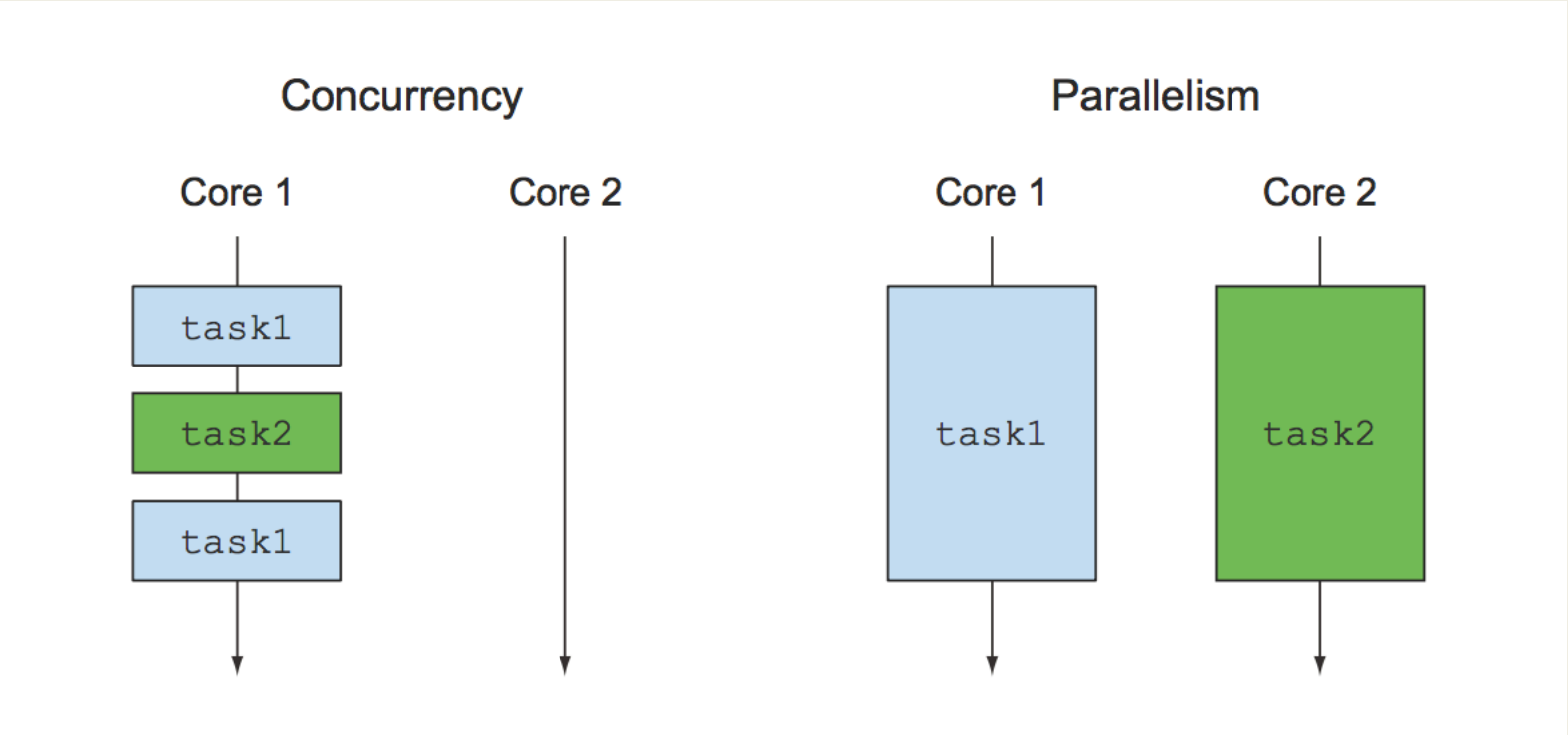
COMPLETABLE FUTURES

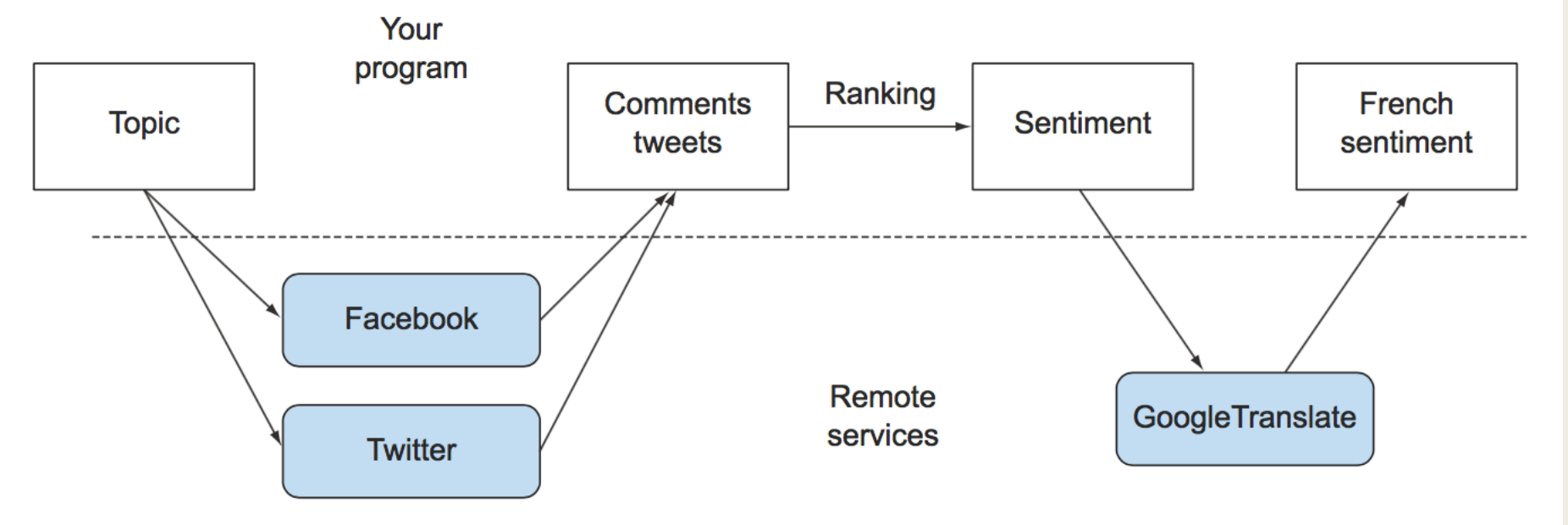
Composable Asynchronous Programming w/ java 8

Burak Yildirim

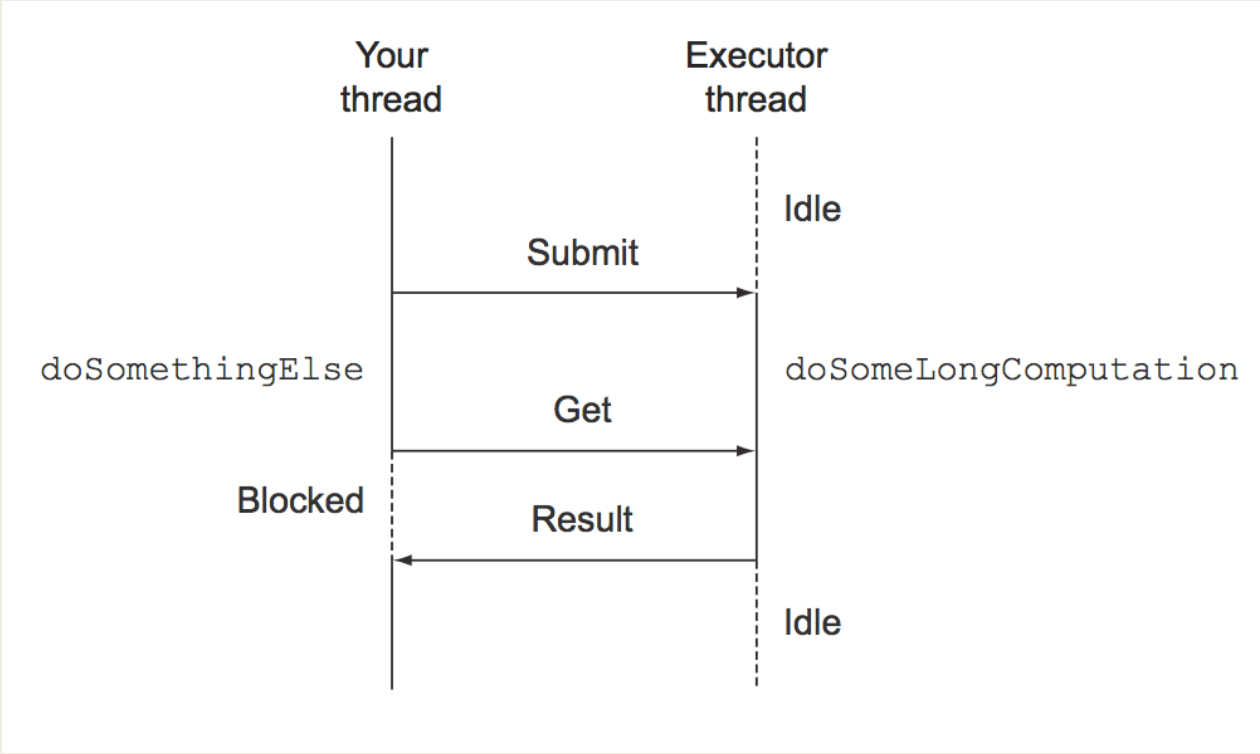
Concurrency vs Parallelism



Asynchronous APIs and Mash-up applications



The good old Java Future (from java 5) until java 8's CompletableFuture



Future Limitations

- When the result of the long computation is available, please send its result to another long computation, and when that's done, combine its result with the result from another query. – wtf ?
- There's no fancy way of doing this in the old Futures !

CompletableFuture implements Future

- Combining two asynchronous computations in one
- Waiting for the completion of all tasks performed by a set of Futures
- Waiting for the completion of only the quickest task in a set of Futures (possibly because they’re trying to calculate the same value in different ways) and retrieving its result
- Programmatically completing a Future (that is, by manually providing the result of the asynchronous operation)
- Reacting to a Future completion (that is, being notified when the completion happens and then having the ability to perform a further action using the result of the Future, instead of being blocked waiting for its result)

CompletableFuture Features

- implementing an async. Api
- non blocking api consumption
- Pipelining asynchronous tasks
- Reacting to a CompletableFuture completion

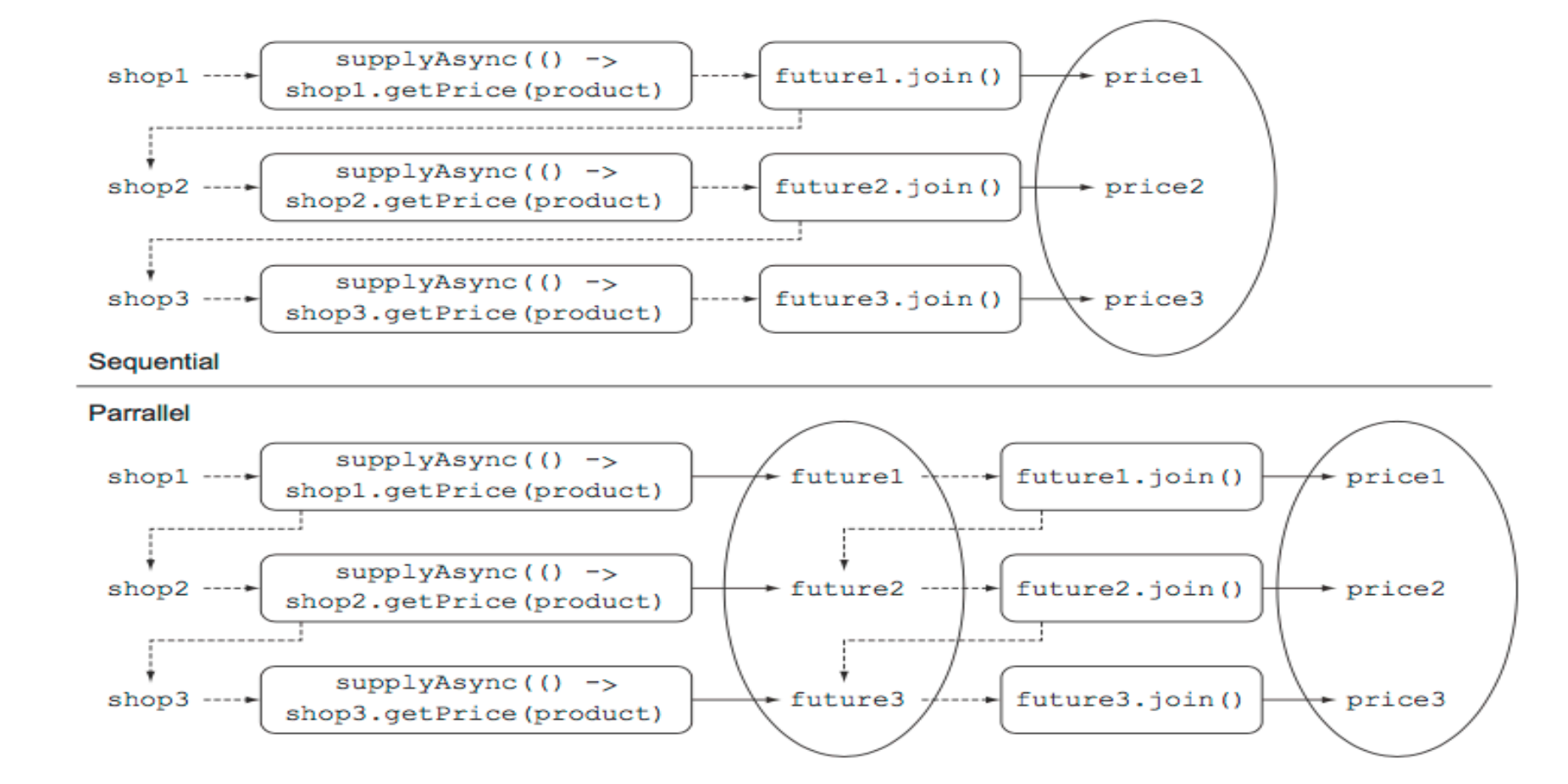
Basics of Completable Future

- CompletableFuture<T> future = …
- future.complete(val)
- future.completeExceptionally(ex)
- CompletableFuture.supplyAsync(() -> foo(bar))

Demo

- Shop, SyncApiExample, AsyncApiExample, NonBlockingApiCalls - demo

Lazy Streams



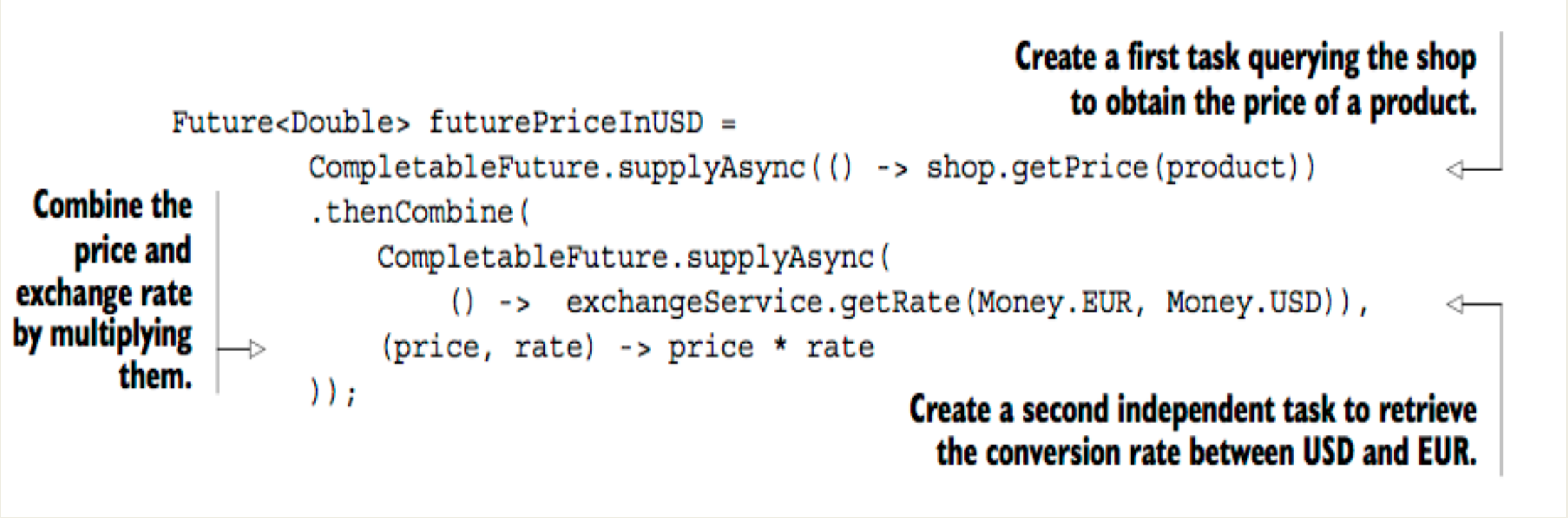
Pipelining Async. Tasks

- thenCompose
- thenComposeAsync
- thanApply (for sync.)

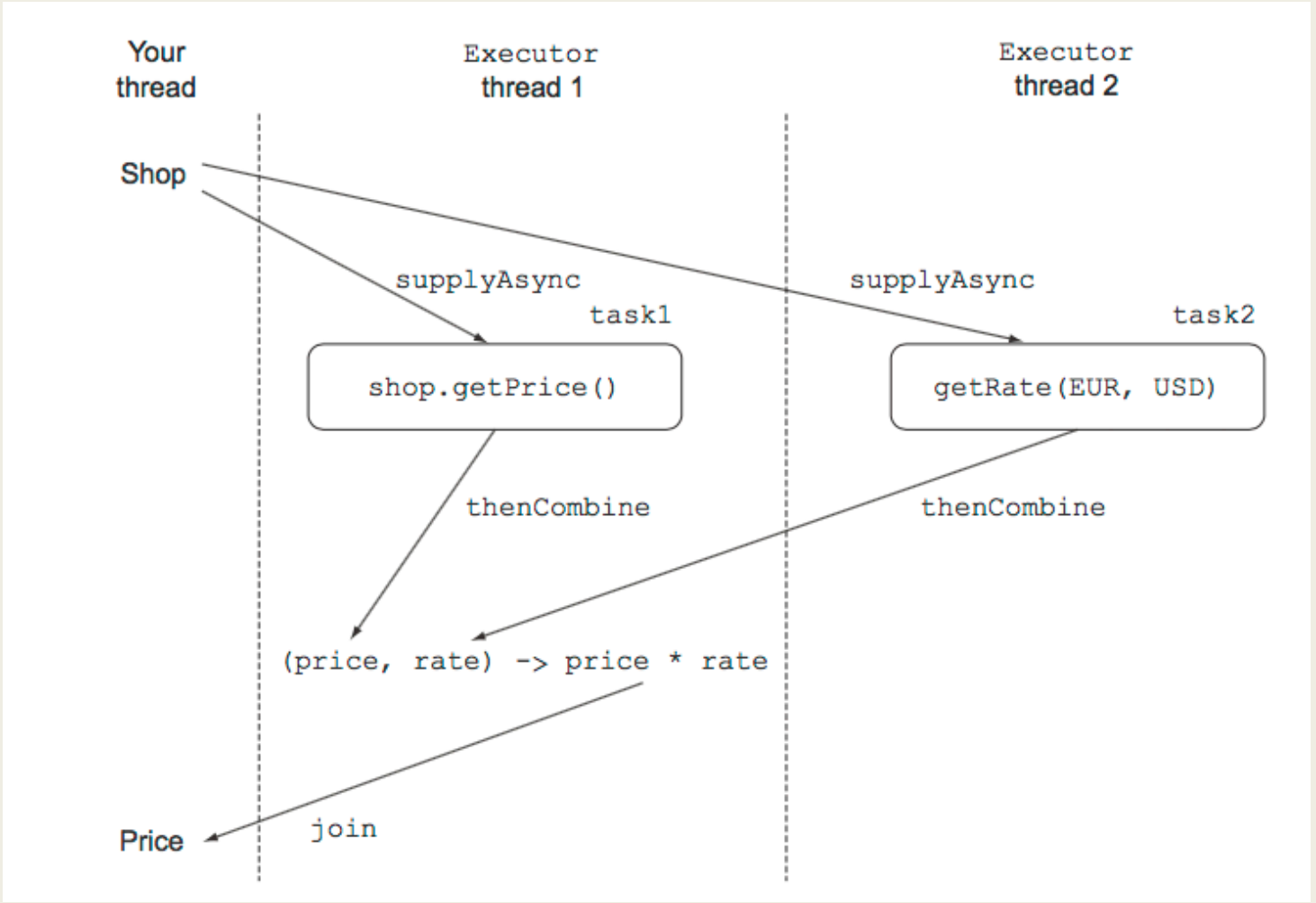
Combining two CompletableFutures—dependent and independent

- thenCombine
- thenCombineAsync

Combining two independent operations



Combining two independent operations – how it works



Reacting to a CompletableFuture completion

- We want to have the application display the price for a given shop as soon as it becomes available, without waiting for the slowest one! Sounds cool !
- thenAccept: *registers* an action on each CompletableFuture; this action consumes the value of the CompletableFuture as soon as it completes.
- findPricesStream("myPhone").map(f -> f.thenAccept(System.out::println));
- CompletableFuture.allOf(futures).join();

Demo

- Reacting to a CompletableFuture completion