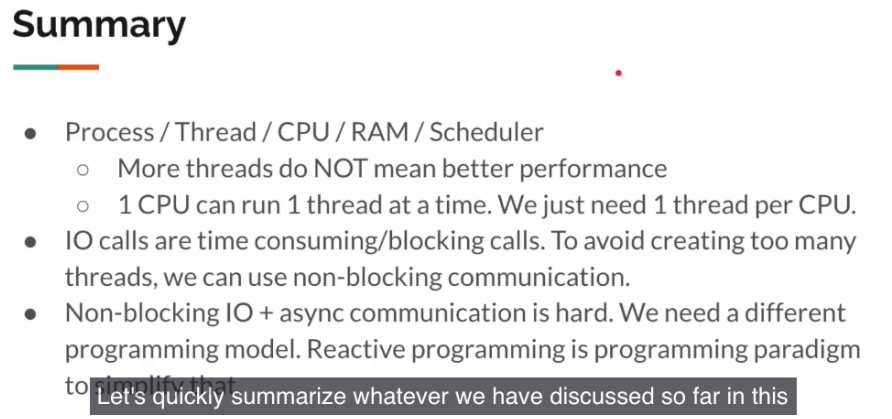
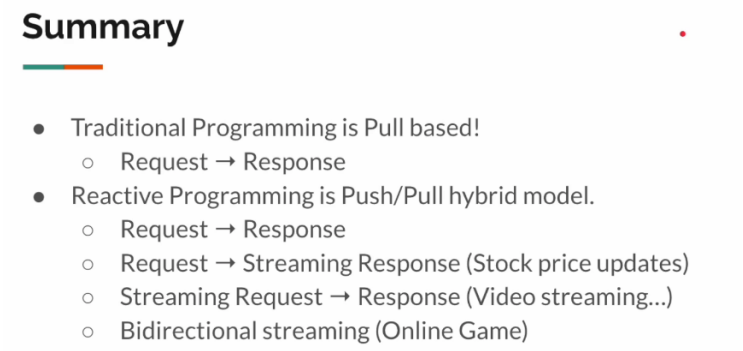
Summary

Wednesday 19 February 2025 17:31

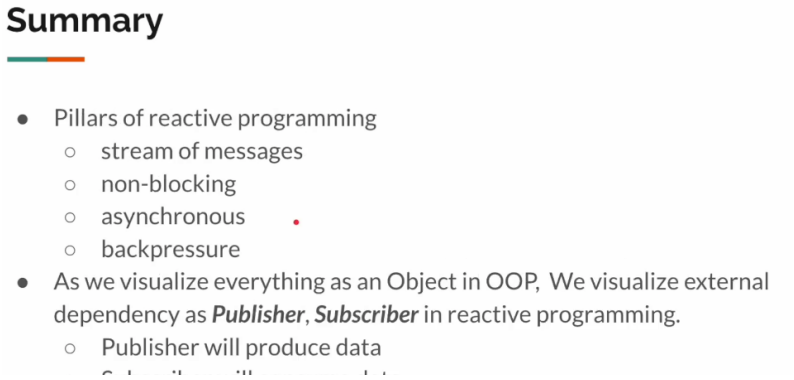
Let's quickly summarize what we have discussed so far.



- More threads do not mean better performance
- 1 CPU can run only 1 thread at a time. So if we have a 10 CPU machine, we just need 10 threads i.e. we just need 1 thread per CPU.
- Often times, people create too many threads. This is mainly because the network calls are very slow or time consuming blocking calls. So to make use of the CPU, people create too many threads.
- But if we use non-blocking communication, we just need one thread per CPU. And this is the most efficient way to handle network calls.
- Non-blocking IO + async communication is difficult. So we need a better programming model. So that why we have Reactive programming to simplify that.

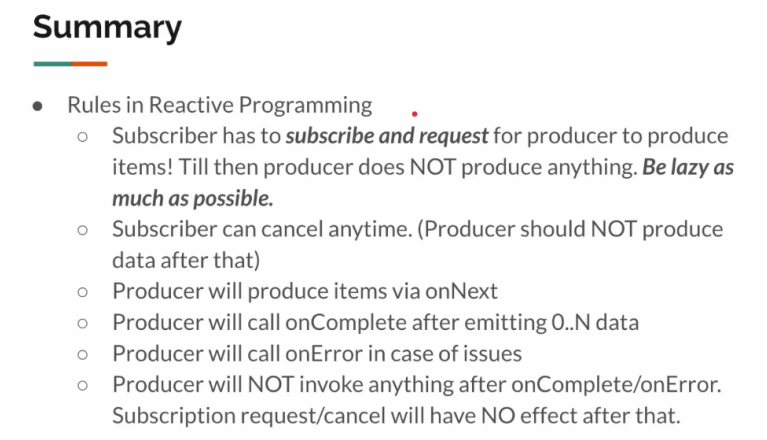


- The traditional programming is Pull based. It supports only the request and response type communication. You will have to send the request to the server to get the response from the server.
- On the other hand, Reactive programming is more of Push/Pull Hybrid model. It does support the request and response, but it also supports three other communication patterns like
  - Request to Streaming Response.
    - For e.g. like Stock Price Updates, the price will change every second. So if you use the request and response model, you will send the request to the server to get the stock price. Again, after 1 sec, you will have to send request to fetch the price and keep on doing the same to get the updates.
    - But, on the other hand, with the Reactive programming, we can send one request to get the streaming response from the remote server and handling the streaming response is a lot easier with Reactive programming.
  - `Similarly, we can also send Streaming Request if you have usecases like Video Streaming, or say, whatever one speaks, the microphone would want to save everything and send it to a remote server. So in this case, I would use a Streaming Request
  - Then, Bidirectional Streaming for online games, etc.



- Subscriber will consume data

- The core pillars or principles of reactive programming.
  - Stream of messages
  - Non-blocking
  - Asynchronous
  - Backpressure
- Reactive programming is all about processing a stream of messages in a non-blocking, asynchronous manner while handling backpressure.
- As we visualize everything as an Object in OOP, we will be visualizing all the external dependencies as a Publisher, Subscriber in reactive programming. Because reactive programming is to make I/O calls more efficient. So its all about external network calls. So this why we visualize external dependencies as Publisher, Subscriber in reactive programming.
  - Anything that produces data will act like a Publisher
  - Anything that asks or consumes data will act like a Subscriber.



These are the rules in Reactive programming

- Subscriber has to subscribe and request for producer to produce items. Till then the producer does not produce anything. So we have to be lazy as much as possible.
- The producer does not have to do any work until it sees the request from the Subscriber.
- The Subscriber can cancel anytime. So we have to be aware of it and the producer should stop producing at that moment.
- The Producer will produce items or share items with the Subscriber using the onNext method.
- The Producer when producing all the items or when the Producer does not have anything to give back to the Subscriber, at that time, it will say onComplete.
- Similarly, when the Producer faces some issues, it will invoke the onError method to notify the error.
- The Producer will not do any work I.e. will not invoke anything after invoking the onComplete or onError method.
- After that the Subscription request or cancel method will not have any effect.