Microsoft

WorkshopPLUS: Azure DevOps Essentials – Git

Git Workflows

Microsoft Services

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet website references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see **Use of Microsoft Copyrighted Content** at
https://www.microsoft.com/en-us/legal/intellectualproperty/permissions/default.aspx

Microsoft®, Internet Explorer®, Outlook®, SkyDrive®, Windows Vista®, Zune®, Xbox 360®, DirectX®, Windows Server® and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Module Overview

- Introduction
- Git Workflow
- Avoiding Push Conflicts
- Best Practices
- Knowledge Measure
- Summary

Lesson 1: Introduction

- After completing this lesson, you will be able to:
  - Describes DevOps and how Git workflow fits in DevOps
  - Understand how Microsoft does DevOps with our 1 Engineering System


“Judge us by the actions we have taken in the recent past, our actions today and in the future”
—Satya Nadella, CEO
Microsoft

Microsoft ♥ Open Source

2012

TypeScript released

Git support added to TFS and Visual Studio

2014

Satya "Microsoft loves Linux"

Microsoft org on GitHub created

.NET Foundation created

2015

Visual Studio Code released

HDInsight (Hadoop/ Ubuntu) announced

Microsoft jointly forms Node.js foundation

2016

.NET Core 1.0

PowerShell Core

Windows Subsystem for Linux in Windows 10

Microsoft joins Linux Foundation

GitHub recognizes Microsoft as a top open source contributor

2017

Microsoft Azure Kubernetes Service launched

Draft, Brigade, Kashti projects submitted to Kubernetes community

Microsoft joins Cloud Native Computing & Cloud Foundry Foundations

SQL 2017 on Linux

Windows source code moved to Git

Azure Databricks (Apache Spark) announced

2018

Visual Studio Code ranked #1 developer tool

Azure Service Fabric Open Sourced

Azure Sphere with Linux kernel

Intent to acquire GitHub announced

~5,000 Microsoft employees committing to open source projects on GitHub

Azure trending to 50% Linux

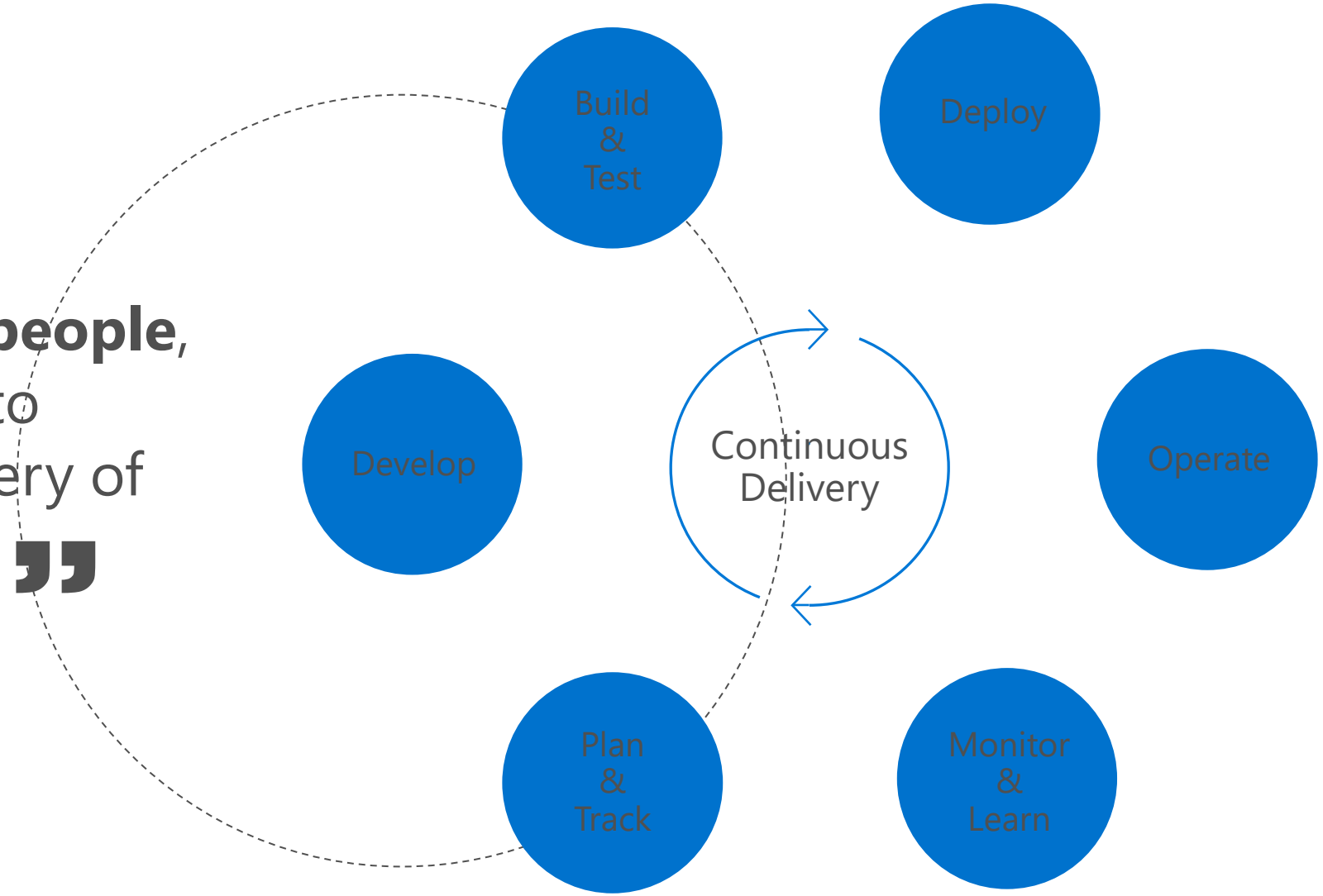
Microsoft continues as largest contributor to open source projects on GitHub

2012 2014 2015 2016 2017 2018

What is DevOps?

People. Process. Products.

> DevOps is the union of **people**, **process**, and **products** to enable continuous delivery of value to your end users.



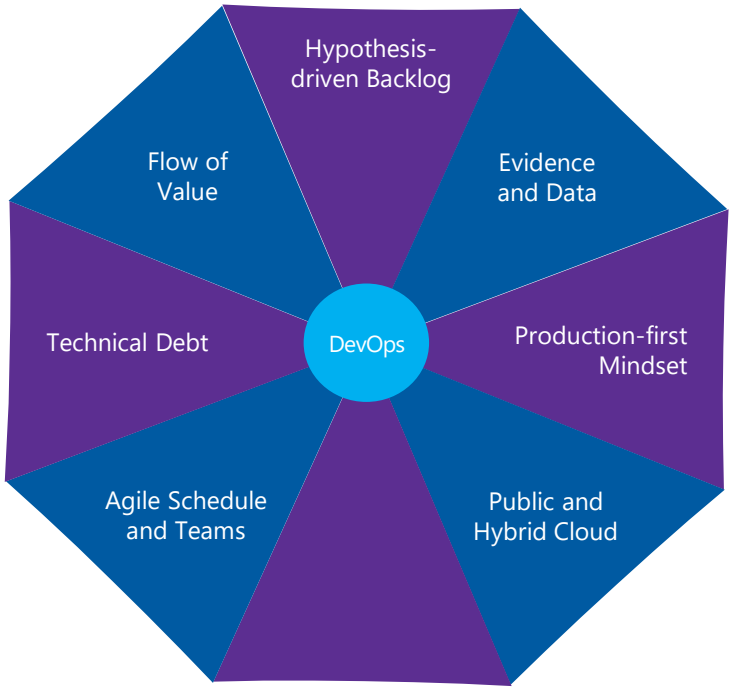
DevOps Practices for Success

Backlog
How an organization **defines and manages requirements** and how effective they are.

Flow
The **flow of value** as code moves through the system from developer environment to production, and the rate at which business value is delivered in the form of software.

Technical Debt
Behaviors and characteristics that define how an organization thinks about the decisions they make that incur **technical debt** and how they discover and manage technical debt that is accrued throughout the application lifecycle.

Agile Schedule and Teams
Behaviors and characteristics that reflect **team organization and work schedule**.



Evidence
Characteristics and behaviors that demonstrate how an organization uses **data in decision making**, including code analysis, test results and real-world usage metrics.

Production
Refers to how an organization **manages software in its production environment**, including how it detects and responds to unexpected events.

Cloud Infrastructure
Refers to the behaviors and characteristics of how the organization approaches and manages the **core infrastructure** that their systems and apps run on.

Azure DevOps

Azure Boards

Deliver value to your users faster using proven agile tools to plan, track, and discuss work across your teams.

Azure Pipelines

Build, test, and deploy with CI/CD that works with any language, platform, and cloud. Connect to GitHub or any other Git provider and deploy continuously.

Azure Repos

Get unlimited, cloud-hosted private Git repos and collaborate to build better code with pull requests and advanced file management.

Azure Test Plans

Test and ship with confidence using manual and exploratory testing tools.

Azure Artifacts

Create, host, and share packages with your team, and add artifacts to your CI/CD pipelines with a single click.

https://azure.com/devops

DevOps at Microsoft

https://aka.ms/DevOpsAtMicrosoft

372k
Pull Requests per month

4.4m
Builds per month

5m
Work items viewed per day

78,000
Deployments per day

2m
Git commits per month

500m
Test executions per day

500k
Work items updated per day

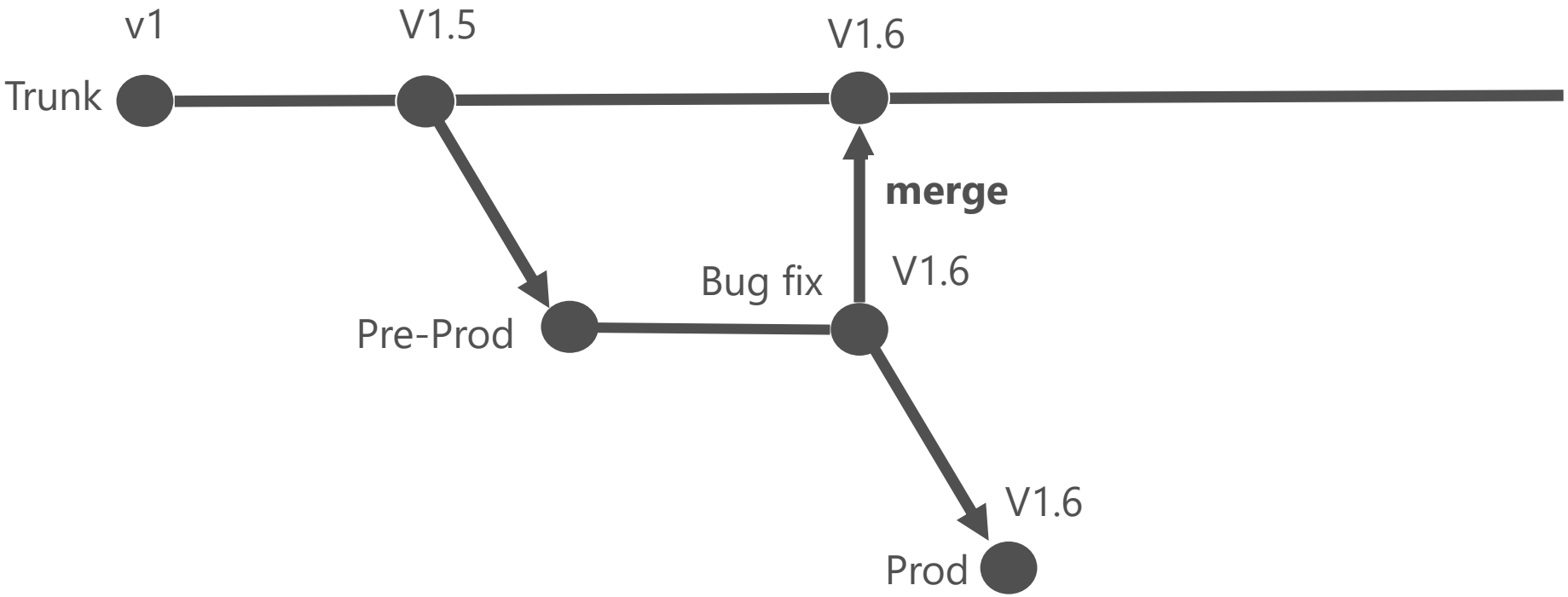
Demo: 1ES

1ES @ Microsoft

Lesson 2: Workflows

- After completing this lesson, you will be able to:
  - Understand what is meant by workflow
  - Understand how it compares to TFVC branching
  - Understand how teams tend to create branching structures
  - Learn what impacts branching strategies
  - Learn how to adopt a branching strategy
  - Understand the different branching strategies
  - Learn what is Microsoft's approach
  - Understand how you will approach branching in Git

What do we mean by workflow



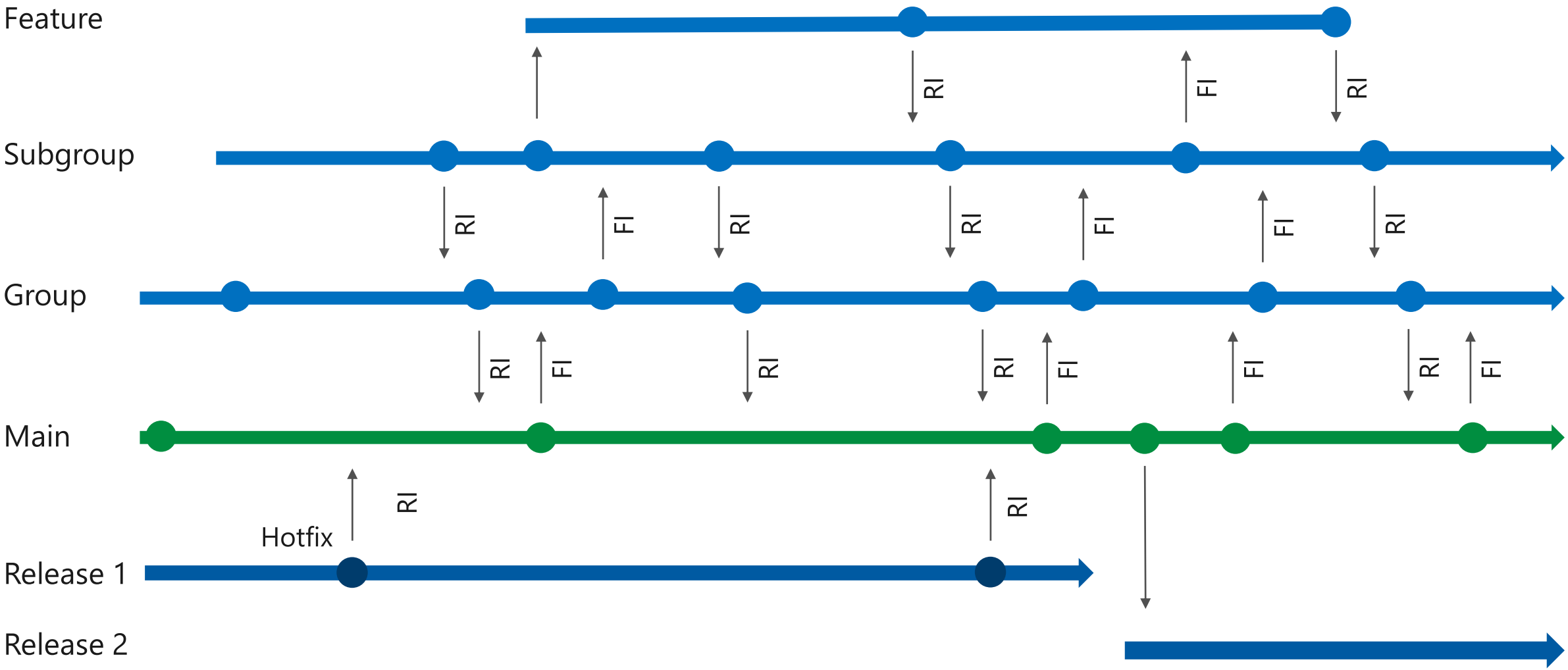
Git is very granular

GIT WORKFLOW

EQUAL TO

TFVC BRANCHING STRATEGY

Typical TFVC Branching Structure



"Organizations which design systems… are constrained to produce designs which are copies of the communication structures of these organizations…"

Conway's Law

Organizations tend to produce branching structures that copy the organization chart.

How to approach branching in Git

Strategy

Workflows

What is Microsoft's Approach

Strategy Depends On

Team Maturity
Organizational Process
Release Vehicle
Existing Workflow

Adopt a Git branching strategy

Distributed version control

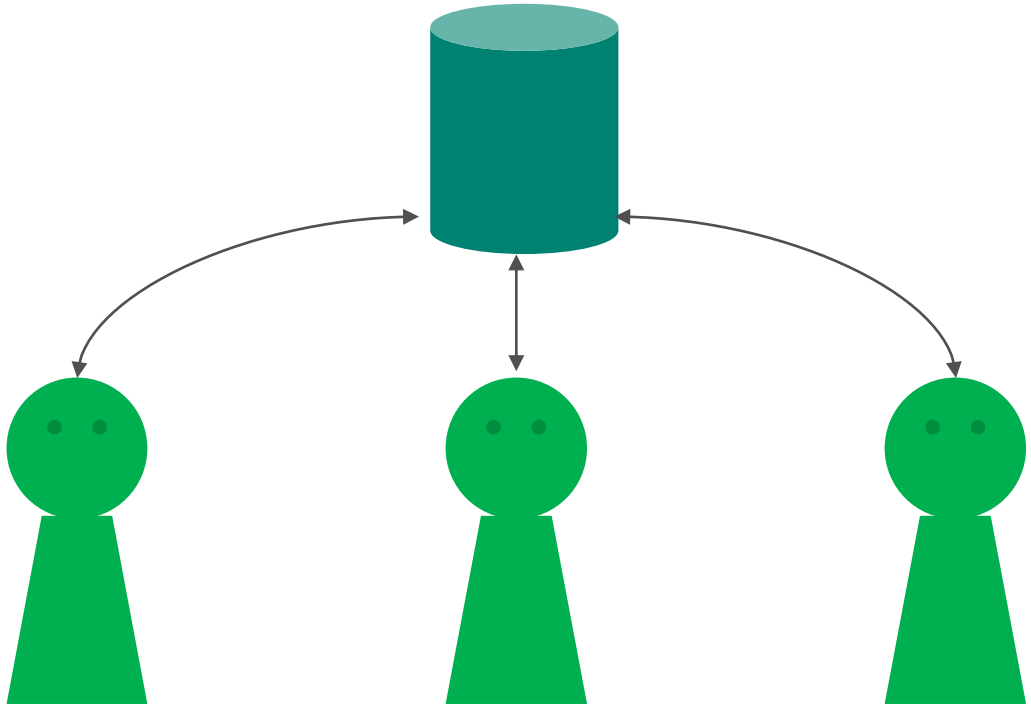
- wide flexibility
- find a balance
- publish, share, review, and iterate on code changes

Adopt a branching strategy

- collaborate better
- spend less time managing version control
- spend more time developing code

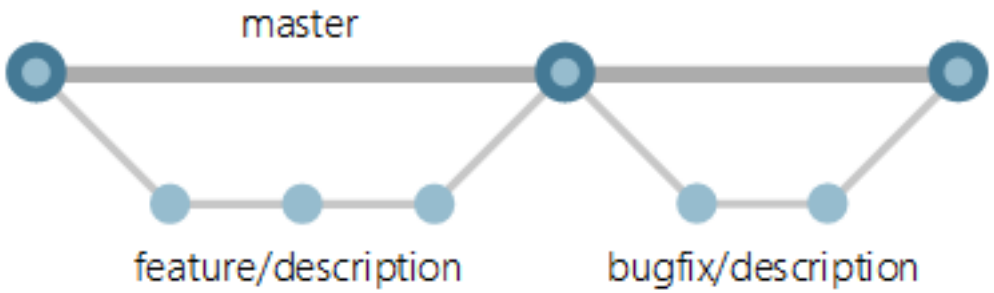
Centralized Workflow

Transitioning to a distributed version control system may seem like a daunting task, but *you don't have to change your existing workflow* to take advantage of Git.



Use feature branches for your work

- Develop your features and fix bugs in feature branches (also known as topic branches) based off your master branch.
- Feature branches isolate work in progress from the completed work in the master branch. Git branches are inexpensive to create and maintain, so even small fixes and changes should have their own feature branch.



- Creating feature branches for all your changes makes reviewing history very simple. Look at the commits made in the branch and look at the pull request that merged the branch.

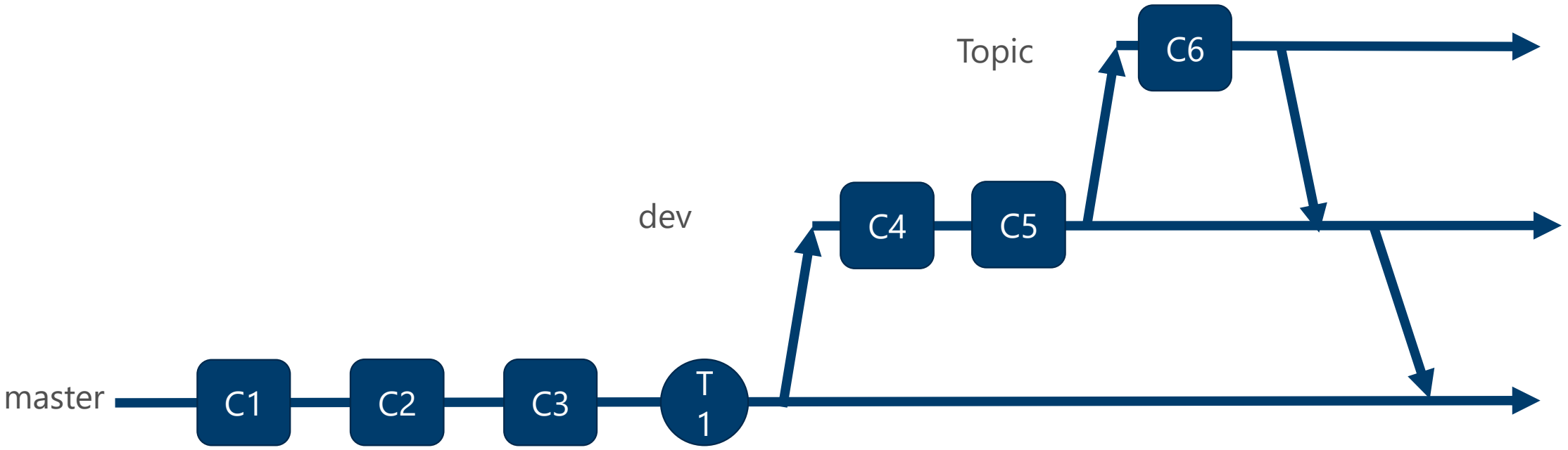
Branching strategies - Development Isolation

Maintain and protect a stable master branch

Allow isolation and concurrent development

Branch one or more dev branches from master

Can be isolated in development branches by feature, organization or temporary collaboration



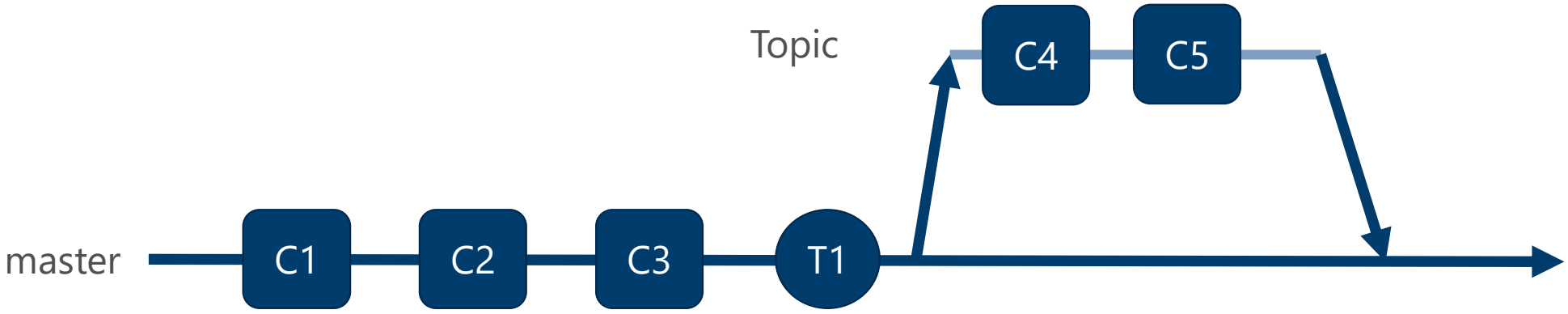
Branching strategies - Topic branch

A topic is a bug or feature that a developer is working on until it's ready to be merged into a main branch.

Not uncommon to create and delete multiple topic branches during a single day

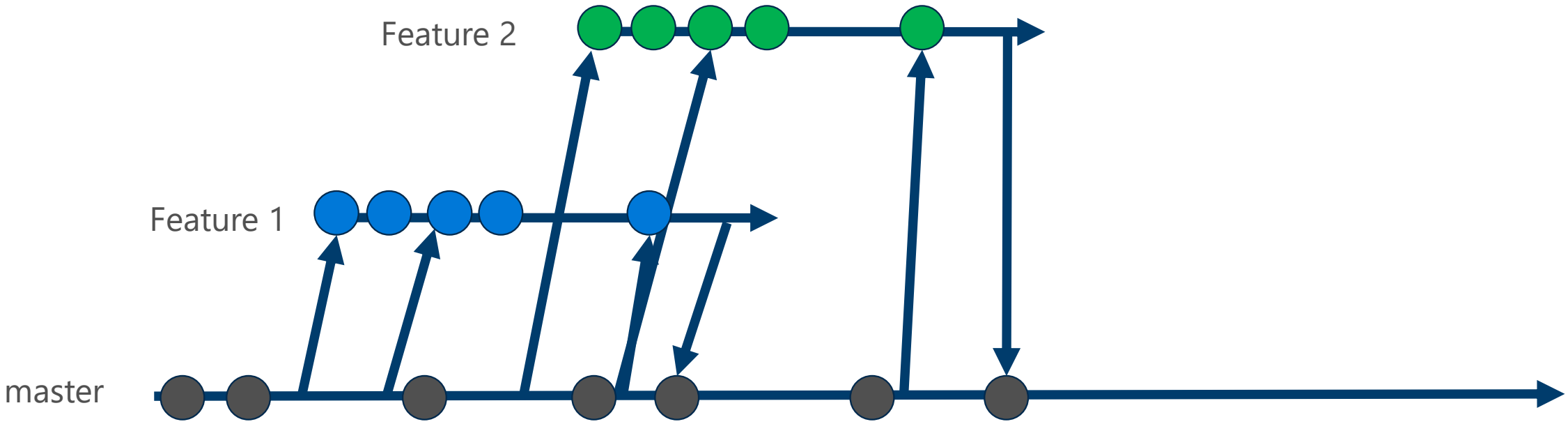
Why use topic branches?

Topic branches allow you to context-switch quickly and completely.
Easier to identify history and changes during code review.
Allows for merge when ready, regardless of the order in which they were created or worked on.
Feature development takes place in a dedicated branch instead of the master branch.
Enables multiple developers to work on a feature without disturbing master.
Master branch should be pristine; Leverage Pull Request.



Branching strategies - Feature Isolation

Isolate features into separate branches
Merge to master takes place only when the feature is stable enough
RI to master will be a single commit — easy to rollback
Always integrate frequently from master to your feature branch

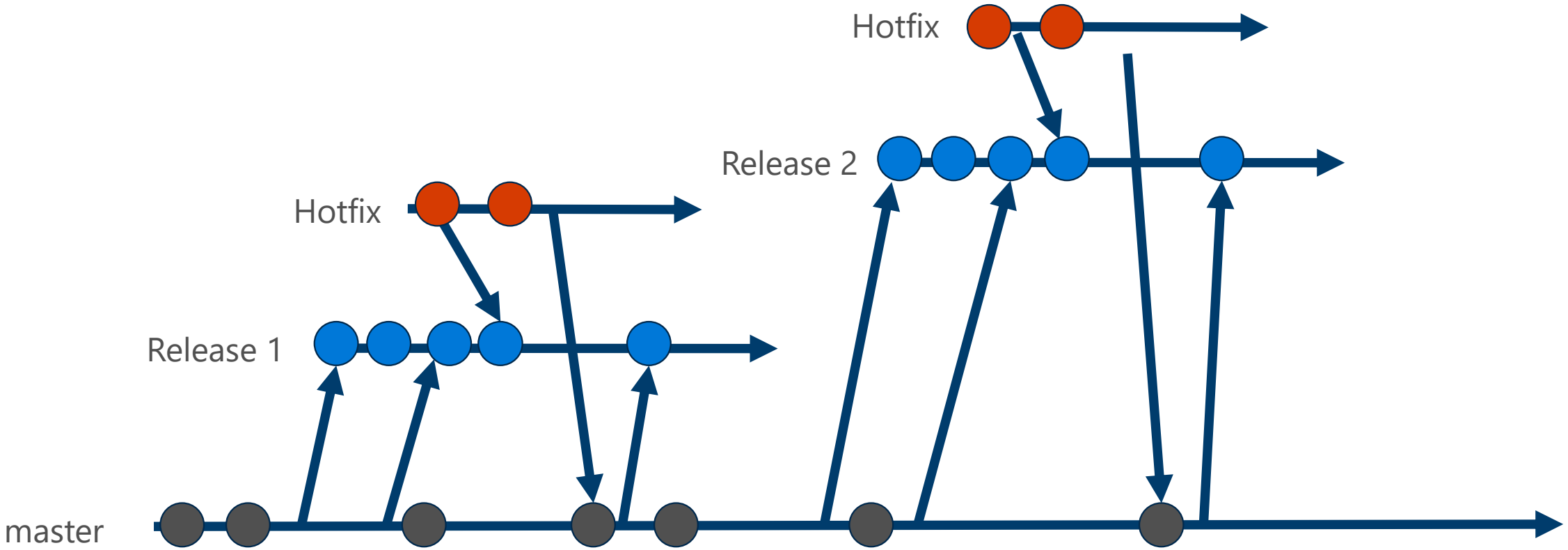


Branching strategies - Servicing and/or Release isolation

One or more release branches from master

Enable concurrent release management, multiple and parallel releases, and accurate snapshots of your codebase at release time

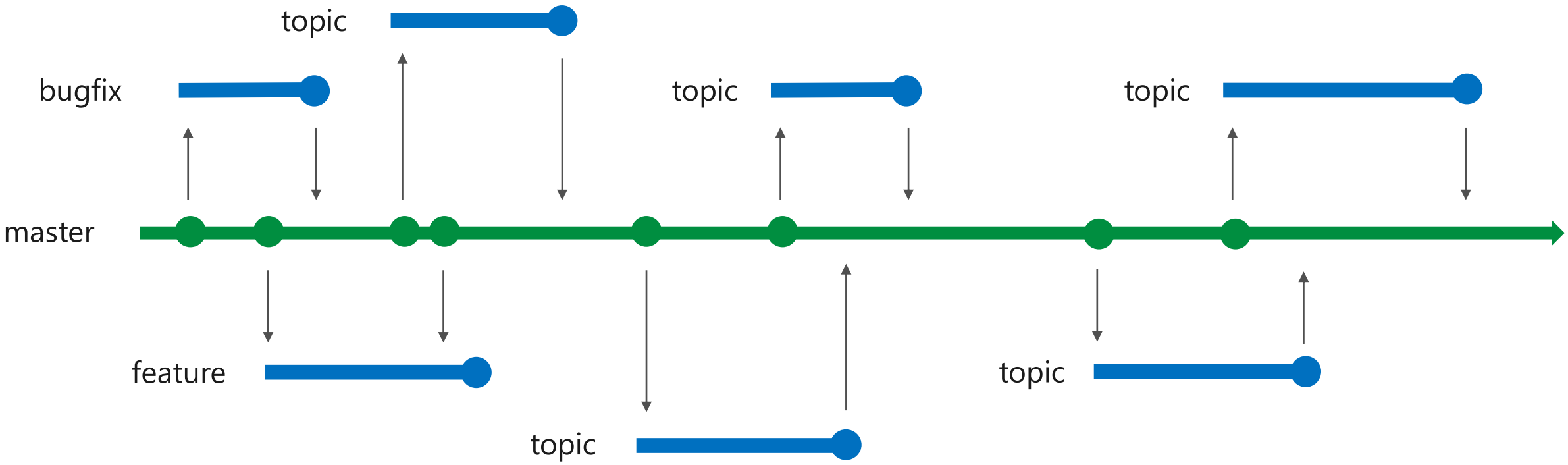
Enable concurrent servicing management of service packs, and accurate snapshots of your codebase at service and release time.



Trunk-Based Development

- Enables the following:
  - Code close to master
  - Small, simple changes
  - Fewer merge conflicts
  - Easy to code review
  - Encourages pull requests
  - Simpler to ship; faster velocity

Trunk-Based Development



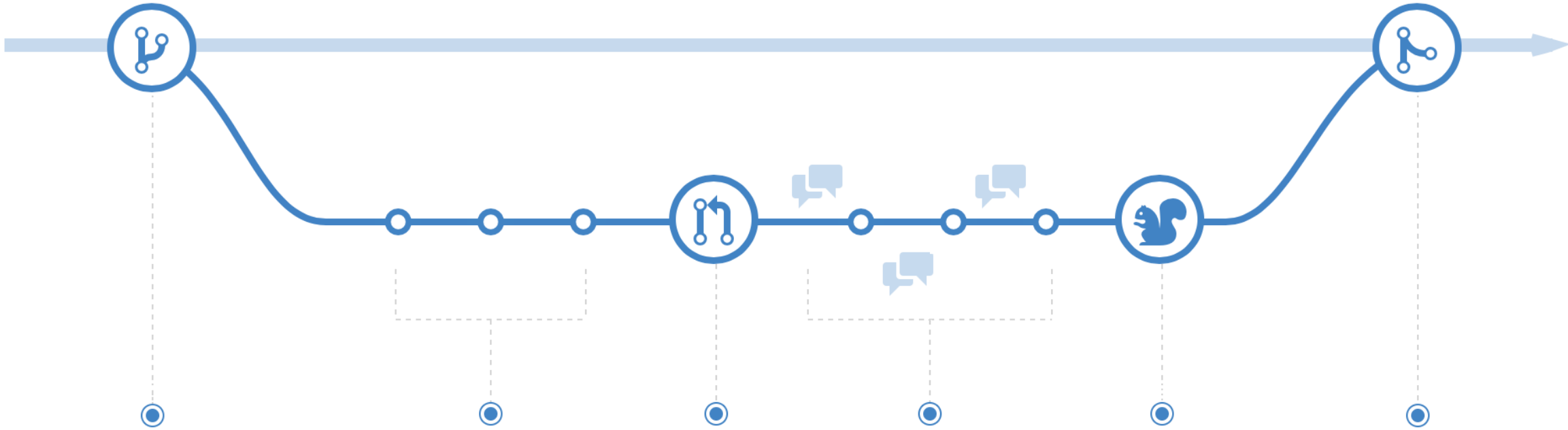
Demo

Trunk-Based Development

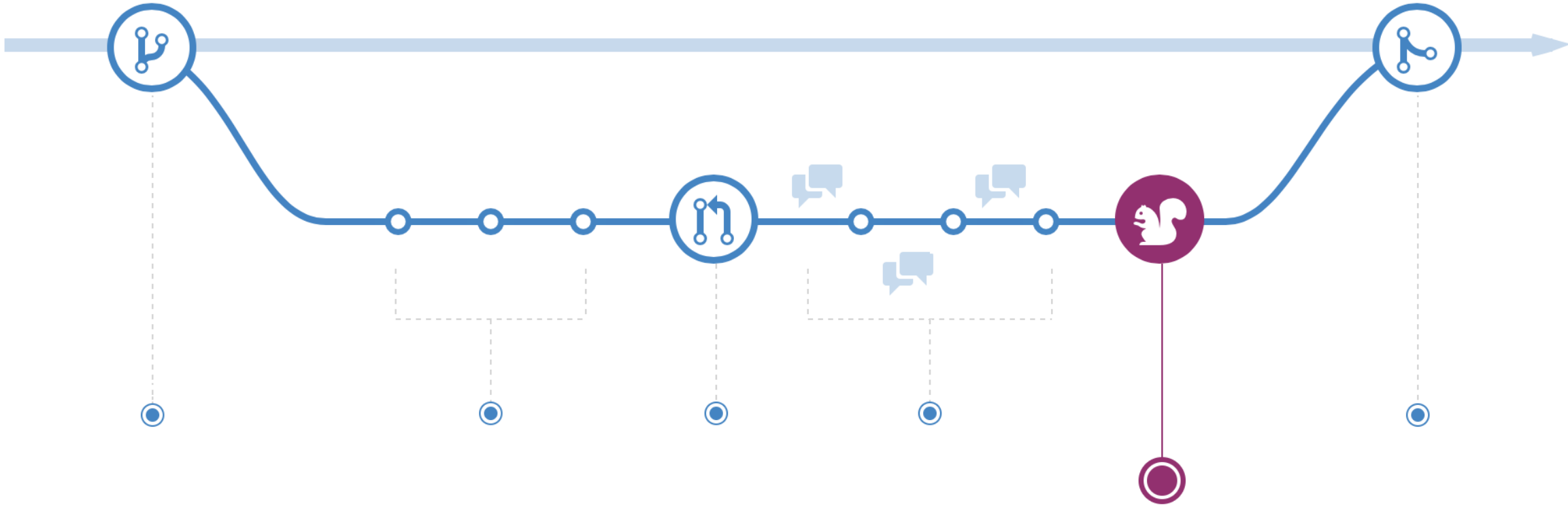
GitHub Flow

1. Lock the master branch
2. Merge master into the branch to deploy
3. Build and run test suite on the branch to deploy
4. Deploy the branch to canary; monitor for problems
5. Deploy the branch to production; monitor for problems
6. Merge the pull request into master; unlock the master branch

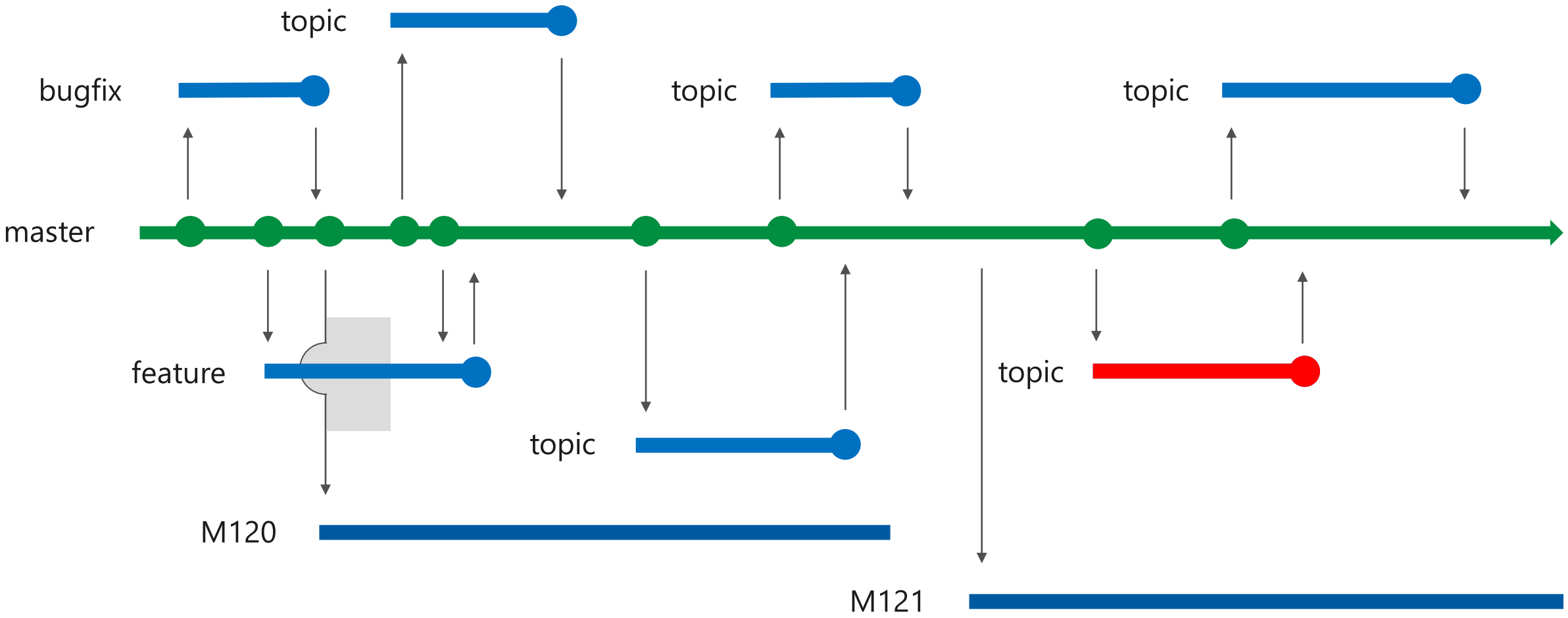
GitHub Flow



GitHub Flow



Release Flow Branching Structure



Demo

Release Flow

Gitflow

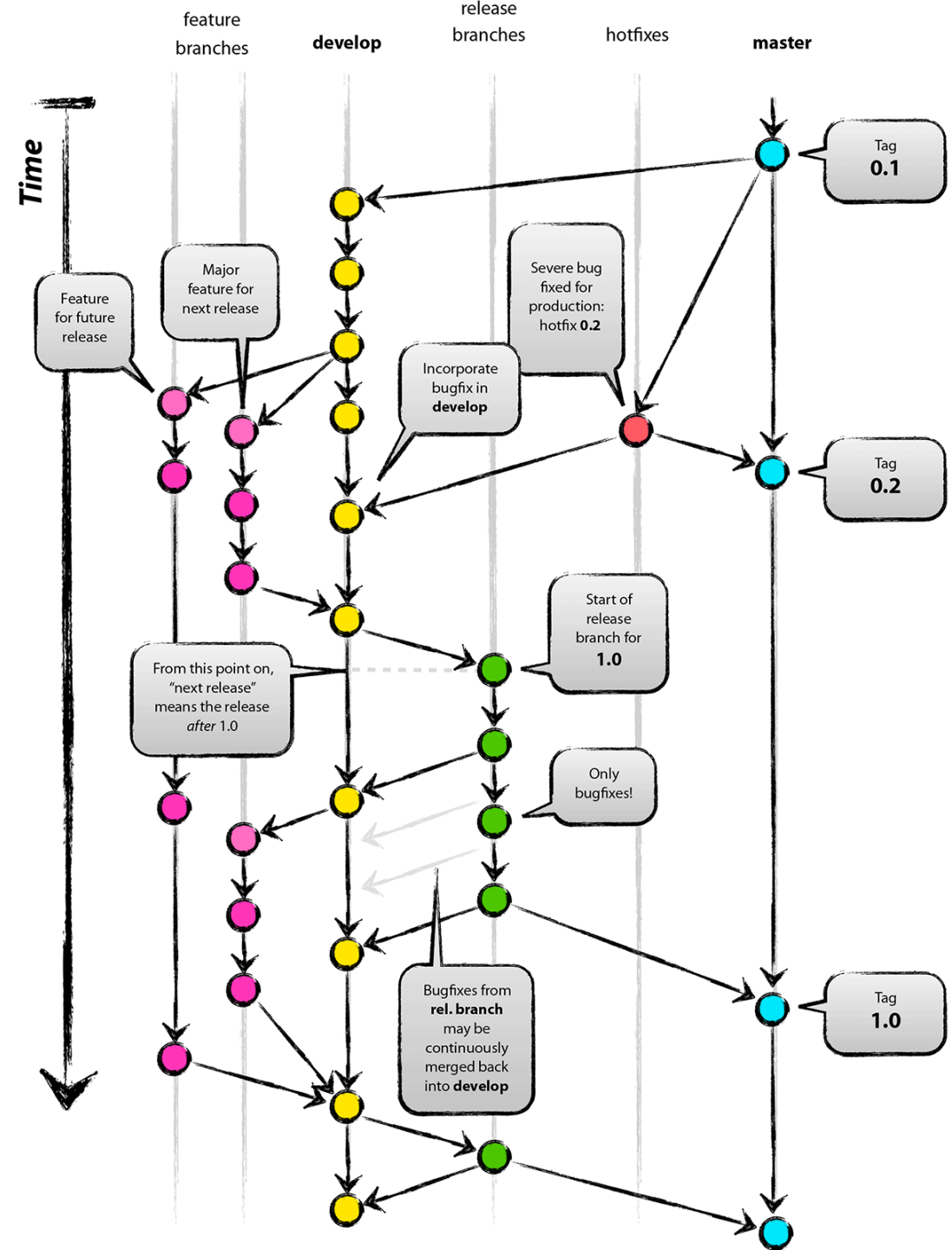
The Gitflow workflow defines a strict branching model designed around the project release.

More complicated than the Feature Workflow.

It assigns very specific roles to different branches and defines how and when they should interact.

Uses feature branches and individual branches for preparing, maintaining, and recording releases.

Gitflow Branching



Gitflow Branching

GitFlow Support

Work item linking

This experience makes it much easier to see the status of workitems as they're being worked on. From each workitem you can create a branch, which creates a link between the workitem and branch. When you create a pull request for that branch it's automatically linked to the workitem as well as the merge commit when you complete the pull request. This takes the hassle out of updating your workitems as they're being developed.

Branches Experience

The new experience is a dramatic improvement and makes it easier to find a branch, see status, and act on it.

Some of the new features are:

"My branches" pivot to see all the branches you created

"All branches" pivot with hierarchy. If you use GitFlow and organize your branches with a / (slash) we'll treat that as a folder so it's easy to work in a repo with lots of branches

Fast filtering on branch names, even partial matches if you use a – (dash) or / (slash) in your branch name

Pull requests status

Easy access to common actions like delete (with undo!), rename, set policies, lock, view history, create branch, etc.

A simple and readable model

Condensing History

- Squash merge
  - Keeps master history clean and easy to follow
  - Linear history through the use of squash merges
  - One commit for each merged branch
  - Can step through this history commit by commit

- Considerations
  - Condensed history of changes
  - Team decides when you should squash merge
  - Delete the source branch

Forking Workflow

The Forking Workflow is fundamentally different than the other workflows.

Instead of using a single server-side repository to act as the "central" codebase, it gives *every* developer a server-side repository.

This means that each contributor has not one, but two Git repositories: a private local one and a public server-side one.

Contributions can be integrated without everybody pushing to a single central repository.

Developers push to *their own* server-side repositories, and only the project maintainer can push to the official repository.

The maintainer accepts commits from any developer without the need write access to the official codebase.

Demo

GitFlow

What is Microsoft's Approach

Windows

Git-Flow branching is more or less designed for a project where only the most recent version of the product is in support and eligible for receiving bugfixes. Git-Flow requires that all releases ship from the 'master' branch (thereby requiring that the contents of the 'release' branch and/or 'hotfix' branch be merged into that 'master' branch prior to release).

Windows supports each OS release for up to ten years, releasing new OS versions two to three times a year. There are many different versions of the product to service at any given time (currently, there are *10* different versions of Windows in at least some level of support).

Ship out of the per-release 'release' branches (so our hotfixes ultimately merge back into 'develop' and 'release-n' rather than 'develop' and 'master').

Feature branches merge back into 'aggregation' branches that in turn merge into 'develop' some time later (after undergoing some amount of testing in conjunction with other new features) instead of a flatter model in which feature branches directly branch off 'develop.

What is Microsoft's Approach

Azure DevOps

Uses the traditional master is "next" model.

Effectively all developers work in master (more specifically a developer managed topic branch off of master, which is designed to merge back into master asap), and master is the next release.

Release branches are created when isolation becomes absolutely necessary.

Lesson 3: Best Practices

- After completing this lesson, you will be able to:
  - Describes simplified branching structure
  - Describe how to embrace DevOps
  - Describe the Good and Bad Habits
  - Describe the use of Pull Requests

Simplified Branching Structure

Code close to master
Small, simple changes
Fewer merge conflicts
Easy to code review
Encourages pull requests
Simpler to ship; faster velocity

Overview

- How Git Manages Branches
- What do we mean by Git Workflow
- Simple to complex workflows
- Guidelines for branching

Best Practice – Embracing DevOps

- Keep it simple and expand complexity as needed
- Strive for high-quality
- Foster a *DevOps culture*
- Promote collaboration flow and increased productivity
- Enable teams to spend more time developing and less time managing code
- Organize your code into shippable units

Best Practice – Good Habits

- Use a consistent naming strategy for your branches
- Build with every check in
- Create a CI/CD pipeline using gated checkins and automated testing
- Use consistent naming conventions for branches
  - features/username/description for work performed by an individual - example, *features/sandra/sdk-java*
  - bugfix/username/bugid for work done specific to an engineering bug - example, *bugfix/takashi/707*
  - releases/version for planned releases - example, *releases/V1.00*
- Frequently reverse integrate (RI) and merge into your main branch
- Encourage consistent code reviews - garbage in, garbage out

Best Practice – Bad Habits

- Getting branch crazy!
  - merging changes comes with complexity and a cost
  - there's no need to have a separate branch per environment
- Cherry-picking to get your code to production
- Attempting to solve **people** or **process** problems with tools

Git Credential Manager

Keep your branch strategy simple

Keep your branch strategy simple by building your strategy from these three concepts:

- Use feature branches for all new features and bug fixes.
- Merge feature branches into the master branch using pull requests.
- Keep a high quality, up-to-date master branch.
- A strategy that extends these concepts and avoids contradictions will result in a version control workflow for your team that is consistent and easy to follow.

Branches should be as short-lived as possible.

- The longer a branch remains the more likely you're going to have conflicts and bugs introduced by attempting to integrate later.
- CI and the pull request system negates need for multiple branches beyond master.
- When supporting multiple releases Release branches are fine.

Use feature flags to manage long-running branches

- Long-lived feature branches present problems when you need to build code on top of the branch before the work in the branch is finished.
- Merge unfinished features into the master branch so others can build off their work but keep them hidden from your users and testers behind feature flags.
- Enable the flag in development to use the feature without the changes affecting anyone else.
- Once the feature is finished, you can remove the flags or use them to roll out the feature to select users and testers.

Review and merge code with pull requests

- The review that takes place in a pull request is critical for improving code quality.
- Only merge branches through pull requests that pass your review process.
- Avoid merging branches to the master branch without a pull request.

Review and merge code with pull requests

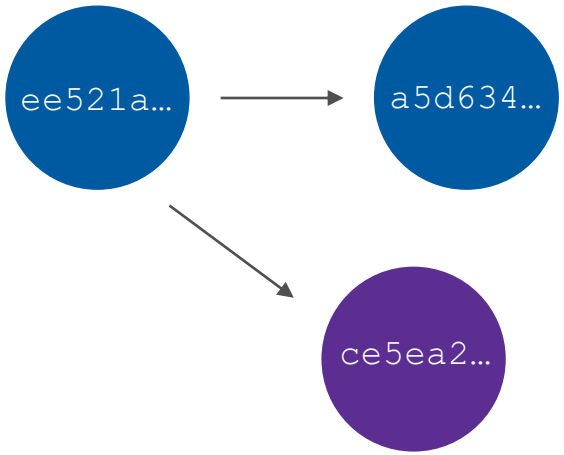
- Reviews in pull requests take time to complete
- Team should agree on what's expected from pull request creators and reviewers.
- Distribute reviewer responsibilities to share ideas across your team and spread out knowledge of your codebase.

Lesson 4: Avoiding Push Conflicts

- After completing this lesson, you will be able to:
  - Know the best approach to handling conflicts when you share by pushing to a shared remote repository.

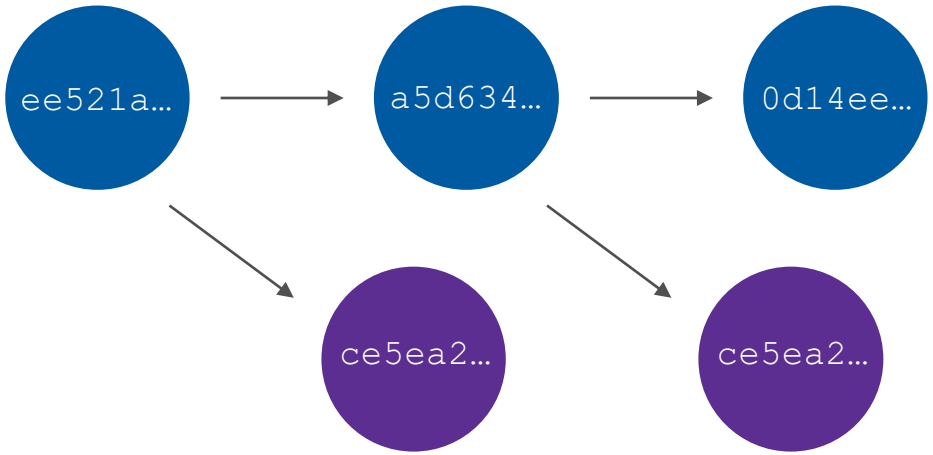
Code Integration

Large teams have contention pushing to the server



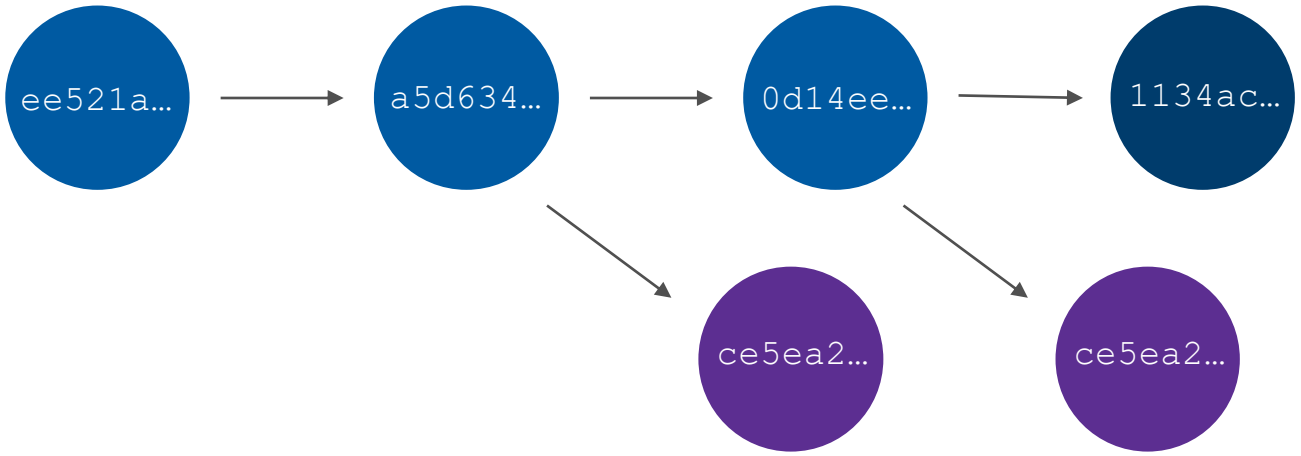
Code Integration

Large teams have contention pushing to the server



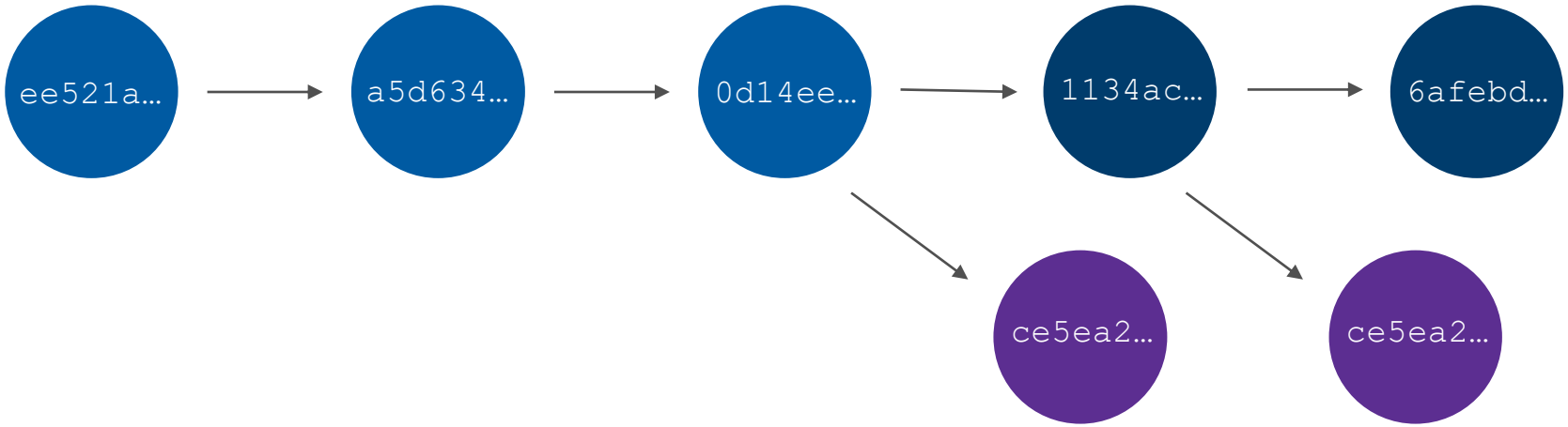
Code Integration

Large teams have contention pushing to the server



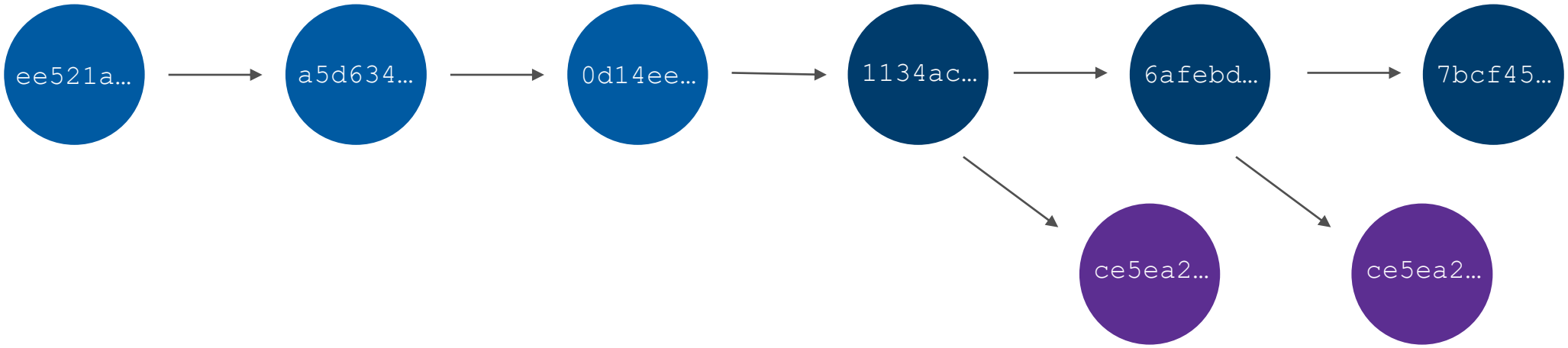
Code Integration

Large teams have contention pushing to the server



Code Integration

Large teams have contention pushing to the server

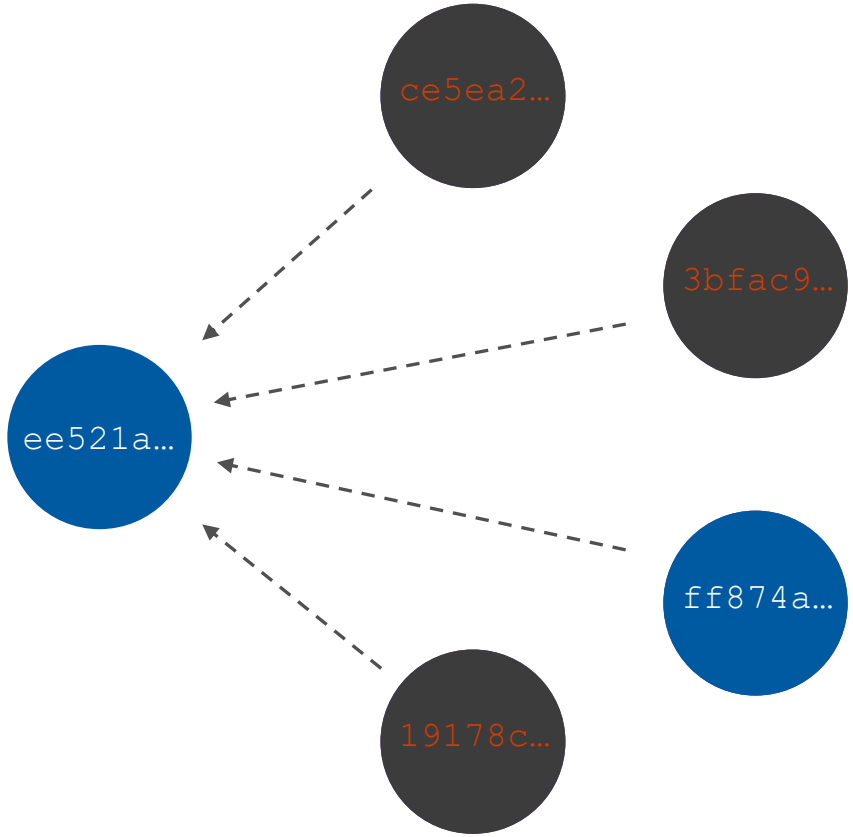


Solution: Pull Requests

- Merge occurs on the server
  - No need to download the other changes locally
  - Pull requests are updated whenever the target changes
- Only need to merge locally when there are conflicts
  - Merge and push changes to your pull request
- Drastically reduces cycle time
- Enforced via branch policies

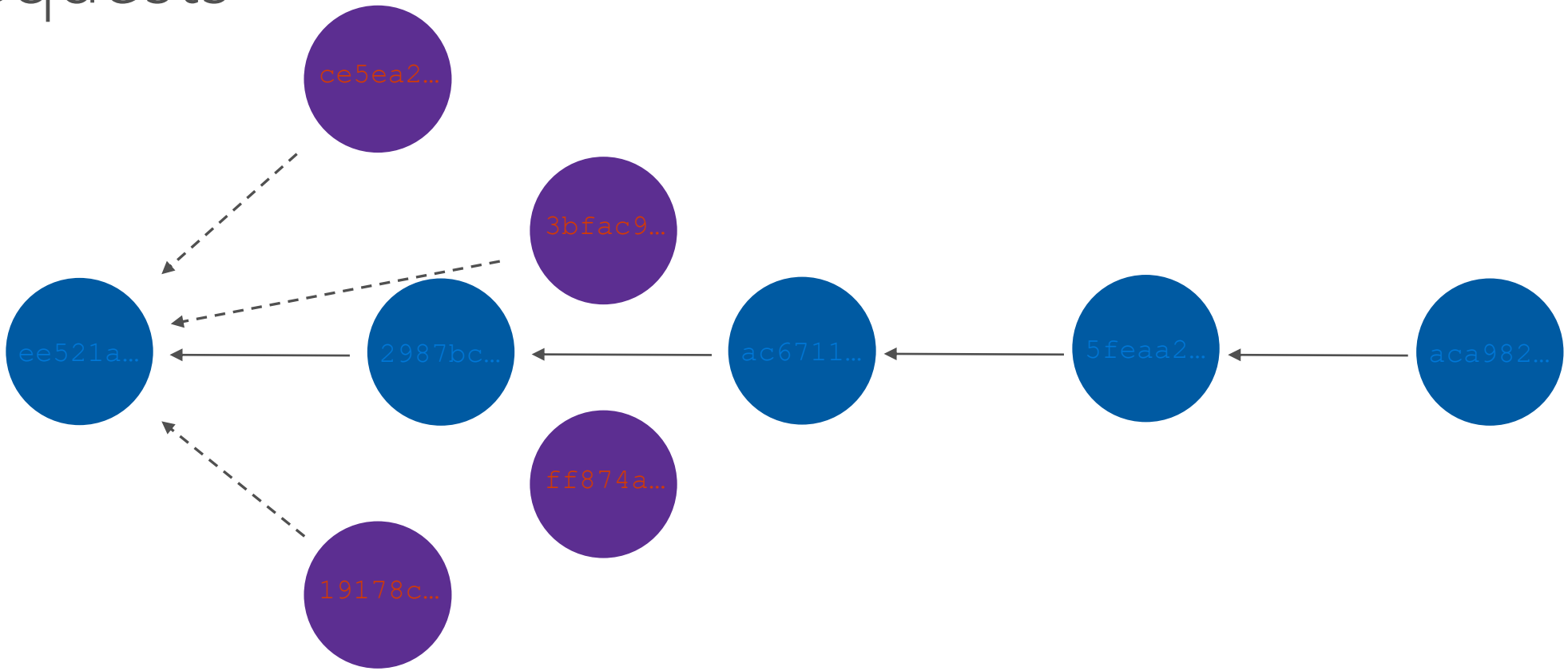
Code Integration

Large teams have contention pushing to the server



Code Integration with Pull Request

Azure DevOps uses a “merge queue” for Pull Requests



Successful pull requests

- Two reviewers is an optimal number based on research.
- If your team already has a code review process, bring pull requests into what you're already doing.
- Take care assigning the same reviewer(s) to a large number of pull requests. Pull requests work better when reviewer responsibilities are shared across the team.
- Provide enough detail in the description to quickly bring reviewers up to speed with your changes.
- Include a build or linked version of your changes running in a staged environment with your pull request so others can easily test the changes.

Keep a high quality, up-to-date master branch

Code in your master branch should

1. pass tests
2. build cleanly
3. always be up to date

Your master branch needs these qualities so that feature branches created by your team start from a known good version of code.

Keep a high quality, up-to-date master branch

Set up a branch policy for your master branch that:

1. Requires a pull request to merge code. This prevents direct pushes to the master branch and ensures discussion of proposed changes.
2. Automatically adds reviewers when a pull request is created. The added team members review the code and comment on the changes in the pull request.
3. Requires a successful build to complete a pull request. Code merged into the master branch should build cleanly.

The build definition for your pull requests should be quick to complete, so it doesn't interfere with the review process.

Knowledge Check

- Git workflow is like TFVC branching strategy.
- Microsoft's Developer Division uses which workflow.
- Git manages which branch you are working on by modifying which file?
- A best practice is to use a consistent naming strategy for branches.
- People and process problems can be solved with tools.
- Gitflow is a simple branching strategy.

Knowledge Check

- Q: Git workflow is like TFVC branching strategy.
  - A: True
- Q: Microsoft's Developer Division uses which workflow.
  - A: Release
- Q: Git manages which branch you are working on by modifying which file?
  - A: HEAD
- Q: A best practice is to use a consistent naming strategy for branches.
  - A: TRUE
  - Q: People and process problems can be solved with tools.
    - A: FALSE
  - Q: Gitflow is a simple branching strategy.
    - A: FALSE

Module Summary

- Take Aways:
  - Microsoft Today and Open Source Software
  - DevOps and where Git fits
  - There are many workflows for you team. Remember the good and bad habits and choose the one that works for your team
  - Apply the best practices to your workflow
  - Leverage Pull Request to avoid push conflicts