package Platform Engineering

import "Team Topologies"

type Principles **interface**

Architect

Cansu Kavili-Örnek

RED HAT OPEN INNOVATION LABS

Why listen to us?

Team Topologies

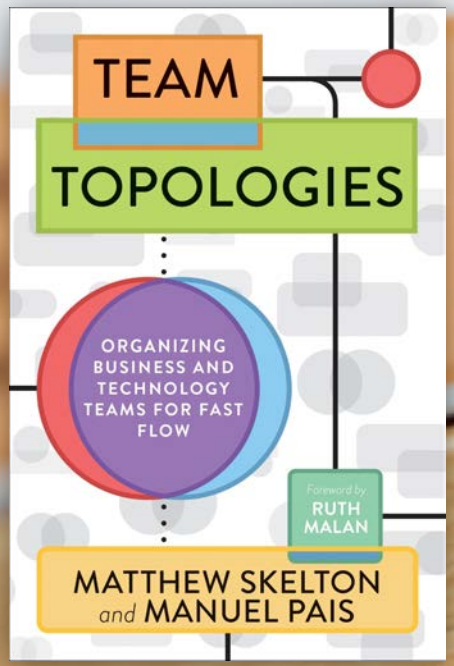
CLOUD NATIVE COMPUTING FOUNDATION

Trouble

Val Yonchev

Team what?

https://teamtopologies.com/



Let's talk Platforms

<Platforms>

Accelerate the flow of value
AND
Reduce cognitive load within the whole system
AND
Enable substantial autonomy of teams consuming them

</Platforms>

```
package Platform;

public class Platform team {

    /*
    *   Pay attention! Not what you thought it means
    */

    a grouping of other team types, which provide a
    compelling internal product to accelerate delivery
    of value by Stream-aligned teams
}
```

Words of Caution

- ➔ Team is the smallest unit of delivery (and measurement)
- ➔ Platforms reduce cognitive load and accelerate flow of value (or they shouldn't exist)
- ➔ Thinnest Viable Platform … is the maximum we should develop
- ➔ Cognitive load drives decisions (team-of-teams design)
- ➔ Treat the platform as a product
- ➔ Teams communicate through APIs

Principles from Team Topologies

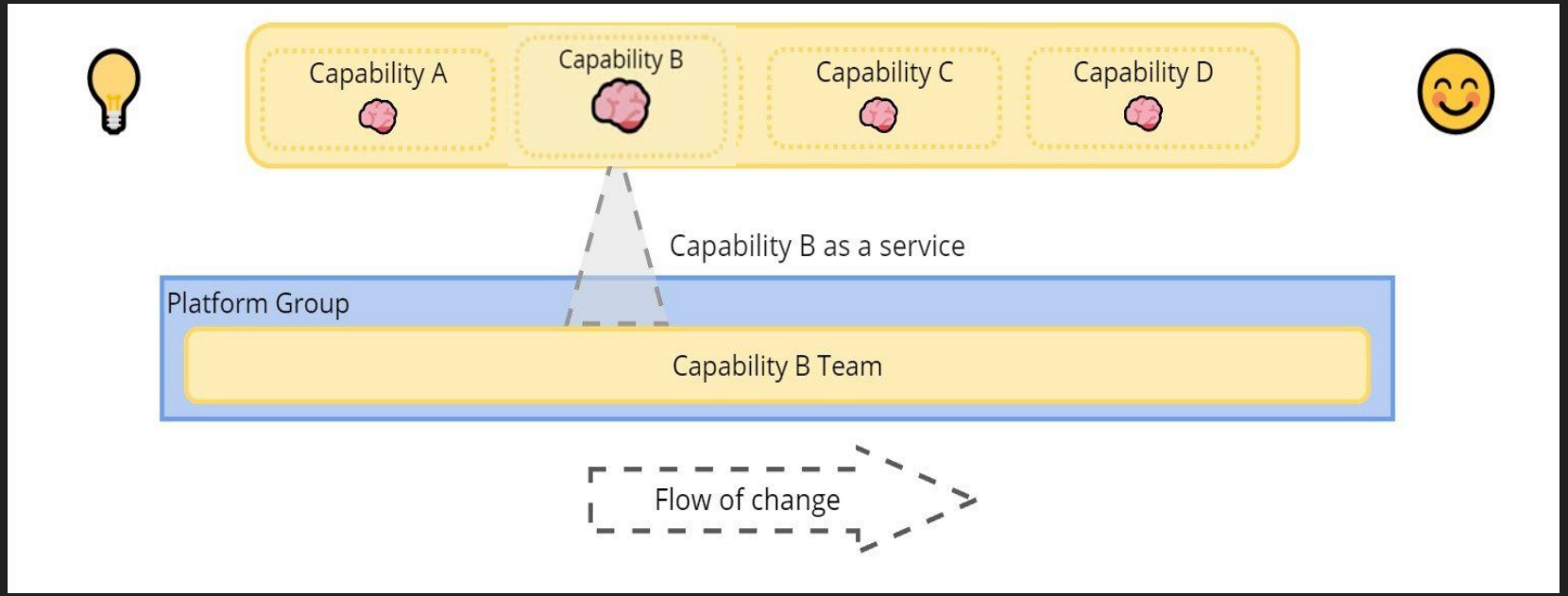
- ➔ Platforms are never intuitive and easy enough
- ➔ Team Topologies is a VERB, not a label

DevelopMENT Experience

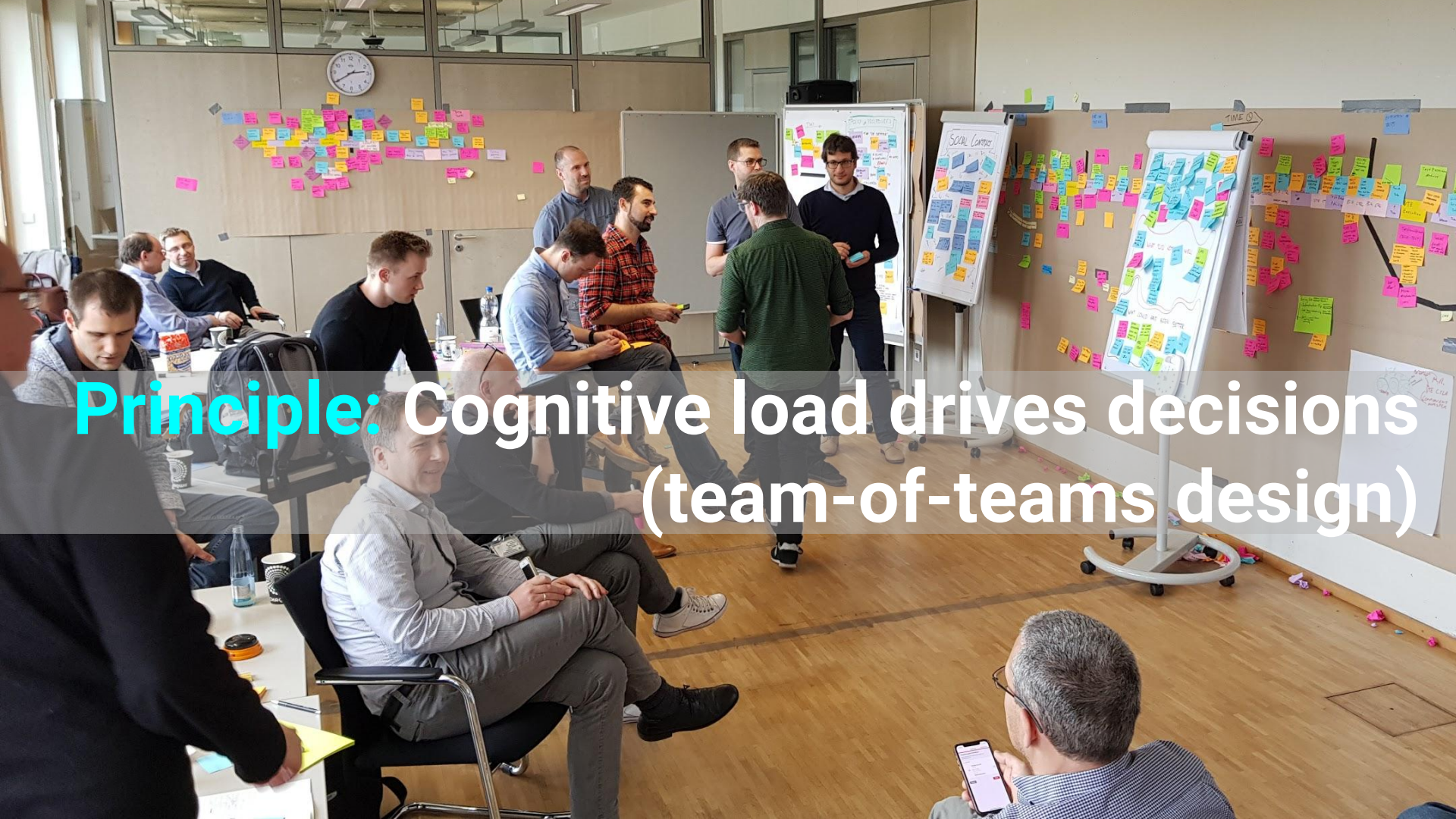
DevelopER Experience

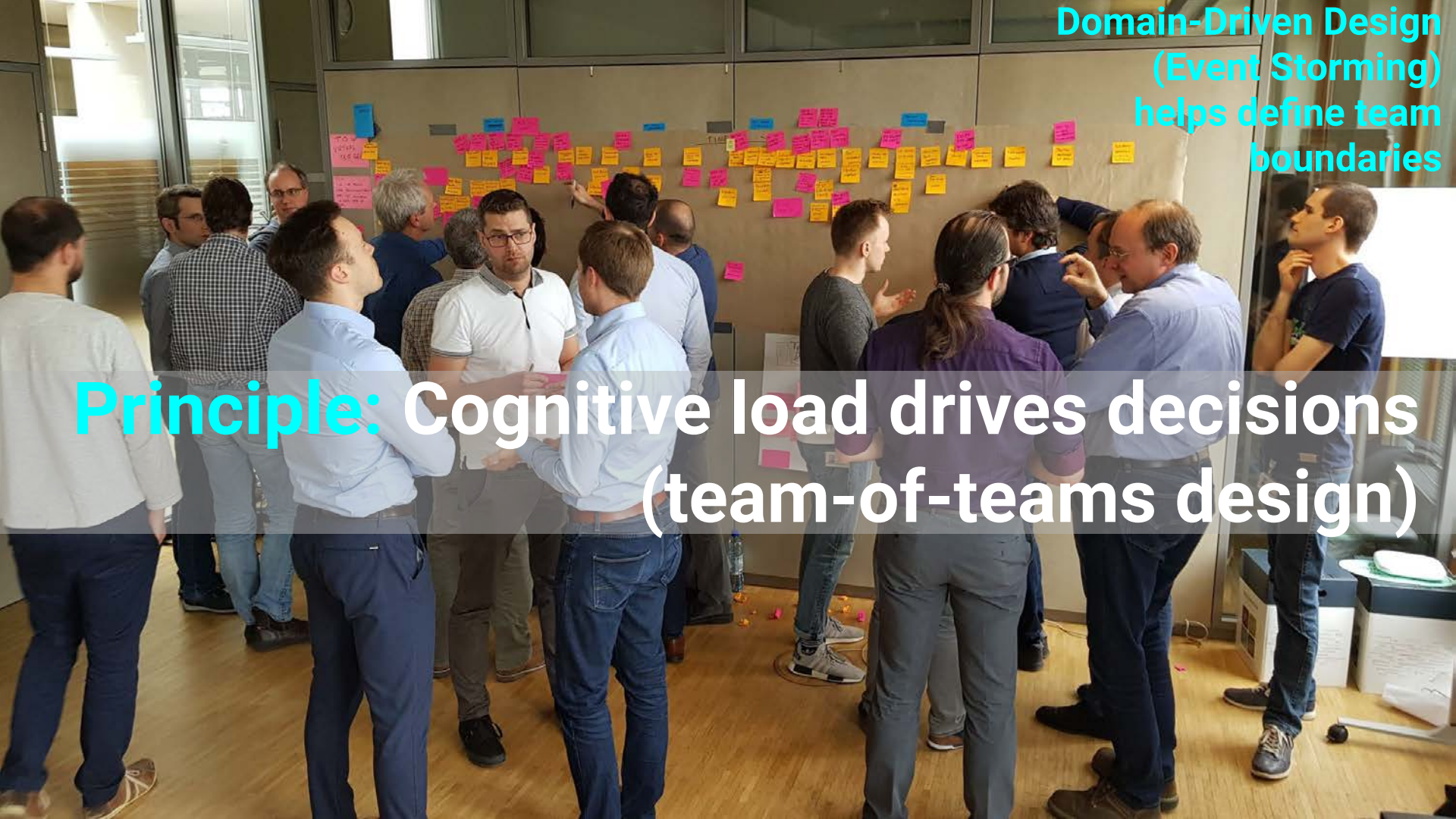
Principle: Team is the smallest unit of delivery (and measurement)

Principle: Platforms reduce cognitive load AND accelerate flow of value (or they shouldn't exist)

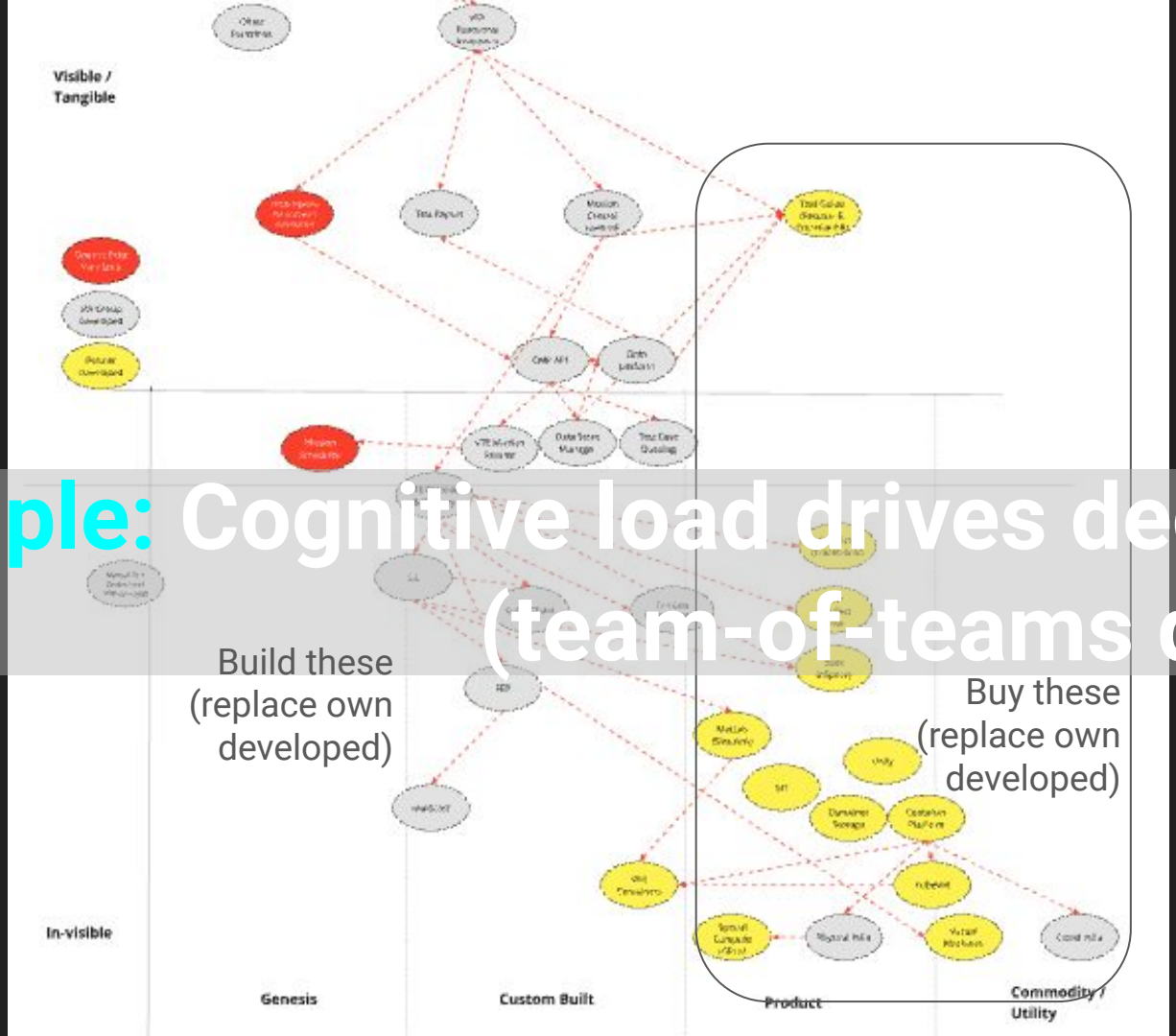


Principle: Thinnest Viable Platform … is the maximum we should develop


Principle: Cognitive load drives decisions
(team-of-teams design)


Domain-Driven Design
(Event Storming)
helps define team
boundaries
Principle: Cognitive load drives decisions
(team-of-teams design)

Wardley Mapping helps define what to build, what to buy

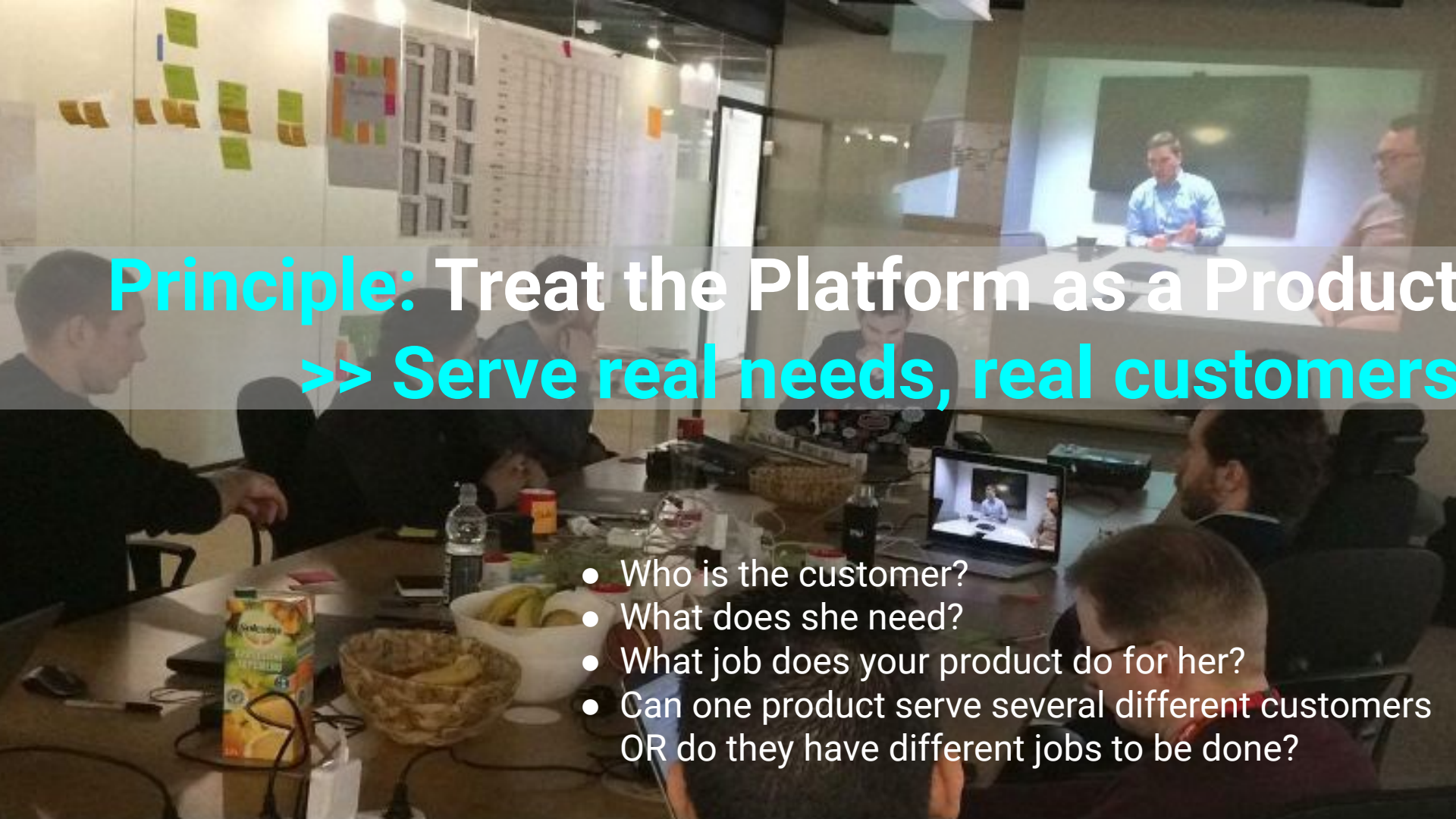


Principle: Cognitive load drives decisions (team-of-teams design)

Build these (replace own developed)

Buy these (replace own developed)

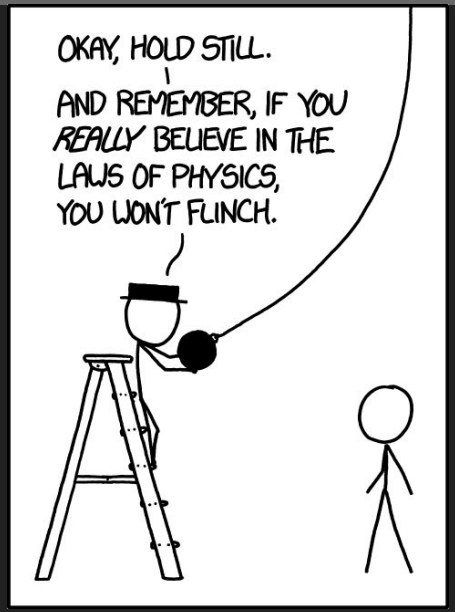
Principle: Treat the Platform as a Product

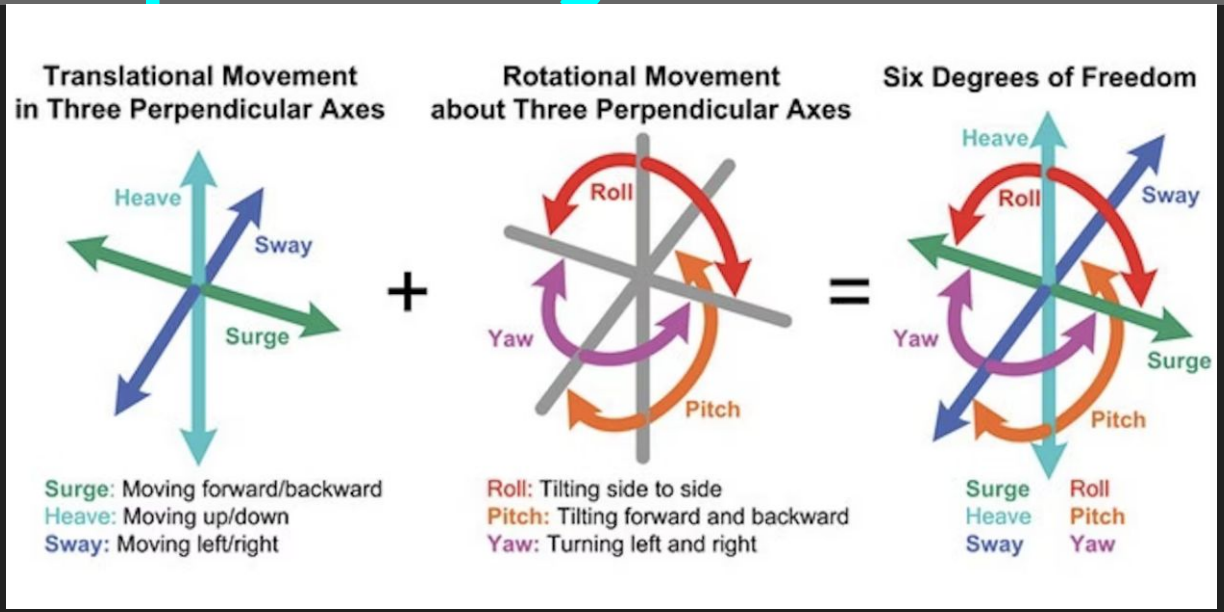


Principle: Treat the Platform as a Product
>> Serve real needs, real customers

- Who is the customer?
- What does she need?
- What job does your product do for her?
- Can one product serve several different customers OR do they have different jobs to be done?

Principle: Treat the Platform as a Product
>> Development Degrees of Freedom





Principle: Treat the Platform as a Product
>> Development Degrees of Freedom

"Black box" Services

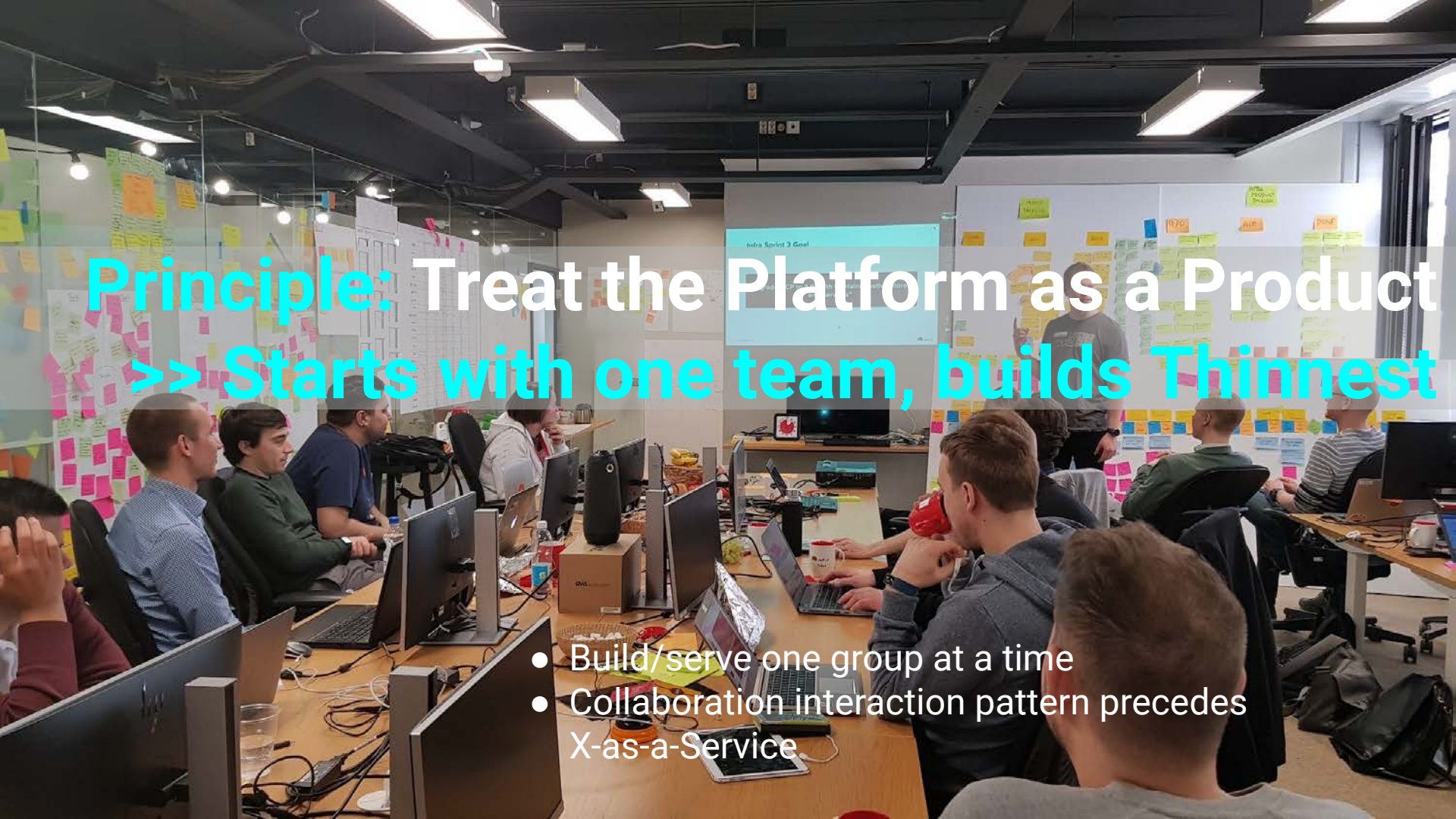
“I love that everything just works!”

Configurable Services

“ I can tweak things the way I need them.”

Modifiable Services

“The platform lets me experiment and innovate, and I can contribute my enhancements back to improve it.”



Principle: Treat the Platform as a Product
>> Starts with one team, builds Thinnest

- Build/serve one group at a time
- Collaboration interaction pattern precedes X-as-a-Service

Principle: Treat the Platform as a Product

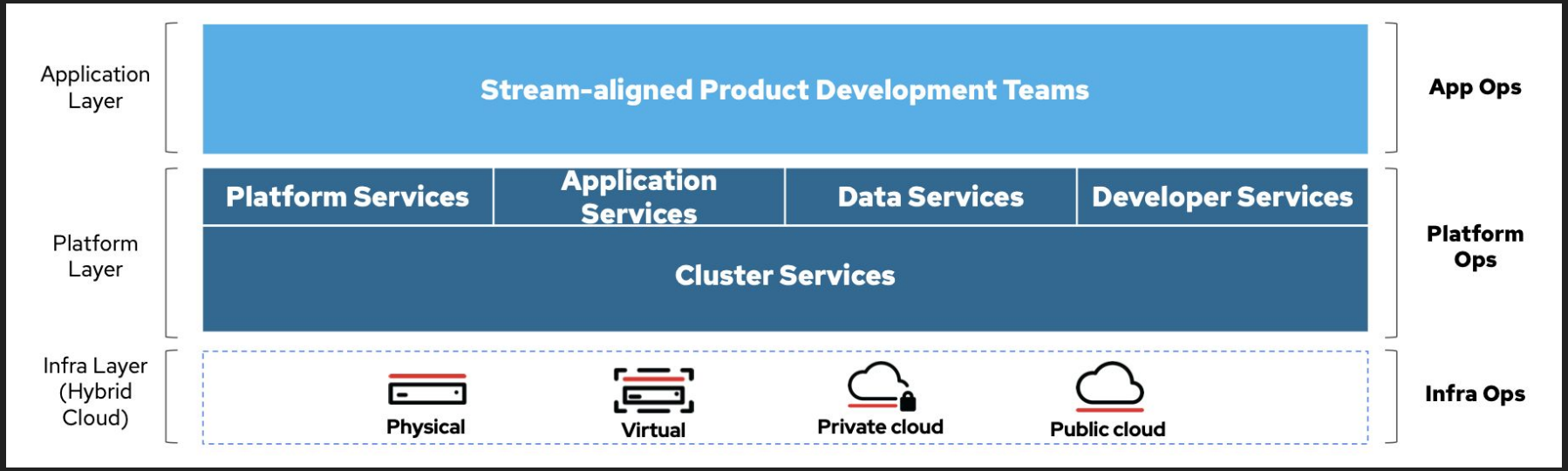
>> Competition drives progress

- Competition drives progress
- Product use is optional
- Technology evolve and sometimes you need to switch from in-house to commodity

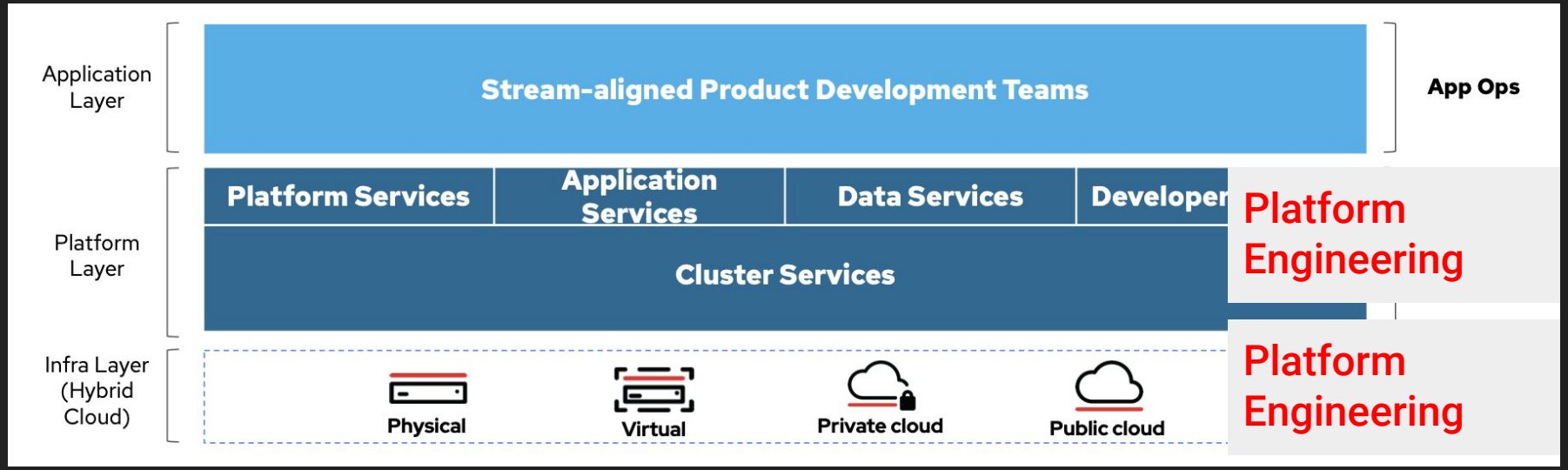

For your information, there's a lot more to Platforms than people think... Platforms are like onions. Onions have layers. Platforms have layers...

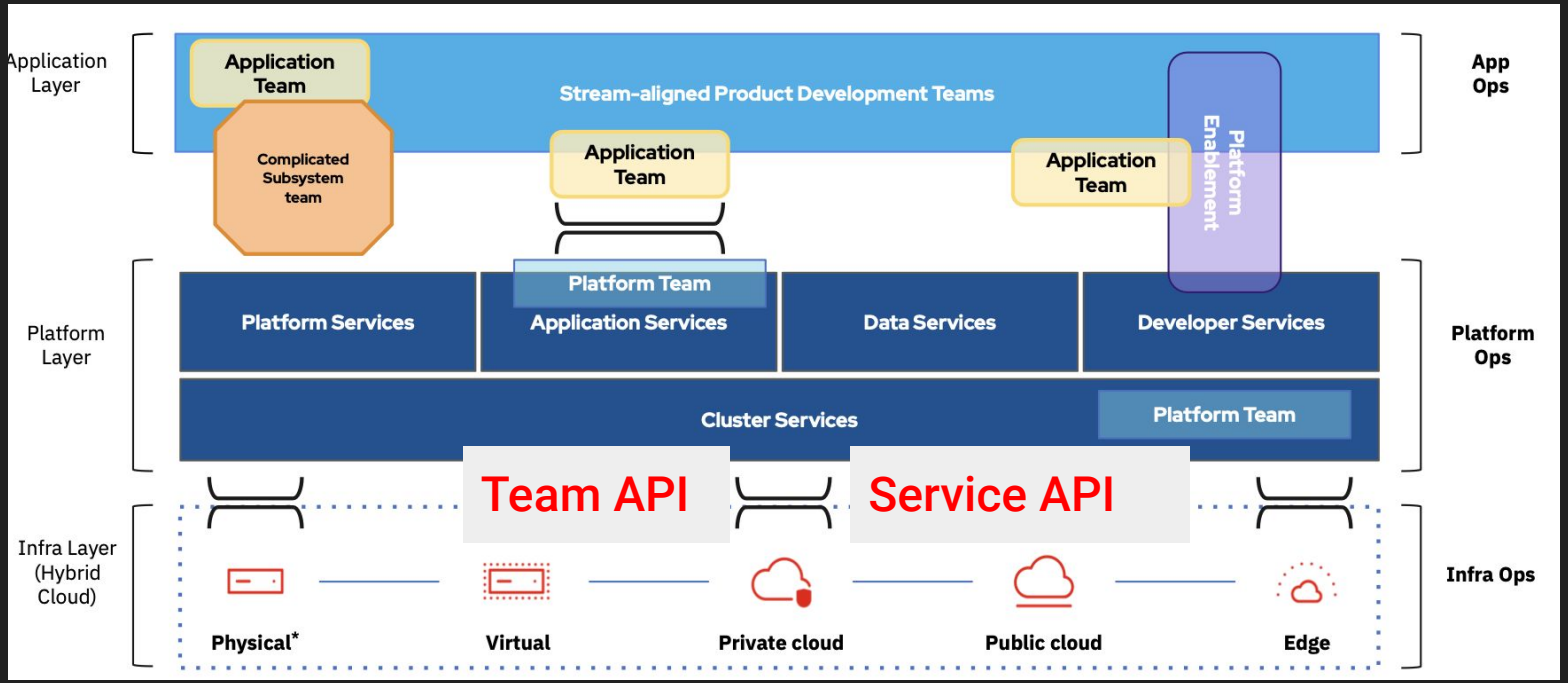
```
package Platform;

public class Platform team {

    /*
    *   Pay attention! Not what you thought it means
    */

    a grouping of other team types, which provide a
    compelling internal product to accelerate delivery
    of value by Stream-aligned teams
}
```

Principle: Treat the Platform as a Product
>> Platform teams have cognitive load and needs too



Principle: Treat the Platform as a Product
>> Platform teams have cognitive load and needs too





Principle: Teams communicate through Team APIs


Principle: Platforms Products are never
intuitive or easy enough

Principle: Team Topologies is a VERB, not a label

- Continuous effort
- Not always great results
- New need every day
- Practice - practice - practice

Platform Engineering is the culture of building platforms services which are:

- Easy to consume
- Reduce cognitive load (of those consuming them)
- Continuously evolved as needs evolve
- Economically viable

book time with Val

cansu@redhat.com
linkedin.com/in/ckavili

val@teamtopologies.com
linkedin.com/in/valyonchev

Thank you for listening!