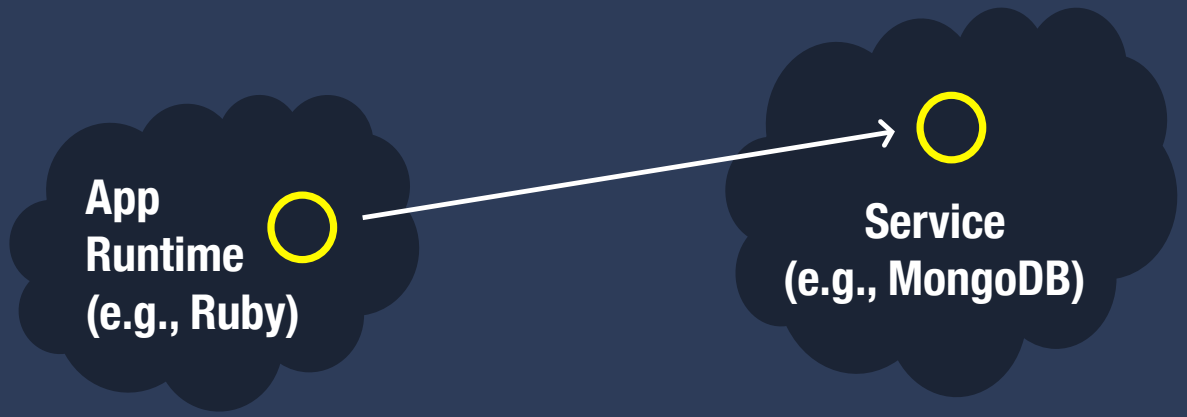
Don't Call Them Middleboxes, Call Them **Middlepipes**

Hani **Jamjoom** – Dan **Williams** – Upendra **Sharma**

IBM T. J. Watson Research Center

PaaS Makes Things Easy

- Abstract out infrastructure resource management
  - e.g., BlueMix, Cloud Foundry, Heroku, Azure, AppEngine
- Simplify consumption of **runtimes and services**
  - e.g., "*I want a Ruby runtime or a MongoDB service*"
  - Automate provisioning, load balancing, auto-scaling, etc.

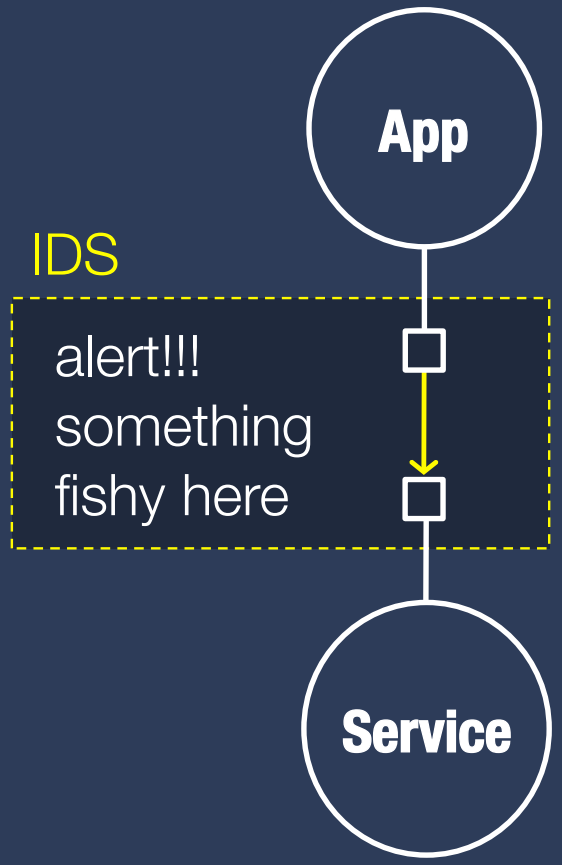


What About NFV & Middleboxes?

- PaaS hides most network configurations
  - Virtual networking, SDN, routing, firewalling
- Opportunity 1: **Simplify consumability of traditional middlebox functionality**
  - Intrusion detection, WAN optimizers, etc.
- Opportunity 2: **Support DevOps lifecycle**
  - Monitoring, circuit breaker, failure injection, A/B testing, etc.

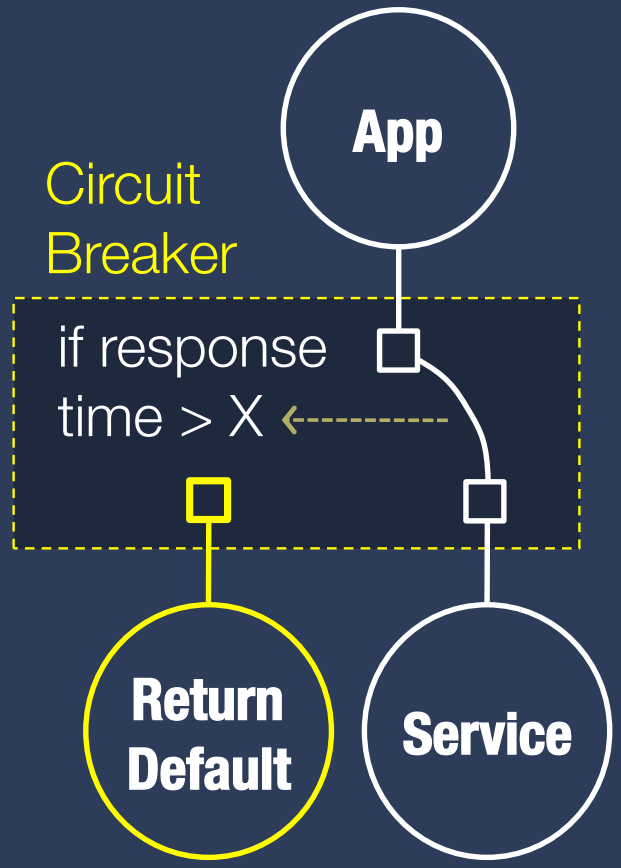
Example 1: **Adding Intrusion Detection**

- Scans packet headers and payloads
- Alerts or drops packets if intrusion is detected
- Typically, IDS/IPS are placed at the entry point of an application
- However, services can be offered by third-party vendors; intrusion can happen from anywhere

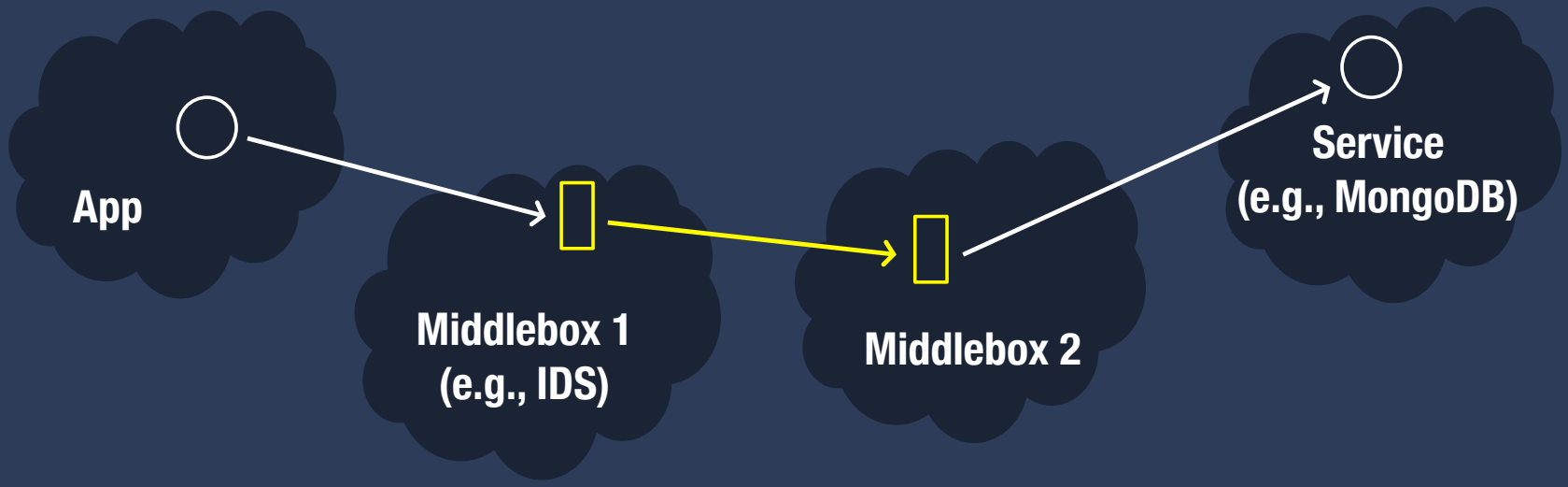


Example 2: **Mimicking Circuit Breaker**

- Stateful monitoring of requests
- Detect failure in downstream services
- Isolate failure quickly
- Return default value, raise exception at app, etc.
- Usually implemented in app logic
- Conceptually, a lot of the functionality can be separated from application logic.



Don’t Shoehorn Middleboxes Into PaaS Services



Issues with **middleboxes-as-services**

- They do not run close to apps
- They are difficult to chain
- They only operate on requests (not packets)
- They do not support callbacks into application

Middlepipes

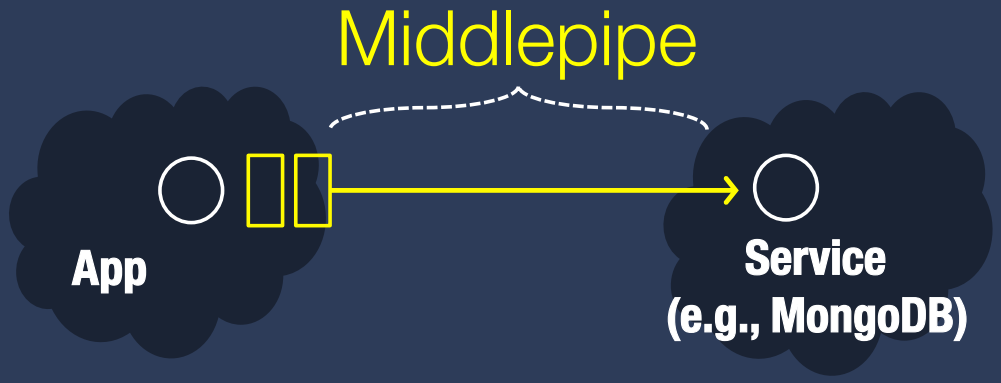
Middlebox-like functionality in a **software-defined pipe abstraction**

R1. Efficient interposition close to invocation

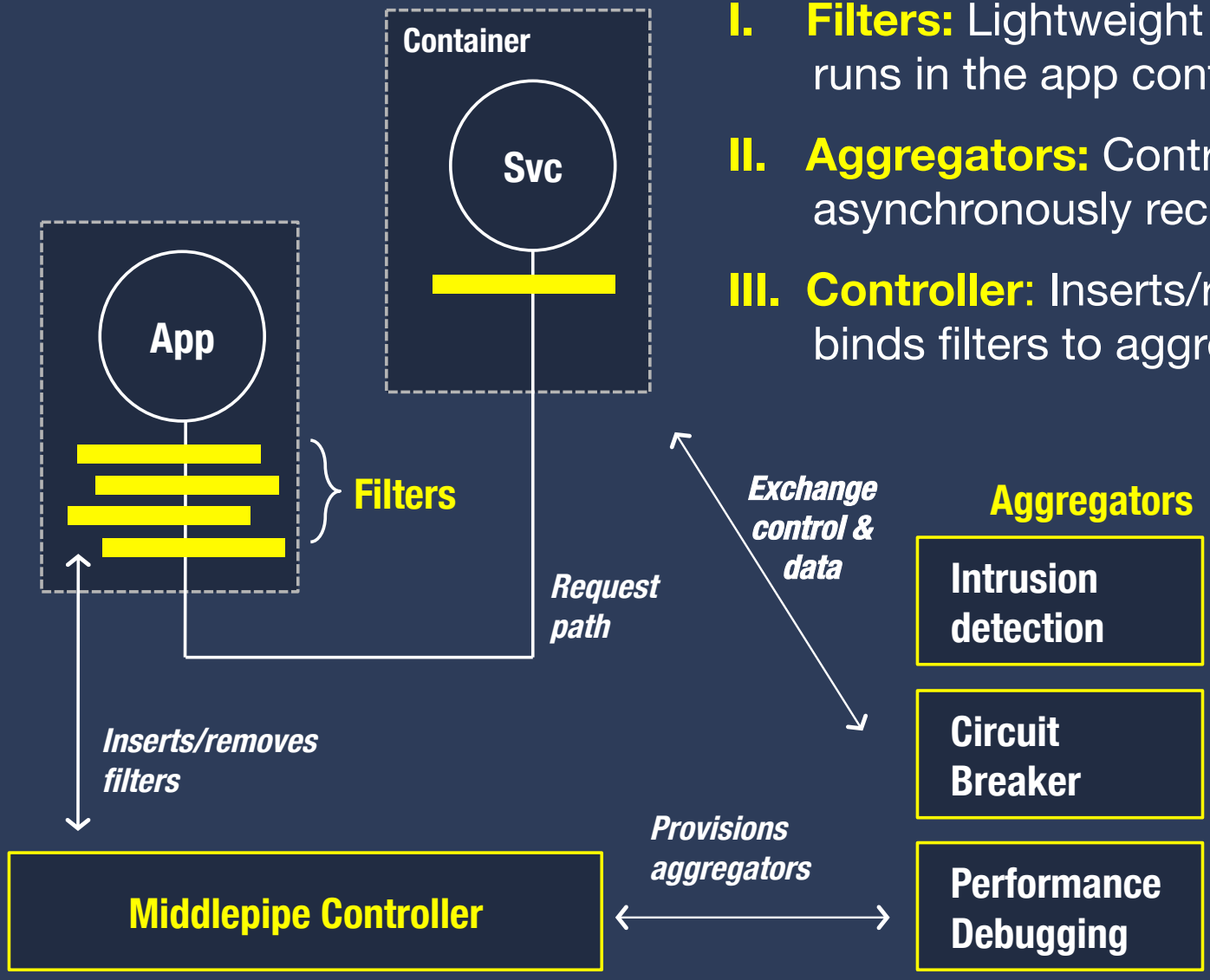
R2. Arbitrary chaining is supported outside of app logic

R3. Access to requests and packets

R4. Can generate callbacks to application



Under The Covers

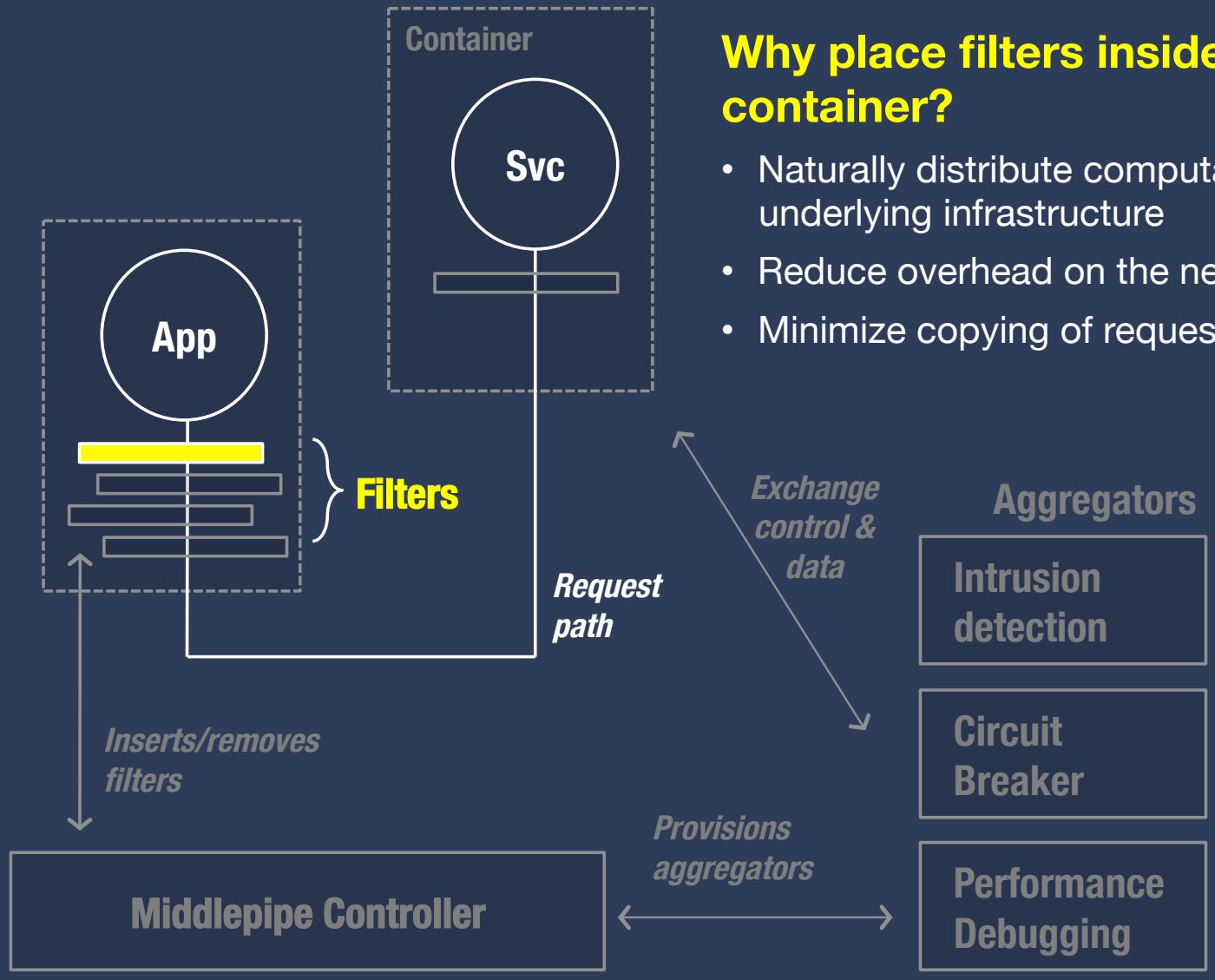


I. **Filters:** Lightweight "code" that runs in the app container

II. **Aggregators:** Control filters and asynchronously receive data

III. **Controller**: Inserts/removes filters; binds filters to aggregators.

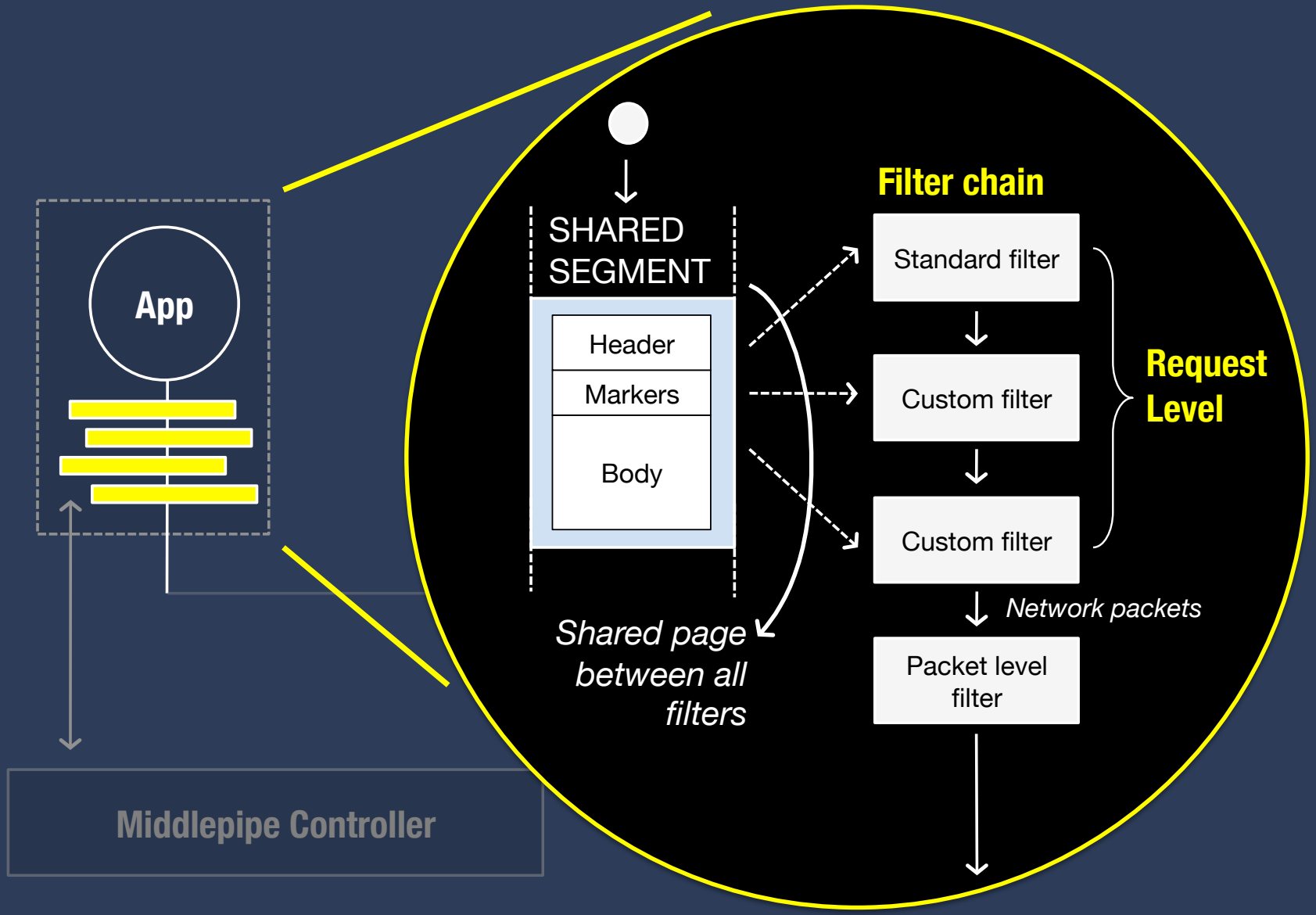
R1. Move Closer to Invocation Path



Why place filters inside App container?

- Naturally distribute computation across the underlying infrastructure
- Reduce overhead on the network substrate
- Minimize copying of requests and packets

R2+3. Chaining Different Filter Types

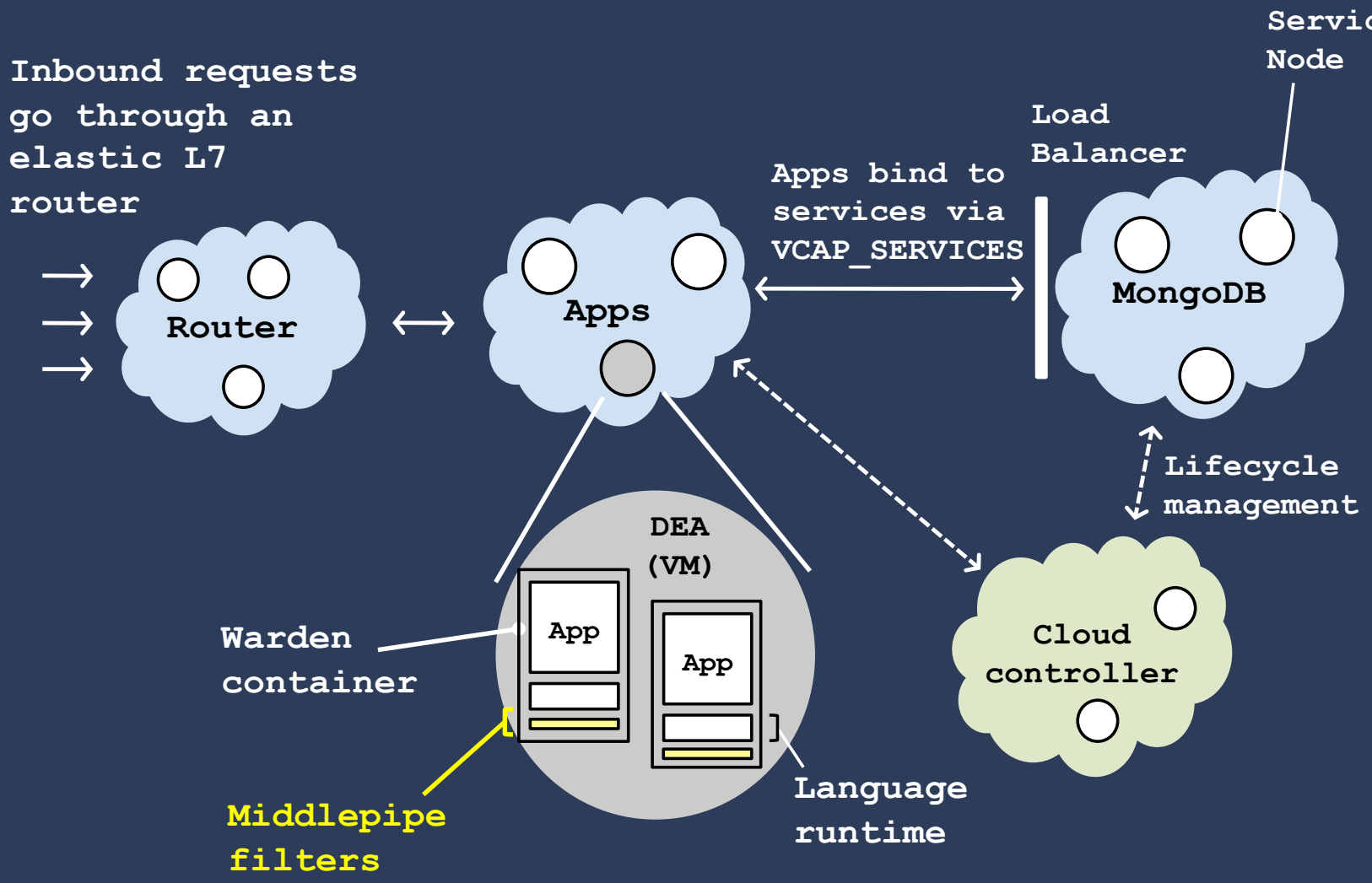


R4. **Supporting Callbacks**

- Thin application library facilitates access to middlepipes
  - Shared memory buffers, etc.

- What if the application needs to be notified?
  - Middlepipes insert “markers” in response
  - Application can look for markers and react (e.g., *library can raise exception*)
  - Other middlepipes can look for markers and react

Embed Inside Cloud Foundry



How to Add Middlepipes

```
$ cf create-middlepipe breaker
```

create instance of middlepipe

```
$ cf bind-middlepipe breaker myapp mongodb
```

bind the “breaker” middlepipe to any communication between my app and mongodb

```
$ cf bind-middlepipe bro myapp mongodb
```

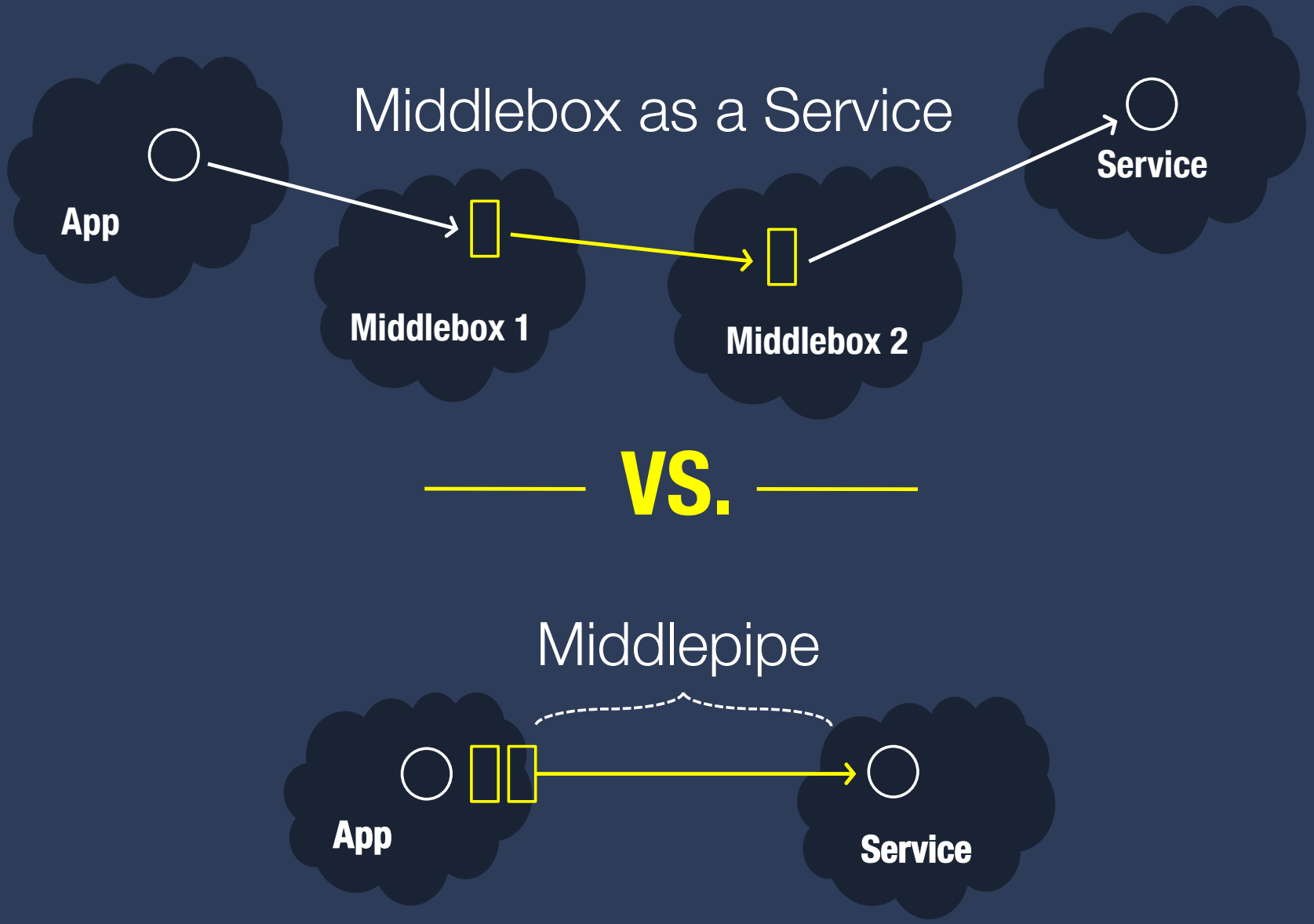
bind the “bro” middlepipe to any communication between my app and mongodb (in addition to the breaker)

Related Work

- **APLOMB** (SIGCOMM'12)
- **CloudNaaS** (SoCC'11)
- **CoMb** (NSDI'12)
- **End to the Middle** (HotOS'09)
- **Split/Merge** (NSDI'13) …

- Emerging of OSS frameworks that focus on "DevOps" lifecycle
  - *e.g., Netflix OSS, Airbnb, Etsy, etc.*
  - Canary testing, Circuit Breaker, Stress testing

Summary


Middlebox as a Service
App
Middlebox 1
Middlebox 2
Service
VS.
Middlepipe
App
Service