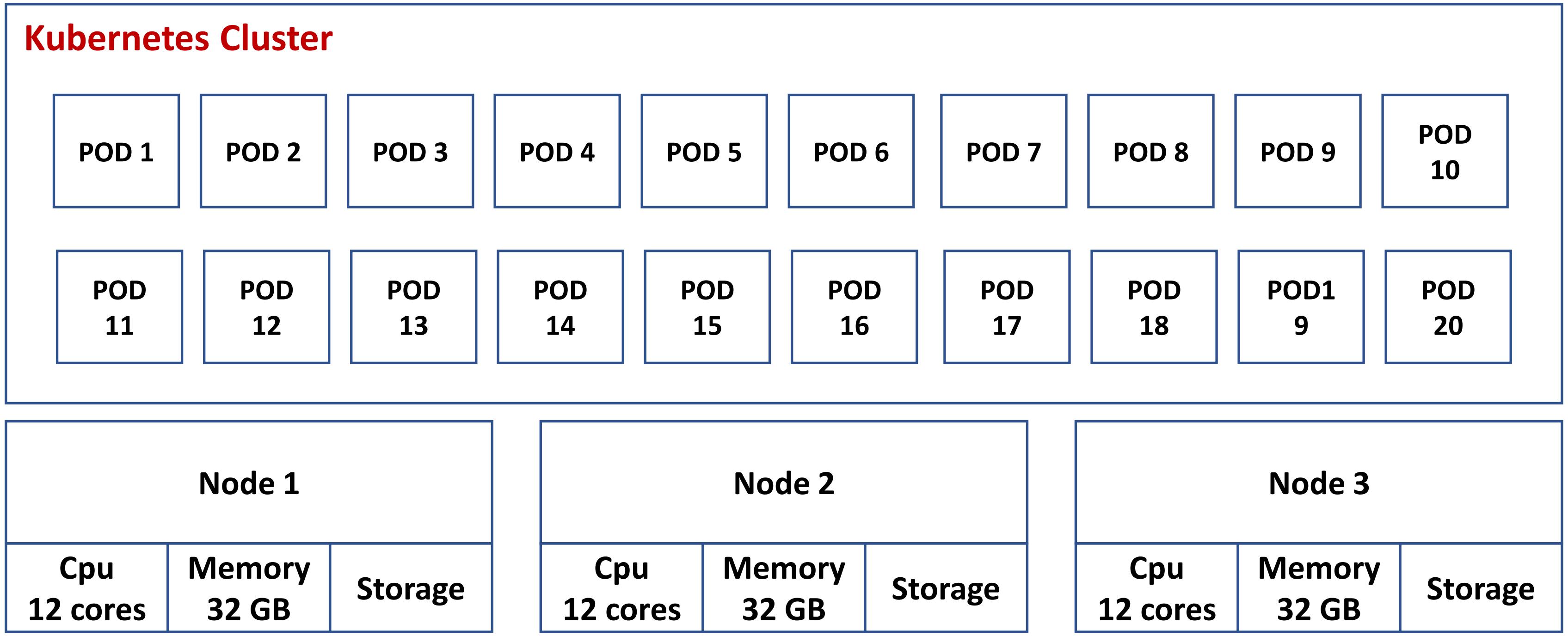
# Practical approach to sizing a Kubernetes cluster

- If total k8s node memory is approx. 96 GB, how many pods can be safely supported on it?
- Assuming all pods are equi-sized and requiring min(request) = 1 GB and max(limit) = 4 GB
- Not all pods will require 4 GB memory simulataneously
- Assuming, reliable gc occurring in the applications within the pods, at any given point of time, one can expect the average memory per pod to be 2.5 GB. So we can expect 96/2.5=38.4 pods to be supported, going by averages
- Max number of pods would be 96/1 = 96 pods
- Min number of pods would be 96/4 = 24 pods
- The number pods supported could range from 24 (low risk) to 96 (high risk of instability)
- Going by averages we should allow 38.4 pods, but, if we study the min-max memory requirements of our application, we can do a better job at predicting our apps "steady state" memory requirements per pod, say 2 GB and plan for 96/2 = 48 pods

- Total pods: 20
- All pods same size
- vCpu: min & max
- Memory: min & max

- Total cpu cores: 36
- Total Memory: 96 GB



**Kubernetes Cluster**

POD 1, POD 2, POD 3, POD 4, POD 5, POD 6, POD 7, POD 8, POD 9, POD 10

POD 11, POD 12, POD 13, POD 14, POD 15, POD 16, POD 17, POD 18, POD1 9, POD 20

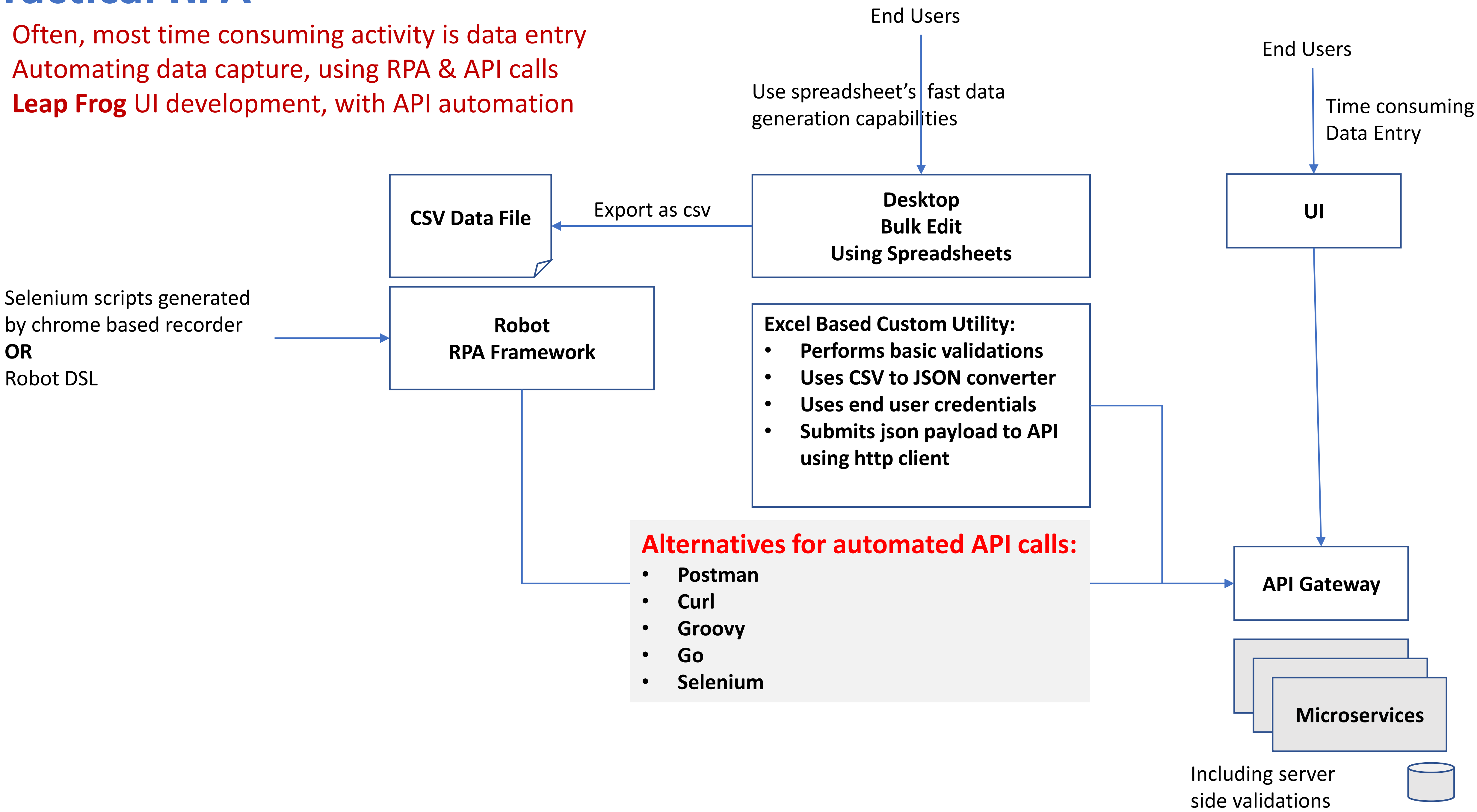
| Node 1 | | |
|---|---|---|
| Cpu<br>12 cores | Memory<br>32 GB | Storage |

| Node 2 | | |
|---|---|---|
| Cpu<br>12 cores | Memory<br>32 GB | Storage |

| Node 3 | | |
|---|---|---|
| Cpu<br>12 cores | Memory<br>32 GB | Storage |

# Practical RPA

- Often, most time consuming activity is data entry
- Automating data capture, using RPA & API calls
- **Leap Frog** UI development, with API automation



End Users

Use spreadsheet's fast data generation capabilities

**Desktop**
**Bulk Edit**
**Using Spreadsheets**

Export as csv

**CSV Data File**

Selenium scripts generated by chrome based recorder
**OR**
Robot DSL

**Robot**
**RPA Framework**

**Excel Based Custom Utility:**

- **Performs basic validations**
- **Uses CSV to JSON converter**
- **Uses end user credentials**
- **Submits json payload to API using http client**

**Alternatives for automated API calls:**

- **Postman**
- **Curl**
- **Groovy**
- **Go**
- **Selenium**

End Users

Time consuming Data Entry

**UI**

**API Gateway**

**Microservices**

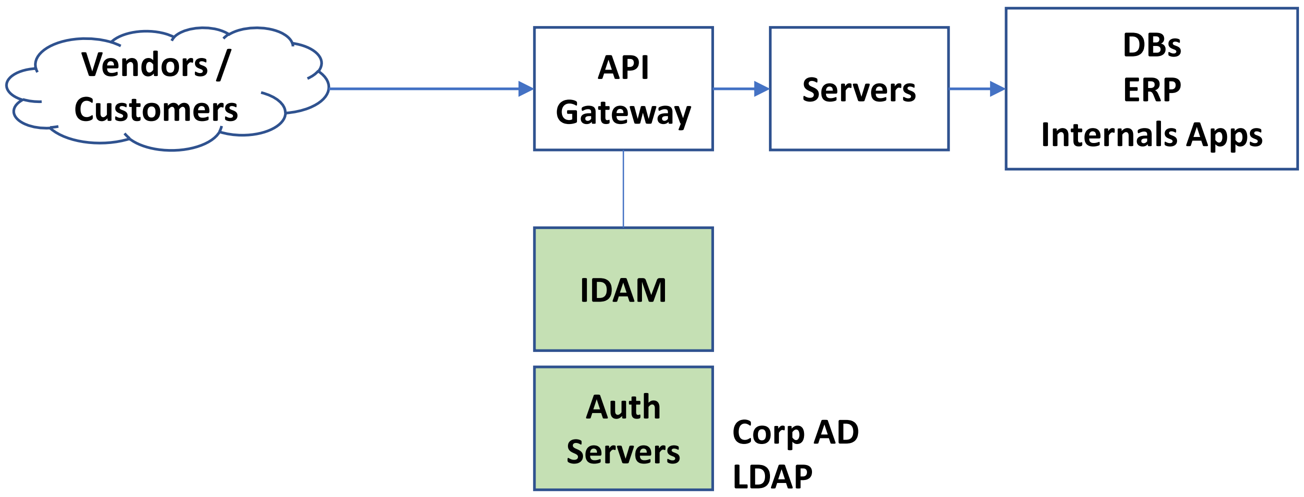
Including server side validations

Cloud based low code development platforms
Eg. PowerApps, Outsystems, Mendix

**Enterprise Requirements from Low Code providers:**

- Secure access from low code servers to enterprise GW
- Secure Auth integrated/SSO with enterprise IAM
- Tenant isolation of access and data in SAAS cloud
  - Dedicated low code servers and access per enterprise
- VPC network extending from SAAS servers to on prem network

## Enterprise Apps Deployments



Vendors / Customers

API Gateway

Servers

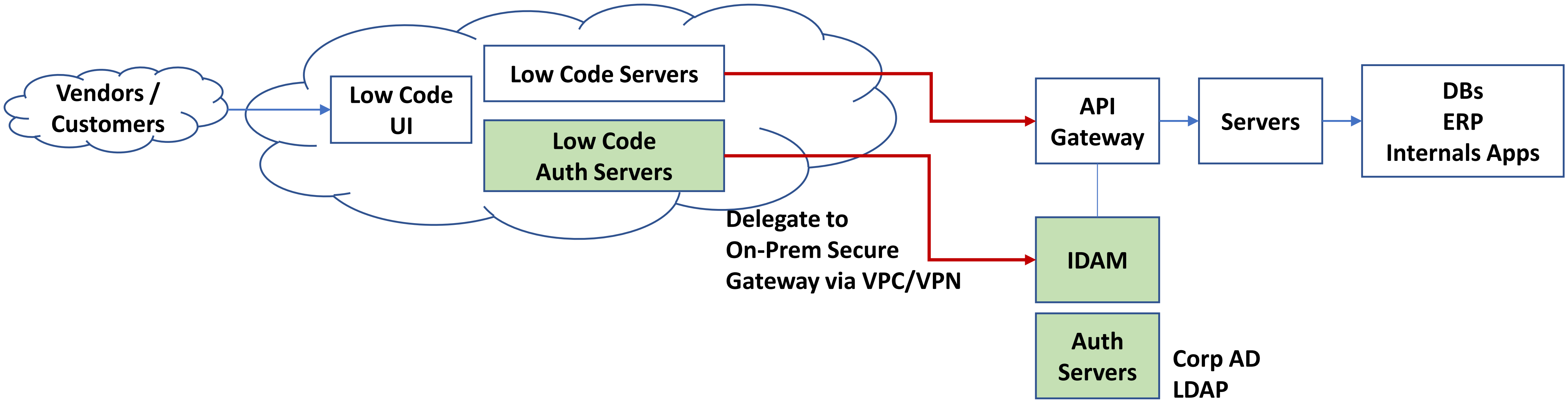
DBs
ERP
Internals Apps

IDAM

Auth Servers

Corp AD
LDAP

## Low Code based Enterprise Apps Deployments



Vendors / Customers

Low Code UI

Low Code Servers

Low Code Auth Servers

Delegate to
On-Prem Secure
Gateway via VPC/VPN

API Gateway

Servers

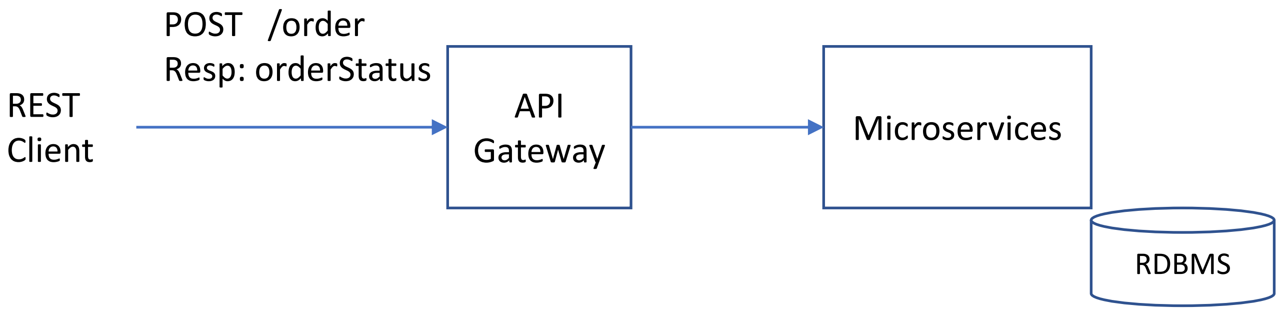
DBs
ERP
Internals Apps

IDAM

Auth Servers

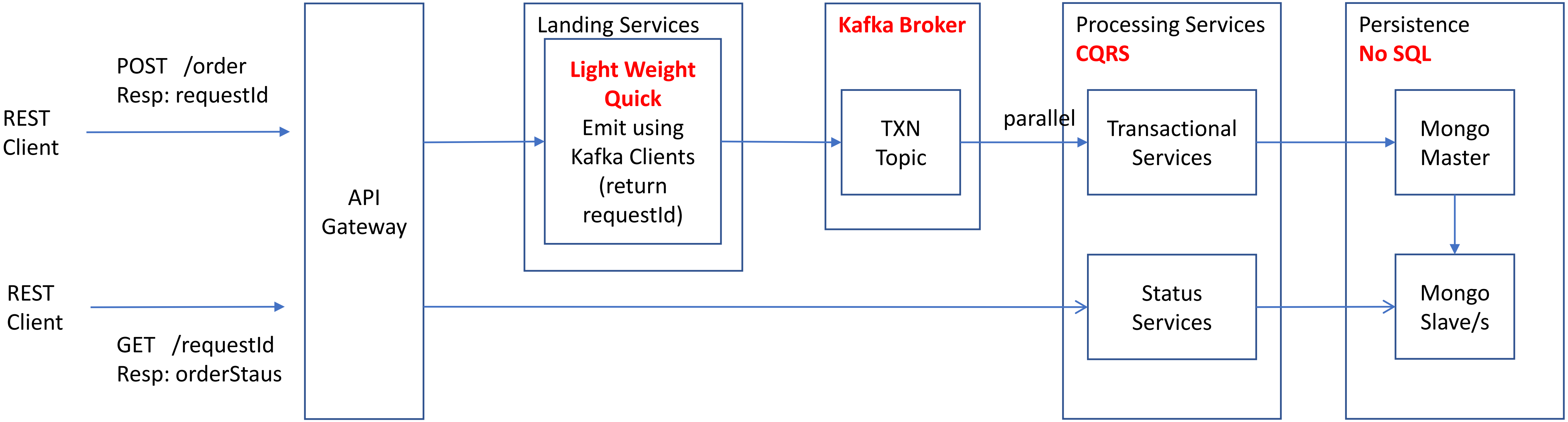
Corp AD
LDAP

# Traditional API Implementation

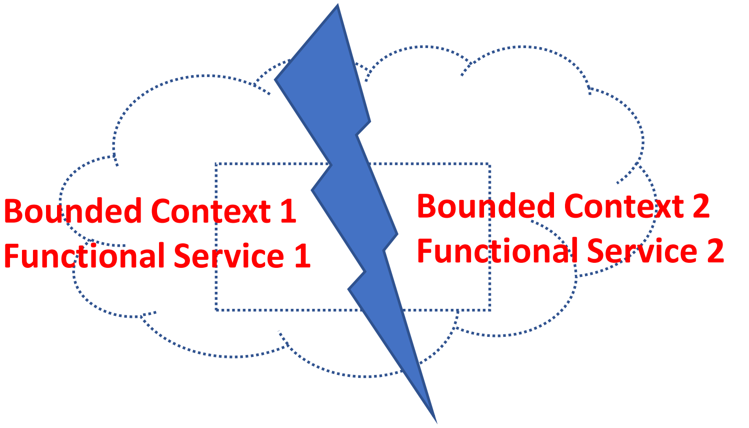


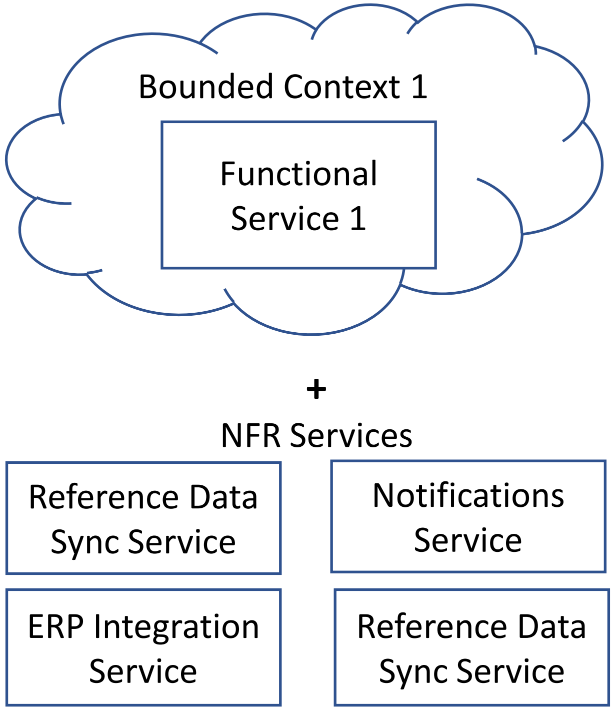
# High Scalability API Implementation

**Aysnc + Scale Out, all the way upto persistence**

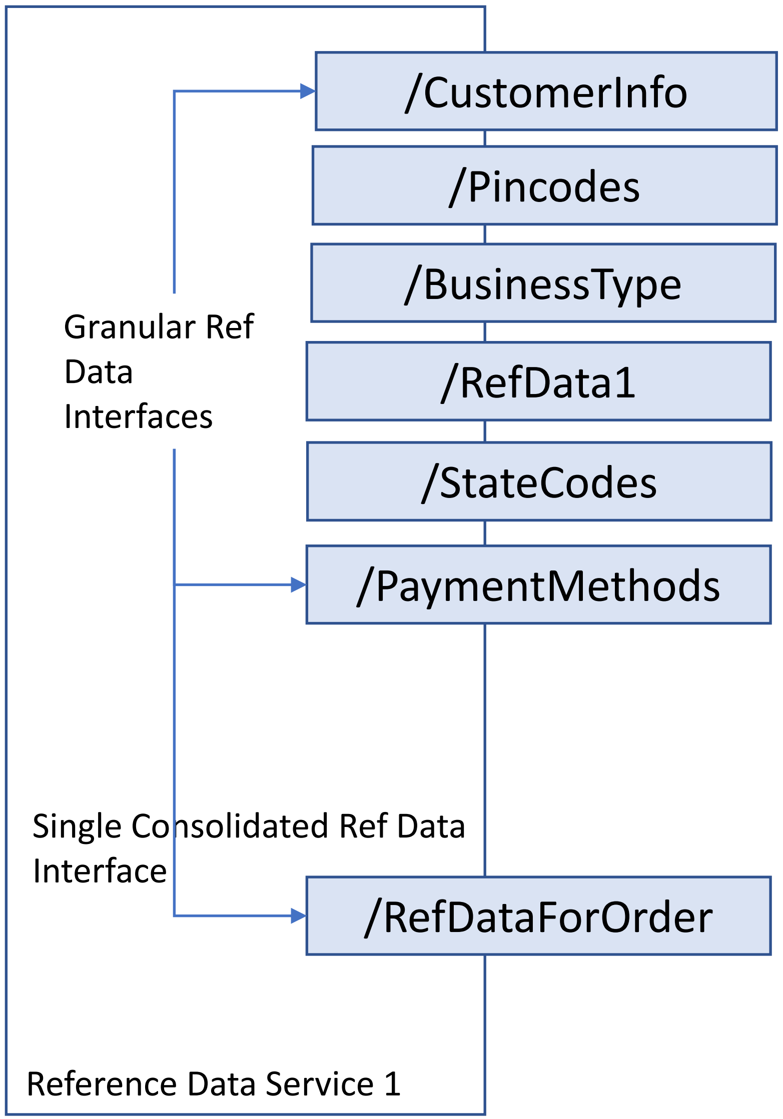


- Don’t forcibily create bounded contexts where none exist
- Instead hive out NFR services



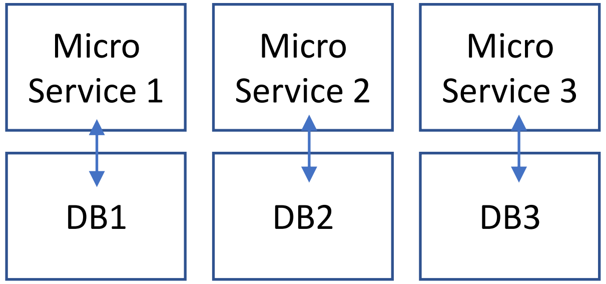


- To reduce http traffic / chattiness between microservices, consolidate API calls, where it makes sense

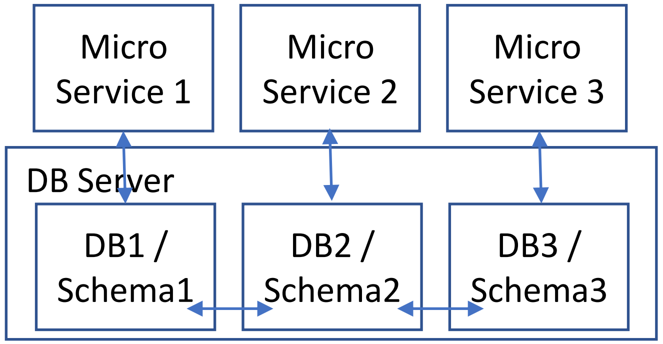


- Distinct Data model / microservice

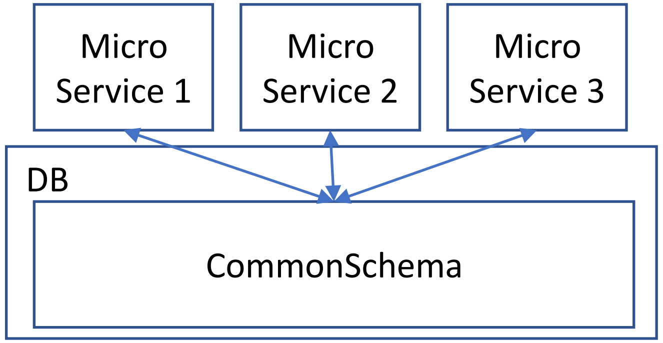
**Ideal**

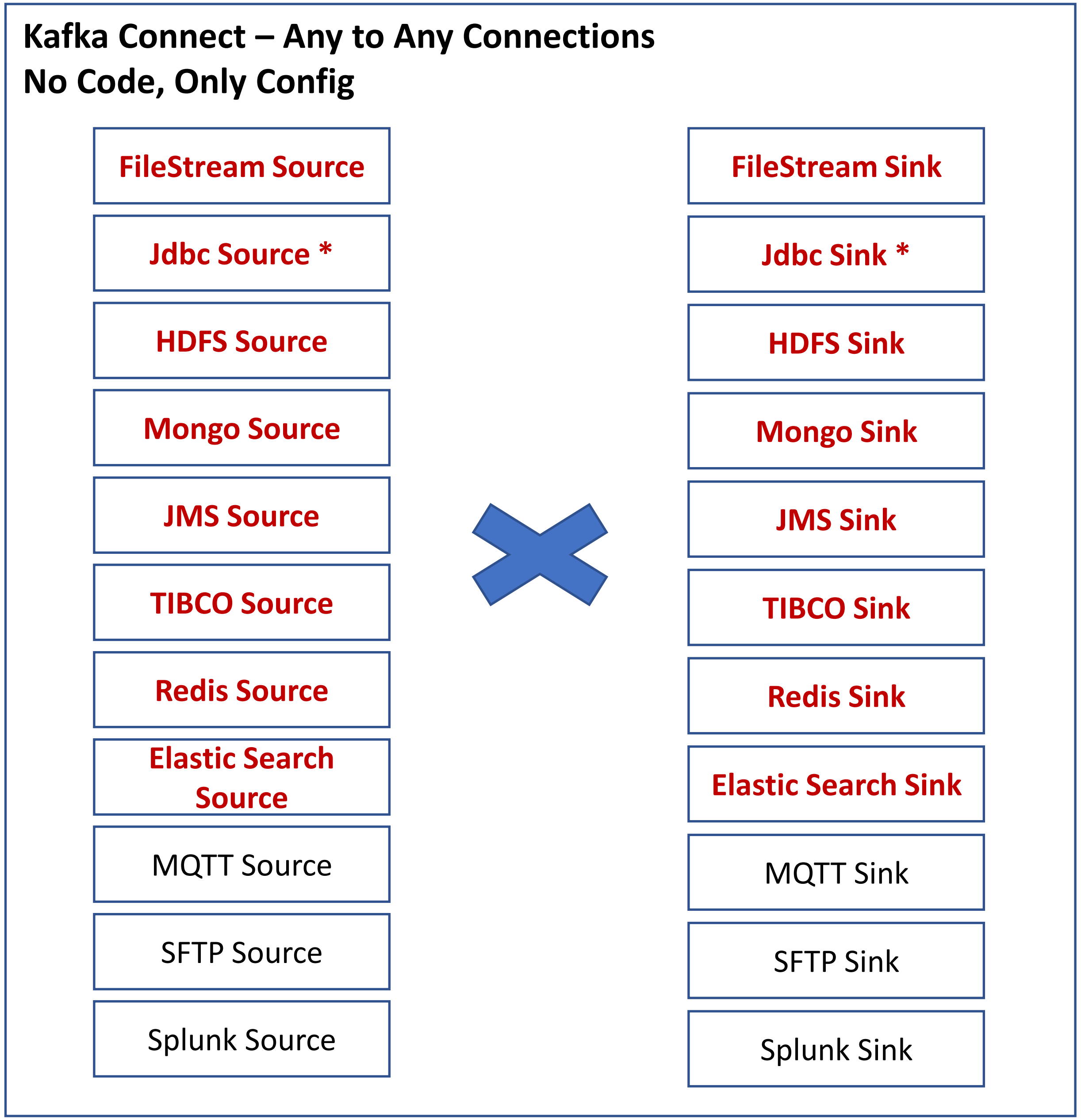


**Less than ideal**



**Rubbish, with microservices**





## Kafka Connect – Any to Any Connections
## No Code, Only Config

| | | |
|---|---|---|
| idocs, logs, csv, json | FileStream Source | FileStream Sink |
| Oracle, Hana, MySQL | Jdbc Source * | Jdbc Sink * |
| Hive | HDFS Source | HDFS Sink |
| Doc DB, bson/json | Mongo Source | Mongo Sink |
| Messaging, Async | JMS Source | JMS Sink |
| Integration, ESB | TIBCO Source | TIBCO Sink |
| Caching | Redis Source | Redis Sink |
| Text search + index | Elastic Search Source | Elastic Search Sink |
| | MQTT Source | MQTT Sink |
| | SFTP Source | SFTP Sink |
| | Splunk Source | Splunk Sink |

* = schema mandatory

# High throughput, batch processing, using Kafka and Mongo DB

Partner Integration based on bulk or event based incoming data

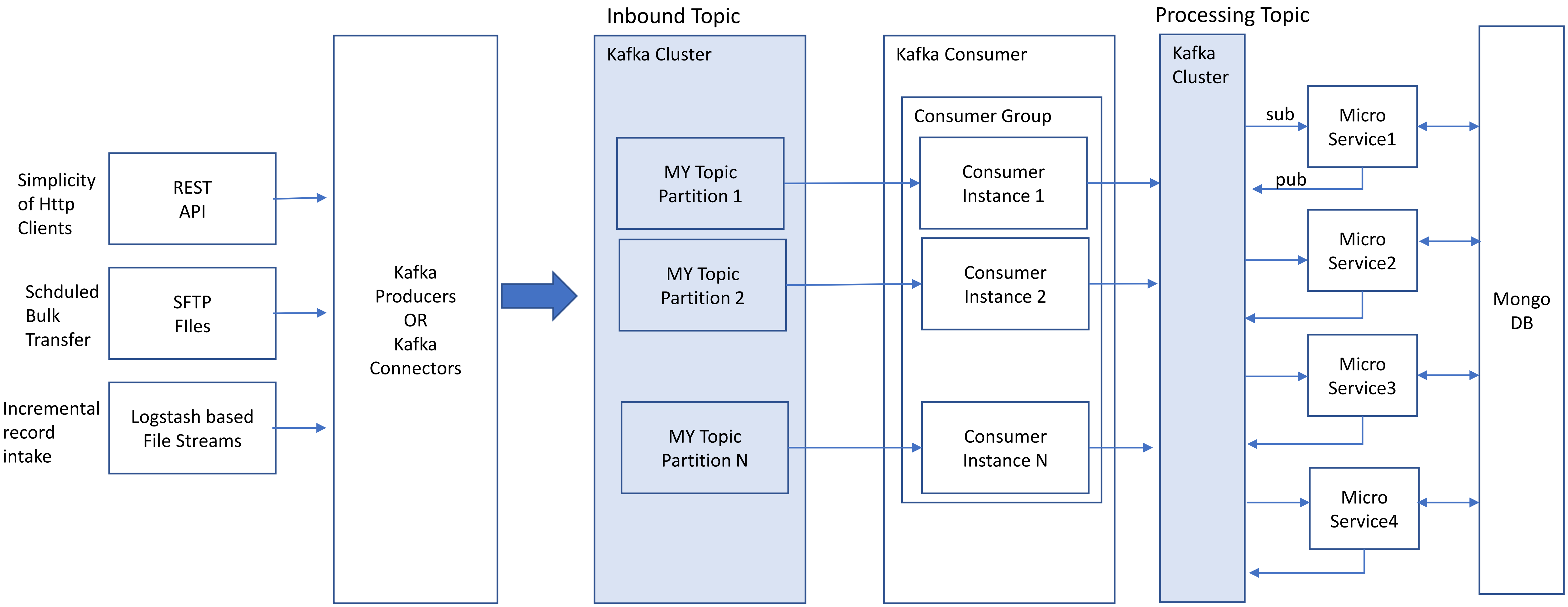
Leverage Kafka Connectors or write custom kafka producers

Leverage Kafka IO Scalability for the landing platform

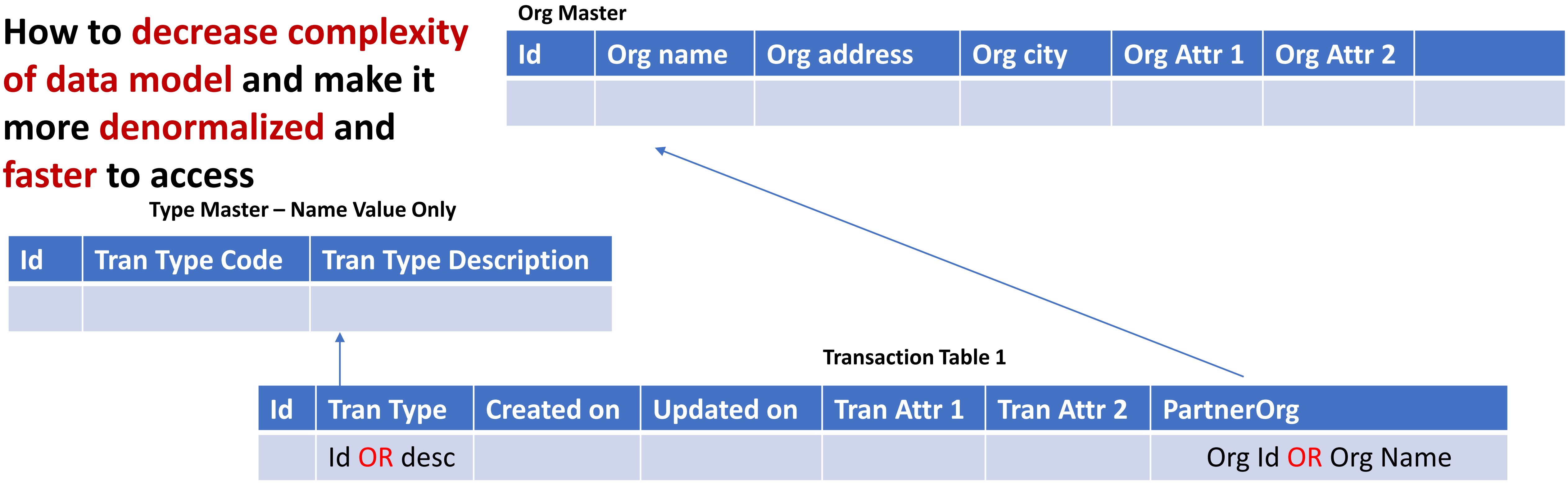
Throttle Consumption as per your resources

Use Kafka based async processing for better throughput (TPS)

Mongo DB for scalable persistence



# How to decrease complexity of data model and make it more denormalized and faster to access



Org Master

| Id | Org name | Org address | Org city | Org Attr 1 | Org Attr 2 |  |
|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |

Type Master – Name Value Only

| Id | Tran Type Code | Tran Type Description |
|---|---|---|
|  |  |  |

Transaction Table 1

| Id | Tran Type | Created on | Updated on | Tran Attr 1 | Tran Attr 2 | PartnerOrg |
|---|---|---|---|---|---|---|
|  | Id OR desc |  |  |  |  | Org Id OR Org Name |

**For name value masters like Type Master:**

- Ascertain if tran type code has business meaning
- If not, code is merely a "made-up" attribute
- The code and description may never change independently

In such cases:

- Best retain Type Master Table but with single column "description" for data driven dropdowns in UI
- Store the Type description directly in the transaction table

**For classical masters like Org Master:**

- Ascertain, which Org attributes need to be shown as part of transaction, say only org name is required
- If orgname is a unique business key and cannot change over time
- Store the orgname directly in the transaction table
- Even org address can be stored in the transaction table directly
- If 2 months later, org address changes, this transaction should ideally still show the old address since that is factually correct !
- Use the Org Master only to show the Org drop downs and "current"Org Details
- No need to have foreign key contraints needlessly, between the transaction tables and masters, **if they can be avoided**