Git

Commit

- Algún estado del sistema
- Queremos conservarlo en el futuro
- Queremos compartirlo con el resto del equipo

Commit

Cada commit contiene:

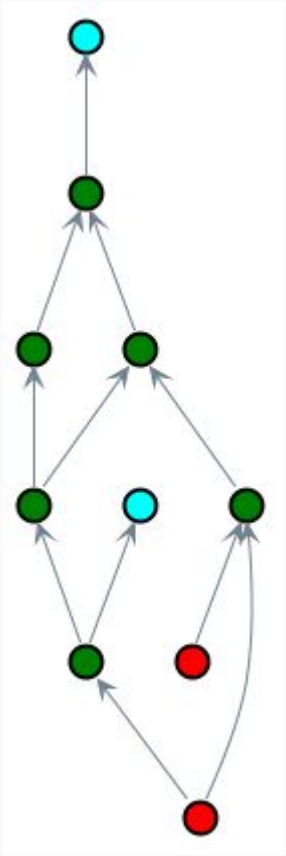
- Archivos
- Fecha
- Mensaje de commit
- Ancestros

Historia

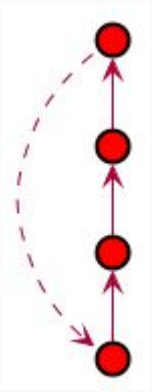
Cada commit tiene ancestros. Su contenido depende del contenido de sus ancestros.

Unico requisito: Sin ciclos

Historia Lineal



Historia Compleja



Historia Imposible

Historia

Git no impone una organización de la historia

El equipo debe decidir cómo organizarse

Combinar ideas base y flujos pre-armados
Involucra otras herramientas además del repositorio

Historia

Dicho esto, existen:

- Convenciones establecidas
- Herramientas preexistentes
  - Muchas son distribuidas con git

Usaremos: **gitlab**

Mensajes de Commit

```
Contexto: Resumen del cambio en este commit
                (línea en blanco)
Este texto explica el contexto del cambio,
incluyendo cosas como:
- Razones que no resultan obvias del
  resumen
- Efectos de los cambios que no resultan
  obvios al ver el código

Metadata que leen otros sistemas:
- Issues resueltos
- Firmas digitales
```

Cosas que no deberían pasar

https://xkcd.com/1296/

Algunos comandos útiles

- `git log <archivo>`
  Historial de cambios de un archivo

- `git log --graph --oneline`
  Historial gráfico en arte ASCII

- `git blame`
  ¿Cuándo fue la última modificación de cada línea?

- `git bisect`
  Busqueda binaria en la historia

Commits Atomicos

Un commit que aplica **un** cambio **completo**.

- Una sola razón para todo lo que cambió
- No depende de cambios externos

Facilita manejar commits:

- Revertir
- Reordenar
- Búsquedas en la historia

Más comandos útiles

Para manejar interrupciones:

- `git stash push -m <comment>`
  `git stash pop`
  Guardar estado de archivos temporalmente

- `git add -p`
  Commits de partes de archivos

- `git commit`
  `git reset <commit>`
  Hacer y deshacer commits

Historia Lineal

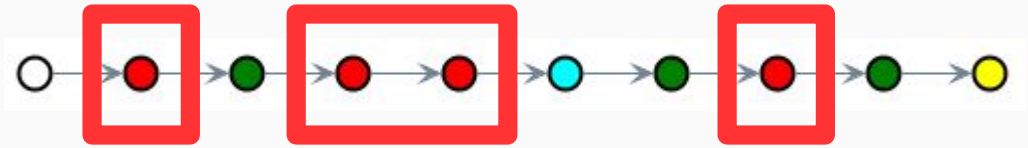
Típica en proyectos:

- Simples
- Poca superposición

Historia Lineal

Desventajas:

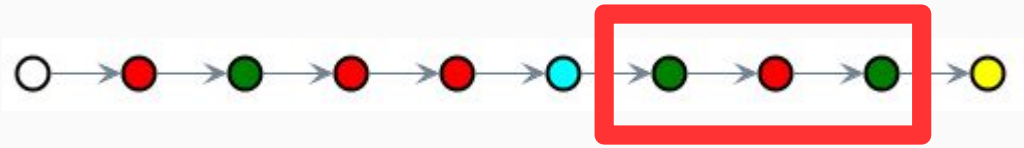
- Cuesta separar líneas de trabajo



Historia Lineal

Desventajas:

- Cuesta separar líneas de trabajo
- Estados intermedios inesperados



Historia Lineal

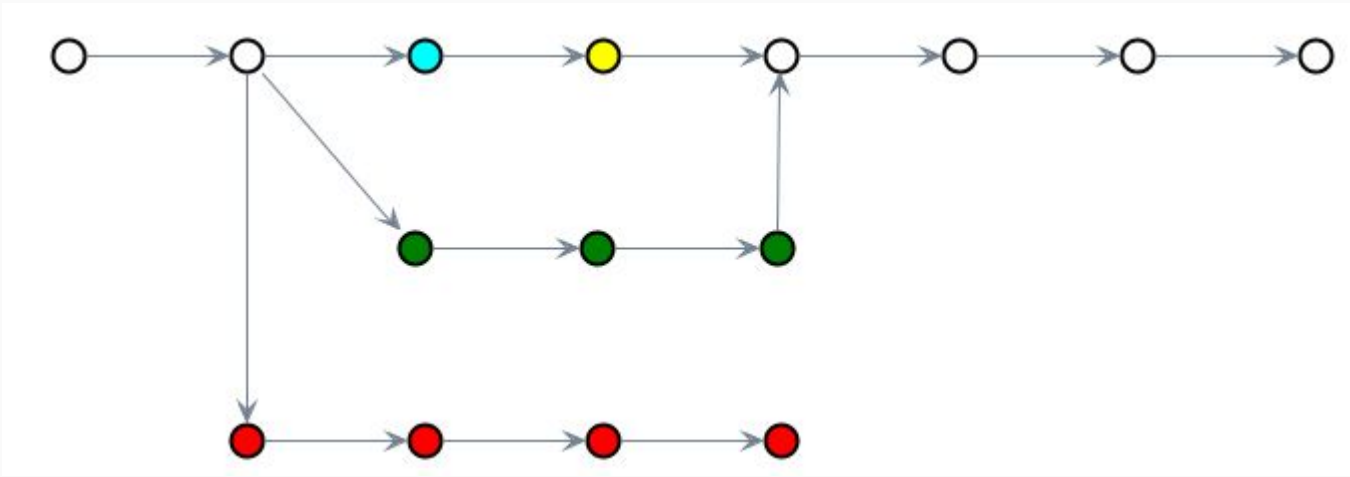
Problema central:

El control de versiones es naturalmente distribuido
(⇒ tambien es concurrente)

Feature Branches

Branches dedicados a hacer un feature

- Menos cambios en cada branch
- Menos gente trabajando en cada branch
- El resto del equipo ve todos los cambios juntos



Feature Branches

Caso particular: **Branch de colaboracion**

- Usado para compartir código en progreso
- Suele ser temporal
  - Menores estándares de calidad
  - Rápidamente eliminada o mejorada
- Opción para evitar colaborar por fuera del repositorio

Conflicto

Al hacer merge, hay **conflicto** cuando los cambios de cada rama interactúan de forma negativa.

Conflicto textual:

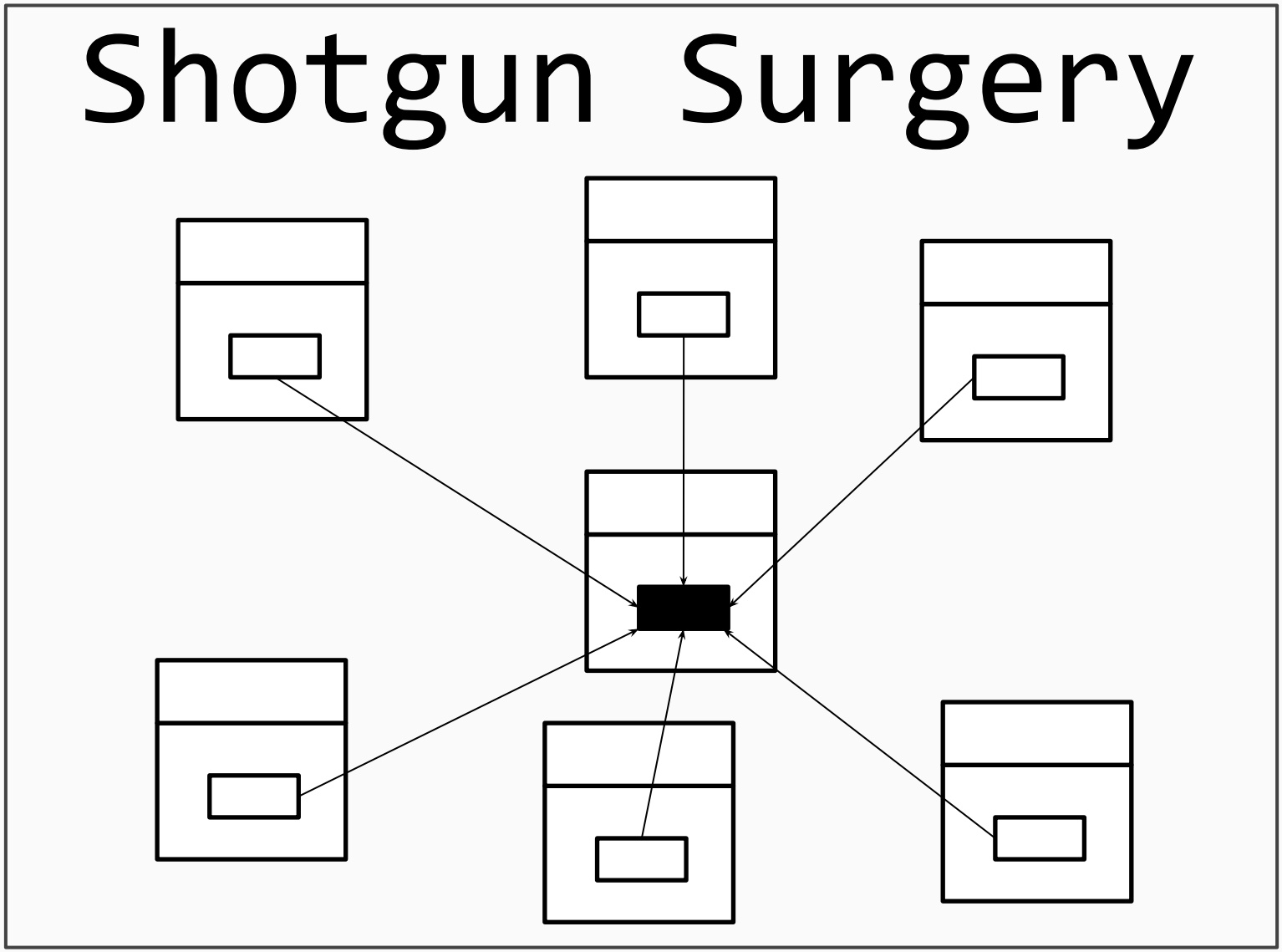
Mismas líneas ⇒ git puede detectarlo

Conflicto semántico:

Misma lógica ⇒ requiere analizar la interacción

Conflicto

Relación con el buen diseño (por ejemplo):



Pull Requests

Tambien llamadas **Merge Requests**

Antes del merge, hay oportunidad de verificar:

- Calidad del código
- Calidad de la aplicación
- Cumplimiento de requerimientos
- Si el cambio conviene

Para iniciar estas verificaciones, existe el proceso de **Pull Request**

Pull Requests

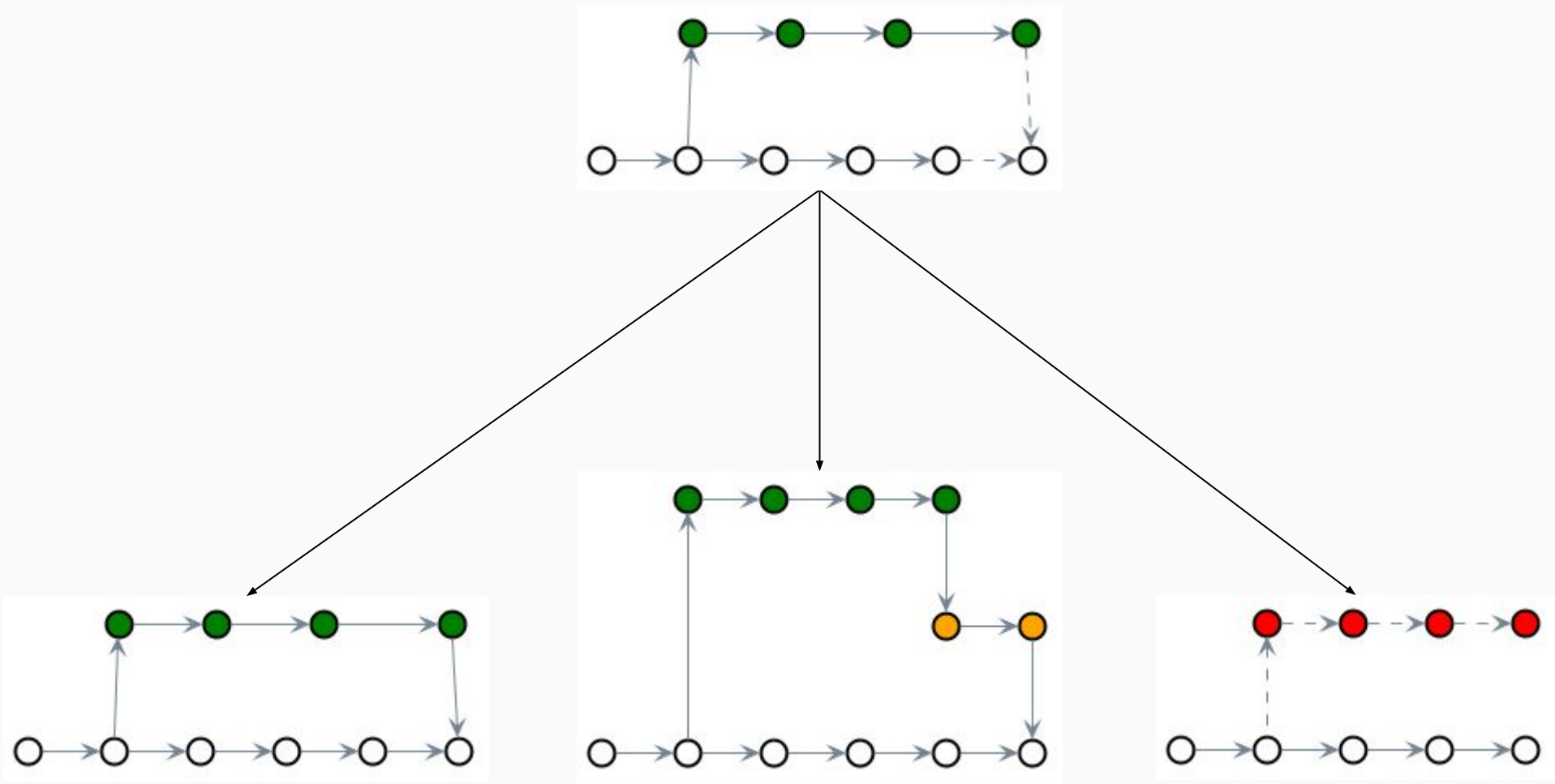
Un buen PR:

- Es atomico:
  - Una sola razón para todo lo que cambió
  - No depende de cambios externos

- Explica:
  - Razones que no resultan obvias del resumen
  - Efectos de los cambios que no resultan obvios al ver el código

Es lo mismo que queremos en commits, pero a mayor escala (horas - días)

Pull Requests



PR Aceptado

PR Aceptado con cambios

PR Rechazado

CI/CD

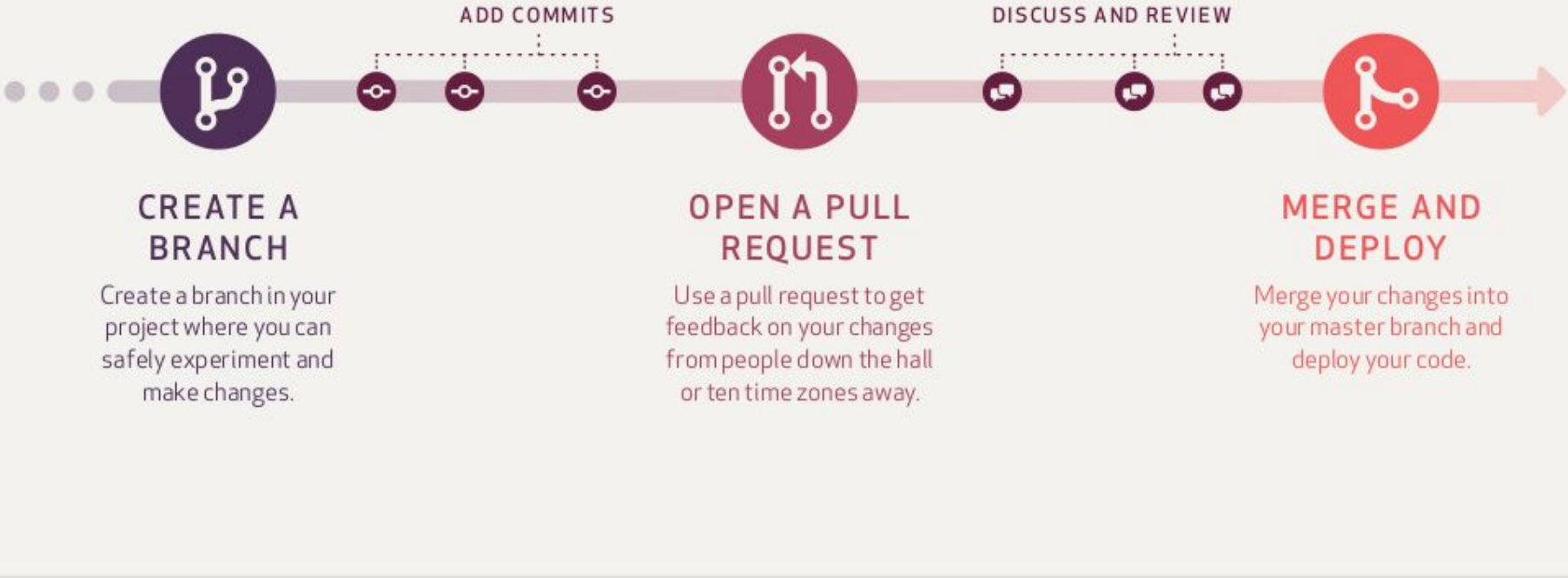
Continuous Integration

- Integrar frecuentemente el trabajo realizado
- Compilar y testear
- Usualmente de forma automatica

Continuous Deployment

- Asegurar que podemos producir un entregable de forma rapida y certera
- Desplegar a entornos de prueba
- Usualmente de forma automatica

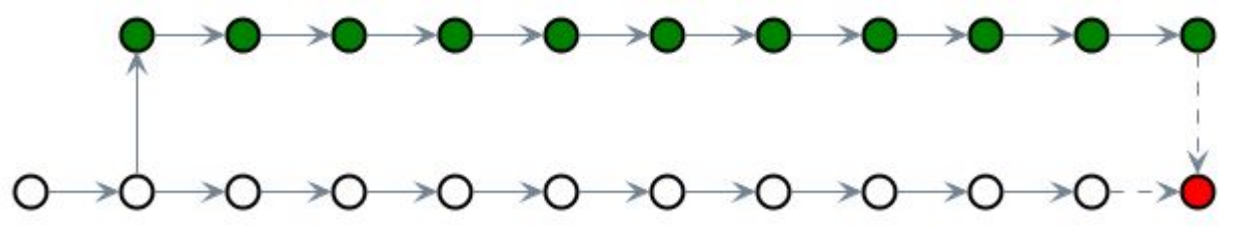
Github Flow


ADD COMMITS
DISCUSS AND REVIEW
CREATE A BRANCH
Create a branch in your project where you can safely experiment and make changes.
OPEN A PULL REQUEST
Use a pull request to get feedback on your changes from people down the hall or ten time zones away.
MERGE AND DEPLOY
Merge your changes into your master branch and deploy your code.

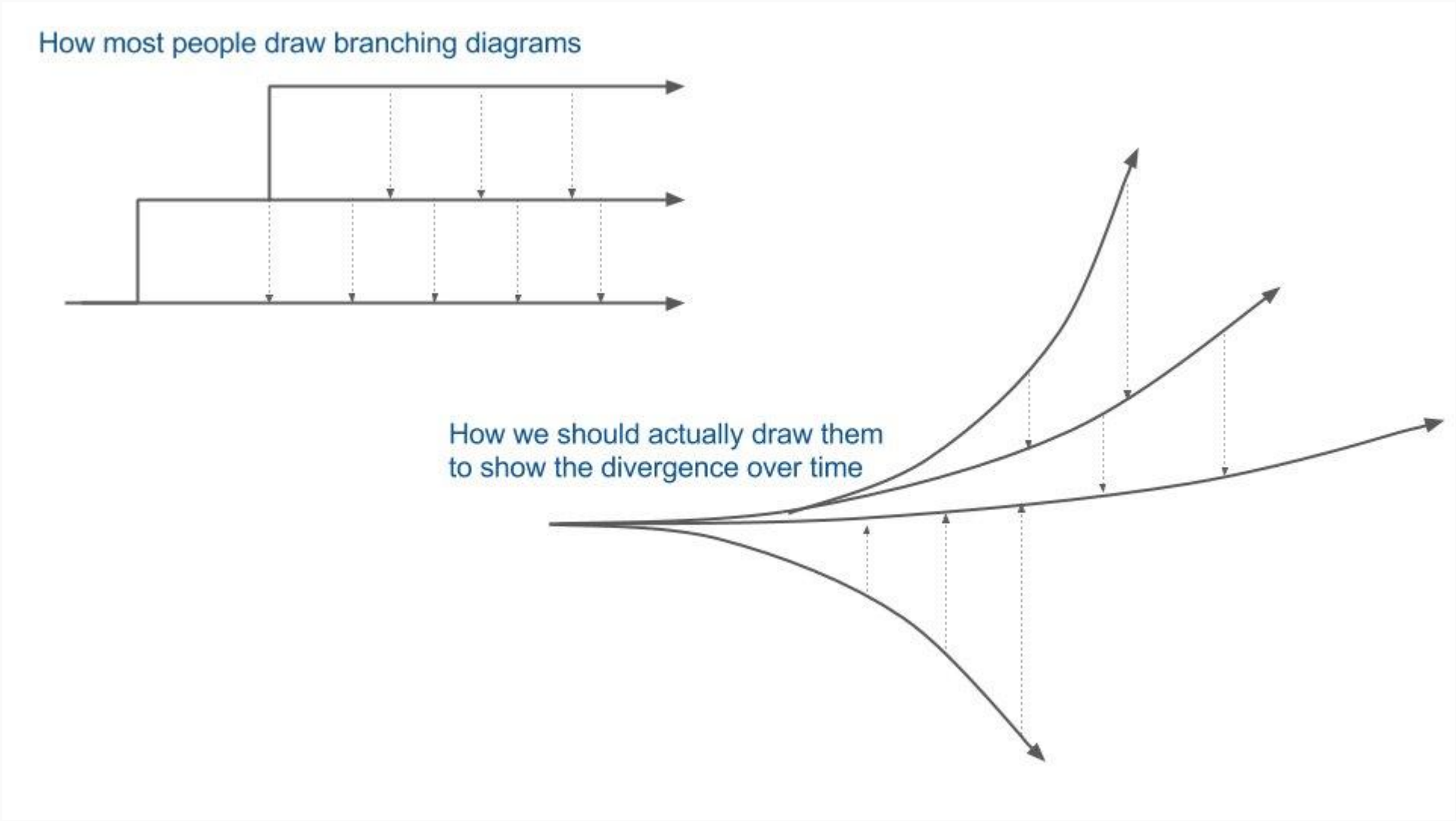
Branches a Largo Plazo

Si mantenemos un branch separando durante demasiado tiempo:

- Bugs hallados y arreglados en ambos branches
- Hay que reconciliar los conflictos



Branches a Largo Plazo

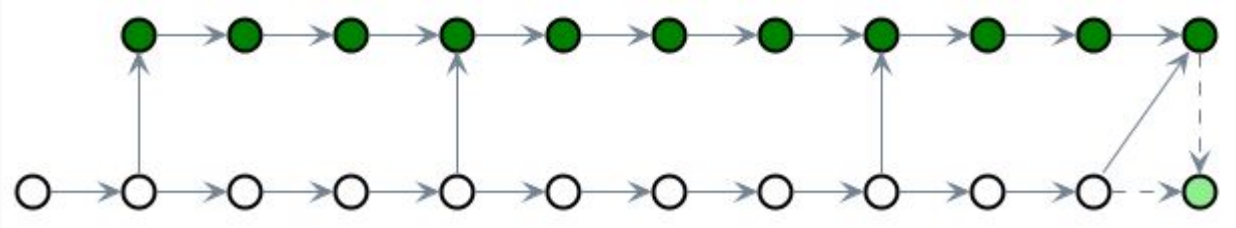


https://twitter.com/jahnnie/status/937917022247120898

Branches a Largo Plazo

Para reducir los problemas, hacer merge intermedios:

- Menos cambios en cada merge
- Esos cambios son más recientes
- Evitamos duplicar trabajo



Branches a Largo Plazo

Mejor solución: No hacer branches a largo plazo

- Modularizar mejor: Separar variantes por ubicación en lugar de tiempo
- Feature flags: Los branches comparten su código, la distinción se hace por configuración

Pero: A veces representa el significado que queremos

- Manejar cuidadosamente la divergencia

Branches de Entorno

Branch a largo plazo que:

- Representa un entorno de despliegue
- Usa CI/CD para desplegar cambios

Algunos nombres comunes:

- production, main, master
- uat
- qa
- development, dev

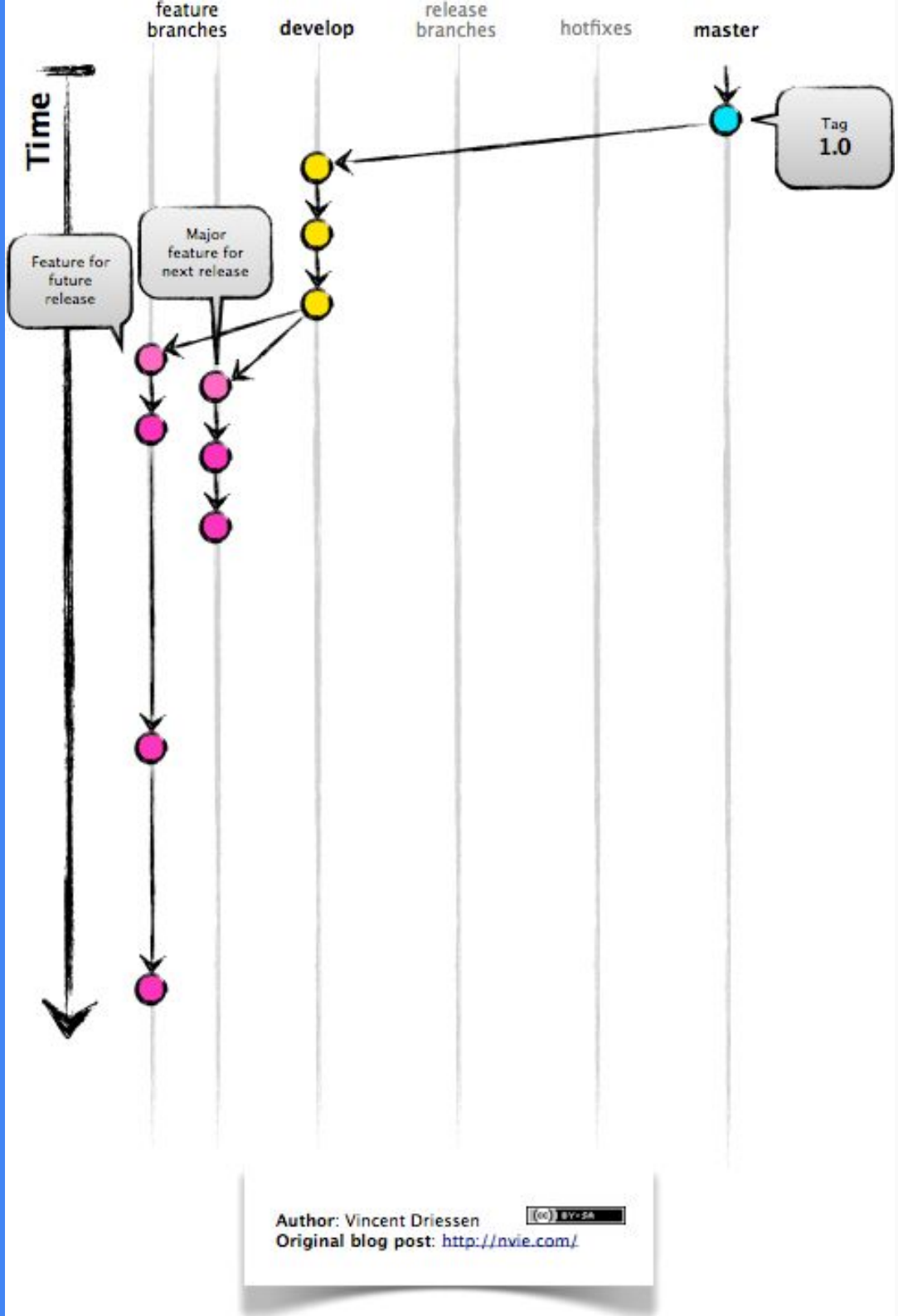
Branches de Entorno

production	uat	qa	development

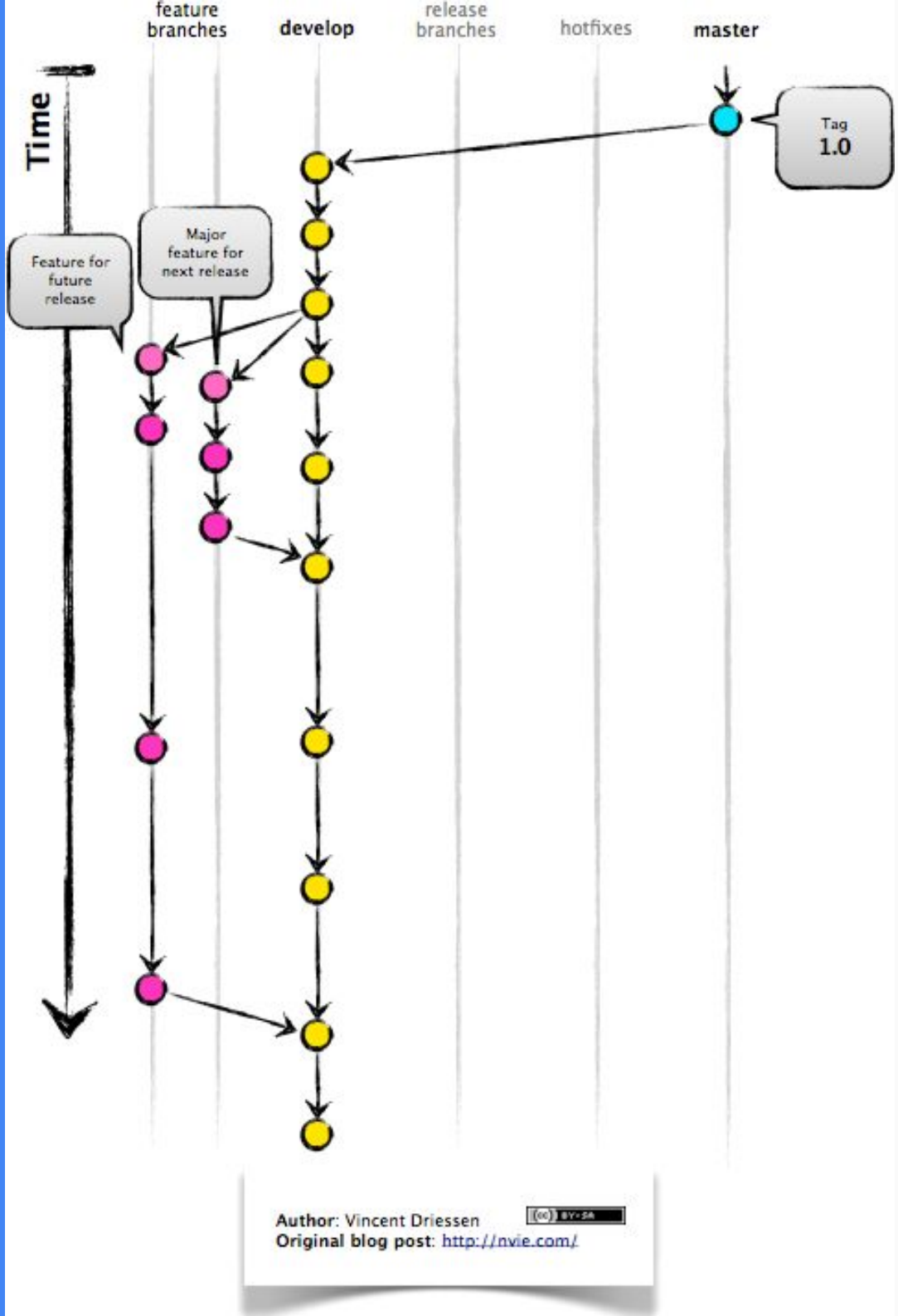
- Elegimos algun conjunto especifico
- Definimos algún orden de más a menos estables

- Hotfixes propagan de más a menos estables
- Desarrollo en entorno menos estable
- Estabilizar antes de pasar a un entorno más estable

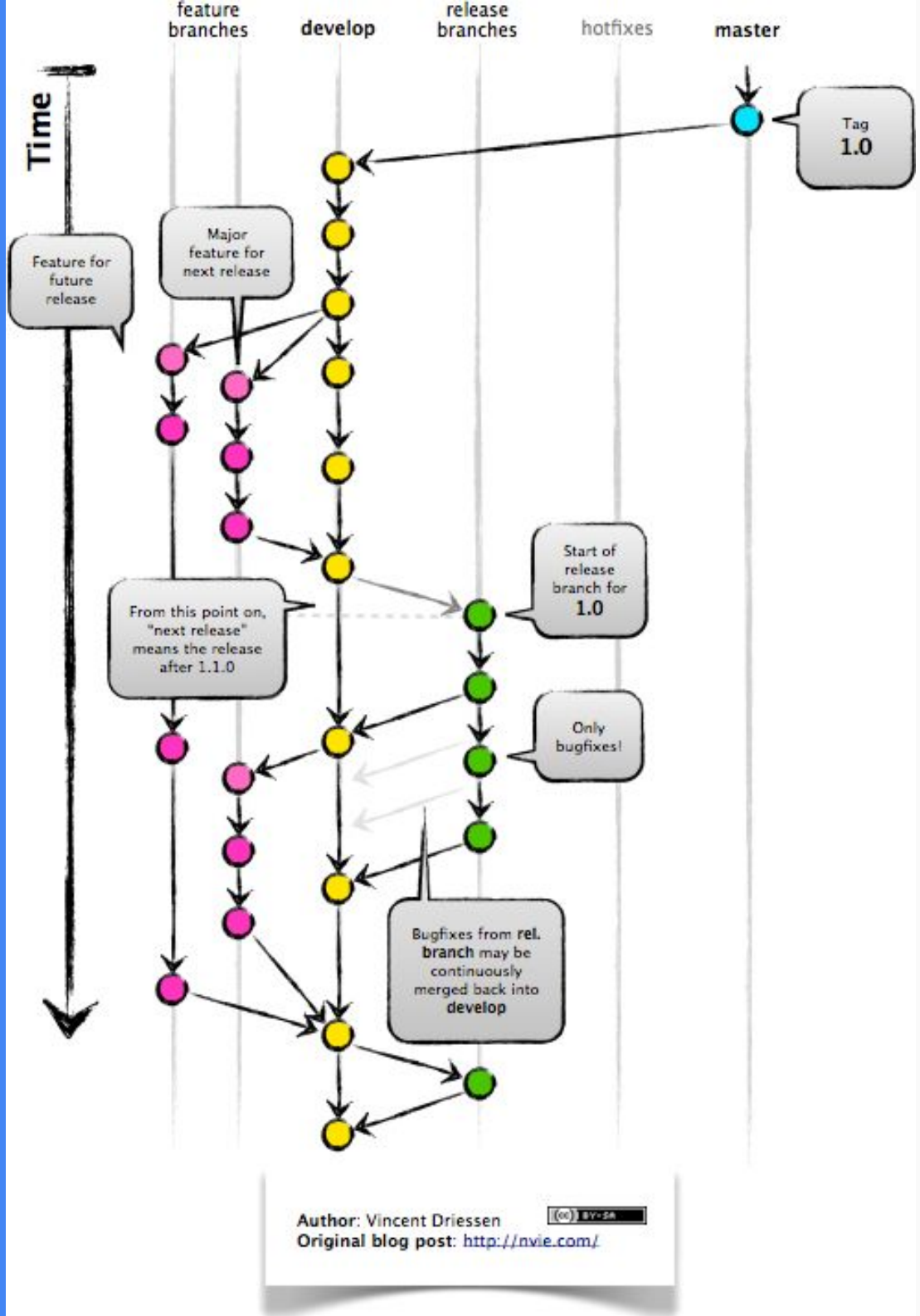
Git Flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 1.0
Feature for future release
Major feature for next release
Author: Vincent Driessen
Original blog post: http://nvie.com/

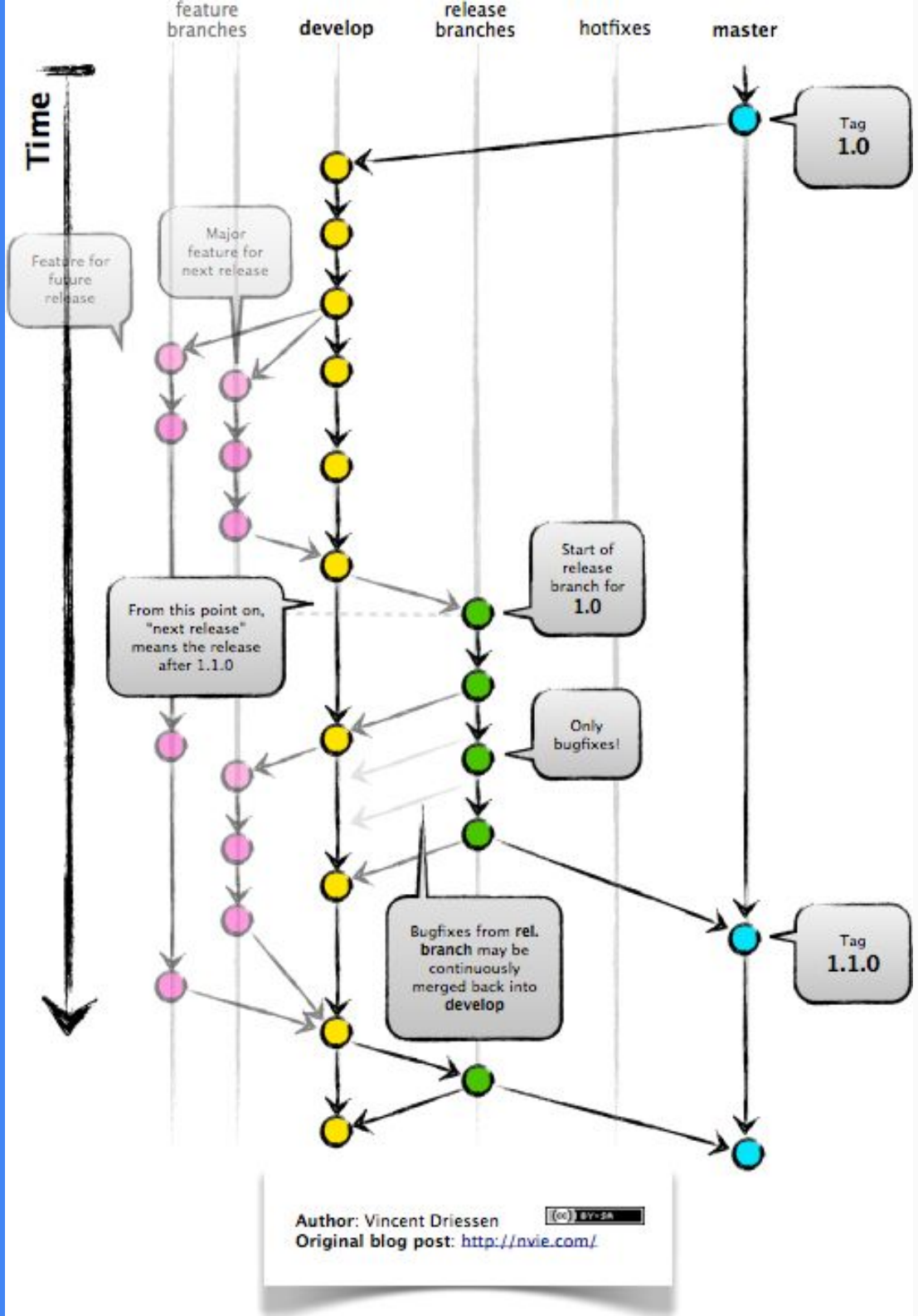
Git Flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 1.0
Feature for future release
Major feature for next release
Author: Vincent Driessen
Original blog post: http://nvie.com/

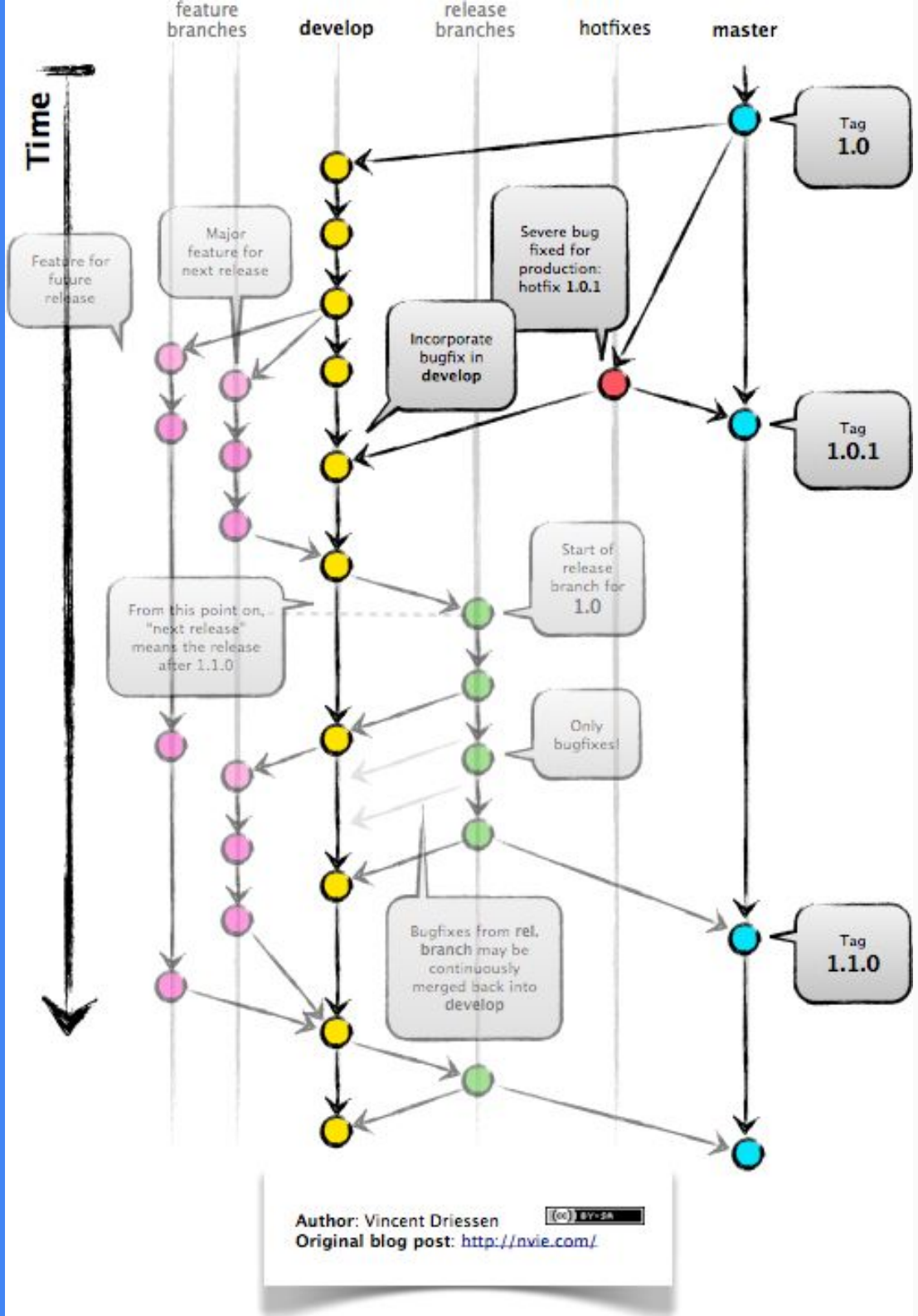
Git Flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 1.0
Feature for future release
Major feature for next release
From this point on, "next release" means the release after 1.1.0
Start of release branch for 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Author: Vincent Driessen
Original blog post: http://nvie.com/

Git Flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 1.0
Feature for future release
Major feature for next release
Start of release branch for 1.0
From this point on, "next release" means the release after 1.1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.1.0
Author: Vincent Driessen
Original blog post: http://nvie.com/

Git Flow


feature branches
develop
release branches
hotfixes
master
Time
Tag 1.0
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 1.0.1
Incorporate bugfix in develop
Tag 1.0.1
Start of release branch for 1.0
From this point on, "next release" means the release after 1.1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.1.0
Author: Vincent Driessen
Original blog post: http://nvie.com/

Git Flow vs Github Flow

Github Flow:

- Verificación dentro de cada PR
- Software con despliegues simples-rapidos-baratos
  - SaaS
  - Aplicaciones web

Git Flow:

- Verificación al preparar cada release
- Software con despliegues complejos-lentos-caros
  - Aplicaciones mobile o de escritorio
  - Instalaciones on-premise
  - Múltiples versiones simultáneas

Preguntas?

Recursos

- Try Git

- Branching Patterns (Martin Fowler)
- How to Write a Git Commit Message (Chris Beams)

- Managing your work on GitHub

- Gitlab CI Documentation

- Git Flow
- GitHub Flow
- GitLab Flow