@sebineubauer
https://github.com/sebastianneubauer
sebastian.neubauer@blue-yonder.com

A Pythonic Approach to Continuous Delivery

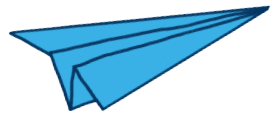
Sebastian Neubauer

Europython 2015

blue yonder

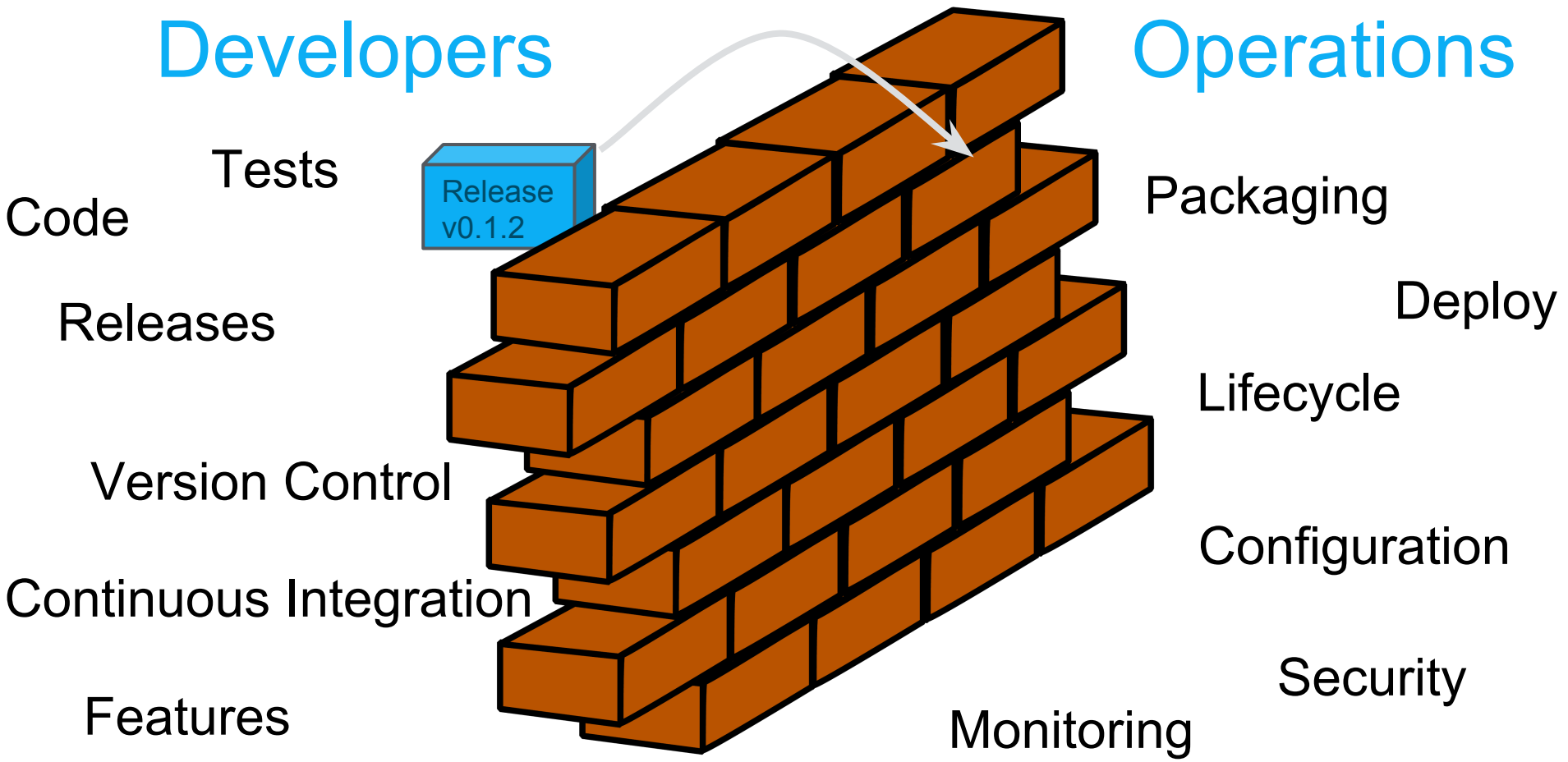
Overview

- What is Continuous Delivery?
  - definitions, analogies...
- How does a delivery pipeline look like?
  - deep dive into boring details…
- I have working python code, how do I start now?
  - we assemble exemplary building blocks to a working production line the pythonic way
- What could possibly go wrong?
  - traps, tips & tricks, failures, unsolved problems, dangers…
- What should the future bring?
  - wishes and dreams of brighter days
- Summary

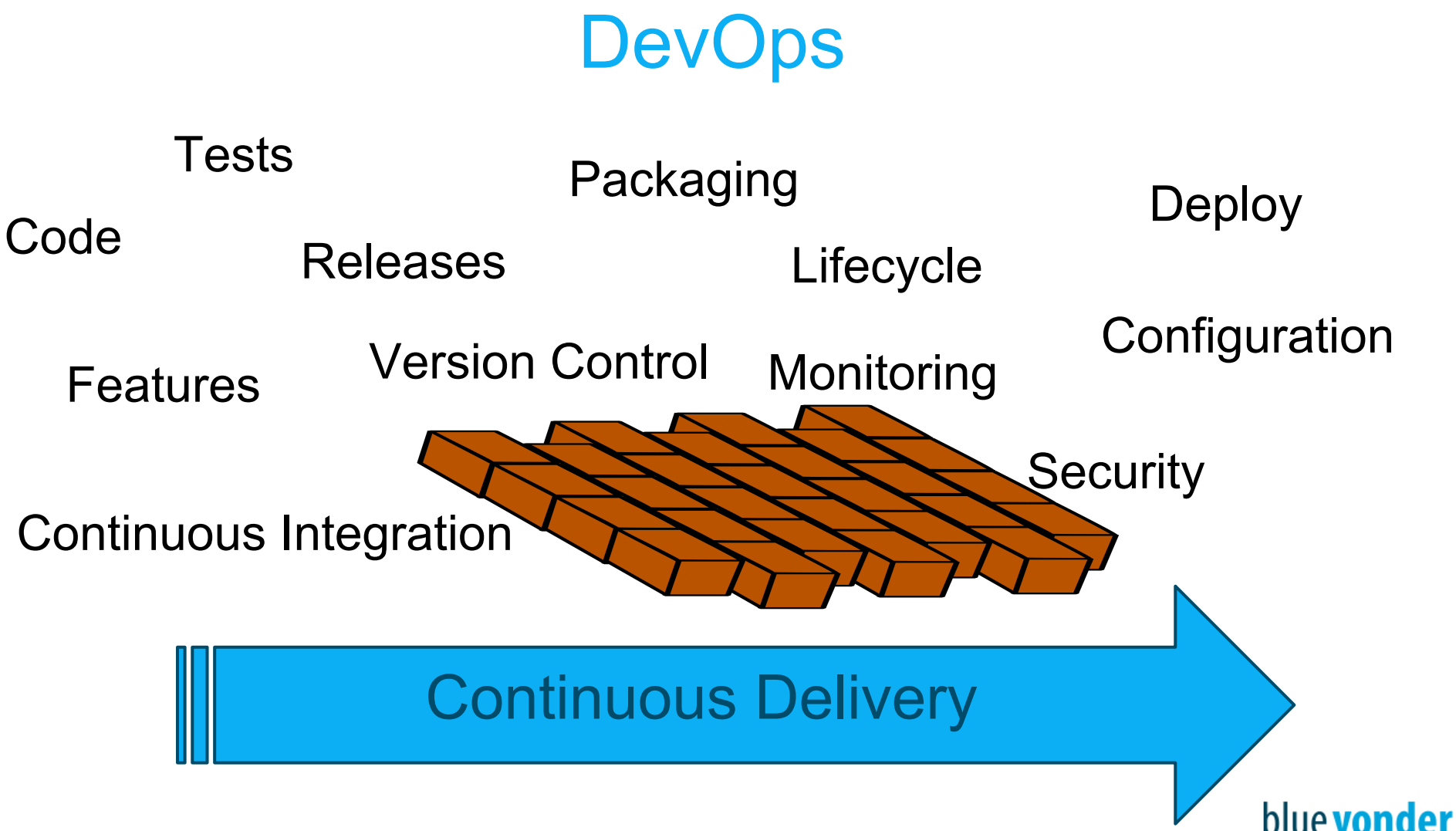


What is Continuous Delivery?

Overcoming the wall of confusion

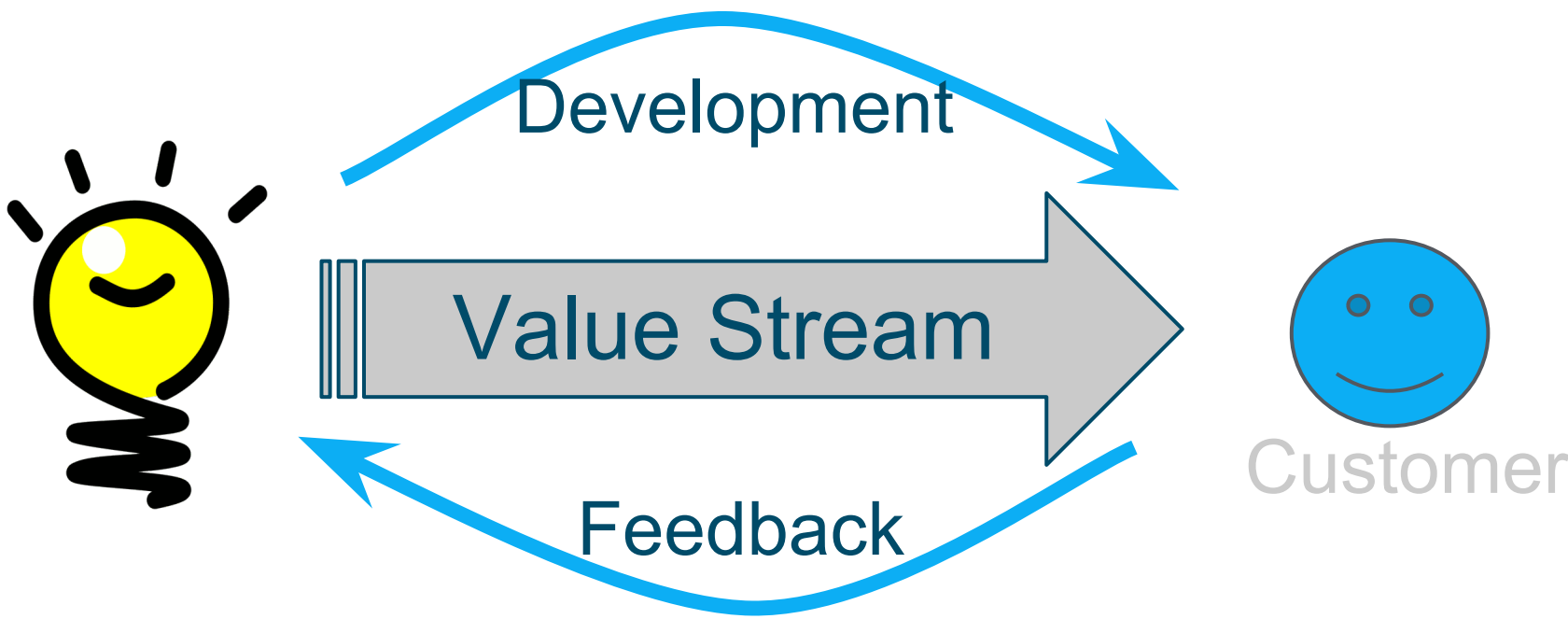


The “DevOps” thing…



Continuous Delivery

- Extending the development into production
- And extending operations into development
- Development includes the entire value stream
- Enables development cycles including customer feedback



Continuous Delivery

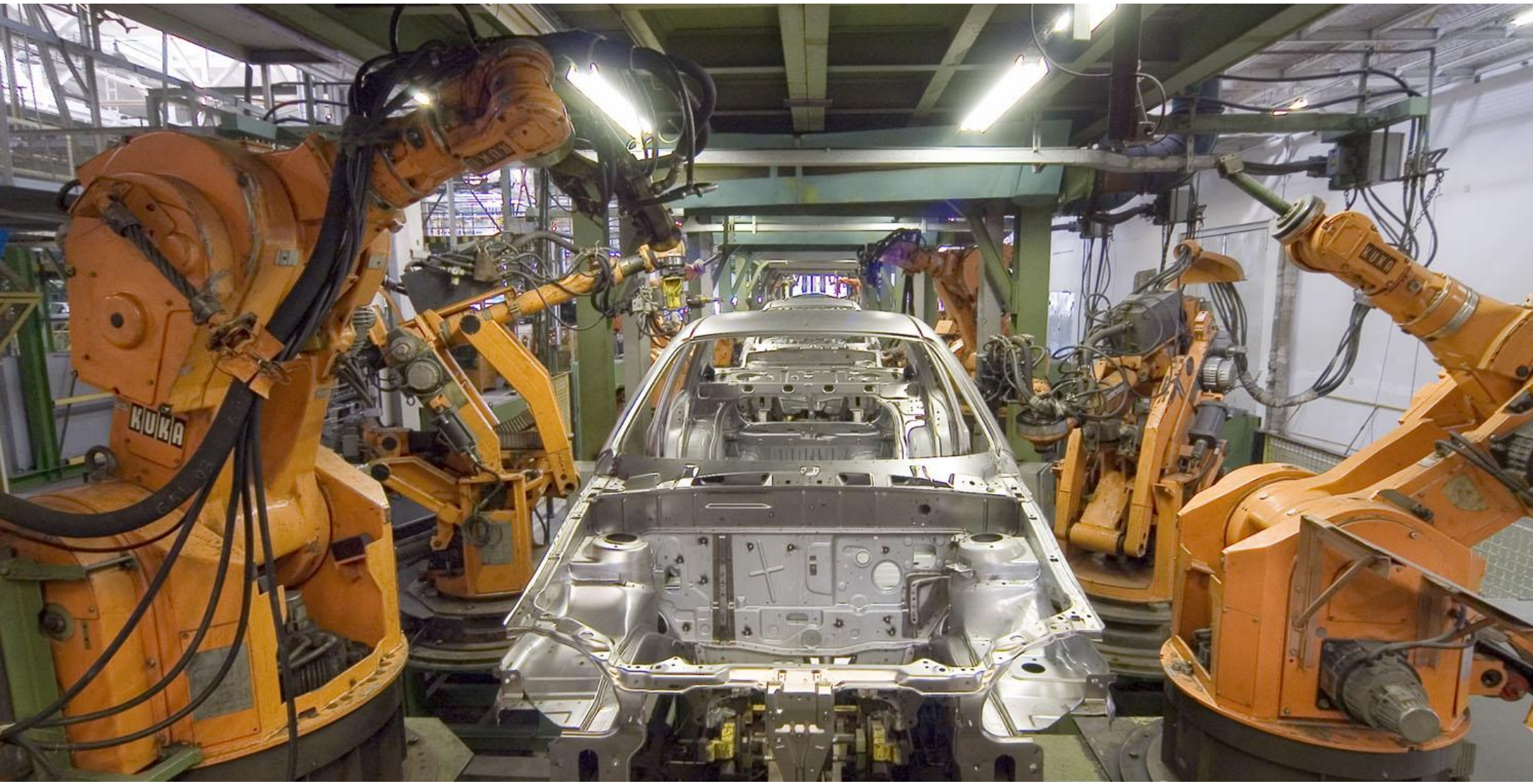
- Release early, release often!
- “Continuous” is far more often than you think
- Explosion of complexity due to increased demands on security, safety, failover, monitoring, tests

Poka-yoke (ポカヨケ)

“A poka-yoke is any mechanism in a lean manufacturing process that helps an equipment operator avoid (*yokeru*) mistakes (*poka*). Its purpose is to eliminate product defects by preventing, correcting, or drawing attention to human errors as they occur.”

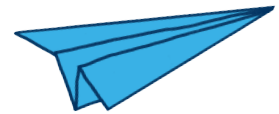
wikipedia.org

Automated Software Production Line



The Delivery Pipeline

blue yonder

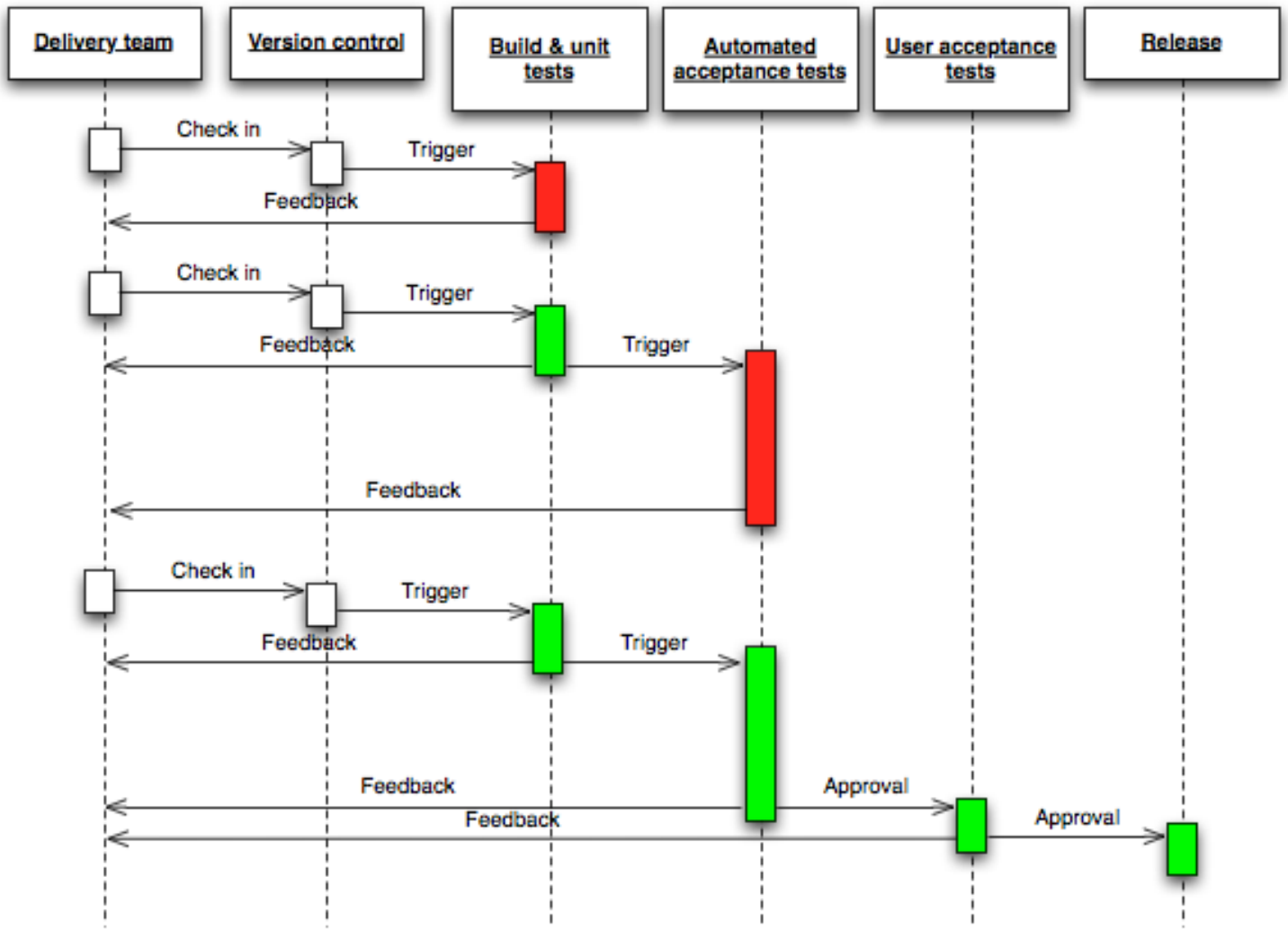


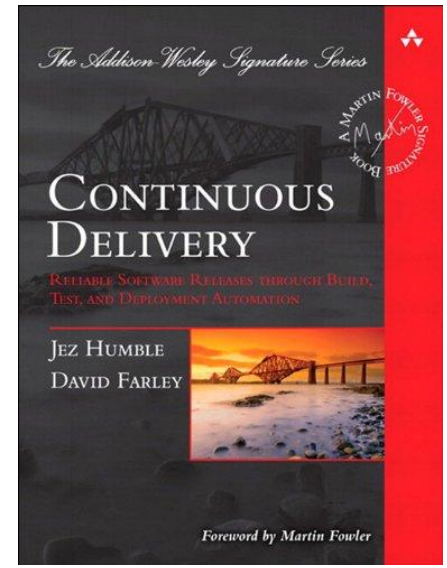
How does a **delivery pipeline** look like?

It's up to you!

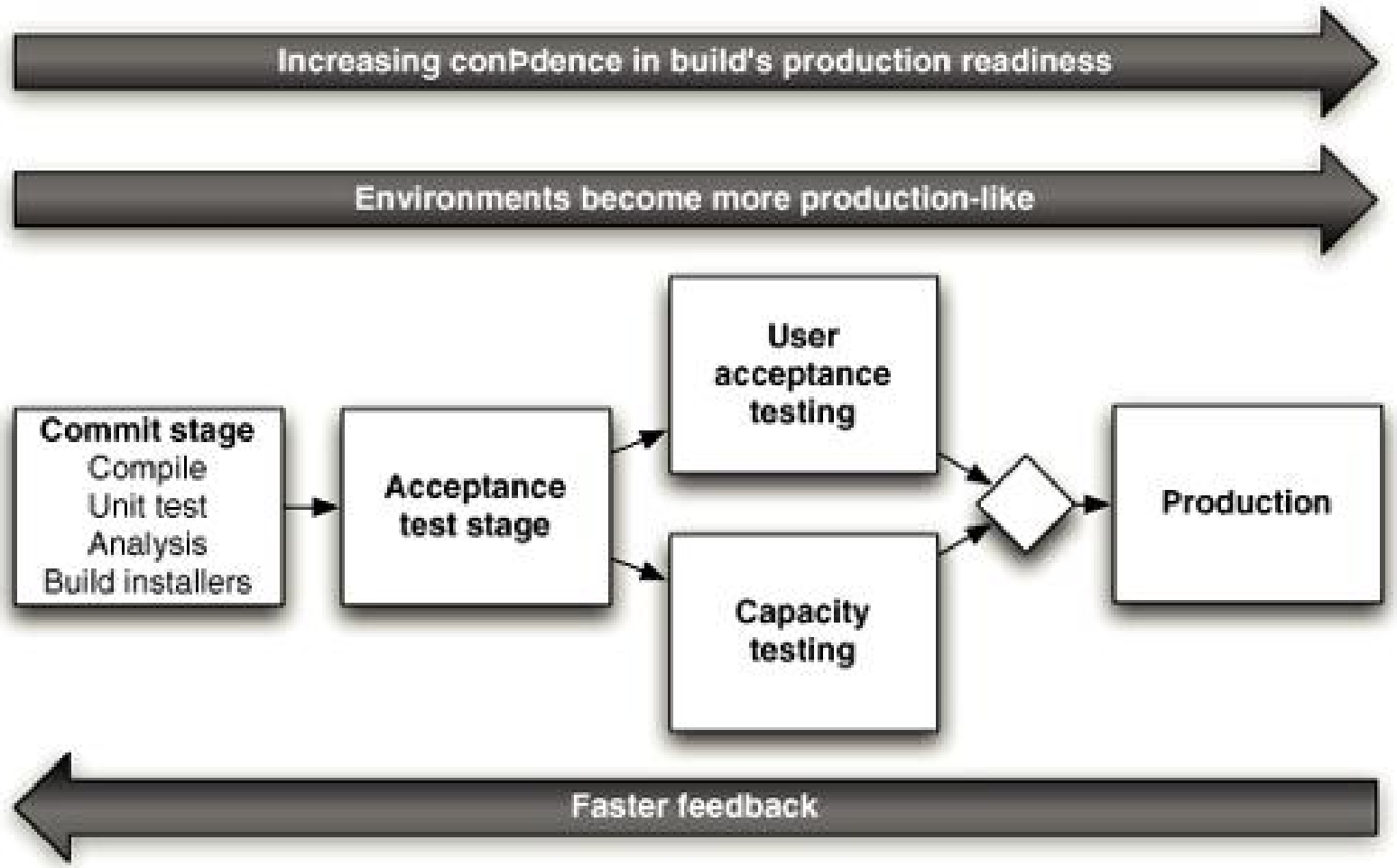
- Each change (commit) is deployed to production unless it is proven to be not production ready
- Design the automated challenges well
- Try to get feedback (failure) as soon as possible
- Start with a walking skeleton

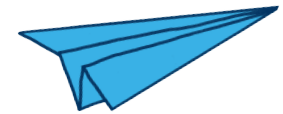
A Typical Pipeline





The Stages


Increasing conPdence in build's production readiness
Environments become more production-like
Commit stage
Compile
Unit test
Analysis
Build installers
Acceptance test stage
User acceptance testing
Capacity testing
Production
Faster feedback



I have working **python code**,
how do I start now?

A Proper Deployment Artifact

- This means put up everything for a proper deployable artifact:
  - python package
  - debian package
  - fancy docker
- It should be uniquely versioned
- It should manage dependencies
- Hint: https://github.com/blue-yonder/pyscaffold

```
>pip install pyscaffold
>putup my_app
```

-> Talk by I. Măriș from Monday:

“Less known packaging features and tricks”

Continuous Integration

- All automated tests are executed each time someone commits to master

```
>python setup.py test
```

- Might be a good idea to split fast unit-tests, from slow integration tests
- Any CI system will do the job: buildbot, travis...

Jenkins

Continuous Integration

- Not creative enough for the challenges?
  - unit tests: only code, no environment dependency
  - integration tests/component tests: allowed to use some environment dependencies: filesystem, http, database
  - static code analysis: pylint, pychecker
  - test coverage
  - doctests
- The result of CI is a fixed artifact with a unique version

```
>python setup.py sdist
```

- If you use pyscaffold, a PEP440 compatible version is generated from the git commit and tag:

```
0.0.1.post0.dev15+g172635
```

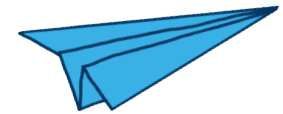
Fill up the Artifact Repository

- Ah, yes you need one, let's use the: http://doc.devpi.net/
- Devpi: secure, on-premise, open source, pypi compatible artifact repository (short: index)

```
>devpi upload
```

-> Talk by Stephan Erb today 12:30 B1

"**Release Management with Devpi**"



"That was the fun part! Now comes **pain, tears and configuration**"

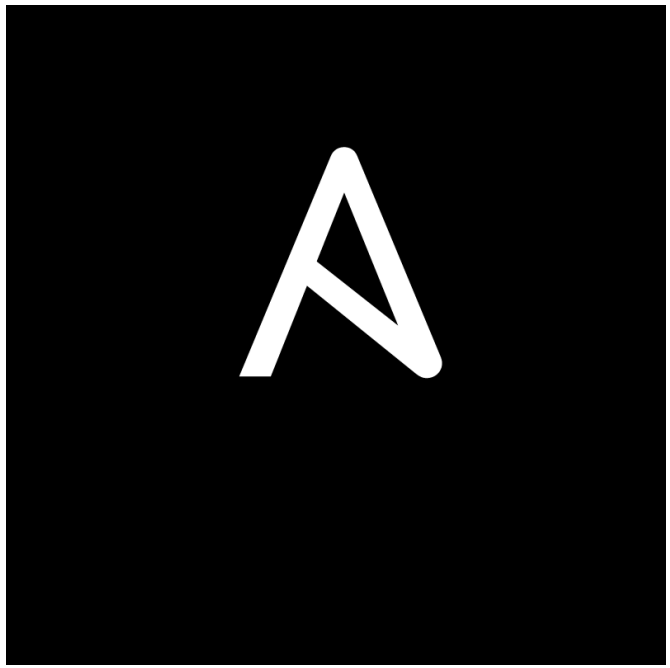
Automated Deploy

- For automated acceptance test, we need a fully functional running instance, deployed in a testing stage / test environment
- It is crucial, that the deployment code we use here, is the same we use later in production
- The testing stage needs to be as close to the production environment as possible
- Hint: After your first guesstimate of the time needed for automation:

Multiply by a factor of 3

Use Configuration Management

- You can use whatever you want for the deploy, even simple bash scripts, but....
- Config management tools will ease your automated deploy by orders of magnitude
- We use ansible, because it's: python, simple, lightweight, declarative,...



Example Ansible Playbook

```
---
- hosts: webservers
  tasks:
  - name: ensure my app is installed
    pip:
      name=my_app
      virtualenv=/my_app_home/venv
      extra_args='-i https://our_devpi/simple --pre -U'
      state=present

  - name: start the app
    shell: /my_app_home/venv/bin/my_app_started
```

Acceptance Tests

- Acceptance tests prove the correct behavior of your app
- Be aware: This behaviour earns your money
- It is your last chance: Bugs that pass here will end up in production!
- Tools you can use: plain unittest, behave, selenium,...

Last step to Production

- You might want to have some additional non-functional tests:
  - performance
  - security
  - explorative
- You might want to have some manual approval (feature flags)
- If possible perform a canary release

Steering of the Pipeline

- It is not trivial to keep control over the various deploy stages: which version passed which tests, where are which versions deployed...
- There are some few tool specialized for CD: go.cd or IBM UrbanCode...
- We use Jenkins, because we have it already
- Job dependencies reflect the stages
- A manual approval for production is done by clicking “Build” :-)

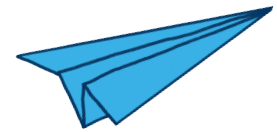
S	W	Name ↓	Last Success	Last Failure	Last Duration	Number of builds
		cds.ps.agent.prod_deploy_current	2 mo 23 days - #15	3 mo 1 day - #9	27 sec	11 0 4
		cds.ps.agent.prod_deploy_deprecated	2 mo 27 days - #4	3 mo 1 day - #3	41 sec	3 0 1
		cds.ps.agent.prod_deploy_latest	14 days - #53	7 days 6 hr - #54	32 sec	38 0 16
		cds.ps.agent.release	7 days 6 hr - #88	4 mo 21 days - #15	39 sec	83 0 5
		cds.ps.agent.staging_deploy_nightly	18 hr - #116	3 days 18 hr - #113	33 sec	96 0 20
		cds.ps.agent.staging_deploy_unstable	14 days - #96	7 days 6 hr - #97	31 sec	54 0 43
		cds.ps.agent.tests	7 days 6 hr - #3986	6 hr 18 min - #3987	49 sec	3955 0 32

blue yonder

What could possibly go **wrong**?

Traps, Tips & Tricks, Dangers...

- Keep it simple stupid!
- Automate all the things, because:
  - you are lazy
  - the complete delivery pipeline is in git:
    - you have predictable recovery
    - you know what is happening
  - machines do it just better
  - you can concentrate on value delivery
- Maintain and refactor your deployment
- For automation you need everything-as-a-service:
  - no tickets, no “you just have to click on”....



What should the **future** bring?

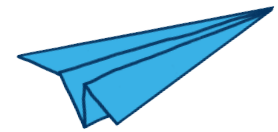
The not so perfect parts...

- Packaging and dependency management in python is not so perfect at the moment
- The two worlds should unite: OS package managers vs. pip
- A pythonic continuous delivery tool is still missing, jenkins is not sufficient:
  - what configuration is deployed where
  - access management
  - awareness of the delivery pipeline
- Many tools are still optimized for a manual workflow

Let's start hacking on it!

Summary

- CD rocks, because:
  - agile: faster feedback iterations
  - automated better than manual
  - collaboration better than silos
- You can build your own CD pipeline, just start today!
- Example building blocks are:
  - pyscaffold for python packages
  - devpi as artifact repository
  - jenkins for CI and steering
  - python unittest for tests
  - ansible for automated deploys
  - courage



Thank you!


If you think of
• literature when you hear **Kafka**
• mythology when you hear **Cassandra**
• animals when you hear **Zookeeper**
… then have a nice day.
If you think of distributed systems, then join us!
www.blue-yonder.com
blue yonder

Images:

slide 7: Cory Doctorow

https://www.flickr.com/photos/doctorow/17599851339/in/photolist-sPeQCK-8rL77m-5f8C9V-MxAHt-7L8phE-kN2oLW-kN24HW-4Le5L9-7m77Ag-8q6foW-5eB9a-iHD6Jx-uckeMj-48K71K-6iyUxC-bxrnpQ-9hZUBe-44LSH-sYFm9J-baBQvp-nTVXsi-7n9P22-9hZWcX-9hZVoM-66EEpg-sCfANm-6sGsY3-82ayUG-Mxs1C-8AharJ-4m2v48-nTW1k2-nTRhGu-q5MCm-nD3Wvv-PCCVj-oE51X-5V668Q-bpvXz-nBryWw-nTVZMi-nTVZv6-nTNrGY-c1z2Fo-c1z26J-6iuLbV-6gfqJX-aztFKe-4rT58o-nxPg3/

Slide 9 Mixabest

https://upload.wikimedia.org/wikipedia/commons/5/5e/KUKA_Industrial_Robots_IR.jpg