API {1st} ARCHITECTURE

Why API Governance ?

API {1st} is not just a new tool. It is about opening up processes via a simple stateless interface so people can easily integrate with them.

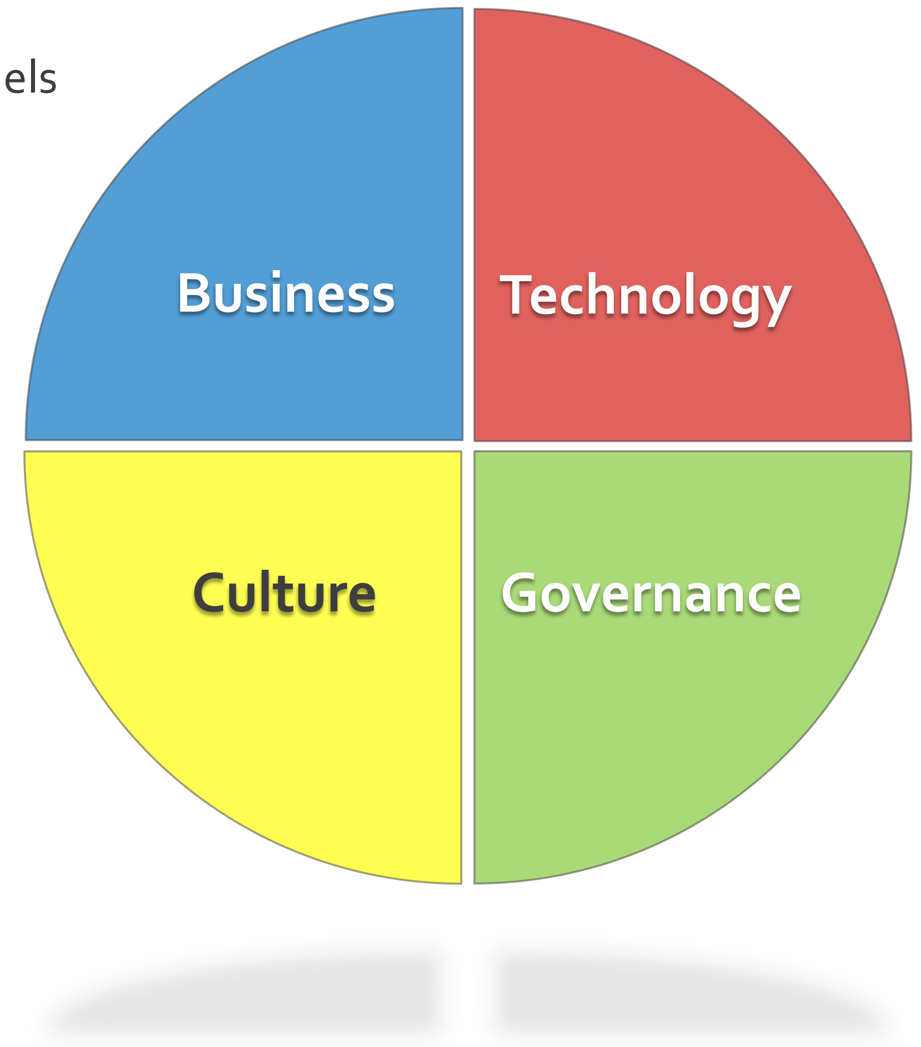
This only works when API's are simple, easy to understand, universal and consistent.

That is where API Governance comes in to play: To secure that we keep on the right track.

API {1st} Strategic Change / 360° view

Discover new Business Models
Target other Consumers
Digital Disruption

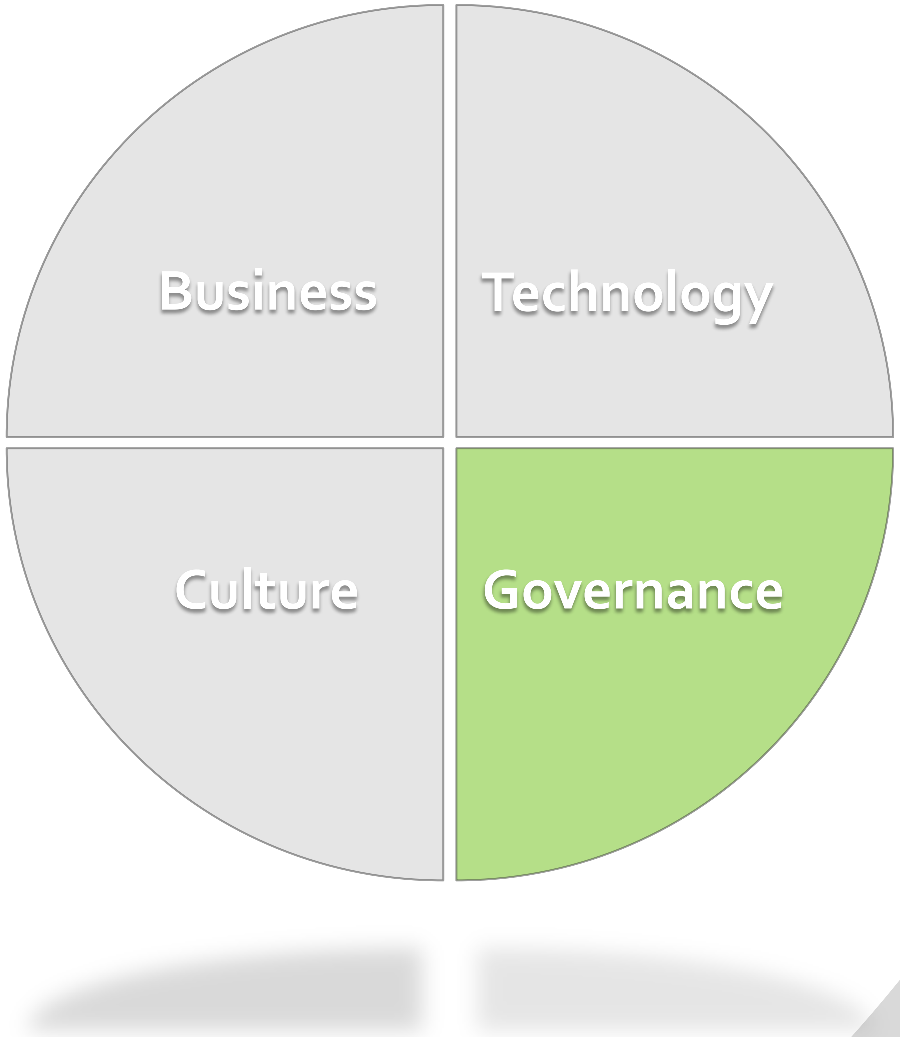
API Management Stack
Development Tools
Anti Patterns



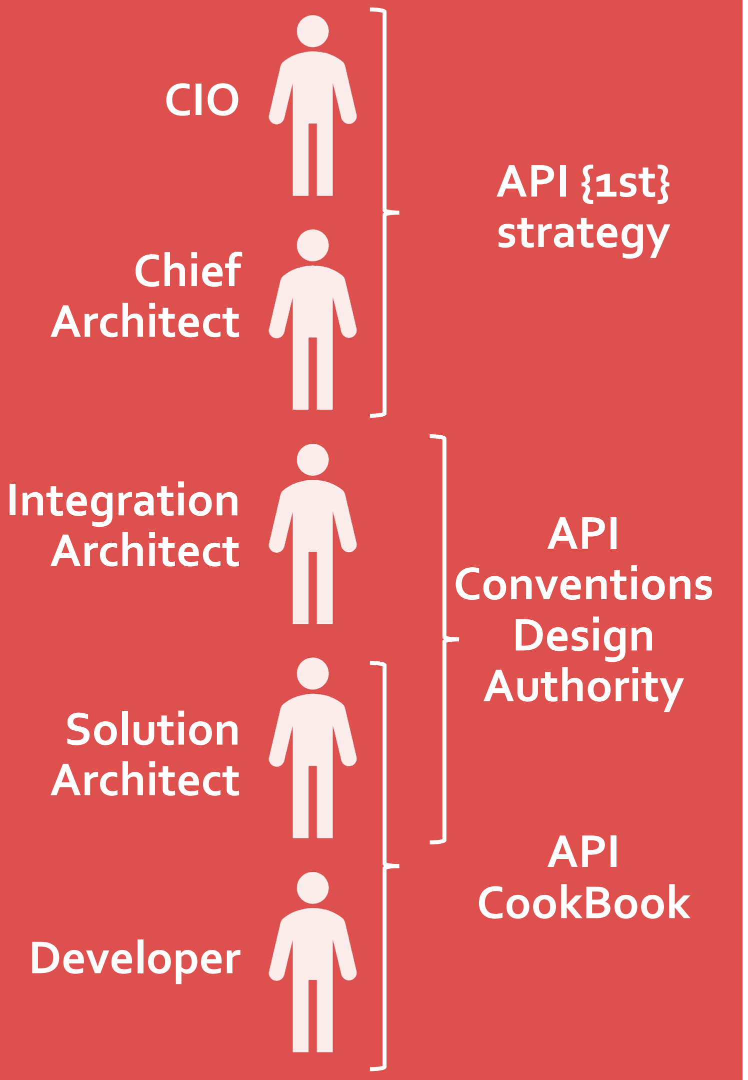
Developer Culture
Organizational Culture
API Maturity

Achieve Business Agility
Improve Return on Investment
Ensure Organizational alignment

AppyThings

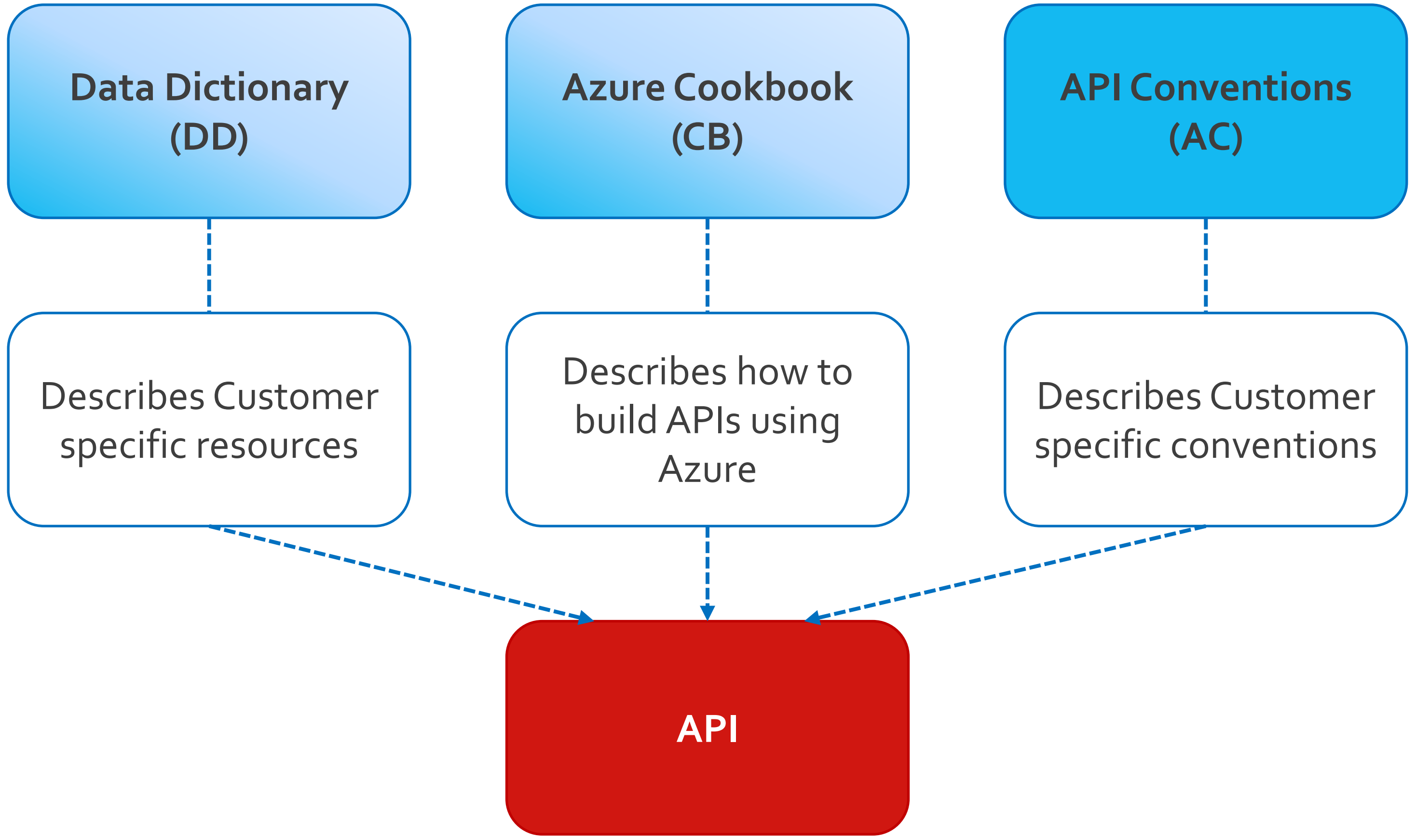


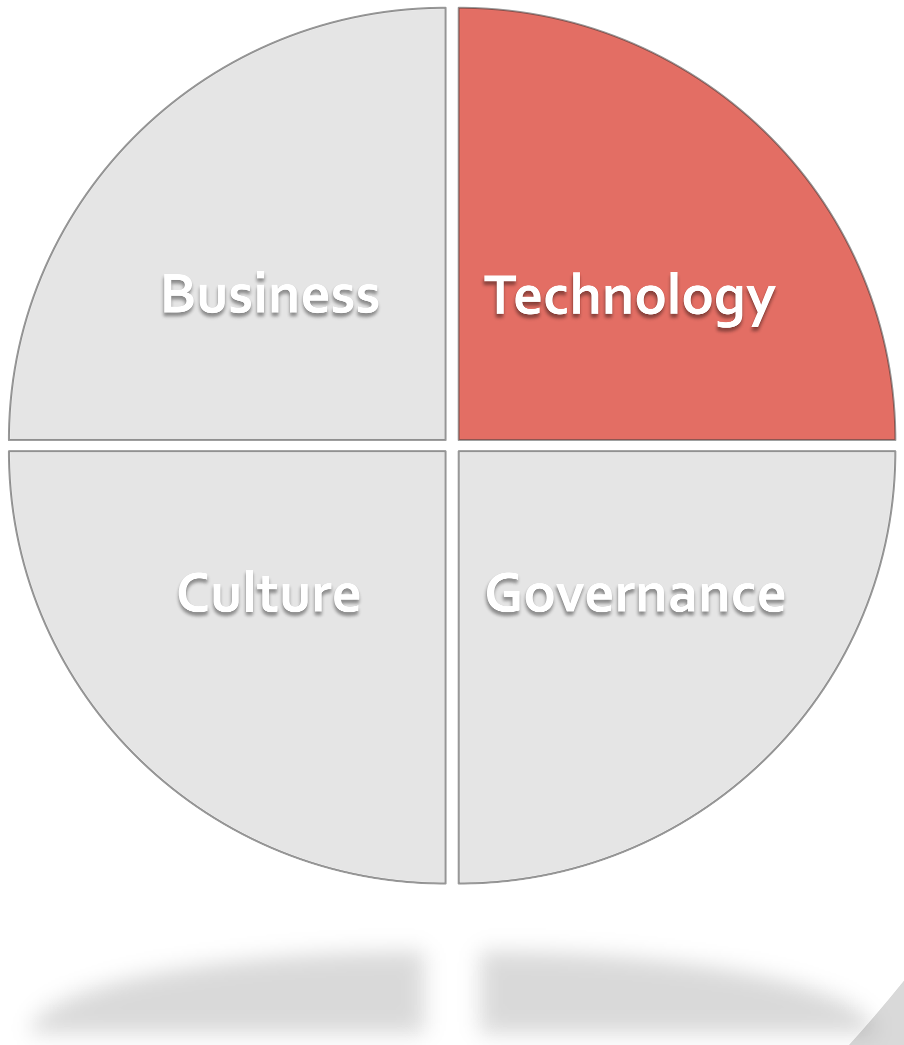
Governance



Position	Role	Comment
CIO	Sponsor	The CIO mandates the strategic change
Chief Architect	Sponsor	The Chief Architect is the evangelist for the API {1st} strategy.
Integration Architect	Design Authority	Responsible for the communication about Conventions, Cookbook, etc. within the organization
Solution Architect	Design Authority	Responsible for the exception handling and creation of new conventions / patterns.
Developer	API Team	Hands-on assistance to the Back-End teams for creating good API's

Building Blocks of API Governance


Data Dictionary (DD)
Azure Cookbook (CB)
API Conventions (AC)
Describes Customer specific resources
Describes how to build APIs using Azure
Describes Customer specific conventions
API



Technology

AppyThings

- **Design the target**
- Map from functional domain
- Decide on Root Resources
- Use IDs from other APIs
- Keep things simple
- Engage with the consumer
- Think consumers!
- High availability
- Inconvenience

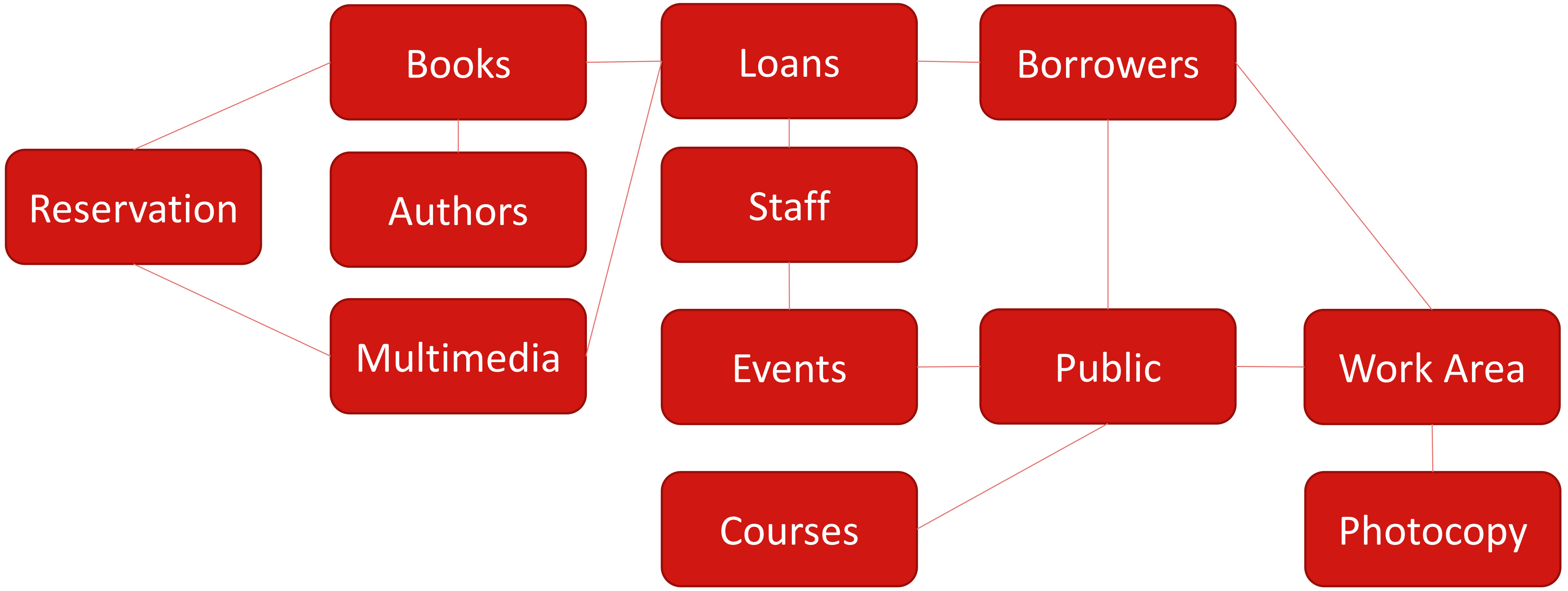
“Design the API to be the final, complete functionality set. Then if necessary, work backwards to what is achievable in the near term.”

- Design the target
- **Map from functional domain**
- Decide on Root Resources
- Use IDs from other APIs
- Keep things simple
- Engage with the consumer
- Think consumers!
- High availability
- Inconvenience

“Existing or potential implementation details shouldn’t influence the design of the API, use terminology and concepts from the functional domain.”

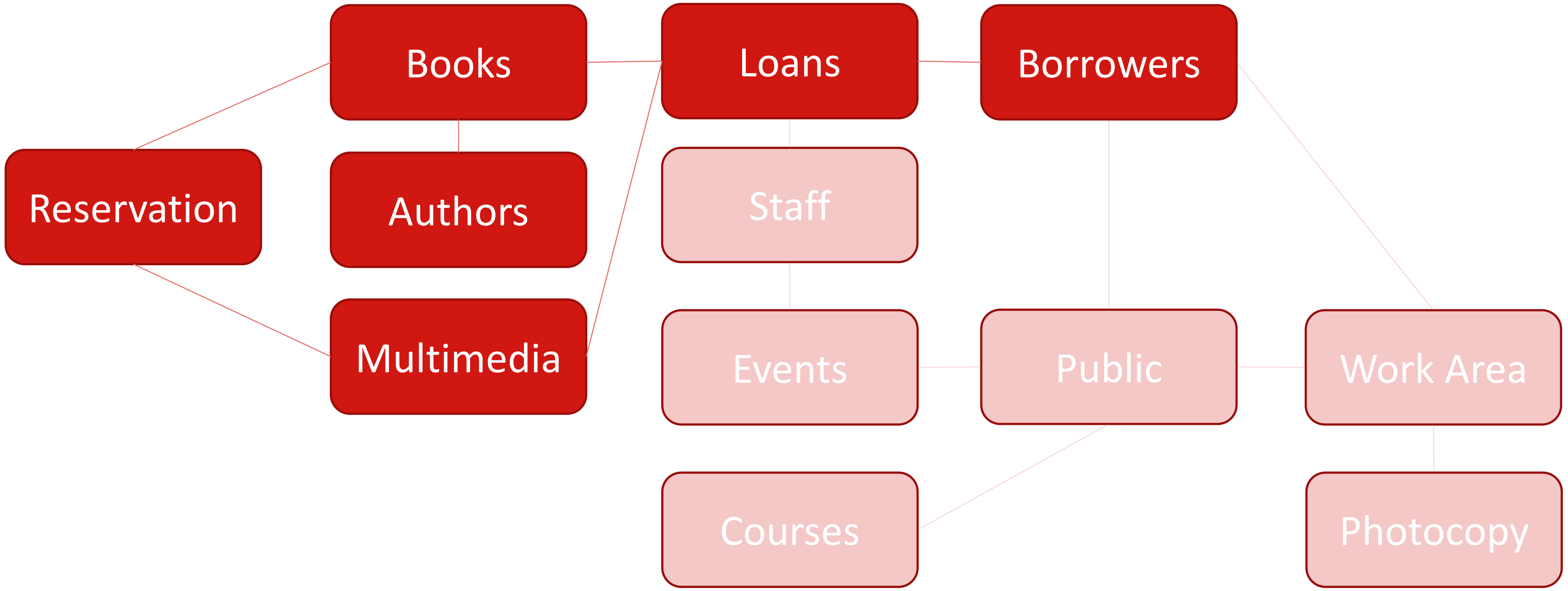
AppyThings

Example – Functional Domain (PUBLIC LIBRARY)



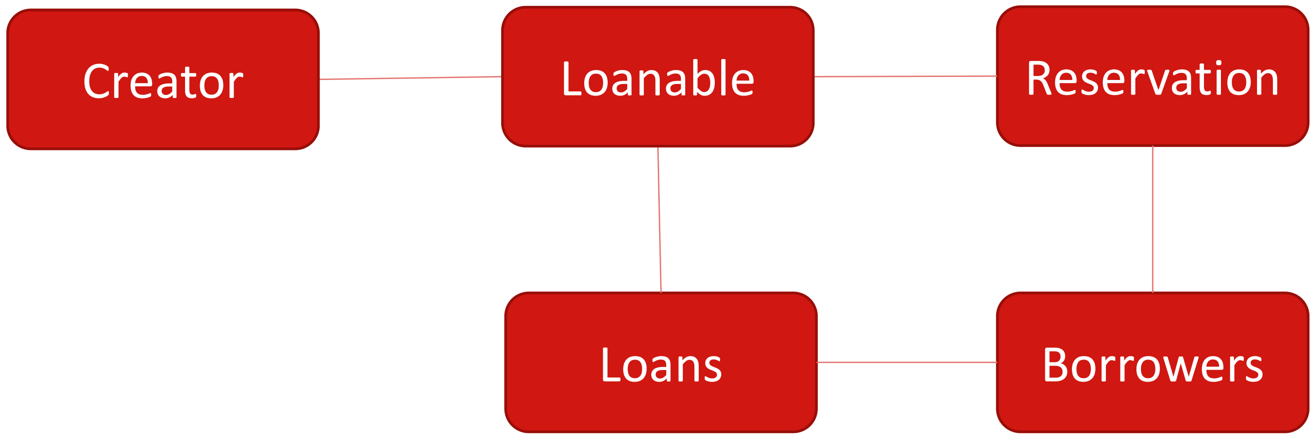
Paths:
- /v1/borrowers
- /v1/authors
- /v1/staff
- /v1/books
- /v1/multimedia
- /v1/loans
- /v1/reservations
- /v1/resources

Example – API DOMain (PUBLIC LIBRARY)



Paths:
- /v1/borrowers
- /v1/authors
- /v1/books
- /v1/multimedia
- /v1/loans
- /v1/reservations

Example – Normalised Domain (PUBLIC LIBRARY)



Paths:

- /v1/loanableAssets
- /v1/loanableAssets/{assetId}/creator
- /v1/loans
- /v1/loans/{loanId}/borrower
- /v1/loans/{loanId}/assets
- /v1/reservations
- /v1/reservations/{resId}/borrower
- /v1/reservations/{resId}/assets

- Design the target
- Map from functional domain
- **Decide on Root Resources**
- Use IDs from other APIs
- Keep things simple
- Engage with the consumer
- Think consumers!
- High availability
- Inconvenience

"Having a single root resource clarifies the entry point and clearly demarcates the primary functional scope of the API."

Example – root resources (PUBLIC LIBRARY)

Rooted on “loans”, API: “Library Loans API”

Paths:

```
/v1/
/v1/{loanId}/borrower
/v1/{loanId}/assets
/v1/loanableAssets
/v1/loanableAssets/{assetId}/creator
/v1/reservations
/v1/reservations/{resId}/borrower
/v1/reservations/{resId}/assets
```

Rooted on “loanableAssets”, API: “Library Assets API”

Paths:

```
/v1/
/v1/{assetId}/loans
/v1/{assetId}/creator
/v1/{assetId}/reservations
/v1/loans
/v1/loans/{loanId}/borrower
/v1/reservations
/v1/reservations/{resId}/borrower
/v1/reservations/{resId}/assets
```

- Design the target
- Map from functional domain
- Decide on Root Resources
- **Use IDs from other APIs**
- Keep things simple
- Engage with the consumer
- Think consumers!
- High availability
- Inconvenience

“Use Goldenised API IDs in resource objects wherever possible, The consumer always prefers to be using Golden data.”

- Design the target
- Map from functional domain
- Decide on Root Resources
- Use IDs from other APIs
- **Keep things simple**
- Engage with the consumer
- Think consumers!
- High availability
- Inconvenience

"Do and offer the minimum to fulfil the consumers requirements. It is quicker to develop, easier to maintain and simpler to consume."

AppyThings

• API Design Guidelines

- Design the target
- Map from functional domain
- Decide on Root Resources
- Use IDs from other APIs
- Keep things simple
- **Engage with the consumer**
- Think consumers!
- High availability
- Inconvenience

“Engage with consumers early on and use the OAS to express the APIs intent.”

- Design the target
- Map from functional domain
- Decide on Root Resources
- Use IDs from other APIs
- Keep things simple
- Engage with the consumer
- **Think consumers!**
- High availability
- Inconvenience

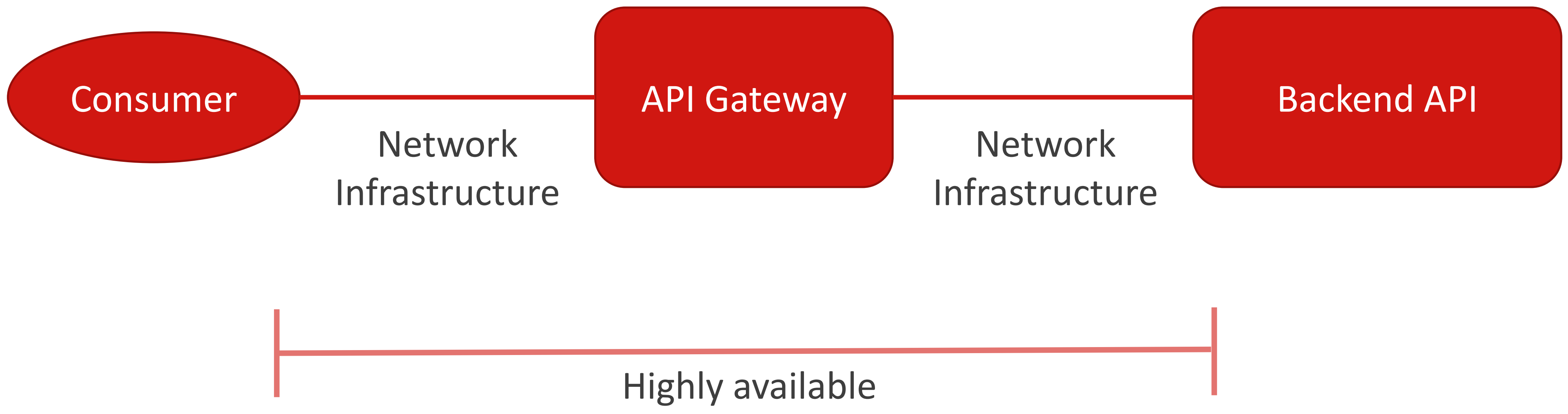
“Designing for multiple consumers from inception helps prepare for scale and can guide the design’s generalisation.”

- Design the target
- Map from functional domain
- Decide on Root Resources
- Use IDs from other APIs
- Keep things simple
- Engage with the consumer
- Think consumers!
- **High availability**
- Inconvenience

“Essential for building trust in an API, a High Availability strategy will sometimes force simplification of an APIs design.”

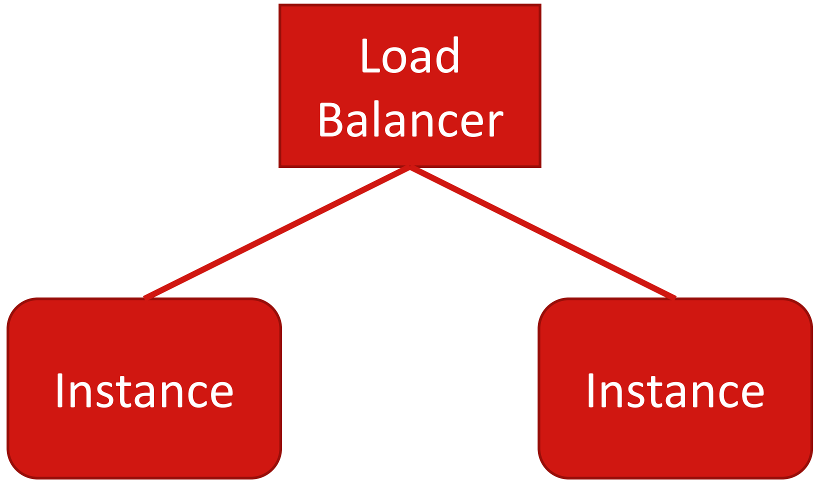
AppyThings

HIGH AVAILABILITY – DESIGN Considerations

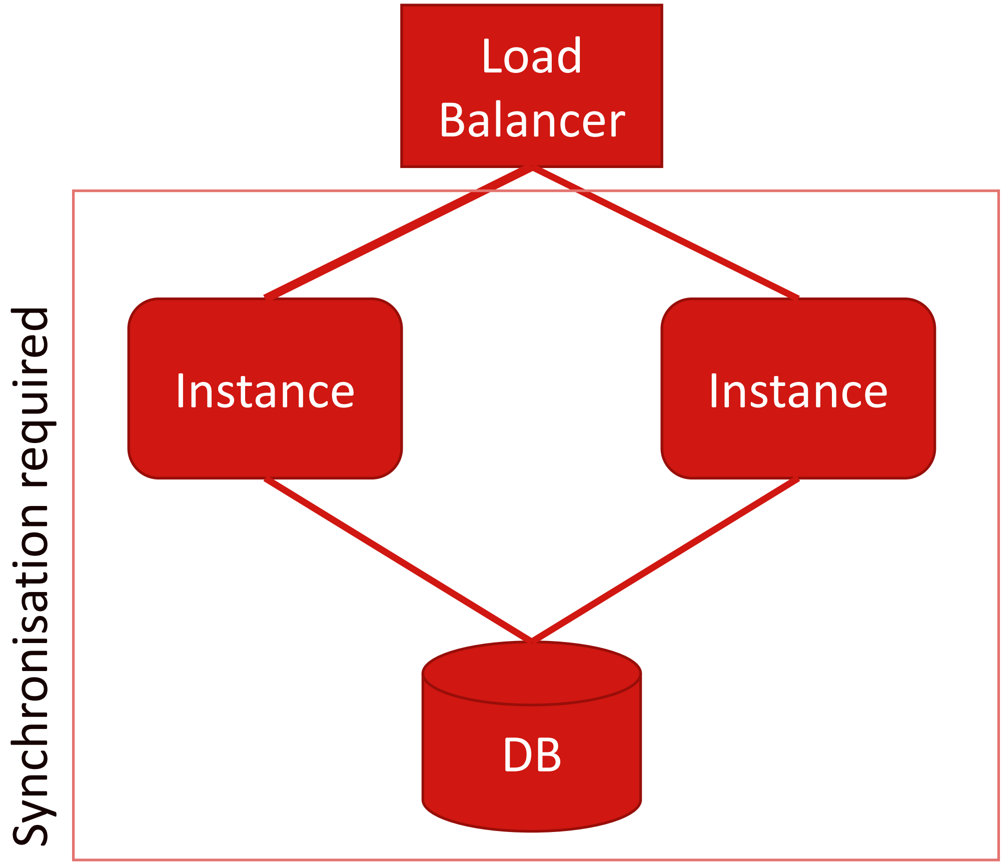


• HIGH AVAILABILITY – DESIGN Considerations

Stateless Design



Stateful Design



- Design the target
- Map from functional domain
- Decide on Root Resources
- Use IDs from other APIs
- Keep things simple
- Engage with the consumer
- Think consumers!
- High availability
- **Inconvenience**

“Convenience endpoints while useful for consumers can be an indicator of an overly complicated API design.”

AppyThings

THINGS TO CONSIDER USING DATA TRANSFER OBJECTS

A data transfer object:

- Represents a resource
- Masks the actual data store
- Represent data that makes sense for (potential) applications
- Handle common relations

  - Ex. Customer and Address tables

  – Include an Address object as a child of the Customers
- Avoid double resource names as much as possible

  - /v1/customerAddress -> /v1/customer/{id}/address
- Uses common language that end-users recognize
- Avoids abbreviations that are only know to employees

MOST COMMON HTTP VERBS

When complying to the restful frameworks the following verbs are used:

- GET one or more resources
- POST to create a new resource
- PUT to update a resource
- PATCH to partially update a resource
- DELETE to delete a resource

MOST COMMON HTTP STATUS CODES

- 2XX success
  - 200 OK
  - 201 Created
- 3XX redirection
  - Probably do not need to use these
- 4XX client error
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not found
- 5XX server error
  - Do not return stack traces in production

.Inspire .Innovate .Perform .Adapt

MEMBER OF APPY ENTERPRISE

Bert van Vugt

CDO

+316 50 80 67 45

bert@AppyThings.nl

www.AppyThings.nl

100%

/AppyThings/