# Lecture 6: continuous deployment – part 2
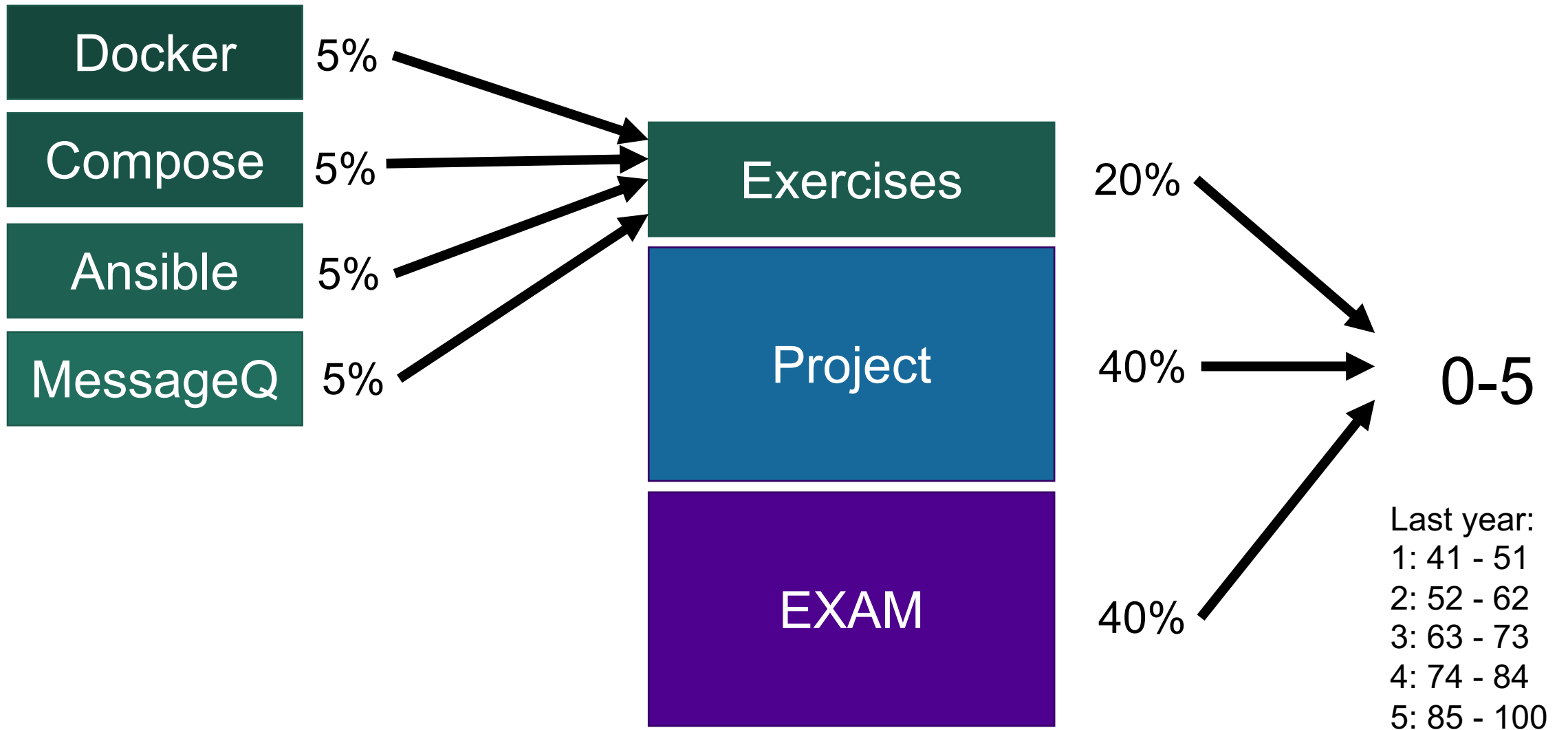
# Outline of today

- Course matters

- Discussion about "homework"

- Continuous Delivery & Deployment

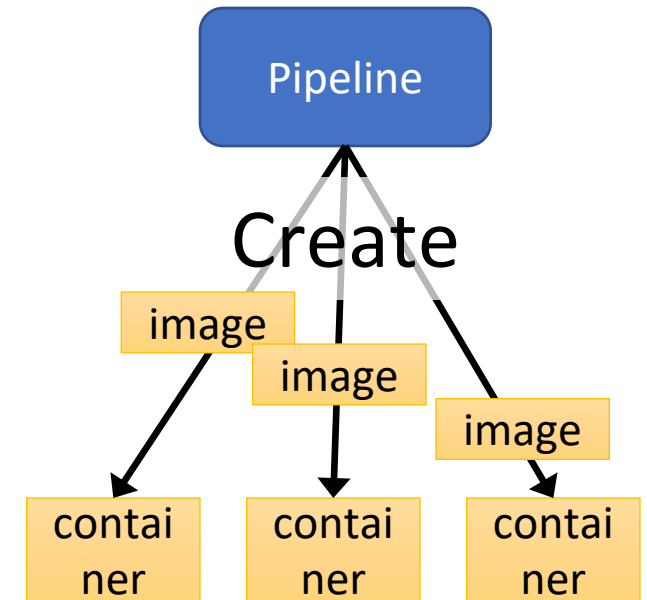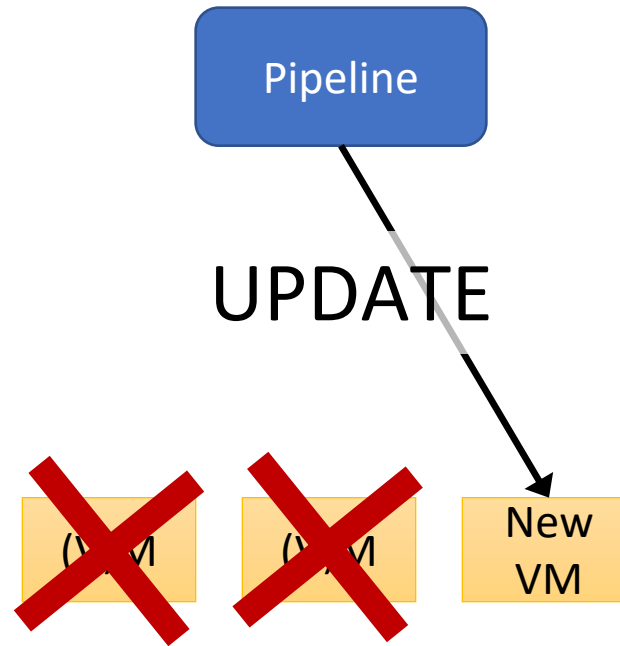# Participants (28.10.2021)

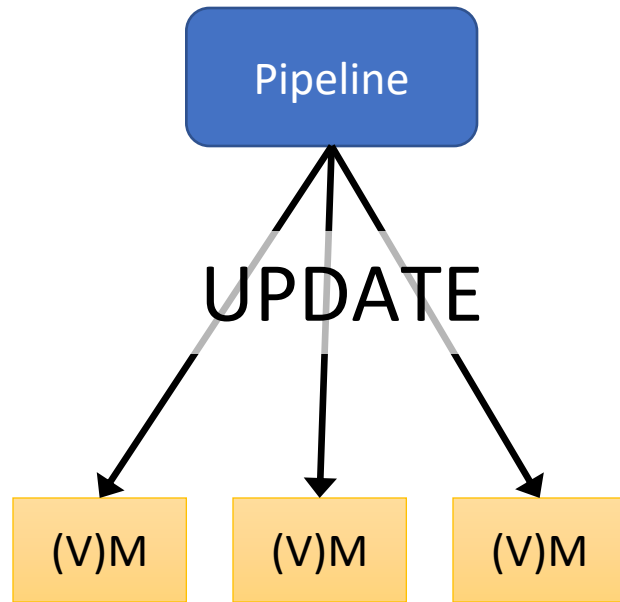|                                    | 28.9 | 5.10 |
|------------------------------------|------|------|
| • Total number of enrolled students: | 72   |      |
| • Answers to survey:               | 63   | 64   |
| • Current participants:            | 60   | 60   |
| • Docker exercise:                 | 54   | 59   |
| • Docker compose exercise:         | 17   | 50   |
| • Ansible exercise:                |      | 2    |

Docker — 5%

Compose — 5%

Ansible — 5%

MessageQ — 5%

Exercises — 20%

Project — 40%

EXAM — 40%

0-5

Last year:
1: 41 - 51
2: 52 - 62
3: 63 - 73
4: 74 - 84
5: 85 - 100

# Ansible exercise

# Backgroud (slide from last week)

# Alternative approaches for delivery

- Set-up everything when image is created
  - Very static
- Make the container to auto-update
  - You need to know in advance what might change
- ~~Put stuff to shared folder (use volume)~~
- Use configuration tools
  - Work also for full virtual machines and computers

# Ansible
# (https://www.ansible.com)

Automation engine for

- Provisioning
- Configuration Management
- App Deployment
- Continuous Delivery
- Security Automation
- Orchestration

uses YAML, in the form of Ansible Playbooks

# Ansible

- Ansible works by connecting to your nodes and pushing out small programs, called "Ansible modules" to them.
- These programs are written to be resource models of the desired state of the system.
- Ansible then executes these modules (over SSH by default), and removes them when finished.
- Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.
- Typically, you'll work with your favourite terminal program, a text editor, and probably a version control system to keep track of changes to your content.
- A short video:
  - https://www.ansible.com/resources/videos/quick-start-video

# Architecture
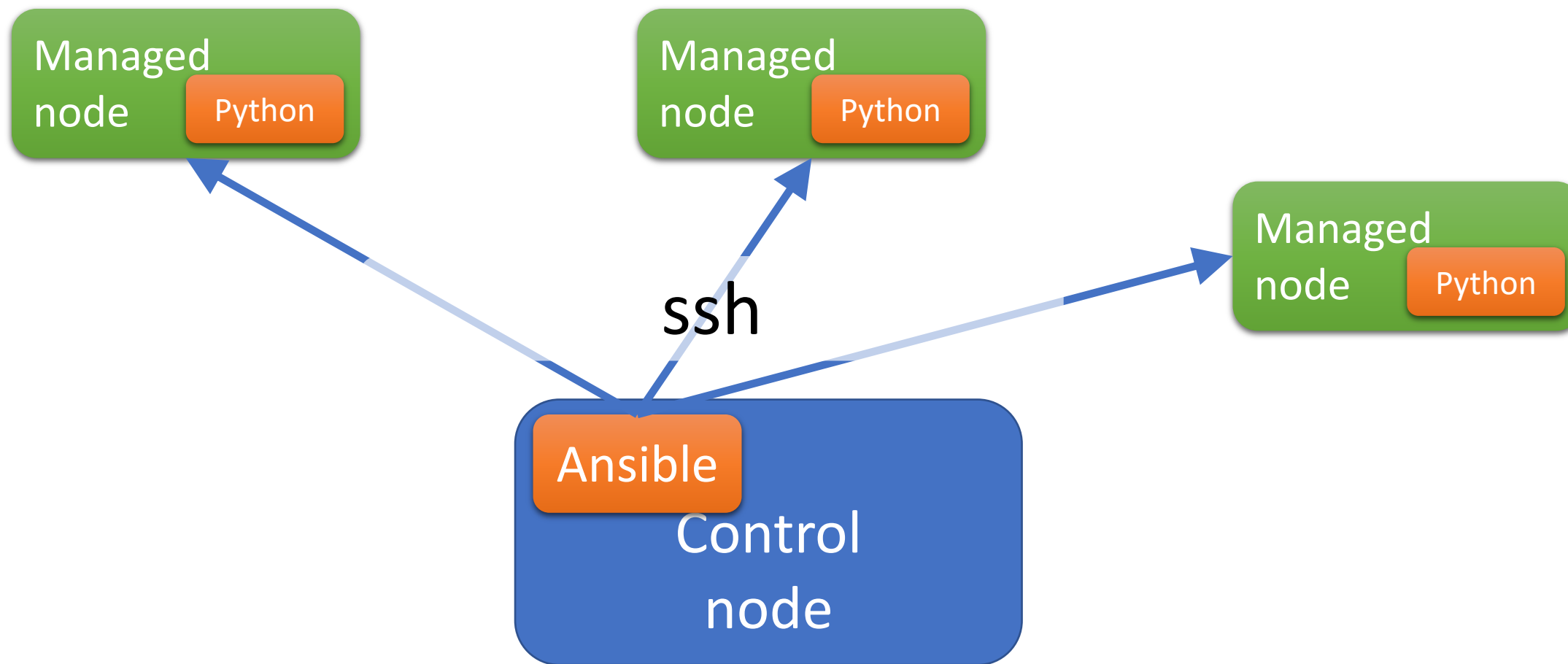
# Docker containers as targets

- Since we do not enough virtual machines,
  lets use Docker images
- Complicates the exercise,
- but allows you to learn more about Docker

Tampereen yliopisto
Tampere University

```
FROM debian

USER root

# Copy application itself:
COPY . /home
WORKDIR /home
RUN apt-get update
RUN apt-get install -y nodejs
ENTRYPOINT node server.js
```
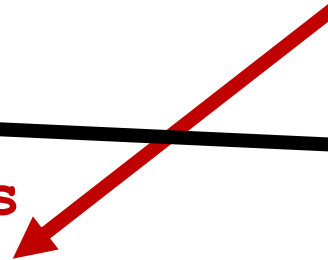
Debugging aid

Time consuming init

**Docker build -t utest**

```
1       FROM  utest


2       RUN apt-get install -y openssh-server

3       RUN sed -i 's/PermitRootLogin prohibit-password/PermitRootLogin yes/'
        /etc/ssh/sshd_config

4       RUN apt-get install -y net-tools

5       RUN useradd -m -s /bin/bash -G sudo -p $(openssl passwd -1 eee) ssluser


6       RUN apt-get install -y python3

7       RUN apt-get install -y sudo


8       ENV PORT=8894

9       EXPOSE 22


10      ENTRYPOINT service ssh start && node server.js
```

# SSH support two alternative ways for authentication

Password

- Used in the previous slide
- Not very secure
- You can use, but gives at most 80% of the maximum points

Public/private keypair

- Public key of your computer is installed to the host
- More secure
- If you want 100% of maximum points, you should use this
(building of the image need to be changed)

Info

- Short:https://unix.stackexchange.com/questions/210228/add-a-user-without-password-but-with-ssh-and-public-key
- Long: https://www.ssh.com/academy/ssh/key

# Example ansible playbook

```yaml
---

- hosts: webservers
  vars:
      http_port: 80
      max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the
          latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config
          file.
    template:
      src: /srv/httpd.j2
      dest:/etc/httpd.conf
    notify:
      - restart apache

  - name: ensure apache is
          running
    service:
      name: httpd
      state: started

  handlers:
    - name: restart apache
      service:
        name: httpd
        state: restarted
```
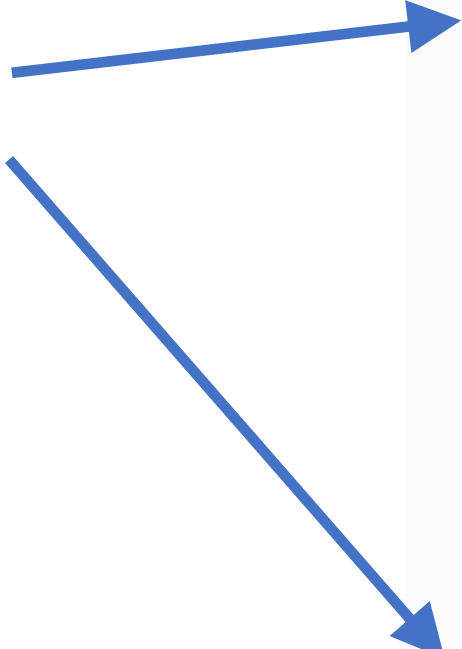
| Operating System | Format | Tool(s) |
|---|---|---|
| Debian | .deb | apt, apt-cache, apt-get, dpkg |
| Ubuntu | .deb | apt, apt-cache, apt-get, dpkg |
| CentOS | .rpm | yum |
| Fedora | .rpm | dnf |
| FreeBSD | Ports, .txz | make, pkg |

# Sidenode apt vs yum examples

| Task | apt (deb) | yum (rpm) | zypper (rpm) |
|---|---|---|---|
| **Install from repository** | apt-get install pkg-name | yum install pkg-name | zypper install pkg-name |
| **Update package** | apt-get install pkg-name | yum update pkg-name | zypper update -t package pkg-name |
| **Remove package** | apt-get remove pkg-name | yum erase pkg-name | zypper remove pkg-name |
| **Install from package file** | dpkg -i pkg-name | yum localinstall pkg-name | zypper install pkg-name |

There can be multiple plays

```yaml
---
- hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf

- hosts: databases
  remote_user: root

  tasks:
  - name: ensure postgresql is at the latest version
    yum:
      name: postgresql
      state: latest
  - name: ensure that postgresql is started
    service:
      name: postgresql
      state: started
```

- Read Ansible tutorial to understand how it works. A good starting point is: https://docs.ansible.com/ansible/latest/user_guide/intro_getting_started.html

- Prepare a docker image that can be used as a target. See details in below.

- Install Ansible in your computer.

- Make simple playbook
  - Check that the image has the latest version of git version management system
  - Queries the uptime (Linux command uptime) of target host

- Return result to Plus

1. Start one container from the image, get its IP-address.
   (in case of password-based authentication you need a manual login after start)
2. Ensure that the IP address is in /etc/ansible/hosts
3. Run the playbook
4. Copy the output (O1)
5. Run the playbook again
6. Copy that output, too (O2)
7. Start a second contained from the image, get its IP-address.
8. Ensure that this IP address is in /etc/ansible/hosts, too.
9. Run the playbook
10. Copy the output (O3)
11. Run the playbook again
12. Copy that output, too (O4)

- Git link of the code (teacher may want do git clone). Use different folder or subrepo from earlier exercises.
- The report should have a "report.pdf" with the following contents.
  - All the copied output (O1,O2,O3,O4)
  - Comments on what was easy and what was difficult.

# Main principles
# (https://continuousdelivery.com/principles/)

- Build quality in

- Work in small batches

- Computers perform repetitive tasks, people solve problems

- Relentlessly pursue continuous improvement

- Everyone is responsible

Sound familiar from somewhere?

# CI – essential practices
## (according to Humbley and Farley)

- Don't check in on a broken code

- Always run all commits tests locally before committing, or get your CI server to do it for you

- Wait for commit tests to pass before moving on

- Never go home on a broken build

- Always be prepared to revert to the previous revisions

- Time-box fixing before reverting

- Don't comment out failing tests

- Take responsible for all breakages that result from your changes

- Test-driven development

# Deployment essential pract.
## (according to Humbley and Farley)

- Only build your binaries once
- Deploy the same way to every environment
- Smoke-test your deployments
- Deploy to copy of production
- Each change should propagate through the pipeline instantly
- If any part of pipeline fails, stop the line

# Back to CD

# Continuous delivery and deployment

(http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment)

# Perceived benefits

- **Improved delivery speed** of software changes Improved speed in the development and deployment of software changes to production environment.

- **Improved productivity in operations** work. Decreased <u>communication</u> problems, bureaucracy, <u>waiting overhead</u> due to removal of manual deployment hand-offs and organisational boundaries; Lowered <u>human error</u> in deployment due to automation and making <u>explicit knowledge of operation-related</u> tasks to software development

- **Improvements in quality**. Increased <u>confidence</u> in deployments and reduction of deployment <u>risk and stress</u>; Improved <u>code quality</u>; Improved <u>product value</u> to customer resulting from production feedback about users and usage.

- **Improvements in organisational-wide culture** and mind-set. Enrichment and wider <u>dissemination of DevOps</u> in the company through discussions and dedicated training groups 'communities of practice'

# Maturity models in software engineering

- The first welknown was
  - Capability Maturity Model developed by Software Engineering Institute at Carnegie Mellon University in 1986
  - Five levels:
    - *Initial* (chaotic, ad hoc, individual heroics) - the starting point for use of a new or undocumented repeat process.
    - *Repeatable* - the process is at least documented sufficiently such that repeating the same steps may be attempted.
    - *Defined* - the process is defined/confirmed as a standard [business process](#)
    - *Capable* - the process is quantitatively managed in accordance with agreed-upon metrics.
    - *Efficient* - process management includes deliberate process optimization/improvement.
- Practical meaning may be questioned, but there has been many followers.

# Maturity model for CD

(https://www.urbancode.com/resource/continuous-delivery-maturity-model/ was at developer.ibm.com)

- **Base**: The base level is enough to "be on the model".
  The team has left fully manual processes behind.
- **Beginner**: At the beginner level, the team is trying to adopt some ECD practices in earnest but is still performing them at a rudimentary level.
- **Intermediate**: Practices are somewhat
  mature and are delivering fewer errors and more efficiency.
  For many teams, intermediate practices may be sufficient.
- **Advanced**: doing something well beyond what most of the rest of the industry and is seeing a great deal of efficiency and error prevention as a result.
- **Extreme**: Elements within the Extreme category are ones that are expensive to achieve but for some teams should be their target. Put another way, most organizations would be crazy to implement them, while this minority would be crazy to not implement them.

# Another

**Base** … started to prioritize work in backlogs, have some process defined which is rudimentarily documented and developers are practicing frequent commits into version control.

**Beginner** … teams stabilize over projects and the organization has typically begun to remove boundaries by including test with development. Multiple backlogs are naturally consolidated into one per team and basic agile methods are adopted ….

**Intermediate** … extended team collaboration when e.g. DBA, CM and Operations are beginning to be a part of the team or at least frequently consulted by the team. Multiple processes are consolidated and all changes, bugs, new features, emergency fixes, etc, follow the same path to production. Decisions are decentralized to the team and component ownership…

**Advanced** … team will have the competence and confidence it needs to be responsible for changes all the way to production. Continuous improvement mechanisms are in place … releases of functionality can be disconnected from the actual deployment, which gives the projects a somewhat different role. A project can focus on producing requirements for one or multiple teams and when all or enough of those have been verified and deployed to production the project can plan and organize the actual release to users separately.

**Expert** …some organizations choose to make a bigger effort and form complete cross functional teams that can be completely autonomous. With extremely short cycle time and a mature delivery pipeline, such organizations have the confidence to adopt a strict roll-forward only strategy to production failures.

# Another

**Base** … one or more legacy systems of monolithic nature in terms of development, build and release. Many organizations at the base maturity level will have a diversified technology stack but have started to consolidate … to get best value from the effort spent on automation.

**Beginner** … the monolithic structure of the system is addressed by splitting the system into modules … this will also naturally drive an API managed approach to describe internal dependencies and also influence applying a structured approach to manage 3rd party libraries … importance of applying version control to database changes will also reveal itself.

**Intermediate**. … a solid architectural base for continuous delivery … feature hiding for the purpose of minimizing repository branching to enable true continuous integration. … modularization will evolve into identifying and breaking out modules into components that are self-contained and separately deployed. … start migrating scattered and ad-hoc managed application and runtime configuration into version control and treat it as part of the application just like any other code.

**Advanced**. … split the entire system into self contained components and adopted a strict api-based approach to inter-communication so that each component can be deployed and released individually … every component is a self-contained releasable unit with business value, you can achieve small and frequent releases and extremely short release cycles..

**Expert …** some organizations will evolve the component based architecture further and value the perfection of reducing as much shared infrastructure as possible by also treating infrastructure as code and tie it to application components. The result is a system that is totally reproducible from source control, from the O/S and all the way up to application. …

Simplified pipeline

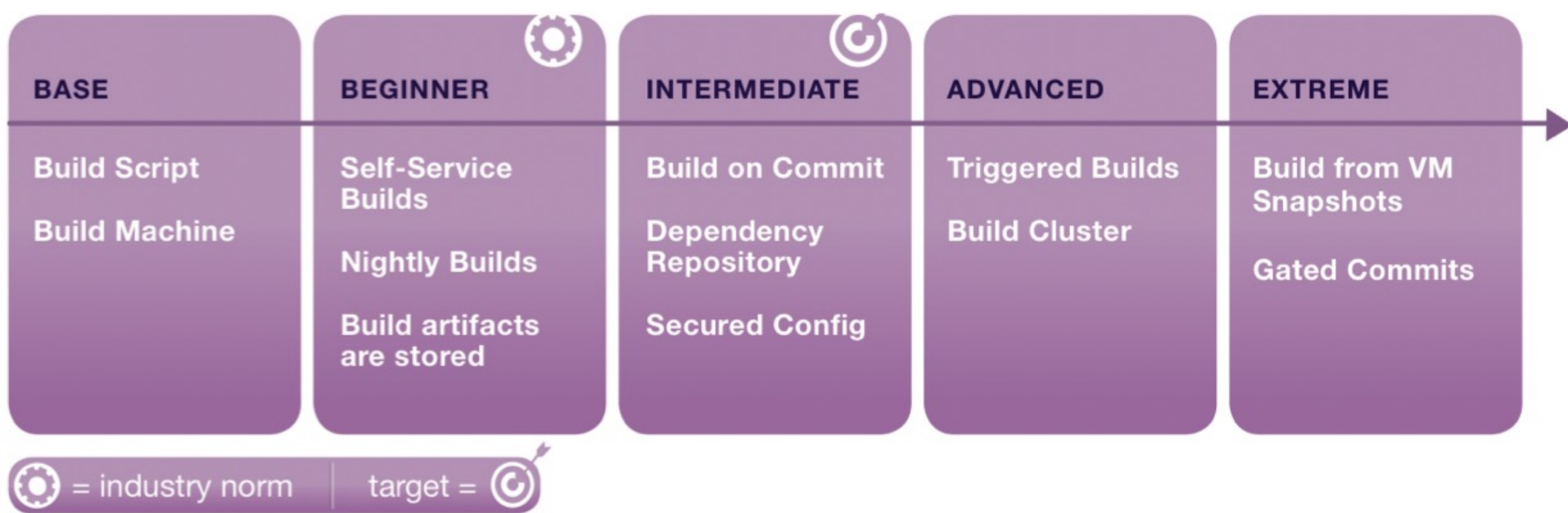Develop & test → VMS → Build → Test → Pack → Deploy → Operate
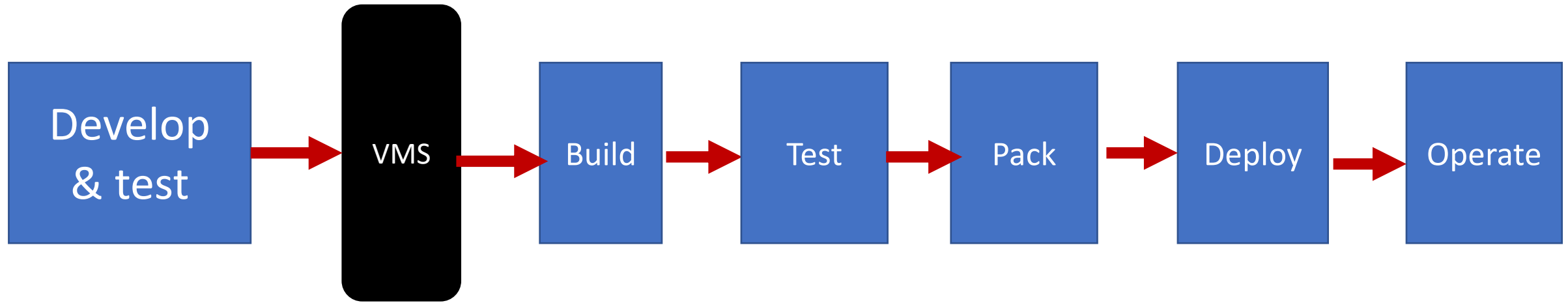
C++
Python

# Build – which tools you know ?

- Make
  - Old
  - Declarative
  - Hard to debug
- Ant
  - Designed for Java
  - Based on XML-based configuration language
- Maven

# BUILDING



| BASE | BEGINNER | INTERMEDIATE | ADVANCED | EXTREME |
|------|----------|--------------|----------|---------|
| Build Script | Self-Service Builds | Build on Commit | Triggered Builds | Build from VM Snapshots |
| Build Machine | Nightly Builds | Dependency Repository | Build Cluster | Gated Commits |
| | Build artifacts are stored | Secured Config | | |

⚙ = industry norm | target = ⟲

# Testing

- Automate, automate, automate
- Know any tools?

**Business**

**Support coding**

**Critique project**

| AUTOMATED<br><br>(functional acceptance tests) | MANUAL<br><br>Showcase<br>Usability testing<br>Exploratory testing |
|---|---|
| Unit tests<br>Integration tests<br>System tests<br><br>AUTOMATED | Nonfunctonal<br><br>acceptance tests<br><br>MANUAL/AUTOM. |

**Technology**

# TESTING

| BASE | BEGINNER | INTERMEDIATE | ADVANCED | EXTREME |
|------|----------|--------------|----------|---------|
| Some Test Automation | Significant test execution at build time | Some Static Analysis<br><br>Automated Functional Tests run nightly | High Code Coverage (expansive tests)<br><br>Security scans<br><br>Risk based manual testing | 100% Coverage |

⚙ = industry norm | target = ↻
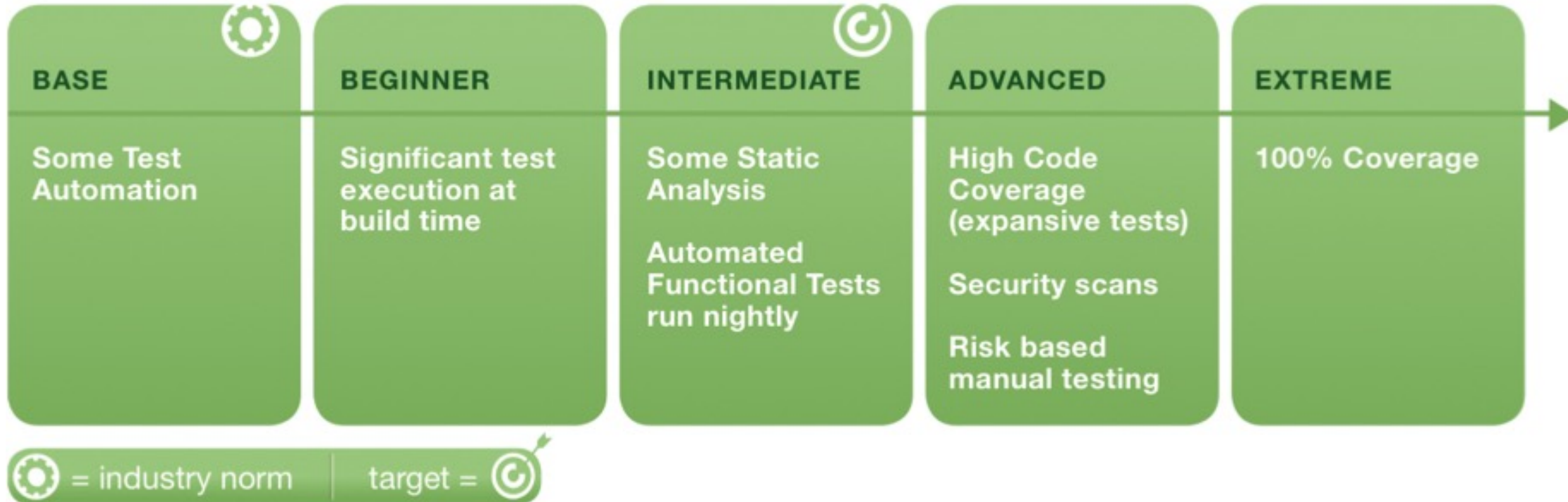
# Pack

- Binaries
- Required Libraries
- Runtime (e.g. Python)
- Manifest file
- Help files
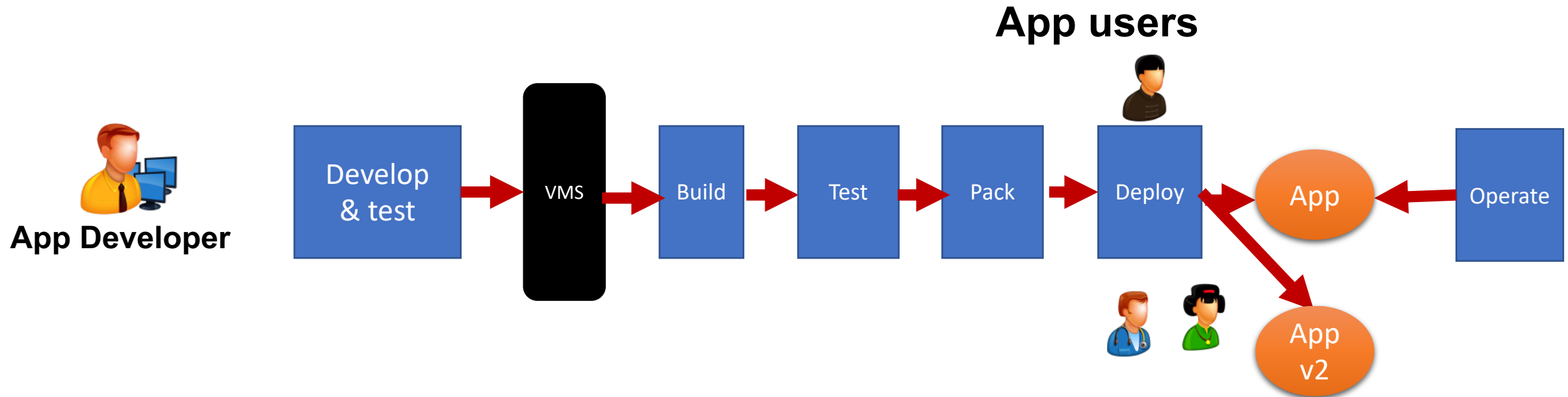- Localization stuff
- …

- Examples
  - Windows install shield
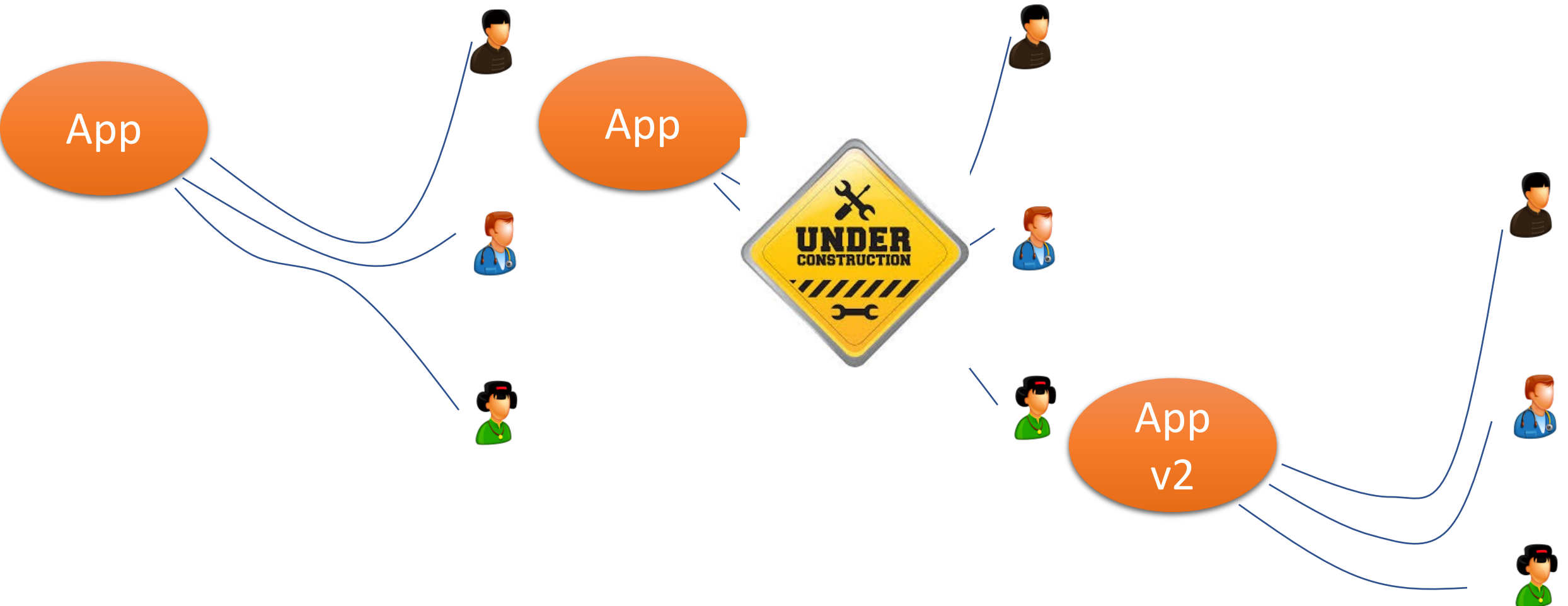  - Java JAR
  - Android APK
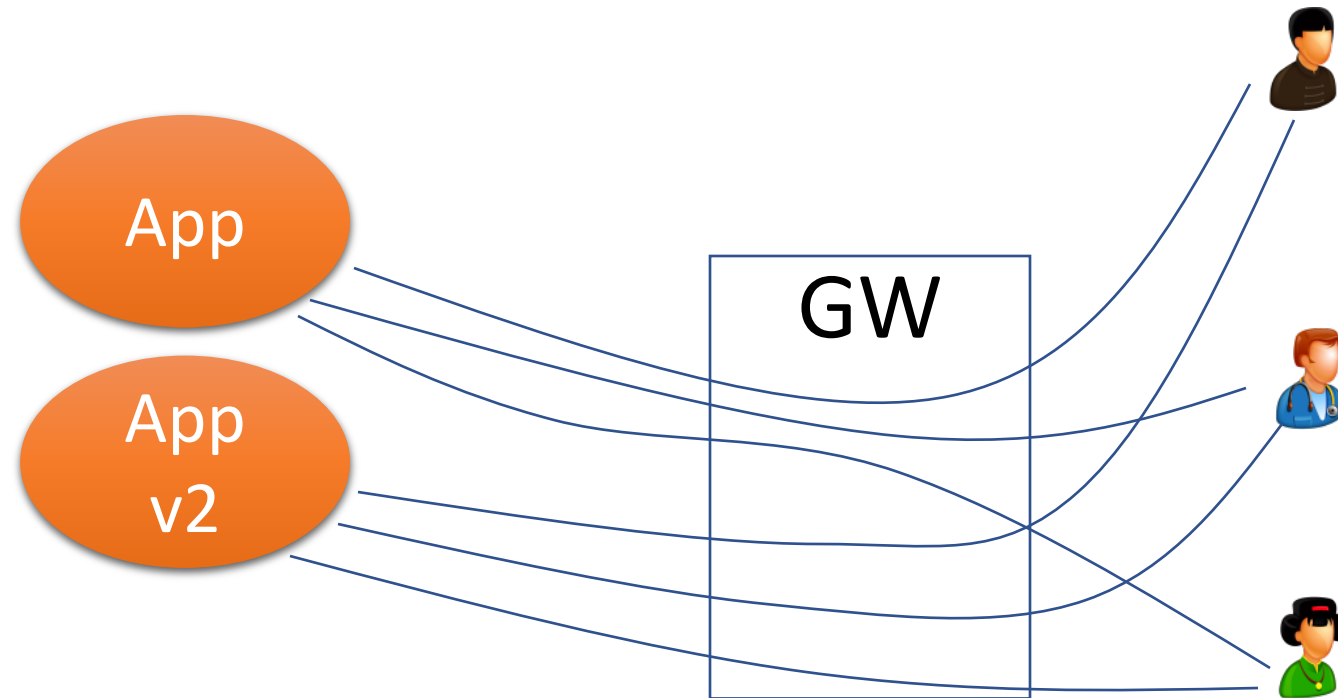
# What else comes to mind?

# Deployment/Delivery

- Humble and Farley write
  - Creating the infrastructure (hardware, networking, middleware, …)
  - Installing correct version of the application
  - Configuring the application with its data

- Sounds a bit difficult?

- Text written before 2011

- First Docker release 2013

A possible strategy to deploy a new version?

**Problems & issues?**
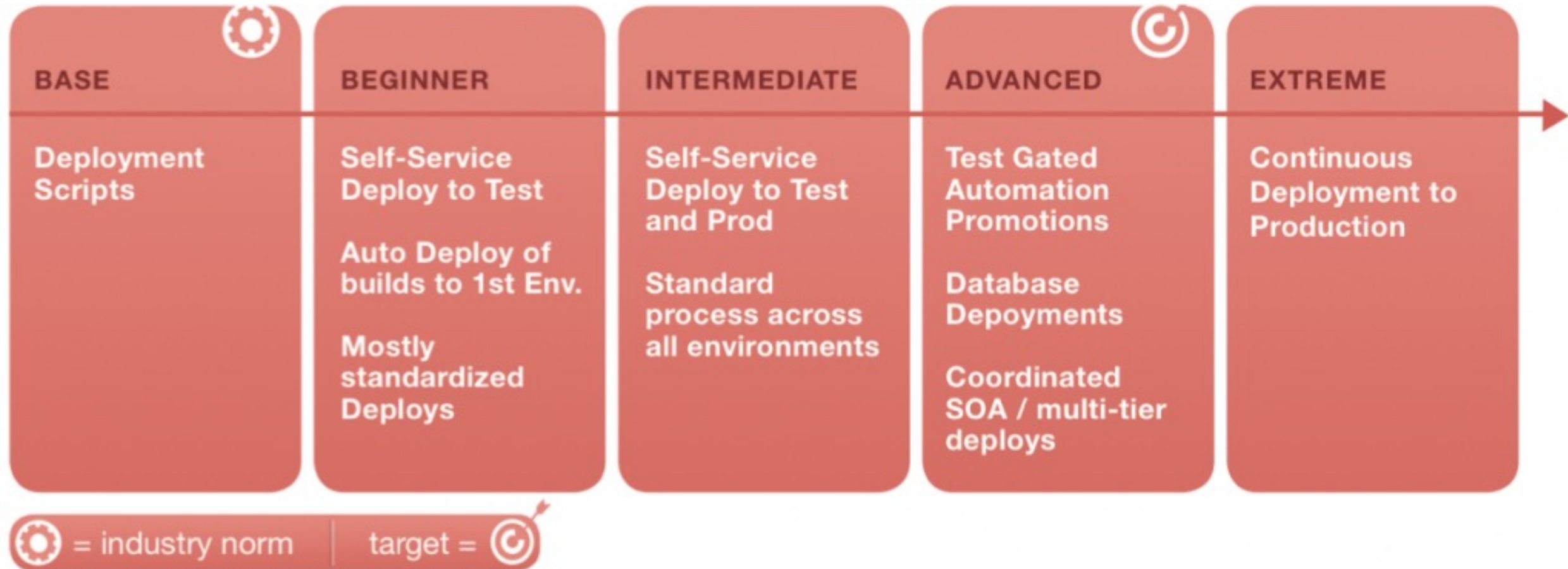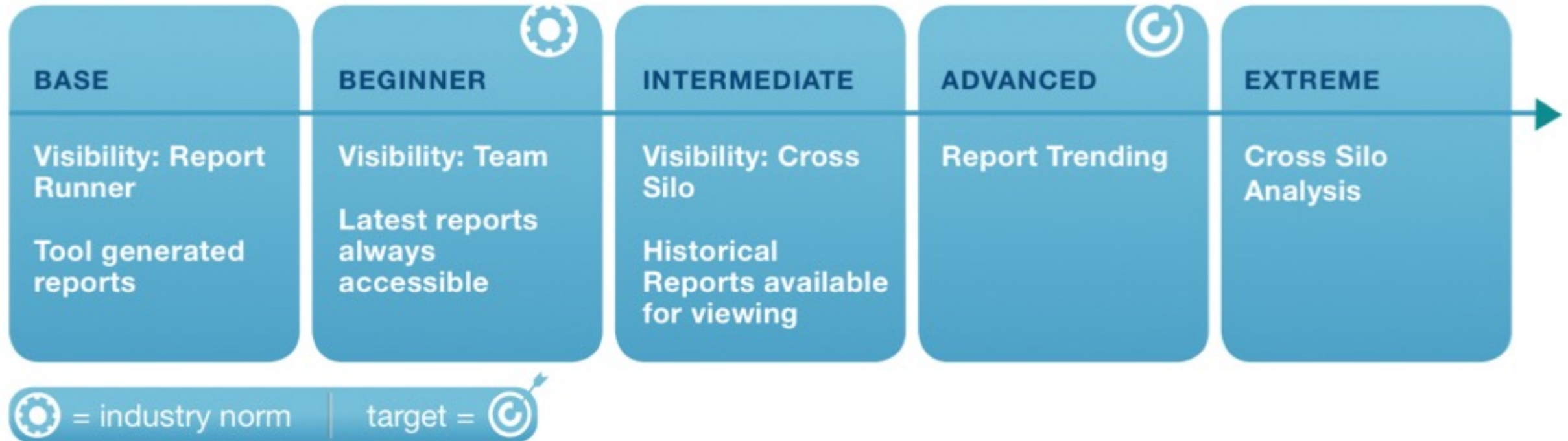
# DEPLOYING



| BASE | BEGINNER | INTERMEDIATE | ADVANCED | EXTREME |
|------|----------|--------------|----------|---------|
| Deployment Scripts | Self-Service Deploy to Test<br><br>Auto Deploy of builds to 1st Env.<br><br>Mostly standardized Deploys | Self-Service Deploy to Test and Prod<br><br>Standard process across all environments | Test Gated Automation Promotions<br><br>Database Depoyments<br><br>Coordinated SOA / multi-tier deploys | Continuous Deployment to Production |

= industry norm | target =

(https://www.urbancode.com/resource/continuous-delivery-maturity-model/
was at developer.ibm.com)

# REPORTING

| BASE | BEGINNER | INTERMEDIATE | ADVANCED | EXTREME |
|------|----------|--------------|----------|---------|
| Visibility: Report Runner | Visibility: Team | Visibility: Cross Silo | Report Trending | Cross Silo Analysis |
| Tool generated reports | Latest reports always accessible | Historical Reports available for viewing | | |

= industry norm | target =

# Deployment strategies

- Basic Deployment (aka Suicide) (https://harness.io/2018/02/deployment-strategies-continuous-delivery/) all nodes are updated at the same time

- Rolling Deployment (https://harness.io/2018/02/deployment-strategies-continuous-delivery/) nodes are updated incrementally,

- BlueGreenDeployment (http://martinfowler.com/bliki/BlueGreenDeployment.html) uses a router of incoming traffic as the tool. In this approach the new version (called green) is set up in parallel with the current (blue). When new (green) is ready, the router is switched to new (green) and blue is left as a backup. If something goes wrong with new, the router can be switched back to old - that means easy "rollback".

- Canary Releases (http://martinfowler.com/bliki/CanaryRelease.html ) implements the deployment incrementally. In this case the router first directs only part of the customers to the new version. If feedback is is good, the other customers are moved to new version, too
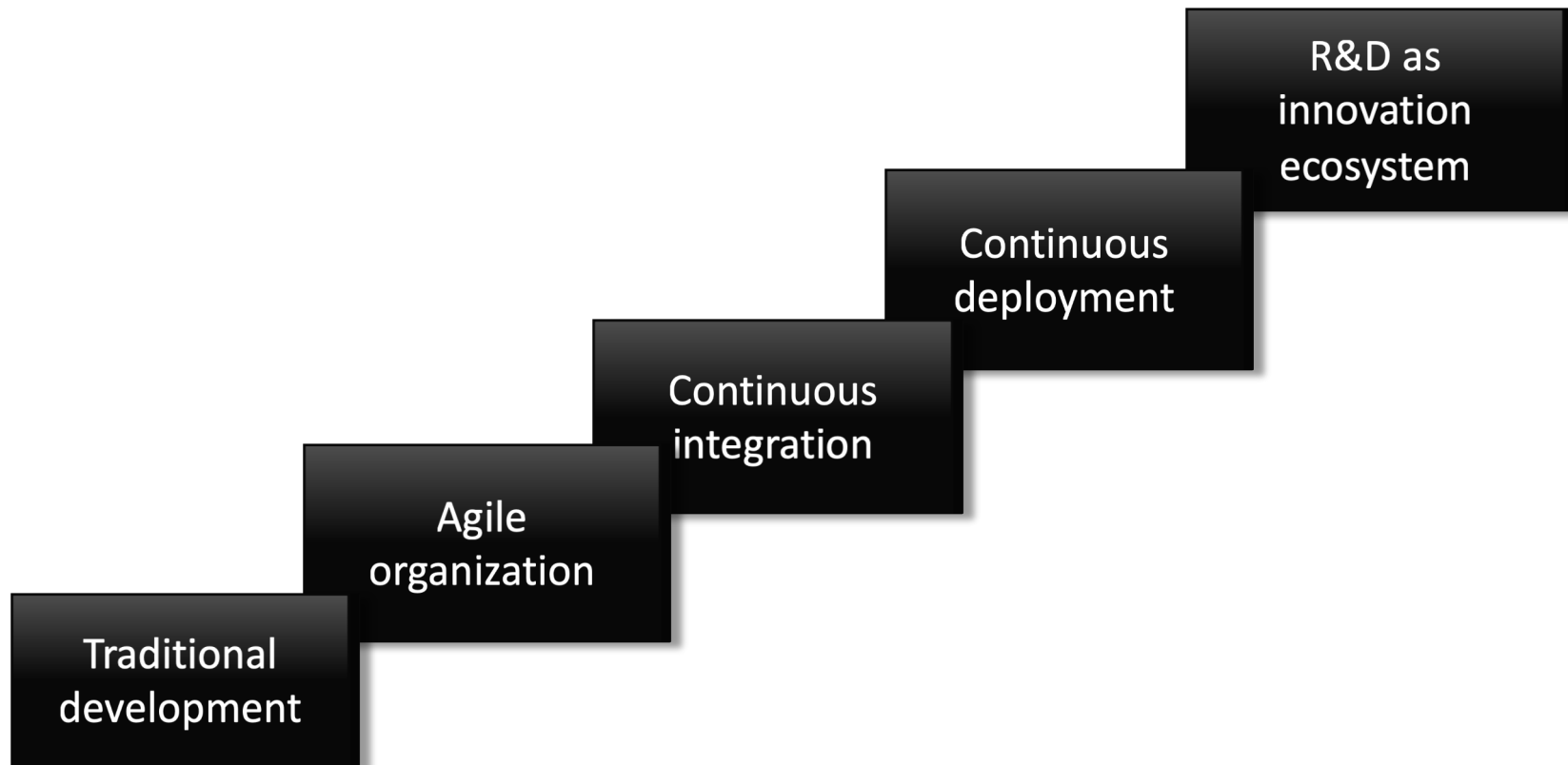
# How about the data?



Creation and initialization
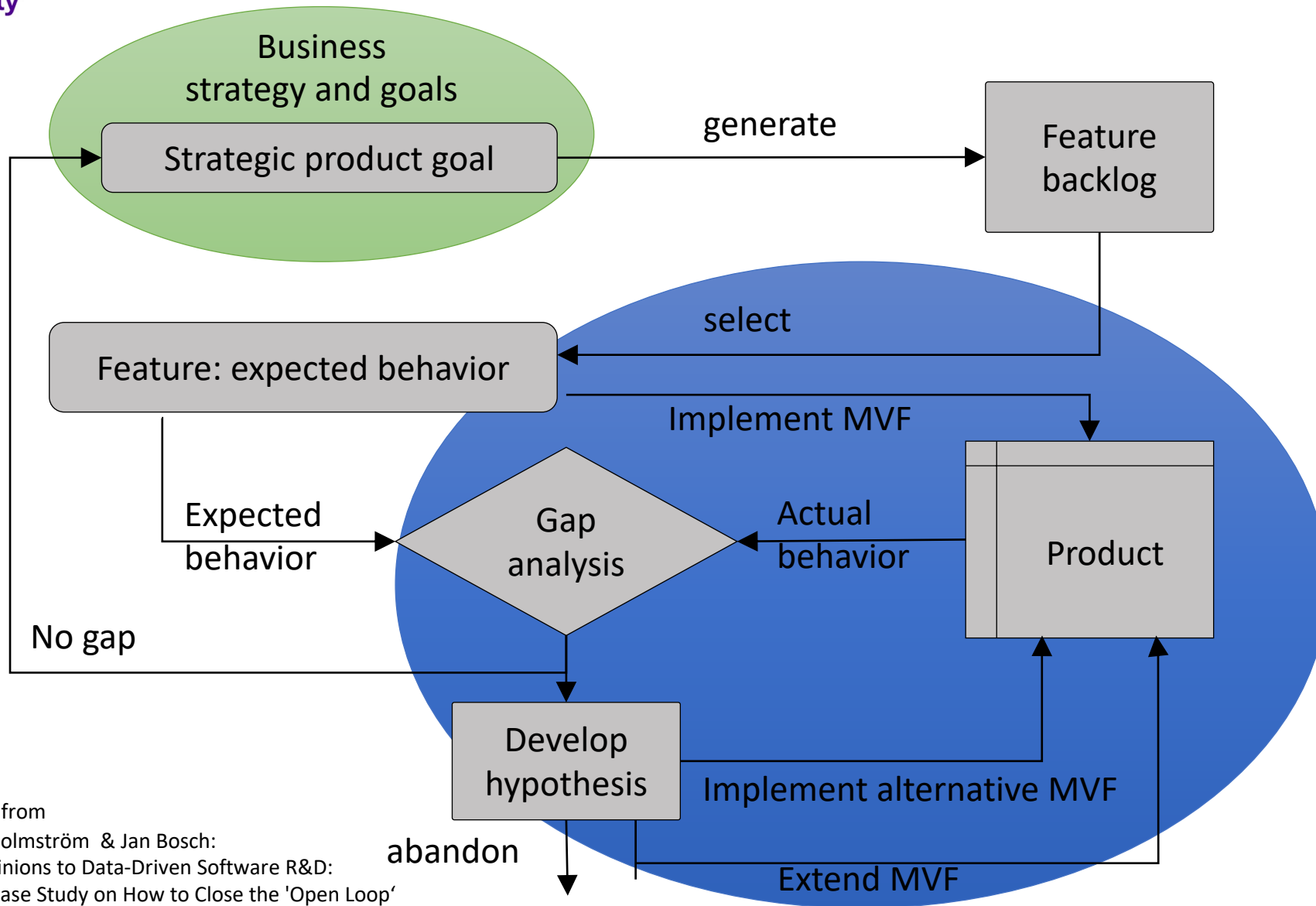
# A/B Testing

**?**

Should our project have A or B?

**!**

Implement a way to collect statistics

Implement A

Implement B

Deploy B

Deploy A

Usage statistics

Usage statistics

Compare
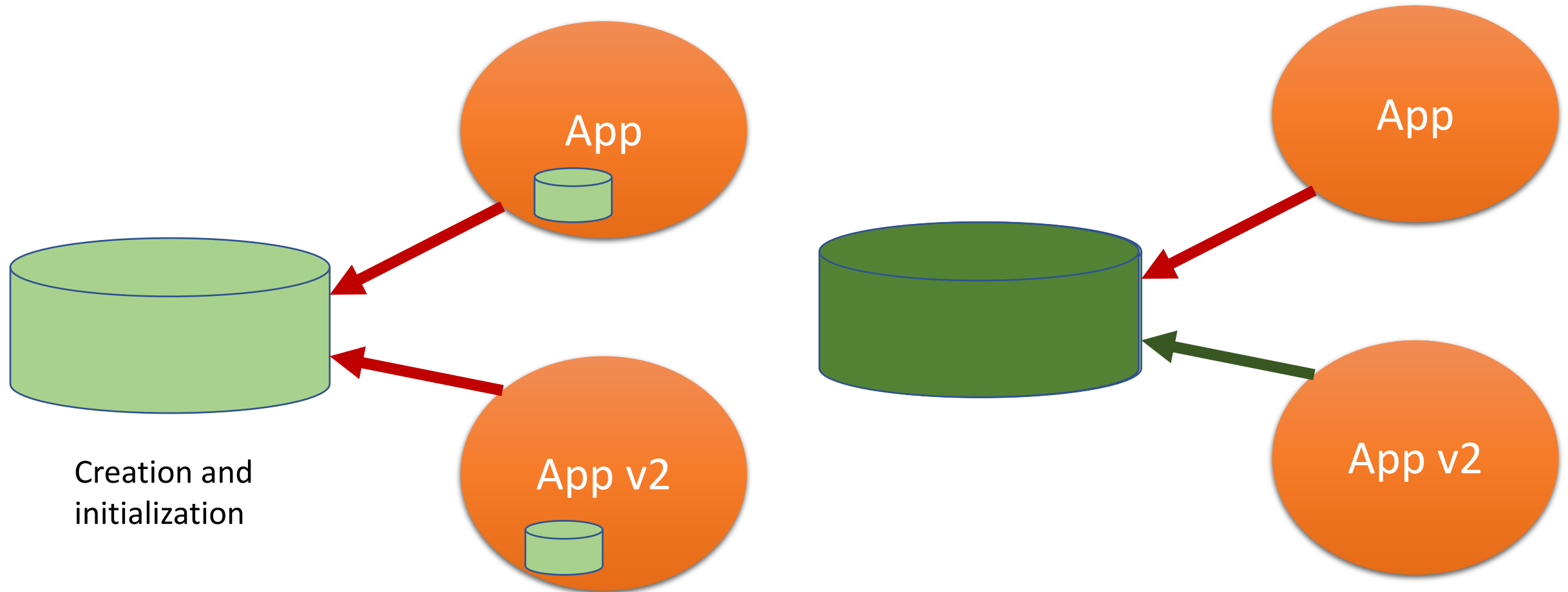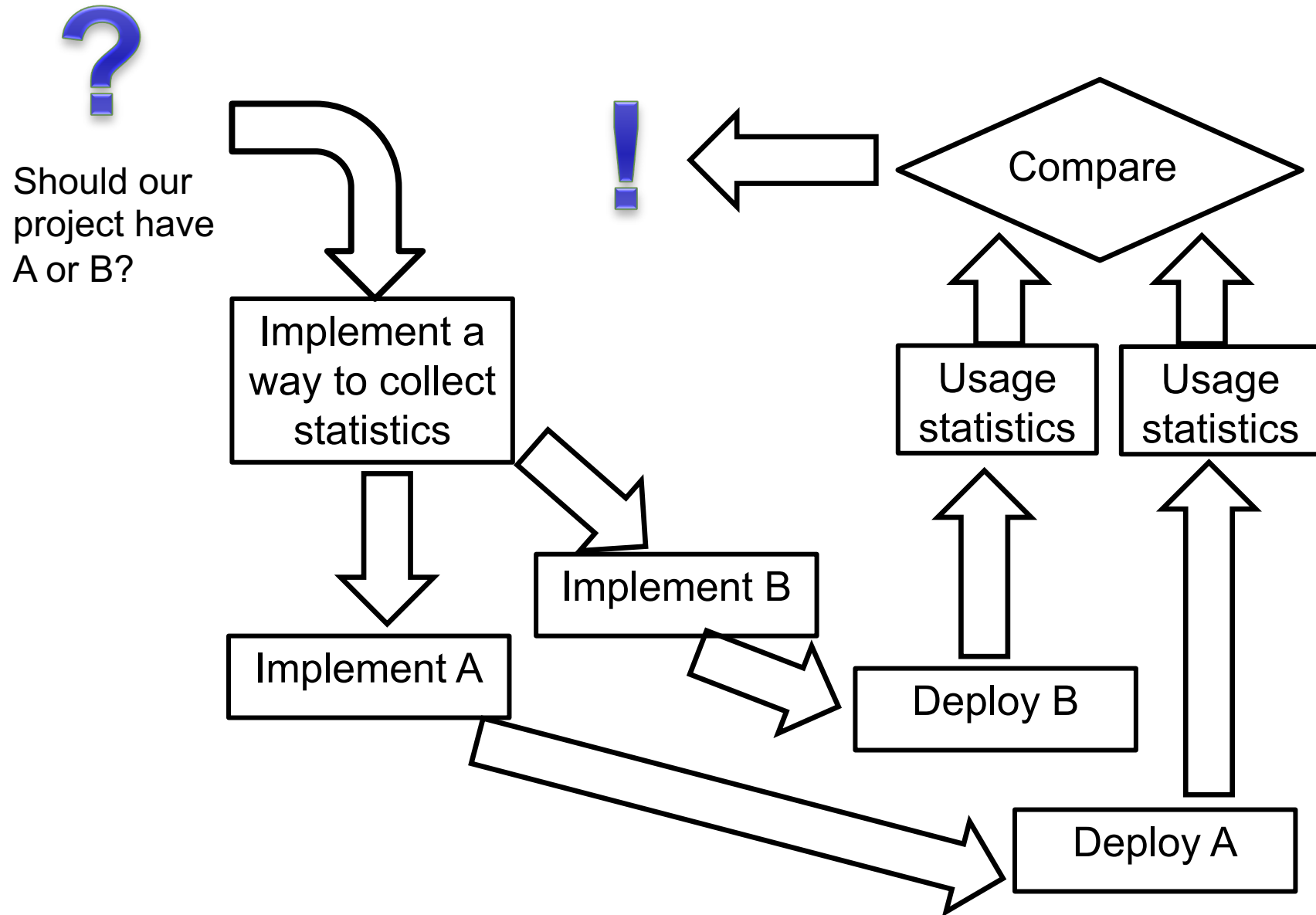
# Stairway to Heaven
# (As described by Jan Bosch)

The HYPEX model (**Hypothesis Experiment Data-Driven Development )**

Business strategy and goals

Strategic product goal

generate

Feature backlog

select

Feature: expected behavior

Implement MVF

Expected behavior

Gap analysis

Actual behavior

Product

No gap

Develop hypothesis

Implement alternative MVF

abandon

Extend MVF

Adopted from
Helena Holmström & Jan Bosch:
From Opinions to Data-Driven Software R&D:
A Multi-case Study on How to Close the 'Open Loop'
Problem

# Data-driven software development

1. Planning of the data collection
2. Deployment of data collection
3. Monitoring of the applications
4. Picking up the relevant data
5. Pre-processing – filtering and formatting – the data
6. Sending and/or saving the data
7. Cleaning and unification of the data
8. Storing the data
9. Visualizations and analysis
10. Decision making