



Gerência de Configuração

Obtendo os slides

- ▶ Disponíveis apenas eletronicamente
- ▶ Obtenha o Git (<http://www.git-scm.com/>)
- ▶ Clone o repositório da disciplina:

```
git clone git://github.com/fernandocastor/scm.git
```

Módulos

1. Introdução

2. Padrões GC

**3. Disciplina de
GC e Ambiente**

**4. Controle de
Mudanças**

**5. Sistemas de
Controle de
Versão**

**6. Integração
Contínua**

**7. Sistemas de
Controle de
Mudanças**

Módulos

1. Introdução

2. Padrões GC

**3. Disciplina de
GC e Ambiente**

**4. Controle de
Mudanças**

**5. Sistemas de
Controle de
Versão**

**6. Integração
Contínua**

**7. Sistemas de
Controle de
Mudanças**

Talvez não tenhamos tempo para cobrir todos os tópicos.



Gerência de Configuração

Introdução

Objetivos

- ▶ Fornecer os principais conceitos relacionados a GC
- ▶ Criar uma visão geral de como GC pode ser aplicada ao seu projeto

Pré-requisitos

▶ Noções de Engenharia de Software

► Problemas na manutenção de software:

- bugs previamente corrigidos que reaparecem
- Features já testadas que somem repentinamente
- Um programa que deixa de funcionar

Estes são problemas frustrantes
provenientes da falta de gerência de
configuração

Outros problemas

- ▶ Ambientes que funcionam somente para parte dos desenvolvedores;
- ▶ Descontrole sobre o que deve ser empacotado no ultimo release do sistema
- ▶ Erros que ocorrem em produção não podem ser reproduzidos em desenvolvimento
- ▶ Problemas de coordenação do trabalho concorrente dos desenvolvedores

Problema da Quebra de Comunicação

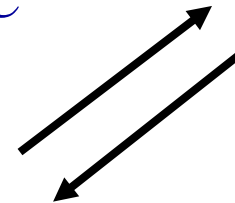
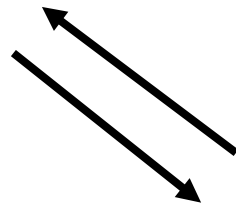
Desenvolvedor A



Desenvolvedor B



Desenvolvedor C



Problema da Quebra de Comunicação (continuação)

▶ Ocorrem pelas mais diversas razões:

- Vocabulários incompatíveis
- Culturas de desenvolvimento diferentes
- Distância geográfica
- Dificuldade de expressão

▶ Quando este problema acontece:

- Os sistemas produzidos não atendem aos requisitos
- Força de trabalho é desperdiçada

Problema dos Dados Compartilhados

Desenvolvedor A



Programa de A

A1

A2

A3

**Componente
Compartilhado**

Desenvolvedor B



Programa de B

B1

B2

B3

Problema dos Dados Compartilhados - Cenário

- ▶ O desenvolvedor A modifica o componente compartilhado
- ▶ Mais tarde, o desenvolvedor B realiza algumas alterações no mesmo
- ▶ Quando B tenta compilar o componente, erros são apontados pelo compilador, mas nenhum deles ocorre na parte que B alterou
- ▶ O desenvolvedor B não tem a menor idéia sobre a causa do problema

Problema dos Dados Compartilhados - Solução simplista

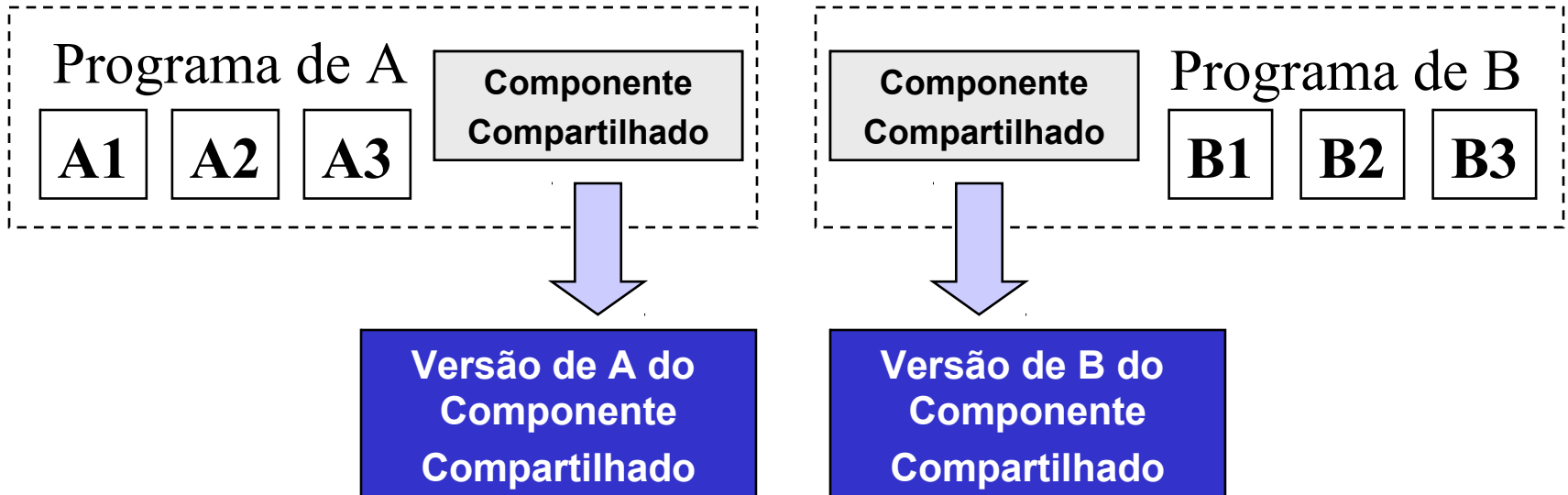
► Solução simplista:

- cada desenvolvedor trabalha em uma cópia “local” do componente
- resolve o Problema dos Dados Compartilhados, mas cria um novo problema

Problema da Manutenção Múltipla

Desenvolvedor A

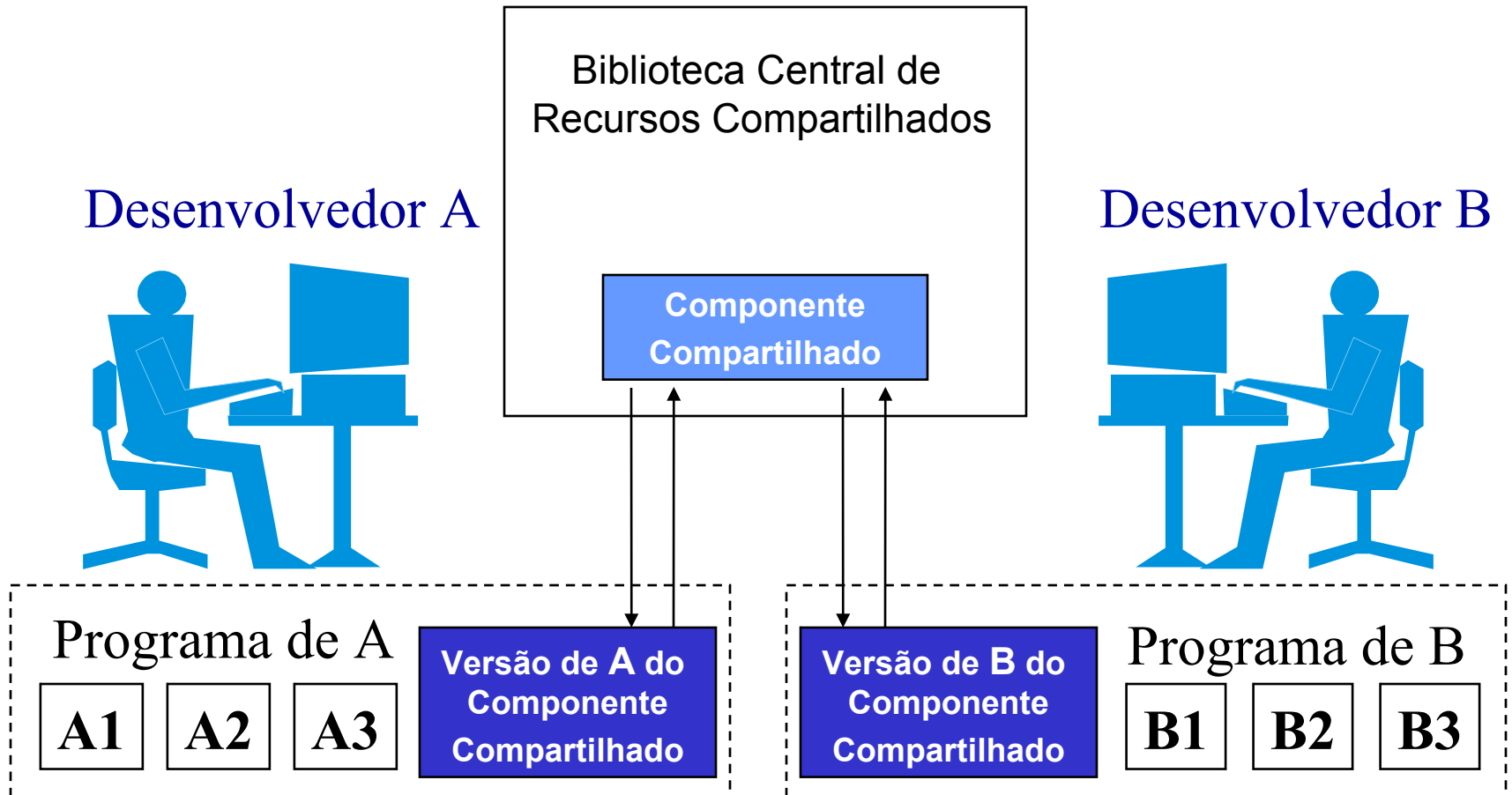
Desenvolvedor B



Problema da Manutenção Múltipla (continuação)

- ▶ Ocorre quando cada desenvolvedor trabalha com uma cópia “local” do que seria o mesmo componente
- ▶ Dificuldade para saber:
 - Que funcionalidades foram implementadas em quais versões do componente
 - Que defeitos foram corrigidos
- ▶ Evitado através de uma biblioteca central de componentes compartilhados
 - Nesse esquema, cada componente é copiado para a biblioteca sempre que alterado
 - Resolve o Problema da Manutenção Múltipla, mas...

Problema da Atualização Simultânea



Problema da Atualização Simultânea - Cenário 1:

- O desenvolvedor A encontra e corrige um defeito em sua versão do componente compartilhado
- Uma vez corrigido, o componente modificado é copiado para a biblioteca central
- O desenvolvedor B encontra e corrige o mesmo defeito em sua versão do componente por não saber que A já tinha feito isso
- O trabalho de A é desperdiçado

Problema da Atualização Simultânea - Cenário 2:

- O desenvolvedor A encontra e corrige um defeito em sua versão do componente compartilhado
- Uma vez corrigido, o componente modificado é copiado para a biblioteca central
- O desenvolvedor B encontra e corrige um outro defeito em sua versão do componente, sem saber do defeito corrigido por A
- O desenvolvedor B copia sua versão do componente para a biblioteca central
- Além de o trabalho de A ser desperdiçado, a versão do componente que se encontra na biblioteca central continua apresentando um defeito
- O desenvolvedor A julga o problema como resolvido

Como Resolver?

- ▶ O problema da atualização simultânea não pode ser resolvido simplesmente copiando componentes compartilhados para uma biblioteca central
- ▶ Algum mecanismo de controle é necessário para gerenciar a entrada e saída dos componentes

Sobre Gerência de Configuração

- ▶ “Em qualquer time, um certo grau de confusão é inevitável. O objetivo é minimizar a confusão de modo que mais trabalho possa ser feito.”
- ▶ “A arte de coordenar o desenvolvimento de software para minimizar esse tipo particular de confusão é chamada de Gerência de Configuração.”
- ▶ W.A. Babich

O que é Gerência de Configuração?

- ▶ Gerência de configuração (GC) é o processo de identificar, organizar e controlar modificações ao software sendo construído
- ▶ A idéia é maximizar a produtividade minimizando os enganos

Objetivos de GC

- ▶ Definir o ambiente de desenvolvimento
- ▶ Políticas para controle de versões garantindo a consistência dos artefatos produzidos
- ▶ Definir procedimentos para solicitações de mudanças
- ▶ Administrar o ambiente e auditar mudanças
- ▶ Facilitar a integração das partes do sistema

Benefícios

- ▶ Aumento de produtividade no desenvolvimento
- ▶ Menores Custos de Manutenção
- ▶ Redução de defeitos
- ▶ Maior rapidez na identificação e correção de problemas



Conceitos Básicos

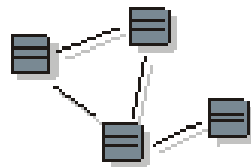


Configuração do Software

- ▶ Um projeto de desenvolvimento de software produz os seguintes itens:
 - Programas (código fonte, programas executáveis, bibliotecas de componentes, etc.)
 - Documentação (manuais do usuário, documento de requisitos, modelo de análise e projeto, etc.)
 - Dados (dados de teste e do projeto)
- ▶ Esses conjuntos de itens são chamados, coletivamente, de configuração do software

Workspace

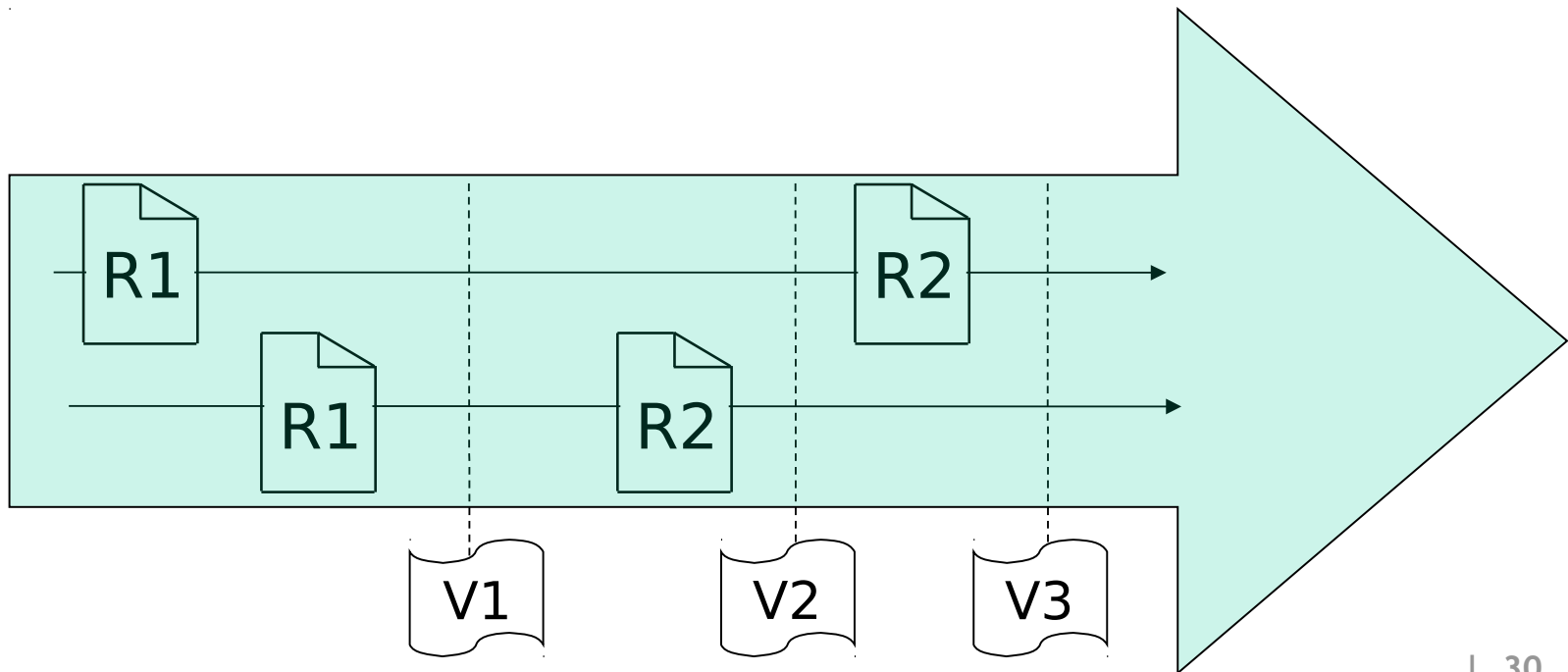
- Todos os artefatos necessários para a execução de uma tarefa
 - O desenvolvedor pode exercer várias tarefas
 - e ter vários workspaces
 - Podem ser gerenciados por uma IDE.
- A configuração do software pode ser composta por vários workspaces

Item de Configuração

- ▶ Conjunto de itens de hardware e software vistos como uma entidade única para fins de gerência de configuração 
- ▶ Sujeito a mudanças e essas devem obedecer às políticas estabelecidas 
- ▶ Cada pedaço de software que pode ser projetado, implementado e testado de forma independente 

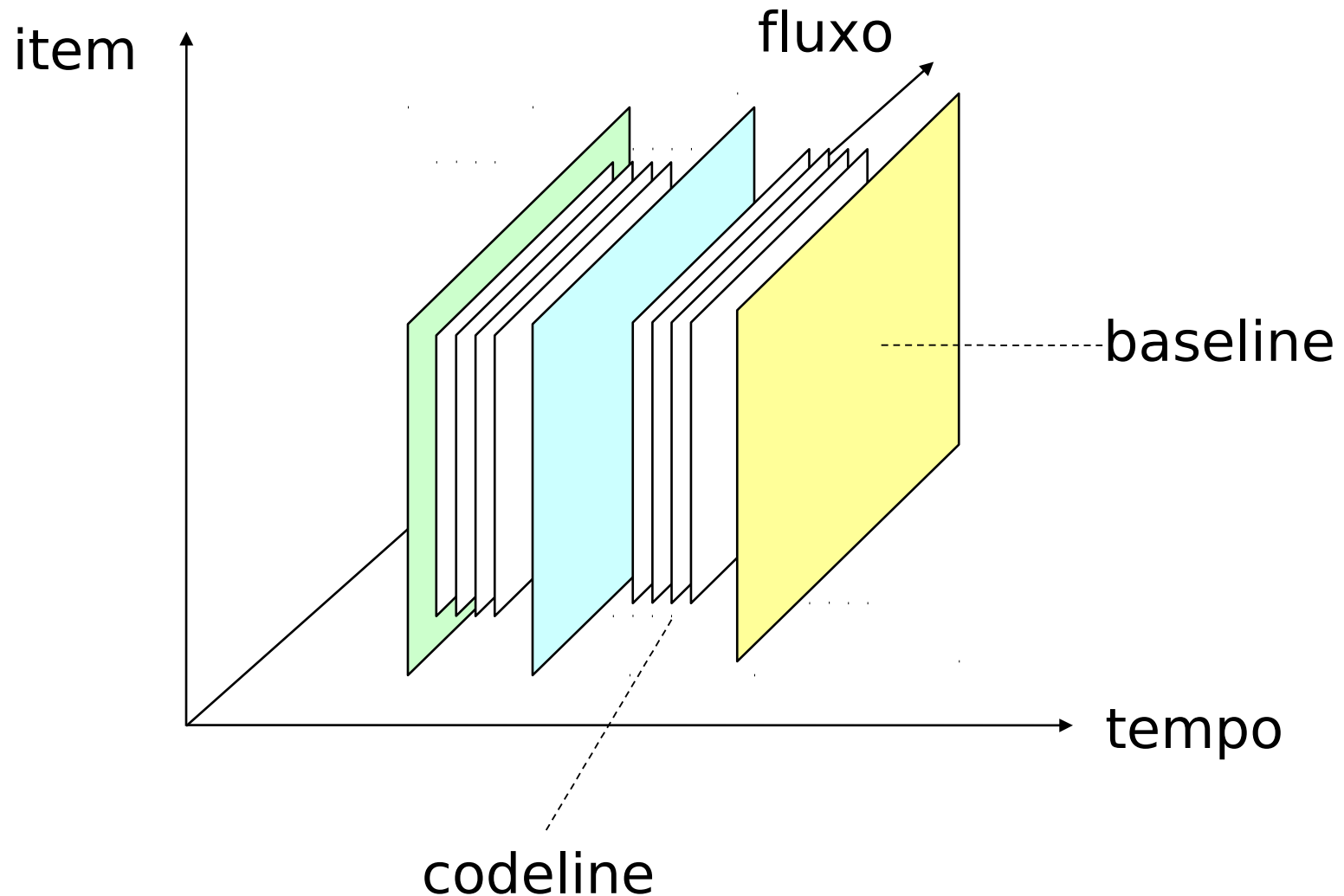
Codeline

- ▶ Uma codeline contém todas as versões de cada artefato sobre um ramo evolucionário do sistema
 - Codelines armazenam todas as mudanças



- ▶ Uma especificação ou produto que foi formalmente revisado e aceito
 - Serve como base para os passos posteriores do desenvolvimento
- ▶ A configuração do software em um ponto discreto no tempo
- ▶ Só pode ser modificado através de procedimentos formais (solicitações de mudança)
- ▶ Um artefato ou conjunto de artefatos só se torna um item de configuração depois que um baseline é estabelecido

Baseline e Codelines



Razões para Criar um Baseline

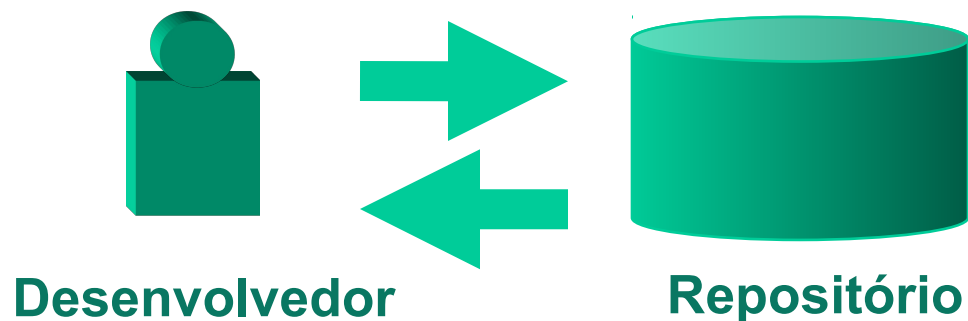
- **Reprodutibilidade** – a habilidade de reproduzir uma versão anterior do sistema
- **Rastreabilidade** – Estabelece uma relação predecessor-sucessor entre artefatos do projeto (projeto satisfaz requisitos, código implementa projeto, etc.)
- **Geração de Relatórios** – A comparação dos conteúdos de dois *baselines* ajuda na depuração e criação de documentação
- **Controle de Mudanças** – referencial para comparações, discussões e negociações

Baselines importantes

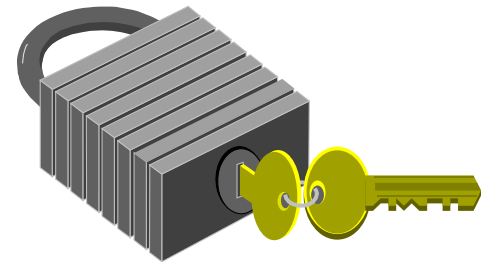
- ▶ Baselines são considerados marcos no processo de desenvolvimento:
 - Funcional : requisitos
 - De Produto : releases, iterações

Repositório

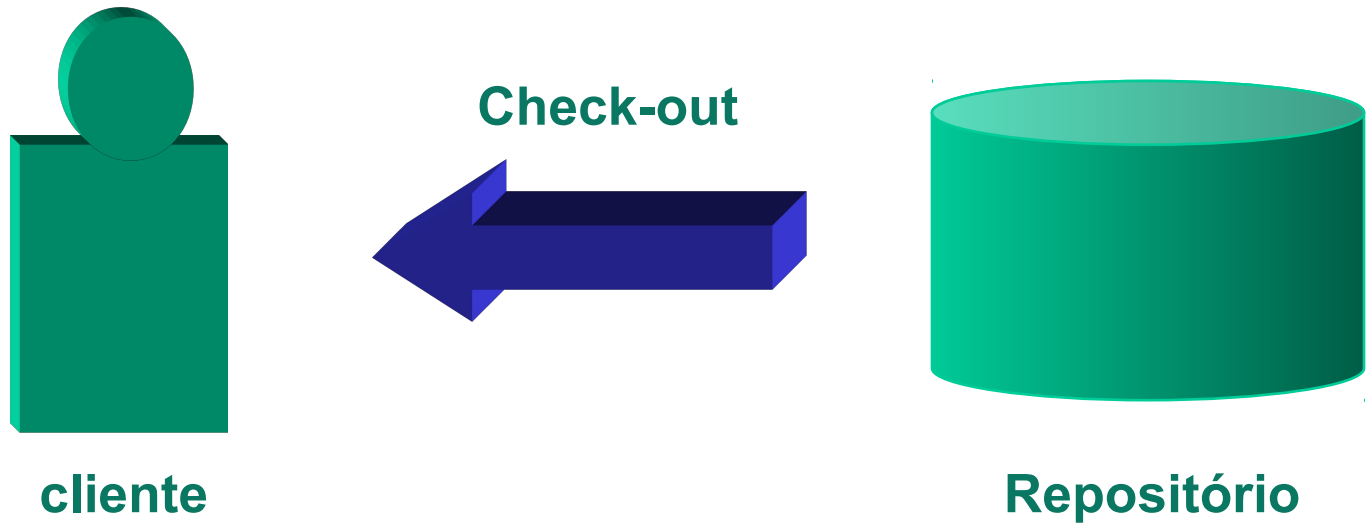
- ▶ Local (físico e lógico) onde os itens de um sistema são guardados
- ▶ Pode conter diversas versões do sistema
- ▶ Utiliza mecanismos de controle de acesso



- ▶ Resolve a Atualização Simultânea
- ▶ Garante que apenas o usuário que detém o lock pode alterar o arquivo
- ▶ Problema: “serializa” o trabalho dos desenvolvedores



Check-Out



Check-Out (continuação)

► Recuperar a (última) versão de um item de configuração guardada no repositório

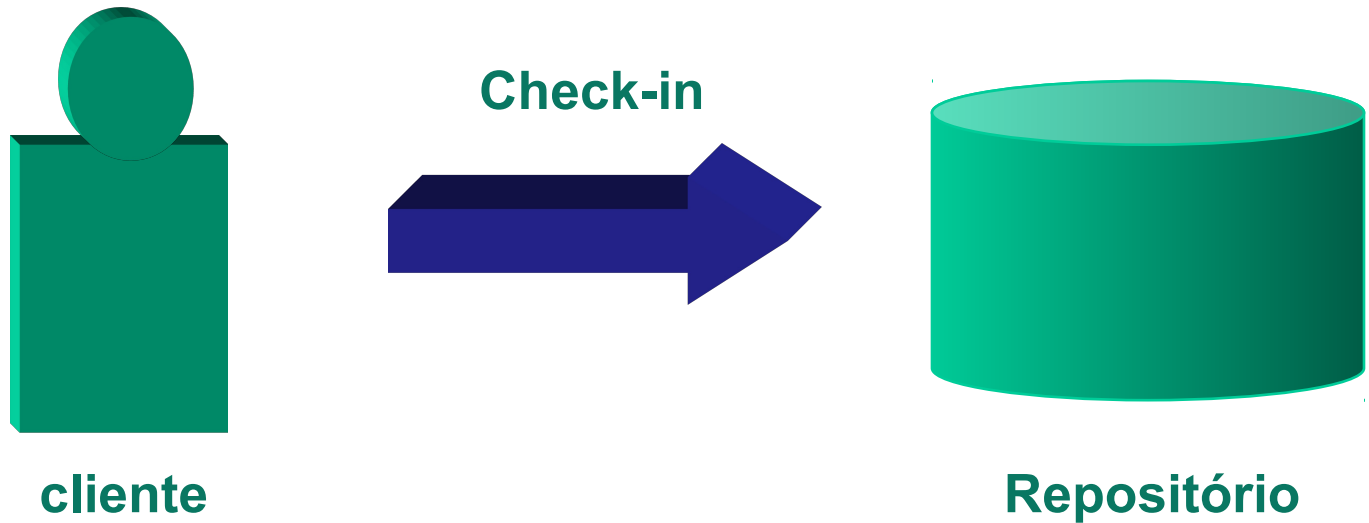
- Escrita

- Verifica que ninguém detém o lock do item de configuração
- Obtém o lock do item
- Cria uma cópia, para edição, no cliente

- Leitura

- Verifica que alguém já detém o lock
- Cria uma cópia, apenas para leitura, no cliente

Check-In

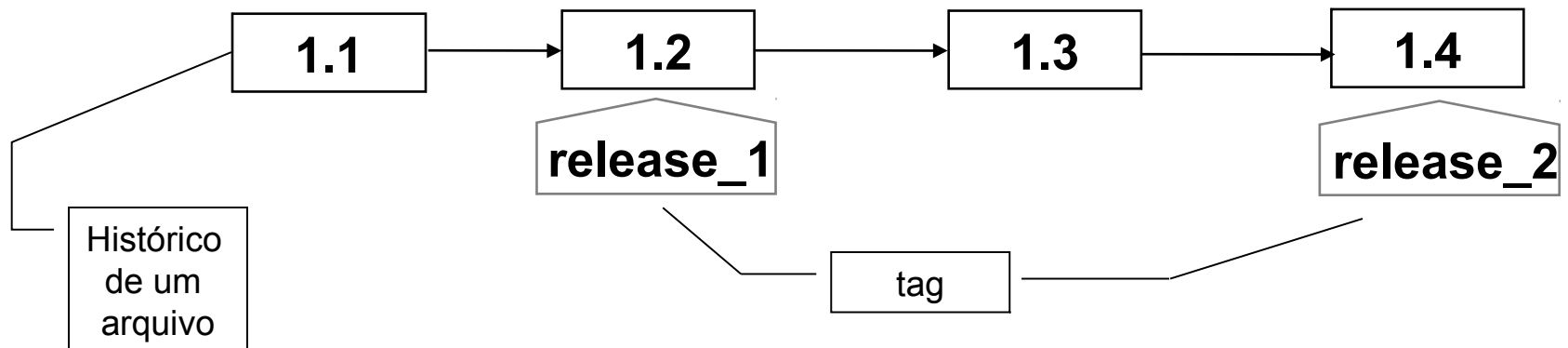


Check-In (continuação)

- ▶ Ação de inserir/atualizar um item de configuração no repositório
 - Verifica o lock do item de configuração, caso o mesmo já exista
 - Verifica e incrementa a versão do item
 - Registra informações das mudanças (autor, data, hora, comentários)
 - Inclui/atualiza o item

- ▶ Rótulos que são associados a conjuntos de item de configuração
- ▶ Um tag referencia um ou mais arquivos em um ou mais diretórios
 - Costuma-se usar tags para:
 - Denominar projeto rotulando todos os arquivos associados ao projeto
 - Denominar uma versão do projeto rotulando todos os arquivos do build ou release

Tags



Conflitos e Resolução

▶ Locks são um mecanismo de controle **pessimista**

- Só um desenvolver de cada vez pode alterar um artefato
- Na prática, isso atrasa o desenvolvimento
- Evita o problema da **atualização simultânea**, porém

▶ Como evitar **todos** os problemas acima?

Otimismo é a Chave!

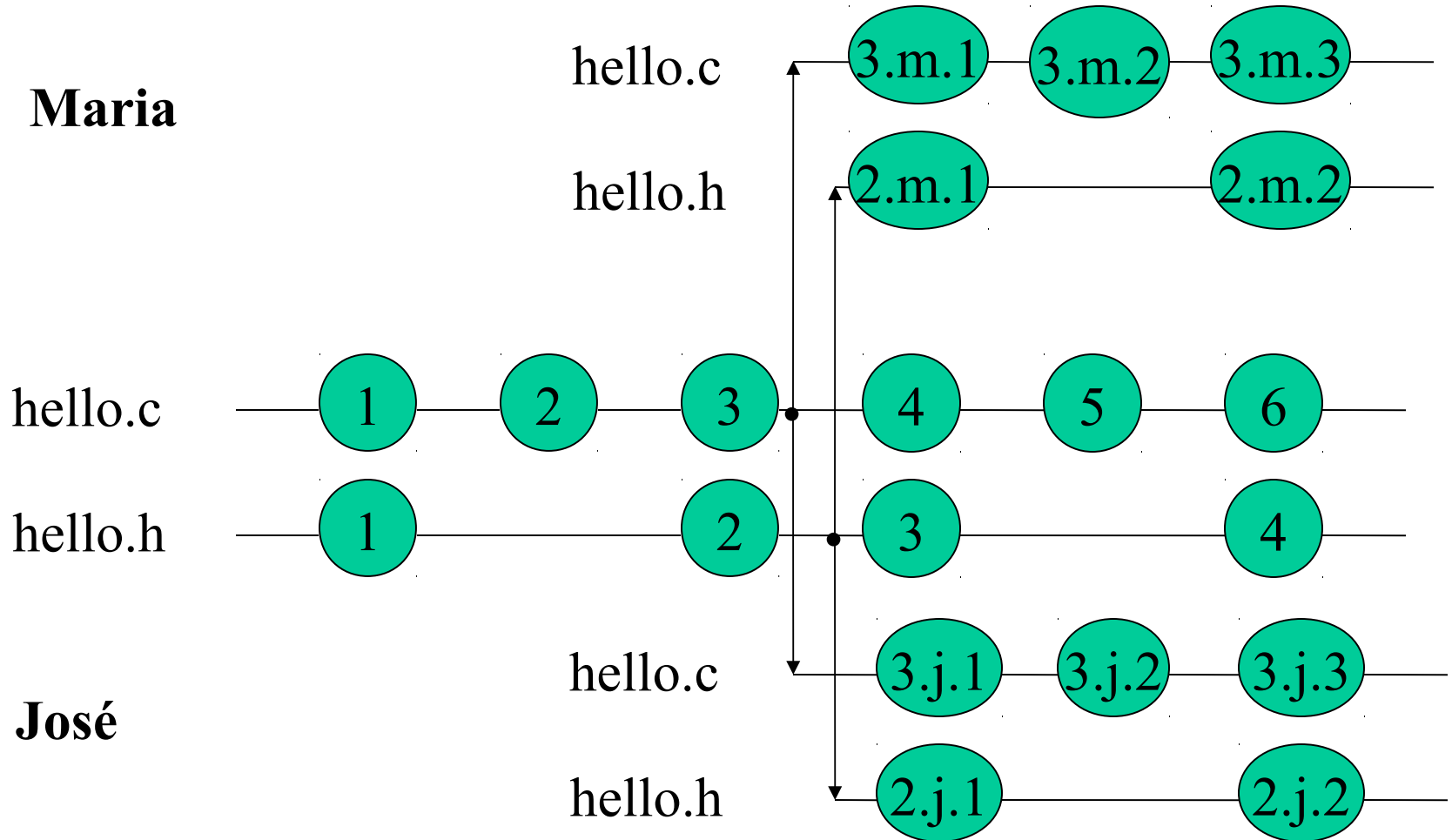
- ▶ Admite-se que **conflitos** podem ocorrer
 - Durante atualizações simultâneas
 - Espera-se que ocorram **pouco**
 - Quando ocorrerem, são **resolvidos**
- ▶ Exemplo...

- ▶ Criação de um fluxo alternativo para atualização de versões de itens de configuração
- ▶ Recurso muito poderoso
- ▶ Regras bem definidas para criação de branches
 - Por que e quando devem ser criados?
 - Quais os passos?
 - Quando retornar ao fluxo principal?

Branch (continuação)

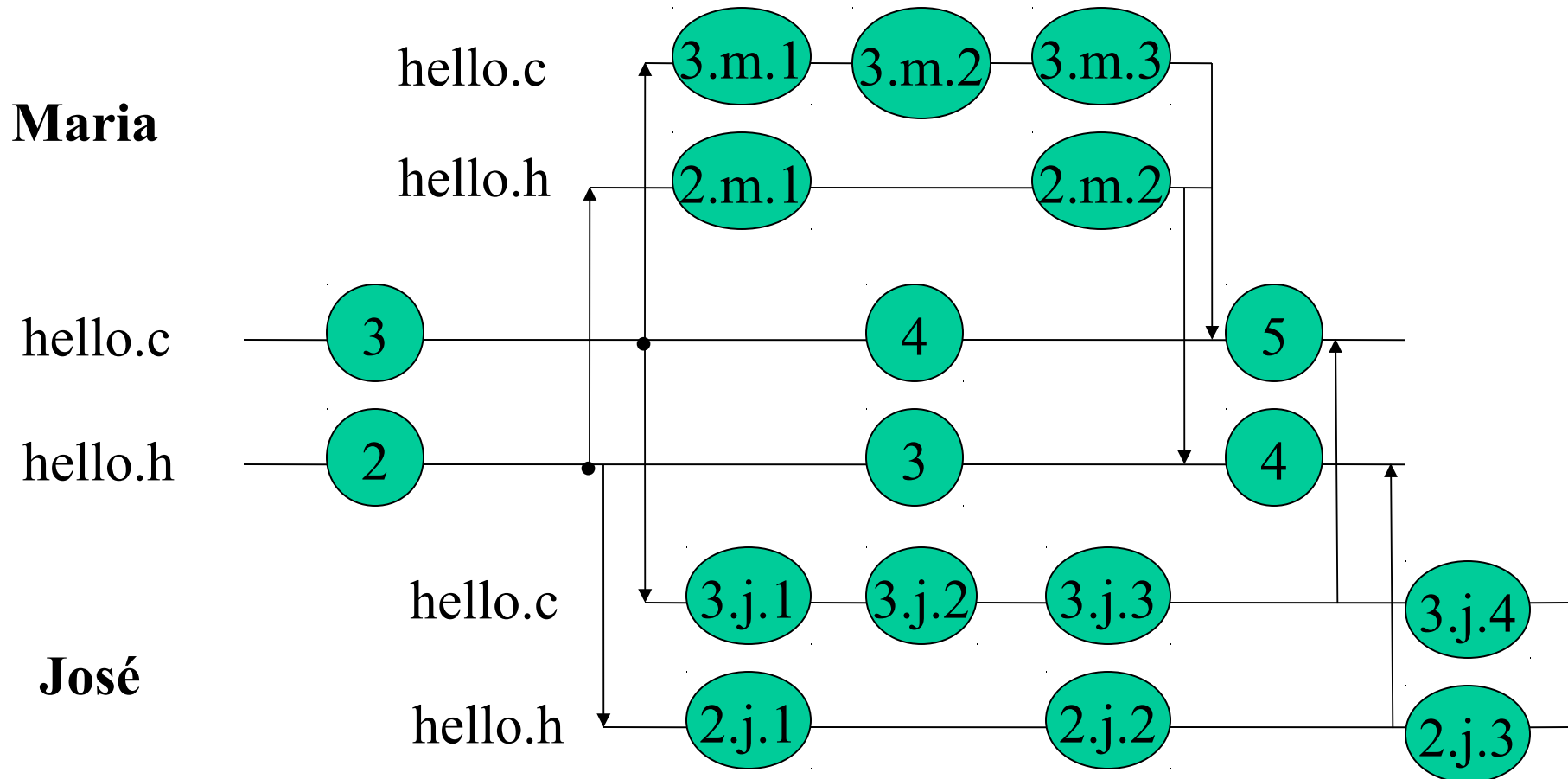
- ▶ Uso de *lock* inviabiliza a criação de *branches*
- ▶ *Branches* normalmente se originam de correções em versões anteriores

Branch (exemplo)

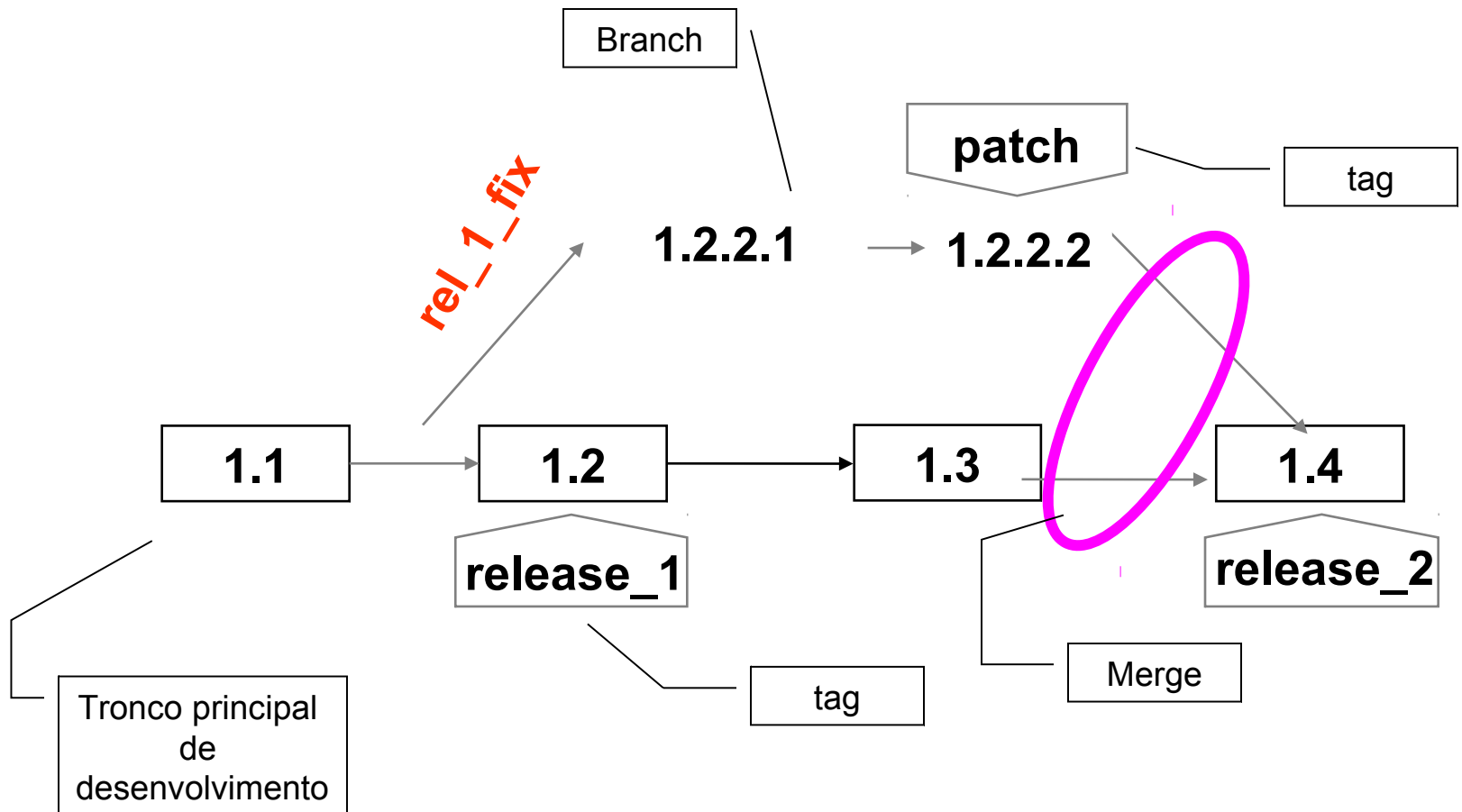


- ▶ Unificação de diferentes versões de um mesmo item de configuração
- ▶ Integração de itens de configuração de um branch com os itens de configuração do fluxo principal
- ▶ Check-out atualizando a área local
- ▶ Algumas ferramentas fornecem um mecanismo automático para realização de merges
 - Mesmo com o uso de ferramentas, em vários casos há necessidade de intervenção humana

Merge (exemplo)



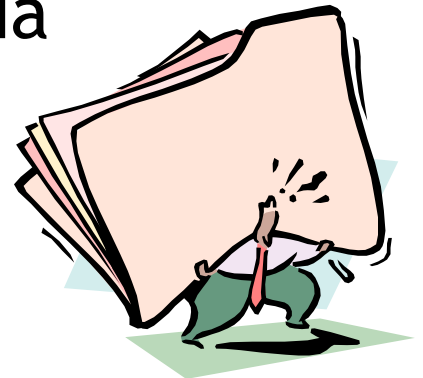
Branching e Merging



- ▶ Representa uma versão ainda incompleta do sistema em desenvolvimento, mas com certa estabilidade
- ▶ Costumam apresentar limitações conhecidas
- ▶ Espaço para integração de funcionalidades

Mais sobre build

- ▶ Incluem não só código fonte, mas documentação, arquivos de configuração, base de dados, etc.
- ▶ A política de geração dos builds deve ser bem definida na estruturação do ambiente
- ▶ A geração de builds deve ser automatizada e realizada com frequência adequada



- ▶ Versão do sistema validada após os diversos tipos de teste
- ▶ Produto de software
- ▶ Supostamente sem erros
- ▶ Entregue ao cliente ou ao mercado
- ▶ Processo iterativo/incremental produz, em geral, mais de um release

Mais sobre release

- ▶ Implantado no cliente
- ▶ Deve ser devidamente mantido
 - Enquanto a linha principal evolui
 - Uso de branches e merges

Oportunidades criadas com GC

▶ Reuso de itens de software

- Artefatos
- Componentes

▶ Automação de processo

- Construção de *builds*
- Geração de releases
- Testes
- Integração

Conclusões

- ▶ GC é um fluxo de apoio ao projeto como um todo
- ▶ Passos iniciais para a adoção de um processo de software
- ▶ Requer uma certa disciplina na manipulação de itens de configuração
- ▶ Apoio de ferramentas sempre que possível

Onde estão os slides da próxima aula?



Obtenha o SVN (<http://subversion.apache.org/>)

► Faça um checkout a partir do repositório:

```
svn checkout
```

```
http://subversion.assembla.com/svn/scm\_2012\_1/
```

