# TOWARD CONTINUOUS DELIVERY OF A NONTRIVIAL DISTRIBUTED SOFTWARE SYSTEM

S. Wai, South African Radio Astronomy Observatory (National Research Foundation), Cape Town, South Africa

## INTRODUCTION

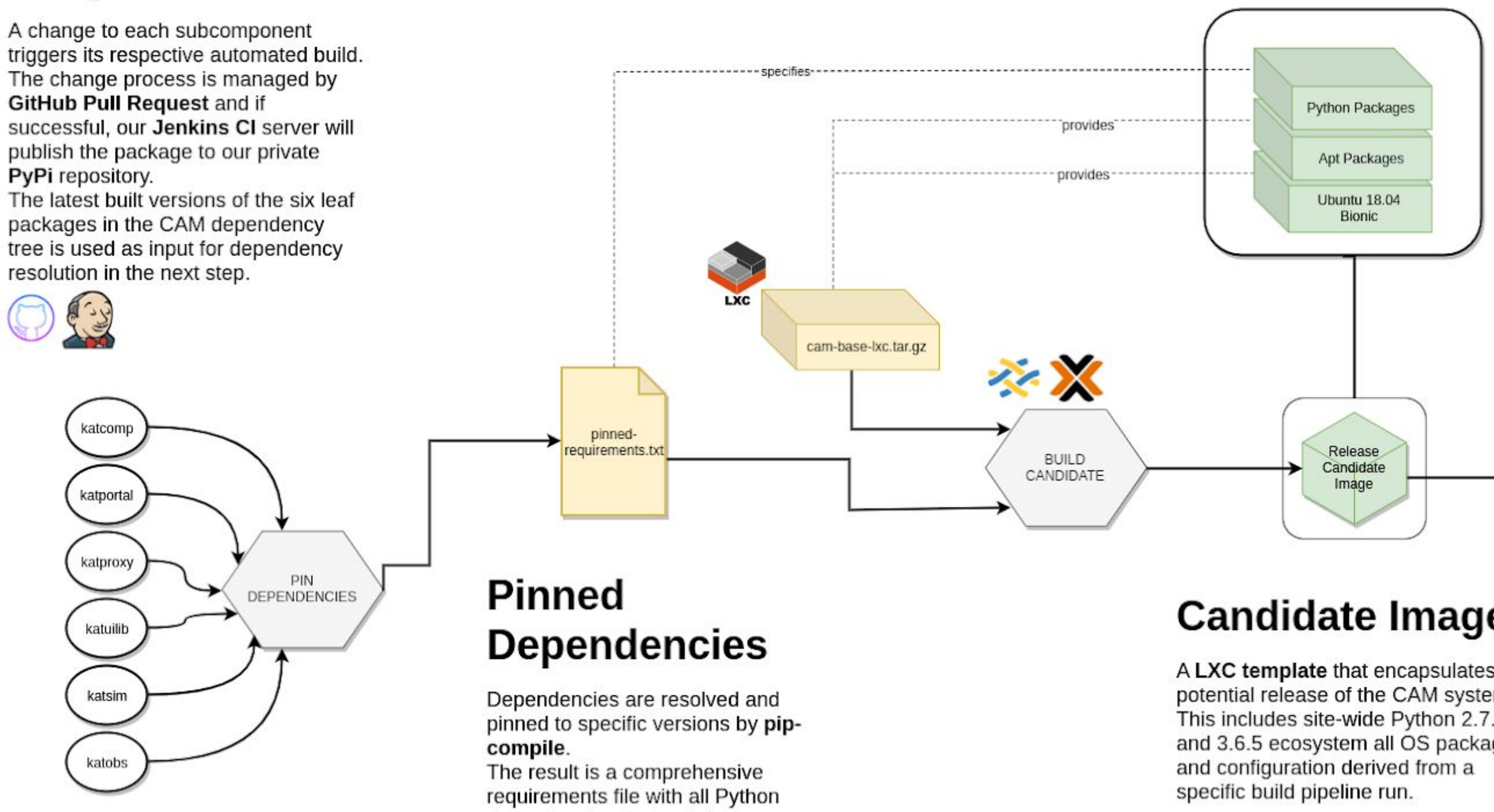
The meerKAT is a 64-dish radio telescope situated in the remote Karoo desert region of South Africa and is the reference implementation for the Square Kilometre Array, which will be the largest telescope in the world. It has a sophisticated control and monitoring system that has leveraged virtualisation technologies and automation from the start. This has enabled the adoption of software development practices such as **continuous integration** and **automated acceptance testing** which allowed for rapid, incremental development of the system.
Despite this, some *last mile* problems were encountered around release which prompted efforts to further optimise its build and deployment process.

Optimising the build and deployment of the MeerKAT radio telescope Control and Monitoring Subsystem

### Continuous Integration

A change to each subcomponent triggers its respective automated build. The change process is managed by **GitHub Pull Request** and if successful, our **Jenkins CI** server will publish the package to our private **PyPi** repository.
The latest built versions of the six leaf packages in the CAM dependency tree is used as input for dependency resolution in the next step.

## BUILD



### Pinned Dependencies

Dependencies are resolved and pinned to specific versions by **pip-compile**.
The result is a comprehensive requirements file with all Python dependencies that satisfy the requirements of the most recent successful builds of *katcomp, katproxy, katportal, katuilib, katsim, katobs.*
This will implicitly depend on the rest of the CAM Python packages that could potentially make a coherent system.

### Candidate Image

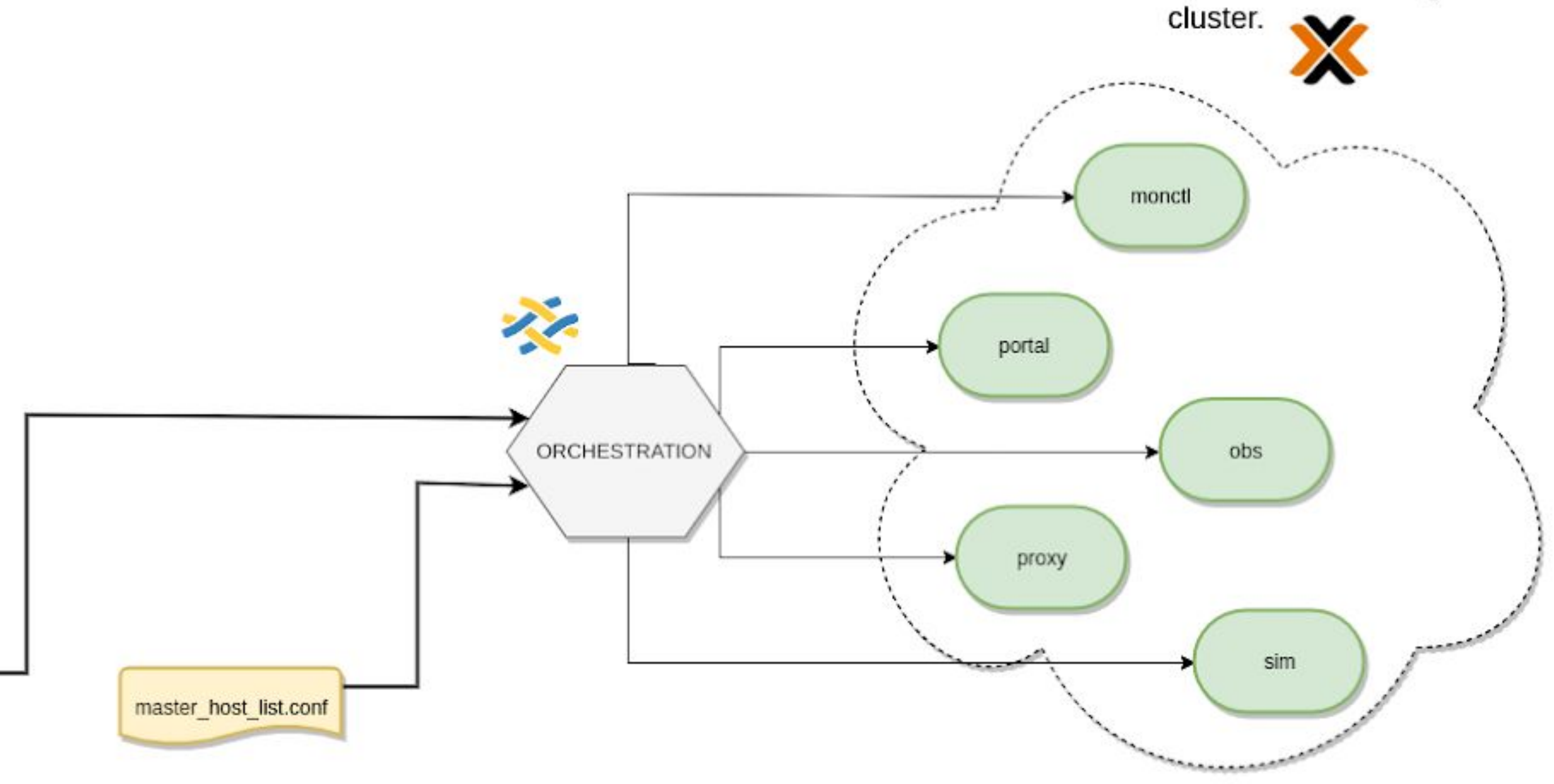
A **LXC template** that encapsulates a potential release of the CAM system. This includes site-wide Python 2.7.15 and 3.6.5 ecosystem all OS packages and configuration derived from a specific build pipeline run.

Every candidate can be deployed to produce a full CAM system, which can then be verified by our Automated Qualification Framework (integration tests and acceptance tests).

## DEPLOY

### Environments

Nodes are deployed as **LXC containers** to a target Proxmox cluster.



### Node Configuration

The node configuration describes a system in terms of a number of nodes that belong to it and the *roles* assigned to them. The deployment scripts will run some operations based on the *roles* in order to specialise them.

This file is the source of truth for all deployed systems and  checked into source control.

### Homogenous Nodes

All nodes are created from the same image and are only differentiated during deployments based on the **master_host_list.conf**.

This is a key design principle that greatly simplifies the release process.

## THE “LAST MILE”

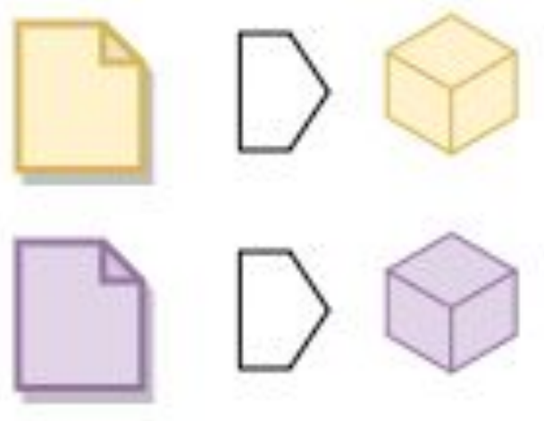
Issues that crop up between integrated changes (code that has been merged into trunk/mainline) and their release to the production environment. These problems results in failures, and troubleshooting efforts during release time. Some informal analysis revealed the following issues:

1. Repeatable deployments not guaranteed due to
   a. unpinned transient dependencies
   b. late convergence of system baseline state by imperative deployment scripts
2. Automated testing did not exercise full deployment procedure
   a. tests ran in a static, long-lived environment/node
3. Unclear separation of *build* and *run* stages
   a. version control release branching scheme used to manage deployments

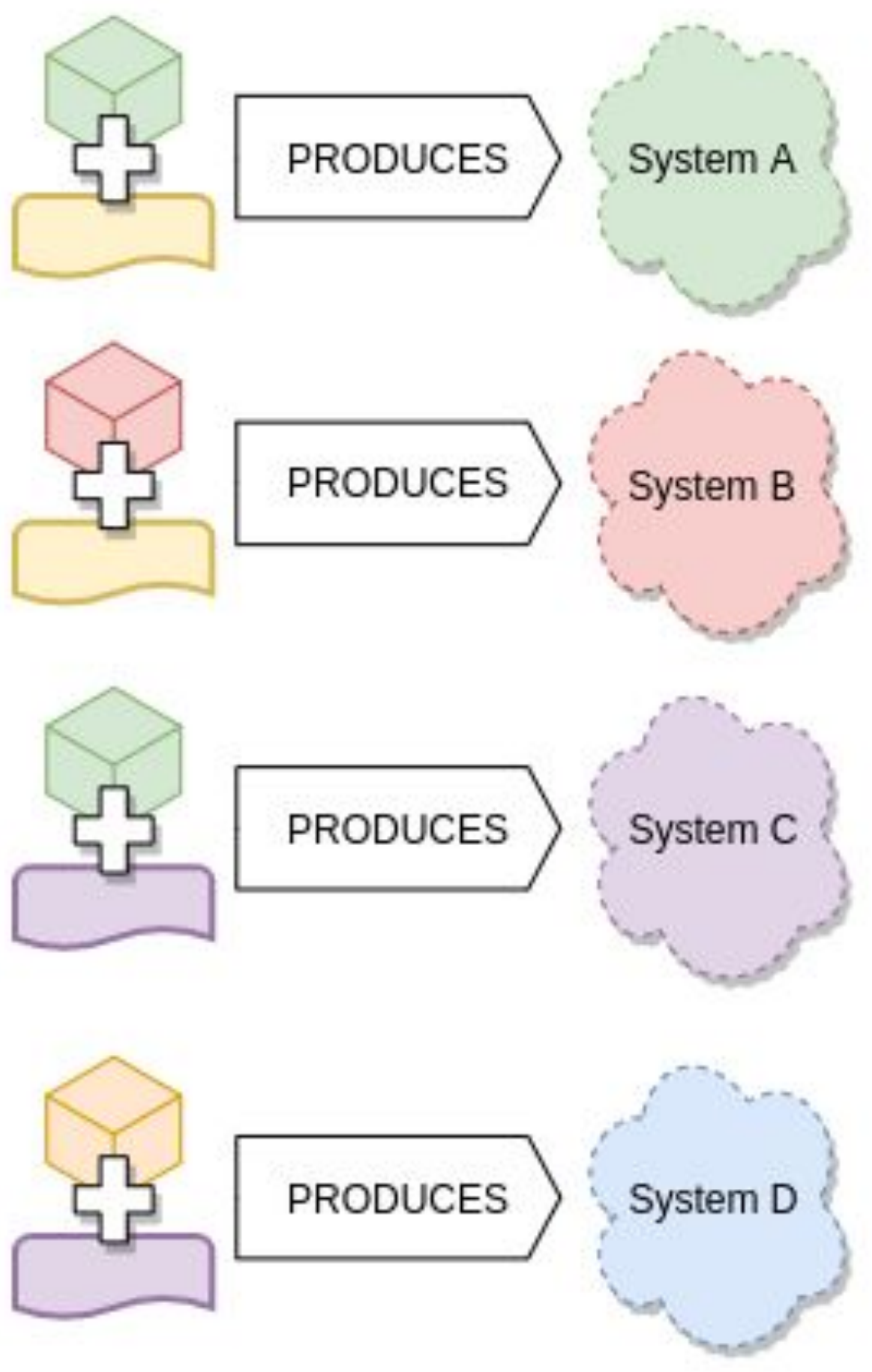
### Repeatable Builds And Deployments

The same *candidate* and *configuration* configuration will produce the same system.

Furthermore, a set of pinned requirements produces the same *candidate:*



This property of **determinism** allows us to increase confidence in changes.



PRODUCES
System A
PRODUCES
System B
PRODUCES
System C
PRODUCES
System D