Managing Data in Microservices

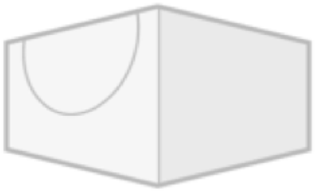
Randy Shoup
@randyshoup
linkedin.com/in/randyshoup

Background

- VP Engineering at Stitch Fix
  - Using technology and data science to revolutionize clothing retail
- Consulting "CTO as a service"
  - Helping companies move fast at scale ☺
- Director of Engineering for Google App Engine
  - World's largest Platform-as-a-Service
- Chief Engineer at eBay
  - Evolving multiple generations of eBay's infrastructure

Stitch Fix

Create Your Style Profile.

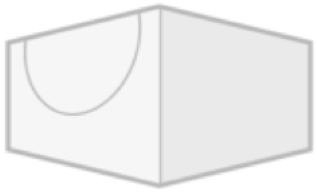


Get Five Hand-picked Items.

Keep What You Like.
Send Back the Rest.

Stitch Fix

Create Your Style Profile.



Get Five Hand-picked Items.

Keep What You Like.
Send Back the Rest.

How do you prefer clothes to fit the top half of your body?

- ✓
- Mostly Tight / Form Fitting
- Prefer Fitted / Showing my Figure
- Straight
- Mostly Loose
- Oversized

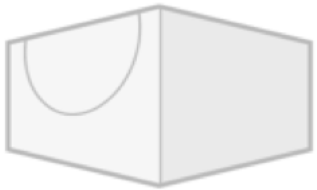
ear for pants, jeans, and skirts?

an waist

Skirts L

Stitch Fix

Create Your Style Profile.

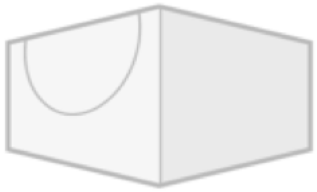


Get Five Hand-picked Items.

Keep What You Like.
Send Back the Rest.

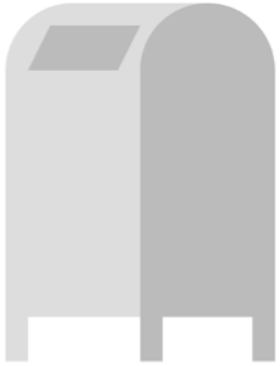
Stitch Fix

Create Your Style Profile.



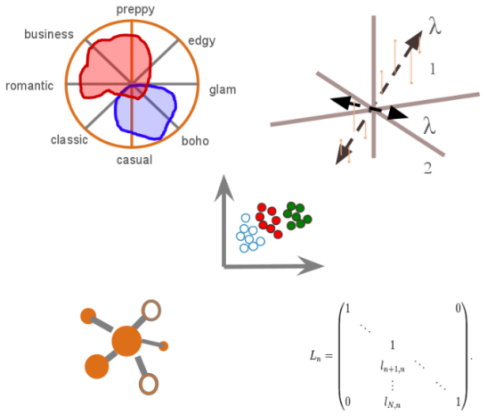
Get Five Hand-picked Items.

Keep What You Like.
Send Back the Rest.



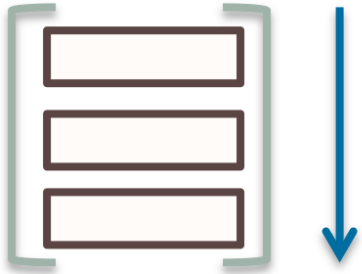
Personalized Recommendations

Inventory



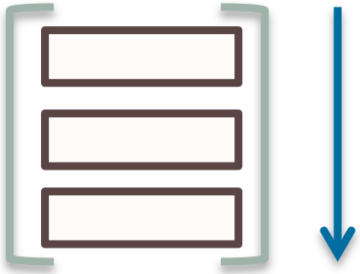
Machine learning

Algorithmic recommendations

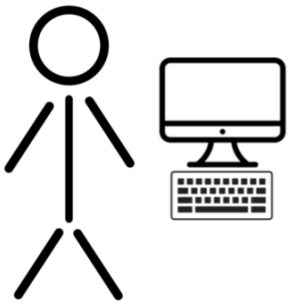


Expert Human Curation

Algorithmic recommendations



Human curation



Data at the Center

- 1:1 Ratio of Data Science to Engineering
  - More than 100 software engineers
  - ~80 data scientists and algorithm developers
  - Unique ratio in our industry
- Apply intelligence to *every* part of the business
  - Buying
  - Inventory management
  - Logistics optimization
  - Styling recommendations
  - Demand prediction
- Humans and machines augmenting each other

Design Goals

- Feature Velocity
  - Teams can move rapidly and independently
- Scalability
  - Components can be scaled independently depending on load
- Resilience
  - Component failures are isolated, do not cascade

High-Performing Organizations

- **Multiple deploys per day** vs. one per month
- **Commit to deploy in less than 1 hour** vs. one week
- **Recover from failure in less than 1 hour** vs. one day
- **Change failure rate of 0-15%** vs. 31-45%

https://puppet.com/resources/whitepaper/state-of-devops-report

High-Performing Organizations

➔2.5x more likely to exceed business goals

- Profitability
- Market share
- Productivity

https://puppet.com/resources/whitepaper/state-of-devops-report

"Tell us how you did things at Google and eBay."

…

"Sure, I will tell you, but you have to promise not to do them! [… yet]"

Evolution to Microservices

- eBay
  - 5th generation today
  - Monolithic Perl → Monolithic C++ → Java → microservices
- Twitter
  - 3rd generation today
  - Monolithic Rails → JS / Rails / Scala → microservices
- Amazon
  - Nth generation today
  - Monolithic Perl / C++ → Java / Scala → microservices

No one starts with microservices

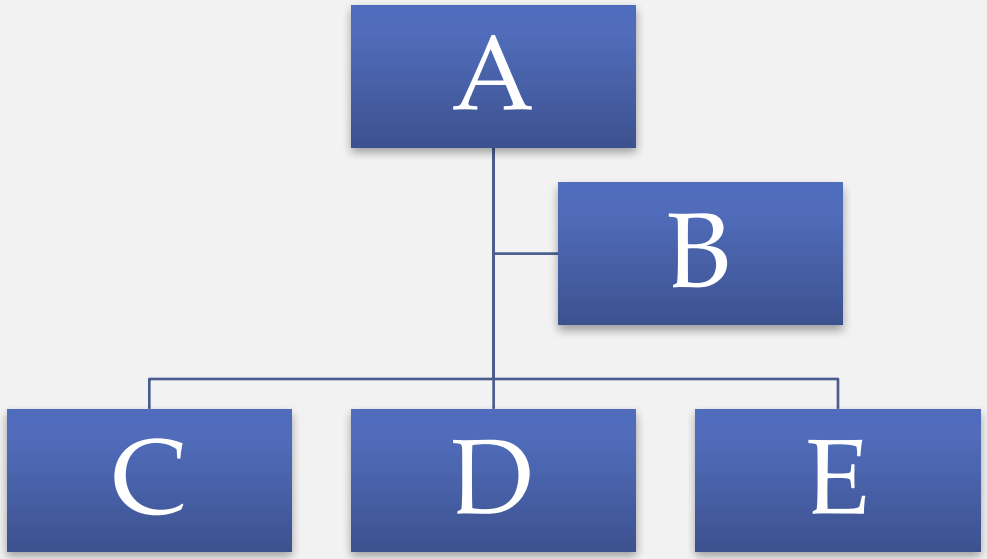
…

Past a certain scale, everyone ends up with microservices

If you don't end up regretting your early technology decisions, you probably over-engineered.

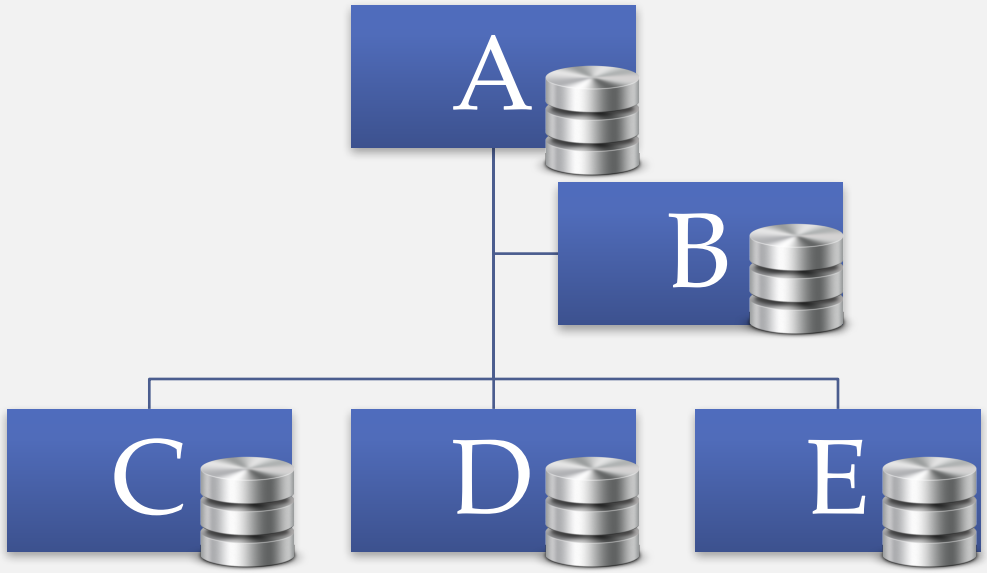
Microservices

- Single-purpose
- Simple, well-defined interface
- Modular and independent



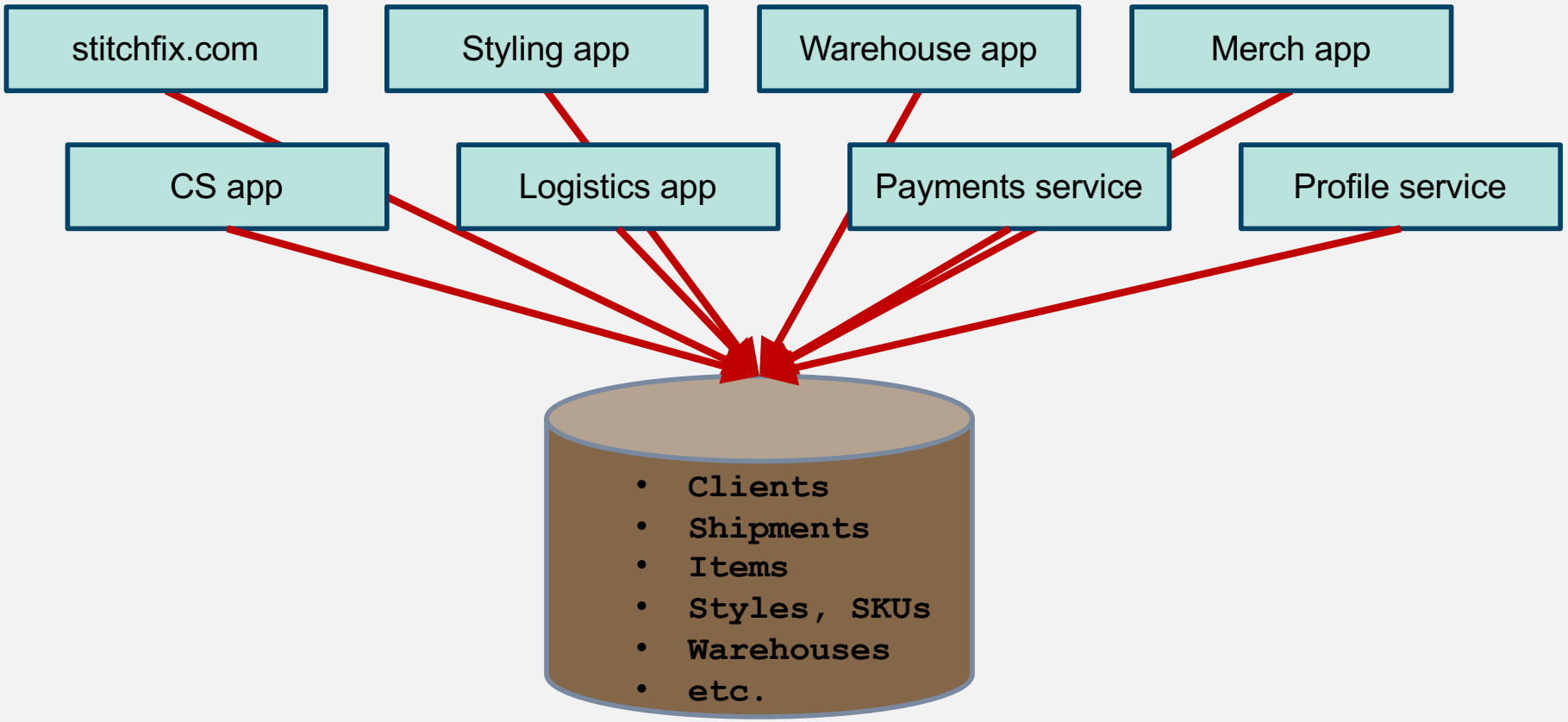
Microservices

- Single-purpose
- Simple, well-defined interface
- Modular and independent
- **Isolated persistence (!)**



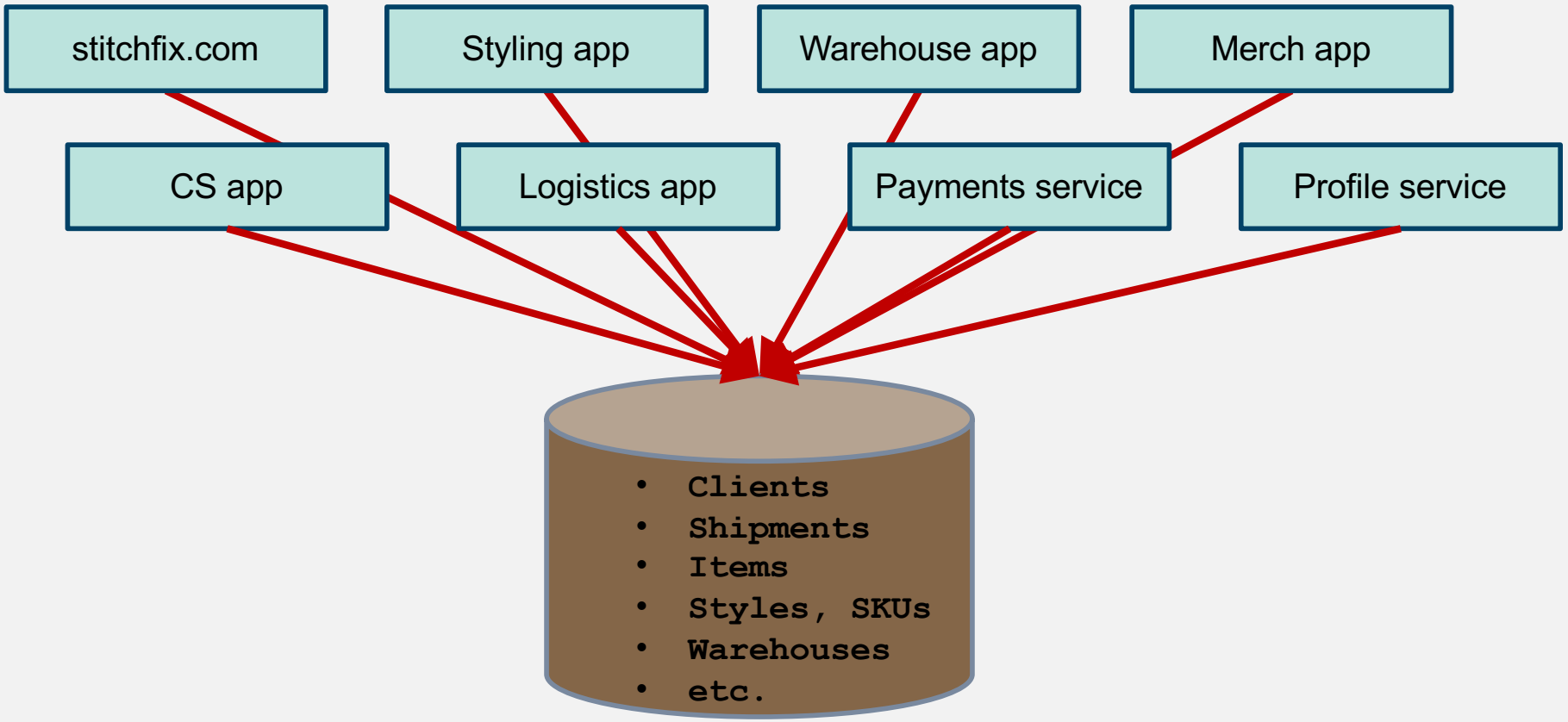
Extracting Microservices

- Problem: Monolithic shared DB



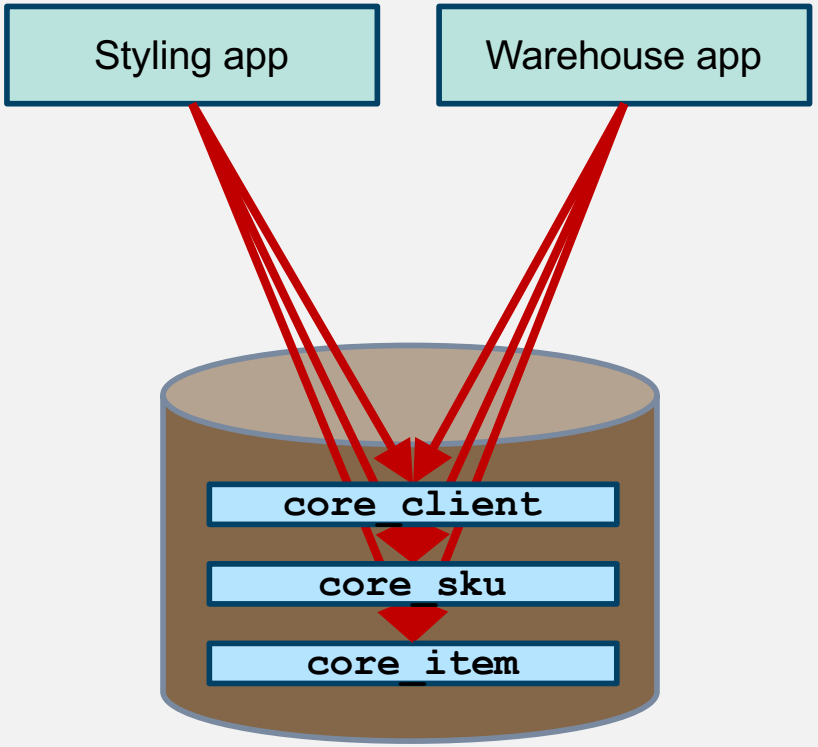
Extracting Microservices

- Decouple applications / services from shared DB



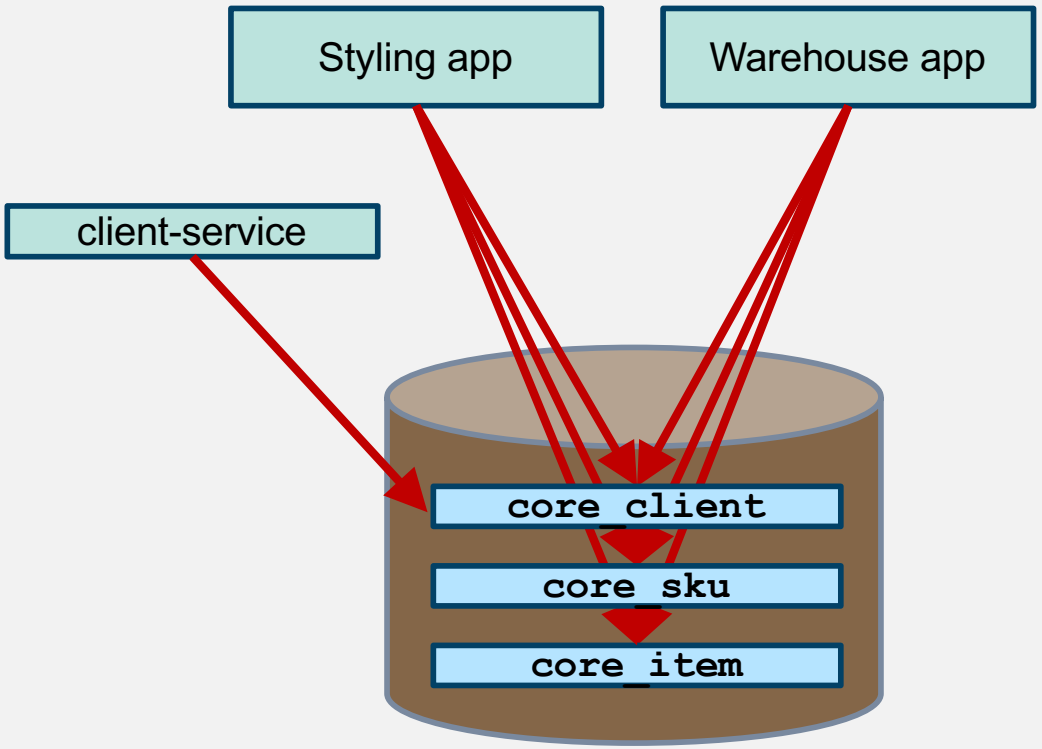
Extracting Microservices

- Decouple applications / services from shared DB



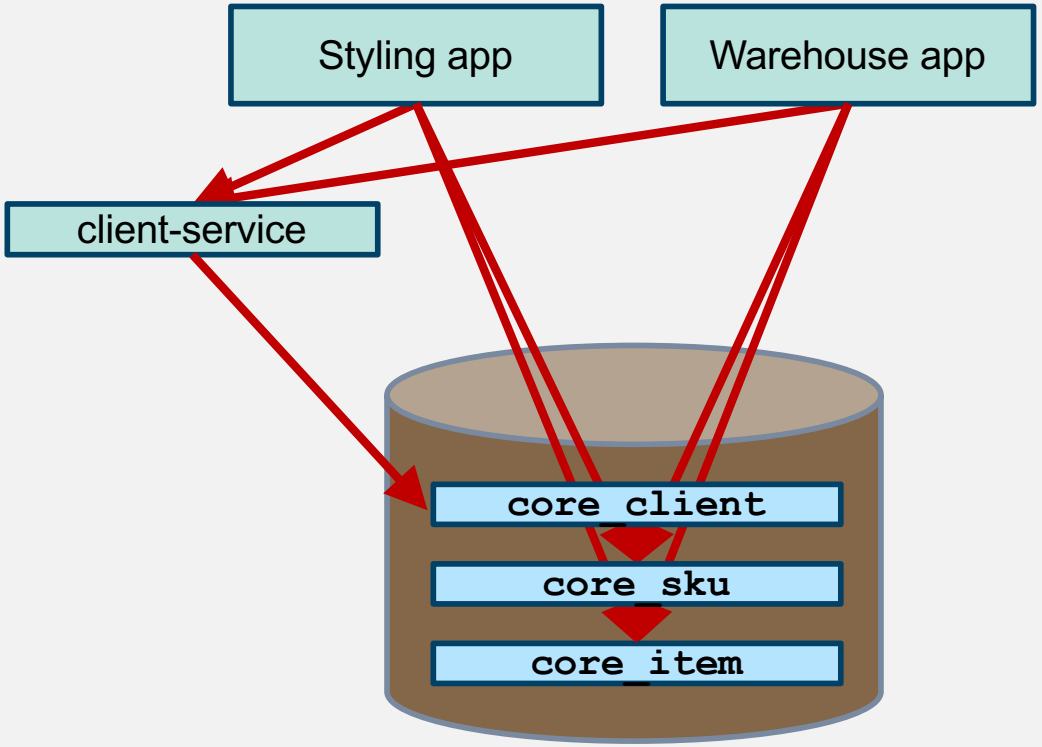
Extracting Microservices

- Step 1: Create a service



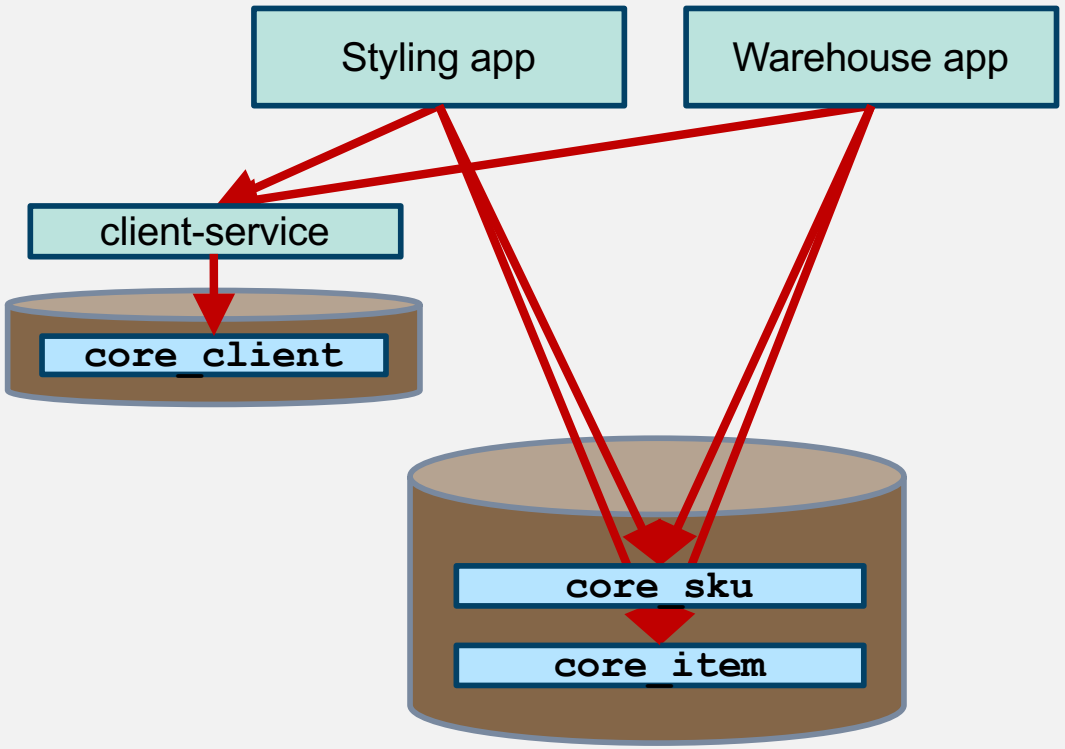
Extracting Microservices

- Step 2: Applications use the service



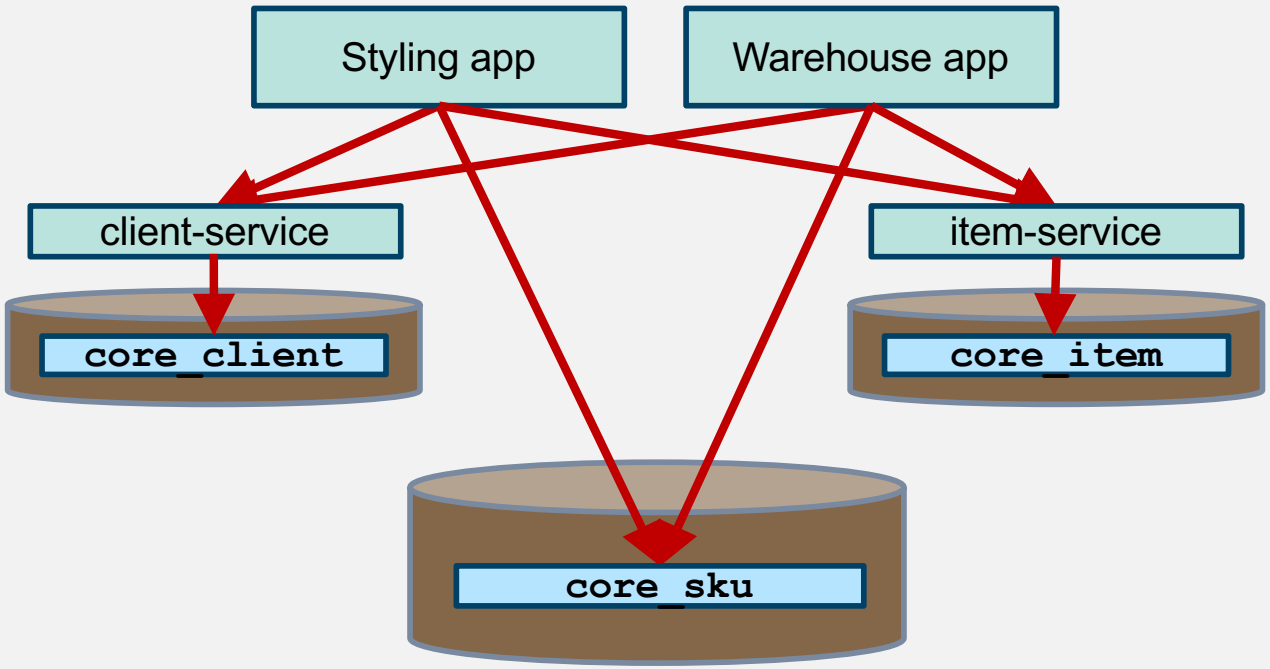
Extracting Microservices

- Step 3: Move data to private database



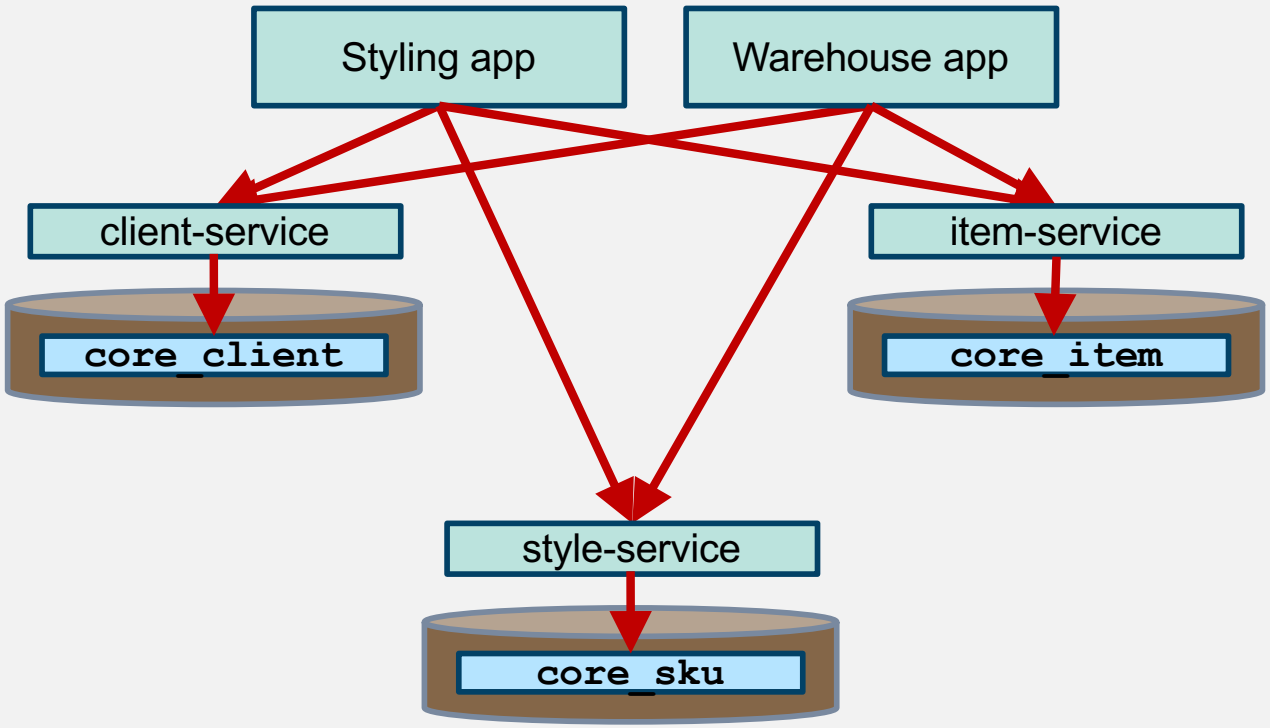
Extracting Microservices

- Step 4: Rinse and Repeat



Extracting Microservices

- Step 4: Rinse and Repeat



With Microservices, how do we do

- Shared Data
- Joins
- Transactions

Events as First-Class Construct

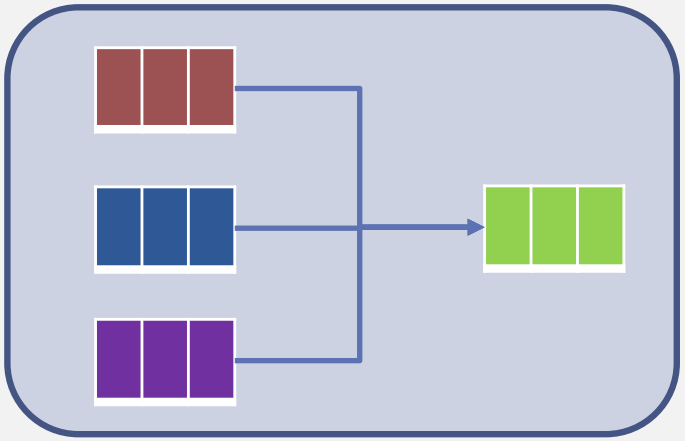
- "A significant change in state"
  - Statement that some interesting thing occurred
- Traditional 3-tier system
  - Presentation ➔ interface / interaction
  - Application ➔ stateless business logic
  - Persistence ➔ database
- Fourth fundamental building block
  - **State changes ➔ events**
  - 0 | 1 | N consumers subscribe to the event, typically asynchronously

Microservices and Events

- Events are a first-class part of a service interface
- A service interface includes
  - Synchronous request-response (REST, gRPC, etc)
  - Events the service produces
  - Events the service consumes
  - Bulk reads and writes (ETL)
- The interface includes *any mechanism for getting data in or out of the service (!)*

Microservice Techniques: Shared Data

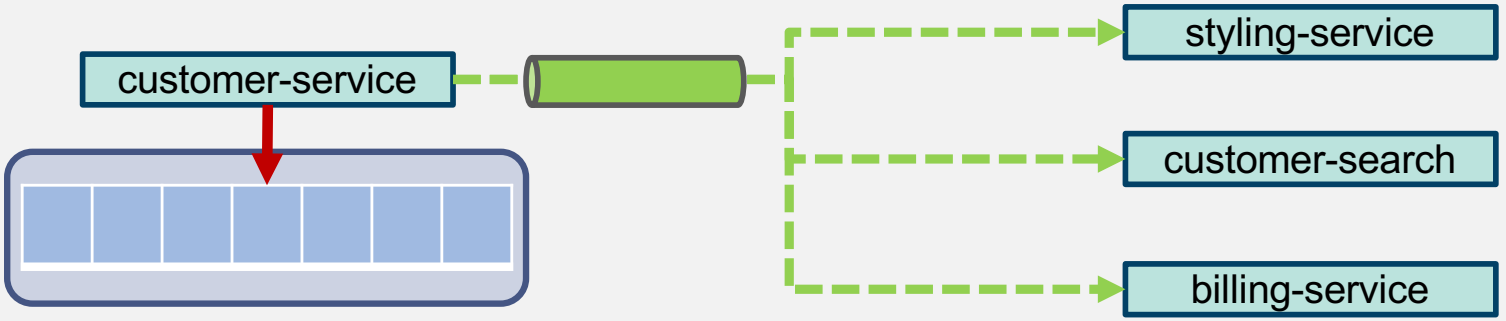
- Monolithic database makes it easy to leverage shared data



- Where does shared data go in a microservices world?

Microservice Techniques: Shared Data

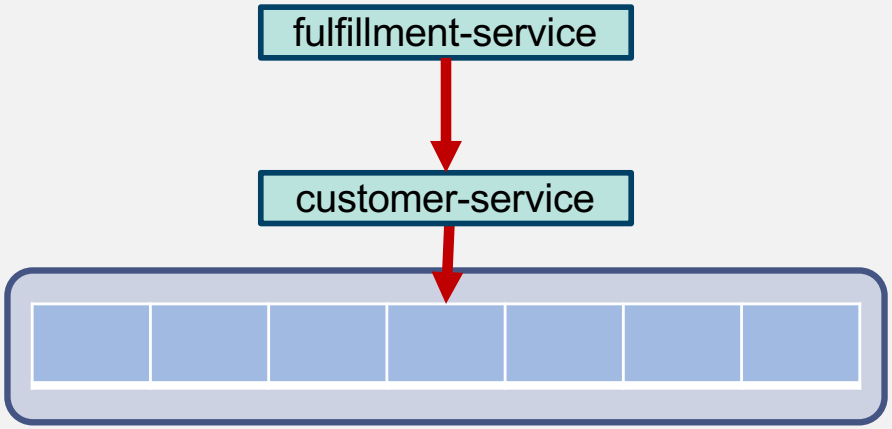
- Principle: Single System of Record
  - Every piece of data is owned by a single service
  - That service is the **canonical system of record** for that data



- Every other copy is a **read-only, non-authoritative cache**

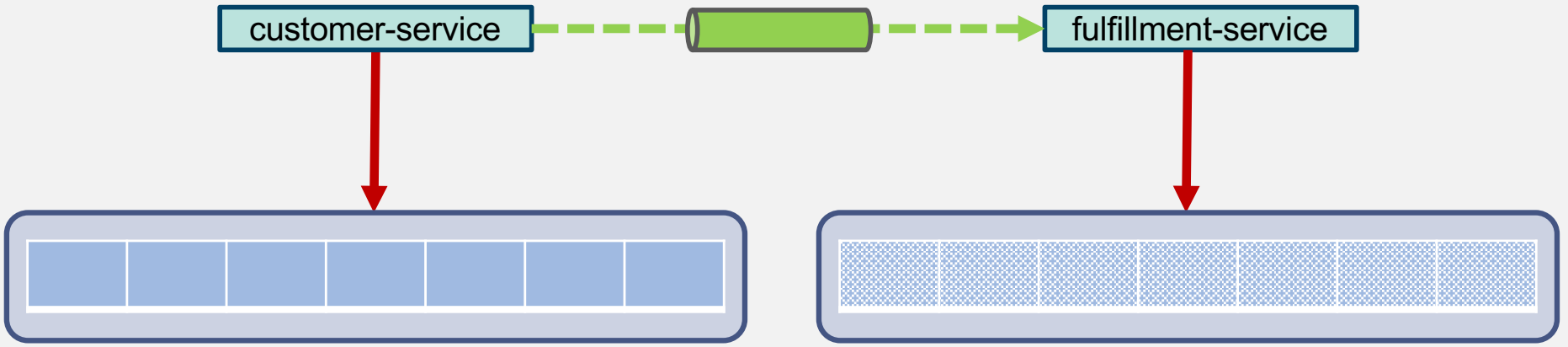
Microservice Techniques: Shared Data

- Approach 1: Synchronous Lookup
  - Customer service owns customer data
  - Fulfillment service calls customer service in real time



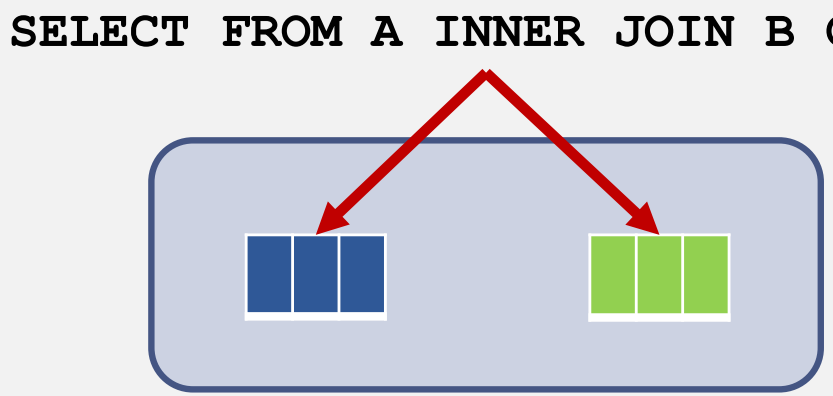
Microservice Techniques: Shared Data

- Approach 2: Async event + local cache
  - Customer service owns customer data
  - Customer service sends `address-updated` event when customer address changes
  - Fulfillment service caches current customer address



Microservice Techniques: Joins

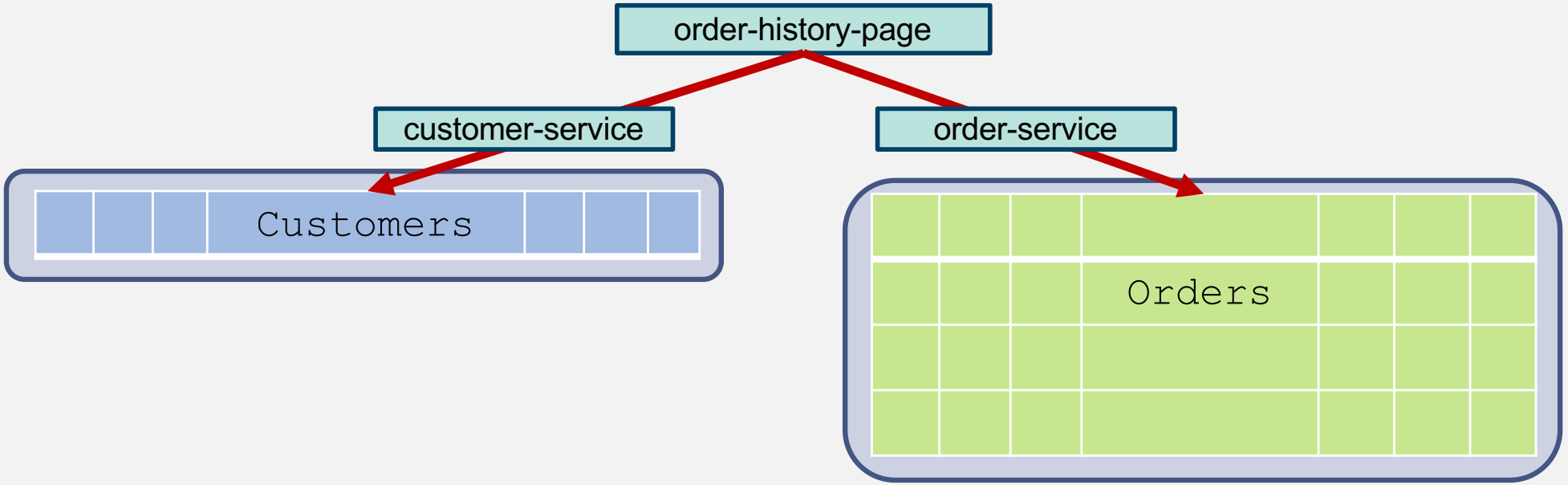
- Monolithic database makes it easy to join tables



- Splitting the data across microservices makes joins very hard

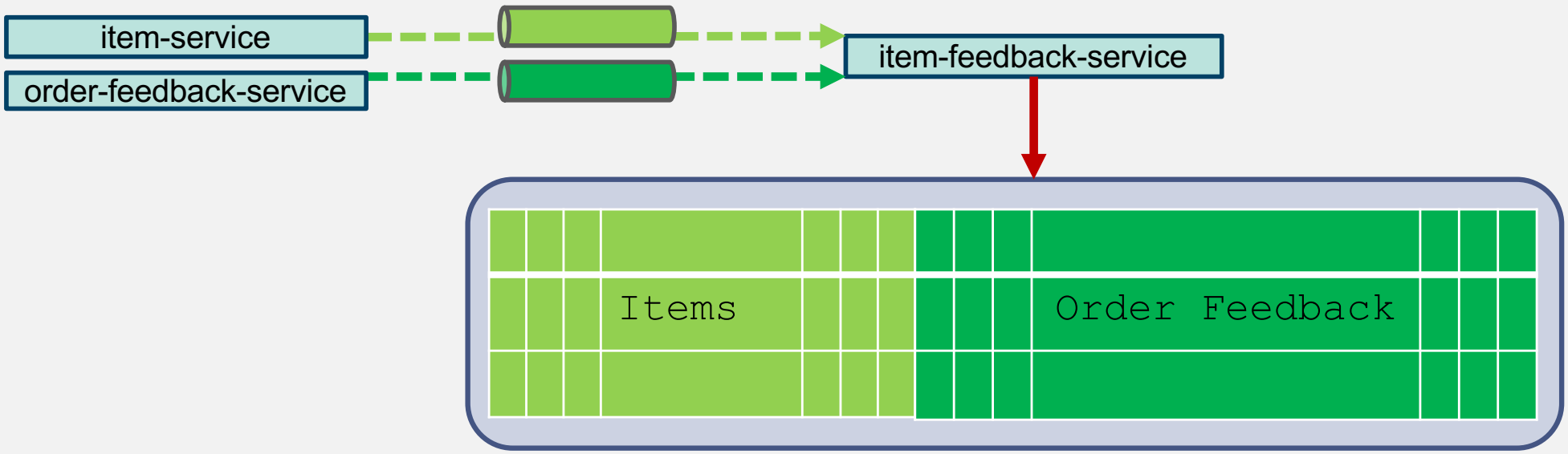
Microservice Techniques: Joins

- Approach 1: Join in Client Application
  - Get a single customer from `customer-service`
  - Query matching orders for that customer from `order-service`



Microservice Techniques: Joins

- Approach 2: Service that "Materializes the View"
  - Listen to events from `item-service`, events from `order-service`
  - Maintain denormalized join of items and orders together in local storage



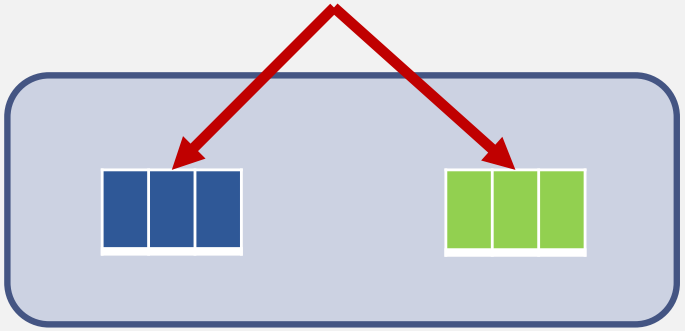
Microservice Techniques: Joins

- Many common systems do this
  - "Materialized view" in database systems
  - Most NoSQL systems
  - Search engines
  - Analytic systems

Microservice Techniques: Workflows and Sagas

- Monolithic database makes transactions across multiple entities easy

```
BEGIN; INSERT INTO A …; UPDATE B...; COMMIT;
```



- Splitting data across services makes transactions very hard

Microservice Techniques: Workflows and Sagas

- Transaction ➔ Saga
  - Model the transaction as a state machine of atomic events
- Reimplement as a workflow

- Roll back by applying compensating operations in reverse

Microservice Techniques: Workflows and Sagas

- Many common systems do this
  - Payment processing
  - Expense approval
  - Any multi-step workflow

With Microservices, how do we do

- Shared Data
- Joins
- Transactions

Thanks!

- Stitch Fix is hiring!
  - www.stitchfix.com/careers
  - Based in San Francisco
  - **Hiring everywhere!**
  - More than half remote, all across US
  - Application development, Platform engineering, Data Science

- Please contact me
  - @randyshoup
  - linkedin.com/in/randyshoup