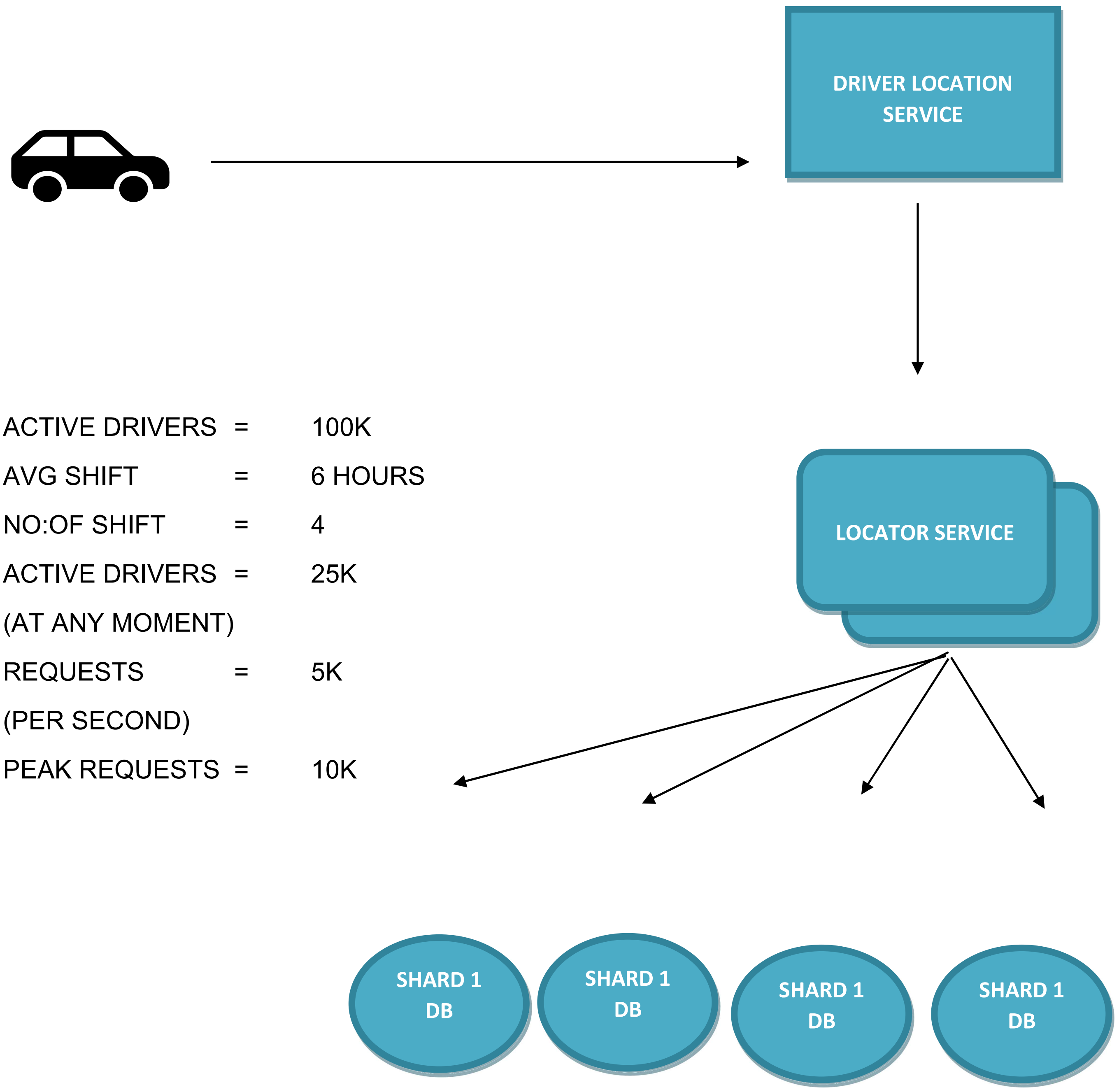
PROJECT

DRIVER LOCATION



ACTIVE DRIVERS	=	100K
AVG SHIFT	=	6 HOURS
NO:OF SHIFT	=	4
ACTIVE DRIVERS	=	25K
(AT ANY MOMENT)		
REQUESTS	=	5K
(PER SECOND)		
PEAK REQUESTS	=	10K

Question 1

- ❖ Define the following cache strategies:
- Cache Aside
- Read Through
- Write Through
- Write behind

a) Cache Aside
The **cache** sits on the side and the application directly talks to both the **cache** and the database. ... The application first checks the **cache**. If the data is found in **cache**, we've **cache** hit. The data is read and returned to the client. If the data is not found in **cache**, we've **cache** miss.
b) Read Through
When an application asks the cache for an entry, for example the `key X`, and `X` is not already in the cache, Coherence will automatically delegate to the CacheStore and ask it to load `X` from the underlying data source. If `X` exists in the data source, the `CacheStore` will load it, return it to Coherence, then Coherence will place it in the cache for future use and finally will return `X` to the application code that requested it. This is called **Read-Through** caching. Refresh-Ahead Cache functionality may further improve read performance (by reducing perceived latency).
c) Write Through
Coherence can handle updates to the data source in two distinct ways, the first being **Write-Through**. In this case, when the application updates a piece of data in the cache (that is, calls `put`(...) to change a cache entry,) the operation will not complete (that is, the `put` will not return) until Coherence has gone through the `CacheStore` and successfully stored the data to the underlying data source. This does not improve write performance at all, since you are still dealing with the latency of the write to the data source. Improving the write performance is the purpose for the *Write-Behind Cache* functionality.

d) Write behind

In the **Write-Behind** scenario, modified cache entries are asynchronously written to the data source after a configured delay, whether after 10 seconds, 20 minutes, a day, a week or even longer. Note that this only applies to cache inserts and updates - cache entries are removed synchronously from the data source. For **Write-Behind** caching, Coherence maintains a write-behind queue of the data that must be updated in the data source. When the application updates `X` in the cache, `X` is added to the write-behind queue (if it isn't there already; otherwise, it is replaced), and after the specified write-behind delay Coherence will call the CacheStore to update the underlying data source with the latest state of `X`. Note that the write-behind delay is relative to the first of a series of modifications—in other words, the data in the data source will never lag behind the cache by more than the write-behind delay.

<u>Question 2</u>

- Explain Message Queues and PUB- SUB Queues. Mention where they are used.

- A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures. Messages are stored on the queue until they are processed and deleted. Each message is processed only once, by a single consumer. Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads.

- Publish/subscribe messaging, or pub/sub messaging, is a form of asynchronous service-to-service communication used in serverless and microservices architectures. In a pub/sub model, any message published to a topic is immediately received by all of the subscribers to the topic. Pub/sub messaging can be used to enable event-driven architectures, or to decouple applications in order to increase performance, reliability and scalability.