Continuous Automated Deployment with

Apache ACE

luminis

Conversing worlds

Who are we

Jan Willem Janssen

Software Architect at Luminis Technologies
committer and PMC member of Apache ACE

@j_w_janssen

Marcel Offermans

Director at Luminis Technologies
Member at the Apache Software Foundation

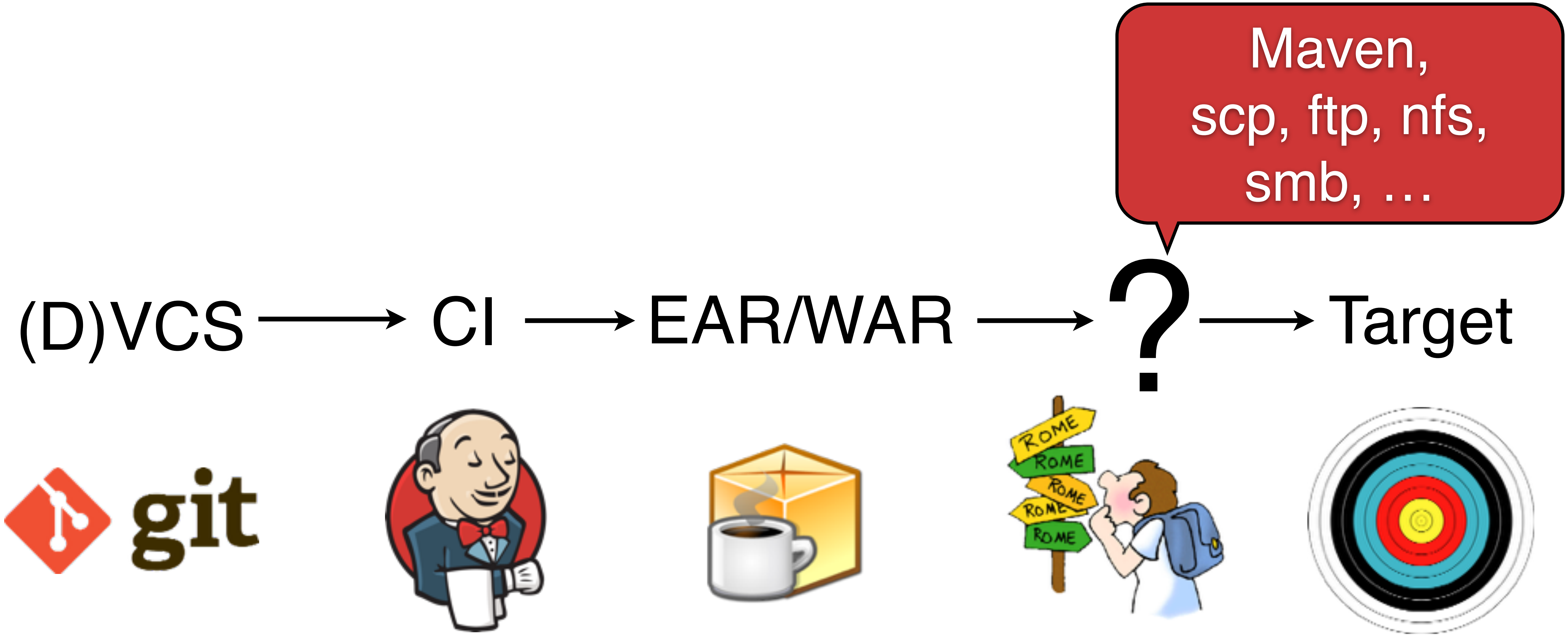
@m4rr5

Agenda

- Traditional Continuous Integration
- From EAR to bundles
- Versioning and baselining
- ACE basics
- Build env. setup with ACE
- Demo

Typical CI workflow

without OSGi and ACE



Maven,
scp, ftp, nfs,
smb, …

(D)VCS → CI → EAR/WAR → ? → Target

Monolithic to Modular

- Single, big, monolithic application
- Versioned whenever “something” changes
- On every change the whole application has to be redeployed

- Many, small modules that can be combined to form applications
- Modules versioned independently
- Only modules that changed need to be redeployed

What to version?

Bundles

So we know their contents has changed

Exported Packages

So we know the interface contract has changed

our focus here is on **bundle versions**, because they tell us **what to update**

Semantic Versioning

tells us what has changed

```
1.0.0.abc
major.minor.micro(.qualifier)
```

- **Major**: Backward incompatible change
- **Minor**: Backward compatible change
- **Micro**: Implementation change
- **Qualifier**: Label, e.g. build number

Baselining

- Compares build with latest release
- Checks if version numbers should be bumped
- Uses byte code analysis

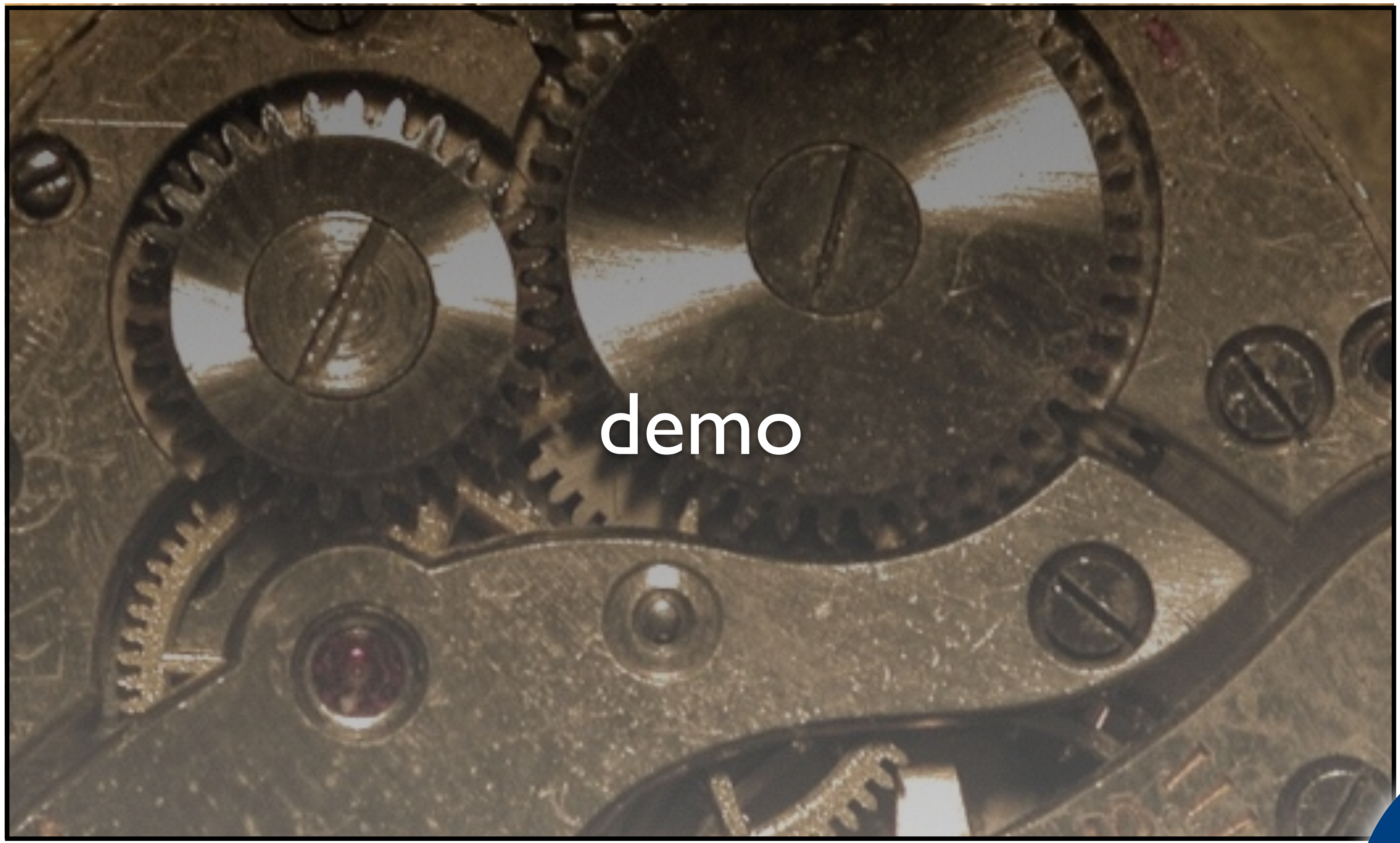
Eclipse with Bndtools

```
-baseline: *
-removeheaders: Bnd-LastModified,Tool,Created-By
```

Baselining

- Bnd annotations
  - `@ProviderType`
    `A type that is provided by the implementor of the contract.`
  - `@ConsumerType`
    `A type that is typically implemented by the consumer of the contract.`

Eclipse with Bndtools

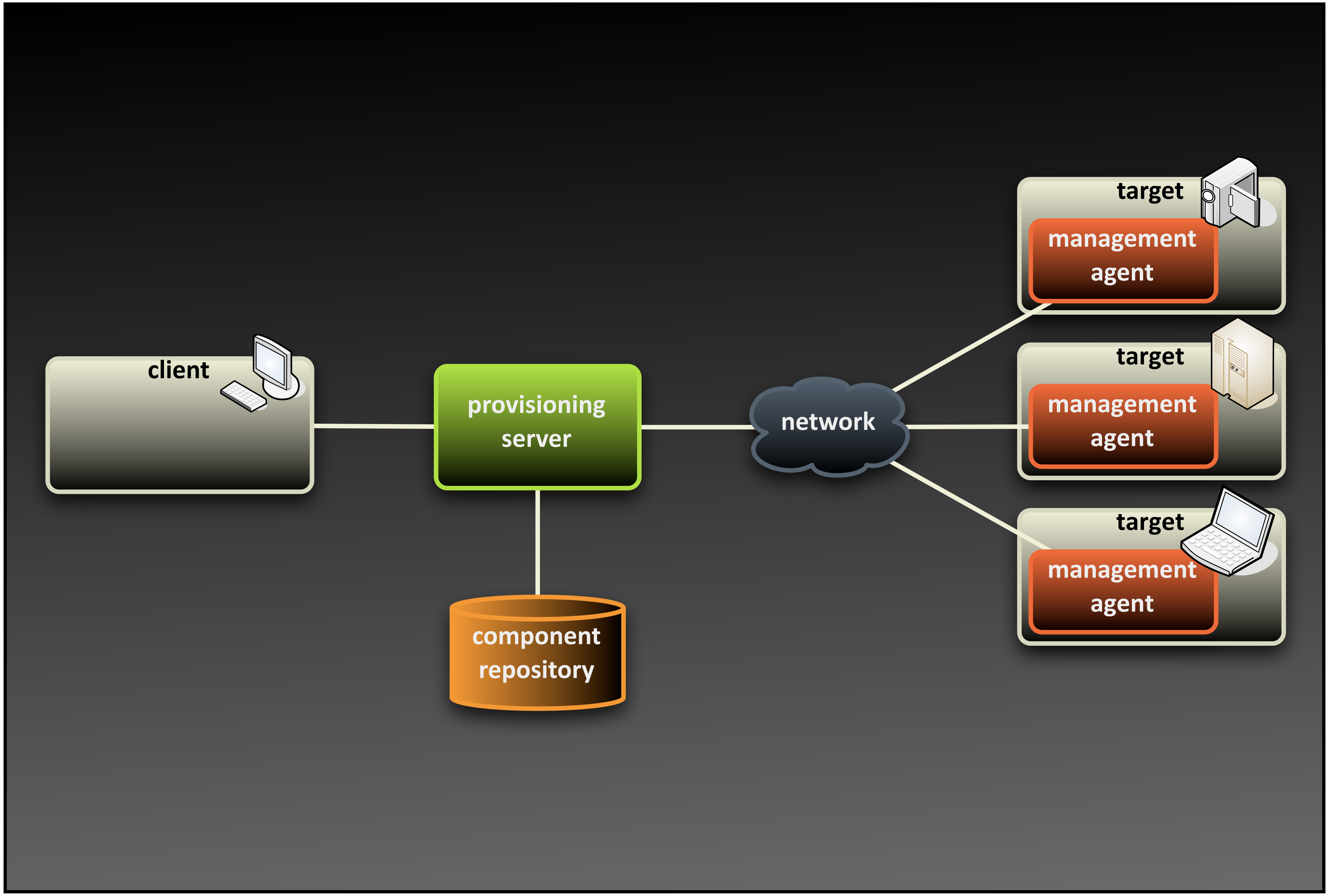


bndtools

Apache ACE

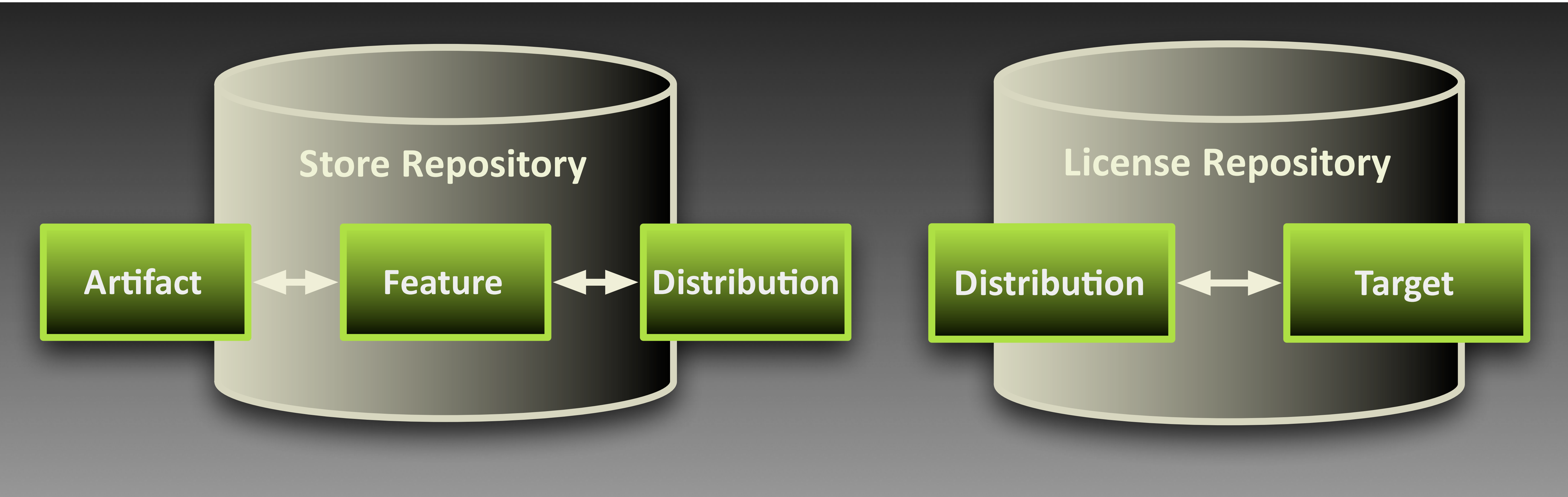
- Software distribution framework
- Manage installation/upgrade of targets
    - bundles
    - configurations
    - etc

ACE basics

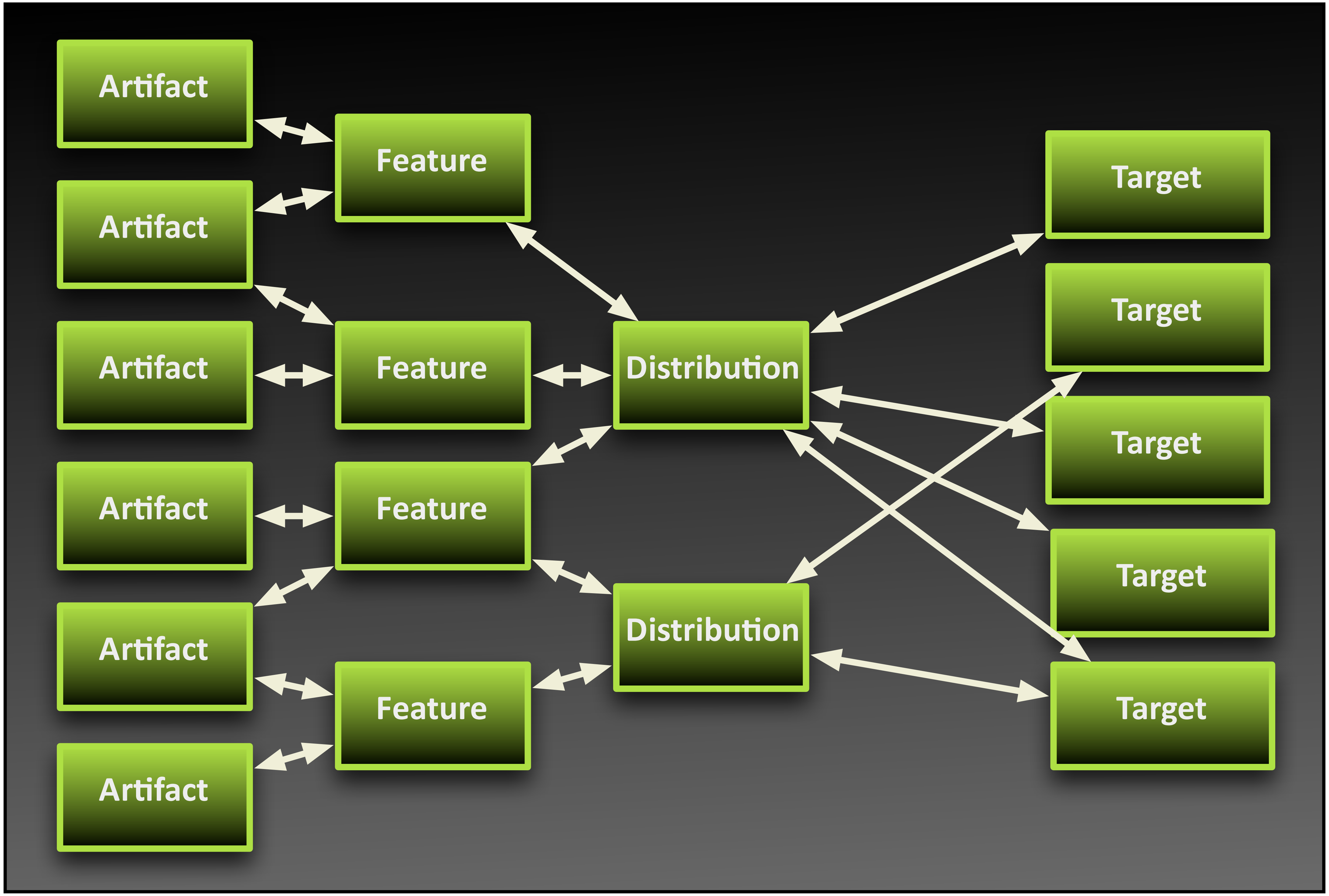

target
management agent
target
management agent
client
provisioning server
network
target
management agent
component repository

ACE basics

- group artifacts into features and distributions to make them manageable
  - IKEA catalogue



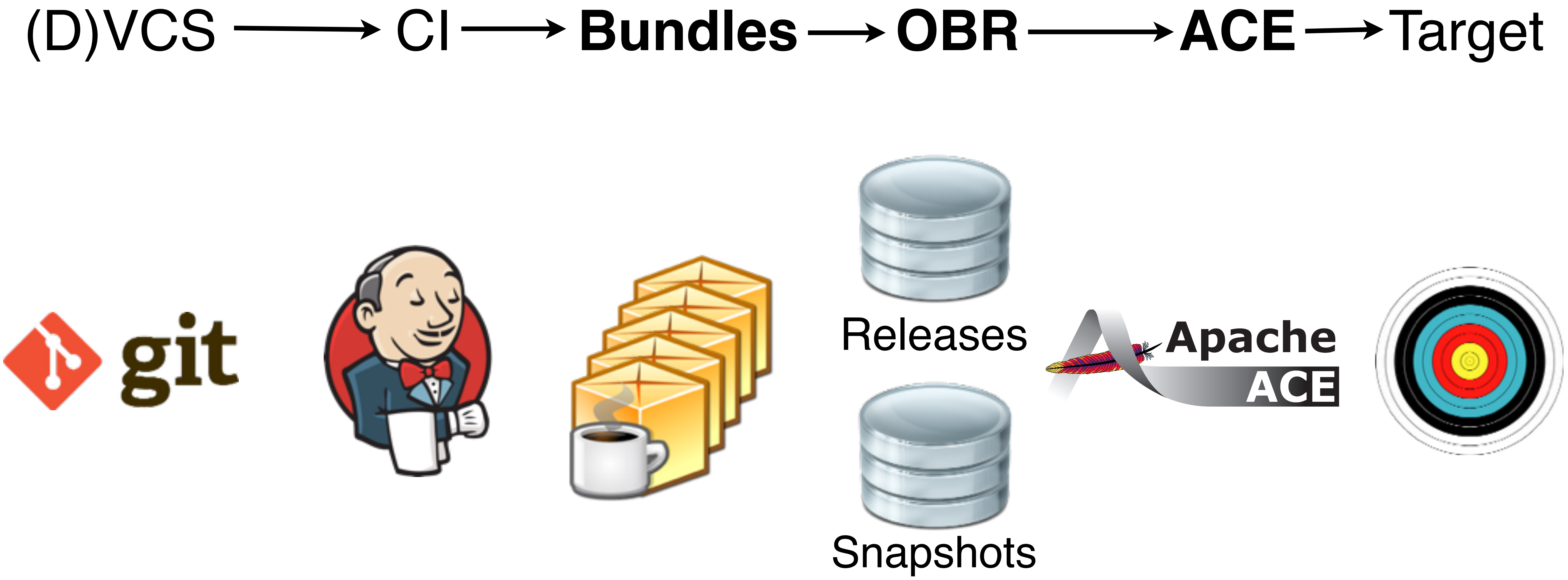
Organising artifacts



Apache ACE

CI workflow

with OSGi & ACE

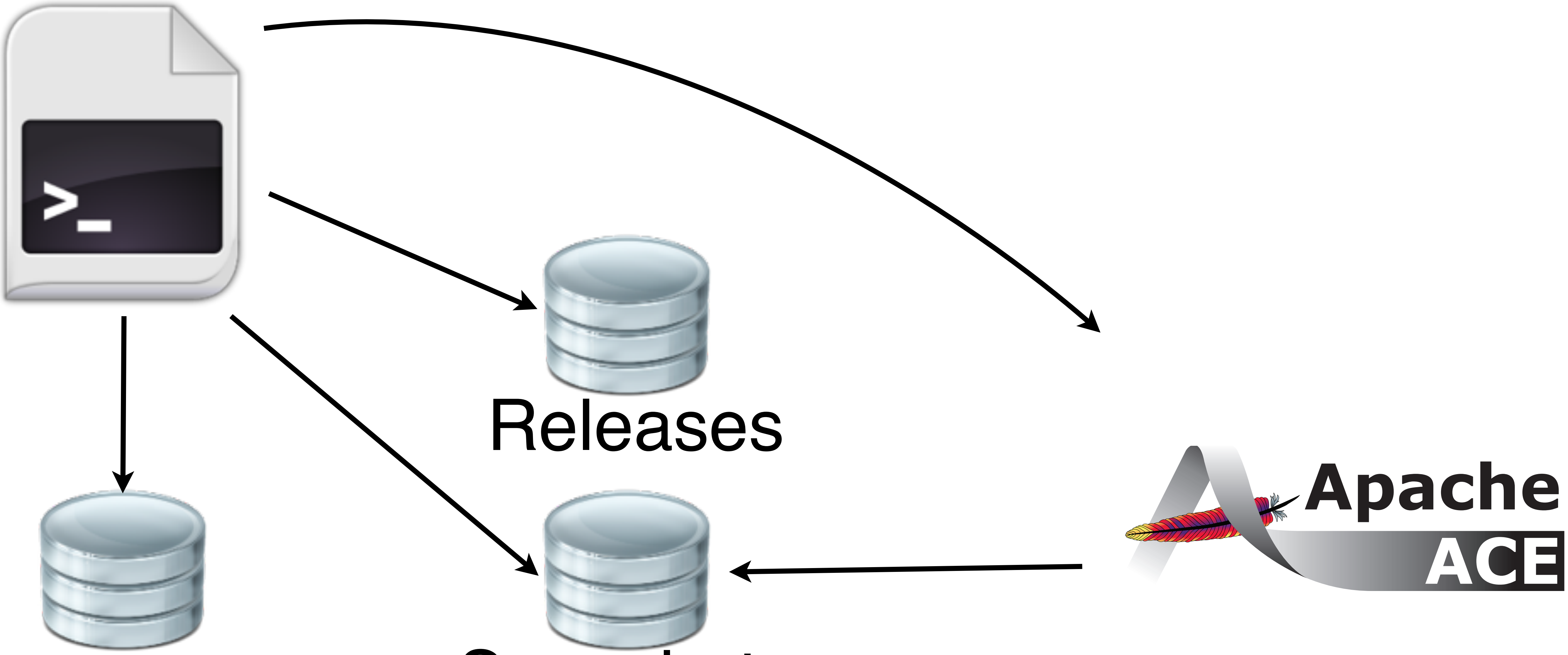


(D)VCS → CI → **Bundles** → **OBR** → **ACE** → Target

Releases

Snapshots

Build env. with ACE



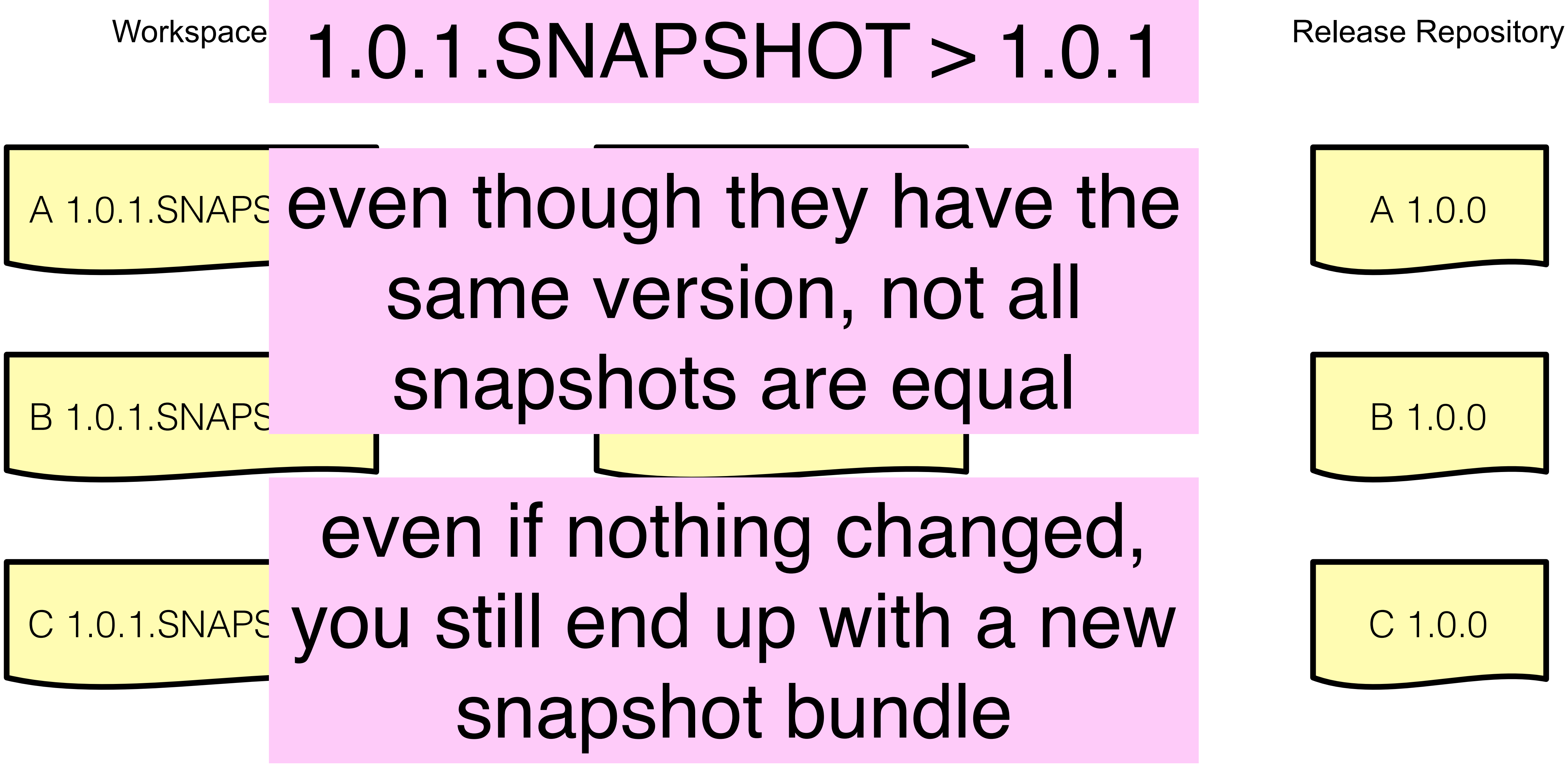
Releases

Snapshots

Bundles from build

Versioning bundles from a build

Assume only bundle A changed since the release



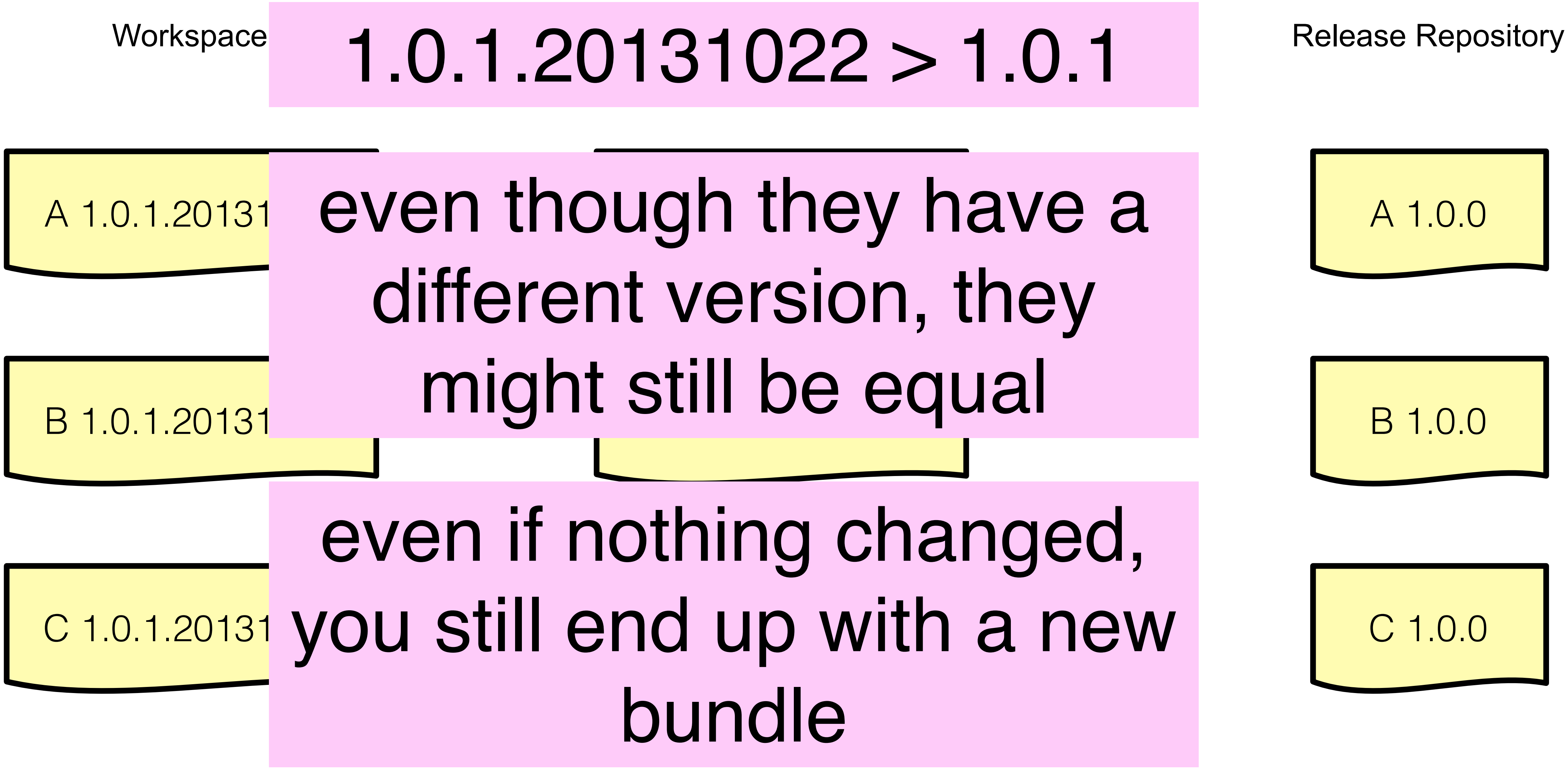
1.0.1.SNAPSHOT > 1.0.1

even though they have the same version, not all snapshots are equal

even if nothing changed, you still end up with a new snapshot bundle

Versioning bundles from a build

Assume only bundle A changed since the release



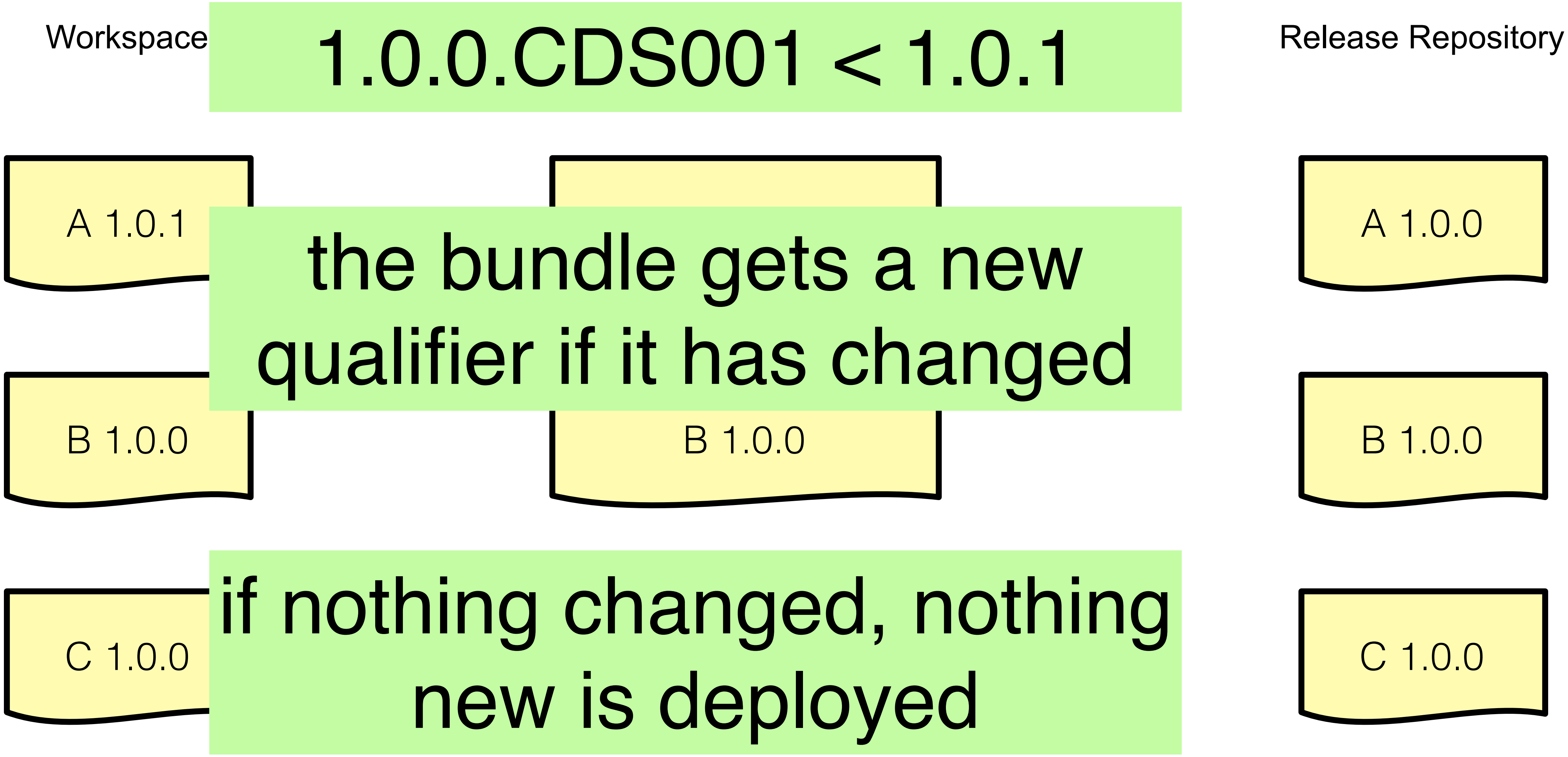
1.0.1.20131022 > 1.0.1

even though they have a different version, they might still be equal

even if nothing changed, you still end up with a new bundle

Versioning bundles from a build

Assume only bundle A changed since the release



1.0.0.CDS001 < 1.0.1

the bundle gets a new qualifier if it has changed

if nothing changed, nothing new is deployed

What do we need?

CI

Script that places bundles in the OBR

ACE

Server with snapshot and release OBR

Target

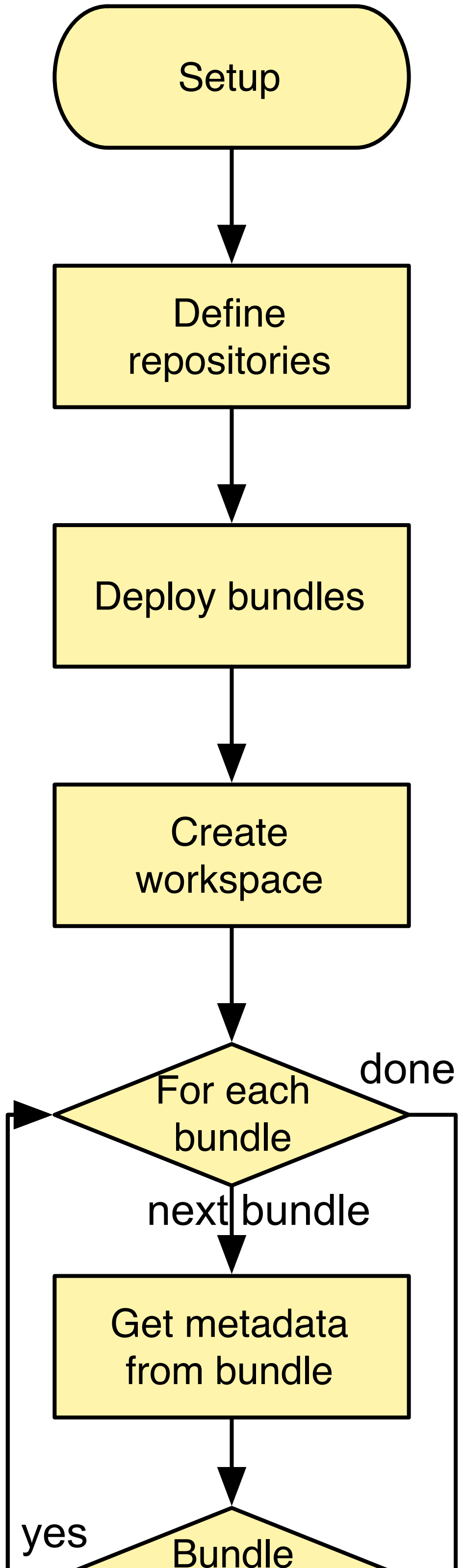
Management agent configured to the ACE instance

GoGo shell

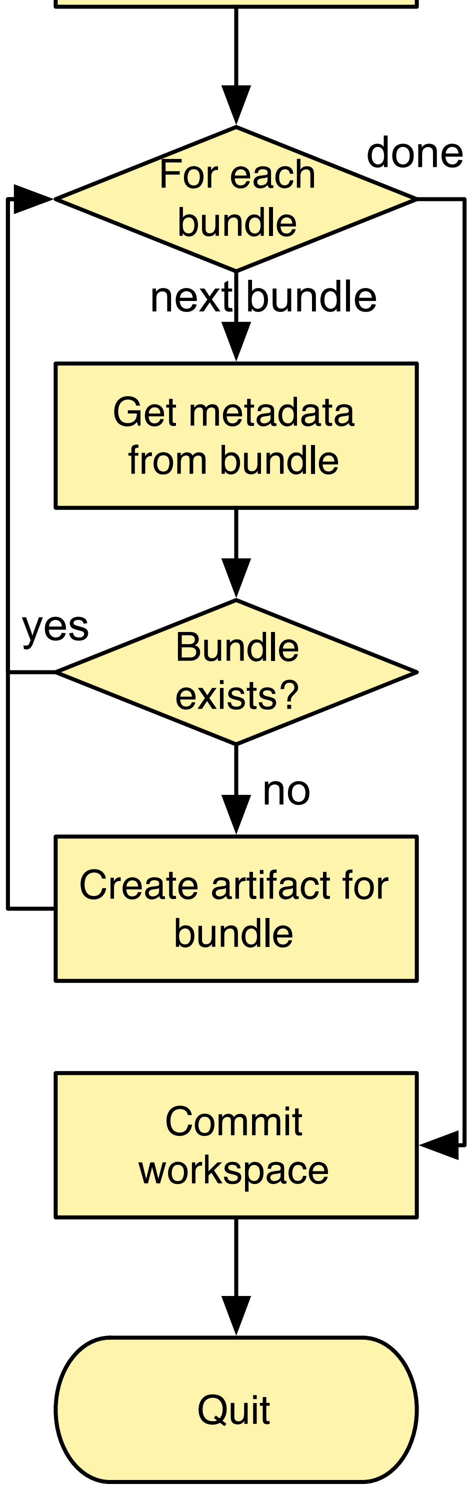
- “Standard” OSGi shell (RFC-147)
- Powerful and extensible
- Available from Apache Felix
- ACE provides commands to interact with its client API and OBRs

Script

```
echo “Define repositories”
sourceindex = (repo:index ../release)
sourcerepo  = (repo:repo R5 $sourceindex)
targetrepo = (repo:repo OBR "http://localhost:8084/obr/repository.xml")
releaserepo = (repo:repo OBR "http://localhost:8083/obr/repository.xml")

echo "Deploying bundles"
deployed = repo:cd $releaserepo $sourcerepo $targetrepo

echo "Create workspace"
workspace = (ace:cw)

echo "For each bundle"
each $deployed {
  echo "Get metadata from bundle"
  identity = $it getIdentity
  version = $it getVersion
  name = "$identity - $version"
  url = $it getUrl
  mimetype = $it getMimetype
  echo "Bundle exists?"
```



```
echo "For each bundle"
each $deployed {
  echo "Get metadata from bundle"
  identity = $it getIdentity
  version = $it getVersion
  name = "$identity - $version"
  url = $it getUrl
  mimetype = $it getMimetype
  echo "Bundle exists?"
  if { (coll:first
          ($workspace la "(&(Bundle-SymbolicName=$identity)
                            (Bundle-Version=$version))")) } {
    echo "$name already exists"
  } {
    echo "Create artifact for bundle"
    $workspace ca [
     artifactName="$name"
     url="$url"
     mimetype="$mimetype"
     Bundle-SymbolicName="$identity"
     Bundle-Version="$version"
    ]
  }
}

echo "Commit workspace"
$workspace commit
exit 0
```

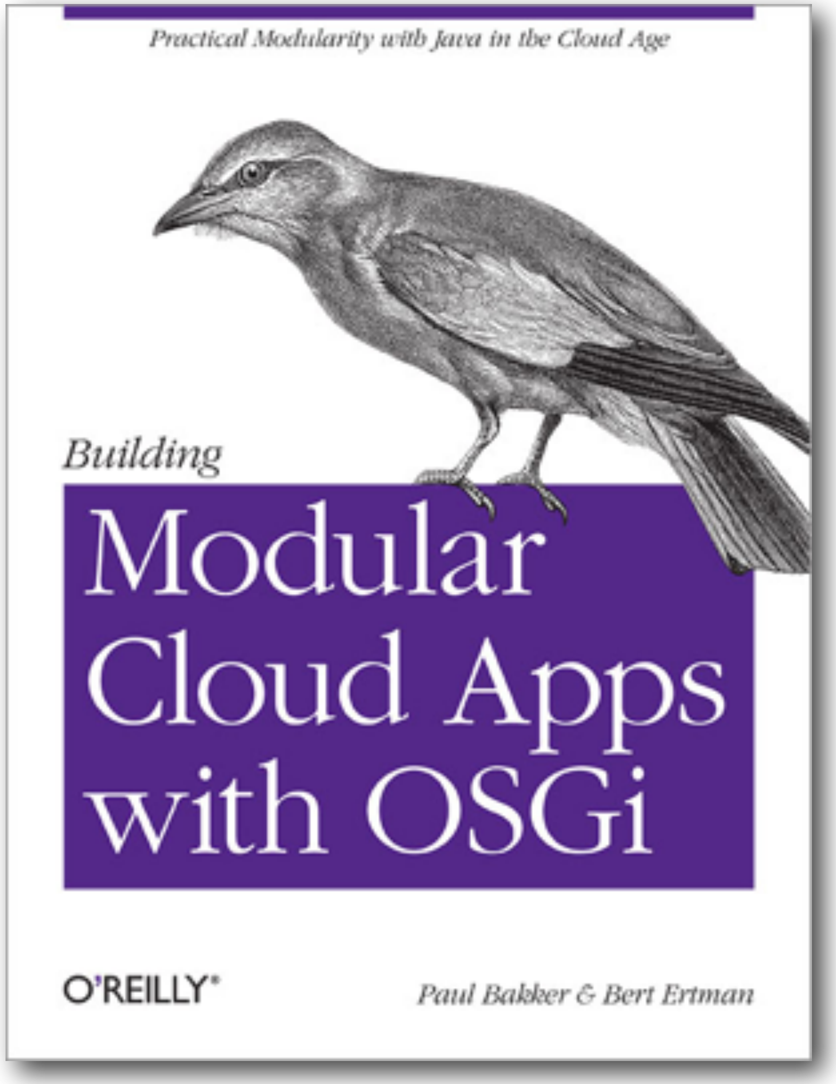


Example

- Amdatu showcase in local git repo
  - `post-commit` or `post-receive` hook
- Jenkins
  - post build step to deploy to ACE
- ACE
  - deploys artifacts to targets

Demo

- With the Amdatu Showcase (extended)



Practical Modularity with Java in the Cloud Age

Building

Modular Cloud Apps with OSGi

O'REILLY®

Paul Bakker & Bert Ertman

Provisioning Server
http://ace.apache.org/

Cloud OSGi services
http://www.amdatu.org/

Eclipse OSGi plugin
http://bndtools.org/

That's us
http://luminis.eu/

Demo code
bitbucket.org/amdatu/showcase/

Dank U

Merci

Danke

Mahalo

Grazie

Obrigado

Gracias

Thank you

Takk