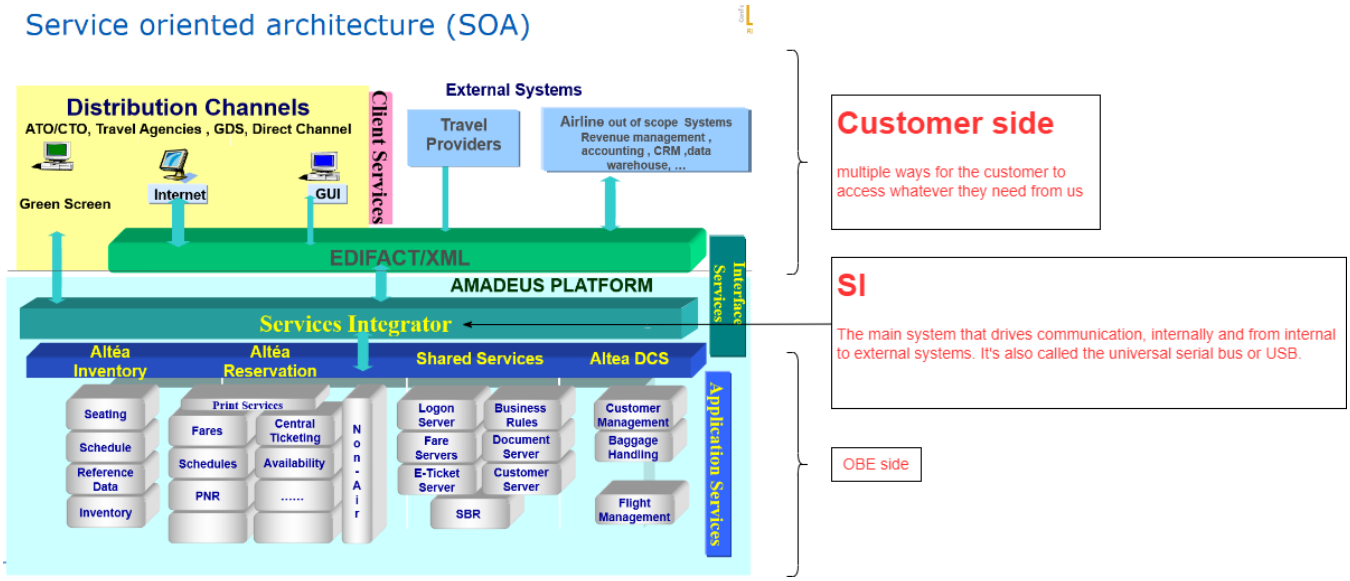
OBE introduction

presentation

The Open Back end concept (~09:00)

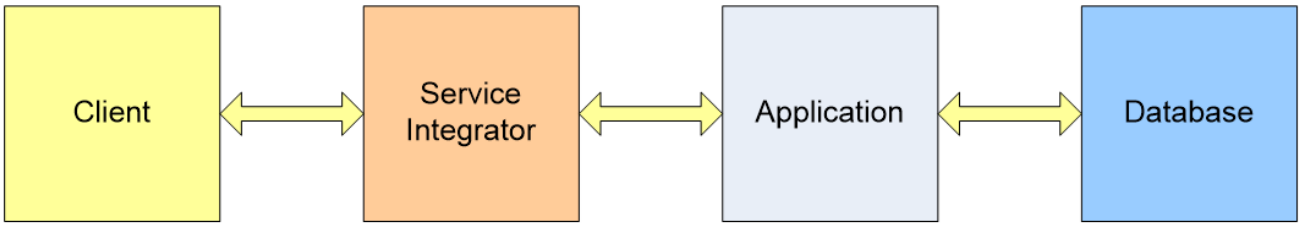


Service Oriented Architecture Concept

When we moved from TPF to Open systems, we shifted from a big monolithic application to a service oriented architecture. What that means in simple terms is that each OBE app is just a summary of certain services. Every application provides a set of unique services. Emphasis on unique: each service is only available once in our system. With this in mind, we can easily understand that applications will want to talk to each other (to provide services to each other). The key part of SOA is that when application communicate, they always have to go through the SI. FYI, a communication between two apps is usually a request for a service and then a reply. Another thing to keep in mind is that each service is a basic processing unit: EDIFACT Message, Cryptic, Webservice Message.

Open Back-End Application Architecture

A very simplified overview of how the architecture works: A client (either internal or external) sends a request. The request is always intercepted by the SI who will forward it to the application responsible for that service, which might or might not use a database to fulfill that service request and replies back. Base on the response, the client might kick off other requests.

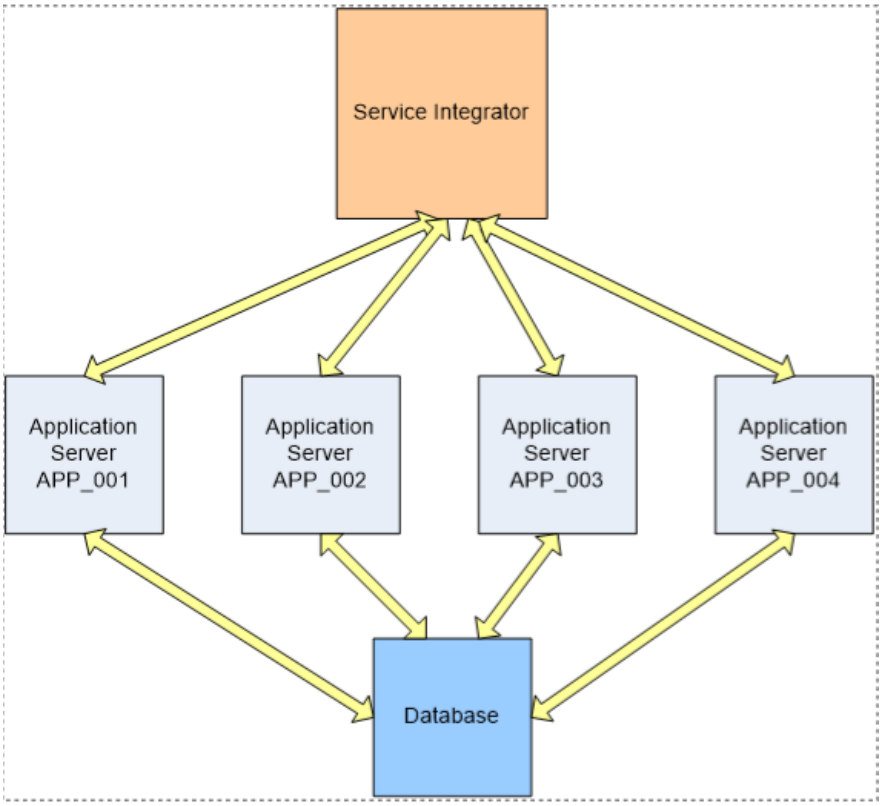


Application concept on OBE

An application is composed of several components. Components are smaller parts of an application, which provide a certain subpart of the catalogue of services that the said application serves. Applications are divided into components, because each component is supposed to be maintained by a different team. In theory, each component is supposed to be more or less independent. 'For the hosting solution part', an OBE app is also composed of several components: A farm and a cluster. The farm is a group of application nodes that are taking care of the service's traffic (Online traffic processing, short tasks). The cluster takes care of Daemon and offline batch processing (long tasks). Let's dive into both of these type of transactions.

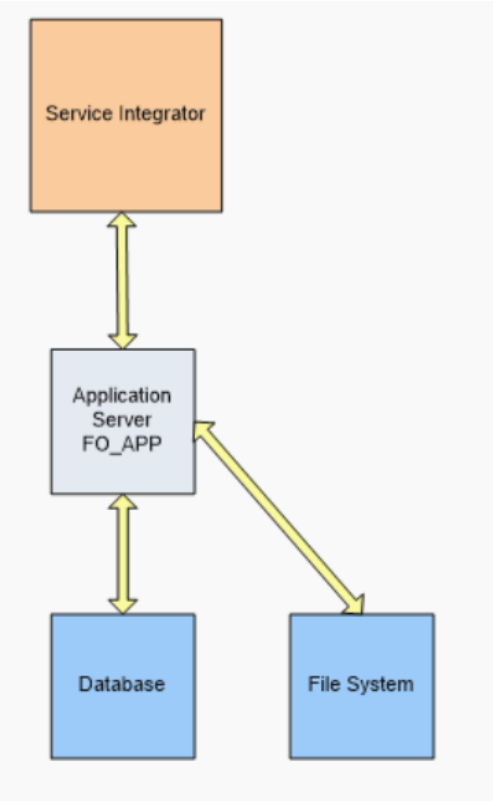
Online Transaction Processing on OBE

The servers that you see here are what we call application servers, or AS, or OTP servers for online transaction processing servers etc... These are what the farm is composed of. They are a copy of each other (The insides of each AS). The SI has knowledge of all four of them, and each one of them is connected to the database. When the SI receives a client request, it performs load balancing on the different servers (on a round robin basis by default, but the distribution strategy can be changed).



Batch Transaction Processing

Batch Transaction Processing tasks are being run in a specific AS. This is where the batch node is, or the fail over node. It still the same software or application that is running in application server, but it is handled differently. It is connected to the database, and to the SI as well. The connection to the SI is mainly there to "talk out": The FO is able to send requests to the SI, on the other hand in does not receive any traffic. The other specificity of the FO_APP is that it has a movable File System, which ensures that the Failover is available at all time (we'll get to this in details later).



Other kinds of cluster nodes

On top of the farm, which is the service part, the failover node which is the batch and the offline activity, we have others as well:

1. GQS Node = Global Queuing Server. Used to queue up items on a Application level (because we can queue up items on every farm node as well, but then you don't have a way to make sure that they are processed in a certain sequence). GQS provides the possibility of sequencing queue items. It accepts downtime.
2. HACS Node = High availability context server. It secures context. We'll go more in details on this later.

> "These cluster nodes are presented to complete the picture of what an application looks like, GQS and HACS are implemented in the same way as a Batch node on a VCS cluster."

What's a VCS cluster?

High availability Concepts

Concepts behind OBE is always about high availability. We always make sure that we don't impact the service when something goes wrong (if we lose an application node, or the DB goes down. In this high availability initiative, we implemented relevant features:

- Farm nodes:
  - Load balancing as we said
  - Traffic can be drained anytime for a certain node: it gives us the possibility to offload and reroute traffic according to our needs
  - Nodes can be stopped (once traffic is drained)
  - This is not really a feature but it serves our high availability needs: we are always over-capacitated, meaning that our hardware capacity is always higher that our peak traffic
- Cluster nodes:
  - Two boxes setting: Active/Passive mode, meaning that if the active node fails, the passive one is ready to take over. (VCS is taking care of this). (Changing the active node is called a fail over)

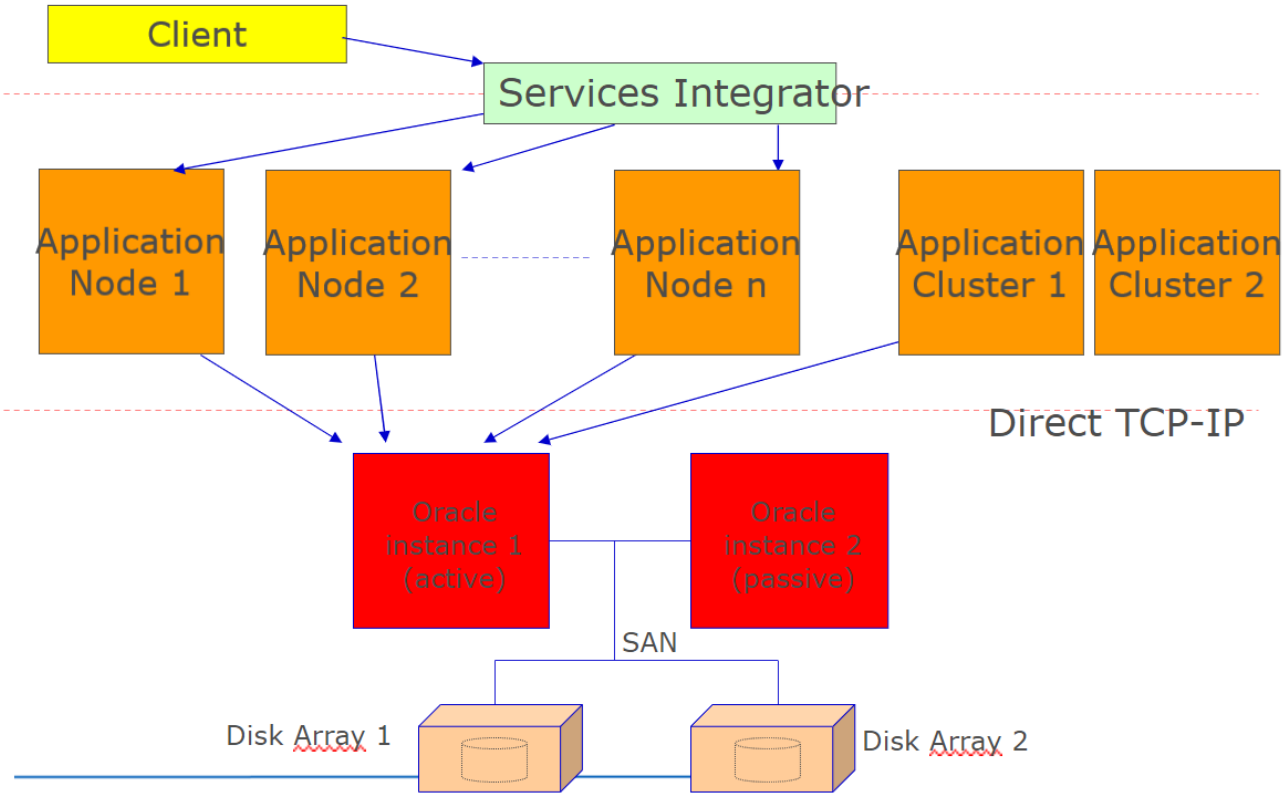
Scalability Concepts

High availability and scalability try to achieve the same goal. Which is that our services are always available, no matter the traffic load or the incident. How does our effort in scalability translate itself in the OBE world?

- OLTP (i.e the Farm part): if we need more capacity we buy more servers
- Database: Peaking we split of databases into multiple dbs when they get too big
- Fail-Over node: Nothing 😒

OBE nodes and database redundancy

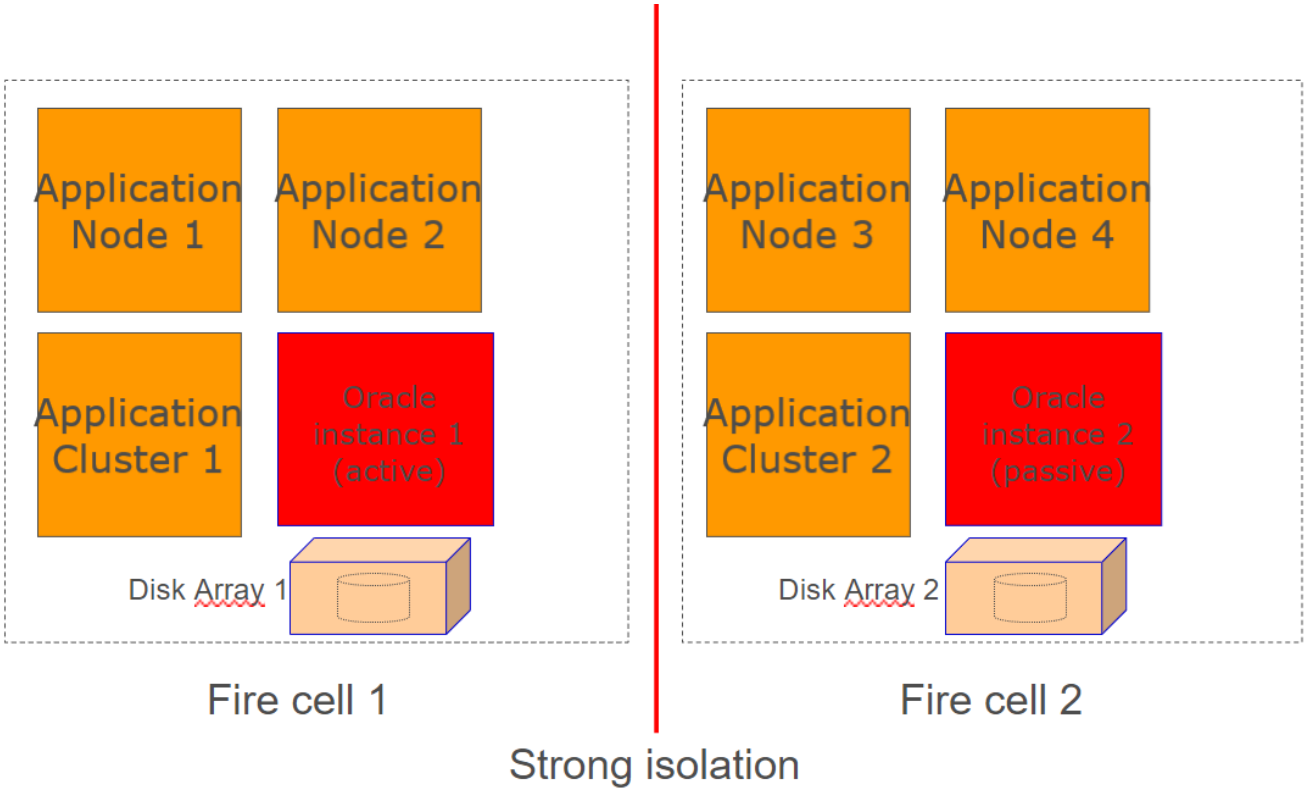
Database redundancy is how high availability translate itself for dbs. Normally, we have one active db instance and two passive instances (below we see just one). However, even though the oracle are running on different hardware, at the end they still use the same storage (SAN storage). SAN is still a bottle-neck :/. For critical applications, we have an additional database infrastructure with dedicated hardware and disk storage, this is what we call EHA = extended high availability. We can switch to this backup db infrastructure in the matter of three or four minutes.



No transition during the presentation, so no transition in my notes, sorry dear reader. Just know that we are moving on to another concept now.

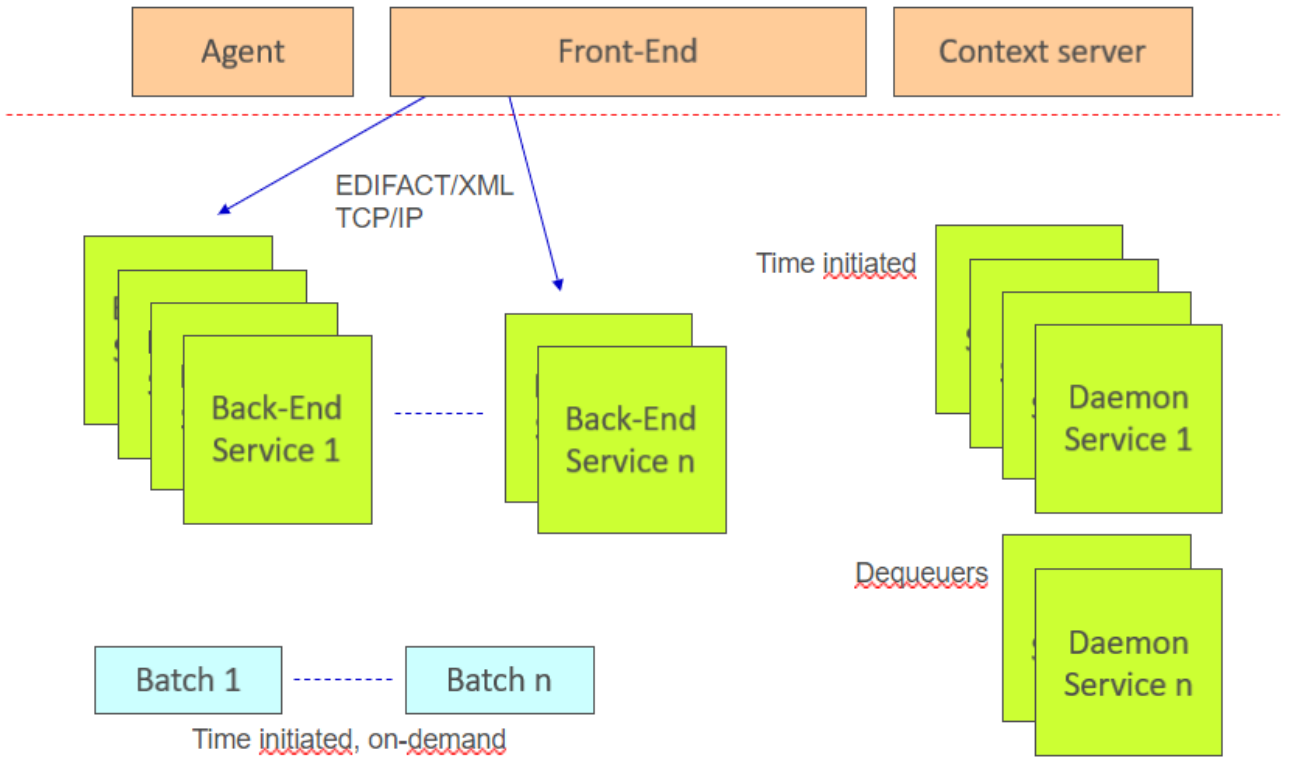
OBE nodes and fire cells

In our datacenter in Erding, we have three fire cells, and each fire cell is divided into two partitions (so in essence you could say that we have 6 fire cells). Normally they are called A1 A2, B1 B2 and Charlie1 Charlie2. You can tell by the name of the Linux host you are working on, in which fire cell they are: obeap1something is in fire cell A1, obeap2something in a2 and so on. The goal behind this division in fire cells is that at anytime we can lose one and they will be no impact to our services. So what we do with application nodes and db servers is that we never run an application completely out of one fire cell (in theory). Meaning that application nodes are always found in different fire cells, same goes for oracle instances, disk arrays etc... The separation is even carried along to the hardware level! Each of these fire cells have their own power grid, air conditioning etc...



One application OBE node/cluster

A more detailed overview of what an application node in the farm looks like:



Every application has these three components: one Agent, one Front-End, and one Context Server.
The Agent will be explained when we'll explain more about the SI. The context server we're gonna come to later as well. The FR basically takes care of the all the communication that happens from the SI to the green levels you see below it. And speaking of which, let's chat a bit about these green boxes:

- Back-ends: They take care of services, these are the ones that reply to the requests coming in through the SI and through the FE.
- Daemons: Can vary according to the app's needs, they correspond to the offline batch processing we talked about earlier.
- Batches:what are thoooooooooooooooooose?

Let's dive in a bit deeper:

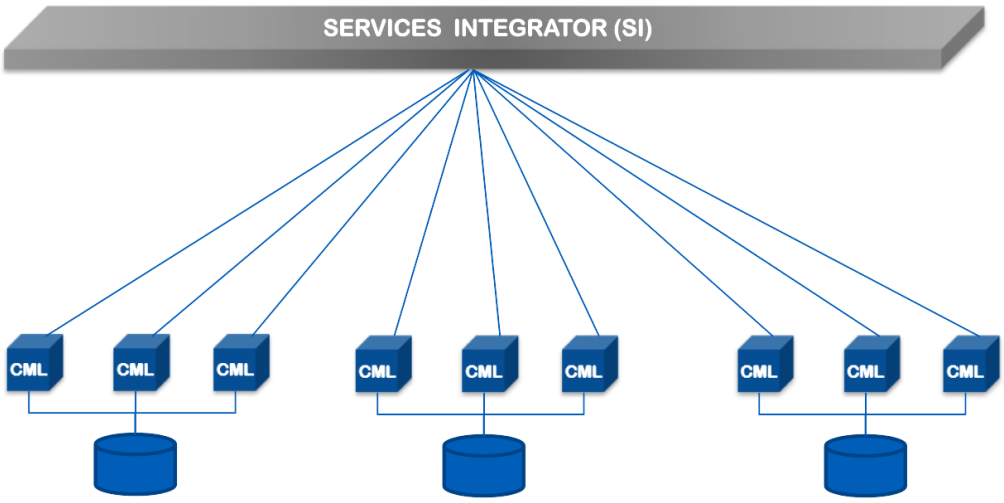
Main layers of an OBE process

Every OBE process is split up in different layers:

- **Middleware layer = Access layer:** Handles how the OBE communicates to the outside world: How the BE talks to the FE, how it understands what an EDIFACT message is, how to build and format a reply... This is the OTF!!
- **Non Graphical Business Model = Business layer:** This is what the application does. This is what the application developer build.
- **Database layer = Persistency Layer:** This contains the code handling the how of database connection and communication.

Application Peak: Application resilience

Application Peaks is a redundancy mechanism where we replicate applications and assign each replicate to a set of customers or geographic region for example. Every peak has its own database.



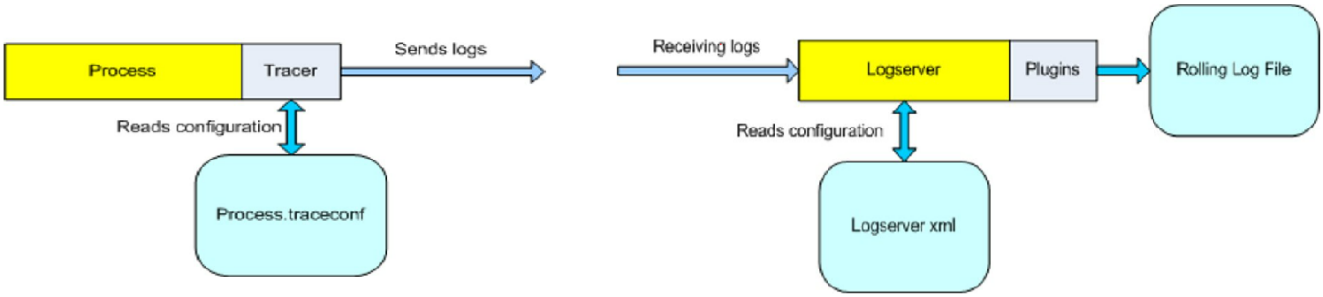
Other objectives for peaking are:

- Increase resilience by minimizing risk:
  - Isolate customers so a failure does not impact all of them at the same time
  - Perform service outages during customer "off peak" time
- Increase scalability at database level:
  - Scalability of the storage layer (this is actually not a new point, this is the database peaking that we talked about before)

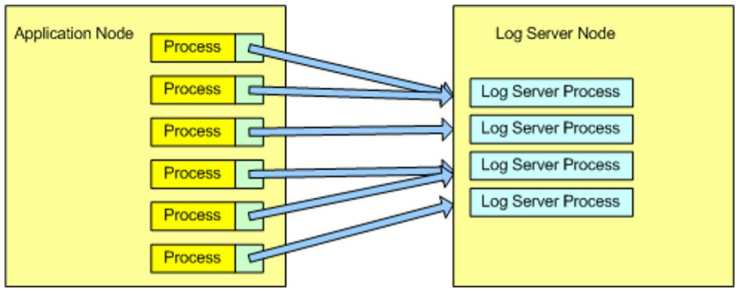
Complete change of topic incoming!

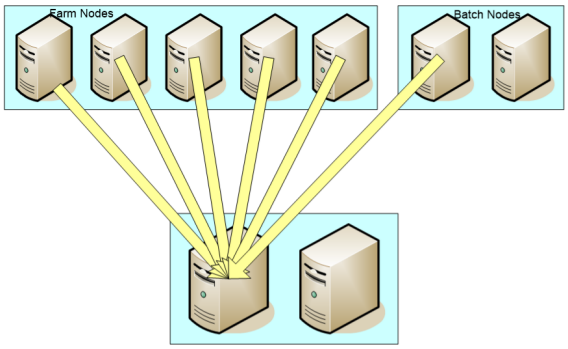
Logging: Client Process view

Every process that is running (every OBE, every FE, every context server...) has a tracer that handles logging. This tracer is given a traceconf that describes what we want to log when a certain event happens and where we want it to be logged.



On the other end, the log server receives the log, buffers them then writes them to disk. I'm gonna stop writing about log handling because the following pictures speak for themselves:





Log Server