Erin Gannon | Spencer Sugarman | Theresa Liberman | Yann Riche | Gerianne Bartocci

# Harmonizing Service Terminology

## Understanding users' language behind cloud architecture

December 2018

Contact:  ergannon@google.com

# Background

# Purpose

“Service” is an overloaded term used in a variety of ways across products, which could be confusing to users. This study is the first in a two-phase effort to:

- Expand on previous naming research to learn how users understand and use the term ‘service’ and other overloaded terms when referring to architecture
- Establish greater clarity in our use of ‘service’ across products by aligning the term with user expectations

Research questions:

- How do users apply the term ‘service’ in the context of our products, if at all?
- How do users conceptualize the term ‘service’ outside a specific architecture?
- Are there better words or qualifiers for the way we currently use, and intend to use, ‘service’ in our products? If so, what are they?

# Method

Two data collection approaches:

- Five in-person focus groups (2-4 people) were conducted to understand what ‘service’ means to our potential users, and how it relates to specific concepts (e.g., Kubernetes service, Knative service, etc.)
- UserTesting.com fielded 3 additional unmoderated responses to get more targeted data

19 external participants [details]:

- 4 primarily use Google Cloud Platform (GCP), 10 primarily use another cloud provider, 5 primarily on-prem
- Mix of Kubernetes experience
- Mix of operator and developer roles

# Study structure

Sketch their own architecture

1

Freely label generic diagrams

2

Label again with word list

3

“What is a service?” discussion

4

[Group discussion followed each step]

Sketch their own architecture

Freely label generic diagrams

Label again with word list

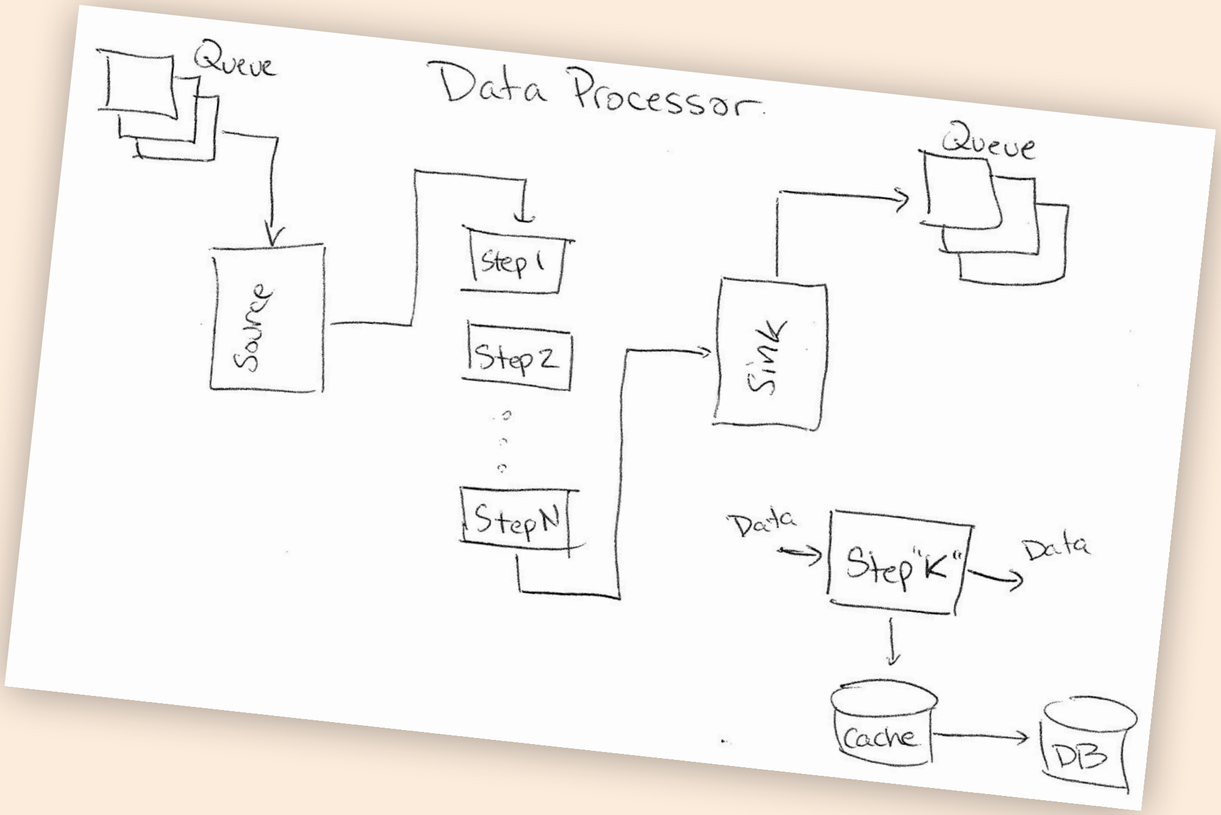
“What is a service?” discussion

1

2

3

4




Participants were asked to sketch a diagram of their own software architecture and label the pieces

Sketch their own architecture

Freely label generic diagrams

Label again with word list

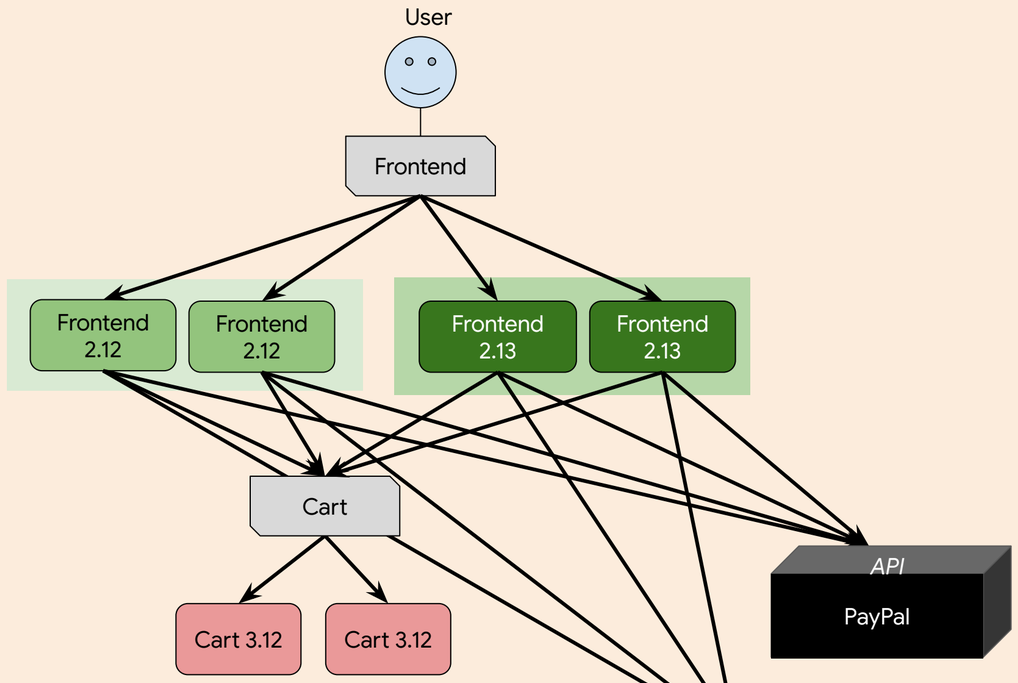
“What is a service?” discussion

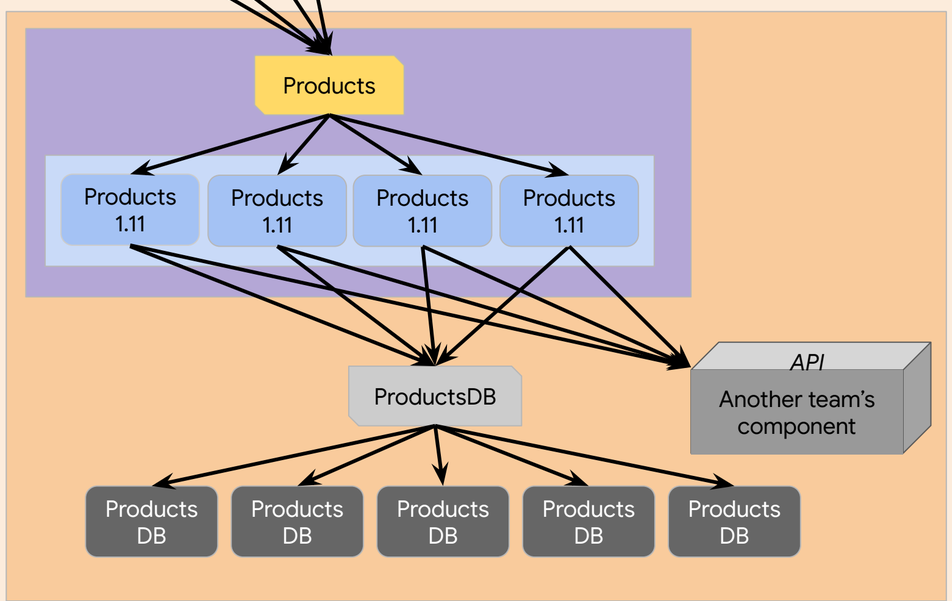
1

2

3

4




This software architecture topology diagram was presented piece by piece, and participants could use any word they choose for the first round of labeling

After the full diagram was labeled and the group discussed their decisions, participants could revise their terms if they changed their mind

See the Appendix for details on what each piece of the diagram represents

Sketch their own architecture

Freely label generic diagrams

Label again with word list

“What is a service?” discussion

1 2 3 4

**Agent**
**Alias**
**App**
**Build**
**Component**
**Dependency**
**Deployment**
**Endpoint**
**Engine**
**Environment**
**Instance**
**Job**
**Load balancer**
**Microservice**
**Module**
**Node**
**Package**
**Proxy**
**Release**
**Replica**
**Replica set**
**Revision**
**Route**
**Service**
**Solution**
**Version**
**Worker**
**Workload**

The same diagram was labeled again using only words from this list

Sketch their own architecture

Freely label generic diagrams

Label again with word list

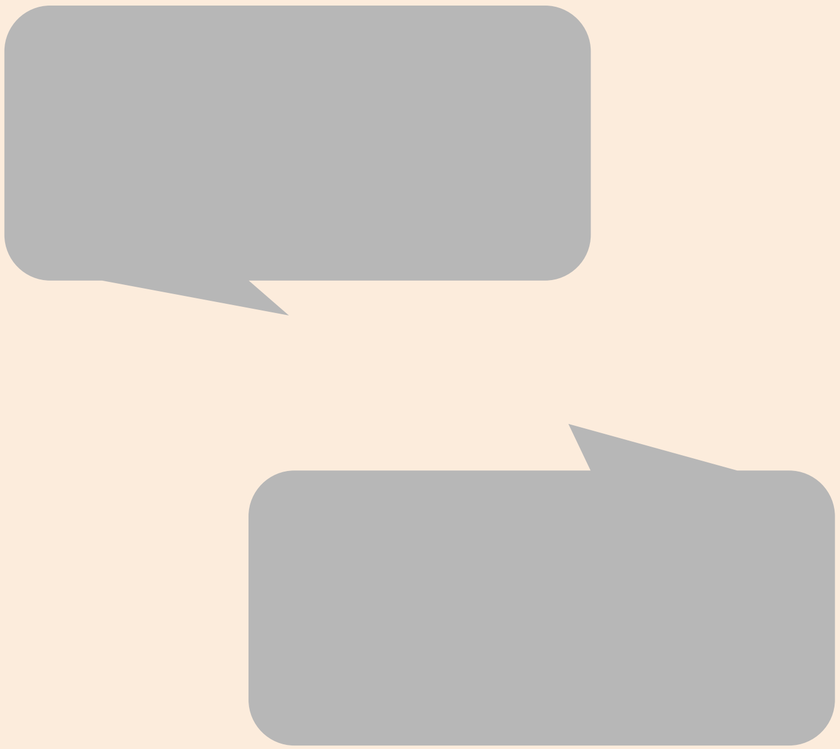
“What is a service?” discussion

1

2

3

4



Finally, participants had a group discussion about what ‘service’ means to them and their team

# Key findings

# Users perceive ‘services’ to be **independent** and **opaque** to the consumer

Services independently provide value without consumers knowing the inner-workings

“A service is an endpoint that does something for you and then returns data. You don’t really know the internals of the service, you just let it do its thing and then it comes back to you. It has defined inputs and outputs so that you know exactly what you need to give it and then what you get back.”

“I don’t have to worry about configuring that service - based on contracts you do what you need to do and you shouldn’t have to tell it how to do its job. It’ll just do it and come back to you.”

Users expect services to perform a **defined business function**

Services handle multiple responsibilities to achieve a specific business goal

"A service is a collection of functions with a very defined scope. So a cart service better not be doing transactions. A transaction service should be its own thing."

"Services execute a collection of tasks/functions within a defined scope."

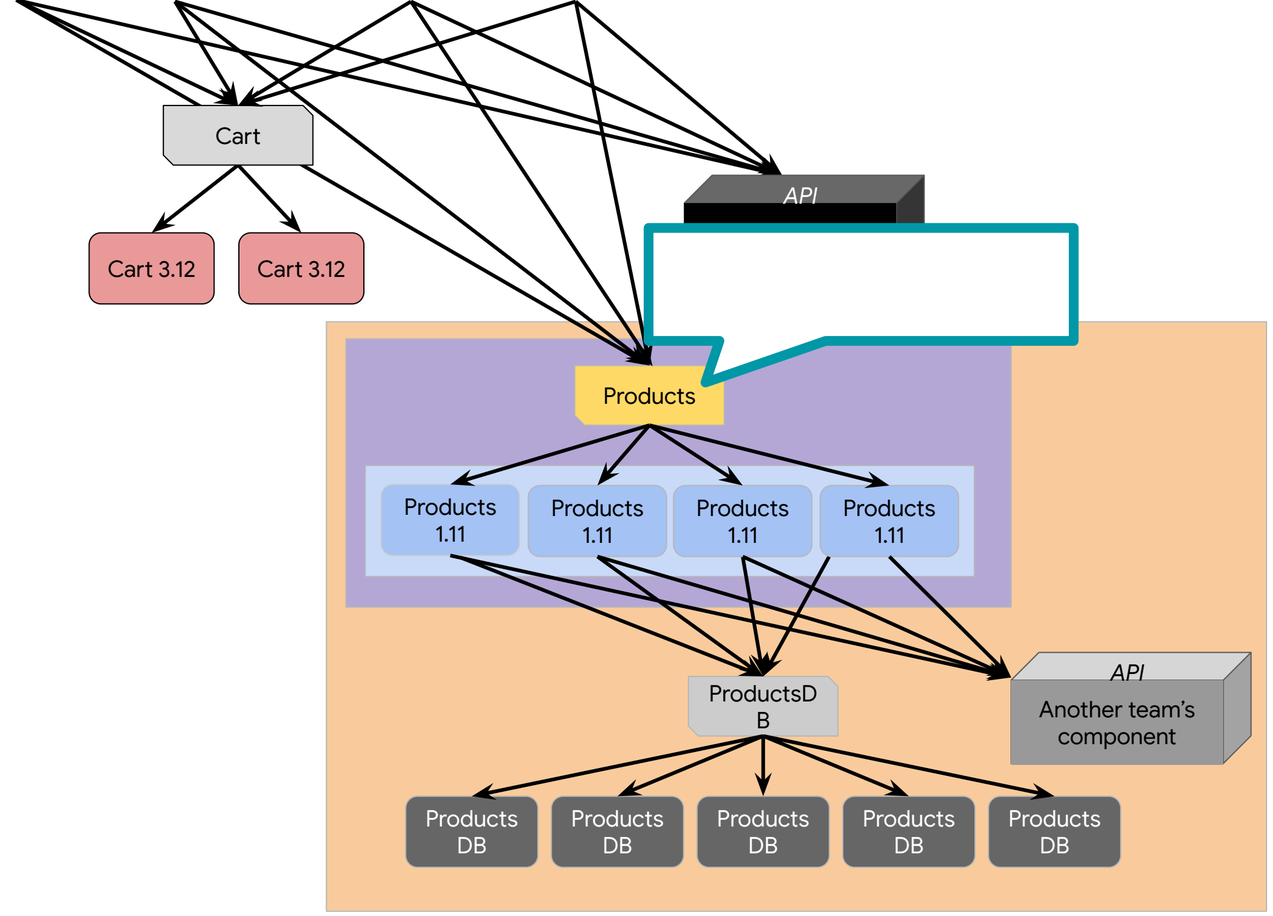
# Users' understanding of terminology is **highly variable**

Users' preferred nomenclature depends on their team, product experience, and cloud provider

When I need clarity I try to avoid using the term service but if I'm just speaking generally I will use it.

"It depends on who you're talking to. You create your own language for something within your team. If you own it and you call it product service, I'll call it that too."

# User perceptions **challenge the Kubernetes use** of “service”




Participants’ use of “service” aligns more with the Knative concept of “service,” while the Kubernetes concept was consistently labeled a load balancer

# Detailed findings

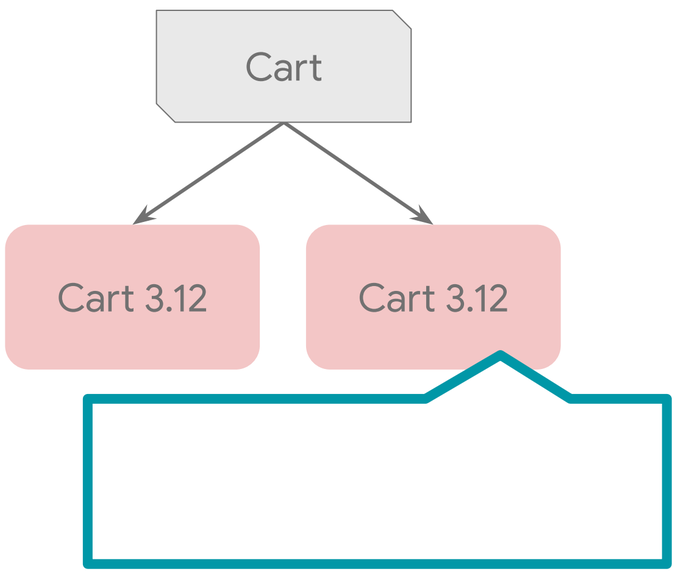
1

2

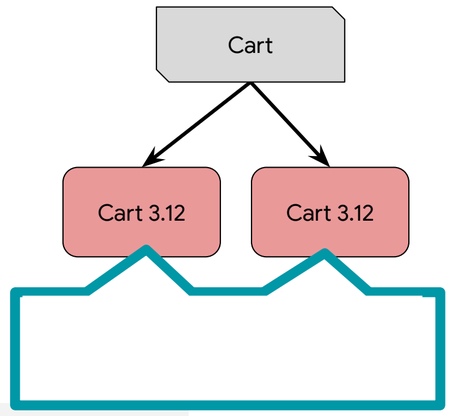
Diagram labelsWhat is a service?

1 2

# Diagram labels



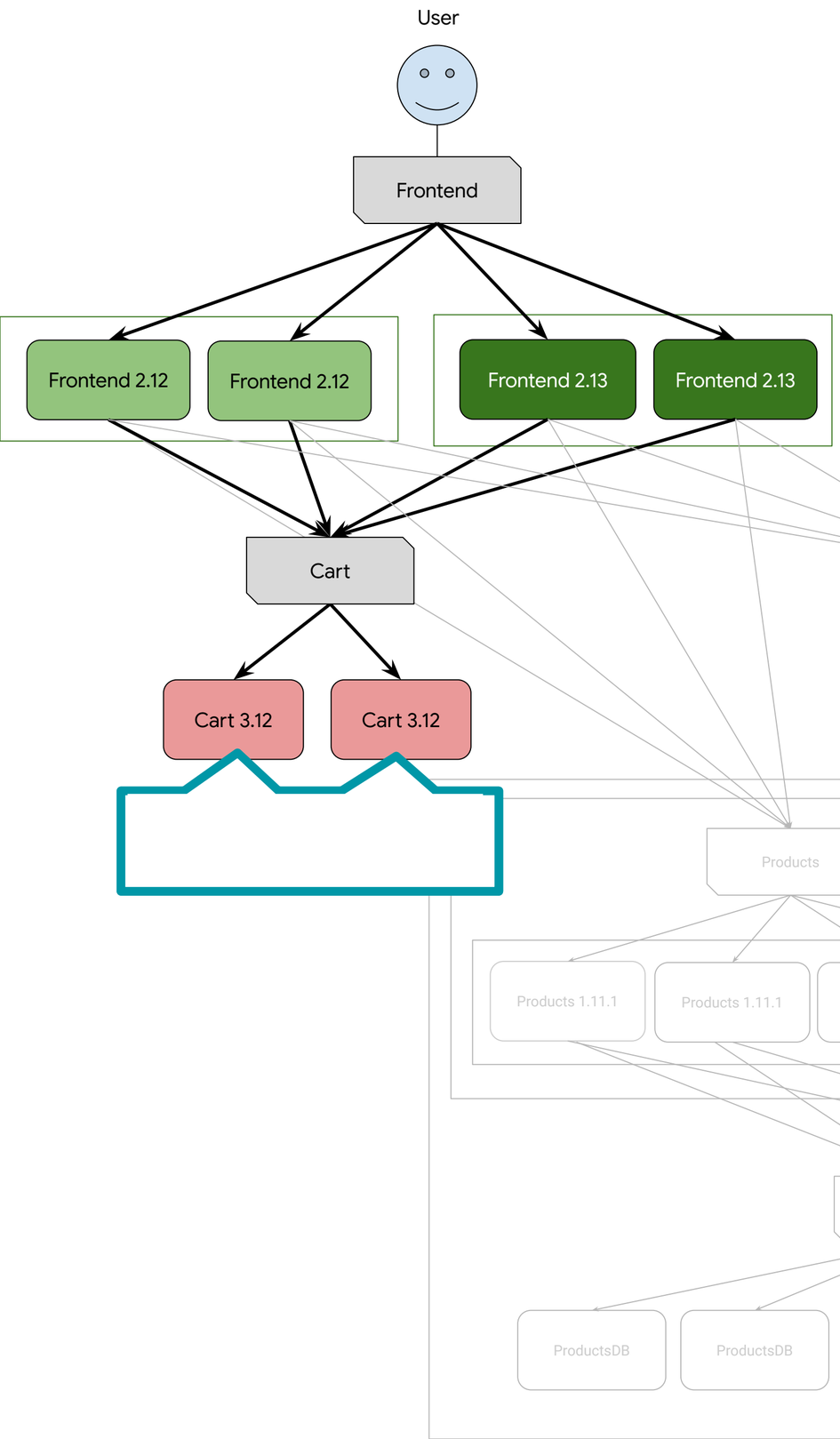
# Red cart box

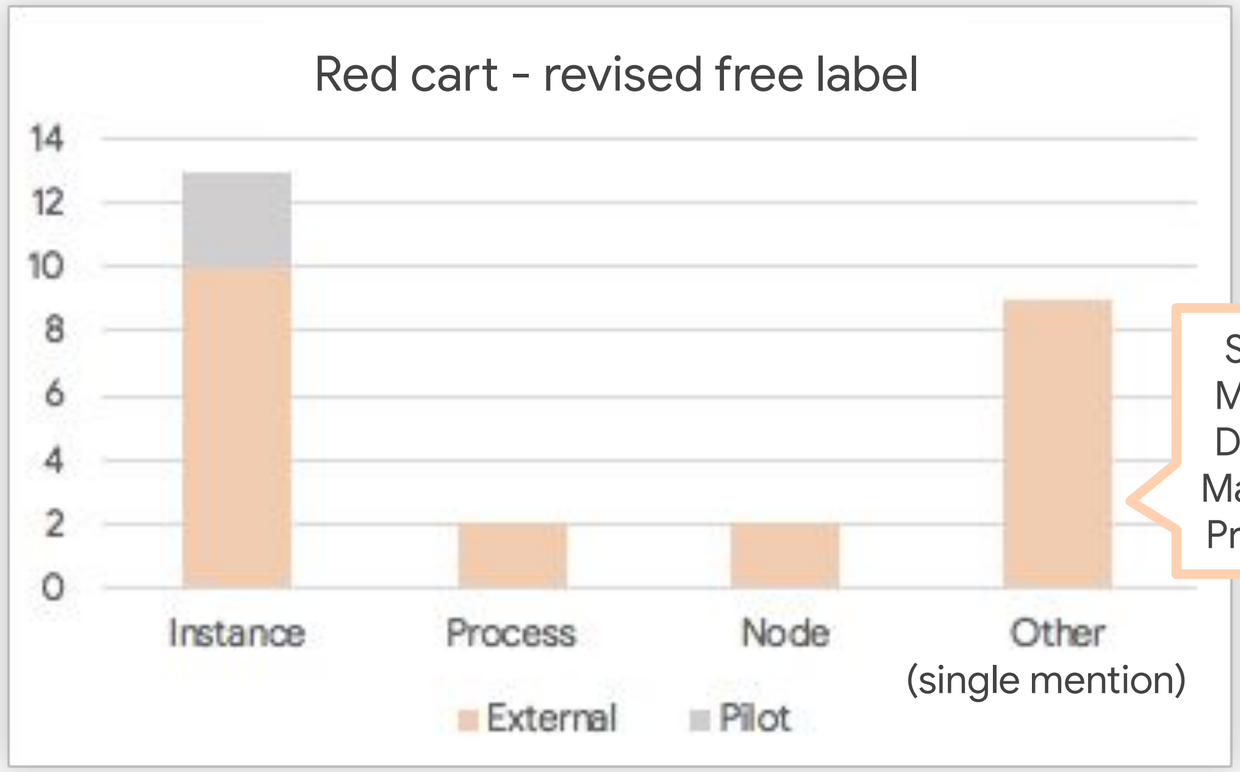


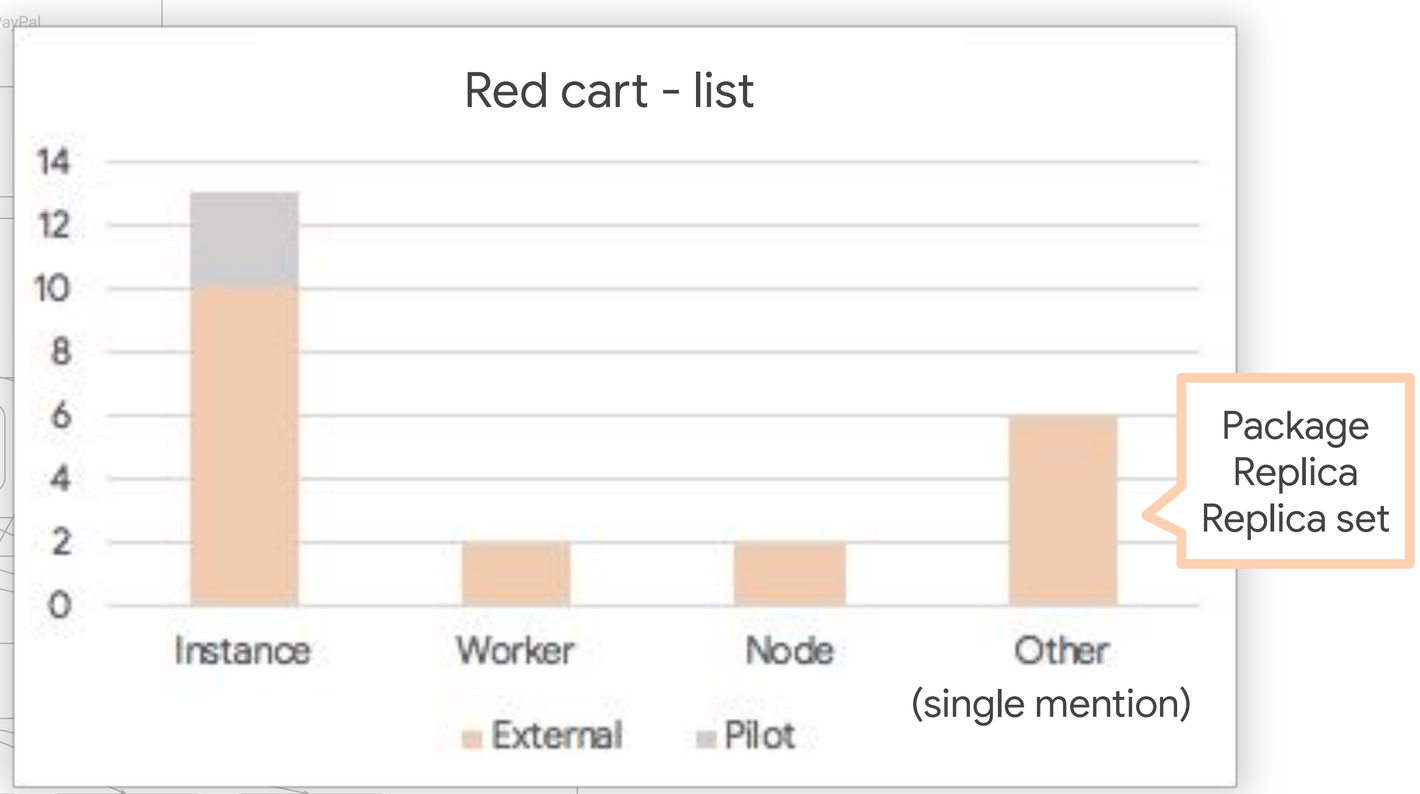

**Significance**: Allows us to validate our Cloud Services Platform object model by understanding how users refer to one copy of the code they're running

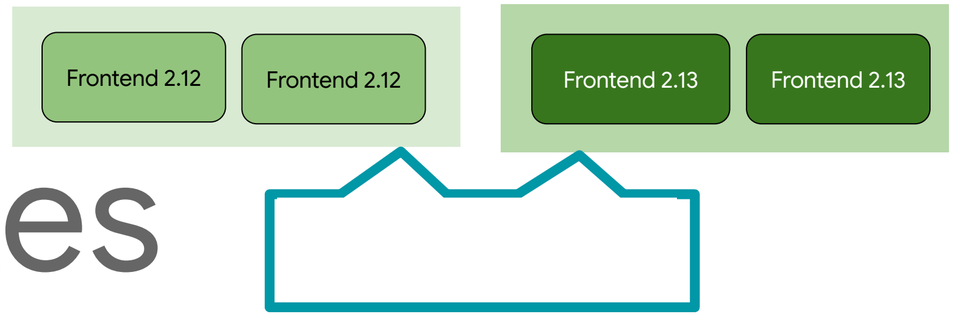
When representing a logical separation, instance was the most common term; shard or node may be a better term for a physical separation

This construct was likened to pods in Kubernetes


User
Frontend
Frontend 2.12
Frontend 2.12
Frontend 2.13
Frontend 2.13
Cart
Cart 3.12
Cart 3.12
PayPal
Products
Products 1.11.1
Products 1.11.1
Products 1.11.1
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB


Red cart - revised free label
14
12
10
8
6
4
2
0
Instance
Process
Node
Other
(single mention)
External
Pilot
Shard
Master
Deploy
Machine
Process


Red cart - list
14
12
10
8
6
4
2
0
Instance
Worker
Node
Other
(single mention)
External
Pilot
Package
Replica
Replica set




# Green bounding boxes

**Significance**: Releasing a new version of software is a common CUJ, so we need to know what to call these different groups in the UI

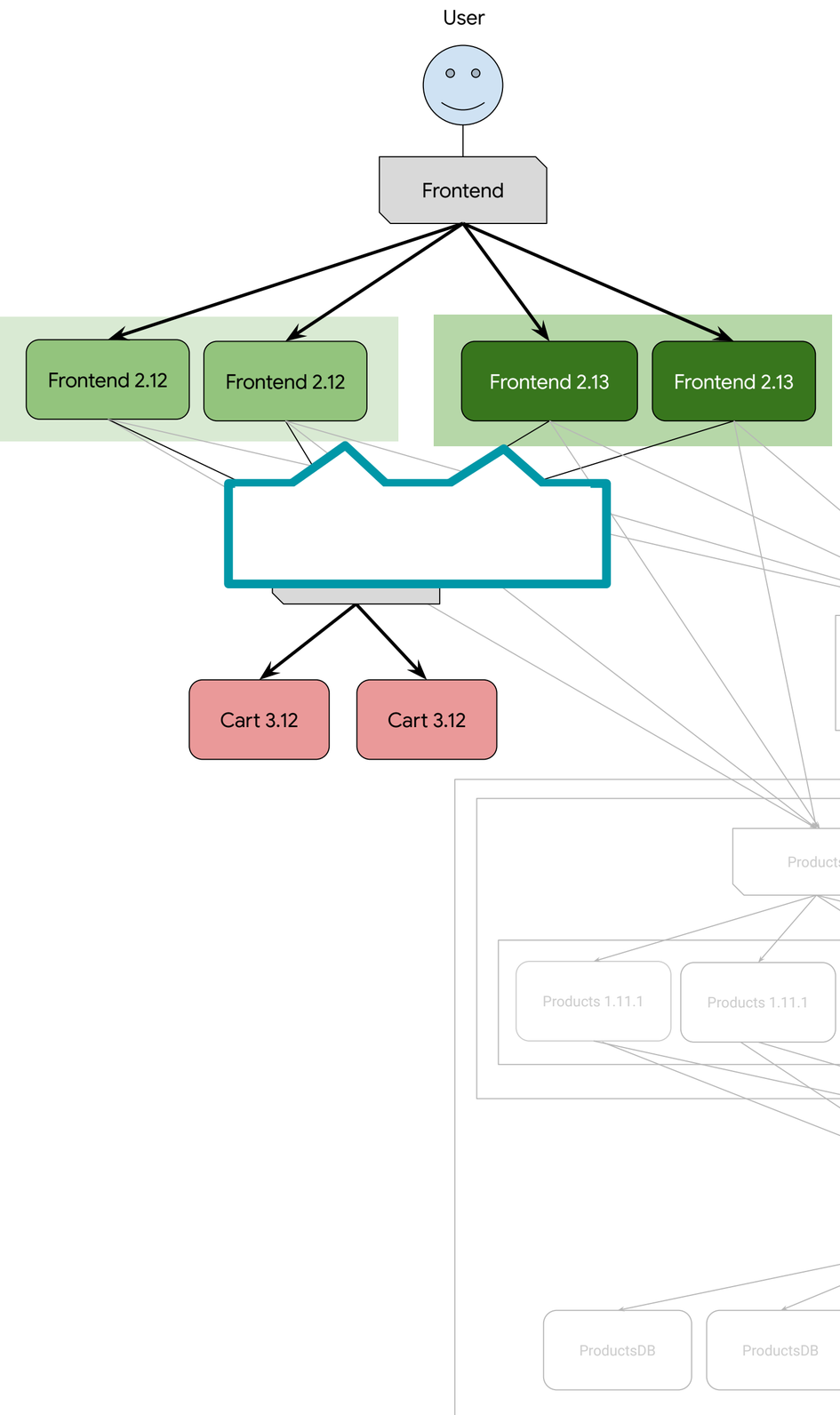
Several terms were used interchangeably, but the most precise word depends on the magnitude of the change between 2.12 and 2.13

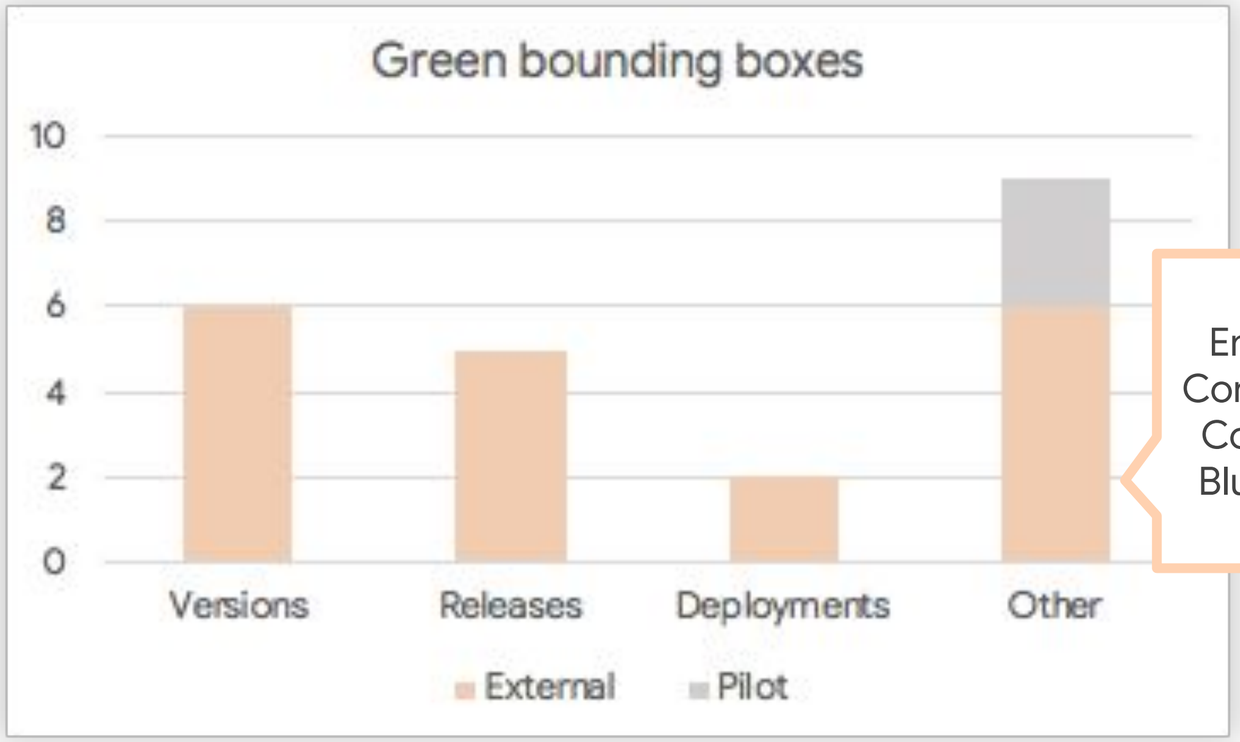
- From smallest to largest, the pattern seems to be: patches/hotfix < versions < releases
- Release implies more mature code that is ready for end-users

These could also be referred to as blue/green or A/B if there are only two; version and release are more flexible terms

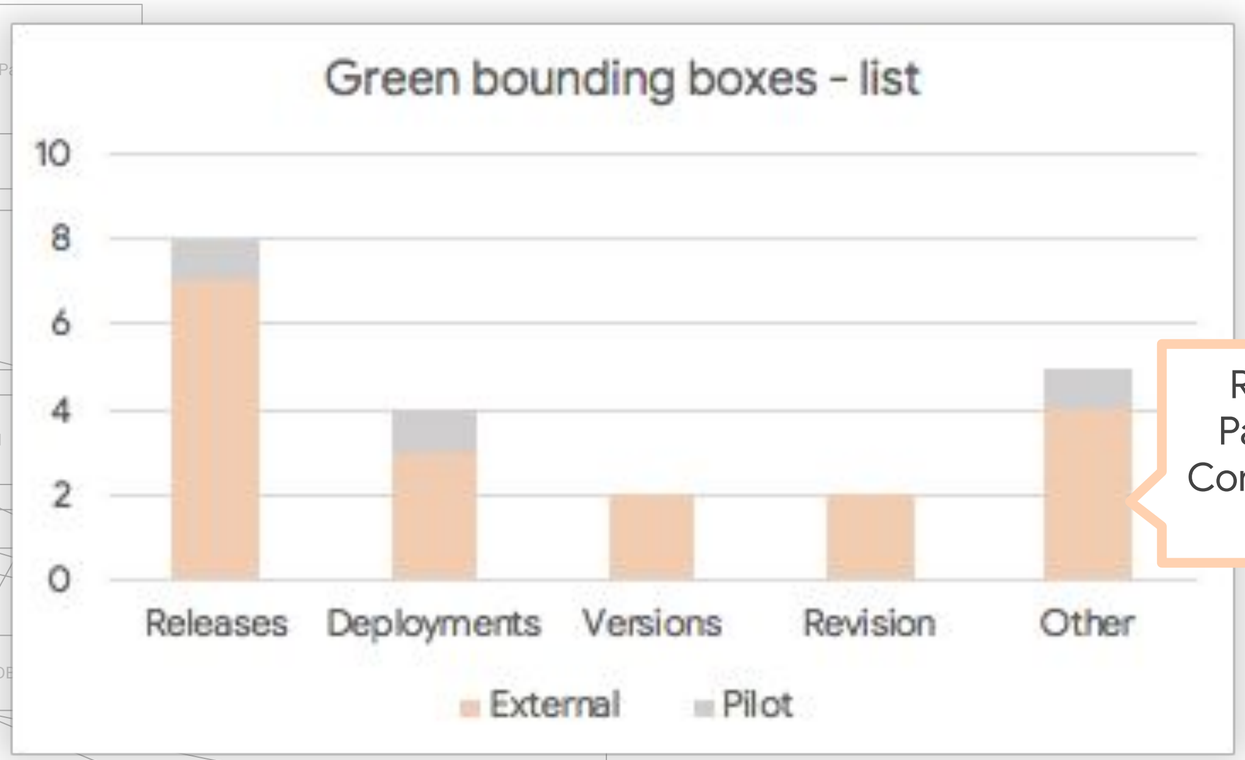
Some participants described these as ‘prod’/’live’ and ‘canary’ deployments

‘Build’ was also considered, but was determined to be too broad since it could incorporate staging environments



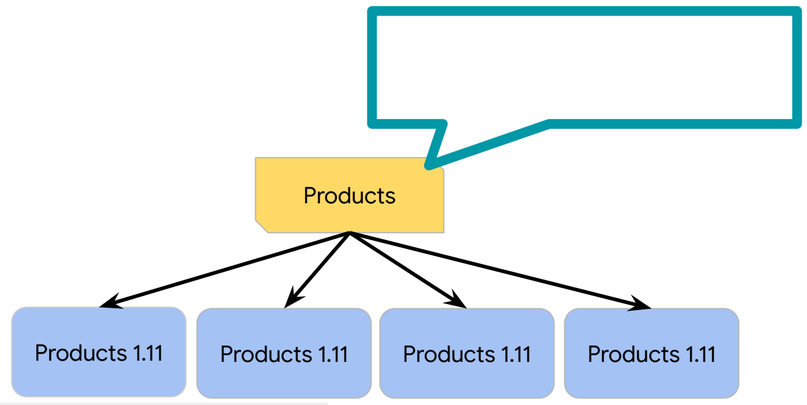




Sets
Endpoints
Components
Containers
Blue/Green
ASG




Replicas
Packages
Components
Builds

# Yellow products box



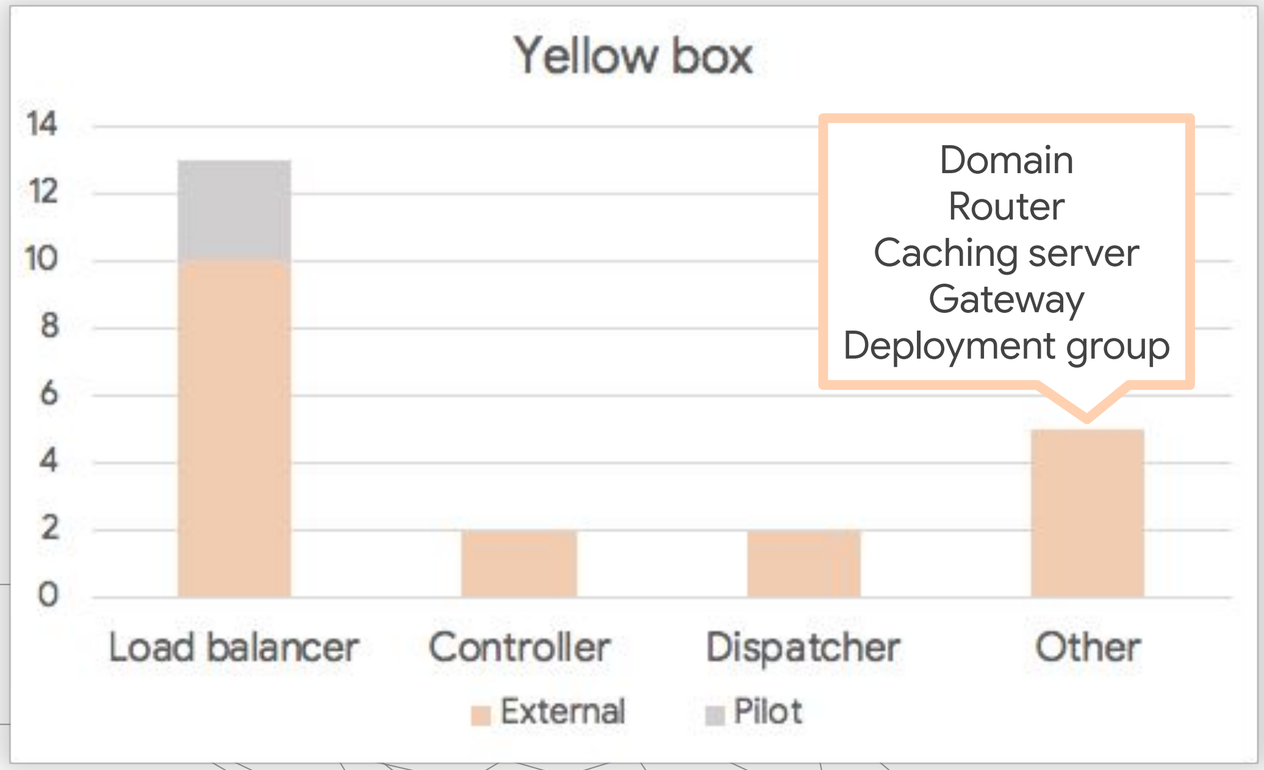

**Significance**: Represents a Kubernetes service, and a key distinction between Kubernetes and Knative ‘service’ concepts

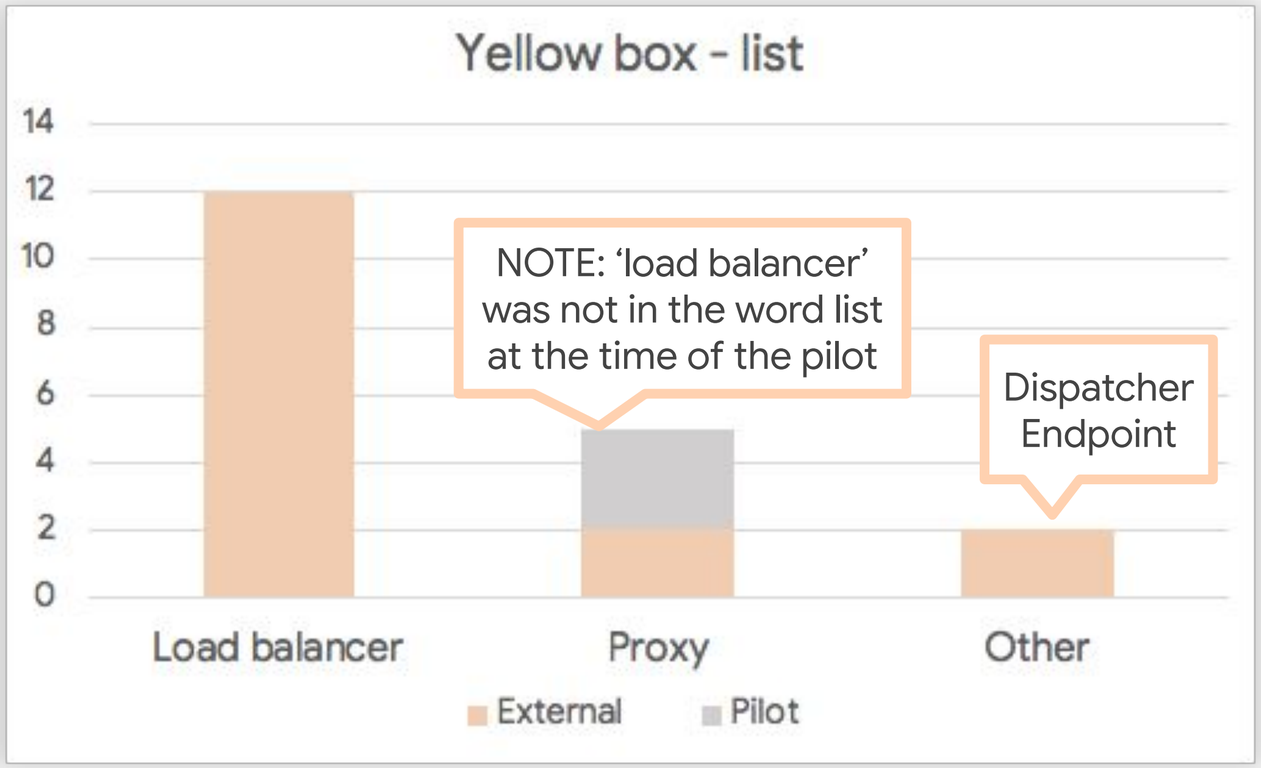
Participants associate the task of routing traffic with load balancers, not services - this means we need to be specific in UI contexts when Kubernetes services converge with Knative services

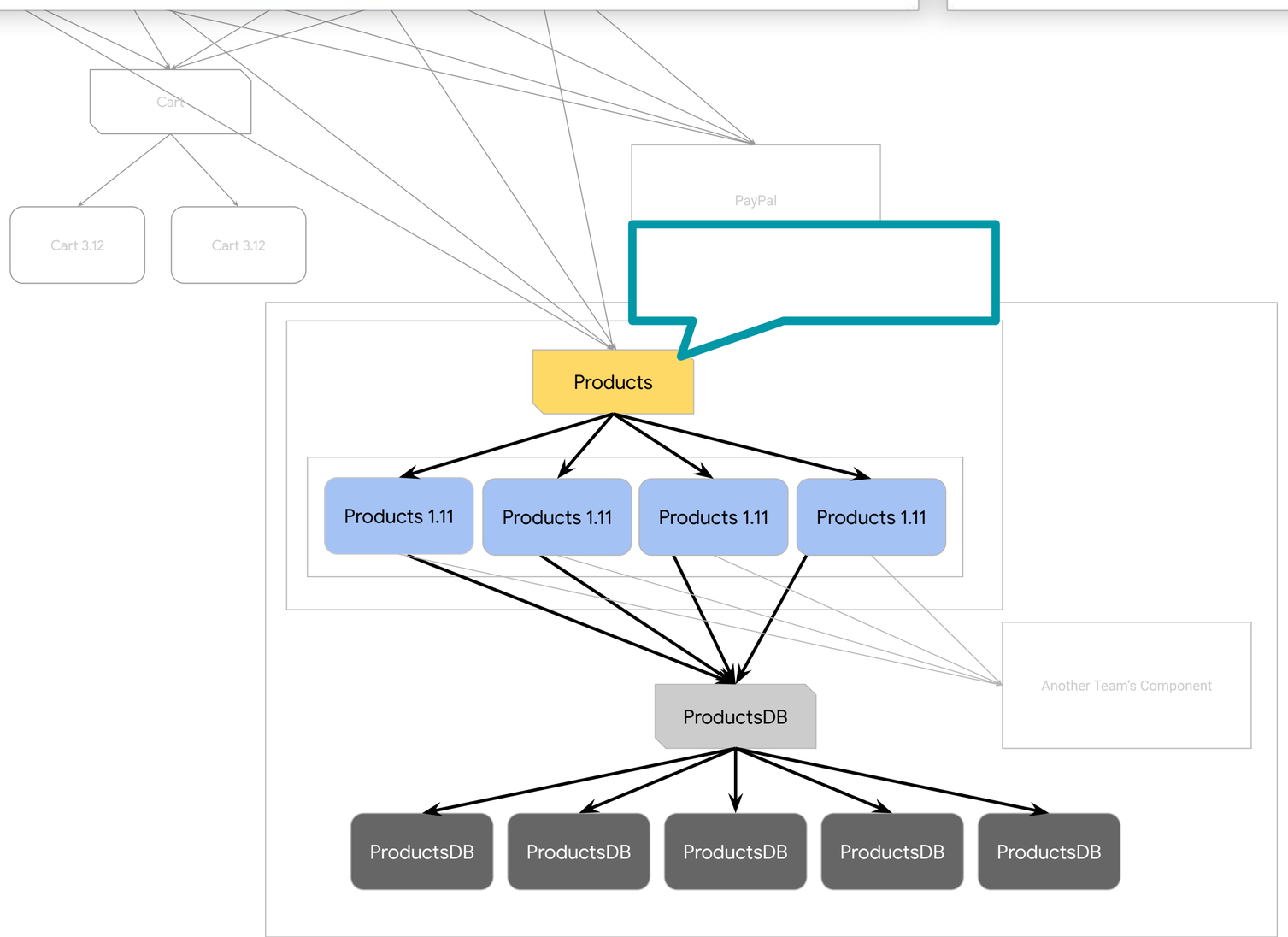
This concept was also named ‘load balancer’ by Kubernetes users in a follow-up study

Load balancer was considered more precise than a term like ‘dispatcher’ because traffic goes to multiple copies of the same thing; ‘gateway’ was considered but has security connotations that may not apply to this example

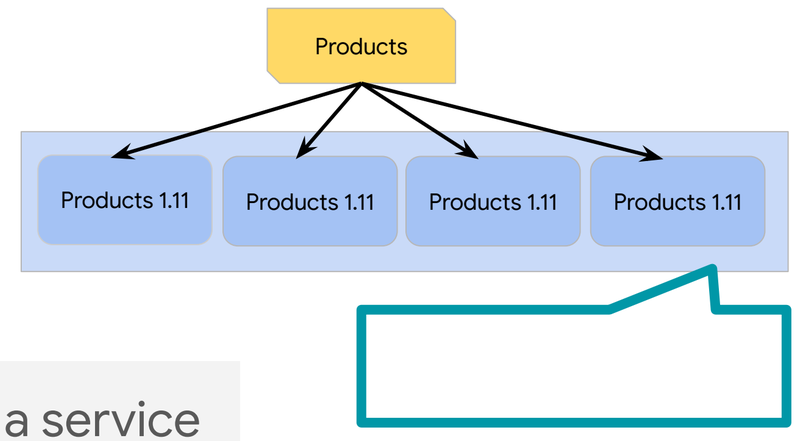
> “A service is not database, it’s not the web server... I suppose it’s not a load balancer either. I wouldn’t consider a load balancer a service.”


Yellow box
14
12
10
8
6
4
2
0
Load balancer
Controller
Dispatcher
Other
External
Pilot
Domain
Router
Caching server
Gateway
Deployment group


Yellow box - list
14
12
10
8
6
4
2
0
Load balancer
Proxy
Other
External
Pilot
NOTE: ‘load balancer’ was not in the word list at the time of the pilot
Dispatcher
Endpoint


Cart
Cart 3.12
Cart 3.12
PayPal
Products
Products 1.11
Products 1.11
Products 1.11
Products 1.11
Another Team’s Component
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB

# Blue bounding box



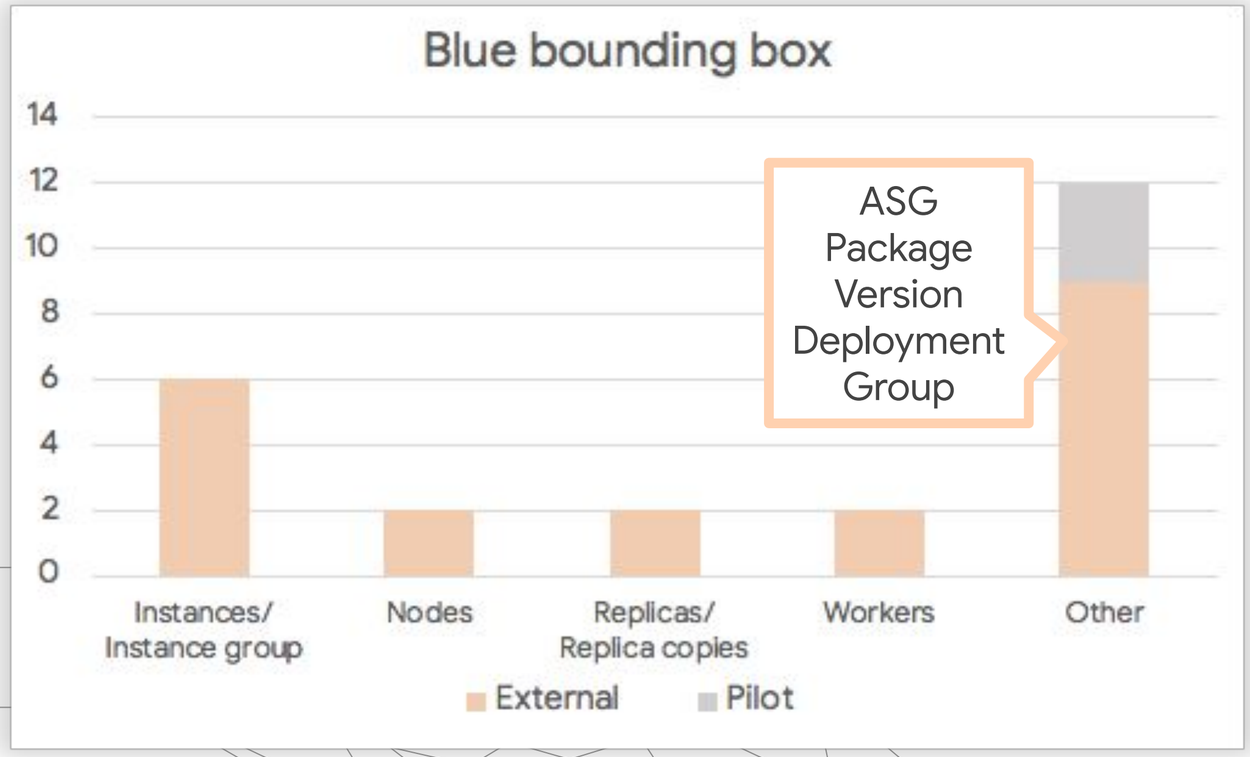

**Significance**: Helps us understand how to term the workloads behind a service

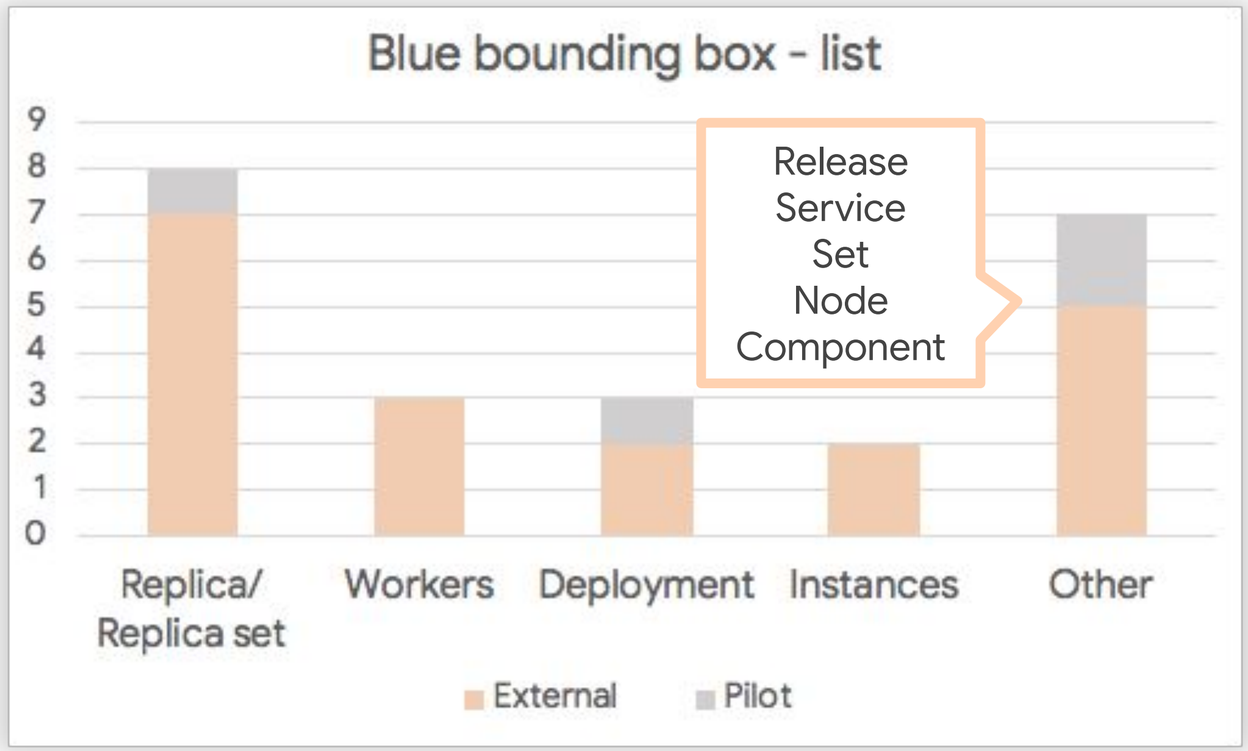
Participants generally did not have a name for this concept

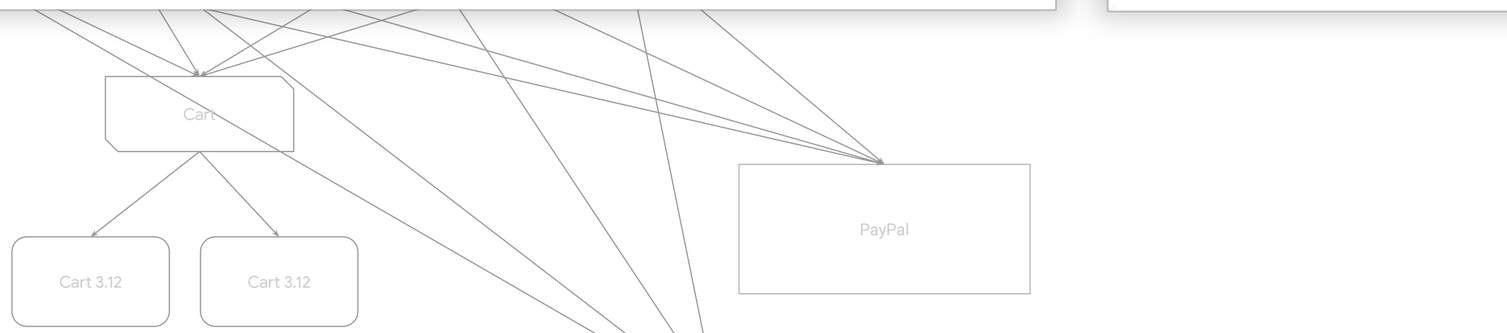
Replica set makes sense because it houses replicas of the same ‘products 1.11’ copies, especially to Kubernetes users - though a couple participants noted that replica has “database connotations” and they wouldn’t actually use it for this “compute concept”

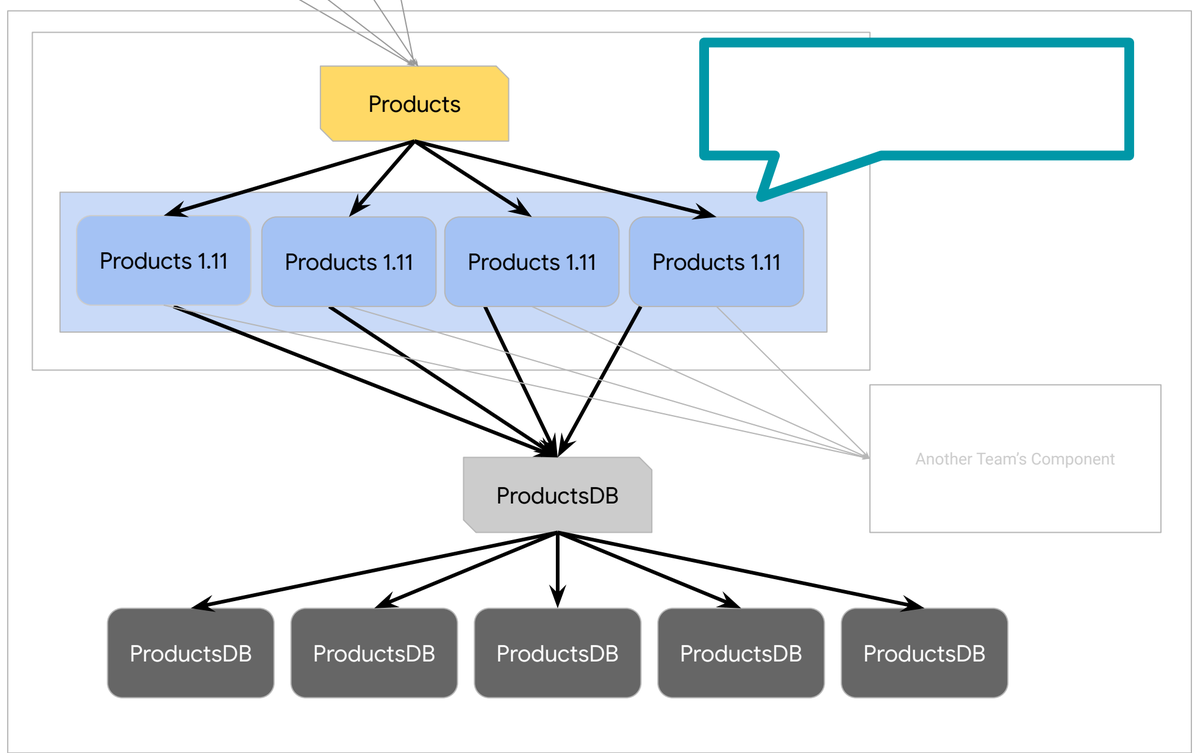
For the same reason, instance group works more generically since it is a group of instances

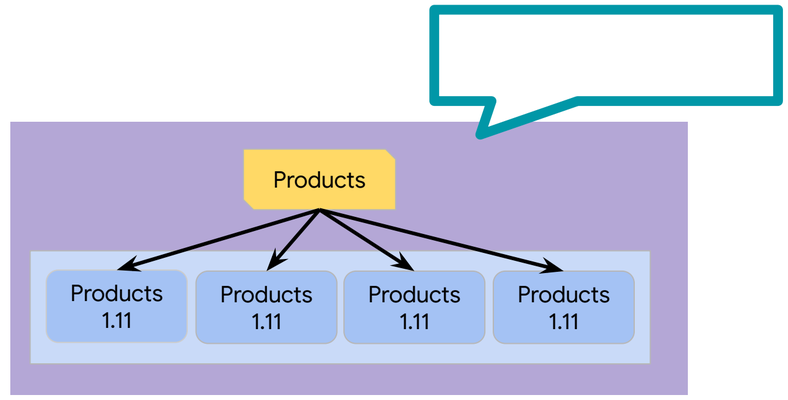
This instance group “is not a service because it doesn’t do anything for me without the load balancer”


Blue bounding box
14
12
10
8
6
4
2
0
ASG
Package
Version
Deployment
Group
Instances/
Instance group
Nodes
Replicas/
Replica copies
Workers
Other
External
Pilot


Blue bounding box - list
9
8
7
6
5
4
3
2
1
0
Release
Service
Set
Node
Component
Replica/
Replica set
Workers
Deployment
Instances
Other
External
Pilot


Cart
Cart 3.12
Cart 3.12
PayPal


Products
Products 1.11
Products 1.11
Products 1.11
Products 1.11
Another Team's Component
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB



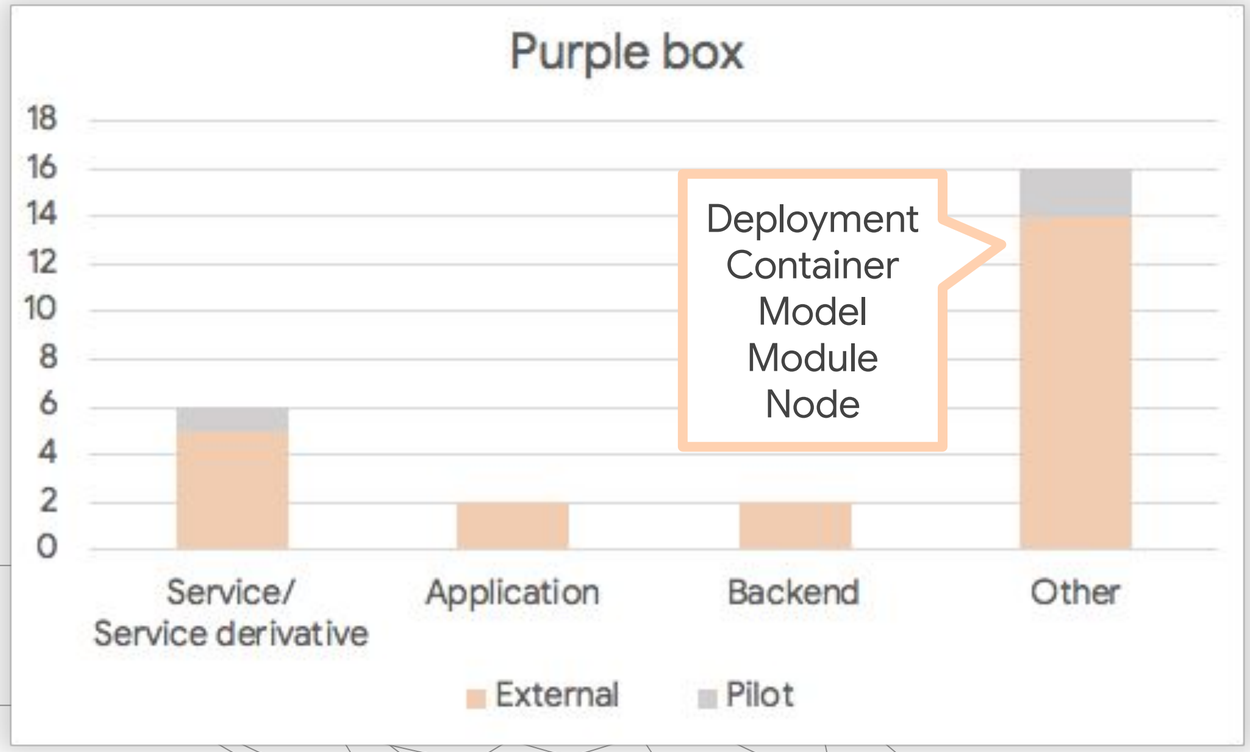

# Purple bounding box

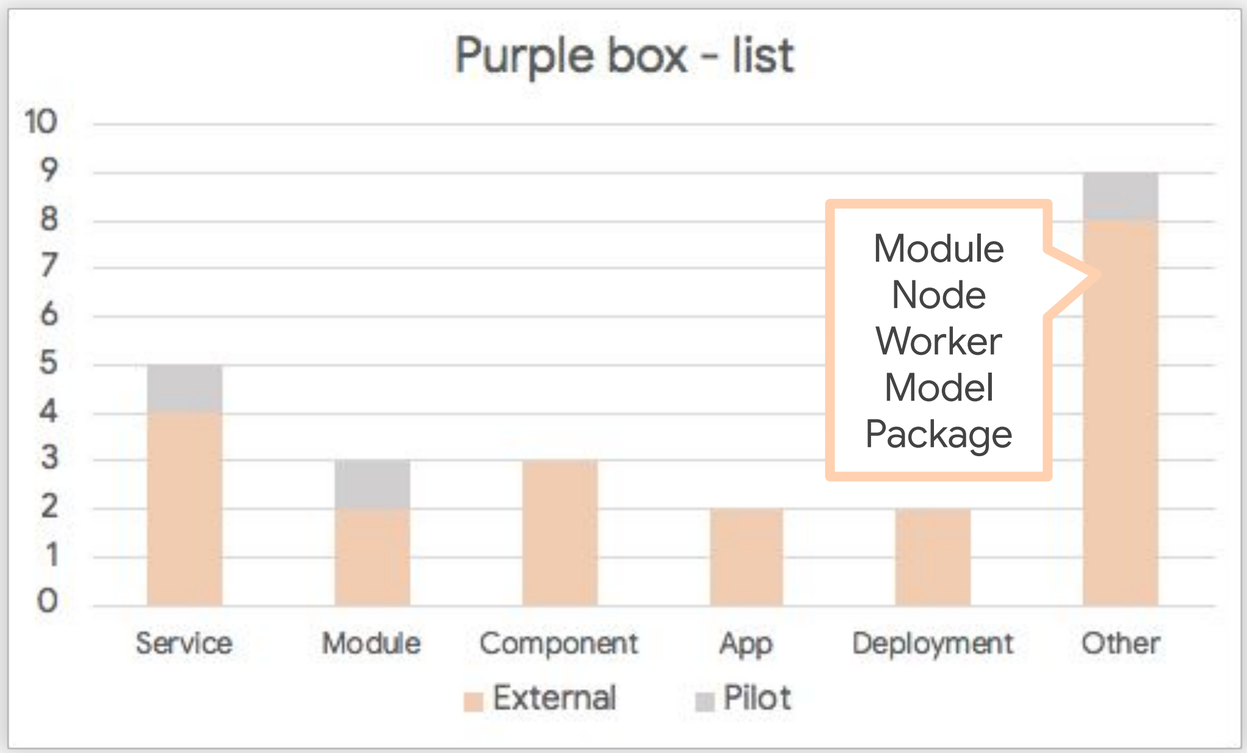
**Significance**: Represents a Knative service, and a key conceptual difference between Knative and Kubernetes, referencing the work of a single service operator/developer

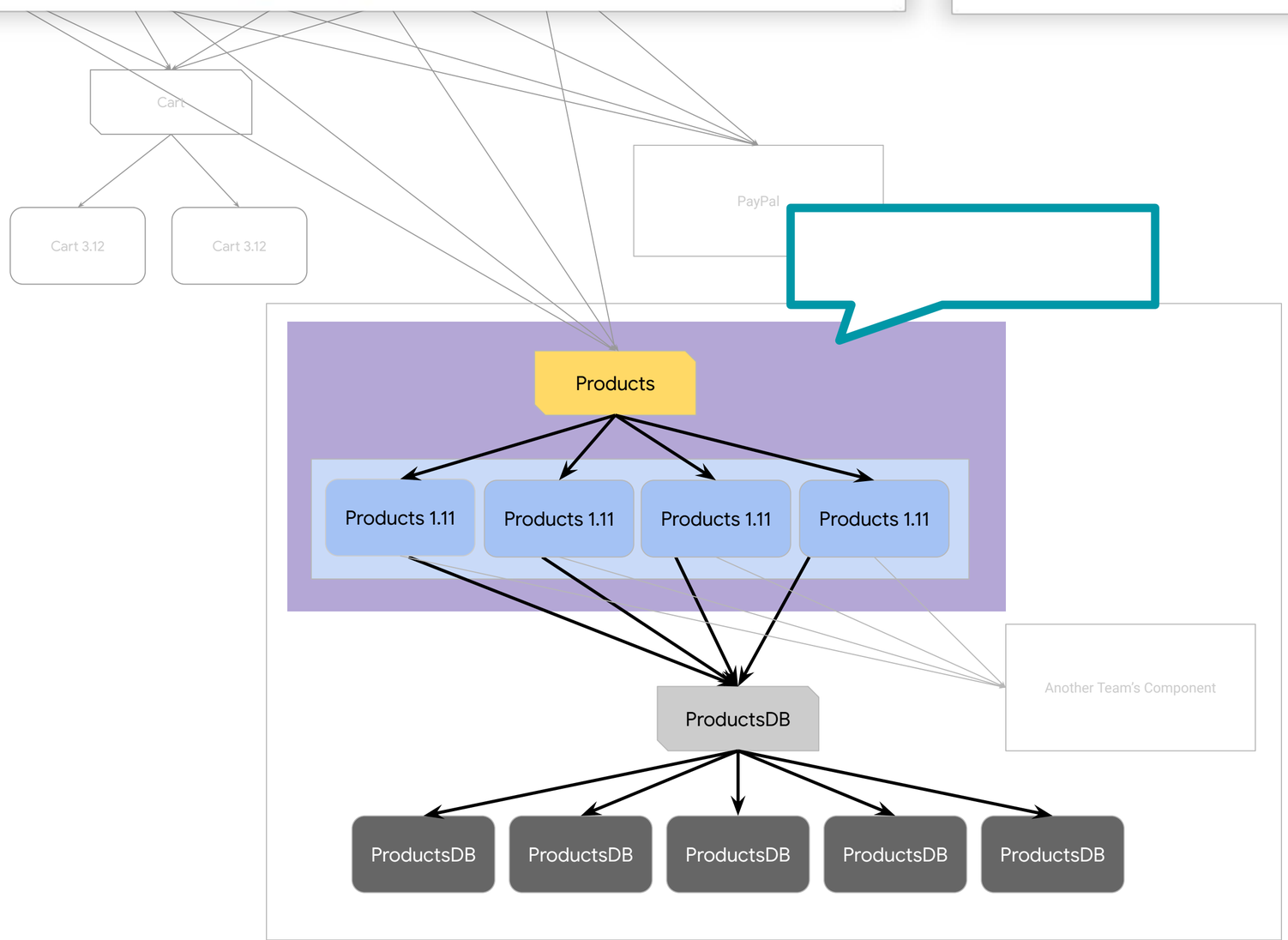
Participants narrowly chose ‘service’ more than other terms, but it was not top-of-mind; however, this concept maps well to participants’ definitions of what services include - a load balancer and instance group

For some, the presence of the load balancer distinguished it from terms like ‘nodes’ and made it a ‘complete service’

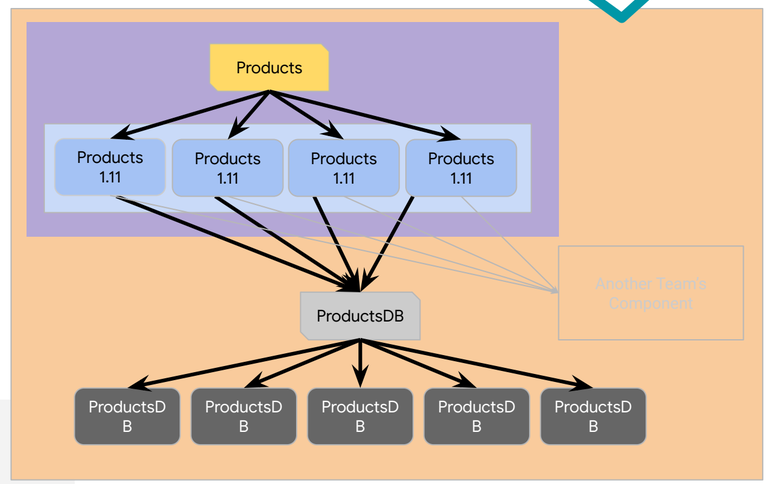
> “I called it a service because it includes the load balancer and instances, making it a complete service.”


Purple box
18
16
14
12
10
8
6
4
2
0
Deployment
Container
Model
Module
Node
Service/
Service derivative
Application
Backend
Other
External
Pilot


Purple box - list
10
9
8
7
6
5
4
3
2
1
0
Module
Node
Worker
Model
Package
Service
Module
Component
App
Deployment
Other
External
Pilot


Cart
Cart 3.12
Cart 3.12
PayPal
Products
Products 1.11
Products 1.11
Products 1.11
Products 1.11
Another Team's Component
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB

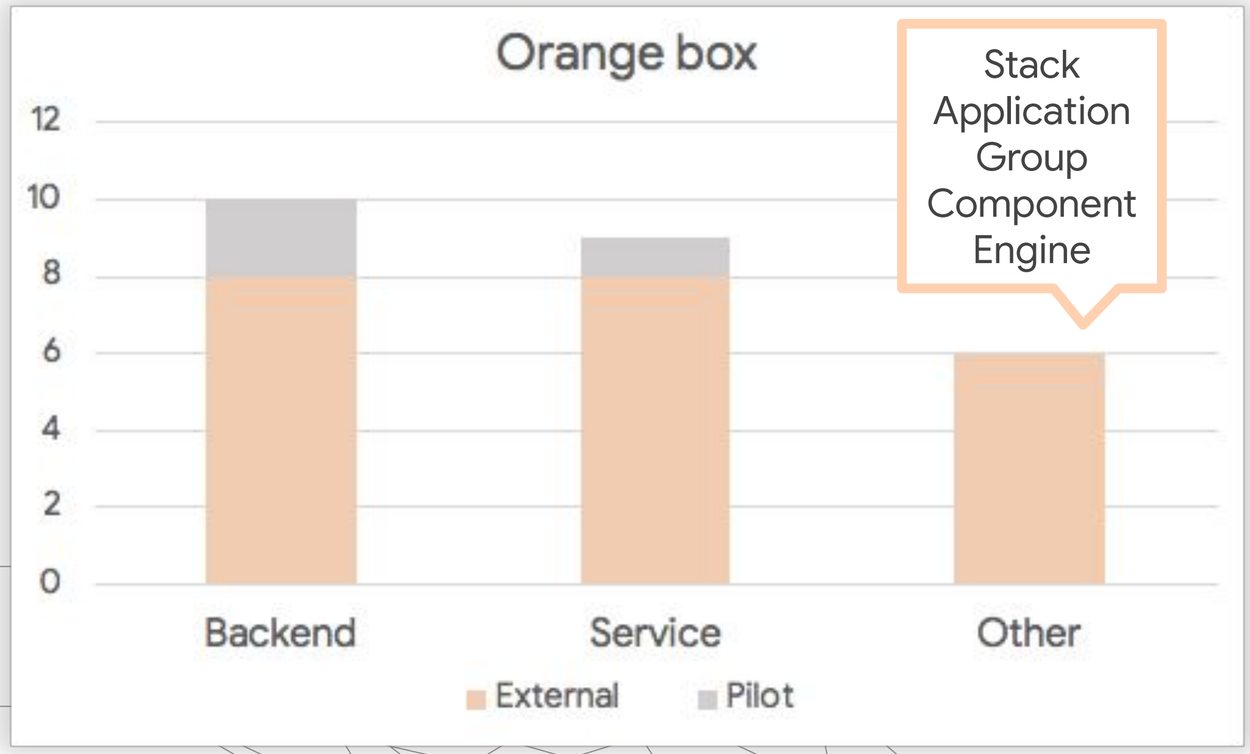
# Orange bounding box

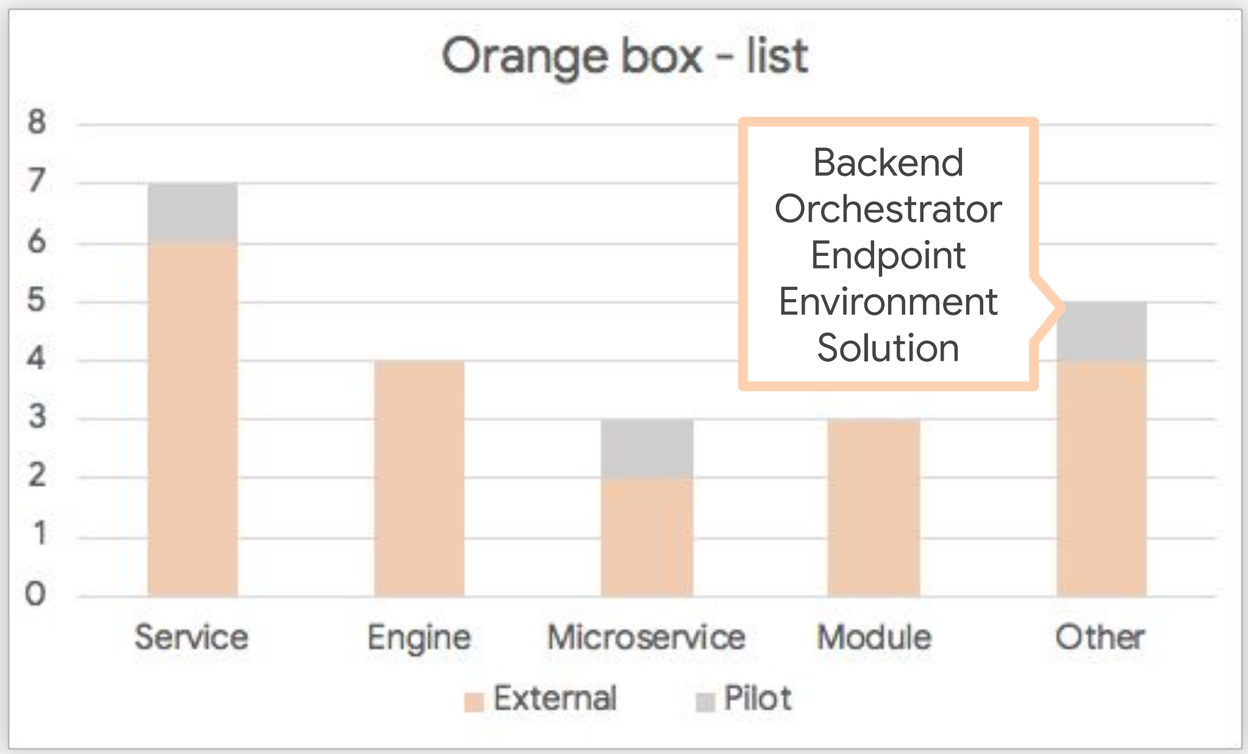


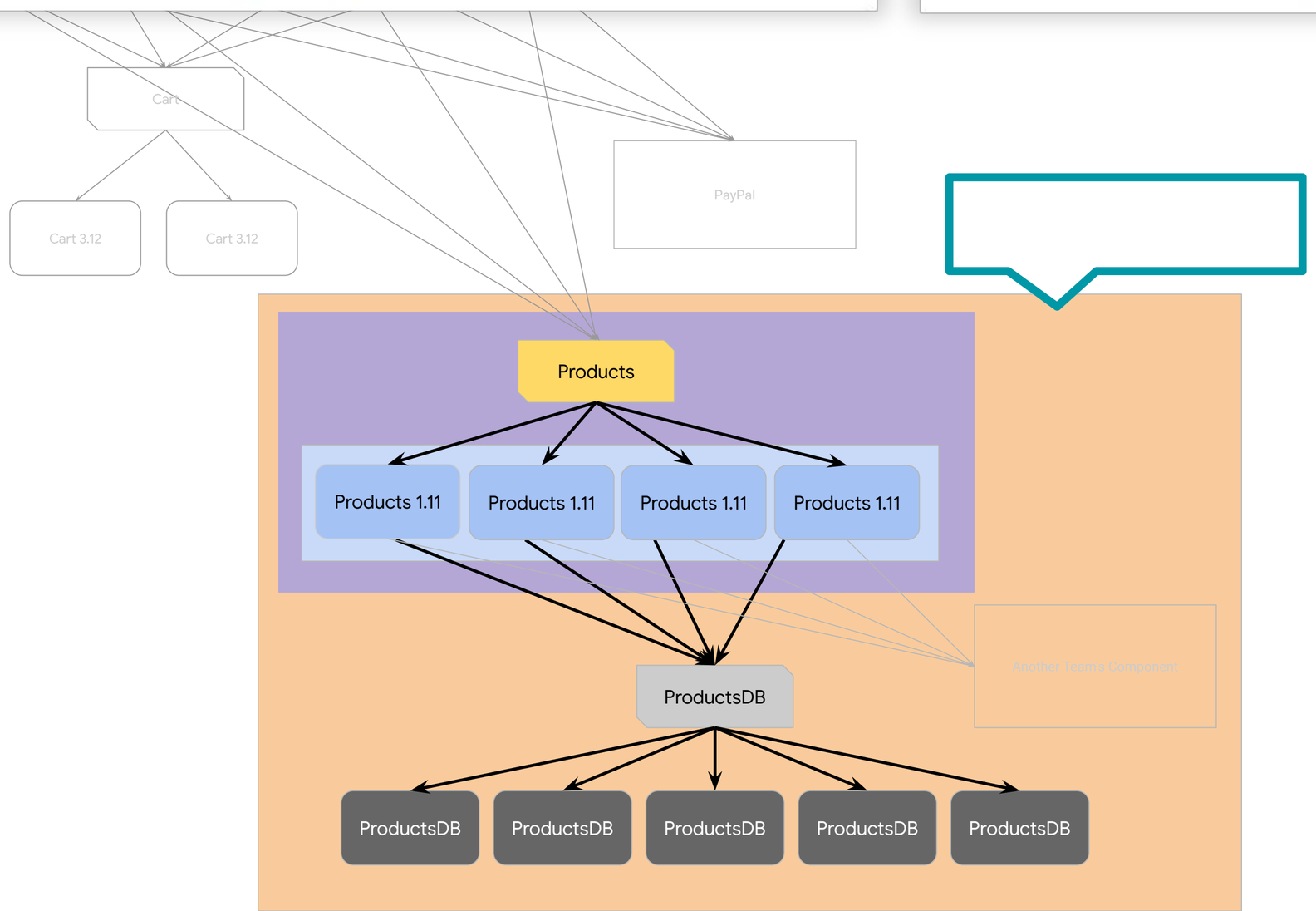

**Significance**: Represents a distinction from the purple box to understand whether users have a name for their [purple box] and its dependencies, and helps us distinguish the producer view from the consumer view of a service

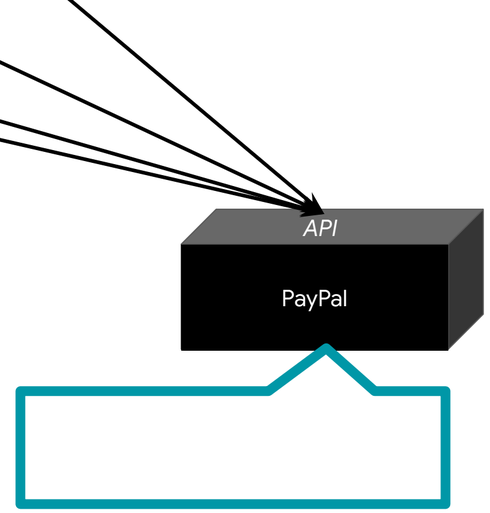
The orange box was interpreted as the backend because it includes the databases and everything the ‘products’ items rely on

Others referred to this concept as a service, likening it to the PayPal API from the perspective of the frontend


Orange box
12
10
8
6
4
2
0
Backend
Service
Other
External
Pilot
Stack
Application
Group
Component
Engine


Orange box - list
8
7
6
5
4
3
2
1
0
Service
Engine
Microservice
Module
Other
External
Pilot
Backend
Orchestrator
Endpoint
Environment
Solution


Cart
Cart 3.12
Cart 3.12
PayPal
Products
Products 1.11
Products 1.11
Products 1.11
Products 1.11
Another Team's Component
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB




# Black PayPal API

**Significance**: Understand how users refer to ‘black-box’ concepts they rely on, but are outside their organization

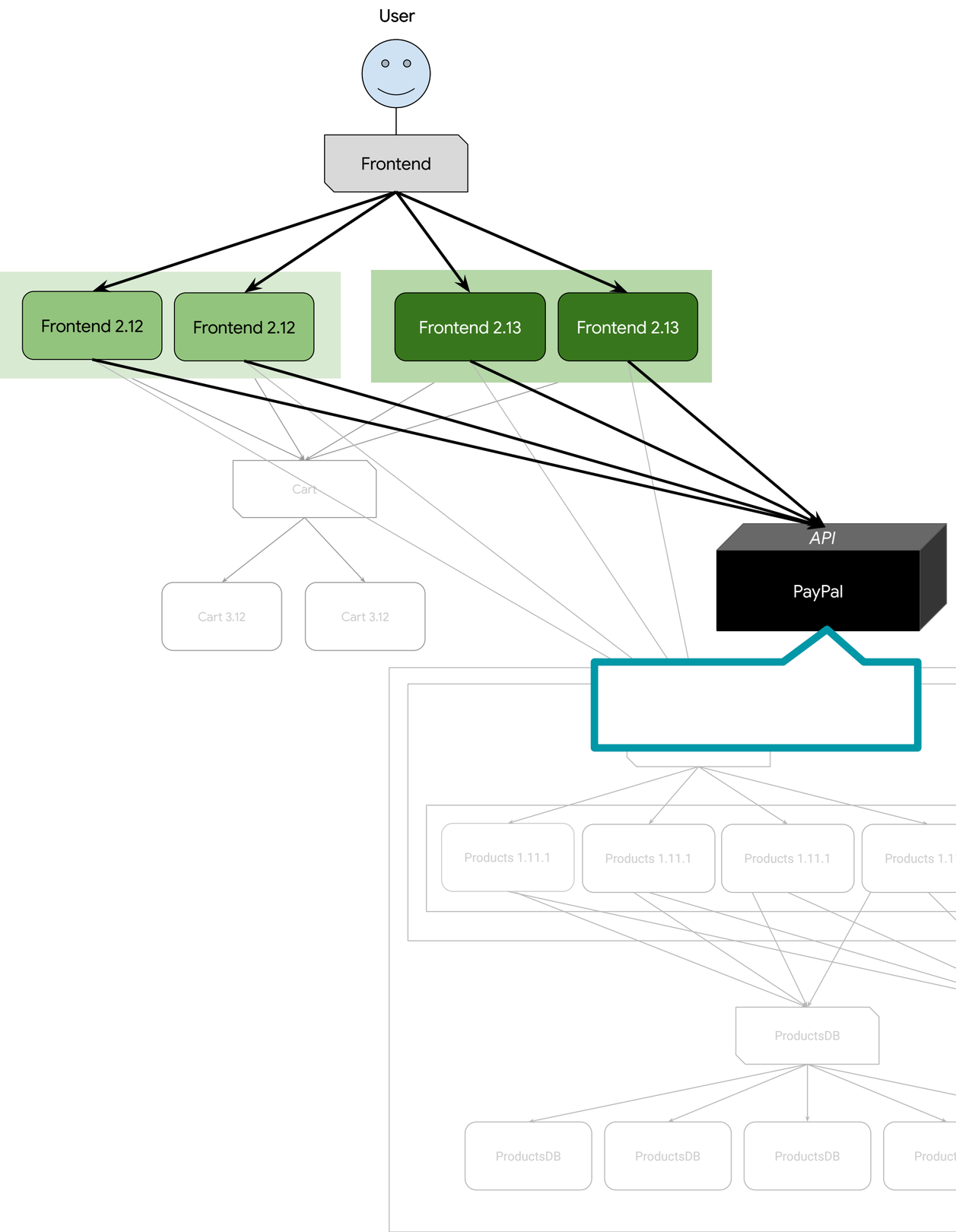
Although participants were encouraged to use a different term when naming this item, ‘API’ was their strongest inclination

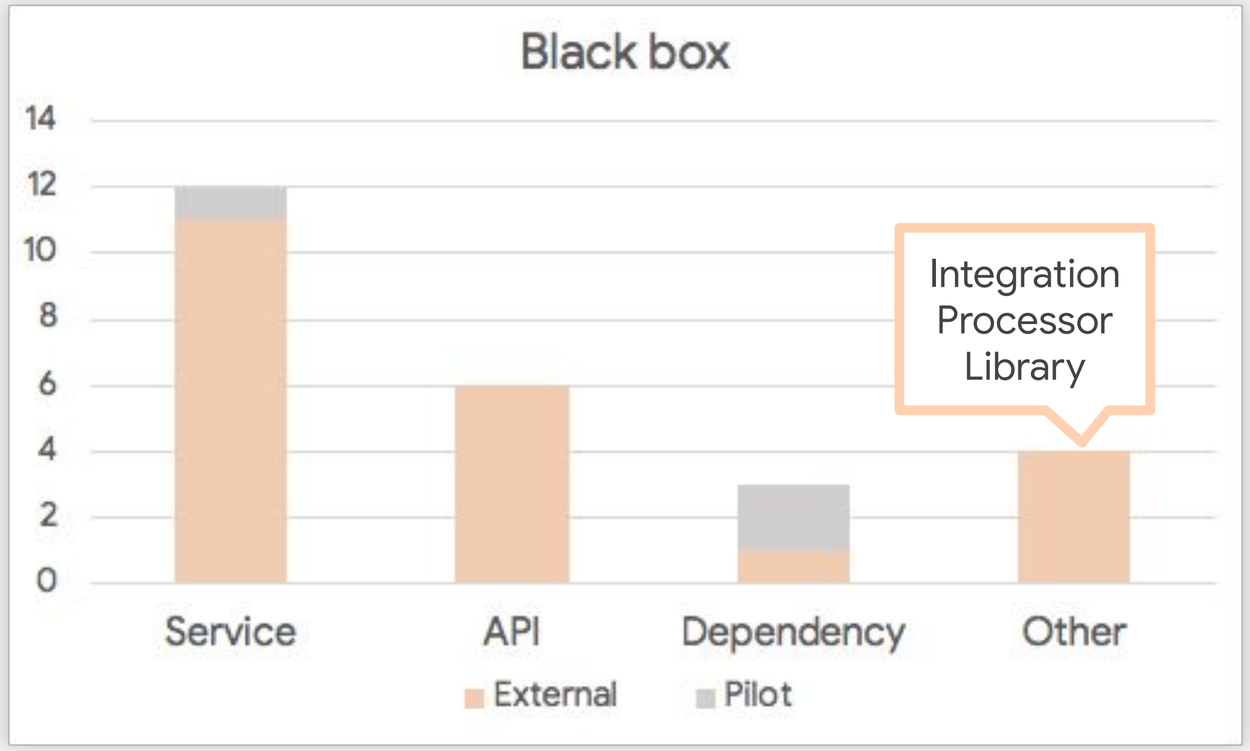
Qualifiers like ‘external,’ ‘vendor,’ ‘third party,’ or ‘partner’ were common

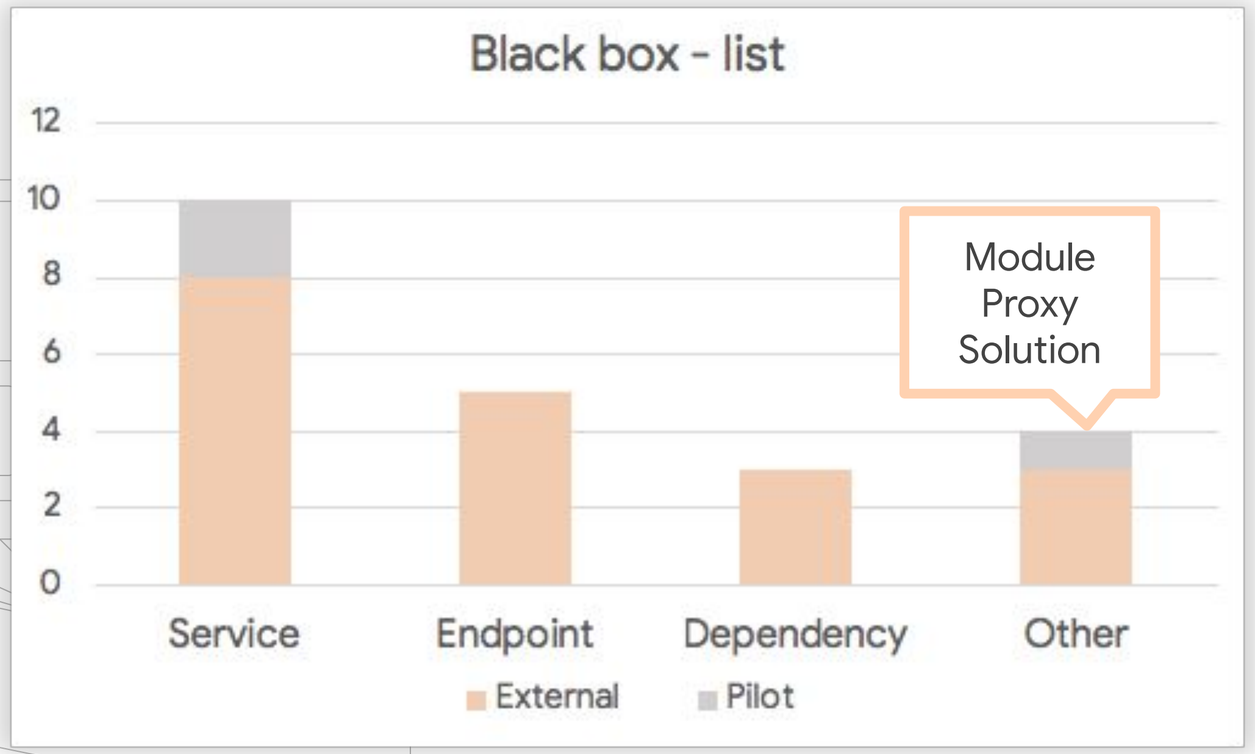
‘Service’ was chosen here because the PayPal API:

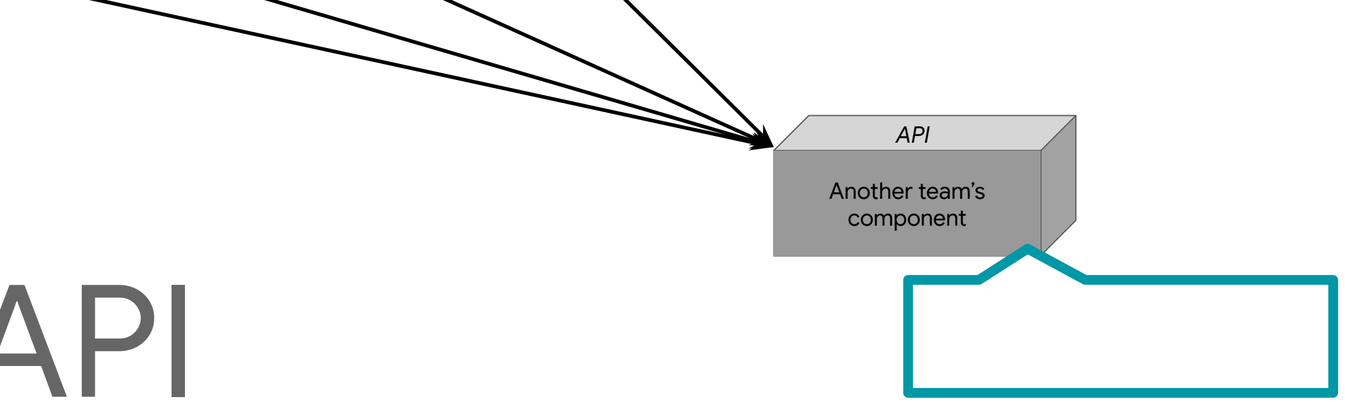
- Is opaque
- Provides a specific business value
- Provides information via HTTP request (distinction from ‘library’ or ‘dependency’)

“We’re not thinking about how they support their system. We just think about the API we use”


User
Frontend
Frontend 2.12
Frontend 2.12
Frontend 2.13
Frontend 2.13
Cart
Cart 3.12
Cart 3.12
API
PayPal
Products 1.11.1
Products 1.11.1
Products 1.11.1
Products 1.11.1
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB


Black box
14
12
10
8
6
4
2
0
Service
API
Dependency
Other
External
Pilot
Integration Processor Library


Black box - list
12
10
8
6
4
2
0
Service
Endpoint
Dependency
Other
External
Pilot
Module Proxy Solution




# Grey internal API

**Significance**: Understand whether a black-box concept would be named differently if it was internal to the organization

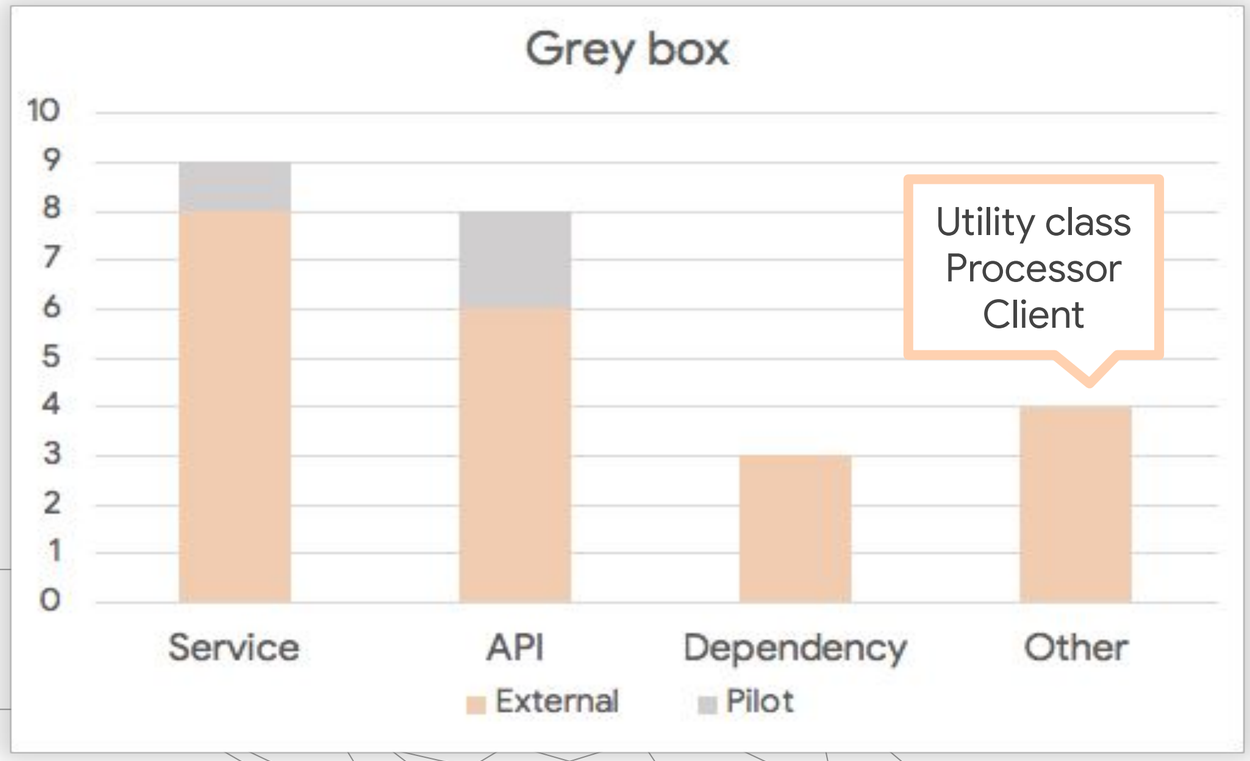
‘Internal’ was used to distinguish from a third-party/external service

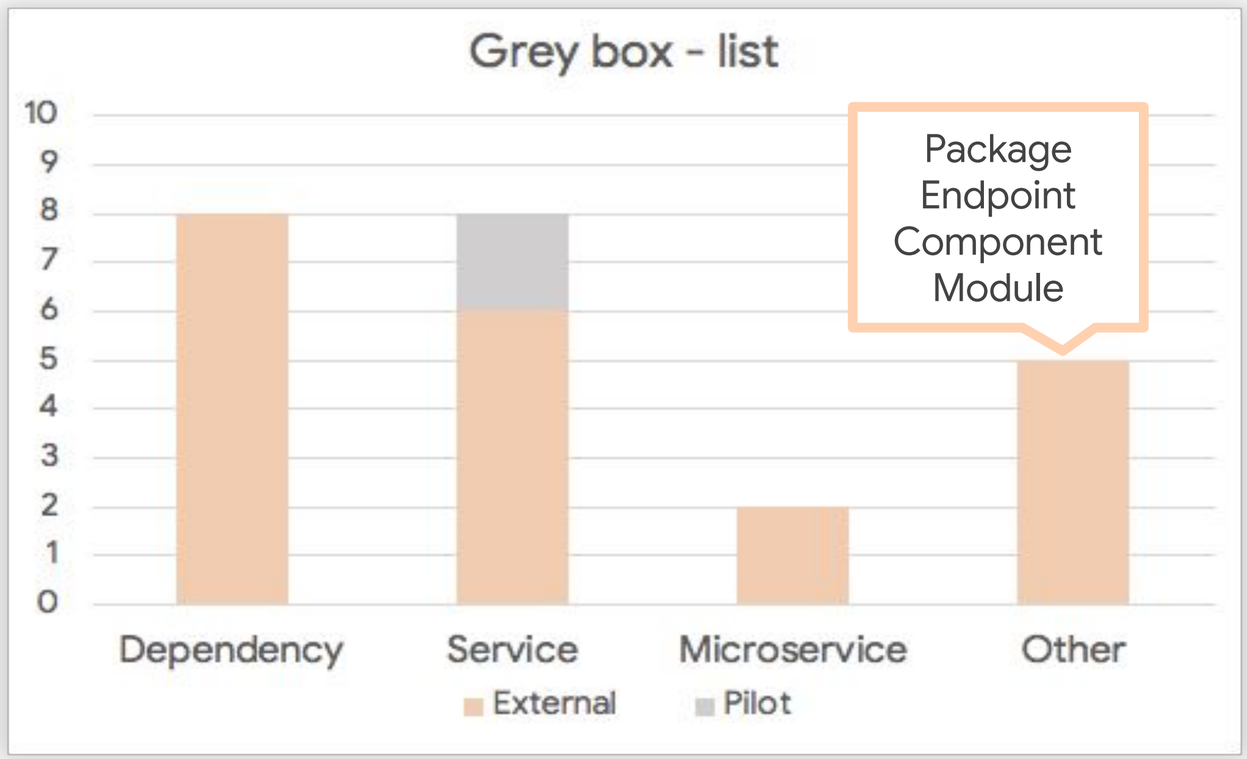
Some thought ‘external’ was still most appropriate for anything outside their team, which could be a source of confusion with the black-box PayPal concept

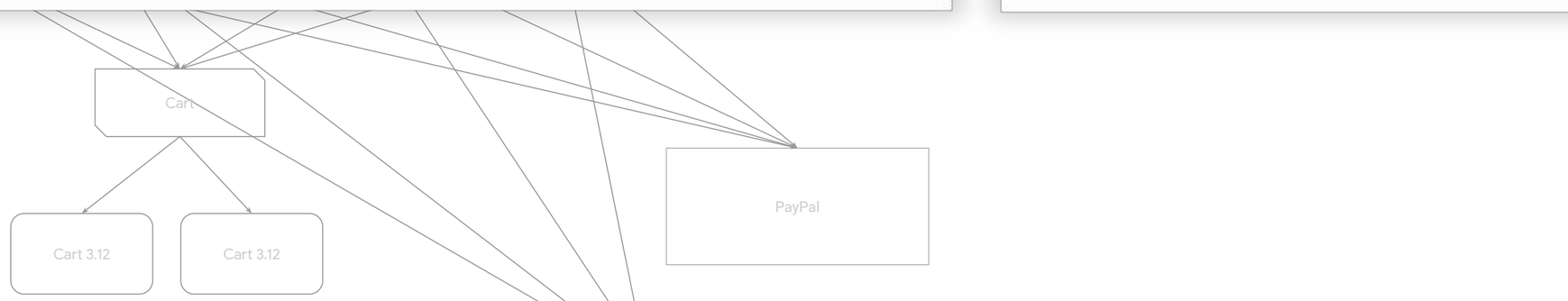
This grey box could be considered a dependency if it is required for the ‘products’ pieces to function - if not, ‘service’ is a better fit

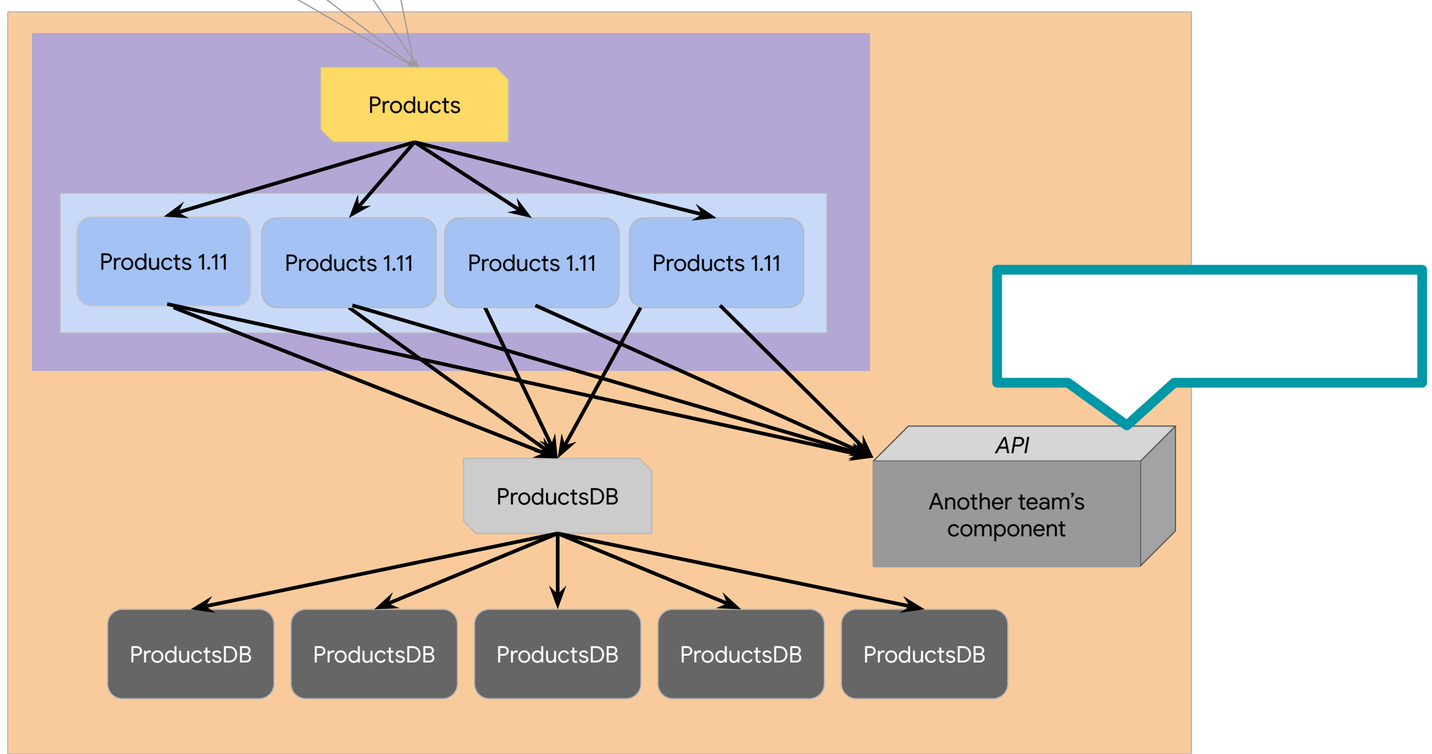
To be most precise, this can also be referred to as the ‘[team that owns it] API/service’

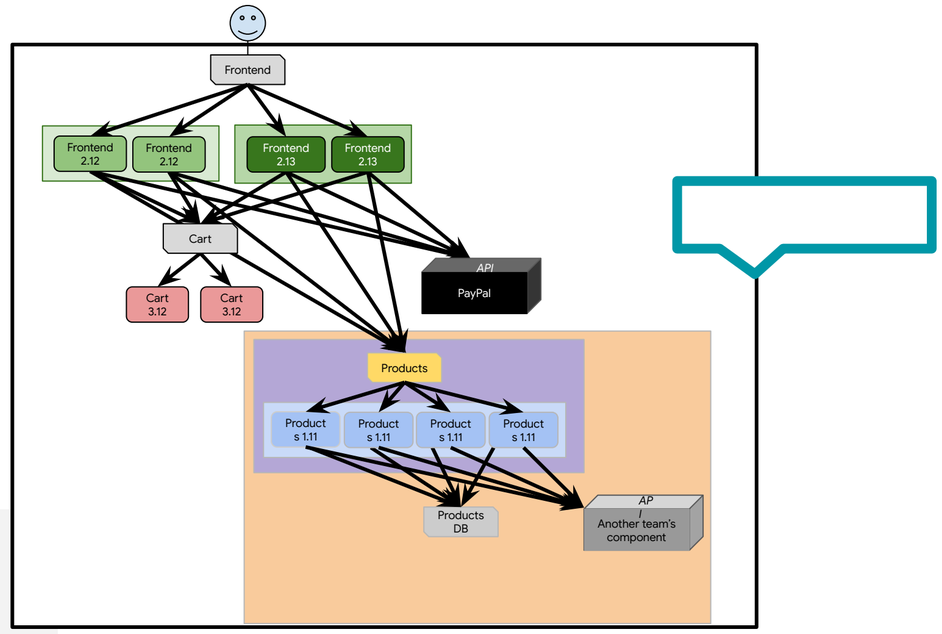
> “If the app needs to use it then it’s more of a dependency. But if it can go forward without it then it’s just getting an assist, and I would call that a service.”


Grey box
10
9
8
7
6
5
4
3
2
1
0
Service
API
Dependency
Other
External
Pilot
Utility class
Processor
Client


Grey box - list
10
9
8
7
6
5
4
3
2
1
0
Dependency
Service
Microservice
Other
External
Pilot
Package
Endpoint
Component
Module


Cart
Cart 3.12
Cart 3.12
PayPal


Products
Products 1.11
Products 1.11
Products 1.11
Products 1.11
ProductsDB
API
Another team's
component
ProductsDB
ProductsDB
ProductsDB
ProductsDB
ProductsDB




# Everything together

**Significance**: Understand how users refer to the entire product they work on

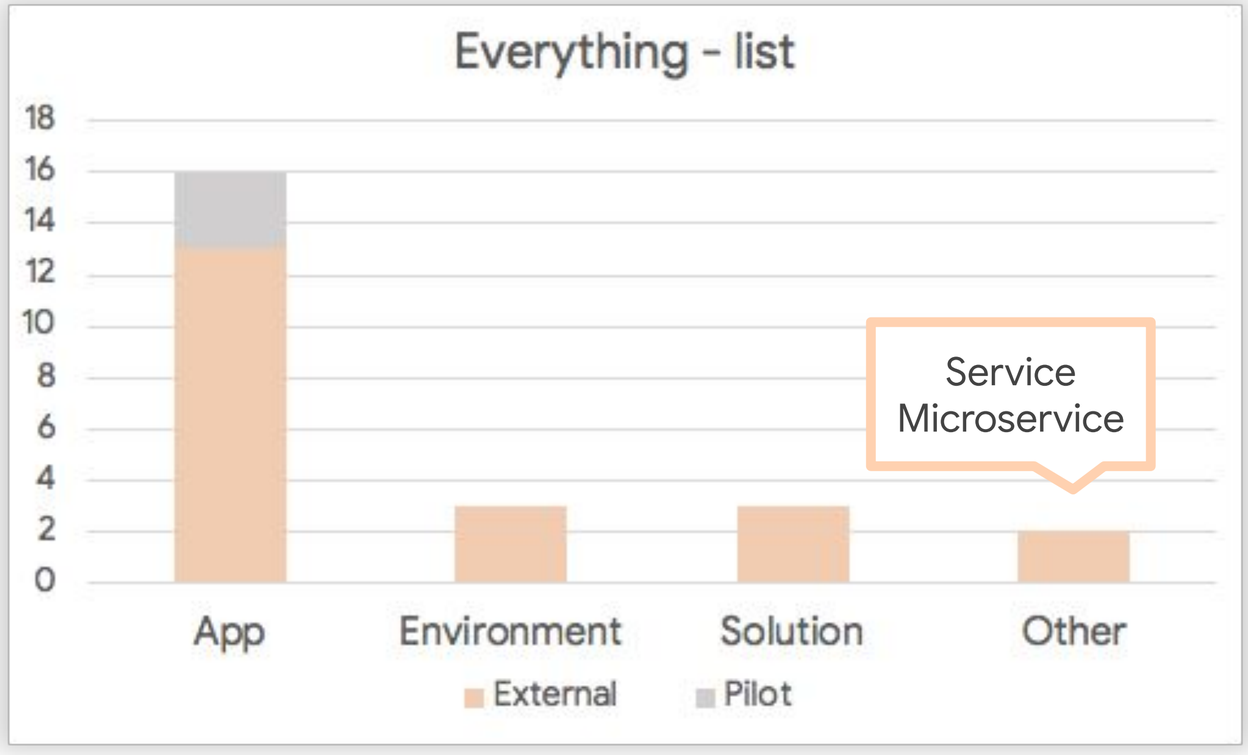
As the person working on this product, ‘application’ makes the most sense; to end-users, ‘website’ was a better choice

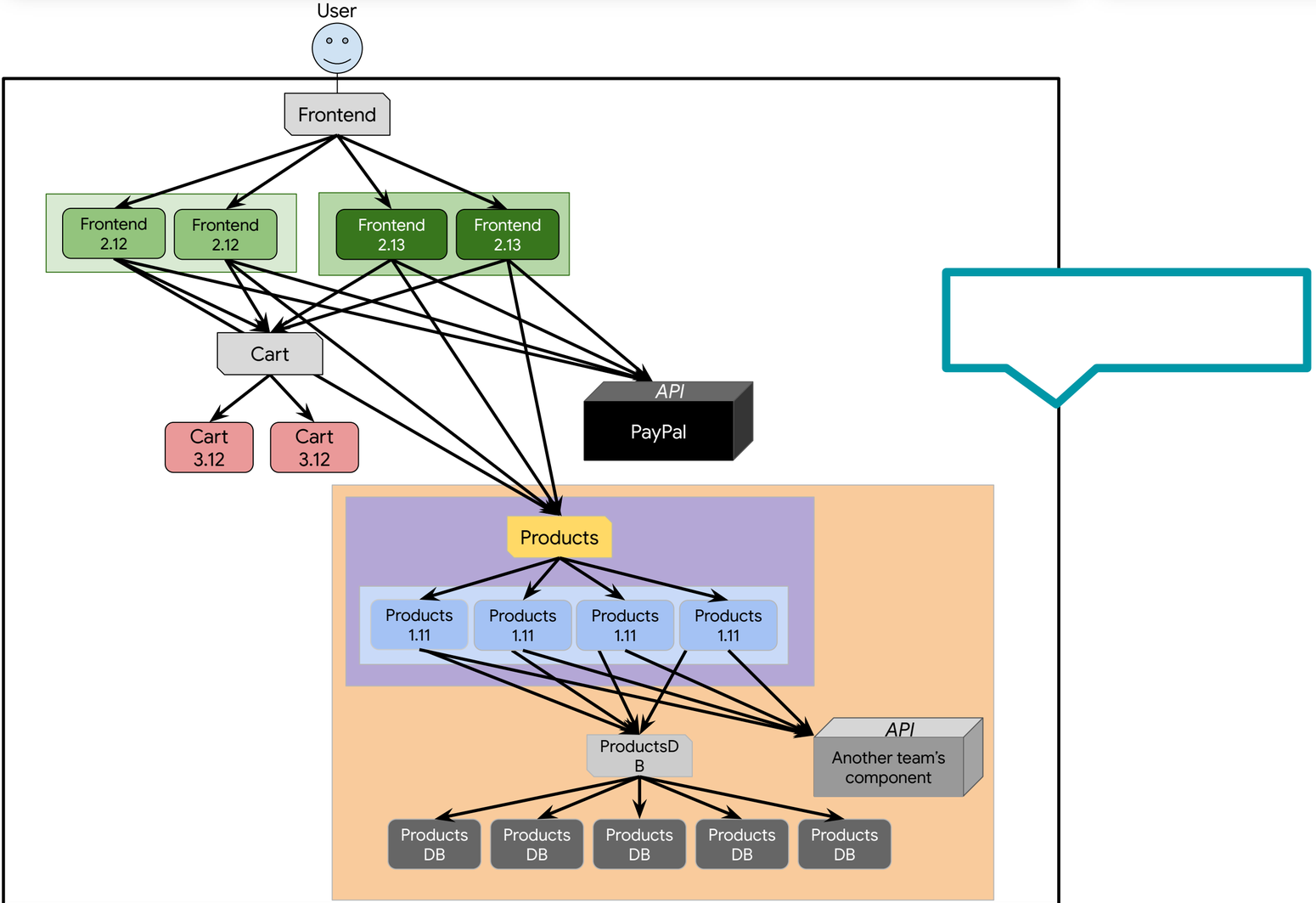
Shortening to ‘app’ may sound too specific to mobile

‘Product’ is all-encompassing, so could also make sense if the diagram refers to every feature of the thing this person works on

“I really would just call it an application. That's because it comprises everything from frontend to backend as well as any external services. To call it anything but application would be really difficult.”

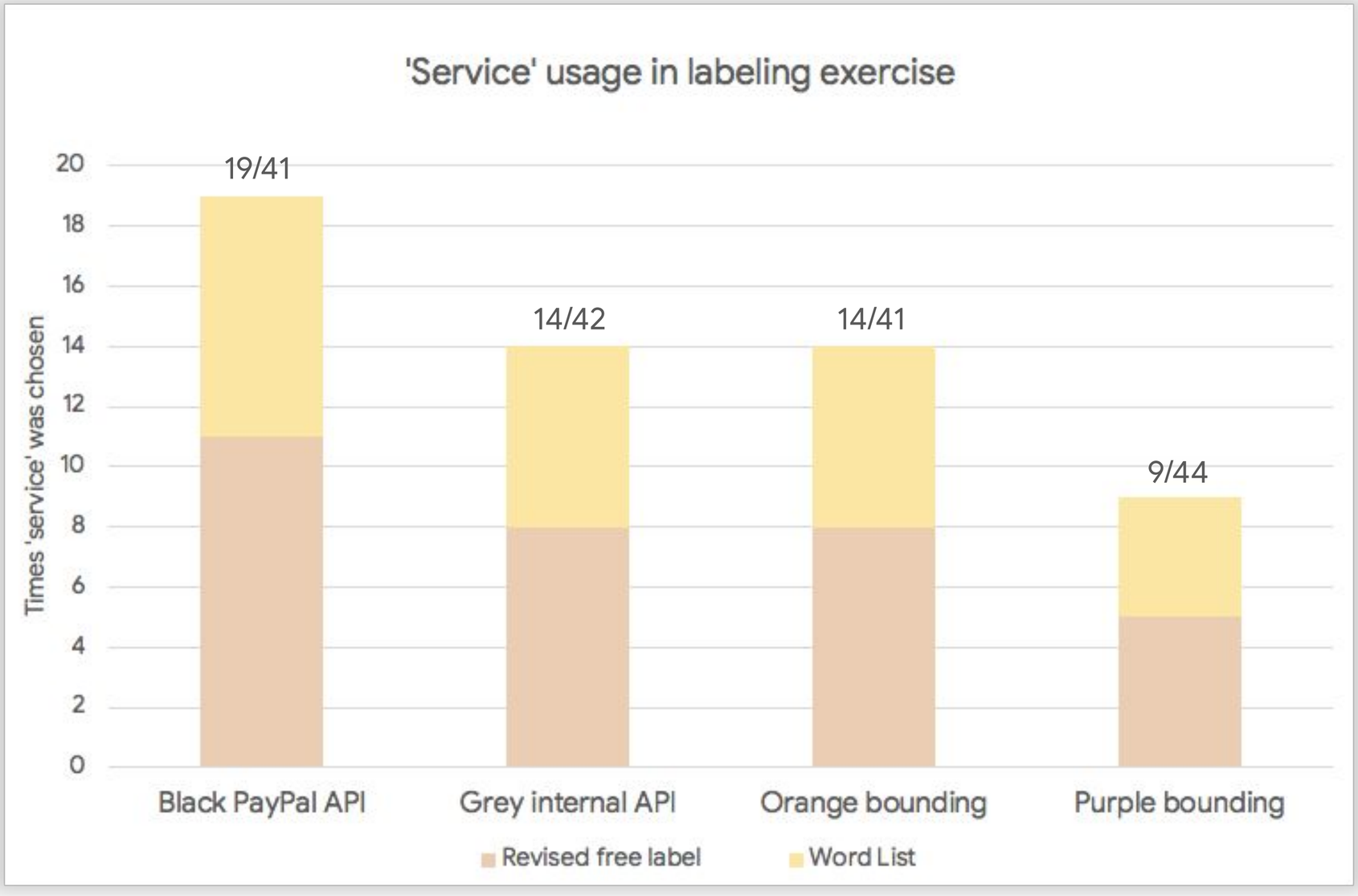

Everything
10
9
8
7
6
5
4
3
2
1
0
App/
App derivative
Website
Product
Stack
Other
External
Pilot
Service
Web service
Checkout page
Experience
Solution


Everything - list
18
16
14
12
10
8
6
4
2
0
App
Environment
Solution
Other
External
Pilot
Service
Microservice


User
Frontend
Frontend 2.12
Frontend 2.12
Frontend 2.13
Frontend 2.13
Cart
Cart 3.12
Cart 3.12
API
PayPal
Products
Products 1.11
Products 1.11
Products 1.11
Products 1.11
ProductsDB
API
Another team's component
Products DB
Products DB
Products DB
Products DB
Products DB

# The term ‘service’ was most applicable to the PayPal API

‘Service’ seems to be less popular for constructs that are granular and transparent [diagram for reference]





1 2

# What is a service?

# Services independently provide value

A service is self-sufficient and performs a valuable business function

Theoretically, a service could continue to functioning even if the database goes down

A service can be thought of as the “source of truth” for the information it provides, typically within a defined scope

“Services provide a complete, useful interaction. The components of a service are not useful in isolation.”

“A service is an entity which provides or does a useful piece of work, usually through a request-response cycle.”

# Services are strictly defined and error-resistant

Services receive requests and send responses with defined rules - users know what input the service needs and can anticipate what they'll get back

Services are resilient to failure (multiple instances running) and syntax errors (sending raw SQL doesn't feel like a service because SQL takes free form input instead of defined rules)

> "When I think of a service, it's something that has a distinct set of rules. There's a list of things that I need to give it in order to get something in return."

> "I think a well defined service doesn't let you make syntax errors.It should be as resilient as possible to people making mistakes whether they're accurate or not."

# Services are the sum of many parts

Rather than specializing in a single task, services handle multiple responsibilities

There can be services within services - e.g., a large backend service could have a smaller logging service and runtime configuration service inside it

The “size” of a service matters - small, singular tasks and calculations may not be considered a service

“Services are broken up into small microservice components. An app has many services talking to each other.”

“Services are not a single elementary component.”

# Services are usually opaque

Participants referred to ‘service’ most consistently as entities which are outside the realm of what they personally work on

Users often don’t know the internal mechanics of a service; it is expected that a service will respond to requests reliably without any manual configuration on their part

> “A service is an endpoint that does something for you and then returns data. You don’t really know the internals of the service, you just let it do its thing and then it comes back to you. It has defined inputs and outputs so that you know exactly what you need to give it and then what you get back.”

> “I don’t have to worry about configuring that service - based on contracts you do what you need to do and you shouldn’t have to tell it how to do its job. It’ll just do it and come back to you.”

# The scope of a service is not entirely agreed upon

Participants generally agreed that a service consists of a load balancer, a compute component, and can include a database

Views were split on whether services are typically considered internal or external - some thought the term ‘service’ implied internal unless specified, while others considered ‘service’ only outside the scope of their work (e.g., calling through SQL is not a service because that’s internal; service is usually called externally through an API)

“A service is something we need that’s outside of our control. External service means not built by the current team, so definitely applies to the black box and maybe the grey box too.”

“Service is something that belongs to us. API or endpoint are older terms for other people’s stuff.”

# Service vs. microservice

Participants were divided on whether a meaningful difference exists between ‘service’ and ‘microservice’

A service often signified a broader “collection” of tasks that sends back something of value; a few distinctions for ‘microservice’ may be:

- Microservices are considered smaller subsets a service
- Microservices complete simpler, more specialized tasks (e.g., lambdas)
- Microservice refers to a methodology rather than a piece of architecture

> “Microservice refers to the idea of breaking up your monolithic codebases into services, but a monolithic codebase could be a service. You might just not call it microservice.”

> "If it had just one small task like shipping cost calculation, that I would rather call microservice compared to just service."

# General findings on nomenclature

Users acquire terminology through their team definitions, the products they use, and their cloud provider

Qualifiers are very common, and often necessary for understanding - given the potential for confusion, people might avoid ‘service’ altogether and opt for more specific terms where possible (e.g., cart service, PayPal service)

Due to the overloading of generic terms, users have to clarify and “translate” the meaning of words like ‘service’ in conversation - without context, differences can be difficult to interpret in UIs and documentation

# Follow-up research

# Additional findings with Kubernetes users

Data collection was extended to gain more participants (n = 18) with moderate or advanced Kubernetes proficiency.

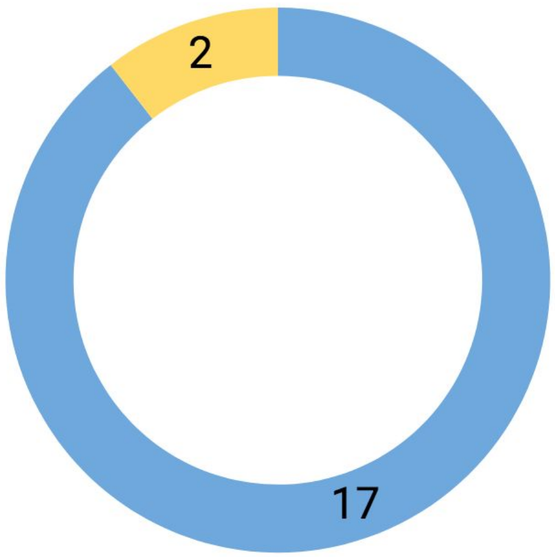
In summary:

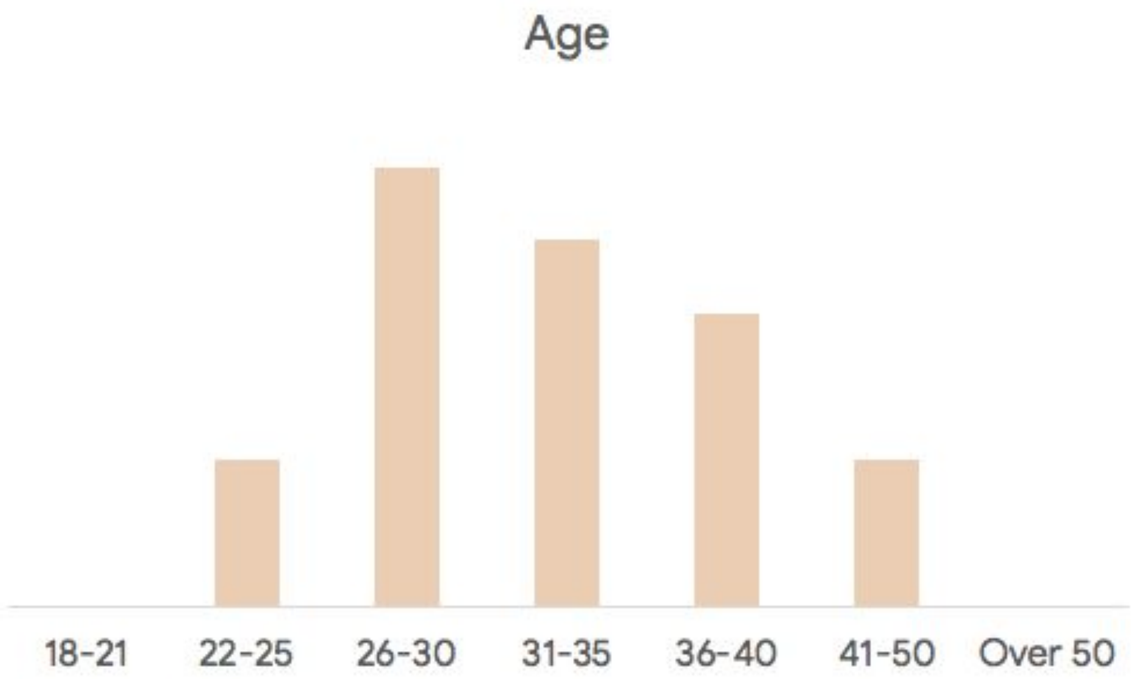
- Kubernetes users also referred to the Kubernetes ‘service’ concept as a load balancer
- ‘Service’ was the most common label for the Knative ‘service’ concept
- When asked to give a name to the “thing I work on,” half of participants reported “service” as the best term

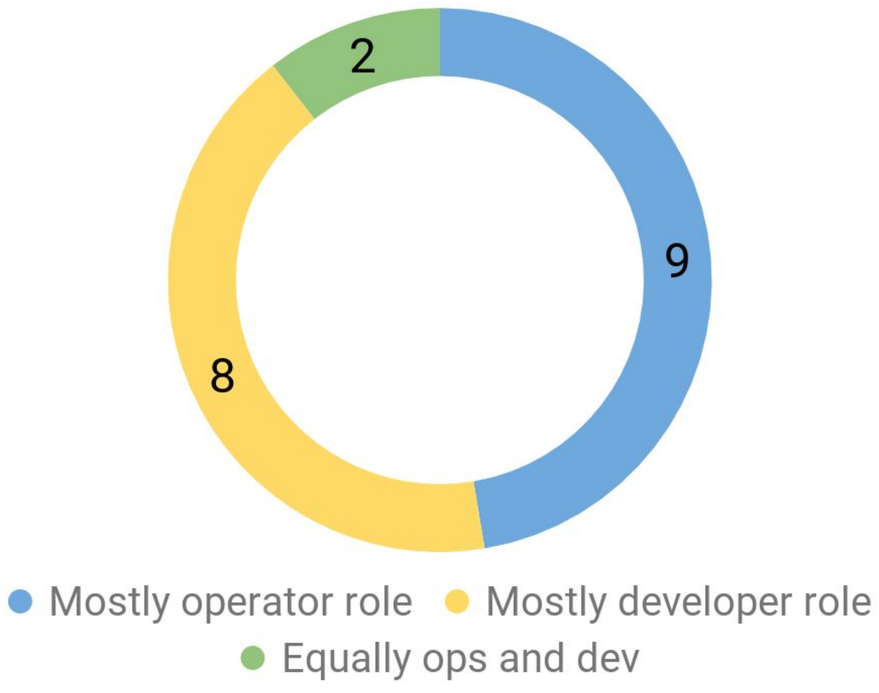
# Appendix

# Participant details

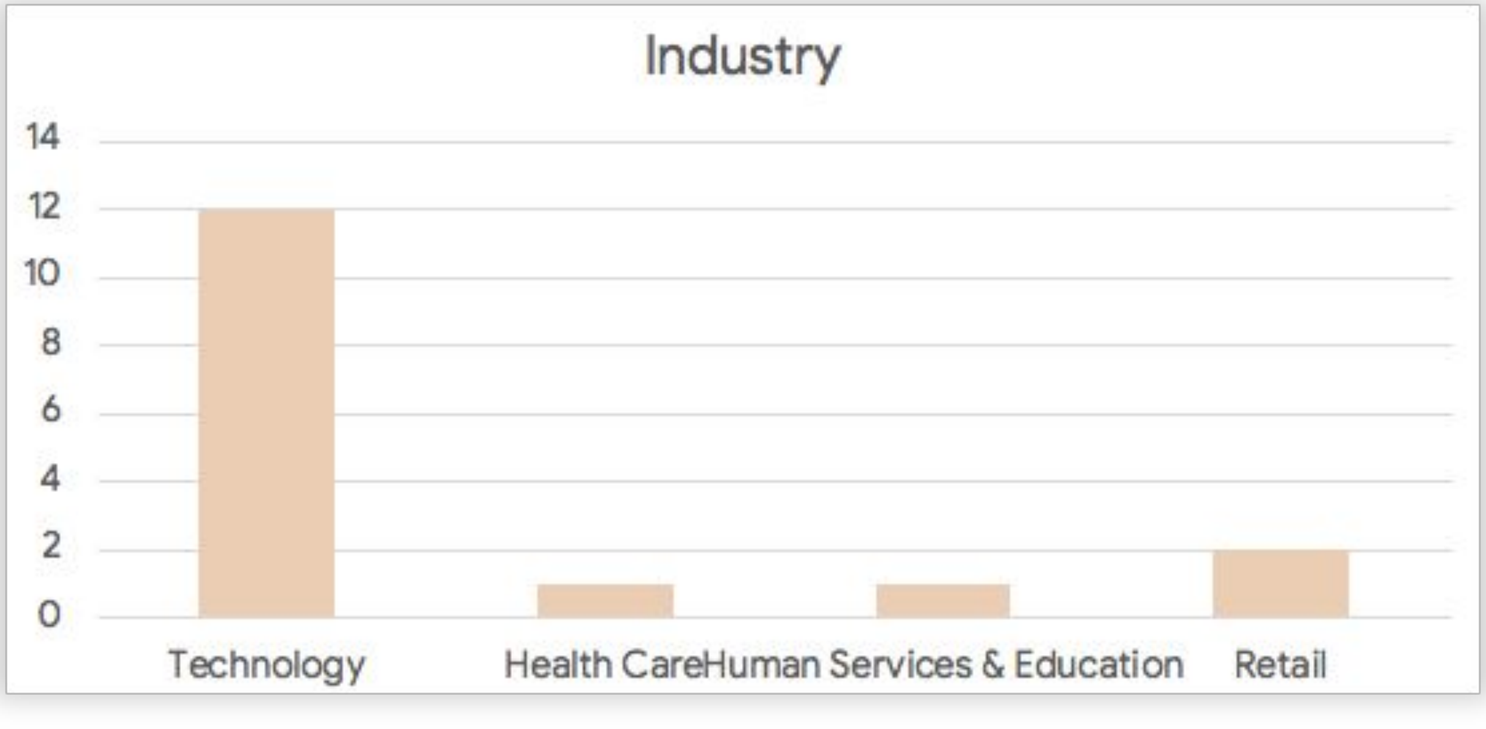
# Participant demographics

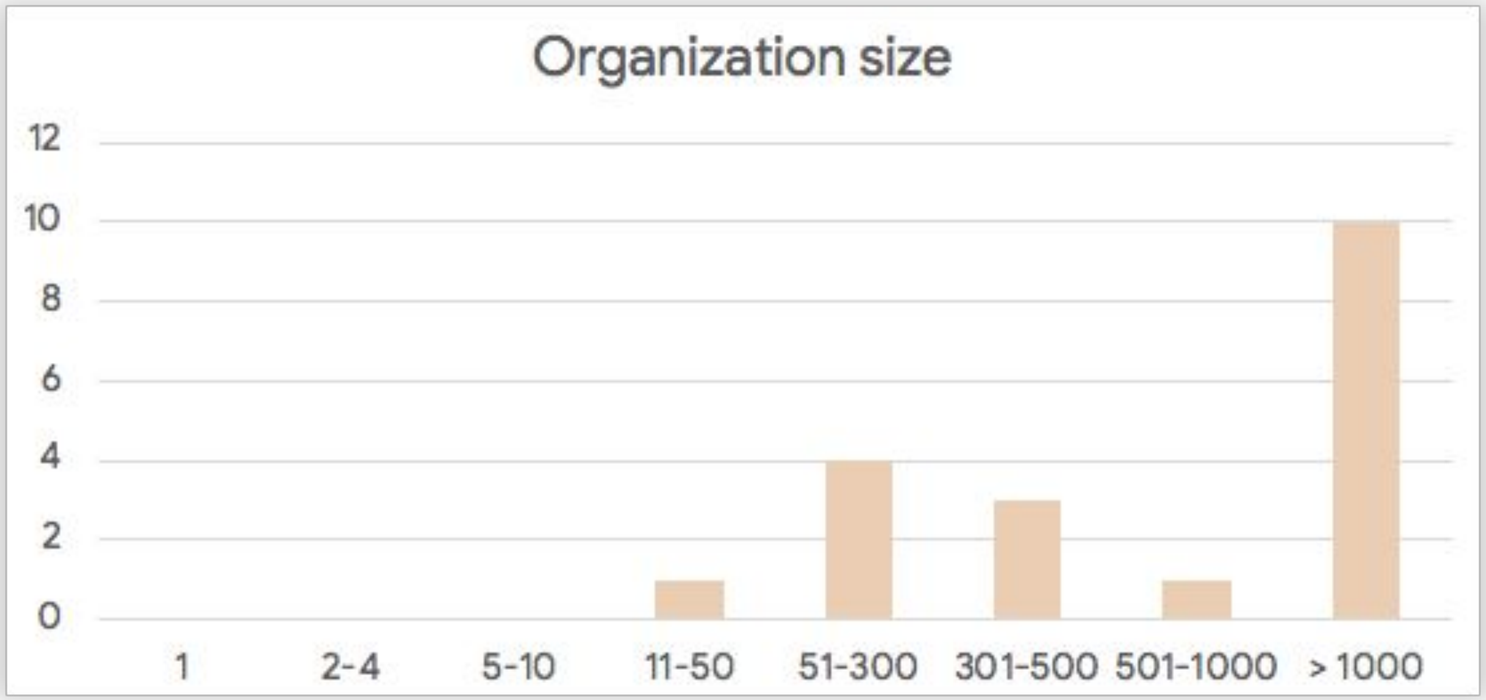


# Organization



# Product experience

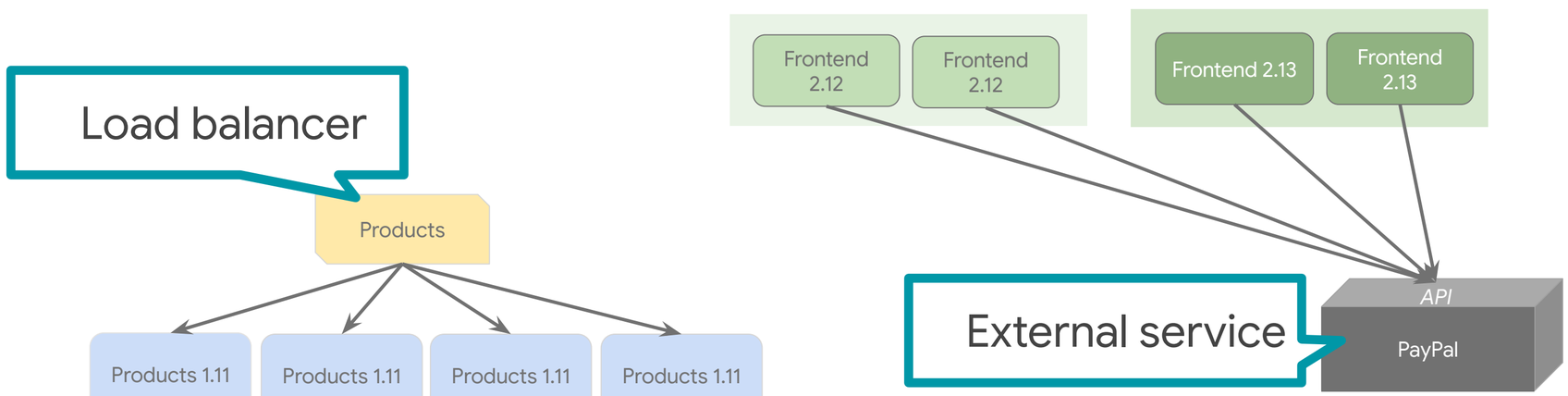
| Non-GCP Cloud | | | GCP | On prem |
|---|---|---|---|---|
| AWS | Azure | Other | | |
| 7 | 1 | 2 | 4 | 5 |

| Kubernetes experience | # of participants |
|---|---|
| Never heard of it | 1 |
| Heard of it | 6 |
| Understand basic concepts, but never used | 5 |
| I've tried it/played around | 3 |
| Could complete basic tasks with minimal guidance | 1 |
| Advanced user | 3 |
| Expert | 0 |

# Labeling summary

# Consistent terms

≥ 50% agreement



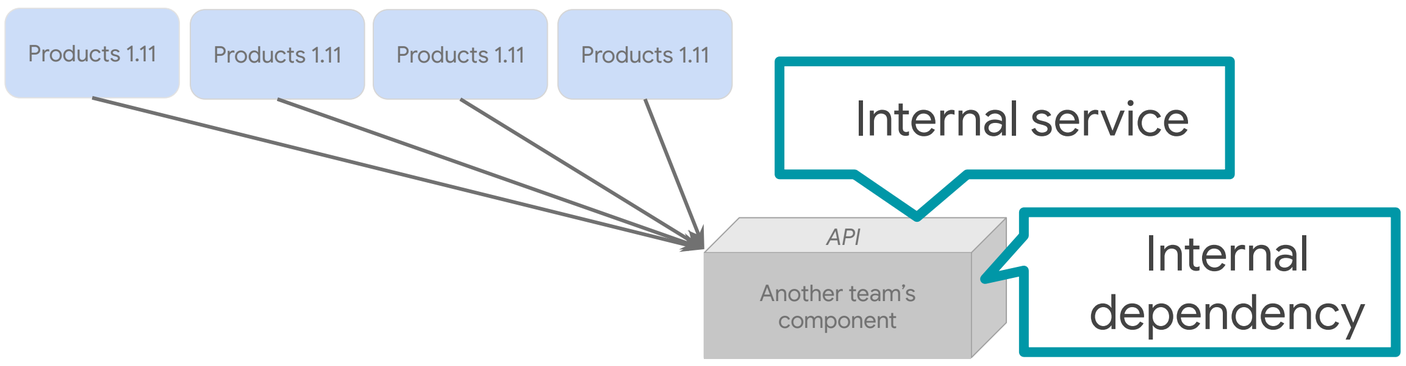

**10/19** labels

**11/22** labels

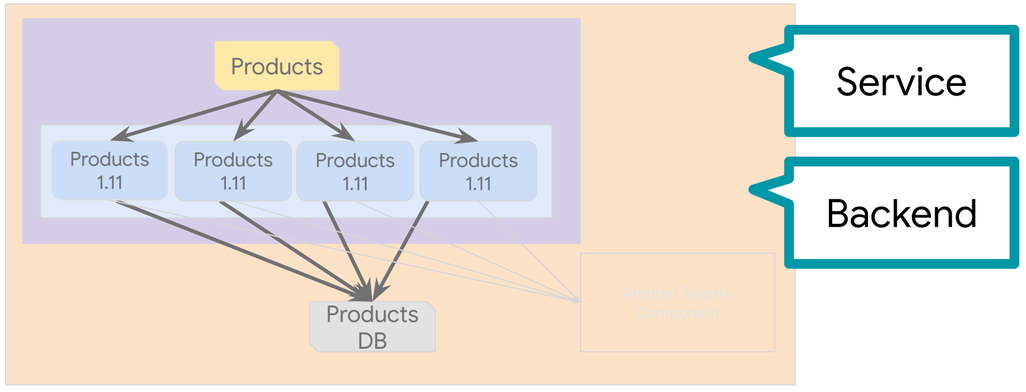
*From revised free labels of external participants; some participants used multiple terms

# Less consistent terms

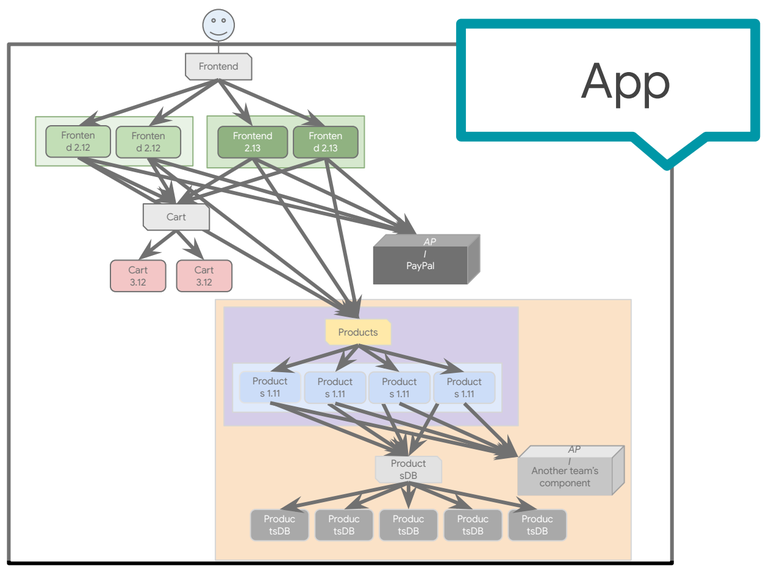
## 33% - 49% agreement



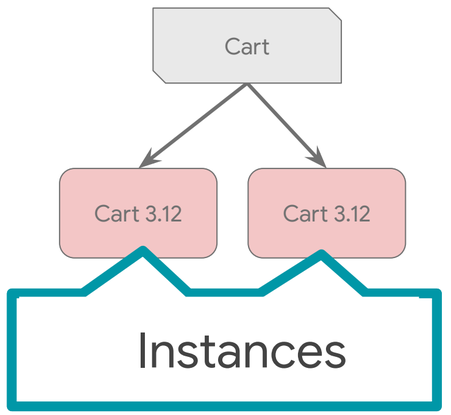

Service: **8/21** labels*
Dependency: **6/21** labels




Service: **8/22** labels
Backend: **8/22** labels




App: **9/21** labels

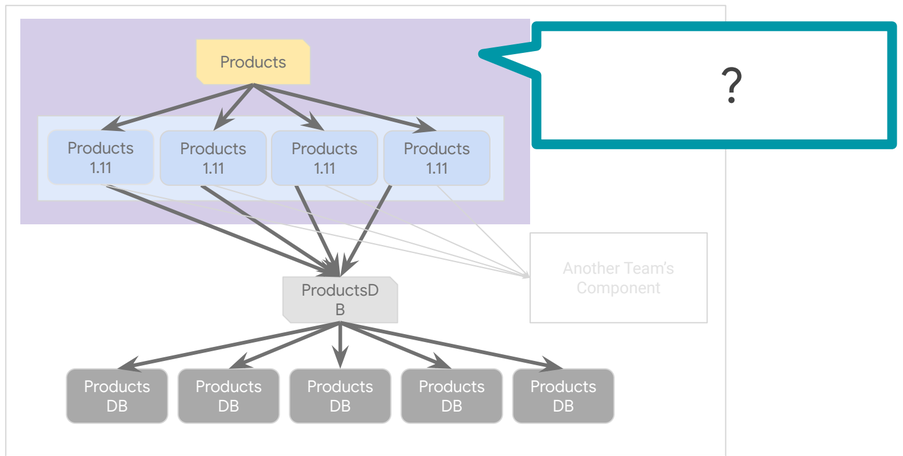



**10/23** labels*

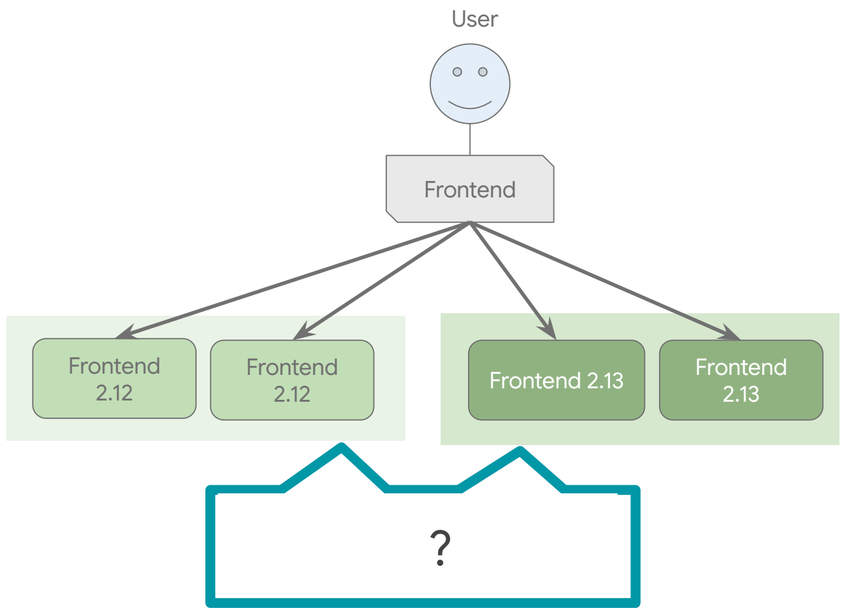
*From revised free labels of external participants; some participants used multiple terms

# Least consistent terms

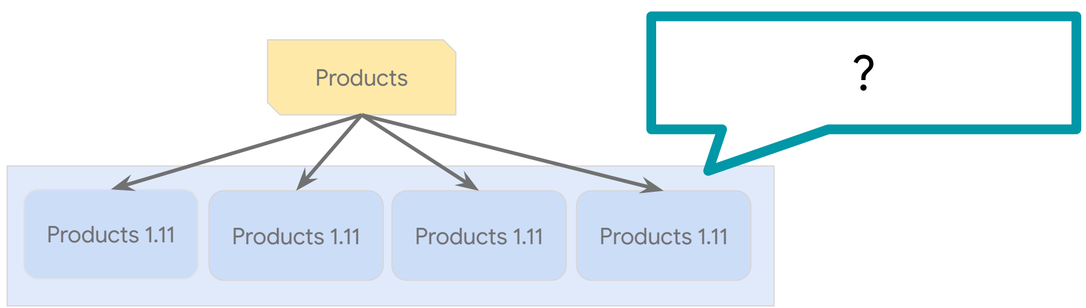
## < 33% agreement




Service: **5/23** labels*



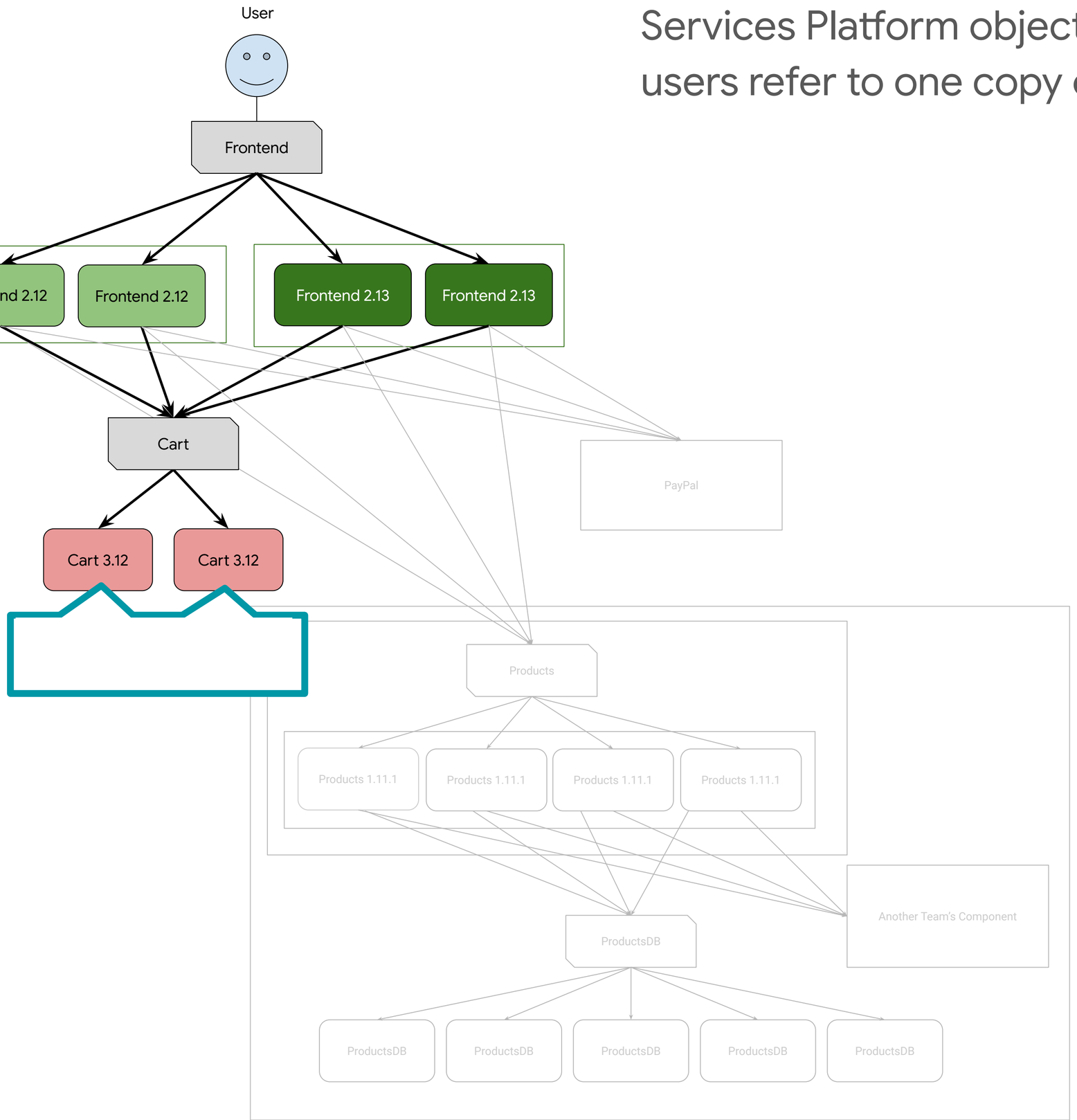

Versions: **6/19** labels
Releases: **5/19** labels



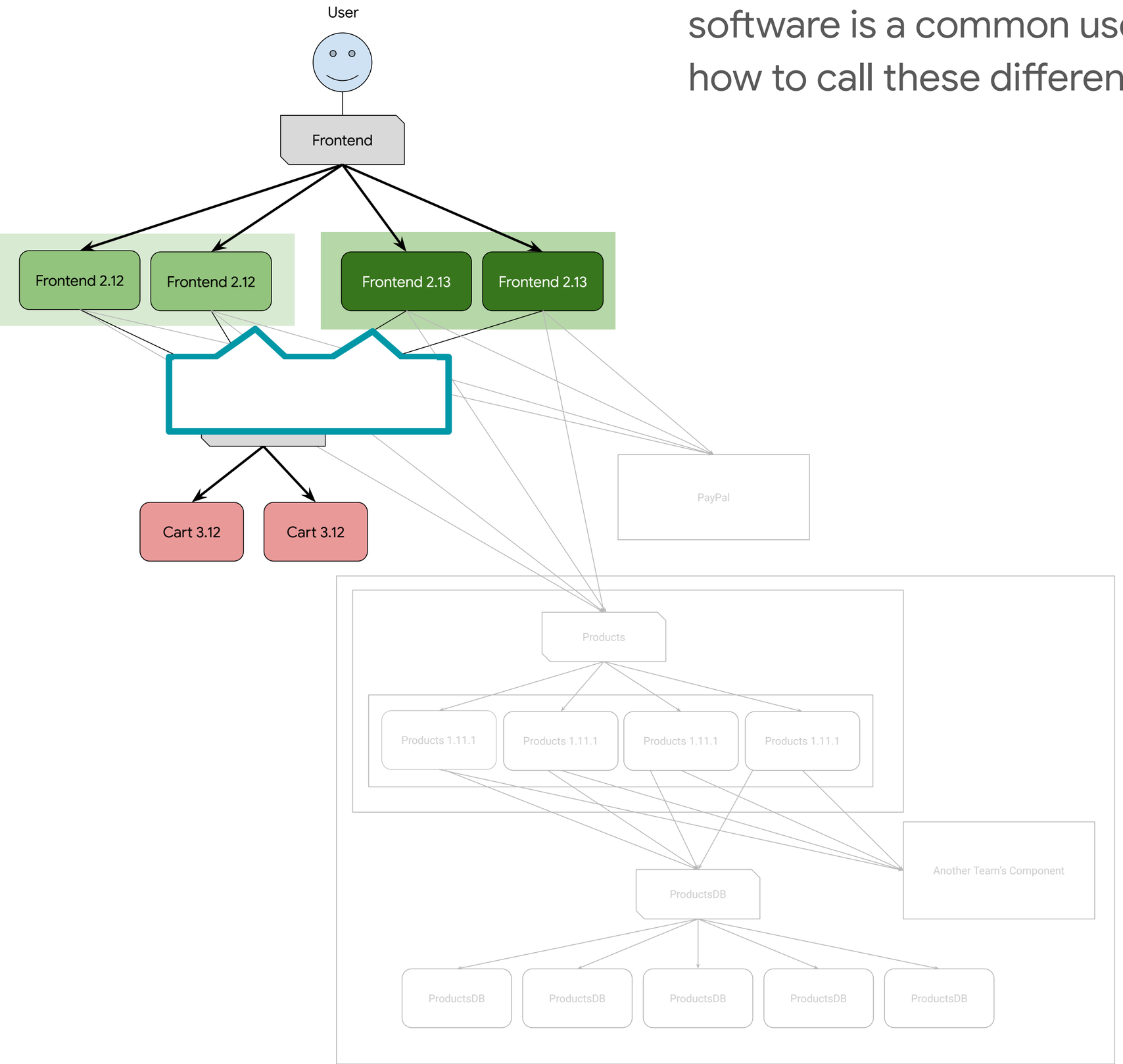

Instances/instance group: **6/21** labels

*From revised free labels; some participants used multiple terms

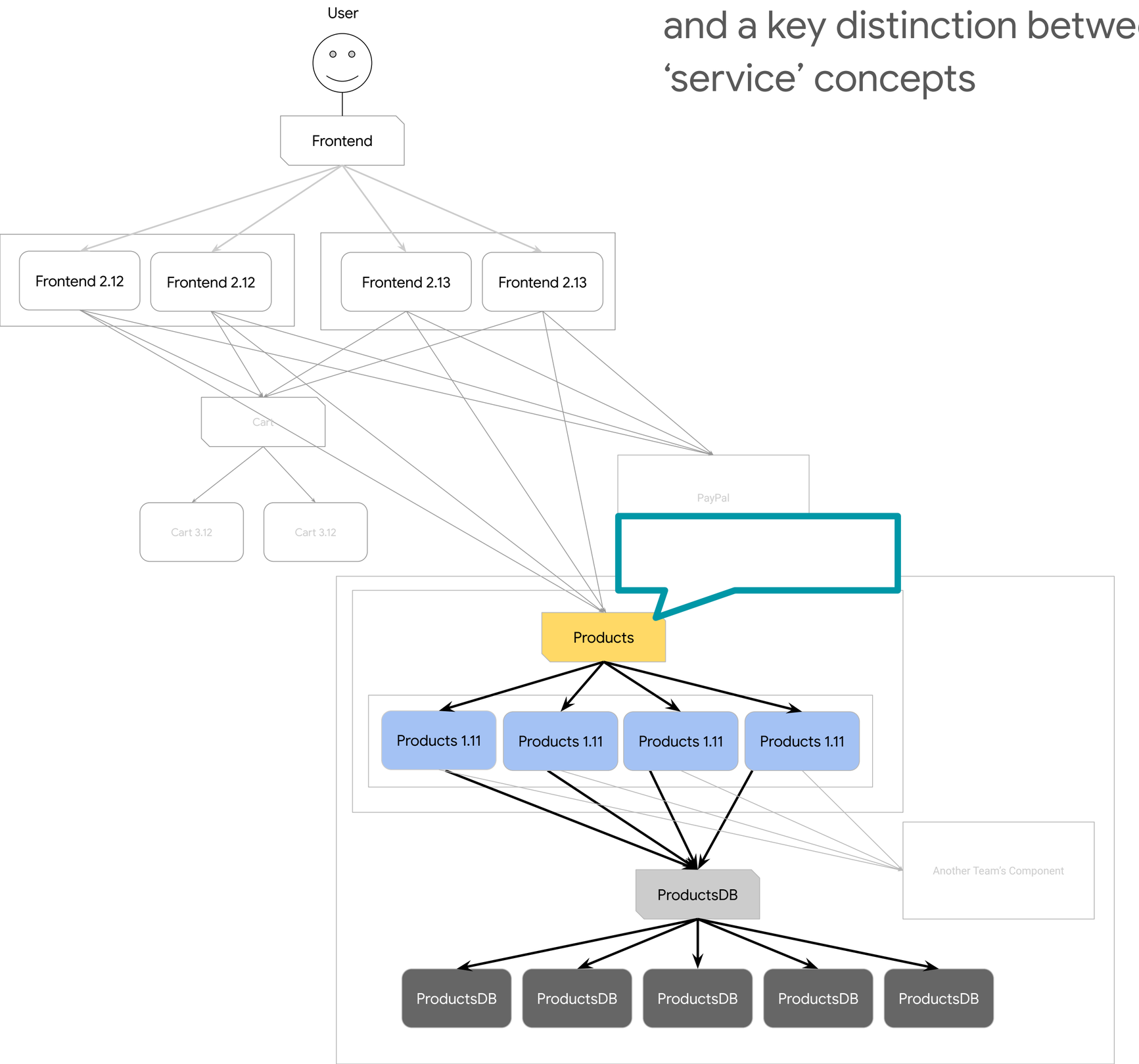
# Diagram implications



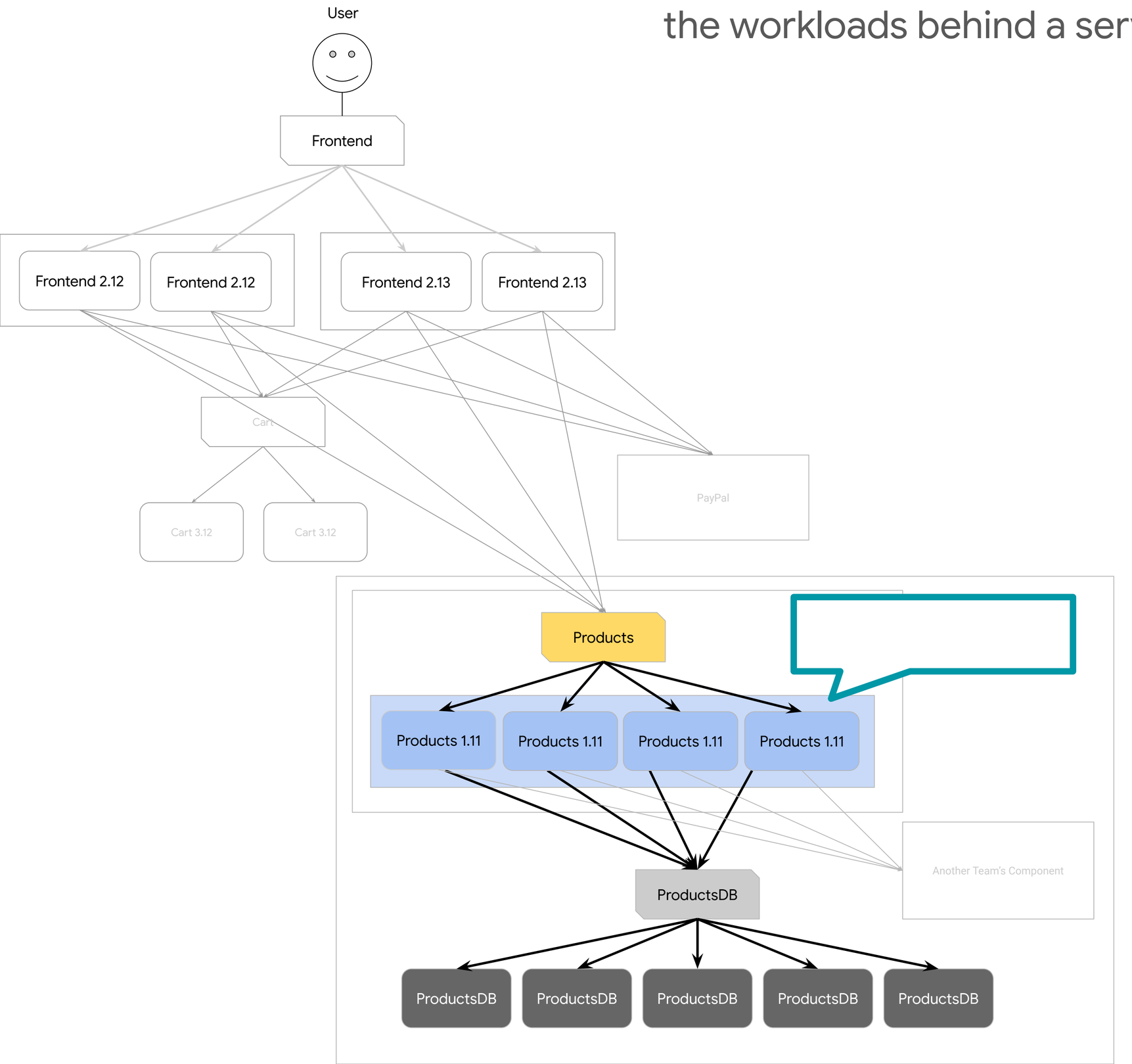

- Why this is important: Allows us to validate our Cloud Services Platform object model by understanding how users refer to one copy of the code they're running



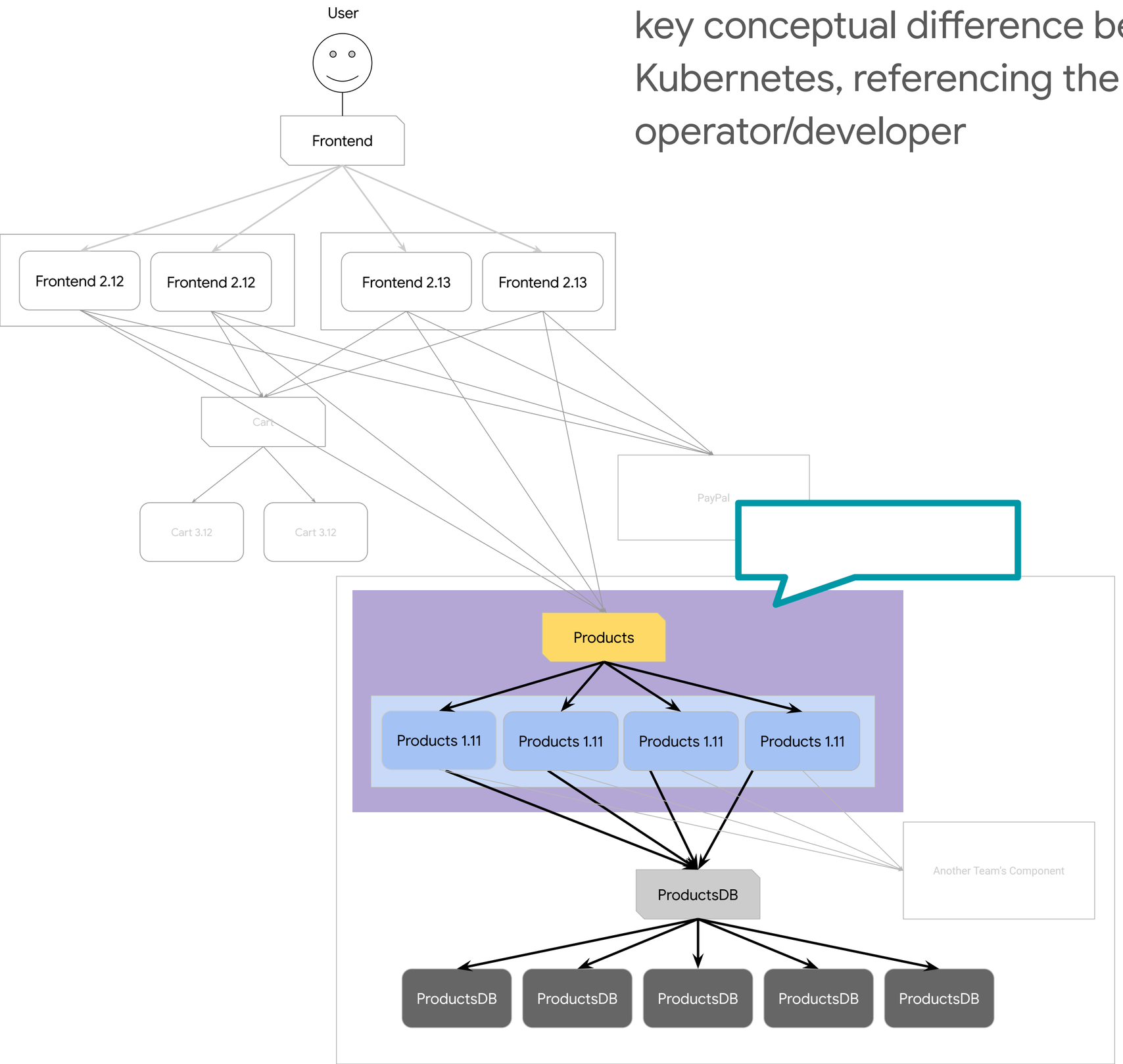

- Why this is important: releasing a new version of software is a common user journey, so we need to know how to call these different groups in the UI



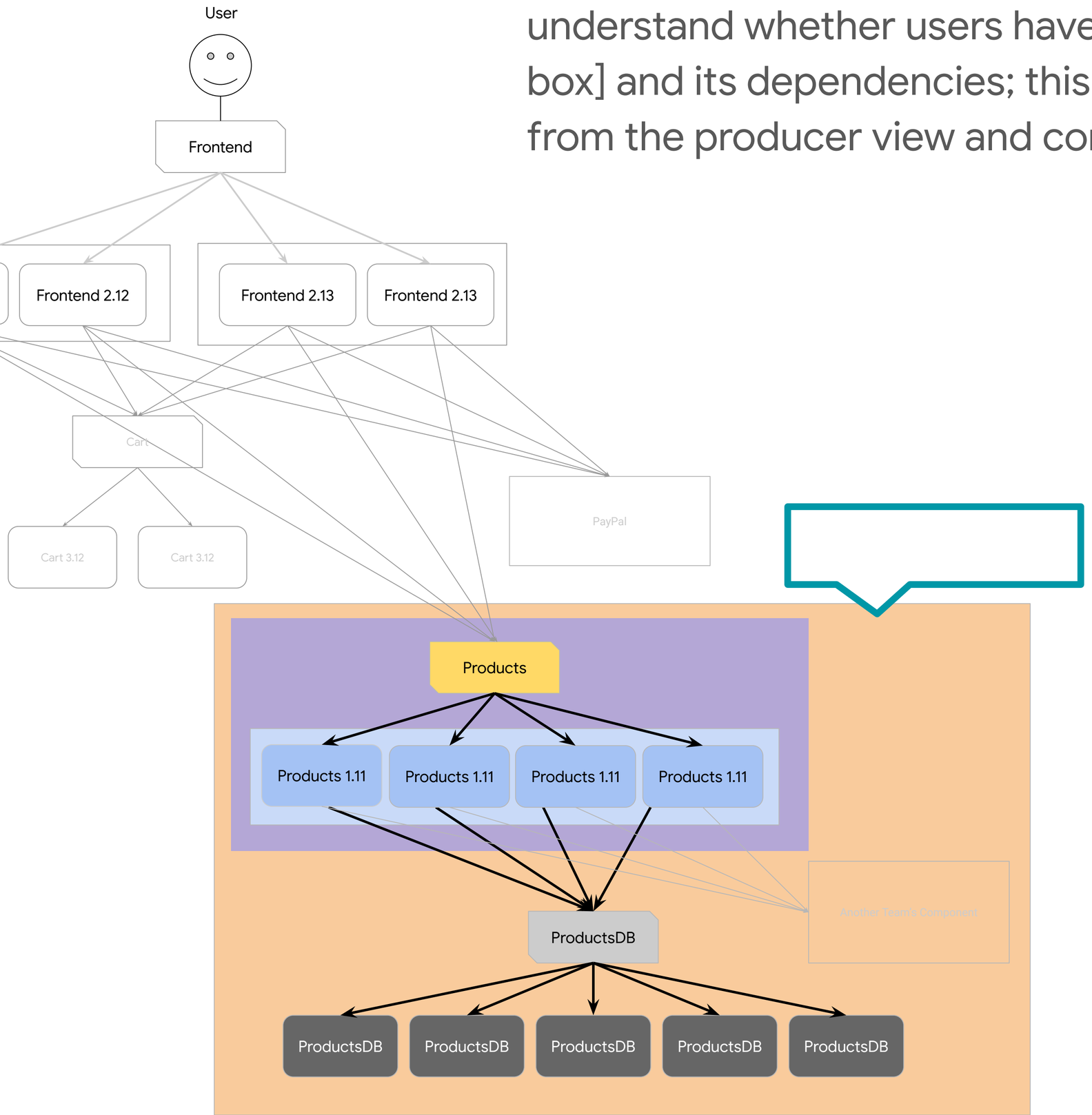

- Why this is important: represents a Kubernetes service and a key distinction between Kubernetes and Knative ‘service’ concepts



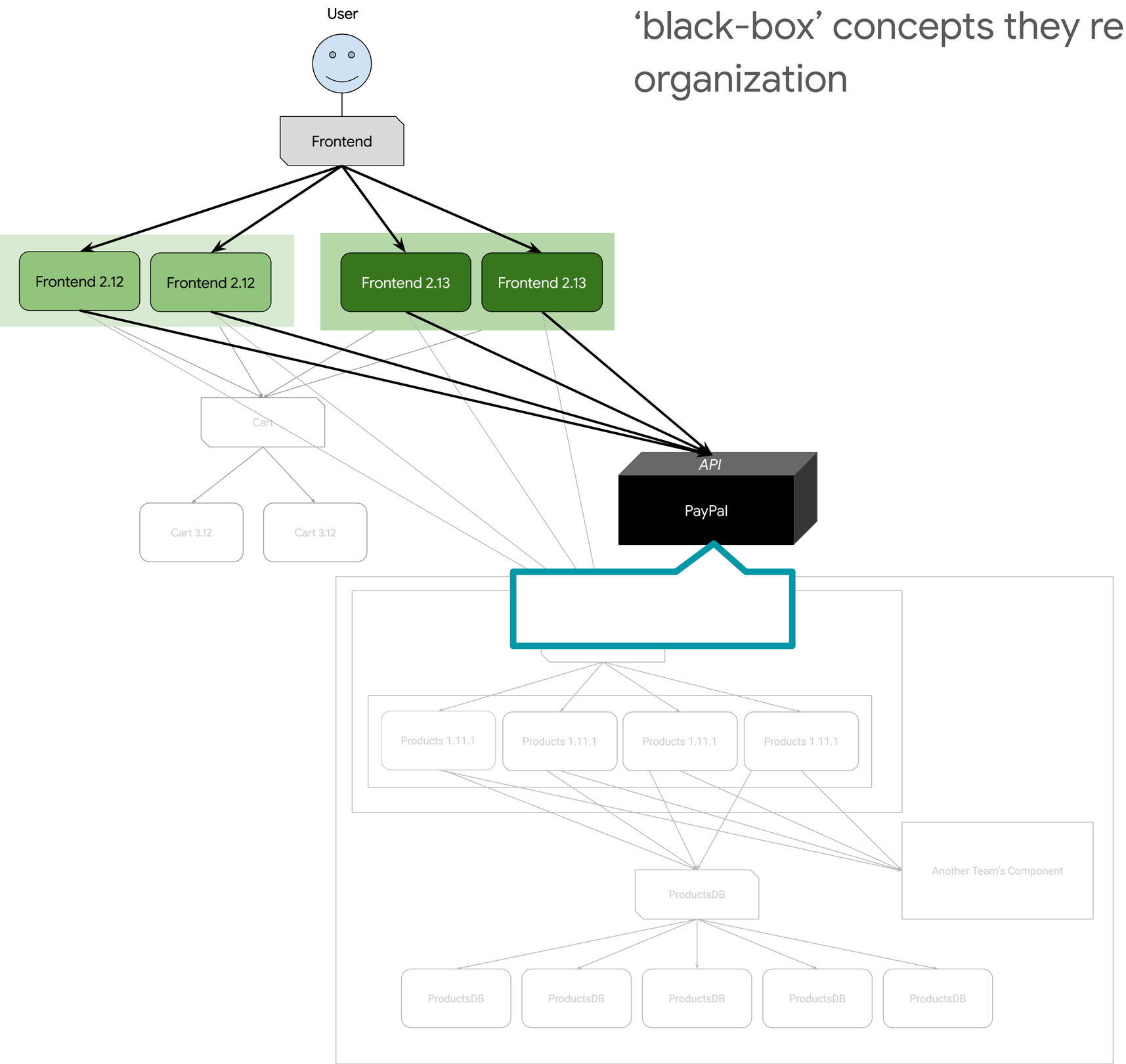

- Why this is important: helps us understand how to term the workloads behind a service



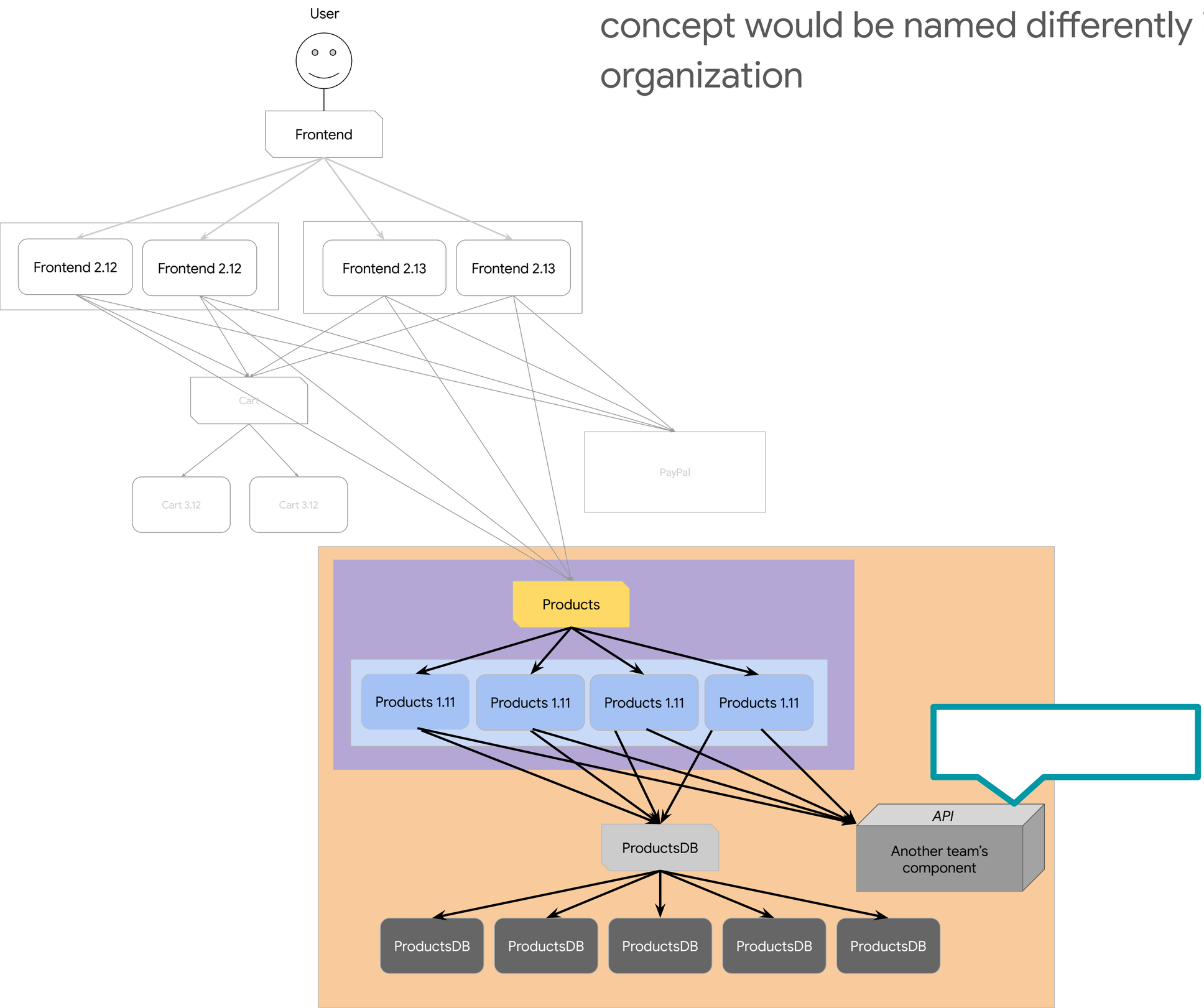

- Why this is important: represents a Knative service, and a key conceptual difference between Knative and Kubernetes, referencing the work of a single service operator/developer




- Why this is important: distinction from the purple box to understand whether users have a name for their [purple box] and its dependencies; this also allows us to distinguish from the producer view and consumer view of a service




- Why this is important: understand how users refer to ‘black-box’ concepts they rely on, but are outside their organization




- Why this is important: understand whether a black-box concept would be named differently if it was internal to the organization

- Why this is important: understand how users refer to the entire product they work on