OMG a Request …

and how the SVED architectural pattern might help

Frank Baier, Integration Architect
Baloise Insurance (CH)

Basel, 17. Oktober 2019

Company Description, Organisation, who am I?

Baloise (CH) / Group

- Staff 3'700/7.300
- listed on SIX Swiss Exchange

Technologies: we collect them all… but strategic: Java

Agility is our DNA

Who am I:

- Started Java with 1.0.2 in 1996, developing software architect
- Like technologies if they solve my issues – or are just fancy
- Enjoy spending time with family&friends playing cards, doing sports, following soccer and the financial market. And if you've got a good beer: get me one, please.

Why we started

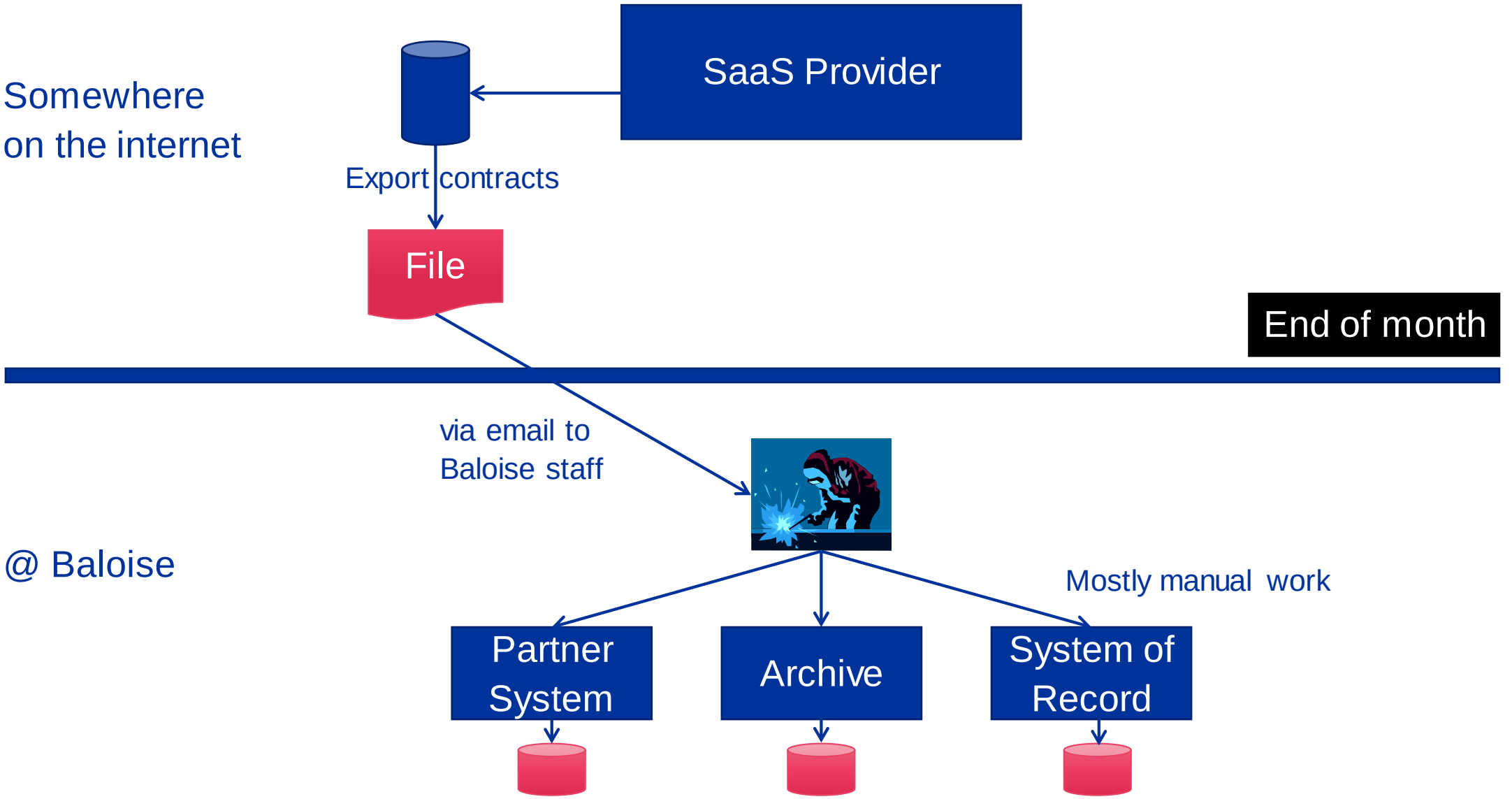
Somewhere
on the internet

Any time during the month

@ Baloise

No contract data in our systems

Why we started



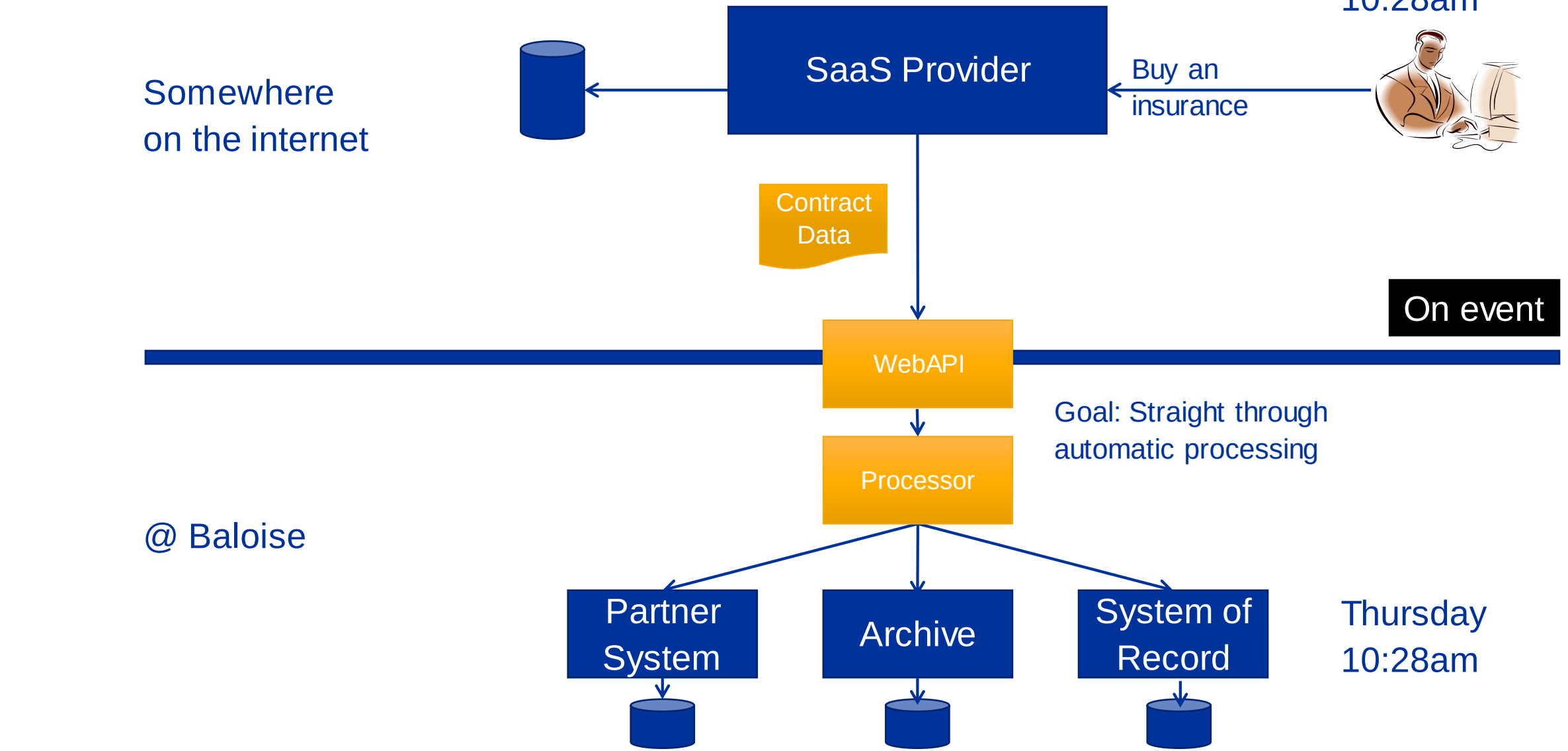
Challenges

- Manual work
  - Expensive
  - Error prone
- Once a month is too late
  - Regulatory restrictions
  - Claims handling gets more complex
  - Commission payment
  - Lack of being able to respond to customer questions
- Increasing volume leads to huge backlog
- Security
- Format issues, e.g. Date

Situation became quickly Inacceptable - currently 2.000 new contracts per month

Solution: Lets build a WebAPI

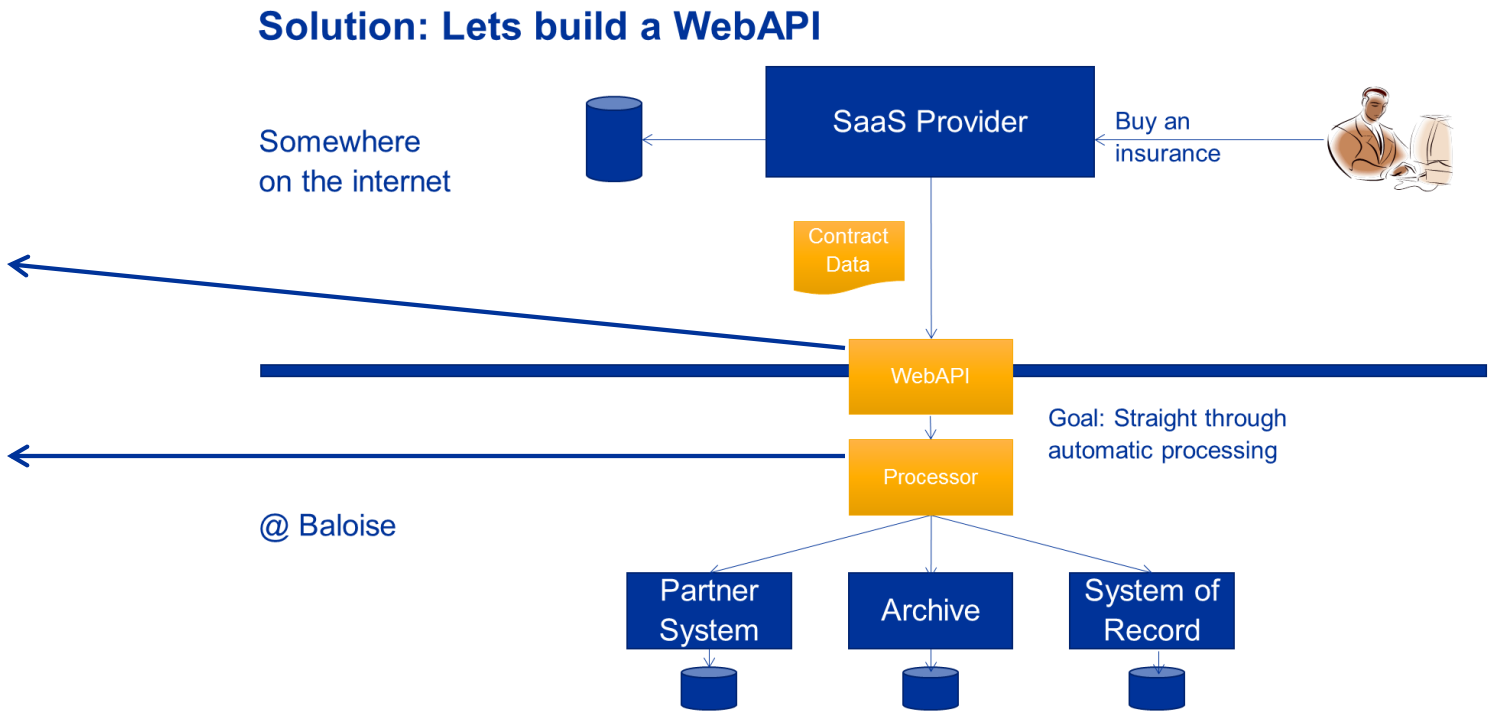
Thursday
10:28am



Two new components – two challenges

How to design, which standard

How to process exactly?



Solution: Lets build a WebAPI - But how?

Security

Technology

Design

Spec Standard

WebAPI

Processor

How to process?

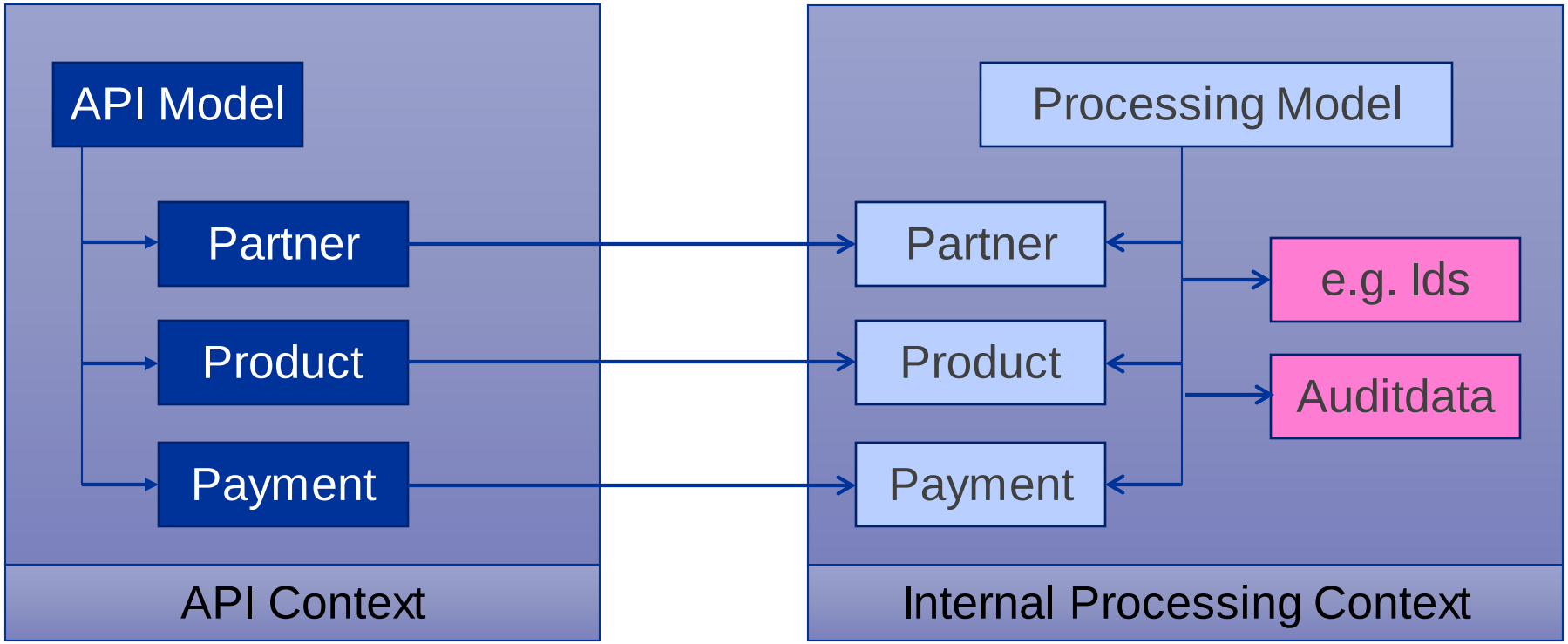
Comparing requirements

Requirements	
Consumer	Provider
• Standard technology • Resilience • Early feedback • Idempotence • Secure Transmission	• Straight through processing • Support operations • High data quality • Reliable processing • Handle incomplete data • Monitoring/Alarming support • Scalability – higher volume • Scalability – more SaaS Provider • Secure Transmission

Introducing the SVED architectural pattern

Assumption		Requirements
Errors will occur Reconciliation and analysis required	Store	Support operations
Consumer will potentially provide invalid data	Validate	High data quality
API Consumer has a different bounded context	Enhance	deal with different bounded contexts
Several loosely coupled downstream systems	Distribute	Reliable processing

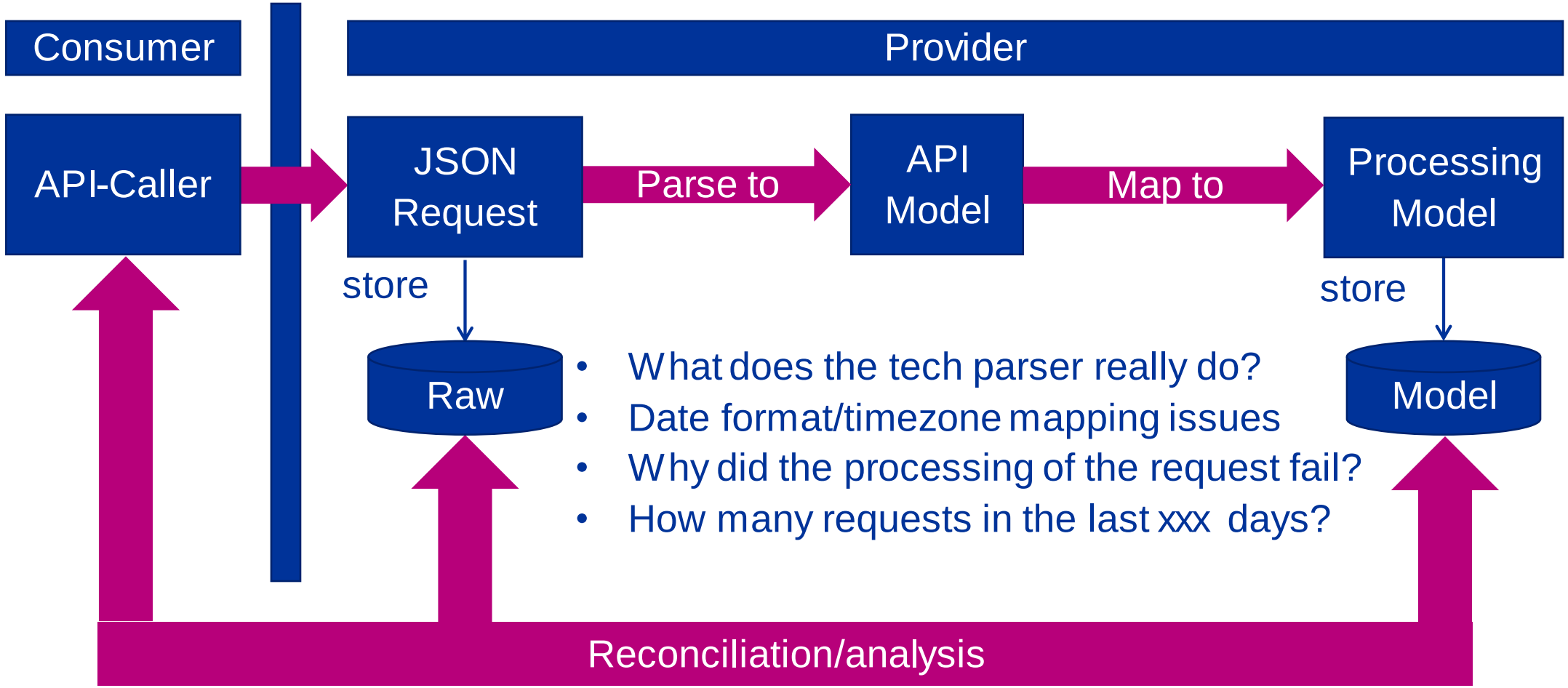
A word about bounded context



Additional data
Different (optimized) structure

Store

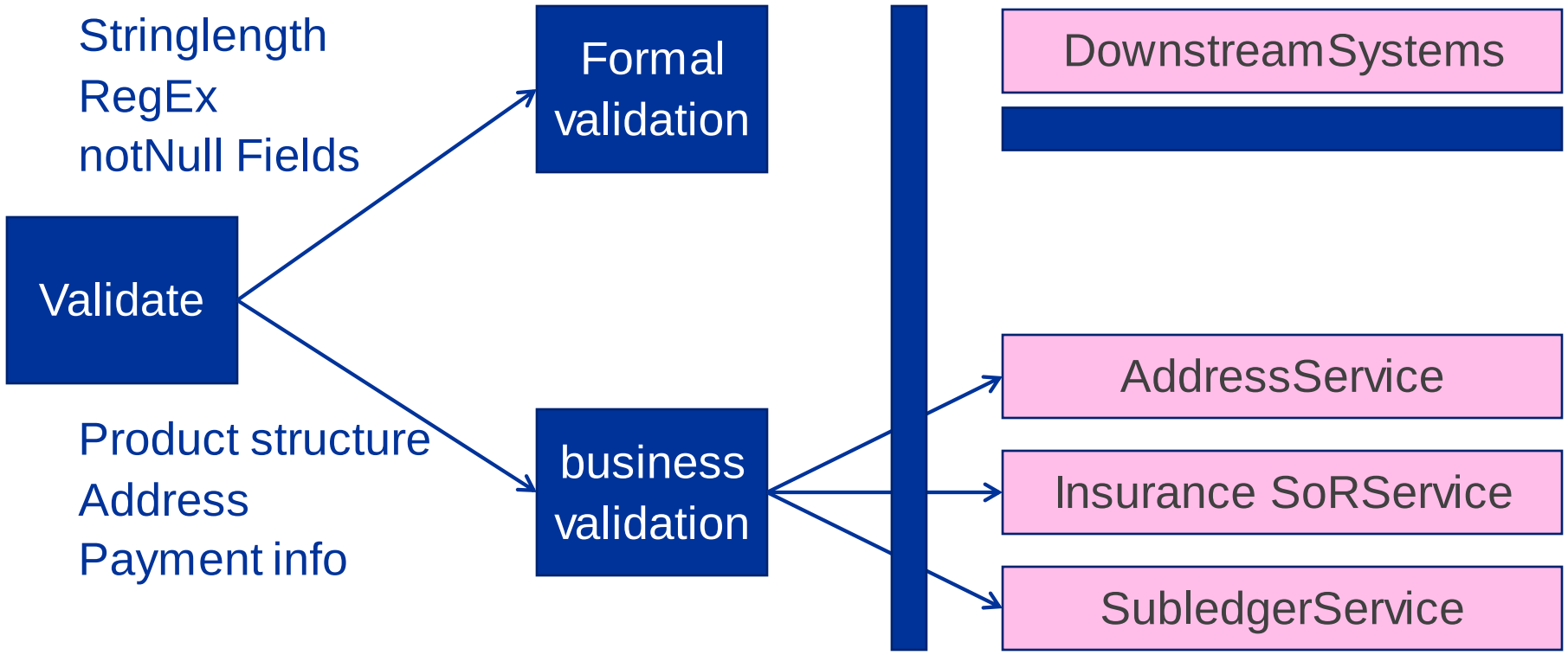
Support operations - Errors will occur - reconciliation and analysis required



- What does the tech parser really do?
- Date format/timezone mapping issues
- Why did the processing of the request fail?
- How many requests in the last xxx days?

Validate

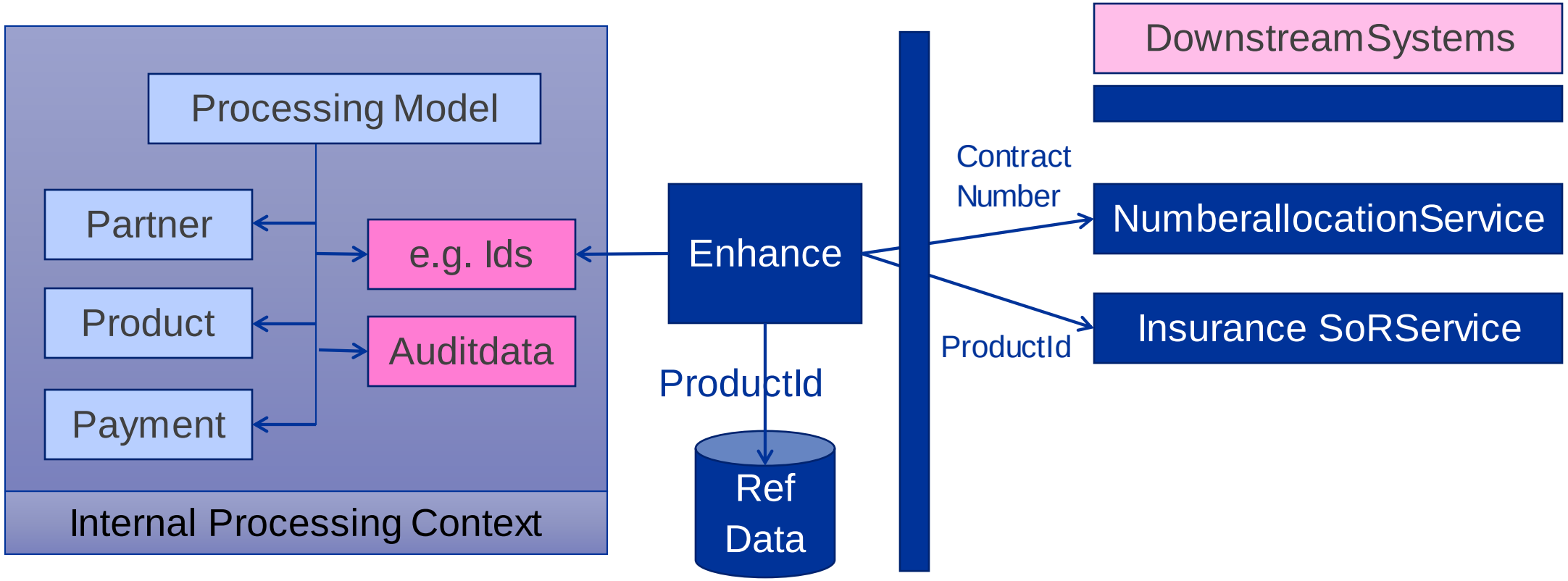
High data quality - Consumer will potentially provide invalid data



API Consumer hasn't to bother about further processing → return result, proceed async

Enhance

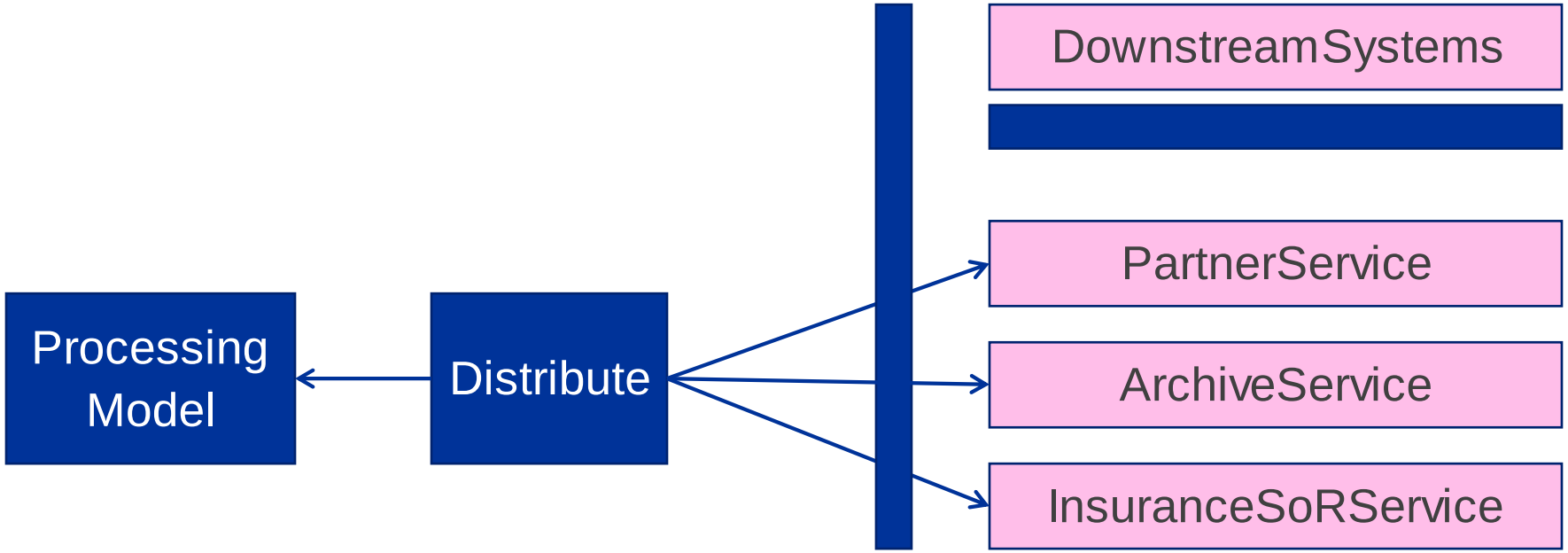
API Consumer has a different bounded context – deal with different data requirements



Enhance provides better semantics and avoids redundancy

Distribute

Several loosely coupled downstream systems - Reliable processing



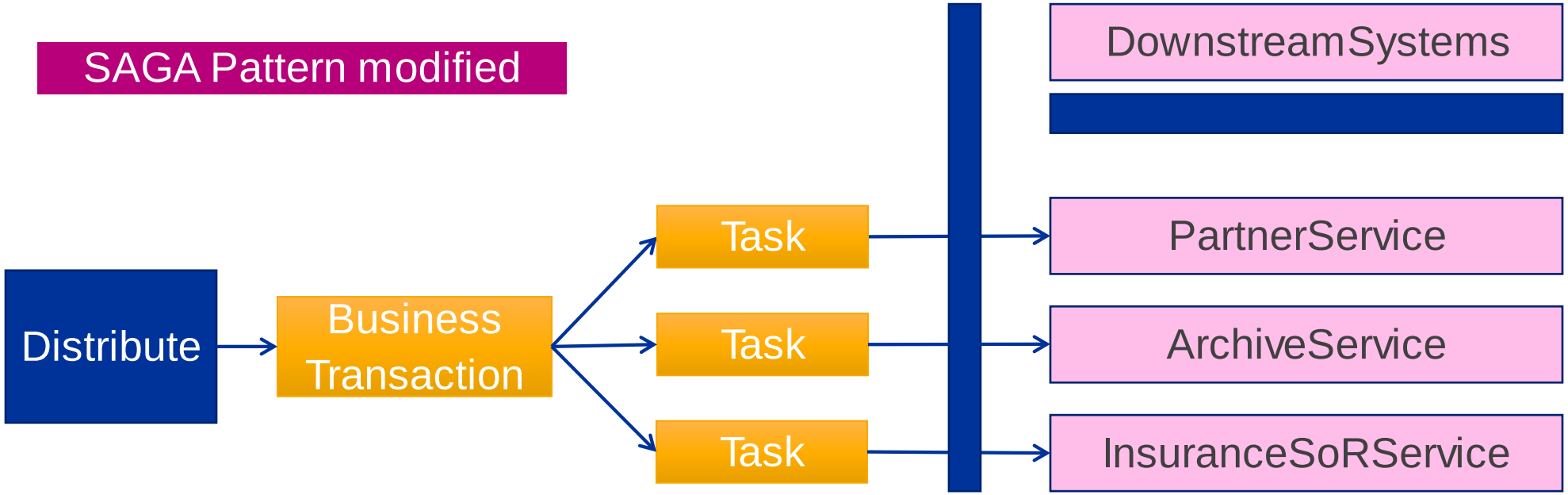
Distribute

Several loosely coupled downstream systems - Reliable processing

Requirements	
mapping	downstream systems have different bounded contexts
extensibility	a further downstream system should easily be integratable
sequence	downstream systems might require to be called in a certain sequence
reliability	every downstream system must process its request errorless
control	Which downstream system has processed already and how?

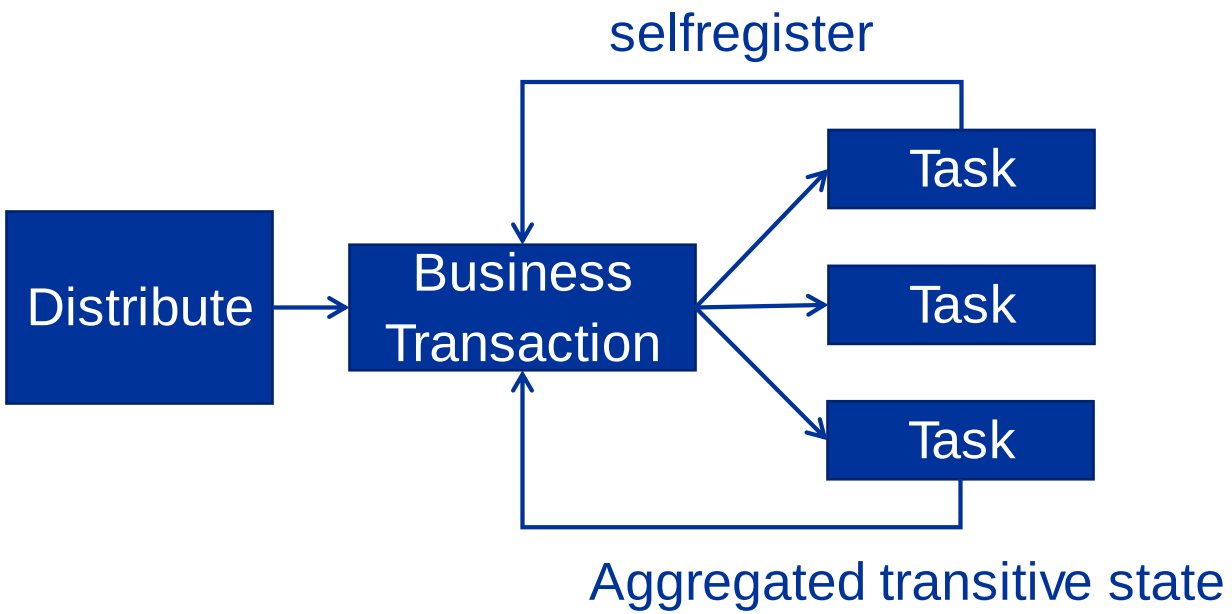
Distribute

Several loosely coupled downstream systems - Reliable processing



Distribute – BusinessTransaction

Several loosely coupled downstream systems - Reliable processing



Distribute – BusinessTransaction

Several loosely coupled downstream systems - Reliable processing

Aggregated transitive status			
Task	**BTX #3**	**BTX #2**	**BTX #1**
createPartner	NEW	DONE	DONE
saveContract	NEW	IN ERROR	DONE
archiveDocument	NEW	NEW	DONE
Aggr. BTX State	NEW	IN ERROR	DONE

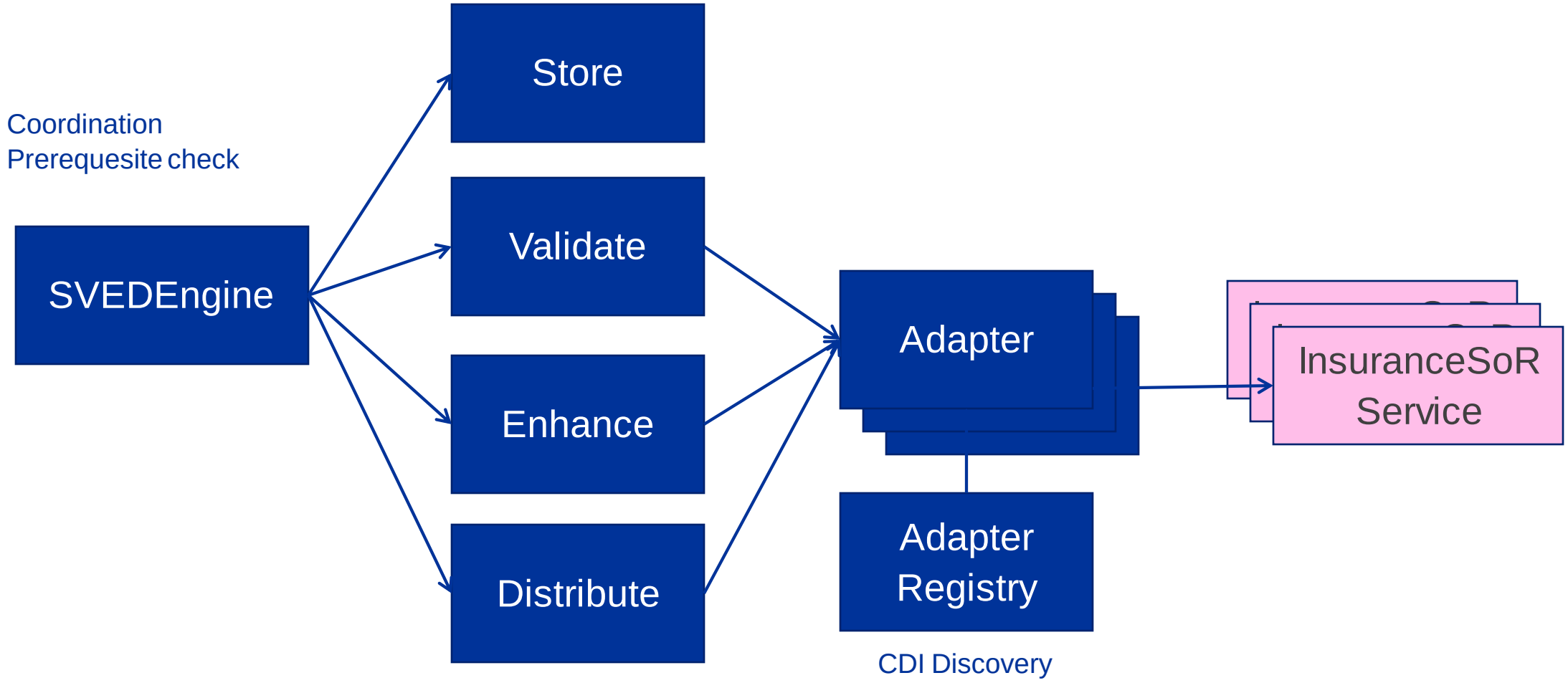
Implementation

Appliced best practices

JavaEE-based	
Idempotence	process repeated calls without creating duplicates → Explicit idempotency vs. implicit idempotency
JSON storage	use Servlet-Filters to store JSON before JSON-Java mapping kicks in – it might fail!
Early feedback	use Java EE @asychnronous annotation for splitting up processing
encapsulation	downstream systems encapsulated by one adapter each
central coordination	Implement a process engine

Implementation

Process Engine and Downstream System encapsulation



Implementation

Alternatives

- Camunda and other BPMN Tools – way to heavyweight compared to lightweight solution
- Apache Camel
  - potential solution but still comes with some complexity, especially in testing
  - Saga Pattern vs. Homegrown BusinessTransaction solution
    - No rollback desired – forward strategy
- Spring – Spring boot or any other fancy microservice framework
  - Possible as well – but we're a Java EE shop

Reality check

promises fulfilled?

Requirements		But …
Support operations	✓	Store in two places required
High data quality	✓	Validation has different flavours
deal with different bounded contexts	✓	Enhance is not easy to explain
Reliable processing	✓	No "but" found yet

Context

Does it apply to my challenges?

- Processing of request of high relevance?
  - Weather forecast request for private person vs. for a flight
  - Stock quotes for an investment game vs. institutional investors
- Several downstream systems involved for validation or processing of the request?
  - Complexity due to coordination of different systems
  - E.g. Online shop checkout: check stock, execute payment, send email, finalize checkout
- Different bounded contexts?
  - Consumer knowledge differs from internal requirements

Lessons learned

- SVED-Pattern implementation works well
  - seven products are currently processed
  - Other types besides insurance contracts will be processed – cancellation, payments. Analysis shows business agnosticity of the pattern
- UI-Dashboard required for administration
- JavaEE still works fine

OpenAPI

- Smartbear donated Swagger Spec to the OpenAPI inititiave (part of Linux foundation)
- Swagger is a set of tools – OpenAPI is the specification the tools rely on
- Why not rename Swagger tools too? → Vendor neutrality
- Backport from OpenAPI to Swagger is available
- API first - Java-based

To open source or not?

Pattern Documentation https://tinyurl.com/yydpoel3

SourceCode

If you're interested, get in touch and let's have a beer

Questions?

Contact:
frank.baier@baloise.ch
@BaierFrank