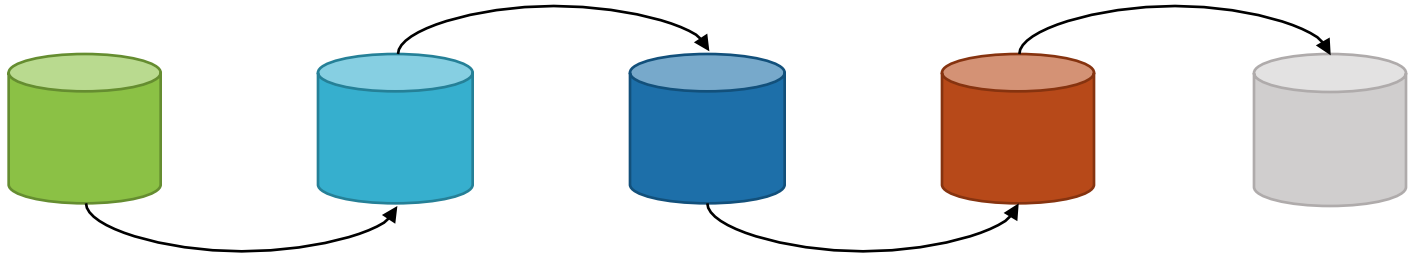
Stored Procedure *as a* Service (SPaaS)

@abhishektiwari https://abhishek-tiwari.com

- solutions manager apac @ Isentia
- leading media intelligence company
- data and product engineering
- data processing with NLP & search focus

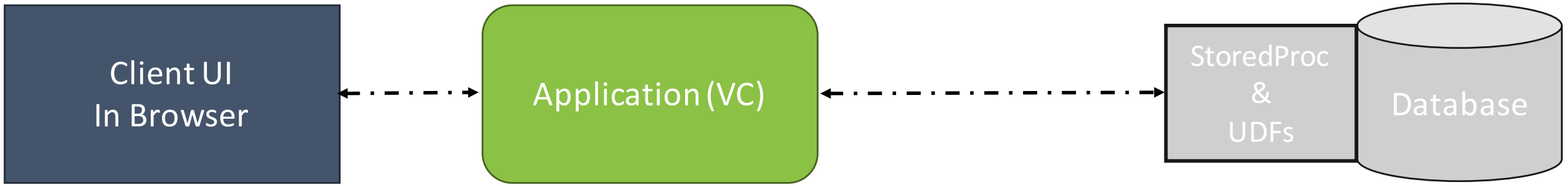
- a **data pipeline** built using legacy technologies
- database as queue, **ETL** using StoredProc
- trigger calling StoredProc calling trigger
- stateful, strongly coupled, monolith databases



- **thin** applications **fat** StoredProc **VLDB**
- more than **5000** StoredProc written over 10Y
- deliver **new** products/services in a short span
- need to move **fast** - continuous delivery

StoredProc

subroutine stored in the database data dictionary



StoredProc

good idea that quickly turned into massive technical debt

StoredProc

Moving data is harder than moving logic

StoredProc Concerns

Good

- improved performance
- security and access control

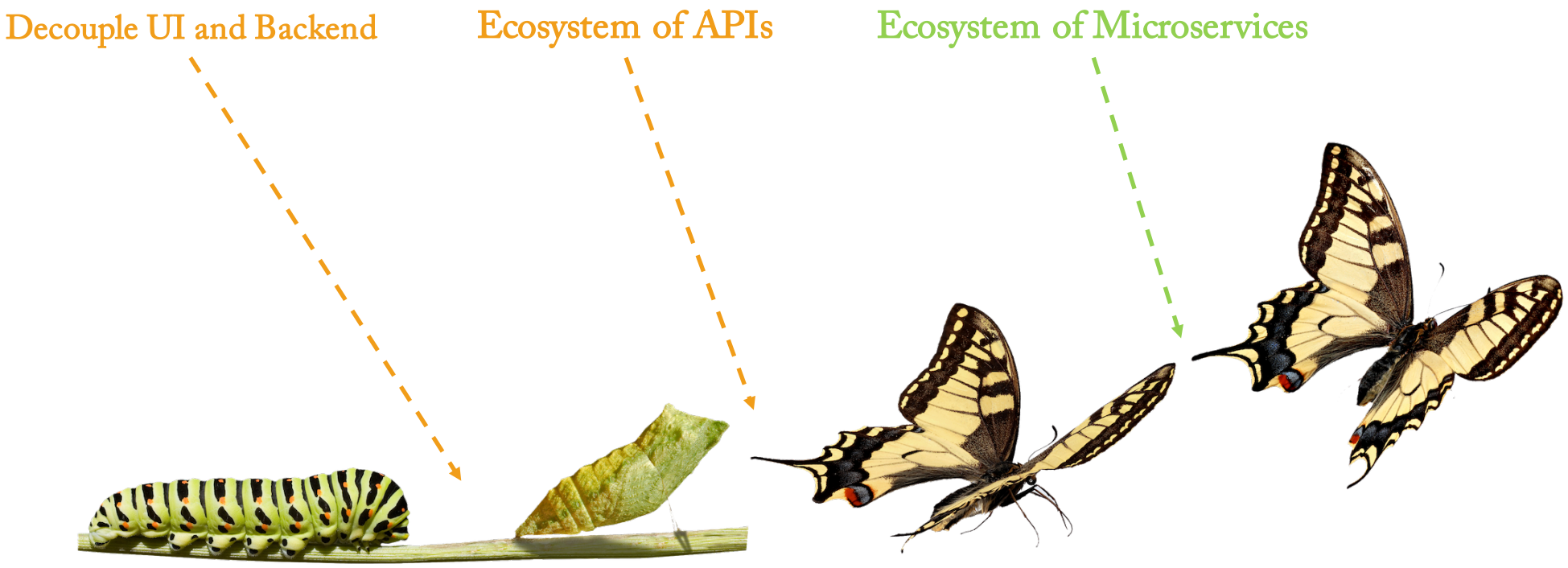
Bad

- highly procedural
- limited constructs
- can't pass objects
- vendor lock-in/specific
- maintainability

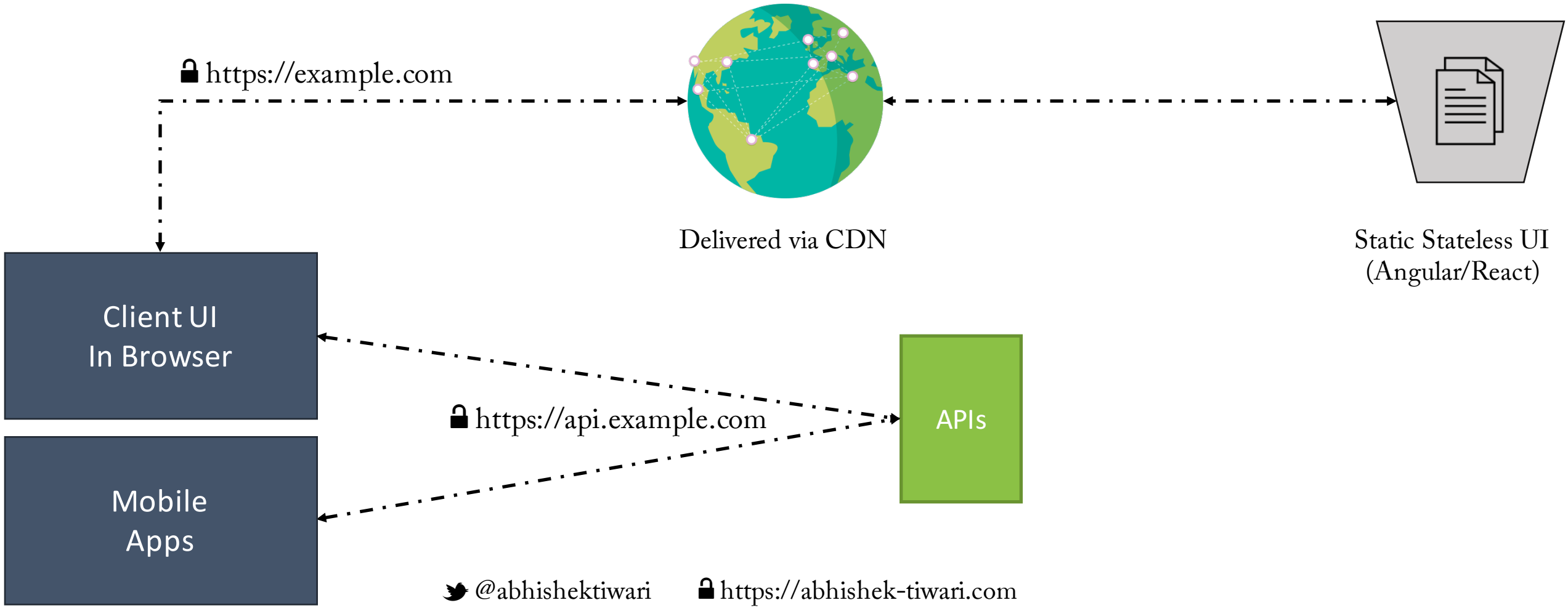
Ugly

- zillion parameters
- business logic
- versioning & testing
- debugging & logging
- continuous delivery
- cross-database StoredProc

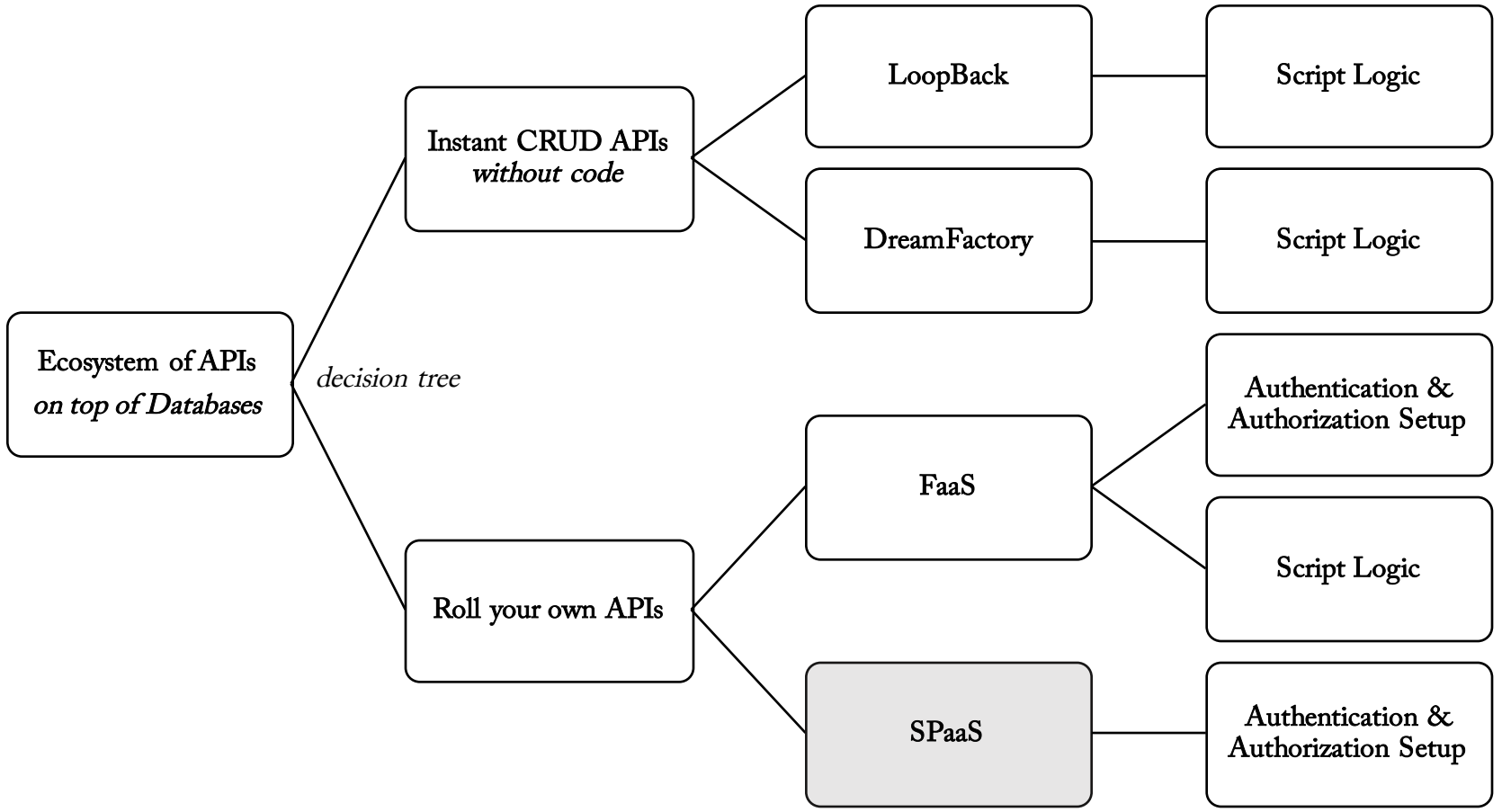
Phased Approach



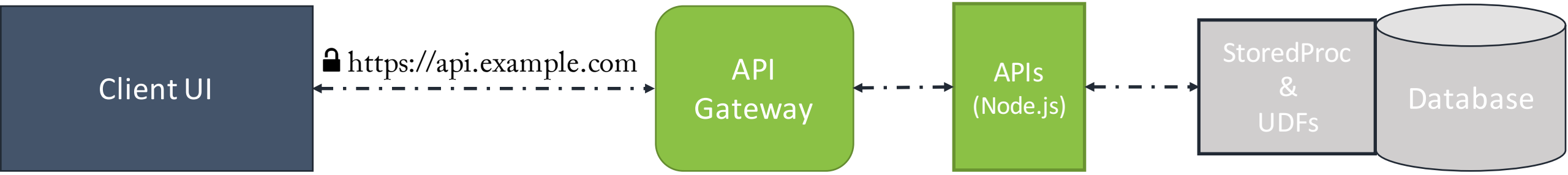
Separate UI layer from the backend layer



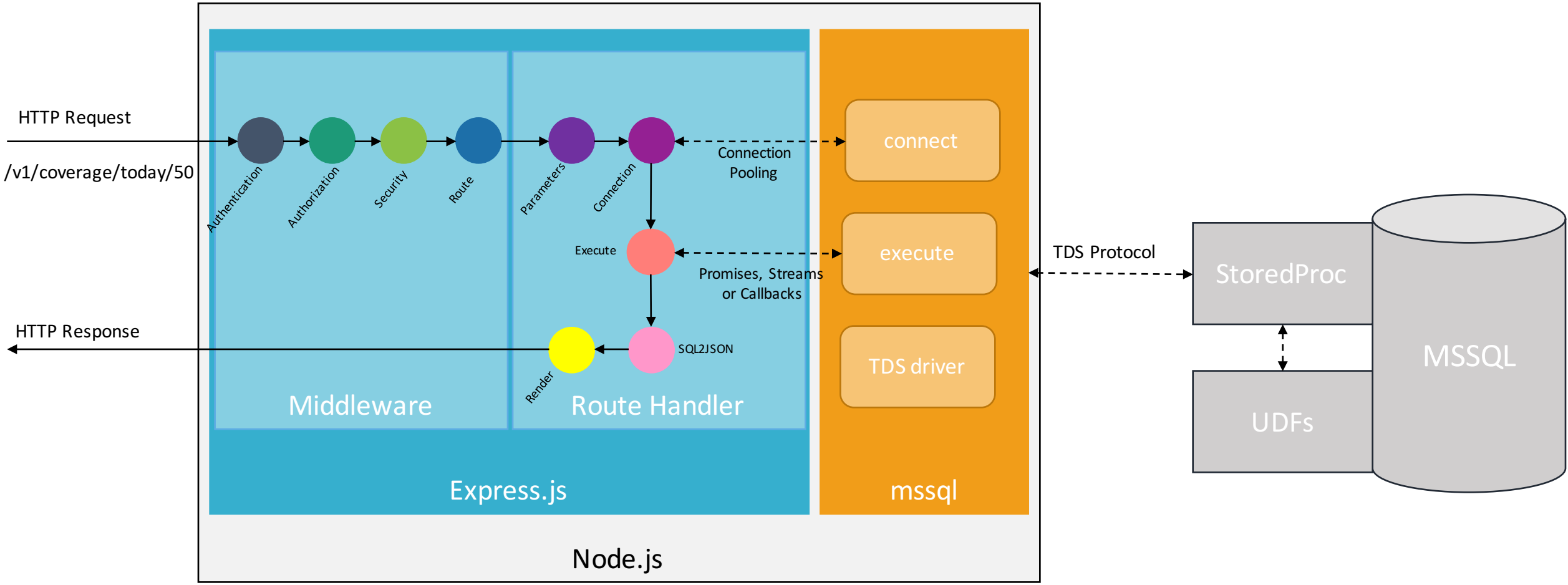
Convert legacy backend layer into ecosystem of APIs



SPaaS


Client UI
https://api.example.com
API Gateway
APIs (Node.js)
StoredProc & UDFs
Database

Wiring up StoredProc

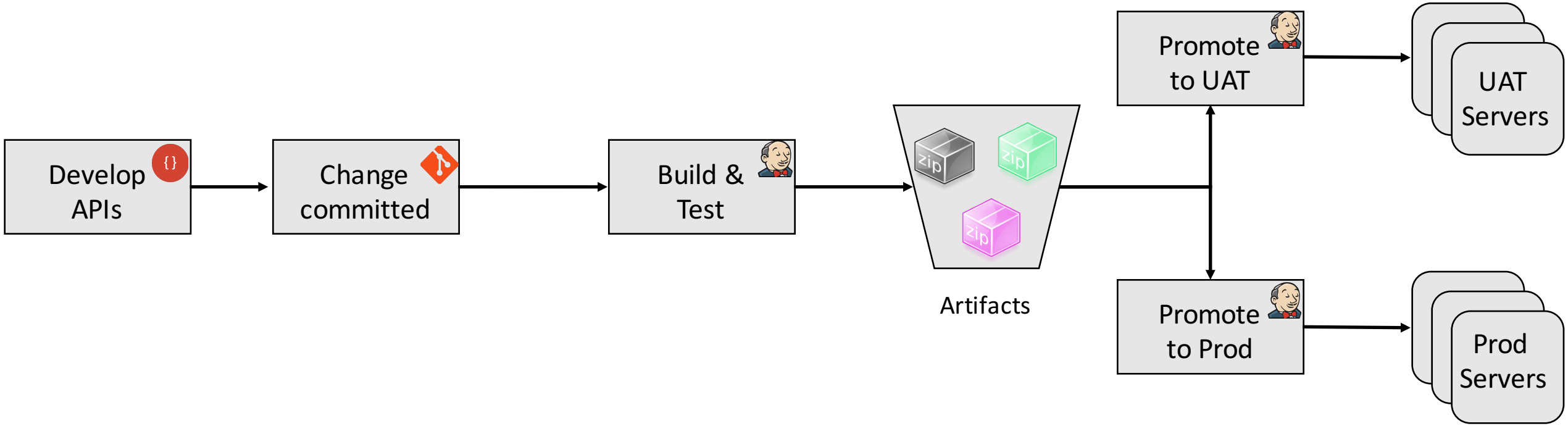


- **self-contained** piece of reusable functionality
- can be executed by **events** or **API endpoints**
- read and write to **database backend**
- can be written in multiple languages

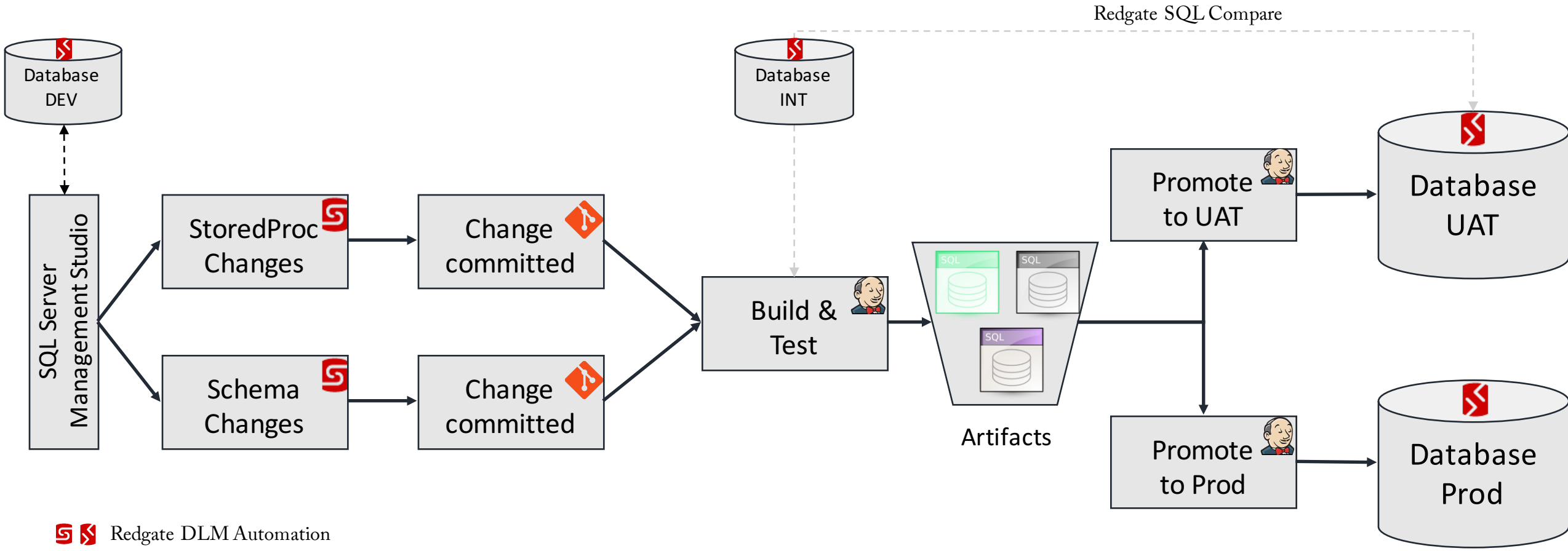
Similar Concerns

- **vendor lock-in** or specific implementations
- tooling for **continuous integration** & **delivery**
- monitoring, logging and **debugging**
- **testing**, discovery, security and latency

SPaaS API Delivery



SPaaS SQL Delivery

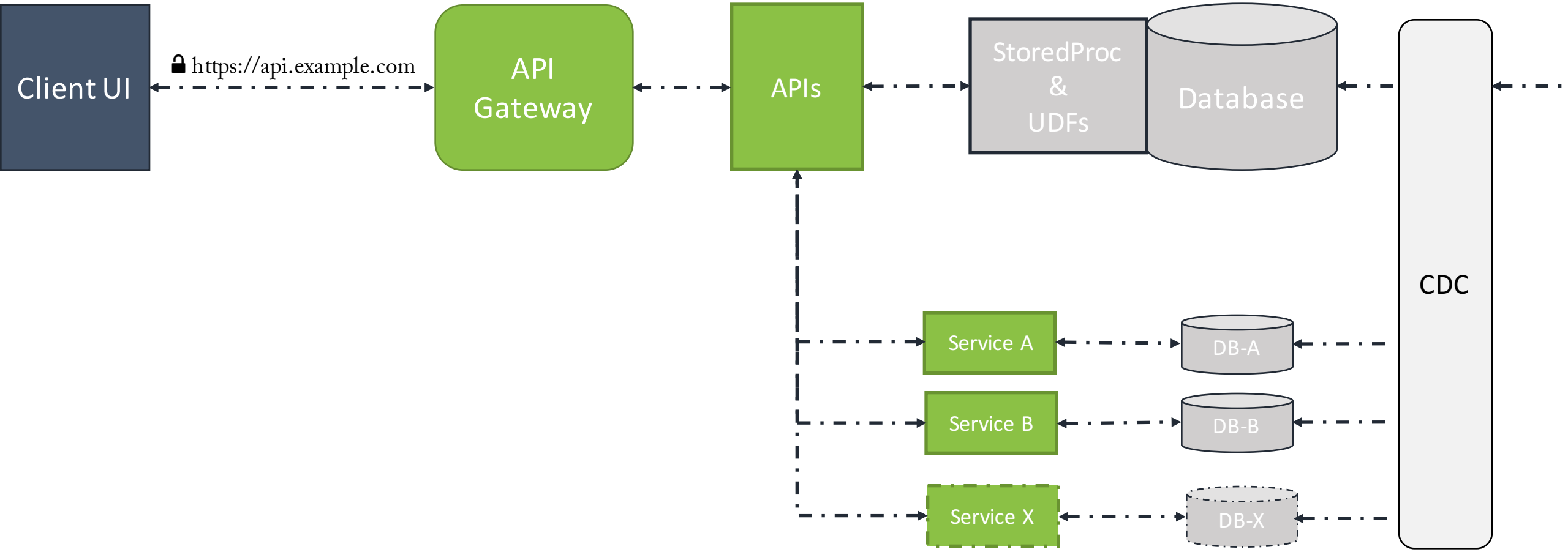


- mock API request & response using **Swagger**
- mock environments as first-class citizen
- **reusable** API tests – contract, load, security
- tooling interoperability using Swagger

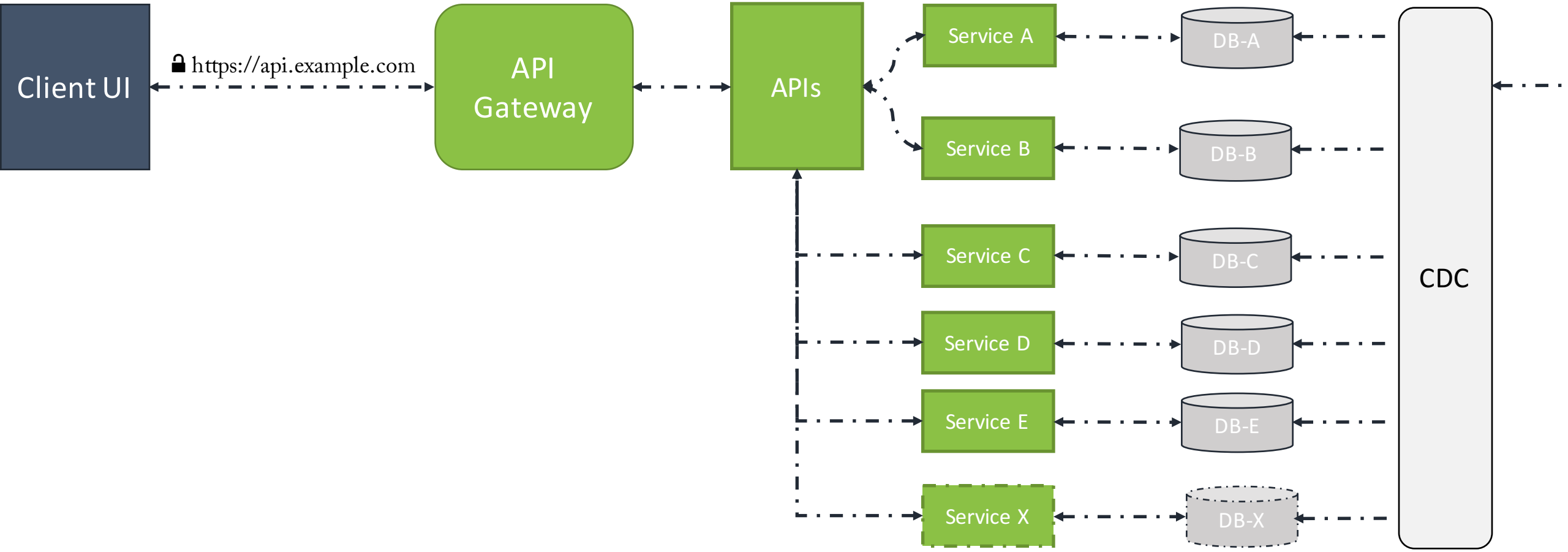
Breaking Monolith

- strict rule - no new StoredProc
- separate business logic & data logic
- build microservices – one at a time
- each microservice owns it's own data store

Transition


Client UI
https://api.example.com
API Gateway
APIs
StoredProc & UDFs
Database
CDC
Service A
DB-A
Service B
DB-B
Service X
DB-X

Future


Client UI
https://api.example.com
API Gateway
APIs
Service A
Service B
Service C
Service D
Service E
Service X
DB-A
DB-B
DB-C
DB-D
DB-E
DB-X
CDC

- more than 200 APIs in less than 6 months
- two new native mobile products
- a new responsive web product
- seamless opportunities – API economy

- team maturity drives **API maturity** model
- experience design influences **API reusability**
- **developer** experience defines overall success
- use **Swagger** as epicentre of you API program

Q&A