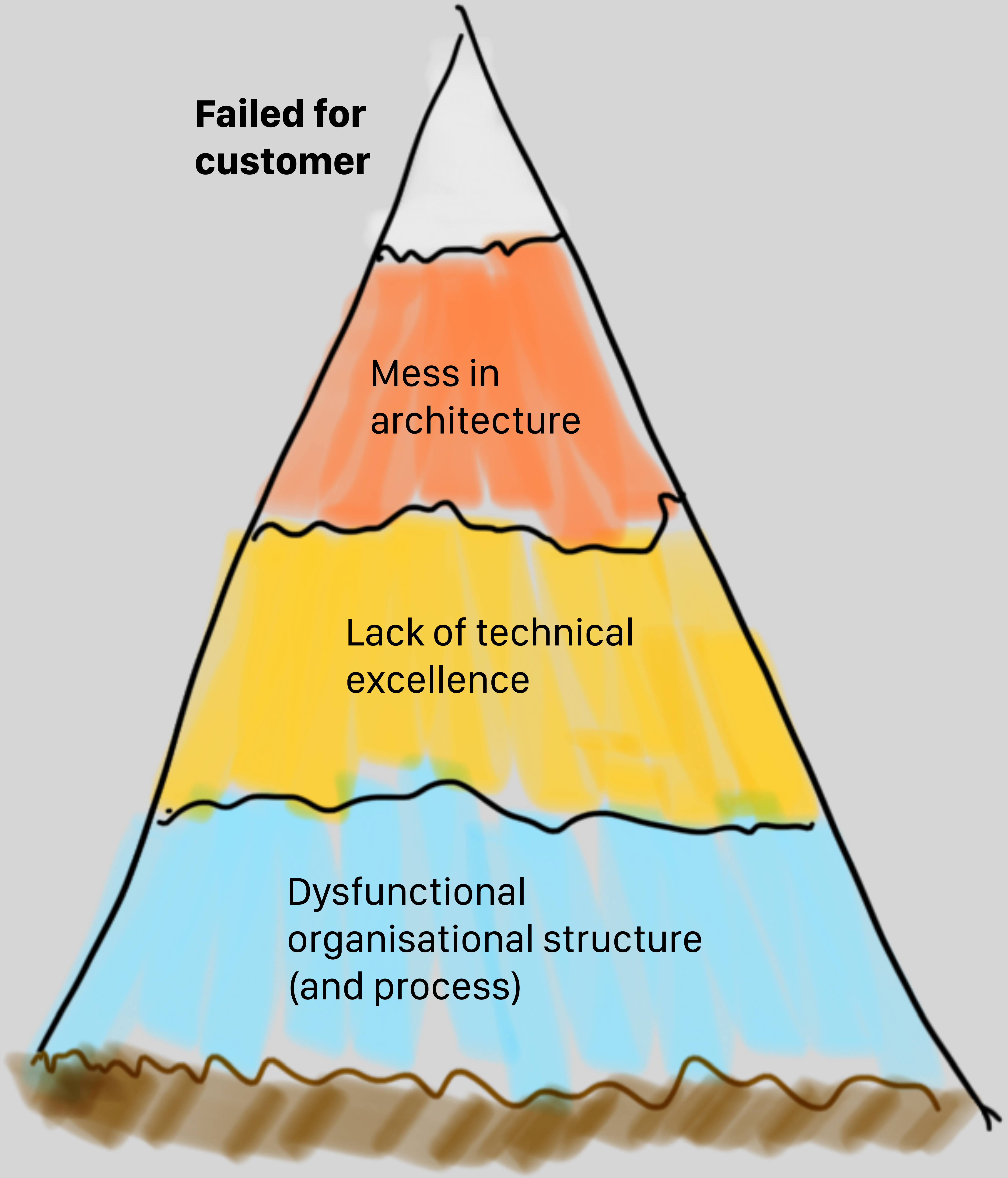
Why is independent teams with microservices a bad idea?

Viktor Grgic @vgrgic, viktor@odd-e.com, Odd-e Netherlands B.V. odd-e.com leanarch.eu less.works

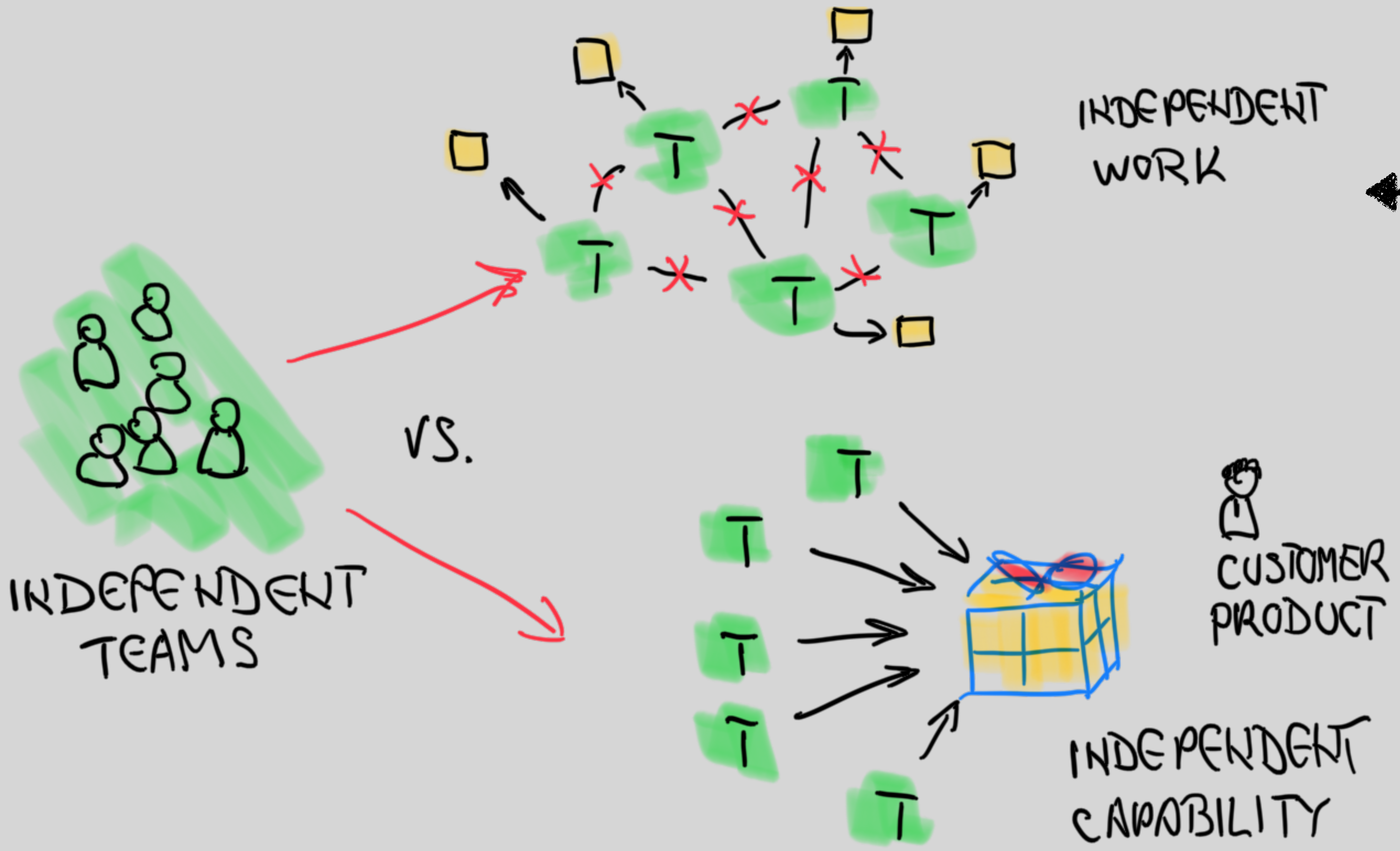
A major direct reason for failed product development is mess in architecture / code

…

also caused by simplistic theoretical nonsense


Failed for customer
Mess in architecture
Lack of technical excellence
Dysfunctional organisational structure (and process)

What do we mean with "independent team"?



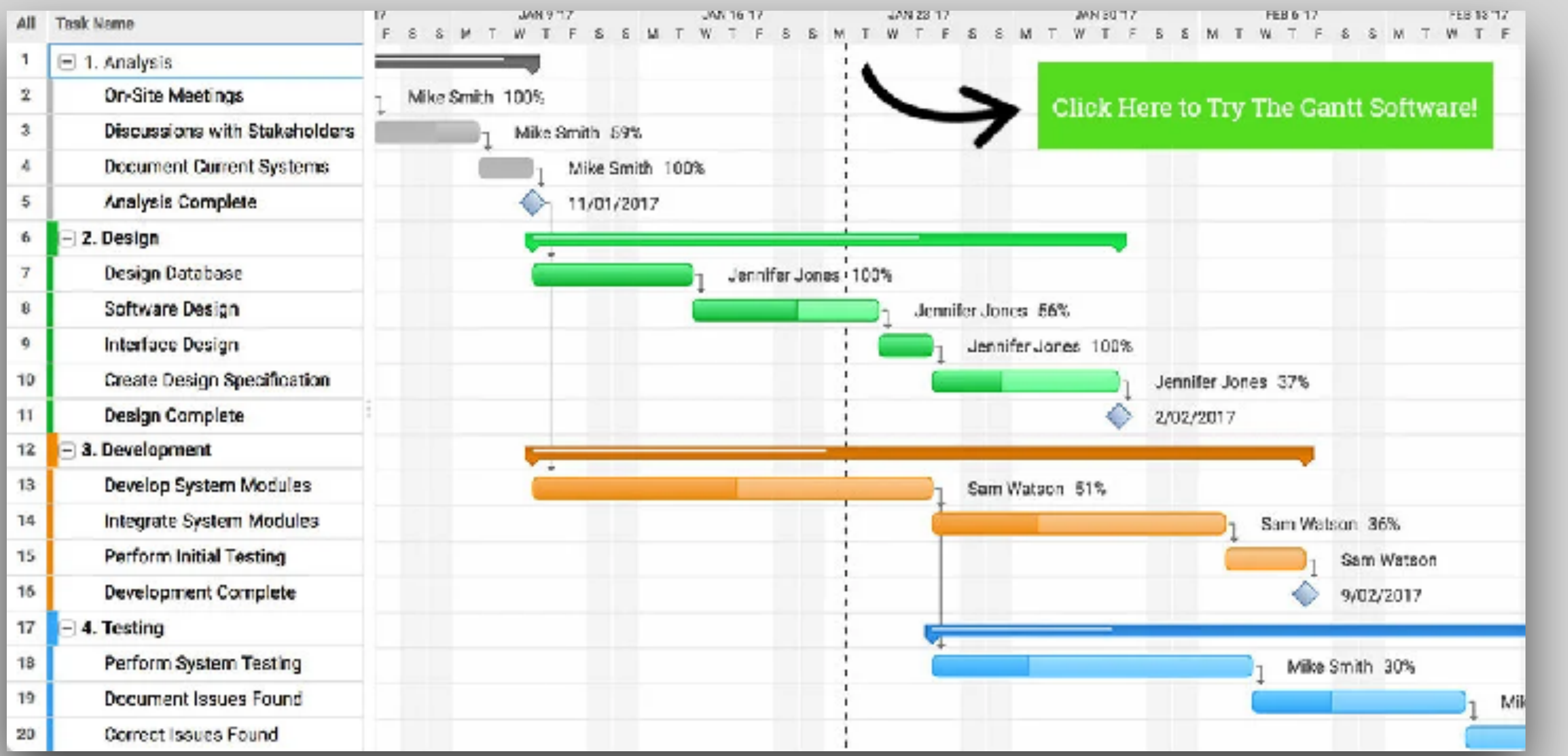
← Avoid

Types of dependencies

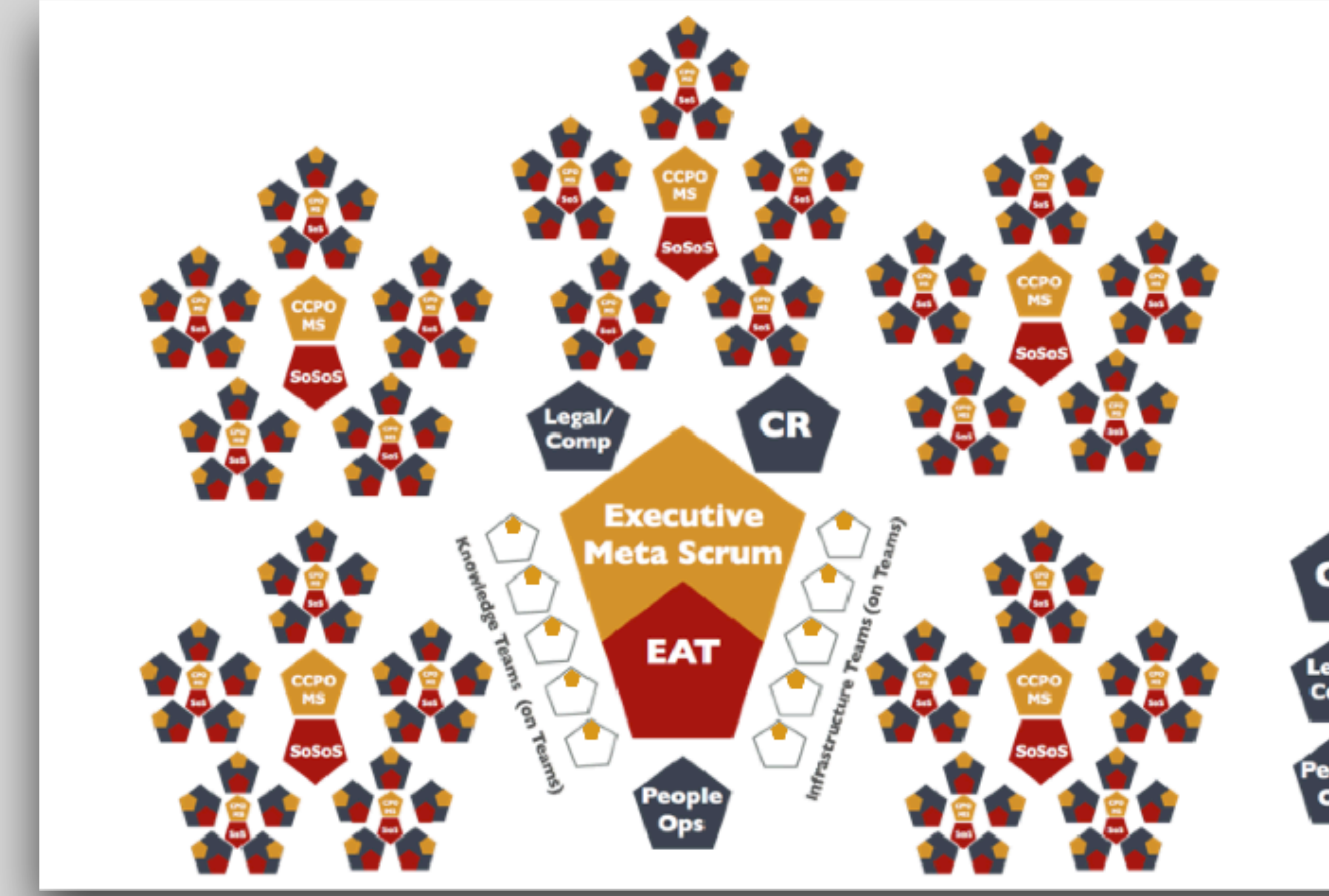
- **Between customer / users requests** (Product Backlog items): large request split up in smaller items require cohesion
- **Design / architecture:** decisions made by teams need to be aligned between each other and product or company strategy
- **(Shared or common) code / components / applications:** especially when developed by separate groups or even companies.
- Information, knowledge, experience
- Ability and authority to decide

“We don’t care” team

Branching postpones and exacerbates
cost of dependencies



DMC
(Dependency Management Circus)

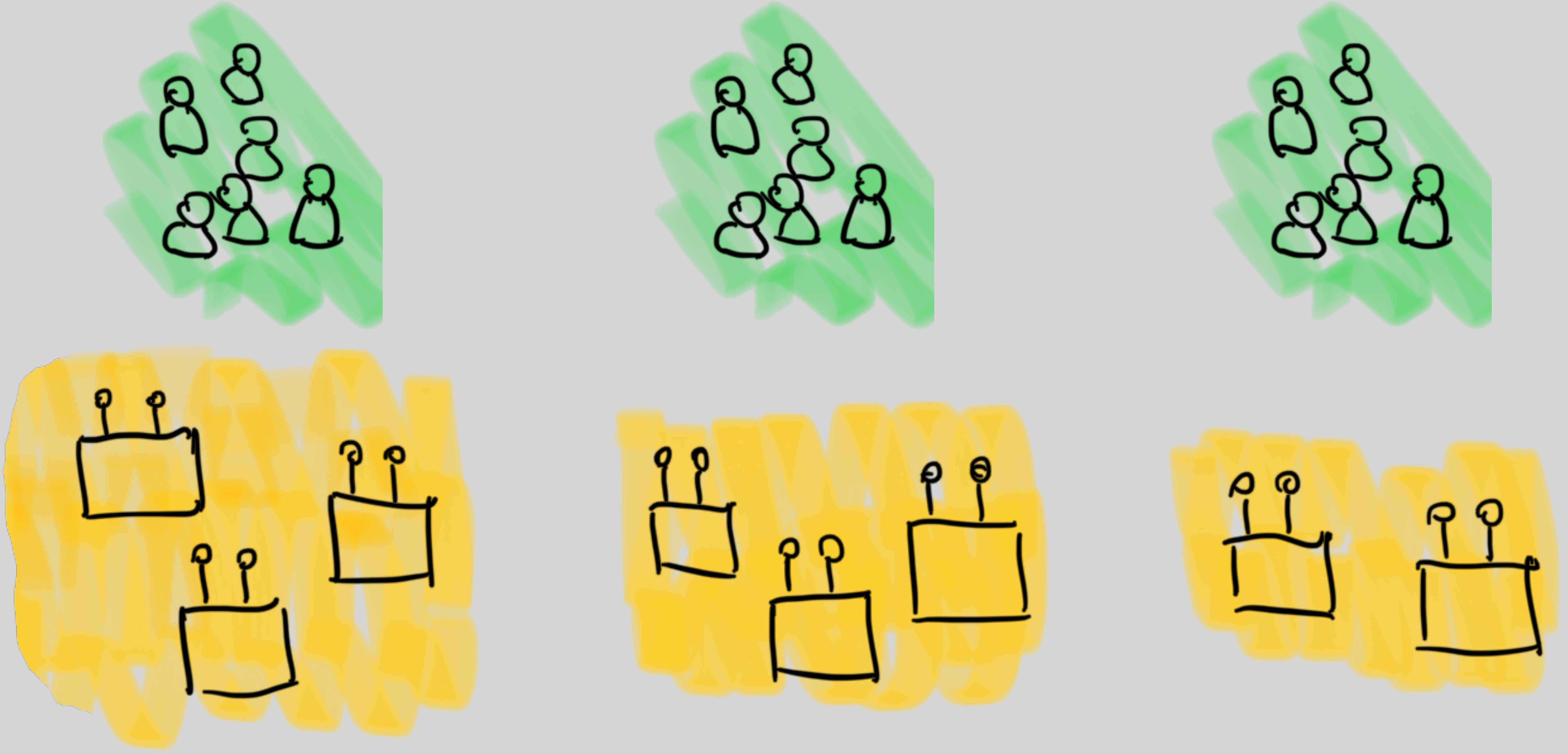


Coordination roles

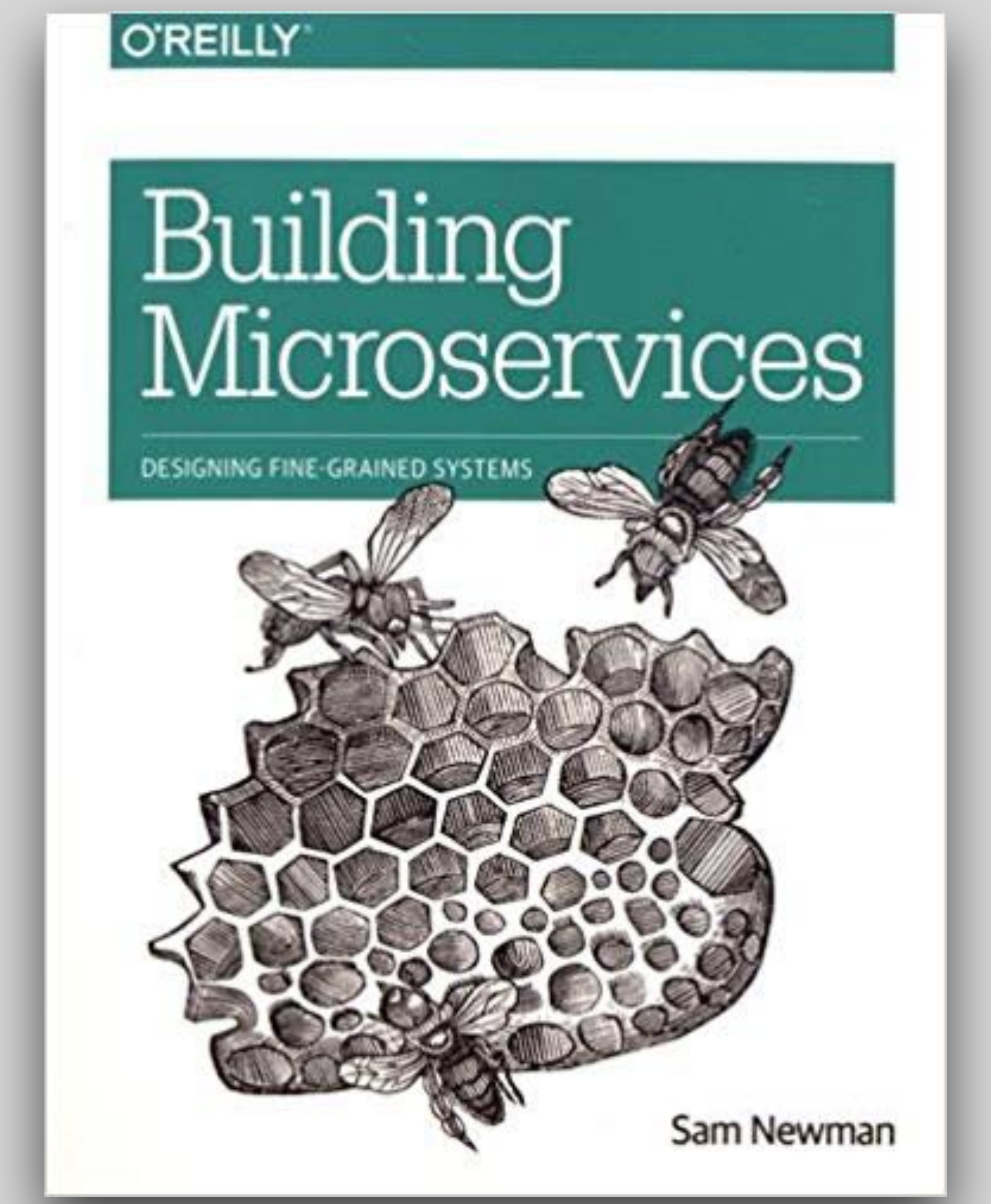
- POs
- SMs
- Train engineers
- PMs
- Team managers
- …

Microservices as a solution?

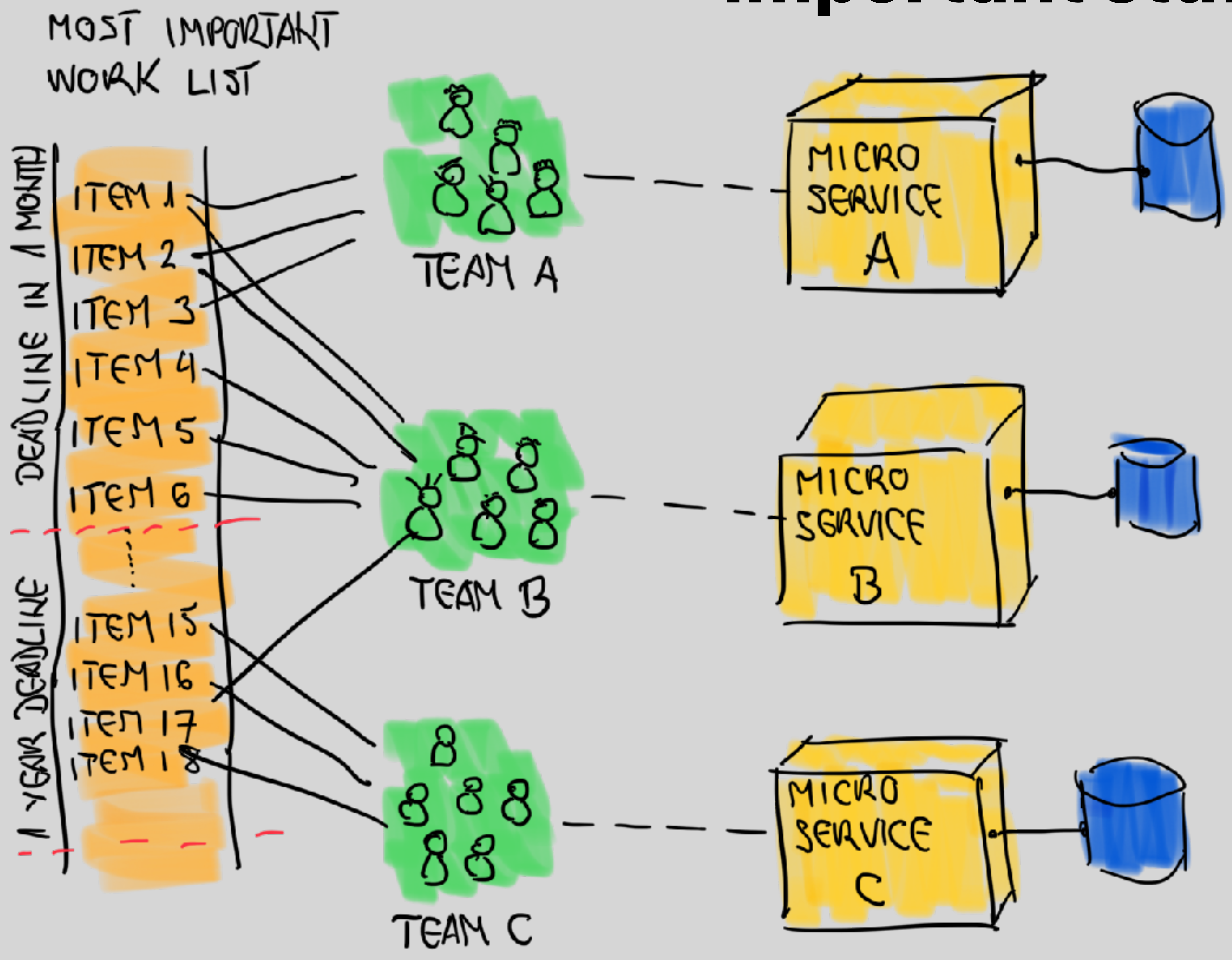
Team A Team B Team C



Product Sales Support



1st problem: Who is delivering important stuff?



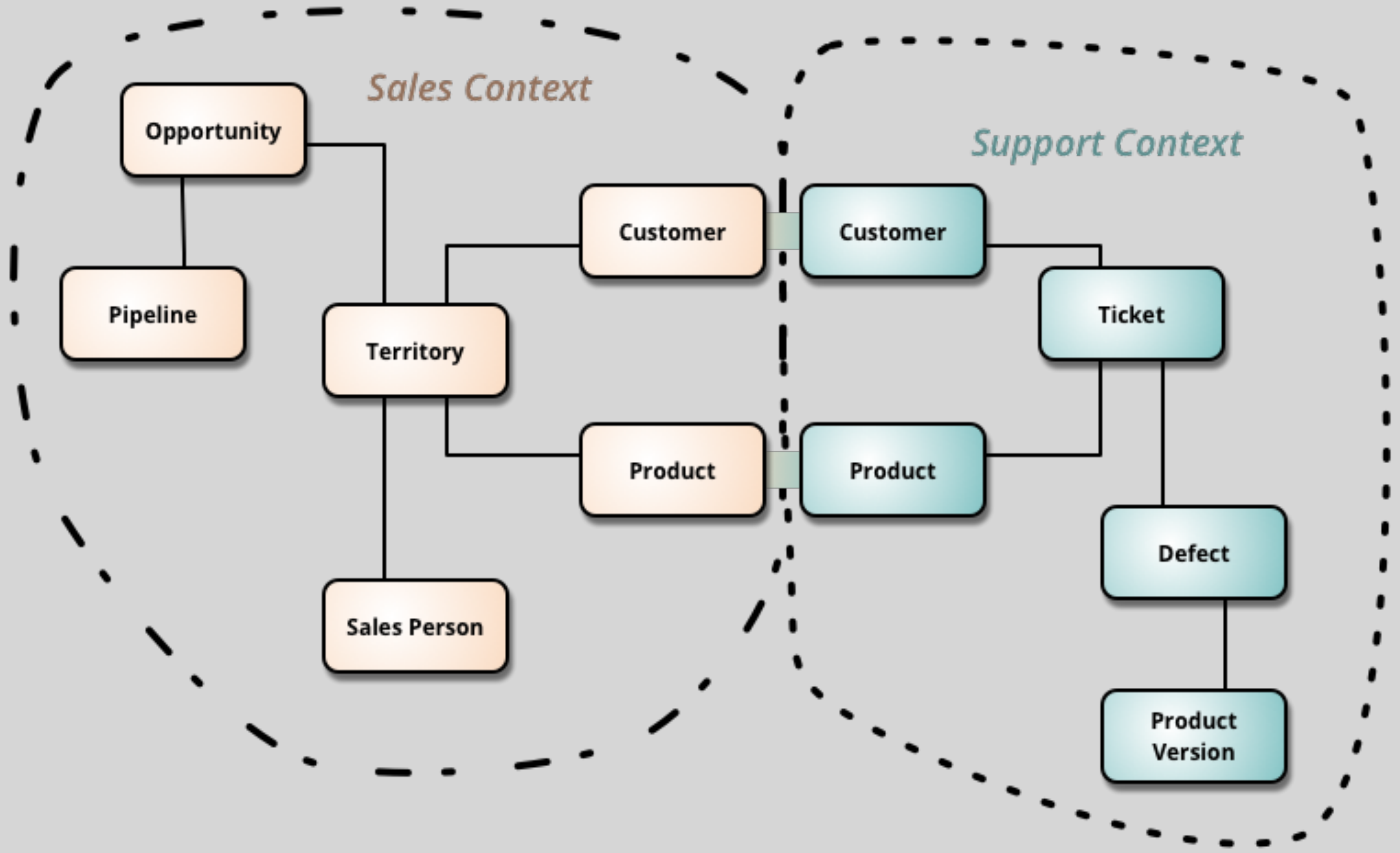
Dirty org/process hacks

- Continuously shift people between teams
- "Prioritize" based on ability and not importance

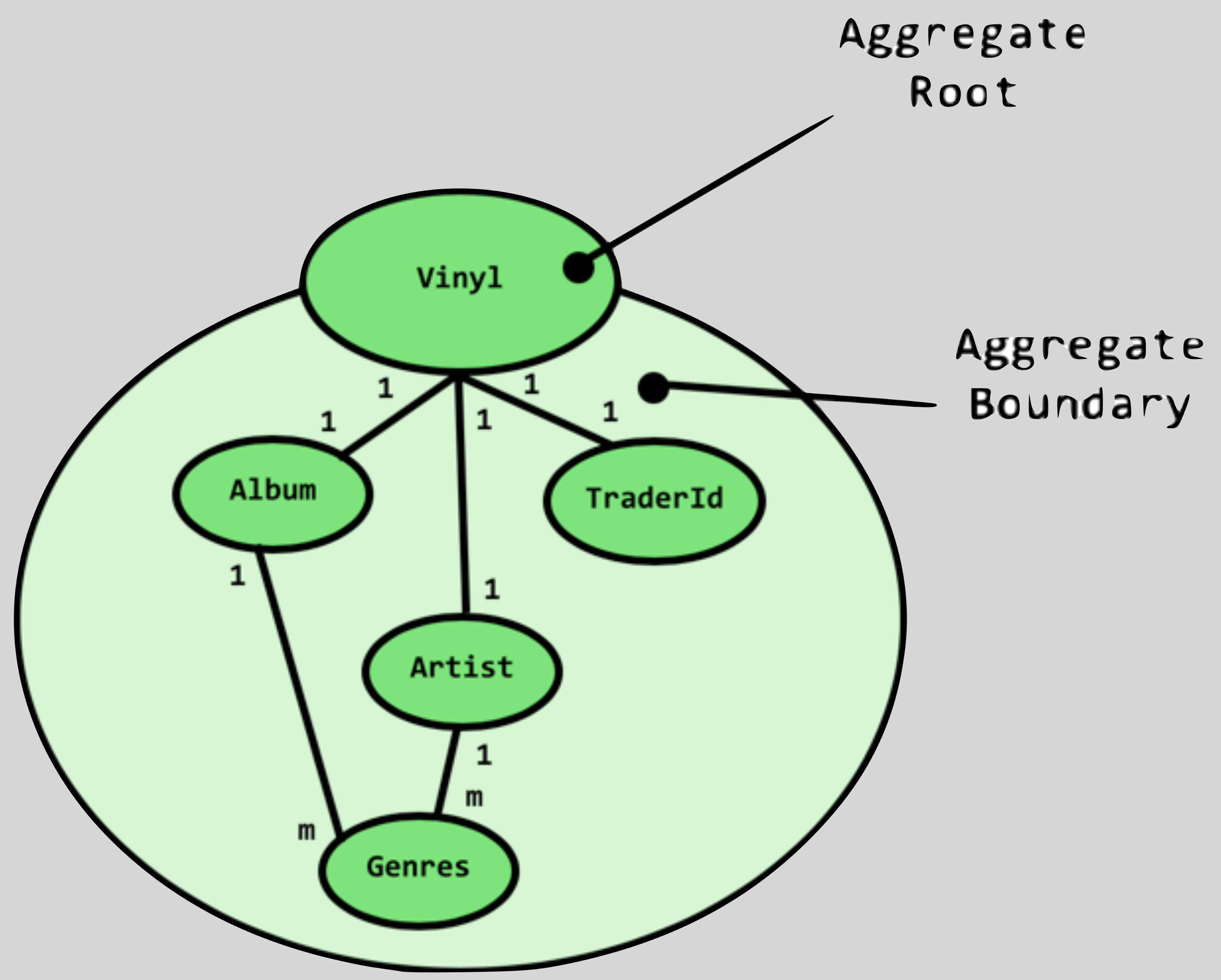
Less dirty hacks

- Teams work on each others' microservices

2n problem: expertise in Domain-Driven Design

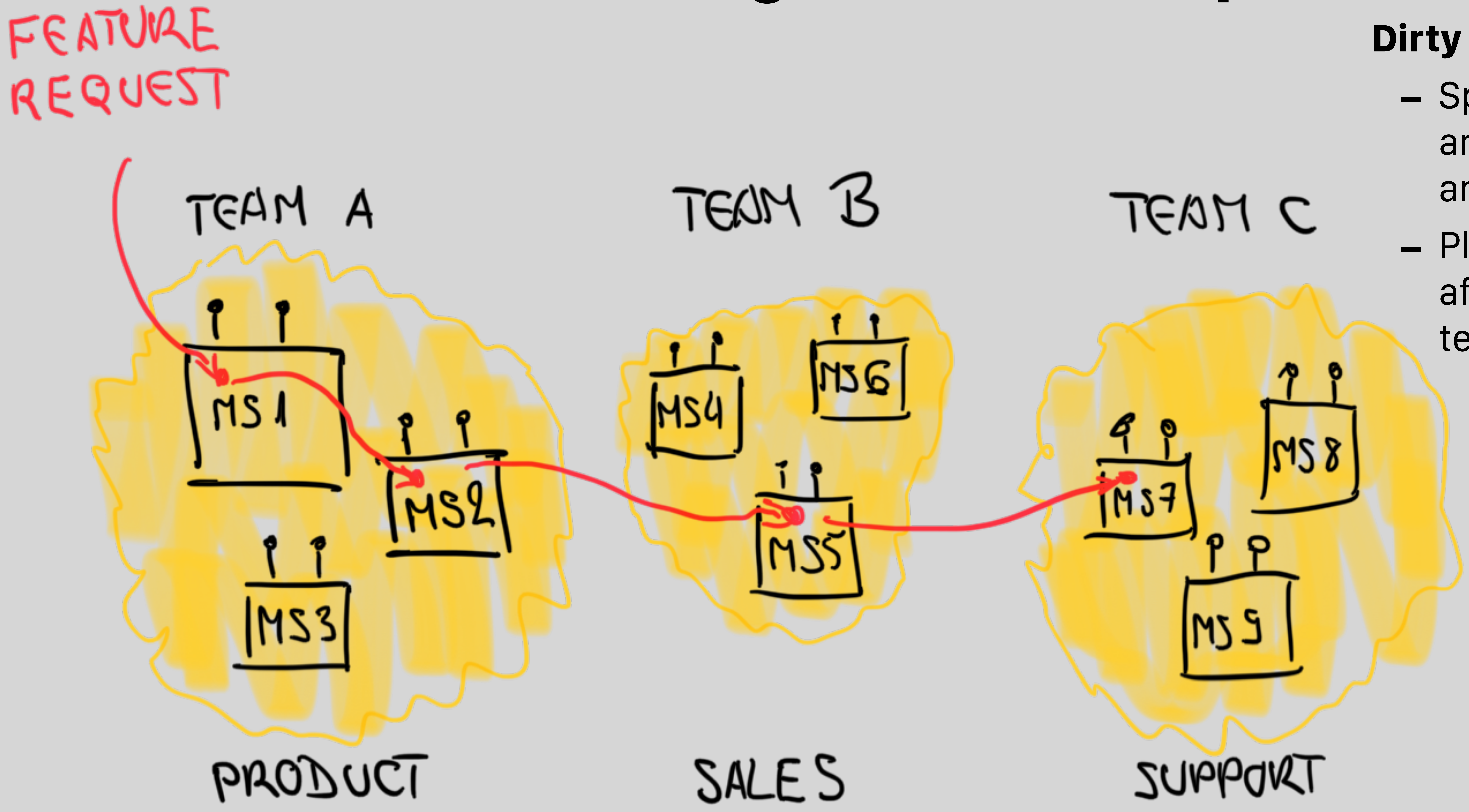


Bounded context



Aggregate

3rd problem:
Handling customer request



Dirty process hacks

- Split items up until they are not customer centric anymore
- Plus have someone afterwards integrate and test in combination

“We have resolved problem of crossing boundaries by a better segregation. Our features don’t cross bounded contexts”

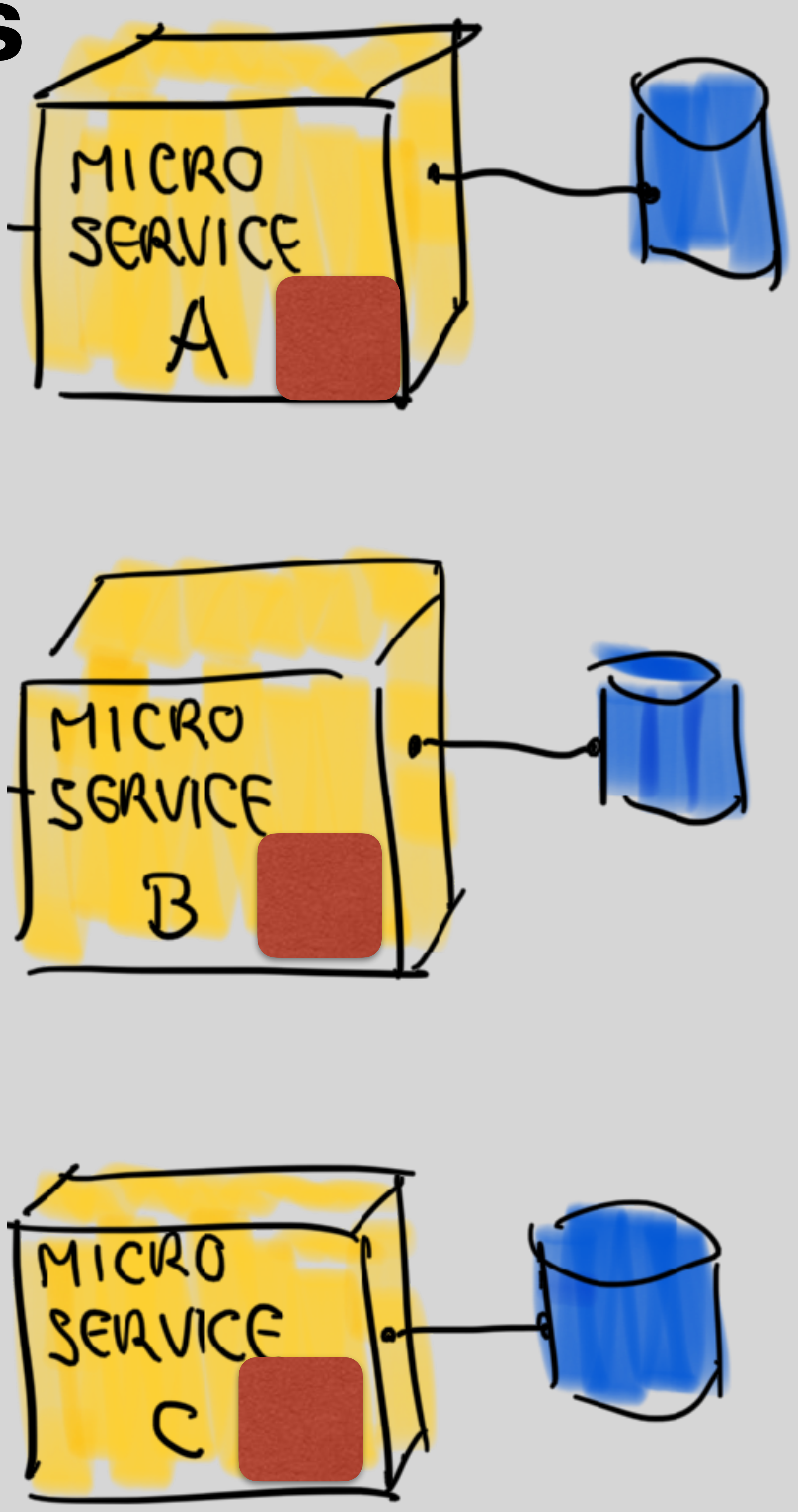
Really?

4th problem: Shared libraries

Lifecycle of shared lib

1) Duplication or need to use capability in other service discovered
2) Shared lib created and packaged in multiple services
3) Every change in shared lib requires deployment of all services (independent deployment gone & impact very large)
4) Dirty hack 1: work with multiple version of a shared lib
5) Nasty hack: When that becomes a mess, fork shared lib
6) Now we have much bigger duplication
7) Kill microservices, go back to monolith

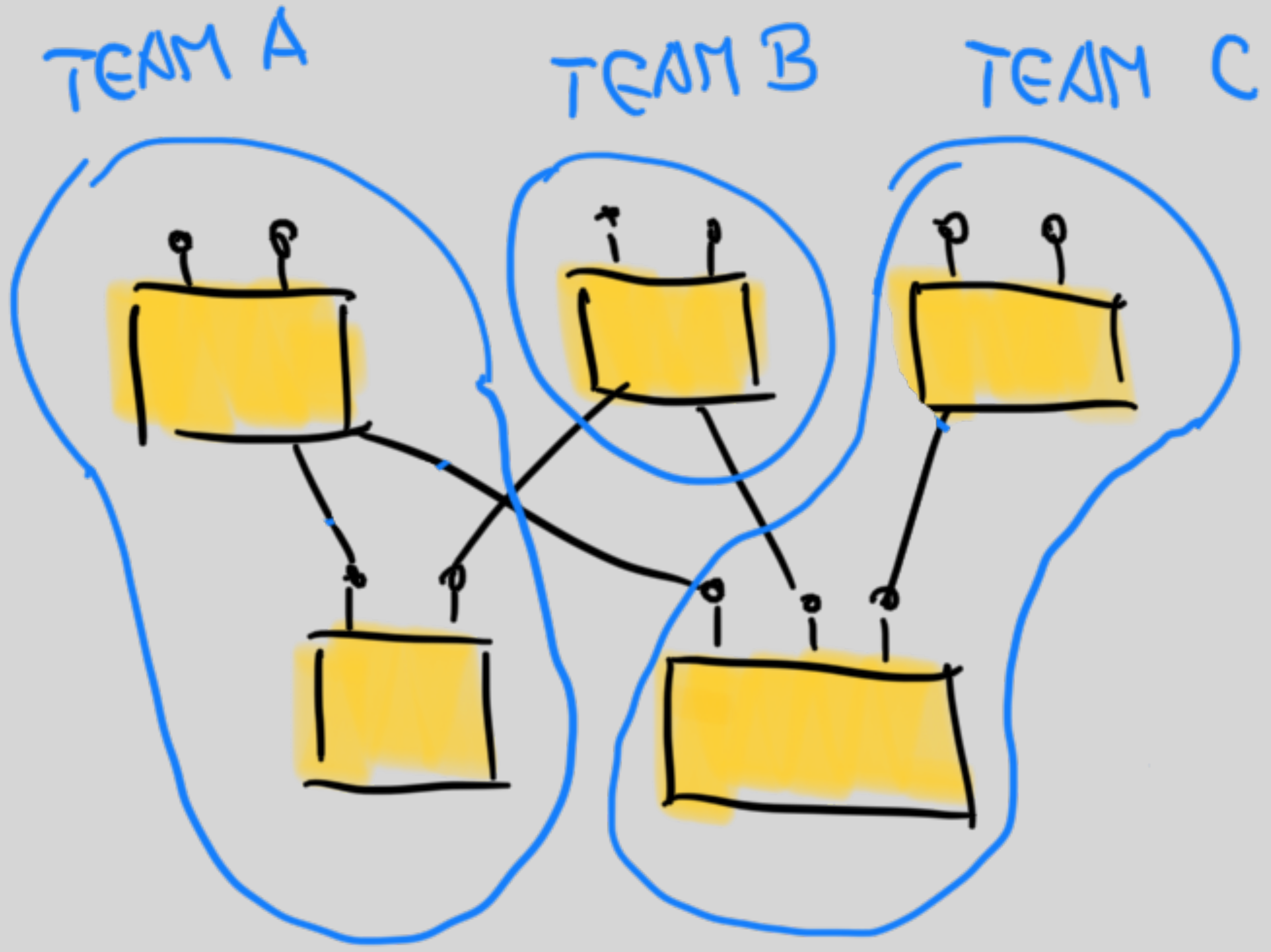
Recorded talk on InfoQ: "To Microservices and Back Again" - by Alexandra Noonan



High cohesion, low coupling is not NO coupling, high cohesion

Coupling between parts is what makes a product useful

5th problem: Shared services

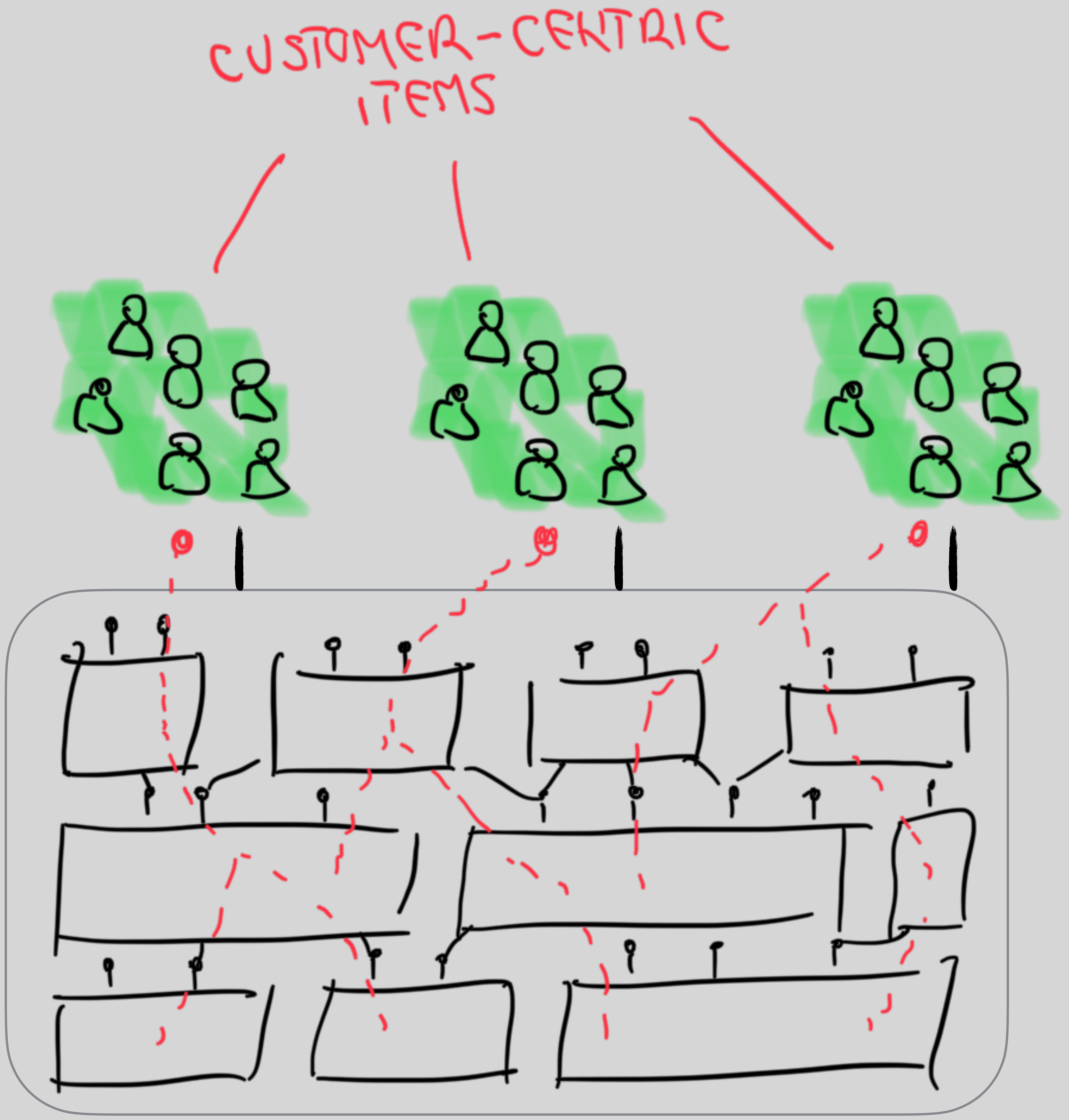


Solutions

1) Teams wait / depend on each other to change shared service
2) Shift shared service expert between teams
3) Split request into even smaller technical requests and manage dependencies
4) Internal open source where anyone can change with code custodians

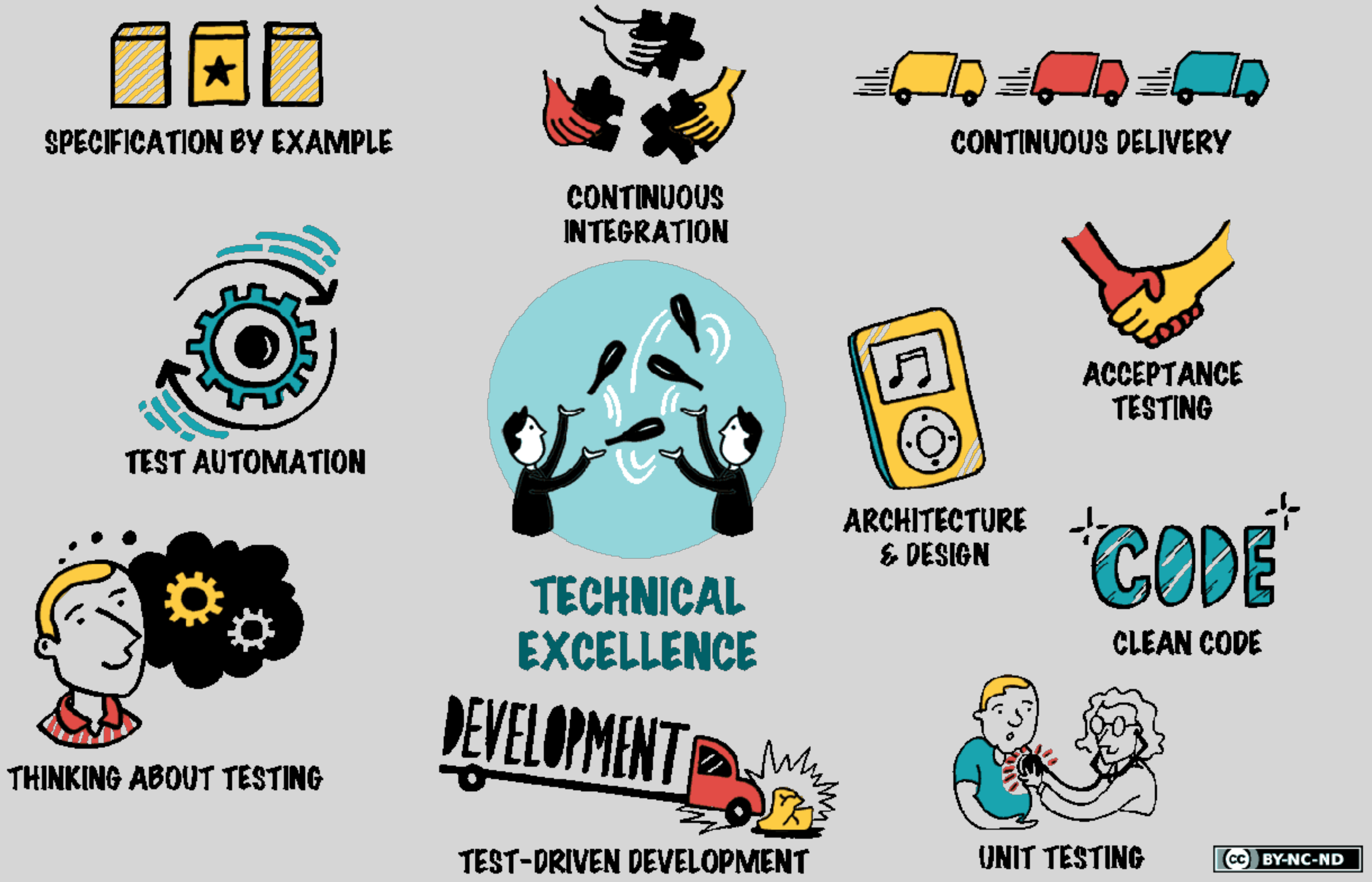
Therefore...

...separate architecture from team structure



Each team works from whole product point of view

Technical Excellence



Don’t avoid or postpone dependencies
Instead, reduce <u>cost</u> of dependencies

Prefer teams coordinating **directly** with each other

Prefer **synchronous** over asynchronous coordination

What about Conway's law

De volgende dag op school spelen de vriendjes van Tip samen, maar Tip doet niet mee. Hij zit een beetje verderop apart en houdt zijn lievelingsspeeltjes stevig vast. Hij wil ze niet delen met de anderen.
Maar nu zit hij daar alleen, terwijl zijn vriendjes zich erg amuseren.
Dit zijn mijn speeltjes, denkt Tip en hij kijkt een beetje boos. Tip heeft het helemaal fout. Op school is het speelgoed van iedereen, dat is net het leuke eraan.

Resources

- LeSS Site: (https://less.works/)
- LeSS slack group: less-works.slack.com (let me know your email and I will add you)
- LeSS Twitter: @less_works
- LeSS LinkedIn group: https://www.linkedin.com/groups/6968022/

- Feel free to contact me for further questions: viktor@odd-e.com