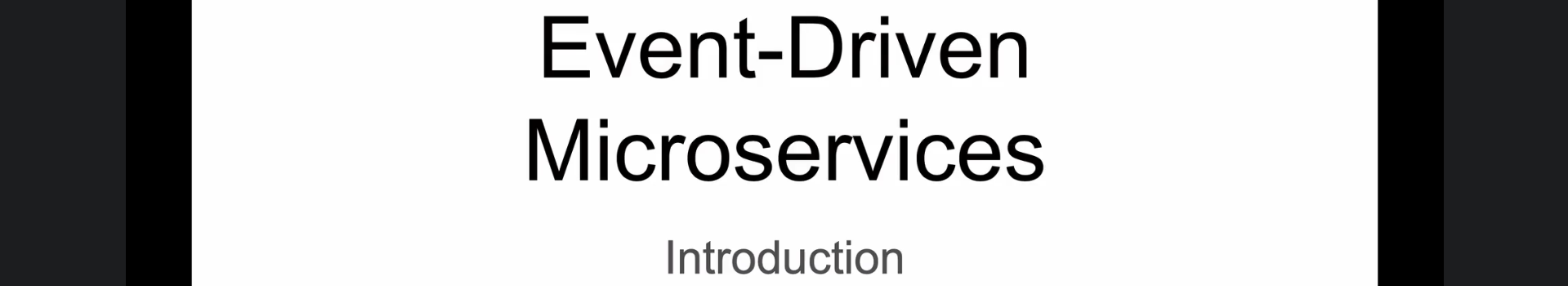
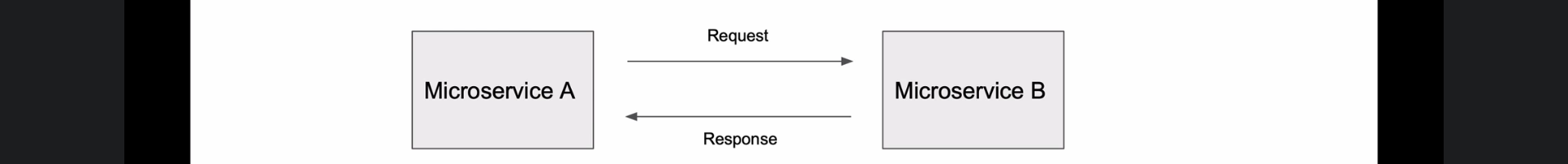
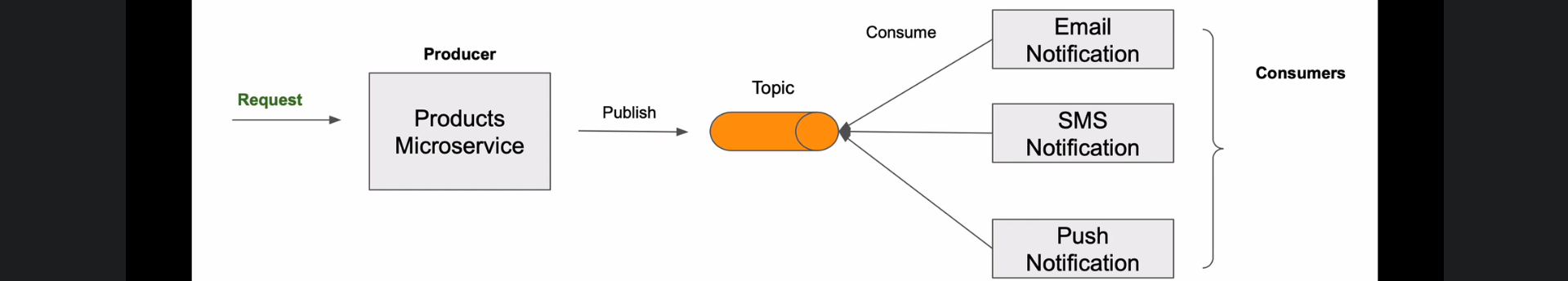
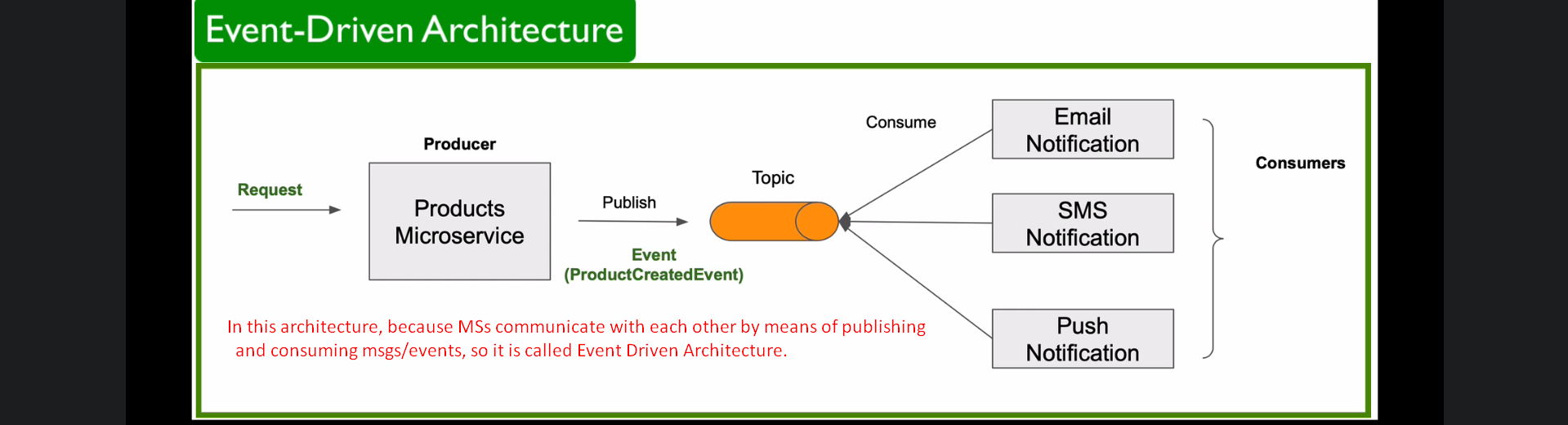
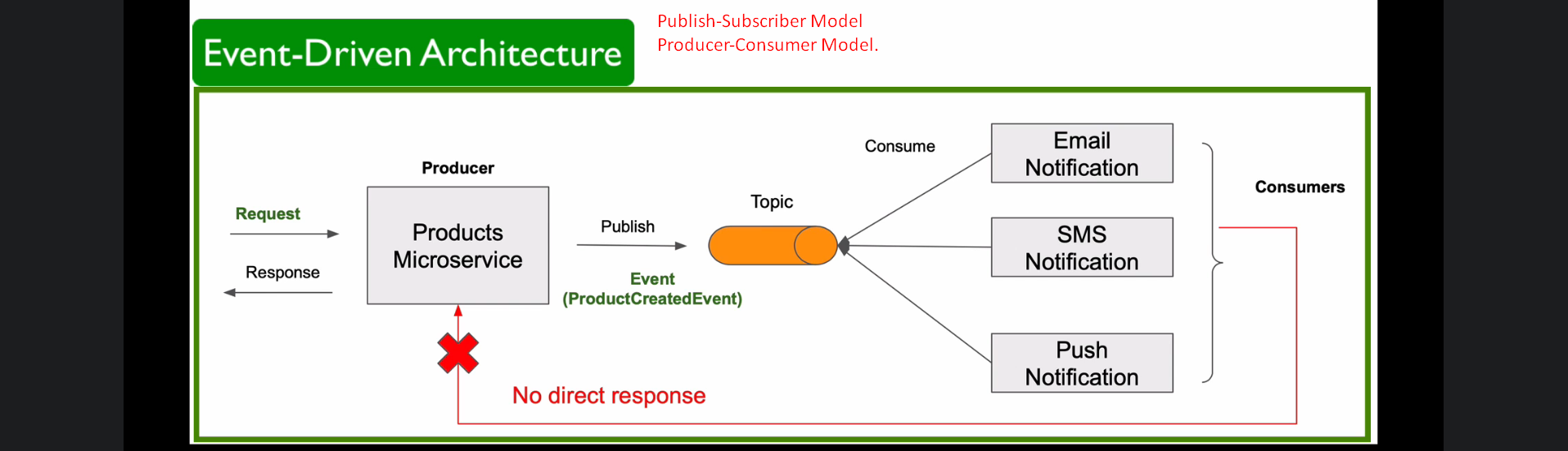
1. 
2.   
   When two applications communicate with each other, usually this communication is in the form of request and response.   
   One app will send HTTP request to other and the other will send the HTTP response.  
   This type of application is useful in many use-cases but not all.
3. What if an app needs to send a msg to multiple MSs at the same time.  
   The direct request and response communication is not convenient in this case. If we have two MSs, maybe then we can deal with.   
   but what if there are dozens of MSs that need to receive the msgs. What if an MS is added later on. How many will be added later on, we don’t know.  
   Here direction “request and response” communication is no longer helpful.  
   This is when Apache Kafka and Event Driven Architecture will be very helpful.
4.   
   MS which needs to send msg to multiple MSs will publish the msg to a Kafka Topic and other MSs will consume the msg from the topic as soon as the msg is available.  
   This **Model** is called **Producer & Consumer** where the MS which sends msg is called **Producer** and the MSs receiving the msg are called **Consumers.**This **Model** is also called **Publish** **& Subscriber**.  
   This is very scalable and extensible architecture.  
   MSs are loosely coupled and completely location transparent to each other.  
   We can add as many receiving MSs as we need. We just need to know what kind of msgs are needed.
5. Now let us assume that Product MS gets a request like CREATE a new PRODUCT.   
   When the PRODUCT MS is done with processing, it will publish an even let’s say PRODUCT-CREATE-EVENT to a Topic and other MSs will receive the event and can act on it.   
   
6. Where simple request and response communication is needed, you don’t need to make system overcomplicated with Event Driven Architecture.
7. Another important point is that in most cases, it is asynchronous and that it is loosely coupled.  
   For example: The publishing MS publishes a msg to a Kafka Topic, the Kafka responds back the calling app right away.   
   The Publishing MS doesn’t wait until every single subscribing MSs received and processed the msg.   
   The Publishing MS is not aware about how many subscribing MSs are there and how many of them have successfully handled the event.  
   Subscribing MSs don’t send a direct response back to the publishing MS.  
   If any subscribing MS is down, the Publishing MS will not know about it.   
   As soon the MS comes back, it can consume the published msgs. But in direct communication (HTTP), if the MS is down, then most likely it will miss the msg.  
   But in Event-Driven Architecture, if down MS comes back later on, still it can receive those missed msgs and can resume its work where it left.
8. 
9. In upcoming lectures, we will learn more about Apache Kafka and how it can be used to build Spring Boot Event-Driven MSs.