Designing Large Systems

What is Quality ?

- It’s a really vague question !

- Quality is always relative
  - Entity of interest
  - Viewpoint
  - Attributes that we want to assess
  - Levels of abstraction

imagesource@gadgetflow.com

How do we assess quality ?

Different Views (Garvin1997)

- Transcendental view: seen/not-defined , mystical
- User view: fitness for purpose
- Manufacturing view: conform to specification
- Product view: inherent characteristics
- Value-based view: willingness to pay

How do we assess quality ?

- Is the following system modifiable?
  - Background color of the user interface is changed merely by modifying a resource file.
  - Dozens of components must be changed to accommodate a new data file format.

- A reasonable answer is
  - **yes** - with respect to changing background color
  - **no** - with respect to changing file format

Quality Attributes

The totality of *features and characteristics of a product, process or service that bear on its ability to satisfy stated or implied needs*

- The features and characteristics are called as quality attributes
  - Ex: Reliability, Robustness, Efficiency, Usability, Safety, Security, Fault-tolerance, Correctness, …..
- Useful in specifying requirements and evaluation
- Can be conflicting or supportive

Quality Attributes – for Credit risk analysis (predicting NPA?)

Types of Quality Attributes

Business Qualities	• Time to Market; Project lifetime; target market; cost and benefit; projected roll out; integration with legacy; etc.
System Qualities (includes Data and Model qualities)	• Performance, Availability, Modifiability, Testability, Usability Reliability, Security, Safety, Explainability, Fairness, Interpretability, Reproducibility, etc.
Architectural Qualities	• Conceptual integrity; Correctness; Completeness; Buildability

System Qualities

- Availability
- Performance
- Security
- Modifiability
- Testability
- Usability
- …

Availability

- Failures and faults
- Mean time to failure, repair
- Downtime

Availability Tactics

- Fault detection
  - Ping/echo
  - Heartbeat
  - Exception
- Fault recovery
  - Preparation
    - Voting
    - Active redundancy (hot backup)
    - Passive redundancy (warm backup)
    - Spare
  - Re-Introduction
    - Shadow operation
    - State resynchronization
    - Checkpoint/rollback
- Fault prevention
  - Removal from service
  - Transactions
  - Process monitor

Performance

- Event arrival patterns
  - Periodic
  - Stochastic
  - Sporadic

- Event servicing
  - Latency – Time between the arrival of stimulus and the system's response to it
  - Jitter – Variation in latency
  - Throughput – Number of transactions the system can process in a second

Performance Tactics

- Resource demand
  - Increase computational efficiency
  - Reduce computational overhead
  - Manage event rate
  - Control frequency of sampling
  - Bound execution times
  - Bound queue sizes
- Resource management
  - Introduce concurrency
  - Maintain multiple copies of data or computations
  - Increase available resources
- Resource arbitration
  - Scheduling policy
    - FIFO
    - Fixed-priority
      - semantic importance – Based on domain characteristic
      - deadline monotonic – Higher priority to shorter deadlines
      - rate monotonic – Higher priority to shorter periods
    - Dynamic priority

MODEL QUALITY

Model Quality - metrics

model: $\bar{x}$ → y

Confusion/Error Matrix ?

True Positive (TP)	**False Positive (FP)**
False Negative (FN)	**True Negative (TN)**

True Positive (Relevant and Retrieved)	**False Positive** (Irrelevant and Retrieved)
False Negative (Relevant and Not retrieved)	True Negative (Irrelevant and Not retrieved)

Precision: Within a given set of positively-labeled results, the fraction that were true positives

```
tp/(tp + fp)
```

Recall: Given a set of positively-labeled results, the fraction of all positives that were retrieved

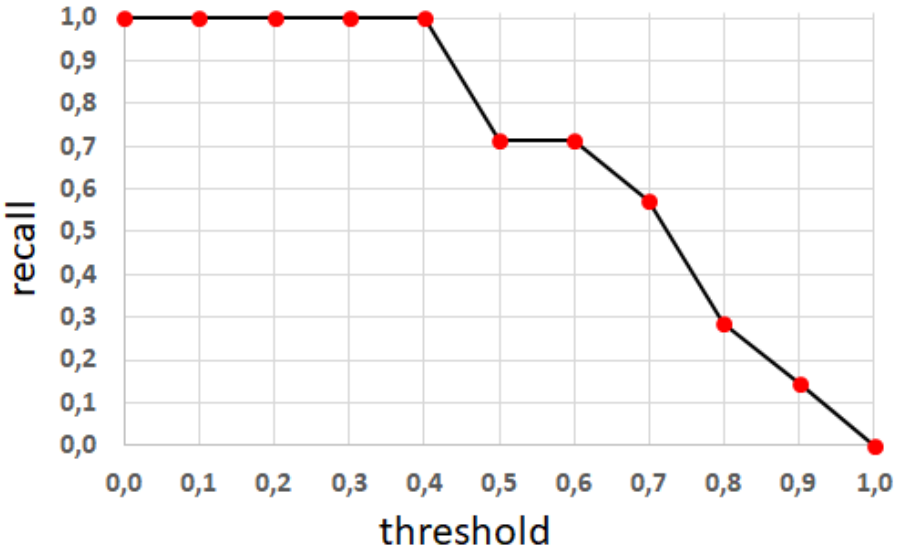
```
tp/(tp + fn)
```

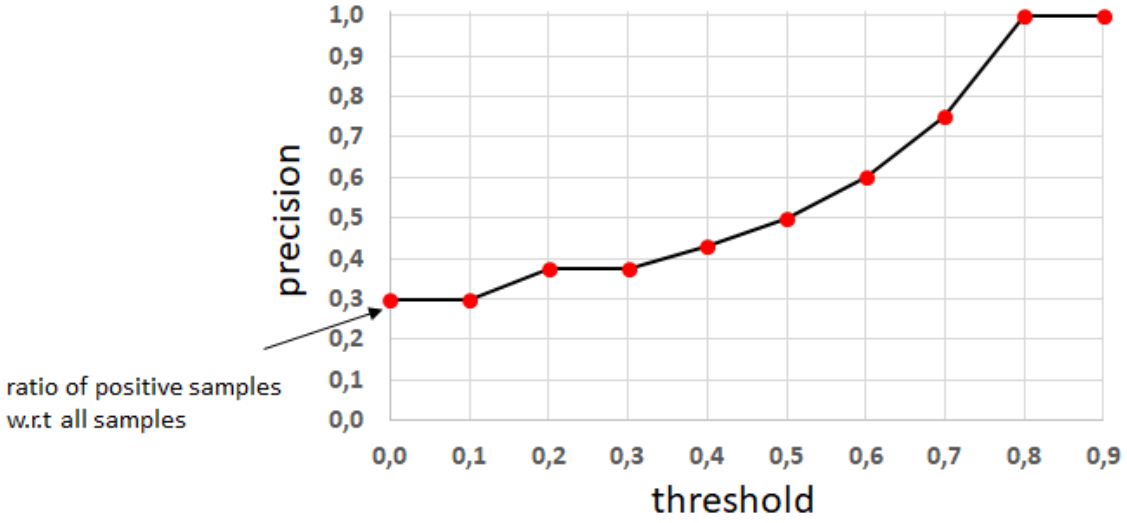
Accuracy: Percentage of correct results

```
(tp+tn)/(tp+tn+fp+fn)
```

True Positive (TP)	**False Positive (FP)**
False Negative (FN)	**True Negative (TN)**

Comparing two models (PR – AUC)





PrecisionRecall – Area Under Curve

$$\text{PR-AUC} = \int_0^1 prec(rec)\, d(rec)$$

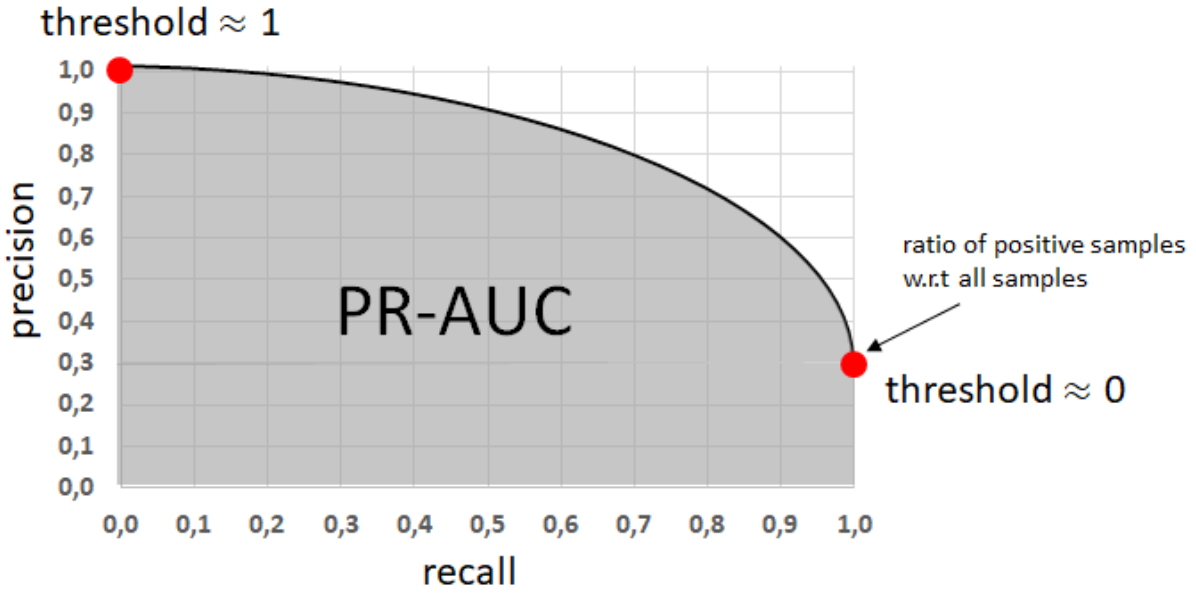


Image source: towardsdatascience.com

Example...for AUC (Receiver Operating Characteristic - ROC)

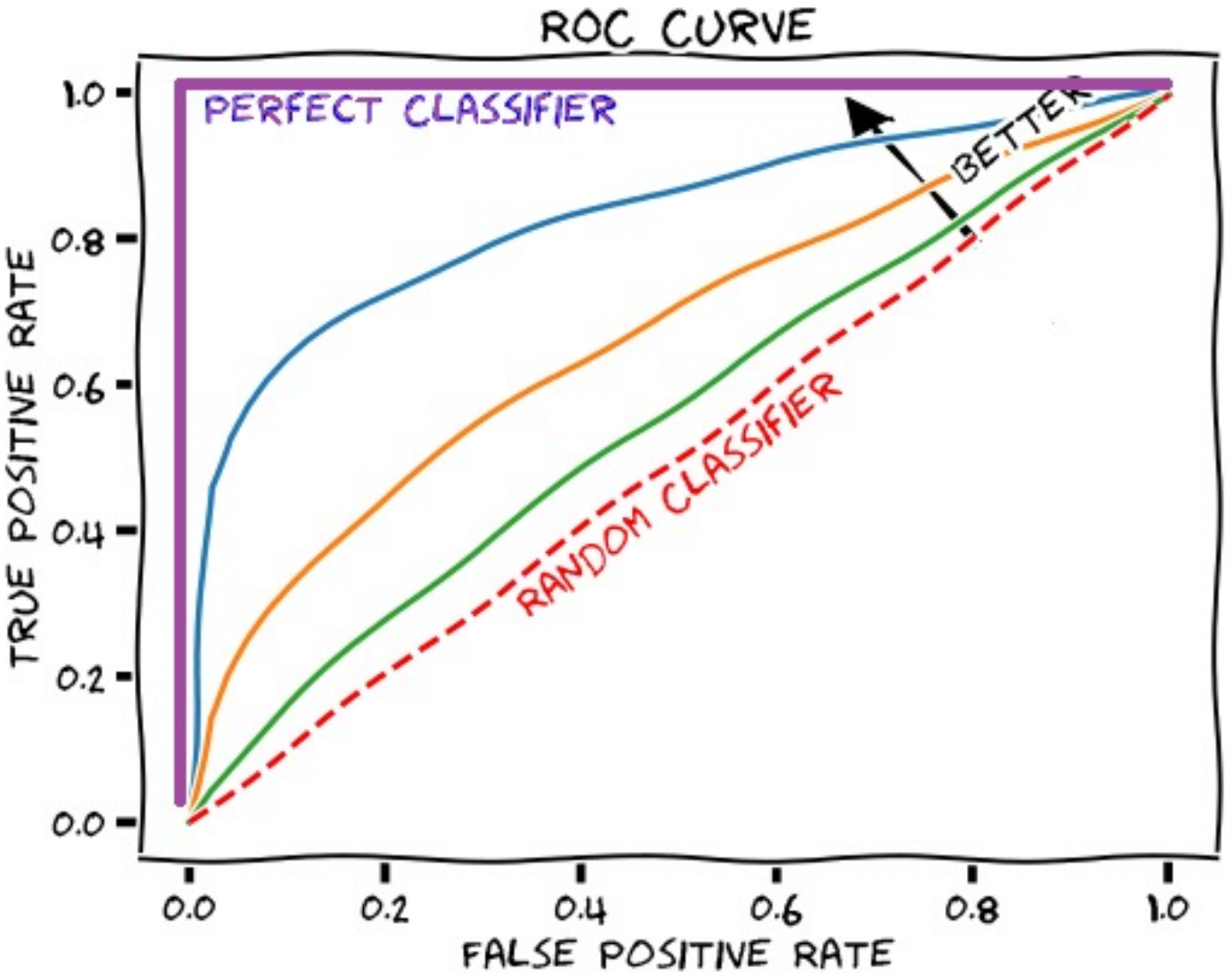


Image source: Wikipedia

Summary – Can be used tradeoff among classification problems

- Decision Tress
- Random Forest
- Gradient Boost
- SVM
- KNN
- …

What about regression problems?

Car name	Car Category	Fuel type	Car body	Drive wheel	…	Predicted price	Actual Price
A	Luxury	Diesel	SUV	4WD		22 lakhs	24 lakhs
B	Mid-level	Diesel	Sedan	2WD		15 lakhs	14 lakhs
C	Entry	Diesel	Hatchback	2WD		10 lakhs	10.5 lakhs
D	Entry	Petrol	Sedan	2WD		9 lakhs	8.8 lakhs
E	Luxury	Petrol	SUV	4WD		25 lakhs	28 lakhs

- Difficult to make a confusion/error matrix for this data set…

- Mean Absolute Percentage Error (MAPE)

$$\mathrm{M} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

A_t – Actual outcome for row t

F_t – Predicted outcome for row t

Possibly create buckets: 0-10 Lakhs; 10-20 lakhs; 20-30 lakhs;

Can we make it into a classification problem?

Evaluation Metrics for Regression models

Car name	Car Category	Fuel type	Car body	Drive wheel	…	Predicted price	Actual Price	**Residual Error (lakhs)**
A	Luxury	Diesel	SUV	4WD		22 lakhs	24 lakhs	2
B	Mid-level	Diesel	Sedan	2WD		15 lakhs	14 lakhs	1
C	Entry	Diesel	Hatchback	2WD		10 lakhs	10.5 lakhs	0.5
D	Entry	Petrol	Sedan	2WD		9 lakhs	8.8 lakhs	0.2
E	Luxury	Petrol	SUV	4WD		25 lakhs	28 lakhs	3

Other Metrics:

- MSE – Mean Squared Error. $\frac{1}{n}\sum_{t=1}^{n}(A_t - ft)^2$
- RMSE – Root Mean Squared Error $\sqrt{\frac{1}{n}\sum_{t=1}^{n}(A_t - ft)^2}$
- RMSLE – Root Mean Squared Log Error. $\sqrt{\frac{1}{n}\sum_{t=1}^{n}(log(A_t + 1) - log(f_t + 1))^2}$

A_t – Actual outcome for row t

F_t – Predicted outcome for row t

DATA QUALITY

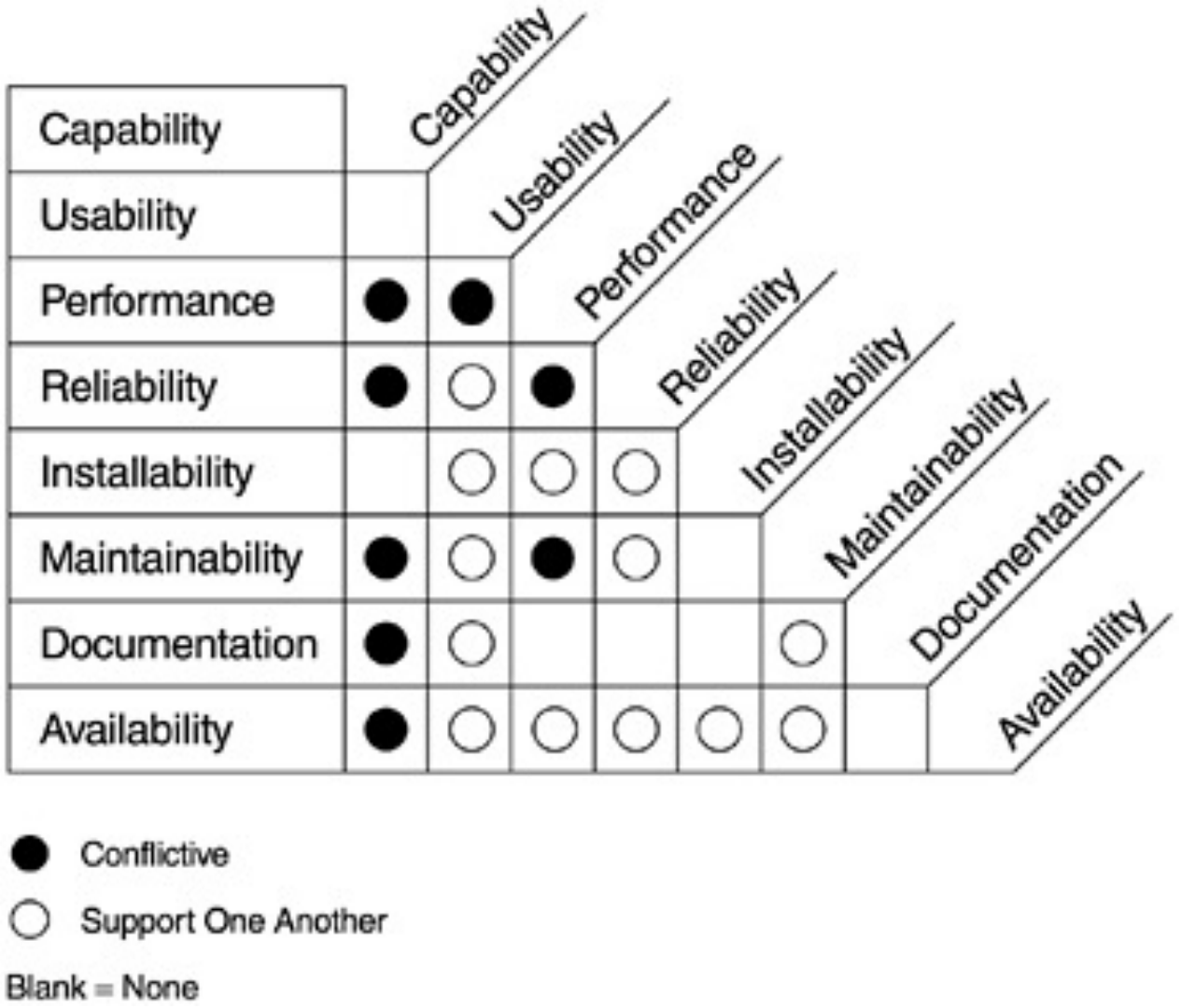
Data Quality - Attributes

- **Accuracy**
  - Depends on training data features and Ground truth labels
  - Must not have Typos, duplicates, missing data, etc.
- **Completeness**
  - Data set must be representative of how its going to be used in production – "world view" needs to be complete
- **Consistency**
  - Laebling of data has to be consistent – as it might involve different type of workers (lablers)
- **Timeliness**
  - Latency between when an event occurred and when it was added to your database

Trade-offs

Achieving Qualities requires trade-offs

- Depending on the context, systems qualities attributes may conflict or compliment



Courtesy: Stephen H. Kan

Imagesource: extremetech.com

Lane Assist Example – what qualities/constraints are important ?

Tradeoff – Where should the component that implements intelligence live?

- Lane Assist Example
  - Device in the Car (have some portable device or device fitted to the car?)
  - Phone (use a smartphone for processing?)
  - Cloud (Processing/Inferencing on cloud?)

What qualities/constraints are important (from system perspective with ML part as a component) ?

Some typical considerations:

- Data input needed for the model
- Data output produced
- Energy consumption/speed of the model
- Latency and other qualities
- Model update frequency
- Operational cost and Telemetry opportunities
- Offline operation
- …

Where should the model (intelligence) live? – Common patterns

- Static Intelligence in the product
  - Intelligence is in the client - without connecting to some service
- Client side intelligence
  - Intelligence is in the client - connects to a service for intelligence updates and telemetry
- Server centric intelligence
  - Intelligence is in the service and requires service call every time execution is needed
- Back-end (Cached) intelligence
  - Executes intelligence offline on common contexts and delivers answers via caching
- Hybrid intelligence
  - Combines multiple approaches – most large scale intelligent systems tend to be hybrid

Static Intelligence in the product

Advantages:

- Cheaper to engineer
- Good enough for many problems
- No need to worry about the gap between users and intelligence creators
- Don’t need to worry about continuous intelligence updates

Disadvantages:

- Difficult to update intelligence
- Risk of unmeasurable intelligence errors
- No data to improve intelligence

Client side Intelligence

Advantages:

- Allows more resources to be allocated for intelligence execution
- Idle resources on client side may not cost much
- Latency at runtime is negligible (esp. since service calls don't exist)

Disadvantages:

- Deciding when and how to push new intelligence to clients
- Keeping every client in-sync with updates
- Resources may be constrained
- Exposes intelligence to the world

Server centric Intelligence

Advantages:

- Allows for models to be updated quickly
- Controlled updates are possible
- Telemetry and Monitoring is easier

Disadvantages:

- Latency in intelligence calls
- Service infrastructure and bandwidth costs
- User bandwidth costs
- Cost of running servers that can execute intelligence in real time

Back-End (Cached) Intelligence

Advantages:

- Effective when analyzing finite number of things
- Can spend resources and time on each intelligence decision (as the processes are running in the backend)
- Some intelligence can live in services; some parts can be in clients too

Disadvantages:

- Expensive to CHANGE the models (as cached results need to be recomputed)
- Works only when context can be used to look up the relevant intelligence

Hybrid Intelligence

Advantages:

- Can take advantage from each approach (do a mix and match)

Disadvantages:

- Can mask the weaknesses of various components
- Hard to determine source of a mistake – when multiple are involved in the intelligence

Architectural decisions – comparing qualities/considerations

Intelligence location	Latency in Updating intelligence	Latency in Execution	Cost of Operation	Offline Operation
Static	Poor	Excellent	Cheap	Yes
Client-side	Variable	Excellent	Based on update rate	Yes
Server-centric	Good	Variable, but includes internet round-trip	Service infrastructure and bandwidth can have significant cost; may cost users in bandwidth	No
Back-end (cached)	Variable	Variable	Based on usage volume	Partial

Can help answer questions like:

- ✓ If your application needs to work on an airplane over the Pacific ocean (with no internet), what are the options for intelligence placement
- ✓ An application needs to function on an airplane, but the primary use case is at a user's home? What are some options to enable the system to shine in both settings?

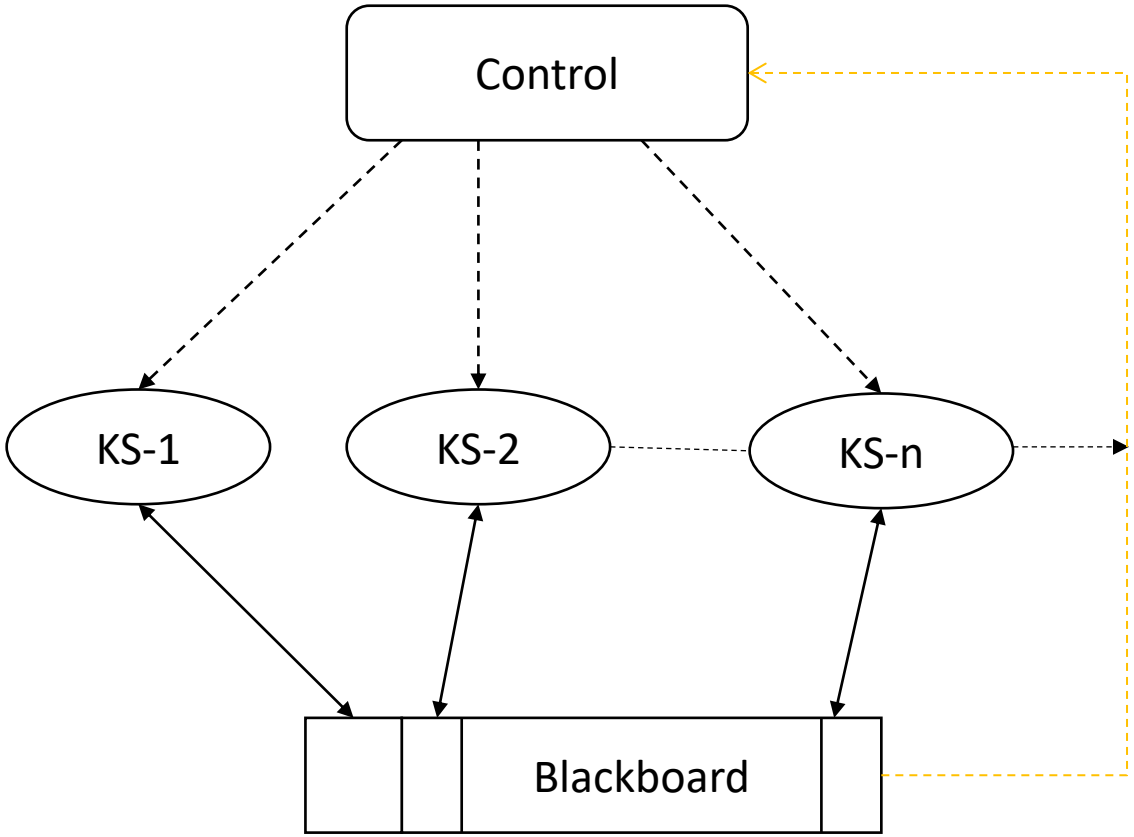
How to realize qualities ?

- Qualities are indirect in nature. Quite a few times they tend to be emergent properties

- Depends on how the functionality is designed
  - How the components are connected to each other
  - What are the constraints and how do they affect the components and relationships
  - Designs help with tradeoff analysis, reason about the quality attribute

- Need to design the system taking qualities into consideration
  - Can Patterns help (UI, Architecture, Detailed design)?
  - **What are patterns?**

Patterns & Anti-patterns (example)



Manual Configurations Management

image@infoQ

Architectural patterns (styles)

- Distributed
- Event-driven
- Batch
- Pipes and filters
- Repository-centric
- Blackboard
- Layered
- MVC
- IR-centric
- Bridge
- Microkernel
- Reflection
- Microservices
- …

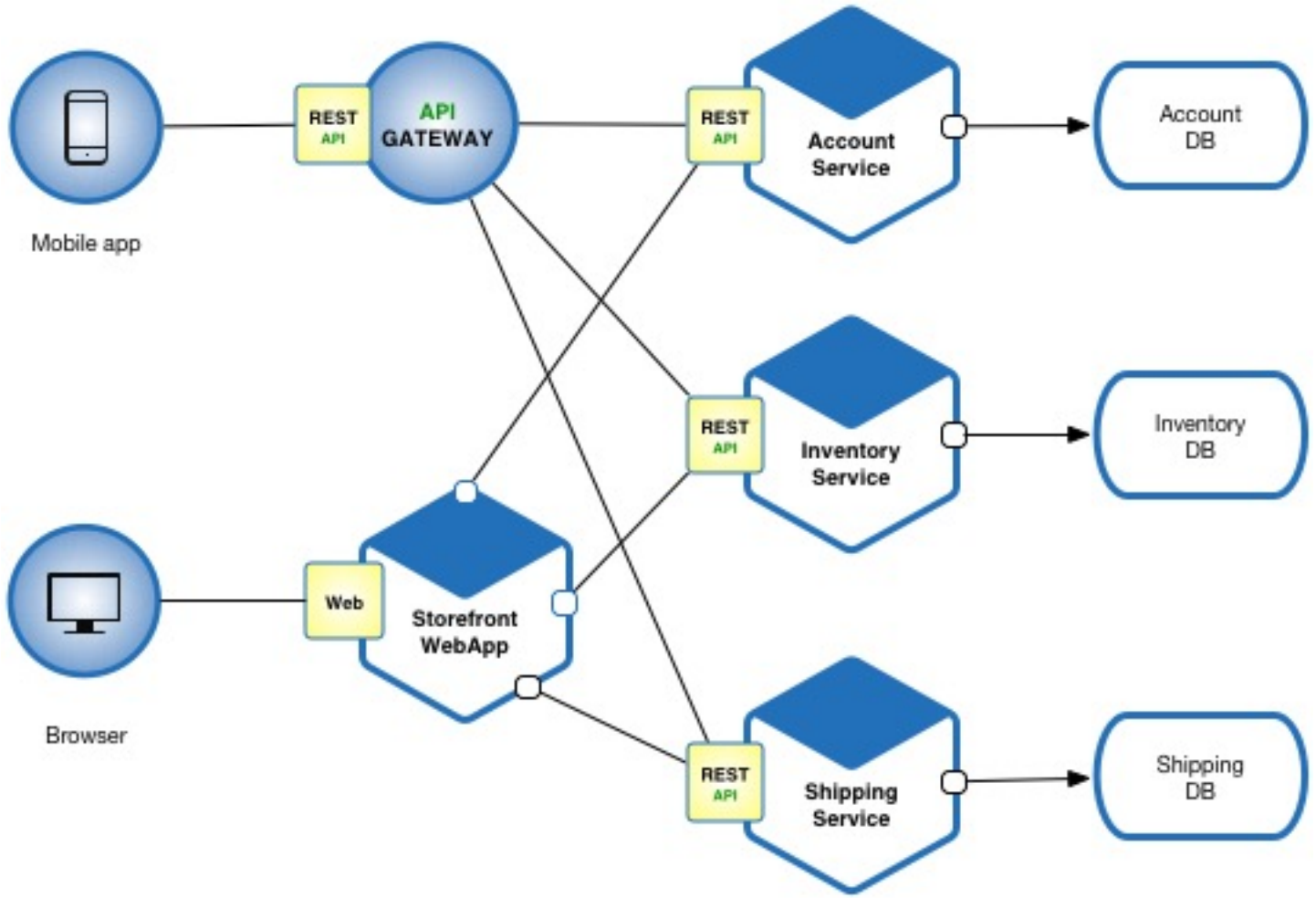
Whatis an Architectural Style?

Microservices

- Service oriented architecture composed of loosely couple elements that have bounded contexts
  - Allows for independent updates of services; Loose coupling ensures updating one service doesn't need updating other services
  - Bounded context meant microservices are self contained. Services can be updated without knowing the internals of other microservices

- How to decompose (how to identify service boundaries)
  - Services can be defined based on business capabilities
  - Services are highly cohesive
  - Loose coupling

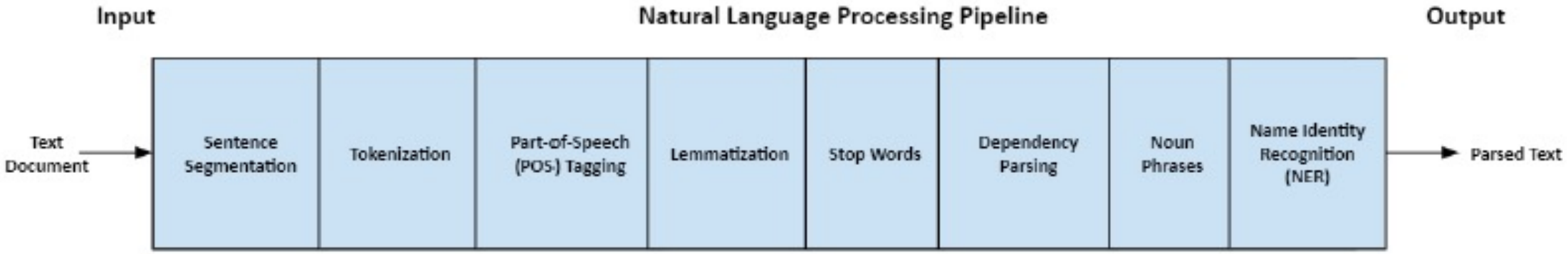
Microservices - Simple Ecommerce example



Source: Microservices.io

Pipe and Filter examples

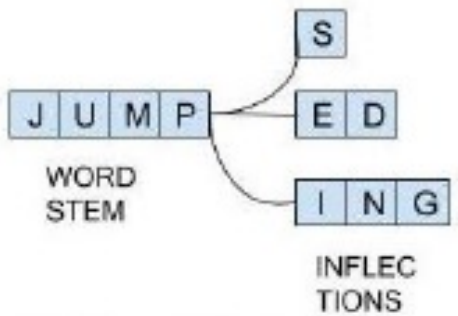
- Natural Language Processing pipeline



Sentence Segmentation

Hello world. This blog post is about sentence segmentation. It is not always easy to determine the end of a sentence. One difficulty of segmentation is periods that do not mark the end of a sentence. An ex. is abbreviations.

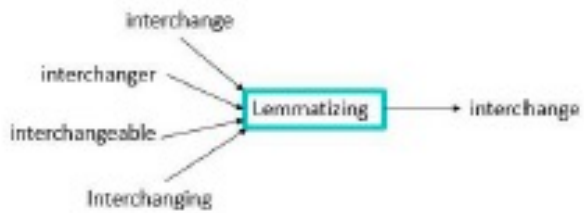
- Hello world.
- This blog post is about sentence segmentation.
- It is not always easy to determine the end of a sentence.
- One difficulty of segmentation is periods that do not mark the end of a sentence.
- An ex. is abbreviations.

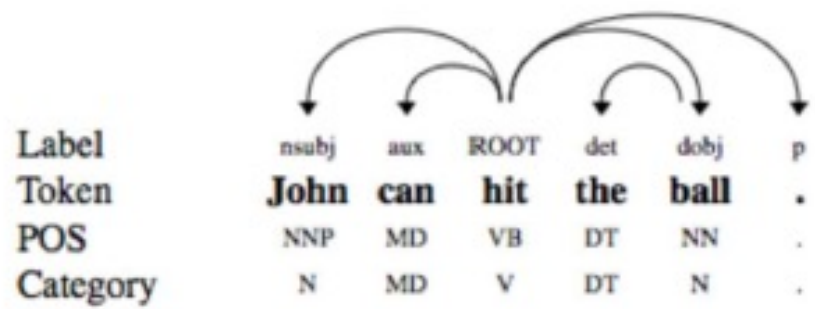


Word stem and its inflections (Source: Text Analytics with Python, Apress/Springer 2016)

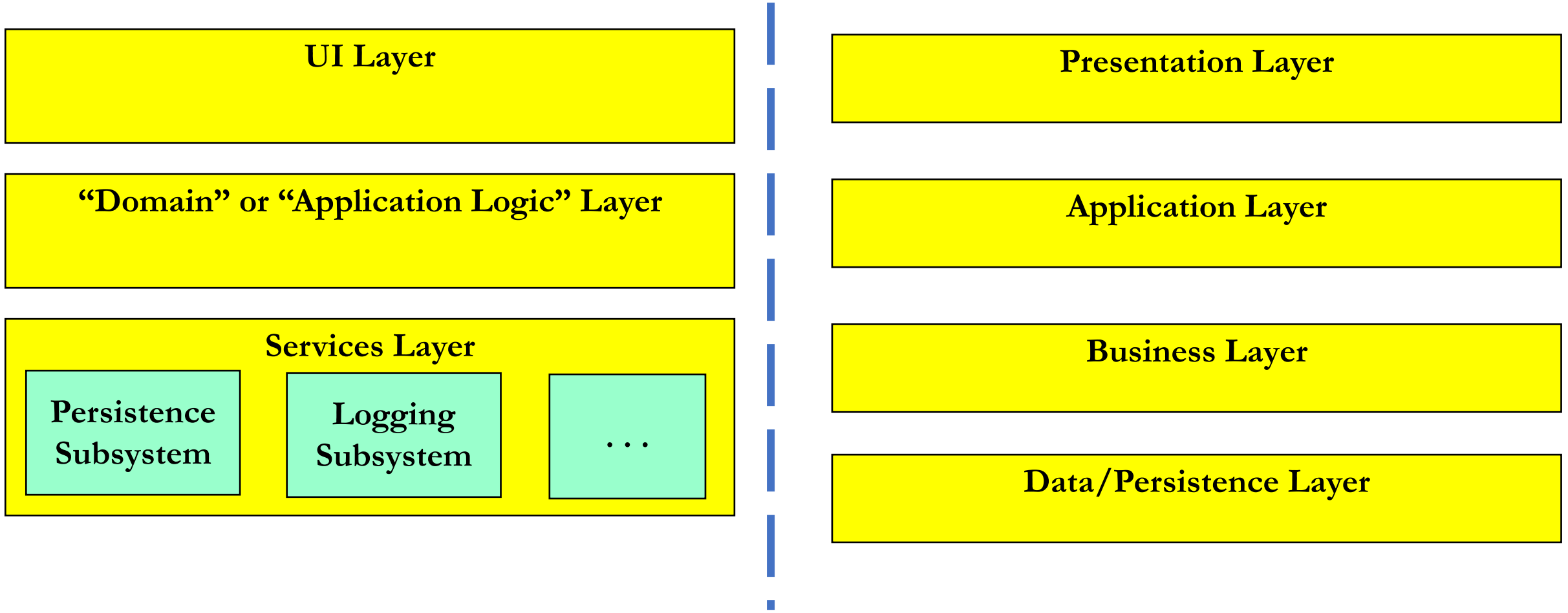
's

Lemmatizing

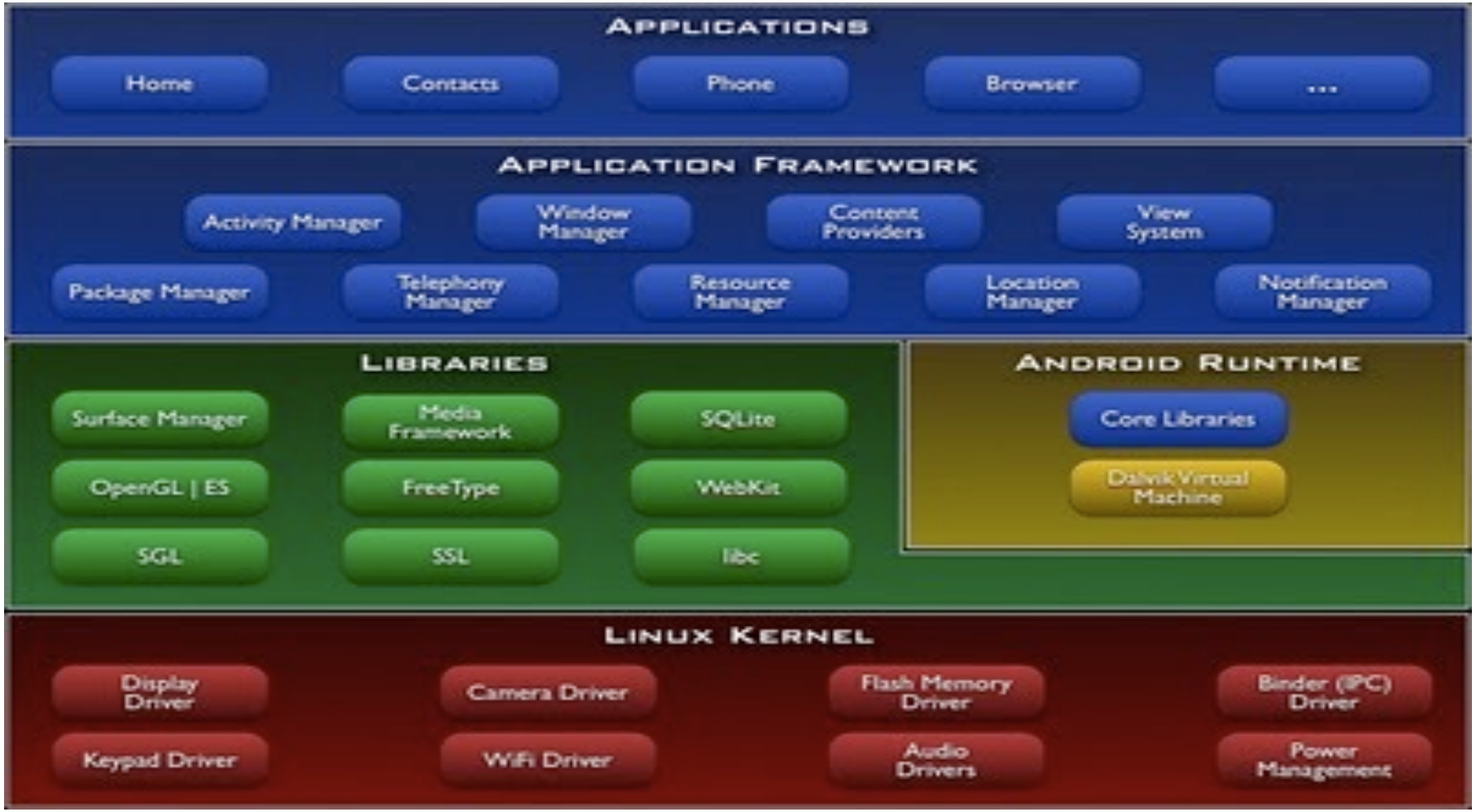




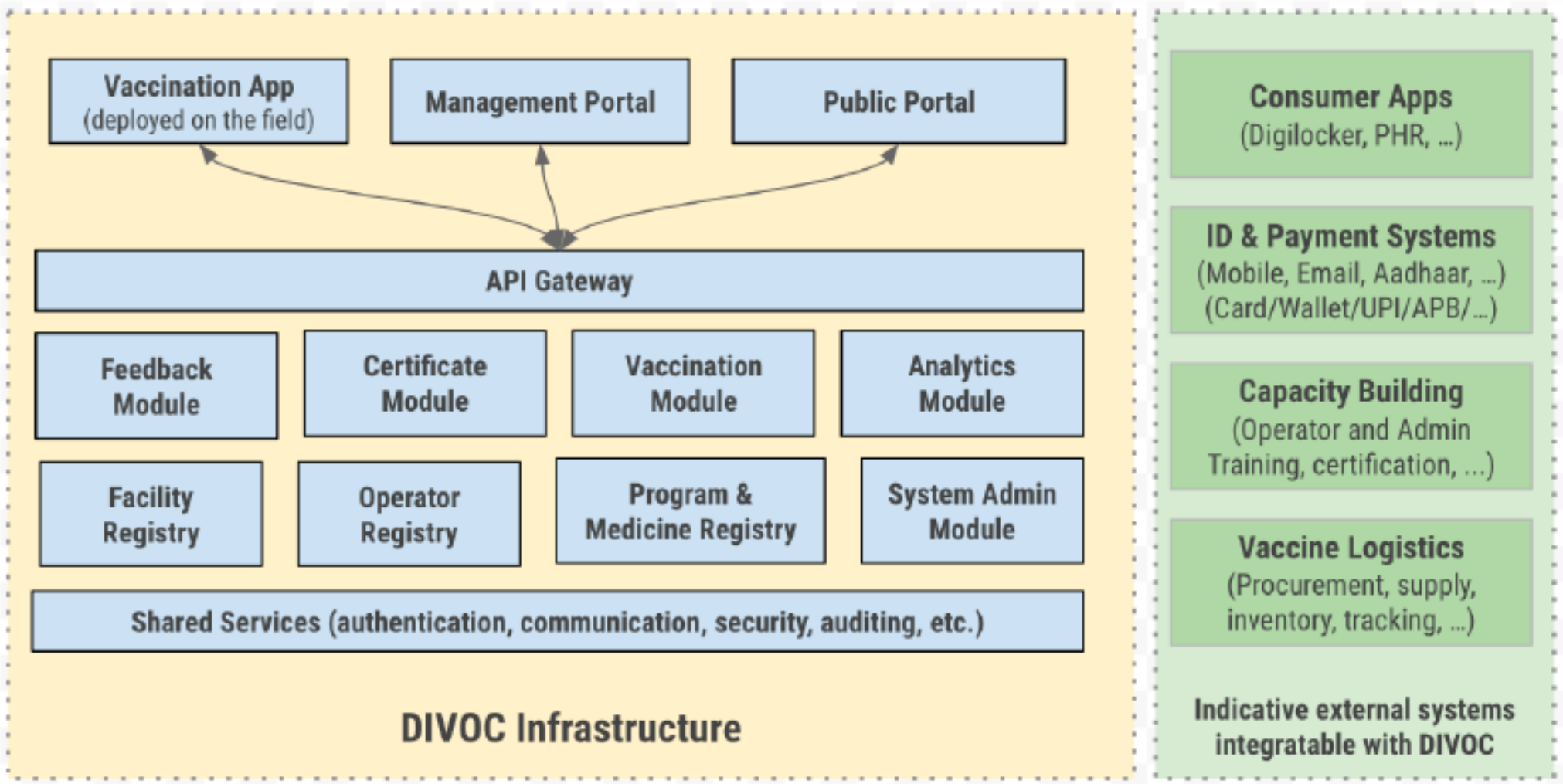
Layered Pattern – (Multi-tier architecture)



Trivia: Architecture of…?

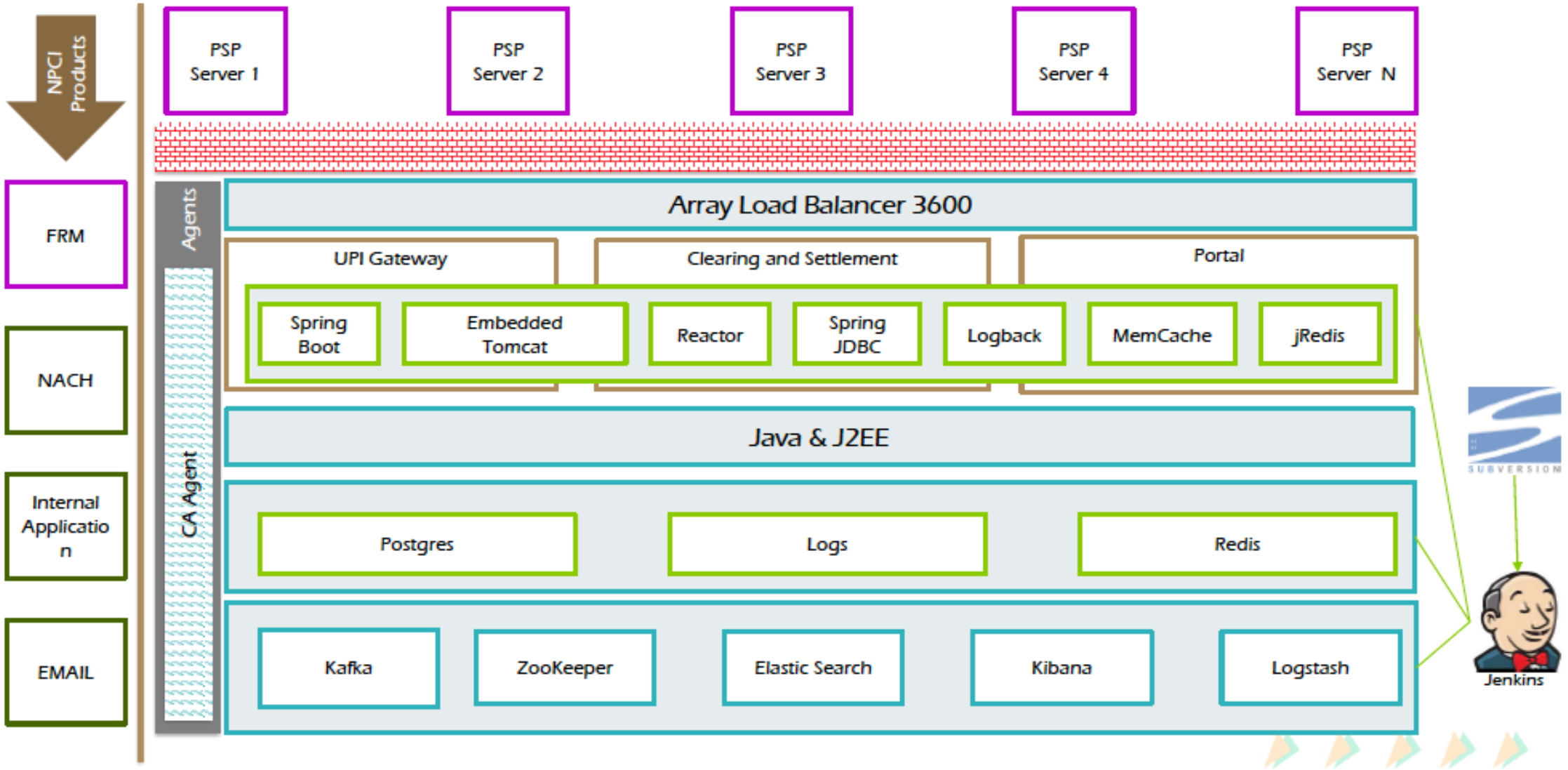


Trivia: (High level) Architecture of … ?



Digital Infrastructure for Verifiable Open Credentialing

Trivia: Technical architecture of … ?



Guiding principles

Develop a comprehensive, efficient and stable product solution which will be real-time and equipped with the latest technology for efficiently handling the projected volumes and reducing the time taken for transaction processing, clearing and settlement.

Modern Application: New generation software solution

Open Architecture: Platform architecture should be open, flexible and dynamic in nature.

Ease of Maintenance: Modular and configurable

High Capacity and Throughput: High throughput and capacity; with the ability to scale up further to meet future requirements.

High Availability: Should have high availability (99.9999%).

Scalability: Provides horizontal, vertical and linear scalability without inherent bottle necks and core design changes.

Fault-Tolerance: Fault-tolerant design Highly configurable and parameterized..

Monitoring Capability: Has Adequate real-time monitoring capabilities.

Analytical : Provide insides, trends and predicts.

Serverless Architectural pattern

Serverless architectures refer to applications that significantly depend on third-party services (knows as Backend as a Service or "BaaS") or on custom code that's run in ephemeral containers (Function as a Service or "FaaS")

Write your code and run your applications without provisioning or managing anything on the server side.

❖ Serverless in a cloud environment means
- Function as a unit of application logic
- No Servers to provision or manage
- Scales with usage
- Pay for what you use and Never pay for idle
- Built in availability and fault tolerance
- Decreased time to market

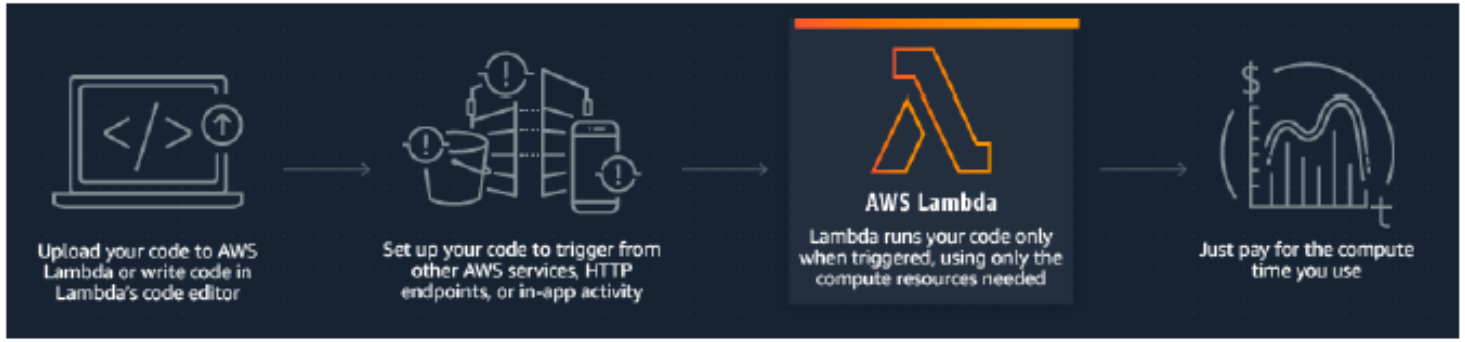
❖ How does it work?
- Function as a unit of application logic
- New invocation based on demand
- Events trigger a function

❖ Serverless Providers
- AWS Lambda
- Google Cloud Functions
- Microsoft Functions
- IBM Openwhisk

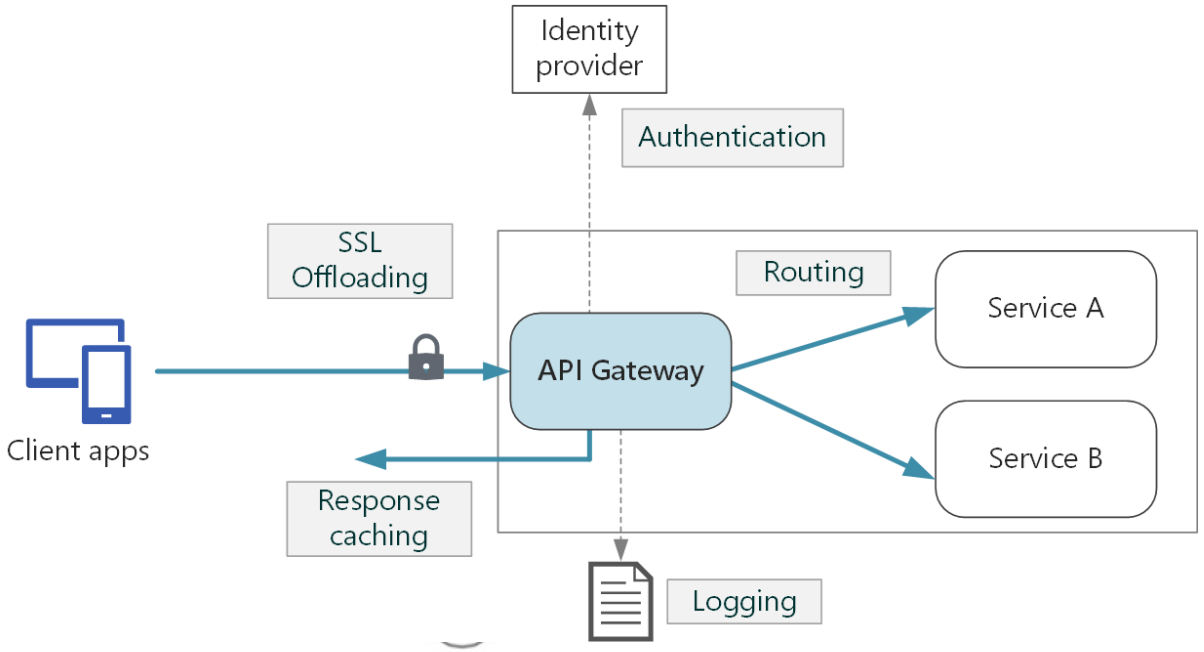
❖ Usecases: Serverless Website | App Authentication | Mass Emailing | Real-time Data Transformation | CRON Jobs | Chatbot | IoT | image conversion | file compression

❖ Industry: Reuters Processes nearly 4000 requests/sec, Coco Cola, ...

❖ Supports Python, Go, Java, C#, NodeJS, ...



Gateway Routing Architecture - Example



API Gateway

Implementing Microservice ML patterns

- Azure Machine learning
- AWS Sagemaker
- Google Kubernettes Engine

Image@Microsoft Azure

Anti-patterns

Antipatters ➔ Inappropriate solution

- Design by committee
- Abstract Inversion
- Vendor Lock-in
- Warm bodies
- Cover your assets
- Reinvent the Wheel
- Interface bloat
- Ambiguous viewpoint
- Big ball of mud
- Gold plating
- Inner-platform effect
- Magic Pushbutton
- Race hazard
- Stovepipe

Do you see Antipatterns? It is time to refactor /rearchitect !

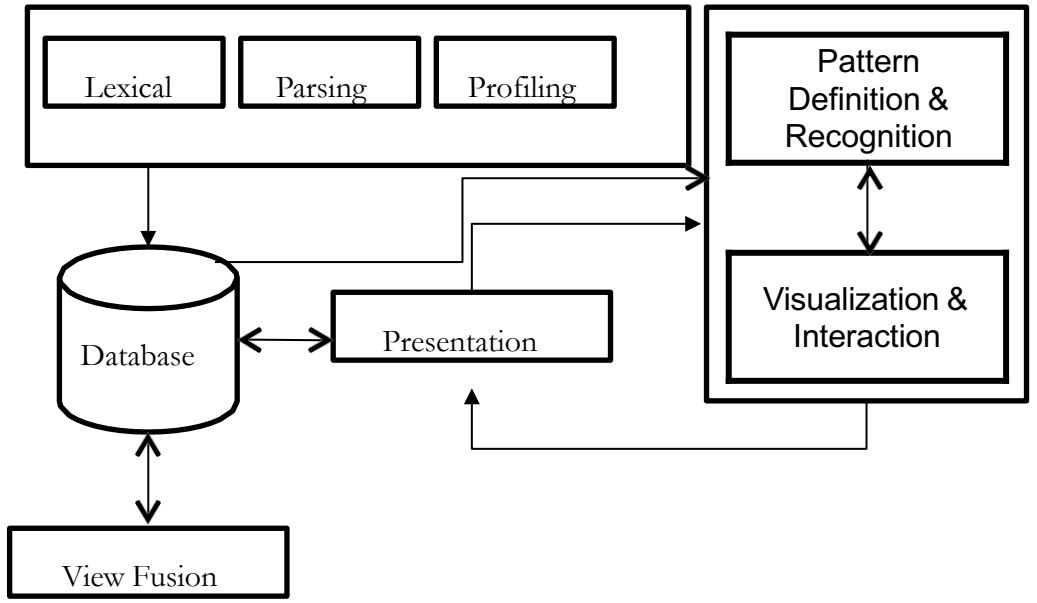
Around 2615 BC… the Bent Pyramid (Imhotep's son)

Do you see Antipatterns? It is time to refactor /rearchitect !

Ray and Maria Stata Center for Computer, Information, and Intelligence Sciences, MIT

Reconstructing Architecture

- **Information extraction**:
  - Collect data
  - Analyze artifacts – code, documents & execution traces
  - Extraction - Files, Functions & Variables
  - Program Comprehension
  - Validate results
- **Database construction**:
  - Convert to standard form
  - Design database to optimize queries
  - Consider saving most used queries as separate tables



- **View fusion**:
  - Combine information – fill in missing information, resolve ambiguities & correct errors
- **Reconstruction**: Apply abstraction

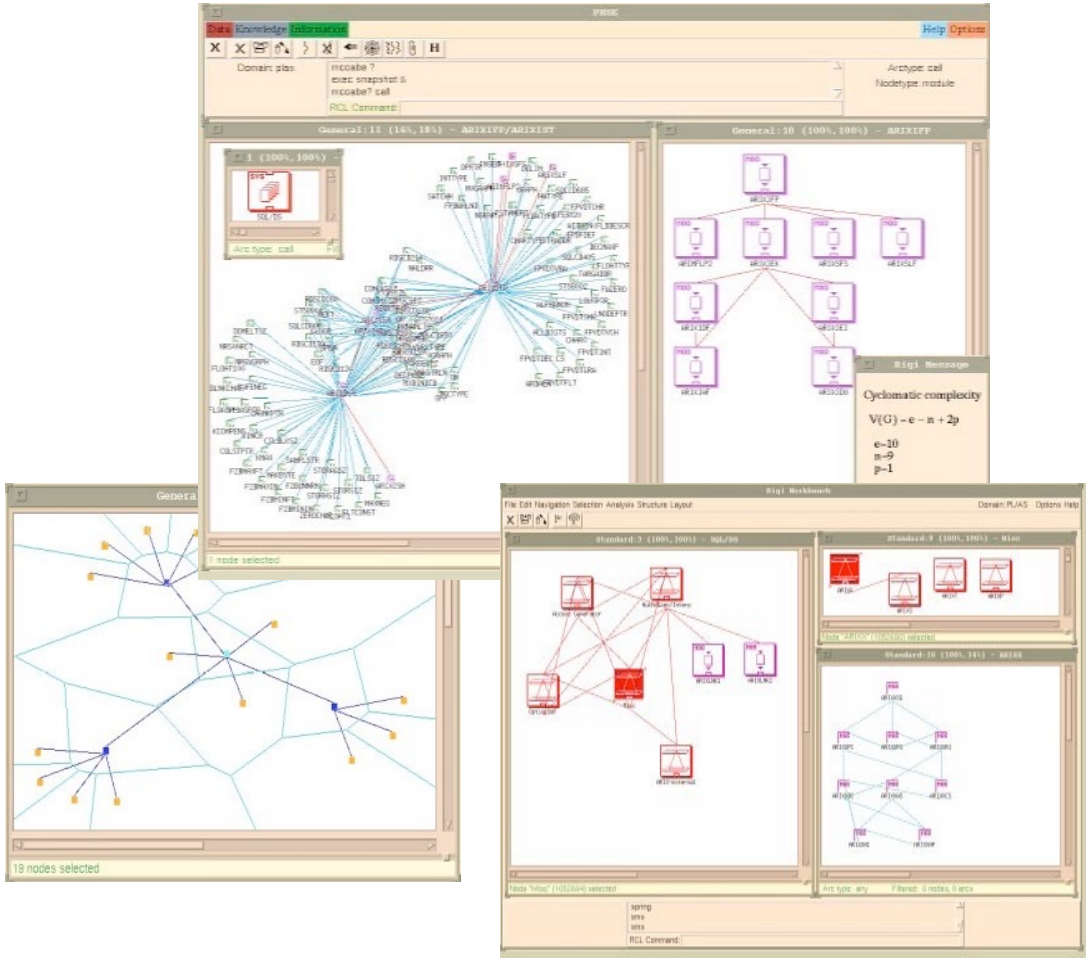
Reconstruction - Visualization

- Scope
- Abstraction
- Specification Methods
- Presentation
- Tools
  - Static Visualization
    - Code Crawler, Rigi, Seesoft, Sniff++, Revolve
  - Dynamic Visualization
    - Imagix4D, Polka

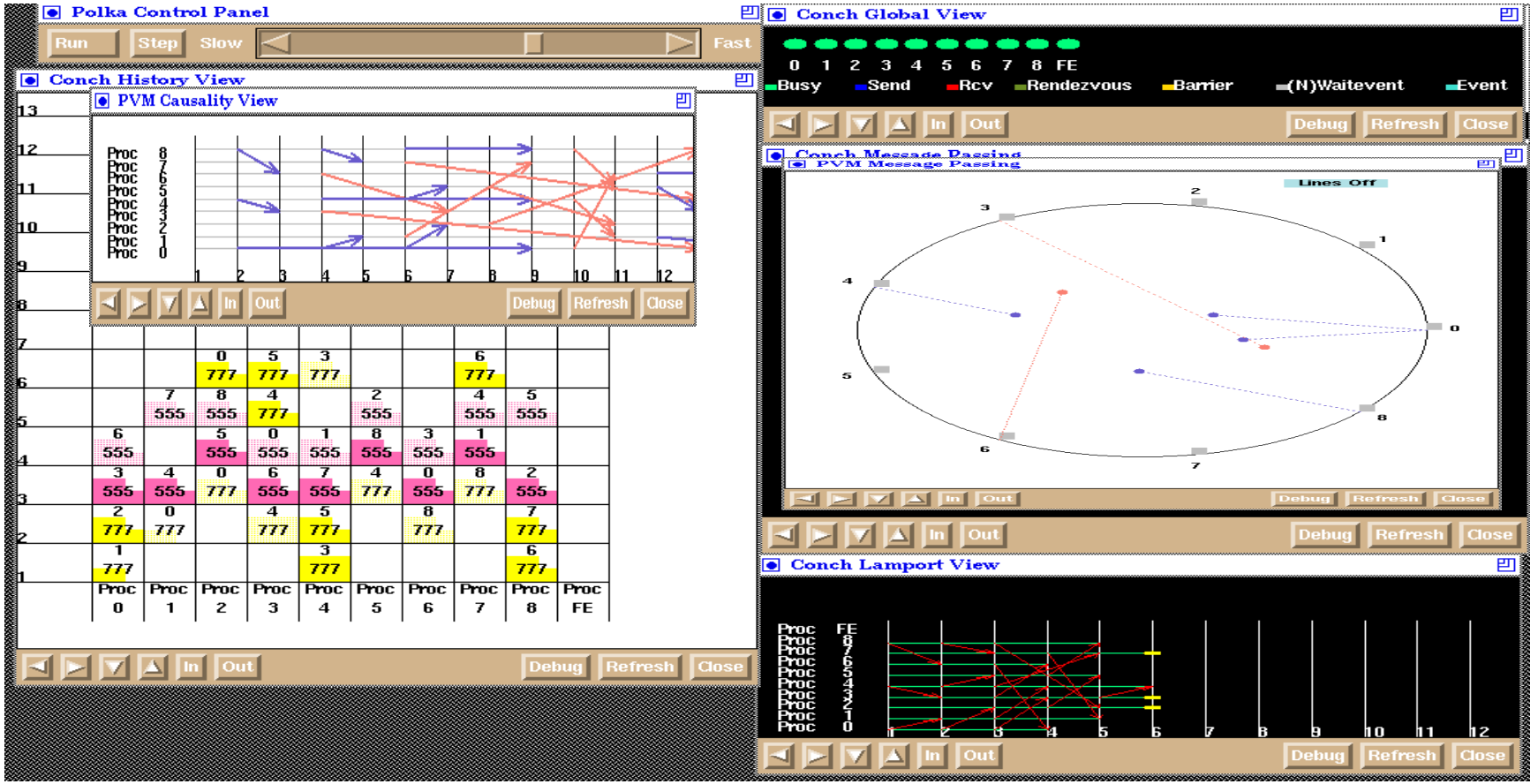
Software Visualization – Static - **Rigi**

- A research and public domain visualization tool, user programmable, analysis for: C, C++, COBOL, PL/AS, LaTeX

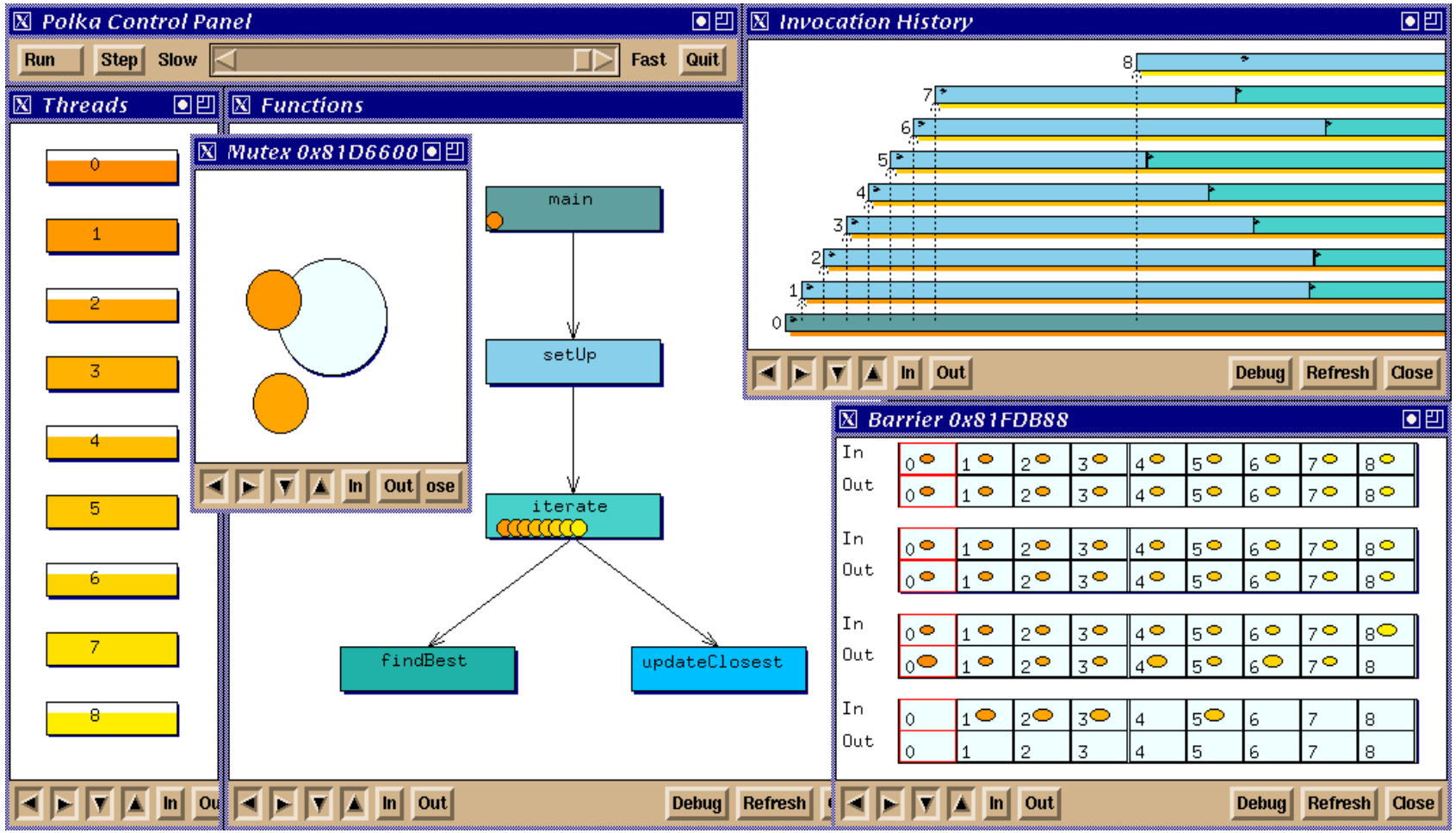
Artifact	Icon representation
system	
subsystem	
module	
proc	
data	
struct	
member	



Software Visualization – Dynamic - POLKA



Message Passing View, History View,



Threads View, Function View, History View, Mutex View, Barrier View