Architectural Inter-Microservice Integration

An Overview

by Georg Schwarz

7. Treffen des Arbeitskreises MSDO, 9th March 2020

Left or Right?

Photo by Rafif Prawira on Unsplash

Microservice Integration

What to integrate…?

- **Microservices with each other?**
- With an external system?
- With infrastructure as Kubernetes?

Focusing on which aspect...?

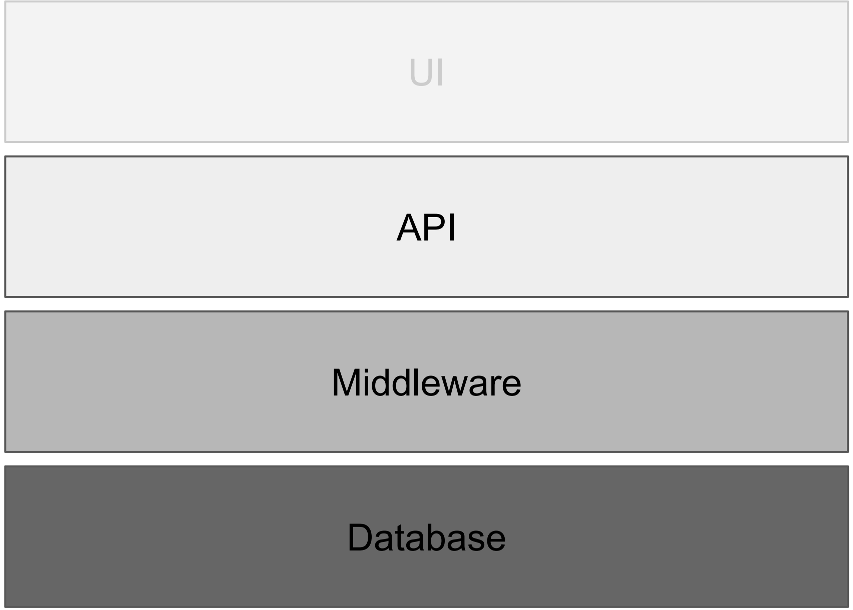
- **Architectural design?**
- Monitoring?
- Security?
- Communication between teams?
- Evolution over time?

Architectural Inter-Microservice Integration

Microservices
vs.
Enterprise Information Integration

Schwarz, Georg-Daniel, and Dirk Riehle. "What Microservices Can Learn From Enterprise Information Integration." *Proceedings of the 53rd Hawaii International Conference on System Sciences*. 2020.

Architectural Levels



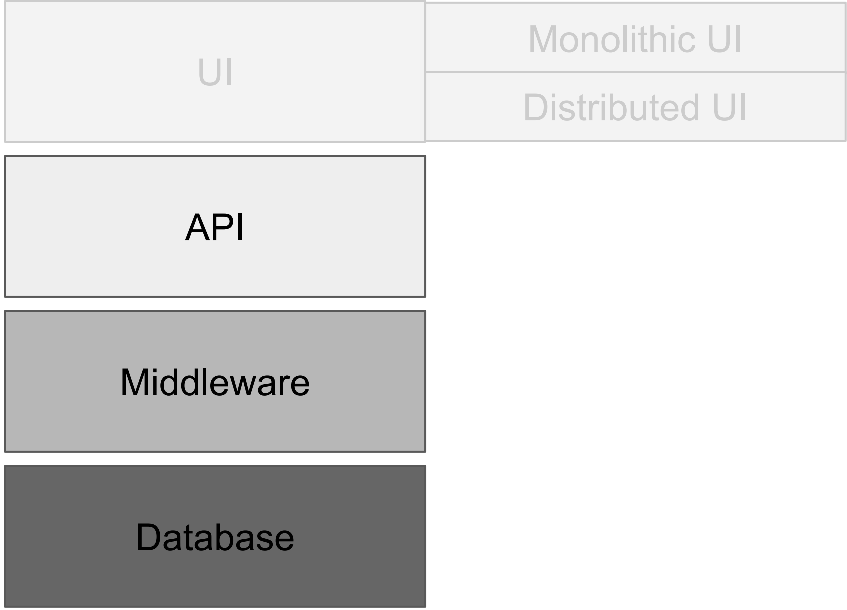
Enterprise Information Integration:

We can integrate on every architectural level of a system [1]

[1] P. Ziegler and K. R. Dittrich, “Three decades of data integration — all problems solved?,” in Building the Information Society, pp. 3–12, Springer, 2004.

Architectural Levels

Microservices:



UI

Monolithic UI ✓

Distributed UI ✓

Omitted in this talk

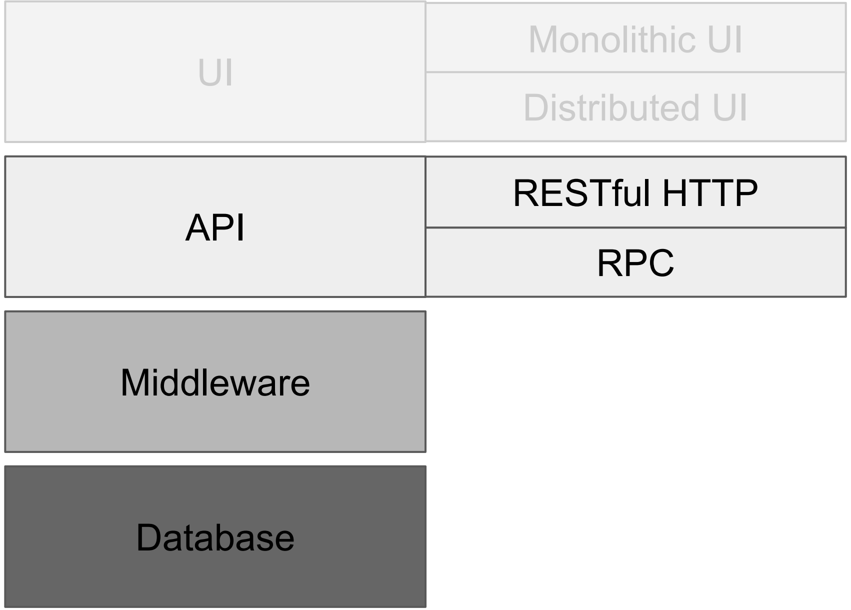
API

Middleware

Database

Architectural Levels

Microservices:



UI

Monolithic UI ✓

Distributed UI ✓

Omitted in this talk

API

RESTful HTTP ✓

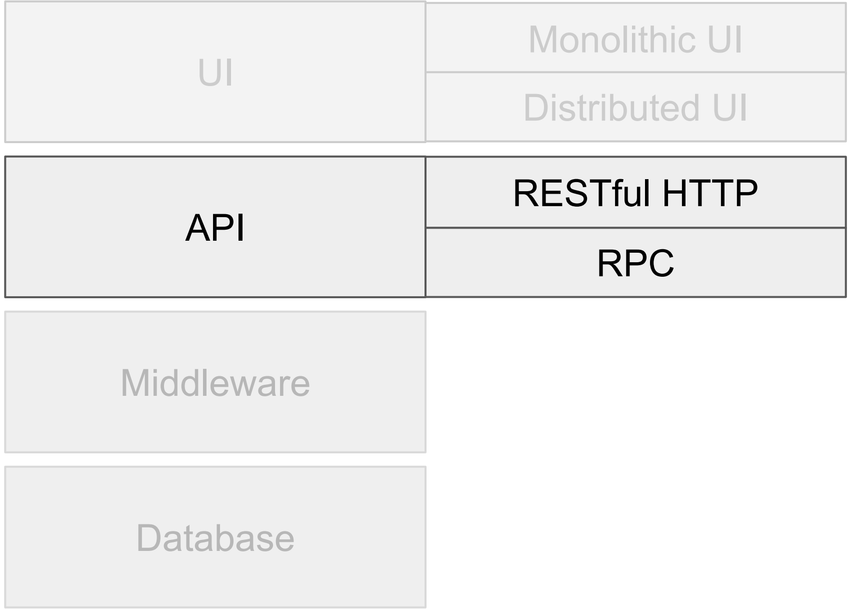
RPC ✓

RESTful HTTP over RPC

Middleware

Database

Architectural Levels - RESTful API over RPC

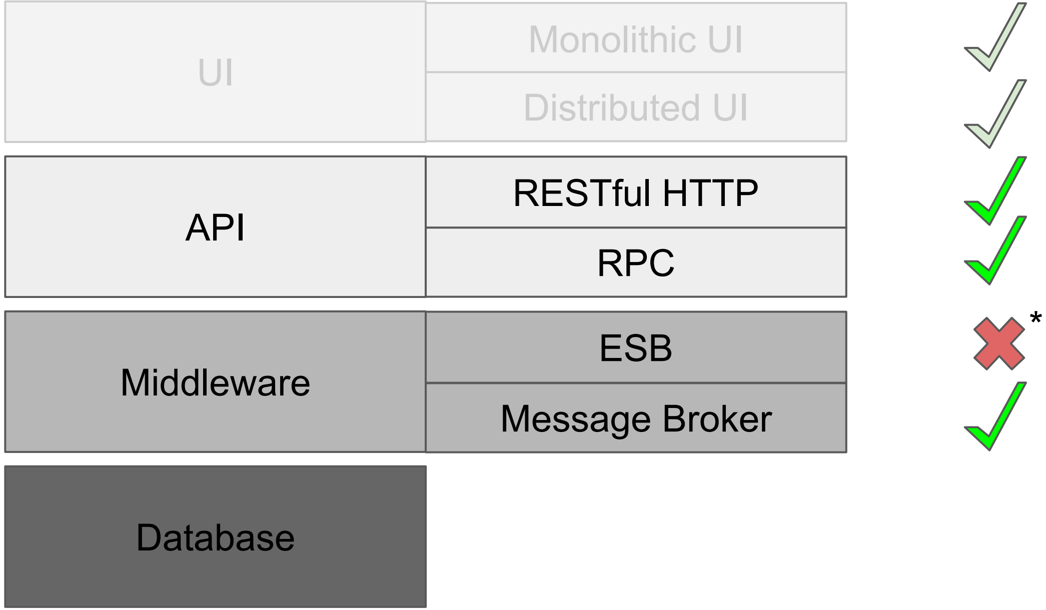


RESTful HTTP over RPC

- HTTP well-known, lots of tools
  - Security
  - Routing
  - Load balancing
  - Caching
- Easier to version - no stub generation
- Technology-independent
- No network transparency

Architectural Levels

Microservices:



UI — Monolithic UI ✓

UI — Distributed UI ✓

API — RESTful HTTP ✓

API — RPC ✓

Middleware — ESB ✗*

Middleware — Message Broker ✓

Database

Omitted in this talk

RESTful HTTP over RPC

Keep the middleware as dumb as possible

* due to potential misuse by pushing business logic into the ESB

Architectural Levels

		Microservices:	
UI	Monolithic UI	✓	*Omitted in this talk*
	Distributed UI	✓	
API	RESTful HTTP	✓	*RESTful HTTP over RPC*
	RPC	✓	
Middleware	ESB	✗	*Keep the middleware as dumb as possible*
	Message Broker	✓	
Database	Indirect Access	?	*Are these even microservices anymore?*
	Direct Access	✗	

Why not at Database Level?

➕

- Simple
- Fast to get started
- Database is fast at joining data

➖

- Expose implementation details
- Break consumers by internal changes
- Tie consumer to DB technology
- Distribute logic to manipulate data to multiple services

=> No independent deployability

Conclusion:
DON’T DIRECTLY ACCESS THE DATA OF OTHERS MICROSERVICES

A Closer Look at Architectural Inter-Microservice Integration *

⚠ Work in Progress ⚠

Input Wanted

* based on most popular gray literature

Goals of Integration

- Independent Deployability
  - Decoupling
  - Interface Versioning
- Scalability (includes sufficient performance)

Non-negotiable

- System extensibility
- Technology Heterogeneity
- System Simplicity
  - Understandable Workflows
  - Failure Handling
  - Complexity should be justified!

Trade-offs based on strategy

Why Do We Integrate?

Cross-cutting features need to

- Trigger distributed behavior
  - Control Flow
- Access data from other microservices
  - Data Flow

(Unvalidated) Theory:
We can combine control and data flow integration approaches to build our architectural inter-microservice integration strategy. *

* Discussion: probably one of both aspects is dominating in system design (control flow follows data flow vs. vice versa)

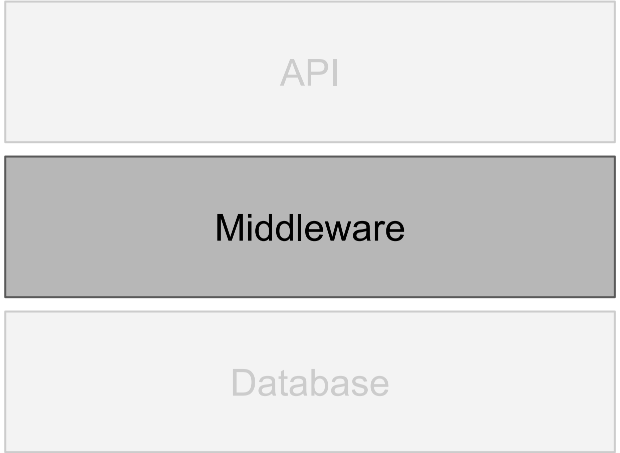
Data Flow Integration

Data Flow Integration

When to get the data from other microservice?

- Get data when we need it
  - Work with references and fetch **on-demand**
  - Get only the data that we need and not more, still can apply caching for optimization
  - Can get “too new” data
- Get data beforehand and cache it
  - Data **replication**
  - Eventual consistency: work on potentially outdated data

Data Flow Integration - Middleware Level

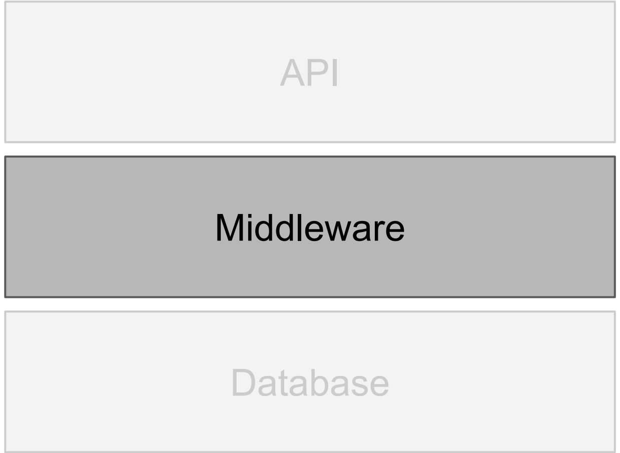


Replicate data via message broker

Event-Driven Architecture

- Listen to events and build up own replication of data (in own format)
- Whole event history necessary
  - Event sourcing
- Or combination with API-level data flow integration
  - Similar to snapshot & delta

Data Flow Integration - Middleware Level


API

Middleware

Database

Replicate data via message broker

+

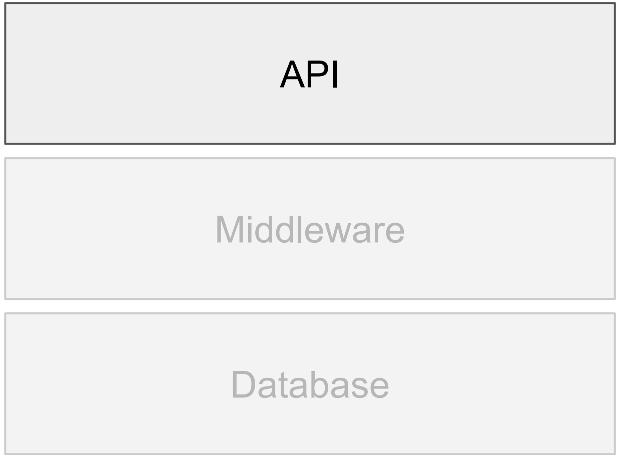
- Decoupling by events
- Keep only data that is necessary in best suiting format
- Easy to add new services
- Use features of message brokers

−

- Harder to reason about async architecture
- Event versioning required
- Message broker as additional dependency

Conclusion: complex but recommended

Data Flow Integration - API Level



Alternative on API level: Event Feeds

ATOM feeds over HTTP

+

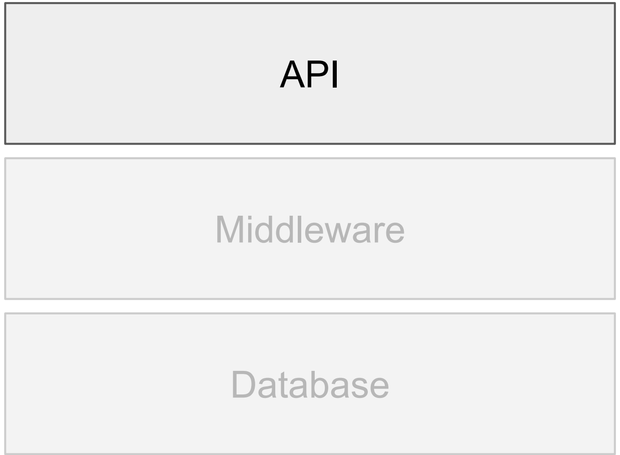
- Advantages from HTTP
  - Security
  - Scaling
  - ....
- Decentralized, no message broker as single point of failure

−

- Implement features of message broker ourselves
  - Polling schedule
  - Competing consumer pattern
  - ….

Conclusion: might be worth a look

Data Flow Integration - API Level



API

Middleware

Database

<u>Replicate</u> data via RESTful API calls

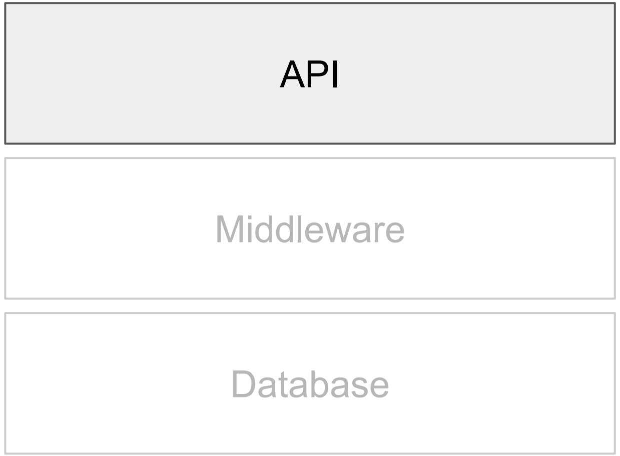
➕

- Simple to implement as consumer

➖

- APIs often not designed for replication
- Breaks down with larger data volumes

Data Flow Integration - API Level



Fetch data <u>on-demand</u> via RESTful API calls

+

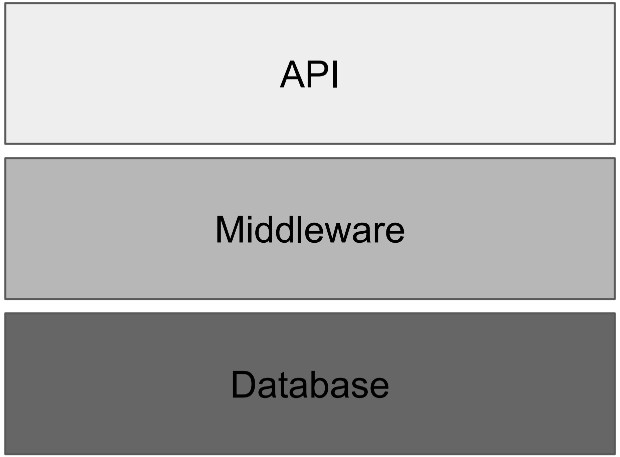
- Request/Response with HTTP is well-understood

−

- Multiple API calls might be necessary if multiple resources required (non-optimized interfaces)

Conclusion: sensible default choice

Data Flow Integration - API Level



Alternative on API level: Query-based Interfaces

E.g. **GraphQL**

- Could potentially solve the non-optimized interface problem
- Evolution instead of versioning?

Looking for interviewees that use query-based interfaces with microservices

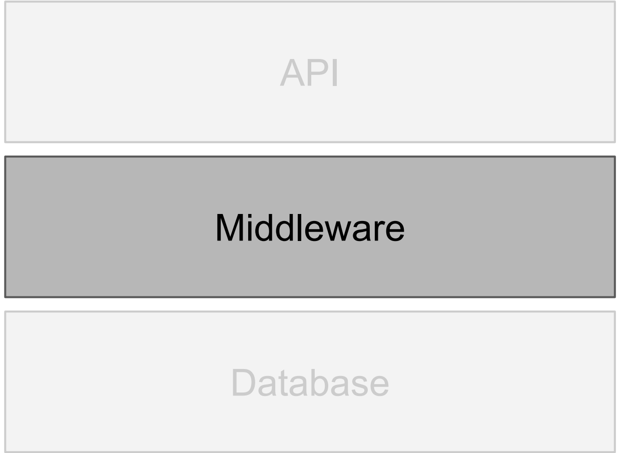
Control-Flow Integration

Control Flow Integration

Orchestration vs. Choreography

- Orchestration by a central brain
    - request/response to trigger other services
- Choreography forms system behavior by emergence of service (re)actions
    - (Async) events represent what happened in the system
        - Event-Driven Architecture

Control Flow Integration - Middleware Level



Choreography via message broker (events)

+

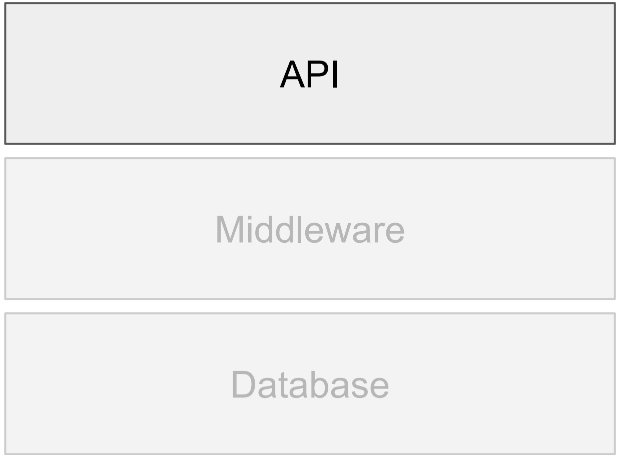
- Decoupling
- Easy to add new services
- Evenly distributed business logic (no central brain)

−

- Business process only implicitly reflected in our system
- Harder to reason about
- Complex failure handling

Conclusion: more complex but recommended

Control Flow Integration - API Level



Orchestration via RESTful API (Request/Response)

➕

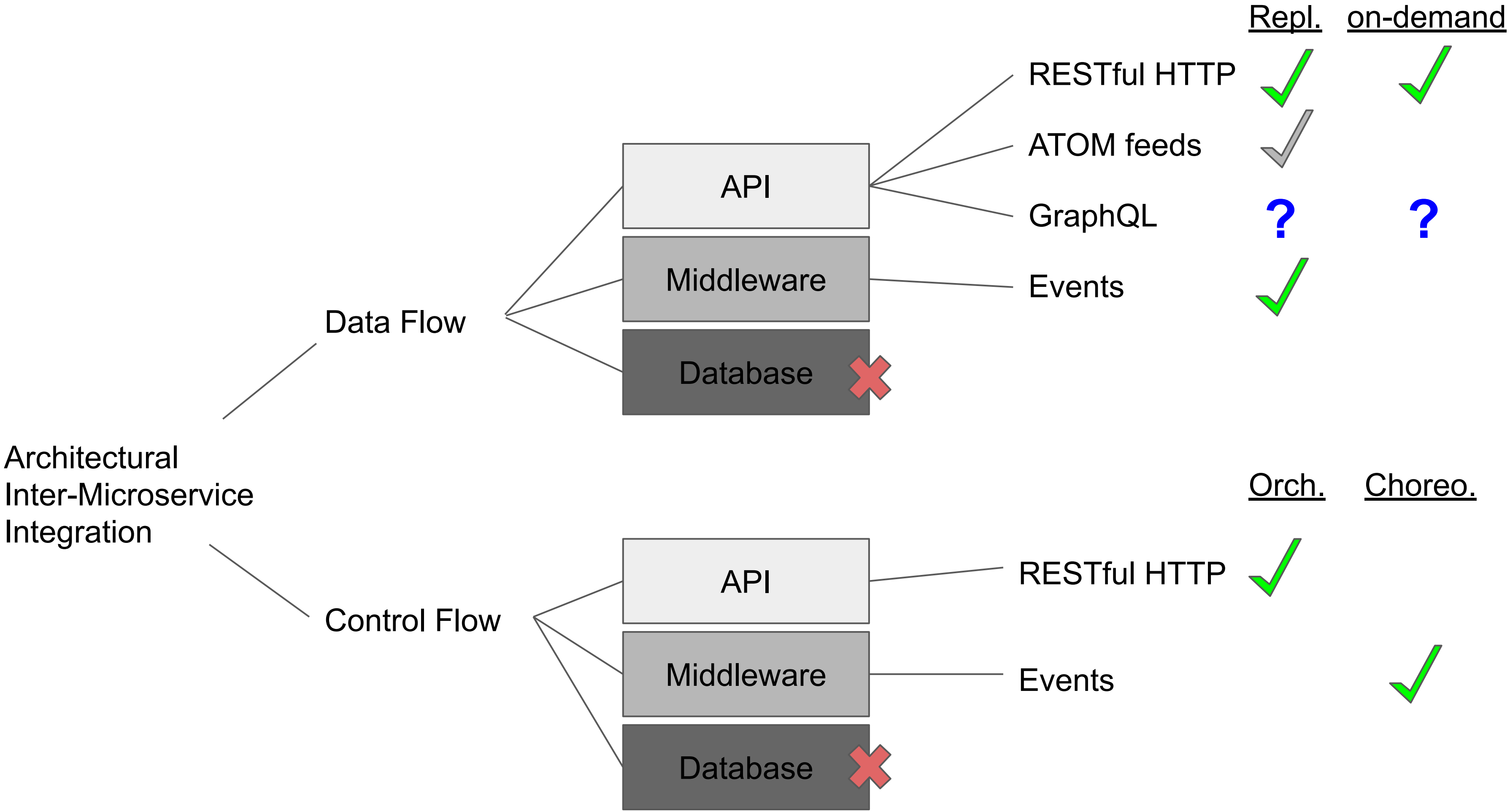
- Request/Response with HTTP is well-understood
- Easier failure handling
- Easier business process modeling

➖

- Resource-orientation might not fit to trigger behaviour
- Danger to build central point for all business logic
- Higher cost of change

Conclusion: also recommended

Summary



Architectural
Inter-Microservice
Integration

- Data Flow
  - API
  - Middleware
  - Database ✗
- Control Flow
  - API
  - Middleware
  - Database ✗

	Repl.	on-demand
RESTful HTTP	✓	✓
ATOM feeds	✓	
GraphQL	?	?
Events	✓	

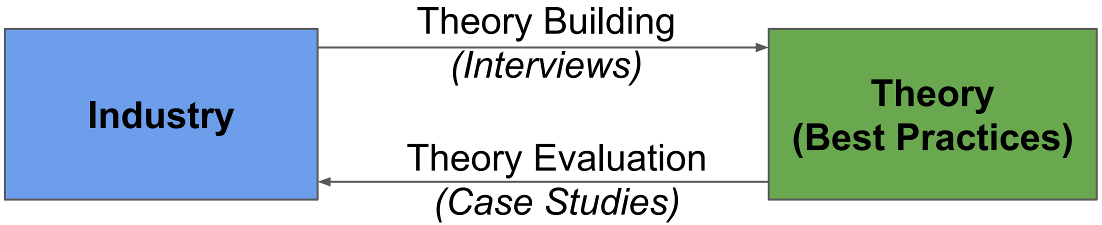
	Orch.	Choreo.
RESTful HTTP	✓	
Events		✓

* probably many more options and more dimensions to consider, e.g. gRPC, Service Meshes

Summary

- Microservices exclude some classical architectural integration strategies
- Still, there are a lot of different options with each pros and cons
  - Hard to get started with microservices!
- There are even more aspects in the area “Inter-Microservice Integration”
- It would be nice to have **patterns** or **best practices** to know which one to choose in which context

Summary



Industry

Theory Building

(Interviews)

Theory
(Best Practices)

Theory Evaluation

(Case Studies)

- It would be nice to have **patterns** or **best practices** to know which one to choose in which context

My Research Goal

Thank you!

Georg Schwarz

PhD student at Professorship for Open Source Software, Friedrich-Alexander University Erlangen-Nürnberg

georg.schwarz@fau.de

🐦 @schwargeo

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG