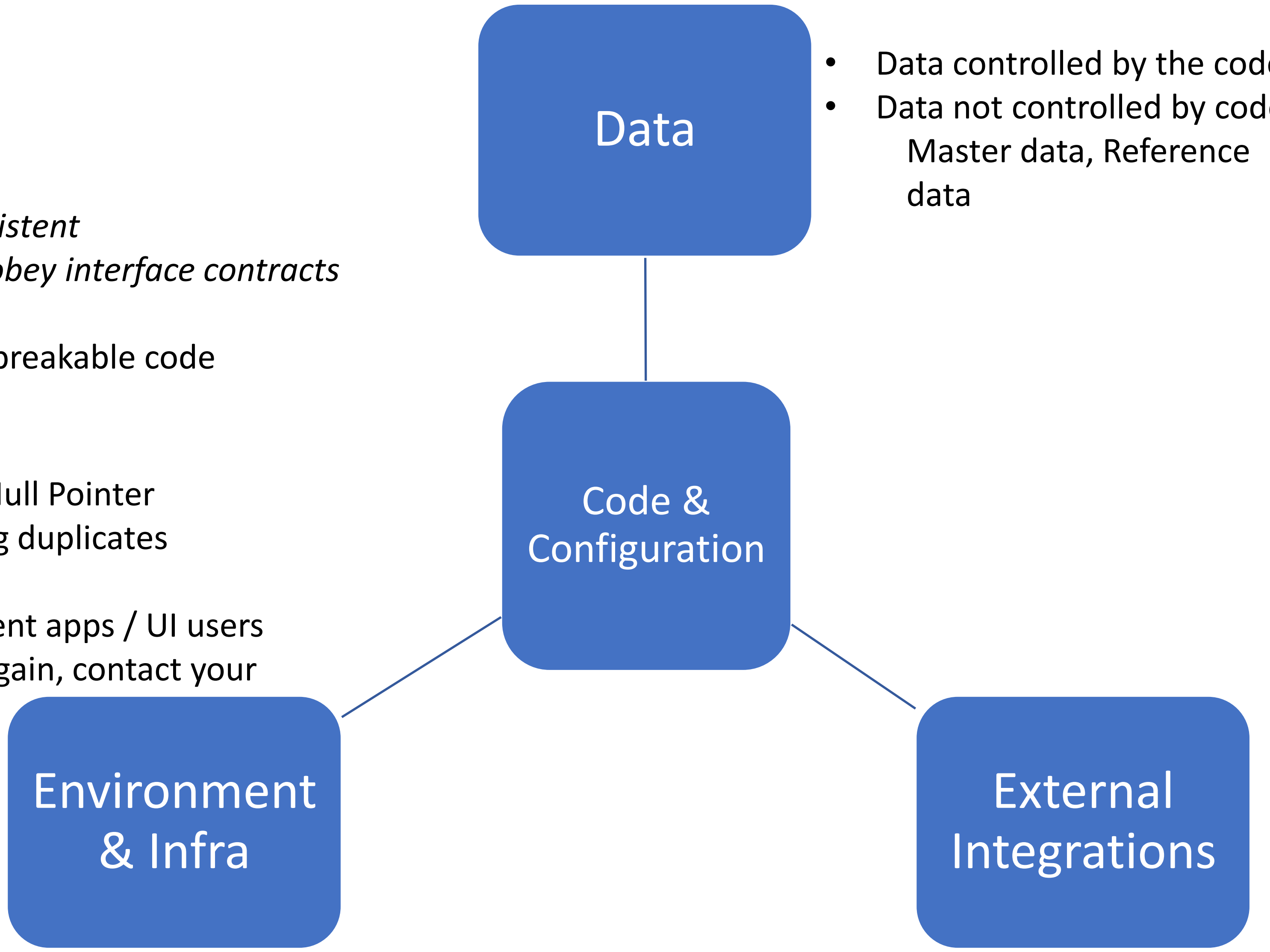
- **Code is written assuming pre-requisites of certain data, environment and integration interfaces**
- If any of the assumptions break the code breaks

Code is not responsible for following:
- *Ensuring that environment and infra remain stable*
- *Ensuring data not controlled by the code, remains consistent*
- *Ensuring External Integrations remain responsive and obey interface contracts*

So, what is good code ? Since it is not possible to write unbreakable code

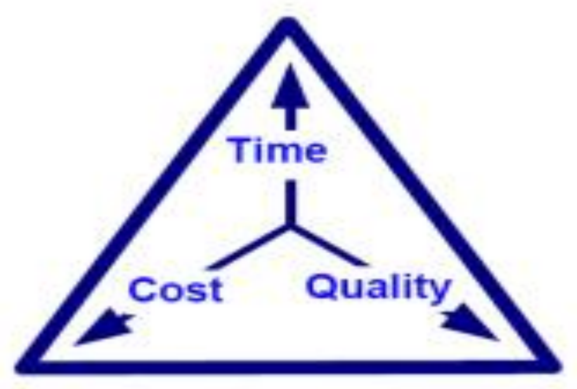
Good Code should
- Not cause exceptions due to programming errors like Null Pointer
- Retry in case of recoverable exceptions without causing duplicates
- Reliably log root causes of unrecoverable exceptions
- Convey appropriate and relevant error messages to client apps / UI users
    - Tell users about remedial actions like please try again, contact your admin@myapp.com



- Data controlled by the code
- Data not controlled by code
    Master data, Reference data

- CPU, Memory, Network, Storage
- Routers, Gateways, Firewalls

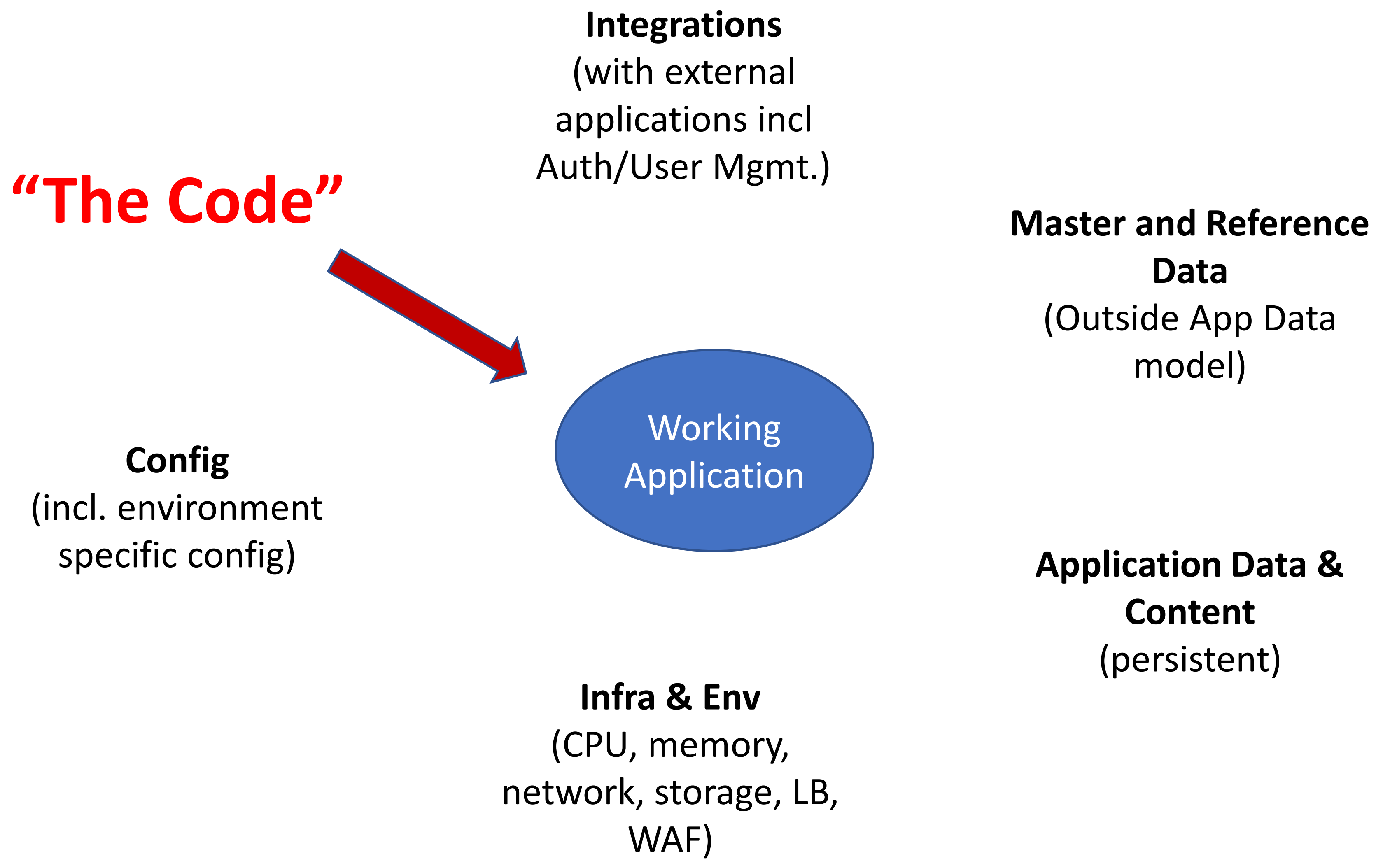
- ERP
- Other Backends Apps
- Partner Apps via APIs
- Other App APIs within Org



Why the cost-quality-time triad may be flawed, especially in context of software projects:

- You can quantify cost and time, but quality is not as easy and accurate
- Cost and time do not have a linear inverse relation, increasing costs often give diminishing returns in later periods
- It assumes that the tasks are low-skilled or mechanical, software development is skill centric
- Quality is dynamic and a function of time and can deteriorate or bounce back
- Integration complexity which increases with time and output produced, is not considered at all

Even if the code is reasonably good, there are other factors which affect a working application.


Integrations (with external applications incl Auth/User Mgmt.)
“The Code”
Master and Reference Data (Outside App Data model)
Working Application
Config (incl. environment specific config)
Application Data & Content (persistent)
Infra & Env (CPU, memory, network, storage, LB, WAF)

Async Processing Pros

- Highly decoupled interaction is possible between clients and servers
- High Scalability to support high TPS can be achieved
- Scale out is easy if processing is stateless
- Throttling on consumers side is easy and imparts control
- Possible to isolate high speed producers from low- speed consumers

Async Cons

- The end-to-end processing is broken down into 2 parts publish and subscribe
- Difficult to maintain transactional semantics of all or nothing
- In Flight loss of messages is possible if error handling is imperfect
- Duplicate messages are possible, especially during restart and recovery
- Correlation id needs to be maintained between producers and consumers
- Efforts to achieve FIFO sequential processing
- Efforts for error handling are more as compared to sync

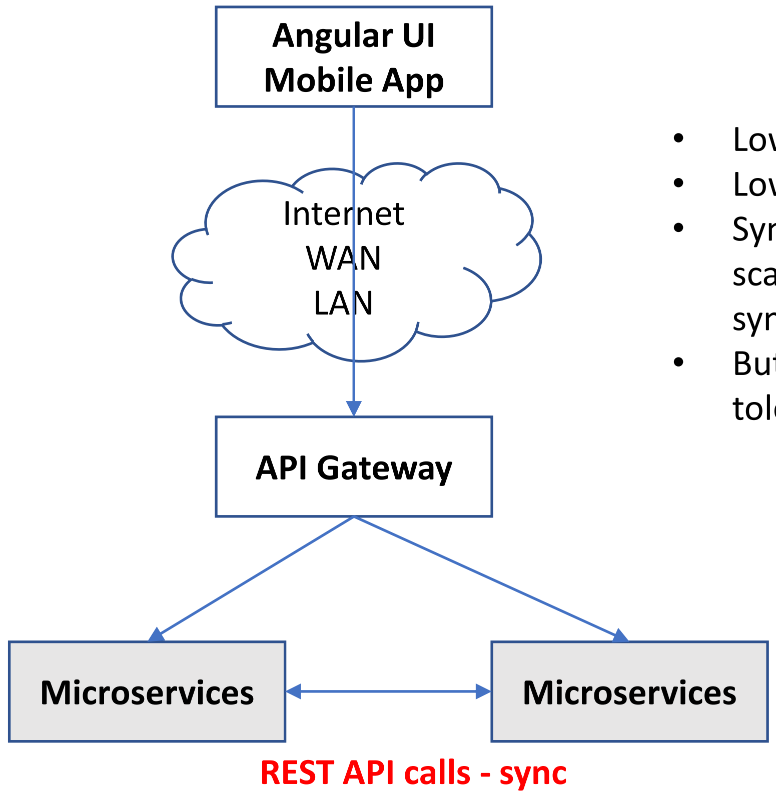
Sync Processing Pros

- Easier to maintain transactional semantics of all or nothing
- Less chances of silent message loss
- Duplicate messages are not easily possible
- No need for Correlation id
- FIFO sequential processing can be controlled by client
- Error handling is standard and easier, since client gets error communicated immediately during call

Sync Cons

- Client and servers are more coupled
- Servers have, to be available during client interaction
- Scalability reduced due to blocking sync calls
- Scale out needs load balancing
- Throttling is, not easy, timeouts need to be monitored
- Consumers need to be at same speed as producers

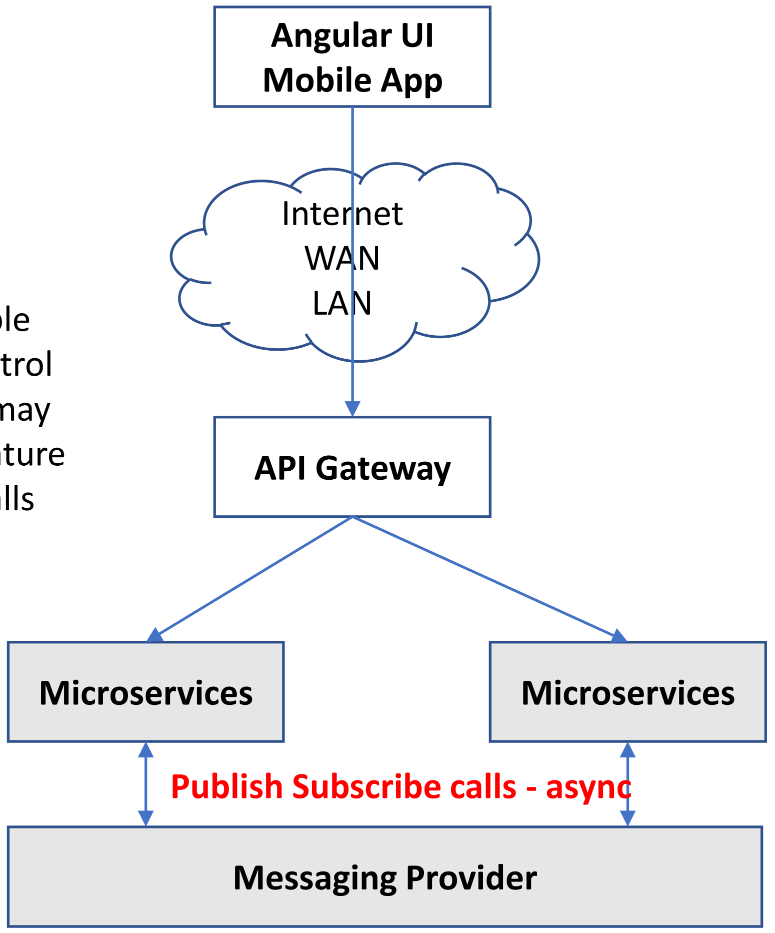
Scalability and Server-Side Latency are often at logger heads with each other



- Low concurrent users
- Low TPM on server side
- Sync REST Calls reduce scalability, due to blocking sync calls inter-microservices
- But server-side latency is tolerable for lower TPMs

Sync REST Calls
Low Latency
Low Scalability

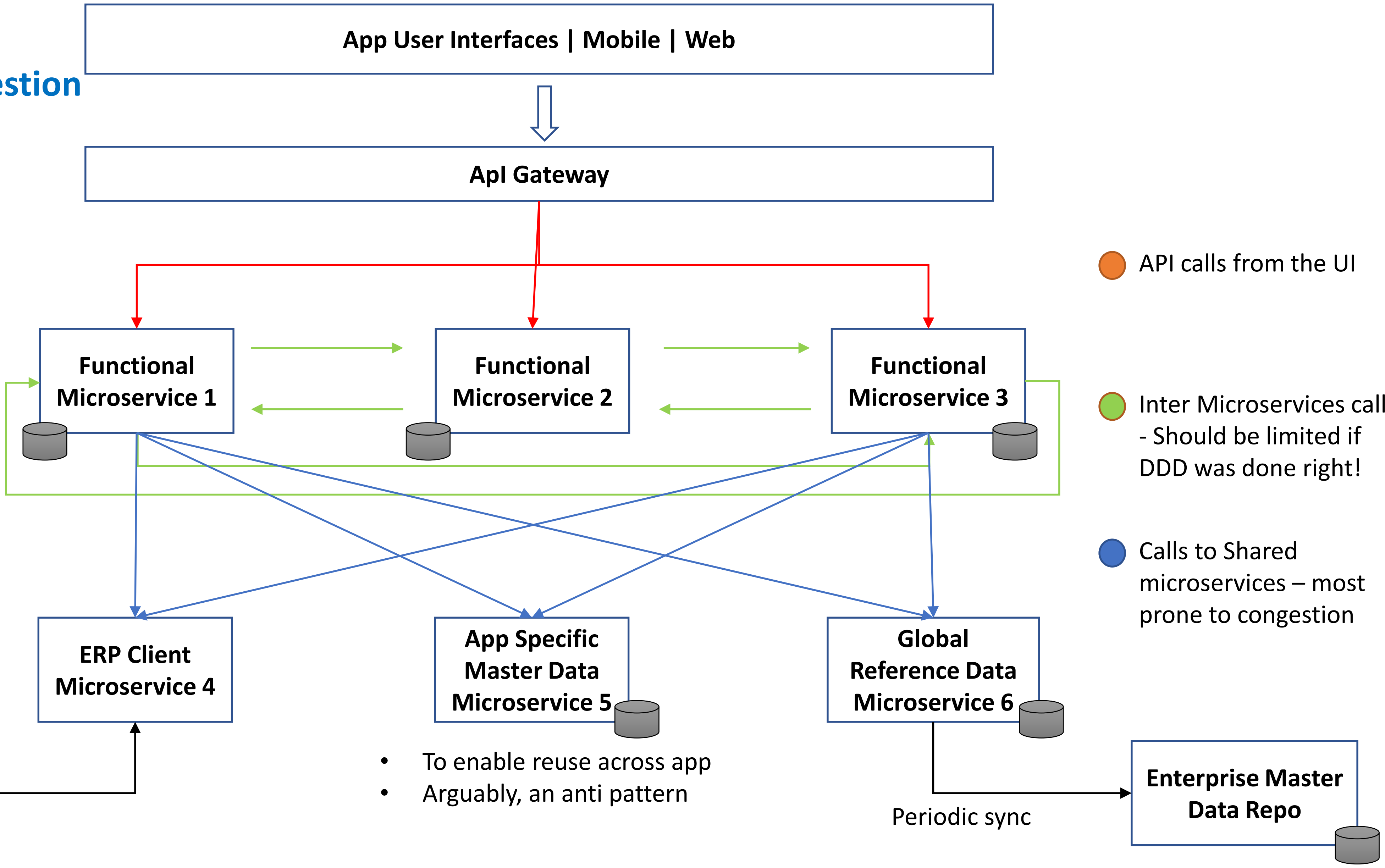
- High concurrent users
- High TPM on server side
- Async Messaging Calls conducive to scalability
- Throttling of TPM possible due to consumption control
- But server-side latency may increase due to async nature of inter-microservices calls



Async Calls
High Latency
High Scalability

Sync Microservices
RPC - Network Congestion

- 1 API call from UI can result in numerous calls In the lower layers
- RPC calls to shared services need to be designed to ***balance data returned vs number of invocations necessary***
- **Optimizing RPC calls to shared services is key to control network congestion in sync microservices arch**


App User Interfaces | Mobile | Web
ApI Gateway
Functional Microservice 1
Functional Microservice 2
Functional Microservice 3
ERP Client Microservice 4
App Specific Master Data Microservice 5
Global Reference Data Microservice 6
ERP Shared Backend
Enterprise Master Data Repo
Periodic sync

Txns in System of records

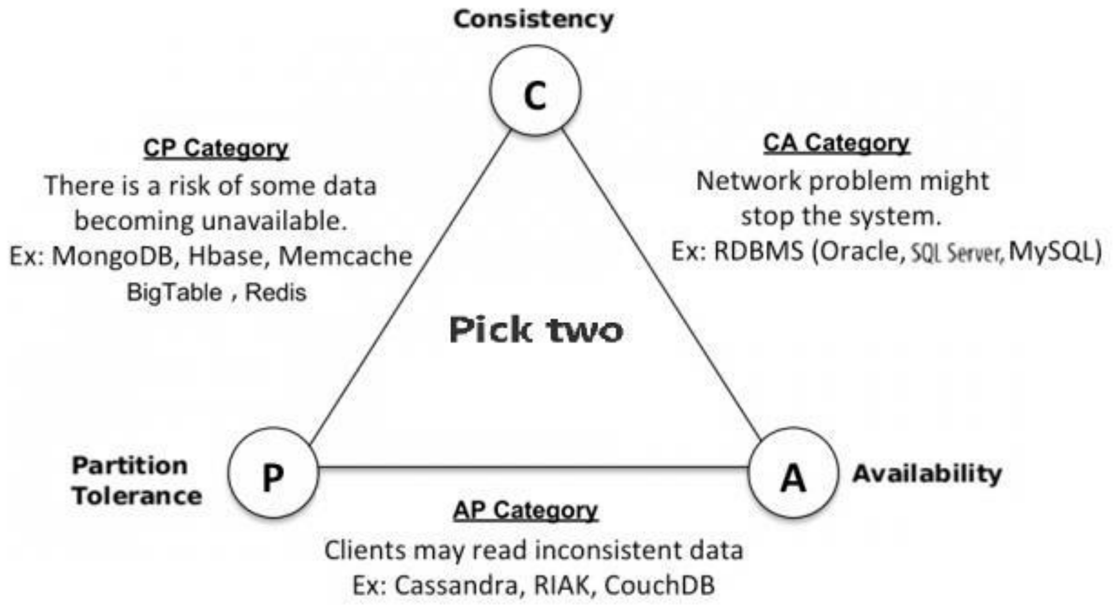
- To enable reuse across app
- Arguably, an anti pattern

● API calls from the UI

● Inter Microservices calls - Should be limited if DDD was done right!

● Calls to Shared microservices – most prone to congestion

ganesh.ghag@gmail.com


Consistency
C
CP Category
There is a risk of some data becoming unavailable.
Ex: MongoDB, Hbase, Memcache BigTable , Redis
CA Category
Network problem might stop the system.
Ex: RDBMS (Oracle, SQL Server, MySQL)
Pick two
Partition Tolerance
P
A
Availability
AP Category
Clients may read inconsistent data
Ex: Cassandra, RIAK, CouchDB

Databases and the CAP Theorem Basics

- Consistency refers to every commited write to DB being read consistently by every DB client
- Availability refers to the DB being available for reads and writes, ideally at all times. It can be expressed as a number. To provide 100% availability for reads and writes, we may need to cluster the DB
- Partition Tolerance comes into the picture when the DB contains a huge data and needs to be distributed using some fom of sharding

When the DB is size is relatively small or medium OR when small unavailability can be tolerated, there is no need to have multiple instances of DB and hence consistency can be guaranteed – but availability(multi instance) and large distributed data(sharding) cannot be supported

When we try to increase the availability, by clustering, the additional overhead of multiple instance synchronization, will cause consistency to be less than perfect

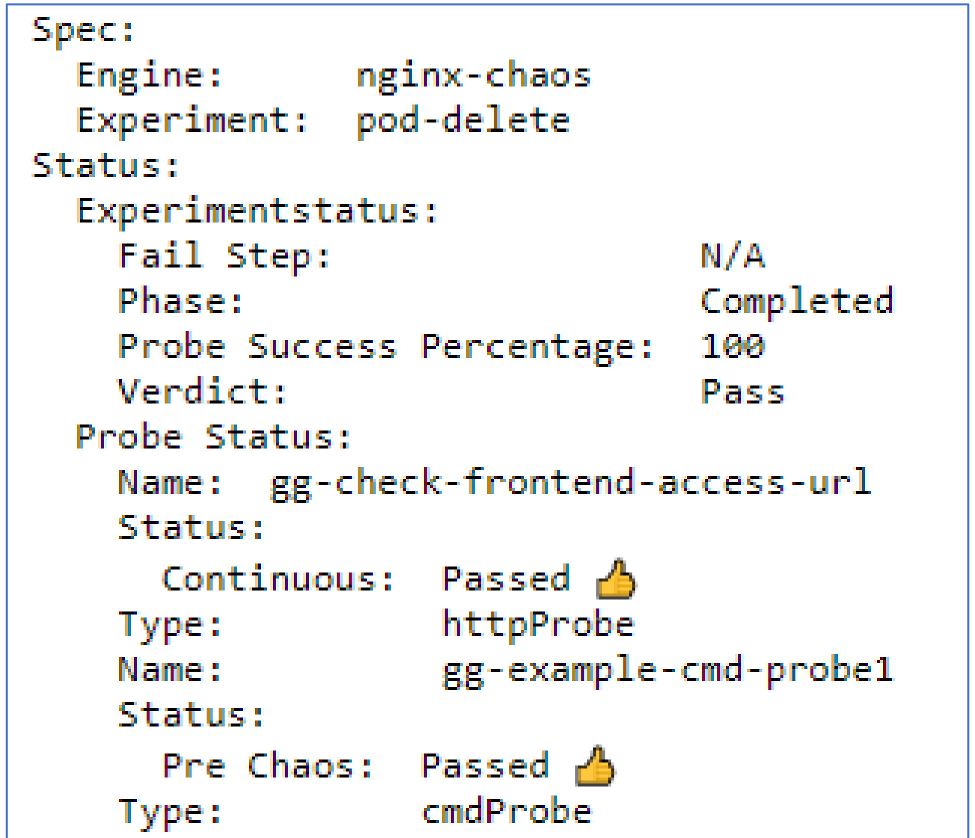
When we try to increase DB data capacity, by distributing DB via say sharding, parts of the network going down can make some DBs unavailable, hence availability will be less than perfect.

Summary:
We can never hope to have consistency, availability and partition tolerance all at a maximum. Everytime we increase one of the 3 parameters, we will compromise the others

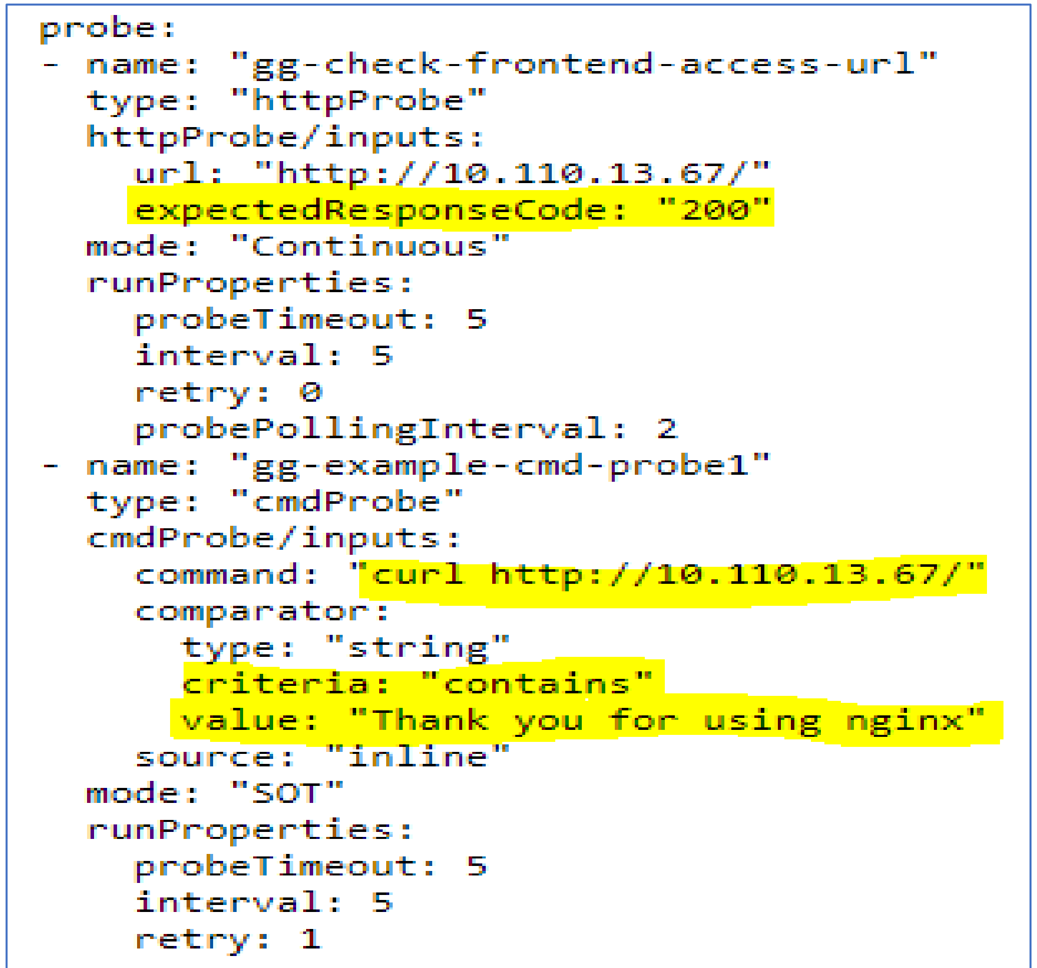
While using litmus based #chaos experiments, the ready made experiments do provide some pre defined checks to assert the pass and fail status of the experiment. But these "built-in" checks may not be sufficient to "infer" that our k8s based application is resilient to outages like pod deletes, pod out of memory, etc.

- **It is advised by litmus to use "custom assertions" by writing "probes" that can be plugged into your experiments, declaratively.**
- **Below are screen shots of a sample http probe and cmd probe, that can be used to check application resiliency.**
- **The http probe checks that a microservice api returns http response 200, within a configured time say 5 seconds.**
- **The cmd probe using curl command to assert that the api response contains a certain text.**
- **Normally cmd probes are used to login to databases, using CLIs and assert, the DB state using SQL commands.**

https://docs.litmuschaos.io/docs/litmus-probe/
https://docs.litmuschaos.io/docs/litmus-probe/#probe-status--deriving-inferences



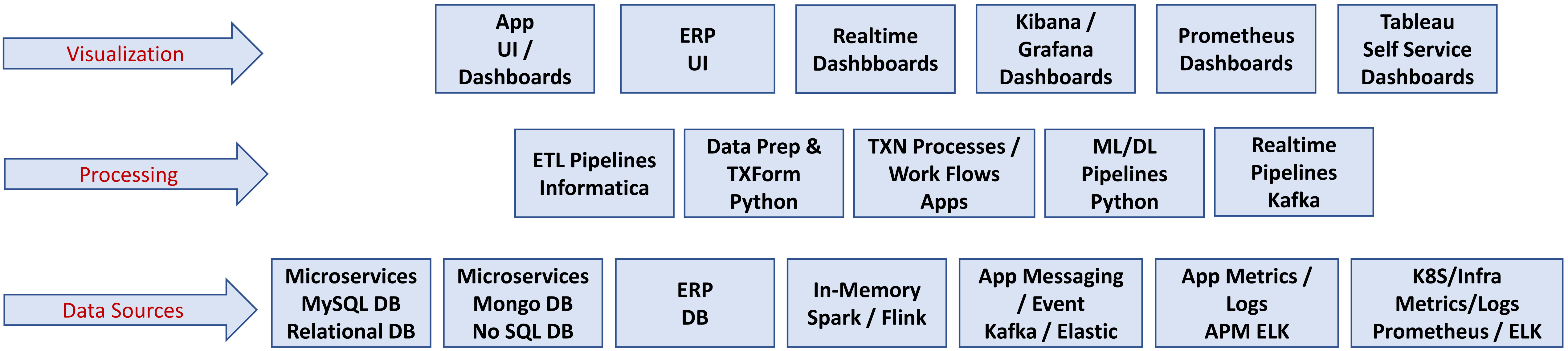
```
Spec:
  Engine:      nginx-chaos
  Experiment:  pod-delete
Status:
  Experimentstatus:
    Fail Step:                  N/A
    Phase:                      Completed
    Probe Success Percentage:   100
    Verdict:                    Pass
  Probe Status:
    Name:  gg-check-frontend-access-url
    Status:
      Continuous:  Passed 👍
    Type:          httpProbe
    Name:          gg-example-cmd-probe1
    Status:
      Pre Chaos:  Passed 👍
    Type:         cmdProbe
```



```
probe:
- name: "gg-check-frontend-access-url"
  type: "httpProbe"
  httpProbe/inputs:
    url: "http://10.110.13.67/"
    expectedResponseCode: "200"
  mode: "Continuous"
  runProperties:
    probeTimeout: 5
    interval: 5
    retry: 0
    probePollingInterval: 2
- name: "gg-example-cmd-probe1"
  type: "cmdProbe"
  cmdProbe/inputs:
    command: "curl http://10.110.13.67/"
    comparator:
      type: "string"
      criteria: "contains"
      value: "Thank you for using nginx"
    source: "inline"
  mode: "SOT"
  runProperties:
    probeTimeout: 5
    interval: 5
    retry: 1
```

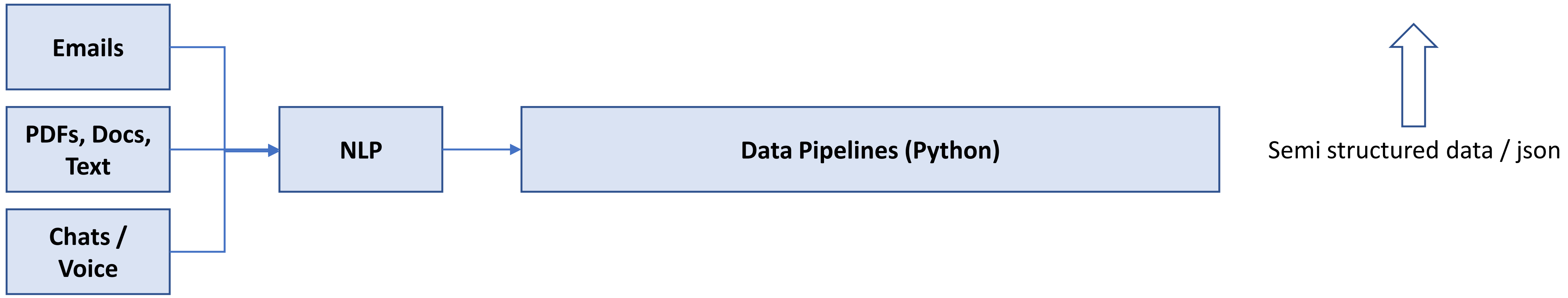
Enterprise Data – Sources, Processing and Visualization

Structured data



Visualization
App UI / Dashboards
ERP UI
Realtime Dashbboards
Kibana / Grafana Dashboards
Prometheus Dashboards
Tableau Self Service Dashboards
Processing
ETL Pipelines Informatica
Data Prep & TXForm Python
TXN Processes / Work Flows Apps
ML/DL Pipelines Python
Realtime Pipelines Kafka
Data Sources
Microservices MySQL DB Relational DB
Microservices Mongo DB No SQL DB
ERP DB
In-Memory Spark / Flink
App Messaging / Event Kafka / Elastic
App Metrics / Logs APM ELK
K8S/Infra Metrics/Logs Prometheus / ELK

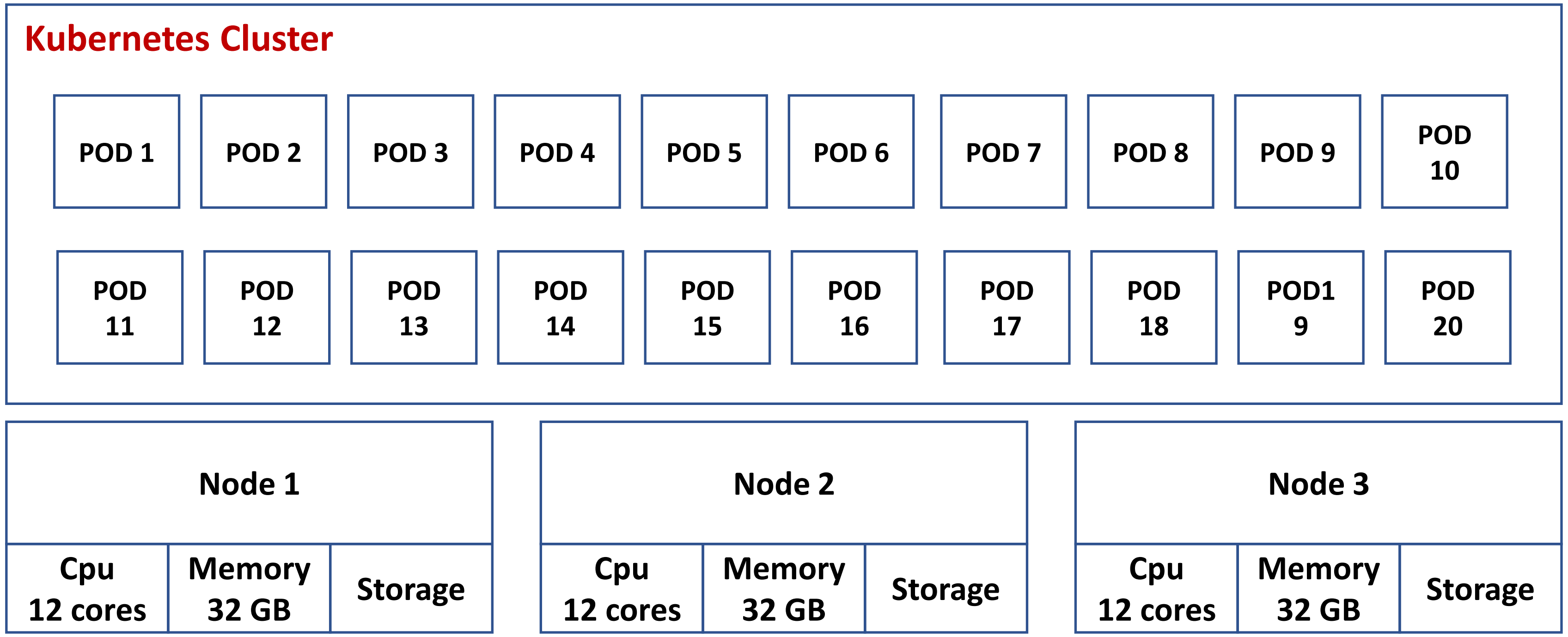
Unstructured data



Practical approach to sizing pods in 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

- APIs need to be organized, categorized & displayed as per the point of view of API consumers/API customers
  - What is the API customer looking for? How can things be made convenient for him?
  - API categories should be “functional” and “intuitive” for searching by API customers
- Not every REST endpoint is an integration API. All APIs exposed for web UI, do not always make, for good integration APIs
  - A microservice may contain 50 REST endpoints, not all need to be faceless integration APIs
  - An intgeration API needs to be well thought out and carved out from point of view of API customer
- Its being called the API Economy! All Application design should be API-Centric and not Web UI centric
  - UIs can change, within 6 months unlike APIs which have a longer life cycle
  - Multiple channels like mobile apps, IOT, chatbot, voice,AR/VR, partner applications can act as clients to the API.
  - Looking at only the Web UI to dictate the API is short sighted
- APIs and documentation
  - Documentation needs to minimal, crisp, with technical clarity, examples/samples, that can be immediately tried out in the browser
  - Working, ready made, try-it-yourself example is the best documentation
  - Links for ancilliary pre-requistes to invloking an API, should be handy
    - How to get the OAuth2 token
    - How to get the vendorId parameter using a related API (use - see also links)
  - Common data structures should be re-used across “all” APIs
    - Instead of a username, password, have a common object like UserContext which is re-used across all APIs and has attributes like username, credentials, among other things
  - APIs should be self sufficient
    - If you provide transactional API to create, of course, you need to provide an API to get, getAll, modify and delete, else who will do the cleanup?
  - Concepts like pagination, filters need to be shared across all the APIs as common data structures