Cracking System Design Interview

Define Functional Requirements

Summarise functional requirements you are able to understand

Suggest other functional requirements you can think are important

- very Important, as this gives interviewer an idea that you can extend a requirement

Ask if there specific functional requirement interviewer is interested in?

- Confirming things are very good, it makes interviewer feels part of the problem and indicates your collaborative behaviour

Prioritise: Suggest priority of requirements according to you and confirm if interviewer agrees with it

- Again very important, as this gives interviewer an idea that you can set priority of your tasks.
- Note: Confirmation

Define Non Functional Requirements

- Latency
  - Latency requirements for Read requests
  - Latency requirements for write requests
- High Availability
  - Yes
  - No
- Consistency
  - Strict?
  - Eventual?
- Scalability
- Ask if there are any specific non functional requirement interviewer wants to cover?
  - Recoverability
  - Maintainability
  - Reliability

Define Extended Requirements

Analytics

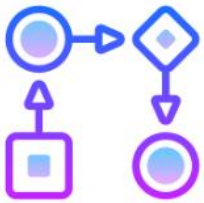
Monitoring

Alerting

Ask if there are any other extended requirements interviewer have in his head?

Like I told you earlier, don't forget to ask :)

Define workflows



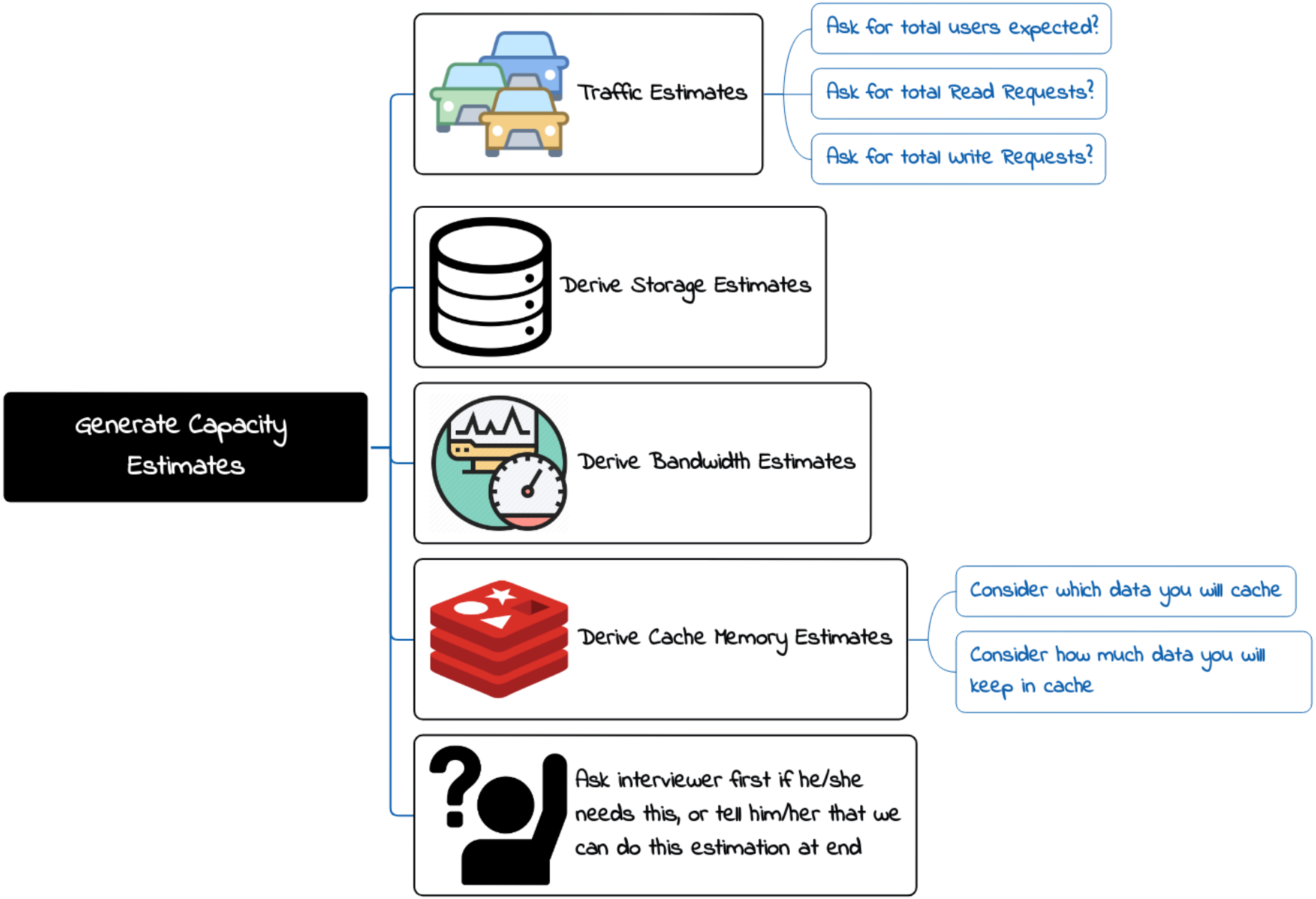
write workflow for all important functionalities

Think in terms of user using your application to define your workflows.

No need to define application components/microservices at this stage

write down basic workflows of background processes

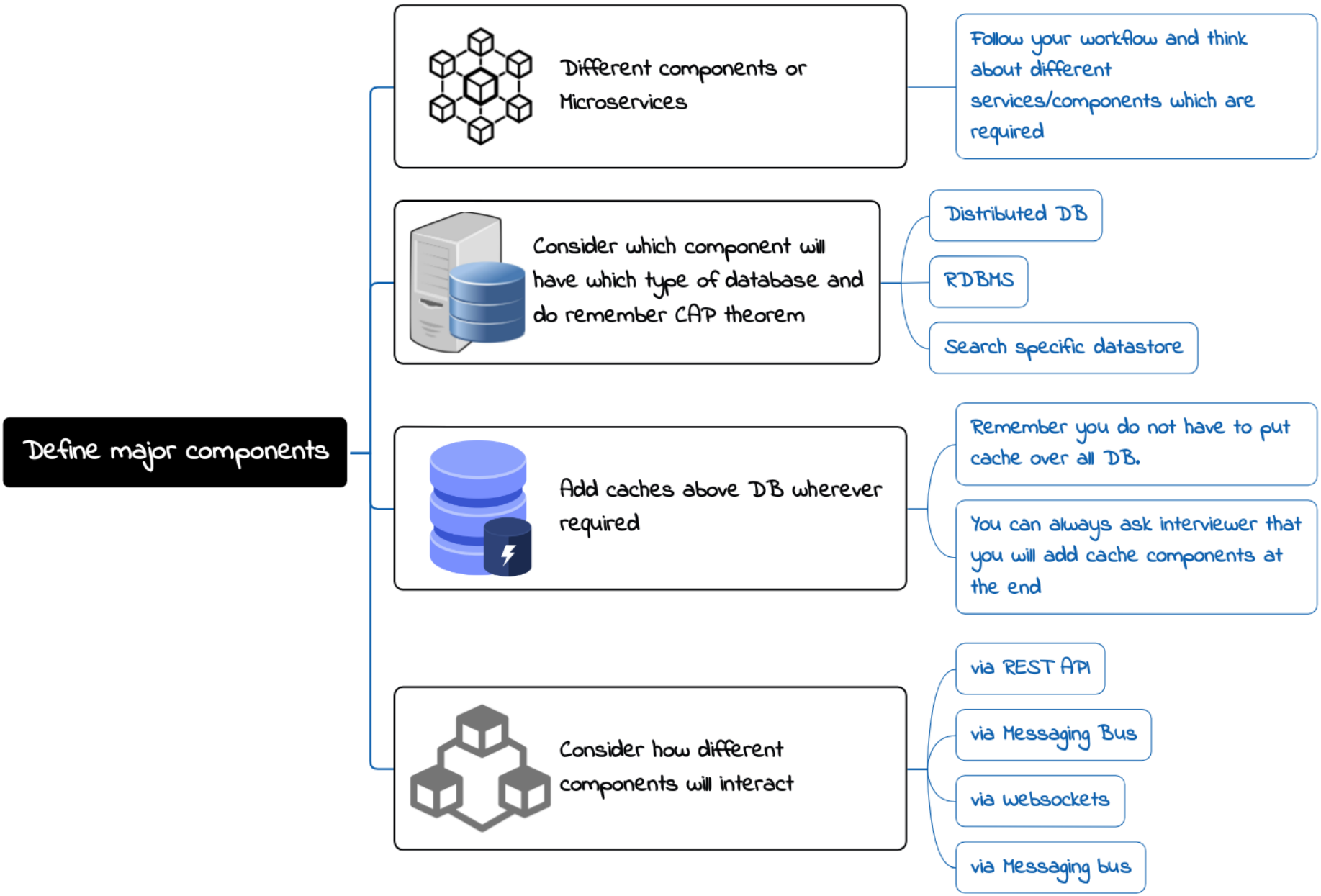
Think about major background processes you may need


Generate Capacity Estimates
Traffic Estimates
Ask for total users expected?
Ask for total Read Requests?
Ask for total write Requests?
Derive Storage Estimates
Derive Bandwidth Estimates
Derive Cache Memory Estimates
Consider which data you will cache
Consider how much data you will keep in cache
Ask interviewer first if he/she needs this, or tell him/her that we can do this estimation at end

Define System APIs

Define major system APIs

- Operation Name
- Input Parameters
- Output Parameters
- If RESTful API then define GET, PUT, POST, DELETE or PATCH


Define major components
Different components or Microservices
Follow your workflow and think about different services/components which are required
Consider which component will have which type of database and do remember CAP theorem
Distributed DB
RDBMS
Search specific datastore
Add caches above DB wherever required
Remember you do not have to put cache over all DB.
You can always ask interviewer that you will add cache components at the end
Consider how different components will interact
via REST API
via Messaging Bus
via websockets
via Messaging bus

Extend your design

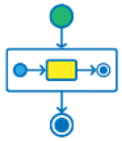
API Gateway

Load balancer

CDN

Circuit Breaker

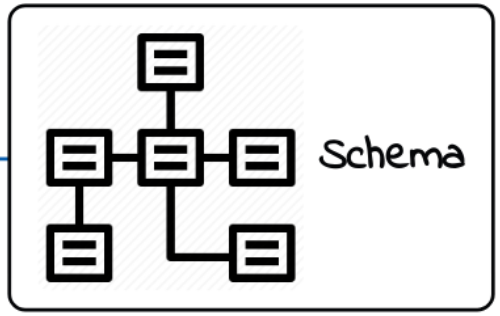
Request Throttling

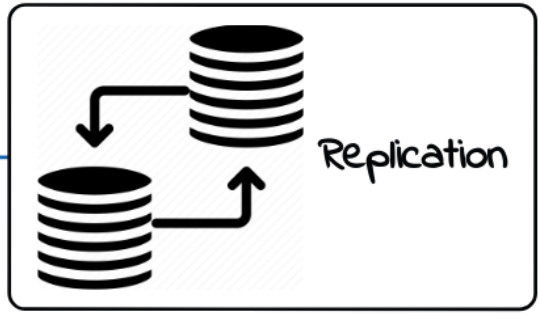


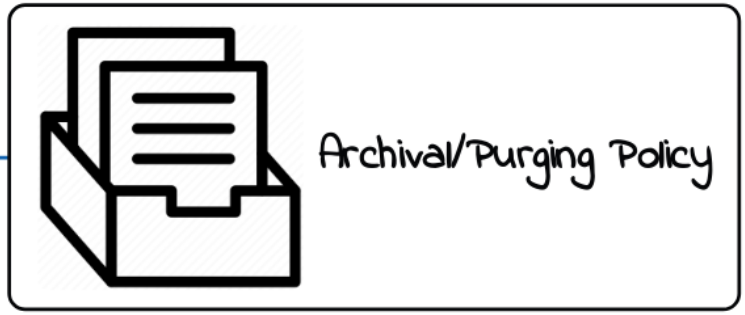
State machine or workflow system

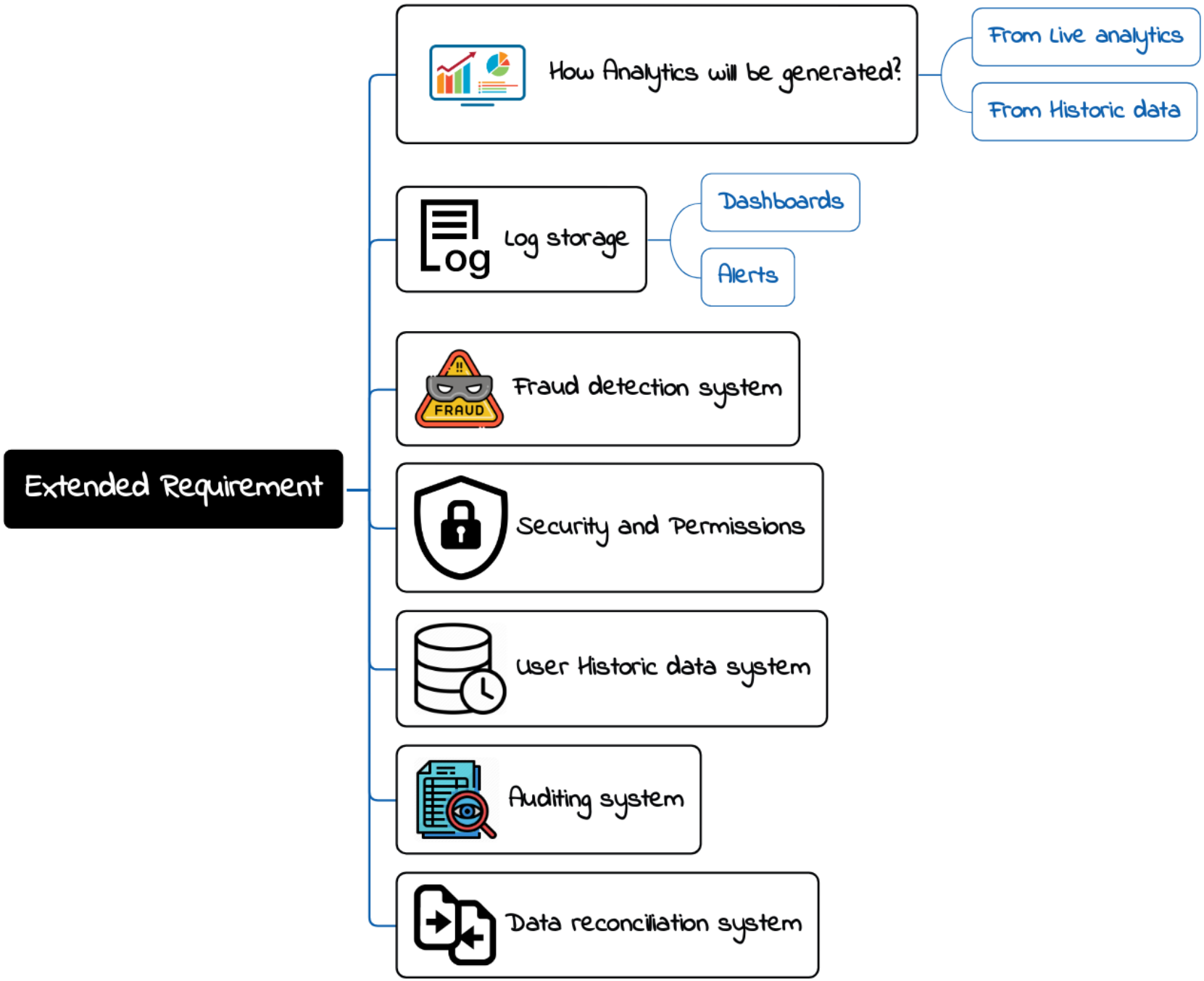
Cache Eviction

Database details








Extended Requirement
How Analytics will be generated?
From Live analytics
From Historic data
Log
Log storage
Dashboards
Alerts
FRAUD
Fraud detection system
Security and Permissions
User Historic data system
Auditing system
Data reconciliation system

Everything Else

- Deployment
  - Containers
  - Kubernetes
  - Service Mesh
- How many environments required?
  - Dev
  - UAT
  - Pre Prod
  - Production
  - Prod Parallel
- CICD
- Testcases
  - Unit Testcases
  - Functional Testcases
  - Integration Testcases
- Code Quality
  - Branching Strategy
  - Sonar Integration
  - Code Reviews