ソフトウェア開発演習I
- 開発リポジトリのマイニングと分析 (1) -

情報科学研究科
ソフトウェア設計学研究室
助教　崔恩瀞(Eunjong choi)
E-mail: choi[at]is.naist.jp

講義概要

- 履修条件
  - ✓ C言語やJava言語の基礎的な文法知識を有すること。プログラミング未経験者はプログラミング基礎演習等を事前に履修することが望ましい。
  - ✓ また、Emacsエディタや UNIXのコマンドラインツール（grep,find等）についての基本的な知識を有することが望ましい。
- 成績評価 (Grading)
  - ✓ 各テーマごとに分析レポートの提出を課す。レポートの分析内容により成績の評価を行う。

講義スケジュール

Schedule of this lecture

	日時	内容
1.	１０月　５日（木）４，５限	開発リポジトリのマイニングと分析
2.	１０月１２日（木）４，５限	開発リポジトリのマイニングと分析
3.	１０月１９日（木）４，５限	開発リポジトリのマイニングと分析
4.	１０月２６日（木）４，５限	開発リポジトリのマイニングと分析
		毎週演習課題を提出してください.
5.	１１月　２日（木）４，５限	コードリーディング
6.	１１月　９日（木）４，５限	コードリーディング
7.	１０月１６日（木）４，５限	コードリーディング
8.	１１月３０日（木）４，５限	コードリーディング

目次

- データマイニングの紹介
- ソフトウェアリポジトリの概要
- バージョン管理システムを対象としたリポジトリマイニング
  - ✓ どんなデータが蓄積されているのかを理解する
  - ✓ データの取得方法を学ぶ
  - ✓ 開発データの分析方法を学ぶ

データマイニングとは？

*“*Knowledge Discovery in Databases (**KDD**) *is the non-trivial process of identifying valid, novel potentially useful, and ultimately understandable patterns in the data”*

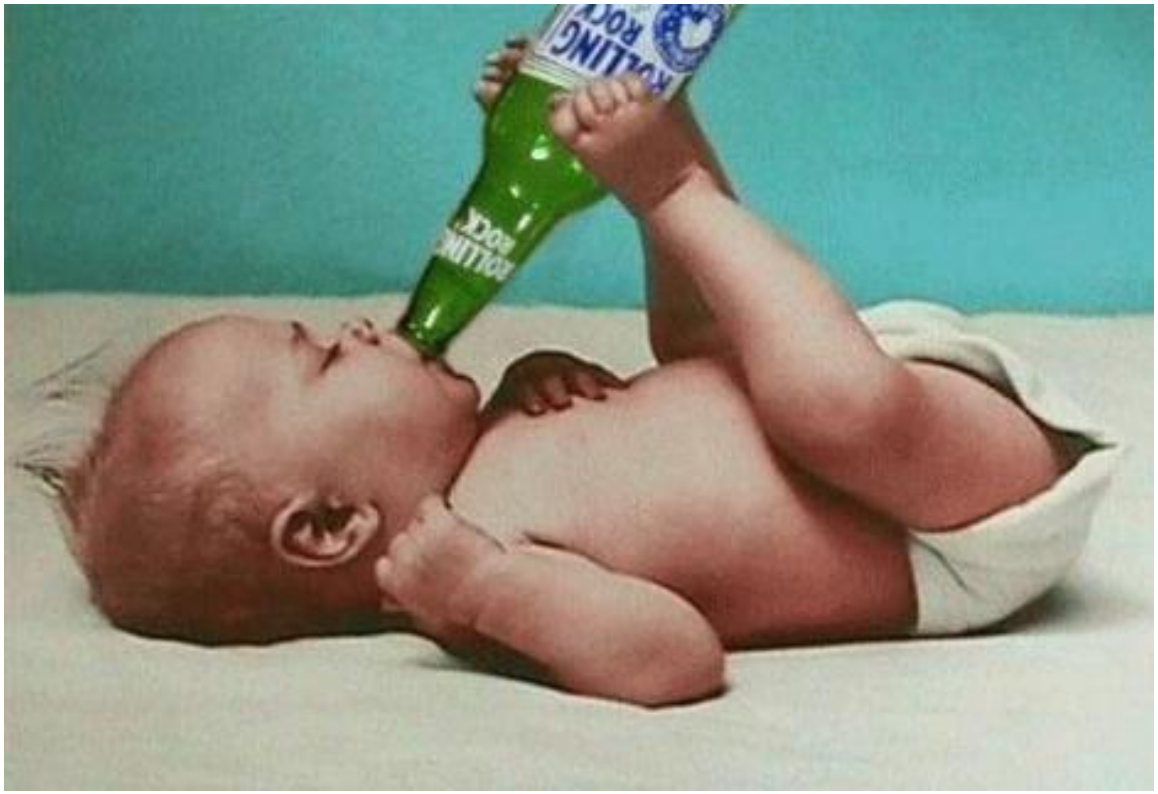
- Valid (たまたま見つかったのではなく，汎化能力)
- Novel (まだ我々が知らない新規なもの)
- Useful (有効であって実際に使うことができ)
- Understandable (我々に理解できる)

データマイニングの歴史

- 1990年代前半
  - ✓ KDDIの研究が推進される
    - ✓ HDやネットワークに蓄積される膨大データから何か有用な事柄を見つけられないか？
- 1990年代後半
  - ✓ はじめて公式に「データマイニング」という言葉が使われる.
    - ✓ ネットワークを含めより膨大なデータから何か有用な事柄を見つけられないか？
- 2000年代
  - ✓ インターネット上に蓄積されたデータが加速度的に増加

データマイニングの例

- 米国の大手スーパーマーケット・チェーンで販売データを分析した結果、顧客はおむつとビールを一緒に買う傾向があることが分かった



ソフトウェア開発データ

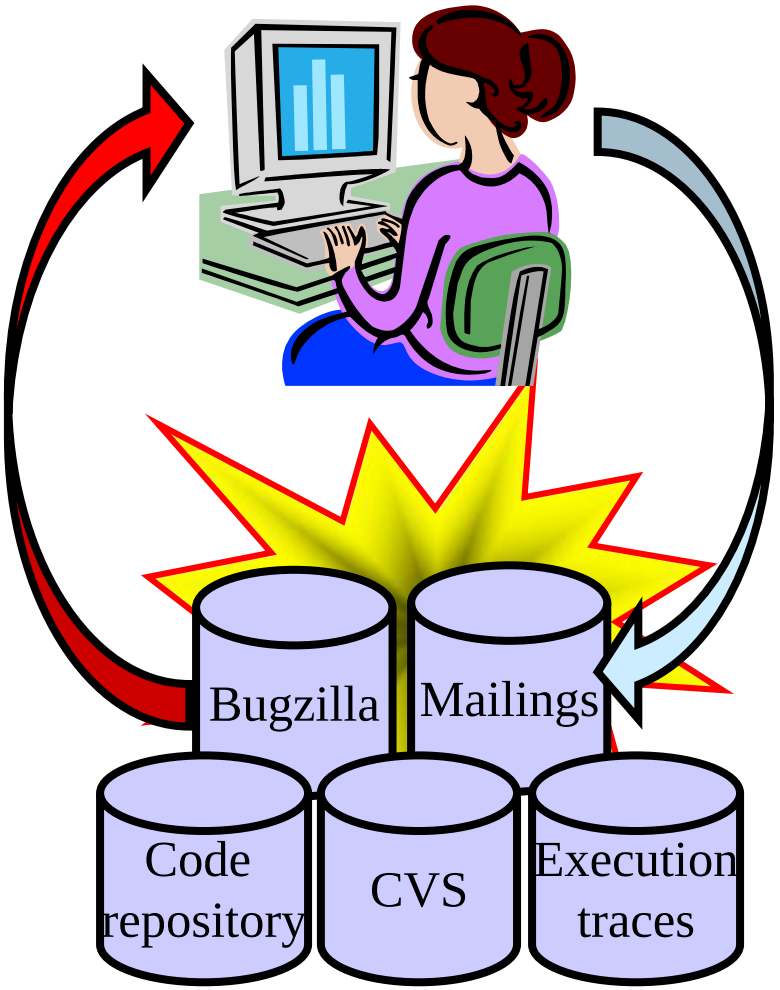
- ソフトウェア開発には膨大な記録(ビッグデータ)が残されている

- ✓ ソースコード
- ✓ ソフトウェアを実行した際の実行トレース
- ✓ コールグラフやプログラム依存グラフ
- ✓ Eメール，バグレポートなど

ソフトウェア開発データから何を知る？

勘や経験に依存しない知識

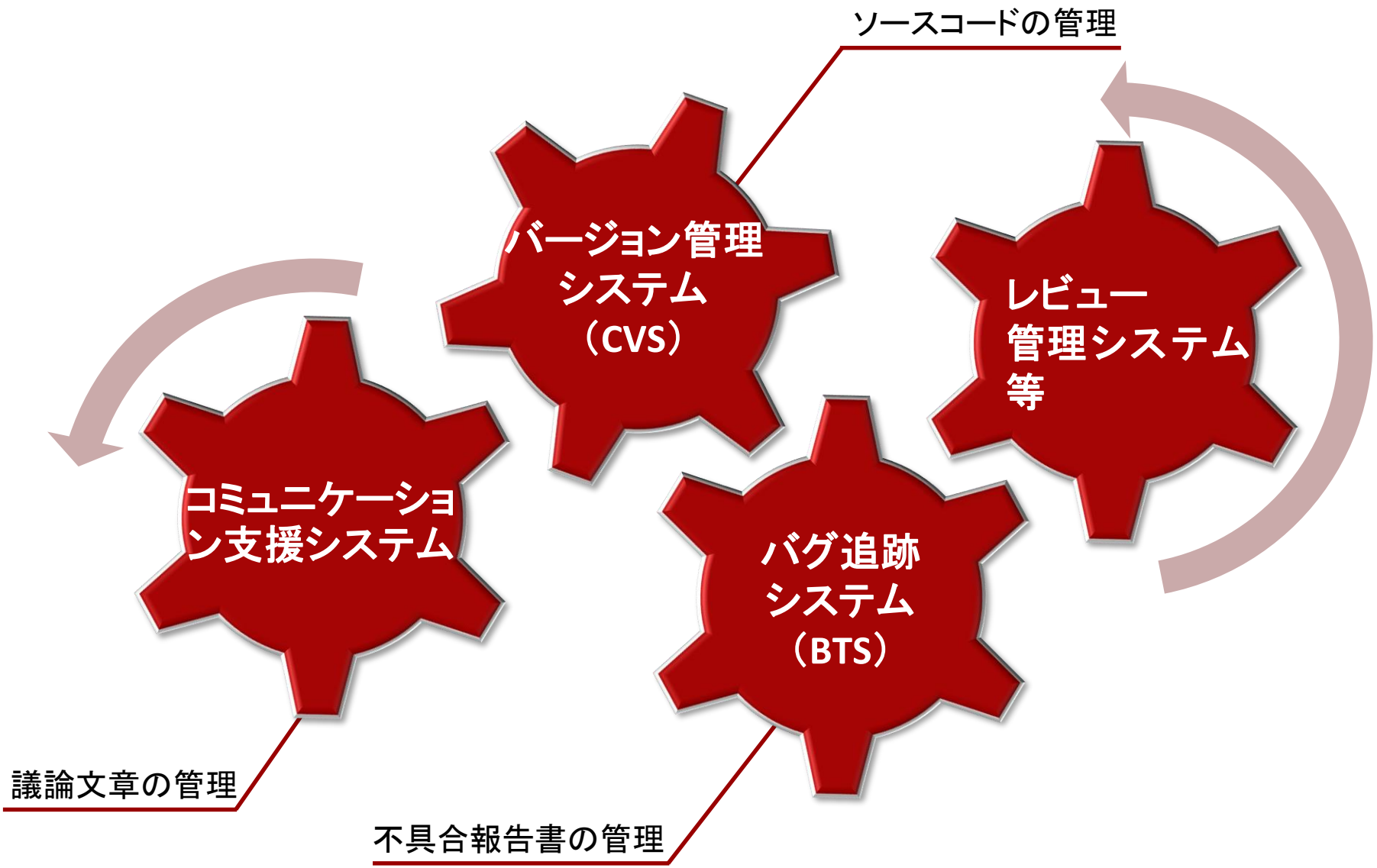
開発計画の立案



リポジトリに保存されている隠れたパターンや傾向を導き出す

リポジトリ
（コンテンツを集中管理するファイルの保管庫）

ソフトウェアリポジトリ

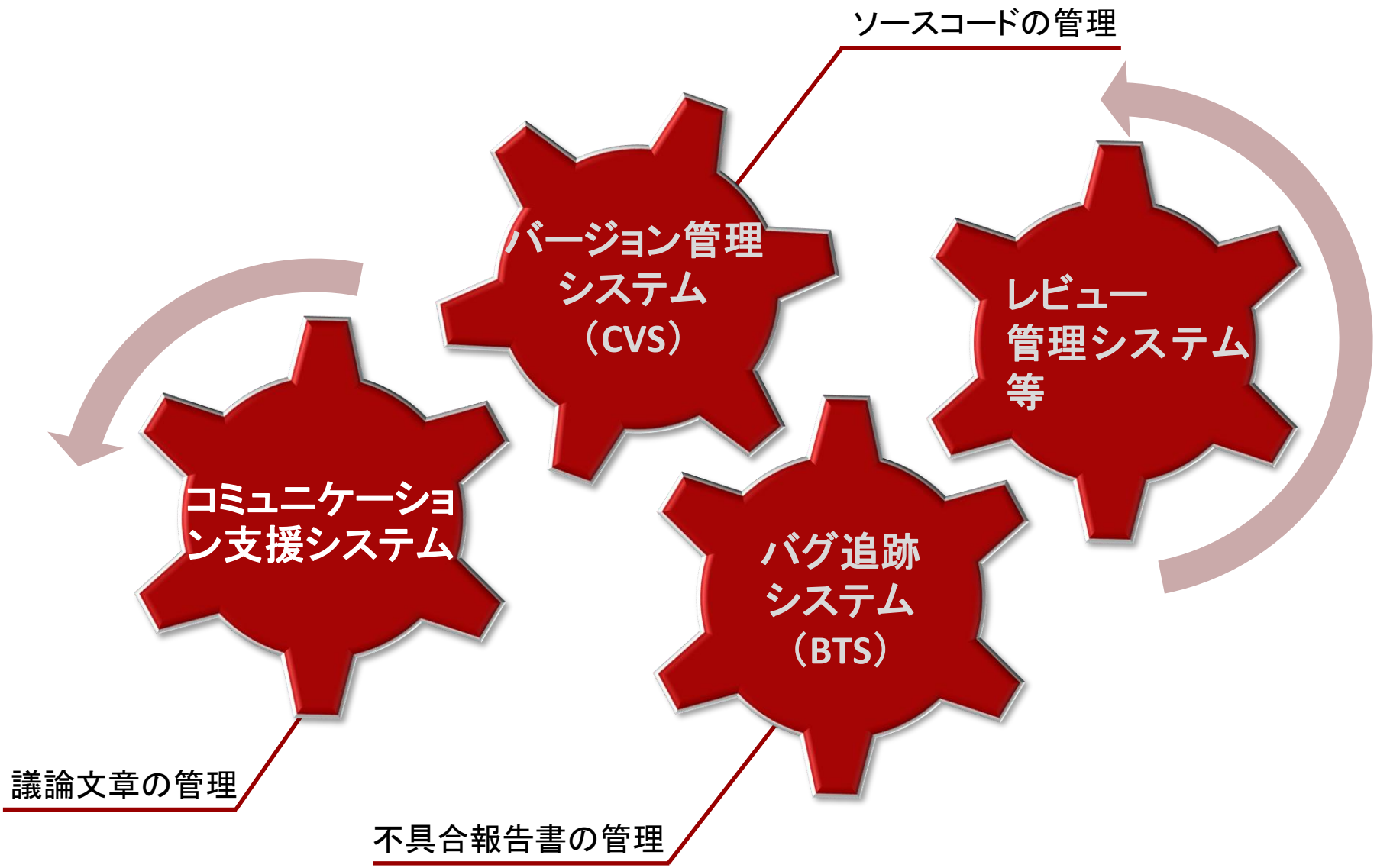


リポジトリマイニングを行う動機

- 大規模ソフトウェア開発では，開発履歴データを蓄積している
  - ✓ 膨大な開発履歴データの中には，**開発者が創意工夫した記録**が埋もれているのでは？

- 隠れたパターンや傾向を掘り起こす（mining）ことでソフトウェア開発を支援する

ソフトウェアリポジトリ



コミュニケーション支援システム

- 開発者間のコミュニケーションには様々な手段が利用される

- その内容は，大小さまざまなプロジェクト上の決定事項が含まれている
- あとから確認するために，保存・整理されている必要がある

コミュニケーション支援システムの例

- メール
  - ✓ メーリングリスト管理システム
    - ✓ 開発者間のメールのやりとりを用意にする
    - ✓ メーリングリストに流れたメールの保管・閲覧機能を提供するものも多い
  - ✓ メーリングリスト閲覧システム
    - ✓ 閲覧機能のみに特化したシステム
- Web
  - ✓ Wiki
    - ✓ 複数人がブラウザのみで更新できるWebページ
  - ✓ Blog・SNS
    - ✓ 日記や掲示板でコメントをつけて会話できるシステム
- このほかにも各種メッセンジャーサービス等々さまざまな手段が存在する

コミュニケーション支援システムのマイニングの例[Sharma2015]

- ソフトウェア開発に興味を持っている79,768名のtwitterユーザのtwitterデータを分析
  - ✓ 八か月(2012年9月～ 2013年4月)のC#, Java, Python, Ruby および Scalaに関するつぶやきを解析
- 以下のResearch Questionsに関して調査
  - ✓ RQ1: What are some hot software engineering related events in Twitter space?
  - ✓ RQ2: What are the categories of software engineering related events in Twitter space?
  - ✓ RQ3: How hot is each event category?

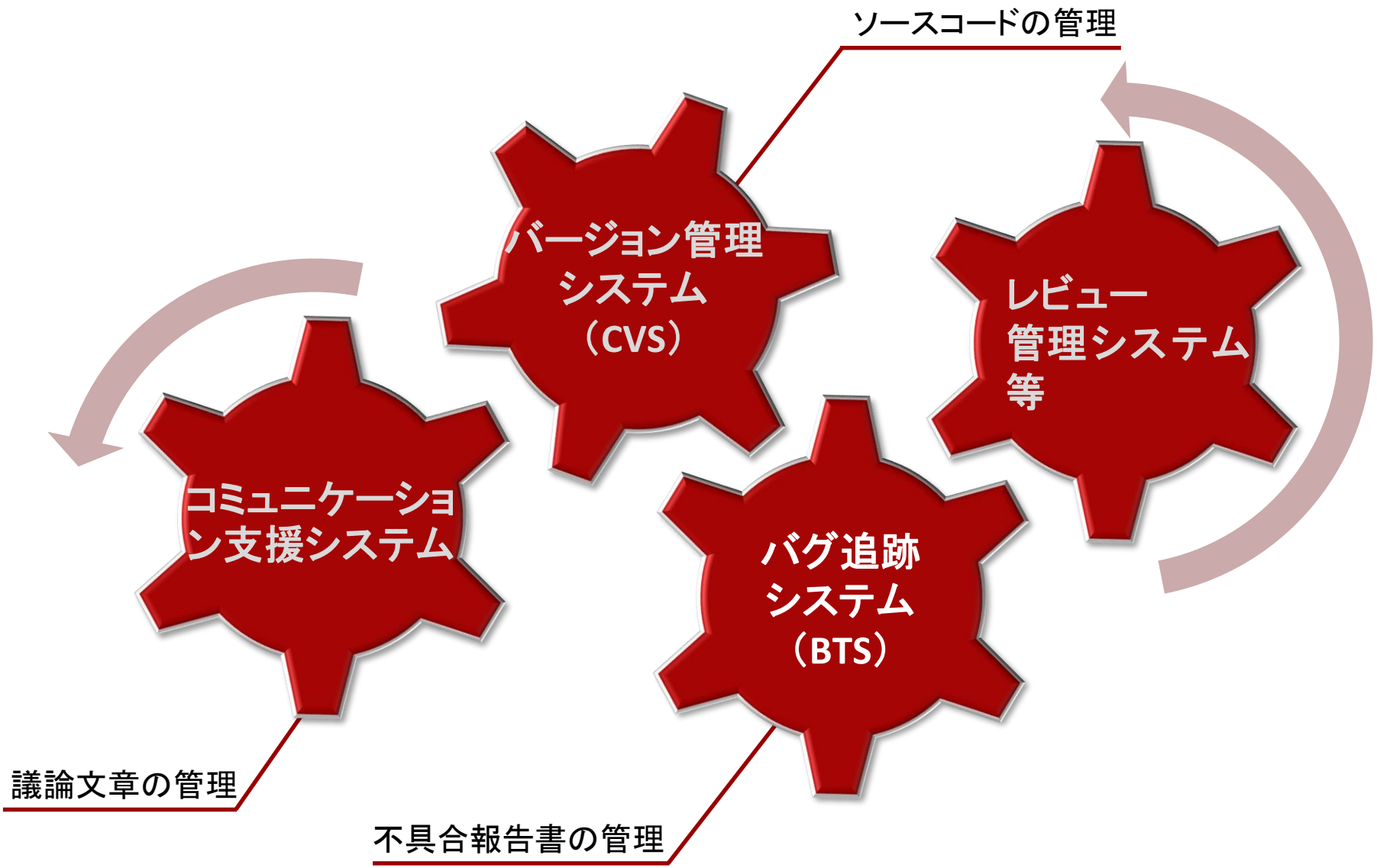
[Sharma2015] A. Sharma, Y. Tian and D. Lo, "What's hot in software engineering Twitter space?,"In Proc. Of *ICSME*,2015, pp. 541-545.

コミュニケーション支援システムのマイニングの例[Sharma2015]

Language	Event Description	Sample Tweet	Tweet Count
Java	Javaにおいてセキュリティの脆弱性が発見された	Feds warn PC users to disable Java	369
C#	何でC#がモバイル開発のための最高の言語か理由を書いたブログが投稿された	post by xamarin why c# is the best language for mobile development	181
Python	英企業が「Python」の商標をEUで出願した	Python trademark at risk in Europe: We need your help!	382
Ruby	Ruby 2.00がリリースされた	Ruby 2.0.0-p0 was released	842
Scala	Spring Frameworkの考案者であるRod Johnson氏がTypesafe社(founded by authors of Scala team)に入社した	Proud to welcome Rod Johnson (@springrod) to the Typesafe board:@typesafe#akka#scala#playframework	15

[Sharma2015] A. Sharma, Y. Tian and D. Lo, "What's hot in software engineering Twitter space?,"In Proc. Of *ICSME*,2015, pp. 541-545.

ソフトウェアリポジトリ

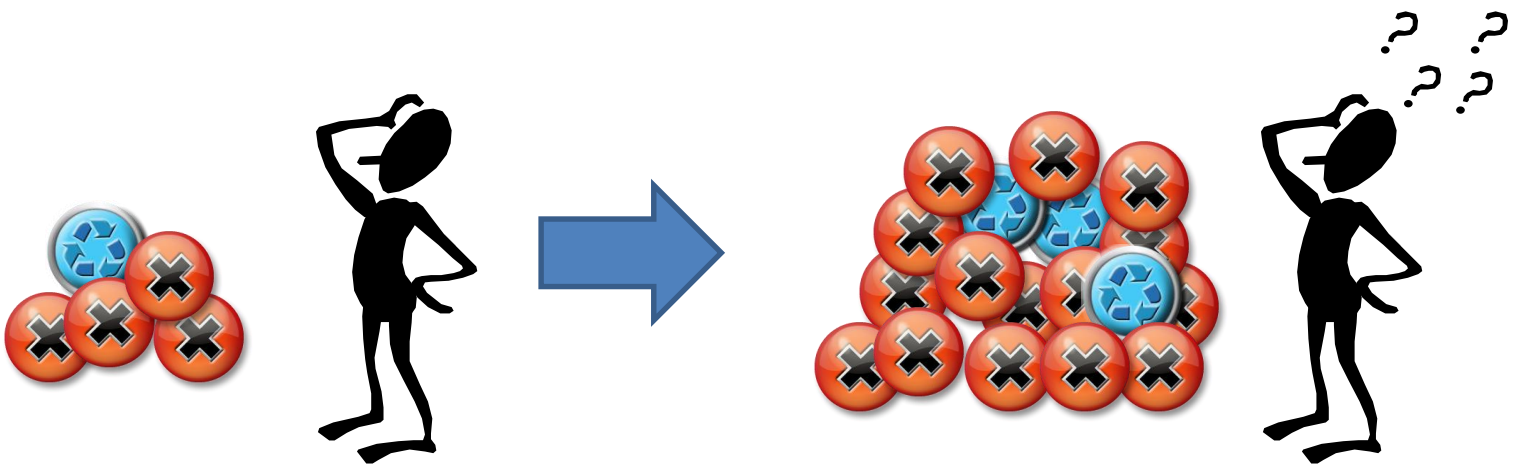


バグトラッキングシステム（BTS）

- ソフトウェア開発工程において，様々なバグ（セキュリティ，パフォーマンス，機能）が発生する
  - ✓ リリース前に修正し忘れたバグがあってはいけない
  - ✓ リリース後に発生したバグは対処漏れがあってはいけない
- バグを体系的に管理する必要がある

BTSはなぜ必要？

- 大規模開発ではバグが大量に発生する
  - ✓ 数百万行，数千万行規模の開発では，バグの記録が数万件に上ることも珍しくない
- 「誰が」どのバグを直している？直した？
- どのバグがすでに対処済み？



BTSに求められる機能

- バグを一元的に管理できること
  - ✓ すべてのバグが集約されていること
- 誰がどのバグを直している/直したのかが把握できること
- バグの対処履歴を把握できること
  - ✓ バグの対処時に，どのような判断をしたのかの記録が残っていること
- 様々なコミュニケーション手段との連携
- 大量のバグを扱えること
  - ✓ システムがスケーラビリティを備えていること
  - ✓ 大量のバグを捌けるインタフェースを備えていること

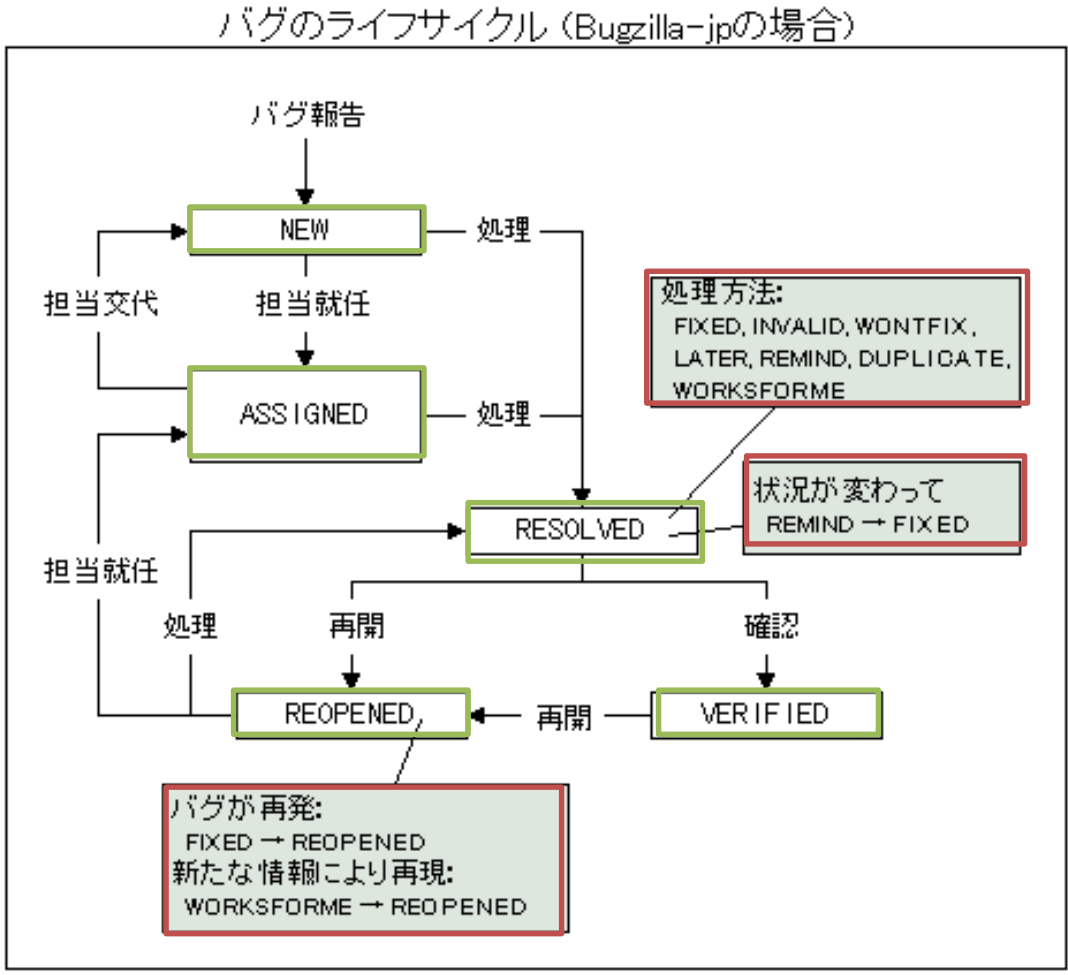
バグのライフサイクル

■ バグの状態は（主に）5つ

New, Assigned, Resolved,
Verified, Reopened

■ 処理方法（Resolved）

- Fixed（修正された）
- Invalid（仕様．バグでない）
- WontFix（修正しないと決定）
- Duplicate（他の報告と重複）
- WorksForMe（再現できない）

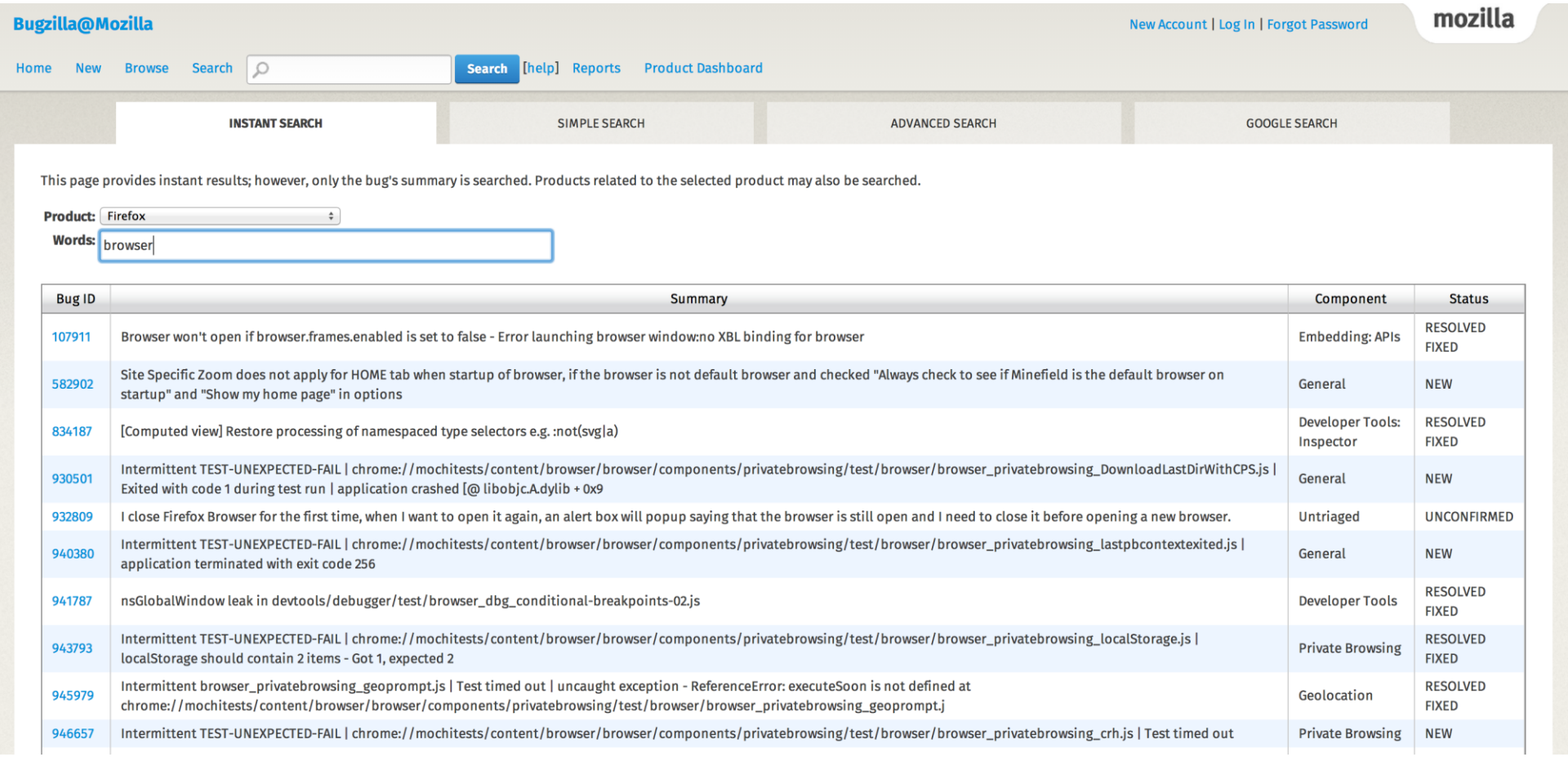


* http://www.mozilla.gr.jp/bugzilla/show_bug.cgi?id=3774**より抜粋**

代表的なBTS

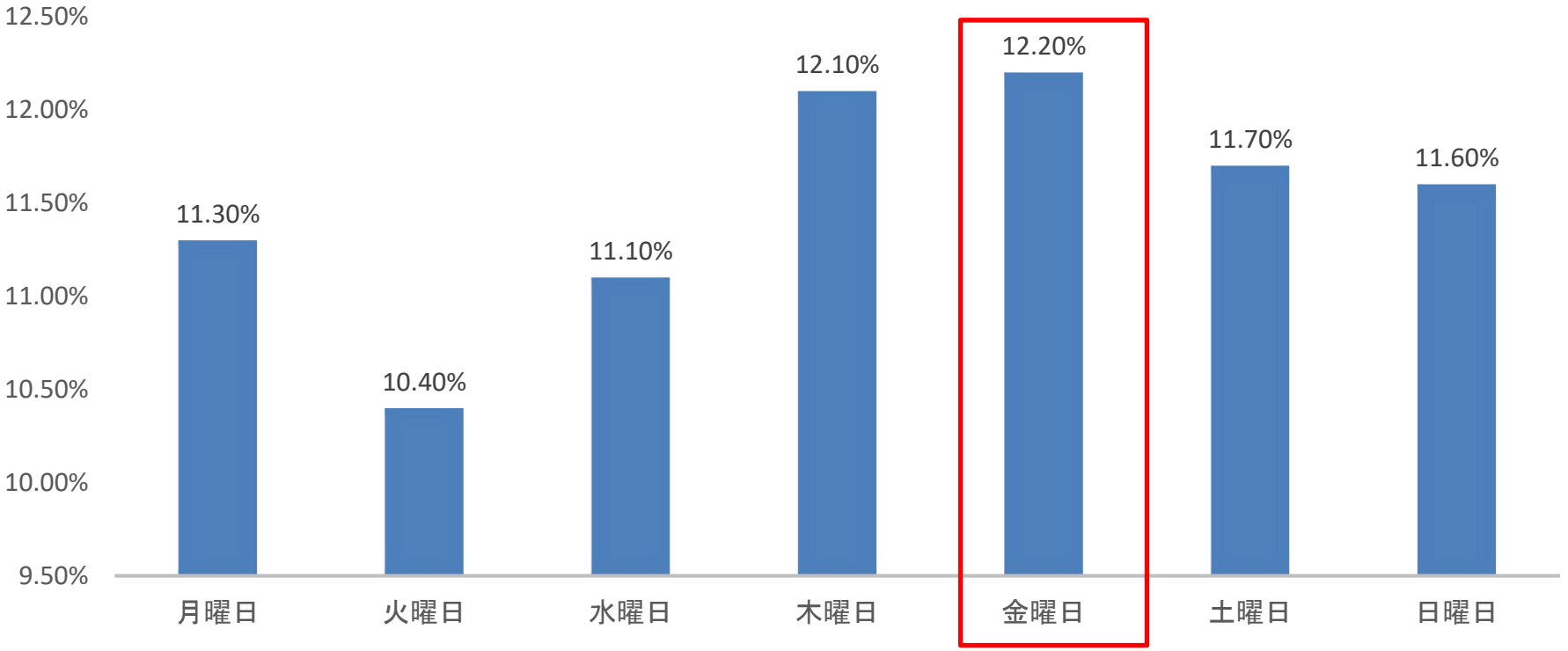
- Bugzilla
  ネットスケープでの開発のために作られたシステム
- mantis
  PHP で実装されたシステム
- 影舞
  日本で開発されたシステム．Ruby で実装されている
- Trac
  バグ管理システムのほか，Wiki や Subversion ブラウザが統合されている。pythonで実装。
- redMine
  一つのサイトで複数プロジェクトの管理が可能。ガントチャートの表示機能なども有する。Ruby On Railsで実装。

BTSのインタフェース（Bugzilla）



BTSのマイニングの例

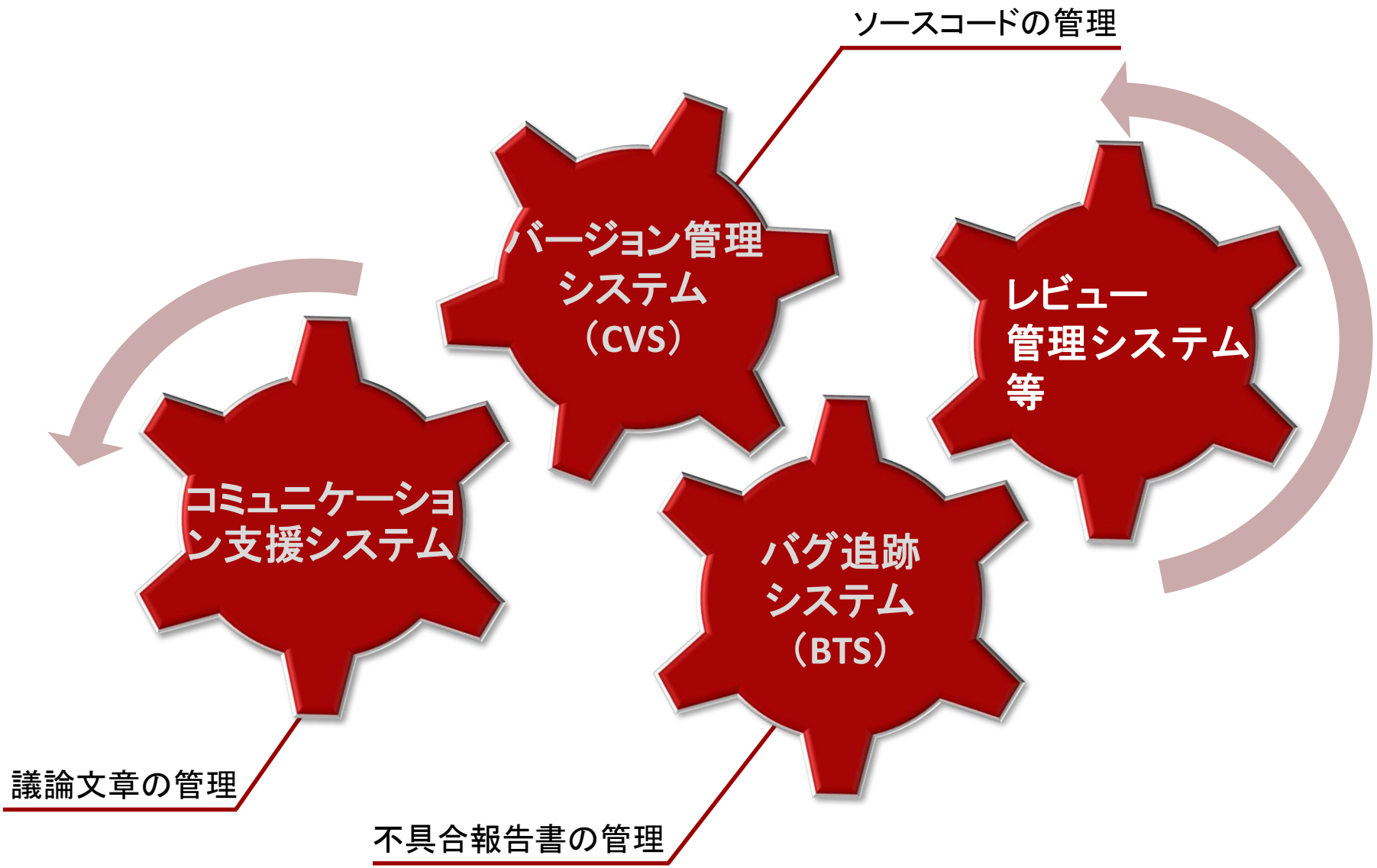
- バグを発生させるコミットを特定し，MOZILLAとECLIPSEプロジェクトを分析した



ECLIPSEプロジェクトのバグを発生させるコミットの割合

[Śliwerski2005] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?,"In Proc. Of *MSR*,2005, pp. 1-5.

ソフトウェアリポジトリ



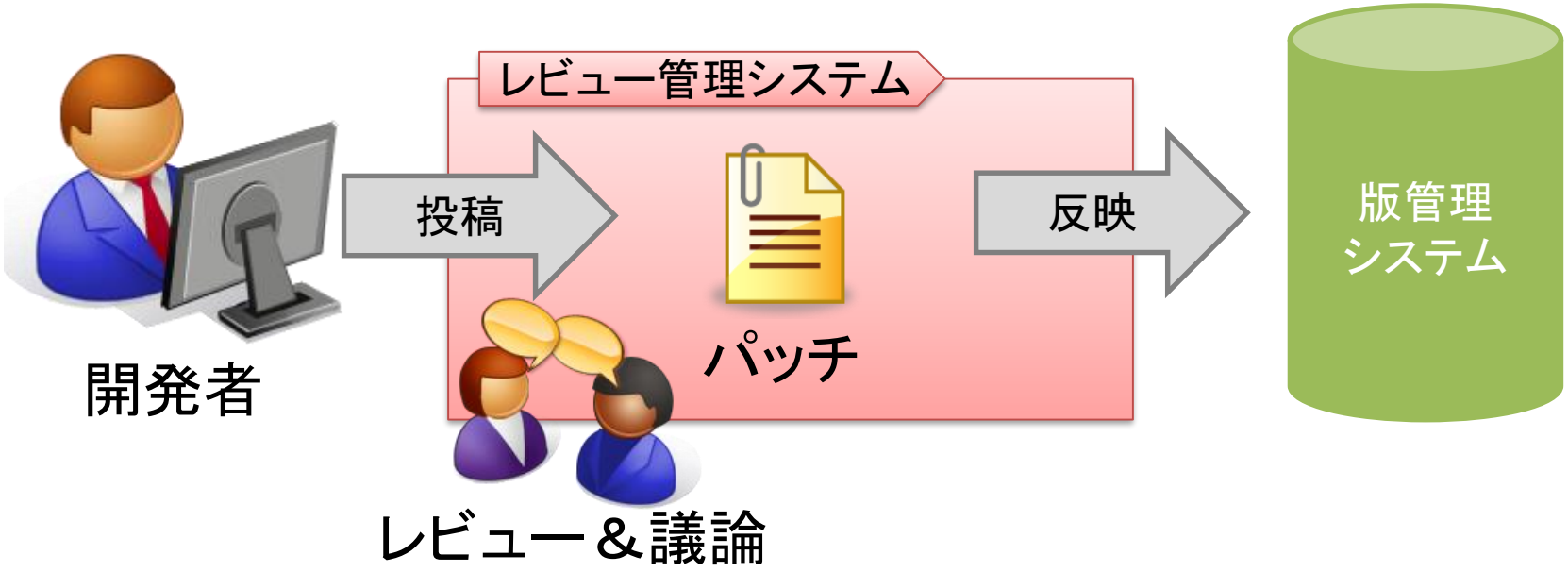
コードレビュー

- 設計文書やソフトウェアのソースコードを人が読み，問題がないかを検査する
  - ✓ プロセス設計の誤りやコードの記述ミス，コーディングルールの違反などを見つけることが可能
  - ✓ 欠陥のおよそ60%を検出可能[Boehm2001]

[Boehm2001] Boehm, B. and Basili, V. R.: Software Defect Reduction Top 10 List, *Computer*, Vol. 34, pp. 135-137 (2001).

レビュー管理システム

- 近年，OSSで広く利用されるようになった
- バージョン管理システムとの連携

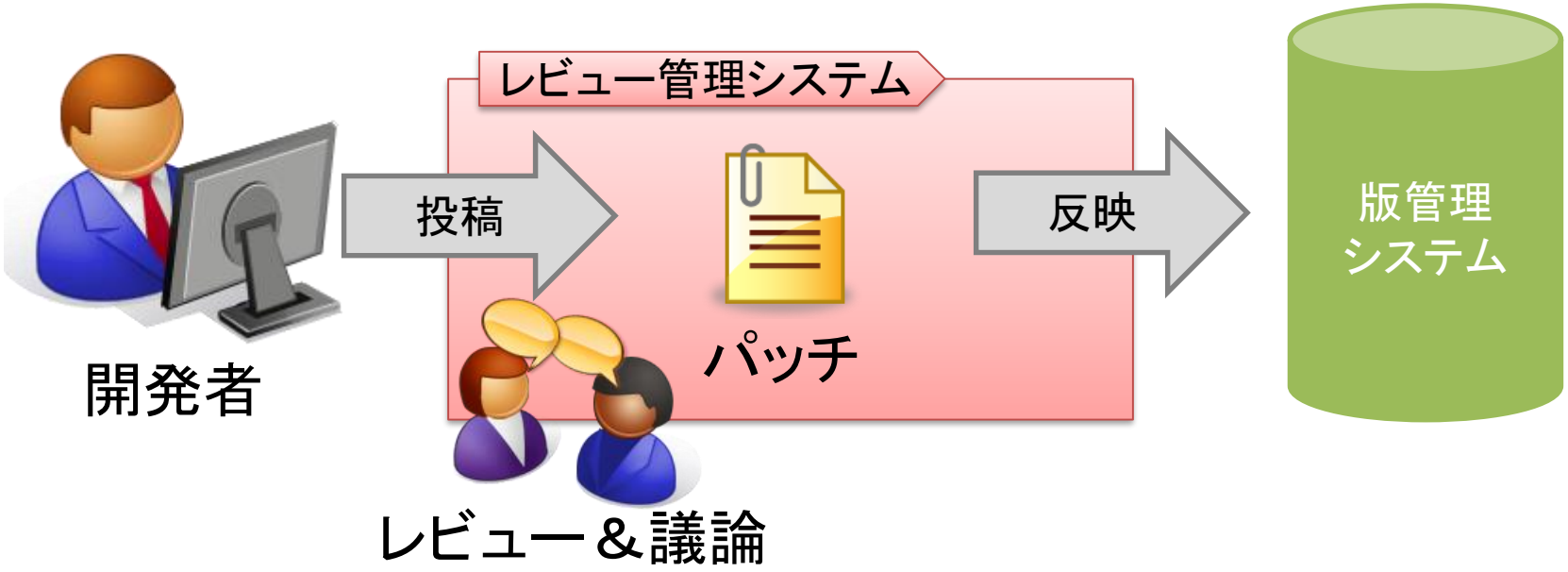


代表的なレビュー管理システム

- Reviewboard
  - ✓ Pythonで書かれたレビューツール，VMware社内で利用されている
- Phabricator
  - ✓ Facebook社でつくられた開発者向けのコミニュケーションツール
- Gerrit
  - ✓ Googleの社員が開発して、Androidの開発で使われているWebベースのコードレビューツール
- GitHub

レビュー管理システムGerrit

- レビューの追跡が可能
- レビューアの割り当てが可能
- ソースコードの変更箇所をわかりやすく表示
- レビューアによるインラインコメント



コードレビューのデータセット

- http://kin-y.github.io/miningReviewRepo/
  - ✓ OpenStack，LibreOffice，Android Open Source Project，Qt，Eclipseのレビューデータ

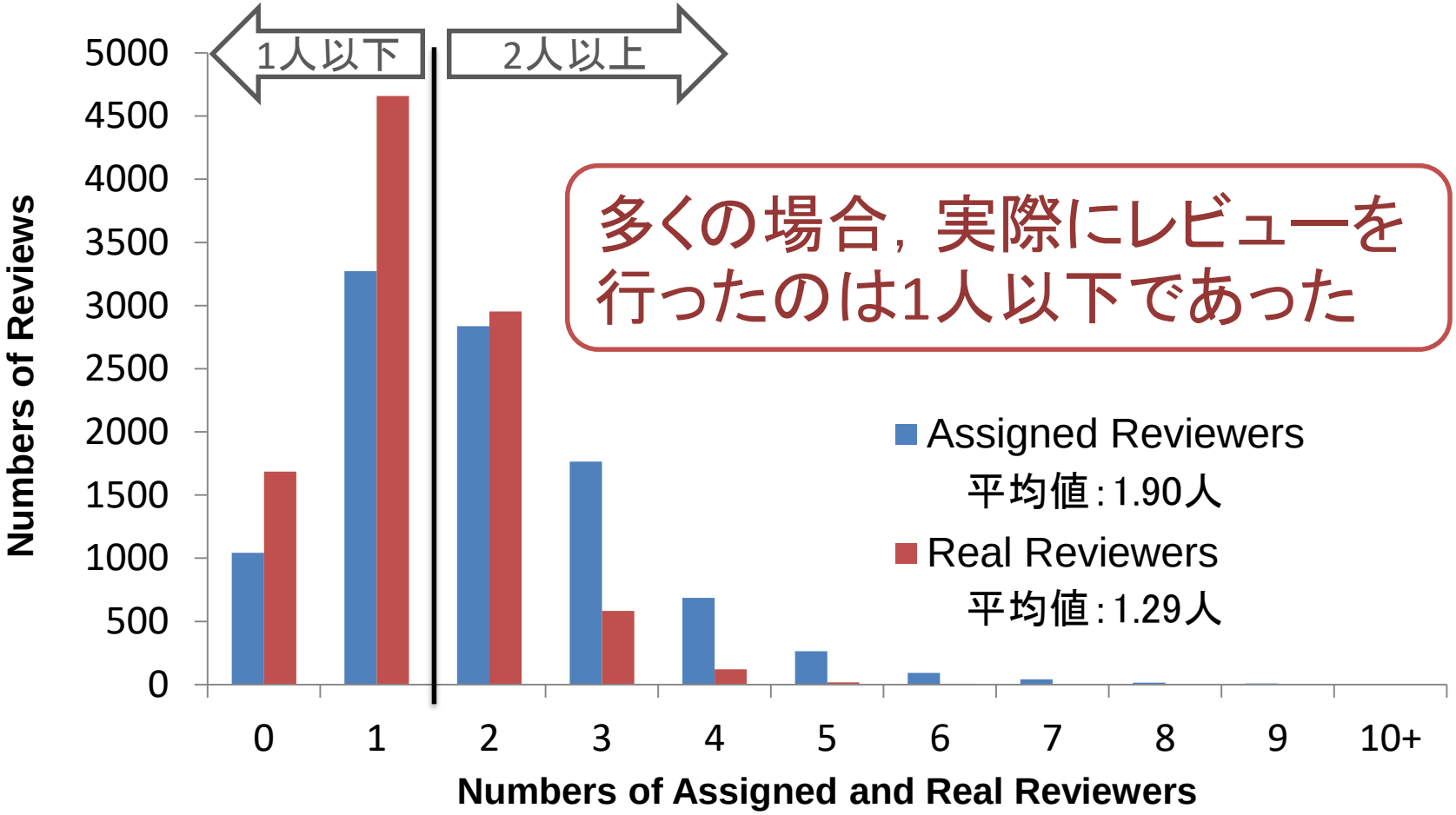
- http://sdlab.naist.jp/reviewmining/
  - ✓ Qt, Chromium Projectのコードレビューデータ

レビュー管理システムのマイニングの例[濵﨑ら]

- Android Open Source Project（AOSP）の2008年10月21日から2012年1月27日までの11,632件のレビューデータを分析

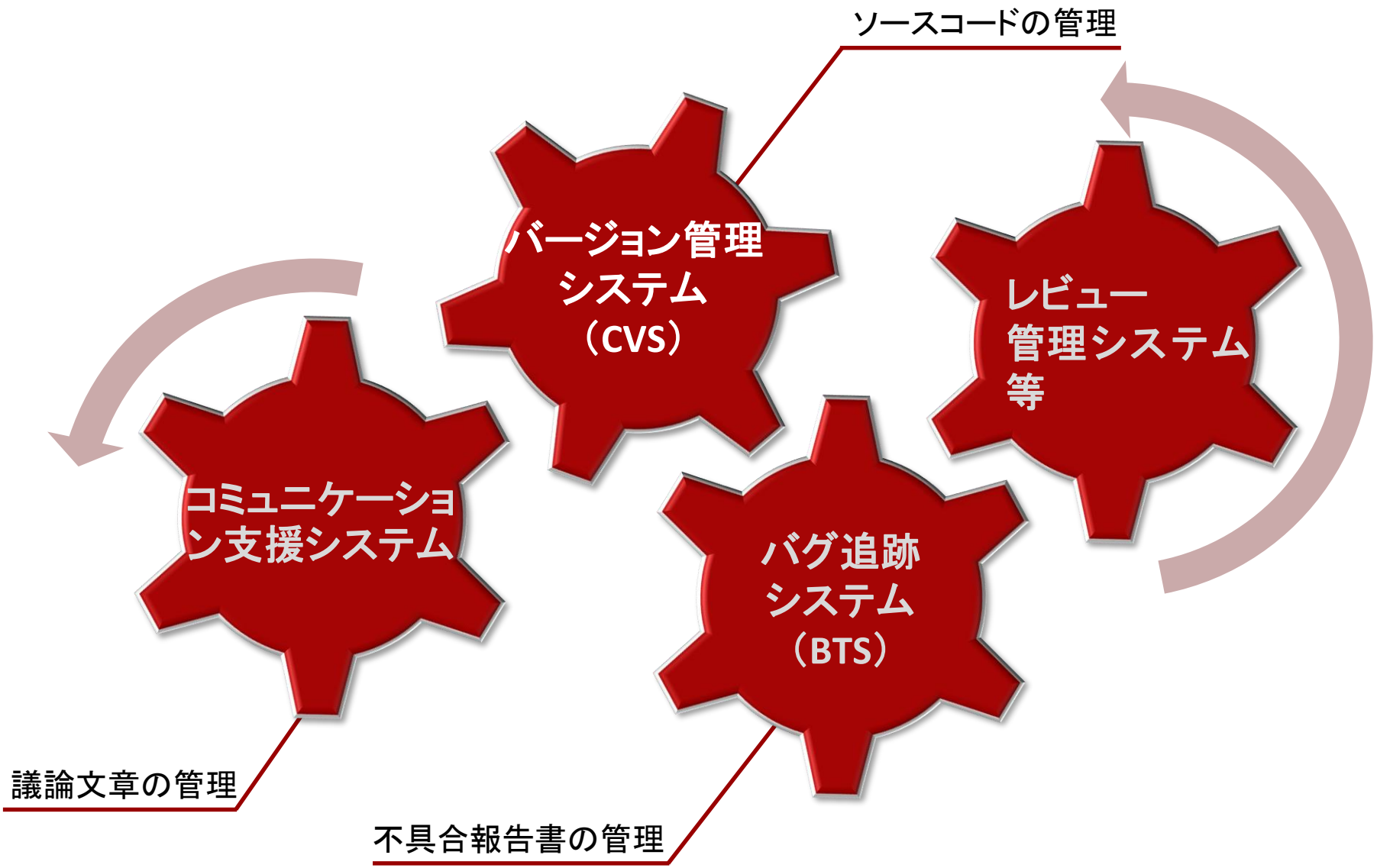
[濵﨑ら] 濵﨑一樹, 藤原賢二, 吉田則裕, RaulaGaikovinaKula, 伏田享平, 飯田元: "**Android Open Source Project**を対象としたパッチレビュー活動の調査", 情報処理学会研究報告, Vol.2012-SE-176/Vol.2012-EMB-25, No.12, pp.1-7, 早稲田大学, 2012年5月

レビュー管理システムのマイニングの例[濵﨑ら]



[濵﨑ら] 濵﨑一樹, 藤原賢二, 吉田則裕, RaulaGaikovinaKula, 伏田享平, 飯田元: "**Android Open Source Project**を対象としたパッチレビュー活動の調査", 情報処理学会研究報告, Vol.2012-SE-176/Vol.2012-EMB-25, No.12, pp.1-7, 早稲田大学, 2012年5月

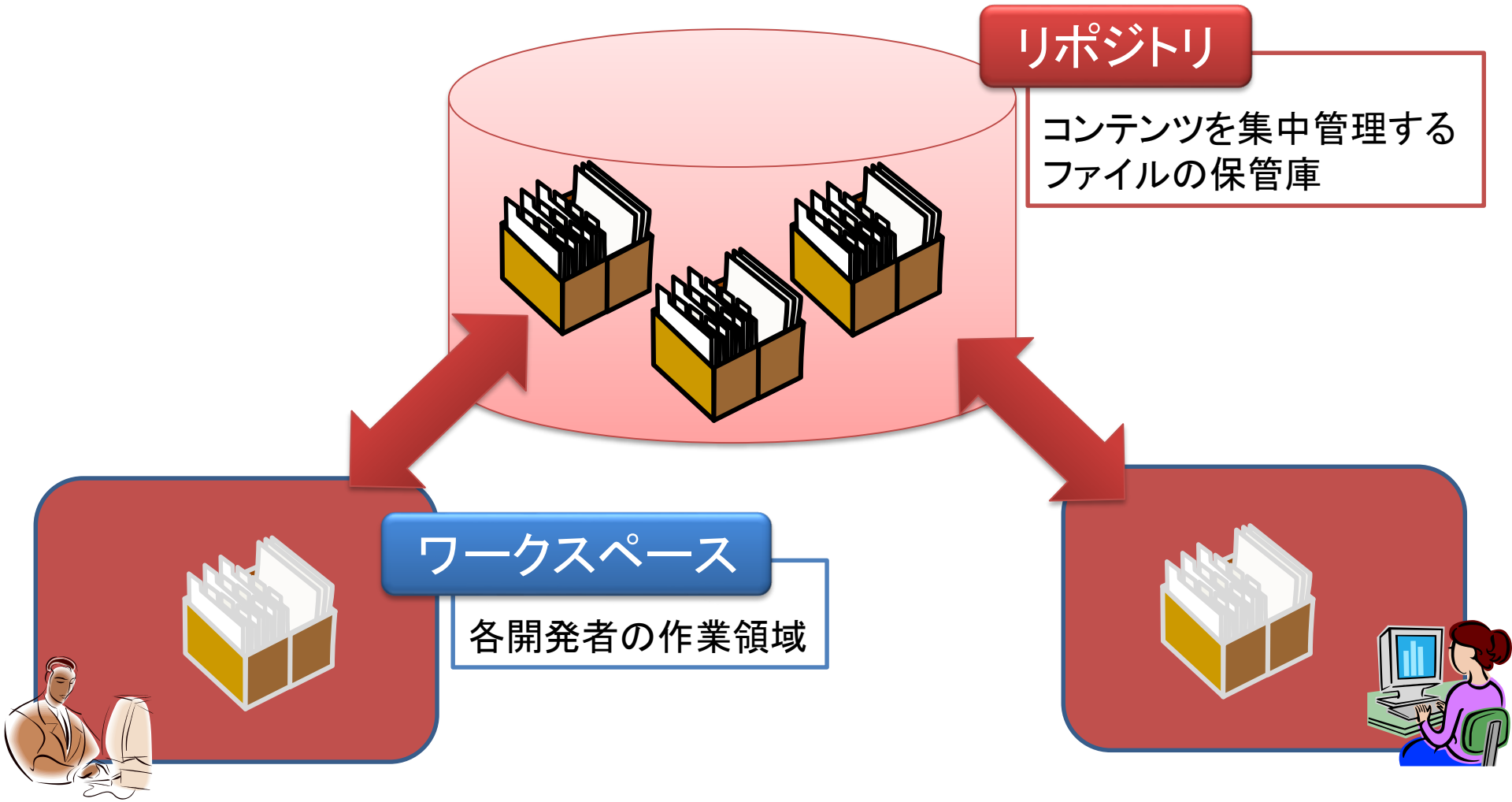
ソフトウェアリポジトリ



バージョン管理システム

- ソフトウェア開発における様々な成果物を保管するためのシステム
- 保管対象は計算機上の「ファイル」
  - ✓ ソースコード，各種ドキュメント（説明書や設計書など），画像や音声など
- 主な機能
  - ✓ ファイル共有・管理の支援
    - ✓ 論文の管理にも利用される ☺
  - ✓ 変更履歴の保存
  - ✓ 派生版の管理機能（ブランチング）

バージョン管理システムの基本的な概念

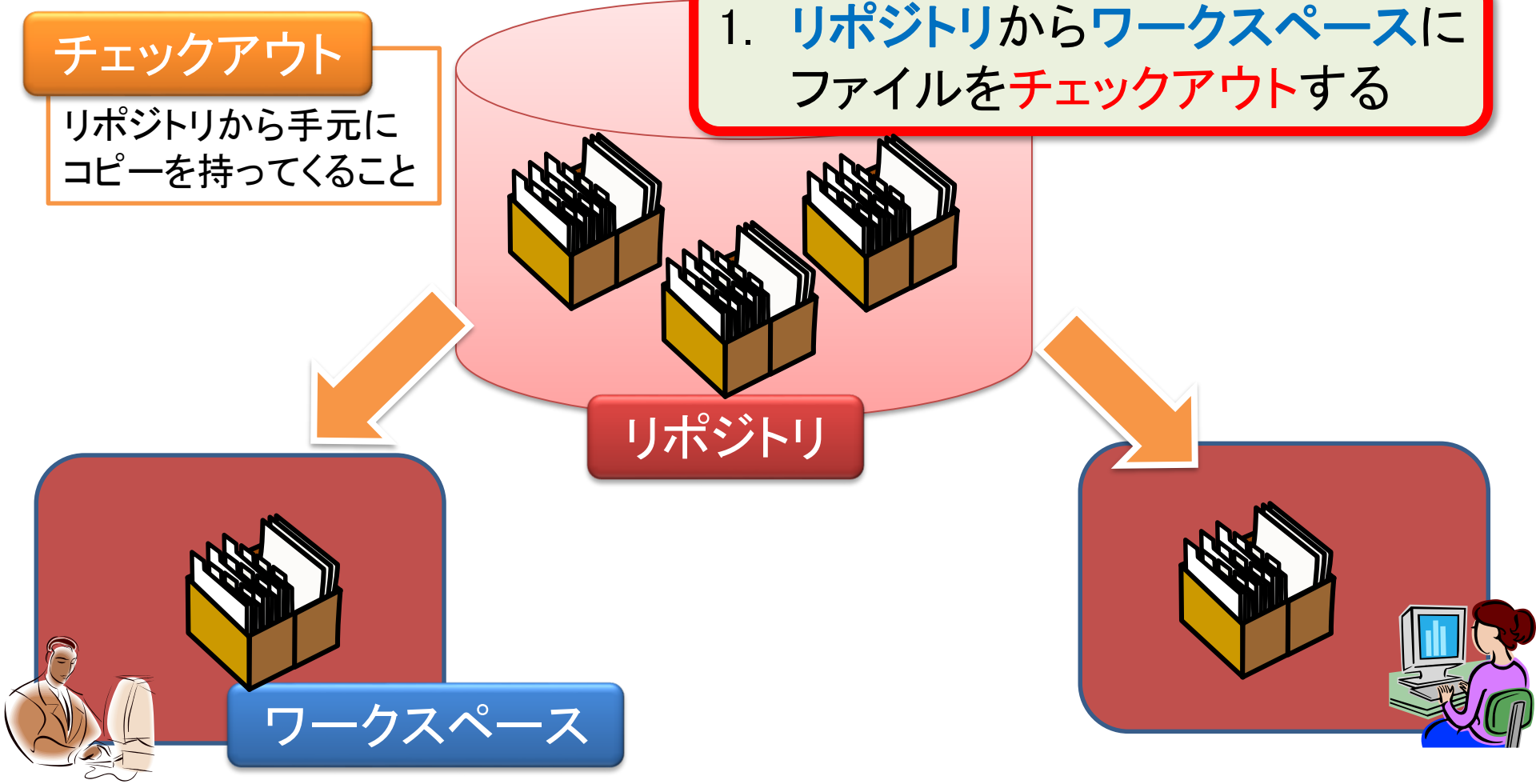


バージョン管理システム利用の流れ（作業開始時）

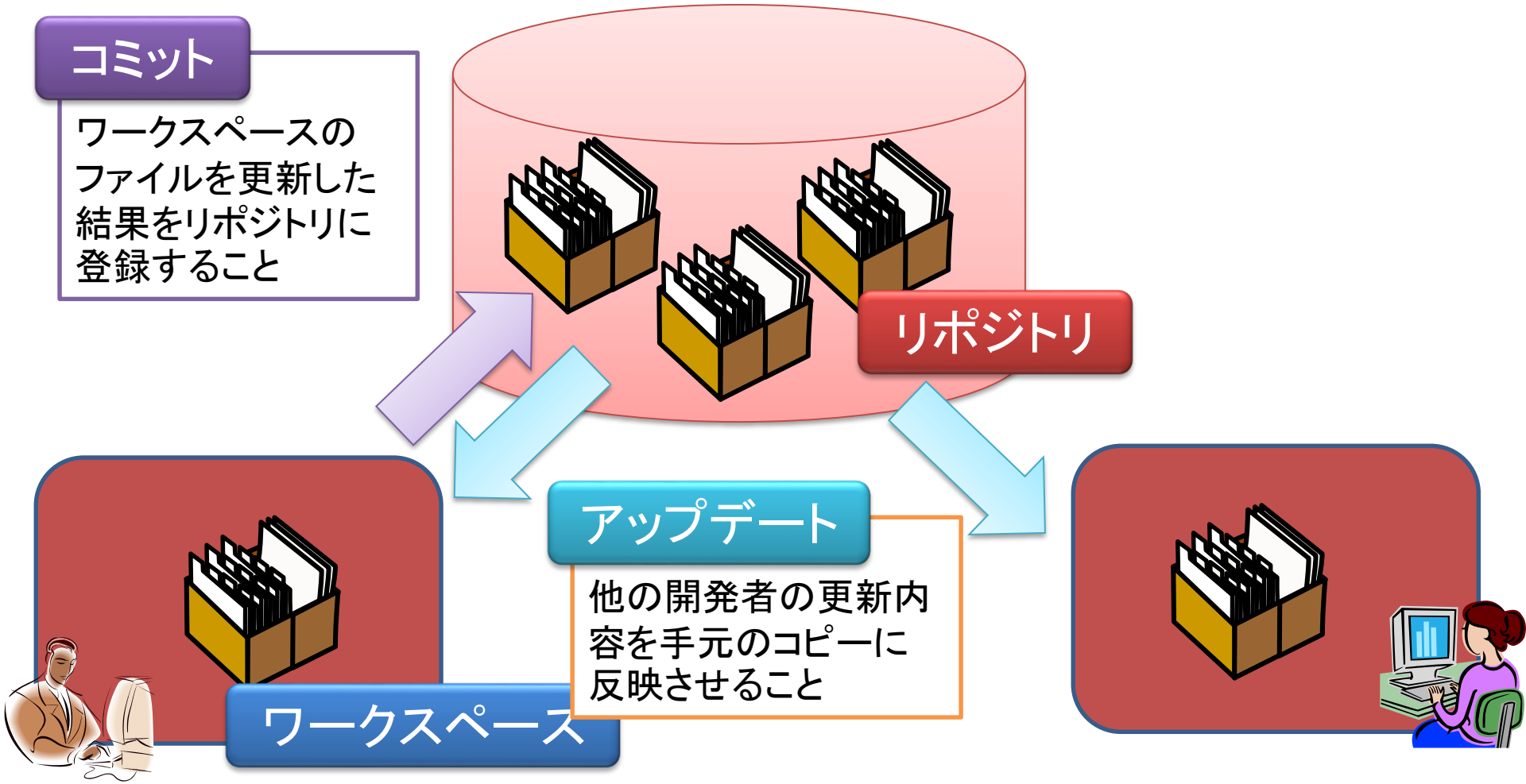
チェックアウト

リポジトリから手元にコピーを持ってくること

1. リポジトリからワークスペースにファイルをチェックアウトする



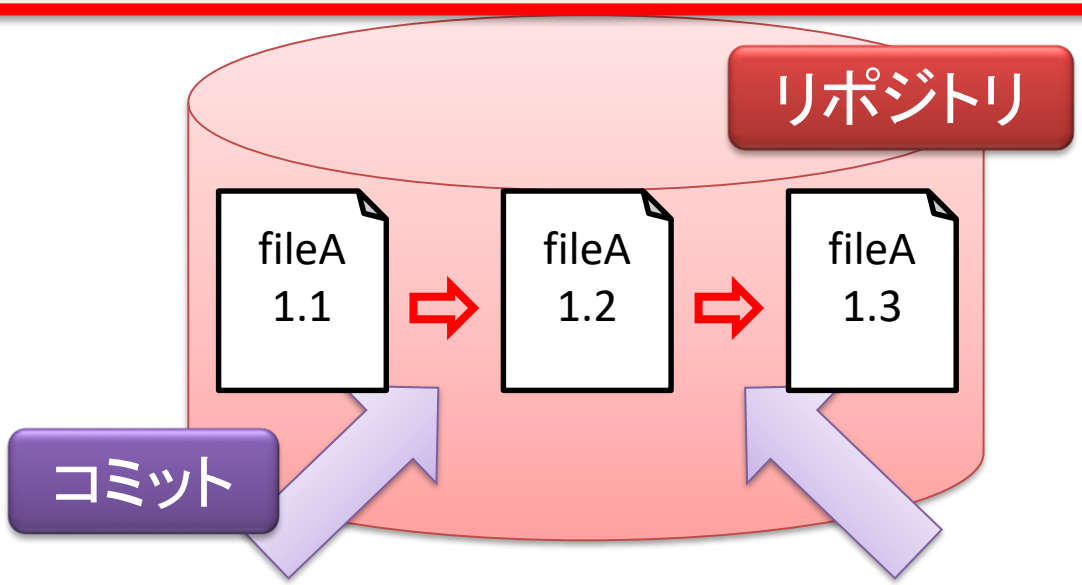
バージョン管理システム利用の流れ（日常の作業）



リビジョン番号

リポジトリは過去の変遷をすべて保存している

リビジョン番号
リポジトリに保存されているプロダクトを一意に特定するID
コミットのたびに新しいリビジョン番号が振られる



主なバージョン管理システム

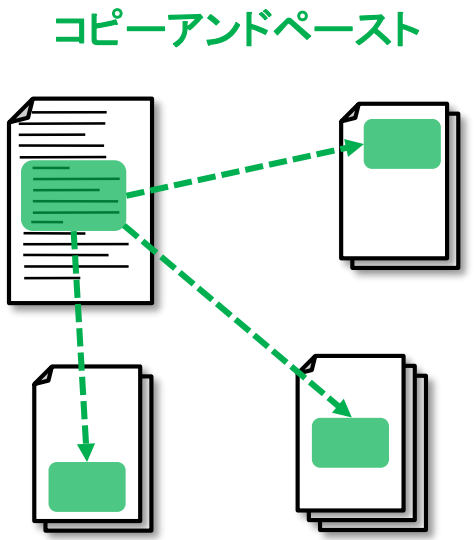
- オープンソース
  - ✓ RCS
  - ✓ CVS
  - ✓ Subversion
  - ✓ Mercurial
  - ✓ Git
- 商用ツール
  - ✓ Visual Source Safe (Microsoft)
    - Visual Studio との親和性から，企業においてよく利用される
  - ✓ Bit Keeper (BitMover Inc.)
    - Linus カーネルの管理に使われていた
  - Rational ClearCase (IBM)

バージョン管理システムのマイニングの例[中山ら]

- コードクローン †との位置関係がコード片の欠陥発生傾向に与える影響をOSS(5プロジェクト)を対象に調査

† コードクローンとは？

- 同一・類似した部分を持つコード片
- ソースコードのコピー&ペーストなどによって発生
- ソフトウェアの保守コストを大きくする要因

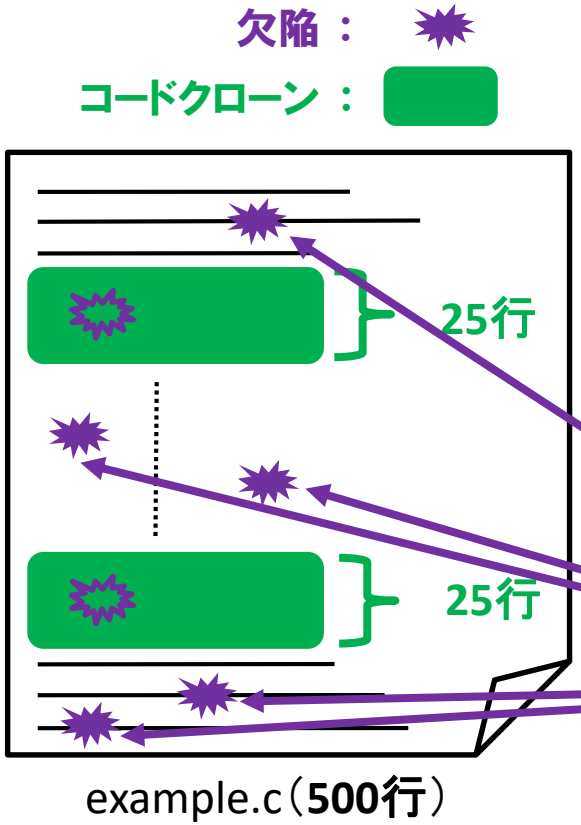


[中山ら]中山 直輝, 吉田 則裕, 藤原 賢二, 飯田 元, 高田 光隆, 高田 広章, "コードクローンとの位置関係に基づく欠陥発生傾向の調査," 情報処理学会論文誌, volume 57, number 2, pages 681-693 2016年2月

バージョン管理システムのマイニングの例[中山ら]

コードクローン内外の**欠陥率**をそれぞれ計算し，
統計的有意差が存在するかどうかを検定

＜欠陥率の計算方法＞



※コードクローン内における欠陥率

$$= \frac{\text{（コードクローン内に存在する欠陥のLOC）}}{\text{（コードクローン内のLOC）}}$$

$$= \frac{2}{25 + 25} = 0.04$$

※コードクローン外における欠陥率

$$= \frac{\text{（コードクローン外に存在する欠陥のLOC）}}{\text{（コードクローン外のLOC）}}$$

$$= \frac{5}{500 - (25 + 25)} = 0.01$$

バージョン管理システムのマイニングの例[中山ら]

検定結果

プロジェクト名	中央値（内）	中央値（外）	p値（Wilcoxonの順位和検定）	有意差
Gimp	0.00141	0.00549	4.71e-06	あり
Evolution	0	0.00475	< 2.20e-16	あり
Nautilus	0	0.00420	< 2.20e-16	あり
httpd	0	0.00629	< 2.20e-16	あり
Gedit	0.00126	0.00427	1.84e-01	なし

欠陥率を比較した結果：

コードクローン内　＜　コードクローン外

Git を使った演習

Gitの特徴

- Linuxのソースコードを管理するために開発されたが，AndroidやRubyをはじめ多数のプロジェクトで採用
- たくさんのホスティングサービス
  GitHub，BitBucketなど
- ブランチの管理が楽
  マージ作業の前に戻したり，マージをやり直したりできる
- ローカルリポジトリとリモートリポジトリ
  オフラインで作業できる

gitを始める前に・・・

今回は，/user/binにあるgitを使いますので，
以下のコマンドを入力しパスを設定してください．

```
set path=(/usr/bin $path)
```

（パスを設定しないと，古いバージョンのgitがデフォルトで
動くようになっていて，正しく動作しません．）

自分で作ったファイルをgitで管理しよう！

- Gitリポジトリを作成

```
> mkdir local
> cd local
> git init
```

git initとは？
Gitリポジトリを作成するコマンド．当該ディレクトリ（ここではlocalフォルダ）でバージョン管理を始めることになる．

ファイルのコミット

- ファイル（ディレクトリ）をインデックス（コミット候補）に登録

```
> git add <file/directry name>
```

- インデックスに追加されたファイルをコミット

```
> git commit -m “<comment>”
```

コメントに何のためのコミットか，どのような変更を加えたか等を記述

ログの確認

- これまでどのようなコミットがされてきたかを確認

```
> git log
Commit   12345de…                                  #コミットID
Author:    Norihiko Kawai <norihi-k@....>          #Author
Date  Thu Nov 4 15:01:42 2013 +0900 #日付

                fix Bug                    #コミットメッセージ

Commit   abc12345…                                 #コミットID
Author:    Hajimu Iida <iida@....>                 #Author
Date  Thu Nov 1 ….
```

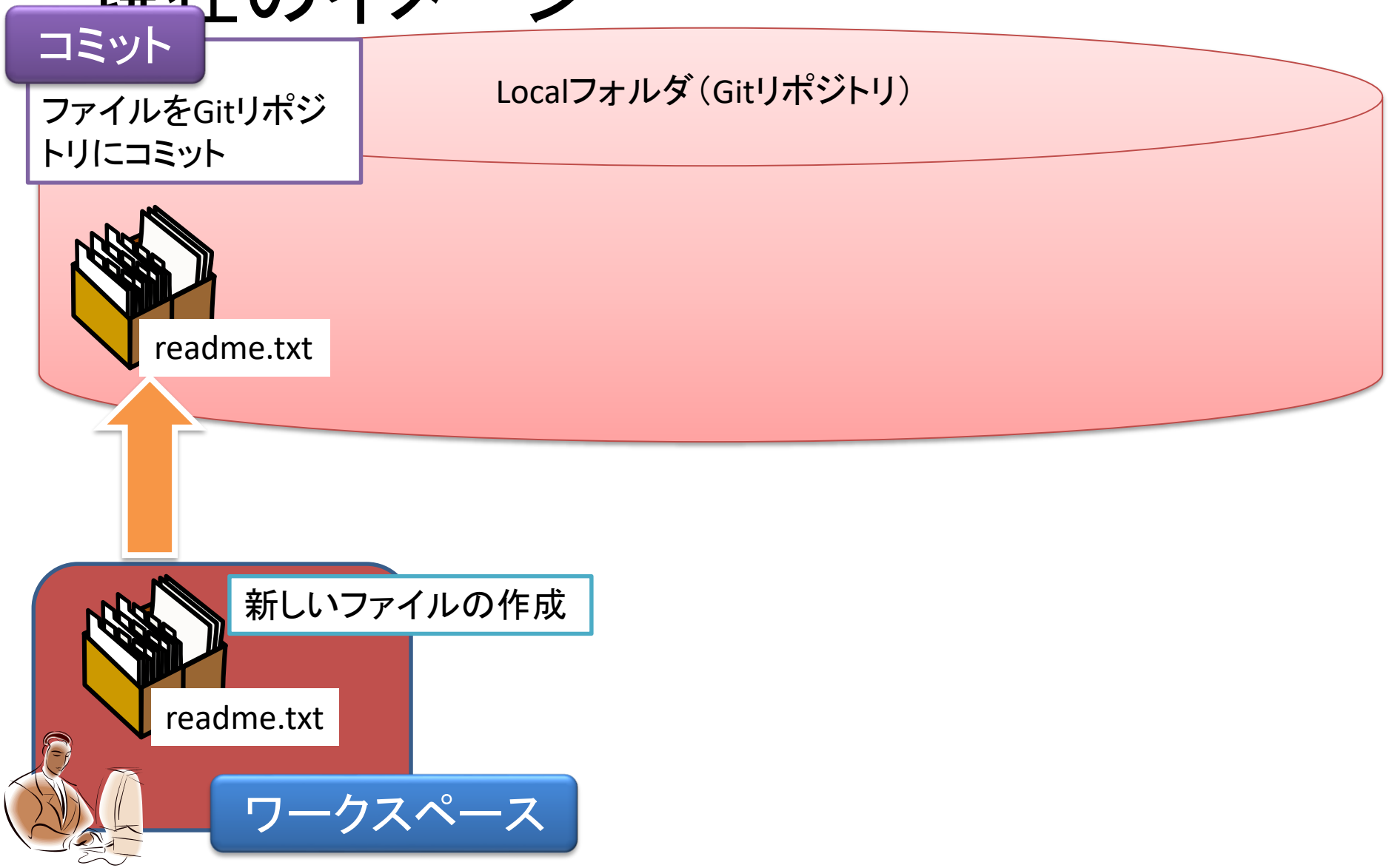
演習

1. gitリポジトリにgit addの説明を書いたテキストファイルを作成し，保存する.

2. 保存したプログラムをコミットする．その時，コミットメッセージも記録すること.

3. “git log”コマンドでファイルがコミットされているか確認

現在のイメージ


コミット
ファイルをGitリポジトリにコミット
Localフォルダ（Gitリポジトリ）
readme.txt
新しいファイルの作成
readme.txt
ワークスペース

演習

1. 先ほど保存したファイルにgit commitの説明を追加し，コミットする．そのとき，どのように変更したのかメッセージを残す．

2. 再度“git log”コマンドでファイルがコミットされているか確認

現在のイメージ

コミット

ファイルをGitリポジトリにコミット

Localフォルダ（Gitリポジトリ）

master

Readme.txt

Readme.txt

masterとは？
リポジトリにファイル群をコミットされ，変更される一系列を”master”と呼ぶ．

アップデート

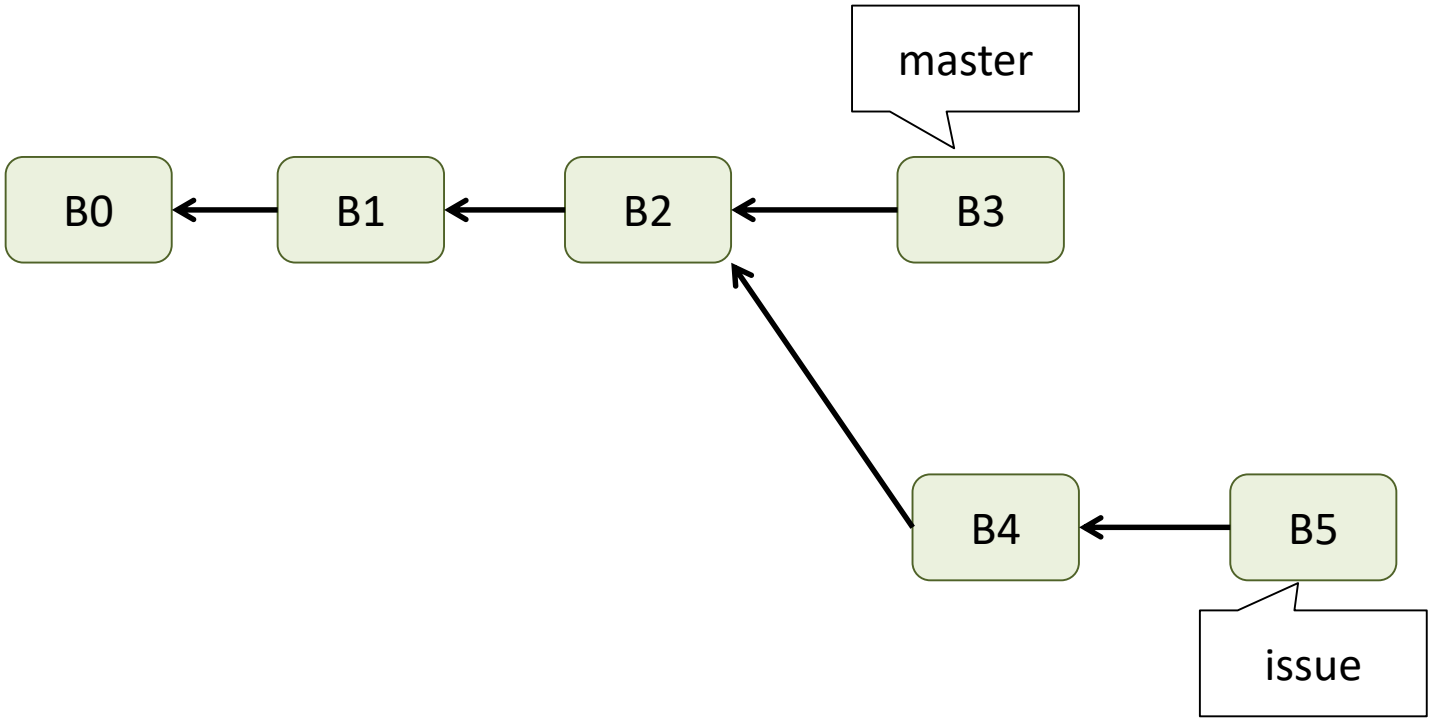
ファイルのアップデート

Readme.txt

ワークスペース

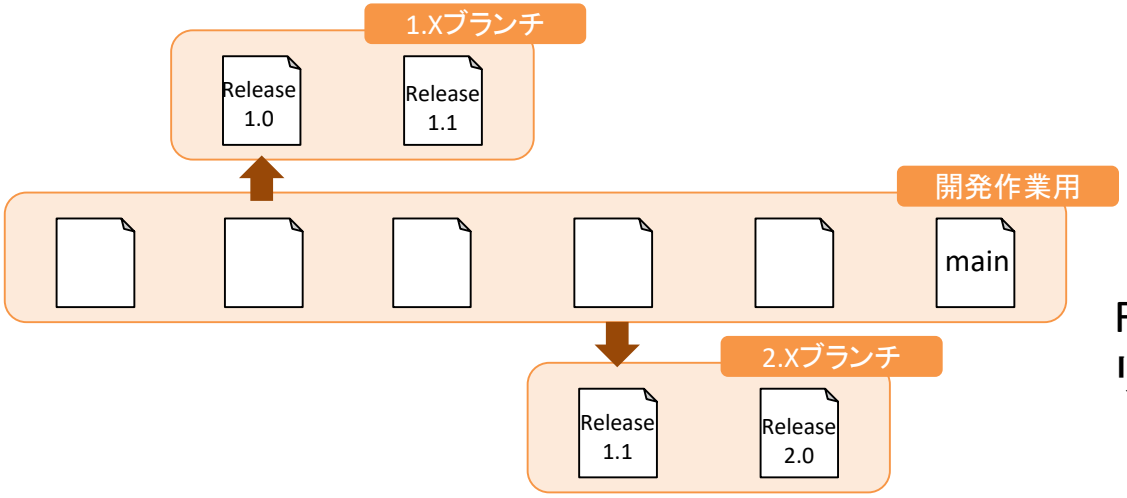
ブランチ

- 枝分かれしたコミット系列
- ブランチを作成することを「ブランチング」「ブランチを切る」などと表現する



ブランチの活用例(1)

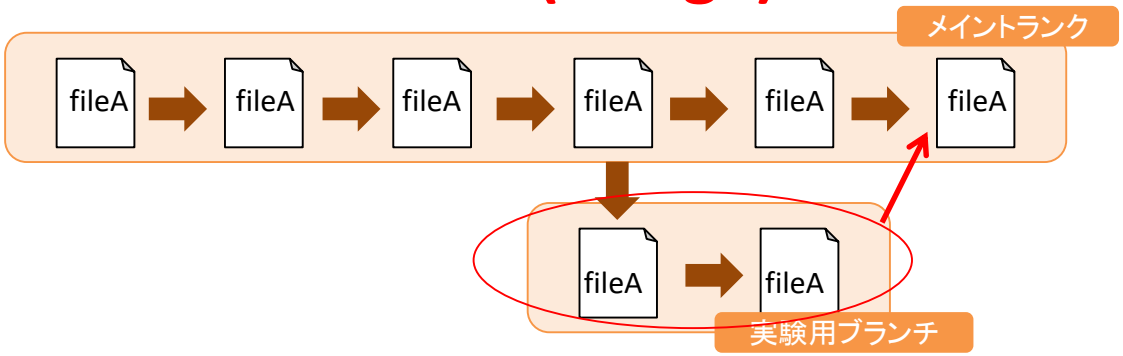
- リリースしたソフトウェアの管理
  - ✓ ソフトウェアに見つかった不具合修正は随時必要
  - ✓ 最新版の開発作業は並行してすすめたい
  - ✓ リリース後の修正版に，開発中の実験的実装が混入してはいけない
- ブランチによって，新機能を実装する履歴と，バグ修正を行う履歴を分ける



FreeBSDプロジェクトのリビジョンツリー概要

ブランチの活用例(2)

- 実験的開発用の履歴
  - ✓ 実験的に試してみたい実装がある
  - ✓ しかし，その方法でうまくいくかどうかはわからない
    - ✓ 最終的に使わない可能性もある
  - ✓ 大規模な変更のため，バージョン管理システムの支援は受けたい
    - ✓ 例: 複数人で開発し，開発期間が長い．．．
- もし実験的な機能を採用するなら，メインの方にそれらの変更を**マージ (merge)** する



ブランチを見てみよう！(git log --graph)

- リポジトリの準備

```
> git clone git://github.com/jquery/jquery.git
```

- git log --graph

```
> cd jquery/src
> git log --graph --pretty=format:"%cd %s" --date=short
data.js
```

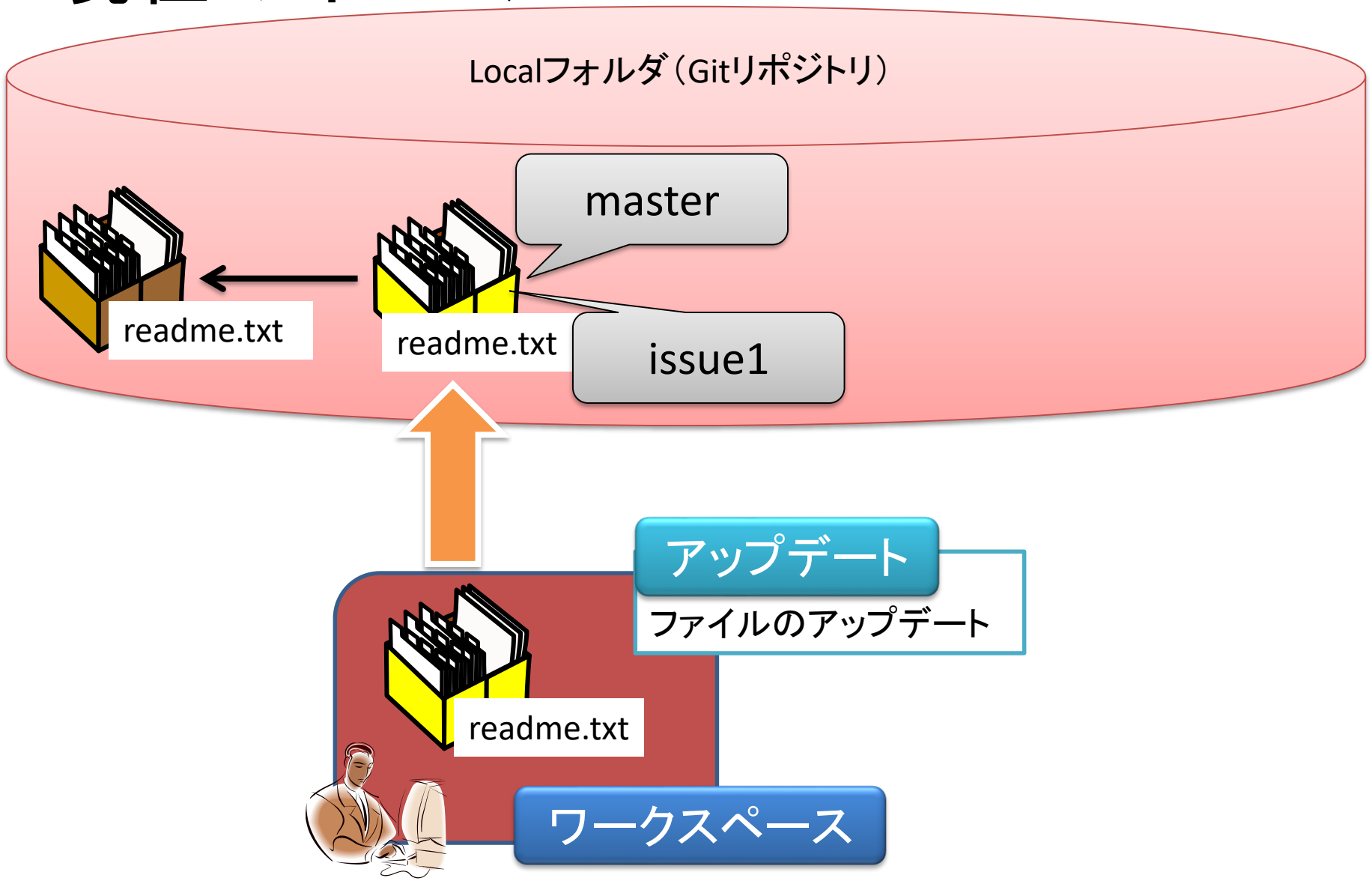
git log --graph とは？
コミットの推移をグラフ表示する.

ブランチの作り方 (git branch)

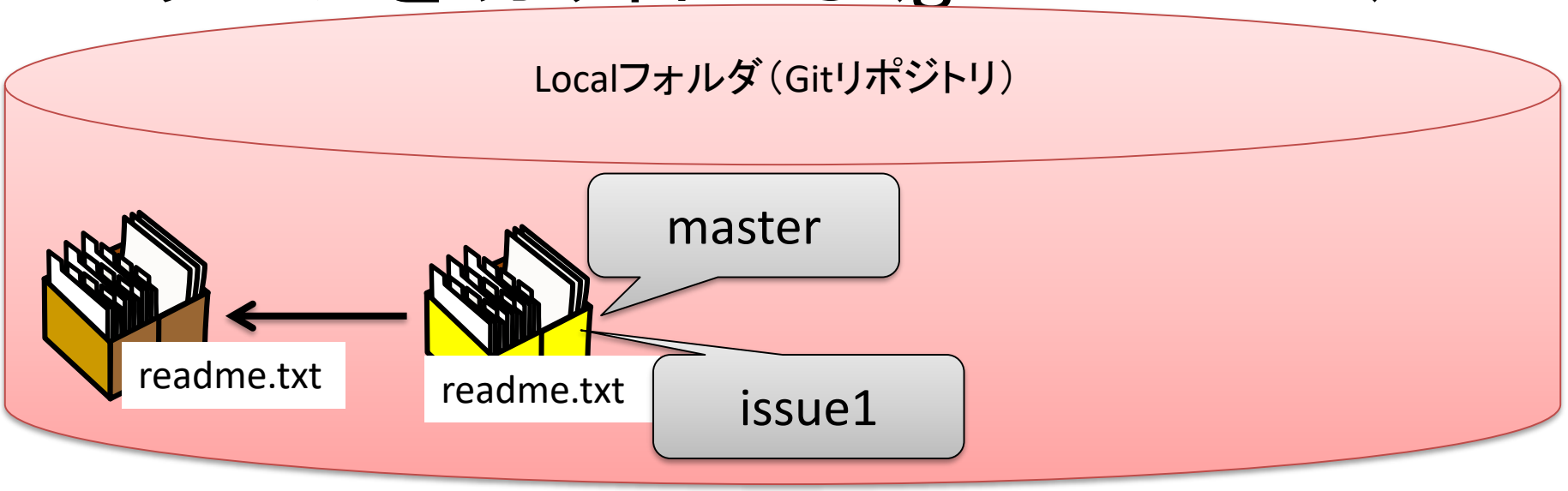
- git branch <branch name>

```
#一度localリポジトリに戻って
> git branch issue1
> git branch
    issue1
  *master
```

現在のイメージ


Localフォルダ（Gitリポジトリ）
master
readme.txt
readme.txt
issue1
アップデート
ファイルのアップデート
readme.txt
ワークスペース

ブランチを切り替える（git checkout）



このまま変更するとどちらのブランチとして変更される？ -> master

- git checkout -b <branch name>

```
> git checkout -b issue1
```

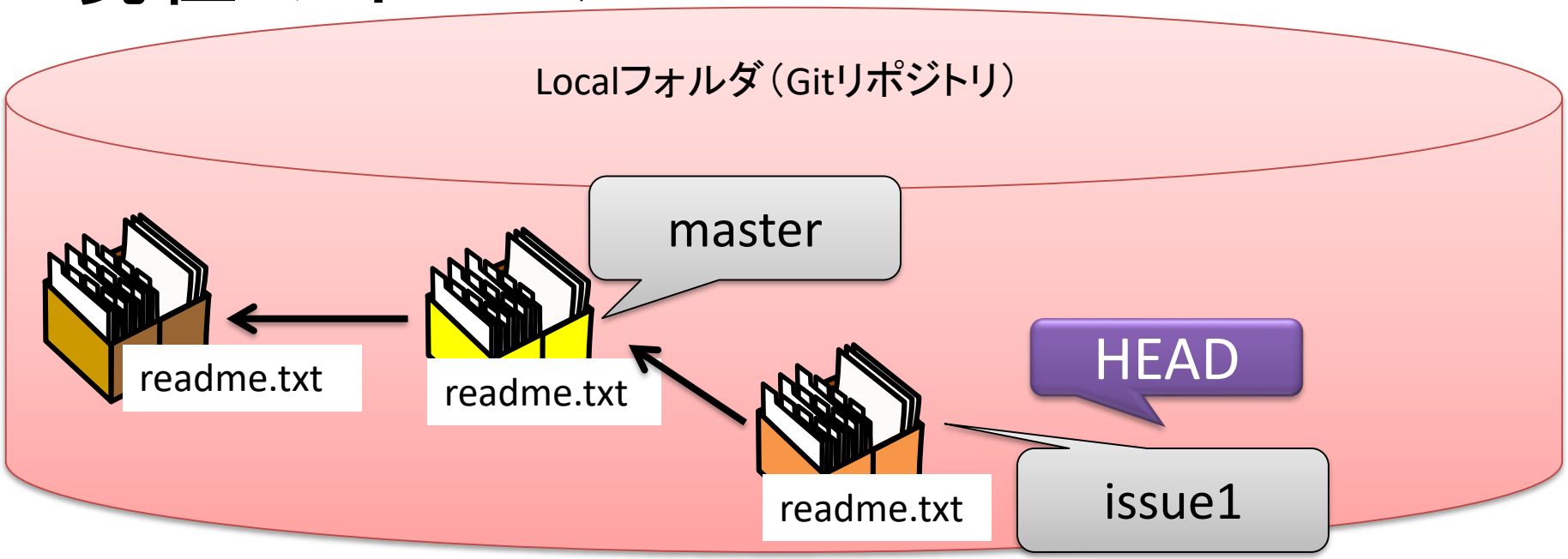
> > git checkout -b issue1
> “-b”のオプションは，ブランチの作成とチェックアウトを同時に行う.

```
> git branch   #現在のブランチを確認
```

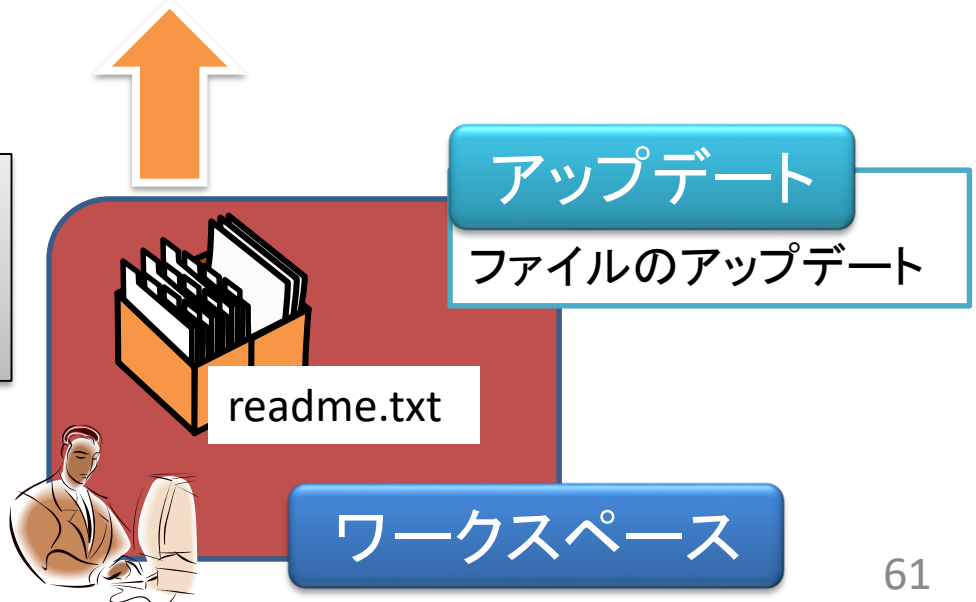
演習

1. issue1ブランチの，先ほど保存したファイルにgit logの説明文を追加し，コミットする．そのとき，どのように変更したのかメッセージを残す．
2. 再度“git log”コマンドでファイルがコミットされているか確認

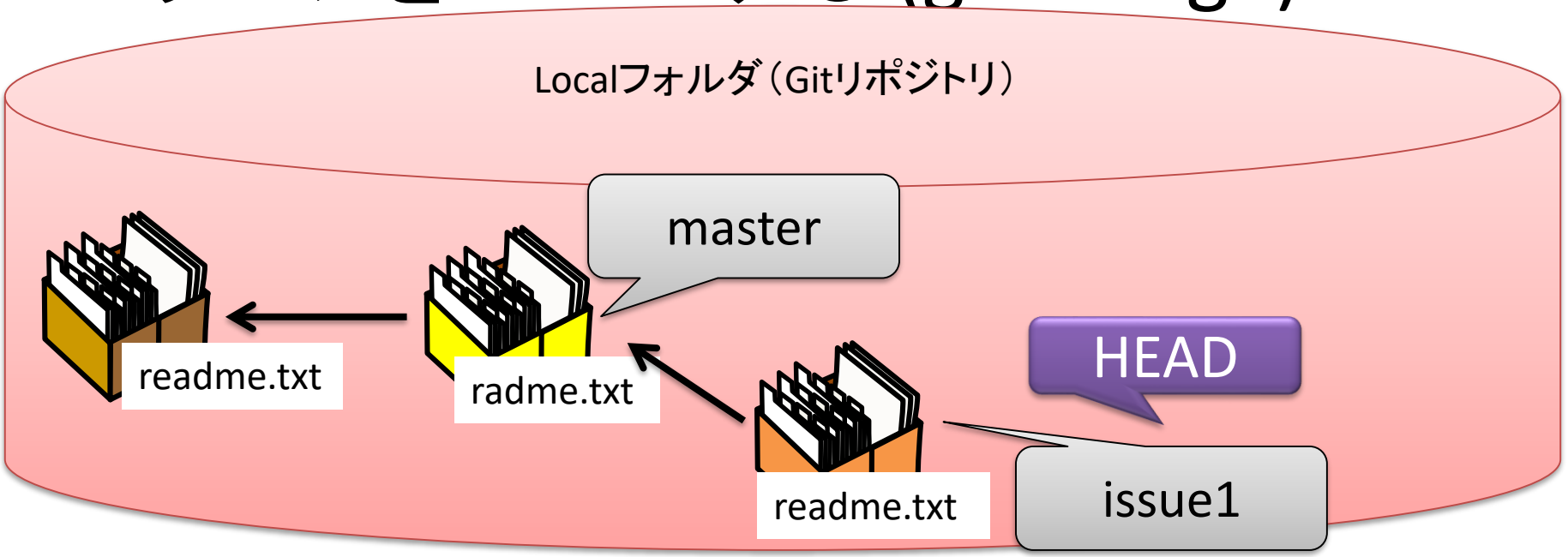
現在のイメージ



HEAD
現在さしているブランチのこと



ブランチをマージする (git merge)



- git merge <commit>

このままマージする”git merge master”と，HEADが指すissue1にマージされる

```
> git checkout master
#masterブランチのファイルの内容を確認
#git commitの説明までが残っているはず
```

演習

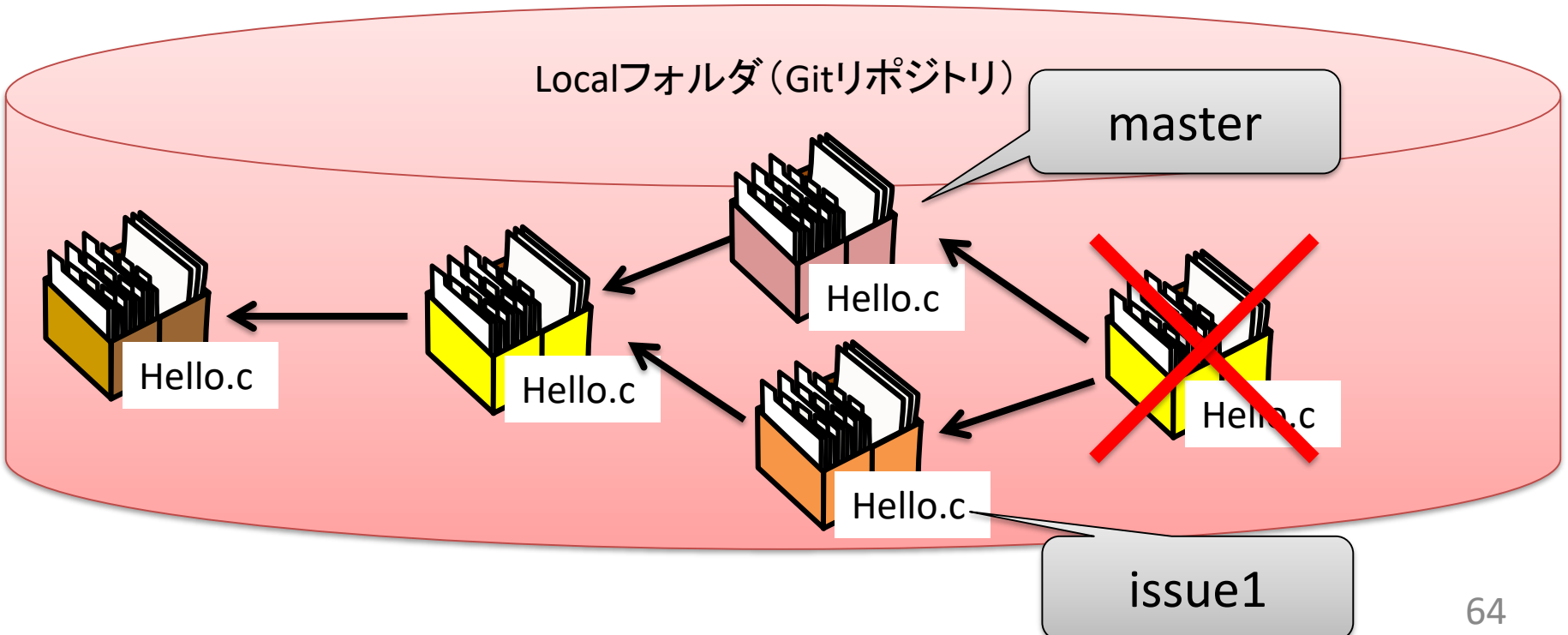
1. Masterブランチのファイルに，issue1ブランチのファイルをマージする.

2. masterブランチのファイル内容を確認

3. 再度“git log”コマンドでファイルがコミットされているか確認

コンフリクト

- 2つのブランチをマージするとき，変更箇所が同じの場合，コンフリクト（競合）が起こる．
- Gitでは変更箇所を自動的に統合する．
  - ✓ 自動で統合できない場合もある．



コンフリクトの例

```
#ブランチング前
・・・・・・・
if (i=0; i<10; i++)
・・・・・・・
```

```
#それぞれのブランチで変更後にマージ．コンフリクトした！
<<<<<<< HEAD
if (i=0; i<8; i++)
=======
if (i=0; i<5; i++)
>>>>>>> issue
```

演習課題

- 以下の作業を行い，作業・動作内容をレポート
  Report Titleは「mining1_学生番号」
  添付ファイルを利用する場合は，report titleと同じファイル名をつけること

1. ブランチを作成し，masterブランチと作成したブランチでファイルを編集・コミットし，作成したブランチをmasterブランチにマージせよ．ただし，コンフリクトさせて，解決すること．また，git log --graphでコミットの推移を確認せよ．（**[要提出]git log –graph結果画面のスクリーンショット**）
2. git logのオプションを3つ以上調べ，そのオプションの説明およびそれらを用いた場合どのように表示が変わるかを確認せよ．（**[要提出]git logのオプションの結果画面のスクリーンショット**）