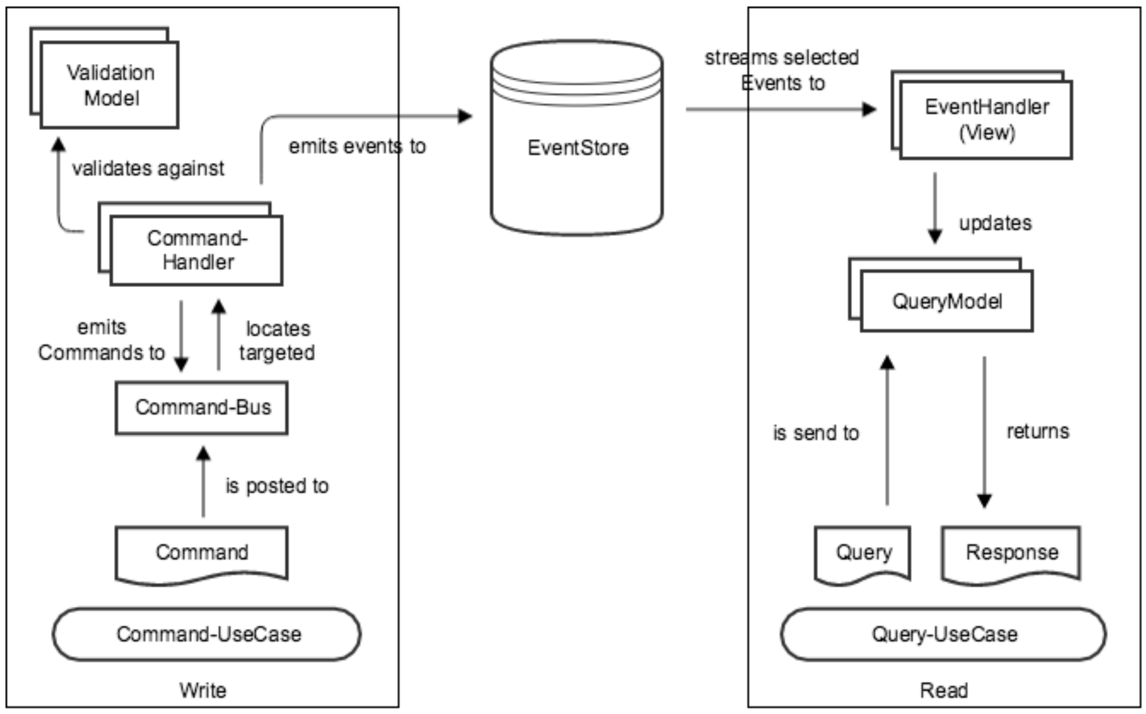

Term	Meaning
Message	Abstract thing to be either consumed or rejected
Event / Fact	Message in the past that already has happend, Events are not rejectable. Events are not targeted to specific Systems as they are observable by anyone. Events are written with the vocabulary of the Sender
Command	Message that is imperative and tells some defined System what to do. Commands are directed, so they use an agreed vocabulary. Commands can be rejected
Effect	Commands have Effects (Any number of either new Commands, or Events) when accepted
SideEffect	An Effect that does not affect the System at hand directly, but is necessary to executed only once (sending a welcome mail). Mostly modeled with a Command emitted as an Effect
CommandHandler	Recieves a Command, validates it (potentially against a WriteModel) and emits Effects (Write side of CQRS)
EventHandler	Recieves an Event and adjusts local model accordingly. Every ReadModel has EventHandlers for every Event they are interested in
EventStream	Ordered Stream of Events that every Model (Read or write) requests from the EventStore. EventStreams might end at some point or infinitely send new Events as they occur (tail-Mode)
EventLog	The ordered sequence of Events persisted
EventStore	A deployable component that has encapsulates access to the EventLog an coordinates Writes / Streaming of Events
Event Transformation	Events are forever. In order not to carry deprecated code that handles a particular Event Type (long deprecated) and avoid code duplication in the handling code, Events may be Transformed before handling (for instance from Version one to Version 2). This work is done by an Event Transformer
Event Upcasting	Building a chain of Transformers from any version of an Event Type provides the handling code with only the last version
Persistent Event Upcasting	In frequently replaying systems, you may want to avoid unnecessary upcasting, by storing the upcasted version(s) of the Event next to the original.
Projection	Any Transformation that takes Events and produces some kind of transient state. (see ReadModel, WriteModel)
ReadModel (QueryModel)	A Projection from the events that clients can read from. A System can have any number of ReadModels aggregating data in a Query-optimized way (for their UseCases). ReadModels can – but do not have to be – eventual consistent
WriteModel (ValidationModel)	WriteModel, because changes needed to be 100% covered by the Effects a CommandHandler emits. A better Strategy is to just write via events and update the WriteModel (Just like any ReadModel) from there. While there can be any number of WriteModels in a System, often there is only one resembling the 'canonical model'. WriteModel often have to be strictly consistent if they are used for validation
View	Other word for Projection. Views often can be externalized to autonomous Services and scaled horizontally
Pull-View	a View that updates itself (by looking at the EventLog and applying fresh Events) before answering any question. Pull-Views are strictly consistent
Push-View	a View that is asynchronously updated with fresh Events as they occur. Push-Views therefore are eventually consistent and interesting for read-heavy UseCases
Snapshot Aggregation	Using a pull-view, a snapshot is persisted with the information of what event was consumed last. This way, when pulling next time, the snapshot can be read and pending events can be aggregated on top, saving the aggregation frmo the beginning of the log. Note that Snapshots have to be removed or migrated, if the View changes in an incompatible way