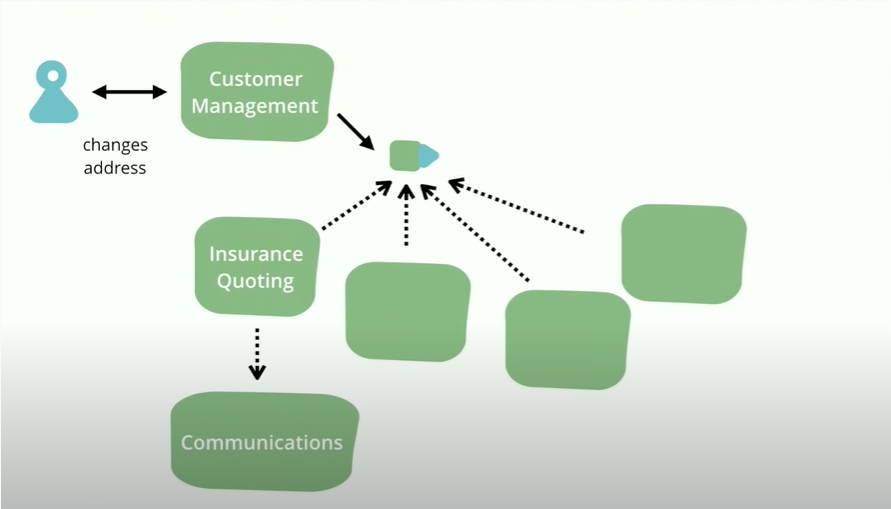
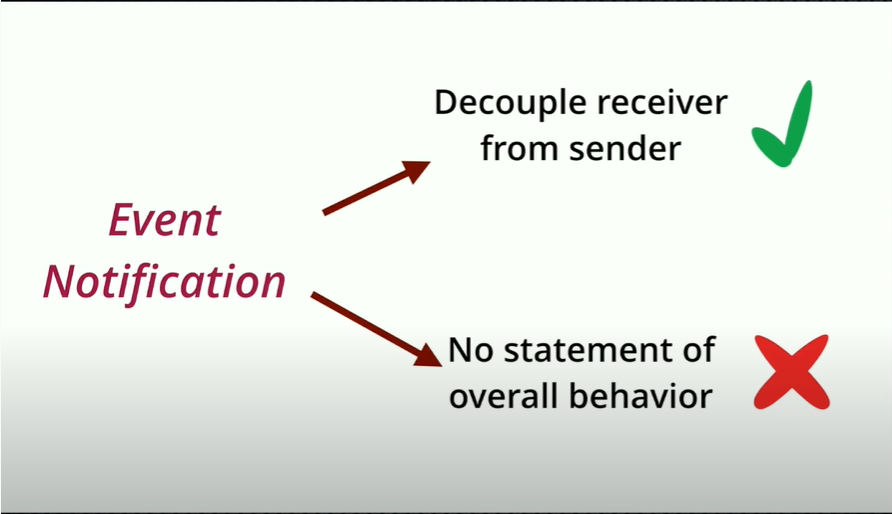
Martin fowler article: <https://martinfowler.com/articles/201701-event-driven.html>

* **When people talk about “events”, they actually mean some quite different things:**

1. **Event Notification:**

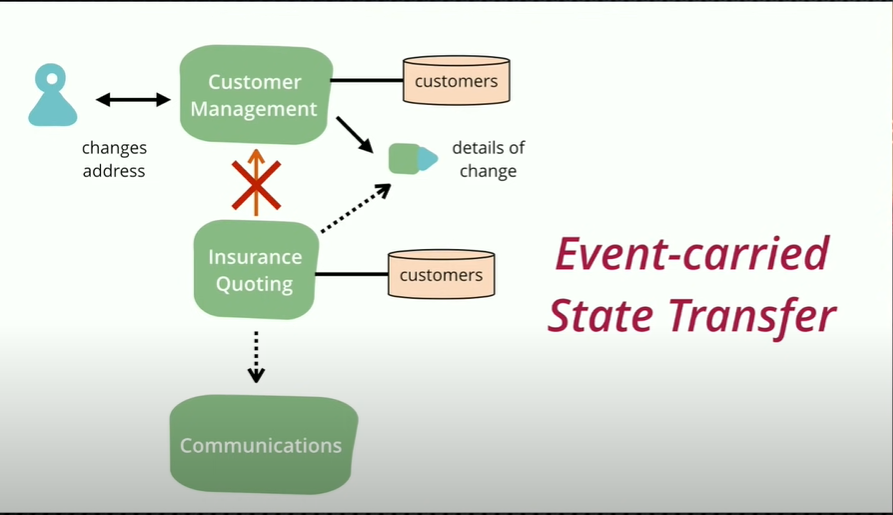
* This happens when a system sends event messages to notify other systems of a change in its domain.
* The source system doesn't really care much about the response.
* **Pros:** Decouple receiver from sender
* **Cons:** No statement of overall behavior

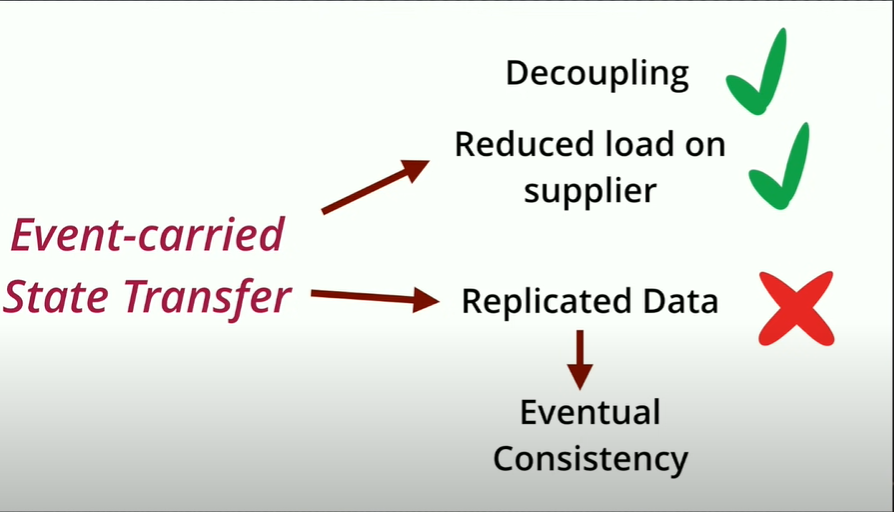




1. **Event-Carried State Transfer:**

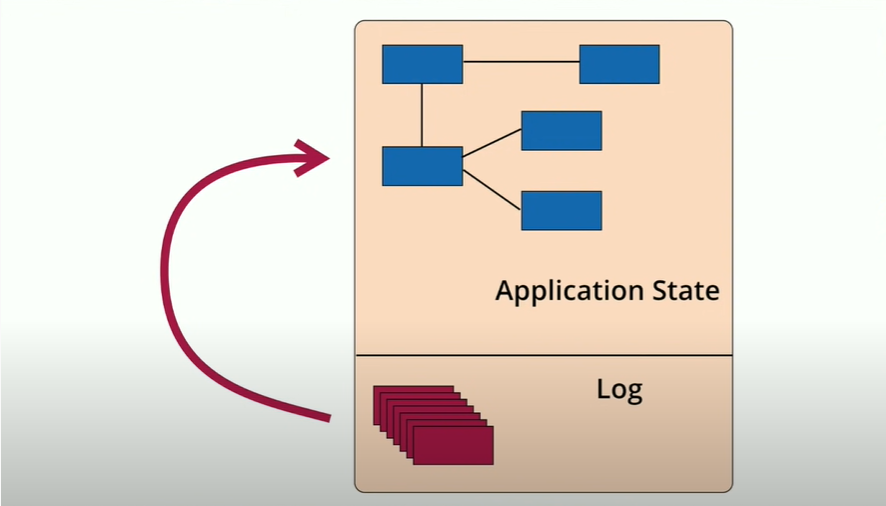
* This pattern shows up when you want to update clients of a system in such a way that they don't need to contact the source system in order to do further work.
* A customer management system might fire off events whenever a customer changes their details (such as an address) with events that contain details of the data that changed. A recipient can then update its own copy of customer data with the changes, so that it never needs to talk to the main customer system in order to do its work in the future.

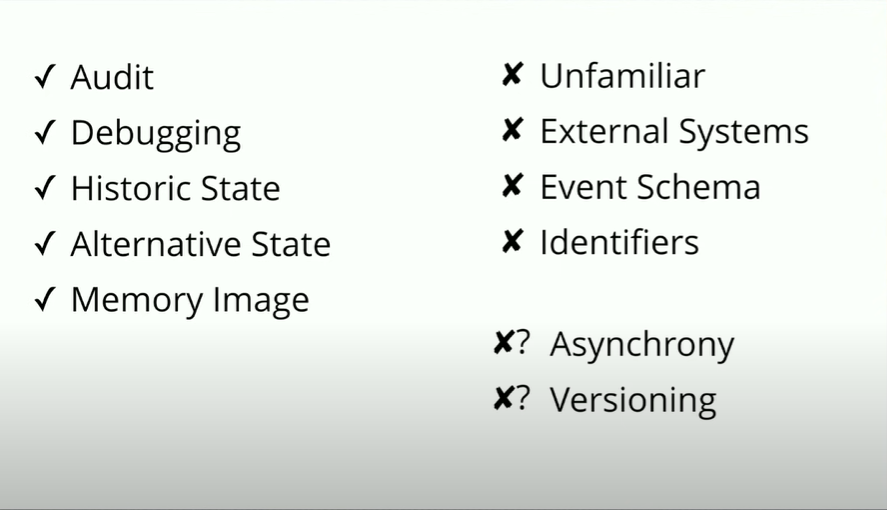




1. **Event Sourcing:**

* Whenever we make a change to the state of a system, we record that state change as an event.
* We can confidently rebuild the system state by reprocessing the events at any time in the future.
* The best example of this is a version-control system. The log of all the commits is the event store and the working copy of the source tree is the system state.
* It is often useful to build snapshots of the working copy so that you don't have to process all the events from scratch every time you need a working copy.





1. **CQRS**

* Is the notion of having separate data structures for reading and writing information.
* Strictly CQRS isn't really about events, since you can use CQRS without any events present in your design. But commonly people do combine CQRS with the earlier patterns here, hence their presence at the summit.

