

# The Flow of Change

Branching and merging in the face of agile development, extreme programming, team collaboration, and parallel releases.

Tony Smith  
European Technical Services Manager  
Perforce Software UK

# What we'll cover

- ◆ The ideal world vs. the real world
- ◆ Branches and modules
- ◆ The “tofu scale”
- ◆ The “baseline protocol”
- ◆ The “golden rule of collaboration”
- ◆ The myth of merging
- ◆ Why we don't drive through hedges

# Ideas you'll come away with

- ◆ How to plan for branching and merging
- ◆ How to simplify a complicated branching and merging scheme

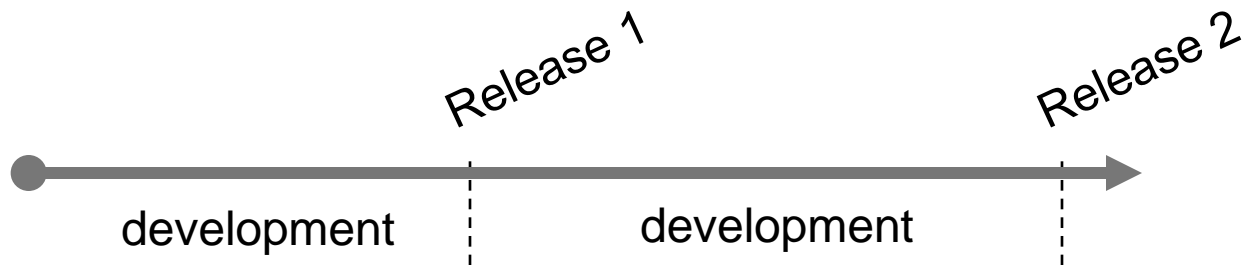
# Software development in the ideal world

- ◆ There are no bugs
- ◆ We have all the time in the world
- ◆ Schedules never slip
- ◆ The first release is perfect
- ◆ Customers always upgrade



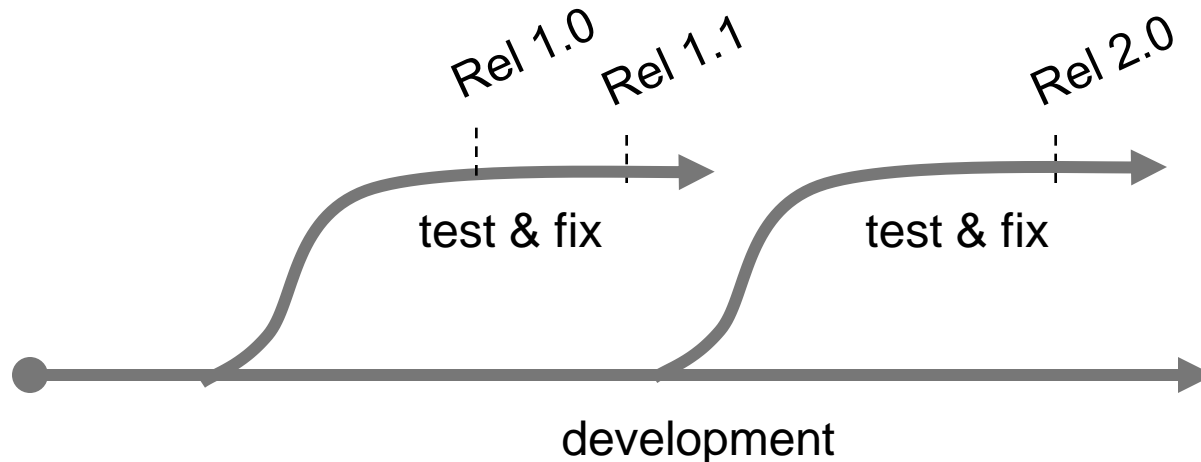
# In the ideal world we make one release

- ◆ In the real world one release is never enough
- ◆ What we can do about it: make periodic releases



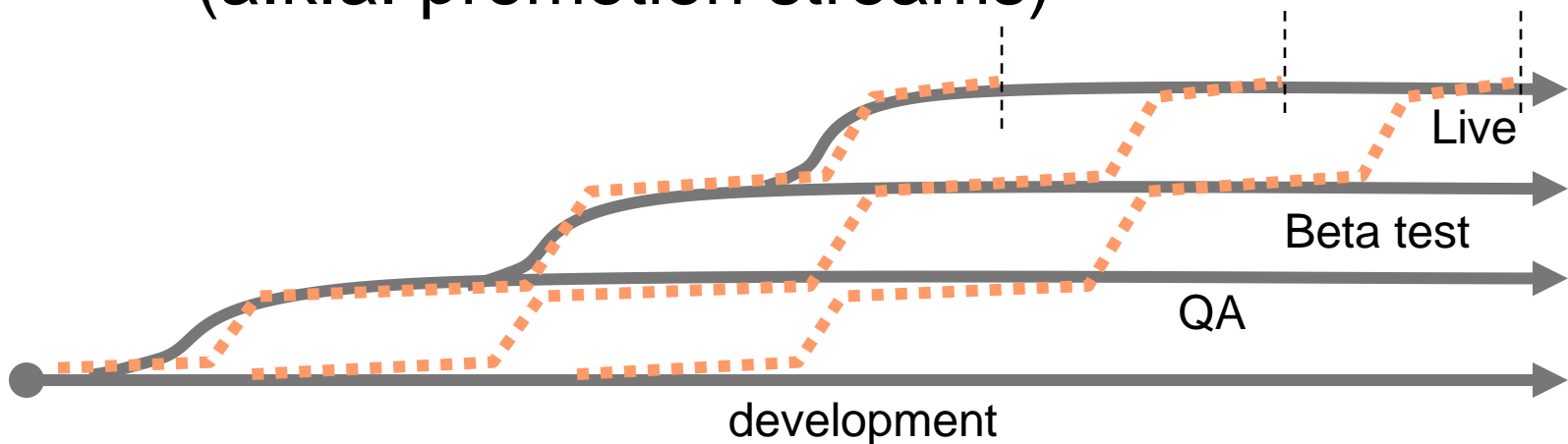
# In the ideal world there are no bugs

- ◆ In the real world we need time to stabilize releases
- ◆ What we do about it: branch for each release



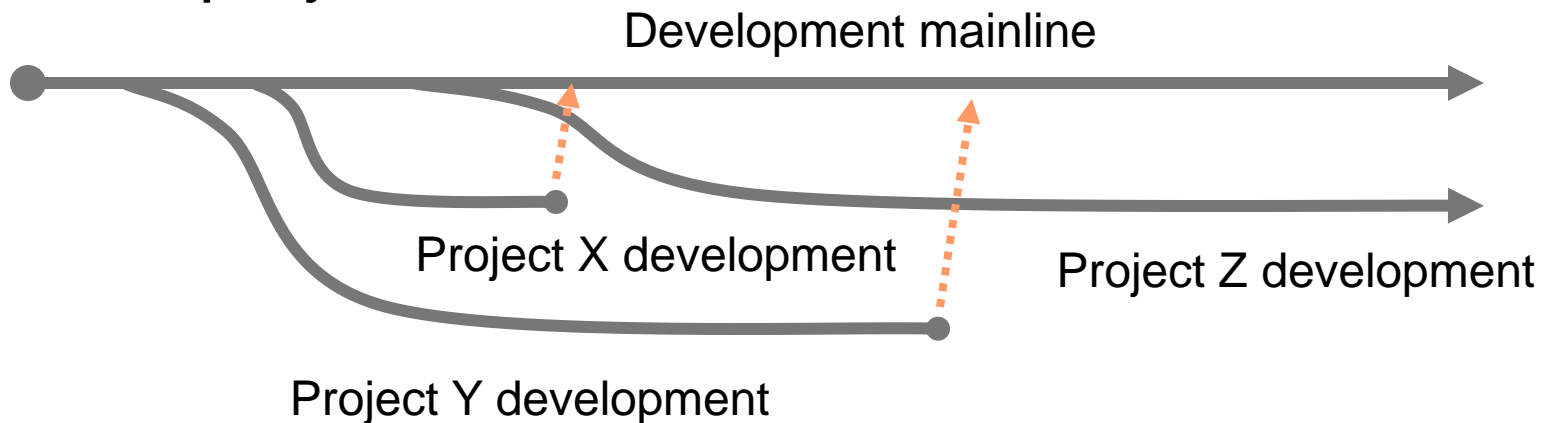
## In the ideal world we have all the time we need

- ◆ In the real world our release cycles can be *very* short
- ◆ What we do about it: use staging branches (a.k.a. promotion streams)

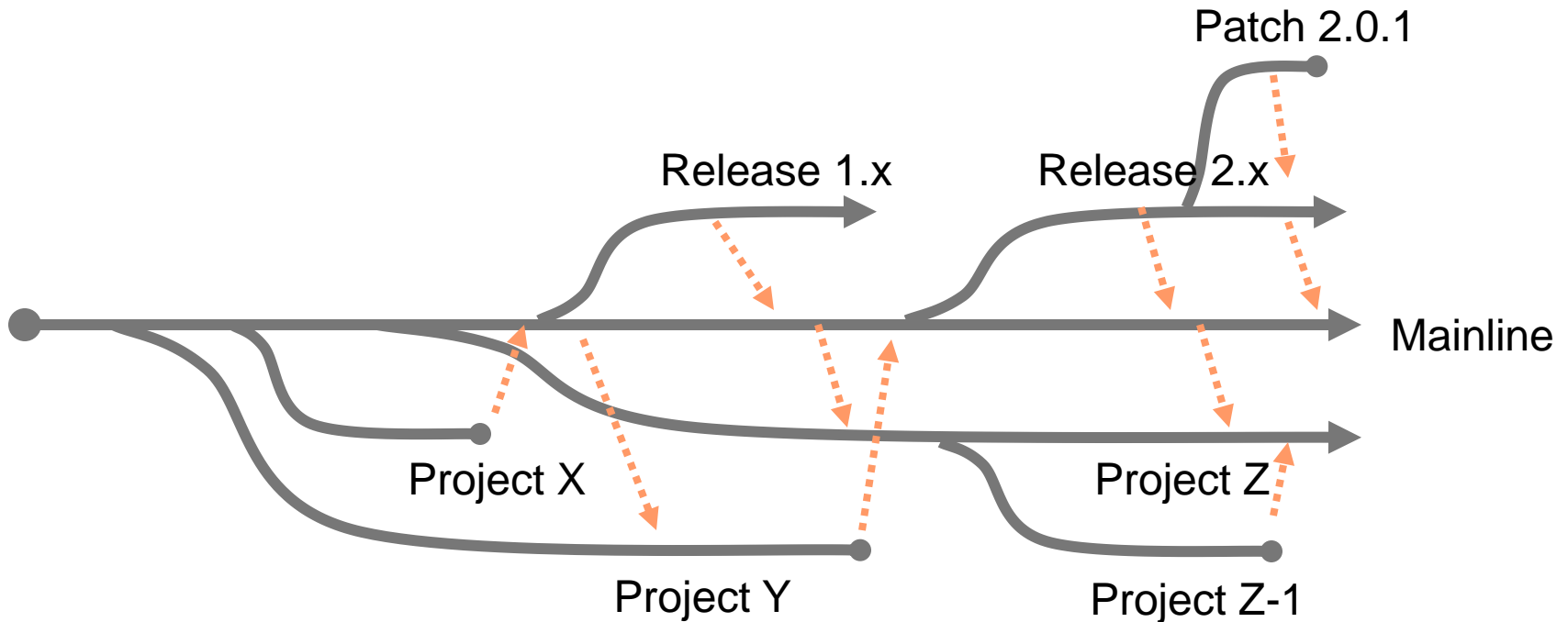


# In the ideal world all development finishes on time

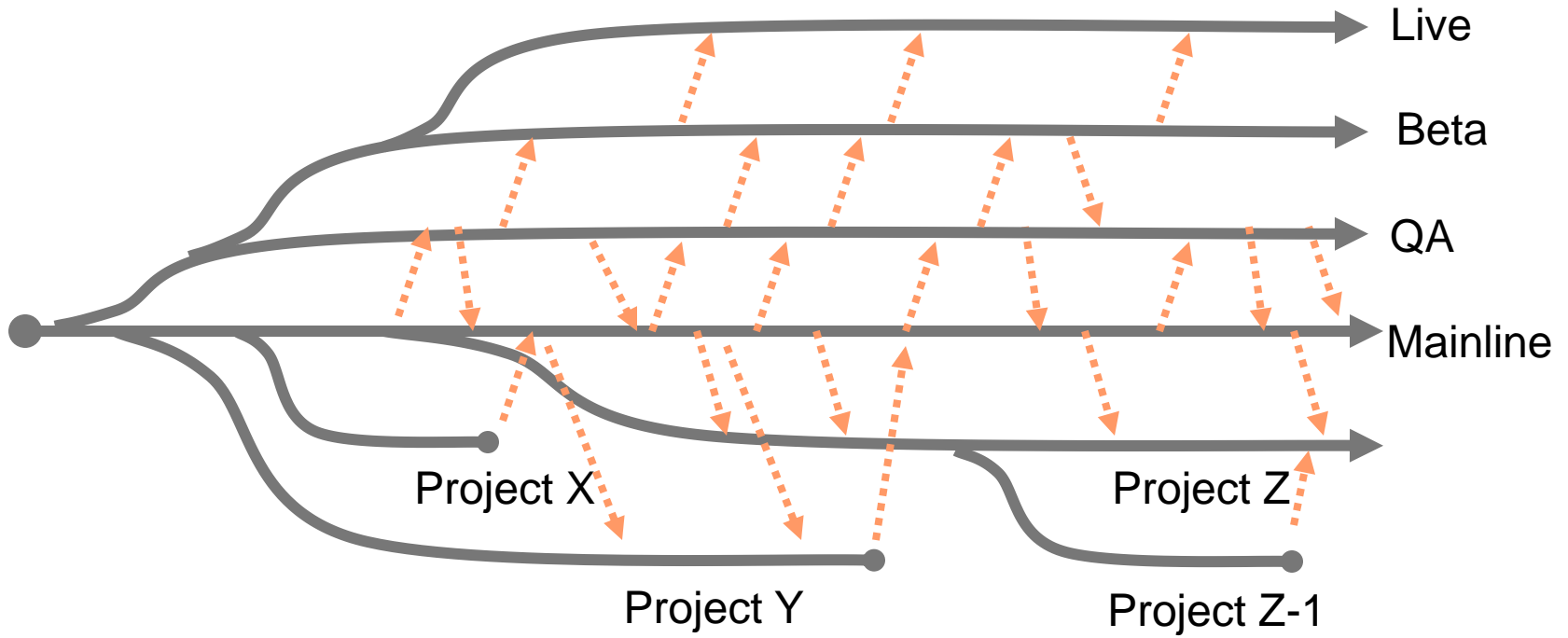
- ◆ In the real world there are unforeseen delays
- ◆ What we can do about it: decouple development projects and branch for each project



# The mainline model and its flow of change



(same thing with promotion streams)

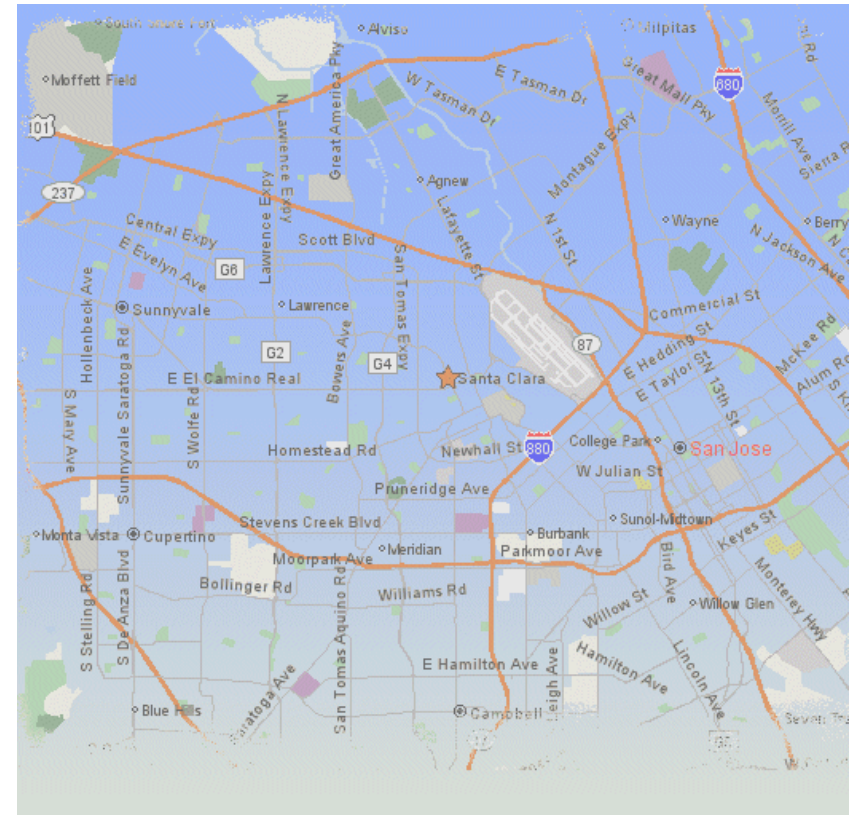


# Chaos?



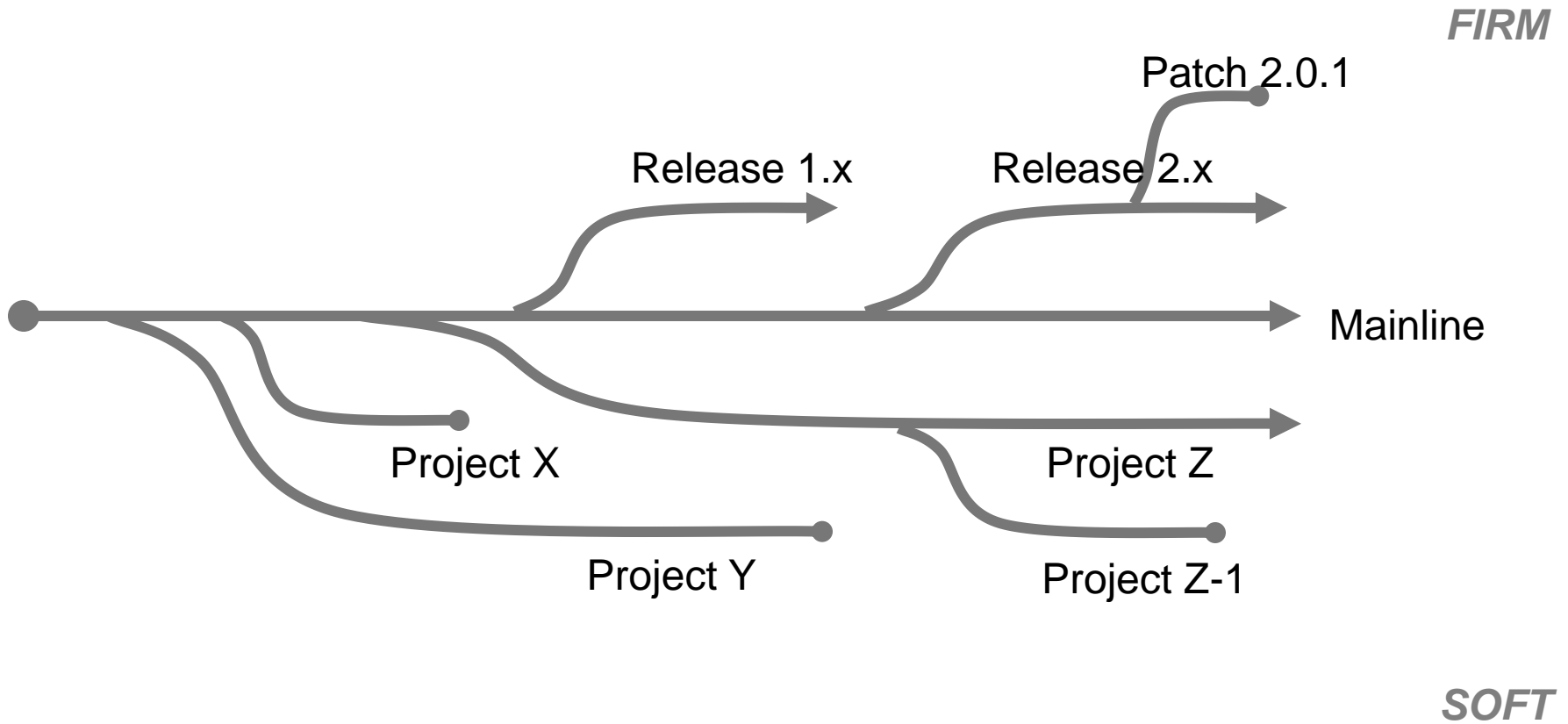
# Not necessarily...

◆ Maps, protocols, and etiquette make driving manageable

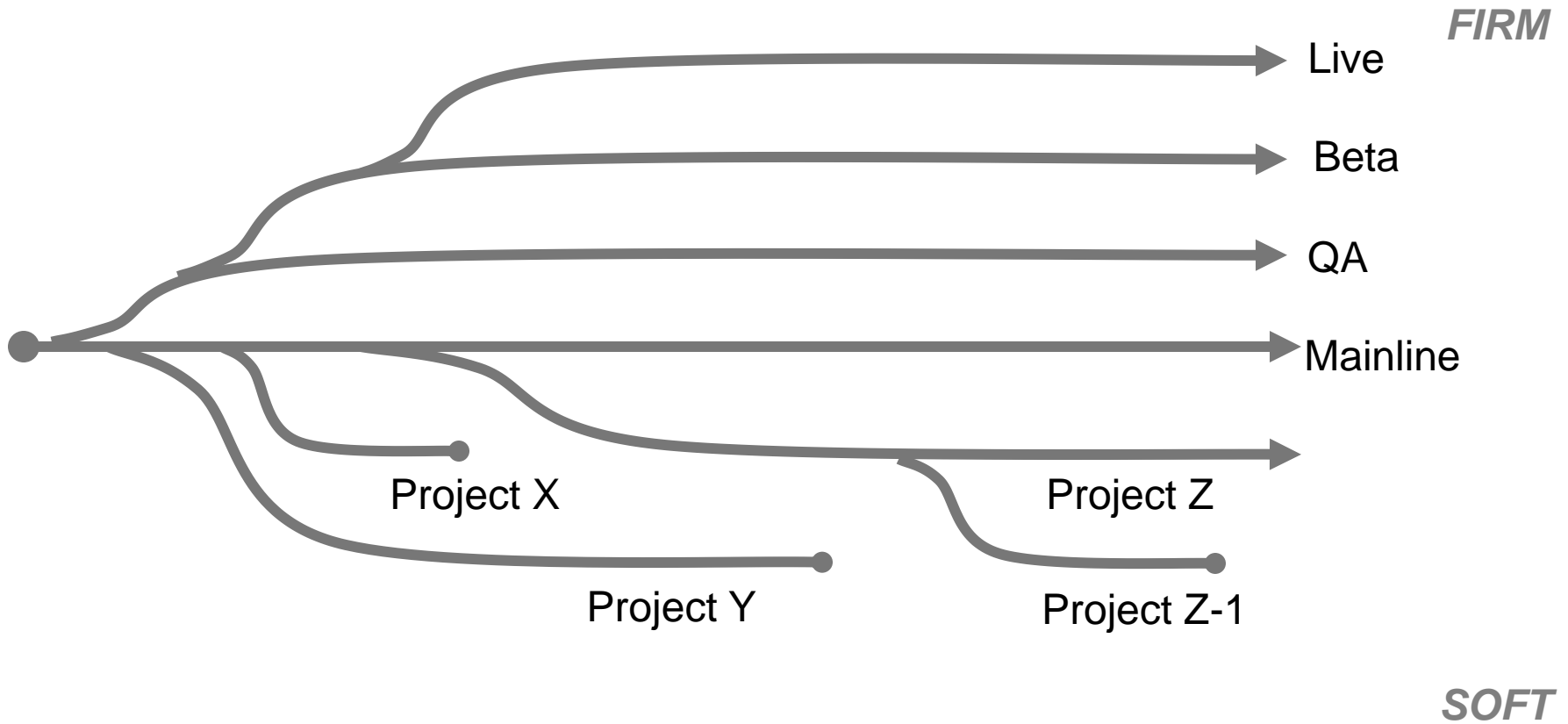


◆ Maps, protocols, and etiquette can make branches manageable

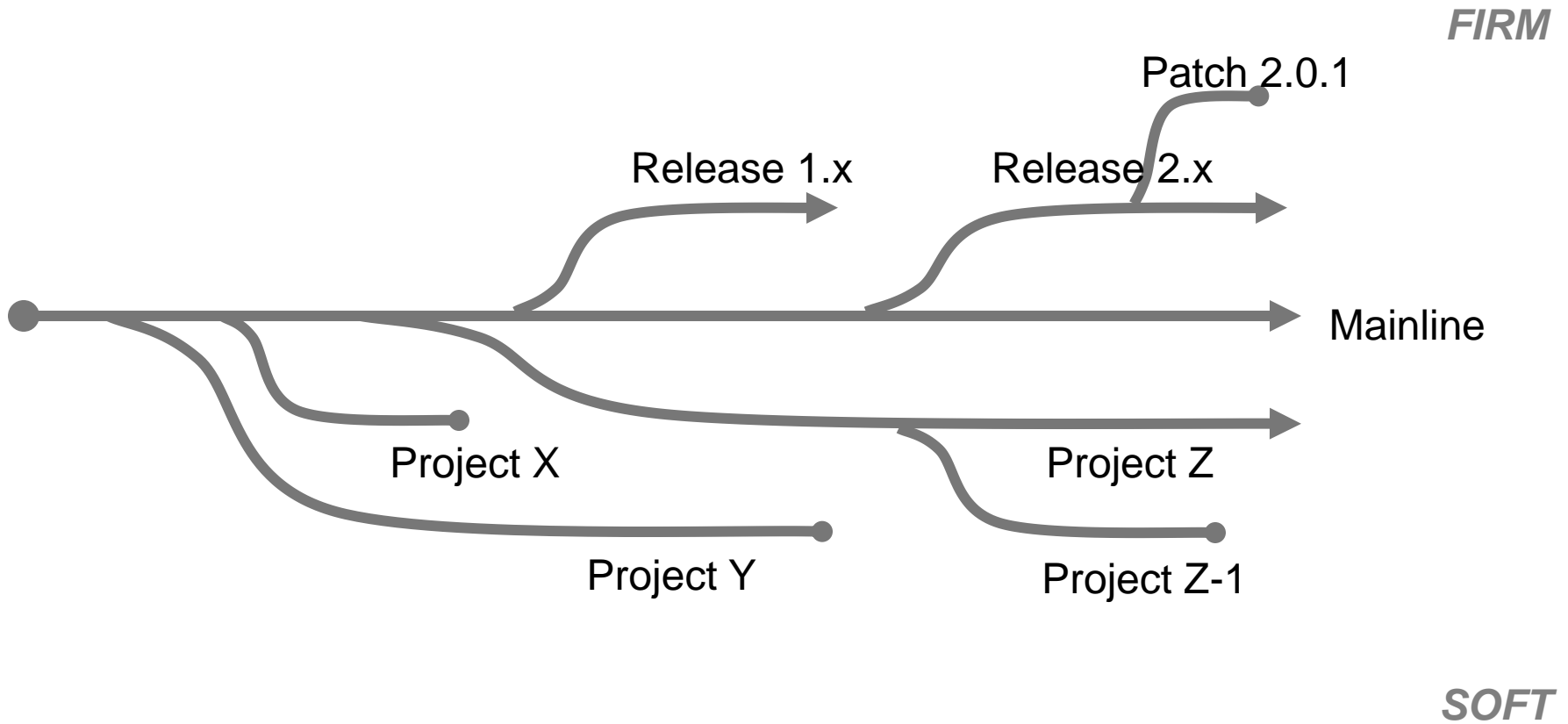
# The “tofu scale”



# The "tofu scale"

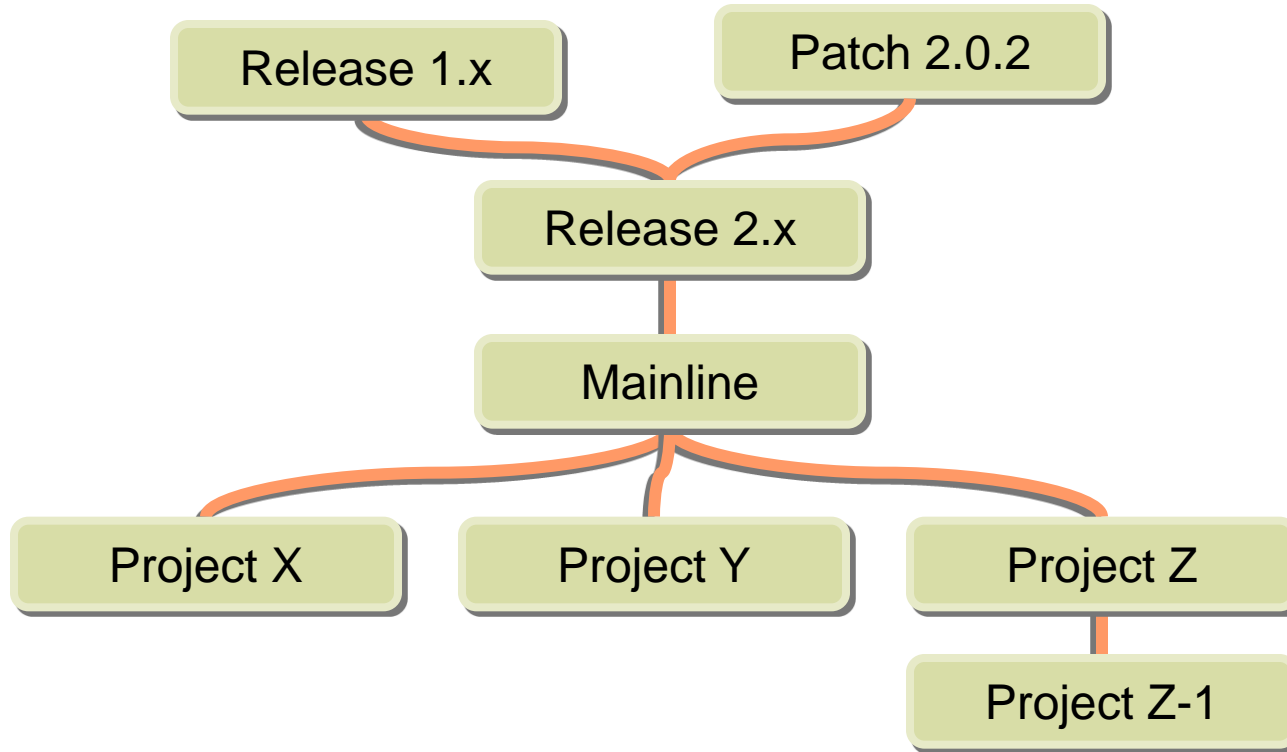


# From timeline...



# ...to baseline map

*FIRM*



*SOFT*

# The “baseline protocol”

## ✦ Change flows between a branch and its baseline

<b>When a branch is:</b>	<b>Change flows from branch to baseline:</b>	<b>Change flows from baseline to branch:</b>
Firmer than its baseline	Continually	“Never”
Softer than its baseline	At points of completion	Continually

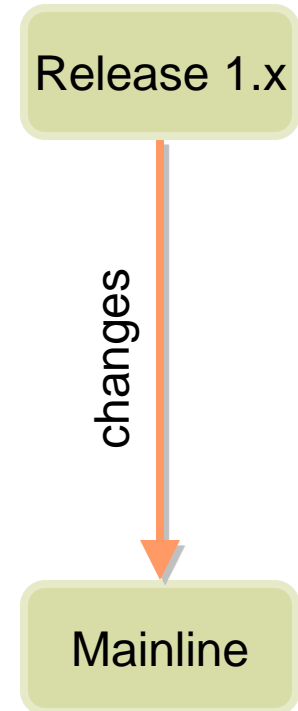
# Why we don't drive through hedges

- ◆ Just as driving through hedges makes the roads confusing and dangerous places to drive...
- ◆ Merging changes between arbitrary branches makes the repository a confusing and dangerous place to check in your code



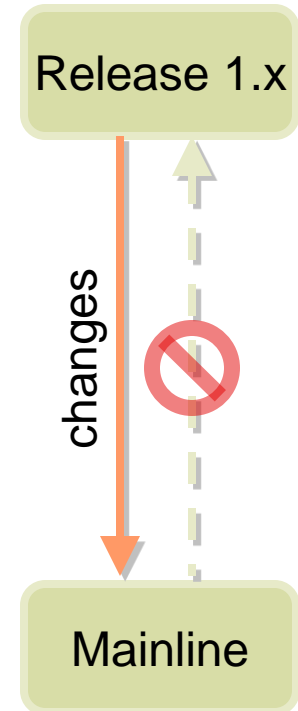
# Example: release branch

- ◆ The flow of change from a release branch to its baseline is continual
  - ◆ Every improvement to a release branch is an improvement to the baseline



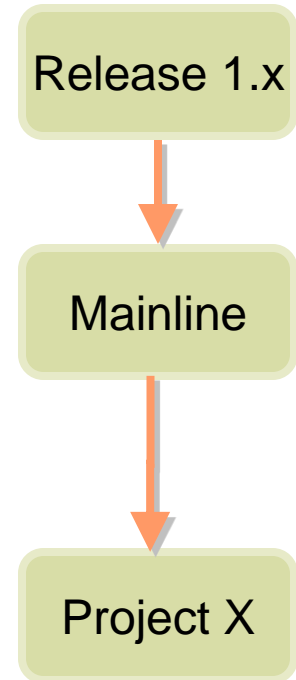
# Example: release branch

- ◆ No change flows *to* a release branch *from* its baseline
  - ◆ Release branch gets more and more stable
  - ◆ Baseline changes don't put release branch at risk



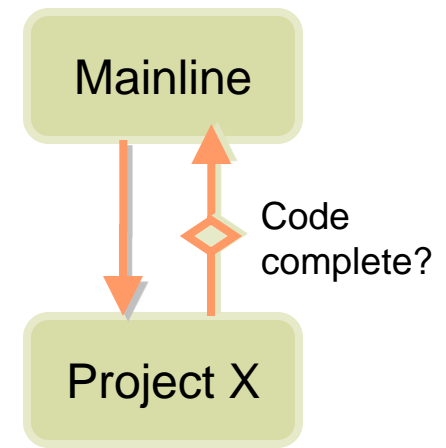
# Example: development branch

- ◆ Change flows continually to a development branch from its baseline
  - ◆ Baseline changes stabilize the development branch
  - ◆ Development branches always have latest bug fixes and patches



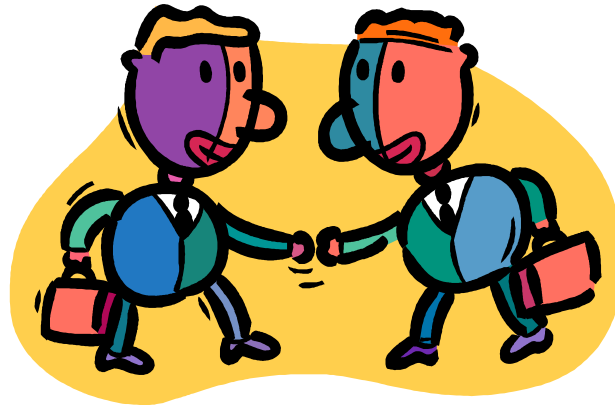
# Example: development branch

- ◆ Gated flow from development branch to its baseline:
  - ◆ *Open* at points of completion in development branch
  - ◆ *Closed* when development is incomplete or dev build is broken



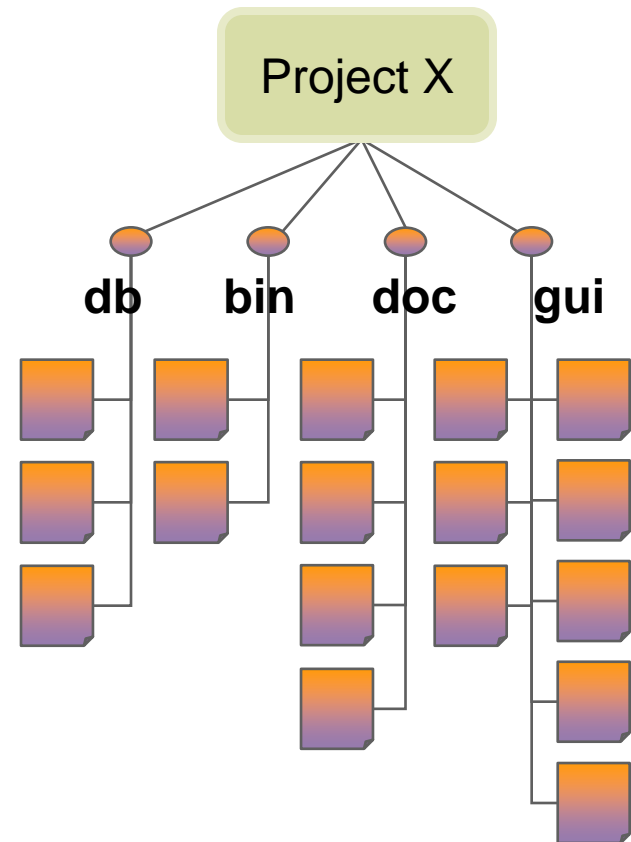
## The Golden Rule of Collaboration

Always accept stabilizing changes.  
Never impose destabilizing changes.



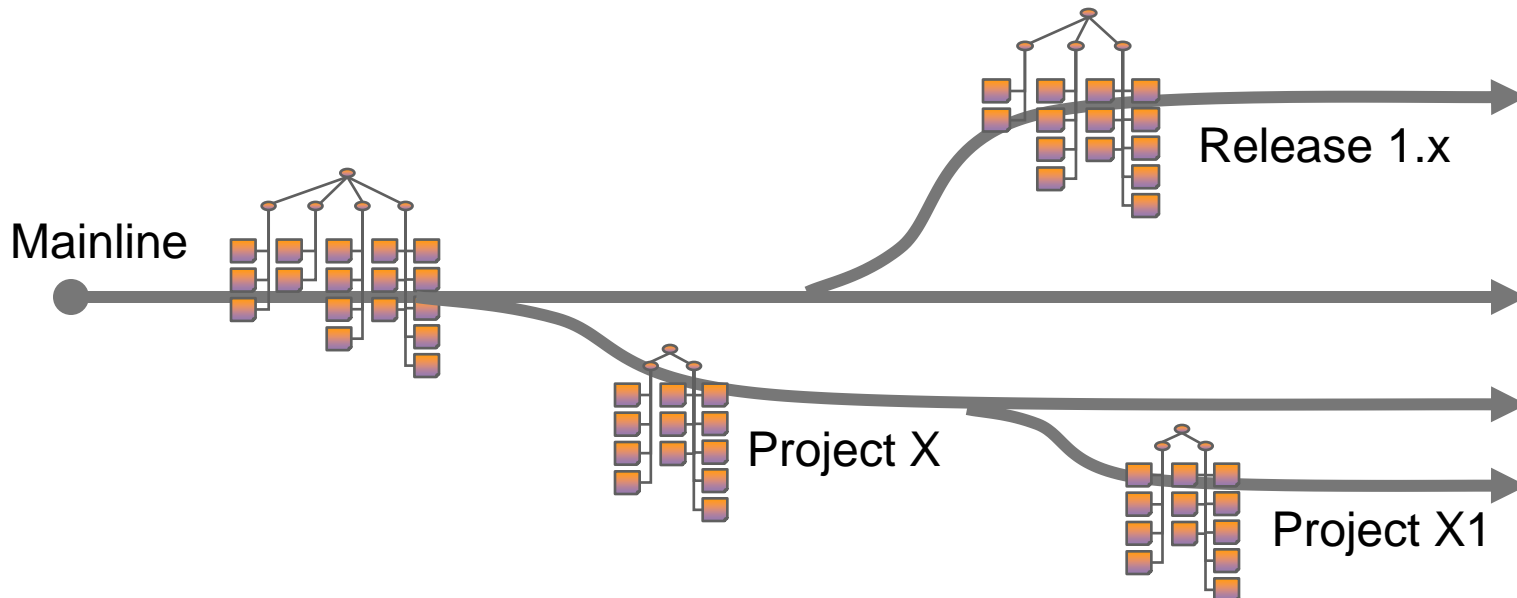
# Modules and Branches

- ◆ A branch is a collection of modules
- ◆ A module is a tree of files
- ◆ We need modules on disk to work on files



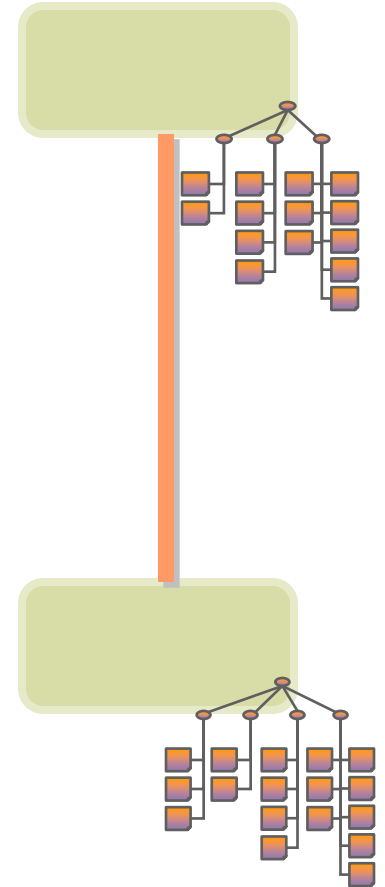
# Modules and branches

- ◆ When you branch, you branch modules
- ◆ Not all modules are needed in every branch



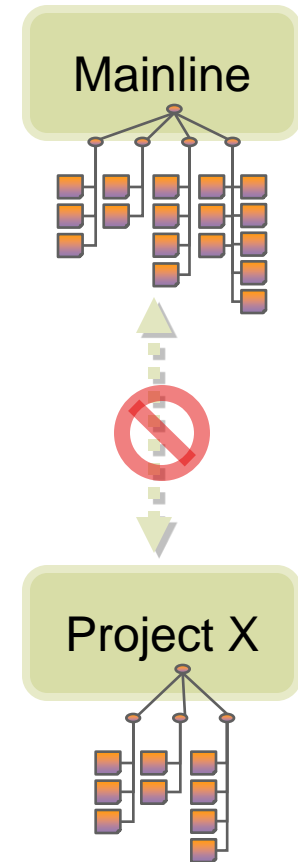
# Modules and branches

- ◆ Change flows between branches by merging or copying between corresponding modules



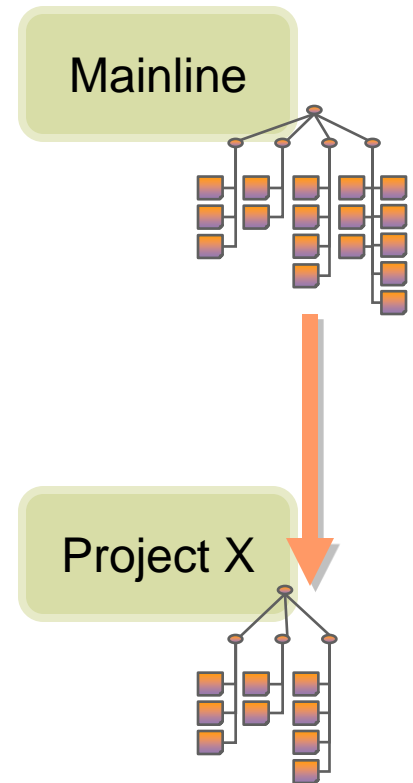
# “Private” modules

- ◆ Same structure and purpose as their baseline counterparts
- ◆ No flow of change between private modules and their baseline counterparts



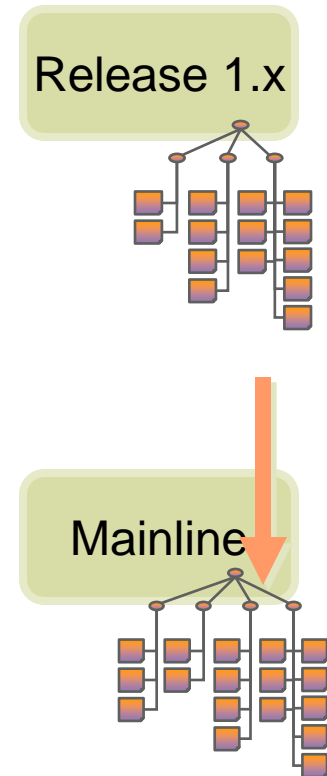
# “Virtual” modules

- ◆ Virtual modules support debugging, testing, building
- ◆ Virtual modules inherit all baseline changes – they mirror the baseline
- ◆ Inactive in branch (i.e., not changed by work in branch)



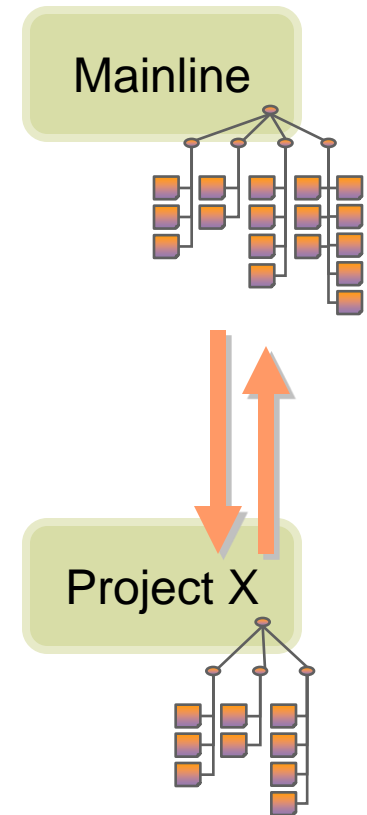
# “Active” modules

- ✦ Active modules are the modules we’re working on in a branch
- ✦ Active modules can change in both branch and baseline
- ✦ In release branches, active modules’ changes always flow to the baseline



# “Active” modules in development branches

- ◆ Change flows both ways
- ◆ Active modules – but *only* the active modules – eventually need merging

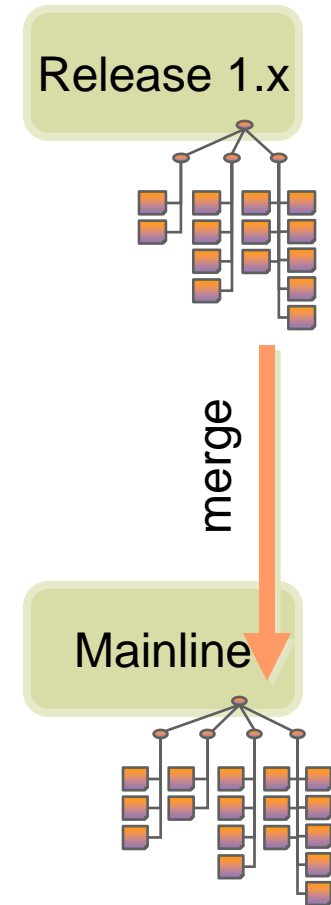


# The myth of merging

- ◆ Is merging dangerous?
  - ◆ Is coding dangerous?
  - ◆ Merging is as “dangerous” as coding
- ◆ Merging can be as safe as coding if done in the right branch

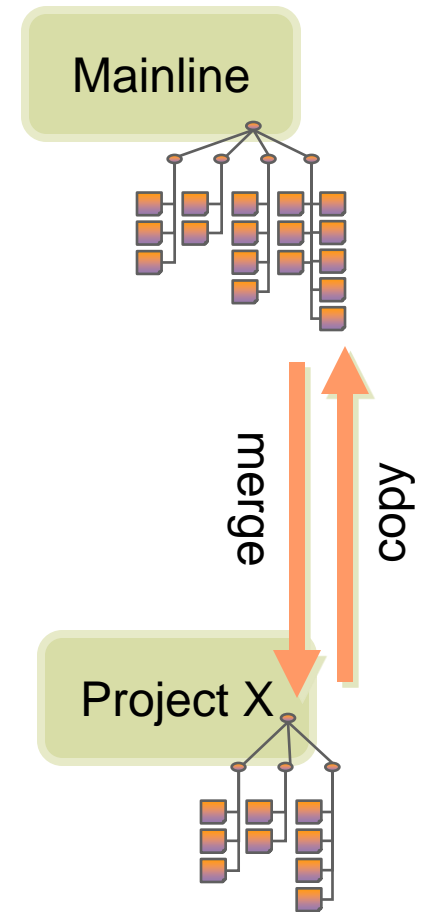
# “Merge down, copy up” protocol

- ◆ *Merge* from firm branch to soft branch
- ◆ *Copy* from soft branch to firm branch
- ◆ Softer branch can accommodate merging better than firm branch can



# “Delivering” completed development

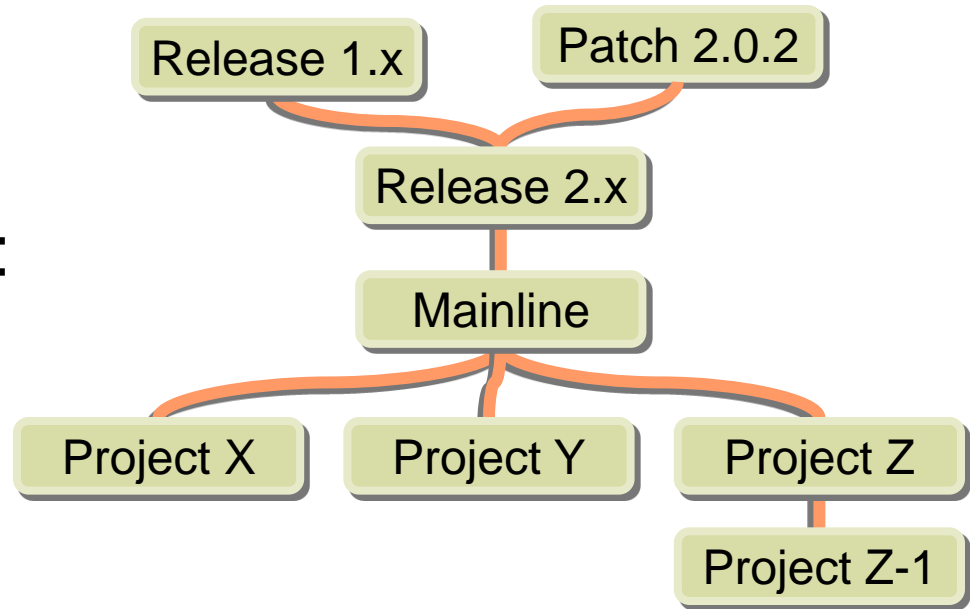
- ◆ From a development branch to its baseline:
  - ◆ *Merge* from baseline to branch first, then...
  - ◆ Test (compile, proofread, etc.)
  - ◆ *Copy* from branch to baseline
- ◆ Etiquette:
  - ◆ Yield to delivery of completed development



# Not chaos at all

◆ Thanks to maps, protocol, and etiquette, we know:

- ◆ The state of each branch
- ◆ How and when to merge changes



◆ Even the largest system is merely an assemblage of regular objects in simple relationships

## In a nutshell...

- ◆ Branches: mainline, release, development
- ◆ Tofu scale: firm above, soft below
- ◆ Change flows between branch and baseline
- ◆ Merge down, copy up
  - ◆ Only “active” modules need merging
- ◆ Always accept stabilizing change, never impose destabilizing change
- ◆ Yield to delivery of completed development
- ◆ Don't drive through hedges

# Read the book!

- ◆ Practical Perforce (O'Reilly)
  - ◆ by Laura Wingerd VP Product Technology at Perforce

Questions?