

# Always Be Delivering

Jimmy Balmert

For full slides:

[https://github.com/jbalmert/always\\_be\\_delivering](https://github.com/jbalmert/always_be_delivering)

# Jimmy Balmer t

Director of Craftsmanship at  
Manifest Solutions in  
Columbus, OH

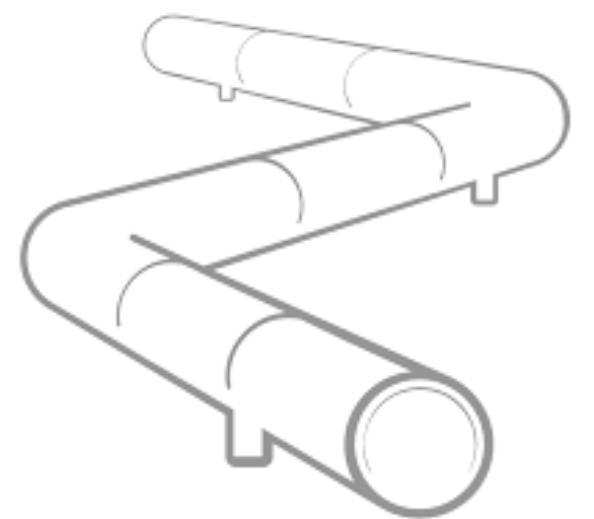
Software craftsman with over  
20 years experience in IT  
across the banking,  
insurance, and health care  
industries.



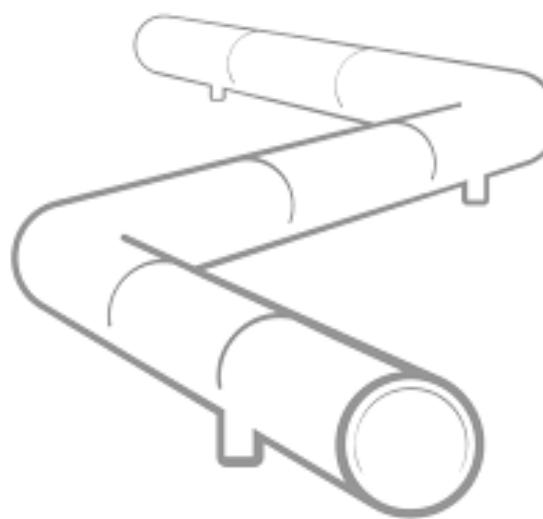
# Agenda

- Introduction
  - Continuous Delivery/Deployment
  - What is a pipeline?
  - Benefits
  - Who Is using CDD?
- Building a pipeline
  - Sam automates the build
  - Sam adds versioning to the build
  - Sam pushes artifacts to Nexus
  - Sam adds tests to the build
  - Sam adds health checks to the app
  - Sam adds acceptance tests to the build
  - An aside about QA
  - Sam adds feature toggles to the app
  - Sam adds parallel tests to the build
  - Sam uses feature toggles for a/b testing
  - Sam automates deployments to production
  - Sam adds blue-green deployments
  - Bonus!

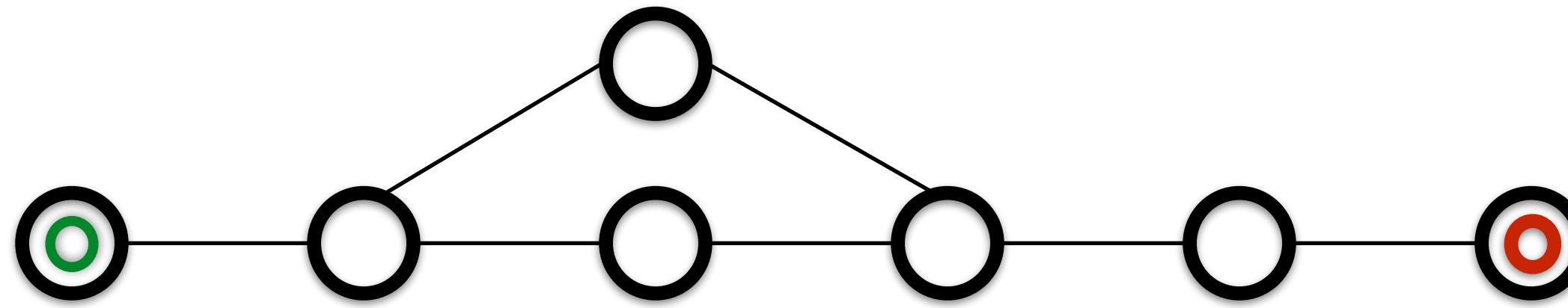
# Continuous Delivery



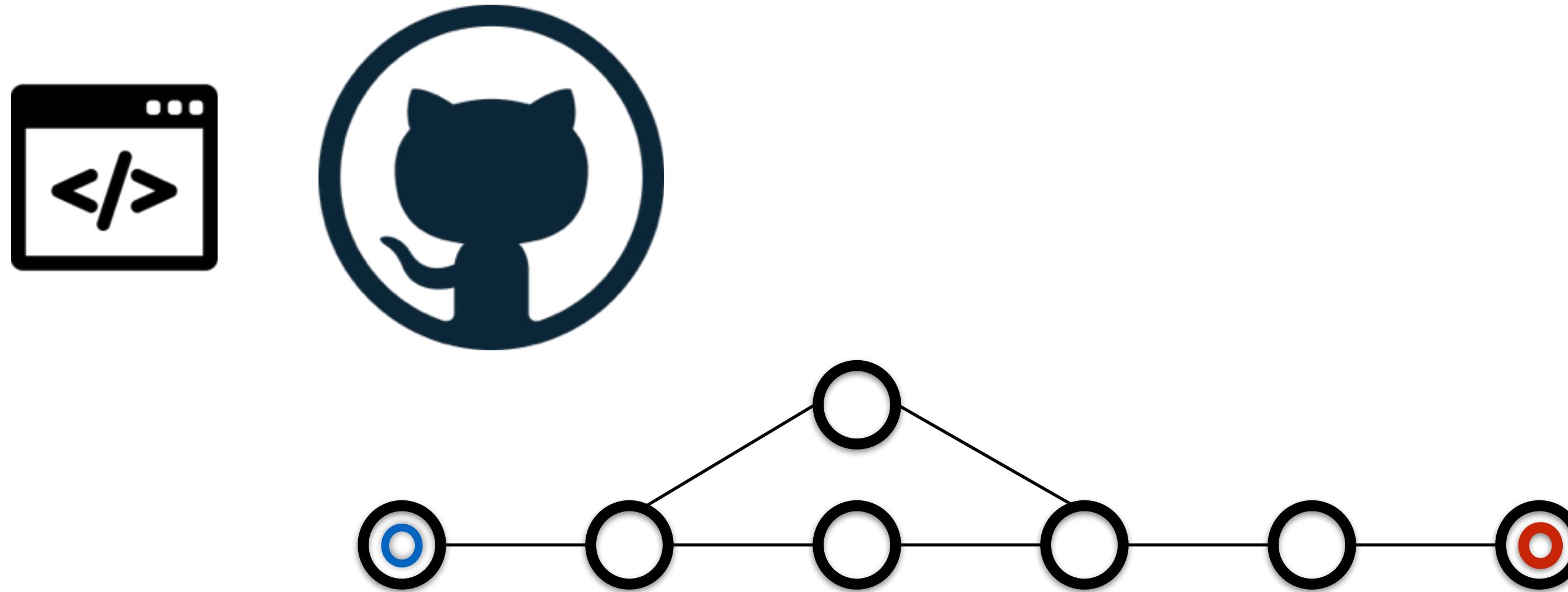
# Continuous Deployment



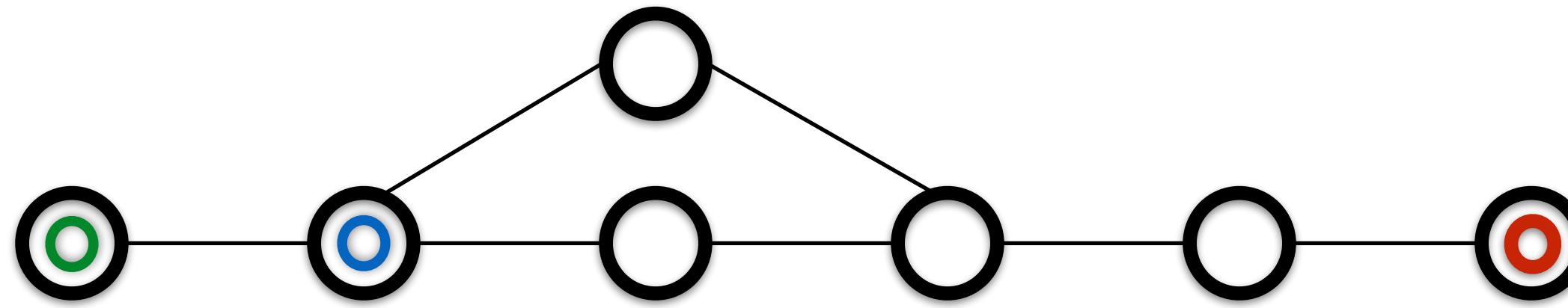
# What is a pipeline?



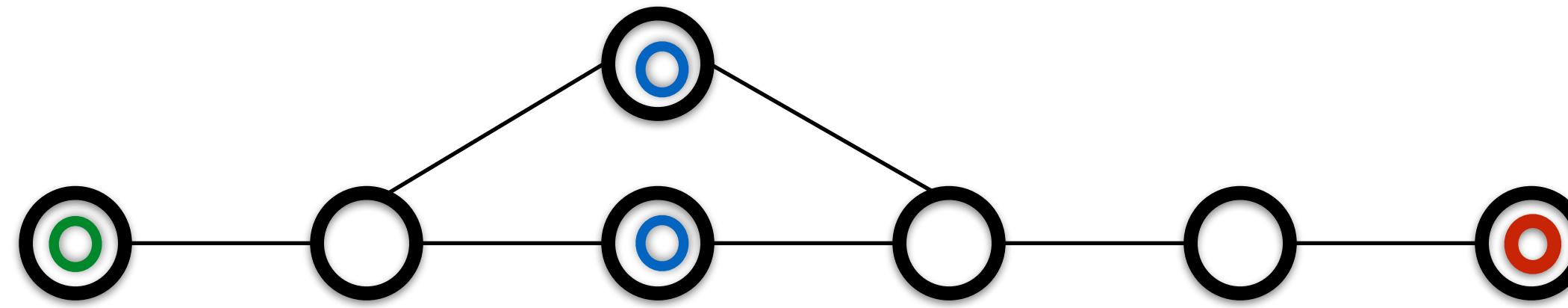
# What is a pipeline?



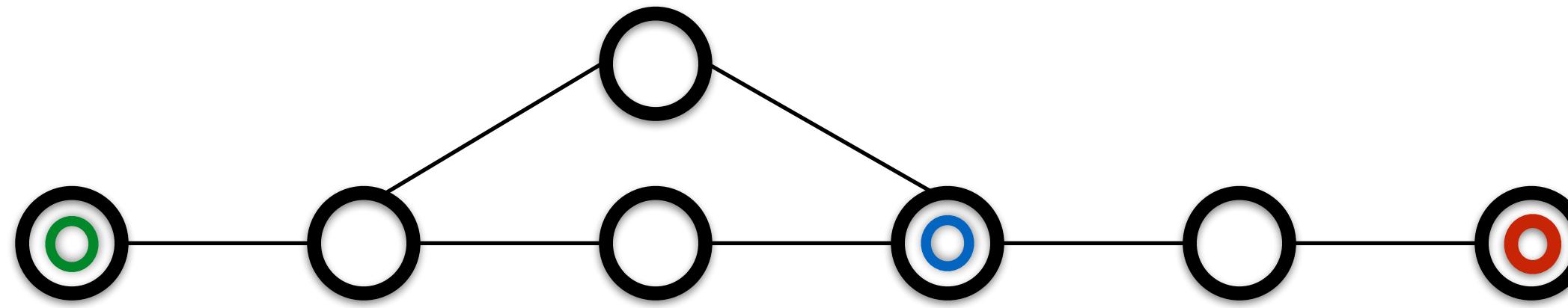
# What is a pipeline?



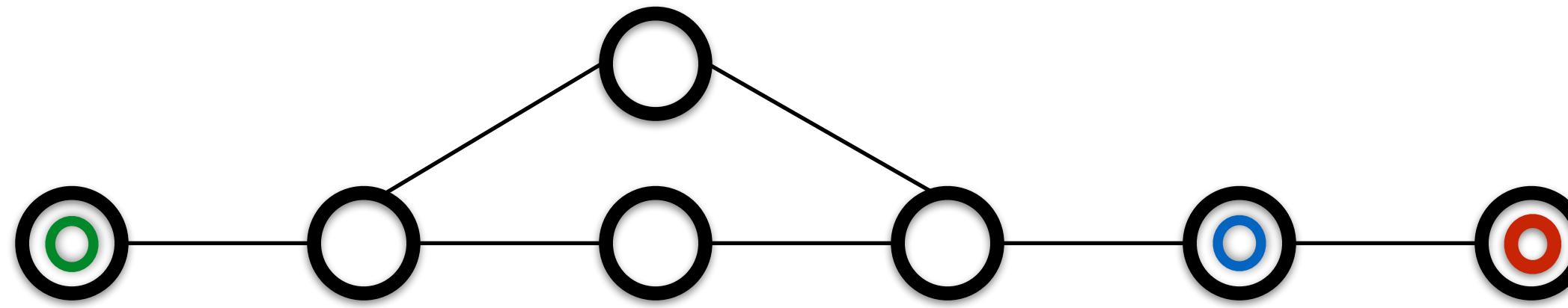
# What is a pipeline?



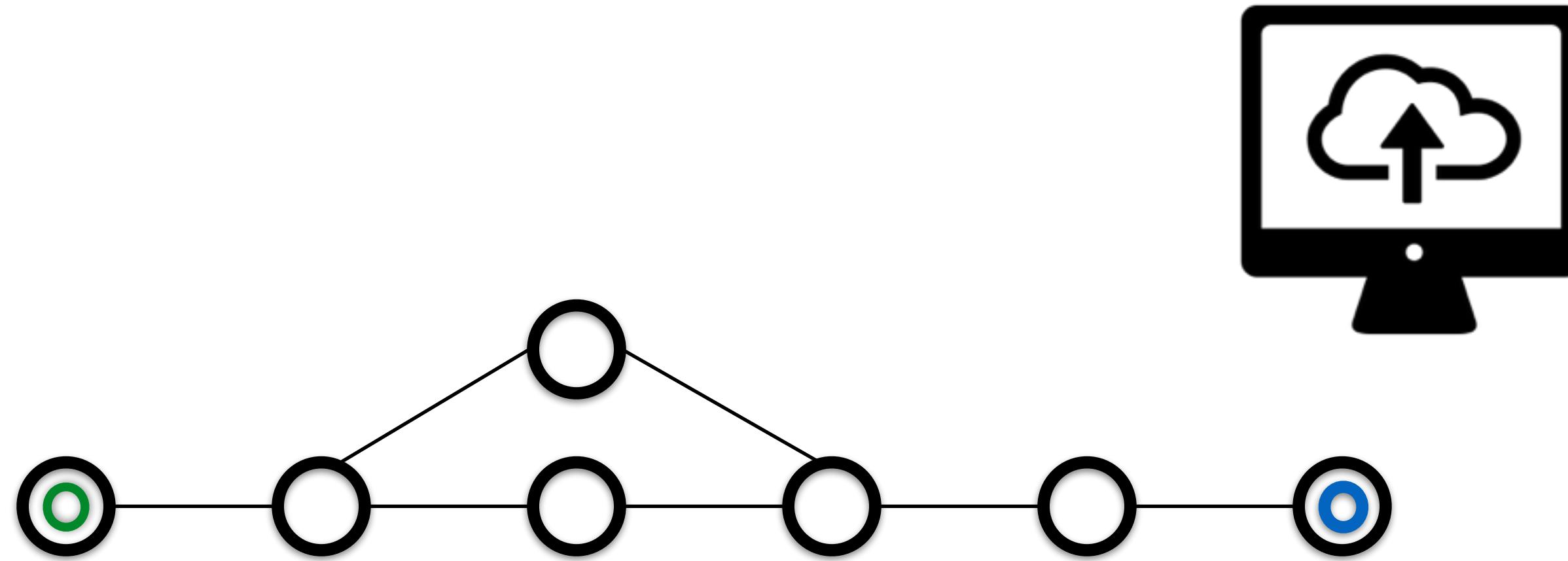
# What is a pipeline?



# What is a pipeline?

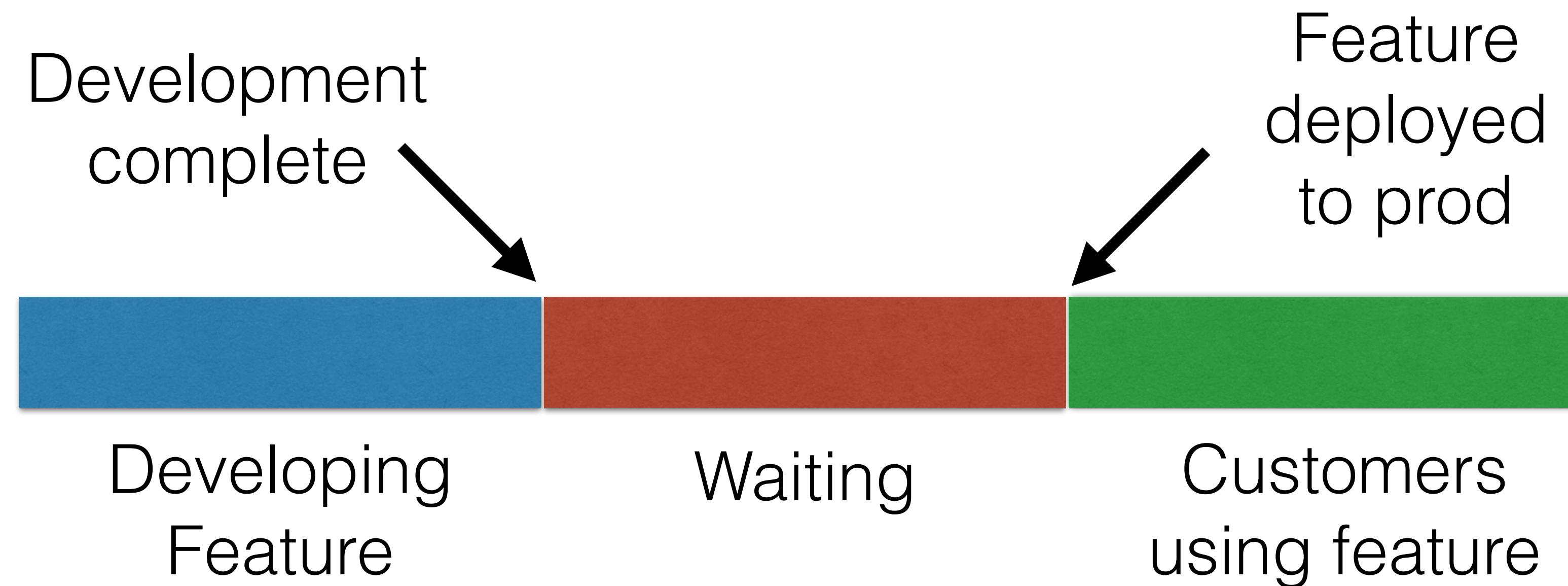


# What is a pipeline?

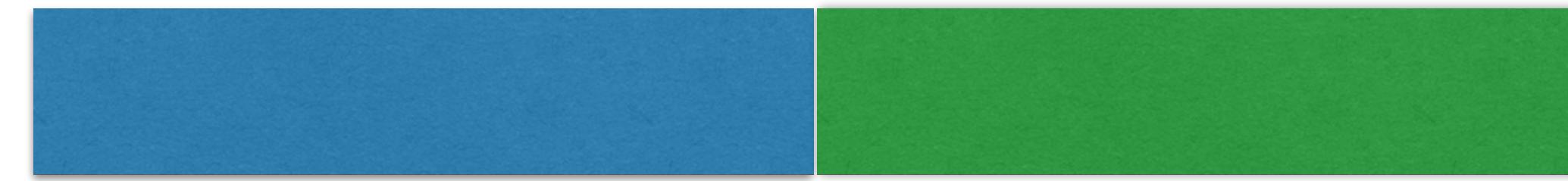


# Benefits

# Standard development timeline

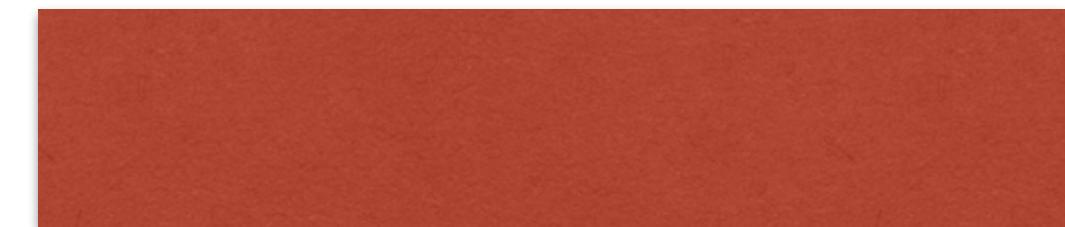


# Continuous Delivery timeline

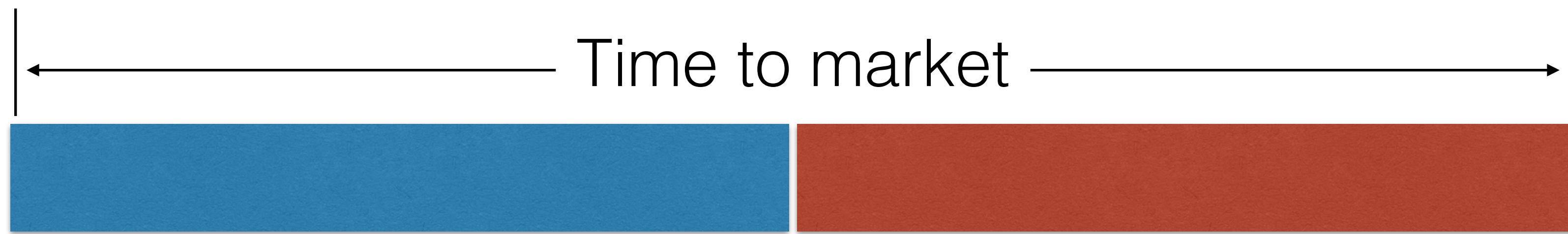


Developing  
Feature

Customers  
using feature



# Standard timeline



# Continuous Delivery timeline

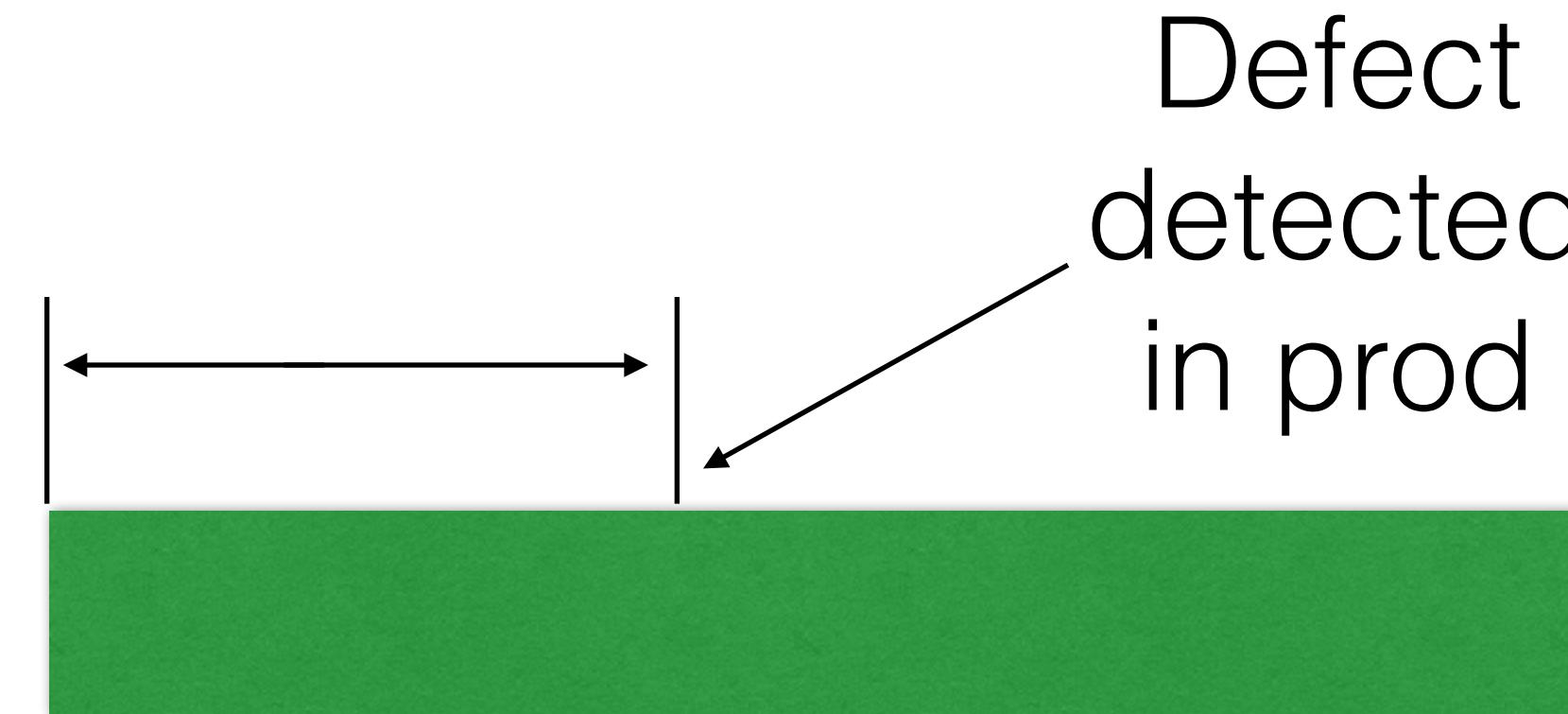


**Get to market faster**

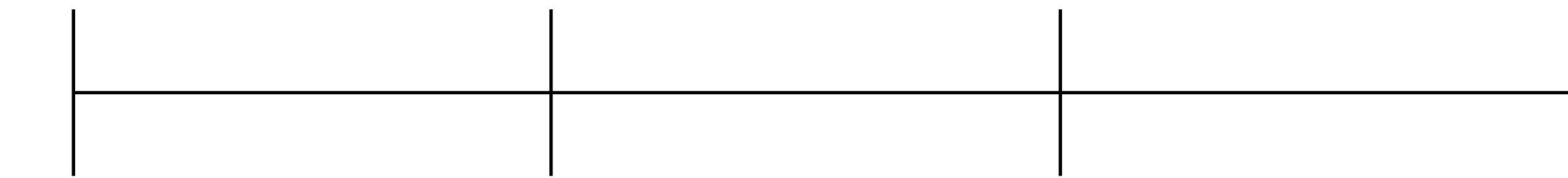
# Standard timeline



# Continuous Delivery timeline



← Confidence + Stress →

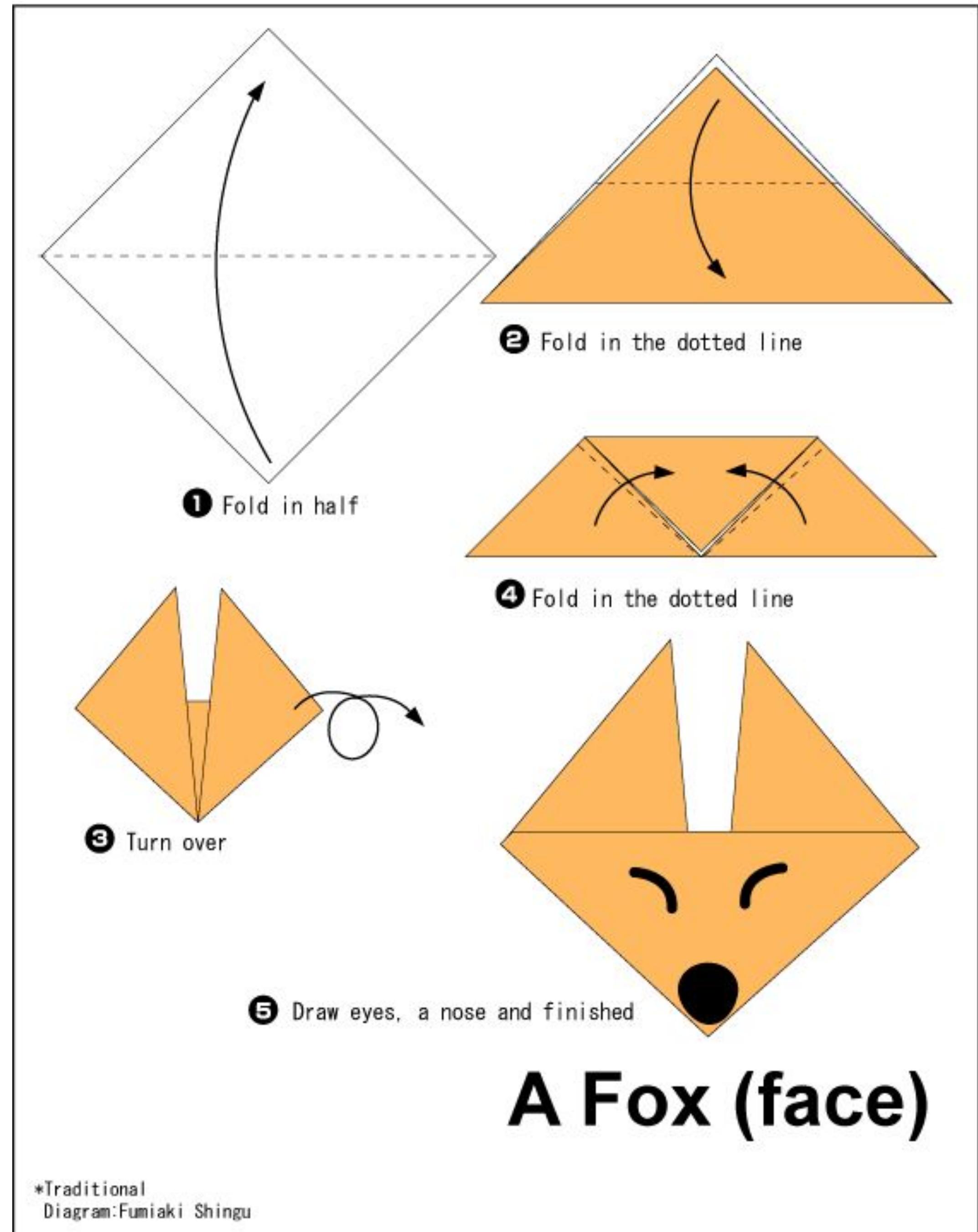


1 day      1 week      1 month      1 year

Time since you worked on a feature

Greater confidence

Lower stress

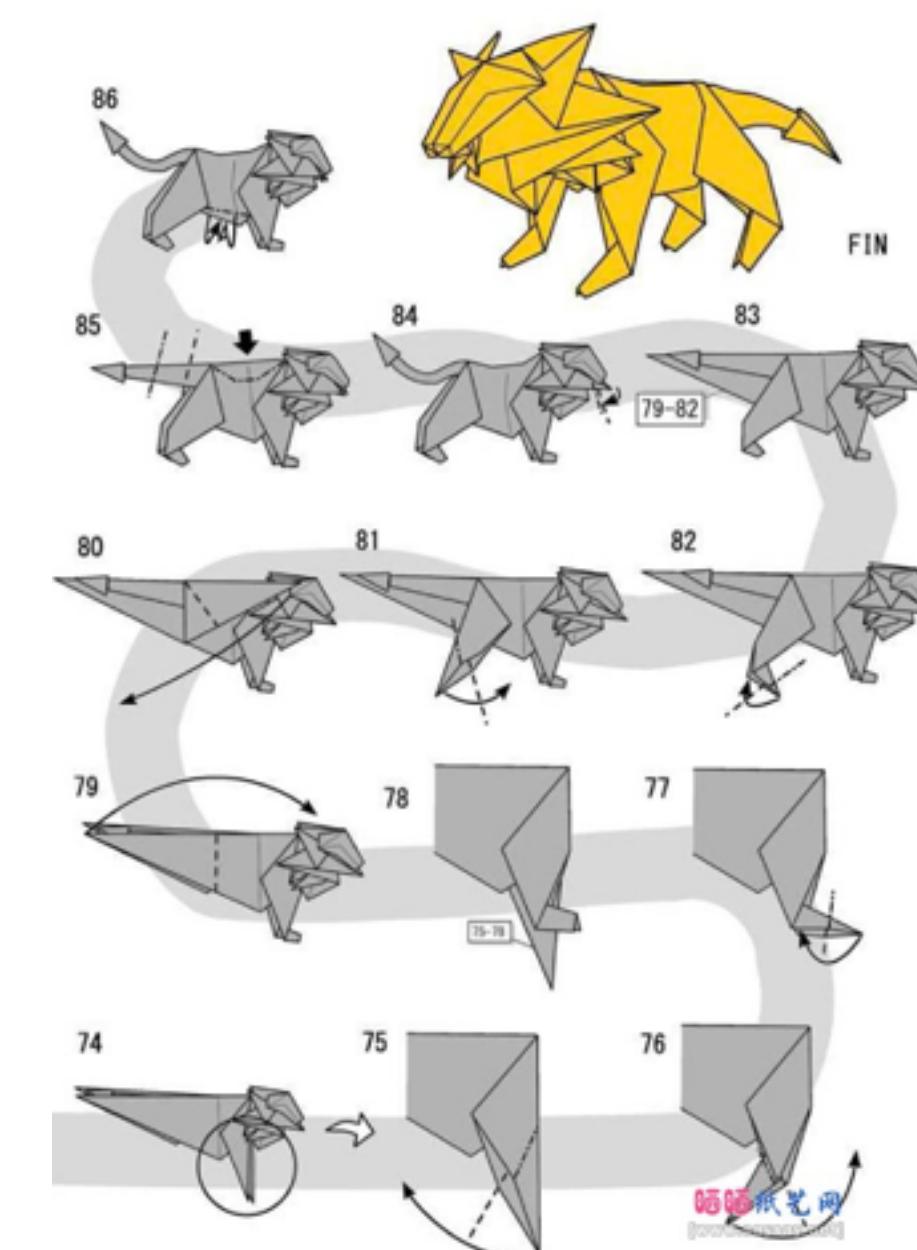
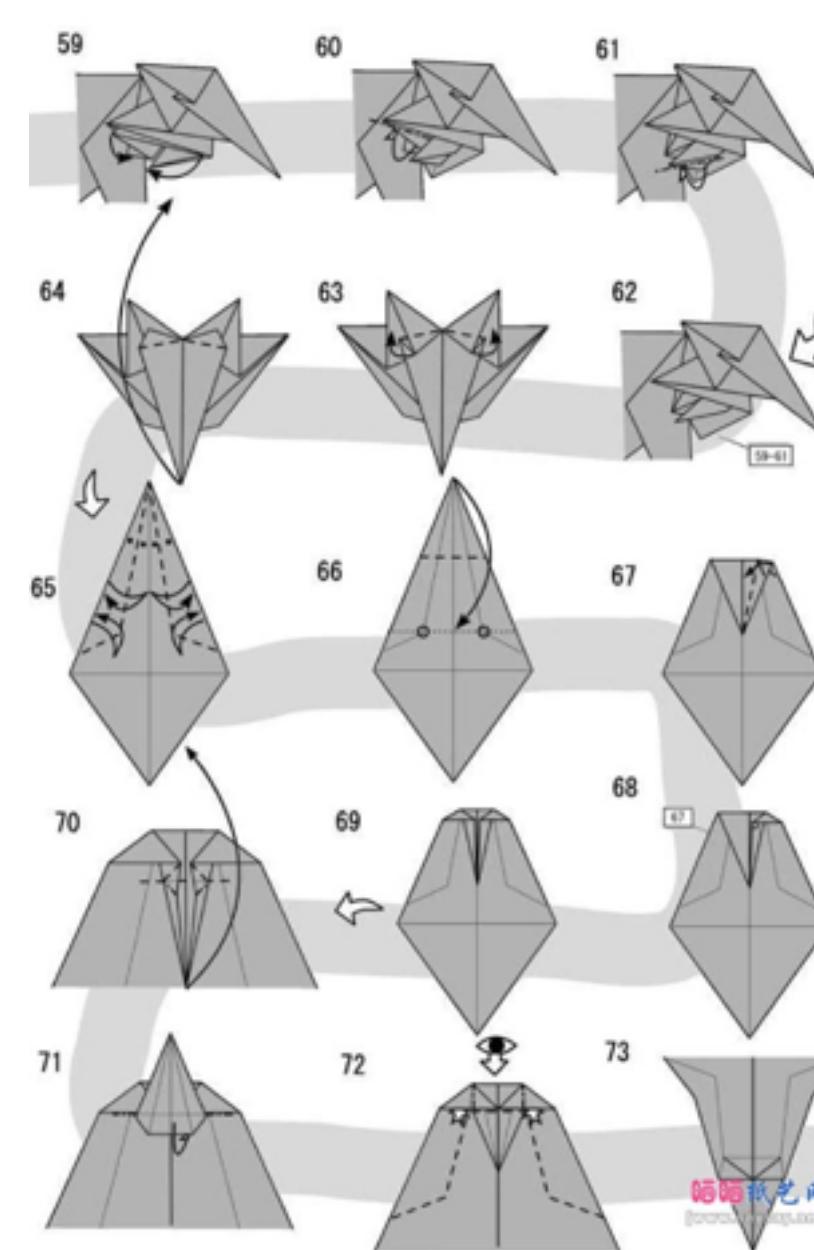
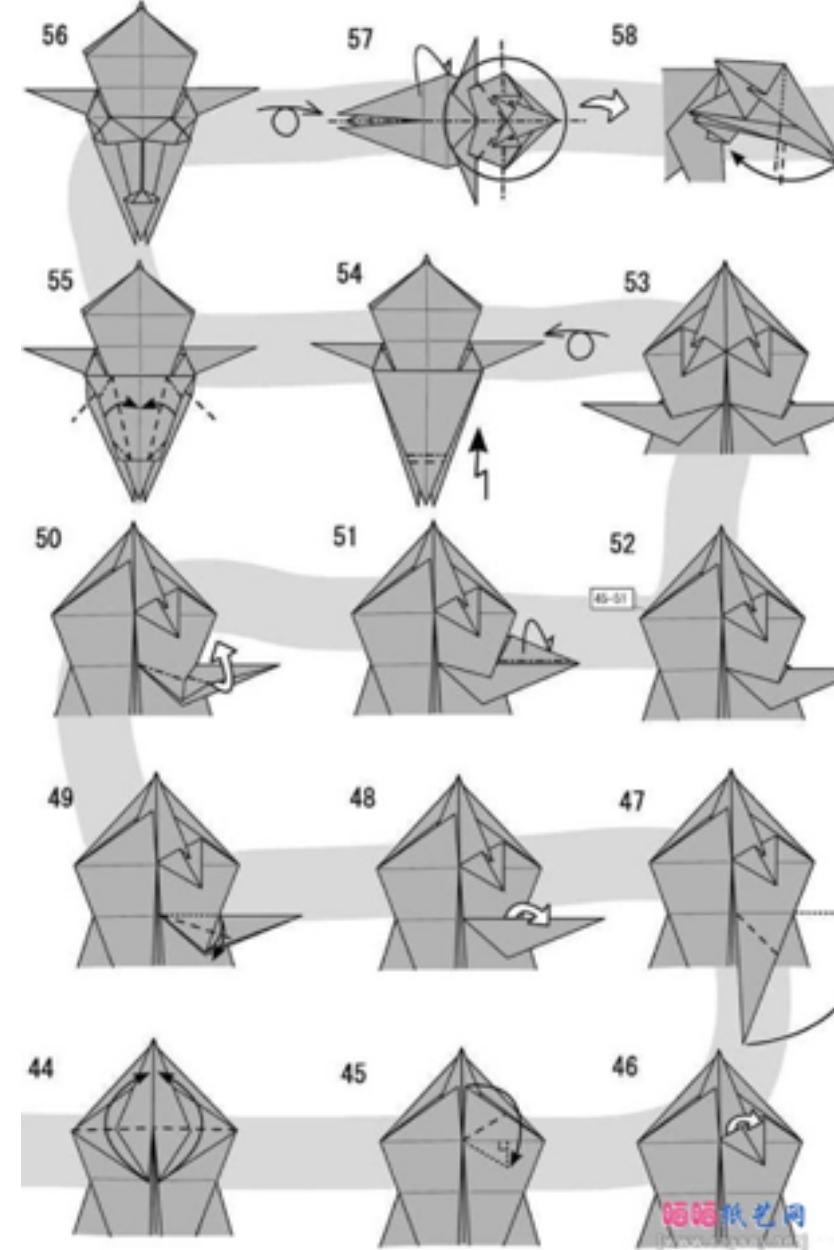
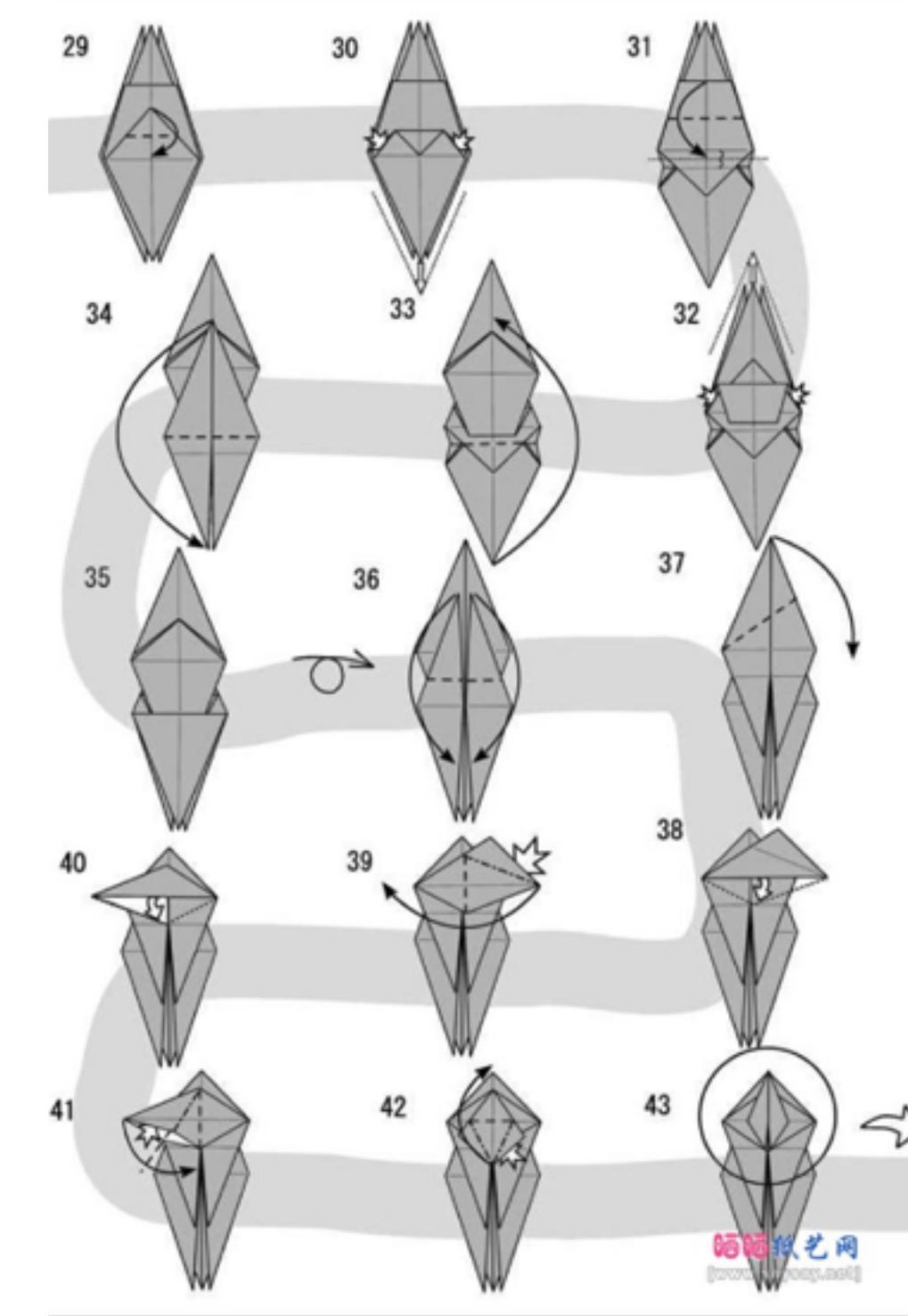
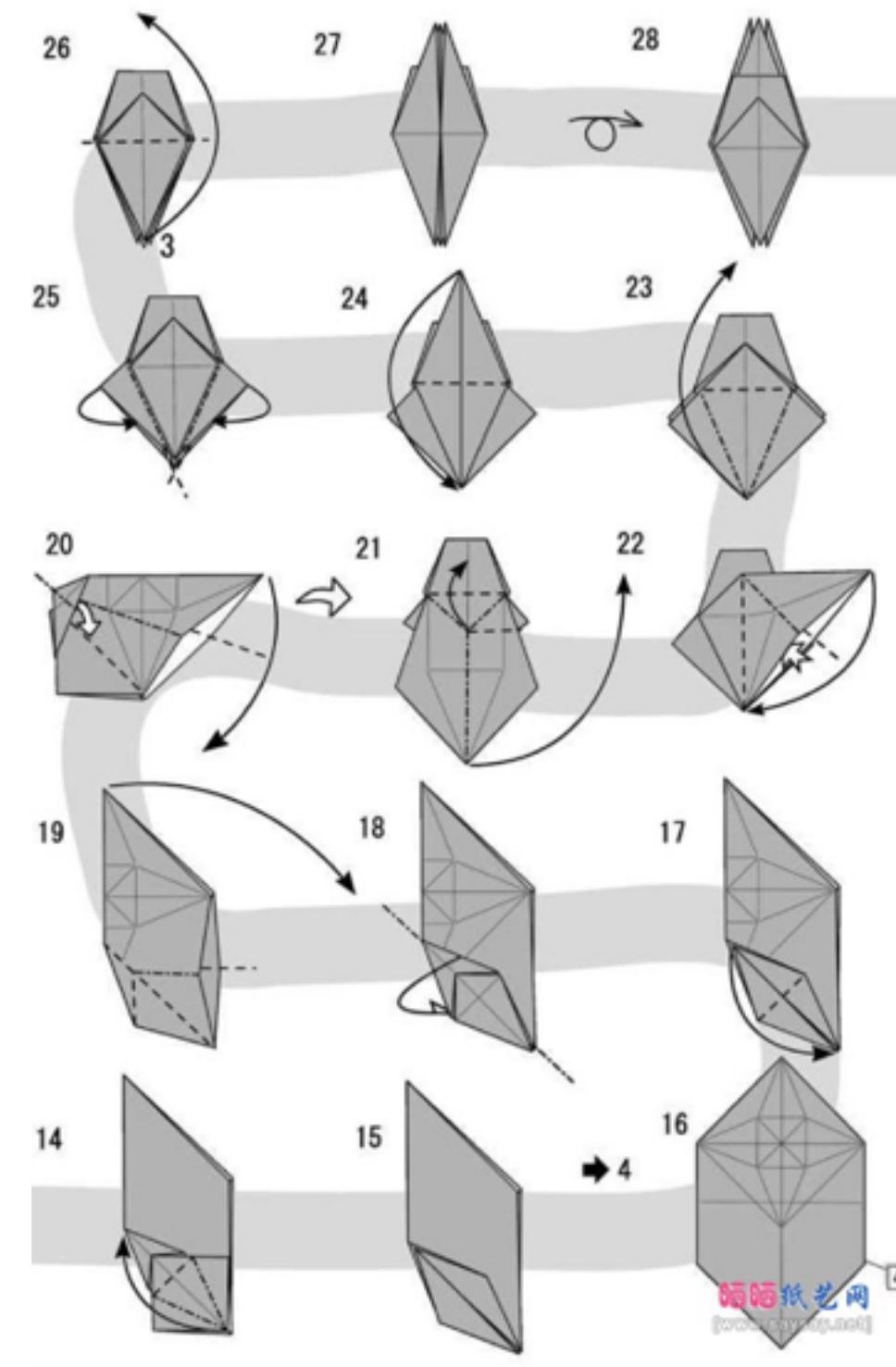
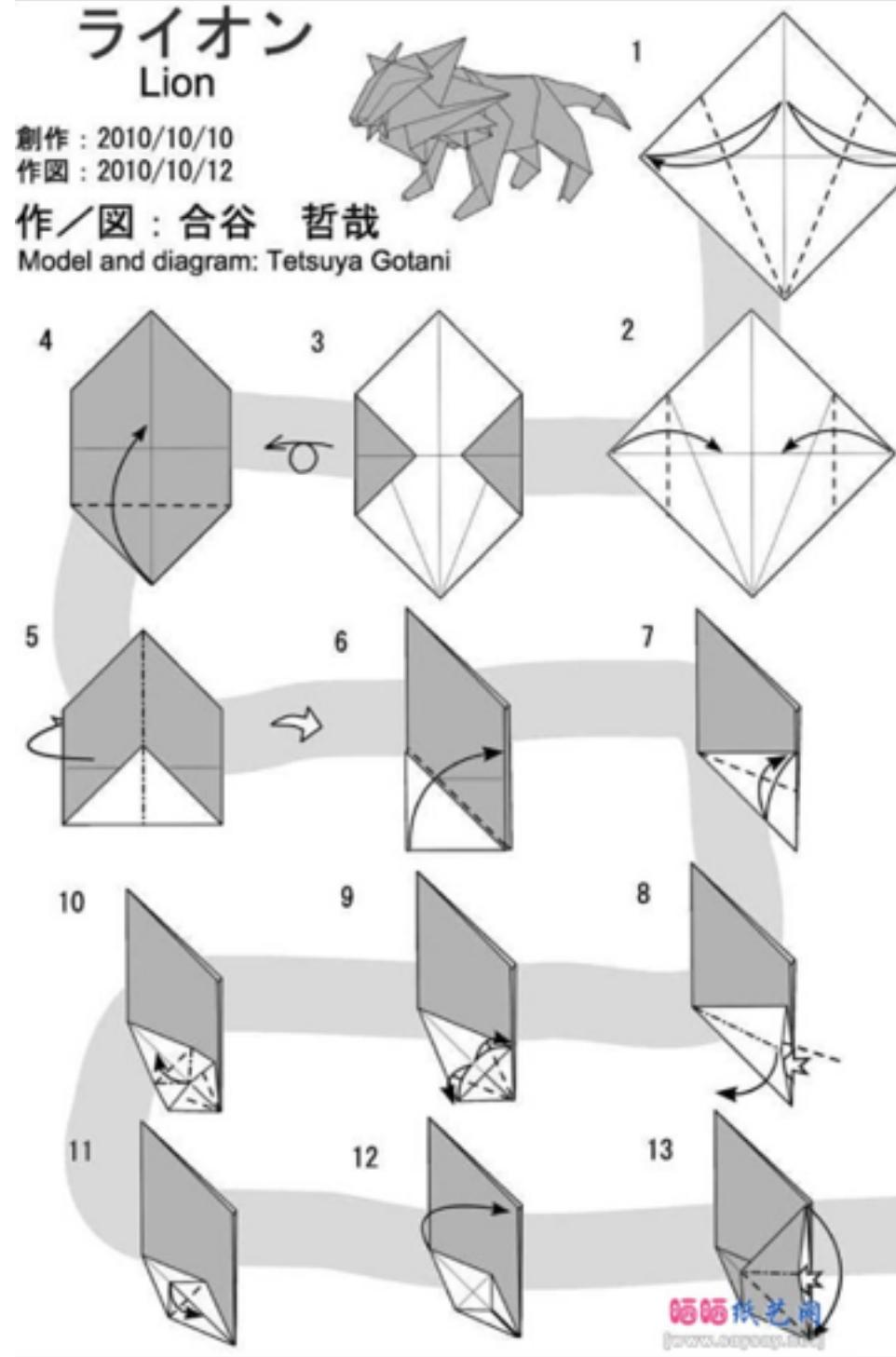


# ライオン

Lion

創作 : 2010/10/10  
作図 : 2010/10/12

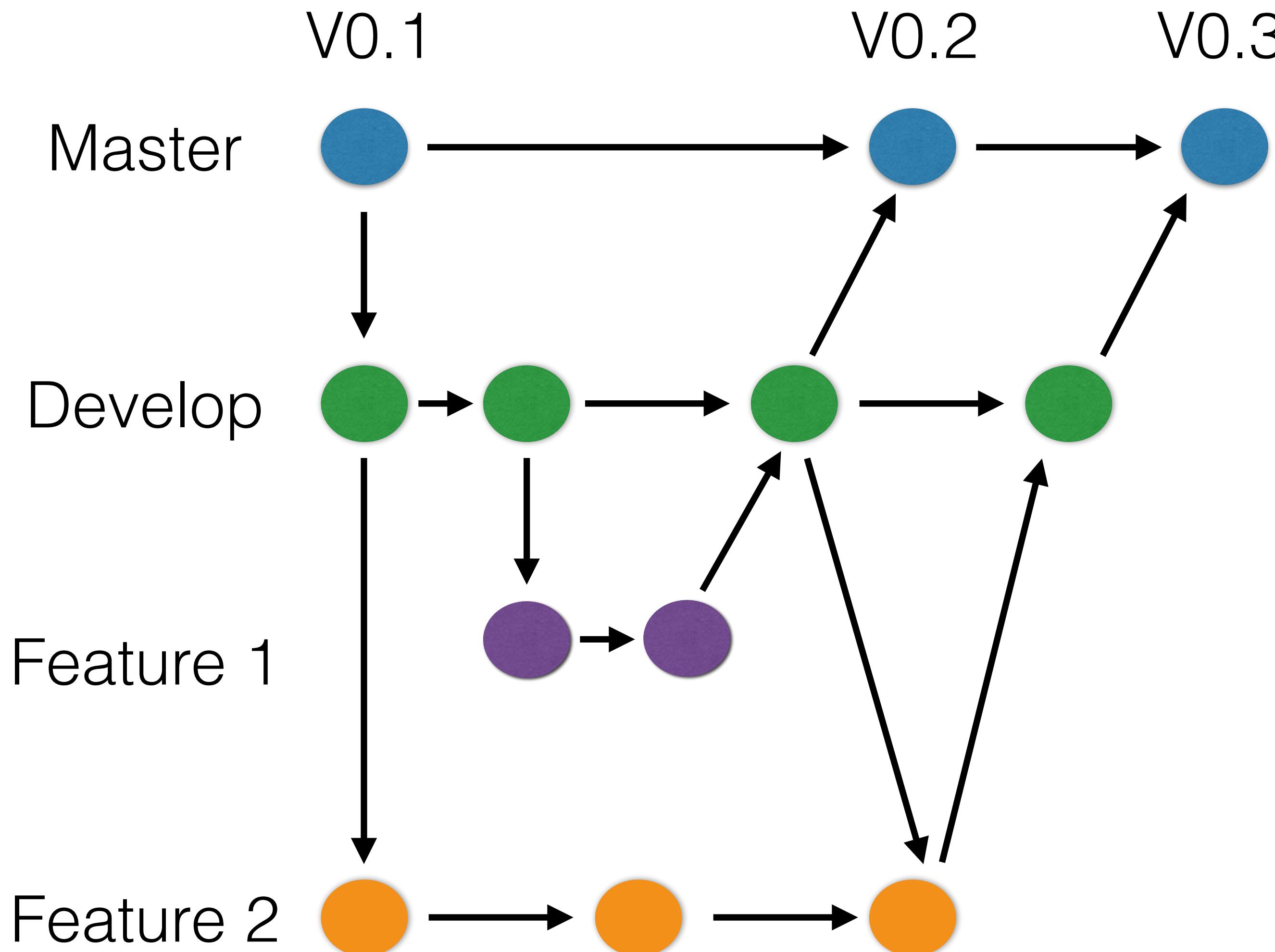
作／図：合谷 哲哉  
Model and diagram: Tetsuya Gotani





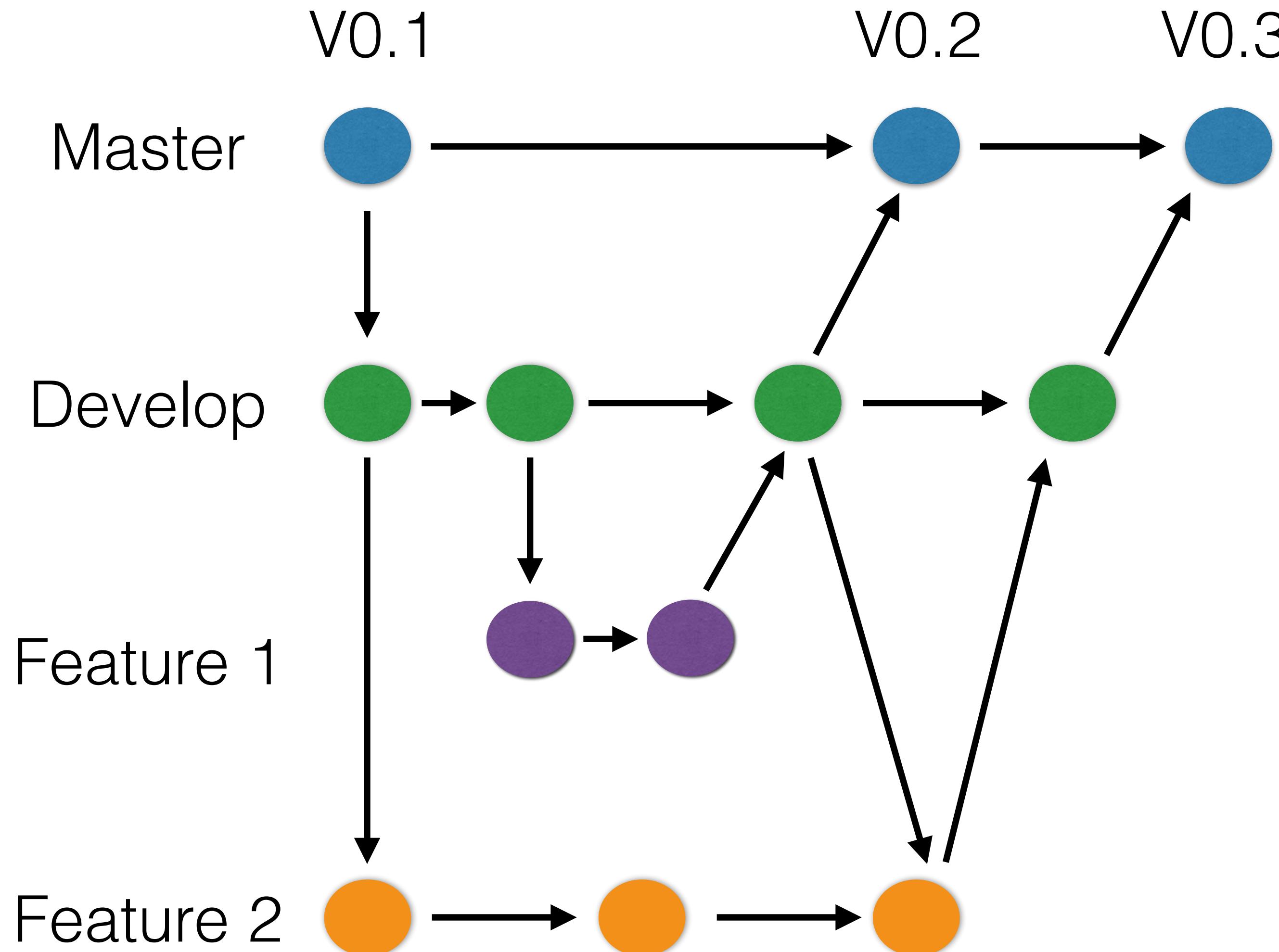
Smaller changes imply  
less risk

# Master = Prod

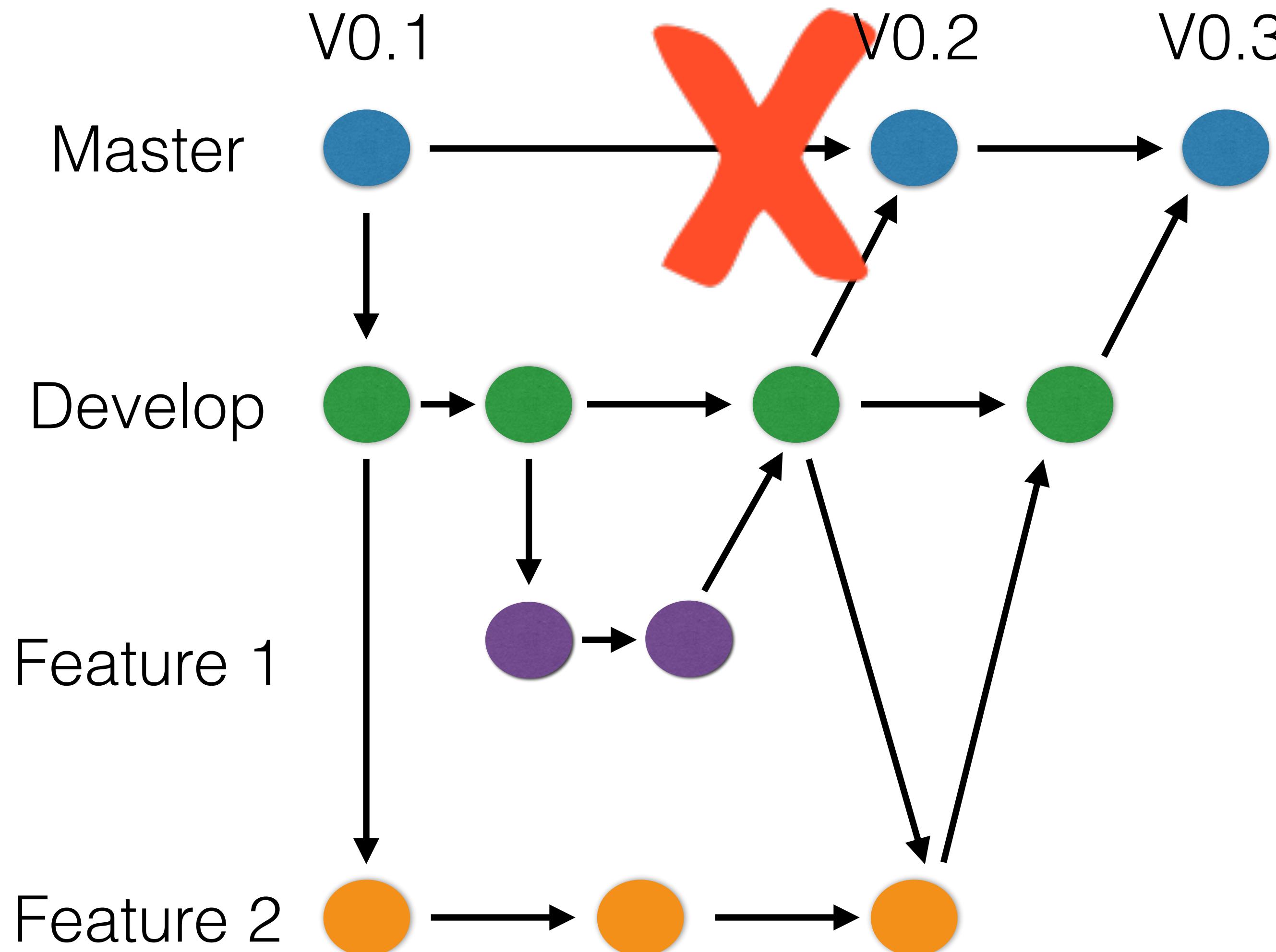


# Master = Latest

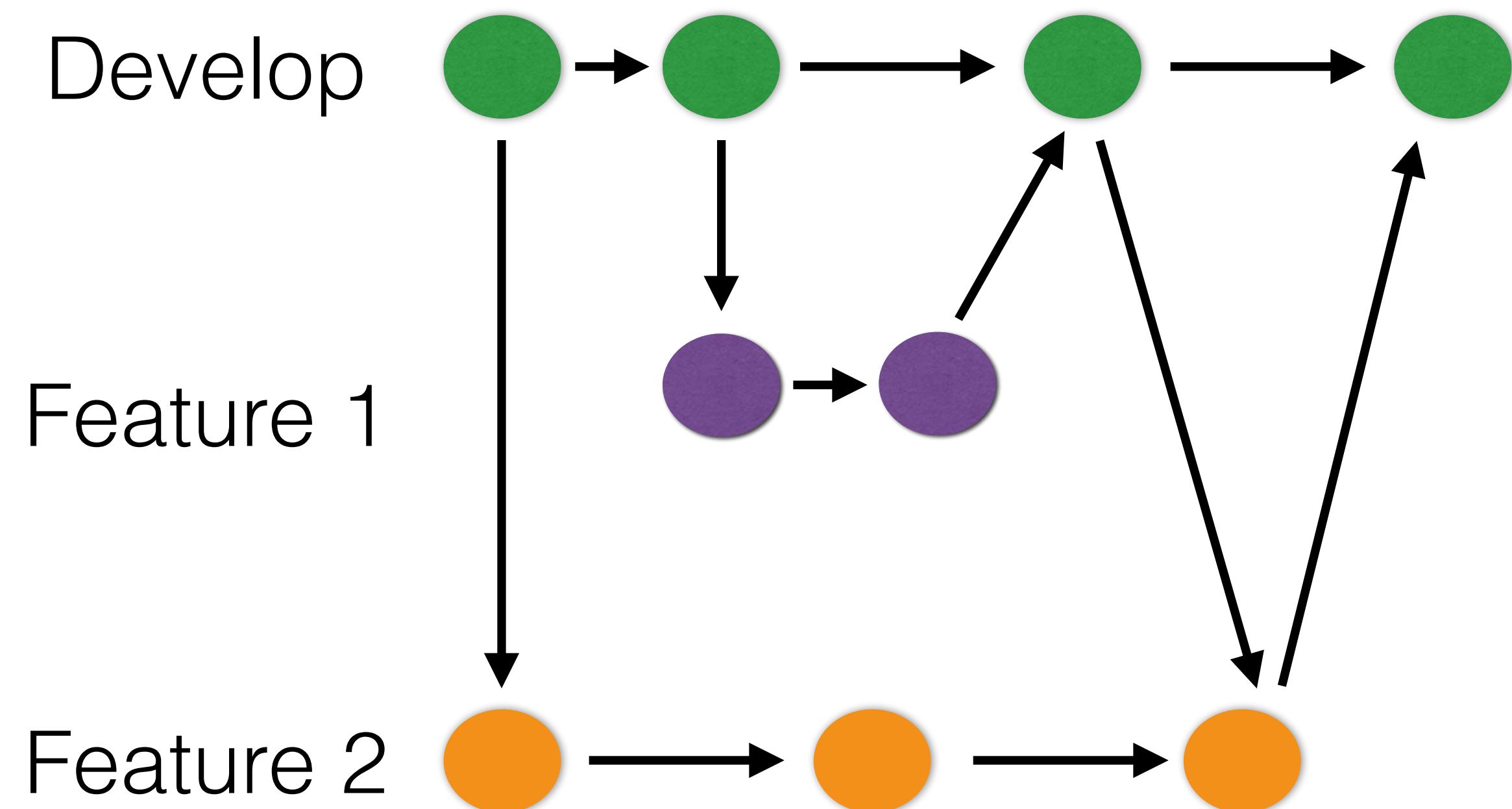
# Master = Latest



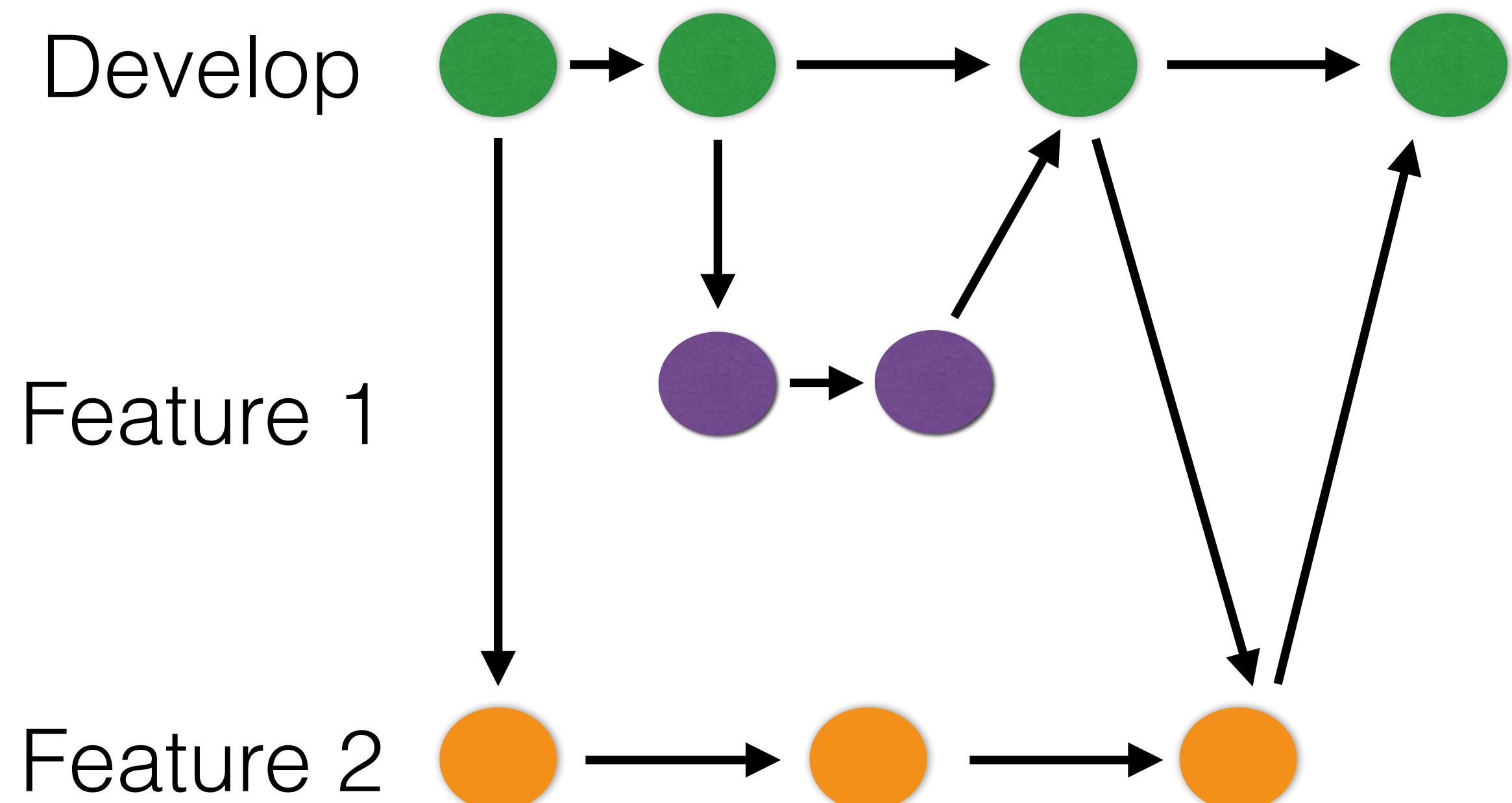
# Master = Latest



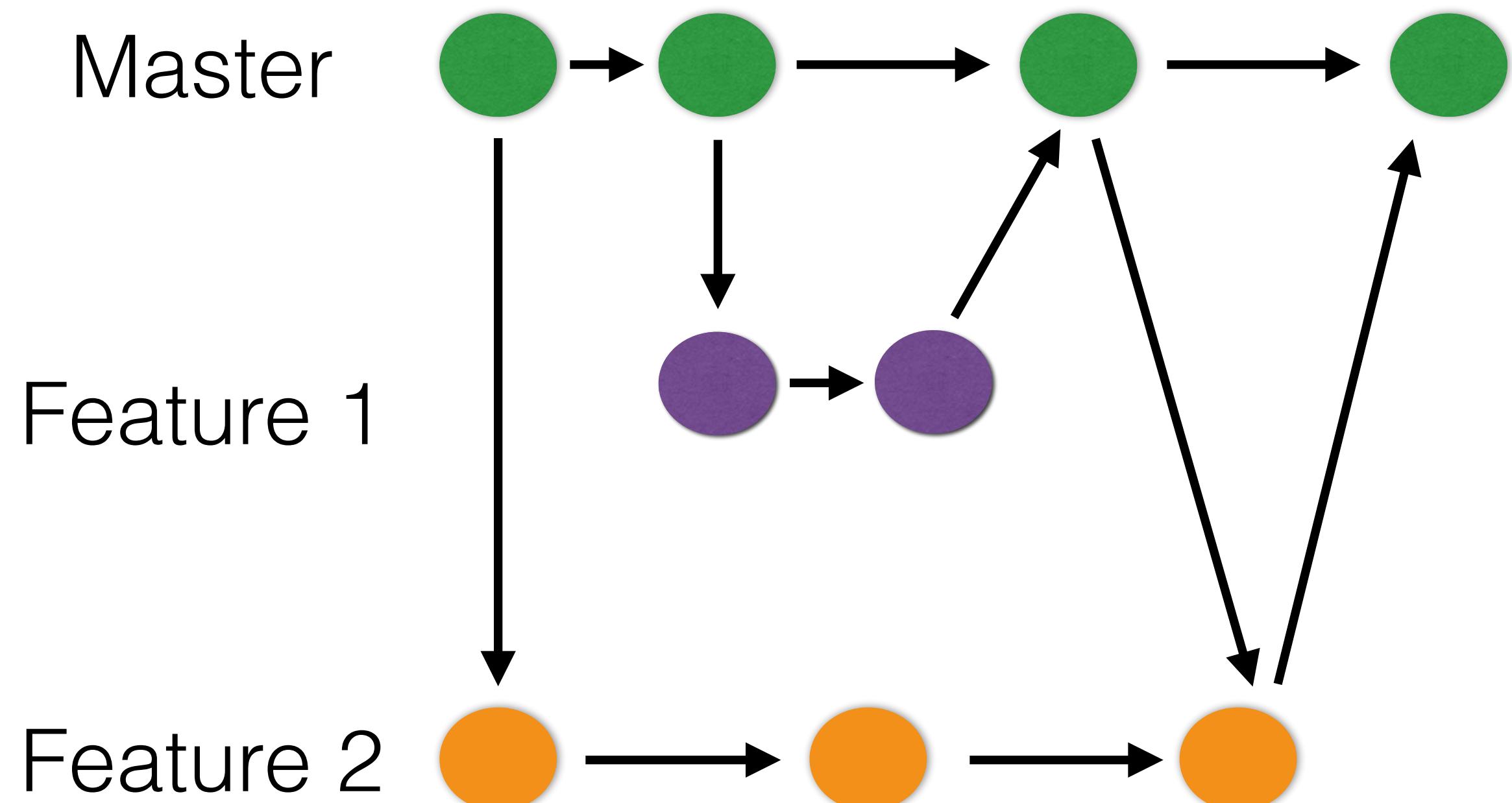
# Master = Latest



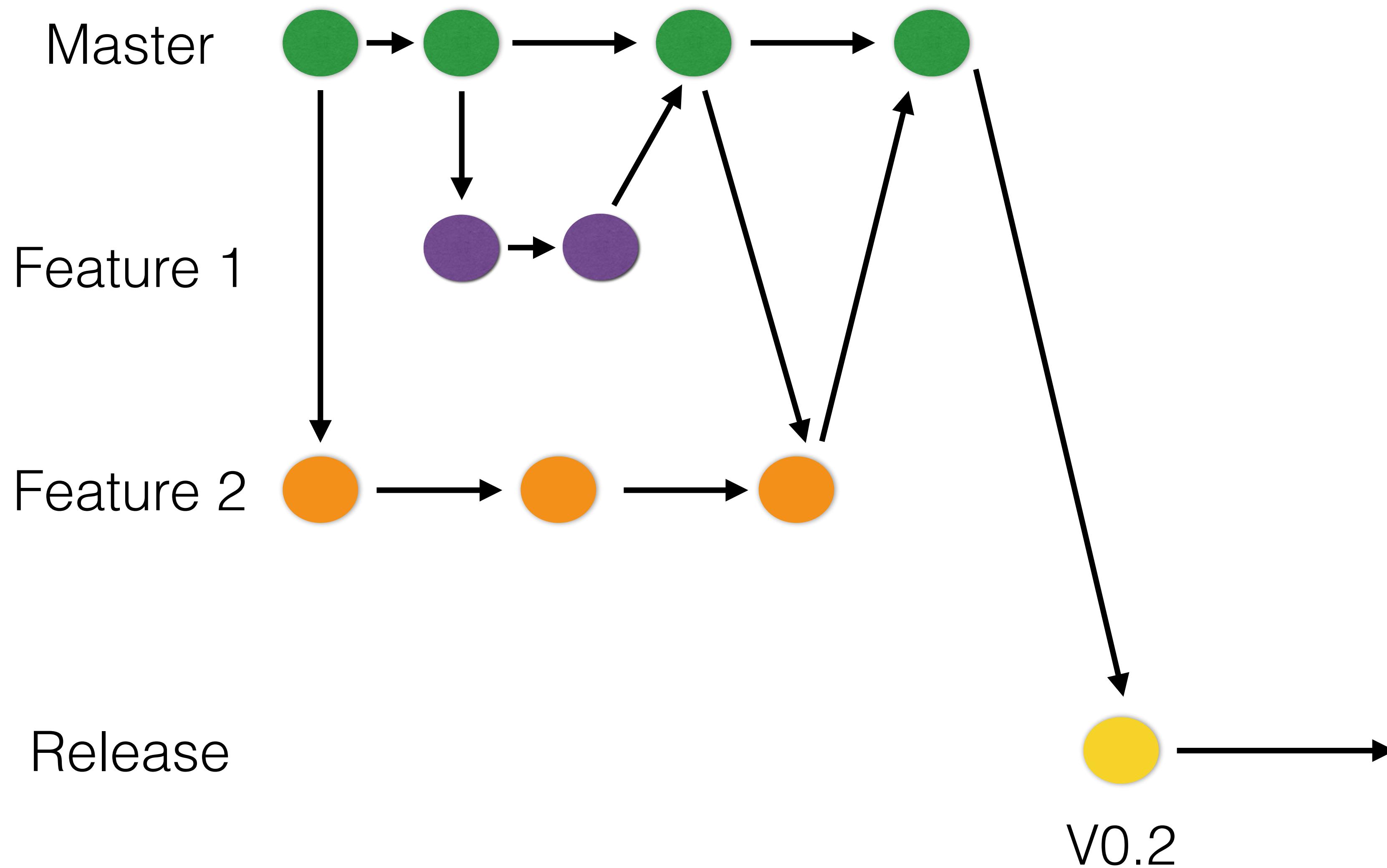
# Master = Latest



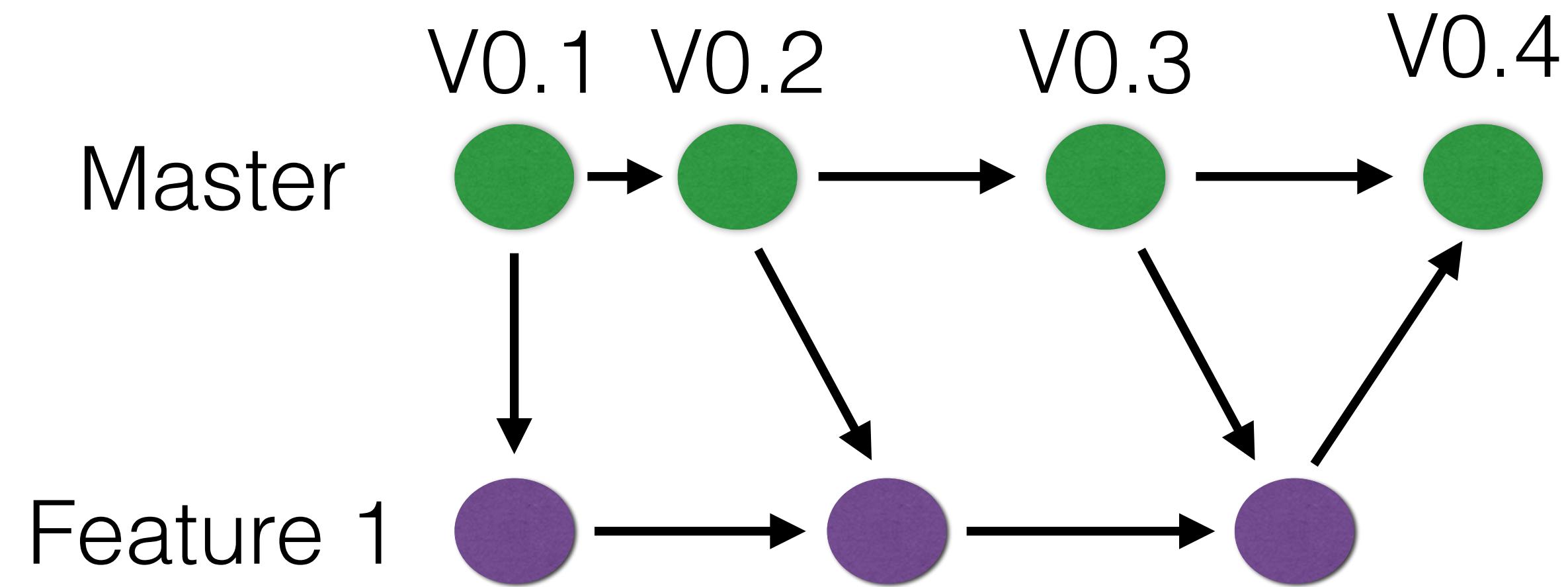
# Master = Latest



# Master = Latest



# Master = Continuous Delivery



Simpler branch  
strategy

But, why a  
pipeline?

	build	test: integration-&-quality	test: functional	test: load-&-security	approval	deploy: prod
Average stage times: (Average full run time: ~5s)	836ms	20min 43s	9ms	7ms	89ms	5ms
#17 <small>Sep 22 15:05</small> No Changes C Retry D Download	538ms	10s	10ms	8ms	72ms (paused for 7s)	4ms
#16 <small>Sep 22 15:04</small> No Changes C Retry D Download	479ms	6s	9ms	9ms	74ms (paused for 6s)	5ms
#15 <small>Sep 22 15:03</small> No Changes C Retry D Download	922ms	6s	10ms	9ms	failed	
#14 <small>Sep 22 15:03</small> No Changes C Retry D Download	1s	8s	12ms	9ms	80ms (paused for 5s)	5ms
#13 <small>Sep 22 15:02</small> No Changes D Download	942ms	9s	13ms	11ms	111ms (paused for 5s)	
#12 <small>Sep 22 15:02</small> No Changes C Retry D Download	1s	6s	13ms	11ms	aborted	

Jenkins / Blue Ocean #423

Branch master  
Commit #601366d

Changes by Michael Neale, Ben Waldo and Ivan Meredith

3 minutes and 42 seconds  
14 minutes ago

Pipeline Changes Tests Artifacts Re-run

Build Test Browser Tests Dev Staging Production

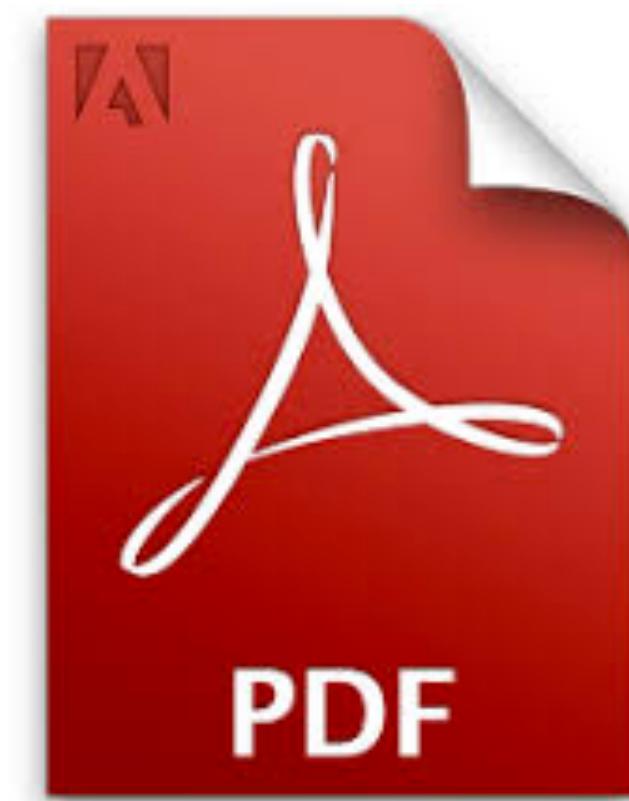
JUnit Firefox  
DBUnit Edge  
Jasmine Safari  
Chrome

Steps – Build

✓ > Start Docker container	3 minutes and 42 seconds
✓ > Warm maven caches	5 seconds
✓ > Install Java Tools	8 seconds
✓ > Maven	6 minutes and 12 seconds

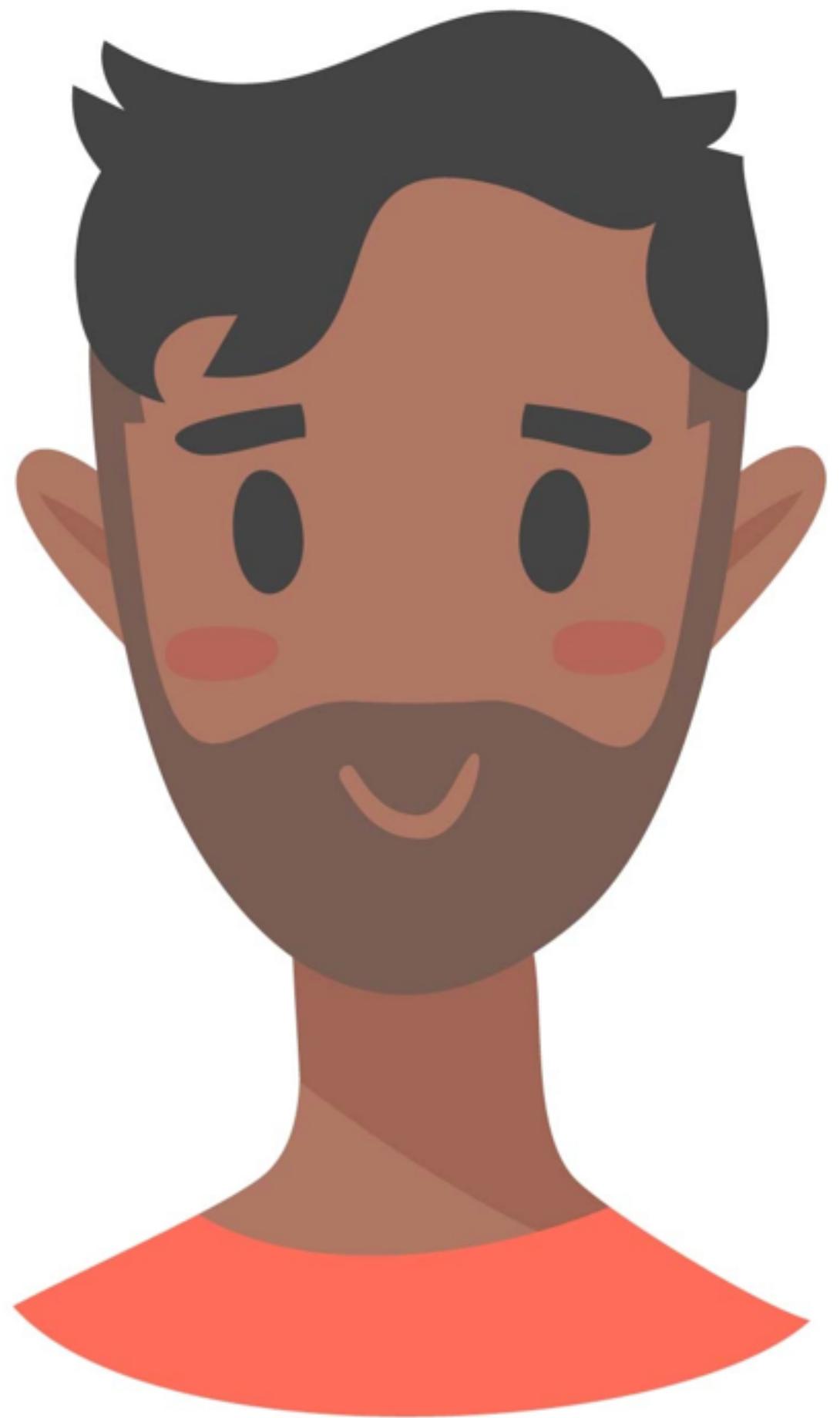
# Visibility

Google



NETFLIX

# Meet The Team



**Sam**

Tech lead



? ? ? ? ?  
? ? ? ? ?  
? ? ? ? ?



Diane  
Developer



Quinn  
QA Lead



Todd  
Tech ops

Sam automates  
the build



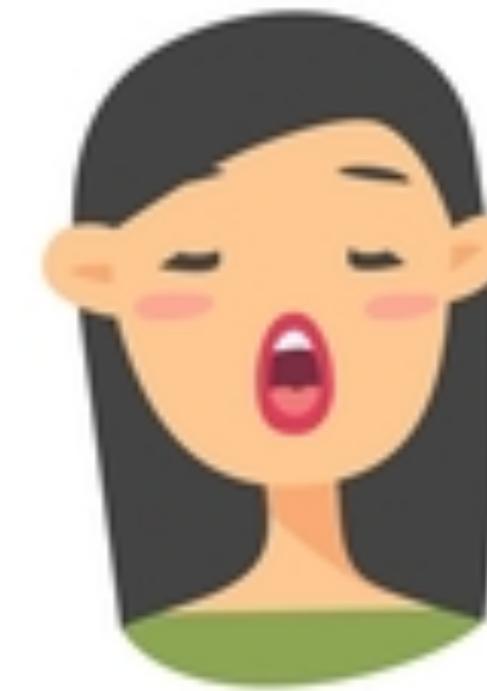
# What is our build process?





What is our  
build process?

We use





What is our  
build process?

We start the build  
after we commit code.





What is our  
build process?

The build creates a  
standalone jar.





What is our  
build process?

We start a deploy  
job to push to QA.





What is our  
build process?

We test and approve  
or reject the build.





Hmm...



Could we start the  
build after each  
commit?



New Item

Enter an item name

build

» Required field



### Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



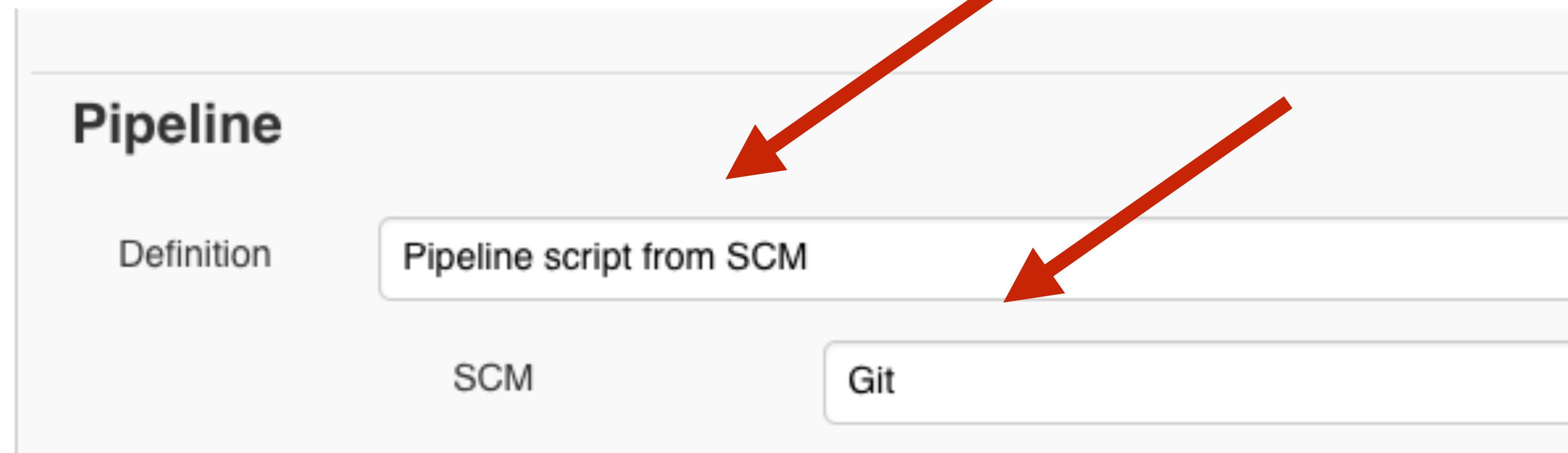
### Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.



### Pipeline

Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



```
2017-12-08 07:55:10:~/Workspaces/personal/cdd-workshop>ls E BE DRAGONS  
Jenkinsfile pom.xml 1 readme.md Automscally deploy target  
2017-12-08 07:55:11:~/Workspaces/personal/cdd-workshop>ss to control
```

pipeline {



}

```
pipeline {  
    agent any
```



```
}
```

```
pipeline {  
    agent any
```



```
        tools {
```

```
    }
```

```
}
```

```
pipeline {  
    agent any
```



```
    tools {  
        maven 'M3'  
    }
```

```
}
```

```
pipeline {  
    agent any
```



```
        tools {  
            maven 'M3'  
        }
```

```
    stages {  
    }  
}
```

```
stage('Build') {  
}  
}
```

```
stage('Build') {  
    steps {  
        }  
    }  
}
```

```
stage('Build') {  
    steps {  
        sh '  
    }  
}  
}
```

```
stage('Build') {  
    steps {  
        sh 'mvn  
            '  
    }  
}
```

```
stage('Build') {  
    steps {  
        sh 'mvn -B  
            '  
    }  
}
```

```
stage('Build') {  
    steps {  
        sh 'mvn -B clean  
            package  
    }  
}
```

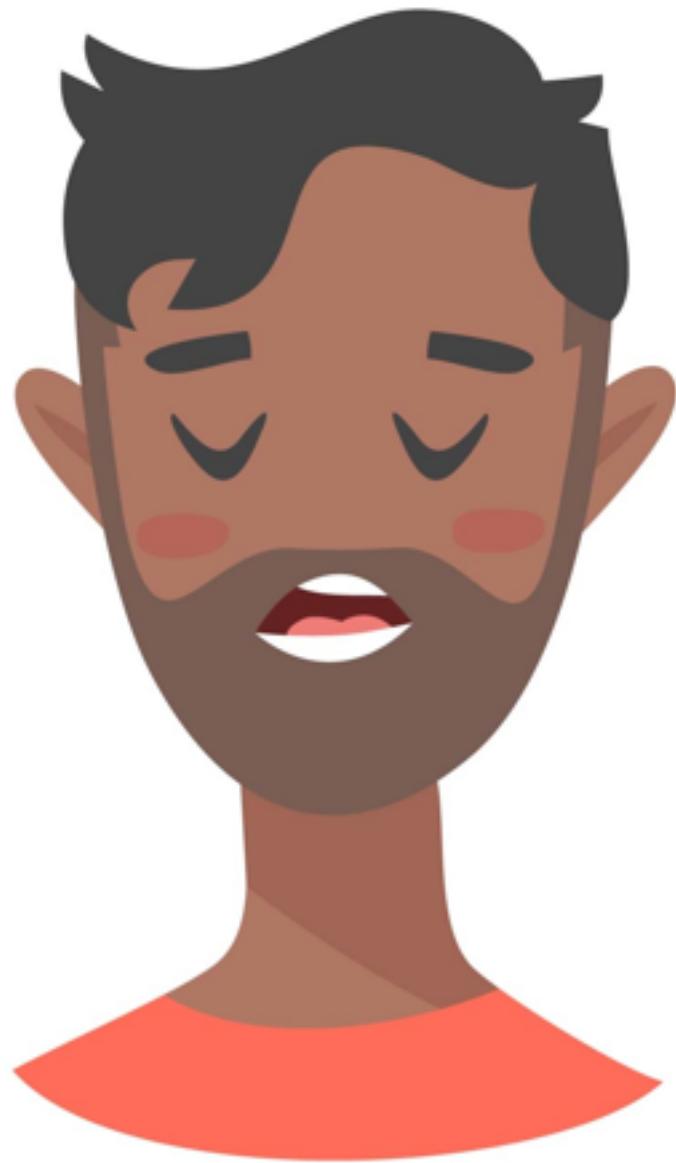
```
stage('Build') {  
    steps {  
        sh 'mvn -B clean  
            package  
            -DskipTests'  
    }  
}
```

Sam adds versioning to the  
the build



Hello?





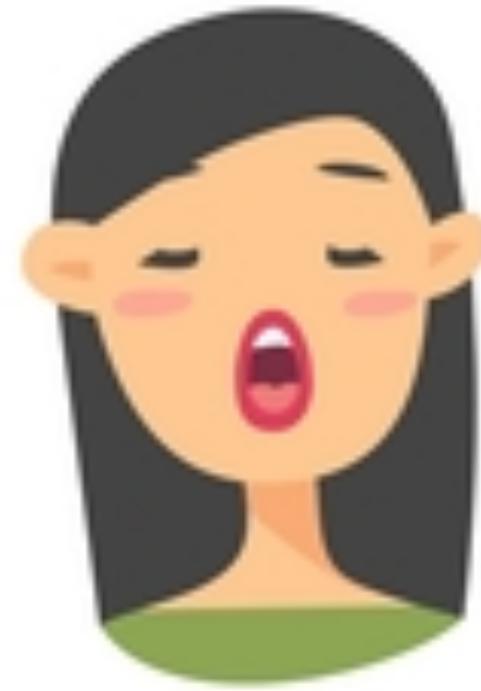
Hello?

The application is not  
responding!





I think I know what's  
happening.





The wrong version of  
the app was deployed.





I'll deploy the correct  
one now.





There! It's working  
now.





Confirmed. The app  
is behaving normally  
now.





Hmm...



How can we know  
we are deploying  
the right jar?





How can we know  
we are deploying  
the right jar?

We could version each  
build that we give to  
QA.





And we could tag revisions in source control.

We could version each build that we give to QA.



```
stage('Build') {
```

```
}
```

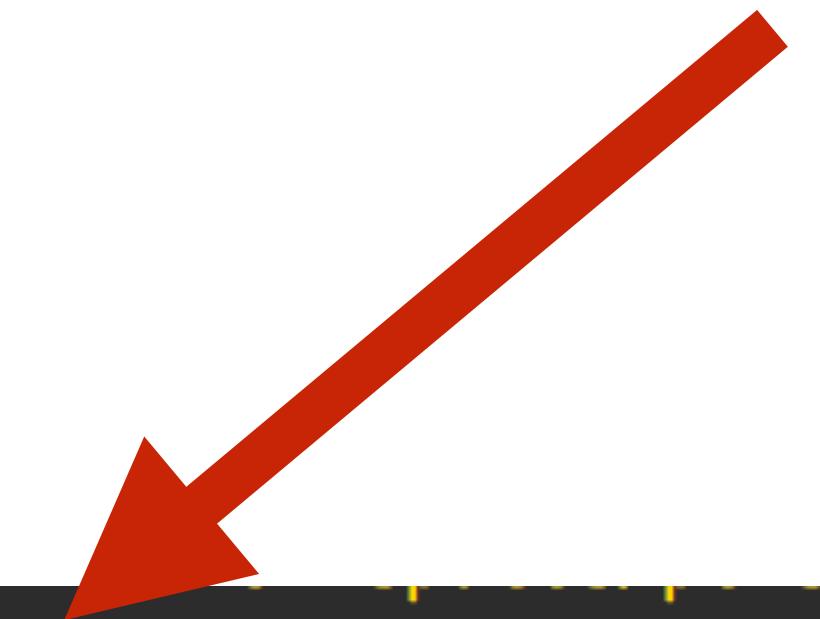
```
stage('Build') {  
    steps {  
        }  
    }  
}
```

```
stage('Build') {  
    steps {  
        sh 'ci/updateGitForPush.sh'  
    }  
}
```

```
stage('Build') {  
    steps {  
        sh 'ci/updateGitForPush.sh'  
        sh 'ci/incrementPomVersion.sh'  
    }  
}
```

```
stage('Build') {  
    steps {  
        sh 'ci/updateGitForPush.sh'  
        sh 'ci/incrementPomVersion.sh'  
        sh 'mvn -B clean package  
            -DskipTests'  
    }  
}
```

Jenkinsfile ci pom.xml readme.md src



# updateGitForPush.sh

```
#/bin/bash
```

## updateGitForPush.sh

```
#/bin/bash
```

```
git checkout master
```

## updateGitForPush.sh

```
#/bin/bash

git checkout master
git reset --hard origin/master
```

## updateGitForPush.sh

```
#!/bin/bash

git checkout master
git reset --hard origin/master
git pull
```

# incrementPomVersion.sh

```
#/bin/bash
```

## incrementPomVersion.sh

```
#/bin/bash
```

```
mvn -B
```

## incrementPomVersion.sh

```
#/bin/bash
```

```
mvn -B build-helper:parse-version
```

## incrementPomVersion.sh

```
#!/bin/bash

mvn -B build-helper:parse-version
     versions:set
```

## incrementPomVersion.sh

```
#!/bin/bash

mvn -B build-helper:parse-version
    versions:set -DnewVersion=
```

## incrementPomVersion.sh

```
#!/bin/bash

mvn -B build-helper:parse-version
    versions:set -DnewVersion=
        \${parsedVersion.majorVersion}.
```

## incrementPomVersion.sh

```
#/bin/bash

mvn -B build-helper:parse-version
versions:set -DnewVersion=
\${parsedVersion.majorVersion}.
\${parsedVersion.minorVersion}.
```

## incrementPomVersion.sh

```
#/bin/bash

mvn -B build-helper:parse-version
versions:set -DnewVersion=
\${parsedVersion.majorVersion}.
\${parsedVersion.minorVersion}.
\${parsedVersion.nextIncrementalVersion}
```

## incrementPomVersion.sh

```
#/bin/bash

mvn -B build-helper:parse-version
    versions:set -DnewVersion=
        \${parsedVersion.majorVersion}.
        \${parsedVersion.minorVersion}.
        \${parsedVersion.nextIncrementalVersion}
versions:commit
```

```
stage('Tag') {
```

```
}
```

```
stage('Tag') {  
    steps {  
        }  
    }  
}
```

```
stage('Tag') {  
    steps {  
        sh 'ci/commitPomVersion.sh'  
    }  
}
```

```
stage('Tag') {  
    steps {  
        sh 'ci/commitPomVersion.sh'  
        sh 'ci/tag.sh'  
    }  
}
```

# commitPomVersion.sh

```
#/bin/bash
```

## commitPomVersion.sh

```
#/bin/bash
```

```
git add pom.xml
```

## commitPomVersion.sh

```
#/bin/bash
```

```
git add pom.xml  
git commit -m
```

## commitPomVersion.sh

```
#/bin/bash
```

```
git add pom.xml
```

```
git commit -m
```

```
"[skip ci] Updating pom.xml version"
```

## commitPomVersion.sh

```
#/bin/bash
```

```
git add pom.xml
```

```
git commit -m
```

```
  "[skip ci] Updating pom.xml version"
```

```
git push origin master
```

tag.sh

#/bin/bash

tag.sh

#/bin/bash

NEXT\_VER=\$(

)

tag.sh

```
#/bin/bash
```

```
NEXT_VER=$(  
    grep
```

```
)
```

## tag.sh

```
#/bin/bash

NEXT_VER=$(  
    grep '^<version>.*</version>$'  
)
```

## tag.sh

```
#/bin/bash

NEXT_VER=$(  
    grep '^<version>.*</version>$'  
    pom.xml  
)
```

## tag.sh

```
#!/bin/bash

NEXT_VER=$(  
    grep '^      <version>.*</version>$'  
    pom.xml | awk -F' [><]' '  
        )
```

## tag.sh

```
#!/bin/bash

NEXT_VER=$(  
    grep '^      <version>.*</version>$'  
    pom.xml | awk -F' [><]' '  
    {print $3}' )
```

## tag.sh

```
#/bin/bash

NEXT_VER=$(  
    grep '^<version>.*</version>$'  
    pom.xml | awk -F' [><]' '  
    {print $3}' )  
git tag -a $NEXT_VER -m
```

## tag.sh

```
#/bin/bash

NEXT_VER=$(  
    grep '^<version>.*</version>$'  
    pom.xml | awk -F' [><]' '  
        {print $3}' )  
git tag -a $NEXT_VER -m  
    "[skip ci] Tagging build as $NEXT_VER"
```

## tag.sh

```
#/bin/bash

NEXT_VER=$(  
    grep '^<version>.*</version>$'  
    pom.xml | awk -F' [><]' '  
        {print $3}' )  
git tag -a $NEXT_VER -m  
    "[skip ci] Tagging build as $NEXT_VER"  
git push origin --tags
```

Sam pushes  
artifacts to Nexus



Hello





Hello

It's good that we're  
tagging builds, but...





Hello

We need to be able to  
prove which artifact  
we are deploying.





# Why is that?

We need to be able to  
prove which artifact  
we are deploying.





## Why is that?

Auditing. Artifacts must be stored where developers only have read access.





We could push to  
a Nexus repo as  
part of the build.

Auditing. Artifacts must  
be stored where  
developers only have  
read access.

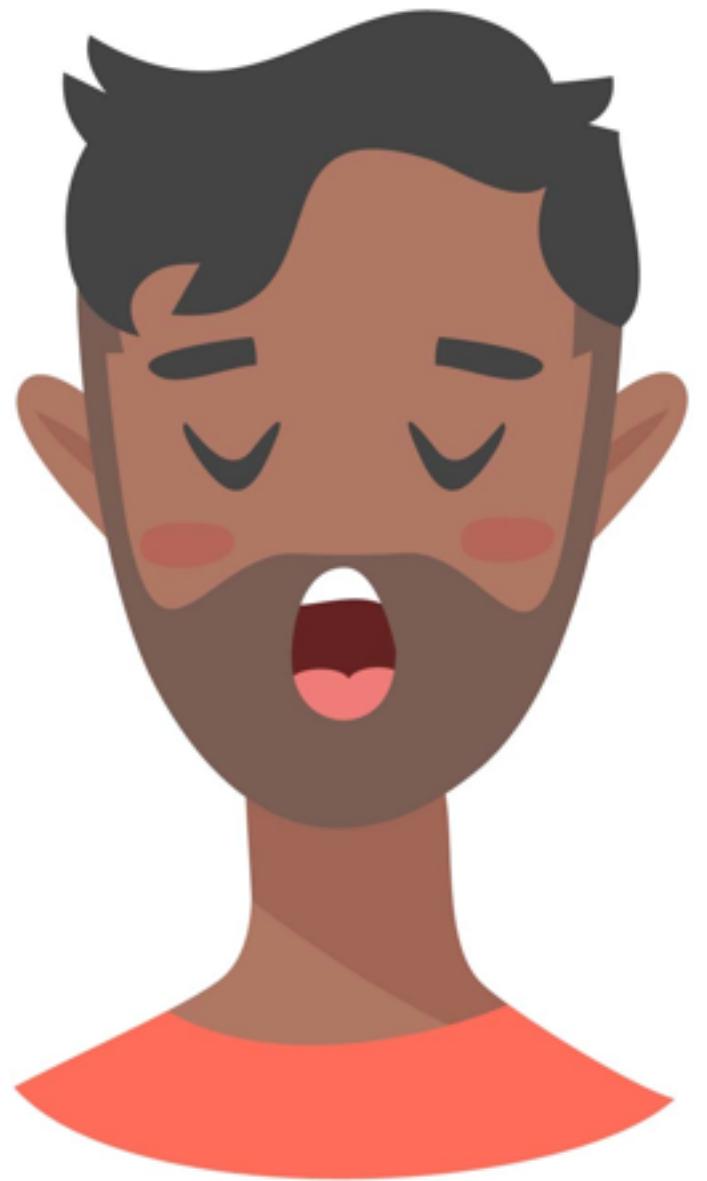


```
stage('Push jar to Nexus') {  
}  
}
```

```
stage('Push jar to Nexus') {  
    steps {  
        }  
    }  
}
```

```
stage('Push jar to Nexus') {  
    steps {  
        sh 'mvn -B deploy -DskipTests'  
    }  
}
```

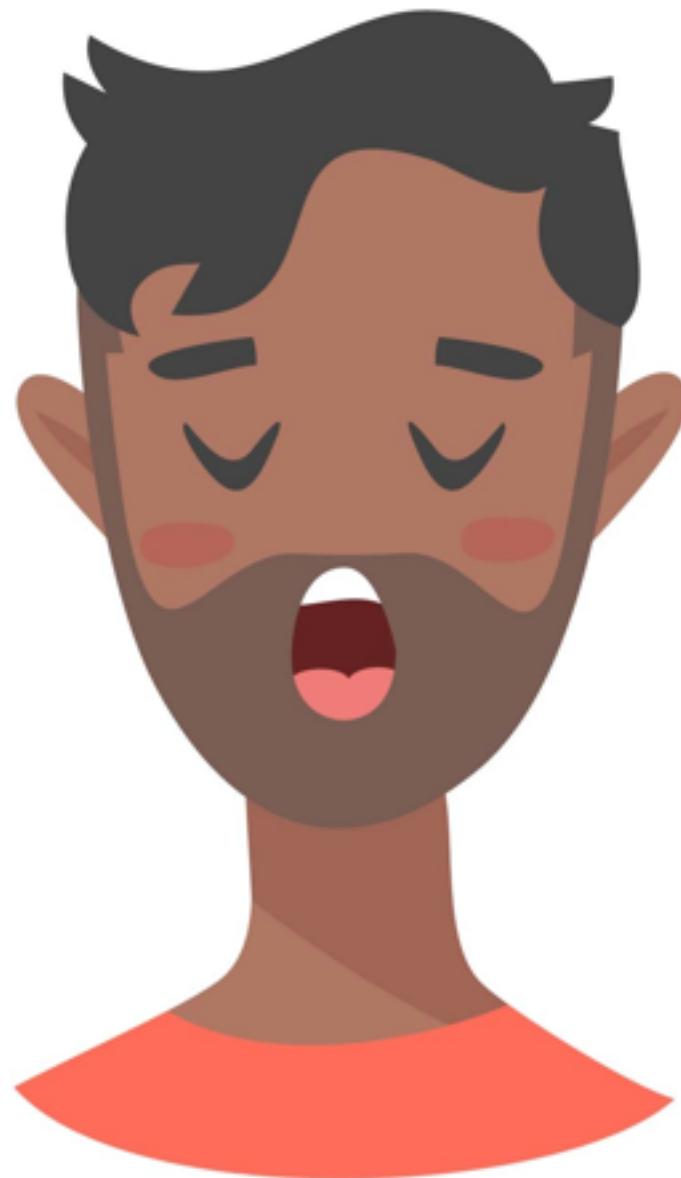
# Sam adds tests to the build



Hello?



Tech Ops



Hello?

The application is not responding again!



Tech Ops



I think I've figured it  
out. I'll need to deploy  
a fix.





All done. The app is  
back up.





Confirmed. Everything  
is back to normal.





Hmm...



How do we reduce  
the risk of bugs  
making it into  
production?





How do we reduce  
the risk of bugs  
making it into  
production?

Well, we have unit  
tests...





How do we reduce  
the risk of bugs  
making it into  
production?

but some of them are  
broken.





How do we reduce  
the risk of bugs  
making it into  
production?

We run them  
locally...





How do we reduce  
the risk of bugs  
making it into  
production?

but, we ignore the  
failures.





Let's fix or remove  
any broken tests.





Let's fix or remove  
any broken tests.

And we can run the  
tests as a part of each  
build.





Can we pair up with  
QA to automate tests  
of our most critical  
features?





Can we pair up with  
QA to automate tests  
of our most critical  
features?

Sure. We can do one  
scenario per sprint.



```
stage('Build') {...}
```

```
stage ('Tag') {...}
```

```
stage('Build') {...}
```

```
stage ('Test') {
```

```
}
```

```
stage ('Tag') {...}
```

```
stage('Build') {...}
```

```
stage ('Test') {
```

```
  steps {
```

```
  }
```

```
}
```

```
stage ('Tag') {...}
```

```
stage('Build') {...}

stage ('Test') {
    steps {
        sh 'mvn -B test'

    }
}

stage ('Tag') {...}
```

```
stage('Build') {...}

stage ('Test') {
    steps {
        sh 'mvn -B test'
        sh 'mvn -B
            failsafe:
            integration-test'
    }
}

stage ('Tag') {...}
```

Sam adds health  
checks to the app



Who detected the  
outages?





Who detected the outages?

Customers called in because they couldn't access the site.





Hmm...



Can we detect issues  
before our customers?





Can we detect issues  
before our customers?

We could add health  
checks and monitors.





And we could use the health checks to detect bad deployments before they go live.

We could add health checks and monitors.



```
stage('Deploy for test') {  
}  
}
```

```
stage('Deploy for test') {  
    steps {  
        }  
    }  
}
```

```
stage('Deploy for test') {  
    steps {  
        sh 'ci/deploy.sh test 8091'  
    }  
}
```

deploy.sh

`#!/bin/bash`

# deploy.sh

```
#!/bin/bash
DIR=$( cd "$( dirname "$
{BASH_SOURCE[0]}" )" && pwd )"
```

# deploy.sh

```
#!/bin/bash
DIR="$( cd "$( dirname "$
{BASH_SOURCE[0]}" )" && pwd )"
target=$1
port=$2
```

## deploy.sh

```
#!/bin/bash
DIR="$( cd "$( dirname "$
{BASH_SOURCE[0]}" )" && pwd )"
target=$1
port=$2
```

\$DIR/shutdown.sh \$port || true

## deploy.sh

```
#!/bin/bash
DIR="$( cd "$( dirname "$
{BASH_SOURCE[0]}" )" && pwd )"
target=$1
port=$2

$DIR/shutdown.sh $port || true
$DIR/startup.sh $target $port
```

## deploy.sh

```
#!/bin/bash
DIR="$( cd "$( dirname "$
{BASH_SOURCE[0]}" )" && pwd )"
target=$1
port=$2
```

```
$DIR/shutdown.sh $port || true
$DIR/startup.sh $target $port
while $DIR/isDown.sh $port
do
```

done

## deploy.sh

```
#!/bin/bash
DIR="$( cd "$( dirname "$
{BASH_SOURCE[0]}" )" && pwd )"
target=$1
port=$2

$DIR/shutdown.sh $port || true
$DIR/startup.sh $target $port
while $DIR/isDown.sh $port
do
    echo "Waiting on http://localhost:
$port"
done
```

## deploy.sh

```
#!/bin/bash
DIR="$( cd "$( dirname "$
{BASH_SOURCE[0]}" )" && pwd )"
target=$1
port=$2

$DIR/shutdown.sh $port || true
$DIR/startup.sh $target $port
while $DIR/isDown.sh $port
do
    echo "Waiting on http://localhost:
$port"
    sleep 5
done
```

Sam adds acceptance  
tests to the build



I analyzed the last issue that caused an outage.





I analyzed the last issue that caused an outage.

What did you find out?





Our unit and  
integration tests  
wouldn't have caught  
it.

What did you find out?





Our unit and integration tests wouldn't have caught it.

It's time to start running our acceptance test suite with each build.





Thanks QA!!!!





Thanks QA!!!!

No problem. Let's see  
those tests in action.



```
stage('Run acceptance tests') {  
}  
}
```

```
stage('Run acceptance tests') {  
    steps {  
        }  
    }  
}
```

```
stage('Run acceptance tests') {  
    steps {  
        sh 'ci/  
            checkoutAcceptanceTests.sh'  
    }  
}
```

```
stage('Run acceptance tests') {  
    steps {  
        sh 'ci/  
            checkoutAcceptanceTests.sh'  
        sh 'cd ../acceptance-tests'  
    }  
}
```

```
stage('Run acceptance tests') {  
    steps {  
        sh 'ci/  
            checkoutAcceptanceTests.sh'  
        sh 'cd ../acceptance-tests'  
        sh 'mvn clean install  
            -Dport=8091'  
    }  
}
```

# checkoutAcceptanceTests.sh

```
#!/bin/bash
```

## checkoutAcceptanceTests.sh

```
#!/bin/bash
```

```
if [ [ ! -d ../acceptance-tests ]]; then
```

```
fi
```

## checkoutAcceptanceTests.sh

```
#!/bin/bash

if [[ ! -d ../acceptance-tests ]]; then
    git clone git://localhost:9418/cdd-
workshop-acceptance-tests

fi
```

## checkoutAcceptanceTests.sh

```
#!/bin/bash

if [[ ! -d ../acceptance-tests ]]; then
  git clone git://localhost:9418/cdd-
workshop-acceptance-tests
  ..../acceptance-tests
fi
```

## checkoutAcceptanceTests.sh

```
#!/bin/bash

if [[ ! -d ../acceptance-tests ]]; then
  git clone git://localhost:9418/cdd-
workshop-acceptance-tests
  ../acceptance-tests
fi
```

cd ../acceptance-tests

## checkoutAcceptanceTests.sh

```
#!/bin/bash

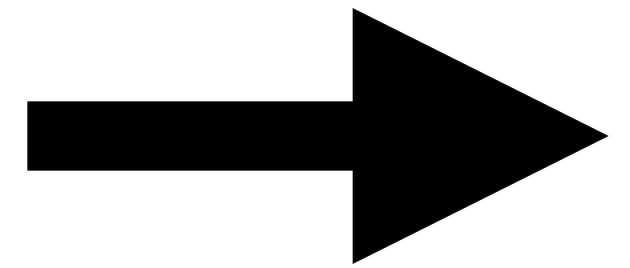
if [[ ! -d ../acceptance-tests ]]; then
    git clone git://localhost:9418/cdd-
workshop-acceptance-tests
    ../acceptance-tests
fi

cd ../acceptance-tests
git pull
```

# An aside about QA...













Sam adds feature  
toggles to the app



I've noticed our stories  
are taking a long  
time to develop.

We break them down to  
the smallest feature  
that delivers value.





Why can't we make stories smaller than a feature?

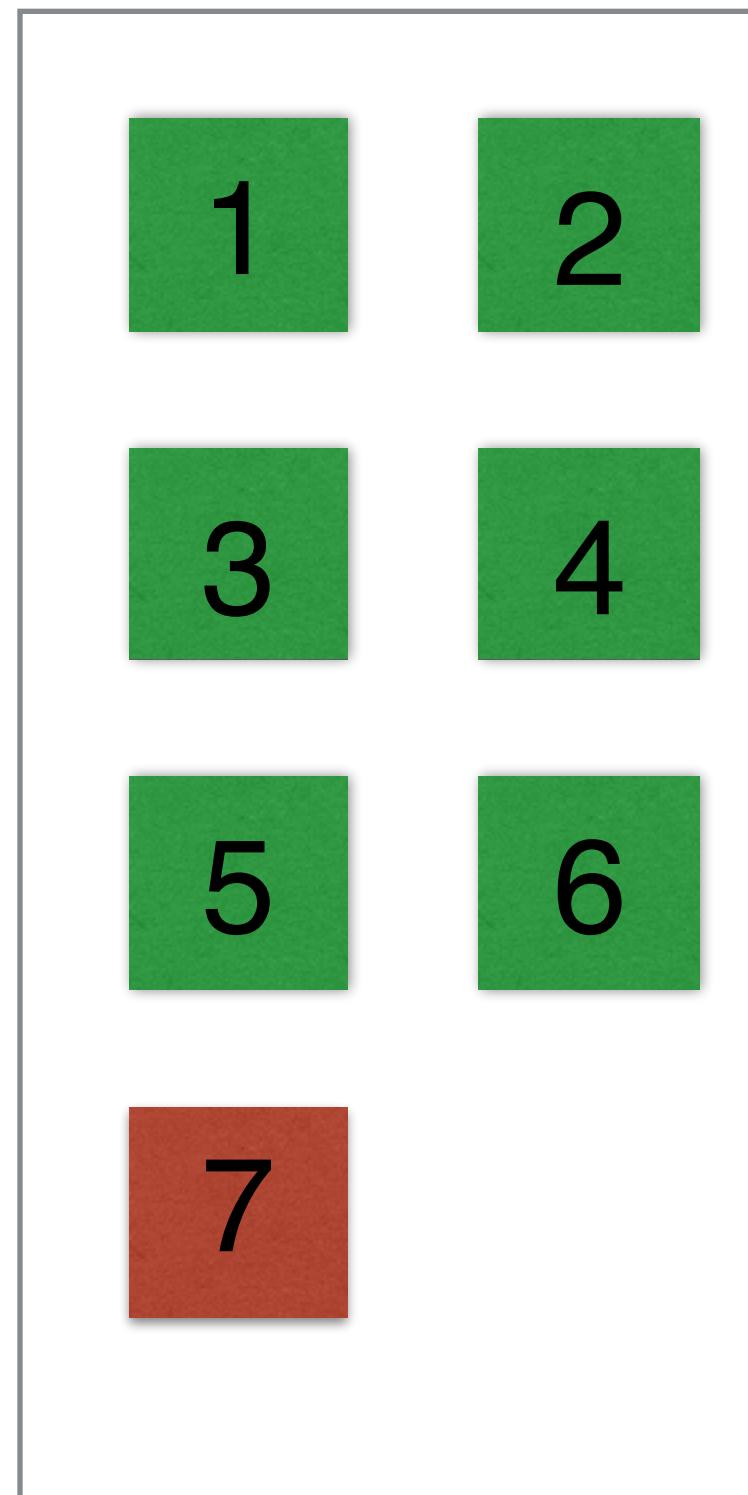
Well, how would we deploy partial features?





Hmm...

# Feature Flags



1 = active

2 = active

3 = active

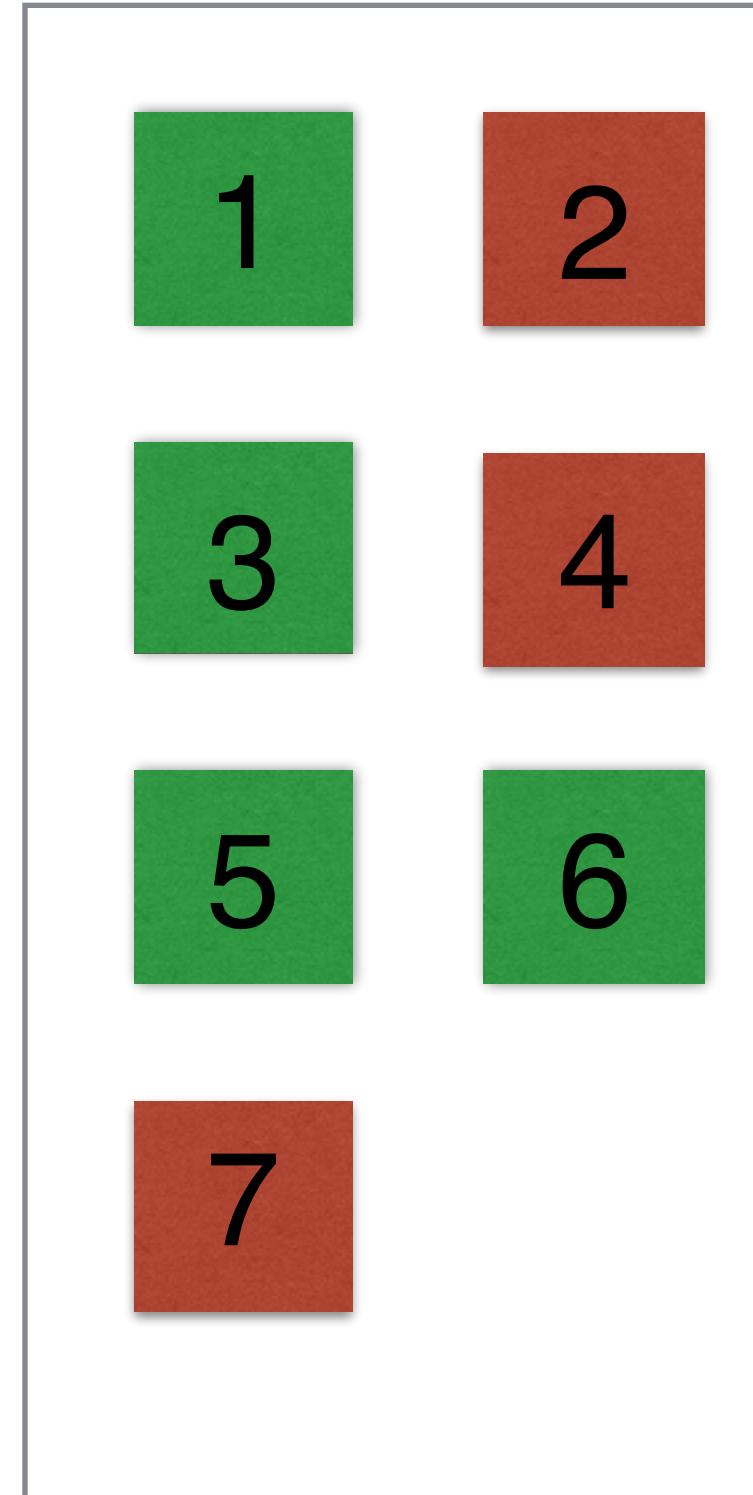
4 = active

5 = active

6 = active

7 = not active

# Feature Flags



1 = active

2 = not active

3 = active

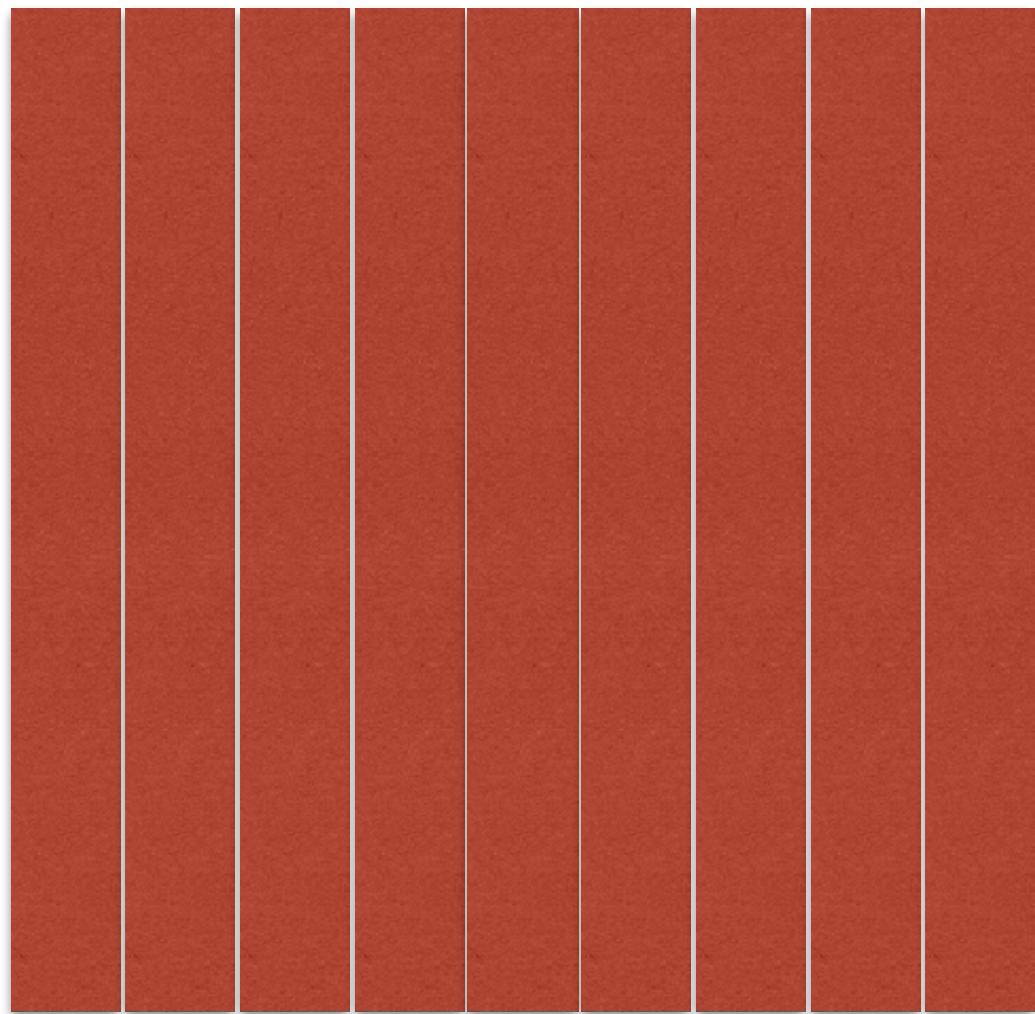
4 = not active

5 = active

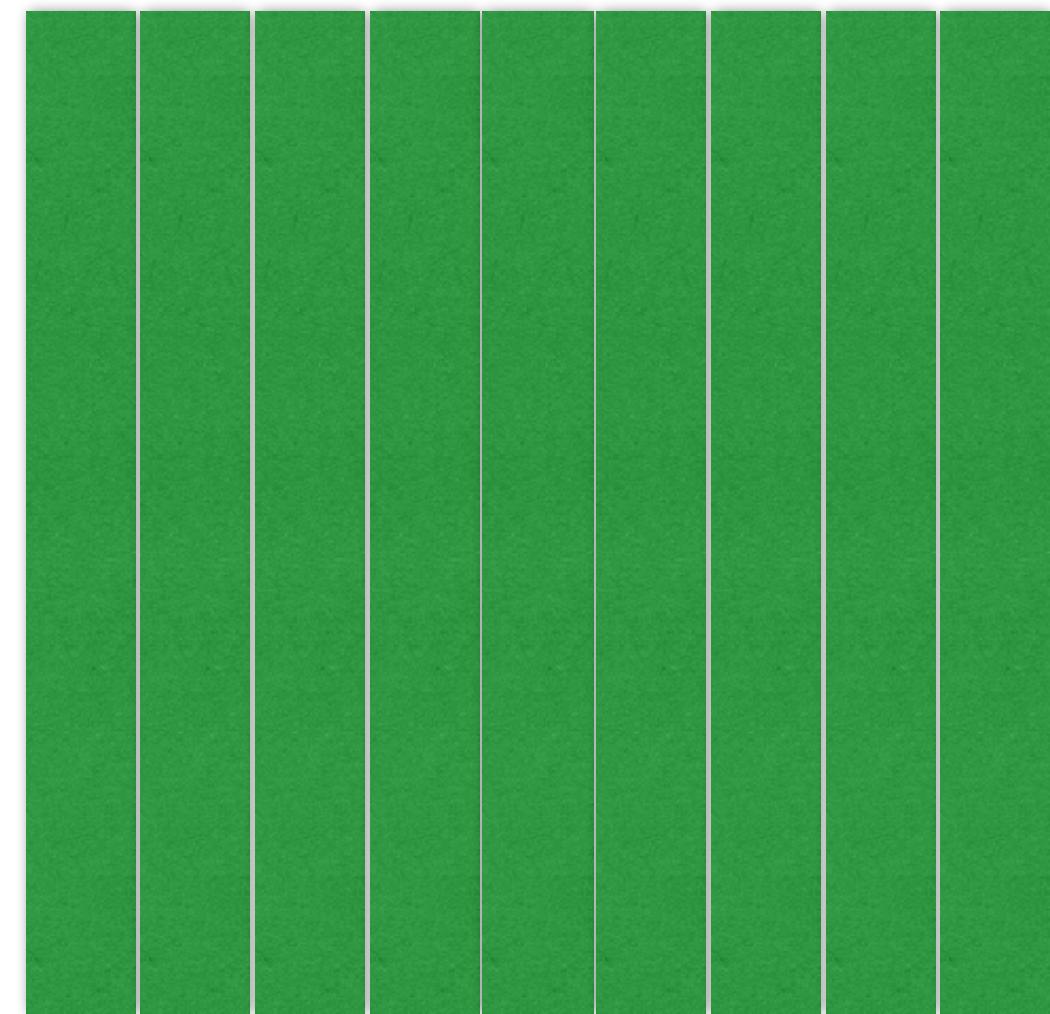
6 = active

7 = not active

# Feature Complete!



# Feature Activated!



Sam adds parallel  
tests to the build



The builds are slowing down.

We keep adding integration and acceptance tests.





I've looked at the tests  
and the ones we have  
are all appropriate.

What can we do make  
them run faster?





Hmm...



What if we partition the tests and run them in parallel?



```
stage ('Test') {  
  
}  
}
```

```
stage ('Test') {  
    steps {  
        }  
    }  
}
```

```
stage ('Test') {
    steps {
        parallel( unitTests: {
            }
        }
    }
}
```

```
stage ('Test') {
    steps {
        parallel( unitTests: {
            sh 'mvn -B test'
        } ,
        }
    }
}
```

```
stage ('Test') {
    steps {
        parallel( unitTests: {
            sh 'mvn -B test'
        },
        integrationTests: {
            }
        }
    }
}
```

```
stage ('Test') {
    steps {
        parallel( unitTests: {
            sh 'mvn -B test'
        },
        integrationTests: {
            sh 'mvn failsafe:integration-test'
        },
    }
}
```

```
stage ('Test') {
    steps {
        parallel( unitTests: {
            sh 'mvn -B test'
        },
        integrationTests: {
            sh 'mvn failsafe:integration-test'
        },
        acceptanceTests: {
            }
        }
    }
}
```

```
stage ('Test') {
    steps {
        parallel( unitTests: {
            sh 'mvn -B test'
        },
        integrationTests: {
            sh 'mvn failsafe:integration-test'
        },
        acceptanceTests: {
            sh 'ci/
                checkoutAcceptanceTests.sh'
            sh 'cd ../acceptance-tests'
            sh 'mvn clean install
                -Dport=8091'
        })
    }
}
```

Sam uses feature  
toggles for a/b  
testing



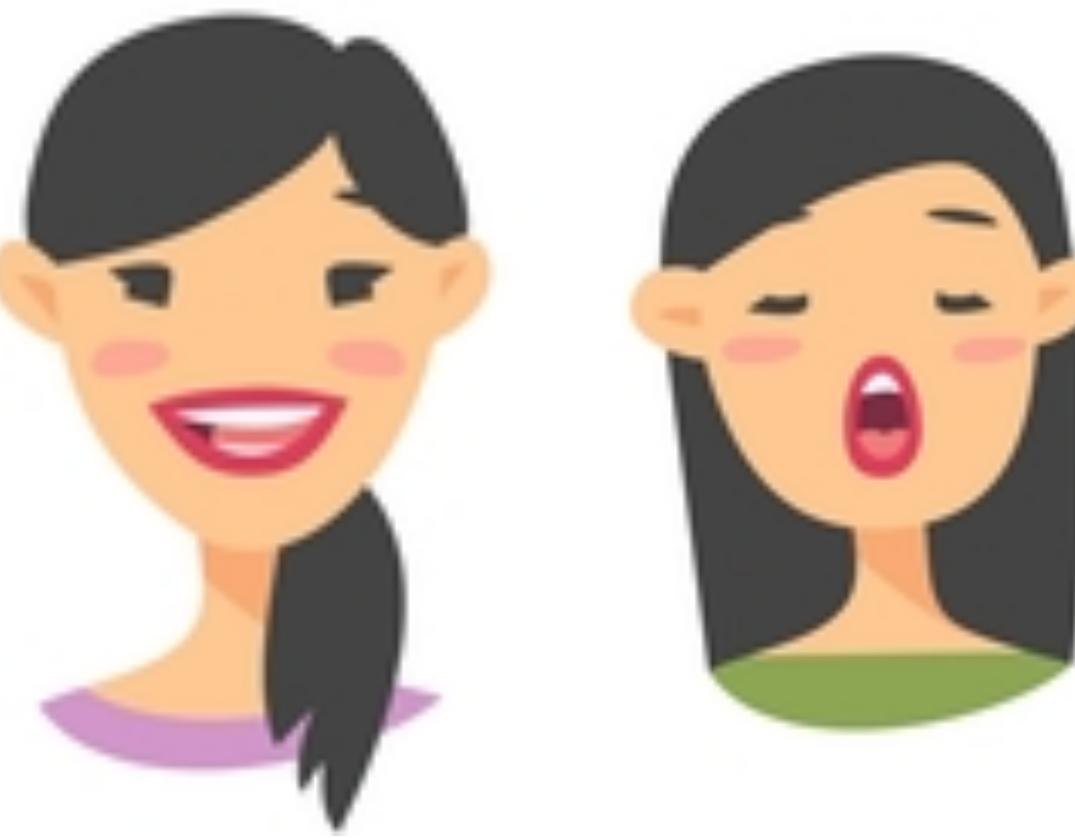
# How can I help?





## How can I help?

We can't agree  
on how to build  
this feature.





How can I help?

Which way should  
we do it?





Hmmm...

Which way should  
we do it?





We could implement  
the feature both ways.

Which way should  
we do it?





We could implement  
the feature both ways.

How?





We'll leverage  
feature toggles to  
switch between  
implementations.

How?





We'll leverage  
feature toggles to  
switch between  
implementations.

How will we know  
which feature to  
keep?





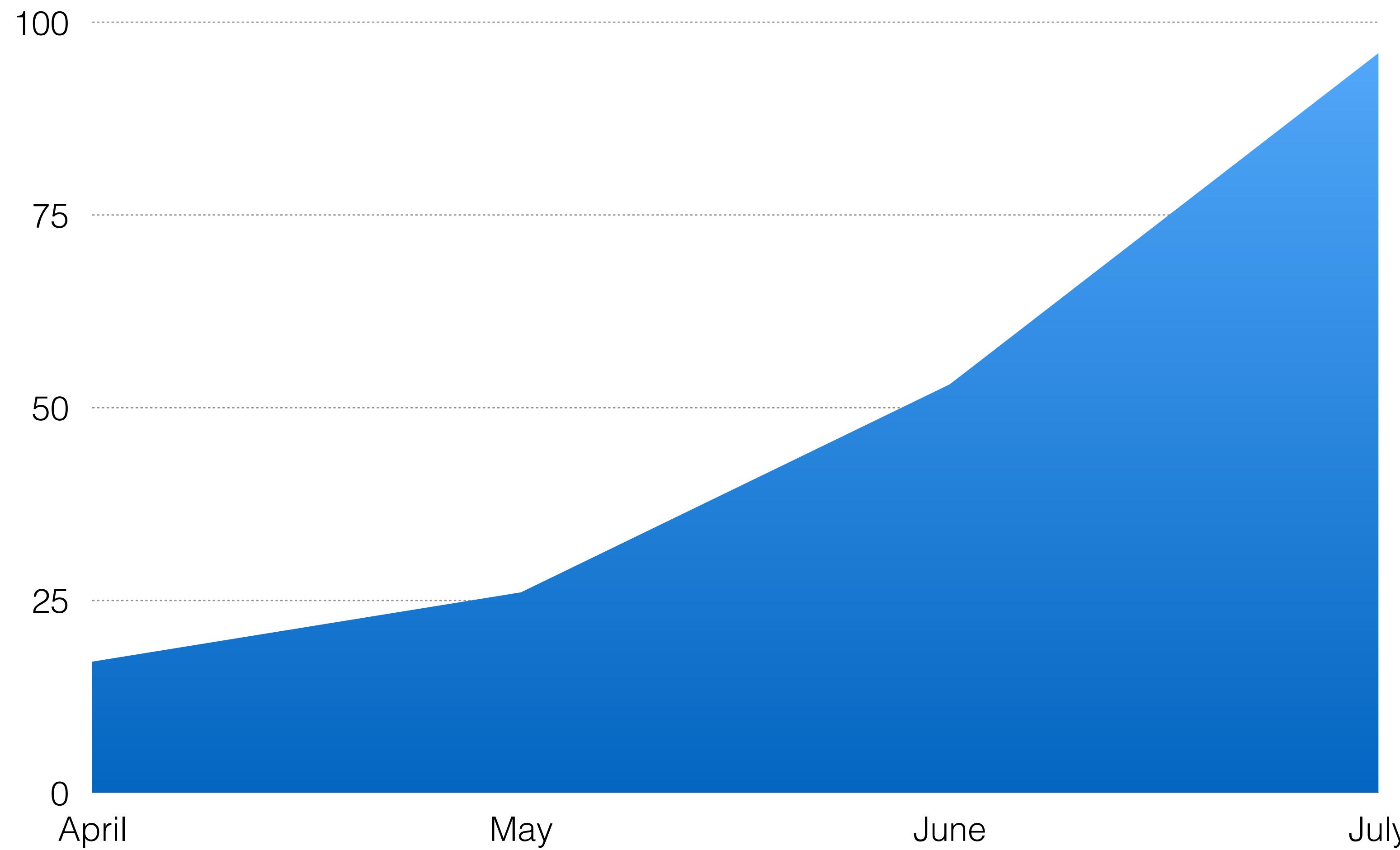
We'll gather metrics  
of each feature from  
users and evaluate.

How will we know  
which feature to  
keep?

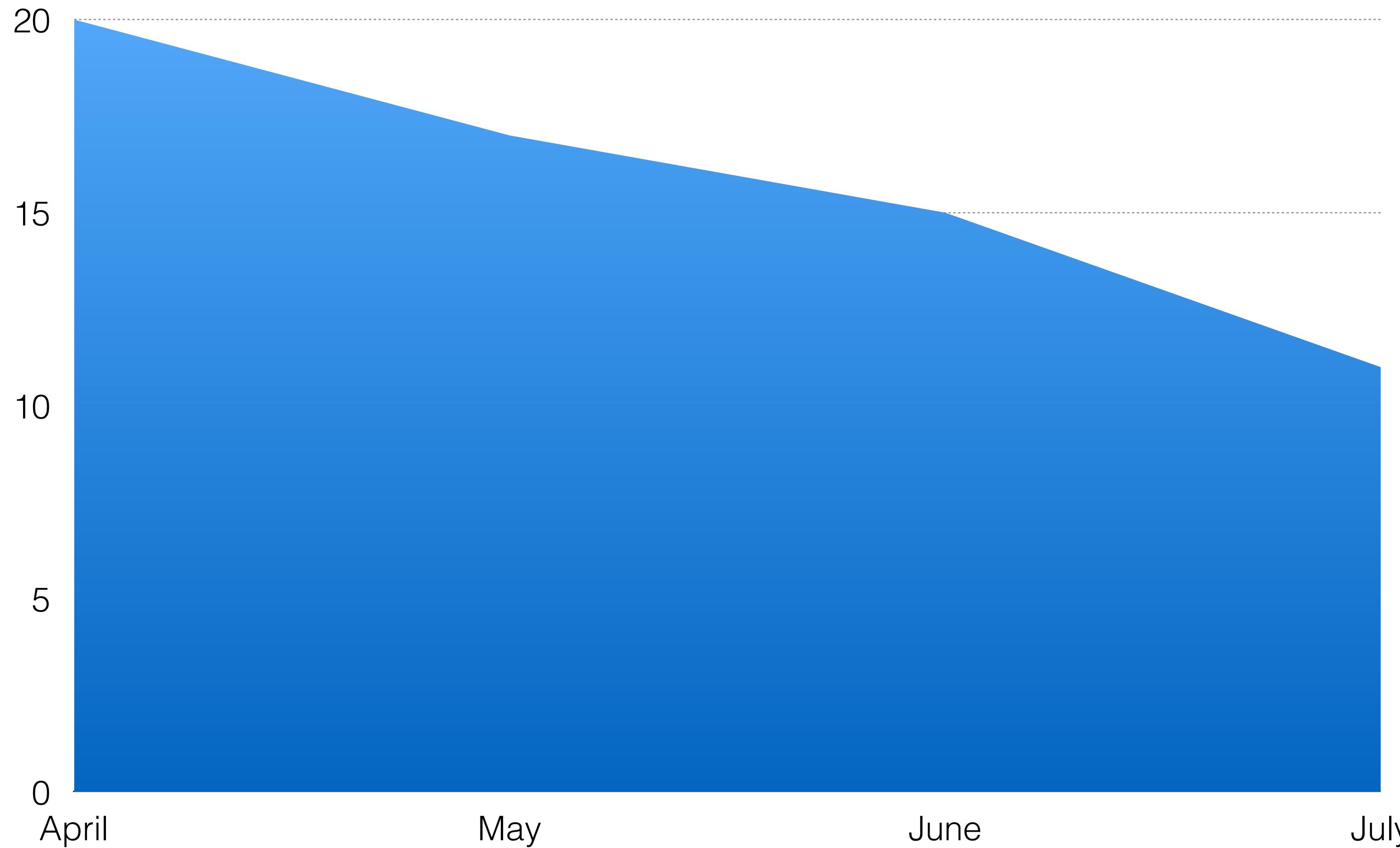


Sam automates  
deployments to  
production

# Code Coverage



# Code Complexity



It has been 163 days  
since our last deployment  
failure.



The team has really  
stepped up. We're  
ready.





The team has really  
stepped up. We're  
ready.

Ready for what?





# Continuous deployment.

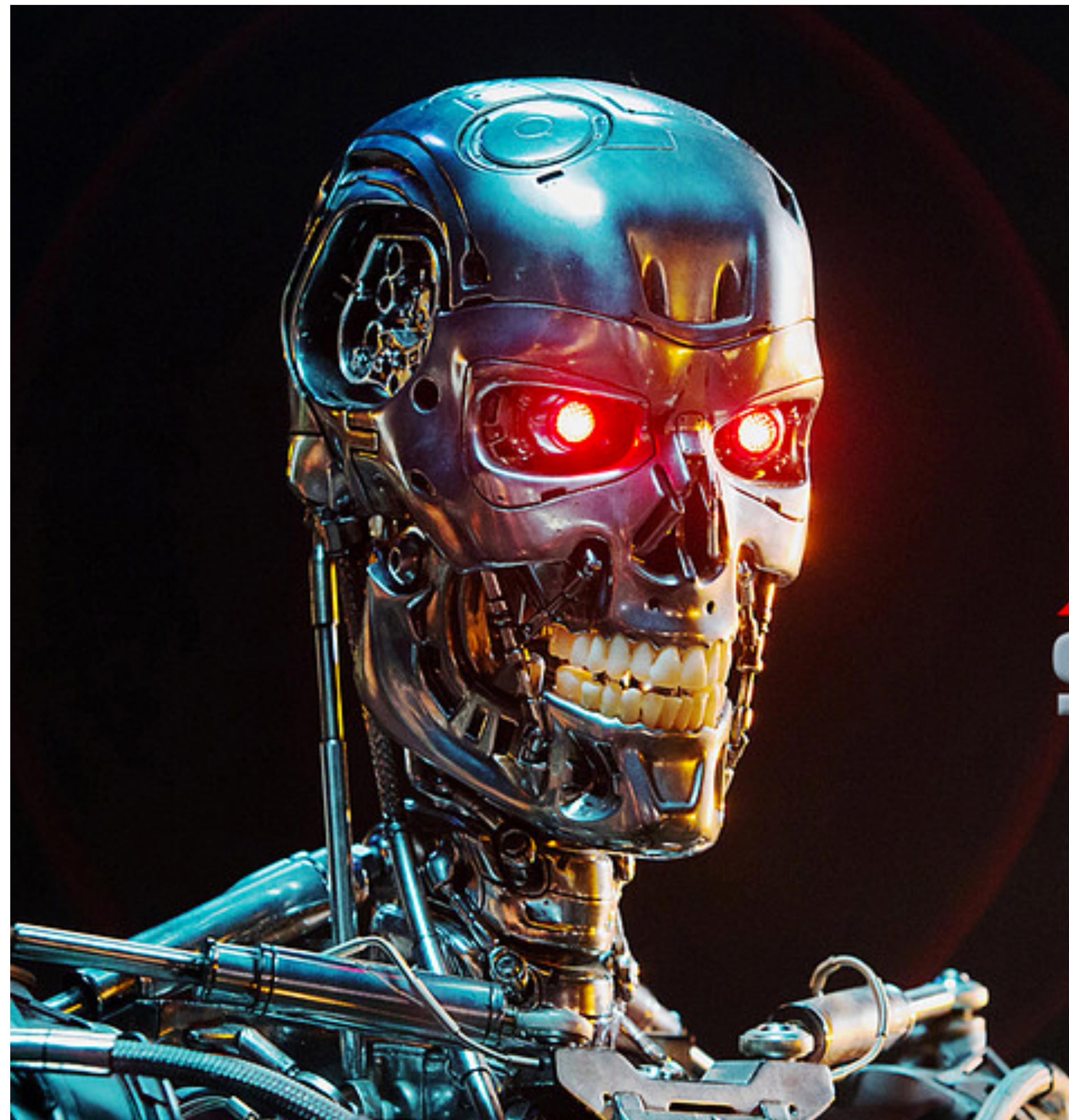
Ready for what?



# Caution!

There be dragons here!





The business should control when we deploy.



The business should control when we deploy.



You're right, but you can still have control.



# How?



You're right, but you  
can still have control.



# How?

We will deploy our changes as soon as they pass quality checks, but...





# How?

You decide when  
changes are activated  
in production.



So, I control when  
features go live like  
I've always done?



You decide when  
changes are activated  
in production.

So, I control when  
features go live like  
I've always done?



That's right!

```
stage('Deploy to prod') {
```

```
}
```

```
stage('Deploy to prod') {  
    steps {  
        }  
    }  
}
```

```
stage('Deploy to prod') {  
    steps {  
        sh 'ci/deploy.sh prod 9001'  
    }  
}
```

```
stage('Deploy to prod') {  
    steps {  
        sh 'ci/deploy.sh prod 9001'  
        sh 'ci/deploy.sh prod 9002'  
    }  
}
```

```
stage('Deploy to prod') {  
    steps {  
        sh 'ci/deploy.sh prod 9001'  
        sh 'ci/deploy.sh prod 9002'  
    }  
}
```

Sam adds blue-green  
deployments



\$10,000

\$100,000



Business  
Partner

Oh? I'll talk to my  
team.



Hmm...



We need to prevent downtime during our deployments.



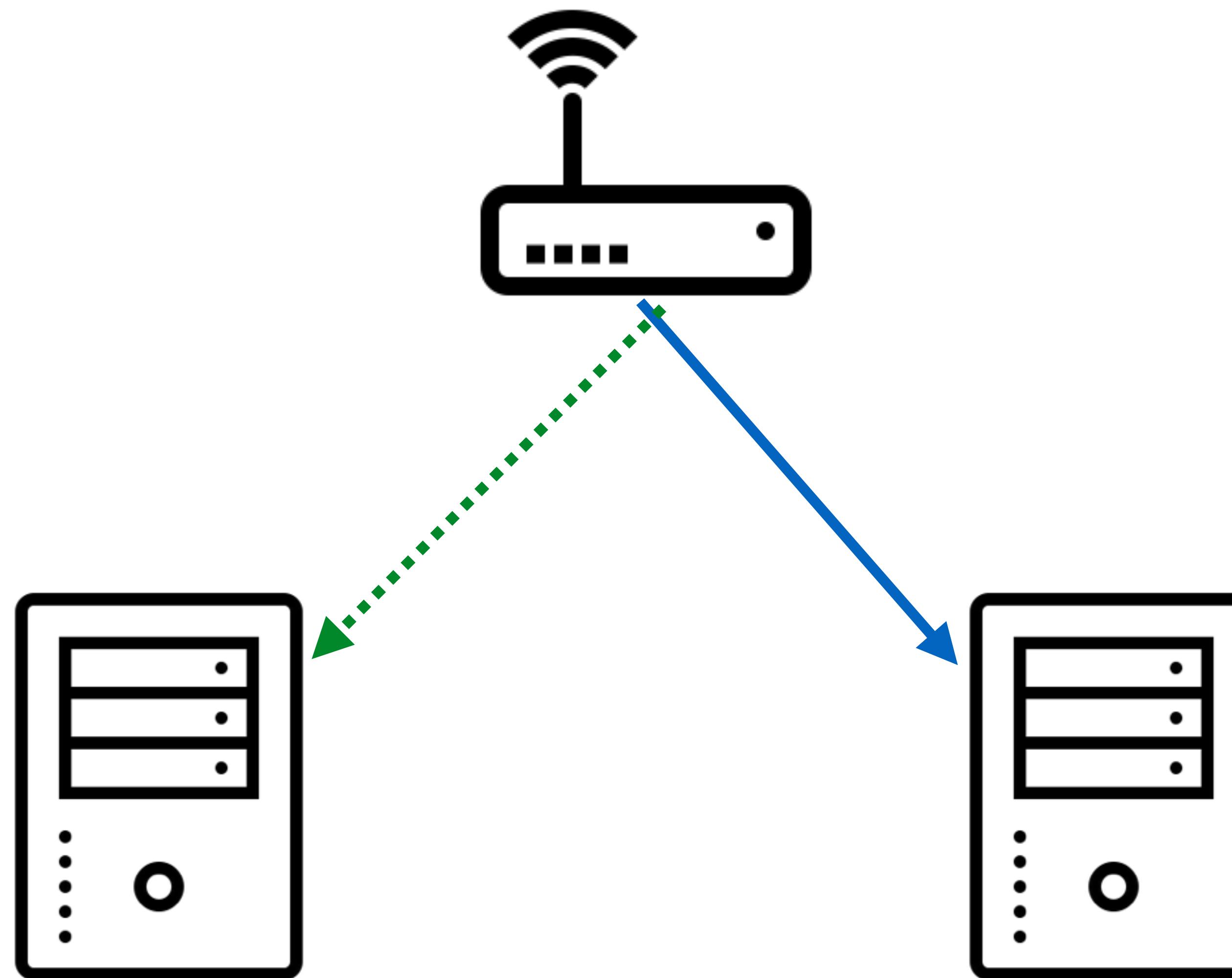


We need to prevent  
downtime during our  
deployments.

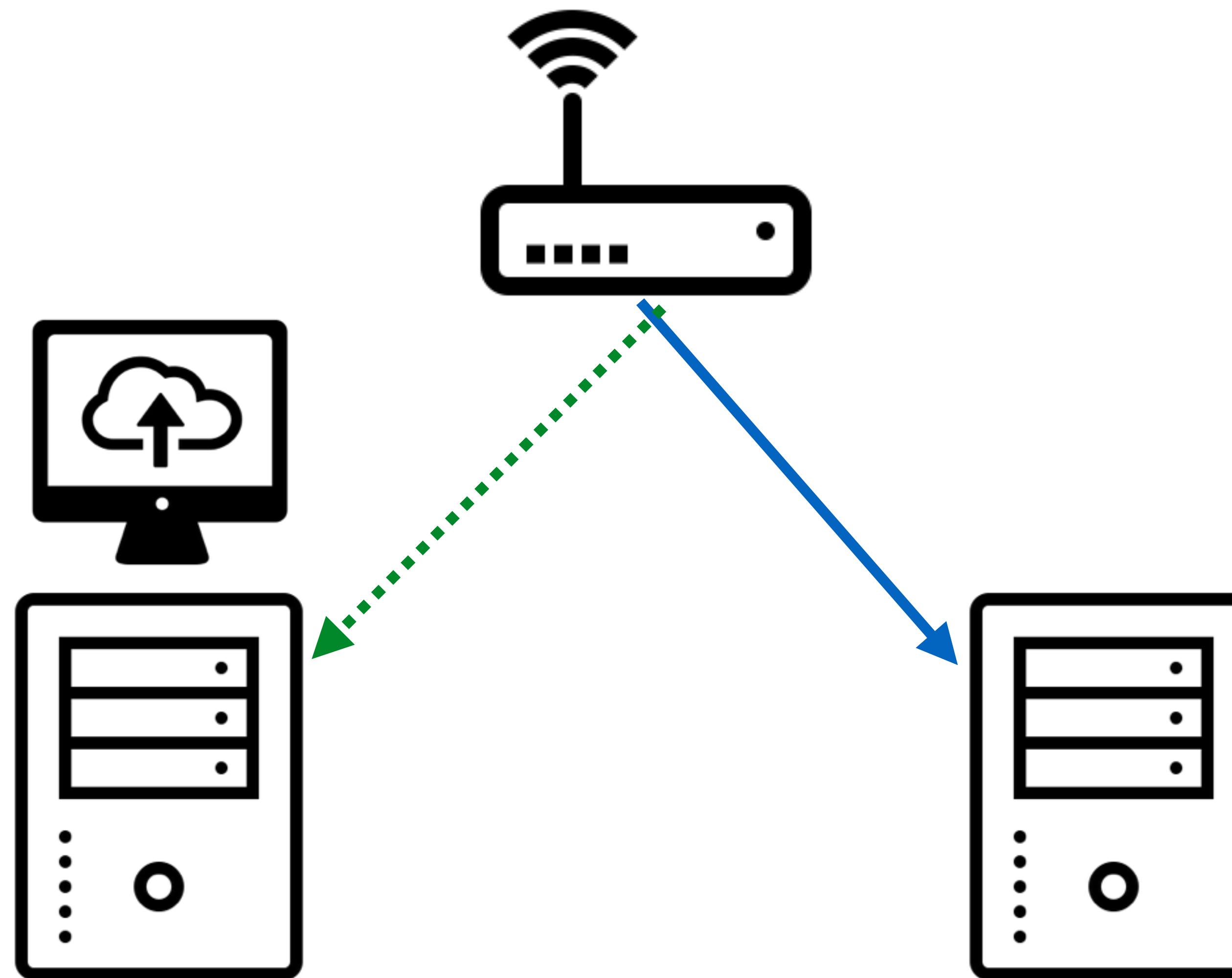
How?



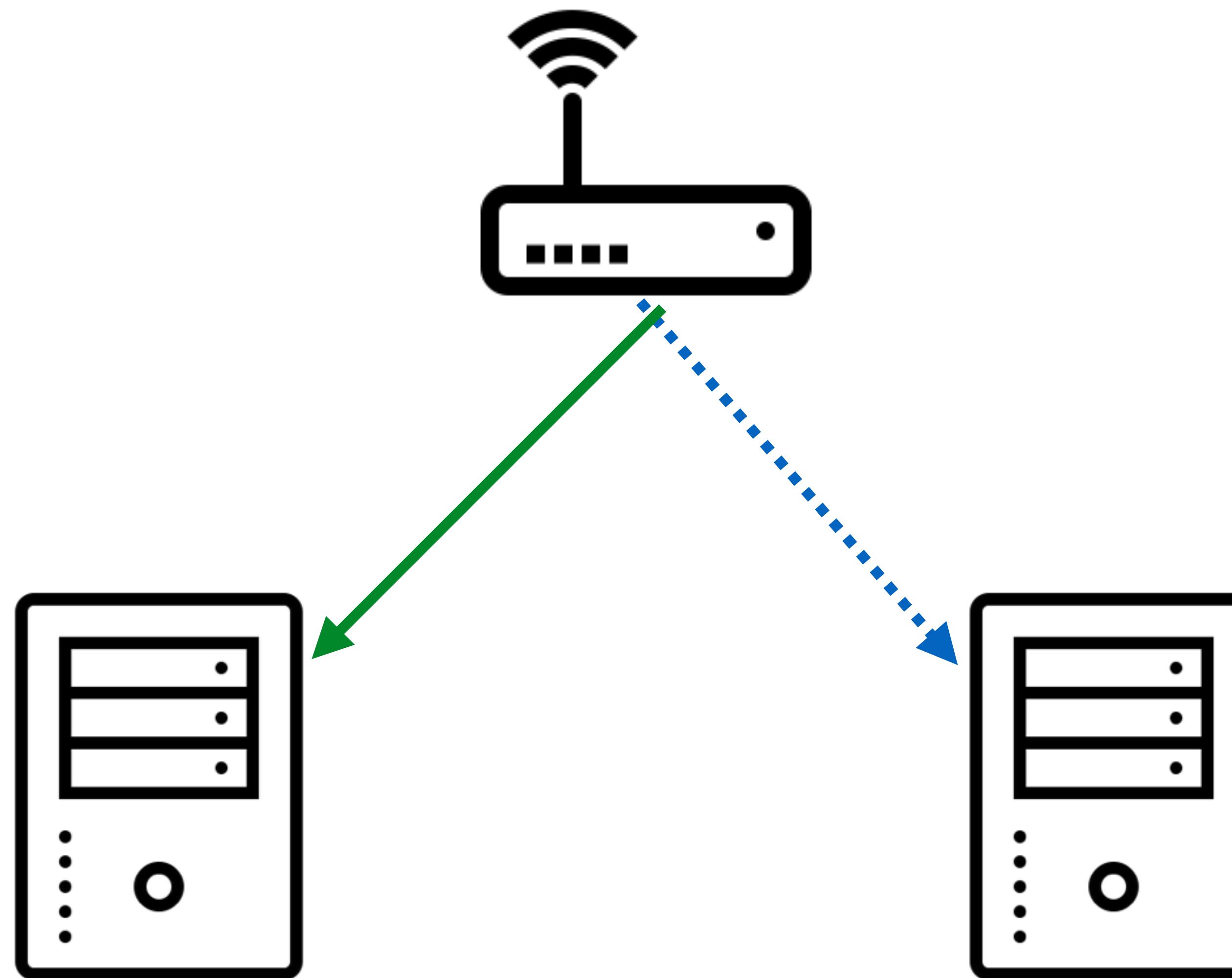
# Blue-green



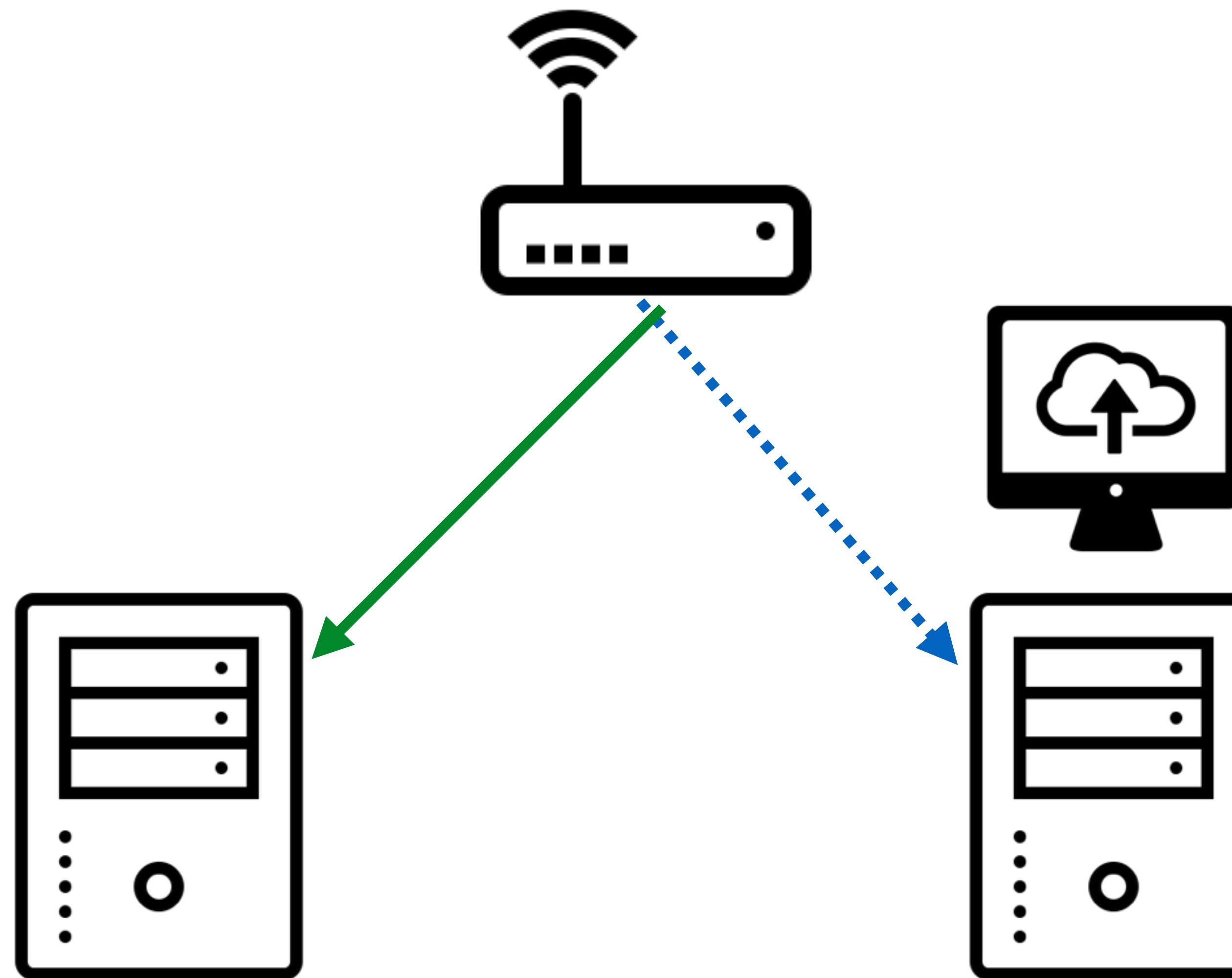
# Blue-green



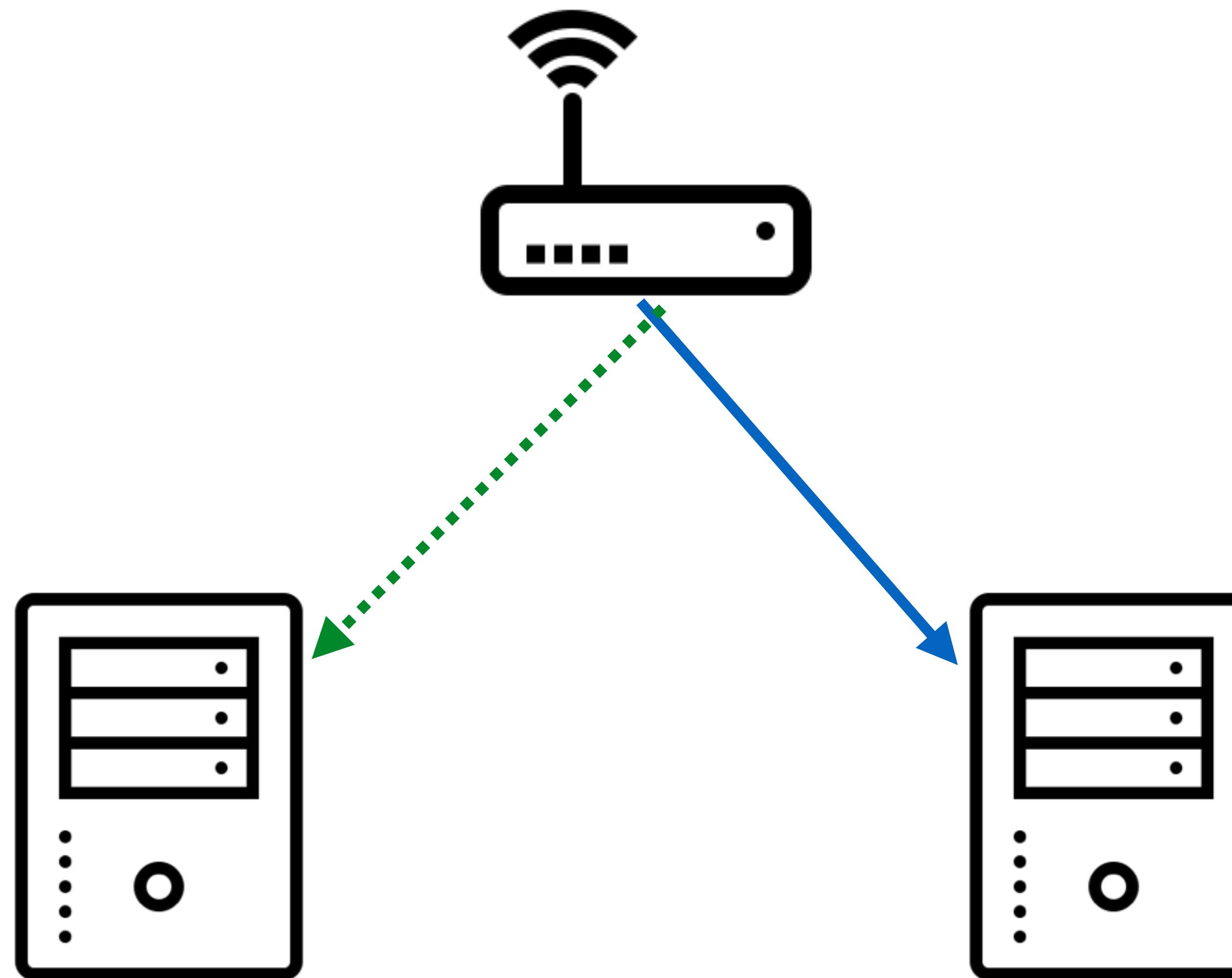
# Blue-green



# Blue-green



# Blue-green



```
stage('Deploy to prod') {  
    steps {  
        sh 'ci/deploy.sh prod 9001'  
        sh 'ci/deploy.sh prod 9002'  
    }  
}
```

```
stage('Deploy to prod') {
    steps {
        //move traffic to 9002 only
        sh 'ci/deploy.sh prod 9001'

        sh 'ci/deploy.sh prod 9002'

    }
}
```

```
stage('Deploy to prod') {
    steps {
        //move traffic to 9002 only
        sh 'ci/deploy.sh prod 9001'
        //move traffic to 9001 only
        sh 'ci/deploy.sh prod 9002'

    }
}
```

```
stage('Deploy to prod') {
    steps {
        //move traffic to 9002 only
        sh 'ci/deploy.sh prod 9001'
        //move traffic to 9001 only
        sh 'ci/deploy.sh prod 9002'
        //resume traffic to both
    }
}
```

# Bonus stage!

Jenkins

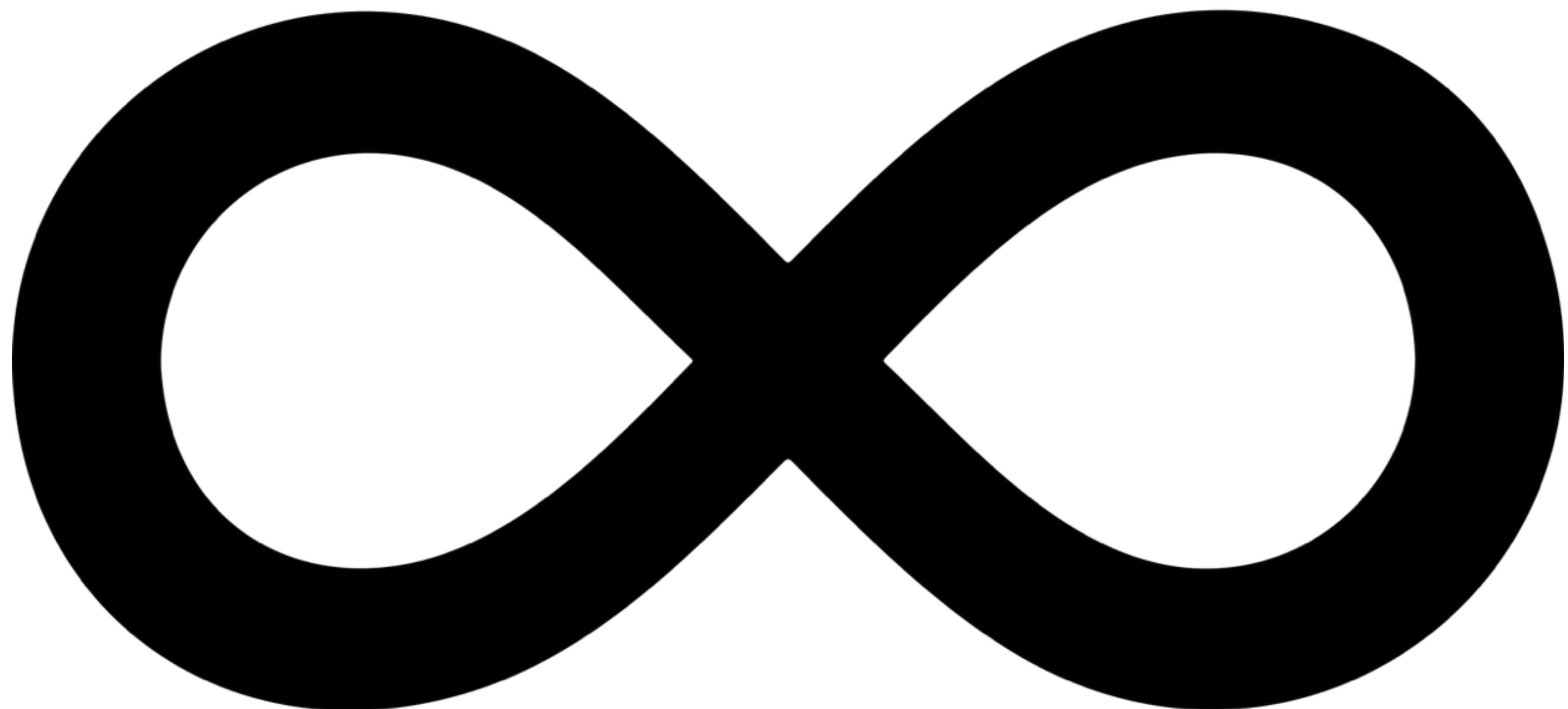
+

Maven

=

+

Versioning



## Enter an item name

pipeline\_launcher

pipeline\_launcher



### Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combin system, and this can be even used for something other than software build.

## Source Code Management

- None
- Git

### Repositories

Repository URL

git://localhost:9418/cdd-workshop

Credentials

- none -

Add

Advanced...

Add Repository

### Branches to build

Branch Specifier (blank for 'any')

\* /master

X

?

Add Branch

## Build Triggers

- Trigger builds remotely (e.g., from scripts) ?
- Build after other projects are built ?
- Build periodically ?
- GitHub hook trigger for GITScm polling ?
- Poll SCM ?

Schedule

H/5 \* \* \* \*



Would last have run at Saturday, March 24, 2018 10:50:45 PM EDT; would next run at Saturday, March 24, 2018 10:55:45 PM EDT.

Ignore post-commit hooks



## Build

### Execute shell

Command

```
if git log -1 --pretty=format:'%an | grep -q "Jenkins"; then
    echo "Build triggered by version update. Aborting..."
    exit
fi
```

X



See [the list of available environment variables](#)

Advanced...

## Post-build Actions

X



### Build other projects

Projects to build

cdd-pipeline

- Trigger only if build is stable
- Trigger even if the build is unstable
- Trigger even if the build fails

Add post-build action ▾

# Benefits

- Get To Market Faster
- Greater Confidence
- Lower Stress
- Less Risk
- Simpler Branching
- Visibility

# Watch For

- Alienating allies
- Emotional responses
- Needs of Stakeholders

# Thank You!

For full slides:

[https://github.com/jbalmert/always\\_be\\_delivering](https://github.com/jbalmert/always_be_delivering)

Twitter: @trembyl