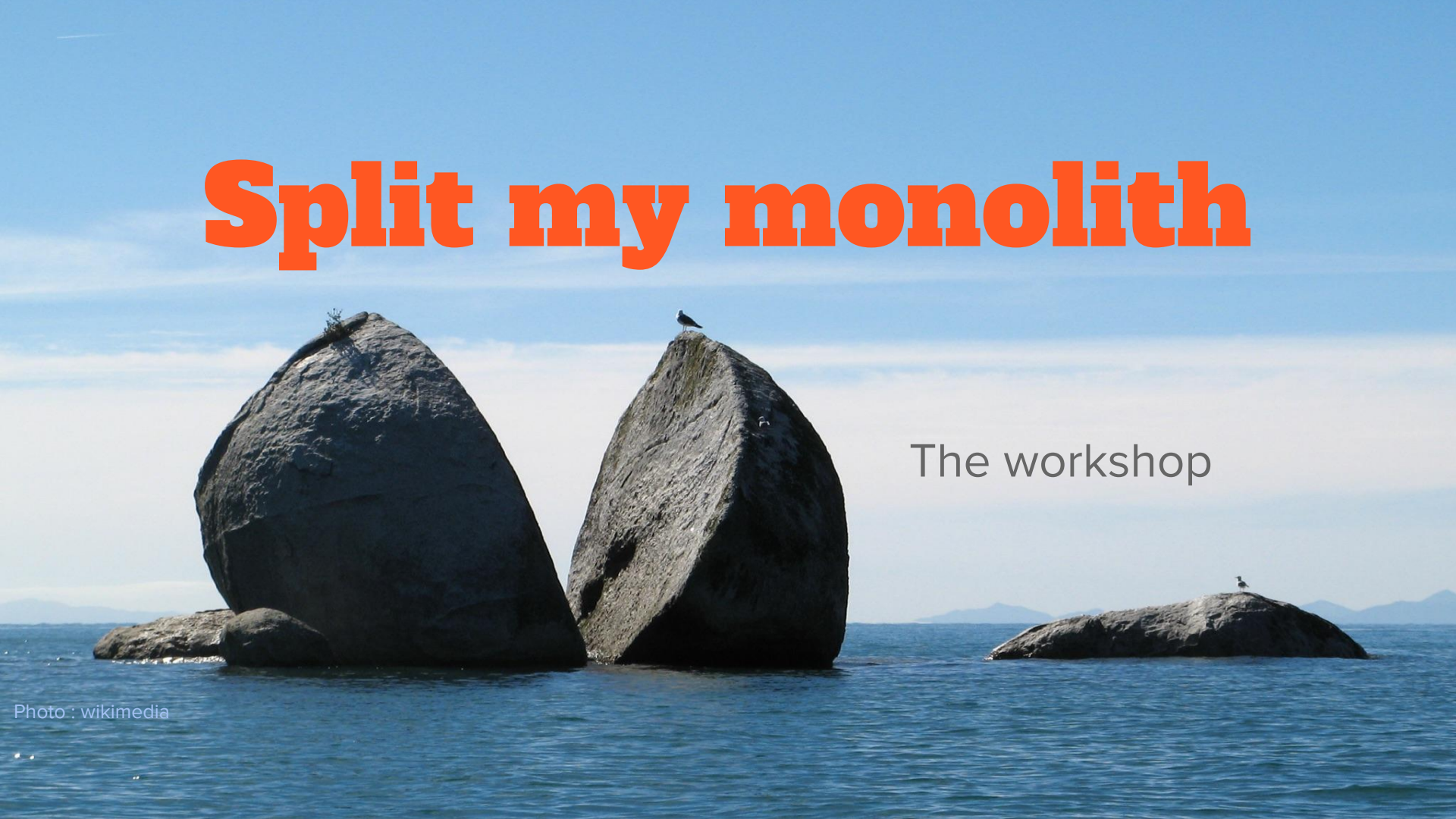
Split my monolith



The workshop

Photo : wikimedia

HACK YOUR JOB

Superindep.fr

LyonTechHub

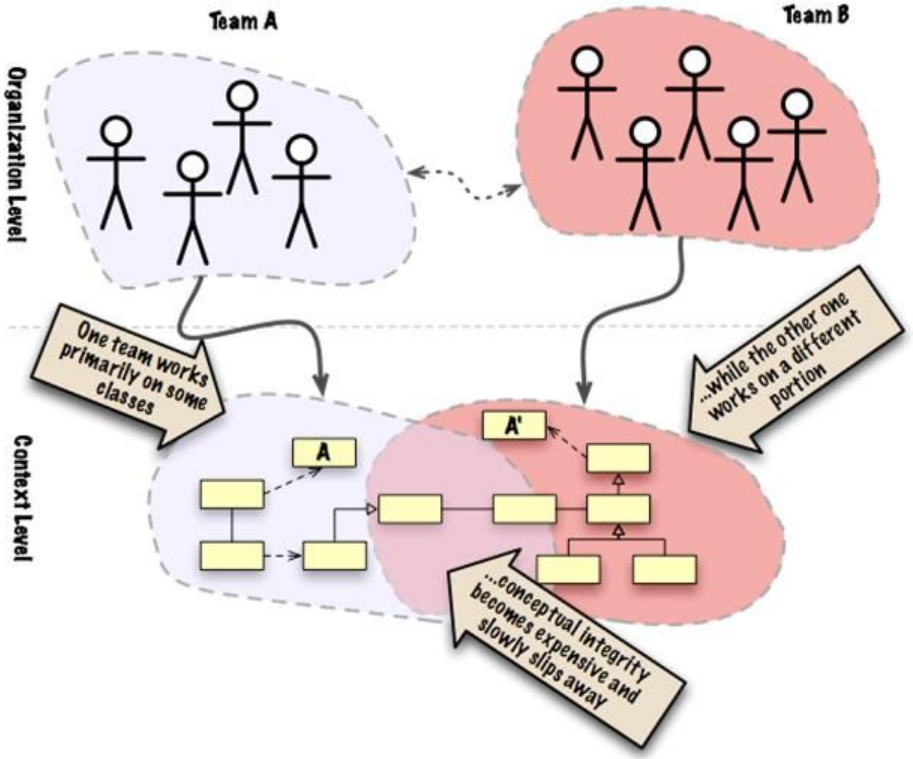
@florentpellet

@clem_bouillier

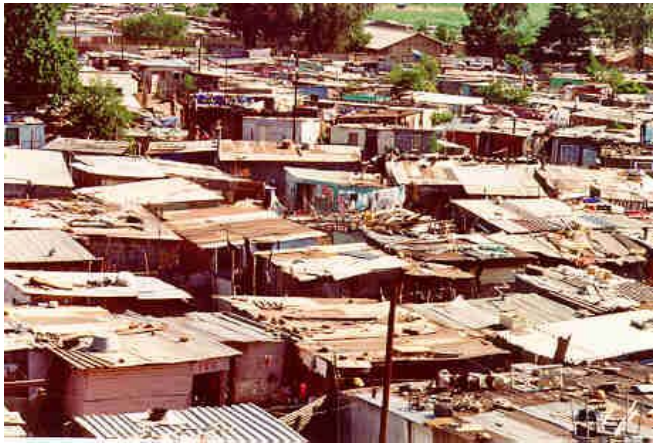
@johan_alps

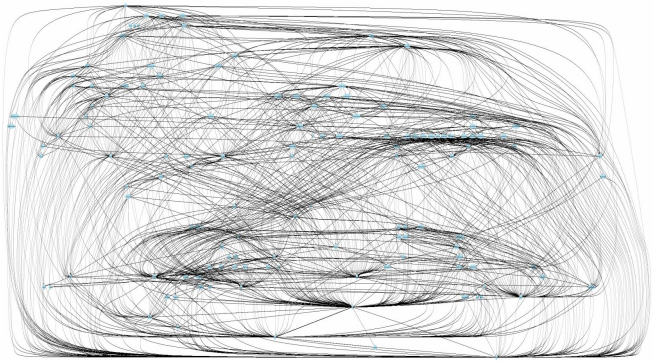
Split?

System resilience / Team scalability



Manage complexity





How?

We all have seen failed modularisations.

Our hypothesis is that it's often about the database

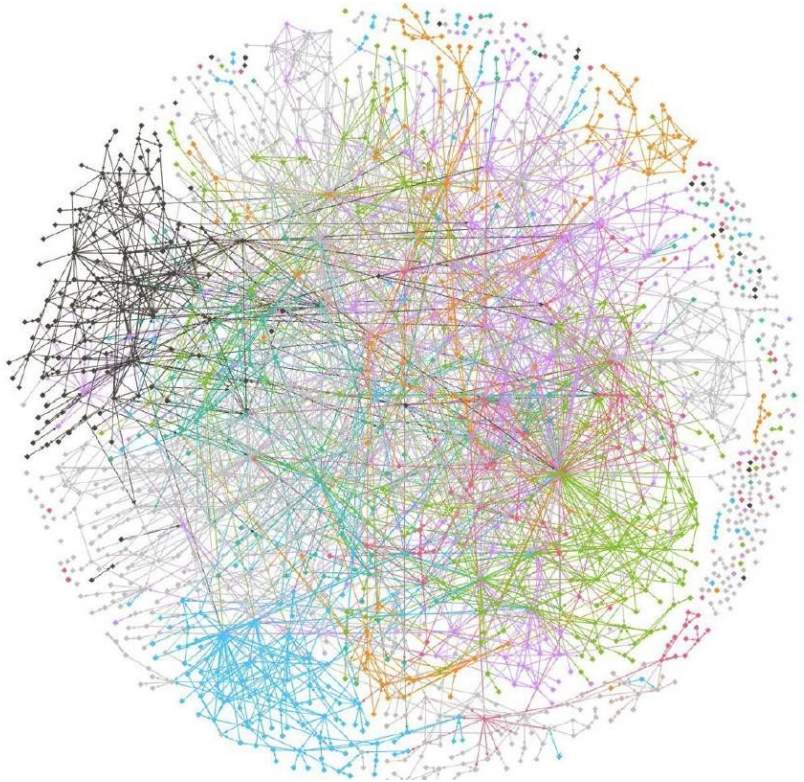
https://twitter.com/mathiasverraes/status/711168935798902785?lang=en

Why is a shared DB bad?

- Is it a clear interface?
- Where is the code that writes to / read from a column?
- Independent deployment?
- Can we evolve the DB schema?

Micro-services ?

Availability



99.999% uptime ?

Ex 99%^20 = 81%

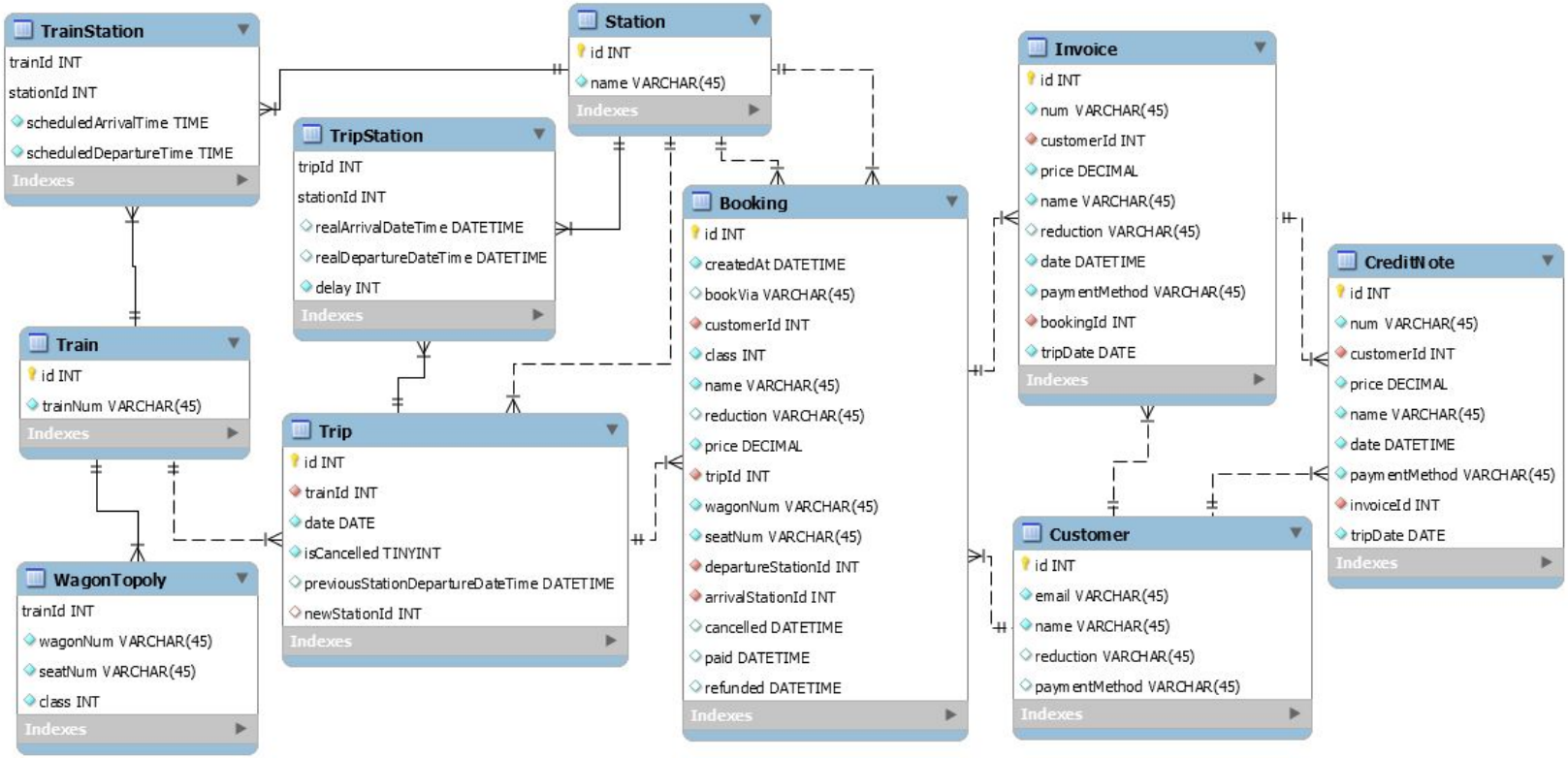
Our Domain for today

Train management

- Train routing:
  - Specify itinerary between two stations
  - Specify train topologies
  - Schedule train

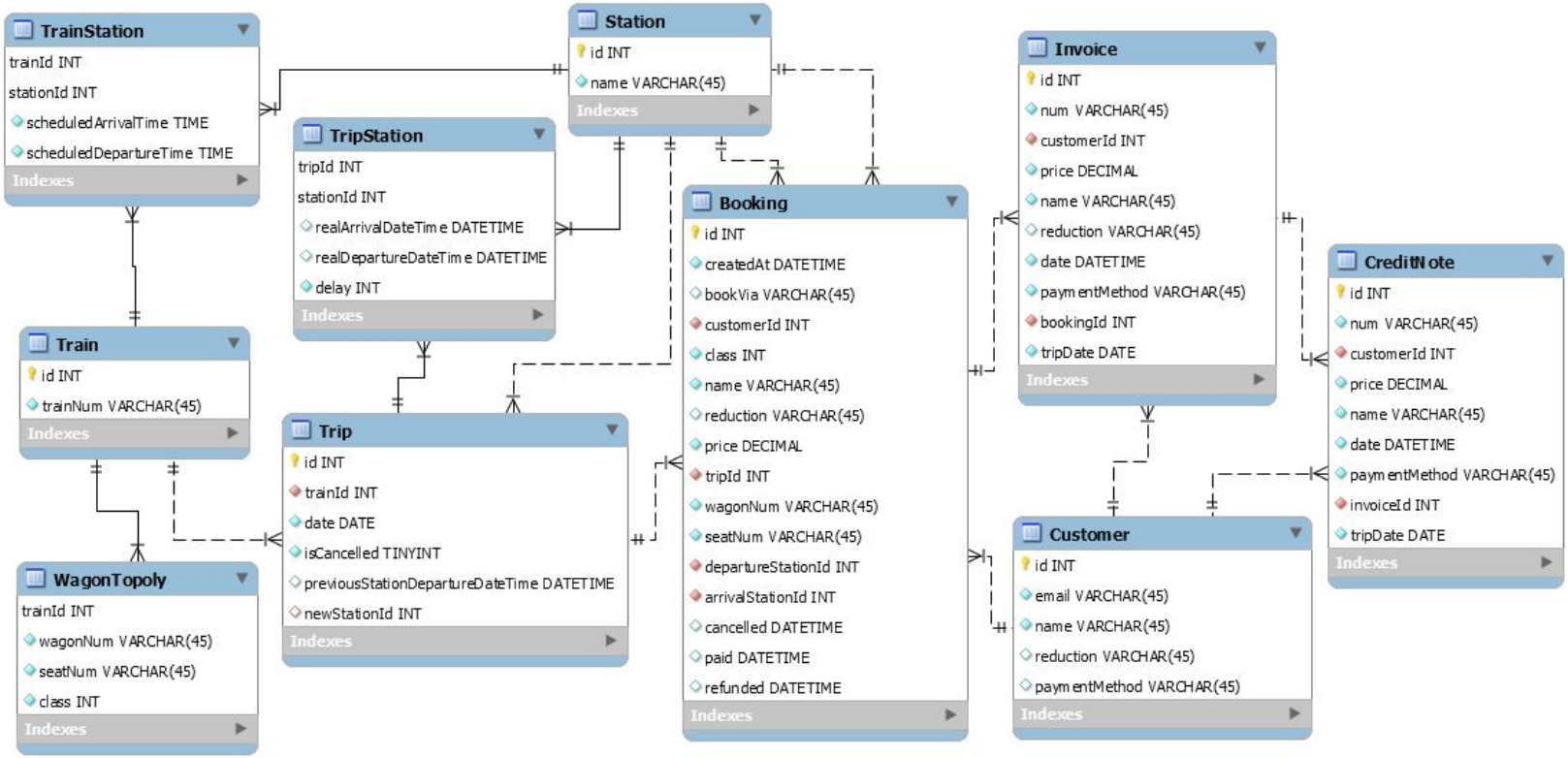
  => a train makes the same trip every day
- Booking:
  - Allows our customers to book a train
- Traffic info:
  - Track train on a trip
  - Detect delay
- Invoicing:
  - Generate invoices and credits notes

Database schema



Exercice 1: Identify contexts in data model

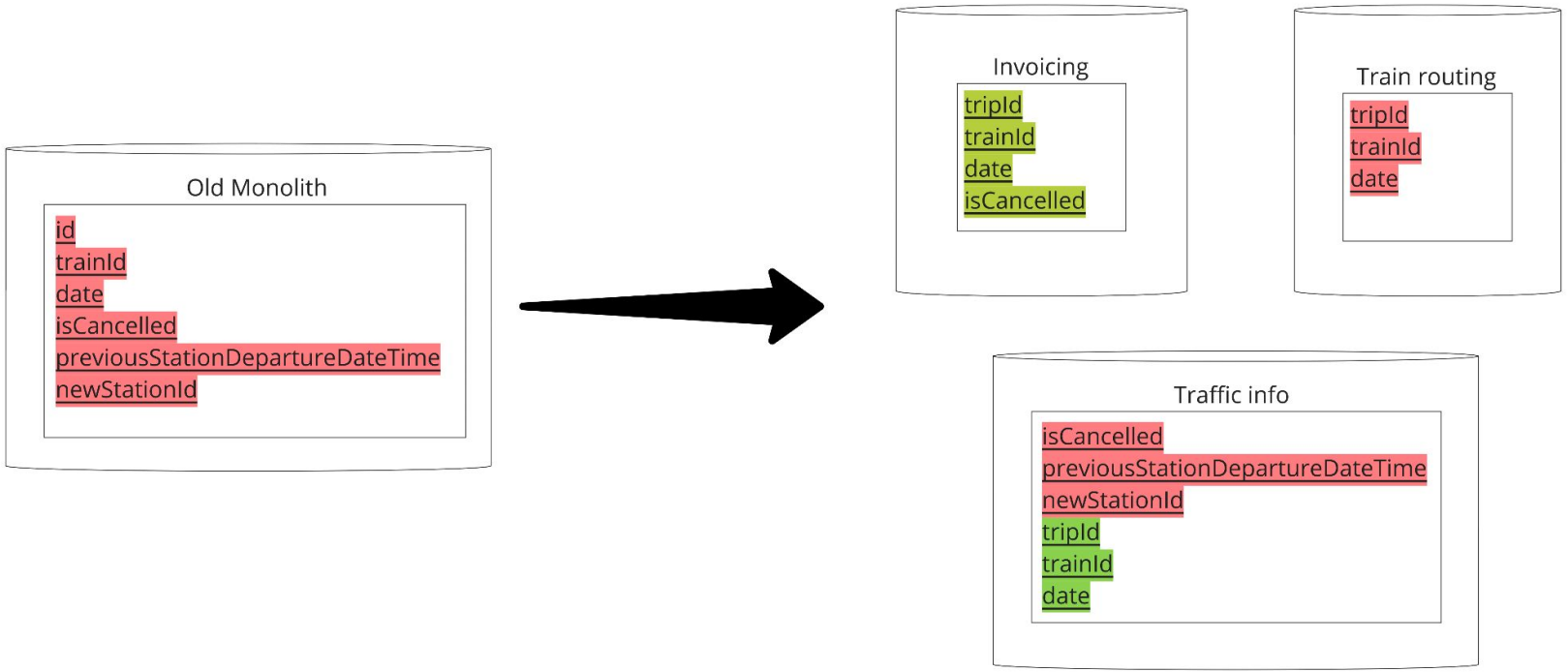
Split database



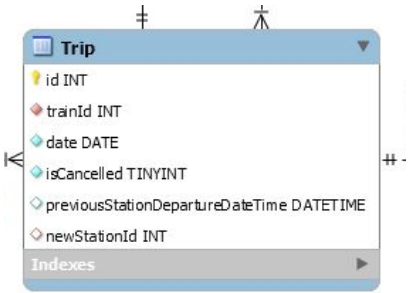
Ideal split of Trip table

Read data

Written data

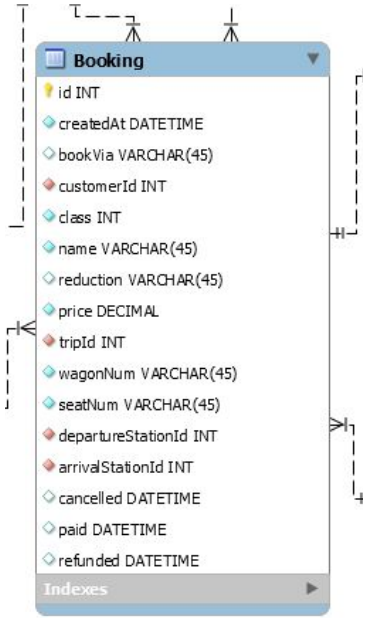


Example: Table Trip

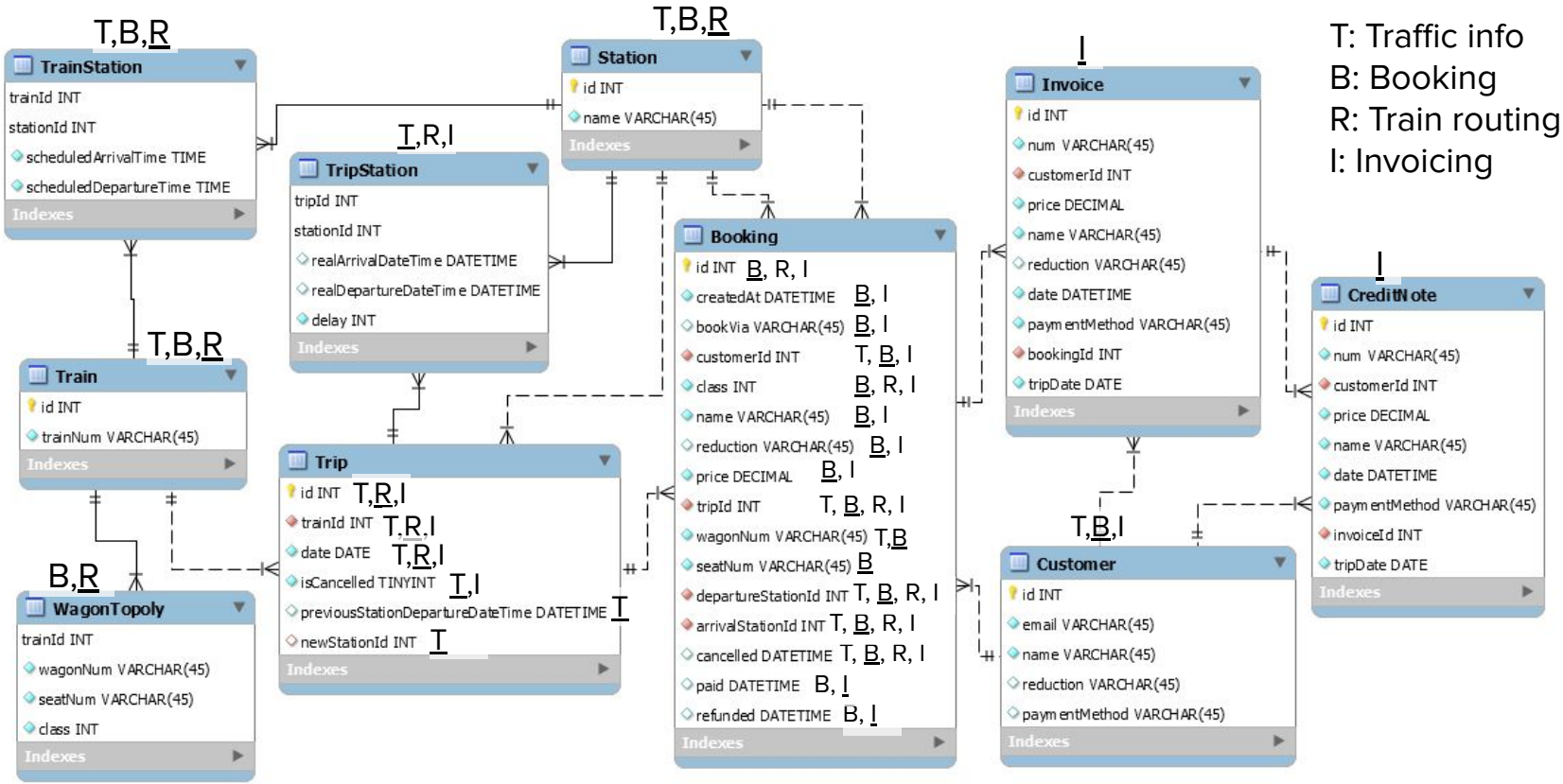


	Traffic Info	Booking	Train routing	Invoicing
Id	R		W	R
TrainId	R		W	R
Date	R		W	R
IsCancelled	W			R
Previous Station Departure DateTime	W			
NewStationId	W			

Exercise: Split the Booking table

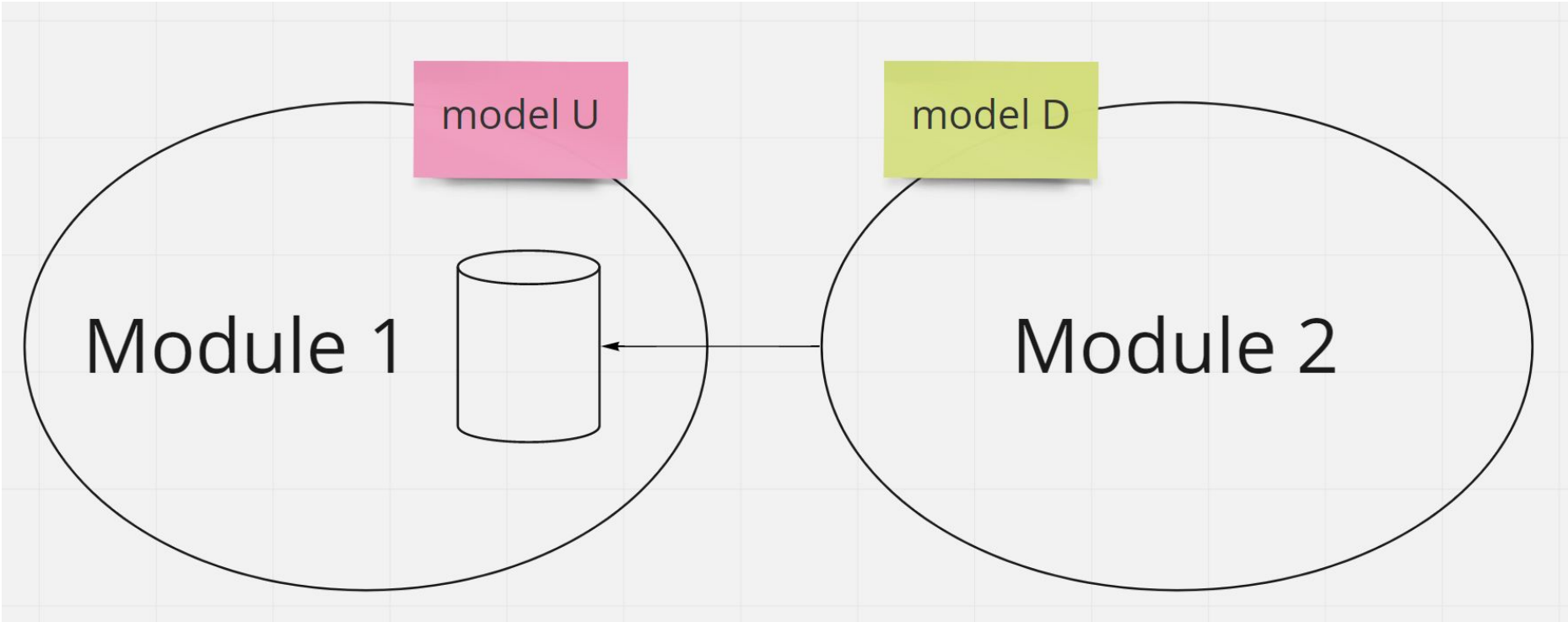


Which domain depends on which data?

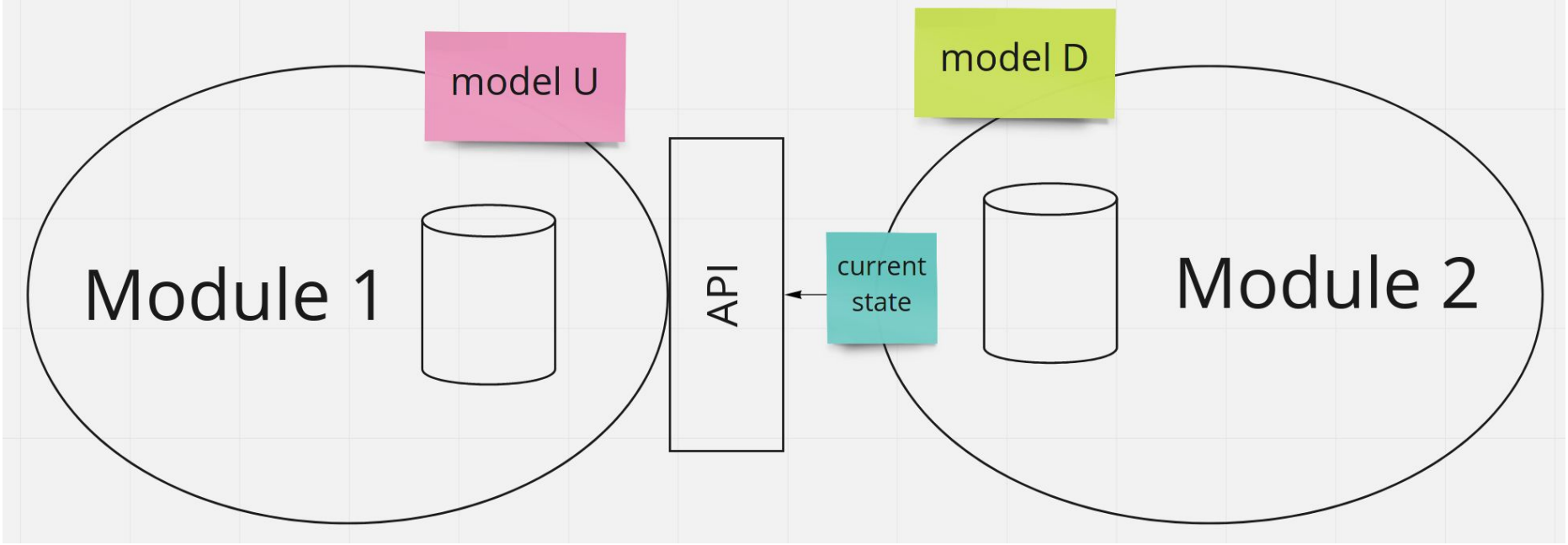


How?

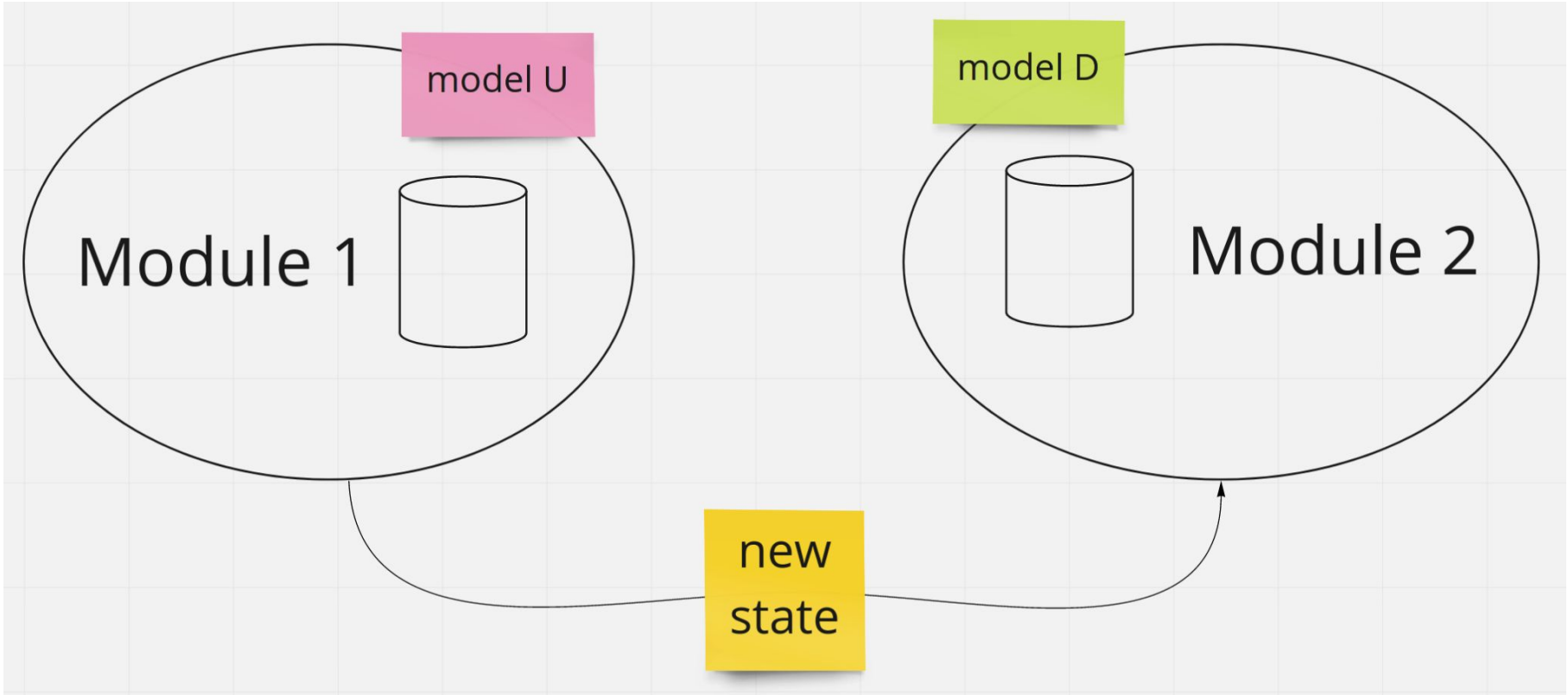
Bubble context



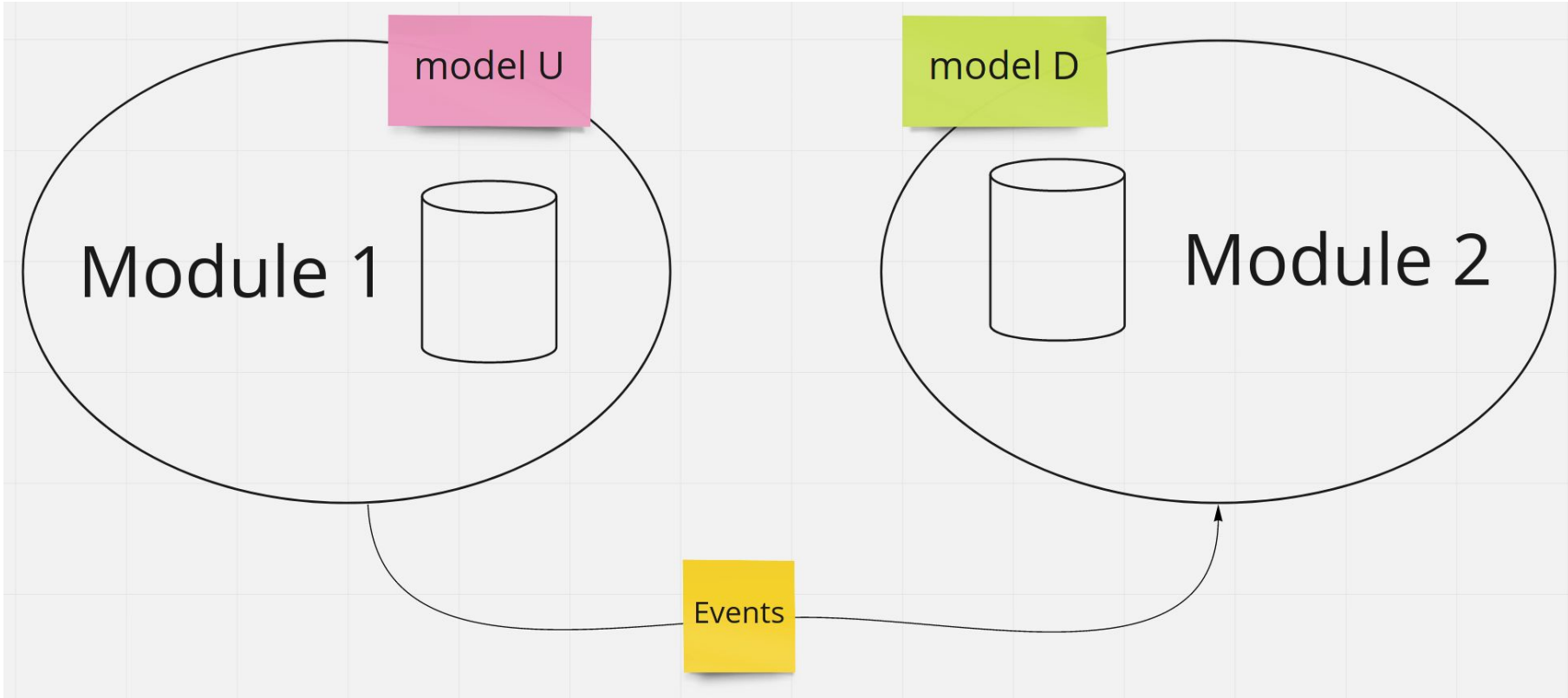
Autonomous bubble - synchronous



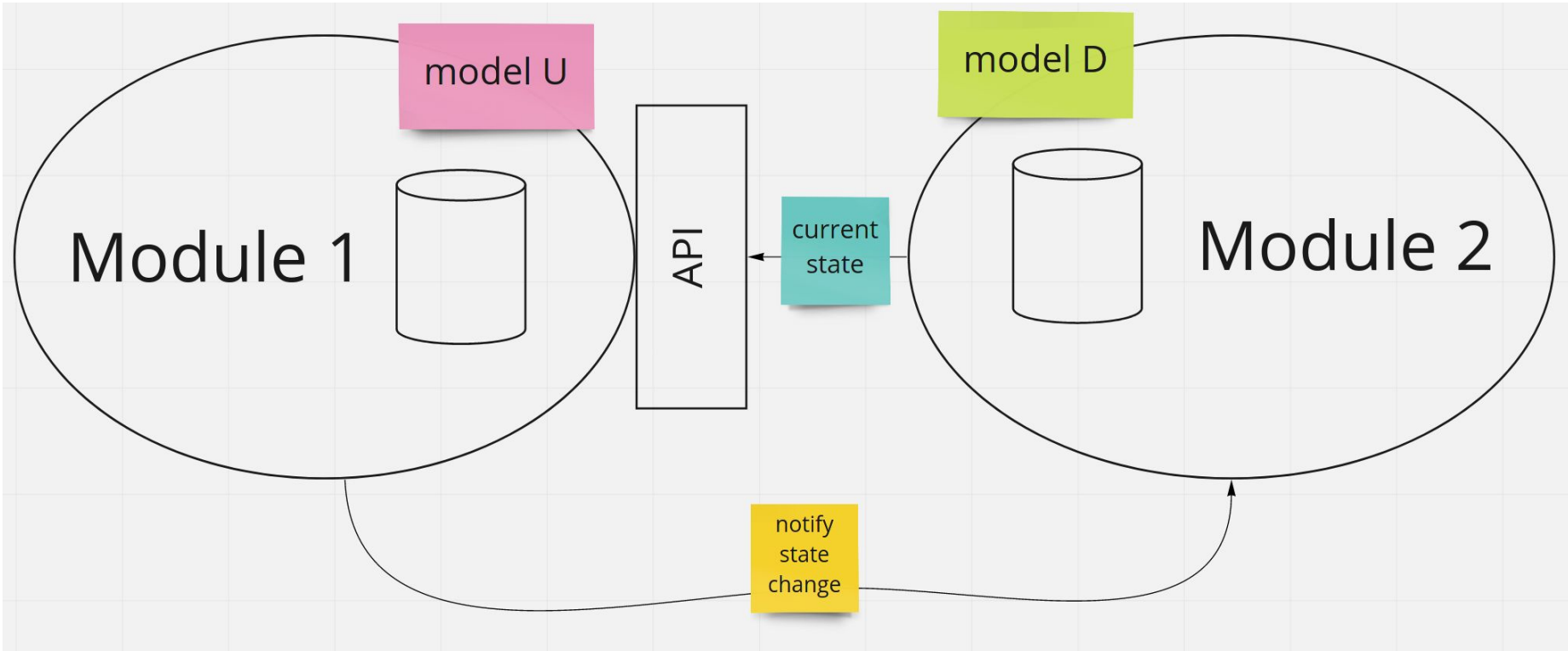
Autonomous bubble - state stream



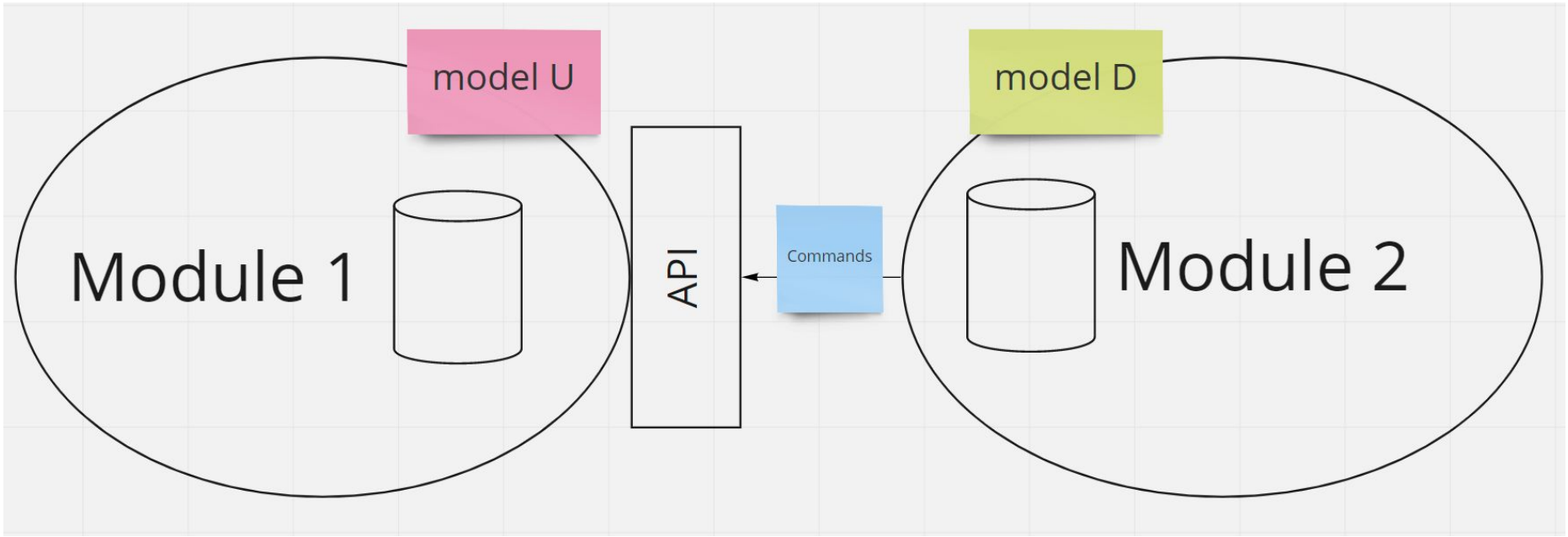
Autonomous bubble - events



Autonomous bubble - notif. stream



Exposing legacy assets as services



Exercices

Scenario 1

Today is the day!
Everything is flooded!
Panic on board!

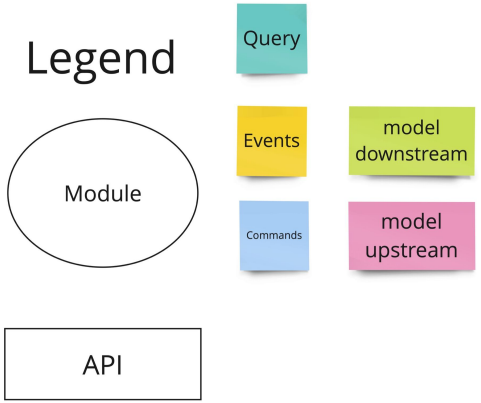
Lots of late trains, we need an immediate solution.

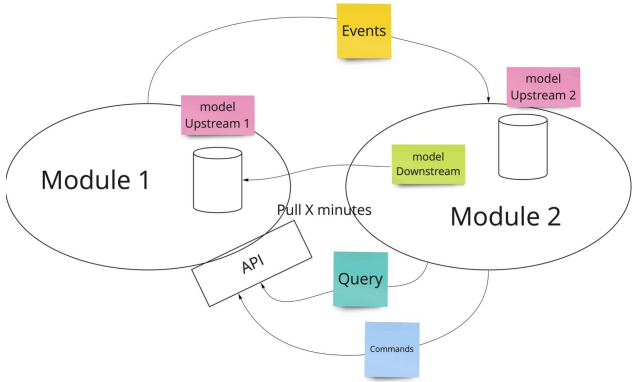
Deadline: **Yesterday**

Scenario 1

Many technical problems occur regularly in recent days

Deadline: **Yesterday**





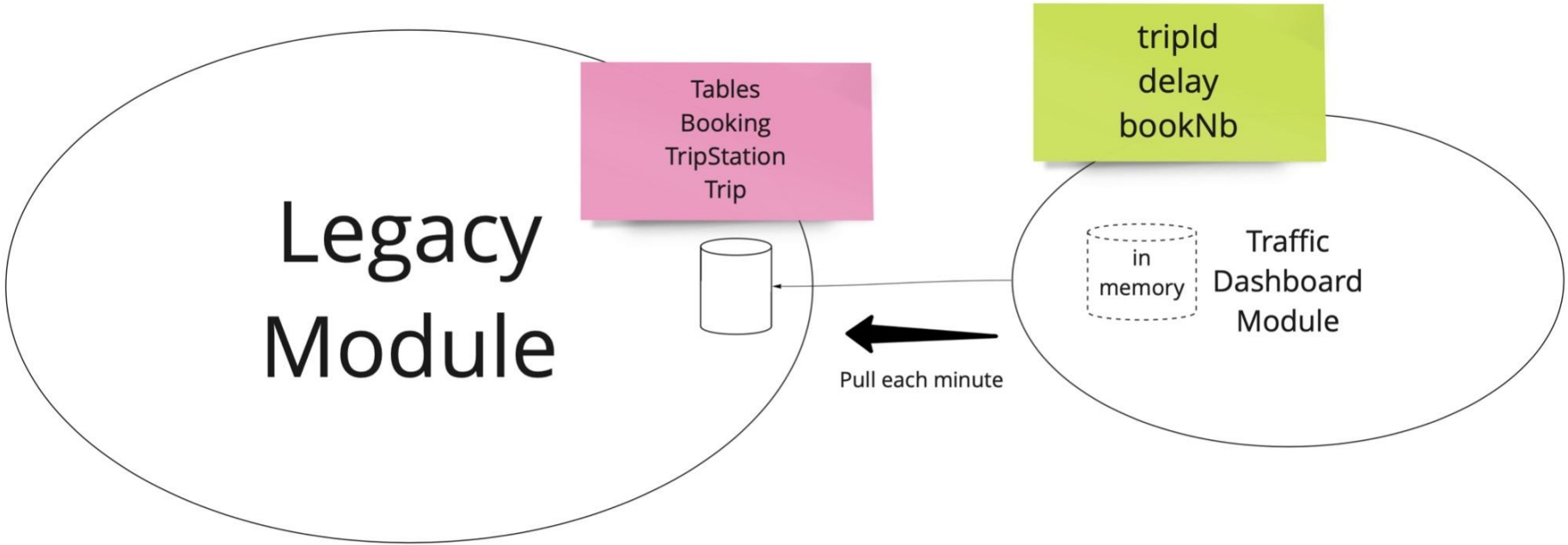
We want to quickly set up a dashboard indicating in "real time" the number of people impacted.

We cannot afford to re-deploy the legacy

Expected :

- Schema with modules, DB, API, messages, models (use the legend)
- Messages frequency / data refresh rate
- Make explicit any isolation
- Detail data for each post-its

Scenario 1 - solution



Scenario 2

The storm is over.

The dashboard made it a lot easier to prioritize various incidents. We'll need a solid architecture to build upon.

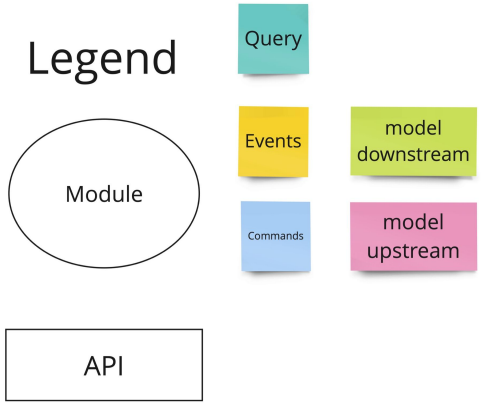
Deadline: **Not defined**

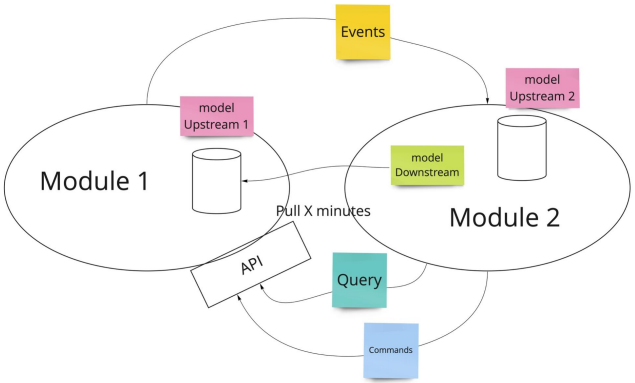
Scenario 2

The dashboard made it a lot easier to prioritize various incidents. We'll need a solid architecture to build upon.

How would you do that given enough time?

Deadline: **Not defined** ⛱

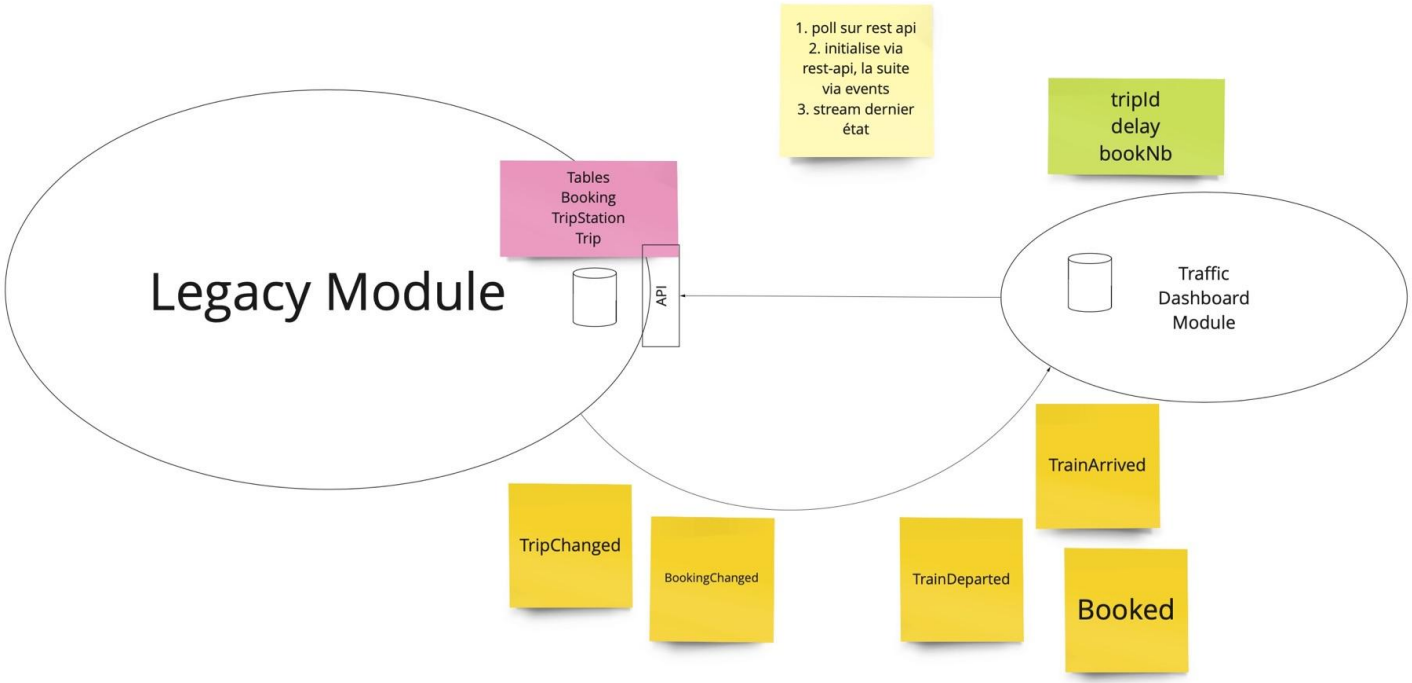




Expected :

- Schema with modules, DB, API, messages, models (use the legend)
- Messages frequency / data refresh rate
- Make explicit any isolation
- Detail data for each post-its

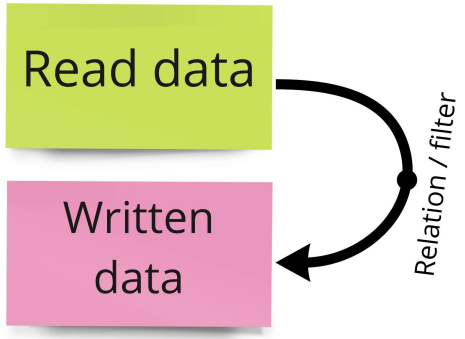
Scenario 2 - a solution



Scenario 2 - data

Many technical problems occur regularly in recent days

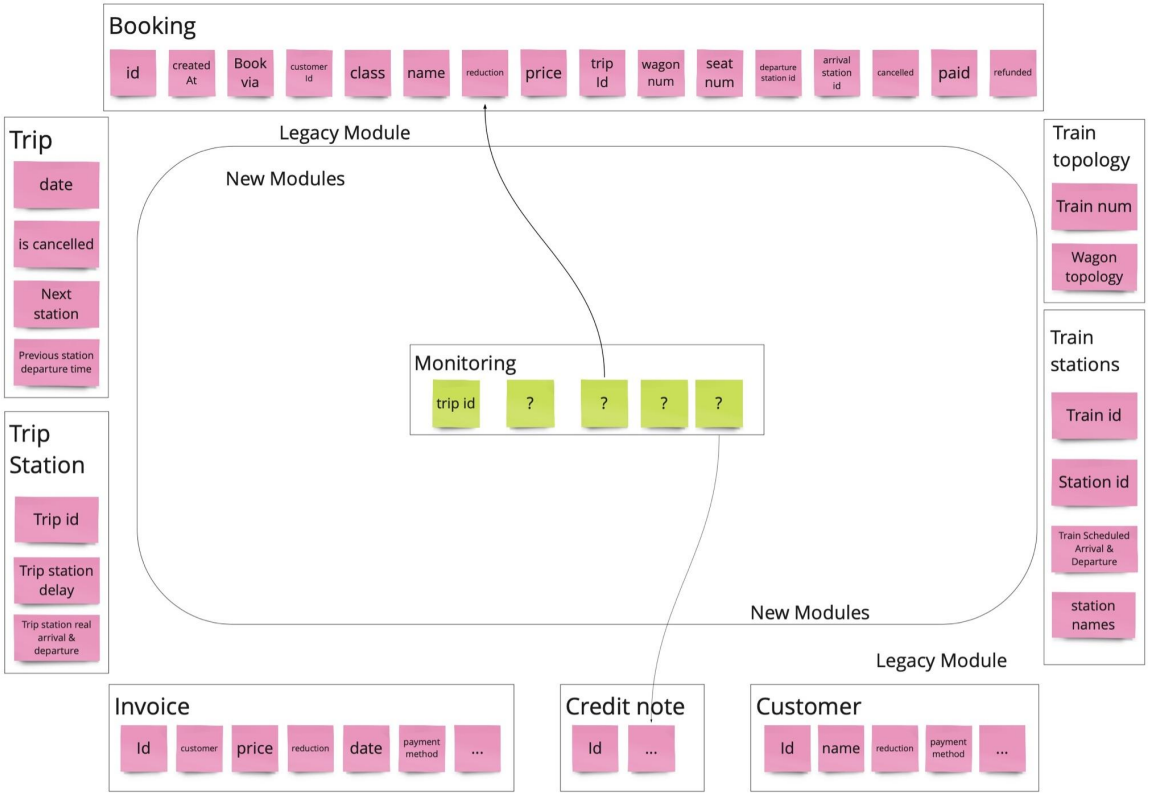
Try another representation based on data model.



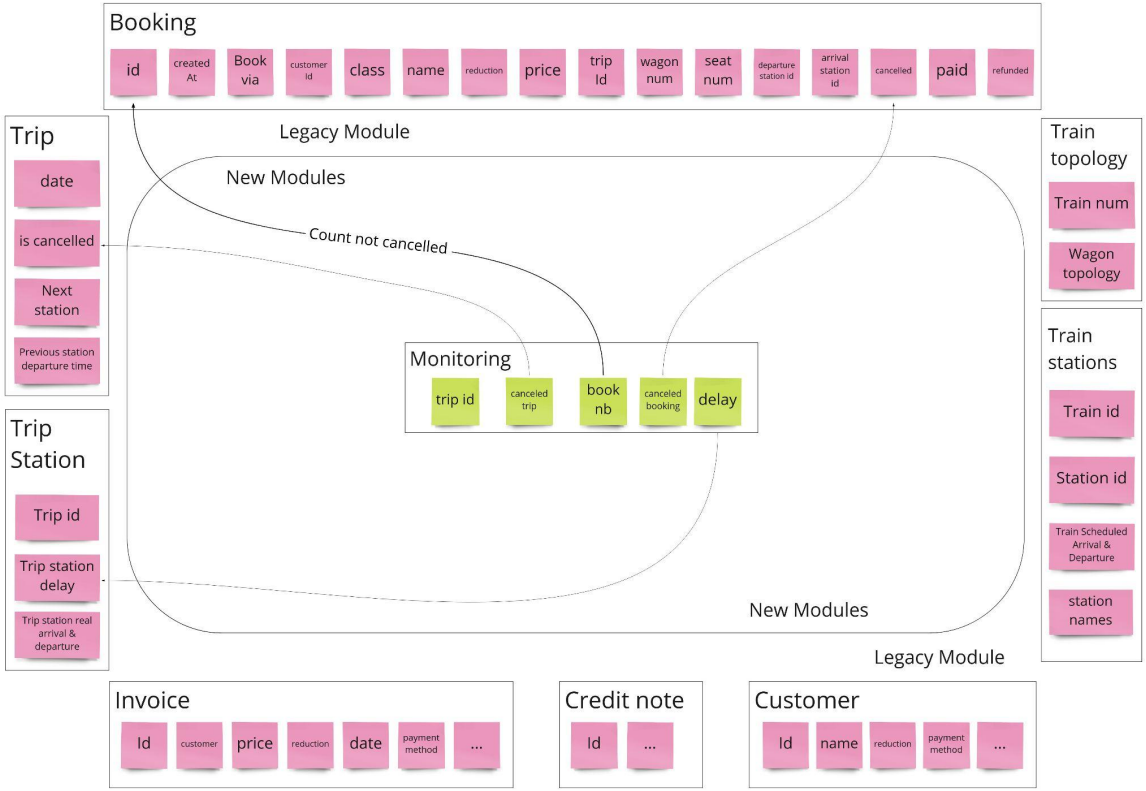
Expected :

- Which data is duplicated
- Where does it come from
- Who is the
  - Writer
  - Reader

Scenario 2 - data, new point of view



Scenario 2 - data - solution



Scenario 3

We have a new “nonon” offer (nogo train).

Improve business opportunity, but success is not guaranteed.

Need to test without impact legacy and reuse maximum functionality to reduce costs.

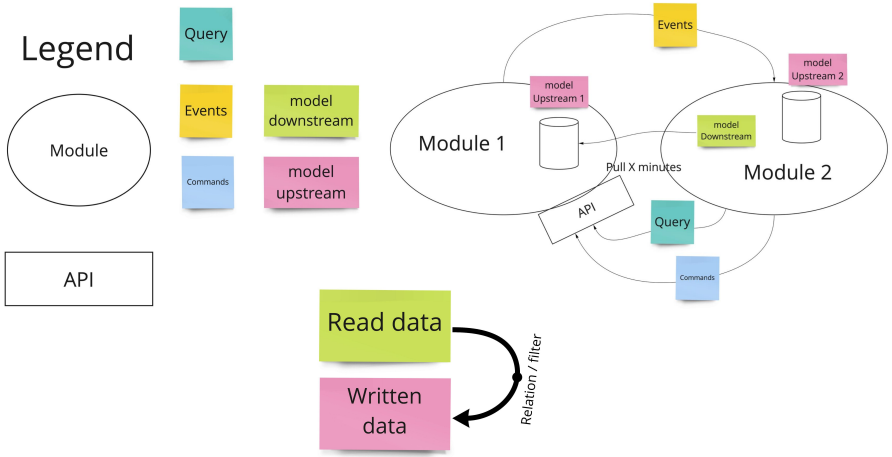
Deadline: **Month**

Scenario 3

We have a new “nonon” offer (nogo train). How to integrate this offer with the current system?

- **New commercial offer** : other reduction cards, options to buy (**wifi**, breakfast, cancellation insurance)
- **Traffic info** and **seat reservation** remain in the **legacy**
- A train is either **completely “nonon”** or legacy offer

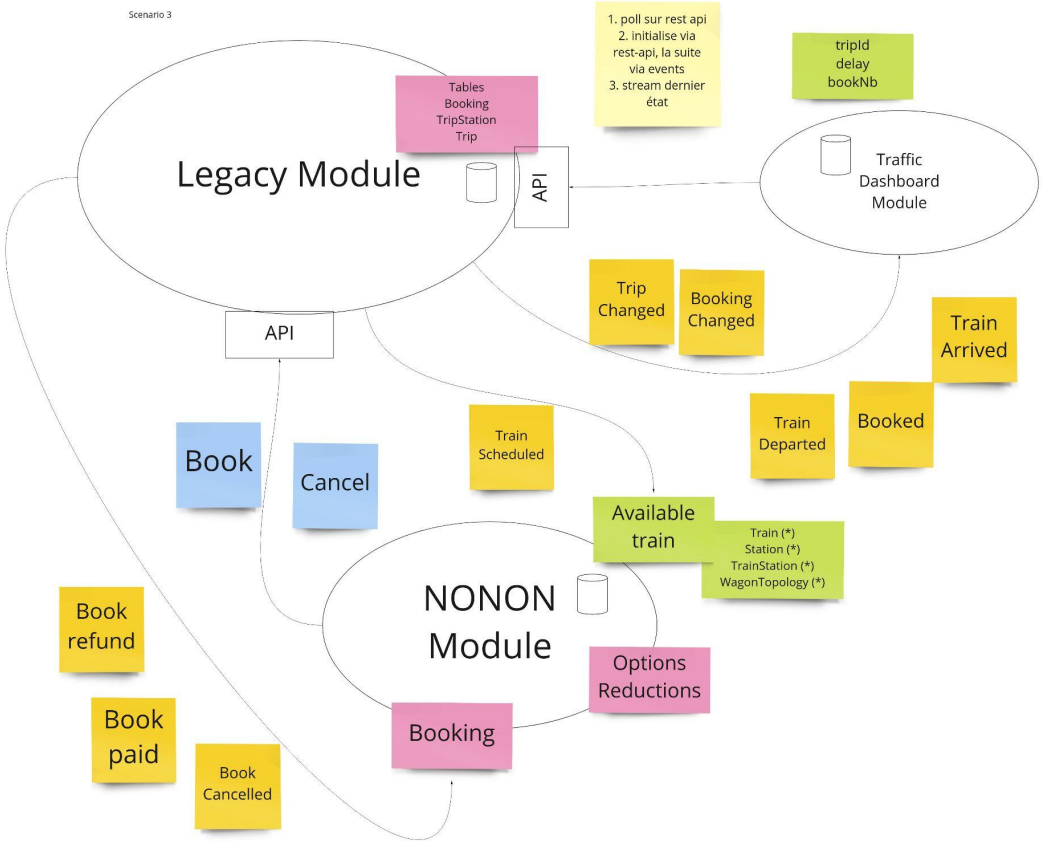
Deadline: **Month**



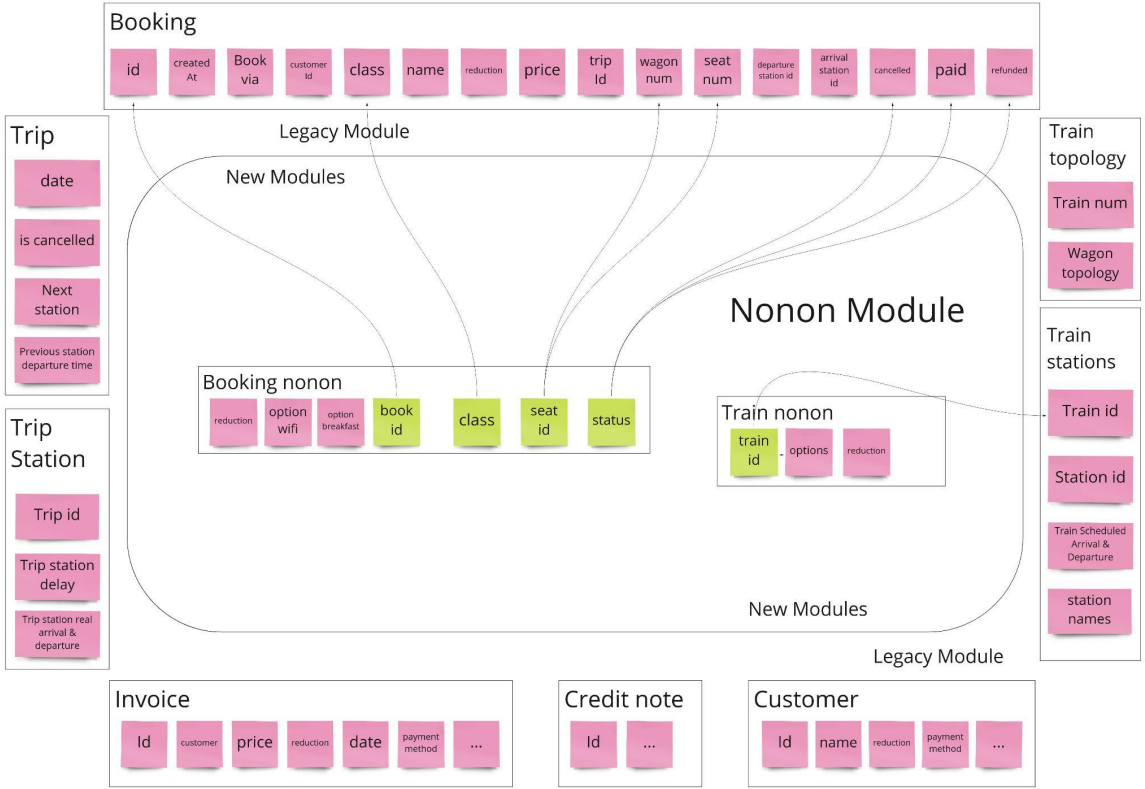
Expected (context and data) :

- Schema with modules, DB, API, messages, models (use the legend)
- Messages frequency / data refresh rate
- Make explicit any isolation
- Which data is duplicated
- Where does it come from

Scenario 3 - solution



Scenario 3 - solution

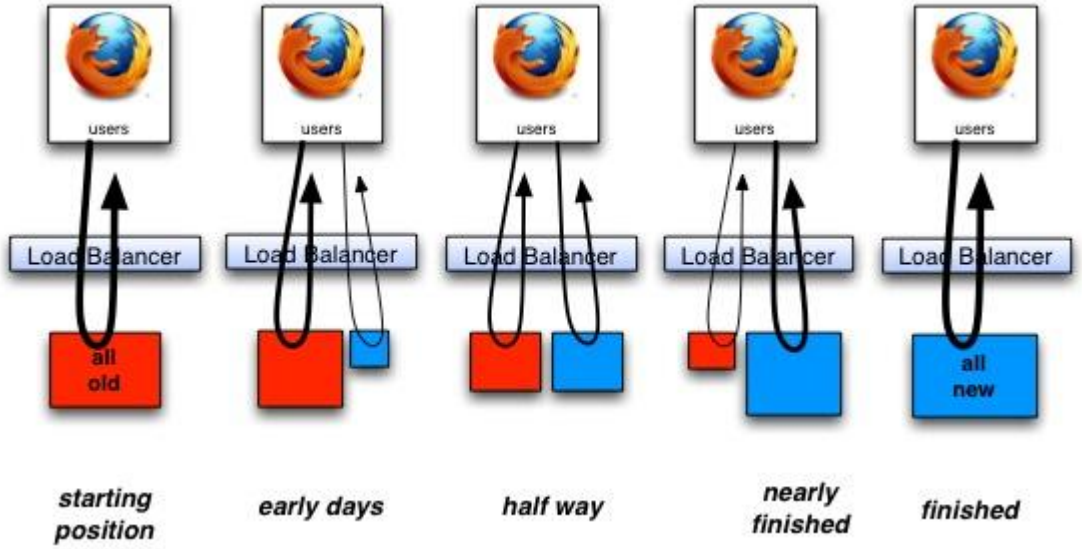


Final Notes

In-process sync event (method call)

Strangler fig pattern

Strangler fig pattern



paulhammant.com

Eventual Consistency is key

Most things can be decoupled if we allow a delay in consistency (1s, 1min, ...)

Ex : The csv exports

CALM theorem

- State will ALWAYS be consistent if we don't delete information

Counter examples:

- Set difference
- Multiple processes doing increment

Database change with minimal interruption

1st deploy: Add the new table/database

2nd deploy: fill the new table with values. Check

3rd deploy: New code writing to the new table (but not reading). Possibly a migration script

4th deploy: New code or feature flip, to read from the new table

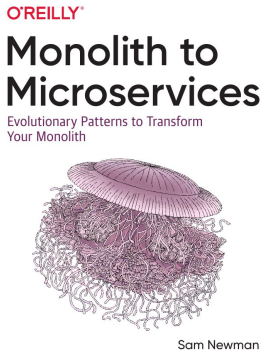
5th deploy: remove old columns/table and old code. (Possibly days later)

Questions

Bibliographie

[Getting started with DDD when surrounded by legacy systems]

De Eric Evans



[Monolith to Microservices]

De Sam Newman


La passion d'apprendre
formation.hackyourjob.com

Thank you !