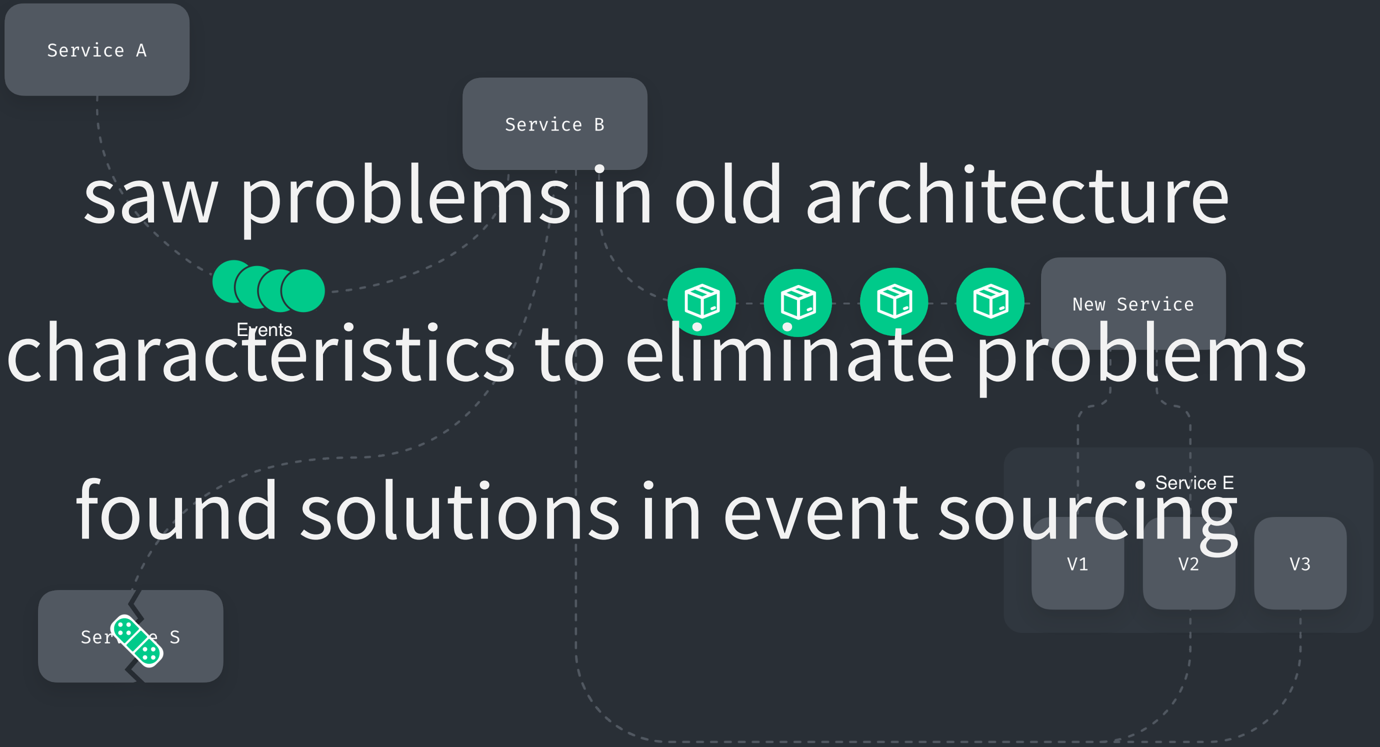
🗑️ old projection in consumer 2

🗑️ old projection in consumer 1

...

~~mutual agreements~~

~~strict coordination~~



saw problems in old architecture

characteristics to eliminate problems

found solutions in event sourcing

Things to take into account

not an off the shelf product
developing a framework is costly
introducing a new service design paradigm is hard
solid patterns, easy to improve

A look from above 🦅

A pattern emerges

Pain 😖 ➡ Normal 😊

Focus on business domain

and iterating... over & over

If It Hurts, Do It More Frequently, and Bring the Pain Forward

- Jez Humble

hurt to validate state 😓
bootstrapping was hurtful 😬
change is cumbersome & hard 🥶

check state all the time 🔍
redelivery as a daily pratice 😎
change as an enjoyable thing 😊

We wont lie

It's not an easy solution, but
these
characteristics & guarantees
lead to
reliable
improvable
microservices

questions? 🤓

thanks 👋

👈 Thomas Bøgh Fangel @tbfangel

Emil Krog Ingerslev @emilingerslev 👉