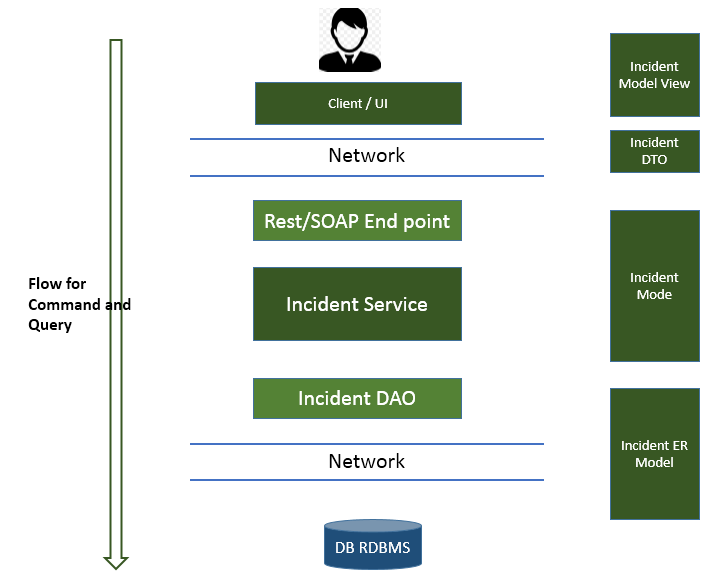
# Micro-service design

## Command Query Responsibility Separation (CQRS) pattern

Here we move from traditional approach of using the same architecture for writing (command) and reading (query) towards event sourcing and why we should separate the read and write and benefits of achieving it and also the challenges.

### Evolving towards event sourcing

Tradition approach of read and write in a typical monolithic application



### Observations:

Below could be some drawbacks to the above architecture

1. We read and write data from same layer/flow.
2. Same model, business service or DAO/DTOs are used for read and write (which might not be always fit to)
3. We use the same deployment model. I.e. reader and writer deployed at the same time.
4. We use the same data store to read and write data
5. We always get the final state of data (after a multiple updates). But we will never know how we reached there. Or what was the state of that particular record on some past business date. (Because we don’t have transaction log always).
6. We can’t scale read and write independently. If read needs more resource that write.
7. No data history or snapshot.
8. Because of using the same data base we need to do read and write in the same data object. Where to improve read performance we may put multiple indexes on database tables where the same index will kill the write performance.
9. This approach tends to create huge monolithic architecture.

One architecture which approaches these kind of challenges is **Event Sourcing.**

## Event Sourcing

Event sourcing is an architectural pattern where the state of and application is being determined by **series of events**. State is computed by series of events.