Leveraging FHIR for Microservice Strategy

Ann Guilinger and Bryan Knight, athenahealth

Boston, 19-21 June | @HL7 @FirelyTeam | #fhirdevdays18 | www.fhirdevdays.com

The Presenters

- Ann Guilinger
  - Developer at athenahealth for 3 years, working on ambulatory EHR product
- Owns a cute dog (Tabitha!)
- Twitter: @aguilinger

The Presenters

Bryan Knight

- Principal Member of Technical Staff
- +12 years development experience
- Pizza Lover

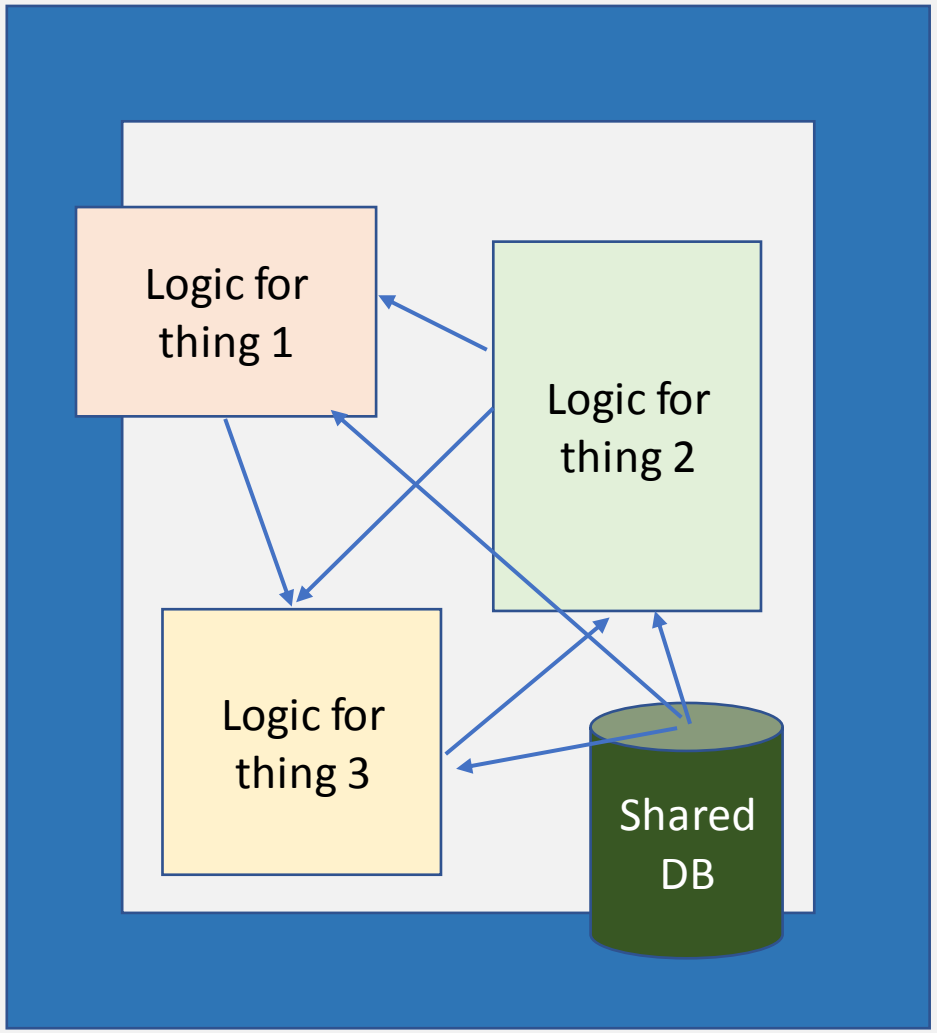
Twitter @bryanjknight

Agenda

- What are the common pitfalls of a monolith application
- How can microservices help
- How can FHIR help with communication
- How to move from the monolith to the microservice

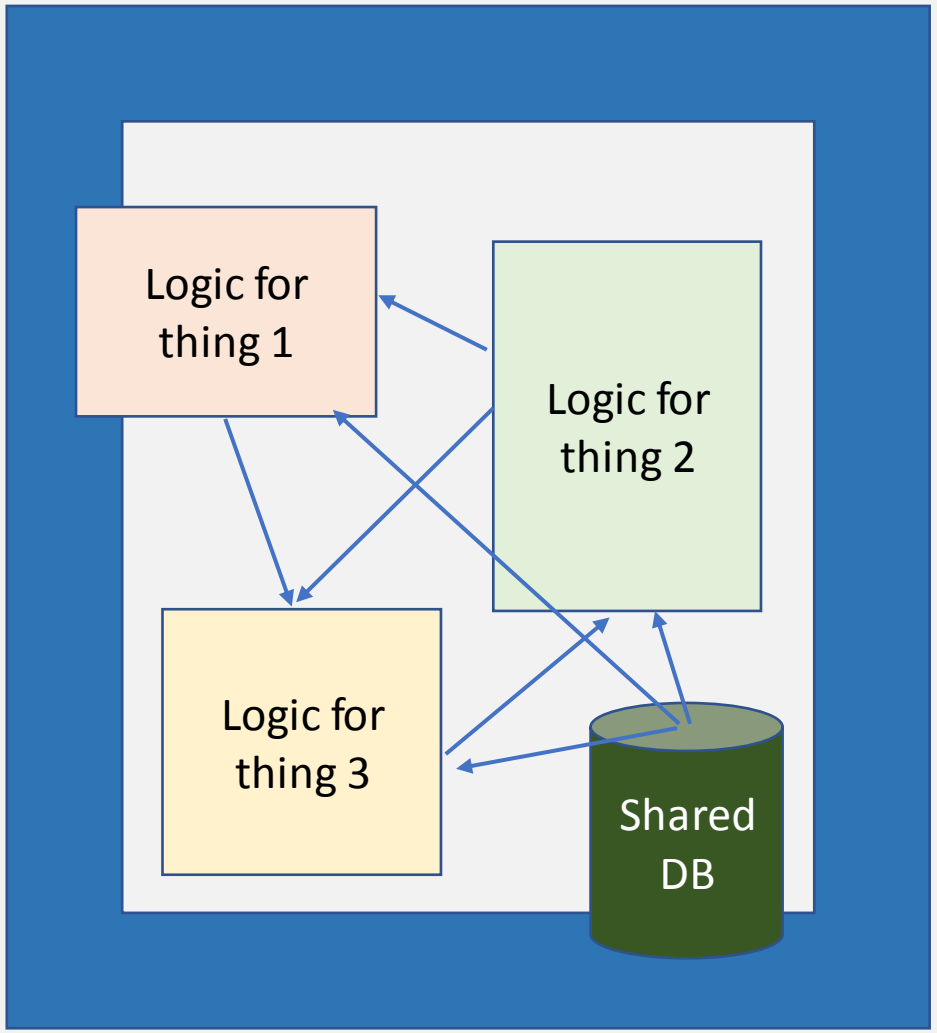
What's a monolith

- Server-side logic in web/cloud-based applications
  - Build/bundle up all logic & run it
  - Change anything, rebundle & deploy
- Problems
  - Tightly coupled
  - Minimal separation of concerns
  - Only as stable as the weakest link
  - Scales as a single unit
  - Complexity scales as application scales



Monolith Problems

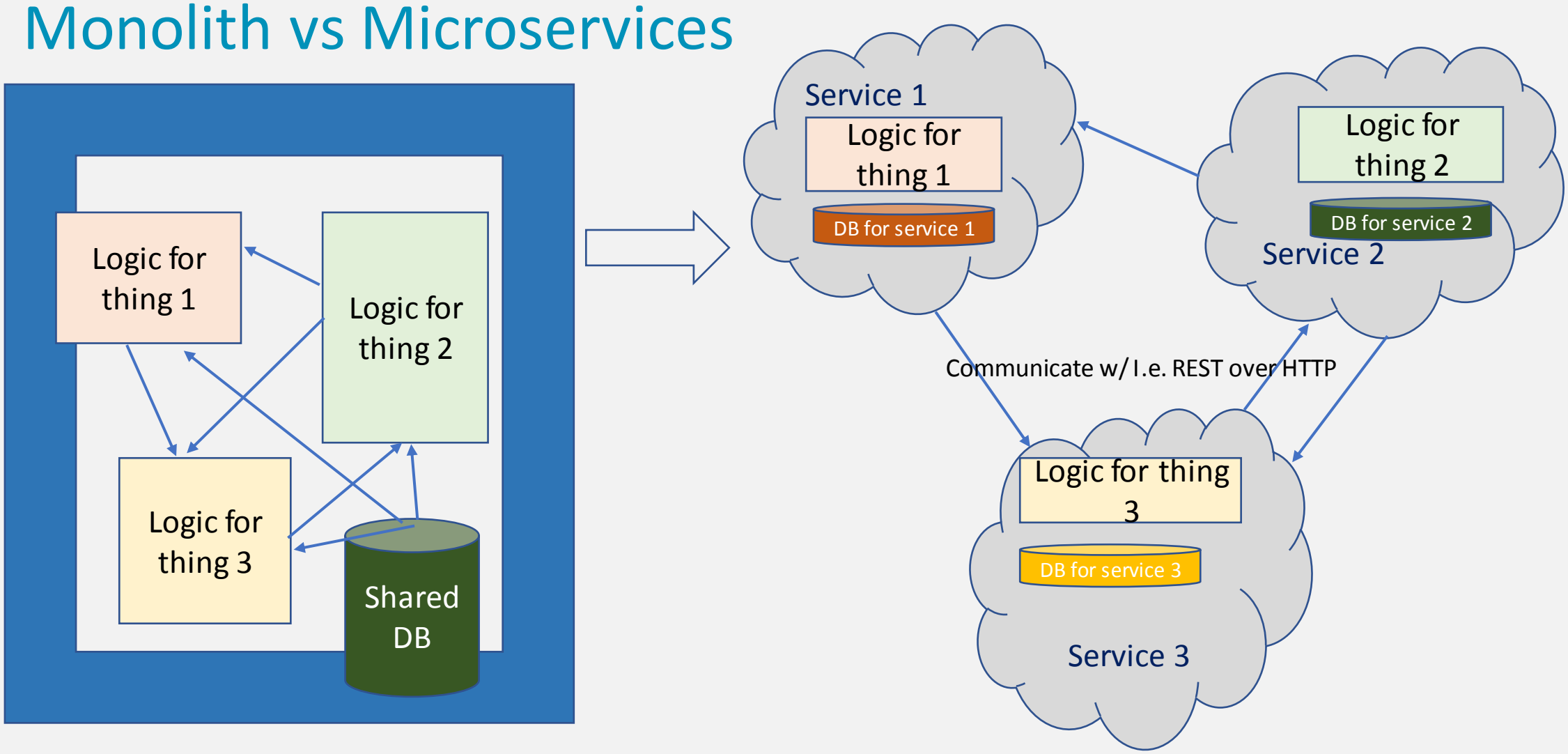
- Scaling the monolith means scaling *everything*
  - Example: “Logic for thing 3” takes x amount of CPU
    - Every server that runs the application now must have at least x CPU
- Adding logic adds complexity
  - Change anything about “Logic for thing 3”, must test Thing 1 and Thing 2 as well



Monolith Problems

- Sharing a database
  - Change the data model, update every place that reads from the DB
  - Single type of database
- Stability
  - Any issue affects the whole application
  - Hard to protect critical workflows

Monolith vs Microservices



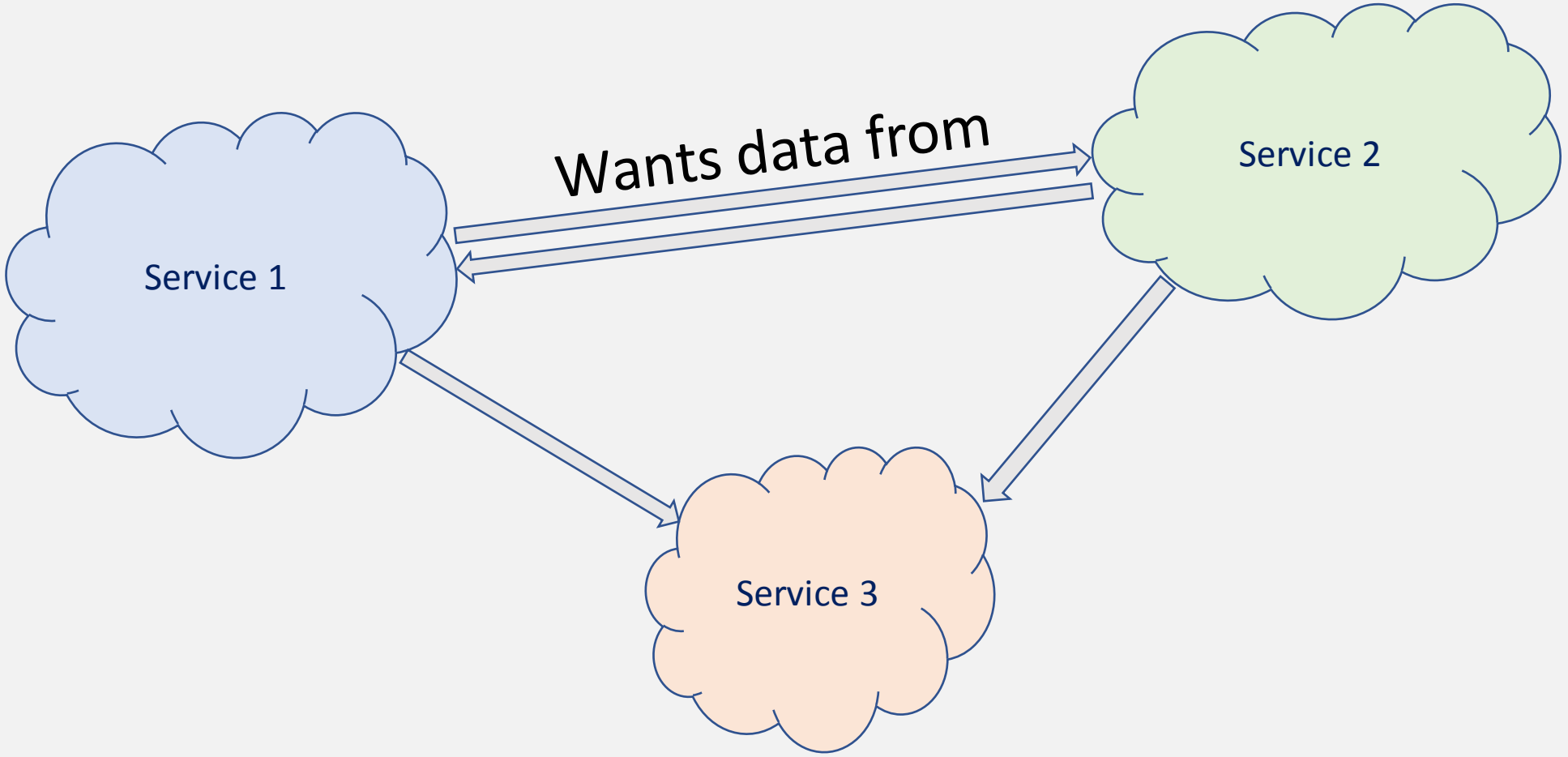
Microservices

- Keeping it simple
  - Clearly defined problem (do one thing and do it well)
  - Clearly defined interface for others to access data
- Using the best tools for the problem
  - What data store is best for my use case?
  - What programming language is best for my use case?
- Scalability and Stability
  - Increasing number of servers instead of increasing the server size
  - Monitoring state

Microservice Communication

- Microservices must keep their own data & keep in sync
  - Data can be accessed by multiple microservices
- Understanding each other's data becomes critical
  - Many services will want the same data/data from other services
    - I.E. Sending an order from the order service should flow back and populate in the chart
  - As data flows, don’t want meaning to be lost/misunderstood

Microservice Communication



Microservice Communication

Now we have the classic problem of interoperability within our application!

Solution:

FHIR

FHIR for Communication

- Can use FHIR-inspired models to communicate between microservices
- Benefits of using FHIR
  - Expertly modeled
  - Standard data model
  - Can leverage open-source tools
    - And hopefully contribute back!
  - Get classic interoperability out of the box
    - External FHIR servers, CDS, quality measurement, 3rd party apps etc...

Breaking up the Monolith

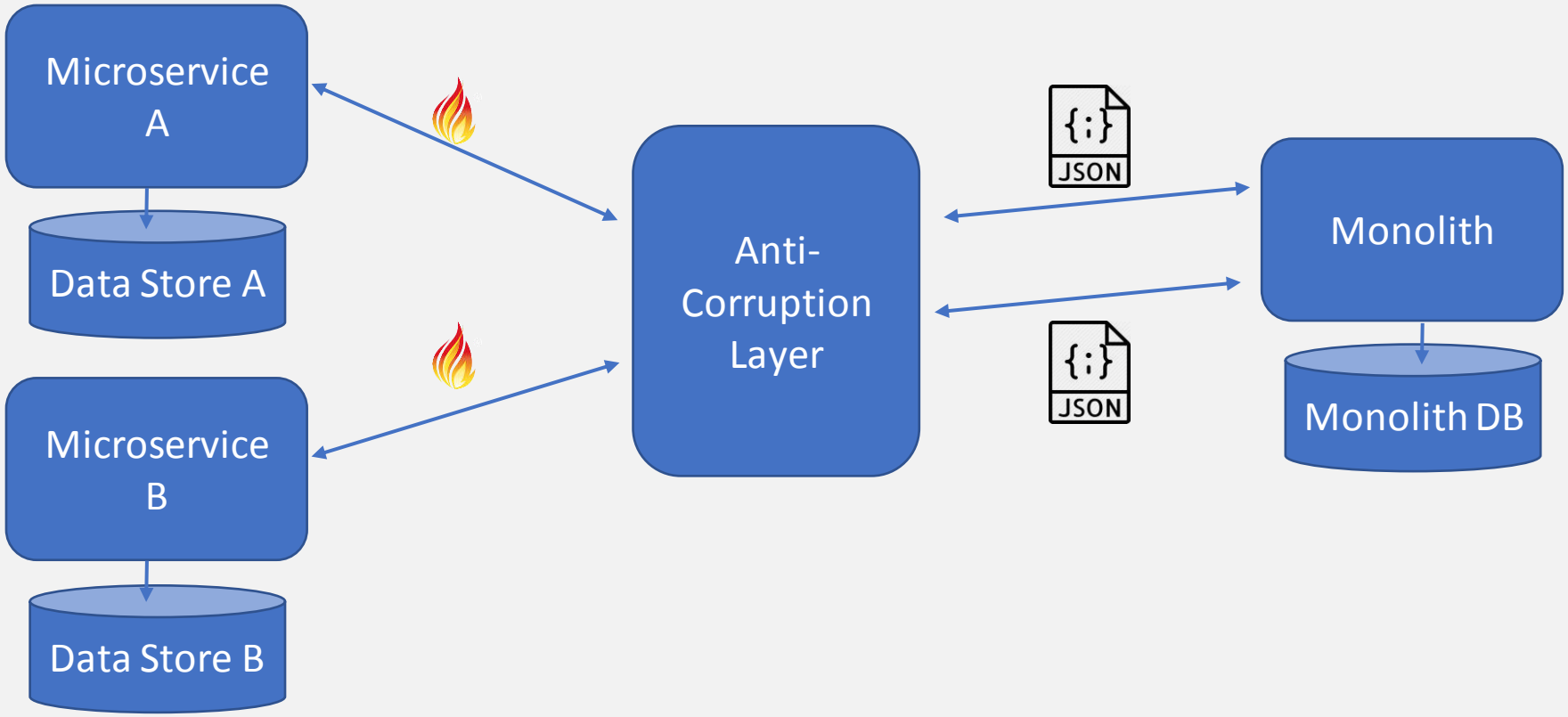
Overly simplified process:

1. Get something simple working that's detached from monolith
2. Intercept events from monolith and send them also to the new microservice (dark launch)
3. Reconcile the two systems for data
4. Switch over via a rollout

Keeping the Lights On During Migration

- An Anti-Corruption Layer (coined by Eric Evans) enables two services to speak in two different models and can translate between the models
  - Allows the monolith's model to be converted into FHIR, which is the canonical model used by the microservices
  - Would convert FHIR messages back to the monolith's model to keep the monolith in sync during a phased migration

Keeping the Lights On During Migration



Getting Data from the Monolith

- Once service is separated, it needs data
- Two major ways:
  - Change data capture/reading from database
  - Putting existing domain entities on a kafka/kineses/etc... queue

Getting Data from the Monolith – Change Data Capture

- Method of finding and reporting out data changes (inserts, updates, deleted)
- Many databases have systems (LogMiner for Oracle, SQL Server CDC)
- Challenges of using CDC to convert to FHIR:
  - If tables don't align with FHIR concepts, recreating can be difficult
    - Can publish only changes to a new FHIR repo - when is data "complete"
    - Can link together tables and recreate models – bad DB design can make this difficult
  - Must wait for data to get to DB, go through CDC, get converted
- Pros: work-flow agnostic – won't miss out on data

Getting Data from the Monolith – Domain Entity Queue

- Domain entities in monolith published to I.E. Kafka stream, ETL to convert domain entities to FHIR
- Benefits:
  - Domain entities (hopefully) closer to FHIR concepts
  - Quicker data access
- Drawbacks:
  - Must know where to intercept models to publish
  - Might get out of sync with DB if failures in either

Conclusion

- Monoliths don't scale long term
- Microservices offer teams the opportunity to build focused services with the right tools
- FHIR can be used as a canonical model for communication across microservices
- Moving data from the monolith to microservices involves capturing data as well as keep the two in sync

Questions?