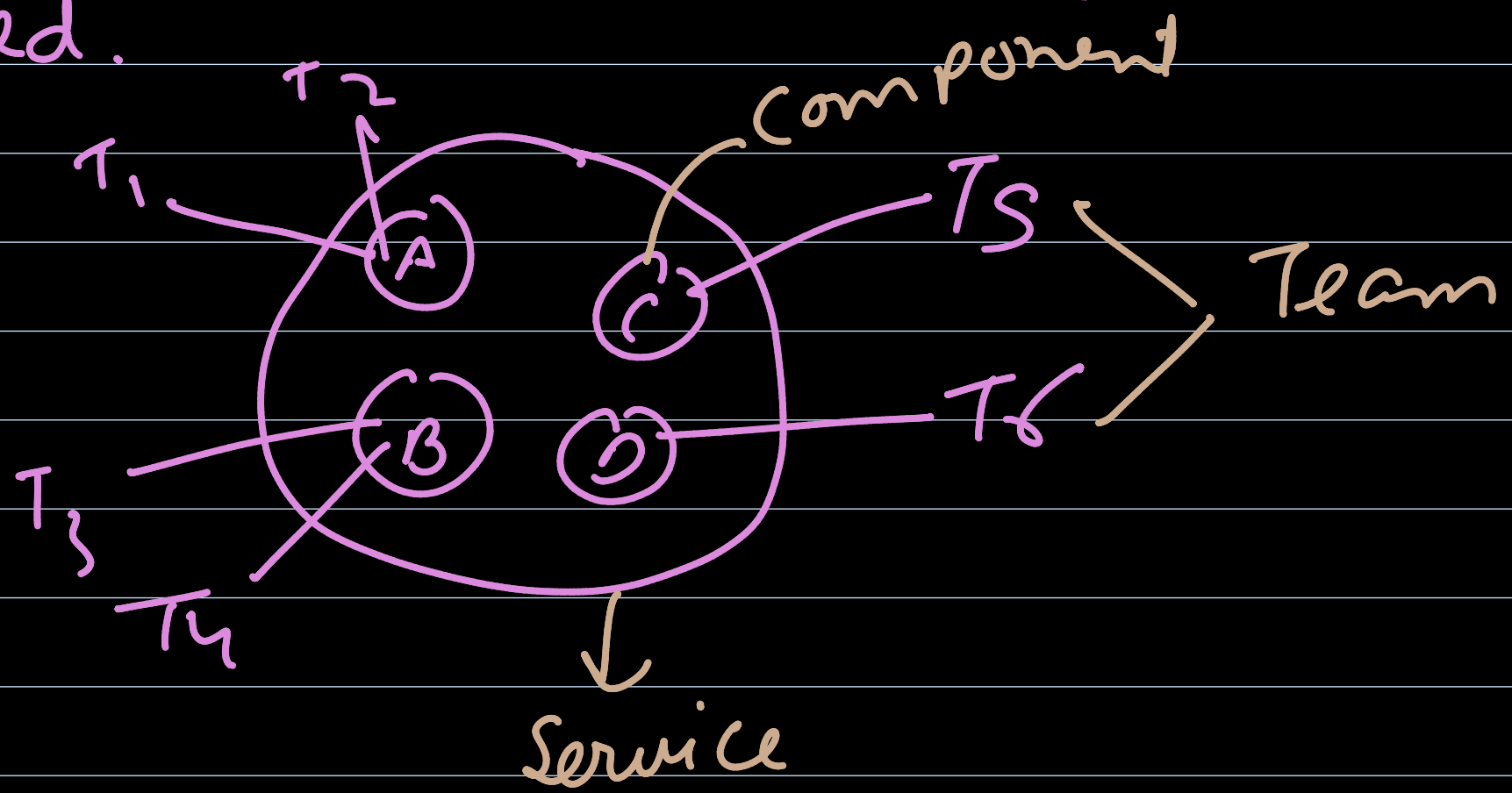
Scoping Microservices

Microservices are awesome at handling these huge projects and shipping new features fast.

But there is a very fundamental concept we need to keep in mind while designing this architecture. We will understand them with following two scenarios.

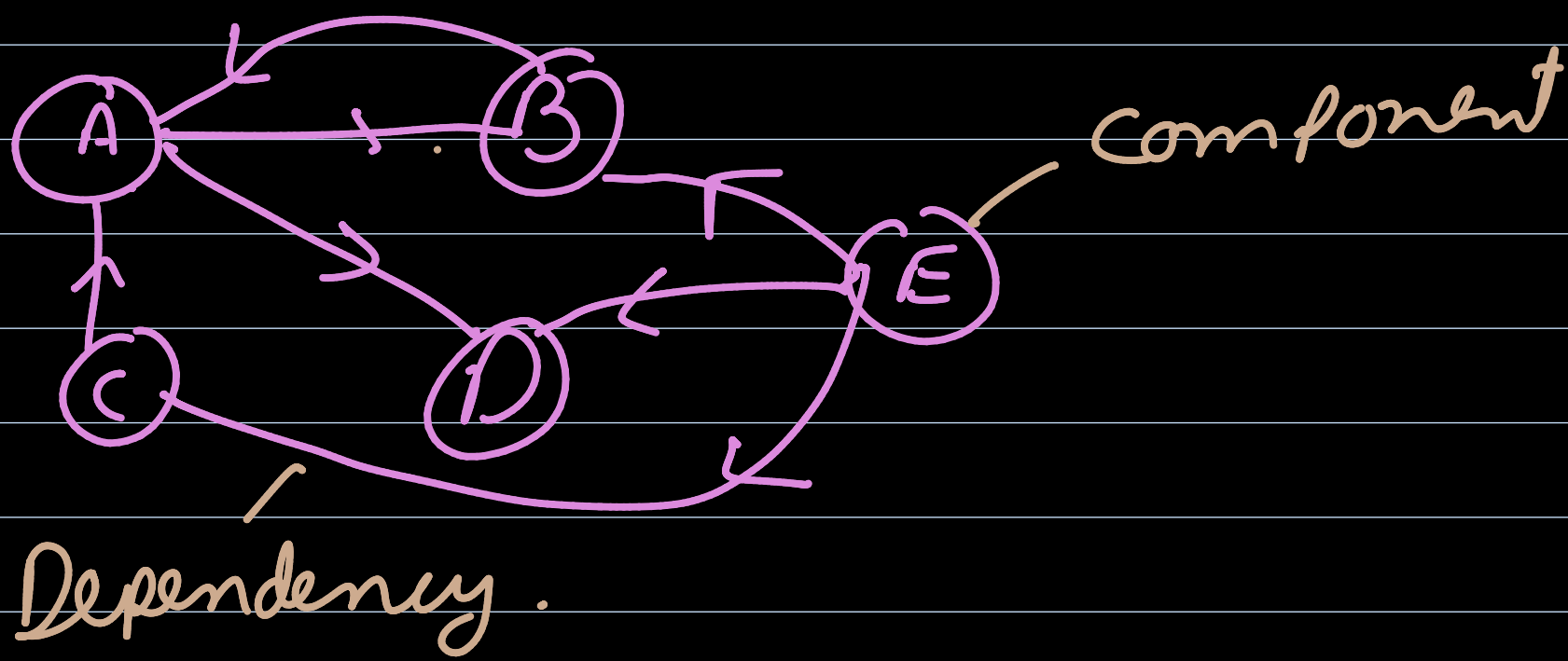
1 Having large no of components

If we have large no of components in a service, then many teams will be working on it. Thus speed and efficacy will be affected.



2. Having fine grained services

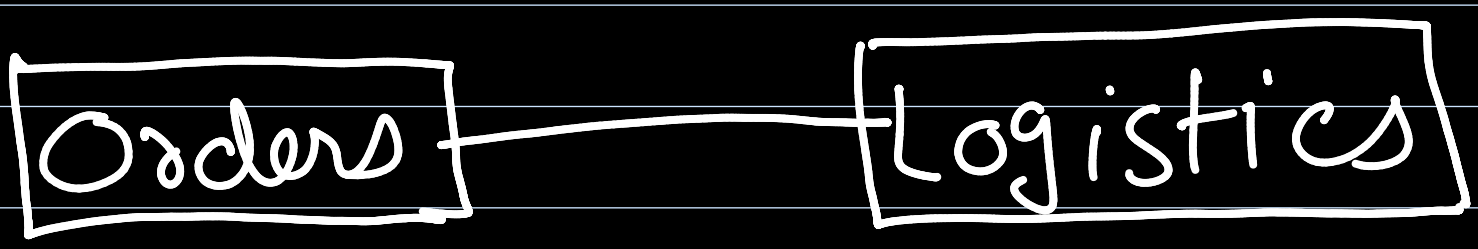
If we have too much of fine grained service then it leads to a complex system with services having a lot of inter-dependency among them. PS:- Not a good thing to maintain either.



These two scenarios above can be managed by the following concepts.

1 Loose Coupling

It implies that change in one service should not affect the other service. This helps in tackling the 1st scenario as we can separate the services.



Lets say we have these two services with us. As long as the communication b/w the two is not changed, till then the changes to one of them is irrelevant.

For eg:-

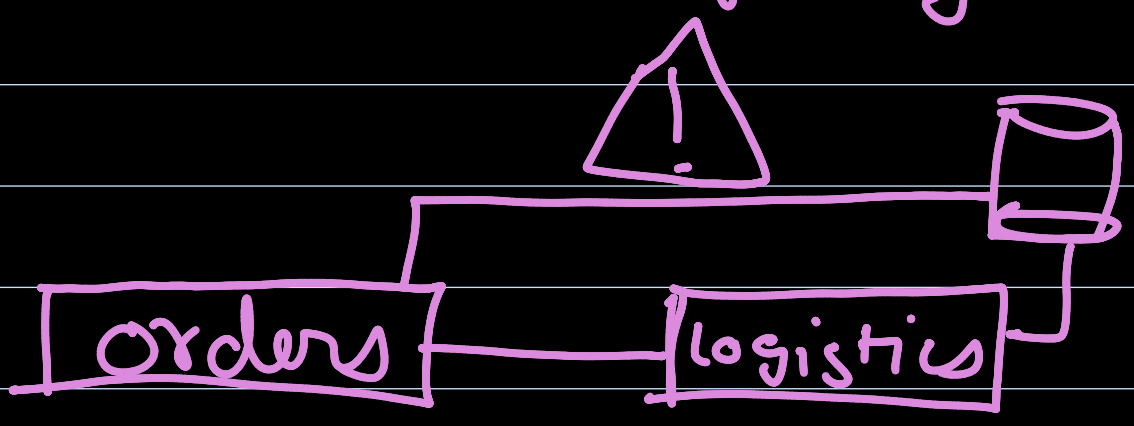
2 services need to know

1. Protocol
2. Rate limits
3. Public APIs
4. Authentication.

They don't need to know

1. DB
2. Architecture
3. PL used etc

Let's take a scenario where orders knows DB of logistics



This is not a good sign because:-

1 It defeats the purpose of logistics servic comm^n

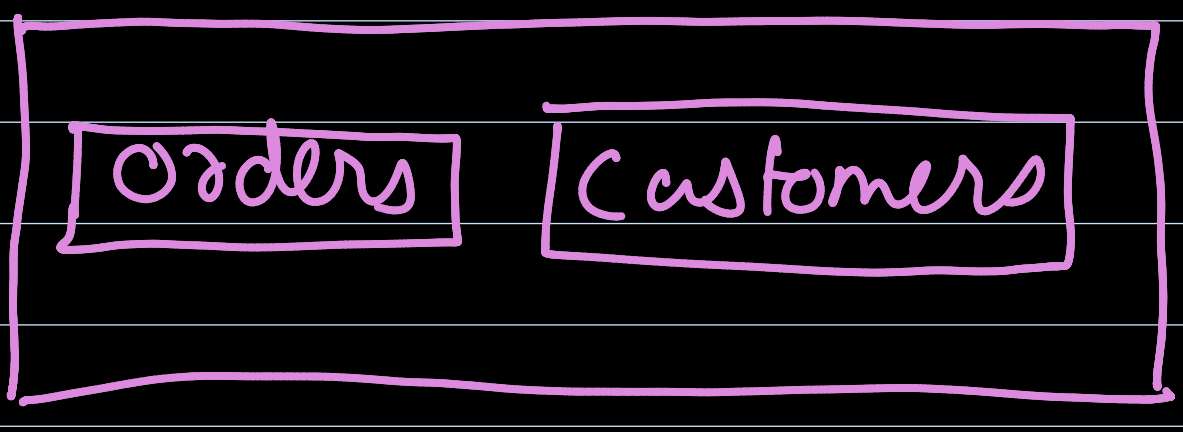
2 If the DB changes or credentials Change, then order service will go down.

2 High Cohesion (Srvs shld operate Indpntly)

This implies two very complimenting poinciples

1 Related behaviour sits together.

2 Un related behaviour resides separately.



In the above scenario, orders service also holds customers data and this is a single service.

Now if there is some change in orders then they have to take consent from Customers team as well.

That's why we should have diff behaviour fitting separate

Two microservices sharing the same codebase

eg: when transitioning from monolith to microservices
you tend to reuse the monolith codebase & fork out services

Now, when some changes in monolith codebase,
you will need to re-deploy other services sharing the codebase.

Deploying multiple services at once is **risky**

Remember -> Logistics vs Monolithic Eg