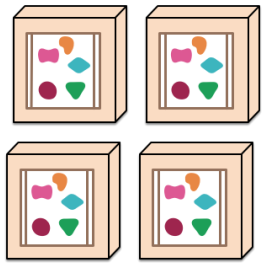
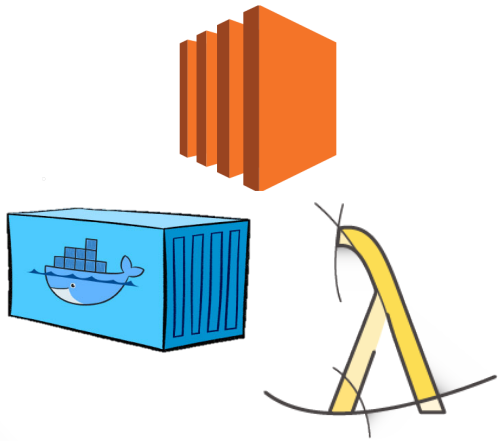
Towards Omnia: a Monitoring Factory for Quality-Aware DevOps

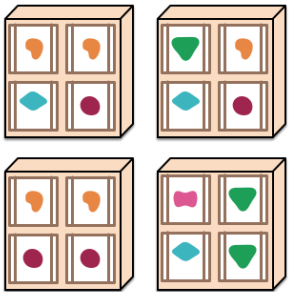
Apr 27th, 2017

Marco MIGLIERINA
Damian A. TAMBURRI

Dev- to Ops: We are moving faster and faster





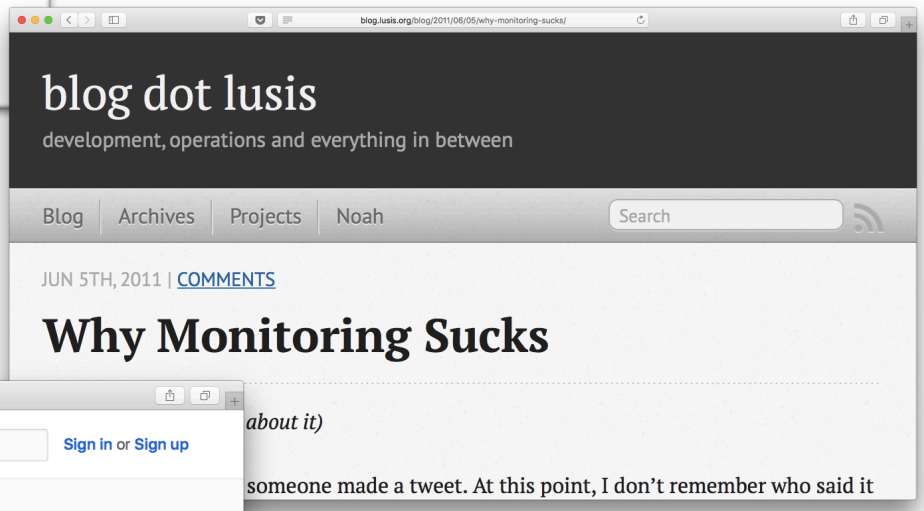


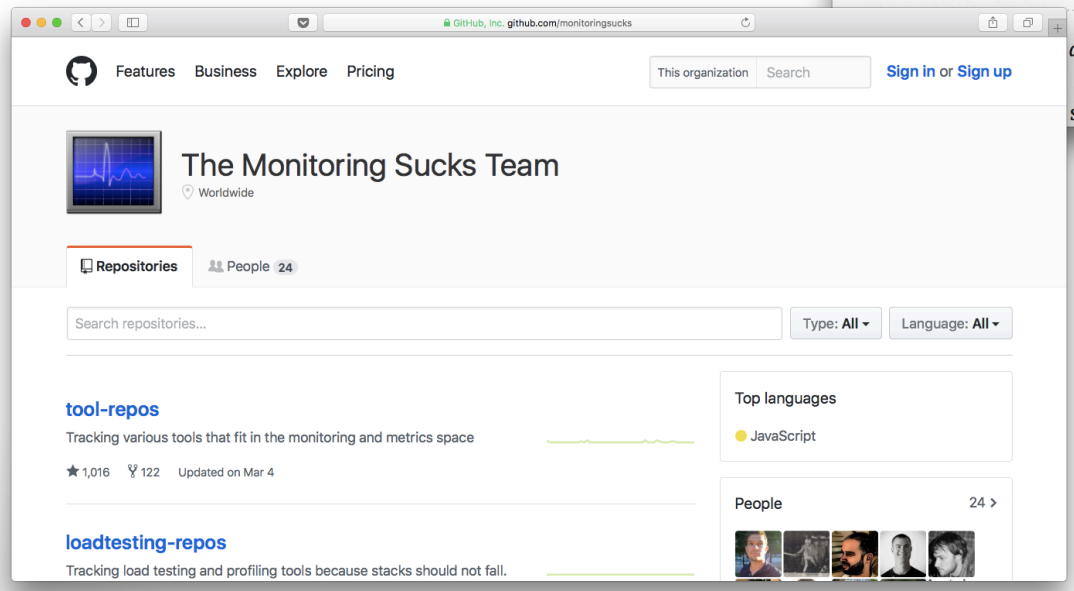
Figures from: http://martinfowler.com/articles/microservices.html and https://www.slideshare.net/CiscoDevNet/enabing-devops-in-an-sdn-world

Observability is essential

Monitoring lagging behind

…back in 2011





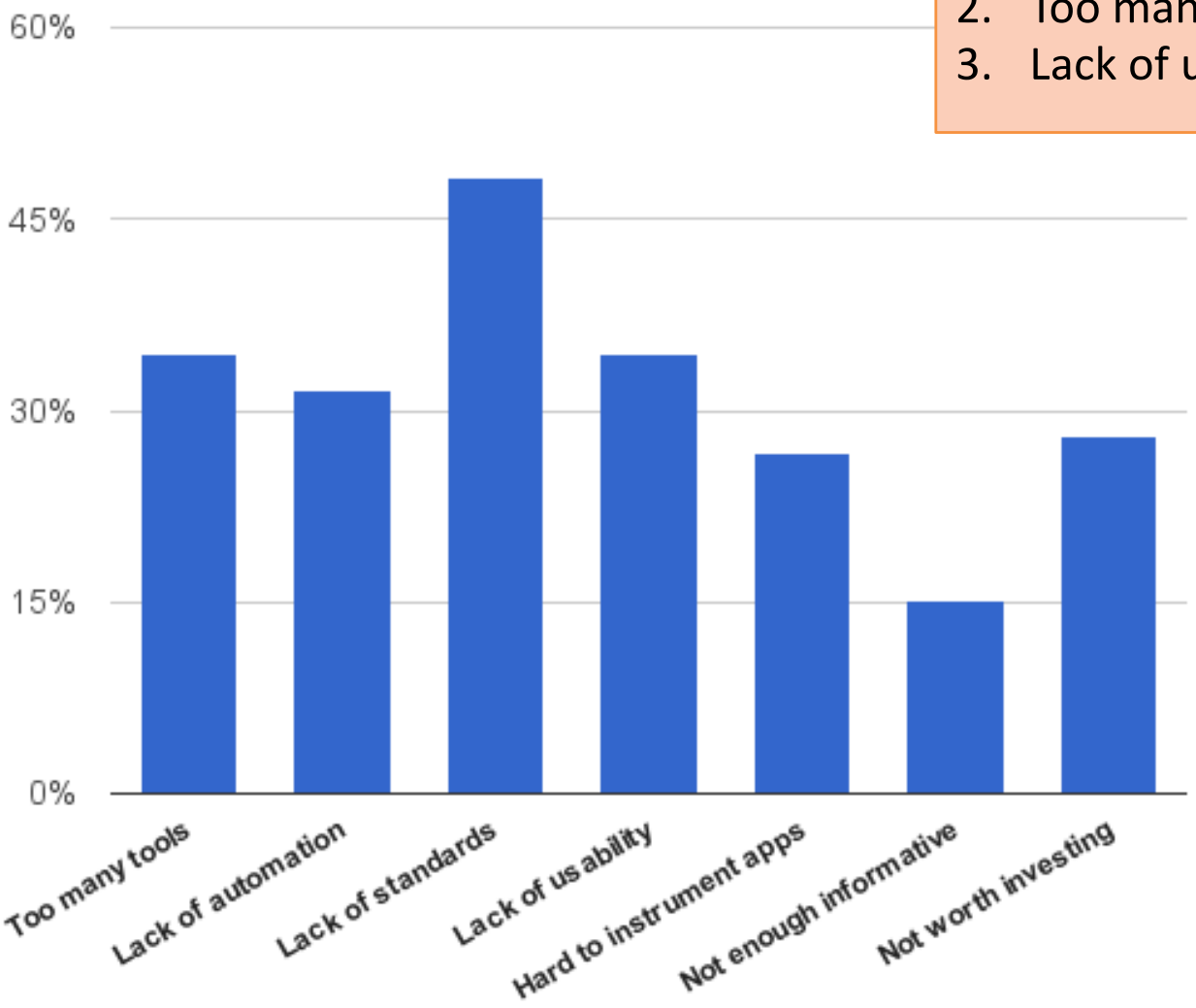
Since then… proliferation of tools and solutions

144 practitioners surveyed

M. Miglierina. Monitoring Modern Distributed Software Applications: Challenges And Solutions. PhD thesis, Politecnico di Milano, 2017.

Main perceived drawback in monitoring?

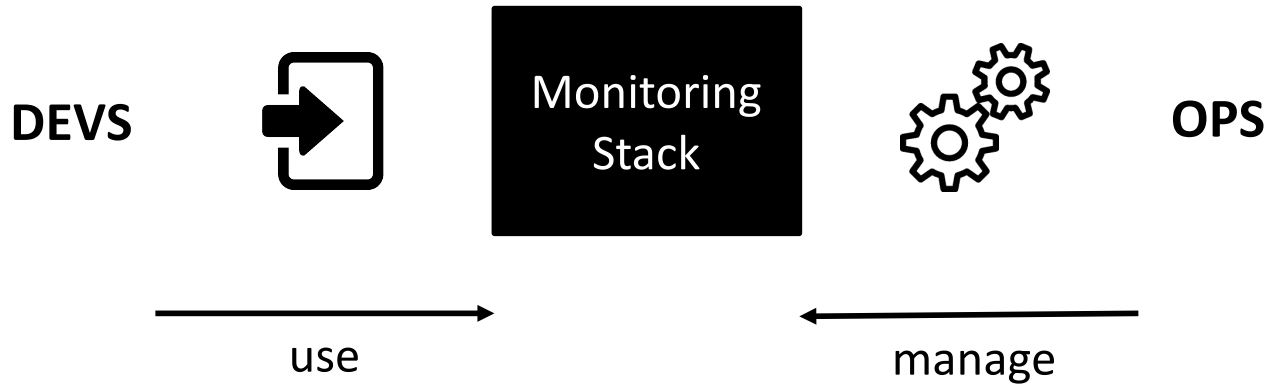
1. Lack of standards
2. Too many tools
3. Lack of usability



M. Miglierina. Monitoring Modern Distributed Software Applications: Challenges And Soloutions. PhD thesis, Politecnico di Milano, 2017.

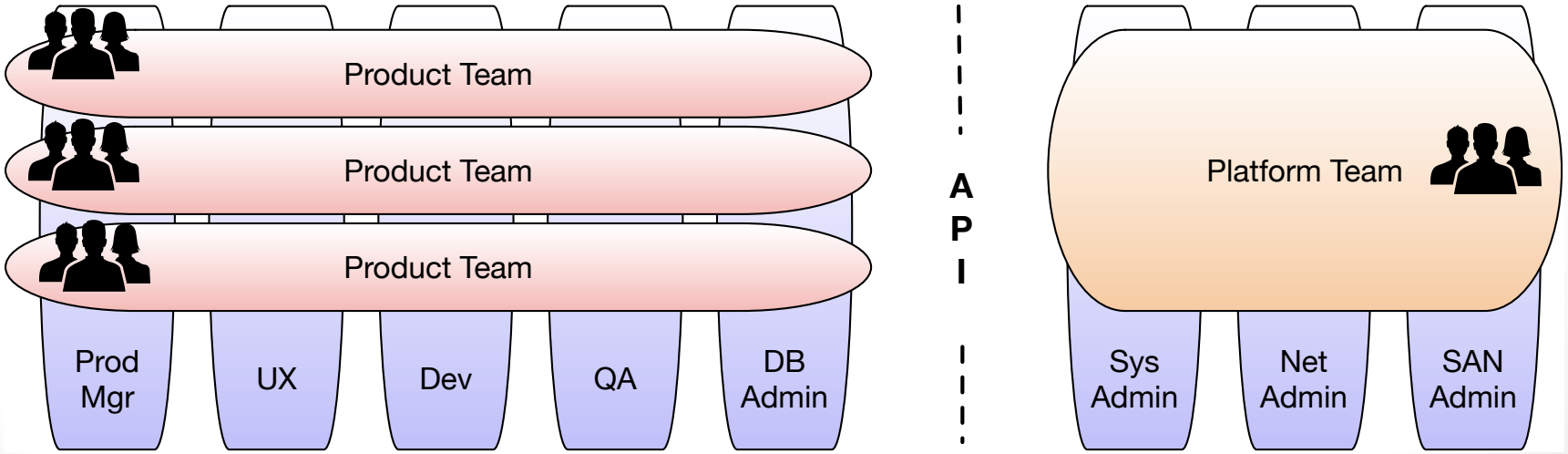
Omnia

- **Main objectives**
  - reduce learning curve and entry cost to monitoring
  - an attempt of standardization
- **How?**
  - One, interoperable, self-service monitoring interface for devs
  - A simple monitoring factory for ops



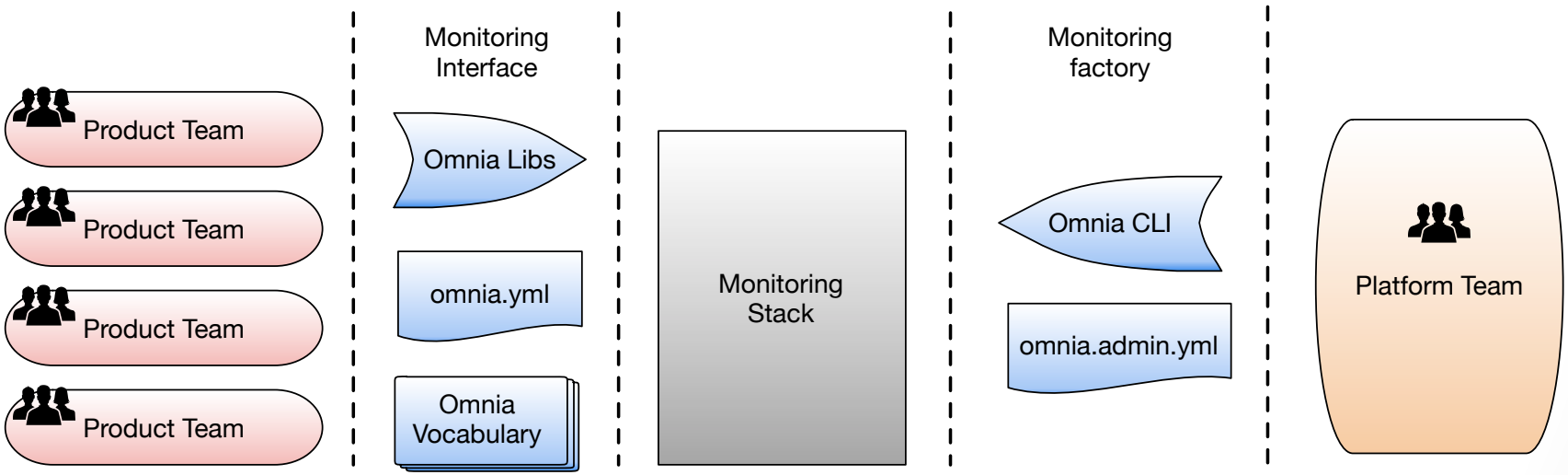
Reference team organization

- Product teams
  - each responsible of its microservice
  - independent workflows
- Platform team
  - provide infrastructure support
  - cross functional wrt product teams

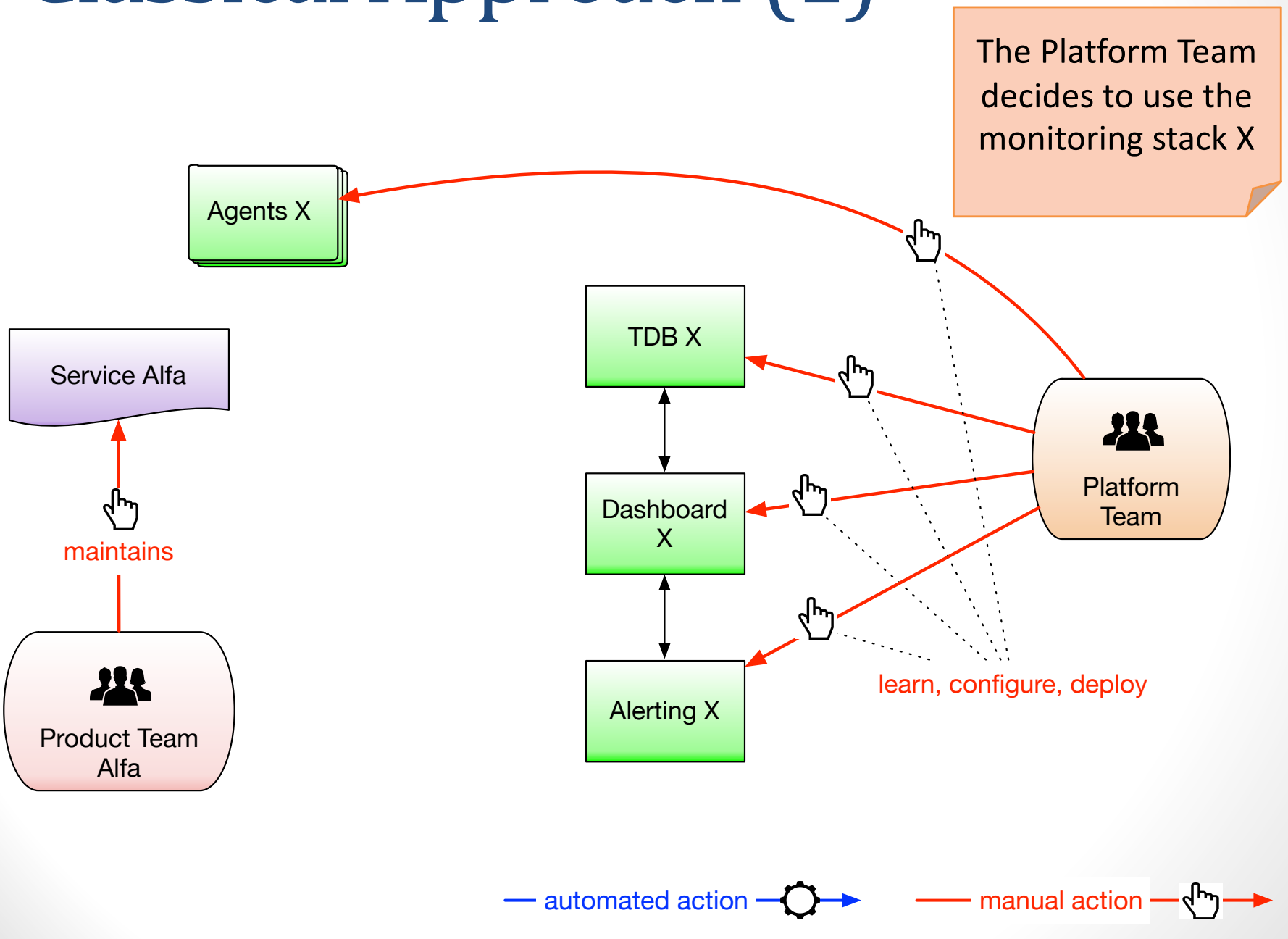


Team organzation at Netflix: https://www.nginx.com/blog/adopting-microservices- at-netflix-lessons-for-team-and-process-design/

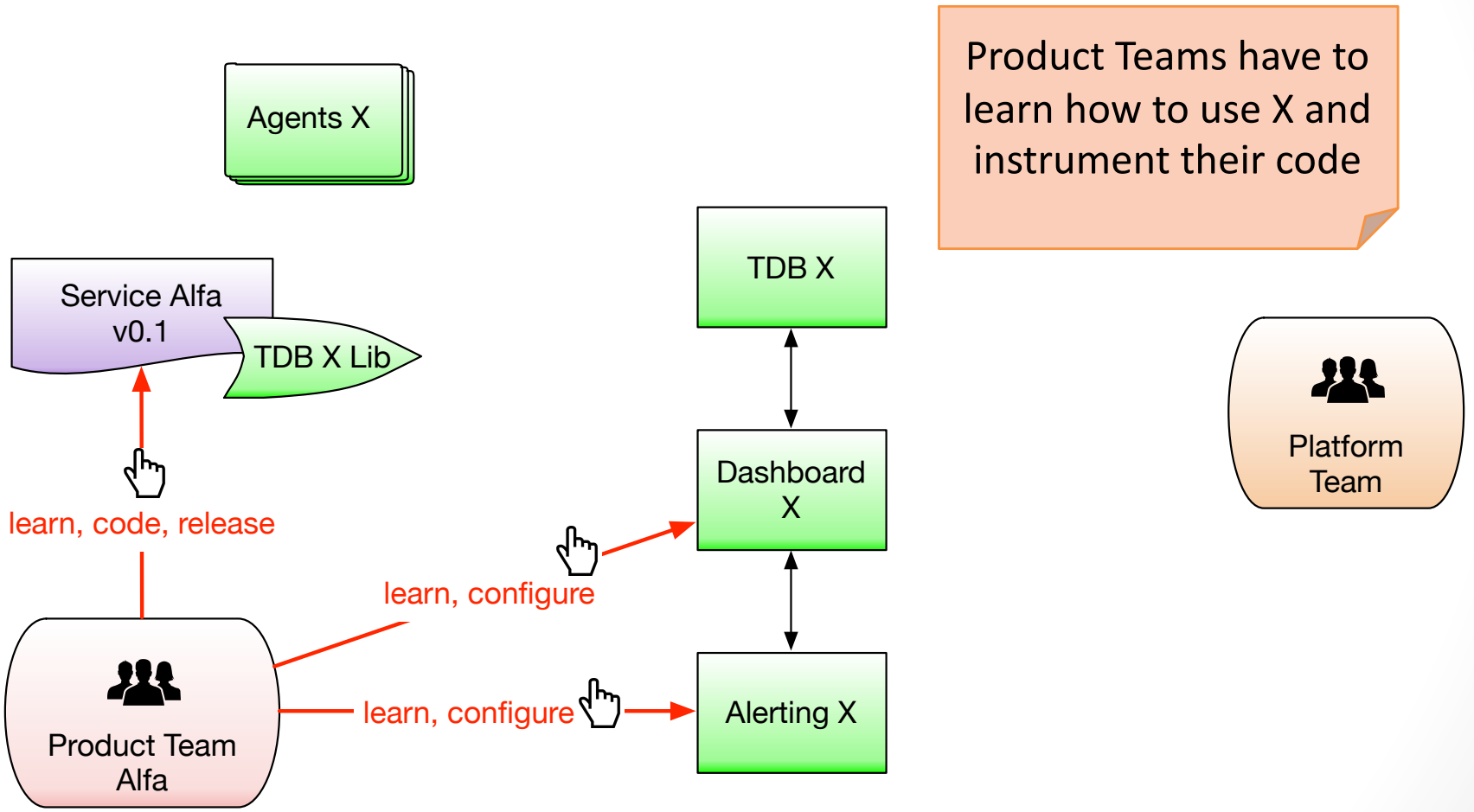
The Omnia components



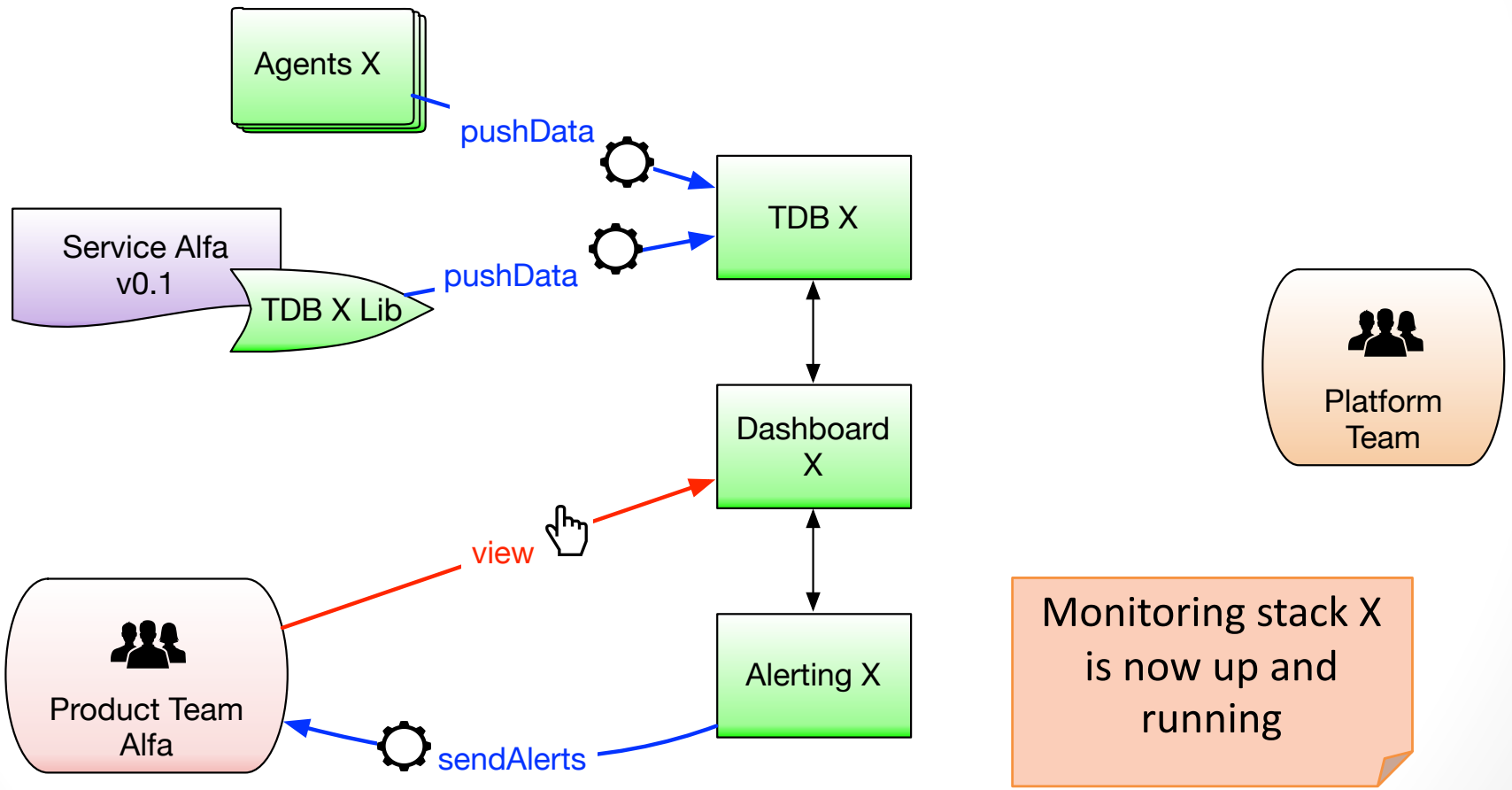
Classical Approach (1)



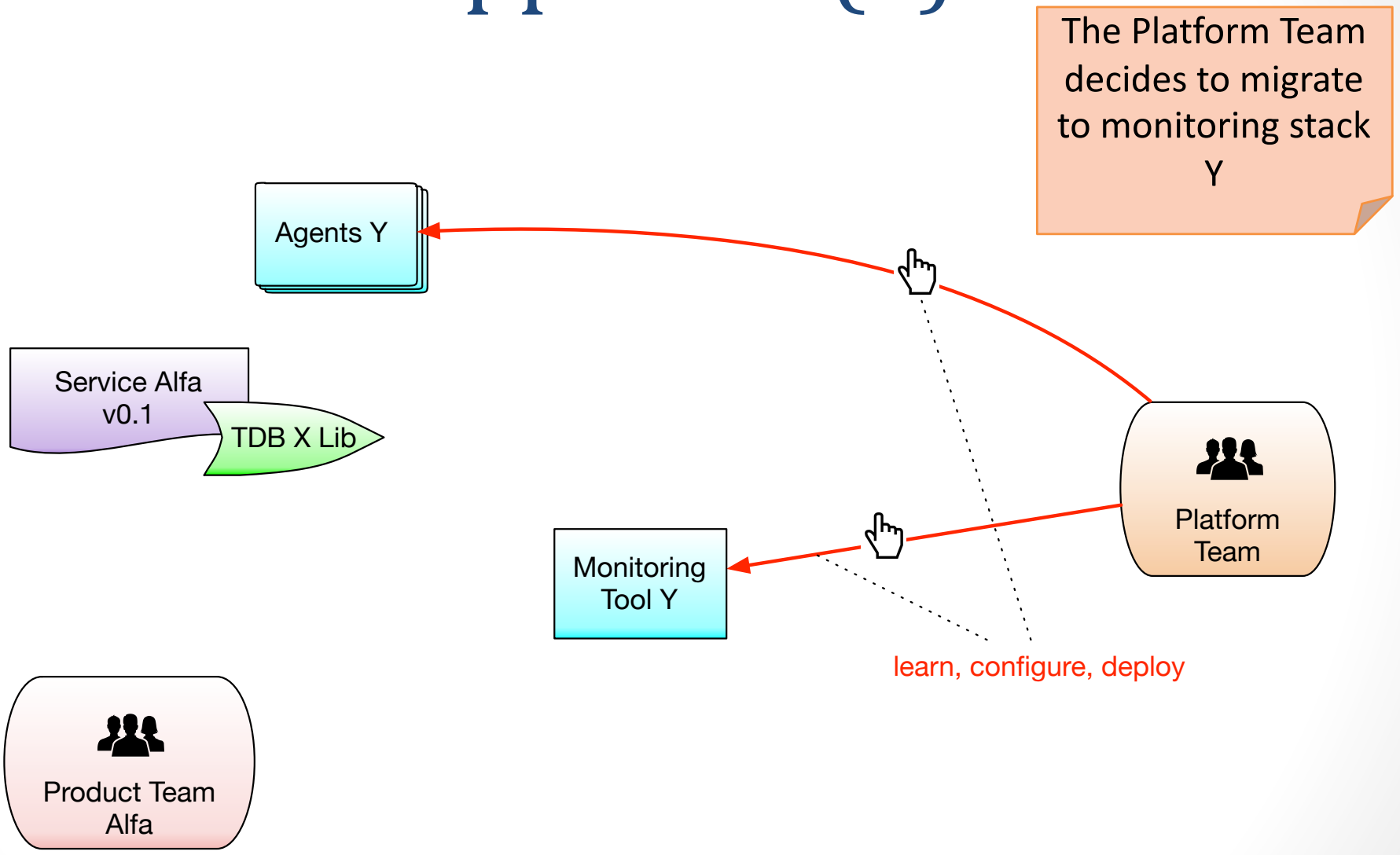
Classical Approach (2)



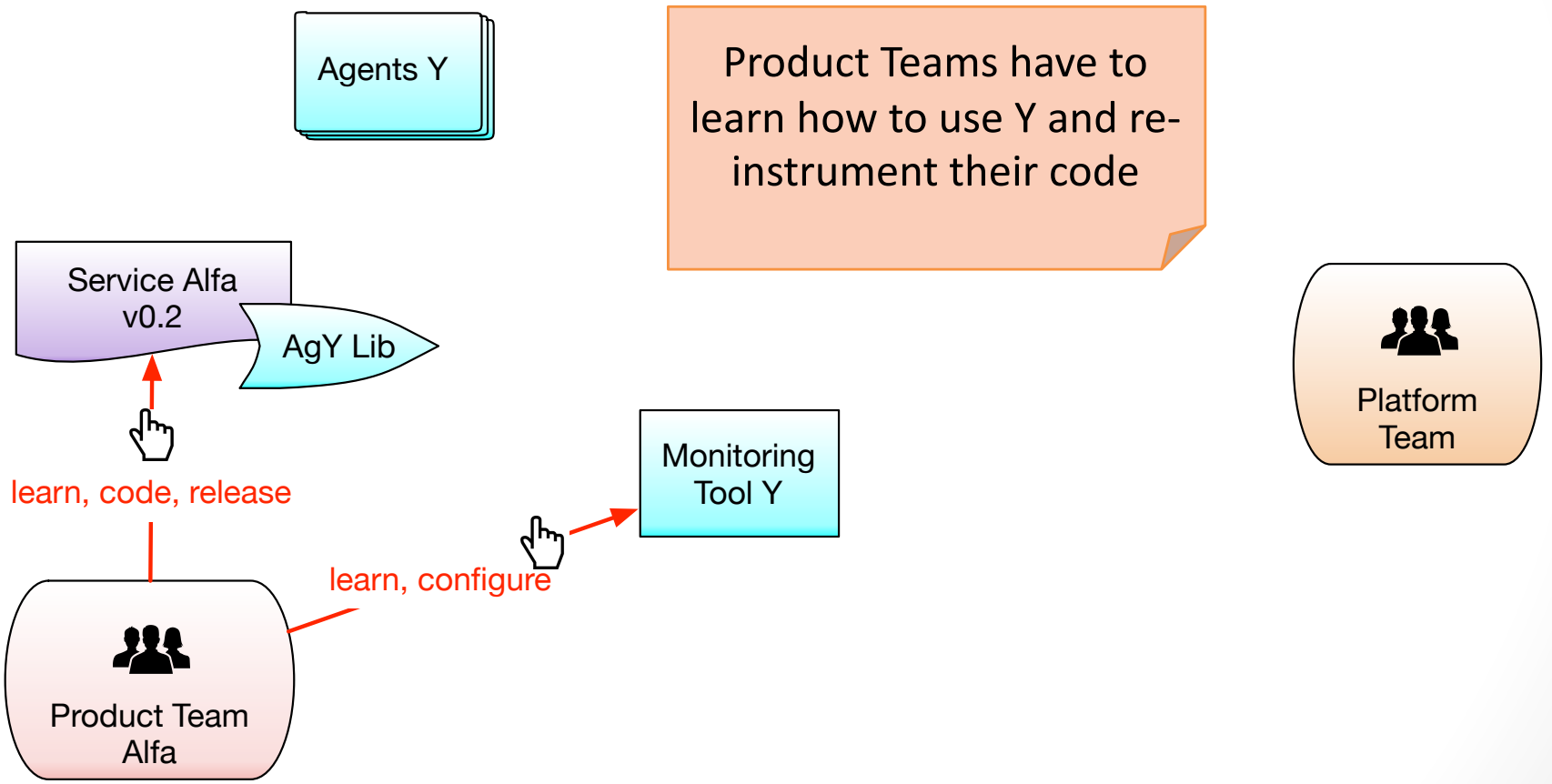
Classical Approach (3)



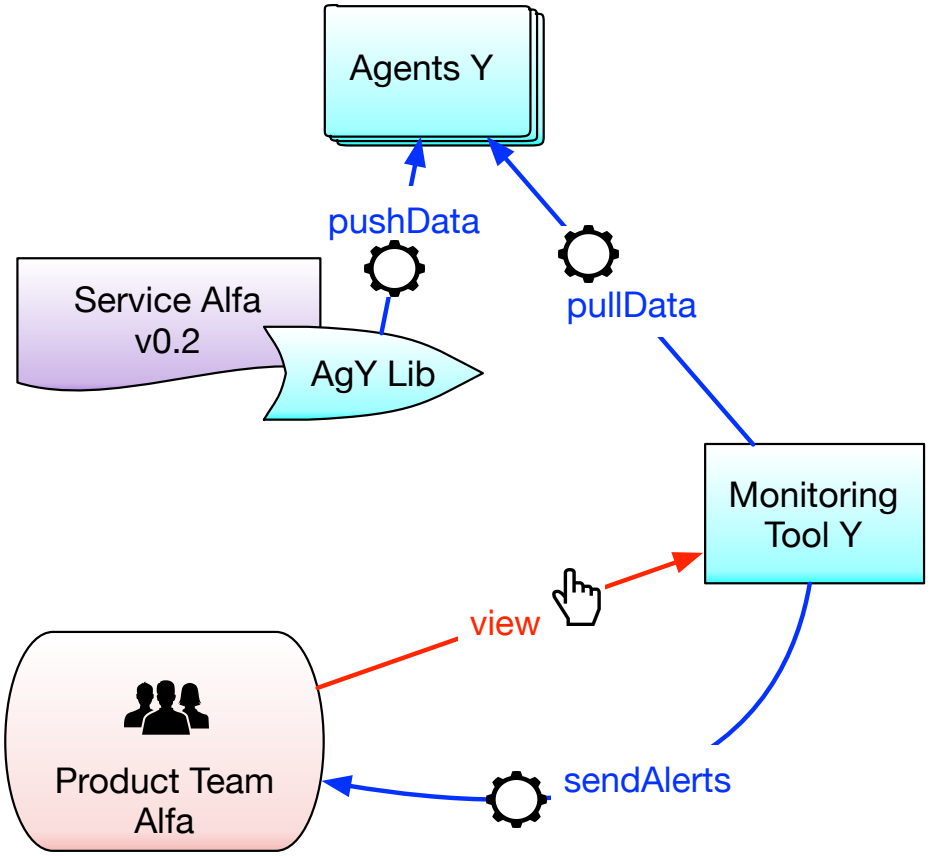
Classical Approach (4)

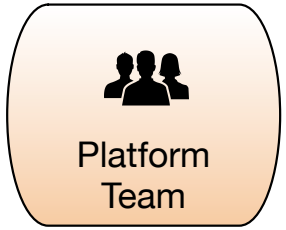


Classical Approach (5)



Classical Approach (6)





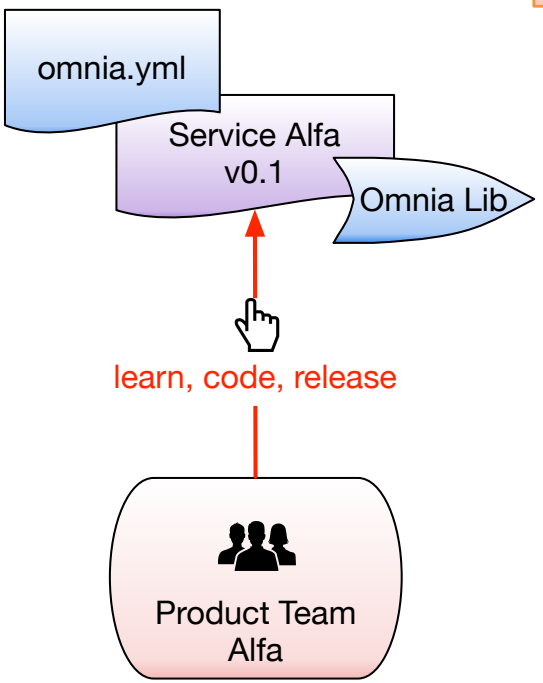
Monitoring stack Y is now up and running

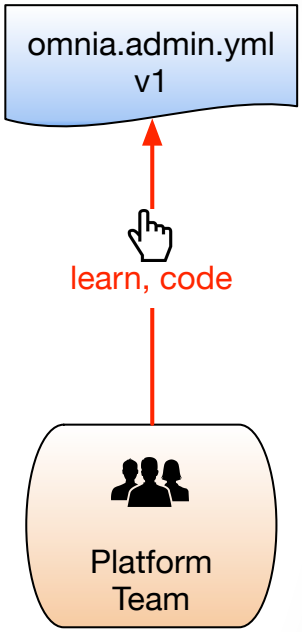
automated action

manual action

Omnia-based approach (1)

By using Omnia a single learning step is required for all teams

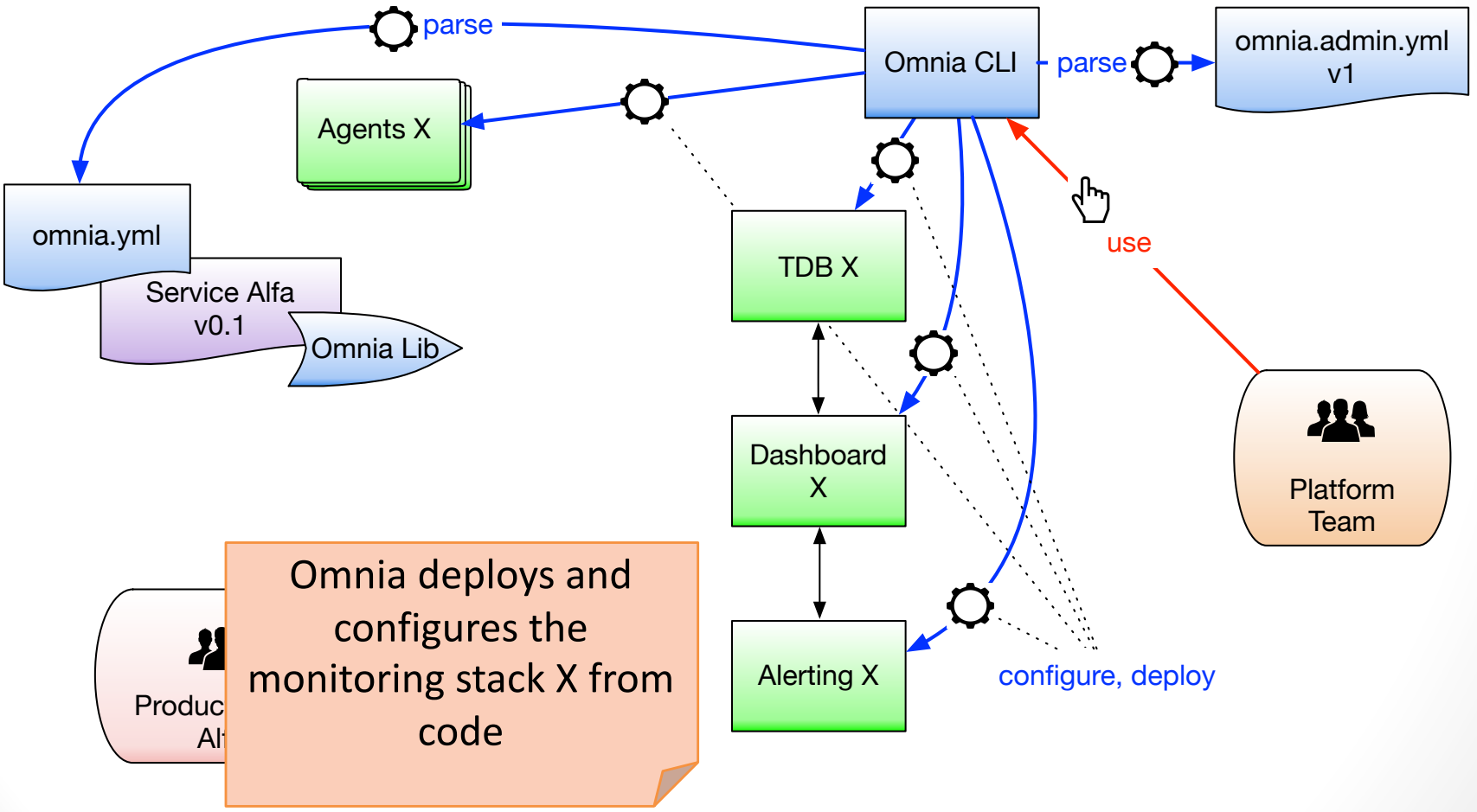




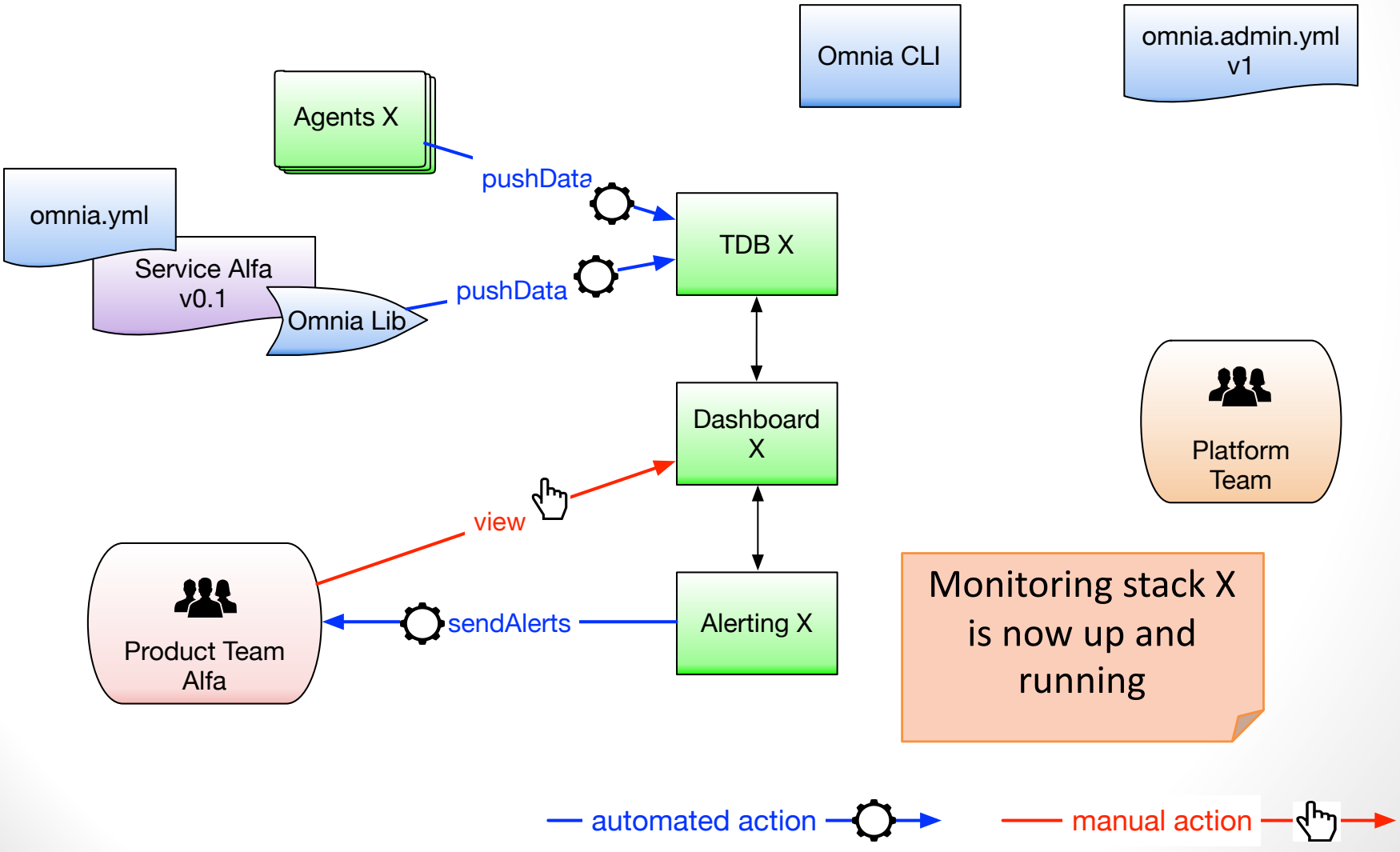
automated action

manual action

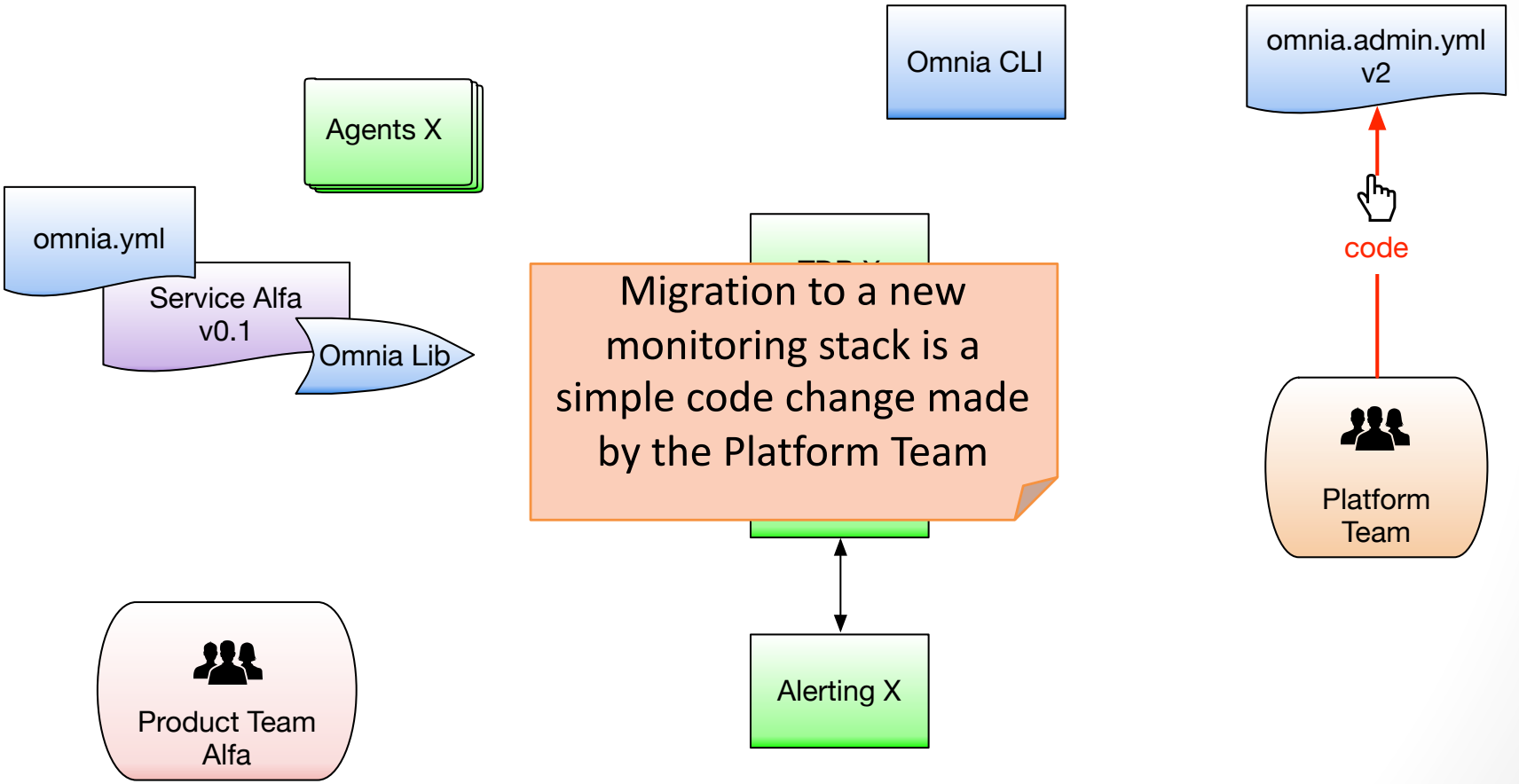
Omnia-based approach (2)



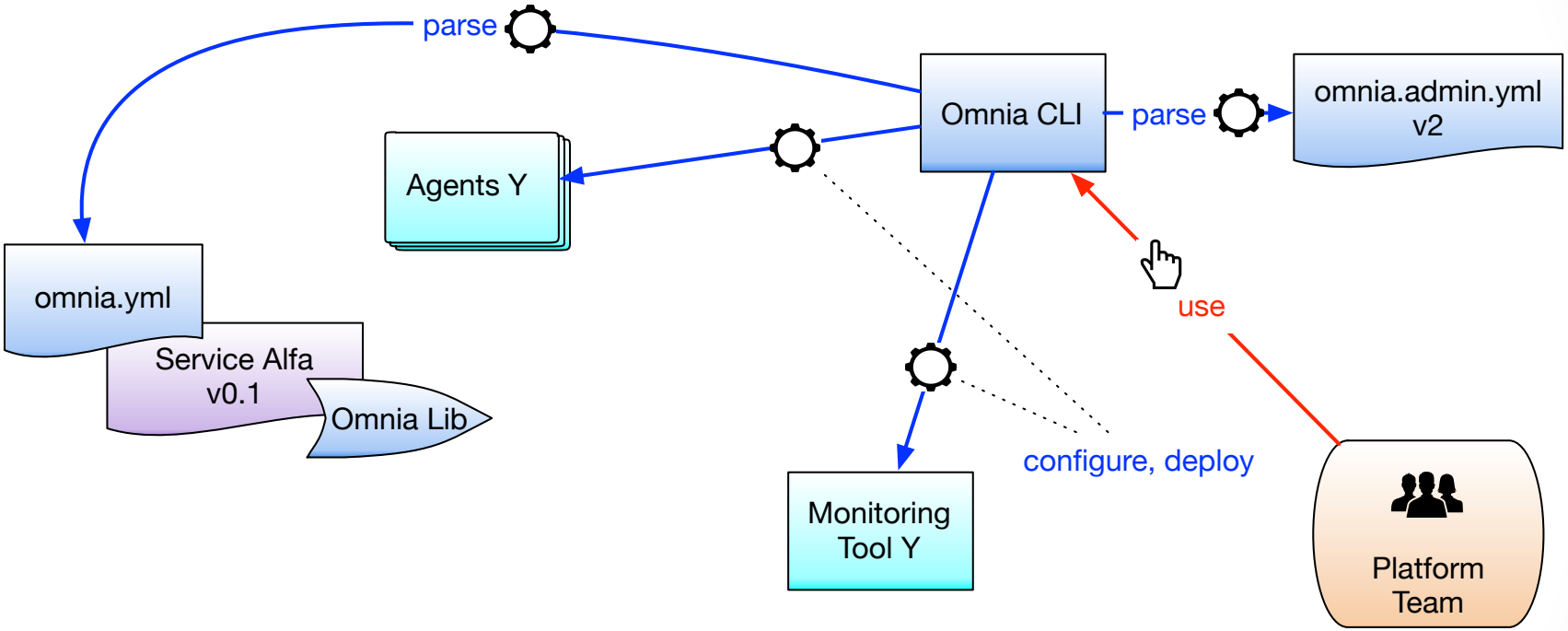
Omnia-based approach (3)



Omnia-based approach (4)



Omnia-based approach (5)

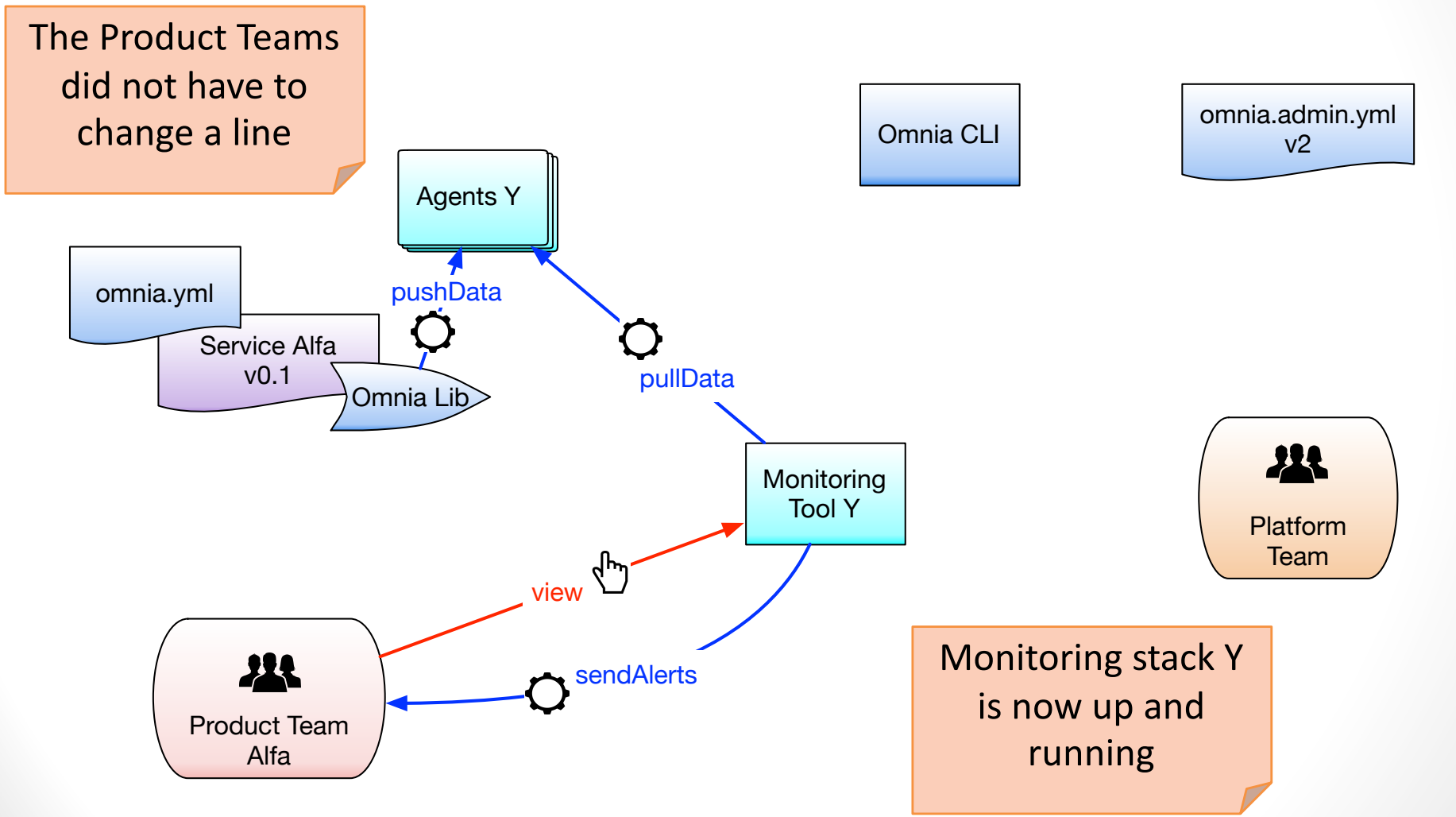


Omnia takes care of reconfiguring and redeploying the stack

automated action

manual action

Omnia-based approach (6)



Omnia Libs

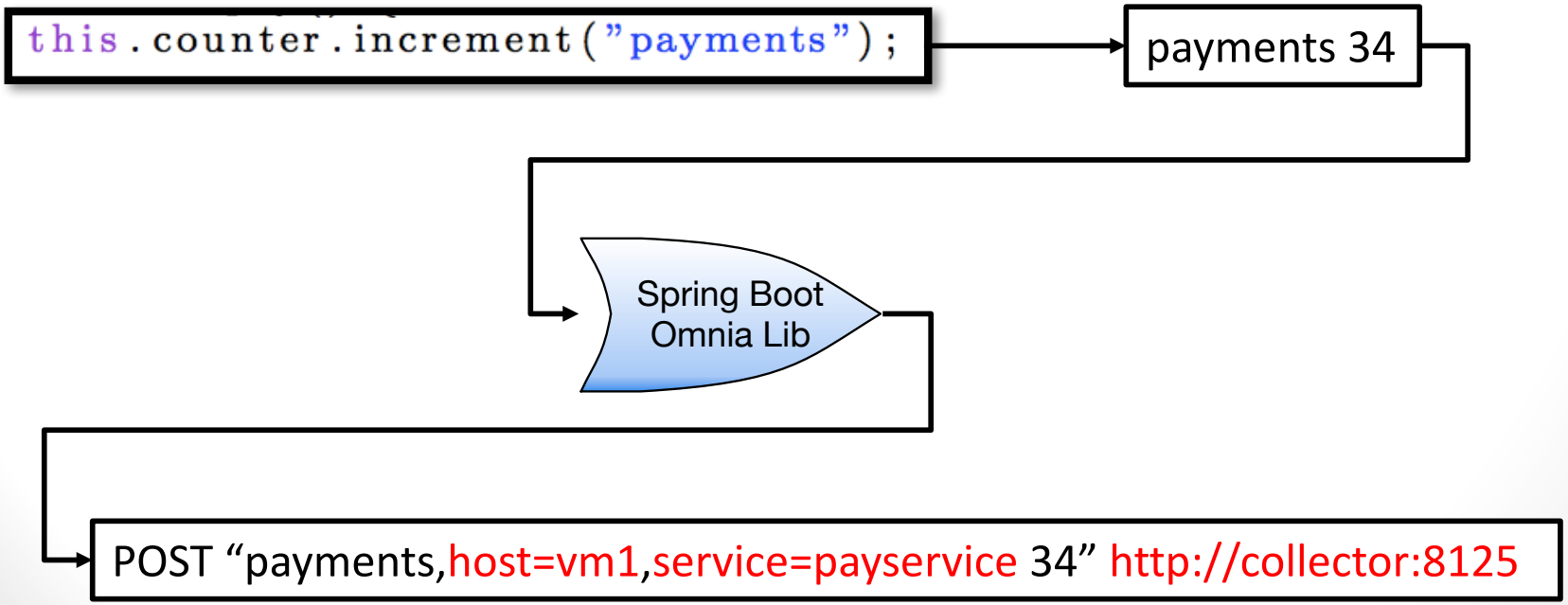
- Requirements:
  - independent of the monitoring stack
  - minimize instrumentation overhead
- Example:

```
@Service
public class MyService {
    private final CounterService counter;

    @Autowired
    public MyService(CounterService counter) {
        this.counter = counter;
    }

    public void pay() {
        // perform the payment
        ...

        // monitoring
        this.counter.increment("payments");
    }
```

Omnia Lib for Spring Boot

Under the hood

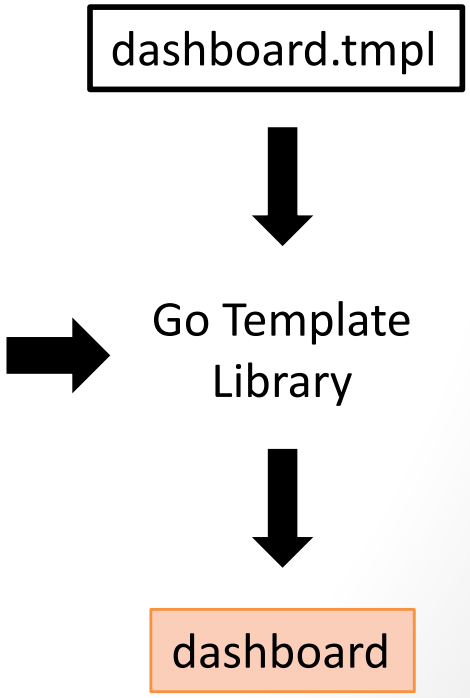
- Convention over configuration:
  - statsd (w/ influxdb tag ext.) protocol
  - automated meta-data decoration
  - default endpoint
- Example:



Omnia.yml: monitoring config as code

- Requirements
  - independent of the monitoring stack
  - versionable with the code
- Example:

```
dashboard:
  timeseries:
    - metric: payments
      compute: rate
    - metric: heap_memory_usage
      compute: average by host
    - metric: cpu_usage_user
      compute: average by host
    - metric: mem_used
      compute: average by host
actions:
  email:
    - condition: cpu_usage_user > 0.8
```



The Omnia vocabulary

- Shared vocabulary for resources

Resource	Description
host	a physical or virtual machine
service	an application
service_id	a unique identifier for an instance of an application
container_id	a unique identifier for a Linux container
container_image	a Linux container image

- and for metrics

Host metrics	Java metrics
cpu_usage_user	*heap_memory_usage*
cpu_usage_system	*thread_count*
cpu_usage_idle	*loaded_class_count*
mem_used	*garbage_collection_time*
mem_used_percent	*thread_count*

Monitoring infra as code

omnia.admin.yml example:

```
provisioner:
  name: docker
  args:
    username: mmiglier
    images_tag: latest
tools:
  telegraf:
    roles:
      - agent
      - collector
    pushes_to:
      - influxdb
  influxdb:
    roles:
      - tdb
  grafana:
    pulls_fr
      - infl
    roles:
      - das
      - act
teams_repos:
  - "github.com/mmiglier/service
  - "github.com/mmiglier/service
```

Reusing existing adapters

v2

```
provisioner:
  name: docker
  args:
    username:
    images_tag
tools:
  collectd:
    roles:
      - collector
      - agent
    pushes_to:
      - collectd_exporter
  collectd_exporter:
    roles:
      - agent
  prometheus:
    pulls_from
      - collec
    roles:
      - tdb
      - actior
  grafana:
    pulls_from:
      - prometheus
    roles:
      - dashboard
teams_repos:
```

Reusing existing monitoring tools

Reusing existing provisioning tools

Conclusion

- Contributions
  - Reduce entry cost and learning curve to monitoring
  - Application of DevOps practices to monitoring
  - Attempt of standardization
- Threats to validity
  - A common interface may simplify tools characteristics
  - However:
    - initial approach to monitoring has simple requirements
    - it could push tool vendors to implement missing features
    - the approach could be applied to existing or new tools
- Future work
  - Extensive evaluation using real world examples