

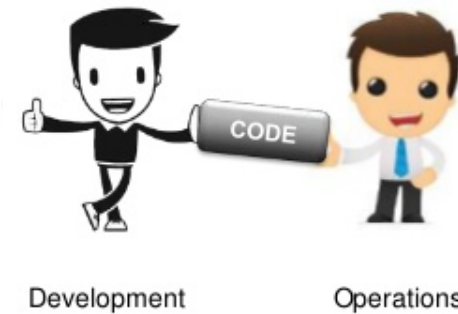
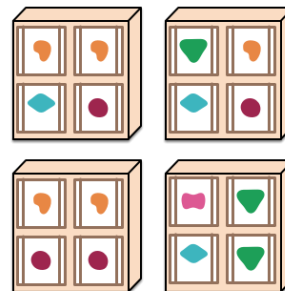
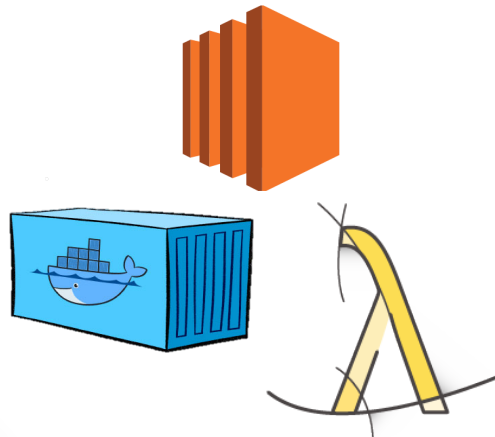
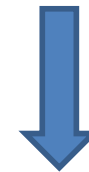
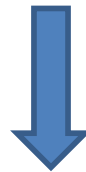
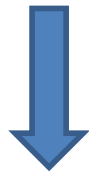
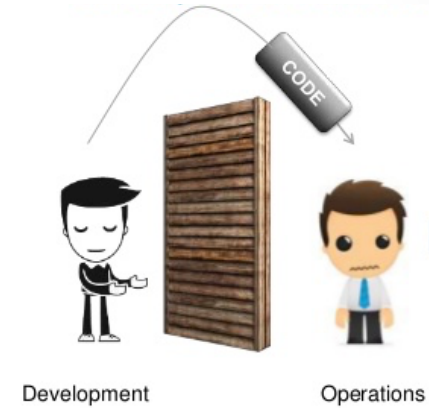
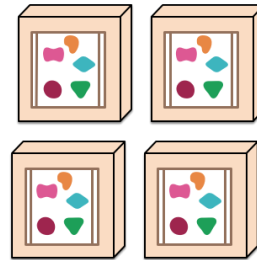


Towards Omnia: a Monitoring Factory for Quality-Aware DevOps

Apr 27th, 2017

Marco MIGLIERINA
Damian A. TAMBURRI

Dev- to Ops: We are moving faster and faster



Observability is essential



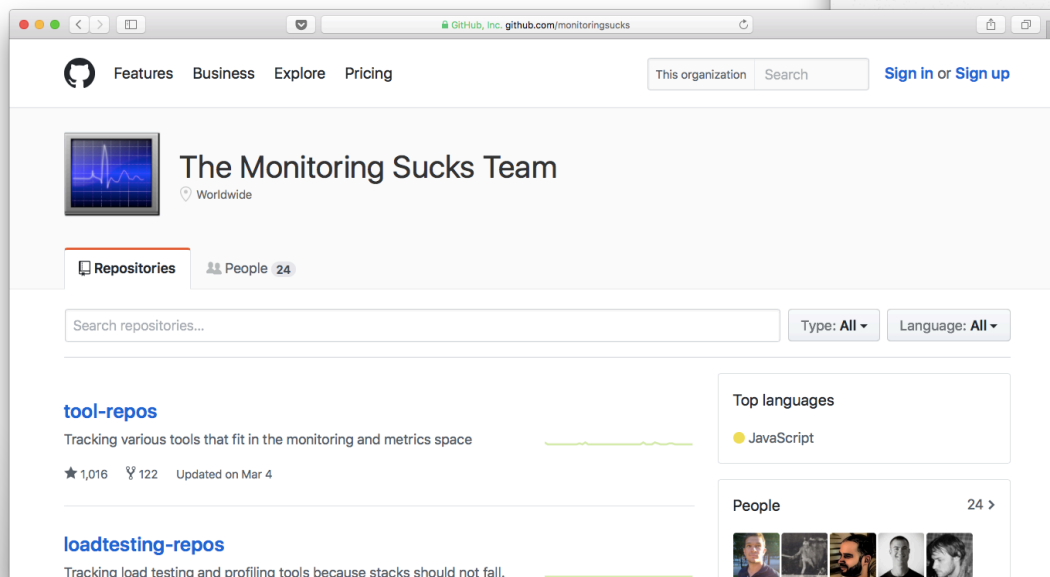
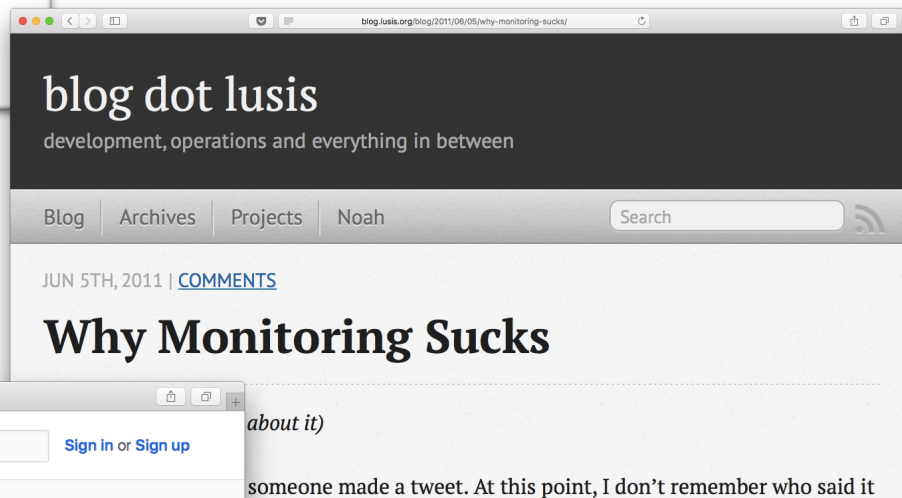
Monitoring lagging behind



@portertech
Sean Porter

Nagios sucks! We all put up w/ it. Good thing we have CM to make it bearable.
[#devops](#) [#chef](#) [#puppet](#)

...back in 2011



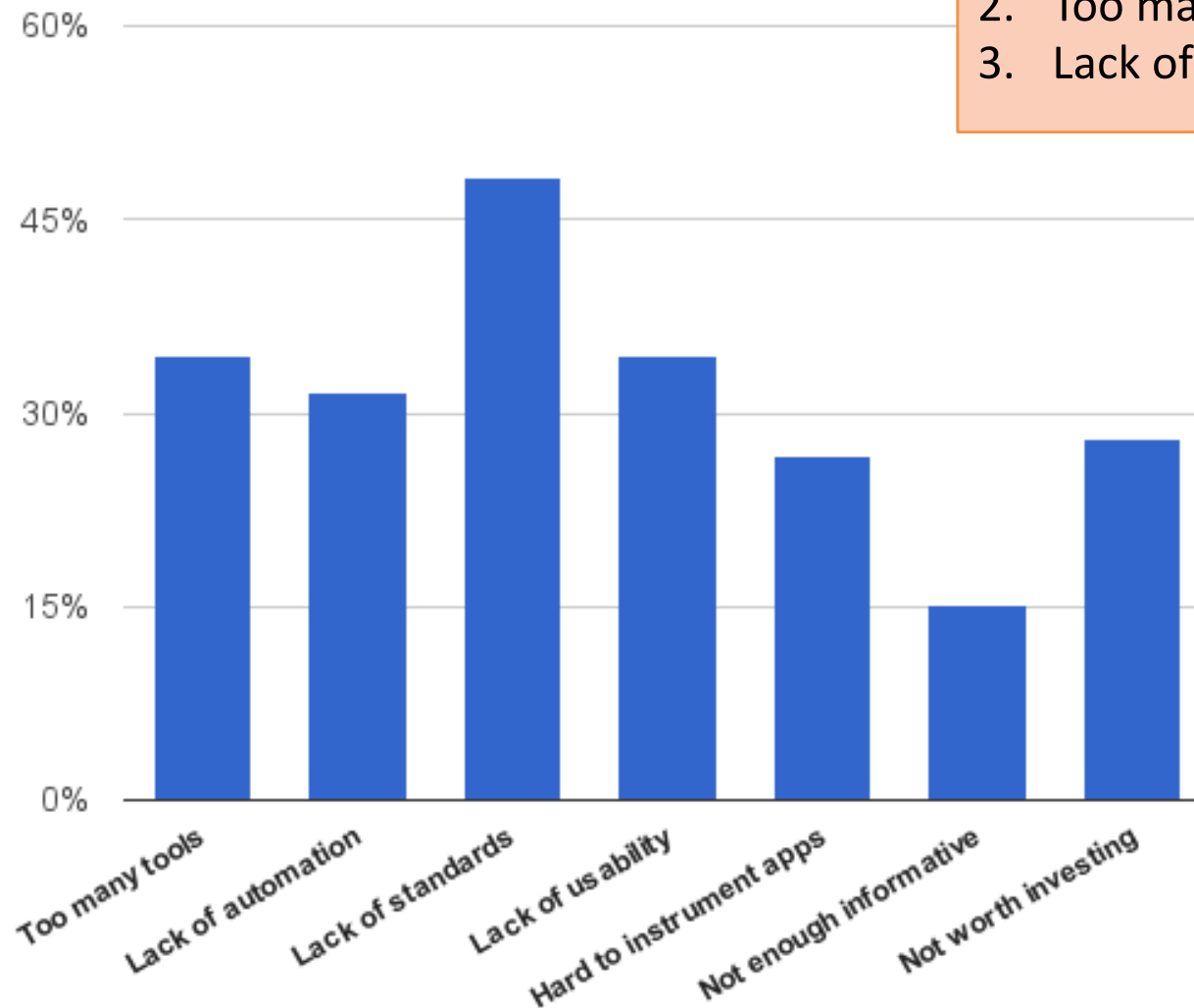
Since then... proliferation of tools and solutions

144 practitioners surveyed



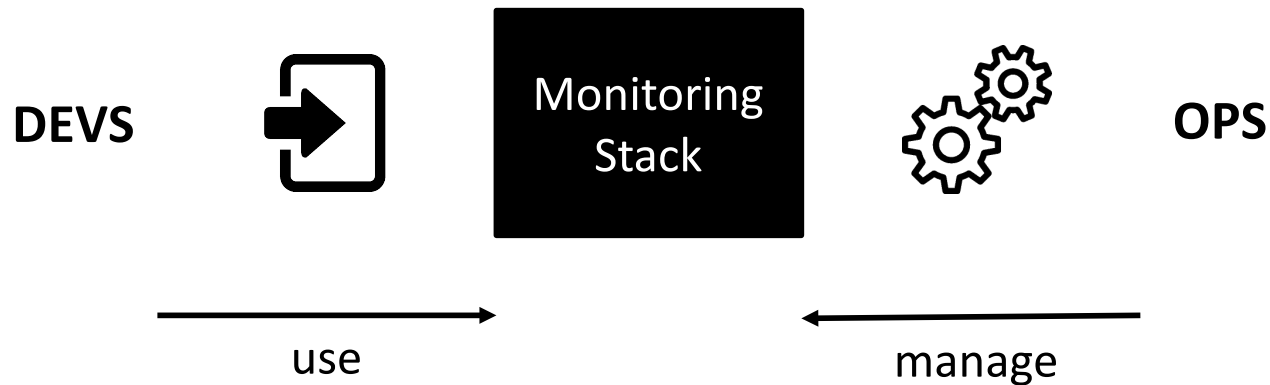
Main perceived drawback in monitoring?

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



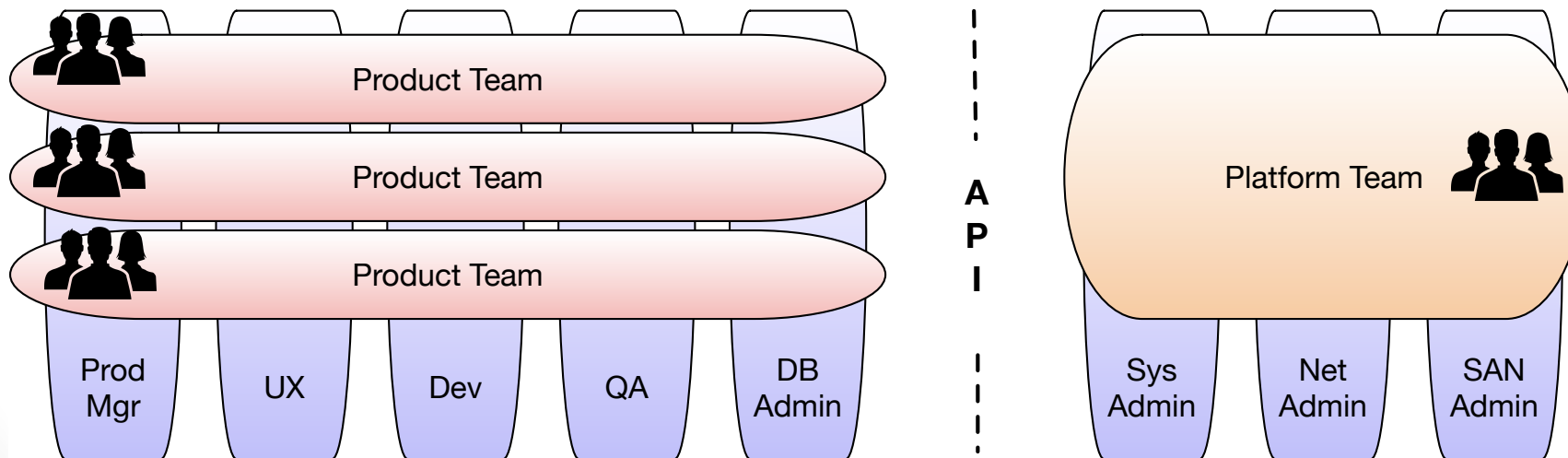
Omnia

- **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

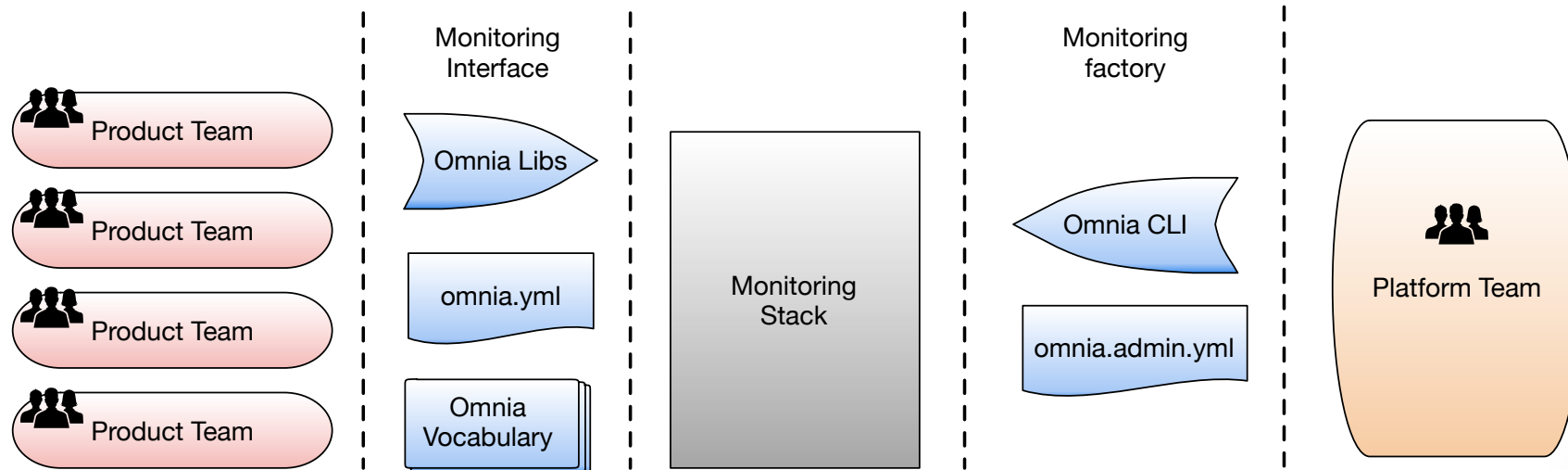


Reference team organization

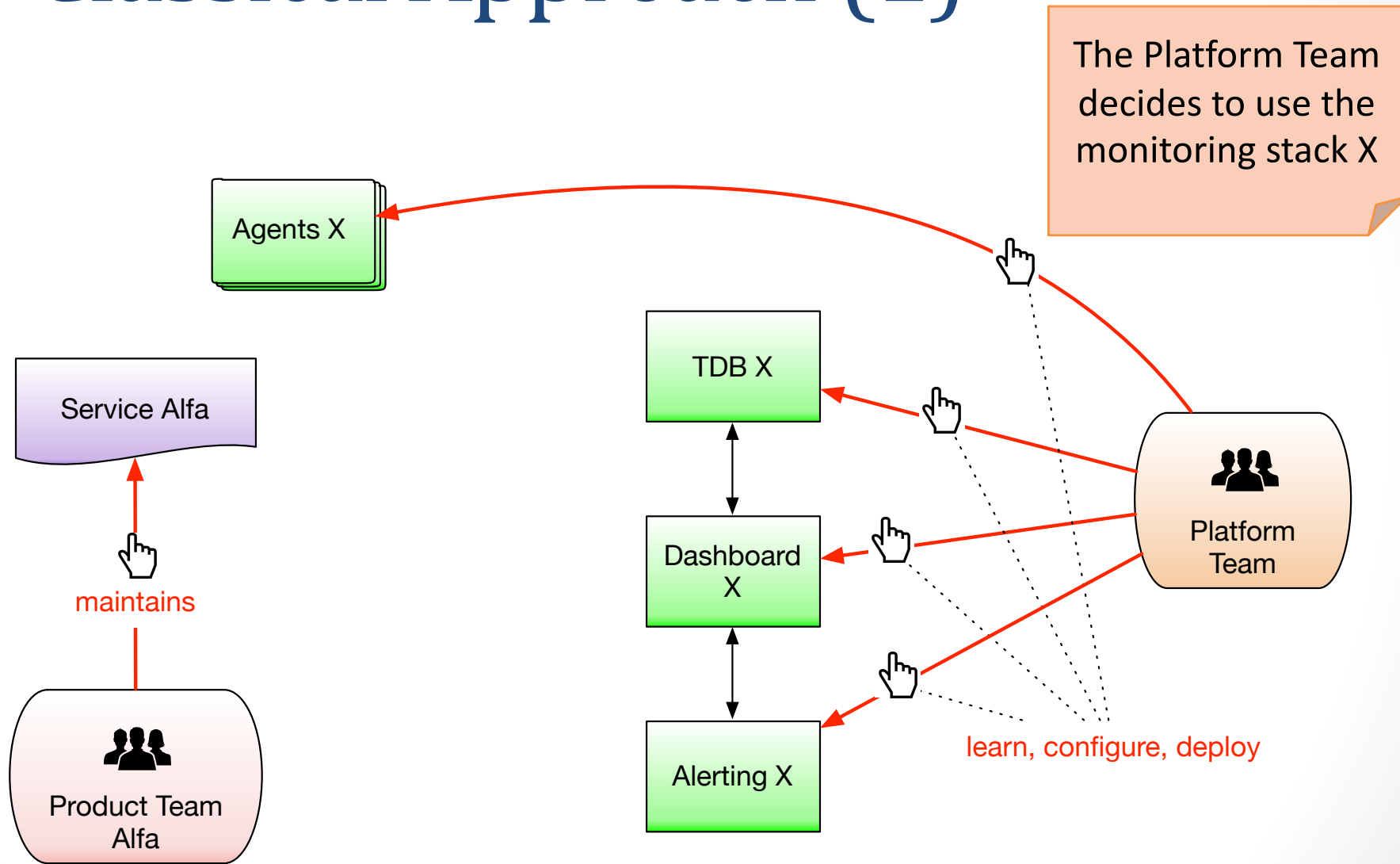
- Product teams
 - each responsible of its microservice
 - independent workflows
- Platform team
 - provide infrastructure support
 - cross functional wrt product teams



The Omnia components



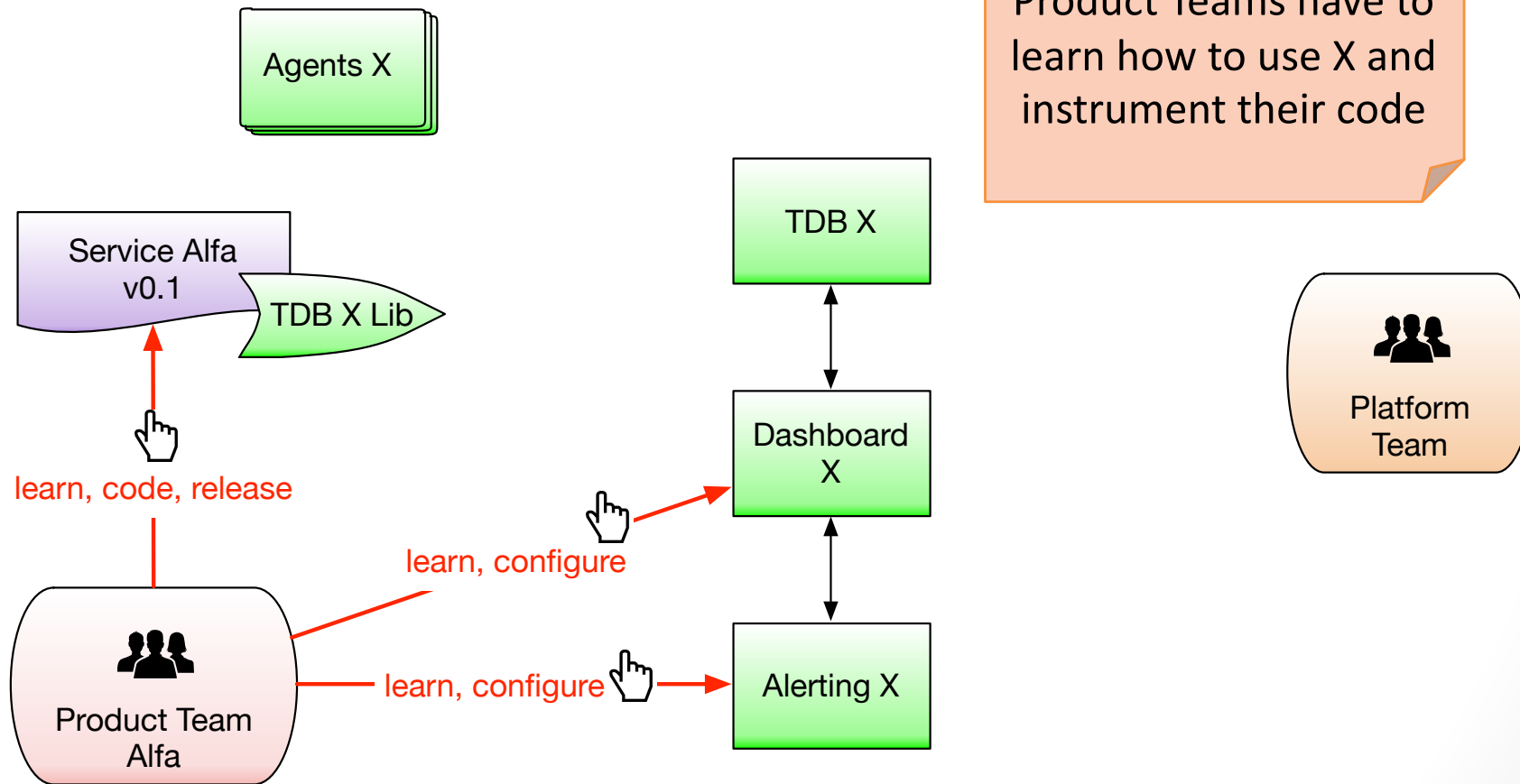
Classical Approach (1)



— automated action —  —

— manual action —  —

Classical Approach (2)

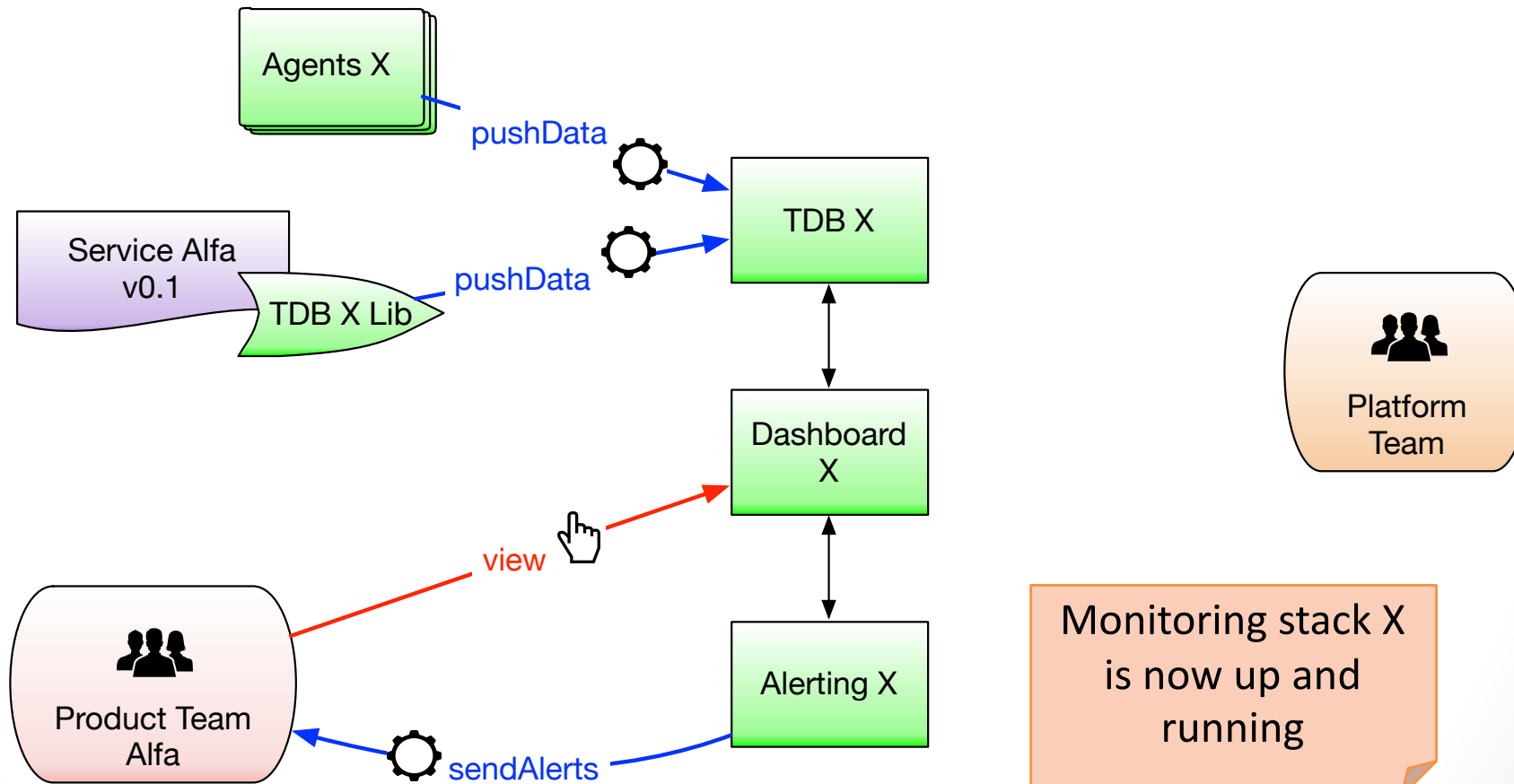


Product Teams have to learn how to use X and instrument their code

— automated action —  —

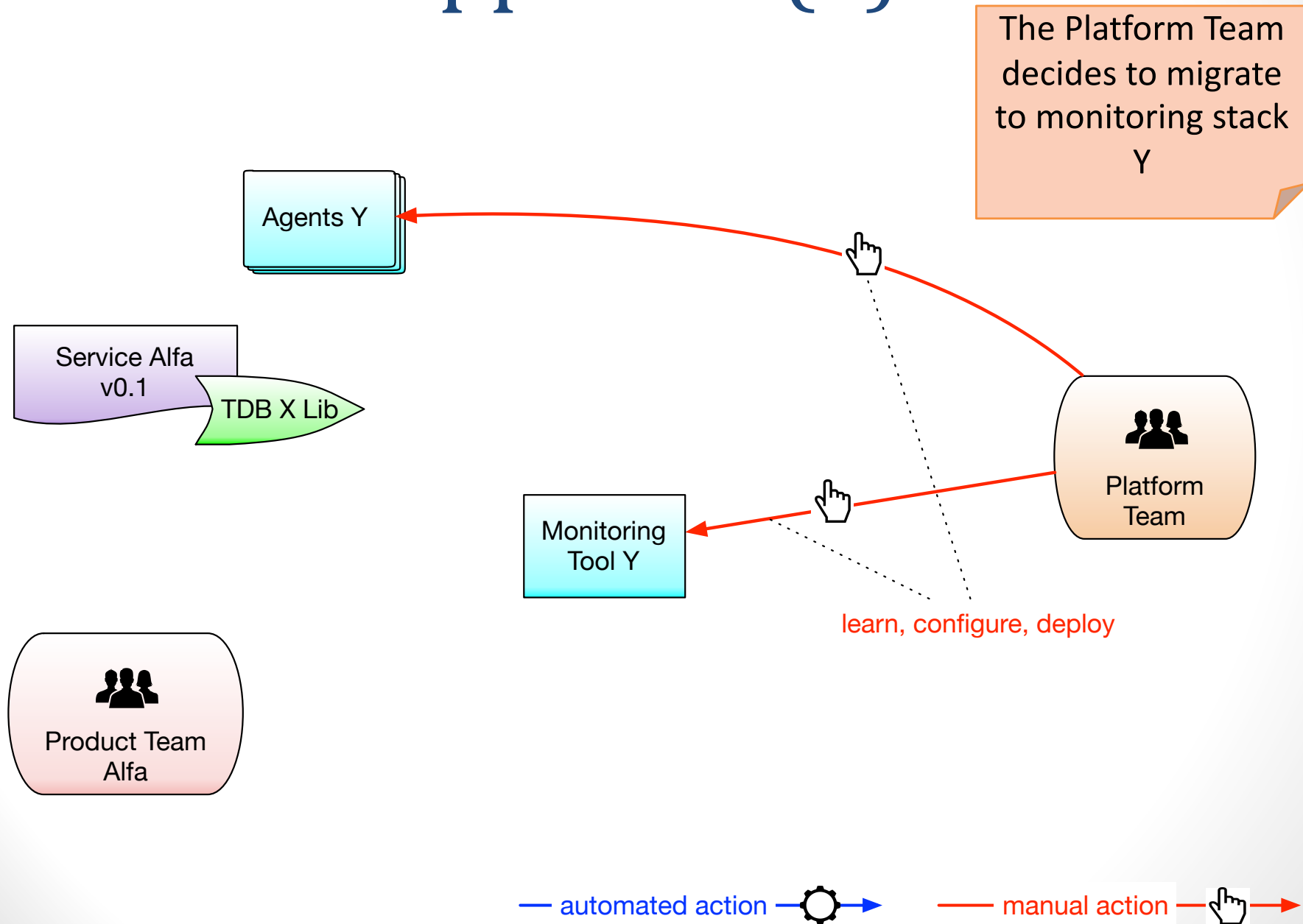
— manual action —  —

Classical Approach (3)

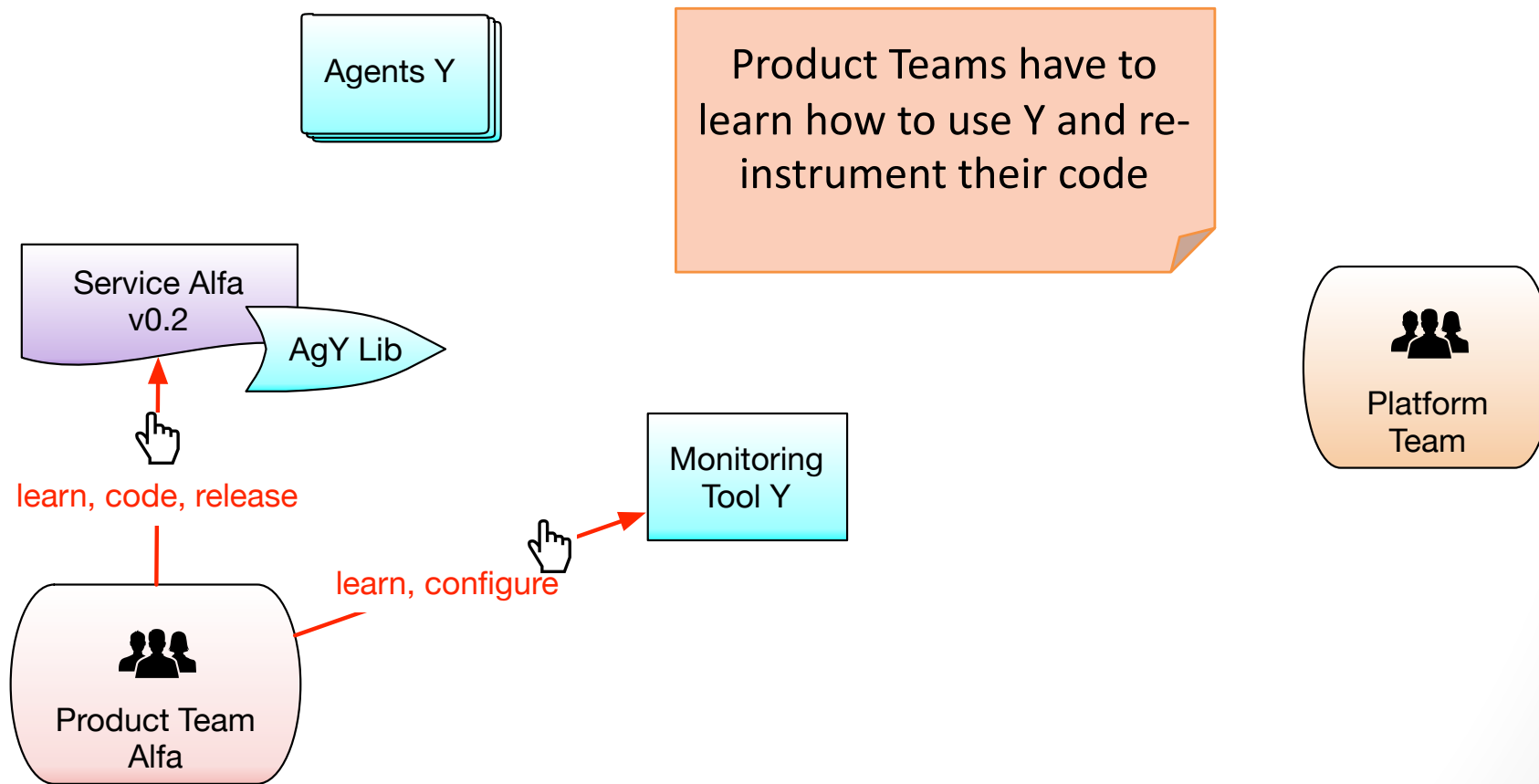


— automated action —  — manual action —  —

Classical Approach (4)



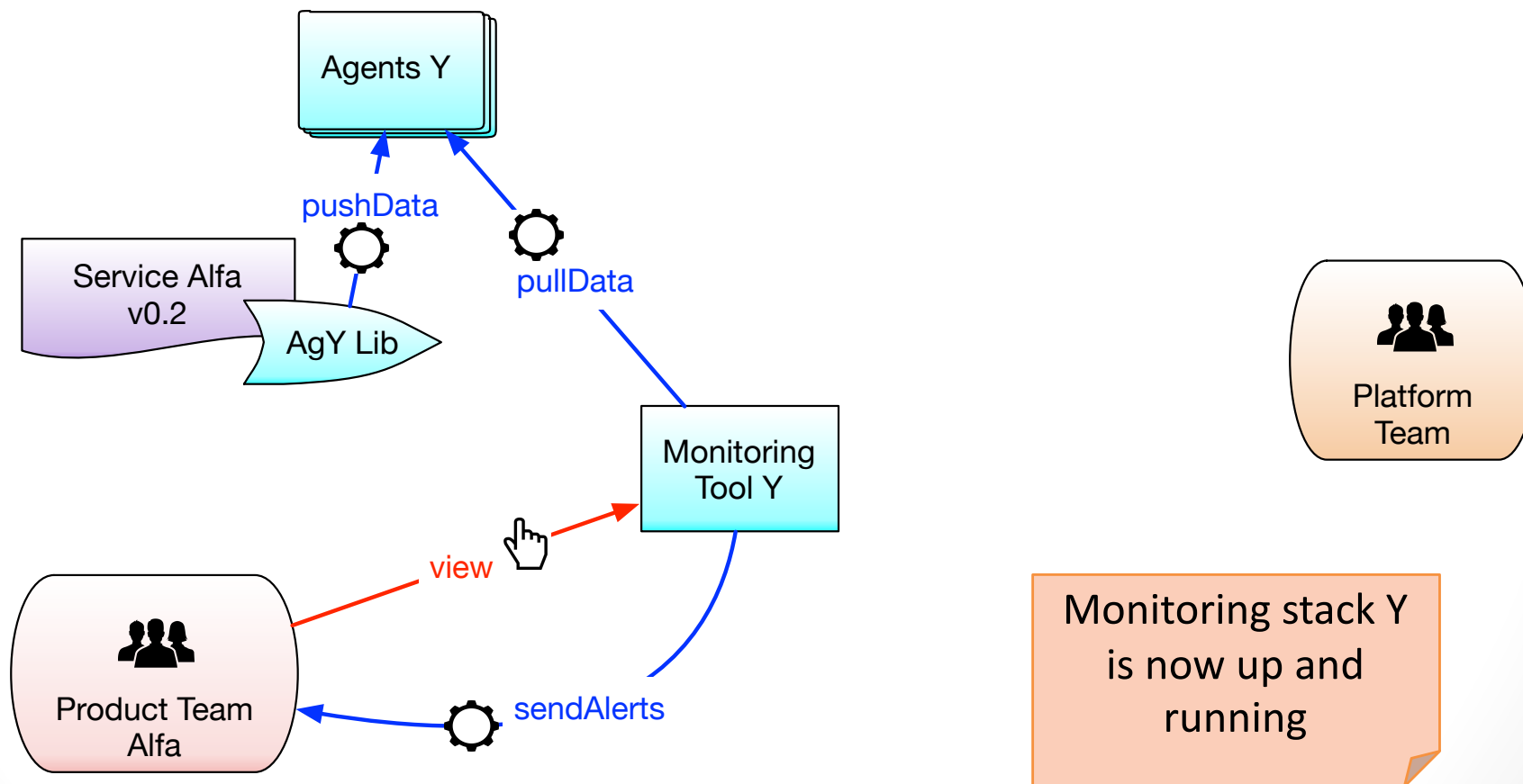
Classical Approach (5)



— automated action —  —

— manual action —  —

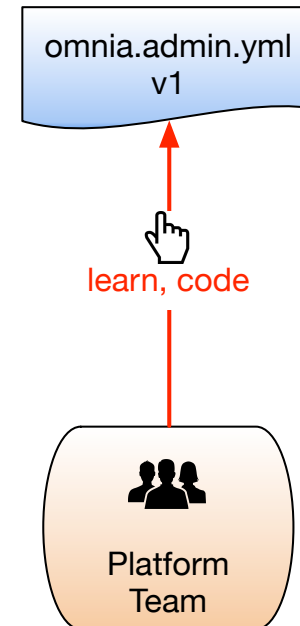
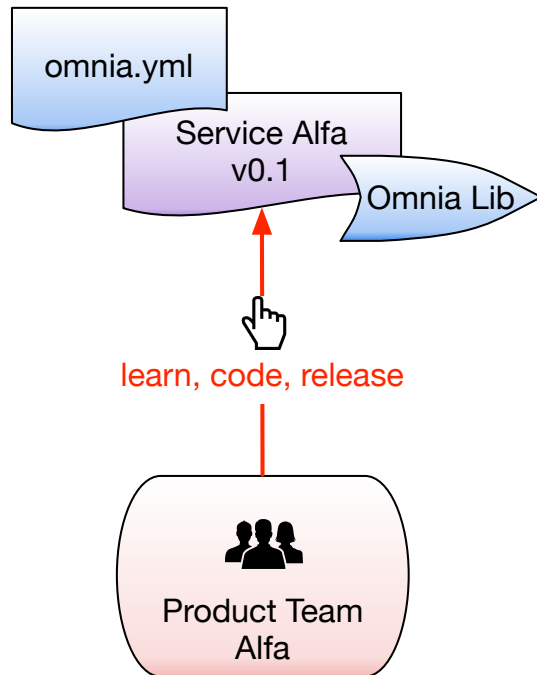
Classical Approach (6)



— automated action —  — manual action —  —

Omnia-based approach (1)

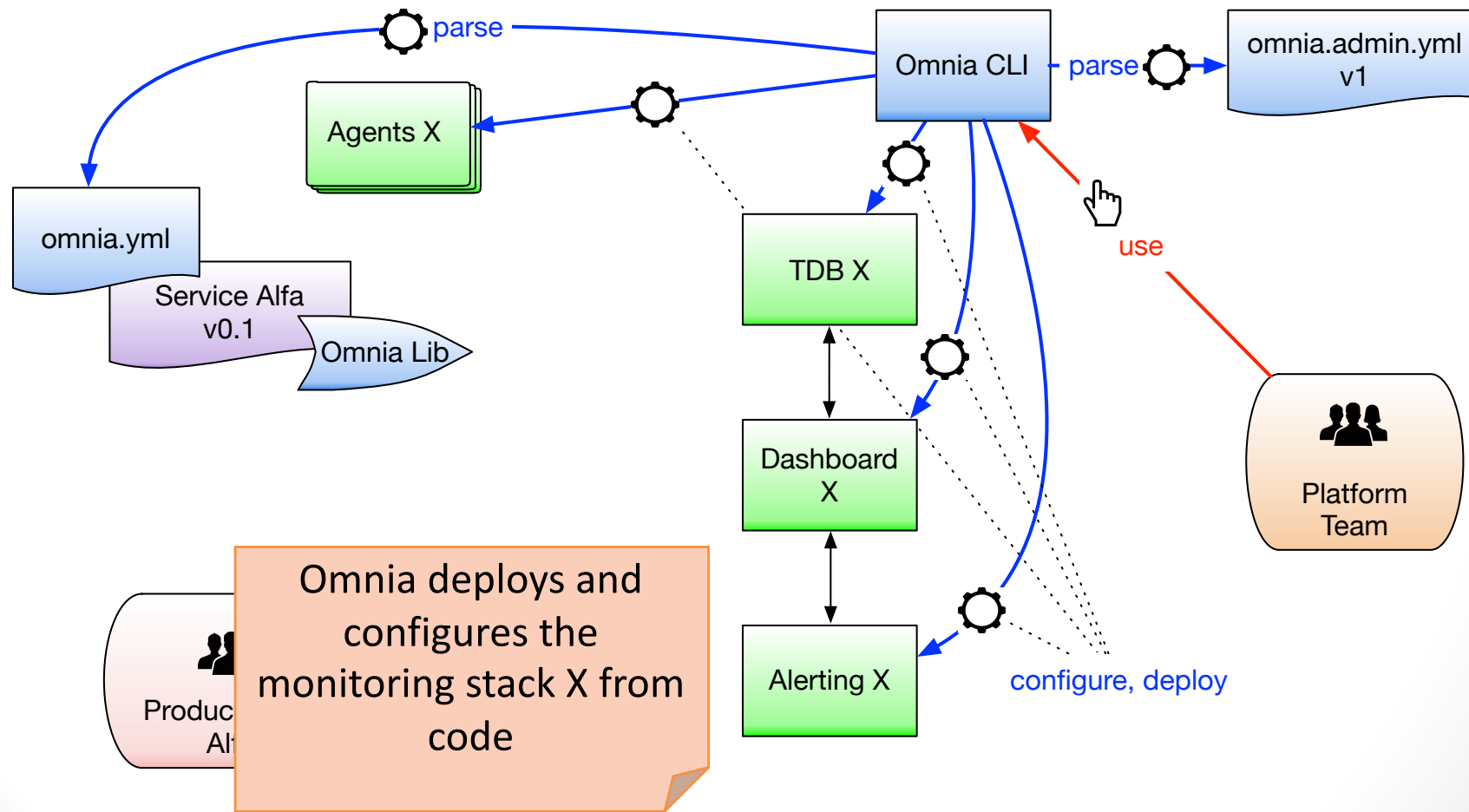
By using Omnia a single learning step is required for all teams



— automated action —  —

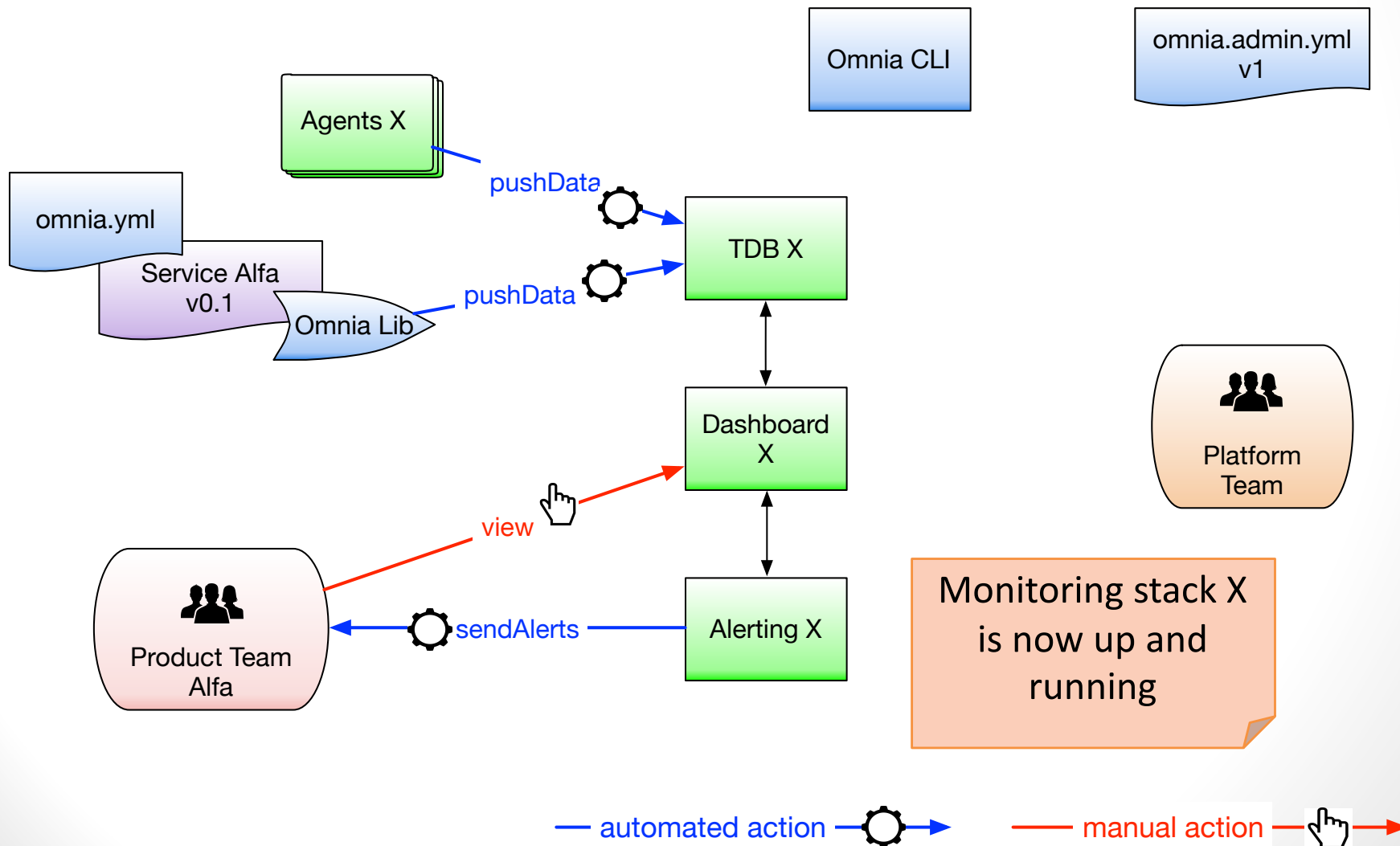
— manual action —  —

Omnia-based approach (2)

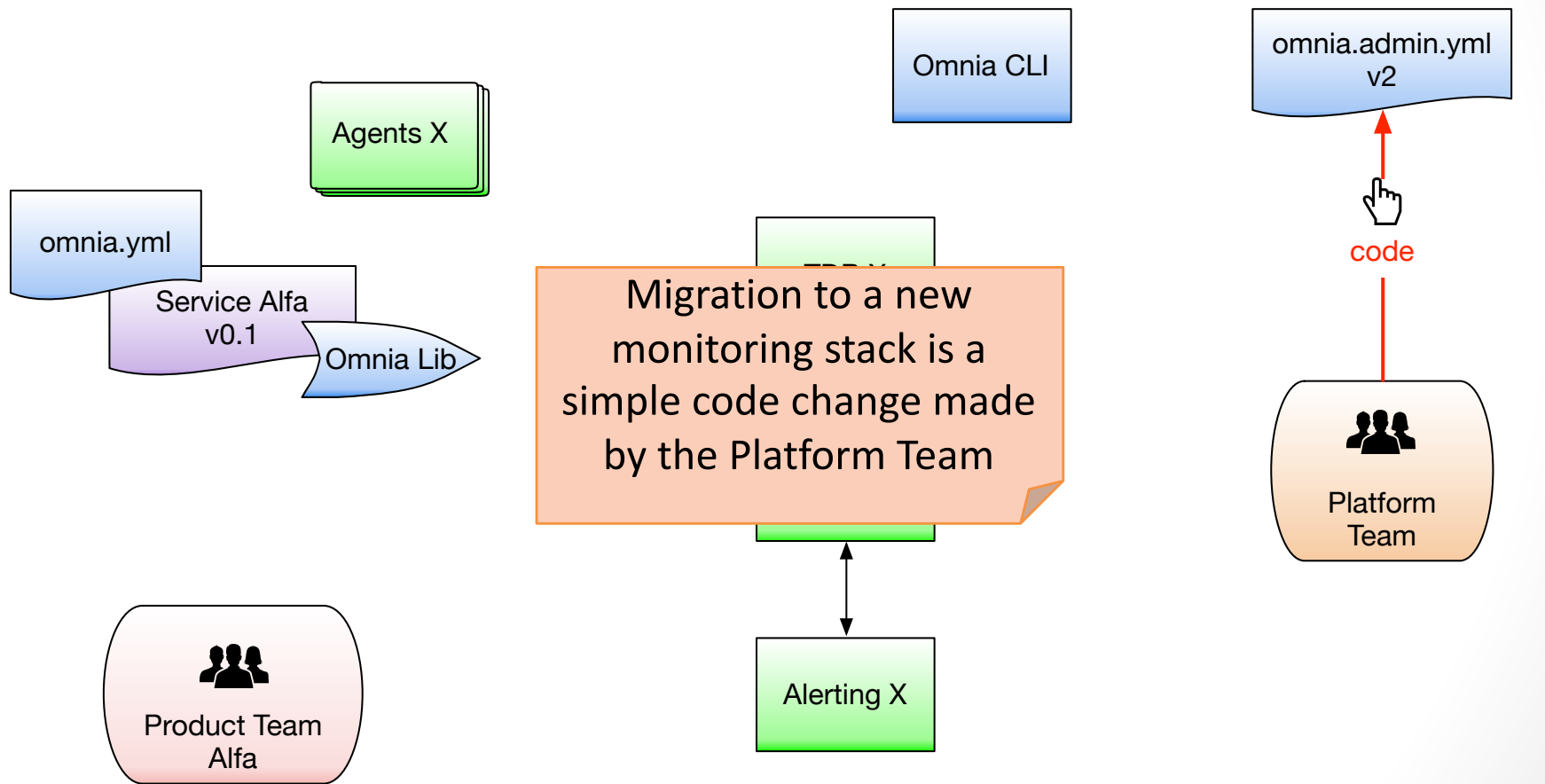


— automated action —  — manual action —  —

Omnia-based approach (3)



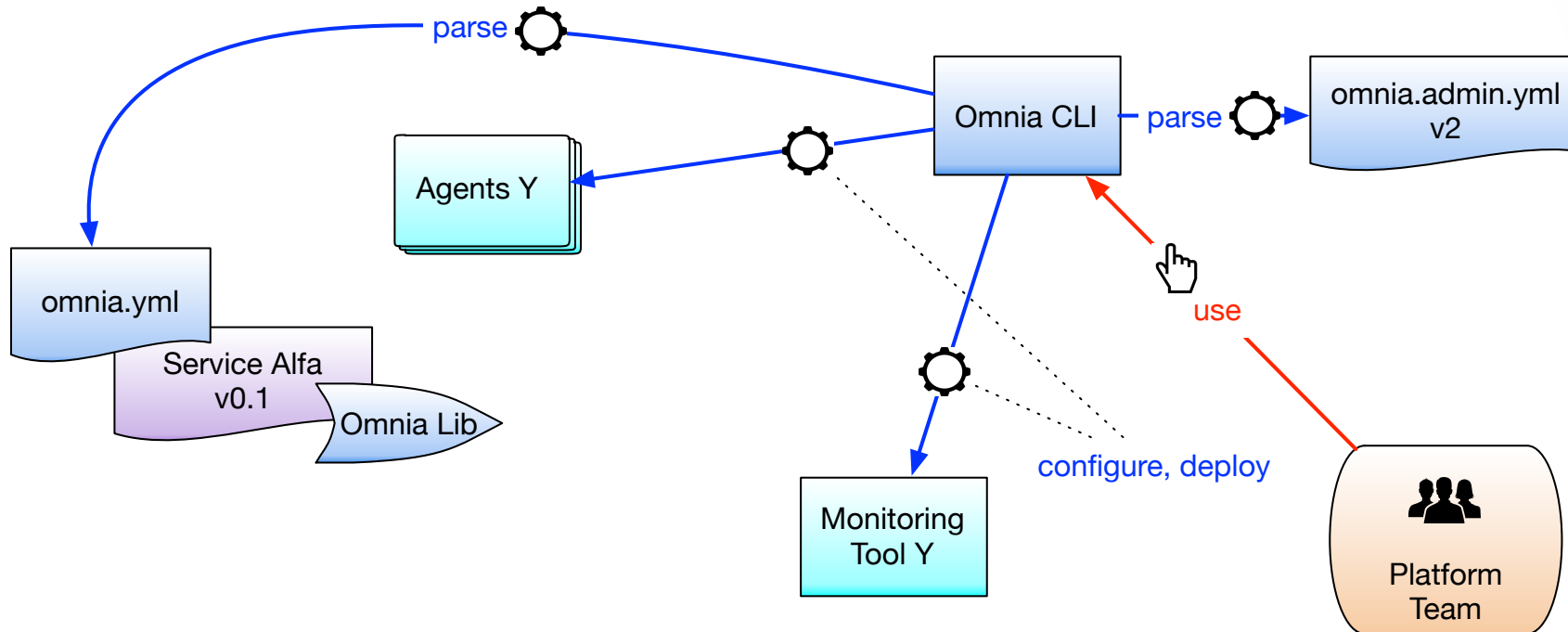
Omnia-based approach (4)



— automated action —  —

— manual action —  —

Omnia-based approach (5)



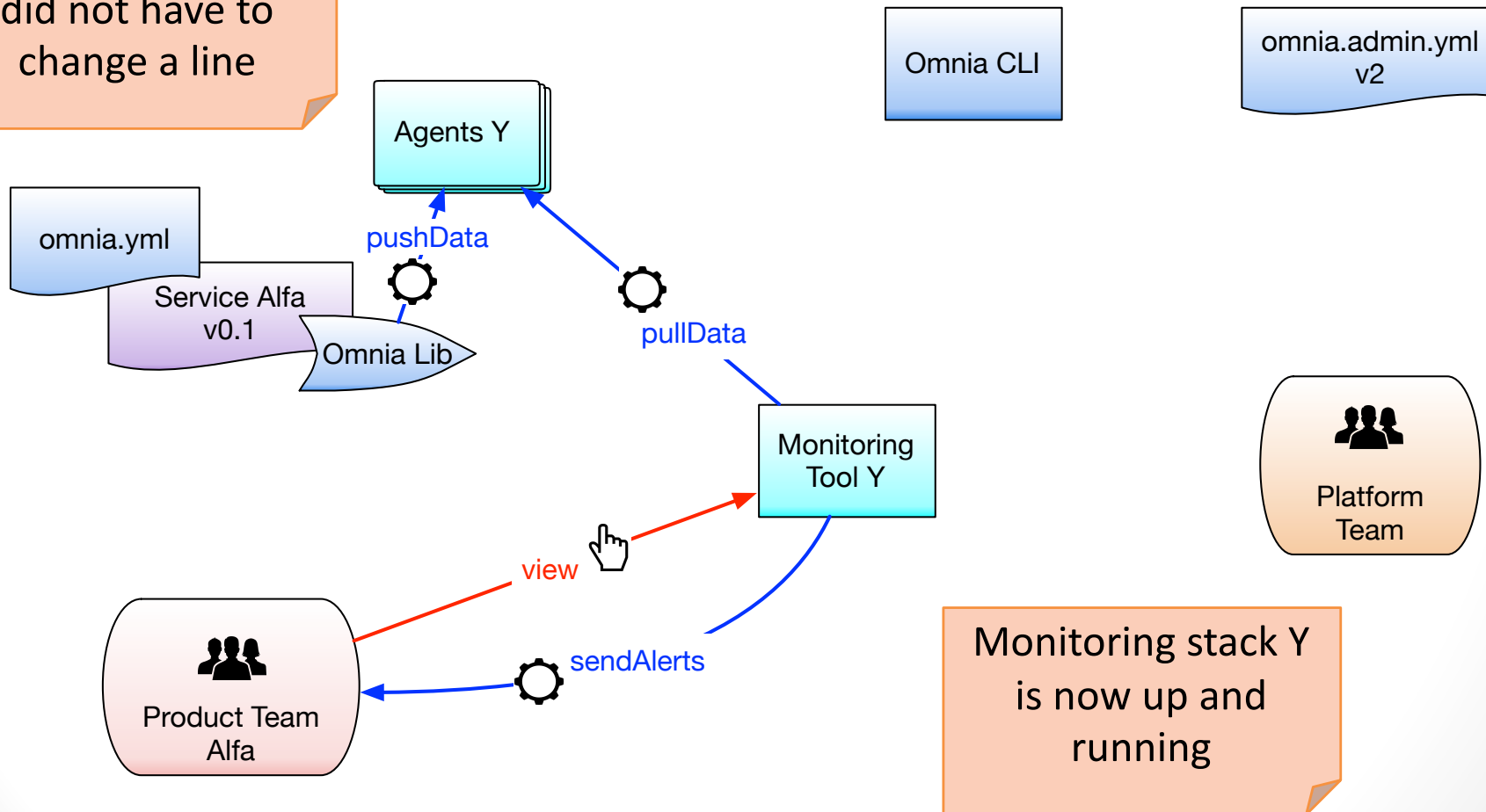
Omnia takes care of reconfiguring and redeploying the stack

— automated action —  —

— manual action —  —

Omnia-based approach (6)

The Product Teams did not have to change a line



— automated action —  — manual action —  —

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;
    }

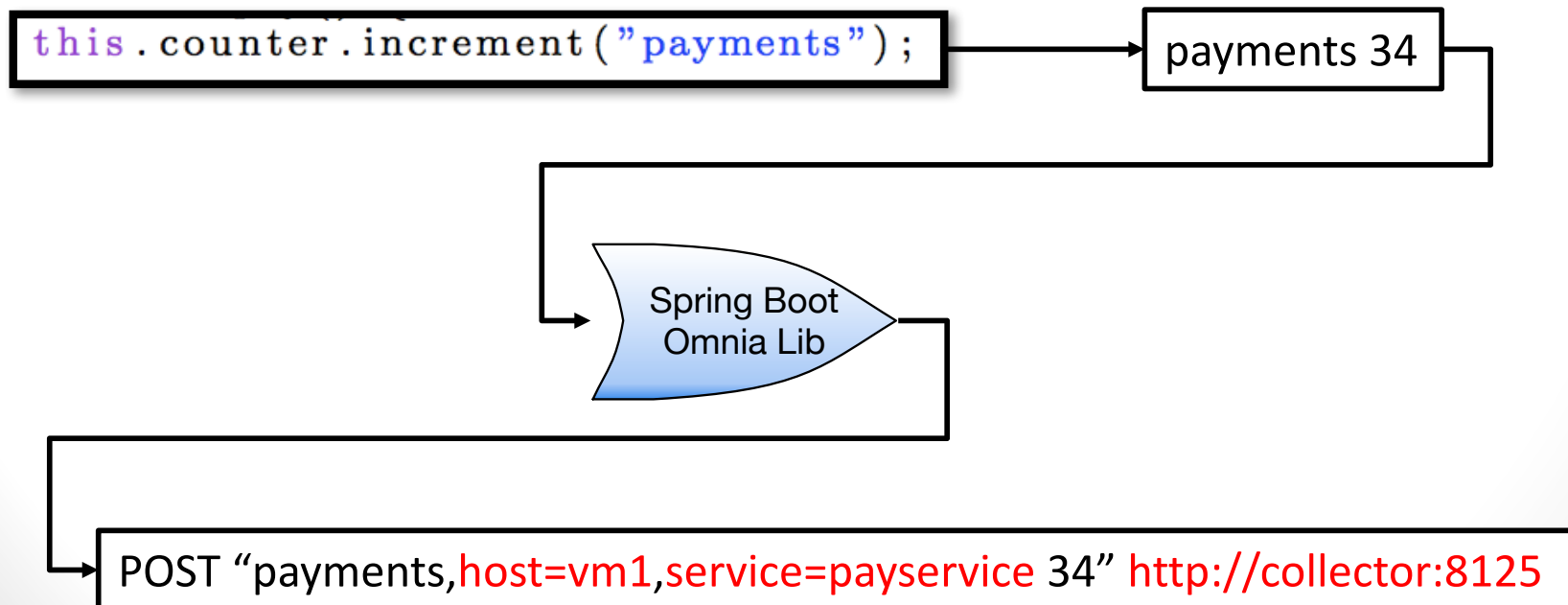
    public void pay() {
        // perform the payment
        ...

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

Omnia Lib for Spring Boot

Under the hood

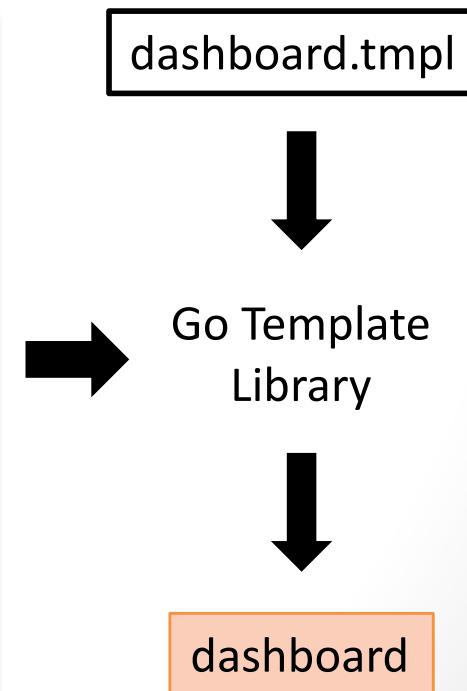
- Convention over configuration:
 - statsd (w/ influxdb tag ext.) protocol
 - automated meta-data decoration
 - default endpoint
- Example:



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
<i>host</i>	a physical or virtual machine
<i>service</i>	an application
<i>service_id</i>	a unique identifier for an instance of an application
<i>container_id</i>	a unique identifier for a Linux container
<i>container_image</i>	a Linux container image

- and for metrics

Host metrics	Java metrics
<i>cpu_usage_user</i>	<i>heap_memory_usage</i>
<i>cpu_usage_system</i>	<i>thread_count</i>
<i>cpu_usage_idle</i>	<i>loaded_class_count</i>
<i>mem_used</i>	<i>garbage_collection_time</i>
<i>mem_used_percent</i>	<i>thread_count</i>

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_from:
      - infl
    roles:
      - das
      - act
teams_repos:
  - "github.com/mmiglier/service"
  - "github.com/mmiglier/service"
```

Reusing existing adapters

```
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
      - action
  grafana:
    pulls_from:
      - prometheus
    roles:
      - dashboard
teams_repos:
```

v2

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