



# Chopping the monolith



# Me, myself and I



- Developer
- Developer advocate



 @nicolas\_frankel



# Disclaimer



- Contains a lot of controversy inside





# The sad state we're in

- Monolith = bad
- Microservices = good



# What are microservices anyway?

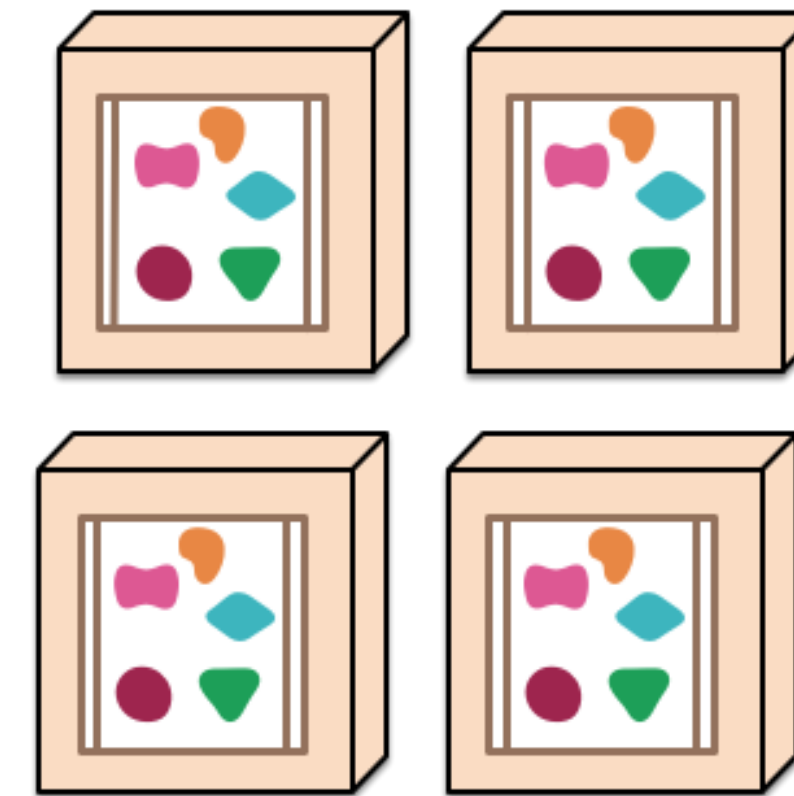


“In short, the microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.”

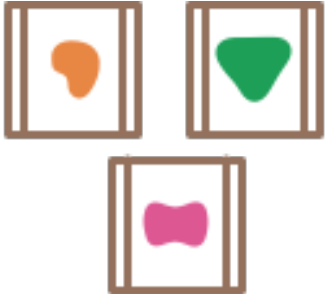
*A monolithic application puts all its functionality into a single process...*



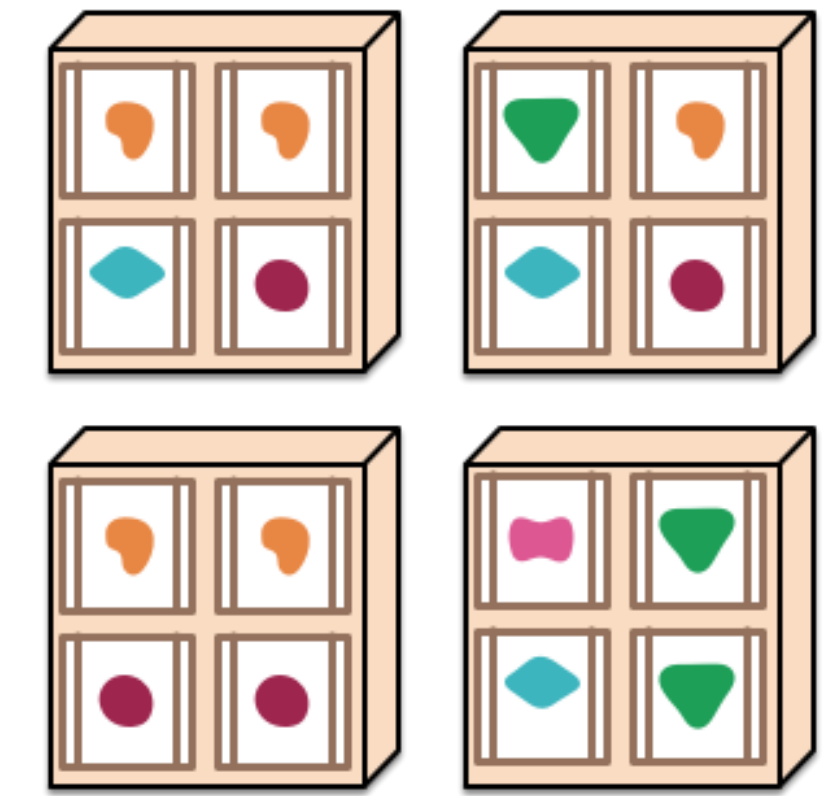
*... and scales by replicating the monolith on multiple servers*



*A microservices architecture puts each element of functionality into a separate service...*



*... and scales by distributing these services across servers, replicating as needed.*



-- <https://martinfowler.com/articles/microservices.html>



# Characteristics of microservices

- Componentization via Services
- Organized around Business Capabilities
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for failure
- Evolutionary Design
- Products not Projects





# Conway's Law



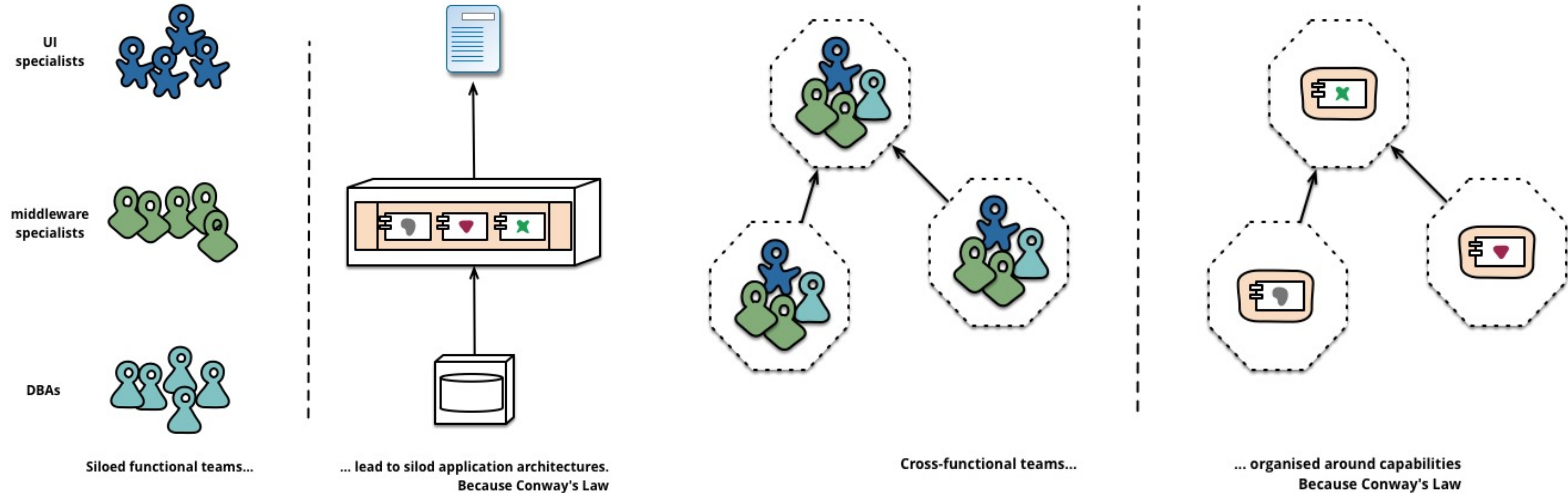
“Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization’s communication structure.”

-- Melvin E. Conway





# Reversing Conway's Law





# Amazon Web Services, a poster child for microservices



“We try to create teams that are no larger than can be fed by two pizzas,” said Bezos. “We call that the two-pizza team rule.”

-- <https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/two-pizza-teams.html>





Whose organization is like this?



 @nicolas\_frankel



# A recipe for failure

1. An architect/lead reads about microservices
2. Remembers only the benefits
3. Applies only the technical aspects
4. Leaves for another job with a shinier CV





# What's the main reason behind microservices?



Benefits	Costs
<ul style="list-style-type: none"><li>• Strong Module Boundaries</li><li>• <u>Independent Deployment</u></li><li>• Technology Diversity</li></ul>	<ul style="list-style-type: none"><li>• Distribution</li><li>• Eventual Consistency</li><li>• Operational Complexity</li></ul>



# Strong module boundaries



- Many other ways to enforce boundaries
- With less downsides





# Technology diversity



- Satisfies some people's aspirations
- Doesn't help the organization as a whole





# Lead time, one of the Golden DevOps metrics





# How did we manage releases “back in the days”?

- Unfrequent releases
  - Release trains
- You don't want to miss the train!
  - Bugfixes are allowed
  - Shove feature into a bugfix





It's not possible to release often and test monoliths well



 @nicolas\_frankel



# The real problem

- Not all parts of an app change at the same speed
- Some are more stable than others
- Reasons for change
  - Business “requirement”
  - Law





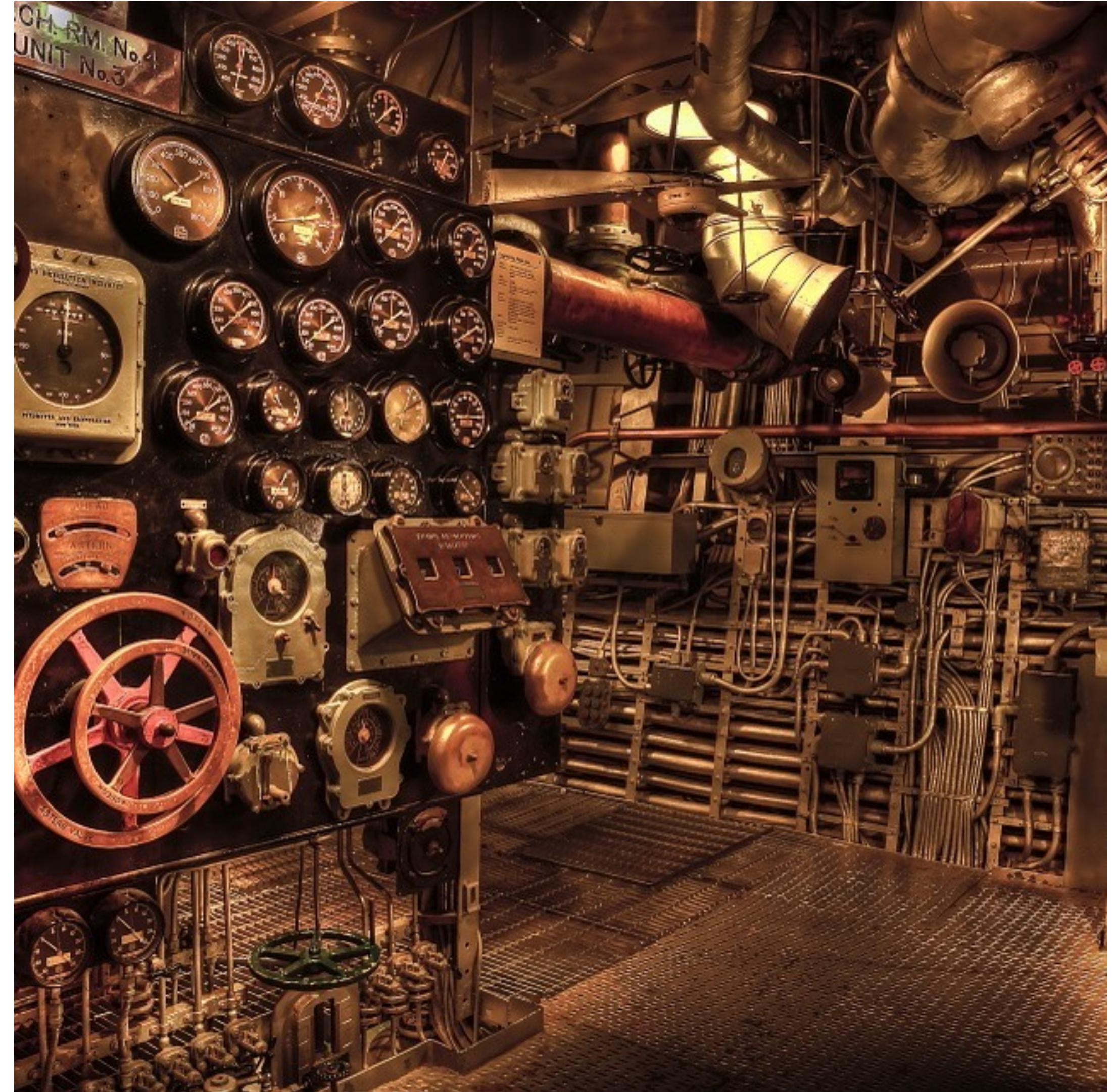
# Rules engine



A business rules engine is a software system that executes one or more business rules in a runtime production environment. The rules might come from legal regulation, company policy, or other sources. A business rule system enables these company policies and other operational decisions **to be defined, tested, executed and maintained separately from application code.**

-- [https://en.wikipedia.org/wiki/Business\\_rules\\_engine](https://en.wikipedia.org/wiki/Business_rules_engine)

 @nicolas\_frankel





# Characteristics of rules engines

- No release
- The business changes the rules how often they want
- With great power comes great responsibility





# Isolate the quick-changing part

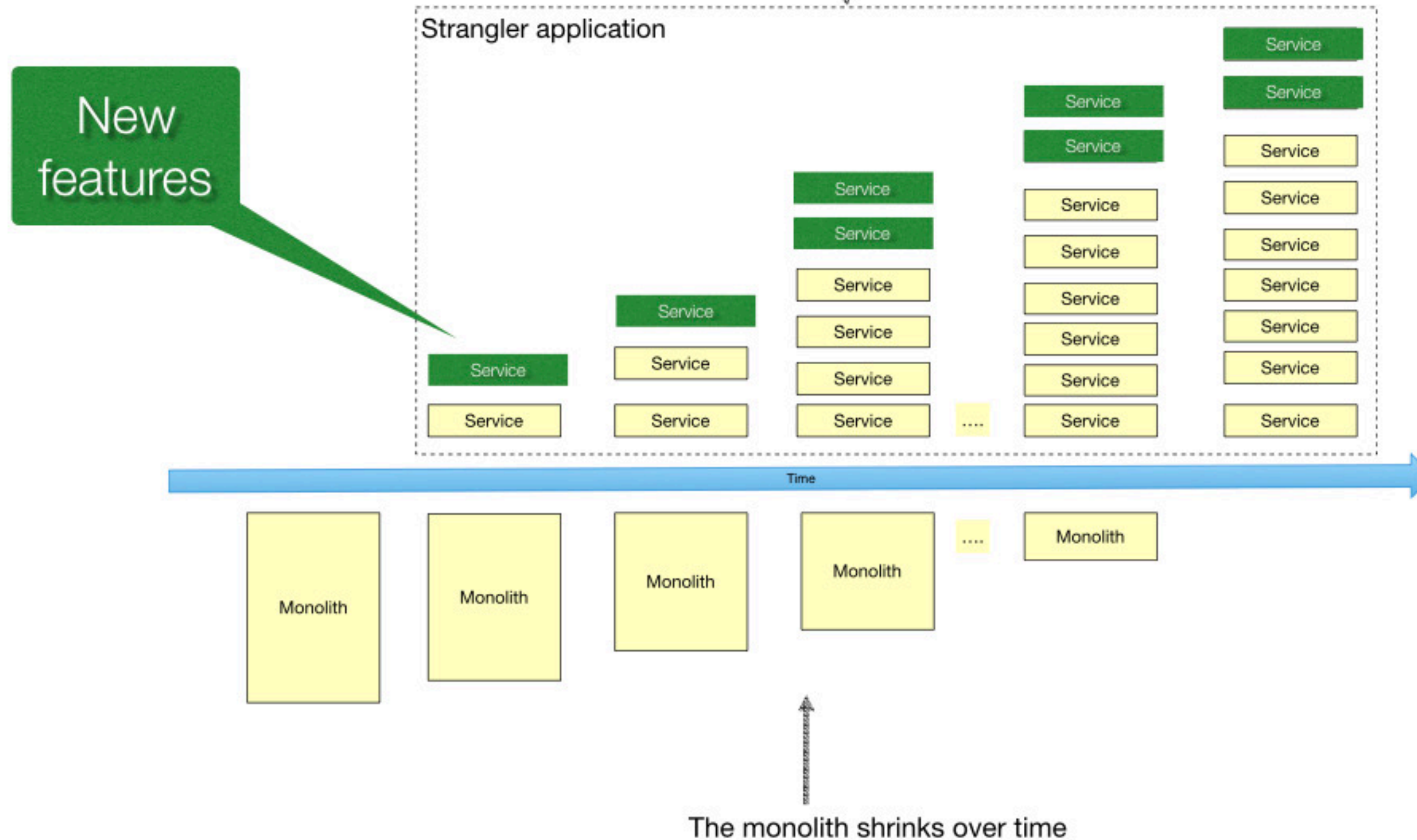
- Alternative implementations:
  - Rules engine
  - Microservice
  - Serverless function
  - Something else?





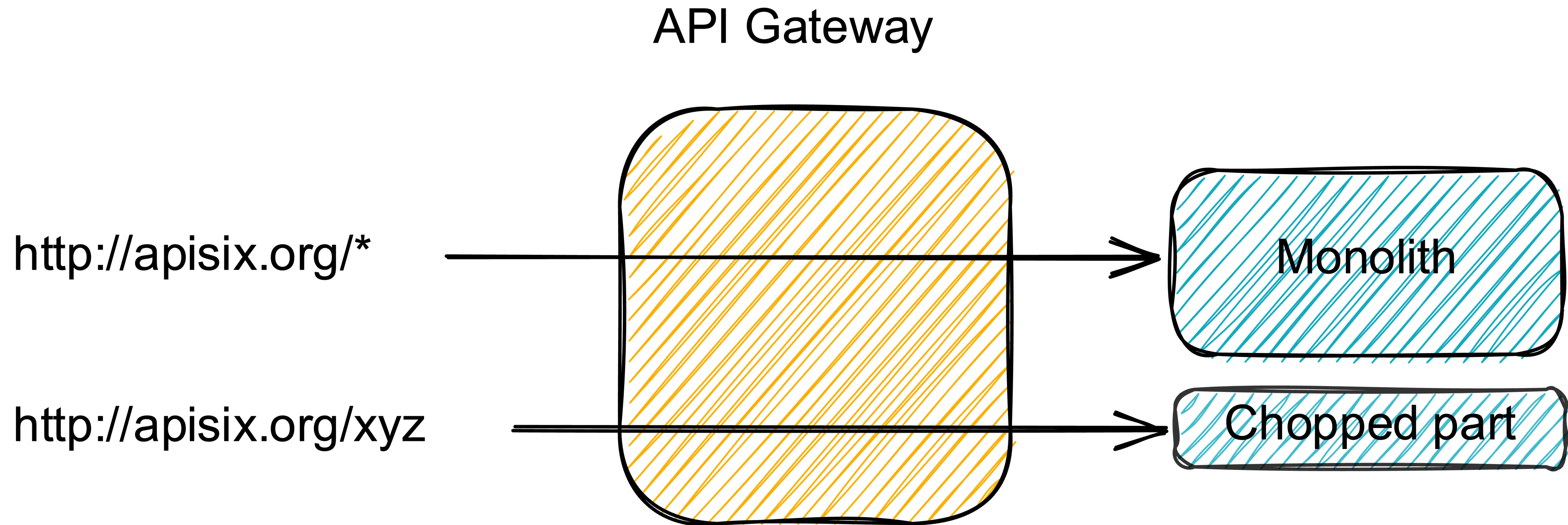
# Strangling the monolith

The strangler application grows larger over time





# “Chop” the part





Example: an e-commerce shop

- Business wants to push some products
  - Too much stock
  - End-of-season leftovers
  - High margin product
  - Flagship product





# Sell more by lowering price

- Pricing should be very flexible
- Impossible to model pricing options ahead of time
- “Chop” the pricing engine
  - Don’t break the clients!





Thanks for your attention!



- @nicolas\_frankel
- <https://bit.ly/chop-monolith/>
- <https://blog.frankel.ch/chopping-monolith/>
- <https://apisix.apache.org/>

 @nicolas\_frankel

